made with doc-o-matic

RAD Studio VCL Reference

Copyright(C) 2008 CodeGear(TM). All Rights Reserved.

Table of Contents

ActnColorMaps Namespace 1

Classes **2**

TStandardColorMap Class 2

TStandardColorMap Members 2

TStandardColorMap Methods 6

TStandardColorMap Properties 6

TStandardColorMap Events 12

TTwilightColorMap Class 13

TTwilightColorMap Members 13

TTwilightColorMap Methods 17

TTwilightColorMap Properties 17

TTwilightColorMap Events 23

TXPColorMap Class 23

TXPColorMap Members 24

TXPColorMap Methods 27

TXPColorMap Properties 28

TXPColorMap Events 33

Constants **35**

ActnColorMaps.cXPBtnFrameColor Constant 35

ActnColorMaps.cXPFrameOuter Constant 35

ActnColorMaps.cXPSelectedColor Constant 35

ActnCtrls Namespace 37

Classes **38**

TActionToolBar Class 38

TActionToolBar Members 39

TActionToolBar Properties 52

TActionToolBar Events 82

TCustomActionCombo Class 118

TCustomActionCombo Members 118

TCustomActionCombo Methods 131

TCustomActionDockBar Class 133

TCustomActionDockBar Members 133

TCustomActionDockBar.Create Constructor 145

TCustomActionDockBar Methods 145
TCustomActionToolBar Class 148
TCustomActionToolBar Members 149
TCustomActionToolBar.Create Constructor 161
TCustomActionToolBar.Destroy Destructor 162
TCustomActionToolBar Methods 162
TCustomActionToolBar Properties 168
TCustomButtonControl Class 168
TCustomButtonControl Members 169
TCustomButtonControl.Create Constructor 177
TCustomButtonControl Fields 178
TCustomButtonControl Methods 178
TCustomButtonControl Properties 185
TCustomComboControl Class 191
TCustomComboControl Members 191
TCustomComboControl.Create Constructor 199
TCustomComboControl.Destroy Destructor 200
TCustomComboControl Methods 200
TCustomComboControl Properties 204
TCustomDropDownButton Class 205
TCustomDropDownButton Members 205
TCustomDropDownButton Methods 214
TCustomDropDownButton Properties 217
TCustomToolScrollBtn Class 217
TCustomToolScrollBtn Members 218
TCustomToolScrollBtn.Create Constructor 227
TCustomToolScrollBtn Methods 227
TCustomUtilityButton Class 228
TCustomUtilityButton Members 228
TCustomUtilityButton.Create Constructor 237
TCustomUtilityButton.Destroy Destructor 237
TCustomUtilityButton Methods 238
TCustomUtilityButton Properties 240
TCustomUtilityButton Events 242

Types **245**
ActnCtrls.TCustomToolScrollBtnClass Type 245

ActnList Namespace 247

Classes **249**
TAction Class 249
TAction Members 250
TAction.Create Constructor 254
TAction Properties 254
TAction Events 261
TActionLink Class 265
TActionLink Members 266
TActionLink Methods 267
TActionList Class 280
TActionList Members 281
TActionList Properties 284
TActionList Events 285
TActionListEnumerator Class 287
TActionListEnumerator Members 287
TActionListEnumerator.Create Constructor 287
TActionListEnumerator Methods 287
TActionListEnumerator Properties 288
TContainedAction Class 289
TContainedAction Members 289
TContainedAction.Destroy Destructor 292
TContainedAction Methods 293
TContainedAction Properties 295
TCustomAction Class 297
TCustomAction Members 298
TCustomAction.Create Constructor 301
TCustomAction.Destroy Destructor 302
TCustomAction Fields 302
TCustomAction Methods 303
TCustomAction Properties 307
TCustomAction Events 315
TCustomActionList Class 315
TCustomActionList Members 316
TCustomActionList.Create Constructor 319
TCustomActionList.Destroy Destructor 319
TCustomActionList Methods 319
TCustomActionList Properties 324
TCustomActionList Events 326
TShortCutList Class 328

TShortCutList Members 329
TShortCutList Methods 332
TShortCutList Properties 333

Functions 334
ActnList.CreateAction Function 334
ActnList.EnumRegisteredActions Function 334
ActnList.RegisterActions Function 335
ActnList.UnRegisterActions Function 336

Structs, Records, Enums 337
ActnList.TActionListState Enumeration 337

Types 338
ActnList.TActionEvent Type 338
ActnList.TActionLinkClass Type 338
ActnList.TContainedActionClass Type 339
ActnList.TEnumActionProc Type 339
ActnList.THintEvent Type 339

Constants 341
ActnList.CreateActionProc Constant 341
ActnList.EnumRegisteredActionsProc Constant 341
ActnList.RegisterActionsProc Constant 342
ActnList.UnRegisterActionsProc Constant 342

ActnMan Namespace 343

Classes 346
TActionBarItem Class 346
TActionBarItem Members 347
TActionBarItem.Create Constructor 349
TActionBarItem Methods 349
TActionBarItem Properties 350
TActionBars Class 352
TActionBars Members 353
TActionBars.Create Constructor 355
TActionBars Methods 355
TActionBars Properties 356
TActionBarStyle Class 358
TActionBarStyle Members 359
TActionBarStyle Methods 359

TActionBarStyleList Class 360
TActionBarStyleList Members 360
TActionBarStyleList Properties 363
TActionClient Class 364
TActionClient Members 364
TActionClient.Create Constructor 366
TActionClient.Destroy Destructor 366
TActionClient Methods 366
TActionClient Properties 369
TActionClientItem Class 374
TActionClientItem Members 375
TActionClientItem.Create Constructor 377
TActionClientItem.Destroy Destructor 377
TActionClientItem Methods 378
TActionClientItem Properties 382
TActionClientLink Class 389
TActionClientLink Members 389
TActionClientLink Fields 391
TActionClientLink Methods 391
TActionClients Class 400
TActionClients Members 401
TActionClients.Create Constructor 403
TActionClients Methods 403
TActionClients Properties 406
TActionClientsCollection Class 408
TActionClientsCollection Members 409
TActionClientsCollection Methods 411
TActionClientsCollection Properties 413
TActionDragBaseClass Class 415
TActionDragBaseClass Members 416
TActionDragBaseClass Properties 417
TActionDragObject Class 418
TActionDragObject Members 418
TActionDragObject.Create Constructor 419
TActionDragObject.Destroy Destructor 420
TActionDragObject Fields 420
TActionDragObject Methods 421
TActionDragObject Properties 421
TActionItemDragObject Class 422

TActionItemDragObject Members 422
TActionItemDragObject Methods 424
TActionItemDragObject Properties 424
TActionListCollection Class 425
TActionListCollection Members 425
TActionListCollection Properties 427
TActionListItem Class 428
TActionListItem Members 429
TActionListItem Methods 430
TActionListItem Properties 431
TActionManager Class 432
TActionManager Members 433
TActionManager Properties 437
TActionManager Events 441
TCategoryDragObject Class 444
TCategoryDragObject Members 445
TCategoryDragObject.Create Constructor 446
TCategoryDragObject Properties 446
TCustomActionBar Class 447
TCustomActionBar Members 447
TCustomActionBar.Create Constructor 459
TCustomActionBar.Destroy Destructor 459
TCustomActionBar Fields 460
TCustomActionBar Methods 460
TCustomActionBar Properties 478
TCustomActionBar Events 491
TCustomActionBarColorMap Class 492
TCustomActionBarColorMap Members 493
TCustomActionBarColorMap.Create Constructor 496
TCustomActionBarColorMap.Destroy Destructor 496
TCustomActionBarColorMap Methods 497
TCustomActionBarColorMap Properties 499
TCustomActionBarColorMap Events 506
TCustomActionClass Class 506
TCustomActionControl Class 507
TCustomActionControl Members 507
TCustomActionControl.Create Constructor 515
TCustomActionControl.Destroy Destructor 516
TCustomActionControl Methods 516

TCustomActionControl Properties 528
TCustomActionControl Events 535
TCustomActionManager Class 537
TCustomActionManager Members 537
TCustomActionManager.Create Constructor 541
TCustomActionManager.Destroy Destructor 542
TCustomActionManager Methods 542
TCustomActionManager Properties 553
TCustomActionManager Events 557
TXToolDockForm Class 559

Interfaces 560
IActionBarDesigner Interface 560
IActionBarDesigner Members 560
IActionBarDesigner Methods 560

Functions 563
ActnMan.NotifyDesigner Function 563

Structs, Records, Enums 564
ActnMan.TBackgroundLayout Enumeration 564
ActnMan.TBarEdge Enumeration 564
ActnMan.TBarOrientation Enumeration 565
ActnMan.TCaptionOptions Enumeration 565
ActnMan.TChangesAllowed Enumeration 566
ActnMan.TCMItemMsg Record 566

Types 568
ActnMan.TActionBarItemClass Type 568
ActnMan.TActionBarsClass Type 568
ActnMan.TActionClientClass Type 569
ActnMan.TActionClientItemClass Type 569
ActnMan.TActionClientLinkClass Type 569
ActnMan.TActionClientsClass Type 570
ActnMan.TActionProc Type 570
ActnMan.TChangesAllowedSet Type 570
ActnMan.TColorMapUpdateProc Type 571
ActnMan.TControlCreatedEvent Type 571
ActnMan.TCustomActionBarClass Type 571
ActnMan.TCustomActionControlClass Type 572
ActnMan.TCustomColorMapClass Type 572

ActnMan.TGetBarItemClassEvent Type 572
ActnMan.TGetBarsClassEvent Type 573
ActnMan.TGetClientItemClassEvent Type 573
ActnMan.TGetClientsClassEvent Type 573
ActnMan.TGetControlClassEvent Type 574
ActnMan.TMergeRange Type 574
ActnMan.TStyleChanged Type 574

Variables **575**
ActnMan.ActionBarDesigner Variable 575
ActnMan.ActionBarStyles Variable 575
ActnMan.DefaultActnBarStyle Variable 575

Constants **577**
ActnMan.caAllChanges Constant 577
ActnMan.cDefaultSchedule Constant 577
ActnMan.CM_ENTERMENULOOP Constant 578
ActnMan.CM_ITEMCLICKED Constant 578
ActnMan.CM_ITEMDROPPOINT Constant 578
ActnMan.CM_ITEMKEYED Constant 579
ActnMan.CM_ITEMSELECTED Constant 579
ActnMan.CM_RESETBAR Constant 579
ActnMan.CM_SCROLLCLICKED Constant 580
ActnMan.ControlDragMode Constant 580

ActnMenus Namespace 581

Classes **583**
TActionBarStyleEx Class 583
TActionBarStyleEx Members 584
TActionBarStyleEx Methods 584
TActionMainMenuBar Class 585
TActionMainMenuBar Members 586
TActionMainMenuBar Properties 600
TActionMainMenuBar Events 624
TCustomActionMainMenuBar Class 661
TCustomActionMainMenuBar Members 662
TCustomActionMainMenuBar.Create Constructor 675
TCustomActionMainMenuBar.Destroy Destructor 676
TCustomActionMainMenuBar Fields 676
TCustomActionMainMenuBar Methods 676

TCustomActionMainMenuBar Properties 683
TCustomActionMenuBar Class 685
TCustomActionMenuBar Members 685
TCustomActionMenuBar.Create Constructor 698
TCustomActionMenuBar.Destroy Destructor 698
TCustomActionMenuBar Fields 699
TCustomActionMenuBar Methods 700
TCustomActionMenuBar Properties 716
TCustomActionMenuBar Events 720
TCustomActionPopupMenu Class 721
TCustomActionPopupMenu Members 722
TCustomActionPopupMenu.Create Constructor 735
TCustomActionPopupMenu.Destroy Destructor 736
TCustomActionPopupMenu Methods 736
TCustomActionPopupMenu Properties 744
TCustomAddRemoveItem Class 745
TCustomAddRemoveItem Members 746
TCustomAddRemoveItem Methods 754
TCustomizeActionClientItem Class 756
TCustomizeActionClientItem Members 756
TCustomizeActionClientItem Methods 759
TCustomizeActionLink Class 759
TCustomizeActionLink Members 760
TCustomizeActionLink Methods 762
TCustomizeActionToolBar Class 762
TCustomizeActionToolBar Members 763
TCustomizeActionToolBar.Create Constructor 777
TCustomizeActionToolBar.Destroy Destructor 777
TCustomizeActionToolBar Methods 777
TCustomizeActionToolBar Properties 780
TCustomMDIMenuButton Class 781
TCustomMDIMenuButton Members 781
TCustomMDIMenuButton Methods 789
TCustomMDIMenuButton Properties 790
TCustomMenuButton Class 790
TCustomMenuButton Members 791
TCustomMenuButton.Create Constructor 799
TCustomMenuButton.Destroy Destructor 800
TCustomMenuButton Methods 800

TCustomMenuButton Properties 804
TCustomMenuExpandBtn Class 805
TCustomMenuExpandBtn Members 805
TCustomMenuExpandBtn.Create Constructor 814
TCustomMenuExpandBtn Methods 815
TCustomMenuExpandBtn Properties 816
TCustomMenuItem Class 816
TCustomMenuItem Members 817
TCustomMenuItem.Create Constructor 825
TCustomMenuItem.Destroy Destructor 825
TCustomMenuItem Fields 826
TCustomMenuItem Methods 826
TCustomMenuItem Properties 831
TMenuStack Class 833
TMenuStack Members 833
TMenuStack.Create Constructor 834
TMenuStack Methods 834
TMenuStack Properties 835

Functions 837
ActnMenus.RegisterActnBarStyle Function 837
ActnMenus.UnRegisterActnBarStyle Function 837

Structs, Records, Enums 839
ActnMenus.TAnimationStyle Enumeration 839
ActnMenus.TMDIButtonStyle Enumeration 839

Types 841
ActnMenus.TAddRemoveItemClass Type 841
ActnMenus.TCustomAddRemoveItemClass Type 841
ActnMenus.TCustomizeActionToolBarClass Type 842
ActnMenus.TCustomMenuExpandBtnClass Type 842
ActnMenus.TCustomPopupClass Type 842
ActnMenus.TExitMenuEvent Type 843
ActnMenus.TGetPopupClassEvent Type 843
ActnMenus.TMenuButtonControlClass Type 843
ActnMenus.TMenuEdges Type 844
ActnMenus.TMenuItemControlClass Type 844
ActnMenus.TMenuPopupEvent Type 844
ActnMenus.TUpdateActnMenusProc Type 845

Variables **846**
ActnMenus.MenuAddRemoveItemClass Variable 846
ActnMenus.MenuButtonControlClass Variable 846
ActnMenus.MenuCustomizePopupClass Variable 847
ActnMenus.MenuItemControlClass Variable 847
ActnMenus.MenuPopupClass Variable 847
ActnMenus.UpdateActnMenusProc Variable 848

ActnPopup Namespace 849

Classes **850**
TCustomActionPopupMenuEx Class 850
TCustomActionPopupMenuEx Members 850
TCustomActionPopupMenuEx Methods 850
TPopupActionBar Class 852
TPopupActionBar Members 852
TPopupActionBar.Create Constructor 856
TPopupActionBar Methods 856
TPopupActionBar Properties 857
TPopupActionBar Events 858

Functions **859**
ActnPopup.NewPopupMenu Function 859

AdaptReq Namespace 861

Classes **863**
EAdapterRequestException Class 863
TActionFieldValueImpl Class 864
TActionFieldValueImpl Members 864
TActionFieldValueImpl.Create Constructor 865
TActionFieldValueImpl.Destroy Destructor 866
TActionFieldValueImpl Methods 866
TActionFieldValuesImpl Class 868
TActionFieldValuesImpl Members 868
TActionFieldValuesImpl.Create Constructor 869
TActionFieldValuesImpl.Destroy Destructor 870
TActionFieldValuesImpl Methods 870
TActionFieldValuesImpl Properties 872
TAdapterActionFieldValuesList Class 873

TAdapterActionFieldValuesList Members 874
TAdapterActionFieldValuesList.Create Constructor 875
TAdapterActionFieldValuesList.Destroy Destructor 875
TAdapterActionFieldValuesList Methods 875
TAdapterActionFieldValuesList Properties 876
TAdapterModes Class 876
TAdapterModes Members 877
TAdapterModes.Create Constructor 878
TAdapterModes.Destroy Destructor 878
TAdapterModes Methods 878
TAdapterModes Properties 879
TAdapterRequestParamsImpl Class 880
TAdapterRequestParamsImpl Members 880
TAdapterRequestParamsImpl.Create Constructor 881
TAdapterRequestParamsImpl Methods 881
TAdapterRequestParamsImpl Properties 883
TBasicActionRequestImpl Class 884
TBasicActionRequestImpl Members 884
TBasicActionRequestImpl.Create Constructor 886
TBasicActionRequestImpl.Destroy Destructor 886
TBasicActionRequestImpl Methods 887
TBasicActionRequestImpl Properties 888
TBasicActionResponseImpl Class 890
TBasicActionResponseImpl Members 890
TBasicActionResponseImpl.Create Constructor 892
TBasicActionResponseImpl Methods 892
TBasicActionResponseImpl Properties 896
TBasicImageRequestImpl Class 898
TBasicImageRequestImpl Members 898
TBasicImageRequestImpl.Create Constructor 900
TBasicImageRequestImpl.Destroy Destructor 900
TBasicImageRequestImpl Methods 900
TBasicImageRequestImpl Properties 901
TBasicImageResponseImpl Class 902
TBasicImageResponseImpl Members 902
TBasicImageResponseImpl.Create Constructor 903
TEnumActionFieldValueRecordsImpl Class 904
TEnumActionFieldValueRecordsImpl Members 904
TEnumActionFieldValueRecordsImpl.Create Constructor 905

TEnumActionFieldValueRecordsImpl.Destroy Destructor 905
TEnumActionFieldValueRecordsImpl Methods 906
TEnumActionFieldValueRecordsImpl Properties 908
TMultiRecordActionRequestImpl Class 909
TMultiRecordActionRequestImpl Members 910
TMultiRecordActionRequestImpl.Create Constructor 912
TMultiRecordActionRequestImpl.Destroy Destructor 912
TMultiRecordActionRequestImpl Properties 913
TSingleRecordActionRequestImpl Class 915
TSingleRecordActionRequestImpl Members 915
TSingleRecordActionRequestImpl.Create Constructor 917
TSingleRecordActionRequestImpl.Destroy Destructor 918
TSingleRecordActionRequestImpl Properties 918

Interfaces 920

IActionFieldValue Interface 920
IActionFieldValue Members 921
IActionFieldValue Methods 921
IActionFieldValue Properties 923
IActionFieldValues Interface 925
IActionFieldValues Members 926
IActionFieldValues Methods 927
IActionFieldValues Properties 930
IActionFieldValuesOfAdapter Interface 933
IActionFieldValuesOfAdapter Members 933
IActionFieldValuesOfAdapter Methods 933
IActionHandleResponse Interface 934
IActionHandleResponse Members 934
IActionHandleResponse Methods 935
IActionRequest Interface 935
IActionRequest Members 936
IActionRequest Methods 936
IActionRequest Properties 937
IActionRespondWith Interface 938
IActionRespondWith Members 939
IActionRespondWith Methods 939
IActionRespondWith Properties 941
IActionResponse Interface 941
IActionResponse Members 942
IActionResponse Methods 942

IActionResponse Properties 944
IAdapterModes Interface 944
IAdapterModes Members 945
IAdapterModes Methods 946
IAdapterModes Properties 947
IAdapterRecordKeys Interface 949
IAdapterRecordKeys Members 950
IAdapterRecordKeys Methods 950
IAdapterRecordKeys Properties 951
IAdapterRequestParams Interface 952
IAdapterRequestParams Members 953
IAdapterRequestParams Methods 953
IAdapterRequestParams Properties 956
IEnumActionFieldValueRecords Interface 957
IEnumActionFieldValueRecords Members 958
IEnumActionFieldValueRecords Methods 959
IEnumActionFieldValueRecords Properties 962
IGetAdapterActionRedirectOptions Interface 963
IGetAdapterActionRedirectOptions Members 964
IGetAdapterActionRedirectOptions Methods 964
IGetAdapterActionRedirectOptions Properties 965
IImageRequest Interface 965
IImageRequest Members 966
IImageRequest Methods 966
IImageRequest Properties 967
IImageResponse Interface 967

Functions **968**
AdaptReq.ExtractStringValue Function 968
AdaptReq.TestRedirectOptions Function 968

Structs, Records, Enums **969**
AdaptReq.TAdapterActionRedirectOption Enumeration 969
AdaptReq.TExecuteStatus Enumeration 969
AdaptReq.TRespondWith Enumeration 970

Types **971**
AdaptReq.TAdapterActionRedirectOptions Type 971

Constants **972**
AdaptReq.roRedirect Constant 972

AdaptReq.sActionRequest Constant 972
AdaptReq.sDelimAdapterMode Constant 973
AdaptReq.sDelimOrigFieldValue Constant 973
AdaptReq.sDelimRecordKeys Constant 973

ADODB Namespace 975

Classes 978
EADOError Class 978
TADOBlobStream Class 979
TADOBlobStream Members 979
TADOBlobStream.Create Constructor 981
TADOBlobStream.Destroy Destructor 981
TADOBlobStream Methods 982
TADOCommand Class 984
TADOCommand Members 985
TADOCommand.Create Constructor 988
TADOCommand.Destroy Destructor 989
TADOCommand Methods 989
TADOCommand Properties 994
TADOConnection Class 1001
TADOConnection Members 1002
TADOConnection.Create Constructor 1006
TADOConnection.Destroy Destructor 1006
TADOConnection Methods 1007
TADOConnection Properties 1020
TADOConnection Events 1033
TADODataSet Class 1041
TADODataSet Members 1042
TADODataSet Methods 1054
TADODataSet Properties 1056
TADODataSet Events 1067
TADOQuery Class 1084
TADOQuery Members 1085
TADOQuery.Create Constructor 1099
TADOQuery.Destroy Destructor 1099
TADOQuery Methods 1099
TADOQuery Properties 1100
TADOStoredProc Class 1106

TADOStoredProc Members 1107
TADOStoredProc.Create Constructor 1120
TADOStoredProc Methods 1121
TADOStoredProc Properties 1122
TADOTable Class 1125
TADOTable Members 1125
TADOTable.Create Constructor 1139
TADOTable Methods 1139
TADOTable Properties 1140
TCustomADODataSet Class 1145
TCustomADODataSet Members 1146
TCustomADODataSet.Create Constructor 1156
TCustomADODataSet.Destroy Destructor 1157
TCustomADODataSet Methods 1157
TCustomADODataSet Properties 1200
TCustomADODataSet Events 1223
TParameter Class 1250
TParameter Members 1251
TParameter Methods 1252
TParameter Properties 1256
TParameters Class 1261
TParameters Members 1261
TParameters Methods 1263
TParameters Properties 1271
TRDSConnection Class 1274
TRDSConnection Members 1275
TRDSConnection.Create Constructor 1278
TRDSConnection.Destroy Destructor 1278
TRDSConnection Methods 1279
TRDSConnection Properties 1281
TRDSConnection Events 1283

Functions 1286

ADODB.CreateUDLFile Function 1286
ADODB.DataLinkDir Function 1286
ADODB.GetDataLinkFiles Function 1287
ADODB.GetDataLinkFiles Function 1287
ADODB.GetProviderNames Function 1287
ADODB.GetProviderNames Function 1288
ADODB.PromptDataLinkFile Function 1288

ADODB.PromptDataSource Function 1288

Structs, Records, Enums 1289

ADODB.TADOLockType Enumeration 1289
ADODB.TAffectRecords Enumeration 1290
ADODB.TCommandType Enumeration 1291
ADODB.TConnectMode Enumeration 1292
ADODB.TConnectOption Enumeration 1293
ADODB.TCursorLocation Enumeration 1293
ADODB.TCursorOption Enumeration 1294
ADODB.TCursorType Enumeration 1295
ADODB.TEventReason Enumeration 1296
ADODB.TEventStatus Enumeration 1298
ADODB.TExecuteOption Enumeration 1299
ADODB.TFilterGroup Enumeration 1299
ADODB.TIsolationLevel Enumeration 1300
ADODB.TMarshalOption Enumeration 1301
ADODB.TObjectState Enumeration 1302
ADODB.TParameterAttribute Enumeration 1302
ADODB.TParameterDirection Enumeration 1303
ADODB.TPersistFormat Enumeration 1303
ADODB.TRecordStatus Enumeration 1304
ADODB.TSchemaInfo Enumeration 1305
ADODB.TSeekOption Enumeration 1308
ADODB.TXactAttribute Enumeration 1308

Types 1310

ADODB._Command Type 1310
ADODB._Connection Type 1311
ADODB._Parameter Type 1311
ADODB._Recordset Type 1311
ADODB.Error Type 1312
ADODB.Errors Type 1312
ADODB.Parameters Type 1312
ADODB.Properties Type 1313
ADODB.Property_ Type 1313
ADODB.PVariantList Type 1313
ADODB.TBeginTransCompleteEvent Type 1314
ADODB.TConnectErrorEvent Type 1314
ADODB.TCursorOptions Type 1314

ADODB.TDataType Type 1315
ADODB.TDisconnectEvent Type 1315
ADODB.TEndOfRecordsetEvent Type 1315
ADODB.TExecuteCompleteEvent Type 1316
ADODB.TExecuteOptions Type 1316
ADODB.TFetchProgressEvent Type 1317
ADODB.TFieldChangeCompleteEvent Type 1317
ADODB.TInfoMessageEvent Type 1317
ADODB.TObjectStates Type 1318
ADODB.TParameterAttributes Type 1318
ADODB.TPropList Type 1319
ADODB.TRecordChangeCompleteEvent Type 1319
ADODB.TRecordsetCreate Type 1320
ADODB.TRecordsetErrorEvent Type 1320
ADODB.TRecordsetEvent Type 1320
ADODB.TRecordsetReasonEvent Type 1321
ADODB.TRecordStatusSet Type 1321
ADODB.TVariantList Type 1322
ADODB.TWillChangeFieldEvent Type 1322
ADODB.TWillChangeRecordEvent Type 1323
ADODB.TWillConnectEvent Type 1323
ADODB.TWillExecuteEvent Type 1323
ADODB.TXactAttributes Type 1324

AppEvnts Namespace 1325

Classes 1326

TApplicationEvents Class 1326
TApplicationEvents Members 1326
TApplicationEvents Events 1329
TCustomApplicationEvents Class 1347
TCustomApplicationEvents Members 1347
TCustomApplicationEvents.Create Constructor 1350
TCustomApplicationEvents Methods 1351
TCustomApplicationEvents Events 1352

AutoAdap Namespace 1371

Classes 1373

TActionLink Class 1373

TActionLink Members 1374
TActionLink.Create Constructor 1375
TActionLink Methods 1375
TAdapterActionArrayStub Class 1376
TAdapterActionArrayStub Members 1377
TAdapterActionArrayStub.Create Constructor 1378
TAdapterActionArrayStub Properties 1378
TAdapterActionStub Class 1379
TAdapterActionStub Members 1379
TAdapterActionStub Methods 1380
TAdapterComponentsStub Class 1383
TAdapterComponentsStub Members 1384
TAdapterComponentsStub Methods 1384
TAdapterErrorsStub Class 1384
TAdapterErrorsStub Members 1385
TAdapterErrorsStub.Create Constructor 1386
TAdapterErrorsStub Methods 1386
TAdapterErrorsStub Properties 1387
TAdapterErrorStub Class 1387
TAdapterErrorStub Members 1388
TAdapterErrorStub.Create Constructor 1389
TAdapterErrorStub Methods 1389
TAdapterFieldStub Class 1390
TAdapterFieldStub Members 1391
TAdapterFieldStub Methods 1393
TAdapterFieldValuesStub Class 1398
TAdapterFieldValuesStub Members 1398
TAdapterFieldValuesStub Methods 1399
TAdapterHiddenFieldsStub Class 1400
TAdapterHiddenFieldsStub Members 1400
TAdapterHiddenFieldsStub.Create Constructor 1402
TAdapterHiddenFieldsStub Methods 1402
TAdapterHiddenFieldsStub Properties 1402
TAdapterHiddenFieldStub Class 1403
TAdapterHiddenFieldStub Members 1403
TAdapterHiddenFieldStub.Create Constructor 1404
TAdapterHiddenFieldStub Methods 1404
TAdapterImageStub Class 1405
TAdapterImageStub Members 1405

TAdapterImageStub.Create Constructor 1407
TAdapterImageStub Methods 1407
TAdapterRecordsStub Class 1407
TAdapterRecordsStub Members 1408
TAdapterRecordsStub.Create Constructor 1409
TAdapterRecordsStub Properties 1409
TAdapterRecordStub Class 1409
TAdapterRecordStub Members 1410
TAdapterRecordStub.Create Constructor 1411
TAdapterRecordStub Properties 1411
TAdapterStub Class 1412
TAdapterStub Members 1412
TAdapterStub Methods 1413
TAdapterSubComponentStub Class 1416
TAdapterSubComponentStub Members 1416
Create Constructor 1417
TAdapterSubComponentStub Methods 1418
TAdapterSubComponentStub Properties 1419
TAdapterValuesListStub Class 1419
TAdapterValuesListStub Members 1419
TAdapterValuesListStub Methods 1420
TBaseAdapterStub Class 1422
TBaseAdapterStub Members 1423
TBaseAdapterStub Fields 1423
TBaseAdapterStub Methods 1424
TBaseRecordsAdapterStub Class 1425
TBaseRecordsAdapterStub Members 1426
TBaseRecordsAdapterStub Methods 1426
TWebApplicationStub Class 1427
TWebApplicationStub Members 1427
TWebApplicationStub.Create Constructor 1429
TWebApplicationStub Methods 1429
TWebEndUserStub Class 1430
TWebEndUserStub Members 1431
TWebEndUserStub.Create Constructor 1432
TWebEndUserStub Methods 1432

Interfaces **1434**
IAdapterErrorsStub Interface 1434
IAdapterErrorsStub Members 1434

IAdapterErrorsStub Methods 1435
IAdapterErrorStub Interface 1435
IAdapterErrorStub Members 1436
IAdapterErrorStub Methods 1436
IAdapterErrorStub Properties 1437
IAdapterFieldStub Interface 1438
IAdapterFieldStub Members 1438
IAdapterFieldStub Methods 1439
IAdapterFieldStub Properties 1444
IAdapterFieldValuesStub Interface 1448
IAdapterFieldValuesStub Members 1448
IAdapterFieldValuesStub Methods 1449
IAdapterFieldValuesStub Properties 1450
IAdapterHiddenFieldsStub Interface 1450
IAdapterHiddenFieldsStub Members 1450
IAdapterHiddenFieldsStub Methods 1450
IAdapterHiddenFieldStub Interface 1451
IAdapterHiddenFieldStub Members 1451
IAdapterHiddenFieldStub Methods 1452
IAdapterHiddenFieldStub Properties 1452
IAdapterImageStub Interface 1453
IAdapterImageStub Members 1453
IAdapterImageStub Methods 1453
IAdapterImageStub Properties 1454
IAdapterRecordStub Interface 1454
IAdapterStub Interface 1455
IAdapterStub Members 1455
IAdapterStub Methods 1456
IAdapterStub Properties 1457
IAdapterValuesListStub Interface 1459
IAdapterValuesListStub Members 1459
IAdapterValuesListStub Methods 1460
IAdapterValuesListStub Properties 1461
IApplicationStub Interface 1462
IApplicationStub Members 1462
IApplicationStub Methods 1463
IApplicationStub Properties 1465
IBaseAdapterStub Interface 1466
IBaseAdapterStub Members 1466

IBaseAdapterStub Methods 1466
IBaseAdapterStub Properties 1467
IBaseRecordsAdapterStub Interface 1468
IBaseRecordsAdapterStub Members 1468
IBaseRecordsAdapterStub Methods 1469
IBaseRecordsAdapterStub Properties 1469
IEndUserStub Interface 1470
IEndUserStub Members 1470
IEndUserStub Methods 1471
IEndUserStub Properties 1472
IImageProducerStub Interface 1473
IImageProducerStub Members 1473
IImageProducerStub Methods 1473

Types 1475
AutoAdap.TAdapterStubClass Type 1475

AutoDisp Namespace 1477

Classes 1478
TWebDispatch Class 1478
TWebDispatch Members 1478
TWebDispatch.Create Constructor 1480
TWebDispatch.Destroy Destructor 1480
TWebDispatch Methods 1480
TWebDispatch Properties 1481
TWebPathInfo Class 1483
TWebPathInfo Members 1484
TWebPathInfo.Create Constructor 1484
TWebPathInfo.Destroy Destructor 1484
TWebPathInfo Properties 1484

AxCtrls Namespace 1487

Classes 1489
TActiveForm Class 1489
TActiveForm Members 1490
TActiveForm Methods 1505
TActiveForm Properties 1506
TActiveFormControl Class 1507

TActiveFormControl Members 1508
TActiveFormControl Methods 1512
TActiveFormFactory Class 1514
TActiveFormFactory Members 1514
TActiveFormFactory Methods 1516
TActiveXControl Class 1517
TActiveXControl Members 1518
TActiveXControl.Destroy Destructor 1522
TActiveXControl Methods 1522
TActiveXControl Properties 1552
TActiveXControlFactory Class 1554
TActiveXControlFactory Members 1555
TActiveXControlFactory.Create Constructor 1556
TActiveXControlFactory.Destroy Destructor 1557
TActiveXControlFactory Methods 1557
TActiveXControlFactory Properties 1559
TActiveXPropertyPage Class 1561
TActiveXPropertyPage Members 1561
TActiveXPropertyPage.Destroy Destructor 1563
TActiveXPropertyPage Methods 1563
TActiveXPropertyPage Properties 1563
TActiveXPropertyPageFactory Class 1564
TActiveXPropertyPageFactory Members 1565
TActiveXPropertyPageFactory.Create Constructor 1566
TActiveXPropertyPageFactory Methods 1566
TAdapterNotifier Class 1567
TAdapterNotifier Members 1568
TAdapterNotifier.Create Constructor 1569
TAdapterNotifier Methods 1569
TConnectionPoint Class 1570
TConnectionPoint Members 1571
TConnectionPoint.Create Constructor 1572
TConnectionPoint.Destroy Destructor 1572
TConnectionPoint Methods 1573
TConnectionPoint Properties 1574
TConnectionPoints Class 1575
TConnectionPoints Members 1575
TConnectionPoints.Create Constructor 1576
TConnectionPoints.Destroy Destructor 1576

TConnectionPoints Methods 1576
TConnectionPoints Properties 1578
TCustomAdapter Class 1578
TCustomAdapter Members 1579
TCustomAdapter.Create Constructor 1580
TCustomAdapter.Destroy Destructor 1581
TCustomAdapter Fields 1581
TCustomAdapter Methods 1581
TFontAdapter Class 1583
TFontAdapter Members 1584
TFontAdapter.Create Constructor 1585
TFontAdapter Methods 1585
TOleGraphic Class 1587
TOleGraphic Members 1587
TOleGraphic Methods 1589
TOleGraphic Properties 1594
TOleStream Class 1595
TOleStream Members 1596
TOleStream.Create Constructor 1597
TOleStream Methods 1597
TPictureAdapter Class 1599
TPictureAdapter Members 1600
TPictureAdapter.Create Constructor 1601
TPictureAdapter Methods 1602
TPropertyPage Class 1603
TPropertyPage Members 1603
TPropertyPage.Create Constructor 1616
TPropertyPage.Destroy Destructor 1617
TPropertyPage Methods 1617
TPropertyPage Properties 1619
TPropertyPage Events 1646
TPropertyPageImpl Class 1679
TPropertyPageImpl Members 1679
TPropertyPageImpl Methods 1681
TPropertyPageImpl Properties 1684
TReflectorWindow Class 1684
TReflectorWindow Members 1685
TReflectorWindow.Create Constructor 1695
TReflectorWindow Methods 1695

TStringsAdapter Class 1696
TStringsAdapter Members 1697
TStringsAdapter.Create Constructor 1698
TStringsAdapter Methods 1699

Interfaces 1702
IAmbientDispatch Interface 1702
IAmbientDispatch Members 1702
IAmbientDispatch Properties 1703
IFontAccess Interface 1706
IFontAccess Members 1707
IFontAccess Methods 1707
IPictureAccess Interface 1707
IPictureAccess Members 1708
IPictureAccess Methods 1708

Functions 1709
AxCtrls.EnumDispatchProperties Function 1709
AxCtrls.GetOleFont Function 1710
AxCtrls.GetOlePicture Function 1710
AxCtrls.GetOleStrings Function 1711
AxCtrls.ParkingWindow Function 1711
AxCtrls.SetOleFont Function 1712
AxCtrls.SetOlePicture Function 1712
AxCtrls.SetOleStrings Function 1713

Structs, Records, Enums 1714
AxCtrls.TConnectionKind Enumeration 1714

Types 1715
AxCtrls.TActiveFormClass Type 1715
AxCtrls.TActiveXControlClass Type 1715
AxCtrls.TDefinePropertyPage Type 1716
AxCtrls.TPropertyPageClass Type 1716

Constants 1717
AxCtrls.Class_DColorPropPage Constant 1717
AxCtrls.Class_DFontPropPage Constant 1717
AxCtrls.Class_DPicturePropPage Constant 1718
AxCtrls.Class_DStringPropPage Constant 1718

BandActn Namespace 1719

Classes **1720**
TCustomizeActionBars Class 1720
TCustomizeActionBars Members 1720
TCustomizeActionBars.Create Constructor 1725
TCustomizeActionBars Methods 1725
TCustomizeActionBars Properties 1726

BDEReg Namespace 1729

Classes **1731**
TBatchMoveSprig Class 1731
TBatchMoveSprig Members 1731
TBatchMoveSprig Methods 1732
TBDEDataSetSprig Class 1732
TBDEDataSetSprig Members 1733
TBDEDataSetSprig Methods 1733
TDatabaseSprig Class 1733
TDatabaseSprig Members 1733
TDatabaseSprig Methods 1734
TDBDataSetSprig Class 1736
TDBDataSetSprig Members 1736
TDBDataSetSprig Methods 1736
TDefaultSessionSprig Class 1738
TDefaultSessionSprig Members 1738
TDefaultSessionSprig Methods 1738
TImpliedDatabaseSprig Class 1739
TImpliedDatabaseSprig Members 1740
TImpliedDatabaseSprig Methods 1740
TImpliedSessionSprig Class 1742
TImpliedSessionSprig Members 1742
TImpliedSessionSprig Methods 1742
TNestedTableSprig Class 1743
TNestedTableSprig Members 1743
TNestedTableSprig Methods 1744
TQuerySprig Class 1744
TQuerySprig Members 1744
TQuerySprig Methods 1745
TSessionSprig Class 1745
TSessionSprig Members 1745

TSessionSprig Methods 1746
TStoredProcSprig Class 1747
TStoredProcSprig Members 1747
TStoredProcSprig Methods 1747
TTableFieldLinkProperty Class 1748
TTableFieldLinkProperty Members 1748
TTableFieldLinkProperty Methods 1749
TTableFieldLinkProperty Properties 1751
TTableSprig Class 1753
TTableSprig Members 1753
TTableSprig Methods 1753
TUpdateSQLSprig Class 1754
TUpdateSQLSprig Members 1754
TUpdateSQLSprig Methods 1755

Functions 1756
BDEReg.Register Function 1756
BDEReg.SprigBDEImpliedDatabaseName Function 1756
BDEReg.SprigBDEImpliedSessionName Function 1757
BDEReg.SprigBDESessionName Function 1757

Constants 1758
BDEReg.cDefaultSessionSprigName Constant 1758
BDEReg.cImpliedDatabaseSprigPrefix Constant 1758
BDEReg.cImpliedSessionSprigPrefix Constant 1758

Borland Namespace 1761

Borland.Data Namespace 1762
Borland.Data.AdoDbxClientProvider Namespace 1762
Classes 1763
Types 1890
Constants 1891
Borland.Data.DataStore Namespace 1892
Classes 1893
Types 2129
Borland.Data.MetaData Namespace 2130
Classes 2131

ButtonGroup Namespace 2197

Classes **2198**
TButtonGroup Class 2198
TButtonGroup Members 2198
TButtonGroup.Create Constructor 2209
TButtonGroup.Destroy Destructor 2209
TButtonGroup Methods 2210
TButtonGroup Properties 2220
TButtonGroup Events 2247
TGrpButtonItem Class 2275
TGrpButtonItem Members 2275
TGrpButtonItem Methods 2277
TGrpButtonItem Properties 2278
TGrpButtonItems Class 2278
TGrpButtonItems Members 2279
TGrpButtonItems.Create Constructor 2280
TGrpButtonItems Methods 2281
TGrpButtonItems Properties 2283

Types **2284**
ButtonGroup.TGrpButtonDrawEvent Type 2284
ButtonGroup.TGrpButtonDrawIconEvent Type 2285
ButtonGroup.TGrpButtonEvent Type 2285
ButtonGroup.TGrpButtonItemClass Type 2286
ButtonGroup.TGrpButtonItemsClass Type 2286
ButtonGroup.TGrpButtonOptions Type 2286
ButtonGroup.TGrpButtonReorderEvent Type 2287

Buttons Namespace 2289

Classes **2290**
TBitBtn Class 2290
TBitBtn Members 2291
TBitBtn.Create Constructor 2302
TBitBtn.Destroy Destructor 2302
TBitBtn Methods 2302
TBitBtn Properties 2306
TBitBtn Events 2326
TBitBtnActionLink Class 2330
TBitBtnActionLink Members 2330
TBitBtnActionLink.Create Constructor 2332

TBitBtnActionLink Fields 2333
TBitBtnActionLink Methods 2334
TSpeedButton Class 2335
TSpeedButton Members 2337
TSpeedButton.Create Constructor 2344
TSpeedButton.Destroy Destructor 2344
TSpeedButton Fields 2344
TSpeedButton Methods 2345
TSpeedButton Properties 2349
TSpeedButton Events 2372
TSpeedButtonActionLink Class 2382
TSpeedButtonActionLink Members 2383
TSpeedButtonActionLink.Create Constructor 2385
TSpeedButtonActionLink Fields 2385
TSpeedButtonActionLink Methods 2386

Functions **2391**
Buttons.DrawButtonFace Function 2391

Structs, Records, Enums **2392**
Buttons.TBitBtnKind Enumeration 2392
Buttons.TButtonLayout Enumeration 2393
Buttons.TButtonState Enumeration 2394
Buttons.TButtonStyle Enumeration 2395

Types **2396**
Buttons.TNumGlyphs Type 2396

CaptionedDockTree Namespace 2397

Classes **2398**
TCaptionedDockTree Class 2398
TCaptionedDockTree Members 2398
Create Constructor 2400
TCaptionedDockTree.Destroy Destructor 2401
TCaptionedDockTree Methods 2402
TCaptionedDockTree Properties 2405
TDockCaptionDrawer Class 2405
TDockCaptionDrawer Members 2406
TDockCaptionDrawer.Create Constructor 2406
TDockCaptionDrawer Methods 2407

TDockCaptionDrawer Properties 2407

Structs, Records, Enums 2409

CaptionedDockTree.TDockCaptionOrientation Enumeration 2409

CaptionedDockTree.TDockCaptionPinButton Enumeration 2409

CaptionedDockTree.TParentFormState Record 2410

Types 2411

CaptionedDockTree.TCaptionedDockTreeClass Type 2411

CaptionedDockTree.TDockCaptionDrawerClass Type 2411

CaptionedDockTree.TDockCaptionHitTest Type 2411

Constants 2413

CaptionedDockTree.dchtCaption Constant 2413

CaptionedDockTree.dchtClose Constant 2413

CaptionedDockTree.dchtCustom Constant 2414

CaptionedDockTree.dchtNone Constant 2414

CaptionedDockTree.dchtPin Constant 2414

CategoryButtons Namespace 2415

Classes 2417

TBaseButtonItem Class 2417

TBaseButtonItem Members 2417

TBaseButtonItem.Create Constructor 2419

TBaseButtonItem Methods 2419

TBaseButtonItem Properties 2422

TBaseButtonItem Events 2423

TButtonCategories Class 2424

TButtonCategories Members 2424

TButtonCategories.Create Constructor 2426

TButtonCategories Methods 2426

TButtonCategories Properties 2428

TButtonCategory Class 2429

TButtonCategory Members 2429

TButtonCategory.Create Constructor 2431

TButtonCategory.Destroy Destructor 2431

TButtonCategory Methods 2431

TButtonCategory Properties 2433

TButtonCollection Class 2436

TButtonCollection Members 2436

TButtonCollection.Create Constructor 2438
TButtonCollection Methods 2438
TButtonCollection Properties 2440
TButtonItem Class 2441
TButtonItem Members 2441
TButtonItem Methods 2443
TButtonItem Properties 2444
TButtonItemActionLink Class 2445
TButtonItemActionLink Members 2445
TButtonItemActionLink Fields 2447
TButtonItemActionLink Methods 2447
TCategoryButtons Class 2452
TCategoryButtons Members 2452
TCategoryButtons.Create Constructor 2464
TCategoryButtons.Destroy Destructor 2464
TCategoryButtons Methods 2464
TCategoryButtons Properties 2479
TCategoryButtons Events 2507

Structs, Records, Enums 2543
CategoryButtons.TCatButtonFlow Enumeration 2543

Types 2544
CategoryButtons.TBaseButtonItemClass Type 2544
CategoryButtons.TButtonCategoriesClass Type 2544
CategoryButtons.TButtonCategoryClass Type 2545
CategoryButtons.TButtonDrawState Type 2545
CategoryButtons.TButtonItemActionLinkClass Type 2546
CategoryButtons.TButtonItemClass Type 2546
CategoryButtons.TCatButtonCopyEvent Type 2546
CategoryButtons.TCatButtonDrawEvent Type 2547
CategoryButtons.TCatButtonDrawIconEvent Type 2547
CategoryButtons.TCatButtonEvent Type 2547
CategoryButtons.TCatButtonGetHint Type 2548
CategoryButtons.TCatButtonOptions Type 2548
CategoryButtons.TCatButtonReorderEvent Type 2549
CategoryButtons.TCategoryCollapseEvent Type 2549
CategoryButtons.TCategoryReorderEvent Type 2549

Constants 2551
CategoryButtons.crDragCopy Constant 2551

CGIApp Namespace 2553

Classes **2554**

TCGIApplication Class 2554

TCGIApplication Members 2555

TCGIApplication.Destroy Destructor 2557

TCGIApplication Methods 2558

TCGIFactory Class 2559

TCGIFactory Members 2559

TCGIFactory.Create Constructor 2560

TCGIFactory Methods 2560

TCGIFactory Properties 2561

CGIHTTP Namespace 2563

Classes **2564**

TCGIRequest Class 2564

TCGIRequest Members 2565

TCGIRequest.Create Constructor 2567

TCGIRequest Methods 2567

TCGIResponse Class 2572

TCGIResponse Members 2573

TCGIResponse.Create Constructor 2575

TCGIResponse Methods 2575

TWinCGIRequest Class 2583

TWinCGIRequest Members 2584

TWinCGIRequest.Create Constructor 2586

TWinCGIRequest.Destroy Destructor 2586

TWinCGIRequest Methods 2587

TWinCGIResponse Class 2590

Types **2591**

CGIHTTP.TCGIRequestClass Type 2591

CGIHTTP.TCGIResponseClass Type 2591

CGIHTTP.TWinCGIRequestClass Type 2592

CGIHTTP.TWinCGIResponseClass Type 2592

CheckLst Namespace 2593

Classes **2594**

TCheckListBox Class 2594
TCheckListBox Members 2594
TCheckListBox.Create Constructor 2606
TCheckListBox.Destroy Destructor 2606
TCheckListBox Methods 2607
TCheckListBox Properties 2613
TCheckListBox Events 2649

Classes Namespace 2685

Classes 2692
EBitsError Class 2693
EClassNotFound Class 2694
EComponentError Class 2694
EFCreateError Class 2695
EFilerError Class 2696
EFileStreamError Class 2697
EFileStreamError Members 2697
EFileStreamError.Create Constructor 2698
EFOpenError Class 2699
EInvalidImage Class 2699
EInvalidOperation Class 2700
EListError Class 2701
EMethodNotFound Class 2701
EOutOfResources Class 2702
EParserError Class 2702
EReadError Class 2703
EResNotFound Class 2704
EStreamError Class 2704
EStringListError Class 2705
EThread Class 2706
EWriteError Class 2706
TBasicAction Class 2707
TBasicAction Members 2707
TBasicAction.Create Constructor 2710
TBasicAction.Destroy Destructor 2711
TBasicAction Fields 2711
TBasicAction Methods 2711
TBasicAction Properties 2715

TBasicAction Events 2716
TBasicActionLink Class 2720
TBasicActionLink Members 2720
TBasicActionLink.Create Constructor 2721
TBasicActionLink.Destroy Destructor 2722
TBasicActionLink Fields 2722
TBasicActionLink Methods 2722
TBasicActionLink Properties 2724
TBasicActionLink Events 2725
TBits Class 2725
TBits Members 2726
TBits.Destroy Destructor 2726
TBits Methods 2727
TBits Properties 2727
TClassFinder Class 2728
TClassFinder Members 2729
TClassFinder.Create Constructor 2729
TClassFinder.Destroy Destructor 2729
TClassFinder Methods 2730
TCollection Class 2731
TCollection Members 2733
TCollection.Create Constructor 2734
TCollection.Destroy Destructor 2735
TCollection Methods 2735
TCollection Properties 2748
TCollectionEnumerator Class 2753
TCollectionEnumerator Members 2753
TCollectionEnumerator.Create Constructor 2754
TCollectionEnumerator Methods 2754
TCollectionEnumerator Properties 2755
TCollectionItem Class 2755
TCollectionItem Members 2756
TCollectionItem.Create Constructor 2757
TCollectionItem.Destroy Destructor 2757
TCollectionItem Methods 2757
TCollectionItem Properties 2760
TComponent Class 2762
TComponent Members 2763
TComponent.Create Constructor 2766

TComponent.Destroy Destructor 2766
TComponent Fields 2767
TComponent Methods 2768
TComponent Properties 2793
TComponentEnumerator Class 2812
TComponentEnumerator Members 2812
TComponentEnumerator.Create Constructor 2813
TComponentEnumerator Methods 2813
TComponentEnumerator Properties 2814
TCustomMemoryStream Class 2815
TCustomMemoryStream Members 2815
TCustomMemoryStream Methods 2817
TCustomMemoryStream Properties 2823
TDataModule Class 2824
TDataModule Members 2824
TDataModule.Create Constructor 2827
TDataModule.CreateNew Constructor 2828
TDataModule.Destroy Destructor 2829
TDataModule Methods 2829
TDataModule Properties 2832
TDataModule Events 2833
TFiler Class 2834
TFiler Members 2835
TFiler.Create Constructor 2836
TFiler.Destroy Destructor 2837
TFiler Methods 2837
TFiler Properties 2839
TFileStream Class 2841
TFileStream Members 2841
Create Constructor 2843
TFileStream.Destroy Destructor 2844
TFileStream Fields 2844
TFileStream Properties 2845
THandleStream Class 2845
THandleStream Members 2846
THandleStream.Create Constructor 2847
THandleStream Fields 2847
THandleStream Methods 2848
THandleStream Properties 2850

TInterfacedPersistent Class 2851
TInterfacedPersistent Members 2852
TInterfacedPersistent Methods 2853
TInterfaceList Class 2855
TInterfaceList Members 2855
TInterfaceList.Create Constructor 2857
TInterfaceList.Destroy Destructor 2857
TInterfaceList Methods 2857
TInterfaceList Properties 2865
TInterfaceListEnumerator Class 2866
TInterfaceListEnumerator Members 2867
TInterfaceListEnumerator.Create Constructor 2867
TInterfaceListEnumerator Methods 2867
TInterfaceListEnumerator Properties 2868
TList Class 2868
TList Members 2869
TList.Destroy Destructor 2870
TList Methods 2871
TList Properties 2895
TListEnumerator Class 2900
TListEnumerator Members 2901
TListEnumerator.Create Constructor 2901
TListEnumerator Methods 2901
TListEnumerator Properties 2902
TMemoryStream Class 2902
TMemoryStream Members 2906
TMemoryStream.Destroy Destructor 2908
TMemoryStream Methods 2908
TMemoryStream Properties 2911
TOwnedCollection Class 2912
TOwnedCollection Members 2912
TOwnedCollection.Create Constructor 2914
TOwnedCollection Methods 2914
TParser Class 2915
TParser Members 2916
TParser.Create Constructor 2917
TParser.Destroy Destructor 2917
TParser Methods 2918
TParser Properties 2925

TParser Events 2927
TPersistent Class 2928
TPersistent Members 2928
TPersistent.Destroy Destructor 2929
TPersistent Methods 2930
TReader Class 2933
TReader Members 2934
TReader.Destroy Destructor 2936
TReader Methods 2937
TReader Properties 2958
TReader Events 2960
TRecall Class 2964
TRecall Members 2964
TRecall.Create Constructor 2965
TRecall.Destroy Destructor 2966
TRecall Methods 2967
TRecall Properties 2968
TResourceStream Class 2968
TResourceStream Members 2969
TResourceStream.Create Constructor 2970
TResourceStream.CreateFromID Constructor 2971
TResourceStream.Destroy Destructor 2972
TResourceStream Methods 2972
TStream Class 2973
TStream Members 2978
TStream Methods 2979
TStream Properties 2996
TStreamAdapter Class 2997
TStreamAdapter Members 2998
TStreamAdapter.Create Constructor 2999
TStreamAdapter.Destroy Destructor 3000
TStreamAdapter Methods 3000
TStreamAdapter Properties 3005
TStringList Class 3006
TStringList Members 3007
TStringList.Destroy Destructor 3009
TStringList Methods 3009
TStringList Properties 3025
TStringList Events 3027

TStrings Class 3029
TStrings Members 3030
TStrings.Destroy Destructor 3032
TStrings Methods 3032
TStrings Properties 3069
TStringsEnumerator Class 3086
TStringsEnumerator Members 3086
TStringsEnumerator.Create Constructor 3087
TStringsEnumerator Methods 3087
TStringsEnumerator Properties 3088
TStringStream Class 3088
TStringStream Members 3092
TStringStream.Create Constructor 3093
TStringStream Methods 3097
TStringStream Properties 3104
TThread Class 3108
TThread Members 3109
TThread.Create Constructor 3110
TThread.Destroy Destructor 3113
TThread Methods 3114
TThread Properties 3133
TThread Events 3148
TThreadList Class 3149
TThreadList Members 3150
TThreadList.Create Constructor 3150
TThreadList.Destroy Destructor 3150
TThreadList Methods 3151
TThreadList Properties 3174
TWriter Class 3175
TWriter Members 3176
TWriter.Destroy Destructor 3178
TWriter Methods 3178
TWriter Properties 3191
TWriter Events 3193

Interfaces 3195

IDesignerNotify Interface 3195
IDesignerNotify Members 3195
IDesignerNotify Methods 3196
IInterfaceComponentReference Interface 3197

IInterfaceComponentReference Members 3197
IInterfaceComponentReference Methods 3197
IInterfaceList Interface 3198
IInterfaceList Members 3198
IInterfaceList Methods 3199
IInterfaceList Properties 3205
IStreamPersist Interface 3207
IStreamPersist Members 3207
IStreamPersist Methods 3207
IStringsAdapter Interface 3208
IStringsAdapter Members 3209
IStringsAdapter Methods 3209
IVarStreamable Interface 3210
IVarStreamable Members 3210
IVarStreamable Methods 3210
IVCLComObject Interface 3211
IVCLComObject Members 3212
IVCLComObject Methods 3212

Functions 3214

Classes.ActivateClassGroup Function 3215
Classes.ActiveClassGroup Function 3216
Classes.AllocateHWnd Function 3217
Classes.AncestorIsValid Function 3217
Classes.BeginGlobalLoading Function 3218
Classes.BinToHex Function 3218
Classes.Bounds Function 3218
Classes.CheckSynchronize Function 3219
Classes.ClassGroupOf Function 3220
Classes.CollectionsEqual Function 3220
Classes.CountGenerations Function 3221
Classes.DeallocateHWnd Function 3221
Classes.EndGlobalLoading Function 3222
Classes.ExtractStrings Function 3222
Classes.FindClass Function 3223
Classes.FindGlobalComponent Function 3223
Classes.FindIdentToInt Function 3224
Classes.FindIntToIdent Function 3224
Classes.FindNestedComponent Function 3225
Classes.FindRootDesigner Function 3225

Classes.FreeObjectInstance Function 3226
Classes.GetClass Function 3226
Classes.GetFixupInstanceNames Function 3228
Classes.GetFixupReferenceNames Function 3228
Classes.GetUltimateOwner Function 3228
Classes.GlobalFixupReferences Function 3229
Classes.GroupDescendentsWith Function 3229
Classes.HexToBin Function 3230
Classes.IdentToInt Function 3230
Classes.InitComponentRes Function 3231
Classes.InitInheritedComponent Function 3231
Classes.IntToIdent Function 3232
Classes.InvalidPoint Function 3233
Classes.IsDefaultPropertyValue Function 3233
Classes.IsUniqueGlobalComponentName Function 3234
Classes.LineStart Function 3234
Classes.MakeObjectInstance Function 3235
Classes.NotifyGlobalLoading Function 3235
Classes.ObjectBinaryToText Function 3235
Classes.ObjectResourceToText Function 3239
Classes.ObjectTextToBinary Function 3240
Classes.ObjectTextToResource Function 3244
Classes.Point Function 3244
Classes.PointsEqual Function 3245
Classes.ReadComponentRes Function 3245
Classes.ReadComponentResEx Function 3246
Classes.ReadComponentResFile Function 3247
Classes.Rect Function 3249
Classes.RedirectFixupReferences Function 3250
Classes.RegisterClass Function 3250
Classes.RegisterClassAlias Function 3254
Classes.RegisterClasses Function 3255
Classes.RegisterComponents Function 3256
Classes.RegisterFindGlobalComponentProc Function 3259
Classes.RegisterIntegerConsts Function 3259
Classes.RegisterNoIcon Function 3260
Classes.RegisterNonActiveX Function 3260
Classes.RemoveFixupReferences Function 3261
Classes.RemoveFixups Function 3261

Classes.SmallPoint Function 3262
Classes.StartClassGroup Function 3262
Classes.TestStreamFormat Function 3263
Classes.UnRegisterClass Function 3263
Classes.UnRegisterClasses Function 3264
Classes.UnregisterFindGlobalComponentProc Function 3264
Classes.UnregisterIntegerConsts Function 3264
Classes.UnRegisterModuleClasses Function 3265
Classes.WriteComponentResFile Function 3266

Structs, Records, Enums 3268

Classes.TActiveXRegType Enumeration 3268
Classes.TAlignment Enumeration 3269
Classes.TBiDiMode Enumeration 3269
Classes.TCollectionNotification Enumeration 3270
Classes.TDuplicates Enumeration 3271
Classes.TFilerFlag Enumeration 3271
Classes.THelpType Enumeration 3272
Classes.TIdentMapEntry Record 3273
Classes.TListAssignOp Enumeration 3273
Classes.TListNotification Enumeration 3274
Classes.TOperation Enumeration 3275
Classes.TSeekOrigin Enumeration 3276
Classes.TStreamOriginalFormat Enumeration 3276
Classes.TStreamOwnership Enumeration 3277
Classes.TStringItem Record 3277
Classes.TSynchronizeRecord Record 3278
Classes.TThreadPriority Enumeration 3278
Classes.TValueType Enumeration 3279
Classes.TVerticalAlignment Enumeration 3280

Types 3282

Classes.PPointerList Type 3283
Classes.PStringItem Type 3284
Classes.PStringItemList Type 3284
Classes.PSynchronizeRecord Type 3285
Classes.TAncestorNotFoundEvent Type 3285
Classes.TBasicActionClass Type 3285
Classes.TBasicActionLinkClass Type 3286
Classes.TCollectionItemClass Type 3286

Classes.TComponentClass Type 3287
Classes.TComponentName Type 3287
Classes.TComponentState Type 3287
Classes.TComponentStyle Type 3288
Classes.TCreateComponentEvent Type 3289
Classes.TFilerFlags Type 3289
Classes.TFindAncestorEvent Type 3290
Classes.TFindComponentClassEvent Type 3290
Classes.TFindComponentInstanceEvent Type 3291
Classes.TFindGlobalComponent Type 3291
Classes.TFindMethodEvent Type 3291
Classes.TFindMethodInstanceEvent Type 3292
Classes.TFindMethodNameEvent Type 3292
Classes.TGetChildProc Type 3292
Classes.TGetClass Type 3293
Classes.TGetLookupInfoEvent Type 3293
Classes.TGetModuleProc Type 3294
Classes.TGetStrProc Type 3294
Classes.THelpContext Type 3295
Classes.TIdentToInt Type 3295
Classes.TIntToIdent Type 3296
Classes.TIsUniqueGlobalComponentName Type 3296
Classes.TLeftRight Type 3297
Classes.TListSortCompare Type 3297
Classes.TNotifyEvent Type 3298
Classes.TParserErrorEvent Type 3298
Classes.TPersistentClass Type 3299
Classes.TPointerList Type 3299
Classes.TReadComponentsProc Type 3300
Classes.TReaderError Type 3300
Classes.TReaderProc Type 3301
Classes.TReferenceNameEvent Type 3301
Classes.TSetNameEvent Type 3302
Classes.TShiftState Type 3302
Classes.TShortCut Type 3303
Classes.TStreamProc Type 3304
Classes.TStringItemList Type 3304
Classes.TStringListSortCompare Type 3305
Classes.TStringsDefined Type 3306

Classes.TThreadMethod Type 3306
Classes.TTopBottom Type 3306
Classes.TWndMethod Type 3307
Classes.TWriterProc Type 3307

Variables 3309
Classes.AddDataModule Variable 3309
Classes.ApplicationHandleException Variable 3310
Classes.ApplicationShowException Variable 3310
Classes.CreateVCLComObjectProc Variable 3310
Classes.CurrentGroup Variable 3311
Classes.GlobalNameSpace Variable 3311
Classes.IsUniqueGlobalComponentNameProc Variable 3311
Classes.RegisterComponentsProc Variable 3312
Classes.RegisterNoIconProc Variable 3313
Classes.RegisterNonActiveXProc Variable 3313
Classes.RemoveDataModule Variable 3314
Classes.WakeMainThread Variable 3314

Constants 3316
Classes.fmCreate Constant 3316
Classes.MaxListSize Constant 3317
Classes.scAlt Constant 3317
Classes.scCtrl Constant 3318
Classes.scNone Constant 3318
Classes.scShift Constant 3319
Classes.soFromBeginning Constant 3319
Classes.soFromCurrent Constant 3320
Classes.soFromEnd Constant 3320
Classes.toEOF Constant 3321
Classes.toFloat Constant 3322
Classes.toInteger Constant 3322
Classes.toString Constant 3323
Classes.toSymbol Constant 3324
Classes.toWString Constant 3324

Clipbrd Namespace 3327

Classes 3328
TClipboard Class 3328
TClipboard Members 3329

TClipboard.Destroy Destructor 3330
TClipboard Methods 3330
TClipboard Properties 3346

Functions 3351
Clipbrd.Clipboard Function 3351
Clipbrd.SetClipboard Function 3353

Variables 3354
Clipbrd.CF_COMPONENT Variable 3354
Clipbrd.CF_PICTURE Variable 3354

CmAdmCtl Namespace 3355

Classes 3356
CoCOMAdminCatalog Class 3356
CoCOMAdminCatalog Members 3356
CoCOMAdminCatalog Methods 3356
CoCOMAdminCatalogCollection Class 3357
CoCOMAdminCatalogCollection Members 3357
CoCOMAdminCatalogCollection Methods 3357
CoCOMAdminCatalogObject Class 3358
CoCOMAdminCatalogObject Members 3358
CoCOMAdminCatalogObject Methods 3358
TCOMAdminCatalog Class 3359
TCOMAdminCatalog Members 3360
TCOMAdminCatalog.Create Constructor 3363
TCOMAdminCatalog.Destroy Destructor 3364
TCOMAdminCatalog Methods 3364
TCOMAdminCatalog Properties 3379
TCOMAdminCatalogCollection Class 3380
TCOMAdminCatalogCollection Members 3381
TCOMAdminCatalogCollection.Create Constructor 3384
TCOMAdminCatalogCollection.Destroy Destructor 3385
TCOMAdminCatalogCollection Methods 3385
TCOMAdminCatalogCollection Properties 3393
TCOMAdminCatalogObject Class 3396
TCOMAdminCatalogObject Members 3396
TCOMAdminCatalogObject.Create Constructor 3400
TCOMAdminCatalogObject.Destroy Destructor 3400
TCOMAdminCatalogObject Methods 3400

TCOMAdminCatalogObject Properties 3405

ComApp Namespace 3409

Classes 3410

TWebAppAutoObjectFactory Class 3410

TWebAppAutoObjectFactory Members 3410

TWebAppAutoObjectFactory.Create Constructor 3410

ComCtrls Namespace 3413

Classes 3419

ECommonCalendarError Class 3420

EDateTimeError Class 3421

EMonthCalError Class 3421

ETreeViewError Class 3422

TAnimate Class 3422

TAnimate Members 3423

TAnimate.Create Constructor 3433

TAnimate Methods 3435

TAnimate Properties 3440

TAnimate Events 3461

TComboBoxEx Class 3463

TComboBoxEx Members 3463

TComboBoxEx Properties 3476

TComboBoxEx Events 3504

TComboBoxExActionLink Class 3531

TComboBoxExActionLink Members 3531

TComboBoxExActionLink Methods 3534

TComboBoxExStrings Class 3535

TComboBoxExStrings Members 3535

TComboBoxExStrings.Create Constructor 3537

TComboBoxExStrings.Destroy Destructor 3537

TComboBoxExStrings Methods 3538

TComboBoxExStrings Properties 3545

TComboExItem Class 3546

TComboExItem Members 3546

TComboExItem Methods 3548

TComboExItem Properties 3551

TComboExItems Class 3552

TComboExItems Members 3553
TComboExItems Methods 3555
TComboExItems Properties 3557
TCommonCalendar Class 3558
TCommonCalendar Members 3558
TCommonCalendar.Create Constructor 3569
TCommonCalendar.Destroy Destructor 3570
TCommonCalendar Methods 3570
TCommonCalendar Properties 3575
TCommonCalendar Events 3581
TConversion Class 3582
TConversion Members 3582
TConversion Methods 3583
TCoolBand Class 3584
TCoolBand Members 3585
TCoolBand.Create Constructor 3586
TCoolBand.Destroy Destructor 3586
TCoolBand Methods 3587
TCoolBand Properties 3588
TCoolBands Class 3596
TCoolBands Members 3597
TCoolBands.Create Constructor 3598
TCoolBands Methods 3599
TCoolBands Properties 3601
TCoolBar Class 3603
TCoolBar Members 3604
TCoolBar.Create Constructor 3614
TCoolBar.Destroy Destructor 3615
TCoolBar Methods 3616
TCoolBar Properties 3625
TCoolBar Events 3649
TCustomComboBoxEx Class 3678
TCustomComboBoxEx Members 3679
TCustomComboBoxEx.Create Constructor 3691
TCustomComboBoxEx.Destroy Destructor 3691
TCustomComboBoxEx Methods 3691
TCustomComboBoxEx Properties 3699
TCustomComboBoxEx Events 3702
TCustomHeaderControl Class 3702

TCustomHeaderControl Members 3703
TCustomHeaderControl.Create Constructor 3714
TCustomHeaderControl.Destroy Destructor 3714
TCustomHeaderControl Methods 3714
TCustomHeaderControl Properties 3723
TCustomHeaderControl Events 3736
TCustomHotKey Class 3739
TCustomHotKey Members 3739
TCustomHotKey.Create Constructor 3750
TCustomHotKey Methods 3750
TCustomHotKey Properties 3752
TCustomHotKey Events 3756
TCustomListView Class 3757
TCustomListView Members 3757
TCustomListView.Create Constructor 3770
TCustomListView.Destroy Destructor 3771
TCustomListView Methods 3772
TCustomListView Properties 3802
TCustomListView Events 3833
TCustomRichEdit Class 3854
TCustomRichEdit Members 3854
TCustomRichEdit.Create Constructor 3866
TCustomRichEdit.Destroy Destructor 3866
TCustomRichEdit Methods 3867
TCustomRichEdit Properties 3877
TCustomRichEdit Events 3888
TCustomStatusBar Class 3892
TCustomStatusBar Members 3892
TCustomStatusBar.Create Constructor 3903
TCustomStatusBar.Destroy Destructor 3903
TCustomStatusBar Methods 3904
TCustomStatusBar Properties 3910
TCustomStatusBar Events 3919
TCustomTabControl Class 3921
TCustomTabControl Members 3921
TCustomTabControl.Create Constructor 3932
TCustomTabControl.Destroy Destructor 3932
TCustomTabControl Methods 3933
TCustomTabControl Properties 3939

TCustomTabControl Events 3951
TCustomTreeView Class 3957
TCustomTreeView Members 3957
TCustomTreeView.Create Constructor 3969
TCustomTreeView.Destroy Destructor 3970
TCustomTreeView Fields 3970
TCustomTreeView Methods 3970
TCustomTreeView Properties 3995
TCustomTreeView Events 4015
TCustomUpDown Class 4027
TCustomUpDown Members 4028
TCustomUpDown.Create Constructor 4038
TCustomUpDown Methods 4038
TCustomUpDown Properties 4041
TCustomUpDown Events 4046
TDateTimePicker Class 4049
TDateTimePicker Members 4049
TDateTimePicker.Create Constructor 4060
TDateTimePicker Methods 4061
TDateTimePicker Properties 4065
TDateTimePicker Events 4096
THeaderControl Class 4118
THeaderControl Members 4119
THeaderControl Properties 4130
THeaderControl Events 4152
THeaderSection Class 4173
THeaderSection Members 4173
THeaderSection.Create Constructor 4175
THeaderSection Methods 4175
THeaderSection Properties 4178
THeaderSections Class 4185
THeaderSections Members 4185
THeaderSections.Create Constructor 4187
THeaderSections Methods 4187
THeaderSections Properties 4190
THotKey Class 4190
THotKey Members 4191
THotKey Properties 4201
THotKey Events 4219

TIconOptions Class 4233
TIconOptions Members 4234
TIconOptions.Create Constructor 4235
TIconOptions Properties 4235
TListColumn Class 4236
TListColumn Members 4237
TListColumn.Create Constructor 4238
TListColumn.Destroy Destructor 4239
TListColumn Methods 4239
TListColumn Properties 4241
TListColumns Class 4245
TListColumns Members 4246
TListColumns.Create Constructor 4247
TListColumns Methods 4248
TListColumns Properties 4250
TListItem Class 4250
TListItem Members 4251
TListItem.Create Constructor 4252
TListItem.Destroy Destructor 4252
TListItem Methods 4253
TListItem Properties 4257
TListItems Class 4270
TListItems Members 4271
TListItems.Create Constructor 4272
TListItems.Destroy Destructor 4272
TListItems Methods 4273
TListItems Properties 4281
TListItemsEnumerator Class 4283
TListItemsEnumerator Members 4283
TListItemsEnumerator.Create Constructor 4284
TListItemsEnumerator Methods 4284
TListItemsEnumerator Properties 4284
TListView Class 4285
TListView Members 4285
TListView Properties 4298
TListView Events 4345
TListViewActionLink Class 4388
TListViewActionLink Members 4389
TListViewActionLink Methods 4391

TMonthCalColors Class 4392
TMonthCalColors Members 4392
TMonthCalColors.Create Constructor 4394
TMonthCalColors Methods 4394
TMonthCalColors Properties 4394
TMonthCalendar Class 4397
TMonthCalendar Members 4397
TMonthCalendar.Create Constructor 4408
TMonthCalendar Methods 4408
TMonthCalendar Properties 4411
TMonthCalendar Events 4437
TPageControl Class 4459
TPageControl Members 4459
TPageControl.Create Constructor 4470
TPageControl.Destroy Destructor 4472
TPageControl Methods 4472
TPageControl Properties 4481
TPageControl Events 4515
TPageScroller Class 4548
TPageScroller Members 4548
TPageScroller.Create Constructor 4558
TPageScroller Methods 4559
TPageScroller Properties 4561
TPageScroller Events 4585
TParaAttributes Class 4606
TParaAttributes Members 4607
TParaAttributes.Create Constructor 4608
TParaAttributes Methods 4608
TParaAttributes Properties 4609
TProgressBar Class 4615
TProgressBar Members 4616
TProgressBar.Create Constructor 4626
TProgressBar Methods 4627
TProgressBar Properties 4632
TProgressBar Events 4664
TRichEdit Class 4685
TRichEdit Members 4686
TRichEdit Properties 4698
TRichEdit Events 4737

TStatusBar Class 4769
TStatusBar Members 4770
TStatusBar Properties 4780
TStatusBar Events 4803
TStatusPanel Class 4825
TStatusPanel Members 4825
TStatusPanel.Create Constructor 4827
TStatusPanel Methods 4827
TStatusPanel Properties 4829
TStatusPanels Class 4834
TStatusPanels Members 4834
TStatusPanels.Create Constructor 4836
TStatusPanels Methods 4836
TStatusPanels Properties 4840
TTabControl Class 4841
TTabControl Members 4842
TTabControl Properties 4853
TTabControl Events 4884
TTabSheet Class 4919
TTabSheet Members 4920
TTabSheet.Create Constructor 4930
TTabSheet.Destroy Destructor 4932
TTabSheet Methods 4932
TTabSheet Properties 4934
TTabSheet Events 4960
TTextAttributes Class 4982
TTextAttributes Members 4982
TTextAttributes.Create Constructor 4983
TTextAttributes Methods 4984
TTextAttributes Properties 4987
TToolBar Class 4994
TToolBar Members 4995
TToolBar.Create Constructor 5006
TToolBar.Destroy Destructor 5008
TToolBar Methods 5008
TToolBar Properties 5020
TToolBar Events 5062
TToolBarDockObject Class 5102
TToolBarDockObject Members 5102

TToolBarDockObject Methods 5104
TToolBarEnumerator Class 5105
TToolBarEnumerator Members 5106
TToolBarEnumerator.Create Constructor 5106
TToolBarEnumerator Methods 5106
TToolBarEnumerator Properties 5107
TToolButton Class 5107
TToolButton Members 5108
TToolButton.Create Constructor 5116
TToolButton Fields 5117
TToolButton Methods 5118
TToolButton Properties 5126
TToolButton Events 5148
TToolButtonActionLink Class 5166
TToolButtonActionLink Members 5167
TToolButtonActionLink Fields 5169
TToolButtonActionLink Methods 5169
TTrackBar Class 5172
TTrackBar Members 5173
TTrackBar.Create Constructor 5184
TTrackBar Methods 5185
TTrackBar Properties 5187
TTrackBar Events 5211
TTreeNode Class 5230
TTreeNode Members 5230
TTreeNode.Create Constructor 5232
TTreeNode.Destroy Destructor 5233
TTreeNode Methods 5233
TTreeNode Properties 5251
TTreeNodes Class 5266
TTreeNodes Members 5266
TTreeNodes.Create Constructor 5268
TTreeNodes.Destroy Destructor 5269
TTreeNodes Methods 5269
TTreeNodes Properties 5286
TTreeNodesEnumerator Class 5291
TTreeNodesEnumerator Members 5291
TTreeNodesEnumerator.Create Constructor 5291
TTreeNodesEnumerator Methods 5291

TTreeNodesEnumerator Properties 5292
TTreeView Class 5293
TTreeView Members 5293
TTreeView Properties 5305
TTreeView Events 5341
TUpDown Class 5380
TUpDown Members 5381
TUpDown Properties 5392
TUpDown Events 5409
TWorkArea Class 5425
TWorkArea Members 5425
TWorkArea.Create Constructor 5427
TWorkArea Methods 5427
TWorkArea Properties 5428
TWorkAreas Class 5429
TWorkAreas Members 5429
TWorkAreas Methods 5431
TWorkAreas Properties 5434

Functions 5435
ComCtrls.CheckCommonControl Function 5435
ComCtrls.CheckToolMenuDropdown Function 5435
ComCtrls.GetComCtlVersion Function 5436
ComCtrls.InitCommonControl Function 5437

Structs, Records, Enums 5438
ComCtrls.TAddMode Enumeration 5439
ComCtrls.TAttributeType Enumeration 5440
ComCtrls.TAutoCompleteOption Enumeration 5440
ComCtrls.TCalDayOfWeek Enumeration 5441
ComCtrls.TComboBoxExStyle Enumeration 5442
ComCtrls.TComboBoxExStyleEx Enumeration 5442
ComCtrls.TCommonAVI Enumeration 5443
ComCtrls.TConsistentAttribute Enumeration 5444
ComCtrls.TConversionFormat Record 5444
ComCtrls.TCoolBandMaximize Enumeration 5445
ComCtrls.TCustomDrawStage Enumeration 5445
ComCtrls.TCustomDrawTarget Enumeration 5446
ComCtrls.TDateTimeKind Enumeration 5447
ComCtrls.TDisplayCode Enumeration 5447

ComCtrls.TDTCalAlignment Enumeration 5447
ComCtrls.TDTDateFormat Enumeration 5448
ComCtrls.TDTDateMode Enumeration 5448
ComCtrls.THeaderSectionStyle Enumeration 5449
ComCtrls.THeaderStyle Enumeration 5449
ComCtrls.THitTest Enumeration 5450
ComCtrls.THKInvalidKey Enumeration 5451
ComCtrls.THKModifier Enumeration 5451
ComCtrls.TIconArrangement Enumeration 5452
ComCtrls.TItemChange Enumeration 5452
ComCtrls.TItemFind Enumeration 5453
ComCtrls.TItemRequests Enumeration 5454
ComCtrls.TItemState Enumeration 5454
ComCtrls.TListArrangement Enumeration 5455
ComCtrls.TListHotTrackStyle Enumeration 5456
ComCtrls.TMultiSelectStyles Enumeration 5456
ComCtrls.TNodeAttachMode Enumeration 5457
ComCtrls.TNodeCache Record 5457
ComCtrls.TNodeDataInfo Record 5458
ComCtrls.TNodeInfo Record 5458
ComCtrls.TNodeState Enumeration 5459
ComCtrls.TNumberingStyle Enumeration 5459
ComCtrls.TPageScrollerButton Enumeration 5460
ComCtrls.TPageScrollerButtonState Enumeration 5460
ComCtrls.TPageScrollerOrientation Enumeration 5461
ComCtrls.TPositionToolTip Enumeration 5462
ComCtrls.TProgressBarOrientation Enumeration 5462
ComCtrls.TRichEditStreamInfo Record 5463
ComCtrls.TSearchDirection Enumeration 5463
ComCtrls.TSearchType Enumeration 5464
ComCtrls.TSectionTrackState Enumeration 5464
ComCtrls.TSortType Enumeration 5465
ComCtrls.TStatusPanelBevel Enumeration 5466
ComCtrls.TStatusPanelStyle Enumeration 5466
ComCtrls.TTabPosition Enumeration 5467
ComCtrls.TTabStyle Enumeration 5467
ComCtrls.TTBDrawingStyle Enumeration 5468
ComCtrls.TTickMark Enumeration 5468
ComCtrls.TTickStyle Enumeration 5469

ComCtrls.TToolButtonState Enumeration 5470
ComCtrls.TToolButtonStyle Enumeration 5470
ComCtrls.TTrackBarOrientation Enumeration 5471
ComCtrls.TUDAlignButton Enumeration 5471
ComCtrls.TUDBtnType Enumeration 5472
ComCtrls.TUDOrientation Enumeration 5472
ComCtrls.TUpDownDirection Enumeration 5473
ComCtrls.TViewStyle Enumeration 5473

Types **5477**

ComCtrls.PConversionFormat Type 5479
ComCtrls.PNodeCache Type 5479
ComCtrls.PNodeInfo Type 5480
ComCtrls.PRichEditStreamInfo Type 5480
ComCtrls.TAutoCompleteOptions Type 5480
ComCtrls.TComboBoxExStyles Type 5481
ComCtrls.TComboExItemClass Type 5481
ComCtrls.TComboExItemsClass Type 5482
ComCtrls.TConsistentAttributes Type 5482
ComCtrls.TConversionClass Type 5482
ComCtrls.TCustomDrawPanelEvent Type 5483
ComCtrls.TCustomDrawSectionEvent Type 5483
ComCtrls.TCustomDrawState Type 5483
ComCtrls.TCustomHCCreateSectionClassEvent Type 5484
ComCtrls.TCustomSectionNotifyEvent Type 5484
ComCtrls.TCustomSectionTrackEvent Type 5485
ComCtrls.TDateTimeColors Type 5485
ComCtrls.TDrawPanelEvent Type 5485
ComCtrls.TDrawSectionEvent Type 5486
ComCtrls.TDrawTabEvent Type 5486
ComCtrls.TDTParseInputEvent Type 5487
ComCtrls.THCCreateSectionClassEvent Type 5487
ComCtrls.THeaderSectionClass Type 5487
ComCtrls.THitTests Type 5488
ComCtrls.THKInvalidKeys Type 5488
ComCtrls.THKModifiers Type 5488
ComCtrls.TItemRequest Type 5489
ComCtrls.TItemStates Type 5489
ComCtrls.TListHotTrackStyles Type 5489
ComCtrls.TListItemClass Type 5490

ComCtrls.TLVAdvancedCustomDrawEvent Type 5490
ComCtrls.TLVAdvancedCustomDrawItemEvent Type 5491
ComCtrls.TLVAdvancedCustomDrawSubItemEvent Type 5491
ComCtrls.TLVChangeEvent Type 5491
ComCtrls.TLVChangingEvent Type 5492
ComCtrls.TLVColumnClickEvent Type 5492
ComCtrls.TLVColumnRClickEvent Type 5492
ComCtrls.TLVCompareEvent Type 5493
ComCtrls.TLVCreateItemClassEvent Type 5493
ComCtrls.TLVCustomDrawEvent Type 5493
ComCtrls.TLVCustomDrawItemEvent Type 5494
ComCtrls.TLVCustomDrawSubItemEvent Type 5494
ComCtrls.TLVDeletedEvent Type 5494
ComCtrls.TLVDrawItemEvent Type 5495
ComCtrls.TLVEditedEvent Type 5495
ComCtrls.TLVEditingEvent Type 5496
ComCtrls.TLVInfoTipEvent Type 5496
ComCtrls.TLVNotifyEvent Type 5496
ComCtrls.TLVOwnerDataEvent Type 5497
ComCtrls.TLVOwnerDataFindEvent Type 5497
ComCtrls.TLVOwnerDataHintEvent Type 5498
ComCtrls.TLVOwnerDataStateChangeEvent Type 5498
ComCtrls.TLVSelectItemEvent Type 5499
ComCtrls.TLVSubItemImageEvent Type 5499
ComCtrls.TMultiSelectStyle Type 5499
ComCtrls.TOnGetMonthInfoEvent Type 5500
ComCtrls.TOwnerDrawState Type 5500
ComCtrls.TPageScrollEvent Type 5501
ComCtrls.TProgressRange Type 5502
ComCtrls.TRichEditProtectChange Type 5502
ComCtrls.TRichEditResizeEvent Type 5502
ComCtrls.TRichEditSaveClipboard Type 5503
ComCtrls.TSBCreatePanelClassEvent Type 5503
ComCtrls.TSearchTypes Type 5503
ComCtrls.TSectionDragEvent Type 5505
ComCtrls.TSectionNotifyEvent Type 5506
ComCtrls.TSectionTrackEvent Type 5506
ComCtrls.TStatusPanelClass Type 5507
ComCtrls.TTabChangingEvent Type 5507

ComCtrls.TTabGetImageEvent Type 5507
ComCtrls.TTBAdvancedCustomDrawBtnEvent Type 5508
ComCtrls.TTBAdvancedCustomDrawEvent Type 5508
ComCtrls.TTBButtonEvent Type 5508
ComCtrls.TTBCustomDrawBtnEvent Type 5509
ComCtrls.TTBCustomDrawEvent Type 5509
ComCtrls.TTBCustomDrawFlags Type 5509
ComCtrls.TTBCustomizeQueryEvent Type 5510
ComCtrls.TTBGradientDrawingOptions Type 5511
ComCtrls.TTBNewButtonEvent Type 5511
ComCtrls.TToolButtonActionLinkClass Type 5511
ComCtrls.TTreeNodeClass Type 5512
ComCtrls.TTVAdvancedCustomDrawEvent Type 5512
ComCtrls.TTVAdvancedCustomDrawItemEvent Type 5513
ComCtrls.TTVChangedEvent Type 5513
ComCtrls.TTVChangingEvent Type 5514
ComCtrls.TTVCollapsingEvent Type 5514
ComCtrls.TTVCompareEvent Type 5515
ComCtrls.TTVCreateNodeClassEvent Type 5516
ComCtrls.TTVCustomDrawEvent Type 5516
ComCtrls.TTVCustomDrawItemEvent Type 5516
ComCtrls.TTVEditedEvent Type 5517
ComCtrls.TTVEditingEvent Type 5518
ComCtrls.TTVExpandedEvent Type 5518
ComCtrls.TTVExpandingEvent Type 5519
ComCtrls.TUDChangingEvent Type 5519
ComCtrls.TUDChangingEventEx Type 5520
ComCtrls.TUDClickEvent Type 5520
ComCtrls.TWidth Type 5520

Constants **5521**

ComCtrls.CN_BANDCHANGE Constant 5521
ComCtrls.CN_DROPDOWNCLOSED Constant 5521
ComCtrls.ColumnHeaderWidth Constant 5522
ComCtrls.ColumnTextWidth Constant 5522
ComCtrls.ComCtlVersionIE3 Constant 5522
ComCtrls.ComCtlVersionIE4 Constant 5523
ComCtrls.ComCtlVersionIE401 Constant 5523
ComCtrls.ComCtlVersionIE5 Constant 5523
ComCtrls.ComCtlVersionIE501 Constant 5524

ComCtrls.ComCtlVersionIE6 Constant 5524

ComObj Namespace 5525

Classes 5528

EOleError Class 5528
EOleException Class 5529
EOleException Members 5529
EOleException.Create Constructor 5530
EOleException Properties 5531
EOleRegistrationError Class 5531
EOleSysError Class 5532
EOleSysError Members 5532
EOleSysError.Create Constructor 5533
EOleSysError Properties 5534
TAutoIntfObject Class 5534
TAutoIntfObject Members 5535
TAutoIntfObject.Create Constructor 5536
TAutoIntfObject Methods 5537
TAutoIntfObject Properties 5539
TAutoObject Class 5540
TAutoObject Members 5541
TAutoObject Methods 5542
TAutoObject Properties 5545
TAutoObjectFactory Class 5545
TAutoObjectFactory Members 5546
TAutoObjectFactory.Create Constructor 5548
TAutoObjectFactory Methods 5548
TAutoObjectFactory Properties 5548
TComClassManager Class 5550
TComClassManager Members 5550
TComClassManager.Create Constructor 5551
TComClassManager.Destroy Destructor 5552
TComClassManager Methods 5552
TComObject Class 5553
TComObject Members 5554
TComObject.Create Constructor 5555
TComObject.CreateAggregated Constructor 5556
TComObject.CreateFromFactory Constructor 5556

TComObject.Destroy Destructor 5557
TComObject Methods 5558
TComObject Properties 5560
TComObjectFactory Class 5563
TComObjectFactory Members 5564
TComObjectFactory.Create Constructor 5565
TComObjectFactory.Destroy Destructor 5566
TComObjectFactory Methods 5566
TComObjectFactory Properties 5571
TComServerObject Class 5575
TComServerObject Members 5576
TComServerObject Methods 5577
TComServerObject Properties 5579
TTypedComObject Class 5581
TTypedComObject Members 5582
TTypedComObject Methods 5583
TTypedComObjectFactory Class 5583
TTypedComObjectFactory Members 5584
TTypedComObjectFactory.Create Constructor 5585
TTypedComObjectFactory Methods 5586
TTypedComObjectFactory Properties 5587

Interfaces 5588

IServerExceptionHandler Interface 5588
IServerExceptionHandler Members 5588
IServerExceptionHandler Methods 5588

Functions 5590

ComObj.ClassIDToProgID Function 5590
ComObj.ComClassManager Function 5591
ComObj.CreateClassID Function 5591
ComObj.CreateComObject Function 5592
ComObj.CreateOleObject Function 5592
ComObj.CreateRegKey Function 5593
ComObj.CreateRemoteComObject Function 5594
ComObj.DeleteRegKey Function 5595
ComObj.DispatchInvoke Function 5595
ComObj.DispatchInvokeError Function 5596
ComObj.GetActiveOleObject Function 5596
ComObj.GetDispatchPropValue Function 5596

ComObj.GetRegStringValue Function 5597
ComObj.GUIDToString Function 5597
ComObj.HandleSafeCallException Function 5598
ComObj.InterfaceConnect Function 5598
ComObj.InterfaceDisconnect Function 5599
ComObj.OleCheck Function 5600
ComObj.OleError Function 5600
ComObj.ProgIDToClassID Function 5601
ComObj.RegisterAsService Function 5601
ComObj.RegisterComServer Function 5602
ComObj.SetDispatchPropValue Function 5602
ComObj.StringToGUID Function 5603
ComObj.StringToLPOLESTR Function 5603

Structs, Records, Enums **5604**
ComObj.TClassInstancing Enumeration 5604
ComObj.TThreadingModel Enumeration 5604

Types **5606**
ComObj.TAutoClass Type 5606
ComObj.TCoAddRefServerProcessProc Type 5606
ComObj.TCoCreateInstanceExProc Type 5607
ComObj.TCoInitializeExProc Type 5607
ComObj.TComClass Type 5607
ComObj.TConnectEvent Type 5608
ComObj.TCoReleaseServerProcessProc Type 5608
ComObj.TCoResumeClassObjectsProc Type 5608
ComObj.TCoSuspendClassObjectsProc Type 5609
ComObj.TFactoryProc Type 5609
ComObj.TTypedComClass Type 5609

Variables **5610**
ComObj.CoAddRefServerProcess Variable 5610
ComObj.CoCreateInstanceEx Variable 5610
ComObj.CoInitFlags Variable 5611
ComObj.CoInitializeEx Variable 5611
ComObj.CoReleaseServerProcess Variable 5612
ComObj.CoResumeClassObjects Variable 5612
ComObj.CoSuspendClassObjects Variable 5612

CompProd Namespace 5613

Classes **5614**
TComponentsPageProducer Class 5614
TComponentsPageProducer Members 5615
TComponentsPageProducer Methods 5618
TComponentsPageProducer Events 5620

ComServ Namespace 5621

Classes **5623**
TComServer Class 5623
TComServer Members 5624
TComServer.Create Constructor 5625
TComServer.Destroy Destructor 5626
TComServer Methods 5626
TComServer Properties 5630
TComServer Events 5632

Functions **5634**
ComServ.DllCanUnloadNow Function 5634
ComServ.DllGetClassObject Function 5635
ComServ.DllRegisterServer Function 5636
ComServ.DllUnregisterServer Function 5636
ComServ.LoadTypeLibrary Function 5637

Structs, Records, Enums **5638**
ComServ.TStartMode Enumeration 5638

Types **5639**
ComServ.TLastReleaseEvent Type 5639

Variables **5640**
ComServ.ComServer Variable 5640

Contnrs Namespace 5641

Classes **5643**
TBucketList Class 5643
TBucketList Members 5644
TBucketList.Create Constructor 5645
TBucketList Methods 5645
TClassList Class 5646
TClassList Members 5646

TClassList Methods 5648
TClassList Properties 5652
TComponentList Class 5653
TComponentList Members 5653
TComponentList.Destroy Destructor 5655
TComponentList Methods 5655
TComponentList Properties 5660
TCustomBucketList Class 5661
TCustomBucketList Members 5662
TCustomBucketList.Destroy Destructor 5663
TCustomBucketList Methods 5663
TCustomBucketList Properties 5668
TIntegerBucketList Class 5670
TIntegerBucketList Members 5670
TIntegerBucketList Methods 5671
TIntegerBucketList Properties 5672
TObjectBucketList Class 5673
TObjectBucketList Members 5674
TObjectBucketList Methods 5675
TObjectBucketList Properties 5676
TObjectList Class 5677
TObjectList Members 5678
Create Constructor 5679
TObjectList Methods 5680
TObjectList Properties 5685
TObjectQueue Class 5687
TObjectQueue Members 5687
TObjectQueue Methods 5688
TObjectStack Class 5690
TObjectStack Members 5690
TObjectStack Methods 5692
TOrderedList Class 5693
TOrderedList Members 5694
TOrderedList.Create Constructor 5695
TOrderedList.Destroy Destructor 5695
TOrderedList Methods 5695
TOrderedList Properties 5699
TQueue Class 5699
TQueue Members 5700

TQueue Methods 5701
TStack Class 5701
TStack Members 5702
TStack Methods 5703

Functions 5705
Contnrs.RaiseListError Function 5705

Structs, Records, Enums 5706
Contnrs.TBucket Record 5706
Contnrs.TBucketItem Record 5707
Contnrs.TBucketListSizes Enumeration 5707

Types 5709
Contnrs.TBucketArray Type 5709
Contnrs.TBucketEvent Type 5709
Contnrs.TBucketItemArray Type 5710
Contnrs.TBucketProc Type 5710

Controls Namespace 5711

Classes 5719
TBaseDragControlObject Class 5719
TBaseDragControlObject Members 5720
TBaseDragControlObject.Create Constructor 5721
TBaseDragControlObject Methods 5722
TBaseDragControlObject Properties 5723
TControl Class 5724
TControl Members 5725
TControl.Create Constructor 5731
TControl.Destroy Destructor 5732
TControl Fields 5733
TControl Methods 5735
TControl Properties 5796
TControl Events 5850
TControlAction Class 5877
TControlAction Members 5877
TControlAction Properties 5881
TControlAction Events 5888
TControlActionLink Class 5892
TControlActionLink Members 5892

TControlActionLink Fields 5894
TControlActionLink Methods 5895
TControlCanvas Class 5903
TControlCanvas Members 5903
TControlCanvas.Destroy Destructor 5906
TControlCanvas Methods 5906
TControlCanvas Properties 5907
TCustomControl Class 5908
TCustomControl Members 5908
TCustomControl.Create Constructor 5918
TCustomControl.Destroy Destructor 5919
TCustomControl Methods 5920
TCustomControl Properties 5921
TCustomControlAction Class 5923
TCustomControlAction Members 5923
TCustomControlAction Properties 5927
TCustomListControl Class 5928
TCustomListControl Members 5929
TCustomListControl Methods 5939
TCustomListControl Properties 5944
TCustomMultiSelectListControl Class 5944
TCustomMultiSelectListControl Members 5945
TCustomMultiSelectListControl Fields 5955
TCustomMultiSelectListControl Methods 5956
TCustomMultiSelectListControl Properties 5957
TCustomPanningWindow Class 5958
TCustomPanningWindow Members 5958
TCustomPanningWindow Methods 5969
TCustomTransparentControl Class 5970
TCustomTransparentControl Members 5970
TCustomTransparentControl.Create Constructor 5980
TCustomTransparentControl Methods 5981
TCustomTransparentControl Properties 5983
TDockTree Class 5983
TDockTree Members 5984
TDockTree.Create Constructor 5985
TDockTree.Destroy Destructor 5986
TDockTree Methods 5987
TDockTree Properties 5996

TDockZone Class 5997
TDockZone Members 5998
TDockZone.Create Constructor 5998
TDockZone Methods 5998
TDockZone Properties 6000
TDragControlObject Class 6003
TDragControlObject Members 6004
TDragControlObject Methods 6005
TDragControlObjectEx Class 6007
TDragControlObjectEx Members 6007
TDragControlObjectEx Methods 6009
TDragDockObject Class 6009
TDragDockObject Members 6010
TDragDockObject.Create Constructor 6011
TDragDockObject.Destroy Destructor 6012
TDragDockObject Methods 6012
TDragDockObject Properties 6015
TDragDockObjectEx Class 6018
TDragDockObjectEx Members 6019
TDragDockObjectEx Methods 6021
TDragImageList Class 6021
TDragImageList Members 6022
TDragImageList Methods 6026
TDragImageList Properties 6029
TDragObject Class 6031
TDragObject Members 6032
TDragObject Methods 6033
TDragObject Properties 6038
TDragObjectEx Class 6041
TDragObjectEx Members 6042
TDragObjectEx Methods 6043
TGraphicControl Class 6044
TGraphicControl Members 6044
TGraphicControl.Create Constructor 6051
TGraphicControl.Destroy Destructor 6052
TGraphicControl Methods 6052
TGraphicControl Properties 6053
THintWindow Class 6055
THintWindow Members 6056

THintWindow.Create Constructor 6066
THintWindow Methods 6066
THintWindow Properties 6071
TImageList Class 6078
TImageList Members 6079
TImageList Properties 6083
TImageList Events 6087
TMargins Class 6087
TMargins Members 6088
TMargins.Create Constructor 6089
TMargins Methods 6090
TMargins Properties 6092
TMargins Events 6096
TMouse Class 6096
TMouse Members 6097
TMouse.Create Constructor 6097
TMouse.Destroy Destructor 6098
TMouse Methods 6098
TMouse Properties 6099
TPadding Class 6103
TPadding Members 6104
TPadding Methods 6105
TPadding Properties 6106
TSizeConstraints Class 6107
TSizeConstraints Members 6107
TSizeConstraints.Create Constructor 6109
TSizeConstraints Methods 6109
TSizeConstraints Properties 6110
TSizeConstraints Events 6111
TWinControl Class 6112
TWinControl Members 6113
TWinControl.Create Constructor 6123
TWinControl.CreateParented Constructor 6123
TWinControl.Destroy Destructor 6124
TWinControl Fields 6124
TWinControl Methods 6125
TWinControl Properties 6191
TWinControl Events 6213
TWinControlActionLink Class 6233

TWinControlActionLink Members 6233
TWinControlActionLink Fields 6235
TWinControlActionLink Methods 6236

Interfaces 6238
IDockManager Interface 6238
IDockManager Members 6239
IDockManager Methods 6239

Functions 6246
Controls.CancelDrag Function 6247
Controls.ChangeBiDiModeAlignment Function 6247
Controls.CursorToIdent Function 6247
Controls.CursorToString Function 6248
Controls.DragDone Function 6249
Controls.FindControl Function 6249
Controls.FindDragTarget Function 6249
Controls.FindVCLWindow Function 6250
Controls.GetCaptureControl Function 6251
Controls.GetCursorValues Function 6251
Controls.GetLongHint Function 6252
Controls.GetShortHint Function 6253
Controls.IdentToCursor Function 6254
Controls.Imm32GetCompositionString Function 6254
Controls.Imm32GetContext Function 6255
Controls.Imm32GetConversionStatus Function 6255
Controls.Imm32IsIME Function 6255
Controls.Imm32NotifyIME Function 6256
Controls.Imm32ReleaseContext Function 6256
Controls.Imm32SetCompositionFont Function 6256
Controls.Imm32SetCompositionWindow Function 6257
Controls.Imm32SetConversionStatus Function 6257
Controls.Imm32SetOpenStatus Function 6257
Controls.InitWndProc Function 6258
Controls.IsAbortResult Function 6258
Controls.IsAnAllResult Function 6259
Controls.IsDragObject Function 6259
Controls.IsNegativeResult Function 6260
Controls.IsPositiveResult Function 6260
Controls.IsVCLControl Function 6261

Controls.MoveWindowOrg Function 6262
Controls.PerformEraseBackground Function 6262
Controls.SendAppMessage Function 6262
Controls.SetCaptureControl Function 6263
Controls.SetImeMode Function 6264
Controls.SetImeName Function 6264
Controls.StringToCursor Function 6264
Controls.StripAllFromResult Function 6265
Controls.Win32NLSEnableIME Function 6266

Structs, Records, Enums 6267

Controls.TAlign Enumeration 6268
Controls.TAlignInfo Record 6269
Controls.TAnchorKind Enumeration 6269
Controls.TBevelCut Enumeration 6270
Controls.TBevelEdge Enumeration 6270
Controls.TBevelKind Enumeration 6271
Controls.TCMCancelMode Record 6272
Controls.TCMChanged Record 6272
Controls.TCMChildKey Record 6273
Controls.TCMControlChange Record 6273
Controls.TCMControlListChange Record 6274
Controls.TCMControlListChanging Record 6274
Controls.TCMCreatePopup Record 6275
Controls.TCMDockClient Record 6275
Controls.TCMDockNotification Record 6276
Controls.TCMDrag Record 6276
Controls.TCMFloat Record 6277
Controls.TCMFocusChanged Record 6277
Controls.TCMMouseActivate Record 6278
Controls.TCMMouseWheel Record 6278
Controls.TCMPopupHWndDestroy Record 6279
Controls.TCMUnDockClient Record 6279
Controls.TControlListItem Record 6280
Controls.TCreateParams Record 6280
Controls.TDockNotifyRec Record 6281
Controls.TDockOrientation Enumeration 6282
Controls.TDragKind Enumeration 6282
Controls.TDragMessage Enumeration 6283
Controls.TDragMode Enumeration 6283

Controls.TDragRec Record 6284
Controls.TDragState Enumeration 6284
Controls.TImeMode Enumeration 6285
Controls.TMouseActivate Enumeration 6287
Controls.TMouseActivateRec Record 6288
Controls.TMouseButton Enumeration 6288
Controls.TPopupFormInfo Record 6289

Types **6290**
Controls.PControlListItem Type 6291
Controls.PDockNotifyRec Type 6291
Controls.PDragRec Type 6292
Controls.PMouseActivateRec Type 6292
Controls.PPopupFormInfo Type 6292
Controls.TAlignInsertBeforeEvent Type 6293
Controls.TAlignPositionEvent Type 6293
Controls.TAlignSet Type 6294
Controls.TAnchors Type 6294
Controls.TAnimateWindowProc Type 6295
Controls.TBevelEdges Type 6295
Controls.TBevelWidth Type 6296
Controls.TBorderWidth Type 6296
Controls.TCanResizeEvent Type 6296
Controls.TCaption Type 6297
Controls.TCMActivate Type 6297
Controls.TCMDeactivate Type 6297
Controls.TCMDesignHitTest Type 6298
Controls.TCMDialogChar Type 6298
Controls.TCMDialogKey Type 6298
Controls.TCMEnter Type 6299
Controls.TCMExit Type 6299
Controls.TCMGotFocus Type 6299
Controls.TCMHitTest Type 6300
Controls.TCMLostFocus Type 6300
Controls.TCMWantSpecialKey Type 6300
Controls.TConstrainedResizeEvent Type 6301
Controls.TConstraintSize Type 6301
Controls.TContextPopupEvent Type 6301
Controls.TControlActionLinkClass Type 6302
Controls.TControlClass Type 6302

Controls.TControlState Type 6302
Controls.TControlStyle Type 6303
Controls.TCursor Type 6305
Controls.TDate Type 6306
Controls.TDockDropEvent Type 6307
Controls.TDockOverEvent Type 6307
Controls.TDockTreeClass Type 6308
Controls.TDragDropEvent Type 6308
Controls.TDragObjectClass Type 6308
Controls.TDragOverEvent Type 6309
Controls.TEndDragEvent Type 6309
Controls.TForEachZoneProc Type 6309
Controls.TGetSiteInfoEvent Type 6310
Controls.THintWindowClass Type 6310
Controls.TImeName Type 6311
Controls.TKeyEvent Type 6311
Controls.TKeyPressEvent Type 6311
Controls.TMarginSize Type 6312
Controls.TModalResult Type 6312
Controls.TMouseActivateEvent Type 6313
Controls.TMouseEvent Type 6313
Controls.TMouseMoveEvent Type 6314
Controls.TMouseWheelEvent Type 6314
Controls.TMouseWheelUpDownEvent Type 6314
Controls.TPanningWindowClass Type 6315
Controls.TScalingFlags Type 6315
Controls.TStartDockEvent Type 6316
Controls.TStartDragEvent Type 6316
Controls.TTabOrder Type 6317
Controls.TTime Type 6317
Controls.TUnDockEvent Type 6317
Controls.TWinControlActionLinkClass Type 6318
Controls.TWinControlClass Type 6318
Controls.TWndMethod Type 6319

Variables **6320**

Controls.AnimateWindowProc Variable 6320
Controls.CreationControl Variable 6320
Controls.DefaultDockTreeClass Variable 6321
Controls.IsVCLControlHook Variable 6321

Controls.Mouse Variable 6321
Controls.NewStyleControls Variable 6322

Constants 6323

Controls.AnchorAlign Constant 6325
Controls.CM_ACTIONEXECUTE Constant 6326
Controls.CM_ACTIONUPDATE Constant 6327
Controls.CM_ACTIVATE Constant 6327
Controls.CM_ALLCHILDRENFLIPPED Constant 6327
Controls.CM_APPKEYDOWN Constant 6328
Controls.CM_APPSYSCOMMAND Constant 6328
Controls.CM_BASE Constant 6328
Controls.CM_BIDIMODECHANGED Constant 6329
Controls.CM_BORDERCHANGED Constant 6329
Controls.CM_BUTTONPRESSED Constant 6329
Controls.CM_CANCELMODE Constant 6330
Controls.CM_CHANGED Constant 6330
Controls.CM_CHILDKEY Constant 6330
Controls.CM_COLORCHANGED Constant 6331
Controls.CM_CONTROLCHANGE Constant 6331
Controls.CM_CONTROLLISTCHANGE Constant 6331
Controls.CM_CONTROLLISTCHANGING Constant 6332
Controls.CM_CREATEPOPUP Constant 6332
Controls.CM_CTL3DCHANGED Constant 6332
Controls.CM_CTLCOLOREDITGLASS Constant 6333
Controls.CM_CURSORCHANGED Constant 6333
Controls.CM_DEACTIVATE Constant 6333
Controls.CM_DESIGNHITTEST Constant 6334
Controls.CM_DESTROYHANDLE Constant 6334
Controls.CM_DIALOGCHAR Constant 6334
Controls.CM_DIALOGHANDLE Constant 6335
Controls.CM_DIALOGKEY Constant 6335
Controls.CM_DOCKCLIENT Constant 6335
Controls.CM_DOCKNOTIFICATION Constant 6336
Controls.CM_DOCWINDOWACTIVATE Constant 6336
Controls.CM_DRAG Constant 6336
Controls.CM_ENABLEDCHANGED Constant 6337
Controls.CM_ENTER Constant 6337
Controls.CM_EXIT Constant 6337
Controls.CM_FLOAT Constant 6338

Controls.CM_FOCUSCHANGED Constant 6338
Controls.CM_FONTCHANGE Constant 6338
Controls.CM_FONTCHANGED Constant 6339
Controls.CM_GETDATALINK Constant 6339
Controls.CM_GOTFOCUS Constant 6339
Controls.CM_HINTSHOW Constant 6340
Controls.CM_HINTSHOWPAUSE Constant 6340
Controls.CM_HITTEST Constant 6340
Controls.CM_ICONCHANGED Constant 6341
Controls.CM_INVALIDATE Constant 6341
Controls.CM_INVALIDATEDOCKHOST Constant 6341
Controls.CM_INVOKEHELP Constant 6342
Controls.CM_ISSHORTCUT Constant 6342
Controls.CM_ISTOOLCONTROL Constant 6342
Controls.CM_LOSTFOCUS Constant 6343
Controls.CM_MENUCHANGED Constant 6343
Controls.CM_MOUSEACTIVATE Constant 6343
Controls.CM_MOUSEENTER Constant 6344
Controls.CM_MOUSELEAVE Constant 6344
Controls.CM_MOUSEWHEEL Constant 6344
Controls.CM_PARENTBIDIMODECHANGED Constant 6345
Controls.CM_PARENTCOLORCHANGED Constant 6345
Controls.CM_PARENTCTL3DCHANGED Constant 6345
Controls.CM_PARENTFONTCHANGED Constant 6346
Controls.CM_PARENTSHOWHINTCHANGED Constant 6346
Controls.CM_POPUPHWNDDESTROY Constant 6346
Controls.CM_RECREATEWND Constant 6347
Controls.CM_RELEASE Constant 6347
Controls.CM_SETACTIVECONTROL Constant 6347
Controls.CM_SHOWHINTCHANGED Constant 6348
Controls.CM_SHOWINGCHANGED Constant 6348
Controls.CM_SYSCOLORCHANGE Constant 6348
Controls.CM_SYSFONTCHANGED Constant 6349
Controls.CM_TABSTOPCHANGED Constant 6349
Controls.CM_TEXTCHANGED Constant 6349
Controls.CM_TIMECHANGE Constant 6350
Controls.CM_UIACTIVATE Constant 6350
Controls.CM_UIDEACTIVATE Constant 6350
Controls.CM_UNDOCKCLIENT Constant 6351

Controls.CM_VISIBLECHANGED Constant 6351
Controls.CM_WANTSPECIALKEY Constant 6351
Controls.CM_WINDOWHOOK Constant 6352
Controls.CM_WININICHANGE Constant 6352
Controls.CN_BASE Constant 6352
Controls.CN_CHAR Constant 6353
Controls.CN_CHARTOITEM Constant 6353
Controls.CN_COMMAND Constant 6353
Controls.CN_COMPAREITEM Constant 6354
Controls.CN_CTLCOLORBTN Constant 6354
Controls.CN_CTLCOLORDLG Constant 6354
Controls.CN_CTLCOLOREDIT Constant 6355
Controls.CN_CTLCOLORLISTBOX Constant 6355
Controls.CN_CTLCOLORMSGBOX Constant 6355
Controls.CN_CTLCOLORSCROLLBAR Constant 6356
Controls.CN_CTLCOLORSTATIC Constant 6356
Controls.CN_DELETEITEM Constant 6356
Controls.CN_DRAWITEM Constant 6357
Controls.CN_HSCROLL Constant 6357
Controls.CN_KEYDOWN Constant 6357
Controls.CN_KEYUP Constant 6358
Controls.CN_MEASUREITEM Constant 6358
Controls.CN_NOTIFY Constant 6358
Controls.CN_PARENTNOTIFY Constant 6359
Controls.CN_SYSCHAR Constant 6359
Controls.CN_SYSKEYDOWN Constant 6359
Controls.CN_VKEYTOITEM Constant 6360
Controls.CN_VSCROLL Constant 6360
Controls.crAppStart Constant 6360
Controls.crArrow Constant 6361
Controls.crCross Constant 6361
Controls.crDefault Constant 6362
Controls.crDrag Constant 6362
Controls.crHandPoint Constant 6363
Controls.crHelp Constant 6363
Controls.crHourGlass Constant 6364
Controls.crHSplit Constant 6364
Controls.crIBeam Constant 6365
Controls.crMultiDrag Constant 6365

Controls.crNo Constant 6366
Controls.crNoDrop Constant 6366
Controls.crNone Constant 6367
Controls.crSize Constant 6367
Controls.crSizeAll Constant 6368
Controls.crSizeNESW Constant 6368
Controls.crSizeNS Constant 6369
Controls.crSizeNWSE Constant 6369
Controls.crSizeWE Constant 6370
Controls.crSQLWait Constant 6370
Controls.crUpArrow Constant 6371
Controls.crVSplit Constant 6371
Controls.CTL3D_ALL Constant 6372
Controls.mrAbort Constant 6372
Controls.mrAll Constant 6372
Controls.mrCancel Constant 6373
Controls.mrIgnore Constant 6373
Controls.mrNo Constant 6374
Controls.mrNone Constant 6374
Controls.mrNoToAll Constant 6375
Controls.mrOk Constant 6375
Controls.mrRetry Constant 6375
Controls.mrYes Constant 6376
Controls.mrYesToAll Constant 6376
Controls.NullDockSite Constant 6377

ConvUtils Namespace 6379

Classes 6381

EConversionError Class 6381
TConvTypeFactor Class 6382
TConvTypeFactor Members 6382
TConvTypeFactor.Create Constructor 6383
TConvTypeFactor Methods 6384
TConvTypeFactor Properties 6385
TConvTypeInfo Class 6386
TConvTypeInfo Members 6387
TConvTypeInfo.Create Constructor 6388
TConvTypeInfo Methods 6388

TConvTypeInfo Properties 6390
TConvTypeProcs Class 6391
TConvTypeProcs Members 6392
TConvTypeProcs.Create Constructor 6393
TConvTypeProcs Methods 6393

Functions 6395
ConvUtils.CompatibleConversionType Function 6396
ConvUtils.CompatibleConversionTypes Function 6396
ConvUtils.Convert Function 6397
ConvUtils.ConvertFrom Function 6398
ConvUtils.ConvertTo Function 6399
ConvUtils.ConvFamilyToDescription Function 6399
ConvUtils.ConvTypeToDescription Function 6400
ConvUtils.ConvTypeToFamily Function 6400
ConvUtils.ConvUnitAdd Function 6401
ConvUtils.ConvUnitCompareValue Function 6402
ConvUtils.ConvUnitDec Function 6402
ConvUtils.ConvUnitDiff Function 6403
ConvUtils.ConvUnitInc Function 6404
ConvUtils.ConvUnitSameValue Function 6404
ConvUtils.ConvUnitToStr Function 6405
ConvUtils.ConvUnitWithinNext Function 6406
ConvUtils.ConvUnitWithinPrevious Function 6406
ConvUtils.DescriptionToConvFamily Function 6407
ConvUtils.DescriptionToConvType Function 6408
ConvUtils.GetConvFamilies Function 6408
ConvUtils.GetConvTypes Function 6409
ConvUtils.RaiseConversionError Function 6409
ConvUtils.RaiseConversionRegError Function 6410
ConvUtils.RegisterConversionFamily Function 6410
ConvUtils.RegisterConversionType Function 6411
ConvUtils.StrToConvUnit Function 6412
ConvUtils.TryConvTypeToFamily Function 6413
ConvUtils.TryStrToConvUnit Function 6414
ConvUtils.UnregisterConversionFamily Function 6414
ConvUtils.UnregisterConversionType Function 6415

Types 6416
ConvUtils.TConversionProc Type 6416

ConvUtils.TConvFamily Type 6416
ConvUtils.TConvFamilyArray Type 6417
ConvUtils.TConvType Type 6417
ConvUtils.TConvTypeArray Type 6417
ConvUtils.TConvTypeList Type 6418

Variables **6419**
ConvUtils.GConvUnitToStrFmt Variable 6419

Constants **6420**
ConvUtils.CIllegalConvFamily Constant 6420
ConvUtils.CIllegalConvType Constant 6420

CorbaObj Namespace 6421

Classes **6423**
ECorbaDispatch Class 6423
ECorbaException Class 6423
ECorbaException Members 6424
ECorbaException Methods 6425
ECorbaException Properties 6425
ECorbaUserException Class 6426
ECorbaUserException Members 6426
ECorbaUserException.Create Constructor 6427
ECorbaUserException Methods 6430
ECorbaUserException Properties 6430
TBOA Class 6431
TBOA Members 6431
TBOA Methods 6431
TCorbaDispatchStub Class 6433
TCorbaDispatchStub Members 6433
TCorbaDispatchStub Methods 6435
TCorbaFactory Class 6436
TCorbaFactory Members 6436
TCorbaFactory.Create Constructor 6437
TCorbaFactory.Destroy Destructor 6438
TCorbaFactory Methods 6438
TCorbaFactory Properties 6440
TCorbaFactoryManager Class 6442
TCorbaFactoryManager Members 6442
TCorbaFactoryManager.Destroy Destructor 6442

TCorbaFactoryManager Methods 6443
TCorbaImplementation Class 6444
TCorbaImplementation Members 6444
TCorbaImplementation.Create Constructor 6445
TCorbaImplementation Fields 6445
TCorbaImplementation Methods 6446
TCorbaInterfaceIDManager Class 6449
TCorbaInterfaceIDManager Members 6449
TCorbaInterfaceIDManager Methods 6449
TCorbaListManager Class 6451
TCorbaListManager Members 6451
TCorbaListManager.Create Constructor 6451
TCorbaListManager.Destroy Destructor 6452
TCorbaListManager Methods 6452
TCorbaObjectFactory Class 6453
TCorbaObjectFactory Members 6453
TCorbaObjectFactory.Create Constructor 6455
TCorbaObjectFactory Methods 6455
TCorbaObjectFactory Properties 6456
TCorbaSkeleton Class 6456
TCorbaSkeleton Members 6456
TCorbaSkeleton.Create Constructor 6458
TCorbaSkeleton.Destroy Destructor 6458
TCorbaSkeleton Fields 6458
TCorbaSkeleton Methods 6459
TCorbaSkeletonManager Class 6460
TCorbaSkeletonManager Members 6460
TCorbaSkeletonManager Methods 6461
TCorbaStub Class 6462
TCorbaStub Members 6462
TCorbaStub.Create Constructor 6463
TCorbaStub.Destroy Destructor 6464
TCorbaStub Fields 6464
TCorbaStub Methods 6465
TCorbaStubManager Class 6466
TCorbaStubManager Members 6466
TCorbaStubManager Methods 6467
TORB Class 6467
TORB Members 6468

TORB Methods 6468

Interfaces 6475

ICorbaObject Interface 6475

ICorbaObject Members 6475

ICorbaObject Methods 6475

Functions 6477

CorbaObj.AnyToObject Function 6477

CorbaObj.BOA Function 6477

CorbaObj.CorbaBind Function 6478

CorbaObj.CorbaBind Function 6478

CorbaObj.CorbaFactoryCreateStub Function 6478

CorbaObj.CorbaInitialize Function 6479

CorbaObj.MakePrincipal Function 6479

CorbaObj.MarshalAny Function 6479

CorbaObj.MarshalObject Function 6480

CorbaObj.MarshalWordBool Function 6480

CorbaObj.ORB Function 6480

CorbaObj.SequenceToVariantArray Function 6481

CorbaObj.UnmarshalAny Function 6481

CorbaObj.UnmarshalObject Function 6481

CorbaObj.UnmarshalText Function 6482

CorbaObj.UnmarshalWideText Function 6482

CorbaObj.UnmarshalWordBool Function 6482

CorbaObj.VariantArrayToSequence Function 6483

Structs, Records, Enums 6484

CorbaObj.TCorbaInstancing Enumeration 6484

CorbaObj.TCorbaThreadModel Enumeration 6484

CorbaObj.TInterfaceIDEntryDesc Record 6485

CorbaObj.TSkeletonEntryDesc Record 6485

CorbaObj.TStubEntryDesc Record 6486

Types 6487

CorbaObj.CorbaBoolean Type 6487

CorbaObj.CorbaULong Type 6487

CorbaObj.IObject Type 6488

CorbaObj.ITypeCode Type 6488

CorbaObj.TAny Type 6488

CorbaObj.TCKind Type 6489

CorbaObj.TCommandLine Type 6489
CorbaObj.TCorbaImplementationClass Type 6489
CorbaObj.TCorbaPrincipal Type 6490
CorbaObj.TCorbaSkeletonClass Type 6490
CorbaObj.TCorbaStubClass Type 6490
CorbaObj.TFactoryList Type 6491
CorbaObj.TInterfaceIDList Type 6491
CorbaObj.TSkeletonList Type 6491
CorbaObj.TStubList Type 6492

Variables 6493
CorbaObj.CorbaInterfaceIDManager Variable 6493
CorbaObj.CorbaSkeletonManager Variable 6493
CorbaObj.CorbaStubManager Variable 6493

CtlPanel Namespace 6495

Classes 6497
EAppletException Class 6497
TAppletApplication Class 6497
TAppletApplication Members 6498
TAppletApplication.Create Constructor 6501
TAppletApplication.Destroy Destructor 6501
TAppletApplication Methods 6502
TAppletApplication Properties 6504
TAppletApplication Events 6506
TAppletModule Class 6507
TAppletModule Members 6508
TAppletModule.Create Constructor 6511
TAppletModule.Destroy Destructor 6511
TAppletModule Methods 6512
TAppletModule Properties 6513
TAppletModule Events 6516

Functions 6519
CtlPanel.CPlApplet Function 6519

Types 6520
CtlPanel.TActivateEvent Type 6520
CtlPanel.TAppletModuleClass Type 6520
CtlPanel.TCountEvent Type 6521

CtlPanel.TCPLAppletClass Type 6521
CtlPanel.TDataModuleClass Type 6522
CtlPanel.TExitEvent Type 6522
CtlPanel.TInitEvent Type 6522
CtlPanel.TInquireEvent Type 6523
CtlPanel.TNewInquireEvent Type 6523
CtlPanel.TOnAppletExceptionEvent Type 6524
CtlPanel.TSetupEvent Type 6524
CtlPanel.TStartWParmsEvent Type 6524
CtlPanel.TStopEvent Type 6525

Variables 6526

CtlPanel.Application Variable 6526

CustomizeDlg Namespace 6527

Classes 6528

TCustomizeDlg Class 6528
TCustomizeDlg Members 6528
TCustomizeDlg Methods 6531
TCustomizeDlg Properties 6532
TCustomizeDlg Events 6534
TCustomizeFrm Class 6535
TCustomizeFrm Members 6535
TCustomizeFrm Fields 6553
TCustomizeFrm Methods 6564
TCustomizeFrm Properties 6573

DataBkr Namespace 6575

Classes 6576

TCRemoteDataModule Class 6576
TCRemoteDataModule Members 6577
TCRemoteDataModule.Create Constructor 6580
TCRemoteDataModule.Destroy Destructor 6580
TCRemoteDataModule Methods 6580
TCRemoteDataModule Properties 6584
TRemoteDataModule Class 6585
TRemoteDataModule Members 6585
TRemoteDataModule.Create Constructor 6589

TRemoteDataModule.Destroy Destructor 6589
TRemoteDataModule Methods 6589
TRemoteDataModule Properties 6598

Functions 6600
DataBkr.DisableSocketTransport Function 6600
DataBkr.DisableWebTransport Function 6601
DataBkr.EnableSocketTransport Function 6601
DataBkr.EnableWebTransport Function 6602
DataBkr.RegisterPooled Function 6602
DataBkr.UnregisterPooled Function 6603

DateUtils Namespace 6605

Functions 6610
DateUtils.CompareDate Function 6614
DateUtils.CompareDateTime Function 6614
DateUtils.CompareTime Function 6615
DateUtils.DateOf Function 6616
DateUtils.DateTimeToJulianDate Function 6616
DateUtils.DateTimeToModifiedJulianDate Function 6617
DateUtils.DateTimeToUnix Function 6617
DateUtils.DayOf Function 6618
DateUtils.DayOfTheMonth Function 6618
DateUtils.DayOfTheWeek Function 6619
DateUtils.DayOfTheYear Function 6620
DateUtils.DaysBetween Function 6621
DateUtils.DaysInAMonth Function 6622
DateUtils.DaysInAYear Function 6622
DateUtils.DaysInMonth Function 6623
DateUtils.DaysInYear Function 6623
DateUtils.DaySpan Function 6624
DateUtils.DecodeDateDay Function 6624
DateUtils.DecodeDateMonthWeek Function 6625
DateUtils.DecodeDateTime Function 6626
DateUtils.DecodeDateWeek Function 6627
DateUtils.DecodeDayOfWeekInMonth Function 6628
DateUtils.EncodeDateDay Function 6629
DateUtils.EncodeDateMonthWeek Function 6629
DateUtils.EncodeDateTime Function 6630

DateUtils.EncodeDateWeek Function 6631
DateUtils.EncodeDayOfWeekInMonth Function 6632
DateUtils.EndOfADay Function 6633
DateUtils.EndOfAMonth Function 6634
DateUtils.EndOfAWeek Function 6635
DateUtils.EndOfAYear Function 6636
DateUtils.EndOfTheDay Function 6636
DateUtils.EndOfTheMonth Function 6637
DateUtils.EndOfTheWeek Function 6637
DateUtils.EndOfTheYear Function 6638
DateUtils.HourOf Function 6638
DateUtils.HourOfTheDay Function 6639
DateUtils.HourOfTheMonth Function 6640
DateUtils.HourOfTheWeek Function 6641
DateUtils.HourOfTheYear Function 6641
DateUtils.HoursBetween Function 6642
DateUtils.HourSpan Function 6643
DateUtils.IncDay Function 6643
DateUtils.IncHour Function 6644
DateUtils.IncMilliSecond Function 6645
DateUtils.IncMinute Function 6645
DateUtils.IncSecond Function 6646
DateUtils.IncWeek Function 6647
DateUtils.IncYear Function 6647
DateUtils.InvalidDateDayError Function 6648
DateUtils.InvalidDateMonthWeekError Function 6648
DateUtils.InvalidDateTimeError Function 6649
DateUtils.InvalidDateWeekError Function 6649
DateUtils.InvalidDayOfWeekInMonthError Function 6650
DateUtils.IsInLeapYear Function 6650
DateUtils.IsPM Function 6650
DateUtils.IsSameDay Function 6651
DateUtils.IsToday Function 6651
DateUtils.IsValidDate Function 6652
DateUtils.IsValidDateDay Function 6652
DateUtils.IsValidDateMonthWeek Function 6653
DateUtils.IsValidDateTime Function 6654
DateUtils.IsValidDateWeek Function 6654
DateUtils.IsValidTime Function 6655

DateUtils.JulianDateToDateTime Function 6656
DateUtils.MilliSecondOf Function 6656
DateUtils.MilliSecondOfTheDay Function 6657
DateUtils.MilliSecondOfTheHour Function 6658
DateUtils.MilliSecondOfTheMinute Function 6659
DateUtils.MilliSecondOfTheMonth Function 6659
DateUtils.MilliSecondOfTheSecond Function 6660
DateUtils.MilliSecondOfTheWeek Function 6661
DateUtils.MilliSecondOfTheYear Function 6661
DateUtils.MilliSecondsBetween Function 6662
DateUtils.MilliSecondSpan Function 6663
DateUtils.MinuteOf Function 6663
DateUtils.MinuteOfTheDay Function 6664
DateUtils.MinuteOfTheHour Function 6665
DateUtils.MinuteOfTheMonth Function 6666
DateUtils.MinuteOfTheWeek Function 6666
DateUtils.MinuteOfTheYear Function 6667
DateUtils.MinutesBetween Function 6668
DateUtils.MinuteSpan Function 6668
DateUtils.ModifiedJulianDateToDateTime Function 6669
DateUtils.MonthOf Function 6670
DateUtils.MonthOfTheYear Function 6670
DateUtils.MonthsBetween Function 6671
DateUtils.MonthSpan Function 6672
DateUtils.NthDayOfWeek Function 6672
DateUtils.RecodeDate Function 6673
DateUtils.RecodeDateTime Function 6674
DateUtils.RecodeDay Function 6675
DateUtils.RecodeHour Function 6675
DateUtils.RecodeMilliSecond Function 6676
DateUtils.RecodeMinute Function 6677
DateUtils.RecodeMonth Function 6678
DateUtils.RecodeSecond Function 6678
DateUtils.RecodeTime Function 6679
DateUtils.RecodeYear Function 6680
DateUtils.SameDate Function 6681
DateUtils.SameDateTime Function 6681
DateUtils.SameTime Function 6682
DateUtils.SecondOf Function 6682

DateUtils.SecondOfTheDay Function 6683
DateUtils.SecondOfTheHour Function 6684
DateUtils.SecondOfTheMinute Function 6685
DateUtils.SecondOfTheMonth Function 6685
DateUtils.SecondOfTheWeek Function 6686
DateUtils.SecondOfTheYear Function 6687
DateUtils.SecondsBetween Function 6687
DateUtils.SecondSpan Function 6688
DateUtils.StartOfADay Function 6689
DateUtils.StartOfAMonth Function 6690
DateUtils.StartOfAWeek Function 6690
DateUtils.StartOfAYear Function 6691
DateUtils.StartOfTheDay Function 6692
DateUtils.StartOfTheMonth Function 6692
DateUtils.StartOfTheWeek Function 6693
DateUtils.StartOfTheYear Function 6693
DateUtils.TimeOf Function 6694
DateUtils.Today Function 6694
DateUtils.Tomorrow Function 6695
DateUtils.TryEncodeDateDay Function 6695
DateUtils.TryEncodeDateMonthWeek Function 6696
DateUtils.TryEncodeDateTime Function 6697
DateUtils.TryEncodeDateWeek Function 6698
DateUtils.TryEncodeDayOfWeekInMonth Function 6699
DateUtils.TryJulianDateToDateTime Function 6700
DateUtils.TryModifiedJulianDateToDateTime Function 6701
DateUtils.TryRecodeDateTime Function 6702
DateUtils.UnixToDateTime Function 6703
DateUtils.WeekOf Function 6703
DateUtils.WeekOfTheMonth Function 6704
DateUtils.WeekOfTheYear Function 6705
DateUtils.WeeksBetween Function 6706
DateUtils.WeeksInAYear Function 6706
DateUtils.WeeksInYear Function 6707
DateUtils.WeekSpan Function 6708
DateUtils.WithinPastDays Function 6708
DateUtils.WithinPastHours Function 6709
DateUtils.WithinPastMilliSeconds Function 6710
DateUtils.WithinPastMinutes Function 6710

DateUtils.WithinPastMonths Function 6711
DateUtils.WithinPastSeconds Function 6712
DateUtils.WithinPastWeeks Function 6712
DateUtils.WithinPastYears Function 6713
DateUtils.YearOf Function 6714
DateUtils.YearsBetween Function 6714
DateUtils.YearSpan Function 6715
DateUtils.Yesterday Function 6716

Variables 6717
DateUtils.ApproxDaysPerMonth Variable 6717
DateUtils.ApproxDaysPerYear Variable 6717

Constants 6718
DateUtils.DayFriday Constant 6718
DateUtils.DayMonday Constant 6718
DateUtils.DaySaturday Constant 6719
DateUtils.DaysPerWeek Constant 6719
DateUtils.DaysPerYear Constant 6719
DateUtils.DaySunday Constant 6720
DateUtils.DayThursday Constant 6720
DateUtils.DayTuesday Constant 6720
DateUtils.DayWednesday Constant 6721
DateUtils.MonthsPerYear Constant 6721
DateUtils.OneHour Constant 6721
DateUtils.OneMillisecond Constant 6722
DateUtils.OneMinute Constant 6722
DateUtils.OneSecond Constant 6722
DateUtils.RecodeLeaveFieldAsIs Constant 6723
DateUtils.WeeksPerFortnight Constant 6723
DateUtils.YearsPerCentury Constant 6723
DateUtils.YearsPerDecade Constant 6724
DateUtils.YearsPerMillennium Constant 6724

DB Namespace 6725

Classes 6730
EDatabaseError Class 6731
EUpdateError Class 6732
EUpdateError Members 6732
EUpdateError.Create Constructor 6733

EUpdateError.Destroy Destructor 6734
EUpdateError Properties 6734
TADTField Class 6736
TADTField Members 6737
TADTField.Create Constructor 6742
TADTField Methods 6743
TAggregateField Class 6743
TAggregateField Members 6744
TAggregateField.Create Constructor 6749
TAggregateField Methods 6750
TAggregateField Properties 6751
TArrayField Class 6760
TArrayField Members 6761
TArrayField.Create Constructor 6766
TArrayField Methods 6767
TArrayField Properties 6767
TAutoIncField Class 6768
TAutoIncField Members 6769
TAutoIncField.Create Constructor 6775
TBCDField Class 6776
TBCDField Members 6777
TBCDField.Create Constructor 6782
TBCDField Methods 6783
TBCDField Properties 6788
TBinaryField Class 6791
TBinaryField Members 6791
TBinaryField.Create Constructor 6797
TBinaryField Methods 6798
TBinaryField Properties 6800
TBlobField Class 6801
TBlobField Members 6802
TBlobField.Create Constructor 6808
TBlobField Methods 6809
TBlobField Properties 6818
TBooleanField Class 6822
TBooleanField Members 6823
TBooleanField.Create Constructor 6828
TBooleanField Methods 6829
TBooleanField Properties 6831

TBytesField Class 6833
TBytesField Members 6834
TBytesField.Create Constructor 6840
TBytesField Methods 6841
TCheckConstraint Class 6841
TCheckConstraint Members 6842
TCheckConstraint Methods 6843
TCheckConstraint Properties 6844
TCheckConstraints Class 6846
TCheckConstraints Members 6846
TCheckConstraints.Create Constructor 6848
TCheckConstraints Methods 6848
TCheckConstraints Properties 6849
TCurrencyField Class 6850
TCurrencyField Members 6850
TCurrencyField.Create Constructor 6856
TCurrencyField Properties 6857
TCustomConnection Class 6858
TCustomConnection Members 6859
TCustomConnection.Create Constructor 6862
TCustomConnection.Destroy Destructor 6863
TCustomConnection Methods 6863
TCustomConnection Properties 6868
TCustomConnection Events 6874
TDataLink Class 6876
TDataLink Members 6877
TDataLink.Create Constructor 6878
TDataLink.Destroy Destructor 6879
TDataLink Methods 6879
TDataLink Properties 6888
TDataSet Class 6894
TDataSet Members 6896
TDataSet.Create Constructor 6904
TDataSet.Destroy Destructor 6904
TDataSet Methods 6905
TDataSet Properties 7003
TDataSet Events 7037
TDataSetDesigner Class 7056
TDataSetDesigner Members 7057

TDataSetDesigner.Create Constructor 7058
TDataSetDesigner.Destroy Destructor 7058
TDataSetDesigner Methods 7059
TDataSetDesigner Properties 7060
TDataSetField Class 7060
TDataSetField Members 7061
TDataSetField.Create Constructor 7067
TDataSetField.Destroy Destructor 7067
TDataSetField Methods 7068
TDataSetField Properties 7069
TDataSource Class 7070
TDataSource Members 7072
TDataSource.Create Constructor 7075
TDataSource.Destroy Destructor 7075
TDataSource Methods 7076
TDataSource Properties 7077
TDataSource Events 7080
TDateField Class 7083
TDateField Members 7083
TDateField.Create Constructor 7089
TDateField Methods 7090
TDateTimeField Class 7090
TDateTimeField Members 7091
TDateTimeField.Create Constructor 7097
TDateTimeField Methods 7098
TDateTimeField Properties 7101
TDefCollection Class 7104
TDefCollection Members 7104
TDefCollection.Create Constructor 7106
TDefCollection Methods 7107
TDefCollection Properties 7110
TDefCollection Events 7111
TDetailDataLink Class 7111
TDetailDataLink Members 7112
TDetailDataLink Methods 7113
TDetailDataLink Properties 7114
TField Class 7115
TField Members 7116
TField.Create Constructor 7121

TField.Destroy Destructor 7122
TField Methods 7122
TField Properties 7145
TField Events 7181
TFieldDef Class 7184
TFieldDef Members 7185
TFieldDef.Create Constructor 7186
TFieldDef.Destroy Destructor 7187
TFieldDef Methods 7187
TFieldDef Properties 7190
TFieldDefList Class 7195
TFieldDefList Members 7195
TFieldDefList Methods 7196
TFieldDefList Properties 7197
TFieldDefs Class 7198
TFieldDefs Members 7199
TFieldDefs.Create Constructor 7201
TFieldDefs Methods 7201
TFieldDefs Properties 7206
TFieldList Class 7207
TFieldList Members 7208
TFieldList Methods 7209
TFieldList Properties 7210
TFields Class 7211
TFields Members 7211
TFields.Create Constructor 7213
TFields.Destroy Destructor 7213
TFields Methods 7213
TFields Properties 7220
TFields Events 7222
TFieldsEnumerator Class 7222
TFieldsEnumerator Members 7223
TFieldsEnumerator.Create Constructor 7223
TFieldsEnumerator Methods 7223
TFieldsEnumerator Properties 7224
TFlatList Class 7225
TFlatList Members 7225
TFlatList.Create Constructor 7226
TFlatList Methods 7226

TFlatList Properties 7228
TFloatField Class 7229
TFloatField Members 7230
TFloatField.Create Constructor 7236
TFloatField Methods 7237
TFloatField Properties 7239
TFMTBCDField Class 7242
TFMTBCDField Members 7243
TFMTBCDField.Create Constructor 7248
TFMTBCDField Methods 7249
TFMTBCDField Properties 7254
TGraphicField Class 7257
TGraphicField Members 7258
TGraphicField.Create Constructor 7264
TGuidField Class 7265
TGuidField Members 7266
TGuidField.Create Constructor 7272
TGuidField Methods 7273
TGuidField Properties 7274
TIDispatchField Class 7274
TIDispatchField Members 7275
TIDispatchField.Create Constructor 7280
TIDispatchField Methods 7281
TIDispatchField Properties 7282
TIndexDef Class 7283
TIndexDef Members 7283
TIndexDef.Create Constructor 7285
TIndexDef Methods 7285
TIndexDef Properties 7286
TIndexDefs Class 7292
TIndexDefs Members 7293
TIndexDefs.Create Constructor 7295
TIndexDefs Methods 7295
TIndexDefs Properties 7300
TIntegerField Class 7301
TIntegerField Members 7303
TIntegerField.Create Constructor 7309
TIntegerField Methods 7310
TIntegerField Properties 7313

TInterfaceField Class 7314
TInterfaceField Members 7315
TInterfaceField.Create Constructor 7320
TInterfaceField Methods 7321
TInterfaceField Properties 7322
TLargeintField Class 7323
TLargeintField Members 7324
TLargeintField.Create Constructor 7329
TLargeintField Methods 7330
TLargeintField Properties 7334
TLookupList Class 7336
TLookupList Members 7336
TLookupList.Create Constructor 7337
TLookupList.Destroy Destructor 7338
TLookupList Methods 7338
TMasterDataLink Class 7339
TMasterDataLink Members 7340
TMasterDataLink.Create Constructor 7342
TMasterDataLink.Destroy Destructor 7342
TMasterDataLink Methods 7343
TMasterDataLink Properties 7345
TMasterDataLink Events 7346
TMemoField Class 7348
TMemoField Members 7348
TMemoField.Create Constructor 7354
TMemoField Methods 7355
TMemoField Properties 7356
TNamedItem Class 7357
TNamedItem Members 7357
TNamedItem Methods 7359
TNamedItem Properties 7359
TNumericField Class 7361
TNumericField Members 7362
TNumericField.Create Constructor 7367
TNumericField Methods 7367
TNumericField Properties 7368
TObjectField Class 7372
TObjectField Members 7373
TObjectField.Create Constructor 7378

TObjectField.Destroy Destructor 7379
TObjectField Methods 7379
TObjectField Properties 7384
TParam Class 7386
TParam Members 7387
Create Constructor 7389
TParam Methods 7390
TParam Properties 7406
TParams Class 7434
TParams Members 7435
Create Constructor 7437
TParams Methods 7437
TParams Properties 7449
TReferenceField Class 7452
TReferenceField Members 7452
TReferenceField.Create Constructor 7458
TReferenceField Methods 7459
TReferenceField Properties 7460
TSmallintField Class 7461
TSmallintField Members 7462
TSmallintField.Create Constructor 7468
TSmallintField Methods 7469
TSQLTimeStampField Class 7469
TSQLTimeStampField Members 7470
TSQLTimeStampField.Create Constructor 7475
TSQLTimeStampField Methods 7476
TSQLTimeStampField Properties 7480
TStringField Class 7483
TStringField Members 7484
TStringField.Create Constructor 7490
TStringField Methods 7491
TStringField Properties 7496
TTimeField Class 7499
TTimeField Members 7500
TTimeField.Create Constructor 7505
TTimeField Methods 7506
TVarBytesField Class 7507
TVarBytesField Members 7507
TVarBytesField.Create Constructor 7513

TVarBytesField Methods 7513
TVariantField Class 7514
TVariantField Members 7515
TVariantField.Create Constructor 7520
TVariantField Methods 7521
TWideDataSet Class 7526
TWideDataSet Members 7526
TWideDataSet Methods 7535
TWideMemoField Class 7541
TWideMemoField Members 7541
TWideMemoField.Create Constructor 7547
TWideMemoField Methods 7548
TWideMemoField Properties 7549
TWideStringField Class 7549
TWideStringField Members 7550
TWideStringField.Create Constructor 7555
TWideStringField Methods 7557
TWideStringField Properties 7559
TWordField Class 7560
TWordField Members 7561
TWordField.Create Constructor 7566
TWordField Methods 7567

Interfaces 7569
IDBApplication Interface 7569
IDBApplication Members 7569
IDBApplication Methods 7570
IDBApplication Properties 7571
IDBScreen Interface 7571
IDBScreen Members 7572
IDBScreen Methods 7572
IDBScreen Properties 7573
IDBSession Interface 7573
IDBSession Members 7574
IDBSession Methods 7574
IProviderSupport Interface 7576
IProviderSupport Members 7576
IProviderSupport Methods 7577
IProviderSupport2 Interface 7587
IProviderSupport2 Members 7588

IProviderSupport2 Methods 7588

Functions 7599

DB.BuffersEqual Function 7599
DB.DatabaseError Function 7599
DB.DatabaseErrorFmt Function 7600
DB.DisposeMem Function 7600
DB.ExtractFieldName Function 7601
DB.GetFieldProperty Function 7602
DB.RegisterFields Function 7602
DB.VarTypeToDataType Function 7603

Structs, Records, Enums 7604

DB.TAutoRefreshFlag Enumeration 7604
DB.TBlobStreamMode Enumeration 7605
DB.TBookmarkFlag Enumeration 7606
DB.TDataAction Enumeration 7606
DB.TDataEvent Enumeration 7607
DB.TDataSetState Enumeration 7608
DB.TDateTimeRec Record 7610
DB.TDBScreenCursor Enumeration 7610
DB.TFieldAttribute Enumeration 7611
DB.TFieldKind Enumeration 7611
DB.TFieldType Enumeration 7612
DB.TFilterOption Enumeration 7615
DB.TGetMode Enumeration 7615
DB.TGetResult Enumeration 7616
DB.TGroupPosInd Enumeration 7616
DB.TIndexOption Enumeration 7617
DB.TLocateOption Enumeration 7618
DB.TLookupListEntry Record 7618
DB.TPacketAttribute Record 7618
DB.TParamType Enumeration 7619
DB.TProviderFlag Enumeration 7620
DB.TPSCommandType Enumeration 7621
DB.TUpdateAction Enumeration 7621
DB.TUpdateKind Enumeration 7622
DB.TUpdateMode Enumeration 7623
DB.TUpdateStatus Enumeration 7623

Types 7625

DB.Largeint Type 7626
DB.PBookmarkFlag Type 7626
DB.PLookupListEntry Type 7626
DB.PPacketAttribute Type 7627
DB.TBlobByteData Type 7627
DB.TBlobData Type 7627
DB.TBlobType Type 7628
DB.TBookmark Type 7628
DB.TBookmarkStr Type 7629
DB.TBufferList Type 7629
DB.TConnectChangeEvent Type 7630
DB.TDataChangeEvent Type 7630
DB.TDataOperation Type 7630
DB.TDataSetClass Type 7631
DB.TDataSetErrorEvent Type 7631
DB.TDataSetNotifyEvent Type 7631
DB.TDateTimeAlias Type 7632
DB.TDefUpdateMethod Type 7632
DB.TFieldAttributes Type 7632
DB.TFieldChars Type 7633
DB.TFieldClass Type 7633
DB.TFieldGetTextEvent Type 7633
DB.TFieldKinds Type 7634
DB.TFieldNotifyEvent Type 7634
DB.TFieldRef Type 7634
DB.TFieldSetTextEvent Type 7635
DB.TFilterOptions Type 7635
DB.TFilterRecordEvent Type 7635
DB.TGroupPosInds Type 7636
DB.TIndexOptions Type 7636
DB.TLocateOptions Type 7637
DB.TParamTypes Type 7637
DB.TProviderFlags Type 7638
DB.TResyncMode Type 7638
DB.TSQLCommandType Type 7638
DB.TUpdateErrorEvent Type 7639
DB.TUpdateRecordEvent Type 7639
DB.TUpdateRecordTypes Type 7639
DB.TUpdateStatusSet Type 7640

Variables **7641**
DB.DBApplication Variable 7641
DB.DBScreen Variable 7641
DB.DefaultFieldClasses Variable 7642
DB.LoginDialogExProc Variable 7642
DB.LoginDialogProc Variable 7642
DB.PasswordDialog Variable 7643
DB.RemoteLoginDialogProc Variable 7643
DB.ScreenCursorProc Variable 7644

Constants **7645**
DB.dsEditModes Constant 7645
DB.dsMaxStringSize Constant 7645
DB.dsWriteModes Constant 7646
DB.FieldTypeNames Constant 7646
DB.FieldTypeVarMap Constant 7646
DB.ftFixedSizeTypes Constant 7647
DB.ftNonTextTypes Constant 7647
DB.ObjectFieldTypes Constant 7648
DB.RegisterFieldsProc Constant 7648

DBActns Namespace 7649

Classes **7650**
TDataSetAction Class 7650
TDataSetAction Members 7651
TDataSetAction Methods 7655
TDataSetAction Properties 7656
TDataSetCancel Class 7657
TDataSetCancel Members 7658
TDataSetCancel Methods 7662
TDataSetCancel Properties 7663
TDataSetDelete Class 7664
TDataSetDelete Members 7665
TDataSetDelete Methods 7669
TDataSetDelete Properties 7670
TDataSetEdit Class 7671
TDataSetEdit Members 7671
TDataSetEdit Methods 7676
TDataSetEdit Properties 7677

TDataSetFirst Class 7677
TDataSetFirst Members 7678
TDataSetFirst Methods 7682
TDataSetFirst Properties 7684
TDataSetInsert Class 7684
TDataSetInsert Members 7685
TDataSetInsert Methods 7689
TDataSetInsert Properties 7690
TDataSetLast Class 7691
TDataSetLast Members 7692
TDataSetLast Methods 7696
TDataSetLast Properties 7697
TDataSetNext Class 7698
TDataSetNext Members 7698
TDataSetNext Methods 7703
TDataSetNext Properties 7704
TDataSetPost Class 7704
TDataSetPost Members 7705
TDataSetPost Methods 7709
TDataSetPost Properties 7710
TDataSetPrior Class 7711
TDataSetPrior Members 7712
TDataSetPrior Methods 7716
TDataSetPrior Properties 7717
TDataSetRefresh Class 7718
TDataSetRefresh Members 7719
TDataSetRefresh Methods 7723
TDataSetRefresh Properties 7724

DBAdapt Namespace 7725

Classes 7728

TBaseDataSetAdapterField Class 7729
TBaseDataSetAdapterField Members 7730
TBaseDataSetAdapterField.Create Constructor 7735
TBaseDataSetAdapterField Methods 7735
TBaseDataSetAdapterField Properties 7751
TBaseDataSetAdapterField Events 7753
TBaseDataSetAdapterImageField Class 7755

TBaseDataSetAdapterInputField Class 7756
TBaseDataSetAdapterInputField Members 7757
TBaseDataSetAdapterInputField Methods 7762
TBaseDataSetAdapterInputField Properties 7765
TBaseDataSetAdapterInputField Events 7766
TCustomDataSetAdapter Class 7766
TCustomDataSetAdapter Members 7768
TCustomDataSetAdapter.Create Constructor 7775
TCustomDataSetAdapter.Destroy Destructor 7776
TCustomDataSetAdapter Methods 7776
TCustomDataSetAdapter Properties 7806
TCustomDataSetAdapter Events 7810
TCustomDataSetAdapterAction Class 7811
TCustomDataSetAdapterAction Members 7813
TCustomDataSetAdapterAction.Create Constructor 7818
TCustomDataSetAdapterAction Methods 7819
TCustomDataSetAdapterAction Properties 7820
TCustomDataSetAdapterAction Events 7822
TCustomDataSetAdapterField Class 7823
TCustomDataSetAdapterField Members 7825
TCustomDataSetAdapterField Methods 7831
TCustomDataSetAdapterField Properties 7837
TCustomDataSetAdapterMemoField Class 7837
TCustomDataSetAdapterMemoField Members 7839
TCustomDataSetAdapterMemoField Methods 7844
TCustomDataSetAdapterRowAction Class 7846
TCustomDataSetAdapterRowAction Members 7848
TCustomDataSetAdapterRowAction.Create Constructor 7853
TCustomDataSetAdapterRowAction Methods 7854
TCustomDataSetValuesList Class 7855
TCustomDataSetValuesList Members 7857
TCustomDataSetValuesList.Create Constructor 7863
TCustomDataSetValuesList.Destroy Destructor 7864
TCustomDataSetValuesList Methods 7864
TCustomDataSetValuesList Properties 7869
TCustomDataSetValuesList Events 7871
TDataSetAdapter Class 7872
TDataSetAdapter Members 7874
TDataSetAdapter Properties 7883

TDataSetAdapter Events 7885
TDataSetAdapterApplyRowAction Class 7890
TDataSetAdapterApplyRowAction Members 7891
TDataSetAdapterApplyRowAction.Create Constructor 7897
TDataSetAdapterApplyRowAction Methods 7897
TDataSetAdapterApplyRowAction Properties 7899
TDataSetAdapterApplyRowAction Events 7901
TDataSetAdapterBrowseRowAction Class 7903
TDataSetAdapterBrowseRowAction Members 7905
TDataSetAdapterBrowseRowAction.Create Constructor 7910
TDataSetAdapterBrowseRowAction Methods 7911
TDataSetAdapterBrowseRowAction Properties 7913
TDataSetAdapterBrowseRowAction Events 7914
TDataSetAdapterCancelRowAction Class 7916
TDataSetAdapterCancelRowAction Members 7918
TDataSetAdapterCancelRowAction.Create Constructor 7924
TDataSetAdapterCancelRowAction Methods 7924
TDataSetAdapterCancelRowAction Properties 7926
TDataSetAdapterCancelRowAction Events 7928
TDataSetAdapterDataLink Class 7930
TDataSetAdapterDataLink Members 7930
TDataSetAdapterDataLink.Create Constructor 7932
TDataSetAdapterDataLink Methods 7932
TDataSetAdapterDataLink Events 7934
TDataSetAdapterDeleteRowAction Class 7934
TDataSetAdapterDeleteRowAction Members 7936
TDataSetAdapterDeleteRowAction Methods 7942
TDataSetAdapterDeleteRowAction Properties 7943
TDataSetAdapterDeleteRowAction Events 7944
TDataSetAdapterEditRowAction Class 7947
TDataSetAdapterEditRowAction Members 7948
TDataSetAdapterEditRowAction.Create Constructor 7954
TDataSetAdapterEditRowAction Methods 7954
TDataSetAdapterEditRowAction Properties 7956
TDataSetAdapterEditRowAction Events 7957
TDataSetAdapterField Class 7960
TDataSetAdapterField Members 7961
TDataSetAdapterField Properties 7967
TDataSetAdapterField Events 7971

TDataSetAdapterFirstRowAction Class 7972
TDataSetAdapterFirstRowAction Members 7974
TDataSetAdapterFirstRowAction Methods 7979
TDataSetAdapterFirstRowAction Properties 7981
TDataSetAdapterFirstRowAction Events 7982
TDataSetAdapterLastRowAction Class 7984
TDataSetAdapterLastRowAction Members 7986
TDataSetAdapterLastRowAction Methods 7991
TDataSetAdapterLastRowAction Properties 7993
TDataSetAdapterLastRowAction Events 7994
TDataSetAdapterMemoField Class 7996
TDataSetAdapterMemoField Members 7998
TDataSetAdapterMemoField Properties 8004
TDataSetAdapterMemoField Events 8007
TDataSetAdapterMoveRowAction Class 8008
TDataSetAdapterMoveRowAction Members 8010
TDataSetAdapterMoveRowAction Methods 8015
TDataSetAdapterNewRowAction Class 8016
TDataSetAdapterNewRowAction Members 8018
TDataSetAdapterNewRowAction.Create Constructor 8024
TDataSetAdapterNewRowAction Methods 8024
TDataSetAdapterNewRowAction Properties 8026
TDataSetAdapterNewRowAction Events 8027
TDataSetAdapterNextRowAction Class 8030
TDataSetAdapterNextRowAction Members 8031
TDataSetAdapterNextRowAction Methods 8037
TDataSetAdapterNextRowAction Properties 8038
TDataSetAdapterNextRowAction Events 8039
TDataSetAdapterPrevRowAction Class 8042
TDataSetAdapterPrevRowAction Members 8043
TDataSetAdapterPrevRowAction Methods 8049
TDataSetAdapterPrevRowAction Properties 8050
TDataSetAdapterPrevRowAction Events 8051
TDataSetAdapterRefreshRowAction Class 8054
TDataSetAdapterRefreshRowAction Members 8055
TDataSetAdapterRefreshRowAction Methods 8061
TDataSetAdapterRefreshRowAction Properties 8062
TDataSetAdapterRefreshRowAction Events 8064
TDataSetAdapterRowAction Class 8066

TDataSetAdapterRowAction Members 8068
TDataSetAdapterRowAction Methods 8073
TDataSetAdapterRowAction Properties 8075
TDataSetAdapterRowAction Events 8077
TDataSetValuesList Class 8079
TDataSetValuesList Members 8081
TDataSetValuesList Properties 8088
TDataSetValuesList Events 8090
TLocateParams Class 8090
TLocateParams Members 8091
TLocateParams.Create Constructor 8092
TLocateParams.Destroy Destructor 8092
TLocateParams Methods 8093
TLocateParams Properties 8094
TLocateParamsList Class 8096
TLocateParamsList Members 8097
TLocateParamsList.Create Constructor 8098
TLocateParamsList.Destroy Destructor 8098
TLocateParamsList Methods 8099
TLocateParamsList Properties 8100

Interfaces 8102
IAdapterDataSetComponent Interface 8102
IAdapterDataSetComponent Members 8102
IAdapterDataSetComponent Methods 8103
IAdapterDataSetComponent Properties 8103
IDataSetAdapterFieldClass Interface 8104
IGetDataSetKeyName Interface 8104
IGetDataSetKeyName Members 8104
IGetDataSetKeyName Methods 8105
IGetDataSetKeyName Properties 8105
IKeyParams Interface 8105
IKeyParams Members 8106
IKeyParams Methods 8106
IKeyParams Properties 8107
IPrepareDataSet Interface 8109
IPrepareDataSet Members 8109
IPrepareDataSet Methods 8109

Functions 8110

DBAdapt.AddLocateParamsString Function 8110

Structs, Records, Enums 8111

DBAdapt.TDataSetAdapterFieldFlag Enumeration 8111

DBAdapt.TDataSetAdapterMode Enumeration 8111

DBAdapt.TDataSetAdapterRequestParamOption Enumeration 8112

Types 8113

DBAdapt.TDataAdapterFieldValidateValueEvent Type 8113

DBAdapt.TDataSetAdapterFieldFlags Type 8113

DBAdapt.TDataSetAdapterFieldGetStringEvent Type 8114

DBAdapt.TDataSetAdapterFieldGetValueEvent Type 8114

DBAdapt.TDataSetAdapterFieldValueEvent Type 8114

DBAdapt.TDataSetAdapterImageFieldClass Type 8115

DBAdapt.TDataSetAdapterModes Type 8115

DBAdapt.TDataSetAdapterRequestParamOptions Type 8115

Variables 8116

DBAdapt.DataSetAdapterImageFieldClass Variable 8116

Constants 8117

DBAdapt.AllActionParamOptions Constant 8117

DBAdapt.AllImageParamOptions Constant 8117

DBAdapt.DataSetAdapterModeNames Constant 8117

DBAdaptImg Namespace 8119

Classes 8120

TCustomDataSetAdapterImageField Class 8120

TCustomDataSetAdapterImageField Members 8122

TCustomDataSetAdapterImageField.Create Constructor 8127

TCustomDataSetAdapterImageField Methods 8128

TCustomDataSetAdapterImageField Events 8137

TDataSetAdapterImageField Class 8138

TDataSetAdapterImageField Members 8139

TDataSetAdapterImageField Properties 8145

TDataSetAdapterImageField Events 8147

Types 8149

DBAdaptImg.TImageDataSetFieldGetImageEvent Type 8149

DBBdeWeb Namespace 8151

Classes **8152**
TQueryTableProducer Class 8152
TQueryTableProducer Members 8153
TQueryTableProducer Methods 8157
TQueryTableProducer Properties 8159
TQueryTableProducer Events 8164

DBCGrids Namespace 8167

Classes **8168**
TDBCtrlGrid Class 8168
TDBCtrlGrid Members 8169
TDBCtrlGrid.Create Constructor 8179
TDBCtrlGrid.Destroy Destructor 8179
TDBCtrlGrid Methods 8180
TDBCtrlGrid Properties 8187
TDBCtrlGrid Events 8213
TDBCtrlGridLink Class 8240
TDBCtrlGridLink Members 8241
TDBCtrlGridLink.Create Constructor 8243
TDBCtrlGridLink Methods 8243
TDBCtrlPanel Class 8244
TDBCtrlPanel Members 8245
TDBCtrlPanel.CreateLinked Constructor 8255
TDBCtrlPanel Methods 8255

Structs, Records, Enums **8257**
DBCGrids.TDBCtrlGridBorder Enumeration 8257
DBCGrids.TDBCtrlGridKey Enumeration 8257
DBCGrids.TDBCtrlGridOrientation Enumeration 8258

Types **8259**
DBCGrids.TPaintPanelEvent Type 8259

DBClient Namespace 8261

Classes **8263**
EDBClient Class 8263
EDBClient Members 8263
EDBClient.Create Constructor 8264
EDBClient Properties 8265

EReconcileError Class 8265
EReconcileError Members 8266
EReconcileError.Create Constructor 8267
EReconcileError Properties 8267
TAggregate Class 8268
TAggregate Members 8269
TAggregate.Create Constructor 8270
TAggregate.Destroy Destructor 8271
TAggregate Methods 8271
TAggregate Properties 8273
TAggregate Events 8279
TAggregates Class 8280
TAggregates Members 8280
TAggregates.Create Constructor 8282
TAggregates Methods 8282
TAggregates Properties 8284
TClientBlobStream Class 8285
TClientBlobStream Members 8286
TClientBlobStream.Create Constructor 8287
TClientBlobStream.Destroy Destructor 8288
TClientBlobStream Methods 8288
TClientDataSet Class 8289
TClientDataSet Members 8291
TClientDataSet Properties 8304
TClientDataSet Events 8331
TConnectionBroker Class 8360
TConnectionBroker Members 8361
TConnectionBroker.Create Constructor 8364
TConnectionBroker Methods 8365
TConnectionBroker Properties 8367
TConnectionBroker Events 8371
TCustomClientDataSet Class 8373
TCustomClientDataSet Members 8374
TCustomClientDataSet.Create Constructor 8386
TCustomClientDataSet.Destroy Destructor 8387
TCustomClientDataSet Methods 8387
TCustomClientDataSet Properties 8456
TCustomClientDataSet Events 8496
TCustomRemoteServer Class 8525

TCustomRemoteServer Members 8526
TCustomRemoteServer.Create Constructor 8529
TCustomRemoteServer Methods 8529
TCustomRemoteServer Properties 8532
TCustomRemoteServer Events 8533

Functions 8534
DBClient.PackageParams Function 8534
DBClient.UnpackParams Function 8534

Structs, Records, Enums 8536
DBClient.TDataPacketFormat Enumeration 8536
DBClient.TDataSetOption Enumeration 8537
DBClient.TFetchOption Enumeration 8537
DBClient.TKeyBuffer Record 8538
DBClient.TRecInfo Record 8538
DBClient.TReconcileAction Enumeration 8539
DBClient.TReconcileInfo Record 8539

Types 8541
DBClient.PKeyBuffer Type 8541
DBClient.PRecInfo Type 8541
DBClient.TAggUpdateEvent Type 8542
DBClient.TCDSKeyBuffer Type 8542
DBClient.TCDSRecInfo Type 8542
DBClient.TDataSetOptions Type 8543
DBClient.TFetchOptions Type 8543
DBClient.TFieldDescList Type 8543
DBClient.TGetUsernameEvent Type 8544
DBClient.TReconcileErrorEvent Type 8544
DBClient.TRemoteEvent Type 8544

Constants 8545
DBClient.AllParamTypes Constant 8545
DBClient.AllRecords Constant 8545

DBClientActns Namespace 8547

Classes 8548
TClientDataSetApply Class 8548
TClientDataSetApply Members 8549
TClientDataSetApply.Create Constructor 8553

TClientDataSetApply.Destroy Destructor 8553
TClientDataSetApply Methods 8554
TClientDataSetApply Properties 8555
TClientDataSetRevert Class 8556
TClientDataSetRevert Members 8557
TClientDataSetRevert Methods 8561
TClientDataSetRevert Properties 8563
TClientDataSetUndo Class 8563
TClientDataSetUndo Members 8564
TClientDataSetUndo Methods 8568
TClientDataSetUndo Properties 8569
TReconcileActionLink Class 8570
TReconcileActionLink Members 8571
TReconcileActionLink Methods 8573
TReconcileActionLink Properties 8573

DBCtrls Namespace 8575

Classes 8577
TDataSourceLink Class 8577
TDataSourceLink Members 8578
TDataSourceLink.Create Constructor 8580
TDataSourceLink Methods 8580
TDBCheckBox Class 8582
TDBCheckBox Members 8583
TDBCheckBox.Create Constructor 8593
TDBCheckBox.Destroy Destructor 8594
TDBCheckBox Methods 8594
TDBCheckBox Properties 8597
TDBCheckBox Events 8628
TDBComboBox Class 8657
TDBComboBox Members 8657
TDBComboBox.Create Constructor 8669
TDBComboBox.Destroy Destructor 8670
TDBComboBox Methods 8670
TDBComboBox Properties 8677
TDBComboBox Events 8710
TDBEdit Class 8735
TDBEdit Members 8736

TDBEdit.Create Constructor 8747
TDBEdit.Destroy Destructor 8748
TDBEdit Methods 8748
TDBEdit Properties 8753
TDBEdit Events 8783
TDBImage Class 8813
TDBImage Members 8813
TDBImage.Create Constructor 8824
TDBImage.Destroy Destructor 8824
TDBImage Methods 8825
TDBImage Properties 8830
TDBImage Events 8855
TDBListBox Class 8884
TDBListBox Members 8885
TDBListBox.Create Constructor 8896
TDBListBox.Destroy Destructor 8897
TDBListBox Methods 8897
TDBListBox Properties 8901
TDBListBox Events 8931
TDBLookupComboBox Class 8963
TDBLookupComboBox Members 8964
TDBLookupComboBox.Create Constructor 8975
TDBLookupComboBox Methods 8975
TDBLookupComboBox Properties 8983
TDBLookupComboBox Events 9014
TDBLookupControl Class 9044
TDBLookupControl Members 9045
TDBLookupControl.Create Constructor 9055
TDBLookupControl.Destroy Destructor 9056
TDBLookupControl Methods 9056
TDBLookupControl Properties 9062
TDBLookupListBox Class 9070
TDBLookupListBox Members 9071
TDBLookupListBox.Create Constructor 9082
TDBLookupListBox Methods 9082
TDBLookupListBox Properties 9090
TDBLookupListBox Events 9120
TDBMemo Class 9149
TDBMemo Members 9150

TDBMemo.Create Constructor 9161
TDBMemo.Destroy Destructor 9162
TDBMemo Methods 9162
TDBMemo Properties 9167
TDBMemo Events 9201
TDBNavigator Class 9230
TDBNavigator Members 9233
TDBNavigator.Create Constructor 9243
TDBNavigator.Destroy Destructor 9244
TDBNavigator Fields 9244
TDBNavigator Methods 9244
TDBNavigator Properties 9249
TDBNavigator Events 9269
TDBRadioGroup Class 9283
TDBRadioGroup Members 9284
TDBRadioGroup.Create Constructor 9294
TDBRadioGroup.Destroy Destructor 9295
TDBRadioGroup Methods 9295
TDBRadioGroup Properties 9299
TDBRadioGroup Events 9328
TDBRichEdit Class 9343
TDBRichEdit Members 9344
TDBRichEdit.Create Constructor 9356
TDBRichEdit.Destroy Destructor 9356
TDBRichEdit Methods 9357
TDBRichEdit Properties 9361
TDBRichEdit Events 9400
TDBText Class 9433
TDBText Members 9433
TDBText.Create Constructor 9441
TDBText.Destroy Destructor 9441
TDBText Methods 9442
TDBText Properties 9444
TDBText Events 9468
TFieldDataLink Class 9487
TFieldDataLink Members 9487
TFieldDataLink.Create Constructor 9489
TFieldDataLink Methods 9490
TFieldDataLink Properties 9494

TFieldDataLink Events 9496
TListSourceLink Class 9498
TListSourceLink Members 9499
TListSourceLink.Create Constructor 9500
TListSourceLink Methods 9501
TNavButton Class 9502
TNavButton Members 9503
TNavButton.Destroy Destructor 9511
TNavButton Methods 9511
TNavButton Properties 9513
TNavDataLink Class 9513
TNavDataLink Members 9514
TNavDataLink.Create Constructor 9515
TNavDataLink.Destroy Destructor 9516
TNavDataLink Methods 9516
TPaintControl Class 9518
TPaintControl Members 9518
TPaintControl.Create Constructor 9518
TPaintControl.Destroy Destructor 9519
TPaintControl Methods 9519
TPaintControl Properties 9520
TPopupDataList Class 9521
TPopupDataList Members 9521
TPopupDataList.Create Constructor 9534
TPopupDataList Methods 9534

Functions **9536**
DBCtrls.DBUseRightToLeftAlignment Function 9536
DBCtrls.OkToChangeFieldAlignment Function 9536

Structs, Records, Enums **9538**
DBCtrls.TDropDownAlign Enumeration 9538
DBCtrls.TNavGlyph Enumeration 9539
DBCtrls.TNavigateBtn Enumeration 9539

Types **9541**
DBCtrls.ENavClick Type 9541
DBCtrls.TNavButtonSet Type 9541
DBCtrls.TNavButtonStyle Type 9541

Constants **9543**

DBCtrls.InitRepeatPause Constant 9543
DBCtrls.RepeatPause Constant 9543
DBCtrls.SpaceSize Constant 9543

DbExcept Namespace 9545

Classes 9546
TDbEngineErrorDlg Class 9546
TDbEngineErrorDlg Members 9546
TDbEngineErrorDlg Fields 9563
TDbEngineErrorDlg Methods 9567
TDbEngineErrorDlg Properties 9571

Variables 9572
DbExcept.DbEngineErrorDlg Variable 9572

DBGrids Namespace 9573

Classes 9575
TBookmarkList Class 9575
TBookmarkList Members 9575
TBookmarkList.Create Constructor 9576
TBookmarkList.Destroy Destructor 9576
TBookmarkList Methods 9577
TBookmarkList Properties 9580
TColumn Class 9583
TColumn Members 9583
TColumn.Create Constructor 9585
TColumn.Destroy Destructor 9585
TColumn Methods 9586
TColumn Properties 9592
TColumnTitle Class 9601
TColumnTitle Members 9601
TColumnTitle.Create Constructor 9603
TColumnTitle.Destroy Destructor 9603
TColumnTitle Methods 9603
TColumnTitle Properties 9606
TCustomDBGrid Class 9608
TCustomDBGrid Members 9609
TCustomDBGrid.Create Constructor 9622

TCustomDBGrid.Destroy Destructor 9622
TCustomDBGrid Fields 9623
TCustomDBGrid Methods 9623
TCustomDBGrid Properties 9653
TCustomDBGrid Events 9664
TDBGrid Class 9668
TDBGrid Members 9670
TDBGrid Properties 9685
TDBGrid Events 9714
TDBGridColumns Class 9746
TDBGridColumns Members 9746
TDBGridColumns.Create Constructor 9748
TDBGridColumns Methods 9749
TDBGridColumns Properties 9752
TGridDataLink Class 9754
TGridDataLink Members 9754
TGridDataLink.Create Constructor 9755
TGridDataLink.Destroy Destructor 9756
TGridDataLink Methods 9756
TGridDataLink Properties 9762

Structs, Records, Enums 9765
DBGrids.TColumnButtonStyle Enumeration 9765
DBGrids.TColumnValue Enumeration 9766
DBGrids.TDBGridColumnsState Enumeration 9767
DBGrids.TDBGridOption Enumeration 9767

Types 9769
DBGrids.TColumnClass Type 9769
DBGrids.TColumnValues Type 9769
DBGrids.TDBGridClickEvent Type 9770
DBGrids.TDBGridOptions Type 9770
DBGrids.TDrawColumnCellEvent Type 9770
DBGrids.TDrawDataCellEvent Type 9771

Constants 9772
DBGrids.cm_DeferLayout Constant 9772
DBGrids.ColumnTitleValues Constant 9772
DBGrids.IndicatorWidth Constant 9772

DBLogDlg Namespace 9775

Classes **9776**
TLoginDialog Class 9776
TLoginDialog Members 9776
TLoginDialog Fields 9793
TLoginDialog Methods 9795

Functions **9797**
DBLogDlg.LoginDialog Function 9797
DBLogDlg.LoginDialogEx Function 9798
DBLogDlg.RemoteLoginDialog Function 9798
DBLogDlg.SetCursorType Function 9799

DBOleCtl Namespace 9801

Classes **9802**
TDataBindings Class 9802
TDataBindings Members 9802
TDataBindings.Create Constructor 9804
TDataBindings Methods 9804
TDataBindings Properties 9806
TDataBindItem Class 9806
TDataBindItem Members 9807
TDataBindItem.Create Constructor 9808
TDataBindItem.Destroy Destructor 9808
TDataBindItem Properties 9809
TDBOleControl Class 9810
TDBOleControl Members 9810
TDBOleControl.Create Constructor 9823
TDBOleControl.Destroy Destructor 9823
TDBOleControl Methods 9824
TDBOleControl Properties 9825

DBPWDlg Namespace 9827

Classes **9828**
TPasswordDialog Class 9828
TPasswordDialog Members 9828
TPasswordDialog Fields 9845
TPasswordDialog Methods 9847

Functions **9849**

DBPWDlg.PasswordDialog Function 9849

DBTables Namespace 9851

Classes 9854

EDBEngineError Class 9854

EDBEngineError Members 9855

EDBEngineError.Create Constructor 9856

EDBEngineError.Destroy Destructor 9856

EDBEngineError Properties 9857

ENoResultSet Class 9858

TBatchMove Class 9858

TBatchMove Members 9859

TBatchMove.Create Constructor 9862

TBatchMove.Destroy Destructor 9863

TBatchMove Methods 9863

TBatchMove Properties 9864

TBDECallback Class 9875

TBDECallback Members 9876

TBDECallback.Create Constructor 9876

TBDECallback.Destroy Destructor 9877

TBDECallback Methods 9878

TBDEDataSet Class 9878

TBDEDataSet Members 9879

TBDEDataSet.Create Constructor 9889

TBDEDataSet.Destroy Destructor 9890

TBDEDataSet Methods 9890

TBDEDataSet Properties 9929

TBDEDataSet Events 9938

TBlobStream Class 9958

TBlobStream Members 9959

TBlobStream.Create Constructor 9960

TBlobStream.Destroy Destructor 9961

TBlobStream Methods 9962

TDatabase Class 9964

TDatabase Members 9965

TDatabase.Create Constructor 9969

TDatabase.Destroy Destructor 9969

TDatabase Methods 9970

TDatabase Properties 9978
TDatabase Events 9992
TDataSetUpdateObject Class 9994
TDataSetUpdateObject Members 9995
TDataSetUpdateObject Methods 9998
TDataSetUpdateObject Properties 9999
TDBDataSet Class 10000
TDBDataSet Members 10000
TDBDataSet.Create Constructor 10013
TDBDataSet Methods 10013
TDBDataSet Properties 10020
TDBDataSet Events 10028
TDBError Class 10029
TDBError Members 10030
TDBError.Create Constructor 10030
TDBError Properties 10031
TIndexFiles Class 10033
TIndexFiles Members 10033
TIndexFiles.Create Constructor 10036
TIndexFiles Methods 10036
TNestedTable Class 10040
TNestedTable Members 10041
TNestedTable.Create Constructor 10053
TNestedTable Methods 10053
TNestedTable Properties 10054
TParamList Class 10056
TParamList Members 10056
TParamList.Create Constructor 10057
TParamList.Destroy Destructor 10057
TParamList Properties 10058
TQuery Class 10059
TQuery Members 10060
TQuery.Create Constructor 10073
TQuery.Destroy Destructor 10073
TQuery Methods 10073
TQuery Properties 10082
TSession Class 10093
TSession Members 10094
TSession.Create Constructor 10098

TSession.Destroy Destructor 10099
TSession Methods 10099
TSession Properties 10120
TSession Events 10128
TSessionList Class 10129
TSessionList Members 10130
TSessionList.Create Constructor 10131
TSessionList.Destroy Destructor 10131
TSessionList Methods 10131
TSessionList Properties 10133
TSQLUpdateObject Class 10135
TSQLUpdateObject Members 10136
TSQLUpdateObject Methods 10138
TStoredProc Class 10139
TStoredProc Members 10140
TStoredProc.Create Constructor 10152
TStoredProc.Destroy Destructor 10153
TStoredProc Methods 10153
TStoredProc Properties 10163
TTable Class 10168
TTable Members 10169
TTable.Create Constructor 10182
TTable.Destroy Destructor 10182
TTable Methods 10183
TTable Properties 10213
TUpdateSQL Class 10229
TUpdateSQL Members 10230
TUpdateSQL.Create Constructor 10233
TUpdateSQL.Destroy Destructor 10233
TUpdateSQL Methods 10233
TUpdateSQL Properties 10238

Functions 10244

DBTables.AnsiToNative Function 10244
DBTables.AnsiToNativeBuf Function 10245
DBTables.Check Function 10245
DBTables.DbiError Function 10246
DBTables.GetFieldSource Function 10246
DBTables.NativeCompareStr Function 10246
DBTables.NativeCompareStrBuf Function 10247

DBTables.NativeCompareText Function 10248
DBTables.NativeCompareTextBuf Function 10248
DBTables.NativeToAnsi Function 10249
DBTables.NativeToAnsiBuf Function 10249
DBTables.RegisterBDEInitProc Function 10250

Structs, Records, Enums 10251
DBTables.TBatchMode Enumeration 10251
DBTables.TBDEKeyBuffer Record 10252
DBTables.TBDERecInfo Record 10252
DBTables.TConfigModes Enumeration 10253
DBTables.TDatabaseEvent Enumeration 10253
DBTables.TLockType Enumeration 10254
DBTables.TParamBindMode Enumeration 10254
DBTables.TRecNoStatus Enumeration 10255
DBTables.TServerDesc Record 10255
DBTables.TTableType Enumeration 10256
DBTables.TTraceFlag Enumeration 10256
DBTables.TTransIsolation Enumeration 10257

Types 10258
DBTables.PKeyBuffer Type 10258
DBTables.PRecInfo Type 10258
DBTables.TBDECallbackEvent Type 10259
DBTables.TBDEInitProc Type 10259
DBTables.TBlobDataArray Type 10259
DBTables.TConfigMode Type 10260
DBTables.TDatabaseLoginEvent Type 10260
DBTables.TDatabaseNotifyEvent Type 10260
DBTables.TDBFlags Type 10261
DBTables.TFieldDescList Type 10261
DBTables.TIndexDescList Type 10261
DBTables.TIndexName Type 10262
DBTables.TLocale Type 10262
DBTables.TPasswordEvent Type 10262
DBTables.TServerDescList Type 10263
DBTables.TSPParamDescList Type 10263
DBTables.TTraceFlags Type 10263
DBTables.TValCheckList Type 10264

Variables 10265

DBTables.GetObjectContextProc Variable 10265
DBTables.Session Variable 10265
DBTables.Sessions Variable 10266

Constants 10267
DBTables.BlobTypeMap Constant 10267
DBTables.cmAll Constant 10268
DBTables.cmPersistent Constant 10268
DBTables.cmSession Constant 10268
DBTables.cmVirtual Constant 10269
DBTables.DataTypeMap Constant 10269
DBTables.dbfDatabase Constant 10269
DBTables.dbfExecProc Constant 10270
DBTables.dbfExecSQL Constant 10270
DBTables.dbfFieldList Constant 10270
DBTables.dbfIndexList Constant 10271
DBTables.dbfOpened Constant 10271
DBTables.dbfPrepared Constant 10271
DBTables.dbfProcDesc Constant 10272
DBTables.dbfProvider Constant 10272
DBTables.dbfStoredProc Constant 10272
DBTables.dbfTable Constant 10273
DBTables.FldSubTypeMap Constant 10273
DBTables.FldTypeMap Constant 10273
DBTables.smTraceBufSize Constant 10274

DBWeb Namespace 10275

Classes 10277
TDataSetTableProducer Class 10277
TDataSetTableProducer Members 10278
TDataSetTableProducer Methods 10282
TDataSetTableProducer Properties 10284
TDataSetTableProducer Events 10289
TDSTableProducer Class 10290
TDSTableProducer Members 10291
TDSTableProducer.Create Constructor 10295
TDSTableProducer.Destroy Destructor 10295
TDSTableProducer Methods 10296
TDSTableProducer Properties 10307

TDSTableProducer Events 10313
TDSTableProducerEditor Class 10315
TDSTableProducerEditor Members 10315
TDSTableProducerEditor.Create Constructor 10316
TDSTableProducerEditor.Destroy Destructor 10316
TDSTableProducerEditor Methods 10317
TDSTableProducerEditor Properties 10318
THTMLTableColumn Class 10318
THTMLTableColumn Members 10319
THTMLTableColumn.Create Constructor 10320
THTMLTableColumn.Destroy Destructor 10321
THTMLTableColumn Methods 10321
THTMLTableColumn Properties 10324
THTMLTableColumns Class 10327
THTMLTableColumns Members 10328
THTMLTableColumns.Create Constructor 10330
THTMLTableColumns Methods 10330
THTMLTableColumns Properties 10334
THTTPDataLink Class 10335
THTTPDataLink Members 10336
THTTPDataLink.Create Constructor 10338
THTTPDataLink.Destroy Destructor 10338
THTTPDataLink Methods 10338
THTTPDataLink Properties 10344

Functions 10346
DBWeb.HtmlTable Function 10346

Structs, Records, Enums 10347
DBWeb.THTMLCaptionAlignment Enumeration 10347
DBWeb.THTMLColumnState Enumeration 10347

Types 10349
DBWeb.TCreateContentEvent Type 10349
DBWeb.THTMLDataSetEmpty Type 10349
DBWeb.THTMLFormatCellEvent Type 10350
DBWeb.THTMLGetTableCaptionEvent Type 10350
DBWeb.THTMLTableColumnClass Type 10350

DBXCommon Namespace 10351

Classes **10355**
TDBXAnsiStringValue Class 10356
TDBXAnsiStringValue Members 10357
TDBXAnsiStringValue.Create Constructor 10358
TDBXAnsiStringValue.Destroy Destructor 10358
TDBXAnsiStringValue Methods 10358
TDBXBcdValue Class 10359
TDBXBcdValue Members 10360
TDBXBcdValue Methods 10361
TDBXBooleanValue Class 10362
TDBXBooleanValue Members 10362
TDBXBooleanValue Methods 10363
TDBXByteArrayValue Class 10364
TDBXByteArrayValue Members 10365
TDBXByteArrayValue.Destroy Destructor 10366
TDBXByteArrayValue Methods 10366
TDBXByteReader Class 10368
TDBXByteReader Members 10368
TDBXByteReader.Create Constructor 10369
TDBXByteReader Fields 10369
TDBXByteReader Methods 10369
TDBXByteReaderEx Class 10373
TDBXByteReaderEx Members 10373
TDBXByteReaderEx Methods 10374
TDBXByteStreamReader Class 10374
TDBXByteStreamReader Members 10374
TDBXByteStreamReader.Create Constructor 10375
TDBXByteStreamReader.Destroy Destructor 10375
TDBXByteStreamReader Fields 10375
TDBXByteStreamReader Methods 10377
TDBXCommand Class 10378
TDBXCommand Members 10379
TDBXCommand.Create Constructor 10380
TDBXCommand.Destroy Destructor 10380
TDBXCommand Fields 10380
TDBXCommand Methods 10381
TDBXCommand Properties 10388
TDBXCommand Events 10390
TDBXCommandEx Class 10391

TDBXCommandEx Members 10391
TDBXCommandEx Methods 10393
TDBXCommandEx Properties 10394
TDBXCommandFactory Class 10395
TDBXCommandFactory Members 10395
TDBXCommandFactory Methods 10395
TDBXCommandTypes Class 10396
TDBXCommandTypes Members 10396
TDBXCommandTypes Constants 10397
TDBXConnection Class 10398
TDBXConnection Members 10399
TDBXConnection.Create Constructor 10400
TDBXConnection.Destroy Destructor 10400
TDBXConnection Fields 10401
TDBXConnection Methods 10402
TDBXConnection Properties 10410
TDBXConnection Events 10411
TDBXConnectionBuilder Class 10412
TDBXConnectionBuilder Members 10412
Create Constructor 10413
TDBXConnectionBuilder.Destroy Destructor 10414
TDBXConnectionBuilder Methods 10414
TDBXConnectionBuilder Properties 10414
TDBXConnectionEx Class 10417
TDBXConnectionEx Members 10418
TDBXConnectionEx.Destroy Destructor 10419
TDBXConnectionEx Methods 10419
TDBXConnectionEx Properties 10420
TDBXConnectionFactory Class 10422
TDBXConnectionFactory Members 10423
TDBXConnectionFactory.Create Constructor 10423
TDBXConnectionFactory.Destroy Destructor 10424
TDBXConnectionFactory Fields 10424
TDBXConnectionFactory Methods 10425
TDBXConnectionFactory Properties 10431
TDBXConnectionFactory Events 10431
TDBXContext Class 10432
TDBXContext Members 10433
Create Constructor 10434

TDBXContext.Destroy Destructor 10434
TDBXContext Fields 10434
TDBXContext Methods 10435
TDBXContext Properties 10436
TDBXContext Events 10437
TDBXCursorValue Class 10438
TDBXCursorValue Members 10439
TDBXCursorValue Methods 10440
TDBXDatabaseMetaData Class 10440
TDBXDatabaseMetaData Members 10441
TDBXDatabaseMetaData.Create Constructor 10441
TDBXDatabaseMetaData Fields 10441
TDBXDatabaseMetaData Properties 10442
TDBXDatabaseMetaDataEx Class 10444
TDBXDatabaseMetaDataEx Members 10445
TDBXDatabaseMetaDataEx.Create Constructor 10445
TDBXDatabaseMetaDataEx Properties 10445
TDBXDataTypes Class 10447
TDBXDataTypes Members 10449
TDBXDataTypes Constants 10449
TDBXDataTypesEx Class 10459
TDBXDataTypesEx Members 10459
TDBXDataTypesEx Constants 10460
TDBXDateValue Class 10461
TDBXDateValue Members 10462
TDBXDateValue Methods 10463
TDBXDelegateDriver Class 10464
TDBXDelegateDriver Members 10464
TDBXDelegateDriver.Create Constructor 10465
TDBXDelegateDriver.Destroy Destructor 10465
TDBXDelegateDriver Methods 10465
TDBXDelegateItem Class 10468
TDBXDelegateItem Members 10468
TDBXDelegateItem.Destroy Destructor 10468
TDBXDelegateItem Fields 10469
TDBXDoubleValue Class 10469
TDBXDoubleValue Members 10470
TDBXDoubleValue Methods 10471
TDBXDriver Class 10472

TDBXDriver Members 10472
TDBXDriver.Destroy Destructor 10473
TDBXDriver Fields 10473
TDBXDriver Methods 10474
TDBXDriver Properties 10476
TDBXDriverEx Class 10477
TDBXDriverEx Members 10477
TDBXDriverEx.Create Constructor 10478
TDBXDriverEx.Destroy Destructor 10478
TDBXDriverEx Fields 10478
TDBXDriverEx Methods 10479
TDBXDriverHelp Class 10480
TDBXDriverHelp Members 10480
TDBXDriverHelp Methods 10480
TDBXDriverHelpEx Class 10482
TDBXDriverHelpEx Members 10482
TDBXDriverHelpEx Methods 10482
TDBXDriverLoader Class 10483
TDBXDriverLoader Members 10484
TDBXDriverLoader Fields 10484
TDBXDriverLoader Methods 10485
TDBXDriverRegistry Class 10485
TDBXDriverRegistry Members 10485
TDBXDriverRegistry.Create Constructor 10486
TDBXDriverRegistry.Destroy Destructor 10486
TDBXError Class 10486
TDBXError Members 10487
TDBXError.Create Constructor 10488
TDBXError Methods 10488
TDBXError Properties 10489
TDBXErrorCodes Class 10489
TDBXErrorCodes Members 10491
TDBXErrorCodes Constants 10492
TDBXIniFileConnectionFactory Class 10501
TDBXIniFileConnectionFactory Members 10501
TDBXIniFileConnectionFactory.Create Constructor 10502
TDBXIniFileConnectionFactory.Destroy Destructor 10502
TDBXIniFileConnectionFactory Methods 10503
TDBXIniFileConnectionFactory Properties 10503

TDBXIniFileConnectionFactoryEx Class 10504
TDBXIniFileConnectionFactoryEx Members 10504
TDBXIniFileConnectionFactoryEx Methods 10505
TDBXInt16Value Class 10506
TDBXInt16Value Members 10506
TDBXInt16Value Methods 10507
TDBXInt32Value Class 10508
TDBXInt32Value Members 10509
TDBXInt32Value Methods 10510
TDBXInt64Value Class 10511
TDBXInt64Value Members 10511
TDBXInt64Value Methods 10512
TDBXIsolations Class 10513
TDBXIsolations Members 10514
TDBXIsolations Constants 10514
TDBXLookAheadStreamReader Class 10516
TDBXLookAheadStreamReader Members 10516
TDBXLookAheadStreamReader.Destroy Destructor 10516
TDBXLookAheadStreamReader Methods 10517
TDBXMemoryConnectionFactory Class 10518
TDBXMemoryConnectionFactory Members 10518
TDBXMemoryConnectionFactory.Create Constructor 10519
TDBXMemoryConnectionFactory.Destroy Destructor 10520
TDBXMemoryConnectionFactory Methods 10520
TDBXMetaDatabaseColumnNames Class 10521
TDBXMetaDatabaseColumnNames Members 10522
TDBXMetaDatabaseColumnNames Constants 10522
TDBXMetaDataCommands Class 10525
TDBXMetaDataCommands Members 10528
TDBXMetaDataCommands Constants 10528
TDBXMetaDataCommandsEx Class 10531
TDBXMetaDataCommandsEx Members 10534
TDBXMetaDataCommandsEx Constants 10534
TDBXMetaDataKeyword Class 10537
TDBXMetaDataKeyword Members 10537
TDBXMetaDataKeyword Constants 10538
TDBXMetaDataProcedureTypes Class 10538
TDBXMetaDataProcedureTypes Members 10539
TDBXMetaDataProcedureTypes Constants 10539

TDBXMetaDataTableTypes Class 10540
TDBXMetaDataTableTypes Members 10540
TDBXMetaDataTableTypes Constants 10540
TDBXMetaDataTableTypesEx Class 10542
TDBXMetaDataTableTypesEx Members 10542
TDBXMetaDataTableTypesEx Constants 10542
TDBXNullValue Class 10543
TDBXNullValue Members 10543
TDBXNullValue.Create Constructor 10544
TDBXNullValue.Destroy Destructor 10544
TDBXNullValue Methods 10545
TDBXParameter Class 10545
TDBXParameter Members 10546
TDBXParameter.Create Constructor 10547
TDBXParameter.Destroy Destructor 10547
TDBXParameter Methods 10547
TDBXParameter Properties 10548
TDBXParameterDirections Class 10549
TDBXParameterDirections Members 10549
TDBXParameterDirections Constants 10549
TDBXParameterEx Class 10551
TDBXParameterEx Members 10551
TDBXParameterEx Methods 10552
TDBXParameterList Class 10553
TDBXParameterList Members 10554
TDBXParameterList.Create Constructor 10554
TDBXParameterList.Destroy Destructor 10555
TDBXParameterList Fields 10555
TDBXParameterList Methods 10555
TDBXParameterList Properties 10558
TDBXProperties Class 10559
TDBXProperties Members 10559
Create Constructor 10560
TDBXProperties.Destroy Destructor 10561
TDBXProperties Methods 10561
TDBXProperties Properties 10563
TDBXPropertiesItem Class 10564
TDBXPropertiesItem Members 10564
TDBXPropertiesItem.Create Constructor 10565

TDBXPropertiesItem.Destroy Destructor 10565
TDBXPropertiesItem Fields 10565
TDBXPropertyNames Class 10566
TDBXPropertyNames Members 10568
TDBXPropertyNames Constants 10568
TDBXPropertyNamesEx Class 10574
TDBXPropertyNamesEx Members 10575
TDBXPropertyNamesEx Constants 10575
TDBXReader Class 10576
TDBXReader Members 10577
TDBXReader.Create Constructor 10578
TDBXReader.Destroy Destructor 10578
TDBXReader Fields 10578
TDBXReader Methods 10579
TDBXReader Properties 10583
TDBXReader Events 10585
TDBXReaderByteReader Class 10585
TDBXReaderByteReader Members 10586
TDBXReaderByteReader.Create Constructor 10586
TDBXReaderByteReader Methods 10586
TDBXRow Class 10590
TDBXRow Members 10590
TDBXRow.Create Constructor 10591
TDBXRow Fields 10591
TDBXRow Methods 10592
TDBXRow Properties 10599
TDBXRowEx Class 10600
TDBXRowEx Members 10600
TDBXRowEx Methods 10601
TDBXStateItem Class 10604
TDBXStateItem Members 10604
TDBXStateItem Methods 10605
TDBXStateItemList Class 10605
TDBXStateItemList Members 10605
TDBXStateItemList.Destroy Destructor 10606
TDBXStateItemList Methods 10606
TDBXStreamReader Class 10607
TDBXStreamReader Members 10607
TDBXStreamReader Methods 10607

TDBXStreamValue Class 10608
TDBXStreamValue Members 10609
TDBXStreamValue.Create Constructor 10610
TDBXStreamValue.Destroy Destructor 10610
TDBXStreamValue Methods 10610
TDBXTimeStampValue Class 10612
TDBXTimeStampValue Members 10612
TDBXTimeStampValue Methods 10613
TDBXTimeValue Class 10614
TDBXTimeValue Members 10615
TDBXTimeValue Methods 10616
TDBXTraceFlags Class 10617
TDBXTraceFlags Members 10618
TDBXTraceFlags Constants 10618
TDBXTraceInfo Record 10622
TDBXTraceInfo Members 10622
TDBXTraceInfo Properties 10623
TDBXTransaction Class 10623
TDBXTransaction Members 10624
TDBXTransaction.Create Constructor 10624
TDBXTransaction.Destroy Destructor 10624
TDBXTransaction Fields 10625
TDBXTransaction Properties 10625
TDBXValue Class 10626
TDBXValue Members 10626
TDBXValue.Create Constructor 10627
TDBXValue.Destroy Destructor 10627
TDBXValue Methods 10628
TDBXValue Properties 10633
TDBXValueList Class 10634
TDBXValueList Members 10634
TDBXValueList.Destroy Destructor 10634
TDBXValueList Fields 10635
TDBXValueList Methods 10635
TDBXValueType Class 10636
TDBXValueType Members 10636
TDBXValueType.Create Constructor 10637
TDBXValueType Fields 10637
TDBXValueType Methods 10638

TDBXValueType Properties 10645
TDBXValueTypeEx Class 10649
TDBXValueTypeEx Members 10649
TDBXValueTypeEx Methods 10650
TDBXValueTypeFlags Class 10651
TDBXValueTypeFlags Members 10652
TDBXValueTypeFlags Constants 10652
TDBXValueTypeFlagsEx Class 10653
TDBXValueTypeFlagsEx Members 10654
TDBXValueTypeFlagsEx Constants 10654
TDBXWideCharsValue Class 10655
TDBXWideCharsValue Members 10655
TDBXWideCharsValue.Create Constructor 10656
TDBXWideCharsValue.Destroy Destructor 10656
TDBXWideCharsValue Methods 10657
TDBXWideStringValue Class 10658
TDBXWideStringValue Members 10658
TDBXWideStringValue.Create Constructor 10659
TDBXWideStringValue.Destroy Destructor 10660
TDBXWideStringValue Methods 10660
TDBXWritableValue Class 10661
TDBXWritableValue Members 10661
TDBXWritableValue.Create Constructor 10662
TDBXWritableValue.Destroy Destructor 10663
TDBXWritableValue Methods 10663

Structs, Records, Enums 10668
DBXCommon.TDBXDriverDef Record 10668

Types 10669
DBXCommon.TDBXChars Type 10669
DBXCommon.TDBXCreateCommandEvent Type 10669
DBXCommon.TDBXDate Type 10670
DBXCommon.TDBXDriverLoaderClass Type 10670
DBXCommon.TDBXErrorCode Type 10670
DBXCommon.TDBXErrorEvent Type 10671
DBXCommon.TDBXIsolation Type 10671
DBXCommon.TDBXParameterDirection Type 10671
DBXCommon.TDBXTime Type 10672
DBXCommon.TDBXTraceEvent Type 10672

DBXCommon.TDBXTraceFlag Type 10673
DBXCommon.TDBXType Type 10673
DBXCommon.TDBXValueArray Type 10673
DBXCommon.TWideStringArray Type 10674

Constants 10675

DBXCommon.DBXDefaultRowSetSize Constant 10675
DBXCommon.DBXVersion25 Constant 10676
DBXCommon.DBXVersion30 Constant 10676
DBXCommon.DBXVersion40 Constant 10676
DBXCommon.SAlreadyPrepared Constant 10677
DBXCommon.SConnectionFactoryInitFailed Constant 10677
DBXCommon.SDefaultErrorMessage Constant 10677
DBXCommon.SDllLoadError Constant 10678
DBXCommon.SDllProcLoadError Constant 10678
DBXCommon.SDriverLoadError Constant 10678
DBXCommon.SInvalidArgument Constant 10679
DBXCommon.SInvalidDelegationDepth Constant 10679
DBXCommon.SInvalidOrdinal Constant 10679
DBXCommon.SInvalidTransaction Constant 10680
DBXCommon.SInvalidTypeAccess Constant 10680
DBXCommon.SNotImplemented Constant 10680
DBXCommon.SReaderClosed Constant 10681
DBXCommon.SReaderNew Constant 10681
DBXCommon.SReadOnlyParameter Constant 10681
DBXCommon.SReadOnlyType Constant 10682
DBXCommon.SRequiredProperty Constant 10682
DBXCommon.SSetFactoryOnce Constant 10682
DBXCommon.SUnknownDriver Constant 10683
DBXCommon.TDBXConnectionFile Constant 10683
DBXCommon.TDBXDriverFile Constant 10683
DBXCommon.TDBXRegistryConnectionValue Constant 10684
DBXCommon.TDBXRegistryDriverValue Constant 10684
DBXCommon.TDBXRegistryKey Constant 10684

DBXMetaDataNames Namespace 10685

Classes 10687

TDBXCatalogsColumns Class 10688
TDBXCatalogsColumns Members 10688

TDBXCatalogsColumns Constants 10688
TDBXCatalogsIndex Class 10689
TDBXCatalogsIndex Members 10689
TDBXCatalogsIndex Constants 10690
TDBXColumnConstraintsColumns Class 10690
TDBXColumnConstraintsColumns Members 10691
TDBXColumnConstraintsColumns Constants 10691
TDBXColumnConstraintsIndex Class 10692
TDBXColumnConstraintsIndex Members 10693
TDBXColumnConstraintsIndex Constants 10693
TDBXColumnsColumns Class 10695
TDBXColumnsColumns Members 10696
TDBXColumnsColumns Constants 10696
TDBXColumnsIndex Class 10701
TDBXColumnsIndex Members 10702
TDBXColumnsIndex Constants 10703
TDBXDataTypesColumns Class 10707
TDBXDataTypesColumns Members 10709
TDBXDataTypesColumns Constants 10709
TDBXDataTypesIndex Class 10716
TDBXDataTypesIndex Members 10718
TDBXDataTypesIndex Constants 10718
TDBXForeignKeyColumnsColumns Class 10725
TDBXForeignKeyColumnsColumns Members 10726
TDBXForeignKeyColumnsColumns Constants 10726
TDBXForeignKeyColumnsIndex Class 10729
TDBXForeignKeyColumnsIndex Members 10730
TDBXForeignKeyColumnsIndex Constants 10731
TDBXForeignKeysColumns Class 10734
TDBXForeignKeysColumns Members 10734
TDBXForeignKeysColumns Constants 10735
TDBXForeignKeysIndex Class 10736
TDBXForeignKeysIndex Members 10736
TDBXForeignKeysIndex Constants 10737
TDBXIndexColumnsColumns Class 10738
TDBXIndexColumnsColumns Members 10739
TDBXIndexColumnsColumns Constants 10739
TDBXIndexColumnsIndex Class 10741
TDBXIndexColumnsIndex Members 10742

TDBXIndexColumnsIndex Constants 10742
TDBXIndexesColumns Class 10744
TDBXIndexesColumns Members 10745
TDBXIndexesColumns Constants 10745
TDBXIndexesIndex Class 10748
TDBXIndexesIndex Members 10748
TDBXIndexesIndex Constants 10749
TDBXMetaDataCollectionIndex Class 10751
TDBXMetaDataCollectionIndex Members 10752
TDBXMetaDataCollectionIndex Constants 10753
TDBXMetaDataCollectionName Class 10758
TDBXMetaDataCollectionName Members 10760
TDBXMetaDataCollectionName Constants 10760
TDBXPackageProcedureParametersColumns Class 10766
TDBXPackageProcedureParametersColumns Members 10767
TDBXPackageProcedureParametersColumns Constants 10768
TDBXPackageProcedureParametersIndex Class 10772
TDBXPackageProcedureParametersIndex Members 10773
TDBXPackageProcedureParametersIndex Constants 10774
TDBXPackageProceduresColumns Class 10778
TDBXPackageProceduresColumns Members 10779
TDBXPackageProceduresColumns Constants 10779
TDBXPackageProceduresIndex Class 10781
TDBXPackageProceduresIndex Members 10782
TDBXPackageProceduresIndex Constants 10782
TDBXPackagesColumns Class 10783
TDBXPackagesColumns Members 10784
TDBXPackagesColumns Constants 10784
TDBXPackagesIndex Class 10785
TDBXPackagesIndex Members 10786
TDBXPackagesIndex Constants 10786
TDBXPackageSourcesColumns Class 10787
TDBXPackageSourcesColumns Members 10788
TDBXPackageSourcesColumns Constants 10788
TDBXPackageSourcesIndex Class 10789
TDBXPackageSourcesIndex Members 10790
TDBXPackageSourcesIndex Constants 10790
TDBXProcedureParametersColumns Class 10792
TDBXProcedureParametersColumns Members 10793

TDBXProcedureParametersColumns Constants 10793
TDBXProcedureParametersIndex Class 10797
TDBXProcedureParametersIndex Members 10798
TDBXProcedureParametersIndex Constants 10799
TDBXProceduresColumns Class 10803
TDBXProceduresColumns Members 10804
TDBXProceduresColumns Constants 10804
TDBXProceduresIndex Class 10805
TDBXProceduresIndex Members 10806
TDBXProceduresIndex Constants 10806
TDBXProcedureSourcesColumns Class 10807
TDBXProcedureSourcesColumns Members 10808
TDBXProcedureSourcesColumns Constants 10808
TDBXProcedureSourcesIndex Class 10810
TDBXProcedureSourcesIndex Members 10811
TDBXProcedureSourcesIndex Constants 10811
TDBXReservedWordsColumns Class 10813
TDBXReservedWordsColumns Members 10814
TDBXReservedWordsColumns Constants 10814
TDBXReservedWordsIndex Class 10814
TDBXReservedWordsIndex Members 10815
TDBXReservedWordsIndex Constants 10815
TDBXRolesColumns Class 10816
TDBXRolesColumns Members 10816
TDBXRolesColumns Constants 10816
TDBXRolesIndex Class 10817
TDBXRolesIndex Members 10817
TDBXRolesIndex Constants 10818
TDBXSchemasColumns Class 10818
TDBXSchemasColumns Members 10819
TDBXSchemasColumns Constants 10819
TDBXSchemasIndex Class 10820
TDBXSchemasIndex Members 10820
TDBXSchemasIndex Constants 10821
TDBXSynonymsColumns Class 10822
TDBXSynonymsColumns Members 10822
TDBXSynonymsColumns Constants 10823
TDBXSynonymsIndex Class 10824
TDBXSynonymsIndex Members 10825

TDBXSynonymsIndex Constants 10825
TDBXTablesColumns Class 10827
TDBXTablesColumns Members 10828
TDBXTablesColumns Constants 10828
TDBXTablesIndex Class 10829
TDBXTablesIndex Members 10830
TDBXTablesIndex Constants 10830
TDBXUsersColumns Class 10832
TDBXUsersColumns Members 10832
TDBXUsersColumns Constants 10832
TDBXUsersIndex Class 10833
TDBXUsersIndex Members 10833
TDBXUsersIndex Constants 10834
TDBXViewsColumns Class 10834
TDBXViewsColumns Members 10835
TDBXViewsColumns Constants 10835
TDBXViewsIndex Class 10836
TDBXViewsIndex Members 10837
TDBXViewsIndex Constants 10837

DBXpressWeb Namespace 10841

Classes 10842
TSQLQueryTableProducer Class 10842
TSQLQueryTableProducer Members 10843
TSQLQueryTableProducer Methods 10847
TSQLQueryTableProducer Properties 10848
TSQLQueryTableProducer Events 10853

DdeMan Namespace 10857

Classes 10858
TDdeClientConv Class 10858
TDdeClientConv Members 10859
TDdeClientConv.Create Constructor 10862
TDdeClientConv.Destroy Destructor 10862
TDdeClientConv Methods 10863
TDdeClientConv Properties 10873
TDdeClientConv Events 10877
TDdeClientItem Class 10878

TDdeClientItem Members 10879
TDdeClientItem.Create Constructor 10881
TDdeClientItem.Destroy Destructor 10882
TDdeClientItem Methods 10882
TDdeClientItem Properties 10883
TDdeClientItem Events 10885
TDdeMgr Class 10886
TDdeMgr Members 10886
TDdeMgr.Create Constructor 10889
TDdeMgr.Destroy Destructor 10889
TDdeMgr Methods 10890
TDdeMgr Properties 10890
TDdeServerConv Class 10891
TDdeServerConv Members 10892
TDdeServerConv.Create Constructor 10895
TDdeServerConv.Destroy Destructor 10895
TDdeServerConv Methods 10896
TDdeServerConv Events 10897
TDdeServerItem Class 10898
TDdeServerItem Members 10899
TDdeServerItem.Create Constructor 10901
TDdeServerItem.Destroy Destructor 10902
TDdeServerItem Methods 10902
TDdeServerItem Properties 10906
TDdeServerItem Events 10908

Functions **10910**
DdeMan.GetPasteLinkInfo Function 10910

Structs, Records, Enums **10911**
DdeMan.TDataMode Enumeration 10911

Types **10912**
DdeMan.TMacroEvent Type 10912

Variables **10913**
DdeMan.ddeMgr Variable 10913

DesignEditors Namespace 10915

Classes **10917**
TBoolProperty Class 10918

TCharProperty Class 10918
TCharProperty Members 10918
TCharProperty Methods 10921
TClassProperty Class 10922
TClassProperty Members 10923
TClassProperty Methods 10925
TComponentEditor Class 10927
TComponentEditor Members 10928
TComponentEditor.Create Constructor 10929
TComponentEditor Methods 10930
TComponentEditor Properties 10934
TComponentGuidelines Class 10935
TComponentGuidelines Members 10936
TComponentGuidelines Methods 10937
TComponentGuidelines Properties 10939
TComponentNameProperty Class 10939
TComponentNameProperty Members 10940
TComponentNameProperty Methods 10942
TComponentProperty Class 10944
TComponentProperty Members 10944
TComponentProperty Methods 10947
TCustomModule Class 10952
TCustomModule Members 10952
TCustomModule.Create Constructor 10953
TCustomModule.Destroy Destructor 10954
TCustomModule Methods 10954
TCustomModule Properties 10956
TDateProperty Class 10957
TDateProperty Members 10958
TDateProperty Methods 10960
TDateTimeProperty Class 10962
TDateTimeProperty Members 10962
TDateTimeProperty Methods 10965
TDefaultEditor Class 10967
TDefaultEditor Members 10968
TDefaultEditor Methods 10969
TDesignerOleDropTarget Class 10970
TDesignerOleDropTarget Members 10971
TDesignerOleDropTarget.Create Constructor 10972

TDesignerOleDropTarget Methods 10972
TEditActionSelectionEditor Class 10973
TEditActionSelectionEditor Members 10974
TEditActionSelectionEditor Methods 10975
TEnumProperty Class 10978
TEnumProperty Members 10978
TEnumProperty Methods 10981
TFloatProperty Class 10983
TFloatProperty Members 10984
TFloatProperty Methods 10986
TInt64Property Class 10988
TInt64Property Members 10988
TInt64Property Methods 10991
TIntegerProperty Class 10993
TIntegerProperty Members 10994
TIntegerProperty Methods 10996
TInterfaceProperty Class 10998
TInterfaceProperty Members 10998
TInterfaceProperty Methods 11001
TMethodProperty Class 11004
TMethodProperty Members 11004
TMethodProperty Methods 11007
TNestedProperty Class 11011
TNestedProperty Members 11011
TNestedProperty.Create Constructor 11014
TNestedProperty.Destroy Destructor 11014
TOrdinalProperty Class 11015
TOrdinalProperty Members 11015
TOrdinalProperty Methods 11018
TPropertyEditor Class 11019
TPropertyEditor Members 11020
TPropertyEditor.Create Constructor 11022
TPropertyEditor.Destroy Destructor 11022
TPropertyEditor Methods 11023
TPropertyEditor Properties 11044
TSelectionEditor Class 11046
TSelectionEditor Members 11046
TSelectionEditor.Create Constructor 11047
TSelectionEditor Methods 11048

TSelectionEditor Properties 11049
TSetElementProperty Class 11049
TSetElementProperty Members 11050
TSetElementProperty.Create Constructor 11052
TSetElementProperty Methods 11053
TSetElementProperty Properties 11056
TSetProperty Class 11056
TSetProperty Members 11057
TSetProperty Methods 11059
TStringProperty Class 11061
TStringProperty Members 11062
TStringProperty Methods 11064
TTimeProperty Class 11066
TTimeProperty Members 11067
TTimeProperty Methods 11069
TVariantProperty Class 11071
TVariantProperty Members 11072
TVariantProperty Methods 11074
TWideCharProperty Class 11077
TWideCharProperty Members 11077
TWideCharProperty Methods 11080
TWideStringProperty Class 11082
TWideStringProperty Members 11082
TWideStringProperty Methods 11085

Interfaces 11088
IDesignGroupChange Interface 11088
IDesignGroupChange Members 11088
IDesignGroupChange Methods 11088

Functions 11090
DesignEditors.AncestorNameMatches Function 11090
DesignEditors.ClassInheritsFrom Function 11090
DesignEditors.FreeEditorGroup Function 11091
DesignEditors.GetComponentEditor Function 11091
DesignEditors.GetComponentProperties Function 11091
DesignEditors.GetSelectionEditors Function 11092
DesignEditors.GetSelectionEditors Function 11092
DesignEditors.GetSelectionEditors Function 11092
DesignEditors.GuidelinesFor Function 11093

DesignEditors.NewEditorGroup Function 11093
DesignEditors.NotifyGroupChange Function 11093
DesignEditors.PossibleStream Function 11094
DesignEditors.UnnotifyGroupChange Function 11094

Structs, Records, Enums 11095
DesignEditors.TInstProp Record 11095

Types 11096
DesignEditors.PInstPropList Type 11096
DesignEditors.TBit Type 11096
DesignEditors.TGetTopLevelComponentFunc Type 11097
DesignEditors.TGroupChangeProc Type 11097
DesignEditors.TInstPropList Type 11097

Variables 11098
DesignEditors.GetTopLevelComponentFunc Variable 11098

Constants 11099
DesignEditors.sClassNotApplicable Constant 11099
DesignEditors.sNotAvailable Constant 11099

DesignIntf Namespace 11101

Classes 11105
TBaseComponentEditor Class 11105
TBaseComponentEditor Members 11106
TBaseComponentEditor.Create Constructor 11107
TBaseComponentGuidelines Class 11107
TBaseComponentGuidelines Members 11108
TBaseComponentGuidelines.Create Constructor 11109
TBaseComponentGuidelines Methods 11109
TBaseComponentGuidelines Properties 11109
TBaseCustomModule Class 11110
TBaseCustomModule Members 11110
TBaseCustomModule.Create Constructor 11111
TBaseCustomModule Methods 11112
TBasePropertyEditor Class 11112
TBasePropertyEditor Members 11113
TBasePropertyEditor.Create Constructor 11114
TBasePropertyEditor Methods 11114
TBaseSelectionEditor Class 11116

TBaseSelectionEditor Members 11116
TBaseSelectionEditor.Create Constructor 11117
TClassWrapper Class 11117
TClassWrapper Members 11118
TClassWrapper.Create Constructor 11119
TClassWrapper Methods 11119
TDesignerSelections Class 11121
TDesignerSelections Members 11121
TDesignerSelections.Copy Constructor 11123
TDesignerSelections.Create Constructor 11123
TDesignerSelections.Destroy Destructor 11124
TDesignerSelections Methods 11124
TDesignerSelections Properties 11126
TDragTarget Class 11127
TDragTarget Members 11127
TDragTarget.Create Constructor 11128
TDragTarget Methods 11128
TDragTarget Properties 11129
TEventInfo Class 11129
TEventInfo Members 11129
Create Constructor 11131
TEventInfo Methods 11131

Interfaces 11134

IActivatable Interface 11134
IActivatable Members 11135
IActivatable Methods 11135
IClass Interface 11135
IClass Members 11136
IClass Methods 11136
IClass Properties 11137
IComponentEditor Interface 11138
IComponentEditor Members 11139
IComponentEditor Methods 11139
IComponentGuidelines Interface 11143
IComponentGuidelines Members 11143
IComponentGuidelines Methods 11144
IComponentGuidelines Properties 11144
ICustomModule Interface 11145
ICustomModule Members 11146

ICustomModule Methods 11146
IDefaultEditor Interface 11148
IDesigner Interface 11149
IDesigner Members 11149
IDesigner Methods 11151
IDesigner Properties 11151
IDesigner60 Interface 11151
IDesigner60 Members 11152
IDesigner60 Methods 11153
IDesigner60 Properties 11166
IDesigner70 Interface 11167
IDesigner70 Members 11167
IDesigner70 Methods 11168
IDesigner70 Properties 11169
IDesigner80 Interface 11169
IDesigner80 Members 11170
IDesigner80 Methods 11171
IDesignerSelections Interface 11172
IDesignerSelections Members 11172
IDesignerSelections Methods 11173
IDesignerSelections Properties 11175
IDesignNotification Interface 11176
IDesignNotification Members 11176
IDesignNotification Methods 11176
IDesignObject Interface 11178
IDesignObject Members 11178
IDesignObject Methods 11179
IDesignObject Properties 11181
IDesignPersistent Interface 11183
IDesignPersistent Members 11183
IDesignPersistent Methods 11183
IDesignPersistent Properties 11184
IDesignWindow Interface 11184
IDesignWindow Members 11185
IDesignWindow Methods 11185
IEditHandler Interface 11186
IEditHandler Members 11186
IEditHandler Methods 11186
IEventInfo Interface 11187

IEventInfo Members 11187
IEventInfo Methods 11187
IHostForm Interface 11189
IHostForm Members 11189
IHostForm Methods 11190
IHostForm Properties 11195
IImplementation Interface 11196
IImplementation Members 11196
IImplementation Methods 11196
IMethodProperty Interface 11197
IProperty Interface 11197
IProperty Members 11198
IProperty Methods 11198
IProperty70 Interface 11205
IProperty70 Members 11205
IProperty70 Methods 11206
IProperty70 Properties 11206
IProperty80 Interface 11207
IProperty80 Members 11207
IProperty80 Methods 11208
IPropertyControl Interface 11208
IPropertyControl Members 11208
IPropertyControl Methods 11208
IPropertyDescription Interface 11209
IPropertyDescription Members 11209
IPropertyDescription Methods 11209
IPropertyHost Interface 11210
IPropertyHost Members 11210
IPropertyHost Methods 11210
IPropertyKind Interface 11211
IPropertyKind Members 11211
IPropertyKind Methods 11212
IPropertyKind Properties 11212
IReferenceProperty Interface 11212
IReferenceProperty Members 11213
IReferenceProperty Methods 11213
ISelectionEditor Interface 11213
ISelectionEditor Members 11214
ISelectionEditor Methods 11214

ISelectionEditorList Interface 11215
ISelectionEditorList Members 11216
ISelectionEditorList Methods 11216
ISelectionEditorList Properties 11217
ISelectionPropertyFilter Interface 11217
ISelectionPropertyFilter Members 11218
ISelectionPropertyFilter Methods 11218
IShowReferenceProperty Interface 11218
IShowReferenceProperty Members 11219
IShowReferenceProperty Methods 11219
IWideProperty Interface 11219
IWideProperty Members 11219
IWideProperty Methods 11220
IWideProperty10 Interface 11221
IWideProperty10 Members 11221
IWideProperty10 Methods 11221
IWidePropertyDescription Interface 11222
IWidePropertyDescription Members 11222
IWidePropertyDescription Methods 11222

Functions 11224

DesignIntf.CreateSelectionList Function 11224
DesignIntf.EnableDemandLoadReport Function 11225
DesignIntf.ForceDemandLoadState Function 11225
DesignIntf.PersistentToDesignObject Function 11225
DesignIntf.RegisterComponentEditor Function 11226
DesignIntf.RegisterComponentGuidelines Function 11226
DesignIntf.RegisterCustomModule Function 11226
DesignIntf.RegisterDesignDragObject Function 11227
DesignIntf.RegisterDesignNotification Function 11227
DesignIntf.RegisterDragTarget Function 11227
DesignIntf.RegisterIDropTarget Function 11228
DesignIntf.RegisterPropertiesInCategory Function 11228
DesignIntf.RegisterPropertiesInCategory Function 11228
DesignIntf.RegisterPropertiesInCategory Function 11229
DesignIntf.RegisterPropertyEditor Function 11229
DesignIntf.RegisterPropertyInCategory Function 11229
DesignIntf.RegisterPropertyInCategory Function 11230
DesignIntf.RegisterPropertyInCategory Function 11230
DesignIntf.RegisterPropertyInCategory Function 11230

DesignIntf.RegisterPropertyMapper Function 11231
DesignIntf.RegisterSelectionEditor Function 11231
DesignIntf.SetPropertyEditorGroup Function 11231
DesignIntf.UnlistPublishedProperty Function 11232
DesignIntf.UnregisterDesignNotification Function 11232

Structs, Records, Enums 11233

DesignIntf.TCustomModuleAttribute Enumeration 11233
DesignIntf.TDemandLoadState Enumeration 11233
DesignIntf.TDesignerGuideType Enumeration 11234
DesignIntf.TEditAction Enumeration 11234
DesignIntf.TEditStates Enumeration 11235
DesignIntf.TPropertyAttribute Enumeration 11236
DesignIntf.TPropKind Enumeration 11237

Types 11238

DesignIntf.IDesignEditQuery Type 11238
DesignIntf.TComponentEditorClass Type 11239
DesignIntf.TComponentGuidelinesClass Type 11239
DesignIntf.TCustomModuleAttributes Type 11239
DesignIntf.TCustomModuleClass Type 11240
DesignIntf.TDragTargetClass Type 11240
DesignIntf.TEditState Type 11240
DesignIntf.TEnableDemandLoadReportProc Type 11241
DesignIntf.TForceDemandLoadStateProc Type 11241
DesignIntf.TGetDesignerEvent Type 11241
DesignIntf.TGetPropProc Type 11242
DesignIntf.TGetWideStrProc Type 11242
DesignIntf.TPropertyAttributes Type 11242
DesignIntf.TPropertyEditorClass Type 11243
DesignIntf.TPropertyEditorFilterFunc Type 11243
DesignIntf.TPropertyMapperFunc Type 11243
DesignIntf.TRegisterComponentEditorProc Type 11244
DesignIntf.TRegisterComponentGuidelines Type 11244
DesignIntf.TRegisterCustomModuleProc Type 11244
DesignIntf.TRegisterDesignDragObject Type 11245
DesignIntf.TRegisterDesignNotification Type 11245
DesignIntf.TRegisterDragTargetProc Type 11245
DesignIntf.TRegisterIDropTarget Type 11246
DesignIntf.TRegisterPropertyEditorProc Type 11246

DesignIntf.TRegisterPropertyInCategoryProc Type 11246
DesignIntf.TRegisterPropertyMapperProc Type 11247
DesignIntf.TRegisterSelectionEditorProc Type 11247
DesignIntf.TSelectionEditorClass Type 11247
DesignIntf.TSetPropertyEditorGroupProc Type 11248

Variables **11249**
DesignIntf.EnableDemandLoadReportProc Variable 11249
DesignIntf.ForceDemandLoadStateProc Variable 11249
DesignIntf.GReferenceExpandable Variable 11250
DesignIntf.GShowReadOnlyProps Variable 11250
DesignIntf.RegisterComponentEditorProc Variable 11250
DesignIntf.RegisterComponentGuidelinesProc Variable 11251
DesignIntf.RegisterCustomModuleProc Variable 11251
DesignIntf.RegisterDesignDragObjectProc Variable 11251
DesignIntf.RegisterDesignNotificationProc Variable 11252
DesignIntf.RegisterDragTargetProc Variable 11252
DesignIntf.RegisterIDropTargetProc Variable 11252
DesignIntf.RegisterPropertyEditorProc Variable 11253
DesignIntf.RegisterPropertyInCategoryProc Variable 11253
DesignIntf.RegisterPropertyMapperProc Variable 11253
DesignIntf.RegisterSelectionEditorProc Variable 11254
DesignIntf.SetPropertyEditorGroupProc Variable 11254
DesignIntf.UnregisterDesignNotificationProc Variable 11254

Constants **11255**
DesignIntf.MaxIdentLength Constant 11255
DesignIntf.sActionCategoryName Constant 11256
DesignIntf.sAppearanceCategoryName Constant 11256
DesignIntf.sBehaviorCategoryName Constant 11256
DesignIntf.sDatabaseCategoryName Constant 11257
DesignIntf.sDataCategoryName Constant 11257
DesignIntf.sDesignCategoryName Constant 11257
DesignIntf.sDragNDropCategoryName Constant 11258
DesignIntf.sFocusCategoryName Constant 11258
DesignIntf.sHelpCategoryName Constant 11258
DesignIntf.sInputCategoryName Constant 11259
DesignIntf.sInvalidFilter Constant 11259
DesignIntf.sKeyCategoryName Constant 11259
DesignIntf.sLayoutCategoryName Constant 11260

DesignIntf.sLegacyCategoryName Constant 11260
DesignIntf.sLinkageCategoryName Constant 11260
DesignIntf.sLocaleCategoryName Constant 11261
DesignIntf.sLocalizableCategoryName Constant 11261
DesignIntf.sMiscellaneousCategoryName Constant 11261
DesignIntf.sMouseCategoryName Constant 11262
DesignIntf.sPropertyChangeCategoryName Constant 11262
DesignIntf.sVisualCategoryName Constant 11262
DesignIntf.sWindowStyleName Constant 11263

Dialogs Namespace 11265

Classes 11269
EPlatformVersionException Class 11269
TColorDialog Class 11270
TColorDialog Members 11270
TColorDialog.Create Constructor 11273
TColorDialog.Destroy Destructor 11274
TColorDialog Methods 11274
TColorDialog Properties 11275
TCommonDialog Class 11277
TCommonDialog Members 11278
TCommonDialog.Create Constructor 11281
TCommonDialog.Destroy Destructor 11281
TCommonDialog Methods 11282
TCommonDialog Properties 11285
TCommonDialog Events 11287
TCustomFileDialog Class 11288
TCustomFileDialog Members 11289
TCustomFileDialog.Create Constructor 11292
TCustomFileDialog.Destroy Destructor 11292
TCustomFileDialog Methods 11293
TCustomFileDialog Properties 11299
TCustomFileDialog Events 11305
TCustomFileOpenDialog Class 11308
TCustomFileOpenDialog Members 11309
TCustomFileOpenDialog Methods 11312
TCustomFileSaveDialog Class 11314
TCustomFileSaveDialog Members 11314

TCustomFileSaveDialog Methods 11318
TCustomTaskDialog Class 11318
TCustomTaskDialog Members 11319
TCustomTaskDialog.Create Constructor 11322
TCustomTaskDialog.Destroy Destructor 11322
TCustomTaskDialog Methods 11323
TCustomTaskDialog Properties 11327
TCustomTaskDialog Events 11334
TFavoriteLinkItem Class 11337
TFavoriteLinkItem Members 11337
TFavoriteLinkItem Methods 11339
TFavoriteLinkItem Properties 11339
TFavoriteLinkItems Class 11340
TFavoriteLinkItems Members 11340
TFavoriteLinkItems Methods 11342
TFavoriteLinkItems Properties 11343
TFavoriteLinkItemsEnumerator Class 11343
TFavoriteLinkItemsEnumerator Members 11344
TFavoriteLinkItemsEnumerator.Create Constructor 11344
TFavoriteLinkItemsEnumerator Methods 11344
TFavoriteLinkItemsEnumerator Properties 11345
TFileOpenDialog Class 11346
TFileOpenDialog Members 11346
TFileOpenDialog Properties 11350
TFileOpenDialog Events 11354
TFileSaveDialog Class 11357
TFileSaveDialog Members 11357
TFileSaveDialog Properties 11361
TFileSaveDialog Events 11365
TFileTypeItem Class 11368
TFileTypeItem Members 11368
TFileTypeItem.Create Constructor 11370
TFileTypeItem.Destroy Destructor 11370
TFileTypeItem Methods 11370
TFileTypeItem Properties 11371
TFileTypeItems Class 11372
TFileTypeItems Members 11373
TFileTypeItems Methods 11375
TFileTypeItems Properties 11376

TFindDialog Class 11376
TFindDialog Members 11377
TFindDialog.Create Constructor 11380
TFindDialog.Destroy Destructor 11380
TFindDialog Methods 11380
TFindDialog Properties 11384
TFindDialog Events 11392
TFontDialog Class 11394
TFontDialog Members 11395
TFontDialog.Create Constructor 11398
TFontDialog.Destroy Destructor 11398
TFontDialog Methods 11398
TFontDialog Properties 11401
TFontDialog Events 11406
TOpenDialog Class 11407
TOpenDialog Members 11408
TOpenDialog.Create Constructor 11411
TOpenDialog.Destroy Destructor 11412
TOpenDialog Methods 11412
TOpenDialog Properties 11420
TOpenDialog Events 11439
TPageSetupDialog Class 11441
TPageSetupDialog Members 11442
TPageSetupDialog.Create Constructor 11446
TPageSetupDialog Methods 11446
TPageSetupDialog Properties 11447
TPageSetupDialog Events 11455
TPrintDialog Class 11458
TPrintDialog Members 11458
TPrintDialog Methods 11462
TPrintDialog Properties 11464
TPrinterSetupDialog Class 11478
TPrinterSetupDialog Members 11479
TPrinterSetupDialog Methods 11482
TReplaceDialog Class 11482
TReplaceDialog Members 11483
TReplaceDialog.Create Constructor 11486
TReplaceDialog Properties 11487
TReplaceDialog Events 11488

TSaveDialog Class 11489
TSaveDialog Members 11490
TSaveDialog Methods 11494
TTaskDialog Class 11494
TTaskDialog Members 11495
TTaskDialog Properties 11498
TTaskDialog Events 11504
TTaskDialogBaseButtonItem Class 11507
TTaskDialogBaseButtonItem Members 11507
TTaskDialogBaseButtonItem.Create Constructor 11509
TTaskDialogBaseButtonItem.Destroy Destructor 11509
TTaskDialogBaseButtonItem Methods 11509
TTaskDialogBaseButtonItem Properties 11511
TTaskDialogButtonItem Class 11513
TTaskDialogButtonItem Members 11513
TTaskDialogButtonItem.Create Constructor 11515
TTaskDialogButtonItem Methods 11515
TTaskDialogButtonItem Properties 11516
TTaskDialogButtons Class 11517
TTaskDialogButtons Members 11517
TTaskDialogButtons Methods 11519
TTaskDialogButtons Properties 11520
TTaskDialogButtonsEnumerator Class 11521
TTaskDialogButtonsEnumerator Members 11521
TTaskDialogButtonsEnumerator.Create Constructor 11522
TTaskDialogButtonsEnumerator Methods 11522
TTaskDialogButtonsEnumerator Properties 11523
TTaskDialogProgressBar Class 11523
TTaskDialogProgressBar Members 11524
TTaskDialogProgressBar.Create Constructor 11525
TTaskDialogProgressBar Methods 11525
TTaskDialogProgressBar Properties 11525
TTaskDialogRadioButtonItem Class 11527
TTaskDialogRadioButtonItem Members 11527
TTaskDialogRadioButtonItem.Create Constructor 11529
TTaskDialogRadioButtonItem Methods 11529

Functions 11531
Dialogs.CreateMessageDialog Function 11531
Dialogs.InputBox Function 11532

Dialogs.InputQuery Function 11533
Dialogs.MessageDlg Function 11535
Dialogs.MessageDlgPos Function 11537
Dialogs.MessageDlgPosHelp Function 11539
Dialogs.PromptForFileName Function 11540
Dialogs.ShowMessage Function 11541
Dialogs.ShowMessageFmt Function 11542
Dialogs.ShowMessagePos Function 11543
Dialogs.TaskMessageDlg Function 11543
Dialogs.TaskMessageDlgPos Function 11545
Dialogs.TaskMessageDlgPosHelp Function 11546

Structs, Records, Enums 11548

Dialogs.TColorDialogOption Enumeration 11548
Dialogs.TFileDialogOption Enumeration 11549
Dialogs.TFileDialogOverwriteResponse Enumeration 11551
Dialogs.TFileDialogShareViolationResponse Enumeration 11552
Dialogs.TFileEditStyle Enumeration 11552
Dialogs.TFindOption Enumeration 11553
Dialogs.TFontDialogDevice Enumeration 11554
Dialogs.TFontDialogOption Enumeration 11555
Dialogs.TMsgDlgBtn Enumeration 11556
Dialogs.TMsgDlgType Enumeration 11558
Dialogs.TOpenOption Enumeration 11559
Dialogs.TOpenOptionEx Enumeration 11561
Dialogs.TPageMeasureUnits Enumeration 11561
Dialogs.TPageSetupDialogOption Enumeration 11562
Dialogs.TPageType Enumeration 11562
Dialogs.TPrintDialogOption Enumeration 11563
Dialogs.TPrinterKind Enumeration 11564
Dialogs.TPrintRange Enumeration 11564
Dialogs.TProgressBarState Enumeration 11565
Dialogs.TTaskDialogCommonButton Enumeration 11565
Dialogs.TTaskDialogFlag Enumeration 11566

Types 11568

Dialogs.TColorDialogOptions Type 11568
Dialogs.TCustomColors Type 11569
Dialogs.TFDApplyEvent Type 11569
Dialogs.TFileDialogCloseEvent Type 11570

Dialogs.TFileDialogFolderChangingEvent Type 11570
Dialogs.TFileDialogOptions Type 11571
Dialogs.TFileDialogOverwriteEvent Type 11572
Dialogs.TFileDialogShareViolationEvent Type 11572
Dialogs.TFindOptions Type 11573
Dialogs.TFindReplaceFunc Type 11574
Dialogs.TFontDialogOptions Type 11574
Dialogs.TIncludeItemEvent Type 11575
Dialogs.TMsgDlgButtons Type 11575
Dialogs.TOFNotifyEx Type 11576
Dialogs.TOpenOptions Type 11576
Dialogs.TOpenOptionsEx Type 11578
Dialogs.TPageSetupBeforePaintEvent Type 11578
Dialogs.TPageSetupDialogOptions Type 11579
Dialogs.TPaintPageEvent Type 11579
Dialogs.TPrintDialogOptions Type 11580
Dialogs.TPrinterOrientation Type 11581
Dialogs.TTaskDialogButtonList Type 11581
Dialogs.TTaskDialogCommonButtons Type 11582
Dialogs.TTaskDialogFlags Type 11582
Dialogs.TTaskDialogIcon Type 11583
Dialogs.TTaskDlgClickEvent Type 11583
Dialogs.TTaskDlgTimerEvent Type 11584

Variables **11585**
Dialogs.ForceCurrentDirectory Variable 11585
Dialogs.UseLatestCommonDialogs Variable 11585

Constants **11586**
Dialogs.MaxCustomColors Constant 11586
Dialogs.mbAbortIgnore Constant 11586
Dialogs.mbAbortRetryIgnore Constant 11587
Dialogs.mbOKCancel Constant 11588
Dialogs.mbYesAllNoAllCancel Constant 11588
Dialogs.mbYesNo Constant 11589
Dialogs.mbYesNoCancel Constant 11589
Dialogs.tdiError Constant 11590
Dialogs.tdiInformation Constant 11590
Dialogs.tdiNone Constant 11591
Dialogs.tdiShield Constant 11591

Dialogs.tdiWarning Constant 11591

DockTabSet Namespace 11593

Classes 11594

ETabDockException Class 11594

TCaptionedTabDockTree Class 11594

TCaptionedTabDockTree Members 11595

TCaptionedTabDockTree Methods 11597

TDockClientInfo Class 11598

TDockClientInfo Members 11598

TDockClientInfo.Create Constructor 11601

TDockClientInfo.Destroy Destructor 11601

TDockClientInfo Methods 11602

TDockClientInfo Properties 11603

TDockTabSet Class 11604

TDockTabSet Members 11604

TDockTabSet.Create Constructor 11616

TDockTabSet.Destroy Destructor 11617

TDockTabSet Methods 11617

TDockTabSet Properties 11622

TDockTabSet Events 11624

TTabDockPanel Class 11625

TTabDockPanel Members 11626

TTabDockPanel.Create Constructor 11638

TTabDockPanel.Destroy Destructor 11638

TTabDockPanel Methods 11639

TTabDockPanel Properties 11641

Interfaces 11643

IAlternateDockHost Interface 11643

IAlternateDockHost Members 11643

IAlternateDockHost Methods 11643

DRIntf Namespace 11645

Functions 11647

DRIntf.AssociateAttr Function 11647

DRIntf.CreateField Function 11648

DRIntf.CurrentDictionary Function 11649

DRIntf.DictionaryActive Function 11649
DRIntf.DictionaryDeactivate Function 11649
DRIntf.FindAttrID Function 11650
DRIntf.FindDatabaseID Function 11650
DRIntf.FindFieldID Function 11651
DRIntf.FindTableID Function 11652
DRIntf.GetAttrID Function 11652
DRIntf.GetAttrName Function 11653
DRIntf.GetAttrNames Function 11653
DRIntf.GetControlClass Function 11654
DRIntf.HasConstraints Function 11654
DRIntf.IsNullID Function 11655
DRIntf.NewAttr Function 11656
DRIntf.QualifyTableName Function 11656
DRIntf.QualifyTableNameByName Function 11657
DRIntf.UnassociateAttr Function 11657
DRIntf.UpdateAttr Function 11658
DRIntf.UpdateConstraints Function 11659
DRIntf.UpdateDataset Function 11659
DRIntf.UpdateField Function 11660

Structs, Records, Enums **11661**
DRIntf.TID Record 11661

Types **11662**
DRIntf.TAttrID Type 11662
DRIntf.TDatabaseID Type 11662
DRIntf.TFieldID Type 11663
DRIntf.TTableID Type 11663

Constants **11664**
DRIntf.NullAttrID Constant 11664
DRIntf.NullDatabaseID Constant 11664
DRIntf.NullFieldID Constant 11665
DRIntf.NullID Constant 11665
DRIntf.NullTableID Constant 11665

DSIntf Namespace 11667

Interfaces **11673**
IDSBase Interface 11673

IDSBase Members 11673
IDSBase Methods 11674
IDSCursor Interface 11682
IDSCursor Members 11683
IDSCursor Methods 11684
IDSWriter Interface 11697
IDSWriter Members 11698
IDSWriter Methods 11698

Functions 11702
DSIntf.CreateDbClientObject Function 11702
DSIntf.DataPacketSize Function 11702
DSIntf.DataPacketToVariant Function 11703
DSIntf.FreeDataPacket Function 11703
DSIntf.RegisterMidasLib Function 11703
DSIntf.StringToVariantArray Function 11704
DSIntf.VariantArrayToString Function 11704
DSIntf.VarToDataPacket Function 11704

Structs, Records, Enums 11705
DSIntf.AGGSTATE Enumeration 11705
DSIntf.AGGVALUE Record 11706
DSIntf.CURProp Enumeration 11706
DSIntf.DBSearchCond Enumeration 11707
DSIntf.DSBOOKMRK Record 11707
DSIntf.DSFLDDesc Record 11708
DSIntf.DSIDXDesc Record 11708
DSIntf.DSProp Enumeration 11709
DSIntf.DSProps Record 11710
DSIntf.GROUPSTATE Enumeration 11711
DSIntf.TDSDataPacketFldDesc Record 11711
DSIntf.TPcktAttrArea Enumeration 11712
DSIntf.TPcktFldStatus Enumeration 11712

Types 11713
DSIntf.DBResult Type 11713
DSIntf.DSAttr Type 11714
DSIntf.dsCBRType Type 11714
DSIntf.DSKEY Type 11714
DSIntf.DSKEYBOOL Type 11715
DSIntf.hDSAggregate Type 11715

DSIntf.hDSFilter Type 11715
DSIntf.MIDASNAME Type 11716
DSIntf.MIDASPATH Type 11716
DSIntf.pDSAttr Type 11716
DSIntf.pDSBOOKMRK Type 11717
DSIntf.pdsCBRType Type 11717
DSIntf.PDSDataPacketFldDesc Type 11717
DSIntf.pDSFLDDesc Type 11718
DSIntf.pDSIDXDesc Type 11718
DSIntf.pDSProps Type 11718
DSIntf.pfCHANGECallBack Type 11719
DSIntf.pfDSCalcField Type 11719
DSIntf.pfDSFilter Type 11719
DSIntf.pfDSReconcile Type 11720
DSIntf.pfDSReconcile_MD Type 11720
DSIntf.pfFLDComp Type 11720
DSIntf.phDSAggregate Type 11721
DSIntf.phDSFilter Type 11721
DSIntf.PLongBool Type 11721
DSIntf.pSAVEPOINT Type 11722
DSIntf.SAVEPOINT Type 11722
DSIntf.TDataPacket Type 11722

Constants **11723**

DSIntf.BLANK_NOTCHANGED Constant 11727
DSIntf.BLANK_NULL Constant 11727
DSIntf.BlobTypeMap Constant 11727
DSIntf.CLSID_DSBase Constant 11728
DSIntf.CLSID_DSCursor Constant 11728
DSIntf.CLSID_DSWriter Constant 11728
DSIntf.DataTypeMap Constant 11729
DSIntf.DBERR_ABORTED Constant 11729
DSIntf.DBERR_BLOBNOTFETCHED Constant 11729
DSIntf.DBERR_BOF Constant 11730
DSIntf.DBERR_CANCELLED Constant 11730
DSIntf.DBERR_CANNOTAPPEND Constant 11730
DSIntf.DBERR_DATAPACKETMISMATCH Constant 11731
DSIntf.DBERR_DELTAISEMPTY Constant 11731
DSIntf.DBERR_DETAILSNOTFETCHED Constant 11731
DSIntf.DBERR_EOF Constant 11732

DSIntf.DBERR_FLYAWAY_WRONGORDER Constant 11732
DSIntf.DBERR_LINKFIELDSNOTUNIQUE Constant 11732
DSIntf.DBERR_NEWERVERSIONREQ Constant 11733
DSIntf.DBERR_NOCASCADEDUPDATES Constant 11733
DSIntf.DBERR_NOMASTERRECORD Constant 11733
DSIntf.DBERR_NOMETADATA Constant 11734
DSIntf.DBERR_NONE Constant 11734
DSIntf.DBERR_NOSUCHINDEX Constant 11734
DSIntf.DBERR_NOTHINGTOUNDO Constant 11735
DSIntf.dsArrayFldType Constant 11735
DSIntf.dsCASCADEDELETES Constant 11735
DSIntf.dsCASCADEUPDATES Constant 11736
DSIntf.dscbrABORT Constant 11736
DSIntf.dscbrAPPLY Constant 11736
DSIntf.dscbrCANCEL Constant 11737
DSIntf.dscbrCORRECT Constant 11737
DSIntf.dscbrIGNORE Constant 11737
DSIntf.dscbrMERGE Constant 11738
DSIntf.dscbrREFRESH Constant 11738
DSIntf.dscbrSKIP Constant 11738
DSIntf.dsCompArrayFldType Constant 11739
DSIntf.dsDELAYEDBIT Constant 11739
DSIntf.dsDetUpd Constant 11739
DSIntf.dsEmbeddedFldType Constant 11740
DSIntf.dsfldADT Constant 11740
DSIntf.dsfldARRAY Constant 11740
DSIntf.dsfldBCD Constant 11741
DSIntf.dsfldBOOL Constant 11741
DSIntf.dsfldBYTES Constant 11741
DSIntf.dsfldDATE Constant 11742
DSIntf.dsfldDATETIME Constant 11742
DSIntf.dsfldEMBEDDEDTBL Constant 11742
DSIntf.dsfldFLOATIEEE Constant 11743
DSIntf.dsfldFMTBCD Constant 11743
DSIntf.dsfldINT Constant 11743
DSIntf.dsfldREF Constant 11744
DSIntf.dsfldTIME Constant 11744
DSIntf.dsfldTIMESTAMP Constant 11744
DSIntf.dsfldUINT Constant 11745

DSIntf.dsfldUNICODE Constant 11745
DSIntf.dsfldUNKNOWN Constant 11745
DSIntf.dsfldZSTRING Constant 11746
DSIntf.dsIncInDelta Constant 11746
DSIntf.dsIsNotVisible Constant 11746
DSIntf.dsIsVisible Constant 11747
DSIntf.dskeyCASEINSENSITIVE Constant 11747
DSIntf.dskeyDESCENDING Constant 11747
DSIntf.dsPseudoFldType Constant 11748
DSIntf.dsRecDeleted Constant 11748
DSIntf.dsRecModified Constant 11748
DSIntf.dsRecNew Constant 11749
DSIntf.dsRecOrg Constant 11749
DSIntf.dsRecUnmodified Constant 11749
DSIntf.dsSizeBitsLen Constant 11750
DSIntf.dsSizeBitsMask Constant 11750
DSIntf.dsTypeBitsMask Constant 11750
DSIntf.dsUnused Constant 11751
DSIntf.dsVaryingFldType Constant 11751
DSIntf.ERRBASE_ALC Constant 11751
DSIntf.ERRCAT_ALC Constant 11752
DSIntf.ERRCODE_ABORTED Constant 11752
DSIntf.ERRCODE_BLOBNOTFETCHED Constant 11752
DSIntf.ERRCODE_CANCELLED Constant 11753
DSIntf.ERRCODE_CANNOTAPPEND Constant 11753
DSIntf.ERRCODE_DATAPACKETMISMATCH Constant 11753
DSIntf.ERRCODE_DELTAISEMPTY Constant 11754
DSIntf.ERRCODE_DETAILSNOTFETCHED Constant 11754
DSIntf.ERRCODE_FLYAWAY_WRONGORDER Constant 11754
DSIntf.ERRCODE_LINKFIELDSNOTUNIQUE Constant 11755
DSIntf.ERRCODE_NEWERVERSIONREQ Constant 11755
DSIntf.ERRCODE_NOCASCADEDUPDATES Constant 11755
DSIntf.ERRCODE_NOMASTERRECORD Constant 11756
DSIntf.ERRCODE_NOMETADATA Constant 11756
DSIntf.ERRCODE_NOTHINGTOUNDO Constant 11756
DSIntf.FieldTypeMap Constant 11757
DSIntf.fldADT Constant 11757
DSIntf.fldARRAY Constant 11757
DSIntf.fldAttrHIDDEN Constant 11758

DSIntf.fldAttrLINK Constant 11758
DSIntf.fldAttrREADONLY Constant 11758
DSIntf.fldAttrREQUIRED Constant 11759
DSIntf.fldBCD Constant 11759
DSIntf.fldBLOB Constant 11759
DSIntf.fldBOOL Constant 11760
DSIntf.fldBYTES Constant 11760
DSIntf.fldCURSOR Constant 11760
DSIntf.fldDATE Constant 11761
DSIntf.fldDATETIME Constant 11761
DSIntf.fldFLOAT Constant 11761
DSIntf.fldFLOATIEEE Constant 11762
DSIntf.fldFMTBCD Constant 11762
DSIntf.fldINT16 Constant 11762
DSIntf.fldINT32 Constant 11763
DSIntf.fldINT64 Constant 11763
DSIntf.fldLOCKINFO Constant 11763
DSIntf.fldREF Constant 11764
DSIntf.fldstACCOLEOBJ Constant 11764
DSIntf.fldstAUTOINC Constant 11764
DSIntf.fldstBFILE Constant 11765
DSIntf.fldstBINARY Constant 11765
DSIntf.fldstDBSOLEOBJ Constant 11765
DSIntf.fldstFIXED Constant 11766
DSIntf.fldstFMTMEMO Constant 11766
DSIntf.fldstGRAPHIC Constant 11766
DSIntf.fldstGUID Constant 11767
DSIntf.fldstHBINARY Constant 11767
DSIntf.fldstHMEMO Constant 11767
DSIntf.fldstMEMO Constant 11768
DSIntf.fldstMONEY Constant 11768
DSIntf.fldstOLEOBJ Constant 11768
DSIntf.fldstORAINTERVAL Constant 11769
DSIntf.fldstORATIMESTAMP Constant 11769
DSIntf.fldstPASSWORD Constant 11769
DSIntf.fldstReference Constant 11770
DSIntf.fldstTYPEDBINARY Constant 11770
DSIntf.fldstWIDEMEMO Constant 11770
DSIntf.FldSubTypeMap Constant 11771

DSIntf.fldTABLE Constant 11771
DSIntf.fldTIME Constant 11771
DSIntf.fldTIMESTAMP Constant 11772
DSIntf.fldUINT16 Constant 11772
DSIntf.fldUINT32 Constant 11772
DSIntf.fldUINT64 Constant 11773
DSIntf.fldUNICODE Constant 11773
DSIntf.fldUNKNOWN Constant 11773
DSIntf.fldVARBYTES Constant 11774
DSIntf.fldWIDESTRING Constant 11774
DSIntf.fldZSTRING Constant 11774
DSIntf.MAXKEYFIELDS Constant 11775
DSIntf.MAXLOGFLDTYPES Constant 11775
DSIntf.mdALLOWLINKCHANGE Constant 11775
DSIntf.mdCASCADEDEL Constant 11776
DSIntf.mdCASCADEMOD Constant 11776
DSIntf.PACKETVERSION_1 Constant 11776
DSIntf.PACKETVERSION_2 Constant 11777
DSIntf.PACKETVERSION_3 Constant 11777
DSIntf.szAUTOINCVALUE Constant 11777
DSIntf.szBDEDEFCL Constant 11778
DSIntf.szBDEDEFX Constant 11778
DSIntf.szBDEDOMCL Constant 11778
DSIntf.szBDEDOMX Constant 11779
DSIntf.szBDERECCL Constant 11779
DSIntf.szBDERECX Constant 11779
DSIntf.szCHANGE_LOG Constant 11780
DSIntf.szCHANGEINDEX Constant 11780
DSIntf.szCONSTRAINTS Constant 11780
DSIntf.szDATASET_CONTEXT Constant 11781
DSIntf.szDATASET_DELTA Constant 11781
DSIntf.szDECIMALS Constant 11781
DSIntf.szDEFAULT_ORDER Constant 11782
DSIntf.szDISABLE_DELETES Constant 11782
DSIntf.szDISABLE_EDITS Constant 11782
DSIntf.szDISABLE_INSERTS Constant 11783
DSIntf.szELEMENTS Constant 11783
DSIntf.szFIELDNAME Constant 11783
DSIntf.szFIELDPROPS Constant 11784

DSIntf.szLCID Constant 11784
DSIntf.szMAXVALUE Constant 11784
DSIntf.szMD_FIELDLINKS Constant 11785
DSIntf.szMD_SEMANTICS Constant 11785
DSIntf.szMINVALUE Constant 11785
DSIntf.szNO_RESET_CALL Constant 11786
DSIntf.szORIGIN Constant 11786
DSIntf.szPRIMARY_KEY Constant 11786
DSIntf.szPROVFLAGS Constant 11787
DSIntf.szREADONLY Constant 11787
DSIntf.szSERVER_COL Constant 11787
DSIntf.szSERVERCALC Constant 11788
DSIntf.szstACCOLEOBJ Constant 11788
DSIntf.szstADTNESTEDTABLE Constant 11788
DSIntf.szstAUTOINC Constant 11789
DSIntf.szstBINARY Constant 11789
DSIntf.szstDBSOLEOBJ Constant 11789
DSIntf.szstFIXEDCHAR Constant 11790
DSIntf.szstFMTMEMO Constant 11790
DSIntf.szstGRAPHIC Constant 11790
DSIntf.szstGUID Constant 11791
DSIntf.szstHBINARY Constant 11791
DSIntf.szstHMEMO Constant 11791
DSIntf.szstMEMO Constant 11792
DSIntf.szstMONEY Constant 11792
DSIntf.szstOLEOBJ Constant 11792
DSIntf.szstREFNESTEDTABLE Constant 11793
DSIntf.szstTYPEDBINARY Constant 11793
DSIntf.szstWideMEMO Constant 11793
DSIntf.szSUBTYPE Constant 11794
DSIntf.szTABLE_NAME Constant 11794
DSIntf.szTYPENAME Constant 11794
DSIntf.szUNIQUE_KEY Constant 11795
DSIntf.szUPDATEMODE Constant 11795
DSIntf.szWIDTH Constant 11795
DSIntf.xmlOFF Constant 11796
DSIntf.xmlON Constant 11796
DSIntf.xmlUNTYPED Constant 11796
DSIntf.xmlUTF8 Constant 11797

DSIntf.xmlXMLDATATYPED Constant 11797
DSIntf.xmlXMLSCHEMA Constant 11797
DSIntf.xmlXMLUTF8 Constant 11798

DSProd Namespace 11799

Classes 11800
TCustomDataSetPageProducer Class 11800
TCustomDataSetPageProducer Members 11801
TCustomDataSetPageProducer Methods 11804
TCustomDataSetPageProducer Properties 11806
TDataSetPageProducer Class 11807
TDataSetPageProducer Members 11808
TDataSetPageProducer Properties 11811
TDataSetPageProducer Events 11814

ExptIntf Namespace 11815

Classes 11817
TIExpert Class 11817
TIExpert Members 11817
TIExpert Methods 11817

Functions 11821
ExptIntf.RegisterLibraryExpert Function 11821

Structs, Records, Enums 11822
ExptIntf.TExpertStyle Enumeration 11822

Types 11823
ExptIntf.TExpertInitProc Type 11823
ExptIntf.TExpertRegisterProc Type 11823
ExptIntf.TExpertState Type 11824
ExptIntf.TExpertTerminateProc Type 11824

Variables 11825
ExptIntf.LibraryExpertProc Variable 11825
ExptIntf.ToolServices Variable 11825

Constants 11826
ExptIntf.ExpertEntryPoint Constant 11826
ExptIntf.isExperts Constant 11826
ExptIntf.ValidExpertVersion Constant 11826

ExtActns Namespace 11829

Classes **11831**
TBrowseURL Class 11832
TBrowseURL Members 11832
TBrowseURL Methods 11836
TBrowseURL Properties 11837
TBrowseURL Events 11841
TCustomFileRun Class 11844
TCustomFileRun Members 11845
TCustomFileRun.Create Constructor 11849
TCustomFileRun Methods 11849
TCustomFileRun Properties 11852
TDownLoadURL Class 11857
TDownLoadURL Members 11857
TDownLoadURL Methods 11862
TDownLoadURL Properties 11864
TDownLoadURL Events 11870
TFileRun Class 11872
TFileRun Members 11873
TFileRun Properties 11878
TFileRun Events 11885
TListControlAction Class 11886
TListControlAction Members 11887
TListControlAction.Create Constructor 11891
TListControlAction Methods 11891
TListControlAction Properties 11893
TListControlClearSelection Class 11893
TListControlClearSelection Members 11894
TListControlClearSelection Methods 11898
TListControlClearSelection Properties 11899
TListControlClearSelection Events 11904
TListControlCopySelection Class 11906
TListControlCopySelection Members 11906
TListControlCopySelection Methods 11910
TListControlCopySelection Properties 11912
TListControlCopySelection Events 11917
TListControlDeleteSelection Class 11919

TListControlDeleteSelection Members 11920
TListControlDeleteSelection Methods 11923
TListControlDeleteSelection Properties 11925
TListControlDeleteSelection Events 11929
TListControlMoveSelection Class 11931
TListControlMoveSelection Members 11932
TListControlMoveSelection Methods 11936
TListControlSelectAll Class 11937
TListControlSelectAll Members 11937
TListControlSelectAll Methods 11941
TListControlSelectAll Properties 11943
TListControlSelectAll Events 11947
TNextTab Class 11949
TNextTab Members 11950
TNextTab Methods 11954
TNextTab Properties 11955
TNextTab Events 11961
TOpenPicture Class 11965
TOpenPicture Members 11965
TOpenPicture Methods 11970
TOpenPicture Properties 11970
TOpenPicture Events 11975
TPreviousTab Class 11977
TPreviousTab Members 11977
TPreviousTab Methods 11982
TPreviousTab Properties 11983
TPreviousTab Events 11988
TRichEditAction Class 11992
TRichEditAction Members 11992
TRichEditAction.Create Constructor 11997
TRichEditAction Methods 11997
TRichEditAlignCenter Class 11998
TRichEditAlignCenter Members 11999
TRichEditAlignCenter Methods 12004
TRichEditAlignLeft Class 12005
TRichEditAlignLeft Members 12005
TRichEditAlignLeft Methods 12010
TRichEditAlignRight Class 12011
TRichEditAlignRight Members 12012

TRichEditAlignRight Methods 12016
TRichEditBold Class 12017
TRichEditBold Members 12018
TRichEditBold Methods 12022
TRichEditBullets Class 12023
TRichEditBullets Members 12024
TRichEditBullets Methods 12028
TRichEditItalic Class 12029
TRichEditItalic Members 12030
TRichEditItalic Methods 12035
TRichEditStrikeOut Class 12036
TRichEditStrikeOut Members 12036
TRichEditStrikeOut Methods 12041
TRichEditUnderline Class 12042
TRichEditUnderline Members 12043
TRichEditUnderline Methods 12047
TSavePicture Class 12048
TSavePicture Members 12049
TSavePicture Methods 12053
TSavePicture Properties 12054
TSavePicture Events 12058
TSendMail Class 12061
TSendMail Members 12061
TSendMail.Destroy Destructor 12065
TSendMail Methods 12065
TSendMail Properties 12067
TSendMail Events 12071
TTabAction Class 12073
TTabAction Members 12074
TTabAction.Create Constructor 12078
TTabAction Methods 12078
TTabAction Properties 12080
TTabAction Events 12081
TURLAction Class 12083
TURLAction Members 12084
TURLAction Methods 12088
TURLAction Properties 12089
TURLAction Events 12093

Structs, Records, Enums 12096

ExtActns.TShowCmd Enumeration 12096
ExtActns.TURLDownloadStatus Enumeration 12097

Types **12099**
ExtActns.TDownloadProgressEvent Type 12099
ExtActns.TValidateTabEvent Type 12099

ExtCtrls Namespace 12101

Classes **12104**
EGridPanelException Class 12105
TBevel Class 12105
TBevel Members 12106
TBevel.Create Constructor 12113
TBevel Methods 12113
TBevel Properties 12113
TBoundLabel Class 12123
TBoundLabel Members 12123
TBoundLabel.Create Constructor 12131
TBoundLabel Methods 12131
TBoundLabel Properties 12132
TBoundLabel Events 12153
TCellCollection Class 12172
TCellCollection Members 12172
TCellCollection Methods 12174
TCellCollection Properties 12177
TCellItem Class 12177
TCellItem Members 12178
TCellItem.Create Constructor 12179
TCellItem Methods 12179
TCellItem Properties 12181
TColorBox Class 12182
TColorBox Members 12183
TColorBox Properties 12195
TColorBox Events 12220
TColorListBox Class 12246
TColorListBox Members 12246
TColorListBox Properties 12258
TColorListBox Events 12283
TColumnCollection Class 12311

TColumnCollection Members 12312
TColumnCollection.Create Constructor 12314
TColumnCollection Methods 12314
TColumnItem Class 12315
TControlBar Class 12316
TControlBar Members 12316
TControlBar Properties 12327
TControlBar Events 12356
TControlCollection Class 12393
TControlCollection Members 12394
TControlCollection.Create Constructor 12395
TControlCollection Methods 12396
TControlCollection Properties 12399
TControlItem Class 12400
TControlItem Members 12400
TControlItem.Create Constructor 12402
TControlItem.Destroy Destructor 12402
TControlItem Methods 12403
TControlItem Properties 12404
TCustomColorBox Class 12406
TCustomColorBox Members 12406
TCustomColorBox.Create Constructor 12419
TCustomColorBox Methods 12419
TCustomColorBox Properties 12423
TCustomColorBox Events 12426
TCustomColorListBox Class 12426
TCustomColorListBox Members 12426
TCustomColorListBox.Create Constructor 12438
TCustomColorListBox Methods 12438
TCustomColorListBox Properties 12442
TCustomColorListBox Events 12446
TCustomControlBar Class 12446
TCustomControlBar Members 12447
TCustomControlBar.Create Constructor 12457
TCustomControlBar.Destroy Destructor 12458
TCustomControlBar Methods 12458
TCustomControlBar Properties 12467
TCustomControlBar Events 12472
TCustomFlowPanel Class 12475

TCustomFlowPanel Members 12475
TCustomFlowPanel.Create Constructor 12486
TCustomFlowPanel.Destroy Destructor 12486
TCustomFlowPanel Methods 12487
TCustomFlowPanel Properties 12488
TCustomGridPanel Class 12489
TCustomGridPanel Members 12490
TCustomGridPanel.Create Constructor 12500
TCustomGridPanel.Destroy Destructor 12501
TCustomGridPanel Methods 12501
TCustomGridPanel Properties 12506
TCustomLabeledEdit Class 12508
TCustomLabeledEdit Members 12509
TCustomLabeledEdit.Create Constructor 12520
TCustomLabeledEdit Methods 12520
TCustomLabeledEdit Properties 12524
TCustomPanel Class 12525
TCustomPanel Members 12526
TCustomPanel.Create Constructor 12536
TCustomPanel Methods 12536
TCustomPanel Properties 12539
TCustomRadioGroup Class 12547
TCustomRadioGroup Members 12548
TCustomRadioGroup.Create Constructor 12558
TCustomRadioGroup.Destroy Destructor 12558
TCustomRadioGroup Methods 12559
TCustomRadioGroup Properties 12561
TCustomTrayIcon Class 12564
TCustomTrayIcon Members 12564
TCustomTrayIcon.Create Constructor 12568
TCustomTrayIcon.Destroy Destructor 12568
TCustomTrayIcon Methods 12568
TCustomTrayIcon Properties 12573
TCustomTrayIcon Events 12577
TFlowPanel Class 12579
TFlowPanel Members 12579
TFlowPanel Properties 12590
TFlowPanel Events 12625
TGridPanel Class 12660

TGridPanel Members 12661
TGridPanel Properties 12672
TGridPanel Events 12709
THeader Class 12744
THeader Members 12744
THeader.Create Constructor 12754
THeader.Destroy Destructor 12755
THeader Methods 12755
THeader Properties 12758
THeader Events 12776
TImage Class 12777
TImage Members 12778
TImage.Create Constructor 12785
TImage.Destroy Destructor 12786
TImage Methods 12786
TImage Properties 12788
TImage Events 12813
TLabeledEdit Class 12833
TLabeledEdit Members 12834
TLabeledEdit Properties 12845
TLabeledEdit Events 12873
TNotebook Class 12902
TNotebook Members 12902
TNotebook.Create Constructor 12912
TNotebook.Destroy Destructor 12913
TNotebook Methods 12913
TNotebook Properties 12916
TNotebook Events 12938
TPage Class 12961
TPage Members 12961
TPage.Create Constructor 12971
TPage Methods 12972
TPage Properties 12973
TPage Events 12980
TPaintBox Class 12981
TPaintBox Members 12982
TPaintBox.Create Constructor 12989
TPaintBox Methods 12989
TPaintBox Properties 12990

TPaintBox Events 13011
TPanel Class 13032
TPanel Members 13033
TPanel Properties 13043
TPanel Events 13080
TRadioGroup Class 13115
TRadioGroup Members 13115
TRadioGroup Properties 13126
TRadioGroup Events 13153
TRowCollection Class 13167
TRowCollection Members 13168
TRowCollection.Create Constructor 13170
TRowCollection Methods 13170
TRowItem Class 13171
TShape Class 13172
TShape Members 13172
TShape.Create Constructor 13179
TShape.Destroy Destructor 13180
TShape Methods 13180
TShape Properties 13181
TShape Events 13194
TSplitter Class 13210
TSplitter Members 13211
TSplitter.Create Constructor 13218
TSplitter.Destroy Destructor 13220
TSplitter Methods 13220
TSplitter Properties 13224
TSplitter Events 13239
TTimer Class 13242
TTimer Members 13242
TTimer.Create Constructor 13245
TTimer.Destroy Destructor 13245
TTimer Methods 13246
TTimer Properties 13246
TTimer Events 13251
TTrayIcon Class 13252
TTrayIcon Members 13252
TTrayIcon Properties 13256
TTrayIcon Events 13259

Functions **13262**
ExtCtrls.Frame3D Function 13262
ExtCtrls.NotebookHandlesNeeded Function 13263

Structs, Records, Enums **13264**
ExtCtrls.TBalloonFlags Enumeration 13264
ExtCtrls.TBandDrawingStyle Enumeration 13265
ExtCtrls.TBandPaintOption Enumeration 13265
ExtCtrls.TBevelShape Enumeration 13266
ExtCtrls.TBevelStyle Enumeration 13267
ExtCtrls.TColorBoxStyles Enumeration 13267
ExtCtrls.TCornerEdge Enumeration 13268
ExtCtrls.TExpandStyle Enumeration 13268
ExtCtrls.TFlowStyle Enumeration 13269
ExtCtrls.TLabelPosition Enumeration 13269
ExtCtrls.TResizeStyle Enumeration 13270
ExtCtrls.TShapeType Enumeration 13270
ExtCtrls.TSizeStyle Enumeration 13271

Types **13272**
ExtCtrls.NaturalNumber Type 13272
ExtCtrls.TBandDragEvent Type 13272
ExtCtrls.TBandInfoEvent Type 13273
ExtCtrls.TBandMoveEvent Type 13273
ExtCtrls.TBandPaintEvent Type 13273
ExtCtrls.TBandPaintOptions Type 13274
ExtCtrls.TBeginBandMoveEvent Type 13274
ExtCtrls.TCanResizeEvent Type 13274
ExtCtrls.TCellSpan Type 13275
ExtCtrls.TColorBoxStyle Type 13275
ExtCtrls.TEndBandMoveEvent Type 13276
ExtCtrls.TGetColorsEvent Type 13276
ExtCtrls.TLBGetColorsEvent Type 13276
ExtCtrls.TPanelBevel Type 13277
ExtCtrls.TRowSize Type 13277
ExtCtrls.TSectionEvent Type 13278
ExtCtrls.TSplitterCanResizeEvent Type 13278

Constants **13279**
ExtCtrls.NoColorSelected Constant 13279

ExtCtrls.WM_SYSTEM_TRAY_MESSAGE Constant 13279

ExtDlgs Namespace 13281

Classes 13282

TOpenPictureDialog Class 13282

TOpenPictureDialog Members 13283

TOpenPictureDialog.Create Constructor 13286

TOpenPictureDialog Methods 13287

TOpenPictureDialog Properties 13288

TOpenTextFileDialog Class 13292

TOpenTextFileDialog Members 13292

TOpenTextFileDialog.Create Constructor 13296

TOpenTextFileDialog.Destroy Destructor 13296

TOpenTextFileDialog Methods 13297

TOpenTextFileDialog Properties 13300

TSavePictureDialog Class 13300

TSavePictureDialog Members 13301

TSavePictureDialog Methods 13305

TSaveTextFileDialog Class 13305

TSaveTextFileDialog Members 13305

TSaveTextFileDialog Methods 13309

Constants 13310

ExtDlgs.DefaultEncodingNames Constant 13310

FileCtrl Namespace 13311

Classes 13313

TDirectoryListBox Class 13313

TDirectoryListBox Members 13314

TDirectoryListBox.Create Constructor 13326

TDirectoryListBox.Destroy Destructor 13326

TDirectoryListBox Fields 13327

TDirectoryListBox Methods 13328

TDirectoryListBox Properties 13335

TDirectoryListBox Events 13367

TDriveComboBox Class 13396

TDriveComboBox Members 13397

TDriveComboBox.Create Constructor 13409

TDriveComboBox.Destroy Destructor 13409
TDriveComboBox Fields 13409
TDriveComboBox Methods 13411
TDriveComboBox Properties 13413
TDriveComboBox Events 13441
TFileListBox Class 13465
TFileListBox Members 13466
TFileListBox.Create Constructor 13478
TFileListBox.Destroy Destructor 13479
TFileListBox Fields 13479
TFileListBox Methods 13482
TFileListBox Properties 13487
TFileListBox Events 13519
TFilterComboBox Class 13547
TFilterComboBox Members 13547
TFilterComboBox.Create Constructor 13559
TFilterComboBox.Destroy Destructor 13560
TFilterComboBox Methods 13560
TFilterComboBox Properties 13562
TFilterComboBox Events 13591

Functions 13616

FileCtrl.DirectoryExists Function 13616
FileCtrl.ForceDirectories Function 13617
FileCtrl.MinimizeName Function 13617
FileCtrl.ProcessPath Function 13618
FileCtrl.SelectDirectory Function 13619

Structs, Records, Enums 13620

FileCtrl.TDriveType Enumeration 13620
FileCtrl.TFileAttr Enumeration 13620
FileCtrl.TSelectDirExtOpt Enumeration 13621
FileCtrl.TSelectDirOpt Enumeration 13622
FileCtrl.TTextCase Enumeration 13623

Types 13624

FileCtrl.TFileType Type 13624
FileCtrl.TSelectDirExtOpts Type 13624
FileCtrl.TSelectDirOpts Type 13625

Constants 13626

FileCtrl.WNTYPE_DRIVE Constant 13626

FMTBcd Namespace 13627

Classes **13630**
EBcdException Class 13630
EBcdOverflowException Class 13630

Functions **13632**
FMTBcd.BcdAdd Function 13633
FMTBcd.BcdCompare Function 13633
FMTBcd.BcdDivide Function 13634
FMTBcd.BcdFromBytes Function 13635
FMTBcd.BcdMultiply Function 13635
FMTBcd.BcdPrecision Function 13636
FMTBcd.BcdScale Function 13636
FMTBcd.BcdSubtract Function 13637
FMTBcd.BcdToBytes Function 13637
FMTBcd.BCDToCurr Function 13637
FMTBcd.BcdToDouble Function 13638
FMTBcd.BcdToInteger Function 13638
FMTBcd.BcdToStr Function 13639
FMTBcd.BcdToStrF Function 13639
FMTBcd.CurrToBCD Function 13640
FMTBcd.DoubleToBcd Function 13641
FMTBcd.FormatBcd Function 13641
FMTBcd.IntegerToBcd Function 13642
FMTBcd.IsBcdNegative Function 13642
FMTBcd.NormalizeBcd Function 13643
FMTBcd.OldBCDToCurr Function 13643
FMTBcd.OldCurrToBCD Function 13644
FMTBcd.RoundAt Function 13644
FMTBcd.StrToBcd Function 13644
FMTBcd.TryStrToBcd Function 13645
FMTBcd.VarFMTBcd Function 13646
FMTBcd.VarFMTBcdCreate Function 13646
FMTBcd.VarIsFMTBcd Function 13647
FMTBcd.VarToBcd Function 13647

Structs, Records, Enums **13649**
FMTBcd.TBcd Record 13649

Types **13650**
FMTBcd.PBcd Type 13650
FMTBcd.TNibbleArray Type 13650

Constants **13651**
FMTBcd._DefaultDecimals Constant 13651
FMTBcd._NoDecimal Constant 13651
FMTBcd.DefaultFMTBcdScale Constant 13652
FMTBcd.MaxBcdPrecision Constant 13652
FMTBcd.MaxBcdScale Constant 13652
FMTBcd.MaxFMTBcdDigits Constant 13653
FMTBcd.MaxFMTBcdFractionSize Constant 13653
FMTBcd.MaxStringDigits Constant 13653
FMTBcd.NullBcd Constant 13654
FMTBcd.SizeOfFraction Constant 13654
FMTBcd.SizeOfTBCD Constant 13654

Forms Namespace 13657

Classes **13660**
TApplication Class 13660
TApplication Members 13661
TApplication.Create Constructor 13666
TApplication.Destroy Destructor 13667
TApplication Methods 13667
TApplication Properties 13703
TApplication Events 13724
TControlScrollBar Class 13744
TControlScrollBar Members 13744
TControlScrollBar Methods 13746
TControlScrollBar Properties 13747
TCustomActiveForm Class 13759
TCustomActiveForm Members 13760
TCustomActiveForm.Create Constructor 13773
TCustomActiveForm Methods 13773
TCustomActiveForm Properties 13774
TCustomActiveForm Events 13798
TCustomDockForm Class 13829
TCustomDockForm Members 13829
TCustomDockForm.Create Constructor 13843

TCustomDockForm Methods 13843
TCustomDockForm Properties 13845
TCustomForm Class 13849
TCustomForm Members 13849
TCustomForm.Create Constructor 13861
TCustomForm.CreateNew Constructor 13862
TCustomForm.Destroy Destructor 13862
TCustomForm Fields 13863
TCustomForm Methods 13863
TCustomForm Properties 13894
TCustomForm Events 13942
TCustomFrame Class 13978
TCustomFrame Members 13979
TCustomFrame.Create Constructor 13989
TCustomFrame Methods 13990
TForm Class 13992
TForm Members 13993
TForm Methods 14006
TForm Properties 14010
TForm Events 14064
TFrame Class 14114
TFrame Members 14115
TFrame Properties 14125
TFrame Events 14150
TGlassFrame Class 14186
TGlassFrame Members 14186
TGlassFrame.Create Constructor 14187
TGlassFrame Methods 14188
TGlassFrame Properties 14189
TGlassFrame Events 14191
TMonitor Class 14191
TMonitor Members 14192
TMonitor Properties 14193
TScreen Class 14197
TScreen Members 14198
TScreen.Create Constructor 14202
TScreen.Destroy Destructor 14202
TScreen Methods 14202
TScreen Properties 14205

TScreen Events 14234
TScrollBox Class 14238
TScrollBox Members 14239
TScrollBox.Create Constructor 14249
TScrollBox Methods 14250
TScrollBox Properties 14251
TScrollBox Events 14277
TScrollingWinControl Class 14312
TScrollingWinControl Members 14313
TScrollingWinControl.Create Constructor 14323
TScrollingWinControl.Destroy Destructor 14323
TScrollingWinControl Methods 14324
TScrollingWinControl Properties 14329
TScrollingWinControl Events 14331

Interfaces 14334
IDesignerHook Interface 14334
IDesignerHook Members 14335
IDesignerHook Methods 14335
IDesignerHook Properties 14339
IOleForm Interface 14340
IOleForm Members 14340
IOleForm Methods 14340

Functions 14342
Forms.AllocateHWnd Function 14342
Forms.DeallocateHWnd Function 14343
Forms.DisableTaskWindows Function 14343
Forms.DoneCtl3D Function 14344
Forms.EnableTaskWindows Function 14344
Forms.ForegroundTask Function 14344
Forms.FreeObjectInstance Function 14345
Forms.GetApplicationMainFormOnTaskBar Function 14345
Forms.GetCustomFormGlassFrame Function 14345
Forms.GetParentForm Function 14346
Forms.InitCtl3D Function 14346
Forms.IsAccel Function 14346
Forms.KeyboardStateToShiftState Function 14347
Forms.KeyDataToShiftState Function 14348
Forms.KeysToShiftState Function 14348

Forms.MakeObjectInstance Function 14349
Forms.RestoreFocusState Function 14349
Forms.SaveFocusState Function 14349
Forms.SetApplicationMainFormOnTaskBar Function 14350
Forms.SetAutoSubClass Function 14350
Forms.SetCustomFormGlassFrame Function 14350
Forms.Subclass3DDlg Function 14351
Forms.Subclass3DWnd Function 14351
Forms.ValidParentForm Function 14351

Structs, Records, Enums 14353

Forms.TActiveFormBorderStyle Enumeration 14353
Forms.TBorderIcon Enumeration 14354
Forms.TCloseAction Enumeration 14355
Forms.TCMHintShow Record 14355
Forms.TCMHintShowPause Record 14356
Forms.TCursorRec Record 14356
Forms.TDefaultMonitor Enumeration 14357
Forms.TFormBorderStyle Enumeration 14357
Forms.TFormStyle Enumeration 14358
Forms.THintInfo Record 14359
Forms.TMonitorDefaultTo Enumeration 14361
Forms.TPopupForm Record 14362
Forms.TPopupMode Enumeration 14362
Forms.TPopupWnd Record 14363
Forms.TPosition Enumeration 14363
Forms.TPrintScale Enumeration 14364
Forms.TScrollBarKind Enumeration 14365
Forms.TScrollBarStyle Enumeration 14366
Forms.TShowAction Enumeration 14366
Forms.TTileMode Enumeration 14367
Forms.TTimerMode Enumeration 14367
Forms.TWindowState Enumeration 14368

Types 14369

Forms.PCursorRec Type 14369
Forms.PHintInfo Type 14370
Forms.TBorderIcons Type 14370
Forms.TBorderStyle Type 14371
Forms.TCloseEvent Type 14371

Forms.TCloseQueryEvent Type 14372
Forms.TCustomFormClass Type 14372
Forms.TCustomFrameClass Type 14373
Forms.TExceptionEvent Type 14373
Forms.TFocusState Type 14374
Forms.TFormClass Type 14374
Forms.TFormState Type 14374
Forms.TGetHandleEvent Type 14375
Forms.THelpEvent Type 14375
Forms.TIdleEvent Type 14376
Forms.TMessageEvent Type 14376
Forms.TPopupFormArray Type 14377
Forms.TPopupWndArray Type 14377
Forms.TScrollBarInc Type 14377
Forms.TSetLayeredWindowAttributes Type 14378
Forms.TSettingChangeEvent Type 14378
Forms.TShortCutEvent Type 14381
Forms.TShowHintEvent Type 14381
Forms.TWindowHook Type 14382

Variables **14383**
Forms.Application Variable 14383
Forms.Ctl3DBtnWndProc Variable 14383
Forms.Ctl3DCtlColorEx Variable 14384
Forms.Ctl3DDlgFramePaint Variable 14384
Forms.HintWindowClass Variable 14384
Forms.Screen Variable 14385
Forms.SetLayeredWindowAttributes Variable 14385

Graphics Namespace 14387

Classes **14395**
EInvalidGraphic Class 14395
EInvalidGraphicOperation Class 14396
TBitmap Class 14396
TBitmap Members 14397
TBitmap.Create Constructor 14400
TBitmap.Destroy Destructor 14401
TBitmap Methods 14401
TBitmap Properties 14418

TBitmapImage Class 14429
TBitmapImage Members 14429
TBitmapImage.Destroy Destructor 14430
TBitmapImage Methods 14430
TBrush Class 14430
TBrush Members 14431
TBrush.Create Constructor 14432
TBrush.Destroy Destructor 14433
TBrush Methods 14433
TBrush Properties 14435
TBrushRecall Class 14439
TBrushRecall Members 14439
TBrushRecall.Create Constructor 14440
TCanvas Class 14441
TCanvas Members 14441
TCanvas.Create Constructor 14443
TCanvas.Destroy Destructor 14446
TCanvas Methods 14447
TCanvas Properties 14484
TCanvas Events 14496
TFont Class 14497
TFont Members 14497
TFont.Create Constructor 14499
TFont.Destroy Destructor 14500
TFont Methods 14500
TFont Properties 14506
TFontRecall Class 14514
TFontRecall Members 14515
TFontRecall.Create Constructor 14516
TGraphic Class 14516
TGraphic Members 14517
TGraphic.Create Constructor 14519
TGraphic Methods 14519
TGraphic Properties 14528
TGraphic Events 14531
TGraphicsObject Class 14532
TGraphicsObject Members 14532
TGraphicsObject Methods 14533
TGraphicsObject Properties 14535

TGraphicsObject Events 14536
TIcon Class 14537
TIcon Members 14537
TIcon.Create Constructor 14539
TIcon.Destroy Destructor 14541
TIcon Methods 14541
TIcon Properties 14546
TIconImage Class 14547
TIconImage Members 14548
TIconImage.Destroy Destructor 14548
TIconImage Methods 14548
TMetafile Class 14549
TMetafile Members 14549
TMetafile.Create Constructor 14551
TMetafile.Destroy Destructor 14553
TMetafile Methods 14554
TMetafile Properties 14561
TMetafileCanvas Class 14564
TMetafileCanvas Members 14564
TMetafileCanvas.Create Constructor 14567
TMetafileCanvas.CreateWithComment Constructor 14568
TMetafileCanvas.Destroy Destructor 14569
TMetafileImage Class 14569
TMetafileImage Members 14570
TMetafileImage.Destroy Destructor 14570
TMetafileImage Methods 14571
TPen Class 14571
TPen Members 14572
TPen.Create Constructor 14573
TPen.Destroy Destructor 14573
TPen Methods 14574
TPen Properties 14577
TPenRecall Class 14582
TPenRecall Members 14582
TPenRecall.Create Constructor 14583
TPicture Class 14584
TPicture Members 14584
TPicture.Create Constructor 14586
TPicture.Destroy Destructor 14586

TPicture Methods 14587
TPicture Properties 14598
TPicture Events 14602
TSharedImage Class 14603
TSharedImage Members 14604
TSharedImage Methods 14604
TSharedImage Properties 14605

Interfaces 14606
IChangeNotifier Interface 14606
IChangeNotifier Members 14606
IChangeNotifier Methods 14606

Functions 14608
Graphics.AllocPatternBitmap Function 14608
Graphics.BytesPerScanline Function 14609
Graphics.CharsetToIdent Function 14609
Graphics.ColorToIdent Function 14610
Graphics.ColorToRGB Function 14610
Graphics.ColorToString Function 14611
Graphics.CopyPalette Function 14612
Graphics.CreateGrayMappedBmp Function 14612
Graphics.CreateGrayMappedRes Function 14613
Graphics.CreateMappedBmp Function 14614
Graphics.CreateMappedRes Function 14614
Graphics.FreeMemoryContexts Function 14615
Graphics.GetCharsetValues Function 14615
Graphics.GetColorValues Function 14619
Graphics.GetDefFontCharSet Function 14619
Graphics.GetDIB Function 14620
Graphics.GetDIBSizes Function 14620
Graphics.GraphicExtension Function 14621
Graphics.GraphicFileMask Function 14624
Graphics.GraphicFilter Function 14624
Graphics.IdentToCharset Function 14627
Graphics.IdentToColor Function 14627
Graphics.PaletteChanged Function 14628
Graphics.StringToColor Function 14628
Graphics.TransparentStretchBlt Function 14629

Structs, Records, Enums 14630

Graphics.TBitmapHandleType Enumeration 14630
Graphics.TBrushData Record 14631
Graphics.TBrushStyle Enumeration 14631
Graphics.TCanvasOrientation Enumeration 14632
Graphics.TCanvasStates Enumeration 14633
Graphics.TCursorOrIcon Record 14633
Graphics.TFillMode Enumeration 14634
Graphics.TFillStyle Enumeration 14634
Graphics.TFontData Record 14635
Graphics.TFontPitch Enumeration 14635
Graphics.TFontStyle Enumeration 14636
Graphics.TIconRec Record 14636
Graphics.TPenData Record 14637
Graphics.TPenMode Enumeration 14637
Graphics.TPenStyle Enumeration 14639
Graphics.TPixelFormat Enumeration 14640
Graphics.TProgressStage Enumeration 14641
Graphics.TResData Record 14641
Graphics.TResource Record 14642
Graphics.TTextFormats Enumeration 14642
Graphics.TTransparentMode Enumeration 14643

Types **14645**
Graphics.HENHMETAFILE Type 14645
Graphics.HMETAFILE Type 14646
Graphics.PColor Type 14646
Graphics.PCursorOrIcon Type 14646
Graphics.PIconRec Type 14647
Graphics.PResource Type 14647
Graphics.TCanvasState Type 14647
Graphics.TColor Type 14648
Graphics.TCopyMode Type 14650
Graphics.TFontCharset Type 14651
Graphics.TFontDataName Type 14652
Graphics.TFontName Type 14653
Graphics.TFontStyles Type 14653
Graphics.TFontStylesBase Type 14653
Graphics.TGraphicClass Type 14654
Graphics.TProgressEvent Type 14655
Graphics.TTextFormat Type 14656

Variables **14657**
Graphics.DDBsOnly Variable 14657
Graphics.DefFontData Variable 14658
Graphics.SystemPalette16 Variable 14658

Constants **14659**
Graphics.cl3DDkShadow Constant 14663
Graphics.cl3DLight Constant 14663
Graphics.clActiveBorder Constant 14664
Graphics.clActiveCaption Constant 14664
Graphics.clAppWorkSpace Constant 14665
Graphics.clAqua Constant 14665
Graphics.clBackground Constant 14666
Graphics.clBlack Constant 14666
Graphics.clBlue Constant 14666
Graphics.clBtnFace Constant 14667
Graphics.clBtnHighlight Constant 14667
Graphics.clBtnShadow Constant 14668
Graphics.clBtnText Constant 14668
Graphics.clCaptionText Constant 14669
Graphics.clCream Constant 14669
Graphics.clDefault Constant 14670
Graphics.clDkGray Constant 14670
Graphics.clFuchsia Constant 14670
Graphics.clGradientActiveCaption Constant 14671
Graphics.clGradientInactiveCaption Constant 14671
Graphics.clGray Constant 14672
Graphics.clGrayText Constant 14672
Graphics.clGreen Constant 14673
Graphics.clHighlight Constant 14673
Graphics.clHighlightText Constant 14674
Graphics.clHotLight Constant 14674
Graphics.clInactiveBorder Constant 14675
Graphics.clInactiveCaption Constant 14675
Graphics.clInactiveCaptionText Constant 14675
Graphics.clInfoBk Constant 14676
Graphics.clInfoText Constant 14676
Graphics.clLime Constant 14677
Graphics.clLtGray Constant 14677

Graphics.clMaroon Constant 14678
Graphics.clMedGray Constant 14678
Graphics.clMenu Constant 14679
Graphics.clMenuBar Constant 14679
Graphics.clMenuHighlight Constant 14679
Graphics.clMenuText Constant 14680
Graphics.clMoneyGreen Constant 14680
Graphics.clNavy Constant 14681
Graphics.clNone Constant 14681
Graphics.clOlive Constant 14682
Graphics.clPurple Constant 14682
Graphics.clRed Constant 14683
Graphics.clScrollBar Constant 14683
Graphics.clSilver Constant 14683
Graphics.clSkyBlue Constant 14684
Graphics.clSystemColor Constant 14684
Graphics.clTeal Constant 14685
Graphics.clWebAliceBlue Constant 14685
Graphics.clWebAntiqueWhite Constant 14686
Graphics.clWebAqua Constant 14686
Graphics.clWebAquamarine Constant 14686
Graphics.clWebAzure Constant 14687
Graphics.clWebBeige Constant 14687
Graphics.clWebBisque Constant 14687
Graphics.clWebBlack Constant 14688
Graphics.clWebBlanchedAlmond Constant 14688
Graphics.clWebBlue Constant 14688
Graphics.clWebBlueViolet Constant 14689
Graphics.clWebBrown Constant 14689
Graphics.clWebBurlywood Constant 14689
Graphics.clWebCadetBlue Constant 14690
Graphics.clWebChartreuse Constant 14690
Graphics.clWebChocolate Constant 14690
Graphics.clWebCoral Constant 14691
Graphics.clWebCornFlowerBlue Constant 14691
Graphics.clWebCornSilk Constant 14691
Graphics.clWebCrimson Constant 14692
Graphics.clWebCyan Constant 14692
Graphics.clWebDarkBlue Constant 14692

Graphics.clWebDarkCyan Constant 14693
Graphics.clWebDarkGoldenRod Constant 14693
Graphics.clWebDarkGray Constant 14693
Graphics.clWebDarkgreen Constant 14694
Graphics.clWebDarkKhaki Constant 14694
Graphics.clWebDarkMagenta Constant 14694
Graphics.clWebDarkOliveGreen Constant 14695
Graphics.clWebDarkOrange Constant 14695
Graphics.clWebDarkOrchid Constant 14695
Graphics.clWebDarkRed Constant 14696
Graphics.clWebDarkSalmon Constant 14696
Graphics.clWebDarkSeaGreen Constant 14696
Graphics.clWebDarkSlateBlue Constant 14697
Graphics.clWebDarkSlategray Constant 14697
Graphics.clWebDarkTurquoise Constant 14697
Graphics.clWebDarkViolet Constant 14698
Graphics.clWebDeepPink Constant 14698
Graphics.clWebDeepskyBlue Constant 14698
Graphics.clWebDimGray Constant 14699
Graphics.clWebDodgerBlue Constant 14699
Graphics.clWebFirebrick Constant 14699
Graphics.clWebFloralWhite Constant 14700
Graphics.clWebForestGreen Constant 14700
Graphics.clWebFuchsia Constant 14700
Graphics.clWebGainsboro Constant 14701
Graphics.clWebGhostWhite Constant 14701
Graphics.clWebGold Constant 14701
Graphics.clWebGoldenRod Constant 14702
Graphics.clWebGray Constant 14702
Graphics.clWebGreen Constant 14702
Graphics.clWebGreenYellow Constant 14703
Graphics.clWebHoneydew Constant 14703
Graphics.clWebHotPink Constant 14703
Graphics.clWebIndianRed Constant 14704
Graphics.clWebIndigo Constant 14704
Graphics.clWebIvory Constant 14704
Graphics.clWebKhaki Constant 14705
Graphics.clWebLavender Constant 14705
Graphics.clWebLavenderBlush Constant 14705

Graphics.clWebLawnGreen Constant 14706
Graphics.clWebLemonChiffon Constant 14706
Graphics.clWebLightBlue Constant 14706
Graphics.clWebLightCoral Constant 14707
Graphics.clWebLightCyan Constant 14707
Graphics.clWebLightGoldenrodYellow Constant 14707
Graphics.clWebLightGreen Constant 14708
Graphics.clWebLightgrey Constant 14708
Graphics.clWebLightPink Constant 14708
Graphics.clWebLightSalmon Constant 14709
Graphics.clWebLightSeaGreen Constant 14709
Graphics.clWebLightSkyBlue Constant 14709
Graphics.clWebLightSlateGray Constant 14710
Graphics.clWebLightSteelBlue Constant 14710
Graphics.clWebLightYellow Constant 14710
Graphics.clWebLime Constant 14711
Graphics.clWebLimeGreen Constant 14711
Graphics.clWebLinen Constant 14711
Graphics.clWebMagenta Constant 14712
Graphics.clWebMaroon Constant 14712
Graphics.clWebMediumAquamarine Constant 14712
Graphics.clWebMediumBlue Constant 14713
Graphics.clWebMediumOrchid Constant 14713
Graphics.clWebMediumPurple Constant 14713
Graphics.clWebMediumSeaGreen Constant 14714
Graphics.clWebMediumSlateBlue Constant 14714
Graphics.clWebMediumSpringGreen Constant 14714
Graphics.clWebMediumTurquoise Constant 14715
Graphics.clWebMediumVioletRed Constant 14715
Graphics.clWebMidnightBlue Constant 14715
Graphics.clWebMintcream Constant 14716
Graphics.clWebMistyRose Constant 14716
Graphics.clWebMoccasin Constant 14716
Graphics.clWebNavajoWhite Constant 14717
Graphics.clWebNavy Constant 14717
Graphics.clWebOldLace Constant 14717
Graphics.clWebOlive Constant 14718
Graphics.clWebOliveDrab Constant 14718
Graphics.clWebOrange Constant 14718

Graphics.clWebOrangeRed Constant 14719
Graphics.clWebOrchid Constant 14719
Graphics.clWebPaleGoldenrod Constant 14719
Graphics.clWebPaleGreen Constant 14720
Graphics.clWebPaleTurquoise Constant 14720
Graphics.clWebPaleVioletRed Constant 14720
Graphics.clWebPapayaWhip Constant 14721
Graphics.clWebPeachPuff Constant 14721
Graphics.clWebPeru Constant 14721
Graphics.clWebPink Constant 14722
Graphics.clWebPlum Constant 14722
Graphics.clWebPowderBlue Constant 14722
Graphics.clWebPurple Constant 14723
Graphics.clWebRed Constant 14723
Graphics.clWebRosyBrown Constant 14723
Graphics.clWebRoyalBlue Constant 14724
Graphics.clWebSaddleBrown Constant 14724
Graphics.clWebSalmon Constant 14724
Graphics.clWebSandyBrown Constant 14725
Graphics.clWebSeaGreen Constant 14725
Graphics.clWebSeashell Constant 14725
Graphics.clWebSienna Constant 14726
Graphics.clWebSilver Constant 14726
Graphics.clWebSkyBlue Constant 14726
Graphics.clWebSlateBlue Constant 14727
Graphics.clWebSlateGray Constant 14727
Graphics.clWebSnow Constant 14727
Graphics.clWebSpringGreen Constant 14728
Graphics.clWebSteelBlue Constant 14728
Graphics.clWebTan Constant 14728
Graphics.clWebTeal Constant 14729
Graphics.clWebThistle Constant 14729
Graphics.clWebTomato Constant 14729
Graphics.clWebTurquoise Constant 14730
Graphics.clWebViolet Constant 14730
Graphics.clWebWheat Constant 14730
Graphics.clWebWhite Constant 14731
Graphics.clWebWhiteSmoke Constant 14731
Graphics.clWebYellow Constant 14731

Graphics.clWebYellowGreen Constant 14732
Graphics.clWhite Constant 14732
Graphics.clWindow Constant 14732
Graphics.clWindowFrame Constant 14733
Graphics.clWindowText Constant 14733
Graphics.clYellow Constant 14734
Graphics.cmBlackness Constant 14734
Graphics.cmDstInvert Constant 14735
Graphics.cmMergeCopy Constant 14735
Graphics.cmMergePaint Constant 14736
Graphics.cmNotSrcCopy Constant 14736
Graphics.cmNotSrcErase Constant 14737
Graphics.cmPatCopy Constant 14737
Graphics.cmPatInvert Constant 14738
Graphics.cmPatPaint Constant 14738
Graphics.cmSrcAnd Constant 14739
Graphics.cmSrcCopy Constant 14739
Graphics.cmSrcErase Constant 14740
Graphics.cmSrcInvert Constant 14740
Graphics.cmSrcPaint Constant 14741
Graphics.cmWhiteness Constant 14741
Graphics.ExtendedColorsCount Constant 14742
Graphics.rc3_Cursor Constant 14743
Graphics.rc3_Icon Constant 14743
Graphics.rc3_StockIcon Constant 14743
Graphics.StandardColorsCount Constant 14744
Graphics.WebColorsCount Constant 14745

GraphUtil Namespace 14747

Functions 14749

GraphUtil.ColorAdjustLuma Function 14749
GraphUtil.ColorHLSToRGB Function 14749
GraphUtil.ColorRGBToHLS Function 14750
GraphUtil.ColorToWebColorName Function 14750
GraphUtil.ColorToWebColorStr Function 14750
GraphUtil.DrawArrow Function 14751
GraphUtil.DrawCheck Function 14751
GraphUtil.DrawChevron Function 14751

GraphUtil.GetHighLightColor Function 14752
GraphUtil.GetShadowColor Function 14752
GraphUtil.GradientFillCanvas Function 14752
GraphUtil.RGBToWebColorName Function 14753
GraphUtil.RGBToWebColorStr Function 14753
GraphUtil.ScaleImage Function 14753
GraphUtil.SortColorArray Function 14754
GraphUtil.WebColorNameToColor Function 14754
GraphUtil.WebColorStrToColor Function 14754
GraphUtil.WebColorToRGB Function 14755

Structs, Records, Enums 14756
GraphUtil.TArrowType Enumeration 14756
GraphUtil.TColorArraySortType Enumeration 14756
GraphUtil.TGradientDirection Enumeration 14757
GraphUtil.TScrollDirection Enumeration 14757

Types 14759
GraphUtil.TColorArray Type 14759

Constants 14760
GraphUtil.WebNamedColors Constant 14760
GraphUtil.WebNamedColorsCount Constant 14762

Grids Namespace 14765

Classes 14767
EInvalidGridOperation Class 14767
TCustomDrawGrid Class 14768
TCustomDrawGrid Members 14768
TCustomDrawGrid Methods 14781
TCustomDrawGrid Properties 14785
TCustomDrawGrid Events 14794
TCustomGrid Class 14800
TCustomGrid Members 14801
TCustomGrid.Create Constructor 14813
TCustomGrid.Destroy Destructor 14813
TCustomGrid Fields 14814
TCustomGrid Methods 14815
TCustomGrid Properties 14842
TDrawGrid Class 14860

TDrawGrid Members 14861
TDrawGrid Properties 14874
TDrawGrid Events 14905
TInplaceEdit Class 14938
TInplaceEdit Members 14938
TInplaceEdit.Create Constructor 14950
TInplaceEdit Methods 14951
TInplaceEdit Properties 14958
TInplaceEditList Class 14959
TInplaceEditList Members 14959
TInplaceEditList.Create Constructor 14960
TInplaceEditList Methods 14960
TInplaceEditList Properties 14965
TInplaceEditList Events 14967
TStringGrid Class 14968
TStringGrid Members 14969
TStringGrid.Create Constructor 14983
TStringGrid.Destroy Destructor 14983
TStringGrid Methods 14983
TStringGrid Properties 14986
TStringGridStrings Class 14988
TStringGridStrings Members 14989
TStringGridStrings.Create Constructor 14991
TStringGridStrings Methods 14992

Structs, Records, Enums 14997
Grids.TEditStyle Enumeration 14997
Grids.TGridAxisDrawInfo Record 14998
Grids.TGridCoord Record 14998
Grids.TGridDrawInfo Record 14999
Grids.TGridOption Enumeration 14999
Grids.TGridRect Record 15001
Grids.TGridState Enumeration 15001

Types 15003
Grids.TDrawCellEvent Type 15003
Grids.TGetEditEvent Type 15004
Grids.TGetExtentsFunc Type 15004
Grids.TGridDrawState Type 15004
Grids.TGridMovement Type 15005

Grids.TGridOptions Type 15005
Grids.TGridScrollDirection Type 15006
Grids.TMovedEvent Type 15007
Grids.TOnGetPickListItems Type 15007
Grids.TSelectCellEvent Type 15007
Grids.TSetEditEvent Type 15008

Constants 15009
Grids.MaxCustomExtents Constant 15009
Grids.MaxShortInt Constant 15009

HelpIntfs Namespace 15011

Classes 15012
EHelpSystemException Class 15012

Interfaces 15013
ICustomHelpViewer Interface 15013
ICustomHelpViewer Members 15014
ICustomHelpViewer Methods 15014
IExtendedHelpViewer Interface 15017
IExtendedHelpViewer Members 15018
IExtendedHelpViewer Methods 15018
IHelpManager Interface 15020
IHelpManager Members 15021
IHelpManager Methods 15021
IHelpSelector Interface 15022
IHelpSelector Members 15023
IHelpSelector Methods 15023
IHelpSelector2 Interface 15024
IHelpSelector2 Members 15024
IHelpSelector2 Methods 15025
IHelpSystem Interface 15025
IHelpSystem Members 15025
IHelpSystem Methods 15026
IHelpSystem2 Interface 15028
IHelpSystem2 Members 15028
IHelpSystem2 Methods 15029
IHelpSystemFlags Interface 15029
IHelpSystemFlags Members 15029
IHelpSystemFlags Methods 15030

ISpecialWinHelpViewer Interface 15031
ISpecialWinHelpViewer Members 15031
ISpecialWinHelpViewer Methods 15032

Functions 15033
HelpIntfs.GetHelpSystem Function 15033
HelpIntfs.RegisterViewer Function 15033

HTTPApp Namespace 15035

Classes 15038
EWebBrokerException Class 15038
TAbstractContentParser Class 15039
TAbstractContentParser Members 15040
TAbstractContentParser.Create Constructor 15041
TAbstractContentParser Methods 15041
TAbstractContentParser Properties 15043
TAbstractWebModuleFactory Class 15043
TAbstractWebModuleFactory Members 15044
TAbstractWebModuleFactory Methods 15045
TAbstractWebModuleFactory Properties 15048
TAbstractWebModuleList Class 15051
TAbstractWebModuleList Members 15052
TAbstractWebModuleList Methods 15053
TAbstractWebModuleList Properties 15057
TAbstractWebModuleList Events 15059
TAbstractWebPageInfo Class 15060
TAbstractWebPageInfo Members 15061
TAbstractWebPageInfo Methods 15061
TAbstractWebPageInfo Properties 15065
TAbstractWebPageModuleFactory Class 15070
TAbstractWebPageModuleFactory Members 15070
TAbstractWebPageModuleFactory.Create Constructor 15071
TAbstractWebPageModuleFactory.Destroy Destructor 15071
TAbstractWebPageModuleFactory Methods 15072
TAbstractWebRequestFile Class 15073
TAbstractWebRequestFile Members 15073
TAbstractWebRequestFile Methods 15074
TAbstractWebRequestFile Properties 15076
TAbstractWebRequestFiles Class 15078

TAbstractWebRequestFiles Members 15079
TAbstractWebRequestFiles Methods 15079
TAbstractWebRequestFiles Properties 15080
TAbstractWebSession Class 15082
TAbstractWebSession Members 15082
TAbstractWebSession Methods 15083
TAbstractWebSession Properties 15087
TContentParser Class 15088
TContentParser Members 15089
TContentParser.Destroy Destructor 15090
TContentParser Methods 15090
TCookie Class 15092
TCookie Members 15092
TCookie.Create Constructor 15093
TCookie Methods 15094
TCookie Properties 15095
TCookieCollection Class 15098
TCookieCollection Members 15098
TCookieCollection.Create Constructor 15100
TCookieCollection Methods 15100
TCookieCollection Properties 15101
TCustomContentProducer Class 15102
TCustomContentProducer Members 15103
TCustomContentProducer Methods 15106
TCustomContentProducer Properties 15109
TCustomWebDispatcher Class 15110
TCustomWebDispatcher Members 15112
TCustomWebDispatcher.Create Constructor 15115
TCustomWebDispatcher.Destroy Destructor 15115
TCustomWebDispatcher Methods 15116
TCustomWebDispatcher Properties 15121
TCustomWebDispatcher Events 15123
TDefaultWebAppServices Class 15125
TDefaultWebAppServices Members 15125
TDefaultWebAppServices Methods 15127
TDefaultWebAppServices Properties 15129
TWebActionItem Class 15130
TWebActionItem Members 15131
TWebActionItem.Create Constructor 15133

TWebActionItem.Destroy Destructor 15133
TWebActionItem Methods 15133
TWebActionItem Properties 15135
TWebActionItem Events 15139
TWebActionItems Class 15140
TWebActionItems Members 15141
TWebActionItems.Create Constructor 15142
TWebActionItems Methods 15143
TWebActionItems Properties 15146
TWebDispatcher Class 15147
TWebDispatcher Members 15148
TWebDispatcher Properties 15152
TWebDispatcher Events 15152
TWebModule Class 15154
TWebModule Members 15155
TWebModule.Create Constructor 15159
TWebModule Properties 15159
TWebModule Events 15160
TWebRequest Class 15161
TWebRequest Members 15162
TWebRequest.Create Constructor 15165
TWebRequest.Destroy Destructor 15165
TWebRequest Methods 15165
TWebRequest Properties 15173
TWebResponse Class 15191
TWebResponse Members 15192
TWebResponse.Create Constructor 15194
TWebResponse.Destroy Destructor 15195
TWebResponse Fields 15195
TWebResponse Methods 15196
TWebResponse Properties 15206

Interfaces 15219

IGetAppDispatcher Interface 15219
IGetAppDispatcher Members 15219
IGetAppDispatcher Methods 15220
IGetWebAppServices Interface 15220
IGetWebAppServices Members 15220
IGetWebAppServices Methods 15221
IGetWebRequestHandler Interface 15221

IGetWebRequestHandler Members 15221
IGetWebRequestHandler Methods 15221
IMultiModuleSupport Interface 15222
IMultiModuleSupport Members 15222
IMultiModuleSupport Methods 15222
IProduceContent Interface 15223
IProduceContent Members 15224
IProduceContent Methods 15224
IProduceContentFrom Interface 15225
IProduceContentFrom Members 15225
IProduceContentFrom Methods 15225
ISetAppDispatcher Interface 15226
ISetAppDispatcher Members 15226
ISetAppDispatcher Methods 15226
IWebAppServices Interface 15227
IWebAppServices Members 15228
IWebAppServices Methods 15228
IWebAppServices Properties 15230
IWebDispatch Interface 15230
IWebDispatch Members 15231
IWebDispatch Methods 15231
IWebDispatch Properties 15232
IWebDispatchActions Interface 15233
IWebDispatcherAccess Interface 15234
IWebDispatcherAccess Members 15235
IWebDispatcherAccess Methods 15235
IWebExceptionHandler Interface 15236
IWebExceptionHandler Members 15236
IWebExceptionHandler Methods 15236
IWebRequestHandler Interface 15237
IWebRequestHandler Members 15237
IWebRequestHandler Methods 15237

Functions **15239**
HTTPApp.DayOfWeekStr Function 15239
HTTPApp.DosPathToUnixPath Function 15240
HTTPApp.ExtractHeaderFields Function 15240
HTTPApp.ExtractHTTPFields Function 15241
HTTPApp.HTMLDecode Function 15241
HTTPApp.HTMLEncode Function 15242

HTTPApp.HTTPDecode Function 15242
HTTPApp.HTTPEncode Function 15243
HTTPApp.MonthStr Function 15243
HTTPApp.ParseDate Function 15244
HTTPApp.RegisterContentParser Function 15244
HTTPApp.StatusString Function 15244
HTTPApp.UnixPathToDosPath Function 15245

Structs, Records, Enums 15246
HTTPApp.TMethodType Enumeration 15246
HTTPApp.TWebModuleCacheMode Enumeration 15246
HTTPApp.TWebModuleCreateMode Enumeration 15247

Types 15248
HTTPApp.TContentParserClass Type 15248
HTTPApp.THTTPMethodEvent Type 15248
HTTPApp.TModuleAddedProc Type 15249
HTTPApp.TWebExceptionEvent Type 15249
HTTPApp.TWebModuleContext Type 15249

Constants 15250
HTTPApp.MAX_DATETIMES Constant 15250
HTTPApp.MAX_INTEGERS Constant 15250
HTTPApp.MAX_STRINGS Constant 15251
HTTPApp.sDateFormat Constant 15251

HTTPProd Namespace 15253

Classes 15255
EScriptError Class 15255
EScriptError Members 15255
EScriptError.Create Constructor 15257
EScriptError.Destroy Destructor 15257
EScriptError Properties 15257
TAbstractScriptEnginesList Class 15258
TAbstractScriptEnginesList Members 15258
TAbstractScriptEnginesList.Create Constructor 15259
TAbstractScriptEnginesList Methods 15260
TAbstractScriptError Class 15260
TAbstractScriptError Members 15260
TAbstractScriptError Methods 15261

TAbstractScriptError Properties 15263
TAbstractScriptErrors Class 15264
TAbstractScriptErrors Members 15265
TAbstractScriptErrors Methods 15266
TAbstractScriptErrors Properties 15267
TAbstractScriptProducer Class 15267
TAbstractScriptProducer Members 15268
TAbstractScriptProducer.Create Constructor 15269
TBasePageProducer Class 15269
TBasePageProducer Members 15270
TBasePageProducer.Create Constructor 15274
TBasePageProducer Methods 15274
TBasePageProducer Properties 15279
TBasePageProducer Events 15280
TCustomPageProducer Class 15281
TCustomPageProducer Members 15281
TCustomPageProducer.Create Constructor 15285
TCustomPageProducer.Destroy Destructor 15285
TCustomPageProducer Methods 15285
TCustomPageProducer Properties 15286
THTMLTableAttributes Class 15288
THTMLTableAttributes Members 15289
THTMLTableAttributes.Create Constructor 15290
THTMLTableAttributes Methods 15290
THTMLTableAttributes Properties 15292
THTMLTableCellAttributes Class 15295
THTMLTableElementAttributes Class 15295
THTMLTableElementAttributes Members 15296
THTMLTableElementAttributes Methods 15297
THTMLTableElementAttributes Properties 15299
THTMLTableHeaderAttributes Class 15300
THTMLTableHeaderAttributes Members 15300
THTMLTableHeaderAttributes Methods 15302
THTMLTableHeaderAttributes Properties 15303
THTMLTableRowAttributes Class 15304
THTMLTagAttributes Class 15304
THTMLTagAttributes Members 15305
THTMLTagAttributes.Create Constructor 15306
THTMLTagAttributes Methods 15306

THTMLTagAttributes Properties 15307
THTMLTagAttributes Events 15308
TPageProducer Class 15309
TPageProducer Members 15310
TPageProducer Properties 15313
TPageProducer Events 15316

Interfaces 15317
IDesignerFileManager Interface 15317
IDesignerFileManager Members 15317
IDesignerFileManager Methods 15317
IGetLocateFileService Interface 15318
IGetLocateFileService Members 15318
IGetLocateFileService Methods 15319
IGetProducerTemplate Interface 15319
IGetProducerTemplate Members 15319
IGetProducerTemplate Methods 15320
ILocateFileService Interface 15320
ILocateFileService Members 15321
ILocateFileService Methods 15321
IScriptContext Interface 15321
IScriptContext Members 15322
IScriptContext Methods 15322
IScriptContext Properties 15322
IScriptProducer Interface 15323
IScriptProducer Members 15323
IScriptProducer Methods 15323
IScriptProducer Properties 15327
IWebVariableName Interface 15328
IWebVariableName Members 15328
IWebVariableName Methods 15329
IWebVariableName Properties 15329
IWebVariablesContainer Interface 15329
IWebVariablesContainer Members 15330
IWebVariablesContainer Methods 15330
IWebVariablesContainer Properties 15331

Functions 15333
HTTPProd.ContentFromScriptFile Function 15333
HTTPProd.ContentFromScriptStream Function 15333

HTTPProd.ContentFromStream Function 15334
HTTPProd.ContentFromString Function 15334
HTTPProd.FindComponentWebModuleContext Function 15334
HTTPProd.GetTagID Function 15335

Structs, Records, Enums 15336
HTTPProd.THTMLAlign Enumeration 15336
HTTPProd.THTMLVAlign Enumeration 15336
HTTPProd.TTag Enumeration 15337

Types 15338
HTTPProd.THandledTagProc Type 15338
HTTPProd.THandleTagProc Type 15338
HTTPProd.THTMLBgColor Type 15339
HTTPProd.THTMLTagEvent Type 15339
HTTPProd.TScriptProducerClass Type 15339

Variables 15340
HTTPProd.DesignerFileManager Variable 15340
HTTPProd.ScriptEnginesList Variable 15340

Constants 15341
HTTPProd.HTMLAlign Constant 15341
HTTPProd.HTMLVAlign Constant 15341

HTTPUtil Namespace 15343

Interfaces 15344
IStreamLoader Interface 15344
IStreamLoader Members 15344
IStreamLoader Methods 15345
IStreamLoader Properties 15348
IStringTokenizer Interface 15350
IStringTokenizer Members 15350
IStringTokenizer Methods 15351
IStringTokenizer Properties 15352

Functions 15353
HTTPUtil.FirstDelimiter Function 15353
HTTPUtil.FirstDelimiter Function 15353
HTTPUtil.GetDefaultStreamLoader Function 15354
HTTPUtil.HTMLEscape Function 15354

HTTPUtil.StartsWith Function 15354
HTTPUtil.StringTokenizer Function 15355
HTTPUtil.StringToStringArray Function 15355

ImgList Namespace 15357

Classes 15358
TChangeLink Class 15358
TChangeLink Members 15358
TChangeLink.Destroy Destructor 15359
TChangeLink Methods 15360
TChangeLink Properties 15360
TChangeLink Events 15361
TCustomImageList Class 15361
TCustomImageList Members 15362
TCustomImageList.Create Constructor 15365
TCustomImageList.CreateSize Constructor 15366
TCustomImageList.Destroy Destructor 15366
TCustomImageList Methods 15367
TCustomImageList Properties 15386
TCustomImageList Events 15393

Structs, Records, Enums 15394
ImgList.TDrawingStyle Enumeration 15394
ImgList.TImageType Enumeration 15395
ImgList.TLoadResource Enumeration 15395
ImgList.TResType Enumeration 15396

Types 15397
ImgList.TImageIndex Type 15397
ImgList.TLoadResources Type 15397
ImgList.TOverlay Type 15397

IMouse Namespace 15399

Classes 15401
TPanningWindow Class 15401
TPanningWindow Members 15401
TPanningWindow.Create Constructor 15412
TPanningWindow.Destroy Destructor 15412
TPanningWindow Methods 15413

TPanningWindow Properties 15421
TPanningWindow Events 15422

Functions **15423**
IMouse.StartPan Function 15423

Structs, Records, Enums **15424**
IMouse.TPanDirection Enumeration 15424
IMouse.TPanOption Enumeration 15424

Types **15426**
IMouse.TPanDirections Type 15426
IMouse.TPanOptions Type 15426
IMouse.TUpdateEvent Type 15426

Constants **15428**
IMouse.crPanAll Constant 15428
IMouse.crPanDown Constant 15428
IMouse.crPanDownLeft Constant 15429
IMouse.crPanDownRight Constant 15429
IMouse.crPanLeft Constant 15429
IMouse.crPanLeftRight Constant 15430
IMouse.crPanRight Constant 15430
IMouse.crPanUp Constant 15430
IMouse.crPanUpDown Constant 15431
IMouse.crPanUpLeft Constant 15431
IMouse.crPanUpRight Constant 15431

IniFiles Namespace 15433

Classes **15434**
EIniFileException Class 15434
TCustomIniFile Class 15434
TCustomIniFile Members 15435
TCustomIniFile.Create Constructor 15436
TCustomIniFile Methods 15437
TCustomIniFile Properties 15449
THashedStringList Class 15450
THashedStringList Members 15451
THashedStringList.Destroy Destructor 15453
THashedStringList Methods 15454
TIniFile Class 15455

TIniFile Members 15456
TIniFile.Destroy Destructor 15457
TIniFile Methods 15458
TMemIniFile Class 15462
TMemIniFile Members 15462
TMemIniFile.Create Constructor 15464
TMemIniFile.Destroy Destructor 15464
TMemIniFile Methods 15465
TMemIniFile Properties 15471
TStringHash Class 15471
TStringHash Members 15472
TStringHash.Create Constructor 15472
TStringHash.Destroy Destructor 15472
TStringHash Methods 15473

Structs, Records, Enums 15475
IniFiles.THashItem Record 15475

Types 15476
IniFiles.PHashItem Type 15476
IniFiles.PPHashItem Type 15476

IntfInfo Namespace 15477

Classes 15479
EInterfaceRTTIException Class 15479

Functions 15480
IntfInfo.GetClsMemberTypeInfo Function 15480
IntfInfo.GetDynArrayElTypeInfo Function 15480
IntfInfo.GetDynArrayNextInfo Function 15481
IntfInfo.GetDynArrayNextInfo2 Function 15481
IntfInfo.GetIntfMetaData Function 15481
IntfInfo.GetIntfMetaData Function 15482
IntfInfo.GetMethNum Function 15482
IntfInfo.GetPropListFlat Function 15482
IntfInfo.IsStoredPropConst Function 15483
IntfInfo.OtherTypeName Function 15483
IntfInfo.SameTypeInfo Function 15483
IntfInfo.TypeNamesMatch Function 15484

Structs, Records, Enums 15485

IntfInfo.TCallConv Enumeration 15485
IntfInfo.TFillMethodArrayOpt Enumeration 15485
IntfInfo.TIntfMetaData Record 15486
IntfInfo.TIntfMethEntry Record 15486
IntfInfo.TIntfParamEntry Record 15487

Types **15488**
IntfInfo.PIntfMetaData Type 15488
IntfInfo.PIntfMethEntry Type 15488
IntfInfo.PIntfParamEntry Type 15489
IntfInfo.TIntfMethEntryArray Type 15489
IntfInfo.TIntfParamEntryArray Type 15489
IntfInfo.TPIntfMethEntryArray Type 15490

Constants **15491**
IntfInfo.CallingConventionName Constant 15491
IntfInfo.TypeInfoNames Constant 15491

InvokeRegistry Namespace 15493

Classes **15496**
ERemotableException Class 15496
ERemotableException Members 15497
Create Constructor 15498
ERemotableException Properties 15499
ETypeRegistryException Class 15500
TDataContext Class 15501
TDataContext Members 15501
TDataContext.Create Constructor 15502
TDataContext.Destroy Destructor 15502
TDataContext Fields 15503
TDataContext Methods 15505
THeaderList Class 15509
THeaderList Members 15510
THeaderList.Create Constructor 15510
THeaderList.Destroy Destructor 15511
THeaderList Methods 15511
THeaderList Properties 15514
TInvContext Class 15516
TInvContext Members 15516
TInvContext Methods 15517

TInvokableClass Class 15520
TInvokableClass Members 15521
TInvokableClass.Create Constructor 15522
TInvokableClass Fields 15522
TInvokableClass Methods 15523
TInvokableClass Properties 15526
TInvokableClassRegistry Class 15526
TInvokableClassRegistry Members 15527
TInvokableClassRegistry.Create Constructor 15529
TInvokableClassRegistry.Destroy Destructor 15530
TInvokableClassRegistry Methods 15530
TRemotable Class 15556
TRemotable Members 15557
TRemotable.Create Constructor 15558
TRemotable.Destroy Destructor 15558
TRemotable Fields 15559
TRemotable Methods 15559
TRemotable Properties 15560
TRemotableTypeRegistry Class 15562
TRemotableTypeRegistry Members 15562
TRemotableTypeRegistry.Create Constructor 15563
TRemotableTypeRegistry.Destroy Destructor 15564
TRemotableTypeRegistry Methods 15564
TRemotableTypeRegistry Properties 15579
TRemotableXS Class 15580
TRemotableXS Members 15581
TRemotableXS Methods 15581
TSOAPAttachment Class 15583
TSOAPAttachment Members 15585
TSOAPAttachment.Create Constructor 15585
TSOAPAttachment.Destroy Destructor 15586
TSOAPAttachment Methods 15586
TSOAPAttachment Properties 15592
TSOAPHeader Class 15596
TSOAPHeader Members 15597
TSOAPHeader Methods 15598
TSOAPHeader Properties 15599
TSOAPHeaders Class 15600
TSOAPHeaders Members 15601

TSOAPHeaders Methods 15602
TSOAPHeadersBase Class 15605
TSOAPHeadersBase Members 15606
TSOAPHeadersBase Fields 15607
TSOAPHeadersBase Methods 15608

Interfaces 15609
IHeadersSetter Interface 15609
IHeadersSetter Members 15609
IHeadersSetter Methods 15609
IObjConverter Interface 15610
IObjConverter Members 15611
IObjConverter Methods 15611
ISOAPHeaders Interface 15613
ISOAPHeaders Members 15614
ISOAPHeaders Methods 15614
ISOAPHeaders Properties 15617

Functions 15619
InvokeRegistry.GetRemotableDataContext Function 15619
InvokeRegistry.InvRegistry Function 15620
InvokeRegistry.RemClassRegistry Function 15620
InvokeRegistry.RemTypeRegistry Function 15621
InvokeRegistry.SetRemotableDataContext Function 15622
InvokeRegistry.SubstituteStrings Function 15623

Structs, Records, Enums 15624
InvokeRegistry.eHeaderMethodType Enumeration 15624
InvokeRegistry.ExtNameMapItem Record 15625
InvokeRegistry.IntfExceptionItem Record 15625
InvokeRegistry.IntfHeaderItem Record 15626
InvokeRegistry.InvRegClassEntry Record 15627
InvokeRegistry.InvRegIntfEntry Record 15627
InvokeRegistry.MethParamNameMapItem Record 15629
InvokeRegistry.ObjectConvertOptions Enumeration 15629
InvokeRegistry.SerializationOptions Enumeration 15630
InvokeRegistry.TDynToClear Record 15631
InvokeRegistry.TIntfInvokeOption Enumeration 15632
InvokeRegistry.TObjMultiOptions Enumeration 15633
InvokeRegistry.TRemHeaderEntry Record 15634
InvokeRegistry.TRemRegEntry Record 15634

Types **15636**
InvokeRegistry.ERemotableExceptionClass Type 15636
InvokeRegistry.InvString Type 15637
InvokeRegistry.PTRemotable Type 15637
InvokeRegistry.TCreateInstanceProc Type 15637
InvokeRegistry.TExceptionItemArray Type 15638
InvokeRegistry.THeaderItemArray Type 15638
InvokeRegistry.THeaderMethodTypeArray Type 15638
InvokeRegistry.TIntfInvokeOptions Type 15639
InvokeRegistry.TInvokableClassClass Type 15639
InvokeRegistry.TInvRegIntfEntryArray Type 15639
InvokeRegistry.TObjectConvertOptions Type 15640
InvokeRegistry.TPascalRemotableClassRegistry Type 15640
InvokeRegistry.TPascalRemotableTypeRegistry Type 15640
InvokeRegistry.TRemotableClass Type 15641
InvokeRegistry.TRemotableClassRegistry Type 15641
InvokeRegistry.TRemotableXSClass Type 15641
InvokeRegistry.TRemRegEntryArray Type 15642
InvokeRegistry.TRequiredArray Type 15642
InvokeRegistry.TSerializationOptions Type 15642
InvokeRegistry.TSOAPAttachmentClass Type 15643
InvokeRegistry.TSOAPHeaderClass Type 15643

Variables **15644**
InvokeRegistry.AppNameSpacePrefix Variable 15644

Constants **15645**
InvokeRegistry.AS_ATTRIBUTE Constant 15645
InvokeRegistry.XMLBase64Types Constant 15645
InvokeRegistry.XMLSchemaInstNamepspaces Constant 15646
InvokeRegistry.XMLSchemaNamepspaces Constant 15646

ISAPIApp Namespace 15647

Classes **15648**
TISAPIApplication Class 15648
TISAPIApplication Members 15649
TISAPIApplication.Create Constructor 15652
TISAPIApplication.Destroy Destructor 15652
TISAPIApplication Methods 15652

TISAPIFactory Class 15654
TISAPIFactory Members 15654
TISAPIFactory.Create Constructor 15655
TISAPIFactory Methods 15655
Functions 15657
ISAPIApp.GetExtensionVersion Function 15657
ISAPIApp.HttpExtensionProc Function 15657
ISAPIApp.TerminateExtension Function 15658
Variables 15659
ISAPIApp.DispatchThread Variable 15659
IsapiHTTP Namespace 15661
Classes 15662
TISAPIRequest Class 15662
TISAPIRequest Members 15662
TISAPIRequest.Create Constructor 15665
TISAPIRequest Methods 15665
TISAPIRequest Properties 15670
TISAPIResponse Class 15670
TISAPIResponse Members 15671
TISAPIResponse.Create Constructor 15673
TISAPIResponse Methods 15674
Java Namespace 15681
Java.Sql Namespace 15682
Classes 15682
SQLException Class 15682
jpeg Namespace 15685
Classes 15686
TJPEGData Class 15686
TJPEGData Members 15687
TJPEGData.Destroy Destructor 15687
TJPEGData Methods 15687
TJPEGImage Class 15688
TJPEGImage Members 15689

TJPEGImage.Create Constructor 15691
TJPEGImage.Destroy Destructor 15692
TJPEGImage Methods 15692
TJPEGImage Properties 15701

Structs, Records, Enums 15706
jpeg.TJPEGDefaults Record 15706
jpeg.TJPEGPerformance Enumeration 15707
jpeg.TJPEGPixelFormat Enumeration 15707
jpeg.TJPEGScale Enumeration 15708

Types 15709
jpeg.TJPEGQualityRange Type 15709

Variables 15710
jpeg.JPEGDefaults Variable 15710

ListActns Namespace 15711

Classes 15713
TCustomListAction Class 15713
TCustomListAction Members 15714
TCustomListAction.Create Constructor 15718
TCustomListAction Methods 15718
TCustomListAction Properties 15721
TCustomListAction Events 15723
TCustomStaticListAction Class 15724
TCustomStaticListAction Members 15725
TCustomStaticListAction.Create Constructor 15729
TCustomStaticListAction.Destroy Destructor 15729
TCustomStaticListAction Methods 15729
TCustomStaticListAction Properties 15731
TCustomStaticListAction Events 15736
TCustomVirtualListAction Class 15738
TCustomVirtualListAction Members 15739
TCustomVirtualListAction Methods 15743
TCustomVirtualListAction Properties 15744
TCustomVirtualListAction Events 15744
TListActionLink Class 15744
TListActionLink Members 15745
TListActionLink Methods 15747

TListControlItem Class 15750
TListControlItem Members 15750
TListControlItem.Create Constructor 15752
TListControlItem Fields 15752
TListControlItem Methods 15753
TListControlItem Properties 15756
TListControlItems Class 15757
TListControlItems Members 15758
TListControlItems.Create Constructor 15760
TListControlItems Methods 15760
TListControlItems Properties 15762
TListControlItems Events 15764
TStaticListAction Class 15765
TStaticListAction Members 15766
TStaticListAction Properties 15770
TStaticListAction Events 15774
TStaticListItems Class 15777
TStaticListItems Members 15777
TStaticListItems Methods 15779
TVirtualListAction Class 15780
TVirtualListAction Members 15781
TVirtualListAction Properties 15786
TVirtualListAction Events 15790

Structs, Records, Enums 15792
ListActns.TListItemsSortType Enumeration 15792

Types 15793
ListActns.TGetItemCountEvent Type 15793
ListActns.TGetItemEvent Type 15793
ListActns.TGetVirtualItemEvent Type 15794
ListActns.TItemSelectedEvent Type 15794
ListActns.TListCompareEvent Type 15794
ListActns.TListControlItemClass Type 15795
ListActns.TListItemsCompare Type 15795

Mask Namespace 15797

Classes 15798
EDBEditError Class 15798
TCustomMaskEdit Class 15798

TCustomMaskEdit Members 15799
TCustomMaskEdit.Create Constructor 15810
TCustomMaskEdit Methods 15811
TCustomMaskEdit Properties 15820
TMaskEdit Class 15824
TMaskEdit Members 15824
TMaskEdit Properties 15836
TMaskEdit Events 15865

Types **15895**
Mask.TMaskedState Type 15895

Masks Namespace 15897

Classes **15898**
EMaskException Class 15898
TMask Class 15898
TMask Members 15899
TMask.Create Constructor 15899
TMask.Destroy Destructor 15900
TMask Methods 15900

Functions **15902**
Masks.MatchesMask Function 15902

MaskUtils Namespace 15903

Functions **15905**
MaskUtils.FormatMaskText Function 15905
MaskUtils.IsLiteralChar Function 15907
MaskUtils.MaskDoFormatText Function 15907
MaskUtils.MaskGetCharType Function 15907
MaskUtils.MaskGetCurrentDirectives Function 15908
MaskUtils.MaskGetFldSeparator Function 15908
MaskUtils.MaskGetMaskBlank Function 15908
MaskUtils.MaskGetMaskSave Function 15909
MaskUtils.MaskIntlLiteralToChar Function 15909
MaskUtils.MaskOffsetToOffset Function 15909
MaskUtils.MaskOffsetToWideOffset Function 15910
MaskUtils.OffsetToMaskOffset Function 15910
MaskUtils.PadInputLiterals Function 15910

Structs, Records, Enums **15911**
MaskUtils.TMaskCharType Enumeration 15911

Types **15912**
MaskUtils.TEditMask Type 15912
MaskUtils.TMaskDirectives Type 15913
MaskUtils.TMaskedText Type 15914

Variables **15915**
MaskUtils.DefaultBlank Variable 15915
MaskUtils.MaskFieldSeparator Variable 15915
MaskUtils.MaskNoSave Variable 15915

Constants **15917**
MaskUtils.mDirLiteral Constant 15917
MaskUtils.mDirLowerCase Constant 15917
MaskUtils.mDirReverse Constant 15918
MaskUtils.mDirUpperCase Constant 15918
MaskUtils.mMskAlpha Constant 15918
MaskUtils.mMskAlphaNum Constant 15919
MaskUtils.mMskAlphaNumOpt Constant 15919
MaskUtils.mMskAlphaOpt Constant 15919
MaskUtils.mMskAscii Constant 15920
MaskUtils.mMskAsciiOpt Constant 15920
MaskUtils.mMskDateSeparator Constant 15920
MaskUtils.mMskNumeric Constant 15921
MaskUtils.mMskNumericOpt Constant 15921
MaskUtils.mMskNumSymOpt Constant 15921
MaskUtils.mMskTimeSeparator Constant 15922

Math Namespace 15923

Classes **15927**
EInvalidArgument Class 15927

Functions **15929**
Math.ArcCos Function 15931
Math.ArcCosh Function 15932
Math.ArcCot Function 15932
Math.ArcCotH Function 15933
Math.ArcCsc Function 15933

Math.ArcCscH Function 15934
Math.ArcSec Function 15934
Math.ArcSecH Function 15934
Math.ArcSin Function 15935
Math.ArcSinh Function 15935
Math.ArcTan2 Function 15936
Math.ArcTanh Function 15936
Math.Ceil Function 15937
Math.ClearExceptions Function 15937
Math.CompareValue Function 15938
Math.Cosecant Function 15938
Math.Cosh Function 15939
Math.Cot Function 15939
Math.Cotan Function 15940
Math.CotH Function 15940
Math.Csc Function 15941
Math.CscH Function 15941
Math.CycleToDeg Function 15942
Math.CycleToGrad Function 15942
Math.CycleToRad Function 15942
Math.DegToCycle Function 15943
Math.DegToGrad Function 15943
Math.DegToRad Function 15944
Math.DivMod Function 15944
Math.DoubleDecliningBalance Function 15944
Math.EnsureRange Function 15945
Math.Floor Function 15945
Math.Frexp Function 15946
Math.FutureValue Function 15946
Math.GetExceptionMask Function 15947
Math.GetPrecisionMode Function 15947
Math.GetRoundMode Function 15948
Math.GradToCycle Function 15949
Math.GradToDeg Function 15949
Math.GradToRad Function 15949
Math.Hypot Function 15950
Math.IfThen Function 15950
Math.InRange Function 15950
Math.InterestPayment Function 15951

Math.InterestRate Function 15952
Math.InternalRateOfReturn Function 15952
Math.IntPower Function 15953
Math.IsInfinite Function 15953
Math.IsNan Function 15954
Math.IsZero Function 15954
Math.Ldexp Function 15955
Math.LnXP1 Function 15955
Math.Log10 Function 15955
Math.Log2 Function 15956
Math.LogN Function 15956
Math.Max Function 15957
Math.MaxIntValue Function 15957
Math.MaxValue Function 15958
Math.Mean Function 15958
Math.MeanAndStdDev Function 15959
Math.Min Function 15959
Math.MinIntValue Function 15960
Math.MinValue Function 15960
Math.MomentSkewKurtosis Function 15961
Math.NetPresentValue Function 15961
Math.Norm Function 15962
Math.NumberOfPeriods Function 15962
Math.Payment Function 15963
Math.PeriodPayment Function 15963
Math.Poly Function 15963
Math.PopnStdDev Function 15964
Math.PopnVariance Function 15964
Math.Power Function 15965
Math.PresentValue Function 15965
Math.RadToCycle Function 15966
Math.RadToDeg Function 15966
Math.RadToGrad Function 15966
Math.RandG Function 15967
Math.RandomFrom Function 15967
Math.RandomRange Function 15968
Math.RoundTo Function 15968
Math.SameValue Function 15969
Math.Sec Function 15970

Math.Secant Function 15970
Math.SecH Function 15971
Math.SetExceptionMask Function 15971
Math.SetPrecisionMode Function 15972
Math.SetRoundMode Function 15972
Math.Sign Function 15973
Math.SimpleRoundTo Function 15973
Math.SinCos Function 15974
Math.Sinh Function 15974
Math.SLNDepreciation Function 15975
Math.StdDev Function 15975
Math.Sum Function 15976
Math.SumInt Function 15976
Math.SumOfSquares Function 15977
Math.SumsAndSquares Function 15977
Math.SYDDepreciation Function 15978
Math.Tan Function 15978
Math.Tanh Function 15979
Math.TotalVariance Function 15979
Math.Variance Function 15980

Structs, Records, Enums 15981
Math.TFPUException Enumeration 15981
Math.TFPUPrecisionMode Enumeration 15982
Math.TFPURoundingMode Enumeration 15983
Math.TPaymentTime Enumeration 15984

Types 15985
Math.TFPUExceptionMask Type 15985
Math.TRoundToRange Type 15986
Math.TValueSign Type 15986

Constants 15988
Math.Infinity Constant 15988
Math.MaxComp Constant 15989
Math.MaxDouble Constant 15989
Math.MaxExtended Constant 15989
Math.MaxSingle Constant 15990
Math.MinComp Constant 15990
Math.MinDouble Constant 15991
Math.MinExtended Constant 15991

Math.MinSingle Constant 15991
Math.NaN Constant 15992
Math.NegativeValue Constant 15992
Math.NegInfinity Constant 15993
Math.PositiveValue Constant 15993
Math.ZeroValue Constant 15994

MConnect Namespace 15995

Classes 15996
TCOMConnection Class 15996
TCOMConnection Members 15997
TCOMConnection Methods 16001
TCustomObjectBroker Class 16002
TCustomObjectBroker Members 16003
TCustomObjectBroker Methods 16006
TDCOMConnection Class 16008
TDCOMConnection Members 16009
TDCOMConnection.Create Constructor 16013
TDCOMConnection Methods 16013
TDCOMConnection Properties 16014
TDispatchAppServer Class 16015
TDispatchAppServer Members 16015
TDispatchAppServer.Create Constructor 16017
TDispatchAppServer Methods 16017
TDispatchConnection Class 16017
TDispatchConnection Members 16018
TDispatchConnection.Create Constructor 16022
TDispatchConnection Methods 16022
TDispatchConnection Properties 16026
TDispatchConnection Events 16031
TOLEnterpriseConnection Class 16033
TOLEnterpriseConnection Members 16034
TOLEnterpriseConnection Methods 16038
TOLEnterpriseConnection Properties 16038
TSharedConnection Class 16039
TSharedConnection Members 16040
TSharedConnection.Create Constructor 16044
TSharedConnection.Destroy Destructor 16044

TSharedConnection Methods 16044
TSharedConnection Properties 16047
TSharedConnection Events 16048

Functions **16051**
MConnect.GetMIDASAppServerList Function 16051

Menus Namespace 16053

Classes **16055**
EMenuError Class 16055
TMainMenu Class 16056
TMainMenu Members 16056
TMainMenu Methods 16060
TMainMenu Properties 16063
TMainMenu Events 16067
TMenu Class 16067
TMenu Members 16068
TMenu.Create Constructor 16072
TMenu.Destroy Destructor 16072
TMenu Methods 16072
TMenu Properties 16081
TMenu Events 16089
TMenuActionLink Class 16090
TMenuActionLink Members 16091
TMenuActionLink Fields 16092
TMenuActionLink Methods 16093
TMenuItem Class 16102
TMenuItem Members 16103
TMenuItem.Create Constructor 16107
TMenuItem.Destroy Destructor 16108
TMenuItem Methods 16109
TMenuItem Properties 16132
TMenuItem Events 16153
TMenuItemEnumerator Class 16160
TMenuItemEnumerator Members 16161
TMenuItemEnumerator.Create Constructor 16161
TMenuItemEnumerator Methods 16161
TMenuItemEnumerator Properties 16162
TMenuItemStack Class 16162

TMenuItemStack Members 16163
TMenuItemStack Methods 16164
TPopupList Class 16164
TPopupList Members 16165
TPopupList Fields 16166
TPopupList Methods 16166
TPopupList Properties 16168
TPopupMenu Class 16169
TPopupMenu Members 16169
TPopupMenu.Create Constructor 16173
TPopupMenu.Destroy Destructor 16173
TPopupMenu Methods 16174
TPopupMenu Properties 16175
TPopupMenu Events 16184

Functions 16186

Menus.AnsiSameCaption Function 16186
Menus.DrawMenuItem Function 16187
Menus.GetHotkey Function 16187
Menus.IsAltGRPressed Function 16187
Menus.NewItem Function 16188
Menus.NewLine Function 16189
Menus.NewMenu Function 16190
Menus.NewPopupMenu Function 16191
Menus.NewSubMenu Function 16192
Menus.ShortCut Function 16193
Menus.ShortCutFromMessage Function 16194
Menus.ShortCutToKey Function 16194
Menus.ShortCutToText Function 16195
Menus.StripHotkey Function 16196
Menus.TextToShortCut Function 16196

Structs, Records, Enums 16198

Menus.TFindItemKind Enumeration 16198
Menus.TMenuAnimations Enumeration 16199
Menus.TMenuBreak Enumeration 16199
Menus.TMenuItemAutoFlag Enumeration 16200
Menus.TPopupAlignment Enumeration 16201
Menus.TTrackButton Enumeration 16201

Types 16203

Menus.PMenuItem Type 16203
Menus.TAdvancedMenuDrawItemEvent Type 16203
Menus.TMenuActionLinkClass Type 16204
Menus.TMenuAnimation Type 16204
Menus.TMenuAutoFlag Type 16205
Menus.TMenuChangeEvent Type 16205
Menus.TMenuDrawItemEvent Type 16206
Menus.TMenuMeasureItemEvent Type 16206

Variables 16207
Menus.PopupList Variable 16207
Menus.ShortCutItems Variable 16207
Menus.ValidMenuHotkeys Variable 16207

Constants 16209
Menus.cDialogSuffix Constant 16209
Menus.cHotkeyPrefix Constant 16209
Menus.cLineCaption Constant 16209

Messages Namespace 16211

Structs, Records, Enums 16223
Messages.TMessage Record 16225
Messages.TMSHMouseWheel Record 16226
Messages.TWMActivate Record 16226
Messages.TWMActivateApp Record 16227
Messages.TWMAskCBFormatName Record 16227
Messages.TWMChangeCBChain Record 16228
Messages.TWMCharToItem Record 16228
Messages.TWMChooseFont_GetLogFont Record 16229
Messages.TWMCommand Record 16229
Messages.TWMCompacting Record 16230
Messages.TWMCompareItem Record 16230
Messages.TWMContextMenu Record 16231
Messages.TWMCopyData Record 16231
Messages.TWMCreate Record 16232
Messages.TWMCtlColor Record 16232
Messages.TWMDDE_Ack Record 16233
Messages.TWMDDE_Advise Record 16233
Messages.TWMDDE_Data Record 16234
Messages.TWMDDE_Execute Record 16234

Messages.TWMDDE_Initiate Record 16235
Messages.TWMDDE_Poke Record 16235
Messages.TWMDDE_Request Record 16236
Messages.TWMDDE_Terminate Record 16236
Messages.TWMDDE_Unadvise Record 16237
Messages.TWMDeleteItem Record 16237
Messages.TWMDevModeChange Record 16238
Messages.TWMDisplayChange Record 16238
Messages.TWMDrawItem Record 16239
Messages.TWMDropFiles Record 16239
Messages.TWMEnable Record 16240
Messages.TWMEndSession Record 16240
Messages.TWMEnterIdle Record 16241
Messages.TWMEnterMenuLoop Record 16241
Messages.TWMEraseBkgnd Record 16242
Messages.TWMGetIcon Record 16242
Messages.TWMGetMinMaxInfo Record 16243
Messages.TWMGetText Record 16243
Messages.TWMHelp Record 16244
Messages.TWMHotKey Record 16244
Messages.TWMHScrollClipboard Record 16245
Messages.TWMInitDialog Record 16245
Messages.TWMInitMenu Record 16246
Messages.TWMInitMenuPopup Record 16246
Messages.TWMKey Record 16247
Messages.TWMKillFocus Record 16247
Messages.TWMMDIActivate Record 16248
Messages.TWMMDICascade Record 16248
Messages.TWMMDICreate Record 16249
Messages.TWMMDIDestroy Record 16249
Messages.TWMMDIMaximize Record 16250
Messages.TWMMDINext Record 16250
Messages.TWMMDIRestore Record 16251
Messages.TWMMDISetMenu Record 16251
Messages.TWMMDITile Record 16252
Messages.TWMMeasureItem Record 16252
Messages.TWMMenuChar Record 16253
Messages.TWMMenuSelect Record 16253
Messages.TWMMouse Record 16254

Messages.TWMMouseActivate Record 16254
Messages.TWMMouseWheel Record 16255
Messages.TWMMove Record 16255
Messages.TWMMoving Record 16256
Messages.TWMNCActivate Record 16256
Messages.TWMNCCalcSize Record 16257
Messages.TWMNCCreate Record 16257
Messages.TWMNCHitMessage Record 16258
Messages.TWMNCHitTest Record 16258
Messages.TWMNCPaint Record 16259
Messages.TWMNextDlgCtl Record 16259
Messages.TWMNoParams Record 16260
Messages.TWMNotify Record 16260
Messages.TWMNotifyFormat Record 16261
Messages.TWMPaint Record 16261
Messages.TWMPaintClipboard Record 16262
Messages.TWMPaletteChanged Record 16262
Messages.TWMPaletteIsChanging Record 16263
Messages.TWMParentNotify Record 16263
Messages.TWMPower Record 16264
Messages.TWMPrint Record 16264
Messages.TWMQueryEndSession Record 16265
Messages.TWMQuit Record 16265
Messages.TWMRenderFormat Record 16266
Messages.TWMScroll Record 16266
Messages.TWMSetCursor Record 16267
Messages.TWMSetFocus Record 16267
Messages.TWMSetFont Record 16268
Messages.TWMSetHotKey Record 16268
Messages.TWMSetIcon Record 16269
Messages.TWMSetRedraw Record 16269
Messages.TWMSetText Record 16270
Messages.TWMSettingChange Record 16270
Messages.TWMShowWindow Record 16271
Messages.TWMSize Record 16271
Messages.TWMSizeClipboard Record 16272
Messages.TWMSpoolerStatus Record 16272
Messages.TWMStyleChange Record 16273
Messages.TWMSysCommand Record 16273

Messages.TWMSysDeadChar Record 16274
Messages.TWMSystemError Record 16274
Messages.TWMTimer Record 16275
Messages.TWMUIState Record 16275
Messages.TWMVScrollClipboard Record 16276
Messages.TWMWindowPosMsg Record 16276
Messages.TWMWinIniChange Record 16277

Types **16278**

Messages.PMessage Type 16279
Messages.TWMCancelMode Type 16280
Messages.TWMChangeUIState Type 16280
Messages.TWMChar Type 16280
Messages.TWMChildActivate Type 16281
Messages.TWMClear Type 16281
Messages.TWMClose Type 16281
Messages.TWMCopy Type 16282
Messages.TWMCtlColorBtn Type 16282
Messages.TWMCtlColorDlg Type 16282
Messages.TWMCtlColorEdit Type 16283
Messages.TWMCtlColorListbox Type 16283
Messages.TWMCtlColorMsgbox Type 16283
Messages.TWMCtlColorScrollbar Type 16284
Messages.TWMCtlColorStatic Type 16284
Messages.TWMCut Type 16284
Messages.TWMDeadChar Type 16285
Messages.TWMDestroy Type 16285
Messages.TWMDestroyClipboard Type 16285
Messages.TWMDrawClipboard Type 16286
Messages.TWMExitMenuLoop Type 16286
Messages.TWMFontChange Type 16286
Messages.TWMGetDlgCode Type 16287
Messages.TWMGetFont Type 16287
Messages.TWMGetHotKey Type 16287
Messages.TWMGetTextLength Type 16288
Messages.TWMHScroll Type 16288
Messages.TWMIconEraseBkgnd Type 16288
Messages.TWMKeyDown Type 16289
Messages.TWMKeyUp Type 16289
Messages.TWMLButtonDblClk Type 16289

Messages.TWMLButtonDown Type 16290
Messages.TWMLButtonUp Type 16290
Messages.TWMMButtonDblClk Type 16290
Messages.TWMMButtonDown Type 16291
Messages.TWMMButtonUp Type 16291
Messages.TWMMDIGetActive Type 16291
Messages.TWMMDIIconArrange Type 16292
Messages.TWMMDIRefreshMenu Type 16292
Messages.TWMMouseMove Type 16292
Messages.TWMNCDestroy Type 16293
Messages.TWMNCLButtonDblClk Type 16293
Messages.TWMNCLButtonDown Type 16293
Messages.TWMNCLButtonUp Type 16294
Messages.TWMNCMButtonDblClk Type 16294
Messages.TWMNCMButtonDown Type 16294
Messages.TWMNCMButtonUp Type 16295
Messages.TWMNCMouseMove Type 16295
Messages.TWMNCRButtonDblClk Type 16295
Messages.TWMNCRButtonDown Type 16296
Messages.TWMNCRButtonUp Type 16296
Messages.TWMPaintIcon Type 16296
Messages.TWMPaste Type 16297
Messages.TWMPrintClient Type 16297
Messages.TWMQueryDragIcon Type 16297
Messages.TWMQueryNewPalette Type 16298
Messages.TWMQueryOpen Type 16298
Messages.TWMQueryUIState Type 16298
Messages.TWMQueueSync Type 16299
Messages.TWMRButtonDblClk Type 16299
Messages.TWMRButtonDown Type 16299
Messages.TWMRButtonUp Type 16300
Messages.TWMRenderAllFormats Type 16300
Messages.TWMStyleChanged Type 16300
Messages.TWMStyleChanging Type 16301
Messages.TWMSysChar Type 16301
Messages.TWMSysColorChange Type 16301
Messages.TWMSysKeyDown Type 16302
Messages.TWMSysKeyUp Type 16302
Messages.TWMTimeChange Type 16302

Messages.TWMUndo Type 16303
Messages.TWMUpdateUIState Type 16303
Messages.TWMVKeyToItem Type 16303
Messages.TWMVScroll Type 16304
Messages.TWMWindowPosChanged Type 16304
Messages.TWMWindowPosChanging Type 16304

Constants 16305

Messages.BM_CLICK Constant 16312
Messages.BM_GETCHECK Constant 16313
Messages.BM_GETIMAGE Constant 16313
Messages.BM_GETSTATE Constant 16313
Messages.BM_SETCHECK Constant 16314
Messages.BM_SETIMAGE Constant 16314
Messages.BM_SETSTATE Constant 16314
Messages.BM_SETSTYLE Constant 16315
Messages.BN_CLICKED Constant 16315
Messages.BN_DBLCLK Constant 16315
Messages.BN_DISABLE Constant 16316
Messages.BN_DOUBLECLICKED Constant 16316
Messages.BN_HILITE Constant 16316
Messages.BN_KILLFOCUS Constant 16317
Messages.BN_PAINT Constant 16317
Messages.BN_PUSHED Constant 16317
Messages.BN_SETFOCUS Constant 16318
Messages.BN_UNHILITE Constant 16318
Messages.BN_UNPUSHED Constant 16318
Messages.CB_ADDSTRING Constant 16319
Messages.CB_DELETESTRING Constant 16319
Messages.CB_DIR Constant 16319
Messages.CB_FINDSTRING Constant 16320
Messages.CB_FINDSTRINGEXACT Constant 16320
Messages.CB_GETCOUNT Constant 16320
Messages.CB_GETCURSEL Constant 16321
Messages.CB_GETDROPPEDCONTROLRECT Constant 16321
Messages.CB_GETDROPPEDSTATE Constant 16321
Messages.CB_GETDROPPEDWIDTH Constant 16322
Messages.CB_GETEDITSEL Constant 16322
Messages.CB_GETEXTENDEDUI Constant 16322
Messages.CB_GETHORIZONTALEXTENT Constant 16323

Messages.CB_GETITEMDATA Constant 16323
Messages.CB_GETITEMHEIGHT Constant 16323
Messages.CB_GETLBTEXT Constant 16324
Messages.CB_GETLBTEXTLEN Constant 16324
Messages.CB_GETLOCALE Constant 16324
Messages.CB_GETTOPINDEX Constant 16325
Messages.CB_INITSTORAGE Constant 16325
Messages.CB_INSERTSTRING Constant 16325
Messages.CB_LIMITTEXT Constant 16326
Messages.CB_MSGMAX Constant 16326
Messages.CB_RESETCONTENT Constant 16326
Messages.CB_SELECTSTRING Constant 16327
Messages.CB_SETCURSEL Constant 16327
Messages.CB_SETDROPPEDWIDTH Constant 16327
Messages.CB_SETEDITSEL Constant 16328
Messages.CB_SETEXTENDEDUI Constant 16328
Messages.CB_SETHORIZONTALEXTENT Constant 16328
Messages.CB_SETITEMDATA Constant 16329
Messages.CB_SETITEMHEIGHT Constant 16329
Messages.CB_SETLOCALE Constant 16329
Messages.CB_SETTOPINDEX Constant 16330
Messages.CB_SHOWDROPDOWN Constant 16330
Messages.CBN_CLOSEUP Constant 16330
Messages.CBN_DBLCLK Constant 16331
Messages.CBN_DROPDOWN Constant 16331
Messages.CBN_EDITCHANGE Constant 16331
Messages.CBN_EDITUPDATE Constant 16332
Messages.CBN_ERRSPACE Constant 16332
Messages.CBN_KILLFOCUS Constant 16332
Messages.CBN_SELCHANGE Constant 16333
Messages.CBN_SELENDCANCEL Constant 16333
Messages.CBN_SELENDOK Constant 16333
Messages.CBN_SETFOCUS Constant 16334
Messages.DM_GETDEFID Constant 16334
Messages.DM_REPOSITION Constant 16334
Messages.DM_SETDEFID Constant 16335
Messages.EM_CANUNDO Constant 16335
Messages.EM_CHARFROMPOS Constant 16335
Messages.EM_EMPTYUNDOBUFFER Constant 16336

Messages.EM_FMTLINES Constant 16336
Messages.EM_GETFIRSTVISIBLELINE Constant 16336
Messages.EM_GETHANDLE Constant 16337
Messages.EM_GETIMESTATUS Constant 16337
Messages.EM_GETLIMITTEXT Constant 16337
Messages.EM_GETLINE Constant 16338
Messages.EM_GETLINECOUNT Constant 16338
Messages.EM_GETMARGINS Constant 16338
Messages.EM_GETMODIFY Constant 16339
Messages.EM_GETPASSWORDCHAR Constant 16339
Messages.EM_GETRECT Constant 16339
Messages.EM_GETSEL Constant 16340
Messages.EM_GETTHUMB Constant 16340
Messages.EM_GETWORDBREAKPROC Constant 16340
Messages.EM_LIMITTEXT Constant 16341
Messages.EM_LINEFROMCHAR Constant 16341
Messages.EM_LINEINDEX Constant 16341
Messages.EM_LINELENGTH Constant 16342
Messages.EM_LINESCROLL Constant 16342
Messages.EM_POSFROMCHAR Constant 16342
Messages.EM_REPLACESEL Constant 16343
Messages.EM_SCROLL Constant 16343
Messages.EM_SCROLLCARET Constant 16343
Messages.EM_SETHANDLE Constant 16344
Messages.EM_SETIMESTATUS Constant 16344
Messages.EM_SETLIMITTEXT Constant 16344
Messages.EM_SETMARGINS Constant 16345
Messages.EM_SETMODIFY Constant 16345
Messages.EM_SETPASSWORDCHAR Constant 16345
Messages.EM_SETREADONLY Constant 16346
Messages.EM_SETRECT Constant 16346
Messages.EM_SETRECTNP Constant 16346
Messages.EM_SETSEL Constant 16347
Messages.EM_SETTABSTOPS Constant 16347
Messages.EM_SETWORDBREAKPROC Constant 16347
Messages.EM_UNDO Constant 16348
Messages.EN_CHANGE Constant 16348
Messages.EN_ERRSPACE Constant 16348
Messages.EN_HSCROLL Constant 16349

Messages.EN_KILLFOCUS Constant 16349
Messages.EN_MAXTEXT Constant 16349
Messages.EN_SETFOCUS Constant 16350
Messages.EN_UPDATE Constant 16350
Messages.EN_VSCROLL Constant 16350
Messages.LB_ADDFILE Constant 16351
Messages.LB_ADDSTRING Constant 16351
Messages.LB_DELETESTRING Constant 16351
Messages.LB_DIR Constant 16352
Messages.LB_FINDSTRING Constant 16352
Messages.LB_FINDSTRINGEXACT Constant 16352
Messages.LB_GETANCHORINDEX Constant 16353
Messages.LB_GETCARETINDEX Constant 16353
Messages.LB_GETCOUNT Constant 16353
Messages.LB_GETCURSEL Constant 16354
Messages.LB_GETHORIZONTALEXTENT Constant 16354
Messages.LB_GETITEMDATA Constant 16354
Messages.LB_GETITEMHEIGHT Constant 16355
Messages.LB_GETITEMRECT Constant 16355
Messages.LB_GETLOCALE Constant 16355
Messages.LB_GETSEL Constant 16356
Messages.LB_GETSELCOUNT Constant 16356
Messages.LB_GETSELITEMS Constant 16356
Messages.LB_GETTEXT Constant 16357
Messages.LB_GETTEXTLEN Constant 16357
Messages.LB_GETTOPINDEX Constant 16357
Messages.LB_INITSTORAGE Constant 16358
Messages.LB_INSERTSTRING Constant 16358
Messages.LB_ITEMFROMPOINT Constant 16358
Messages.LB_MSGMAX Constant 16359
Messages.LB_RESETCONTENT Constant 16359
Messages.LB_SELECTSTRING Constant 16359
Messages.LB_SELITEMRANGE Constant 16360
Messages.LB_SELITEMRANGEEX Constant 16360
Messages.LB_SETANCHORINDEX Constant 16360
Messages.LB_SETCARETINDEX Constant 16361
Messages.LB_SETCOLUMNWIDTH Constant 16361
Messages.LB_SETCOUNT Constant 16361
Messages.LB_SETCURSEL Constant 16362

Messages.LB_SETHORIZONTALEXTENT Constant 16362
Messages.LB_SETITEMDATA Constant 16362
Messages.LB_SETITEMHEIGHT Constant 16363
Messages.LB_SETLOCALE Constant 16363
Messages.LB_SETSEL Constant 16363
Messages.LB_SETTABSTOPS Constant 16364
Messages.LB_SETTOPINDEX Constant 16364
Messages.LBN_DBLCLK Constant 16364
Messages.LBN_ERRSPACE Constant 16365
Messages.LBN_KILLFOCUS Constant 16365
Messages.LBN_SELCANCEL Constant 16365
Messages.LBN_SELCHANGE Constant 16366
Messages.LBN_SETFOCUS Constant 16366
Messages.PSM_PAGEINFO Constant 16366
Messages.PSM_SHEETINFO Constant 16367
Messages.SBM_ENABLE_ARROWS Constant 16367
Messages.SBM_GETPOS Constant 16367
Messages.SBM_GETRANGE Constant 16368
Messages.SBM_GETSCROLLINFO Constant 16368
Messages.SBM_SETPOS Constant 16368
Messages.SBM_SETRANGE Constant 16369
Messages.SBM_SETRANGEREDRAW Constant 16369
Messages.SBM_SETSCROLLINFO Constant 16369
Messages.WM_ACTIVATE Constant 16370
Messages.WM_ACTIVATEAPP Constant 16370
Messages.WM_APP Constant 16370
Messages.WM_APPCOMMAND Constant 16371
Messages.WM_ASKCBFORMATNAME Constant 16371
Messages.WM_CANCELJOURNAL Constant 16371
Messages.WM_CANCELMODE Constant 16372
Messages.WM_CAPTURECHANGED Constant 16372
Messages.WM_CHANGECBCHAIN Constant 16372
Messages.WM_CHANGEUISTATE Constant 16373
Messages.WM_CHAR Constant 16373
Messages.WM_CHARTOITEM Constant 16373
Messages.WM_CHILDACTIVATE Constant 16374
Messages.WM_CLEAR Constant 16374
Messages.WM_CLOSE Constant 16374
Messages.WM_COALESCE_FIRST Constant 16375

Messages.WM_COALESCE_LAST Constant 16375
Messages.WM_COMMAND Constant 16375
Messages.WM_COMMNOTIFY Constant 16376
Messages.WM_COMPACTING Constant 16376
Messages.WM_COMPAREITEM Constant 16376
Messages.WM_CONTEXTMENU Constant 16377
Messages.WM_COPY Constant 16377
Messages.WM_COPYDATA Constant 16377
Messages.WM_CREATE Constant 16378
Messages.WM_CTLCOLOR Constant 16378
Messages.WM_CTLCOLORBTN Constant 16378
Messages.WM_CTLCOLORDLG Constant 16379
Messages.WM_CTLCOLOREDIT Constant 16379
Messages.WM_CTLCOLORLISTBOX Constant 16379
Messages.WM_CTLCOLORMSGBOX Constant 16380
Messages.WM_CTLCOLORSCROLLBAR Constant 16380
Messages.WM_CTLCOLORSTATIC Constant 16380
Messages.WM_CUT Constant 16381
Messages.WM_DDE_ACK Constant 16381
Messages.WM_DDE_ADVISE Constant 16381
Messages.WM_DDE_DATA Constant 16382
Messages.WM_DDE_EXECUTE Constant 16382
Messages.WM_DDE_FIRST Constant 16382
Messages.WM_DDE_INITIATE Constant 16383
Messages.WM_DDE_LAST Constant 16383
Messages.WM_DDE_POKE Constant 16383
Messages.WM_DDE_REQUEST Constant 16384
Messages.WM_DDE_TERMINATE Constant 16384
Messages.WM_DDE_UNADVISE Constant 16384
Messages.WM_DEADCHAR Constant 16385
Messages.WM_DELETEITEM Constant 16385
Messages.WM_DESTROY Constant 16385
Messages.WM_DESTROYCLIPBOARD Constant 16386
Messages.WM_DEVICECHANGE Constant 16386
Messages.WM_DEVMODECHANGE Constant 16386
Messages.WM_DISPLAYCHANGE Constant 16387
Messages.WM_DRAWCLIPBOARD Constant 16387
Messages.WM_DRAWITEM Constant 16387
Messages.WM_DROPFILES Constant 16388

Messages.WM_DWMCOLORIZATIONCOLORCHANGED Constant 16388
Messages.WM_DWMCOMPOSITIONCHANGED Constant 16388
Messages.WM_DWMNCRENDERINGCHANGED Constant 16389
Messages.WM_DWMWINDOWMAXIMIZEDCHANGE Constant 16389
Messages.WM_ENABLE Constant 16389
Messages.WM_ENDSESSION Constant 16390
Messages.WM_ENTERIDLE Constant 16390
Messages.WM_ENTERMENULOOP Constant 16390
Messages.WM_ENTERSIZEMOVE Constant 16391
Messages.WM_ERASEBKGND Constant 16391
Messages.WM_EXITMENULOOP Constant 16391
Messages.WM_EXITSIZEMOVE Constant 16392
Messages.WM_FONTCHANGE Constant 16392
Messages.WM_GETDLGCODE Constant 16392
Messages.WM_GETFONT Constant 16393
Messages.WM_GETHOTKEY Constant 16393
Messages.WM_GETICON Constant 16393
Messages.WM_GETMINMAXINFO Constant 16394
Messages.WM_GETOBJECT Constant 16394
Messages.WM_GETTEXT Constant 16394
Messages.WM_GETTEXTLENGTH Constant 16395
Messages.WM_HANDHELDFIRST Constant 16395
Messages.WM_HANDHELDLAST Constant 16395
Messages.WM_HELP Constant 16396
Messages.WM_HOTKEY Constant 16396
Messages.WM_HSCROLL Constant 16396
Messages.WM_HSCROLLCLIPBOARD Constant 16397
Messages.WM_ICONERASEBKGND Constant 16397
Messages.WM_IME_CHAR Constant 16397
Messages.WM_IME_COMPOSITION Constant 16398
Messages.WM_IME_COMPOSITIONFULL Constant 16398
Messages.WM_IME_CONTROL Constant 16398
Messages.WM_IME_ENDCOMPOSITION Constant 16399
Messages.WM_IME_KEYDOWN Constant 16399
Messages.WM_IME_KEYLAST Constant 16399
Messages.WM_IME_KEYUP Constant 16400
Messages.WM_IME_NOTIFY Constant 16400
Messages.WM_IME_REQUEST Constant 16400
Messages.WM_IME_SELECT Constant 16401

Messages.WM_IME_SETCONTEXT Constant 16401
Messages.WM_IME_STARTCOMPOSITION Constant 16401
Messages.WM_INITDIALOG Constant 16402
Messages.WM_INITMENU Constant 16402
Messages.WM_INITMENUPOPUP Constant 16402
Messages.WM_INPUT Constant 16403
Messages.WM_INPUTLANGCHANGE Constant 16403
Messages.WM_INPUTLANGCHANGEREQUEST Constant 16403
Messages.WM_KEYDOWN Constant 16404
Messages.WM_KEYFIRST Constant 16404
Messages.WM_KEYLAST Constant 16404
Messages.WM_KEYUP Constant 16405
Messages.WM_KILLFOCUS Constant 16405
Messages.WM_LBUTTONDBLCLK Constant 16405
Messages.WM_LBUTTONDOWN Constant 16406
Messages.WM_LBUTTONUP Constant 16406
Messages.WM_MBUTTONDBLCLK Constant 16406
Messages.WM_MBUTTONDOWN Constant 16407
Messages.WM_MBUTTONUP Constant 16407
Messages.WM_MDIACTIVATE Constant 16407
Messages.WM_MDICASCADE Constant 16408
Messages.WM_MDICREATE Constant 16408
Messages.WM_MDIDESTROY Constant 16408
Messages.WM_MDIGETACTIVE Constant 16409
Messages.WM_MDIICONARRANGE Constant 16409
Messages.WM_MDIMAXIMIZE Constant 16409
Messages.WM_MDINEXT Constant 16410
Messages.WM_MDIREFRESHMENU Constant 16410
Messages.WM_MDIRESTORE Constant 16410
Messages.WM_MDISETMENU Constant 16411
Messages.WM_MDITILE Constant 16411
Messages.WM_MEASUREITEM Constant 16411
Messages.WM_MENUCHAR Constant 16412
Messages.WM_MENUCOMMAND Constant 16412
Messages.WM_MENUDRAG Constant 16412
Messages.WM_MENUGETOBJECT Constant 16413
Messages.WM_MENURBUTTONUP Constant 16413
Messages.WM_MENUSELECT Constant 16413
Messages.WM_MOUSEACTIVATE Constant 16414

Messages.WM_MOUSEFIRST Constant 16414
Messages.WM_MOUSEHOVER Constant 16414
Messages.WM_MOUSELAST Constant 16415
Messages.WM_MOUSELEAVE Constant 16415
Messages.WM_MOUSEMOVE Constant 16415
Messages.WM_MOUSEWHEEL Constant 16416
Messages.WM_MOVE Constant 16416
Messages.WM_MOVING Constant 16416
Messages.WM_NCACTIVATE Constant 16417
Messages.WM_NCCALCSIZE Constant 16417
Messages.WM_NCCREATE Constant 16417
Messages.WM_NCDESTROY Constant 16418
Messages.WM_NCHITTEST Constant 16418
Messages.WM_NCLBUTTONDBLCLK Constant 16418
Messages.WM_NCLBUTTONDOWN Constant 16419
Messages.WM_NCLBUTTONUP Constant 16419
Messages.WM_NCMBUTTONDBLCLK Constant 16419
Messages.WM_NCMBUTTONDOWN Constant 16420
Messages.WM_NCMBUTTONUP Constant 16420
Messages.WM_NCMOUSEHOVER Constant 16420
Messages.WM_NCMOUSELEAVE Constant 16421
Messages.WM_NCMOUSEMOVE Constant 16421
Messages.WM_NCPAINT Constant 16421
Messages.WM_NCRBUTTONDBLCLK Constant 16422
Messages.WM_NCRBUTTONDOWN Constant 16422
Messages.WM_NCRBUTTONUP Constant 16422
Messages.WM_NCXBUTTONDBLCLK Constant 16423
Messages.WM_NCXBUTTONDOWN Constant 16423
Messages.WM_NCXBUTTONUP Constant 16423
Messages.WM_NEXTDLGCTL Constant 16424
Messages.WM_NEXTMENU Constant 16424
Messages.WM_NOTIFY Constant 16424
Messages.WM_NOTIFYFORMAT Constant 16425
Messages.WM_NULL Constant 16425
Messages.WM_PAINT Constant 16425
Messages.WM_PAINTCLIPBOARD Constant 16426
Messages.WM_PAINTICON Constant 16426
Messages.WM_PALETTECHANGED Constant 16426
Messages.WM_PALETTEISCHANGING Constant 16427

Messages.WM_PARENTNOTIFY Constant 16427
Messages.WM_PASTE Constant 16427
Messages.WM_PENWINFIRST Constant 16428
Messages.WM_PENWINLAST Constant 16428
Messages.WM_POWER Constant 16428
Messages.WM_POWERBROADCAST Constant 16429
Messages.WM_PRINT Constant 16429
Messages.WM_PRINTCLIENT Constant 16429
Messages.WM_QUERYDRAGICON Constant 16430
Messages.WM_QUERYENDSESSION Constant 16430
Messages.WM_QUERYNEWPALETTE Constant 16430
Messages.WM_QUERYOPEN Constant 16431
Messages.WM_QUERYUISTATE Constant 16431
Messages.WM_QUEUESYNC Constant 16431
Messages.WM_QUIT Constant 16432
Messages.WM_RBUTTONDBLCLK Constant 16432
Messages.WM_RBUTTONDOWN Constant 16432
Messages.WM_RBUTTONUP Constant 16433
Messages.WM_RENDERALLFORMATS Constant 16433
Messages.WM_RENDERFORMAT Constant 16433
Messages.WM_SETCURSOR Constant 16434
Messages.WM_SETFOCUS Constant 16434
Messages.WM_SETFONT Constant 16434
Messages.WM_SETHOTKEY Constant 16435
Messages.WM_SETICON Constant 16435
Messages.WM_SETREDRAW Constant 16435
Messages.WM_SETTEXT Constant 16436
Messages.WM_SETTINGCHANGE Constant 16436
Messages.WM_SHOWWINDOW Constant 16436
Messages.WM_SIZE Constant 16437
Messages.WM_SIZECLIPBOARD Constant 16437
Messages.WM_SIZING Constant 16437
Messages.WM_SPOOLERSTATUS Constant 16438
Messages.WM_STYLECHANGED Constant 16438
Messages.WM_STYLECHANGING Constant 16438
Messages.WM_SYSCHAR Constant 16439
Messages.WM_SYSCOLORCHANGE Constant 16439
Messages.WM_SYSCOMMAND Constant 16439
Messages.WM_SYSDEADCHAR Constant 16440

Messages.WM_SYSKEYDOWN Constant 16440
Messages.WM_SYSKEYUP Constant 16440
Messages.WM_SYSTEMERROR Constant 16441
Messages.WM_TABLET_FIRST Constant 16441
Messages.WM_TABLET_LAST Constant 16441
Messages.WM_TCARD Constant 16442
Messages.WM_THEMECHANGED Constant 16442
Messages.WM_TIMECHANGE Constant 16442
Messages.WM_TIMER Constant 16443
Messages.WM_UNDO Constant 16443
Messages.WM_UNICHAR Constant 16443
Messages.WM_UNINITMENUPOPUP Constant 16444
Messages.WM_UPDATEUISTATE Constant 16444
Messages.WM_USER Constant 16444
Messages.WM_USERCHANGED Constant 16445
Messages.WM_VKEYTOITEM Constant 16445
Messages.WM_VSCROLL Constant 16445
Messages.WM_VSCROLLCLIPBOARD Constant 16446
Messages.WM_WINDOWPOSCHANGED Constant 16446
Messages.WM_WINDOWPOSCHANGING Constant 16446
Messages.WM_WININICHANGE Constant 16447
Messages.WM_WTSSESSION_CHANGE Constant 16447

Midas Namespace 16449

Interfaces 16450
IAppServer Interface 16450
IAppServer Members 16451
IAppServer Methods 16451
IAppServerDisp Interface 16457
IAppServerDisp Members 16457
IAppServerDisp Methods 16457

Constants 16460
Midas.IID_IAppServer Constant 16460
Midas.LIBID_Midas Constant 16460

MidasCon Namespace 16461

Classes 16462

TMidasConnection Class 16462
TMidasConnection Members 16462
TMidasConnection.Create Constructor 16466
TMidasConnection Methods 16467
TMidasConnection Properties 16468
TRemoteServer Class 16470

Structs, Records, Enums 16472
MidasCon.TConnectType Enumeration 16472

MidItems Namespace 16473

Classes 16478
TApplyUpdatesButton Class 16480
TApplyUpdatesButton Members 16481
TApplyUpdatesButton.Create Constructor 16484
TApplyUpdatesButton Methods 16484
TApplyUpdatesButton Properties 16486
TCaptionAttributes Class 16489
TCaptionAttributes Members 16489
TCaptionAttributes.Create Constructor 16490
TCaptionAttributes Properties 16491
TCustomDataForm Class 16493
TCustomDataForm Members 16494
TCustomDataForm Methods 16498
TCustomDataGrid Class 16499
TCustomDataGrid Members 16501
TCustomDataGrid.Create Constructor 16505
TCustomDataGrid.Destroy Destructor 16506
TCustomDataGrid Methods 16506
TCustomDataGrid Properties 16509
TCustomDataNavigator Class 16511
TCustomDataNavigator Members 16511
TCustomDataNavigator Methods 16516
TCustomDataNavigator Properties 16518
TCustomFieldGroup Class 16519
TCustomFieldGroup Members 16520
TCustomFieldGroup.Create Constructor 16525
TCustomFieldGroup.Destroy Destructor 16525
TCustomFieldGroup Methods 16526

TCustomFieldGroup Properties 16527
TCustomLayoutGroup Class 16528
TCustomLayoutGroup Members 16529
TCustomLayoutGroup.Create Constructor 16533
TCustomLayoutGroup.Destroy Destructor 16533
TCustomLayoutGroup Methods 16534
TCustomLayoutGroup Properties 16535
TCustomQueryButtons Class 16537
TCustomQueryButtons Members 16538
TCustomQueryButtons Methods 16542
TCustomQueryButtons Properties 16544
TCustomQueryForm Class 16545
TCustomQueryForm Members 16546
TCustomQueryForm.Create Constructor 16550
TCustomQueryForm Methods 16551
TCustomQueryForm Properties 16553
TDataForm Class 16554
TDataForm Members 16555
TDataForm Properties 16559
TDataGrid Class 16560
TDataGrid Members 16562
TDataGrid Properties 16567
TDataNavigator Class 16569
TDataNavigator Members 16569
TDataNavigator Properties 16574
TDataSetButton Class 16576
TDataSetButton Members 16577
TDataSetButton Fields 16581
TDataSetButton Methods 16581
TDataSetButton Properties 16583
TDeleteButton Class 16586
TDeleteButton Members 16587
TDeleteButton.Create Constructor 16591
TFieldGroup Class 16591
TFieldGroup Members 16592
TFieldGroup Methods 16597
TFieldGroup Properties 16598
TFieldRadioGroup Class 16599
TFieldRadioGroup Members 16600

TFieldRadioGroup Properties 16605
TFieldSelectOptions Class 16612
TFieldSelectOptions Members 16613
TFieldSelectOptions Properties 16618
TFieldStatus Class 16624
TFieldStatus Members 16625
TFieldStatus Properties 16629
TFieldText Class 16631
TFieldText Members 16632
TFieldText Properties 16636
TFieldTextArea Class 16640
TFieldTextArea Members 16641
TFieldTextArea Properties 16646
TFirstButton Class 16651
TFirstButton Members 16652
TFirstButton.Create Constructor 16655
TFormLayout Class 16656
TFormLayout Members 16656
TFormLayout.Create Constructor 16658
TFormLayout Methods 16658
TFormLayout Properties 16663
TGridAttributes Class 16664
TGridAttributes Members 16665
TGridAttributes.Create Constructor 16666
TGridAttributes Methods 16666
TGridAttributes Properties 16667
TGridRowAttributes Class 16671
TGridRowAttributes Members 16672
TGridRowAttributes.Create Constructor 16673
TGridRowAttributes Methods 16673
TGridRowAttributes Properties 16674
TInsertButton Class 16677
TInsertButton Members 16678
TInsertButton.Create Constructor 16682
TLastButton Class 16682
TLastButton Members 16683
TLastButton.Create Constructor 16687
TLayoutGroup Class 16687
TLayoutGroup Members 16688

TLayoutGroup Properties 16692
TNextButton Class 16694
TNextButton Members 16695
TNextButton.Create Constructor 16699
TNextPageButton Class 16699
TNextPageButton Members 16700
TNextPageButton.Create Constructor 16703
TPostButton Class 16704
TPostButton Members 16705
TPostButton.Create Constructor 16708
TPriorButton Class 16709
TPriorButton Members 16709
TPriorButton.Create Constructor 16713
TPriorPageButton Class 16713
TPriorPageButton Members 16714
TPriorPageButton.Create Constructor 16718
TQueryButton Class 16718
TQueryButton Members 16719
TQueryButton.Create Constructor 16723
TQueryButton Fields 16723
TQueryButton Methods 16724
TQueryButton Properties 16727
TQueryButtons Class 16730
TQueryButtons Members 16731
TQueryButtons Properties 16735
TQueryFieldGroup Class 16736
TQueryFieldGroup Members 16738
TQueryFieldGroup Methods 16743
TQueryFieldGroup Properties 16744
TQueryForm Class 16745
TQueryForm Members 16746
TQueryForm Properties 16751
TQueryRadioGroup Class 16753
TQueryRadioGroup Members 16754
TQueryRadioGroup Methods 16759
TQueryRadioGroup Properties 16760
TQuerySelectOptions Class 16768
TQuerySelectOptions Members 16770
TQuerySelectOptions Methods 16774

TQuerySelectOptions Properties 16776
TQueryText Class 16783
TQueryText Members 16784
TQueryText.Create Constructor 16788
TQueryText Methods 16788
TQueryText Properties 16790
TQueryTextArea Class 16795
TQueryTextArea Members 16796
TQueryTextArea.Create Constructor 16800
TQueryTextArea.Destroy Destructor 16801
TQueryTextArea Methods 16801
TQueryTextArea Properties 16803
TResetQueryButton Class 16809
TResetQueryButton Members 16809
TResetQueryButton.Create Constructor 16813
TResetQueryButton Methods 16813
TSelectOptionsColumn Class 16814
TSelectOptionsColumn Members 16815
TSelectOptionsColumn Methods 16819
TSelectOptionsColumn Properties 16820
TStatusColumn Class 16826
TStatusColumn Members 16827
TStatusColumn Methods 16830
TStatusColumn Properties 16831
TSubmitQueryButton Class 16833
TSubmitQueryButton Members 16834
TSubmitQueryButton.Create Constructor 16838
TSubmitQueryButton Methods 16838
TTextAreaColumn Class 16838
TTextAreaColumn Members 16839
TTextAreaColumn Methods 16844
TTextAreaColumn Properties 16844
TTextColumn Class 16849
TTextColumn Members 16850
TTextColumn Methods 16854
TTextColumn Properties 16855
TUndoButton Class 16858
TUndoButton Members 16859
TUndoButton.Create Constructor 16863

TWebButton Class 16863
TWebButton Members 16864
TWebButton.Create Constructor 16868
TWebButton Methods 16868
TWebButton Properties 16870
TWebControlGroup Class 16872
TWebControlGroup Members 16873
TWebControlGroup.Create Constructor 16877
TWebControlGroup.Destroy Destructor 16877
TWebControlGroup Methods 16877
TWebControlGroup Properties 16883
TWebDataDisplay Class 16884
TWebDataDisplay Members 16885
TWebDataDisplay.Create Constructor 16889
TWebDataDisplay.Destroy Destructor 16889
TWebDataDisplay Methods 16889
TWebDataDisplay Properties 16896
TWebDataInput Class 16900
TWebDataInput Members 16901
TWebDataInput Methods 16905
TWebDataInput Properties 16909
TWebDisplay Class 16911
TWebDisplay Members 16911
TWebDisplay.Create Constructor 16912
TWebDisplay Methods 16912
TWebDisplay Properties 16914
TWebDisplay Events 16914
TWebDisplayReferenceGroup Class 16915
TWebDisplayReferenceGroup Members 16915
TWebDisplayReferenceGroup.Create Constructor 16919
TWebDisplayReferenceGroup.Destroy Destructor 16919
TWebDisplayReferenceGroup Methods 16920
TWebDisplayReferenceGroup Properties 16921
TWebForm Class 16922
TWebForm Members 16923
TWebForm.Create Constructor 16926
TWebForm.Destroy Destructor 16926
TWebForm Methods 16927
TWebForm Properties 16932

TWebListInput Class 16935
TWebListInput Members 16936
TWebListInput.Create Constructor 16940
TWebListInput.Destroy Destructor 16941
TWebListInput Methods 16941
TWebListInput Properties 16944
TWebRadioGroupInput Class 16947
TWebRadioGroupInput Members 16948
TWebRadioGroupInput.Create Constructor 16952
TWebRadioGroupInput Methods 16953
TWebRadioGroupInput Properties 16954
TWebSelectOptionsInput Class 16956
TWebSelectOptionsInput Members 16957
TWebSelectOptionsInput.Create Constructor 16961
TWebSelectOptionsInput Methods 16962
TWebSelectOptionsInput Properties 16963
TWebStatus Class 16963
TWebStatus Members 16964
TWebStatus.Create Constructor 16968
TWebStatus Methods 16968
TWebStatus Properties 16970
TWebTextAreaInput Class 16971
TWebTextAreaInput Members 16973
TWebTextAreaInput.Create Constructor 16977
TWebTextAreaInput Methods 16977
TWebTextAreaInput Properties 16978
TWebTextInput Class 16980
TWebTextInput Members 16982
TWebTextInput.Create Constructor 16986
TWebTextInput Methods 16986
TWebTextInput Properties 16987
TXMLButton Class 16989
TXMLButton Members 16990
TXMLButton.Create Constructor 16993
TXMLButton.Destroy Destructor 16994
TXMLButton Methods 16994
TXMLButton Properties 16996
TXMLData Class 16998
TXMLData Members 16998

TXMLData.Create Constructor 16999
TXMLData Methods 17000
TXMLData Properties 17001
TXMLData Events 17002
TXMLDataParent Class 17003
TXMLDataParent Members 17003
TXMLDataParent Methods 17004
TXMLDataParent Properties 17005
TXMLDataSet Class 17006
TXMLDataSet Members 17006
TXMLDataSet.Create Constructor 17008
TXMLDataSet.Destroy Destructor 17008
TXMLDataSet Methods 17008
TXMLDataSet Properties 17012
TXMLDataSetParent Class 17014
TXMLDataSetParent Members 17015
TXMLDataSetParent Methods 17016
TXMLDataSetParent Properties 17018
TXMLDisplayGroup Class 17019
TXMLDisplayGroup Members 17020
TXMLDisplayGroup.Create Constructor 17024
TXMLDisplayGroup.Destroy Destructor 17024
TXMLDisplayGroup Methods 17025
TXMLDisplayGroup Properties 17032
TXMLDisplayParent Class 17034
TXMLDisplayParent Members 17034
TXMLDisplayParent.Create Constructor 17036
TXMLDisplayParent Methods 17036
TXMLDisplayParent Properties 17037
TXMLDisplayReferenceButton Class 17037
TXMLDisplayReferenceButton Members 17039
TXMLDisplayReferenceButton.Create Constructor 17042
TXMLDisplayReferenceButton.Destroy Destructor 17042
TXMLDisplayReferenceButton Methods 17043
TXMLDisplayReferenceButton Properties 17045
TXMLDisplayReferenceGroup Class 17046
TXMLDisplayReferenceGroup Members 17047
TXMLDisplayReferenceGroup Methods 17051
TXMLDisplayReferenceGroup Properties 17051

Interfaces **17053**
IDataSetComponent Interface 17053
IDataSetComponent Members 17054
IDataSetComponent Methods 17054
IDataSetComponent Properties 17055
IDataSetField Interface 17057
IDataSetField Members 17057
IDataSetField Methods 17057
IDataSetField Properties 17058
IDataSetFields Interface 17059
IDataSetFields Members 17059
IDataSetFields Methods 17059
IDataSetFields Properties 17061
IFormatColumn Interface 17061
IFormatColumn Members 17061
IFormatColumn Methods 17062
IHTMLField Interface 17062
IHTMLField Members 17063
IHTMLField Methods 17063
IHTMLField Properties 17063
IHTMLForm Interface 17064
IHTMLForm Members 17064
IHTMLForm Methods 17065
IHTMLForm Properties 17066
IInetXWebDataSet Interface 17066
IInetXWebDataSet Members 17067
IInetXWebDataSet Methods 17067
IQueryField Interface 17070
IQueryField Members 17070
IQueryField Methods 17071
IQueryField Properties 17072
IQueryFields Interface 17073
IQueryFields Members 17073
IQueryFields Methods 17073
IQueryForm Interface 17074
IRestoreDefaults Interface 17074
IRestoreDefaults Members 17075
IRestoreDefaults Methods 17075
IStatusField Interface 17075

IValidateField Interface 17076
IValidateField Members 17076
IValidateField Methods 17076
IValidateFields Interface 17077
IValidateFields Members 17077
IValidateFields Methods 17077
IValidateFields Properties 17078
IWebDisplay Interface 17079
IWebDisplay Members 17079
IWebDisplay Methods 17079
IWebDisplay Properties 17080
IWebDisplayReference Interface 17080
IWebDisplayReference Members 17080
IWebDisplayReference Methods 17081
IWebDisplayReference Properties 17081
IXMLDisplay Interface 17082
IXMLDisplay Members 17082
IXMLDisplay Methods 17082
IXMLDisplay Properties 17083

Functions 17085

MidItems.AddBoolAttrib Function 17085
MidItems.AddCustomAttrib Function 17085
MidItems.AddIntAttrib Function 17086
MidItems.AddQuotedAttrib Function 17086
MidItems.AddStringAttrib Function 17086
MidItems.CanAddClassHelper Function 17087
MidItems.CreateDefaultButtonClasses Function 17087
MidItems.DeclareSubmitForm Function 17087
MidItems.FindDispatcher Function 17088
MidItems.FindProducer Function 17088
MidItems.FormatColumnData Function 17088
MidItems.FormatColumnHeading Function 17089
MidItems.GetItemValuesFromDataSet Function 17089

Structs, Records, Enums 17090

MidItems.TCaptionPosition Enumeration 17090
MidItems.TFormMethod Enumeration 17091
MidItems.TLayoutState Enumeration 17091
MidItems.TTextAreaWrap Enumeration 17092

Types **17093**
MidItems.IMidasWebDataSet Type 17093
MidItems.TWebButtonClass Type 17093
MidItems.TWebDataDisplayClass Type 17094
MidItems.TWebDataInputClass Type 17094
MidItems.TWebStatusClass Type 17094

Constants **17095**
MidItems.FormMethodNames Constant 17095
MidItems.HTMLAlignValues Constant 17096
MidItems.HTMLVAlignValues Constant 17096
MidItems.sActionAttr Constant 17096
MidItems.sAlignAttr Constant 17097
MidItems.sBgColorAttr Constant 17097
MidItems.sBorderAttr Constant 17097
MidItems.sCellPaddingAttr Constant 17098
MidItems.sCellSpacingAttr Constant 17098
MidItems.sCheckedAttr Constant 17098
MidItems.sClassAttr Constant 17099
MidItems.sColsAttr Constant 17099
MidItems.sColSpanAttr Constant 17099
MidItems.sEncTypeAttr Constant 17100
MidItems.sImageHeightAttr Constant 17100
MidItems.sImageWidthAttr Constant 17100
MidItems.sMaxLengthAttr Constant 17101
MidItems.sMethodAttr Constant 17101
MidItems.sMultipleAttr Constant 17101
MidItems.sNameAttr Constant 17102
MidItems.sOnClickAttr Constant 17102
MidItems.sReadOnlyAttr Constant 17102
MidItems.sRowsAttr Constant 17103
MidItems.sSelectedAttr Constant 17103
MidItems.sSizeAttr Constant 17103
MidItems.sStyleAttr Constant 17104
MidItems.sTabIndexAttr Constant 17104
MidItems.sTypeAttr Constant 17104
MidItems.sVAlignAttr Constant 17105
MidItems.sValueAttr Constant 17105
MidItems.sWidthAttr Constant 17105

MidItems.sWrapAttr Constant 17106
MidItems.TextAreaWrap Constant 17106

MidProd Namespace 17107

Classes **17109**
TBaseInetXPageProducer Class 17109
TBaseInetXPageProducer Members 17110
TBaseInetXPageProducer.Create Constructor 17115
TBaseInetXPageProducer.Destroy Destructor 17115
TBaseInetXPageProducer Methods 17116
TBaseInetXPageProducer Properties 17126
TBaseInetXPageProducer Events 17129
TCustomInetXPageProducer Class 17131
TCustomInetXPageProducer Members 17132
TCustomInetXPageProducer.Create Constructor 17137
TCustomInetXPageProducer.Destroy Destructor 17137
TCustomInetXPageProducer Methods 17138
TCustomInetXPageProducer Properties 17140
TInetXPageElements Class 17142
TInetXPageElements Members 17142
TInetXPageElements Fields 17143
TInetXPageElements Methods 17144
TInetXPageProducer Class 17145
TInetXPageProducer Members 17146
TInetXPageProducer Properties 17151
TInetXPageProducer Events 17156
TMidasPageProducer Class 17158

Functions **17160**
MidProd.AddStdIncludes Function 17160
MidProd.PathInfoToRelativePath Function 17160

Structs, Records, Enums **17161**
MidProd.TStdInclude Enumeration 17161

Types **17162**
MidProd.TBaseMidasPageProducer Type 17162
MidProd.TCustomMidasPageProducer Type 17162
MidProd.TMidasPageElements Type 17163
MidProd.TStdIncludes Type 17163

MidProd.TXMLDataEvent Type 17163

Constants **17164**

MidProd.sXMLBroker Constant 17164

MidProd.sXMLReadyVar Constant 17164

MPlayer Namespace 17165

Classes **17167**

EMCIDeviceError Class 17167

TMediaPlayer Class 17167

TMediaPlayer Members 17169

TMediaPlayer.Create Constructor 17180

TMediaPlayer.Destroy Destructor 17180

TMediaPlayer Methods 17181

TMediaPlayer Properties 17196

TMediaPlayer Events 17221

Structs, Records, Enums **17229**

MPlayer.TMPBtnType Enumeration 17229

MPlayer.TMPButton Record 17230

MPlayer.TMPDevCaps Enumeration 17231

MPlayer.TMPDeviceTypes Enumeration 17231

MPlayer.TMPGlyph Enumeration 17233

MPlayer.TMPModes Enumeration 17233

MPlayer.TMPNotifyValues Enumeration 17234

MPlayer.TMPTimeFormats Enumeration 17235

Types **17237**

MPlayer.EMPNotify Type 17237

MPlayer.EMPPostNotify Type 17237

MPlayer.TButtonSet Type 17238

MPlayer.TMPDevCapsSet Type 17238

Mtsobj Namespace 17239

Classes **17240**

TMtsAutoObject Class 17240

TMtsAutoObject Members 17240

TMtsAutoObject Methods 17242

TMtsAutoObject Properties 17247

MtsRdm Namespace 17249

Classes **17250**
TMtsDataModule Class 17250
TMtsDataModule Members 17251
TMtsDataModule.Create Constructor 17254
TMtsDataModule Methods 17255
TMtsDataModule Properties 17265
TMtsDataModule Events 17267

Mtx Namespace 17269

Interfaces **17271**
IEnumNames Interface 17271
IEnumNames Members 17271
IEnumNames Methods 17272
IGetContextProperties Interface 17273
IGetContextProperties Members 17273
IGetContextProperties Methods 17273
IObjectContext Interface 17274
IObjectContext Members 17274
IObjectContext Methods 17275
IObjectContextActivity Interface 17277
IObjectContextActivity Members 17277
IObjectContextActivity Methods 17277
IObjectControl Interface 17278
IObjectControl Members 17278
IObjectControl Methods 17278
ISecurityProperty Interface 17279
ISecurityProperty Members 17280
ISecurityProperty Methods 17280
ISharedProperty Interface 17281
ISharedProperty Members 17281
ISharedProperty Methods 17282
ISharedProperty Properties 17282
ISharedPropertyGroup Interface 17283
ISharedPropertyGroup Members 17283
ISharedPropertyGroup Methods 17284
ISharedPropertyGroup Properties 17285

ISharedPropertyGroupManager Interface 17285
ISharedPropertyGroupManager Members 17286
ISharedPropertyGroupManager Methods 17286
ISharedPropertyGroupManager Properties 17287
ITransactionContext Interface 17288
ITransactionContext Members 17288
ITransactionContext Methods 17288
ITransactionContextEx Interface 17289
ITransactionContextEx Members 17290
ITransactionContextEx Methods 17290

Functions 17292
Mtx.CreateSharedPropertyGroup Function 17292
Mtx.CreateSharedPropertyGroupManager Function 17293
Mtx.CreateTransactionContext Function 17293
Mtx.CreateTransactionContextEx Function 17294
Mtx.GetObjectContext Function 17295
Mtx.SafeRef Function 17295

Constants 17297
Mtx.CLASS_SharedPropertyGroupManager Constant 17297
Mtx.CLASS_TransactionContext Constant 17297
Mtx.CLASS_TransactionContextEx Constant 17298
Mtx.LockMethod Constant 17298
Mtx.LockSetGet Constant 17298
Mtx.mtsErrCtxAborted Constant 17299
Mtx.mtsErrCtxAborting Constant 17299
Mtx.mtsErrCtxActivityTimeout Constant 17299
Mtx.mtsErrCtxNoContext Constant 17300
Mtx.mtsErrCtxNoSecurity Constant 17300
Mtx.mtsErrCtxNotRegistered Constant 17300
Mtx.mtsErrCtxOldReference Constant 17301
Mtx.mtsErrCtxRoleNotFound Constant 17301
Mtx.mtsErrCtxTMNotAvailable Constant 17301
Mtx.mtsErrCtxWrongThread Constant 17302
Mtx.Process Constant 17302
Mtx.Standard Constant 17302

Mxarrays Namespace 17303

Classes 17305

EArrayError Class 17305
EDynArrayRangeError Class 17306
ELowCapacityError Class 17306
EUnsupportedTypeError Class 17307
TBaseArray Class 17307
TBaseArray Members 17308
TBaseArray.Create Constructor 17309
TBaseArray.Destroy Destructor 17310
TBaseArray Methods 17310
TBaseArray Properties 17318
TCurrencyArray Class 17322
TCurrencyArray Members 17323
TCurrencyArray.Create Constructor 17324
TCurrencyArray Methods 17325
TCurrencyArray Properties 17327
TCustomArray Class 17328
TCustomArray Members 17328
TCustomArray.Create Constructor 17329
TCustomArray.Destroy Destructor 17329
TCustomArray Methods 17329
TCustomArray Properties 17334
TDoubleArray Class 17337
TDoubleArray Members 17337
TDoubleArray.Create Constructor 17339
TDoubleArray Methods 17339
TDoubleArray Properties 17342
TIndexArray Class 17342
TIndexArray Members 17342
TIndexArray.Create Constructor 17343
TIndexArray.Destroy Destructor 17343
TIndexArray Fields 17343
TIndexArray Methods 17344
TIndexArray Properties 17345
TIntArray Class 17346
TIntArray Members 17346
TIntArray.Create Constructor 17348
TIntArray Methods 17348
TIntArray Properties 17350
TPointerArray Class 17351

TPointerArray Members 17351
TPointerArray.Create Constructor 17353
TPointerArray Methods 17353
TPointerArray Properties 17355
TSingleArray Class 17356
TSingleArray Members 17356
TSingleArray.Create Constructor 17358
TSingleArray Methods 17358
TSingleArray Properties 17361
TSmallIntArray Class 17361
TSmallIntArray Members 17362
TSmallIntArray.Create Constructor 17363
TSmallIntArray Methods 17364
TSmallIntArray Properties 17366
TStringArray Class 17367
TStringArray Members 17367
TStringArray.Create Constructor 17369
TStringArray Methods 17369
TStringArray Properties 17374
TThreadCustomArray Class 17374
TThreadCustomArray Members 17375
TThreadCustomArray.Create Constructor 17375
TThreadCustomArray.Destroy Destructor 17375
TThreadCustomArray Methods 17375
TTwoDimArray Class 17377
TTwoDimArray Members 17377
TTwoDimArray.Create Constructor 17377
TTwoDimArray.Destroy Destructor 17378
TTwoDimArray Fields 17378
TTwoDimArray Methods 17378
TTwoDimArray Properties 17379
TWordArray Class 17380
TWordArray Members 17380
TWordArray.Create Constructor 17382
TWordArray Methods 17382
TWordArray Properties 17384

Functions **17386**
Mxarrays.GetMemoryCapacity Function 17386
Mxarrays.SetMemoryCapacity Function 17386

Structs, Records, Enums **17387**
Mxarrays.TArrayFlags Enumeration 17387
Mxarrays.TDuplicates Enumeration 17387
Mxarrays.TSortOrder Enumeration 17388
Mxarrays.TStringItem Record 17388

Types **17390**
Mxarrays.PIndexElements Type 17390
Mxarrays.PMatrixElements Type 17390
Mxarrays.TArrayFlagSet Type 17391
Mxarrays.TCompareProc Type 17391
Mxarrays.TDynArrayNDX Type 17391
Mxarrays.TIndexElements Type 17392
Mxarrays.TIndexNDX Type 17392
Mxarrays.TMatrixElements Type 17392
Mxarrays.TMatrixNDX Type 17393

Constants **17394**
Mxarrays.vMaxCol Constant 17394
Mxarrays.vMaxRow Constant 17394

MXCOMMON Namespace 17395

Classes **17397**
EDimensionMapError Class 17397
TBinData Class 17397
TBinData Members 17398
TBinData.create Constructor 17399
TBinData.destroy Destructor 17399
TBinData Methods 17399
TBinData Properties 17404
TDimensionItem Class 17405
TDimensionItem Members 17406
TDimensionItem.Create Constructor 17408
TDimensionItem Methods 17408
TDimensionItem Properties 17410
TDimensionItems Class 17415
TDimensionItems Members 17415
TDimensionItems.Create Constructor 17417
TDimensionItems Methods 17417

TDimensionItems Properties 17421
TTicks Class 17424
TTicks Members 17424
TTicks.Create Constructor 17424
TTicks.Destroy Destructor 17425
TTicks Methods 17425
TTicks Properties 17426

Functions 17427
MXCOMMON.AddToQuerySelect Function 17427
MXCOMMON.BuildDataSetMap Function 17427
MXCOMMON.CheckIfEmptyQuery Function 17428
MXCOMMON.FieldTypeToVarType Function 17428
MXCOMMON.FormatVariant Function 17428
MXCOMMON.GetAggName Function 17429
MXCOMMON.HookToDataBase Function 17429
MXCOMMON.IsAggValid Function 17429
MXCOMMON.IsBDEAvailable Function 17430
MXCOMMON.NextArg Function 17430
MXCOMMON.UpdateDesigner Function 17430
MXCOMMON.VerifyRTQuery Function 17431

Structs, Records, Enums 17432
MXCOMMON.TActiveFlags Enumeration 17432
MXCOMMON.TBinType Enumeration 17432
MXCOMMON.TCDNotifyType Enumeration 17433
MXCOMMON.TDimFlags Enumeration 17433
MXCOMMON.TQueryError Enumeration 17434
MXCOMMON.TTickInfoRec Record 17434

Types 17436
MXCOMMON.PTickInfoRec Type 17436
MXCOMMON.TBinTypes Type 17436
MXCOMMON.TDimensionItemClass Type 17436

MXDB Namespace 17439

Classes 17441
EArrayError Class 17441
EDimIndexError Class 17441
TDecisionCube Class 17442

TDecisionCube Members 17443
TDecisionCube.Create Constructor 17447
TDecisionCube.Destroy Destructor 17447
TDecisionCube Methods 17447
TDecisionCube Properties 17450
TDecisionCube Events 17455
TDecisionDataLink Class 17458
TDecisionDataLink Members 17458
TDecisionDataLink.Create Constructor 17459
TDecisionDataLink.Destroy Destructor 17460
TDecisionDataLink Fields 17460
TDecisionDataLink Methods 17461
TDecisionDataLink Properties 17461
TDecisionSource Class 17462
TDecisionSource Members 17463
TDecisionSource.Create Constructor 17467
TDecisionSource.Destroy Destructor 17467
TDecisionSource Fields 17467
TDecisionSource Methods 17469
TDecisionSource Properties 17485
TDecisionSource Events 17493
TDimInfoArray Class 17497
TDimInfoArray Members 17497
TDimInfoArray.Create Constructor 17498
TDimInfoArray.Destroy Destructor 17498
TDimInfoArray Methods 17498
TDimInfoArray Properties 17500
TIArray Class 17501
TIArray Members 17501
TIArray.Create Constructor 17502
TIArray.Destroy Destructor 17502
TIArray Methods 17502
TIArray Properties 17503
TPivotState Class 17504
TPivotState Members 17505
TPivotState.Create Constructor 17505
TPivotState.Destroy Destructor 17505
TPivotState Fields 17506
TPivotState Methods 17508

TQADecisionSource Class 17508

Structs, Records, Enums 17510

MXDB.TDecisionControlType Enumeration 17510
MXDB.TDecisionDataEvent Enumeration 17510
MXDB.TDimGroup Enumeration 17511
MXDB.TDimInfo Record 17511
MXDB.TDimRange Record 17512
MXDB.TDimState Enumeration 17512
MXDB.TRowStates Enumeration 17513

Types 17514

MXDB.PArrayDimInfo Type 17514
MXDB.PArrayInt Type 17514
MXDB.PDimInfo Type 17515
MXDB.TArrayDimInfo Type 17515
MXDB.TArrayInt Type 17515
MXDB.TBigStr Type 17516
MXDB.TRowState Type 17516

MXGRAPH Namespace 17517

Classes 17518

TCustomDecisionGraph Class 17518
TCustomDecisionGraph Members 17518
TCustomDecisionGraph.Create Constructor 17519
TCustomDecisionGraph.Destroy Destructor 17519
TCustomDecisionGraph Methods 17520
TCustomDecisionGraph Properties 17520
TDecisionGraph Class 17521
TDecisionGraph Members 17522
TDecisionGraph Properties 17524
TDecisionGraphDataLink Class 17546
TDecisionGraphDataLink Members 17546
TDecisionGraphDataLink.Create Constructor 17547
TDecisionGraphDataLink.Destroy Destructor 17548
TDecisionGraphDataLink Methods 17548

Structs, Records, Enums 17549

MXGRAPH.TSeriesType Enumeration 17549

Types 17550

MXGRAPH.TDimEvent Type 17550

MXGRID Namespace 17551

Classes 17553
TCustomDecisionGrid Class 17553
TCustomDecisionGrid Members 17554
TCustomDecisionGrid.Create Constructor 17566
TCustomDecisionGrid.Destroy Destructor 17566
TCustomDecisionGrid Fields 17567
TCustomDecisionGrid Methods 17567
TCustomDecisionGrid Properties 17574
TCustomDecisionGrid Events 17584
TDecisionGrid Class 17586
TDecisionGrid Members 17587
TDecisionGrid Properties 17600
TDecisionGrid Events 17630
TDecisionGridDataLink Class 17658
TDecisionGridDataLink Members 17658
TDecisionGridDataLink.Create Constructor 17660
TDecisionGridDataLink.Destroy Destructor 17660
TDecisionGridDataLink Methods 17660
TDisplayDim Class 17661
TDisplayDim Members 17661
TDisplayDim.Create Constructor 17662
TDisplayDim Methods 17663
TDisplayDim Properties 17664
TDisplayDims Class 17666
TDisplayDims Members 17667
TDisplayDims Fields 17669
TDisplayDims Methods 17669
TDisplayDims Properties 17670
TValueArray Class 17671

Structs, Records, Enums 17672
MXGRID.TButtonPlace Enumeration 17672
MXGRID.TDDNotifyType Enumeration 17672
MXGRID.TDecisionCellType Enumeration 17673
MXGRID.TDecisionDrawStates Enumeration 17673
MXGRID.TDecisionGridOption Enumeration 17674

MXGRID.TDecisionGridState Enumeration 17675
MXGRID.TDecisionPivotState Enumeration 17675

Types 17676
MXGRID.TDecisionDrawCellEvent Type 17676
MXGRID.TDecisionDrawState Type 17676
MXGRID.TDecisionExamineCellEvent Type 17677
MXGRID.TDecisionGridOptions Type 17677
MXGRID.TDisplayDimClass Type 17677

MXPIVSRC Namespace 17679

Classes 17680
TDecisionPivot Class 17680
TDecisionPivot Members 17681
TDecisionPivot.Create Constructor 17691
TDecisionPivot.Destroy Destructor 17692
TDecisionPivot Methods 17692
TDecisionPivot Properties 17696
TDecisionPivot Events 17724
TPivotDataLink Class 17737
TPivotDataLink Members 17738
TPivotDataLink.Create Constructor 17739
TPivotDataLink.Destroy Destructor 17739
TPivotDataLink Methods 17739

Structs, Records, Enums 17741
MXPIVSRC.TDecisionButtonPosition Enumeration 17741
MXPIVSRC.TDecisionPivotOption Enumeration 17741

Types 17743
MXPIVSRC.TDecisionPivotOptions Type 17743

Mxstore Namespace 17745

Classes 17747
ECacheError Class 17747
TAggDefinition Class 17748
TAggDefinition Members 17748
TAggDefinition Properties 17748
TBuilderDim Class 17749
TBuilderDim Members 17749

TBuilderDim.Create Constructor 17750
TBuilderDim.Destroy Destructor 17750
TBuilderDim Methods 17751
TBuilderDim Properties 17753
TCubeDim Class 17754
TCubeDim Members 17754
TCubeDim.Create Constructor 17756
TCubeDim.Destroy Destructor 17757
TCubeDim Methods 17757
TCubeDim Properties 17762
TCubeDim Events 17766
TCubeDims Class 17766
TCubeDims Members 17767
TCubeDims.Create Constructor 17769
TCubeDims Methods 17769
TCubeDims Properties 17771
TCustomDataStore Class 17772
TCustomDataStore Members 17772
TCustomDataStore.Create Constructor 17776
TCustomDataStore.Destroy Destructor 17777
TCustomDataStore Methods 17777
TCustomDataStore Properties 17786
TCustomDataStore Events 17795
TDataCache Class 17797
TDataCache Members 17798
TDataCache.Create Constructor 17799
TDataCache.Destroy Destructor 17799
TDataCache Methods 17799
TDataCache Properties 17806
TDimension Class 17809
TDimension Members 17809
TDimension.Create Constructor 17810
TDimension.Destroy Destructor 17811
TDimension Methods 17811
TDimension Properties 17812
TFieldDefinition Class 17814
TFieldDefinition Members 17815
TFieldDefinition.Create Constructor 17815
TFieldDefinition Methods 17815

TFieldDefinition Properties 17816
TIndexInfo Class 17818
TIndexInfo Members 17818
TIndexInfo.Create Constructor 17818
TIndexInfo.Destroy Destructor 17819
TIndexInfo Methods 17819
TIndexInfo Properties 17821
TMultiDimDataLink Class 17822
TMultiDimDataLink Members 17822
TMultiDimDataLink.Create Constructor 17824
TMultiDimDataLink.Destroy Destructor 17824
TMultiDimDataLink Methods 17825
TSummary Class 17828
TSummary Members 17828
TSummary.Create Constructor 17829
TSummary.Destroy Destructor 17829
TSummary Methods 17830
TSummary Properties 17832
TSummary Events 17834

Functions **17836**
Mxstore.TestMatch Function 17836

Structs, Records, Enums **17837**
Mxstore.TBuildType Enumeration 17837
Mxstore.TCacheStateFlags Enumeration 17837
Mxstore.TCalcTotalsFlags Enumeration 17838
Mxstore.TCubeDataState Enumeration 17838
Mxstore.TCubeState Enumeration 17839
Mxstore.TErrorAction Enumeration 17839
Mxstore.TFormatType Enumeration 17840
Mxstore.TIndexFlag Enumeration 17840
Mxstore.TIndexInfoRec Record 17841
Mxstore.TLookupStateFlags Enumeration 17841

Types **17843**
Mxstore.PIndexInfoRec Type 17843
Mxstore.TAggProc Type 17843
Mxstore.TCacheState Type 17844
Mxstore.TCalcTotals Type 17844
Mxstore.TCapacityErrorEvent Type 17844

Mxstore.TCubeDimClass Type 17845
Mxstore.TCubeDimTransformEvent Type 17845
Mxstore.TCubeNotifyEvent Type 17845
Mxstore.TCubeRefreshEvent Type 17846
Mxstore.TDerivedAggProc Type 17846
Mxstore.TDimFlagSet Type 17846
Mxstore.TFieldArgs Type 17847
Mxstore.TIndexFlags Type 17847
Mxstore.TLookupState Type 17847
Mxstore.TSumMethod Type 17848

Constants **17849**
Mxstore.LargeValueCount Constant 17849
Mxstore.MaxBinDimensions Constant 17849
Mxstore.NonSparseAgg Constant 17850
Mxstore.SparseAgg Constant 17850
Mxstore.SparseSum Constant 17850
Mxstore.SparseUnknown Constant 17851
Mxstore.SubTotal Constant 17851

MXTABLES Namespace 17853

Classes **17854**
TDecisionQuery Class 17854
TQueryDim Class 17855
TQueryDim Members 17855
TQueryDim Properties 17857
TQueryDims Class 17857
TQueryDims Members 17857
TQueryDims.Create Constructor 17859
TQueryDims Methods 17860
TQueryDims Properties 17861

Functions **17862**
MXTABLES.AddDimensionItem Function 17862
MXTABLES.BuildDimensionMap Function 17862
MXTABLES.BuildMasterDimensionMap Function 17863
MXTABLES.RemoveDimensionItem Function 17863

Types **17864**
MXTABLES.TQueryDimClass Type 17864

ObjBrkr Namespace 17865

Classes **17866**
EBrokerException Class 17866
TServerCollection Class 17866
TServerCollection Members 17867
TServerCollection.Create Constructor 17869
TServerCollection Methods 17869
TServerCollection Properties 17870
TServerItem Class 17871
TServerItem Members 17872
TServerItem.Create Constructor 17873
TServerItem Methods 17874
TServerItem Properties 17874
TSimpleObjectBroker Class 17877
TSimpleObjectBroker Members 17877
TSimpleObjectBroker.Create Constructor 17880
TSimpleObjectBroker.Destroy Destructor 17880
TSimpleObjectBroker Methods 17881
TSimpleObjectBroker Properties 17884

OleAuto Namespace 17887

Classes **17889**
EOleError Class 17889
EOleException Class 17889
EOleException Members 17890
EOleException.Create Constructor 17891
EOleException Properties 17891
EOleSysError Class 17892
EOleSysError Members 17892
EOleSysError.Create Constructor 17893
EOleSysError Properties 17893
TAutoDispatch Class 17894
TAutoDispatch Members 17894
TAutoDispatch.Create Constructor 17895
TAutoDispatch Methods 17895
TAutoDispatch Properties 17897
TAutomation Class 17897

TAutomation Members 17898
TAutomation.Create Constructor 17898
TAutomation.Destroy Destructor 17898
TAutomation Methods 17899
TAutomation Properties 17899
TAutomation Events 17900
TAutoObject Class 17901
TAutoObject Members 17901
TAutoObject.Create Constructor 17902
TAutoObject.Destroy Destructor 17902
TAutoObject Methods 17903
TAutoObject Properties 17904
TRegistryClass Class 17905
TRegistryClass Members 17905
TRegistryClass.Create Constructor 17905
TRegistryClass.Destroy Destructor 17906
TRegistryClass Methods 17906

Functions 17907

OleAuto.ClassIDToProgID Function 17907
OleAuto.ClassIDToString Function 17907
OleAuto.CreateOleObject Function 17908
OleAuto.DispInvoke Function 17908
OleAuto.DispInvokeError Function 17908
OleAuto.DllCanUnloadNow Function 17909
OleAuto.DllGetClassObject Function 17909
OleAuto.DllRegisterServer Function 17909
OleAuto.DllUnregisterServer Function 17910
OleAuto.GetActiveOleObject Function 17910
OleAuto.OleCheck Function 17910
OleAuto.OleError Function 17911
OleAuto.ProgIDToClassID Function 17911
OleAuto.StringToClassID Function 17911
OleAuto.VarFromInterface Function 17912
OleAuto.VarToAutoObject Function 17912
OleAuto.VarToInterface Function 17912

Structs, Records, Enums 17913

OleAuto.TAutoClassInfo Record 17913
OleAuto.TAutoClassInstancing Enumeration 17913

OleAuto.TStartMode Enumeration 17914

Types **17915**

OleAuto.TAutoClass Type 17915

OleAuto.TLastReleaseEvent Type 17915

Variables **17916**

OleAuto.Automation Variable 17916

Constants **17917**

OleAuto.MaxDispArgs Constant 17917

OleCtnrs Namespace 17919

Classes **17921**

TOleContainer Class 17921

TOleContainer Members 17922

TOleContainer.Create Constructor 17933

TOleContainer.Destroy Destructor 17934

TOleContainer Methods 17934

TOleContainer Properties 17950

TOleContainer Events 17980

TOleForm Class 18006

TOleForm Members 18007

TOleForm.Create Constructor 18008

TOleForm.Destroy Destructor 18008

Interfaces **18009**

IVCLFrameForm Interface 18009

IVCLFrameForm Members 18009

IVCLFrameForm Methods 18009

Functions **18011**

OleCtnrs.DestroyMetaPict Function 18011

Structs, Records, Enums **18012**

OleCtnrs.TAutoActivate Enumeration 18012

OleCtnrs.TCreateInfo Record 18012

OleCtnrs.TCreateType Enumeration 18013

OleCtnrs.TObjectState Enumeration 18013

OleCtnrs.TSizeMode Enumeration 18014

OleCtnrs.TVerbInfo Record 18014

Types **18016**

OleCtnrs.TObjectMoveEvent Type 18016

Constants 18017

OleCtnrs.ovDiscardUndoState Constant 18017
OleCtnrs.ovHide Constant 18017
OleCtnrs.ovInPlaceActivate Constant 18018
OleCtnrs.ovOpen Constant 18018
OleCtnrs.ovPrimary Constant 18018
OleCtnrs.ovShow Constant 18019
OleCtnrs.ovUIActivate Constant 18019

OleCtrls Namespace 18021

Classes 18023

EOleCtrlError Class 18023
TEnumPropDesc Class 18023
TEnumPropDesc Members 18024
TEnumPropDesc.Create Constructor 18024
TEnumPropDesc.Destroy Destructor 18025
TEnumPropDesc Methods 18025
TEventDispatch Class 18026
TEventDispatch Members 18027
TEventDispatch.Create Constructor 18028
TEventDispatch Methods 18028
TEventDispatch Properties 18030
TOleControl Class 18030
TOleControl Members 18031
TOleControl.Create Constructor 18043
TOleControl.Destroy Destructor 18043
TOleControl Methods 18044
TOleControl Properties 18085
TOleControl Events 18087

Functions 18088

OleCtrls.FontToOleFont Function 18088
OleCtrls.OleFontToFont Function 18088

Structs, Records, Enums 18090

OleCtrls.TControlData Record 18090
OleCtrls.TControlData2 Record 18091
OleCtrls.TEnumValue Record 18091

Types **18093**
OleCtrls.PControlData Type 18093
OleCtrls.PControlData2 Type 18093
OleCtrls.PEnumValueList Type 18094
OleCtrls.TEnumValueList Type 18094
OleCtrls.TOleEnum Type 18094
OleCtrls.TServiceQuery Type 18095

Variables **18096**
OleCtrls.EmptyParam Variable 18096

OleServer Namespace 18097

Classes **18098**
TOleServer Class 18098
TOleServer Members 18099
TOleServer.Create Constructor 18101
TOleServer.Destroy Destructor 18102
TOleServer Methods 18103
TOleServer Properties 18109
TServerEventDispatch Class 18112
TServerEventDispatch Members 18112
TServerEventDispatch.Create Constructor 18113
TServerEventDispatch Methods 18113
TServerEventDispatch Properties 18115

Structs, Records, Enums **18117**
OleServer.TConnectKind Enumeration 18117
OleServer.TServerData Record 18117

Types **18119**
OleServer.PServerData Type 18119
OleServer.TVariantArray Type 18119

OPConvert Namespace 18121

Interfaces **18122**
IOPConvert Interface 18122
IOPConvert Members 18123
IOPConvert Methods 18123
IOPConvert Properties 18130

Structs, Records, Enums **18132**
OPConvert.TSOAPConvertOption Enumeration 18132
Types **18133**
OPConvert.TSOAPConvertOptions Type 18133

OPToSOAPDomConv Namespace 18135

Classes **18137**
ESOAPDomConvertError Class 18137
TOPToSoapDomConvert Class 18137
TOPToSoapDomConvert Members 18138
TOPToSoapDomConvert.Create Constructor 18143
TOPToSoapDomConvert.Destroy Destructor 18143
TOPToSoapDomConvert Methods 18144
TOPToSoapDomConvert Properties 18148
TSOAPDomConv Class 18150
TSOAPDomConv Members 18151
TSOAPDomConv.Create Constructor 18155
TSOAPDomConv.Destroy Destructor 18156
TSOAPDomConv Methods 18156
TSOAPDomConv Properties 18176
TSOAPDomConv Events 18176
Functions **18178**
OPToSOAPDomConv.GetOrdPropEx Function 18178
Structs, Records, Enums **18179**
OPToSOAPDomConv.ConvNodeState Enumeration 18179
OPToSOAPDomConv.TMultiRefNodeElem Record 18179
OPToSOAPDomConv.TMultiRefNodeMapElem Record 18180
OPToSOAPDomConv.TSOAPArrayElemDesc Record 18180
Types **18182**
OPToSOAPDomConv.TMemberDataNotReceivedEvent Type 18182
OPToSOAPDomConv.TMultiRefNodeMap Type 18182
OPToSOAPDomConv.TMultiRefNodes Type 18183
OPToSOAPDomConv.TSOAPArrayDesc Type 18183
OPToSOAPDomConv.TUnhandledNodeEvent Type 18183
OPToSOAPDomConv.TXMLNodeArray Type 18184
Variables **18185**

OPToSOAPDomConv.DefArrayElemName Variable 18185

Constants **18186**

OPToSOAPDomConv.SVarArrayType Constant 18186

OrbPas Namespace 18187

Interfaces **18189**

IBOA Interface 18189

IBOA Members 18189

IBOA Methods 18190

ICorbaObj Interface 18191

ICorbaObj Members 18191

ICorbaObj Methods 18191

IMarshalInBuffer Interface 18192

IMarshalInBuffer Members 18192

IMarshalInBuffer Methods 18193

IMarshalOutBuffer Interface 18196

IMarshalOutBuffer Members 18197

IMarshalOutBuffer Methods 18197

IORB Interface 18201

IORB Members 18201

IORB Methods 18201

ISkeleton Interface 18205

ISkeleton Members 18205

ISkeleton Methods 18206

ISkeletonObject Interface 18206

ISkeletonObject Members 18207

ISkeletonObject Methods 18207

IStub Interface 18208

IStub Members 18208

IStub Methods 18208

IStubObject Interface 18211

IStubObject Members 18211

IStubObject Methods 18211

ITypeCode Interface 18212

ITypeCode Members 18212

ITypeCode Methods 18213

Functions **18217**

OrbPas.AnyToVariant Function 18217

OrbPas.BindStub Function 18217
OrbPas.CorbaAnyType Function 18218
OrbPas.CorbaDuplicateAny Function 18218
OrbPas.CorbaReleaseAny Function 18218
OrbPas.CorbaStringFree Function 18219
OrbPas.CorbaWStringFree Function 18219
OrbPas.CreateSkeleton Function 18219
OrbPas.CreateUserException Function 18220
OrbPas.InitORB Function 18220
OrbPas.RegisterUserException Function 18220
OrbPas.SequenceToVariant Function 18221
OrbPas.UnRegisterUserException Function 18221
OrbPas.VariantToAny Function 18221

Structs, Records, Enums **18222**
OrbPas.TCKind Enumeration 18222
OrbPas.TStructMember Record 18223

Types **18224**
OrbPas.CorbaBoolean Type 18224
OrbPas.CorbaULong Type 18224
OrbPas.PCorbaAny Type 18225
OrbPas.PCorbaInterfaceDef Type 18225
OrbPas.PCorbaObject Type 18225
OrbPas.PCorbaOrb Type 18226
OrbPas.PExceptionDescription Type 18226
OrbPas.PMarshalInbuffer Type 18226
OrbPas.PMarshalOutbuffer Type 18227
OrbPas.PUserExceptionProxy Type 18227
OrbPas.TArgv Type 18227
OrbPas.TCopyUserExceptionProc Type 18228
OrbPas.TStructMembers Type 18228
OrbPas.TThrowUserExceptionProc Type 18228
OrbPas.TUserExceptionFactoryProc Type 18229

Outline Namespace 18231

Classes **18233**
EOutlineError Class 18233
OutlineError Class 18234
TCustomOutline Class 18234

TCustomOutline Members 18235
TCustomOutline.Create Constructor 18248
TCustomOutline.Destroy Destructor 18248
TCustomOutline Methods 18249
TCustomOutline Properties 18268
TCustomOutline Events 18278
TOutline Class 18279
TOutline Members 18280
TOutline Properties 18293
TOutline Events 18319
TOutlineNode Class 18349
TOutlineNode Members 18350
TOutlineNode.Create Constructor 18351
TOutlineNode.Destroy Destructor 18352
TOutlineNode Methods 18352
TOutlineNode Properties 18360

Structs, Records, Enums 18366
Outline.TAttachMode Enumeration 18366
Outline.TOutlineBitmap Enumeration 18367
Outline.TOutlineNodeCompare Enumeration 18367
Outline.TOutlineOption Enumeration 18368
Outline.TOutlineStyle Enumeration 18368
Outline.TOutlineType Enumeration 18369

Types 18371
Outline.EOutlineChange Type 18371
Outline.TBitmapArray Type 18371
Outline.TBitmapArrayRange Type 18372
Outline.TChangeRange Type 18372
Outline.TOutlineBitmaps Type 18372
Outline.TOutlineOptions Type 18373

PagItems Namespace 18375

Classes 18376
TBasePageItemsProducer Class 18376
TBasePageItemsProducer Members 18377
TBasePageItemsProducer.Create Constructor 18380
TBasePageItemsProducer.Destroy Destructor 18381
TBasePageItemsProducer Methods 18381

TBasePageItemsProducer Properties 18382
TPageItemsProducer Class 18383
TPageItemsProducer Members 18384
TPageItemsProducer.Create Constructor 18388
TPageItemsProducer.Destroy Destructor 18388
TPageItemsProducer Methods 18388
TPageItemsProducer Properties 18389

Printers Namespace 18391

Classes 18392
EPrinter Class 18392
TPrinter Class 18392
TPrinter Members 18393
TPrinter.Create Constructor 18394
TPrinter.Destroy Destructor 18395
TPrinter Methods 18395
TPrinter Properties 18402

Functions 18412
Printers.AssignPrn Function 18412
Printers.Printer Function 18412
Printers.SetPrinter Function 18414

Structs, Records, Enums 18416
Printers.TPrinterCapability Enumeration 18416
Printers.TPrinterOrientation Enumeration 18416
Printers.TPrinterState Enumeration 18417

Types 18418
Printers.TPrinterCapabilities Type 18418

Provider Namespace 18419

Classes 18421
EDSWriter Class 18421
EDSWriter Members 18422
EDSWriter.Create Constructor 18423
EDSWriter Properties 18423
TBaseProvider Class 18423
TBaseProvider Members 18425
TBaseProvider.Create Constructor 18429

TBaseProvider.Destroy Destructor 18430
TBaseProvider Fields 18430
TBaseProvider Methods 18432
TBaseProvider Properties 18439
TBaseProvider Events 18442
TCustomPacketWriter Class 18445
TCustomPacketWriter Members 18445
TCustomPacketWriter.Create Constructor 18446
TCustomPacketWriter.Destroy Destructor 18446
TCustomPacketWriter Fields 18447
TCustomPacketWriter Methods 18448
TCustomPacketWriter Properties 18449
TCustomProvider Class 18449
TCustomProvider Members 18450
TCustomProvider.Create Constructor 18454
TCustomProvider.Destroy Destructor 18454
TCustomProvider Fields 18455
TCustomProvider Methods 18458
TCustomProvider Properties 18473
TCustomProvider Events 18475
TCustomResolver Class 18483
TCustomResolver Members 18484
TCustomResolver.Create Constructor 18487
TCustomResolver.Destroy Destructor 18487
TCustomResolver Fields 18488
TCustomResolver Methods 18489
TCustomResolver Properties 18496
TDataPacketWriter Class 18497
TDataPacketWriter Members 18497
TDataPacketWriter.Destroy Destructor 18499
TDataPacketWriter Fields 18500
TDataPacketWriter Methods 18501
TDataPacketWriter Properties 18507
TDataPacketWriter Events 18508
TDataSetProvider Class 18508
TDataSetProvider Members 18509
TDataSetProvider.Create Constructor 18514
TDataSetProvider.Destroy Destructor 18514
TDataSetProvider Fields 18515

TDataSetProvider Methods 18517
TDataSetProvider Properties 18528
TDataSetProvider Events 18533
TDataSetResolver Class 18544
TDataSetResolver Members 18544
TDataSetResolver.Create Constructor 18548
TDataSetResolver Fields 18548
TDataSetResolver Methods 18549
TDataSetResolver Properties 18553
TLocalAppServer Class 18553
TLocalAppServer Members 18554
Create Constructor 18555
TLocalAppServer.Destroy Destructor 18556
TLocalAppServer Fields 18556
TLocalAppServer Methods 18557
TPacketDataSet Class 18564
TPacketDataSet Members 18565
TPacketDataSet.Create Constructor 18580
TPacketDataSet Fields 18580
TPacketDataSet Methods 18582
TPacketDataSet Properties 18587
TProvider Class 18589
TSQLResolver Class 18589
TSQLResolver Members 18590
TSQLResolver.Create Constructor 18593
TSQLResolver.Destroy Destructor 18594
TSQLResolver Fields 18594
TSQLResolver Methods 18595
TSQLResolver Properties 18600
TUpdateTree Class 18600
TUpdateTree Members 18601
TUpdateTree.Create Constructor 18603
TUpdateTree.Destroy Destructor 18603
TUpdateTree Fields 18603
TUpdateTree Methods 18606
TUpdateTree Properties 18609

Interfaces **18613**
IProviderContainer Interface 18613
IProviderContainer Members 18613

IProviderContainer Methods 18613

Functions **18615**

Provider.GetObjectProperty Function 18615

Provider.GetStringProperty Function 18615

Provider.VarArrayFromStrings Function 18615

Provider.VarArrayFromWideStrings Function 18616

Structs, Records, Enums **18617**

Provider.TGetRecordOption Enumeration 18617

Provider.TProviderOption Enumeration 18618

Provider.TPutFieldInfo Record 18620

Provider.TResolverResponse Enumeration 18620

Types **18622**

Provider.PPutFieldInfo Type 18622

Provider.TAfterUpdateRecordEvent Type 18622

Provider.TBeforeCommitEvent Type 18623

Provider.TBeforeUpdateRecordEvent Type 18623

Provider.TDataRequestEvent Type 18623

Provider.TGetDSProps Type 18624

Provider.TGetParamsEvent Type 18624

Provider.TGetRecordOptions Type 18624

Provider.TGetTableNameEvent Type 18625

Provider.TInfoArray Type 18625

Provider.TOnValidate Type 18625

Provider.TProviderDataEvent Type 18626

Provider.TProviderOptions Type 18626

Provider.TPutFieldProc Type 18626

Provider.TResolverErrorEvent Type 18627

Variables **18628**

Provider.InformixLob Variable 18628

Constants **18629**

Provider.MetaDataOption Constant 18629

Provider.ResetOption Constant 18630

Provider.XMLOption Constant 18630

Provider.XMLUTF8Option Constant 18631

Registry Namespace 18633

Classes **18634**
ERegistryException Class 18634
TRegIniFile Class 18635
TRegIniFile Members 18635
Create Constructor 18638
TRegIniFile Methods 18638
TRegIniFile Properties 18644
TRegistry Class 18645
TRegistry Members 18645
Create Constructor 18647
TRegistry.Destroy Destructor 18648
TRegistry Methods 18649
TRegistry Properties 18672
TRegistryIniFile Class 18675
TRegistryIniFile Members 18676
Create Constructor 18677
TRegistryIniFile.Destroy Destructor 18678
TRegistryIniFile Methods 18678
TRegistryIniFile Properties 18689

Structs, Records, Enums **18691**
Registry.TRegDataInfo Record 18691
Registry.TRegDataType Enumeration 18691
Registry.TRegKeyInfo Record 18692

ReqFiles Namespace 18695

Classes **18696**
TWebRequestFile Class 18696
TWebRequestFile Members 18697
TWebRequestFile.Create Constructor 18698
TWebRequestFile.Destroy Destructor 18698
TWebRequestFile Methods 18699
TWebRequestFiles Class 18700
TWebRequestFiles Members 18701
TWebRequestFiles.Create Constructor 18702
TWebRequestFiles.Destroy Destructor 18702
TWebRequestFiles Methods 18702
TWebRequestFileStream Class 18704
TWebRequestFileStream Members 18705

TWebRequestFileStream.Create Constructor 18706
TWebRequestFileStream Methods 18706

ReqMulti Namespace 18709

Classes 18710
TMultipartContentParser Class 18710
TMultipartContentParser Members 18711
TMultipartContentParser.Destroy Destructor 18712
TMultipartContentParser Methods 18712

Rio Namespace 18717

Classes 18718
TRIO Class 18718
TRIO Members 18719
TRIO.Create Constructor 18722
TRIO.Destroy Destructor 18722
TRIO Fields 18723
TRIO Methods 18724
TRIO Properties 18727
TRIO Events 18729

Interfaces 18733
IRIOAccess Interface 18733
IRIOAccess Members 18733
IRIOAccess Methods 18734
IRIOAccess Properties 18734

Types 18735
Rio.TAfterExecuteEvent Type 18735
Rio.TBeforeExecuteEvent Type 18735

Constants 18736
Rio.StubSize Constant 18736

ScktComp Namespace 18737

Classes 18739
ESocketError Class 18739
TAbstractSocket Class 18740
TAbstractSocket Members 18741

TAbstractSocket Methods 18744
TAbstractSocket Properties 18749
TClientSocket Class 18752
TClientSocket Members 18753
TClientSocket.Create Constructor 18756
TClientSocket.Destroy Destructor 18757
TClientSocket Methods 18757
TClientSocket Properties 18758
TClientSocket Events 18762
TClientWinSocket Class 18766
TClientWinSocket Members 18766
TClientWinSocket Methods 18768
TClientWinSocket Properties 18769
TCustomServerSocket Class 18770
TCustomServerSocket Members 18770
TCustomServerSocket.Destroy Destructor 18774
TCustomServerSocket Fields 18774
TCustomServerSocket Methods 18775
TCustomServerSocket Properties 18782
TCustomServerSocket Events 18783
TCustomSocket Class 18789
TCustomSocket Members 18790
TCustomSocket Methods 18793
TCustomSocket Events 18794
TCustomWinSocket Class 18799
TCustomWinSocket Members 18800
TCustomWinSocket.Create Constructor 18801
TCustomWinSocket.Destroy Destructor 18802
TCustomWinSocket Methods 18802
TCustomWinSocket Properties 18816
TCustomWinSocket Events 18824
TServerAcceptThread Class 18825
TServerAcceptThread Members 18825
TServerAcceptThread.Create Constructor 18826
TServerAcceptThread Methods 18826
TServerAcceptThread Properties 18830
TServerClientThread Class 18831
TServerClientThread Members 18832
TServerClientThread.Create Constructor 18833

TServerClientThread.Destroy Destructor 18833
TServerClientThread Methods 18834
TServerClientThread Properties 18838
TServerClientWinSocket Class 18840
TServerClientWinSocket Members 18840
TServerClientWinSocket.Create Constructor 18842
TServerClientWinSocket.Destroy Destructor 18842
TServerClientWinSocket Properties 18843
TServerSocket Class 18843
TServerSocket Members 18844
TServerSocket.Create Constructor 18848
TServerSocket Properties 18848
TServerSocket Events 18852
TServerWinSocket Class 18858
TServerWinSocket Members 18859
TServerWinSocket.Create Constructor 18861
TServerWinSocket.Destroy Destructor 18861
TServerWinSocket Methods 18861
TServerWinSocket Properties 18867
TServerWinSocket Events 18870
TWinSocketStream Class 18876
TWinSocketStream Members 18877
TWinSocketStream.Create Constructor 18878
TWinSocketStream.Destroy Destructor 18879
TWinSocketStream Methods 18879
TWinSocketStream Properties 18882

Functions **18883**
ScktComp.SetErrorProc Function 18883

Structs, Records, Enums **18884**
ScktComp.TAsyncStyle Enumeration 18884
ScktComp.TClientType Enumeration 18885
ScktComp.TCMLookupComplete Record 18885
ScktComp.TCMSocketMessage Record 18886
ScktComp.TErrorEvent Enumeration 18886
ScktComp.TLookupState Enumeration 18887
ScktComp.TServerType Enumeration 18887
ScktComp.TSocketEvent Enumeration 18887

Types **18889**

ScktComp.TAsyncStyles Type 18889
ScktComp.TGetSocketEvent Type 18889
ScktComp.TGetThreadEvent Type 18890
ScktComp.TSocketErrorEvent Type 18890
ScktComp.TSocketErrorProc Type 18891
ScktComp.TSocketEventEvent Type 18891
ScktComp.TSocketNotifyEvent Type 18891
ScktComp.TThreadNotifyEvent Type 18892

Constants 18893

ScktComp.CM_DEFERFREE Constant 18893
ScktComp.CM_LOOKUPCOMPLETE Constant 18893
ScktComp.CM_SOCKETMESSAGE Constant 18893

SConnect Namespace 18895

Classes 18897

EInterpreterError Class 18897
ESocketConnectionError Class 18898
TCustomDataBlockInterpreter Class 18898
TCustomDataBlockInterpreter Members 18899
TCustomDataBlockInterpreter Methods 18899
TDataBlock Class 18903
TDataBlock Members 18903
TDataBlock.Create Constructor 18905
TDataBlock.Destroy Destructor 18905
TDataBlock Methods 18906
TDataBlock Properties 18910
TDataBlockInterpreter Class 18913
TDataBlockInterpreter Members 18913
TDataBlockInterpreter.Create Constructor 18914
TDataBlockInterpreter.Destroy Destructor 18914
TDataBlockInterpreter Methods 18915
TDataDispatch Class 18921
TDataDispatch Members 18921
TDataDispatch.Create Constructor 18922
TDataDispatch.Destroy Destructor 18922
TDataDispatch Methods 18923
TDataDispatch Properties 18924
TPacketInterceptFactory Class 18925

TPacketInterceptFactory Members 18925
TPacketInterceptFactory Methods 18927
TSocketConnection Class 18928
TSocketConnection Members 18929
TSocketConnection.Create Constructor 18933
TSocketConnection Methods 18933
TSocketConnection Properties 18935
TSocketTransport Class 18939
TSocketTransport Members 18940
TSocketTransport.Create Constructor 18941
TSocketTransport.Destroy Destructor 18941
TSocketTransport Methods 18942
TSocketTransport Properties 18944
TStreamedConnection Class 18945
TStreamedConnection Members 18946
TStreamedConnection.Create Constructor 18950
TStreamedConnection.Destroy Destructor 18951
TStreamedConnection Methods 18951
TStreamedConnection Properties 18958
TTransportThread Class 18961
TTransportThread Members 18962
TTransportThread.Create Constructor 18963
TTransportThread.Destroy Destructor 18963
TTransportThread Methods 18963
TTransportThread Properties 18967
TWebConnection Class 18968
TWebConnection Members 18969
TWebConnection.Create Constructor 18973
TWebConnection.Destroy Destructor 18974
TWebConnection Methods 18974
TWebConnection Properties 18978

Interfaces 18983

IDataBlock Interface 18983
IDataBlock Members 18984
IDataBlock Methods 18984
IDataBlock Properties 18989
IDataIntercept Interface 18991
IDataIntercept Members 18992
IDataIntercept Methods 18992

ISendDataBlock Interface 18993
ISendDataBlock Members 18994
ISendDataBlock Methods 18994
ITransport Interface 18995
ITransport Members 18995
ITransport Methods 18996
ITransport Properties 18998

Functions 18999
SConnect.GetPacketInterceptorList Function 18999
SConnect.LoadWinSock2 Function 18999

Structs, Records, Enums 19000
SConnect.TVarFlag Enumeration 19000

Types 19001
SConnect.PIntArray Type 19001
SConnect.PVariantArray Type 19001
SConnect.TIntArray Type 19002
SConnect.TVarFlags Type 19002
SConnect.TVariantArray Type 19002

Variables 19003
SConnect.WSACloseEvent Variable 19003
SConnect.WSACreateEvent Variable 19003
SConnect.WSAEventSelect Variable 19004
SConnect.WSAResetEvent Variable 19004

Constants 19005
SConnect.asCreateObject Constant 19005
SConnect.asError Constant 19005
SConnect.asFreeObject Constant 19006
SConnect.asGetAppServers Constant 19006
SConnect.asGetGUID Constant 19006
SConnect.asGetID Constant 19007
SConnect.asGetServers Constant 19007
SConnect.asInvoke Constant 19007
SConnect.asMask Constant 19008
SConnect.asSoapCommand Constant 19008
SConnect.CallSig Constant 19008
SConnect.ResultSig Constant 19009
SConnect.THREAD_EXCEPTION Constant 19009

SConnect.THREAD_RECEIVEDSTREAM Constant 19009
SConnect.THREAD_REPLACETRANSPORT Constant 19010
SConnect.THREAD_SENDNOTIFY Constant 19010
SConnect.THREAD_SENDSTREAM Constant 19010

SessColn Namespace 19011

Classes 19012
ESessionsException Class 19012
TAbstractSessionIDGen Class 19012
TAbstractSessionIDGen Members 19013
TAbstractSessionIDGen Methods 19013
TAbstractSessionStatusChange Class 19013
TAbstractSessionStatusChange Members 19014
TAbstractSessionStatusChange Methods 19014
TSessionIDGen Class 19015
TSessionIDGen Members 19015
TSessionIDGen.Create Constructor 19015
TSessionIDGen Methods 19016
TSessionItem Class 19016
TSessionItem Members 19016
TSessionItem.Create Constructor 19018
TSessionItem.Destroy Destructor 19018
TSessionItem Methods 19018
TSessionItem Properties 19020
TSessions Class 19022
TSessions Members 19022
TSessions.Create Constructor 19025
TSessions.Destroy Destructor 19025
TSessions Methods 19025
TSessions Properties 19033

Structs, Records, Enums 19035
SessColn.TSessionStatusFlag Enumeration 19035

Types 19036
SessColn.TSessionStatusFlags Type 19036

Variables 19037
SessColn.Sessions Variable 19037

SHDocVw Namespace 19039

Classes **19045**
CoCScriptErrorList Class 19045
CoCScriptErrorList Members 19045
CoCScriptErrorList Methods 19046
CoInternetExplorer Class 19046
CoInternetExplorer Members 19047
CoInternetExplorer Methods 19047
CoSearchAssistantOC Class 19047
CoSearchAssistantOC Members 19048
CoSearchAssistantOC Methods 19048
CoShellBrowserWindow Class 19049
CoShellBrowserWindow Members 19049
CoShellBrowserWindow Methods 19049
CoShellNameSpace Class 19050
CoShellNameSpace Members 19050
CoShellNameSpace Methods 19050
CoShellUIHelper Class 19051
CoShellUIHelper Members 19051
CoShellUIHelper Methods 19052
CoShellWindows Class 19052
CoShellWindows Members 19053
CoShellWindows Methods 19053
TInternetExplorer Class 19053
TInternetExplorer Members 19054
TInternetExplorer.Create Constructor 19059
TInternetExplorer.Destroy Destructor 19060
TInternetExplorer Methods 19061
TInternetExplorer Properties 19083
TInternetExplorer Events 19093
TShellNameSpace Class 19101
TShellNameSpace Members 19101
TShellNameSpace.Create Constructor 19105
TShellNameSpace.Destroy Destructor 19106
TShellNameSpace Methods 19107
TShellNameSpace Properties 19117
TShellNameSpace Events 19120

TShellUIHelper Class 19121
TShellUIHelper Members 19121
TShellUIHelper.Create Constructor 19125
TShellUIHelper.Destroy Destructor 19125
TShellUIHelper Methods 19126
TShellUIHelper Properties 19134
TShellWindows Class 19135
TShellWindows Members 19135
TShellWindows.Create Constructor 19139
TShellWindows.Destroy Destructor 19139
TShellWindows Methods 19140
TShellWindows Properties 19146
TShellWindows Events 19146
TWebBrowser Class 19147
TWebBrowser Members 19148
TWebBrowser Methods 19162
TWebBrowser Properties 19174
TWebBrowser Events 19199
TWebBrowser_V1 Class 19221
TWebBrowser_V1 Members 19221
TWebBrowser_V1 Methods 19235
TWebBrowser_V1 Properties 19240
TWebBrowser_V1 Events 19260

Interfaces 19276
_SearchAssistantEvents Interface 19276
_SearchAssistantEvents Members 19277
_SearchAssistantEvents Methods 19277
DShellNameSpaceEvents Interface 19278
DShellNameSpaceEvents Members 19278
DShellNameSpaceEvents Methods 19278
DShellWindowsEvents Interface 19279
DShellWindowsEvents Members 19280
DShellWindowsEvents Methods 19280
DWebBrowserEvents Interface 19281
DWebBrowserEvents Members 19281
DWebBrowserEvents Methods 19281
DWebBrowserEvents2 Interface 19286
DWebBrowserEvents2 Members 19286
DWebBrowserEvents2 Methods 19287

IScriptErrorList Interface 19295
IScriptErrorList Members 19295
IScriptErrorList Methods 19296
IScriptErrorListDisp Interface 19299
IScriptErrorListDisp Members 19299
IScriptErrorListDisp Methods 19300
ISearch Interface 19303
ISearch Members 19304
ISearch Methods 19304
ISearch Properties 19305
ISearchAssistantOC Interface 19306
ISearchAssistantOC Members 19306
ISearchAssistantOC Methods 19307
ISearchAssistantOC Properties 19314
ISearchAssistantOC2 Interface 19316
ISearchAssistantOC2 Members 19316
ISearchAssistantOC2 Methods 19317
ISearchAssistantOC2 Properties 19318
ISearchAssistantOC2Disp Interface 19318
ISearchAssistantOC2Disp Members 19319
ISearchAssistantOC2Disp Methods 19319
ISearchAssistantOC2Disp Properties 19324
ISearchAssistantOC3 Interface 19326
ISearchAssistantOC3 Members 19326
ISearchAssistantOC3 Methods 19327
ISearchAssistantOC3 Properties 19328
ISearchAssistantOC3Disp Interface 19329
ISearchAssistantOC3Disp Members 19329
ISearchAssistantOC3Disp Methods 19330
ISearchAssistantOC3Disp Properties 19334
ISearchAssistantOCDisp Interface 19337
ISearchAssistantOCDisp Members 19337
ISearchAssistantOCDisp Methods 19338
ISearchAssistantOCDisp Properties 19342
ISearchDisp Interface 19344
ISearchDisp Members 19345
ISearchDisp Properties 19345
ISearches Interface 19346
ISearches Members 19346

ISearches Methods 19347
ISearches Properties 19348
ISearchesDisp Interface 19348
ISearchesDisp Members 19349
ISearchesDisp Methods 19349
ISearchesDisp Properties 19350
IShellFavoritesNameSpace Interface 19350
IShellFavoritesNameSpace Members 19351
IShellFavoritesNameSpace Methods 19351
IShellFavoritesNameSpace Properties 19354
IShellFavoritesNameSpaceDisp Interface 19355
IShellFavoritesNameSpaceDisp Members 19355
IShellFavoritesNameSpaceDisp Methods 19356
IShellFavoritesNameSpaceDisp Properties 19359
IShellNameSpace Interface 19359
IShellNameSpace Members 19360
IShellNameSpace Methods 19361
IShellNameSpace Properties 19366
IShellNameSpaceDisp Interface 19368
IShellNameSpaceDisp Members 19369
IShellNameSpaceDisp Methods 19369
IShellNameSpaceDisp Properties 19373
IShellUIHelper Interface 19376
IShellUIHelper Members 19376
IShellUIHelper Methods 19377
IShellUIHelperDisp Interface 19380
IShellUIHelperDisp Members 19380
IShellUIHelperDisp Methods 19381
IShellWindows Interface 19384
IShellWindows Members 19384
IShellWindows Methods 19385
IShellWindows Properties 19388
IShellWindowsDisp Interface 19388
IShellWindowsDisp Members 19388
IShellWindowsDisp Methods 19389
IShellWindowsDisp Properties 19391
IWebBrowser Interface 19392
IWebBrowser Members 19392
IWebBrowser Methods 19393

IWebBrowser Properties 19399
IWebBrowser2 Interface 19403
IWebBrowser2 Members 19403
IWebBrowser2 Methods 19405
IWebBrowser2 Properties 19410
IWebBrowser2Disp Interface 19412
IWebBrowser2Disp Members 19412
IWebBrowser2Disp Methods 19413
IWebBrowser2Disp Properties 19417
IWebBrowserApp Interface 19425
IWebBrowserApp Members 19425
IWebBrowserApp Methods 19427
IWebBrowserApp Properties 19432
IWebBrowserAppDisp Interface 19434
IWebBrowserAppDisp Members 19435
IWebBrowserAppDisp Methods 19435
IWebBrowserAppDisp Properties 19438
IWebBrowserDisp Interface 19444
IWebBrowserDisp Members 19444
IWebBrowserDisp Methods 19445
IWebBrowserDisp Properties 19447

Types **19451**
SHDocVw.BrowserNavConstants Type 19453
SHDocVw.CommandStateChangeConstants Type 19453
SHDocVw.CScriptErrorList Type 19453
SHDocVw.InternetExplorer Type 19454
SHDocVw.OLECMDEXECOPT Type 19454
SHDocVw.OLECMDF Type 19455
SHDocVw.OLECMDID Type 19455
SHDocVw.POleVariant1 Type 19456
SHDocVw.RefreshConstants Type 19456
SHDocVw.SearchAssistantOC Type 19456
SHDocVw.SecureLockIconConstants Type 19457
SHDocVw.ShellBrowserWindow Type 19457
SHDocVw.ShellNameSpace Type 19457
SHDocVw.ShellUIHelper Type 19458
SHDocVw.ShellWindowFindWindowOptions Type 19458
SHDocVw.ShellWindows Type 19458
SHDocVw.ShellWindowTypeConstants Type 19459

SHDocVw.tagREADYSTATE Type 19459
SHDocVw.TInternetExplorerBeforeNavigate2 Type 19459
SHDocVw.TInternetExplorerClientToHostWindow Type 19460
SHDocVw.TInternetExplorerCommandStateChange Type 19460
SHDocVw.TInternetExplorerDocumentComplete Type 19460
SHDocVw.TInternetExplorerFileDownload Type 19461
SHDocVw.TInternetExplorerNavigateComplete2 Type 19461
SHDocVw.TInternetExplorerNavigateError Type 19461
SHDocVw.TInternetExplorerNewWindow2 Type 19462
SHDocVw.TInternetExplorerOnFullScreen Type 19462
SHDocVw.TInternetExplorerOnMenuBar Type 19462
SHDocVw.TInternetExplorerOnStatusBar Type 19463
SHDocVw.TInternetExplorerOnTheaterMode Type 19463
SHDocVw.TInternetExplorerOnToolBar Type 19463
SHDocVw.TInternetExplorerOnVisible Type 19464
SHDocVw.TInternetExplorerPrintTemplateInstantiation Type 19464
SHDocVw.TInternetExplorerPrintTemplateTeardown Type 19464
SHDocVw.TInternetExplorerPrivacyImpactedStateChange Type 19465
SHDocVw.TInternetExplorerProgressChange Type 19465
SHDocVw.TInternetExplorerPropertyChange Type 19465
SHDocVw.TInternetExplorerSetSecureLockIcon Type 19466
SHDocVw.TInternetExplorerStatusTextChange Type 19466
SHDocVw.TInternetExplorerTitleChange Type 19466
SHDocVw.TInternetExplorerUpdatePageStatus Type 19467
SHDocVw.TInternetExplorerWindowClosing Type 19467
SHDocVw.TInternetExplorerWindowSetHeight Type 19467
SHDocVw.TInternetExplorerWindowSetLeft Type 19468
SHDocVw.TInternetExplorerWindowSetResizable Type 19468
SHDocVw.TInternetExplorerWindowSetTop Type 19468
SHDocVw.TInternetExplorerWindowSetWidth Type 19469
SHDocVw.TShellNameSpaceFavoritesSelectionChange Type 19469
SHDocVw.TShellWindowsWindowRegistered Type 19470
SHDocVw.TShellWindowsWindowRevoked Type 19470
SHDocVw.TWebBrowser_V1BeforeNavigate Type 19470
SHDocVw.TWebBrowser_V1CommandStateChange Type 19471
SHDocVw.TWebBrowser_V1FrameBeforeNavigate Type 19471
SHDocVw.TWebBrowser_V1FrameNavigateComplete Type 19471
SHDocVw.TWebBrowser_V1FrameNewWindow Type 19472
SHDocVw.TWebBrowser_V1NavigateComplete Type 19472

SHDocVw.TWebBrowser_V1NewWindow Type 19472
SHDocVw.TWebBrowser_V1ProgressChange Type 19473
SHDocVw.TWebBrowser_V1PropertyChange Type 19473
SHDocVw.TWebBrowser_V1Quit Type 19474
SHDocVw.TWebBrowser_V1StatusTextChange Type 19474
SHDocVw.TWebBrowser_V1TitleChange Type 19474
SHDocVw.TWebBrowserBeforeNavigate2 Type 19475
SHDocVw.TWebBrowserClientToHostWindow Type 19475
SHDocVw.TWebBrowserCommandStateChange Type 19475
SHDocVw.TWebBrowserDocumentComplete Type 19476
SHDocVw.TWebBrowserFileDownload Type 19476
SHDocVw.TWebBrowserNavigateComplete2 Type 19476
SHDocVw.TWebBrowserNavigateError Type 19477
SHDocVw.TWebBrowserNewWindow2 Type 19477
SHDocVw.TWebBrowserOnFullScreen Type 19477
SHDocVw.TWebBrowserOnMenuBar Type 19478
SHDocVw.TWebBrowserOnStatusBar Type 19478
SHDocVw.TWebBrowserOnTheaterMode Type 19478
SHDocVw.TWebBrowserOnToolBar Type 19479
SHDocVw.TWebBrowserOnVisible Type 19479
SHDocVw.TWebBrowserPrintTemplateInstantiation Type 19479
SHDocVw.TWebBrowserPrintTemplateTeardown Type 19480
SHDocVw.TWebBrowserPrivacyImpactedStateChange Type 19480
SHDocVw.TWebBrowserProgressChange Type 19480
SHDocVw.TWebBrowserPropertyChange Type 19481
SHDocVw.TWebBrowserSetSecureLockIcon Type 19481
SHDocVw.TWebBrowserStatusTextChange Type 19481
SHDocVw.TWebBrowserTitleChange Type 19482
SHDocVw.TWebBrowserUpdatePageStatus Type 19482
SHDocVw.TWebBrowserWindowClosing Type 19482
SHDocVw.TWebBrowserWindowSetHeight Type 19483
SHDocVw.TWebBrowserWindowSetLeft Type 19483
SHDocVw.TWebBrowserWindowSetResizable Type 19483
SHDocVw.TWebBrowserWindowSetTop Type 19484
SHDocVw.TWebBrowserWindowSetWidth Type 19484
SHDocVw.WebBrowser Type 19484
SHDocVw.WebBrowser_V1 Type 19485

Constants **19486**

SHDocVw.CGID_IWebBrowser Constant 19488

SHDocVw.CLASS_CScriptErrorList Constant 19489
SHDocVw.CLASS_InternetExplorer Constant 19489
SHDocVw.CLASS_SearchAssistantOC Constant 19489
SHDocVw.CLASS_ShellBrowserWindow Constant 19490
SHDocVw.CLASS_ShellNameSpace Constant 19490
SHDocVw.CLASS_ShellUIHelper Constant 19490
SHDocVw.CLASS_ShellWindows Constant 19491
SHDocVw.CLASS_WebBrowser Constant 19491
SHDocVw.CLASS_WebBrowser_V1 Constant 19491
SHDocVw.CSC_NAVIGATEBACK Constant 19492
SHDocVw.CSC_NAVIGATEFORWARD Constant 19492
SHDocVw.CSC_UPDATECOMMANDS Constant 19492
SHDocVw.DIID__SearchAssistantEvents Constant 19493
SHDocVw.DIID_DShellNameSpaceEvents Constant 19493
SHDocVw.DIID_DShellWindowsEvents Constant 19493
SHDocVw.DIID_DWebBrowserEvents Constant 19494
SHDocVw.DIID_DWebBrowserEvents2 Constant 19494
SHDocVw.dtlOcxPage Constant 19494
SHDocVw.dtlServerPage Constant 19495
SHDocVw.HTMLID_FIND Constant 19495
SHDocVw.HTMLID_OPTIONS Constant 19495
SHDocVw.HTMLID_VIEWSOURCE Constant 19496
SHDocVw.IID_IScriptErrorList Constant 19496
SHDocVw.IID_ISearch Constant 19496
SHDocVw.IID_ISearchAssistantOC Constant 19497
SHDocVw.IID_ISearchAssistantOC2 Constant 19497
SHDocVw.IID_ISearchAssistantOC3 Constant 19497
SHDocVw.IID_ISearches Constant 19498
SHDocVw.IID_IShellFavoritesNameSpace Constant 19498
SHDocVw.IID_IShellNameSpace Constant 19498
SHDocVw.IID_IShellUIHelper Constant 19499
SHDocVw.IID_IShellWindows Constant 19499
SHDocVw.IID_IWebBrowser Constant 19499
SHDocVw.IID_IWebBrowser2 Constant 19500
SHDocVw.IID_IWebBrowserApp Constant 19500
SHDocVw.LIBID_SHDocVw Constant 19500
SHDocVw.navAllowAutosearch Constant 19501
SHDocVw.navBrowserBar Constant 19501
SHDocVw.navNoHistory Constant 19501

SHDocVw.navNoReadFromCache Constant 19502
SHDocVw.navNoWriteToCache Constant 19502
SHDocVw.navOpenInNewWindow Constant 19502
SHDocVw.OLECMDEXECOPT_DODEFAULT Constant 19503
SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER Constant 19503
SHDocVw.OLECMDEXECOPT_PROMPTUSER Constant 19503
SHDocVw.OLECMDEXECOPT_SHOWHELP Constant 19504
SHDocVw.OLECMDF_DEFHIDEONCTXTMENU Constant 19504
SHDocVw.OLECMDF_ENABLED Constant 19504
SHDocVw.OLECMDF_INVISIBLE Constant 19505
SHDocVw.OLECMDF_LATCHED Constant 19505
SHDocVw.OLECMDF_NINCHED Constant 19505
SHDocVw.OLECMDF_SUPPORTED Constant 19506
SHDocVw.OLECMDID_ALLOWUILESSSAVEAS Constant 19506
SHDocVw.OLECMDID_CLEARSELECTION Constant 19506
SHDocVw.OLECMDID_CLOSE Constant 19507
SHDocVw.OLECMDID_COPY Constant 19507
SHDocVw.OLECMDID_CUT Constant 19507
SHDocVw.OLECMDID_DELETE Constant 19508
SHDocVw.OLECMDID_DONTDOWNLOADCSS Constant 19508
SHDocVw.OLECMDID_ENABLE_INTERACTION Constant 19508
SHDocVw.OLECMDID_FIND Constant 19509
SHDocVw.OLECMDID_GETPRINTTEMPLATE Constant 19509
SHDocVw.OLECMDID_GETZOOMRANGE Constant 19509
SHDocVw.OLECMDID_HIDETOOLBARS Constant 19510
SHDocVw.OLECMDID_HTTPEQUIV Constant 19510
SHDocVw.OLECMDID_HTTPEQUIV_DONE Constant 19510
SHDocVw.OLECMDID_NEW Constant 19511
SHDocVw.OLECMDID_ONTOOLBARACTIVATED Constant 19511
SHDocVw.OLECMDID_ONUNLOAD Constant 19511
SHDocVw.OLECMDID_OPEN Constant 19512
SHDocVw.OLECMDID_PAGESETUP Constant 19512
SHDocVw.OLECMDID_PASTE Constant 19512
SHDocVw.OLECMDID_PASTESPECIAL Constant 19513
SHDocVw.OLECMDID_PREREFRESH Constant 19513
SHDocVw.OLECMDID_PRINT Constant 19513
SHDocVw.OLECMDID_PRINT2 Constant 19514
SHDocVw.OLECMDID_PRINTPREVIEW Constant 19514
SHDocVw.OLECMDID_PRINTPREVIEW2 Constant 19514

SHDocVw.OLECMDID_PROPERTIES Constant 19515
SHDocVw.OLECMDID_PROPERTYBAG2 Constant 19515
SHDocVw.OLECMDID_REDO Constant 19515
SHDocVw.OLECMDID_REFRESH Constant 19516
SHDocVw.OLECMDID_SAVE Constant 19516
SHDocVw.OLECMDID_SAVEAS Constant 19516
SHDocVw.OLECMDID_SAVECOPYAS Constant 19517
SHDocVw.OLECMDID_SELECTALL Constant 19517
SHDocVw.OLECMDID_SETDOWNLOADSTATE Constant 19517
SHDocVw.OLECMDID_SETPRINTTEMPLATE Constant 19518
SHDocVw.OLECMDID_SETPROGRESSMAX Constant 19518
SHDocVw.OLECMDID_SETPROGRESSPOS Constant 19518
SHDocVw.OLECMDID_SETPROGRESSTEXT Constant 19519
SHDocVw.OLECMDID_SETTITLE Constant 19519
SHDocVw.OLECMDID_SHOWFIND Constant 19519
SHDocVw.OLECMDID_SHOWMESSAGE Constant 19520
SHDocVw.OLECMDID_SHOWPAGESETUP Constant 19520
SHDocVw.OLECMDID_SHOWPRINT Constant 19520
SHDocVw.OLECMDID_SHOWSCRIPTERROR Constant 19521
SHDocVw.OLECMDID_SPELL Constant 19521
SHDocVw.OLECMDID_STOP Constant 19521
SHDocVw.OLECMDID_STOPDOWNLOAD Constant 19522
SHDocVw.OLECMDID_UNDO Constant 19522
SHDocVw.OLECMDID_UPDATECOMMANDS Constant 19522
SHDocVw.OLECMDID_UPDATEPAGESTATUS Constant 19523
SHDocVw.OLECMDID_ZOOM Constant 19523
SHDocVw.READYSTATE_COMPLETE Constant 19523
SHDocVw.READYSTATE_INTERACTIVE Constant 19524
SHDocVw.READYSTATE_LOADED Constant 19524
SHDocVw.READYSTATE_LOADING Constant 19524
SHDocVw.READYSTATE_UNINITIALIZED Constant 19525
SHDocVw.REFRESH_COMPLETELY Constant 19525
SHDocVw.REFRESH_CONTINUE Constant 19525
SHDocVw.REFRESH_IFEXPIRED Constant 19526
SHDocVw.REFRESH_NORMAL Constant 19526
SHDocVw.secureLockIconMixed Constant 19526
SHDocVw.secureLockIconSecure128Bit Constant 19527
SHDocVw.secureLockIconSecure40Bit Constant 19527
SHDocVw.secureLockIconSecure56Bit Constant 19527

SHDocVw.secureLockIconSecureFortezza Constant 19528
SHDocVw.secureLockIconSecureUnknownBits Constant 19528
SHDocVw.secureLockIconUnsecure Constant 19528
SHDocVw.SHDocVwMajorVersion Constant 19529
SHDocVw.SHDocVwMinorVersion Constant 19529
SHDocVw.SWC_3RDPARTY Constant 19529
SHDocVw.SWC_BROWSER Constant 19530
SHDocVw.SWC_CALLBACK Constant 19530
SHDocVw.SWC_EXPLORER Constant 19530
SHDocVw.SWFO_COOKIEPASSED Constant 19531
SHDocVw.SWFO_INCLUDEPENDING Constant 19531
SHDocVw.SWFO_NEEDDISPATCH Constant 19531

ShellAnimations Namespace 19533

Classes 19534
TShellResources Class 19534

SimpleDS Namespace 19537

Classes 19538
TInternalSQLDataSet Class 19538
TInternalSQLDataSet Members 19538
TInternalSQLDataSet Properties 19550
TSimpleDataSet Class 19555
TSimpleDataSet Members 19556
TSimpleDataSet.Create Constructor 19570
TSimpleDataSet.Destroy Destructor 19571
TSimpleDataSet Methods 19571
TSimpleDataSet Properties 19574
TSimpleDataSet Events 19597

SiteComp Namespace 19627

Classes 19631
TAbstractDesigntimeWarnings Class 19631
TAbstractDesigntimeWarnings Members 19631
TAbstractDesigntimeWarnings Methods 19632
TAbstractInetFileType Class 19633
TAbstractInetFileType Members 19633

TAbstractInetFileType Methods 19634
TAbstractInetFileType Properties 19635
TNotifyList Class 19636
TNotifyList Members 19636
TNotifyList.Create Constructor 19637
TNotifyList.Destroy Destructor 19637
TNotifyList Methods 19638
TNotifyList Properties 19640
TScriptComponent Class 19641
TScriptComponent Members 19641
TScriptComponent.Create Constructor 19642
TScriptComponent Methods 19642
TScriptComponent Properties 19643
TScriptObject Class 19643

Interfaces 19645
IActionImageProducer Interface 19646
IActionImageProducer Members 19647
IActionImageProducer Methods 19647
IAdapterAccess Interface 19649
IAdapterAccess Members 19649
IAdapterAccess Methods 19649
IAdapterActionAccess Interface 19650
IAdapterActionAccess Members 19651
IAdapterActionAccess Methods 19651
IAdapterActionAttributes Interface 19651
IAdapterActionAttributes Members 19652
IAdapterActionAttributes Methods 19652
IAdapterActionAttributes Properties 19652
IAdapterDispatcher Interface 19653
IAdapterDispatcher Members 19653
IAdapterDispatcher Methods 19654
IAdapterDispatchParams Interface 19655
IAdapterDispatchParams Members 19656
IAdapterDispatchParams Methods 19656
IAdapterDispatchParams Properties 19658
IAdapterError Interface 19660
IAdapterError Members 19660
IAdapterError Methods 19660
IAdapterError Properties 19661

IAdapterErrors Interface 19663
IAdapterErrors Members 19663
IAdapterErrors Methods 19663
IAdapterErrors Properties 19665
IAdapterFieldAccess Interface 19666
IAdapterFieldAccess Members 19667
IAdapterFieldAccess Methods 19667
IAdapterFieldAttributes Interface 19667
IAdapterFieldAttributes Members 19668
IAdapterFieldAttributes Methods 19668
IAdapterFieldAttributes Properties 19669
IAdapterHiddenField Interface 19669
IAdapterHiddenField Members 19670
IAdapterHiddenField Methods 19670
IAdapterHiddenField Properties 19671
IAdapterHiddenFields Interface 19671
IAdapterHiddenFields Members 19672
IAdapterHiddenFields Methods 19672
IAdapterHiddenFields Properties 19672
IAdapterItemRequestParams Interface 19673
IAdapterItemRequestParams Members 19674
IAdapterItemRequestParams Methods 19674
IAdapterItemRequestParams Properties 19676
IAdapterMode Interface 19677
IAdapterMode Members 19678
IAdapterMode Methods 19678
IAdapterMode Properties 19679
IAdapterRequestHandler Interface 19680
IAdapterRequestHandler Members 19680
IAdapterRequestHandler Methods 19680
IAsFormatted Interface 19681
IAsFormatted Members 19682
IAsFormatted Methods 19682
IDefaultPageFileName Interface 19682
IDefaultPageFileName Members 19683
IDefaultPageFileName Methods 19683
IFieldValuesAdapter Interface 19683
IFieldValuesAdapter Members 19684
IFieldValuesAdapter Methods 19684

IFindIncludeFile Interface 19684
IFindIncludeFile Members 19685
IFindIncludeFile Methods 19685
IGetAdapterActions Interface 19685
IGetAdapterActions Members 19686
IGetAdapterActions Methods 19686
IGetAdapterDisplayCharacteristics Interface 19686
IGetAdapterDisplayCharacteristics Members 19687
IGetAdapterDisplayCharacteristics Methods 19687
IGetAdapterErrors Interface 19687
IGetAdapterErrors Members 19688
IGetAdapterErrors Methods 19688
IGetAdapterFields Interface 19688
IGetAdapterFields Members 19689
IGetAdapterFields Methods 19689
IGetAdapterHiddenFields Interface 19689
IGetAdapterHiddenFields Members 19689
IGetAdapterHiddenFields Methods 19690
IGetAdapterImage Interface 19690
IGetAdapterImage Members 19691
IGetAdapterImage Methods 19691
IGetAdapterItemRequestParams Interface 19691
IGetAdapterItemRequestParams Members 19692
IGetAdapterItemRequestParams Methods 19692
IGetAdapterRequestDefaultResponse Interface 19692
IGetAdapterRequestDefaultResponse Members 19693
IGetAdapterRequestDefaultResponse Methods 19693
IGetAdapterRequestDefaultResponse Properties 19694
IGetAdapterValuesList Interface 19694
IGetAdapterValuesList Members 19695
IGetAdapterValuesList Methods 19695
IGetDefaultAction Interface 19695
IGetDefaultAction Members 19695
IGetDefaultAction Methods 19696
IGetDefaultAction Properties 19696
IGetDesigntimeWarnings Interface 19697
IGetDesigntimeWarnings Members 19697
IGetDesigntimeWarnings Methods 19697
IGetFieldValuesAdapter Interface 19698

IGetFieldValuesAdapter Members 19698
IGetFieldValuesAdapter Methods 19698
IGetHTMLStyle Interface 19699
IGetHTMLStyle Members 19699
IGetHTMLStyle Methods 19699
IGetProducerComponent Interface 19702
IGetProducerComponent Members 19702
IGetProducerComponent Methods 19702
IGetScriptObject Interface 19703
IGetScriptObject Members 19703
IGetScriptObject Methods 19703
IGetScriptObjectClassKey Interface 19704
IGetScriptObjectClassKey Members 19704
IGetScriptObjectClassKey Methods 19704
IGetWebAppComponents Interface 19705
IGetWebAppComponents Members 19705
IGetWebAppComponents Methods 19705
IGetWebAppComponents Properties 19707
IIDEFileManager Interface 19708
IIDEFileManager Members 19708
IIDEFileManager Methods 19708
IImageProducer Interface 19709
IImageProducer Members 19709
IImageProducer Methods 19710
IInternetEnvOptions Interface 19710
IInternetEnvOptions Members 19710
IInternetEnvOptions Methods 19711
IInternetEnvOptions Properties 19712
IInternetExecuteScript Interface 19713
IInternetExecuteScript Members 19713
IInternetExecuteScript Methods 19713
IInternetFileTypes Interface 19714
IInternetFileTypes Members 19714
IInternetFileTypes Methods 19715
IInternetScriptDebugger Interface 19715
IInternetScriptDebugger Members 19715
IInternetScriptDebugger Methods 19716
IIsAdapterActionList Interface 19716
IIsAdapterActionList Members 19717

IIsAdapterActionList Methods 19717
IIsAdapterActionList Properties 19717
IIterateIntfSupport Interface 19718
IIterateIntfSupport Members 19718
IIterateIntfSupport Methods 19718
IIterateObjectSupport Interface 19719
IIterateObjectSupport Members 19719
IIterateObjectSupport Methods 19719
IIteratorIndex Interface 19720
IIteratorIndex Members 19720
IIteratorIndex Methods 19720
IIteratorIndex Properties 19721
IIteratorSupport Interface 19722
IIteratorSupport Members 19723
IIteratorSupport Methods 19723
INotifyAdapterChange Interface 19724
INotifyAdapterChange Members 19724
INotifyAdapterChange Methods 19725
INotifyList Interface 19725
INotifyList Members 19726
INotifyList Methods 19726
IPageDispatcher Interface 19727
IPageDispatcher Members 19727
IPageDispatcher Methods 19728
IPageDispatcher Properties 19731
IPageResult Interface 19732
IPageResult Members 19732
IPageResult Methods 19732
ISessionsService Interface 19733
ISessionsService Members 19734
ISessionsService Methods 19734
ISessionsServiceConnectID Interface 19736
ISessionsServiceConnectID Members 19736
ISessionsServiceConnectID Methods 19736
ISessionsServicePersist Interface 19737
ISessionsServicePersist Members 19737
ISessionsServicePersist Methods 19737
ISessionsServiceStoreID Interface 19739
ISessionsServiceStoreID Members 19739

ISessionsServiceStoreID Methods 19739
IValuesListAdapter Interface 19740
IValuesListAdapter Members 19741
IValuesListAdapter Methods 19741
IWebActionsList Interface 19743
IWebActionsList Members 19744
IWebActionsList Methods 19744
IWebActionsList Properties 19745
IWebApplicationInfo Interface 19745
IWebApplicationInfo Members 19746
IWebApplicationInfo Methods 19746
IWebApplicationInfo Properties 19747
IWebContainerSubComponent Interface 19748
IWebDataFields Interface 19749
IWebDataFields Members 19749
IWebDataFields Methods 19749
IWebDataFields Properties 19750
IWebEnabled Interface 19751
IWebEnabled Members 19751
IWebEnabled Methods 19751
IWebFormComponentContainer Interface 19752
IWebFormComponentContainer Members 19752
IWebFormComponentContainer Methods 19752
IWebFormComponentContainer Properties 19753
IWebGetActionName Interface 19754
IWebGetActionName Members 19754
IWebGetActionName Methods 19754
IWebGetActionName Properties 19755
IWebGetDisplayLabel Interface 19755
IWebGetDisplayLabel Members 19756
IWebGetDisplayLabel Methods 19756
IWebGetDisplayWidth Interface 19756
IWebGetDisplayWidth Members 19756
IWebGetDisplayWidth Methods 19757
IWebGetFieldName Interface 19757
IWebGetFieldName Members 19757
IWebGetFieldName Methods 19758
IWebGetFieldName Properties 19758
IWebGetFieldValue Interface 19758

IWebGetFieldValue Members 19759
IWebGetFieldValue Methods 19759
IWebGetFieldValue Properties 19759
IWebGetFieldValues Interface 19760
IWebGetFieldValues Members 19760
IWebGetFieldValues Methods 19760
IWebGetMaxLength Interface 19763
IWebGetMaxLength Members 19763
IWebGetMaxLength Methods 19763
IWebGetSessionIDs Interface 19764
IWebGetSessionIDs Members 19764
IWebGetSessionIDs Methods 19764
IWebImageHREF Interface 19765
IWebImageHREF Members 19765
IWebImageHREF Methods 19765
IWebInputName Interface 19766
IWebInputName Members 19766
IWebInputName Methods 19767
IWebInputName Properties 19767
IWebIsDefaultAction Interface 19768
IWebIsDefaultAction Members 19768
IWebIsDefaultAction Methods 19768
IWebIsDefaultField Interface 19769
IWebIsDefaultField Members 19769
IWebIsDefaultField Methods 19769
IWebSession Interface 19770
IWebSession Members 19770
IWebSession Methods 19770
IWebSessionAttributes Interface 19771
IWebSessionAttributes Members 19771
IWebSessionAttributes Methods 19772
IWebSessionAttributes Properties 19774
IWebSessionID Interface 19776
IWebSessionID Members 19776
IWebSessionID Methods 19776
IWebSessionID Properties 19777
IWebSessionIDs Interface 19777
IWebSessionIDs Members 19778
IWebSessionIDs Methods 19778

IWebSessionIDs Properties 19779
IWebSessionValues Interface 19780
IWebSessionValues Members 19780
IWebSessionValues Methods 19780
IWebSessionValues Properties 19781
IWebSetActionName Interface 19782
IWebSetActionName Members 19782
IWebSetActionName Methods 19782
IWebSetFieldName Interface 19783
IWebSetFieldName Members 19783
IWebSetFieldName Methods 19783
IWebUserList Interface 19784
IWebUserList Members 19784
IWebUserList Methods 19785
IWebValueIsEqual Interface 19786
IWebValueIsEqual Members 19786
IWebValueIsEqual Methods 19786

Functions 19788
SiteComp.FullyQualifiedFieldName Function 19788
SiteComp.FullyQualifiedName Function 19788

Structs, Records, Enums 19789
SiteComp.TAdapterActionHTMLElementType Enumeration 19789
SiteComp.TAdapterDisplayCharacteristic Enumeration 19789
SiteComp.TAdapterDisplayHTMLElementType Enumeration 19790
SiteComp.TAdapterDisplayViewModeType Enumeration 19790
SiteComp.TAdapterInputHTMLElementType Enumeration 19791
SiteComp.TAdapterItemRequestParamOption Enumeration 19792
SiteComp.TDispatchPageFlag Enumeration 19792
SiteComp.TQualifyOption Enumeration 19792

Types 19794
SiteComp.TAdapterDisplayCharacteristics Type 19794
SiteComp.TAdapterItemRequestParamOptions Type 19794
SiteComp.TDispatchPageFlags Type 19795
SiteComp.TSessionID Type 19795

Variables 19796
SiteComp.ExecuteScript Variable 19796
SiteComp.GetHTMLSampleImage Variable 19796

SiteComp.IDEFileManager Variable 19797
SiteComp.InetFileTypes Variable 19797
SiteComp.InternetEnvOptions Variable 19797
SiteComp.ScriptDebugger Variable 19798

Constants 19799

SiteComp.AdapterActionHTMLElementTypeNames Constant 19799
SiteComp.AdapterDisplayHTMLElementTypeNames Constant 19799
SiteComp.AdapterDisplayViewModeTypeNames Constant 19800
SiteComp.AdapterInputHTMLElementTypeNames Constant 19800

SiteProd Namespace 19801

Classes 19802

TAdapterPageProducer Class 19802
TAdapterPageProducer Members 19803
TAdapterPageProducer Properties 19808
TAdapterPageProducer Events 19812
TBaseAdapterPageProducer Class 19813
TBaseAdapterPageProducer Members 19814
TBaseAdapterPageProducer.Create Constructor 19818
TBaseAdapterPageProducer.Destroy Destructor 19818
TBaseAdapterPageProducer Methods 19819
TBaseAdapterPageProducer Properties 19830
TBaseAdapterPageProducer Events 19831
TCustomAdapterPageProducer Class 19832
TCustomAdapterPageProducer Members 19833
TCustomAdapterPageProducer.Create Constructor 19838
TCustomAdapterPageProducer.Destroy Destructor 19839
TCustomAdapterPageProducer Methods 19839
TCustomAdapterPageProducer Properties 19842
TSitePageElements Class 19844
TSitePageElements Members 19844
TSitePageElements Fields 19845

SOAPAttachIntf Namespace 19847

Classes 19848

TSoapDataList Class 19848
TSoapDataList Members 19848

TSoapDataList Methods 19850

Interfaces **19853**

IMimeAttachmentHandler Interface 19853

IMimeAttachmentHandler Members 19853

IMimeAttachmentHandler Methods 19854

IMimeAttachmentHandler Properties 19859

IMimeAttachmentHandler Events 19860

Types **19862**

SOAPAttachIntf.TOnGetAttachmentEvent Type 19862

SOAPAttachIntf.TOnSendAttachmentEvent Type 19862

SOAPConn Namespace 19863

Classes **19864**

TSoapConnection Class 19864

TSoapConnection Members 19865

TSoapConnection.Create Constructor 19869

TSoapConnection.Destroy Destructor 19869

TSoapConnection Methods 19870

TSoapConnection Properties 19874

TSoapConnection Events 19879

SOAPDm Namespace 19883

Classes **19884**

TSoapDataModule Class 19884

TSoapDataModule Members 19885

TSoapDataModule.Create Constructor 19888

TSoapDataModule.Destroy Destructor 19889

TSoapDataModule Methods 19889

TSoapDataModule Properties 19908

SOAPDomConv Namespace 19911

Classes **19912**

TSOAPDOMProcessor Class 19912

TSOAPDOMProcessor Members 19912

TSOAPDOMProcessor.Destroy Destructor 19915

TSOAPDOMProcessor Fields 19915

TSOAPDOMProcessor Methods 19916

Interfaces 19921

IDOMHeaderProcessor Interface 19921

IDOMHeaderProcessor Members 19922

IDOMHeaderProcessor Methods 19922

Structs, Records, Enums 19924

SOAPDomConv.TDOMHeaderProcessorEntry Record 19924

Types 19925

SOAPDomConv.TDOMHeaderProcessorArray Type 19925

SOAPHTTPClient Namespace 19927

Classes 19928

THTTPRIO Class 19928

THTTPRIO Members 19929

THTTPRIO.Create Constructor 19932

THTTPRIO.Destroy Destructor 19932

THTTPRIO Methods 19933

THTTPRIO Properties 19934

SOAPHTTPDisp Namespace 19939

Classes 19940

THTTPSoapDispatchNode Class 19940

THTTPSoapDispatchNode Members 19940

THTTPSoapDispatchNode Fields 19943

THTTPSoapDispatchNode Methods 19943

THTTPSoapDispatchNode Properties 19945

Interfaces 19946

IHTTPSoapDispatch Interface 19946

IHTTPSoapDispatch Members 19946

IHTTPSoapDispatch Methods 19947

SOAPHTTPPasInv Namespace 19949

Classes 19950

THTTPSoapCppInvoker Class 19950

THTTPSoapPascalInvoker Class 19951

THTTPSoapPascalInvoker Members 19952

THTTPSoapPascalInvoker Methods 19955

SOAPHTTPTrans Namespace 19957

Classes **19958**

ESOAPHTTPException Class 19958

ESOAPHTTPException Members 19958

ESOAPHTTPException.Create Constructor 19959

ESOAPHTTPException.CreateFmt Constructor 19960

ESOAPHTTPException Properties 19960

THTTPReqResp Class 19961

THTTPReqResp Members 19962

THTTPReqResp.Create Constructor 19966

THTTPReqResp.Destroy Destructor 19966

THTTPReqResp Methods 19966

THTTPReqResp Properties 19975

THTTPReqResp Events 19981

Interfaces **19983**

IHTTPReqResp Interface 19983

IHTTPReqResp Members 19983

IHTTPReqResp Methods 19983

Structs, Records, Enums **19985**

SOAPHTTPTrans.SOAPInvokeOptions Enumeration 19985

Types **19986**

SOAPHTTPTrans.TBeforePostEvent Type 19986

SOAPHTTPTrans.TPostingDataEvent Type 19986

SOAPHTTPTrans.TReceivingDataEvent Type 19987

SOAPHTTPTrans.TSOAPInvokeOptions Type 19987

SOAPLinked Namespace 19989

Classes **19990**

TLinkedRIO Class 19990

TLinkedRIO Members 19991

TLinkedRIO.Create Constructor 19994

TLinkedRIO.CreateFile Constructor 19994

TLinkedRIO.Destroy Destructor 19995

TLinkedRIO Methods 19995

TLinkedRIO Properties 19997

TLinkedWebNode Class 19997
TLinkedWebNode Members 19998
TLinkedWebNode.Create Constructor 20001
TLinkedWebNode.Destroy Destructor 20001
TLinkedWebNode Methods 20001
TLinkedWebNode Properties 20004
TLogLinkedWebNode Class 20005
TLogLinkedWebNode Members 20006
TLogLinkedWebNode.Create Constructor 20008
TLogLinkedWebNode.Destroy Destructor 20009
TLogLinkedWebNode Methods 20009
TLogLinkedWebNode Properties 20010

SOAPMidas Namespace 20011

Interfaces 20012
IAppServerSOAP Interface 20012
IAppServerSOAP Members 20012
IAppServerSOAP Methods 20013

Functions 20017
SOAPMidas.RegDefIAppServerInvClass Function 20017

SOAPPasInv Namespace 20019

Classes 20020
TSoapPascalInvoker Class 20020
TSoapPascalInvoker Members 20021
TSoapPascalInvoker.Create Constructor 20024
TSoapPascalInvoker.Destroy Destructor 20024
TSoapPascalInvoker Fields 20024
TSoapPascalInvoker Methods 20026
TSoapPascalInvoker Properties 20028
TSoapPascalInvoker Events 20028

Interfaces 20032
IAdapterConverter Interface 20032
IAdapterConverter Members 20032
IAdapterConverter Methods 20032
IInvokerConverter Interface 20033
IInvokerConverter Members 20033

IInvokerConverter Methods 20033

Functions **20035**

SOAPPasInv.GetSOAPHeaders Function 20035

Types **20036**

SOAPPasInv.TAfterDispatchEvent Type 20036

SOAPPasInv.TBeforeDispatchEvent Type 20036

SOAPPasInv.TBeforeDispatchEvent2 Type 20037

SOAPPasInv.TOnExceptionEvent Type 20037

Sockets Namespace 20039

Classes **20041**

ESocketError Class 20041

TBaseSocket Class 20042

TBaseSocket Members 20042

TBaseSocket.Create Constructor 20046

TBaseSocket.Destroy Destructor 20046

TBaseSocket Methods 20046

TBaseSocket Properties 20054

TBaseSocket Events 20057

TClientSocketThread Class 20059

TClientSocketThread Members 20059

TClientSocketThread.Create Constructor 20060

TClientSocketThread.Destroy Destructor 20061

TClientSocketThread Methods 20061

TClientSocketThread Properties 20063

TCustomIpClient Class 20063

TCustomIpClient Members 20064

TCustomIpClient.Create Constructor 20068

TCustomIpClient Methods 20068

TCustomIpClient Properties 20071

TCustomIpClient Events 20071

TCustomTcpServer Class 20072

TCustomTcpServer Members 20073

TCustomTcpServer.Create Constructor 20076

TCustomTcpServer.Destroy Destructor 20077

TCustomTcpServer Methods 20077

TCustomTcpServer Properties 20080

TCustomTcpServer Events 20081

TIpSocket Class 20082
TIpSocket Members 20083
TIpSocket.Create Constructor 20086
TIpSocket Methods 20087
TIpSocket Properties 20091
TRawSocket Class 20094
TRawSocket Members 20094
TRawSocket.Create Constructor 20098
TRawSocket Properties 20098
TServerSocketThread Class 20099
TServerSocketThread Members 20099
TServerSocketThread.Create Constructor 20100
TServerSocketThread.Destroy Destructor 20101
TServerSocketThread Methods 20101
TServerSocketThread Properties 20103
TServerSocketThread Events 20104
TTcpClient Class 20105
TTcpClient Members 20106
TTcpClient Properties 20109
TTcpClient Events 20112
TTcpServer Class 20115
TTcpServer Members 20115
TTcpServer Properties 20119
TTcpServer Events 20121
TUdpSocket Class 20123
TUdpSocket Members 20124
TUdpSocket.Create Constructor 20128
TUdpSocket Properties 20128
TUdpSocket Events 20131

Structs, Records, Enums 20135
Sockets.TIPHeader Record 20135
Sockets.TServerSocketBlockMode Enumeration 20136
Sockets.TSocketDomain Enumeration 20136
Sockets.TSocketType Enumeration 20138

Types 20140
Sockets.TGetThreadEvent Type 20140
Sockets.TSocketAcceptEvent Type 20140
Sockets.TSocketBlockMode Type 20141

Sockets.TSocketDataEvent Type 20141
Sockets.TSocketErrorEvent Type 20141
Sockets.TSocketHost Type 20142
Sockets.TSocketNotifyEvent Type 20142
Sockets.TSocketPort Type 20142
Sockets.TSocketProtocol Type 20143

Constants 20144
Sockets.CRLF Constant 20144

SqlExpr Namespace 20145

Classes 20147
SPParamDesc Class 20147
SPParamDesc Members 20148
SPParamDesc Fields 20148
TCustomSQLDataSet Class 20150
TCustomSQLDataSet Members 20151
TCustomSQLDataSet.Create Constructor 20160
TCustomSQLDataSet.Destroy Destructor 20161
TCustomSQLDataSet Methods 20161
TCustomSQLDataSet Properties 20186
TCustomSQLDataSet Events 20203
TFLDDesc Class 20205
TFLDDesc Members 20206
TFLDDesc Fields 20206
TFLDDesc Properties 20209
TSQLBlobStream Class 20213
TSQLBlobStream Members 20213
TSQLBlobStream.Create Constructor 20215
TSQLBlobStream.Destroy Destructor 20216
TSQLBlobStream Methods 20216
TSQLConnection Class 20217
TSQLConnection Members 20218
TSQLConnection.Create Constructor 20222
TSQLConnection.Destroy Destructor 20222
TSQLConnection Methods 20223
TSQLConnection Properties 20241
TSQLConnection Events 20257
TSQLDataLink Class 20260

TSQLDataLink Members 20261
TSQLDataLink.Create Constructor 20263
TSQLDataLink Methods 20263
TSQLDataSet Class 20265
TSQLDataSet Members 20266
TSQLDataSet.Create Constructor 20278
TSQLDataSet Methods 20278
TSQLDataSet Properties 20279
TSQLMonitor Class 20285
TSQLMonitor Members 20285
TSQLMonitor.Create Constructor 20288
TSQLMonitor.Destroy Destructor 20289
TSQLMonitor Methods 20289
TSQLMonitor Properties 20292
TSQLMonitor Events 20296
TSQLQuery Class 20298
TSQLQuery Members 20299
TSQLQuery.Create Constructor 20310
TSQLQuery.Destroy Destructor 20311
TSQLQuery Methods 20311
TSQLQuery Properties 20312
TSQLStoredProc Class 20318
TSQLStoredProc Members 20319
TSQLStoredProc.Create Constructor 20331
TSQLStoredProc Methods 20331
TSQLStoredProc Properties 20333
TSQLTable Class 20337
TSQLTable Members 20338
TSQLTable.Create Constructor 20349
TSQLTable.Destroy Destructor 20350
TSQLTable Methods 20350
TSQLTable Properties 20352
TTransactionItem Class 20359
TTransactionItem Members 20359
TTransactionItem Fields 20360

Functions 20361

SqlExpr.FreeProcParams Function 20361
SqlExpr.GetConnectionNames Function 20361
SqlExpr.GetConnectionRegistryFile Function 20362

SqlExpr.GetDriverNames Function 20362
SqlExpr.GetDriverRegistryFile Function 20363
SqlExpr.LoadParamListItems Function 20363

Structs, Records, Enums 20364
SqlExpr.EConnectFlag Enumeration 20364
SqlExpr.TConnectionState Enumeration 20365
SqlExpr.TConnectionUserType Enumeration 20365
SqlExpr.TSchemaType Enumeration 20366
SqlExpr.TSQLExceptionType Enumeration 20367
SqlExpr.TSQLSchemaInfo Record 20368
SqlExpr.TTableScope Enumeration 20368
SqlExpr.TTransactionDesc Record 20369
SqlExpr.TTransIsolationLevel Enumeration 20369

Types 20371
SqlExpr.TFieldDescList Type 20371
SqlExpr.TFieldList Type 20371
SqlExpr.TGetDriverFunc Type 20372
SqlExpr.TLocale Type 20372
SqlExpr.TLocaleCode Type 20372
SqlExpr.TParseInsertSqlEvent Type 20373
SqlExpr.TParseSqlEvent Type 20373
SqlExpr.TSQLConnectionLoginEvent Type 20373
SqlExpr.TTableScopes Type 20374
SqlExpr.TTraceEvent Type 20374
SqlExpr.TTraceLogEvent Type 20374

Constants 20375
SqlExpr.BlobTypeMap Constant 20375
SqlExpr.DataTypeMap Constant 20376
SqlExpr.DefaultCursor Constant 20376
SqlExpr.DefaultMaxBlobSize Constant 20376
SqlExpr.DefaultRowsetSize Constant 20377
SqlExpr.FldSubTypeMap Constant 20377
SqlExpr.FldTypeMap Constant 20377
SqlExpr.HourGlassCursor Constant 20378
SqlExpr.SAnd Constant 20378
SqlExpr.SOrderBy Constant 20379
SqlExpr.SParam Constant 20379
SqlExpr.SSelect Constant 20379

SqlExpr.SSelectSpaces Constant 20380
SqlExpr.SSelectStar Constant 20380
SqlExpr.SSelectStarFrom Constant 20380
SqlExpr.SUB_TYPE_MEMO Constant 20381
SqlExpr.SWhere Constant 20381
SqlExpr.TErrorMessageSize Constant 20381

SqlTimSt Namespace 20383

Functions 20385
SqlTimSt.CheckSqlTimeStamp Function 20385
SqlTimSt.DateTimeToSQLTimeStamp Function 20386
SqlTimSt.SQLDayOfWeek Function 20386
SqlTimSt.SQLTimeStampToDateTime Function 20387
SqlTimSt.SQLTimeStampToStr Function 20387
SqlTimSt.StrToSQLTimeStamp Function 20388
SqlTimSt.TryStrToSQLTimeStamp Function 20388
SqlTimSt.VarIsSQLTimeStamp Function 20389
SqlTimSt.VarSQLTimeStamp Function 20389
SqlTimSt.VarSQLTimeStampCreate Function 20390
SqlTimSt.VarToSQLTimeStamp Function 20391

Structs, Records, Enums 20392
SqlTimSt.TSQLTimeStamp Record 20392

Types 20394
SqlTimSt.PSQLTimeStamp Type 20394

Constants 20395
SqlTimSt.NullSQLTimeStamp Constant 20395

StdActnMenus Namespace 20397

Classes 20398
TStandardAddRemoveItem Class 20398
TStandardAddRemoveItem Members 20399
TStandardAddRemoveItem Methods 20407
TStandardButtonControl Class 20408
TStandardButtonControl Members 20409
TStandardButtonControl Methods 20417
TStandardCustomizePopup Class 20418
TStandardCustomizePopup Members 20419

TStandardCustomizePopup Methods 20433
TStandardDropDownButton Class 20434
TStandardDropDownButton Members 20435
TStandardDropDownButton Methods 20443
TStandardMenuButton Class 20444
TStandardMenuButton Members 20445
TStandardMenuButton Methods 20454
TStandardMenuExpandBtn Class 20455
TStandardMenuExpandBtn Members 20455
TStandardMenuExpandBtn Methods 20464
TStandardMenuItem Class 20465
TStandardMenuItem Members 20466
TStandardMenuItem Methods 20474
TStandardMenuPopup Class 20478
TStandardMenuPopup Members 20478
TStandardMenuPopup Methods 20492
TStandardToolScrollBtn Class 20494
TStandardToolScrollBtn Members 20494
TStandardToolScrollBtn Methods 20503

Functions **20504**
StdActnMenus.RegisterStandardMenus Function 20504

StdActns Namespace 20505

Classes **20507**
TBrowseForFolder Class 20508
TBrowseForFolder Members 20508
TBrowseForFolder Methods 20512
TBrowseForFolder Properties 20514
TBrowseForFolder Events 20519
TColorSelect Class 20521
TColorSelect Members 20521
TColorSelect Methods 20526
TColorSelect Properties 20526
TColorSelect Events 20531
TCommonDialogAction Class 20533
TCommonDialogAction Members 20534
TCommonDialogAction.Create Constructor 20538
TCommonDialogAction.Destroy Destructor 20538

TCommonDialogAction Fields 20539
TCommonDialogAction Methods 20539
TCommonDialogAction Properties 20542
TCommonDialogAction Events 20542
TEditAction Class 20544
TEditAction Members 20545
TEditAction.Destroy Destructor 20549
TEditAction Methods 20549
TEditAction Properties 20551
TEditCopy Class 20552
TEditCopy Members 20552
TEditCopy Methods 20557
TEditCut Class 20557
TEditCut Members 20558
TEditCut Methods 20563
TEditDelete Class 20563
TEditDelete Members 20564
TEditDelete Methods 20568
TEditPaste Class 20570
TEditPaste Members 20570
TEditPaste Methods 20575
TEditSelectAll Class 20576
TEditSelectAll Members 20577
TEditSelectAll Methods 20581
TEditUndo Class 20582
TEditUndo Members 20583
TEditUndo Methods 20587
TFileAction Class 20588
TFileAction Members 20589
TFileAction Methods 20593
TFileAction Properties 20593
TFileExit Class 20594
TFileExit Members 20594
TFileExit Methods 20598
TFileExit Properties 20599
TFileExit Events 20603
TFileOpen Class 20605
TFileOpen Members 20606
TFileOpen.Create Constructor 20610

TFileOpen Methods 20610
TFileOpen Properties 20611
TFileOpen Events 20616
TFileOpenWith Class 20619
TFileOpenWith Members 20620
TFileOpenWith Methods 20624
TFileOpenWith Properties 20625
TFileOpenWith Events 20627
TFilePageSetup Class 20629
TFilePageSetup Members 20630
TFilePageSetup Methods 20634
TFilePageSetup Properties 20634
TFilePageSetup Events 20639
TFilePrintSetup Class 20642
TFilePrintSetup Members 20642
TFilePrintSetup Methods 20646
TFilePrintSetup Properties 20647
TFilePrintSetup Events 20651
TFileSaveAs Class 20654
TFileSaveAs Members 20655
TFileSaveAs.Create Constructor 20659
TFileSaveAs Methods 20659
TFileSaveAs Properties 20660
TFileSaveAs Events 20663
TFontEdit Class 20666
TFontEdit Members 20666
TFontEdit Methods 20670
TFontEdit Properties 20671
TFontEdit Events 20676
THelpAction Class 20678
THelpAction Members 20679
THelpAction.Create Constructor 20683
THelpAction Methods 20683
THelpContents Class 20684
THelpContents Members 20685
THelpContents Methods 20689
THelpContextAction Class 20690
THelpContextAction Members 20691
THelpContextAction Methods 20695

THelpOnHelp Class 20696
THelpOnHelp Members 20697
THelpOnHelp Methods 20701
THelpTopicSearch Class 20701
THelpTopicSearch Members 20702
THelpTopicSearch Methods 20706
THintAction Class 20707
THintAction Members 20708
THintAction.Create Constructor 20712
THintAction Properties 20712
TPrintDlg Class 20713
TPrintDlg Members 20714
TPrintDlg Methods 20718
TPrintDlg Properties 20718
TPrintDlg Events 20723
TSearchAction Class 20725
TSearchAction Members 20726
TSearchAction.Create Constructor 20730
TSearchAction.Destroy Destructor 20731
TSearchAction Fields 20731
TSearchAction Methods 20732
TSearchFind Class 20734
TSearchFind Members 20735
TSearchFind Methods 20739
TSearchFind Properties 20740
TSearchFind Events 20744
TSearchFindFirst Class 20747
TSearchFindFirst Members 20748
TSearchFindFirst.Create Constructor 20752
TSearchFindNext Class 20753
TSearchFindNext Members 20753
TSearchFindNext.Create Constructor 20757
TSearchFindNext Methods 20758
TSearchFindNext Properties 20759
TSearchFindNext Events 20764
TSearchReplace Class 20765
TSearchReplace Members 20766
TSearchReplace Methods 20771
TSearchReplace Properties 20772

TSearchReplace Events 20776
TWindowAction Class 20779
TWindowAction Members 20780
TWindowAction.Create Constructor 20784
TWindowAction Methods 20784
TWindowAction Properties 20786
TWindowArrange Class 20786
TWindowArrange Members 20787
TWindowArrange Methods 20791
TWindowCascade Class 20792
TWindowCascade Members 20793
TWindowCascade Methods 20797
TWindowClose Class 20798
TWindowClose Members 20798
TWindowClose Methods 20803
TWindowMinimizeAll Class 20804
TWindowMinimizeAll Members 20804
TWindowMinimizeAll Methods 20809
TWindowTileHorizontal Class 20809
TWindowTileHorizontal Members 20810
TWindowTileHorizontal Methods 20815
TWindowTileVertical Class 20815
TWindowTileVertical Members 20816
TWindowTileVertical Methods 20820

Structs, Records, Enums 20822
StdActns.TBrowseForFolderOption Enumeration 20822

Types 20824
StdActns.TBrowseForFolderCallbackEvent Type 20824
StdActns.TBrowseOptions Type 20824
StdActns.TCommonDialogClass Type 20825

StdConvs Namespace 20827

Functions 20832
StdConvs.CelsiusToFahrenheit Function 20832
StdConvs.FahrenheitToCelsius Function 20832
StdConvs.InitStdConvs Function 20833

Variables 20834

StdConvs.auAcres Variable 20836
StdConvs.auAres Variable 20837
StdConvs.auCentares Variable 20837
StdConvs.auHectares Variable 20837
StdConvs.auSquareCentimeters Variable 20838
StdConvs.auSquareDecameters Variable 20838
StdConvs.auSquareDecimeters Variable 20838
StdConvs.auSquareFeet Variable 20839
StdConvs.auSquareHectometers Variable 20839
StdConvs.auSquareInches Variable 20839
StdConvs.auSquareKilometers Variable 20840
StdConvs.auSquareMeters Variable 20840
StdConvs.auSquareMiles Variable 20840
StdConvs.auSquareMillimeters Variable 20841
StdConvs.auSquareRods Variable 20841
StdConvs.auSquareYards Variable 20841
StdConvs.cbArea Variable 20842
StdConvs.cbDistance Variable 20842
StdConvs.cbMass Variable 20842
StdConvs.cbTemperature Variable 20843
StdConvs.cbTime Variable 20843
StdConvs.cbVolume Variable 20843
StdConvs.duAngstroms Variable 20844
StdConvs.duAstronomicalUnits Variable 20844
StdConvs.duCentimeters Variable 20844
StdConvs.duChains Variable 20845
StdConvs.duCubits Variable 20845
StdConvs.duDecameters Variable 20845
StdConvs.duDecimeters Variable 20846
StdConvs.duFathoms Variable 20846
StdConvs.duFeet Variable 20846
StdConvs.duFurlongs Variable 20847
StdConvs.duGigameters Variable 20847
StdConvs.duHands Variable 20847
StdConvs.duHectometers Variable 20848
StdConvs.duInches Variable 20848
StdConvs.duKilometers Variable 20848
StdConvs.duLightYears Variable 20849
StdConvs.duLinks Variable 20849

StdConvs.duMegameters Variable 20849
StdConvs.duMeters Variable 20850
StdConvs.duMicromicrons Variable 20850
StdConvs.duMicrons Variable 20850
StdConvs.duMiles Variable 20851
StdConvs.duMillimeters Variable 20851
StdConvs.duMillimicrons Variable 20851
StdConvs.duNauticalMiles Variable 20852
StdConvs.duPaces Variable 20852
StdConvs.duParsecs Variable 20852
StdConvs.duPicas Variable 20853
StdConvs.duPoints Variable 20853
StdConvs.duRods Variable 20853
StdConvs.duYards Variable 20854
StdConvs.muCentigrams Variable 20854
StdConvs.muDecagrams Variable 20854
StdConvs.muDecigrams Variable 20855
StdConvs.muDrams Variable 20855
StdConvs.muGrains Variable 20855
StdConvs.muGrams Variable 20856
StdConvs.muHectograms Variable 20856
StdConvs.muKilograms Variable 20856
StdConvs.muLongTons Variable 20857
StdConvs.muMetricTons Variable 20857
StdConvs.muMicrograms Variable 20857
StdConvs.muMilligrams Variable 20858
StdConvs.muNanograms Variable 20858
StdConvs.muOunces Variable 20858
StdConvs.muPounds Variable 20859
StdConvs.muStones Variable 20859
StdConvs.muTons Variable 20859
StdConvs.tuCelsius Variable 20860
StdConvs.tuCenturies Variable 20860
StdConvs.tuDateTime Variable 20860
StdConvs.tuDays Variable 20861
StdConvs.tuDecades Variable 20861
StdConvs.tuFahrenheit Variable 20861
StdConvs.tuFortnights Variable 20862
StdConvs.tuHours Variable 20862

StdConvs.tuJulianDate Variable 20862
StdConvs.tuKelvin Variable 20863
StdConvs.tuMillennia Variable 20863
StdConvs.tuMilliSeconds Variable 20863
StdConvs.tuMinutes Variable 20864
StdConvs.tuModifiedJulianDate Variable 20864
StdConvs.tuMonths Variable 20864
StdConvs.tuRankine Variable 20865
StdConvs.tuReaumur Variable 20865
StdConvs.tuSeconds Variable 20865
StdConvs.tuWeeks Variable 20866
StdConvs.tuYears Variable 20866
StdConvs.vuAcreFeet Variable 20866
StdConvs.vuAcreInches Variable 20867
StdConvs.vuCentiLiters Variable 20867
StdConvs.vuCordFeet Variable 20867
StdConvs.vuCords Variable 20868
StdConvs.vuCubicCentimeters Variable 20868
StdConvs.vuCubicDecameters Variable 20868
StdConvs.vuCubicDecimeters Variable 20869
StdConvs.vuCubicFeet Variable 20869
StdConvs.vuCubicHectometers Variable 20869
StdConvs.vuCubicInches Variable 20870
StdConvs.vuCubicKilometers Variable 20870
StdConvs.vuCubicMeters Variable 20870
StdConvs.vuCubicMiles Variable 20871
StdConvs.vuCubicMillimeters Variable 20871
StdConvs.vuCubicYards Variable 20871
StdConvs.vuDecaLiters Variable 20872
StdConvs.vuDecasteres Variable 20872
StdConvs.vuDeciLiters Variable 20872
StdConvs.vuDecisteres Variable 20873
StdConvs.vuDryBuckets Variable 20873
StdConvs.vuDryBushels Variable 20873
StdConvs.vuDryGallons Variable 20874
StdConvs.vuDryPecks Variable 20874
StdConvs.vuDryPints Variable 20874
StdConvs.vuDryQuarts Variable 20875
StdConvs.vuFluidCups Variable 20875

StdConvs.vuFluidGallons Variable 20875
StdConvs.vuFluidGills Variable 20876
StdConvs.vuFluidOunces Variable 20876
StdConvs.vuFluidPints Variable 20876
StdConvs.vuFluidQuarts Variable 20877
StdConvs.vuFluidTablespoons Variable 20877
StdConvs.vuFluidTeaspoons Variable 20877
StdConvs.vuHectoLiters Variable 20878
StdConvs.vuKiloLiters Variable 20878
StdConvs.vuLiters Variable 20878
StdConvs.vuMilliLiters Variable 20879
StdConvs.vuSteres Variable 20879
StdConvs.vuUKBuckets Variable 20879
StdConvs.vuUKBushels Variable 20880
StdConvs.vuUKGallons Variable 20880
StdConvs.vuUKGills Variable 20880
StdConvs.vuUKOunces Variable 20881
StdConvs.vuUKPecks Variable 20881
StdConvs.vuUKPints Variable 20881
StdConvs.vuUKPottles Variable 20882
StdConvs.vuUKQuarts Variable 20882

Constants 20883

StdConvs.CubicMetersPerAcreFoot Constant 20884
StdConvs.CubicMetersPerAcreInch Constant 20884
StdConvs.CubicMetersPerCord Constant 20885
StdConvs.CubicMetersPerCordFoot Constant 20885
StdConvs.CubicMetersPerCubicFoot Constant 20885
StdConvs.CubicMetersPerCubicInch Constant 20886
StdConvs.CubicMetersPerCubicMile Constant 20886
StdConvs.CubicMetersPerCubicYard Constant 20886
StdConvs.CubicMetersPerUKBucket Constant 20887
StdConvs.CubicMetersPerUKBushel Constant 20887
StdConvs.CubicMetersPerUKGallon Constant 20887
StdConvs.CubicMetersPerUKGill Constant 20888
StdConvs.CubicMetersPerUKOunce Constant 20888
StdConvs.CubicMetersPerUKPeck Constant 20888
StdConvs.CubicMetersPerUKPint Constant 20889
StdConvs.CubicMetersPerUKPottle Constant 20889
StdConvs.CubicMetersPerUKQuart Constant 20889

StdConvs.CubicMetersPerUSDryBucket Constant 20890
StdConvs.CubicMetersPerUSDryBushel Constant 20890
StdConvs.CubicMetersPerUSDryGallon Constant 20890
StdConvs.CubicMetersPerUSDryPeck Constant 20891
StdConvs.CubicMetersPerUSDryPint Constant 20891
StdConvs.CubicMetersPerUSDryQuart Constant 20891
StdConvs.CubicMetersPerUSFluidCup Constant 20892
StdConvs.CubicMetersPerUSFluidGallon Constant 20892
StdConvs.CubicMetersPerUSFluidGill Constant 20892
StdConvs.CubicMetersPerUSFluidOunce Constant 20893
StdConvs.CubicMetersPerUSFluidPint Constant 20893
StdConvs.CubicMetersPerUSFluidQuart Constant 20893
StdConvs.CubicMetersPerUSFluidTablespoon Constant 20894
StdConvs.CubicMetersPerUSFluidTeaspoon Constant 20894
StdConvs.GramsPerDrams Constant 20894
StdConvs.GramsPerGrains Constant 20895
StdConvs.GramsPerLongTons Constant 20895
StdConvs.GramsPerOunces Constant 20895
StdConvs.GramsPerPound Constant 20896
StdConvs.GramsPerStones Constant 20896
StdConvs.GramsPerTons Constant 20896
StdConvs.MetersPerAstronomicalUnit Constant 20897
StdConvs.MetersPerChain Constant 20897
StdConvs.MetersPerCubit Constant 20897
StdConvs.MetersPerFathom Constant 20898
StdConvs.MetersPerFoot Constant 20898
StdConvs.MetersPerFurlong Constant 20898
StdConvs.MetersPerHand Constant 20899
StdConvs.MetersPerInch Constant 20899
StdConvs.MetersPerLightSecond Constant 20899
StdConvs.MetersPerLightYear Constant 20900
StdConvs.MetersPerLink Constant 20900
StdConvs.MetersPerMile Constant 20900
StdConvs.MetersPerNauticalMiles Constant 20901
StdConvs.MetersPerPace Constant 20901
StdConvs.MetersPerParsec Constant 20901
StdConvs.MetersPerPica Constant 20902
StdConvs.MetersPerPoint Constant 20902
StdConvs.MetersPerRod Constant 20902

StdConvs.MetersPerYard Constant 20903
StdConvs.SquareMetersPerAcre Constant 20903
StdConvs.SquareMetersPerSquareFoot Constant 20903
StdConvs.SquareMetersPerSquareInch Constant 20904
StdConvs.SquareMetersPerSquareMile Constant 20904
StdConvs.SquareMetersPerSquareRod Constant 20904
StdConvs.SquareMetersPerSquareYard Constant 20905

StdCtrls Namespace 20907

Classes 20909
TButton Class 20909
TButton Members 20910
TButton.Create Constructor 20920
TButton Methods 20921
TButton Properties 20923
TButton Events 20948
TButtonActionLink Class 20976
TButtonActionLink Members 20977
TButtonActionLink Fields 20979
TButtonActionLink Methods 20979
TButtonControl Class 20981
TButtonControl Members 20981
TButtonControl.Create Constructor 20992
TButtonControl Methods 20992
TButtonControl Properties 20994
TCheckBox Class 20996
TCheckBox Members 20997
TCheckBox Properties 21008
TCheckBox Events 21036
TComboBox Class 21064
TComboBox Members 21065
TComboBox Properties 21077
TComboBox Events 21111
TCustomCheckBox Class 21138
TCustomCheckBox Members 21138
TCustomCheckBox.Create Constructor 21148
TCustomCheckBox Methods 21149
TCustomCheckBox Properties 21152

TCustomCombo Class 21154
TCustomCombo Members 21155
TCustomCombo.Create Constructor 21166
TCustomCombo.Destroy Destructor 21166
TCustomCombo Fields 21167
TCustomCombo Methods 21169
TCustomCombo Properties 21183
TCustomCombo Events 21196
TCustomComboBox Class 21201
TCustomComboBox Members 21202
TCustomComboBox.Create Constructor 21214
TCustomComboBox.Destroy Destructor 21214
TCustomComboBox Methods 21214
TCustomComboBox Properties 21219
TCustomComboBox Events 21224
TCustomComboBoxStrings Class 21225
TCustomComboBoxStrings Members 21226
TCustomComboBoxStrings Methods 21228
TCustomComboBoxStrings Properties 21232
TCustomEdit Class 21232
TCustomEdit Members 21233
TCustomEdit.Create Constructor 21244
TCustomEdit Methods 21244
TCustomEdit Properties 21259
TCustomEdit Events 21275
TCustomGroupBox Class 21275
TCustomGroupBox Members 21276
TCustomGroupBox.Create Constructor 21286
TCustomGroupBox Methods 21286
TCustomLabel Class 21288
TCustomLabel Members 21288
TCustomLabel.Create Constructor 21295
TCustomLabel Methods 21296
TCustomLabel Properties 21299
TCustomListBox Class 21308
TCustomListBox Members 21309
TCustomListBox.Create Constructor 21320
TCustomListBox.Destroy Destructor 21320
TCustomListBox Fields 21321

TCustomListBox Methods 21321
TCustomListBox Properties 21341
TCustomListBox Events 21357
TCustomMemo Class 21362
TCustomMemo Members 21363
TCustomMemo.Create Constructor 21374
TCustomMemo.Destroy Destructor 21374
TCustomMemo Methods 21375
TCustomMemo Properties 21379
TCustomStaticText Class 21387
TCustomStaticText Members 21387
TCustomStaticText.Create Constructor 21397
TCustomStaticText Methods 21398
TCustomStaticText Properties 21400
TEdit Class 21402
TEdit Members 21403
TEdit Properties 21414
TEdit Events 21445
TGroupBox Class 21474
TGroupBox Members 21474
TGroupBox Properties 21485
TGroupBox Events 21511
TLabel Class 21544
TLabel Members 21545
TLabel Properties 21552
TLabel Events 21579
TListBox Class 21598
TListBox Members 21598
TListBox Properties 21610
TListBox Events 21644
TMemo Class 21678
TMemo Members 21679
TMemo Properties 21690
TMemo Events 21724
TRadioButton Class 21753
TRadioButton Members 21754
TRadioButton.Create Constructor 21764
TRadioButton Methods 21764
TRadioButton Properties 21767

TRadioButton Events 21793
TScrollBar Class 21821
TScrollBar Members 21822
TScrollBar.Create Constructor 21832
TScrollBar Methods 21833
TScrollBar Properties 21835
TScrollBar Events 21866
TStaticText Class 21887
TStaticText Members 21888
TStaticText Properties 21898
TStaticText Events 21925

Structs, Records, Enums 21945
StdCtrls.TCheckBoxState Enumeration 21945
StdCtrls.TComboBoxStyle Enumeration 21946
StdCtrls.TEditCharCase Enumeration 21946
StdCtrls.TEllipsisPosition Enumeration 21947
StdCtrls.TListBoxStyle Enumeration 21948
StdCtrls.TScrollCode Enumeration 21949
StdCtrls.TScrollStyle Enumeration 21950
StdCtrls.TSelection Record 21950
StdCtrls.TStaticBorderStyle Enumeration 21951
StdCtrls.TTextLayout Enumeration 21951

Types 21953
StdCtrls.TButtonActionLinkClass Type 21953
StdCtrls.TCustomComboBoxStringsClass Type 21953
StdCtrls.TDrawItemEvent Type 21954
StdCtrls.TLBFindDataEvent Type 21954
StdCtrls.TLBGetDataEvent Type 21955
StdCtrls.TLBGetDataObjectEvent Type 21955
StdCtrls.TMeasureItemEvent Type 21955
StdCtrls.TOwnerDrawState Type 21956
StdCtrls.TScrollEvent Type 21957

StrUtils Namespace 21959

Functions 21962
StrUtils.AnsiContainsStr Function 21963
StrUtils.AnsiContainsText Function 21964
StrUtils.AnsiEndsStr Function 21964

StrUtils.AnsiEndsText Function 21965
StrUtils.AnsiIndexStr Function 21965
StrUtils.AnsiIndexText Function 21966
StrUtils.AnsiLeftStr Function 21966
StrUtils.AnsiMatchStr Function 21967
StrUtils.AnsiMatchText Function 21967
StrUtils.AnsiMidStr Function 21968
StrUtils.AnsiReplaceStr Function 21969
StrUtils.AnsiReplaceText Function 21969
StrUtils.AnsiResemblesText Function 21970
StrUtils.AnsiReverseString Function 21970
StrUtils.AnsiRightStr Function 21971
StrUtils.AnsiStartsStr Function 21971
StrUtils.AnsiStartsText Function 21972
StrUtils.ContainsStr Function 21972
StrUtils.ContainsText Function 21973
StrUtils.DecodeSoundexInt Function 21973
StrUtils.DecodeSoundexWord Function 21974
StrUtils.DupeString Function 21974
StrUtils.EndsStr Function 21975
StrUtils.EndsText Function 21975
StrUtils.IfThen Function 21976
StrUtils.IndexStr Function 21976
StrUtils.IndexText Function 21977
StrUtils.LeftBStr Function 21977
StrUtils.LeftStr Function 21978
StrUtils.MatchStr Function 21979
StrUtils.MatchText Function 21979
StrUtils.MidBStr Function 21980
StrUtils.MidStr Function 21980
StrUtils.PosEx Function 21981
StrUtils.RandomFrom Function 21981
StrUtils.ReplaceStr Function 21982
StrUtils.ReplaceText Function 21982
StrUtils.ResemblesText Function 21983
StrUtils.ReverseString Function 21983
StrUtils.RightBStr Function 21984
StrUtils.RightStr Function 21984
StrUtils.SearchBuf Function 21985

StrUtils.Soundex Function 21986
StrUtils.SoundexCompare Function 21986
StrUtils.SoundexInt Function 21987
StrUtils.SoundexProc Function 21987
StrUtils.SoundexSimilar Function 21988
StrUtils.SoundexWord Function 21989
StrUtils.StartsStr Function 21989
StrUtils.StartsText Function 21990
StrUtils.StuffString Function 21990

Structs, Records, Enums 21992
StrUtils.TStringSeachOption Enumeration 21992

Types 21993
StrUtils.TCompareTextProc Type 21993
StrUtils.TSoundexIntLength Type 21993
StrUtils.TSoundexLength Type 21994
StrUtils.TStringSearchOptions Type 21994

Variables 21995
StrUtils.AnsiResemblesProc Variable 21995
StrUtils.ResemblesProc Variable 21995

Constants 21997
StrUtils.WordDelimiters Constant 21997

SvcMgr Namespace 21999

Classes 22001
TDependencies Class 22001
TDependencies Members 22001
TDependencies.Create Constructor 22003
TDependencies Methods 22003
TDependencies Properties 22004
TDependency Class 22005
TDependency Members 22006
TDependency Methods 22007
TDependency Properties 22007
TEventLogger Class 22009
TEventLogger Members 22009
TEventLogger.Create Constructor 22010
TEventLogger.Destroy Destructor 22010

TEventLogger Methods 22010
TService Class 22011
TService Members 22012
TService.CreateNew Constructor 22015
TService.Destroy Destructor 22016
TService Methods 22016
TService Properties 22020
TService Events 22028
TServiceApplication Class 22032
TServiceApplication Members 22033
TServiceApplication.Create Constructor 22036
TServiceApplication.Destroy Destructor 22036
TServiceApplication Methods 22036
TServiceApplication Properties 22039
TServiceThread Class 22040
TServiceThread Members 22041
TServiceThread.Create Constructor 22042
TServiceThread Methods 22042

Structs, Records, Enums 22047
SvcMgr.TCurrentStatus Enumeration 22047
SvcMgr.TErrorSeverity Enumeration 22048
SvcMgr.TServiceType Enumeration 22048
SvcMgr.TStartType Enumeration 22049

Types 22051
SvcMgr.TContinueEvent Type 22051
SvcMgr.TPauseEvent Type 22051
SvcMgr.TServiceController Type 22052
SvcMgr.TServiceEvent Type 22052
SvcMgr.TStartEvent Type 22052
SvcMgr.TStopEvent Type 22053

Variables 22054
SvcMgr.Application Variable 22054

Constants 22055
SvcMgr.CM_SERVICE_CONTROL_CODE Constant 22055

SyncObjs Namespace 22057

Classes 22058

TCriticalSection Class 22058
TCriticalSection Members 22059
TCriticalSection.Create Constructor 22059
TCriticalSection.Destroy Destructor 22060
TCriticalSection Fields 22060
TCriticalSection Methods 22060
TEvent Class 22063
TEvent Members 22063
Create Constructor 22064
TEvent Methods 22065
THandleObject Class 22066
THandleObject Members 22067
THandleObject.Create Constructor 22068
THandleObject.Destroy Destructor 22068
THandleObject Fields 22069
THandleObject Methods 22069
THandleObject Properties 22070
TMutex Class 22071
TMutex Members 22071
Create Constructor 22072
TMutex Methods 22073
TSimpleEvent Class 22074
TSynchroObject Class 22075
TSynchroObject Members 22075
TSynchroObject Methods 22076

Structs, Records, Enums 22078
SyncObjs.TWaitResult Enumeration 22078

System Namespace 22079

Classes 22092
AnsiString Class 22093
AnsiString Members 22094
AnsiString::~AnsiString Destructor 22096
AnsiString Constructor 22096
AnsiString Structures 22100
AnsiString Enumerations 22101
AnsiString Methods 22102
AnsiString Operators 22123

AnsiString Friends 22127
AutoCmd Class 22127
AutoCmd Members 22128
AutoCmd::~AutoCmd Destructor 22129
AutoCmd Constructor 22129
AutoCmd Data Members 22130
AutoCmd Methods 22131
AutoCmd Operators 22138
Currency Class 22145
Currency Members 22145
Currency Constructor 22147
Currency Operators 22148
Currency Friends 22163
CurrencyBase Class 22166
CurrencyBase Members 22166
CurrencyBase Data Members 22166
DelphiInterface Template 22167
DelphiInterface Members 22167
DelphiInterface::~DelphiInterface Destructor 22168
DelphiInterface Constructor 22168
DelphiInterface Methods 22169
DelphiInterface Operators 22169
DynamicArray Template 22172
DynamicArray Members 22174
DynamicArray::~DynamicArray Destructor 22174
DynamicArray Constructor 22175
DynamicArray Methods 22175
DynamicArray Operators 22179
DynamicArray Properties 22180
DynArrayException Class 22182
DynArrayNullData Class 22183
DynArrayOutOfRange Class 22183
DynArrayOutOfRange Members 22183
DynArrayOutOfRange::DynArrayOutOfRange Constructor 22184
DynArrayOutOfRange Data Members 22184
Function Class 22184
Function Members 22185
Function Constructor 22186
Function Methods 22186

NamedParm Class 22187
NamedParm Members 22188
NamedParm Constructor 22188
NamedParm Data Members 22191
NamedParm Methods 22192
OleVariant Template 22193
OleVariant Members 22195
OleVariant::~OleVariant Destructor 22197
OleVariant Constructor 22198
OleVariant Operators 22209
OpenArray Template 22216
OpenArray Members 22216
OpenArray::~OpenArray Destructor 22216
OpenArray Constructor 22216
OpenArray Methods 22222
OpenArray Operators 22223
OpenArrayCount Template 22223
OpenArrayCount Members 22224
OpenArrayCount Constructor 22224
OpenArrayCount Methods 22230
Procedure Class 22230
Procedure Members 22231
Procedure Constructor 22232
Procedure Methods 22232
PropertyGet Class 22233
PropertyGet Members 22234
PropertyGet Constructor 22235
PropertyGet Methods 22235
PropertySet Class 22236
PropertySet Members 22237
PropertySet Constructor 22238
PropertySet Methods 22238
SmallString Template 22239
SmallString Members 22239
SmallString Constructor 22240
SmallString Operators 22241
SmallStringBase Template 22242
SmallStringBase Members 22242
SmallStringBase Data Members 22242

sPointer Template 22243
sPointer Members 22243
sPointer::~sPointer Destructor 22244
sPointer Constructor 22244
sPointer Operators 22245
StaticArray Structure 22245
StaticArray Members 22246
StaticArray Data Members 22246
StaticArray Operators 22246
TAggregatedObject Class 22248
TAggregatedObject Members 22248
TAggregatedObject::Create Constructor 22249
TAggregatedObject Methods 22250
TAggregatedObject Properties 22251
TClassHelperBase Class 22251
TClassHelperBase Members 22252
TClassHelperBase._Create Constructor 22253
TClassHelperBase Fields 22253
TContainedObject Class 22253
TContainedObject Members 22254
TContainedObject Methods 22255
TDateTime Class 22256
TDateTime Members 22257
TDateTime Constructor 22258
TDateTime Enumerations 22260
TDateTime Methods 22260
TDateTime Operators 22266
TDateTimeBase Class 22275
TDateTimeBase Members 22275
TDateTimeBase Data Members 22275
TInterfacedObject Class 22276
TInterfacedObject Members 22276
TInterfacedObject Data Members 22277
TInterfacedObject Methods 22278
TInterfacedObject Properties 22281
TMetaClass Class 22281
TMetaClass Members 22283
TMetaClass Methods 22283
TObject Class 22285

TObject Members 22286
TObject::Create Constructor 22287
TObject::Destroy Destructor 22287
TObject Methods 22288
Variant Class 22307
Variant Members 22309
Variant::~Variant Destructor 22311
Variant Constructor 22312
Variant Methods 22316
Variant Operators 22341
Variant Friends 22375
VariantError Class 22375
VariantError Members 22375
VariantError::VariantError Constructor 22376
VariantError Data Members 22376
VariantError Operators 22376
WideString Class 22377
WideString Members 22377
WideString::~WideString Destructor 22378
WideString Constructor 22379
WideString Methods 22381
WideString Operators 22386

Interfaces 22394
IDispatch Interface 22394
IDispatch Members 22394
IDispatch Methods 22394
IInterface Interface 22396
IInterface Members 22396
IInterface Methods 22397
IInvokable Interface 22398

Functions 22400
System::!= Function 22405
System::__currMul Function 22405
System._CVR_PROBE Function 22405
System._CVR_STMTPROBE Function 22406
System::_roundToInt64 Function 22406
System::+ Function 22406
System::< Function 22407

System::<< Function 22407
System::<< Function 22407
System::<< Function 22408
System::<= Function 22408
System::== Function 22408
System::> Function 22409
System::>= Function 22409
System::>> Function 22409
System::>> Function 22410
System.Abs Function 22410
System.AcquireExceptionObject Function 22410
System.AddModuleUnloadProc Function 22411
System.Addr Function 22412
System.AllocMem Function 22413
System.AnsiToUtf8 Function 22415
System.Append Function 22415
System.ArcTan Function 22416
System.Assert Function 22417
System.Assigned Function 22419
System.AssignFile Function 22420
System.AttemptToUseSharedMemoryManager Function 22422
System.BeginThread Function 22422
System.BlockRead Function 22423
System.BlockWrite Function 22425
System.Break Function 22426
System.ChDir Function 22429
System.Chr Function 22430
System.Close Function 22432
System.CloseFile Function 22433
System.CompToCurrency Function 22435
System.CompToDouble Function 22435
System.Concat Function 22436
System.Continue Function 22437
System.Copy Function 22438
System.Cos Function 22440
System.CurrencyToComp Function 22441
System.Dec Function 22441
System.Delete Function 22443
System.Dispose Function 22444

System.DoubleToComp Function 22446
System.DynArrayClear Function 22446
System.DynArrayDim Function 22446
System.DynArraySetLength Function 22447
System.EndThread Function 22447
System.EnumModules Function 22447
System.EnumResourceModules Function 22448
System.Eof Function 22449
System.Eoln Function 22450
System.Erase Function 22451
System.Error Function 22451
System.ExceptAddr Function 22452
System.ExceptObject Function 22452
System.Exclude Function 22453
System.Exit Function 22453
System.Exp Function 22455
System.FilePos Function 22456
System.FileSize Function 22457
System.FillChar Function 22459
System.Finalize Function 22460
System.FindClassHInstance Function 22461
System.FindHInstance Function 22461
System.FindResourceHInstance Function 22462
System.Flush Function 22462
System.FPower10 Function 22463
System.Frac Function 22463
System.FreeMem Function 22464
System.FreeMemory Function 22466
System.Get8087CW Function 22466
System.GetDir Function 22467
System.GetHeapStatus Function 22468
System.GetLastError Function 22469
System.GetMem Function 22469
System.GetMemory Function 22470
System.GetMemoryManager Function 22472
System.GetMemoryManagerState Function 22472
System.GetMemoryMap Function 22473
System.GetMinimumBlockAlignment Function 22473
System.GetModuleFileName Function 22474

System.GetVariantManager Function 22474
System.Halt Function 22475
System.Hi Function 22476
System.High Function 22476
System.Inc Function 22478
System.Include Function 22479
System.Initialize Function 22480
System.Insert Function 22480
System.Int Function 22481
System.IOResult Function 22482
System.IsMemoryManagerSet Function 22483
System.IsVariantManagerSet Function 22483
System.Length Function 22484
System.Ln Function 22485
System.Lo Function 22486
System.LoadResourceModule Function 22486
System.LoadResString Function 22487
System.Low Function 22487
System.MkDir Function 22488
System.Move Function 22490
System.New Function 22491
System.Odd Function 22493
System.OleStrToString Function 22493
System.OleStrToStrVar Function 22494
System.Ord Function 22494
System.ParamCount Function 22495
System.ParamStr Function 22496
System.Pi Function 22497
System.Pos Function 22497
System.Pred Function 22499
System.Ptr Function 22501
System.PUCS4Chars Function 22501
System.RaiseList Function 22502
System.Random Function 22502
System.Randomize Function 22503
System.Read Function 22504
System.ReadLn Function 22506
System.ReallocMem Function 22507
System.ReallocMemory Function 22508

System.RegisterExpectedMemoryLeak Function 22508
System.RegisterModule Function 22509
System.ReleaseExceptionObject Function 22509
System.RemoveModuleUnloadProc Function 22510
System.Rename Function 22510
System.Reset Function 22511
System.Rewrite Function 22513
System.RmDir Function 22515
System.Round Function 22516
System.RunError Function 22517
System.Seek Function 22518
System.SeekEof Function 22519
System.SeekEoln Function 22520
System.Set8087CW Function 22520
System.SetInOutRes Function 22521
System.SetLastError Function 22521
System.SetLength Function 22522
System.SetLineBreakStyle Function 22522
System.SetMemoryManager Function 22523
System.SetMinimumBlockAlignment Function 22524
System.SetMultiByteConversionCodePage Function 22524
System.SetRaiseList Function 22525
System.SetString Function 22525
System.SetTextBuf Function 22526
System.SetVariantManager Function 22527
System.ShareMemoryManager Function 22527
System.Sin Function 22528
System.SizeOf Function 22528
System.Slice Function 22529
System.Sqr Function 22529
System.Sqrt Function 22530
System.Str Function 22530
System.StringOfChar Function 22531
System.StringToOleStr Function 22531
System.StringToWideChar Function 22532
System.Succ Function 22532
System.Swap Function 22533
System.SysAllocMem Function 22533
System.SysFreeMem Function 22534

System.SysGetMem Function 22534
System.SysReallocMem Function 22535
System.SysRegisterExpectedMemoryLeak Function 22536
System.SysUnregisterExpectedMemoryLeak Function 22536
System.TextStart Function 22536
System.Trunc Function 22537
System.Truncate Function 22537
System.UCS4StringToWideString Function 22538
System.UnicodeToUtf8 Function 22538
System.UniqueString Function 22539
System.UnregisterExpectedMemoryLeak Function 22539
System.UnregisterModule Function 22540
System.UpCase Function 22540
System.UTF8Decode Function 22541
System.UTF8Encode Function 22541
System.Utf8ToAnsi Function 22542
System.Utf8ToUnicode Function 22542
System.Val Function 22543
System.VarArrayRedim Function 22544
System::VarArrayRedim Function 22544
System::VarCast Function 22545
System.VarClear Function 22545
System::VarClear Function 22546
System.WideCharLenToString Function 22546
System.WideCharLenToStrVar Function 22546
System.WideCharToString Function 22547
System.WideCharToStrVar Function 22547
System.WideStringToUCS4String Function 22548
System.Write Function 22548
System.WriteLn Function 22550

Structs, Records, Enums 22552

System.PackageInfoTable Record 22553
System.PackageUnitEntry Record 22553
System.TCallDesc Record 22553
System.TChunkStatus Enumeration 22554
System.TCVModInfo Record 22555
System.TDispatchMessage Record 22555
System.TDispDesc Record 22556
System.TDynArrayTypeInfo Record 22556

System.TFileRec Record 22557
System.TGUID Record 22558
System.THeapStatus Record 22558
System.TInitContext Record 22560
System::TInterfaceEntry Record 22560
System::TInterfaceTable Record 22561
System.TLibModule Record 22561
System.TMemoryManager Record 22562
System.TMemoryManagerEx Record 22563
System.TMemoryManagerState Record 22564
System.TMethod Record 22565
System.TMinimumBlockAlignment Enumeration 22566
System.TModuleUnloadRec Record 22566
System.TResStringRec Record 22567
System.TRuntimeError Enumeration 22567
System.TSmallBlockTypeState Record 22568
System.TTextLineBreakStyle Enumeration 22569
System.TTextRec Record 22569
System.TVarArray Record 22570
System.TVarArrayBound Record 22571
System.TVarData Record 22571
System.TVariantManager Record 22575
System::TVarRec Record 22577
System::TCallDesc Structure 22578
System::TDispDesc Structure 22578

Types **22579**

System::ArrayOfboolean Type 22581
System::ArrayOfbyte Type 22582
System::ArrayOfdouble Type 22582
System::ArrayOfint Type 22582
System::ArrayOflong Type 22583
System::ArrayOfshort Type 22583
System::ArrayOfstring Type 22583
System::ArrayOfwidestring Type 22584
System.GetPackageInfoTable Type 22584
System.HGLOBAL Type 22584
System.HINST Type 22585
System.HMODULE Type 22585
System.HRESULT Type 22585

System.HRSRC Type 22586
System.IntegerArray Type 22586
System.IUnknown Type 22586
System.PackageInfo Type 22587
System.PAnsiString Type 22587
System.PBoolean Type 22587
System.PByte Type 22588
System::PCallDesc Type 22588
System.PCardinal Type 22589
System.PComp Type 22589
System.PCurrency Type 22589
System.PCVModInfo Type 22590
System.PDate Type 22590
System.PDateTime Type 22591
System.PDispatch Type 22591
System::PDispDesc Type 22591
System.PDouble Type 22592
System.PDynArrayTypeInfo Type 22592
System.PError Type 22592
System.PExtended Type 22593
System.PGUID Type 22593
System.PInitContext Type 22593
System.PInt64 Type 22594
System.PInteger Type 22594
System.PIntegerArray Type 22595
System::PInterfaceEntry Type 22595
System::PInterfaceTable Type 22595
System.PLibModule Type 22596
System.PLongint Type 22596
System.PLongWord Type 22596
System.PMemoryManager Type 22597
System.PMemoryManagerEx Type 22597
System.PModuleUnloadRec Type 22598
System.PointerArray Type 22598
System.POleVariant Type 22598
System.PPAnsiChar Type 22599
System.PPChar Type 22599
System.PPCharArray Type 22599
System.PPDispatch Type 22600

System.PPointer Type 22600
System.PPointerArray Type 22600
System.PPUnknown Type 22601
System.PPWideChar Type 22601
System.PResStringRec Type 22601
System.PShortInt Type 22602
System.PShortString Type 22602
System.PSingle Type 22602
System.PSmallInt Type 22603
System.PString Type 22603
System.PTextBuf Type 22604
System.PUCS2Char Type 22604
System.PUCS4Char Type 22604
System.PUCS4CharArray Type 22605
System.PUnitEntryTable Type 22605
System.PUnknown Type 22605
System.PUTF8String Type 22606
System.PVarArray Type 22606
System.PVarArrayBoundArray Type 22606
System.PVarArrayCoorArray Type 22607
System.PVarData Type 22607
System.PVariant Type 22607
System.PVariantManager Type 22608
System::PVarRec Type 22608
System.PWideString Type 22608
System.PWord Type 22609
System.PWordBool Type 22609
System::SCODE Type 22609
System.TAssertErrorProc Type 22610
System::TBooleanDynArray Type 22610
System.TBoundArray Type 22610
System::TByteDynArray Type 22611
System::TCardinalDynArray Type 22611
System.TClass Type 22611
System.TDateTime Type 22612
System.TDLLProc Type 22613
System.TDLLProcEx Type 22613
System::TDoubleDynArray Type 22614
System.TEnumModuleFunc Type 22614

System.TEnumModuleFuncLW Type 22614
System.TFileIOFunc Type 22615
System.THandle Type 22615
System::TInt64DynArray Type 22615
System::TIntegerDynArray Type 22616
System.TInterfacedClass Type 22616
System::TLongWordDynArray Type 22616
System.TMemoryMap Type 22617
System.TModuleUnloadProc Type 22617
System.TModuleUnloadProcLW Type 22618
System.TPCharArray Type 22618
System.TResourceHandle Type 22619
System.TSafeCallErrorProc Type 22619
System::TShortIntDynArray Type 22619
System::TSingleDynArray Type 22620
System.TSmallBlockTypeStates Type 22620
System::TSmallIntDynArray Type 22620
System::TStringDynArray Type 22621
System.TSystemThreadEndProc Type 22621
System.TSystemThreadFuncProc Type 22621
System.TTextBuf Type 22622
System.TTextIOFunc Type 22622
System.TThreadFunc Type 22622
System.TUCS4CharArray Type 22623
System.TVarArrayBoundArray Type 22623
System.TVarArrayCoorArray Type 22623
System.TVarOp Type 22624
System.TVarType Type 22625
System::TWideStringDynArray Type 22625
System::TWordDynArray Type 22626
System.UCS2Char Type 22626
System.UCS4Char Type 22626
System.UCS4String Type 22627
System.UnitEntryTable Type 22627
System.UTF8String Type 22627

Variables **22629**

System.AbstractErrorProc Variable 22630
System.AllocMemCount Variable 22630
System.AllocMemSize Variable 22631

System.AssertErrorProc Variable 22633
System.CmdLine Variable 22635
System.CmdShow Variable 22635
System.DebugHook Variable 22636
System.Default8087CW Variable 22636
System.DefaultTextLineBreakStyle Variable 22636
System.DispCallByIDProc Variable 22637
System.DLLProc Variable 22637
System.envp Variable 22638
System.ErrorAddr Variable 22638
System.ErrorProc Variable 22639
System.ErrOutput Variable 22640
System.ExceptClsProc Variable 22640
System.ExceptionClass Variable 22641
System.ExceptObjProc Variable 22641
System.ExceptProc Variable 22641
System.ExitCode Variable 22642
System.ExitProc Variable 22643
System.ExitProcessProc Variable 22644
System.FileMode Variable 22644
System.HeapAllocFlags Variable 22645
System.HPrevInst Variable 22645
System.InitProc Variable 22646
System.Input Variable 22647
System.IsConsole Variable 22647
System.IsLibrary Variable 22648
System.IsMultiThread Variable 22648
System.JITEnable Variable 22649
System.LibModuleList Variable 22650
System.MainInstance Variable 22650
System.MainThreadID Variable 22651
System::MaxDispArgs Variable 22651
System.ModuleUnloadList Variable 22651
System.NeverSleepOnMMThreadContention Variable 22652
System.NoErrMsg Variable 22652
System.Output Variable 22653
System.RaiseExceptionProc Variable 22653
System.RandSeed Variable 22654
System.ReportMemoryLeaksOnShutdown Variable 22654

System.RTLUnwindProc Variable 22655
System.SafeCallErrorProc Variable 22655
System.SystemThreadEndProc Variable 22656
System.SystemThreadFuncProc Variable 22656
System.Test8086 Variable 22656
System.Test8087 Variable 22657
System.TestFDIV Variable 22657
System.VarAddRefProc Variable 22657
System.VarClearProc Variable 22658
System.VarCopyProc Variable 22658
System.VarToLStrProc Variable 22658
System.VarToWStrProc Variable 22659

Constants 22660

System.CompilerVersion Constant 22662
System.CPUi386 Constant 22662
System.CPUi486 Constant 22663
System.CPUPentium Constant 22663
System.E_NOINTERFACE Constant 22663
System.E_NOTIMPL Constant 22664
System.E_UNEXPECTED Constant 22664
System.fmClosed Constant 22664
System.fmInOut Constant 22665
System.fmInput Constant 22665
System.fmOutput Constant 22665
System.MaxInt Constant 22666
System.MaxLongint Constant 22666
System.NumSmallBlockTypes Constant 22666
System.opAdd Constant 22667
System.opAnd Constant 22667
System.opCmpEQ Constant 22668
System.opCmpGE Constant 22668
System.opCmpGT Constant 22669
System.opCmpLE Constant 22669
System.opCmpLT Constant 22670
System.opCmpNE Constant 22670
System.opCompare Constant 22671
System.opDivide Constant 22671
System.opIntDivide Constant 22672
System.opModulus Constant 22672

System.opMultiply Constant 22673
System.opNegate Constant 22673
System.opNot Constant 22674
System.opOr Constant 22674
System.opShiftLeft Constant 22675
System.opShiftRight Constant 22675
System.opSubtract Constant 22676
System.opXor Constant 22676
System.RTLVersion Constant 22677
System.S_FALSE Constant 22677
System.S_OK Constant 22677
System.sLineBreak Constant 22678
System.tfCRLF Constant 22678
System.varAny Constant 22678
System.varArray Constant 22679
System.varBoolean Constant 22680
System.varByRef Constant 22680
System.varByte Constant 22681
System.varCurrency Constant 22681
System.varDate Constant 22682
System.varDispatch Constant 22682
System.varDouble Constant 22683
System.varEmpty Constant 22683
System.varError Constant 22684
System.varInt64 Constant 22684
System.varInteger Constant 22685
System.varLongWord Constant 22685
System.varNull Constant 22686
System.varOleStr Constant 22686
System.varShortInt Constant 22687
System.varSingle Constant 22687
System.varSmallint Constant 22688
System.varStrArg Constant 22688
System.varString Constant 22688
System.varTypeMask Constant 22689
System.varUnknown Constant 22690
System.varVariant Constant 22690
System.varWord Constant 22691
System.vmtAddRef Constant 22691

System.vmtAfterConstruction Constant 22691
System.vmtAutoTable Constant 22692
System.vmtBeforeDestruction Constant 22692
System.vmtClassName Constant 22692
System.vmtCreateObject Constant 22693
System.vmtDefaultHandler Constant 22693
System.vmtDestroy Constant 22693
System.vmtDispatch Constant 22694
System.vmtDynamicTable Constant 22694
System.vmtFieldTable Constant 22694
System.vmtFreeInstance Constant 22695
System.vmtInitTable Constant 22695
System.vmtInstanceSize Constant 22695
System.vmtIntfTable Constant 22696
System.vmtMethodTable Constant 22696
System.vmtNewInstance Constant 22696
System.vmtParent Constant 22697
System.vmtQueryInterface Constant 22697
System.vmtRelease Constant 22697
System.vmtSafeCallException Constant 22698
System.vmtSelfPtr Constant 22698
System.vmtTypeInfo Constant 22698
System.vtAnsiString Constant 22699
System.vtBoolean Constant 22699
System.vtChar Constant 22700
System.vtClass Constant 22700
System.vtCurrency Constant 22701
System.vtExtended Constant 22701
System.vtInt64 Constant 22702
System.vtInteger Constant 22702
System.vtInterface Constant 22703
System.vtObject Constant 22703
System.vtPChar Constant 22704
System.vtPointer Constant 22704
System.vtPWideChar Constant 22705
System.vtString Constant 22705
System.vtVariant Constant 22706
System.vtWideChar Constant 22706
System.vtWideString Constant 22706

SysUtils Namespace 22709

Classes **22721**
EAbort Class 22722
EAbstractError Class 22722
EAccessViolation Class 22723
EAssertionFailed Class 22723
EControlC Class 22724
EConvertError Class 22724
EDivByZero Class 22726
EExternal Class 22727
EExternal Members 22728
EExternal Fields 22729
EExternalException Class 22729
EHeapException Class 22730
EHeapException Members 22731
EHeapException Methods 22732
EInOutError Class 22732
EInOutError Members 22734
EInOutError Fields 22735
EIntError Class 22735
EIntfCastError Class 22736
EIntOverflow Class 22736
EInvalidCast Class 22737
EInvalidContainer Class 22738
EInvalidInsert Class 22738
EInvalidOp Class 22738
EInvalidPointer Class 22739
EMathError Class 22739
EOSError Class 22740
EOSError Members 22741
EOSError Fields 22742
EOutOfMemory Class 22742
EOverflow Class 22743
EPackageError Class 22744
EPrivilege Class 22744
EPropReadOnly Class 22745
EPropWriteOnly Class 22745

ERangeError Class 22745
ESafecallException Class 22747
EStackOverflow Class 22747
EUnderflow Class 22748
EVariantError Class 22748
EWin32Error Class 22749
Exception Class 22750
Exception Members 22751
Exception.Create Constructor 22752
Exception.CreateFmt Constructor 22754
Exception.CreateFmtHelp Constructor 22755
Exception.CreateHelp Constructor 22755
CreateRes Constructor 22756
CreateResFmt Constructor 22756
CreateResFmtHelp Constructor 22757
CreateResHelp Constructor 22759
Exception Properties 22760
EZeroDivide Class 22761
TLanguages Class 22762
TLanguages Members 22762
TLanguages.Create Constructor 22763
TLanguages Methods 22763
TLanguages Properties 22763
TMultiReadExclusiveWriteSynchronizer Class 22766
TMultiReadExclusiveWriteSynchronizer Members 22767
TMultiReadExclusiveWriteSynchronizer.Create Constructor 22769
TMultiReadExclusiveWriteSynchronizer.Destroy Destructor 22769
TMultiReadExclusiveWriteSynchronizer Methods 22769
TMultiReadExclusiveWriteSynchronizer Properties 22771
TSimpleRWSync Class 22772
TSimpleRWSync Members 22772
TSimpleRWSync.Create Constructor 22773
TSimpleRWSync.Destroy Destructor 22774
TSimpleRWSync Methods 22774
TThreadLocalCounter Class 22775
TThreadLocalCounter Members 22776
TThreadLocalCounter.Destroy Destructor 22776
TThreadLocalCounter Methods 22776

Interfaces 22778

IReadWriteSync Interface 22778
IReadWriteSync Members 22778
IReadWriteSync Methods 22778

Functions 22780
SysUtils.Abort Function 22786
SysUtils.AddExitProc Function 22787
SysUtils.AddTerminateProc Function 22787
SysUtils.AdjustLineBreaks Function 22788
SysUtils.AnsiCompareFileName Function 22788
SysUtils.AnsiCompareStr Function 22789
SysUtils.AnsiCompareText Function 22790
SysUtils.AnsiDequotedStr Function 22790
SysUtils.AnsiExtractQuotedStr Function 22791
SysUtils.AnsiLastChar Function 22791
SysUtils.AnsiLowerCase Function 22792
SysUtils.AnsiLowerCaseFileName Function 22792
SysUtils.AnsiPos Function 22793
SysUtils.AnsiQuotedStr Function 22793
SysUtils.AnsiSameStr Function 22794
SysUtils.AnsiSameText Function 22794
SysUtils.AnsiStrComp Function 22795
SysUtils.AnsiStrIComp Function 22795
SysUtils.AnsiStrLastChar Function 22796
SysUtils.AnsiStrLComp Function 22796
SysUtils.AnsiStrLIComp Function 22797
SysUtils.AnsiStrLower Function 22798
SysUtils.AnsiStrPos Function 22798
SysUtils.AnsiStrRScan Function 22799
SysUtils.AnsiStrScan Function 22799
SysUtils.AnsiStrUpper Function 22800
SysUtils.AnsiUpperCase Function 22800
SysUtils.AnsiUpperCaseFileName Function 22801
SysUtils.AppendStr Function 22801
SysUtils.AssignStr Function 22802
SysUtils.Beep Function 22802
SysUtils.BoolToStr Function 22804
SysUtils.ByteToCharIndex Function 22806
SysUtils.ByteToCharLen Function 22807
SysUtils.ByteType Function 22807

SysUtils.CallTerminateProcs Function 22808
SysUtils.ChangeFileExt Function 22808
SysUtils.ChangeFilePath Function 22810
SysUtils.CharLength Function 22810
SysUtils.CharToByteIndex Function 22811
SysUtils.CharToByteLen Function 22811
SysUtils.CheckWin32Version Function 22812
SysUtils.CompareMem Function 22812
SysUtils.CompareStr Function 22813
SysUtils.CompareText Function 22814
SysUtils.CreateDir Function 22817
SysUtils.CreateGUID Function 22819
SysUtils.CurrentYear Function 22819
SysUtils.CurrToStr Function 22819
SysUtils.CurrToStrF Function 22820
SysUtils.Date Function 22820
SysUtils.DateTimeToFileDate Function 22821
SysUtils.DateTimeToStr Function 22822
SysUtils.DateTimeToString Function 22823
SysUtils.DateTimeToSystemTime Function 22825
SysUtils.DateTimeToTimeStamp Function 22825
SysUtils.DateToStr Function 22825
SysUtils.DayOfWeek Function 22827
SysUtils.DecodeDate Function 22828
SysUtils.DecodeDateFully Function 22829
SysUtils.DecodeTime Function 22830
SysUtils.DeleteFile Function 22831
SysUtils.DirectoryExists Function 22832
SysUtils.DiskFree Function 22833
SysUtils.DiskSize Function 22835
SysUtils.DisposeStr Function 22836
SysUtils.EncodeDate Function 22836
SysUtils.EncodeTime Function 22837
SysUtils.ExceptionErrorMessage Function 22839
SysUtils.ExcludeTrailingBackslash Function 22839
SysUtils.ExcludeTrailingPathDelimiter Function 22840
SysUtils.ExpandFileName Function 22840
SysUtils.ExpandFileNameCase Function 22841
SysUtils.ExpandUNCFileName Function 22842

SysUtils.ExtractFileDir Function 22843
SysUtils.ExtractFileDrive Function 22843
SysUtils.ExtractFileExt Function 22844
SysUtils.ExtractFileName Function 22844
SysUtils.ExtractFilePath Function 22846
SysUtils.ExtractRelativePath Function 22848
SysUtils.ExtractShortPathName Function 22849
SysUtils.FileAge Function 22850
SysUtils.FileClose Function 22852
SysUtils.FileCreate Function 22855
SysUtils.FileDateToDateTime Function 22858
SysUtils.FileExists Function 22860
SysUtils.FileGetAttr Function 22863
SysUtils.FileGetDate Function 22866
SysUtils.FileIsReadOnly Function 22866
SysUtils.FileOpen Function 22867
SysUtils.FileRead Function 22870
SysUtils.FileSearch Function 22872
SysUtils.FileSeek Function 22873
SysUtils.FileSetAttr Function 22875
SysUtils.FileSetDate Function 22878
SysUtils.FileSetReadOnly Function 22878
SysUtils.FileWrite Function 22879
SysUtils.FinalizePackage Function 22882
SysUtils.FindClose Function 22882
SysUtils.FindCmdLineSwitch Function 22884
SysUtils.FindFirst Function 22884
SysUtils.FindNext Function 22887
SysUtils.FloatToCurr Function 22889
SysUtils.FloatToDateTime Function 22889
SysUtils.FloatToDecimal Function 22890
SysUtils.FloatToStr Function 22890
SysUtils.FloatToStrF Function 22891
SysUtils.FloatToText Function 22892
SysUtils.FloatToTextFmt Function 22893
SysUtils.FmtLoadStr Function 22893
SysUtils.FmtStr Function 22894
SysUtils.ForceDirectories Function 22895
SysUtils.Format Function 22897

SysUtils.FormatBuf Function 22900
SysUtils.FormatCurr Function 22901
SysUtils.FormatDateTime Function 22902
SysUtils.FormatFloat Function 22904
SysUtils.FreeAndNil Function 22907
SysUtils.GDAL Function 22909
SysUtils.GetCurrentDir Function 22910
SysUtils.GetEnvironmentVariable Function 22911
SysUtils.GetFileVersion Function 22912
SysUtils.GetFormatSettings Function 22912
SysUtils.GetLocaleChar Function 22913
SysUtils.GetLocaleFormatSettings Function 22913
SysUtils.GetLocaleStr Function 22913
SysUtils.GetModuleName Function 22914
SysUtils.GetPackageDescription Function 22915
SysUtils.GetPackageInfo Function 22915
SysUtils.GetTime Function 22916
SysUtils.GUIDToString Function 22916
SysUtils.IncAMonth Function 22917
SysUtils.IncludeTrailingBackslash Function 22918
SysUtils.IncludeTrailingPathDelimiter Function 22918
SysUtils.IncMonth Function 22919
SysUtils.InitializePackage Function 22920
SysUtils.IntToHex Function 22920
SysUtils.IntToStr Function 22921
SysUtils.IsAssembly Function 22922
SysUtils.IsDelimiter Function 22923
SysUtils.IsEqualGUID Function 22923
SysUtils.IsLeapYear Function 22924
SysUtils.IsPathDelimiter Function 22925
SysUtils.IsValidIdent Function 22925
SysUtils.Languages Function 22926
SysUtils.LastDelimiter Function 22926
SysUtils.LoadPackage Function 22927
SysUtils.LoadStr Function 22927
SysUtils.LowerCase Function 22928
SysUtils.MSecsToTimeStamp Function 22929
SysUtils.NewStr Function 22929
SysUtils.NextCharIndex Function 22930

SysUtils.Now Function 22930
SysUtils.OutOfMemoryError Function 22931
SysUtils.QuotedStr Function 22931
SysUtils.RaiseLastOSError Function 22932
SysUtils.RaiseLastWin32Error Function 22932
SysUtils.RCS Function 22933
SysUtils.RemoveDir Function 22933
SysUtils.RenameFile Function 22934
SysUtils.ReplaceDate Function 22936
SysUtils.ReplaceTime Function 22937
SysUtils.RPR Function 22938
SysUtils.SafeLoadLibrary Function 22938
SysUtils.SameFileName Function 22939
SysUtils.SameStr Function 22939
SysUtils.SameText Function 22940
SysUtils.SetCurrentDir Function 22940
SysUtils.ShowException Function 22941
SysUtils.Sleep Function 22941
SysUtils.StrAlloc Function 22942
SysUtils.StrBufSize Function 22942
SysUtils.StrByteType Function 22943
SysUtils.StrCat Function 22943
SysUtils.StrCharLength Function 22944
SysUtils.StrComp Function 22945
SysUtils.StrCopy Function 22946
SysUtils.StrDispose Function 22947
SysUtils.StrECopy Function 22948
SysUtils.StrEnd Function 22949
SysUtils.StrFmt Function 22950
SysUtils.StrIComp Function 22951
SysUtils.StringReplace Function 22951
SysUtils.StringToGUID Function 22952
SysUtils.StrLCat Function 22952
SysUtils.StrLComp Function 22953
SysUtils.StrLCopy Function 22953
SysUtils.StrLen Function 22954
SysUtils.StrLFmt Function 22954
SysUtils.StrLIComp Function 22955
SysUtils.StrLower Function 22956

SysUtils.StrMove Function 22956
SysUtils.StrNew Function 22957
SysUtils.StrNextChar Function 22958
SysUtils.StrPas Function 22958
SysUtils.StrPCopy Function 22958
SysUtils.StrPLCopy Function 22959
SysUtils.StrPos Function 22959
SysUtils.StrRScan Function 22960
SysUtils.StrScan Function 22960
SysUtils.StrToBool Function 22961
SysUtils.StrToBoolDef Function 22961
SysUtils.StrToCurr Function 22962
SysUtils.StrToCurrDef Function 22963
SysUtils.StrToDate Function 22963
SysUtils.StrToDateDef Function 22965
SysUtils.StrToDateTime Function 22965
SysUtils.StrToDateTimeDef Function 22966
SysUtils.StrToFloat Function 22967
SysUtils.StrToFloatDef Function 22969
SysUtils.StrToInt Function 22970
SysUtils.StrToInt64 Function 22970
SysUtils.StrToInt64Def Function 22971
SysUtils.StrToIntDef Function 22971
SysUtils.StrToTime Function 22972
SysUtils.StrToTimeDef Function 22972
SysUtils.StrUpper Function 22973
SysUtils.Supports Function 22973
SysUtils.SysErrorMessage Function 22974
SysUtils.SystemTimeToDateTime Function 22974
SysUtils.TextToFloat Function 22975
SysUtils.Time Function 22976
SysUtils.TimeStampToDateTime Function 22976
SysUtils.TimeStampToMSecs Function 22977
SysUtils.TimeToStr Function 22977
SysUtils.Trim Function 22978
SysUtils.TrimLeft Function 22979
SysUtils.TrimRight Function 22979
SysUtils.TryEncodeDate Function 22980
SysUtils.TryEncodeTime Function 22980

SysUtils.TryFloatToCurr Function 22981
SysUtils.TryFloatToDateTime Function 22982
SysUtils.TryStrToBool Function 22982
SysUtils.TryStrToCurr Function 22983
SysUtils.TryStrToDate Function 22983
SysUtils.TryStrToDateTime Function 22984
SysUtils.TryStrToFloat Function 22985
SysUtils.TryStrToInt Function 22986
SysUtils.TryStrToInt64 Function 22986
SysUtils.TryStrToTime Function 22987
SysUtils.UnloadPackage Function 22987
SysUtils.UpperCase Function 22988
SysUtils.WideCompareStr Function 22989
SysUtils.WideCompareText Function 22990
SysUtils.WideFmtStr Function 22990
SysUtils.WideFormat Function 22991
SysUtils.WideFormatBuf Function 22991
SysUtils.WideLowerCase Function 22992
SysUtils.WideSameStr Function 22992
SysUtils.WideSameText Function 22993
SysUtils.WideUpperCase Function 22993
SysUtils.Win32Check Function 22994
SysUtils.WrapText Function 22994

Structs, Records, Enums 22996

SysUtils.Int64Rec Record 22996
SysUtils.LongRec Record 22997
SysUtils.TExceptionRecord Record 22997
SysUtils.TFilenameCaseMatch Enumeration 22998
SysUtils.TFloatFormat Enumeration 22999
SysUtils.TFloatRec Record 23000
SysUtils.TFloatValue Enumeration 23001
SysUtils.TFormatSettings Record 23001
SysUtils.TLangRec Record 23002
SysUtils.TLocaleOptions Enumeration 23003
SysUtils.TMbcsByteType Enumeration 23004
SysUtils.TNameType Enumeration 23005
SysUtils.TSearchRec Record 23005
SysUtils.TSysLocale Record 23008
SysUtils.TThreadInfo Record 23009

SysUtils.TTimeStamp Record 23009
SysUtils.WordRec Record 23010

Types 23012
SysUtils.ExceptClass Type 23012
SysUtils.PByteArray Type 23013
SysUtils.PDayTable Type 23013
SysUtils.PExceptionRecord Type 23014
SysUtils.PThreadInfo Type 23014
SysUtils.PWordArray Type 23014
SysUtils.TByteArray Type 23015
SysUtils.TBytes Type 23015
SysUtils.TDayTable Type 23015
SysUtils.TFileName Type 23016
SysUtils.TIntegerSet Type 23016
SysUtils.TMREWSync Type 23017
SysUtils.TPackageInfoProc Type 23017
SysUtils.TProcedure Type 23018
SysUtils.TReplaceFlags Type 23018
SysUtils.TSysCharSet Type 23019
SysUtils.TTerminateProc Type 23019
SysUtils.TValidatePackageProc Type 23019
SysUtils.TWordArray Type 23020

Variables 23021
SysUtils.CurrencyDecimals Variable 23022
SysUtils.CurrencyFormat Variable 23022
SysUtils.CurrencyString Variable 23023
SysUtils.DateSeparator Variable 23023
SysUtils.DecimalSeparator Variable 23023
SysUtils.EmptyStr Variable 23024
SysUtils.EmptyWideStr Variable 23024
SysUtils.EraNames Variable 23024
SysUtils.EraYearOffsets Variable 23025
SysUtils.FalseBoolStrs Variable 23025
SysUtils.GetDiskFreeSpaceEx Variable 23026
SysUtils.HexDisplayPrefix Variable 23026
SysUtils.LeadBytes Variable 23026
SysUtils.ListSeparator Variable 23027
SysUtils.LongDateFormat Variable 23027

SysUtils.LongDayNames Variable 23027
SysUtils.LongMonthNames Variable 23028
SysUtils.LongTimeFormat Variable 23028
SysUtils.NegCurrFormat Variable 23029
SysUtils.NullStr Variable 23029
SysUtils.NullWideStr Variable 23030
SysUtils.ShortDateFormat Variable 23030
SysUtils.ShortDayNames Variable 23031
SysUtils.ShortMonthNames Variable 23031
SysUtils.ShortTimeFormat Variable 23031
SysUtils.SysLocale Variable 23032
SysUtils.ThousandSeparator Variable 23032
SysUtils.TimeAMString Variable 23033
SysUtils.TimePMString Variable 23033
SysUtils.TimeSeparator Variable 23033
SysUtils.TrueBoolStrs Variable 23034
SysUtils.TwoDigitYearCenturyWindow Variable 23034
SysUtils.Win32BuildNumber Variable 23035
SysUtils.Win32CSDVersion Variable 23035
SysUtils.Win32MajorVersion Variable 23036
SysUtils.Win32MinorVersion Variable 23036
SysUtils.Win32Platform Variable 23037

Constants **23038**

SysUtils.DateDelta Constant 23039
SysUtils.DefaultFalseBoolStr Constant 23039
SysUtils.DefaultTrueBoolStr Constant 23040
SysUtils.DriveDelim Constant 23041
SysUtils.faAnyFile Constant 23041
SysUtils.faArchive Constant 23042
SysUtils.faDirectory Constant 23042
SysUtils.faHidden Constant 23043
SysUtils.faReadOnly Constant 23044
SysUtils.faSymLink Constant 23044
SysUtils.faSysFile Constant 23045
SysUtils.faVolumeID Constant 23045
SysUtils.fmOpenRead Constant 23046
SysUtils.fmOpenReadWrite Constant 23047
SysUtils.fmOpenWrite Constant 23047
SysUtils.fmShareCompat Constant 23048

SysUtils.fmShareDenyNone Constant 23048
SysUtils.fmShareDenyRead Constant 23049
SysUtils.fmShareDenyWrite Constant 23049
SysUtils.fmShareExclusive Constant 23049
SysUtils.HoursPerDay Constant 23050
SysUtils.MaxCurrency Constant 23050
SysUtils.MaxDateTime Constant 23051
SysUtils.MaxEraCount Constant 23051
SysUtils.MinCurrency Constant 23051
SysUtils.MinDateTime Constant 23052
SysUtils.MinsPerDay Constant 23052
SysUtils.MinsPerHour Constant 23053
SysUtils.MonthDays Constant 23054
SysUtils.MSecsPerDay Constant 23054
SysUtils.MSecsPerSec Constant 23055
SysUtils.PathDelim Constant 23055
SysUtils.PathSep Constant 23055
SysUtils.pfBCB4Produced Constant 23056
SysUtils.pfConsumerBCB Constant 23056
SysUtils.pfConsumerCompat Constant 23057
SysUtils.pfConsumerDelphi Constant 23057
SysUtils.pfConsumerMask Constant 23058
SysUtils.pfDelphi4Produced Constant 23058
SysUtils.pfDesignOnly Constant 23059
SysUtils.pfExeModule Constant 23059
SysUtils.pfIgnoreDupUnits Constant 23060
SysUtils.pfLibraryModule Constant 23060
SysUtils.pfModuleTypeMask Constant 23060
SysUtils.pfNeverBuild Constant 23061
SysUtils.pfPackageModule Constant 23061
SysUtils.pfProducerMask Constant 23062
SysUtils.pfProducerUndefined Constant 23062
SysUtils.pfRunOnly Constant 23063
SysUtils.pfV3Produced Constant 23063
SysUtils.SecsPerDay Constant 23064
SysUtils.SecsPerMin Constant 23064
SysUtils.SwitchChars Constant 23065
SysUtils.ufImplicitUnit Constant 23065
SysUtils.ufMainUnit Constant 23066

SysUtils.ufOrgWeakUnit Constant 23066
SysUtils.ufPackageUnit Constant 23067
SysUtils.ufWeakPackageUnit Constant 23067
SysUtils.ufWeakUnit Constant 23068
SysUtils.UnixDateDelta Constant 23068

TabNotBk Namespace 23069

Classes 23070
TTabbedNotebook Class 23070
TTabbedNotebook Members 23070
TTabbedNotebook.Create Constructor 23081
TTabbedNotebook.Destroy Destructor 23082
TTabbedNotebook Methods 23082
TTabbedNotebook Properties 23088
TTabbedNotebook Events 23106
TTabPage Class 23112
TTabPage Members 23112
TTabPage.Create Constructor 23122
TTabPage Methods 23123
TTabPage Properties 23124

Types 23131
TabNotBk.TPageChangeEvent Type 23131

Constants 23132
TabNotBk.CM_TABFONTCHANGED Constant 23132

Tabs Namespace 23133

Classes 23134
TScroller Class 23134
TScroller Members 23134
TScroller.Create Constructor 23145
TScroller.Destroy Destructor 23145
TScroller Methods 23146
TScroller Properties 23146
TScroller Events 23148
TTabList Class 23149
TTabList Members 23149
TTabList.Create Constructor 23152

TTabList Methods 23152
TTabSet Class 23159
TTabSet Members 23159
TTabSet.Create Constructor 23170
TTabSet.Destroy Destructor 23170
TTabSet Methods 23171
TTabSet Properties 23177
TTabSet Events 23203

Structs, Records, Enums 23229
Tabs.TEdgePart Enumeration 23229
Tabs.TEdgeType Enumeration 23229
Tabs.TScrollBtn Enumeration 23230
Tabs.TScrollOrientation Enumeration 23230
Tabs.TTabSetTabStyle Enumeration 23231

Types 23232
Tabs.TDrawTabEvent Type 23232
Tabs.TMeasureTabEvent Type 23232
Tabs.TTabChangeEvent Type 23233
Tabs.TTabSetDrawTabEvent Type 23233
Tabs.TTabStyle Type 23234

TConnect Namespace 23235

Classes 23236
TLocalConnection Class 23236
TLocalConnection Members 23237
TLocalConnection.Create Constructor 23240
TLocalConnection.Destroy Destructor 23241
TLocalConnection Methods 23241
TLocalConnection Properties 23250

ToolWin Namespace 23253

Classes 23254
TToolDockForm Class 23254
TToolDockForm Members 23254
TToolDockForm.Create Constructor 23268
TToolDockForm Methods 23269
TToolDockObject Class 23271

TToolDockObject Members 23271
TToolDockObject.Create Constructor 23273
TToolDockObject Methods 23273
TToolWindow Class 23274
TToolWindow Members 23275
TToolWindow.Create Constructor 23285
TToolWindow Methods 23285
TToolWindow Properties 23287

Structs, Records, Enums 23289
ToolWin.TEdgeBorder Enumeration 23289
ToolWin.TEdgeStyle Enumeration 23290
ToolWin.TSizingOrientation Enumeration 23290

Types 23292
ToolWin.TEdgeBorders Type 23292

Types Namespace 23293

Functions 23295
Types.Bounds Function 23295
Types.CenterPoint Function 23295
Types.EqualRect Function 23296
Types.IntersectRect Function 23296
Types.IsRectEmpty Function 23297
Types.OffsetRect Function 23297
Types.Point Function 23298
Types.PtInRect Function 23298
Types.Rect Function 23299
Types.SmallPoint Function 23299
Types.UnionRect Function 23300

Structs, Records, Enums 23301
Types.tagSIZE Record 23301
Types.TPoint Record 23301
Types.TRect Record 23302
Types.TSmallPoint Record 23303

Types 23304
Types.DWORD Type 23304
Types.PByte Type 23305
Types.PDouble Type 23305

Types.PInteger Type 23306
Types.PLongint Type 23306
Types.PPoint Type 23306
Types.PRect Type 23307
Types.PSize Type 23307
Types.PSmallInt Type 23308
Types.PSmallPoint Type 23308
Types.SIZE Type 23308
Types.tagPOINT Type 23309
Types.TBooleanDynArray Type 23309
Types.TByteDynArray Type 23309
Types.TCardinalDynArray Type 23310
Types.TDoubleDynArray Type 23310
Types.TInt64DynArray Type 23311
Types.TIntegerDynArray Type 23311
Types.TLongWordDynArray Type 23311
Types.TShortIntDynArray Type 23312
Types.TSingleDynArray Type 23312
Types.TSize Type 23313
Types.TSmallIntDynArray Type 23313
Types.TStringDynArray Type 23313
Types.TValueRelationship Type 23314
Types.TWideStringDynArray Type 23314
Types.TWordDynArray Type 23315

Constants 23316

Types.EqualsValue Constant 23316
Types.GreaterThanValue Constant 23316
Types.LessThanValue Constant 23317
Types.RT_RCDATA Constant 23317

TypeTrans Namespace 23319

Classes 23320

ETypeTransException Class 23320
TTypeTranslator Class 23320
TTypeTranslator Members 23321
TTypeTranslator.Create Constructor 23321
TTypeTranslator.Destroy Destructor 23321
TTypeTranslator Methods 23321

Functions **23324**
TypeTrans.FloatToStrEx Function 23324
TypeTrans.GetEnumValueEx Function 23324
TypeTrans.SetEnumPropEx Function 23325
TypeTrans.StrToFloatEx Function 23325

Variables **23326**
TypeTrans.TypeTranslator Variable 23326

TypInfo Namespace 23327

Classes **23331**
EPropertyConvertError Class 23331
EPropertyError Class 23331
TPublishableVariantType Class 23332
TPublishableVariantType Members 23333
TPublishableVariantType Methods 23335

Functions **23338**
TypInfo.FreeAndNilProperties Function 23339
TypInfo.GetDynArrayProp Function 23340
TypInfo.GetEnumName Function 23341
TypInfo.GetEnumProp Function 23341
TypInfo.GetEnumValue Function 23342
TypInfo.GetFloatProp Function 23343
TypInfo.GetInt64Prop Function 23343
TypInfo.GetInterfaceProp Function 23344
TypInfo.GetMethodProp Function 23345
TypInfo.GetObjectProp Function 23346
TypInfo.GetObjectPropClass Function 23347
TypInfo.GetOrdProp Function 23347
TypInfo.GetPropInfo Function 23348
TypInfo.GetPropInfos Function 23349
TypInfo.GetPropList Function 23350
TypInfo.GetPropValue Function 23351
TypInfo.GetSetElementName Function 23351
TypInfo.GetSetElementValue Function 23352
TypInfo.GetSetProp Function 23352
TypInfo.GetStrProp Function 23353
TypInfo.GetTypeData Function 23354

TypInfo.GetVariantProp Function 23354
TypInfo.GetWideStrProp Function 23355
TypInfo.IsPublishedProp Function 23356
TypInfo.IsStoredProp Function 23356
TypInfo.PropIsType Function 23357
TypInfo.PropType Function 23358
TypInfo.SetDynArrayProp Function 23358
TypInfo.SetEnumProp Function 23359
TypInfo.SetFloatProp Function 23360
TypInfo.SetInt64Prop Function 23361
TypInfo.SetInterfaceProp Function 23362
TypInfo.SetMethodProp Function 23362
TypInfo.SetObjectProp Function 23363
TypInfo.SetOrdProp Function 23364
TypInfo.SetPropValue Function 23365
TypInfo.SetSetProp Function 23366
TypInfo.SetStrProp Function 23367
TypInfo.SetToString Function 23367
TypInfo.SetVariantProp Function 23368
TypInfo.SetWideStrProp Function 23369
TypInfo.SortPropList Function 23370
TypInfo.StringToSet Function 23370

Structs, Records, Enums 23372

TypInfo.TFloatType Enumeration 23372
TypInfo.TIntfFlag Enumeration 23373
TypInfo.TMethodKind Enumeration 23373
TypInfo.TOrdType Enumeration 23374
TypInfo.TParamFlag Enumeration 23375
TypInfo.TPropData Record 23376
TypInfo.TPropInfo Record 23376
TypInfo.TTypeData Record 23377
TypInfo.TTypeInfo Record 23379
TypInfo.TTypeKind Enumeration 23379

Types 23381

TypInfo.PPropInfo Type 23381
TypInfo.PPropList Type 23382
TypInfo.PPTypeInfo Type 23382
TypInfo.PTypeData Type 23383

TypInfo.PTypeInfo Type 23383
TypInfo.ShortStringBase Type 23384
TypInfo.TIntfFlags Type 23384
TypInfo.TIntfFlagsBase Type 23385
TypInfo.TParamFlags Type 23386
TypInfo.TParamFlagsBase Type 23386
TypInfo.TPropInfoProc Type 23386
TypInfo.TPropList Type 23387
TypInfo.TTypeKinds Type 23387

Variables **23388**
TypInfo.BooleanIdents Variable 23388
TypInfo.DotSep Variable 23388

Constants **23389**
TypInfo.tkAny Constant 23389
TypInfo.tkMethods Constant 23389
TypInfo.tkProperties Constant 23389

UDDIHelper Namespace 23391

Functions **23392**
UDDIHelper.GetBindingkeyAccessPoint Function 23392

ValEdit Namespace 23393

Classes **23394**
TItemProp Class 23394
TItemProp Members 23394
TItemProp.Create Constructor 23395
TItemProp.Destroy Destructor 23396
TItemProp Methods 23396
TItemProp Properties 23398
TValueListEditor Class 23401
TValueListEditor Members 23401
TValueListEditor.Create Constructor 23415
TValueListEditor.Destroy Destructor 23415
TValueListEditor Methods 23415
TValueListEditor Properties 23427
TValueListEditor Events 23460
TValueListStrings Class 23498

TValueListStrings Members 23498
TValueListStrings.Create Constructor 23501
TValueListStrings.Destroy Destructor 23501
TValueListStrings Methods 23501
TValueListStrings Properties 23507

Structs, Records, Enums 23509
ValEdit.TDisplayOption Enumeration 23509
ValEdit.TKeyOption Enumeration 23510

Types 23511
ValEdit.TDisplayOptions Type 23511
ValEdit.TGetPickListEvent Type 23511
ValEdit.TItemProps Type 23512
ValEdit.TKeyOptions Type 23512
ValEdit.TOnValidateEvent Type 23512

VarCmplx Namespace 23513

Functions 23515
VarCmplx.VarAsComplex Function 23516
VarCmplx.VarComplex Function 23516
VarCmplx.VarComplexAbs Function 23517
VarCmplx.VarComplexAbsSqr Function 23518
VarCmplx.VarComplexAngle Function 23518
VarCmplx.VarComplexArcCos Function 23519
VarCmplx.VarComplexArcCosH Function 23520
VarCmplx.VarComplexArcCot Function 23521
VarCmplx.VarComplexArcCotH Function 23522
VarCmplx.VarComplexArcCsc Function 23523
VarCmplx.VarComplexArcCscH Function 23524
VarCmplx.VarComplexArcSec Function 23525
VarCmplx.VarComplexArcSecH Function 23527
VarCmplx.VarComplexArcSin Function 23528
VarCmplx.VarComplexArcSinH Function 23529
VarCmplx.VarComplexArcTan Function 23530
VarCmplx.VarComplexArcTanH Function 23531
VarCmplx.VarComplexConjugate Function 23532
VarCmplx.VarComplexCos Function 23533
VarCmplx.VarComplexCosH Function 23534
VarCmplx.VarComplexCot Function 23535

VarCmplx.VarComplexCotH Function 23536
VarCmplx.VarComplexCreate Function 23537
VarCmplx.VarComplexCsc Function 23538
VarCmplx.VarComplexCscH Function 23539
VarCmplx.VarComplexExp Function 23540
VarCmplx.VarComplexFromPolar Function 23541
VarCmplx.VarComplexInverse Function 23541
VarCmplx.VarComplexLn Function 23542
VarCmplx.VarComplexLog10 Function 23543
VarCmplx.VarComplexLog2 Function 23543
VarCmplx.VarComplexLogN Function 23544
VarCmplx.VarComplexPower Function 23545
VarCmplx.VarComplexSec Function 23545
VarCmplx.VarComplexSecH Function 23546
VarCmplx.VarComplexSign Function 23548
VarCmplx.VarComplexSimplify Function 23548
VarCmplx.VarComplexSin Function 23548
VarCmplx.VarComplexSinH Function 23550
VarCmplx.VarComplexSqr Function 23551
VarCmplx.VarComplexSqrt Function 23551
VarCmplx.VarComplexTan Function 23552
VarCmplx.VarComplexTanH Function 23553
VarCmplx.VarComplexTimesImaginary Function 23554
VarCmplx.VarComplexTimesNegI Function 23555
VarCmplx.VarComplexTimesPosI Function 23555
VarCmplx.VarComplexTimesReal Function 23556
VarCmplx.VarComplexToPolar Function 23556
VarCmplx.VarIsComplex Function 23557

Variables **23559**
VarCmplx.ComplexNumberDefuzzAtZero Variable 23559
VarCmplx.ComplexNumberSymbol Variable 23559
VarCmplx.ComplexNumberSymbolBeforeImaginary Variable 23560

VarConv Namespace 23561

Functions **23562**
VarConv.VarAsConvert Function 23562
VarConv.VarConvert Function 23563
VarConv.VarConvertCreate Function 23564

VarConv.VarIsConvert Function 23565

Variants Namespace 23567

Classes **23572**
EVariantArrayCreateError Class 23572
EVariantArrayLockedError Class 23573
EVariantBadIndexError Class 23574
EVariantBadVarTypeError Class 23574
EVariantDispatchError Class 23575
EVariantInvalidArgError Class 23575
EVariantInvalidNullOpError Class 23576
EVariantInvalidOpError Class 23577
EVariantNotAnArrayError Class 23577
EVariantNotImplError Class 23578
EVariantOutOfMemoryError Class 23578
EVariantOverflowError Class 23579
EVariantRangeCheckError Class 23580
EVariantTypeCastError Class 23580
EVariantUnexpectedError Class 23581
TCustomVariantType Class 23581
TCustomVariantType Members 23582
Create Constructor 23584
TCustomVariantType.Destroy Destructor 23585
TCustomVariantType Methods 23585
TCustomVariantType Properties 23606
TInvokeableVariantType Class 23606
TInvokeableVariantType Members 23608
TInvokeableVariantType Methods 23609

Interfaces **23614**
IVarInstanceReference Interface 23614
IVarInstanceReference Members 23614
IVarInstanceReference Methods 23614
IVarInvokeable Interface 23615
IVarInvokeable Members 23615
IVarInvokeable Methods 23615

Functions **23617**
Variants._DispInvoke Function 23619
Variants._OleVarFromInt Function 23620

Variants._OleVarFromInt64 Function 23620
Variants._OleVarFromLStr Function 23620
Variants._OleVarFromPStr Function 23621
Variants._OleVarFromVar Function 23621
Variants._VarAdd Function 23621
Variants._VarAddRef Function 23622
Variants._VarAnd Function 23622
Variants._VarArrayGet Function 23622
Variants._VarArrayPut Function 23623
Variants._VarArrayRedim Function 23623
Variants._VarCast Function 23623
Variants._VarCastOle Function 23624
Variants._VarClear Function 23624
Variants._VarClr Function 23624
Variants._VarCmpEQ Function 23625
Variants._VarCmpGE Function 23625
Variants._VarCmpGT Function 23625
Variants._VarCmpLE Function 23626
Variants._VarCmpLT Function 23626
Variants._VarCmpNE Function 23626
Variants._VarCopy Function 23627
Variants._VarCopyNoInd Function 23627
Variants._VarDiv Function 23627
Variants._VarFromBool Function 23628
Variants._VarFromCurr Function 23628
Variants._VarFromDisp Function 23628
Variants._VarFromDynArray Function 23629
Variants._VarFromInt Function 23629
Variants._VarFromInt64 Function 23629
Variants._VarFromIntf Function 23630
Variants._VarFromLStr Function 23630
Variants._VarFromPStr Function 23630
Variants._VarFromReal Function 23631
Variants._VarFromTDateTime Function 23631
Variants._VarFromWStr Function 23631
Variants._VarMod Function 23632
Variants._VarMul Function 23632
Variants._VarNeg Function 23632
Variants._VarNot Function 23633

Variants._VarOr Function 23633
Variants._VarRDiv Function 23633
Variants._VarShl Function 23634
Variants._VarShr Function 23634
Variants._VarSub Function 23634
Variants._VarToBool Function 23635
Variants._VarToCurrency Function 23635
Variants._VarToDisp Function 23635
Variants._VarToDynArray Function 23636
Variants._VarToInt64 Function 23636
Variants._VarToInteger Function 23636
Variants._VarToIntf Function 23637
Variants._VarToLStr Function 23637
Variants._VarToPStr Function 23637
Variants._VarToReal Function 23638
Variants._VarToWStr Function 23638
Variants._VarXor Function 23638
Variants.DynArrayFromVariant Function 23639
Variants.DynArrayToVariant Function 23639
Variants.FindCustomVariantType Function 23640
Variants.FindVarData Function 23640
Variants.HandleConversionException Function 23641
Variants.Null Function 23641
Variants.Unassigned Function 23643
Variants.VarArrayAsPSafeArray Function 23643
Variants.VarArrayCreate Function 23644
Variants.VarArrayCreateError Function 23645
Variants.VarArrayDimCount Function 23645
Variants.VarArrayGet Function 23645
Variants.VarArrayHighBound Function 23646
Variants.VarArrayLock Function 23646
Variants.VarArrayLowBound Function 23647
Variants.VarArrayOf Function 23648
Variants.VarArrayPut Function 23648
Variants.VarArrayRef Function 23649
Variants.VarArrayUnlock Function 23649
Variants.VarAsError Function 23650
Variants.VarAsType Function 23650
Variants.VarCastError Function 23651

Variants.VarCheckEmpty Function 23651
Variants.VarCompareValue Function 23652
Variants.VarCopyNoInd Function 23652
Variants.VarEnsureRange Function 23653
Variants.VarFromDateTime Function 23653
Variants.VarInRange Function 23654
Variants.VarInvalidNullOp Function 23654
Variants.VarInvalidOp Function 23655
Variants.VarIsArray Function 23655
Variants.VarIsByRef Function 23656
Variants.VarIsClear Function 23656
Variants.VarIsCustom Function 23657
Variants.VarIsEmpty Function 23657
Variants.VarIsEmptyParam Function 23658
Variants.VarIsError Function 23659
Variants.VarIsFloat Function 23659
Variants.VarIsNull Function 23660
Variants.VarIsNumeric Function 23660
Variants.VarIsOrdinal Function 23661
Variants.VarIsStr Function 23661
Variants.VarIsType Function 23662
Variants.VarOverflowError Function 23662
Variants.VarRangeCheckError Function 23663
Variants.VarResultCheck Function 23663
Variants.VarSameValue Function 23664
Variants.VarSupports Function 23664
Variants.VarToDateTime Function 23665
Variants.VarToStr Function 23665
Variants.VarToStrDef Function 23666
Variants.VarToWideStr Function 23666
Variants.VarToWideStrDef Function 23667
Variants.VarType Function 23667
Variants.VarTypeAsText Function 23668
Variants.VarTypeIsValidArrayType Function 23669
Variants.VarTypeIsValidElementType Function 23669

Structs, Records, Enums **23671**

Variants.TBooleanToStringRule Enumeration 23671
Variants.TNullCompareRule Enumeration 23671
Variants.TVarCompareResult Enumeration 23672

Variants.TVariantRelationship Enumeration 23672

Types **23674**

Variants.TAnyProc Type 23674

Variants.TCustomVariantTypeClass Type 23674

Variants.TVarDataArray Type 23675

Variants.TVarDispProc Type 23675

Variables **23676**

Variants.BooleanToStringRule Variable 23676

Variants.ChangeAnyProc Variable 23677

Variants.ClearAnyProc Variable 23677

Variants.EmptyParam Variable 23677

Variants.NullAsStringValue Variable 23678

Variants.NullEqualityRule Variable 23678

Variants.NullMagnitudeRule Variable 23679

Variants.NullStrictConvert Variable 23679

Variants.OleVariantInt64AsDouble Variable 23680

Variants.PackVarCreation Variable 23680

Variants.RefAnyProc Variable 23681

Variants.VarDispProc Variable 23681

VCLCom Namespace 23683

Classes **23684**

TComponentFactory Class 23684

TComponentFactory Members 23684

TComponentFactory.Create Constructor 23686

TComponentFactory Methods 23686

WebAdapt Namespace 23689

Classes **23694**

EAdapterException Class 23696

EAdapterException Members 23696

EAdapterException.CreateID Constructor 23697

EAdapterException Properties 23698

EAdapterFieldException Class 23698

EAdapterFieldException Members 23699

EAdapterFieldException.Create Constructor 23700

EAdapterFieldException Properties 23700

TAbstractFieldValuesAdapter Class 23701
TAbstractFieldValuesAdapter Members 23702
TAbstractFieldValuesAdapter Methods 23708
TAdapter Class 23713
TAdapter Members 23715
TAdapter Properties 23721
TAdapter Events 23722
TAdapterAction Class 23726
TAdapterAction Members 23728
TAdapterAction Properties 23733
TAdapterAction Events 23735
TAdapterActions Class 23738
TAdapterActions Members 23738
TAdapterActions Methods 23742
TAdapterActions Properties 23744
TAdapterApplicationField Class 23745
TAdapterApplicationField Members 23747
TAdapterApplicationField Properties 23751
TAdapterApplicationTitleField Class 23752
TAdapterApplicationTitleField Members 23753
TAdapterApplicationTitleField Methods 23758
TAdapterBooleanField Class 23758
TAdapterBooleanField Members 23760
TAdapterBooleanField Properties 23766
TAdapterBooleanField Events 23768
TAdapterDefaultActions Class 23770
TAdapterDefaultActions Members 23770
TAdapterDefaultActions.Destroy Destructor 23774
TAdapterDefaultActions Methods 23775
TAdapterDefaultActions Properties 23777
TAdapterDefaultFields Class 23778
TAdapterDefaultFields Members 23779
TAdapterDefaultFields.Destroy Destructor 23783
TAdapterDefaultFields Methods 23783
TAdapterDefaultFields Properties 23786
TAdapterEndUserDisplayNameField Class 23787
TAdapterEndUserDisplayNameField Members 23788
TAdapterEndUserDisplayNameField Methods 23792
TAdapterEndUserField Class 23793

TAdapterEndUserField Members 23795
TAdapterEndUserField Properties 23799
TAdapterEndUserLoggedInField Class 23800
TAdapterEndUserLoggedInField Members 23802
TAdapterEndUserLoggedInField Methods 23806
TAdapterError Class 23807
TAdapterError Members 23808
TAdapterError.Create Constructor 23809
TAdapterError Methods 23809
TAdapterErrorsList Class 23810
TAdapterErrorsList Members 23811
TAdapterErrorsList.Create Constructor 23813
TAdapterErrorsList.Destroy Destructor 23814
TAdapterErrorsList Methods 23814
TAdapterErrorsList Properties 23817
TAdapterField Class 23818
TAdapterField Members 23820
TAdapterField Properties 23825
TAdapterField Events 23829
TAdapterFields Class 23830
TAdapterFields Members 23831
TAdapterFields Methods 23835
TAdapterFields Properties 23837
TAdapterFileField Class 23837
TAdapterFileField Members 23839
TAdapterFileField Properties 23845
TAdapterFileField Events 23848
TAdapterGotoPageAction Class 23849
TAdapterGotoPageAction Members 23851
TAdapterGotoPageAction Properties 23856
TAdapterGotoPageAction Events 23857
TAdapterHiddenField Class 23860
TAdapterHiddenField Members 23860
TAdapterHiddenField.Create Constructor 23861
TAdapterHiddenField Methods 23862
TAdapterHiddenFieldsList Class 23862
TAdapterHiddenFieldsList Members 23863
TAdapterHiddenFieldsList.Create Constructor 23865
TAdapterHiddenFieldsList.Destroy Destructor 23865

TAdapterHiddenFieldsList Methods 23865
TAdapterHiddenFieldsList Properties 23868
TAdapterImageField Class 23869
TAdapterImageField Members 23870
TAdapterImageField Properties 23876
TAdapterImageField Events 23880
TAdapterLoginField Class 23882
TAdapterLoginField Members 23884
TAdapterLoginField Properties 23888
TAdapterMemoField Class 23889
TAdapterMemoField Members 23891
TAdapterMemoField Properties 23897
TAdapterMemoField Events 23899
TAdapterMultiValueField Class 23900
TAdapterMultiValueField Members 23902
TAdapterMultiValueField Properties 23908
TAdapterMultiValueField Events 23911
TAdapterNamedDisplayField Class 23913
TAdapterNamedDisplayField Members 23915
TAdapterNamedDisplayField.Create Constructor 23919
TAdapterNamedDisplayField Methods 23919
TAdapterNamedDisplayField Properties 23923
TAdapterNamedField Class 23925
TAdapterNamedField Members 23926
TAdapterNamedField Methods 23930
TAdapterNamedField Properties 23942
TAdapterNextPageAction Class 23945
TAdapterNextPageAction Members 23947
TAdapterNextPageAction Methods 23952
TAdapterNextPageAction Properties 23953
TAdapterNextPageAction Events 23955
TAdapterNextPageField Class 23957
TAdapterNextPageField Members 23959
TAdapterNextPageField.Destroy Destructor 23963
TAdapterNextPageField Methods 23964
TAdapterNextPageField Properties 23966
TAdapterObjectErrorsList Class 23968
TAdapterObjectErrorsList Members 23968
TAdapterObjectErrorsList.Create Constructor 23970

TAdapterObjectErrorsList Methods 23970
TAdapterPasswordField Class 23971
TAdapterPasswordField Members 23973
TAdapterPasswordField Methods 23977
TAdapterPasswordField Properties 23978
TAdapterPrevPageAction Class 23979
TAdapterPrevPageAction Members 23981
TAdapterPrevPageAction Methods 23986
TAdapterPrevPageAction Properties 23987
TAdapterPrevPageAction Events 23989
TAdapterSubComponent Class 23991
TAdapterSubComponent Members 23992
TAdapterSubComponent Methods 23995
TAdapterSubComponent Properties 24006
TAdapterUpdateField Class 24007
TAdapterUpdateField Members 24009
TAdapterUpdateField.Create Constructor 24014
TAdapterUpdateField Methods 24014
TAdapterUpdateField Properties 24022
TAdapterUpdateField Events 24023
TAdapterUserNameField Class 24024
TAdapterUserNameField Members 24026
TAdapterUserNameField Methods 24030
TAdapterUserNameField Properties 24031
TApplicationAdapter Class 24032
TApplicationAdapter Members 24034
TApplicationAdapter Properties 24040
TApplicationAdapter Events 24041
TBaseAdapterAction Class 24043
TBaseAdapterAction Members 24044
TBaseAdapterAction Methods 24049
TBaseAdapterAction Properties 24072
TBaseAdapterAction Events 24075
TBaseAdapterErrorsList Class 24077
TBaseAdapterErrorsList Members 24078
TBaseAdapterErrorsList Methods 24079
TBaseAdapterErrorsList Properties 24085
TBaseAdapterMultiValueField Class 24086
TBaseAdapterMultiValueField Members 24087

TBaseAdapterMultiValueField.Create Constructor 24092
TBaseAdapterMultiValueField.Destroy Destructor 24093
TBaseAdapterMultiValueField Methods 24093
TBaseEndUserAdapter Class 24097
TBaseEndUserAdapter Members 24099
TBaseEndUserAdapter Methods 24105
TBaseEndUserAdapter Properties 24118
TBaseEndUserAdapter Events 24119
TBaseValuesListAdapter Class 24120
TBaseValuesListAdapter Members 24121
TBaseValuesListAdapter Methods 24127
TBaseValuesListField Class 24132
TBaseValuesListField Members 24133
TBaseValuesListField Methods 24138
TBaseValuesListField Properties 24138
TCustomAdapter Class 24139
TCustomAdapter Members 24141
TCustomAdapter.Create Constructor 24146
TCustomAdapter.Destroy Destructor 24146
TCustomAdapter Methods 24147
TCustomAdapter Properties 24181
TCustomAdapter Events 24184
TCustomAdapterAction Class 24188
TCustomAdapterAction Members 24190
TCustomAdapterAction Methods 24195
TCustomAdapterAction Properties 24198
TCustomAdapterAction Events 24198
TCustomAdapterBooleanField Class 24201
TCustomAdapterBooleanField Members 24202
TCustomAdapterBooleanField Methods 24207
TCustomAdapterBooleanField Events 24210
TCustomAdapterField Class 24211
TCustomAdapterField Members 24213
TCustomAdapterField Methods 24218
TCustomAdapterField Events 24219
TCustomAdapterFileField Class 24220
TCustomAdapterFileField Members 24222
TCustomAdapterFileField Methods 24228
TCustomAdapterFileField Properties 24231

TCustomAdapterFileField Events 24232
TCustomAdapterGotoPageAction Class 24233
TCustomAdapterGotoPageAction Members 24234
TCustomAdapterGotoPageAction Methods 24240
TCustomAdapterGotoPageAction Events 24244
TCustomAdapterImageField Class 24244
TCustomAdapterImageField Members 24246
TCustomAdapterImageField Methods 24252
TCustomAdapterImageField Properties 24258
TCustomAdapterImageField Events 24259
TCustomAdapterMemoField Class 24262
TCustomAdapterMemoField Members 24264
TCustomAdapterMemoField Methods 24269
TCustomAdapterMemoField Events 24272
TCustomAdapterMultiValueField Class 24273
TCustomAdapterMultiValueField Members 24275
TCustomAdapterMultiValueField Methods 24281
TCustomAdapterMultiValueField Properties 24284
TCustomAdapterMultiValueField Events 24284
TCustomAdapterPageAction Class 24286
TCustomAdapterPageAction Members 24287
TCustomAdapterPageAction Methods 24293
TCustomApplicationAdapter Class 24296
TCustomApplicationAdapter Members 24297
TCustomApplicationAdapter Methods 24303
TCustomApplicationAdapter Properties 24305
TCustomApplicationAdapterAction Class 24306
TCustomApplicationAdapterAction Members 24307
TCustomApplicationAdapterAction Properties 24312
TCustomEndUserAdapter Class 24313
TCustomEndUserAdapter Members 24314
TCustomEndUserAdapter Methods 24321
TCustomEndUserAdapter Events 24323
TCustomEndUserAdapterAction Class 24326
TCustomEndUserAdapterAction Members 24327
TCustomEndUserAdapterAction Properties 24333
TCustomEndUserSessionAdapter Class 24333
TCustomEndUserSessionAdapter Members 24335
TCustomEndUserSessionAdapter Methods 24342

TCustomEndUserSessionAdapter Properties 24344
TCustomLoginFormAdapter Class 24344
TCustomLoginFormAdapter Members 24346
TCustomLoginFormAdapter Methods 24352
TCustomLoginFormAdapter Properties 24359
TCustomLoginFormAdapter Events 24361
TCustomLoginFormAdapterAction Class 24363
TCustomLoginFormAdapterAction Members 24364
TCustomLoginFormAdapterAction Properties 24369
TCustomPagedAdapter Class 24370
TCustomPagedAdapterAction Class 24371
TCustomPagedAdapterAction Members 24372
TCustomPagedAdapterAction Methods 24377
TCustomPagedAdapterAction Properties 24378
TCustomStringsValuesList Class 24378
TCustomStringsValuesList Members 24380
TCustomStringsValuesList.Create Constructor 24386
TCustomStringsValuesList.Destroy Destructor 24386
TCustomStringsValuesList Methods 24387
TCustomStringsValuesList Properties 24390
TCustomStringsValuesList Events 24391
TDefaultFieldsAdapter Class 24392
TDefaultFieldsAdapter Members 24393
TDefaultFieldsAdapter Methods 24399
TDefaultFieldsAdapter Properties 24402
TDefaultFieldsPagedAdapter Class 24403
TDefaultFieldsPagedAdapter Members 24404
TDefaultFieldsPagedAdapter Methods 24410
TDefaultFieldsPagedAdapter Properties 24418
TDefaultFieldsPagedAdapter Events 24420
TEndUserAdapter Class 24421
TEndUserAdapter Members 24423
TEndUserAdapter Properties 24430
TEndUserAdapter Events 24431
TEndUserAdapterLoginFormAction Class 24436
TEndUserAdapterLoginFormAction Members 24437
TEndUserAdapterLoginFormAction Methods 24443
TEndUserAdapterLoginFormAction Properties 24444
TEndUserAdapterLoginFormAction Events 24445

TEndUserAdapterLogoutAction Class 24447
TEndUserAdapterLogoutAction Members 24448
TEndUserAdapterLogoutAction Methods 24454
TEndUserAdapterLogoutAction Properties 24455
TEndUserAdapterLogoutAction Events 24456
TEndUserSessionAdapter Class 24458
TEndUserSessionAdapter Members 24460
TEndUserSessionAdapter Properties 24467
TEndUserSessionAdapter Events 24469
TFieldValuesAdapter Class 24471
TFieldValuesAdapter Members 24473
TFieldValuesAdapter.Create Constructor 24479
TFieldValuesAdapter Methods 24479
TFieldValuesAdapter Properties 24482
TFieldValuesValueField Class 24483
TFieldValuesValueField Members 24484
TFieldValuesValueField Methods 24489
TFieldValuesValueField Properties 24490
TImplementedAdapterAction Class 24491
TLogicalPageList Class 24492
TLogicalPageList Members 24493
TLogicalPageList.Create Constructor 24500
TLoginFormAdapter Class 24500
TLoginFormAdapter Members 24502
TLoginFormAdapter Properties 24509
TLoginFormAdapter Events 24510
TLoginFormAdapterLoginAction Class 24514
TLoginFormAdapterLoginAction Members 24515
TLoginFormAdapterLoginAction Methods 24520
TLoginFormAdapterLoginAction Properties 24522
TLoginFormAdapterLoginAction Events 24523
TPagedAdapter Class 24525
TPagedAdapter Members 24527
TPagedAdapter Properties 24533
TPagedAdapter Events 24535
TStringsValuesList Class 24540
TStringsValuesList Members 24542
TStringsValuesList Properties 24548
TStringsValuesList Events 24550

TUpdateFileList Class 24550
TUpdateFileList Members 24551
TUpdateFileList.Create Constructor 24551
TUpdateFileList Properties 24551
TValuesListNameField Class 24552
TValuesListNameField Members 24553
TValuesListNameField Methods 24557
TValuesListValueField Class 24558
TValuesListValueField Members 24559
TValuesListValueField Methods 24564

Interfaces 24566
IAdapterEditor Interface 24566
IAdapterEditor Members 24567
IAdapterEditor Methods 24567
IAdapterNotifyAdapterChange Interface 24567
IAdapterNotifyAdapterChange Members 24568
IAdapterNotifyAdapterChange Methods 24568
ICheckOrigValue Interface 24568
ICheckOrigValue Members 24569
ICheckOrigValue Methods 24569
ICheckValueChange Interface 24569
ICheckValueChange Members 24570
ICheckValueChange Methods 24570
IClearAdapterValues Interface 24570
IClearAdapterValues Members 24571
IClearAdapterValues Methods 24571
ICreateActionRequestContext Interface 24571
ICreateActionRequestContext Members 24571
ICreateActionRequestContext Methods 24572
IEchoActionFieldValue Interface 24572
IEchoActionFieldValue Members 24572
IEchoActionFieldValue Methods 24573
IEchoActionFieldValue Properties 24573
IEchoAdapterFieldValues Interface 24574
IEchoAdapterFieldValues Members 24574
IEchoAdapterFieldValues Methods 24574
IEchoAdapterFieldValues Properties 24575
IExecuteValue Interface 24575
IExecuteValue Members 24576

IExecuteValue Methods 24576
IFullyQualifyInputNames Interface 24577
IFullyQualifyInputNames Members 24577
IFullyQualifyInputNames Methods 24577
IGetAdapterErrorsList Interface 24578
IGetAdapterErrorsList Members 24578
IGetAdapterErrorsList Methods 24578
IGetAdapterModeNames Interface 24579
IGetAdapterModeNames Members 24579
IGetAdapterModeNames Methods 24579
IGetRecordKeys Interface 24580
IGetRecordKeys Members 24580
IGetRecordKeys Methods 24580
IIdentifyAdapter Interface 24581
ISetSubComponents Interface 24581
ISetSubComponents Members 24581
ISetSubComponents Methods 24582
ISetWebFieldValue Interface 24582
ISetWebFieldValue Members 24583
ISetWebFieldValue Methods 24583
IUpdateValue Interface 24583
IUpdateValue Members 24583
IUpdateValue Methods 24584
IValidateValue Interface 24584
IValidateValue Members 24584
IValidateValue Methods 24585
IWebActionsSubcomponent Interface 24585
IWebFieldsSubcomponent Interface 24585
IWebGetFieldPrevValue Interface 24586
IWebGetFieldPrevValue Members 24586
IWebGetFieldPrevValue Methods 24586
IWebGetFieldPrevValue Properties 24587

Functions **24589**
WebAdapt.AddDefaultActions Function 24589
WebAdapt.AddDefaultFields Function 24589
WebAdapt.FindAdapterAction Function 24590
WebAdapt.FindAdapterField Function 24590
WebAdapt.FindAdapterInParent Function 24590
WebAdapt.MakeAdapterRequestIdentifier Function 24591

WebAdapt.MakeValidAdapterFieldName Function 24591
WebAdapt.RaiseFileUploadNotSupported Function 24591
WebAdapt.RaiseMultipleFilesException Function 24592
WebAdapt.RaiseMultipleValuesException Function 24592

Structs, Records, Enums 24593

WebAdapt.TAdapterHiddenFieldOption Enumeration 24593
WebAdapt.TIteratorMethod Enumeration 24593

Types 24595

WebAdapt.IActionFieldValue Type 24596
WebAdapt.IActionRequest Type 24596
WebAdapt.TAbstractWebRequestFile Type 24596
WebAdapt.TActionExecuteEvent Type 24597
WebAdapt.TActionExecuteHandledEvent Type 24597
WebAdapt.TActionGetEnabledEvent Type 24597
WebAdapt.TActionGetParamsEvent Type 24598
WebAdapt.TActionRequestFieldsEvent Type 24598
WebAdapt.TActionRequestFieldsHandledEvent Type 24598
WebAdapt.TAdapterActionExecuteEvent Type 24599
WebAdapt.TAdapterActionExecuteHandledEvent Type 24599
WebAdapt.TAdapterActionGetParamsEvent Type 24599
WebAdapt.TAdapterActionsClass Type 24600
WebAdapt.TAdapterFieldBooleanValueEvent Type 24600
WebAdapt.TAdapterFieldFileEvent Type 24600
WebAdapt.TAdapterFieldFileListEvent Type 24601
WebAdapt.TAdapterFieldGetBooleanEvent Type 24601
WebAdapt.TAdapterFieldGetMultiValueEvent Type 24601
WebAdapt.TAdapterFieldGetMultiValueStringEvent Type 24602
WebAdapt.TAdapterFieldGetStringEvent Type 24602
WebAdapt.TAdapterFieldGetValueCountEvent Type 24602
WebAdapt.TAdapterFieldGetValueEvent Type 24603
WebAdapt.TAdapterFieldHasValueEvent Type 24603
WebAdapt.TAdapterFieldsClass Type 24603
WebAdapt.TAdapterFieldStringEvent Type 24604
WebAdapt.TAdapterFieldStringsEvent Type 24604
WebAdapt.TAdapterFieldValidateValueEvent Type 24604
WebAdapt.TAdapterFieldValueEvent Type 24605
WebAdapt.TAdapterHiddenFieldOptions Type 24605
WebAdapt.TEndUserGetNameEvent Type 24605

WebAdapt.TEndUserGetUserIDEvent Type 24606
WebAdapt.TEndUserHandledEvent Type 24606
WebAdapt.TEndUserIsLoggedInEvent Type 24606
WebAdapt.TEndUserRightsEvent Type 24607
WebAdapt.TEndUserUserIDEvent Type 24607
WebAdapt.TGetEOFEvent Type 24607
WebAdapt.TGetRecordCountEvent Type 24608
WebAdapt.TGetRecordEvent Type 24608
WebAdapt.TGetRecordIndexEvent Type 24608
WebAdapt.THandledEvent Type 24609
WebAdapt.TImageFieldGetHREFEvent Type 24609
WebAdapt.TImageFieldGetImageEvent Type 24609
WebAdapt.TImageFieldGetParamsEvent Type 24610
WebAdapt.TIterateRecordsEvent Type 24610
WebAdapt.TLoginUserIDEvent Type 24610
WebAdapt.TObjectList Type 24611
WebAdapt.TValidateAdapterFieldEvent Type 24611
WebAdapt.TWebFormGetPageNameEvent Type 24611

Variables **24612**
WebAdapt.NotifyPossibleHTMLChange Variable 24612

Constants **24613**
WebAdapt.sActionRequestLong Constant 24613
WebAdapt.sCurrentPage Constant 24613
WebAdapt.sGotoPageAction Constant 24614
WebAdapt.sImageRequest Constant 24614
WebAdapt.sLoginPassword Constant 24614
WebAdapt.sLoginUserName Constant 24615
WebAdapt.sNextPageAction Constant 24615
WebAdapt.sPrevPageAction Constant 24615

WebBroker Namespace 24617

Classes **24618**
TWebApplication Class 24618
TWebApplication Members 24619
TWebApplication.Create Constructor 24622
TWebApplication.Destroy Destructor 24622
TWebApplication Methods 24622
TWebApplication Properties 24624

Types **24625**
WebBroker.THandleShutdownException Type 24625
WebBroker.TServerExceptionEvent Type 24625

Variables **24626**
WebBroker.Application Variable 24626
WebBroker.HandleShutdownException Variable 24626

WebBrokerSOAP Namespace 24627

Classes **24628**
THTTPSoapDispatcher Class 24628
THTTPSoapDispatcher Members 24629
THTTPSoapDispatcher.Create Constructor 24631
THTTPSoapDispatcher.Destroy Destructor 24632
THTTPSoapDispatcher Methods 24632
THTTPSoapDispatcher Properties 24634

Functions **24636**
WebBrokerSOAP.GetSOAPWebModule Function 24636

WebCntxt Namespace 24637

Classes **24639**
TAbstractAdapterRequest Class 24639
TAbstractAdapterRequest Members 24640
TAbstractAdapterRequest.Create Constructor 24641
TAbstractAdapterRequest Properties 24641
TAbstractAdapterResponse Class 24642
TAbstractAdapterResponse Members 24642
TAbstractAdapterResponse.Create Constructor 24644
TAbstractAdapterResponse Properties 24644
TAbstractWebContext Class 24644
TAbstractWebContext Members 24645
TAbstractWebContext.Create Constructor 24646
TAbstractWebContext.Destroy Destructor 24647
TAbstractWebContext Methods 24647
TAbstractWebContext Properties 24655
TWebContext Class 24660
TWebContext Members 24661
TWebContext.Create Constructor 24662

TWebContext.Destroy Destructor 24663
TWebContext Methods 24663

Interfaces 24672
IWebEndUser Interface 24672
IWebEndUser Members 24673
IWebEndUser Methods 24673
IWebEndUser Properties 24678
IWebPageInfo Interface 24680
IWebPageInfo Members 24681
IWebPageInfo Methods 24681
IWebPageInfo Properties 24683

Functions 24687
WebCntxt.WebContext Function 24687

Structs, Records, Enums 24688
WebCntxt.TFindPageOption Enumeration 24688

Types 24689
WebCntxt.TFindPageOptions Type 24689
WebCntxt.TGetWebContextProc Type 24689
WebCntxt.TGetWebEndUserProc Type 24690
WebCntxt.TSetWebContextProc Type 24690

Variables 24691
WebCntxt.GetModuleFileNameProc Variable 24691
WebCntxt.GetWebContextProc Variable 24691
WebCntxt.GetWebEndUserProc Variable 24692
WebCntxt.SetWebContextProc Variable 24692

WebComp Namespace 24693

Classes 24695
TLayoutAttributes Class 24695
TLayoutAttributes Members 24696
TLayoutAttributes.Create Constructor 24696
TLayoutAttributes Fields 24697
TLayoutWebContent Class 24698
TLayoutWebContent Members 24698
TLayoutWebContent.Create Constructor 24700
TLayoutWebContent Methods 24700

TLayoutWebContent Properties 24705
TWebComponentClassDescription Class 24706
TWebComponentClassDescription Members 24706
TWebComponentClassDescription Fields 24707
TWebComponentList Class 24708
TWebComponentList Members 24709
TWebComponentList.Create Constructor 24711
TWebComponentList.Destroy Destructor 24711
TWebComponentList Methods 24712
TWebComponentList Properties 24717
TWebComponentList Events 24719
TWebComponentsEditorHelper Class 24719
TWebComponentsEditorHelper Members 24720
TWebComponentsEditorHelper Methods 24721
TWebContainedComponent Class 24722
TWebContainedComponent Members 24723
TWebContainedComponent.Create Constructor 24726
TWebContainedComponent.Destroy Destructor 24726
TWebContainedComponent Methods 24726
TWebContainedComponent Properties 24730
TWebContainedComponentHelper Class 24731
TWebContainedComponentHelper Members 24732
TWebContainedComponentHelper.Create Constructor 24733
TWebContainedComponentHelper.Destroy Destructor 24733
TWebContainedComponentHelper Methods 24733
TWebContainedComponentHelper Properties 24737
TWebContainedComponentHelper Events 24738
TWebContainedContainerComponent Class 24738
TWebContainedContainerComponent Members 24739
TWebContainedContainerComponent.Create Constructor 24742
TWebContainedContainerComponent.Destroy Destructor 24743
TWebContainedContainerComponent Methods 24743
TWebContainedContainerComponent Properties 24749
TWebContainerComponent Class 24751
TWebContainerComponent Members 24751
TWebContainerComponent.Create Constructor 24754
TWebContainerComponent.Destroy Destructor 24755
TWebContainerComponent Methods 24755
TWebContainerComponent Properties 24757

TWebContainerComponentHelper Class 24758
TWebContainerComponentHelper Members 24759
TWebContainerComponentHelper.Create Constructor 24760
TWebContainerComponentHelper.Destroy Destructor 24760
TWebContainerComponentHelper Methods 24761
TWebContainerComponentHelper Properties 24764
TWebContainerComponentHelper Events 24765
TWebContentOptions Class 24766
TWebContentOptions Members 24766
TWebContentOptions.Create Constructor 24767
TWebContentOptions Properties 24767

Interfaces 24769
IAddFieldsEditor Interface 24769
IAddFieldsEditor Members 24769
IAddFieldsEditor Methods 24770
IGetWebComponentList Interface 24771
IGetWebComponentList Members 24771
IGetWebComponentList Methods 24771
IGetWebComponentNodeName Interface 24772
IGetWebComponentNodeName Members 24772
IGetWebComponentNodeName Methods 24773
IGetXMLStream Interface 24773
IGetXMLStream Members 24773
IGetXMLStream Methods 24774
ILayoutWebContent Interface 24774
ILayoutWebContent Members 24775
ILayoutWebContent Methods 24775
INotifyActivateModule Interface 24776
INotifyActivateModule Members 24777
INotifyActivateModule Methods 24777
INotifyWebActivate Interface 24777
INotifyWebActivate Members 24778
INotifyWebActivate Methods 24778
IProducerEditorViewSupport Interface 24779
IProducerEditorViewSupport Members 24779
IProducerEditorViewSupport Methods 24779
ISetWebContentOptions Interface 24781
ISetWebContentOptions Members 24781
ISetWebContentOptions Methods 24782

ITopLevelWebComponent Interface 24782
IWebComponent Interface 24782
IWebComponent Members 24783
IWebComponent Methods 24783
IWebComponent Properties 24785
IWebComponentCommands Interface 24786
IWebComponentCommands Members 24787
IWebComponentCommands Methods 24787
IWebComponentContainer Interface 24788
IWebComponentContainer Members 24788
IWebComponentContainer Methods 24789
IWebComponentContainer Properties 24789
IWebComponentEditor Interface 24790
IWebComponentEditor Members 24790
IWebComponentEditor Methods 24791
IWebComponentEditorHelper Interface 24791
IWebComponentEditorHelper Members 24792
IWebComponentEditorHelper Methods 24792
IWebContent Interface 24792
IWebContent Members 24793
IWebContent Methods 24793
IWebVariablesContainerName Interface 24794
IWebVariablesContainerName Members 24794
IWebVariablesContainerName Methods 24794
IWebVariablesContainerName Properties 24795

Functions **24796**
WebComp.DesigningComponent Function 24796
WebComp.EnumRegisteredWebComponents Function 24796
WebComp.FindWebComponentEditorHelper Function 24797
WebComp.QualifyFileName Function 24797
WebComp.RegisterWebComponents Function 24797
WebComp.UnRegisterWebComponents Function 24798
WebComp.ValidateWebParent Function 24798

Structs, Records, Enums **24799**
WebComp.TLabelPosition Enumeration 24799
WebComp.TWebContentFlag Enumeration 24799

Types **24801**
WebComp.TEnumWebComponentsProc Type 24801

WebComp.TGetComponentListEvent Type 24801
WebComp.TLayout Type 24802
WebComp.TSetComponentListEvent Type 24802
WebComp.TWebComponentContainer Type 24802
WebComp.TWebContentFlags Type 24803

Constants 24804
WebComp.EnumRegisteredWebComponentsProc Constant 24804
WebComp.FindWebComponentEditorHelperProc Constant 24804
WebComp.RegisterWebComponentsProc Constant 24805
WebComp.UnRegisterWebComponentsProc Constant 24805

WebContnrs Namespace 24807

Classes 24808
ENamedVariantsError Class 24808
TAbstractNamedVariants Class 24808
TAbstractNamedVariants Members 24809
TAbstractNamedVariants.Destroy Destructor 24810
TAbstractNamedVariants Methods 24811
TAbstractNamedVariants Properties 24821
TNamedVariantsList Class 24824
TNamedVariantsList Members 24825
TNamedVariantsList.Destroy Destructor 24826
TNamedVariantsList Methods 24827

Structs, Records, Enums 24832
WebContnrs.TNamedVariantItem Record 24832

Types 24833
WebContnrs.PNamedVariantItem Type 24833
WebContnrs.PNamedVariantList Type 24833
WebContnrs.TNamedVariantList Type 24833

WebDisp Namespace 24835

Classes 24838
EAdapterDispatcherException Class 24838
EPageDispatcherException Class 24839
ESessionException Class 24840
TAdapterDispatcher Class 24840
TAdapterDispatcher Members 24841

TAdapterDispatcher Events 24844
TAdapterDispatchParams Class 24848
TAdapterDispatchParams Members 24849
TAdapterDispatchParams.Create Constructor 24850
TAdapterDispatchParams.Destroy Destructor 24851
TAdapterDispatchParams Methods 24851
TAdapterDispatchParams Properties 24853
TAdapterItemRequestParams Class 24855
TAdapterItemRequestParams Members 24855
TAdapterItemRequestParams.Create Constructor 24857
TAdapterItemRequestParams.Destroy Destructor 24857
TAdapterItemRequestParams Methods 24857
TAdapterItemRequestParams Properties 24859
TBaseRequestHandler Class 24860
TBaseRequestHandler Members 24861
TBaseRequestHandler Methods 24863
TBaseRequestHandler Properties 24865
TCustomAdapterDispatcher Class 24865
TCustomAdapterDispatcher Members 24866
TCustomAdapterDispatcher Methods 24869
TCustomAdapterDispatcher Events 24876
TCustomLocateFileService Class 24880
TCustomLocateFileService Members 24881
TCustomLocateFileService Methods 24884
TCustomLocateFileService Events 24888
TCustomPageDispatcher Class 24889
TCustomPageDispatcher Members 24890
TCustomPageDispatcher Methods 24894
TCustomPageDispatcher Properties 24900
TCustomPageDispatcher Events 24900
TCustomWebAppComponents Class 24906
TCustomWebAppComponents Members 24907
TCustomWebAppComponents Methods 24911
TCustomWebAppComponents Properties 24918
TCustomWebAppComponents Events 24922
TInvalidWebSession Class 24923
TInvalidWebSession Members 24923
TInvalidWebSession.Create Constructor 24924
TInvalidWebSession Methods 24925

TLocateFileService Class 24928
TLocateFileService Members 24928
TLocateFileService Events 24931
TMultiModuleWebAppServices Class 24933
TMultiModuleWebAppServices Members 24934
TMultiModuleWebAppServices Methods 24937
TMultiModuleWebAppServices Properties 24946
TMultiModuleWebAppServices Events 24947
TPageDispatcher Class 24949
TPageDispatcher Members 24949
TPageDispatcher Properties 24953
TPageDispatcher Events 24953
TWebAppComponents Class 24958
TWebAppComponents Members 24959
TWebAppComponents Properties 24963
TWebAppComponents Events 24966
TWebSession Class 24968
TWebSession Members 24969
Create Constructor 24970
TWebSession Methods 24971

Interfaces 24975
IWebSessionAccess Interface 24975
IWebSessionAccess Members 24975
IWebSessionAccess Methods 24975

Functions 24977
WebDisp.AdapterRequestParamsAsStrings Function 24977
WebDisp.DispatchPageName Function 24977
WebDisp.EncodeParamNameValues Function 24978
WebDisp.EncodeParamsAsHTTPQuery Function 24978
WebDisp.ExpandScriptException Function 24978
WebDisp.ExtractParamNameValues Function 24979
WebDisp.FindDefaultAppPageName Function 24979
WebDisp.PageNameOfRequest Function 24979
WebDisp.RaiseDispatchBlankPageNameException Function 24980
WebDisp.RaisePageDoesNotProvideContent Function 24980
WebDisp.RaisePageNotFound Function 24980
WebDisp.RedirectToPageName Function 24981
WebDisp.RegisterAdapterRequestIdentifier Function 24981

WebDisp.SafeRedirect Function 24981

Structs, Records, Enums **24983**

WebDisp.TPageAccessDenied Enumeration 24983

Types **24984**

WebDisp.TActivateWebModuleEvent Type 24984

WebDisp.TCanViewPageEvent Type 24984

WebDisp.TDispatchActionEvent Type 24985

WebDisp.TDispatchActionHandledEvent Type 24985

WebDisp.TDispatchImageEvent Type 24985

WebDisp.TDispatchImageHandledEvent Type 24986

WebDisp.TDispatchPageEvent Type 24986

WebDisp.TDispatchPageHandledEvent Type 24986

WebDisp.TDispatchPageParamsHandledEvent Type 24987

WebDisp.TGetAdapterRequestParamsEvent Type 24987

WebDisp.TLocateFileServiceFindFileEvent Type 24987

WebDisp.TLocateFileServiceFindStreamEvent Type 24988

WebDisp.TLoginPagedHandledEvent Type 24988

WebDisp.TLoginRequiredEvent Type 24988

WebDisp.TPageAccessDeniedEvent Type 24989

WebDisp.TWebRequest Type 24989

WebDisp.TWebResponse Type 24989

Constants **24991**

WebDisp.sFailurePageParamName Constant 24991

WebDisp.sPageParamName Constant 24991

WebDisp.sRequestIDParamName Constant 24992

WebDisp.sSessionCookieName Constant 24992

WebDisp.sSuccessPageParamName Constant 24992

WebFact Namespace 24993

Classes **24994**

TBaseWebModuleFactory Class 24994

TBaseWebModuleFactory Members 24995

TBaseWebModuleFactory.Create Constructor 24995

TBaseWebModuleFactory Methods 24996

TBaseWebPageInfo Class 24999

TBaseWebPageInfo Members 25000

TBaseWebPageInfo.Create Constructor 25000

TBaseWebPageInfo Methods 25001
TBaseWebPageModuleFactory Class 25004
TBaseWebPageModuleFactory Members 25005
TBaseWebPageModuleFactory.Create Constructor 25006
TBaseWebPageModuleFactory Methods 25006
TWebAppDataModuleFactory Class 25009
TWebAppDataModuleFactory Members 25010
TWebAppDataModuleFactory.Create Constructor 25011
TWebAppDataModuleFactory Methods 25011
TWebAppPageModuleFactory Class 25012
TWebAppPageModuleFactory Members 25013
TWebAppPageModuleFactory.Create Constructor 25013
TWebAppPageModuleFactory Methods 25014
TWebDataModuleFactory Class 25014
TWebDataModuleFactory Members 25015
TWebDataModuleFactory.Create Constructor 25016
TWebPageInfo Class 25016
TWebPageInfo Members 25017
TWebPageInfo.Create Constructor 25018
TWebPageModuleFactory Class 25019
TWebPageModuleFactory Members 25020
TWebPageModuleFactory.Create Constructor 25020

Structs, Records, Enums 25022
WebFact.TWebPageAccessFlags Enumeration 25022

Types 25023
WebFact.TWebPageAccess Type 25023

WebForm Namespace 25025

Classes 25028
TAbstractControlRequirements Class 25029
TAbstractControlRequirements Members 25029
TAbstractControlRequirements Methods 25030
TAbstractFormRequirements Class 25032
TAbstractFormRequirements Members 25032
TAbstractFormRequirements Methods 25033
TAdapterActionButton Class 25034
TAdapterActionButton Members 25035
TAdapterActionButton Properties 25039

TAdapterActionsListHelper Class 25046
TAdapterActionsListHelper Members 25046
TAdapterActionsListHelper.Create Constructor 25047
TAdapterActionsListHelper Methods 25048
TAdapterActionsListHelper Properties 25052
TAdapterCommandColumn Class 25052
TAdapterCommandColumn Members 25054
TAdapterCommandColumn Properties 25058
TAdapterCommandGroup Class 25062
TAdapterCommandGroup Members 25064
TAdapterCommandGroup Properties 25068
TAdapterControlGroup Class 25070
TAdapterControlGroup Members 25072
TAdapterControlGroup.Create Constructor 25076
TAdapterControlGroup.Destroy Destructor 25076
TAdapterControlGroup Methods 25076
TAdapterControlGroup Properties 25084
TAdapterDataDisplay Class 25086
TAdapterDataDisplay Members 25087
TAdapterDataDisplay.Create Constructor 25092
TAdapterDataDisplay Methods 25092
TAdapterDataDisplay Properties 25097
TAdapterDisplayColumn Class 25099
TAdapterDisplayColumn Members 25100
TAdapterDisplayColumn Properties 25104
TAdapterDisplayField Class 25109
TAdapterDisplayField Members 25110
TAdapterDisplayField Properties 25115
TAdapterDisplayReferenceGroup Class 25122
TAdapterDisplayReferenceGroup Members 25123
TAdapterDisplayReferenceGroup.Create Constructor 25127
TAdapterDisplayReferenceGroup.Destroy Destructor 25128
TAdapterDisplayReferenceGroup Methods 25128
TAdapterDisplayReferenceGroup Properties 25131
TAdapterEditColumn Class 25132
TAdapterEditColumn Members 25133
TAdapterEditColumn Properties 25138
TAdapterErrorList Class 25145
TAdapterErrorList Members 25146

TAdapterErrorList Properties 25149
TAdapterFieldGroup Class 25150
TAdapterFieldGroup Members 25151
TAdapterFieldGroup Properties 25156
TAdapterForm Class 25159
TAdapterForm Members 25160
TAdapterForm Properties 25164
TAdapterFormLayout Class 25166
TAdapterFormLayout Members 25166
TAdapterFormLayout.Create Constructor 25168
TAdapterFormLayout Methods 25168
TAdapterFormLayout Properties 25174
TAdapterGrid Class 25175
TAdapterGrid Members 25177
TAdapterGrid Properties 25182
TControlRequirements Class 25185
TControlRequirements Members 25186
TControlRequirements.Create Constructor 25186
TControlRequirements.Destroy Destructor 25186
TControlRequirements Methods 25187
TControlRequirements Properties 25189
TCustomAdapterActionButton Class 25189
TCustomAdapterActionButton Members 25190
TCustomAdapterActionButton.Create Constructor 25194
TCustomAdapterActionButton.Destroy Destructor 25195
TCustomAdapterActionButton Methods 25195
TCustomAdapterActionButton Properties 25200
TCustomAdapterCommandColumn Class 25205
TCustomAdapterCommandColumn Members 25206
TCustomAdapterCommandColumn.Create Constructor 25210
TCustomAdapterCommandColumn.Destroy Destructor 25211
TCustomAdapterCommandColumn Methods 25211
TCustomAdapterCommandColumn Properties 25215
TCustomAdapterCommandGroup Class 25220
TCustomAdapterCommandGroup Members 25221
TCustomAdapterCommandGroup.Create Constructor 25225
TCustomAdapterCommandGroup.Destroy Destructor 25225
TCustomAdapterCommandGroup Methods 25226
TCustomAdapterCommandGroup Properties 25228

TCustomAdapterDisplayColumn Class 25230
TCustomAdapterDisplayColumn Members 25231
TCustomAdapterDisplayColumn.Create Constructor 25236
TCustomAdapterDisplayColumn Methods 25236
TCustomAdapterDisplayColumn Properties 25239
TCustomAdapterDisplayField Class 25240
TCustomAdapterDisplayField Members 25241
TCustomAdapterDisplayField.Create Constructor 25246
TCustomAdapterDisplayField Methods 25246
TCustomAdapterDisplayField Properties 25248
TCustomAdapterEditColumn Class 25252
TCustomAdapterEditColumn Members 25253
TCustomAdapterEditColumn Methods 25258
TCustomAdapterErrorList Class 25259
TCustomAdapterErrorList Members 25259
TCustomAdapterErrorList.Create Constructor 25262
TCustomAdapterErrorList.Destroy Destructor 25263
TCustomAdapterErrorList Methods 25263
TCustomAdapterErrorList Properties 25265
TCustomAdapterFieldGroup Class 25265
TCustomAdapterFieldGroup Members 25267
TCustomAdapterFieldGroup.Create Constructor 25271
TCustomAdapterFieldGroup.Destroy Destructor 25272
TCustomAdapterFieldGroup Methods 25272
TCustomAdapterFieldGroup Properties 25275
TCustomAdapterForm Class 25276
TCustomAdapterForm Members 25277
TCustomAdapterForm.Create Constructor 25281
TCustomAdapterForm.Destroy Destructor 25281
TCustomAdapterForm Methods 25282
TCustomAdapterForm Properties 25284
TCustomAdapterGrid Class 25286
TCustomAdapterGrid Members 25288
TCustomAdapterGrid.Create Constructor 25292
TCustomAdapterGrid.Destroy Destructor 25293
TCustomAdapterGrid Methods 25293
TCustomAdapterGrid Properties 25296
TDesigntimeWarnings Class 25297
TDesigntimeWarnings Members 25297

TDesigntimeWarnings.Create Constructor 25298
TDesigntimeWarnings.Destroy Destructor 25298
TDesigntimeWarnings Methods 25298
TDesigntimeWarnings Properties 25299
TGeneratedFunctions Class 25300
TGeneratedFunctions Members 25300
TGeneratedFunctions.Create Constructor 25300
TGeneratedFunctions.Destroy Destructor 25301
TGeneratedFunctions Methods 25301
TGeneratedFunctions Properties 25301

Interfaces 25303
IAdapterDisplay Interface 25303
IGetAdapterFormLayout Interface 25303
IGetAdapterFormLayout Members 25304
IGetAdapterFormLayout Methods 25304
IGetAdapterModeOfDisplay Interface 25304
IGetAdapterModeOfDisplay Members 25305
IGetAdapterModeOfDisplay Methods 25305
IGetAdapterOfDisplay Interface 25305
IGetAdapterOfDisplay Members 25306
IGetAdapterOfDisplay Methods 25306
IGetAdapterVariableName Interface 25306
IGetAdapterVariableName Members 25307
IGetAdapterVariableName Methods 25307
IGetAdapterVariableName Properties 25307
IGetControlRequirements Interface 25308
IGetControlRequirements Members 25308
IGetControlRequirements Methods 25308
IGetFormRequirements Interface 25309
IGetFormRequirements Members 25309
IGetFormRequirements Methods 25309
IGetVariablesContainerOfDisplay Interface 25310
IGetVariablesContainerOfDisplay Members 25310
IGetVariablesContainerOfDisplay Methods 25310

Functions 25312
WebForm.CalcDisplayFieldHTMLElementType Function 25312
WebForm.CalcDisplayFieldViewMode Function 25312
WebForm.FindVariablesContainer Function 25313

WebForm.FindVariablesContainerInParent Function 25313
WebForm.GetAdapterModeOfAdapterInParent Function 25313
WebForm.MakeAdapterHTMLName Function 25314
WebForm.MakeAdapterVariableName Function 25314

Structs, Records, Enums 25315
WebForm.TActionButtonHideOption Enumeration 25315
WebForm.TColumnHTMLElementType Enumeration 25316
WebForm.TCommandHTMLElementType Enumeration 25316
WebForm.TDisplayFieldHideOption Enumeration 25317
WebForm.TDisplayFieldHTMLElementType Enumeration 25318
WebForm.TDisplayFieldViewMode Enumeration 25318
WebForm.TInputFieldHTMLElementType Enumeration 25319
WebForm.TLayoutState Enumeration 25320

Types 25321
WebForm.TActionButtonHideOptions Type 25321
WebForm.TAdapterDataDisplayClass Type 25321
WebForm.TDisplayFieldHideOptions Type 25321

Constants 25323
WebForm.sDisplayText Constant 25323
WebForm.sFieldText Constant 25323
WebForm.sFormGet Constant 25324
WebForm.sFormPost Constant 25324
WebForm.sMultipartForm Constant 25324

WebModu Namespace 25325

Classes 25327
EWebModuleException Class 25327
TAdapterActionName Class 25328
TAdapterActionName Members 25328
TAdapterActionName.Create Constructor 25329
TAdapterActionName Methods 25330
TAdapterActionName Properties 25331
TCustomWebAppDataModule Class 25332
TCustomWebAppDataModule Members 25333
TCustomWebAppDataModule.Create Constructor 25336
TCustomWebAppDataModule.CreateNew Constructor 25337
TCustomWebAppDataModule.Destroy Destructor 25337

TCustomWebAppDataModule Methods 25337
TCustomWebAppDataModule Properties 25339
TCustomWebAppPageModule Class 25339
TCustomWebAppPageModule Members 25340
TCustomWebAppPageModule.Create Constructor 25344
TCustomWebAppPageModule.CreateNew Constructor 25345
TCustomWebAppPageModule.Destroy Destructor 25345
TCustomWebAppPageModule Methods 25345
TCustomWebAppPageModule Properties 25347
TCustomWebDataModule Class 25348
TCustomWebDataModule Members 25349
TCustomWebDataModule.Create Constructor 25352
TCustomWebDataModule.CreateNew Constructor 25352
TCustomWebDataModule.Destroy Destructor 25353
TCustomWebDataModule Methods 25353
TCustomWebDataModule Properties 25354
TCustomWebDataModule Events 25356
TCustomWebPageModule Class 25357
TCustomWebPageModule Members 25358
TCustomWebPageModule.Create Constructor 25362
TCustomWebPageModule.CreateNew Constructor 25362
TCustomWebPageModule.Destroy Destructor 25363
TCustomWebPageModule Methods 25363
TCustomWebPageModule Properties 25364
TCustomWebPageModule Events 25366
TSiteModuleHelper Class 25368
TSiteModuleHelper Members 25369
TSiteModuleHelper.Create Constructor 25370
TSiteModuleHelper.Destroy Destructor 25370
TSiteModuleHelper Methods 25371
TSiteModuleHelper Properties 25374
TSiteModuleHelper Events 25376
TSitePageModuleHelper Class 25377
TSitePageModuleHelper Members 25377
TSitePageModuleHelper.Create Constructor 25379
TSitePageModuleHelper.Destroy Destructor 25380
TSitePageModuleHelper Methods 25380
TSitePageModuleHelper Properties 25385
TSitePageModuleHelper Events 25386

TWebAppDataModule Class 25388
TWebAppDataModule Members 25388
TWebAppDataModule Properties 25392
TWebAppDataModule Events 25393
TWebAppPageModule Class 25394
TWebAppPageModule Members 25395
TWebAppPageModule Properties 25399
TWebAppPageModule Events 25400
TWebAppPageModuleHelper Class 25403
TWebAppPageModuleHelper Members 25403
TWebAppPageModuleHelper Methods 25405
TWebDataModule Class 25405
TWebDataModule Members 25406
TWebDataModule Events 25409
TWebNotifyContainer Class 25410
TWebNotifyContainer Members 25411
TWebNotifyContainer.Create Constructor 25411
TWebNotifyContainer.Destroy Destructor 25412
TWebNotifyContainer Fields 25412
TWebNotifyContainer Methods 25413
TWebPageModule Class 25414
TWebPageModule Members 25415
TWebPageModule Properties 25419
TWebPageModule Events 25420
TWebPageModuleHelper Class 25422
TWebPageModuleHelper Members 25423
TWebPageModuleHelper Methods 25425

Functions **25426**
WebModu.FindNamedModuleVariable Function 25426
WebModu.GetModuleVariable Function 25426
WebModu.GetModuleVariableCount Function 25426

Types **25428**
WebModu.TGetParamsEvent Type 25428

WebNode Namespace 25429

Interfaces **25430**
IWebNode Interface 25430
IWebNode Members 25431

IWebNode Methods 25431
IWebNode Properties 25433

WebReq Namespace 25435

Classes 25436
TWebModuleFactoryList Class 25436
TWebModuleFactoryList Members 25436
TWebModuleFactoryList.Create Constructor 25437
TWebModuleFactoryList.Destroy Destructor 25438
TWebModuleFactoryList Methods 25438
TWebModuleFactoryList Properties 25439
TWebModuleList Class 25440
TWebModuleList Members 25441
TWebModuleList.Create Constructor 25442
TWebModuleList.Destroy Destructor 25442
TWebModuleList Methods 25443
TWebModuleList Properties 25448
TWebRequestHandler Class 25448
TWebRequestHandler Members 25449
TWebRequestHandler.Create Constructor 25452
TWebRequestHandler.Destroy Destructor 25452
TWebRequestHandler Methods 25452
TWebRequestHandler Properties 25455

Functions 25459
WebReq.WebRequestHandler Function 25459

Structs, Records, Enums 25460
WebReq.TRequestNotification Enumeration 25460

Variables 25461
WebReq.WebRequestHandlerProc Variable 25461

WebScript Namespace 25463

Classes 25465
ERecursiveIncludeFile Class 25465
EScriptEvalError Class 25466
TAbstractScriptEngine Class 25466
TAbstractScriptEngine Members 25466
TAbstractScriptEngine.Create Constructor 25467

TAbstractScriptEngine.Destroy Destructor 25467
TAbstractScriptEngine Methods 25468
TAbstractScriptEngine Properties 25469
TAbstractScriptObjectFactories Class 25470
TAbstractScriptObjectFactories Members 25470
TAbstractScriptObjectFactories Methods 25471
TAbstractScriptObjectFactories Properties 25472
TAbstractScriptObjectFactory Class 25473
TAbstractScriptObjectFactory Members 25473
TAbstractScriptObjectFactory Methods 25474
TCurrentWebPageInfo Class 25475
TCurrentWebPageInfo Members 25476
TCurrentWebPageInfo.Create Constructor 25477
TCurrentWebPageInfo Methods 25477
THTMLItem Class 25480
THTMLItem Members 25480
THTMLItem.Create Constructor 25481
THTMLItem Methods 25481
THTMLItem Properties 25482
THTMLItems Class 25483
THTMLItems Members 25484
THTMLItems.Create Constructor 25485
THTMLItems.Destroy Destructor 25485
THTMLItems Methods 25486
THTMLItems Properties 25487
TLineNumberMap Class 25488
TLineNumberMap Members 25488
TLineNumberMap Methods 25489
TLineNumberMap Properties 25490
TScriptCodeItem Class 25490
TScriptCodeItem Members 25491
TScriptCodeItem Methods 25491
TScriptEnginesList Class 25492
TScriptEnginesList Members 25493
TScriptEnginesList.Create Constructor 25493
TScriptEnginesList.Destroy Destructor 25494
TScriptEnginesList Methods 25494
TScriptEnginesList Properties 25496
TScriptError Class 25497

TScriptError Members 25497
TScriptError.Create Constructor 25498
TScriptError Fields 25498
TScriptError Methods 25500
TScriptErrors Class 25501
TScriptErrors Members 25502
TScriptErrors.Create Constructor 25503
TScriptErrors.Destroy Destructor 25503
TScriptErrors Methods 25504
TScriptFile Class 25505
TScriptFile Members 25505
TScriptFile.Create Constructor 25505
TScriptFile.Destroy Destructor 25506
TScriptFile Fields 25506
TScriptFile Methods 25506
TScriptFile Properties 25507
TScriptGlobalObjects Class 25508
TScriptGlobalObjects Members 25509
TScriptGlobalObjects.Create Constructor 25510
TScriptGlobalObjects.Destroy Destructor 25510
TScriptGlobalObjects Methods 25510
TScriptGlobalObjects Properties 25512
TScriptIncludeFile Class 25513
TScriptIncludeFile Members 25513
TScriptIncludeFile.Create Constructor 25514
TScriptIncludeFile Methods 25514
TScriptIncludeFiles Class 25514
TScriptIncludeFiles Members 25515
TScriptIncludeFiles.Create Constructor 25516
TScriptIncludeFiles Methods 25517
TScriptIncludeFiles Properties 25517
TScriptIncludeItem Class 25518
TScriptIncludeItem Members 25518
TScriptIncludeItem.Create Constructor 25519
TScriptIncludeItem Methods 25520
TScriptItem Class 25520
TScriptItem Members 25520
TScriptItem.Create Constructor 25521
TScriptItem Methods 25521

TScriptItem Properties 25522
TScriptItems Class 25522
TScriptItems Members 25523
TScriptItems.Create Constructor 25524
TScriptItems Methods 25524
TScriptItems Properties 25525
TScriptObjectFactories Class 25526
TScriptObjectFactories Members 25526
TScriptObjectFactories.Create Constructor 25527
TScriptObjectFactories.Destroy Destructor 25528
TScriptObjectFactories Methods 25528
TScriptObjectFactories Properties 25529
TScriptProducer Class 25530
TScriptProducer Members 25531
TScriptProducer.Create Constructor 25532
TScriptProducer.Destroy Destructor 25533
TScriptProducer Fields 25533
TScriptProducer Methods 25534
TScriptProducer Properties 25539
TVariableLookup Class 25540
TVariableLookup Members 25541
TVariableLookup.Create Constructor 25541
TVariableLookup Methods 25542
TVariableLookup Properties 25542

Structs, Records, Enums 25544
WebScript.TGlobalScriptVariables Enumeration 25544
WebScript.TIncludeType Enumeration 25545
WebScript.TLineNumbers Record 25545

Types 25546
WebScript.CreateComponentStubFactory Type 25546
WebScript.PLineNumbers Type 25546

Constants 25547
WebScript.GlobalScriptVariableNames Constant 25547

WebServExp Namespace 25549

Classes 25551
TWebServExp Class 25551

TWebServExp Members 25551
TWebServExp.Create Constructor 25553
TWebServExp.Destroy Destructor 25553
TWebServExp Methods 25554
TWebServExp Properties 25555
TWebServExp Events 25557

Interfaces 25559
IWebServExp Interface 25559
IWebServExp Members 25559
IWebServExp Methods 25560
IWebServExpAccess Interface 25561
IWebServExpAccess Members 25561
IWebServExpAccess Methods 25561

Functions 25562
WebServExp.GetAliasBaseTypeInfo Function 25562
WebServExp.GetBindingType Function 25562
WebServExp.IsBaseClassTypeInfo Function 25563

Structs, Records, Enums 25564
WebServExp.ArgumentType Enumeration 25564
WebServExp.MessageType Enumeration 25564
WebServExp.TSchemaType Record 25565

Types 25566
WebServExp.TAfterPublishingWSDLEvent Type 25566
WebServExp.TBeforePublishingTypesEvent Type 25566
WebServExp.TPublishingTypeEvent Type 25567
WebServExp.TSchemaTypeArray Type 25567

WebSess Namespace 25569

Classes 25570
TCustomSessionsService Class 25570
TCustomSessionsService Members 25571
TCustomSessionsService.Create Constructor 25574
TCustomSessionsService Methods 25574
TCustomSessionsService Properties 25580
TCustomSessionsService Events 25581
TSessionsService Class 25582
TSessionsService Members 25583

TSessionsService Properties 25586
TSessionsService Events 25587
TWebSessionIDsImpl Class 25588
TWebSessionIDsImpl Members 25588
TWebSessionIDsImpl.Create Constructor 25590
TWebSessionIDsImpl.Destroy Destructor 25590
TWebSessionIDsImpl Methods 25591
TWebSessionImpl Class 25592
TWebSessionImpl Members 25592
TWebSessionImpl.Create Constructor 25594
TWebSessionImpl.Destroy Destructor 25594
TWebSessionImpl Methods 25595
TWebSessionImpl Properties 25598

Structs, Records, Enums 25601
WebSess.TEndSessionReason Enumeration 25601

Types 25602
WebSess.TEndSessionEvent Type 25602
WebSess.TStartSessionEvent Type 25602

WebUsers Namespace 25603

Classes 25604
EUserIDNotFoundException Class 25604
EValidateUserException Class 25604
EWebUsersListException Class 25605
TCustomWebUserList Class 25606
TCustomWebUserList Members 25606
TCustomWebUserList.Create Constructor 25609
TCustomWebUserList.Destroy Destructor 25610
TCustomWebUserList Methods 25610
TCustomWebUserList Properties 25611
TCustomWebUserList Events 25612
TWebUserItem Class 25615
TWebUserItem Members 25615
TWebUserItem.Create Constructor 25616
TWebUserItem.Destroy Destructor 25617
TWebUserItem Methods 25617
TWebUserItem Properties 25618
TWebUserItems Class 25620

TWebUserItems Members 25620
TWebUserItems.Create Constructor 25622
TWebUserItems Methods 25622
TWebUserItems Properties 25625
TWebUserList Class 25625
TWebUserList Members 25626
TWebUserList Properties 25629
TWebUserList Events 25630

Structs, Records, Enums 25633
WebUsers.TValidateUserError Enumeration 25633

Types 25634
WebUsers.TCheckAccessRightsEvent Type 25634
WebUsers.TCheckAccessRightsHandledEvent Type 25634
WebUsers.TValidateUserErrorEvent Type 25635
WebUsers.TValidateUserEvent Type 25635
WebUsers.TValidateUserHandledEvent Type 25635

WSDLBind Namespace 25637

Classes 25640
TBinding Class 25640
TBinding Members 25641
TBinding Methods 25645
TBindingOperation Class 25647
TBindingOperation Members 25648
TBindingOperation Methods 25652
TBindingOperations Class 25654
TBindingOperations Members 25655
TBindingOperations Methods 25660
TBindings Class 25661
TBindings Members 25661
TBindings Methods 25666
TDefinition Class 25667
TDefinition Members 25667
TDefinition Methods 25672
TDocumentation Class 25676
TDocumentationCollection Class 25676
TDocumentationCollection Members 25676
TDocumentationCollection Methods 25682

TDocumented Class 25683
TDocumented Members 25683
TDocumented Methods 25687
TFault Class 25689
TFaults Class 25689
TFaults Members 25690
TFaults Methods 25695
TImport Class 25696
TImport Members 25696
TImport Methods 25701
TImports Class 25702
TImports Members 25702
TImports Methods 25707
TMessage Class 25708
TMessage Members 25709
TMessage Methods 25713
TMessages Class 25715
TMessages Members 25715
TMessages Methods 25720
TOperation Class 25721
TOperation Members 25722
TOperation Methods 25726
TOperations Class 25728
TOperations Members 25729
TOperations Methods 25734
TParam Class 25735
TParam Members 25735
TParam Methods 25740
TPart Class 25741
TPart Members 25742
TPart Methods 25746
TParts Class 25748
TParts Members 25748
TParts Methods 25754
TPort Class 25755
TPort Members 25755
TPort Methods 25760
TPorts Class 25761
TPorts Members 25761

TPorts Methods 25766
TPortType Class 25767
TPortType Members 25768
TPortType Methods 25772
TPortTypes Class 25774
TPortTypes Members 25774
TPortTypes Methods 25779
TService Class 25780
TService Members 25781
TService Methods 25785
TServices Class 25787
TServices Members 25787
TServices Methods 25792
TTypes Class 25793
TTypes Members 25793
TTypes Methods 25798
TWSDLDocument Class 25799
TWSDLDocument Members 25800
TWSDLDocument Methods 25806
TWSDLDocument Properties 25808
TXMLSchemaDefs Class 25808
TXMLSchemaDefs Members 25809
TXMLSchemaDefs Methods 25814

Interfaces 25815
IBinding Interface 25815
IBinding Members 25816
IBinding Methods 25817
IBinding Properties 25818
IBindingInput Interface 25819
IBindingOperation Interface 25820
IBindingOperation Members 25820
IBindingOperation Methods 25822
IBindingOperation Properties 25823
IBindingOperations Interface 25824
IBindingOperations Members 25824
IBindingOperations Methods 25826
IBindingOperations Properties 25827
IBindingOutput Interface 25827
IBindings Interface 25828

IBindings Members 25828
IBindings Methods 25829
IBindings Properties 25830
IDefinition Interface 25831
IDefinition Members 25831
IDefinition Methods 25833
IDefinition Properties 25835
IDocumentation Interface 25837
IDocumentationCollection Interface 25838
IDocumentationCollection Members 25838
IDocumentationCollection Methods 25840
IDocumentationCollection Properties 25840
IDocumented Interface 25841
IDocumented Members 25841
IDocumented Methods 25842
IDocumented Properties 25843
IFault Interface 25843
IFaults Interface 25844
IFaults Members 25844
IFaults Methods 25846
IFaults Properties 25846
IImport Interface 25847
IImport Members 25847
IImport Methods 25849
IImport Properties 25850
IImports Interface 25850
IImports Members 25851
IImports Methods 25852
IImports Properties 25853
IMessage Interface 25853
IMessage Members 25854
IMessage Methods 25855
IMessage Properties 25856
IMessages Interface 25857
IMessages Members 25857
IMessages Methods 25859
IMessages Properties 25859
IOperation Interface 25860
IOperation Members 25860

IOperation Methods 25862
IOperation Properties 25863
IOperations Interface 25864
IOperations Members 25864
IOperations Methods 25866
IOperations Properties 25867
IParam Interface 25868
IParam Members 25868
IParam Methods 25869
IParam Properties 25871
IPart Interface 25871
IPart Members 25872
IPart Methods 25873
IPart Properties 25875
IParts Interface 25876
IParts Members 25876
IParts Methods 25877
IParts Properties 25878
IPort Interface 25879
IPort Members 25879
IPort Methods 25881
IPort Properties 25882
IPorts Interface 25882
IPorts Members 25883
IPorts Methods 25884
IPorts Properties 25885
IPortType Interface 25885
IPortType Members 25886
IPortType Methods 25887
IPortType Properties 25888
IPortTypes Interface 25889
IPortTypes Members 25889
IPortTypes Methods 25891
IPortTypes Properties 25891
IService Interface 25892
IService Members 25892
IService Methods 25894
IService Properties 25894
IServices Interface 25895

IServices Members 25895
IServices Methods 25897
IServices Properties 25898
ITypes Interface 25898
ITypes Members 25899
ITypes Methods 25900
ITypes Properties 25900
IWSDLDocument Interface 25901
IWSDLDocument Members 25901
IWSDLDocument Methods 25903
IWSDLDocument Properties 25904
IXMLSchemaDefs Interface 25905
IXMLSchemaDefs Members 25905
IXMLSchemaDefs Methods 25906
IXMLSchemaDefs Properties 25907

Functions 25908
WSDLBind.GetDefinition Function 25908
WSDLBind.LoadWSDLDoc Function 25908
WSDLBind.NewWSDLDoc Function 25908

Constants 25910
WSDLBind.SBinding Constant 25910
WSDLBind.SDefinitions Constant 25910
WSDLBind.SFault Constant 25911
WSDLBind.SImport Constant 25911
WSDLBind.SInput Constant 25911
WSDLBind.SLocation Constant 25912
WSDLBind.SMessage Constant 25912
WSDLBind.SNamespace Constant 25912
WSDLBind.SOperation Constant 25913
WSDLBind.SOutput Constant 25913
WSDLBind.SPart Constant 25913
WSDLBind.SPort Constant 25914
WSDLBind.SPortType Constant 25914
WSDLBind.SService Constant 25914
WSDLBind.SType Constant 25915
WSDLBind.STypes Constant 25915

WSDLIntf Namespace 25917

Classes **25919**
TWideStrings Class 25919
TWideStrings Members 25920
TWideStrings.Create Constructor 25920
TWideStrings.Destroy Destructor 25920
TWideStrings Methods 25921
TWideStrings Properties 25923

Structs, Records, Enums **25925**
WSDLIntf.TWebServiceBindingType Enumeration 25925
WSDLIntf.TWSDLElement Enumeration 25925
WSDLIntf.TWString Record 25926

Types **25927**
WSDLIntf.TWSDLElements Type 25927

Constants **25928**
WSDLIntf.Directives Constant 25929
WSDLIntf.DynArrayTypes Constant 25929
WSDLIntf.Httpns Constant 25930
WSDLIntf.Operators Constant 25930
WSDLIntf.OtherReservedWords Constant 25930
WSDLIntf.ReservedWords Constant 25931
WSDLIntf.SAddress Constant 25931
WSDLIntf.SAnySimpleType Constant 25932
WSDLIntf.SAnyType Constant 25932
WSDLIntf.SArray Constant 25932
WSDLIntf.SArrayOf Constant 25933
WSDLIntf.SArrayType Constant 25933
WSDLIntf.SBinding Constant 25933
WSDLIntf.SBody Constant 25934
WSDLIntf.ScalarArrayTypes Constant 25934
WSDLIntf.SComplexType Constant 25934
WSDLIntf.SDefinitions Constant 25935
WSDLIntf.SDynArray Constant 25935
WSDLIntf.SElement Constant 25935
WSDLIntf.SEncodingStyle Constant 25936
WSDLIntf.SFault Constant 25936
WSDLIntf.SHeader Constant 25936
WSDLIntf.SImport Constant 25937

WSDLIntf.SInOut Constant 25937
WSDLIntf.SInput Constant 25937
WSDLIntf.SLocation Constant 25938
WSDLIntf.SMessage Constant 25938
WSDLIntf.SName Constant 25938
WSDLIntf.SNameSpace Constant 25939
WSDLIntf.SNsPrefix Constant 25939
WSDLIntf.Soapns Constant 25939
WSDLIntf.SOperation Constant 25940
WSDLIntf.SOutput Constant 25940
WSDLIntf.SPart Constant 25940
WSDLIntf.SParts Constant 25941
WSDLIntf.SPort Constant 25941
WSDLIntf.SPortType Constant 25941
WSDLIntf.SRequest Constant 25942
WSDLIntf.SResponse Constant 25942
WSDLIntf.SReturn Constant 25942
WSDLIntf.SSchema Constant 25943
WSDLIntf.SService Constant 25943
WSDLIntf.SSoapAction Constant 25943
WSDLIntf.SStyle Constant 25944
WSDLIntf.STns Constant 25944
WSDLIntf.STransport Constant 25944
WSDLIntf.SType Constant 25945
WSDLIntf.STypes Constant 25945
WSDLIntf.SUnknown Constant 25945
WSDLIntf.SUse Constant 25946
WSDLIntf.SVerb Constant 25946
WSDLIntf.tns Constant 25946
WSDLIntf.Wsdlns Constant 25947
WSDLIntf.xsdns Constant 25947

WSDLItems Namespace 25949

Classes **25951**
TWSDLItems Class 25951
TWSDLItems Members 25952
Create Constructor 25959
TWSDLItems Methods 25960

TWSDLItems Properties 25987

Interfaces **25989**

IHeaderInfo Interface 25989

IHeaderInfo Members 25989

IHeaderInfo Methods 25990

IHeaderInfo Properties 25993

IQualifiedName Interface 25994

IQualifiedName Members 25995

IQualifiedName Methods 25995

IQualifiedName Properties 25997

IWSDLItems Interface 25998

IWSDLItems Members 25998

IWSDLItems Methods 25998

Functions **26000**

WSDLItems.HasDefinition Function 26000

WSDLItems.HasDefinition Function 26000

WSDLItems.NewQualifiedName Function 26000

Structs, Records, Enums **26002**

WSDLItems.IterateImportOptions Enumeration 26002

Types **26003**

WSDLItems.IHeaderInfoArray Type 26003

WSDLItems.IQualifiedNameArray Type 26003

WSDLItems.TWSDLIterateProc Type 26003

WSDLNode Namespace 26005

Classes **26006**

EWSDLLoadException Class 26006

TWSDLClientNode Class 26006

TWSDLClientNode Members 26007

TWSDLClientNode Methods 26009

TWSDLClientNode Properties 26013

TWSDLView Class 26013

TWSDLView Members 26014

TWSDLView Fields 26017

TWSDLView Methods 26017

TWSDLView Properties 26018

Functions **26022**
WSDLNode.ActivateWSDL Function 26022

WSDLPub Namespace 26023

Classes **26024**
TWSDLHTMLPublish Class 26024
TWSDLHTMLPublish Members 26025
TWSDLHTMLPublish.Create Constructor 26028
TWSDLHTMLPublish.Destroy Destructor 26028
TWSDLHTMLPublish Methods 26029
TWSDLHTMLPublish Properties 26034
TWSDLHTMLPublish Events 26035
TWSDLPublish Class 26037
TWSDLPublish Members 26037
TWSDLPublish Methods 26039
TWSDLPublish Properties 26040
TWSDLPublish Events 26040

Interfaces **26042**
IWSDLPublish Interface 26042
IWSDLPublish Members 26042
IWSDLPublish Methods 26042

Functions **26044**
WSDLPub.WSDLPubFactory Function 26044

Structs, Records, Enums **26045**
WSDLPub.SOAPPublishOptions Enumeration 26045

Types **26046**
WSDLPub.TBeforePublishingWSDLEvent Type 26046
WSDLPub.TSOAPPublishOptions Type 26046

XMLBrokr Namespace 26047

Classes **26049**
TXMLBroker Class 26049
TXMLBroker Members 26050
TXMLBroker.Create Constructor 26054
TXMLBroker.Destroy Destructor 26054
TXMLBroker Methods 26054

TXMLBroker Properties 26069
TXMLBroker Events 26077
TXMLParams Class 26081
TXMLParams Members 26082
TXMLParams Methods 26084

Interfaces 26086
INotifyConnectionChange Interface 26086
INotifyConnectionChange Members 26086
INotifyConnectionChange Methods 26086

Functions 26088
XMLBrokr.FormatXML Function 26088

Structs, Records, Enums 26089
XMLBrokr.TXMLOption Enumeration 26089

Types 26090
XMLBrokr.TGetErrorResponseEvent Type 26090
XMLBrokr.TGetResponseEvent Type 26090
XMLBrokr.TRequestRecordsEvent Type 26091
XMLBrokr.TRequestUpdateEvent Type 26091
XMLBrokr.TXMLOptions Type 26091

Constants 26092
XMLBrokr.sPostDelta Constant 26092
XMLBrokr.sProducer Constant 26092
XMLBrokr.sRedirect Constant 26092

XMLDoc Namespace 26095

Classes 26097
TXMLDocument Class 26097
TXMLDocument Members 26098
Create Constructor 26103
TXMLDocument.Destroy Destructor 26104
TXMLDocument Methods 26104
TXMLDocument Properties 26132
TXMLDocument Events 26144
TXMLNode Class 26147
TXMLNode Members 26148
TXMLNode.Create Constructor 26152

TXMLNode.CreateHosted Constructor 26152
TXMLNode.Destroy Destructor 26153
TXMLNode Methods 26153
TXMLNode Properties 26189
TXMLNode Events 26194
TXMLNodeCollection Class 26196
TXMLNodeCollection Members 26197
TXMLNodeCollection Methods 26201
TXMLNodeCollection Properties 26209
TXMLNodeList Class 26211
TXMLNodeList Members 26211
TXMLNodeList.Create Constructor 26213
TXMLNodeList.Destroy Destructor 26214
TXMLNodeList Methods 26214
TXMLNodeList Properties 26225
TXMLNodeList Events 26227

Interfaces 26228
IXMLDocumentAccess Interface 26228
IXMLDocumentAccess Members 26228
IXMLDocumentAccess Methods 26228
IXMLDocumentAccess Properties 26229
IXMLNodeAccess Interface 26230
IXMLNodeAccess Members 26230
IXMLNodeAccess Methods 26232
IXMLNodeAccess Properties 26237

Functions 26238
XMLDoc.CheckEncoding Function 26238
XMLDoc.CreateDOMNode Function 26238
XMLDoc.DetectCharEncoding Function 26239
XMLDoc.FormatXMLData Function 26240
XMLDoc.LoadXMLData Function 26240
XMLDoc.LoadXMLDocument Function 26241
XMLDoc.NewXMLDocument Function 26241
XMLDoc.XMLDocError Function 26242
XMLDoc.XMLStringToWideString Function 26242

Structs, Records, Enums 26243
XMLDoc.TNodeChange Enumeration 26243
XMLDoc.TNodeClassInfo Record 26244

XMLDoc.TNodeListOperation Enumeration 26244
XMLDoc.TXMLDocumentSource Enumeration 26245
XMLDoc.TXMLPrologItem Enumeration 26246

Types 26247
XMLDoc.TNodeChangeEvent Type 26247
XMLDoc.TNodeClassArray Type 26247
XMLDoc.TNodeListNotification Type 26248
XMLDoc.TXMLNodeArray Type 26249
XMLDoc.TXMLNodeClass Type 26249
XMLDoc.TXMLNodeCollectionClass Type 26250

Variables 26251
XMLDoc.UseXSDBooleanStrings Variable 26251

xmldom Namespace 26253

Classes 26256
DOMException Class 26256
DOMException Members 26256
DOMException Fields 26257
EDOMParseError Class 26258
EDOMParseError Members 26258
EDOMParseError.Create Constructor 26259
EDOMParseError Properties 26260
TDOMVendor Class 26263
TDOMVendor Members 26263
TDOMVendor Methods 26263
TDOMVendorList Class 26265
TDOMVendorList Members 26265
TDOMVendorList Methods 26266
TDOMVendorList Properties 26268

Interfaces 26270
IDOMAttr Interface 26270
IDOMAttr Members 26271
IDOMAttr Methods 26272
IDOMAttr Properties 26273
IDOMCDATASection Interface 26274
IDOMCharacterData Interface 26275
IDOMCharacterData Members 26275

IDOMCharacterData Methods 26276
IDOMCharacterData Properties 26278
IDOMComment Interface 26279
IDOMDocument Interface 26279
IDOMDocument Members 26279
IDOMDocument Methods 26281
IDOMDocument Properties 26285
IDOMDocumentFragment Interface 26286
IDOMDocumentType Interface 26287
IDOMDocumentType Members 26287
IDOMDocumentType Methods 26288
IDOMDocumentType Properties 26290
IDOMElement Interface 26291
IDOMElement Members 26292
IDOMElement Methods 26293
IDOMElement Properties 26297
IDOMEntity Interface 26298
IDOMEntity Members 26298
IDOMEntity Methods 26299
IDOMEntity Properties 26300
IDOMEntityReference Interface 26301
IDOMImplementation Interface 26301
IDOMImplementation Members 26301
IDOMImplementation Methods 26302
IDOMNamedNodeMap Interface 26303
IDOMNamedNodeMap Members 26303
IDOMNamedNodeMap Methods 26303
IDOMNamedNodeMap Properties 26305
IDOMNode Interface 26306
IDOMNode Members 26307
IDOMNode Methods 26308
IDOMNode Properties 26316
IDOMNodeEx Interface 26320
IDOMNodeEx Members 26320
IDOMNodeEx Methods 26321
IDOMNodeEx Properties 26323
IDOMNodeList Interface 26323
IDOMNodeList Members 26324
IDOMNodeList Methods 26324

IDOMNodeList Properties 26325
IDOMNodeSelect Interface 26325
IDOMNodeSelect Members 26326
IDOMNodeSelect Methods 26326
IDOMNotation Interface 26327
IDOMNotation Members 26327
IDOMNotation Methods 26328
IDOMNotation Properties 26329
IDOMParseError Interface 26329
IDOMParseError Members 26330
IDOMParseError Methods 26330
IDOMParseError Properties 26332
IDOMParseOptions Interface 26334
IDOMParseOptions Members 26334
IDOMParseOptions Methods 26334
IDOMParseOptions Properties 26336
IDOMPersist Interface 26338
IDOMPersist Members 26338
IDOMPersist Methods 26338
IDOMPersist Properties 26341
IDOMProcessingInstruction Interface 26342
IDOMProcessingInstruction Members 26342
IDOMProcessingInstruction Methods 26343
IDOMProcessingInstruction Properties 26344
IDOMText Interface 26345
IDOMText Members 26345
IDOMText Methods 26346
IDOMXMLProlog Interface 26347
IDOMXMLProlog Members 26347
IDOMXMLProlog Methods 26347
IDOMXMLProlog Properties 26349
IDOMXSLProcessor Interface 26350
IDOMXSLProcessor Members 26350
IDOMXSLProcessor Methods 26351
IDOMXSLProcessor Properties 26353

Functions **26354**
xmldom.DOMVendorNotSupported Function 26354
xmldom.ExtractLocalName Function 26354
xmldom.ExtractPrefix Function 26355

xmldom.GetDOM Function 26356
xmldom.GetDOMNodeEx Function 26356
xmldom.GetDOMVendor Function 26357
xmldom.IsPrefixed Function 26357
xmldom.MakeNodeName Function 26358
xmldom.NodeMatches Function 26358
xmldom.RegisterDOMVendor Function 26359
xmldom.SameNamespace Function 26359
xmldom.UnRegisterDOMVendor Function 26360

Types **26362**
xmldom.DOMNodeType Type 26362
xmldom.DOMString Type 26362
xmldom.DOMTimeStamp Type 26363
xmldom.TAsyncEventHandler Type 26363
xmldom.TDOMVendorArray Type 26364

Variables **26365**
xmldom.DefaultDOMVendor Variable 26365
xmldom.DOMVendors Variable 26365

Constants **26366**
xmldom.ATTRIBUTE_NODE Constant 26366
xmldom.CDATA_SECTION_NODE Constant 26367
xmldom.COMMENT_NODE Constant 26367
xmldom.DOCUMENT_FRAGMENT_NODE Constant 26367
xmldom.DOCUMENT_NODE Constant 26368
xmldom.DOCUMENT_TYPE_NODE Constant 26368
xmldom.DOMSTRING_SIZE_ERR Constant 26368
xmldom.DOMWrapperVersion Constant 26369
xmldom.ELEMENT_NODE Constant 26369
xmldom.ENTITY_NODE Constant 26369
xmldom.ENTITY_REFERENCE_NODE Constant 26370
xmldom.HIERARCHY_REQUEST_ERR Constant 26370
xmldom.INDEX_SIZE_ERR Constant 26370
xmldom.INUSE_ATTRIBUTE_ERR Constant 26371
xmldom.INVALID_ACCESS_ERR Constant 26371
xmldom.INVALID_CHARACTER_ERR Constant 26371
xmldom.INVALID_MODIFICATION_ERR Constant 26372
xmldom.INVALID_STATE_ERR Constant 26372
xmldom.NAMESPACE_ERR Constant 26372

xmldom.NO_DATA_ALLOWED_ERR Constant 26373
xmldom.NO_MODIFICATION_ALLOWED_ERR Constant 26373
xmldom.NOT_FOUND_ERR Constant 26373
xmldom.NOT_SUPPORTED_ERR Constant 26374
xmldom.NOTATION_NODE Constant 26374
xmldom.NSDelim Constant 26374
xmldom.PROCESSING_INSTRUCTION_NODE Constant 26375
xmldom.SEncoding Constant 26375
xmldom.SHttp Constant 26375
xmldom.SStandalone Constant 26376
xmldom.SVersion Constant 26376
xmldom.SXML Constant 26376
xmldom.SXMLNamespaceURI Constant 26377
xmldom.SXMLNS Constant 26377
xmldom.SXMLPrefixNamespaceURI Constant 26377
xmldom.SYNTAX_ERR Constant 26378
xmldom.TEXT_NODE Constant 26378
xmldom.WRONG_DOCUMENT_ERR Constant 26378

XMLIntf Namespace 26379

Classes 26381
EXMLDocError Class 26381

Interfaces 26382
IXMLDocument Interface 26382
IXMLDocument Members 26383
IXMLDocument Methods 26384
IXMLDocument Properties 26403
IXMLNode Interface 26411
IXMLNode Members 26412
IXMLNode Methods 26413
IXMLNode Properties 26429
IXMLNodeCollection Interface 26440
IXMLNodeCollection Members 26440
IXMLNodeCollection Methods 26442
IXMLNodeCollection Properties 26444
IXMLNodeList Interface 26445
IXMLNodeList Members 26446
IXMLNodeList Methods 26446

IXMLNodeList Properties 26456

Structs, Records, Enums 26458

XMLIntf.TNodeType Enumeration 26458

XMLIntf.TParseOption Enumeration 26460

XMLIntf.TXMLDocOption Enumeration 26460

XMLIntf.TXMLEncodingType Enumeration 26461

Types 26463

XMLIntf.TParseOptions Type 26463

XMLIntf.TXMLDocOptions Type 26463

Constants 26464

XMLIntf.NodeTypeNames Constant 26464

Xmlxform Namespace 26465

Classes 26466

TranslateException Class 26466

TXMLTransform Class 26467

TXMLTransform Members 26468

TXMLTransform Methods 26471

TXMLTransform Properties 26473

TXMLTransform Events 26479

TXMLTransformClient Class 26482

TXMLTransformClient Members 26482

TXMLTransformClient.Create Constructor 26485

TXMLTransformClient.Destroy Destructor 26485

TXMLTransformClient Methods 26486

TXMLTransformClient Properties 26489

TXMLTransformProvider Class 26492

TXMLTransformProvider Members 26493

TXMLTransformProvider.Create Constructor 26497

TXMLTransformProvider.Destroy Destructor 26497

TXMLTransformProvider Methods 26498

TXMLTransformProvider Properties 26499

TXMLTransformProvider Events 26502

Types 26509

Xmlxform.TRowEvent Type 26509

Xmlxform.TTranslateEvent Type 26509

XSBuiltIns Namespace 26511

Classes **26513**
EXSDateTimeException Class 26513
EXSDecimalException Class 26514
EXSHexBinaryException Class 26514
TXSCustomDateTime Class 26515
TXSCustomDateTime Members 26516
TXSCustomDateTime.Create Constructor 26517
TXSCustomDateTime.Destroy Destructor 26517
TXSCustomDateTime Fields 26518
TXSCustomDateTime Methods 26518
TXSCustomDateTime Properties 26527
TXSDate Class 26534
TXSDate Members 26534
TXSDate.Create Constructor 26535
TXSDate Methods 26536
TXSDate Properties 26539
TXSDateTime Class 26541
TXSDateTime Members 26541
TXSDateTime Methods 26543
TXSDateTime Properties 26544
TXSDecimal Class 26545
TXSDecimal Members 26546
TXSDecimal Methods 26546
TXSDecimal Properties 26547
TXSDuration Class 26548
TXSDuration Members 26549
TXSDuration.Create Constructor 26551
TXSDuration Methods 26551
TXSDuration Properties 26552
TXSHexBinary Class 26553
TXSHexBinary Members 26554
TXSHexBinary Methods 26554
TXSHexBinary Properties 26555
TXSString Class 26556
TXSString Members 26556
TXSString Methods 26557

TXSTime Class 26558
TXSTime Members 26559
TXSTime Methods 26560
TXSTime Properties 26565

Functions 26571
XSBuiltIns.DateTimeToXMLTime Function 26571
XSBuiltIns.DateTimeToXSDateTime Function 26571
XSBuiltIns.GetDataFromFile Function 26572
XSBuiltIns.InitXSTypes Function 26572
XSBuiltIns.SoapFloatToStr Function 26573
XSBuiltIns.SoapStrToFloat Function 26573
XSBuiltIns.XMLTimeToDateTime Function 26573

Types 26575
XSBuiltIns.TXSTimeInstant Type 26575

Constants 26576
XSBuiltIns.IncrementAmount Constant 26576
XSBuiltIns.MaxDay Constant 26577
XSBuiltIns.MaxMonth Constant 26577
XSBuiltIns.MinDay Constant 26577
XSBuiltIns.MinMonth Constant 26578
XSBuiltIns.SDecimal Constant 26578
XSBuiltIns.SHexMarker Constant 26578
XSBuiltIns.SLocalTimeMarker Constant 26579
XSBuiltIns.SNAN Constant 26579
XSBuiltIns.SNegative Constant 26579
XSBuiltIns.SoapDecimalSeparator Constant 26580
XSBuiltIns.SoapTimePrefix Constant 26580
XSBuiltIns.SPlus Constant 26580
XSBuiltIns.SSciNotationMarker Constant 26581
XSBuiltIns.XMLDateSeparator Constant 26581
XSBuiltIns.XMLDayPos Constant 26581
XSBuiltIns.XMLDefaultYearDigits Constant 26582
XSBuiltIns.XMLDurationDay Constant 26582
XSBuiltIns.XMLDurationHour Constant 26582
XSBuiltIns.XMLDurationMinute Constant 26583
XSBuiltIns.XMLDurationMonth Constant 26583
XSBuiltIns.XMLDurationSecond Constant 26583
XSBuiltIns.XMLDurationStart Constant 26584

XSBuiltIns.XMLDurationYear Constant 26584
XSBuiltIns.XMLHourOffsetMinusMarker Constant 26584
XSBuiltIns.XMLHourOffsetPlusMarker Constant 26585
XSBuiltIns.XMLMilSecPos Constant 26585
XSBuiltIns.XMLMonthPos Constant 26585
XSBuiltIns.XMLTimeSeparator Constant 26586
XSBuiltIns.XMLYearPos Constant 26586

XSLProd Namespace 26587

Classes 26588
EXSLPageProducerError Class 26588
TCustomXSLPageProducer Class 26588
TCustomXSLPageProducer Members 26589
TCustomXSLPageProducer Methods 26596
TCustomXSLPageProducer Properties 26603
TXSLPageProducer Class 26605
TXSLPageProducer Members 26606
TXSLPageProducer Properties 26613

ZLib Namespace 26615

Classes 26617
ECompressionError Class 26617
EDecompressionError Class 26617
EZlibError Class 26618
TCompressionStream Class 26618
TCompressionStream Members 26619
TCompressionStream.Create Constructor 26620
TCompressionStream.Destroy Destructor 26620
TCompressionStream Methods 26621
TCompressionStream Properties 26622
TCompressionStream Events 26623
TCustomZlibStream Class 26623
TCustomZlibStream Members 26624
TCustomZlibStream.Create Constructor 26625
TCustomZlibStream Methods 26625
TCustomZlibStream Events 26626
TDecompressionStream Class 26627
TDecompressionStream Members 26627

TDecompressionStream.Create Constructor 26628
TDecompressionStream.Destroy Destructor 26629
TDecompressionStream Methods 26629
TDecompressionStream Events 26631

Functions **26632**
ZLib.CompressBuf Function 26632
ZLib.DecompressBuf Function 26632
ZLib.DecompressToUserBuf Function 26633
ZLib.deflate Function 26633
ZLib.deflateEnd Function 26633
ZLib.deflateInit_ Function 26634
ZLib.inflate Function 26634
ZLib.inflateEnd Function 26634
ZLib.inflateInit_ Function 26635
ZLib.inflateReset Function 26635
ZLib.zlibAllocMem Function 26635
ZLib.zlibFreeMem Function 26636

Structs, Records, Enums **26637**
ZLib.TCompressionLevel Enumeration 26637
ZLib.TZStreamRec Record 26638

Types **26639**
ZLib.TAlloc Type 26639
ZLib.TFree Type 26639

Constants **26640**
ZLib.Z_ASCII Constant 26640
ZLib.Z_BEST_COMPRESSION Constant 26641
ZLib.Z_BEST_SPEED Constant 26641
ZLib.Z_BINARY Constant 26641
ZLib.Z_BUF_ERROR Constant 26642
ZLib.Z_DATA_ERROR Constant 26642
ZLib.Z_DEFAULT_COMPRESSION Constant 26642
ZLib.Z_DEFAULT_STRATEGY Constant 26643
ZLib.Z_DEFLATED Constant 26643
ZLib.Z_ERRNO Constant 26643
ZLib.Z_FILTERED Constant 26644
ZLib.Z_FINISH Constant 26644
ZLib.Z_FULL_FLUSH Constant 26644

ZLib.Z_HUFFMAN_ONLY Constant 26645
ZLib.Z_MEM_ERROR Constant 26645
ZLib.Z_NEED_DICT Constant 26645
ZLib.Z_NO_COMPRESSION Constant 26646
ZLib.Z_NO_FLUSH Constant 26646
ZLib.Z_OK Constant 26646
ZLib.Z_PARTIAL_FLUSH Constant 26647
ZLib.Z_STREAM_END Constant 26647
ZLib.Z_STREAM_ERROR Constant 26647
ZLib.Z_SYNC_FLUSH Constant 26648
ZLib.Z_UNKNOWN Constant 26648
ZLib.Z_VERSION_ERROR Constant 26648
ZLib.zlib_Version Constant 26649

Index

a

1 ActnColorMaps Namespace

This is namespace ActnColorMaps.

Topics

Name	Description
Classes (see page 2)	The following table lists classes in this documentation.
Constants (see page 35)	The following table lists constants in this documentation.

Classes

	Name	Description
	TStandardColorMap (see page 2)	TStandarColorMap defines the default look and feel of individual action band components.
	TTwilightColorMap (see page 13)	TTwilightColorMap defines the default look and feel of individual action band components.
	TXPColorMap (see page 23)	TXPColorMap defines the default look and feel of individual action band components.

Constants

Name	Description
cXPBtnFrameColor (see page 35)	This is constant ActnColorMaps.cXPBtnFrameColor.
cXPFrameOuter (see page 35)	This is constant ActnColorMaps.cXPFrameOuter.
cXPSelectedColor (see page 35)	This is constant ActnColorMaps.cXPSelectedColor.

1.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TStandardColorMap (see page 2)	TStandarColorMap defines the default look and feel of individual action band components.
	TTwilightColorMap (see page 13)	TTwilightColorMap defines the default look and feel of individual action band components.
	TXPColorMap (see page 23)	TXPColorMap defines the default look and feel of individual action band components.

1.1.1 TStandardColorMap Class

TStandarColorMap defines the default look and feel of individual action band components.

Class Hierarchy

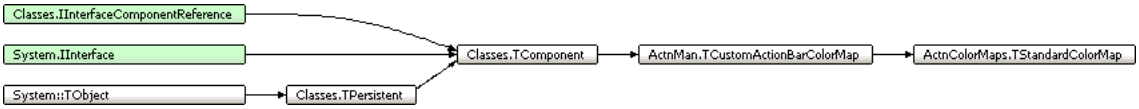

Pascal

```
TStandardColorMap = class(TCustomActionBarColorMap);
```

C++

```
class TStandardColorMap : public TCustomActionBarColorMap;
```

File

ActnColorMaps

Description

Like XPColorMap, use StandardColorMap to apply a unique look and feel to individual action band components.

Note: To centrally apply a unique look and feel to all action band components or to change the default, use Action Manager's Style property.

See Also

TActionManager (see page 432)

TXPColorMap (see page 23)

TTwilightColorMap (see page 13)

1.1.1.1 TStandardColorMap Members

The following tables list the members exposed by TStandardColorMap.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	Add (see page 497)	This is Add, a member of class TCustomActionBarColorMap.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 498)	Forwards notification messages to all owned components.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	Remove (see page 498)	This is Remove, a member of class TCustomActionBarColorMap.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetColor (see page 6)	This is SetColor, a member of class TStandardColorMap.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetFontColor (see page 499)	This is SetFontColor, a member of class TCustomActionBarColorMap.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	UpdateActionBars (see page 499)	This is UpdateActionBars, a member of class TCustomActionBarColorMap.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.

	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 497)	Provides a custom Assign procedure for TCustomActionBarColorMap (see page 492) objects.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 496)	Creates and initializes a TCustomActionBarColorMap (see page 492) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 496)	Destroys an instance of TCustomActionBarColorMap (see page 492).
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateColors (see page 6)	Updates an action component to reflect the current color property.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignFocus (see page 501)	Changes the look and feel of your applications.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnColorChange (see page 12)	Occurs when one of the action's color properties changes.

Published Properties

	Name	Description
	BtnFrameColor (see page 7)	Changes the look and feel of your applications.
	BtnSelectedColor (see page 7)	Changes the look and feel of your applications.
	BtnSelectedFont (see page 7)	Changes the look and feel of your applications.
	Color (see page 8)	Specifies the background color for action bands.
	DisabledColor (see page 8)	Changes the look and feel of your applications.
	DisabledFontColor (see page 8)	Changes the look and feel of your applications.
	DisabledFontShadow (see page 8)	Changes the look and feel of your applications.
	FontColor (see page 9)	Changes the look and feel of your applications.
	FrameBottomRightInner (see page 9)	Changes the look and feel of your applications.
	FrameBottomRightOuter (see page 9)	Changes the look and feel of your applications.
	FrameTopLeftInner (see page 10)	Changes the look and feel of your applications.
	FrameTopLeftOuter (see page 10)	Changes the look and feel of your applications.
	HighlightColor (see page 10)	Changes the look and feel of your applications.
	HotColor (see page 10)	Changes the look and feel of your applications.
	HotFontColor (see page 11)	Changes the look and feel of your applications.
	MenuColor (see page 11)	Changes the look and feel of your applications.
	Name (see page 2805)	Specifies the name of the component as referenced in code.

	SelectedColor (see page 11)	Changes the look and feel of your applications.
	SelectedFontColor (see page 11)	Changes the look and feel of your applications.
	ShadowColor (see page 12)	Changes the look and feel of your applications.
	Tag (see page 2808)	Stores an integer value as part of a component.
	UnusedColor (see page 12)	Changes the look and feel of your applications.

1.1.1.2 TStandardColorMap Methods

The methods of the TStandardColorMap class are listed here.

Protected Methods

	Name	Description
	SetColor (see page 6)	This is SetColor, a member of class TStandardColorMap.

Public Methods

	Name	Description
	UpdateColors (see page 6)	Updates an action component to reflect the current color property.

1.1.1.2.1 TStandardColorMap.SetColor Method

Pascal

```
procedure SetColor(const Value: TColor); override;
```

C++

```
virtual __fastcall SetColor(const TColor Value);
```

Description

This is SetColor, a member of class TStandardColorMap.

1.1.1.2.2 TStandardColorMap.UpdateColors Method

Updates an action component to reflect the current color property.

Pascal

```
procedure UpdateColors; override;
```

C++

```
virtual __fastcall UpdateColors();
```

Description

UpdateColors is called automatically to update associated action components.

1.1.1.3 TStandardColorMap Properties

The properties of the TStandardColorMap class are listed here.

Published Properties

	Name	Description
	BtnFrameColor (see page 7)	Changes the look and feel of your applications.
	BtnSelectedColor (see page 7)	Changes the look and feel of your applications.
	BtnSelectedFont (see page 7)	Changes the look and feel of your applications.

	Color (see page 8)	Specifies the background color for action bands.
	DisabledColor (see page 8)	Changes the look and feel of your applications.
	DisabledFontColor (see page 8)	Changes the look and feel of your applications.
	DisabledFontShadow (see page 8)	Changes the look and feel of your applications.
	FontColor (see page 9)	Changes the look and feel of your applications.
	FrameBottomRightInner (see page 9)	Changes the look and feel of your applications.
	FrameBottomRightOuter (see page 9)	Changes the look and feel of your applications.
	FrameTopLeftInner (see page 10)	Changes the look and feel of your applications.
	FrameTopLeftOuter (see page 10)	Changes the look and feel of your applications.
	HighlightColor (see page 10)	Changes the look and feel of your applications.
	HotColor (see page 10)	Changes the look and feel of your applications.
	HotFontColor (see page 11)	Changes the look and feel of your applications.
	MenuColor (see page 11)	Changes the look and feel of your applications.
	SelectedColor (see page 11)	Changes the look and feel of your applications.
	SelectedFontColor (see page 11)	Changes the look and feel of your applications.
	ShadowColor (see page 12)	Changes the look and feel of your applications.
	UnusedColor (see page 12)	Changes the look and feel of your applications.

1.1.1.3.1 TStandardColorMap.BtnFrameColor Property

Changes the look and feel of your applications.

Pascal

```
property BtnFrameColor: TColor;
```

C++

```
__property TColor BtnFrameColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.3.2 TStandardColorMap.BtnSelectedColor Property

Changes the look and feel of your applications.

Pascal

```
property BtnSelectedColor: TColor;
```

C++

```
__property TColor BtnSelectedColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.3.3 TStandardColorMap.BtnSelectedFont Property

Changes the look and feel of your applications.

Pascal

```
property BtnSelectedFont: TColor;
```

C++

```
__property TColor BtnSelectedFont;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.3.4 TStandardColorMap.Color Property

Specifies the background color for action bands.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

The Color property determines the background color. You can set Color to one of the constants defined in the Graphics (see page 14387) unit (such as clBlue), or to an explicit RGB integer value.

1.1.1.3.5 TStandardColorMap.DisabledColor Property

Changes the look and feel of your applications.

Pascal

```
property DisabledColor: TColor;
```

C++

```
__property TColor DisabledColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.3.6 TStandardColorMap.DisabledFontColor Property

Changes the look and feel of your applications.

Pascal

```
property DisabledFontColor: TColor;
```

C++

```
__property TColor DisabledFontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.3.7 TStandardColorMap.DisabledFontShadow Property

Changes the look and feel of your applications.

Pascal

```
property DisabledFontShadow: TColor;
```

C++

```
__property TColor DisabledFontShadow;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.3.8 TStandardColorMap.FontColor Property

Changes the look and feel of your applications.

Pascal

```
property FontColor: TColor;
```

C++

```
__property TColor FontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.3.9 TStandardColorMap.FrameBottomRightInner Property

Changes the look and feel of your applications.

Pascal

```
property FrameBottomRightInner: TColor;
```

C++

```
__property TColor FrameBottomRightInner;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.3.10 TStandardColorMap.FrameBottomRightOuter Property

Changes the look and feel of your applications.

Pascal

```
property FrameBottomRightOuter: TColor;
```

C++

```
__property TColor FrameBottomRightOuter;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.3.11 TStandardColorMap.FrameTopLeftInner Property

Changes the look and feel of your applications.

Pascal

```
property FrameTopLeftInner: TColor;
```

C++

```
__property TColor FrameTopLeftInner;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.3.12 TStandardColorMap.FrameTopLeftOuter Property

Changes the look and feel of your applications.

Pascal

```
property FrameTopLeftOuter: TColor;
```

C++

```
__property TColor FrameTopLeftOuter;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.3.13 TStandardColorMap.HighlightColor Property

Changes the look and feel of your applications.

Pascal

```
property HighlightColor: TColor;
```

C++

```
__property TColor HighlightColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.3.14 TStandardColorMap.HotColor Property

Changes the look and feel of your applications.

Pascal

```
property HotColor: TColor;
```

C++

```
__property TColor HotColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.3.15 TStandardColorMap.HotFontColor Property

Changes the look and feel of your applications.

Pascal

```
property HotFontColor: TColor;
```

C++

```
__property TColor HotFontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.3.16 TStandardColorMap.MenuColor Property

Changes the look and feel of your applications.

Pascal

```
property MenuColor: TColor;
```

C++

```
__property TColor MenuColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.3.17 TStandardColorMap.SelectedColor Property

Changes the look and feel of your applications.

Pascal

```
property SelectedColor: TColor;
```

C++

```
__property TColor SelectedColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.3.18 TStandardColorMap.SelectedFontColor Property

Changes the look and feel of your applications.

Pascal

```
property SelectedFontColor: TColor;
```

C++

```
__property TColor SelectedFontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.3.19 TStandardColorMap.ShadowColor Property

Changes the look and feel of your applications.

Pascal

```
property ShadowColor: TColor;
```

C++

```
__property TColor ShadowColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.3.20 TStandardColorMap.UnusedColor Property

Changes the look and feel of your applications.

Pascal

```
property UnusedColor: TColor;
```

C++

```
__property TColor UnusedColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.1.4 TStandardColorMap Events

The events of the TStandardColorMap class are listed here.

Published Events

	Name	Description
	OnColorChange (see page 12)	Occurs when one of the action's color properties changes.

1.1.1.4.1 TStandardColorMap.OnColorChange Event

Occurs when one of the action's color properties changes.

Pascal

```
property OnColorChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnColorChange;
```

Description

Applications can't use the protected OnChange event. It is used internally to manage the relationship between the properties of the action and the corresponding properties of the action's clients.

Component writers can use OnChange in descendant objects to respond when the action's properties change.

1.1.2 TTwilightColorMap Class

TTwilightColorMap defines the default look and feel of individual action band components.

Class Hierarchy

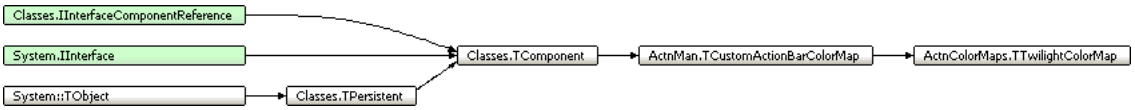

Pascal

```
TTwilightColorMap = class(TCustomActionBarColorMap);
```

C++

```
class TTwilightColorMap : public TCustomActionBarColorMap;
```

File

ActnColorMaps

Description

Like XPColorMap, use TwilightColorMap to apply a unique look and feel to individual action band components.

Note: To centrally apply a unique look and feel to all action band components or to change the default, use Action Manager's Style property.

See Also

TActionManager (see page 432)

TXPColorMap (see page 23)

TStandardColorMap (see page 2)

1.1.2.1 TTwilightColorMap Members

The following tables list the members exposed by TTwilightColorMap.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	Add (see page 497)	This is Add, a member of class TCustomActionBarColorMap.

	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 498)	Forwards notification messages to all owned components.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	Remove (see page 498)	This is Remove, a member of class TCustomActionBarColorMap.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetColor (see page 499)	This is SetColor, a member of class TCustomActionBarColorMap.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFontColor (see page 499)	This is SetFontColor, a member of class TCustomActionBarColorMap.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	UpdateActionBars (see page 499)	This is UpdateActionBars, a member of class TCustomActionBarColorMap.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 497)	Provides a custom Assign procedure for TCustomActionBarColorMap (see page 492) objects.
	Assign (see page 2930)	Copies the contents of another, similar object.

	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 496)	Creates and initializes a TCustomActionBarColorMap (see page 492) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 496)	Destroys an instance of TCustomActionBarColorMap (see page 492).
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateColors (see page 17)	Updates an action component to reflect the current color property.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignFocus (see page 501)	Changes the look and feel of your applications.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnColorChange (see page 23)	Occurs when one of the action's color properties changes.

Published Properties

	Name	Description
	BtnFrameColor (see page 17)	Changes the look and feel of your applications.
	BtnSelectedColor (see page 18)	Changes the look and feel of your applications.
	BtnSelectedFont (see page 18)	Changes the look and feel of your applications.
	Color (see page 18)	Specifies the background color for action bands.
	DisabledColor (see page 19)	Changes the look and feel of your applications.
	DisabledFontColor (see page 19)	Changes the look and feel of your applications.
	DisabledFontShadow (see page 19)	Changes the look and feel of your applications.
	FontColor (see page 19)	Changes the look and feel of your applications.
	FrameBottomRightInner (see page 20)	Changes the look and feel of your applications.
	FrameBottomRightOuter (see page 20)	Changes the look and feel of your applications.
	FrameTopLeftInner (see page 20)	Changes the look and feel of your applications.
	FrameTopLeftOuter (see page 20)	Changes the look and feel of your applications.
	HighlightColor (see page 21)	Changes the look and feel of your applications.
	HotColor (see page 21)	Changes the look and feel of your applications.
	HotFontColor (see page 21)	Changes the look and feel of your applications.
	MenuColor (see page 21)	Changes the look and feel of your applications.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SelectedColor (see page 22)	Changes the look and feel of your applications.
	SelectedFontColor (see page 22)	Changes the look and feel of your applications.
	ShadowColor (see page 22)	Changes the look and feel of your applications.
	Tag (see page 2808)	Stores an integer value as part of a component.
	UnusedColor (see page 23)	Changes the look and feel of your applications.

1.1.2.2 TTwilightColorMap Methods

The methods of the TTwilightColorMap class are listed here.

Public Methods

	Name	Description
	UpdateColors (see page 17)	Updates an action component to reflect the current color property.

1.1.2.2.1 TTwilightColorMap.UpdateColors Method

Updates an action component to reflect the current color property.

Pascal

```
procedure UpdateColors; override;
```

C++

```
virtual __fastcall UpdateColors();
```

Description

UpdateColors is called automatically to update associated action components.

1.1.2.3 TTwilightColorMap Properties

The properties of the TTwilightColorMap class are listed here.

Published Properties

	Name	Description
	BtnFrameColor (see page 17)	Changes the look and feel of your applications.
	BtnSelectedColor (see page 18)	Changes the look and feel of your applications.
	BtnSelectedFont (see page 18)	Changes the look and feel of your applications.
	Color (see page 18)	Specifies the background color for action bands.
	DisabledColor (see page 19)	Changes the look and feel of your applications.
	DisabledFontColor (see page 19)	Changes the look and feel of your applications.
	DisabledFontShadow (see page 19)	Changes the look and feel of your applications.
	FontColor (see page 19)	Changes the look and feel of your applications.
	FrameBottomRightInner (see page 20)	Changes the look and feel of your applications.
	FrameBottomRightOuter (see page 20)	Changes the look and feel of your applications.
	FrameTopLeftInner (see page 20)	Changes the look and feel of your applications.
	FrameTopLeftOuter (see page 20)	Changes the look and feel of your applications.
	HighlightColor (see page 21)	Changes the look and feel of your applications.
	HotColor (see page 21)	Changes the look and feel of your applications.
	HotFontColor (see page 21)	Changes the look and feel of your applications.
	MenuColor (see page 21)	Changes the look and feel of your applications.
	SelectedColor (see page 22)	Changes the look and feel of your applications.
	SelectedFontColor (see page 22)	Changes the look and feel of your applications.
	ShadowColor (see page 22)	Changes the look and feel of your applications.
	UnusedColor (see page 23)	Changes the look and feel of your applications.

1.1.2.3.1 TTwilightColorMap.BtnFrameColor Property

Changes the look and feel of your applications.

Pascal

```
property BtnFrameColor: TColor;
```

C++

```
__property TColor BtnFrameColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.3.2 TTwilightColorMap.BtnSelectedColor Property

Changes the look and feel of your applications.

Pascal

```
property BtnSelectedColor: TColor;
```

C++

```
__property TColor BtnSelectedColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.3.3 TTwilightColorMap.BtnSelectedFont Property

Changes the look and feel of your applications.

Pascal

```
property BtnSelectedFont: TColor;
```

C++

```
__property TColor BtnSelectedFont;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.3.4 TTwilightColorMap.Color Property

Specifies the background color for action bands.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

The Color property determines the background color. You can set Color to one of the constants defined in the Graphics unit (such as clBlue), or to an explicit RGB integer value.

1.1.2.3.5 TTwilightColorMap.DisabledColor Property

Changes the look and feel of your applications.

Pascal

```
property DisabledColor: TColor;
```

C++

```
__property TColor DisabledColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.3.6 TTwilightColorMap.DisabledFontColor Property

Changes the look and feel of your applications.

Pascal

```
property DisabledFontColor: TColor;
```

C++

```
__property TColor DisabledFontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.3.7 TTwilightColorMap.DisabledFontShadow Property

Changes the look and feel of your applications.

Pascal

```
property DisabledFontShadow: TColor;
```

C++

```
__property TColor DisabledFontShadow;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.3.8 TTwilightColorMap.FontColor Property

Changes the look and feel of your applications.

Pascal

```
property FontColor: TColor;
```

C++

```
__property TColor FontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.3.9 TTwilightColorMap.FrameBottomRightInner Property

Changes the look and feel of your applications.

Pascal

```
property FrameBottomRightInner: TColor;
```

C++

```
__property TColor FrameBottomRightInner;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.3.10 TTwilightColorMap.FrameBottomRightOuter Property

Changes the look and feel of your applications.

Pascal

```
property FrameBottomRightOuter: TColor;
```

C++

```
__property TColor FrameBottomRightOuter;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.3.11 TTwilightColorMap.FrameTopLeftInner Property

Changes the look and feel of your applications.

Pascal

```
property FrameTopLeftInner: TColor;
```

C++

```
__property TColor FrameTopLeftInner;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.3.12 TTwilightColorMap.FrameTopLeftOuter Property

Changes the look and feel of your applications.

Pascal

```
property FrameTopLeftOuter: TColor;
```

C++

```
__property TColor FrameTopLeftOuter;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.3.13 TTwilightColorMap.HighlightColor Property

Changes the look and feel of your applications.

Pascal

```
property HighlightColor: TColor;
```

C++

```
__property TColor HighlightColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.3.14 TTwilightColorMap.HotColor Property

Changes the look and feel of your applications.

Pascal

```
property HotColor: TColor;
```

C++

```
__property TColor HotColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.3.15 TTwilightColorMap.HotFontColor Property

Changes the look and feel of your applications.

Pascal

```
property HotFontColor: TColor;
```

C++

```
__property TColor HotFontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.3.16 TTwilightColorMap.MenuColor Property

Changes the look and feel of your applications.

Pascal

```
property MenuColor: TColor;
```

C++

```
__property TColor MenuColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.3.17 TTwilightColorMap.SelectedColor Property

Changes the look and feel of your applications.

Pascal

```
property SelectedColor: TColor;
```

C++

```
__property TColor SelectedColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.3.18 TTwilightColorMap.SelectedFontColor Property

Changes the look and feel of your applications.

Pascal

```
property SelectedFontColor: TColor;
```

C++

```
__property TColor SelectedFontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.3.19 TTwilightColorMap.ShadowColor Property

Changes the look and feel of your applications.

Pascal

```
property ShadowColor: TColor;
```

C++

```
__property TColor ShadowColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.3.20 TTwilightColorMap.UnusedColor Property

Changes the look and feel of your applications.

Pascal

```
property UnusedColor: TColor;
```

C++

```
__property TColor UnusedColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.2.4 TTwilightColorMap Events

The events of the TTwilightColorMap class are listed here.

Published Events

	Name	Description
	OnColorChange (see page 23)	Occurs when one of the action's color properties changes.

1.1.2.4.1 TTwilightColorMap.OnColorChange Event

Occurs when one of the action's color properties changes.

Pascal

```
property OnColorChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnColorChange;
```

Description

Applications can't use the protected OnChange event. It is used internally to manage the relationship between the properties of the action and the corresponding properties of the action's clients.

Component writers can use OnChange in descendant objects to respond when the action's properties change.

1.1.3 TXPColorMap Class

TXPColorMap defines the default look and feel of individual action band components.

Class Hierarchy

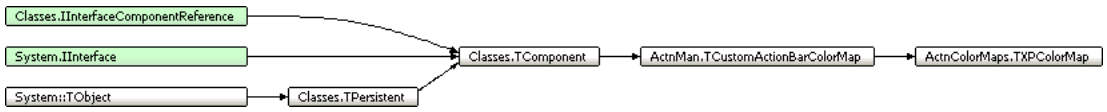

Pascal

```
TXPColorMap = class(TCustomActionBarColorMap);
```

C++

```
class TXPColorMap : public TCustomActionBarColorMap;
```

File

ActnColorMaps

Description

Like StandardColorMap, use XPColorMap to apply a unique look and feel to individual action band components.

Note: To centrally apply a unique look and feel to all action band components or to change the default, use Action Manager's Style property.

See Also

TActionManager (see page 432)

TStandardColorMap (see page 2)

TTwilightColorMap (see page 13)

1.1.3.1 TXPColorMap Members

The following tables list the members exposed by TXPColorMap.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	Add (see page 497)	This is Add, a member of class TCustomActionBarColorMap.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 498)	Forwards notification messages to all owned components.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.

	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	Remove (see page 498)	This is Remove, a member of class TCustomActionBarColorMap.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetColor (see page 499)	This is SetColor, a member of class TCustomActionBarColorMap.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFontColor (see page 499)	This is SetFontColor, a member of class TCustomActionBarColorMap.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	UpdateActionBars (see page 499)	This is UpdateActionBars, a member of class TCustomActionBarColorMap.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 497)	Provides a custom Assign procedure for TCustomActionBarColorMap (see page 492) objects.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 496)	Creates and initializes a TCustomActionBarColorMap (see page 492) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 496)	Destroys an instance of TCustomActionBarColorMap (see page 492).
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.

	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateColors (see page 27)	Updates an action component to reflect the current color property.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignFocus (see page 501)	Changes the look and feel of your applications.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnColorChange (see page 34)	Occurs when one of the action's color properties changes.

Published Properties

	Name	Description
	BtnFrameColor (see page 28)	Changes the look and feel of your applications.
	BtnSelectedColor (see page 28)	Changes the look and feel of your applications.
	BtnSelectedFont (see page 29)	Changes the look and feel of your applications.
	Color (see page 29)	Specifies the background color for action bands.
	DisabledColor (see page 29)	Changes the look and feel of your applications.
	DisabledFontColor (see page 29)	Changes the look and feel of your applications.
	DisabledFontShadow (see page 30)	Changes the look and feel of your applications.
	FontColor (see page 30)	Changes the look and feel of your applications.
	FrameBottomRightInner (see page 30)	Changes the look and feel of your applications.
	FrameBottomRightOuter (see page 30)	Changes the look and feel of your applications.
	FrameTopLeftInner (see page 31)	Changes the look and feel of your applications.
	FrameTopLeftOuter (see page 31)	Changes the look and feel of your applications.
	HighlightColor (see page 31)	Changes the look and feel of your applications.
	HotColor (see page 31)	Changes the look and feel of your applications.
	HotFontColor (see page 32)	Changes the look and feel of your applications.
	MenuColor (see page 32)	Changes the look and feel of your applications.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SelectedColor (see page 32)	Changes the look and feel of your applications.
	SelectedFontColor (see page 33)	Changes the look and feel of your applications.
	ShadowColor (see page 33)	Changes the look and feel of your applications.
	Tag (see page 2808)	Stores an integer value as part of a component.
	UnusedColor (see page 33)	Changes the look and feel of your applications.

1.1.3.2 TXPColorMap Methods

The methods of the TXPColorMap class are listed here.

Public Methods

	Name	Description
	UpdateColors (see page 27)	Updates an action component to reflect the current color property.

1.1.3.2.1 TXPColorMap.UpdateColors Method

Updates an action component to reflect the current color property.

Pascal

```
procedure UpdateColors; override;
```

C++

```
virtual __fastcall UpdateColors();
```

Description

UpdateColors is called automatically to update associated action components.

1.1.3.3 TXPColorMap Properties

The properties of the TXPColorMap class are listed here.

Published Properties

	Name	Description
	BtnFrameColor (see page 28)	Changes the look and feel of your applications.
	BtnSelectedColor (see page 28)	Changes the look and feel of your applications.
	BtnSelectedFont (see page 29)	Changes the look and feel of your applications.
	Color (see page 29)	Specifies the background color for action bands.
	DisabledColor (see page 29)	Changes the look and feel of your applications.
	DisabledFontColor (see page 29)	Changes the look and feel of your applications.
	DisabledFontShadow (see page 30)	Changes the look and feel of your applications.
	FontColor (see page 30)	Changes the look and feel of your applications.
	FrameBottomRightInner (see page 30)	Changes the look and feel of your applications.
	FrameBottomRightOuter (see page 30)	Changes the look and feel of your applications.
	FrameTopLeftInner (see page 31)	Changes the look and feel of your applications.
	FrameTopLeftOuter (see page 31)	Changes the look and feel of your applications.
	HighlightColor (see page 31)	Changes the look and feel of your applications.
	HotColor (see page 31)	Changes the look and feel of your applications.
	HotFontColor (see page 32)	Changes the look and feel of your applications.
	MenuColor (see page 32)	Changes the look and feel of your applications.
	SelectedColor (see page 32)	Changes the look and feel of your applications.
	SelectedFontColor (see page 33)	Changes the look and feel of your applications.
	ShadowColor (see page 33)	Changes the look and feel of your applications.
	UnusedColor (see page 33)	Changes the look and feel of your applications.

1.1.3.3.1 TXPColorMap.BtnFrameColor Property

Changes the look and feel of your applications.

Pascal

```
property BtnFrameColor: TColor;
```

C++

```
__property TColor BtnFrameColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.3.3.2 TXPColorMap.BtnSelectedColor Property

Changes the look and feel of your applications.

Pascal

```
property BtnSelectedColor: TColor;
```

C++

```
__property TColor BtnSelectedColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your

applications.

1.1.3.3.3 TXPColorMap.BtnSelectedFont Property

Changes the look and feel of your applications.

Pascal

```
property BtnSelectedFont: TColor;
```

C++

```
__property TColor BtnSelectedFont;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.3.3.4 TXPColorMap.Color Property

Specifies the background color for action bands.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

The Color property determines the background color. You can set Color to one of the constants defined in the Graphics unit (such as clBlue), or to an explicit RGB integer value.

1.1.3.3.5 TXPColorMap.DisabledColor Property

Changes the look and feel of your applications.

Pascal

```
property DisabledColor: TColor;
```

C++

```
__property TColor DisabledColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.3.3.6 TXPColorMap.DisabledFontColor Property

Changes the look and feel of your applications.

Pascal

```
property DisabledFontColor: TColor;
```

C++

```
__property TColor DisabledFontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.3.3.7 TXPColorMap.DisabledFontShadow Property

Changes the look and feel of your applications.

Pascal

```
property DisabledFontShadow: TColor;
```

C++

```
__property TColor DisabledFontShadow;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.3.3.8 TXPColorMap.FontColor Property

Changes the look and feel of your applications.

Pascal

```
property FontColor: TColor;
```

C++

```
__property TColor FontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.3.3.9 TXPColorMap.FrameBottomRightInner Property

Changes the look and feel of your applications.

Pascal

```
property FrameBottomRightInner: TColor;
```

C++

```
__property TColor FrameBottomRightInner;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.3.3.10 TXPColorMap.FrameBottomRightOuter Property

Changes the look and feel of your applications.

Pascal

```
property FrameBottomRightOuter: TColor;
```

C++

```
__property TColor FrameBottomRightOuter;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.3.3.11 TXPColorMap.FrameTopLeftInner Property

Changes the look and feel of your applications.

Pascal

```
property FrameTopLeftInner: TColor;
```

C++

```
__property TColor FrameTopLeftInner;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.3.3.12 TXPColorMap.FrameTopLeftOuter Property

Changes the look and feel of your applications.

Pascal

```
property FrameTopLeftOuter: TColor;
```

C++

```
__property TColor FrameTopLeftOuter;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.3.3.13 TXPColorMap.HighlightColor Property

Changes the look and feel of your applications.

Pascal

```
property HighlightColor: TColor;
```

C++

```
__property TColor HighlightColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.3.3.14 TXPColorMap.HotColor Property

Changes the look and feel of your applications.

Pascal

```
property HotColor: TColor;
```

C++

```
__property TColor HotColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.3.3.15 TXPColorMap.HotFontColor Property

Changes the look and feel of your applications.

Pascal

```
property HotFontColor: TColor;
```

C++

```
__property TColor HotFontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.3.3.16 TXPColorMap.MenuColor Property

Changes the look and feel of your applications.

Pascal

```
property MenuColor: TColor;
```

C++

```
__property TColor MenuColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.3.3.17 TXPColorMap.SelectedColor Property

Changes the look and feel of your applications.

Pascal

```
property SelectedColor: TColor;
```

C++

```
__property TColor SelectedColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.3.3.18 TXPColorMap.SelectedFontColor Property

Changes the look and feel of your applications.

Pascal

```
property SelectedFontColor: TColor;
```

C++

```
__property TColor SelectedFontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.3.3.19 TXPColorMap.ShadowColor Property

Changes the look and feel of your applications.

Pascal

```
property ShadowColor: TColor;
```

C++

```
__property TColor ShadowColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.3.3.20 TXPColorMap.UnusedColor Property

Changes the look and feel of your applications.

Pascal

```
property UnusedColor: TColor;
```

C++

```
__property TColor UnusedColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

1.1.3.4 TXPColorMap Events

The events of the TXPColorMap class are listed here.

Published Events

	Name	Description
	OnColorChange (see page 34)	Occurs when one of the action's color properties changes.

1.1.3.4.1 TXPColorMap.OnColorChange Event

Occurs when one of the action's color properties changes.

Pascal

```
property OnColorChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnColorChange;
```

Description

Applications can't use the protected OnChange event. It is used internally to manage the relationship between the properties of the action and the corresponding properties of the action's clients.

Component writers can use OnChange in descendant objects to respond when the action's properties change.

1.2 Constants

The following table lists constants in this documentation.

Constants

Name	Description
cXPBtnFrameColor (see page 35)	This is constant ActnColorMaps.cXPBtnFrameColor.
cXPFrameOuter (see page 35)	This is constant ActnColorMaps.cXPFrameOuter.
cXPSelectedColor (see page 35)	This is constant ActnColorMaps.cXPSelectedColor.

1.2.1 ActnColorMaps.cXPBtnFrameColor Constant

Pascal

```
cXPBtnFrameColor = $00C66931;
```

C++

```
cXPBtnFrameColor = 0x00C66931;
```

File

ActnColorMaps

Description

This is constant ActnColorMaps.cXPBtnFrameColor.

1.2.2 ActnColorMaps.cXPFrameOuter Constant

Pascal

```
cXPFrameOuter = $007A868A;
```

C++

```
cXPFrameOuter = 0x007A868A;
```

File

ActnColorMaps

Description

This is constant ActnColorMaps.cXPFrameOuter.

1.2.3 ActnColorMaps.cXPSelectedColor Constant

Pascal

```
cXPSelectedColor = $00EFD3C6;
```

C++

```
cXPSelectedColor = 0x00EFD3C6;
```

File

ActnColorMaps

Description

This is constant ActnColorMaps.cXPSelectedColor.

2 ActnCtrls Namespace

This is namespace ActnCtrls.

Topics

Name	Description
Classes (see page 38)	The following table lists classes in this documentation.
Types (see page 245)	The following table lists types in this documentation.

Classes

	Name	Description
	TActionToolBar (see page 38)	TActionToolBar is a rendering engine that renders action client items as tool buttons.
	TCustomActionCombo (see page 118)	This is class ActnCtrls.TCustomActionCombo.
	TCustomActionDockBar (see page 133)	TCustomActionDockBar is the ancestor for action bands that require the ability to dock.
	TCustomActionToolBar (see page 148)	TCustomActionToolBar is the base class for the TActionToolBar (see page 38) component.
	TCustomButtonControl (see page 168)	TCustomButtonControl is the base class for ActionBand button controls.
	TCustomComboControl (see page 191)	This is class ActnCtrls.TCustomComboControl.
	TCustomDropDownButton (see page 205)	This is class ActnCtrls.TCustomDropDownButton.
	TCustomToolScrollBtn (see page 217)	This is class ActnCtrls.TCustomToolScrollBtn.
	TCustomUtilityButton (see page 228)	This is class ActnCtrls.TCustomUtilityButton.

Types

Name	Description
TCustomToolScrollBtnClass (see page 245)	This is type ActnCtrls.TCustomToolScrollBtnClass.

2.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TActionToolBar (see page 38)	TActionToolBar is a rendering engine that renders action client items as tool buttons.
	TCustomActionCombo (see page 118)	This is class ActnCtrls.TCustomActionCombo.
	TCustomActionDockBar (see page 133)	TCustomActionDockBar is the ancestor for action bands that require the ability to dock.
	TCustomActionToolBar (see page 148)	TCustomActionToolBar is the base class for the TActionToolBar (see page 38) component.
	TCustomButtonControl (see page 168)	TCustomButtonControl is the base class for ActionBand button controls.
	TCustomComboControl (see page 191)	This is class ActnCtrls.TCustomComboControl.
	TCustomDropDownButton (see page 205)	This is class ActnCtrls.TCustomDropDownButton.
	TCustomToolScrollBtn (see page 217)	This is class ActnCtrls.TCustomToolScrollBtn.
	TCustomUtilityButton (see page 228)	This is class ActnCtrls.TCustomUtilityButton.

2.1.1 TActionToolBar Class

TActionToolBar is a rendering engine that renders action client items as tool buttons.

Class Hierarchy

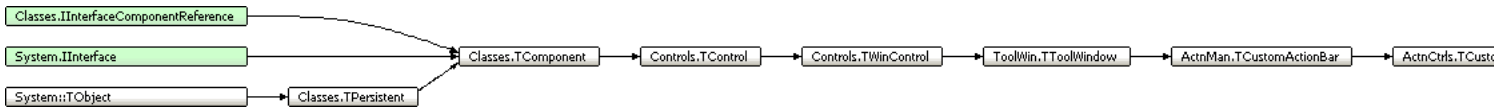

Pascal

```
TActionToolBar = class(TCustomActionToolBar);
```

C++

```
class TActionToolBar : public TCustomActionToolBar;
```

File

ActnCtrls

Description

TActionToolBar is a specialized container (like TPanel) that renders a group of items on screen in the order specified in the Action Manager. All of the items are created dynamically when the TActionToolBar component is connected to an item from the Action Manager.

The items themselves are internal components (similar to TSpeedButton) created by the TActionToolBar and appear consistent with the look and function of a tool bar. The layout of the items on the TActionToolBar is managed automatically and specified by the Action Manager.

Note: If you want to use TActionToolBar on a TControlBar component, you should set the AutoSize property to **False**. When the AutoSize property is set to True (default), the TActionToolBar control wraps its controls and expands itself in size, accordingly. This does not allow the TActionToolBar component to display properly on a TControlBar component. When TActionToolBar is

set to **False**, the TActionToolBar hides items on the end of the bar and displays a dropdown button that provides access to those items. The control displays in the proper size.

See Also

TActionManager (see page 432)

Adding Icons to Menus and Toolbars

Creating Customizable Toolbars and Menus

Creating Toolbars and Menus

Organizing Actions for Toolbars and Menus

Selecting Menu and Toolbar Styles

Setting Up Action Bands

What Is an Action?

2.1.1.1 TActionToolBar Members

The following tables list the members exposed by TActionToolBar.

Protected Events

	Name	Description
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefaultColorMap (see page 460)	This is FDefaultColorMap, a member of class TCustomActionBar.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSavedWrapState (see page 460)	This is FSavedWrapState, a member of class TCustomActionBar.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.

	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 462)	Aligns any controls for which the control is the parent within a specified area of the control.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	AutoSizingChanged (see page 163)	This is AutoSizingChanged, a member of class TCustomActionToolBar.
	AutoSizingChanged (see page 462)	This is AutoSizingChanged, a member of class TCustomActionBar.
	BackgroundSize (see page 462)	This is BackgroundSize, a member of class TCustomActionBar.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcButtonWidth (see page 163)	This is CalcButtonWidth, a member of class TCustomActionToolBar.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcInsertPosition (see page 462)	This is CalcInsertPosition, a member of class TCustomActionBar.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Clear (see page 463)	This is Clear, a member of class TCustomActionBar.
	Click (see page 5743)	Respond to user click.
	CMBorderChanged (see page 23285)	This is CMBorderChanged, a member of class TToolWindow.
	CMColorchanged (see page 163)	This is CMColorchanged, a member of class TCustomActionToolBar.
	CMCtl3DChanged (see page 23286)	This is CMCtl3DChanged, a member of class TToolWindow.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateControl (see page 163)	This is CreateControl, a member of class TCustomActionToolBar.
	CreateControl (see page 463)	This is CreateControl, a member of class TCustomActionBar.
	CreateControls (see page 164)	This is CreateControls, a member of class TCustomActionToolBar.
	CreateControls (see page 463)	This is CreateControls, a member of class TCustomActionBar.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 463)	Returns placement order of custom-aligned child control.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 464)	Returns aligned position of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.

	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
V	DesignWndProc (see page 465)	This is DesignWndProc, a member of class TCustomActionBar.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DisableHiding (see page 164)	This is DisableHiding, a member of class TCustomActionToolBar.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
V	DoControlCreated (see page 465)	This is DoControlCreated, a member of class TCustomActionBar.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
V	DoDropActions (see page 465)	This is DoDropActions, a member of class TCustomActionBar.
V	DoDropCategory (see page 164)	This is DoDropCategory, a member of class TCustomActionToolBar.
V	DoDropCategory (see page 465)	This is DoDropCategory, a member of class TCustomActionBar.
V	DoDropItem (see page 466)	This is DoDropItem, a member of class TCustomActionBar.
V	DoEndDock (see page 146)	OnEndDock event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetControlClass (see page 466)	This is DoGetControlClass, a member of class TCustomActionBar.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
V	DoStartDock (see page 147)	OnStartDock event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 466)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawBackground (see page 165)	This is DrawBackground, a member of class TCustomActionToolBar.
V	DrawBackground (see page 467)	This is DrawBackground, a member of class TCustomActionBar.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawSeparator (see page 165)	This is DrawSeparator, a member of class TCustomActionToolBar.

	EnableHiding (see page 165)	This is EnableHiding, a member of class TCustomActionToolBar.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindAccelItem (see page 467)	This is FindAccelItem, a member of class TCustomActionBar.
	FindItemAt (see page 468)	This is FindItemAt, a member of class TCustomActionBar.
	FindItemCaption (see page 468)	This is FindItemCaption, a member of class TCustomActionBar.
	FindLast (see page 468)	This is FindLast, a member of class TCustomActionBar.
	FindLeastUsedItem (see page 469)	This is FindLeastUsedItem, a member of class TCustomActionBar.
	FindNearestControl (see page 469)	This is FindNearestControl, a member of class TCustomActionBar.
	FindNext (see page 470)	This is FindNext, a member of class TCustomActionBar.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FindNextVisibleItem (see page 470)	This is FindNextVisibleItem, a member of class TCustomActionBar.
	FindPrevious (see page 470)	This is FindPrevious, a member of class TCustomActionBar.
	FindPreviousVisibleItem (see page 470)	This is FindPreviousVisibleItem, a member of class TCustomActionBar.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionControl (see page 471)	This is GetActionControl, a member of class TCustomActionBar.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBannerWidth (see page 471)	This is GetBannerWidth, a member of class TCustomActionBar.
	GetBarHeight (see page 471)	This is GetBarHeight, a member of class TCustomActionBar.
	GetBarWidth (see page 471)	This is GetBarWidth, a member of class TCustomActionBar.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlClass (see page 165)	This is GetControlClass, a member of class TCustomActionToolBar.
	GetControlClass (see page 471)	This is GetControlClass, a member of class TCustomActionBar.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDefaultColorMapClass (see page 472)	This is GetDefaultColorMapClass, a member of class TCustomActionBar.
	GetDesignMode (see page 472)	This is GetDesignMode, a member of class TCustomActionBar.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 147)	Returns the class used for the control when it is floating.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupClass (see page 166)	This is GetPopupClass, a member of class TCustomActionToolBar.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetPopupMenuClass (see page 472)	This is GetPopupMenuClass, a member of class TCustomActionBar.
	GetScrollBtnClass (see page 166)	This is GetScrollBtnClass, a member of class TCustomActionToolBar.
	GetSiteInfo (see page 147)	Generates an OnGetSiteInfo event

	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasItems (see page 472)	This is HasItems, a member of class TCustomActionBar.
	HideUnusedItems (see page 166)	This is HideUnusedItems, a member of class TCustomActionToolBar.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
V	ItemCount (see page 473)	This is ItemCount, a member of class TCustomActionBar.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 148)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 473)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	NCPaint (see page 473)	This is NCPaint, a member of class TCustomActionBar.
V	NCPaint (see page 23286)	This is NCPaint, a member of class TToolWindow.
V	Notification (see page 473)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 474)	This is Paint, a member of class TCustomActionBar.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 474)	Renders the image of a windowed control.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
V	Reset (see page 166)	This is Reset, a member of class TCustomActionToolBar.

	Reset (see page 475)	This is Reset, a member of class TCustomActionBar.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	ScrollBtnClick (see page 166)	This is ScrollBtnClick, a member of class TCustomActionToolBar.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActionClient (see page 475)	This is SetActionClient, a member of class TCustomActionBar.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetAutoSizing (see page 475)	This is SetAutoSizing, a member of class TCustomActionBar.
	SetBiDiMode (see page 475)	Sets the control's bi-directional mode.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetColorMap (see page 476)	This is SetColorMap, a member of class TCustomActionBar.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignMode (see page 476)	This is SetDesignMode, a member of class TCustomActionBar.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetOrientation (see page 167)	This is SetOrientation, a member of class TCustomActionToolBar.
	SetOrientation (see page 477)	This is SetOrientation, a member of class TCustomActionBar.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetPersistentHotKeys (see page 477)	This is SetPersistentHotKeys, a member of class TCustomActionBar.
	SetSpacing (see page 477)	This is SetSpacing, a member of class TCustomActionBar.
	SetupDefaultColorMap (see page 477)	This is SetupDefaultColorMap, a member of class TCustomActionBar.
	SetupDropDownBtn (see page 167)	This is SetupDropDownBtn, a member of class TCustomActionToolBar.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.

V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	VisibleChanging (see page 478)	Provides the interface for a method that responds to changes in the control's visibility.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMNCCalcSize (see page 23286)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCPaint (see page 23286)	This is WMNCPaint, a member of class TToolWindow.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
R	HRowCount (see page 488)	Indicates the number of columns of items that the action band displays.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	Items (see page 488)	Lists the action clients that correspond to the child controls of this action band.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
R	VRowCount (see page 490)	Indicates the number of rows of items that the action band displays.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
R	Widest (see page 491)	This is Widest, a member of class TCustomActionBar.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CalcDockedWidth (see page 146)	This is CalcDockedWidth, a member of class TCustomActionDockBar.
	CalcDockedWidth (see page 163)	This is CalcDockedWidth, a member of class TCustomActionToolBar.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 145)	Creates and initializes a TCustomActionDockBar (see page 133) instance.
	Create (see page 161)	Creates and initializes a TCustomActionToolBar (see page 148) instance.
	Create (see page 459)	Creates and initializes a TCustomActionBar (see page 447) instance.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 5731)	Creates an instance of TControl (see page 5724).
	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 23285)	Creates and initializes a TToolWindow (see page 23274) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 162)	Destroys an instance of TCustomActionToolBar (see page 148).
	Destroy (see page 459)	Destroys an instance of TCustomActionBar (see page 447).
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 146)	Used internally to dock the action band.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 164)	Generates an OnDragDrop event.
	DragDrop (see page 466)	Generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindFirst (see page 467)	Finds the first TActionClientItem (see page 374) in the Items (see page 488) list.
	FindFirstVisibleItem (see page 468)	Finds the first visible TActionClientItem (see page 374) in the Items (see page 488) list.
	FindLastVisibleItem (see page 469)	Finds the last visibleTActionClientItem in the Items (see page 488) list.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
V	RecreateControls (see page 475)	Destroys and recreates all of the action band's action client controls.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 167)	Sets the Left, Top, Width, and Height properties in a single operation.
V	SetBounds (see page 476)	Sets the bounds based on whether the popup is expanded or not
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Style (see page 477)	Identifies the style object that gives the application its default look and feel.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.

	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	ActionClient (see page 479)	Specifies the action client associated with this action band.
R	ActionControls (see page 479)	Lists the controls on the action band.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AutoSizing (see page 480)	Specifies whether the action band adjusts its size automatically to fit around its child controls.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	Canvas (see page 58)	Represents the drawing surface of the action band.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	ContextBar (see page 485)	Indicates whether items shown in this action band represent the items in a popup menu.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designable (see page 485)	Specifies whether the action band can be customized (used internally by the IDE).
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignMode (see page 486)	Specifies whether the action band is being customized or being used in the IDE.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
R	HiddenCount (see page 168)	Indicates the number of hidden items on the tool action band.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.

	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	VertSeparator (see page 490)	Indicates whether a vertical separator bar should appear between item lines.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 83)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 84)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 84)	Occurs when an attempt is made to resize the control.
	OnClick (see page 85)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 87)	Adjust resize constraints.
	OnContextPopup (see page 87)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnControlCreated (see page 88)	Occurs when a control is created on the action band.
	OnDblClick (see page 88)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 89)	Occurs when another control is docked to the control.
	OnDockOver (see page 91)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 93)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 95)	Occurs when the user drags an object over a control.
	OnEndDock (see page 97)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 98)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 99)	Occurs when a control receives the input focus.
	OnExit (see page 101)	Occurs when the input focus shifts away from one control to another.
	OnGetControlClass (see page 103)	Occurs when an item is to be placed on the action band.
	OnGetSiteInfo (see page 103)	Returns the control's docking information.
	OnMouseActivate (see page 105)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 106)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 109)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 111)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 113)	Occurs when the action band is painted.
	OnResize (see page 113)	Occurs immediately after the control is resized.
	OnStartDock (see page 114)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 115)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 115)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	ActionManager (see page 52)	Specifies the action manager component that is associated with the action band.
	Align (see page 53)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AllowHiding (see page 56)	Determines whether the action band can be hidden.
	Anchors (see page 57)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 57)	Specifies the bi-directional mode for the control.
	Caption (see page 58)	Specifies a text string that identifies the control to the user.
	Color (see page 61)	Specifies the background color of the control.
	ColorMap (see page 61)	Indicates the colormap for individual action band components.
	Constraints (see page 62)	Specifies the size constraints for the control.
	Cursor (see page 63)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 65)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 66)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 66)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EdgeBorders (see page 68)	Determines which sides of the control have visible borders.
	EdgeInner (see page 68)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 68)	Determines the border style for the control's outer edge.
	Enabled (see page 69)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 69)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HorzMargin (see page 70)	Specifies the width of the horizontal margins in pixels.
	HorzSeparator (see page 71)	Indicates whether a horizontal separator bar appears between item lines.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Orientation (see page 71)	Specifies how the action band lays out its child items.
	ParentBackground (see page 71)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 72)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 72)	Determines where a control looks for its color information.
	ParentFont (see page 73)	Determines where a control looks for its font information.
	ParentShowHint (see page 73)	Determines where a control looks to find out if its Help Hint should be shown.
	PersistentHotKeys (see page 74)	Specifies whether keyboard accelerator shortcuts are always displayed.
	PopupMenu (see page 74)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 79)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Spacing (see page 80)	Determines where the image and text appear on an action band's rendered control that represents the action client.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	VertMargin (see page 81)	Specifies the width of the vertical margins in pixels.
	Visible (see page 81)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

2.1.1.2 TActionToolBar Properties

The properties of the TActionToolBar class are listed here.

Public Properties

	Name	Description
	Canvas (see page 58)	Represents the drawing surface of the action band.

Published Properties

	Name	Description
	ActionManager (see page 52)	Specifies the action manager component that is associated with the action band.
	Align (see page 53)	Determines how the control aligns within its container (parent control).
	AllowHiding (see page 56)	Determines whether the action band can be hidden.
	Anchors (see page 57)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 57)	Specifies the bi-directional mode for the control.
	Caption (see page 58)	Specifies a text string that identifies the control to the user.
	Color (see page 61)	Specifies the background color of the control.
	ColorMap (see page 61)	Indicates the colormap for individual action band components.
	Constraints (see page 62)	Specifies the size constraints for the control.
	Cursor (see page 63)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 65)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 66)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 66)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EdgeBorders (see page 68)	Determines which sides of the control have visible borders.
	EdgeInner (see page 68)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 68)	Determines the border style for the control's outer edge.
	Enabled (see page 69)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 69)	Controls the attributes of text written on or in the control.
	HorzMargin (see page 70)	Specifies the width of the horizontal margins in pixels.
	HorzSeparator (see page 71)	Indicates whether a horizontal separator bar appears between item lines.
	Orientation (see page 71)	Specifies how the action band lays out its child items.
	ParentBackground (see page 71)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 72)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 72)	Determines where a control looks for its color information.
	ParentFont (see page 73)	Determines where a control looks for its font information.
	ParentShowHint (see page 73)	Determines where a control looks to find out if its Help Hint should be shown.
	PersistentHotKeys (see page 74)	Specifies whether keyboard accelerator shortcuts are always displayed.
	PopupMenu (see page 74)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 79)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Spacing (see page 80)	Determines where the image and text appear on an action band's rendered control that represents the action client.
	VertMargin (see page 81)	Specifies the width of the vertical margins in pixels.
	Visible (see page 81)	Determines whether the component appears onscreen.

2.1.1.2.1 TActionToolBar.ActionManager Property

Specifies the action manager component that is associated with the action band.

Pascal

```
property ActionManager: TCustomActionManager;
```

C++

```
__property TCustomActionManager ActionManager;
```

Description

Use ActionManager to access the action manager that uses this action band to render UI elements it manages. Typically, this association is created when you drag actions from the action manager and drop them on the action bar. When an action is dropped on an action band and the ActionManager property is not assigned, the ActionManager property is automatically assigned to the action manager that manages the action that was dropped on the action band.

At runtime, you can perform the equivalent operation to dragging an action onto the action band by setting the ActionBar property of one of the action clients that the action manager's ActionBars property lists so that it specifies this component.

See Also

ActionBars (see page 553)

ActionClient (see page 479)

ActionBar (see page 370)

2.1.1.2.2 TActionToolBar.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
```

```
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
```

```
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
```

```
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

2.1.1.2.3 TActionToolBar.AllowHiding Property

Determines whether the action band can be hidden.

Pascal

```
property AllowHiding: Boolean;
```

C++

```
__property Boolean AllowHiding;
```

Description

Use AllowHiding to specify whether the action band can be hidden at runtime, if the application allows runtime customization by the end user. Setting AllowHiding to true allows the action band to be hidden at runtime. If AllowHiding is set to false, the action cannot be hidden. Certain action bands, such as the main menu action band, should set this AllowHiding to false.

2.1.1.2.4 TActionToolBar.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

2.1.1.2.5 TActionToolBar.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency,

ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

2.1.1.2.6 TActionToolBar.Canvas Property

Represents the drawing surface of the action band.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

TCustomActionBar uses Canvas internally to render the background on which its child controls appear.

2.1.1.2.7 TActionToolBar.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
```

```
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
```

```
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

2.1.1.2.8 TActionToolBar.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

2.1.1.2.9 TActionToolBar.ColorMap Property

Indicates the colormap for individual action band components.

Pascal

```
property ColorMap: TCustomActionBarColorMap;
```

C++

```
__property TCustomActionBarColorMap ColorMap;
```

Description

Use the ColorMap property to change the default colormap for individual action band components. By default, the ColorMap property is set according to the first registered TActionBarStyleEx class. Otherwise, if the action band is linked to an action manager, the property defaults to the ColorMap as specified by the Style property of the action manager.

Note: To centrally set or change the default look and feel of all action band components, use ActionManager's Style property. To centrally control the colors used by all action bands within an application, connect the action bands to a single colormap component.

See Also

TActionManager (see page 432)

TXPColorMap (see page 23)

TStandardColorMap (see page 2)

TTwilightColorMap (see page 13)

Style (see page 556)

2.1.1.2.10 TActionToolBar.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

2.1.1.2.11 TActionToolBar.Cursor Property

Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.

Pascal

```
property Cursor: TCursor;
```

C++

```
__property TCursor Cursor;
```

Description

Change the value of Cursor to provide feedback to the user when the mouse pointer enters the control. The value of Cursor is the index of the cursor in the list of cursors maintained by the global variable, Screen. In addition to the built-in cursors provided by TScreen, applications can add custom cursors to the list.

C++ Examples:

```
/*
This example shows how to add custom cursors to an
application.  The following code makes this cursor available
to the application via the constant crMyCursor, and sets it
as the global cursor to the application.
*/
Graphics::TBitmap *bmpMask, *bmpColor;
TIconInfo *iconInfo;

const crMyCursor = 5;

  void __fastcall TForm1::Button2Click(TObject *Sender)
{
  bmpMask = new Graphics::TBitmap();
  bmpColor = new Graphics::TBitmap();
  iconInfo = new TIconInfo();

  bmpMask->LoadFromFile("../SquareMask.bmp");
  bmpColor->LoadFromFile("../Square.bmp");

  iconInfo->fIcon = false;
  iconInfo->xHotspot = 15;
  iconInfo->yHotspot = 15;
  iconInfo->hbmMask = bmpMask->Handle;
  iconInfo->hbmColor = bmpColor->Handle;

  Screen->Cursors[crMyCursor] = CreateIconIndirect(iconInfo);

  Screen->Cursor = crMyCursor;

  delete bmpMask;
  delete bmpColor;
  delete iconInfo;
}
```

Delphi Examples:

```
{
This example uses a splitter and two image controls to
create a "slide show" effect.  The example displays an image
(the current "slide"), which can be hidden behind a screen
that looks like a venetian blind. When the blind is pulled
down to completely cover the current slide, the slide changes
to a different image.  When the blind is raised, the new
image appears.
To prepare this example, follow these steps:
```

```
1. Place an image control on the form.
Set the picture property to a bitmap of a venetian blind,
Set the Stretch property to True.
Set the Align property to alTop.
2. Add a splitter control to the form.
Set the Align property to alTop.
Set the Cursor property to crVSplit.
Set the OnMoved event handler to the code that appears below.
Set the MinSize property to 1.
3. Add a second image control to the form.
Set the Align property to alClient.
Set the Stretch property to True.
4. Select the form and add the OnCreate and OnDestroy event
   handlers that appear below.
5. Add the global variables below to the unit.
}
var
  CurImage: Integer;
  BMPs: Array[1..5] of TBitmap;

procedure TForm1.Splitter1Moved(Sender: TObject);
begin
  { check if the blind is closed }
  if Image2.Height <= 1 then
  begin
    { the blind is closed, change the current slide }
    CurImage := CurImage + 1;
    if CurImage = 6 then
       CurImage := 1;
    Image2.Picture.Bitmap := BMPs[CurImage];
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { Initialize the array of slides.
    These must be freed in the OnDestroy event handler. }
  BMPs[1] := TBitmap.Create;
  BMPs[1].LoadFromFile('Factory.bmp');
  BMPs[2] := TBitmap.Create;
  BMPs[2].LoadFromFile('Rhododendron.bmp');
  BMPs[3] := TBitmap.Create;
  BMPs[3].LoadFromFile('littleB_64.bmp');
  BMPs[4] := TBitmap.Create;
  BMPs[4].LoadFromFile('Soap bubbles.bmp');
  BMPs[5] := TBitmap.Create;
  BMPs[5].LoadFromFile('Gone Fishing.bmp');
  CurImage := 1;
  Image2.Picture.Bitmap := BMPs[1];
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  { clean up }
  for I := 1 to 5 do
    BMPs[I].Free;
end;
{
This example shows how to add custom cursors to an
application.  The following code makes this cursor available
to the application via the constant crMyCursor, and sets it
as the global cursor to the application.
}
var
```

```
    bmpMask : TBitmap;
    bmpColor : TBitmap;
    iconInfo : TIconInfo;
  const
    crMyCursor = 5;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    bmpMask := TBitmap.Create;
    bmpColor := TBitmap.Create;

    bmpMask.LoadFromFile('SquareMask.bmp');
    bmpColor.LoadFromFile('Square.bmp');

    with iconInfo do
    begin
      fIcon := false;
      xHotspot := 15;
      yHotspot := 15;
      hbmMask := bmpMask.Handle;
      hbmColor := bmpColor.Handle;
    end;

    Screen.Cursors[crMyCursor] := CreateIconIndirect(iconInfo);

    Screen.Cursor := crMyCursor;

    bmpMask.Free;
    bmpColor.Free;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    DestroyIcon(Screen.Cursors[crMyCursor]);
  end;
```

See Also

DragCursor (see page 5818)

Cursors (see page 14212)

Screen (see page 14385)

ProcessMessages (see page 13691)

2.1.1.2.12 TActionToolBar.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

2.1.1.2.13 TActionToolBar.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

2.1.1.2.14 TActionToolBar.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
```

```
    else
      return;
    TreeView1->Selected->MoveTo(Item, AttachMode);
  }

  void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
        int X, int Y, TDragState State, bool &Accept)
  {
    Accept = (Source == (TObject *)TreeView1);
  }
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

2.1.1.2.15 TActionToolBar.EdgeBorders Property

Determines which sides of the control have visible borders.

Pascal

```
property EdgeBorders: TEdgeBorders;
```

C++

```
__property TEdgeBorders EdgeBorders;
```

Description

Use the EdgeBorders property to add or remove borders. The appearance of the borders can be set with the EdgeInner and EdgeOuter properties.

See Also

EdgeInner (see page 23287)

EdgeOuter (see page 23288)

2.1.1.2.16 TActionToolBar.EdgeInner Property

Determines the border style for the control's inner edge.

Pascal

```
property EdgeInner: TEdgeStyle;
```

C++

```
__property TEdgeStyle EdgeInner;
```

Description

Use EdgeInner to provide the tool window with a three-dimensional beveled look. A tool window has two borders on each side, an outer border drawn next to the outer edge, and an inner border drawn inside the outer border. EdgeInner specifies the appearance of the inner border. It can be raised (esRaised), lowered (esLowered), or left off altogether (esNone).

Note: The EdgeInner property has an effect only on the sides specified by the EdgeBorders property.

Use EdgeInner with EdgeOuter to give the component a three-dimensional look.

See Also

EdgeBorders (see page 23287)

EdgeOuter (see page 23288)

2.1.1.2.17 TActionToolBar.EdgeOuter Property

Determines the border style for the control's outer edge.

Pascal

```
property EdgeOuter: TEdgeStyle;
```

C++

```
__property TEdgeStyle EdgeOuter;
```

Description

Use EdgeOuter to provide the tool window with a three-dimensional beveled look. A tool window has two borders on each side,

an outer border drawn next to the outer edge, and an inner border drawn inside the outer border. EdgeOuter specifies the appearance of the outer border. It can be raised (esRaised), lowered (esLowered), or left off altogether (esNone).

Note: The EdgeInner property has an effect only on the sides specified by the EdgeBorders property.

Use EdgeOuter with EdgeInner to give the component a three-dimensional look.

See Also

EdgeBorders (see page 23287)

EdgeInner (see page 23287)

2.1.1.2.18 TActionToolBar.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

2.1.1.2.19 TActionToolBar.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
```

```
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

2.1.1.2.20 TActionToolBar.HorzMargin Property

Specifies the width of the horizontal margins in pixels.

Pascal

```
property HorzMargin: Integer;
```

C++

```
__property int HorzMargin;
```

Description

HorzMargin specifies the width of the action band's left and right horizontal margins in pixels.

See Also

VertMargin (see page 490)

VertSeparator (see page 490)

2.1.1.2.21 TActionToolBar.HorzSeparator Property

Indicates whether a horizontal separator bar appears between item lines.

Pascal

```
property HorzSeparator: Boolean;
```

C++

```
__property Boolean HorzSeparator;
```

Description

An action band can arrange its child controls using multiple lines. Items which cannot be displayed on one line wrap to the next in the same way that words wrap on most word processors. Set HorzSeparator to true if you want a separator bar to appear between the lines.

See Also

VertSeparator (see page 490)

HorzMargin (see page 487)

Orientation (see page 488)

2.1.1.2.22 TActionToolBar.Orientation Property

Specifies how the action band lays out its child items.

Pascal

```
property Orientation: TBarOrientation;
```

C++

```
__property TBarOrientation Orientation;
```

Description

The possible values of Orientation are:

Value	Meaning
boLeftToRight	Items are rendered in rows, from left to right.
boRightToLeft	Items are rendered in rows, from right to left.
boTopToBottom	Items are rendered in columns, from top to bottom.
boBottomToTop	Items are rendered in columns, from bottom to top.

See Also

VertSeparator (see page 490)

HorzSeparator (see page 487)

ActionControls (see page 479)

2.1.1.2.23 TActionToolBar.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground: Boolean;
```

C++

```
__property Boolean ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

2.1.1.2.24 TActionToolBar.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

2.1.1.2.25 TActionToolBar.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

2.1.1.2.26 TActionToolBar.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

2.1.1.2.27 TActionToolBar.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

2.1.1.2.28 TActionToolBar.PersistentHotKeys Property

Specifies whether keyboard accelerator shortcuts are always displayed.

Pascal

```
property PersistentHotKeys: Boolean;
```

C++

```
__property Boolean PersistentHotKeys;
```

Description

Use PersistentHotKeys to specify whether the action band always displays keyboard accelerator shortcuts. Setting PersistentHotKeys to true causes the underline for the keyboard shortcut key to always appear. If PersistentHotKeys is set to false, the underline for the keyboard shortcut key is only displayed if the user has used the keyboard to navigate through the action band's items. If the user is only using the mouse to navigate through the action band's items and PersistentHotKeys is set to false, the underline under the character denoted as the shortcut key is not drawn.

Note: Accelerator shortcuts are specified by preceding the appropriate character in the item's caption with an ampersand.

See Also

Caption (see page 384)

2.1.1.2.29 TActionToolBar.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
```

```
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
```

```
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
```

```
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
```

```
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

2.1.1.2.30 TActionToolBar.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
```

```
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

2.1.1.2.31 TActionToolBar.Spacing Property

Determines where the image and text appear on an action band's rendered control that represents the action client.

Pascal

```
property Spacing: Integer;
```

C++

```
__property int Spacing;
```

Description

Spacing is used internally to set the number of pixels that should appear between the image specified in an action client's ImageIndex and the text specified in its Caption property.

See Also

Caption (see page 384)

ImageIndex (see page 385)

2.1.1.2.32 TActionToolBar.VertMargin Property

Specifies the width of the vertical margins in pixels.

Pascal

```
property VertMargin: Integer;
```

C++

```
__property int VertMargin;
```

Description

VertMargin specifies the width of the action band's top and bottom vertical margins in pixels.

See Also

HorzMargin (see page 487)

HorzSeparator (see page 487)

2.1.1.2.33 TActionToolBar.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
```

```
      pPage->Caption = AnsiString("Page") + IntToStr(i);
      TEdit *pEdit = new TEdit(this);
      pEdit->Parent = pPage;
      pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
      pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
    }
    PageControl1Change(Sender);
  }
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

2.1.1.3 TActionToolBar Events

The events of the TActionToolBar class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 83)	Occurs when an object with custom alignment is aligned.

	OnAlignPosition (see page 84)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 84)	Occurs when an attempt is made to resize the control.
	OnClick (see page 85)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 87)	Adjust resize constraints.
	OnContextPopup (see page 87)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnControlCreated (see page 88)	Occurs when a control is created on the action band.
	OnDblClick (see page 88)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 89)	Occurs when another control is docked to the control.
	OnDockOver (see page 91)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 93)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 95)	Occurs when the user drags an object over a control.
	OnEndDock (see page 97)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 98)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 99)	Occurs when a control receives the input focus.
	OnExit (see page 101)	Occurs when the input focus shifts away from one control to another.
	OnGetControlClass (see page 103)	Occurs when an item is to be placed on the action band.
	OnGetSiteInfo (see page 103)	Returns the control's docking information.
	OnMouseActivate (see page 105)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 106)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 109)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 111)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 113)	Occurs when the action band is painted.
	OnResize (see page 113)	Occurs immediately after the control is resized.
	OnStartDock (see page 114)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 115)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 115)	Occurs when the application tries to undock a control that is docked to the windowed control.

2.1.1.3.1 TActionToolBar.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore: TAlignInsertBeforeEvent;
```

C++

```
__property TAlignInsertBeforeEvent OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore (see page 6137)

Align (see page 5798)

TAlignInsertBeforeEvent (see page 6293)

2.1.1.3.2 TActionToolBar.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition: TAlignPositionEvent;
```

C++

```
__property TAlignPositionEvent OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition (see page 6138)

Align (see page 5798)

Anchors (see page 5802)

TAlignInfo (see page 6269)

TAlignPositionEvent (see page 6293)

2.1.1.3.3 TActionToolBar.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize: TCanResizeEvent;
```

C++

```
__property TCanResizeEvent OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize (see page 5853)

SetBounds (see page 5783)

OnResize (see page 5870)

2.1.1.3.4 TActionToolBar.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
```

```
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

2.1.1.3.5 TActionToolBar.OnConstrainedResize Event

Adjust resize constraints.

Pascal

```
property OnConstrainedResize: TConstrainedResizeEvent;
```

C++

```
__property TConstrainedResizeEvent OnConstrainedResize;
```

Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

Note: The OnConstrainedResize handler is called immediately after the OnCanResize handler.

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

Constraints (see page 5812)

OnResize (see page 5870)

CanAutoSize (see page 5741)

2.1.1.3.6 TActionToolBar.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The

OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

2.1.1.3.7 TActionToolBar.OnControlCreated Event

Occurs when a control is created on the action band.

Pascal

```
property OnControlCreated: TControlCreatedEvent;
```

C++

```
__property TControlCreatedEvent OnControlCreated;
```

Description

Write an OnControlCreated event handler to enhance or replace the type of UI element that is rendered on the action band.

Sender is the action band component that just created a new control.

AnItem is the new child control that was just created. The OnControlCreated event handler can initialize the properties of this control or replace it with a custom control.

See Also

OnGetControlClass (see page 492)

2.1.1.3.8 TActionToolBar.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

2.1.1.3.9 TActionToolBar.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop: TDockDropEvent;
```

C++

```
__property TDockDropEvent OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
```

```
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
```

```
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockOver (see page 6216)

2.1.1.3.10 TActionToolBar.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver: TDockOverEvent;
```

C++

```
__property TDockOverEvent OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
```

```
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
```

```
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnGetSiteInfo (see page 6223)

TDragDockObject (see page 6009)

2.1.1.3.11 TActionToolBar.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
```

```
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

2.1.1.3.12 TActionToolBar.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
```

```
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
```

```
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

2.1.1.3.13 TActionToolBar.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

2.1.1.3.14 TActionToolBar.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
```

```
    S := (Sender as TLabel).Name + ' was dropped... and ';
    if Target <> nil then
      S := S + 'accepted!'
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

2.1.1.3.15 TActionToolBar.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
```

```
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

2.1.1.3.16 TActionToolBar.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
```

```
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

2.1.1.3.17 TActionToolBar.OnGetControlClass Event

Occurs when an item is to be placed on the action band.

Pascal

```
property OnGetControlClass: TGetControlClassEvent;
```

C++

```
__property TGetControlClassEvent OnGetControlClass;
```

Description

Write an OnGetControlClass event handler to specify the class of item that the action band should create to represent a specified action client.

Sender is the action band that needs to generate a child control to represent an action client.

AnItem is the action client that the new control should represent.

ControlClass returns the class reference for the new control. This should be a descendant of TCustomActionControl.

See Also

TCustomActionControl (see page 507)

OnControlCreated (see page 491)

2.1.1.3.18 TActionToolBar.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo: TGetSiteInfoEvent;
```

C++

```
__property TGetSiteInfoEvent OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
```

```
    //OnDockDrop gets called AFTER the client has actually docked,
    //so we check for DockClientCount = 1 before making the dock panel visible.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, True, nil);
    (Sender as TPanel).DockManager.ResetBounds(True);
    //Make DockManager repaints it's clients.
  end;

  procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer; State: TDragState;
    var Accept: Boolean);
  var
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

TDragDockObject (see page 6009)

2.1.1.3.19 TActionToolBar.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse

button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

2.1.1.3.20 TActionToolBar.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area

of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

2.1.1.3.21 TActionToolBar.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

2.1.1.3.22 TActionToolBar.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
```

```
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
```

```
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

2.1.1.3.23 TActionToolBar.OnPaint Event

Occurs when the action band is painted.

Pascal

```
property OnPaint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPaint;
```

Description

Write an OnPaint event handler to change the way the action band renders itself.

See Also

Canvas (see page 481)

2.1.1.3.24 TActionToolBar.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

2.1.1.3.25 TActionToolBar.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

2.1.1.3.26 TActionToolBar.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

2.1.1.3.27 TActionToolBar.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock: TUnDockEvent;
```

C++

```
__property TUnDockEvent OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
```

```
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;
```

```
procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

2.1.2 TCustomActionCombo Class

Class Hierarchy

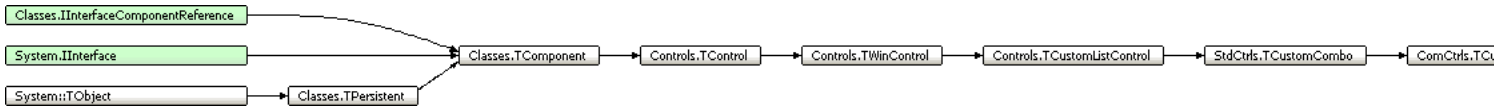

Pascal

```
TCustomActionCombo = class(TCustomComboBoxEx);
```

C++

```
class TCustomActionCombo : public TCustomComboBoxEx;
```

File

ActnCtrls

Description

This is class ActnCtrls.TCustomActionCombo.

2.1.2.1 TCustomActionCombo Members

The following tables list the members exposed by TCustomActionCombo.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.

	OnChange (see page 21197)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnCloseUp (see page 21199)	Occurs when the drop-down list closes up due to some user action.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDropDown (see page 21199)	Occurs when the user opens the drop-down list.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnSelect (see page 21201)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefEditProc (see page 21167)	This is FDefEditProc, a member of class TCustomCombo.
	FDefListProc (see page 21167)	This is FDefListProc, a member of class TCustomCombo.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FDropHandle (see page 21168)	This is FDropHandle, a member of class TCustomCombo.
	FDroppingDown (see page 21168)	This is FDroppingDown, a member of class TCustomCombo.
	FEditHandle (see page 21168)	This is FEditHandle, a member of class TCustomCombo.

	FEditInstance (see page 21168)	This is FEditInstance, a member of class TCustomCombo.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFocusChanged (see page 21168)	This is FFocusChanged, a member of class TCustomCombo.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FIsFocused (see page 21169)	This is FIsFocused, a member of class TCustomCombo.
	FListHandle (see page 21169)	This is FListHandle, a member of class TCustomCombo.
	FListInstance (see page 21169)	This is FListInstance, a member of class TCustomCombo.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveIndex (see page 21169)	This is FSaveIndex, a member of class TCustomCombo.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActionChange (see page 3692)	Responds when the action for which this combo box is a client changes.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustDropDown (see page 21171)	Adjusts the size and position of the drop down list.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
V	BeginAutoDrag (see page 131)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21171)	Generates an OnChange (see page 21197) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	CloseUp (see page 21172)	Generates an OnCloseUp (see page 21199) event.
	CMColorChanged (see page 3693)	Responds when the Color property changes.
	CMParentColorChanged (see page 3693)	Responds when the ParentColor property changes.
	CNNotify (see page 3693)	Responds to notifications of changes in the combo box.
	ComboExWndProc (see page 3694)	Responds to Windows messages sent to the combo box.
V	ComboWndProc (see page 131)	This is ComboWndProc, a member of class TCustomActionCombo.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.

	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 3694)	Initializes the window-creation parameter record.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 3694)	Creates the underlying Windows control that implements the extended combo box.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CreateWnd (see page 21174)	Creates the Windows control that represents the combo box.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
V	DesignWndProc (see page 132)	This is DesignWndProc, a member of class TCustomActionCombo.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWindowHandle (see page 21174)	Destroys all of the windows controls that implement the combo box.
V	DestroyWnd (see page 3695)	Destroys the Windows control that represents the combo box.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 132)	OnDragOver event dispatcher.

	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DropDown (see page 21175)	Generates an OnDropDown (see page 21199) event.
V	EditWndProc (see page 21175)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
V	GetActionLinkClass (see page 3695)	Returns the action link class that is associated with extended combo boxes.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetDroppedDown (see page 21176)	Returns the value of the DroppedDown (see page 21185) property.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetItemCount (see page 3696)	Returns the number of items in the Items property list.
A	GetItemCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
V	GetItemHt (see page 3696)	Returns the height, in pixels, allocated for each item in the drop-down list.
A	GetItemHt (see page 21177)	Returns the value of the ItemHeight (see page 21186) property.
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
V	GetItemIndex (see page 21177)	Returns the value of the ItemIndex property.
V	GetItemsClass (see page 3696)	Returns the class used to implement the Items property.
A	GetItemsClass (see page 21177)	Indicates the class that implements the Items (see page 21187) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelLength (see page 21178)	Returns the value of the SelLength (see page 21190) property.
	GetSelStart (see page 21178)	Returns the value of the SelStart (see page 21193) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).

	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	ListWndProc (see page 21178)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 21179)	Initializes the combo box after its form is loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 3697)	Responds to notifications that objects are about to be inserted or removed.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Select (see page 21179)	Generates an OnSelect (see page 21201) event.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.

	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropDownCount (see page 3697)	Sets the value of the DropDownCount property.
	SetDropDownCount (see page 21180)	Sets the value of the DropDownCount (see page 21185) property.
	SetDroppedDown (see page 21180)	Sets the value of the DroppedDown (see page 21185) property.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItemHeight (see page 21181)	Sets the value of the ItemHeight (see page 21186) property.
	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
	SetItemIndex (see page 21181)	Sets the value of the ItemIndex property.
	SetItems (see page 21181)	Sets the value of the Items (see page 21187) property.
	SetMaxLength (see page 21182)	Sets the value of the MaxLength (see page 21189) property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelLength (see page 21182)	Sets the value of the SelLength (see page 21190) property.
	SetSelStart (see page 21182)	Sets the value of the SelStart (see page 21193) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.

	WMHelp (see page 3697)	This is WMHelp, a member of class TCustomComboBoxEx.
	WMLButtonDown (see page 3698)	Responds when the user clicks the left mouse button on the combo box.
V	WndProc (see page 3698)	Processes messages received by the combo box.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WndProc (see page 21183)	Processes messages received by the combo box.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownCount (see page 21185)	Specifies the maximum number of items displayed in the drop-down list.
R	EditHandle (see page 21185)	Returns the window handle for the edit portion of the combo box.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	ItemCount (see page 21186)	Specifies the number of items in the drop-down list.
	ItemHeight (see page 21186)	Specifies the height, in pixels, of the items in the drop-down list.
R	ListHandle (see page 21189)	Provides access to the window handle of the list of items in the combo box.
	MaxLength (see page 21189)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 21190)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnBeginEdit (see page 3702)	Occurs when the user starts editing the text of the combo box.
	OnEndEdit (see page 3702)	Occurs when the user finishes editing the text of the combo box.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
A	AddItem (see page 5940)	Adds an item to the list control.
V	AddItem (see page 21170)	Adds an item to the drop-down list of items.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 5940)	Deletes all of the items from the list control.
V	Clear (see page 21172)	Deletes all text from the edit box and all items from the Items (see page 21187) list.
A	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
V	ClearSelection (see page 21172)	Deselects any selected items in the drop-down list.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
A	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
V	CopySelection (see page 21173)	Copies the selected item in the drop-down list to another list control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 3691)	Creates and initializes a TCustomComboBoxEx (see page 3678) instance.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 21166)	Creates and initializes a TCustomCombo (see page 21154) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	DeleteSelected (see page 5941)	Deletes all the selected items.
V	DeleteSelected (see page 21174)	Deletes the selected item from the drop-down list.
V	Destroy (see page 2766)	Disposes of the component and its owned components.

	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 3691)	Destroys the combo box and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 21166)	Destroys the combo box and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 132)	OnDragDrop event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 3695)	Indicates whether the combo box has input focus.
	Focused (see page 6150)	Determines whether the control has input focus.
	Focused (see page 21175)	Indicates whether the combo box has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
A	SelectAll (see page 5943)	Selects all items or all text in the selected item.
V	SelectAll (see page 21180)	Selects the entire block of text in the edit box.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.

V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AutoCompleteOptions (see page 3699)	Specifies options that control autocompletion.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 21184)	Provides access to the drawing surface of the combo box.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropDownCount (see page 3699)	Specifies the maximum number of items displayed in the drop-down list.
	DroppedDown (see page 21185)	Indicates whether the drop-down list is currently displayed.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.

	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	Images (see page 3700)	Specifies the images that appear next to items in the combo box list.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	Items (see page 21187)	Provides access to the list of items (strings) in the list portion of the combo box.
	ItemsEx (see page 3700)	Lists the items that appear in the list portion of the combo box.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelLength (see page 21190)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 21193)	Specifies the position of the first selected character in the edit portion of the combo box.
	SelText (see page 3701)	Represents the selected text in the edit region.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	Style (see page 3701)	Determines the display style of the combo box.
	StyleEx (see page 3701)	Determines the appearance and behavior of the items in the list.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TabStop (see page 21196)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

2.1.2.2 TCustomActionCombo Methods

The methods of the TCustomActionCombo class are listed here.

Protected Methods

	Name	Description
	BeginAutoDrag (see page 131)	Begin Automatic Drag.
	ComboWndProc (see page 131)	This is ComboWndProc, a member of class TCustomActionCombo.
	DesignWndProc (see page 132)	This is DesignWndProc, a member of class TCustomActionCombo.
	DragOver (see page 132)	OnDragOver event dispatcher.

Public Methods

	Name	Description
	DragDrop (see page 132)	OnDragDrop event dispatcher.

2.1.2.2.1 TCustomActionCombo.BeginAutoDrag Method

Begin Automatic Drag.

Pascal

```
procedure BeginAutoDrag; override;
```

C++

```
virtual __fastcall BeginAutoDrag();
```

Description

BeginAutoDrag is called automatically when a user drags a control whose DragMode is dmAutomatic. Do not call BeginAutoDrag in application code.

As implemented in TControl, BeginAutoDrag calls BeginDrag, passing in the global Mouse variable's DragImmediate and DragThreshold properties.

See Also

DragKind (see page 5819)

DragMode (see page 5819)

EndDrag (see page 5757)

Mouse (see page 6321)

DragImmediate (see page 6100)

DragThreshold (see page 6101)

Mouse (see page 6321)

2.1.2.2.2 TCustomActionCombo.ComboWndProc Method

Pascal

```
procedure ComboWndProc(var Message: TMessage; ComboWnd: HWND; ComboProc: Pointer); override;
```

C++

```
virtual __fastcall ComboWndProc(TMessage Message, HWND ComboWnd, void * ComboProc);
```

Description

This is ComboWndProc, a member of class TCustomActionCombo.

2.1.2.2.3 TCustomActionCombo.DesignWndProc Method

Pascal

```
function DesignWndProc(var Message: TMessage): Boolean; override;
```

C++

```
virtual __fastcall Boolean DesignWndProc(TMessage Message);
```

Description

This is DesignWndProc, a member of class TCustomActionCombo.

2.1.2.2.4 TCustomActionCombo.DragDrop Method

OnDragDrop event dispatcher.

Pascal

```
procedure DragDrop(Source: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DragDrop(TObject * Source, int X, int Y);
```

Description

When creating a component that descends from TControl, override DragDrop to add additional code that executes before the OnDragDrop event handler is called.

The Source parameter is the object that was dropped onto the control. The X and Y parameters are the mouse coordinates where the object was dropped.

See Also

DragCanceled (see page 5754)

OnDragDrop (see page 5855)

2.1.2.2.5 TCustomActionCombo.DragOver Method

OnDragOver event dispatcher.

Pascal

```
procedure DragOver(Source: TObject; X: Integer; Y: Integer; State: TDragState; var Accept:
Boolean); override;
```

C++

```
virtual __fastcall DragOver(TObject * Source, int X, int Y, TDragState State, Boolean Accept);
```

Description

Override DragOver to add additional code that executes before the OnDragOver event handler is called.

DragOver sets the Accept parameter to true to indicate that the user can drop the dragged object on the control. It sets Accept to false to indicate that the user cannot drop the dragged object on the control.

The Source parameter is the object being dragged. The State parameter indicates how the dragged object is moving in relation

to the control. X and Y indicate the current position of the mouse.

See Also

DragDrop (see page 5754)

OnDragOver (see page 5857)

2.1.3 TCustomActionDockBar Class

TCustomActionDockBar is the ancestor for action bands that require the ability to dock.

Class Hierarchy

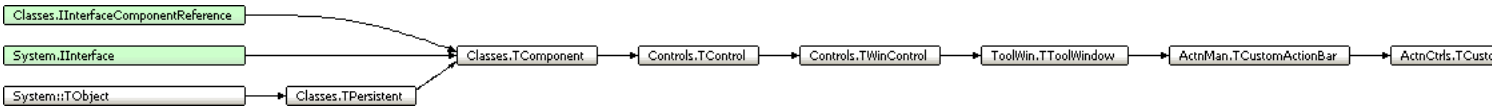

Pascal

```
TCustomActionDockBar = class(TCustomActionBar);
```

C++

```
class TCustomActionDockBar : public TCustomActionBar;
```

File

ActnCtrls

Description

TCustomActionDockBar is the ancestor for all action bands that require the ability to dock.

See Also

TActionToolBar (see page 38)

TActionMainMenuBar (see page 585)

2.1.3.1 TCustomActionDockBar Members

The following tables list the members exposed by TCustomActionDockBar.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.

	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefaultColorMap (see page 460)	This is FDefaultColorMap, a member of class TCustomActionBar.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSavedWrapState (see page 460)	This is FSavedWrapState, a member of class TCustomActionBar.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.

	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 462)	Aligns any controls for which the control is the parent within a specified area of the control.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	AutoSizingChanged (see page 462)	This is AutoSizingChanged, a member of class TCustomActionBar.
	BackgroundSize (see page 462)	This is BackgroundSize, a member of class TCustomActionBar.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcInsertPosition (see page 462)	This is CalcInsertPosition, a member of class TCustomActionBar.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Clear (see page 463)	This is Clear, a member of class TCustomActionBar.
	Click (see page 5743)	Respond to user click.
	CMBorderChanged (see page 23285)	This is CMBorderChanged, a member of class TToolWindow.
	CMCtl3DChanged (see page 23286)	This is CMCtl3DChanged, a member of class TToolWindow.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateControl (see page 463)	This is CreateControl, a member of class TCustomActionBar.
	CreateControls (see page 463)	This is CreateControls, a member of class TCustomActionBar.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 463)	Returns placement order of custom-aligned child control.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 464)	Returns aligned position of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.

	DesignWndProc (see page 465)	This is DesignWndProc, a member of class TCustomActionBar.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoControlCreated (see page 465)	This is DoControlCreated, a member of class TCustomActionBar.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoDropActions (see page 465)	This is DoDropActions, a member of class TCustomActionBar.
	DoDropCategory (see page 465)	This is DoDropCategory, a member of class TCustomActionBar.
	DoDropItem (see page 466)	This is DoDropItem, a member of class TCustomActionBar.
	DoEndDock (see page 146)	OnEndDock event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetControlClass (see page 466)	This is DoGetControlClass, a member of class TCustomActionBar.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 147)	OnStartDock event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 466)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawBackground (see page 467)	This is DrawBackground, a member of class TCustomActionBar.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindAccelItem (see page 467)	This is FindAccelItem, a member of class TCustomActionBar.
	FindItemAt (see page 468)	This is FindItemAt, a member of class TCustomActionBar.
	FindItemCaption (see page 468)	This is FindItemCaption, a member of class TCustomActionBar.
	FindLast (see page 468)	This is FindLast, a member of class TCustomActionBar.
	FindLeastUsedItem (see page 469)	This is FindLeastUsedItem, a member of class TCustomActionBar.
	FindNearestControl (see page 469)	This is FindNearestControl, a member of class TCustomActionBar.
	FindNext (see page 470)	This is FindNext, a member of class TCustomActionBar.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FindNextVisibleItem (see page 470)	This is FindNextVisibleItem, a member of class TCustomActionBar.
	FindPrevious (see page 470)	This is FindPrevious, a member of class TCustomActionBar.
	FindPreviousVisibleItem (see page 470)	This is FindPreviousVisibleItem, a member of class TCustomActionBar.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionControl (see page 471)	This is GetActionControl, a member of class TCustomActionBar.

	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBannerWidth (see page 471)	This is GetBannerWidth, a member of class TCustomActionBar.
	GetBarHeight (see page 471)	This is GetBarHeight, a member of class TCustomActionBar.
	GetBarWidth (see page 471)	This is GetBarWidth, a member of class TCustomActionBar.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlClass (see page 471)	This is GetControlClass, a member of class TCustomActionBar.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDefaultColorMapClass (see page 472)	This is GetDefaultColorMapClass, a member of class TCustomActionBar.
	GetDesignMode (see page 472)	This is GetDesignMode, a member of class TCustomActionBar.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 147)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetPopupMenuClass (see page 472)	This is GetPopupMenuClass, a member of class TCustomActionBar.
	GetSiteInfo (see page 147)	Generates an OnGetSiteInfo event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasItems (see page 472)	This is HasItems, a member of class TCustomActionBar.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	ItemCount (see page 473)	This is ItemCount, a member of class TCustomActionBar.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 148)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.

	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	NCPaint (see page 473)	This is NCPaint, a member of class TCustomActionBar.
V	NCPaint (see page 23286)	This is NCPaint, a member of class TToolWindow.
V	Notification (see page 473)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 474)	This is Paint, a member of class TCustomActionBar.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 474)	Renders the image of a windowed control.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
V	Reset (see page 475)	This is Reset, a member of class TCustomActionBar.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
V	SetActionClient (see page 475)	This is SetActionClient, a member of class TCustomActionBar.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetAutoSizing (see page 475)	This is SetAutoSizing, a member of class TCustomActionBar.
V	SetBiDiMode (see page 475)	Sets the control's bi-directional mode.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
V	SetColorMap (see page 476)	This is SetColorMap, a member of class TCustomActionBar.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDesignMode (see page 476)	This is SetDesignMode, a member of class TCustomActionBar.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.

	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetOrientation (see page 477)	This is SetOrientation, a member of class TCustomActionBar.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetPersistentHotKeys (see page 477)	This is SetPersistentHotKeys, a member of class TCustomActionBar.
	SetSpacing (see page 477)	This is SetSpacing, a member of class TCustomActionBar.
	SetupDefaultColorMap (see page 477)	This is SetupDefaultColorMap, a member of class TCustomActionBar.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 478)	Provides the interface for a method that responds to changes in the control's visibility.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMNCCalcSize (see page 23286)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCPaint (see page 23286)	This is WMNCPaint, a member of class TToolWindow.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.

	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
R	HRowCount (see page 488)	Indicates the number of columns of items that the action band displays.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	Items (see page 488)	Lists the action clients that correspond to the child controls of this action band.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
R	VRowCount (see page 490)	Indicates the number of rows of items that the action band displays.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
R	Widest (see page 491)	This is Widest, a member of class TCustomActionBar.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnControlCreated (see page 491)	Occurs when a control is created on the action band.
	OnGetControlClass (see page 492)	Occurs when an item is to be placed on the action band.
	OnPaint (see page 492)	Occurs when the action band is painted.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.

	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CalcDockedWidth (see page 146)	This is CalcDockedWidth, a member of class TCustomActionDockBar.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 145)	Creates and initializes a TCustomActionDockBar (see page 133) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 459)	Destroys an instance of TCustomActionBar (see page 447).
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 146)	Used internally to dock the action band.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 466)	Generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindFirst (see page 467)	Finds the first TActionClientItem (see page 374) in the Items (see page 488) list.
	FindFirstVisibleItem (see page 468)	Finds the first visible TActionClientItem (see page 374) in the Items (see page 488) list.
	FindLastVisibleItem (see page 469)	Finds the last visibleTActionClientItem in the Items (see page 488) list.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
V	RecreateControls (see page 475)	Destroys and recreates all of the action band's action client controls.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 476)	Sets the bounds based on whether the popup is expanded or not
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Style (see page 477)	Identifies the style object that gives the application its default look and feel.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	ActionClient (see page 479)	Specifies the action client associated with this action band.
R	ActionControls (see page 479)	Lists the controls on the action band.
	ActionManager (see page 480)	Specifies the action manager component that is associated with the action band.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AllowHiding (see page 480)	Determines whether the action band can be hidden.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AutoSizing (see page 480)	Specifies whether the action band adjusts its size automatically to fit around its child controls.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 481)	Represents the drawing surface of the action band.
	Caption (see page 481)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 483)	Specifies the background color of the control.

	ColorMap (see page 484)	Indicates the colormap for individual action band components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ContextBar (see page 485)	Indicates whether items shown in this action band represent the items in a popup menu.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designable (see page 485)	Specifies whether the action band can be customized (used internally by the IDE).
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignMode (see page 486)	Specifies whether the action band is being customized or being used in the IDE.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EdgeBorders (see page 23287)	Determines which sides of the control have visible borders.
	EdgeInner (see page 23287)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 23288)	Determines the border style for the control's outer edge.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Font (see page 486)	Controls the attributes of text written on or in the control.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HorzMargin (see page 487)	Specifies the width of the horizontal margins in pixels.
	HorzSeparator (see page 487)	Indicates whether a horizontal separator bar appears between item lines.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	Orientation (see page 488)	Specifies how the action band lays out its child items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PersistentHotKeys (see page 489)	Specifies whether keyboard accelerator shortcuts are always displayed.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.

	Spacing (see page 489)	Determines where the image and text appear on an action band's rendered control that represents the action client.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	VertMargin (see page 490)	Specifies the width of the vertical margins in pixels.
	VertSeparator (see page 490)	Indicates whether a vertical separator bar should appear between item lines.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

2.1.3.2 TCustomActionDockBar.Create Constructor

Creates and initializes a TCustomActionDockBar (see page 133) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomActionDockBar(TComponent * AOwner);
```

Description

Create generates a TCustomActionDockBar (see page 133) and initializes its properties.

```
__fastcall virtual TCustomActionDockBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall (HWND ParentWindow) : Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

2.1.3.3 TCustomActionDockBar Methods

The methods of the TCustomActionDockBar class are listed here.

Protected Methods

	Name	Description
	DoEndDock (see page 146)	OnEndDock event dispatcher.
	DoStartDock (see page 147)	OnStartDock event dispatcher.
	GetFloatingDockSiteClass (see page 147)	Returns the class used for the control when it is floating.
	GetSiteInfo (see page 147)	Generates an OnGetSiteInfo event
	Loaded (see page 148)	Initializes the control after it is loaded from a stream.

Public Methods

	Name	Description
	CalcDockedWidth (see page 146)	This is CalcDockedWidth, a member of class TCustomActionDockBar.
	Dock (see page 146)	Used internally to dock the action band.

2.1.3.3.1 TCustomActionDockBar.CalcDockedWidth Method

Pascal

```
function CalcDockedWidth: Integer; virtual;
```

C++

```
virtual __fastcall int CalcDockedWidth();
```

Description

This is CalcDockedWidth, a member of class TCustomActionDockBar.

2.1.3.3.2 TCustomActionDockBar.Dock Method

Used internally to dock the action band.

Pascal

```
procedure Dock(NewDockSite: TWinControl; ARect: TRect); override;
```

C++

```
virtual __fastcall Dock(TWinControl * NewDockSite, TRect ARect);
```

Description

Do not call Dock. It is called automatically to implement many of the details of docking the action band to a dock site.

To dock the action band programmatically, use ManualDock instead.

The NewDockSite parameter specifies the docking site to which the action band is being docked. The ARect parameter indicates the new dimensions of the action band when it is docked to NewDockSite.

See Also

ManualDock (see page 5771)

2.1.3.3.3 TCustomActionDockBar.DoEndDock Method

OnEndDock event dispatcher.

Pascal

```
procedure DoEndDock(Target: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DoEndDock(TObject * Target, int X, int Y);
```

Description

OnEndDock calls any event handler attached to the OnEndDock event. When creating a component that descends from TControl, override OnEndDock to provide other responses in addition to the inherited event-handler call when the user finishes a drag-and-dock operation.

See Also

DoStartDock (see page 5753)

OnEndDock (see page 5859)

2.1.3.3.4 TCustomActionDockBar.DoStartDock Method

OnStartDock event dispatcher.

Pascal

```
procedure DoStartDock(var DragObject: TDragObject); override;
```

C++

```
virtual __fastcall DoStartDock(TDragObject DragObject);
```

Description

DoStartDock calls any event handler attached to the OnStartDock event. When creating a component that descends from TControl, override DoStartDock to provide other responses in addition to the inherited event-handler call when the user starts a drag-and-dock operation.

See Also

DoEndDock (see page 5750)

OnStartDock (see page 5871)

2.1.3.3.5 TCustomActionDockBar.GetFloatingDockSiteClass Method

Returns the class used for the control when it is floating.

Pascal

```
function GetFloatingDockSiteClass: TWinControlClass; override;
```

C++

```
virtual __fastcall TWinControlClass GetFloatingDockSiteClass();
```

Description

GetFloatingDockSiteClass is the read implementation of the FloatingDockSiteClass property. Override this method to change the implementation of FloatingDockSiteClass.

See Also

FloatingDockSiteClass (see page 5823)

2.1.3.3.6 TCustomActionDockBar.GetSiteInfo Method

Generates an OnGetSiteInfo event

Pascal

```
procedure GetSiteInfo(Client: TControl; var InfluenceRect: TRect; MousePos: TPoint; var
```

```
CanDock: Boolean); override;
```

C++

```
virtual __fastcall GetSiteInfo(TControl * Client, TRect InfluenceRect, TPoint MousePos,
Boolean CanDock);
```

Description

GetSiteInfo is called automatically during drag-and-dock operations to initialize a TDragDockObject with information about where a dragged object can be docked, if at all.

Client is a potential docking client.

InfluenceRect returns the area in which the client would be docked.

MousePos is the current mouse position.

CanDock returns whether the client can dock to the windowed control.

As implemented in TWinControl, GetSiteInfo generates an influence rectangle equal to BoundsRect expanded by 10 pixels on every side, and then calls the OnGetSiteInfo event handler.

See Also

OnGetSiteInfo (see page 6223)

BoundsRect (see page 5803)

2.1.3.3.7 TCustomActionDockBar.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded (see page 2782)

Action (see page 5797)

ActionChange (see page 5737)

2.1.4 TCustomActionToolBar Class

TCustomActionToolBar is the base class for the TActionToolBar (see page 38) component.

Class Hierarchy

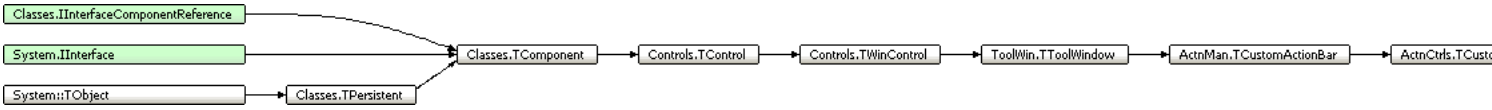

Pascal

```
TCustomActionToolBar = class(TCustomActionDockBar);
```

C++

```
class TCustomActionToolBar : public TCustomActionDockBar;
```

File

ActnCtrls

Description

TCustomActionToolBar is the base class for components that generate tool bars based on the actions that an Action Manager specifies. TCustomActionToolBar descendants generate internal components for the controls in the tool bar, where each component represents an action in the Action manager. The layout of these items is managed automatically, and specified by the Action Manager.

See Also

TCoolBar (see page 3603)

TActionToolBar (see page 38)

TActionManager (see page 432)

2.1.4.1 TCustomActionToolBar Members

The following tables list the members exposed by TCustomActionToolBar.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.

	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefaultColorMap (see page 460)	This is FDefaultColorMap, a member of class TCustomActionBar.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSavedWrapState (see page 460)	This is FSavedWrapState, a member of class TCustomActionBar.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 462)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.

	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	AutoSizingChanged (see page 163)	This is AutoSizingChanged, a member of class TCustomActionToolBar.
	BackgroundSize (see page 462)	This is BackgroundSize, a member of class TCustomActionBar.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcButtonWidth (see page 163)	This is CalcButtonWidth, a member of class TCustomActionToolBar.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcInsertPosition (see page 462)	This is CalcInsertPosition, a member of class TCustomActionBar.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Clear (see page 463)	This is Clear, a member of class TCustomActionBar.
	Click (see page 5743)	Respond to user click.
	CMBorderChanged (see page 23285)	This is CMBorderChanged, a member of class TToolWindow.
	CMColorchanged (see page 163)	This is CMColorchanged, a member of class TCustomActionToolBar.
	CMCtl3DChanged (see page 23286)	This is CMCtl3DChanged, a member of class TToolWindow.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateControl (see page 163)	This is CreateControl, a member of class TCustomActionToolBar.
	CreateControls (see page 164)	This is CreateControls, a member of class TCustomActionToolBar.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 463)	Returns placement order of custom-aligned child control.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 464)	Returns aligned position of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 465)	This is DesignWndProc, a member of class TCustomActionBar.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DisableHiding (see page 164)	This is DisableHiding, a member of class TCustomActionToolBar.

	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
V	DoControlCreated (see page 465)	This is DoControlCreated, a member of class TCustomActionBar.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
V	DoDropActions (see page 465)	This is DoDropActions, a member of class TCustomActionBar.
V	DoDropCategory (see page 164)	This is DoDropCategory, a member of class TCustomActionToolBar.
V	DoDropItem (see page 466)	This is DoDropItem, a member of class TCustomActionBar.
V	DoEndDock (see page 146)	OnEndDock event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetControlClass (see page 466)	This is DoGetControlClass, a member of class TCustomActionBar.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
V	DoStartDock (see page 147)	OnStartDock event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 466)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawBackground (see page 165)	This is DrawBackground, a member of class TCustomActionToolBar.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawSeparator (see page 165)	This is DrawSeparator, a member of class TCustomActionToolBar.
	EnableHiding (see page 165)	This is EnableHiding, a member of class TCustomActionToolBar.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindAccelItem (see page 467)	This is FindAccelItem, a member of class TCustomActionBar.
	FindItemAt (see page 468)	This is FindItemAt, a member of class TCustomActionBar.
	FindItemCaption (see page 468)	This is FindItemCaption, a member of class TCustomActionBar.
V	FindLast (see page 468)	This is FindLast, a member of class TCustomActionBar.
	FindLeastUsedItem (see page 469)	This is FindLeastUsedItem, a member of class TCustomActionBar.
	FindNearestControl (see page 469)	This is FindNearestControl, a member of class TCustomActionBar.
V	FindNext (see page 470)	This is FindNext, a member of class TCustomActionBar.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FindNextVisibleItem (see page 470)	This is FindNextVisibleItem, a member of class TCustomActionBar.
V	FindPrevious (see page 470)	This is FindPrevious, a member of class TCustomActionBar.
	FindPreviousVisibleItem (see page 470)	This is FindPreviousVisibleItem, a member of class TCustomActionBar.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
V	GetActionControl (see page 471)	This is GetActionControl, a member of class TCustomActionBar.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.

	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBannerWidth (see page 471)	This is GetBannerWidth, a member of class TCustomActionBar.
	GetBarHeight (see page 471)	This is GetBarHeight, a member of class TCustomActionBar.
	GetBarWidth (see page 471)	This is GetBarWidth, a member of class TCustomActionBar.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlClass (see page 165)	This is GetControlClass, a member of class TCustomActionToolBar.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDefaultColorMapClass (see page 472)	This is GetDefaultColorMapClass, a member of class TCustomActionBar.
	GetDesignMode (see page 472)	This is GetDesignMode, a member of class TCustomActionBar.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 147)	Returns the class used for the control when it is floating.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupClass (see page 166)	This is GetPopupClass, a member of class TCustomActionToolBar.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetPopupMenuClass (see page 472)	This is GetPopupMenuClass, a member of class TCustomActionBar.
	GetScrollBtnClass (see page 166)	This is GetScrollBtnClass, a member of class TCustomActionToolBar.
	GetSiteInfo (see page 147)	Generates an OnGetSiteInfo event
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasItems (see page 472)	This is HasItems, a member of class TCustomActionBar.
	HideUnusedItems (see page 166)	This is HideUnusedItems, a member of class TCustomActionToolBar.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	ItemCount (see page 473)	This is ItemCount, a member of class TCustomActionBar.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.

	Loaded (see page 148)	Initializes the control after it is loaded from a stream.
	Loaded (see page 473)	Initializes the control after it is loaded from a stream.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	NCPaint (see page 473)	This is NCPaint, a member of class TCustomActionBar.
	NCPaint (see page 23286)	This is NCPaint, a member of class TToolWindow.
	Notification (see page 473)	Responds to notifications that components are being created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 474)	This is Paint, a member of class TCustomActionBar.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 474)	Renders the image of a windowed control.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Reset (see page 166)	This is Reset, a member of class TCustomActionToolBar.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	ScrollBtnClick (see page 166)	This is ScrollBtnClick, a member of class TCustomActionToolBar.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActionClient (see page 475)	This is SetActionClient, a member of class TCustomActionBar.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetAutoSizing (see page 475)	This is SetAutoSizing, a member of class TCustomActionBar.
	SetBiDiMode (see page 475)	Sets the control's bi-directional mode.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.

	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetColorMap (see page 476)	This is SetColorMap, a member of class TCustomActionBar.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignMode (see page 476)	This is SetDesignMode, a member of class TCustomActionBar.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetOrientation (see page 167)	This is SetOrientation, a member of class TCustomActionToolBar.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetPersistentHotKeys (see page 477)	This is SetPersistentHotKeys, a member of class TCustomActionBar.
	SetSpacing (see page 477)	This is SetSpacing, a member of class TCustomActionBar.
	SetupDefaultColorMap (see page 477)	This is SetupDefaultColorMap, a member of class TCustomActionBar.
	SetupDropDownBtn (see page 167)	This is SetupDropDownBtn, a member of class TCustomActionToolBar.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 478)	Provides the interface for a method that responds to changes in the control's visibility.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMNCCalcSize (see page 23286)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCPaint (see page 23286)	This is WMNCPaint, a member of class TToolWindow.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.

	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
R	HRowCount (see page 488)	Indicates the number of columns of items that the action band displays.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	Items (see page 488)	Lists the action clients that correspond to the child controls of this action band.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
R	VRowCount (see page 490)	Indicates the number of rows of items that the action band displays.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
R	Widest (see page 491)	This is Widest, a member of class TCustomActionBar.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnControlCreated (see page 491)	Occurs when a control is created on the action band.
	OnGetControlClass (see page 492)	Occurs when an item is to be placed on the action band.
	OnPaint (see page 492)	Occurs when the action band is painted.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
V	CalcDockedWidth (see page 163)	This is CalcDockedWidth, a member of class TCustomActionToolBar.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 161)	Creates and initializes a TCustomActionToolBar (see page 148) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 162)	Destroys an instance of TCustomActionToolBar (see page 148).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Dock (see page 146)	Used internally to dock the action band.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
V	DragDrop (see page 164)	Generates an OnDragDrop event.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.

	FindFirst (see page 467)	Finds the first TActionClientItem (see page 374) in the Items (see page 488) list.
	FindFirstVisibleItem (see page 468)	Finds the first visible TActionClientItem (see page 374) in the Items (see page 488) list.
	FindLastVisibleItem (see page 469)	Finds the last visibleTActionClientItem in the Items (see page 488) list.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	Realign (see page 6171)	Forces the control to realign children.
V	RecreateControls (see page 475)	Destroys and recreates all of the action band's action client controls.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 167)	Sets the Left, Top, Width, and Height properties in a single operation.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Style (see page 477)	Identifies the style object that gives the application its default look and feel.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	ActionClient (see page 479)	Specifies the action client associated with this action band.
R	ActionControls (see page 479)	Lists the controls on the action band.
	ActionManager (see page 480)	Specifies the action manager component that is associated with the action band.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AllowHiding (see page 480)	Determines whether the action band can be hidden.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AutoSizing (see page 480)	Specifies whether the action band adjusts its size automatically to fit around its child controls.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 481)	Represents the drawing surface of the action band.
	Caption (see page 481)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.

R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 483)	Specifies the background color of the control.
	ColorMap (see page 484)	Indicates the colormap for individual action band components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ContextBar (see page 485)	Indicates whether items shown in this action band represent the items in a popup menu.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designable (see page 485)	Specifies whether the action band can be customized (used internally by the IDE).
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignMode (see page 486)	Specifies whether the action band is being customized or being used in the IDE.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EdgeBorders (see page 23287)	Determines which sides of the control have visible borders.
	EdgeInner (see page 23287)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 23288)	Determines the border style for the control's outer edge.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Font (see page 486)	Controls the attributes of text written on or in the control.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
R	HiddenCount (see page 168)	Indicates the number of hidden items on the tool action band.
	HorzMargin (see page 487)	Specifies the width of the horizontal margins in pixels.
	HorzSeparator (see page 487)	Indicates whether a horizontal separator bar appears between item lines.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	Orientation (see page 488)	Specifies how the action band lays out its child items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.

	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PersistentHotKeys (see page 489)	Specifies whether keyboard accelerator shortcuts are always displayed.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	Spacing (see page 489)	Determines where the image and text appear on an action band's rendered control that represents the action client.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	VertMargin (see page 490)	Specifies the width of the vertical margins in pixels.
	VertSeparator (see page 490)	Indicates whether a vertical separator bar should appear between item lines.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

2.1.4.2 TCustomActionToolBar.Create Constructor

Creates and initializes a TCustomActionToolBar (see page 148) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomActionToolBar(TComponent * AOwner);
```

Description

Create generates a TCustomActionToolBar (see page 148) and initializes its properties.

```
__fastcall virtual TCustomActionToolBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall (HWND ParentWindow) : Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

2.1.4.3 TCustomActionToolBar.Destroy Destructor

Destroys an instance of TCustomActionToolBar (see page 148).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomActionToolBar();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TCustomActionToolBar (see page 148) reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

Destroy (see page 5732)

2.1.4.4 TCustomActionToolBar Methods

The methods of the TCustomActionToolBar class are listed here.

Protected Methods

	Name	Description
V	AutoSizingChanged (see page 163)	This is AutoSizingChanged, a member of class TCustomActionToolBar.
	CalcButtonWidth (see page 163)	This is CalcButtonWidth, a member of class TCustomActionToolBar.
	CMColorchanged (see page 163)	This is CMColorchanged, a member of class TCustomActionToolBar.
V	CreateControl (see page 163)	This is CreateControl, a member of class TCustomActionToolBar.
V	CreateControls (see page 164)	This is CreateControls, a member of class TCustomActionToolBar.
	DisableHiding (see page 164)	This is DisableHiding, a member of class TCustomActionToolBar.
V	DoDropCategory (see page 164)	This is DoDropCategory, a member of class TCustomActionToolBar.
V	DrawBackground (see page 165)	This is DrawBackground, a member of class TCustomActionToolBar.
V	DrawSeparator (see page 165)	This is DrawSeparator, a member of class TCustomActionToolBar.
	EnableHiding (see page 165)	This is EnableHiding, a member of class TCustomActionToolBar.
V	GetControlClass (see page 165)	This is GetControlClass, a member of class TCustomActionToolBar.
V	GetPopupClass (see page 166)	This is GetPopupClass, a member of class TCustomActionToolBar.
	GetScrollBtnClass (see page 166)	This is GetScrollBtnClass, a member of class TCustomActionToolBar.
	HideUnusedItems (see page 166)	This is HideUnusedItems, a member of class TCustomActionToolBar.
V	Reset (see page 166)	This is Reset, a member of class TCustomActionToolBar.
	ScrollBtnClick (see page 166)	This is ScrollBtnClick, a member of class TCustomActionToolBar.
V	SetOrientation (see page 167)	This is SetOrientation, a member of class TCustomActionToolBar.
	SetupDropDownBtn (see page 167)	This is SetupDropDownBtn, a member of class TCustomActionToolBar.

Public Methods

	Name	Description
V	CalcDockedWidth (see page 163)	This is CalcDockedWidth, a member of class TCustomActionToolBar.
V	DragDrop (see page 164)	Generates an OnDragDrop event.
V	SetBounds (see page 167)	Sets the Left, Top, Width, and Height properties in a single operation.

2.1.4.4.1 TCustomActionToolBar.AutoSizingChanged Method

Pascal

```
procedure AutoSizingChanged; override;
```

C++

```
virtual __fastcall AutoSizingChanged();
```

Description

This is AutoSizingChanged, a member of class TCustomActionToolBar.

2.1.4.4.2 TCustomActionToolBar.CalcButtonWidth Method

Pascal

```
function CalcButtonWidth: Integer;
```

C++

```
__fastcall int CalcButtonWidth();
```

Description

This is CalcButtonWidth, a member of class TCustomActionToolBar.

2.1.4.4.3 TCustomActionToolBar.CalcDockedWidth Method

Pascal

```
function CalcDockedWidth: Integer; override;
```

C++

```
virtual __fastcall int CalcDockedWidth();
```

Description

This is CalcDockedWidth, a member of class TCustomActionToolBar.

2.1.4.4.4 TCustomActionToolBar.CMColorchanged Method

Pascal

```
procedure CMColorchanged(var Message: TMessage); message CM_COLORCHANGED;
```

C++

```
__fastcall CMColorchanged(TMessage Message);
```

Description

This is CMColorchanged, a member of class TCustomActionToolBar.

2.1.4.4.5 TCustomActionToolBar.CreateControl Method

Pascal

```
function CreateControl(AnItem: TActionClientItem): TCustomActionControl; override;
```

C++

```
virtual __fastcall TCustomActionControl CreateControl(TActionClientItem AnItem);
```

Description

This is CreateControl, a member of class TCustomActionToolBar.

2.1.4.4.6 TCustomActionToolBar.CreateControls Method

Pascal

```
procedure CreateControls; override;
```

C++

```
virtual __fastcall CreateControls();
```

Description

This is CreateControls, a member of class TCustomActionToolBar.

2.1.4.4.7 TCustomActionToolBar.DisableHiding Method

Pascal

```
procedure DisableHiding;
```

C++

```
__fastcall DisableHiding();
```

Description

This is DisableHiding, a member of class TCustomActionToolBar.

2.1.4.4.8 TCustomActionToolBar.DoDropCategory Method

Pascal

```
procedure DoDropCategory(Source: TCategoryDragObject; const X: Integer; const Y: Integer);
override;
```

C++

```
virtual __fastcall DoDropCategory(TCategoryDragObject Source, const int X, const int Y);
```

Description

This is DoDropCategory, a member of class TCustomActionToolBar.

2.1.4.4.9 TCustomActionToolBar.DragDrop Method

Generates an OnDragDrop event.

Pascal

```
procedure DragDrop(Source: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DragDrop(TObject * Source, int X, int Y);
```

Description

DragDrop is called automatically when the user drags and drops a control onto the tool bar. This method generates an OnDragDrop event, and then updates its list of hidden controls to reflect the presence of the new child control.

Source is the dragged control that is dropped onto the tool bar.

X and Y are the mouse coordinates where Source was dropped.

See Also

DragCanceled (see page 5754)

OnDragDrop (see page 5855)

HiddenCount (see page 168)

2.1.4.4.10 TCustomActionToolBar.DrawBackground Method

Pascal

```
procedure DrawBackground; override;
```

C++

```
virtual __fastcall DrawBackground();
```

Description

This is DrawBackground, a member of class TCustomActionToolBar.

2.1.4.4.11 TCustomActionToolBar.DrawSeparator Method

Pascal

```
procedure DrawSeparator(const Pos: Integer; const Offset: Integer); virtual;
```

C++

```
virtual __fastcall DrawSeparator(const int Pos, const int Offset);
```

Description

This is DrawSeparator, a member of class TCustomActionToolBar.

2.1.4.4.12 TCustomActionToolBar.EnableHiding Method

Pascal

```
procedure EnableHiding;
```

C++

```
__fastcall EnableHiding();
```

Description

This is EnableHiding, a member of class TCustomActionToolBar.

2.1.4.4.13 TCustomActionToolBar.GetControlClass Method

Pascal

```
function GetControlClass(AnItem: TActionClientItem): TCustomActionControlClass; override;
```

C++

```
virtual __fastcall TCustomActionControlClass GetControlClass(TActionClientItem AnItem);
```

Description

This is GetControlClass, a member of class TCustomActionToolBar.

2.1.4.4.14 TCustomActionToolBar.GetPopupClass Method

Pascal

```
function GetPopupClass: TCustomActionBarClass; virtual;
```

C++

```
virtual __fastcall TCustomActionBarClass GetPopupClass();
```

Description

This is GetPopupClass, a member of class TCustomActionToolBar.

2.1.4.4.15 TCustomActionToolBar.GetScrollBtnClass Method

Pascal

```
function GetScrollBtnClass: TCustomToolScrollBtnClass;
```

C++

```
__fastcall TCustomToolScrollBtnClass GetScrollBtnClass();
```

Description

This is GetScrollBtnClass, a member of class TCustomActionToolBar.

2.1.4.4.16 TCustomActionToolBar.HideUnusedItems Method

Pascal

```
procedure HideUnusedItems;
```

C++

```
__fastcall HideUnusedItems();
```

Description

This is HideUnusedItems, a member of class TCustomActionToolBar.

2.1.4.4.17 TCustomActionToolBar.Reset Method

Pascal

```
procedure Reset; override;
```

C++

```
virtual __fastcall Reset();
```

Description

This is Reset, a member of class TCustomActionToolBar.

2.1.4.4.18 TCustomActionToolBar.ScrollBtnClick Method

Pascal

```
procedure ScrollBtnClick(Sender: TObject);
```

C++

```
__fastcall ScrollBtnClick(TObject * Sender);
```

Description

This is ScrollBtnClick, a member of class TCustomActionToolBar.

2.1.4.4.19 TCustomActionToolBar.SetBounds Method

Sets the Left, Top, Width, and Height properties in a single operation.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to change all of the component's boundary properties at one time. The same effect can be achieved by setting the Left, Top, Width, and Height properties separately, but SetBounds changes all four properties at once, ensuring that the tool bar will not repaint between changes.

Specify the values for the Left, Top, Width, and Height properties as the value of the ALeft, ATop, AWidth, and AHeight parameters, respectively.

SetBounds overrides the inherited method to manage the hidden controls properly when the toolbar is resized.

See Also

SetBounds (see page 6178)

Left (see page 5830)

Top (see page 5844)

Width (see page 5849)

Height (see page 5825)

HiddenCount (see page 168)

2.1.4.4.20 TCustomActionToolBar.SetOrientation Method

Pascal

```
procedure SetOrientation(const Value: TBarOrientation); override;
```

C++

```
virtual __fastcall SetOrientation(const TBarOrientation Value);
```

Description

This is SetOrientation, a member of class TCustomActionToolBar.

2.1.4.4.21 TCustomActionToolBar.SetupDropDownBtn Method

Pascal

```
procedure SetupDropDownBtn;
```

C++

```
__fastcall SetupDropDownBtn();
```

Description

This is SetupDropDownBtn, a member of class TCustomActionToolBar.

2.1.4.5 TCustomActionToolBar Properties

The properties of the TCustomActionToolBar class are listed here.

Public Properties

	Name	Description
R	HiddenCount (see page 168)	Indicates the number of hidden items on the tool action band.

2.1.4.5.1 TCustomActionToolBar.HiddenCount Property

Indicates the number of hidden items on the tool action band.

Pascal

```
property HiddenCount: Integer;
```

C++

```
__property int HiddenCount;
```

Description

Access HiddenCount to determine how many of the action tool bar's items are hidden. Hidden items are items that do not appear on the tool bar because the user has not used them recently. When HiddenCount is greater than 0, the tool bar displays an indicator that the user can click to display the hidden items.

See Also

ActionControls (see page 479)

AllowHiding (see page 480)

2.1.5 TCustomButtonControl Class

TCustomButtonControl is the base class for ActionBand button controls.

Class Hierarchy

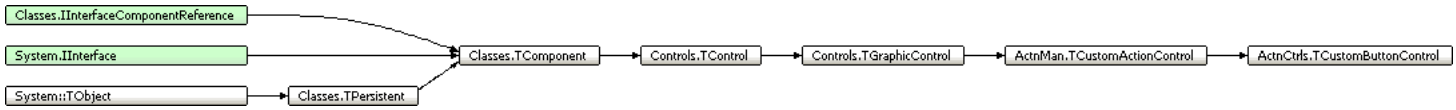

Pascal

```
TCustomButtonControl = class(TCustomActionControl);
```

C++

```
class TCustomButtonControl : public TCustomActionControl;
```

File

ActnCtrls

Description

TCustomButtonControl is a base class for several ActionBand components. Most notably, the standard menu button (TStandardMenuButton) derives from this class.

Do not create instances of TCustomButtonControl. If you would like to replace the standard menu button behaviors, use the class TCustomMenuButton as a parent class for your replacement class.

See Also

TCustomMenuButton (see page 790)

TCustomMenuItem (see page 816)

TStandardMenuButton (see page 20444)

TStandardMenuItem (see page 20465)

2.1.5.1 TCustomButtonControl Members

The following tables list the members exposed by TCustomButtonControl.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.

	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFlat (see page 178)	This is FFlat, a member of class TCustomButtonControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FState (see page 178)	This is FState, a member of class TCustomButtonControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	BeginAutoDrag (see page 517)	Begin Automatic Drag.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CalcLayout (see page 518)	This is CalcLayout, a member of class TCustomActionControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	CMDesignHitTest (see page 519)	This is CMDesignHitTest, a member of class TCustomActionControl.
	CMDialogChar (see page 180)	This is CMDialogChar, a member of class TCustomButtonControl.
	CMEnabledChanged (see page 180)	This is CMEnabledChanged, a member of class TCustomButtonControl.
	CMFontChanged (see page 519)	This is CMFontChanged, a member of class TCustomActionControl.
	CMItemDropPoint (see page 519)	This is CMItemDropPoint, a member of class TCustomActionControl.
	CMItemSelected (see page 519)	This is CMItemSelected, a member of class TCustomActionControl.
	CMMouseEnter (see page 180)	This is CMMouseEnter, a member of class TCustomButtonControl.
	CMMouseLeave (see page 180)	This is CMMouseLeave, a member of class TCustomButtonControl.
	CMTextChanged (see page 180)	This is CMTextChanged, a member of class TCustomButtonControl.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DesignWndProc (see page 519)	This is DesignWndProc, a member of class TCustomActionControl.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
V	DoDragDrop (see page 520)	This is DoDragDrop, a member of class TCustomActionControl.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.

	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
V	DoStartDrag (see page 520)	OnStartDrag event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 521)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawBackground (see page 181)	Draws a background image on the control.
V	DrawDesignFocus (see page 521)	This is DrawDesignFocus, a member of class TCustomActionControl.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawDragDropPoint (see page 522)	This is DrawDragDropPoint, a member of class TCustomActionControl.
V	DrawFrame (see page 181)	This is DrawFrame, a member of class TCustomButtonControl.
V	DrawGlyph (see page 522)	Draws a symbol on the control.
V	DrawLargeGlyph (see page 181)	This is DrawLargeGlyph, a member of class TCustomButtonControl.
V	DrawSeparator (see page 522)	This is DrawSeparator, a member of class TCustomActionControl.
V	DrawShadowedText (see page 523)	This is DrawShadowedText, a member of class TCustomActionControl.
V	DrawText (see page 523)	Draws text on the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	GetAction (see page 523)	This is GetAction, a member of class TCustomActionControl.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetImageSize (see page 523)	This is GetImageSize, a member of class TCustomActionControl.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
V	GetSeparator (see page 524)	This is GetSeparator, a member of class TCustomActionControl.
V	GetShowCaption (see page 524)	This is GetShowCaption, a member of class TCustomActionControl.
V	GetShowShortCut (see page 181)	This is GetShowShortCut, a member of class TCustomButtonControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasGlyph (see page 524)	This is HasGlyph, a member of class TCustomActionControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsChecked (see page 525)	This is IsChecked, a member of class TCustomActionControl.
V	IsGrouped (see page 525)	This is IsGrouped, a member of class TCustomActionControl.

	Loaded (see page 182)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 182)	OnMouseDown event dispatcher.
	MouseMove (see page 183)	Respond to mouse moving over control area..
	MouseUp (see page 183)	OnMouseUp event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Paint (see page 184)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionChanged (see page 526)	This is PositionChanged, a member of class TCustomActionControl.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetUsageData (see page 526)	This is ResetUsageData, a member of class TCustomActionControl.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActionClient (see page 184)	This is SetActionClient, a member of class TCustomButtonControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropPoint (see page 527)	This is SetDropPoint, a member of class TCustomActionControl.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetGlyphLayout (see page 185)	This is SetGlyphLayout, a member of class TCustomButtonControl.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelected (see page 185)	This is SetSelected, a member of class TCustomButtonControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateSelection (see page 527)	This is UpdateSelection, a member of class TCustomActionControl.
	UpdateTextBounds (see page 527)	This is UpdateTextBounds, a member of class TCustomActionControl.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMLButtonDblClk (see page 185)	This is WMLButtonDblClk, a member of class TCustomButtonControl.
	WMSetText (see page 528)	This is WMSetText, a member of class TCustomActionControl.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GlyphPos (see page 532)	This is GlyphPos, a member of class TCustomActionControl.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	MouseInControl (see page 189)	This is MouseInControl, a member of class TCustomButtonControl.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnClick (see page 535)	Occurs when the user clicks the control.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
V	CalcBounds (see page 179)	Calculates the size of the control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Click (see page 179)	Executes the control's function.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 177)	Creates an instance of TCustomButtonControl (see page 168).
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 516)	Destroys an instance of TCustomActionControl (see page 507).
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
V	DragDrop (see page 520)	Performs actions and generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.

	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 524)	Calls the action client's Update method if the control is associated with an action client.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	Keyed (see page 525)	Executes the control's function in response to a keyboard action.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 184)	Sets the Left, Top, Width, and Height properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	ActionBar (see page 528)	The ActionBand object which contains the custom action control
	ActionClient (see page 529)	Refers to the control's action client
	Align (see page 5798)	Determines how the control aligns within its container (parent control).

	AllowAllUp (see page 186)	Indicates whether all the buttons in a given group can be up at the same time.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Caption (see page 186)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Down (see page 189)	Tells whether the button is down (selected) or up (unselected).
	DropPoint (see page 532)	This is DropPoint, a member of class TCustomActionControl.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
	Flat (see page 189)	Dictates whether the button should have a 2D look instead of the usual 3D look.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GlyphLayout (see page 532)	Describes where the image should be on the control
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Margins (see page 532)	Describes the boundaries of the control for drawing purposes.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	Selected (see page 533)	Indicates whether the control is currently selected.
R	Separator (see page 533)	Indicates whether the control is a separator.
	ShowCaption (see page 189)	Indicates whether the control's caption should be shown.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
	ShowShortCut (see page 190)	Indicates whether the control's short cut should be shown.
	SmallIcon (see page 534)	Indicates whether the control's small icon should be shown.
	Spacing (see page 534)	Controls (see page 5711) the separation between the control's icon and caption.
	State (see page 190)	Describes the current state of the button.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TextBounds (see page 534)	A bounding rectangle for the control caption.

	TrackButton (see page 190)	Specifies which mouse button activates the button control.
	Transparent (see page 534)	Shows whether the control is transparent or not.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

2.1.5.2 TCustomButtonControl.Create Constructor

Creates an instance of TCustomButtonControl (see page 168).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomButtonControl(TComponent * AOwner);
```

Description

Calling Create constructs and initializes an instance of TCustomButtonControl (see page 168). However, you should never attempt to instantiate a TCustomButtonControl (see page 168). This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component that is responsible for freeing the button control. It becomes the value of the Owner property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner (see page 2807)

Create (see page 5731)

Destroy

2.1.5.3 TCustomButtonControl Fields

The fields of the TCustomButtonControl class are listed here.

Protected Fields

	Name	Description
	FFlat (see page 178)	This is FFlat, a member of class TCustomButtonControl.
	FState (see page 178)	This is FState, a member of class TCustomButtonControl.

2.1.5.3.1 TCustomButtonControl.FFlat Field

Pascal

```
FFlat: Boolean;
```

C++

```
Boolean FFlat;
```

Description

This is FFlat, a member of class TCustomButtonControl.

2.1.5.3.2 TCustomButtonControl.FState Field

Pascal

```
FState: TButtonState;
```

C++

```
TButtonState FState;
```

Description

This is FState, a member of class TCustomButtonControl.

2.1.5.4 TCustomButtonControl Methods

The methods of the TCustomButtonControl class are listed here.

Protected Methods

	Name	Description
	CMDialogChar (see page 180)	This is CMDialogChar, a member of class TCustomButtonControl.
	CMEnabledChanged (see page 180)	This is CMEnabledChanged, a member of class TCustomButtonControl.
	CMMouseEnter (see page 180)	This is CMMouseEnter, a member of class TCustomButtonControl.
	CMMouseLeave (see page 180)	This is CMMouseLeave, a member of class TCustomButtonControl.
	CMTextChanged (see page 180)	This is CMTextChanged, a member of class TCustomButtonControl.
V	DrawBackground (see page 181)	Draws a background image on the control.
V	DrawFrame (see page 181)	This is DrawFrame, a member of class TCustomButtonControl.
V	DrawLargeGlyph (see page 181)	This is DrawLargeGlyph, a member of class TCustomButtonControl.
V	GetShowShortCut (see page 181)	This is GetShowShortCut, a member of class TCustomButtonControl.
V	Loaded (see page 182)	Initializes the control after it is loaded from a stream.
V	MouseDown (see page 182)	OnMouseDown event dispatcher.
V	MouseMove (see page 183)	Respond to mouse moving over control area..
V	MouseUp (see page 183)	OnMouseUp event dispatcher.

	Paint (see page 184)	Render control surface.
	SetActionClient (see page 184)	This is SetActionClient, a member of class TCustomButtonControl.
	SetGlyphLayout (see page 185)	This is SetGlyphLayout, a member of class TCustomButtonControl.
	SetSelected (see page 185)	This is SetSelected, a member of class TCustomButtonControl.
	WMLButtonDblClk (see page 185)	This is WMLButtonDblClk, a member of class TCustomButtonControl.

Public Methods

	Name	Description
	CalcBounds (see page 179)	Calculates the size of the control.
	Click (see page 179)	Executes the control's function.
	SetBounds (see page 184)	Sets the Left, Top, Width, and Height properties all at once.

2.1.5.4.1 TCustomButtonControl.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

This method is largely the same as CalcBounds (TCustomActionControl), with one minor exception. The width of the control is set to be one pixel wider. For more information, see the documentation for the inherited method.

See Also

CalcBounds (see page 518)

2.1.5.4.2 TCustomButtonControl.Click Method

Executes the control's function.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

This override of the method inherited from TCustomActionControl is almost identical to the inherited method. The only difference is that if the button control is being manipulated at design time, extra actions are taken to ensure that the control is properly selected.

See Also

Click (see page 518)

Click (see page 5743)

UsageCount (see page 388)

2.1.5.4.3 TCustomButtonControl.CMDialogChar Method

Pascal

```
procedure CMDialogChar(var Message: TCMDialogChar); message CM_DIALOGCHAR;
```

C++

```
__fastcall CMDialogChar(TCMDialogChar Message);
```

Description

This is CMDialogChar, a member of class TCustomButtonControl.

2.1.5.4.4 TCustomButtonControl.CMEnabledChanged Method

Pascal

```
procedure CMEnabledChanged(var Message: TMessage); message CM_ENABLEDCHANGED;
```

C++

```
__fastcall CMEnabledChanged(TMessage Message);
```

Description

This is CMEnabledChanged, a member of class TCustomButtonControl.

2.1.5.4.5 TCustomButtonControl.CMMouseEnter Method

Pascal

```
procedure CMMouseEnter(var Message: TMessage); message CM_MOUSEENTER;
```

C++

```
__fastcall CMMouseEnter(TMessage Message);
```

Description

This is CMMouseEnter, a member of class TCustomButtonControl.

2.1.5.4.6 TCustomButtonControl.CMMouseLeave Method

Pascal

```
procedure CMMouseLeave(var Message: TMessage); message CM_MOUSELEAVE;
```

C++

```
__fastcall CMMouseLeave(TMessage Message);
```

Description

This is CMMouseLeave, a member of class TCustomButtonControl.

2.1.5.4.7 TCustomButtonControl.CMTextChanged Method

Pascal

```
procedure CMTextChanged(var Message: TMessage); message CM_TEXTCHANGED;
```

C++

```
__fastcall CMTextChanged(TMessage Message);
```

Description

This is CMTextChanged, a member of class TCustomButtonControl.

2.1.5.4.8 TCustomButtonControl.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph (see page 522)

DrawText (see page 523)

2.1.5.4.9 TCustomButtonControl.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); virtual;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TCustomButtonControl.

2.1.5.4.10 TCustomButtonControl.DrawLargeGlyph Method

Pascal

```
procedure DrawLargeGlyph(Location: TPoint); override;
```

C++

```
virtual __fastcall DrawLargeGlyph(TPoint Location);
```

Description

This is DrawLargeGlyph, a member of class TCustomButtonControl.

2.1.5.4.11 TCustomButtonControl.GetShowShortCut Method

Pascal

```
function GetShowShortCut: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetShowShortCut();
```

Description

This is GetShowShortCut, a member of class TCustomButtonControl.

2.1.5.4.12 TCustomButtonControl.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded (see page 2782)

Action (see page 5797)

ActionChange (see page 5737)

2.1.5.4.13 TCustomButtonControl.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove (see page 5773)

MouseUp (see page 5774)

OnMouseDown (see page 5862)

2.1.5.4.14 TCustomButtonControl.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown (see page 5773)

MouseUp (see page 5774)

OnMouseMove (see page 5865)

2.1.5.4.15 TCustomButtonControl.MouseUp Method

OnMouseUp event dispatcher.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user releases a previously pressed mouse button while the cursor's hotspot is over the control.

A control calls MouseUp in response to any of the Windows mouse-up messages (WM_LBUTTONUP, WM_MBUTTONUP, WM_RBUTTONUP), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

See Also

MouseDown (see page 5773)

MouseMove (see page 5773)

OnMouseUp (see page 5867)

2.1.5.4.16 TCustomButtonControl.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas (see page 6053)

2.1.5.4.17 TCustomButtonControl.SetActionClient Method

Pascal

```
procedure SetActionClient(Value: TActionClientItem); override;
```

C++

```
virtual __fastcall SetActionClient(TActionClientItem Value);
```

Description

This is SetActionClient, a member of class TCustomButtonControl.

2.1.5.4.18 TCustomButtonControl.SetBounds Method

Sets the Left, Top, Width, and Height properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

This override of the method inherited from TControl simultaneously sets all boundary properties at once, and thus avoiding the possibility of a control redraw between the time one value sets and another. In addition, some layout calculations are also performed.

See Also

SetBounds (see page 5783)

2.1.5.4.19 TCustomButtonControl.SetGlyphLayout Method

Pascal

```
procedure SetGlyphLayout(const Value: TButtonLayout); override;
```

C++

```
virtual __fastcall SetGlyphLayout(const TButtonLayout Value);
```

Description

This is SetGlyphLayout, a member of class TCustomButtonControl.

2.1.5.4.20 TCustomButtonControl.SetSelected Method

Pascal

```
procedure SetSelected(Value: Boolean); override;
```

C++

```
virtual __fastcall SetSelected(Boolean Value);
```

Description

This is SetSelected, a member of class TCustomButtonControl.

2.1.5.4.21 TCustomButtonControl.WMLButtonDblClk Method

Pascal

```
procedure WMLButtonDblClk(var Message: TWMLButtonDown); message WM_LBUTTONDBLCLK;
```

C++

```
__fastcall WMLButtonDblClk(TWMLButtonDown Message);
```

Description

This is WMLButtonDblClk, a member of class TCustomButtonControl.

2.1.5.5 TCustomButtonControl Properties

The properties of the TCustomButtonControl class are listed here.

Protected Properties

	Name	Description
	MouseInControl (see page 189)	This is MouseInControl, a member of class TCustomButtonControl.

Public Properties

	Name	Description
	AllowAllUp (see page 186)	Indicates whether all the buttons in a given group can be up at the same time.
	Caption (see page 186)	Specifies a text string that identifies the control to the user.
	Down (see page 189)	Tells whether the button is down (selected) or up (unselected).
	Flat (see page 189)	Dictates whether the button should have a 2D look instead of the usual 3D look.
	ShowCaption (see page 189)	Indicates whether the control's caption should be shown.
	ShowShortCut (see page 190)	Indicates whether the control's short cut should be shown.
	State (see page 190)	Describes the current state of the button.
	TrackButton (see page 190)	Specifies which mouse button activates the button control.

2.1.5.5.1 TCustomButtonControl.AllowAllUp Property

Indicates whether all the buttons in a given group can be up at the same time.

Pascal

```
property AllowAllUp: Boolean;
```

C++

```
__property Boolean AllowAllUp;
```

Description

Set AllowAllUp to true to allow all buttons in a given group to be deselected simultaneously. If AllowAllUp is false, then the button group in question must have exactly one button selected at any given time. Groups of buttons designed this way are sometimes called "radio buttons."

See Also

AllowAllUp (see page 2354)

AllowAllUp (see page 5127)

2.1.5.5.2 TCustomButtonControl.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
```

```
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

2.1.5.5.3 TCustomButtonControl.Down Property

Tells whether the button is down (selected) or up (unselected).

Pascal

```
property Down: Boolean;
```

C++

```
__property Boolean Down;
```

Description

Use Down to determine if the button is down or not. This property can also be used to change the state of the button by changing its value. This property works the same as the Down properties in TSpeedButton and TToolButton.

See Also

Down (see page 2357)

Down (see page 5128)

2.1.5.5.4 TCustomButtonControl.Flat Property

Dictates whether the button should have a 2D look instead of the usual 3D look.

Pascal

```
property Flat: Boolean;
```

C++

```
__property Boolean Flat;
```

Description

Set Flat to true if you want the button to display the button without the edge bevel that gives buttons a 3D look. This property's behavior should be identical to that of the Flat property of TSpeedButton.

See Also

Flat (see page 2357)

2.1.5.5.5 TCustomButtonControl.MouseInControl Property

Pascal

```
property MouseInControl: Boolean;
```

C++

```
__property Boolean MouseInControl;
```

Description

This is MouseInControl, a member of class TCustomButtonControl.

2.1.5.5.6 TCustomButtonControl.ShowCaption Property

Indicates whether the control's caption should be shown.

Pascal

```
property ShowCaption: Boolean;
```

C++

```
__property Boolean ShowCaption;
```

Description

Read ShowCaption to determine whether or not the custom action control's caption, a text description of the control's function, should be shown.

2.1.5.5.7 TCustomButtonControl.ShowShortCut Property

Indicates whether the control's short cut should be shown.

Pascal

```
property ShowShortCut: Boolean;
```

C++

```
__property Boolean ShowShortCut;
```

Description

Read ShowShortCut to determine whether or not the custom action control's short cut keystrokes should be shown.

2.1.5.5.8 TCustomButtonControl.State Property

Describes the current state of the button.

Pascal

```
property State: TButtonState;
```

C++

```
__property TButtonState State;
```

Description

Use State as an indicator of the state of the button control. The possible values are as follows:

Value	State Description
bsUp	The button control is up, or unselected.
bsDisabled	The button is not enabled, which generally means it is greyed out. It cannot be selected by the user.
bsDown	The button control is down, or selected.
bsExclusive	The button is a "radio button." It is one of a group of buttons, exactly one of which must be selected at any given time. Selecting another button will cause the previously selected button to become deselected.

2.1.5.5.9 TCustomButtonControl.TrackButton Property

Specifies which mouse button activates the button control.

Pascal

```
property TrackButton: TMouseButton;
```

C++

```
__property TMouseButton TrackButton;
```

Description

Use TrackButton to specify which mouse button activates the button control.

See Also

TMouseButton (see page 6288)

2.1.6 TCustomComboControl Class

Class Hierarchy

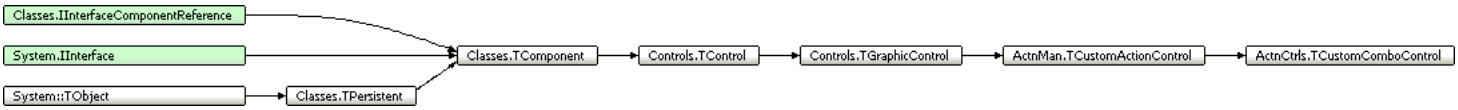

Pascal

```
TCustomComboControl = class(TCustomActionControl);
```

C++

```
class TCustomComboControl : public TCustomActionControl;
```

File

ActnCtrls

Description

This is class ActnCtrls.TCustomComboControl.

2.1.6.1 TCustomComboControl Members

The following tables list the members exposed by TCustomComboControl.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.

	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	BeginAutoDrag (see page 517)	Begin Automatic Drag.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CalcLayout (see page 518)	This is CalcLayout, a member of class TCustomActionControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	CMDesignHitTest (see page 519)	This is CMDesignHitTest, a member of class TCustomActionControl.
	CMFontChanged (see page 519)	This is CMFontChanged, a member of class TCustomActionControl.
	CMItemDropPoint (see page 519)	This is CMItemDropPoint, a member of class TCustomActionControl.
	CMItemSelected (see page 519)	This is CMItemSelected, a member of class TCustomActionControl.
	CMVisiblechanged (see page 201)	This is CMVisiblechanged, a member of class TCustomComboControl.
	ComboClick (see page 202)	This is ComboClick, a member of class TCustomComboControl.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 519)	This is DesignWndProc, a member of class TCustomActionControl.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDragDrop (see page 520)	This is DoDragDrop, a member of class TCustomActionControl.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 520)	OnStartDrag event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 521)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawBackground (see page 521)	Draws a background image on the control.
	DrawDesignFocus (see page 521)	This is DrawDesignFocus, a member of class TCustomActionControl.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawDragDropPoint (see page 522)	This is DrawDragDropPoint, a member of class TCustomActionControl.
	DrawGlyph (see page 522)	Draws a symbol on the control.
	DrawLargeGlyph (see page 522)	This is DrawLargeGlyph, a member of class TCustomActionControl.
	DrawSeparator (see page 522)	This is DrawSeparator, a member of class TCustomActionControl.
	DrawShadowedText (see page 523)	This is DrawShadowedText, a member of class TCustomActionControl.
	DrawText (see page 523)	Draws text on the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	GetAction (see page 523)	This is GetAction, a member of class TCustomActionControl.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetImageSize (see page 523)	This is GetImageSize, a member of class TCustomActionControl.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSeparator (see page 524)	This is GetSeparator, a member of class TCustomActionControl.

	GetShowCaption (see page 524)	This is GetShowCaption, a member of class TCustomActionControl.
	GetShowShortCut (see page 524)	This is GetShowShortCut, a member of class TCustomActionControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasGlyph (see page 524)	This is HasGlyph, a member of class TCustomActionControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsChecked (see page 525)	This is IsChecked, a member of class TCustomActionControl.
	IsGrouped (see page 525)	This is IsGrouped, a member of class TCustomActionControl.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Paint (see page 202)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionChanged (see page 526)	This is PositionChanged, a member of class TCustomActionControl.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetUsageData (see page 526)	This is ResetUsageData, a member of class TCustomActionControl.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActionClient (see page 202)	This is SetActionClient, a member of class TCustomComboControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 203)	Sets the drag mode for the control.
	SetDropPoint (see page 527)	This is SetDropPoint, a member of class TCustomActionControl.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetGlyphLayout (see page 527)	This is SetGlyphLayout, a member of class TCustomActionControl.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 204)	Sets the parent of the control.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelected (see page 527)	This is SetSelected, a member of class TCustomActionControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.

	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateSelection (see page 527)	This is UpdateSelection, a member of class TCustomActionControl.
	UpdateTextBounds (see page 527)	This is UpdateTextBounds, a member of class TCustomActionControl.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	VisibleChanging (see page 204)	Provides the interface for a method that responds to changes in the control's visibility.
	WMSetText (see page 528)	This is WMSetText, a member of class TCustomActionControl.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GlyphPos (see page 532)	This is GlyphPos, a member of class TCustomActionControl.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnClick (see page 535)	Occurs when the user clicks the control.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
V	CalcBounds (see page 201)	Calculates the size of the control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Click (see page 201)	Executes the control's function in response to a mouse click.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 199)	Creates an instance of TCustomActionControl.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 200)	Destroys an instance of TCustomActionControl.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
V	DragDrop (see page 520)	Performs actions and generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 524)	Calls the action client's Update method if the control is associated with an action client.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	Keyed (see page 525)	Executes the control's function in response to a keyboard action.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 202)	Sets the Left, Top, Width, and Height properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.

	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	ActionBar (see page 528)	The ActionBand object which contains the custom action control
	ActionClient (see page 529)	Refers to the control's action client
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Caption (see page 529)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComboBox (see page 205)	This is ComboBox, a member of class TCustomComboControl.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DropPoint (see page 532)	This is DropPoint, a member of class TCustomActionControl.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GlyphLayout (see page 532)	Describes where the image should be on the control
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Margins (see page 532)	Describes the boundaries of the control for drawing purposes.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	Selected (see page 533)	Indicates whether the control is currently selected.
R	Separator (see page 533)	Indicates whether the control is a separator.
R	ShowCaption (see page 533)	Indicates whether the control's caption should be shown.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	ShowShortCut (see page 533)	Indicates whether the control's short cut should be shown.

	SmallIcon (see page 534)	Indicates whether the control's small icon should be shown.
	Spacing (see page 534)	Controls (see page 5711) the separation between the control's icon and caption.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TextBounds (see page 534)	A bounding rectangle for the control caption.
	Transparent (see page 534)	Shows whether the control is transparent or not.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

2.1.6.2 TCustomComboControl.Create Constructor

Creates an instance of TCustomActionControl.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomComboControl(TComponent * AOwner);
```

Description

Calling Create constructs and initializes an instance of TCustomActionControl. However, you should never attempt to instantiate a TCustomActionControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component that is responsible for freeing the TCustomActionControl instance. It becomes the value of the Owner property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner (see page 2807)

Create (see page 5731)

Destroy (see page 516)

2.1.6.3 TCustomComboControl.Destroy Destructor

Destroys an instance of TCustomActionControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomComboControl();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the graphic control reference is not nil, and only then calls Destroy.

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Frees the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

Create (see page 515)

Free (see page 22300)

2.1.6.4 TCustomComboControl Methods

The methods of the TCustomComboControl class are listed here.

Protected Methods

	Name	Description
	CMVisiblechanged (see page 201)	This is CMVisiblechanged, a member of class TCustomComboControl.
	ComboClick (see page 202)	This is ComboClick, a member of class TCustomComboControl.
V	Paint (see page 202)	Render control surface.
V	SetActionClient (see page 202)	This is SetActionClient, a member of class TCustomComboControl.
V	SetDragMode (see page 203)	Sets the drag mode for the control.
V	SetParent (see page 204)	Sets the parent of the control.
V	VisibleChanging (see page 204)	Provides the interface for a method that responds to changes in the control's visibility.

Public Methods

	Name	Description
V	CalcBounds (see page 201)	Calculates the size of the control.

	Click (see page 201)	Executes the control's function in response to a mouse click.
	SetBounds (see page 202)	Sets the Left, Top, Width, and Height properties all at once.

2.1.6.4.1 TCustomComboControl.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as the presence of an icon, caption, and shortcuts. CalcBounds computes the boundaries of the control considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds (see page 5783)

2.1.6.4.2 TCustomComboControl.Click Method

Executes the control's function in response to a mouse click.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

This override of the method inherited from TControl executes the control's function in response to a user's mouse click. Usage data, which is used to determine whether a control is used frequently enough to be shown by default, is updated as well.

See Also

Click (see page 5743)

UsageCount (see page 388)

Keyed (see page 525)

2.1.6.4.3 TCustomComboControl.CMVisiblechanged Method

Pascal

```
procedure CMVisiblechanged(var Message: TMessage); message CM_VISIBLECHANGED;
```

C++

```
__fastcall CMVisiblechanged(TMessage Message);
```

Description

This is CMVisiblechanged, a member of class TCustomComboControl.

2.1.6.4.4 TCustomComboControl.ComboClick Method

Pascal

```
procedure ComboClick(Sender: TObject);
```

C++

```
__fastcall ComboClick(TObject * Sender);
```

Description

This is ComboClick, a member of class TCustomComboControl.

2.1.6.4.5 TCustomComboControl.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas (see page 6053)

2.1.6.4.6 TCustomComboControl.SetActionClient Method

Pascal

```
procedure SetActionClient(Value: TActionClientItem); override;
```

C++

```
virtual __fastcall SetActionClient(TActionClientItem Value);
```

Description

This is SetActionClient, a member of class TCustomComboControl.

2.1.6.4.7 TCustomComboControl.SetBounds Method

Sets the Left, Top, Width, and Height properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to change all of the component's boundary properties at one time. The same effect can be achieved by setting the Left, Top, Width, and Height properties separately, but SetBounds changes all four properties at once ensuring that the control will not repaint between changes.

Specify the values for the Left, Top, Width, and Height properties as the value of the ALeft, ATop, AWidth, and AHeight parameters, respectively.

Calling SetBounds does not necessarily result in the Left, Top, Width, and Height properties changing to the specified values. Before the properties are changed, the AutoSize or Constraints property may limit the changes, and an OnCanResize (or OnConstrainedResize) event handler may change the new values. After the control's Left, Top, Width, and Height properties are changed, SetBounds generates an OnResize event.

Note: Component writers can change the Left, Top, Width, and Height properties while bypassing all resize events and constraint or autosize logic by using the UpdateBoundsRect method instead.

See Also

Height (see page 5825)

Left (see page 5830)

Top (see page 5844)

Width (see page 5849)

BoundsRect (see page 5803)

AutoSize (see page 5803)

Constraints (see page 5812)

OnCanResize (see page 5851)

OnConstrainedResize (see page 5853)

OnResize (see page 5870)

UpdateBoundsRect (see page 5789)

2.1.6.4.8 TCustomComboControl.SetDragMode Method

Sets the drag mode for the control.

Pascal

```
procedure SetDragMode(Value: TDragMode); override;
```

C++

```
virtual __fastcall SetDragMode(TDragMode Value);
```

Description

SetDragMode is the protected write implementation for the DragMode property. Override SetDragMode to execute additional code when the value of the DragMode property changes.

See Also

DragMode (see page 5819)

2.1.6.4.9 TCustomComboControl.SetParent Method

Sets the parent of the control.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. Override SetParent to execute additional code when the value of the Parent property changes.

The AParent parameter specifies the new parent of the control.

If the control already has a parent, SetParent removes the control from that parent's list of controls by calling the parent's RemoveControl method. If AParent is not nil (Delphi) or NULL (C++), SetParent calls its InsertControl method to add the control to its list of controls.

The TControl destructor calls SetParent(nil (Delphi) or NULL (C++)) to remove the control from its parent's control list before destroying the component.

See Also

Destroy (see page 5732)

InsertControl (see page 6160)

Parent (see page 5832)

RemoveControl (see page 6172)

2.1.6.4.10 TCustomComboControl.VisibleChanging Method

Provides the interface for a method that responds to changes in the control's visibility.

Pascal

```
procedure VisibleChanging; override;
```

C++

```
virtual __fastcall VisibleChanging();
```

Description

VisibleChanging is called before the value of a control's Visible property changes. As implemented in TControl, it does nothing. Override VisibleChanging to respond to changes in the control's visibility. For example, if a control should not change its visibility, override VisibleChanging to raise an exception when an attempt is made to alter the control's visibility.

See Also

Visible (see page 5847)

2.1.6.5 TCustomComboControl Properties

The properties of the TCustomComboControl class are listed here.

Public Properties

	Name	Description
	ComboBox (see page 205)	This is ComboBox, a member of class TCustomComboControl.

2.1.6.5.1 TCustomComboControl.ComboBox Property

Pascal

```
property ComboBox: TCustomActionCombo;
```

C++

```
__property TCustomActionCombo ComboBox;
```

Description

This is ComboBox, a member of class TCustomComboControl.

2.1.7 TCustomDropDownButton Class

Class Hierarchy

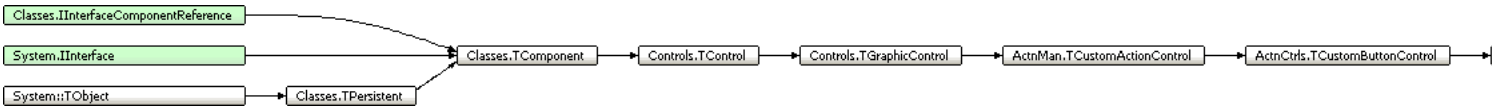

Pascal

```
TCustomDropDownButton = class(TCustomButtonControl);
```

C++

```
class TCustomDropDownButton : public TCustomButtonControl;
```

File

ActnCtrls

Description

This is class ActnCtrls.TCustomDropDownButton.

2.1.7.1 TCustomDropDownButton Members

The following tables list the members exposed by TCustomDropDownButton.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.

	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFlat (see page 178)	This is FFlat, a member of class TCustomButtonControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FState (see page 178)	This is FState, a member of class TCustomButtonControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	BeginAutoDrag (see page 517)	Begin Automatic Drag.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CalcLayout (see page 518)	This is CalcLayout, a member of class TCustomActionControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.

	CMDesignHitTest (see page 519)	This is CMDesignHitTest, a member of class TCustomActionControl.
	CMDialogChar (see page 180)	This is CMDialogChar, a member of class TCustomButtonControl.
	CMEnabledChanged (see page 180)	This is CMEnabledChanged, a member of class TCustomButtonControl.
	CMFontChanged (see page 519)	This is CMFontChanged, a member of class TCustomActionControl.
	CMItemDropPoint (see page 519)	This is CMItemDropPoint, a member of class TCustomActionControl.
	CMItemSelected (see page 519)	This is CMItemSelected, a member of class TCustomActionControl.
	CMMouseEnter (see page 180)	This is CMMouseEnter, a member of class TCustomButtonControl.
	CMMouseLeave (see page 180)	This is CMMouseLeave, a member of class TCustomButtonControl.
	CMTextChanged (see page 180)	This is CMTextChanged, a member of class TCustomButtonControl.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DesignWndProc (see page 519)	This is DesignWndProc, a member of class TCustomActionControl.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
V	DoDragDrop (see page 520)	This is DoDragDrop, a member of class TCustomActionControl.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
V	DoStartDrag (see page 520)	OnStartDrag event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 521)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawBackground (see page 181)	Draws a background image on the control.
V	DrawBackground (see page 521)	Draws a background image on the control.
V	DrawDesignFocus (see page 521)	This is DrawDesignFocus, a member of class TCustomActionControl.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawDragDropPoint (see page 522)	This is DrawDragDropPoint, a member of class TCustomActionControl.
V	DrawFrame (see page 215)	This is DrawFrame, a member of class TCustomDropDownButton.
V	DrawGlyph (see page 522)	Draws a symbol on the control.
V	DrawLargeGlyph (see page 181)	This is DrawLargeGlyph, a member of class TCustomButtonControl.
V	DrawLargeGlyph (see page 522)	This is DrawLargeGlyph, a member of class TCustomActionControl.
V	DrawSeparator (see page 522)	This is DrawSeparator, a member of class TCustomActionControl.
V	DrawShadowedText (see page 523)	This is DrawShadowedText, a member of class TCustomActionControl.
V	DrawText (see page 523)	Draws text on the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	GetAction (see page 523)	This is GetAction, a member of class TCustomActionControl.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.

	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetImageSize (see page 523)	This is GetImageSize, a member of class TCustomActionControl.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupClass (see page 215)	This is GetPopupClass, a member of class TCustomDropDownButton.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSeparator (see page 524)	This is GetSeparator, a member of class TCustomActionControl.
	GetShowCaption (see page 524)	This is GetShowCaption, a member of class TCustomActionControl.
	GetShowShortCut (see page 181)	This is GetShowShortCut, a member of class TCustomButtonControl.
	GetShowShortCut (see page 524)	This is GetShowShortCut, a member of class TCustomActionControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasGlyph (see page 524)	This is HasGlyph, a member of class TCustomActionControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsChecked (see page 525)	This is IsChecked, a member of class TCustomActionControl.
	IsGrouped (see page 525)	This is IsGrouped, a member of class TCustomActionControl.
	Loaded (see page 182)	Initializes the control after it is loaded from a stream.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 216)	OnMouseDown event dispatcher.
	MouseMove (see page 183)	Respond to mouse moving over control area..
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseOverDropDown (see page 216)	This is MouseOverDropDown, a member of class TCustomDropDownButton.
	MouseUp (see page 183)	OnMouseUp event dispatcher.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Paint (see page 216)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionChanged (see page 526)	This is PositionChanged, a member of class TCustomActionControl.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetUsageData (see page 526)	This is ResetUsageData, a member of class TCustomActionControl.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.

V	SetActionClient (see page 184)	This is SetActionClient, a member of class TCustomButtonControl.
V	SetActionClient (see page 526)	This is SetActionClient, a member of class TCustomActionControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropPoint (see page 527)	This is SetDropPoint, a member of class TCustomActionControl.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
V	SetGlyphLayout (see page 185)	This is SetGlyphLayout, a member of class TCustomButtonControl.
V	SetGlyphLayout (see page 527)	This is SetGlyphLayout, a member of class TCustomActionControl.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetSelected (see page 185)	This is SetSelected, a member of class TCustomButtonControl.
V	SetSelected (see page 527)	This is SetSelected, a member of class TCustomActionControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateSelection (see page 527)	This is UpdateSelection, a member of class TCustomActionControl.
	UpdateTextBounds (see page 527)	This is UpdateTextBounds, a member of class TCustomActionControl.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMLButtonDblClk (see page 185)	This is WMLButtonDblClk, a member of class TCustomButtonControl.
	WMSetText (see page 528)	This is WMSetText, a member of class TCustomActionControl.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.

	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
R	DroppedDown (see page 217)	This is DroppedDown, a member of class TCustomDropDownButton.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GlyphPos (see page 532)	This is GlyphPos, a member of class TCustomActionControl.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	MouseInControl (see page 189)	This is MouseInControl, a member of class TCustomButtonControl.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnClick (see page 535)	Occurs when the user clicks the control.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
V	CalcBounds (see page 214)	Calculates the size of the control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Click (see page 215)	Executes the control's function.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 177)	Creates an instance of TCustomButtonControl (see page 168).
V	Create (see page 515)	Creates an instance of TCustomActionControl (see page 507).
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6051)	Creates an instance of TGraphicControl (see page 6044).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

	Destroy (see page 516)	Destroys an instance of TCustomActionControl (see page 507).
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 520)	Performs actions and generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	DropDownClick (see page 215)	This is DropDownClick, a member of class TCustomDropDownButton.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 524)	Calls the action client's Update method if the control is associated with an action client.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.

	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
V	Keyed (see page 525)	Executes the control's function in response to a keyboard action.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 184)	Sets the Left, Top, Width, and Height properties all at once.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	ActionBar (see page 528)	The ActionBand object which contains the custom action control
	ActionClient (see page 529)	Refers to the control's action client
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	AllowAllUp (see page 186)	Indicates whether all the buttons in a given group can be up at the same time.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Caption (see page 186)	Specifies a text string that identifies the control to the user.
	Caption (see page 529)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.

	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Down (see page 189)	Tells whether the button is down (selected) or up (unselected).
	DropPoint (see page 532)	This is DropPoint, a member of class TCustomActionControl.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
	Flat (see page 189)	Dictates whether the button should have a 2D look instead of the usual 3D look.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GlyphLayout (see page 532)	Describes where the image should be on the control
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Margins (see page 532)	Describes the boundaries of the control for drawing purposes.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	Selected (see page 533)	Indicates whether the control is currently selected.
R	Separator (see page 533)	Indicates whether the control is a separator.
	ShowCaption (see page 189)	Indicates whether the control's caption should be shown.
R	ShowCaption (see page 533)	Indicates whether the control's caption should be shown.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
	ShowShortCut (see page 190)	Indicates whether the control's short cut should be shown.
R	ShowShortCut (see page 533)	Indicates whether the control's short cut should be shown.
	SmallIcon (see page 534)	Indicates whether the control's small icon should be shown.
	Spacing (see page 534)	Controls (see page 5711) the separation between the control's icon and caption.
	State (see page 190)	Describes the current state of the button.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TextBounds (see page 534)	A bounding rectangle for the control caption.
	TrackButton (see page 190)	Specifies which mouse button activates the button control.
	Transparent (see page 534)	Shows whether the control is transparent or not.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

2.1.7.2 TCustomDropDownButton Methods

The methods of the TCustomDropDownButton class are listed here.

Protected Methods

	Name	Description
	DrawFrame (see page 215)	This is DrawFrame, a member of class TCustomDropDownButton.
	GetPopupClass (see page 215)	This is GetPopupClass, a member of class TCustomDropDownButton.
	MouseDown (see page 216)	OnMouseDown event dispatcher.
	MouseOverDropDown (see page 216)	This is MouseOverDropDown, a member of class TCustomDropDownButton.
	Paint (see page 216)	Render control surface.

Public Methods

	Name	Description
	CalcBounds (see page 214)	Calculates the size of the control.
	Click (see page 215)	Executes the control's function.
	DropDownClick (see page 215)	This is DropDownClick, a member of class TCustomDropDownButton.

2.1.7.2.1 TCustomDropDownButton.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

This method is largely the same as CalcBounds (TCustomActionControl), with one minor exception. The width of the control is set to be one pixel wider. For more information, see the documentation for the inherited method.

See Also

CalcBounds (see page 518)

2.1.7.2.2 TCustomDropDownButton.Click Method

Executes the control's function.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

This override of the method inherited from TCustomActionControl is almost identical to the inherited method. The only difference is that if the button control is being manipulated at design time, extra actions are taken to ensure that the control is properly selected.

See Also

Click (see page 518)

Click (see page 5743)

UsageCount (see page 388)

2.1.7.2.3 TCustomDropDownButton.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TCustomDropDownButton.

2.1.7.2.4 TCustomDropDownButton.DropDownClick Method

Pascal

```
procedure DropDownClick; virtual;
```

C++

```
virtual __fastcall DropDownClick();
```

Description

This is DropDownClick, a member of class TCustomDropDownButton.

2.1.7.2.5 TCustomDropDownButton.GetPopupClass Method

Pascal

```
function GetPopupClass: TCustomActionBarClass; virtual;
```

C++

```
virtual __fastcall TCustomActionBarClass GetPopupClass();
```

Description

This is GetPopupClass, a member of class TCustomDropDownButton.

2.1.7.2.6 TCustomDropDownButton.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove (see page 5773)

MouseUp (see page 5774)

OnMouseDown (see page 5862)

2.1.7.2.7 TCustomDropDownButton.MouseOverDropDown Method

Pascal

```
function MouseOverDropDown: Boolean;
```

C++

```
__fastcall Boolean MouseOverDropDown();
```

Description

This is MouseOverDropDown, a member of class TCustomDropDownButton.

2.1.7.2.8 TCustomDropDownButton.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas (see page 6053)

2.1.7.3 TCustomDropDownButton Properties

The properties of the TCustomDropDownButton class are listed here.

Protected Properties

	Name	Description
	DroppedDown (see page 217)	This is DroppedDown, a member of class TCustomDropDownButton.

2.1.7.3.1 TCustomDropDownButton.DroppedDown Property

Pascal

```
property DroppedDown: Boolean;
```

C++

```
__property Boolean DroppedDown;
```

Description

This is DroppedDown, a member of class TCustomDropDownButton.

2.1.8 TCustomToolScrollBtn Class

Class Hierarchy

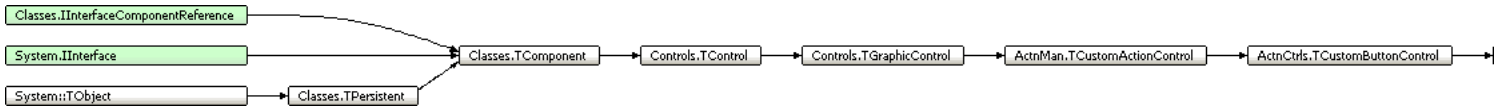

Pascal

```
TCustomToolScrollBtn = class(TCustomUtilityButton);
```

C++

```
class TCustomToolScrollBtn : public TCustomUtilityButton;
```

File

ActnCtrls

Description

This is class ActnCtrls.TCustomToolScrollBtn.

2.1.8.1 TCustomToolScrollBtn Members

The following tables list the members exposed by TCustomToolScrollBtn.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFlat (see page 178)	This is FFlat, a member of class TCustomButtonControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FState (see page 178)	This is FState, a member of class TCustomButtonControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	BeginAutoDrag (see page 517)	Begin Automatic Drag.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CalcLayout (see page 518)	This is CalcLayout, a member of class TCustomActionControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
	CMDesignHitTest (see page 519)	This is CMDesignHitTest, a member of class TCustomActionControl.
	CMDialogChar (see page 180)	This is CMDialogChar, a member of class TCustomButtonControl.
	CMEnabledChanged (see page 180)	This is CMEnabledChanged, a member of class TCustomButtonControl.
	CMFontChanged (see page 519)	This is CMFontChanged, a member of class TCustomActionControl.
	CMItemDropPoint (see page 519)	This is CMItemDropPoint, a member of class TCustomActionControl.
	CMItemSelected (see page 519)	This is CMItemSelected, a member of class TCustomActionControl.
	CMMouseEnter (see page 180)	This is CMMouseEnter, a member of class TCustomButtonControl.
	CMMouseEnter (see page 238)	This is CMMouseEnter, a member of class TCustomUtilityButton.
	CMMouseLeave (see page 180)	This is CMMouseLeave, a member of class TCustomButtonControl.
	CMMouseLeave (see page 239)	This is CMMouseLeave, a member of class TCustomUtilityButton.
	CMTextChanged (see page 180)	This is CMTextChanged, a member of class TCustomButtonControl.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DesignWndProc (see page 519)	This is DesignWndProc, a member of class TCustomActionControl.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
V	DoDragDrop (see page 520)	This is DoDragDrop, a member of class TCustomActionControl.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
V	DoStartDrag (see page 520)	OnStartDrag event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.

	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 521)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawArrows (see page 227)	This is DrawArrows, a member of class TCustomToolScrollBtn.
	DrawBackground (see page 181)	Draws a background image on the control.
	DrawBackground (see page 521)	Draws a background image on the control.
	DrawDesignFocus (see page 521)	This is DrawDesignFocus, a member of class TCustomActionControl.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawDragDropPoint (see page 522)	This is DrawDragDropPoint, a member of class TCustomActionControl.
	DrawFrame (see page 181)	This is DrawFrame, a member of class TCustomButtonControl.
	DrawGlyph (see page 522)	Draws a symbol on the control.
	DrawLargeGlyph (see page 181)	This is DrawLargeGlyph, a member of class TCustomButtonControl.
	DrawLargeGlyph (see page 522)	This is DrawLargeGlyph, a member of class TCustomActionControl.
	DrawSeparator (see page 522)	This is DrawSeparator, a member of class TCustomActionControl.
	DrawShadowedText (see page 523)	This is DrawShadowedText, a member of class TCustomActionControl.
	DrawText (see page 523)	Draws text on the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	GetAction (see page 523)	This is GetAction, a member of class TCustomActionControl.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetImageSize (see page 523)	This is GetImageSize, a member of class TCustomActionControl.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSeparator (see page 524)	This is GetSeparator, a member of class TCustomActionControl.
	GetShowCaption (see page 524)	This is GetShowCaption, a member of class TCustomActionControl.
	GetShowShortCut (see page 181)	This is GetShowShortCut, a member of class TCustomButtonControl.
	GetShowShortCut (see page 524)	This is GetShowShortCut, a member of class TCustomActionControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasGlyph (see page 524)	This is HasGlyph, a member of class TCustomActionControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsChecked (see page 525)	This is IsChecked, a member of class TCustomActionControl.
	IsGrouped (see page 525)	This is IsGrouped, a member of class TCustomActionControl.
	Loaded (see page 182)	Initializes the control after it is loaded from a stream.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.

	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 182)	OnMouseDown event dispatcher.
V	MouseDown (see page 239)	OnMouseDown event dispatcher.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
V	MouseMove (see page 183)	Respond to mouse moving over control area..
	MouseMove (see page 5773)	Respond to mouse moving over control area..
V	MouseUp (see page 183)	OnMouseUp event dispatcher.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	OnDelay (see page 240)	This is OnDelay, a member of class TCustomUtilityButton.
V	Paint (see page 184)	Render control surface.
V	Paint (see page 240)	Render control surface.
V	Paint (see page 526)	Render control surface.
V	Paint (see page 6052)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	PositionChanged (see page 526)	This is PositionChanged, a member of class TCustomActionControl.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetUsageData (see page 526)	This is ResetUsageData, a member of class TCustomActionControl.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
V	SetActionClient (see page 184)	This is SetActionClient, a member of class TCustomButtonControl.
V	SetActionClient (see page 526)	This is SetActionClient, a member of class TCustomActionControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropPoint (see page 527)	This is SetDropPoint, a member of class TCustomActionControl.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
V	SetGlyphLayout (see page 185)	This is SetGlyphLayout, a member of class TCustomButtonControl.
V	SetGlyphLayout (see page 527)	This is SetGlyphLayout, a member of class TCustomActionControl.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetSelected (see page 185)	This is SetSelected, a member of class TCustomButtonControl.
V	SetSelected (see page 527)	This is SetSelected, a member of class TCustomActionControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.

	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateSelection (see page 527)	This is UpdateSelection, a member of class TCustomActionControl.
	UpdateTextBounds (see page 527)	This is UpdateTextBounds, a member of class TCustomActionControl.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMLButtonDblClk (see page 185)	This is WMLButtonDblClk, a member of class TCustomButtonControl.
	WMSetText (see page 528)	This is WMSetText, a member of class TCustomActionControl.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GlyphPos (see page 532)	This is GlyphPos, a member of class TCustomActionControl.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	MouseInControl (see page 189)	This is MouseInControl, a member of class TCustomButtonControl.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnClick (see page 243)	Occurs when the user clicks the control.
	OnClick (see page 535)	Occurs when the user clicks the control.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
V	CalcBounds (see page 179)	Calculates the size of the control.
V	CalcBounds (see page 518)	Calculates the size of the control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Click (see page 179)	Executes the control's function.
V	Click (see page 238)	Executes the control's function.
V	Click (see page 518)	Executes the control's function in response to a mouse click.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 227)	Creates an instance of TCustomButtonControl.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 237)	Destroys an instance of TCustomActionControl.
V	Destroy (see page 516)	Destroys an instance of TCustomActionControl (see page 507).
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
V	DragDrop (see page 520)	Performs actions and generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 524)	Calls the action client's Update method if the control is associated with an action client.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
V	Keyed (see page 525)	Executes the control's function in response to a keyboard action.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 184)	Sets the Left, Top, Width, and Height properties all at once.

V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	ActionBar (see page 528)	The ActionBand object which contains the custom action control
	ActionClient (see page 529)	Refers to the control's action client
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	AllowAllUp (see page 186)	Indicates whether all the buttons in a given group can be up at the same time.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	ArrowSize (see page 240)	This is ArrowSize, a member of class TCustomUtilityButton.
	ArrowType (see page 241)	This is ArrowType, a member of class TCustomUtilityButton.
	AutoScroll (see page 241)	This is AutoScroll, a member of class TCustomUtilityButton.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Caption (see page 186)	Specifies a text string that identifies the control to the user.
	Caption (see page 529)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 241)	Specifies the background color of the control.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Direction (see page 242)	This is Direction, a member of class TCustomUtilityButton.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Down (see page 189)	Tells whether the button is down (selected) or up (unselected).
	DropPoint (see page 532)	This is DropPoint, a member of class TCustomActionControl.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.

R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
	Flat (see page 189)	Dictates whether the button should have a 2D look instead of the usual 3D look.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GlyphLayout (see page 532)	Describes where the image should be on the control
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Margins (see page 532)	Describes the boundaries of the control for drawing purposes.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	RepeatRate (see page 242)	This is RepeatRate, a member of class TCustomUtilityButton.
	Selected (see page 533)	Indicates whether the control is currently selected.
R	Separator (see page 533)	Indicates whether the control is a separator.
	ShowCaption (see page 189)	Indicates whether the control's caption should be shown.
R	ShowCaption (see page 533)	Indicates whether the control's caption should be shown.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
	ShowShortCut (see page 190)	Indicates whether the control's short cut should be shown.
R	ShowShortCut (see page 533)	Indicates whether the control's short cut should be shown.
	SmallIcon (see page 534)	Indicates whether the control's small icon should be shown.
	Spacing (see page 534)	Controls (see page 5711) the separation between the control's icon and caption.
	State (see page 190)	Describes the current state of the button.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TextBounds (see page 534)	A bounding rectangle for the control caption.
	TrackButton (see page 190)	Specifies which mouse button activates the button control.
	Transparent (see page 534)	Shows whether the control is transparent or not.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

2.1.8.2 TCustomToolScrollBtn.Create Constructor

Creates an instance of TCustomButtonControl.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomToolScrollBtn(TComponent * AOwner);
```

Description

Calling Create constructs and initializes an instance of TCustomButtonControl. However, you should never attempt to instantiate a TCustomButtonControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component that is responsible for freeing the button control. It becomes the value of the Owner property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner (see page 2807)

Create (see page 5731)

Destroy

2.1.8.3 TCustomToolScrollBtn Methods

The methods of the TCustomToolScrollBtn class are listed here.

Protected Methods

	Name	Description
	DrawArrows (see page 227)	This is DrawArrows, a member of class TCustomToolScrollBtn.

2.1.8.3.1 TCustomToolScrollBtn.DrawArrows Method

Pascal

```
procedure DrawArrows; override;
```

C++

```
virtual __fastcall DrawArrows();
```

Description

This is DrawArrows, a member of class TCustomToolScrollBtn.

2.1.9 TCustomUtilityButton Class

Class Hierarchy

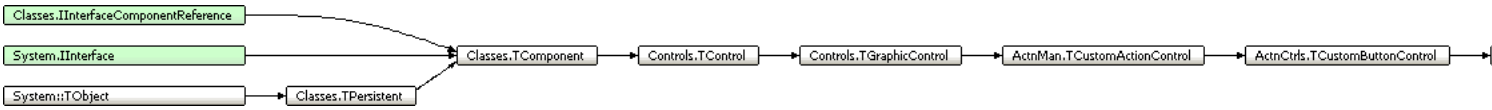

Pascal

```
TCustomUtilityButton = class(TCustomButtonControl);
```

C++

```
class TCustomUtilityButton : public TCustomButtonControl;
```

File

ActnCtrls

Description

This is class ActnCtrls.TCustomUtilityButton.

2.1.9.1 TCustomUtilityButton Members

The following tables list the members exposed by TCustomUtilityButton.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.

	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFlat (see page 178)	This is FFlat, a member of class TCustomButtonControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FState (see page 178)	This is FState, a member of class TCustomButtonControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	BeginAutoDrag (see page 517)	Begin Automatic Drag.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CalcLayout (see page 518)	This is CalcLayout, a member of class TCustomActionControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	CMDesignHitTest (see page 519)	This is CMDesignHitTest, a member of class TCustomActionControl.
	CMDialogChar (see page 180)	This is CMDialogChar, a member of class TCustomButtonControl.
	CMEnabledChanged (see page 180)	This is CMEnabledChanged, a member of class TCustomButtonControl.
	CMFontChanged (see page 519)	This is CMFontChanged, a member of class TCustomActionControl.
	CMItemDropPoint (see page 519)	This is CMItemDropPoint, a member of class TCustomActionControl.
	CMItemSelected (see page 519)	This is CMItemSelected, a member of class TCustomActionControl.
	CMMouseEnter (see page 238)	This is CMMouseEnter, a member of class TCustomUtilityButton.
	CMMouseLeave (see page 239)	This is CMMouseLeave, a member of class TCustomUtilityButton.
	CMTextChanged (see page 180)	This is CMTextChanged, a member of class TCustomButtonControl.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 519)	This is DesignWndProc, a member of class TCustomActionControl.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDragDrop (see page 520)	This is DoDragDrop, a member of class TCustomActionControl.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 520)	OnStartDrag event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 521)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawArrows (see page 239)	This is DrawArrows, a member of class TCustomUtilityButton.
	DrawBackground (see page 181)	Draws a background image on the control.
	DrawBackground (see page 521)	Draws a background image on the control.
	DrawDesignFocus (see page 521)	This is DrawDesignFocus, a member of class TCustomActionControl.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawDragDropPoint (see page 522)	This is DrawDragDropPoint, a member of class TCustomActionControl.
	DrawFrame (see page 181)	This is DrawFrame, a member of class TCustomButtonControl.
	DrawGlyph (see page 522)	Draws a symbol on the control.
	DrawLargeGlyph (see page 181)	This is DrawLargeGlyph, a member of class TCustomButtonControl.
	DrawLargeGlyph (see page 522)	This is DrawLargeGlyph, a member of class TCustomActionControl.
	DrawSeparator (see page 522)	This is DrawSeparator, a member of class TCustomActionControl.
	DrawShadowedText (see page 523)	This is DrawShadowedText, a member of class TCustomActionControl.
	DrawText (see page 523)	Draws text on the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	GetAction (see page 523)	This is GetAction, a member of class TCustomActionControl.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetImageSize (see page 523)	This is GetImageSize, a member of class TCustomActionControl.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.

	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
V	GetSeparator (see page 524)	This is GetSeparator, a member of class TCustomActionControl.
V	GetShowCaption (see page 524)	This is GetShowCaption, a member of class TCustomActionControl.
V	GetShowShortCut (see page 181)	This is GetShowShortCut, a member of class TCustomButtonControl.
V	GetShowShortCut (see page 524)	This is GetShowShortCut, a member of class TCustomActionControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasGlyph (see page 524)	This is HasGlyph, a member of class TCustomActionControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsChecked (see page 525)	This is IsChecked, a member of class TCustomActionControl.
V	IsGrouped (see page 525)	This is IsGrouped, a member of class TCustomActionControl.
V	Loaded (see page 182)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 239)	OnMouseDown event dispatcher.
V	MouseMove (see page 183)	Respond to mouse moving over control area..
	MouseMove (see page 5773)	Respond to mouse moving over control area..
V	MouseUp (see page 183)	OnMouseUp event dispatcher.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	OnDelay (see page 240)	This is OnDelay, a member of class TCustomUtilityButton.
V	Paint (see page 240)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	PositionChanged (see page 526)	This is PositionChanged, a member of class TCustomActionControl.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetUsageData (see page 526)	This is ResetUsageData, a member of class TCustomActionControl.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
V	SetActionClient (see page 184)	This is SetActionClient, a member of class TCustomButtonControl.
V	SetActionClient (see page 526)	This is SetActionClient, a member of class TCustomActionControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropPoint (see page 527)	This is SetDropPoint, a member of class TCustomActionControl.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
V	SetGlyphLayout (see page 185)	This is SetGlyphLayout, a member of class TCustomButtonControl.
V	SetGlyphLayout (see page 527)	This is SetGlyphLayout, a member of class TCustomActionControl.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.

V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetSelected (see page 185)	This is SetSelected, a member of class TCustomButtonControl.
V	SetSelected (see page 527)	This is SetSelected, a member of class TCustomActionControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateSelection (see page 527)	This is UpdateSelection, a member of class TCustomActionControl.
	UpdateTextBounds (see page 527)	This is UpdateTextBounds, a member of class TCustomActionControl.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMLButtonDblClk (see page 185)	This is WMLButtonDblClk, a member of class TCustomButtonControl.
	WMSetText (see page 528)	This is WMSetText, a member of class TCustomActionControl.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GlyphPos (see page 532)	This is GlyphPos, a member of class TCustomActionControl.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	MouseInControl (see page 189)	This is MouseInControl, a member of class TCustomButtonControl.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.

	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnClick (see page 243)	Occurs when the user clicks the control.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
V	CalcBounds (see page 179)	Calculates the size of the control.
V	CalcBounds (see page 518)	Calculates the size of the control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Click (see page 238)	Executes the control's function.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 237)	Creates an instance of TCustomButtonControl.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 237)	Destroys an instance of TCustomActionControl.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
V	DragDrop (see page 520)	Performs actions and generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 524)	Calls the action client's Update method if the control is associated with an action client.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
V	Keyed (see page 525)	Executes the control's function in response to a keyboard action.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 184)	Sets the Left, Top, Width, and Height properties all at once.

V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	ActionBar (see page 528)	The ActionBand object which contains the custom action control
	ActionClient (see page 529)	Refers to the control's action client
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	AllowAllUp (see page 186)	Indicates whether all the buttons in a given group can be up at the same time.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	ArrowSize (see page 240)	This is ArrowSize, a member of class TCustomUtilityButton.
	ArrowType (see page 241)	This is ArrowType, a member of class TCustomUtilityButton.
	AutoScroll (see page 241)	This is AutoScroll, a member of class TCustomUtilityButton.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Caption (see page 186)	Specifies a text string that identifies the control to the user.
	Caption (see page 529)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 241)	Specifies the background color of the control.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Direction (see page 242)	This is Direction, a member of class TCustomUtilityButton.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Down (see page 189)	Tells whether the button is down (selected) or up (unselected).
	DropPoint (see page 532)	This is DropPoint, a member of class TCustomActionControl.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.

R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
	Flat (see page 189)	Dictates whether the button should have a 2D look instead of the usual 3D look.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GlyphLayout (see page 532)	Describes where the image should be on the control
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Margins (see page 532)	Describes the boundaries of the control for drawing purposes.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	RepeatRate (see page 242)	This is RepeatRate, a member of class TCustomUtilityButton.
	Selected (see page 533)	Indicates whether the control is currently selected.
R	Separator (see page 533)	Indicates whether the control is a separator.
	ShowCaption (see page 189)	Indicates whether the control's caption should be shown.
R	ShowCaption (see page 533)	Indicates whether the control's caption should be shown.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
	ShowShortCut (see page 190)	Indicates whether the control's short cut should be shown.
R	ShowShortCut (see page 533)	Indicates whether the control's short cut should be shown.
	SmallIcon (see page 534)	Indicates whether the control's small icon should be shown.
	Spacing (see page 534)	Controls (see page 5711) the separation between the control's icon and caption.
	State (see page 190)	Describes the current state of the button.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TextBounds (see page 534)	A bounding rectangle for the control caption.
	TrackButton (see page 190)	Specifies which mouse button activates the button control.
	Transparent (see page 534)	Shows whether the control is transparent or not.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

2.1.9.2 TCustomUtilityButton.Create Constructor

Creates an instance of TCustomButtonControl.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomUtilityButton(TComponent * AOwner);
```

Description

Calling Create constructs and initializes an instance of TCustomButtonControl. However, you should never attempt to instantiate a TCustomButtonControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component that is responsible for freeing the button control. It becomes the value of the Owner property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner (see page 2807)

Create (see page 5731)

Destroy

2.1.9.3 TCustomUtilityButton.Destroy Destructor

Destroys an instance of TCustomActionControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomUtilityButton();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the graphic control reference is not nil, and only then calls Destroy.

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Frees the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

Create (see page 515)

Free (see page 22300)

2.1.9.4 TCustomUtilityButton Methods

The methods of the TCustomUtilityButton class are listed here.

Protected Methods

	Name	Description
	CMMouseEnter (see page 238)	This is CMMouseEnter, a member of class TCustomUtilityButton.
	CMMouseLeave (see page 239)	This is CMMouseLeave, a member of class TCustomUtilityButton.
V	DrawArrows (see page 239)	This is DrawArrows, a member of class TCustomUtilityButton.
V	MouseDown (see page 239)	OnMouseDown event dispatcher.
	OnDelay (see page 240)	This is OnDelay, a member of class TCustomUtilityButton.
V	Paint (see page 240)	Render control surface.

Public Methods

	Name	Description
V	Click (see page 238)	Executes the control's function.

2.1.9.4.1 TCustomUtilityButton.Click Method

Executes the control's function.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

This override of the method inherited from TCustomActionControl is almost identical to the inherited method. The only difference is that if the button control is being manipulated at design time, extra actions are taken to ensure that the control is properly selected.

See Also

Click (see page 518)

Click (see page 5743)

UsageCount (see page 388)

2.1.9.4.2 TCustomUtilityButton.CMMouseEnter Method

Pascal

```
procedure CMMouseEnter(var Message: TMessage); message CM_MOUSEENTER;
```

C++

```
__fastcall CMMouseEnter(TMessage Message);
```

Description

This is CMMouseEnter, a member of class TCustomUtilityButton.

2.1.9.4.3 TCustomUtilityButton.CMMouseLeave Method

Pascal

```
procedure CMMouseLeave(var Message: TMessage); message CM_MOUSELEAVE;
```

C++

```
__fastcall CMMouseLeave(TMessage Message);
```

Description

This is CMMouseLeave, a member of class TCustomUtilityButton.

2.1.9.4.4 TCustomUtilityButton.DrawArrows Method

Pascal

```
procedure DrawArrows; virtual;
```

C++

```
virtual __fastcall DrawArrows();
```

Description

This is DrawArrows, a member of class TCustomUtilityButton.

2.1.9.4.5 TCustomUtilityButton.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove (see page 5773)

MouseUp (see page 5774)

OnMouseDown (see page 5862)

2.1.9.4.6 TCustomUtilityButton.OnDelay Method

Pascal

```
procedure OnDelay(Sender: TObject);
```

C++

```
__fastcall OnDelay(TObject * Sender);
```

Description

This is OnDelay, a member of class TCustomUtilityButton.

2.1.9.4.7 TCustomUtilityButton.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas (see page 6053)

2.1.9.5 TCustomUtilityButton Properties

The properties of the TCustomUtilityButton class are listed here.

Public Properties

	Name	Description
	ArrowSize (see page 240)	This is ArrowSize, a member of class TCustomUtilityButton.
	ArrowType (see page 241)	This is ArrowType, a member of class TCustomUtilityButton.
	AutoScroll (see page 241)	This is AutoScroll, a member of class TCustomUtilityButton.
	Color (see page 241)	Specifies the background color of the control.
	Direction (see page 242)	This is Direction, a member of class TCustomUtilityButton.
	RepeatRate (see page 242)	This is RepeatRate, a member of class TCustomUtilityButton.

2.1.9.5.1 TCustomUtilityButton.ArrowSize Property

Pascal

```
property ArrowSize: Integer;
```

C++

```
__property int ArrowSize;
```

Description

This is ArrowSize, a member of class TCustomUtilityButton.

2.1.9.5.2 TCustomUtilityButton.ArrowType Property

Pascal

```
property ArrowType: TArrowType;
```

C++

```
__property TArrowType ArrowType;
```

Description

This is ArrowType, a member of class TCustomUtilityButton.

2.1.9.5.3 TCustomUtilityButton.AutoScroll Property

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

This is AutoScroll, a member of class TCustomUtilityButton.

2.1.9.5.4 TCustomUtilityButton.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

2.1.9.5.5 TCustomUtilityButton.Direction Property

Pascal

```
property Direction: TScrollDirection;
```

C++

```
__property TScrollDirection Direction;
```

Description

This is Direction, a member of class TCustomUtilityButton.

2.1.9.5.6 TCustomUtilityButton.RepeatRate Property

Pascal

```
property RepeatRate: Integer;
```

C++

```
__property int RepeatRate;
```

Description

This is RepeatRate, a member of class TCustomUtilityButton.

2.1.9.6 TCustomUtilityButton Events

The events of the TCustomUtilityButton class are listed here.

Public Events

	Name	Description
	OnClick (see page 243)	Occurs when the user clicks the control.

2.1.9.6.1 TCustomUtilityButton.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
```

```
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

2.2 Types

The following table lists types in this documentation.

Types

Name	Description
TCustomToolScrollBtnClass (see page 245)	This is type ActnCtrls.TCustomToolScrollBtnClass.

2.2.1 ActnCtrls.TCustomToolScrollBtnClass Type

Pascal

```
TCustomToolScrollBtnClass = class of TCustomToolScrollBtn;
```

C++

```
TCustomToolScrollBtn TCustomToolScrollBtnClass;
```

File

ActnCtrls

Description

This is type ActnCtrls.TCustomToolScrollBtnClass.

3 ActnList Namespace

This is namespace ActnList.

Topics

Name	Description
Classes (see page 249)	The following table lists classes in this documentation.
Functions (see page 334)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 337)	The following table lists structs, records, enums in this documentation.
Types (see page 338)	The following table lists types in this documentation.
Constants (see page 341)	The following table lists constants in this documentation.

Classes

	Name	Description
	TAction (see page 249)	TAction is the generic class for action objects that implement user commands across a range of different controls.
	TActionLink (see page 265)	TActionLink is the base class for action links that connect actions to commonly used properties of VCL objects, components and controls.
	TActionList (see page 280)	TActionList maintains a list of actions that can be used by components and controls, such as menu items and buttons.
	TActionListEnumerator (see page 287)	This is class ActnList.TActionListEnumerator.
	TContainedAction (see page 289)	TContainedAction is the base class for actions that can be grouped by category and that can appear in action lists or action managers.
	TCustomAction (see page 297)	TCustomAction is the base class for actions meant to be used with menu items and controls.
	TCustomActionList (see page 315)	TCustomActionList is the base class for action list objects that manage a set of actions.
	TShortCutList (see page 328)	TShortCutList manages an array of keyboard shortcuts.

Constants

Name	Description
CreateActionProc (see page 341)	This is constant ActnList.CreateActionProc.
EnumRegisteredActionsProc (see page 341)	This is constant ActnList.EnumRegisteredActionsProc.
RegisterActionsProc (see page 342)	This is constant ActnList.RegisterActionsProc.
UnRegisterActionsProc (see page 342)	This is constant ActnList.UnRegisterActionsProc.

Functions

	Name	Description
	CreateAction (see page 334)	Creates an action of a specified type.

	EnumRegisteredActions (see page 334)	Iterates through the list of registered actions, passing them to a user-defined callback.
	RegisterActions (see page 335)	Registers action classes so that they appear in the list of predefined actions in the action list editor or customize dialog.
	UnRegisterActions (see page 336)	Unregisters a set of actions.

Structs, Records, Enums

	Name	Description
	TActionListState (see page 337)	This is record ActnList.TActionListState.

Types

Name	Description
TActionEvent (see page 338)	This is type ActnList.TActionEvent.
TActionLinkClass (see page 338)	This is type ActnList.TActionLinkClass.
TContainedActionClass (see page 339)	This is type ActnList.TContainedActionClass.
TEnumActionProc (see page 339)	This is type ActnList.TEnumActionProc.
THintEvent (see page 339)	THintEvent is the type of the OnHint event handler.

3.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TAction (see page 249)	TAction is the generic class for action objects that implement user commands across a range of different controls.
	TActionLink (see page 265)	TActionLink is the base class for action links that connect actions to commonly used properties of VCL objects, components and controls.
	TActionList (see page 280)	TActionList maintains a list of actions that can be used by components and controls, such as menu items and buttons.
	TActionListEnumerator (see page 287)	This is class ActnList.TActionListEnumerator.
	TContainedAction (see page 289)	TContainedAction is the base class for actions that can be grouped by category and that can appear in action lists or action managers.
	TCustomAction (see page 297)	TCustomAction is the base class for actions meant to be used with menu items and controls.
	TCustomActionList (see page 315)	TCustomActionList is the base class for action list objects that manage a set of actions.
	TShortCutList (see page 328)	TShortCutList manages an array of keyboard shortcuts.

3.1.1 TAction Class

TAction is the generic class for action objects that implement user commands across a range of different controls.

Class Hierarchy

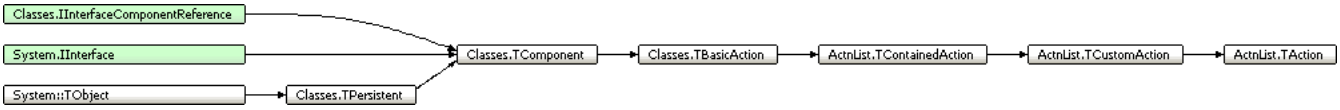

Pascal

```
TAction = class(TCustomAction);
```

C++

```
class TAction : public TCustomAction;
```

File

ActnList

Description

TAction is a generic action component. It can be added to action managers or action lists when there is no predefined action class that implements the desired response to user commands. Unlike predefined action classes, which have built-in methods that respond when users click the client controls, TAction has no built-in response to user commands. Instead, when using TAction, you can provide the response when the action "fires" by writing an OnExecute (see page 261) event handler, and configure the properties of TAction to reflect current conditions by writing an OnUpdate (see page 263) event handler.

Action objects are used to centralize the response to user commands (actions) and to represent user interface elements in applications that use action bands.

TAction can also act as a base class for predefined action classes. You can derive from TAction if you wish to retain the

published scope of TCustomAction (see page 297) members, or derive from TCustomAction (see page 297) if you wish to modify their scope.

For predefined actions that augment the behavior of TAction, see the action classes in the QStdActns unit.

For predefined actions that augment the behavior of TAction, see the action classes in the StdActns (see page 20505) , DBActns (see page 7649) , and ExtActns (see page 11829) units.

See Also

TActionLink (see page 265)

TActionList (see page 280)

TCustomAction (see page 297)

TActionManager (see page 432)

TActionMainMenuBar (see page 585)

TActionToolBar (see page 38)

Creating Toolbars and Menus

Demo Programs: Actions

What Happens When an Action Fires

How Actions Find Their Targets

Organizing Actions for Toolbars and Menus

Predefined Action Classes

Registering Actions

Setting Up Action Bands

Setting Up Action Lists

Updating Actions

Using Action Lists

What Is an Action?

Writing Action Components

3.1.1.1 TAction Members

The following tables list the members exposed by TAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.

	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

3.1.1.2 TAction.Create Constructor

Creates and initializes a TAction (see page 249) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TAction(TComponent * AOwner);
```

Description

Applications do not usually create action objects explicitly. Actions are created automatically when you add them to an action manager or action list component at design time. When instantiating TAction (see page 249) at runtime, assign a TActionList (see page 280) or TActionManager component to the new action's ActionList (see page 295) property after calling Create.

See Also

ActionList (see page 295)

3.1.1.3 TAction Properties

The properties of the TAction class are listed here.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

3.1.1.3.1 TAction.AutoCheck Property

Controls whether the Checked property toggles when the action executes.

Pascal

```
property AutoCheck: Boolean;
```

C++

```
__property Boolean AutoCheck;
```

Description

AutoCheck causes the checked property to toggle when the action executes. This allows the action's Checked property to remain in sync with the client's Checked property (or equivalent). If the client is a menu item (where the Checked property does not necessarily toggle), AutoCheck is propagated to the AutoCheck property of the menu item.

See Also

Checked (see page 308)

Execute (see page 304)

AutoCheck (see page 16134)

3.1.1.3.2 TAction.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

3.1.1.3.3 TAction.Checked Property

Indicates whether client controls and menu items appear checked.

Pascal

```
property Checked: Boolean;
```

C++

```
__property Boolean Checked;
```

Description

Checked specifies the checked state for the action. The value of Checked is propagated to client controls and menu items.

Note: If the action has a GroupIndex value greater than 0, then setting Checked to true causes all other actions in the group to have their Checked property set to false.

Note: Use the AutoCheck property to ensure that the action's Checked property stays synchronized with client's Checked properties.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

GroupIndex (see page 311)

AutoCheck (see page 308)

3.1.1.3.4 TAction.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

3.1.1.3.5 TAction.GroupIndex Property

Indicates whether the action acts like a radio button with other actions in the same action list.

Pascal

```
property GroupIndex: Integer;
```

C++

```
__property int GroupIndex;
```

Description

GroupIndex is used to define groups of actions that act like radio buttons. When GroupIndex is greater than 0, it identifies the

group to which the action belongs. When the Checked property of any action in that group is set to true, the Checked property of all other actions in the group is set to false. That is, only one action in the group can be checked at a time.

Note: All actions in a group must be listed by the same action list or action manager.

See Also

Checked (see page 308)

GroupIndex (see page 2360)

3.1.1.3.6 TAction.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

3.1.1.3.7 TAction.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be

associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

3.1.1.3.8 TAction.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

3.1.1.3.9 TAction.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

3.1.1.3.10 TAction.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

3.1.1.3.11 TAction.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

3.1.1.3.12 TAction.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

3.1.1.3.13 TAction.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

3.1.1.4 TAction Events

The events of the TAction class are listed here.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

3.1.1.4.1 TAction.OnExecute Event

Occurs when the client event that is linked to it fires.

Pascal

```
property OnExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExecute;
```

Description

Write an OnExecute event handler when you want to respond when the user triggers the client object's default event (typically an OnClick event). For most target clients, OnExecute is triggered on a Click method, and is associated with the OnClick event.

OnExecute also occurs when the user types the shortcut (or one of the secondary shortcuts) associated with the action or its client.

Warning: If you assign an OnExecute event handler to a predefined action, the default behavior of that action will not occur.

C++ Examples:

```
/*
This example requires a TApplicationEvents, a TActionList
and a TListBox in the form.  Select the TApplicationEvents,
double click on the OnActionExecute event and add the
following code to the handler.  Create an TAction named
```

```
MyAction in a TActionList.  The TActionList must have no
OnExecute event handler, and the TAction has an OnExecute
event handler called MyActionExecute.  Then set an object's
(such as a button) Action to MyAction.  The Handled
parameter of the event handler returns false by default. If
the handler handles the event, it should change Handled to
true, thereby preventing further attempts to handle the
action.
*/
void __fastcall TAppEventForm::ApplicationEventsActionExecute(
      TBasicAction *Action, bool &Handled)
{
    lbOther->Items->Add("Event OnActionExecute");
}

void __fastcall TAppEventForm::ActionExecute(TObject *Sender)
{
    ShowMessage("Action executed");
}
/*
When a client component or control is clicked, the OnExecute
event occurs for it’s associated action. For example, the
following code illustrates the OnExecute event handler for
an action that toggles the visibility of a toolbar when the
action is executed. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TActionList must have no OnExecute event handler, and the
TAction has an OnExecute event handler called Action1Execute.
Then set an object's (such as a button or menu item) Action
to Action1.  Set the TToolbar color to something that will
make it obvious when it is visible.  Also remember to uncheck
Project|Options|Applications|Enable runtime themes.
*/
void __fastcall TMainForm::Action1Execute(TObject *Sender)
{
  // Toggle Toolbar1's visibility
  ToolBar1->Visible = !ToolBar1->Visible;
  if (ToolBar1->Visible)
    lbOther->Items->Add("Event Action1Execute: toolbar visible");
  else
    lbOther->Items->Add("Event Action1Execute: toolbar not visible");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents, a TActionList
and a TListBox in the form.  Select the TApplicationEvents,
double click on the OnActionExecute event and add the
following code to the handler.  Create an TAction named
MyAction in a TActionList.  The TActionList must have no
OnExecute event handler, and the TAction has an OnExecute
event handler called MyActionExecute.  Then set an object's
(such as a button) Action to MyAction.  The Handled
parameter of the event handler returns false by default. If
the handler handles the event, it should change Handled to
true, thereby preventing further attempts to handle the
action.
}
procedure TMainForm.ApplicationEventsActionExecute(Action: TBasicAction;
  var Handled: Boolean);
begin
  lbOther.Items.Add('Event OnActionExecute');
end;

procedure TMainForm.MyActionExecute(Sender: TObject);
```

```
begin
  ShowMessage('Action executed');
end;
{
When a client component or control is clicked, the OnExecute
event occurs for it’s associated action. For example, the
following code illustrates the OnExecute event handler for
an action that toggles the visibility of a toolbar when the
action is executed. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TActionList must have no OnExecute event handler, and the
TAction has an OnExecute event handler called Action1Execute.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Execute(Sender: TObject);
begin
    { Toggle Toolbar1’s visibility }
    ToolBar1.Visible := not ToolBar1.Visible;
  if ToolBar1.Visible then
    lbOther.Items.Add('Event Action1Execute: toolbar visible')
  else
    lbOther.Items.Add('Event Action1Execute: toolbar not visible')
end;
```

See Also

OnClick (see page 5851)

ShortCut (see page 314)

3.1.1.4.2 TAction.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

3.1.1.4.3 TAction.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

3.1.2 TActionLink Class

TActionLink is the base class for action links that connect actions to commonly used properties of VCL objects, components and controls.

Class Hierarchy

Pascal

```
TActionLink = class(TBasicActionLink);
```

C++

```
class TActionLink : public TBasicActionLink;
```

File

ActnList

Description

Most applications do not work directly with TActionLink objects. Instead, the clients of actions already include a specialized TActionLink descendant that manages the relationship between the client object and an action. TActionLink introduces methods to manage the relationship between properties on an action object and the corresponding properties of the client object. Descendant classes override these methods, implementing them for the particular properties of a class of client.

Component writers can use TActionLink as a base class for new action link classes that manage the relationship between actions and families of client objects that are not controls or menu items.

TActionLink introduces interfaces for methods that support the following properties, most of which are linked with properties of the same name:

Caption

Checked

Enabled

HelpContext

Hint

ImageIndex

ShortCut

Visible

See Also

TAction (see page 249)

TControl (see page 5724)

Creating Toolbars and Menus

Demo Programs: Actions

What Happens When an Action Fires

How Actions Find Their Targets

Organizing Actions for Toolbars and Menus

Predefined Action Classes

Registering Actions

Setting Up Action Bands

Updating Actions

Using Action Lists

What Is an Action?

Writing Action Components

3.1.2.1 TActionLink Members

The following tables list the members exposed by TActionLink.

Protected Fields

	Name	Description
	FAction (see page 2722)	This is FAction, a member of class TBasicActionLink.

Protected Methods

	Name	Description
	AssignClient (see page 2723)	This is AssignClient, a member of class TBasicActionLink.
	Change (see page 2723)	This is Change, a member of class TBasicActionLink.
	IsCaptionLinked (see page 268)	Indicates whether the Caption properties for the action and client are linked.
	IsCheckedLinked (see page 269)	Indicates whether the Checked properties for the action and client are linked.
	IsEnabledLinked (see page 269)	Indicates whether the Enabled properties for the action and client are linked.
	IsGroupIndexLinked (see page 270)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsHelpContextLinked (see page 270)	Indicates whether the HelpContext properties for the action and client controls are linked.
	IsHelpLinked (see page 271)	Indicates whether the help properties for the action and client are linked.
	IsHintLinked (see page 271)	Indicates whether the Hint properties for the action and client are linked.
	IsImageIndexLinked (see page 272)	Indicates whether the image index for the action and the client are linked.
	IsOnExecuteLinked (see page 2723)	This is IsOnExecuteLinked, a member of class TBasicActionLink.
	IsShortCutLinked (see page 272)	Indicates whether the ShortCut properties for the action and client are linked.
	IsVisibleLinked (see page 273)	Indicates whether the Visible properties for the action and client are linked.
	SetAction (see page 2724)	This is SetAction, a member of class TBasicActionLink.
	SetAutoCheck (see page 274)	Introduces an interface for setting the AutoCheck property of a client.
	SetCaption (see page 274)	Introduces an interface for setting the caption on a client.
	SetChecked (see page 274)	Introduces an interface for setting the checked state of a client.
	SetEnabled (see page 275)	Introduces an interface for setting the enabled state of a client.
	SetGroupIndex (see page 275)	Introduces an interface for setting the GroupIndex property of a client.
	SetHelpContext (see page 276)	Introduces an interface for setting the help context of a client.
	SetHelpKeyword (see page 276)	Introduces an interface for setting the help keyword of a client.
	SetHelpType (see page 277)	Introduces an interface for setting the HelpType property of a client.
	SetHint (see page 278)	Introduces an interface for setting the hint for a client.
	SetImageIndex (see page 278)	Introduces an interface for setting the image index for a client.
	SetOnExecute (see page 2724)	This is SetOnExecute, a member of class TBasicActionLink.
	SetShortCut (see page 279)	Introduces an interface for setting the shortcut for a client.
	SetVisible (see page 279)	Introduces an interface for setting the visible state of a client.

Public Events

	Name	Description
	OnChange (see page 2725)	Occurs when an OnChange event is triggered on the action.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2721)	Instantiates and initializes a TBasicActionLink (see page 2720) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2722)	Destroys an instance of an action link object.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 2723)	Invokes the action's Execute method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Update (see page 2724)	Invokes the action's Update method.

Public Properties

	Name	Description
	Action (see page 2725)	Indicates which action the link connects to its client.

3.1.2.2 TActionLink Methods

The methods of the TActionLink class are listed here.

Protected Methods

	Name	Description
	IsCaptionLinked (see page 268)	Indicates whether the Caption properties for the action and client are linked.
	IsCheckedLinked (see page 269)	Indicates whether the Checked properties for the action and client are linked.

	IsEnabledLinked (see page 269)	Indicates whether the Enabled properties for the action and client are linked.
	IsGroupIndexLinked (see page 270)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsHelpContextLinked (see page 270)	Indicates whether the HelpContext properties for the action and client controls are linked.
	IsHelpLinked (see page 271)	Indicates whether the help properties for the action and client are linked.
	IsHintLinked (see page 271)	Indicates whether the Hint properties for the action and client are linked.
	IsImageIndexLinked (see page 272)	Indicates whether the image index for the action and the client are linked.
	IsShortCutLinked (see page 272)	Indicates whether the ShortCut properties for the action and client are linked.
	IsVisibleLinked (see page 273)	Indicates whether the Visible properties for the action and client are linked.
	SetAutoCheck (see page 274)	Introduces an interface for setting the AutoCheck property of a client.
	SetCaption (see page 274)	Introduces an interface for setting the caption on a client.
	SetChecked (see page 274)	Introduces an interface for setting the checked state of a client.
	SetEnabled (see page 275)	Introduces an interface for setting the enabled state of a client.
	SetGroupIndex (see page 275)	Introduces an interface for setting the GroupIndex property of a client.
	SetHelpContext (see page 276)	Introduces an interface for setting the help context of a client.
	SetHelpKeyword (see page 276)	Introduces an interface for setting the help keyword of a client.
	SetHelpType (see page 277)	Introduces an interface for setting the HelpType property of a client.
	SetHint (see page 278)	Introduces an interface for setting the hint for a client.
	SetImageIndex (see page 278)	Introduces an interface for setting the image index for a client.
	SetShortCut (see page 279)	Introduces an interface for setting the shortcut for a client.
	SetVisible (see page 279)	Introduces an interface for setting the visible state of a client.

3.1.2.2.1 TActionLink.IsCaptionLinked Method

Indicates whether the Caption properties for the action and client are linked.

Pascal

```
function IsCaptionLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsCaptionLinked();
```

Description

TActionLink (see page 265) descendants call IsCaptionLinked to determine whether they need to propagate changes to the Caption property of the action so that they are reflected by the client object. When IsCaptionLinked returns true, the action controls the Caption property: Changes to the Caption property of the action are reflected by changes to the corresponding property of the client object. When IsCaptionLinked returns false, the client object does not reflect the Caption property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink (see page 265), IsCaptionLinked merely checks that the Action property specifies a descendant of TCustomAction (see page 297) (that is, it is an action that has a Caption property). Descendant classes override IsCaptionLinked to determine whether the Caption property is linked with the appropriate client property.

See Also

Caption (see page 308)

Caption (see page 5806)

SetCaption (see page 274)

Action (see page 2725)

3.1.2.2.2 TActionLink.IsCheckedLinked Method

Indicates whether the Checked properties for the action and client are linked.

Pascal

```
function IsCheckedLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsCheckedLinked();
```

Description

TActionLink (see page 265) descendants call IsCheckedLinked to determine whether they need to propagate changes to the Checked property of the action so that they are reflected by the client object. When IsCheckedLinked returns true, the action controls the Checked property: Changes to the Checked property of the action are reflected by changes to the corresponding property of the client object. When IsCheckedLinked returns false, the client object does not reflect the Checked property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink (see page 265), IsCheckedLinked merely checks that the Action property specifies a descendant of TCustomAction (see page 297) (that is, it is an action that has a Checked property). Descendant classes override IsCheckedLinked to determine whether the Checked property is linked with the appropriate client property.

See Also

Checked (see page 308)

SetChecked (see page 274)

Action (see page 2725)

3.1.2.2.3 TActionLink.IsEnabledLinked Method

Indicates whether the Enabled properties for the action and client are linked.

Pascal

```
function IsEnabledLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsEnabledLinked();
```

Description

TActionLink (see page 265) descendants call IsEnabledLinked to determine whether they need to propagate changes to the Enabled property of the action so that they are reflected by the client object. When IsEnabledLinked returns true, the action controls the Enabled property: Changes to the Enabled property of the action are reflected by changes to the corresponding property of the client object. When IsEnabledLinked returns false, the client object does not reflect the Enabled property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink (see page 265), IsEnabledLinked merely checks that the Action property specifies a descendant of TCustomAction (see page 297) (that is, it is an action that has an Enabled property). Descendant classes override IsEnabledLinked to determine whether the Enabled property is linked with the appropriate client property.

See Also

Enabled (see page 310)

Enabled (see page 5821)

SetEnabled (see page 275)

Action (see page 2725)

3.1.2.2.4 TActionLink.IsGroupIndexLinked Method

Indicates whether the GroupIndex property for the action controls client behavior.

Pascal

```
function IsGroupIndexLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsGroupIndexLinked();
```

Description

TActionLink (see page 265) descendants can call IsGroupIndexLinked to determine whether they need to propagate changes to the GroupIndex property of the action so that they are reflected by the client object. When IsGroupIndexLinked returns true, the action controls the GroupIndex property: Changes to the GroupIndex property of the action are reflected in the client object. When IsGroupIndexLinked returns false, the client object is not influenced by the GroupIndex property of its action.

As implemented in TActionLink (see page 265), IsGroupIndexLinked merely checks that the Action property specifies a descendant of TCustomAction (see page 297), and if so, returns true.

See Also

GroupIndex (see page 311)

SetGroupIndex (see page 275)

Action (see page 2725)

3.1.2.2.5 TActionLink.IsHelpContextLinked Method

Indicates whether the HelpContext properties for the action and client controls are linked.

Pascal

```
function IsHelpContextLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsHelpContextLinked();
```

Description

TActionLink (see page 265) descendants can call IsHelpContextLinked to determine whether they need to propagate changes to the HelpContext property of the action so that they are reflected by the client object. When IsHelpContextLinked returns true, the action controls the HelpContext property: Changes to the HelpContext property of the action are reflected in the client object. When IsHelpContextLinked returns false, the client object is not influenced by the HelpContext property of its action.

As implemented in TActionLink (see page 265), IsHelpContextLinked merely checks that the Action property specifies a descendant of TCustomAction (see page 297), and if so, returns true.

See Also

HelpContext (see page 311)

SetHelpContext (see page 276)

Action (see page 2725)

IsHelpLinked (see page 271)

3.1.2.2.6 TActionLink.IsHelpLinked Method

Indicates whether the help properties for the action and client are linked.

Pascal

```
function IsHelpLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsHelpLinked();
```

Description

TActionLink (see page 265) descendants call IsHelpLinked to determine whether they need to propagate changes to the HelpContext, HelpKeyword, and HelpType properties of the action so that they are reflected by client objects. When IsHelpLinked returns true, the action controls these properties: Changes to any of the help properties of the action are reflected by changes to the corresponding property of the client object. When IsHelpLinked returns false, the client object does not reflect the help properties of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink (see page 265), IsHelpLinked merely checks that the Action property specifies a descendant of TCustomAction (see page 297) (that is, it is an action that has HelpContext, HelpKeyword, and HelpType properties). Descendant classes override IsHelpLinked to determine whether these properties are linked with the appropriate client property.

See Also

HelpContext (see page 311)

HelpKeyword (see page 311)

HelpType (see page 312)

HelpKeyword (see page 5826)

HelpType (see page 5827)

HelpType

SetHelpContext (see page 276)

SetHelpKeyword (see page 276)

SetHelpType (see page 277)

Action (see page 2725)

3.1.2.2.7 TActionLink.IsHintLinked Method

Indicates whether the Hint properties for the action and client are linked.

Pascal

```
function IsHintLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsHintLinked();
```

Description

TActionLink (see page 265) descendants call IsHintLinked to determine whether they need to propagate changes to the Hint property of the action so that they are reflected by the client object. When IsHintLinked returns true, the action controls the Hint property: Changes to the Hint property of the action are reflected by changes to the corresponding property of the client object.

When IsHintLinked returns false, the client object does not reflect the Hint property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink (see page 265), IsHintLinked merely checks that the Action property specifies a descendant of TCustomAction (see page 297) (that is, it is an action that has a Hint property). Descendant classes override IsHintLinked to determine whether the Hint property is linked with the appropriate client property.

See Also

Hint (see page 312)

Hint (see page 5827)

Hint (see page 16148)

SetHint (see page 278)

Action (see page 2725)

3.1.2.2.8 TActionLink.IsImageIndexLinked Method

Indicates whether the image index for the action and the client are linked.

Pascal

```
function IsImageIndexLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsImageIndexLinked();
```

Description

TActionLink (see page 265) descendants call IsImageIndexLinked to determine whether they need to propagate changes to the ImageIndex property of the action so that they are reflected by the client object. When IsImageIndexLinked returns true, the action controls the ImageIndex property: Changes to the ImageIndex property of the action are reflected by changes to the corresponding property of the client object. When IsImageIndexLinked returns false, the client object does not reflect the ImageIndex property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink (see page 265), IsImageIndexLinked merely checks that the Action property specifies a descendant of TCustomAction (see page 297) (that is, it is an action that has an ImageIndex property). Descendant classes override IsImageIndexLinked to determine whether the ImageIndex property is linked with the appropriate client property.

See Also

ImageIndex (see page 313)

ImageIndex (see page 16148)

SetImageIndex (see page 278)

Action (see page 2725)

3.1.2.2.9 TActionLink.IsShortCutLinked Method

Indicates whether the ShortCut properties for the action and client are linked.

Pascal

```
function IsShortCutLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsShortCutLinked();
```

Description

TActionLink (see page 265) descendants call IsShortCutLinked to determine whether they need to propagate changes to the ShortCut property of the action so that they are reflected by the client object. When IsShortCutLinked returns true, the action controls the ShortCut property: Changes to the ShortCut property of the action are reflected by changes to the corresponding property of the client object. When IsShortCutLinked returns false, the client object does not reflect the ShortCut property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink (see page 265), IsShortCutLinked merely checks that the Action property specifies a descendant of TCustomAction (see page 297) (that is, it is an action that has a ShortCut property). Descendant classes override IsShortCutLinked to determine whether the ShortCut property is linked with the appropriate client property.

See Also

ShortCut (see page 314)

ShortCut (see page 16152)

SetShortCut (see page 279)

Action (see page 2725)

3.1.2.2.10 TActionLink.IsVisibleLinked Method

Indicates whether the Visible properties for the action and client are linked.

Pascal

```
function IsVisibleLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsVisibleLinked();
```

Description

TActionLink (see page 265) descendants call IsVisibleLinked to determine whether they need to propagate changes to the Visible property of the action so that they are reflected by the client object. When IsVisibleLinked returns true, the action controls the Visible property: Changes to the Visible property of the action are reflected by changes to the corresponding property of the client object. When IsVisibleLinked returns false, the client object does not reflect the Visible property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink (see page 265), IsVisibleLinked merely checks that the Action property specifies a descendant of TCustomAction (see page 297) (that is, it is an action that has a Visible property). Descendant classes override IsVisibleLinked to determine whether the Visible property is linked with the appropriate client property.

See Also

Visible (see page 314)

Visible (see page 5847)

SetShortCut (see page 279)

Action (see page 2725)

3.1.2.2.11 TActionLink.SetAutoCheck Method

Introduces an interface for setting the AutoCheck property of a client.

Pascal

```
procedure SetAutoCheck(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetAutoCheck(Boolean Value);
```

Description

The associated action calls SetAutoCheck when its AutoCheck property changes so that the action link can propagate the new value to the client object.

Value specifies the new value of the AutoCheck property.

As implemented in TActionLink (see page 265), SetAutoCheck does nothing. Descendant classes override SetAutoCheck to set the client property that corresponds to the action's AutoCheck property.

See Also

AutoCheck (see page 308)

AutoCheck (see page 16134)

Action (see page 2725)

3.1.2.2.12 TActionLink.SetCaption Method

Introduces an interface for setting the caption on a client.

Pascal

```
procedure SetCaption(const Value: string); virtual;
```

C++

```
virtual __fastcall SetCaption(const AnsiString Value);
```

Description

The associated action calls SetCaption when its Caption property changes so that the action link can propagate the new Caption value to the client object.

Value specifies the string to which the caption should be set.

As implemented in TActionLink (see page 265), SetCaption does nothing. Descendant classes override SetCaption to set the client property that corresponds to the action's Caption property if the IsCaptionLinked (see page 268) method returns true.

See Also

Caption (see page 308)

Caption (see page 5806)

IsCaptionLinked (see page 268)

Action (see page 2725)

3.1.2.2.13 TActionLink.SetChecked Method

Introduces an interface for setting the checked state of a client.

Pascal

```
procedure SetChecked(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetChecked(Boolean Value);
```

Description

The associated action calls SetChecked when its Checked property changes so that the action link can propagate the new Checked value to the client object.

Value specifies the new value of the Checked property.

As implemented in TActionLink (see page 265), SetChecked does nothing. Descendant classes override SetChecked to set the client property that corresponds to the action's Checked property if the IsCheckedLinked (see page 269) method returns true.

See Also

Checked (see page 308)

Action (see page 2725)

IsCheckedLinked (see page 269)

3.1.2.2.14 TActionLink.SetEnabled Method

Introduces an interface for setting the enabled state of a client.

Pascal

```
procedure SetEnabled(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetEnabled(Boolean Value);
```

Description

The associated action calls SetEnabled when its Enabled property changes so that the action link can propagate the new Enabled value to the client object.

Value specifies the new value of the Enabled property.

As implemented in TActionLink (see page 265), SetEnabled does nothing. Descendant classes override SetEnabled to set the client property that corresponds to the action's Enabled property if the IsEnabledLinked (see page 269) method returns true.

See Also

Enabled (see page 310)

Enabled (see page 5821)

Action (see page 2725)

IsEnabledLinked (see page 269)

3.1.2.2.15 TActionLink.SetGroupIndex Method

Introduces an interface for setting the GroupIndex property of a client.

Pascal

```
procedure SetGroupIndex(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetGroupIndex(int Value);
```

Description

The associated action calls SetGroupIndex when its GroupIndex property changes so that the action link can propagate the new GroupIndex value to the client object.

Value specifies the new value of the GroupIndex property.

As implemented in TActionLink (see page 265), SetGroupIndex does nothing. It is not strictly necessary for descendant classes to override SetGroupIndex to set the client property that corresponds to the action's GroupIndex property, because the action itself implements the GroupIndex feature. That is, when an action's Checked property is set to true, it unchecks other actions in the same action list if they have the same GroupIndex value.

See Also

GroupIndex (see page 311)

GroupIndex (see page 16146)

Action (see page 2725)

GroupIndex (see page 2360)

Checked (see page 308)

IsGroupIndexLinked (see page 270)

3.1.2.2.16 TActionLink.SetHelpContext Method

Introduces an interface for setting the help context of a client.

Pascal

```
procedure SetHelpContext(Value: THelpContext); virtual;
```

C++

```
virtual __fastcall SetHelpContext(THelpContext Value);
```

Description

The associated action calls SetHelpContext when its HelpContext property changes so that the action link can propagate the new HelpContext value to the client object.

Value specifies the new value of the HelpContext property.

As implemented in TActionLink (see page 265), SetHelpContext does nothing. Descendant classes override SetHelpContext to set the client property that corresponds to the action's HelpContext property if the IsHelpLinked (see page 271) method returns true.

See Also

HelpContext (see page 311)

Action (see page 2725)

IsHelpLinked (see page 271)

3.1.2.2.17 TActionLink.SetHelpKeyword Method

Introduces an interface for setting the help keyword of a client.

Pascal

```
procedure SetHelpKeyword(const Value: string); virtual;
```

C++

```
virtual __fastcall SetHelpKeyword(const AnsiString Value);
```

Description

The associated action calls SetHelpKeyword when its HelpKeyword property changes so that the action link can propagate the new HelpKeyword value to the client object.

Value specifies the new value of the HelpKeyword property.

As implemented in TActionLink (see page 265), SetHelpKeyword does nothing. Descendant classes override SetHelpKeyword to set the client property that corresponds to the action's HelpKeyword property if the IsHelpLinked (see page 271) method returns true.

See Also

HelpKeyword (see page 311)

HelpKeyword (see page 5826)

HelpKeyword

Action (see page 2725)

IsHelpLinked (see page 271)

3.1.2.2.18 TActionLink.SetHelpType Method

Introduces an interface for setting the HelpType property of a client.

Pascal

```
procedure SetHelpType(Value: THelpType); virtual;
```

C++

```
virtual __fastcall SetHelpType(THelpType Value);
```

Description

The associated action calls SetHelpType when its HelpType property changes so that the action link can propagate the new HelpType value to the client object.

Value specifies the new value of the HelpType property.

As implemented in TActionLink (see page 265), SetHelpType does nothing. Descendant classes override SetHelpType to set the client property that corresponds to the action's HelpType property if the IsHelpLinked (see page 271) method returns true.

See Also

HelpType (see page 312)

HelpType (see page 5827)

HelpType

Action (see page 2725)

IsHelpLinked (see page 271)

3.1.2.2.19 TActionLink.SetHint Method

Introduces an interface for setting the hint for a client.

Pascal

```
procedure SetHint(const Value: string); virtual;
```

C++

```
virtual __fastcall SetHint(const AnsiString Value);
```

Description

The associated action calls SetHint when its Hint property changes so that the action link can propagate the new Hint value to the client object.

Value specifies the new value of the Hint property.

As implemented in TActionLink (see page 265), SetHint does nothing. Descendant classes override SetHint to set the client property that corresponds to the action's Hint property if the IsHintLinked (see page 271) method returns true.

See Also

Hint (see page 312)

Hint (see page 5827)

Hint (see page 16148)

Action (see page 2725)

IsHintLinked (see page 271)

3.1.2.2.20 TActionLink.SetImageIndex Method

Introduces an interface for setting the image index for a client.

Pascal

```
procedure SetImageIndex(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetImageIndex(int Value);
```

Description

The associated action calls SetImageIndex when its ImageIndex property changes so that the action link can propagate the new ImageIndex value to the client object.

Value specifies the new value of the ImageIndex property.

As implemented in TActionLink (see page 265), SetImageIndex does nothing. Descendant classes override SetImageIndex to set the client property that corresponds to the action's ImageIndex property if the IsImageIndexLinked (see page 272) method returns true.

See Also

ImageIndex (see page 313)

ImageIndex (see page 16148)

Action (see page 2725)

IsImageIndexLinked (see page 272)

3.1.2.2.21 TActionLink.SetShortCut Method

Introduces an interface for setting the shortcut for a client.

Pascal

```
procedure SetShortCut(Value: TShortCut); virtual;
```

C++

```
virtual __fastcall SetShortCut(TShortCut Value);
```

Description

The associated action calls SetShortCut when its ShortCut property changes so that the action link can propagate the new ShortCut value to the client object.

Value specifies the new value of the ShortCut property.

As implemented in TActionLink (see page 265), SetShortCut does nothing. Descendant classes override SetShortCut to set the client property that corresponds to the action's ShortCut property if the IsShortCutLinked (see page 272) method returns true.

See Also

ShortCut (see page 314)

ShortCut (see page 16152)

Action (see page 2725)

IsShortCutLinked (see page 272)

3.1.2.2.22 TActionLink.SetVisible Method

Introduces an interface for setting the visible state of a client.

Pascal

```
procedure SetVisible(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetVisible(Boolean Value);
```

Description

The associated action calls SetVisible when its Visible property changes so that the action link can propagate the new Visible value to the client object.

Value specifies the new value of the Visible property.

As implemented in TActionLink (see page 265), SetVisible does nothing. Descendant classes override SetVisible to set the client property that corresponds to the action's Visible property if the IsVisibleLinked (see page 273) method returns true.

See Also

Visible (see page 314)

Visible (see page 5847)

Action (see page 2725)

IsVisibleLinked (see page 273)

3.1.3 TActionList Class

TActionList maintains a list of actions that can be used by components and controls, such as menu items and buttons.

Class Hierarchy

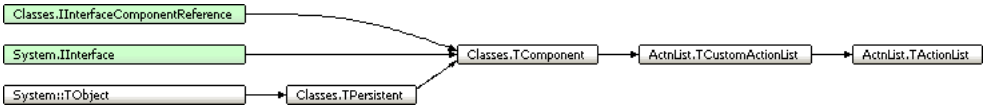

Pascal

```
TActionList = class(TCustomActionList);
```

C++

```
class TActionList : public TCustomActionList;
```

File

ActnList

Description

Use Action lists to centralize the response to user commands (actions). Action list components maintain a list of actions that are available to the client controls in an application. Add action list components to your form or data module from the standard page of the component palette. Double-click the action list to display the action list editor, from which you can add, delete, and rearrange actions.

Note: TActionList is intended primarily for managing the actions in cross-platform applications. While it is always possible to maintain the actions for an application using TActionList, it can be simpler to use an action manager instead.

See Also

TAction (see page 249)

TActionLink (see page 265)

TCustomActionList (see page 315)

Creating Toolbars and Menus

Demo Programs: Actions

What Happens When an Action Fires

How Actions Find Their Targets

Organizing Actions for Toolbars and Menus

Predefined Action Classes

Registering Actions

Setting Up Action Bands

Setting Up Action Lists

Updating Actions

Using Action Lists

What Is an Action?

Writing Action Components

3.1.3.1 TActionList Members

The following tables list the members exposed by TActionList.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAction (see page 320)	Adds an action to the list.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 320)	Generates an OnChange (see page 326) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
V	GetChildren (see page 321)	Calls a specified method for each child action of the action list.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 322)	Responds to notifications when objects are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RemoveAction (see page 323)	Removes an action from the list.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetChildOrder (see page 323)	Sets the order that actions are passed to a callback by the GetChildren (see page 321) method.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetImages (see page 323)	Sets the value of the Images (see page 325) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.

	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetState (see page 324)	This is SetState, a member of class TCustomActionList.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 319)	Creates and initializes a TCustomActionList (see page 315) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 319)	Disposes of the instance of an action list object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 321)	Generates an OnExecute (see page 327) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 321)	Returns a TComponent enumerator.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsShortCut (see page 322)	Evaluates an input key combination to determine whether it is a shortcut for one of the actions in the list.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	UpdateAction (see page 324)	Generates an OnUpdate (see page 327) event.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionCount (see page 324)	Indicates the number of actions in the action list.
	Actions (see page 325)	Lists the actions maintained by the action list.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 285)	Occurs when a change occurs in the action list.
	OnExecute (see page 285)	Occurs when a client object triggers an action in the list.
	OnStateChange (see page 286)	Occurs after an OnChange event.
	OnUpdate (see page 286)	Occurs when the application is idle so that the action list can update a specific action in the list.

Published Properties

	Name	Description
	Images (see page 284)	Lists the images available for actions in the action list.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	State (see page 284)	Indicates whether the actions in the action list respond when the user triggers a client object.
	Tag (see page 2808)	Stores an integer value as part of a component.

3.1.3.2 TActionList Properties

The properties of the TActionList class are listed here.

Published Properties

	Name	Description
	Images (see page 284)	Lists the images available for actions in the action list.
	State (see page 284)	Indicates whether the actions in the action list respond when the user triggers a client object.

3.1.3.2.1 TActionList.Images Property

Lists the images available for actions in the action list.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Images is a list of bitmaps that can be displayed to the left of any action in the action list. An action is associated with an image from this image list by its ImageIndex property. This image can then appear on client controls such as menu items, tool buttons, and so on.

See Also

Actions (see page 325)

ImageIndex (see page 313)

3.1.3.2.2 TActionList.State Property

Indicates whether the actions in the action list respond when the user triggers a client object.

Pascal

```
property State: TActionListState;
```

C++

```
__property TActionListState State;
```

Description

Use State to turn an action list on or off. State can have any of the following values:

Value	Meaning
asNormal	Actions perform their normal response when triggered by client objects or hotkeys.

asSuspended	The actions in the action list do not respond when client objects tell them to "fire". The Enabled property for all actions in the list is unchanged.
asSuspendedEnabled	The actions in the action list do not respond when client objects tell them to "fire". The Enabled property for all actions in the list is set to true. Used for designing ActionBars.

See Also

Execute (see page 304)

Enabled (see page 310)

3.1.3.3 TActionList Events

The events of the TActionList class are listed here.

Published Events

	Name	Description
	OnChange (see page 285)	Occurs when a change occurs in the action list.
	OnExecute (see page 285)	Occurs when a client object triggers an action in the list.
	OnStateChange (see page 286)	Occurs after an OnChange event.
	OnUpdate (see page 286)	Occurs when the application is idle so that the action list can update a specific action in the list.

3.1.3.3.1 TActionList.OnChange Event

Occurs when a change occurs in the action list.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to update your application in response to changes in the action list. Changes to the action list that trigger an OnChange event include changes to an action category or to the action list's image list.

Note: The action list automatically informs the actions in the list when an OnChange event occurs. You need only provide an OnChange event handler for application-specific responses.

3.1.3.3.2 TActionList.OnExecute Event

Occurs when a client object triggers an action in the list.

Pascal

```
property OnExecute: TActionEvent;
```

C++

```
__property TActionEvent OnExecute;
```

Description

Write an OnExecute event handler to respond when an action in the list fires.

When the user invokes a client object, the action list's OnExecute event is the first event to occur in response. If the OnExecute

event handler sets its Handled parameter to true, the action is considered handled, and processing stops there. This blocks the execution of any other event handlers or any built-in response by a predefined action class in response to the user action.

If the OnExecute event handler does not set its Handled parameter to true, the application's OnActionExecute event occurs next. If the OnActionExecute event does not respond to the user input, the action object's OnExecute event occurs. Finally, if the action does not have an OnExecute event handler, the application locates a target object and executes any predefined response implemented by the action object.

See Also

Execute (see page 304)

OnExecute (see page 2716)

OnActionExecute (see page 13725)

3.1.3.3.3 TActionList.OnStateChange Event

Occurs after an OnChange event.

Pascal

```
property OnStateChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnStateChange;
```

Description

Write an OnStateChange event handler to respond after the OnChange event occurs.

See Also

OnChange (see page 326)

3.1.3.3.4 TActionList.OnUpdate Event

Occurs when the application is idle so that the action list can update a specific action in the list.

Pascal

```
property OnUpdate: TActionEvent;
```

C++

```
__property TActionEvent OnUpdate;
```

Description

Write an OnUpdate event handler to configure the properties of a specified action in the list so that it reflects current conditions.

When the application is idle, it cycles through every action in turn giving it a chance to update itself to reflect current conditions. For each action, the first event to occur in this sequence is the action list's OnUpdate event. If the OnUpdate event handler sets its Handled parameter to true, the action is considered updated, and processing stops there. This blocks the execution of any other event handlers or any built-in response by a predefined action class.

If the OnUpdate event handler does not set its Handled parameter to true, the application's OnActionUpdate event occurs next. If the OnActionUpdate event does not update the action, the action object's OnUpdate event occurs. Finally, if the action does not have an OnUpdate event handler, the application executes any predefined update method implemented by the action object.

See Also

OnUpdate (see page 2718)

OnActionUpdate (see page 13726)

3.1.4 TActionListEnumerator Class

Class Hierarchy

ActnList.TActionListEnumerator

Pascal

```
TActionListEnumerator = class;
```

C++

```
class TActionListEnumerator;
```

File

ActnList

Description

This is class ActnList.TActionListEnumerator.

3.1.4.1 TActionListEnumerator Members

The following tables list the members exposed by TActionListEnumerator.

Public Methods

	Name	Description
	Create (see page 287)	This is Create, a member of class TActionListEnumerator.
	GetCurrent (see page 288)	This is GetCurrent, a member of class TActionListEnumerator.
	MoveNext (see page 288)	This is MoveNext, a member of class TActionListEnumerator.

Public Properties

	Name	Description
R	Current (see page 288)	This is Current, a member of class TActionListEnumerator.

3.1.4.2 TActionListEnumerator.Create Constructor

Pascal

```
constructor Create(AActionList: TCustomActionList);
```

C++

```
__fastcall TActionListEnumerator(TCustomActionList AActionList);
```

Description

This is Create, a member of class TActionListEnumerator.

3.1.4.3 TActionListEnumerator Methods

The methods of the TActionListEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 288)	This is GetCurrent, a member of class TActionListEnumerator.
	MoveNext (see page 288)	This is MoveNext, a member of class TActionListEnumerator.

3.1.4.3.1 TActionListEnumerator.GetCurrent Method

Pascal

```
function GetCurrent: TContainedAction;
```

C++

```
__fastcall TContainedAction GetCurrent();
```

Description

This is GetCurrent, a member of class TActionListEnumerator.

3.1.4.3.2 TActionListEnumerator.MoveNext Method

Pascal

```
function MoveNext: Boolean;
```

C++

```
__fastcall Boolean MoveNext();
```

Description

This is MoveNext, a member of class TActionListEnumerator.

3.1.4.4 TActionListEnumerator Properties

The properties of the TActionListEnumerator class are listed here.

Public Properties

	Name	Description
R	Current (see page 288)	This is Current, a member of class TActionListEnumerator.

3.1.4.4.1 TActionListEnumerator.Current Property

Pascal

```
property Current: TContainedAction;
```

C++

```
__property TContainedAction Current;
```

Description

This is Current, a member of class TActionListEnumerator.

3.1.5 TContainedAction Class

TContainedAction is the base class for actions that can be grouped by category and that can appear in action lists or action managers.

Class Hierarchy

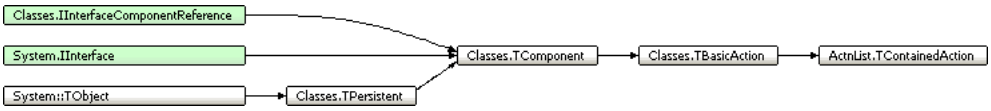

Pascal

```
TContainedAction = class(TBasicAction);
```

C++

```
class TContainedAction : public TBasicAction;
```

File

ActnList

Description

TContainedAction introduces properties and methods that enable actions to appear in an action list or action manager. The properties of TContainedAction contain information about the relationship between the action and its action list or action manager. Methods of TContainedAction override those of the base class to take into account the action list or action manager that owns the action.

Objects of type TContainedAction can be associated with a category of action types. This category is used at design time, to organize the actions in an action list or action manager, and at runtime to organize them in the customize dialog.

See Also

TActionList (see page 280)

TActionManager (see page 432)

3.1.5.1 TContainedAction Members

The following tables list the members exposed by TContainedAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.

	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.

	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 293)	Responds when the client control "fires".
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

3.1.5.2 TContainedAction.Destroy Destructor

Destroys the instance of the contained action.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TContainedAction();
```

Description

If the action is contained in an action list or action manager, Destroy removes the action from that list. Then it calls the inherited Destroy.

3.1.5.3 TContainedAction Methods

The methods of the TContainedAction class are listed here.

Protected Methods

	Name	Description
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.

Public Methods

	Name	Description
	Execute (see page 293)	Responds when the client control "fires".
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	Update (see page 295)	Indicates whether the action updates itself.

3.1.5.3.1 TContainedAction.Execute Method

Responds when the client control "fires".

Pascal

```
function Execute: Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute();
```

Description

Execute is called automatically when a client control "fires" (for example, when the user clicks a button or selects a menu item). It returns true if an event handler is found to handle the action, false otherwise.

Execute first attempts to handle the action by generating an OnExecute event on the action list that contains this action (if the action belongs to an action list). If the action list's OnExecute event handler does not handle the action, Execute generates an OnActionExecute event on the application itself. If neither the action list nor the application handles the action in response to these events, Execute generates an OnExecute event on itself. If this action has no OnExecute event handler, it instructs the application to locate the current target control and call the ExecuteTarget method, which is the mechanism by which predefined action classes perform their function.

See Also

OnExecute (see page 2716)

OnExecute (see page 327)

OnActionExecute (see page 13725)

ExecuteTarget (see page 2712)

3.1.5.3.2 TContainedAction.GetParentComponent Method

Returns the action list or action manager that contains the action.

Pascal

```
function GetParentComponent: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetParentComponent();
```

Description

GetParentComponent is called by the streaming system that loads and saves components. It ensures that objects are loaded and saved with their parent objects. TContainedAction (see page 289) overrides the inherited method to identify the associated action list or action manager as the action's parent.

See Also

ActionList (see page 295)

3.1.5.3.3 TContainedAction.HasParent Method

Indicates whether the action is contained in an action list or action manager.

Pascal

```
function HasParent: Boolean; override;
```

C++

```
virtual __fastcall Boolean HasParent();
```

Description

HasParent is called by the streaming system that loads and saves components. It ensures that objects are loaded and saved with their parent objects. TContainedAction (see page 289) overrides the inherited method to return true if the action is contained in an action list or action manager.

See Also

ActionList (see page 295)

GetParentComponent (see page 293)

3.1.5.3.4 TContainedAction.ReadState Method

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

This is ReadState, a member of class TContainedAction.

3.1.5.3.5 TContainedAction.SetParentComponent Method

Pascal

```
procedure SetParentComponent(AParent: TComponent); override;
```

C++

```
virtual __fastcall SetParentComponent(TComponent * AParent);
```

Description

This is SetParentComponent, a member of class TContainedAction.

3.1.5.3.6 TContainedAction.Update Method

Indicates whether the action updates itself.

Pascal

```
function Update: Boolean; override;
```

C++

```
virtual __fastcall Boolean Update();
```

Description

Update is called automatically when the application is idle to give the action a chance to update itself according to current conditions. It returns true if an event handler is found to update the action, false otherwise.

Update first attempts to update the action by generating an OnUpdate event on the action list that contains this action (if the action belongs to an action list). If the action list's OnUpdate event handler does not update the action, Update generates an OnActionUpdate event on the application itself. If neither the action list nor the application updates the action in response to these events, Update generates an OnUpdate event on itself. If this action has no OnUpdate event handler, it instructs the application to locate the current target control and call the UpdateTarget method, which is the mechanism by which predefined action classes update themselves.

See Also

OnUpdate (see page 2718)

OnActionUpdate (see page 13726)

OnUpdate (see page 327)

UpdateTarget (see page 2715)

3.1.5.4 TContainedAction Properties

The properties of the TContainedAction class are listed here.

Public Properties

	Name	Description
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.

Published Properties

	Name	Description
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.

3.1.5.4.1 TContainedAction.ActionList Property

Indicates the action list or action manager that contains the action.

Pascal

```
property ActionList: TCustomActionList;
```

C++

```
__property TCustomActionList ActionList;
```

Description

ActionList indicates which action list or action manager contains the action. Only contained actions (instances of TContainedAction (see page 289) or its descendants) can appear in an action list or action manager. An action list provides a visual interface for working with actions, which are then available to be associated with client controls in the application. An action manager uses the actions as an abstract representation of user interface elements, which it generates dynamically at runtime.

When creating an action dynamically at runtime, set the ActionList property to add the action to an action list or action manager.

See Also

Category (see page 296)

TActionList (see page 280)

TActionManager (see page 432)

3.1.5.4.2 TContainedAction.Category Property

Specifies the category under which the action appears in its action list or action manager.

Pascal

```
property Category: string;
```

C++

```
__property AnsiString Category;
```

Description

Category is the group, or category, to which the action belongs. An action manager or action list displays its actions grouped by category. In addition, action bars associated with an action manager can generate user interface elements that correspond to action categories.

Action categories group actions by similarities, usually of behavior or functionality. At design-time you can modify or set the Category for an action by selecting the action from the action list or object tree view, and choosing Category in the Object Inspector.

See Also

ActionList (see page 295)

TActionMainMenuBar (see page 585)

TActionToolBar (see page 38)

TActionManager (see page 432)

3.1.5.4.3 TContainedAction.Index Property

Specifies the index of the action in its action list or action manager.

Pascal

```
property Index: Integer;
```

C++

```
__property int Index;
```

Description

Index indicates the position of the action in its action list or action manager. You can change an action's Index at design time by

dragging it to a new position in the action list editor or customize dialog.

The index of actions in an action manager determines the order in which they appear in user interface elements such as menus and tool bars.

See Also

ActionList (see page 295)

Actions (see page 325)

3.1.6 TCustomAction Class

TCustomAction is the base class for actions meant to be used with menu items and controls.

Class Hierarchy

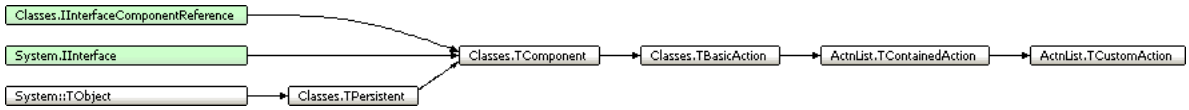

Pascal

```
TCustomAction = class(TContainedAction);
```

C++

```
class TCustomAction : public TContainedAction;
```

File

ActnList

Description

TCustomAction introduces support for the properties and methods of menu items and controls that are linked to action objects. Use TCustomAction as a base class when deriving your own actions that publish specific properties of associated controls.

Action objects centralize the response to user commands (actions) and represent user interface elements in applications that use action bars. They provide an easy way to synchronize, for example, the enabled state and caption of a speed button and a menu item, and handle the response when users click on these components. Each such component, called the client, has its properties dynamically updated by the action and forwards user actions to the action for a response.

You can work with actions at design-time in the action list editor of an action list or the customize dialog of an action manager. The action list or action manager is a container for actions, which it organizes into categories.

Component and control properties and events that are supported in TCustomAction, either directly or through an ancestor, are:

Caption (see page 308)

Checked (see page 308)

Enabled (see page 310)

HelpContext (see page 311)

Hint (see page 312)

ImageIndex (see page 313)

ShortCut (see page 314)

Visible (see page 314)

OnHint (see page 315)

OnUpdate

OnExecute

See Also

TAction (see page 249)

TActionLink (see page 265)

3.1.6.1 TCustomAction Members

The following tables list the members exposed by TCustomAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.

	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.

	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 304)	Responds when a client control "fires".
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
	Update (see page 295)	Indicates whether the action updates itself.
	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Properties

	Name	Description
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

3.1.6.2 TCustomAction.Create Constructor

Instantiates and initializes a TCustomAction (see page 297) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAction(TComponent * AOwner);
```

Description

Applications do not need to call the constructor directly. Actions are created automatically when you add them to an action manager or action list component at design time. If you want to create an action at runtime, assign a TActionList (see page 280) or TActionManager component to its ActionList (see page 295) property after calling Create.

AOwner is the component that is responsible for freeing the action. It becomes the value of the Owner property.

See Also

ActionList (see page 295)

Destroy (see page 302)

Owner (see page 2807)

3.1.6.3 TCustomAction.Destroy Destructor

Frees an instance of TCustomAction (see page 297).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAction();
```

Description

Do not call the destructor directly in an application. Instead, call Free. Free ensures that the action is not nil before calling the destructor.

If the action is contained in an action list or action manager, Destroy removes the action from that list.

See Also

Create (see page 301)

Actions (see page 325)

3.1.6.4 TCustomAction Fields

The fields of the TCustomAction class are listed here.

Protected Fields

	Name	Description
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

3.1.6.4.1 TCustomAction.FImage Field

Pascal

```
FImage: TObject;
```

C++

```
TObject * FImage;
```

Description

This is FImage, a member of class TCustomAction.

3.1.6.4.2 TCustomAction.FMask Field

Pascal

```
FMask: TObject;
```

C++

```
TObject * FMask;
```

Description

This is FMask, a member of class TCustomAction.

3.1.6.5 TCustomAction Methods

The methods of the TCustomAction class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 303)	Copies the properties of this action to a destination object.
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetName (see page 307)	Sets the value of the Name property.

Public Methods

	Name	Description
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 304)	Responds when a client control "fires".

3.1.6.5.1 TCustomAction.AssignTo Method

Copies the properties of this action to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

AssignTo is overridden so that existing objects of type TCustomAction (see page 297) can be copied to other action objects of the same type. AssignTo ensures the correct assignment of property values.

See Also

Assign (see page 2930)

3.1.6.5.2 TCustomAction.DoHint Method

Generates an OnHint (see page 315) event.

Pascal

```
function DoHint(var HintStr: string): Boolean; dynamic;
```

C++

```
__fastcall Boolean DoHint(AnsiString HintStr);
```

Description

Typically, Applications do not call the DoHint method. It is called automatically when the user pauses the mouse over one of the action's client controls or menu items.

DoHint checks for an OnHint (see page 315) event handler and passes HintStr to it, if the handler exists. DoHint returns true if no event handler is assigned or if the OnHint (see page 315) event handler indicates that the hint should be displayed using the default popup window.

See Also

OnHint (see page 315)

3.1.6.5.3 TCustomAction.Execute Method

Responds when a client control "fires".

Pascal

```
function Execute: Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute();
```

Description

Execute is called automatically when a client control "fires" (for example, when the user clicks a button or selects a menu item). It returns true if an event handler is found to handle the action, false if there was no event handler or if the action was not enabled.

Execute first ensures that the action is updated. Then, if the Enabled (see page 310) property is true, it attempts to handle the action by generating an OnExecute event on the action list that contains this action (if the action belongs to an action list). If the action list's OnExecute event handler does not handle the action, Execute generates an OnActionExecute event on the application itself. If neither the action list nor the application handles the action in response to these events, Execute generates an OnExecute event on itself. If this action has no OnExecute event handler, Execute instructs the application to locate the current target control and call the ExecuteTarget method, which is the mechanism by which predefined action classes perform their function.

C++ Examples:

```
/*
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString NewFileName, OldFileName;
  TFileStream *NewFile, *OldFile;
  SavePictureDialog1->DefaultExt = GraphicExtension(__classid(Graphics::TBitmap));
  SavePictureDialog1->Filter = GraphicFilter(__classid(Graphics::TBitmap));
  OldFileName = "factory.bmp";
  if (SavePictureDialog1->Execute())
  {
    NewFileName = SavePictureDialog1->FileName;
```

```
    OldFile = new TFileStream(OldFileName, (fmOpenRead | fmShareDenyWrite));
    try
    {
      NewFile = new TFileStream(NewFileName, (fmCreate | fmShareDenyRead));
      try
      {
        NewFile->CopyFrom(OldFile, OldFile->Size);
      }
      __finally
      {
        FreeAndNil(&NewFile);
      };
    }
    __finally
    {
      FreeAndNil(&OldFile);
    };
  };
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Image1->Picture->LoadFromFile("../factory.bmp");
}
```

Delphi Examples:

```
{
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
}
procedure TForm1.Button1Click(Sender: TObject);
var NewFileName, OldFileName: string;
    NewFile, OldFile: TFileStream;
begin
  SavePictureDialog1.DefaultExt := GraphicExtension(TBitmap);
  SavePictureDialog1.Filter := GraphicFilter(TBitmap);
  OldFileName := 'factory.bmp';
  if SavePictureDialog1.Execute then
  begin
    NewFileName := SavePictureDialog1.FileName;
    OldFile := TFileStream.Create(
      OldFileName, fmOpenRead or fmShareDenyWrite);
    try
      NewFile := TFileStream.Create(
        NewFileName, fmCreate or fmShareDenyRead);
      try
        NewFile.CopyFrom(OldFile, OldFile.Size);
      finally
        FreeAndNil(NewFile);
      end;
    finally
      FreeAndNil(OldFile);
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Picture.LoadFromFile('factory.bmp');
end;
```

See Also

Enabled (see page 310)

OnExecute (see page 2716)

OnExecute (see page 327)

OnActionExecute (see page 13725)

ExecuteTarget (see page 2712)

3.1.6.5.4 TCustomAction.HandleShortCut Method

Executes the action when the user types on of its short cuts.

Pascal

```
function HandleShortCut: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean HandleShortCut();
```

Description

Applications can't call the protected HandleShortCut method. The action list's IsShortCut method calls HandleShortCut when it finds a match between a shortcut key combination and the action's ShortCut (see page 314) or SecondaryShortCuts (see page 314) property.

As implemented in TCustomAction (see page 297), HandleShortCut calls Execute (see page 304) to cause the action to perform its function. HandleShortCut returns the value that Execute (see page 304) returns.

Descendant classes can override HandleShortCut to respond in other ways when the user types the action's shortcut keys.

See Also

IsShortCut (see page 322)

ShortCut (see page 314)

SecondaryShortCuts (see page 314)

Execute (see page 304)

3.1.6.5.5 TCustomAction.SetHelpContext Method

Pascal

```
procedure SetHelpContext(Value: THelpContext); virtual;
```

C++

```
virtual __fastcall SetHelpContext(THelpContext Value);
```

Description

This is SetHelpContext, a member of class TCustomAction.

3.1.6.5.6 TCustomAction.SetHelpKeyword Method

Pascal

```
procedure SetHelpKeyword(const Value: string); virtual;
```

C++

```
virtual __fastcall SetHelpKeyword(const AnsiString Value);
```

Description

This is SetHelpKeyword, a member of class TCustomAction.

3.1.6.5.7 TCustomAction.SetName Method

Sets the value of the Name property.

Pascal

```
procedure SetName(const Value: TComponentName); override;
```

C++

```
virtual __fastcall SetName(const TComponentName Value);
```

Description

SetName is the protected write implementation of the Name property. It calls the ChangeName method, which performs the actual name change. TCustomAction (see page 297) overrides SetName so that it also changes the Caption (see page 308) property if Caption (see page 308) is the same as Name.

3.1.6.6 TCustomAction Properties

The properties of the TCustomAction class are listed here.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Properties

	Name	Description
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

3.1.6.6.1 TCustomAction.AutoCheck Property

Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.

Pascal

```
property AutoCheck: Boolean;
```

C++

```
__property Boolean AutoCheck;
```

Description

AutoCheck causes the checked property to toggle when the action executes. This allows the action's Checked (see page 308) property to remain in sync with the client's Checked (see page 308) property (or equivalent). If the client is a menu item (where the Checked (see page 308) property does not necessarily toggle), AutoCheck is propagated to the AutoCheck property of the menu item.

See Also

Checked (see page 308)

Execute (see page 304)

AutoCheck (see page 16134)

3.1.6.6.2 TCustomAction.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

3.1.6.6.3 TCustomAction.Checked Property

Indicates whether client controls and menu items appear checked.

Pascal

```
property Checked: Boolean;
```

C++

```
__property Boolean Checked;
```

Description

Checked specifies the checked state for the action. The value of Checked is propagated to client controls and menu items.

Note: If the action has a GroupIndex (see page 311) value greater than 0, then setting Checked to true causes all other actions in the group to have their Checked property set to false.

Note: Use the AutoCheck (see page 308) property to ensure that the action's Checked property stays synchronized with client's Checked properties.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
```

```
end;
```

See Also

GroupIndex (see page 311)

AutoCheck (see page 308)

3.1.6.6.4 TCustomAction.DisableIfNoHandler Property

Indicates whether client components should be disabled if no handler is found.

Pascal

```
property DisableIfNoHandler: Boolean;
```

C++

```
__property Boolean DisableIfNoHandler;
```

Description

DisableIfNoHandler specifies whether the Enabled (see page 310) property of client components should be set to false if no handler can be found for the action.

When the user triggers one of the action's clients, the application makes a series of attempts to determine how the action should respond. This includes checking a number of event handlers, and, if it fails to find an event handler, locating a target control and checking for a predefined response that applies to that target control. When DisableIfNoHandler is true, if the application completes this sequence without finding any way for the action to respond to the user input, The action's Enabled (see page 310) property is set to false.

In addition, for some predefined actions, DisableIfNoHandler causes the action to disable itself if an event handler is required for the action to perform a meaningful task. For example, if you do not supply an OnAccept event handler to TFileOpen, then if DisableIfNoHandler is true, the action disables itself because there is no way for the application to respond when the user selects a file name in the dialog.

See Also

Enabled (see page 310)

HandlesTarget (see page 2713)

3.1.6.6.5 TCustomAction.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

3.1.6.6.6 TCustomAction.GroupIndex Property

Indicates whether the action acts like a radio button with other actions in the same action list.

Pascal

```
property GroupIndex: Integer;
```

C++

```
__property int GroupIndex;
```

Description

GroupIndex is used to define groups of actions that act like radio buttons. When GroupIndex is greater than 0, it identifies the group to which the action belongs. When the Checked (see page 308) property of any action in that group is set to true, the Checked (see page 308) property of all other actions in the group is set to false. That is, only one action in the group can be checked at a time.

Note: All actions in a group must be listed by the same action list or action manager.

See Also

Checked (see page 308)

GroupIndex (see page 2360)

3.1.6.6.7 TCustomAction.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType (see page 312) property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

3.1.6.6.8 TCustomAction.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType (see page 312) property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

3.1.6.6.9 TCustomAction.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext (see page 311) or the HelpKeyword (see page 311) property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

3.1.6.6.10 TCustomAction.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

3.1.6.6.11 TCustomAction.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

3.1.6.6.12 TCustomAction.SavedEnabledState Property

Saves the value of the Enabled (see page 310) property when the action is suspended.

Pascal

```
property SavedEnabledState: Boolean;
```

C++

```
__property Boolean SavedEnabledState;
```

Description

TCustomAction (see page 297) uses SavedEnabledState internally to save the value of the Enabled (see page 310) property when its action list has a state of asSuspendedEnabled.

See Also

Enabled (see page 310)

State (see page 326)

3.1.6.6.13 TCustomAction.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut (see page 314) property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut (see page 314) property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute (see page 304) method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

3.1.6.6.14 TCustomAction.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

3.1.6.6.15 TCustomAction.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

3.1.6.7 TCustomAction Events

The events of the TCustomAction class are listed here.

Public Events

	Name	Description
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.

3.1.6.7.1 TCustomAction.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

3.1.7 TCustomActionList Class

TCustomActionList is the base class for action list objects that manage a set of actions.

Class Hierarchy

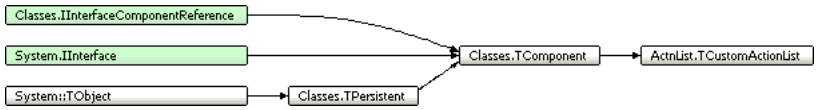

Pascal

```
TCustomActionList = class(TComponent);
```

C++

```
class TCustomActionList : public TComponent;
```

File

ActnList

Description

Action lists manage a collection of action objects so that they are available to the components in an application. These action objects centralize the response of the application to user input, and update the properties of all controls that trigger that response.

TCustomActionList descendants provide the user interface for working with actions at design time. These include TActionList (see page 280), which lets you set up actions using the action list editor, and TActionManager, which lets you configure actions using the customize dialog.

Action lists maintain a list of actions that can be grouped by category. Actions (see page 325) in an action list are TContainedAction (see page 289) or descendant types.

See Also

TActionList (see page 280)

TContainedAction (see page 289)

TActionManager (see page 432)

3.1.7.1 TCustomActionList Members

The following tables list the members exposed by TCustomActionList.

Protected Events

	Name	Description
	OnChange (see page 326)	Occurs when a change occurs in the action list.
	OnExecute (see page 327)	Occurs when a client object triggers an action in the list.
	OnUpdate (see page 327)	Occurs when the application is idle so that the action list can update a specific action in the list.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAction (see page 320)	Adds an action to the list.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 320)	Generates an OnChange (see page 326) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
V	GetChildren (see page 321)	Calls a specified method for each child action of the action list.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 322)	Responds to notifications when objects are created or destroyed.

	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RemoveAction (see page 323)	Removes an action from the list.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 323)	Sets the order that actions are passed to a callback by the GetChildren (see page 321) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetImages (see page 323)	Sets the value of the Images (see page 325) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetState (see page 324)	This is SetState, a member of class TCustomActionList.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnStateChange (see page 327)	Occurs after an OnChange (see page 326) event.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 319)	Creates and initializes a TCustomActionList (see page 315) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 319)	Disposes of the instance of an action list object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 321)	Generates an OnExecute (see page 327) event.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 321)	Returns a TComponent enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsShortCut (see page 322)	Evaluates an input key combination to determine whether it is a shortcut for one of the actions in the list.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	UpdateAction (see page 324)	Generates an OnUpdate (see page 327) event.

Public Properties

	Name	Description
R	ActionCount (see page 324)	Indicates the number of actions in the action list.
	Actions (see page 325)	Lists the actions maintained by the action list.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Images (see page 325)	Lists the images available for actions in the action list.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.

	State (see page 326)	Indicates whether the actions in the action list respond when the user triggers a client object.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

3.1.7.2 TCustomActionList.Create Constructor

Creates and initializes a TCustomActionList (see page 315) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomActionList(TComponent * AOwner);
```

Description

Call Create to instantiate an action list at runtime. Action lists added to forms at design time are created automatically.

The AOwner parameter is the owner of the action list, typically this is a form or data module.

See Also

Destroy (see page 319)

3.1.7.3 TCustomActionList.Destroy Destructor

Disposes of the instance of an action list object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomActionList();
```

Description

You normally do not need to call the destructor for an action list if it has an owner. The action list is destroyed automatically by its owner, which was passed to its constructor when it was created. In situations where the custom action list object needs to be manually destroyed, do so by calling its Free method.

See Also

Create (see page 319)

3.1.7.4 TCustomActionList Methods

The methods of the TCustomActionList class are listed here.

Protected Methods

	Name	Description
	AddAction (see page 320)	Adds an action to the list.

	Change (see page 320)	Generates an OnChange (see page 326) event.
	GetChildren (see page 321)	Calls a specified method for each child action of the action list.
	Notification (see page 322)	Responds to notifications when objects are created or destroyed.
	RemoveAction (see page 323)	Removes an action from the list.
	SetChildOrder (see page 323)	Sets the order that actions are passed to a callback by the GetChildren (see page 321) method.
	SetImages (see page 323)	Sets the value of the Images (see page 325) property.
	SetState (see page 324)	This is SetState, a member of class TCustomActionList.

Public Methods

	Name	Description
	ExecuteAction (see page 321)	Generates an OnExecute (see page 327) event.
	GetEnumerator (see page 321)	Returns a TComponent enumerator.
	IsShortCut (see page 322)	Evaluates an input key combination to determine whether it is a shortcut for one of the actions in the list.
	UpdateAction (see page 324)	Generates an OnUpdate (see page 327) event.

3.1.7.4.1 TCustomActionList.AddAction Method

Adds an action to the list.

Pascal

```
procedure AddAction(Action: TContainedAction);
```

C++

```
__fastcall AddAction(TContainedAction Action);
```

Description

Applications can't call this protected method. AddAction is used internally to add an action to the Actions (see page 325) property array.

To add an action to an action list, set the action's ActionList property.

See Also

Actions (see page 325)

RemoveAction (see page 323)

ActionList (see page 295)

3.1.7.4.2 TCustomActionList.Change Method

Generates an OnChange (see page 326) event.

Pascal

```
procedure Change; virtual;
```

C++

```
virtual __fastcall Change();
```

Description

Change is called when a change, such as a modification to the categories of the actions in the list, occurs in the action list.

Change:

1. Invokes an OnChange (see page 326) handler, if one has been assigned.
2. Informs all actions in the list that something has changed so that they can update themselves.

See Also

OnChange (see page 326)

3.1.7.4.3 TCustomActionList.ExecuteAction Method

Generates an OnExecute (see page 327) event.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

Typically, applications do not call the ExecuteAction method. Instead, the action indicated by the Action parameter calls ExecuteAction in response to a user command. It is possible, however, to call ExecuteAction to generate an OnExecute (see page 327) event even though no client control has been triggered.

ExecuteAction invokes the OnExecute (see page 327) event handler for the action list. It returns true if the event handler handles the action, false otherwise.

See Also

OnExecute (see page 327)

3.1.7.4.4 TCustomActionList.GetChildren Method

Calls a specified method for each child action of the action list.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is used by the component streaming system. For each action in the action list, GetChildren calls the method passed in Proc. The actions are passed to the callback in the order that they are loaded or saved by the component streaming system.

3.1.7.4.5 TCustomActionList.GetEnumerator Method

Returns a TComponent enumerator.

Pascal

```
function GetEnumerator: TActionListEnumerator;
```

C++

```
__fastcall TActionListEnumerator GetEnumerator();
```

Description

GetEnumerator returns a TComponentEnumerator reference, which enumerates the components contained within a specified containing component.

To process all these sub components, call the TComponentEnumerator GetCurrent method within a While MoveNext do loop.

See Also

TComponentEnumerator (see page 2812)

3.1.7.4.6 TCustomActionList.IsShortCut Method

Evaluates an input key combination to determine whether it is a shortcut for one of the actions in the list.

Pascal

```
function IsShortCut(var Message: TWMKey): Boolean;
```

C++

```
__fastcall Boolean IsShortCut(TWMKey Message);
```

Description

Typically, applications do not need to call IsShortCut. When the form's IsShortCut method does not evaluate a shortcut key by some other means (such as an event handler), it calls IsShortCut for every associated action list until it finds one that can handle the shortcut.

The action list's IsShortCut method, in turn, checks the specified keystroke against the ShortCut property and SecondaryShortCuts properties of each action in the list. If it finds a match, it tells the action to handle the shortcut, and then returns true. If IsShortCut can't find an action to handle the shortcut, it returns false.

Message encapsulates the Windows keystroke message that is to be checked to see if it is a shortcut.

Key is the key code for the potential shortcut key.

Shift specifies the state of the Shift, Alt, and Ctrl keys that may be combined with Key.

KeyText is the string representation of the potential shortcut key.

See Also

ShortCut (see page 314)

SecondaryShortCuts (see page 314)

IsShortCut (see page 13880)

3.1.7.4.7 TCustomActionList.Notification Method

Responds to notifications when objects are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification notifies the action list that the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

TCustomActionList (see page 315) overrides Notification to update the Images (see page 325) property when images from the list are destroyed and the Actions (see page 325) property when actions in the list are destroyed.

See Also

Images (see page 325)

3.1.7.4.8 TCustomActionList.RemoveAction Method

Removes an action from the list.

Pascal

```
procedure RemoveAction(Action: TContainedAction);
```

C++

```
__fastcall RemoveAction(TContainedAction Action);
```

Description

Applications can't call this protected method. RemoveAction is used internally to remove an action from the Actions (see page 325) property array. RemoveAction removes the action and sets its ActionList property to nil (Delphi) or NULL (C++).

To remove an action from an action list, change the action's ActionList property.

See Also

Actions (see page 325)

AddAction (see page 320)

ActionList (see page 295)

3.1.7.4.9 TCustomActionList.SetChildOrder Method

Sets the order that actions are passed to a callback by the GetChildren (see page 321) method.

Pascal

```
procedure SetChildOrder(Component: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Component, int Order);
```

Description

SetChildOrder specifies the order in which the component streaming system loads and saves the action components owned by this action list. It ensures that the actions are loaded and saved in the same order that they appear in the Actions (see page 325) property array. You should not need to call this method directly.

See Also

GetChildren (see page 321)

Actions (see page 325)

3.1.7.4.10 TCustomActionList.SetImages Method

Sets the value of the Images (see page 325) property.

Pascal

```
procedure SetImages(Value: TCustomImageList); virtual;
```

C++

```
virtual __fastcall SetImages(TCustomImageList Value);
```

Description

SetImages is the protected write implementation of the Images (see page 325) property. It registers the action list with the

image list specified by Value so that it will be informed of any changes to the image list.

See Also

Images (see page 325)

3.1.7.4.11 TCustomActionList.SetState Method

Pascal

```
procedure SetState(const Value: TActionListState); virtual;
```

C++

```
virtual __fastcall SetState(const TActionListState Value);
```

Description

This is SetState, a member of class TCustomActionList.

3.1.7.4.12 TCustomActionList.UpdateAction Method

Generates an OnUpdate (see page 327) event.

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

Typically, applications do not call the UpdateAction method. Instead, the action indicated by the Action parameter calls UpdateAction when informed by the application that it can update itself. It is possible, however, to call UpdateAction to generate an OnUpdate (see page 327) event even though this has not been initiated by the application.

UpdateAction invokes the OnUpdate (see page 327) event handler for the action list. It returns true if the event handler updates the action, false otherwise.

See Also

OnUpdate (see page 327)

3.1.7.5 TCustomActionList Properties

The properties of the TCustomActionList class are listed here.

Public Properties

	Name	Description
R	ActionCount (see page 324)	Indicates the number of actions in the action list.
	Actions (see page 325)	Lists the actions maintained by the action list.
	Images (see page 325)	Lists the images available for actions in the action list.
	State (see page 326)	Indicates whether the actions in the action list respond when the user triggers a client object.

3.1.7.5.1 TCustomActionList.ActionCount Property

Indicates the number of actions in the action list.

Pascal

```
property ActionCount: Integer;
```

C++

```
__property int ActionCount;
```

Description

ActionCount indicates how many actions are contained in the action list.

See Also

Actions (see page 325)

3.1.7.5.2 TCustomActionList.Actions Property

Lists the actions maintained by the action list.

Pascal

```
property Actions [Index: Integer]: TContainedAction;
```

C++

```
__property TContainedAction Actions[int Index];
```

Description

Actions is an indexed property containing a list of the actions in the action list component.

Index identifies the action by index, where 0 is the index of the first action, 1 is the index of the second action, and so on. The upper limit for Index is given by the ActionCount (see page 324) property.

See Also

ActionCount (see page 324)

operator_sb

3.1.7.5.3 TCustomActionList.Images Property

Lists the images available for actions in the action list.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Images is a list of bitmaps that can be displayed to the left of any action in the action list. An action is associated with an image from this image list by its ImageIndex property. This image can then appear on client controls such as menu items, tool buttons, and so on.

See Also

Actions (see page 325)

ImageIndex (see page 313)

3.1.7.5.4 TCustomActionList.State Property

Indicates whether the actions in the action list respond when the user triggers a client object.

Pascal

```
property State: TActionListState;
```

C++

```
__property TActionListState State;
```

Description

Use State to turn an action list on or off. State can have any of the following values:

Value	Meaning
asNormal	Actions (see page 325) perform their normal response when triggered by client objects or hotkeys.
asSuspended	The actions in the action list do not respond when client objects tell them to "fire". The Enabled property for all actions in the list is unchanged.
asSuspendedEnabled	The actions in the action list do not respond when client objects tell them to "fire". The Enabled property for all actions in the list is set to true. Used for designing ActionBars.

See Also

Execute (see page 304)

Enabled (see page 310)

3.1.7.6 TCustomActionList Events

The events of the TCustomActionList class are listed here.

Protected Events

	Name	Description
	OnChange (see page 326)	Occurs when a change occurs in the action list.
	OnExecute (see page 327)	Occurs when a client object triggers an action in the list.
	OnUpdate (see page 327)	Occurs when the application is idle so that the action list can update a specific action in the list.

Public Events

	Name	Description
	OnStateChange (see page 327)	Occurs after an OnChange (see page 326) event.

3.1.7.6.1 TCustomActionList.OnChange Event

Occurs when a change occurs in the action list.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to update your application in response to changes in the action list. Changes to the action list

that trigger an OnChange event include changes to an action category or to the action list's image list.

Note: The action list automatically informs the actions in the list when an OnChange event occurs. You need only provide an OnChange event handler for application-specific responses.

3.1.7.6.2 TCustomActionList.OnExecute Event

Occurs when a client object triggers an action in the list.

Pascal

```
property OnExecute: TActionEvent;
```

C++

```
__property TActionEvent OnExecute;
```

Description

Write an OnExecute event handler to respond when an action in the list fires.

When the user invokes a client object, the action list's OnExecute event is the first event to occur in response. If the OnExecute event handler sets its Handled parameter to true, the action is considered handled, and processing stops there. This blocks the execution of any other event handlers or any built-in response by a predefined action class in response to the user action.

If the OnExecute event handler does not set its Handled parameter to true, the application's OnActionExecute event occurs next. If the OnActionExecute event does not respond to the user input, the action object's OnExecute event occurs. Finally, if the action does not have an OnExecute event handler, the application locates a target object and executes any predefined response implemented by the action object.

See Also

Execute (see page 304)

OnExecute (see page 2716)

OnActionExecute (see page 13725)

3.1.7.6.3 TCustomActionList.OnStateChange Event

Occurs after an OnChange (see page 326) event.

Pascal

```
property OnStateChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnStateChange;
```

Description

Write an OnStateChange event handler to respond after the OnChange (see page 326) event occurs.

See Also

OnChange (see page 326)

3.1.7.6.4 TCustomActionList.OnUpdate Event

Occurs when the application is idle so that the action list can update a specific action in the list.

Pascal

```
property OnUpdate: TActionEvent;
```

C++

```
__property TActionEvent OnUpdate;
```

Description

Write an OnUpdate event handler to configure the properties of a specified action in the list so that it reflects current conditions.

When the application is idle, it cycles through every action in turn giving it a chance to update itself to reflect current conditions. For each action, the first event to occur in this sequence is the action list's OnUpdate event. If the OnUpdate event handler sets its Handled parameter to true, the action is considered updated, and processing stops there. This blocks the execution of any other event handlers or any built-in response by a predefined action class.

If the OnUpdate event handler does not set its Handled parameter to true, the application's OnActionUpdate event occurs next. If the OnActionUpdate event does not update the action, the action object's OnUpdate event occurs. Finally, if the action does not have an OnUpdate event handler, the application executes any predefined update method implemented by the action object.

See Also

OnUpdate (see page 2718)

OnActionUpdate (see page 13726)

3.1.8 TShortCutList Class

TShortCutList manages an array of keyboard shortcuts.

Class Hierarchy

Pascal

```
TShortCutList = class(TStringList);
```

C++

```
class TShortCutList : public TStringList;
```

File

ActnList

Description

TShortCutList is a specialized list for storing TShortCut values. TShortCutList is the type of the SecondaryShortCuts property.

TShortCutList is a descendant of TStringList, which manages a collection of strings. As such, TShortCutList adds, deletes, and searches for shortcuts using their text representation. (You can convert between a shortcut and its text representation using the TextToShortCut and SHortCutToText functions). TShortCutList stores the shortcut that corresponds to a text representation as the value of the Objects property for that string. Do not change the value of the Objects property, or the short cut list will not operate correctly.

See Also

TextToShortCut

ShortCutToText

TShortCut

TCustomAction (see page 297)

3.1.8.1 TShortCutList Members

The following tables list the members exposed by TShortCutList.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 3013)	This is Changed, a member of class TStringList.
	Changing (see page 3013)	This is Changing, a member of class TStringList.
	CompareStrings (see page 3014)	Compares two strings.
	CompareStrings (see page 3048)	Compares two strings.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 3049)	Reads and writes the Strings (see page 3080) property as if it were published.
	Error (see page 3052)	This is the overview for the Error method overload.
	ExtractName (see page 3053)	Returns the name portion of a string that is a name value pair.
	Get (see page 3017)	Returns a string given its index.
	Get (see page 3053)	Returns a string given its index.
	GetCapacity (see page 3017)	Returns the currently allocated size of the list of strings.
	GetCapacity (see page 3054)	Returns the currently allocated size of the list of strings.
	GetCount (see page 3018)	Returns the number of strings in the list
	GetCount (see page 3054)	Returns the number of strings in the list
	GetObject (see page 3018)	Returns the object associated with the string at a specified index.
	GetObject (see page 3055)	Returns the object associated with the string at a specified index.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTextStr (see page 3056)	Returns the value of the Text (see page 3084) property.
	InsertItem (see page 3020)	This is InsertItem, a member of class TStringList.
	Put (see page 3022)	Changes the value of the string with a specified index.
	Put (see page 3064)	Changes the value of the string with a specified index.
	PutObject (see page 3022)	Changes the object associated with the string at a specified index.
	PutObject (see page 3065)	Changes the object associated with the string at a specified index.
	SetCapacity (see page 3022)	Changes the amount of memory allocated to hold strings in the list.
	SetCapacity (see page 3068)	Changes the amount of memory allocated to hold strings in the list.
	SetTextStr (see page 3069)	Sets the Text (see page 3084) property.
	SetUpdateState (see page 3023)	Performs internal adjustments before or after a series of updates.
	SetUpdateState (see page 3069)	Performs internal adjustments before or after a series of updates.

Protected Properties

	Name	Description
	UpdateCount (see page 3085)	Indicates the number of calls to BeginUpdate (see page 3045) that have not been matched by a call to EndUpdate (see page 3050).

Public Events

	Name	Description
	OnChange (see page 3028)	Occurs immediately after the list of strings changes.
	OnChanging (see page 3028)	Occurs immediately before the list of strings changes.

Public Methods

	Name	Description
	Add (see page 332)	Adds a new shortcut to the list.

V	AddObject (see page 3012)	Adds a string to the list, and associates an object with the string.
V	AddObject (see page 3036)	Adds a string to the list, and associates an object with the string.
V	AddStrings (see page 3039)	Adds a group of strings to the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 3041)	Adds a string to the list.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	Assign (see page 3043)	Sets the strings in the list, and possibly associated objects, from a source object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 3045)	Enables the TStrings (see page 3029) object to track when the list of strings is changing.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 3013)	Deletes all the strings from the list.
A	Clear (see page 3047)	Introduces an abstract (Delphi) or pure virtual (C++) method to empty the list and any associated objects.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	CustomSort (see page 3014)	Sorts the strings in the list in a customized order.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Delete (see page 3015)	Removes the string specified by the Index parameter.
A	Delete (see page 3049)	Introduces an abstract (Delphi) or pure virtual (C++) method to delete a specified string from the list.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 3009)	Destroys an instance of TStringList (see page 3006).
V	Destroy (see page 3032)	Destroys the TPersistent instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 3050)	Enables the TStrings (see page 3029) object to keep track of when the list of strings has finished changing.
	Equals (see page 3051)	Compares the list of strings to the list from another TStrings (see page 3029) object and returns true if the two lists match.
V	Exchange (see page 3015)	Swaps the position of two strings in the list.
V	Exchange (see page 3053)	Swaps the position of two strings in the list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
V	Find (see page 3016)	Locates the index for a string in a sorted list and indicates whether a string with that value already exists in the list.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 3054)	Returns a TStrings (see page 3029) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetText (see page 3055)	Allocates a text buffer and fills it with the value of the Text (see page 3084) property.
V	IndexOf (see page 3019)	Returns the position of a string in the list.
V	IndexOf (see page 3056)	Returns the position of a string in the list.
V	IndexOfName (see page 3057)	Returns the position of the first name-value pair with the specified name.
V	IndexOfObject (see page 3059)	Returns the index of the first string in the list associated with a given object.
	IndexOfShortCut (see page 332)	Returns the position of a shortcut in the list.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Insert (see page 3019)	Inserts a string to the list at the position specified by Index.
A	Insert (see page 3059)	Introduces abstract (Delphi) or pure virtual (C++) method to insert a string at a specified position.
V	InsertObject (see page 3020)	Inserts a string into the list at the specified position, and associates it with an object.
V	InsertObject (see page 3060)	Inserts a string into the list at the specified position, and associates it with an object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	LoadFromFile (see page 3061)	Fills the list with the lines of text in a specified file.
V	LoadFromStream (see page 3062)	Fills the list with lines of text read from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	Move (see page 3063)	Changes the position of a string in the list.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SaveToFile (see page 3065)	Saves the strings in the list to the specified file.
V	SaveToStream (see page 3067)	Writes the value of the Text (see page 3084) property to a stream object.
V	SetText (see page 3068)	Sets the Text (see page 3084) property.
V	Sort (see page 3023)	Sorts the strings in the list in ascending order.

Public Properties

	Name	Description
	Capacity (see page 3070)	Indicates the number of strings the TStrings (see page 3029) object can hold.
	CaseSensitive (see page 3025)	Controls (see page 5711) whether strings are located, sorted, and identified as duplicates in a case-sensitive or case-insensitive manner.
	CommaText (see page 3070)	Lists the strings in the TStrings (see page 3029) object in system data format (SDF).
R	Count (see page 3071)	Introduces an abstract property to represent the number of strings in the list.
	DelimitedText (see page 3074)	Represents all the strings in the TStrings (see page 3029) object as a single delimited string.
	Delimiter (see page 3074)	Specifies the delimiter used by the DelimitedText (see page 3074) property.
	Duplicates (see page 3025)	Specifies whether duplicate strings can be added to sorted lists.
	LineBreak (see page 3075)	Defines the line break character(s).
R	Names (see page 3075)	Indicates the name part of strings that are name-value pairs.
	NameValueSeparator (see page 3077)	Indicates the character used to separate names from values.
	Objects (see page 3077)	Represents a set of objects that are associated one with each of the strings in the Strings (see page 3080) property.
	QuoteChar (see page 3080)	Specifies the quote character used by the DelimitedText (see page 3074) property.
R	ShortCuts (see page 333)	Lists the shortcuts in the list.
	Sorted (see page 3026)	Specifies whether the strings in the list should be automatically sorted.
	StrictDelimiter (see page 3080)	Determines how the Delimiter (see page 3074) property is used.
	Strings (see page 3080)	References the strings in the list by their positions.
	StringsAdapter (see page 3084)	Implements an IStringsAdapter (see page 3208) interface for the TStrings (see page 3029) object.
	Text (see page 3084)	Lists the strings in the TStrings (see page 3029) object as a single string with the individual strings delimited by carriage returns and line feeds.
	ValueFromIndex (see page 3085)	Represents the value part of a string with a given index, on strings that are name-value pairs.
	Values (see page 3086)	Represents the value part of a string associated with a given name, on strings that are name-value pairs.

3.1.8.2 TShortCutList Methods

The methods of the TShortCutList class are listed here.

Public Methods

	Name	Description
	Add (see page 332)	Adds a new shortcut to the list.
	IndexOfShortCut (see page 332)	Returns the position of a shortcut in the list.

3.1.8.2.1 TShortCutList.Add Method

Adds a new shortcut to the list.

Pascal

```
function Add(const S: String): Integer; override;
```

C++

```
virtual __fastcall int Add(const AnsiString S);
```

Description

Call Add to add a shortcut to the list. If the list is sorted, the shortcut is added to the appropriate position in the sort order. If the list is not sorted, the shortcut is added to the end of the list. Add returns the position of the shortcut in the list, where the first item in the list has a value of 0.

S is the string representation of the shortcut to add. You can obtain this from a shortcut using the global ShortCutToText function. TShortCutList (see page 328) saves the string S as a new string in the list and the corresponding shortcut as an Object associated with that string.

Note: For sorted lists, Add raises an EListError exception if the string S already appears in the list and Duplicates is set to dupError. If Duplicates is set to dupIgnore, trying to add a duplicate string to a sorted list does nothing.

See Also

Append (see page 3041)

Delete (see page 3015)

Duplicates (see page 3025)

Insert (see page 3019)

Sorted (see page 3026)

TextToShortCut (see page 16196)

ShortCutToText (see page 16195)

Objects

3.1.8.2.2 TShortCutList.IndexOfShortCut Method

Returns the position of a shortcut in the list.

Pascal

```
function IndexOfShortCut(const Shortcut: TShortCut): Integer;
```

C++

```
__fastcall int IndexOfShortCut(const TShortCut Shortcut);
```

Description

Call IndexOfShortCut to locate a shortcut in the list. IndexOfShortCut returns the position of the first occurrence of Shortcut in the list, where 0 is the position of the first shortcut, 1 is the position of the second shortcut, and so on. If the shortcut does not appear in the list, IndexOfShortCut returns –1.

Note: To locate a shortcut based on its string representation rather than the shortcut value, use the Find or IndexOf method instead.

See Also

Find (see page 3016)

IndexOf (see page 3019)

3.1.8.3 TShortCutList Properties

The properties of the TShortCutList class are listed here.

Public Properties

	Name	Description
	ShortCuts (see page 333)	Lists the shortcuts in the list.

3.1.8.3.1 TShortCutList.ShortCuts Property

Lists the shortcuts in the list.

Pascal

```
property ShortCuts [Index: Integer]: TShortCut;
```

C++

```
__property TShortCut ShortCuts[int Index];
```

Description

Use ShortCuts to access the shortcuts in the list by position. The Index parameter indicates the index of the shortcut, where 0 is the index of the first shortcut, 1 is the index of the second shortcut, and so on.

ShortCuts is the same as the Objects property, except that the value is given as a TShortCut value instead of as a reference to an object instance.

Warning: Do not set the Objects property, or the short cut list will fail to work correctly.

ShortCuts is a read-only property. To add or delete shortcuts in the list, use the Add (see page 332) and Delete methods. To locate a specific shortcut in the list, use the IndexOfShortCut (see page 332) method.

See Also

Objects

Add (see page 332)

Delete (see page 3015)

IndexOfShortCut (see page 332)

Count

3.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CreateAction (see page 334)	Creates an action of a specified type.
	EnumRegisteredActions (see page 334)	Iterates through the list of registered actions, passing them to a user-defined callback.
	RegisterActions (see page 335)	Registers action classes so that they appear in the list of predefined actions in the action list editor or customize dialog.
	UnRegisterActions (see page 336)	Unregisters a set of actions.

3.2.1 ActnList.CreateAction Function

Creates an action of a specified type.

Pascal

```
function CreateAction(AOwner: TComponent; ActionClass: TBasicActionClass): TBasicAction;
```

C++

```
TBasicAction CreateAction(TComponent * AOwner, TBasicActionClass ActionClass);
```

File

ActnList

Description

CreateAction creates an action of type ActionClass. AOwner is typically the form or data module. CreateAction is called automatically when you add an action to an action list or action manager at design time. Calling this function does the same thing as calling ActionClass.Create (Delphi) or the constructor of the specified action class (C++). The only exception is that CreateAction uses the Resource parameter of the RegisterActions (see page 335) procedure to initialize the values of the new action object.

See Also

RegisterActions (see page 335)

EnumRegisteredActions (see page 334)

3.2.2 ActnList.EnumRegisteredActions Function

Iterates through the list of registered actions, passing them to a user-defined callback.

Pascal

```
procedure EnumRegisteredActions(Proc: TEnumActionProc; Info: Pointer);
```

C++

```
EnumRegisteredActions(TEnumActionProc Proc, void * Info);
```

File

ActnList

Description

EnumRegisteredActions iterates the list of registered actions, passing them to the user-defined callback, Proc.

Category is the category of the action. For TContainedAction (see page 289) descendants, this is the Category property; for ancestor classes you can pass in an empty string. The ActionClass parameter describes the class type of the action object. The Info parameter of the callback is the same as the Info parameter passed to EnumRegisteredActions. It allows the user to pass custom information to the callback.

See Also

RegisterActions (see page 335)

TContainedAction (see page 289)

3.2.3 ActnList.RegisterActions Function

Registers action classes so that they appear in the list of predefined actions in the action list editor or customize dialog.

Pascal

```
procedure RegisterActions(const CategoryName: string; const AClasses: array of
TBasicActionClass; Resource: TComponentClass);
```

C++

```
RegisterActions(const AnsiString CategoryName, const array of TBasicActionClass AClasses,
TComponentClass Resource);
```

File

ActnList

Description

RegisterActions allows a class to appear in the action list editor or customize dialog as a selection when you choose New Action or Add Standard Action. Predefined actions are registered automatically. To install your own action classes so that they can be added to an action list or action manager, call RegisterActions to register them.

The CategoryName parameter specifies the value of the actions' Category property.

The AClasses parameter is an array of custom action class names.

Note: In C++, the AClasses_Size parameter specifies the index of the last entry in the AClasses array (one less than the number of classes to register).

The Resource parameter lets you assign default values to the properties of the actions being registered. Resource is the class of a TDataModule descendant that contains instances of the action classes listed by AClasses, where the properties of those instances are set to the default values. This parameter can be set to nil (Delphi) or NULL (C++) if there is no need to assign default values.

See Also

UnRegisterActions (see page 336)

TBasicAction

TContainedAction (see page 289)

3.2.4 ActnList.UnRegisterActions Function

Unregisters a set of actions.

Pascal

```
procedure UnRegisterActions(const AClasses: array of TBasicActionClass);
```

C++

```
UnRegisterActions(const array of TBasicActionClass AClasses);
```

File

ActnList

Description

UnRegisterActions unregisters a set of actions that were registered with the RegisterActions (see page 335) procedure. The AClasses parameter is a list of registered action classes (descendants of TBasicAction) that are to be unregistered. In C++, the AClasses_Size parameter indicates the index of the last action class specified in AClasses (one less than the total number of classes).

See Also

RegisterActions (see page 335)

3.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TActionListState (see page 337)	This is record ActnList.TActionListState.

3.3.1 ActnList.TActionListState Enumeration

Pascal

```
TActionListState = (
  asNormal,
  asSuspended,
  asSuspendedEnabled
);
```

C++

```
enum TActionListState {
  asNormal,
  asSuspended,
  asSuspendedEnabled
};
```

File

ActnList

Description

This is record ActnList.TActionListState.

3.4 Types

The following table lists types in this documentation.

Types

Name	Description
TActionEvent (see page 338)	This is type ActnList.TActionEvent.
TActionLinkClass (see page 338)	This is type ActnList.TActionLinkClass.
TContainedActionClass (see page 339)	This is type ActnList.TContainedActionClass.
TEnumActionProc (see page 339)	This is type ActnList.TEnumActionProc.
THintEvent (see page 339)	THintEvent is the type of the OnHint event handler.

3.4.1 ActnList.TActionEvent Type

Pascal

```
TActionEvent = procedure (Action: TBasicAction; var Handled: Boolean) of object;
```

C++

```
(Action: TBasicAction; var Handled: Boolean) ( TActionEvent)();
```

File

ActnList

Description

This is type ActnList.TActionEvent.

3.4.2 ActnList.TActionLinkClass Type

Pascal

```
TActionLinkClass = class of TActionLink;
```

C++

```
TActionLink TActionLinkClass;
```

File

ActnList

Description

This is type ActnList.TActionLinkClass.

3.4.3 ActnList.TContainedActionClass Type

Pascal

```
TContainedActionClass = class of TContainedAction;
```

C++

```
TContainedAction TContainedActionClass;
```

File

ActnList

Description

This is type ActnList.TContainedActionClass.

3.4.4 ActnList.TEnumActionProc Type

Pascal

```
TEnumActionProc = procedure (const Category: string; ActionClass: TBasicActionClass; Info:
Pointer) of object;
```

C++

```
(const Category: string; ActionClass: TBasicActionClass; Info: Pointer) ( TEnumActionProc)();
```

File

ActnList

Description

This is type ActnList.TEnumActionProc.

3.4.5 ActnList.THintEvent Type

THintEvent is the type of the OnHint event handler.

Pascal

```
THintEvent = procedure (var HintStr: string; var CanShow: Boolean) of object;
```

C++

```
(var HintStr: string; var CanShow: Boolean) ( THintEvent)();
```

File

ActnList

Description

HintStr is the string that will appear in the pop-up help hint. The event handler can change this string to change the hint that appears to the user.

CanShow indicates whether the hint specified by HintStr should be shown using the default popup window after the event

handler exits. Set this to false to suppress the default behavior when the event handler displays the hint.

3.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CreateActionProc (see page 341)	This is constant ActnList.CreateActionProc.
EnumRegisteredActionsProc (see page 341)	This is constant ActnList.EnumRegisteredActionsProc.
RegisterActionsProc (see page 342)	This is constant ActnList.RegisterActionsProc.
UnRegisterActionsProc (see page 342)	This is constant ActnList.UnRegisterActionsProc.

3.5.1 ActnList.CreateActionProc Constant

Pascal

```
CreateActionProc: function (AOwner: TComponent; ActionClass: TBasicActionClass): TBasicAction
= nil;
```

C++

```
function (AOwner: TComponent; ActionClass: TBasicActionClass): TBasicAction CreateActionProc =
nil;
```

File

ActnList

Description

This is constant ActnList.CreateActionProc.

3.5.2 ActnList.EnumRegisteredActionsProc Constant

Pascal

```
EnumRegisteredActionsProc: procedure (Proc: TEnumActionProc; Info: Pointer) = nil;
```

C++

```
procedure (Proc: TEnumActionProc; Info: Pointer) EnumRegisteredActionsProc = nil;
```

File

ActnList

Description

This is constant ActnList.EnumRegisteredActionsProc.

3.5.3 ActnList.RegisterActionsProc Constant

Pascal

```
RegisterActionsProc: procedure (const CategoryName: string; const AClasses: array of
TBasicActionClass; Resource: TComponentClass) = nil;
```

C++

```
procedure (const CategoryName: string; const AClasses: array of TBasicActionClass; Resource:
TComponentClass) RegisterActionsProc = nil;
```

File

ActnList

Description

This is constant ActnList.RegisterActionsProc.

3.5.4 ActnList.UnRegisterActionsProc Constant

Pascal

```
UnRegisterActionsProc: procedure (const AClasses: array of TBasicActionClass) = nil;
```

C++

```
procedure (const AClasses: array of TBasicActionClass) UnRegisterActionsProc = nil;
```

File

ActnList

Description

This is constant ActnList.UnRegisterActionsProc.

4 ActnMan Namespace

This is namespace ActnMan.

Topics

Name	Description
Classes (see page 346)	The following table lists classes in this documentation.
Interfaces (see page 560)	The following table lists interfaces in this documentation.
Functions (see page 563)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 564)	The following table lists structs, records, enums in this documentation.
Types (see page 568)	The following table lists types in this documentation.
Variables (see page 575)	The following table lists variables in this documentation.
Constants (see page 577)	The following table lists constants in this documentation.

Classes

	Name	Description
	TActionBarItem (see page 346)	TActionBarItem objects are used by the action manager to represent all of the action bars connected to the action manager in an application.
	TActionBars (see page 352)	TActionBars is a collection of TActionBarItem (see page 346) objects used internally by the action manager to represent all of the action bars connected to the action manager in an application.
	TActionBarStyle (see page 358)	TActionBarStyle is the base class for styles in ActionManager.
	TActionBarStyleList (see page 360)	This is class ActnMan.TActionBarStyleList.
	TActionClient (see page 364)	TActionClient is the ancestor for TActionBartItem and TActionClientItem (see page 374) classes.
	TActionClientItem (see page 374)	TActionClientItem is a streamable wrapper that represents an action or category of actions in the action manager.
	TActionClientLink (see page 389)	This is class ActnMan.TActionClientLink.
	TActionClients (see page 400)	TActionClients is a container for TActionClient (see page 364) objects.
	TActionClientsCollection (see page 408)	TActionClientsCollection is a collection of action clients that maintains information about the action manager that owns the collection.
	TActionDragBaseClass (see page 415)	TActionDragBaseClass is a base class used for managing drag operations.
	TActionDragObject (see page 418)	This is class ActnMan.TActionDragObject.
	TActionItemDragObject (see page 422)	This is class ActnMan.TActionItemDragObject.
	TActionListCollection (see page 425)	TActionListCollection is a collection that maintains information about action lists and their captions.
	TActionListItem (see page 428)	TActionListItem represents an item in a action list collection.
	TActionManager (see page 432)	TActionManager provides a mechanism to manage and display all actions contained in an application.

	TCategoryDragObject (see page 444)	This is class ActnMan.TCategoryDragObject.
	TCustomActionBar (see page 447)	TCustomActionBar is the base class for all action bands.
	TCustomActionBarColorMap (see page 492)	TCustomActionBarColorMap is the base class for the action band colormaps that controls the color of the action band and the items that appear on it.
	TCustomActionClass (see page 506)	This is class ActnMan.TCustomActionClass.
	TCustomActionControl (see page 507)	TCustomActionControl is the base class for ActionBand custom controls.
	TCustomActionManager (see page 537)	TCustomActionManager is the base class for the TActionManager (see page 432) component.
	TXToolDockForm (see page 559)	This is class ActnMan.TXToolDockForm.

Constants

Name	Description
caAllChanges (see page 577)	This is constant ActnMan.caAllChanges.
cDefaultSchedule (see page 577)	This is constant ActnMan.cDefaultSchedule.
CM_ENTERMENULOOP (see page 578)	This is constant ActnMan.CM_ENTERMENULOOP.
CM_ITEMCLICKED (see page 578)	This is constant ActnMan.CM_ITEMCLICKED.
CM_ITEMDROPPOINT (see page 578)	This is constant ActnMan.CM_ITEMDROPPOINT.
CM_ITEMKEYED (see page 579)	This is constant ActnMan.CM_ITEMKEYED.
CM_ITEMSELECTED (see page 579)	This is constant ActnMan.CM_ITEMSELECTED.
CM_RESETBAR (see page 579)	This is constant ActnMan.CM_RESETBAR.
CM_SCROLLCLICKED (see page 580)	This is constant ActnMan.CM_SCROLLCLICKED.
ControlDragMode (see page 580)	This is constant ActnMan.ControlDragMode.

Functions

	Name	Description
	NotifyDesigner (see page 563)	This is function ActnMan.NotifyDesigner.

Interfaces

	Name	Description
	IActionBarDesigner (see page 560)	This is class ActnMan.IActionBarDesigner.

Structs, Records, Enums

	Name	Description
	TBackgroundLayout (see page 564)	This is record ActnMan.TBackgroundLayout.
	TBarEdge (see page 564)	This is record ActnMan.TBarEdge.
	TBarOrientation (see page 565)	This is record ActnMan.TBarOrientation.
	TCaptionOptions (see page 565)	This is record ActnMan.TCaptionOptions.
	TChangesAllowed (see page 566)	This is record ActnMan.TChangesAllowed.
	TCMItemMsg (see page 566)	This is record ActnMan.TCMItemMsg.

Types

Name	Description
TActionBarItemClass (see page 568)	This is type ActnMan.TActionBarItemClass.
TActionBarsClass (see page 568)	This is type ActnMan.TActionBarsClass.
TActionClientClass (see page 569)	This is type ActnMan.TActionClientClass.
TActionClientItemClass (see page 569)	This is type ActnMan.TActionClientItemClass.
TActionClientLinkClass (see page 569)	This is type ActnMan.TActionClientLinkClass.
TActionClientsClass (see page 570)	This is type ActnMan.TActionClientsClass.
TActionProc (see page 570)	This is type ActnMan.TActionProc.
TChangesAllowedSet (see page 570)	This is type ActnMan.TChangesAllowedSet.
TColorMapUpdateProc (see page 571)	This is type ActnMan.TColorMapUpdateProc.
TControlCreatedEvent (see page 571)	This is type ActnMan.TControlCreatedEvent.
TCustomActionBarClass (see page 571)	This is type ActnMan.TCustomActionBarClass.
TCustomActionControlClass (see page 572)	This is type ActnMan.TCustomActionControlClass.
TCustomColorMapClass (see page 572)	This is type ActnMan.TCustomColorMapClass.

TGetBarItemClassEvent (see page 572)	This is type ActnMan.TGetBarItemClassEvent.
TGetBarsClassEvent (see page 573)	This is type ActnMan.TGetBarsClassEvent.
TGetClientItemClassEvent (see page 573)	This is type ActnMan.TGetClientItemClassEvent.
TGetClientsClassEvent (see page 573)	This is type ActnMan.TGetClientsClassEvent.
TGetControlClassEvent (see page 574)	This is type ActnMan.TGetControlClassEvent.
TMergeRange (see page 574)	This is type ActnMan.TMergeRange.
TStyleChanged (see page 574)	This is type ActnMan.TStyleChanged.

Variables

Name	Description
ActionBarDesigner (see page 575)	This is variable ActnMan.ActionBarDesigner.
ActionBarStyles (see page 575)	This is variable ActnMan.ActionBarStyles.
DefaultActnBarStyle (see page 575)	This is variable ActnMan.DefaultActnBarStyle.

4.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TActionBarItem (see page 346)	TActionBarItem objects are used by the action manager to represent all of the action bars connected to the action manager in an application.
	TActionBars (see page 352)	TActionBars is a collection of TActionBarItem (see page 346) objects used internally by the action manager to represent all of the action bars connected to the action manager in an application.
	TActionBarStyle (see page 358)	TActionBarStyle is the base class for styles in ActionManager.
	TActionBarStyleList (see page 360)	This is class ActnMan.TActionBarStyleList.
	TActionClient (see page 364)	TActionClient is the ancestor for TActionBartItem and TActionClientItem (see page 374) classes.
	TActionClientItem (see page 374)	TActionClientItem is a streamable wrapper that represents an action or category of actions in the action manager.
	TActionClientLink (see page 389)	This is class ActnMan.TActionClientLink.
	TActionClients (see page 400)	TActionClients is a container for TActionClient (see page 364) objects.
	TActionClientsCollection (see page 408)	TActionClientsCollection is a collection of action clients that maintains information about the action manager that owns the collection.
	TActionDragBaseClass (see page 415)	TActionDragBaseClass is a base class used for managing drag operations.
	TActionDragObject (see page 418)	This is class ActnMan.TActionDragObject.
	TActionItemDragObject (see page 422)	This is class ActnMan.TActionItemDragObject.
	TActionListCollection (see page 425)	TActionListCollection is a collection that maintains information about action lists and their captions.
	TActionListItem (see page 428)	TActionListItem represents an item in a action list collection.
	TActionManager (see page 432)	TActionManager provides a mechanism to manage and display all actions contained in an application.
	TCategoryDragObject (see page 444)	This is class ActnMan.TCategoryDragObject.
	TCustomActionBar (see page 447)	TCustomActionBar is the base class for all action bands.
	TCustomActionBarColorMap (see page 492)	TCustomActionBarColorMap is the base class for the action band colormaps that controls the color of the action band and the items that appear on it.
	TCustomActionClass (see page 506)	This is class ActnMan.TCustomActionClass.
	TCustomActionControl (see page 507)	TCustomActionControl is the base class for ActionBand custom controls.
	TCustomActionManager (see page 537)	TCustomActionManager is the base class for the TActionManager (see page 432) component.
	TXToolDockForm (see page 559)	This is class ActnMan.TXToolDockForm.

4.1.1 TActionBarItem Class

TActionBarItem objects are used by the action manager to represent all of the action bars connected to the action manager in an application.

Class Hierarchy

Pascal

```
TActionBarItem = class(TActionClient);
```

C++

```
class TActionBarItem : public TActionClient;
```

File

ActnMan

Description

TActionBarItem is used to wrap a category of actions into a streamable format that is managed by an action manager. TActionBarItem is used as a mechanism to store layout (ordering) information for a set of actions. This layout information is used by the action manager to render the actions in the UI.

Instances of TActionBarItem are created automatically by the TActionManager (see page 432) component when an action or category of actions is added to an action band.

Note: Do not create or destroy instances of TActionBarItem directly, always allow the action manager to handle this for you.

See Also

TActionManager (see page 432)

4.1.1.1 TActionBarItem Members

The following tables list the members exposed by TActionBarItem.

Protected Methods

	Name	Description
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	ClearActionBar (see page 367)	This is ClearActionBar, a member of class TActionClient.
V	ColorChanged (see page 349)	This is ColorChanged, a member of class TActionBarItem.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FreeChildren (see page 368)	This is FreeChildren, a member of class TActionClient.
V	GetDisplayName (see page 350)	Returns the name of the collection item as it appears in the collection editor.
V	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	HasVisibleItems (see page 368)	This is HasVisibleItems, a member of class TActionClient.
	IsChildItem (see page 368)	This is IsChildItem, a member of class TActionClient.
V	SetChangesAllowed (see page 369)	This is SetChangesAllowed, a member of class TActionClient.
V	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
V	SetColor (see page 369)	This is SetColor, a member of class TActionClient.
V	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
V	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.
V	SetVisible (see page 350)	This is SetVisible, a member of class TActionBarItem.

Protected Properties

	Name	Description
R	ActionBars (see page 351)	This is ActionBars, a member of class TActionBarItem.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 367)	Copies the contents of another object to the current one.
V	Assign (see page 2930)	Copies the contents of another, similar object.

V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 349)	Creates and initializes a TActionBarItem (see page 346) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 366)	Destroys the action client and frees its memory.
V	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	HasBackground (see page 368)	Indicates whether a background image exists.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Refresh (see page 350)	Causes the controls on the ActionBar (see page 351) to be realigned.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Accessible (see page 370)	This is Accessible, a member of class TActionClient.
	ChildActionBar (see page 372)	Specifies the action band that is used to render the TActionClients (see page 400) object specified in the Items (see page 373) property.
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	HasItems (see page 373)	Specifies whether the action client has sub-items associated with the action client.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
R	OwningCollection (see page 373)	This is OwningCollection, a member of class TActionClient.

Published Properties

	Name	Description
	ActionBar (see page 351)	Specifies the action band the action client is rendered on.
	AutoSize (see page 351)	Specifies whether the ActionBar (see page 351) control associated with this item sizes itself automatically to accommodate its contents.

	Background (see page 370)	Specifies the image that is drawn on the action client's action band for the action client.
	BackgroundLayout (see page 371)	Specifies how the image specified in the Background (see page 370) property is rendered on the action band used by the action client to render any sub-items that are contained by the action client.
	ChangesAllowed (see page 371)	Indicates which kinds of changes to the action client are permissible.
	Color (see page 372)	Specifies the background color for TClientAction when it is rendered on a action band.
	ContextItems (see page 372)	Specifies the TActionClients (see page 400) object that is rendered as a popup menu when a item is right clicked.
	GlyphLayout (see page 352)	Describes where the images should be for all the actions on the ActionBand associated with TActionBarItem (see page 346).
	Items (see page 373)	Specifies the TActionClients (see page 400) object that contains the sub-items of the action client.
	Tag (see page 374)	This is Tag, a member of class TActionClient.
	Visible (see page 374)	Determines whether the action client is visible.

4.1.1.2 TActionBarItem.Create Constructor

Creates and initializes a TActionBarItem (see page 346) object.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TActionBarItem(TCollection Collection);
```

Description

The Create method takes as a parameter the name of a TCollection instance. Create is called automatically by action manager when a new instance of a TActionBarItem (see page 346) is required.

4.1.1.3 TActionBarItem Methods

The methods of the TActionBarItem class are listed here.

Protected Methods

	Name	Description
	ColorChanged (see page 349)	This is ColorChanged, a member of class TActionBarItem.
	GetDisplayName (see page 350)	Returns the name of the collection item as it appears in the collection editor.
	SetVisible (see page 350)	This is SetVisible, a member of class TActionBarItem.

Public Methods

	Name	Description
	Refresh (see page 350)	Causes the controls on the ActionBar (see page 351) to be realigned.

4.1.1.3.1 TActionBarItem.ColorChanged Method

Pascal

```
procedure ColorChanged; override;
```

C++

```
virtual __fastcall ColorChanged();
```

Description

This is ColorChanged, a member of class TActionBarItem.

4.1.1.3.2 TActionBarItem.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName (see page 2761)

GetNamePath (see page 2759)

4.1.1.3.3 TActionBarItem.Refresh Method

Causes the controls on the ActionBar (see page 351) to be realigned.

Pascal

```
procedure Refresh; override;
```

C++

```
virtual __fastcall Refresh();
```

Description

If the TActionBarItem (see page 346) is associated with an ActionBar (see page 351), Refresh causes the controls on the ActionBar (see page 351) to be realigned.

4.1.1.3.4 TActionBarItem.SetVisible Method

Pascal

```
procedure SetVisible(const Value: Boolean); override;
```

C++

```
virtual __fastcall SetVisible(const Boolean Value);
```

Description

This is SetVisible, a member of class TActionBarItem.

4.1.1.4 TActionBarItem Properties

The properties of the TActionBarItem class are listed here.

Protected Properties

	Name	Description
R	ActionBars (see page 351)	This is ActionBars, a member of class TActionBarItem.

Published Properties

	Name	Description
	ActionBar (see page 351)	Specifies the action band the action client is rendered on.
	AutoSize (see page 351)	Specifies whether the ActionBar (see page 351) control associated with this item sizes itself automatically to accommodate its contents.
	GlyphLayout (see page 352)	Describes where the images should be for all the actions on the ActionBand associated with TActionBarItem (see page 346).

4.1.1.4.1 TActionBarItem.ActionBar Property

Specifies the action band the action client is rendered on.

Pascal

```
property ActionBar: TCustomActionBar;
```

C++

```
__property TCustomActionBar ActionBar;
```

Description

ActionBar is used internally to specify the action band that the action client is rendered on.

4.1.1.4.2 TActionBarItem.ActionBars Property

Pascal

```
property ActionBars: TActionBars;
```

C++

```
__property TActionBars ActionBars;
```

Description

This is ActionBars, a member of class TActionBarItem.

4.1.1.4.3 TActionBarItem.AutoSize Property

Specifies whether the ActionBar (see page 351) control associated with this item sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the TActionBar control associated with this item sizes itself automatically. For example, AutoSize controls whether or not TCustomActionToolbars wrap. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

4.1.1.4.4 TActionBarItem.GlyphLayout Property

Describes where the images should be for all the actions on the ActionBand associated with TActionBarItem (see page 346).

Pascal

```
property GlyphLayout: TButtonLayout;
```

C++

```
__property TButtonLayout GlyphLayout;
```

Description

GlyphLayout describes where the image should be placed on the control. The description is in terms of a TButtonLayout value; see the documentation for that type for more information.

4.1.2 TActionBars Class

TActionBars is a collection of TActionBarItem (see page 346) objects used internally by the action manager to represent all of the action bars connected to the action manager in an application.

Class Hierarchy

Pascal

```
TActionBars = class(TActionClientsCollection);
```

C++

```
class TActionBars : public TActionClientsCollection;
```

File

ActnMan

Description

TActionBars is used by the action manager to maintain current layout information about all the action bars connected to the action manager in an application. TActionBars allows the action manager to write this information to disk in a file or to a stream. TActionBars also allows the action manager to reconstruct all of the action bars based on information contained in a specified file or a stream.

See Also

TCollection (see page 2731)

TCollectionItem (see page 2755)

TPersistent (see page 2928)

TPersistent (see page 2928)

Creating Dynamic Menus

Creating Most Recently Used Lists

Hiding Unused Items and Categories in Action Bands

4.1.2.1 TActionBars Members

The following tables list the members exposed by TActionBars.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	DoShowHint (see page 355)	This is DoShowHint, a member of class TActionBars.
	GetActionManager (see page 412)	Initializes the collection after the constructor is finished.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2915)	Returns the owner of an object.
	GetOwner (see page 2932)	Returns the owner of an object.
	InternalRethinkHotkeys (see page 412)	Adjusts the captions of child items so that every item has an accelerator and there are no duplicate accelerator keys.
	Loaded (see page 356)	This is Loaded, a member of class TActionBars.
V	Notify (see page 356)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
	ActionClients (see page 413)	Lists the action clients in this collection.
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	ParentItem (see page 415)	Indicates the action client object that uses this collection to maintain its subitems.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 355)	Creates a new TActionBarItem (see page 346) instance and adds it to the Items array.
V	AfterConstruction (see page 411)	Initializes the collection after the constructor is finished.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 355)	Creates and initializes a TActionBars (see page 352) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IterateClients (see page 412)	Executes a callback for every child item in the collection and any child items of those children.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ActionBars (see page 357)	Contains a collection of all of the action bars connected to the action manager.
R	ActionManager (see page 414)	Indicates the action manager that owns the TActionClientsCollection (see page 408) object.
	AutoHotKeys (see page 414)	Specifies whether accelerator keys are auto-generated for items that do not have them assigned.
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 2751)	Lists the items in the collection.

Published Properties

	Name	Description
	Customizable (see page 357)	Indicates whether the action clients collection is customizable.
	HintShortCuts (see page 357)	A forward to the application object's property of the same name.
	SessionCount (see page 358)	Returns the number of times the application has been run.
	ShowHints (see page 358)	Indicates whether hints should be shown for items in the collection.

4.1.2.2 TActionBars.Create Constructor

Creates and initializes a TActionBars (see page 352) object.

Pascal

```
constructor Create(AOwner: TPersistent; ItemClass: TCollectionItemClass);
```

C++

```
__fastcall TActionBars(TPersistent * AOwner, TCollectionItemClass ItemClass);
```

Description

The Create method takes as parameters the owner and the name of a TCollectionItemClass instance. Create is called automatically by action manager when an instance of a TActionBars (see page 352) is required.

4.1.2.3 TActionBars Methods

The methods of the TActionBars class are listed here.

Protected Methods

	Name	Description
	DoShowHint (see page 355)	This is DoShowHint, a member of class TActionBars.
	Loaded (see page 356)	This is Loaded, a member of class TActionBars.
	Notify (see page 356)	Responds when items are added to or removed from the collection.

Public Methods

	Name	Description
	Add (see page 355)	Creates a new TActionBarItem (see page 346) instance and adds it to the Items array.

4.1.2.3.1 TActionBars.Add Method

Creates a new TActionBarItem (see page 346) instance and adds it to the Items array.

Pascal

```
function Add: TActionBarItem;
```

C++

```
__fastcall TActionBarItem Add();
```

Description

Add is called automatically when an action band is connected to an action manager component.

4.1.2.3.2 TActionBars.DoShowHint Method

Pascal

```
procedure DoShowHint(AClient: TActionClient);
```

C++

```
__fastcall DoShowHint(TActionClient AClient);
```

Description

This is DoShowHint, a member of class TActionBars.

4.1.2.3.3 TActionBars.Loaded Method

Pascal

```
procedure Loaded;
```

C++

```
__fastcall Loaded();
```

Description

This is Loaded, a member of class TActionBars.

4.1.2.3.4 TActionBars.Notify Method

Responds when items are added to or removed from the collection.

Pascal

```
procedure Notify(Item: TCollectionItem; Action: TCollectionNotification); override;
```

C++

```
virtual __fastcall Notify(TCollectionItem Item, TCollectionNotification Action);
```

Description

Notify is called automatically when the items in the collection change.

Item is the item that was just added to or that is about to be removed from the collection.

Action indicates whether item was added, is about to be removed, or is about to be deleted:

As implemented in TCollection, Notify calls Added when Action is cnAdded and calls Deleting when Action is cnDeleting. TCollection ignores the cnExtracting action. Descendant classes can override Notify to modify this behavior.

See Also

Added (see page 2736)

Deleting (see page 2741)

4.1.2.4 TActionBars Properties

The properties of the TActionBars class are listed here.

Public Properties

	Name	Description
R	ActionBars (see page 357)	Contains a collection of all of the action bars connected to the action manager.

Published Properties

	Name	Description
	Customizable (see page 357)	Indicates whether the action clients collection is customizable.
	HintShortCuts (see page 357)	A forward to the application object's property of the same name.
	SessionCount (see page 358)	Returns the number of times the application has been run.
	ShowHints (see page 358)	Indicates whether hints should be shown for items in the collection.

4.1.2.4.1 TActionBars.ActionBars Property

Contains a collection of all of the action bars connected to the action manager.

Pascal

```
property ActionBars [const Index: Integer]: TActionBarItem;
```

C++

```
__property TActionBarItem ActionBars[int const Index];
```

Description

The ActionBars property contains a collection of all of the action bands connected to the action manager. When an action is dropped on an action band or the ActionManager (see page 414) property of an action band is set to the action manager component, the action band (in the form of a TActionBarItem (see page 346)) is automatically added to this collection.

See Also

ActionManager (see page 414)

operator_[]

4.1.2.4.2 TActionBars.Customizable Property

Indicates whether the action clients collection is customizable.

Pascal

```
property Customizable: Boolean;
```

C++

```
__property Boolean Customizable;
```

Description

The default setting for Customizable is true, which indicates that the collection can be customized. Set Customizable to false if you don't want to allow users to change the items in this collection using the customize dialog. Collections that are not customizable appear grayed out in the customize dialog.

4.1.2.4.3 TActionBars.HintShortCuts Property

A forward to the application object's property of the same name.

Pascal

```
property HintShortCuts: Boolean;
```

C++

```
__property Boolean HintShortCuts;
```

Description

The application object contains a property called HintShortCuts which determines whether or not the application's popup hint windows display shortcut information. The HintShortCuts property in TActionBars (see page 352) is simply another way to access the application's HintShortCuts property.

See Also

HintShortCuts (see page 13716)

4.1.2.4.4 TActionBars.SessionCount Property

Returns the number of times the application has been run.

Pascal

```
property SessionCount: Integer;
```

C++

```
__property int SessionCount;
```

Description

The SessionCount property stores the number of times the application has been run. SessionCount is used internally, by the action manager, in the calculation to determine whether a TActionClientItem (see page 374) is hidden or displayed on an action bar.

See Also

PrioritySchedule (see page 555)

LastSession (see page 385)

UsageCount (see page 388)

4.1.2.4.5 TActionBars.ShowHints Property

Indicates whether hints should be shown for items in the collection.

Pascal

```
property ShowHints: Boolean;
```

C++

```
__property Boolean ShowHints;
```

Description

Set ShowHints to false if you don't want hints shown for objects in the collection. The default value is true.

See Also

ShowHint (see page 13721)

4.1.3 TActionBarStyle Class

TActionBarStyle is the base class for styles in ActionManager.

Class Hierarchy

Pascal

```
TActionBarStyle = class;
```

C++

```
class TActionBarStyle;
```

File

ActnMan

Description

TActionBarStyle provides the base methods for defining styles in the ActionManager component.

See Also

TActionManager (see page 432)

4.1.3.1 TActionBarStyle Members

The following tables list the members exposed by TActionBarStyle.

Public Methods

	Name	Description
	GetColorMapClass (see page 359)	Returns the class type of the colormap for an action bar.
	GetControlClass (see page 359)	Returns the control class type on an action bar item.
	GetStyleName (see page 360)	Returns the style associated with action manager.

4.1.3.2 TActionBarStyle Methods

The methods of the TActionBarStyle class are listed here.

Public Methods

	Name	Description
	GetColorMapClass (see page 359)	Returns the class type of the colormap for an action bar.
	GetControlClass (see page 359)	Returns the control class type on an action bar item.
	GetStyleName (see page 360)	Returns the style associated with action manager.

4.1.3.2.1 TActionBarStyle.GetColorMapClass Method

Returns the class type of the colormap for an action bar.

Pascal

```
function GetColorMapClass(ActionBar: TCustomActionBar): TCustomColorMapClass; virtual;
abstract;
```

C++

```
virtual __fastcall TCustomColorMapClass GetColorMapClass(TCustomActionBar ActionBar) = 0;
```

Description

Returns the class type TCustomActionBarColorMap (see page 492), which is a colormap associated with an action bar. Override this method to customize the appearance of your applications.

4.1.3.2.2 TActionBarStyle.GetControlClass Method

Returns the control class type on an action bar item.

Pascal

```
function GetControlClass(ActionBar: TCustomActionBar; AnItem: TActionClientItem):
TCustomActionControlClass; virtual; abstract;
```

C++

```
virtual __fastcall TCustomActionControlClass GetControlClass(TCustomActionBar ActionBar,
TActionClientItem AnItem) = 0;
```

Description

Returns the control class type on an item in the action bar, such as a button or menu item.

4.1.3.2.3 TActionBarStyle.GetStyleName Method

Returns the style associated with action manager.

Pascal

```
function GetStyleName: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetStyleName() = 0;
```

Description

Returns the value of the Style property.

4.1.4 TActionBarStyleList Class

Class Hierarchy

System::TObject → Classes.TPersistent → Classes.TStrings → Classes.TStringList → ActnMan.TActionBarStyleList

Pascal

```
TActionBarStyleList = class(TStringList);
```

C++

```
class TActionBarStyleList : public TStringList;
```

File

ActnMan

Description

This is class ActnMan.TActionBarStyleList.

4.1.4.1 TActionBarStyleList Members

The following tables list the members exposed by TActionBarStyleList.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 3013)	This is Changed, a member of class TStringList.
	Changing (see page 3013)	This is Changing, a member of class TStringList.
	CompareStrings (see page 3014)	Compares two strings.
	CompareStrings (see page 3048)	Compares two strings.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

V	DefineProperties (see page 3049)	Reads and writes the Strings (see page 3080) property as if it were published.
	Error (see page 3052)	This is the overview for the Error method overload.
	ExtractName (see page 3053)	Returns the name portion of a string that is a name value pair.
V	Get (see page 3017)	Returns a string given its index.
A	Get (see page 3053)	Returns a string given its index.
V	GetCapacity (see page 3017)	Returns the currently allocated size of the list of strings.
V	GetCapacity (see page 3054)	Returns the currently allocated size of the list of strings.
V	GetCount (see page 3018)	Returns the number of strings in the list
A	GetCount (see page 3054)	Returns the number of strings in the list
V	GetObject (see page 3018)	Returns the object associated with the string at a specified index.
V	GetObject (see page 3055)	Returns the object associated with the string at a specified index.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetTextStr (see page 3056)	Returns the value of the Text (see page 3084) property.
V	InsertItem (see page 3020)	This is InsertItem, a member of class TStringList.
V	Put (see page 3022)	Changes the value of the string with a specified index.
V	Put (see page 3064)	Changes the value of the string with a specified index.
V	PutObject (see page 3022)	Changes the object associated with the string at a specified index.
V	PutObject (see page 3065)	Changes the object associated with the string at a specified index.
V	SetCapacity (see page 3022)	Changes the amount of memory allocated to hold strings in the list.
V	SetCapacity (see page 3068)	Changes the amount of memory allocated to hold strings in the list.
V	SetTextStr (see page 3069)	Sets the Text (see page 3084) property.
V	SetUpdateState (see page 3023)	Performs internal adjustments before or after a series of updates.
V	SetUpdateState (see page 3069)	Performs internal adjustments before or after a series of updates.

Protected Properties

	Name	Description
R	UpdateCount (see page 3085)	Indicates the number of calls to BeginUpdate (see page 3045) that have not been matched by a call to EndUpdate (see page 3050).

Public Events

	Name	Description
	OnChange (see page 3028)	Occurs immediately after the list of strings changes.
	OnChanging (see page 3028)	Occurs immediately before the list of strings changes.

Public Methods

	Name	Description
V	Add (see page 3010)	Adds a new string to the list.
V	Add (see page 3033)	Adds a string at the end of the list.
V	AddObject (see page 3012)	Adds a string to the list, and associates an object with the string.
V	AddObject (see page 3036)	Adds a string to the list, and associates an object with the string.
V	AddStrings (see page 3039)	Adds a group of strings to the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 3041)	Adds a string to the list.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	Assign (see page 3043)	Sets the strings in the list, and possibly associated objects, from a source object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 3045)	Enables the TStrings (see page 3029) object to track when the list of strings is changing.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

V	Clear (see page 3013)	Deletes all the strings from the list.
A	Clear (see page 3047)	Introduces an abstract (Delphi) or pure virtual (C++) method to empty the list and any associated objects.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	CustomSort (see page 3014)	Sorts the strings in the list in a customized order.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Delete (see page 3015)	Removes the string specified by the Index parameter.
A	Delete (see page 3049)	Introduces an abstract (Delphi) or pure virtual (C++) method to delete a specified string from the list.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 3009)	Destroys an instance of TStringList (see page 3006).
V	Destroy (see page 3032)	Destroys the TPersistent instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 3050)	Enables the TStrings (see page 3029) object to keep track of when the list of strings has finished changing.
	Equals (see page 3051)	Compares the list of strings to the list from another TStrings (see page 3029) object and returns true if the two lists match.
V	Exchange (see page 3015)	Swaps the position of two strings in the list.
V	Exchange (see page 3053)	Swaps the position of two strings in the list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
V	Find (see page 3016)	Locates the index for a string in a sorted list and indicates whether a string with that value already exists in the list.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 3054)	Returns a TStrings (see page 3029) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetText (see page 3055)	Allocates a text buffer and fills it with the value of the Text (see page 3084) property.
V	IndexOf (see page 3019)	Returns the position of a string in the list.
V	IndexOf (see page 3056)	Returns the position of a string in the list.
V	IndexOfName (see page 3057)	Returns the position of the first name-value pair with the specified name.
V	IndexOfObject (see page 3059)	Returns the index of the first string in the list associated with a given object.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Insert (see page 3019)	Inserts a string to the list at the position specified by Index.
A	Insert (see page 3059)	Introduces abstract (Delphi) or pure virtual (C++) method to insert a string at a specified position.
V	InsertObject (see page 3020)	Inserts a string into the list at the specified position, and associates it with an object.
V	InsertObject (see page 3060)	Inserts a string into the list at the specified position, and associates it with an object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	LoadFromFile (see page 3061)	Fills the list with the lines of text in a specified file.
V	LoadFromStream (see page 3062)	Fills the list with lines of text read from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	Move (see page 3063)	Changes the position of a string in the list.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SaveToFile (see page 3065)	Saves the strings in the list to the specified file.

V	SaveToStream (see page 3067)	Writes the value of the Text (see page 3084) property to a stream object.
V	SetText (see page 3068)	Sets the Text (see page 3084) property.
V	Sort (see page 3023)	Sorts the strings in the list in ascending order.

Public Properties

	Name	Description
	Capacity (see page 3070)	Indicates the number of strings the TStrings (see page 3029) object can hold.
	CaseSensitive (see page 3025)	Controls (see page 5711) whether strings are located, sorted, and identified as duplicates in a case-sensitive or case-insensitive manner.
	CommaText (see page 3070)	Lists the strings in the TStrings (see page 3029) object in system data format (SDF).
R	Count (see page 3071)	Introduces an abstract property to represent the number of strings in the list.
	DelimitedText (see page 3074)	Represents all the strings in the TStrings (see page 3029) object as a single delimited string.
	Delimiter (see page 3074)	Specifies the delimiter used by the DelimitedText (see page 3074) property.
	Duplicates (see page 3025)	Specifies whether duplicate strings can be added to sorted lists.
	LineBreak (see page 3075)	Defines the line break character(s).
R	Names (see page 3075)	Indicates the name part of strings that are name-value pairs.
	NameValueSeparator (see page 3077)	Indicates the character used to separate names from values.
	Objects (see page 3077)	Represents a set of objects that are associated one with each of the strings in the Strings (see page 3080) property.
	QuoteChar (see page 3080)	Specifies the quote character used by the DelimitedText (see page 3074) property.
	Sorted (see page 3026)	Specifies whether the strings in the list should be automatically sorted.
	StrictDelimiter (see page 3080)	Determines how the Delimiter (see page 3074) property is used.
	Strings (see page 3080)	References the strings in the list by their positions.
	StringsAdapter (see page 3084)	Implements an IStringsAdapter (see page 3208) interface for the TStrings (see page 3029) object.
R	Style (see page 363)	This is Style, a member of class TActionBarStyleList.
	Text (see page 3084)	Lists the strings in the TStrings (see page 3029) object as a single string with the individual strings delimited by carriage returns and line feeds.
	ValueFromIndex (see page 3085)	Represents the value part of a string with a given index, on strings that are name-value pairs.
	Values (see page 3086)	Represents the value part of a string associated with a given name, on strings that are name-value pairs.

4.1.4.2 TActionBarStyleList Properties

The properties of the TActionBarStyleList class are listed here.

Public Properties

	Name	Description
R	Style (see page 363)	This is Style, a member of class TActionBarStyleList.

4.1.4.2.1 TActionBarStyleList.Style Property

Pascal

```
property Style [Index: Integer]: TActionBarStyle;
```

C++

```
__property TActionBarStyle Style[int Index];
```

Description

This is Style, a member of class TActionBarStyleList.

4.1.5 TActionClient Class

TActionClient is the ancestor for TActionBartItem and TActionClientItem (see page 374) classes.

Class Hierarchy

System::TObject → Classes.TPersistent → Classes.TCollectionItem → ActnMan.TActionClient

Pascal

```
TActionClient = class(TCollectionItem);
```

C++

```
class TActionClient : public TCollectionItem;
```

File

ActnMan

Description

TActionClient is the ancestor for the TActionBartItem and TActionClientItem (see page 374) classes.

See Also

TActionManager (see page 432)

TActionClients (see page 400)

4.1.5.1 TActionClient Members

The following tables list the members exposed by TActionClient.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	ClearActionBar (see page 367)	This is ClearActionBar, a member of class TActionClient.
	ColorChanged (see page 367)	This is ColorChanged, a member of class TActionClient.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FreeChildren (see page 368)	This is FreeChildren, a member of class TActionClient.
	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	HasVisibleItems (see page 368)	This is HasVisibleItems, a member of class TActionClient.
	IsChildItem (see page 368)	This is IsChildItem, a member of class TActionClient.
	SetChangesAllowed (see page 369)	This is SetChangesAllowed, a member of class TActionClient.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetColor (see page 369)	This is SetColor, a member of class TActionClient.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.
	SetVisible (see page 369)	This is SetVisible, a member of class TActionClient.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 367)	Copies the contents of another object to the current one.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 366)	Creates and initializes an action client object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 366)	Destroys the action client and frees its memory.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	HasBackground (see page 368)	Indicates whether a background image exists.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Refresh (see page 369)	Causes the action client to be redrawn.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Accessible (see page 370)	This is Accessible, a member of class TActionClient.
	ActionBar (see page 370)	Specifies the action band the action client is rendered on.
	ChildActionBar (see page 372)	Specifies the action band that is used to render the TActionClients (see page 400) object specified in the Items (see page 373) property.
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	HasItems (see page 373)	Specifies whether the action client has sub-items associated with the action client.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
R	OwningCollection (see page 373)	This is OwningCollection, a member of class TActionClient.

Published Properties

	Name	Description
	Background (see page 370)	Specifies the image that is drawn on the action client's action band for the action client.
	BackgroundLayout (see page 371)	Specifies how the image specified in the Background (see page 370) property is rendered on the action band used by the action client to render any sub-items that are contained by the action client.
	ChangesAllowed (see page 371)	Indicates which kinds of changes to the action client are permissible.
	Color (see page 372)	Specifies the background color for TClientAction when it is rendered on a action band.
	ContextItems (see page 372)	Specifies the TActionClients (see page 400) object that is rendered as a popup menu when a item is right clicked.
	Items (see page 373)	Specifies the TActionClients (see page 400) object that contains the sub-items of the action client.
	Tag (see page 374)	This is Tag, a member of class TActionClient.
	Visible (see page 374)	Determines whether the action client is visible.

4.1.5.2 TActionClient.Create Constructor

Creates and initializes an action client object.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TActionClient(TCollection Collection);
```

Description

The Create method takes as a parameter the name of a TCollection instance. Create is called automatically by action manager when a new instance of a TActionClient (see page 364) is required.

4.1.5.3 TActionClient.Destroy Destructor

Destroys the action client and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TActionClient();
```

Description

Destroy eliminates the TActionClient (see page 364) instance.

See Also

Free (see page 22300)

4.1.5.4 TActionClient Methods

The methods of the TActionClient class are listed here.

Protected Methods

	Name	Description
	ClearActionBar (see page 367)	This is ClearActionBar, a member of class TActionClient.
V	ColorChanged (see page 367)	This is ColorChanged, a member of class TActionClient.
	FreeChildren (see page 368)	This is FreeChildren, a member of class TActionClient.
	HasVisibleItems (see page 368)	This is HasVisibleItems, a member of class TActionClient.
	IsChildItem (see page 368)	This is IsChildItem, a member of class TActionClient.
V	SetChangesAllowed (see page 369)	This is SetChangesAllowed, a member of class TActionClient.
V	SetColor (see page 369)	This is SetColor, a member of class TActionClient.
V	SetVisible (see page 369)	This is SetVisible, a member of class TActionClient.

Public Methods

	Name	Description
V	Assign (see page 367)	Copies the contents of another object to the current one.
	HasBackground (see page 368)	Indicates whether a background image exists.
V	Refresh (see page 369)	Causes the action client to be redrawn.

4.1.5.4.1 TActionClient.Assign Method

Copies the contents of another object to the current one.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use the Assign procedure to copy the contents of the Source object to the current one. This procedure is an override of the inherited TPersistent method, and is designed to accurately copy the properties of one TActionClient (see page 364) to another if both objects are TActionClients (see page 400). If not, the inherited method is called. See the documentation for the inherited method for more details about the Assign method.

See Also

Assign (see page 2930)

4.1.5.4.2 TActionClient.ClearActionBar Method

Pascal

```
procedure ClearActionBar;
```

C++

```
__fastcall ClearActionBar();
```

Description

This is ClearActionBar, a member of class TActionClient.

4.1.5.4.3 TActionClient.ColorChanged Method

Pascal

```
procedure ColorChanged; virtual;
```

C++

```
virtual __fastcall ColorChanged();
```

Description

This is ColorChanged, a member of class TActionClient.

4.1.5.4.4 TActionClient.FreeChildren Method

Pascal

```
procedure FreeChildren;
```

C++

```
__fastcall FreeChildren();
```

Description

This is FreeChildren, a member of class TActionClient.

4.1.5.4.5 TActionClient.HasBackground Method

Indicates whether a background image exists.

Pascal

```
function HasBackground: Boolean;
```

C++

```
__fastcall Boolean HasBackground();
```

Description

Use the HasBackground method to determine whether or not a TActionClient (see page 364) has a background image.

4.1.5.4.6 TActionClient.HasVisibleItems Method

Pascal

```
function HasVisibleItems: Boolean;
```

C++

```
__fastcall Boolean HasVisibleItems();
```

Description

This is HasVisibleItems, a member of class TActionClient.

4.1.5.4.7 TActionClient.IsChildItem Method

Pascal

```
function IsChildItem(AClient: TActionClient): Boolean;
```

C++

```
__fastcall Boolean IsChildItem(TActionClient AClient);
```

Description

This is IsChildItem, a member of class TActionClient.

4.1.5.4.8 TActionClient.Refresh Method

Causes the action client to be redrawn.

Pascal

```
procedure Refresh; virtual;
```

C++

```
virtual __fastcall Refresh();
```

Description

Use Refresh to redraw the action client immediately.

4.1.5.4.9 TActionClient.SetChangesAllowed Method

Pascal

```
procedure SetChangesAllowed(const Value: TChangesAllowedSet); virtual;
```

C++

```
virtual __fastcall SetChangesAllowed(const TChangesAllowedSet Value);
```

Description

This is SetChangesAllowed, a member of class TActionClient.

4.1.5.4.10 TActionClient.SetColor Method

Pascal

```
procedure SetColor(const Value: TColor); virtual;
```

C++

```
virtual __fastcall SetColor(const TColor Value);
```

Description

This is SetColor, a member of class TActionClient.

4.1.5.4.11 TActionClient.SetVisible Method

Pascal

```
procedure SetVisible(const Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetVisible(const Boolean Value);
```

Description

This is SetVisible, a member of class TActionClient.

4.1.5.5 TActionClient Properties

The properties of the TActionClient class are listed here.

Public Properties

	Name	Description
	Accessible (see page 370)	This is Accessible, a member of class TActionClient.
	ActionBar (see page 370)	Specifies the action band the action client is rendered on.
	ChildActionBar (see page 372)	Specifies the action band that is used to render the TActionClients (see page 400) object specified in the Items (see page 373) property.
R	HasItems (see page 373)	Specifies whether the action client has sub-items associated with the action client.
R	OwningCollection (see page 373)	This is OwningCollection, a member of class TActionClient.

Published Properties

	Name	Description
	Background (see page 370)	Specifies the image that is drawn on the action client's action band for the action client.
	BackgroundLayout (see page 371)	Specifies how the image specified in the Background (see page 370) property is rendered on the action band used by the action client to render any sub-items that are contained by the action client.
	ChangesAllowed (see page 371)	Indicates which kinds of changes to the action client are permissible.
	Color (see page 372)	Specifies the background color for TClientAction when it is rendered on a action band.
	ContextItems (see page 372)	Specifies the TActionClients (see page 400) object that is rendered as a popup menu when a item is right clicked.
	Items (see page 373)	Specifies the TActionClients (see page 400) object that contains the sub-items of the action client.
	Tag (see page 374)	This is Tag, a member of class TActionClient.
	Visible (see page 374)	Determines whether the action client is visible.

4.1.5.5.1 TActionClient.Accessible Property

Pascal

```
property Accessible: IAccessible;
```

C++

```
__property IAccessible Accessible;
```

Description

This is Accessible, a member of class TActionClient.

4.1.5.5.2 TActionClient.ActionBar Property

Specifies the action band the action client is rendered on.

Pascal

```
property ActionBar: TCustomActionBar;
```

C++

```
__property TCustomActionBar ActionBar;
```

Description

ActionBar is used internally to specify the action band that the action client is rendered on.

4.1.5.5.3 TActionClient.Background Property

Specifies the image that is drawn on the action client's action band for the action client.

Pascal

```
property Background: TPicture;
```

C++

```
__property TPicture Background;
```

Description

Use Background to specify the image that is drawn on the action client's action band for the action client object itself.

See Also

BackgroundLayout (see page 371)

4.1.5.5.4 TActionClient.BackgroundLayout Property

Specifies how the image specified in the Background (see page 370) property is rendered on the action band used by the action client to render any sub-items that are contained by the action client.

Pascal

```
property BackgroundLayout: TBackgroundLayout;
```

C++

```
__property TBackgroundLayout BackgroundLayout;
```

Description

The values of BackgroundLayout are as follows:.

blNormal	Image is drawn normally, starting at coordinate 0,0 of the action band and the sub-item captions are rendered with a transparent background.
blStretch	Image is stretch drawn to fit the client area of the action band, starting at coordinate 0,0 of the action band and sub-item captions are rendered with a transparent background.
blTile	Image is tiled to fill the action band, sub-items are rendered with a transparent background
blLeftBanner	Image is appended to the left side of the action band.
blRightBanner	Image is appended to the right side of the action band.

See Also

Background (see page 370)

4.1.5.5.5 TActionClient.ChangesAllowed Property

Indicates which kinds of changes to the action client are permissible.

Pascal

```
property ChangesAllowed: TChangesAllowedSet;
```

C++

```
__property TChangesAllowedSet ChangesAllowed;
```

Description

Although action clients are part of a customizable user interface, allowing all changes may not necessarily be desirable. Use the ChangesAllowed property to indicate which changes to the action client are allowable. Possible values in the set are:

Value	Description
caModify	The item can be modified internally. For example, the visible caption can be changed.
caMove	The item can be relocated on the action band.
caDelete	The item can be removed from the action band.

Note: The property is a set, and can have multiple values at the same time. In fact, the default is to allow all three values (all changes are possible).

4.1.5.5.6 TActionClient.ChildActionBar Property

Specifies the action band that is used to render the TActionClients (see page 400) object specified in the Items (see page 373) property.

Pascal

```
property ChildActionBar: TCustomActionBar;
```

C++

```
__property TCustomActionBar ChildActionBar;
```

Description

ChildActionBar is used internally by the action manager and is automatically assigned when a TActionClient (see page 364) contains a nested TActionClients (see page 400) object. An example of this would be when the action client is a menu item that contains sub-menu items. The action band used to render the sub-items is automatically created for the sub-menu items and its reference is assigned to the ChildActionBar property.

See Also

Items (see page 373)

4.1.5.5.7 TActionClient.Color Property

Specifies the background color for TClientAction when it is rendered on a action band.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

The Color property determines the background color. You can set Color to one of the constants defined in the Graphics (see page 14387) unit (such as clBlue), or to an explicit RGB integer value.

4.1.5.5.8 TActionClient.ContextItems Property

Specifies the TActionClients (see page 400) object that is rendered as a popup menu when a item is right clicked.

Pascal

```
property ContextItems: TActionClients;
```

C++

```
__property TActionClients ContextItems;
```

Description

Use ContextItems to specify the TActionClients (see page 400) object that is rendered as a popup menu when a item is right clicked.

4.1.5.5.9 TActionClient.HasItems Property

Specifies whether the action client has sub-items associated with the action client.

Pascal

```
property HasItems: Boolean;
```

C++

```
__property Boolean HasItems;
```

Description

HasItems is used internally to determine if the action client has a TActionClients (see page 400) object assigned to the Items (see page 373) property.

See Also

Items (see page 373)

4.1.5.5.10 TActionClient.Items Property

Specifies the TActionClients (see page 400) object that contains the sub-items of the action client.

Pascal

```
property Items: TActionClients;
```

C++

```
__property TActionClients Items;
```

Description

If a TActionClientItem (see page 374) is used to represent a category of actions and not just a single action, the Items property is automatically set to the TActionClients (see page 400) object that contains the sub-items by the action manager. An example of this would be when an action category such as the standard actions "Edit" category is placed on a main menu action band. The Items property would contain the standard actions such as TEditCopy, TEditPaste, and TEditCut in a TActionClients (see page 400) object.

4.1.5.5.11 TActionClient.OwningCollection Property

Pascal

```
property OwningCollection: TActionClientsCollection;
```

C++

```
__property TActionClientsCollection OwningCollection;
```

Description

This is OwningCollection, a member of class TActionClient.

4.1.5.5.12 TActionClient.Tag Property

Pascal

```
property Tag: Integer;
```

C++

```
__property int Tag;
```

Description

This is Tag, a member of class TActionClient.

4.1.5.5.13 TActionClient.Visible Property

Determines whether the action client is visible.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action client.

4.1.6 TActionClientItem Class

TActionClientItem is a streamable wrapper that represents an action or category of actions in the action manager.

Class Hierarchy

Pascal

```
TActionClientItem = class(TActionClient);
```

C++

```
class TActionClientItem : public TActionClient;
```

File

ActnMan

Description

TActionClientItem is used internally to wrap a category of actions, standard action, or a user defined custom action into a streamable format that is managed by an action manager. TActionClientItem is used as a mechanism to store layout (ordering) information for an action. This layout information contained by the action client item is used by the action manager to render the action onto an action band as a visual representation of the action.

Instances of TActionClientItem are created automatically by the TActionManager (see page 432) component when an action or category of actions is added to an action band.

Note: Do not create or destroy instances of TActionClientItem directly, always allow the action manager to handle this for you.

See Also

TActionManager (see page 432)

Creating Dynamic Menus

Creating Most Recently Used Lists

Hiding Unused Items and Categories in Action Bands

4.1.6.1 TActionClientItem Members

The following tables list the members exposed by TActionClientItem.

Protected Methods

	Name	Description
	ActionChange (see page 378)	This is ActionChange, a member of class TActionClientItem.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	ClearActionBar (see page 367)	This is ClearActionBar, a member of class TActionClient.
V	ColorChanged (see page 379)	This is ColorChanged, a member of class TActionClientItem.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FreeChildren (see page 368)	This is FreeChildren, a member of class TActionClient.
	GetActionLinkClass (see page 379)	This is GetActionLinkClass, a member of class TActionClientItem.
V	GetDisplayName (see page 379)	Returns the name of the collection item as it appears in the collection editor.
V	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	HasVisibleItems (see page 368)	This is HasVisibleItems, a member of class TActionClient.
	IsChildItem (see page 368)	This is IsChildItem, a member of class TActionClient.
	ResetUsageData (see page 380)	This is ResetUsageData, a member of class TActionClientItem.
V	SetChangesAllowed (see page 380)	This is SetChangesAllowed, a member of class TActionClientItem.
V	SetCollection (see page 381)	Sets the value of the Collection property.
V	SetColor (see page 369)	This is SetColor, a member of class TActionClient.
V	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
V	SetIndex (see page 381)	Moves the collection item to a specified position in the parent collection.
V	SetVisible (see page 381)	This is SetVisible, a member of class TActionClientItem.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 378)	Copies the contents of another TActionClientItem (see page 374) to the object where the method is executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 377)	Creates and initializes a TActionClientItem (see page 374) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 377)	Destroys the action client and frees its memory.

V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	HasBackground (see page 368)	Indicates whether a background image exists.
	HasGlyph (see page 379)	Indicates whether or not there is a glyph (small icon) associated with an action on the action band.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 380)	Updates the action associated with the action client item.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Refresh (see page 380)	Causes the size of the action client item's control (if any) to be recalculated.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Unused (see page 382)	Determines whether a TActionClientItem (see page 374) is unused and should be hidden on an action band.

Public Properties

	Name	Description
	Accessible (see page 370)	This is Accessible, a member of class TActionClient.
	ActionBar (see page 370)	Specifies the action band the action client is rendered on.
R	ActionClients (see page 383)	Returns the TActionClients (see page 400) object that contains the TActionClientItem (see page 374).
	ActionLink (see page 383)	Refers to the action client item's action link.
	CheckUnused (see page 384)	Indicates whether the TActionClientItem (see page 374) object should be checked to see if it is unused and, if so, hidden from view.
	ChildActionBar (see page 372)	Specifies the action band that is used to render the TActionClients (see page 400) object specified in the Items (see page 373) property.
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	Control (see page 384)	Specifies the action band control that is rendered on an action band for the TActionClientItem (see page 374) object.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	HasItems (see page 373)	Specifies whether the action client has sub-items associated with the action client.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
R	OwningCollection (see page 373)	This is OwningCollection, a member of class TActionClient.
R	ParentItem (see page 386)	Refers to the parent item of the action client item.
R	Separator (see page 386)	Returns whether the TActionClientItem (see page 374) is rendered as a separator control on an action band.
R	ShortCutText (see page 387)	Contains the text label (like "Ctrl+S") for the shortcut.

Published Properties

	Name	Description
	Action (see page 382)	Specifies the action category or action that the TActionClientItem (see page 374) represents.

	Background (see page 370)	Specifies the image that is drawn on the action client's action band for the action client.
	BackgroundLayout (see page 371)	Specifies how the image specified in the Background (see page 370) property is rendered on the action band used by the action client to render any sub-items that are contained by the action client.
	Caption (see page 384)	Specifies the caption the action manager uses when the TActionClientItem (see page 374) is rendered on an action band.
	ChangesAllowed (see page 371)	Indicates which kinds of changes to the action client are permissible.
	Color (see page 372)	Specifies the background color for TClientAction when it is rendered on a action band.
	ContextItems (see page 372)	Specifies the TActionClients (see page 400) object that is rendered as a popup menu when a item is right clicked.
	HelpContext (see page 385)	This is HelpContext, a member of class TActionClientItem.
	ImageIndex (see page 385)	Contains the ImageIndex property for the client action item.
	Items (see page 373)	Specifies the TActionClients (see page 400) object that contains the sub-items of the action client.
	LastSession (see page 385)	Specifies the date an action client was last used.
	MergeIndex (see page 386)	Used to help merge action lists.
	ShortCut (see page 387)	Contains the ShortCut property for the action client item.
	ShowCaption (see page 387)	Determines whether the caption for the TActionClientItem (see page 374) is drawn.
	ShowGlyph (see page 388)	Indicates whether an image should be shown for the action.
	ShowShortCut (see page 388)	Determines whether the underbar character underneath the shortcut character is displayed.
	Tag (see page 374)	This is Tag, a member of class TActionClient.
	UsageCount (see page 388)	Specifies the total number of times an item has been used.
	Visible (see page 374)	Determines whether the action client is visible.

4.1.6.2 TActionClientItem.Create Constructor

Creates and initializes a TActionClientItem (see page 374) object.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TActionClientItem(TCollection Collection);
```

Description

The Create method takes as a parameter the name of a TCollection instance. Create is called automatically by action manager when a new instance of a TActionClientItem (see page 374) is required.

4.1.6.3 TActionClientItem.Destroy Destructor

Destroys the action client and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TActionClientItem();
```

Description

Destroy eliminates the TActionClientItem (see page 374) instance. Destroyis called automatically when the action manager needs to destroy the instance of a TActionClientItem (see page 374) object.

4.1.6.4 TActionClientItem Methods

The methods of the TActionClientItem class are listed here.

Protected Methods

	Name	Description
	ActionChange (see page 378)	This is ActionChange, a member of class TActionClientItem.
V	ColorChanged (see page 379)	This is ColorChanged, a member of class TActionClientItem.
	GetActionLinkClass (see page 379)	This is GetActionLinkClass, a member of class TActionClientItem.
V	GetDisplayName (see page 379)	Returns the name of the collection item as it appears in the collection editor.
	ResetUsageData (see page 380)	This is ResetUsageData, a member of class TActionClientItem.
V	SetChangesAllowed (see page 380)	This is SetChangesAllowed, a member of class TActionClientItem.
V	SetCollection (see page 381)	Sets the value of the Collection property.
V	SetIndex (see page 381)	Moves the collection item to a specified position in the parent collection.
V	SetVisible (see page 381)	This is SetVisible, a member of class TActionClientItem.

Public Methods

	Name	Description
V	Assign (see page 378)	Copies the contents of another TActionClientItem (see page 374) to the object where the method is executed.
	HasGlyph (see page 379)	Indicates whether or not there is a glyph (small icon) associated with an action on the action band.
V	InitiateAction (see page 380)	Updates the action associated with the action client item.
V	Refresh (see page 380)	Causes the size of the action client item's control (if any) to be recalculated.
	Unused (see page 382)	Determines whether a TActionClientItem (see page 374) is unused and should be hidden on an action band.

4.1.6.4.1 TActionClientItem.ActionChange Method

Pascal

```
procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); dynamic;
```

C++

```
__fastcall ActionChange(TObject * Sender, Boolean CheckDefaults);
```

Description

This is ActionChange, a member of class TActionClientItem.

4.1.6.4.2 TActionClientItem.Assign Method

Copies the contents of another TActionClientItem (see page 374) to the object where the method is executed.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use Assign to copy the contents of one TActionClientItem (see page 374) instance to another.

4.1.6.4.3 TActionClientItem.ColorChanged Method

Pascal

```
procedure ColorChanged; override;
```

C++

```
virtual __fastcall ColorChanged();
```

Description

This is ColorChanged, a member of class TActionClientItem.

4.1.6.4.4 TActionClientItem.GetActionLinkClass Method

Pascal

```
function GetActionLinkClass: TActionClientLinkClass; dynamic;
```

C++

```
__fastcall TActionClientLinkClass GetActionLinkClass();
```

Description

This is GetActionLinkClass, a member of class TActionClientItem.

4.1.6.4.5 TActionClientItem.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName (see page 2761)

GetNamePath (see page 2759)

4.1.6.4.6 TActionClientItem.HasGlyph Method

Indicates whether or not there is a glyph (small icon) associated with an action on the action band.

Pascal

```
function HasGlyph: Boolean;
```

C++

```
__fastcall Boolean HasGlyph();
```

Description

Use HasGlyph to determine whether the action associated with the TActionClientItem (see page 374) has a glyph or not. A glyph is a symbol that represents the action, and is often placed next to the action's caption. One common example in

applications is a small pair of scissors, which is frequently used to represent the Cut action.

4.1.6.4.7 TActionClientItem.InitiateAction Method

Updates the action associated with the action client item.

Pascal

```
procedure InitiateAction; virtual;
```

C++

```
virtual __fastcall InitiateAction();
```

Description

InitiateAction is used by the application to update the action periodically, so that the visible button or menu item accurately reflects the internal state of the program. To learn more about what happens when InitiateAction is called, see the documentation for TControl's InitiateAction method, which is a very similar method.

See Also

InitiateAction (see page 5769)

4.1.6.4.8 TActionClientItem.Refresh Method

Causes the size of the action client item's control (if any) to be recalculated.

Pascal

```
procedure Refresh; override;
```

C++

```
virtual __fastcall Refresh();
```

Description

Use Refresh to cause the size of the action client item's control, if there is one, to be recalculated. In general, the intent of Refresh methods is to cause the control to be redrawn.

See Also

CalcBounds (see page 518)

4.1.6.4.9 TActionClientItem.ResetUsageData Method

Pascal

```
procedure ResetUsageData;
```

C++

```
__fastcall ResetUsageData();
```

Description

This is ResetUsageData, a member of class TActionClientItem.

4.1.6.4.10 TActionClientItem.SetChangesAllowed Method

Pascal

```
procedure SetChangesAllowed(const Value: TChangesAllowedSet); override;
```

C++

```
virtual __fastcall SetChangesAllowed(const TChangesAllowedSet Value);
```

Description

This is SetChangesAllowed, a member of class TActionClientItem.

4.1.6.4.11 TActionClientItem.SetCollection Method

Sets the value of the Collection property.

Pascal

```
procedure SetCollection(Value: TCollection); override;
```

C++

```
virtual __fastcall SetCollection(TCollection Value);
```

Description

SetCollection is the protected write implementation of the Collection property.

Value specifies the collection to which this item should be added.

SetCollection removes this item from the collection to which it currently belongs and adds it to the collection specified by Value.

See Also

Collection (see page 2761)

4.1.6.4.12 TActionClientItem.SetIndex Method

Moves the collection item to a specified position in the parent collection.

Pascal

```
procedure SetIndex(Value: Integer); override;
```

C++

```
virtual __fastcall SetIndex(int Value);
```

Description

SetIndex is the protected write implementation of the Index property. The Value parameter specifies the new (0-based) position for the item in its parent collection.

See Also

Index (see page 2762)

4.1.6.4.13 TActionClientItem.SetVisible Method

Pascal

```
procedure SetVisible(const Value: Boolean); override;
```

C++

```
virtual __fastcall SetVisible(const Boolean Value);
```

Description

This is SetVisible, a member of class TActionClientItem.

4.1.6.4.14 TActionClientItem.Unused Method

Determines whether a TActionClientItem (see page 374) is unused and should be hidden on an action band.

Pascal

```
function Unused: Boolean;
```

C++

```
__fastcall Boolean Unused();
```

Description

Unused is called internally to specify whether a TActionClientItem (see page 374) is unused and should be hidden or can be displayed on an action band.

4.1.6.5 TActionClientItem Properties

The properties of the TActionClientItem class are listed here.

Public Properties

	Name	Description
R	ActionClients (see page 383)	Returns the TActionClients (see page 400) object that contains the TActionClientItem (see page 374).
	ActionLink (see page 383)	Refers to the action client item's action link.
	CheckUnused (see page 384)	Indicates whether the TActionClientItem (see page 374) object should be checked to see if it is unused and, if so, hidden from view.
	Control (see page 384)	Specifies the action band control that is rendered on an action band for the TActionClientItem (see page 374) object.
R	ParentItem (see page 386)	Refers to the parent item of the action client item.
R	Separator (see page 386)	Returns whether the TActionClientItem (see page 374) is rendered as a separator control on an action band.
R	ShortCutText (see page 387)	Contains the text label (like "Ctrl+S") for the shortcut.

Published Properties

	Name	Description
	Action (see page 382)	Specifies the action category or action that the TActionClientItem (see page 374) represents.
	Caption (see page 384)	Specifies the caption the action manager uses when the TActionClientItem (see page 374) is rendered on an action band.
	HelpContext (see page 385)	This is HelpContext, a member of class TActionClientItem.
	ImageIndex (see page 385)	Contains the ImageIndex property for the client action item.
	LastSession (see page 385)	Specifies the date an action client was last used.
	MergeIndex (see page 386)	Used to help merge action lists.
	ShortCut (see page 387)	Contains the ShortCut property for the action client item.
	ShowCaption (see page 387)	Determines whether the caption for the TActionClientItem (see page 374) is drawn.
	ShowGlyph (see page 388)	Indicates whether an image should be shown for the action.
	ShowShortCut (see page 388)	Determines whether the underbar character underneath the shortcut character is displayed.
	UsageCount (see page 388)	Specifies the total number of times an item has been used.

4.1.6.5.1 TActionClientItem.Action Property

Specifies the action category or action that the TActionClientItem (see page 374) represents.

Pascal

```
property Action: TContainedAction;
```

C++

```
__property TContainedAction Action;
```

Description

When an action category or action is added to an action band, the specified action is automatically assigned to the Action property of the TActionClientItem (see page 374) object by the action manager.

Note: If the TActionClientItem (see page 374) represents an action band separator control, the Action property is nil (Delphi) or NULL (C++).

See Also

Caption (see page 384)

4.1.6.5.2 TActionClientItem.ActionClients Property

Returns the TActionClients (see page 400) object that contains the TActionClientItem (see page 374).

Pascal

```
property ActionClients: TActionClients;
```

C++

```
__property TActionClients ActionClients;
```

Description

The ActionClients property is used internally to return the TActionClients (see page 400) object that contains the TActionClientItem (see page 374).

See Also

TActionClients (see page 400)

4.1.6.5.3 TActionClientItem.ActionLink Property

Refers to the action client item's action link.

Pascal

```
property ActionLink: TActionClientLink;
```

C++

```
__property TActionClientLink ActionLink;
```

Description

Objects do not generally deal with actions directly. Instead, there is an action link object (subclassed from TActionClientLink (see page 389) and, less directly, TActionLink) which is responsible for managing the connection between the action and the object. ActionLink is a reference to that object. For more information about action links, see the help topic for TActionLink referenced in the "See also" field above.

See Also

TActionLink (see page 265)

4.1.6.5.4 TActionClientItem.Caption Property

Specifies the caption the action manager uses when the TActionClientItem (see page 374) is rendered on an action band.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the caption that is displayed for the UI element that is rendered on an action band. By default, the Caption is blank and the action manager uses the caption of the underlying action that the TActionClientItem (see page 374) represents. Changing the Caption property of the TActionClientItem (see page 374) does not change the caption of the underlying action. If an action band contains a TCustomizeBandAction standard action, the user of the application can change this property. Because the caption can be changed at runtime, the default caption that will be used, should be set in the action itself and not in the TActionClientItem (see page 374) object.

Note: Setting the Caption property to a dash character causes the TActionClientItem (see page 374) to be rendered as a separator control, such as a menu item separator, by the action band.

4.1.6.5.5 TActionClientItem.CheckUnused Property

Indicates whether the TActionClientItem (see page 374) object should be checked to see if it is unused and, if so, hidden from view.

Pascal

```
property CheckUnused: Boolean;
```

C++

```
__property Boolean CheckUnused;
```

Description

Unused (see page 382) contents of ActionBands can be hidden from the user. Use CheckUnused to determine whether a particular action client item goes through this kind of checking. The default value is true, which means that the item can be hidden. Note that CheckUnused is merely an indicator, not a control. If you want the object to be visible regardless of how much it is used, do not change CheckUnused. Instead, set UsageCount (see page 388) to -1.

See Also

LastSession (see page 385)

PrioritySchedule (see page 555)

SessionCount (see page 358)

UsageCount (see page 388)

4.1.6.5.6 TActionClientItem.Control Property

Specifies the action band control that is rendered on an action band for the TActionClientItem (see page 374) object.

Pascal

```
property Control: TCustomActionControl;
```

C++

```
__property TCustomActionControl Control;
```

Description

The Control property is used internally to specify the specific action band control that is rendered on an action band for the TActionClientItem (see page 374).

4.1.6.5.7 TActionClientItem.HelpContext Property

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

This is HelpContext, a member of class TActionClientItem.

4.1.6.5.8 TActionClientItem.ImageIndex Property

Contains the ImageIndex property for the client action item.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action manager that contains the action client item. It identifies an image that is associated with controls and menu items that is drawn when this client action item is rendered on an action band by the action manager.

See Also

Images (see page 325)

4.1.6.5.9 TActionClientItem.LastSession Property

Specifies the date an action client was last used.

Pascal

```
property LastSession: Integer;
```

C++

```
__property int LastSession;
```

Description

LastSession is used internally to specify the session number that the TActionClientItem (see page 374) last used in the application. The action manager uses LastSession to in its calculations to determine whether or not to hide or display the TActionClientItem (see page 374) on an action band.

See Also

PrioritySchedule (see page 555)

UsageCount (see page 388)

SessionCount (see page 358)

4.1.6.5.10 TActionClientItem.MergeIndex Property

Used to help merge action lists.

Pascal

```
property MergeIndex: TMergeRange;
```

C++

```
__property TMergeRange MergeIndex;
```

Description

MergeIndex is intended to help merge existing lists of action items with those of the TActionManager (see page 432) object without redundancy. At present, this feature is not yet fully implemented, but it may become useful in a future release.

4.1.6.5.11 TActionClientItem.ParentItem Property

Refers to the parent item of the action client item.

Pascal

```
property ParentItem: TActionClient;
```

C++

```
__property TActionClient ParentItem;
```

Description

Use ParentItem to refer to the parent item of the TActionClientItem (see page 374) object.

4.1.6.5.12 TActionClientItem.Separator Property

Returns whether the TActionClientItem (see page 374) is rendered as a separator control on an action band.

Pascal

```
property Separator: Boolean;
```

C++

```
__property Boolean Separator;
```

Description

If the Caption (see page 384) property is set to a dash character, Separator is automatically set to true . Separator is called internally by the action manager to determine whether the TActionClientItem (see page 374) is rendered as a separator control on an action band.

See Also

Caption (see page 384)

4.1.6.5.13 TActionClientItem.ShortCut Property

Contains the ShortCut property for the action client item.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

Use ShortCut to specify the ShortCut that is displayed in the caption for the UI element that is rendered on an action band. By default, the ShortCut is blank and the action manager uses the ShortCut of the underlying action that the TActionClientItem (see page 374) represents. Changing the ShortCut property of the TActionClientItem (see page 374) does not change the ShortCut of the underlying action. If an action band contains a TCustomizeBandAction standard action, the user of the application can change this property. Because the shortcut key can be changed at runtime, the default shortcut key that will be used, should be set in the action itself and not in the TActionClientItem (see page 374) object.

See Also

ShortCut (see page 314)

PersistentHotKeys (see page 489)

4.1.6.5.14 TActionClientItem.ShortCutText Property

Contains the text label (like "Ctrl+S") for the shortcut.

Pascal

```
property ShortCutText: string;
```

C++

```
__property AnsiString ShortCutText;
```

Description

ShortCutText contains a text description of the shortcut is activated. For example, the ShortCutText property for a save action might read "Ctrl+S". Note that this property is read-only; it is set automatically elsewhere.

See Also

ShortCut (see page 387)

4.1.6.5.15 TActionClientItem.ShowCaption Property

Determines whether the caption for the TActionClientItem (see page 374) is drawn.

Pascal

```
property ShowCaption: Boolean;
```

C++

```
__property Boolean ShowCaption;
```

Description

ShowCaption determines whether the caption of the TActionClient (see page 364) is displayed on an action band's UI element. Set ShowCaption to false to hide the item's caption when the item is rendered on an action band. Note that the CaptionOptions

property of TActionClients (see page 400) objects takes precedence over ShowCaption. See the documentation for TActionClients (see page 400)' CaptionOptions for more details.

See Also

Caption (see page 384)

CaptionOptions (see page 407)

4.1.6.5.16 TActionClientItem.ShowGlyph Property

Indicates whether an image should be shown for the action.

Pascal

```
property ShowGlyph: Boolean;
```

C++

```
__property Boolean ShowGlyph;
```

Description

An item on an action band may have an image associated with it. ShowGlyph determines whether or not that image should be shown when the item is drawn.

See Also

Images (see page 325)

4.1.6.5.17 TActionClientItem.ShowShortCut Property

Determines whether the underbar character underneath the shortcut character is displayed.

Pascal

```
property ShowShortCut: Boolean;
```

C++

```
__property Boolean ShowShortCut;
```

Description

ShowShortCut is set and used internally to determine whether the underbar character underneath the shortcut character is displayed. The value of ShowShortCut is determined by the value set in the PersistentHotKeys property of the action band that will render the TActionClientItem (see page 374).

See Also

PersistentHotKeys (see page 489)

4.1.6.5.18 TActionClientItem.UsageCount Property

Specifies the total number of times an item has been used.

Pascal

```
property UsageCount: Integer;
```

C++

```
__property int UsageCount;
```

Description

UsageCount is used internally to specify the total number of times that the TActionClientItem (see page 374) has been used in the application. The action manager maintains this value and uses UsageCount to in its calculations to determine whether or not to hide or display the TActionClientItem (see page 374) on an action band.

See Also

LastSession (see page 385)

PrioritySchedule (see page 555)

SessionCount (see page 358)

CheckUnused (see page 384)

4.1.7 TActionClientLink Class

Class Hierarchy

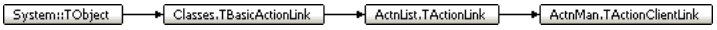

Pascal

```
TActionClientLink = class(TActionLink);
```

C++

```
class TActionClientLink : public TActionLink;
```

File

ActnMan

Description

This is class ActnMan.TActionClientLink.

4.1.7.1 TActionClientLink Members

The following tables list the members exposed by TActionClientLink.

Protected Fields

	Name	Description
	FAction (see page 2722)	This is FAction, a member of class TBasicActionLink.
	FClient (see page 391)	This is FClient, a member of class TActionClientLink.

Protected Methods

	Name	Description
	AssignClient (see page 392)	This is AssignClient, a member of class TActionClientLink.
	Change (see page 2723)	This is Change, a member of class TBasicActionLink.
	DoShowHint (see page 392)	This is DoShowHint, a member of class TActionClientLink.
	IsCaptionLinked (see page 392)	Indicates whether the Caption properties for the action and client are linked.
	IsCheckedLinked (see page 269)	Indicates whether the Checked properties for the action and client are linked.
	IsEnabledLinked (see page 393)	Indicates whether the Enabled properties for the action and client are linked.
	IsGroupIndexLinked (see page 270)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsHelpContextLinked (see page 393)	Indicates whether the HelpContext properties for the action and client controls are linked.

	IsHelpLinked (see page 271)	Indicates whether the help properties for the action and client are linked.
	IsHintLinked (see page 394)	Indicates whether the Hint properties for the action and client are linked.
	IsImageIndexLinked (see page 394)	Indicates whether the image index for the action and the client are linked.
	IsOnExecuteLinked (see page 395)	This is IsOnExecuteLinked, a member of class TActionClientLink.
	IsShortCutLinked (see page 395)	Indicates whether the ShortCut properties for the action and client are linked.
	IsVisibleLinked (see page 396)	Indicates whether the Visible properties for the action and client are linked.
	SetAction (see page 2724)	This is SetAction, a member of class TBasicActionLink.
	SetAutoCheck (see page 274)	Introduces an interface for setting the AutoCheck property of a client.
	SetCaption (see page 396)	Introduces an interface for setting the caption on a client.
	SetChecked (see page 397)	Introduces an interface for setting the checked state of a client.
	SetEnabled (see page 397)	Introduces an interface for setting the enabled state of a client.
	SetGroupIndex (see page 275)	Introduces an interface for setting the GroupIndex property of a client.
	SetHelpContext (see page 398)	Introduces an interface for setting the help context of a client.
	SetHelpKeyword (see page 276)	Introduces an interface for setting the help keyword of a client.
	SetHelpType (see page 277)	Introduces an interface for setting the HelpType property of a client.
	SetHint (see page 398)	Introduces an interface for setting the hint for a client.
	SetImageIndex (see page 399)	Introduces an interface for setting the image index for a client.
	SetOnExecute (see page 399)	This is SetOnExecute, a member of class TActionClientLink.
	SetShortCut (see page 399)	Introduces an interface for setting the shortcut for a client.
	SetVisible (see page 400)	Introduces an interface for setting the visible state of a client.

Public Events

	Name	Description
	OnChange (see page 2725)	Occurs when an OnChange event is triggered on the action.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2721)	Instantiates and initializes a TBasicActionLink (see page 2720) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2722)	Destroys an instance of an action link object.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 2723)	Invokes the action's Execute method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Update (see page 2724)	Invokes the action's Update method.

Public Properties

	Name	Description
	Action (see page 2725)	Indicates which action the link connects to its client.

4.1.7.2 TActionClientLink Fields

The fields of the TActionClientLink class are listed here.

Protected Fields

	Name	Description
	FClient (see page 391)	This is FClient, a member of class TActionClientLink.

4.1.7.2.1 TActionClientLink.FClient Field

Pascal

```
FClient: TActionClientItem;
```

C++

```
TActionClientItem FClient;
```

Description

This is FClient, a member of class TActionClientLink.

4.1.7.3 TActionClientLink Methods

The methods of the TActionClientLink class are listed here.

Protected Methods

	Name	Description
V	AssignClient (see page 392)	This is AssignClient, a member of class TActionClientLink.
V	DoShowHint (see page 392)	This is DoShowHint, a member of class TActionClientLink.
V	IsCaptionLinked (see page 392)	Indicates whether the Caption properties for the action and client are linked.
V	IsEnabledLinked (see page 393)	Indicates whether the Enabled properties for the action and client are linked.
V	IsHelpContextLinked (see page 393)	Indicates whether the HelpContext properties for the action and client controls are linked.
V	IsHintLinked (see page 394)	Indicates whether the Hint properties for the action and client are linked.
V	IsImageIndexLinked (see page 394)	Indicates whether the image index for the action and the client are linked.
V	IsOnExecuteLinked (see page 395)	This is IsOnExecuteLinked, a member of class TActionClientLink.
V	IsShortCutLinked (see page 395)	Indicates whether the ShortCut properties for the action and client are linked.
V	IsVisibleLinked (see page 396)	Indicates whether the Visible properties for the action and client are linked.
V	SetCaption (see page 396)	Introduces an interface for setting the caption on a client.
V	SetChecked (see page 397)	Introduces an interface for setting the checked state of a client.
V	SetEnabled (see page 397)	Introduces an interface for setting the enabled state of a client.
V	SetHelpContext (see page 398)	Introduces an interface for setting the help context of a client.

	SetHint (see page 398)	Introduces an interface for setting the hint for a client.
	SetImageIndex (see page 399)	Introduces an interface for setting the image index for a client.
	SetOnExecute (see page 399)	This is SetOnExecute, a member of class TActionClientLink.
	SetShortCut (see page 399)	Introduces an interface for setting the shortcut for a client.
	SetVisible (see page 400)	Introduces an interface for setting the visible state of a client.

4.1.7.3.1 TActionClientLink.AssignClient Method

Pascal

```
procedure AssignClient(AClient: TObject); override;
```

C++

```
virtual __fastcall AssignClient(TObject * AClient);
```

Description

This is AssignClient, a member of class TActionClientLink.

4.1.7.3.2 TActionClientLink.DoShowHint Method

Pascal

```
function DoShowHint(var HintStr: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoShowHint(AnsiString HintStr);
```

Description

This is DoShowHint, a member of class TActionClientLink.

4.1.7.3.3 TActionClientLink.IsCaptionLinked Method

Indicates whether the Caption properties for the action and client are linked.

Pascal

```
function IsCaptionLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCaptionLinked();
```

Description

TActionLink descendants call IsCaptionLinked to determine whether they need to propagate changes to the Caption property of the action so that they are reflected by the client object. When IsCaptionLinked returns true, the action controls the Caption property: Changes to the Caption property of the action are reflected by changes to the corresponding property of the client object. When IsCaptionLinked returns false, the client object does not reflect the Caption property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink, IsCaptionLinked merely checks that the Action property specifies a descendant of TCustomAction (that is, it is an action that has a Caption property). Descendant classes override IsCaptionLinked to determine whether the Caption property is linked with the appropriate client property.

See Also

Caption (see page 308)

Caption (see page 5806)

SetCaption (see page 274)

Action (see page 2725)

4.1.7.3.4 TActionClientLink.IsEnabledLinked Method

Indicates whether the Enabled properties for the action and client are linked.

Pascal

```
function IsEnabledLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsEnabledLinked();
```

Description

TActionLink descendants call IsEnabledLinked to determine whether they need to propagate changes to the Enabled property of the action so that they are reflected by the client object. When IsEnabledLinked returns true, the action controls the Enabled property: Changes to the Enabled property of the action are reflected by changes to the corresponding property of the client object. When IsEnabledLinked returns false, the client object does not reflect the Enabled property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink, IsEnabledLinked merely checks that the Action property specifies a descendant of TCustomAction (that is, it is an action that has an Enabled property). Descendant classes override IsEnabledLinked to determine whether the Enabled property is linked with the appropriate client property.

See Also

Enabled (see page 310)

Enabled (see page 5821)

SetEnabled (see page 275)

Action (see page 2725)

4.1.7.3.5 TActionClientLink.IsHelpContextLinked Method

Indicates whether the HelpContext properties for the action and client controls are linked.

Pascal

```
function IsHelpContextLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsHelpContextLinked();
```

Description

TActionLink descendants can call IsHelpContextLinked to determine whether they need to propagate changes to the HelpContext property of the action so that they are reflected by the client object. When IsHelpContextLinked returns true, the action controls the HelpContext property: Changes to the HelpContext property of the action are reflected in the client object. When IsHelpContextLinked returns false, the client object is not influenced by the HelpContext property of its action.

As implemented in TActionLink, IsHelpContextLinked merely checks that the Action property specifies a descendant of TCustomAction, and if so, returns true.

See Also

HelpContext (see page 311)

SetHelpContext (see page 276)

Action (see page 2725)

IsHelpLinked (see page 271)

4.1.7.3.6 TActionClientLink.IsHintLinked Method

Indicates whether the Hint properties for the action and client are linked.

Pascal

```
function IsHintLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsHintLinked();
```

Description

TActionLink descendants call IsHintLinked to determine whether they need to propagate changes to the Hint property of the action so that they are reflected by the client object. When IsHintLinked returns true, the action controls the Hint property: Changes to the Hint property of the action are reflected by changes to the corresponding property of the client object. When IsHintLinked returns false, the client object does not reflect the Hint property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink, IsHintLinked merely checks that the Action property specifies a descendant of TCustomAction (that is, it is an action that has a Hint property). Descendant classes override IsHintLinked to determine whether the Hint property is linked with the appropriate client property.

See Also

Hint (see page 312)

Hint (see page 5827)

Hint (see page 16148)

SetHint (see page 278)

Action (see page 2725)

4.1.7.3.7 TActionClientLink.IsImageIndexLinked Method

Indicates whether the image index for the action and the client are linked.

Pascal

```
function IsImageIndexLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsImageIndexLinked();
```

Description

TActionLink descendants call IsImageIndexLinked to determine whether they need to propagate changes to the ImageIndex property of the action so that they are reflected by the client object. When IsImageIndexLinked returns true, the action controls the ImageIndex property: Changes to the ImageIndex property of the action are reflected by changes to the corresponding property of the client object. When IsImageIndexLinked returns false, the client object does not reflect the ImageIndex property

of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink, IsImageIndexLinked merely checks that the Action property specifies a descendant of TCustomAction (that is, it is an action that has an ImageIndex property). Descendant classes override IsImageIndexLinked to determine whether the ImageIndex property is linked with the appropriate client property.

See Also

ImageIndex (see page 313)

ImageIndex (see page 16148)

SetImageIndex (see page 278)

Action (see page 2725)

4.1.7.3.8 TActionClientLink.IsOnExecuteLinked Method

Pascal

```
function IsOnExecuteLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsOnExecuteLinked();
```

Description

This is IsOnExecuteLinked, a member of class TActionClientLink.

4.1.7.3.9 TActionClientLink.IsShortCutLinked Method

Indicates whether the ShortCut properties for the action and client are linked.

Pascal

```
function IsShortCutLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsShortCutLinked();
```

Description

TActionLink descendants call IsShortCutLinked to determine whether they need to propagate changes to the ShortCut property of the action so that they are reflected by the client object. When IsShortCutLinked returns true, the action controls the ShortCut property: Changes to the ShortCut property of the action are reflected by changes to the corresponding property of the client object. When IsShortCutLinked returns false, the client object does not reflect the ShortCut property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink, IsShortCutLinked merely checks that the Action property specifies a descendant of TCustomAction (that is, it is an action that has a ShortCut property). Descendant classes override IsShortCutLinked to determine whether the ShortCut property is linked with the appropriate client property.

See Also

ShortCut (see page 314)

ShortCut (see page 16152)

SetShortCut (see page 279)

Action (see page 2725)

4.1.7.3.10 TActionClientLink.IsVisibleLinked Method

Indicates whether the Visible properties for the action and client are linked.

Pascal

```
function IsVisibleLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsVisibleLinked();
```

Description

TActionLink descendants call IsVisibleLinked to determine whether they need to propagate changes to the Visible property of the action so that they are reflected by the client object. When IsVisibleLinked returns true, the action controls the Visible property: Changes to the Visible property of the action are reflected by changes to the corresponding property of the client object. When IsVisibleLinked returns false, the client object does not reflect the Visible property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink, IsVisibleLinked merely checks that the Action property specifies a descendant of TCustomAction (that is, it is an action that has a Visible property). Descendant classes override IsVisibleLinked to determine whether the Visible property is linked with the appropriate client property.

See Also

Visible (see page 314)

Visible (see page 5847)

SetShortCut (see page 279)

Action (see page 2725)

4.1.7.3.11 TActionClientLink.SetCaption Method

Introduces an interface for setting the caption on a client.

Pascal

```
procedure SetCaption(const Value: string); override;
```

C++

```
virtual __fastcall SetCaption(const AnsiString Value);
```

Description

The associated action calls SetCaption when its Caption property changes so that the action link can propagate the new Caption value to the client object.

Value specifies the string to which the caption should be set.

As implemented in TActionLink, SetCaption does nothing. Descendant classes override SetCaption to set the client property that corresponds to the action's Caption property if the IsCaptionLinked method returns true.

See Also

Caption (see page 308)

Caption (see page 5806)

IsCaptionLinked (see page 268)

Action (see page 2725)

4.1.7.3.12 TActionClientLink.SetChecked Method

Introduces an interface for setting the checked state of a client.

Pascal

```
procedure SetChecked(Value: Boolean); override;
```

C++

```
virtual __fastcall SetChecked(Boolean Value);
```

Description

The associated action calls SetChecked when its Checked property changes so that the action link can propagate the new Checked value to the client object.

Value specifies the new value of the Checked property.

As implemented in TActionLink, SetChecked does nothing. Descendant classes override SetChecked to set the client property that corresponds to the action's Checked property if the IsCheckedLinked method returns true.

See Also

Checked (see page 308)

Action (see page 2725)

IsCheckedLinked (see page 269)

4.1.7.3.13 TActionClientLink.SetEnabled Method

Introduces an interface for setting the enabled state of a client.

Pascal

```
procedure SetEnabled(Value: Boolean); override;
```

C++

```
virtual __fastcall SetEnabled(Boolean Value);
```

Description

The associated action calls SetEnabled when its Enabled property changes so that the action link can propagate the new Enabled value to the client object.

Value specifies the new value of the Enabled property.

As implemented in TActionLink, SetEnabled does nothing. Descendant classes override SetEnabled to set the client property that corresponds to the action's Enabled property if the IsEnabledLinked method returns true.

See Also

Enabled (see page 310)

Enabled (see page 5821)

Action (see page 2725)

IsEnabledLinked (see page 269)

4.1.7.3.14 TActionClientLink.SetHelpContext Method

Introduces an interface for setting the help context of a client.

Pascal

```
procedure SetHelpContext(Value: THelpContext); override;
```

C++

```
virtual __fastcall SetHelpContext(THelpContext Value);
```

Description

The associated action calls SetHelpContext when its HelpContext property changes so that the action link can propagate the new HelpContext value to the client object.

Value specifies the new value of the HelpContext property.

As implemented in TActionLink, SetHelpContext does nothing. Descendant classes override SetHelpContext to set the client property that corresponds to the action's HelpContext property if the IsHelpLinked method returns true.

See Also

HelpContext (see page 311)

Action (see page 2725)

IsHelpLinked (see page 271)

4.1.7.3.15 TActionClientLink.SetHint Method

Introduces an interface for setting the hint for a client.

Pascal

```
procedure SetHint(const Value: string); override;
```

C++

```
virtual __fastcall SetHint(const AnsiString Value);
```

Description

The associated action calls SetHint when its Hint property changes so that the action link can propagate the new Hint value to the client object.

Value specifies the new value of the Hint property.

As implemented in TActionLink, SetHint does nothing. Descendant classes override SetHint to set the client property that corresponds to the action's Hint property if the IsHintLinked method returns true.

See Also

Hint (see page 312)

Hint (see page 5827)

Hint (see page 16148)

Action (see page 2725)

IsHintLinked (see page 271)

4.1.7.3.16 TActionClientLink.SetImageIndex Method

Introduces an interface for setting the image index for a client.

Pascal

```
procedure SetImageIndex(Value: Integer); override;
```

C++

```
virtual __fastcall SetImageIndex(int Value);
```

Description

The associated action calls SetImageIndex when its ImageIndex property changes so that the action link can propagate the new ImageIndex value to the client object.

Value specifies the new value of the ImageIndex property.

As implemented in TActionLink, SetImageIndex does nothing. Descendant classes override SetImageIndex to set the client property that corresponds to the action's ImageIndex property if the IsImageIndexLinked method returns true.

See Also

ImageIndex (see page 313)

ImageIndex (see page 16148)

Action (see page 2725)

IsImageIndexLinked (see page 272)

4.1.7.3.17 TActionClientLink.SetOnExecute Method

Pascal

```
procedure SetOnExecute(Value: TNotifyEvent); override;
```

C++

```
virtual __fastcall SetOnExecute(TNotifyEvent Value);
```

Description

This is SetOnExecute, a member of class TActionClientLink.

4.1.7.3.18 TActionClientLink.SetShortCut Method

Introduces an interface for setting the shortcut for a client.

Pascal

```
procedure SetShortCut(Value: TShortCut); override;
```

C++

```
virtual __fastcall SetShortCut(TShortCut Value);
```

Description

The associated action calls SetShortCut when its ShortCut property changes so that the action link can propagate the new ShortCut value to the client object.

Value specifies the new value of the ShortCut property.

As implemented in TActionLink, SetShortCut does nothing. Descendant classes override SetShortCut to set the client property

that corresponds to the action's ShortCut property if the IsShortCutLinked method returns true.

See Also

ShortCut (see page 314)

ShortCut (see page 16152)

Action (see page 2725)

IsShortCutLinked (see page 272)

4.1.7.3.19 TActionClientLink.SetVisible Method

Introduces an interface for setting the visible state of a client.

Pascal

```
procedure SetVisible(Value: Boolean); override;
```

C++

```
virtual __fastcall SetVisible(Boolean Value);
```

Description

The associated action calls SetVisible when its Visible property changes so that the action link can propagate the new Visible value to the client object.

Value specifies the new value of the Visible property.

As implemented in TActionLink, SetVisible does nothing. Descendant classes override SetVisible to set the client property that corresponds to the action's Visible property if the IsVisibleLinked method returns true.

See Also

Visible (see page 314)

Visible (see page 5847)

Action (see page 2725)

IsVisibleLinked (see page 273)

4.1.8 TActionClients Class

TActionClients is a container for TActionClient (see page 364) objects.

Class Hierarchy

Pascal

```
TActionClients = class(TActionClientsCollection);
```

C++

```
class TActionClients : public TActionClientsCollection;
```

File

ActnMan

Description

Each TActionClients object holds a collection of TClientAction objects. The TActionClients object is used internally by the TActionManager (see page 432) component to maintain information about the action clients managed by the an action manager. Instances of the TActionClients object should never be created directly or the methods contained in TActionClients called directly.

See Also

TActionClient (see page 364)

Creating Dynamic Menus

Creating Most Recently Used Lists

Hiding Unused Items and Categories in Action Bands

4.1.8.1 TActionClients Members

The following tables list the members exposed by TActionClients.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	DoHideUnused (see page 404)	This is DoHideUnused, a member of class TActionClients.
	DoShowCaptions (see page 404)	This is DoShowCaptions, a member of class TActionClients.
	GetActionManager (see page 412)	Initializes the collection after the constructor is finished.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2915)	Returns the owner of an object.
	GetOwner (see page 2932)	Returns the owner of an object.
	InternalRethinkHotkeys (see page 412)	Adjusts the captions of child items so that every item has an accelerator and there are no duplicate accelerator keys.
V	Notify (see page 405)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
	ToggleSmallIcons (see page 405)	This is ToggleSmallIcons, a member of class TActionClients.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	ParentItem (see page 415)	Indicates the action client object that uses this collection to maintain its subitems.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 404)	Creates a new TActionClient (see page 364) instance and adds it to the Items array.
V	AfterConstruction (see page 411)	Initializes the collection after the constructor is finished.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 404)	Copies the contents of one TActionClients (see page 400) object to another.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 403)	Creates and initializes a TActionClients (see page 400) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IterateClients (see page 412)	Executes a callback for every child item in the collection and any child items of those children.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	VisibleCount (see page 405)	Returns the number of visible TActionClient (see page 364) objects.

Public Properties

	Name	Description
	ActionClients (see page 406)	Provides indexed access to the TActionClient (see page 364) objects in the collection.
R	ActionManager (see page 406)	Contains a reference to the TActionManager (see page 432) component that maintains the TActionClients (see page 400) collection.
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 2751)	Lists the items in the collection.

Published Properties

	Name	Description
	AutoHotKeys (see page 407)	Specifies whether accelerator keys are auto-generated for items that do not have them assigned.
	CaptionOptions (see page 407)	Contains options for how captions should be displayed.
	Customizable (see page 407)	Indicates whether the action clients collection is customizable.
	HideUnused (see page 408)	Determines whether unused TClientAction objects are displayed in an action band.
	SmallIcons (see page 408)	Determines whether the small icons associated with the TActionClient (see page 364) objects contained by the TClientActions object are displayed in an action band.

4.1.8.2 TActionClients.Create Constructor

Creates and initializes a TActionClients (see page 400) object.

Pascal

```
constructor Create(AOwner: TPersistent; ItemClass: TCollectionItemClass);
```

C++

```
__fastcall TActionClients(TPersistent * AOwner, TCollectionItemClass ItemClass);
```

Description

The Create method takes as a parameter the name of a TPersistent instance. Create is called automatically by the action manager when a new instance of a TActionClientItem (see page 374) is required.

4.1.8.3 TActionClients Methods

The methods of the TActionClients class are listed here.

Protected Methods

	Name	Description
	DoHideUnused (see page 404)	This is DoHideUnused, a member of class TActionClients.
	DoShowCaptions (see page 404)	This is DoShowCaptions, a member of class TActionClients.
V	Notify (see page 405)	Responds when items are added to or removed from the collection.
	ToggleSmallIcons (see page 405)	This is ToggleSmallIcons, a member of class TActionClients.

Public Methods

	Name	Description
	Add (see page 404)	Creates a new TActionClient (see page 364) instance and adds it to the Items array.
V	Assign (see page 404)	Copies the contents of one TActionClients (see page 400) object to another.
	VisibleCount (see page 405)	Returns the number of visible TActionClient (see page 364) objects.

4.1.8.3.1 TActionClients.Add Method

Creates a new TActionClient (see page 364) instance and adds it to the Items array.

Pascal

```
function Add: TActionClientItem;
```

C++

```
__fastcall TActionClientItem Add();
```

Description

Add is used internally to create a TActionClient (see page 364) object in the TActionClients (see page 400) collection. The new TActionClient (see page 364) object is placed at the end of the Items array.

See Also

Add (see page 2736)

4.1.8.3.2 TActionClients.Assign Method

Copies the contents of one TActionClients (see page 400) object to another.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

This is simply an implementation of Assign (which copies information between objects) specifically for TActionClients (see page 400) objects. See the documentation for the inherited methods for more information.

See Also

Assign (see page 2736)

Assign (see page 2930)

4.1.8.3.3 TActionClients.DoHideUnused Method

Pascal

```
procedure DoHideUnused(AClient: TActionClient);
```

C++

```
__fastcall DoHideUnused(TActionClient AClient);
```

Description

This is DoHideUnused, a member of class TActionClients.

4.1.8.3.4 TActionClients.DoShowCaptions Method

Pascal

```
procedure DoShowCaptions(AClient: TActionClient);
```

C++

```
__fastcall DoShowCaptions(TActionClient AClient);
```

Description

This is DoShowCaptions, a member of class TActionClients.

4.1.8.3.5 TActionClients.Notify Method

Responds when items are added to or removed from the collection.

Pascal

```
procedure Notify(Item: TCollectionItem; Action: TCollectionNotification); override;
```

C++

```
virtual __fastcall Notify(TCollectionItem Item, TCollectionNotification Action);
```

Description

Notify is called automatically when the items in the collection change.

Item is the item that was just added to or that is about to be removed from the collection.

Action indicates whether item was added, is about to be removed, or is about to be deleted:

As implemented in TCollection, Notify calls Added when Action is cnAdded and calls Deleting when Action is cnDeleting. TCollection ignores the cnExtracting action. Descendant classes can override Notify to modify this behavior.

See Also

Added (see page 2736)

Deleting (see page 2741)

4.1.8.3.6 TActionClients.ToggleSmallIcons Method

Pascal

```
procedure ToggleSmallIcons(Clients: TActionClients; Small: Boolean);
```

C++

```
__fastcall ToggleSmallIcons(TActionClients Clients, Boolean Small);
```

Description

This is ToggleSmallIcons, a member of class TActionClients.

4.1.8.3.7 TActionClients.VisibleCount Method

Returns the number of visible TActionClient (see page 364) objects.

Pascal

```
function VisibleCount: Integer;
```

C++

```
__fastcall int VisibleCount();
```

Description

VisibleCount is used internally to return the number of visible TActionClient (see page 364) objects that can be rendered on an

action band.

4.1.8.4 TActionClients Properties

The properties of the TActionClients class are listed here.

Public Properties

	Name	Description
	ActionClients (see page 406)	Provides indexed access to the TActionClient (see page 364) objects in the collection.
R	ActionManager (see page 406)	Contains a reference to the TActionManager (see page 432) component that maintains the TActionClients (see page 400) collection.

Published Properties

	Name	Description
	AutoHotKeys (see page 407)	Specifies whether accelerator keys are auto-generated for items that do not have them assigned.
	CaptionOptions (see page 407)	Contains options for how captions should be displayed.
	Customizable (see page 407)	Indicates whether the action clients collection is customizable.
	HideUnused (see page 408)	Determines whether unused TClientAction objects are displayed in an action band.
	SmallIcons (see page 408)	Determines whether the small icons associated with the TActionClient (see page 364) objects contained by the TClientActions object are displayed in an action band.

4.1.8.4.1 TActionClients.ActionClients Property

Provides indexed access to the TActionClient (see page 364) objects in the collection.

Pascal

```
property ActionClients [const Index: Integer]: TActionClientItem;
```

C++

```
__property TActionClientItem ActionClients[int const Index];
```

Description

The ActionClients property is used internally by the action manager to provide access to the individual TClientAction objects managed by the action manager. The TActionClients (see page 400) property should never be accessed directly.

See Also

TActionClient (see page 364)

TActionManager (see page 432)

operator_[]

4.1.8.4.2 TActionClients.ActionManager Property

Contains a reference to the TActionManager (see page 432) component that maintains the TActionClients (see page 400) collection.

Pascal

```
property ActionManager: TCustomActionManager;
```

C++

```
__property TCustomActionManager ActionManager;
```

Description

ActionManager is automatically set to the TActionManager (see page 432) component that created the TActionClients (see page 400) object.

See Also

TActionManager (see page 432)

4.1.8.4.3 TActionClients.AutoHotKeys Property

Specifies whether accelerator keys are auto-generated for items that do not have them assigned.

Pascal

```
property AutoHotKeys: Boolean;
```

C++

```
__property Boolean AutoHotKeys;
```

Description

Use AutoHotKeys to specify whether accelerator keys are auto-generated for items that do not have them assigned. If AutoHotKeys is set to True, accelerator keys are automatically assigned to each action client item that does not have one assigned.

4.1.8.4.4 TActionClients.CaptionOptions Property

Contains options for how captions should be displayed.

Pascal

```
property CaptionOptions: TCaptionOptions;
```

C++

```
__property TCaptionOptions CaptionOptions;
```

Description

CaptionOptions contains the rules for displaying captions on action client items in the collection. If CaptionOptions is set to coNone, no captions will be displayed for any item. If it is set to coSelective, a determination will be made separately for each item based on the value of its ShowCaption property. A value of coAll will cause all items to display captions. Note that this property takes precedence over the ShowCaption property of individual TActionClientItem (see page 374) objects.

See Also

ShowCaption (see page 387)

Caption (see page 432)

4.1.8.4.5 TActionClients.Customizable Property

Indicates whether the action clients collection is customizable.

Pascal

```
property Customizable: Boolean;
```

C++

```
__property Boolean Customizable;
```

Description

The default setting for Customizable is true, which indicates that the collection can be customized. Set Customizable to false if you don't want to allow users to change the items in this collection using the customize dialog. Collections that are not customizable appear grayed out in the customize dialog.

4.1.8.4.6 TActionClients.HideUnused Property

Determines whether unused TClientAction objects are displayed in an action band.

Pascal

```
property HideUnused: Boolean;
```

C++

```
__property Boolean HideUnused;
```

Description

HideUnused is used internally to determine whether unused or less frequently used TClientAction objects contained by the TActionClient (see page 364) object are displayed in an action band by the action manager component.

4.1.8.4.7 TActionClients.SmallIcons Property

Determines whether the small icons associated with the TActionClient (see page 364) objects contained by the TClientActions object are displayed in an action band.

Pascal

```
property SmallIcons: Boolean;
```

C++

```
__property Boolean SmallIcons;
```

Description

Use SmallIcons to specify whether small icons are used in place of large icons when an action band renders a UI element. Setting SmallIcons to true causes the small version of the icon to be displayed.

4.1.9 TActionClientsCollection Class

TActionClientsCollection is a collection of action clients that maintains information about the action manager that owns the collection.

Class Hierarchy

Pascal

```
TActionClientsCollection = class(TOwnedCollection);
```

C++

```
class TActionClientsCollection : public TOwnedCollection;
```

File

ActnMan

Description

TActionClientsCollection is the base class for collections that list action client objects (TActionClient (see page 364) descendants) for an action manager or for another action client. Each TActionClientsCollection descendant contains a specific type of TActionClient (see page 364) descenant:

TActionClients (see page 400) contains TActionClientItem (see page 374) objects. Each TActionClientItem (see page 374) represents an action the user can execute.

TActionBars (see page 352) contains TActionBarItem (see page 346) objects. Each TActionBarItem (see page 346) represents a set of actions that appear on a menu or tool bar.

In addition to listing its collection of TActionClient (see page 364) descendants, TActionClientsCollection maintains information about the action manager that owns the collection and lets the Object Inspector display the collection's name at design time.

See Also

TPersistent (see page 2928)

TPersistent (see page 2928)

TActionClient (see page 364)

TCustomActionManager (see page 537)

TActionClient (see page 364)

TActionClients (see page 400)

TActionBars (see page 352)

4.1.9.1 TActionClientsCollection Members

The following tables list the members exposed by TActionClientsCollection.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetActionManager (see page 412)	Initializes the collection after the constructor is finished.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2915)	Returns the owner of an object.
	GetOwner (see page 2932)	Returns the owner of an object.
	InternalRethinkHotkeys (see page 412)	Adjusts the captions of child items so that every item has an accelerator and there are no duplicate accelerator keys.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.

V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
	ActionClients (see page 413)	Lists the action clients in this collection.
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	ParentItem (see page 415)	Indicates the action client object that uses this collection to maintain its subitems.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 2736)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
V	AfterConstruction (see page 411)	Initializes the collection after the constructor is finished.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 2734)	Creates and initializes a collection.
	Create (see page 2914)	Creates and initializes a TOwnedCollection (see page 2912) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IterateClients (see page 412)	Executes a callback for every child item in the collection and any child items of those children.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ActionManager (see page 414)	Indicates the action manager that owns the TActionClientsCollection (see page 408) object.
	AutoHotKeys (see page 414)	Specifies whether accelerator keys are auto-generated for items that do not have them assigned.
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
	Customizable (see page 415)	Indicates whether the action clients collection is customizable.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 2751)	Lists the items in the collection.

4.1.9.2 TActionClientsCollection Methods

The methods of the TActionClientsCollection class are listed here.

Protected Methods

	Name	Description
	GetActionManager (see page 412)	Initializes the collection after the constructor is finished.
	InternalRethinkHotkeys (see page 412)	Adjusts the captions of child items so that every item has an accelerator and there are no duplicate accelerator keys.

Public Methods

	Name	Description
V	AfterConstruction (see page 411)	Initializes the collection after the constructor is finished.
	IterateClients (see page 412)	Executes a callback for every child item in the collection and any child items of those children.

4.1.9.2.1 TActionClientsCollection.AfterConstruction Method

Initializes the collection after the constructor is finished.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called immediately after the last constructor has finished executing. TActionClientsCollection (see page 408) overrides this method to perform some final initializations.

See Also

AfterConstruction (see page 22289)

Customizable (see page 415)

4.1.9.2.2 TActionClientsCollection.GetActionManager Method

Initializes the collection after the constructor is finished.

Pascal

```
function GetActionManager: TCustomActionManager;
```

C++

```
__fastcall TCustomActionManager GetActionManager();
```

Description

GetActionManager is the protected read implementation of the ActionManager (see page 414) property.

This method returns the value of the Owner property, after casting it to a TCustomActionManager (see page 537) object.

See Also

ActionManager (see page 414)

Owner (see page 2807)

4.1.9.2.3 TActionClientsCollection.InternalRethinkHotkeys Method

Adjusts the captions of child items so that every item has an accelerator and there are no duplicate accelerator keys.

Pascal

```
function InternalRethinkHotkeys(ForceRethink: Boolean): Boolean;
```

C++

```
__fastcall Boolean InternalRethinkHotkeys(Boolean ForceRethink);
```

Description

TActionClientsCollection (see page 408) calls InternalRethinkHotkeys internally to assign accelerator keys to its child items. This method tries to ensure that all child items have accelerator keys on their captions, and that no two child items have the same accelerator key.

If any of the child items has its own set of child items, and that collection of child items has its AutoHotKeys (see page 414) property set to true, then InternalRethinkHotkeys causes the accelerator keys to be assigned for those items as well.

See Also

Caption (see page 384)

ActionClients (see page 413)

AutoHotKeys (see page 407)

4.1.9.2.4 TActionClientsCollection.IterateClients Method

Executes a callback for every child item in the collection and any child items of those children.

Pascal

```
procedure IterateClients(Clients: TActionClientsCollection; ActionProc: TActionProc);
```

C++

```
__fastcall IterateClients(TActionClientsCollection Clients, TActionProc ActionProc);
```

Description

Use IterateClients to perform some operation on every action client in a collection, including action clients that are children of the action clients in the collection. IterateClients executes the callback for each item in an action clients collection, and then, if the child item has its own children, calls itself recursively for that set of grandchildren.

Clients is the collection of action clients on which to execute the callback.

ActionProc is the callback to execute. For every action client listed by Clients (or one of its children), IterateClients calls ActionProc with the AClient parameter set to the action client. If ActionProc is nil (Delphi) or NULL (C++), IterateClients causes the action clients to repaint themselves by calling their Refresh method.

Note: IterateClients does not start by executing the items in this collection's ActionClients (see page 413) property, but rather by executing the items in the ActionClients (see page 413) property of the Clients parameter.

See Also

ActionClients (see page 413)

Refresh (see page 369)

4.1.9.3 TActionClientsCollection Properties

The properties of the TActionClientsCollection class are listed here.

Protected Properties

	Name	Description
	ActionClients (see page 413)	Lists the action clients in this collection.
	ParentItem (see page 415)	Indicates the action client object that uses this collection to maintain its subitems.

Public Properties

	Name	Description
R	ActionManager (see page 414)	Indicates the action manager that owns the TActionClientsCollection (see page 408) object.
	AutoHotKeys (see page 414)	Specifies whether accelerator keys are auto-generated for items that do not have them assigned.
	Customizable (see page 415)	Indicates whether the action clients collection is customizable.

4.1.9.3.1 TActionClientsCollection.ActionClients Property

Lists the action clients in this collection.

Pascal

```
property ActionClients [const Index: Integer]: TActionClient;
```

C++

```
__property TActionClient ActionClients[int const Index];
```

Description

ActionClients lists the child items that make up this collection. This is the same as the Items property, except that the value is of type TActionClient (see page 364) instead of TCollectionItem.

Index is the index of the child item, where 0 specifies the first item, 1 specifies the second item, and so on. The Count property provides an upper bound to the value of Index.

ActionClients is the default property for TActionClientsCollection (see page 408). This means you do not need to use the property name in Delphi code. That is, the line

```
Client := ActionManager.ActionBars.ActionClients[0];
```

can also be written

```
Client := ActionManager .ActionBars[0];
```

Note: To add action clients to this list, use the Add method. To remove action clients from this list, use the Delete method.

See Also

Items (see page 2751)

Count (see page 2749)

Add (see page 2736)

Delete (see page 2741)

operator_[]

4.1.9.3.2 TActionClientsCollection.ActionManager Property

Indicates the action manager that owns the TActionClientsCollection (see page 408) object.

Pascal

```
property ActionManager: TCustomActionManager;
```

C++

```
__property TCustomActionManager ActionManager;
```

Description

ActionManager is a reference to the action manager that owns the TActionClientsCollection (see page 408) object. This is the same as the value of the Owner property, except that the type of this property is TCustomActionManager (see page 537).

The TActionClientsCollection (see page 408) descendant represents either the set of action bands that belong to ActionManager (the value of its ActionBars property), or the set of action client items that belong to one of those action bands.

See Also

ActionBars (see page 553)

Owner (see page 2807)

4.1.9.3.3 TActionClientsCollection.AutoHotKeys Property

Specifies whether accelerator keys are auto-generated for items that do not have them assigned.

Pascal

```
property AutoHotKeys: Boolean;
```

C++

```
__property Boolean AutoHotKeys;
```

Description

Use AutoHotKeys to specify whether accelerator keys are auto-generated for items that do not have them assigned. If AutoHotKeys is set to True, accelerator keys are automatically assigned to each action client item that does not have one assigned.

4.1.9.3.4 TActionClientsCollection.Customizable Property

Indicates whether the action clients collection is customizable.

Pascal

```
property Customizable: Boolean;
```

C++

```
__property Boolean Customizable;
```

Description

The default setting for Customizable is true, which indicates that the collection can be customized. Set Customizable to false if you don't want to allow users to change the items in this collection using the customize dialog. Collections that are not customizable appear grayed out in the customize dialog.

4.1.9.3.5 TActionClientsCollection.ParentItem Property

Indicates the action client object that uses this collection to maintain its subitems.

Pascal

```
property ParentItem: TActionClient;
```

C++

```
__property TActionClient ParentItem;
```

Description

ParentItem is the TActionClient (see page 364) descendant that uses this object to implement its Items property. ParentItem corresponds to an action band, while the items in this collection correspond to the child items of that action band.

If the TActionClientsCollection (see page 408) object does not represent the child items of a TActionClient (see page 364) descendant, ParentItem is nil (Delphi) or NULL (C++). For example, on the TActionClientsCollection (see page 408) object that implements the ActionBars property of an action manager, the value of ParentItem is nil (Delphi) or NULL (C++).

See Also

ActionManager (see page 414)

Items (see page 373)

ActionBars (see page 553)

4.1.10 TActionDragBaseClass Class

TActionDragBaseClass is a base class used for managing drag operations.

Class Hierarchy

Pascal

```
TActionDragBaseClass = class(TDragObjectEx);
```

C++

```
class TActionDragBaseClass : public TDragObjectEx;
```

File

ActnMan

Description

TActionDragBaseClass is created automatically to manage drag operations in action bands.

See Also

TDragControlObject (see page 6003)

4.1.10.1 TActionDragBaseClass Members

The following tables list the members exposed by TActionDragBaseClass.

Protected Methods

	Name	Description
	Finished (see page 6035)	Responds when the mouse is released indicating the drag operation is complete.
	GetDragCursor (see page 6035)	Indicates which cursor should be displayed in response to mouse moves.
	GetDragImages (see page 6035)	Returns an image list containing the image to be displayed while dragging.
	MainWndProc (see page 6037)	This is MainWndProc, a member of class TDragObject.
	WndProc (see page 6038)	This is WndProc, a member of class TDragObject.

Public Methods

	Name	Description
	AfterConstruction (see page 6033)	Responds after the last constructor has executed.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 6034)	Copies the properties of another drag control object.
	BeforeDestruction (see page 6034)	Responds before the first destructor executes.
	BeforeDestruction (see page 6043)	Allows the application to detect that the drag object needs to be freed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetName (see page 6036)	Returns a string that identifies the drag operation.
	HideDragImage (see page 6036)	Provides the interface for a method that hides the current drag image.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Instance (see page 6037)	Returns the instance (module) handle of the EXE or DLL that created the drag object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	ShowDragImage (see page 6037)	Provides the interface for a method that hides the current drag image.

Public Properties

	Name	Description
	ActionManager (see page 417)	Contains a reference to the TActionManager (see page 432) component.
	AlwaysShowDragImages (see page 6038)	This is AlwaysShowDragImages, a member of class TDragObject.
	Cancelling (see page 6038)	Specifies whether the drag operation is being canceled.
	DragHandle (see page 6039)	Specifies the window handle of the dragged object.
	DragPos (see page 6039)	Specifies the position of the dragged object.
	DragTarget (see page 6039)	Specifies the dragged object's target.
	DragTargetPos (see page 6040)	Specifies the position of the drag object relative to the drag target.
R	Dropped (see page 6040)	Drag ended with drop.
R	MouseDeltaX (see page 6040)	Specifies the x-coordinate of the mouse position relative to the position of the dragged object's target.
R	MouseDeltaY (see page 6041)	Specifies the y-coordinate of the mouse position relative to the position of the dragged object's target.
	RightClickCancels (see page 6041)	This is RightClickCancels, a member of class TDragObject.

4.1.10.2 TActionDragBaseClass Properties

The properties of the TActionDragBaseClass class are listed here.

Public Properties

	Name	Description
	ActionManager (see page 417)	Contains a reference to the TActionManager (see page 432) component.

4.1.10.2.1 TActionDragBaseClass.ActionManager Property

Contains a reference to the TActionManager (see page 432) component.

Pascal

```
property ActionManager: TCustomActionManager;
```

C++

```
__property TCustomActionManager ActionManager;
```

Description

ActionManager is automatically set to the TActionManager (see page 432) component for managing action band drag actions.

See Also

TActionManager (see page 432)

4.1.11 TActionDragObject Class

Class Hierarchy

Pascal

```
TActionDragObject = class(TActionDragBaseClass);
```

C++

```
class TActionDragObject : public TActionDragBaseClass;
```

File

ActnMan

Description

This is class ActnMan.TActionDragObject.

4.1.11.1 TActionDragObject Members

The following tables list the members exposed by TActionDragObject.

Protected Fields

	Name	Description
	FActionClasses (see page 421)	This is FActionClasses, a member of class TActionDragObject.

Protected Methods

	Name	Description
V	Finished (see page 6035)	Responds when the mouse is released indicating the drag operation is complete.
V	GetDragCursor (see page 6035)	Indicates which cursor should be displayed in response to mouse moves.
V	GetDragImages (see page 6035)	Returns an image list containing the image to be displayed while dragging.
	MainWndProc (see page 6037)	This is MainWndProc, a member of class TDragObject.
V	WndProc (see page 6038)	This is WndProc, a member of class TDragObject.

Public Methods

	Name	Description
	AddAction (see page 421)	This is AddAction, a member of class TActionDragObject.
	AddActionClass (see page 421)	This is AddActionClass, a member of class TActionDragObject.
V	AfterConstruction (see page 6033)	Responds after the last constructor has executed.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 6034)	Copies the properties of another drag control object.
V	BeforeDestruction (see page 6034)	Responds before the first destructor executes.
V	BeforeDestruction (see page 6043)	Allows the application to detect that the drag object needs to be freed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 419)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 420)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetName (see page 6036)	Returns a string that identifies the drag operation.
V	HideDragImage (see page 6036)	Provides the interface for a method that hides the current drag image.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Instance (see page 6037)	Returns the instance (module) handle of the EXE or DLL that created the drag object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	ShowDragImage (see page 6037)	Provides the interface for a method that hides the current drag image.

Public Properties

	Name	Description
R	ActionCount (see page 422)	This is ActionCount, a member of class TActionDragObject.
	ActionManager (see page 417)	Contains a reference to the TActionManager (see page 432) component.
R	Actions (see page 422)	This is Actions, a member of class TActionDragObject.
	AlwaysShowDragImages (see page 6038)	This is AlwaysShowDragImages, a member of class TDragObject.
	Cancelling (see page 6038)	Specifies whether the drag operation is being canceled.
	DragHandle (see page 6039)	Specifies the window handle of the dragged object.
	DragPos (see page 6039)	Specifies the position of the dragged object.
	DragTarget (see page 6039)	Specifies the dragged object's target.
	DragTargetPos (see page 6040)	Specifies the position of the drag object relative to the drag target.
R	Dropped (see page 6040)	Drag ended with drop.
R	MouseDeltaX (see page 6040)	Specifies the x-coordinate of the mouse position relative to the position of the dragged object's target.
R	MouseDeltaY (see page 6041)	Specifies the y-coordinate of the mouse position relative to the position of the dragged object's target.
	RightClickCancels (see page 6041)	This is RightClickCancels, a member of class TDragObject.

4.1.11.2 TActionDragObject.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TActionDragObject();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

4.1.11.3 TActionDragObject.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TActionDragObject();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

4.1.11.4 TActionDragObject Fields

The fields of the TActionDragObject class are listed here.

Protected Fields

	Name	Description
	FActionClasses (see page 421)	This is FActionClasses, a member of class TActionDragObject.

4.1.11.4.1 TActionDragObject.FActionClasses Field

Pascal

```
FActionClasses: Boolean;
```

C++

```
Boolean FActionClasses;
```

Description

This is FActionClasses, a member of class TActionDragObject.

4.1.11.5 TActionDragObject Methods

The methods of the TActionDragObject class are listed here.

Public Methods

	Name	Description
	AddAction (see page 421)	This is AddAction, a member of class TActionDragObject.
	AddActionClass (see page 421)	This is AddActionClass, a member of class TActionDragObject.

4.1.11.5.1 TActionDragObject.AddAction Method

Pascal

```
procedure AddAction(AnAction: TContainedAction);
```

C++

```
__fastcall AddAction(TContainedAction AnAction);
```

Description

This is AddAction, a member of class TActionDragObject.

4.1.11.5.2 TActionDragObject.AddActionClass Method

Pascal

```
procedure AddActionClass(AnActionClass: TCustomActionClass);
```

C++

```
__fastcall AddActionClass(TCustomActionClass AnActionClass);
```

Description

This is AddActionClass, a member of class TActionDragObject.

4.1.11.6 TActionDragObject Properties

The properties of the TActionDragObject class are listed here.

Public Properties

	Name	Description
R	ActionCount (see page 422)	This is ActionCount, a member of class TActionDragObject.
R	Actions (see page 422)	This is Actions, a member of class TActionDragObject.

4.1.11.6.1 TActionDragObject.ActionCount Property

Pascal

```
property ActionCount: Integer;
```

C++

```
__property int ActionCount;
```

Description

This is ActionCount, a member of class TActionDragObject.

4.1.11.6.2 TActionDragObject.Actions Property

Pascal

```
property Actions [Index: Integer]: TContainedAction;
```

C++

```
__property TContainedAction Actions[int Index];
```

Description

This is Actions, a member of class TActionDragObject.

4.1.12 TActionItemDragObject Class

Class Hierarchy

Pascal

```
TActionItemDragObject = class(TActionDragBaseClass);
```

C++

```
class TActionItemDragObject : public TActionDragBaseClass;
```

File

ActnMan

Description

This is class ActnMan.TActionItemDragObject.

4.1.12.1 TActionItemDragObject Members

The following tables list the members exposed by TActionItemDragObject.

Protected Methods

	Name	Description
	Finished (see page 424)	Responds when the mouse is released indicating the drag operation is complete.
	GetDragCursor (see page 6035)	Indicates which cursor should be displayed in response to mouse moves.
	GetDragImages (see page 6035)	Returns an image list containing the image to be displayed while dragging.
	MainWndProc (see page 6037)	This is MainWndProc, a member of class TDragObject.
	WndProc (see page 6038)	This is WndProc, a member of class TDragObject.

Public Methods

	Name	Description
	AfterConstruction (see page 6033)	Responds after the last constructor has executed.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 6034)	Copies the properties of another drag control object.
	BeforeDestruction (see page 6034)	Responds before the first destructor executes.
	BeforeDestruction (see page 6043)	Allows the application to detect that the drag object needs to be freed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetName (see page 6036)	Returns a string that identifies the drag operation.
	HideDragImage (see page 6036)	Provides the interface for a method that hides the current drag image.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Instance (see page 6037)	Returns the instance (module) handle of the EXE or DLL that created the drag object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ShowDragImage (see page 6037)	Provides the interface for a method that hides the current drag image.

Public Properties

	Name	Description
	ActionManager (see page 417)	Contains a reference to the TActionManager (see page 432) component.
	AlwaysShowDragImages (see page 6038)	This is AlwaysShowDragImages, a member of class TDragObject.

	Cancelling (see page 6038)	Specifies whether the drag operation is being canceled.
	ClientItem (see page 425)	This is ClientItem, a member of class TActionItemDragObject.
	DragHandle (see page 6039)	Specifies the window handle of the dragged object.
	DragPos (see page 6039)	Specifies the position of the dragged object.
	DragTarget (see page 6039)	Specifies the dragged object's target.
	DragTargetPos (see page 6040)	Specifies the position of the drag object relative to the drag target.
R	Dropped (see page 6040)	Drag ended with drop.
R	MouseDeltaX (see page 6040)	Specifies the x-coordinate of the mouse position relative to the position of the dragged object's target.
R	MouseDeltaY (see page 6041)	Specifies the y-coordinate of the mouse position relative to the position of the dragged object's target.
	RightClickCancels (see page 6041)	This is RightClickCancels, a member of class TDragObject.

4.1.12.2 TActionItemDragObject Methods

The methods of the TActionItemDragObject class are listed here.

Protected Methods

	Name	Description
V	Finished (see page 424)	Responds when the mouse is released indicating the drag operation is complete.

4.1.12.2.1 TActionItemDragObject.Finished Method

Responds when the mouse is released indicating the drag operation is complete.

Pascal

```
procedure Finished(Target: TObject; X: Integer; Y: Integer; Accepted: Boolean); override;
```

C++

```
virtual __fastcall Finished(TObject * Target, int X, int Y, Boolean Accepted);
```

Description

Finished does nothing as implemented in TDragObject. Override this method to provide more meaningful behavior. The Target is the object that accepted the drag. If the drag operation is canceled or a drag-and-dock operation ends with the dragged object in a free-floating position, Target is nil (Delphi) or NULL (C++). X and Y indicate the position where the drag ended. Accepted is true if the Target accepted the dragged object.

Finished is overridden and implemented in TBaseDragControlObject.

See Also

Finished (see page 5723)

4.1.12.3 TActionItemDragObject Properties

The properties of the TActionItemDragObject class are listed here.

Public Properties

	Name	Description
	ClientItem (see page 425)	This is ClientItem, a member of class TActionItemDragObject.

4.1.12.3.1 TActionItemDragObject.ClientItem Property

Pascal

```
property ClientItem: TActionClientItem;
```

C++

```
__property TActionClientItem ClientItem;
```

Description

This is ClientItem, a member of class TActionItemDragObject.

4.1.13 TActionListCollection Class

TActionListCollection is a collection that maintains information about action lists and their captions.

Class Hierarchy

Pascal

```
TActionListCollection = class(TOwnedCollection);
```

C++

```
class TActionListCollection : public TOwnedCollection;
```

File

ActnMan

Description

ActionBand components, unlike other menu bar and tool bar components, do not use a single action list object. Instead, they have access to multiple lists through their action manager object. Those lists are contained in a TActionListCollection object, which is mostly just an array of action lists and their corresponding captions. Each list/caption pair is encapsulated by a TActionListItem (see page 428) object in the list.

You probably won't need to create your own instances or subclasses of TActionListCollection. Your TActionManager (see page 432) object will handle them for you.

See Also

TCollection (see page 2731)

TCustomActionManager (see page 537)

TActionManager (see page 432)

TActionListItem (see page 428)

4.1.13.1 TActionListCollection Members

The following tables list the members exposed by TActionListCollection.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2915)	Returns the owner of an object.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 2736)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 2734)	Creates and initializes a collection.
	Create (see page 2914)	Creates and initializes a TOwnedCollection (see page 2912) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.

V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ActionManager (see page 427)	Refers to the action manager associated with the collection.
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 2751)	Lists the items in the collection.
	ListItems (see page 428)	A list of action lists and their associated captions.

4.1.13.2 TActionListCollection Properties

The properties of the TActionListCollection class are listed here.

Public Properties

	Name	Description
R	ActionManager (see page 427)	Refers to the action manager associated with the collection.
	ListItems (see page 428)	A list of action lists and their associated captions.

4.1.13.2.1 TActionListCollection.ActionManager Property

Refers to the action manager associated with the collection.

Pascal

```
property ActionManager: TCustomActionManager;
```

C++

```
__property TCustomActionManager ActionManager;
```

Description

Use ActionManager to refer to the action list collection's action manager object.

See Also

TCustomActionManager (see page 537)

TActionManager (see page 432)

4.1.13.2.2 TActionListCollection.ListItems Property

A list of action lists and their associated captions.

Pascal

```
property ListItems [Index: Integer]: TActionListItem;
```

C++

```
__property TActionListItem ListItems[int Index];
```

Description

ListItems is a list of TActionListItem (see page 428) objects, which each contain an action list and the associated caption.

See Also

TActionListItem (see page 428)

operator_[]

4.1.14 TActionListItem Class

TActionListItem represents an item in a action list collection.

Class Hierarchy

Pascal

```
TActionListItem = class(TCollectionItem);
```

C++

```
class TActionListItem : public TCollectionItem;
```

File

ActnMan

Description

A TActionListItem is an element of an action list collection. It consists of two parts: an action list and a caption associated with that list. Because the caption is not part of the list itself, it is possible to have the list appear in two different places with different names.

TActionListItem objects are created through an action manager object; you do not need to create them yourself.

See Also

TActionListCollection (see page 425)

TActionManager (see page 432)

4.1.14.1 TActionListItem Members

The following tables list the members exposed by TActionListItem.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 430)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	Owner (see page 431)	This is Owner, a member of class TActionListItem.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 430)	A method for copying the contents of one TActionListItem (see page 428) to another.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2757)	Creates and initializes a TCollectionItem (see page 2755) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	ActionList (see page 431)	Specifies an action list.
	Caption (see page 432)	The name displayed in the ActionBand components containing the action list.

4.1.14.2 TActionListItem Methods

The methods of the TActionListItem class are listed here.

Protected Methods

	Name	Description
V	GetDisplayName (see page 430)	Returns the name of the collection item as it appears in the collection editor.
	Owner (see page 431)	This is Owner, a member of class TActionListItem.

Public Methods

	Name	Description
V	Assign (see page 430)	A method for copying the contents of one TActionListItem (see page 428) to another.

4.1.14.2.1 TActionListItem.Assign Method

A method for copying the contents of one TActionListItem (see page 428) to another.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use Assign to copy the contents from one TActionListItem (see page 428) to another. See the documentation for the inherited method for more information.

See Also

Assign (see page 2930)

4.1.14.2.2 TActionListItem.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName (see page 2761)

GetNamePath (see page 2759)

4.1.14.2.3 TActionListItem.Owner Method

Pascal

```
function Owner: TActionListCollection;
```

C++

```
__fastcall TActionListCollection Owner();
```

Description

This is Owner, a member of class TActionListItem.

4.1.14.3 TActionListItem Properties

The properties of the TActionListItem class are listed here.

Published Properties

	Name	Description
	ActionList (see page 431)	Specifies an action list.
	Caption (see page 432)	The name displayed in the ActionBand components containing the action list.

4.1.14.3.1 TActionListItem.ActionList Property

Specifies an action list.

Pascal

```
property ActionList: TCustomActionList;
```

C++

```
__property TCustomActionList ActionList;
```

Description

ActionList is a reference to the TActionList item's action list, which is the primary reason for the TActionListItem (see page 428)'s existence.

See Also

TActionList (see page 280)

TCustomActionList (see page 315)

4.1.14.3.2 TActionListItem.Caption Property

The name displayed in the ActionBand components containing the action list.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

The Caption string is the name displayed by the action band components when referring to ActionList (see page 431). Visibility of the caption is controlled by the CaptionOptions property of the associatedTActionClients object and the ShowCaption field of the associated TActionClientItem (see page 374) object. Refer to the documentation for these properties for more information.

See Also

CaptionOptions (see page 407)

ShowCaption (see page 387)

4.1.15 TActionManager Class

TActionManager provides a mechanism to manage and display all actions contained in an application.

Class Hierarchy

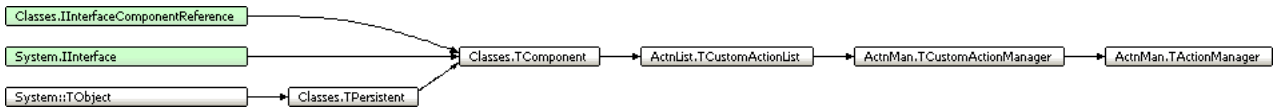

Pascal

```
TActionManager = class(TCustomActionManager);
```

C++

```
class TActionManager : public TCustomActionManager;
```

File

ActnMan

Description

The Action Manager component allows you to specify layout information for custom user-defined actions and standard actions that are contained in an application and a set of controls that can "render" this information as UI elements.

The Action Manager allows you to add any of the standard actions to your application and drag and drop any action in your application on to an action band, where it will automatically render as a UI element.

The Action Manager contains all of the layout information necessary to create menus, toolbars and Navigation bars (Outlook bars) automatically from this information. The layout information includes:

Order of appearance

Reference to an action which includes the action's Caption, ShortCut, Visible etc.

Color

Background

The Action manager stores this information in a streamable object (TActionClientItem (see page 374)) that can be written to disk, as a text or binary file. The Action Manager then uses this information to reconstruct the UI elements when the application is started again and display them on an action band component. Because the Action Manager is able to maintain state information, it is also adds the ability to allow the user of the application to customize the UI elements that are displayed on the action bands. This is easily done by dragging the standard action, TCustomizeBandAction, from the action manager editor and dropping it on an action band.

Additionally, the Action Manager maintains usage data about the items. This allows you to automatically hide infrequently used items in the UI (User Interface), making for a cleaner more effective UI.

See Also

TAction (see page 249)

TActionLink (see page 265)

TCustomActionList (see page 315)

TActionMainMenuBar (see page 585)

TActionToolBar (see page 38)

Creating Dynamic Menus

Creating Most Recently Used Lists

Creating Customizable Toolbars and Menus

Creating Toolbars and Menus

Hiding Unused Items and Categories in Action Bands

Organizing Actions for Toolbars and Menus

Setting Up Action Bands

What Is an Action?

4.1.15.1 TActionManager Members

The following tables list the members exposed by TActionManager.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAction (see page 320)	Adds an action to the list.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 320)	Generates an OnChange (see page 326) event.
V	Change (see page 545)	Generates an OnChange event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CreateActionBars (see page 545)	This is CreateActionBars, a member of class TCustomActionManager.

	DefineProperties (see page 545)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FindActionBar (see page 547)	This is FindActionBar, a member of class TCustomActionManager.
	FindActionClient (see page 547)	This is FindActionClient, a member of class TCustomActionManager.
	FindClient (see page 547)	This is FindClient, a member of class TCustomActionManager.
	GetActionBarItemClass (see page 548)	This is GetActionBarItemClass, a member of class TCustomActionManager.
	GetActionBarsClass (see page 548)	This is GetActionBarsClass, a member of class TCustomActionManager.
	GetActionClientItemClass (see page 548)	This is GetActionClientItemClass, a member of class TCustomActionManager.
	GetActionClientsClass (see page 549)	This is GetActionClientsClass, a member of class TCustomActionManager.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 321)	Calls a specified method for each child action of the action list.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetUsage (see page 549)	This is GetUsage, a member of class TCustomActionManager.
	HasLinkedActionLists (see page 549)	This is HasLinkedActionLists, a member of class TCustomActionManager.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 549)	Initializes the component after the form file has been read into memory.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	LoadMenu (see page 551)	This is LoadMenu, a member of class TCustomActionManager.
	Notification (see page 322)	Responds to notifications when objects are created or destroyed.
	Notification (see page 551)	Responds to notifications when objects are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RemoveAction (see page 323)	Removes an action from the list.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 323)	Sets the order that actions are passed to a callback by the GetChildren (see page 321) method.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetImages (see page 323)	Sets the value of the Images (see page 325) property.
	SetImages (see page 552)	Sets the value of the Images property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetState (see page 324)	This is SetState, a member of class TCustomActionList.
	SetState (see page 553)	This is SetState, a member of class TCustomActionManager.
	SetupActionBars (see page 553)	This is SetupActionBars, a member of class TCustomActionManager.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Loading (see page 555)	This is Loading, a member of class TCustomActionManager.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddAction (see page 543)	Inserts a new action into an existing actionband.
	AddCategory (see page 543)	Adds all of the actions from a category to an action band.
	AddSeparator (see page 544)	Adds a separator to an action band.
	AfterConstruction (see page 544)	Loads the layout information needed to reconstruct the action bands by the action manager from a file.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 544)	Provides a custom Assign procedure for TCustomActionManager (see page 537) objects.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 319)	Creates and initializes a TCustomActionList (see page 315) object.
	Create (see page 541)	Instantiates and initializes a TCustomActionManager (see page 537) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DeleteActionItems (see page 546)	Removes items from an ActionBars (see page 553) collection.
	DeleteItem (see page 546)	Deletes an action band item.
	Destroy (see page 319)	Disposes of the instance of an action list object.
	Destroy (see page 542)	Disposes of the instance of an action manager object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 321)	Generates an OnExecute (see page 327) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindItemByAction (see page 547)	Returns the first item linked to an action.
	FindItemByCaption (see page 548)	Returns the first item with a matching caption.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 321)	Returns a TComponent enumerator.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsShortCut (see page 322)	Evaluates an input key combination to determine whether it is a shortcut for one of the actions in the list.
	LoadFromFile (see page 550)	Loads the layout information needed to reconstruct the action bands by the action manager from a file.
	LoadFromStream (see page 550)	Loads the layout information needed to reconstruct the action bands by the action manager from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	ResetActionBar (see page 551)	Resets the action bands to their default state.
	ResetUsageData (see page 551)	Resets the usage data of the action bands.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 552)	Saves layout information needed to reconstruct the action bands by the action manager to a file.
	SaveToStream (see page 552)	Saves layout information needed to reconstruct the action bands by the action manager to a stream.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	UpdateAction (see page 324)	Generates an OnUpdate (see page 327) event.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionCount (see page 324)	Indicates the number of actions in the action list.
	Actions (see page 325)	Lists the actions maintained by the action list.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	DefaultActionBars (see page 554)	Contains a collection of the application default action bands.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 441)	Occurs when a change occurs in the action list.
	OnExecute (see page 442)	Occurs when a client object triggers an action in the list.
	OnGetBarItemClass (see page 442)	Occurs before TActionManager creates TActionBars items.
	OnGetBarsClass (see page 442)	Occurs before TActionManager creates its ActionBars collection.
	OnGetClientItemClass (see page 443)	Occurs before TActionManager creates any ActionClientItems.
	OnGetClientsClass (see page 443)	Occurs when attempting to add items to an ActionClient.
	OnStateChange (see page 443)	Occurs after an OnChange event.
	OnStyleChanged (see page 444)	Occurs after a change in the Style (see page 440) property.
	OnUpdate (see page 444)	Occurs when the application is idle so that the action list can update a specific action in the list.

Published Properties

	Name	Description
	ActionBars (see page 438)	Contains a collection of all the action bands in the application.
	FileName (see page 438)	Specifies the name of the file that the action manager component uses to read and write layout information.
	Images (see page 438)	Lists the images available for actions in the action list.
	LinkedActionLists (see page 439)	Contains a collection of action lists available from action bands and menus.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	PrioritySchedule (see page 439)	Specifies a schedule that indicates how long it takes to mark an item as unused.
	State (see page 440)	Indicates whether the actions in the action list respond when the user triggers a client object.
	Style (see page 440)	Identifies the style object that gives the application its default look and feel.
	Tag (see page 2808)	Stores an integer value as part of a component.

4.1.15.2 TActionManager Properties

The properties of the TActionManager class are listed here.

Published Properties

	Name	Description
	ActionBars (see page 438)	Contains a collection of all the action bands in the application.
	FileName (see page 438)	Specifies the name of the file that the action manager component uses to read and write layout information.
	Images (see page 438)	Lists the images available for actions in the action list.

	LinkedActionLists (see page 439)	Contains a collection of action lists available from action bands and menus.
	PrioritySchedule (see page 439)	Specifies a schedule that indicates how long it takes to mark an item as unused.
	State (see page 440)	Indicates whether the actions in the action list respond when the user triggers a client object.
	Style (see page 440)	Identifies the style object that gives the application its default look and feel.

4.1.15.2.1 TActionManager.ActionBars Property

Contains a collection of all the action bands in the application.

Pascal

```
property ActionBars: TActionBars;
```

C++

```
__property TActionBars ActionBars;
```

Description

ActionBars is used internally by the action manager component to maintain a collection of all the action bands connected to the action manager in the application. An action band is automatically added to the collection when an action or category of actions is dropped on an action band or the ActionManager property in an action band component is set in the Object Inspector.

See Also

DefaultActionBars (see page 554)

4.1.15.2.2 TActionManager.FileName Property

Specifies the name of the file that the action manager component uses to read and write layout information.

Pascal

```
property FileName: TFileName;
```

C++

```
__property TFileName FileName;
```

Description

Use FileName to specify the name of the file that the action manager will use to read and write layout information for the action bands that are connected to the action manager. The action manager uses this file to store the current state of the action bands that are connected to the action manager when the application is closed. When an application is started, the action manager uses the layout information contained in this file to reconstruct the action bands back to the state when the application last closed. All customizations made to the action bands are saved in this file.

4.1.15.2.3 TActionManager.Images Property

Lists the images available for actions in the action list.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Images is a list of bitmaps that can be displayed to the left of any action in the action list. An action is associated with an image from this image list by its ImageIndex property. This image can then appear on client controls such as menu items, tool buttons,

and so on.

See Also

Actions (see page 325)

ImageIndex (see page 313)

4.1.15.2.4 TActionManager.LinkedActionLists Property

Contains a collection of action lists available from action bands and menus.

Pascal

```
property LinkedActionLists: TActionListCollection;
```

C++

```
__property TActionListCollection LinkedActionLists;
```

Description

LinkedActionLists is a linked list of TActionListCollection objects. LinkedActionLists are simple data structures with two fields. One field is a TCustomActionList object, which is a list of possible actions. The other field is a caption, which defines the action list's description appearing on the action band or menu.

Use LinkedActionLists to merge in action lists from other sources outside the action manager. Those action lists will become accessible from action bands and menus. Application users can exploit this feature, for example, to reorganize items from a traditional main menu bar.

See Also

TActionListCollection (see page 425)

4.1.15.2.5 TActionManager.PrioritySchedule Property

Specifies a schedule that indicates how long it takes to mark an item as unused.

Pascal

```
property PrioritySchedule: TStringList;
```

C++

```
__property TStringList PrioritySchedule;
```

Description

Based on usage statistics and layout space, the action manager uses PrioritySchedule to determine when a UI element representing a TActionClientItem is hidden or displayed on an action band. (This is not the same as the TActionClientItem's visibility, as set by the TActionClientItem's Visible property.)

To determine when to display or hide a UI element that is rendered by the action band for a specific TActionClientItem, the action manager maintains a total count for each item of the total number of sessions, the session number it was used in, and the current session number. The number of sessions is defined as the number of times the application has been launched.

To determine if an item should be displayed, the program looks up the value of UsageCount in the left hand column in PrioritySchedule. If UsageCount is greater than the maximum tabulated value, that maximum is used for calculation purposes. The corresponding value in the right hand column is the number of sessions an item can be unused before it becomes hidden.

An action band will always show as many controls as it has space to show; the action band will wrap rows or columns, if necessary. TActionClientItems with their UsageCount property set to -1, will always be shown. You can use the UsageCount property to ensure that specific action band controls are always shown.

The default PrioritySchedule is as follows:

Number of sessions in which the action band item was used	Number of sessions the item will remain unhidden after its last use
0, 1	3
2	6
3	9
4, 5	12
6 – 8	17
9 – 13	23
14 – 24	29
25 or more	31

See Also

SessionCount (see page 358)

LastSession (see page 385)

UsageCount (see page 388)

4.1.15.2.6 TActionManager.State Property

Indicates whether the actions in the action list respond when the user triggers a client object.

Pascal

```
property State: TActionListState;
```

C++

```
__property TActionListState State;
```

Description

Use State to turn an action list on or off. State can have any of the following values:

Value	Meaning
asNormal	Actions perform their normal response when triggered by client objects or hotkeys.
asSuspended	The actions in the action list do not respond when client objects tell them to "fire". The Enabled property for all actions in the list is unchanged.
asSuspendedEnabled	The actions in the action list do not respond when client objects tell them to "fire". The Enabled property for all actions in the list is set to true. Used for designing ActionBars.

See Also

Execute (see page 304)

Enabled (see page 310)

4.1.15.2.7 TActionManager.Style Property

Identifies the style object that gives the application its default look and feel.

Pascal

```
property Style: TActionBarStyle;
```

C++

```
__property TActionBarStyle Style;
```

Description

Use Style to access the application's Style object. This object manages which type of controls are used on the action bands to present a particular thematic look. To change the overall appearance of action bands in your application, change the style property accordingly. Delphi includes two styles by default: Standard and XP style. You can further augment these styles by using colormap components to control the colors that render the different style of controls.

To register new styles create a descendant of TActionBarStyleEx and register it using RegisterActnBarStyle.

See Also

TActionBarStyleEx (see page 583)

TStandardColorMap (see page 2)

TXPColorMap (see page 23)

RegisterActnBarStyle (see page 837)

4.1.15.3 TActionManager Events

The events of the TActionManager class are listed here.

Published Events

	Name	Description
	OnChange (see page 441)	Occurs when a change occurs in the action list.
	OnExecute (see page 442)	Occurs when a client object triggers an action in the list.
	OnGetBarItemClass (see page 442)	Occurs before TActionManager creates TActionBars items.
	OnGetBarsClass (see page 442)	Occurs before TActionManager creates its ActionBars collection.
	OnGetClientItemClass (see page 443)	Occurs before TActionManager creates any ActionClientItems.
	OnGetClientsClass (see page 443)	Occurs when attempting to add items to an ActionClient.
	OnStateChange (see page 443)	Occurs after an OnChange event.
	OnStyleChanged (see page 444)	Occurs after a change in the Style (see page 440) property.
	OnUpdate (see page 444)	Occurs when the application is idle so that the action list can update a specific action in the list.

4.1.15.3.1 TActionManager.OnChange Event

Occurs when a change occurs in the action list.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to update your application in response to changes in the action list. Changes to the action list that trigger an OnChange event include changes to an action category or to the action list's image list.

Note: The action list automatically informs the actions in the list when an OnChange event occurs. You need only provide an

OnChange event handler for application-specific responses.

4.1.15.3.2 TActionManager.OnExecute Event

Occurs when a client object triggers an action in the list.

Pascal

```
property OnExecute: TActionEvent;
```

C++

```
__property TActionEvent OnExecute;
```

Description

Write an OnExecute event handler to respond when an action in the list fires.

When the user invokes a client object, the action list's OnExecute event is the first event to occur in response. If the OnExecute event handler sets its Handled parameter to true, the action is considered handled, and processing stops there. This blocks the execution of any other event handlers or any built-in response by a predefined action class in response to the user action.

If the OnExecute event handler does not set its Handled parameter to true, the application's OnActionExecute event occurs next. If the OnActionExecute event does not respond to the user input, the action object's OnExecute event occurs. Finally, if the action does not have an OnExecute event handler, the application locates a target object and executes any predefined response implemented by the action object.

See Also

Execute (see page 304)

OnExecute (see page 2716)

OnActionExecute (see page 13725)

4.1.15.3.3 TActionManager.OnGetBarItemClass Event

Occurs before TActionManager creates TActionBars items.

Pascal

```
property OnGetBarItemClass: TGetBarItemClassEvent;
```

C++

```
__property TGetBarItemClassEvent OnGetBarItemClass;
```

Description

Write an OnGetBarItemClass event handler to change the type of the class used for the ActionBarItem. The OnGetBarItemClass event occurs after a find action fetches the item's class, but before the class is sent to the action client. OnGetBarItemClass provides an opportunity to change the class used for the items within the TActionBars collection.

4.1.15.3.4 TActionManager.OnGetBarsClass Event

Occurs before TActionManager creates its ActionBars collection.

Pascal

```
property OnGetBarsClass: TGetBarsClassEvent;
```

C++

```
__property TGetBarsClassEvent OnGetBarsClass;
```

Description

Write an OnGetBarsClass event handler to change the type of the TActionBars collection to a descedant of TActionBars. OnGetBarsClass provides an opportunity to change the class used for the TActionBars collection. By implementing this event and returning a descendant of TActionBars, the developer can add new properties and new functionality to the collection for use within action bands.

4.1.15.3.5 TActionManager.OnGetClientItemClass Event

Occurs before TActionManager creates any ActionClientItems.

Pascal

```
property OnGetClientItemClass: TGetClientItemClassEvent;
```

C++

```
__property TGetClientItemClassEvent OnGetClientItemClass;
```

Description

Write an OnGetClientItemClass event handler to change the type of the TactionClientItem class that is created and used by the ActionManager. OnGetClientItemClass provides an opportunity to return a descendant of TactionClientItem, which has new properties, events and methods for use within action bands.

4.1.15.3.6 TActionManager.OnGetClientsClass Event

Occurs when attempting to add items to an ActionClient.

Pascal

```
property OnGetClientsClass: TGetClientsClassEvent;
```

C++

```
__property TGetClientsClassEvent OnGetClientsClass;
```

Description

Write an OnGetClientsClass event handler to change the type of the TactionClients class that is created and used by the ActionManager. OnGetClientsClass provides an opportunity to return a descendant of TActionClients which implements new properties, events and methods for use within action bands.

4.1.15.3.7 TActionManager.OnStateChange Event

Occurs after an OnChange event.

Pascal

```
property OnStateChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnStateChange;
```

Description

Write an OnStateChange event handler to respond after the OnChange event occurs.

See Also

OnChange (see page 326)

4.1.15.3.8 TActionManager.OnStyleChanged Event

Occurs after a change in the Style (see page 440) property.

Pascal

```
property OnStyleChanged: TStyleChanged;
```

C++

```
__property TStyleChanged OnStyleChanged;
```

Description

Write an OnStyleChanged event handler to respond after the Style (see page 440) property changes.

See Also

Style Property

4.1.15.3.9 TActionManager.OnUpdate Event

Occurs when the application is idle so that the action list can update a specific action in the list.

Pascal

```
property OnUpdate: TActionEvent;
```

C++

```
__property TActionEvent OnUpdate;
```

Description

Write an OnUpdate event handler to configure the properties of a specified action in the list so that it reflects current conditions.

When the application is idle, it cycles through every action in turn giving it a chance to update itself to reflect current conditions. For each action, the first event to occur in this sequence is the action list's OnUpdate event. If the OnUpdate event handler sets its Handled parameter to true, the action is considered updated, and processing stops there. This blocks the execution of any other event handlers or any built-in response by a predefined action class.

If the OnUpdate event handler does not set its Handled parameter to true, the application's OnActionUpdate event occurs next. If the OnActionUpdate event does not update the action, the action object's OnUpdate event occurs. Finally, if the action does not have an OnUpdate event handler, the application executes any predefined update method implemented by the action object.

See Also

OnUpdate (see page 2718)

OnActionUpdate (see page 13726)

4.1.16 TCategoryDragObject Class

Class Hierarchy

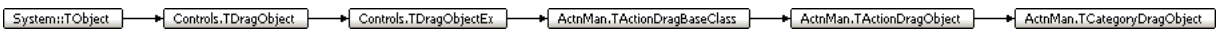

Pascal

```
TCategoryDragObject = class(TActionDragObject);
```

C++

```
class TCategoryDragObject : public TActionDragObject;
```

File

ActnMan

Description

This is class ActnMan.TCategoryDragObject.

4.1.16.1 TCategoryDragObject Members

The following tables list the members exposed by TCategoryDragObject.

Protected Fields

	Name	Description
	FActionClasses (see page 421)	This is FActionClasses, a member of class TActionDragObject.

Protected Methods

	Name	Description
V	Finished (see page 6035)	Responds when the mouse is released indicating the drag operation is complete.
V	GetDragCursor (see page 6035)	Indicates which cursor should be displayed in response to mouse moves.
V	GetDragImages (see page 6035)	Returns an image list containing the image to be displayed while dragging.
	MainWndProc (see page 6037)	This is MainWndProc, a member of class TDragObject.
V	WndProc (see page 6038)	This is WndProc, a member of class TDragObject.

Public Methods

	Name	Description
	AddAction (see page 421)	This is AddAction, a member of class TActionDragObject.
	AddActionClass (see page 421)	This is AddActionClass, a member of class TActionDragObject.
V	AfterConstruction (see page 6033)	Responds after the last constructor has executed.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 6034)	Copies the properties of another drag control object.
V	BeforeDestruction (see page 6034)	Responds before the first destructor executes.
V	BeforeDestruction (see page 6043)	Allows the application to detect that the drag object needs to be freed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 446)	This is Create, a member of class TCategoryDragObject.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 420)	Disposes of an object instance.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetName (see page 6036)	Returns a string that identifies the drag operation.
V	HideDragImage (see page 6036)	Provides the interface for a method that hides the current drag image.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Instance (see page 6037)	Returns the instance (module) handle of the EXE or DLL that created the drag object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	ShowDragImage (see page 6037)	Provides the interface for a method that hides the current drag image.

Public Properties

	Name	Description
R	ActionCount (see page 422)	This is ActionCount, a member of class TActionDragObject.
	ActionManager (see page 417)	Contains a reference to the TActionManager (see page 432) component.
R	Actions (see page 422)	This is Actions, a member of class TActionDragObject.
	AlwaysShowDragImages (see page 6038)	This is AlwaysShowDragImages, a member of class TDragObject.
	Cancelling (see page 6038)	Specifies whether the drag operation is being canceled.
R	Category (see page 447)	This is Category, a member of class TCategoryDragObject.
	DragHandle (see page 6039)	Specifies the window handle of the dragged object.
	DragPos (see page 6039)	Specifies the position of the dragged object.
	DragTarget (see page 6039)	Specifies the dragged object's target.
	DragTargetPos (see page 6040)	Specifies the position of the drag object relative to the drag target.
R	Dropped (see page 6040)	Drag ended with drop.
R	MouseDeltaX (see page 6040)	Specifies the x-coordinate of the mouse position relative to the position of the dragged object's target.
R	MouseDeltaY (see page 6041)	Specifies the y-coordinate of the mouse position relative to the position of the dragged object's target.
	RightClickCancels (see page 6041)	This is RightClickCancels, a member of class TDragObject.

4.1.16.2 TCategoryDragObject.Create Constructor

Pascal

```
constructor Create(ACategory: string);
```

C++

```
__fastcall TCategoryDragObject(AnsiString ACategory);
```

Description

This is Create, a member of class TCategoryDragObject.

4.1.16.3 TCategoryDragObject Properties

The properties of the TCategoryDragObject class are listed here.

Public Properties

	Name	Description
	Category (see page 447)	This is Category, a member of class TCategoryDragObject.

4.1.16.3.1 TCategoryDragObject.Category Property

Pascal

```
property Category: string;
```

C++

```
__property AnsiString Category;
```

Description

This is Category, a member of class TCategoryDragObject.

4.1.17 TCustomActionBar Class

TCustomActionBar is the base class for all action bands.

Class Hierarchy

Pascal

```
TCustomActionBar = class(TToolWindow);
```

C++

```
class TCustomActionBar : public TToolWindow;
```

File

ActnMan

Description

TCustomActionBar is the base class for all the action bands that the action manager component uses as rendering engines to display actions and categories of actions as UI elements.

See Also

TActionMainMenuBar (see page 585)

TActionToolBar (see page 38)

4.1.17.1 TCustomActionBar Members

The following tables list the members exposed by TCustomActionBar.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.

	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefaultColorMap (see page 460)	This is FDefaultColorMap, a member of class TCustomActionBar.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

	FSavedWrapState (see page 460)	This is FSavedWrapState, a member of class TCustomActionBar.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 462)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
V	AutoSizingChanged (see page 462)	This is AutoSizingChanged, a member of class TCustomActionBar.
	BackgroundSize (see page 462)	This is BackgroundSize, a member of class TCustomActionBar.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcInsertPosition (see page 462)	This is CalcInsertPosition, a member of class TCustomActionBar.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
V	Clear (see page 463)	This is Clear, a member of class TCustomActionBar.
	Click (see page 5743)	Respond to user click.
	CMBorderChanged (see page 23285)	This is CMBorderChanged, a member of class TToolWindow.
	CMCtl3DChanged (see page 23286)	This is CMCtl3DChanged, a member of class TToolWindow.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
V	CreateControl (see page 463)	This is CreateControl, a member of class TCustomActionBar.
V	CreateControls (see page 463)	This is CreateControls, a member of class TCustomActionBar.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 463)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 464)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 465)	This is DesignWndProc, a member of class TCustomActionBar.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoControlCreated (see page 465)	This is DoControlCreated, a member of class TCustomActionBar.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoDropActions (see page 465)	This is DoDropActions, a member of class TCustomActionBar.
	DoDropCategory (see page 465)	This is DoDropCategory, a member of class TCustomActionBar.
	DoDropItem (see page 466)	This is DoDropItem, a member of class TCustomActionBar.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetControlClass (see page 466)	This is DoGetControlClass, a member of class TCustomActionBar.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 466)	OnDragOver event dispatcher.
	DrawBackground (see page 467)	This is DrawBackground, a member of class TCustomActionBar.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindAccelItem (see page 467)	This is FindAccelItem, a member of class TCustomActionBar.
	FindItemAt (see page 468)	This is FindItemAt, a member of class TCustomActionBar.
	FindItemCaption (see page 468)	This is FindItemCaption, a member of class TCustomActionBar.
	FindLast (see page 468)	This is FindLast, a member of class TCustomActionBar.
	FindLeastUsedItem (see page 469)	This is FindLeastUsedItem, a member of class TCustomActionBar.
	FindNearestControl (see page 469)	This is FindNearestControl, a member of class TCustomActionBar.
	FindNext (see page 470)	This is FindNext, a member of class TCustomActionBar.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FindNextVisibleItem (see page 470)	This is FindNextVisibleItem, a member of class TCustomActionBar.
	FindPrevious (see page 470)	This is FindPrevious, a member of class TCustomActionBar.

	FindPreviousVisibleItem (see page 470)	This is FindPreviousVisibleItem, a member of class TCustomActionBar.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
V	GetActionControl (see page 471)	This is GetActionControl, a member of class TCustomActionBar.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
V	GetBannerWidth (see page 471)	This is GetBannerWidth, a member of class TCustomActionBar.
V	GetBarHeight (see page 471)	This is GetBarHeight, a member of class TCustomActionBar.
V	GetBarWidth (see page 471)	This is GetBarWidth, a member of class TCustomActionBar.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlClass (see page 471)	This is GetControlClass, a member of class TCustomActionBar.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDefaultColorMapClass (see page 472)	This is GetDefaultColorMapClass, a member of class TCustomActionBar.
V	GetDesignMode (see page 472)	This is GetDesignMode, a member of class TCustomActionBar.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetPopupMenuClass (see page 472)	This is GetPopupMenuClass, a member of class TCustomActionBar.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasItems (see page 472)	This is HasItems, a member of class TCustomActionBar.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
V	ItemCount (see page 473)	This is ItemCount, a member of class TCustomActionBar.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.

	Loaded (see page 473)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	NCPaint (see page 473)	This is NCPaint, a member of class TCustomActionBar.
	Notification (see page 473)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 474)	This is Paint, a member of class TCustomActionBar.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 474)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Reset (see page 475)	This is Reset, a member of class TCustomActionBar.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActionClient (see page 475)	This is SetActionClient, a member of class TCustomActionBar.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetAutoSizing (see page 475)	This is SetAutoSizing, a member of class TCustomActionBar.
	SetBiDiMode (see page 475)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetColorMap (see page 476)	This is SetColorMap, a member of class TCustomActionBar.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignMode (see page 476)	This is SetDesignMode, a member of class TCustomActionBar.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.

	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetOrientation (see page 477)	This is SetOrientation, a member of class TCustomActionBar.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetPersistentHotKeys (see page 477)	This is SetPersistentHotKeys, a member of class TCustomActionBar.
V	SetSpacing (see page 477)	This is SetSpacing, a member of class TCustomActionBar.
	SetupDefaultColorMap (see page 477)	This is SetupDefaultColorMap, a member of class TCustomActionBar.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	VisibleChanging (see page 478)	Provides the interface for a method that responds to changes in the control's visibility.
	WMNCCalcSize (see page 23286)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCPaint (see page 23286)	This is WMNCPaint, a member of class TToolWindow.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.

	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
R	HRowCount (see page 488)	Indicates the number of columns of items that the action band displays.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	Items (see page 488)	Lists the action clients that correspond to the child controls of this action band.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
R	VRowCount (see page 490)	Indicates the number of rows of items that the action band displays.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
R	Widest (see page 491)	This is Widest, a member of class TCustomActionBar.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnControlCreated (see page 491)	Occurs when a control is created on the action band.
	OnGetControlClass (see page 492)	Occurs when an item is to be placed on the action band.
	OnPaint (see page 492)	Occurs when the action band is painted.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 459)	Creates and initializes a TCustomActionBar (see page 447) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 459)	Destroys an instance of TCustomActionBar (see page 447).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
V	DragDrop (see page 466)	Generates an OnDragDrop event.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
V	FindFirst (see page 467)	Finds the first TActionClientItem (see page 374) in the Items (see page 488) list.
	FindFirstVisibleItem (see page 468)	Finds the first visible TActionClientItem (see page 374) in the Items (see page 488) list.
	FindLastVisibleItem (see page 469)	Finds the last visibleTActionClientItem in the Items (see page 488) list.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.

	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
V	RecreateControls (see page 475)	Destroys and recreates all of the action band's action client controls.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 476)	Sets the bounds based on whether the popup is expanded or not

	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Style (see page 477)	Identifies the style object that gives the application its default look and feel.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	ActionClient (see page 479)	Specifies the action client associated with this action band.
R	ActionControls (see page 479)	Lists the controls on the action band.
	ActionManager (see page 480)	Specifies the action manager component that is associated with the action band.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AllowHiding (see page 480)	Determines whether the action band can be hidden.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AutoSizing (see page 480)	Specifies whether the action band adjusts its size automatically to fit around its child controls.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 481)	Represents the drawing surface of the action band.
	Caption (see page 481)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 483)	Specifies the background color of the control.
	ColorMap (see page 484)	Indicates the colormap for individual action band components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ContextBar (see page 485)	Indicates whether items shown in this action band represent the items in a popup menu.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designable (see page 485)	Specifies whether the action band can be customized (used internally by the IDE).

	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignMode (see page 486)	Specifies whether the action band is being customized or being used in the IDE.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EdgeBorders (see page 23287)	Determines which sides of the control have visible borders.
	EdgeInner (see page 23287)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 23288)	Determines the border style for the control's outer edge.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Font (see page 486)	Controls the attributes of text written on or in the control.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HorzMargin (see page 487)	Specifies the width of the horizontal margins in pixels.
	HorzSeparator (see page 487)	Indicates whether a horizontal separator bar appears between item lines.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	Orientation (see page 488)	Specifies how the action band lays out its child items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PersistentHotKeys (see page 489)	Specifies whether keyboard accelerator shortcuts are always displayed.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	Spacing (see page 489)	Determines where the image and text appear on an action band's rendered control that represents the action client.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	VertMargin (see page 490)	Specifies the width of the vertical margins in pixels.
	VertSeparator (see page 490)	Indicates whether a vertical separator bar should appear between item lines.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

4.1.17.2 TCustomActionBar.Create Constructor

Creates and initializes a TCustomActionBar (see page 447) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomActionBar(TComponent * AOwner);
```

Description

Create generates a TCustomActionBar (see page 447) and initializes its properties.

AOwner is the component, typically the form, that becomes the value of the Owner property. The Owner of a component is responsible for freeing it and for initiating the process that loads it from or saves it to a form file.

See Also

Owner (see page 2807)

4.1.17.3 TCustomActionBar.Destroy Destructor

Destroys an instance of TCustomActionBar (see page 447).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomActionBar();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TCustomActionBar (see page 447) reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

Destroy (see page 5732)

4.1.17.4 TCustomActionBar Fields

The fields of the TCustomActionBar class are listed here.

Protected Fields

	Name	Description
	FDefaultColorMap (see page 460)	This is FDefaultColorMap, a member of class TCustomActionBar.
	FSavedWrapState (see page 460)	This is FSavedWrapState, a member of class TCustomActionBar.

4.1.17.4.1 TCustomActionBar.FDefaultColorMap Field

Pascal

```
FDefaultColorMap: TCustomActionBarColorMap;
```

C++

```
TCustomActionBarColorMap FDefaultColorMap;
```

Description

This is FDefaultColorMap, a member of class TCustomActionBar.

4.1.17.4.2 TCustomActionBar.FSavedWrapState Field

Pascal

```
FSavedWrapState: Boolean;
```

C++

```
Boolean FSavedWrapState;
```

Description

This is FSavedWrapState, a member of class TCustomActionBar.

4.1.17.5 TCustomActionBar Methods

The methods of the TCustomActionBar class are listed here.

Protected Methods

	Name	Description
	AlignControls (see page 462)	Aligns any controls for which the control is the parent within a specified area of the control.
	AutoSizingChanged (see page 462)	This is AutoSizingChanged, a member of class TCustomActionBar.
	BackgroundSize (see page 462)	This is BackgroundSize, a member of class TCustomActionBar.
	CalcInsertPosition (see page 462)	This is CalcInsertPosition, a member of class TCustomActionBar.
	Clear (see page 463)	This is Clear, a member of class TCustomActionBar.
	CreateControl (see page 463)	This is CreateControl, a member of class TCustomActionBar.
	CreateControls (see page 463)	This is CreateControls, a member of class TCustomActionBar.

	CustomAlignInsertBefore (see page 463)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 464)	Returns aligned position of custom-aligned child control.
	DesignWndProc (see page 465)	This is DesignWndProc, a member of class TCustomActionBar.
	DoControlCreated (see page 465)	This is DoControlCreated, a member of class TCustomActionBar.
	DoDropActions (see page 465)	This is DoDropActions, a member of class TCustomActionBar.
	DoDropCategory (see page 465)	This is DoDropCategory, a member of class TCustomActionBar.
	DoDropItem (see page 466)	This is DoDropItem, a member of class TCustomActionBar.
	DoGetControlClass (see page 466)	This is DoGetControlClass, a member of class TCustomActionBar.
	DragOver (see page 466)	OnDragOver event dispatcher.
	DrawBackground (see page 467)	This is DrawBackground, a member of class TCustomActionBar.
	FindAccelItem (see page 467)	This is FindAccelItem, a member of class TCustomActionBar.
	FindItemAt (see page 468)	This is FindItemAt, a member of class TCustomActionBar.
	FindItemCaption (see page 468)	This is FindItemCaption, a member of class TCustomActionBar.
	FindLast (see page 468)	This is FindLast, a member of class TCustomActionBar.
	FindLeastUsedItem (see page 469)	This is FindLeastUsedItem, a member of class TCustomActionBar.
	FindNearestControl (see page 469)	This is FindNearestControl, a member of class TCustomActionBar.
	FindNext (see page 470)	This is FindNext, a member of class TCustomActionBar.
	FindNextVisibleItem (see page 470)	This is FindNextVisibleItem, a member of class TCustomActionBar.
	FindPrevious (see page 470)	This is FindPrevious, a member of class TCustomActionBar.
	FindPreviousVisibleItem (see page 470)	This is FindPreviousVisibleItem, a member of class TCustomActionBar.
	GetActionControl (see page 471)	This is GetActionControl, a member of class TCustomActionBar.
	GetBannerWidth (see page 471)	This is GetBannerWidth, a member of class TCustomActionBar.
	GetBarHeight (see page 471)	This is GetBarHeight, a member of class TCustomActionBar.
	GetBarWidth (see page 471)	This is GetBarWidth, a member of class TCustomActionBar.
	GetControlClass (see page 471)	This is GetControlClass, a member of class TCustomActionBar.
	GetDefaultColorMapClass (see page 472)	This is GetDefaultColorMapClass, a member of class TCustomActionBar.
	GetDesignMode (see page 472)	This is GetDesignMode, a member of class TCustomActionBar.
	GetPopupMenuClass (see page 472)	This is GetPopupMenuClass, a member of class TCustomActionBar.
	HasItems (see page 472)	This is HasItems, a member of class TCustomActionBar.
	ItemCount (see page 473)	This is ItemCount, a member of class TCustomActionBar.
	Loaded (see page 473)	Initializes the control after it is loaded from a stream.
	NCPaint (see page 473)	This is NCPaint, a member of class TCustomActionBar.
	Notification (see page 473)	Responds to notifications that components are being created or destroyed.
	Paint (see page 474)	This is Paint, a member of class TCustomActionBar.
	PaintWindow (see page 474)	Renders the image of a windowed control.
	Reset (see page 475)	This is Reset, a member of class TCustomActionBar.
	SetActionClient (see page 475)	This is SetActionClient, a member of class TCustomActionBar.
	SetAutoSizing (see page 475)	This is SetAutoSizing, a member of class TCustomActionBar.
	SetBiDiMode (see page 475)	Sets the control's bi-directional mode.
	SetColorMap (see page 476)	This is SetColorMap, a member of class TCustomActionBar.
	SetDesignMode (see page 476)	This is SetDesignMode, a member of class TCustomActionBar.
	SetOrientation (see page 477)	This is SetOrientation, a member of class TCustomActionBar.
	SetPersistentHotKeys (see page 477)	This is SetPersistentHotKeys, a member of class TCustomActionBar.
	SetSpacing (see page 477)	This is SetSpacing, a member of class TCustomActionBar.
	SetupDefaultColorMap (see page 477)	This is SetupDefaultColorMap, a member of class TCustomActionBar.
	VisibleChanging (see page 478)	Provides the interface for a method that responds to changes in the control's visibility.

Public Methods

	Name	Description
	DragDrop (see page 466)	Generates an OnDragDrop event.
	FindFirst (see page 467)	Finds the first TActionClientItem (see page 374) in the Items (see page 488) list.
	FindFirstVisibleItem (see page 468)	Finds the first visible TActionClientItem (see page 374) in the Items (see page 488) list.

	FindLastVisibleItem (see page 469)	Finds the last visibleTActionClientItem in the Items (see page 488) list.
	RecreateControls (see page 475)	Destroys and recreates all of the action band's action client controls.
	SetBounds (see page 476)	Sets the bounds based on whether the popup is expanded or not
	Style (see page 477)	Identifies the style object that gives the application its default look and feel.

4.1.17.5.1 TCustomActionBar.AlignControls Method

Aligns any controls for which the control is the parent within a specified area of the control.

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

Use AlignControls to align all controls within an area of a form, panel, group box, scroll box, or any container control. Specify the area in which to align the controls as the value of the Rect parameter.

AlignControls uses the Align property value for each child control to determine how to align it. The AControl parameter can be nil (Delphi) or NULL (C++). If you specify a control in AControl, that control takes precedence in alignment over other, similarly-aligned controls.

See Also

Align (see page 5798)

4.1.17.5.2 TCustomActionBar.AutoSizingChanged Method

Pascal

```
procedure AutoSizingChanged; virtual;
```

C++

```
virtual __fastcall AutoSizingChanged();
```

Description

This is AutoSizingChanged, a member of class TCustomActionBar.

4.1.17.5.3 TCustomActionBar.BackgroundSize Method

Pascal

```
function BackgroundSize: TPoint;
```

C++

```
__fastcall TPoint BackgroundSize();
```

Description

This is BackgroundSize, a member of class TCustomActionBar.

4.1.17.5.4 TCustomActionBar.CalcInsertPosition Method

Pascal

```
function CalcInsertPosition(Control: TCustomActionControl; X: Integer; Y: Integer): Integer;
```

C++

```
__fastcall int CalcInsertPosition(TCustomActionControl Control, int X, int Y);
```

Description

This is CalcInsertPosition, a member of class TCustomActionBar.

4.1.17.5.5 TCustomActionBar.Clear Method

Pascal

```
procedure Clear; virtual;
```

C++

```
virtual __fastcall Clear();
```

Description

This is Clear, a member of class TCustomActionBar.

4.1.17.5.6 TCustomActionBar.CreateControl Method

Pascal

```
function CreateControl(AnItem: TActionClientItem): TCustomActionControl; virtual;
```

C++

```
virtual __fastcall TCustomActionControl CreateControl(TActionClientItem AnItem);
```

Description

This is CreateControl, a member of class TCustomActionBar.

4.1.17.5.7 TCustomActionBar.CreateControls Method

Pascal

```
procedure CreateControls; virtual;
```

C++

```
virtual __fastcall CreateControls();
```

Description

This is CreateControls, a member of class TCustomActionBar.

4.1.17.5.8 TCustomActionBar.CustomAlignInsertBefore Method

Returns placement order of custom-aligned child control.

Pascal

```
function CustomAlignInsertBefore(C1: TControl; C2: TControl): Boolean; override;
```

C++

```
virtual __fastcall Boolean CustomAlignInsertBefore(TControl * C1, TControl * C2);
```

Description

CustomAlignInsertBefore determines the order in which custom-aligned child controls are placed. The control calls CustomAlignInsertBefore once for each pair of child controls with an Align property of **alCustom**. CustomAlignInsertBefore

returns true if **C2** should be presented before **C1**, and false otherwise.

Once the custom-aligned child controls are sorted, their specific positions are determined by calls to CustomAlignPosition.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

As implemented in TWinControl, CustomAlignInsertBefore returns false if OnAlignInsertBefore is not implemented, so the controls are placed in the order they appear in the Controls property.

See Also

CustomAlignPosition (see page 6138)

Align (see page 5798)

Controls (see page 6197)

OnAlignInsertBefore (see page 6213)

4.1.17.5.9 TCustomActionBar.CustomAlignPosition Method

Returns aligned position of custom-aligned child control.

Pascal

```
procedure CustomAlignPosition(Control: TControl; var NewLeft: Integer; var NewTop: Integer;
var NewWidth: Integer; var NewHeight: Integer; var AlignRect: TRect; AlignInfo: TAlignInfo);
override;
```

C++

```
virtual __fastcall CustomAlignPosition(TControl * Control, int NewLeft, int NewTop, int
NewWidth, int NewHeight, TRect AlignRect, TAlignInfo AlignInfo);
```

Description

CustomAlignPosition determines the specific position of custom-aligned child controls. CustomAlignPosition is called once for each child control with an Align property of **alCustom**, in an order previously determined by calls to CustomAlignInsertBefore.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition.

These are the parameters:

Control	The child control being placed.
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. CustomAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. CustomAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, CustomAlignPosition does nothing.	

See Also

CustomAlignInsertBefore (see page 6137)

Align (see page 5798)

OnAlignPosition (see page 6214)

TAlignInfo (see page 6269)

4.1.17.5.10 TCustomActionBar.DesignWndProc Method

Pascal

```
function DesignWndProc(var Message: TMessage): Boolean; override;
```

C++

```
virtual __fastcall Boolean DesignWndProc(TMessage Message);
```

Description

This is DesignWndProc, a member of class TCustomActionBar.

4.1.17.5.11 TCustomActionBar.DoControlCreated Method

Pascal

```
procedure DoControlCreated(var Control: TCustomActionControl); virtual;
```

C++

```
virtual __fastcall DoControlCreated(TCustomActionControl Control);
```

Description

This is DoControlCreated, a member of class TCustomActionBar.

4.1.17.5.12 TCustomActionBar.DoDropActions Method

Pascal

```
procedure DoDropActions(Source: TActionDragObject; const X: Integer; const Y: Integer);
virtual;
```

C++

```
virtual __fastcall DoDropActions(TActionDragObject Source, const int X, const int Y);
```

Description

This is DoDropActions, a member of class TCustomActionBar.

4.1.17.5.13 TCustomActionBar.DoDropCategory Method

Pascal

```
procedure DoDropCategory(Source: TCategoryDragObject; const X: Integer; const Y: Integer);
virtual;
```

C++

```
virtual __fastcall DoDropCategory(TCategoryDragObject Source, const int X, const int Y);
```

Description

This is DoDropCategory, a member of class TCustomActionBar.

4.1.17.5.14 TCustomActionBar.DoDropItem Method

Pascal

```
procedure DoDropItem(Source: TActionItemDragObject; const X: Integer; const Y: Integer);
virtual;
```

C++

```
virtual __fastcall DoDropItem(TActionItemDragObject Source, const int X, const int Y);
```

Description

This is DoDropItem, a member of class TCustomActionBar.

4.1.17.5.15 TCustomActionBar.DoGetControlClass Method

Pascal

```
function DoGetControlClass(AnItem: TActionClientItem): TCustomActionControlClass; dynamic;
```

C++

```
__fastcall TCustomActionControlClass DoGetControlClass(TActionClientItem AnItem);
```

Description

This is DoGetControlClass, a member of class TCustomActionBar.

4.1.17.5.16 TCustomActionBar.DragDrop Method

Generates an OnDragDrop event.

Pascal

```
procedure DragDrop(Source: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DragDrop(TObject * Source, int X, int Y);
```

Description

DragDrop is called automatically when the user drops a control onto the TCustomActionBar (see page 447) descendant. When creating an action band that descends from TCustomActionBar (see page 447), override DragDrop to add additional code that executes when the OnDragDrop event handler is called.

The Source parameter is the object that was dropped onto the control. The X and Y parameters are the mouse coordinates where the object was dropped.

See Also

DragCanceled (see page 5754)

OnDragDrop (see page 5855)

4.1.17.5.17 TCustomActionBar.DragOver Method

OnDragOver event dispatcher.

Pascal

```
procedure DragOver(Source: TObject; X: Integer; Y: Integer; State: TDragState; var Accept:
Boolean); override;
```

C++

```
virtual __fastcall DragOver(TObject * Source, int X, int Y, TDragState State, Boolean Accept);
```

Description

Override DragOver to add additional code that executes before the OnDragOver event handler is called.

DragOver sets the Accept parameter to true to indicate that the user can drop the dragged object on the control. It sets Accept to false to indicate that the user cannot drop the dragged object on the control.

The Source parameter is the object being dragged. The State parameter indicates how the dragged object is moving in relation to the control. X and Y indicate the current position of the mouse.

See Also

DragDrop (see page 5754)

OnDragOver (see page 5857)

4.1.17.5.18 TCustomActionBar.DrawBackground Method

Pascal

```
procedure DrawBackground; virtual;
```

C++

```
virtual __fastcall DrawBackground();
```

Description

This is DrawBackground, a member of class TCustomActionBar.

4.1.17.5.19 TCustomActionBar.FindAccelItem Method

Pascal

```
function FindAccelItem(const Accel: Word): TActionClientItem;
```

C++

```
__fastcall TActionClientItem FindAccelItem(const Word Accel);
```

Description

This is FindAccelItem, a member of class TCustomActionBar.

4.1.17.5.20 TCustomActionBar.FindFirst Method

Finds the first TActionClientItem (see page 374) in the Items (see page 488) list.

Pascal

```
function FindFirst: TActionClientItem; virtual;
```

C++

```
virtual __fastcall TActionClientItem FindFirst();
```

Description

Use FindFirst to find the first TActionClientItem (see page 374) in the custom action bar's Items (see page 488) property.

See Also

Items (see page 488)

FindFirstVisibleItem (see page 468)

FindLastVisibleItem (see page 469)

4.1.17.5.21 TCustomActionBar.FindFirstVisibleItem Method

Finds the first visible TActionClientItem (see page 374) in the Items (see page 488) list.

Pascal

```
function FindFirstVisibleItem: TActionClientItem;
```

C++

```
__fastcall TActionClientItem FindFirstVisibleItem();
```

Description

Use FindFirst (see page 467) to find the first visible TActionClientItem (see page 374) in the custom action bar's Items (see page 488) property.

See Also

Items (see page 488)

FindFirst (see page 467)

FindLastVisibleItem (see page 469)

4.1.17.5.22 TCustomActionBar.FindItemAt Method

Pascal

```
function FindItemAt(const P: TPoint): TCustomActionControl;
```

C++

```
__fastcall TCustomActionControl FindItemAt(const TPoint P);
```

Description

This is FindItemAt, a member of class TCustomActionBar.

4.1.17.5.23 TCustomActionBar.FindItemCaption Method

Pascal

```
function FindItemCaption(const ACaption: string): TCustomActionControl;
```

C++

```
__fastcall TCustomActionControl FindItemCaption(const AnsiString ACaption);
```

Description

This is FindItemCaption, a member of class TCustomActionBar.

4.1.17.5.24 TCustomActionBar.FindLast Method

Pascal

```
function FindLast: TActionClientItem; virtual;
```

C++

```
virtual __fastcall TActionClientItem FindLast();
```

Description

This is FindLast, a member of class TCustomActionBar.

4.1.17.5.25 TCustomActionBar.FindLastVisibleItem Method

Finds the last visibleTActionClientItem in the Items (see page 488) list.

Pascal

```
function FindLastVisibleItem: TActionClientItem;
```

C++

```
__fastcall TActionClientItem FindLastVisibleItem();
```

Description

Use FindFirst (see page 467) to find the last visible TActionClientItem (see page 374) in the custom action bar's Items (see page 488) property.

See Also

Items (see page 488)

FindFirst (see page 467)

FindFirstVisibleItem (see page 468)

4.1.17.5.26 TCustomActionBar.FindLeastUsedItem Method

Pascal

```
function FindLeastUsedItem(const Visible: Boolean = True): TActionClientItem;
```

C++

```
__fastcall TActionClientItem FindLeastUsedItem(const Boolean Visible = True);
```

Description

This is FindLeastUsedItem, a member of class TCustomActionBar.

4.1.17.5.27 TCustomActionBar.FindNearestControl Method

Pascal

```
function FindNearestControl(const Point: TPoint): TCustomActionControl;
```

C++

```
__fastcall TCustomActionControl FindNearestControl(const TPoint Point);
```

Description

This is FindNearestControl, a member of class TCustomActionBar.

4.1.17.5.28 TCustomActionBar.FindNext Method

Pascal

```
function FindNext(AClient: TActionClientItem; const Wrap: Boolean = True): TActionClientItem;
virtual;
```

C++

```
virtual __fastcall TActionClientItem FindNext(TActionClientItem AClient, const Boolean Wrap =
True);
```

Description

This is FindNext, a member of class TCustomActionBar.

4.1.17.5.29 TCustomActionBar.FindNextVisibleItem Method

Pascal

```
function FindNextVisibleItem(AClient: TActionClientItem): TActionClientItem;
```

C++

```
__fastcall TActionClientItem FindNextVisibleItem(TActionClientItem AClient);
```

Description

This is FindNextVisibleItem, a member of class TCustomActionBar.

4.1.17.5.30 TCustomActionBar.FindPrevious Method

Pascal

```
function FindPrevious(AClient: TActionClientItem; const Wrap: Boolean = True):
TActionClientItem; virtual;
```

C++

```
virtual __fastcall TActionClientItem FindPrevious(TActionClientItem AClient, const Boolean
Wrap = True);
```

Description

This is FindPrevious, a member of class TCustomActionBar.

4.1.17.5.31 TCustomActionBar.FindPreviousVisibleItem Method

Pascal

```
function FindPreviousVisibleItem(AClient: TActionClientItem): TActionClientItem;
```

C++

```
__fastcall TActionClientItem FindPreviousVisibleItem(TActionClientItem AClient);
```

Description

This is FindPreviousVisibleItem, a member of class TCustomActionBar.

4.1.17.5.32 TCustomActionBar.GetActionControl Method

Pascal

```
function GetActionControl(const Index: Integer): TCustomActionControl; virtual;
```

C++

```
virtual __fastcall TCustomActionControl GetActionControl(const int Index);
```

Description

This is GetActionControl, a member of class TCustomActionBar.

4.1.17.5.33 TCustomActionBar.GetBannerWidth Method

Pascal

```
function GetBannerWidth(BarEdge: TBarEdge): Integer; virtual;
```

C++

```
virtual __fastcall int GetBannerWidth(TBarEdge BarEdge);
```

Description

This is GetBannerWidth, a member of class TCustomActionBar.

4.1.17.5.34 TCustomActionBar.GetBarHeight Method

Pascal

```
function GetBarHeight: Integer; virtual;
```

C++

```
virtual __fastcall int GetBarHeight();
```

Description

This is GetBarHeight, a member of class TCustomActionBar.

4.1.17.5.35 TCustomActionBar.GetBarWidth Method

Pascal

```
function GetBarWidth: Integer; virtual;
```

C++

```
virtual __fastcall int GetBarWidth();
```

Description

This is GetBarWidth, a member of class TCustomActionBar.

4.1.17.5.36 TCustomActionBar.GetControlClass Method

Pascal

```
function GetControlClass(AnItem: TActionClientItem): TCustomActionControlClass; virtual;
```

C++

```
virtual __fastcall TCustomActionControlClass GetControlClass(TActionClientItem AnItem);
```

Description

This is GetControlClass, a member of class TCustomActionBar.

4.1.17.5.37 TCustomActionBar.GetDefaultColorMapClass Method

Pascal

```
function GetDefaultColorMapClass: TCustomColorMapClass; virtual;
```

C++

```
virtual __fastcall TCustomColorMapClass GetDefaultColorMapClass();
```

Description

This is GetDefaultColorMapClass, a member of class TCustomActionBar.

4.1.17.5.38 TCustomActionBar.GetDesignMode Method

Pascal

```
function GetDesignMode: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetDesignMode();
```

Description

This is GetDesignMode, a member of class TCustomActionBar.

4.1.17.5.39 TCustomActionBar.GetPopupMenuClass Method

Pascal

```
function GetPopupMenuClass: TCustomActionBarClass;
```

C++

```
__fastcall TCustomActionBarClass GetPopupMenuClass();
```

Description

This is GetPopupMenuClass, a member of class TCustomActionBar.

4.1.17.5.40 TCustomActionBar.HasItems Method

Pascal

```
function HasItems: Boolean;
```

C++

```
__fastcall Boolean HasItems();
```

Description

This is HasItems, a member of class TCustomActionBar.

4.1.17.5.41 TCustomActionBar.ItemCount Method

Pascal

```
function ItemCount: Integer; virtual;
```

C++

```
virtual __fastcall int ItemCount();
```

Description

This is ItemCount, a member of class TCustomActionBar.

4.1.17.5.42 TCustomActionBar.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded (see page 2782)

Action (see page 5797)

ActionChange (see page 5737)

4.1.17.5.43 TCustomActionBar.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TCustomActionBar.

4.1.17.5.44 TCustomActionBar.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

4.1.17.5.45 TCustomActionBar.Paint Method

Pascal

```
procedure Paint; virtual;
```

C++

```
virtual __fastcall Paint();
```

Description

This is Paint, a member of class TCustomActionBar.

4.1.17.5.46 TCustomActionBar.PaintWindow Method

Renders the image of a windowed control.

Pascal

```
procedure PaintWindow(DC: HDC); override;
```

C++

```
virtual __fastcall PaintWindow(HDC DC);
```

Description

Call PaintWindow to repaint the control. PaintWindow sends a WM_PAINT message to the windowed control's DefaultHandler method, setting the message record's WParam field to the value passed in DC and the other parameter and result fields to zeros.

See Also

DefaultHandler (see page 6139)

PaintTo (see page 6167)

DoubleBuffered (see page 6205)

4.1.17.5.47 TCustomActionBar.RecreateControls Method

Destroys and recreates all of the action band's action client controls.

Pascal

```
procedure RecreateControls; virtual;
```

C++

```
virtual __fastcall RecreateControls();
```

Description

Call RecreateControls to recreate all of the action client child controls contained by the action band from scratch. RecreateControls destroys the action client controls and creates a new component according to the current property settings.

4.1.17.5.48 TCustomActionBar.Reset Method

Pascal

```
procedure Reset; virtual;
```

C++

```
virtual __fastcall Reset();
```

Description

This is Reset, a member of class TCustomActionBar.

4.1.17.5.49 TCustomActionBar.SetActionClient Method

Pascal

```
procedure SetActionClient(const Value: TActionClient); virtual;
```

C++

```
virtual __fastcall SetActionClient(const TActionClient Value);
```

Description

This is SetActionClient, a member of class TCustomActionBar.

4.1.17.5.50 TCustomActionBar.SetAutoSizing Method

Pascal

```
procedure SetAutoSizing(const Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetAutoSizing(const Boolean Value);
```

Description

This is SetAutoSizing, a member of class TCustomActionBar.

4.1.17.5.51 TCustomActionBar.SetBiDiMode Method

Sets the control's bi-directional mode.

Pascal

```
procedure SetBiDiMode(Value: TBiDiMode); override;
```

C++

```
virtual __fastcall SetBiDiMode(TBiDiMode Value);
```

Description

SetBiDiMode is the write implementation of the control's BiDiMode property. Override this method to change the implementation of BiDiMode.

See Also

BiDiMode (see page 5803)

4.1.17.5.52 TCustomActionBar.SetBounds Method

Sets the bounds based on whether the popup is expanded or not

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

SetBounds calls the inherited method after altering the AWidth and AHeight parameters (if necessary) to include the width and height of the expanded popup menu. For more information about SetBounds, see the documentation for the inherited method.

See Also

SetBounds (see page 6178)

4.1.17.5.53 TCustomActionBar.SetColorMap Method

Pascal

```
procedure SetColorMap(const Value: TCustomActionBarColorMap); virtual;
```

C++

```
virtual __fastcall SetColorMap(const TCustomActionBarColorMap Value);
```

Description

This is SetColorMap, a member of class TCustomActionBar.

4.1.17.5.54 TCustomActionBar.SetDesignMode Method

Pascal

```
procedure SetDesignMode(const Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetDesignMode(const Boolean Value);
```

Description

This is SetDesignMode, a member of class TCustomActionBar.

4.1.17.5.55 TCustomActionBar.SetOrientation Method

Pascal

```
procedure SetOrientation(const Value: TBarOrientation); virtual;
```

C++

```
virtual __fastcall SetOrientation(const TBarOrientation Value);
```

Description

This is SetOrientation, a member of class TCustomActionBar.

4.1.17.5.56 TCustomActionBar.SetPersistentHotKeys Method

Pascal

```
procedure SetPersistentHotKeys(const Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetPersistentHotKeys(const Boolean Value);
```

Description

This is SetPersistentHotKeys, a member of class TCustomActionBar.

4.1.17.5.57 TCustomActionBar.SetSpacing Method

Pascal

```
procedure SetSpacing(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetSpacing(int Value);
```

Description

This is SetSpacing, a member of class TCustomActionBar.

4.1.17.5.58 TCustomActionBar.SetupDefaultColorMap Method

Pascal

```
function SetupDefaultColorMap: TCustomActionBarColorMap;
```

C++

```
__fastcall TCustomActionBarColorMap SetupDefaultColorMap();
```

Description

This is SetupDefaultColorMap, a member of class TCustomActionBar.

4.1.17.5.59 TCustomActionBar.Style Method

Identifies the style object that gives the application its default look and feel.

Pascal

```
function Style: TActionBarStyle; virtual;
```

C++

```
virtual __fastcall TActionBarStyle Style();
```

Description

Use Style to access the application's Style object. This object has a number of methods that influence which type of controls are created on the action band.

See Also

Style (see page 556)

4.1.17.5.60 TCustomActionBar.VisibleChanging Method

Provides the interface for a method that responds to changes in the control's visibility.

Pascal

```
procedure VisibleChanging; override;
```

C++

```
virtual __fastcall VisibleChanging();
```

Description

VisibleChanging is called before the value of a control's Visible property changes. As implemented in TControl, it does nothing. Override VisibleChanging to respond to changes in the control's visibility. For example, if a control should not change its visibility, override VisibleChanging to raise an exception when an attempt is made to alter the control's visibility.

See Also

Visible (see page 5847)

4.1.17.6 TCustomActionBar Properties

The properties of the TCustomActionBar class are listed here.

Protected Properties

	Name	Description
R	HRowCount (see page 488)	Indicates the number of columns of items that the action band displays.
R	Items (see page 488)	Lists the action clients that correspond to the child controls of this action band.
R	VRowCount (see page 490)	Indicates the number of rows of items that the action band displays.
R	Widest (see page 491)	This is Widest, a member of class TCustomActionBar.

Public Properties

	Name	Description
	ActionClient (see page 479)	Specifies the action client associated with this action band.
R	ActionControls (see page 479)	Lists the controls on the action band.
	ActionManager (see page 480)	Specifies the action manager component that is associated with the action band.
	AllowHiding (see page 480)	Determines whether the action band can be hidden.
	AutoSizing (see page 480)	Specifies whether the action band adjusts its size automatically to fit around its child controls.
R	Canvas (see page 481)	Represents the drawing surface of the action band.
	Caption (see page 481)	Specifies a text string that identifies the control to the user.
	Color (see page 483)	Specifies the background color of the control.
	ColorMap (see page 484)	Indicates the colormap for individual action band components.
	ContextBar (see page 485)	Indicates whether items shown in this action band represent the items in a popup menu.

	Designable (see page 485)	Specifies whether the action band can be customized (used internally by the IDE).
	DesignMode (see page 486)	Specifies whether the action band is being customized or being used in the IDE.
	Font (see page 486)	Controls the attributes of text written on or in the control.
	HorzMargin (see page 487)	Specifies the width of the horizontal margins in pixels.
	HorzSeparator (see page 487)	Indicates whether a horizontal separator bar appears between item lines.
	Orientation (see page 488)	Specifies how the action band lays out its child items.
	PersistentHotKeys (see page 489)	Specifies whether keyboard accelerator shortcuts are always displayed.
	Spacing (see page 489)	Determines where the image and text appear on an action band's rendered control that represents the action client.
	VertMargin (see page 490)	Specifies the width of the vertical margins in pixels.
	VertSeparator (see page 490)	Indicates whether a vertical separator bar should appear between item lines.

4.1.17.6.1 TCustomActionBar.ActionClient Property

Specifies the action client associated with this action band.

Pascal

```
property ActionClient: TActionClient;
```

C++

```
__property TActionClient ActionClient;
```

Description

Use ActionClient to access the action client that corresponds to this action band. Action clients are the components that the action manager uses to represent nodes in a hierarchy whose leaves correspond to actions. This association between action client and action bar is set up by specifying this action bar as the value of the action client's ActionBar property. The action client can be found as one of the items listed by the action manager's ActionBars property.

See Also

TActionManager (see page 432)

ActionBars (see page 553)

ActionBar (see page 370)

4.1.17.6.2 TCustomActionBar.ActionControls Property

Lists the controls on the action band.

Pascal

```
property ActionControls [const Index: Integer]: TCustomActionControl;
```

C++

```
__property TCustomActionControl ActionControls[int const Index];
```

Description

Use ActionControls to access any of the controls available on the action band. Each control in the list is the user interface element generated to represent one of the TActionClientItem (see page 374) objects that the action manager stores for children of this action band. The TActionClientItem (see page 374) objects are children of the component that is the value of this action band's ActionClient (see page 479) property.

Index is the index of the child control, where 0 specifies the first control, 1 specifies the second control, and so on.

See Also

ActionClient (see page 479)

ContextBar (see page 485)

4.1.17.6.3 TCustomActionBar.ActionManager Property

Specifies the action manager component that is associated with the action band.

Pascal

```
property ActionManager: TCustomActionManager;
```

C++

```
__property TCustomActionManager ActionManager;
```

Description

Use ActionManager to access the action manager that uses this action band to render UI elements it manages. Typically, this association is created when you drag actions from the action manager and drop them on the action bar. When an action is dropped on an action band and the ActionManager property is not assigned, the ActionManager property is automatically assigned to the action manager that manages the action that was dropped on the action band.

At runtime, you can perform the equivalent operation to dragging an action onto the action band by setting the ActionBar property of one of the action clients that the action manager's ActionBars property lists so that it specifies this component.

See Also

ActionBars (see page 553)

ActionClient (see page 479)

ActionBar (see page 370)

4.1.17.6.4 TCustomActionBar.AllowHiding Property

Determines whether the action band can be hidden.

Pascal

```
property AllowHiding: Boolean;
```

C++

```
__property Boolean AllowHiding;
```

Description

Use AllowHiding to specify whether the action band can be hidden at runtime, if the application allows runtime customization by the end user. Setting AllowHiding to true allows the action band to be hidden at runtime. If AllowHiding is set to false, the action cannot be hidden. Certain action bands, such as the main menu action band, should set this AllowHiding to false.

4.1.17.6.5 TCustomActionBar.AutoSizing Property

Specifies whether the action band adjusts its size automatically to fit around its child controls.

Pascal

```
property AutoSizing: Boolean;
```

C++

```
__property Boolean AutoSizing;
```

Description

Use AutoSizing to specify whether the action band resizes dynamically to fit around the controls that the ActionControls (see

page 479) property lists. Set AutoSizing to true to cause the action band to resize itself so that child controls fit neatly into its client area. Set AutoSizing to false if you want the action band to be fixed in size.

4.1.17.6.6 TCustomActionBar.Canvas Property

Represents the drawing surface of the action band.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

TCustomActionBar (see page 447) uses Canvas internally to render the background on which its child controls appear.

4.1.17.6.7 TCustomActionBar.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
```

```
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
```

```
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

4.1.17.6.8 TCustomActionBar.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

4.1.17.6.9 TCustomActionBar.ColorMap Property

Indicates the colormap for individual action band components.

Pascal

```
property ColorMap: TCustomActionBarColorMap;
```

C++

```
__property TCustomActionBarColorMap ColorMap;
```

Description

Use the ColorMap property to change the default colormap for individual action band components. By default, the ColorMap property is set according to the first registered TActionBarStyleEx class. Otherwise, if the action band is linked to an action manager, the property defaults to the ColorMap as specified by the Style (see page 477) property of the action manager.

Note: To centrally set or change the default look and feel of all action band components, use ActionManager (see page 480)'s

Style (see page 477) property. To centrally control the colors used by all action bands within an application, connect the action bands to a single colormap component.

See Also

TActionManager (see page 432)

TXPColorMap (see page 23)

TStandardColorMap (see page 2)

TTwilightColorMap (see page 13)

Style (see page 556)

4.1.17.6.10 TCustomActionBar.ContextBar Property

Indicates whether items shown in this action band represent the items in a popup menu.

Pascal

```
property ContextBar: Boolean;
```

C++

```
__property Boolean ContextBar;
```

Description

ContextBar controls which items this action band lists in its ActionControls (see page 479) property. The action band's associated action client (the value of the ActionClient (see page 479) property) can contain two lists of items:

One list (the action client's Items (see page 488) property) lists all the child items the action band can display when it is a regular UI element such as a main menu or tool bar. TCustomActionBar (see page 447) uses this list when ContextBar is false.

The other list (the action client's ContextItems property) lists all the child items that should appear if the action band appears in response to a right click. (For example, the ContextItems property is used when the action band represents a context menu). TCustomActionBar (see page 447) uses this list when ContextBar is true.

See Also

Items (see page 488)

ContextItems (see page 372)

ActionClient (see page 479)

ActionControls (see page 479)

4.1.17.6.11 TCustomActionBar.Designable Property

Specifies whether the action band can be customized (used internally by the IDE).

Pascal

```
property Designable: Boolean;
```

C++

```
__property Boolean Designable;
```

Description

Designable is used internally to control whether or not the action band can be customized either at runtime using the customize dialog, or at design time within the IDE. For example, the action band automatically supports drag-and-drop operations only

when Designable is true.

4.1.17.6.12 TCustomActionBar.DesignMode Property

Specifies whether the action band is being customized or being used in the IDE.

Pascal

```
property DesignMode: Boolean;
```

C++

```
__property Boolean DesignMode;
```

Description

DesignMode is used internally to determine whether the action band is being customized, either at runtime using the customize dialog, or at design time within the IDE. For example, the action band automatically supports drag-and-drop operations only when DesignMode is true.

4.1.17.6.13 TCustomActionBar.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

4.1.17.6.14 TCustomActionBar.HorzMargin Property

Specifies the width of the horizontal margins in pixels.

Pascal

```
property HorzMargin: Integer;
```

C++

```
__property int HorzMargin;
```

Description

HorzMargin specifies the width of the action band's left and right horizontal margins in pixels.

See Also

VertMargin (see page 490)

VertSeparator (see page 490)

4.1.17.6.15 TCustomActionBar.HorzSeparator Property

Indicates whether a horizontal separator bar appears between item lines.

Pascal

```
property HorzSeparator: Boolean;
```

C++

```
__property Boolean HorzSeparator;
```

Description

An action band can arrange its child controls using multiple lines. Items (see page 488) which cannot be displayed on one line wrap to the next in the same way that words wrap on most word processors. Set HorzSeparator to true if you want a separator bar to appear between the lines.

See Also

VertSeparator (see page 490)

HorzMargin (see page 487)

Orientation (see page 488)

4.1.17.6.16 TCustomActionBar.HRowCount Property

Indicates the number of columns of items that the action band displays.

Pascal

```
property HRowCount: Integer;
```

C++

```
__property int HRowCount;
```

Description

If the action band arranges its items in columns (that is, when the Orientation (see page 488) property is boTopToBottom or boBottomToTop), the action band uses HRowCount internally to determine how many columns are needed to display all of the items that are not hidden.

See Also

VertSeparator (see page 490)

VRowCount (see page 490)

Orientation (see page 488)

4.1.17.6.17 TCustomActionBar.Items Property

Lists the action clients that correspond to the child controls of this action band.

Pascal

```
property Items: TActionClients;
```

C++

```
__property TActionClients Items;
```

Description

Items is used internally to access the components that the action manager lists to represent the child controls of this action band. Each action client object in the Items collection corresponds to one of the controls that the ActionControls (see page 479) property lists. The entire collection of items belongs to the action client specified by this action band's ActionClient (see page 479) property.

See Also

ActionControls (see page 479)

ActionClient (see page 479)

ContextBar (see page 485)

4.1.17.6.18 TCustomActionBar.Orientation Property

Specifies how the action band lays out its child items.

Pascal

```
property Orientation: TBarOrientation;
```

C++

```
__property TBarOrientation Orientation;
```

Description

The possible values of Orientation are:

Value	Meaning
boLeftToRight	Items (see page 488) are rendered in rows, from left to right.
boRightToLeft	Items (see page 488) are rendered in rows, from right to left.
boTopToBottom	Items (see page 488) are rendered in columns, from top to bottom.
boBottomToTop	Items (see page 488) are rendered in columns, from bottom to top.

See Also

VertSeparator (see page 490)

HorzSeparator (see page 487)

ActionControls (see page 479)

4.1.17.6.19 TCustomActionBar.PersistentHotKeys Property

Specifies whether keyboard accelerator shortcuts are always displayed.

Pascal

```
property PersistentHotKeys: Boolean;
```

C++

```
__property Boolean PersistentHotKeys;
```

Description

Use PersistentHotKeys to specify whether the action band always displays keyboard accelerator shortcuts. Setting PersistentHotKeys to true causes the underline for the keyboard shortcut key to always appear. If PersistentHotKeys is set to false, the underline for the keyboard shortcut key is only displayed if the user has used the keyboard to navigate through the action band's items. If the user is only using the mouse to navigate through the action band's items and PersistentHotKeys is set to false, the underline under the character denoted as the shortcut key is not drawn.

Note: Accelerator shortcuts are specified by preceding the appropriate character in the item's caption with an ampersand.

See Also

Caption (see page 384)

4.1.17.6.20 TCustomActionBar.Spacing Property

Determines where the image and text appear on an action band's rendered control that represents the action client.

Pascal

```
property Spacing: Integer;
```

C++

```
__property int Spacing;
```

Description

Spacing is used internally to set the number of pixels that should appear between the image specified in an action client's ImageIndex and the text specified in its Caption (see page 481) property.

See Also

Caption (see page 384)

ImageIndex (see page 385)

4.1.17.6.21 TCustomActionBar.VertMargin Property

Specifies the width of the vertical margins in pixels.

Pascal

```
property VertMargin: Integer;
```

C++

```
__property int VertMargin;
```

Description

VertMargin specifies the width of the action band's top and bottom vertical margins in pixels.

See Also

HorzMargin (see page 487)

HorzSeparator (see page 487)

4.1.17.6.22 TCustomActionBar.VertSeparator Property

Indicates whether a vertical separator bar should appear between item lines.

Pascal

```
property VertSeparator: Boolean;
```

C++

```
__property Boolean VertSeparator;
```

Description

An action band can arrange its child controls using multiple columns. Items (see page 488) which cannot be displayed on one column wrap to the next. Set VertSeparator to true if you want a separator bar to appear between the columns.

See Also

HorzSeparator (see page 487)

Orientation (see page 488)

4.1.17.6.23 TCustomActionBar.VRowCount Property

Indicates the number of rows of items that the action band displays.

Pascal

```
property VRowCount: Integer;
```

C++

```
__property int VRowCount;
```

Description

If the action band arranges its items in rows (that is, when the Orientation (see page 488) property is boLeftToRight or boRightToLeft), the action band uses VRowCount internally to determine how many rows are needed to display all of the items that are not hidden.

See Also

HorzSeparator (see page 487)

HRowCount (see page 488)

Orientation (see page 488)

4.1.17.6.24 TCustomActionBar.Widest Property

Pascal

```
property Widest: Integer;
```

C++

```
__property int Widest;
```

Description

This is Widest, a member of class TCustomActionBar.

4.1.17.7 TCustomActionBar Events

The events of the TCustomActionBar class are listed here.

Public Events

	Name	Description
	OnControlCreated (see page 491)	Occurs when a control is created on the action band.
	OnGetControlClass (see page 492)	Occurs when an item is to be placed on the action band.
	OnPaint (see page 492)	Occurs when the action band is painted.

4.1.17.7.1 TCustomActionBar.OnControlCreated Event

Occurs when a control is created on the action band.

Pascal

```
property OnControlCreated: TControlCreatedEvent;
```

C++

```
__property TControlCreatedEvent OnControlCreated;
```

Description

Write an OnControlCreated event handler to enhance or replace the type of UI element that is rendered on the action band.

Sender is the action band component that just created a new control.

AnItem is the new child control that was just created. The OnControlCreated event handler can initialize the properties of this control or replace it with a custom control.

See Also

OnGetControlClass (see page 492)

4.1.17.7.2 TCustomActionBar.OnGetControlClass Event

Occurs when an item is to be placed on the action band.

Pascal

```
property OnGetControlClass: TGetControlClassEvent;
```

C++

```
__property TGetControlClassEvent OnGetControlClass;
```

Description

Write an OnGetControlClass event handler to specify the class of item that the action band should create to represent a specified action client.

Sender is the action band that needs to generate a child control to represent an action client.

AnItem is the action client that the new control should represent.

ControlClass returns the class reference for the new control. This should be a descendant of TCustomActionControl (see page 507).

See Also

TCustomActionControl (see page 507)

OnControlCreated (see page 491)

4.1.17.7.3 TCustomActionBar.OnPaint Event

Occurs when the action band is painted.

Pascal

```
property OnPaint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPaint;
```

Description

Write an OnPaint event handler to change the way the action band renders itself.

See Also

Canvas (see page 481)

4.1.18 TCustomActionBarColorMap Class

TCustomActionBarColorMap is the base class for the action band colormaps that controls the color of the action band and the items that appear on it.

Class Hierarchy

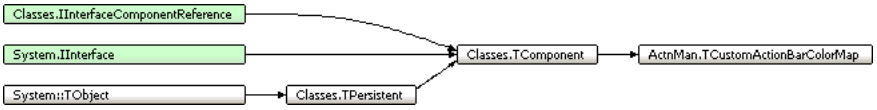

Pascal

```
TCustomActionBarColorMap = class(TComponent);
```

C++

```
class TCustomActionBarColorMap : public TComponent;
```

File

ActnMan

Description

TCustomActionBarColorMap provides the properties, methods, and events for colormaps, such as StandardColorMap and XPColorMap. Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

See Also

TActionManager (see page 432)

TCustomActionManager (see page 537)

TXPColorMap (see page 23)

TStandardColorMap (see page 2)

TTwilightColorMap (see page 13)

4.1.18.1 TCustomActionBarColorMap Members

The following tables list the members exposed by TCustomActionBarColorMap.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	Add (see page 497)	This is Add, a member of class TCustomActionBarColorMap.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 498)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	Remove (see page 498)	This is Remove, a member of class TCustomActionBarColorMap.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetColor (see page 499)	This is SetColor, a member of class TCustomActionBarColorMap.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFontColor (see page 499)	This is SetFontColor, a member of class TCustomActionBarColorMap.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	UpdateActionBars (see page 499)	This is UpdateActionBars, a member of class TCustomActionBarColorMap.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnColorChange (see page 506)	Occurs when one of the action's color properties changes.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 497)	Provides a custom Assign procedure for TCustomActionBarColorMap (see page 492) objects.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 496)	Creates and initializes a TCustomActionBarColorMap (see page 492) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 496)	Destroys an instance of TCustomActionBarColorMap (see page 492).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateColors (see page 499)	Occurs when an ActionBand component receives a CM_WININICHANGE message.

Public Properties

	Name	Description
	BtnFrameColor (see page 500)	Changes the look and feel of your applications.
	BtnSelectedColor (see page 500)	Changes the look and feel of your applications.
	BtnSelectedFont (see page 501)	Changes the look and feel of your applications.

	Color (see page 501)	Specifies the background color for action bands.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignFocus (see page 501)	Changes the look and feel of your applications.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisabledColor (see page 501)	Changes the look and feel of your applications.
	DisabledFontColor (see page 502)	Changes the look and feel of your applications.
	DisabledFontShadow (see page 502)	Changes the look and feel of your applications.
	FontColor (see page 502)	Changes the look and feel of your applications.
	FrameBottomRightInner (see page 502)	Changes the look and feel of your applications.
	FrameBottomRightOuter (see page 503)	Changes the look and feel of your applications.
	FrameTopLeftInner (see page 503)	Changes the look and feel of your applications.
	FrameTopLeftOuter (see page 503)	Changes the look and feel of your applications.
	HighlightColor (see page 503)	Changes the look and feel of your applications.
	HotColor (see page 504)	Changes the look and feel of your applications.
	HotFontColor (see page 504)	Changes the look and feel of your applications.
	MenuColor (see page 504)	Changes the look and feel of your applications.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SelectedColor (see page 505)	Changes the look and feel of your applications.
	SelectedFontColor (see page 505)	Changes the look and feel of your applications.
	ShadowColor (see page 505)	Changes the look and feel of your applications.
	UnusedColor (see page 505)	Changes the look and feel of your applications.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

4.1.18.2 TCustomActionBarColorMap.Create Constructor

Creates and initializes a TCustomActionBarColorMap (see page 492) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomActionBarColorMap(TComponent * AOwner);
```

Description

Applications do not usually create action objects explicitly. Actions are created automatically when you add them to an action manager or action list component at design time.

4.1.18.3 TCustomActionBarColorMap.Destroy Destructor

Destroys an instance of TCustomActionBarColorMap (see page 492).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomActionBarColorMap();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which checks that the TCustomActionBarColorMap (see page 492) reference is not nil before calling Destroy.

See Also

Create (see page 496)

Free (see page 22300)

4.1.18.4 TCustomActionBarColorMap Methods

The methods of the TCustomActionBarColorMap class are listed here.

Protected Methods

	Name	Description
	Add (see page 497)	This is Add, a member of class TCustomActionBarColorMap.
V	Notification (see page 498)	Forwards notification messages to all owned components.
	Remove (see page 498)	This is Remove, a member of class TCustomActionBarColorMap.
V	SetColor (see page 499)	This is SetColor, a member of class TCustomActionBarColorMap.
V	SetFontColor (see page 499)	This is SetFontColor, a member of class TCustomActionBarColorMap.
	UpdateActionBars (see page 499)	This is UpdateActionBars, a member of class TCustomActionBarColorMap.

Public Methods

	Name	Description
V	Assign (see page 497)	Provides a custom Assign procedure for TCustomActionBarColorMap (see page 492) objects.
V	UpdateColors (see page 499)	Occurs when an ActionBand component receives a CM_WININICHANGE message.

4.1.18.4.1 TCustomActionBarColorMap.Add Method

Pascal

```
procedure Add(ActionBar: TCustomActionBar);
```

C++

```
__fastcall Add(TCustomActionBar ActionBar);
```

Description

This is Add, a member of class TCustomActionBarColorMap.

4.1.18.4.2 TCustomActionBarColorMap.Assign Method

Provides a custom Assign procedure for TCustomActionBarColorMap (see page 492) objects.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use this override of the inherited TPersistent procedure to assign the properties of one TCustomActionBarColorMap (see page 492) object to another. The ActionBars, Images, PrioritySchedule, and LinkedActionLists items of the current TCustomActionBarColorMap (see page 492) all refer to the same objects as in the Source. In the case of the FileName property, it becomes a copy of the Source property of the same name. If Source is not an action manager, the inherited procedure is called.

See Also

Assign (see page 2930)

4.1.18.4.3 TCustomActionBarColorMap.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

4.1.18.4.4 TCustomActionBarColorMap.Remove Method

Pascal

```
procedure Remove(ActionBar: TCustomActionBar);
```

C++

```
__fastcall Remove(TCustomActionBar ActionBar);
```

Description

This is Remove, a member of class TCustomActionBarColorMap.

4.1.18.4.5 TCustomActionBarColorMap.SetColor Method

Pascal

```
procedure SetColor(const Value: TColor); virtual;
```

C++

```
virtual __fastcall SetColor(const TColor Value);
```

Description

This is SetColor, a member of class TCustomActionBarColorMap.

4.1.18.4.6 TCustomActionBarColorMap.SetFontColor Method

Pascal

```
procedure SetFontColor(const Value: TColor); virtual;
```

C++

```
virtual __fastcall SetFontColor(const TColor Value);
```

Description

This is SetFontColor, a member of class TCustomActionBarColorMap.

4.1.18.4.7 TCustomActionBarColorMap.UpdateActionBars Method

Pascal

```
procedure UpdateActionBars(UpdateProc: TColorMapUpdateProc; Color: TColor);
```

C++

```
__fastcall UpdateActionBars(TColorMapUpdateProc UpdateProc, TColor Color);
```

Description

This is UpdateActionBars, a member of class TCustomActionBarColorMap.

4.1.18.4.8 TCustomActionBarColorMap.UpdateColors Method

Occurs when an ActionBand component receives a CM_WININICHANGE message.

Pascal

```
procedure UpdateColors; virtual;
```

C++

```
virtual __fastcall UpdateColors();
```

Description

UpdateColors is called automatically when an ActionBand component receives a CM_WININICHANGE message.

4.1.18.5 TCustomActionBarColorMap Properties

The properties of the TCustomActionBarColorMap class are listed here.

Public Properties

	Name	Description
	BtnFrameColor (see page 500)	Changes the look and feel of your applications.
	BtnSelectedColor (see page 500)	Changes the look and feel of your applications.
	BtnSelectedFont (see page 501)	Changes the look and feel of your applications.
	Color (see page 501)	Specifies the background color for action bands.
	DesignFocus (see page 501)	Changes the look and feel of your applications.
	DisabledColor (see page 501)	Changes the look and feel of your applications.
	DisabledFontColor (see page 502)	Changes the look and feel of your applications.
	DisabledFontShadow (see page 502)	Changes the look and feel of your applications.
	FontColor (see page 502)	Changes the look and feel of your applications.
	FrameBottomRightInner (see page 502)	Changes the look and feel of your applications.
	FrameBottomRightOuter (see page 503)	Changes the look and feel of your applications.
	FrameTopLeftInner (see page 503)	Changes the look and feel of your applications.
	FrameTopLeftOuter (see page 503)	Changes the look and feel of your applications.
	HighlightColor (see page 503)	Changes the look and feel of your applications.
	HotColor (see page 504)	Changes the look and feel of your applications.
	HotFontColor (see page 504)	Changes the look and feel of your applications.
	MenuColor (see page 504)	Changes the look and feel of your applications.
	SelectedColor (see page 505)	Changes the look and feel of your applications.
	SelectedFontColor (see page 505)	Changes the look and feel of your applications.
	ShadowColor (see page 505)	Changes the look and feel of your applications.
	UnusedColor (see page 505)	Changes the look and feel of your applications.

4.1.18.5.1 TCustomActionBarColorMap.BtnFrameColor Property

Changes the look and feel of your applications.

Pascal

```
property BtnFrameColor: TColor;
```

C++

```
__property TColor BtnFrameColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.2 TCustomActionBarColorMap.BtnSelectedColor Property

Changes the look and feel of your applications.

Pascal

```
property BtnSelectedColor: TColor;
```

C++

```
__property TColor BtnSelectedColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.3 TCustomActionBarColorMap.BtnSelectedFont Property

Changes the look and feel of your applications.

Pascal

```
property BtnSelectedFont: TColor;
```

C++

```
__property TColor BtnSelectedFont;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.4 TCustomActionBarColorMap.Color Property

Specifies the background color for action bands.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

The Color property determines the background color. You can set Color to one of the constants defined in the Graphics (see page 14387) unit (such as clBlue), or to an explicit RGB integer value.

4.1.18.5.5 TCustomActionBarColorMap.DesignFocus Property

Changes the look and feel of your applications.

Pascal

```
property DesignFocus: TColor;
```

C++

```
__property TColor DesignFocus;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.6 TCustomActionBarColorMap.DisabledColor Property

Changes the look and feel of your applications.

Pascal

```
property DisabledColor: TColor;
```

C++

```
__property TColor DisabledColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.7 TCustomActionBarColorMap.DisabledFontColor Property

Changes the look and feel of your applications.

Pascal

```
property DisabledFontColor: TColor;
```

C++

```
__property TColor DisabledFontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.8 TCustomActionBarColorMap.DisabledFontShadow Property

Changes the look and feel of your applications.

Pascal

```
property DisabledFontShadow: TColor;
```

C++

```
__property TColor DisabledFontShadow;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.9 TCustomActionBarColorMap.FontColor Property

Changes the look and feel of your applications.

Pascal

```
property FontColor: TColor;
```

C++

```
__property TColor FontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.10 TCustomActionBarColorMap.FrameBottomRightInner Property

Changes the look and feel of your applications.

Pascal

```
property FrameBottomRightInner: TColor;
```

C++

```
__property TColor FrameBottomRightInner;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.11 TCustomActionBarColorMap.FrameBottomRightOuter Property

Changes the look and feel of your applications.

Pascal

```
property FrameBottomRightOuter: TColor;
```

C++

```
__property TColor FrameBottomRightOuter;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.12 TCustomActionBarColorMap.FrameTopLeftInner Property

Changes the look and feel of your applications.

Pascal

```
property FrameTopLeftInner: TColor;
```

C++

```
__property TColor FrameTopLeftInner;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.13 TCustomActionBarColorMap.FrameTopLeftOuter Property

Changes the look and feel of your applications.

Pascal

```
property FrameTopLeftOuter: TColor;
```

C++

```
__property TColor FrameTopLeftOuter;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.14 TCustomActionBarColorMap.HighlightColor Property

Changes the look and feel of your applications.

Pascal

```
property HighlightColor: TColor;
```

C++

```
__property TColor HighlightColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.15 TCustomActionBarColorMap.HotColor Property

Changes the look and feel of your applications.

Pascal

```
property HotColor: TColor;
```

C++

```
__property TColor HotColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.16 TCustomActionBarColorMap.HotFontColor Property

Changes the look and feel of your applications.

Pascal

```
property HotFontColor: TColor;
```

C++

```
__property TColor HotFontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.17 TCustomActionBarColorMap.MenuColor Property

Changes the look and feel of your applications.

Pascal

```
property MenuColor: TColor;
```

C++

```
__property TColor MenuColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.18 TCustomActionBarColorMap.SelectedColor Property

Changes the look and feel of your applications.

Pascal

```
property SelectedColor: TColor;
```

C++

```
__property TColor SelectedColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.19 TCustomActionBarColorMap.SelectedFontColor Property

Changes the look and feel of your applications.

Pascal

```
property SelectedFontColor: TColor;
```

C++

```
__property TColor SelectedFontColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.20 TCustomActionBarColorMap.ShadowColor Property

Changes the look and feel of your applications.

Pascal

```
property ShadowColor: TColor;
```

C++

```
__property TColor ShadowColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.5.21 TCustomActionBarColorMap.UnusedColor Property

Changes the look and feel of your applications.

Pascal

```
property UnusedColor: TColor;
```

C++

```
__property TColor UnusedColor;
```

Description

Use Object Inspector for the respective colormaps to change the default settings and customize the look and feel of your applications.

4.1.18.6 TCustomActionBarColorMap Events

The events of the TCustomActionBarColorMap class are listed here.

Public Events

	Name	Description
	OnColorChange (see page 506)	Occurs when one of the action's color properties changes.

4.1.18.6.1 TCustomActionBarColorMap.OnColorChange Event

Occurs when one of the action's color properties changes.

Pascal

```
property OnColorChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnColorChange;
```

Description

Applications can't use the protected OnChange event. It is used internally to manage the relationship between the properties of the action and the corresponding properties of the action's clients.

Component writers can use OnChange in descendant objects to respond when the action's properties change.

4.1.19 TCustomActionClass Class

Class Hierarchy

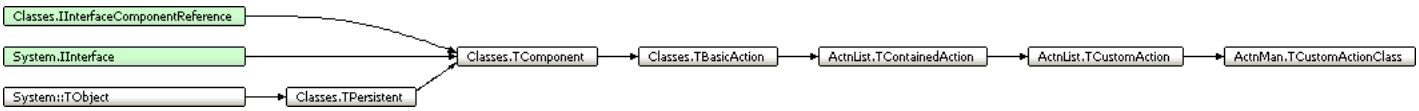

Pascal

```
TCustomActionClass = class(TCustomAction);
```

C++

```
class TCustomActionClass : public TCustomAction;
```

File

ActnMan

Description

This is class ActnMan.TCustomActionClass.

4.1.20 TCustomActionControl Class

TCustomActionControl is the base class for ActionBand custom controls.

Class Hierarchy

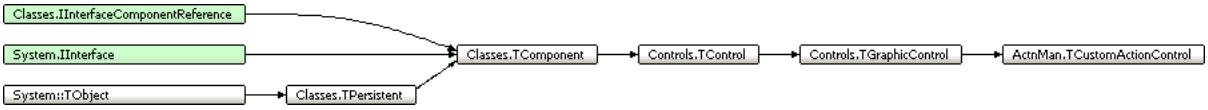

Pascal

```
TCustomActionControl = class(TGraphicControl);
```

C++

```
class TCustomActionControl : public TGraphicControl;
```

File

ActnMan

Description

TCustomActionControl is a base class for several ActionBand components. Most notably, the standard menu item (TStandardMenuItem) and menu button (TStandardMenuButton) derive from this class.

Do not create instances of TCustomActionControl. If you would like to replace the standard menu item and/or menu button behaviors, use the classes TCustomMenuItem and/or TCustomMenuButton as parent classes for your replacement classes.

See Also

TCustomMenuButton (see page 790)

TCustomMenuItem (see page 816)

TStandardMenuButton (see page 20444)

TStandardMenuItem (see page 20465)

4.1.20.1 TCustomActionControl Members

The following tables list the members exposed by TCustomActionControl.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	BeginAutoDrag (see page 517)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CalcLayout (see page 518)	This is CalcLayout, a member of class TCustomActionControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	CMDesignHitTest (see page 519)	This is CMDesignHitTest, a member of class TCustomActionControl.
	CMFontChanged (see page 519)	This is CMFontChanged, a member of class TCustomActionControl.
	CMItemDropPoint (see page 519)	This is CMItemDropPoint, a member of class TCustomActionControl.
	CMItemSelected (see page 519)	This is CMItemSelected, a member of class TCustomActionControl.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.

	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 519)	This is DesignWndProc, a member of class TCustomActionControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDragDrop (see page 520)	This is DoDragDrop, a member of class TCustomActionControl.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 520)	OnStartDrag event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 521)	OnDragOver event dispatcher.
	DrawBackground (see page 521)	Draws a background image on the control.
	DrawDesignFocus (see page 521)	This is DrawDesignFocus, a member of class TCustomActionControl.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawDragDropPoint (see page 522)	This is DrawDragDropPoint, a member of class TCustomActionControl.
	DrawGlyph (see page 522)	Draws a symbol on the control.
	DrawLargeGlyph (see page 522)	This is DrawLargeGlyph, a member of class TCustomActionControl.
	DrawSeparator (see page 522)	This is DrawSeparator, a member of class TCustomActionControl.
	DrawShadowedText (see page 523)	This is DrawShadowedText, a member of class TCustomActionControl.
	DrawText (see page 523)	Draws text on the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	GetAction (see page 523)	This is GetAction, a member of class TCustomActionControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetImageSize (see page 523)	This is GetImageSize, a member of class TCustomActionControl.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSeparator (see page 524)	This is GetSeparator, a member of class TCustomActionControl.
	GetShowCaption (see page 524)	This is GetShowCaption, a member of class TCustomActionControl.

	GetShowShortCut (see page 524)	This is GetShowShortCut, a member of class TCustomActionControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasGlyph (see page 524)	This is HasGlyph, a member of class TCustomActionControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsChecked (see page 525)	This is IsChecked, a member of class TCustomActionControl.
	IsGrouped (see page 525)	This is IsGrouped, a member of class TCustomActionControl.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Paint (see page 526)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionChanged (see page 526)	This is PositionChanged, a member of class TCustomActionControl.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetUsageData (see page 526)	This is ResetUsageData, a member of class TCustomActionControl.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActionClient (see page 526)	This is SetActionClient, a member of class TCustomActionControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropPoint (see page 527)	This is SetDropPoint, a member of class TCustomActionControl.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetGlyphLayout (see page 527)	This is SetGlyphLayout, a member of class TCustomActionControl.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelected (see page 527)	This is SetSelected, a member of class TCustomActionControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.

	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateSelection (see page 527)	This is UpdateSelection, a member of class TCustomActionControl.
	UpdateTextBounds (see page 527)	This is UpdateTextBounds, a member of class TCustomActionControl.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMSetText (see page 528)	This is WMSetText, a member of class TCustomActionControl.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GlyphPos (see page 532)	This is GlyphPos, a member of class TCustomActionControl.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnClick (see page 535)	Occurs when the user clicks the control.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.

	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
V	CalcBounds (see page 518)	Calculates the size of the control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Click (see page 518)	Executes the control's function in response to a mouse click.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 515)	Creates an instance of TCustomActionControl (see page 507).
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 516)	Destroys an instance of TCustomActionControl (see page 507).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
V	DragDrop (see page 520)	Performs actions and generates an OnDragDrop event.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.

	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 524)	Calls the action client's Update method if the control is associated with an action client.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
V	Keyed (see page 525)	Executes the control's function in response to a keyboard action.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.

R	ActionBar (see page 528)	The ActionBand object which contains the custom action control
	ActionClient (see page 529)	Refers to the control's action client
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Caption (see page 529)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DropPoint (see page 532)	This is DropPoint, a member of class TCustomActionControl.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GlyphLayout (see page 532)	Describes where the image should be on the control
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Margins (see page 532)	Describes the boundaries of the control for drawing purposes.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	Selected (see page 533)	Indicates whether the control is currently selected.
R	Separator (see page 533)	Indicates whether the control is a separator.
R	ShowCaption (see page 533)	Indicates whether the control's caption should be shown.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	ShowShortCut (see page 533)	Indicates whether the control's short cut should be shown.
	SmallIcon (see page 534)	Indicates whether the control's small icon should be shown.
	Spacing (see page 534)	Controls (see page 5711) the separation between the control's icon and caption.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TextBounds (see page 534)	A bounding rectangle for the control caption.
	Transparent (see page 534)	Shows whether the control is transparent or not.

	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

4.1.20.2 TCustomActionControl.Create Constructor

Creates an instance of TCustomActionControl (see page 507).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomActionControl(TComponent * AOwner);
```

Description

Calling Create constructs and initializes an instance of TCustomActionControl (see page 507). However, you should never attempt to instantiate a TCustomActionControl (see page 507). This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component that is responsible for freeing the TCustomActionControl (see page 507) instance. It becomes the value of the Owner property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner (see page 2807)

Create (see page 5731)

Destroy (see page 516)

4.1.20.3 TCustomActionControl.Destroy Destructor

Destroys an instance of TCustomActionControl (see page 507).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomActionControl();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the graphic control reference is not nil, and only then calls Destroy.

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Frees the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create (see page 515) method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

Create (see page 515)

Free (see page 22300)

4.1.20.4 TCustomActionControl Methods

The methods of the TCustomActionControl class are listed here.

Protected Methods

	Name	Description
V	BeginAutoDrag (see page 517)	Begin Automatic Drag.
V	CalcLayout (see page 518)	This is CalcLayout, a member of class TCustomActionControl.
	CMDesignHitTest (see page 519)	This is CMDesignHitTest, a member of class TCustomActionControl.
	CMFontChanged (see page 519)	This is CMFontChanged, a member of class TCustomActionControl.
	CMItemDropPoint (see page 519)	This is CMItemDropPoint, a member of class TCustomActionControl.
	CMItemSelected (see page 519)	This is CMItemSelected, a member of class TCustomActionControl.
V	DesignWndProc (see page 519)	This is DesignWndProc, a member of class TCustomActionControl.
V	DoDragDrop (see page 520)	This is DoDragDrop, a member of class TCustomActionControl.
V	DoStartDrag (see page 520)	OnStartDrag event dispatcher.
V	DragOver (see page 521)	OnDragOver event dispatcher.
V	DrawBackground (see page 521)	Draws a background image on the control.
V	DrawDesignFocus (see page 521)	This is DrawDesignFocus, a member of class TCustomActionControl.
	DrawDragDropPoint (see page 522)	This is DrawDragDropPoint, a member of class TCustomActionControl.
V	DrawGlyph (see page 522)	Draws a symbol on the control.
V	DrawLargeGlyph (see page 522)	This is DrawLargeGlyph, a member of class TCustomActionControl.

	DrawSeparator (see page 522)	This is DrawSeparator, a member of class TCustomActionControl.
	DrawShadowedText (see page 523)	This is DrawShadowedText, a member of class TCustomActionControl.
	DrawText (see page 523)	Draws text on the control.
	GetAction (see page 523)	This is GetAction, a member of class TCustomActionControl.
	GetImageSize (see page 523)	This is GetImageSize, a member of class TCustomActionControl.
	GetSeparator (see page 524)	This is GetSeparator, a member of class TCustomActionControl.
	GetShowCaption (see page 524)	This is GetShowCaption, a member of class TCustomActionControl.
	GetShowShortCut (see page 524)	This is GetShowShortCut, a member of class TCustomActionControl.
	HasGlyph (see page 524)	This is HasGlyph, a member of class TCustomActionControl.
	IsChecked (see page 525)	This is IsChecked, a member of class TCustomActionControl.
	IsGrouped (see page 525)	This is IsGrouped, a member of class TCustomActionControl.
	Paint (see page 526)	Render control surface.
	PositionChanged (see page 526)	This is PositionChanged, a member of class TCustomActionControl.
	ResetUsageData (see page 526)	This is ResetUsageData, a member of class TCustomActionControl.
	SetActionClient (see page 526)	This is SetActionClient, a member of class TCustomActionControl.
	SetDropPoint (see page 527)	This is SetDropPoint, a member of class TCustomActionControl.
	SetGlyphLayout (see page 527)	This is SetGlyphLayout, a member of class TCustomActionControl.
	SetSelected (see page 527)	This is SetSelected, a member of class TCustomActionControl.
	UpdateSelection (see page 527)	This is UpdateSelection, a member of class TCustomActionControl.
	UpdateTextBounds (see page 527)	This is UpdateTextBounds, a member of class TCustomActionControl.
	WMSetText (see page 528)	This is WMSetText, a member of class TCustomActionControl.

Public Methods

	Name	Description
	CalcBounds (see page 518)	Calculates the size of the control.
	Click (see page 518)	Executes the control's function in response to a mouse click.
	DragDrop (see page 520)	Performs actions and generates an OnDragDrop event.
	InitiateAction (see page 524)	Calls the action client's Update method if the control is associated with an action client.
	Keyed (see page 525)	Executes the control's function in response to a keyboard action.

4.1.20.4.1 TCustomActionControl.BeginAutoDrag Method

Begin Automatic Drag.

Pascal

```
procedure BeginAutoDrag; override;
```

C++

```
virtual __fastcall BeginAutoDrag();
```

Description

BeginAutoDrag is called automatically when a user drags a control whose DragMode is dmAutomatic. Do not call BeginAutoDrag in application code.

As implemented in TControl, BeginAutoDrag calls BeginDrag, passing in the global Mouse variable's DragImmediate and DragThreshold properties.

See Also

DragKind (see page 5819)

DragMode (see page 5819)

EndDrag (see page 5757)

Mouse (see page 6321)

DragImmediate (see page 6100)

DragThreshold (see page 6101)

Mouse (see page 6321)

4.1.20.4.2 TCustomActionControl.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; virtual;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as the presence of an icon, caption, and shortcuts. CalcBounds computes the boundaries of the control considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds (see page 5783)

4.1.20.4.3 TCustomActionControl.CalcLayout Method

Pascal

```
procedure CalcLayout; virtual;
```

C++

```
virtual __fastcall CalcLayout();
```

Description

This is CalcLayout, a member of class TCustomActionControl.

4.1.20.4.4 TCustomActionControl.Click Method

Executes the control's function in response to a mouse click.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

This override of the method inherited from TControl executes the control's function in response to a user's mouse click. Usage data, which is used to determine whether a control is used frequently enough to be shown by default, is updated as well.

See Also

Click (see page 5743)

UsageCount (see page 388)

Keyed (see page 525)

4.1.20.4.5 TCustomActionControl.CMDesignHitTest Method

Pascal

```
procedure CMDesignHitTest(var Message: TCMDesignHitTest); message CM_DESIGNHITTEST;
```

C++

```
__fastcall CMDesignHitTest(TCMDesignHitTest Message);
```

Description

This is CMDesignHitTest, a member of class TCustomActionControl.

4.1.20.4.6 TCustomActionControl.CMFontChanged Method

Pascal

```
procedure CMFontChanged(var Message: TMessage); message CM_FONTCHANGED;
```

C++

```
__fastcall CMFontChanged(TMessage Message);
```

Description

This is CMFontChanged, a member of class TCustomActionControl.

4.1.20.4.7 TCustomActionControl.CMItemDropPoint Method

Pascal

```
procedure CMItemDropPoint(var Message: TCMItemMsg); message CM_ITEMDROPPOINT;
```

C++

```
__fastcall CMItemDropPoint(TCMItemMsg Message);
```

Description

This is CMItemDropPoint, a member of class TCustomActionControl.

4.1.20.4.8 TCustomActionControl.CMItemSelected Method

Pascal

```
procedure CMItemSelected(var Message: TCMItemMsg); message CM_ITEMSELECTED;
```

C++

```
__fastcall CMItemSelected(TCMItemMsg Message);
```

Description

This is CMItemSelected, a member of class TCustomActionControl.

4.1.20.4.9 TCustomActionControl.DesignWndProc Method

Pascal

```
function DesignWndProc(var Message: TMessage): Boolean; override;
```

C++

```
virtual __fastcall Boolean DesignWndProc(TMessage Message);
```

Description

This is DesignWndProc, a member of class TCustomActionControl.

4.1.20.4.10 TCustomActionControl.DoDragDrop Method

Pascal

```
procedure DoDragDrop(DragObject: TObject; X: Integer; Y: Integer); virtual;
```

C++

```
virtual __fastcall DoDragDrop(TObject * DragObject, int X, int Y);
```

Description

This is DoDragDrop, a member of class TCustomActionControl.

4.1.20.4.11 TCustomActionControl.DoStartDrag Method

OnStartDrag event dispatcher.

Pascal

```
procedure DoStartDrag(var DragObject: TDragObject); override;
```

C++

```
virtual __fastcall DoStartDrag(TDragObject DragObject);
```

Description

DoStartDrag calls any event handler attached to the OnStartDrag event. When creating a component that descends from TControl, override DoStartDrag to provide other responses in addition to the inherited event-handler call when the user starts a drag-and-drop operation.

See Also

DoEndDrag (see page 5751)

OnStartDrag (see page 5871)

4.1.20.4.12 TCustomActionControl.DragDrop Method

Performs actions and generates an OnDragDrop event.

Pascal

```
procedure DragDrop(Source: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DragDrop(TObject * Source, int X, int Y);
```

Description

This override of the method inherited from TControl executes instructions and generates an OnDragDrop event. The method is called when a user drags and drops an item (Source) onto the custom action control at coordinates (X,Y).

See Also

DragDrop (see page 5754)

4.1.20.4.13 TCustomActionControl.DragOver Method

OnDragOver event dispatcher.

Pascal

```
procedure DragOver(Source: TObject; X: Integer; Y: Integer; State: TDragState; var Accept:
Boolean); override;
```

C++

```
virtual __fastcall DragOver(TObject * Source, int X, int Y, TDragState State, Boolean Accept);
```

Description

Override DragOver to add additional code that executes before the OnDragOver event handler is called.

DragOver sets the Accept parameter to true to indicate that the user can drop the dragged object on the control. It sets Accept to false to indicate that the user cannot drop the dragged object on the control.

The Source parameter is the object being dragged. The State parameter indicates how the dragged object is moving in relation to the control. X and Y indicate the current position of the mouse.

See Also

DragDrop (see page 5754)

OnDragOver (see page 5857)

4.1.20.4.14 TCustomActionControl.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); virtual;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph (see page 522)

DrawText (see page 523)

4.1.20.4.15 TCustomActionControl.DrawDesignFocus Method

Pascal

```
procedure DrawDesignFocus(var PaintRect: TRect); virtual;
```

C++

```
virtual __fastcall DrawDesignFocus(TRect PaintRect);
```

Description

This is DrawDesignFocus, a member of class TCustomActionControl.

4.1.20.4.16 TCustomActionControl.DrawDragDropPoint Method

Pascal

```
procedure DrawDragDropPoint;
```

C++

```
__fastcall DrawDragDropPoint();
```

Description

This is DrawDragDropPoint, a member of class TCustomActionControl.

4.1.20.4.17 TCustomActionControl.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); virtual;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground (see page 521)

DrawText (see page 523)

4.1.20.4.18 TCustomActionControl.DrawLargeGlyph Method

Pascal

```
procedure DrawLargeGlyph(Location: TPoint); virtual;
```

C++

```
virtual __fastcall DrawLargeGlyph(TPoint Location);
```

Description

This is DrawLargeGlyph, a member of class TCustomActionControl.

4.1.20.4.19 TCustomActionControl.DrawSeparator Method

Pascal

```
procedure DrawSeparator(const Offset: Integer); virtual;
```

C++

```
virtual __fastcall DrawSeparator(const int Offset);
```

Description

This is DrawSeparator, a member of class TCustomActionControl.

4.1.20.4.20 TCustomActionControl.DrawShadowedText Method

Pascal

```
procedure DrawShadowedText(Rect: TRect; Flags: Cardinal; Text: string; TextColor: TColor;
ShadowColor: TColor); virtual;
```

C++

```
virtual __fastcall DrawShadowedText(TRect Rect, unsigned Flags, AnsiString Text, TColor
TextColor, TColor ShadowColor);
```

Description

This is DrawShadowedText, a member of class TCustomActionControl.

4.1.20.4.21 TCustomActionControl.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: string); virtual;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, AnsiString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground (see page 521)

DrawGlyph (see page 522)

4.1.20.4.22 TCustomActionControl.GetAction Method

Pascal

```
function GetAction: TBasicAction; override;
```

C++

```
virtual __fastcall TBasicAction GetAction();
```

Description

This is GetAction, a member of class TCustomActionControl.

4.1.20.4.23 TCustomActionControl.GetImageSize Method

Pascal

```
function GetImageSize: TPoint; virtual;
```

C++

```
virtual __fastcall TPoint GetImageSize();
```

Description

This is GetImageSize, a member of class TCustomActionControl.

4.1.20.4.24 TCustomActionControl.GetSeparator Method

Pascal

```
function GetSeparator: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetSeparator();
```

Description

This is GetSeparator, a member of class TCustomActionControl.

4.1.20.4.25 TCustomActionControl.GetShowCaption Method

Pascal

```
function GetShowCaption: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetShowCaption();
```

Description

This is GetShowCaption, a member of class TCustomActionControl.

4.1.20.4.26 TCustomActionControl.GetShowShortCut Method

Pascal

```
function GetShowShortCut: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetShowShortCut();
```

Description

This is GetShowShortCut, a member of class TCustomActionControl.

4.1.20.4.27 TCustomActionControl.HasGlyph Method

Pascal

```
function HasGlyph: Boolean;
```

C++

```
__fastcall Boolean HasGlyph();
```

Description

This is HasGlyph, a member of class TCustomActionControl.

4.1.20.4.28 TCustomActionControl.InitiateAction Method

Calls the action client's Update method if the control is associated with an action client.

Pascal

```
procedure InitiateAction; override;
```

C++

```
virtual __fastcall InitiateAction();
```

Description

This method performs the same function as TControl's InitiateAction method. See the documentation for that method for more details.

See Also

InitiateAction (see page 5769)

4.1.20.4.29 TCustomActionControl.IsChecked Method

Pascal

```
function IsChecked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsChecked();
```

Description

This is IsChecked, a member of class TCustomActionControl.

4.1.20.4.30 TCustomActionControl.IsGrouped Method

Pascal

```
function IsGrouped: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsGrouped();
```

Description

This is IsGrouped, a member of class TCustomActionControl.

4.1.20.4.31 TCustomActionControl.Keyed Method

Executes the control's function in response to a keyboard action.

Pascal

```
procedure Keyed; virtual;
```

C++

```
virtual __fastcall Keyed();
```

Description

Use Keyed to execute the control's function in response to a user request from the keyboard, as opposed to a mouse click. Usage data, which is used to determine whether a control is used frequently enough to be shown by default, is updated as well.

See Also

Click (see page 518)

UsageCount (see page 388)

4.1.20.4.32 TCustomActionControl.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas (see page 6053)

4.1.20.4.33 TCustomActionControl.PositionChanged Method

Pascal

```
procedure PositionChanged; virtual;
```

C++

```
virtual __fastcall PositionChanged();
```

Description

This is PositionChanged, a member of class TCustomActionControl.

4.1.20.4.34 TCustomActionControl.ResetUsageData Method

Pascal

```
procedure ResetUsageData;
```

C++

```
__fastcall ResetUsageData();
```

Description

This is ResetUsageData, a member of class TCustomActionControl.

4.1.20.4.35 TCustomActionControl.SetActionClient Method

Pascal

```
procedure SetActionClient(Value: TActionClientItem); virtual;
```

C++

```
virtual __fastcall SetActionClient(TActionClientItem Value);
```

Description

This is SetActionClient, a member of class TCustomActionControl.

4.1.20.4.36 TCustomActionControl.SetDropPoint Method

Pascal

```
procedure SetDropPoint(Value: Boolean);
```

C++

```
__fastcall SetDropPoint(Boolean Value);
```

Description

This is SetDropPoint, a member of class TCustomActionControl.

4.1.20.4.37 TCustomActionControl.SetGlyphLayout Method

Pascal

```
procedure SetGlyphLayout(const Value: TButtonLayout); virtual;
```

C++

```
virtual __fastcall SetGlyphLayout(const TButtonLayout Value);
```

Description

This is SetGlyphLayout, a member of class TCustomActionControl.

4.1.20.4.38 TCustomActionControl.SetSelected Method

Pascal

```
procedure SetSelected(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetSelected(Boolean Value);
```

Description

This is SetSelected, a member of class TCustomActionControl.

4.1.20.4.39 TCustomActionControl.UpdateSelection Method

Pascal

```
procedure UpdateSelection;
```

C++

```
__fastcall UpdateSelection();
```

Description

This is UpdateSelection, a member of class TCustomActionControl.

4.1.20.4.40 TCustomActionControl.UpdateTextBounds Method

Pascal

```
procedure UpdateTextBounds;
```

C++

```
__fastcall UpdateTextBounds();
```

Description

This is UpdateTextBounds, a member of class TCustomActionControl.

4.1.20.4.41 TCustomActionControl.WMSetText Method

Pascal

```
procedure WMSetText(var Message: TWMSetText); message WM_SETTEXT;
```

C++

```
__fastcall WMSetText(TWMSetText Message);
```

Description

This is WMSetText, a member of class TCustomActionControl.

4.1.20.5 TCustomActionControl Properties

The properties of the TCustomActionControl class are listed here.

Protected Properties

	Name	Description
	GlyphPos (see page 532)	This is GlyphPos, a member of class TCustomActionControl.

Public Properties

	Name	Description
R	ActionBar (see page 528)	The ActionBand object which contains the custom action control
	ActionClient (see page 529)	Refers to the control's action client
	Caption (see page 529)	Specifies a text string that identifies the control to the user.
	DropPoint (see page 532)	This is DropPoint, a member of class TCustomActionControl.
	GlyphLayout (see page 532)	Describes where the image should be on the control
	Margins (see page 532)	Describes the boundaries of the control for drawing purposes.
	Selected (see page 533)	Indicates whether the control is currently selected.
R	Separator (see page 533)	Indicates whether the control is a separator.
R	ShowCaption (see page 533)	Indicates whether the control's caption should be shown.
R	ShowShortCut (see page 533)	Indicates whether the control's short cut should be shown.
	SmallIcon (see page 534)	Indicates whether the control's small icon should be shown.
	Spacing (see page 534)	Controls (see page 5711) the separation between the control's icon and caption.
	TextBounds (see page 534)	A bounding rectangle for the control caption.
	Transparent (see page 534)	Shows whether the control is transparent or not.

4.1.20.5.1 TCustomActionControl.ActionBar Property

The ActionBand object which contains the custom action control

Pascal

```
property ActionBar: TCustomActionBar;
```

C++

```
__property TCustomActionBar ActionBar;
```

Description

Custom action controls are objects on an ActionBand user interface object, such as a TActionMainMenuBar. Use ActionBar to

refer to that object.

See Also

TCustomActionBar (see page 447)

4.1.20.5.2 TCustomActionControl.ActionClient Property

Refers to the control's action client

Pascal

```
property ActionClient: TActionClientItem;
```

C++

```
__property TActionClientItem ActionClient;
```

Description

Use ActionClient to obtain a reference to a TCustomActionControl (see page 507)'s action client item. The action client item serves as an intermediary between the visible control and the action initiated by that control. See the documentation for the TActionClientItem (see page 374) object (referenced above) for more information.

See Also

TActionClientItem (see page 374)

4.1.20.5.3 TCustomActionControl.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
```

```
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
```

```
end;
```

4.1.20.5.4 TCustomActionControl.DropPoint Property

Pascal

```
property DropPoint: Boolean;
```

C++

```
__property Boolean DropPoint;
```

Description

This is DropPoint, a member of class TCustomActionControl.

4.1.20.5.5 TCustomActionControl.GlyphLayout Property

Describes where the image should be on the control

Pascal

```
property GlyphLayout: TButtonLayout;
```

C++

```
__property TButtonLayout GlyphLayout;
```

Description

GlyphLayout describes where the image should be placed on the control. The description is in terms of a TButtonLayout value; see the documentation for that type (referenced above) for more information.

See Also

TButtonLayout (see page 2393)

4.1.20.5.6 TCustomActionControl.GlyphPos Property

Pascal

```
property GlyphPos: TPoint;
```

C++

```
__property TPoint GlyphPos;
```

Description

This is GlyphPos, a member of class TCustomActionControl.

4.1.20.5.7 TCustomActionControl.Margins Property

Describes the boundaries of the control for drawing purposes.

Pascal

```
property Margins: TRect;
```

C++

```
__property TRect Margins;
```

Description

Margins is a TRect which describes the area of the control which can contain writing and/or drawings. The units of the TRect are

pixels.

4.1.20.5.8 TCustomActionControl.Selected Property

Indicates whether the control is currently selected.

Pascal

```
property Selected: Boolean;
```

C++

```
__property Boolean Selected;
```

Description

If a control is selected (by mouse click, keyboard or other action) Selected is true; otherwise it is false. One example of a situation where Selected is useful is when the control is to be redrawn, because generally the control should be drawn differently if it is selected.

4.1.20.5.9 TCustomActionControl.Separator Property

Indicates whether the control is a separator.

Pascal

```
property Separator: Boolean;
```

C++

```
__property Boolean Separator;
```

Description

Menus (see page 16053) and toolbars may include separators, which often appear as vertical or horizonal bars. Separators are also instances of TCustomActionControl (see page 507) or one of its descendants. Use Separator to see if the current control is a separator.

4.1.20.5.10 TCustomActionControl.ShowCaption Property

Indicates whether the control's caption should be shown.

Pascal

```
property ShowCaption: Boolean;
```

C++

```
__property Boolean ShowCaption;
```

Description

Read ShowCaption to determine whether or not the custom action control's caption, a text description of the control's function, should be shown.

4.1.20.5.11 TCustomActionControl.ShowShortCut Property

Indicates whether the control's short cut should be shown.

Pascal

```
property ShowShortCut: Boolean;
```

C++

```
__property Boolean ShowShortCut;
```

Description

Read ShowShortCut to determine whether or not the custom action control's short cut keystrokes should be shown.

4.1.20.5.12 TCustomActionControl.SmallIcon Property

Indicates whether the control's small icon should be shown.

Pascal

```
property SmallIcon: Boolean;
```

C++

```
__property Boolean SmallIcon;
```

Description

Set SmallIcon to true if you want the small icon associated with the control to be shown. Otherwise, set it to false.

4.1.20.5.13 TCustomActionControl.Spacing Property

Controls (see page 5711) the separation between the control's icon and caption.

Pascal

```
property Spacing: Integer;
```

C++

```
__property int Spacing;
```

Description

The value stored in Spacing is the number of pixels between the control's icon and caption. If either is absent from the control, the value of Spacing becomes irrelevant. When the control is drawn, the value used in calculations will be zero.

4.1.20.5.14 TCustomActionControl.TextBounds Property

A bounding rectangle for the control caption.

Pascal

```
property TextBounds: TRect;
```

C++

```
__property TRect TextBounds;
```

Description

TextBounds is a bounding rectangle for the control's caption.

4.1.20.5.15 TCustomActionControl.Transparent Property

Shows whether the control is transparent or not.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

If the action client has a background image to display, the control becomes transparent so that the background will display properly. Use Transparent to see if the control is transparent and, therefore, if its action client has a custom background.

See Also

Background (see page 370)

4.1.20.6 TCustomActionControl Events

The events of the TCustomActionControl class are listed here.

Public Events

	Name	Description
	OnClick (see page 535)	Occurs when the user clicks the control.

4.1.20.6.1 TCustomActionControl.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
```

```
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
```

```
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

4.1.21 TCustomActionManager Class

TCustomActionManager is the base class for the TActionManager (see page 432) component.

Class Hierarchy

Pascal

```
TCustomActionManager = class(TCustomActionList);
```

C++

```
class TCustomActionManager : public TCustomActionList;
```

File

ActnMan

Description

TActionManager (see page 432) is a component that allows you to manage actions and categories of actions as UI elements at design-time and runtime. The action manager editor is the user interface for working with actions and categories of actions. You can visually add, delete, and regroup these UI elements on the action band components by dragging and dropping them from the action manager editor or by moving them around after they have been dropped on an action band.

See Also

TActionManager (see page 432)

TActionToolBar (see page 38)

TActionMainMenuBar (see page 585)

4.1.21.1 TCustomActionManager Members

The following tables list the members exposed by TCustomActionManager.

Protected Events

	Name	Description
	OnChange (see page 326)	Occurs when a change occurs in the action list.
	OnExecute (see page 327)	Occurs when a client object triggers an action in the list.
	OnGetBarItemClass (see page 557)	Occurs before TActionManager (see page 432) creates TActionBars (see page 352) items.
	OnGetBarsClass (see page 558)	Occurs before TActionManager (see page 432) creates its ActionBars (see page 553) collection.
	OnGetClientItemClass (see page 558)	Occurs before TActionManager (see page 432) creates any ActionClientItems.
	OnGetClientsClass (see page 558)	Occurs when attempting to add items to an ActionClient.
	OnUpdate (see page 327)	Occurs when the application is idle so that the action list can update a specific action in the list.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 545)	Generates an OnChange event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CreateActionBars (see page 545)	This is CreateActionBars, a member of class TCustomActionManager.
V	DefineProperties (see page 545)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	FindActionBar (see page 547)	This is FindActionBar, a member of class TCustomActionManager.
	FindActionClient (see page 547)	This is FindActionClient, a member of class TCustomActionManager.
	FindClient (see page 547)	This is FindClient, a member of class TCustomActionManager.
	GetActionBarItemClass (see page 548)	This is GetActionBarItemClass, a member of class TCustomActionManager.
	GetActionBarsClass (see page 548)	This is GetActionBarsClass, a member of class TCustomActionManager.
	GetActionClientItemClass (see page 548)	This is GetActionClientItemClass, a member of class TCustomActionManager.
	GetActionClientsClass (see page 549)	This is GetActionClientsClass, a member of class TCustomActionManager.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
V	GetChildren (see page 321)	Calls a specified method for each child action of the action list.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	GetUsage (see page 549)	This is GetUsage, a member of class TCustomActionManager.
	HasLinkedActionLists (see page 549)	This is HasLinkedActionLists, a member of class TCustomActionManager.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 549)	Initializes the component after the form file has been read into memory.
	LoadMenu (see page 551)	This is LoadMenu, a member of class TCustomActionManager.
V	Notification (see page 551)	Responds to notifications when objects are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.

	RemoveAction (see page 323)	Removes an action from the list.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetChildOrder (see page 323)	Sets the order that actions are passed to a callback by the GetChildren (see page 321) method.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetImages (see page 552)	Sets the value of the Images property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetState (see page 553)	This is SetState, a member of class TCustomActionManager.
	SetupActionBars (see page 553)	This is SetupActionBars, a member of class TCustomActionManager.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Loading (see page 555)	This is Loading, a member of class TCustomActionManager.

Public Events

	Name	Description
	OnStateChange (see page 327)	Occurs after an OnChange (see page 326) event.
	OnStyleChanged (see page 558)	This is OnStyleChanged, a member of class TCustomActionManager.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddAction (see page 543)	Inserts a new action into an existing actionband.
	AddCategory (see page 543)	Adds all of the actions from a category to an action band.
	AddSeparator (see page 544)	Adds a separator to an action band.
V	AfterConstruction (see page 544)	Loads the layout information needed to reconstruct the action bands by the action manager from a file.
V	Assign (see page 544)	Provides a custom Assign procedure for TCustomActionManager (see page 537) objects.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 541)	Instantiates and initializes a TCustomActionManager (see page 537) object.

	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DeleteActionItems (see page 546)	Removes items from an ActionBars (see page 553) collection.
	DeleteItem (see page 546)	Deletes an action band item.
	Destroy (see page 542)	Disposes of the instance of an action manager object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 321)	Generates an OnExecute (see page 327) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindItemByAction (see page 547)	Returns the first item linked to an action.
	FindItemByCaption (see page 548)	Returns the first item with a matching caption.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 321)	Returns a TComponent enumerator.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsShortCut (see page 322)	Evaluates an input key combination to determine whether it is a shortcut for one of the actions in the list.
	LoadFromFile (see page 550)	Loads the layout information needed to reconstruct the action bands by the action manager from a file.
	LoadFromStream (see page 550)	Loads the layout information needed to reconstruct the action bands by the action manager from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	ResetActionBar (see page 551)	Resets the action bands to their default state.
	ResetUsageData (see page 551)	Resets the usage data of the action bands.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.

V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 552)	Saves layout information needed to reconstruct the action bands by the action manager to a file.
	SaveToStream (see page 552)	Saves layout information needed to reconstruct the action bands by the action manager to a stream.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	UpdateAction (see page 324)	Generates an OnUpdate (see page 327) event.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	ActionBars (see page 553)	Contains a collection of all the action bands in the application.
R	ActionCount (see page 324)	Indicates the number of actions in the action list.
	Actions (see page 325)	Lists the actions maintained by the action list.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	DefaultActionBars (see page 554)	Contains a collection of the application default action bands.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	FileName (see page 554)	Specifies the name of the file that the action manager component uses to read and write layout information.
	Images (see page 554)	Lists the images available for actions in the action list.
	LinkedActionLists (see page 555)	Contains a collection of action lists available from action bands and menus.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	PrioritySchedule (see page 555)	Specifies a schedule that indicates how long it takes to mark an item as unused.
	State (see page 326)	Indicates whether the actions in the action list respond when the user triggers a client object.
	Style (see page 556)	Identifies the style object that gives the application its default look and feel.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

4.1.21.2 TCustomActionManager.Create Constructor

Instantiates and initializes a TCustomActionManager (see page 537) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomActionManager(TComponent * AOwner);
```

Description

When using the form designer, action manager instances are created automatically when you drop an action manager component on your form. The AOwner parameter is the owner of the action manager, typically this is the form.

You should never attempt to instantiate a TCustomActionManager (see page 537) object in an application. Instead, instantiate

and use the descendant class, TActionManager (see page 432).

See Also

Destroy (see page 542)

4.1.21.3 TCustomActionManager.Destroy Destructor

Disposes of the instance of an action manager object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomActionManager();
```

Description

You do not need to call the destructor for an action manager if it has an owner. The action manager is destroyed automatically by its owner that was passed to its constructor when it was created. In situations where the custom action manager object needs to be manually destroyed, do so by calling its Free method, not this destructor.

See Also

Create (see page 541)

4.1.21.4 TCustomActionManager Methods

The methods of the TCustomActionManager class are listed here.

Protected Methods

	Name	Description
	Change (see page 545)	Generates an OnChange event.
	CreateActionBars (see page 545)	This is CreateActionBars, a member of class TCustomActionManager.
	DefineProperties (see page 545)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	FindActionBar (see page 547)	This is FindActionBar, a member of class TCustomActionManager.
	FindActionClient (see page 547)	This is FindActionClient, a member of class TCustomActionManager.
	FindClient (see page 547)	This is FindClient, a member of class TCustomActionManager.
	GetActionBarItemClass (see page 548)	This is GetActionBarItemClass, a member of class TCustomActionManager.
	GetActionBarsClass (see page 548)	This is GetActionBarsClass, a member of class TCustomActionManager.
	GetActionClientItemClass (see page 548)	This is GetActionClientItemClass, a member of class TCustomActionManager.
	GetActionClientsClass (see page 549)	This is GetActionClientsClass, a member of class TCustomActionManager.
	GetUsage (see page 549)	This is GetUsage, a member of class TCustomActionManager.
	HasLinkedActionLists (see page 549)	This is HasLinkedActionLists, a member of class TCustomActionManager.
	Loaded (see page 549)	Initializes the component after the form file has been read into memory.
	LoadMenu (see page 551)	This is LoadMenu, a member of class TCustomActionManager.
	Notification (see page 551)	Responds to notifications when objects are created or destroyed.
	SetImages (see page 552)	Sets the value of the Images property.
	SetState (see page 553)	This is SetState, a member of class TCustomActionManager.
	SetupActionBars (see page 553)	This is SetupActionBars, a member of class TCustomActionManager.

Public Methods

	Name	Description
	AddAction (see page 543)	Inserts a new action into an existing actionband.

	AddCategory (see page 543)	Adds all of the actions from a category to an action band.
	AddSeparator (see page 544)	Adds a separator to an action band.
V	AfterConstruction (see page 544)	Loads the layout information needed to reconstruct the action bands by the action manager from a file.
V	Assign (see page 544)	Provides a custom Assign procedure for TCustomActionManager (see page 537) objects.
	DeleteActionItems (see page 546)	Removes items from an ActionBars (see page 553) collection.
	DeleteItem (see page 546)	Deletes an action band item.
	FindItemByAction (see page 547)	Returns the first item linked to an action.
	FindItemByCaption (see page 548)	Returns the first item with a matching caption.
	LoadFromFile (see page 550)	Loads the layout information needed to reconstruct the action bands by the action manager from a file.
	LoadFromStream (see page 550)	Loads the layout information needed to reconstruct the action bands by the action manager from a stream.
	ResetActionBar (see page 551)	Resets the action bands to their default state.
	ResetUsageData (see page 551)	Resets the usage data of the action bands.
	SaveToFile (see page 552)	Saves layout information needed to reconstruct the action bands by the action manager to a file.
	SaveToStream (see page 552)	Saves layout information needed to reconstruct the action bands by the action manager to a stream.

4.1.21.4.1 TCustomActionManager.AddAction Method

Inserts a new action into an existing actionband.

Pascal

```
function AddAction(AnAction: TCustomAction; AClient: TActionClient; After: Boolean = True):
TActionClientItem;
```

C++

```
__fastcall TActionClientItem AddAction(TCustomAction AnAction, TActionClient AClient, Boolean
After = True);
```

Description

AddAction adds an action to an action band. Use this method to insert a new action into an existing actionband. Use FindItemByCaption (see page 548) and FindItemByAction (see page 547) methods to locate the position at which you want to insert the new item.

See Also

LoadFromStream (see page 550)

FileName (see page 554)

4.1.21.4.2 TCustomActionManager.AddCategory Method

Adds all of the actions from a category to an action band.

Pascal

```
function AddCategory(ACategory: string; AClient: TActionClient; After: Boolean = True):
TActionClientItem;
```

C++

```
__fastcall TActionClientItem AddCategory(AnsiString ACategory, TActionClient AClient, Boolean
After = True);
```

Description

AddCategory adds all of the actions from a category to an action band.

See Also

LoadFromStream (see page 550)

FileName (see page 554)

4.1.21.4.3 TCustomActionManager.AddSeparator Method

Adds a separator to an action band.

Pascal

```
function AddSeparator(AnItem: TActionClientItem; After: Boolean = True): TActionClientItem;
```

C++

```
__fastcall TActionClientItem AddSeparator(TActionClientItem AnItem, Boolean After = True);
```

Description

AddSeparator adds a separator to an action band. Use FindItemByCaption (see page 548) and FindItemByAction (see page 547) to locate the separator insertion point, either before or after the found item.

See Also

LoadFromStream (see page 550)

FileName (see page 554)

4.1.21.4.4 TCustomActionManager.AfterConstruction Method

Loads the layout information needed to reconstruct the action bands by the action manager from a file.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction reads the file specified in the FileName (see page 554) property of the action manager component and loads the layout information needed to reconstruct the action bands connected to the action manager.

See Also

LoadFromStream (see page 550)

FileName (see page 554)

4.1.21.4.5 TCustomActionManager.Assign Method

Provides a custom Assign procedure for TCustomActionManager (see page 537) objects.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use this override of the inherited TPersistent procedure to assign the properties of one TCustomActionManager (see page 537) object to another. The ActionBars (see page 553), Images (see page 554), PrioritySchedule (see page 555), and LinkedActionLists (see page 555) items of the current TCustomActionManager (see page 537) all refer to the same objects as in the Source. In the case of the FileName (see page 554) property, it becomes a copy of the Source property of the same name. If Source is not an action manager, the inherited procedure is called.

See Also

Assign (see page 2930)

4.1.21.4.6 TCustomActionManager.Change Method

Generates an OnChange event.

Pascal

```
procedure Change; override;
```

C++

```
virtual __fastcall Change();
```

Description

Change is called when a change, such as a modification to the categories of the actions in the list, occurs in the action list.

Change:

1. Invokes an OnChange handler, if one has been assigned.
2. Informs all actions in the list that something has changed so that they can update themselves.

See Also

OnChange (see page 326)

4.1.21.4.7 TCustomActionManager.CreateActionBars Method

Pascal

```
function CreateActionBars: TActionBars;
```

C++

```
__fastcall TActionBars CreateActionBars();
```

Description

This is CreateActionBars, a member of class TCustomActionManager.

4.1.21.4.8 TCustomActionManager.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor (see page 2839)

DefineBinaryProperty (see page 2837)

DefineProperties (see page 2931)

DefineProperty (see page 2838)

4.1.21.4.9 TCustomActionManager.DeleteActionItems Method

Removes items from an ActionBars (see page 553) collection.

Pascal

```
procedure DeleteActionItems(Actions: array of TCustomAction);
```

C++

```
__fastcall DeleteActionItems(array of TCustomAction Actions);
```

Description

DeleteActionItems removes items from the ActionBars (see page 553) collection which are linked to the Actions specified.

See Also

LoadFromStream (see page 550)

FileName (see page 554)

4.1.21.4.10 TCustomActionManager.DeleteItem Method

Deletes an action band item.

Pascal

```
procedure DeleteItem(Caption: string);
```

C++

```
__fastcall DeleteItem(AnsiString Caption);
```

Description

DeleteItem deletes an action band item after locating it based on its caption.

See Also

LoadFromStream (see page 550)

FileName (see page 554)

4.1.21.4.11 TCustomActionManager.FindActionBar Method

Pascal

```
function FindActionBar(ActionBar: TCustomActionBar; Clients: TActionClientsCollection = nil):
TActionClient;
```

C++

```
__fastcall TActionClient FindActionBar(TCustomActionBar ActionBar, TActionClientsCollection
Clients = nil);
```

Description

This is FindActionBar, a member of class TCustomActionManager.

4.1.21.4.12 TCustomActionManager.FindActionClient Method

Pascal

```
function FindActionClient(AnAction: TContainedAction; Clients: TActionClientsCollection =
nil): TActionClientItem;
```

C++

```
__fastcall TActionClientItem FindActionClient(TContainedAction AnAction,
TActionClientsCollection Clients = nil);
```

Description

This is FindActionClient, a member of class TCustomActionManager.

4.1.21.4.13 TCustomActionManager.FindClient Method

Pascal

```
procedure FindClient(AClient: TActionClient);
```

C++

```
__fastcall FindClient(TActionClient AClient);
```

Description

This is FindClient, a member of class TCustomActionManager.

4.1.21.4.14 TCustomActionManager.FindItemByAction Method

Returns the first item linked to an action.

Pascal

```
function FindItemByAction(Action: TCustomAction): TActionClientItem;
```

C++

```
__fastcall TActionClientItem FindItemByAction(TCustomAction Action);
```

Description

FindItemByAction takes an action and returns the first item that is linked to that action.

See Also

LoadFromStream (see page 550)

FileName (see page 554)

Creating Most Recently Used Lists

4.1.21.4.15 TCustomActionManager.FindItemByCaption Method

Returns the first item with a matching caption.

Pascal

```
function FindItemByCaption(ACaption: string): TActionClientItem;
```

C++

```
__fastcall TActionClientItem FindItemByCaption(AnsiString ACaption);
```

Description

FindItemByCaption takes a caption and returns the first item with a matching caption.

See Also

LoadFromStream (see page 550)

FileName (see page 554)

Creating Most Recently Used Lists

4.1.21.4.16 TCustomActionManager.GetActionBarItemClass Method

Pascal

```
function GetActionBarItemClass: TActionBarItemClass; dynamic;
```

C++

```
__fastcall TActionBarItemClass GetActionBarItemClass();
```

Description

This is GetActionBarItemClass, a member of class TCustomActionManager.

4.1.21.4.17 TCustomActionManager.GetActionBarsClass Method

Pascal

```
function GetActionBarsClass: TActionBarsClass; dynamic;
```

C++

```
__fastcall TActionBarsClass GetActionBarsClass();
```

Description

This is GetActionBarsClass, a member of class TCustomActionManager.

4.1.21.4.18 TCustomActionManager.GetActionClientItemClass Method

Pascal

```
function GetActionClientItemClass: TActionClientItemClass; dynamic;
```

C++

```
__fastcall TActionClientItemClass GetActionClientItemClass();
```

Description

This is GetActionClientItemClass, a member of class TCustomActionManager.

4.1.21.4.19 TCustomActionManager.GetActionClientsClass Method

Pascal

```
function GetActionClientsClass: TActionClientsClass; dynamic;
```

C++

```
__fastcall TActionClientsClass GetActionClientsClass();
```

Description

This is GetActionClientsClass, a member of class TCustomActionManager.

4.1.21.4.20 TCustomActionManager.GetUsage Method

Pascal

```
function GetUsage(AnItem: TActionClientItem): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetUsage(TActionClientItem AnItem);
```

Description

This is GetUsage, a member of class TCustomActionManager.

4.1.21.4.21 TCustomActionManager.HasLinkedActionLists Method

Pascal

```
function HasLinkedActionLists: Boolean;
```

C++

```
__fastcall Boolean HasLinkedActionLists();
```

Description

This is HasLinkedActionLists, a member of class TCustomActionManager.

4.1.21.4.22 TCustomActionManager.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the

streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState (see page 2804)

Create (see page 2766)

TReader (see page 2933)

4.1.21.4.23 TCustomActionManager.LoadFromFile Method

Loads the layout information needed to reconstruct the action bands by the action manager from a file.

Pascal

```
procedure LoadFromFile(const Filename: string);
```

C++

```
__fastcall LoadFromFile(const AnsiString Filename);
```

Description

LoadFromFile reads the file specified in the FileName (see page 554) property of the action manager component and loads the layout information needed to reconstruct the action bands connected to the action manager.

See Also

LoadFromStream (see page 550)

FileName (see page 554)

4.1.21.4.24 TCustomActionManager.LoadFromStream Method

Loads the layout information needed to reconstruct the action bands by the action manager from a stream.

Pascal

```
procedure LoadFromStream(S: TStream);
```

C++

```
__fastcall LoadFromStream(TStream S);
```

Description

LoadFromStream reads the stream specified in the Stream parameter and loads the layout information needed to reconstruct the action bands connected to the action manager contained in the stream.

See Also

LoadFromFile (see page 550)

TStream (see page 2973)

4.1.21.4.25 TCustomActionManager.LoadMenu Method

Pascal

```
procedure LoadMenu(Clients: TActionClients; AMenu: TMenuItem);
```

C++

```
__fastcall LoadMenu(TActionClients Clients, TMenuItem AMenu);
```

Description

This is LoadMenu, a member of class TCustomActionManager.

4.1.21.4.26 TCustomActionManager.Notification Method

Responds to notifications when objects are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification notifies the action list that the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

TCustomActionList overrides Notification to update the Images property when images from the list are destroyed and the Actions property when actions in the list are destroyed.

See Also

Images (see page 325)

4.1.21.4.27 TCustomActionManager.ResetActionBar Method

Resets the action bands to their default state.

Pascal

```
procedure ResetActionBar(Index: Integer);
```

C++

```
__fastcall ResetActionBar(int Index);
```

Description

Use ResetActionBar to return the action bands to their default state, which is stored in the property DefaultActionBars (see page 554).

See Also

DefaultActionBars (see page 554)

4.1.21.4.28 TCustomActionManager.ResetUsageData Method

Resets the usage data of the action bands.

Pascal

```
procedure ResetUsageData;
```

C++

```
__fastcall ResetUsageData();
```

Description

TCustomActionManager (see page 537) keeps a file of usage data whose name is FileName (see page 554). Usage data determines which items on the ActionBars (see page 553) are displayed at any given time. Call ResetUsageData to delete that file. The user interface will appear in the same state it was in when the program was run for the first time.

4.1.21.4.29 TCustomActionManager.SaveToFile Method

Saves layout information needed to reconstruct the action bands by the action manager to a file.

Pascal

```
procedure SaveToFile(const Filename: string);
```

C++

```
__fastcall SaveToFile(const AnsiString Filename);
```

Description

SaveToFile saves the layout information to a file specified in the FileName (see page 554) property. The action manager uses the layout information in the file to reconstruct the action bands.

4.1.21.4.30 TCustomActionManager.SaveToStream Method

Saves layout information needed to reconstruct the action bands by the action manager to a stream.

Pascal

```
procedure SaveToStream(S: TStream);
```

C++

```
__fastcall SaveToStream(TStream S);
```

Description

SaveToStream saves layout information to a stream specified in the Stream parameter. The action manager uses the layout information in the stream to reconstruct the action bands.

4.1.21.4.31 TCustomActionManager.SetImages Method

Sets the value of the Images property.

Pascal

```
procedure SetImages(Value: TCustomImageList); override;
```

C++

```
virtual __fastcall SetImages(TCustomImageList Value);
```

Description

SetImages is the protected write implementation of the Images property. It registers the action list with the image list specified by Value so that it will be informed of any changes to the image list.

See Also

Images (see page 325)

4.1.21.4.32 TCustomActionManager.SetState Method

Pascal

```
procedure SetState(const Value: TActionListState); override;
```

C++

```
virtual __fastcall SetState(const TActionListState Value);
```

Description

This is SetState, a member of class TCustomActionManager.

4.1.21.4.33 TCustomActionManager.SetupActionBars Method

Pascal

```
procedure SetupActionBars(ActionBars: TActionClientsCollection);
```

C++

```
__fastcall SetupActionBars(TActionClientsCollection ActionBars);
```

Description

This is SetupActionBars, a member of class TCustomActionManager.

4.1.21.5 TCustomActionManager Properties

The properties of the TCustomActionManager class are listed here.

Protected Properties

	Name	Description
R	Loading (see page 555)	This is Loading, a member of class TCustomActionManager.

Public Properties

	Name	Description
	ActionBars (see page 553)	Contains a collection of all the action bands in the application.
R	DefaultActionBars (see page 554)	Contains a collection of the application default action bands.
	FileName (see page 554)	Specifies the name of the file that the action manager component uses to read and write layout information.
	Images (see page 554)	Lists the images available for actions in the action list.
	LinkedActionLists (see page 555)	Contains a collection of action lists available from action bands and menus.
	PrioritySchedule (see page 555)	Specifies a schedule that indicates how long it takes to mark an item as unused.
	Style (see page 556)	Identifies the style object that gives the application its default look and feel.

4.1.21.5.1 TCustomActionManager.ActionBars Property

Contains a collection of all the action bands in the application.

Pascal

```
property ActionBars: TActionBars;
```

C++

```
__property TActionBars ActionBars;
```

Description

ActionBars is used internally by the action manager component to maintain a collection of all the action bands connected to the action manager in the application. An action band is automatically added to the collection when an action or category of actions is dropped on an action band or the ActionManager property in an action band component is set in the Object Inspector.

See Also

DefaultActionBars (see page 554)

4.1.21.5.2 TCustomActionManager.DefaultActionBars Property

Contains a collection of the application default action bands.

Pascal

```
property DefaultActionBars: TActionBars;
```

C++

```
__property TActionBars DefaultActionBars;
```

Description

DefaultActionBars is a collection of action bands similar to ActionBars (see page 553). These bands are the default action bands, and are not user interface elements in the application. Instead, they are used by the method ResetActionBar (see page 551) to restore the user's action bands to their default values.

See Also

ActionBars (see page 553)

ResetActionBar (see page 551)

4.1.21.5.3 TCustomActionManager.FileName Property

Specifies the name of the file that the action manager component uses to read and write layout information.

Pascal

```
property FileName: TFileName;
```

C++

```
__property TFileName FileName;
```

Description

Use FileName to specify the name of the file that the action manager will use to read and write layout information for the action bands that are connected to the action manager. The action manager uses this file to store the current state of the action bands that are connected to the action manager when the application is closed. When an application is started, the action manager uses the layout information contained in this file to reconstruct the action bands back to the state when the application last closed. All customizations made to the action bands are saved in this file.

4.1.21.5.4 TCustomActionManager.Images Property

Lists the images available for actions in the action list.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Images is a list of bitmaps that can be displayed to the left of any action in the action list. An action is associated with an image from this image list by its ImageIndex property. This image can then appear on client controls such as menu items, tool buttons, and so on.

See Also

Actions (see page 325)

ImageIndex (see page 313)

4.1.21.5.5 TCustomActionManager.LinkedActionLists Property

Contains a collection of action lists available from action bands and menus.

Pascal

```
property LinkedActionLists: TActionListCollection;
```

C++

```
__property TActionListCollection LinkedActionLists;
```

Description

LinkedActionLists is a linked list of TActionListCollection (see page 425) objects. LinkedActionLists are simple data structures with two fields. One field is a TCustomActionList object, which is a list of possible actions. The other field is a caption, which defines the action list's description appearing on the action band or menu.

Use LinkedActionLists to merge in action lists from other sources outside the action manager. Those action lists will become accessible from action bands and menus. Application users can exploit this feature, for example, to reorganize items from a traditional main menu bar.

See Also

TActionListCollection (see page 425)

4.1.21.5.6 TCustomActionManager.Loading Property

Pascal

```
property Loading: Boolean;
```

C++

```
__property Boolean Loading;
```

Description

This is Loading, a member of class TCustomActionManager.

4.1.21.5.7 TCustomActionManager.PrioritySchedule Property

Specifies a schedule that indicates how long it takes to mark an item as unused.

Pascal

```
property PrioritySchedule: TStringList;
```

C++

```
__property TStringList PrioritySchedule;
```

Description

Based on usage statistics and layout space, the action manager uses PrioritySchedule to determine when a UI element representing a TActionClientItem (see page 374) is hidden or displayed on an action band. (This is not the same as the TActionClientItem (see page 374)'s visibility, as set by the TActionClientItem (see page 374)'s Visible property.)

To determine when to display or hide a UI element that is rendered by the action band for a specific TActionClientItem (see page 374), the action manager maintains a total count for each item of the total number of sessions, the session number it was used in, and the current session number. The number of sessions is defined as the number of times the application has been launched.

To determine if an item should be displayed, the program looks up the value of UsageCount in the left hand column in PrioritySchedule. If UsageCount is greater than the maximum tabulated value, that maximum is used for calculation purposes. The corresponding value in the right hand column is the number of sessions an item can be unused before it becomes hidden.

An action band will always show as many controls as it has space to show; the action band will wrap rows or columns, if necessary. TActionClientItems with their UsageCount property set to -1, will always be shown. You can use the UsageCount property to ensure that specific action band controls are always shown.

The default PrioritySchedule is as follows:

Number of sessions in which the action band item was used	Number of sessions the item will remain unhidden after its last use
0, 1	3
2	6
3	9
4, 5	12
6 – 8	17
9 – 13	23
14 – 24	29
25 or more	31

See Also

SessionCount (see page 358)

LastSession (see page 385)

UsageCount (see page 388)

4.1.21.5.8 TCustomActionManager.Style Property

Identifies the style object that gives the application its default look and feel.

Pascal

```
property Style: TActionBarStyle;
```

C++

```
__property TActionBarStyle Style;
```

Description

Use Style to access the application's Style object. This object manages which type of controls are used on the action bands to present a particular thematic look. To change the overall appearance of action bands in your application, change the style property accordingly. Delphi includes two styles by default: Standard and XP style. You can further augment these styles by using colormap components to control the colors that render the different style of controls.

To register new styles create a descendant of TActionBarStyleEx and register it using RegisterActnBarStyle.

See Also

TActionBarStyleEx (see page 583)

TStandardColorMap (see page 2)

TXPColorMap (see page 23)

RegisterActnBarStyle (see page 837)

4.1.21.6 TCustomActionManager Events

The events of the TCustomActionManager class are listed here.

Protected Events

	Name	Description
	OnGetBarItemClass (see page 557)	Occurs before TActionManager (see page 432) creates TActionBars (see page 352) items.
	OnGetBarsClass (see page 558)	Occurs before TActionManager (see page 432) creates its ActionBars (see page 553) collection.
	OnGetClientItemClass (see page 558)	Occurs before TActionManager (see page 432) creates any ActionClientItems.
	OnGetClientsClass (see page 558)	Occurs when attempting to add items to an ActionClient.

Public Events

	Name	Description
	OnStyleChanged (see page 558)	This is OnStyleChanged, a member of class TCustomActionManager.

4.1.21.6.1 TCustomActionManager.OnGetBarItemClass Event

Occurs before TActionManager (see page 432) creates TActionBars (see page 352) items.

Pascal

```
property OnGetBarItemClass: TGetBarItemClassEvent;
```

C++

```
__property TGetBarItemClassEvent OnGetBarItemClass;
```

Description

Write an OnGetBarItemClass event handler to change the type of the class used for the ActionBarItem. The OnGetBarItemClass event occurs after a find action fetches the item's class, but before the class is sent to the action client. OnGetBarItemClass provides an opportunity to change the class used for the items within the TActionBars (see page 352) collection.

4.1.21.6.2 TCustomActionManager.OnGetBarsClass Event

Occurs before TActionManager (see page 432) creates its ActionBars (see page 553) collection.

Pascal

```
property OnGetBarsClass: TGetBarsClassEvent;
```

C++

```
__property TGetBarsClassEvent OnGetBarsClass;
```

Description

Write an OnGetBarsClass event handler to change the type of the TActionBars (see page 352) collection to a descedant of TActionBars (see page 352). OnGetBarsClass provides an opportunity to change the class used for the TActionBars (see page 352) collection. By implementing this event and returning a descendant of TActionBars (see page 352), the developer can add new properties and new functionality to the collection for use within action bands.

4.1.21.6.3 TCustomActionManager.OnGetClientItemClass Event

Occurs before TActionManager (see page 432) creates any ActionClientItems.

Pascal

```
property OnGetClientItemClass: TGetClientItemClassEvent;
```

C++

```
__property TGetClientItemClassEvent OnGetClientItemClass;
```

Description

Write an OnGetClientItemClass event handler to change the type of the TactionClientItem class that is created and used by the ActionManager. OnGetClientItemClass provides an opportunity to return a descendant of TactionClientItem, which has new properties, events and methods for use within action bands.

4.1.21.6.4 TCustomActionManager.OnGetClientsClass Event

Occurs when attempting to add items to an ActionClient.

Pascal

```
property OnGetClientsClass: TGetClientsClassEvent;
```

C++

```
__property TGetClientsClassEvent OnGetClientsClass;
```

Description

Write an OnGetClientsClass event handler to change the type of the TactionClients class that is created and used by the ActionManager. OnGetClientsClass provides an opportunity to return a descendant of TActionClients (see page 400) which implements new properties, events and methods for use within action bands.

4.1.21.6.5 TCustomActionManager.OnStyleChanged Event

Pascal

```
property OnStyleChanged: TStyleChanged;
```

C++

```
__property TStyleChanged OnStyleChanged;
```

Description

This is OnStyleChanged, a member of class TCustomActionManager.

4.1.22 TXToolDockForm Class

Class Hierarchy

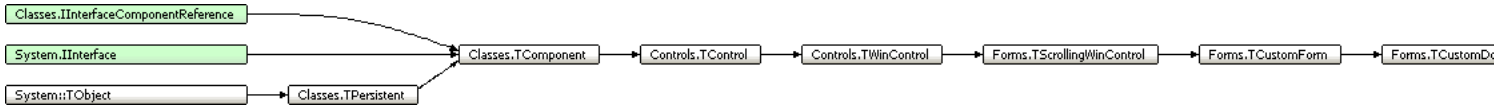

Pascal

```
TXToolDockForm = class(TToolDockForm);
```

C++

```
class TXToolDockForm : public TToolDockForm;
```

File

ActnMan

Description

This is class ActnMan.TXToolDockForm.

4.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IActionBarDesigner (see page 560)	This is class ActnMan.IActionBarDesigner.

4.2.1 IActionBarDesigner Interface

Class Hierarchy

ActnMan.IActionBarDesigner

Pascal

```
IActionBarDesigner = interface;
```

C++

```
__interface IActionBarDesigner;
```

File

ActnMan

Description

This is class ActnMan.IActionBarDesigner.

4.2.1.1 IActionBarDesigner Members

The following tables list the members exposed by IActionBarDesigner.

Public Methods

	Name	Description
	CreateAction (see page 561)	This is CreateAction, a member of class IActionBarDesigner.
	EditAction (see page 561)	This is EditAction, a member of class IActionBarDesigner.
	Modified (see page 561)	This is Modified, a member of class IActionBarDesigner.
	SetActiveMenu (see page 561)	This is SetActiveMenu, a member of class IActionBarDesigner.
	SetItemSelection (see page 562)	This is SetItemSelection, a member of class IActionBarDesigner.
	SetSelection (see page 562)	This is SetSelection, a member of class IActionBarDesigner.

4.2.1.2 IActionBarDesigner Methods

The methods of the IActionBarDesigner class are listed here.

Public Methods

	Name	Description
	CreateAction (see page 561)	This is CreateAction, a member of class IActionBarDesigner.

	EditAction (see page 561)	This is EditAction, a member of class IActionBarDesigner.
	Modified (see page 561)	This is Modified, a member of class IActionBarDesigner.
	SetActiveMenu (see page 561)	This is SetActiveMenu, a member of class IActionBarDesigner.
	SetItemSelection (see page 562)	This is SetItemSelection, a member of class IActionBarDesigner.
	SetSelection (see page 562)	This is SetSelection, a member of class IActionBarDesigner.

4.2.1.2.1 IActionBarDesigner.CreateAction Method

Pascal

```
function CreateAction(AnActionClass: TContainedActionClass): TContainedAction;
```

C++

```
__fastcall TContainedAction CreateAction(TContainedActionClass AnActionClass);
```

Description

This is CreateAction, a member of class IActionBarDesigner.

4.2.1.2.2 IActionBarDesigner.EditAction Method

Pascal

```
procedure EditAction(Action: TContainedAction);
```

C++

```
__fastcall EditAction(TContainedAction Action);
```

Description

This is EditAction, a member of class IActionBarDesigner.

4.2.1.2.3 IActionBarDesigner.Modified Method

Pascal

```
procedure Modified(ActionBar: TCustomActionBar);
```

C++

```
__fastcall Modified(TCustomActionBar ActionBar);
```

Description

This is Modified, a member of class IActionBarDesigner.

4.2.1.2.4 IActionBarDesigner.SetActiveMenu Method

Pascal

```
procedure SetActiveMenu(Menu: TCustomActionBar);
```

C++

```
__fastcall SetActiveMenu(TCustomActionBar Menu);
```

Description

This is SetActiveMenu, a member of class IActionBarDesigner.

4.2.1.2.5 IActionBarDesigner.SetItemSelection Method

Pascal

```
procedure SetItemSelection(const Items: array of TActionClient);
```

C++

```
__fastcall SetItemSelection(const array of TActionClient Items);
```

Description

This is SetItemSelection, a member of class IActionBarDesigner.

4.2.1.2.6 IActionBarDesigner.SetSelection Method

Pascal

```
procedure SetSelection(APersistent: TPersistent);
```

C++

```
__fastcall SetSelection(TPersistent * APersistent);
```

Description

This is SetSelection, a member of class IActionBarDesigner.

4.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	NotifyDesigner (see page 563)	This is function ActnMan.NotifyDesigner.

4.3.1 ActnMan.NotifyDesigner Function

Pascal

```
procedure NotifyDesigner(ActionBar: TCustomActionBar);
```

C++

```
NotifyDesigner(TCustomActionBar ActionBar);
```

File

ActnMan

Description

This is function ActnMan.NotifyDesigner.

4.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TBackgroundLayout (see page 564)	This is record ActnMan.TBackgroundLayout.
	TBarEdge (see page 564)	This is record ActnMan.TBarEdge.
	TBarOrientation (see page 565)	This is record ActnMan.TBarOrientation.
	TCaptionOptions (see page 565)	This is record ActnMan.TCaptionOptions.
	TChangesAllowed (see page 566)	This is record ActnMan.TChangesAllowed.

Records

	Name	Description
	TCMItemMsg (see page 566)	This is record ActnMan.TCMItemMsg.

4.4.1 ActnMan.TBackgroundLayout Enumeration

Pascal

```
TBackgroundLayout = (
  blNormal,
  blStretch,
  blTile,
  blLeftBanner,
  blRightBanner
);
```

C++

```
enum TBackgroundLayout {
  blNormal,
  blStretch,
  blTile,
  blLeftBanner,
  blRightBanner
};
```

File

ActnMan

Description

This is record ActnMan.TBackgroundLayout.

4.4.2 ActnMan.TBarEdge Enumeration

Pascal

```
TBarEdge = (
  beLeft,
```

```
    beRight,
    beEither
  );
```

C++

```
  enum TBarEdge {
    beLeft,
    beRight,
    beEither
  };
```

File

ActnMan

Description

This is record ActnMan.TBarEdge.

4.4.3 ActnMan.TBarOrientation Enumeration

Pascal

```
  TBarOrientation = (
    boLeftToRight,
    boRightToLeft,
    boTopToBottom,
    boBottomToTop
  );
```

C++

```
  enum TBarOrientation {
    boLeftToRight,
    boRightToLeft,
    boTopToBottom,
    boBottomToTop
  };
```

File

ActnMan

Description

This is record ActnMan.TBarOrientation.

4.4.4 ActnMan.TCaptionOptions Enumeration

Pascal

```
  TCaptionOptions = (
    coNone,
    coSelective,
    coAll
  );
```

C++

```
  enum TCaptionOptions {
    coNone,
```

```
    coSelective,
    coAll
  };
```

File

ActnMan

Description

This is record ActnMan.TCaptionOptions.

4.4.5 ActnMan.TChangesAllowed Enumeration

Pascal

```
  TChangesAllowed = (
    caModify,
    caMove,
    caDelete
  );
```

C++

```
  enum TChangesAllowed {
    caModify,
    caMove,
    caDelete
  };
```

File

ActnMan

Description

This is record ActnMan.TChangesAllowed.

4.4.6 ActnMan.TCMItemMsg Record

Pascal

```
  TCMItemMsg = record
    Msg: Cardinal;
    Unused: Integer;
    Sender: TCustomActionControl;
    Result: Longint;
  end;
```

C++

```
  struct TCMItemMsg {
    unsigned Msg;
    int Unused;
    TCustomActionControl Sender;
    Longint Result;
  };
```

File

ActnMan

Description

This is record ActnMan.TCMItemMsg.

4.5 Types

The following table lists types in this documentation.

Types

Name	Description
TActionBarItemClass (see page 568)	This is type ActnMan.TActionBarItemClass.
TActionBarsClass (see page 568)	This is type ActnMan.TActionBarsClass.
TActionClientClass (see page 569)	This is type ActnMan.TActionClientClass.
TActionClientItemClass (see page 569)	This is type ActnMan.TActionClientItemClass.
TActionClientLinkClass (see page 569)	This is type ActnMan.TActionClientLinkClass.
TActionClientsClass (see page 570)	This is type ActnMan.TActionClientsClass.
TActionProc (see page 570)	This is type ActnMan.TActionProc.
TChangesAllowedSet (see page 570)	This is type ActnMan.TChangesAllowedSet.
TColorMapUpdateProc (see page 571)	This is type ActnMan.TColorMapUpdateProc.
TControlCreatedEvent (see page 571)	This is type ActnMan.TControlCreatedEvent.
TCustomActionBarClass (see page 571)	This is type ActnMan.TCustomActionBarClass.
TCustomActionControlClass (see page 572)	This is type ActnMan.TCustomActionControlClass.
TCustomColorMapClass (see page 572)	This is type ActnMan.TCustomColorMapClass.
TGetBarItemClassEvent (see page 572)	This is type ActnMan.TGetBarItemClassEvent.
TGetBarsClassEvent (see page 573)	This is type ActnMan.TGetBarsClassEvent.
TGetClientItemClassEvent (see page 573)	This is type ActnMan.TGetClientItemClassEvent.
TGetClientsClassEvent (see page 573)	This is type ActnMan.TGetClientsClassEvent.
TGetControlClassEvent (see page 574)	This is type ActnMan.TGetControlClassEvent.
TMergeRange (see page 574)	This is type ActnMan.TMergeRange.
TStyleChanged (see page 574)	This is type ActnMan.TStyleChanged.

4.5.1 ActnMan.TActionBarItemClass Type

Pascal

```
TActionBarItemClass = class of TActionBarItem;
```

C++

```
TActionBarItem TActionBarItemClass;
```

File

ActnMan

Description

This is type ActnMan.TActionBarItemClass.

4.5.2 ActnMan.TActionBarsClass Type

Pascal

```
TActionBarsClass = class of TActionBars;
```

C++

```
TActionBars TActionBarsClass;
```

File

ActnMan

Description

This is type ActnMan.TActionBarsClass.

4.5.3 ActnMan.TActionClientClass Type

Pascal

```
TActionClientClass = class of TActionClient;
```

C++

```
TActionClient TActionClientClass;
```

File

ActnMan

Description

This is type ActnMan.TActionClientClass.

4.5.4 ActnMan.TActionClientItemClass Type

Pascal

```
TActionClientItemClass = class of TActionClientItem;
```

C++

```
TActionClientItem TActionClientItemClass;
```

File

ActnMan

Description

This is type ActnMan.TActionClientItemClass.

4.5.5 ActnMan.TActionClientLinkClass Type

Pascal

```
TActionClientLinkClass = class of TActionClientLink;
```

C++

```
TActionClientLink TActionClientLinkClass;
```

File

ActnMan

Description

This is type ActnMan.TActionClientLinkClass.

4.5.6 ActnMan.TActionClientsClass Type

Pascal

```
TActionClientsClass = class of TActionClients;
```

C++

```
TActionClients TActionClientsClass;
```

File

ActnMan

Description

This is type ActnMan.TActionClientsClass.

4.5.7 ActnMan.TActionProc Type

Pascal

```
TActionProc = procedure (AClient: TActionClient) of object;
```

C++

```
(AClient: TActionClient) ( TActionProc)();
```

File

ActnMan

Description

This is type ActnMan.TActionProc.

4.5.8 ActnMan.TChangesAllowedSet Type

Pascal

```
TChangesAllowedSet = set of TChangesAllowed;
```

C++

```
TChangesAllowed TChangesAllowedSet;
```

File

ActnMan

Description

This is type ActnMan.TChangesAllowedSet.

4.5.9 ActnMan.TColorMapUpdateProc Type

Pascal

```
TColorMapUpdateProc = procedure (ActionBar: TCustomActionBar; Colors:
TCustomActionBarColorMap);
```

C++

```
(ActionBar: TCustomActionBar; Colors: TCustomActionBarColorMap) ( TColorMapUpdateProc)();
```

File

ActnMan

Description

This is type ActnMan.TColorMapUpdateProc.

4.5.10 ActnMan.TControlCreatedEvent Type

Pascal

```
TControlCreatedEvent = procedure (Sender: TObject; var Control: TCustomActionControl) of
object;
```

C++

```
(Sender: TObject; var Control: TCustomActionControl) ( TControlCreatedEvent)();
```

File

ActnMan

Description

This is type ActnMan.TControlCreatedEvent.

4.5.11 ActnMan.TCustomActionBarClass Type

Pascal

```
TCustomActionBarClass = class of TCustomActionBar;
```

C++

```
TCustomActionBar TCustomActionBarClass;
```

File

ActnMan

Description

This is type ActnMan.TCustomActionBarClass.

4.5.12 ActnMan.TCustomActionControlClass Type

Pascal

```
TCustomActionControlClass = class of TCustomActionControl;
```

C++

```
TCustomActionControl TCustomActionControlClass;
```

File

ActnMan

Description

This is type ActnMan.TCustomActionControlClass.

4.5.13 ActnMan.TCustomColorMapClass Type

Pascal

```
TCustomColorMapClass = class of TCustomActionBarColorMap;
```

C++

```
TCustomActionBarColorMap TCustomColorMapClass;
```

File

ActnMan

Description

This is type ActnMan.TCustomColorMapClass.

4.5.14 ActnMan.TGetBarItemClassEvent Type

Pascal

```
TGetBarItemClassEvent = procedure (Sender: TCustomActionManager; var BarItemClass:
TActionBarItemClass) of object;
```

C++

```
(Sender: TCustomActionManager; var BarItemClass: TActionBarItemClass) (
TGetBarItemClassEvent)();
```

File

ActnMan

Description

This is type ActnMan.TGetBarItemClassEvent.

4.5.15 ActnMan.TGetBarsClassEvent Type

Pascal

```
TGetBarsClassEvent = procedure (Sender: TCustomActionManager; BarsClass: TActionBarsClass) of
object;
```

C++

```
(Sender: TCustomActionManager; BarsClass: TActionBarsClass) ( TGetBarsClassEvent)();
```

File

ActnMan

Description

This is type ActnMan.TGetBarsClassEvent.

4.5.16 ActnMan.TGetClientItemClassEvent Type

Pascal

```
TGetClientItemClassEvent = procedure (Sender: TCustomActionManager; var ClientItemClass:
TActionClientItemClass) of object;
```

C++

```
(Sender: TCustomActionManager; var ClientItemClass: TActionClientItemClass) (
TGetClientItemClassEvent)();
```

File

ActnMan

Description

This is type ActnMan.TGetClientItemClassEvent.

4.5.17 ActnMan.TGetClientsClassEvent Type

Pascal

```
TGetClientsClassEvent = procedure (Sender: TCustomActionManager; var ClientsClass:
TActionClientsClass) of object;
```

C++

```
(Sender: TCustomActionManager; var ClientsClass: TActionClientsClass) (
TGetClientsClassEvent)();
```

File

ActnMan

Description

This is type ActnMan.TGetClientsClassEvent.

4.5.18 ActnMan.TGetControlClassEvent Type

Pascal

```
TGetControlClassEvent = procedure (Sender: TCustomActionBar; AnItem: TActionClient; var
ControlClass: TCustomActionControlClass) of object;
```

C++

```
(Sender: TCustomActionBar; AnItem: TActionClient; var ControlClass: TCustomActionControlClass)
( TGetControlClassEvent)();
```

File

ActnMan

Description

This is type ActnMan.TGetControlClassEvent.

4.5.19 ActnMan.TMergeRange Type

Pascal

```
TMergeRange = 0..High(Integer);
```

C++

```
0..High(Integer) TMergeRange;
```

File

ActnMan

Description

This is type ActnMan.TMergeRange.

4.5.20 ActnMan.TStyleChanged Type

Pascal

```
TStyleChanged = procedure (Sender: TCustomActionManager) of object;
```

C++

```
(Sender: TCustomActionManager) ( TStyleChanged)();
```

File

ActnMan

Description

This is type ActnMan.TStyleChanged.

4.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
ActionBarDesigner (see page 575)	This is variable ActnMan.ActionBarDesigner.
ActionBarStyles (see page 575)	This is variable ActnMan.ActionBarStyles.
DefaultActnBarStyle (see page 575)	This is variable ActnMan.DefaultActnBarStyle.

4.6.1 ActnMan.ActionBarDesigner Variable

Pascal

```
ActionBarDesigner: IActionBarDesigner = nil;
```

C++

```
IActionBarDesigner ActionBarDesigner = nil;
```

File

ActnMan

Description

This is variable ActnMan.ActionBarDesigner.

4.6.2 ActnMan.ActionBarStyles Variable

Pascal

```
ActionBarStyles: TActionBarStyleList;
```

C++

```
TActionBarStyleList ActionBarStyles;
```

File

ActnMan

Description

This is variable ActnMan.ActionBarStyles.

4.6.3 ActnMan.DefaultActnBarStyle Variable

Pascal

```
DefaultActnBarStyle: string;
```

C++

```
AnsiString DefaultActnBarStyle;
```

File

ActnMan

Description

This is variable ActnMan.DefaultActnBarStyle.

4.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
caAllChanges (see page 577)	This is constant ActnMan.caAllChanges.
cDefaultSchedule (see page 577)	This is constant ActnMan.cDefaultSchedule.
CM_ENTERMENULOOP (see page 578)	This is constant ActnMan.CM_ENTERMENULOOP.
CM_ITEMCLICKED (see page 578)	This is constant ActnMan.CM_ITEMCLICKED.
CM_ITEMDROPPOINT (see page 578)	This is constant ActnMan.CM_ITEMDROPPOINT.
CM_ITEMKEYED (see page 579)	This is constant ActnMan.CM_ITEMKEYED.
CM_ITEMSELECTED (see page 579)	This is constant ActnMan.CM_ITEMSELECTED.
CM_RESETBAR (see page 579)	This is constant ActnMan.CM_RESETBAR.
CM_SCROLLCLICKED (see page 580)	This is constant ActnMan.CM_SCROLLCLICKED.
ControlDragMode (see page 580)	This is constant ActnMan.ControlDragMode.

4.7.1 ActnMan.caAllChanges Constant

Pascal

```
caAllChanges = [caModify, caMove, caDelete];
```

C++

```
caAllChanges = [caModify, caMove, caDelete];
```

File

ActnMan

Description

This is constant ActnMan.caAllChanges.

4.7.2 ActnMan.cDefaultSchedule Constant

Pascal

```
cDefaultSchedule: string = '0=3'#13#10'1=3'#13#10'2=6'#13#10'3=9'#13#10 +
'4=12'#13#10'5=12'#13#10'6=17'#13#10'7=17'#13#10 +
'8=17'#13#10'9=23'#13#10'10=23'#13#10'11=23'#13#10 +
'12=23'#13#10'13=23'#13#10'14=31'#13#10'15=31'#13#10 +
'16=31'#13#10'17=31'#13#10'18=31'#13#10'19=31'#13#10 +
'20=31'#13#10'21=31'#13#10'22=31'#13#10'23=31'#13#10 + '24=31'#13#10'25=31'#13#10;
```

C++

```
AnsiString cDefaultSchedule = '0=3'#13#10'1=3'#13#10'2=6'#13#10'3=9'#13#10 +
'4=12'#13#10'5=12'#13#10'6=17'#13#10'7=17'#13#10 +
'8=17'#13#10'9=23'#13#10'10=23'#13#10'11=23'#13#10 +
'12=23'#13#10'13=23'#13#10'14=31'#13#10'15=31'#13#10 +
'16=31'#13#10'17=31'#13#10'18=31'#13#10'19=31'#13#10 +
```

```
'20=31'#13#10'21=31'#13#10'22=31'#13#10'23=31'#13#10 + '24=31'#13#10'25=31'#13#10;
```

File

ActnMan

Description

This is constant ActnMan.cDefaultSchedule.

4.7.3 ActnMan.CM_ENTERMENULOOP Constant

Pascal

```
CM_ENTERMENULOOP = CM_BASE + $0410;
```

C++

```
CM_ENTERMENULOOP = CM_BASE + $0410;
```

File

ActnMan

Description

This is constant ActnMan.CM_ENTERMENULOOP.

4.7.4 ActnMan.CM_ITEMCLICKED Constant

Pascal

```
CM_ITEMCLICKED = CM_BASE + $0403;
```

C++

```
CM_ITEMCLICKED = CM_BASE + $0403;
```

File

ActnMan

Description

This is constant ActnMan.CM_ITEMCLICKED.

4.7.5 ActnMan.CM_ITEMDROPPOINT Constant

Pascal

```
CM_ITEMDROPPOINT = CM_BASE + $0411;
```

C++

```
CM_ITEMDROPPOINT = CM_BASE + $0411;
```

File

ActnMan

Description

This is constant ActnMan.CM_ITEMDROPPOINT.

4.7.6 ActnMan.CM_ITEMKEYED Constant

Pascal

```
CM_ITEMKEYED = CM_BASE + $0404;
```

C++

```
CM_ITEMKEYED = CM_BASE + $0404;
```

File

ActnMan

Description

This is constant ActnMan.CM_ITEMKEYED.

4.7.7 ActnMan.CM_ITEMSELECTED Constant

Pascal

```
CM_ITEMSELECTED = CM_BASE + $0402;
```

C++

```
CM_ITEMSELECTED = CM_BASE + $0402;
```

File

ActnMan

Description

This is constant ActnMan.CM_ITEMSELECTED.

4.7.8 ActnMan.CM_RESETBAR Constant

Pascal

```
CM_RESETBAR = CM_BASE + $0409;
```

C++

```
CM_RESETBAR = CM_BASE + $0409;
```

File

ActnMan

Description

This is constant ActnMan.CM_RESETBAR.

4.7.9 ActnMan.CM_SCROLLCLICKED Constant

Pascal

```
CM_SCROLLCLICKED = CM_BASE + $0406;
```

C++

```
CM_SCROLLCLICKED = CM_BASE + $0406;
```

File

ActnMan

Description

This is constant ActnMan.CM_SCROLLCLICKED.

4.7.10 ActnMan.ControlDragMode Constant

Pascal

```
ControlDragMode: array[Boolean] of TDragMode = (dmManual, dmAutomatic);
```

C++

```
array[Boolean] of TDragMode ControlDragMode = (dmManual, dmAutomatic);
```

File

ActnMan

Description

This is constant ActnMan.ControlDragMode.

5 ActnMenus Namespace

This is namespace ActnMenus.

Topics

Name	Description
Classes (see page 583)	The following table lists classes in this documentation.
Functions (see page 837)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 839)	The following table lists structs, records, enums in this documentation.
Types (see page 841)	The following table lists types in this documentation.
Variables (see page 846)	The following table lists variables in this documentation.

Classes

	Name	Description
	TActionBarStyleEx (see page 583)	TActionBarStyleEx is the implementation class for styles in Action Manager.
	TActionMainMenuBar (see page 585)	TActionMainMenuBar is a rendering engine that renders action client items as menu items.
	TCustomActionMainMenuBar (see page 661)	TCustomActionMainMenuBar is the ancestor for the TActionMainMenuBar (see page 585) component.
	TCustomActionMenuBar (see page 685)	TCustomActionMenuBar is the ancestor for action bands that render actions as menu item UI elements.
	TCustomActionPopupMenu (see page 721)	TCustomActionPopupMenu is the ancestor for popup action bands that render actions as popup menu item UI elements.
	TCustomAddRemoveItem (see page 745)	TCustomAddRemoveItem is the base class for the ActionBand menus for customizing action bands.
	TCustomizeActionClientItem (see page 756)	This is class ActnMenus.TCustomizeActionClientItem.
	TCustomizeActionLink (see page 759)	This is class ActnMenus.TCustomizeActionLink.
	TCustomizeActionToolBar (see page 762)	This is class ActnMenus.TCustomizeActionToolBar.
	TCustomMDIMenuButton (see page 781)	This is class ActnMenus.TCustomMDIMenuButton.
	TCustomMenuButton (see page 790)	TCustomMenuButton is the base class for ActionBand button controls.
	TCustomMenuExpandBtn (see page 805)	This is class ActnMenus.TCustomMenuExpandBtn.
	TCustomMenuItem (see page 816)	TCustomMenuItem is the base class for the ActionBand custom control TStandardMenuItem.
	TMenuStack (see page 833)	This is class ActnMenus.TMenuStack.

Functions

	Name	Description
	RegisterActnBarStyle (see page 837)	Registers an action band style so that it is available for use from TActionManager's Style property within the IDE.

	UnRegisterActnBarStyle (see page 837)	Unregisters an action band style from the IDE.

Structs, Records, Enums

	Name	Description
	TAnimationStyle (see page 839)	This is record ActnMenus.TAnimationStyle.
	TMDIButtonStyle (see page 839)	This is record ActnMenus.TMDIButtonStyle.

Types

Name	Description
TAddRemoveItemClass (see page 841)	This is type ActnMenus.TAddRemoveItemClass.
TCustomAddRemoveItemClass (see page 841)	This is type ActnMenus.TCustomAddRemoveItemClass.
TCustomizeActionToolBarClass (see page 842)	This is type ActnMenus.TCustomizeActionToolBarClass.
TCustomMenuExpandBtnClass (see page 842)	This is type ActnMenus.TCustomMenuExpandBtnClass.
TCustomPopupClass (see page 842)	This is type ActnMenus.TCustomPopupClass.
TExitMenuEvent (see page 843)	This is type ActnMenus.TExitMenuEvent.
TGetPopupClassEvent (see page 843)	This is type ActnMenus.TGetPopupClassEvent.
TMenuButtonControlClass (see page 843)	This is type ActnMenus.TMenuButtonControlClass.
TMenuEdges (see page 844)	This is type ActnMenus.TMenuEdges.
TMenuItemControlClass (see page 844)	This is type ActnMenus.TMenuItemControlClass.
TMenuPopupEvent (see page 844)	This is type ActnMenus.TMenuPopupEvent.
TUpdateActnMenusProc (see page 845)	This is type ActnMenus.TUpdateActnMenusProc.

Variables

Name	Description
MenuAddRemoveItemClass (see page 846)	This is variable ActnMenus.MenuAddRemoveItemClass.
MenuButtonControlClass (see page 846)	This is variable ActnMenus.MenuButtonControlClass.
MenuCustomizePopupClass (see page 847)	This is variable ActnMenus.MenuCustomizePopupClass.
MenuItemControlClass (see page 847)	This is variable ActnMenus.MenuItemControlClass.
MenuPopupClass (see page 847)	This is variable ActnMenus.MenuPopupClass.
UpdateActnMenusProc (see page 848)	This is variable ActnMenus.UpdateActnMenusProc.

5.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TActionBarStyleEx (see page 583)	TActionBarStyleEx is the implementation class for styles in Action Manager.
	TActionMainMenuBar (see page 585)	TActionMainMenuBar is a rendering engine that renders action client items as menu items.
	TCustomActionMainMenuBar (see page 661)	TCustomActionMainMenuBar is the ancestor for the TActionMainMenuBar (see page 585) component.
	TCustomActionMenuBar (see page 685)	TCustomActionMenuBar is the ancestor for action bands that render actions as menu item UI elements.
	TCustomActionPopupMenu (see page 721)	TCustomActionPopupMenu is the ancestor for popup action bands that render actions as popup menu item UI elements.
	TCustomAddRemoveItem (see page 745)	TCustomAddRemoveItem is the base class for the ActionBand menus for customizing action bands.
	TCustomizeActionClientItem (see page 756)	This is class ActnMenus.TCustomizeActionClientItem.
	TCustomizeActionLink (see page 759)	This is class ActnMenus.TCustomizeActionLink.
	TCustomizeActionToolBar (see page 762)	This is class ActnMenus.TCustomizeActionToolBar.
	TCustomMDIMenuButton (see page 781)	This is class ActnMenus.TCustomMDIMenuButton.
	TCustomMenuButton (see page 790)	TCustomMenuButton is the base class for ActionBand button controls.
	TCustomMenuExpandBtn (see page 805)	This is class ActnMenus.TCustomMenuExpandBtn.
	TCustomMenuItem (see page 816)	TCustomMenuItem is the base class for the ActionBand custom control TStandardMenuItem.
	TMenuStack (see page 833)	This is class ActnMenus.TMenuStack.

5.1.1 TActionBarStyleEx Class

TActionBarStyleEx is the implementation class for styles in Action Manager.

Class Hierarchy

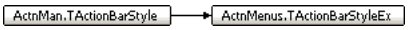

Pascal

```
TActionBarStyleEx = class(TActionBarStyle);
```

C++

```
class TActionBarStyleEx : public TActionBarStyle;
```

File

ActnMenus

Description

When you select alternate styles for action band components, Action Manager uses TActionBarStyleEx to implement the approprite controls.

Note: When switching between alternate styles in Action Manager (such as XP and standard), the appropriate controls will be added to the uses clause automatically (such as XPStyleActnCtrls or StdStyleActnCtrls). If you want to reduce the amount of code that will be linked into your application, remove the unused control statement from the uses clause.

See Also

TActionManager (see page 432)

5.1.1.1 TActionBarStyleEx Members

The following tables list the members exposed by TActionBarStyleEx.

Public Methods

	Name	Description
	GetAddRemoveItemClass (see page 584)	Returns the base class for ActionBand menus for customizing action bands.
	GetColorMapClass (see page 359)	Returns the class type of the colormap for an action bar.
	GetControlClass (see page 359)	Returns the control class type on an action bar item.
	GetPopupClass (see page 584)	Returns the popup class added to a menu action band.
	GetScrollBtnClass (see page 585)	This is GetScrollBtnClass, a member of class TActionBarStyleEx.
	GetStyleName (see page 360)	Returns the style associated with action manager.

5.1.1.2 TActionBarStyleEx Methods

The methods of the TActionBarStyleEx class are listed here.

Public Methods

	Name	Description
	GetAddRemoveItemClass (see page 584)	Returns the base class for ActionBand menus for customizing action bands.
	GetPopupClass (see page 584)	Returns the popup class added to a menu action band.
	GetScrollBtnClass (see page 585)	This is GetScrollBtnClass, a member of class TActionBarStyleEx.

5.1.1.2.1 TActionBarStyleEx.GetAddRemoveItemClass Method

Returns the base class for ActionBand menus for customizing action bands.

Pascal

```
function GetAddRemoveItemClass(ActionBar: TCustomActionBar): TCustomAddRemoveItemClass;
virtual; abstract;
```

C++

```
virtual __fastcall TCustomAddRemoveItemClass GetAddRemoveItemClass(TCustomActionBar ActionBar)
= 0;
```

Description

Call GetAddRemoveItemClass to customize action bands.

See Also

TCustomAddRemoveItem (see page 745)

5.1.1.2.2 TActionBarStyleEx.GetPopupClass Method

Returns the popup class added to a menu action band.

Pascal

```
function GetPopupClass(ActionBar: TCustomActionBar): TCustomPopupClass; virtual; abstract;
```

C++

```
virtual __fastcall TCustomPopupClass GetPopupClass(TCustomActionBar ActionBar) = 0;
```

Description

Call GetPopupClass to find the popup menu associated with a menu action band. Override GetPopupClass to change which popup class is associated with a menu action band depending on the state of the control.

See Also

TCustomActionPopupMenu (see page 721)

5.1.1.2.3 TActionBarStyleEx.GetScrollBtnClass Method

Pascal

```
function GetScrollBtnClass: TCustomToolScrollBtnClass; virtual; abstract;
```

C++

```
virtual __fastcall TCustomToolScrollBtnClass GetScrollBtnClass() = 0;
```

Description

This is GetScrollBtnClass, a member of class TActionBarStyleEx.

5.1.2 TActionMainMenuBar Class

TActionMainMenuBar is a rendering engine that renders action client items as menu items.

Class Hierarchy

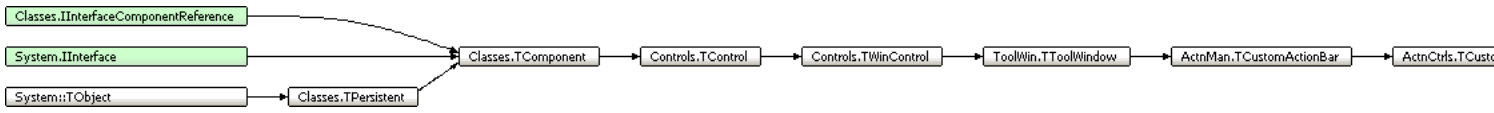

Pascal

```
TActionMainMenuBar = class(TCustomActionMainMenuBar);
```

C++

```
class TActionMainMenuBar : public TCustomActionMainMenuBar;
```

File

ActnMenus

Description

TActionMainMenuBar is a specialized container (like TPanel) that renders a group of items on screen in the order specified in the Action Manager. All of the items are created dynamically when the TActionMainMenuBar component is connected to an item from the Action Manager.

The items themselves are internal components (similar to TMenuItem) created by the TActionMainMenuBar and appear consistent with the look and function of a main menu. The layout of the items on the TActionMainMenuBar is managed automatically and specified by the Action Manager.

5.1.2.1 TActionMainMenuBar Members

The following tables list the members exposed by TActionMainMenuBar.

Protected Events

	Name	Description
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FCachedMenu (see page 699)	This is FCachedMenu, a member of class TCustomActionMenuBar.
	FChildMenu (see page 699)	This is FChildMenu, a member of class TCustomActionMenuBar.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefaultColorMap (see page 460)	This is FDefaultColorMap, a member of class TCustomActionBar.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExpandTimer (see page 700)	This is FExpandTimer, a member of class TCustomActionMenuBar.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FPopupTimer (see page 700)	This is FPopupTimer, a member of class TCustomActionMenuBar.
	FSavedWrapState (see page 460)	This is FSavedWrapState, a member of class TCustomActionBar.
	FSelectedItem (see page 700)	This is FSelectedItem, a member of class TCustomActionMenuBar.
	FWindowMenuItem (see page 676)	This is FWindowMenuItem, a member of class TCustomActionMainMenuBar.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 462)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.

	Animate (see page 701)	This is Animate, a member of class TCustomActionMenuBar.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
V	AutoSizingChanged (see page 462)	This is AutoSizingChanged, a member of class TCustomActionBar.
	BackgroundSize (see page 462)	This is BackgroundSize, a member of class TCustomActionBar.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcInsertPosition (see page 462)	This is CalcInsertPosition, a member of class TCustomActionBar.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CleanupStack (see page 701)	This is CleanupStack, a member of class TCustomActionMenuBar.
V	Clear (see page 463)	This is Clear, a member of class TCustomActionBar.
	ClearSubMenus (see page 702)	This is ClearSubMenus, a member of class TCustomActionMenuBar.
	Click (see page 5743)	Respond to user click.
	CMBorderChanged (see page 23285)	This is CMBorderChanged, a member of class TToolWindow.
	CMCtl3DChanged (see page 23286)	This is CMCtl3DChanged, a member of class TToolWindow.
	CMWininichange (see page 678)	This is CMWininichange, a member of class TCustomActionMainMenuBar.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
V	CreateControl (see page 463)	This is CreateControl, a member of class TCustomActionBar.
V	CreateControl (see page 678)	This is CreateControl, a member of class TCustomActionMainMenuBar.
V	CreateControl (see page 702)	This is CreateControl, a member of class TCustomActionMenuBar.
V	CreateControls (see page 463)	This is CreateControls, a member of class TCustomActionBar.
V	CreateControls (see page 678)	This is CreateControls, a member of class TCustomActionMainMenuBar.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreatePopup (see page 702)	This is CreatePopup, a member of class TCustomActionMenuBar.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 463)	Returns placement order of custom-aligned child control.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 464)	Returns aligned position of custom-aligned child control.
V	CustomAlignPosition (see page 678)	Returns aligned position of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.

	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 465)	This is DesignWndProc, a member of class TCustomActionBar.
	DesignWndProc (see page 679)	This is DesignWndProc, a member of class TCustomActionMainMenuBar.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoControlCreated (see page 465)	This is DoControlCreated, a member of class TCustomActionBar.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoDropActions (see page 465)	This is DoDropActions, a member of class TCustomActionBar.
	DoDropCategory (see page 465)	This is DoDropCategory, a member of class TCustomActionBar.
	DoDropItem (see page 466)	This is DoDropItem, a member of class TCustomActionBar.
	DoEndDock (see page 146)	OnEndDock event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetControlClass (see page 466)	This is DoGetControlClass, a member of class TCustomActionBar.
	DoGetPopupClass (see page 703)	This is DoGetPopupClass, a member of class TCustomActionMenuBar.
	DoItemClicked (see page 703)	This is DoItemClicked, a member of class TCustomActionMenuBar.
	DoItemKeyed (see page 703)	This is DoItemKeyed, a member of class TCustomActionMenuBar.
	DoItemSelected (see page 703)	This is DoItemSelected, a member of class TCustomActionMenuBar.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMenuDelay (see page 703)	This is DoMenuDelay, a member of class TCustomActionMenuBar.
	DoMouseIdle (see page 704)	This is DoMouseIdle, a member of class TCustomActionMenuBar.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoneMenuLoop (see page 679)	This is DoneMenuLoop, a member of class TCustomActionMainMenuBar.
	DoneMenuLoop (see page 704)	This is DoneMenuLoop, a member of class TCustomActionMenuBar.
	DoPopup (see page 704)	This is DoPopup, a member of class TCustomActionMenuBar.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 147)	OnStartDock event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 466)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawBackground (see page 467)	This is DrawBackground, a member of class TCustomActionBar.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.

	ExecAction (see page 704)	This is ExecAction, a member of class TCustomActionMenuBar.
	Expand (see page 704)	Causes the menu bar to show all items, including the hidden ones.
	ExpandTimer (see page 705)	This is ExpandTimer, a member of class TCustomActionMenuBar.
	FindAccelItem (see page 467)	This is FindAccelItem, a member of class TCustomActionBar.
	FindItemAt (see page 468)	This is FindItemAt, a member of class TCustomActionBar.
	FindItemCaption (see page 468)	This is FindItemCaption, a member of class TCustomActionBar.
	FindLast (see page 468)	This is FindLast, a member of class TCustomActionBar.
	FindLeastUsedItem (see page 469)	This is FindLeastUsedItem, a member of class TCustomActionBar.
	FindNearestControl (see page 469)	This is FindNearestControl, a member of class TCustomActionBar.
	FindNext (see page 470)	This is FindNext, a member of class TCustomActionBar.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FindNextVisibleItem (see page 470)	This is FindNextVisibleItem, a member of class TCustomActionBar.
	FindPrevious (see page 470)	This is FindPrevious, a member of class TCustomActionBar.
	FindPreviousVisibleItem (see page 470)	This is FindPreviousVisibleItem, a member of class TCustomActionBar.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionControl (see page 471)	This is GetActionControl, a member of class TCustomActionBar.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetActiveMDIChild (see page 679)	This is GetActiveMDIChild, a member of class TCustomActionMainMenuBar.
	GetBannerWidth (see page 471)	This is GetBannerWidth, a member of class TCustomActionBar.
	GetBarHeight (see page 471)	This is GetBarHeight, a member of class TCustomActionBar.
	GetBarWidth (see page 471)	This is GetBarWidth, a member of class TCustomActionBar.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlClass (see page 471)	This is GetControlClass, a member of class TCustomActionBar.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDefaultColorMapClass (see page 472)	This is GetDefaultColorMapClass, a member of class TCustomActionBar.
	GetDefaultSounds (see page 705)	This is GetDefaultSounds, a member of class TCustomActionMenuBar.
	GetDesignMode (see page 472)	This is GetDesignMode, a member of class TCustomActionBar.
	GetDesignMode (see page 705)	This is GetDesignMode, a member of class TCustomActionMenuBar.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 147)	Returns the class used for the control when it is floating.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetMouseHoverItem (see page 705)	This is GetMouseHoverItem, a member of class TCustomActionMenuBar.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupClass (see page 706)	This is GetPopupClass, a member of class TCustomActionMenuBar.

	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetPopupMenuClass (see page 472)	This is GetPopupMenuClass, a member of class TCustomActionBar.
	GetSelected (see page 706)	This is GetSelected, a member of class TCustomActionMenuBar.
V	GetSiteInfo (see page 147)	Generates an OnGetSiteInfo event
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasItems (see page 472)	This is HasItems, a member of class TCustomActionBar.
	Idle (see page 706)	This is Idle, a member of class TCustomActionMenuBar.
V	InitMenuLoop (see page 706)	This is InitMenuLoop, a member of class TCustomActionMenuBar.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsDesignMsg (see page 707)	This is IsDesignMsg, a member of class TCustomActionMenuBar.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
V	ItemCount (see page 473)	This is ItemCount, a member of class TCustomActionBar.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 148)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 473)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 707)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndHook (see page 680)	This is MainWndHook, a member of class TCustomActionMainMenuBar.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MDIChildMaximized (see page 680)	This is MDIChildMaximized, a member of class TCustomActionMainMenuBar.
	MDIChildRestored (see page 680)	This is MDIChildRestored, a member of class TCustomActionMainMenuBar.
V	MouseActivate (see page 707)	This is MouseActivate, a member of class TCustomActionMenuBar.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	NCPaint (see page 473)	This is NCPaint, a member of class TCustomActionBar.
V	NCPaint (see page 23286)	This is NCPaint, a member of class TToolWindow.
V	NewPopup (see page 680)	This is NewPopup, a member of class TCustomActionMainMenuBar.
V	NewPopup (see page 707)	This is NewPopup, a member of class TCustomActionMenuBar.
V	Notification (see page 473)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 681)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 708)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 474)	This is Paint, a member of class TCustomActionBar.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 474)	Renders the image of a windowed control.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.

	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	ProcessMenuLoop (see page 708)	This is ProcessMenuLoop, a member of class TCustomActionMenuBar.
	ProcessMessage (see page 708)	This is ProcessMessage, a member of class TCustomActionMenuBar.
	ProcessMessages (see page 709)	This is ProcessMessages, a member of class TCustomActionMenuBar.
	ProcessMouseMsg (see page 681)	This is ProcessMouseMsg, a member of class TCustomActionMainMenuBar.
	ProcessMouseMsg (see page 709)	This is ProcessMouseMsg, a member of class TCustomActionMenuBar.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	RefreshMDIMenu (see page 681)	This is RefreshMDIMenu, a member of class TCustomActionMainMenuBar.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Reset (see page 475)	This is Reset, a member of class TCustomActionBar.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Select (see page 709)	This is Select, a member of class TCustomActionMenuBar.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectItem (see page 710)	This is SelectItem, a member of class TCustomActionMenuBar.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActionClient (see page 475)	This is SetActionClient, a member of class TCustomActionBar.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetAutoSizing (see page 475)	This is SetAutoSizing, a member of class TCustomActionBar.
	SetBiDiMode (see page 475)	Sets the control's bi-directional mode.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetColorMap (see page 476)	This is SetColorMap, a member of class TCustomActionBar.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignMode (see page 476)	This is SetDesignMode, a member of class TCustomActionBar.
	SetDesignMode (see page 681)	Specifies whether the whether the action band is being customized or being used in the IDE.
	SetDesignMode (see page 710)	Specifies whether the whether the action band is being customized or being used in the IDE.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetOrientation (see page 477)	This is SetOrientation, a member of class TCustomActionBar.
V	SetParent (see page 710)	Sets the parent of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetPersistentHotKeys (see page 477)	This is SetPersistentHotKeys, a member of class TCustomActionBar.
V	SetPersistentHotKeys (see page 711)	This is SetPersistentHotKeys, a member of class TCustomActionMenuBar.
V	SetSpacing (see page 477)	This is SetSpacing, a member of class TCustomActionBar.
	SetupDefaultColorMap (see page 477)	This is SetupDefaultColorMap, a member of class TCustomActionBar.
	SetupWindowMenu (see page 682)	This is SetupWindowMenu, a member of class TCustomActionMainMenuBar.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	VisibleChanging (see page 478)	Provides the interface for a method that responds to changes in the control's visibility.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMKeyDown (see page 682)	This is WMKeyDown, a member of class TCustomActionMainMenuBar.
	WMNCCalcSize (see page 23286)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCHitTest (see page 682)	This is WMNCHitTest, a member of class TCustomActionMainMenuBar.
	WMNCPaint (see page 23286)	This is WMNCPaint, a member of class TToolWindow.
	WMSysCommand (see page 683)	This is WMSysCommand, a member of class TCustomActionMainMenuBar.
	WMSysKeyDown (see page 683)	This is WMSysKeyDown, a member of class TCustomActionMainMenuBar.
	WMSysKeyUp (see page 683)	This is WMSysKeyUp, a member of class TCustomActionMainMenuBar.
V	WndProc (see page 711)	Provides specific message responses for the control.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DelayItem (see page 717)	This is DelayItem, a member of class TCustomActionMenuBar.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	Expandable (see page 717)	This is Expandable, a member of class TCustomActionMenuBar.
	Expanded (see page 718)	This is Expanded, a member of class TCustomActionMenuBar.
R	HRowCount (see page 488)	Indicates the number of columns of items that the action band displays.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	Items (see page 488)	Lists the action clients that correspond to the child controls of this action band.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	MouseControl (see page 718)	This is MouseControl, a member of class TCustomActionMenuBar.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentMenu (see page 719)	This is ParentMenu, a member of class TCustomActionMenuBar.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
R	PopupStack (see page 719)	This is PopupStack, a member of class TCustomActionMenuBar.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
R	Selected (see page 719)	This is Selected, a member of class TCustomActionMenuBar.
	Text (see page 5842)	Contains a text string associated with the control.
R	VRowCount (see page 490)	Indicates the number of rows of items that the action band displays.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
R	Widest (see page 491)	This is Widest, a member of class TCustomActionBar.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.

	CalcDockedWidth (see page 146)	This is CalcDockedWidth, a member of class TCustomActionDockBar.
	CanFocus (see page 677)	Indicates whether a TActionMainMenuBar (see page 585) component can receive focus.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	CloseMenu (see page 677)	Closes the menu currently being displayed.
	CloseMenu (see page 702)	Closes the menu currently being displayed.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 145)	Creates and initializes a TCustomActionDockBar (see page 133) instance.
	Create (see page 459)	Creates and initializes a TCustomActionBar (see page 447) instance.
	Create (see page 675)	Creates and initializes a TCustomActionMainMenuBar (see page 661) instance.
	Create (see page 698)	Creates and initializes a TCustomActionMenuBar (see page 685) instance.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 5731)	Creates an instance of TControl (see page 5724).
	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 23285)	Creates and initializes a TToolWindow (see page 23274) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 459)	Destroys an instance of TCustomActionBar (see page 447).
	Destroy (see page 676)	Destroys an instance of TCustomActionMainMenuBar (see page 661).
	Destroy (see page 698)	Destroys an instance of TCustomActionMenuBar (see page 685).
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 146)	Used internally to dock the action band.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 466)	Generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.

	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
V	FindFirst (see page 467)	Finds the first TActionClientItem (see page 374) in the Items (see page 488) list.
	FindFirstVisibleItem (see page 468)	Finds the first visible TActionClientItem (see page 374) in the Items (see page 488) list.
	FindLastVisibleItem (see page 469)	Finds the last visibleTActionClientItem in the Items (see page 488) list.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	RecreateControls (see page 475)	Destroys and recreates all of the action band's action client controls.
	RecreateControls (see page 709)	Destroys and recreates all of the action band's action client controls.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 476)	Sets the bounds based on whether the popup is expanded or not
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetDesignVisible (see page 6178)	Makes a control visible at design time.
	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Style (see page 477)	Identifies the style object that gives the application its default look and feel.
	Style (see page 711)	Retrieves the style used by the ActionBand menu that started the menu loop so the submenus have a consistent appearance.
	TrackMenu (see page 682)	Executes the action associated with the menu item.
	TrackMenu (see page 711)	Executes the action associated with the menu item.
	Update (see page 5789)	Processes any pending paint messages immediately.
	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.

	ActionClient (see page 479)	Specifies the action client associated with this action band.
R	ActionControls (see page 479)	Lists the controls on the action band.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AutoSizing (see page 480)	Specifies whether the action band adjusts its size automatically to fit around its child controls.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	Canvas (see page 607)	Represents the drawing surface of the action band.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	ContextBar (see page 485)	Indicates whether items shown in this action band represent the items in a popup menu.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designable (see page 485)	Specifies whether the action band can be customized (used internally by the IDE).
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignMode (see page 486)	Specifies whether the action band is being customized or being used in the IDE.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HorzSeparator (see page 487)	Indicates whether a horizontal separator bar appears between item lines.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	Inactive (see page 683)	Specifies whether the action band is disabled or enabled.
	InMenuLoop (see page 718)	Indicates whether items in the menu can be selected.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	Orientation (see page 488)	Specifies how the action band lays out its child items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.

	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentControl (see page 718)	Refers to the parent object of the TCustomActionBarControl.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	RootMenu (see page 719)	Refers to the root menu of the ActionBand menu.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	VertSeparator (see page 490)	Indicates whether a vertical separator bar should appear between item lines.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 625)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 626)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 627)	Occurs when an attempt is made to resize the control.
	OnClick (see page 627)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 629)	Adjust resize constraints.
	OnContextPopup (see page 629)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnControlCreated (see page 630)	Occurs when a control is created on the action band.
	OnDblClick (see page 631)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 631)	Occurs when another control is docked to the control.
	OnDockOver (see page 633)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 635)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 637)	Occurs when the user drags an object over a control.
	OnEndDock (see page 640)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 640)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 641)	Occurs when a control receives the input focus.
	OnEnterMenuLoop (see page 643)	Occurs when an ActionBand menu loop has begun.
	OnExit (see page 644)	Occurs when the input focus shifts away from one control to another.
	OnExitMenuLoop (see page 645)	Occurs when an ActionBand menu loop has ended and the menu is closed.
	OnGetControlClass (see page 645)	Occurs when an item is to be placed on the action band.
	OnGetPopupClass (see page 646)	Occurs prior to an ActionBand popupmenu appearing on screen.
	OnGetSiteInfo (see page 646)	Returns the control's docking information.
	OnMouseActivate (see page 648)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 649)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 651)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 654)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 656)	Occurs when the action band is painted.
	OnPopup (see page 656)	Occurs just before the pop-up menu appears.
	OnResize (see page 657)	Occurs immediately after the control is resized.
	OnStartDock (see page 657)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 658)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

	OnUnDock (see page 658)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	ActionManager (see page 601)	Specifies the action manager component that is associated with the action band.
	Align (see page 601)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AllowHiding (see page 605)	Determines whether the action band can be hidden.
	Anchors (see page 605)	Specifies how the control is anchored to its parent.
	AnimateDuration (see page 605)	This is AnimateDuration, a member of class TActionMainMenuBar.
	AnimationStyle (see page 606)	Specifies the type of animation the menu bar will display as it opens.
	AutoFocus (see page 606)	This is AutoFocus, a member of class TActionMainMenuBar.
	BiDiMode (see page 606)	Specifies the bi-directional mode for the control.
	Caption (see page 607)	Specifies a text string that identifies the control to the user.
	Color (see page 610)	Specifies the background color of the control.
	ColorMap (see page 610)	Indicates the colormap for individual action band components.
	Constraints (see page 611)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 612)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 612)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 612)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EdgeBorders (see page 614)	Determines which sides of the control have visible borders.
	EdgeInner (see page 614)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 615)	Determines the border style for the control's outer edge.
	Enabled (see page 615)	Controls whether the control responds to mouse, keyboard, and timer events.
	ExpandDelay (see page 616)	Specifies, in milliseconds, the delay period that a menu action band uses before showing hidden or unused items.
	Font (see page 616)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HorzMargin (see page 617)	Specifies the width of the horizontal margins in pixels.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBackground (see page 617)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 617)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 618)	Determines where a control looks for its color information.
	ParentFont (see page 618)	Determines where a control looks for its font information.
	ParentShowHint (see page 619)	Determines where a control looks to find out if its Help Hint should be shown.
	PersistentHotKeys (see page 619)	Specifies whether keyboard accelerator shortcuts are always displayed.
	Shadows (see page 620)	Refers to the TActionMainMenuBar component's shadow.
	ShowHint (see page 620)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Spacing (see page 622)	Determines where the image and text appear on an action band's rendered control that represents the action client.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.

	UseSystemFont (see page 622)	Specifies whether the default system font for menu items is used.
	VertMargin (see page 622)	Specifies the width of the vertical margins in pixels.
	Visible (see page 623)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	WindowMenu (see page 624)	Specifies the caption of the window menu.

5.1.2.2 TActionMainMenuBar Properties

The properties of the TActionMainMenuBar class are listed here.

Public Properties

	Name	Description
	Canvas (see page 607)	Represents the drawing surface of the action band.

Published Properties

	Name	Description
	ActionManager (see page 601)	Specifies the action manager component that is associated with the action band.
	Align (see page 601)	Determines how the control aligns within its container (parent control).
	AllowHiding (see page 605)	Determines whether the action band can be hidden.
	Anchors (see page 605)	Specifies how the control is anchored to its parent.
	AnimateDuration (see page 605)	This is AnimateDuration, a member of class TActionMainMenuBar.
	AnimationStyle (see page 606)	Specifies the type of animation the menu bar will display as it opens.
	AutoFocus (see page 606)	This is AutoFocus, a member of class TActionMainMenuBar.
	BiDiMode (see page 606)	Specifies the bi-directional mode for the control.
	Caption (see page 607)	Specifies a text string that identifies the control to the user.
	Color (see page 610)	Specifies the background color of the control.
	ColorMap (see page 610)	Indicates the colormap for individual action band components.
	Constraints (see page 611)	Specifies the size constraints for the control.
	DragCursor (see page 612)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 612)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 612)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EdgeBorders (see page 614)	Determines which sides of the control have visible borders.
	EdgeInner (see page 614)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 615)	Determines the border style for the control's outer edge.
	Enabled (see page 615)	Controls whether the control responds to mouse, keyboard, and timer events.
	ExpandDelay (see page 616)	Specifies, in milliseconds, the delay period that a menu action band uses before showing hidden or unused items.
	Font (see page 616)	Controls the attributes of text written on or in the control.
	HorzMargin (see page 617)	Specifies the width of the horizontal margins in pixels.
	ParentBackground (see page 617)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 617)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 618)	Determines where a control looks for its color information.
	ParentFont (see page 618)	Determines where a control looks for its font information.
	ParentShowHint (see page 619)	Determines where a control looks to find out if its Help Hint should be shown.
	PersistentHotKeys (see page 619)	Specifies whether keyboard accelerator shortcuts are always displayed.
	Shadows (see page 620)	Refers to the TActionMainMenuBar component's shadow.
	ShowHint (see page 620)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Spacing (see page 622)	Determines where the image and text appear on an action band's rendered control that represents the action client.
	UseSystemFont (see page 622)	Specifies whether the default system font for menu items is used.
	VertMargin (see page 622)	Specifies the width of the vertical margins in pixels.
	Visible (see page 623)	Determines whether the component appears onscreen.
	WindowMenu (see page 624)	Specifies the caption of the window menu.

5.1.2.2.1 TActionMainMenuBar.ActionManager Property

Specifies the action manager component that is associated with the action band.

Pascal

```
property ActionManager: TCustomActionManager;
```

C++

```
__property TCustomActionManager ActionManager;
```

Description

Use ActionManager to access the action manager that uses this action band to render UI elements it manages. Typically, this association is created when you drag actions from the action manager and drop them on the action bar. When an action is dropped on an action band and the ActionManager property is not assigned, the ActionManager property is automatically assigned to the action manager that manages the action that was dropped on the action band.

At runtime, you can perform the equivalent operation to dragging an action onto the action band by setting the ActionBar property of one of the action clients that the action manager's ActionBars property lists so that it specifies this component.

See Also

ActionBars (see page 553)

ActionClient (see page 479)

ActionBar (see page 370)

5.1.2.2.2 TActionMainMenuBar.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
```

```
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
```

```
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

5.1.2.2.3 TActionMainMenuBar.AllowHiding Property

Determines whether the action band can be hidden.

Pascal

```
property AllowHiding: Boolean;
```

C++

```
__property Boolean AllowHiding;
```

Description

Use AllowHiding to specify whether the action band can be hidden at runtime, if the application allows runtime customization by the end user. Setting AllowHiding to true allows the action band to be hidden at runtime. If AllowHiding is set to false, the action cannot be hidden. Certain action bands, such as the main menu action band, should set this AllowHiding to false.

5.1.2.2.4 TActionMainMenuBar.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

5.1.2.2.5 TActionMainMenuBar.AnimateDuration Property

Pascal

```
property AnimateDuration: Integer;
```

C++

```
__property int AnimateDuration;
```

Description

This is AnimateDuration, a member of class TActionMainMenuBar.

5.1.2.2.6 TActionMainMenuBar.AnimationStyle Property

Specifies the type of animation the menu bar will display as it opens.

Pascal

```
property AnimationStyle: TAnimationStyle;
```

C++

```
__property TAnimationStyle AnimationStyle;
```

Description

Use AnimationStyle to change the style of animation used by the items in the menu bar as they open. The options are:.

Value	Animation Description
asNone	No animation is used; the menu appears fully formed.
asDefault	The operating system default animation style is used.
asUnfold	The menu unfolds diagonally down from the upper left corner.
asSlide	The menu slides from the top down.
asFade	The menu fades in from invisible to visible.

5.1.2.2.7 TActionMainMenuBar.AutoFocus Property

Pascal

```
property AutoFocus: Boolean;
```

C++

```
__property Boolean AutoFocus;
```

Description

This is AutoFocus, a member of class TActionMainMenuBar.

5.1.2.2.8 TActionMainMenuBar.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data

aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

5.1.2.2.9 TActionMainMenuBar.Canvas Property

Represents the drawing surface of the action band.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

TCustomActionBar uses Canvas internally to render the background on which its child controls appear.

5.1.2.2.10 TActionMainMenuBar.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
```

```
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
```

```
end;
```

5.1.2.2.11 TActionMainMenuBar.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

5.1.2.2.12 TActionMainMenuBar.ColorMap Property

Indicates the colormap for individual action band components.

Pascal

```
property ColorMap: TCustomActionBarColorMap;
```

C++

```
__property TCustomActionBarColorMap ColorMap;
```

Description

Use the ColorMap property to change the default colormap for individual action band components. By default, the ColorMap property is set according to the first registered TActionBarStyleEx class. Otherwise, if the action band is linked to an action manager, the property defaults to the ColorMap as specified by the Style property of the action manager.

Note: To centrally set or change the default look and feel of all action band components, use ActionManager's Style property. To centrally control the colors used by all action bands within an application, connect the action bands to a single colormap component.

See Also

TActionManager (see page 432)

TXPColorMap (see page 23)

TStandardColorMap (see page 2)

TTwilightColorMap (see page 13)

Style (see page 556)

5.1.2.2.13 TActionMainMenuBar.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

5.1.2.2.14 TActionMainMenuBar.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

5.1.2.2.15 TActionMainMenuBar.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

5.1.2.2.16 TActionMainMenuBar.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

5.1.2.2.17 TActionMainMenuBar.EdgeBorders Property

Determines which sides of the control have visible borders.

Pascal

```
property EdgeBorders: TEdgeBorders;
```

C++

```
__property TEdgeBorders EdgeBorders;
```

Description

Use the EdgeBorders property to add or remove borders. The appearance of the borders can be set with the EdgeInner and EdgeOuter properties.

See Also

EdgeInner (see page 23287)

EdgeOuter (see page 23288)

5.1.2.2.18 TActionMainMenuBar.EdgeInner Property

Determines the border style for the control's inner edge.

Pascal

```
property EdgeInner: TEdgeStyle;
```

C++

```
__property TEdgeStyle EdgeInner;
```

Description

Use EdgeInner to provide the tool window with a three-dimensional beveled look. A tool window has two borders on each side, an outer border drawn next to the outer edge, and an inner border drawn inside the outer border. EdgeInner specifies the appearance of the inner border. It can be raised (esRaised), lowered (esLowered), or left off altogether (esNone).

Note: The EdgeInner property has an effect only on the sides specified by the EdgeBorders property.

Use EdgeInner with EdgeOuter to give the component a three-dimensional look.

See Also

EdgeBorders (see page 23287)

EdgeOuter (see page 23288)

5.1.2.2.19 TActionMainMenuBar.EdgeOuter Property

Determines the border style for the control's outer edge.

Pascal

```
property EdgeOuter: TEdgeStyle;
```

C++

```
__property TEdgeStyle EdgeOuter;
```

Description

Use EdgeOuter to provide the tool window with a three-dimensional beveled look. A tool window has two borders on each side, an outer border drawn next to the outer edge, and an inner border drawn inside the outer border. EdgeOuter specifies the appearance of the outer border. It can be raised (esRaised), lowered (esLowered), or left off altogether (esNone).

Note: The EdgeInner property has an effect only on the sides specified by the EdgeBorders property.

Use EdgeOuter with EdgeInner to give the component a three-dimensional look.

See Also

EdgeBorders (see page 23287)

EdgeInner (see page 23287)

5.1.2.2.20 TActionMainMenuBar.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

5.1.2.2.21 TActionMainMenuBar.ExpandDelay Property

Specifies, in milliseconds, the delay period that a menu action band uses before showing hidden or unused items.

Pascal

```
property ExpandDelay: Integer;
```

C++

```
__property int ExpandDelay;
```

Description

Use ExpandDelay to specify, in milliseconds, the delay period that a menu action band uses before showing hidden or unused items. The default is 4000 milliseconds.

5.1.2.2.22 TActionMainMenuBar.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
```

```
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

5.1.2.2.23 TActionMainMenuBar.HorzMargin Property

Specifies the width of the horizontal margins in pixels.

Pascal

```
property HorzMargin: Integer;
```

C++

```
__property int HorzMargin;
```

Description

HorzMargin specifies the width of the action band's left and right horizontal margins in pixels.

See Also

VertMargin (see page 490)

VertSeparator (see page 490)

5.1.2.2.24 TActionMainMenuBar.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground: Boolean;
```

C++

```
__property Boolean ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

5.1.2.2.25 TActionMainMenuBar.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

5.1.2.2.26 TActionMainMenuBar.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

5.1.2.2.27 TActionMainMenuBar.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

5.1.2.2.28 TActionMainMenuBar.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

5.1.2.2.29 TActionMainMenuBar.PersistentHotKeys Property

Specifies whether keyboard accelerator shortcuts are always displayed.

Pascal

```
property PersistentHotKeys: Boolean;
```

C++

```
__property Boolean PersistentHotKeys;
```

Description

Use PersistentHotKeys to specify whether the action band always displays keyboard accelerator shortcuts. Setting PersistentHotKeys to true causes the underline for the keyboard shortcut key to always appear. If PersistentHotKeys is set to false, the underline for the keyboard shortcut key is only displayed if the user has used the keyboard to navigate through the action band's items. If the user is only using the mouse to navigate through the action band's items and PersistentHotKeys is set to false, the underline under the character denoted as the shortcut key is not drawn.

Note: Accelerator shortcuts are specified by preceding the appropriate character in the item's caption with an ampersand.

See Also

Caption (see page 384)

5.1.2.2.30 TActionMainMenuBar.Shadows Property

Refers to the TActionMainMenuBar component's shadow.

Pascal

```
property Shadows: Boolean;
```

C++

```
__property Boolean Shadows;
```

Description

Use Shadows to determine whether or not the menus should display with shadows. The user can disable shadows through the OS, in which case the Shadows setting is ignored.

5.1.2.2.31 TActionMainMenuBar.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

5.1.2.2.32 TActionMainMenuBar.Spacing Property

Determines where the image and text appear on an action band's rendered control that represents the action client.

Pascal

```
property Spacing: Integer;
```

C++

```
__property int Spacing;
```

Description

Spacing is used internally to set the number of pixels that should appear between the image specified in an action client's ImageIndex and the text specified in its Caption property.

See Also

Caption (see page 384)

ImageIndex (see page 385)

5.1.2.2.33 TActionMainMenuBar.UseSystemFont Property

Specifies whether the default system font for menu items is used.

Pascal

```
property UseSystemFont: Boolean;
```

C++

```
__property Boolean UseSystemFont;
```

Description

Use UseSystemFont to specify whether the action band uses the default system menu font when it draws the captions for its items. By default, UseSystemFont is set to true. When the Font property is changed from the default values, UseSystemFont is automatically set to false. Setting UseSystemFont to true causes the font specified in the MenuFont property of the TScreen object to be used instead of the font specified in the action band's Font property.

See Also

TFont (see page 14497)

TextOut (see page 14475)

TextRect (see page 14480)

Assign

5.1.2.2.34 TActionMainMenuBar.VertMargin Property

Specifies the width of the vertical margins in pixels.

Pascal

```
property VertMargin: Integer;
```

C++

```
__property int VertMargin;
```

Description

VertMargin specifies the width of the action band's top and bottom vertical margins in pixels.

See Also

HorzMargin (see page 487)

HorzSeparator (see page 487)

5.1.2.2.35 TActionMainMenuBar.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
```

```
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

5.1.2.2.36 TActionMainMenuBar.WindowMenu Property

Specifies the caption of the window menu.

Pascal

```
property WindowMenu: string;
```

C++

```
__property AnsiString WindowMenu;
```

Description

Specifies the caption of the window menu for MDI window operations, such as Cascade windows. This is used only for MDI applications.

5.1.2.3 TActionMainMenuBar Events

The events of the TActionMainMenuBar class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 625)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 626)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 627)	Occurs when an attempt is made to resize the control.
	OnClick (see page 627)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 629)	Adjust resize constraints.
	OnContextPopup (see page 629)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnControlCreated (see page 630)	Occurs when a control is created on the action band.
	OnDblClick (see page 631)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 631)	Occurs when another control is docked to the control.
	OnDockOver (see page 633)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 635)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 637)	Occurs when the user drags an object over a control.
	OnEndDock (see page 640)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 640)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 641)	Occurs when a control receives the input focus.
	OnEnterMenuLoop (see page 643)	Occurs when an ActionBand menu loop has begun.
	OnExit (see page 644)	Occurs when the input focus shifts away from one control to another.
	OnExitMenuLoop (see page 645)	Occurs when an ActionBand menu loop has ended and the menu is closed.
	OnGetControlClass (see page 645)	Occurs when an item is to be placed on the action band.
	OnGetPopupClass (see page 646)	Occurs prior to an ActionBand popupmenu appearing on screen.
	OnGetSiteInfo (see page 646)	Returns the control's docking information.
	OnMouseActivate (see page 648)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 649)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 651)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 654)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 656)	Occurs when the action band is painted.
	OnPopup (see page 656)	Occurs just before the pop-up menu appears.
	OnResize (see page 657)	Occurs immediately after the control is resized.
	OnStartDock (see page 657)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 658)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 658)	Occurs when the application tries to undock a control that is docked to the windowed control.

5.1.2.3.1 TActionMainMenuBar.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore: TAlignInsertBeforeEvent;
```

C++

```
__property TAlignInsertBeforeEvent OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore (see page 6137)

Align (see page 5798)

TAlignInsertBeforeEvent (see page 6293)

5.1.2.3.2 TActionMainMenuBar.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition: TAlignPositionEvent;
```

C++

```
__property TAlignPositionEvent OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition (see page 6138)

Align (see page 5798)

Anchors (see page 5802)

TAlignInfo (see page 6269)

TAlignPositionEvent (see page 6293)

5.1.2.3.3 TActionMainMenuBar.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize: TCanResizeEvent;
```

C++

```
__property TCanResizeEvent OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize (see page 5853)

SetBounds (see page 5783)

OnResize (see page 5870)

5.1.2.3.4 TActionMainMenuBar.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
```

```
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

5.1.2.3.5 TActionMainMenuBar.OnConstrainedResize Event

Adjust resize constraints.

Pascal

```
property OnConstrainedResize: TConstrainedResizeEvent;
```

C++

```
__property TConstrainedResizeEvent OnConstrainedResize;
```

Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

Note: The OnConstrainedResize handler is called immediately after the OnCanResize handler.

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

Constraints (see page 5812)

OnResize (see page 5870)

CanAutoSize (see page 5741)

5.1.2.3.6 TActionMainMenuBar.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

5.1.2.3.7 TActionMainMenuBar.OnControlCreated Event

Occurs when a control is created on the action band.

Pascal

```
property OnControlCreated: TControlCreatedEvent;
```

C++

```
__property TControlCreatedEvent OnControlCreated;
```

Description

Write an OnControlCreated event handler to enhance or replace the type of UI element that is rendered on the action band.

Sender is the action band component that just created a new control.

AnItem is the new child control that was just created. The OnControlCreated event handler can initialize the properties of this control or replace it with a custom control.

See Also

OnGetControlClass (see page 492)

5.1.2.3.8 TActionMainMenuBar.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

5.1.2.3.9 TActionMainMenuBar.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop: TDockDropEvent;
```

C++

```
__property TDockDropEvent OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
```

```
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
```

```
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockOver (see page 6216)

5.1.2.3.10 TActionMainMenuBar.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver: TDockOverEvent;
```

C++

```
__property TDockOverEvent OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
```

```
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
```

```
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnGetSiteInfo (see page 6223)

TDragDockObject (see page 6009)

5.1.2.3.11 TActionMainMenuBar.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
```

```
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:
```

```
procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

5.1.2.3.12 TActionMainMenuBar.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler

is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;
```

```
procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

5.1.2.3.13 TActionMainMenuBar.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

5.1.2.3.14 TActionMainMenuBar.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

5.1.2.3.15 TActionMainMenuBar.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
```

```
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

5.1.2.3.16 TActionMainMenuBar.OnEnterMenuLoop Event

Occurs when an ActionBand menu loop has begun.

Pascal

```
property OnEnterMenuLoop: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnterMenuLoop;
```

Description

Write an OnEnterMenuLoop event handler to modify the menu as necessary prior to the menu appearing on screen.

5.1.2.3.17 TActionMainMenuBar.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

5.1.2.3.18 TActionMainMenuBar.OnExitMenuLoop Event

Occurs when an ActionBand menu loop has ended and the menu is closed.

Pascal

```
property OnExitMenuLoop: TExitMenuEvent;
```

C++

```
__property TExitMenuEvent OnExitMenuLoop;
```

Description

Write an OnExitMenuLoop event handler to do any additional processing once the menu has closed.

5.1.2.3.19 TActionMainMenuBar.OnGetControlClass Event

Occurs when an item is to be placed on the action band.

Pascal

```
property OnGetControlClass: TGetControlClassEvent;
```

C++

```
__property TGetControlClassEvent OnGetControlClass;
```

Description

Write an OnGetControlClass event handler to specify the class of item that the action band should create to represent a specified action client.

Sender is the action band that needs to generate a child control to represent an action client.

AnItem is the action client that the new control should represent.

ControlClass returns the class reference for the new control. This should be a descendant of TCustomActionControl.

See Also

TCustomActionControl (see page 507)

OnControlCreated (see page 491)

5.1.2.3.20 TActionMainMenuBar.OnGetPopupClass Event

Occurs prior to an ActionBand popupmenu appearing on screen.

Pascal

```
property OnGetPopupClass: TGetPopupClassEvent;
```

C++

```
__property TGetPopupClassEvent OnGetPopupClass;
```

Description

Write an OnGetPopupClass event handler to control which popup menu class is used to display an ActionBand menu.

5.1.2.3.21 TActionMainMenuBar.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo: TGetSiteInfoEvent;
```

C++

```
__property TGetSiteInfoEvent OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/
```

```
void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;
```

```
procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

TDragDockObject (see page 6009)

5.1.2.3.22 TActionMainMenuBar.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

5.1.2.3.23 TActionMainMenuBar.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
```

```
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
```

```
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

5.1.2.3.24 TActionMainMenuBar.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
```

```
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
```

```
      end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

5.1.2.3.25 TActionMainMenuBar.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}
```

```
// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
```

```
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

5.1.2.3.26 TActionMainMenuBar.OnPaint Event

Occurs when the action band is painted.

Pascal

```
property OnPaint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPaint;
```

Description

Write an OnPaint event handler to change the way the action band renders itself.

See Also

Canvas (see page 481)

5.1.2.3.27 TActionMainMenuBar.OnPopup Event

Occurs just before the pop-up menu appears.

Pascal

```
property OnPopup: TMenuPopupEvent;
```

C++

```
__property TMenuPopupEvent OnPopup;
```

Description

Write an OnPopup event handler to take specific action just before the pop-up menu appears.

5.1.2.3.28 TActionMainMenuBar.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

5.1.2.3.29 TActionMainMenuBar.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is

automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

5.1.2.3.30 TActionMainMenuBar.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

5.1.2.3.31 TActionMainMenuBar.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock: TUnDockEvent;
```

C++

```
__property TUnDockEvent OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
```

```
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
```

```
    var Accept: Boolean);
  var
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

5.1.3 TCustomActionMainMenuBar Class

TCustomActionMainMenuBar is the ancestor for the TActionMainMenuBar (see page 585) component.

Class Hierarchy

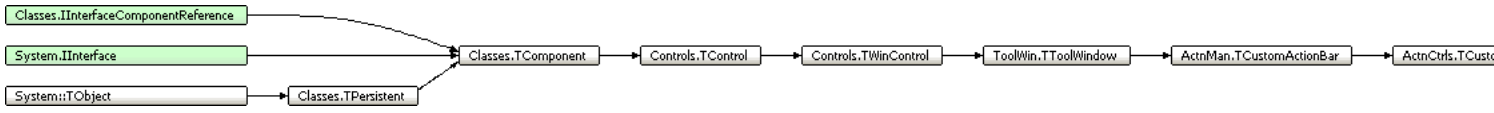

Pascal

```
TCustomActionMainMenuBar = class(TCustomActionMenuBar);
```

C++

```
class TCustomActionMainMenuBar : public TCustomActionMenuBar;
```

File

ActnMenus

Description

TCustomActionMainMenuBar is the ancestor for the TActionMainMenuBar (see page 585) component.

See Also

TActionMainMenuBar (see page 585)

Adding Icons to Menus and Toolbars

Creating Customizable Toolbars and Menus

5.1.3.1 TCustomActionMainMenuBar Members

The following tables list the members exposed by TCustomActionMainMenuBar.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.

	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FCachedMenu (see page 699)	This is FCachedMenu, a member of class TCustomActionMenuBar.
	FChildMenu (see page 699)	This is FChildMenu, a member of class TCustomActionMenuBar.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefaultColorMap (see page 460)	This is FDefaultColorMap, a member of class TCustomActionBar.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExpandTimer (see page 700)	This is FExpandTimer, a member of class TCustomActionMenuBar.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FPopupTimer (see page 700)	This is FPopupTimer, a member of class TCustomActionMenuBar.
	FSavedWrapState (see page 460)	This is FSavedWrapState, a member of class TCustomActionBar.
	FSelectedItem (see page 700)	This is FSelectedItem, a member of class TCustomActionMenuBar.
	FWindowMenuItem (see page 676)	This is FWindowMenuItem, a member of class TCustomActionMainMenuBar.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 462)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	Animate (see page 701)	This is Animate, a member of class TCustomActionMenuBar.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
V	AutoSizingChanged (see page 462)	This is AutoSizingChanged, a member of class TCustomActionBar.
	BackgroundSize (see page 462)	This is BackgroundSize, a member of class TCustomActionBar.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcInsertPosition (see page 462)	This is CalcInsertPosition, a member of class TCustomActionBar.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.

	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CleanupStack (see page 701)	This is CleanupStack, a member of class TCustomActionMenuBar.
	Clear (see page 463)	This is Clear, a member of class TCustomActionBar.
	ClearSubMenus (see page 702)	This is ClearSubMenus, a member of class TCustomActionMenuBar.
	Click (see page 5743)	Respond to user click.
	CMBorderChanged (see page 23285)	This is CMBorderChanged, a member of class TToolWindow.
	CMCtl3DChanged (see page 23286)	This is CMCtl3DChanged, a member of class TToolWindow.
	CMWininichange (see page 678)	This is CMWininichange, a member of class TCustomActionMainMenuBar.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateControl (see page 678)	This is CreateControl, a member of class TCustomActionMainMenuBar.
	CreateControls (see page 678)	This is CreateControls, a member of class TCustomActionMainMenuBar.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreatePopup (see page 702)	This is CreatePopup, a member of class TCustomActionMenuBar.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 463)	Returns placement order of custom-aligned child control.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 678)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 679)	This is DesignWndProc, a member of class TCustomActionMainMenuBar.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoControlCreated (see page 465)	This is DoControlCreated, a member of class TCustomActionBar.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoDropActions (see page 465)	This is DoDropActions, a member of class TCustomActionBar.
	DoDropCategory (see page 465)	This is DoDropCategory, a member of class TCustomActionBar.
	DoDropItem (see page 466)	This is DoDropItem, a member of class TCustomActionBar.

	DoEndDock (see page 146)	OnEndDock event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetControlClass (see page 466)	This is DoGetControlClass, a member of class TCustomActionBar.
	DoGetPopupClass (see page 703)	This is DoGetPopupClass, a member of class TCustomActionMenuBar.
	DoItemClicked (see page 703)	This is DoItemClicked, a member of class TCustomActionMenuBar.
	DoItemKeyed (see page 703)	This is DoItemKeyed, a member of class TCustomActionMenuBar.
	DoItemSelected (see page 703)	This is DoItemSelected, a member of class TCustomActionMenuBar.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMenuDelay (see page 703)	This is DoMenuDelay, a member of class TCustomActionMenuBar.
	DoMouseIdle (see page 704)	This is DoMouseIdle, a member of class TCustomActionMenuBar.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoneMenuLoop (see page 679)	This is DoneMenuLoop, a member of class TCustomActionMainMenuBar.
	DoPopup (see page 704)	This is DoPopup, a member of class TCustomActionMenuBar.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 147)	OnStartDock event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 466)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawBackground (see page 467)	This is DrawBackground, a member of class TCustomActionBar.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	ExecAction (see page 704)	This is ExecAction, a member of class TCustomActionMenuBar.
	Expand (see page 704)	Causes the menu bar to show all items, including the hidden ones.
	ExpandTimer (see page 705)	This is ExpandTimer, a member of class TCustomActionMenuBar.
	FindAccelItem (see page 467)	This is FindAccelItem, a member of class TCustomActionBar.
	FindItemAt (see page 468)	This is FindItemAt, a member of class TCustomActionBar.
	FindItemCaption (see page 468)	This is FindItemCaption, a member of class TCustomActionBar.
	FindLast (see page 468)	This is FindLast, a member of class TCustomActionBar.
	FindLeastUsedItem (see page 469)	This is FindLeastUsedItem, a member of class TCustomActionBar.
	FindNearestControl (see page 469)	This is FindNearestControl, a member of class TCustomActionBar.
	FindNext (see page 470)	This is FindNext, a member of class TCustomActionBar.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FindNextVisibleItem (see page 470)	This is FindNextVisibleItem, a member of class TCustomActionBar.
	FindPrevious (see page 470)	This is FindPrevious, a member of class TCustomActionBar.
	FindPreviousVisibleItem (see page 470)	This is FindPreviousVisibleItem, a member of class TCustomActionBar.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionControl (see page 471)	This is GetActionControl, a member of class TCustomActionBar.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetActiveMDIChild (see page 679)	This is GetActiveMDIChild, a member of class TCustomActionMainMenuBar.

	GetBannerWidth (see page 471)	This is GetBannerWidth, a member of class TCustomActionBar.
	GetBarHeight (see page 471)	This is GetBarHeight, a member of class TCustomActionBar.
	GetBarWidth (see page 471)	This is GetBarWidth, a member of class TCustomActionBar.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlClass (see page 471)	This is GetControlClass, a member of class TCustomActionBar.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDefaultColorMapClass (see page 472)	This is GetDefaultColorMapClass, a member of class TCustomActionBar.
	GetDefaultSounds (see page 705)	This is GetDefaultSounds, a member of class TCustomActionMenuBar.
	GetDesignMode (see page 472)	This is GetDesignMode, a member of class TCustomActionBar.
	GetDesignMode (see page 705)	This is GetDesignMode, a member of class TCustomActionMenuBar.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 147)	Returns the class used for the control when it is floating.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetMouseHoverItem (see page 705)	This is GetMouseHoverItem, a member of class TCustomActionMenuBar.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupClass (see page 706)	This is GetPopupClass, a member of class TCustomActionMenuBar.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetPopupMenuClass (see page 472)	This is GetPopupMenuClass, a member of class TCustomActionBar.
	GetSelected (see page 706)	This is GetSelected, a member of class TCustomActionMenuBar.
	GetSiteInfo (see page 147)	Generates an OnGetSiteInfo event
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasItems (see page 472)	This is HasItems, a member of class TCustomActionBar.
	Idle (see page 706)	This is Idle, a member of class TCustomActionMenuBar.
	InitMenuLoop (see page 706)	This is InitMenuLoop, a member of class TCustomActionMenuBar.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsDesignMsg (see page 707)	This is IsDesignMsg, a member of class TCustomActionMenuBar.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.

	ItemCount (see page 473)	This is ItemCount, a member of class TCustomActionBar.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 148)	Initializes the control after it is loaded from a stream.
	Loaded (see page 473)	Initializes the control after it is loaded from a stream.
	Loaded (see page 707)	Initializes the control after it is loaded from a stream.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndHook (see page 680)	This is MainWndHook, a member of class TCustomActionMainMenuBar.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MDIChildMaximized (see page 680)	This is MDIChildMaximized, a member of class TCustomActionMainMenuBar.
	MDIChildRestored (see page 680)	This is MDIChildRestored, a member of class TCustomActionMainMenuBar.
	MouseActivate (see page 707)	This is MouseActivate, a member of class TCustomActionMenuBar.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	NCPaint (see page 473)	This is NCPaint, a member of class TCustomActionBar.
	NCPaint (see page 23286)	This is NCPaint, a member of class TToolWindow.
	NewPopup (see page 680)	This is NewPopup, a member of class TCustomActionMainMenuBar.
	Notification (see page 681)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 474)	This is Paint, a member of class TCustomActionBar.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 474)	Renders the image of a windowed control.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	ProcessMenuLoop (see page 708)	This is ProcessMenuLoop, a member of class TCustomActionMenuBar.
	ProcessMessage (see page 708)	This is ProcessMessage, a member of class TCustomActionMenuBar.
	ProcessMessages (see page 709)	This is ProcessMessages, a member of class TCustomActionMenuBar.
	ProcessMouseMsg (see page 681)	This is ProcessMouseMsg, a member of class TCustomActionMainMenuBar.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	RefreshMDIMenu (see page 681)	This is RefreshMDIMenu, a member of class TCustomActionMainMenuBar.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Reset (see page 475)	This is Reset, a member of class TCustomActionBar.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.

	Select (see page 709)	This is Select, a member of class TCustomActionMenuBar.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectItem (see page 710)	This is SelectItem, a member of class TCustomActionMenuBar.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
V	SetActionClient (see page 475)	This is SetActionClient, a member of class TCustomActionBar.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetAutoSizing (see page 475)	This is SetAutoSizing, a member of class TCustomActionBar.
V	SetBiDiMode (see page 475)	Sets the control's bi-directional mode.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
V	SetColorMap (see page 476)	This is SetColorMap, a member of class TCustomActionBar.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDesignMode (see page 681)	Specifies whether the whether the action band is being customized or being used in the IDE.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetOrientation (see page 477)	This is SetOrientation, a member of class TCustomActionBar.
V	SetParent (see page 710)	Sets the parent of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetPersistentHotKeys (see page 477)	This is SetPersistentHotKeys, a member of class TCustomActionBar.
V	SetPersistentHotKeys (see page 711)	This is SetPersistentHotKeys, a member of class TCustomActionMenuBar.
V	SetSpacing (see page 477)	This is SetSpacing, a member of class TCustomActionBar.
	SetupDefaultColorMap (see page 477)	This is SetupDefaultColorMap, a member of class TCustomActionBar.
	SetupWindowMenu (see page 682)	This is SetupWindowMenu, a member of class TCustomActionMainMenuBar.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.

V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	VisibleChanging (see page 478)	Provides the interface for a method that responds to changes in the control's visibility.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMKeyDown (see page 682)	This is WMKeyDown, a member of class TCustomActionMainMenuBar.
	WMNCCalcSize (see page 23286)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCHitTest (see page 682)	This is WMNCHitTest, a member of class TCustomActionMainMenuBar.
	WMNCPaint (see page 23286)	This is WMNCPaint, a member of class TToolWindow.
	WMSysCommand (see page 683)	This is WMSysCommand, a member of class TCustomActionMainMenuBar.
	WMSysKeyDown (see page 683)	This is WMSysKeyDown, a member of class TCustomActionMainMenuBar.
	WMSysKeyUp (see page 683)	This is WMSysKeyUp, a member of class TCustomActionMainMenuBar.
V	WndProc (see page 711)	Provides specific message responses for the control.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AnimateDuration (see page 716)	This is AnimateDuration, a member of class TCustomActionMenuBar.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DelayItem (see page 717)	This is DelayItem, a member of class TCustomActionMenuBar.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Expandable (see page 717)	This is Expandable, a member of class TCustomActionMenuBar.
	Expanded (see page 718)	This is Expanded, a member of class TCustomActionMenuBar.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
R	HRowCount (see page 488)	Indicates the number of columns of items that the action band displays.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.

R	Items (see page 488)	Lists the action clients that correspond to the child controls of this action band.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	MouseControl (see page 718)	This is MouseControl, a member of class TCustomActionMenuBar.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentMenu (see page 719)	This is ParentMenu, a member of class TCustomActionMenuBar.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
R	PopupStack (see page 719)	This is PopupStack, a member of class TCustomActionMenuBar.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
R	Selected (see page 719)	This is Selected, a member of class TCustomActionMenuBar.
	Text (see page 5842)	Contains a text string associated with the control.
R	VRowCount (see page 490)	Indicates the number of rows of items that the action band displays.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
R	Widest (see page 491)	This is Widest, a member of class TCustomActionBar.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnControlCreated (see page 491)	Occurs when a control is created on the action band.
	OnEnterMenuLoop (see page 720)	Occurs when an ActionBand menu loop has begun.
	OnExitMenuLoop (see page 721)	Occurs when an ActionBand menu loop has ended and the menu is closed.
	OnGetControlClass (see page 492)	Occurs when an item is to be placed on the action band.
	OnGetPopupClass (see page 721)	Occurs prior to an ActionBand popupmenu appearing on screen.
	OnPaint (see page 492)	Occurs when the action band is painted.
	OnPopup (see page 721)	Occurs just before the pop-up menu appears.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
V	CalcDockedWidth (see page 146)	This is CalcDockedWidth, a member of class TCustomActionDockBar.
V	CanFocus (see page 677)	Indicates whether a TActionMainMenuBar (see page 585) component can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	CloseMenu (see page 677)	Closes the menu currently being displayed.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 675)	Creates and initializes a TCustomActionMainMenuBar (see page 661) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 676)	Destroys an instance of TCustomActionMainMenuBar (see page 661).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Dock (see page 146)	Used internally to dock the action band.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
V	DragDrop (see page 466)	Generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
V	FindFirst (see page 467)	Finds the first TActionClientItem (see page 374) in the Items (see page 488) list.
	FindFirstVisibleItem (see page 468)	Finds the first visible TActionClientItem (see page 374) in the Items (see page 488) list.
	FindLastVisibleItem (see page 469)	Finds the last visibleTActionClientItem in the Items (see page 488) list.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.

	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
V	RecreateControls (see page 475)	Destroys and recreates all of the action band's action client controls.
V	RecreateControls (see page 709)	Destroys and recreates all of the action band's action client controls.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.

	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 476)	Sets the bounds based on whether the popup is expanded or not
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Style (see page 477)	Identifies the style object that gives the application its default look and feel.
V	Style (see page 711)	Retrieves the style used by the ActionBand menu that started the menu loop so the submenus have a consistent appearance.
V	TrackMenu (see page 682)	Executes the action associated with the menu item.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	ActionClient (see page 479)	Specifies the action client associated with this action band.
R	ActionControls (see page 479)	Lists the controls on the action band.
	ActionManager (see page 480)	Specifies the action manager component that is associated with the action band.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AllowHiding (see page 480)	Determines whether the action band can be hidden.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AnimationStyle (see page 716)	Specifies the type of animation the menu bar will display as it opens.
	AutoFocus (see page 717)	This is AutoFocus, a member of class TCustomActionMenuBar.
	AutoSizing (see page 480)	Specifies whether the action band adjusts its size automatically to fit around its child controls.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 481)	Represents the drawing surface of the action band.
	Caption (see page 481)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 483)	Specifies the background color of the control.
	ColorMap (see page 484)	Indicates the colormap for individual action band components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ContextBar (see page 485)	Indicates whether items shown in this action band represent the items in a popup menu.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designable (see page 485)	Specifies whether the action band can be customized (used internally by the IDE).
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignMode (see page 486)	Specifies whether the action band is being customized or being used in the IDE.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EdgeBorders (see page 23287)	Determines which sides of the control have visible borders.
	EdgeInner (see page 23287)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 23288)	Determines the border style for the control's outer edge.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
	ExpandDelay (see page 717)	Specifies, in milliseconds, the delay period that a menu action band uses before showing hidden or unused items.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Font (see page 486)	Controls the attributes of text written on or in the control.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HorzMargin (see page 487)	Specifies the width of the horizontal margins in pixels.
	HorzSeparator (see page 487)	Indicates whether a horizontal separator bar appears between item lines.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	Inactive (see page 683)	Specifies whether the action band is disabled or enabled.
	InMenuLoop (see page 718)	Indicates whether items in the menu can be selected.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	Orientation (see page 488)	Specifies how the action band lays out its child items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentControl (see page 718)	Refers to the parent object of the TCustomActionBarControl.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PersistentHotKeys (see page 684)	Specifies whether keyboard accelerator shortcuts are always displayed.
	RootMenu (see page 719)	Refers to the root menu of the ActionBand menu.
	Shadows (see page 684)	Refers to the TActionMainMenuBar (see page 585) component's shadow.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.

R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	Spacing (see page 489)	Determines where the image and text appear on an action band's rendered control that represents the action client.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	UseSystemFont (see page 720)	Specifies whether the default system font for menu items is used.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	VertMargin (see page 490)	Specifies the width of the vertical margins in pixels.
	VertSeparator (see page 490)	Indicates whether a vertical separator bar should appear between item lines.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.
	WindowMenu (see page 684)	Specifies the caption of the window menu.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

5.1.3.2 TCustomActionMainMenuBar.Create Constructor

Creates and initializes a TCustomActionMainMenuBar (see page 661) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomActionMainMenuBar(TComponent * AOwner);
```

Description

Create generates a TCustomActionMainMenuBar (see page 661) and initializes its properties.

```
__fastcall virtual TCustomActionMainMenuBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall TCustomActionMainMenuBar (HWND ParentWindow) :
Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

5.1.3.3 TCustomActionMainMenuBar.Destroy Destructor

Destroys an instance of TCustomActionMainMenuBar (see page 661).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomActionMainMenuBar();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TCustomActionMainMenuBar (see page 661) reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

Destroy (see page 5732)

5.1.3.4 TCustomActionMainMenuBar Fields

The fields of the TCustomActionMainMenuBar class are listed here.

Protected Fields

	Name	Description
	FWindowMenuItem (see page 676)	This is FWindowMenuItem, a member of class TCustomActionMainMenuBar.

5.1.3.4.1 TCustomActionMainMenuBar.FWindowMenuItem Field

Pascal

```
FWindowMenuItem: TActionClientItem;
```

C++

```
TActionClientItem FWindowMenuItem;
```

Description

This is FWindowMenuItem, a member of class TCustomActionMainMenuBar.

5.1.3.5 TCustomActionMainMenuBar Methods

The methods of the TCustomActionMainMenuBar class are listed here.

Protected Methods

	Name	Description
	CMWininichange (see page 678)	This is CMWininichange, a member of class TCustomActionMainMenuBar.
V	CreateControl (see page 678)	This is CreateControl, a member of class TCustomActionMainMenuBar.
V	CreateControls (see page 678)	This is CreateControls, a member of class TCustomActionMainMenuBar.
V	CustomAlignPosition (see page 678)	Returns aligned position of custom-aligned child control.
V	DesignWndProc (see page 679)	This is DesignWndProc, a member of class TCustomActionMainMenuBar.
V	DoneMenuLoop (see page 679)	This is DoneMenuLoop, a member of class TCustomActionMainMenuBar.

	GetActiveMDIChild (see page 679)	This is GetActiveMDIChild, a member of class TCustomActionMainMenuBar.
	MainWndHook (see page 680)	This is MainWndHook, a member of class TCustomActionMainMenuBar.
	MDIChildMaximized (see page 680)	This is MDIChildMaximized, a member of class TCustomActionMainMenuBar.
	MDIChildRestored (see page 680)	This is MDIChildRestored, a member of class TCustomActionMainMenuBar.
V	NewPopup (see page 680)	This is NewPopup, a member of class TCustomActionMainMenuBar.
V	Notification (see page 681)	Responds to notifications that components are being created or destroyed.
V	ProcessMouseMsg (see page 681)	This is ProcessMouseMsg, a member of class TCustomActionMainMenuBar.
	RefreshMDIMenu (see page 681)	This is RefreshMDIMenu, a member of class TCustomActionMainMenuBar.
V	SetDesignMode (see page 681)	Specifies whether the whether the action band is being customized or being used in the IDE.
	SetupWindowMenu (see page 682)	This is SetupWindowMenu, a member of class TCustomActionMainMenuBar.
	WMKeyDown (see page 682)	This is WMKeyDown, a member of class TCustomActionMainMenuBar.
	WMNCHitTest (see page 682)	This is WMNCHitTest, a member of class TCustomActionMainMenuBar.
	WMSysCommand (see page 683)	This is WMSysCommand, a member of class TCustomActionMainMenuBar.
	WMSysKeyDown (see page 683)	This is WMSysKeyDown, a member of class TCustomActionMainMenuBar.
	WMSysKeyUp (see page 683)	This is WMSysKeyUp, a member of class TCustomActionMainMenuBar.

Public Methods

	Name	Description
V	CanFocus (see page 677)	Indicates whether a TActionMainMenuBar (see page 585) component can receive focus.
V	CloseMenu (see page 677)	Closes the menu currently being displayed.
V	TrackMenu (see page 682)	Executes the action associated with the menu item.

5.1.3.5.1 TCustomActionMainMenuBar.CanFocus Method

Indicates whether a TActionMainMenuBar (see page 585) component can receive focus.

Pascal

```
function CanFocus: Boolean; override;
```

C++

```
virtual __fastcall Boolean CanFocus();
```

Description

By default, TActionMainMenuBar (see page 585) cannot receive input focus and should not participate in tab order. Therefore, CanFocus always returns false.

See Also

Enabled (see page 5821)

Parent (see page 5832)

Visible (see page 5847)

5.1.3.5.2 TCustomActionMainMenuBar.CloseMenu Method

Closes the menu currently being displayed.

Pascal

```
procedure CloseMenu; override;
```

C++

```
virtual __fastcall CloseMenu();
```

Description

CloseMenu is called to close the menu currently being displayed, including any submenus that may be open.

5.1.3.5.3 TCustomActionMainMenuBar.CMWininichange Method

Pascal

```
procedure CMWininichange(var Message: TWMWinIniChange); message CM_WININICHANGE;
```

C++

```
__fastcall CMWininichange(TWMWinIniChange Message);
```

Description

This is CMWininichange, a member of class TCustomActionMainMenuBar.

5.1.3.5.4 TCustomActionMainMenuBar.CreateControl Method

Pascal

```
function CreateControl(AnItem: TActionClientItem): TCustomActionControl; override;
```

C++

```
virtual __fastcall TCustomActionControl CreateControl(TActionClientItem AnItem);
```

Description

This is CreateControl, a member of class TCustomActionMainMenuBar.

5.1.3.5.5 TCustomActionMainMenuBar.CreateControls Method

Pascal

```
procedure CreateControls; override;
```

C++

```
virtual __fastcall CreateControls();
```

Description

This is CreateControls, a member of class TCustomActionMainMenuBar.

5.1.3.5.6 TCustomActionMainMenuBar.CustomAlignPosition Method

Returns aligned position of custom-aligned child control.

Pascal

```
procedure CustomAlignPosition(Control: TControl; var NewLeft: Integer; var NewTop: Integer;
var NewWidth: Integer; var NewHeight: Integer; var AlignRect: TRect; AlignInfo: TAlignInfo);
override;
```

C++

```
virtual __fastcall CustomAlignPosition(TControl * Control, int NewLeft, int NewTop, int
NewWidth, int NewHeight, TRect AlignRect, TAlignInfo AlignInfo);
```

Description

CustomAlignPosition determines the specific position of custom-aligned child controls. CustomAlignPosition is called once for each child control with an Align property of **alCustom**, in an order previously determined by calls to CustomAlignInsertBefore.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition.

These are the parameters:

Control	The child control being placed.
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. CustomAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. CustomAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, CustomAlignPosition does nothing.	

See Also

CustomAlignInsertBefore (see page 6137)

Align (see page 5798)

OnAlignPosition (see page 6214)

TAlignInfo (see page 6269)

5.1.3.5.7 TCustomActionMainMenuBar.DesignWndProc Method

Pascal

```
function DesignWndProc(var Message: TMessage): Boolean; override;
```

C++

```
virtual __fastcall Boolean DesignWndProc(TMessage Message);
```

Description

This is DesignWndProc, a member of class TCustomActionMainMenuBar.

5.1.3.5.8 TCustomActionMainMenuBar.DoneMenuLoop Method

Pascal

```
procedure DoneMenuLoop; override;
```

C++

```
virtual __fastcall DoneMenuLoop();
```

Description

This is DoneMenuLoop, a member of class TCustomActionMainMenuBar.

5.1.3.5.9 TCustomActionMainMenuBar.GetActiveMDIChild Method

Pascal

```
procedure GetActiveMDIChild;
```

C++

```
__fastcall GetActiveMDIChild();
```

Description

This is GetActiveMDIChild, a member of class TCustomActionMainMenuBar.

5.1.3.5.10 TCustomActionMainMenuBar.MainWndHook Method

Pascal

```
function MainWndHook(var Message: TMessage): Boolean;
```

C++

```
__fastcall Boolean MainWndHook(TMessage Message);
```

Description

This is MainWndHook, a member of class TCustomActionMainMenuBar.

5.1.3.5.11 TCustomActionMainMenuBar.MDIChildMaximized Method

Pascal

```
procedure MDIChildMaximized(ChildForm: TCustomForm);
```

C++

```
__fastcall MDIChildMaximized(TCustomForm ChildForm);
```

Description

This is MDIChildMaximized, a member of class TCustomActionMainMenuBar.

5.1.3.5.12 TCustomActionMainMenuBar.MDIChildRestored Method

Pascal

```
procedure MDIChildRestored;
```

C++

```
__fastcall MDIChildRestored();
```

Description

This is MDIChildRestored, a member of class TCustomActionMainMenuBar.

5.1.3.5.13 TCustomActionMainMenuBar.NewPopup Method

Pascal

```
function NewPopup: TCustomActionPopupMenu; override;
```

C++

```
virtual __fastcall TCustomActionPopupMenu NewPopup();
```

Description

This is NewPopup, a member of class TCustomActionMainMenuBar.

5.1.3.5.14 TCustomActionMainMenuBar.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

5.1.3.5.15 TCustomActionMainMenuBar.ProcessMouseMsg Method

Pascal

```
procedure ProcessMouseMsg(var Msg: tagMSG); override;
```

C++

```
virtual __fastcall ProcessMouseMsg(tagMSG Msg);
```

Description

This is ProcessMouseMsg, a member of class TCustomActionMainMenuBar.

5.1.3.5.16 TCustomActionMainMenuBar.RefreshMDIMenu Method

Pascal

```
procedure RefreshMDIMenu;
```

C++

```
__fastcall RefreshMDIMenu();
```

Description

This is RefreshMDIMenu, a member of class TCustomActionMainMenuBar.

5.1.3.5.17 TCustomActionMainMenuBar.SetDesignMode Method

Specifies whether the whether the action band is being customized or being used in the IDE.

Pascal

```
procedure SetDesignMode(const Value: Boolean); override;
```

C++

```
virtual __fastcall SetDesignMode(const Boolean Value);
```

Description

SetDesignMode is used internally to determine whether the action band is being customized or being used within the IDE.

5.1.3.5.18 TCustomActionMainMenuBar.SetupWindowMenu Method

Pascal

```
procedure SetupWindowMenu;
```

C++

```
__fastcall SetupWindowMenu();
```

Description

This is SetupWindowMenu, a member of class TCustomActionMainMenuBar.

5.1.3.5.19 TCustomActionMainMenuBar.TrackMenu Method

Executes the action associated with the menu item.

Pascal

```
procedure TrackMenu; override;
```

C++

```
virtual __fastcall TrackMenu();
```

Description

TrackMenu is called internally to execute the action associated with the selected menu item..

5.1.3.5.20 TCustomActionMainMenuBar.WMKeyDown Method

Pascal

```
procedure WMKeyDown(var Message: TWMKeyDown); message WM_KEYDOWN;
```

C++

```
__fastcall WMKeyDown(TWMKeyDown Message);
```

Description

This is WMKeyDown, a member of class TCustomActionMainMenuBar.

5.1.3.5.21 TCustomActionMainMenuBar.WMNCHitTest Method

Pascal

```
procedure WMNCHitTest(var Message: TWMNCHitTest); message WM_NCHITTEST;
```

C++

```
__fastcall WMNCHitTest(TWMNCHitTest Message);
```

Description

This is WMNCHitTest, a member of class TCustomActionMainMenuBar.

5.1.3.5.22 TCustomActionMainMenuBar.WMSysCommand Method

Pascal

```
procedure WMSysCommand(var Message: TWMSysCommand); message WM_SYSCOMMAND;
```

C++

```
__fastcall WMSysCommand(TWMSysCommand Message);
```

Description

This is WMSysCommand, a member of class TCustomActionMainMenuBar.

5.1.3.5.23 TCustomActionMainMenuBar.WMSysKeyDown Method

Pascal

```
procedure WMSysKeyDown(var Message: TWMSysKeyDown); message WM_SYSKEYDOWN;
```

C++

```
__fastcall WMSysKeyDown(TWMSysKeyDown Message);
```

Description

This is WMSysKeyDown, a member of class TCustomActionMainMenuBar.

5.1.3.5.24 TCustomActionMainMenuBar.WMSysKeyUp Method

Pascal

```
procedure WMSysKeyUp(var Message: TWMSysKeyUp); message WM_SYSKEYUP;
```

C++

```
__fastcall WMSysKeyUp(TWMSysKeyUp Message);
```

Description

This is WMSysKeyUp, a member of class TCustomActionMainMenuBar.

5.1.3.6 TCustomActionMainMenuBar Properties

The properties of the TCustomActionMainMenuBar class are listed here.

Public Properties

	Name	Description
	Inactive (see page 683)	Specifies whether the action band is disabled or enabled.
	PersistentHotKeys (see page 684)	Specifies whether keyboard accelerator shortcuts are always displayed.
	Shadows (see page 684)	Refers to the TActionMainMenuBar (see page 585) component's shadow.
	WindowMenu (see page 684)	Specifies the caption of the window menu.

5.1.3.6.1 TCustomActionMainMenuBar.Inactive Property

Specifies whether the action band is disabled or enabled.

Pascal

```
property Inactive: Boolean;
```

C++

```
__property Boolean Inactive;
```

Description

Inactive is used internally to determine whether the action band is enabled or disabled. When the state changes, Inactive calls Invalidate to force a repaint of the action band.

5.1.3.6.2 TCustomActionMainMenuBar.PersistentHotKeys Property

Specifies whether keyboard accelerator shortcuts are always displayed.

Pascal

```
property PersistentHotKeys: Boolean;
```

C++

```
__property Boolean PersistentHotKeys;
```

Description

Use PersistentHotKeys to specify whether the action band always displays keyboard accelerator shortcuts. Setting PersistentHotKeys to true causes the underline for the keyboard shortcut key to always appear. If PersistentHotKeys is set to false, the underline for the keyboard shortcut key is only displayed if the user has used the keyboard to navigate through the action band's items. If the user is only using the mouse to navigate through the action band's items and PersistentHotKeys is set to false, the underline under the character denoted as the shortcut key is not drawn.

Note: Accelerator shortcuts are specified by preceding the appropriate character in the item's caption with an ampersand.

See Also

Caption (see page 384)

5.1.3.6.3 TCustomActionMainMenuBar.Shadows Property

Refers to the TActionMainMenuBar (see page 585) component's shadow.

Pascal

```
property Shadows: Boolean;
```

C++

```
__property Boolean Shadows;
```

Description

Use Shadows to determine whether or not the menus should display with shadows. The user can disable shadows through the OS, in which case the Shadows setting is ignored.

5.1.3.6.4 TCustomActionMainMenuBar.WindowMenu Property

Specifies the caption of the window menu.

Pascal

```
property WindowMenu: string;
```

C++

```
__property AnsiString WindowMenu;
```

Description

Specifies the caption of the window menu for MDI window operations, such as Cascade windows. This is used only for MDI applications.

5.1.4 TCustomActionMenuBar Class

TCustomActionMenuBar is the ancestor for action bands that render actions as menu item UI elements.

Class Hierarchy

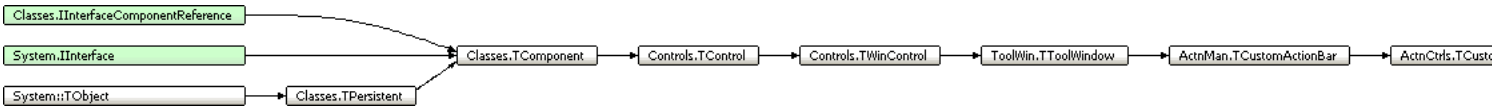

Pascal

```
TCustomActionMenuBar = class(TCustomActionDockBar);
```

C++

```
class TCustomActionMenuBar : public TCustomActionDockBar;
```

File

ActnMenus

Description

TCustomActionMenuBar is the ancestor for all action bands that render actions as menu item UI elements.

See Also

TActionToolBar (see page 38)

TActionMainMenuBar (see page 585)

5.1.4.1 TCustomActionMenuBar Members

The following tables list the members exposed by TCustomActionMenuBar.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.

	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FCachedMenu (see page 699)	This is FCachedMenu, a member of class TCustomActionMenuBar.
	FChildMenu (see page 699)	This is FChildMenu, a member of class TCustomActionMenuBar.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefaultColorMap (see page 460)	This is FDefaultColorMap, a member of class TCustomActionBar.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExpandTimer (see page 700)	This is FExpandTimer, a member of class TCustomActionMenuBar.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FPopupTimer (see page 700)	This is FPopupTimer, a member of class TCustomActionMenuBar.
	FSavedWrapState (see page 460)	This is FSavedWrapState, a member of class TCustomActionBar.
	FSelectedItem (see page 700)	This is FSelectedItem, a member of class TCustomActionMenuBar.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 462)	Aligns any controls for which the control is the parent within a specified area of the control.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	Animate (see page 701)	This is Animate, a member of class TCustomActionMenuBar.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	AutoSizingChanged (see page 462)	This is AutoSizingChanged, a member of class TCustomActionBar.
	BackgroundSize (see page 462)	This is BackgroundSize, a member of class TCustomActionBar.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcInsertPosition (see page 462)	This is CalcInsertPosition, a member of class TCustomActionBar.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CleanupStack (see page 701)	This is CleanupStack, a member of class TCustomActionMenuBar.
	Clear (see page 463)	This is Clear, a member of class TCustomActionBar.
	ClearSubMenus (see page 702)	This is ClearSubMenus, a member of class TCustomActionMenuBar.
	Click (see page 5743)	Respond to user click.
	CMBorderChanged (see page 23285)	This is CMBorderChanged, a member of class TToolWindow.
	CMCtl3DChanged (see page 23286)	This is CMCtl3DChanged, a member of class TToolWindow.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateControl (see page 702)	This is CreateControl, a member of class TCustomActionMenuBar.
	CreateControls (see page 463)	This is CreateControls, a member of class TCustomActionBar.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreatePopup (see page 702)	This is CreatePopup, a member of class TCustomActionMenuBar.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 463)	Returns placement order of custom-aligned child control.

	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 464)	Returns aligned position of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 465)	This is DesignWndProc, a member of class TCustomActionBar.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoControlCreated (see page 465)	This is DoControlCreated, a member of class TCustomActionBar.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoDropActions (see page 465)	This is DoDropActions, a member of class TCustomActionBar.
	DoDropCategory (see page 465)	This is DoDropCategory, a member of class TCustomActionBar.
	DoDropItem (see page 466)	This is DoDropItem, a member of class TCustomActionBar.
	DoEndDock (see page 146)	OnEndDock event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetControlClass (see page 466)	This is DoGetControlClass, a member of class TCustomActionBar.
	DoGetPopupClass (see page 703)	This is DoGetPopupClass, a member of class TCustomActionMenuBar.
	DoItemClicked (see page 703)	This is DoItemClicked, a member of class TCustomActionMenuBar.
	DoItemKeyed (see page 703)	This is DoItemKeyed, a member of class TCustomActionMenuBar.
	DoItemSelected (see page 703)	This is DoItemSelected, a member of class TCustomActionMenuBar.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMenuDelay (see page 703)	This is DoMenuDelay, a member of class TCustomActionMenuBar.
	DoMouseIdle (see page 704)	This is DoMouseIdle, a member of class TCustomActionMenuBar.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoneMenuLoop (see page 704)	This is DoneMenuLoop, a member of class TCustomActionMenuBar.
	DoPopup (see page 704)	This is DoPopup, a member of class TCustomActionMenuBar.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 147)	OnStartDock event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.

	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 466)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawBackground (see page 467)	This is DrawBackground, a member of class TCustomActionBar.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	ExecAction (see page 704)	This is ExecAction, a member of class TCustomActionMenuBar.
V	Expand (see page 704)	Causes the menu bar to show all items, including the hidden ones.
	ExpandTimer (see page 705)	This is ExpandTimer, a member of class TCustomActionMenuBar.
	FindAccelItem (see page 467)	This is FindAccelItem, a member of class TCustomActionBar.
	FindItemAt (see page 468)	This is FindItemAt, a member of class TCustomActionBar.
	FindItemCaption (see page 468)	This is FindItemCaption, a member of class TCustomActionBar.
V	FindLast (see page 468)	This is FindLast, a member of class TCustomActionBar.
	FindLeastUsedItem (see page 469)	This is FindLeastUsedItem, a member of class TCustomActionBar.
	FindNearestControl (see page 469)	This is FindNearestControl, a member of class TCustomActionBar.
V	FindNext (see page 470)	This is FindNext, a member of class TCustomActionBar.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FindNextVisibleItem (see page 470)	This is FindNextVisibleItem, a member of class TCustomActionBar.
V	FindPrevious (see page 470)	This is FindPrevious, a member of class TCustomActionBar.
	FindPreviousVisibleItem (see page 470)	This is FindPreviousVisibleItem, a member of class TCustomActionBar.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
V	GetActionControl (see page 471)	This is GetActionControl, a member of class TCustomActionBar.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
V	GetBannerWidth (see page 471)	This is GetBannerWidth, a member of class TCustomActionBar.
V	GetBarHeight (see page 471)	This is GetBarHeight, a member of class TCustomActionBar.
V	GetBarWidth (see page 471)	This is GetBarWidth, a member of class TCustomActionBar.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlClass (see page 471)	This is GetControlClass, a member of class TCustomActionBar.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDefaultColorMapClass (see page 472)	This is GetDefaultColorMapClass, a member of class TCustomActionBar.
	GetDefaultSounds (see page 705)	This is GetDefaultSounds, a member of class TCustomActionMenuBar.
V	GetDesignMode (see page 705)	This is GetDesignMode, a member of class TCustomActionMenuBar.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 147)	Returns the class used for the control when it is floating.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).

	GetMouseHoverItem (see page 705)	This is GetMouseHoverItem, a member of class TCustomActionMenuBar.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupClass (see page 706)	This is GetPopupClass, a member of class TCustomActionMenuBar.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetPopupMenuClass (see page 472)	This is GetPopupMenuClass, a member of class TCustomActionBar.
	GetSelected (see page 706)	This is GetSelected, a member of class TCustomActionMenuBar.
	GetSiteInfo (see page 147)	Generates an OnGetSiteInfo event
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasItems (see page 472)	This is HasItems, a member of class TCustomActionBar.
	Idle (see page 706)	This is Idle, a member of class TCustomActionMenuBar.
	InitMenuLoop (see page 706)	This is InitMenuLoop, a member of class TCustomActionMenuBar.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsDesignMsg (see page 707)	This is IsDesignMsg, a member of class TCustomActionMenuBar.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	ItemCount (see page 473)	This is ItemCount, a member of class TCustomActionBar.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 707)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 707)	This is MouseActivate, a member of class TCustomActionMenuBar.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	NCPaint (see page 473)	This is NCPaint, a member of class TCustomActionBar.
	NCPaint (see page 23286)	This is NCPaint, a member of class TToolWindow.
	NewPopup (see page 707)	This is NewPopup, a member of class TCustomActionMenuBar.
	Notification (see page 708)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 474)	This is Paint, a member of class TCustomActionBar.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 474)	Renders the image of a windowed control.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	ProcessMenuLoop (see page 708)	This is ProcessMenuLoop, a member of class TCustomActionMenuBar.
	ProcessMessage (see page 708)	This is ProcessMessage, a member of class TCustomActionMenuBar.
	ProcessMessages (see page 709)	This is ProcessMessages, a member of class TCustomActionMenuBar.

	ProcessMouseMsg (see page 709)	This is ProcessMouseMsg, a member of class TCustomActionMenuBar.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Reset (see page 475)	This is Reset, a member of class TCustomActionBar.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Select (see page 709)	This is Select, a member of class TCustomActionMenuBar.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectItem (see page 710)	This is SelectItem, a member of class TCustomActionMenuBar.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActionClient (see page 475)	This is SetActionClient, a member of class TCustomActionBar.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetAutoSizing (see page 475)	This is SetAutoSizing, a member of class TCustomActionBar.
	SetBiDiMode (see page 475)	Sets the control's bi-directional mode.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetColorMap (see page 476)	This is SetColorMap, a member of class TCustomActionBar.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignMode (see page 710)	Specifies whether the whether the action band is being customized or being used in the IDE.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetOrientation (see page 477)	This is SetOrientation, a member of class TCustomActionBar.
	SetParent (see page 710)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetPersistentHotKeys (see page 711)	This is SetPersistentHotKeys, a member of class TCustomActionMenuBar.
	SetSpacing (see page 477)	This is SetSpacing, a member of class TCustomActionBar.
	SetupDefaultColorMap (see page 477)	This is SetupDefaultColorMap, a member of class TCustomActionBar.

	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	VisibleChanging (see page 478)	Provides the interface for a method that responds to changes in the control's visibility.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMNCCalcSize (see page 23286)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCPaint (see page 23286)	This is WMNCPaint, a member of class TToolWindow.
V	WndProc (see page 711)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AnimateDuration (see page 716)	This is AnimateDuration, a member of class TCustomActionMenuBar.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DelayItem (see page 717)	This is DelayItem, a member of class TCustomActionMenuBar.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Expandable (see page 717)	This is Expandable, a member of class TCustomActionMenuBar.
	Expanded (see page 718)	This is Expanded, a member of class TCustomActionMenuBar.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.

R	HRowCount (see page 488)	Indicates the number of columns of items that the action band displays.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	Items (see page 488)	Lists the action clients that correspond to the child controls of this action band.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	MouseControl (see page 718)	This is MouseControl, a member of class TCustomActionMenuBar.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentMenu (see page 719)	This is ParentMenu, a member of class TCustomActionMenuBar.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
R	PopupStack (see page 719)	This is PopupStack, a member of class TCustomActionMenuBar.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
R	Selected (see page 719)	This is Selected, a member of class TCustomActionMenuBar.
	Text (see page 5842)	Contains a text string associated with the control.
R	VRowCount (see page 490)	Indicates the number of rows of items that the action band displays.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
R	Widest (see page 491)	This is Widest, a member of class TCustomActionBar.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnControlCreated (see page 491)	Occurs when a control is created on the action band.
	OnEnterMenuLoop (see page 720)	Occurs when an ActionBand menu loop has begun.
	OnExitMenuLoop (see page 721)	Occurs when an ActionBand menu loop has ended and the menu is closed.
	OnGetControlClass (see page 492)	Occurs when an item is to be placed on the action band.
	OnGetPopupClass (see page 721)	Occurs prior to an ActionBand popupmenu appearing on screen.
	OnPaint (see page 492)	Occurs when the action band is painted.
	OnPopup (see page 721)	Occurs just before the pop-up menu appears.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
V	CalcDockedWidth (see page 146)	This is CalcDockedWidth, a member of class TCustomActionDockBar.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.

	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	CloseMenu (see page 702)	Closes the menu currently being displayed.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 698)	Creates and initializes a TCustomActionMenuBar (see page 685) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 698)	Destroys an instance of TCustomActionMenuBar (see page 685).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Dock (see page 146)	Used internally to dock the action band.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
V	DragDrop (see page 466)	Generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
V	FindFirst (see page 467)	Finds the first TActionClientItem (see page 374) in the Items (see page 488) list.
	FindFirstVisibleItem (see page 468)	Finds the first visible TActionClientItem (see page 374) in the Items (see page 488) list.
	FindLastVisibleItem (see page 469)	Finds the last visibleTActionClientItem in the Items (see page 488) list.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
V	RecreateControls (see page 709)	Destroys and recreates all of the action band's action client controls.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.

	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 476)	Sets the bounds based on whether the popup is expanded or not
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Style (see page 711)	Retrieves the style used by the ActionBand menu that started the menu loop so the submenus have a consistent appearance.
V	TrackMenu (see page 711)	Executes the action associated with the menu item.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	ActionClient (see page 479)	Specifies the action client associated with this action band.
R	ActionControls (see page 479)	Lists the controls on the action band.
	ActionManager (see page 480)	Specifies the action manager component that is associated with the action band.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AllowHiding (see page 480)	Determines whether the action band can be hidden.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AnimationStyle (see page 716)	Specifies the type of animation the menu bar will display as it opens.
	AutoFocus (see page 717)	This is AutoFocus, a member of class TCustomActionMenuBar.
	AutoSizing (see page 480)	Specifies whether the action band adjusts its size automatically to fit around its child controls.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 481)	Represents the drawing surface of the action band.
	Caption (see page 481)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 483)	Specifies the background color of the control.
	ColorMap (see page 484)	Indicates the colormap for individual action band components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ContextBar (see page 485)	Indicates whether items shown in this action band represent the items in a popup menu.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designable (see page 485)	Specifies whether the action band can be customized (used internally by the IDE).
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignMode (see page 486)	Specifies whether the action band is being customized or being used in the IDE.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EdgeBorders (see page 23287)	Determines which sides of the control have visible borders.
	EdgeInner (see page 23287)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 23288)	Determines the border style for the control's outer edge.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
	ExpandDelay (see page 717)	Specifies, in milliseconds, the delay period that a menu action band uses before showing hidden or unused items.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Font (see page 486)	Controls the attributes of text written on or in the control.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HorzMargin (see page 487)	Specifies the width of the horizontal margins in pixels.
	HorzSeparator (see page 487)	Indicates whether a horizontal separator bar appears between item lines.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	InMenuLoop (see page 718)	Indicates whether items in the menu can be selected.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	Orientation (see page 488)	Specifies how the action band lays out its child items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentControl (see page 718)	Refers to the parent object of the TCustomActionBarControl.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PersistentHotKeys (see page 489)	Specifies whether keyboard accelerator shortcuts are always displayed.
	RootMenu (see page 719)	Refers to the root menu of the ActionBand menu.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.

	Spacing (see page 489)	Determines where the image and text appear on an action band's rendered control that represents the action client.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	UseSystemFont (see page 720)	Specifies whether the default system font for menu items is used.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	VertMargin (see page 490)	Specifies the width of the vertical margins in pixels.
	VertSeparator (see page 490)	Indicates whether a vertical separator bar should appear between item lines.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

5.1.4.2 TCustomActionMenuBar.Create Constructor

Creates and initializes a TCustomActionMenuBar (see page 685) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomActionMenuBar(TComponent * AOwner);
```

Description

Create generates a TCustomActionMenuBar (see page 685) and initializes its properties.

```
__fastcall virtual TCustomActionMenuBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall TCustomActionMenuBar (HWND ParentWindow) :
Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

5.1.4.3 TCustomActionMenuBar.Destroy Destructor

Destroys an instance of TCustomActionMenuBar (see page 685).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomActionMenuBar();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TCustomActionMenuBar (see page 685) reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

Destroy (see page 5732)

5.1.4.4 TCustomActionMenuBar Fields

The fields of the TCustomActionMenuBar class are listed here.

Protected Fields

	Name	Description
	FCachedMenu (see page 699)	This is FCachedMenu, a member of class TCustomActionMenuBar.
	FChildMenu (see page 699)	This is FChildMenu, a member of class TCustomActionMenuBar.
	FExpandTimer (see page 700)	This is FExpandTimer, a member of class TCustomActionMenuBar.
	FPopupTimer (see page 700)	This is FPopupTimer, a member of class TCustomActionMenuBar.
	FSelectedItem (see page 700)	This is FSelectedItem, a member of class TCustomActionMenuBar.

5.1.4.4.1 TCustomActionMenuBar.FCachedMenu Field

Pascal

```
FCachedMenu: TCustomActionPopupMenu;
```

C++

```
TCustomActionPopupMenu FCachedMenu;
```

Description

This is FCachedMenu, a member of class TCustomActionMenuBar.

5.1.4.4.2 TCustomActionMenuBar.FChildMenu Field

Pascal

```
FChildMenu: TCustomActionMenuBar;
```

C++

```
TCustomActionMenuBar FChildMenu;
```

Description

This is FChildMenu, a member of class TCustomActionMenuBar.

5.1.4.4.3 TCustomActionMenuBar.FExpandTimer Field

Pascal

```
FExpandTimer: TTimer;
```

C++

```
TTimer FExpandTimer;
```

Description

This is FExpandTimer, a member of class TCustomActionMenuBar.

5.1.4.4.4 TCustomActionMenuBar.FPopupTimer Field

Pascal

```
FPopupTimer: TTimer;
```

C++

```
TTimer FPopupTimer;
```

Description

This is FPopupTimer, a member of class TCustomActionMenuBar.

5.1.4.4.5 TCustomActionMenuBar.FSelectedItem Field

Pascal

```
FSelectedItem: TActionClientItem;
```

C++

```
TActionClientItem FSelectedItem;
```

Description

This is FSelectedItem, a member of class TCustomActionMenuBar.

5.1.4.5 TCustomActionMenuBar Methods

The methods of the TCustomActionMenuBar class are listed here.

Protected Methods

	Name	Description
	Animate (see page 701)	This is Animate, a member of class TCustomActionMenuBar.
	CleanupStack (see page 701)	This is CleanupStack, a member of class TCustomActionMenuBar.
	ClearSubMenus (see page 702)	This is ClearSubMenus, a member of class TCustomActionMenuBar.
V	CreateControl (see page 702)	This is CreateControl, a member of class TCustomActionMenuBar.
V	CreatePopup (see page 702)	This is CreatePopup, a member of class TCustomActionMenuBar.
	DoGetPopupClass (see page 703)	This is DoGetPopupClass, a member of class TCustomActionMenuBar.
V	DoItemClicked (see page 703)	This is DoItemClicked, a member of class TCustomActionMenuBar.
V	DoItemKeyed (see page 703)	This is DoItemKeyed, a member of class TCustomActionMenuBar.
	DoItemSelected (see page 703)	This is DoItemSelected, a member of class TCustomActionMenuBar.
	DoMenuDelay (see page 703)	This is DoMenuDelay, a member of class TCustomActionMenuBar.
	DoMouseIdle (see page 704)	This is DoMouseIdle, a member of class TCustomActionMenuBar.
V	DoneMenuLoop (see page 704)	This is DoneMenuLoop, a member of class TCustomActionMenuBar.

	DoPopup (see page 704)	This is DoPopup, a member of class TCustomActionMenuBar.
	ExecAction (see page 704)	This is ExecAction, a member of class TCustomActionMenuBar.
	Expand (see page 704)	Causes the menu bar to show all items, including the hidden ones.
	ExpandTimer (see page 705)	This is ExpandTimer, a member of class TCustomActionMenuBar.
	GetDefaultSounds (see page 705)	This is GetDefaultSounds, a member of class TCustomActionMenuBar.
	GetDesignMode (see page 705)	This is GetDesignMode, a member of class TCustomActionMenuBar.
	GetMouseHoverItem (see page 705)	This is GetMouseHoverItem, a member of class TCustomActionMenuBar.
	GetPopupClass (see page 706)	This is GetPopupClass, a member of class TCustomActionMenuBar.
	GetSelected (see page 706)	This is GetSelected, a member of class TCustomActionMenuBar.
	Idle (see page 706)	This is Idle, a member of class TCustomActionMenuBar.
	InitMenuLoop (see page 706)	This is InitMenuLoop, a member of class TCustomActionMenuBar.
	IsDesignMsg (see page 707)	This is IsDesignMsg, a member of class TCustomActionMenuBar.
	Loaded (see page 707)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 707)	This is MouseActivate, a member of class TCustomActionMenuBar.
	NewPopup (see page 707)	This is NewPopup, a member of class TCustomActionMenuBar.
	Notification (see page 708)	Responds to notifications that components are being created or destroyed.
	ProcessMenuLoop (see page 708)	This is ProcessMenuLoop, a member of class TCustomActionMenuBar.
	ProcessMessage (see page 708)	This is ProcessMessage, a member of class TCustomActionMenuBar.
	ProcessMessages (see page 709)	This is ProcessMessages, a member of class TCustomActionMenuBar.
	ProcessMouseMsg (see page 709)	This is ProcessMouseMsg, a member of class TCustomActionMenuBar.
	Select (see page 709)	This is Select, a member of class TCustomActionMenuBar.
	SelectItem (see page 710)	This is SelectItem, a member of class TCustomActionMenuBar.
	SetDesignMode (see page 710)	Specifies whether the whether the action band is being customized or being used in the IDE.
	SetParent (see page 710)	Sets the parent of the control.
	SetPersistentHotKeys (see page 711)	This is SetPersistentHotKeys, a member of class TCustomActionMenuBar.
	WndProc (see page 711)	Provides specific message responses for the control.

Public Methods

	Name	Description
	CloseMenu (see page 702)	Closes the menu currently being displayed.
	RecreateControls (see page 709)	Destroys and recreates all of the action band's action client controls.
	Style (see page 711)	Retrieves the style used by the ActionBand menu that started the menu loop so the submenus have a consistent appearance.
	TrackMenu (see page 711)	Executes the action associated with the menu item.

5.1.4.5.1 TCustomActionMenuBar.Animate Method

Pascal

```
procedure Animate(Show: Boolean = True);
```

C++

```
__fastcall Animate(Boolean Show = True);
```

Description

This is Animate, a member of class TCustomActionMenuBar.

5.1.4.5.2 TCustomActionMenuBar.CleanupStack Method

Pascal

```
procedure CleanupStack;
```

C++

```
__fastcall CleanupStack();
```

Description

This is CleanupStack, a member of class TCustomActionMenuBar.

5.1.4.5.3 TCustomActionMenuBar.ClearSubMenus Method

Pascal

```
procedure ClearSubMenus;
```

C++

```
__fastcall ClearSubMenus();
```

Description

This is ClearSubMenus, a member of class TCustomActionMenuBar.

5.1.4.5.4 TCustomActionMenuBar.CloseMenu Method

Closes the menu currently being displayed.

Pascal

```
procedure CloseMenu; virtual;
```

C++

```
virtual __fastcall CloseMenu();
```

Description

CloseMenu is called to close the menu currently being displayed.

5.1.4.5.5 TCustomActionMenuBar.CreateControl Method

Pascal

```
function CreateControl(AnItem: TActionClientItem): TCustomActionControl; override;
```

C++

```
virtual __fastcall TCustomActionControl CreateControl(TActionClientItem AnItem);
```

Description

This is CreateControl, a member of class TCustomActionMenuBar.

5.1.4.5.6 TCustomActionMenuBar.CreatePopup Method

Pascal

```
function CreatePopup(AOwner: TCustomActionMenuBar; Item: TCustomActionControl):
TCustomActionPopupMenu; virtual;
```

C++

```
virtual __fastcall TCustomActionPopupMenu CreatePopup(TCustomActionMenuBar AOwner,
TCustomActionControl Item);
```

Description

This is CreatePopup, a member of class TCustomActionMenuBar.

5.1.4.5.7 TCustomActionMenuBar.DoGetPopupClass Method

Pascal

```
function DoGetPopupClass: TCustomPopupClass; dynamic;
```

C++

```
__fastcall TCustomPopupClass DoGetPopupClass();
```

Description

This is DoGetPopupClass, a member of class TCustomActionMenuBar.

5.1.4.5.8 TCustomActionMenuBar.DoItemClicked Method

Pascal

```
function DoItemClicked(AnItem: TCustomActionControl): TActionClientItem; virtual;
```

C++

```
virtual __fastcall TActionClientItem DoItemClicked(TCustomActionControl AnItem);
```

Description

This is DoItemClicked, a member of class TCustomActionMenuBar.

5.1.4.5.9 TCustomActionMenuBar.DoItemKeyed Method

Pascal

```
function DoItemKeyed(AnItem: TCustomActionControl): TActionClientItem; virtual;
```

C++

```
virtual __fastcall TActionClientItem DoItemKeyed(TCustomActionControl AnItem);
```

Description

This is DoItemKeyed, a member of class TCustomActionMenuBar.

5.1.4.5.10 TCustomActionMenuBar.DoItemSelected Method

Pascal

```
procedure DoItemSelected(AnItem: TCustomActionControl);
```

C++

```
__fastcall DoItemSelected(TCustomActionControl AnItem);
```

Description

This is DoItemSelected, a member of class TCustomActionMenuBar.

5.1.4.5.11 TCustomActionMenuBar.DoMenuDelay Method

Pascal

```
procedure DoMenuDelay(Sender: TObject);
```

C++

```
__fastcall DoMenuDelay(TObject * Sender);
```

Description

This is DoMenuDelay, a member of class TCustomActionMenuBar.

5.1.4.5.12 TCustomActionMenuBar.DoMouseIdle Method

Pascal

```
function DoMouseIdle: TControl; deprecated;
```

C++

```
__fastcall TControl * DoMouseIdle();
```

Description

This is DoMouseIdle, a member of class TCustomActionMenuBar.

5.1.4.5.13 TCustomActionMenuBar.DoneMenuLoop Method

Pascal

```
procedure DoneMenuLoop; virtual;
```

C++

```
virtual __fastcall DoneMenuLoop();
```

Description

This is DoneMenuLoop, a member of class TCustomActionMenuBar.

5.1.4.5.14 TCustomActionMenuBar.DoPopup Method

Pascal

```
procedure DoPopup(Item: TCustomActionControl); virtual;
```

C++

```
virtual __fastcall DoPopup(TCustomActionControl Item);
```

Description

This is DoPopup, a member of class TCustomActionMenuBar.

5.1.4.5.15 TCustomActionMenuBar.ExecAction Method

Pascal

```
procedure ExecAction(Action: TContainedAction); virtual;
```

C++

```
virtual __fastcall ExecAction(TContainedAction Action);
```

Description

This is ExecAction, a member of class TCustomActionMenuBar.

5.1.4.5.16 TCustomActionMenuBar.Expand Method

Causes the menu bar to show all items, including the hidden ones.

Pascal

```
procedure Expand(Full: Boolean); virtual;
```

C++

```
virtual __fastcall Expand(Boolean Full);
```

Description

Use Expand to show all of the items in a menu, even if they are marked as hidden. The Full parameter is used to indicate whether or not submenus should display all items as well.

5.1.4.5.17 TCustomActionMenuBar.ExpandTimer Method

Pascal

```
procedure ExpandTimer(Sender: TObject);
```

C++

```
__fastcall ExpandTimer(TObject * Sender);
```

Description

This is ExpandTimer, a member of class TCustomActionMenuBar.

5.1.4.5.18 TCustomActionMenuBar.GetDefaultSounds Method

Pascal

```
procedure GetDefaultSounds; deprecated;
```

C++

```
__fastcall GetDefaultSounds();
```

Description

This is GetDefaultSounds, a member of class TCustomActionMenuBar.

5.1.4.5.19 TCustomActionMenuBar.GetDesignMode Method

Pascal

```
function GetDesignMode: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetDesignMode();
```

Description

This is GetDesignMode, a member of class TCustomActionMenuBar.

5.1.4.5.20 TCustomActionMenuBar.GetMouseHoverItem Method

Pascal

```
function GetMouseHoverItem(Msg: TMsg): TCustomActionControl;
```

C++

```
__fastcall TCustomActionControl GetMouseHoverItem(TMsg Msg);
```

Description

This is GetMouseHoverItem, a member of class TCustomActionMenuBar.

5.1.4.5.21 TCustomActionMenuBar.GetPopupClass Method

Pascal

```
function GetPopupClass: TCustomPopupClass; virtual;
```

C++

```
virtual __fastcall TCustomPopupClass GetPopupClass();
```

Description

This is GetPopupClass, a member of class TCustomActionMenuBar.

5.1.4.5.22 TCustomActionMenuBar.GetSelected Method

Pascal

```
function GetSelected: TActionClientItem;
```

C++

```
__fastcall TActionClientItem GetSelected();
```

Description

This is GetSelected, a member of class TCustomActionMenuBar.

5.1.4.5.23 TCustomActionMenuBar.Idle Method

Pascal

```
procedure Idle(const Msg: TMsg);
```

C++

```
__fastcall Idle(const TMsg Msg);
```

Description

This is Idle, a member of class TCustomActionMenuBar.

5.1.4.5.24 TCustomActionMenuBar.InitMenuLoop Method

Pascal

```
procedure InitMenuLoop; virtual;
```

C++

```
virtual __fastcall InitMenuLoop();
```

Description

This is InitMenuLoop, a member of class TCustomActionMenuBar.

5.1.4.5.25 TCustomActionMenuBar.IsDesignMsg Method

Pascal

```
function IsDesignMsg(var Msg: TMsg): Boolean;
```

C++

```
__fastcall Boolean IsDesignMsg(TMsg Msg);
```

Description

This is IsDesignMsg, a member of class TCustomActionMenuBar.

5.1.4.5.26 TCustomActionMenuBar.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded (see page 2782)

Action (see page 5797)

ActionChange (see page 5737)

5.1.4.5.27 TCustomActionMenuBar.MouseActivate Method

Pascal

```
function MouseActivate(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer;
HitTest: Integer): TMouseActivate; override;
```

C++

```
virtual __fastcall TMouseActivate MouseActivate(TMouseButton Button, TShiftState Shift, int X,
int Y, int HitTest);
```

Description

This is MouseActivate, a member of class TCustomActionMenuBar.

5.1.4.5.28 TCustomActionMenuBar.NewPopup Method

Pascal

```
function NewPopup: TCustomActionPopupMenu; virtual;
```

C++

```
virtual __fastcall TCustomActionPopupMenu NewPopup();
```

Description

This is NewPopup, a member of class TCustomActionMenuBar.

5.1.4.5.29 TCustomActionMenuBar.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

5.1.4.5.30 TCustomActionMenuBar.ProcessMenuLoop Method

Pascal

```
function ProcessMenuLoop: TActionClientItem; virtual;
```

C++

```
virtual __fastcall TActionClientItem ProcessMenuLoop();
```

Description

This is ProcessMenuLoop, a member of class TCustomActionMenuBar.

5.1.4.5.31 TCustomActionMenuBar.ProcessMessage Method

Pascal

```
function ProcessMessage(var Msg: TMsg): Boolean;
```

C++

```
__fastcall Boolean ProcessMessage(TMsg Msg);
```

Description

This is ProcessMessage, a member of class TCustomActionMenuBar.

5.1.4.5.32 TCustomActionMenuBar.ProcessMessages Method

Pascal

```
procedure ProcessMessages;
```

C++

```
__fastcall ProcessMessages();
```

Description

This is ProcessMessages, a member of class TCustomActionMenuBar.

5.1.4.5.33 TCustomActionMenuBar.ProcessMouseMsg Method

Pascal

```
procedure ProcessMouseMsg(var Msg: TMsg); virtual;
```

C++

```
virtual __fastcall ProcessMouseMsg(TMsg Msg);
```

Description

This is ProcessMouseMsg, a member of class TCustomActionMenuBar.

5.1.4.5.34 TCustomActionMenuBar.RecreateControls Method

Destroys and recreates all of the action band's action client controls.

Pascal

```
procedure RecreateControls; override;
```

C++

```
virtual __fastcall RecreateControls();
```

Description

Call RecreateControls to recreate all of the action client child controls contained by the action band from scratch. RecreateControls destroys the action client controls and creates a new component according to the current property settings.

5.1.4.5.35 TCustomActionMenuBar.Select Method

Pascal

```
procedure Select(const Forward: Boolean);
```

C++

```
__fastcall Select(const Boolean Forward);
```

Description

This is Select, a member of class TCustomActionMenuBar.

5.1.4.5.36 TCustomActionMenuBar.SelectItem Method

Pascal

```
procedure SelectItem(AnItem: TCustomActionControl);
```

C++

```
__fastcall SelectItem(TCustomActionControl AnItem);
```

Description

This is SelectItem, a member of class TCustomActionMenuBar.

5.1.4.5.37 TCustomActionMenuBar.SetDesignMode Method

Specifies whether the whether the action band is being customized or being used in the IDE.

Pascal

```
procedure SetDesignMode(const Value: Boolean); override;
```

C++

```
virtual __fastcall SetDesignMode(const Boolean Value);
```

Description

SetDesignMode is used internally to determine whether the action band is being customized or being used within the IDE.

5.1.4.5.38 TCustomActionMenuBar.SetParent Method

Sets the parent of the control.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. Override SetParent to execute additional code when the value of the Parent property changes.

The AParent parameter specifies the new parent of the control.

If the control already has a parent, SetParent removes the control from that parent's list of controls by calling the parent's RemoveControl method. If AParent is not nil (Delphi) or NULL (C++), SetParent calls its InsertControl method to add the control to its list of controls.

The TControl destructor calls SetParent(nil (Delphi) or NULL (C++)) to remove the control from its parent's control list before destroying the component.

See Also

Destroy (see page 5732)

InsertControl (see page 6160)

Parent (see page 5832)

RemoveControl (see page 6172)

5.1.4.5.39 TCustomActionMenuBar.SetPersistentHotKeys Method

Pascal

```
procedure SetPersistentHotKeys(const Value: Boolean); override;
```

C++

```
virtual __fastcall SetPersistentHotKeys(const Boolean Value);
```

Description

This is SetPersistentHotKeys, a member of class TCustomActionMenuBar.

5.1.4.5.40 TCustomActionMenuBar.Style Method

Retrieves the style used by the ActionBand menu that started the menu loop so the submenus have a consistent appearance.

Pascal

```
function Style: TActionBarStyle; override;
```

C++

```
virtual __fastcall TActionBarStyle Style();
```

Description

Use Style to access the Style of the menu which started the menu loop and therefore determines the appearance of submenus.

5.1.4.5.41 TCustomActionMenuBar.TrackMenu Method

Executes the action associated with the menu item.

Pascal

```
procedure TrackMenu; virtual;
```

C++

```
virtual __fastcall TrackMenu();
```

Description

TrackMenu is called internally to execute the action associated with the selected menu item..

5.1.4.5.42 TCustomActionMenuBar.WndProc Method

Provides specific message responses for the control.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

Override WndProc to change the initial Windows message handler for the control. The WindowProc property is initialized to point to the WndProc method.

WndProc for TWinControl overrides the inherited method to define responses for focus, mouse, and keyboard messages. It sends all others to its inherited WndProc.

When overriding WndProc to provide specialized responses to messages, call the inherited WndProc at the end to dispatch any unhandled messages.

C++ Examples:

```
/*
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
*/
class TMyListBoxDescendant : public TListBox
{
__published:    // IDE-managed Components
    void __fastcall SubClassWndProc(Messages::TMessage &Message);
    void __fastcall ToggleSubClass(bool On);
    void __fastcall OnDrawItemProc(
      TWinControl *Control, int Index, const TRect &Rect,
      TOwnerDrawState State);
private:    // User declarations
public:     // User declarations
    __fastcall TMyListBoxDescendant(TComponent* Owner);
};

TForm1 *Form1;
TMyListBoxDescendant *MyListBoxDescendant1;
Graphics::TBitmap *bitmap0;

const WM_STYLEMESSAGE = WM_USER + 2000;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

__fastcall TMyListBoxDescendant::TMyListBoxDescendant(TComponent* Owner)
    : TListBox(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
}

void __fastcall TForm1::SubClassRadioGroup1Click(TObject *Sender)
{
  MyListBoxDescendant1->ToggleSubClass(SubClassRadioGroup1->ItemIndex == 0);
}

void __fastcall TMyListBoxDescendant::SubClassWndProc(Messages::TMessage &Message)
```

```
{
  if (Message.Msg == WM_STYLEMESSAGE)
    Style = (TListBoxStyle)Message.WParam;
  else
    WndProc(Message);
}

void __fastcall TMyListBoxDescendant::ToggleSubClass(bool On)
{
  if (On)
    WindowProc = SubClassWndProc;
  else
    WindowProc = WndProc;
}

void __fastcall TMyListBoxDescendant::OnDrawItemProc(TWinControl *Control, int Index,
      const TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *bitmap; // temporary variable for item's bitmap
  int Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *canvas = ((TListBox *)Control)->Canvas;
  canvas->FillRect(Rect); // clear the rectangle
  bitmap = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
  if (bitmap)
  {
    canvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, bitmap->Width, bitmap->Height),
      bitmap, Bounds(0, 0, bitmap->Width, bitmap->Height), clRed); // render bitmap
    Offset += bitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  canvas->TextOut(Rect.Left + Offset, Rect.Top, ((TListBox *)Control)->Items->Strings[Index]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyListBoxDescendant1 = new TMyListBoxDescendant(Form1);
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Parent = Form1;
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Left =
    SubClassRadioGroup1->Left + SubClassRadioGroup1->Width + 30;;
  MyListBoxDescendant1->Top = SubClassRadioGroup1->Top;
  MyListBoxDescendant1->Height = SubClassRadioGroup1->Height;
  MyListBoxDescendant1->OnDrawItem =
    MyListBoxDescendant1->OnDrawItemProc;

  bitmap0 = new Graphics::TBitmap;
  ImageList1->GetBitmap(0, bitmap0);
  MyListBoxDescendant1->Items->AddObject("Butterfly", bitmap0);

  SubClassRadioGroup1->Items->Add("SubClassWndProc");
  SubClassRadioGroup1->Items->Add("WndProc");
  SubClassRadioGroup1->ItemIndex = 2;
}
```

Delphi Examples:

```
{
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
}
```

```
type
  TMyListBoxDescendant = class(TlistBox)
    procedure SubClassWndProc(var Message: TMessage);
    procedure ToggleSubClass(On: Boolean);
    procedure OnDrawItemProc(
      Control: TWinControl;
      Index: Integer;
      Rect:TRect;
      State: TOwnerDrawState);
  end;
  TForm1 = class(TForm)
    SubClassRadioGroup1: TRadioGroup;
    Button1: TButton;
    ImageList1: TImageList;
    Button2: TButton;
    procedure SubClassRadioGroup1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  MyListBoxDescendant1: TMyListBoxDescendant;
  bitmap0: TBitmap;

implementation

{$R *.dfm}

const WM_STYLEMESSAGE = WM_USER + 2000;

procedure TForm1.Button1Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
end;

procedure TMyListBoxDescendant.SubClassWndProc(var Message: TMessage);
begin
  if (Message.Msg = WM_STYLEMESSAGE) then
    Style:= TListBoxStyle(Message.WParam)
  else
    WndProc(Message);
end;

procedure TMyListBoxDescendant.ToggleSubClass(On: Boolean);
begin
```

```
    if On then
      WindowProc := SubClassWndProc
    else
      WindowProc := WndProc;
  end;

  procedure TForm1.SubClassRadioGroup1Click(Sender: TObject);
  begin
    MyListBoxDescendant1.ToggleSubClass(
      SubClassRadioGroup1.ItemIndex = 0);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    MyListBoxDescendant1:= TMyListBoxDescendant.Create(self);
    MyListBoxDescendant1.Visible:= True;
    MyListBoxDescendant1.Parent:= Form1;
    MyListBoxDescendant1.Visible:= True;
    MyListBoxDescendant1.Left:=
      SubClassRadioGroup1.Left + SubClassRadioGroup1.Width + 30;;
    MyListBoxDescendant1.Top:= SubClassRadioGroup1.Top;
    MyListBoxDescendant1.Height:= SubClassRadioGroup1.Height;
    MyListBoxDescendant1.OnDrawItem:=
      MyListBoxDescendant1.OnDrawItemProc;

    bitmap0 := TBitmap.Create;
    ImageList1.GetBitmap(0, bitmap0);
    MyListBoxDescendant1.Items.AddObject('Butterfly', bitmap0);

    SubClassRadioGroup1.Items.Add('SubClassWndProc');
    SubClassRadioGroup1.Items.Add('WndProc');
    SubClassRadioGroup1.ItemIndex := 2;
  end;

  procedure TMyListBoxDescendant.OnDrawItemProc(
    Control: TWinControl;
    Index: Integer;
    Rect:TRect;
    State: TOwnerDrawState);
  var
    Bitmap: TBitmap;      { temporary variable for the item's bitmap }
    Offset: Integer;      { text offset width }
  begin
    { draw on control canvas, not on the form }
    with (Control as TListBox).Canvas do
    begin
      FillRect(Rect);       { clear the rectangle }
      Offset := 2;          { provide default offset }
      Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
      if Bitmap <> nil then
      begin
        BrushCopy(
          Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
          Bitmap,
          Bounds(0, 0, Bitmap.Width, Bitmap.Height),
          clRed);  {render bitmap}
        Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
      end;
      TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
  text }
    end;
  end;
```

See Also

DefWndProc (see page 6199)

MainWndProc (see page 6165)

WndProc (see page 5791)

WindowProc (see page 5872)

5.1.4.6 TCustomActionMenuBar Properties

The properties of the TCustomActionMenuBar class are listed here.

Protected Properties

	Name	Description
	AnimateDuration (see page 716)	This is AnimateDuration, a member of class TCustomActionMenuBar.
	DelayItem (see page 717)	This is DelayItem, a member of class TCustomActionMenuBar.
	Expandable (see page 717)	This is Expandable, a member of class TCustomActionMenuBar.
	Expanded (see page 718)	This is Expanded, a member of class TCustomActionMenuBar.
	MouseControl (see page 718)	This is MouseControl, a member of class TCustomActionMenuBar.
	ParentMenu (see page 719)	This is ParentMenu, a member of class TCustomActionMenuBar.
R	PopupStack (see page 719)	This is PopupStack, a member of class TCustomActionMenuBar.
R	Selected (see page 719)	This is Selected, a member of class TCustomActionMenuBar.

Public Properties

	Name	Description
	AnimationStyle (see page 716)	Specifies the type of animation the menu bar will display as it opens.
	AutoFocus (see page 717)	This is AutoFocus, a member of class TCustomActionMenuBar.
	ExpandDelay (see page 717)	Specifies, in milliseconds, the delay period that a menu action band uses before showing hidden or unused items.
	InMenuLoop (see page 718)	Indicates whether items in the menu can be selected.
	ParentControl (see page 718)	Refers to the parent object of the TCustomActionBarControl.
	RootMenu (see page 719)	Refers to the root menu of the ActionBand menu.
	UseSystemFont (see page 720)	Specifies whether the default system font for menu items is used.

5.1.4.6.1 TCustomActionMenuBar.AnimateDuration Property

Pascal

```
property AnimateDuration: Integer;
```

C++

```
__property int AnimateDuration;
```

Description

This is AnimateDuration, a member of class TCustomActionMenuBar.

5.1.4.6.2 TCustomActionMenuBar.AnimationStyle Property

Specifies the type of animation the menu bar will display as it opens.

Pascal

```
property AnimationStyle: TAnimationStyle;
```

C++

```
__property TAnimationStyle AnimationStyle;
```

Description

Use AnimationStyle to change the style of animation used by the items in the menu bar as they open. The options are:.

Value	Animation Description
asNone	No animation is used; the menu appears fully formed.
asDefault	The operating system default animation style is used.
asUnfold	The menu unfolds diagonally down from the upper left corner.
asSlide	The menu slides from the top down.
asFade	The menu fades in from invisible to visible.

5.1.4.6.3 TCustomActionMenuBar.AutoFocus Property

Pascal

```
property AutoFocus: Boolean;
```

C++

```
__property Boolean AutoFocus;
```

Description

This is AutoFocus, a member of class TCustomActionMenuBar.

5.1.4.6.4 TCustomActionMenuBar.DelayItem Property

Pascal

```
property DelayItem: TCustomActionControl;
```

C++

```
__property TCustomActionControl DelayItem;
```

Description

This is DelayItem, a member of class TCustomActionMenuBar.

5.1.4.6.5 TCustomActionMenuBar.Expandable Property

Pascal

```
property Expandable: Boolean;
```

C++

```
__property Boolean Expandable;
```

Description

This is Expandable, a member of class TCustomActionMenuBar.

5.1.4.6.6 TCustomActionMenuBar.ExpandDelay Property

Specifies, in milliseconds, the delay period that a menu action band uses before showing hidden or unused items.

Pascal

```
property ExpandDelay: Integer;
```

C++

```
__property int ExpandDelay;
```

Description

Use ExpandDelay to specify, in milliseconds, the delay period that a menu action band uses before showing hidden or unused items. The default is 4000 milliseconds.

5.1.4.6.7 TCustomActionMenuBar.Expanded Property

Pascal

```
property Expanded: Boolean;
```

C++

```
__property Boolean Expanded;
```

Description

This is Expanded, a member of class TCustomActionMenuBar.

5.1.4.6.8 TCustomActionMenuBar.InMenuLoop Property

Indicates whether items in the menu can be selected.

Pascal

```
property InMenuLoop: Boolean;
```

C++

```
__property Boolean InMenuLoop;
```

Description

Use InMenuLoop to determine if the menu bar has focus or not, which would allow items on the menu bar to be selected.

5.1.4.6.9 TCustomActionMenuBar.MouseControl Property

Pascal

```
property MouseControl: TControl;
```

C++

```
__property TControl * MouseControl;
```

Description

This is MouseControl, a member of class TCustomActionMenuBar.

5.1.4.6.10 TCustomActionMenuBar.ParentControl Property

Refers to the parent object of the TCustomActionBarControl.

Pascal

```
property ParentControl: TCustomActionControl;
```

C++

```
__property TCustomActionControl ParentControl;
```

Description

Use ParentControl to obtain a reference to the parent object which created the TCustomActionMenuBar (see page 685) object. This property is used mostly for popup menus, where the content of the menu depends upon the object in which the menu is appearing (the parent object). ParentControl is simply a reference to that object.

5.1.4.6.11 TCustomActionMenuBar.ParentMenu Property

Pascal

```
property ParentMenu: TCustomActionMenuBar;
```

C++

```
__property TCustomActionMenuBar ParentMenu;
```

Description

This is ParentMenu, a member of class TCustomActionMenuBar.

5.1.4.6.12 TCustomActionMenuBar.PopupStack Property

Pascal

```
property PopupStack: TMenuStack;
```

C++

```
__property TMenuStack PopupStack;
```

Description

This is PopupStack, a member of class TCustomActionMenuBar.

5.1.4.6.13 TCustomActionMenuBar.RootMenu Property

Refers to the root menu of the ActionBand menu.

Pascal

```
property RootMenu: TCustomActionMenuBar;
```

C++

```
__property TCustomActionMenuBar RootMenu;
```

Description

Application menus frequently contain submenus. It is sometimes important to know what the root menu is of any given menu; for example, selecting an item on a submenu may require that some change be made to the root menu object. Use RootMenu to determine the root menu of any TCustomActionMenuBar (see page 685) object.

5.1.4.6.14 TCustomActionMenuBar.Selected Property

Pascal

```
property Selected: TActionClientItem;
```

C++

```
__property TActionClientItem Selected;
```

Description

This is Selected, a member of class TCustomActionMenuBar.

5.1.4.6.15 TCustomActionMenuBar.UseSystemFont Property

Specifies whether the default system font for menu items is used.

Pascal

```
property UseSystemFont: Boolean;
```

C++

```
__property Boolean UseSystemFont;
```

Description

Use UseSystemFont to specify whether the action band uses the default system menu font when it draws the captions for its items. By default, UseSystemFont is set to true. When the Font property is changed from the default values, UseSystemFont is automatically set to false. Setting UseSystemFont to true causes the font specified in the MenuFont property of the TScreen object to be used instead of the font specified in the action band's Font property.

See Also

TFont (see page 14497)

TextOut (see page 14475)

TextRect (see page 14480)

Assign

5.1.4.7 TCustomActionMenuBar Events

The events of the TCustomActionMenuBar class are listed here.

Public Events

	Name	Description
	OnEnterMenuLoop (see page 720)	Occurs when an ActionBand menu loop has begun.
	OnExitMenuLoop (see page 721)	Occurs when an ActionBand menu loop has ended and the menu is closed.
	OnGetPopupClass (see page 721)	Occurs prior to an ActionBand popupmenu appearing on screen.
	OnPopup (see page 721)	Occurs just before the pop-up menu appears.

5.1.4.7.1 TCustomActionMenuBar.OnEnterMenuLoop Event

Occurs when an ActionBand menu loop has begun.

Pascal

```
property OnEnterMenuLoop: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnterMenuLoop;
```

Description

Write an OnEnterMenuLoop event handler to modify the menu as necessary prior to the menu appearing on screen.

5.1.4.7.2 TCustomActionMenuBar.OnExitMenuLoop Event

Occurs when an ActionBand menu loop has ended and the menu is closed.

Pascal

```
property OnExitMenuLoop: TExitMenuEvent;
```

C++

```
__property TExitMenuEvent OnExitMenuLoop;
```

Description

Write an OnExitMenuLoop event handler to do any additional processing once the menu has closed.

5.1.4.7.3 TCustomActionMenuBar.OnGetPopupClass Event

Occurs prior to an ActionBand popupmenu appearing on screen.

Pascal

```
property OnGetPopupClass: TGetPopupClassEvent;
```

C++

```
__property TGetPopupClassEvent OnGetPopupClass;
```

Description

Write an OnGetPopupClass event handler to control which popup menu class is used to display an ActionBand menu.

5.1.4.7.4 TCustomActionMenuBar.OnPopup Event

Occurs just before the pop-up menu appears.

Pascal

```
property OnPopup: TMenuPopupEvent;
```

C++

```
__property TMenuPopupEvent OnPopup;
```

Description

Write an OnPopup event handler to take specific action just before the pop-up menu appears.

5.1.5 TCustomActionPopupMenu Class

TCustomActionPopupMenu is the ancestor for popup action bands that render actions as popup menu item UI elements.

Class Hierarchy

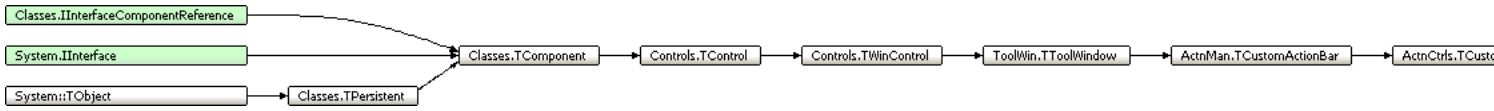

Pascal

```
TCustomActionPopupMenu = class(TCustomActionMenuBar);
```

C++

```
class TCustomActionPopupMenu : public TCustomActionMenuBar;
```

File

ActnMenus

Description

TCustomActionPopupMenu is the ancestor for all popup action bands that render actions as popup menu item UI elements, including the standard form, TStandardMenuPopup. If you would like to replace TStandardMenuPopup with a style of your own, subclass TCustomActionPopupMenu to provide your own behaviors. To make the change for your application, set the value of MenuPopupClass (see page 847) (a global variable) to your own subclass of TCustomActionPopupMenu instead of TStandardMenuPopup.

See Also

TStandardMenuPopup (see page 20478)

5.1.5.1 TCustomActionPopupMenu Members

The following tables list the members exposed by TCustomActionPopupMenu.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FCachedMenu (see page 699)	This is FCachedMenu, a member of class TCustomActionMenuBar.
	FChildMenu (see page 699)	This is FChildMenu, a member of class TCustomActionMenuBar.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefaultColorMap (see page 460)	This is FDefaultColorMap, a member of class TCustomActionBar.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExpandTimer (see page 700)	This is FExpandTimer, a member of class TCustomActionMenuBar.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FPopupTimer (see page 700)	This is FPopupTimer, a member of class TCustomActionMenuBar.
	FSavedWrapState (see page 460)	This is FSavedWrapState, a member of class TCustomActionBar.
	FSelectedItem (see page 700)	This is FSelectedItem, a member of class TCustomActionMenuBar.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AddEdges (see page 737)	This is AddEdges, a member of class TCustomActionPopupMenu.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 462)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	Animate (see page 701)	This is Animate, a member of class TCustomActionMenuBar.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
V	AutoSizingChanged (see page 462)	This is AutoSizingChanged, a member of class TCustomActionBar.

	BackgroundSize (see page 462)	This is BackgroundSize, a member of class TCustomActionBar.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcInsertPosition (see page 462)	This is CalcInsertPosition, a member of class TCustomActionBar.
V	CanAutoSize (see page 737)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CleanupStack (see page 701)	This is CleanupStack, a member of class TCustomActionMenuBar.
V	Clear (see page 463)	This is Clear, a member of class TCustomActionBar.
	ClearSubMenus (see page 702)	This is ClearSubMenus, a member of class TCustomActionMenuBar.
	Click (see page 5743)	Respond to user click.
	CMBorderChanged (see page 23285)	This is CMBorderChanged, a member of class TToolWindow.
	CMCtl3DChanged (see page 23286)	This is CMCtl3DChanged, a member of class TToolWindow.
	CMMouseLeave (see page 738)	This is CMMouseLeave, a member of class TCustomActionPopupMenu.
	CMVisibleChanged (see page 738)	This is CMVisibleChanged, a member of class TCustomActionPopupMenu.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
V	CreateControl (see page 738)	This is CreateControl, a member of class TCustomActionPopupMenu.
V	CreateControls (see page 738)	This is CreateControls, a member of class TCustomActionPopupMenu.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 739)	Initializes a window-creation parameter data structure.
V	CreatePopup (see page 702)	This is CreatePopup, a member of class TCustomActionMenuBar.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 463)	Returns placement order of custom-aligned child control.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 464)	Returns aligned position of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
V	DesignWndProc (see page 739)	This is DesignWndProc, a member of class TCustomActionPopupMenu.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
V	DisplayShadow (see page 739)	This is DisplayShadow, a member of class TCustomActionPopupMenu.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.

	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
V	DoControlCreated (see page 465)	This is DoControlCreated, a member of class TCustomActionBar.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
V	DoDropActions (see page 465)	This is DoDropActions, a member of class TCustomActionBar.
V	DoDropCategory (see page 465)	This is DoDropCategory, a member of class TCustomActionBar.
V	DoDropItem (see page 466)	This is DoDropItem, a member of class TCustomActionBar.
V	DoEndDock (see page 146)	OnEndDock event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetControlClass (see page 466)	This is DoGetControlClass, a member of class TCustomActionBar.
	DoGetPopupClass (see page 703)	This is DoGetPopupClass, a member of class TCustomActionMenuBar.
V	DoItemClicked (see page 703)	This is DoItemClicked, a member of class TCustomActionMenuBar.
V	DoItemKeyed (see page 703)	This is DoItemKeyed, a member of class TCustomActionMenuBar.
	DoItemSelected (see page 703)	This is DoItemSelected, a member of class TCustomActionMenuBar.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMenuDelay (see page 703)	This is DoMenuDelay, a member of class TCustomActionMenuBar.
	DoMouseIdle (see page 704)	This is DoMouseIdle, a member of class TCustomActionMenuBar.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
V	DoneMenuLoop (see page 704)	This is DoneMenuLoop, a member of class TCustomActionMenuBar.
V	DoPopup (see page 704)	This is DoPopup, a member of class TCustomActionMenuBar.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
V	DoStartDock (see page 147)	OnStartDock event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 466)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawBackground (see page 740)	This is DrawBackground, a member of class TCustomActionPopupMenu.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	ExecAction (see page 740)	This is ExecAction, a member of class TCustomActionPopupMenu.
	ExpandClick (see page 740)	This is ExpandClick, a member of class TCustomActionPopupMenu.
	ExpandTimer (see page 705)	This is ExpandTimer, a member of class TCustomActionMenuBar.
	FindAccelItem (see page 467)	This is FindAccelItem, a member of class TCustomActionBar.
	FindItemAt (see page 468)	This is FindItemAt, a member of class TCustomActionBar.
	FindItemCaption (see page 468)	This is FindItemCaption, a member of class TCustomActionBar.
V	FindLast (see page 468)	This is FindLast, a member of class TCustomActionBar.
	FindLeastUsedItem (see page 469)	This is FindLeastUsedItem, a member of class TCustomActionBar.
	FindNearestControl (see page 469)	This is FindNearestControl, a member of class TCustomActionBar.
V	FindNext (see page 470)	This is FindNext, a member of class TCustomActionBar.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FindNextVisibleItem (see page 470)	This is FindNextVisibleItem, a member of class TCustomActionBar.
V	FindPrevious (see page 470)	This is FindPrevious, a member of class TCustomActionBar.

	FindPreviousVisibleItem (see page 470)	This is FindPreviousVisibleItem, a member of class TCustomActionBar.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
V	GetActionControl (see page 471)	This is GetActionControl, a member of class TCustomActionBar.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
V	GetBannerWidth (see page 471)	This is GetBannerWidth, a member of class TCustomActionBar.
V	GetBarHeight (see page 471)	This is GetBarHeight, a member of class TCustomActionBar.
V	GetBarWidth (see page 471)	This is GetBarWidth, a member of class TCustomActionBar.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 741)	Returns the value of the ClientRect property.
V	GetControlClass (see page 471)	This is GetControlClass, a member of class TCustomActionBar.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDefaultColorMapClass (see page 472)	This is GetDefaultColorMapClass, a member of class TCustomActionBar.
	GetDefaultSounds (see page 705)	This is GetDefaultSounds, a member of class TCustomActionMenuBar.
V	GetDesignMode (see page 472)	This is GetDesignMode, a member of class TCustomActionBar.
V	GetDesignMode (see page 705)	This is GetDesignMode, a member of class TCustomActionMenuBar.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetExpandBtnClass (see page 741)	This is GetExpandBtnClass, a member of class TCustomActionPopupMenu.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 147)	Returns the class used for the control when it is floating.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetMouseHoverItem (see page 705)	This is GetMouseHoverItem, a member of class TCustomActionMenuBar.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
V	GetPopupClass (see page 706)	This is GetPopupClass, a member of class TCustomActionMenuBar.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetPopupMenuClass (see page 472)	This is GetPopupMenuClass, a member of class TCustomActionBar.
	GetSelected (see page 706)	This is GetSelected, a member of class TCustomActionMenuBar.
V	GetSiteInfo (see page 147)	Generates an OnGetSiteInfo event
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasItems (see page 472)	This is HasItems, a member of class TCustomActionBar.
V	HideShadow (see page 741)	This is HideShadow, a member of class TCustomActionPopupMenu.
	Idle (see page 706)	This is Idle, a member of class TCustomActionMenuBar.
V	InitMenuLoop (see page 706)	This is InitMenuLoop, a member of class TCustomActionMenuBar.

	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsDesignMsg (see page 707)	This is IsDesignMsg, a member of class TCustomActionMenuBar.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
V	ItemCount (see page 473)	This is ItemCount, a member of class TCustomActionBar.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 148)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 473)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 707)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
V	MouseActivate (see page 707)	This is MouseActivate, a member of class TCustomActionMenuBar.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	NCPaint (see page 473)	This is NCPaint, a member of class TCustomActionBar.
V	NCPaint (see page 23286)	This is NCPaint, a member of class TToolWindow.
V	NewPopup (see page 741)	This is NewPopup, a member of class TCustomActionPopupMenu.
V	Notification (see page 742)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 474)	This is Paint, a member of class TCustomActionBar.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 474)	Renders the image of a windowed control.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	PositionPopup (see page 742)	This is PositionPopup, a member of class TCustomActionPopupMenu.
V	ProcessMenuLoop (see page 708)	This is ProcessMenuLoop, a member of class TCustomActionMenuBar.
	ProcessMessage (see page 708)	This is ProcessMessage, a member of class TCustomActionMenuBar.
	ProcessMessages (see page 709)	This is ProcessMessages, a member of class TCustomActionMenuBar.
V	ProcessMouseMsg (see page 709)	This is ProcessMouseMsg, a member of class TCustomActionMenuBar.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
V	Reset (see page 475)	This is Reset, a member of class TCustomActionBar.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.

	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Select (see page 709)	This is Select, a member of class TCustomActionMenuBar.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectItem (see page 710)	This is SelectItem, a member of class TCustomActionMenuBar.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
V	SetActionClient (see page 475)	This is SetActionClient, a member of class TCustomActionBar.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetAutoSizing (see page 475)	This is SetAutoSizing, a member of class TCustomActionBar.
V	SetBiDiMode (see page 475)	Sets the control's bi-directional mode.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
V	SetColorMap (see page 743)	This is SetColorMap, a member of class TCustomActionPopupMenu.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDesignMode (see page 476)	This is SetDesignMode, a member of class TCustomActionBar.
V	SetDesignMode (see page 710)	Specifies whether the whether the action band is being customized or being used in the IDE.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetOrientation (see page 743)	This is SetOrientation, a member of class TCustomActionPopupMenu.
V	SetParent (see page 710)	Sets the parent of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetPersistentHotKeys (see page 477)	This is SetPersistentHotKeys, a member of class TCustomActionBar.
V	SetPersistentHotKeys (see page 711)	This is SetPersistentHotKeys, a member of class TCustomActionMenuBar.
V	SetSpacing (see page 477)	This is SetSpacing, a member of class TCustomActionBar.
	SetupDefaultColorMap (see page 477)	This is SetupDefaultColorMap, a member of class TCustomActionBar.
	SetupExpandBtn (see page 743)	This is SetupExpandBtn, a member of class TCustomActionPopupMenu.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.

	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	VisibleChanging (see page 743)	Provides the interface for a method that responds to changes in the control's visibility.
	WMKeyDown (see page 744)	This is WMKeyDown, a member of class TCustomActionPopupMenu.
	WMNCCalcSize (see page 23286)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCPaint (see page 23286)	This is WMNCPaint, a member of class TToolWindow.
	WMPrint (see page 744)	This is WMPrint, a member of class TCustomActionPopupMenu.
	WMPrintClient (see page 744)	This is WMPrintClient, a member of class TCustomActionPopupMenu.
V	WndProc (see page 711)	Provides specific message responses for the control.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AnimateDuration (see page 716)	This is AnimateDuration, a member of class TCustomActionMenuBar.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DelayItem (see page 717)	This is DelayItem, a member of class TCustomActionMenuBar.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Expandable (see page 717)	This is Expandable, a member of class TCustomActionMenuBar.
	Expanded (see page 718)	This is Expanded, a member of class TCustomActionMenuBar.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
R	HRowCount (see page 488)	Indicates the number of columns of items that the action band displays.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).

	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	Items (see page 488)	Lists the action clients that correspond to the child controls of this action band.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	MouseControl (see page 718)	This is MouseControl, a member of class TCustomActionMenuBar.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentMenu (see page 719)	This is ParentMenu, a member of class TCustomActionMenuBar.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
R	PopupStack (see page 719)	This is PopupStack, a member of class TCustomActionMenuBar.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
R	Selected (see page 719)	This is Selected, a member of class TCustomActionMenuBar.
	Text (see page 5842)	Contains a text string associated with the control.
R	VRowCount (see page 490)	Indicates the number of rows of items that the action band displays.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
R	Widest (see page 491)	This is Widest, a member of class TCustomActionBar.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnControlCreated (see page 491)	Occurs when a control is created on the action band.
	OnEnterMenuLoop (see page 720)	Occurs when an ActionBand menu loop has begun.
	OnExitMenuLoop (see page 721)	Occurs when an ActionBand menu loop has ended and the menu is closed.
	OnGetControlClass (see page 492)	Occurs when an item is to be placed on the action band.
	OnGetPopupClass (see page 721)	Occurs prior to an ActionBand popupmenu appearing on screen.
	OnPaint (see page 492)	Occurs when the action band is painted.
	OnPopup (see page 721)	Occurs just before the pop-up menu appears.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
V	CalcDockedWidth (see page 146)	This is CalcDockedWidth, a member of class TCustomActionDockBar.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	CloseMenu (see page 737)	Closes the menu currently being displayed.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 735)	Creates and initializes a TCustomActionPopupMenu (see page 721) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 736)	Destroys an instance of TCustomActionMenuBar (see page 685).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Dock (see page 146)	Used internally to dock the action band.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
V	DragDrop (see page 466)	Generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	Expand (see page 740)	Performs the same kinds of functions as the inherited method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
V	FindFirst (see page 467)	Finds the first TActionClientItem (see page 374) in the Items (see page 488) list.
	FindFirstVisibleItem (see page 468)	Finds the first visible TActionClientItem (see page 374) in the Items (see page 488) list.
	FindLastVisibleItem (see page 469)	Finds the last visibleTActionClientItem in the Items (see page 488) list.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	Popup (see page 742)	Causes the popup menu to appear.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
V	RecreateControls (see page 475)	Destroys and recreates all of the action band's action client controls.
V	RecreateControls (see page 709)	Destroys and recreates all of the action band's action client controls.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.

	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 476)	Sets the bounds based on whether the popup is expanded or not
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Style (see page 477)	Identifies the style object that gives the application its default look and feel.
V	Style (see page 711)	Retrieves the style used by the ActionBand menu that started the menu loop so the submenus have a consistent appearance.
V	TrackMenu (see page 711)	Executes the action associated with the menu item.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	ActionClient (see page 479)	Specifies the action client associated with this action band.
R	ActionControls (see page 479)	Lists the controls on the action band.
	ActionManager (see page 480)	Specifies the action manager component that is associated with the action band.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AllowHiding (see page 480)	Determines whether the action band can be hidden.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AnimationStyle (see page 716)	Specifies the type of animation the menu bar will display as it opens.
	AutoFocus (see page 717)	This is AutoFocus, a member of class TCustomActionMenuBar.
	AutoSizing (see page 480)	Specifies whether the action band adjusts its size automatically to fit around its child controls.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 481)	Represents the drawing surface of the action band.
	Caption (see page 481)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 483)	Specifies the background color of the control.
	ColorMap (see page 484)	Indicates the colormap for individual action band components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ContextBar (see page 485)	Indicates whether items shown in this action band represent the items in a popup menu.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designable (see page 485)	Specifies whether the action band can be customized (used internally by the IDE).
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignMode (see page 486)	Specifies whether the action band is being customized or being used in the IDE.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EdgeBorders (see page 23287)	Determines which sides of the control have visible borders.
	EdgeInner (see page 23287)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 23288)	Determines the border style for the control's outer edge.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
	ExpandBtn (see page 744)	Refers to the popup menu's expand button.
	ExpandDelay (see page 717)	Specifies, in milliseconds, the delay period that a menu action band uses before showing hidden or unused items.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Font (see page 486)	Controls the attributes of text written on or in the control.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HorzMargin (see page 487)	Specifies the width of the horizontal margins in pixels.
	HorzSeparator (see page 487)	Indicates whether a horizontal separator bar appears between item lines.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	InMenuLoop (see page 718)	Indicates whether items in the menu can be selected.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	Orientation (see page 488)	Specifies how the action band lays out its child items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentControl (see page 718)	Refers to the parent object of the TCustomActionBarControl.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PersistentHotKeys (see page 489)	Specifies whether keyboard accelerator shortcuts are always displayed.
	RootMenu (see page 719)	Refers to the root menu of the ActionBand menu.

	Shadow (see page 745)	Refers to the popup menu's shadow.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	Spacing (see page 489)	Determines where the image and text appear on an action band's rendered control that represents the action client.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	UseSystemFont (see page 720)	Specifies whether the default system font for menu items is used.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	VertMargin (see page 490)	Specifies the width of the vertical margins in pixels.
	VertSeparator (see page 490)	Indicates whether a vertical separator bar should appear between item lines.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

5.1.5.2 TCustomActionPopupMenu.Create Constructor

Creates and initializes a TCustomActionPopupMenu (see page 721) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomActionPopupMenu(TComponent * AOwner);
```

Description

Create generates a TCustomActionPopupMenu (see page 721) and initializes its properties.

```
__fastcall virtual TCustomActionMenuBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall TCustomActionMenuBar (HWND ParentWindow) :
Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

5.1.5.3 TCustomActionPopupMenu.Destroy Destructor

Destroys an instance of TCustomActionMenuBar (see page 685).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomActionPopupMenu();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TCustomActionMenuBar (see page 685) reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

Destroy (see page 5732)

5.1.5.4 TCustomActionPopupMenu Methods

The methods of the TCustomActionPopupMenu class are listed here.

Protected Methods

	Name	Description
V	AddEdges (see page 737)	This is AddEdges, a member of class TCustomActionPopupMenu.
V	CanAutoSize (see page 737)	Indicates how the control can be resized.
	CMMouseLeave (see page 738)	This is CMMouseLeave, a member of class TCustomActionPopupMenu.
	CMVisibleChanged (see page 738)	This is CMVisibleChanged, a member of class TCustomActionPopupMenu.
V	CreateControl (see page 738)	This is CreateControl, a member of class TCustomActionPopupMenu.
V	CreateControls (see page 738)	This is CreateControls, a member of class TCustomActionPopupMenu.
V	CreateParams (see page 739)	Initializes a window-creation parameter data structure.
V	DesignWndProc (see page 739)	This is DesignWndProc, a member of class TCustomActionPopupMenu.
V	DisplayShadow (see page 739)	This is DisplayShadow, a member of class TCustomActionPopupMenu.
V	DrawBackground (see page 740)	This is DrawBackground, a member of class TCustomActionPopupMenu.
V	ExecAction (see page 740)	This is ExecAction, a member of class TCustomActionPopupMenu.
	ExpandClick (see page 740)	This is ExpandClick, a member of class TCustomActionPopupMenu.
V	GetClientRect (see page 741)	Returns the value of the ClientRect property.
V	GetExpandBtnClass (see page 741)	This is GetExpandBtnClass, a member of class TCustomActionPopupMenu.
V	HideShadow (see page 741)	This is HideShadow, a member of class TCustomActionPopupMenu.
V	NewPopup (see page 741)	This is NewPopup, a member of class TCustomActionPopupMenu.
V	Notification (see page 742)	Responds to notifications that components are being created or destroyed.
V	PositionPopup (see page 742)	This is PositionPopup, a member of class TCustomActionPopupMenu.
V	SetColorMap (see page 743)	This is SetColorMap, a member of class TCustomActionPopupMenu.
V	SetOrientation (see page 743)	This is SetOrientation, a member of class TCustomActionPopupMenu.
	SetupExpandBtn (see page 743)	This is SetupExpandBtn, a member of class TCustomActionPopupMenu.
V	VisibleChanging (see page 743)	Provides the interface for a method that responds to changes in the control's visibility.
	WMKeyDown (see page 744)	This is WMKeyDown, a member of class TCustomActionPopupMenu.
	WMPrint (see page 744)	This is WMPrint, a member of class TCustomActionPopupMenu.
	WMPrintClient (see page 744)	This is WMPrintClient, a member of class TCustomActionPopupMenu.

Public Methods

	Name	Description
	CloseMenu (see page 737)	Closes the menu currently being displayed.
	Expand (see page 740)	Performs the same kinds of functions as the inherited method.
	Popup (see page 742)	Causes the popup menu to appear.

5.1.5.4.1 TCustomActionPopupMenu.AddEdges Method

Pascal

```
procedure AddEdges(AnItem: TCustomMenuItem); virtual;
```

C++

```
virtual __fastcall AddEdges(TCustomMenuItem AnItem);
```

Description

This is AddEdges, a member of class TCustomActionPopupMenu.

5.1.5.4.2 TCustomActionPopupMenu.CanAutoSize Method

Indicates how the control can be resized.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize property is true and an attempt is made to resize the control. It allows controls to implement the AutoSize property before the resize sequence that begins with the OnCanResize and OnConstrainedResize events and ends with the OnResize event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the control. CanAutoSize adjusts these values so that the control's size adjusts to its contents. If the adjustment results in values of NewWidth and NewHeight that differ from the control's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width and Height, CanAutoSize returns false, and the resize sequence is aborted.

As implemented in TWinControl, CanAutoSize returns true, adjusting NewWidth and NewHeight so that all the control's children fit.

See Also

Constraints (see page 5812)

ConstrainedResize (see page 6133)

AdjustSize (see page 6129)

5.1.5.4.3 TCustomActionPopupMenu.CloseMenu Method

Closes the menu currently being displayed.

Pascal

```
procedure CloseMenu; override;
```

C++

```
virtual __fastcall CloseMenu();
```

Description

CloseMenu is called to close the menu currently being displayed. The only difference between this routine and the inherited one is that the Visible property is set to false.

See Also

CloseMenu (see page 702)

Visible (see page 5847)

5.1.5.4.4 TCustomActionPopupMenu.CMMouseLeave Method

Pascal

```
procedure CMMouseLeave(var Message: TMessage); message CM_MOUSELEAVE;
```

C++

```
__fastcall CMMouseLeave(TMessage Message);
```

Description

This is CMMouseLeave, a member of class TCustomActionPopupMenu.

5.1.5.4.5 TCustomActionPopupMenu.CMVisibleChanged Method

Pascal

```
procedure CMVisibleChanged(var Message: TMessage); message CM_VISIBLECHANGED;
```

C++

```
__fastcall CMVisibleChanged(TMessage Message);
```

Description

This is CMVisibleChanged, a member of class TCustomActionPopupMenu.

5.1.5.4.6 TCustomActionPopupMenu.CreateControl Method

Pascal

```
function CreateControl(AnItem: TActionClientItem): TCustomActionControl; override;
```

C++

```
virtual __fastcall TCustomActionControl CreateControl(TActionClientItem AnItem);
```

Description

This is CreateControl, a member of class TCustomActionPopupMenu.

5.1.5.4.7 TCustomActionPopupMenu.CreateControls Method

Pascal

```
procedure CreateControls; override;
```

C++

```
virtual __fastcall CreateControls();
```

Description

This is CreateControls, a member of class TCustomActionPopupMenu.

5.1.5.4.8 TCustomActionPopupMenu.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

5.1.5.4.9 TCustomActionPopupMenu.DesignWndProc Method

Pascal

```
function DesignWndProc(var Message: TMessage): Boolean; override;
```

C++

```
virtual __fastcall Boolean DesignWndProc(TMessage Message);
```

Description

This is DesignWndProc, a member of class TCustomActionPopupMenu.

5.1.5.4.10 TCustomActionPopupMenu.DisplayShadow Method

Pascal

```
procedure DisplayShadow; virtual;
```

C++

```
virtual __fastcall DisplayShadow();
```

Description

This is DisplayShadow, a member of class TCustomActionPopupMenu.

5.1.5.4.11 TCustomActionPopupMenu.DrawBackground Method

Pascal

```
procedure DrawBackground; override;
```

C++

```
virtual __fastcall DrawBackground();
```

Description

This is DrawBackground, a member of class TCustomActionPopupMenu.

5.1.5.4.12 TCustomActionPopupMenu.ExecAction Method

Pascal

```
procedure ExecAction(Action: TContainedAction); override;
```

C++

```
virtual __fastcall ExecAction(TContainedAction Action);
```

Description

This is ExecAction, a member of class TCustomActionPopupMenu.

5.1.5.4.13 TCustomActionPopupMenu.Expand Method

Performs the same kinds of functions as the inherited method.

Pascal

```
procedure Expand(Full: Boolean); override;
```

C++

```
virtual __fastcall Expand(Boolean Full);
```

Description

This implementation of Expand serves the same purpose as the inherited method. It performs actions specific to popup menus. For more information, see the documentation for the inherited method.

See Also

Expand (see page 704)

5.1.5.4.14 TCustomActionPopupMenu.ExpandClick Method

Pascal

```
procedure ExpandClick(Sender: TObject);
```

C++

```
__fastcall ExpandClick(TObject * Sender);
```

Description

This is ExpandClick, a member of class TCustomActionPopupMenu.

5.1.5.4.15 TCustomActionPopupMenu.GetClientRect Method

Returns the value of the ClientRect property.

Pascal

```
function GetClientRect: TRect; override;
```

C++

```
virtual __fastcall TRect GetClientRect();
```

Description

GetClientRect is the protected implementation of the ClientRect property. GetClientRect returns a rectangle with the rectangle's Top and Left fields set to zero, and its Bottom and Right fields set to the control's ClientHeight and ClientWidth, respectively.

See Also

ClientHeight (see page 5808)

ClientWidth (see page 5811)

GetClientOrigin (see page 6153)

5.1.5.4.16 TCustomActionPopupMenu.GetExpandBtnClass Method

Pascal

```
function GetExpandBtnClass: TCustomMenuExpandBtnClass; virtual;
```

C++

```
virtual __fastcall TCustomMenuExpandBtnClass GetExpandBtnClass();
```

Description

This is GetExpandBtnClass, a member of class TCustomActionPopupMenu.

5.1.5.4.17 TCustomActionPopupMenu.HideShadow Method

Pascal

```
procedure HideShadow; virtual;
```

C++

```
virtual __fastcall HideShadow();
```

Description

This is HideShadow, a member of class TCustomActionPopupMenu.

5.1.5.4.18 TCustomActionPopupMenu.NewPopup Method

Pascal

```
function NewPopup: TCustomActionPopupMenu; override;
```

C++

```
virtual __fastcall TCustomActionPopupMenu NewPopup();
```

Description

This is NewPopup, a member of class TCustomActionPopupMenu.

5.1.5.4.19 TCustomActionPopupMenu.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

5.1.5.4.20 TCustomActionPopupMenu.Popup Method

Causes the popup menu to appear.

Pascal

```
procedure Popup(X: Integer; Y: Integer);
```

C++

```
__fastcall Popup(int X, int Y);
```

Description

Call Popup to display the popup menu at the specified coordinates (in pixels) on screen. This method is meant to behave the same as the procedure of the same name in TPopupMenu.

See Also

Popup (see page 16174)

5.1.5.4.21 TCustomActionPopupMenu.PositionPopup Method

Pascal

```
procedure PositionPopup(AnOwner: TCustomActionBar; ParentItem: TCustomActionControl); virtual;
```

C++

```
virtual __fastcall PositionPopup(TCustomActionBar AnOwner, TCustomActionControl ParentItem);
```

Description

This is PositionPopup, a member of class TCustomActionPopupMenu.

5.1.5.4.22 TCustomActionPopupMenu.SetColorMap Method

Pascal

```
procedure SetColorMap(const Value: TCustomActionBarColorMap); override;
```

C++

```
virtual __fastcall SetColorMap(const TCustomActionBarColorMap Value);
```

Description

This is SetColorMap, a member of class TCustomActionPopupMenu.

5.1.5.4.23 TCustomActionPopupMenu.SetOrientation Method

Pascal

```
procedure SetOrientation(const Value: TBarOrientation); override;
```

C++

```
virtual __fastcall SetOrientation(const TBarOrientation Value);
```

Description

This is SetOrientation, a member of class TCustomActionPopupMenu.

5.1.5.4.24 TCustomActionPopupMenu.SetupExpandBtn Method

Pascal

```
procedure SetupExpandBtn;
```

C++

```
__fastcall SetupExpandBtn();
```

Description

This is SetupExpandBtn, a member of class TCustomActionPopupMenu.

5.1.5.4.25 TCustomActionPopupMenu.VisibleChanging Method

Provides the interface for a method that responds to changes in the control's visibility.

Pascal

```
procedure VisibleChanging; override;
```

C++

```
virtual __fastcall VisibleChanging();
```

Description

VisibleChanging is called before the value of a control's Visible property changes. As implemented in TControl, it does nothing. Override VisibleChanging to respond to changes in the control's visibility. For example, if a control should not change its visibility, override VisibleChanging to raise an exception when an attempt is made to alter the control's visibility.

See Also

Visible (see page 5847)

5.1.5.4.26 TCustomActionPopupMenu.WMKeyDown Method

Pascal

```
procedure WMKeyDown(var Message: TWMKeyDown); message WM_KEYDOWN;
```

C++

```
__fastcall WMKeyDown(TWMKeyDown Message);
```

Description

This is WMKeyDown, a member of class TCustomActionPopupMenu.

5.1.5.4.27 TCustomActionPopupMenu.WMPrint Method

Pascal

```
procedure WMPrint(var Message: TMessage); message WM_PRINT;
```

C++

```
__fastcall WMPrint(TMessage Message);
```

Description

This is WMPrint, a member of class TCustomActionPopupMenu.

5.1.5.4.28 TCustomActionPopupMenu.WMPrintClient Method

Pascal

```
procedure WMPrintClient(var Message: TWMPrintClient); message WM_PRINTCLIENT;
```

C++

```
__fastcall WMPrintClient(TWMPrintClient Message);
```

Description

This is WMPrintClient, a member of class TCustomActionPopupMenu.

5.1.5.5 TCustomActionPopupMenu Properties

The properties of the TCustomActionPopupMenu class are listed here.

Public Properties

	Name	Description
	ExpandBtn (see page 744)	Refers to the popup menu's expand button.
	Shadow (see page 745)	Refers to the popup menu's shadow.

5.1.5.5.1 TCustomActionPopupMenu.ExpandBtn Property

Refers to the popup menu's expand button.

Pascal

```
property ExpandBtn: TCustomMenuExpandBtn;
```

C++

```
__property TCustomMenuExpandBtn ExpandBtn;
```

Description

Use ExpandBtn to access the popup menu action band's expand button object. An action band may have hidden items. For example, items which are used infrequently may be hidden to make the menus appear less cluttered. Still, the user may wish to use hidden items from time to time. They become visible again when the user selects the expand bar.

5.1.5.5.2 TCustomActionPopupMenu.Shadow Property

Refers to the popup menu's shadow.

Pascal

```
property Shadow: Boolean;
```

C++

```
__property Boolean Shadow;
```

Description

Use Shadow to control whether or not the menu displays with or without a shadow.

5.1.6 TCustomAddRemoveItem Class

TCustomAddRemoveItem is the base class for the ActionBand menus for customizing action bands.

Class Hierarchy

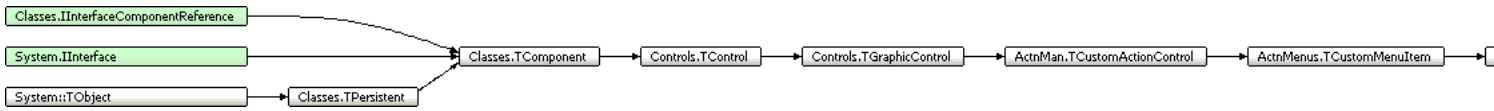

Pascal

```
TCustomAddRemoveItem = class(TCustomMenuItem);
```

C++

```
class TCustomAddRemoveItem : public TCustomMenuItem;
```

File

ActnMenus

Description

TCustomAddRemoveItem is a base class for TStandardAddRemoveItem, the standard ActionBand menu for customizing action bands at runtime.

Do not create instances of TCustomAddRemoveItem. If you would like to replace the standard customization menu, use the class TCustomAddRemoveItem as a parent class for your replacement class. To make the change for your application, set the value of MenuAddRemoveItemClass (see page 846) (a global variable) to your own subclass of TCustomAddRemoveItem instead of TStandardAddRemoveItem.

See Also

TStandardAddRemoveItem (see page 20398)

5.1.6.1 TCustomAddRemoveItem Members

The following tables list the members exposed by TCustomAddRemoveItem.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FNoPrefix (see page 826)	This is FNoPrefix, a member of class TCustomMenuItem.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.

	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	BeginAutoDrag (see page 517)	Begin Automatic Drag.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CalcLayout (see page 518)	This is CalcLayout, a member of class TCustomActionControl.
V	CalcLayout (see page 827)	This is CalcLayout, a member of class TCustomMenuItem.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
	CMDesignHitTest (see page 519)	This is CMDesignHitTest, a member of class TCustomActionControl.
	CMFontChanged (see page 519)	This is CMFontChanged, a member of class TCustomActionControl.
	CMItemDropPoint (see page 519)	This is CMItemDropPoint, a member of class TCustomActionControl.
	CMItemSelected (see page 519)	This is CMItemSelected, a member of class TCustomActionControl.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DesignWndProc (see page 519)	This is DesignWndProc, a member of class TCustomActionControl.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
V	DoDragDrop (see page 520)	This is DoDragDrop, a member of class TCustomActionControl.
V	DoDragDrop (see page 828)	This is DoDragDrop, a member of class TCustomMenuItem.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
V	DoStartDrag (see page 520)	OnStartDrag event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 521)	OnDragOver event dispatcher.
V	DragOver (see page 828)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawBackground (see page 521)	Draws a background image on the control.
V	DrawDesignFocus (see page 521)	This is DrawDesignFocus, a member of class TCustomActionControl.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawDragDropPoint (see page 522)	This is DrawDragDropPoint, a member of class TCustomActionControl.
V	DrawEdge (see page 829)	This is DrawEdge, a member of class TCustomMenuItem.
V	DrawGlyph (see page 522)	Draws a symbol on the control.

	DrawLargeGlyph (see page 522)	This is DrawLargeGlyph, a member of class TCustomActionControl.
	DrawSeparator (see page 522)	This is DrawSeparator, a member of class TCustomActionControl.
	DrawShadowedText (see page 523)	This is DrawShadowedText, a member of class TCustomActionControl.
	DrawShadowedText (see page 829)	This is DrawShadowedText, a member of class TCustomMenuItem.
	DrawSubMenuGlyph (see page 829)	This is DrawSubMenuGlyph, a member of class TCustomMenuItem.
	DrawText (see page 523)	Draws text on the control.
	DrawText (see page 829)	This is DrawText, a member of class TCustomMenuItem.
	DrawUnusedEdges (see page 830)	This is DrawUnusedEdges, a member of class TCustomMenuItem.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	GetAction (see page 523)	This is GetAction, a member of class TCustomActionControl.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetImageSize (see page 523)	This is GetImageSize, a member of class TCustomActionControl.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSeparator (see page 524)	This is GetSeparator, a member of class TCustomActionControl.
	GetShowCaption (see page 524)	This is GetShowCaption, a member of class TCustomActionControl.
	GetShowShortCut (see page 524)	This is GetShowShortCut, a member of class TCustomActionControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasGlyph (see page 524)	This is HasGlyph, a member of class TCustomActionControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionVisible (see page 755)	Indicates whether the add/remove item is visible in the menu of the customize dialog.
	IsChecked (see page 525)	This is IsChecked, a member of class TCustomActionControl.
	IsGrouped (see page 525)	This is IsGrouped, a member of class TCustomActionControl.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 755)	Responds when the user releases the mouse button when the mouse cursor is on this menu item.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Paint (see page 526)	Render control surface.
	Paint (see page 831)	Render control surface.

	Paint (see page 6052)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionChanged (see page 526)	This is PositionChanged, a member of class TCustomActionControl.
	PositionChanged (see page 831)	This is PositionChanged, a member of class TCustomMenuItem.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetUsageData (see page 526)	This is ResetUsageData, a member of class TCustomActionControl.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActionClient (see page 526)	This is SetActionClient, a member of class TCustomActionControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropPoint (see page 527)	This is SetDropPoint, a member of class TCustomActionControl.
	SetEnabled (see page 756)	Sets the value of the Enabled property.
	SetGlyphLayout (see page 527)	This is SetGlyphLayout, a member of class TCustomActionControl.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelected (see page 527)	This is SetSelected, a member of class TCustomActionControl.
	SetSelected (see page 831)	This is SetSelected, a member of class TCustomMenuItem.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateSelection (see page 527)	This is UpdateSelection, a member of class TCustomActionControl.
	UpdateTextBounds (see page 527)	This is UpdateTextBounds, a member of class TCustomActionControl.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.

	WMSetText (see page 528)	This is WMSetText, a member of class TCustomActionControl.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GlyphPos (see page 532)	This is GlyphPos, a member of class TCustomActionControl.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	Menu (see page 832)	This is Menu, a member of class TCustomMenuItem.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	MouseSelected (see page 832)	This is MouseSelected, a member of class TCustomMenuItem.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ShortCutBounds (see page 832)	This is ShortCutBounds, a member of class TCustomMenuItem.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnClick (see page 535)	Occurs when the user clicks the control.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
V	CalcBounds (see page 754)	Calculates the size of the add/remove menu item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Click (see page 518)	Executes the control's function in response to a mouse click.
V	Click (see page 827)	Executes the control's function.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 515)	Creates an instance of TCustomActionControl (see page 507).
V	Create (see page 825)	Creates an instance of TCustomMenuItem (see page 816).
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6051)	Creates an instance of TGraphicControl (see page 6044).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 516)	Destroys an instance of TCustomActionControl (see page 507).
V	Destroy (see page 825)	Destroys an instance of TCustomMenuItem (see page 816).
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
V	DragDrop (see page 520)	Performs actions and generates an OnDragDrop event.
V	DragDrop (see page 828)	Performs actions and generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.

	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 524)	Calls the action client's Update method if the control is associated with an action client.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
V	Keyed (see page 525)	Executes the control's function in response to a keyboard action.
V	Keyed (see page 830)	Executes the menu item's function in response to a user's keyboard command.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	ActionBar (see page 528)	The ActionBand object which contains the custom action control
	ActionClient (see page 529)	Refers to the control's action client
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Caption (see page 529)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DropPoint (see page 532)	This is DropPoint, a member of class TCustomActionControl.
	Edges (see page 832)	Dictates which edges are drawn when the menu item is drawn.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GlyphLayout (see page 532)	Describes where the image should be on the control
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Margins (see page 532)	Describes the boundaries of the control for drawing purposes.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	Selected (see page 533)	Indicates whether the control is currently selected.
R	Separator (see page 533)	Indicates whether the control is a separator.
R	ShowCaption (see page 533)	Indicates whether the control's caption should be shown.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	ShowShortCut (see page 533)	Indicates whether the control's short cut should be shown.
	SmallIcon (see page 534)	Indicates whether the control's small icon should be shown.

	Spacing (see page 534)	Controls (see page 5711) the separation between the control's icon and caption.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TextBounds (see page 534)	A bounding rectangle for the control caption.
	Transparent (see page 534)	Shows whether the control is transparent or not.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

5.1.6.2 TCustomAddRemoveItem Methods

The methods of the TCustomAddRemoveItem class are listed here.

Protected Methods

	Name	Description
	IsActionVisible (see page 755)	Indicates whether the add/remove item is visible in the menu of the customize dialog.
	MouseUp (see page 755)	Responds when the user releases the mouse button when the mouse cursor is on this menu item.
	SetEnabled (see page 756)	Sets the value of the Enabled property.

Public Methods

	Name	Description
	CalcBounds (see page 754)	Calculates the size of the add/remove menu item.

5.1.6.2.1 TCustomAddRemoveItem.CalcBounds Method

Calculates the size of the add/remove menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

CalcBounds updates the Height, Width, and TextBounds properties to reflect current conditions (such as the item's caption and the current font).

See Also

Height (see page 5825)

Width (see page 5849)

TextBounds (see page 534)

5.1.6.2.2 TCustomAddRemoveItem.IsActionVisible Method

Indicates whether the add/remove item is visible in the menu of the customize dialog.

Pascal

```
function IsActionVisible: Boolean;
```

C++

```
__fastcall Boolean IsActionVisible();
```

Description

IsActionVisible returns true if the Add/Remove item is currently visible in the menu of the customize dialog. The menu can automatically hide items that have not been used recently.

IsActionVisible locates the action client object that corresponds to this menu item, if any. If this menu item has a corresponding action client in the action manager, IsActionVisible returns the value of that object's Visible property. If this menu item does not have a corresponding action client, IsActionVisible returns false.

See Also

Visible (see page 374)

Action (see page 382)

ActionClient (see page 529)

5.1.6.2.3 TCustomAddRemoveItem.MouseUp Method

Responds when the user releases the mouse button when the mouse cursor is on this menu item.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

MouseUp is called automatically when the user releases the mouse button over this menu item. This method generates an OnMouseUp event, and forces a repaint of the menu item.

Button indicates which mouse button the user released (left, right, or middle).

Shift indicates the state of the Ctrl, Alt, and Shift keys when the mouse was released.

X and Y indicate the screen coordinates of the mouse cursor's hotspot.

See Also

OnMouseUp (see page 5867)

5.1.6.2.4 TCustomAddRemoveItem.SetEnabled Method

Sets the value of the Enabled property.

Pascal

```
procedure SetEnabled(Value: Boolean); override;
```

C++

```
virtual __fastcall SetEnabled(Boolean Value);
```

Description

SetEnabled is the protected write implementation of the Enabled property.

Value is the new value to set. If Value is true, the add/remove menu item is enabled so that users can click it. If Value is false, the add/remove item is disabled, so that it appears gray and has no effect when clicked.

See Also

Enabled (see page 5821)

5.1.7 TCustomizeActionClientItem Class

Class Hierarchy

System::TObject → Classes.TPersistent → Classes.TCollectionItem → ActnMan.TActionClient → ActnMan.TActionClientItem → ActnMenus.TCustomizeActionClientItem

Pascal

```
TCustomizeActionClientItem = class(TActionClientItem);
```

C++

```
class TCustomizeActionClientItem : public TActionClientItem;
```

File

ActnMenus

Description

This is class ActnMenus.TCustomizeActionClientItem.

5.1.7.1 TCustomizeActionClientItem Members

The following tables list the members exposed by TCustomizeActionClientItem.

Protected Methods

	Name	Description
	ActionChange (see page 378)	This is ActionChange, a member of class TActionClientItem.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	ClearActionBar (see page 367)	This is ClearActionBar, a member of class TActionClient.

	ColorChanged (see page 367)	This is ColorChanged, a member of class TActionClient.
	ColorChanged (see page 379)	This is ColorChanged, a member of class TActionClientItem.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FreeChildren (see page 368)	This is FreeChildren, a member of class TActionClient.
	GetActionLinkClass (see page 759)	This is GetActionLinkClass, a member of class TCustomizeActionClientItem.
	GetDisplayName (see page 379)	Returns the name of the collection item as it appears in the collection editor.
	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	HasVisibleItems (see page 368)	This is HasVisibleItems, a member of class TActionClient.
	IsChildItem (see page 368)	This is IsChildItem, a member of class TActionClient.
	ResetUsageData (see page 380)	This is ResetUsageData, a member of class TActionClientItem.
	SetChangesAllowed (see page 369)	This is SetChangesAllowed, a member of class TActionClient.
	SetChangesAllowed (see page 380)	This is SetChangesAllowed, a member of class TActionClientItem.
	SetCollection (see page 381)	Sets the value of the Collection property.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetColor (see page 369)	This is SetColor, a member of class TActionClient.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 381)	Moves the collection item to a specified position in the parent collection.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.
	SetVisible (see page 369)	This is SetVisible, a member of class TActionClient.
	SetVisible (see page 381)	This is SetVisible, a member of class TActionClientItem.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 367)	Copies the contents of another object to the current one.
	Assign (see page 378)	Copies the contents of another TActionClientItem (see page 374) to the object where the method is executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 366)	Creates and initializes an action client object.
	Create (see page 377)	Creates and initializes a TActionClientItem (see page 374) object.
	Create (see page 2757)	Creates and initializes a TCollectionItem (see page 2755) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 366)	Destroys the action client and frees its memory.
	Destroy (see page 377)	Destroys the action client and frees its memory.
	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	HasBackground (see page 368)	Indicates whether a background image exists.
	HasGlyph (see page 379)	Indicates whether or not there is a glyph (small icon) associated with an action on the action band.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 380)	Updates the action associated with the action client item.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Refresh (see page 369)	Causes the action client to be redrawn.
V	Refresh (see page 380)	Causes the size of the action client item's control (if any) to be recalculated.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Unused (see page 382)	Determines whether a TActionClientItem (see page 374) is unused and should be hidden on an action band.

Public Properties

	Name	Description
	Accessible (see page 370)	This is Accessible, a member of class TActionClient.
	ActionBar (see page 370)	Specifies the action band the action client is rendered on.
R	ActionClients (see page 383)	Returns the TActionClients (see page 400) object that contains the TActionClientItem (see page 374).
	ActionLink (see page 383)	Refers to the action client item's action link.
	CheckUnused (see page 384)	Indicates whether the TActionClientItem (see page 374) object should be checked to see if it is unused and, if so, hidden from view.
	ChildActionBar (see page 372)	Specifies the action band that is used to render the TActionClients (see page 400) object specified in the Items (see page 373) property.
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	Control (see page 384)	Specifies the action band control that is rendered on an action band for the TActionClientItem (see page 374) object.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	HasItems (see page 373)	Specifies whether the action client has sub-items associated with the action client.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
R	OwningCollection (see page 373)	This is OwningCollection, a member of class TActionClient.
R	ParentItem (see page 386)	Refers to the parent item of the action client item.
R	Separator (see page 386)	Returns whether the TActionClientItem (see page 374) is rendered as a separator control on an action band.
R	ShortCutText (see page 387)	Contains the text label (like "Ctrl+S") for the shortcut.

Published Properties

	Name	Description
	Action (see page 382)	Specifies the action category or action that the TActionClientItem (see page 374) represents.
	Background (see page 370)	Specifies the image that is drawn on the action client's action band for the action client.
	BackgroundLayout (see page 371)	Specifies how the image specified in the Background (see page 370) property is rendered on the action band used by the action client to render any sub-items that are contained by the action client.

	Caption (see page 384)	Specifies the caption the action manager uses when the TActionClientItem (see page 374) is rendered on an action band.
	ChangesAllowed (see page 371)	Indicates which kinds of changes to the action client are permissible.
	Color (see page 372)	Specifies the background color for TClientAction when it is rendered on a action band.
	ContextItems (see page 372)	Specifies the TActionClients (see page 400) object that is rendered as a popup menu when a item is right clicked.
	HelpContext (see page 385)	This is HelpContext, a member of class TActionClientItem.
	ImageIndex (see page 385)	Contains the ImageIndex property for the client action item.
	Items (see page 373)	Specifies the TActionClients (see page 400) object that contains the sub-items of the action client.
	LastSession (see page 385)	Specifies the date an action client was last used.
	MergeIndex (see page 386)	Used to help merge action lists.
	ShortCut (see page 387)	Contains the ShortCut property for the action client item.
	ShowCaption (see page 387)	Determines whether the caption for the TActionClientItem (see page 374) is drawn.
	ShowGlyph (see page 388)	Indicates whether an image should be shown for the action.
	ShowShortCut (see page 388)	Determines whether the underbar character underneath the shortcut character is displayed.
	Tag (see page 374)	This is Tag, a member of class TActionClient.
	UsageCount (see page 388)	Specifies the total number of times an item has been used.
	Visible (see page 374)	Determines whether the action client is visible.

5.1.7.2 TCustomizeActionClientItem Methods

The methods of the TCustomizeActionClientItem class are listed here.

Protected Methods

	Name	Description
	GetActionLinkClass (see page 759)	This is GetActionLinkClass, a member of class TCustomizeActionClientItem.

5.1.7.2.1 TCustomizeActionClientItem.GetActionLinkClass Method

Pascal

```
function GetActionLinkClass: TActionClientLinkClass; override;
```

C++

```
virtual __fastcall TActionClientLinkClass GetActionLinkClass();
```

Description

This is GetActionLinkClass, a member of class TCustomizeActionClientItem.

5.1.8 TCustomizeActionLink Class

Class Hierarchy

Pascal

```
TCustomizeActionLink = class(TActionClientLink);
```

C++

```
class TCustomizeActionLink : public TActionClientLink;
```

File

ActnMenus

Description

This is class ActnMenus.TCustomizeActionLink.

5.1.8.1 TCustomizeActionLink Members

The following tables list the members exposed by TCustomizeActionLink.

Protected Fields

	Name	Description
	FAction (see page 2722)	This is FAction, a member of class TBasicActionLink.
	FClient (see page 391)	This is FClient, a member of class TActionClientLink.

Protected Methods

	Name	Description
	AssignClient (see page 392)	This is AssignClient, a member of class TActionClientLink.
	AssignClient (see page 2723)	This is AssignClient, a member of class TBasicActionLink.
	Change (see page 2723)	This is Change, a member of class TBasicActionLink.
	DoShowHint (see page 392)	This is DoShowHint, a member of class TActionClientLink.
	IsCaptionLinked (see page 268)	Indicates whether the Caption properties for the action and client are linked.
	IsCaptionLinked (see page 392)	Indicates whether the Caption properties for the action and client are linked.
	IsCheckedLinked (see page 269)	Indicates whether the Checked properties for the action and client are linked.
	IsEnabledLinked (see page 269)	Indicates whether the Enabled properties for the action and client are linked.
	IsEnabledLinked (see page 393)	Indicates whether the Enabled properties for the action and client are linked.
	IsGroupIndexLinked (see page 270)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsHelpContextLinked (see page 270)	Indicates whether the HelpContext properties for the action and client controls are linked.
	IsHelpContextLinked (see page 393)	Indicates whether the HelpContext properties for the action and client controls are linked.
	IsHelpLinked (see page 271)	Indicates whether the help properties for the action and client are linked.
	IsHintLinked (see page 271)	Indicates whether the Hint properties for the action and client are linked.
	IsHintLinked (see page 394)	Indicates whether the Hint properties for the action and client are linked.
	IsImageIndexLinked (see page 272)	Indicates whether the image index for the action and the client are linked.
	IsImageIndexLinked (see page 394)	Indicates whether the image index for the action and the client are linked.
	IsOnExecuteLinked (see page 395)	This is IsOnExecuteLinked, a member of class TActionClientLink.
	IsOnExecuteLinked (see page 2723)	This is IsOnExecuteLinked, a member of class TBasicActionLink.
	IsShortCutLinked (see page 272)	Indicates whether the ShortCut properties for the action and client are linked.
	IsShortCutLinked (see page 395)	Indicates whether the ShortCut properties for the action and client are linked.
	IsVisibleLinked (see page 273)	Indicates whether the Visible properties for the action and client are linked.
	IsVisibleLinked (see page 396)	Indicates whether the Visible properties for the action and client are linked.
	SetAction (see page 2724)	This is SetAction, a member of class TBasicActionLink.
	SetAutoCheck (see page 274)	Introduces an interface for setting the AutoCheck property of a client.
	SetCaption (see page 274)	Introduces an interface for setting the caption on a client.
	SetCaption (see page 396)	Introduces an interface for setting the caption on a client.
	SetChecked (see page 274)	Introduces an interface for setting the checked state of a client.
	SetChecked (see page 397)	Introduces an interface for setting the checked state of a client.
	SetEnabled (see page 762)	Introduces an interface for setting the enabled state of a client.
	SetGroupIndex (see page 275)	Introduces an interface for setting the GroupIndex property of a client.

	SetHelpContext (see page 276)	Introduces an interface for setting the help context of a client.
	SetHelpContext (see page 398)	Introduces an interface for setting the help context of a client.
	SetHelpKeyword (see page 276)	Introduces an interface for setting the help keyword of a client.
	SetHelpType (see page 277)	Introduces an interface for setting the HelpType property of a client.
	SetHint (see page 278)	Introduces an interface for setting the hint for a client.
	SetHint (see page 398)	Introduces an interface for setting the hint for a client.
	SetImageIndex (see page 278)	Introduces an interface for setting the image index for a client.
	SetImageIndex (see page 399)	Introduces an interface for setting the image index for a client.
	SetOnExecute (see page 399)	This is SetOnExecute, a member of class TActionClientLink.
	SetOnExecute (see page 2724)	This is SetOnExecute, a member of class TBasicActionLink.
	SetShortCut (see page 279)	Introduces an interface for setting the shortcut for a client.
	SetShortCut (see page 399)	Introduces an interface for setting the shortcut for a client.
	SetVisible (see page 279)	Introduces an interface for setting the visible state of a client.
	SetVisible (see page 400)	Introduces an interface for setting the visible state of a client.

Public Events

	Name	Description
	OnChange (see page 2725)	Occurs when an OnChange event is triggered on the action.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2721)	Instantiates and initializes a TBasicActionLink (see page 2720) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2722)	Destroys an instance of an action link object.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 2723)	Invokes the action's Execute method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Update (see page 2724)	Invokes the action's Update method.

Public Properties

	Name	Description
	Action (see page 2725)	Indicates which action the link connects to its client.

5.1.8.2 TCustomizeActionLink Methods

The methods of the TCustomizeActionLink class are listed here.

Protected Methods

	Name	Description
	SetEnabled (see page 762)	Introduces an interface for setting the enabled state of a client.

5.1.8.2.1 TCustomizeActionLink.SetEnabled Method

Introduces an interface for setting the enabled state of a client.

Pascal

```
procedure SetEnabled(Value: Boolean); override;
```

C++

```
virtual __fastcall SetEnabled(Boolean Value);
```

Description

The associated action calls SetEnabled when its Enabled property changes so that the action link can propagate the new Enabled value to the client object.

Value specifies the new value of the Enabled property.

As implemented in TActionLink, SetEnabled does nothing. Descendant classes override SetEnabled to set the client property that corresponds to the action's Enabled property if the IsEnabledLinked method returns true.

See Also

Enabled (see page 310)

Enabled (see page 5821)

Action (see page 2725)

IsEnabledLinked (see page 269)

5.1.9 TCustomizeActionToolBar Class

Class Hierarchy

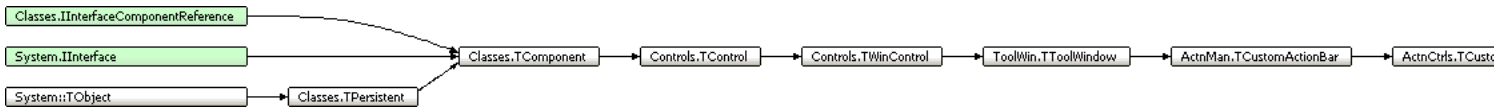

Pascal

```
TCustomizeActionToolBar = class(TCustomActionPopupMenu);
```

C++

```
class TCustomizeActionToolBar : public TCustomActionPopupMenu;
```

File

ActnMenus

Description

This is class ActnMenus.TCustomizeActionToolBar.

5.1.9.1 TCustomizeActionToolBar Members

The following tables list the members exposed by TCustomizeActionToolBar.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.

	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FCachedMenu (see page 699)	This is FCachedMenu, a member of class TCustomActionMenuBar.
	FChildMenu (see page 699)	This is FChildMenu, a member of class TCustomActionMenuBar.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefaultColorMap (see page 460)	This is FDefaultColorMap, a member of class TCustomActionBar.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExpandTimer (see page 700)	This is FExpandTimer, a member of class TCustomActionMenuBar.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FPopupTimer (see page 700)	This is FPopupTimer, a member of class TCustomActionMenuBar.
	FSavedWrapState (see page 460)	This is FSavedWrapState, a member of class TCustomActionBar.
	FSelectedItem (see page 700)	This is FSelectedItem, a member of class TCustomActionMenuBar.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AddEdges (see page 737)	This is AddEdges, a member of class TCustomActionPopupMenu.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 462)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	Animate (see page 701)	This is Animate, a member of class TCustomActionMenuBar.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
V	AutoSizingChanged (see page 462)	This is AutoSizingChanged, a member of class TCustomActionBar.
	BackgroundSize (see page 462)	This is BackgroundSize, a member of class TCustomActionBar.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcInsertPosition (see page 462)	This is CalcInsertPosition, a member of class TCustomActionBar.
V	CanAutoSize (see page 778)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.

	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CleanupStack (see page 701)	This is CleanupStack, a member of class TCustomActionMenuBar.
	Clear (see page 778)	This is Clear, a member of class TCustomizeActionToolBar.
	ClearSubMenus (see page 702)	This is ClearSubMenus, a member of class TCustomActionMenuBar.
	Click (see page 5743)	Respond to user click.
	CMBorderChanged (see page 23285)	This is CMBorderChanged, a member of class TToolWindow.
	CMCtl3DChanged (see page 23286)	This is CMCtl3DChanged, a member of class TToolWindow.
	CMMouseLeave (see page 738)	This is CMMouseLeave, a member of class TCustomActionPopupMenu.
	CMVisibleChanged (see page 738)	This is CMVisibleChanged, a member of class TCustomActionPopupMenu.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateControl (see page 463)	This is CreateControl, a member of class TCustomActionBar.
	CreateControl (see page 702)	This is CreateControl, a member of class TCustomActionMenuBar.
	CreateControl (see page 738)	This is CreateControl, a member of class TCustomActionPopupMenu.
	CreateControls (see page 463)	This is CreateControls, a member of class TCustomActionBar.
	CreateControls (see page 738)	This is CreateControls, a member of class TCustomActionPopupMenu.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 739)	Initializes a window-creation parameter data structure.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreatePopup (see page 702)	This is CreatePopup, a member of class TCustomActionMenuBar.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 463)	Returns placement order of custom-aligned child control.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 464)	Returns aligned position of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 465)	This is DesignWndProc, a member of class TCustomActionBar.
	DesignWndProc (see page 739)	This is DesignWndProc, a member of class TCustomActionPopupMenu.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DisplayShadow (see page 739)	This is DisplayShadow, a member of class TCustomActionPopupMenu.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.

	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
V	DoControlCreated (see page 465)	This is DoControlCreated, a member of class TCustomActionBar.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
V	DoDropActions (see page 465)	This is DoDropActions, a member of class TCustomActionBar.
V	DoDropCategory (see page 465)	This is DoDropCategory, a member of class TCustomActionBar.
V	DoDropItem (see page 466)	This is DoDropItem, a member of class TCustomActionBar.
V	DoEndDock (see page 146)	OnEndDock event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetControlClass (see page 466)	This is DoGetControlClass, a member of class TCustomActionBar.
	DoGetPopupClass (see page 703)	This is DoGetPopupClass, a member of class TCustomActionMenuBar.
V	DoItemClicked (see page 778)	This is DoItemClicked, a member of class TCustomizeActionToolBar.
V	DoItemKeyed (see page 703)	This is DoItemKeyed, a member of class TCustomActionMenuBar.
	DoItemSelected (see page 703)	This is DoItemSelected, a member of class TCustomActionMenuBar.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMenuDelay (see page 703)	This is DoMenuDelay, a member of class TCustomActionMenuBar.
	DoMouseIdle (see page 704)	This is DoMouseIdle, a member of class TCustomActionMenuBar.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
V	DoneMenuLoop (see page 704)	This is DoneMenuLoop, a member of class TCustomActionMenuBar.
V	DoPopup (see page 704)	This is DoPopup, a member of class TCustomActionMenuBar.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
V	DoStartDock (see page 147)	OnStartDock event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 466)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawBackground (see page 467)	This is DrawBackground, a member of class TCustomActionBar.
V	DrawBackground (see page 740)	This is DrawBackground, a member of class TCustomActionPopupMenu.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	ExecAction (see page 704)	This is ExecAction, a member of class TCustomActionMenuBar.
V	ExecAction (see page 740)	This is ExecAction, a member of class TCustomActionPopupMenu.
V	Expand (see page 704)	Causes the menu bar to show all items, including the hidden ones.
	ExpandClick (see page 740)	This is ExpandClick, a member of class TCustomActionPopupMenu.
	ExpandTimer (see page 705)	This is ExpandTimer, a member of class TCustomActionMenuBar.
	FindAccelItem (see page 467)	This is FindAccelItem, a member of class TCustomActionBar.
	FindItemAt (see page 468)	This is FindItemAt, a member of class TCustomActionBar.
	FindItemCaption (see page 468)	This is FindItemCaption, a member of class TCustomActionBar.
V	FindLast (see page 468)	This is FindLast, a member of class TCustomActionBar.
	FindLeastUsedItem (see page 469)	This is FindLeastUsedItem, a member of class TCustomActionBar.
	FindNearestControl (see page 469)	This is FindNearestControl, a member of class TCustomActionBar.
V	FindNext (see page 470)	This is FindNext, a member of class TCustomActionBar.

	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FindNextVisibleItem (see page 470)	This is FindNextVisibleItem, a member of class TCustomActionBar.
	FindPrevious (see page 470)	This is FindPrevious, a member of class TCustomActionBar.
	FindPreviousVisibleItem (see page 470)	This is FindPreviousVisibleItem, a member of class TCustomActionBar.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionControl (see page 779)	This is GetActionControl, a member of class TCustomizeActionToolBar.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetAddRemoveItemClass (see page 779)	This is GetAddRemoveItemClass, a member of class TCustomizeActionToolBar.
	GetBannerWidth (see page 471)	This is GetBannerWidth, a member of class TCustomActionBar.
	GetBarHeight (see page 471)	This is GetBarHeight, a member of class TCustomActionBar.
	GetBarWidth (see page 471)	This is GetBarWidth, a member of class TCustomActionBar.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 741)	Returns the value of the ClientRect property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlClass (see page 471)	This is GetControlClass, a member of class TCustomActionBar.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDefaultColorMapClass (see page 472)	This is GetDefaultColorMapClass, a member of class TCustomActionBar.
	GetDefaultSounds (see page 705)	This is GetDefaultSounds, a member of class TCustomActionMenuBar.
	GetDesignMode (see page 472)	This is GetDesignMode, a member of class TCustomActionBar.
	GetDesignMode (see page 705)	This is GetDesignMode, a member of class TCustomActionMenuBar.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetExpandBtnClass (see page 741)	This is GetExpandBtnClass, a member of class TCustomActionPopupMenu.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 147)	Returns the class used for the control when it is floating.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetMouseHoverItem (see page 705)	This is GetMouseHoverItem, a member of class TCustomActionMenuBar.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupClass (see page 706)	This is GetPopupClass, a member of class TCustomActionMenuBar.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetPopupMenuClass (see page 472)	This is GetPopupMenuClass, a member of class TCustomActionBar.
	GetSelected (see page 706)	This is GetSelected, a member of class TCustomActionMenuBar.
	GetSiteInfo (see page 147)	Generates an OnGetSiteInfo event
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetSubMenuControlClass (see page 779)	This is GetSubMenuControlClass, a member of class TCustomizeActionToolBar.
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.

	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasItems (see page 472)	This is HasItems, a member of class TCustomActionBar.
V	HideShadow (see page 741)	This is HideShadow, a member of class TCustomActionPopupMenu.
	Idle (see page 706)	This is Idle, a member of class TCustomActionMenuBar.
V	InitMenuLoop (see page 706)	This is InitMenuLoop, a member of class TCustomActionMenuBar.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsDesignMsg (see page 707)	This is IsDesignMsg, a member of class TCustomActionMenuBar.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 148)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 473)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 707)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
V	MouseActivate (see page 707)	This is MouseActivate, a member of class TCustomActionMenuBar.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	NCPaint (see page 473)	This is NCPaint, a member of class TCustomActionBar.
V	NCPaint (see page 23286)	This is NCPaint, a member of class TToolWindow.
V	NewPopup (see page 780)	This is NewPopup, a member of class TCustomizeActionToolBar.
V	Notification (see page 473)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 708)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 742)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 474)	This is Paint, a member of class TCustomActionBar.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 474)	Renders the image of a windowed control.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	PositionPopup (see page 742)	This is PositionPopup, a member of class TCustomActionPopupMenu.
V	ProcessMenuLoop (see page 708)	This is ProcessMenuLoop, a member of class TCustomActionMenuBar.
	ProcessMessage (see page 708)	This is ProcessMessage, a member of class TCustomActionMenuBar.
	ProcessMessages (see page 709)	This is ProcessMessages, a member of class TCustomActionMenuBar.
V	ProcessMouseMsg (see page 709)	This is ProcessMouseMsg, a member of class TCustomActionMenuBar.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.

	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Reset (see page 475)	This is Reset, a member of class TCustomActionBar.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Select (see page 709)	This is Select, a member of class TCustomActionMenuBar.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectItem (see page 710)	This is SelectItem, a member of class TCustomActionMenuBar.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActionClient (see page 780)	This is SetActionClient, a member of class TCustomizeActionToolBar.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetAutoSizing (see page 475)	This is SetAutoSizing, a member of class TCustomActionBar.
	SetBiDiMode (see page 475)	Sets the control's bi-directional mode.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetColorMap (see page 476)	This is SetColorMap, a member of class TCustomActionBar.
	SetColorMap (see page 743)	This is SetColorMap, a member of class TCustomActionPopupMenu.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignMode (see page 476)	This is SetDesignMode, a member of class TCustomActionBar.
	SetDesignMode (see page 710)	Specifies whether the whether the action band is being customized or being used in the IDE.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetOrientation (see page 477)	This is SetOrientation, a member of class TCustomActionBar.
	SetOrientation (see page 743)	This is SetOrientation, a member of class TCustomActionPopupMenu.
	SetParent (see page 710)	Sets the parent of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.

V	SetPersistentHotKeys (see page 477)	This is SetPersistentHotKeys, a member of class TCustomActionBar.
V	SetPersistentHotKeys (see page 711)	This is SetPersistentHotKeys, a member of class TCustomActionMenuBar.
V	SetSpacing (see page 477)	This is SetSpacing, a member of class TCustomActionBar.
	SetupDefaultColorMap (see page 477)	This is SetupDefaultColorMap, a member of class TCustomActionBar.
	SetupExpandBtn (see page 743)	This is SetupExpandBtn, a member of class TCustomActionPopupMenu.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	VisibleChanging (see page 478)	Provides the interface for a method that responds to changes in the control's visibility.
V	VisibleChanging (see page 743)	Provides the interface for a method that responds to changes in the control's visibility.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMKeyDown (see page 744)	This is WMKeyDown, a member of class TCustomActionPopupMenu.
	WMNCCalcSize (see page 23286)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCPaint (see page 23286)	This is WMNCPaint, a member of class TToolWindow.
	WMPrint (see page 744)	This is WMPrint, a member of class TCustomActionPopupMenu.
	WMPrintClient (see page 744)	This is WMPrintClient, a member of class TCustomActionPopupMenu.
V	WndProc (see page 711)	Provides specific message responses for the control.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
R	AddlItem (see page 780)	This is AddlItem, a member of class TCustomizeActionToolBar.
	AnimateDuration (see page 716)	This is AnimateDuration, a member of class TCustomActionMenuBar.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.

	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DelayItem (see page 717)	This is DelayItem, a member of class TCustomActionMenuBar.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Expandable (see page 717)	This is Expandable, a member of class TCustomActionMenuBar.
	Expanded (see page 718)	This is Expanded, a member of class TCustomActionMenuBar.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
R	HRowCount (see page 488)	Indicates the number of columns of items that the action band displays.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	Items (see page 488)	Lists the action clients that correspond to the child controls of this action band.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	MouseControl (see page 718)	This is MouseControl, a member of class TCustomActionMenuBar.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentMenu (see page 719)	This is ParentMenu, a member of class TCustomActionMenuBar.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
R	PopupStack (see page 719)	This is PopupStack, a member of class TCustomActionMenuBar.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
R	Selected (see page 719)	This is Selected, a member of class TCustomActionMenuBar.
	Text (see page 5842)	Contains a text string associated with the control.
R	VRowCount (see page 490)	Indicates the number of rows of items that the action band displays.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
R	Widest (see page 491)	This is Widest, a member of class TCustomActionBar.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnControlCreated (see page 491)	Occurs when a control is created on the action band.
	OnEnterMenuLoop (see page 720)	Occurs when an ActionBand menu loop has begun.
	OnExitMenuLoop (see page 721)	Occurs when an ActionBand menu loop has ended and the menu is closed.
	OnGetControlClass (see page 492)	Occurs when an item is to be placed on the action band.
	OnGetPopupClass (see page 721)	Occurs prior to an ActionBand popupmenu appearing on screen.
	OnPaint (see page 492)	Occurs when the action band is painted.
	OnPopup (see page 721)	Occurs just before the pop-up menu appears.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.

	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
V	CalcDockedWidth (see page 146)	This is CalcDockedWidth, a member of class TCustomActionDockBar.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	CloseMenu (see page 702)	Closes the menu currently being displayed.
V	CloseMenu (see page 737)	Closes the menu currently being displayed.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 777)	Creates and initializes a TCustomActionPopupMenu instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 777)	Destroys an instance of TCustomActionMenuBar.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Dock (see page 146)	Used internally to dock the action band.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
V	DragDrop (see page 466)	Generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	Expand (see page 740)	Performs the same kinds of functions as the inherited method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.

	FindFirst (see page 467)	Finds the first TActionClientItem (see page 374) in the Items (see page 488) list.
	FindFirstVisibleItem (see page 468)	Finds the first visible TActionClientItem (see page 374) in the Items (see page 488) list.
	FindLastVisibleItem (see page 469)	Finds the last visibleTActionClientItem in the Items (see page 488) list.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ItemCount (see page 779)	This is ItemCount, a member of class TCustomizeActionToolBar.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	Popup (see page 742)	Causes the popup menu to appear.

	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
V	RecreateControls (see page 475)	Destroys and recreates all of the action band's action client controls.
V	RecreateControls (see page 709)	Destroys and recreates all of the action band's action client controls.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 476)	Sets the bounds based on whether the popup is expanded or not
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Style (see page 477)	Identifies the style object that gives the application its default look and feel.
V	Style (see page 711)	Retrieves the style used by the ActionBand menu that started the menu loop so the submenus have a consistent appearance.
V	TrackMenu (see page 711)	Executes the action associated with the menu item.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	ActionClient (see page 479)	Specifies the action client associated with this action band.
R	ActionControls (see page 479)	Lists the controls on the action band.
	ActionManager (see page 480)	Specifies the action manager component that is associated with the action band.
R	AdditionalItem (see page 780)	This is AdditionalItem, a member of class TCustomizeActionToolBar.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AllowHiding (see page 480)	Determines whether the action band can be hidden.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.

	AnimationStyle (see page 716)	Specifies the type of animation the menu bar will display as it opens.
	AutoFocus (see page 717)	This is AutoFocus, a member of class TCustomActionMenuBar.
	AutoSizing (see page 480)	Specifies whether the action band adjusts its size automatically to fit around its child controls.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 481)	Represents the drawing surface of the action band.
	Caption (see page 481)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 483)	Specifies the background color of the control.
	ColorMap (see page 484)	Indicates the colormap for individual action band components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ContextBar (see page 485)	Indicates whether items shown in this action band represent the items in a popup menu.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designable (see page 485)	Specifies whether the action band can be customized (used internally by the IDE).
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignMode (see page 486)	Specifies whether the action band is being customized or being used in the IDE.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EdgeBorders (see page 23287)	Determines which sides of the control have visible borders.
	EdgeInner (see page 23287)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 23288)	Determines the border style for the control's outer edge.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
	ExpandBtn (see page 744)	Refers to the popup menu's expand button.
	ExpandDelay (see page 717)	Specifies, in milliseconds, the delay period that a menu action band uses before showing hidden or unused items.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.

	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Font (see page 486)	Controls the attributes of text written on or in the control.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HorzMargin (see page 487)	Specifies the width of the horizontal margins in pixels.
	HorzSeparator (see page 487)	Indicates whether a horizontal separator bar appears between item lines.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	InMenuLoop (see page 718)	Indicates whether items in the menu can be selected.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	Orientation (see page 488)	Specifies how the action band lays out its child items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentControl (see page 718)	Refers to the parent object of the TCustomActionBarControl.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PersistentHotKeys (see page 489)	Specifies whether keyboard accelerator shortcuts are always displayed.
	RootMenu (see page 719)	Refers to the root menu of the ActionBand menu.
	Shadow (see page 745)	Refers to the popup menu's shadow.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	Spacing (see page 489)	Determines where the image and text appear on an action band's rendered control that represents the action client.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	UseSystemFont (see page 720)	Specifies whether the default system font for menu items is used.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	VertMargin (see page 490)	Specifies the width of the vertical margins in pixels.
	VertSeparator (see page 490)	Indicates whether a vertical separator bar should appear between item lines.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.

	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

5.1.9.2 TCustomizeActionToolBar.Create Constructor

Creates and initializes a TCustomActionPopupMenu instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomizeActionToolBar(TComponent * AOwner);
```

Description

Create generates a TCustomActionPopupMenu and initializes its properties.

```
__fastcall virtual TCustomActionMenuBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall TCustomActionMenuBar (HWND ParentWindow) :
Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

5.1.9.3 TCustomizeActionToolBar.Destroy Destructor

Destroys an instance of TCustomActionMenuBar.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomizeActionToolBar();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TCustomActionMenuBar reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

Destroy (see page 5732)

5.1.9.4 TCustomizeActionToolBar Methods

The methods of the TCustomizeActionToolBar class are listed here.

Protected Methods

	Name	Description
	CanAutoSize (see page 778)	Indicates how the control can be resized.
	Clear (see page 778)	This is Clear, a member of class TCustomizeActionToolBar.
	DoItemClicked (see page 778)	This is DoItemClicked, a member of class TCustomizeActionToolBar.
	GetActionControl (see page 779)	This is GetActionControl, a member of class TCustomizeActionToolBar.
	GetAddRemoveItemClass (see page 779)	This is GetAddRemoveItemClass, a member of class TCustomizeActionToolBar.
	GetSubMenuControlClass (see page 779)	This is GetSubMenuControlClass, a member of class TCustomizeActionToolBar.
	NewPopup (see page 780)	This is NewPopup, a member of class TCustomizeActionToolBar.
	SetActionClient (see page 780)	This is SetActionClient, a member of class TCustomizeActionToolBar.

Public Methods

	Name	Description
	ItemCount (see page 779)	This is ItemCount, a member of class TCustomizeActionToolBar.

5.1.9.4.1 TCustomizeActionToolBar.CanAutoSize Method

Indicates how the control can be resized.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize property is true and an attempt is made to resize the control. It allows controls to implement the AutoSize property before the resize sequence that begins with the OnCanResize and OnConstrainedResize events and ends with the OnResize event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the control. CanAutoSize adjusts these values so that the control's size adjusts to its contents. If the adjustment results in values of NewWidth and NewHeight that differ from the control's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width and Height, CanAutoSize returns false, and the resize sequence is aborted.

As implemented in TWinControl, CanAutoSize returns true, adjusting NewWidth and NewHeight so that all the control's children fit.

See Also

Constraints (see page 5812)

ConstrainedResize (see page 6133)

AdjustSize (see page 6129)

5.1.9.4.2 TCustomizeActionToolBar.Clear Method

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

This is Clear, a member of class TCustomizeActionToolBar.

5.1.9.4.3 TCustomizeActionToolBar.DoItemClicked Method

Pascal

```
function DoItemClicked(AnItem: TCustomActionControl): TActionClientItem; override;
```

C++

```
virtual __fastcall TActionClientItem DoItemClicked(TCustomActionControl AnItem);
```

Description

This is DoItemClicked, a member of class TCustomizeActionToolBar.

5.1.9.4.4 TCustomizeActionToolBar.GetActionControl Method

Pascal

```
function GetActionControl(const Index: Integer): TCustomActionControl; override;
```

C++

```
virtual __fastcall TCustomActionControl GetActionControl(const int Index);
```

Description

This is GetActionControl, a member of class TCustomizeActionToolBar.

5.1.9.4.5 TCustomizeActionToolBar.GetAddRemoveItemClass Method

Pascal

```
function GetAddRemoveItemClass: TCustomAddRemoveItemClass; virtual;
```

C++

```
virtual __fastcall TCustomAddRemoveItemClass GetAddRemoveItemClass();
```

Description

This is GetAddRemoveItemClass, a member of class TCustomizeActionToolBar.

5.1.9.4.6 TCustomizeActionToolBar.GetSubMenuControlClass Method

Pascal

```
procedure GetSubMenuControlClass(Sender: TCustomActionBar; AnItem: TActionClient; var
ControlClass: TCustomActionControlClass); virtual;
```

C++

```
virtual __fastcall GetSubMenuControlClass(TCustomActionBar Sender, TActionClient AnItem,
TCustomActionControlClass ControlClass);
```

Description

This is GetSubMenuControlClass, a member of class TCustomizeActionToolBar.

5.1.9.4.7 TCustomizeActionToolBar.ItemCount Method

Pascal

```
function ItemCount: Integer; override;
```

C++

```
virtual __fastcall int ItemCount();
```

Description

This is ItemCount, a member of class TCustomizeActionToolBar.

5.1.9.4.8 TCustomizeActionToolBar.NewPopup Method

Pascal

```
function NewPopup: TCustomActionPopupMenu; override;
```

C++

```
virtual __fastcall TCustomActionPopupMenu NewPopup();
```

Description

This is NewPopup, a member of class TCustomizeActionToolBar.

5.1.9.4.9 TCustomizeActionToolBar.SetActionClient Method

Pascal

```
procedure SetActionClient(const Value: TActionClient); override;
```

C++

```
virtual __fastcall SetActionClient(const TActionClient Value);
```

Description

This is SetActionClient, a member of class TCustomizeActionToolBar.

5.1.9.5 TCustomizeActionToolBar Properties

The properties of the TCustomizeActionToolBar class are listed here.

Protected Properties

	Name	Description
R	AddlItem (see page 780)	This is AddlItem, a member of class TCustomizeActionToolBar.

Public Properties

	Name	Description
R	AdditionalItem (see page 780)	This is AdditionalItem, a member of class TCustomizeActionToolBar.

5.1.9.5.1 TCustomizeActionToolBar.AdditionalItem Property

Pascal

```
property AdditionalItem: TActionClientItem;
```

C++

```
__property TActionClientItem AdditionalItem;
```

Description

This is AdditionalItem, a member of class TCustomizeActionToolBar.

5.1.9.5.2 TCustomizeActionToolBar.AddlItem Property

Pascal

```
property AddlItem: TActionClientItem;
```

C++

```
__property TActionClientItem AddlItem;
```

Description

This is AddlItem, a member of class TCustomizeActionToolBar.

5.1.10 TCustomMDIMenuButton Class

Class Hierarchy

Pascal

```
TCustomMDIMenuButton = class(TSpeedButton);
```

C++

```
class TCustomMDIMenuButton : public TSpeedButton;
```

File

ActnMenus

Description

This is class ActnMenus.TCustomMDIMenuButton.

5.1.10.1 TCustomMDIMenuButton Members

The following tables list the members exposed by TCustomMDIMenuButton.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.

	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FState (see page 2345)	This is FState, a member of class TSpeedButton.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActionChange (see page 2345)	Responds when the action associated with the control changes.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	CopyImage (see page 2346)	This is CopyImage, a member of class TSpeedButton.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.

	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
V	GetActionLinkClass (see page 2346)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetPalette (see page 2347)	Returns the handle to a palette for use by the control.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2347)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 2348)	OnMouseDown event dispatcher.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
V	MouseMove (see page 2348)	Respond to mouse moving over control area..
	MouseMove (see page 5773)	Respond to mouse moving over control area..
V	MouseUp (see page 2349)	OnMouseUp event dispatcher.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 789)	Render control surface.

	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.

	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	MouseInControl (see page 2361)	This is MouseInControl, a member of class TSpeedButton.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Click (see page 2346)	Generates an OnClick (see page 2372) event or triggers the associated action.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 2344)	Creates and initializes an instance of TSpeedButton (see page 2335).
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6051)	Creates an instance of TGraphicControl (see page 6044).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2344)	Destroys an instance of TSpeedButton (see page 2335).
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).

	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	ButtonStyle (see page 790)	This is ButtonStyle, a member of class TCustomMDIMenuButton.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.

	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Events

	Name	Description
	OnClick (see page 2372)	Occurs when the user clicks the control.
	OnDblClick (see page 2374)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnMouseActivate (see page 2374)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2375)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2377)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2378)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2378)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2380)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Published Properties

	Name	Description
	Action (see page 2350)	Designates the action associated with the control.
	Align (see page 2350)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AllowAllUp (see page 2354)	Specifies whether all speed buttons in the group that contains this speed button can be unselected at the same time.
	Anchors (see page 2355)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2355)	Specifies the bi-directional mode for the control.
	Caption (see page 2356)	Specifies the caption, or label, that appears on the speed button.
	Constraints (see page 2356)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Down (see page 2357)	Specifies whether the button is selected (down) or unselected (up).
	Enabled (see page 2357)	Controls whether the control responds to mouse, keyboard, and timer events.
	Flat (see page 2357)	Determines whether the button has a 3D border that provides a raised or lowered look.
	Font (see page 2358)	Controls the attributes of text written on or in the control.
	Glyph (see page 2359)	Specifies the bitmap that appears on the speed button.

	GroupIndex (see page 2360)	Allows speed buttons to work together as a group.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Layout (see page 2360)	Determines where the image or text appears on the speed button.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margin (see page 2361)	Specifies the number of pixels between the edge of the button and the image or caption drawn on its surface.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NumGlyphs (see page 2361)	Specifies the number of images included in the Glyph (see page 2359) property.
	ParentBiDiMode (see page 2362)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 2362)	Determines where a control looks for its font information.
	ParentShowHint (see page 2363)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2363)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2368)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Spacing (see page 2369)	Determines where the image and text appear on a speed button.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Transparent (see page 2370)	Specifies whether the background of the button is transparent.
	Visible (see page 2370)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

5.1.10.2 TCustomMDIMenuButton Methods

The methods of the TCustomMDIMenuButton class are listed here.

Protected Methods

	Name	Description
	Paint (see page 789)	Render control surface.

5.1.10.2.1 TCustomMDIMenuButton.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas (see page 6053)

5.1.10.3 TCustomMDIMenuButton Properties

The properties of the TCustomMDIMenuButton class are listed here.

Public Properties

	Name	Description
	ButtonStyle (see page 790)	This is ButtonStyle, a member of class TCustomMDIMenuButton.

5.1.10.3.1 TCustomMDIMenuButton.ButtonStyle Property

Pascal

```
property ButtonStyle: TMDIButtonStyle;
```

C++

```
__property TMDIButtonStyle ButtonStyle;
```

Description

This is ButtonStyle, a member of class TCustomMDIMenuButton.

5.1.11 TCustomMenuButton Class

TCustomMenuButton is the base class for ActionBand button controls.

Class Hierarchy

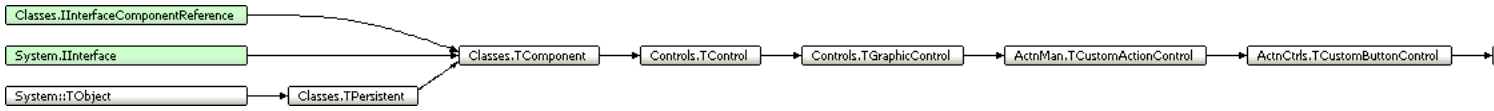

Pascal

```
TCustomMenuButton = class(TCustomButtonControl);
```

C++

```
class TCustomMenuButton : public TCustomButtonControl;
```

File

ActnMenus

Description

TCustomMenuButton is a base class for the standard menu button class TStandardMenuButton. Do not create instances of TCustomMenuButton. If you would like to replace the standard menu button behaviors, use the class TCustomMenuButton as a parent class for your replacement class. To make the change for your application, set the value of MenuButtonControlClass (see page 846) (a global variable) to your own subclass of TCustomMenuButton instead of TStandardMenuButton.

See Also

TCustomMenuItem (see page 816)

TStandardMenuButton (see page 20444)

TStandardMenuItem (see page 20465)

5.1.11.1 TCustomMenuButton Members

The following tables list the members exposed by TCustomMenuButton.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFlat (see page 178)	This is FFlat, a member of class TCustomButtonControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FState (see page 178)	This is FState, a member of class TCustomButtonControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	BeginAutoDrag (see page 801)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CalcLayout (see page 518)	This is CalcLayout, a member of class TCustomActionControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	CMDesignHitTest (see page 519)	This is CMDesignHitTest, a member of class TCustomActionControl.
	CMDialogChar (see page 180)	This is CMDialogChar, a member of class TCustomButtonControl.
	CMEnabledChanged (see page 180)	This is CMEnabledChanged, a member of class TCustomButtonControl.
	CMFontChanged (see page 519)	This is CMFontChanged, a member of class TCustomActionControl.
	CMItemDropPoint (see page 519)	This is CMItemDropPoint, a member of class TCustomActionControl.
	CMItemSelected (see page 519)	This is CMItemSelected, a member of class TCustomActionControl.
	CMMouseEnter (see page 801)	This is CMMouseEnter, a member of class TCustomMenuButton.
	CMMouseLeave (see page 180)	This is CMMouseLeave, a member of class TCustomButtonControl.
	CMTextChanged (see page 180)	This is CMTextChanged, a member of class TCustomButtonControl.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DesignWndProc (see page 519)	This is DesignWndProc, a member of class TCustomActionControl.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
V	DoDragDrop (see page 802)	This is DoDragDrop, a member of class TCustomMenuButton.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
V	DoStartDrag (see page 520)	OnStartDrag event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 802)	OnDragOver event dispatcher.
V	DrawBackground (see page 181)	Draws a background image on the control.
V	DrawBackground (see page 521)	Draws a background image on the control.

	DrawDesignFocus (see page 521)	This is DrawDesignFocus, a member of class TCustomActionControl.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawDragDropPoint (see page 522)	This is DrawDragDropPoint, a member of class TCustomActionControl.
	DrawFrame (see page 181)	This is DrawFrame, a member of class TCustomButtonControl.
	DrawGlyph (see page 522)	Draws a symbol on the control.
	DrawLargeGlyph (see page 181)	This is DrawLargeGlyph, a member of class TCustomButtonControl.
	DrawLargeGlyph (see page 522)	This is DrawLargeGlyph, a member of class TCustomActionControl.
	DrawSeparator (see page 522)	This is DrawSeparator, a member of class TCustomActionControl.
	DrawShadowedText (see page 523)	This is DrawShadowedText, a member of class TCustomActionControl.
	DrawText (see page 802)	Draws text on the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	GetAction (see page 523)	This is GetAction, a member of class TCustomActionControl.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetImageSize (see page 523)	This is GetImageSize, a member of class TCustomActionControl.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSeparator (see page 524)	This is GetSeparator, a member of class TCustomActionControl.
	GetShowCaption (see page 524)	This is GetShowCaption, a member of class TCustomActionControl.
	GetShowShortCut (see page 181)	This is GetShowShortCut, a member of class TCustomButtonControl.
	GetShowShortCut (see page 524)	This is GetShowShortCut, a member of class TCustomActionControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasGlyph (see page 524)	This is HasGlyph, a member of class TCustomActionControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsChecked (see page 525)	This is IsChecked, a member of class TCustomActionControl.
	IsGrouped (see page 525)	This is IsGrouped, a member of class TCustomActionControl.
	Loaded (see page 182)	Initializes the control after it is loaded from a stream.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 803)	OnMouseDown event dispatcher.
	MouseMove (see page 183)	Respond to mouse moving over control area..
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 804)	OnMouseUp event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.

	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Paint (see page 804)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionChanged (see page 526)	This is PositionChanged, a member of class TCustomActionControl.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetUsageData (see page 526)	This is ResetUsageData, a member of class TCustomActionControl.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActionClient (see page 184)	This is SetActionClient, a member of class TCustomButtonControl.
	SetActionClient (see page 526)	This is SetActionClient, a member of class TCustomActionControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropPoint (see page 527)	This is SetDropPoint, a member of class TCustomActionControl.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetGlyphLayout (see page 185)	This is SetGlyphLayout, a member of class TCustomButtonControl.
	SetGlyphLayout (see page 527)	This is SetGlyphLayout, a member of class TCustomActionControl.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelected (see page 185)	This is SetSelected, a member of class TCustomButtonControl.
	SetSelected (see page 527)	This is SetSelected, a member of class TCustomActionControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateSelection (see page 527)	This is UpdateSelection, a member of class TCustomActionControl.
	UpdateTextBounds (see page 527)	This is UpdateTextBounds, a member of class TCustomActionControl.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.

V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMLButtonDblClk (see page 185)	This is WMLButtonDblClk, a member of class TCustomButtonControl.
	WMSetText (see page 528)	This is WMSetText, a member of class TCustomActionControl.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GlyphPos (see page 532)	This is GlyphPos, a member of class TCustomActionControl.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	Menu (see page 805)	This is Menu, a member of class TCustomMenuButton.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	MouseInControl (see page 189)	This is MouseInControl, a member of class TCustomButtonControl.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnClick (see page 535)	Occurs when the user clicks the control.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
V	CalcBounds (see page 179)	Calculates the size of the control.
V	CalcBounds (see page 518)	Calculates the size of the control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Click (see page 801)	Executes the control's function.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 799)	Creates an instance of TCustomMenuButton (see page 790).
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 800)	Destroys an instance of TCustomMenuButton (see page 790).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
V	DragDrop (see page 520)	Performs actions and generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 524)	Calls the action client's Update method if the control is associated with an action client.

V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
V	Keyed (see page 803)	Executes the button's function in response to a keyboard command.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 184)	Sets the Left, Top, Width, and Height properties all at once.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	ActionBar (see page 528)	The ActionBand object which contains the custom action control
	ActionClient (see page 529)	Refers to the control's action client
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	AllowAllUp (see page 186)	Indicates whether all the buttons in a given group can be up at the same time.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.

	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Caption (see page 186)	Specifies a text string that identifies the control to the user.
	Caption (see page 529)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Down (see page 189)	Tells whether the button is down (selected) or up (unselected).
	DropPoint (see page 532)	This is DropPoint, a member of class TCustomActionControl.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
	Flat (see page 189)	Dictates whether the button should have a 2D look instead of the usual 3D look.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GlyphLayout (see page 532)	Describes where the image should be on the control
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Margins (see page 532)	Describes the boundaries of the control for drawing purposes.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	Selected (see page 533)	Indicates whether the control is currently selected.
R	Separator (see page 533)	Indicates whether the control is a separator.
	ShowCaption (see page 189)	Indicates whether the control's caption should be shown.
R	ShowCaption (see page 533)	Indicates whether the control's caption should be shown.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
	ShowShortCut (see page 190)	Indicates whether the control's short cut should be shown.
R	ShowShortCut (see page 533)	Indicates whether the control's short cut should be shown.
	SmallIcon (see page 534)	Indicates whether the control's small icon should be shown.
	Spacing (see page 534)	Controls (see page 5711) the separation between the control's icon and caption.
	State (see page 190)	Describes the current state of the button.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TextBounds (see page 534)	A bounding rectangle for the control caption.

	TrackButton (see page 190)	Specifies which mouse button activates the button control.
	Transparent (see page 534)	Shows whether the control is transparent or not.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

5.1.11.2 TCustomMenuButton.Create Constructor

Creates an instance of TCustomMenuButton (see page 790).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomMenuButton(TComponent * AOwner);
```

Description

Calling Create constructs and initializes an instance of TCustomMenuButton (see page 790). However, you should never attempt to instantiate a TCustomMenuButton (see page 790). This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component that is responsible for freeing the menu button instance. It becomes the value of the Owner property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner (see page 2807)

Create (see page 5731)

Destroy (see page 800)

5.1.11.3 TCustomMenuButton.Destroy Destructor

Destroys an instance of TCustomMenuButton (see page 790).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomMenuButton();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the graphic control reference is not nil, and only then calls Destroy.

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Frees the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create (see page 799) method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

Create (see page 799)

Free (see page 22300)

5.1.11.4 TCustomMenuButton Methods

The methods of the TCustomMenuButton class are listed here.

Protected Methods

	Name	Description
	BeginAutoDrag (see page 801)	Begin Automatic Drag.
	CMMouseEnter (see page 801)	This is CMMouseEnter, a member of class TCustomMenuButton.
	DoDragDrop (see page 802)	This is DoDragDrop, a member of class TCustomMenuButton.
	DragOver (see page 802)	OnDragOver event dispatcher.
	DrawText (see page 802)	Draws text on the control.
	MouseDown (see page 803)	OnMouseDown event dispatcher.
	MouseUp (see page 804)	OnMouseUp event dispatcher.
	Paint (see page 804)	Render control surface.

Public Methods

	Name	Description
	Click (see page 801)	Executes the control's function.
	Keyed (see page 803)	Executes the button's function in response to a keyboard command.

5.1.11.4.1 TCustomMenuButton.BeginAutoDrag Method

Begin Automatic Drag.

Pascal

```
procedure BeginAutoDrag; override;
```

C++

```
virtual __fastcall BeginAutoDrag();
```

Description

BeginAutoDrag is called automatically when a user drags a control whose DragMode is dmAutomatic. Do not call BeginAutoDrag in application code.

As implemented in TControl, BeginAutoDrag calls BeginDrag, passing in the global Mouse variable's DragImmediate and DragThreshold properties.

See Also

DragKind (see page 5819)

DragMode (see page 5819)

EndDrag (see page 5757)

Mouse (see page 6321)

DragImmediate (see page 6100)

DragThreshold (see page 6101)

Mouse (see page 6321)

5.1.11.4.2 TCustomMenuButton.Click Method

Executes the control's function.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

This override of the method inherited from TCustomButtonControl is essentially identical to the inherited method, with the exception of a few internal calls.

See Also

Click (see page 518)

Click (see page 179)

5.1.11.4.3 TCustomMenuButton.CMMouseEnter Method

Pascal

```
procedure CMMouseEnter(var Message: TMessage); message CM_MOUSEENTER;
```

C++

```
__fastcall CMMouseEnter(TMessage Message);
```

Description

This is CMMouseEnter, a member of class TCustomMenuButton.

5.1.11.4.4 TCustomMenuButton.DoDragDrop Method

Pascal

```
procedure DoDragDrop(DragObject: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DoDragDrop(TObject * DragObject, int X, int Y);
```

Description

This is DoDragDrop, a member of class TCustomMenuButton.

5.1.11.4.5 TCustomMenuButton.DragOver Method

OnDragOver event dispatcher.

Pascal

```
procedure DragOver(Source: TObject; X: Integer; Y: Integer; State: TDragState; var Accept:
Boolean); override;
```

C++

```
virtual __fastcall DragOver(TObject * Source, int X, int Y, TDragState State, Boolean Accept);
```

Description

Override DragOver to add additional code that executes before the OnDragOver event handler is called.

DragOver sets the Accept parameter to true to indicate that the user can drop the dragged object on the control. It sets Accept to false to indicate that the user cannot drop the dragged object on the control.

The Source parameter is the object being dragged. The State parameter indicates how the dragged object is moving in relation to the control. X and Y indicate the current position of the mouse.

See Also

DragDrop (see page 5754)

OnDragOver (see page 5857)

5.1.11.4.6 TCustomMenuButton.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, AnsiString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground (see page 521)

DrawGlyph (see page 522)

5.1.11.4.7 TCustomMenuButton.Keyed Method

Executes the button's function in response to a keyboard command.

Pascal

```
procedure Keyed; override;
```

C++

```
virtual __fastcall Keyed();
```

Description

This method is mostly the same as the inherited method. The only difference is that the button becomes selected.

See Also

Keyed (see page 525)

5.1.11.4.8 TCustomMenuButton.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove (see page 5773)

MouseUp (see page 5774)

OnMouseDown (see page 5862)

5.1.11.4.9 TCustomMenuButton.MouseUp Method

OnMouseUp event dispatcher.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user releases a previously pressed mouse button while the cursor's hotspot is over the control.

A control calls MouseUp in response to any of the Windows mouse-up messages (WM_LBUTTONUP, WM_MBUTTONUP, WM_RBUTTONUP), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

See Also

MouseDown (see page 5773)

MouseMove (see page 5773)

OnMouseUp (see page 5867)

5.1.11.4.10 TCustomMenuButton.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas (see page 6053)

5.1.11.5 TCustomMenuButton Properties

The properties of the TCustomMenuButton class are listed here.

Protected Properties

	Name	Description
R	Menu (see page 805)	This is Menu, a member of class TCustomMenuButton.

5.1.11.5.1 TCustomMenuButton.Menu Property

Pascal

```
property Menu: TCustomActionMenuBar;
```

C++

```
__property TCustomActionMenuBar Menu;
```

Description

This is Menu, a member of class TCustomMenuButton.

5.1.12 TCustomMenuExpandBtn Class

Class Hierarchy

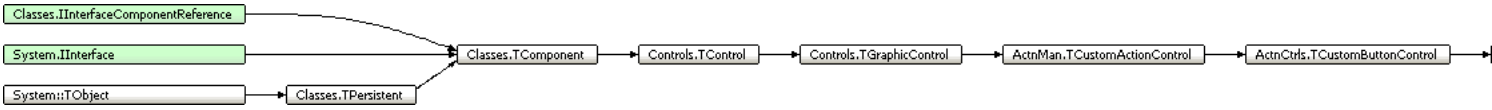

Pascal

```
TCustomMenuExpandBtn = class(TCustomUtilityButton);
```

C++

```
class TCustomMenuExpandBtn : public TCustomUtilityButton;
```

File

ActnMenus

Description

This is class ActnMenus.TCustomMenuExpandBtn.

5.1.12.1 TCustomMenuExpandBtn Members

The following tables list the members exposed by TCustomMenuExpandBtn.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFlat (see page 178)	This is FFlat, a member of class TCustomButtonControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FState (see page 178)	This is FState, a member of class TCustomButtonControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	BeginAutoDrag (see page 517)	Begin Automatic Drag.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CalcLayout (see page 518)	This is CalcLayout, a member of class TCustomActionControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
	CMDesignHitTest (see page 519)	This is CMDesignHitTest, a member of class TCustomActionControl.
	CMDialogChar (see page 180)	This is CMDialogChar, a member of class TCustomButtonControl.

	CMEnabledChanged (see page 180)	This is CMEnabledChanged, a member of class TCustomButtonControl.
	CMFontChanged (see page 519)	This is CMFontChanged, a member of class TCustomActionControl.
	CMItemDropPoint (see page 519)	This is CMItemDropPoint, a member of class TCustomActionControl.
	CMItemSelected (see page 519)	This is CMItemSelected, a member of class TCustomActionControl.
	CMMouseEnter (see page 180)	This is CMMouseEnter, a member of class TCustomButtonControl.
	CMMouseEnter (see page 238)	This is CMMouseEnter, a member of class TCustomUtilityButton.
	CMMouseLeave (see page 180)	This is CMMouseLeave, a member of class TCustomButtonControl.
	CMMouseLeave (see page 239)	This is CMMouseLeave, a member of class TCustomUtilityButton.
	CMTextChanged (see page 180)	This is CMTextChanged, a member of class TCustomButtonControl.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DesignWndProc (see page 519)	This is DesignWndProc, a member of class TCustomActionControl.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
V	DoDragDrop (see page 520)	This is DoDragDrop, a member of class TCustomActionControl.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
V	DoStartDrag (see page 520)	OnStartDrag event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 815)	OnDragOver event dispatcher.
V	DrawArrows (see page 239)	This is DrawArrows, a member of class TCustomUtilityButton.
V	DrawBackground (see page 815)	Draws a background image on the control.
V	DrawDesignFocus (see page 521)	This is DrawDesignFocus, a member of class TCustomActionControl.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawDragDropPoint (see page 522)	This is DrawDragDropPoint, a member of class TCustomActionControl.
V	DrawFrame (see page 816)	This is DrawFrame, a member of class TCustomMenuExpandBtn.
V	DrawGlyph (see page 522)	Draws a symbol on the control.
V	DrawLargeGlyph (see page 181)	This is DrawLargeGlyph, a member of class TCustomButtonControl.
V	DrawLargeGlyph (see page 522)	This is DrawLargeGlyph, a member of class TCustomActionControl.
V	DrawSeparator (see page 522)	This is DrawSeparator, a member of class TCustomActionControl.
V	DrawShadowedText (see page 523)	This is DrawShadowedText, a member of class TCustomActionControl.
V	DrawText (see page 523)	Draws text on the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	GetAction (see page 523)	This is GetAction, a member of class TCustomActionControl.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.

	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetImageSize (see page 523)	This is GetImageSize, a member of class TCustomActionControl.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSeparator (see page 524)	This is GetSeparator, a member of class TCustomActionControl.
	GetShowCaption (see page 524)	This is GetShowCaption, a member of class TCustomActionControl.
	GetShowShortCut (see page 181)	This is GetShowShortCut, a member of class TCustomButtonControl.
	GetShowShortCut (see page 524)	This is GetShowShortCut, a member of class TCustomActionControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasGlyph (see page 524)	This is HasGlyph, a member of class TCustomActionControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsChecked (see page 525)	This is IsChecked, a member of class TCustomActionControl.
	IsGrouped (see page 525)	This is IsGrouped, a member of class TCustomActionControl.
	Loaded (see page 182)	Initializes the control after it is loaded from a stream.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 182)	OnMouseDown event dispatcher.
	MouseDown (see page 239)	OnMouseDown event dispatcher.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 183)	Respond to mouse moving over control area..
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 183)	OnMouseUp event dispatcher.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	OnDelay (see page 240)	This is OnDelay, a member of class TCustomUtilityButton.
	Paint (see page 184)	Render control surface.
	Paint (see page 240)	Render control surface.
	Paint (see page 526)	Render control surface.
	Paint (see page 6052)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionChanged (see page 526)	This is PositionChanged, a member of class TCustomActionControl.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetUsageData (see page 526)	This is ResetUsageData, a member of class TCustomActionControl.
	Resize (see page 5780)	Respond to control resize.

	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
V	SetActionClient (see page 184)	This is SetActionClient, a member of class TCustomButtonControl.
V	SetActionClient (see page 526)	This is SetActionClient, a member of class TCustomActionControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropPoint (see page 527)	This is SetDropPoint, a member of class TCustomActionControl.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
V	SetGlyphLayout (see page 185)	This is SetGlyphLayout, a member of class TCustomButtonControl.
V	SetGlyphLayout (see page 527)	This is SetGlyphLayout, a member of class TCustomActionControl.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetSelected (see page 185)	This is SetSelected, a member of class TCustomButtonControl.
V	SetSelected (see page 527)	This is SetSelected, a member of class TCustomActionControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateSelection (see page 527)	This is UpdateSelection, a member of class TCustomActionControl.
	UpdateTextBounds (see page 527)	This is UpdateTextBounds, a member of class TCustomActionControl.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMLButtonDblClk (see page 185)	This is WMLButtonDblClk, a member of class TCustomButtonControl.
	WMSetText (see page 528)	This is WMSetText, a member of class TCustomActionControl.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.

	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GlyphPos (see page 532)	This is GlyphPos, a member of class TCustomActionControl.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	Menu (see page 816)	This is Menu, a member of class TCustomMenuExpandBtn.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	MouseInControl (see page 189)	This is MouseInControl, a member of class TCustomButtonControl.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnClick (see page 243)	Occurs when the user clicks the control.
	OnClick (see page 535)	Occurs when the user clicks the control.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
V	CalcBounds (see page 179)	Calculates the size of the control.
V	CalcBounds (see page 518)	Calculates the size of the control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Click (see page 179)	Executes the control's function.
V	Click (see page 238)	Executes the control's function.
V	Click (see page 518)	Executes the control's function in response to a mouse click.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 814)	Creates an instance of TCustomButtonControl.

	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 237)	Destroys an instance of TCustomActionControl.
	Destroy (see page 516)	Destroys an instance of TCustomActionControl (see page 507).
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 520)	Performs actions and generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 524)	Calls the action client's Update method if the control is associated with an action client.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	Keyed (see page 525)	Executes the control's function in response to a keyboard action.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 184)	Sets the Left, Top, Width, and Height properties all at once.
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	ActionBar (see page 528)	The ActionBand object which contains the custom action control
	ActionClient (see page 529)	Refers to the control's action client
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	AllowAllUp (see page 186)	Indicates whether all the buttons in a given group can be up at the same time.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	ArrowSize (see page 240)	This is ArrowSize, a member of class TCustomUtilityButton.
	ArrowType (see page 241)	This is ArrowType, a member of class TCustomUtilityButton.
	AutoScroll (see page 241)	This is AutoScroll, a member of class TCustomUtilityButton.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Caption (see page 186)	Specifies a text string that identifies the control to the user.

	Caption (see page 529)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 241)	Specifies the background color of the control.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Direction (see page 242)	This is Direction, a member of class TCustomUtilityButton.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Down (see page 189)	Tells whether the button is down (selected) or up (unselected).
	DropPoint (see page 532)	This is DropPoint, a member of class TCustomActionControl.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
	Flat (see page 189)	Dictates whether the button should have a 2D look instead of the usual 3D look.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GlyphLayout (see page 532)	Describes where the image should be on the control
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Margins (see page 532)	Describes the boundaries of the control for drawing purposes.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	RepeatRate (see page 242)	This is RepeatRate, a member of class TCustomUtilityButton.
	Selected (see page 533)	Indicates whether the control is currently selected.
R	Separator (see page 533)	Indicates whether the control is a separator.
	ShowCaption (see page 189)	Indicates whether the control's caption should be shown.
R	ShowCaption (see page 533)	Indicates whether the control's caption should be shown.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
	ShowShortCut (see page 190)	Indicates whether the control's short cut should be shown.
R	ShowShortCut (see page 533)	Indicates whether the control's short cut should be shown.
	SmallIcon (see page 534)	Indicates whether the control's small icon should be shown.
	Spacing (see page 534)	Controls (see page 5711) the separation between the control's icon and caption.
	State (see page 190)	Describes the current state of the button.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TextBounds (see page 534)	A bounding rectangle for the control caption.

	TrackButton (see page 190)	Specifies which mouse button activates the button control.
	Transparent (see page 534)	Shows whether the control is transparent or not.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

5.1.12.2 TCustomMenuExpandBtn.Create Constructor

Creates an instance of TCustomButtonControl.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomMenuExpandBtn(TComponent * AOwner);
```

Description

Calling Create constructs and initializes an instance of TCustomButtonControl. However, you should never attempt to instantiate a TCustomButtonControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component that is responsible for freeing the button control. It becomes the value of the Owner property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner (see page 2807)

Create (see page 5731)

Destroy

5.1.12.3 TCustomMenuExpandBtn Methods

The methods of the TCustomMenuExpandBtn class are listed here.

Protected Methods

	Name	Description
	DragOver (see page 815)	OnDragOver event dispatcher.
	DrawBackground (see page 815)	Draws a background image on the control.
	DrawFrame (see page 816)	This is DrawFrame, a member of class TCustomMenuExpandBtn.

5.1.12.3.1 TCustomMenuExpandBtn.DragOver Method

OnDragOver event dispatcher.

Pascal

```
procedure DragOver(Source: TObject; X: Integer; Y: Integer; State: TDragState; var Accept:
Boolean); override;
```

C++

```
virtual __fastcall DragOver(TObject * Source, int X, int Y, TDragState State, Boolean Accept);
```

Description

Override DragOver to add additional code that executes before the OnDragOver event handler is called.

DragOver sets the Accept parameter to true to indicate that the user can drop the dragged object on the control. It sets Accept to false to indicate that the user cannot drop the dragged object on the control.

The Source parameter is the object being dragged. The State parameter indicates how the dragged object is moving in relation to the control. X and Y indicate the current position of the mouse.

See Also

DragDrop (see page 5754)

OnDragOver (see page 5857)

5.1.12.3.2 TCustomMenuExpandBtn.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph (see page 522)

DrawText (see page 523)

5.1.12.3.3 TCustomMenuExpandBtn.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TCustomMenuExpandBtn.

5.1.12.4 TCustomMenuExpandBtn Properties

The properties of the TCustomMenuExpandBtn class are listed here.

Protected Properties

	Name	Description
	Menu (see page 816)	This is Menu, a member of class TCustomMenuExpandBtn.

5.1.12.4.1 TCustomMenuExpandBtn.Menu Property

Pascal

```
property Menu: TCustomActionMenuBar;
```

C++

```
__property TCustomActionMenuBar Menu;
```

Description

This is Menu, a member of class TCustomMenuExpandBtn.

5.1.13 TCustomMenuItem Class

TCustomMenuItem is the base class for the ActionBand custom control TStandardMenuItem.

Class Hierarchy

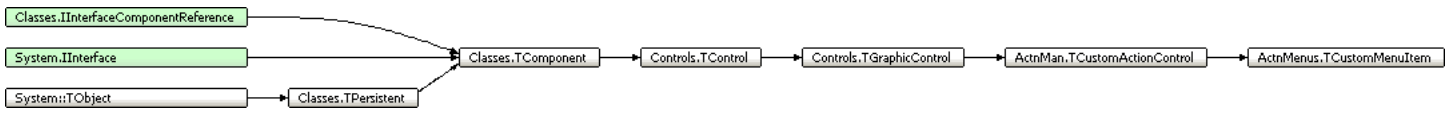

Pascal

```
TCustomMenuItem = class(TCustomActionControl);
```

C++

```
class TCustomMenuItem : public TCustomActionControl;
```

File

ActnMenus

Description

TCustomMenuItem is a base class for TStandardMenuItem, the standard ActionBand menu item class.

Do not create instances of TCustomMenuItem. If you would like to replace the standard menu item, use the class TCustomMenuItem as a parent class for your replacement class. To make the change for your application, set the value of MenuItemControlClass (see page 847) (a global variable) to your own subclass of TCustomMenuItem instead of TStandardMenuItem.

See Also

TStandardMenuItem (see page 20465)

5.1.13.1 TCustomMenuItem Members

The following tables list the members exposed by TCustomMenuItem.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FNoPrefix (see page 826)	This is FNoPrefix, a member of class TCustomMenuItem.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	BeginAutoDrag (see page 517)	Begin Automatic Drag.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CalcLayout (see page 827)	This is CalcLayout, a member of class TCustomMenuItem.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	CMDesignHitTest (see page 519)	This is CMDesignHitTest, a member of class TCustomActionControl.
	CMFontChanged (see page 519)	This is CMFontChanged, a member of class TCustomActionControl.
	CMItemDropPoint (see page 519)	This is CMItemDropPoint, a member of class TCustomActionControl.
	CMItemSelected (see page 519)	This is CMItemSelected, a member of class TCustomActionControl.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DesignWndProc (see page 519)	This is DesignWndProc, a member of class TCustomActionControl.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
V	DoDragDrop (see page 828)	This is DoDragDrop, a member of class TCustomMenuItem.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
V	DoStartDrag (see page 520)	OnStartDrag event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 828)	OnDragOver event dispatcher.

	DrawBackground (see page 521)	Draws a background image on the control.
	DrawDesignFocus (see page 521)	This is DrawDesignFocus, a member of class TCustomActionControl.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawDragDropPoint (see page 522)	This is DrawDragDropPoint, a member of class TCustomActionControl.
	DrawEdge (see page 829)	This is DrawEdge, a member of class TCustomMenuItem.
	DrawGlyph (see page 522)	Draws a symbol on the control.
	DrawLargeGlyph (see page 522)	This is DrawLargeGlyph, a member of class TCustomActionControl.
	DrawSeparator (see page 522)	This is DrawSeparator, a member of class TCustomActionControl.
	DrawShadowedText (see page 829)	This is DrawShadowedText, a member of class TCustomMenuItem.
	DrawSubMenuGlyph (see page 829)	This is DrawSubMenuGlyph, a member of class TCustomMenuItem.
	DrawText (see page 829)	This is DrawText, a member of class TCustomMenuItem.
	DrawUnusedEdges (see page 830)	This is DrawUnusedEdges, a member of class TCustomMenuItem.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	GetAction (see page 523)	This is GetAction, a member of class TCustomActionControl.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetImageSize (see page 523)	This is GetImageSize, a member of class TCustomActionControl.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSeparator (see page 524)	This is GetSeparator, a member of class TCustomActionControl.
	GetShowCaption (see page 524)	This is GetShowCaption, a member of class TCustomActionControl.
	GetShowShortCut (see page 524)	This is GetShowShortCut, a member of class TCustomActionControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasGlyph (see page 524)	This is HasGlyph, a member of class TCustomActionControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsChecked (see page 525)	This is IsChecked, a member of class TCustomActionControl.
	IsGrouped (see page 525)	This is IsGrouped, a member of class TCustomActionControl.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 830)	OnMouseUp event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.

	Paint (see page 831)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionChanged (see page 831)	This is PositionChanged, a member of class TCustomMenuItem.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetUsageData (see page 526)	This is ResetUsageData, a member of class TCustomActionControl.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActionClient (see page 526)	This is SetActionClient, a member of class TCustomActionControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropPoint (see page 527)	This is SetDropPoint, a member of class TCustomActionControl.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetGlyphLayout (see page 527)	This is SetGlyphLayout, a member of class TCustomActionControl.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelected (see page 831)	This is SetSelected, a member of class TCustomMenuItem.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateSelection (see page 527)	This is UpdateSelection, a member of class TCustomActionControl.
	UpdateTextBounds (see page 527)	This is UpdateTextBounds, a member of class TCustomActionControl.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMSetText (see page 528)	This is WMSetText, a member of class TCustomActionControl.
	WndProc (see page 5791)	Provides specific message responses for the control.

V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GlyphPos (see page 532)	This is GlyphPos, a member of class TCustomActionControl.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	Menu (see page 832)	This is Menu, a member of class TCustomMenuItem.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	MouseSelected (see page 832)	This is MouseSelected, a member of class TCustomMenuItem.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ShortCutBounds (see page 832)	This is ShortCutBounds, a member of class TCustomMenuItem.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnClick (see page 535)	Occurs when the user clicks the control.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
V	CalcBounds (see page 827)	Calculates the size of the menu item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Click (see page 827)	Executes the control's function.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 825)	Creates an instance of TCustomMenuItem (see page 816).
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 825)	Destroys an instance of TCustomMenuItem (see page 816).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
V	DragDrop (see page 828)	Performs actions and generates an OnDragDrop event.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 524)	Calls the action client's Update method if the control is associated with an action client.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.

	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
V	Keyed (see page 830)	Executes the menu item's function in response to a user's keyboard command.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	ActionBar (see page 528)	The ActionBand object which contains the custom action control
	ActionClient (see page 529)	Refers to the control's action client
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Caption (see page 529)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.

R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DropPoint (see page 532)	This is DropPoint, a member of class TCustomActionControl.
	Edges (see page 832)	Dictates which edges are drawn when the menu item is drawn.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GlyphLayout (see page 532)	Describes where the image should be on the control
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Margins (see page 532)	Describes the boundaries of the control for drawing purposes.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	Selected (see page 533)	Indicates whether the control is currently selected.
R	Separator (see page 533)	Indicates whether the control is a separator.
R	ShowCaption (see page 533)	Indicates whether the control's caption should be shown.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	ShowShortCut (see page 533)	Indicates whether the control's short cut should be shown.
	SmallIcon (see page 534)	Indicates whether the control's small icon should be shown.
	Spacing (see page 534)	Controls (see page 5711) the separation between the control's icon and caption.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TextBounds (see page 534)	A bounding rectangle for the control caption.
	Transparent (see page 534)	Shows whether the control is transparent or not.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic

	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

5.1.13.2 TCustomMenuItem.Create Constructor

Creates an instance of TCustomMenuItem (see page 816).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomMenuItem(TComponent * AOwner);
```

Description

Calling Create constructs and initializes an instance of TCustomMenuItem (see page 816). However, you should never attempt to instantiate a TCustomMenuItem (see page 816). This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes the TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create (see page 5731)

Destroy (see page 825)

5.1.13.3 TCustomMenuItem.Destroy Destructor

Destroys an instance of TCustomMenuItem (see page 816).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomMenuItem();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the graphic control reference is not nil, and only then calls Destroy.

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Frees the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create (see page 825) method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

Create (see page 825)

Free (see page 22300)

5.1.13.4 TCustomMenuItem Fields

The fields of the TCustomMenuItem class are listed here.

Protected Fields

	Name	Description
	FNoPrefix (see page 826)	This is FNoPrefix, a member of class TCustomMenuItem.

5.1.13.4.1 TCustomMenuItem.FNoPrefix Field

Pascal

```
FNoPrefix: string;
```

C++

```
AnsiString FNoPrefix;
```

Description

This is FNoPrefix, a member of class TCustomMenuItem.

5.1.13.5 TCustomMenuItem Methods

The methods of the TCustomMenuItem class are listed here.

Protected Methods

	Name	Description
	CalcLayout (see page 827)	This is CalcLayout, a member of class TCustomMenuItem.
	DoDragDrop (see page 828)	This is DoDragDrop, a member of class TCustomMenuItem.
	DragOver (see page 828)	OnDragOver event dispatcher.
	DrawEdge (see page 829)	This is DrawEdge, a member of class TCustomMenuItem.
	DrawShadowedText (see page 829)	This is DrawShadowedText, a member of class TCustomMenuItem.
	DrawSubMenuGlyph (see page 829)	This is DrawSubMenuGlyph, a member of class TCustomMenuItem.
	DrawText (see page 829)	This is DrawText, a member of class TCustomMenuItem.
	DrawUnusedEdges (see page 830)	This is DrawUnusedEdges, a member of class TCustomMenuItem.
	MouseUp (see page 830)	OnMouseUp event dispatcher.

	Paint (see page 831)	Render control surface.
	PositionChanged (see page 831)	This is PositionChanged, a member of class TCustomMenuItem.
	SetSelected (see page 831)	This is SetSelected, a member of class TCustomMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 827)	Calculates the size of the menu item.
	Click (see page 827)	Executes the control's function.
	DragDrop (see page 828)	Performs actions and generates an OnDragDrop event.
	Keyed (see page 830)	Executes the menu item's function in response to a user's keyboard command.

5.1.13.5.1 TCustomMenuItem.CalcBounds Method

Calculates the size of the menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as whether or not the menu item is a separator or not. CalcBounds computes the boundaries of the menu item considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds (see page 5783)

5.1.13.5.2 TCustomMenuItem.CalcLayout Method

Pascal

```
procedure CalcLayout; override;
```

C++

```
virtual __fastcall CalcLayout();
```

Description

This is CalcLayout, a member of class TCustomMenuItem.

5.1.13.5.3 TCustomMenuItem.Click Method

Executes the control's function.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

This override of the method inherited from TCustomActionControl does little except bring the form containing the item into focus, and prevent the inherited method from being called. The reason for this lack of action is that at runtime, no action should be

taken until the primary mouse button is pressed and released, which makes the OnClick event handler responsible for taking action. Click takes place only when the mouse button is first pressed, prior to release.

See Also

OnClick (see page 5851)

5.1.13.5.4 TCustomMenuItem.DoDragDrop Method

Pascal

```
procedure DoDragDrop(DragObject: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DoDragDrop(TObject * DragObject, int X, int Y);
```

Description

This is DoDragDrop, a member of class TCustomMenuItem.

5.1.13.5.5 TCustomMenuItem.DragDrop Method

Performs actions and generates an OnDragDrop event.

Pascal

```
procedure DragDrop(Source: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DragDrop(TObject * Source, int X, int Y);
```

Description

This override of the method inherited from TCustomActionControl calls the inherited routine and performs other actions to ensure that the focus is correct.

See Also

DragDrop (see page 520)

DragDrop (see page 5754)

5.1.13.5.6 TCustomMenuItem.DragOver Method

OnDragOver event dispatcher.

Pascal

```
procedure DragOver(Source: TObject; X: Integer; Y: Integer; State: TDragState; var Accept:
Boolean); override;
```

C++

```
virtual __fastcall DragOver(TObject * Source, int X, int Y, TDragState State, Boolean Accept);
```

Description

Override DragOver to add additional code that executes before the OnDragOver event handler is called.

DragOver sets the Accept parameter to true to indicate that the user can drop the dragged object on the control. It sets Accept to false to indicate that the user cannot drop the dragged object on the control.

The Source parameter is the object being dragged. The State parameter indicates how the dragged object is moving in relation to the control. X and Y indicate the current position of the mouse.

See Also

DragDrop (see page 5754)

OnDragOver (see page 5857)

5.1.13.5.7 TCustomMenuItem.DrawEdge Method

Pascal

```
procedure DrawEdge(Edges: TMenuEdges); virtual;
```

C++

```
virtual __fastcall DrawEdge(TMenuEdges Edges);
```

Description

This is DrawEdge, a member of class TCustomMenuItem.

5.1.13.5.8 TCustomMenuItem.DrawShadowedText Method

Pascal

```
procedure DrawShadowedText(Rect: TRect; Flags: Cardinal; Text: string; TextColor: TColor;
ShadowColor: TColor); override;
```

C++

```
virtual __fastcall DrawShadowedText(TRect Rect, unsigned Flags, AnsiString Text, TColor
TextColor, TColor ShadowColor);
```

Description

This is DrawShadowedText, a member of class TCustomMenuItem.

5.1.13.5.9 TCustomMenuItem.DrawSubMenuGlyph Method

Pascal

```
procedure DrawSubMenuGlyph; virtual;
```

C++

```
virtual __fastcall DrawSubMenuGlyph();
```

Description

This is DrawSubMenuGlyph, a member of class TCustomMenuItem.

5.1.13.5.10 TCustomMenuItem.DrawText Method

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: String); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, AnsiString Text);
```

Description

This is DrawText, a member of class TCustomMenuItem.

5.1.13.5.11 TCustomMenuItem.DrawUnusedEdges Method

Pascal

```
procedure DrawUnusedEdges; virtual;
```

C++

```
virtual __fastcall DrawUnusedEdges();
```

Description

This is DrawUnusedEdges, a member of class TCustomMenuItem.

5.1.13.5.12 TCustomMenuItem.Keyed Method

Executes the menu item's function in response to a user's keyboard command.

Pascal

```
procedure Keyed; override;
```

C++

```
virtual __fastcall Keyed();
```

Description

This method is largely identical to the inherited method in TCustomActionControl. Additional actions are taken specifically for TCustomMenuItem (see page 816) objects.

See Also

Keyed (see page 525)

5.1.13.5.13 TCustomMenuItem.MouseUp Method

OnMouseUp event dispatcher.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user releases a previously pressed mouse button while the cursor's hotspot is over the control.

A control calls MouseUp in response to any of the Windows mouse-up messages (WM_LBUTTONUP, WM_MBUTTONUP, WM_RBUTTONUP), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

See Also

MouseDown (see page 5773)

MouseMove (see page 5773)

OnMouseUp (see page 5867)

5.1.13.5.14 TCustomMenuItem.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas (see page 6053)

5.1.13.5.15 TCustomMenuItem.PositionChanged Method

Pascal

```
procedure PositionChanged; override;
```

C++

```
virtual __fastcall PositionChanged();
```

Description

This is PositionChanged, a member of class TCustomMenuItem.

5.1.13.5.16 TCustomMenuItem.SetSelected Method

Pascal

```
procedure SetSelected(Value: Boolean); override;
```

C++

```
virtual __fastcall SetSelected(Boolean Value);
```

Description

This is SetSelected, a member of class TCustomMenuItem.

5.1.13.6 TCustomMenuItem Properties

The properties of the TCustomMenuItem class are listed here.

Protected Properties

	Name	Description
R	Menu (see page 832)	This is Menu, a member of class TCustomMenuItem.
R	MouseSelected (see page 832)	This is MouseSelected, a member of class TCustomMenuItem.
	ShortCutBounds (see page 832)	This is ShortCutBounds, a member of class TCustomMenuItem.

Public Properties

	Name	Description
	Edges (see page 832)	Dictates which edges are drawn when the menu item is drawn.

5.1.13.6.1 TCustomMenuItem.Edges Property

Dictates which edges are drawn when the menu item is drawn.

Pascal

```
property Edges: TMenuEdges;
```

C++

```
__property TMenuEdges Edges;
```

Description

Use Edges to dictate which edges of the menu item should be drawn when the menu item is drawn. The values ebLeft, ebTop, ebRight, and ebBottom indicate that the left, top, right, and bottom edges (respectively) should be drawn.

5.1.13.6.2 TCustomMenuItem.Menu Property

Pascal

```
property Menu: TCustomActionMenuBar;
```

C++

```
__property TCustomActionMenuBar Menu;
```

Description

This is Menu, a member of class TCustomMenuItem.

5.1.13.6.3 TCustomMenuItem.MouseSelected Property

Pascal

```
property MouseSelected: Boolean;
```

C++

```
__property Boolean MouseSelected;
```

Description

This is MouseSelected, a member of class TCustomMenuItem.

5.1.13.6.4 TCustomMenuItem.ShortCutBounds Property

Pascal

```
property ShortCutBounds: TRect;
```

C++

```
__property TRect ShortCutBounds;
```

Description

This is ShortCutBounds, a member of class TCustomMenuItem.

5.1.14 TMenuStack Class

Class Hierarchy

System::TObject → Contnrs.TOrderedList → Contnrs.TStack → ActnMenus.TMenuStack

Pascal

```
TMenuStack = class(TStack);
```

C++

```
class TMenuStack : public TStack;
```

File

ActnMenus

Description

This is class ActnMenus.TMenuStack.

5.1.14.1 TMenuStack Members

The following tables list the members exposed by TMenuStack.

Protected Methods

	Name	Description
V	PeekItem (see page 5697)	Returns a pointer to the next item in the list.
V	PopItem (see page 5697)	Removes and returns a pointer to the next item in the list.
A	PushItem (see page 5698)	Adds an item to the list.
V	PushItem (see page 5703)	Adds an item to the list.

Protected Properties

	Name	Description
R	Bars (see page 836)	This is Bars, a member of class TMenuStack.
R	List (see page 5699)	Identifies the internal list used to store and manage the items in the ordered list.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AtLeast (see page 5696)	Checks the size of the list.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Count (see page 5696)	Returns the size of the list.
	Create (see page 834)	This is Create, a member of class TMenuStack.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5695)	Destroys the TOrderedList (see page 5693) instance.

	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Peek (see page 834)	Returns a pointer to the next item in the list.
	Pop (see page 835)	Removes and returns the next item in the list.
	Push (see page 835)	This is Push, a member of class TMenuStack.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

5.1.14.2 TMenuStack.Create Constructor

Pascal

```
constructor Create(AMenu: TCustomActionMenuBar);
```

C++

```
__fastcall TMenuStack(TCustomActionMenuBar AMenu);
```

Description

This is Create, a member of class TMenuStack.

5.1.14.3 TMenuStack Methods

The methods of the TMenuStack class are listed here.

Public Methods

	Name	Description
	Peek (see page 834)	Returns a pointer to the next item in the list.
	Pop (see page 835)	Removes and returns the next item in the list.
	Push (see page 835)	This is Push, a member of class TMenuStack.

5.1.14.3.1 TMenuStack.Peek Method

Returns a pointer to the next item in the list.

Pascal

```
function Peek: TCustomActionMenuBar;
```

C++

```
__fastcall TCustomActionMenuBar Peek();
```

Description

Call Peek to access the list without removing an item. Peek returns a pointer to the next item in the list.

Each TOrderedList descendant implements Peek to return a specific item from the list. For example, TQueue returns the item that was added first, while TStack returns the item that was added last.

To remove an item from the list, call Pop.

See Also

Push (see page 5698)

Pop (see page 5697)

5.1.14.3.2 TMenuStack.Pop Method

Removes and returns the next item in the list.

Pascal

```
function Pop: TCustomActionMenuBar;
```

C++

```
__fastcall TCustomActionMenuBar Pop();
```

Description

Call Pop to retrieve an item from the list. Pop removes the next item from the list, then returns a pointer to the deleted item.

Each TOrderedList descendant implements Pop to remove and return a specific item from the list. For example, TQueue returns the item that was added first, while TStack returns the item that was added last.

To access the list without removing an item, call Peek.

See Also

Push (see page 5698)

Peek (see page 5696)

5.1.14.3.3 TMenuStack.Push Method

Pascal

```
procedure Push(Container: TCustomActionMenuBar);
```

C++

```
__fastcall Push(TCustomActionMenuBar Container);
```

Description

This is Push, a member of class TMenuStack.

5.1.14.4 TMenuStack Properties

The properties of the TMenuStack class are listed here.

Protected Properties

	Name	Description
	Bars (see page 836)	This is Bars, a member of class TMenuStack.

5.1.14.4.1 TMenuStack.Bars Property

Pascal

```
property Bars [const Index: Integer]: TCustomActionMenuBar;
```

C++

```
__property TCustomActionMenuBar Bars[int const Index];
```

Description

This is Bars, a member of class TMenuStack.

5.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	RegisterActnBarStyle (see page 837)	Registers an action band style so that it is available for use from TActionManager's Style property within the IDE.
	UnRegisterActnBarStyle (see page 837)	Unregisters an action band style from the IDE.

5.2.1 ActnMenus.RegisterActnBarStyle Function

Registers an action band style so that it is available for use from TActionManager's Style property within the IDE.

Pascal

```
procedure RegisterActnBarStyle(AStyle: TActionBarStyleEx);
```

C++

```
RegisterActnBarStyle(TActionBarStyleEx AStyle);
```

File

ActnMenus

Description

RegisterActnBarStyle registers a style so that is is available at design-time through TActionManager's Style property in the object inspector. After creating a descendant of TActionBarStyleEx (see page 583), call RegisterActnBarStyle to use the style in the IDE.

See Also

UnRegisterActnBarStyle (see page 837)

RegisterActions

5.2.2 ActnMenus.UnRegisterActnBarStyle Function

Unregisters an action band style from the IDE.

Pascal

```
procedure UnRegisterActnBarStyle(AStyle: TActionBarStyleEx);
```

C++

```
UnRegisterActnBarStyle(TActionBarStyleEx AStyle);
```

File

ActnMenus

Description

UnRegisterActnBarStyle unregisters an action band style from the IDE thus removing it from the list displayed in the Object Inspector for TActionManager's Style property.

See Also

RegisterActnBarStyle (see page 837)

UnRegisterActions

5.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAnimationStyle (see page 839)	This is record ActnMenus.TAnimationStyle.
	TMDIButtonStyle (see page 839)	This is record ActnMenus.TMDIButtonStyle.

5.3.1 ActnMenus.TAnimationStyle Enumeration

Pascal

```
TAnimationStyle = (
  asNone,
  asDefault,
  asUnFold,
  asSlide,
  asFade
);
```

C++

```
enum TAnimationStyle {
  asNone,
  asDefault,
  asUnFold,
  asSlide,
  asFade
};
```

File

ActnMenus

Description

This is record ActnMenus.TAnimationStyle.

5.3.2 ActnMenus.TMDIButtonStyle Enumeration

Pascal

```
TMDIButtonStyle = (
  mbMinimize,
  mbRestore,
  mbClose
);
```

C++

```
enum TMDIButtonStyle {
  mbMinimize,
  mbRestore,
```

```
    mbClose
  };
```

File

ActnMenus

Description

This is record ActnMenus.TMDIButtonStyle.

5.4 Types

The following table lists types in this documentation.

Types

Name	Description
TAddRemoveItemClass (see page 841)	This is type ActnMenus.TAddRemoveItemClass.
TCustomAddRemoveItemClass (see page 841)	This is type ActnMenus.TCustomAddRemoveItemClass.
TCustomizeActionToolBarClass (see page 842)	This is type ActnMenus.TCustomizeActionToolBarClass.
TCustomMenuExpandBtnClass (see page 842)	This is type ActnMenus.TCustomMenuExpandBtnClass.
TCustomPopupClass (see page 842)	This is type ActnMenus.TCustomPopupClass.
TExitMenuEvent (see page 843)	This is type ActnMenus.TExitMenuEvent.
TGetPopupClassEvent (see page 843)	This is type ActnMenus.TGetPopupClassEvent.
TMenuButtonControlClass (see page 843)	This is type ActnMenus.TMenuButtonControlClass.
TMenuEdges (see page 844)	This is type ActnMenus.TMenuEdges.
TMenuItemControlClass (see page 844)	This is type ActnMenus.TMenuItemControlClass.
TMenuPopupEvent (see page 844)	This is type ActnMenus.TMenuPopupEvent.
TUpdateActnMenusProc (see page 845)	This is type ActnMenus.TUpdateActnMenusProc.

5.4.1 ActnMenus.TAddRemoveItemClass Type

Pascal

```
TAddRemoveItemClass = class of TCustomAddRemoveItem;
```

C++

```
TCustomAddRemoveItem TAddRemoveItemClass;
```

File

ActnMenus

Description

This is type ActnMenus.TAddRemoveItemClass.

5.4.2 ActnMenus.TCustomAddRemoveItemClass Type

Pascal

```
TCustomAddRemoveItemClass = class of TCustomAddRemoveItem;
```

C++

```
TCustomAddRemoveItem TCustomAddRemoveItemClass;
```

File

ActnMenus

Description

This is type ActnMenus.TCustomAddRemoveItemClass.

5.4.3 ActnMenus.TCustomizeActionToolBarClass Type

Pascal

```
TCustomizeActionToolBarClass = class of TCustomizeActionToolBar;
```

C++

```
TCustomizeActionToolBar TCustomizeActionToolBarClass;
```

File

ActnMenus

Description

This is type ActnMenus.TCustomizeActionToolBarClass.

5.4.4 ActnMenus.TCustomMenuExpandBtnClass Type

Pascal

```
TCustomMenuExpandBtnClass = class of TCustomMenuExpandBtn;
```

C++

```
TCustomMenuExpandBtn TCustomMenuExpandBtnClass;
```

File

ActnMenus

Description

This is type ActnMenus.TCustomMenuExpandBtnClass.

5.4.5 ActnMenus.TCustomPopupClass Type

Pascal

```
TCustomPopupClass = class of TCustomActionPopupMenu;
```

C++

```
TCustomActionPopupMenu TCustomPopupClass;
```

File

ActnMenus

Description

This is type ActnMenus.TCustomPopupClass.

5.4.6 ActnMenus.TExitMenuEvent Type

Pascal

```
TExitMenuEvent = procedure (Sender: TCustomActionMenuBar; Cancelled: Boolean) of object;
```

C++

```
(Sender: TCustomActionMenuBar; Cancelled: Boolean) ( TExitMenuEvent)();
```

File

ActnMenus

Description

This is type ActnMenus.TExitMenuEvent.

5.4.7 ActnMenus.TGetPopupClassEvent Type

Pascal

```
TGetPopupClassEvent = procedure (Sender: TObject; var PopupClass: TCustomPopupClass) of object;
```

C++

```
(Sender: TObject; var PopupClass: TCustomPopupClass) ( TGetPopupClassEvent)();
```

File

ActnMenus

Description

This is type ActnMenus.TGetPopupClassEvent.

5.4.8 ActnMenus.TMenuButtonControlClass Type

Pascal

```
TMenuButtonControlClass = class of TCustomMenuButton;
```

C++

```
TCustomMenuButton TMenuButtonControlClass;
```

File

ActnMenus

Description

This is type ActnMenus.TMenuButtonControlClass.

5.4.9 ActnMenus.TMenuEdges Type

Pascal

```
TMenuEdges = set of TEdgeBorder;
```

C++

```
TEdgeBorder TMenuEdges;
```

File

ActnMenus

Description

This is type ActnMenus.TMenuEdges.

5.4.10 ActnMenus.TMenuItemControlClass Type

Pascal

```
TMenuItemControlClass = class of TCustomMenuItem;
```

C++

```
TCustomMenuItem TMenuItemControlClass;
```

File

ActnMenus

Description

This is type ActnMenus.TMenuItemControlClass.

5.4.11 ActnMenus.TMenuPopupEvent Type

Pascal

```
TMenuPopupEvent = procedure (Sender: TObject; Item: TCustomActionControl) of object;
```

C++

```
(Sender: TObject; Item: TCustomActionControl) ( TMenuPopupEvent)();
```

File

ActnMenus

Description

This is type ActnMenus.TMenuPopupEvent.

5.4.12 ActnMenus.TUpdateActnMenusProc Type

Pascal

```
TUpdateActnMenusProc = procedure;
```

C++

```
( TUpdateActnMenusProc)();
```

File

ActnMenus

Description

This is type ActnMenus.TUpdateActnMenusProc.

5.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
MenuAddRemoveItemClass (see page 846)	This is variable ActnMenus.MenuAddRemoveItemClass.
MenuButtonControlClass (see page 846)	This is variable ActnMenus.MenuButtonControlClass.
MenuCustomizePopupClass (see page 847)	This is variable ActnMenus.MenuCustomizePopupClass.
MenuItemControlClass (see page 847)	This is variable ActnMenus.MenuItemControlClass.
MenuPopupClass (see page 847)	This is variable ActnMenus.MenuPopupClass.
UpdateActnMenusProc (see page 848)	This is variable ActnMenus.UpdateActnMenusProc.

5.5.1 ActnMenus.MenuAddRemoveItemClass Variable

Pascal

```
MenuAddRemoveItemClass: TAddRemoveItemClass deprecated;
```

C++

```
TAddRemoveItemClass deprecated MenuAddRemoveItemClass;
```

File

ActnMenus

Description

This is variable ActnMenus.MenuAddRemoveItemClass.

5.5.2 ActnMenus.MenuButtonControlClass Variable

Pascal

```
MenuButtonControlClass: TMenuButtonControlClass deprecated;
```

C++

```
TMenuButtonControlClass deprecated MenuButtonControlClass;
```

File

ActnMenus

Description

This is variable ActnMenus.MenuButtonControlClass.

5.5.3 ActnMenus.MenuCustomizePopupClass Variable

Pascal

```
MenuCustomizePopupClass: TCustomizeActionToolBarClass deprecated;
```

C++

```
TCustomizeActionToolBarClass deprecated MenuCustomizePopupClass;
```

File

ActnMenus

Description

This is variable ActnMenus.MenuCustomizePopupClass.

5.5.4 ActnMenus.MenuItemControlClass Variable

Pascal

```
MenuItemControlClass: TMenuItemControlClass deprecated;
```

C++

```
TMenuItemControlClass deprecated MenuItemControlClass;
```

File

ActnMenus

Description

This is variable ActnMenus.MenuItemControlClass.

5.5.5 ActnMenus.MenuPopupClass Variable

Pascal

```
MenuPopupClass: TCustomPopupClass deprecated;
```

C++

```
TCustomPopupClass deprecated MenuPopupClass;
```

File

ActnMenus

Description

This is variable ActnMenus.MenuPopupClass.

5.5.6 ActnMenus.UpdateActnMenusProc Variable

Pascal

```
UpdateActnMenusProc: TUpdateActnMenusProc;
```

C++

```
TUpdateActnMenusProc UpdateActnMenusProc;
```

File

ActnMenus

Description

This is variable ActnMenus.UpdateActnMenusProc.

6 ActnPopup Namespace

This is namespace ActnPopup.

Topics

Name	Description
Classes (see page 850)	The following table lists classes in this documentation.
Functions (see page 859)	The following table lists functions in this documentation.

Classes

	Name	Description
	TCustomActionPopupMenuEx (see page 850)	This is class ActnPopup.TCustomActionPopupMenuEx.
	TPopupActionBar (see page 852)	This is class ActnPopup.TPopupActionBar.

Functions

	Name	Description
	NewPopupMenu (see page 859)	Creates and initializes a popup menu.

6.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCustomActionPopupMenuEx (see page 850)	This is class ActnPopup.TCustomActionPopupMenuEx.
	TPopupActionBar (see page 852)	This is class ActnPopup.TPopupActionBar.

6.1.1 TCustomActionPopupMenuEx Class

Class Hierarchy

Pascal

```
TCustomActionPopupMenuEx = class(TXPStylePopupMenu);
```

C++

```
class TCustomActionPopupMenuEx : public TXPStylePopupMenu;
```

File

ActnPopup

Description

This is class ActnPopup.TCustomActionPopupMenuEx.

6.1.1.1 TCustomActionPopupMenuEx Members

The following tables list the members exposed by TCustomActionPopupMenuEx.

Protected Methods

	Name	Description
	DoPopup (see page 851)	This is DoPopup, a member of class TCustomActionPopupMenuEx.
	ExecAction (see page 851)	This is ExecAction, a member of class TCustomActionPopupMenuEx.
	GetPopupClass (see page 851)	This is GetPopupClass, a member of class TCustomActionPopupMenuEx.

Public Methods

	Name	Description
	LoadMenu (see page 851)	This is LoadMenu, a member of class TCustomActionPopupMenuEx.

6.1.1.2 TCustomActionPopupMenuEx Methods

The methods of the TCustomActionPopupMenuEx class are listed here.

Protected Methods

	Name	Description
	DoPopup (see page 851)	This is DoPopup, a member of class TCustomActionPopupMenuEx.
	ExecAction (see page 851)	This is ExecAction, a member of class TCustomActionPopupMenuEx.
	GetPopupClass (see page 851)	This is GetPopupClass, a member of class TCustomActionPopupMenuEx.

Public Methods

	Name	Description
	LoadMenu (see page 851)	This is LoadMenu, a member of class TCustomActionPopupMenuEx.

6.1.1.2.1 TCustomActionPopupMenuEx.DoPopup Method

Pascal

```
procedure DoPopup(Item: TCustomActionControl); override;
```

C++

```
virtual __fastcall DoPopup(TCustomActionControl Item);
```

Description

This is DoPopup, a member of class TCustomActionPopupMenuEx.

6.1.1.2.2 TCustomActionPopupMenuEx.ExecAction Method

Pascal

```
procedure ExecAction(Action: TContainedAction); override;
```

C++

```
virtual __fastcall ExecAction(TContainedAction Action);
```

Description

This is ExecAction, a member of class TCustomActionPopupMenuEx.

6.1.1.2.3 TCustomActionPopupMenuEx.GetPopupClass Method

Pascal

```
function GetPopupClass: TCustomPopupClass; override;
```

C++

```
virtual __fastcall TCustomPopupClass GetPopupClass();
```

Description

This is GetPopupClass, a member of class TCustomActionPopupMenuEx.

6.1.1.2.4 TCustomActionPopupMenuEx.LoadMenu Method

Pascal

```
procedure LoadMenu(Clients: TActionClients; AMenu: TMenuItem);
```

C++

```
__fastcall LoadMenu(TActionClients Clients, TMenuItem AMenu);
```

Description

This is LoadMenu, a member of class TCustomActionPopupMenuEx.

6.1.2 TPopupActionBar Class

Class Hierarchy

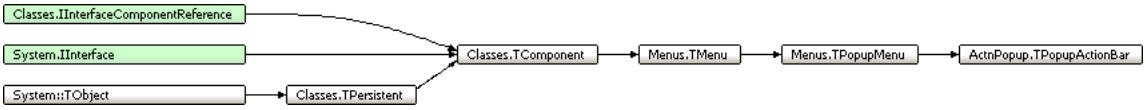

Pascal

```
TPopupActionBar = class(TPopupMenu);
```

C++

```
class TPopupActionBar : public TPopupMenu;
```

File

ActnPopup

Description

This is class ActnPopup.TPopupActionBar.

6.1.2.1 TPopupActionBar Members

The following tables list the members exposed by TPopupActionBar.

Protected Events

	Name	Description
	OnChange (see page 16090)	Occurs when the menu changes.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AdjustBiDiBehavior (see page 16073)	Applies the BiDiMode (see page 16083) property to the menu when the Windows control is created.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoBiDiModeChanged (see page 16074)	Updates the menu when the BiDiMode (see page 16083) property changes.
V	DoChange (see page 16075)	Generates an OnChange (see page 16090) event.
	DoGetMenuString (see page 16075)	Returns the caption for an item in the menu.

	DoPopup (see page 16174)	Generates an OnPopup (see page 16184) event.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 16076)	Executes a callback for child menu items.
	GetHandle (see page 16076)	Returns the value of the Handle (see page 16083) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsOwnerDraw (see page 16077)	Indicates whether menu items can contain more than simple captions.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 16078)	Calls the DoChange (see page 16075) method when the menu is first loaded into memory.
	MenuChanged (see page 16079)	Responds when the menu changes.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 16079)	Responds when components are added or deleted.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 16081)	Ensures that the menu item indexes reflect the order in which they are loaded and saved.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetPopupPoint (see page 16175)	This is SetPopupPoint, a member of class TPopupMenu.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateItems (see page 16081)	Causes all menu items to update themselves.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UseRightToLeftAlignment (see page 16175)	Specifies whether the control's alignment is in a right-to-left mode.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.

	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 856)	Creates an instance of TPopupMenu.
	CreateMenuItem (see page 16073)	This is CreateMenuItem, a member of class TMenu.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 16072)	Destroys the menu component.
	Destroy (see page 16173)	Destroys an instance of TPopupMenu (see page 16169).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DispatchCommand (see page 16074)	Calls the Click method of the menu item with the specified value as its Command property.
	DispatchPopup (see page 16074)	Dispatches a pop-up menu command.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindItem (see page 16076)	Locates and returns a menu item.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHelpContext (see page 16077)	Returns the help context ID associated with a menu item.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 16077)	Indicates whether menu items read from right to left.
	IsShortCut (see page 16078)	Dispatches a key message if it represents a shortcut key for an item in the menu.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentBiDiModeChanged (see page 16079)	This is the overview for the ParentBiDiModeChanged method overload.
V	Popup (see page 856)	Displays the pop-up menu onscreen.
	ProcessMenuChar (see page 16080)	Responds to keyboard input for the menu.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	AutoHotkeys (see page 16082)	Determines whether the accelerator keys for menu items can be reset automatically.
	AutoLineReduction (see page 16082)	Determines whether redundant separator bars are automatically removed from the menu.
	BiDiMode (see page 16083)	Determines the layout of the menu in Middle East locales.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Handle (see page 16083)	Provides access to the Windows menu handle for the menu.
	Images (see page 16083)	Lists the images that can appear beside individual menu items.
R	MenuActive (see page 857)	This is MenuActive, a member of class TPopupActionBar.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	OwnerDraw (see page 16088)	Specifies whether the menu items are drawn by the application.
	ParentBiDiMode (see page 16089)	Indicates whether the BiDiMode (see page 16083) property is inherited from the control that uses the menu.
	PopupComponent (see page 16181)	Indicates the component that last displayed the popup menu in response to a right mouse click.
R	PopupPoint (see page 16183)	Indicates the location where the pop-up menu appears.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	WindowHandle (see page 16089)	Provides access to the handle of the window that uses the menu.

Published Events

	Name	Description
	OnChange (see page 16184)	Occurs when the menu changes.
	OnGetControlClass (see page 858)	This is OnGetControlClass, a member of class TPopupActionBar.
	OnPopup (see page 16184)	Occurs just before the pop-up menu appears.

Published Properties

	Name	Description
	Alignment (see page 16176)	Determines where the pop-up menu appears when the user clicks the right mouse button.

	AutoHotkeys (see page 16176)	Determines whether the accelerator keys for menu items can be reset automatically.
	AutoLineReduction (see page 16177)	Determines whether redundant separator bars are automatically removed from the menu.
	AutoPopup (see page 16178)	Determines whether the pop-up menu appears automatically when the user clicks the right mouse button.
	BiDiMode (see page 16178)	Determines the layout of the menu in Middle East locales.
	HelpContext (see page 16179)	Specifies a context ID for the entire pop-up menu.
	Images (see page 16179)	Lists the images that can appear beside individual menu items.
R	Items (see page 16084)	Describes the elements of the menu.
	MenuAnimation (see page 16180)	Specifies how the menu appears when it is first displayed.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OwnerDraw (see page 16180)	Specifies whether the menu items are drawn by the application.
	ParentBiDiMode (see page 16180)	Indicates whether the BiDiMode property is inherited from the control that uses the menu.
	PopupMenu (see page 858)	This is PopupMenu, a member of class TPopupActionBar.
	Tag (see page 2808)	Stores an integer value as part of a component.
	TrackButton (see page 16183)	Specifies which mouse button activates the popup menu when it is associated with a toolbar button.

6.1.2.2 TPopupActionBar.Create Constructor

Creates an instance of TPopupMenu.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TPopupActionBar(TComponent * AOwner);
```

Description

Call Create to create to create a pop-up menu at runtime. Pop-up menus placed in forms or data modules at design time are created automatically.

See Also

Create (see page 16072)

6.1.2.3 TPopupActionBar Methods

The methods of the TPopupActionBar class are listed here.

Public Methods

	Name	Description
V	Popup (see page 856)	Displays the pop-up menu onscreen.

6.1.2.3.1 TPopupActionBar.Popup Method

Displays the pop-up menu onscreen.

Pascal

```
procedure Popup(X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall Popup(int X, int Y);
```

Description

Call Popup to bring up the pop-up menu. The menu appears at the screen coordinates indicated by the values (in pixels) of X and Y.

Delphi Examples:

```
{
This example uses a pop-up menu. When the user presses the
mouse button, the pop-up menu appears where the user
clicked the mouse:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := False;
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  PopupMenu1.Popup(Left + X, Top + Y);
end;
```

See Also

Alignment (see page 16176)

AutoPopup (see page 16178)

OnPopup (see page 16184)

6.1.2.4 TPopupActionBar Properties

The properties of the TPopupActionBar class are listed here.

Public Properties

	Name	Description
R	MenuActive (see page 857)	This is MenuActive, a member of class TPopupActionBar.

Published Properties

	Name	Description
	PopupMenu (see page 858)	This is PopupMenu, a member of class TPopupActionBar.

6.1.2.4.1 TPopupActionBar.MenuActive Property

Pascal

```
property MenuActive: Boolean;
```

C++

```
__property Boolean MenuActive;
```

Description

This is MenuActive, a member of class TPopupActionBar.

6.1.2.4.2 TPopupActionBar.PopupMenu Property

Pascal

```
property PopupMenu: TCustomActionPopupMenuEx;
```

C++

```
__property TCustomActionPopupMenuEx PopupMenu;
```

Description

This is PopupMenu, a member of class TPopupActionBar.

6.1.2.5 TPopupActionBar Events

The events of the TPopupActionBar class are listed here.

Published Events

	Name	Description
	OnGetControlClass (see page 858)	This is OnGetControlClass, a member of class TPopupActionBar.

6.1.2.5.1 TPopupActionBar.OnGetControlClass Event

Pascal

```
property OnGetControlClass: TGetControlClassEvent;
```

C++

```
__property TGetControlClassEvent OnGetControlClass;
```

Description

This is OnGetControlClass, a member of class TPopupActionBar.

6.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	NewPopupMenu (see page 859)	Creates and initializes a popup menu.

6.2.1 ActnPopup.NewPopupMenu Function

Creates and initializes a popup menu.

Pascal

```
function NewPopupMenu(Owner: TComponent; const AName: string; Alignment: TPopupAlignment;
AutoPopup: Boolean; const Items: array of TMenuItem): TPopupMenu;
```

C++

```
TPopupMenu NewPopupMenu(TComponent * Owner, const AnsiString AName, TPopupAlignment Alignment,
Boolean AutoPopup, const array of TMenuItem Items);
```

File

ActnPopup

Description

Call NewPopupMenu to create and initialize a popup menu programmatically. NewPopupMenu returns the TPopupMenu component that represents the popup menu, but does not associate that popup menu with any control.

The Owner parameter specifies the component that is responsible for freeing the menu.

The AName parameter specifies the name of the popup menu, which is used to refer to it in code.

The Alignment parameter determines where the menu appears relative to the mouse when the user right clicks on the control that uses the menu.

The AutoPopup parameter specifies whether the menu should appear automatically when the user right clicks the associated control.

The Items parameter is an array of menu items that make up the top level menu items. To create the menu items for the Items parameter, use the NewItem function.

Note: The Items_Size parameter specifies the index of the last menu item in Items (one less than the number of menu items).

See Also

NewMenu (see page 16190)

NewSubMenu (see page 16192)

NewItem (see page 16188)

7 AdaptReq Namespace

This is namespace AdaptReq.

Topics

Name	Description
Classes (see page 863)	The following table lists classes in this documentation.
Interfaces (see page 920)	The following table lists interfaces in this documentation.
Functions (see page 968)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 969)	The following table lists structs, records, enums in this documentation.
Types (see page 971)	The following table lists types in this documentation.
Constants (see page 972)	The following table lists constants in this documentation.

Classes

	Name	Description
	EAdapterRequestException (see page 863)	EAdapterRequestException is raised by WebSnap adapter objects when a request fails.
	TActionFieldValueImpl (see page 864)	This is class AdaptReq.TActionFieldValueImpl.
	TActionFieldValuesImpl (see page 868)	This is class AdaptReq.TActionFieldValuesImpl.
	TAdapterActionFieldValuesList (see page 873)	This is class AdaptReq.TAdapterActionFieldValuesList.
	TAdapterModes (see page 876)	This is class AdaptReq.TAdapterModes.
	TAdapterRequestParamsImpl (see page 880)	This is class AdaptReq.TAdapterRequestParamsImpl.
	TBasicActionRequestImpl (see page 884)	This is class AdaptReq.TBasicActionRequestImpl.
	TBasicActionResponseImpl (see page 890)	This is class AdaptReq.TBasicActionResponseImpl.
	TBasicImageRequestImpl (see page 898)	This is class AdaptReq.TBasicImageRequestImpl.
	TBasicImageResponseImpl (see page 902)	This is class AdaptReq.TBasicImageResponseImpl.
	TEnumActionFieldValueRecordsImpl (see page 904)	This is class AdaptReq.TEnumActionFieldValueRecordsImpl.
	TMultiRecordActionRequestImpl (see page 909)	This is class AdaptReq.TMultiRecordActionRequestImpl.
	TSingleRecordActionRequestImpl (see page 915)	This is class AdaptReq.TSingleRecordActionRequestImpl.

Constants

Name	Description
roRedirect (see page 972)	This is constant AdaptReq.roRedirect.
sActionRequest (see page 972)	This is constant AdaptReq.sActionRequest.
sDelimAdapterMode (see page 973)	This is constant AdaptReq.sDelimAdapterMode.
sDelimOrigFieldValue (see page 973)	This is constant AdaptReq.sDelimOrigFieldValue.
sDelimRecordKeys (see page 973)	This is constant AdaptReq.sDelimRecordKeys.

Functions

	Name	Description
	ExtractStringValue (see page 968)	This is function AdaptReq.ExtractStringValue.
	TestRedirectOptions (see page 968)	This is function AdaptReq.TestRedirectOptions.

Interfaces

	Name	Description
	IActionFieldValue (see page 920)	IActionFieldValue is the interface for obtaining the value of an adapter field.
	IActionFieldValues (see page 925)	IActionFieldValues is the interface for extracting field values from the current HTTP request.
	IActionFieldValuesOfAdapter (see page 933)	IActionFieldValuesOfAdapter is the interface for determining whether a request includes field values that pertain to a specific adapter.
	IActionHandleResponse (see page 934)	This is class AdaptReq.IActionHandleResponse.
	IActionRequest (see page 935)	IActionRequest is the interface for interacting with action requests.
	IActionRespondWith (see page 938)	This is class AdaptReq.IActionRespondWith.
	IActionResponse (see page 941)	IActionResponse is the interface for interacting with action responses.
	IAdapterModes (see page 944)	IAdapterModes is the interface for extracting the mode of an adapter from an HTTP request.
	IAdapterRecordKeys (see page 949)	IAdapterRecordKeys is the interface for extracting name/value pairs from an HTTP request that identify a record for an adapter.
	IAdapterRequestParams (see page 952)	IAdapterRequestParams is the interface for extracting name/value pairs from an HTTP request message.
	IEnumActionFieldValueRecords (see page 957)	IEnumActionFieldValueRecords is the interface for iterating over the data in an HTTP request for each of several adapter records.
	IGetAdapterActionRedirectOptions (see page 963)	This is class AdaptReq.IGetAdapterActionRedirectOptions.
	IImageRequest (see page 965)	IImageRequest is the interface for extracting the name of a requested image from an HTTP request message.
	IImageResponse (see page 967)	This is class AdaptReq.IImageResponse.

Structs, Records, Enums

	Name	Description
	TAdapterActionRedirectOption (see page 969)	This is record AdaptReq.TAdapterActionRedirectOption.
	TExecuteStatus (see page 969)	This is record AdaptReq.TExecuteStatus.
	TRespondWith (see page 970)	This is record AdaptReq.TRespondWith.

Types

Name	Description
TAdapterActionRedirectOptions (see page 971)	This is type AdaptReq.TAdapterActionRedirectOptions.

7.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EAdapterRequestException (see page 863)	EAdapterRequestException is raised by WebSnap adapter objects when a request fails.
	TActionFieldValueImpl (see page 864)	This is class AdaptReq.TActionFieldValueImpl.
	TActionFieldValuesImpl (see page 868)	This is class AdaptReq.TActionFieldValuesImpl.
	TAdapterActionFieldValuesList (see page 873)	This is class AdaptReq.TAdapterActionFieldValuesList.
	TAdapterModes (see page 876)	This is class AdaptReq.TAdapterModes.
	TAdapterRequestParamsImpl (see page 880)	This is class AdaptReq.TAdapterRequestParamsImpl.
	TBasicActionRequestImpl (see page 884)	This is class AdaptReq.TBasicActionRequestImpl.
	TBasicActionResponseImpl (see page 890)	This is class AdaptReq.TBasicActionResponseImpl.
	TBasicImageRequestImpl (see page 898)	This is class AdaptReq.TBasicImageRequestImpl.
	TBasicImageResponseImpl (see page 902)	This is class AdaptReq.TBasicImageResponseImpl.
	TEnumActionFieldValueRecordsImpl (see page 904)	This is class AdaptReq.TEnumActionFieldValueRecordsImpl.
	TMultiRecordActionRequestImpl (see page 909)	This is class AdaptReq.TMultiRecordActionRequestImpl.
	TSingleRecordActionRequestImpl (see page 915)	This is class AdaptReq.TSingleRecordActionRequestImpl.

7.1.1 EAdapterRequestException Class

EAdapterRequestException is raised by WebSnap adapter objects when a request fails.

Class Hierarchy

Pascal

```
EAdapterRequestException = class(EWebBrokerException);
```

C++

```
class EAdapterRequestException : public EWebBrokerException;
```

File

AdaptReq

Description

EAdapterRequestException is raised when a WebSnap adapter object encounters an error while processing a request.

7.1.2 TActionFieldValueImpl Class

Class Hierarchy

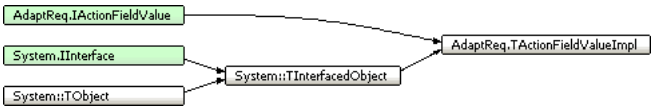

Pascal

```
TActionFieldValueImpl = class(TInterfacedObject, IActionFieldValue);
```

C++

```
class TActionFieldValueImpl : public TInterfacedObject, public IActionFieldValue;
```

File

AdaptReq

Description

This is class AdaptReq.TActionFieldValueImpl.

7.1.2.1 TActionFieldValueImpl Members

The following tables list the members exposed by TActionFieldValueImpl.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetFieldName (see page 866)	Returns the value of the FieldName property.
	GetFile (see page 867)	Returns the value of the Files property.
	GetFileCount (see page 867)	Returns the value of the FileCount property.
	GetValue (see page 867)	Returns the value of the Values property.
	GetValueCount (see page 868)	Returns the value of the ValueCount property.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 865)	This is Create, a member of class TActionFieldValueImpl.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 866)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	FieldName (see page 923)	Indicates the name of the adapter field whose value this interface represents.
R	FileCount (see page 924)	Indicates the number of file references that the Files (see page 924) property lists.
R	Files (see page 924)	Lists the file references that make up this adapter field's value.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	ValueCount (see page 925)	Indicates the number of values that the Values (see page 925) property lists.
R	Values (see page 925)	Lists the values that belong to the adapter field.

7.1.2.2 TActionFieldValueImpl.Create Constructor

Pascal

```
constructor Create(const AFieldName: string; AStrings: TStrings);
```

C++

```
__fastcall TActionFieldValueImpl(const AnsiString AFieldName, TStrings AStrings);
```

Description

This is Create, a member of class TActionFieldValueImpl.

7.1.2.3 TActionFieldValueImpl.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TActionFieldValueImpl();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

7.1.2.4 TActionFieldValueImpl Methods

The methods of the TActionFieldValueImpl class are listed here.

Protected Methods

	Name	Description
	GetFieldName (see page 866)	Returns the value of the FieldName property.
	GetFile (see page 867)	Returns the value of the Files property.
	GetFileCount (see page 867)	Returns the value of the FileCount property.
	GetValue (see page 867)	Returns the value of the Values property.
	GetValueCount (see page 868)	Returns the value of the ValueCount property.

7.1.2.4.1 TActionFieldValueImpl.GetFieldName Method

Returns the value of the FieldName property.

Pascal

```
function GetFieldName: string;
```

C++

```
__fastcall AnsiString GetFieldName();
```

Description

GetFieldName is the read implementation of the FieldName property.

See Also

FieldName (see page 923)

7.1.2.4.2 TActionFieldValueImpl.GetFile Method

Returns the value of the Files property.

Pascal

```
function GetFile(I: Integer): TAbstractWebRequestFile;
```

C++

```
__fastcall TAbstractWebRequestFile GetFile(int I);
```

Description

GetFile is the read implementation of the Files property.

I is the index of the file reference to read.

See Also

Files (see page 924)

7.1.2.4.3 TActionFieldValueImpl.GetFileCount Method

Returns the value of the FileCount property.

Pascal

```
function GetFileCount: Integer;
```

C++

```
__fastcall int GetFileCount();
```

Description

GetFileCount is the read implementation of the FileCount property.

See Also

FileCount (see page 924)

7.1.2.4.4 TActionFieldValueImpl.GetValue Method

Returns the value of the Values property.

Pascal

```
function GetValue(I: Integer): Variant;
```

C++

```
__fastcall Variant GetValue(int I);
```

Description

GetValue is the read implementation of the Values property.

I is the index of the field value to read.

See Also

Values (see page 925)

7.1.2.4.5 TActionFieldValueImpl.GetValueCount Method

Returns the value of the ValueCount property.

Pascal

```
function GetValueCount: Integer;
```

C++

```
__fastcall int GetValueCount();
```

Description

GetValueCount is the read implementation of the ValueCount property.

See Also

ValueCount (see page 925)

7.1.3 TActionFieldValuesImpl Class

Class Hierarchy

Pascal

```
TActionFieldValuesImpl = class(TObject);
```

C++

```
class TActionFieldValuesImpl : public TObject;
```

File

AdaptReq

Description

This is class AdaptReq.TActionFieldValuesImpl.

7.1.3.1 TActionFieldValuesImpl Members

The following tables list the members exposed by TActionFieldValuesImpl.

Protected Methods

	Name	Description
	AddValue (see page 870)	This is AddValue, a member of class TActionFieldValuesImpl.
	GetFieldName (see page 871)	This is GetFieldName, a member of class TActionFieldValuesImpl.
	GetRecordKeyCount (see page 871)	This is GetRecordKeyCount, a member of class TActionFieldValuesImpl.
	GetRecordKeyName (see page 871)	This is GetRecordKeyName, a member of class TActionFieldValuesImpl.
	GetRecordKeyValue (see page 871)	This is GetRecordKeyValue, a member of class TActionFieldValuesImpl.
	IndexOfField (see page 872)	This is IndexOfField, a member of class TActionFieldValuesImpl.
	OrigValueOfField (see page 872)	This is OrigValueOfField, a member of class TActionFieldValuesImpl.

	ValueOfField (see page 872)	This is ValueOfField, a member of class TActionFieldValuesImpl.

Protected Properties

	Name	Description
R	FieldCount (see page 872)	This is FieldCount, a member of class TActionFieldValuesImpl.
R	FieldNames (see page 873)	This is FieldNames, a member of class TActionFieldValuesImpl.
R	OrigValues (see page 873)	This is OrigValues, a member of class TActionFieldValuesImpl.
R	Values (see page 873)	This is Values, a member of class TActionFieldValuesImpl.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 869)	This is Create, a member of class TActionFieldValuesImpl.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 870)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

7.1.3.2 TActionFieldValuesImpl.Create Constructor

Pascal

```
constructor Create(AValues: TStrings);
```

C++

```
__fastcall TActionFieldValuesImpl(TStrings AValues);
```

Description

This is Create, a member of class TActionFieldValuesImpl.

7.1.3.3 TActionFieldValuesImpl.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TActionFieldValuesImpl();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

7.1.3.4 TActionFieldValuesImpl Methods

The methods of the TActionFieldValuesImpl class are listed here.

Protected Methods

	Name	Description
	AddValue (see page 870)	This is AddValue, a member of class TActionFieldValuesImpl.
	GetFieldName (see page 871)	This is GetFieldName, a member of class TActionFieldValuesImpl.
	GetRecordKeyCount (see page 871)	This is GetRecordKeyCount, a member of class TActionFieldValuesImpl.
	GetRecordKeyName (see page 871)	This is GetRecordKeyName, a member of class TActionFieldValuesImpl.
	GetRecordKeyValue (see page 871)	This is GetRecordKeyValue, a member of class TActionFieldValuesImpl.
	IndexOfField (see page 872)	This is IndexOfField, a member of class TActionFieldValuesImpl.
	OrigValueOfField (see page 872)	This is OrigValueOfField, a member of class TActionFieldValuesImpl.
	ValueOfField (see page 872)	This is ValueOfField, a member of class TActionFieldValuesImpl.

7.1.3.4.1 TActionFieldValuesImpl.AddValue Method

Pascal

```
function AddValue(const AName: string): IActionFieldValue;
```

C++

```
__fastcall IActionFieldValue AddValue(const AnsiString AName);
```

Description

This is AddValue, a member of class TActionFieldValuesImpl.

7.1.3.4.2 TActionFieldValuesImpl.GetFieldName Method

Pascal

```
function GetFieldName(I: Integer): string;
```

C++

```
__fastcall AnsiString GetFieldName(int I);
```

Description

This is GetFieldName, a member of class TActionFieldValuesImpl.

7.1.3.4.3 TActionFieldValuesImpl.GetRecordKeyCount Method

Pascal

```
function GetRecordKeyCount: Integer;
```

C++

```
__fastcall int GetRecordKeyCount();
```

Description

This is GetRecordKeyCount, a member of class TActionFieldValuesImpl.

7.1.3.4.4 TActionFieldValuesImpl.GetRecordKeyName Method

Pascal

```
function GetRecordKeyName(I: Integer): string;
```

C++

```
__fastcall AnsiString GetRecordKeyName(int I);
```

Description

This is GetRecordKeyName, a member of class TActionFieldValuesImpl.

7.1.3.4.5 TActionFieldValuesImpl.GetRecordKeyValue Method

Pascal

```
function GetRecordKeyValue(I: Integer): string;
```

C++

```
__fastcall AnsiString GetRecordKeyValue(int I);
```

Description

This is GetRecordKeyValue, a member of class TActionFieldValuesImpl.

7.1.3.4.6 TActionFieldValuesImpl.IndexOfField Method

Pascal

```
function IndexOfField(const AName: string): Integer;
```

C++

```
__fastcall int IndexOfField(const AnsiString AName);
```

Description

This is IndexOfField, a member of class TActionFieldValuesImpl.

7.1.3.4.7 TActionFieldValuesImpl.OrigValueOfField Method

Pascal

```
function OrigValueOfField(const AName: string): IActionFieldValue;
```

C++

```
__fastcall IActionFieldValue OrigValueOfField(const AnsiString AName);
```

Description

This is OrigValueOfField, a member of class TActionFieldValuesImpl.

7.1.3.4.8 TActionFieldValuesImpl.ValueOfField Method

Pascal

```
function ValueOfField(const AName: string): IActionFieldValue;
```

C++

```
__fastcall IActionFieldValue ValueOfField(const AnsiString AName);
```

Description

This is ValueOfField, a member of class TActionFieldValuesImpl.

7.1.3.5 TActionFieldValuesImpl Properties

The properties of the TActionFieldValuesImpl class are listed here.

Protected Properties

	Name	Description
R	FieldCount (see page 872)	This is FieldCount, a member of class TActionFieldValuesImpl.
R	FieldNames (see page 873)	This is FieldNames, a member of class TActionFieldValuesImpl.
R	OrigValues (see page 873)	This is OrigValues, a member of class TActionFieldValuesImpl.
R	Values (see page 873)	This is Values, a member of class TActionFieldValuesImpl.

7.1.3.5.1 TActionFieldValuesImpl.FieldCount Property

Pascal

```
property FieldCount: Integer;
```

C++

```
__property int FieldCount;
```

Description

This is FieldCount, a member of class TActionFieldValuesImpl.

7.1.3.5.2 TActionFieldValuesImpl.FieldNames Property

Pascal

```
property FieldNames [I: Integer]: string;
```

C++

```
__property AnsiString FieldNames[int I];
```

Description

This is FieldNames, a member of class TActionFieldValuesImpl.

7.1.3.5.3 TActionFieldValuesImpl.OrigValues Property

Pascal

```
property OrigValues [I: Integer]: IActionFieldValue;
```

C++

```
__property IActionFieldValue OrigValues[int I];
```

Description

This is OrigValues, a member of class TActionFieldValuesImpl.

7.1.3.5.4 TActionFieldValuesImpl.Values Property

Pascal

```
property Values [I: Integer]: IActionFieldValue;
```

C++

```
__property IActionFieldValue Values[int I];
```

Description

This is Values, a member of class TActionFieldValuesImpl.

7.1.4 TAdapterActionFieldValuesList Class

Class Hierarchy

Pascal

```
TAdapterActionFieldValuesList = class(TObject);
```

C++

```
class TAdapterActionFieldValuesList : public TObject;
```

File

AdaptReq

Description

This is class AdaptReq.TAdapterActionFieldValuesList.

7.1.4.1 TAdapterActionFieldValuesList Members

The following tables list the members exposed by TAdapterActionFieldValuesList.

Protected Methods

	Name	Description
	LocateActionFieldValuesOfAdapter (see page 876)	This is LocateActionFieldValuesOfAdapter, a member of class TAdapterActionFieldValuesList.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 875)	This is Create, a member of class TAdapterActionFieldValuesList.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 875)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	CurrentActionFieldValuesList (see page 876)	This is CurrentActionFieldValuesList, a member of class TAdapterActionFieldValuesList.

7.1.4.2 TAdapterActionFieldValuesList.Create Constructor

Pascal

```
constructor Create(ARequest: TWebRequest);
```

C++

```
__fastcall TAdapterActionFieldValuesList(TWebRequest ARequest);
```

Description

This is Create, a member of class TAdapterActionFieldValuesList.

7.1.4.3 TAdapterActionFieldValuesList.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterActionFieldValuesList();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

7.1.4.4 TAdapterActionFieldValuesList Methods

The methods of the TAdapterActionFieldValuesList class are listed here.

Protected Methods

	Name	Description
	LocateActionFieldValuesOfAdapter (see page 876)	This is LocateActionFieldValuesOfAdapter, a member of class TAdapterActionFieldValuesList.

7.1.4.4.1 TAdapterActionFieldValuesList.LocateActionFieldValuesOfAdapter Method

Pascal

```
function LocateActionFieldValuesOfAdapter(AAdapter: TComponent): Boolean;
```

C++

```
__fastcall Boolean LocateActionFieldValuesOfAdapter(TComponent * AAdapter);
```

Description

This is LocateActionFieldValuesOfAdapter, a member of class TAdapterActionFieldValuesList.

7.1.4.5 TAdapterActionFieldValuesList Properties

The properties of the TAdapterActionFieldValuesList class are listed here.

Public Properties

	Name	Description
R	CurrentActionFieldValuesList (see page 876)	This is CurrentActionFieldValuesList, a member of class TAdapterActionFieldValuesList.

7.1.4.5.1 TAdapterActionFieldValuesList.CurrentActionFieldValuesList Property

Pascal

```
property CurrentActionFieldValuesList: TEnumActionFieldValueRecordsImpl;
```

C++

```
__property TEnumActionFieldValueRecordsImpl CurrentActionFieldValuesList;
```

Description

This is CurrentActionFieldValuesList, a member of class TAdapterActionFieldValuesList.

7.1.5 TAdapterModes Class

Class Hierarchy

Pascal

```
TAdapterModes = class(TObject);
```

C++

```
class TAdapterModes : public TObject;
```

File

AdaptReq

Description

This is class AdaptReq.TAdapterModes.

7.1.5.1 TAdapterModes Members

The following tables list the members exposed by TAdapterModes.

Protected Methods

	Name	Description
	FindAdapterMode (see page 879)	This is FindAdapterMode, a member of class TAdapterModes.
	GetAdapterCount (see page 879)	This is GetAdapterCount, a member of class TAdapterModes.
	GetAdapterMode (see page 879)	This is GetAdapterMode, a member of class TAdapterModes.
	GetAdapterName (see page 879)	This is GetAdapterName, a member of class TAdapterModes.

Protected Properties

	Name	Description
R	AdapterCount (see page 880)	This is AdapterCount, a member of class TAdapterModes.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 878)	This is Create, a member of class TAdapterModes.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 878)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

7.1.5.2 TAdapterModes.Create Constructor

Pascal

```
constructor Create(ARequest: TWebRequest);
```

C++

```
__fastcall TAdapterModes(TWebRequest ARequest);
```

Description

This is Create, a member of class TAdapterModes.

7.1.5.3 TAdapterModes.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterModes();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

7.1.5.4 TAdapterModes Methods

The methods of the TAdapterModes class are listed here.

Protected Methods

	Name	Description
	FindAdapterMode (see page 879)	This is FindAdapterMode, a member of class TAdapterModes.
	GetAdapterCount (see page 879)	This is GetAdapterCount, a member of class TAdapterModes.

	GetAdapterMode (see page 879)	This is GetAdapterMode, a member of class TAdapterModes.
	GetAdapterName (see page 879)	This is GetAdapterName, a member of class TAdapterModes.

7.1.5.4.1 TAdapterModes.FindAdapterMode Method

Pascal

```
function FindAdapterMode(AAdapter: TObject; var AMode: string): Boolean;
```

C++

```
__fastcall Boolean FindAdapterMode(TObject * AAdapter, AnsiString AMode);
```

Description

This is FindAdapterMode, a member of class TAdapterModes.

7.1.5.4.2 TAdapterModes.GetAdapterCount Method

Pascal

```
function GetAdapterCount: Integer;
```

C++

```
__fastcall int GetAdapterCount();
```

Description

This is GetAdapterCount, a member of class TAdapterModes.

7.1.5.4.3 TAdapterModes.GetAdapterMode Method

Pascal

```
function GetAdapterMode(I: Integer): string;
```

C++

```
__fastcall AnsiString GetAdapterMode(int I);
```

Description

This is GetAdapterMode, a member of class TAdapterModes.

7.1.5.4.4 TAdapterModes.GetAdapterName Method

Pascal

```
function GetAdapterName(I: Integer): string;
```

C++

```
__fastcall AnsiString GetAdapterName(int I);
```

Description

This is GetAdapterName, a member of class TAdapterModes.

7.1.5.5 TAdapterModes Properties

The properties of the TAdapterModes class are listed here.

Protected Properties

	Name	Description
R	AdapterCount (see page 880)	This is AdapterCount, a member of class TAdapterModes.

7.1.5.5.1 TAdapterModes.AdapterCount Property

Pascal

```
property AdapterCount: Integer;
```

C++

```
__property int AdapterCount;
```

Description

This is AdapterCount, a member of class TAdapterModes.

7.1.6 TAdapterRequestParamsImpl Class

Class Hierarchy

Pascal

```
TAdapterRequestParamsImpl = class(TObject);
```

C++

```
class TAdapterRequestParamsImpl : public TObject;
```

File

AdaptReq

Description

This is class AdaptReq.TAdapterRequestParamsImpl.

7.1.6.1 TAdapterRequestParamsImpl Members

The following tables list the members exposed by TAdapterRequestParamsImpl.

Protected Methods

	Name	Description
	AssignTo (see page 882)	This is AssignTo, a member of class TAdapterRequestParamsImpl.
	GetParamCount (see page 882)	This is GetParamCount, a member of class TAdapterRequestParamsImpl.
	GetParamName (see page 882)	This is GetParamName, a member of class TAdapterRequestParamsImpl.
	GetParamValue (see page 882)	This is GetParamValue, a member of class TAdapterRequestParamsImpl.
	IndexOfParam (see page 883)	This is IndexOfParam, a member of class TAdapterRequestParamsImpl.
	ValueOfParam (see page 883)	This is ValueOfParam, a member of class TAdapterRequestParamsImpl.

Protected Properties

	Name	Description
R	ParamCount (see page 883)	This is ParamCount, a member of class TAdapterRequestParamsImpl.
R	ParamNames (see page 883)	This is ParamNames, a member of class TAdapterRequestParamsImpl.

R	ParamValues (see page 884)	This is ParamValues, a member of class TAdapterRequestParamsImpl.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 881)	This is Create, a member of class TAdapterRequestParamsImpl.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

7.1.6.2 TAdapterRequestParamsImpl.Create Constructor

Pascal

```
constructor Create(ADispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall TAdapterRequestParamsImpl(IAdapterDispatchParams ADispatchParams);
```

Description

This is Create, a member of class TAdapterRequestParamsImpl.

7.1.6.3 TAdapterRequestParamsImpl Methods

The methods of the TAdapterRequestParamsImpl class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 882)	This is AssignTo, a member of class TAdapterRequestParamsImpl.
	GetParamCount (see page 882)	This is GetParamCount, a member of class TAdapterRequestParamsImpl.
	GetParamName (see page 882)	This is GetParamName, a member of class TAdapterRequestParamsImpl.
	GetParamValue (see page 882)	This is GetParamValue, a member of class TAdapterRequestParamsImpl.
	IndexOfParam (see page 883)	This is IndexOfParam, a member of class TAdapterRequestParamsImpl.
	ValueOfParam (see page 883)	This is ValueOfParam, a member of class TAdapterRequestParamsImpl.

7.1.6.3.1 TAdapterRequestParamsImpl.AssignTo Method

Pascal

```
procedure AssignTo(AStrings: TStrings);
```

C++

```
__fastcall AssignTo(TStrings AStrings);
```

Description

This is AssignTo, a member of class TAdapterRequestParamsImpl.

7.1.6.3.2 TAdapterRequestParamsImpl.GetParamCount Method

Pascal

```
function GetParamCount: Integer;
```

C++

```
__fastcall int GetParamCount();
```

Description

This is GetParamCount, a member of class TAdapterRequestParamsImpl.

7.1.6.3.3 TAdapterRequestParamsImpl.GetParamName Method

Pascal

```
function GetParamName(I: Integer): string;
```

C++

```
__fastcall AnsiString GetParamName(int I);
```

Description

This is GetParamName, a member of class TAdapterRequestParamsImpl.

7.1.6.3.4 TAdapterRequestParamsImpl.GetParamValue Method

Pascal

```
function GetParamValue(I: Integer): string;
```

C++

```
__fastcall AnsiString GetParamValue(int I);
```

Description

This is GetParamValue, a member of class TAdapterRequestParamsImpl.

7.1.6.3.5 TAdapterRequestParamsImpl.IndexOfParam Method

Pascal

```
function IndexOfParam(const AName: string): Integer;
```

C++

```
__fastcall int IndexOfParam(const AnsiString AName);
```

Description

This is IndexOfParam, a member of class TAdapterRequestParamsImpl.

7.1.6.3.6 TAdapterRequestParamsImpl.ValueOfParam Method

Pascal

```
function ValueOfParam(const AName: string): string;
```

C++

```
__fastcall AnsiString ValueOfParam(const AnsiString AName);
```

Description

This is ValueOfParam, a member of class TAdapterRequestParamsImpl.

7.1.6.4 TAdapterRequestParamsImpl Properties

The properties of the TAdapterRequestParamsImpl class are listed here.

Protected Properties

	Name	Description
R	ParamCount (see page 883)	This is ParamCount, a member of class TAdapterRequestParamsImpl.
R	ParamNames (see page 883)	This is ParamNames, a member of class TAdapterRequestParamsImpl.
R	ParamValues (see page 884)	This is ParamValues, a member of class TAdapterRequestParamsImpl.

7.1.6.4.1 TAdapterRequestParamsImpl.ParamCount Property

Pascal

```
property ParamCount: Integer;
```

C++

```
__property int ParamCount;
```

Description

This is ParamCount, a member of class TAdapterRequestParamsImpl.

7.1.6.4.2 TAdapterRequestParamsImpl.ParamNames Property

Pascal

```
property ParamNames [I: Integer]: string;
```

C++

```
__property AnsiString ParamNames[int I];
```

Description

This is ParamNames, a member of class TAdapterRequestParamsImpl.

7.1.6.4.3 TAdapterRequestParamsImpl.ParamValues Property

Pascal

```
property ParamValues [I: Integer]: string;
```

C++

```
__property AnsiString ParamValues[int I];
```

Description

This is ParamValues, a member of class TAdapterRequestParamsImpl.

7.1.7 TBasicActionRequestImpl Class

Class Hierarchy

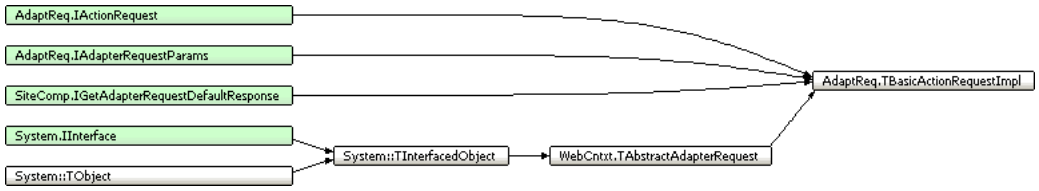

Pascal

```
TBasicActionRequestImpl = class(TAbstractAdapterRequest, IActionRequest,
IAdapterRequestParams, IGetAdapterRequestDefaultResponse);
```

C++

```
class TBasicActionRequestImpl : public TAbstractAdapterRequest, public IActionRequest, public
IAdapterRequestParams, public IGetAdapterRequestDefaultResponse;
```

File

AdaptReq

Description

This is class AdaptReq.TBasicActionRequestImpl.

7.1.7.1 TBasicActionRequestImpl Members

The following tables list the members exposed by TBasicActionRequestImpl.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetActionName (see page 887)	Returns the value of the ActionName property.
	GetFailurePage (see page 887)	This is GetFailurePage, a member of class TBasicActionRequestImpl.
	GetHandled (see page 888)	Returns the value of the Handled property.
	GetSuccessPage (see page 888)	This is GetSuccessPage, a member of class TBasicActionRequestImpl.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetHandled (see page 888)	Sets the value of the Handled property.

Protected Properties

	Name	Description
R	Request (see page 24642)	This is Request, a member of class TAbstractAdapterRequest.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AssignTo (see page 954)	Fills a string list to contain the name/value pairs from the HTTP request message.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 886)	This is Create, a member of class TBasicActionRequestImpl.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 886)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetParamCount (see page 954)	Returns the value of the ParamCount (see page 956) property.
	GetParamName (see page 954)	Returns the value of the ParamNames (see page 956) property.
	GetParamValue (see page 955)	Returns the value of the ParamValues (see page 957) property.
	IndexOfParam (see page 955)	Returns the index of a specified name/value pair given its name.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ValueOfParam (see page 955)	Returns the value associated with a specified name.

Public Properties

	Name	Description
R	ActionName (see page 889)	Identifies the action to server-side scripts.
R	ActionParams (see page 889)	This is ActionParams, a member of class TBasicActionRequestImpl.
R	DispatchParams (see page 889)	This is DispatchParams, a member of class TBasicActionRequestImpl.
R	FailurePage (see page 889)	This is FailurePage, a member of class TBasicActionRequestImpl.
	Handled (see page 938)	Indicates whether the request has been handled.
R	ParamCount (see page 956)	Indicates the number of name/value pairs that can be accessed using this interface.
R	ParamNames (see page 956)	Lists the names of the name/value pairs that can be accessed using this interface.
R	ParamValues (see page 957)	Lists the values of the name/value pairs that can be accessed using this interface.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	SuccessPage (see page 890)	This is SuccessPage, a member of class TBasicActionRequestImpl.

7.1.7.2 TBasicActionRequestImpl.Create Constructor

Pascal

```
constructor Create(ADispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall TBasicActionRequestImpl(IAdapterDispatchParams ADispatchParams);
```

Description

This is Create, a member of class TBasicActionRequestImpl.

7.1.7.3 TBasicActionRequestImpl.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBasicActionRequestImpl();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a

virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

7.1.7.4 TBasicActionRequestImpl Methods

The methods of the TBasicActionRequestImpl class are listed here.

Protected Methods

	Name	Description
	GetActionName (see page 887)	Returns the value of the ActionName property.
	GetFailurePage (see page 887)	This is GetFailurePage, a member of class TBasicActionRequestImpl.
	GetHandled (see page 888)	Returns the value of the Handled property.
	GetSuccessPage (see page 888)	This is GetSuccessPage, a member of class TBasicActionRequestImpl.
	SetHandled (see page 888)	Sets the value of the Handled property.

7.1.7.4.1 TBasicActionRequestImpl.GetActionName Method

Returns the value of the ActionName property.

Pascal

```
function GetActionName: string;
```

C++

```
__fastcall AnsiString GetActionName();
```

Description

GetActionName is the read implementation of the ActionName property.

See Also

ActionName (see page 938)

7.1.7.4.2 TBasicActionRequestImpl.GetFailurePage Method

Pascal

```
function GetFailurePage: string;
```

C++

```
__fastcall AnsiString GetFailurePage();
```

Description

This is GetFailurePage, a member of class TBasicActionRequestImpl.

7.1.7.4.3 TBasicActionRequestImpl.GetHandled Method

Returns the value of the Handled property.

Pascal

```
function GetHandled: Boolean;
```

C++

```
__fastcall Boolean GetHandled();
```

Description

GetHandled is the read implementation of the Handled property.

See Also

Handled (see page 938)

7.1.7.4.4 TBasicActionRequestImpl.GetSuccessPage Method

Pascal

```
function GetSuccessPage: string;
```

C++

```
__fastcall AnsiString GetSuccessPage();
```

Description

This is GetSuccessPage, a member of class TBasicActionRequestImpl.

7.1.7.4.5 TBasicActionRequestImpl.SetHandled Method

Sets the value of the Handled property.

Pascal

```
procedure SetHandled(AValue: Boolean);
```

C++

```
__fastcall SetHandled(Boolean AValue);
```

Description

SetHandled is the write implementation of the Handled property.

See Also

Handled (see page 938)

7.1.7.5 TBasicActionRequestImpl Properties

The properties of the TBasicActionRequestImpl class are listed here.

Public Properties

	Name	Description
R	ActionName (see page 889)	Identifies the action to server-side scripts.
R	ActionParams (see page 889)	This is ActionParams, a member of class TBasicActionRequestImpl.
R	DispatchParams (see page 889)	This is DispatchParams, a member of class TBasicActionRequestImpl.

R	FailurePage (see page 889)	This is FailurePage, a member of class TBasicActionRequestImpl.
R	SuccessPage (see page 890)	This is SuccessPage, a member of class TBasicActionRequestImpl.

7.1.7.5.1 TBasicActionRequestImpl.ActionName Property

Identifies the action to server-side scripts.

Pascal

```
property ActionName: string;
```

C++

```
__property AnsiString ActionName;
```

Description

The ActionName property defines the name used to access the adapter action from server-side scripts.

7.1.7.5.2 TBasicActionRequestImpl.ActionParams Property

Pascal

```
property ActionParams: TAdapterRequestParamsImpl;
```

C++

```
__property TAdapterRequestParamsImpl ActionParams;
```

Description

This is ActionParams, a member of class TBasicActionRequestImpl.

7.1.7.5.3 TBasicActionRequestImpl.DispatchParams Property

Pascal

```
property DispatchParams: IAdapterDispatchParams;
```

C++

```
__property IAdapterDispatchParams DispatchParams;
```

Description

This is DispatchParams, a member of class TBasicActionRequestImpl.

7.1.7.5.4 TBasicActionRequestImpl.FailurePage Property

Pascal

```
property FailurePage: string;
```

C++

```
__property AnsiString FailurePage;
```

Description

This is FailurePage, a member of class TBasicActionRequestImpl.

7.1.7.5.5 TBasicActionRequestImpl.SuccessPage Property

Pascal

```
property SuccessPage: string;
```

C++

```
__property AnsiString SuccessPage;
```

Description

This is SuccessPage, a member of class TBasicActionRequestImpl.

7.1.8 TBasicActionResponseImpl Class

Class Hierarchy

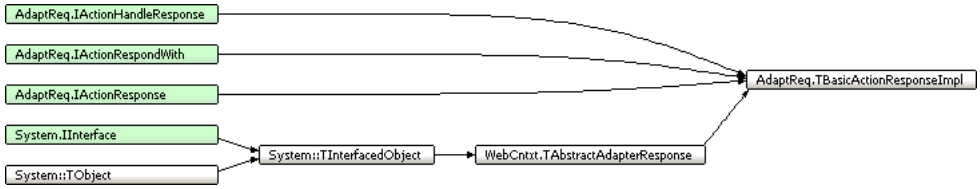

Pascal

```
TBasicActionResponseImpl = class(TAbstractAdapterResponse, IActionResponse,
IActionRespondWith, IActionHandleResponse);
```

C++

```
class TBasicActionResponseImpl : public TAbstractAdapterResponse, public IActionResponse,
public IActionRespondWith, public IActionHandleResponse;
```

File

AdaptReq

Description

This is class AdaptReq.TBasicActionResponseImpl.

7.1.8.1 TBasicActionResponseImpl Members

The following tables list the members exposed by TBasicActionResponseImpl.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetExecuteStatus (see page 892)	Returns the value of the ExecuteStatus property.

	GetHandled (see page 893)	Returns the value of the Handled property.
	GetRedirectOptions (see page 893)	This is GetRedirectOptions, a member of class TBasicActionResponseImpl.
	GetRespondWith (see page 893)	This is GetRespondWith, a member of class TBasicActionResponseImpl.
	HandleResponse (see page 894)	This is HandleResponse, a member of class TBasicActionResponseImpl.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RespondWithComponentPage (see page 894)	This is RespondWithComponentPage, a member of class TBasicActionResponseImpl.
	RespondWithFailurePage (see page 894)	This is RespondWithFailurePage, a member of class TBasicActionResponseImpl.
	RespondWithPage (see page 894)	This is RespondWithPage, a member of class TBasicActionResponseImpl.
	RespondWithSuccessPage (see page 894)	This is RespondWithSuccessPage, a member of class TBasicActionResponseImpl.
	RespondWithURL (see page 895)	This is RespondWithURL, a member of class TBasicActionResponseImpl.
	SetExecuteStatus (see page 895)	Sets the value of the ExecuteStatus property.
	SetHandled (see page 895)	Sets the value of the Handled property.
	SetRedirectOptions (see page 895)	This is SetRedirectOptions, a member of class TBasicActionResponseImpl.

Protected Properties

	Name	Description
R	Response (see page 24644)	This is Response, a member of class TAbstractAdapterResponse.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 892)	This is Create, a member of class TBasicActionResponseImpl.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	ExecuteStatus (see page 896)	Indicates the status of executing the action.
R	FailurePage (see page 896)	This is FailurePage, a member of class TBasicActionResponseImpl.
	Handled (see page 896)	Indicates whether the incoming request has been handled.
	RedirectOptions (see page 897)	This is RedirectOptions, a member of class TBasicActionResponseImpl.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	RespondWith (see page 897)	This is RespondWith, a member of class TBasicActionResponseImpl.
R	SuccessPage (see page 897)	This is SuccessPage, a member of class TBasicActionResponseImpl.

7.1.8.2 TBasicActionResponseImpl.Create Constructor

Pascal

```
constructor Create(AActionRequest: TBasicActionRequestImpl);
```

C++

```
__fastcall TBasicActionResponseImpl(TBasicActionRequestImpl AActionRequest);
```

Description

This is Create, a member of class TBasicActionResponseImpl.

7.1.8.3 TBasicActionResponseImpl Methods

The methods of the TBasicActionResponseImpl class are listed here.

Protected Methods

	Name	Description
	GetExecuteStatus (see page 892)	Returns the value of the ExecuteStatus property.
	GetHandled (see page 893)	Returns the value of the Handled property.
	GetRedirectOptions (see page 893)	This is GetRedirectOptions, a member of class TBasicActionResponseImpl.
	GetRespondWith (see page 893)	This is GetRespondWith, a member of class TBasicActionResponseImpl.
	HandleResponse (see page 894)	This is HandleResponse, a member of class TBasicActionResponseImpl.
	RespondWithComponentPage (see page 894)	This is RespondWithComponentPage, a member of class TBasicActionResponseImpl.
	RespondWithFailurePage (see page 894)	This is RespondWithFailurePage, a member of class TBasicActionResponseImpl.
	RespondWithPage (see page 894)	This is RespondWithPage, a member of class TBasicActionResponseImpl.
	RespondWithSuccessPage (see page 894)	This is RespondWithSuccessPage, a member of class TBasicActionResponseImpl.
	RespondWithURL (see page 895)	This is RespondWithURL, a member of class TBasicActionResponseImpl.
	SetExecuteStatus (see page 895)	Sets the value of the ExecuteStatus property.
	SetHandled (see page 895)	Sets the value of the Handled property.
	SetRedirectOptions (see page 895)	This is SetRedirectOptions, a member of class TBasicActionResponseImpl.

7.1.8.3.1 TBasicActionResponseImpl.GetExecuteStatus Method

Returns the value of the ExecuteStatus property.

Pascal

```
function GetExecuteStatus: TExecuteStatus;
```

C++

```
__fastcall TExecuteStatus GetExecuteStatus();
```

Description

GetExecuteStatus is the read implementation of the ExecuteStatus property.

See Also

ExecuteStatus (see page 944)

7.1.8.3.2 TBasicActionResponseImpl.GetHandled Method

Returns the value of the Handled property.

Pascal

```
function GetHandled: Boolean;
```

C++

```
__fastcall Boolean GetHandled();
```

Description

GetHandled is the read implementation of the Handled property.

See Also

Handled (see page 944)

7.1.8.3.3 TBasicActionResponseImpl.GetRedirectOptions Method

Pascal

```
function GetRedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__fastcall TAdapterActionRedirectOptions GetRedirectOptions();
```

Description

This is GetRedirectOptions, a member of class TBasicActionResponseImpl.

7.1.8.3.4 TBasicActionResponseImpl.GetRespondWith Method

Pascal

```
function GetRespondWith: TRespondWith;
```

C++

```
__fastcall TRespondWith GetRespondWith();
```

Description

This is GetRespondWith, a member of class TBasicActionResponseImpl.

7.1.8.3.5 TBasicActionResponseImpl.HandleResponse Method

Pascal

```
procedure HandleResponse;
```

C++

```
__fastcall HandleResponse();
```

Description

This is HandleResponse, a member of class TBasicActionResponseImpl.

7.1.8.3.6 TBasicActionResponseImpl.RespondWithComponentPage Method

Pascal

```
procedure RespondWithComponentPage(const AComponent: TComponent);
```

C++

```
__fastcall RespondWithComponentPage(const TComponent * AComponent);
```

Description

This is RespondWithComponentPage, a member of class TBasicActionResponseImpl.

7.1.8.3.7 TBasicActionResponseImpl.RespondWithFailurePage Method

Pascal

```
procedure RespondWithFailurePage;
```

C++

```
__fastcall RespondWithFailurePage();
```

Description

This is RespondWithFailurePage, a member of class TBasicActionResponseImpl.

7.1.8.3.8 TBasicActionResponseImpl.RespondWithPage Method

Pascal

```
procedure RespondWithPage(const APageName: string);
```

C++

```
__fastcall RespondWithPage(const AnsiString APageName);
```

Description

This is RespondWithPage, a member of class TBasicActionResponseImpl.

7.1.8.3.9 TBasicActionResponseImpl.RespondWithSuccessPage Method

Pascal

```
procedure RespondWithSuccessPage;
```

C++

```
__fastcall RespondWithSuccessPage();
```

Description

This is RespondWithSuccessPage, a member of class TBasicActionResponseImpl.

7.1.8.3.10 TBasicActionResponseImpl.RespondWithURL Method

Pascal

```
procedure RespondWithURL(const AURL: string);
```

C++

```
__fastcall RespondWithURL(const AnsiString AURL);
```

Description

This is RespondWithURL, a member of class TBasicActionResponseImpl.

7.1.8.3.11 TBasicActionResponseImpl.SetExecuteStatus Method

Sets the value of the ExecuteStatus property.

Pascal

```
procedure SetExecuteStatus(Value: TExecuteStatus);
```

C++

```
__fastcall SetExecuteStatus(TExecuteStatus Value);
```

Description

SetExecuteStatus is the write implementation of the ExecuteStatus property.

See Also

ExecuteStatus (see page 944)

7.1.8.3.12 TBasicActionResponseImpl.SetHandled Method

Sets the value of the Handled property.

Pascal

```
procedure SetHandled(Value: Boolean);
```

C++

```
__fastcall SetHandled(Boolean Value);
```

Description

SetHandled is the write implementation of the Handled property.

See Also

Handled (see page 944)

7.1.8.3.13 TBasicActionResponseImpl.SetRedirectOptions Method

Pascal

```
procedure SetRedirectOptions(AOptions: TAdapterActionRedirectOptions);
```

C++

```
__fastcall SetRedirectOptions(TAdapterActionRedirectOptions AOptions);
```

Description

This is SetRedirectOptions, a member of class TBasicActionResponseImpl.

7.1.8.4 TBasicActionResponseImpl Properties

The properties of the TBasicActionResponseImpl class are listed here.

Public Properties

	Name	Description
	ExecuteStatus (see page 896)	Indicates the status of executing the action.
R	FailurePage (see page 896)	This is FailurePage, a member of class TBasicActionResponseImpl.
	Handled (see page 896)	Indicates whether the incoming request has been handled.
	RedirectOptions (see page 897)	This is RedirectOptions, a member of class TBasicActionResponseImpl.
R	RespondWith (see page 897)	This is RespondWith, a member of class TBasicActionResponseImpl.
R	SuccessPage (see page 897)	This is SuccessPage, a member of class TBasicActionResponseImpl.

7.1.8.4.1 TBasicActionResponseImpl.ExecuteStatus Property

Indicates the status of executing the action.

Pascal

```
property ExecuteStatus: TExecuteStatus;
```

C++

```
__property TExecuteStatus ExecuteStatus;
```

Description

The ExecuteStatus property indicates whether the request was completely handled, and if so, whether the attempt to handle the request was successful.

See Also

Handled (see page 944)

7.1.8.4.2 TBasicActionResponseImpl.FailurePage Property

Pascal

```
property FailurePage: string;
```

C++

```
__property AnsiString FailurePage;
```

Description

This is FailurePage, a member of class TBasicActionResponseImpl.

7.1.8.4.3 TBasicActionResponseImpl.Handled Property

Indicates whether the incoming request has been handled.

Pascal

```
property Handled: Boolean;
```

C++

```
__property Boolean Handled;
```

Description

Note: WebSnap sets Handled to true when the response has been handled.

7.1.8.4.4 TBasicActionResponseImpl.RedirectOptions Property

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

This is RedirectOptions, a member of class TBasicActionResponseImpl.

7.1.8.4.5 TBasicActionResponseImpl.RespondWith Property

Pascal

```
property RespondWith: TRespondWith;
```

C++

```
__property TRespondWith RespondWith;
```

Description

This is RespondWith, a member of class TBasicActionResponseImpl.

7.1.8.4.6 TBasicActionResponseImpl.SuccessPage Property

Pascal

```
property SuccessPage: string;
```

C++

```
__property AnsiString SuccessPage;
```

Description

This is SuccessPage, a member of class TBasicActionResponseImpl.

7.1.9 TBasicImageRequestImpl Class

Class Hierarchy

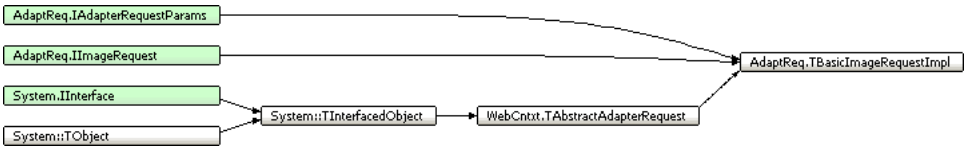

Pascal

```
TBasicImageRequestImpl = class(TAbstractAdapterRequest, IImageRequest, IAdapterRequestParams);
```

C++

```
class TBasicImageRequestImpl : public TAbstractAdapterRequest, public IImageRequest, public
IAdapterRequestParams;
```

File

AdaptReq

Description

This is class AdaptReq.TBasicImageRequestImpl.

7.1.9.1 TBasicImageRequestImpl Members

The following tables list the members exposed by TBasicImageRequestImpl.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetImageName (see page 901)	Returns the value of the ImageName property.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Protected Properties

	Name	Description
R	ActionRequestParams (see page 901)	This is ActionRequestParams, a member of class TBasicImageRequestImpl.
R	ImageName (see page 902)	Indicates the name of the requested image.
R	Request (see page 24642)	This is Request, a member of class TAbstractAdapterRequest.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.

V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AssignTo (see page 954)	Fills a string list to contain the name/value pairs from the HTTP request message.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 900)	This is Create, a member of class TBasicImageRequestImpl.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 900)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetParamCount (see page 954)	Returns the value of the ParamCount (see page 956) property.
	GetParamName (see page 954)	Returns the value of the ParamNames (see page 956) property.
	GetParamValue (see page 955)	Returns the value of the ParamValues (see page 957) property.
	IndexOfParam (see page 955)	Returns the index of a specified name/value pair given its name.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ValueOfParam (see page 955)	Returns the value associated with a specified name.

Public Properties

	Name	Description
R	DispatchParams (see page 901)	This is DispatchParams, a member of class TBasicImageRequestImpl.
R	ParamCount (see page 956)	Indicates the number of name/value pairs that can be accessed using this interface.
R	ParamNames (see page 956)	Lists the names of the name/value pairs that can be accessed using this interface.
R	ParamValues (see page 957)	Lists the values of the name/value pairs that can be accessed using this interface.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

7.1.9.2 TBasicImageRequestImpl.Create Constructor

Pascal

```
constructor Create(ADispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall TBasicImageRequestImpl(IAdapterDispatchParams ADispatchParams);
```

Description

This is Create, a member of class TBasicImageRequestImpl.

7.1.9.3 TBasicImageRequestImpl.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBasicImageRequestImpl();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

7.1.9.4 TBasicImageRequestImpl Methods

The methods of the TBasicImageRequestImpl class are listed here.

Protected Methods

	Name	Description
	GetImageName (see page 901)	Returns the value of the ImageName property.

7.1.9.4.1 TBasicImageRequestImpl.GetImageName Method

Returns the value of the ImageName property.

Pascal

```
function GetImageName: string;
```

C++

```
__fastcall AnsiString GetImageName();
```

Description

GetImageName is the read implementation of the ImageName property.

See Also

ImageName (see page 967)

7.1.9.5 TBasicImageRequestImpl Properties

The properties of the TBasicImageRequestImpl class are listed here.

Protected Properties

	Name	Description
R	ActionRequestParams (see page 901)	This is ActionRequestParams, a member of class TBasicImageRequestImpl.
R	ImageName (see page 902)	Indicates the name of the requested image.

Public Properties

	Name	Description
R	DispatchParams (see page 901)	This is DispatchParams, a member of class TBasicImageRequestImpl.

7.1.9.5.1 TBasicImageRequestImpl.ActionRequestParams Property

Pascal

```
property ActionRequestParams: TAdapterRequestParamsImpl;
```

C++

```
__property TAdapterRequestParamsImpl ActionRequestParams;
```

Description

This is ActionRequestParams, a member of class TBasicImageRequestImpl.

7.1.9.5.2 TBasicImageRequestImpl.DispatchParams Property

Pascal

```
property DispatchParams: IAdapterDispatchParams;
```

C++

```
__property IAdapterDispatchParams DispatchParams;
```

Description

This is DispatchParams, a member of class TBasicImageRequestImpl.

7.1.9.5.3 TBasicImageRequestImpl.ImageName Property

Indicates the name of the requested image.

Pascal

```
property ImageName: string;
```

C++

```
__property AnsiString ImageName;
```

Description

Read ImageName to obtain the name of the image that the HTTP request message requests.

7.1.10 TBasicImageResponseImpl Class

Class Hierarchy

Pascal

```
TBasicImageResponseImpl = class(TAbstractAdapterResponse, IImageResponse);
```

C++

```
class TBasicImageResponseImpl : public TAbstractAdapterResponse, public IImageResponse;
```

File

AdaptReq

Description

This is class AdaptReq.TBasicImageResponseImpl.

7.1.10.1 TBasicImageResponseImpl Members

The following tables list the members exposed by TBasicImageResponseImpl.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Protected Properties

	Name	Description
R	Response (see page 24644)	This is Response, a member of class TAbstractAdapterResponse.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 903)	This is Create, a member of class TBasicImageResponseImpl.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

7.1.10.2 TBasicImageResponseImpl.Create Constructor

Pascal

```
constructor Create(AImageRequest: TBasicImageRequestImpl);
```

C++

```
__fastcall TBasicImageResponseImpl(TBasicImageRequestImpl AImageRequest);
```

Description

This is Create, a member of class TBasicImageResponseImpl.

7.1.11 TEnumActionFieldValueRecordsImpl Class

Class Hierarchy

Pascal

```
TEnumActionFieldValueRecordsImpl = class(TObject);
```

C++

```
class TEnumActionFieldValueRecordsImpl : public TObject;
```

File

AdaptReq

Description

This is class AdaptReq.TEnumActionFieldValueRecordsImpl.

7.1.11.1 TEnumActionFieldValueRecordsImpl Members

The following tables list the members exposed by TEnumActionFieldValueRecordsImpl.

Protected Methods

	Name	Description
	EndIterator (see page 906)	This is EndIterator, a member of class TEnumActionFieldValueRecordsImpl.
	FindPopulatedRecord (see page 907)	This is FindPopulatedRecord, a member of class TEnumActionFieldValueRecordsImpl.
	GetEOF (see page 907)	This is GetEOF, a member of class TEnumActionFieldValueRecordsImpl.
	GetFirstRecord (see page 907)	This is GetFirstRecord, a member of class TEnumActionFieldValueRecordsImpl.
	GetInIterator (see page 907)	This is GetInIterator, a member of class TEnumActionFieldValueRecordsImpl.
	GetNextRecord (see page 907)	This is GetNextRecord, a member of class TEnumActionFieldValueRecordsImpl.
	GetRecord (see page 908)	This is GetRecord, a member of class TEnumActionFieldValueRecordsImpl.
	GetRecordCount (see page 908)	This is GetRecordCount, a member of class TEnumActionFieldValueRecordsImpl.

Protected Properties

	Name	Description
R	Eof (see page 909)	This is Eof, a member of class TEnumActionFieldValueRecordsImpl.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 905)	This is Create, a member of class TEnumActionFieldValueRecordsImpl.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 905)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	AdapterName (see page 908)	This is AdapterName, a member of class TEnumActionFieldValueRecordsImpl.
R	CurrentFieldValues (see page 909)	This is CurrentFieldValues, a member of class TEnumActionFieldValueRecordsImpl.

7.1.11.2 TEnumActionFieldValueRecordsImpl.Create Constructor

Pascal

```
constructor Create(const AAdapterName: string; AFieldValues: TStrings);
```

C++

```
__fastcall TEnumActionFieldValueRecordsImpl(const AnsiString AAdapterName, TStrings
AFieldValues);
```

Description

This is Create, a member of class TEnumActionFieldValueRecordsImpl.

7.1.11.3 TEnumActionFieldValueRecordsImpl.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TEnumActionFieldValueRecordsImpl();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

7.1.11.4 TEnumActionFieldValueRecordsImpl Methods

The methods of the TEnumActionFieldValueRecordsImpl class are listed here.

Protected Methods

	Name	Description
	EndIterator (see page 906)	This is EndIterator, a member of class TEnumActionFieldValueRecordsImpl.
	FindPopulatedRecord (see page 907)	This is FindPopulatedRecord, a member of class TEnumActionFieldValueRecordsImpl.
	GetEOF (see page 907)	This is GetEOF, a member of class TEnumActionFieldValueRecordsImpl.
	GetFirstRecord (see page 907)	This is GetFirstRecord, a member of class TEnumActionFieldValueRecordsImpl.
	GetInIterator (see page 907)	This is GetInIterator, a member of class TEnumActionFieldValueRecordsImpl.
	GetNextRecord (see page 907)	This is GetNextRecord, a member of class TEnumActionFieldValueRecordsImpl.
	GetRecord (see page 908)	This is GetRecord, a member of class TEnumActionFieldValueRecordsImpl.
	GetRecordCount (see page 908)	This is GetRecordCount, a member of class TEnumActionFieldValueRecordsImpl.

7.1.11.4.1 TEnumActionFieldValueRecordsImpl.EndIterator Method

Pascal

```
procedure EndIterator;
```

C++

```
__fastcall EndIterator();
```

Description

This is EndIterator, a member of class TEnumActionFieldValueRecordsImpl.

7.1.11.4.2 TEnumActionFieldValueRecordsImpl.FindPopulatedRecord Method

Pascal

```
function FindPopulatedRecord: Boolean;
```

C++

```
__fastcall Boolean FindPopulatedRecord();
```

Description

This is FindPopulatedRecord, a member of class TEnumActionFieldValueRecordsImpl.

7.1.11.4.3 TEnumActionFieldValueRecordsImpl.GetEOF Method

Pascal

```
function GetEOF: Boolean;
```

C++

```
__fastcall Boolean GetEOF();
```

Description

This is GetEOF, a member of class TEnumActionFieldValueRecordsImpl.

7.1.11.4.4 TEnumActionFieldValueRecordsImpl.GetFirstRecord Method

Pascal

```
function GetFirstRecord: Boolean;
```

C++

```
__fastcall Boolean GetFirstRecord();
```

Description

This is GetFirstRecord, a member of class TEnumActionFieldValueRecordsImpl.

7.1.11.4.5 TEnumActionFieldValueRecordsImpl.GetInIterator Method

Pascal

```
function GetInIterator: Boolean;
```

C++

```
__fastcall Boolean GetInIterator();
```

Description

This is GetInIterator, a member of class TEnumActionFieldValueRecordsImpl.

7.1.11.4.6 TEnumActionFieldValueRecordsImpl.GetNextRecord Method

Pascal

```
function GetNextRecord: Boolean;
```

C++

```
__fastcall Boolean GetNextRecord();
```

Description

This is GetNextRecord, a member of class TEnumActionFieldValueRecordsImpl.

7.1.11.4.7 TEnumActionFieldValueRecordsImpl.GetRecord Method

Pascal

```
function GetRecord(AIndex: Integer): Boolean;
```

C++

```
__fastcall Boolean GetRecord(int AIndex);
```

Description

This is GetRecord, a member of class TEnumActionFieldValueRecordsImpl.

7.1.11.4.8 TEnumActionFieldValueRecordsImpl.GetRecordCount Method

Pascal

```
function GetRecordCount: Integer;
```

C++

```
__fastcall int GetRecordCount();
```

Description

This is GetRecordCount, a member of class TEnumActionFieldValueRecordsImpl.

7.1.11.5 TEnumActionFieldValueRecordsImpl Properties

The properties of the TEnumActionFieldValueRecordsImpl class are listed here.

Protected Properties

	Name	Description
R	Eof (see page 909)	This is Eof, a member of class TEnumActionFieldValueRecordsImpl.

Public Properties

	Name	Description
R	AdapterName (see page 908)	This is AdapterName, a member of class TEnumActionFieldValueRecordsImpl.
R	CurrentFieldValues (see page 909)	This is CurrentFieldValues, a member of class TEnumActionFieldValueRecordsImpl.

7.1.11.5.1 TEnumActionFieldValueRecordsImpl.AdapterName Property

Pascal

```
property AdapterName: string;
```

C++

```
__property AnsiString AdapterName;
```

Description

This is AdapterName, a member of class TEnumActionFieldValueRecordsImpl.

7.1.11.5.2 TEnumActionFieldValueRecordsImpl.CurrentFieldValues Property

Pascal

```
property CurrentFieldValues: TActionFieldValuesImpl;
```

C++

```
__property TActionFieldValuesImpl CurrentFieldValues;
```

Description

This is CurrentFieldValues, a member of class TEnumActionFieldValueRecordsImpl.

7.1.11.5.3 TEnumActionFieldValueRecordsImpl.Eof Property

Pascal

```
property Eof: Boolean;
```

C++

```
__property Boolean Eof;
```

Description

This is Eof, a member of class TEnumActionFieldValueRecordsImpl.

7.1.12 TMultiRecordActionRequestImpl Class

Class Hierarchy

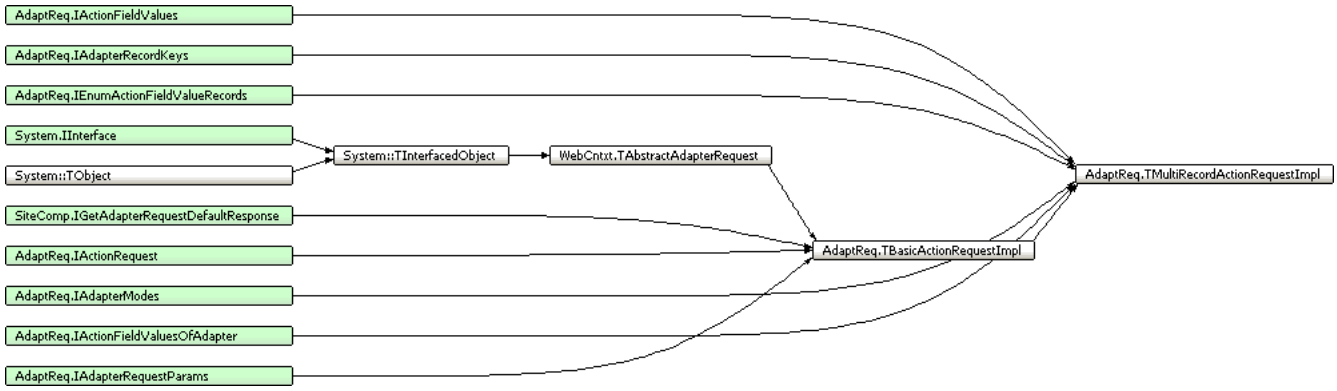

Pascal

```
TMultiRecordActionRequestImpl = class(TBasicActionRequestImpl, IAdapterModes,
IActionFieldValues, IEnumActionFieldValueRecords, IAdapterRecordKeys,
IActionFieldValuesOfAdapter);
```

C++

```
class TMultiRecordActionRequestImpl : public TBasicActionRequestImpl, public IAdapterModes,
public IActionFieldValues, public IEnumActionFieldValueRecords, public IAdapterRecordKeys,
public IActionFieldValuesOfAdapter;
```

File

AdaptReq

Description

This is class AdaptReq.TMultiRecordActionRequestImpl.

7.1.12.1 TMultiRecordActionRequestImpl Members

The following tables list the members exposed by TMultiRecordActionRequestImpl.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetActionName (see page 887)	Returns the value of the ActionName property.
	GetFailurePage (see page 887)	This is GetFailurePage, a member of class TBasicActionRequestImpl.
	GetHandled (see page 888)	Returns the value of the Handled property.
	GetSuccessPage (see page 888)	This is GetSuccessPage, a member of class TBasicActionRequestImpl.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetHandled (see page 888)	Sets the value of the Handled property.

Protected Properties

	Name	Description
R	ActionFieldValues (see page 913)	This is ActionFieldValues, a member of class TMultiRecordActionRequestImpl.
R	ActionFieldValues2 (see page 914)	This is ActionFieldValues2, a member of class TMultiRecordActionRequestImpl.
R	ActionFieldValuesList (see page 914)	This is ActionFieldValuesList, a member of class TMultiRecordActionRequestImpl.
R	AdapterActionFieldValuesList (see page 914)	This is AdapterActionFieldValuesList, a member of class TMultiRecordActionRequestImpl.
R	AdapterModes (see page 914)	Lists the modes for the adapters that the AdapterNames property lists.
R	Request (see page 24642)	This is Request, a member of class TAbstractAdapterRequest.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddValue (see page 927)	Adds a new field to the list.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AssignTo (see page 954)	Fills a string list to contain the name/value pairs from the HTTP request message.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

	Create (see page 912)	This is Create, a member of class TMultiRecordActionRequestImpl.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 912)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 959)	Cleans up after iterating over the records in the request.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAdapterMode (see page 946)	Retrieves the mode for a specified adapter.
	FindPopulatedRecord (see page 959)	Detects whether any of the records of the request contain data.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetActionName (see page 937)	Returns the value of the ActionName (see page 938) property.
	GetAdapterCount (see page 946)	Returns the value of the AdapterCount (see page 947) property.
	GetAdapterMode (see page 947)	Returns the value of the AdapterModes (see page 948) property.
	GetAdapterName (see page 947)	Returns the value of the AdapterNames (see page 948) property.
	GetEOF (see page 960)	Returns the value of the EOF property.
	GetFailurePage (see page 19693)	This is GetFailurePage, a member of class IGetAdapterRequestDefaultResponse.
	GetFieldCount (see page 927)	Returns the value of the FieldCount (see page 930) property.
	GetFieldName (see page 928)	Returns the value of the FieldNames (see page 931) property.
	GetFirstRecord (see page 960)	Positions the request on the first record.
	GetHandled (see page 937)	Returns the value of the Handled (see page 938) property.
	GetInIterator (see page 960)	Returns the value of the InIterator (see page 962) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNextRecord (see page 961)	Repositions the request on the next record.
	GetOrigValue (see page 928)	Returns the value of the OrigValues (see page 931) property.
	GetParamCount (see page 954)	Returns the value of the ParamCount (see page 956) property.
	GetParamName (see page 954)	Returns the value of the ParamNames (see page 956) property.
	GetParamValue (see page 955)	Returns the value of the ParamValues (see page 957) property.
	GetRecord (see page 961)	Positions the request on a specified record.
	GetRecordCount (see page 962)	Returns the value of the RecordCount (see page 963) property.
	GetRecordKeyCount (see page 950)	Returns the number of keys stored in the HTTP request.
	GetRecordKeyName (see page 950)	Returns the value of the RecordKeyNames (see page 951) property.
	GetRecordKeyValue (see page 951)	Returns the value of the RecordKeyValues (see page 952) property.
	GetSuccessPage (see page 19693)	This is GetSuccessPage, a member of class IGetAdapterRequestDefaultResponse.
	GetValue (see page 928)	Returns the value of the Values (see page 932) property.
	IndexOfField (see page 929)	Returns the index of a specified field.
	IndexOfParam (see page 955)	Returns the index of a specified name/value pair given its name.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LocateActionFieldValuesOfAdapter (see page 934)	Indicates whether the request includes field values for a specified adapter.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	OrigValueOfField (see page 929)	Returns the original value of a specified field.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetHandled (see page 937)	Sets the value of the Handled (see page 938) property.

	ValueOfField (see page 930)	Returns the current value for a specified field.
	ValueOfParam (see page 955)	Returns the value associated with a specified name.

Public Properties

	Name	Description
R	ActionName (see page 938)	Identifies the action to server-side scripts.
R	ActionName (see page 889)	Identifies the action to server-side scripts.
R	ActionParams (see page 889)	This is ActionParams, a member of class TBasicActionRequestImpl.
R	AdapterCount (see page 947)	Indicates the number of adapters whose modes are stored in the current request.
R	AdapterNames (see page 948)	Lists the names of the adapters for which the HTTP request contains mode information.
R	DispatchParams (see page 889)	This is DispatchParams, a member of class TBasicActionRequestImpl.
R	Eof (see page 962)	This is Eof, a member of class IEnumActionFieldValueRecords.
R	FailurePage (see page 889)	This is FailurePage, a member of class TBasicActionRequestImpl.
R	FailurePage (see page 19694)	This is FailurePage, a member of class IGetAdapterRequestDefaultResponse.
R	FieldCount (see page 930)	Indicates the number of field values stored in the current request.
R	FieldNames (see page 931)	Lists the names of the fields whose values are stored in the current request.
	Handled (see page 938)	Indicates whether the request has been handled.
R	InIterator (see page 962)	Indicates whether the request is explicitly positioned on an adapter record.
R	OrigValues (see page 931)	Lists the values of the fields in this request as they were first presented to the end user.
R	ParamCount (see page 956)	Indicates the number of name/value pairs that can be accessed using this interface.
R	ParamNames (see page 956)	Lists the names of the name/value pairs that can be accessed using this interface.
R	ParamValues (see page 957)	Lists the values of the name/value pairs that can be accessed using this interface.
R	RecordCount (see page 963)	Indicates the number of records of information the request contains.
R	RecordKeyNames (see page 951)	Lists the names of the keys stored in the HTTP request.
R	RecordKeyValues (see page 952)	Lists the key values stored in the HTTP request.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	SuccessPage (see page 890)	This is SuccessPage, a member of class TBasicActionRequestImpl.
R	SuccessPage (see page 19694)	This is SuccessPage, a member of class IGetAdapterRequestDefaultResponse.
R	Values (see page 932)	Lists the values of the fields in this request as they appeared when the user sent the request.

7.1.12.2 TMultiRecordActionRequestImpl.Create Constructor

Pascal

```
constructor Create(ADispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall TMultiRecordActionRequestImpl(IAdapterDispatchParams ADispatchParams);
```

Description

This is Create, a member of class TMultiRecordActionRequestImpl.

7.1.12.3 TMultiRecordActionRequestImpl.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMultiRecordActionRequestImpl();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

7.1.12.4 TMultiRecordActionRequestImpl Properties

The properties of the TMultiRecordActionRequestImpl class are listed here.

Protected Properties

	Name	Description
R	ActionFieldValues (see page 913)	This is ActionFieldValues, a member of class TMultiRecordActionRequestImpl.
R	ActionFieldValues2 (see page 914)	This is ActionFieldValues2, a member of class TMultiRecordActionRequestImpl.
R	ActionFieldValuesList (see page 914)	This is ActionFieldValuesList, a member of class TMultiRecordActionRequestImpl.
R	AdapterActionFieldValuesList (see page 914)	This is AdapterActionFieldValuesList, a member of class TMultiRecordActionRequestImpl.
R	AdapterModes (see page 914)	Lists the modes for the adapters that the AdapterNames property lists.

7.1.12.4.1 TMultiRecordActionRequestImpl.ActionFieldValues Property

Pascal

```
property ActionFieldValues: TActionFieldValuesImpl;
```

C++

```
__property TActionFieldValuesImpl ActionFieldValues;
```

Description

This is ActionFieldValues, a member of class TMultiRecordActionRequestImpl.

7.1.12.4.2 TMultiRecordActionRequestImpl.ActionFieldValues2 Property

Pascal

```
property ActionFieldValues2: TActionFieldValuesImpl;
```

C++

```
__property TActionFieldValuesImpl ActionFieldValues2;
```

Description

This is ActionFieldValues2, a member of class TMultiRecordActionRequestImpl.

7.1.12.4.3 TMultiRecordActionRequestImpl.ActionFieldValuesList Property

Pascal

```
property ActionFieldValuesList: TEnumActionFieldValueRecordsImpl;
```

C++

```
__property TEnumActionFieldValueRecordsImpl ActionFieldValuesList;
```

Description

This is ActionFieldValuesList, a member of class TMultiRecordActionRequestImpl.

7.1.12.4.4 TMultiRecordActionRequestImpl.AdapterActionFieldValuesList Property

Pascal

```
property AdapterActionFieldValuesList: TAdapterActionFieldValuesList;
```

C++

```
__property TAdapterActionFieldValuesList AdapterActionFieldValuesList;
```

Description

This is AdapterActionFieldValuesList, a member of class TMultiRecordActionRequestImpl.

7.1.12.4.5 TMultiRecordActionRequestImpl.AdapterModes Property

Lists the modes for the adapters that the AdapterNames property lists.

Pascal

```
property AdapterModes: TAdapterModes;
```

C++

```
__property TAdapterModes AdapterModes;
```

Description

Read AdapterModes to extract an adapter mode from the HTTP request. The AdapterNames property gives the corresponding adapter to which the mode should be applied.

I is the index of the adapter, where 0 indicates the first adapter, 1 indicates the second adapter, and so on. The AdapterCount property provides an upper bound for this index.

Use AdapterModes when iterating over all the modes in the request. To obtain the mode for a specific adapter, use the

FindAdapterMode method instead.

See Also

AdapterCount (see page 947)

AdapterNames (see page 948)

FindAdapterMode (see page 946)

7.1.13 TSingleRecordActionRequestImpl Class

Class Hierarchy

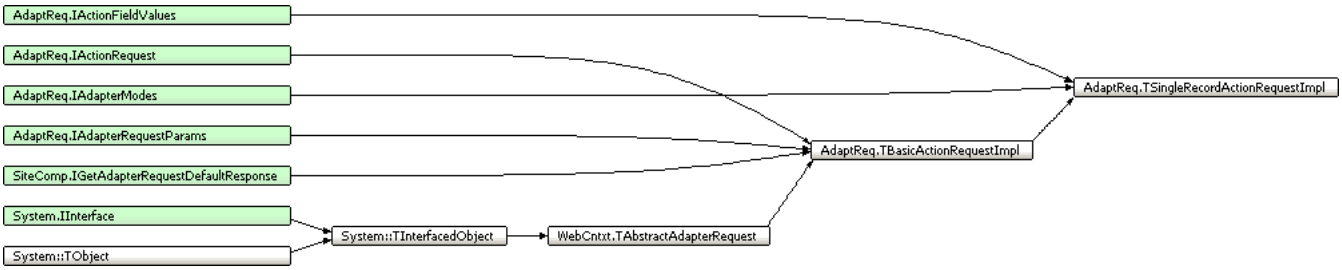

Pascal

```
TSingleRecordActionRequestImpl = class(TBasicActionRequestImpl, IAdapterModes,
IActionFieldValues);
```

C++

```
class TSingleRecordActionRequestImpl : public TBasicActionRequestImpl, public IAdapterModes,
public IActionFieldValues;
```

File

AdaptReq

Description

This is class AdaptReq.TSingleRecordActionRequestImpl.

7.1.13.1 TSingleRecordActionRequestImpl Members

The following tables list the members exposed by TSingleRecordActionRequestImpl.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetActionName (see page 887)	Returns the value of the ActionName property.
	GetFailurePage (see page 887)	This is GetFailurePage, a member of class TBasicActionRequestImpl.

	GetHandled (see page 888)	Returns the value of the Handled property.
	GetSuccessPage (see page 888)	This is GetSuccessPage, a member of class TBasicActionRequestImpl.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetHandled (see page 888)	Sets the value of the Handled property.

Protected Properties

	Name	Description
R	ActionFieldValues (see page 918)	This is ActionFieldValues, a member of class TSingleRecordActionRequestImpl.
R	AdapterModes (see page 919)	Lists the modes for the adapters that the AdapterNames property lists.
R	Request (see page 24642)	This is Request, a member of class TAbstractAdapterRequest.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddValue (see page 927)	Adds a new field to the list.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AssignTo (see page 954)	Fills a string list to contain the name/value pairs from the HTTP request message.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 917)	This is Create, a member of class TSingleRecordActionRequestImpl.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 918)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAdapterMode (see page 946)	Retrieves the mode for a specified adapter.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetActionName (see page 937)	Returns the value of the ActionName (see page 938) property.
	GetAdapterCount (see page 946)	Returns the value of the AdapterCount (see page 947) property.
	GetAdapterMode (see page 947)	Returns the value of the AdapterModes (see page 948) property.
	GetAdapterName (see page 947)	Returns the value of the AdapterNames (see page 948) property.
	GetFailurePage (see page 19693)	This is GetFailurePage, a member of class IGetAdapterRequestDefaultResponse.
	GetFieldCount (see page 927)	Returns the value of the FieldCount (see page 930) property.
	GetFieldName (see page 928)	Returns the value of the FieldNames (see page 931) property.
	GetHandled (see page 937)	Returns the value of the Handled (see page 938) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetOrigValue (see page 928)	Returns the value of the OrigValues (see page 931) property.
	GetParamCount (see page 954)	Returns the value of the ParamCount (see page 956) property.
	GetParamName (see page 954)	Returns the value of the ParamNames (see page 956) property.
	GetParamValue (see page 955)	Returns the value of the ParamValues (see page 957) property.

	GetSuccessPage (see page 19693)	This is GetSuccessPage, a member of class IGetAdapterRequestDefaultResponse.
	GetValue (see page 928)	Returns the value of the Values (see page 932) property.
	IndexOfField (see page 929)	Returns the index of a specified field.
	IndexOfParam (see page 955)	Returns the index of a specified name/value pair given its name.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	OrigValueOfField (see page 929)	Returns the original value of a specified field.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetHandled (see page 937)	Sets the value of the Handled (see page 938) property.
	ValueOfField (see page 930)	Returns the current value for a specified field.
	ValueOfParam (see page 955)	Returns the value associated with a specified name.

Public Properties

	Name	Description
R	ActionName (see page 938)	Identifies the action to server-side scripts.
R	ActionName (see page 889)	Identifies the action to server-side scripts.
R	ActionParams (see page 889)	This is ActionParams, a member of class TBasicActionRequestImpl.
R	AdapterCount (see page 947)	Indicates the number of adapters whose modes are stored in the current request.
R	AdapterNames (see page 948)	Lists the names of the adapters for which the HTTP request contains mode information.
R	DispatchParams (see page 889)	This is DispatchParams, a member of class TBasicActionRequestImpl.
R	FailurePage (see page 889)	This is FailurePage, a member of class TBasicActionRequestImpl.
R	FailurePage (see page 19694)	This is FailurePage, a member of class IGetAdapterRequestDefaultResponse.
R	FieldCount (see page 930)	Indicates the number of field values stored in the current request.
R	FieldNames (see page 931)	Lists the names of the fields whose values are stored in the current request.
	Handled (see page 938)	Indicates whether the request has been handled.
R	OrigValues (see page 931)	Lists the values of the fields in this request as they were first presented to the end user.
R	ParamCount (see page 956)	Indicates the number of name/value pairs that can be accessed using this interface.
R	ParamNames (see page 956)	Lists the names of the name/value pairs that can be accessed using this interface.
R	ParamValues (see page 957)	Lists the values of the name/value pairs that can be accessed using this interface.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	SuccessPage (see page 890)	This is SuccessPage, a member of class TBasicActionRequestImpl.
R	SuccessPage (see page 19694)	This is SuccessPage, a member of class IGetAdapterRequestDefaultResponse.
R	Values (see page 932)	Lists the values of the fields in this request as they appeared when the user sent the request.

7.1.13.2 TSingleRecordActionRequestImpl.Create Constructor

Pascal

```
constructor Create(ADispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall TSingleRecordActionRequestImpl(IAdapterDispatchParams ADispatchParams);
```

Description

This is Create, a member of class TSingleRecordActionRequestImpl.

7.1.13.3 TSingleRecordActionRequestImpl.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSingleRecordActionRequestImpl();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

7.1.13.4 TSingleRecordActionRequestImpl Properties

The properties of the TSingleRecordActionRequestImpl class are listed here.

Protected Properties

	Name	Description
R	ActionFieldValues (see page 918)	This is ActionFieldValues, a member of class TSingleRecordActionRequestImpl.
R	AdapterModes (see page 919)	Lists the modes for the adapters that the AdapterNames property lists.

7.1.13.4.1 TSingleRecordActionRequestImpl.ActionFieldValues Property

Pascal

```
property ActionFieldValues: TActionFieldValuesImpl;
```

C++

```
__property TActionFieldValuesImpl ActionFieldValues;
```

Description

This is ActionFieldValues, a member of class TSingleRecordActionRequestImpl.

7.1.13.4.2 TSingleRecordActionRequestImpl.AdapterModes Property

Lists the modes for the adapters that the AdapterNames property lists.

Pascal

```
property AdapterModes: TAdapterModes;
```

C++

```
__property TAdapterModes AdapterModes;
```

Description

Read AdapterModes to extract an adapter mode from the HTTP request. The AdapterNames property gives the corresponding adapter to which the mode should be applied.

I is the index of the adapter, where 0 indicates the first adapter, 1 indicates the second adapter, and so on. The AdapterCount property provides an upper bound for this index.

Use AdapterModes when iterating over all the modes in the request. To obtain the mode for a specific adapter, use the FindAdapterMode method instead.

See Also

AdapterCount (see page 947)

AdapterNames (see page 948)

FindAdapterMode (see page 946)

7.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IActionFieldValue (see page 920)	IActionFieldValue is the interface for obtaining the value of an adapter field.
	IActionFieldValues (see page 925)	IActionFieldValues is the interface for extracting field values from the current HTTP request.
	IActionFieldValuesOfAdapter (see page 933)	IActionFieldValuesOfAdapter is the interface for determining whether a request includes field values that pertain to a specific adapter.
	IActionHandleResponse (see page 934)	This is class AdaptReq.IActionHandleResponse.
	IActionRequest (see page 935)	IActionRequest is the interface for interacting with action requests.
	IActionRespondWith (see page 938)	This is class AdaptReq.IActionRespondWith.
	IActionResponse (see page 941)	IActionResponse is the interface for interacting with action responses.
	IAdapterModes (see page 944)	IAdapterModes is the interface for extracting the mode of an adapter from an HTTP request.
	IAdapterRecordKeys (see page 949)	IAdapterRecordKeys is the interface for extracting name/value pairs from an HTTP request that identify a record for an adapter.
	IAdapterRequestParams (see page 952)	IAdapterRequestParams is the interface for extracting name/value pairs from an HTTP request message.
	IEnumActionFieldValueRecords (see page 957)	IEnumActionFieldValueRecords is the interface for iterating over the data in an HTTP request for each of several adapter records.
	IGetAdapterActionRedirectOptions (see page 963)	This is class AdaptReq.IGetAdapterActionRedirectOptions.
	IImageRequest (see page 965)	IImageRequest is the interface for extracting the name of a requested image from an HTTP request message.
	IImageResponse (see page 967)	This is class AdaptReq.IImageResponse.

7.2.1 IActionFieldValue Interface

IActionFieldValue is the interface for obtaining the value of an adapter field.

Class Hierarchy

AdaptReq.IActionFieldValue

Pascal

```
IActionFieldValue = interface;
```

C++

```
__interface IActionFieldValue;
```

File

AdaptReq

Description

Use IActionFieldValue to read the value of an adapter field. The properties on this interface provide information about the value or values of an adapter field. These include

The FieldName (see page 923) property, which gives the name of the adapter field whose value you can read.

The FileCount (see page 924) and Files (see page 924) properties, which let you read one or more file references when the

value of the field is a file reference (for example, on an image field).

The ValueCount (see page 925) and Values (see page 925) properties, which let you read one or more values when the value of the field can be expressed as a Variant.

IActionFieldValue can provide access to multiple values when the adapter field is associated with a list of values. For example, when a list box in an HTML form represents an adapter field, IActionFieldValue provides access to every choice in that list box. When IActionFieldValue properties list several values for a field, the first value (the one with index 0) is the selected value (the one assigned to the field).

Note: Method declarations that use IActionFieldValue use the _di_IActionFieldValue type instead. This type is a DelphiInterface wrapper around the IActionFieldValue interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IActionFieldValue >
_di_IActionFieldValue;
```

See Also

TAdapterNamedField (see page 23925)

TAdapterNamedField (see page 23925)

IActionFieldValues (see page 925)

7.2.1.1 IActionFieldValue Members

The following tables list the members exposed by IActionFieldValue.

Public Methods

	Name	Description
	GetFieldName (see page 922)	Returns the value of the FieldName (see page 923) property.
	GetFile (see page 922)	Returns the value of the Files (see page 924) property.
	GetFileCount (see page 922)	Returns the value of the FileCount (see page 924) property.
	GetValue (see page 923)	Returns the value of the Values (see page 925) property.
	GetValueCount (see page 923)	Returns the value of the ValueCount (see page 925) property.

Public Properties

	Name	Description
R	FieldName (see page 923)	Indicates the name of the adapter field whose value this interface represents.
R	FileCount (see page 924)	Indicates the number of file references that the Files (see page 924) property lists.
R	Files (see page 924)	Lists the file references that make up this adapter field's value.
R	ValueCount (see page 925)	Indicates the number of values that the Values (see page 925) property lists.
R	Values (see page 925)	Lists the values that belong to the adapter field.

7.2.1.2 IActionFieldValue Methods

The methods of the IActionFieldValue class are listed here.

Public Methods

	Name	Description
	GetFieldName (see page 922)	Returns the value of the FieldName (see page 923) property.
	GetFile (see page 922)	Returns the value of the Files (see page 924) property.
	GetFileCount (see page 922)	Returns the value of the FileCount (see page 924) property.

	GetValue (see page 923)	Returns the value of the Values (see page 925) property.
	GetValueCount (see page 923)	Returns the value of the ValueCount (see page 925) property.

7.2.1.2.1 IActionFieldValue.GetFieldName Method

Returns the value of the FieldName (see page 923) property.

Pascal

```
function GetFieldName: string;
```

C++

```
__fastcall AnsiString GetFieldName();
```

Description

GetFieldName is the read implementation of the FieldName (see page 923) property.

See Also

FieldName (see page 923)

7.2.1.2.2 IActionFieldValue.GetFile Method

Returns the value of the Files (see page 924) property.

Pascal

```
function GetFile(I: Integer): TAbstractWebRequestFile;
```

C++

```
__fastcall TAbstractWebRequestFile GetFile(int I);
```

Description

GetFile is the read implementation of the Files (see page 924) property.

I is the index of the file reference to read.

See Also

Files (see page 924)

7.2.1.2.3 IActionFieldValue.GetFileCount Method

Returns the value of the FileCount (see page 924) property.

Pascal

```
function GetFileCount: Integer;
```

C++

```
__fastcall int GetFileCount();
```

Description

GetFileCount is the read implementation of the FileCount (see page 924) property.

See Also

FileCount (see page 924)

7.2.1.2.4 IActionFieldValue.GetValue Method

Returns the value of the Values (see page 925) property.

Pascal

```
function GetValue(I: Integer): Variant;
```

C++

```
__fastcall Variant GetValue(int I);
```

Description

GetValue is the read implementation of the Values (see page 925) property.

I is the index of the field value to read.

See Also

Values (see page 925)

7.2.1.2.5 IActionFieldValue.GetValueCount Method

Returns the value of the ValueCount (see page 925) property.

Pascal

```
function GetValueCount: Integer;
```

C++

```
__fastcall int GetValueCount();
```

Description

GetValueCount is the read implementation of the ValueCount (see page 925) property.

See Also

ValueCount (see page 925)

7.2.1.3 IActionFieldValue Properties

The properties of the IActionFieldValue class are listed here.

Public Properties

	Name	Description
R	FieldName (see page 923)	Indicates the name of the adapter field whose value this interface represents.
R	FileCount (see page 924)	Indicates the number of file references that the Files (see page 924) property lists.
R	Files (see page 924)	Lists the file references that make up this adapter field's value.
R	ValueCount (see page 925)	Indicates the number of values that the Values (see page 925) property lists.
R	Values (see page 925)	Lists the values that belong to the adapter field.

7.2.1.3.1 IActionFieldValue.FieldName Property

Indicates the name of the adapter field whose value this interface represents.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;
```

Description

Read FieldName to determine the name of the adapter field. This corresponds to the FieldName property of the adapter field component.

See Also

FieldName (see page 23944)

7.2.1.3.2 IActionFieldValue.FileCount Property

Indicates the number of file references that the Files (see page 924) property lists.

Pascal

```
property FileCount: Integer;
```

C++

```
__property int FileCount;
```

Description

Read FileCount to determine how many file references are assigned to the adapter field. FileCount provides an upper bound to the index on the Files (see page 924) property.

When FileCount is zero, there is no file reference assigned to the adapter field. This can be because the field's value is not a file reference, or because the field's value is not assigned.

When FileCount is greater than 1, the adapter field is associated with multiple field references. In this case, the first reference (the one with index 0) is the selected value (the one assigned to the field).

See Also

Files (see page 924)

7.2.1.3.3 IActionFieldValue.Files Property

Lists the file references that make up this adapter field's value.

Pascal

```
property Files [I: Integer]: TAbstractWebRequestFile;
```

C++

```
__property TAbstractWebRequestFile Files[int I];
```

Description

Read Files to access the content of a file that is the value of the adapter field. Files is an array of TAbstractWebRequestFile objects, where each object describes the content of a file and provides a stream from which to read the file's value.

I is the index of a file, where 0 indicates the first file, 1 indicates the second file, and so on. The FileCount (see page 924) property provides an upper bound on the value of I.

Note: If Files lists more than one file, the first file is the selected value of the adapter field (the one assigned to the field). Additional files indicate the other possible field values.

See Also

FileCount (see page 924)

Values (see page 925)

7.2.1.3.4 IActionFieldValue.ValueCount Property

Indicates the number of values that the Values (see page 925) property lists.

Pascal

```
property ValueCount: Integer;
```

C++

```
__property int ValueCount;
```

Description

Read ValueCount to determine how many values are assigned to the adapter field. ValueCount provides an upper bound to the index on the Values (see page 925) property.

When ValueCount is zero, there is no value assigned to the adapter field. This can be because the field's value is a file reference, or because the field's value is not assigned.

When ValueCount is greater than 1, the adapter field is associated with multiple values. In this case, the first reference (the one with index 0) is the selected value (the one assigned to the field).

See Also

Values (see page 925)

7.2.1.3.5 IActionFieldValue.Values Property

Lists the values that belong to the adapter field.

Pascal

```
property Values [I: Integer]: Variant;
```

C++

```
__property Variant Values[int I];
```

Description

Read Values to access the value of the adapter field. Values is an array of Variants (see page 23567), where each Variant describes a field value.

I is the index of a value, where 0 indicates the first value, 1 indicates the second value, and so on. The ValueCount (see page 925) property provides an upper bound on the value of I.

Note: If Values lists more than one value, the first value is the selected value of the adapter field (the one assigned to the field). Additional values indicate the other possible field values.

See Also

ValueCount (see page 925)

Files (see page 924)

7.2.2 IActionFieldValues Interface

IActionFieldValues is the interface for extracting field values from the current HTTP request.

Class Hierarchy

AdaptReq.IActionFieldValues

Pascal

```
IActionFieldValues = interface;
```

C++

```
__interface IActionFieldValues;
```

File

AdaptReq

Description

Use IActionFieldValues to obtain information about adapter field values that are stored in an HTTP request. You can obtain an IActionFieldValues interface by casting (Delphi) or calling QueryInterface on (C++) the interface that is the AdapterRequest property of the global Web context.

IActionFieldValues lists the names of the fields that were represented in the HTML form from which the current request originated, along with their values. It also lists the original values those fields held when the form was first presented to the end user.

Note: Method declarations that use IActionFieldValues use the _di_IActionFieldValues type instead. This type is a DelphiInterface wrapper around the IActionFieldValues interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IActionFieldValues >
_di_IActionFieldValues;
```

See Also

IActionRequest (see page 935)

IActionFieldValuesOfAdapter (see page 933)

IAdapterModes (see page 944)

IAdapterRecordKeys (see page 949)

IAdapterRequestParams (see page 952)

IImageRequest (see page 965)

TAbstractWebContext (see page 24644)

WebContext

7.2.2.1 IActionFieldValues Members

The following tables list the members exposed by IActionFieldValues.

Public Methods

	Name	Description
	AddValue (see page 927)	Adds a new field to the list.
	GetFieldCount (see page 927)	Returns the value of the FieldCount (see page 930) property.
	GetFieldName (see page 928)	Returns the value of the FieldNames (see page 931) property.
	GetOrigValue (see page 928)	Returns the value of the OrigValues (see page 931) property.
	GetValue (see page 928)	Returns the value of the Values (see page 932) property.

	IndexOfField (see page 929)	Returns the index of a specified field.
	OrigValueOfField (see page 929)	Returns the original value of a specified field.
	ValueOfField (see page 930)	Returns the current value for a specified field.

Public Properties

	Name	Description
R	FieldCount (see page 930)	Indicates the number of field values stored in the current request.
R	FieldNames (see page 931)	Lists the names of the fields whose values are stored in the current request.
R	OrigValues (see page 931)	Lists the values of the fields in this request as they were first presented to the end user.
R	Values (see page 932)	Lists the values of the fields in this request as they appeared when the user sent the request.

7.2.2.2 IActionFieldValues Methods

The methods of the IActionFieldValues class are listed here.

Public Methods

	Name	Description
	AddValue (see page 927)	Adds a new field to the list.
	GetFieldCount (see page 927)	Returns the value of the FieldCount (see page 930) property.
	GetFieldName (see page 928)	Returns the value of the FieldNames (see page 931) property.
	GetOrigValue (see page 928)	Returns the value of the OrigValues (see page 931) property.
	GetValue (see page 928)	Returns the value of the Values (see page 932) property.
	IndexOfField (see page 929)	Returns the index of a specified field.
	OrigValueOfField (see page 929)	Returns the original value of a specified field.
	ValueOfField (see page 930)	Returns the current value for a specified field.

7.2.2.2.1 IActionFieldValues.AddValue Method

Adds a new field to the list.

Pascal

```
function AddValue(const AName: string): IActionFieldValue;
```

C++

```
__fastcall IActionFieldValue AddValue(const AnsiString AName);
```

Description

Typically, WebSnap applications do not need to call AddValue, because the IActionFieldValues (see page 925) interface is used for reading the values that are stored in a request message, not for adding to them. You can, however, use this method to add a new field to the list so that it appears as if it arrived with the current request message.

AName is the name of the new field. This must not match any of the names currently listed by the FieldNames (see page 931) property.

AddValue returns the interface for the value of the newly added field.

See Also

FieldNames (see page 931)

7.2.2.2.2 IActionFieldValues.GetFieldCount Method

Returns the value of the FieldCount (see page 930) property.

Pascal

```
function GetFieldCount: Integer;
```

C++

```
__fastcall int GetFieldCount();
```

Description

GetFieldCount is the read implementation of the FieldCount (see page 930) property.

See Also

FieldCount (see page 930)

7.2.2.2.3 IActionFieldValues.GetFieldName Method

Returns the value of the FieldNames (see page 931) property.

Pascal

```
function GetFieldName(I: Integer): string;
```

C++

```
__fastcall AnsiString GetFieldName(int I);
```

Description

GetFieldName is the read implementation of the FieldNames (see page 931) property.

I is the index of the field name to retrieve.

See Also

FieldNames (see page 931)

7.2.2.2.4 IActionFieldValues.GetOrigValue Method

Returns the value of the OrigValues (see page 931) property.

Pascal

```
function GetOrigValue(I: Integer): IActionFieldValue;
```

C++

```
__fastcall IActionFieldValue GetOrigValue(int I);
```

Description

GetOrigValue is the read implementation of the OrigValues (see page 931) property.

I is the index of the original field value to retrieve.

See Also

OrigValues (see page 931)

7.2.2.2.5 IActionFieldValues.GetValue Method

Returns the value of the Values (see page 932) property.

Pascal

```
function GetValue(I: Integer): IActionFieldValue;
```

C++

```
__fastcall IActionFieldValue GetValue(int I);
```

Description

GetValue is the read implementation of the Values (see page 932) property.

I is the index of the field value to retrieve.

See Also

Values (see page 932)

7.2.2.2.6 IActionFieldValues.IndexOfField Method

Returns the index of a specified field.

Pascal

```
function IndexOfField(const AName: string): Integer;
```

C++

```
__fastcall int IndexOfField(const AnsiString AName);
```

Description

IndexOfField returns the index of the field with the specified name. This method performs the reverse lookup from reading the FieldNames (see page 931) property.

AName is the name of the field to locate.

IndexOfField returns the index for the specified field. This index identifies the position of the field in the FieldNames (see page 931), Values (see page 932), and OrigValues (see page 931) property arrays. If the current HTTP request does not include a field with the specified name, IndexOfField returns –1.

See Also

FieldNames (see page 931)

OrigValueOfField (see page 929)

ValueOfField (see page 930)

Values (see page 932)

OrigValues (see page 931)

7.2.2.2.7 IActionFieldValues.OrigValueOfField Method

Returns the original value of a specified field.

Pascal

```
function OrigValueOfField(const AName: string): IActionFieldValue;
```

C++

```
__fastcall IActionFieldValue OrigValueOfField(const AnsiString AName);
```

Description

OrigValueOfField returns the value of a specified field as it was when the HTML form was first presented to the end user (before the user made any changes). Use OrigValueOfField to access a field value by name rather than by index.

AName is the name of the field whose original value you want.

OrigValueOfField returns an interface for reading the original value, as it was stored as a hidden field in the HTML form. If the current HTTP request does not include a field with the specified name, or if the HTML form did not store an original value for this field as a hidden field, OrigValueOfField returns nil (Delphi) or NULL (C++).

See Also

IndexOfField (see page 929)

ValueOfField (see page 930)

OrigValues (see page 931)

7.2.2.2.8 IActionFieldValues.ValueOfField Method

Returns the current value for a specified field.

Pascal

```
function ValueOfField(const AName: string): IActionFieldValue;
```

C++

```
__fastcall IActionFieldValue ValueOfField(const AnsiString AName);
```

Description

ValueOfField returns the value that the HTTP request provides for a specified field. Use ValueOfField to access a field value by name rather than by index.

AName is the name of the field whose value you want.

ValueOfField returns an interface for reading the value, as it was after any edits the user made to the HTML form. If the current HTTP request does not include a field with the specified name, ValueOfField returns nil (Delphi) or NULL (C++).

See Also

IndexOfField (see page 929)

OrigValueOfField (see page 929)

Values (see page 932)

7.2.2.3 IActionFieldValues Properties

The properties of the IActionFieldValues class are listed here.

Public Properties

	Name	Description
R	FieldCount (see page 930)	Indicates the number of field values stored in the current request.
R	FieldNames (see page 931)	Lists the names of the fields whose values are stored in the current request.
R	OrigValues (see page 931)	Lists the values of the fields in this request as they were first presented to the end user.
R	Values (see page 932)	Lists the values of the fields in this request as they appeared when the user sent the request.

7.2.2.3.1 IActionFieldValues.FieldCount Property

Indicates the number of field values stored in the current request.

Pascal

```
property FieldCount: Integer;
```

C++

```
__property int FieldCount;
```

Description

Read FieldCount to determine how many field values you can extract from the current HTTP request. FieldCount provides an upper bound to the index on the FieldNames (see page 931), Values (see page 932), and OrigValues (see page 931) property arrays.

See Also

FieldNames (see page 931)

OrigValues (see page 931)

Values (see page 932)

7.2.2.3.2 IActionFieldValues.FieldNames Property

Lists the names of the fields whose values are stored in the current request.

Pascal

```
property FieldNames [I: Integer]: string;
```

C++

```
__property AnsiString FieldNames[int I];
```

Description

Read FieldNames to obtain the names of the fields about which the current request provides information.

I is the index of the field whose name this property identifies, where 0 indicates the first field, 1 indicates the second field, and so on. The FieldCount (see page 930) property provides an upper bound for this index.

Use FieldNames when iterating over all the fields in the request. To obtain information about a specific field, use the ValueOfField (see page 930) or OrigValueOfField (see page 929) method.

See Also

FieldCount (see page 930)

Values (see page 932)

OrigValues (see page 931)

IndexOfField (see page 929)

ValueOfField (see page 930)

OrigValueOfField (see page 929)

7.2.2.3.3 IActionFieldValues.OrigValues Property

Lists the values of the fields in this request as they were first presented to the end user.

Pascal

```
property OrigValues [I: Integer]: IActionFieldValue;
```

C++

```
__property IActionFieldValue OrigValues[int I];
```

Description

Read OrigValues to access the original value of a field. This is the value of the field when the HTML form was first presented to the end user, before the user made any changes. If the HTML form did not store original field values as hidden fields, the value of OrigValues is nil (Delphi) or NULL (C++).

I is the index of the field whose original value this property identifies, where 0 indicates the first field, 1 indicates the second field, and so on. The FieldCount (see page 930) property provides an upper bound for this index.

Tip: Use the FieldNames (see page 931) property to determine the name of the field associated with a specified index.

Use OrigValues when iterating over all the fields in the request. To obtain the original value of a specific field, use the OrigValueOfField (see page 929) method instead.

See Also

FieldCount (see page 930)

Values (see page 932)

FieldNames (see page 931)

OrigValueOfField (see page 929)

7.2.2.3.4 IActionFieldValues.Values Property

Lists the values of the fields in this request as they appeared when the user sent the request.

Pascal

```
property Values [I: Integer]: IActionFieldValue;
```

C++

```
__property IActionFieldValue Values[int I];
```

Description

Read Values to access the current value for a field. This value includes any edits that the end user may have made using the HTML form.

I is the index of the field whose value this property identifies, where 0 indicates the first field, 1 indicates the second field, and so on. The FieldCount (see page 930) property provides an upper bound for this index.

Tip: Use the FieldNames (see page 931) property to determine the name of the field associated with a specified index.

Use Values when iterating over all the fields in the request. To obtain the value for a specific field, use the ValueOfField (see page 930) method instead.

See Also

FieldCount (see page 930)

OrigValues (see page 931)

FieldNames (see page 931)

ValueOfField (see page 930)

7.2.3 IActionFieldValuesOfAdapter Interface

IActionFieldValuesOfAdapter is the interface for determining whether a request includes field values that pertain to a specific adapter.

Class Hierarchy

AdaptReq.IActionFieldValuesOfAdapter

Pascal

```
IActionFieldValuesOfAdapter = interface;
```

C++

```
__interface IActionFieldValuesOfAdapter;
```

File

AdaptReq

Description

Use IActionFieldValuesOfAdapter to obtain determine whether an HTTP request lists field values for a specific adapter component. This interface is supported by adapter request objects that represent requests originating from an HTML form that represents information from multiple adapters.

You can obtain an IActionFieldValuesOfAdapter interface by casting (Delphi) or calling QueryInterface on (C++) the interface that is the AdapterRequest property of the global Web context.

IActionFieldValuesOfAdapter defines a single method, LocateActionFieldValuesOfAdapter (see page 934), which indicates whether the request includes field values for a specified adapter. If the request does include such a list, fetching the IActionFieldValues (see page 925) interface for the request yields the list of field values for the specified adapter.

See Also

IActionRequest (see page 935)

IActionFieldValues (see page 925)

TAbstractWebContext (see page 24644)

WebContext

7.2.3.1 IActionFieldValuesOfAdapter Members

The following tables list the members exposed by IActionFieldValuesOfAdapter.

Public Methods

	Name	Description
	LocateActionFieldValuesOfAdapter (see page 934)	Indicates whether the request includes field values for a specified adapter.

7.2.3.2 IActionFieldValuesOfAdapter Methods

The methods of the IActionFieldValuesOfAdapter class are listed here.

Public Methods

	Name	Description
	LocateActionFieldValuesOfAdapter (see page 934)	Indicates whether the request includes field values for a specified adapter.

7.2.3.2.1 IActionFieldValuesOfAdapter.LocateActionFieldValuesOfAdapter Method

Indicates whether the request includes field values for a specified adapter.

Pascal

```
function LocateActionFieldValuesOfAdapter(AAdapter: TComponent): Boolean;
```

C++

```
__fastcall Boolean LocateActionFieldValuesOfAdapter(TComponent * AAdapter);
```

Description

LocateActionFieldValuesOfAdapter returns true if the request includes field values for the specified adapter, false if there are no field values for the specified adapter.

AAdapter is the name of the adapter component whose fields you want to access.

If the request contains field information for the specified adapter, LocateActionFieldValuesOfAdapter positions the request on the set of field values for AAdapter before returning true. This means that if you next cast (Delphi) or call QueryInterface on (C++) this interface to obtain an IActionFieldValues (see page 925) interface, you obtain an interface for the fields associated with the specified adapter.

See Also

IActionFieldValues (see page 925)

7.2.4 IActionHandleResponse Interface

Class Hierarchy

AdaptReq.IActionHandleResponse

Pascal

```
IActionHandleResponse = interface;
```

C++

```
__interface IActionHandleResponse;
```

File

AdaptReq

Description

This is class AdaptReq.IActionHandleResponse.

7.2.4.1 IActionHandleResponse Members

The following tables list the members exposed by IActionHandleResponse.

Public Methods

	Name	Description
	HandleResponse (see page 935)	This is HandleResponse, a member of class IActionHandleResponse.

7.2.4.2 IActionHandleResponse Methods

The methods of the IActionHandleResponse class are listed here.

Public Methods

	Name	Description
	HandleResponse (see page 935)	This is HandleResponse, a member of class IActionHandleResponse.

7.2.4.2.1 IActionHandleResponse.HandleResponse Method

Pascal

```
procedure HandleResponse;
```

C++

```
__fastcall HandleResponse();
```

Description

This is HandleResponse, a member of class IActionHandleResponse.

7.2.5 IActionRequest Interface

IActionRequest is the interface for interacting with action requests.

Class Hierarchy

AdaptReq.IActionRequest

Pascal

```
IActionRequest = interface;
```

C++

```
__interface IActionRequest;
```

File

AdaptReq

Description

IActionRequest is used by a wide variety of routines and methods to interact with action requests. Use the Supports function to determine whether a request object implements IActionRequest and to obtain the interface.

You can often cast this interface to another interface (Delphi) or use the QueryInterface method to obtain another interface from this one (C++) that lets you extract specialized information from the request. The following interfaces are often supported by the request object that implements this interface:

IActionFieldValues (see page 925)

IActionFieldValuesOfAdapter (see page 933)

IAdapterRequestParams (see page 952)

IImageRequest (see page 965)

IAdapterRecordKeys (see page 949)

IAdapterModes (see page 944)

IEnumActionFieldValueRecords (see page 957)

See Also

IActionResponse (see page 941)

Supports

TAbstractWebContext (see page 24644)

IActionFieldValues (see page 925)

IAdapterModes (see page 944)

IActionFieldValuesOfAdapter (see page 933)

IAdapterRecordKeys (see page 949)

IAdapterRequestParams (see page 952)

IImageRequest (see page 965)

IEnumActionFieldValueRecords (see page 957)

7.2.5.1 IActionRequest Members

The following tables list the members exposed by IActionRequest.

Public Methods

	Name	Description
	GetActionName (see page 937)	Returns the value of the ActionName (see page 938) property.
	GetHandled (see page 937)	Returns the value of the Handled (see page 938) property.
	SetHandled (see page 937)	Sets the value of the Handled (see page 938) property.

Public Properties

	Name	Description
R	ActionName (see page 938)	Identifies the action to server-side scripts.
	Handled (see page 938)	Indicates whether the request has been handled.

7.2.5.2 IActionRequest Methods

The methods of the IActionRequest class are listed here.

Public Methods

	Name	Description
	GetActionName (see page 937)	Returns the value of the ActionName (see page 938) property.
	GetHandled (see page 937)	Returns the value of the Handled (see page 938) property.
	SetHandled (see page 937)	Sets the value of the Handled (see page 938) property.

7.2.5.2.1 IActionRequest.GetActionName Method

Returns the value of the ActionName (see page 938) property.

Pascal

```
function GetActionName: string;
```

C++

```
__fastcall AnsiString GetActionName();
```

Description

GetActionName is the read implementation of the ActionName (see page 938) property.

See Also

ActionName (see page 938)

7.2.5.2.2 IActionRequest.GetHandled Method

Returns the value of the Handled (see page 938) property.

Pascal

```
function GetHandled: Boolean;
```

C++

```
__fastcall Boolean GetHandled();
```

Description

GetHandled is the read implementation of the Handled (see page 938) property.

See Also

Handled (see page 938)

7.2.5.2.3 IActionRequest.SetHandled Method

Sets the value of the Handled (see page 938) property.

Pascal

```
procedure SetHandled(AValue: Boolean);
```

C++

```
__fastcall SetHandled(Boolean AValue);
```

Description

SetHandled is the write implementation of the Handled (see page 938) property.

See Also

Handled (see page 938)

7.2.5.3 IActionRequest Properties

The properties of the IActionRequest class are listed here.

Public Properties

	Name	Description
	ActionName (see page 938)	Identifies the action to server-side scripts.
	Handled (see page 938)	Indicates whether the request has been handled.

7.2.5.3.1 IActionRequest.ActionName Property

Identifies the action to server-side scripts.

Pascal

```
property ActionName: string;
```

C++

```
__property AnsiString ActionName;
```

Description

The ActionName property defines the name used to access the adapter action from server-side scripts.

7.2.5.3.2 IActionRequest.Handled Property

Indicates whether the request has been handled.

Pascal

```
property Handled: Boolean;
```

C++

```
__property Boolean Handled;
```

Description

Note: WebSnap sets Handled to true when the request has been handled.

7.2.6 IActionRespondWith Interface

Class Hierarchy

AdaptReq.IActionRespondWith

Pascal

```
IActionRespondWith = interface;
```

C++

```
__interface IActionRespondWith;
```

File

AdaptReq

Description

This is class AdaptReq.IActionRespondWith.

7.2.6.1 IActionRespondWith Members

The following tables list the members exposed by IActionRespondWith.

Public Methods

	Name	Description
	GetRedirectOptions (see page 939)	This is GetRedirectOptions, a member of class IActionRespondWith.
	GetRespondWith (see page 939)	This is GetRespondWith, a member of class IActionRespondWith.
	RespondWithComponentPage (see page 940)	This is RespondWithComponentPage, a member of class IActionRespondWith.
	RespondWithPage (see page 940)	This is RespondWithPage, a member of class IActionRespondWith.
	RespondWithURL (see page 940)	This is RespondWithURL, a member of class IActionRespondWith.
	SetExecuteStatus (see page 940)	This is SetExecuteStatus, a member of class IActionRespondWith.
	SetRedirectOptions (see page 941)	This is SetRedirectOptions, a member of class IActionRespondWith.

Public Properties

	Name	Description
	RedirectOptions (see page 941)	This is RedirectOptions, a member of class IActionRespondWith.
R	RespondWith (see page 941)	This is RespondWith, a member of class IActionRespondWith.

7.2.6.2 IActionRespondWith Methods

The methods of the IActionRespondWith class are listed here.

Public Methods

	Name	Description
	GetRedirectOptions (see page 939)	This is GetRedirectOptions, a member of class IActionRespondWith.
	GetRespondWith (see page 939)	This is GetRespondWith, a member of class IActionRespondWith.
	RespondWithComponentPage (see page 940)	This is RespondWithComponentPage, a member of class IActionRespondWith.
	RespondWithPage (see page 940)	This is RespondWithPage, a member of class IActionRespondWith.
	RespondWithURL (see page 940)	This is RespondWithURL, a member of class IActionRespondWith.
	SetExecuteStatus (see page 940)	This is SetExecuteStatus, a member of class IActionRespondWith.
	SetRedirectOptions (see page 941)	This is SetRedirectOptions, a member of class IActionRespondWith.

7.2.6.2.1 IActionRespondWith.GetRedirectOptions Method

Pascal

```
function GetRedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__fastcall TAdapterActionRedirectOptions GetRedirectOptions();
```

Description

This is GetRedirectOptions, a member of class IActionRespondWith.

7.2.6.2.2 IActionRespondWith.GetRespondWith Method

Pascal

```
function GetRespondWith: TRespondWith;
```

C++

```
__fastcall TRespondWith GetRespondWith();
```

Description

This is GetRespondWith, a member of class IActionRespondWith.

7.2.6.2.3 IActionRespondWith.RespondWithComponentPage Method

Pascal

```
procedure RespondWithComponentPage(const AComponent: TComponent);
```

C++

```
__fastcall RespondWithComponentPage(const TComponent * AComponent);
```

Description

This is RespondWithComponentPage, a member of class IActionRespondWith.

7.2.6.2.4 IActionRespondWith.RespondWithPage Method

Pascal

```
procedure RespondWithPage(const APageName: string);
```

C++

```
__fastcall RespondWithPage(const AnsiString APageName);
```

Description

This is RespondWithPage, a member of class IActionRespondWith.

7.2.6.2.5 IActionRespondWith.RespondWithURL Method

Pascal

```
procedure RespondWithURL(const AURL: string);
```

C++

```
__fastcall RespondWithURL(const AnsiString AURL);
```

Description

This is RespondWithURL, a member of class IActionRespondWith.

7.2.6.2.6 IActionRespondWith.SetExecuteStatus Method

Pascal

```
procedure SetExecuteStatus(Value: TExecuteStatus);
```

C++

```
__fastcall SetExecuteStatus(TExecuteStatus Value);
```

Description

This is SetExecuteStatus, a member of class IActionRespondWith.

7.2.6.2.7 IActionRespondWith.SetRedirectOptions Method

Pascal

```
procedure SetRedirectOptions(AOptions: TAdapterActionRedirectOptions);
```

C++

```
__fastcall SetRedirectOptions(TAdapterActionRedirectOptions AOptions);
```

Description

This is SetRedirectOptions, a member of class IActionRespondWith.

7.2.6.3 IActionRespondWith Properties

The properties of the IActionRespondWith class are listed here.

Public Properties

	Name	Description
	RedirectOptions (see page 941)	This is RedirectOptions, a member of class IActionRespondWith.
R	RespondWith (see page 941)	This is RespondWith, a member of class IActionRespondWith.

7.2.6.3.1 IActionRespondWith.RedirectOptions Property

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

This is RedirectOptions, a member of class IActionRespondWith.

7.2.6.3.2 IActionRespondWith.RespondWith Property

Pascal

```
property RespondWith: TRespondWith;
```

C++

```
__property TRespondWith RespondWith;
```

Description

This is RespondWith, a member of class IActionRespondWith.

7.2.7 IActionResponse Interface

IActionResponse is the interface for interacting with action responses.

Class Hierarchy

Pascal

```
IActionResponse = interface;
```

C++

```
__interface IActionResponse;
```

File

AdaptReq

Description

IActionResponse is used by a variety of routines and methods to interact with action response. Use the Supports function to determine whether a request object implements IActionResponse and to obtain the interface.

See Also

IActionRequest (see page 935)

Supports

7.2.7.1 IActionResponse Members

The following tables list the members exposed by IActionResponse.

Public Methods

	Name	Description
	GetExecuteStatus (see page 942)	Returns the value of the ExecuteStatus (see page 944) property.
	GetHandled (see page 943)	Returns the value of the Handled (see page 944) property.
	SetExecuteStatus (see page 943)	Sets the value of the ExecuteStatus (see page 944) property.
	SetHandled (see page 943)	Sets the value of the Handled (see page 944) property.

Public Properties

	Name	Description
	ExecuteStatus (see page 944)	Indicates the status of executing the action.
	Handled (see page 944)	Indicates whether the incoming request has been handled.

7.2.7.2 IActionResponse Methods

The methods of the IActionResponse class are listed here.

Public Methods

	Name	Description
	GetExecuteStatus (see page 942)	Returns the value of the ExecuteStatus (see page 944) property.
	GetHandled (see page 943)	Returns the value of the Handled (see page 944) property.
	SetExecuteStatus (see page 943)	Sets the value of the ExecuteStatus (see page 944) property.
	SetHandled (see page 943)	Sets the value of the Handled (see page 944) property.

7.2.7.2.1 IActionResponse.GetExecuteStatus Method

Returns the value of the ExecuteStatus (see page 944) property.

Pascal

```
function GetExecuteStatus: TExecuteStatus;
```

C++

```
__fastcall TExecuteStatus GetExecuteStatus();
```

Description

GetExecuteStatus is the read implementation of the ExecuteStatus (see page 944) property.

See Also

ExecuteStatus (see page 944)

7.2.7.2.2 IActionResponse.GetHandled Method

Returns the value of the Handled (see page 944) property.

Pascal

```
function GetHandled: Boolean;
```

C++

```
__fastcall Boolean GetHandled();
```

Description

GetHandled is the read implementation of the Handled (see page 944) property.

See Also

Handled (see page 944)

7.2.7.2.3 IActionResponse.SetExecuteStatus Method

Sets the value of the ExecuteStatus (see page 944) property.

Pascal

```
procedure SetExecuteStatus(Value: TExecuteStatus);
```

C++

```
__fastcall SetExecuteStatus(TExecuteStatus Value);
```

Description

SetExecuteStatus is the write implementation of the ExecuteStatus (see page 944) property.

See Also

ExecuteStatus (see page 944)

7.2.7.2.4 IActionResponse.SetHandled Method

Sets the value of the Handled (see page 944) property.

Pascal

```
procedure SetHandled(Value: Boolean);
```

C++

```
__fastcall SetHandled(Boolean Value);
```

Description

SetHandled is the write implementation of the Handled (see page 944) property.

See Also

Handled (see page 944)

7.2.7.3 IActionResponse Properties

The properties of the IActionResponse class are listed here.

Public Properties

	Name	Description
	ExecuteStatus (see page 944)	Indicates the status of executing the action.
	Handled (see page 944)	Indicates whether the incoming request has been handled.

7.2.7.3.1 IActionResponse.ExecuteStatus Property

Indicates the status of executing the action.

Pascal

```
property ExecuteStatus: TExecuteStatus;
```

C++

```
__property TExecuteStatus ExecuteStatus;
```

Description

The ExecuteStatus property indicates whether the request was completely handled, and if so, whether the attempt to handle the request was successful.

See Also

Handled (see page 944)

7.2.7.3.2 IActionResponse.Handled Property

Indicates whether the incoming request has been handled.

Pascal

```
property Handled: Boolean;
```

C++

```
__property Boolean Handled;
```

Description

Note: WebSnap sets Handled to true when the response has been handled.

7.2.8 IAdapterModes Interface

IAdapterModes is the interface for extracting the mode of an adapter from an HTTP request.

Class Hierarchy

AdaptReq.IAdapterModes

Pascal

```
IAdapterModes = interface;
```

C++

```
__interface IAdapterModes;
```

File

AdaptReq

Description

Use IAdapterModes to obtain mode information for a specified adapter from an HTTP request message. You can obtain an IAdapterModes interface by casting (Delphi) or calling QueryInterface on (C++) the interface that is the AdapterRequest property of the global Web context.

Not all adapters have modes. For adapters that do support modes, the mode indicates the types of tasks that adapter can perform. Adapters that support modes include the mode as a hidden field in HTML forms so that they can re-establish the correct mode when processing requests that originate from that form.

IAdapterModes lists the modes of any adapters that support modes and that contributed to the HTML form from which this request originated.

See Also

IActionRequest (see page 935)

IAdapterRequestParams (see page 952)

IAdapterRecordKeys (see page 949)

IImageRequest (see page 965)

TAbstractWebContext (see page 24644)

WebContext

TCustomDataSetAdapter (see page 7766)

7.2.8.1 IAdapterModes Members

The following tables list the members exposed by IAdapterModes.

Public Methods

	Name	Description
	FindAdapterMode (see page 946)	Retrieves the mode for a specified adapter.
	GetAdapterCount (see page 946)	Returns the value of the AdapterCount (see page 947) property.
	GetAdapterMode (see page 947)	Returns the value of the AdapterModes (see page 948) property.
	GetAdapterName (see page 947)	Returns the value of the AdapterNames (see page 948) property.

Public Properties

	Name	Description
R	AdapterCount (see page 947)	Indicates the number of adapters whose modes are stored in the current request.
R	AdapterModes (see page 948)	Lists the modes for the adapters that the AdapterNames (see page 948) property lists.
R	AdapterNames (see page 948)	Lists the names of the adapters for which the HTTP request contains mode information.

7.2.8.2 IAdapterModes Methods

The methods of the IAdapterModes class are listed here.

Public Methods

	Name	Description
	FindAdapterMode (see page 946)	Retrieves the mode for a specified adapter.
	GetAdapterCount (see page 946)	Returns the value of the AdapterCount (see page 947) property.
	GetAdapterMode (see page 947)	Returns the value of the AdapterModes (see page 948) property.
	GetAdapterName (see page 947)	Returns the value of the AdapterNames (see page 948) property.

7.2.8.2.1 IAdapterModes.FindAdapterMode Method

Retrieves the mode for a specified adapter.

Pascal

```
function FindAdapterMode(AAdapter: TObject; var AMode: string): Boolean;
```

C++

```
__fastcall Boolean FindAdapterMode(TObject * AAdapter, AnsiString AMode);
```

Description

FindAdapterMode extracts the mode for a specified adapter from the HTTP request.

AAdapter is the adapter object whose mode you want to extract. The variable name for the adapter is one of the strings that the AdapterNames (see page 948) property lists.

AMode returns the mode string, if any, that the request stores for the specified adapter.

FindAdapterMode returns true if the request has a mode stored for the specified adapter. When FindAdapterMode returns true , the AMode parameter returns the mode string. FindAdapterMode returns false if the request does not include a mode for the specified adapter. When FindAdapterMode returns false, the AMode parameter is unchanged.

See Also

AdapterNames (see page 948)

AdapterModes (see page 948)

7.2.8.2.2 IAdapterModes.GetAdapterCount Method

Returns the value of the AdapterCount (see page 947) property.

Pascal

```
function GetAdapterCount: Integer;
```

C++

```
__fastcall int GetAdapterCount();
```

Description

GetAdapterCount is the read implementation of the AdapterCount (see page 947) property.

See Also

AdapterCount (see page 947)

7.2.8.2.3 IAdapterModes.GetAdapterMode Method

Returns the value of the AdapterModes (see page 948) property.

Pascal

```
function GetAdapterMode(I: Integer): string;
```

C++

```
__fastcall AnsiString GetAdapterMode(int I);
```

Description

GetAdapterMode is the read implementation of the AdapterModes (see page 948) property.

I is the index of the adapter whose mode this method returns.

See Also

AdapterModes (see page 948)

7.2.8.2.4 IAdapterModes.GetAdapterName Method

Returns the value of the AdapterNames (see page 948) property.

Pascal

```
function GetAdapterName(I: Integer): string;
```

C++

```
__fastcall AnsiString GetAdapterName(int I);
```

Description

GetAdapterName is the read implementation of the AdapterNames (see page 948) property.

I is the index of the adapter whose name this method returns.

See Also

AdapterNames (see page 948)

7.2.8.3 IAdapterModes Properties

The properties of the IAdapterModes class are listed here.

Public Properties

	Name	Description
R	AdapterCount (see page 947)	Indicates the number of adapters whose modes are stored in the current request.
R	AdapterModes (see page 948)	Lists the modes for the adapters that the AdapterNames (see page 948) property lists.
R	AdapterNames (see page 948)	Lists the names of the adapters for which the HTTP request contains mode information.

7.2.8.3.1 IAdapterModes.AdapterCount Property

Indicates the number of adapters whose modes are stored in the current request.

Pascal

```
property AdapterCount: Integer;
```

C++

```
__property int AdapterCount;
```

Description

Read AdapterCount to determine how many adapter modes you can extract from the current HTTP request. AdapterCount provides an upper bound to the index on the AdapterNames (see page 948) and AdapterModes (see page 948) property arrays.

See Also

AdapterNames (see page 948)

AdapterModes (see page 948)

7.2.8.3.2 IAdapterModes.AdapterModes Property

Lists the modes for the adapters that the AdapterNames (see page 948) property lists.

Pascal

```
property AdapterModes [I: Integer]: string;
```

C++

```
__property AnsiString AdapterModes[int I];
```

Description

Read AdapterModes to extract an adapter mode from the HTTP request. The AdapterNames (see page 948) property gives the corresponding adapter to which the mode should be applied.

I is the index of the adapter, where 0 indicates the first adapter, 1 indicates the second adapter, and so on. The AdapterCount (see page 947) property provides an upper bound for this index.

Use AdapterModes when iterating over all the modes in the request. To obtain the mode for a specific adapter, use the FindAdapterMode (see page 946) method instead.

See Also

AdapterCount (see page 947)

AdapterNames (see page 948)

FindAdapterMode (see page 946)

7.2.8.3.3 IAdapterModes.AdapterNames Property

Lists the names of the adapters for which the HTTP request contains mode information.

Pascal

```
property AdapterNames [I: Integer]: string;
```

C++

```
__property AnsiString AdapterNames[int I];
```

Description

Read AdapterNames to obtain the names of the adapters that have mode information included in the HTTP request. Each

adapter name is the name of the variable that refers to the adapter in server-side script.

I is the index of the adapter, where 0 indicates the first adapter, 1 indicates the second adapter, and so on. The AdapterCount (see page 947) property provides an upper bound for this index.

Use AdapterNames when iterating over all the adapters in the request. To obtain the mode of a specified Adapter given its name, use the FindAdapterMode (see page 946) method.

See Also

AdapterCount (see page 947)

AdapterModes (see page 948)

FindAdapterMode (see page 946)

7.2.9 IAdapterRecordKeys Interface

IAdapterRecordKeys is the interface for extracting name/value pairs from an HTTP request that identify a record for an adapter.

Class Hierarchy

AdaptReq.IAdapterRecordKeys

Pascal

```
IAdapterRecordKeys = interface;
```

C++

```
__interface IAdapterRecordKeys;
```

File

AdaptReq

Description

Use IAdapterRecordKeys to obtain key values from an HTTP request message. You can obtain an IAdapterRecordKeys interface by casting (Delphi) or calling QueryInterface on (C++)the interface that is the AdapterRequest property of the global Web context.

Some adapters use key values to define the relationship between values in the HTTP request and adapter data values. For example, TDataSetAdapter identifies dataset records using key field values. The adapter adds these key values to HTML forms as hidden fields. Requests that originate from those forms include the key values so that the adapter can locate the record to which a request applies.

IAdapterRecordKeys lists the names and values of any keys an adapter has added to the HTML form.

See Also

IActionRequest (see page 935)

IAdapterRequestParams (see page 952)

IAdapterModes (see page 944)

IImageRequest (see page 965)

TAbstractWebContext (see page 24644)

WebContext

IEnumActionFieldValueRecords (see page 957)

IKeyParams (see page 8105)

TCustomAdapter

7.2.9.1 IAdapterRecordKeys Members

The following tables list the members exposed by IAdapterRecordKeys.

Public Methods

	Name	Description
	GetRecordKeyCount (see page 950)	Returns the number of keys stored in the HTTP request.
	GetRecordKeyName (see page 950)	Returns the value of the RecordKeyNames (see page 951) property.
	GetRecordKeyValue (see page 951)	Returns the value of the RecordKeyValues (see page 952) property.

Public Properties

	Name	Description
R	RecordKeyNames (see page 951)	Lists the names of the keys stored in the HTTP request.
R	RecordKeyValues (see page 952)	Lists the key values stored in the HTTP request.

7.2.9.2 IAdapterRecordKeys Methods

The methods of the IAdapterRecordKeys class are listed here.

Public Methods

	Name	Description
	GetRecordKeyCount (see page 950)	Returns the number of keys stored in the HTTP request.
	GetRecordKeyName (see page 950)	Returns the value of the RecordKeyNames (see page 951) property.
	GetRecordKeyValue (see page 951)	Returns the value of the RecordKeyValues (see page 952) property.

7.2.9.2.1 IAdapterRecordKeys.GetRecordKeyCount Method

Returns the number of keys stored in the HTTP request.

Pascal

```
function GetRecordKeyCount: Integer;
```

C++

```
__fastcall int GetRecordKeyCount();
```

Description

GetRecordKeyCount returns the number of keys this interface makes available. This value provides an upper bound to the index for the RecordKeyValues (see page 952) and RecordKeyNames (see page 951) property arrays.

See Also

RecordKeyValues (see page 952)

RecordKeyNames (see page 951)

7.2.9.2.2 IAdapterRecordKeys.GetRecordKeyName Method

Returns the value of the RecordKeyNames (see page 951) property.

Pascal

```
function GetRecordKeyName(I: Integer): string;
```

C++

```
__fastcall AnsiString GetRecordKeyName(int I);
```

Description

GetRecordKeyName is the read implementation of the RecordKeyNames (see page 951) property.

I is the index of the key whose name this method returns.

See Also

RecordKeyNames (see page 951)

7.2.9.2.3 IAdapterRecordKeys.GetRecordKeyValue Method

Returns the value of the RecordKeyValues (see page 952) property.

Pascal

```
function GetRecordKeyValue(I: Integer): string;
```

C++

```
__fastcall AnsiString GetRecordKeyValue(int I);
```

Description

GetRecordKeyValue is the read implementation of the RecordKeyValues (see page 952) property.

I is the index of the key whose value this method returns.

See Also

RecordKeyValues (see page 952)

7.2.9.3 IAdapterRecordKeys Properties

The properties of the IAdapterRecordKeys class are listed here.

Public Properties

	Name	Description
R	RecordKeyNames (see page 951)	Lists the names of the keys stored in the HTTP request.
R	RecordKeyValues (see page 952)	Lists the key values stored in the HTTP request.

7.2.9.3.1 IAdapterRecordKeys.RecordKeyNames Property

Lists the names of the keys stored in the HTTP request.

Pascal

```
property RecordKeyNames [I: Integer]: string;
```

C++

```
__property AnsiString RecordKeyNames[int I];
```

Description

Read RecordKeyNames to obtain the names of the keys that the adapter uses to locate the record associated with the HTTP

request. The RecordKeyValues (see page 952) property lists the values of those keys.

I is the index of the key, where 0 indicates the first key, 1 indicates the second key, and so on. The GetRecordKeyCount (see page 950) method returns an upper bound for this index.

See Also

GetRecordKeyCount (see page 950)

RecordKeyValues (see page 952)

7.2.9.3.2 IAdapterRecordKeys.RecordKeyValues Property

Lists the key values stored in the HTTP request.

Pascal

```
property RecordKeyValues [I: Integer]: string;
```

C++

```
__property AnsiString RecordKeyValues[int I];
```

Description

Read RecordKeyValues to extract the value of a key for the adapter to use when locating the record associated with the HTTP request. The RecordKeyNames (see page 951) property gives the name of the key.

I is the index of the key, where 0 indicates the first key, 1 indicates the second key, and so on. The GetRecordKeyCount (see page 950) method returns an upper bound for this index.

See Also

GetRecordKeyCount (see page 950)

RecordKeyNames (see page 951)

7.2.10 IAdapterRequestParams Interface

IAdapterRequestParams is the interface for extracting name/value pairs from an HTTP request message.

Class Hierarchy

Pascal

```
IAdapterRequestParams = interface;
```

C++

```
__interface IAdapterRequestParams;
```

File

AdaptReq

Description

Use IAdapterRequestParams to obtain the parameters needed to execute an adapter action from an HTTP request message. You can obtain an IAdapterRequestParams interface by casting (Delphi) or calling QueryInterface on (C++) the interface that is the AdapterRequest property of the global Web context.

The name/value pairs that IAdapterRequestParams makes available are added as hidden fields to an HTML form by the adapter's EncodeActionParams method. Use the IAdapterRequestParams interface of a request that originates from the form containing those hidden fields to retrieve their names and values.

See Also

IActionRequest (see page 935)

IActionFieldValues (see page 925)

IAdapterModes (see page 944)

IAdapterRecordKeys (see page 949)

IImageRequest (see page 965)

TAbstractWebContext (see page 24644)

WebContext

TCustomAdapter

TCustomAdapter

7.2.10.1 IAdapterRequestParams Members

The following tables list the members exposed by IAdapterRequestParams.

Public Methods

	Name	Description
	AssignTo (see page 954)	Fills a string list to contain the name/value pairs from the HTTP request message.
	GetParamCount (see page 954)	Returns the value of the ParamCount (see page 956) property.
	GetParamName (see page 954)	Returns the value of the ParamNames (see page 956) property.
	GetParamValue (see page 955)	Returns the value of the ParamValues (see page 957) property.
	IndexOfParam (see page 955)	Returns the index of a specified name/value pair given its name.
	ValueOfParam (see page 955)	Returns the value associated with a specified name.

Public Properties

	Name	Description
R	ParamCount (see page 956)	Indicates the number of name/value pairs that can be accessed using this interface.
R	ParamNames (see page 956)	Lists the names of the name/value pairs that can be accessed using this interface.
R	ParamValues (see page 957)	Lists the values of the name/value pairs that can be accessed using this interface.

7.2.10.2 IAdapterRequestParams Methods

The methods of the IAdapterRequestParams class are listed here.

Public Methods

	Name	Description
	AssignTo (see page 954)	Fills a string list to contain the name/value pairs from the HTTP request message.
	GetParamCount (see page 954)	Returns the value of the ParamCount (see page 956) property.
	GetParamName (see page 954)	Returns the value of the ParamNames (see page 956) property.
	GetParamValue (see page 955)	Returns the value of the ParamValues (see page 957) property.
	IndexOfParam (see page 955)	Returns the index of a specified name/value pair given its name.
	ValueOfParam (see page 955)	Returns the value associated with a specified name.

7.2.10.2.1 IAdapterRequestParams.AssignTo Method

Fills a string list to contain the name/value pairs from the HTTP request message.

Pascal

```
procedure AssignTo(AStrings: TStrings);
```

C++

```
__fastcall AssignTo(TStrings AStrings);
```

Description

AssignTo replaces the content of a string list with strings of the form 'Name=Value', where Name and Value represent the names and values of name/value pairs extracted from the HTTP request message. After a call to AssignTo, the string list contains the same information that this interface makes available.

AStrings is the string list to fill with the name/value pairs from the HTTP request.

See Also

ParamNames (see page 956)

ParamValues (see page 957)

7.2.10.2.2 IAdapterRequestParams.GetParamCount Method

Returns the value of the ParamCount (see page 956) property.

Pascal

```
function GetParamCount: Integer;
```

C++

```
__fastcall int GetParamCount();
```

Description

GetParamCount is the read implementation of the ParamCount (see page 956) property.

See Also

ParamCount (see page 956)

7.2.10.2.3 IAdapterRequestParams.GetParamName Method

Returns the value of the ParamNames (see page 956) property.

Pascal

```
function GetParamName(I: Integer): string;
```

C++

```
__fastcall AnsiString GetParamName(int I);
```

Description

GetParamName is the read implementation of the ParamNames (see page 956) property.

I is the index of the name to retrieve.

See Also

ParamNames (see page 956)

7.2.10.2.4 IAdapterRequestParams.GetParamValue Method

Returns the value of the ParamValues (see page 957) property.

Pascal

```
function GetParamValue(I: Integer): string;
```

C++

```
__fastcall AnsiString GetParamValue(int I);
```

Description

GetParamValue is the read implementation of the ParamValues (see page 957) property.

I is the index of the value to retrieve.

See Also

ParamValues (see page 957)

7.2.10.2.5 IAdapterRequestParams.IndexOfParam Method

Returns the index of a specified name/value pair given its name.

Pascal

```
function IndexOfParam(const AName: string): Integer;
```

C++

```
__fastcall int IndexOfParam(const AnsiString AName);
```

Description

IndexOfParam returns the index of the value associated with a specified name. This method performs the reverse lookup from reading the ParamNames (see page 956) property.

AName is the name of the name/value pair to locate.

IndexOfParam returns the index for the specified name/value pair. This index identifies the position of the pair in the ParamNames (see page 956) and ParamValues (see page 957) property arrays. If the current HTTP request does not include a name/value pair with the specified name, IndexOfParam returns –1.

See Also

ParamNames (see page 956)

ValueOfParam (see page 955)

ParamValues (see page 957)

7.2.10.2.6 IAdapterRequestParams.ValueOfParam Method

Returns the value associated with a specified name.

Pascal

```
function ValueOfParam(const AName: string): string;
```

C++

```
__fastcall AnsiString ValueOfParam(const AnsiString AName);
```

Description

ValueOfParam returns the value part of the name/value pair with the specified name. Use ValueOfParam to access a value by name rather than by index.

AName is the name part of the name/value pair.

ValueOfParam returns the associated value string. If the current HTTP request does not include a parameter with the specified name, ValueOfParam returns an empty string.

See Also

IndexOfParam (see page 955)

ParamValues (see page 957)

7.2.10.3 IAdapterRequestParams Properties

The properties of the IAdapterRequestParams class are listed here.

Public Properties

	Name	Description
R	ParamCount (see page 956)	Indicates the number of name/value pairs that can be accessed using this interface.
R	ParamNames (see page 956)	Lists the names of the name/value pairs that can be accessed using this interface.
R	ParamValues (see page 957)	Lists the values of the name/value pairs that can be accessed using this interface.

7.2.10.3.1 IAdapterRequestParams.ParamCount Property

Indicates the number of name/value pairs that can be accessed using this interface.

Pascal

```
property ParamCount: Integer;
```

C++

```
__property int ParamCount;
```

Description

Read ParamCount to determine how many name/value pairs you can extract from the current HTTP request. ParamCount provides an upper bound to the index on the ParamNames (see page 956) and ParamValues (see page 957) property arrays.

See Also

ParamNames (see page 956)

ParamValues (see page 957)

7.2.10.3.2 IAdapterRequestParams.ParamNames Property

Lists the names of the name/value pairs that can be accessed using this interface.

Pascal

```
property ParamNames [I: Integer]: string;
```

C++

```
__property AnsiString ParamNames[int I];
```

Description

Read ParamNames to obtain the name parts of the name/value pairs you can extract from the current HTTP request.

I is the index of the name/value pair, where 0 indicates the first pair, 1 indicates the second pair, and so on. The ParamCount (see page 956) property provides an upper bound for this index.

Use ParamNames when iterating over all the name/value pairs in the request. To obtain the value of a specified parameter given its name, use the ValueOfParam (see page 955) method.

See Also

ParamCount (see page 956)

ParamValues (see page 957)

IndexOfParam (see page 955)

ValueOfParam (see page 955)

7.2.10.3.3 IAdapterRequestParams.ParamValues Property

Lists the values of the name/value pairs that can be accessed using this interface.

Pascal

```
property ParamValues [I: Integer]: string;
```

C++

```
__property AnsiString ParamValues[int I];
```

Description

Read ParamValues to access the value of a name/value pair given its position.

I is the index of the name/value pair, where 0 indicates the first pair, 1 indicates the second pair, and so on. The ParamCount (see page 956) property provides an upper bound for this index.

Tip: Use the ParamNames (see page 956) property to determine the name associated with a specified index.

Use ParamValues when iterating over all the name/value pairs in the request. To obtain the value for a specific name, use the ValueOfParam (see page 955) method instead.

See Also

ParamCount (see page 956)

ParamNames (see page 956)

ValueOfParam (see page 955)

7.2.11 IEnumActionFieldValueRecords Interface

IEnumActionFieldValueRecords is the interface for iterating over the data in an HTTP request for each of several adapter records.

Class Hierarchy

AdaptReq.IEnumActionFieldValueRecords

Pascal

```
IEnumActionFieldValueRecords = interface;
```

C++

```
__interface IEnumActionFieldValueRecords;
```

File

AdaptReq

Description

Use IEnumActionFieldValueRecords when extracting information from an HTTP request message for an adapter that supports multiple records of information. Typically, this interface is used to process requests from HTML forms that use a grid to represent information from an adapter.

You can obtain an IEnumActionFieldValueRecords interface by casting (Delphi) or calling QueryInterface on (C++) the interface that is the AdapterRequest property of the global Web context.

IEnumActionFieldValueRecords includes methods to "position" the request on a particular record. Once the request is positioned on a record, the IActionFieldValues (see page 925) interfaces gives the values of adapter fields for that record only. After processing those values, use IEnumActionFieldValueRecords to position the request on the next record.

See Also

IActionRequest (see page 935)

IAdapterModes (see page 944)

IAdapterRecordKeys (see page 949)

IAdapterRequestParams (see page 952)

IImageRequest (see page 965)

TAbstractWebContext (see page 24644)

WebContext

IActionFieldValues (see page 925)

7.2.11.1 IEnumActionFieldValueRecords Members

The following tables list the members exposed by IEnumActionFieldValueRecords.

Public Methods

	Name	Description
	EndIterator (see page 959)	Cleans up after iterating over the records in the request.
	FindPopulatedRecord (see page 959)	Detects whether any of the records of the request contain data.
	GetEOF (see page 960)	Returns the value of the EOF property.
	GetFirstRecord (see page 960)	Positions the request on the first record.
	GetInIterator (see page 960)	Returns the value of the InIterator (see page 962) property.
	GetNextRecord (see page 961)	Repositions the request on the next record.
	GetRecord (see page 961)	Positions the request on a specified record.
	GetRecordCount (see page 962)	Returns the value of the RecordCount (see page 963) property.

Public Properties

	Name	Description
R	Eof (see page 962)	This is Eof, a member of class IEnumActionFieldValueRecords.
R	InIterator (see page 962)	Indicates whether the request is explicitly positioned on an adapter record.
R	RecordCount (see page 963)	Indicates the number of records of information the request contains.

7.2.11.2 IEnumActionFieldValueRecords Methods

The methods of the IEnumActionFieldValueRecords class are listed here.

Public Methods

	Name	Description
	EndIterator (see page 959)	Cleans up after iterating over the records in the request.
	FindPopulatedRecord (see page 959)	Detects whether any of the records of the request contain data.
	GetEOF (see page 960)	Returns the value of the EOF property.
	GetFirstRecord (see page 960)	Positions the request on the first record.
	GetInIterator (see page 960)	Returns the value of the InIterator (see page 962) property.
	GetNextRecord (see page 961)	Repositions the request on the next record.
	GetRecord (see page 961)	Positions the request on a specified record.
	GetRecordCount (see page 962)	Returns the value of the RecordCount (see page 963) property.

7.2.11.2.1 IEnumActionFieldValueRecords.EndIterator Method

Cleans up after iterating over the records in the request.

Pascal

```
procedure EndIterator;
```

C++

```
__fastcall EndIterator();
```

Description

Call EndIterator when you are finished iterating over the records in the HTTP request. EndIterator performs all necessary cleanup, and sets the InIterator (see page 962) property to false.

See Also

InIterator (see page 962)

7.2.11.2.2 IEnumActionFieldValueRecords.FindPopulatedRecord Method

Detects whether any of the records of the request contain data.

Pascal

```
function FindPopulatedRecord: Boolean;
```

C++

```
__fastcall Boolean FindPopulatedRecord();
```

Description

Call FindPopulatedRecord to determine whether any of the records on which you can position the request contain field values. FindPopulatedRecord returns true if there is at least one record on which you can position the request and then obtain a non-empty IActionFieldValues (see page 925) interface.

See Also

IActionFieldValues (see page 925)

7.2.11.2.3 IEnumActionFieldValueRecords.GetEOF Method

Returns the value of the EOF property.

Pascal

```
function GetEOF: Boolean;
```

C++

```
__fastcall Boolean GetEOF();
```

Description

GetEOF is the read implementation of the EOF property.

See Also

EOF (see page 962)

7.2.11.2.4 IEnumActionFieldValueRecords.GetFirstRecord Method

Positions the request on the first record.

Pascal

```
function GetFirstRecord: Boolean;
```

C++

```
__fastcall Boolean GetFirstRecord();
```

Description

Call GetFirstRecord to start iterating over the records in the HTTP request. GetFirstRecord positions the request on the first record. That is, after a call to GetFirstRecord, if you obtain an IActionFieldValues (see page 925) interface for the request, it will extract the field values for the first record.

GetFirstRecord returns true if it successfully positions the request on the first record and sets the InIterator (see page 962) property to true. GetFirstRecord returns false if it is unable to position the request on the first record. When GetFirstRecord returns false, the value of the InIterator (see page 962) property is false, and the value of the EOF property is true.

See Also

InIterator (see page 962)

GetNextRecord (see page 961)

GetRecord (see page 961)

IActionFieldValues (see page 925)

7.2.11.2.5 IEnumActionFieldValueRecords.GetInIterator Method

Returns the value of the InIterator (see page 962) property.

Pascal

```
function GetInIterator: Boolean;
```

C++

```
__fastcall Boolean GetInIterator();
```

Description

GetInIterator is the read implementation of the InIterator (see page 962) property.

See Also

InIterator (see page 962)

7.2.11.2.6 IEnumActionFieldValueRecords.GetNextRecord Method

Repositions the request on the next record.

Pascal

```
function GetNextRecord: Boolean;
```

C++

```
__fastcall Boolean GetNextRecord();
```

Description

Call GetNextRecord to move to the next record of the request. After a call to GetNextRecord, if you obtain an IActionFieldValues (see page 925) interface for the request, it will extract the field values for the next record.

GetNextRecord returns true if it successfully positions the request on the next record. GetNextRecord returns false if it is unable to position the request on the next record. When GetFirstRecord (see page 960) returns false, the value of the InIterator (see page 962) property is false, and the value of the EOF property is true.

See Also

GetFirstRecord (see page 960)

GetRecord (see page 961)

EOF (see page 962)

IActionFieldValues (see page 925)

7.2.11.2.7 IEnumActionFieldValueRecords.GetRecord Method

Positions the request on a specified record.

Pascal

```
function GetRecord(AIndex: Integer): Boolean;
```

C++

```
__fastcall Boolean GetRecord(int AIndex);
```

Description

Call GetRecord to move to a specified record of the request. After a call to GetRecord, if you obtain an IActionFieldValues (see page 925) interface for the request, it will extract the field values for the specified record.

AIndex is the index of the record to which you want to move, where 0 specifies the first record, 1 specifies the second record, and so on. AIndex should be less than the value of the RecordCount (see page 963) property.

GetRecord returns true if it successfully positions the request on the specified record. GetRecord returns false if it is unable to position the request on the specified record.

See Also

RecordCount (see page 963)

GetFirstRecord (see page 960)

GetNextRecord (see page 961)

IActionFieldValues (see page 925)

7.2.11.2.8 IEnumActionFieldValueRecords.GetRecordCount Method

Returns the value of the RecordCount (see page 963) property.

Pascal

```
function GetRecordCount: Integer;
```

C++

```
__fastcall int GetRecordCount();
```

Description

GetRecordCount is the read implementation of the RecordCount (see page 963) property.

See Also

RecordCount (see page 963)

7.2.11.3 IEnumActionFieldValueRecords Properties

The properties of the IEnumActionFieldValueRecords class are listed here.

Public Properties

	Name	Description
R	Eof (see page 962)	This is Eof, a member of class IEnumActionFieldValueRecords.
R	InIterator (see page 962)	Indicates whether the request is explicitly positioned on an adapter record.
R	RecordCount (see page 963)	Indicates the number of records of information the request contains.

7.2.11.3.1 IEnumActionFieldValueRecords.Eof Property

Pascal

```
property Eof: Boolean;
```

C++

```
__property Boolean Eof;
```

Description

This is Eof, a member of class IEnumActionFieldValueRecords.

7.2.11.3.2 IEnumActionFieldValueRecords.InIterator Property

Indicates whether the request is explicitly positioned on an adapter record.

Pascal

```
property InIterator: Boolean;
```

C++

```
__property Boolean InIterator;
```

Description

Read InIterator to determine whether the request is currently iterating over the records of information. InIterator is first set to true when you call the GetFirstRecord (see page 960) method to position the request on the first record. It remains true until you position the request on the last record (until the EOF property becomes true) or until you call the EndIterator (see page 959) method to stop iterating.

Note: In the degenerate case where RecordCount (see page 963) is 0, InIterator is always false, even after a call to GetFirstRecord (see page 960).

See Also

GetFirstRecord (see page 960)

EndIterator (see page 959)

EOF (see page 962)

RecordCount (see page 963)

7.2.11.3.3 IEnumActionFieldValueRecords.RecordCount Property

Indicates the number of records of information the request contains.

Pascal

```
property RecordCount: Integer;
```

C++

```
__property int RecordCount;
```

Description

Read RecordCount to determine the number of records on which you can position the request. After a call to the GetFirstRecord (see page 960) method, this is the number of times you can call GetNextRecord (see page 961) before the EOF property becomes true.

See Also

GetNextRecord (see page 961)

GetFirstRecord (see page 960)

EOF (see page 962)

7.2.12 IGetAdapterActionRedirectOptions Interface

Class Hierarchy

AdaptReq.IGetAdapterActionRedirectOptions

Pascal

```
IGetAdapterActionRedirectOptions = interface;
```

C++

```
__interface IGetAdapterActionRedirectOptions;
```

File

AdaptReq

Description

This is class AdaptReq.IGetAdapterActionRedirectOptions.

7.2.12.1 IGetAdapterActionRedirectOptions Members

The following tables list the members exposed by IGetAdapterActionRedirectOptions.

Public Methods

	Name	Description
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.

Public Properties

	Name	Description
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.

7.2.12.2 IGetAdapterActionRedirectOptions Methods

The methods of the IGetAdapterActionRedirectOptions class are listed here.

Public Methods

	Name	Description
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.

7.2.12.2.1 IGetAdapterActionRedirectOptions.GetFailureOptions Method

Pascal

```
function GetFailureOptions: TAdapterActionRedirectOptions;
```

C++

```
__fastcall TAdapterActionRedirectOptions GetFailureOptions();
```

Description

This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.

7.2.12.2.2 IGetAdapterActionRedirectOptions.GetSuccessOptions Method

Pascal

```
function GetSuccessOptions: TAdapterActionRedirectOptions;
```

C++

```
__fastcall TAdapterActionRedirectOptions GetSuccessOptions();
```

Description

This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.

7.2.12.3 IGetAdapterActionRedirectOptions Properties

The properties of the IGetAdapterActionRedirectOptions class are listed here.

Public Properties

	Name	Description
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.

7.2.12.3.1 IGetAdapterActionRedirectOptions.FailureOptions Property

Pascal

```
property FailureOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions FailureOptions;
```

Description

This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.

7.2.12.3.2 IGetAdapterActionRedirectOptions.SuccessOptions Property

Pascal

```
property SuccessOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions SuccessOptions;
```

Description

This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.

7.2.13 IImageRequest Interface

IImageRequest is the interface for extracting the name of a requested image from an HTTP request message.

Class Hierarchy

AdaptReq.IImageRequest

Pascal

```
IImageRequest = interface;
```

C++

```
__interface IImageRequest;
```

File

AdaptReq

Description

Use IImageRequest to obtain the name of the image that an HTTP request message requests. You can obtain an IImageRequest interface by casting (Delphi) or calling QueryInterface on (C++) the interface that is the AdapterRequest property of the global Web context.

See Also

IActionRequest (see page 935)

IAdapterRequestParams (see page 952)

IAdapterModes (see page 944)

IAdapterRecordKeys (see page 949)

TAbstractWebContext (see page 24644)

WebContext

IEnumActionFieldValueRecords (see page 957)

7.2.13.1 IImageRequest Members

The following tables list the members exposed by IImageRequest.

Public Methods

	Name	Description
	GetImageName (see page 966)	Returns the value of the ImageName (see page 967) property.

Public Properties

	Name	Description
R	ImageName (see page 967)	Indicates the name of the requested image.

7.2.13.2 IImageRequest Methods

The methods of the IImageRequest class are listed here.

Public Methods

	Name	Description
	GetImageName (see page 966)	Returns the value of the ImageName (see page 967) property.

7.2.13.2.1 IImageRequest.GetImageName Method

Returns the value of the ImageName (see page 967) property.

Pascal

```
function GetImageName: string;
```

C++

```
__fastcall AnsiString GetImageName();
```

Description

GetImageName is the read implementation of the ImageName (see page 967) property.

See Also

ImageName (see page 967)

7.2.13.3 IImageRequest Properties

The properties of the IImageRequest class are listed here.

Public Properties

	Name	Description
R	ImageName (see page 967)	Indicates the name of the requested image.

7.2.13.3.1 IImageRequest.ImageName Property

Indicates the name of the requested image.

Pascal

```
property ImageName: string;
```

C++

```
__property AnsiString ImageName;
```

Description

Read ImageName to obtain the name of the image that the HTTP request message requests.

7.2.14 IImageResponse Interface

Class Hierarchy

AdaptReq.IImageResponse

Pascal

```
IImageResponse = interface;
```

C++

```
__interface IImageResponse;
```

File

AdaptReq

Description

This is class AdaptReq.IImageResponse.

7.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	ExtractStringValue (see page 968)	This is function AdaptReq.ExtractStringValue.
	TestRedirectOptions (see page 968)	This is function AdaptReq.TestRedirectOptions.

7.3.1 AdaptReq.ExtractStringValue Function

Pascal

```
function ExtractStringValue(const S: string): string;
```

C++

```
AnsiString ExtractStringValue(const AnsiString S);
```

File

AdaptReq

Description

This is function AdaptReq.ExtractStringValue.

7.3.2 AdaptReq.TestRedirectOptions Function

Pascal

```
function TestRedirectOptions(AOptions: TAdapterActionRedirectOptions; AMethodType:
TMethodType): Boolean;
```

C++

```
Boolean TestRedirectOptions(TAdapterActionRedirectOptions AOptions, TMethodType AMethodType);
```

File

AdaptReq

Description

This is function AdaptReq.TestRedirectOptions.

7.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAdapterActionRedirectOption (see page 969)	This is record AdaptReq.TAdapterActionRedirectOption.
	TExecuteStatus (see page 969)	This is record AdaptReq.TExecuteStatus.
	TRespondWith (see page 970)	This is record AdaptReq.TRespondWith.

7.4.1 AdaptReq.TAdapterActionRedirectOption Enumeration

Pascal

```
TAdapterActionRedirectOption = (
  roRedirectHTTPPost,
  roRedirectHTTPGet
);
```

C++

```
enum TAdapterActionRedirectOption {
  roRedirectHTTPPost,
  roRedirectHTTPGet
};
```

File

AdaptReq

Description

This is record AdaptReq.TAdapterActionRedirectOption.

7.4.2 AdaptReq.TExecuteStatus Enumeration

Pascal

```
TExecuteStatus = (
  esSuccess,
  esFail,
  esUndefined,
  esOther
);
```

C++

```
enum TExecuteStatus {
  esSuccess,
  esFail,
  esUndefined,
  esOther
};
```

File

AdaptReq

Description

This is record AdaptReq.TExecuteStatus.

7.4.3 AdaptReq.TRespondWith Enumeration

Pascal

```
TRespondWith = (
  rwPage,
  rwURL,
  rwComponentPage,
  rwUndefined
);
```

C++

```
enum TRespondWith {
  rwPage,
  rwURL,
  rwComponentPage,
  rwUndefined
};
```

File

AdaptReq

Description

This is record AdaptReq.TRespondWith.

7.5 Types

The following table lists types in this documentation.

Types

Name	Description
TAdapterActionRedirectOptions (see page 971)	This is type AdaptReq.TAdapterActionRedirectOptions.

7.5.1 AdaptReq.TAdapterActionRedirectOptions Type

Pascal

```
TAdapterActionRedirectOptions = set of TAdapterActionRedirectOption;
```

C++

```
TAdapterActionRedirectOption TAdapterActionRedirectOptions;
```

File

AdaptReq

Description

This is type AdaptReq.TAdapterActionRedirectOptions.

7.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
roRedirect (see page 972)	This is constant AdaptReq.roRedirect.
sActionRequest (see page 972)	This is constant AdaptReq.sActionRequest.
sDelimAdapterMode (see page 973)	This is constant AdaptReq.sDelimAdapterMode.
sDelimOrigFieldValue (see page 973)	This is constant AdaptReq.sDelimOrigFieldValue.
sDelimRecordKeys (see page 973)	This is constant AdaptReq.sDelimRecordKeys.

7.6.1 AdaptReq.roRedirect Constant

Pascal

```
roRedirect = [roRedirectHTTPPost, roRedirectHTTPGet];
```

C++

```
roRedirect = [roRedirectHTTPPost, roRedirectHTTPGet];
```

File

AdaptReq

Description

This is constant AdaptReq.roRedirect.

7.6.2 AdaptReq.sActionRequest Constant

Pascal

```
sActionRequest = '__act';
```

C++

```
sActionRequest = '__act';
```

File

AdaptReq

Description

This is constant AdaptReq.sActionRequest.

7.6.3 AdaptReq.sDelimAdapterMode Constant

Pascal

```
sDelimAdapterMode = '__am.';
```

C++

```
sDelimAdapterMode = '__am.';
```

File

AdaptReq

Description

This is constant AdaptReq.sDelimAdapterMode.

7.6.4 AdaptReq.sDelimOrigFieldValue Constant

Pascal

```
sDelimOrigFieldValue = '__ov.';
```

C++

```
sDelimOrigFieldValue = '__ov.';
```

File

AdaptReq

Description

This is constant AdaptReq.sDelimOrigFieldValue.

7.6.5 AdaptReq.sDelimRecordKeys Constant

Pascal

```
sDelimRecordKeys = '__key.';
```

C++

```
sDelimRecordKeys = '__key.';
```

File

AdaptReq

Description

This is constant AdaptReq.sDelimRecordKeys.

8 ADODB Namespace

This is namespace ADODB.

Topics

Name	Description
Classes (see page 978)	The following table lists classes in this documentation.
Functions (see page 1286)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 1289)	The following table lists structs, records, enums in this documentation.
Types (see page 1310)	The following table lists types in this documentation.

Classes

	Name	Description
	EADOError (see page 978)	EADOError is the exception class for database errors generated by ADO databases.
	TADOBlobStream (see page 979)	TADOBlobStream is a stream object that reads from and writes to field objects that represent Binary large object (BLOB) fields.
	TADOCommand (see page 984)	TADOCommand represents an ADO command object.
	TADOConnection (see page 1001)	TADOConnection connects to an ADO data store.
	TADODataSet (see page 1041)	TADODataSet represents a dataset retrieved from an ADO data store.
	TADOQuery (see page 1084)	TADOQuery provides the means for issuing SQL (see page 1105) against an ADO data store.
	TADOStoredProc (see page 1106)	TADOStoredProc encapsulates a stored procedure in an ADO database.
	TADOTable (see page 1125)	TADOTable is a dataset component that encapsulates a table accessed through an ADO data store.
	TCustomADODataSet (see page 1145)	TCustomADODataSet is the virtualized base class for all ADO dataset components.
	TParameter (see page 1250)	TParameter represents a parameter.
	TParameters (see page 1261)	TParameters manages a collection of parameters.
	TRDSConnection (see page 1274)	TRDSConnection implements an RDS DataSpace object.

Functions

	Name	Description
	CreateUDLFile (see page 1286)	This is function ADODB.CreateUDLFile.
	DataLinkDir (see page 1286)	This is function ADODB.DataLinkDir.
	GetDataLinkFiles (see page 1287)	This is function ADODB.GetDataLinkFiles.
	GetDataLinkFiles (see page 1287)	This is function ADODB.GetDataLinkFiles.
	GetProviderNames (see page 1287)	This is function ADODB.GetProviderNames.

	GetProviderNames (see page 1288)	This is function ADODB.GetProviderNames.
	PromptDataLinkFile (see page 1288)	This is function ADODB.PromptDataLinkFile.
	PromptDataSource (see page 1288)	This is function ADODB.PromptDataSource.

Structs, Records, Enums

	Name	Description
	TADOLockType (see page 1289)	TADOLockType specifies the types of locks used.
	TAffectRecords (see page 1290)	This is record ADODB.TAffectRecords.
	TCommandType (see page 1291)	TCommandType values designate the type of a command.
	TConnectMode (see page 1292)	TConnectMode values indicate the persimissions available to a connection.
	TConnectOption (see page 1293)	TConnectOption indicates whether the connection to the ADO data store is synchronous or asynchronous.
	TCursorLocation (see page 1293)	TCursorLocation values specify whether to use client-side or server-side cursor library.
	TCursorOption (see page 1294)	TCursorOptions (see page 1314) values indicate the functionality a recordset provides.
	TCursorType (see page 1295)	TCursorType valies indicate the type of cursor to use for a recordset.
	TEventReason (see page 1296)	TEventReason values indicate the reason an event occurred.
	TEventStatus (see page 1298)	TEventStatus values indicate the success or failure of an operation.
	TExecuteOption (see page 1299)	This is record ADODB.TExecuteOption.
	TFilterGroup (see page 1299)	This is record ADODB.TFilterGroup.
	TIsolationLevel (see page 1300)	TIsolationLevel values specify the level of isolation for a connection.
	TMarshalOption (see page 1301)	This is record ADODB.TMarshalOption.
	TObjectState (see page 1302)	This is record ADODB.TObjectState.
	TParameterAttribute (see page 1302)	This is record ADODB.TParameterAttribute.
	TParameterDirection (see page 1303)	TParameterDirection specifies the direction of a parameter.
	TPersistFormat (see page 1303)	TPersistFormat values indicate the format for a saved recordset.
	TRecordStatus (see page 1304)	This is record ADODB.TRecordStatus.
	TSchemaInfo (see page 1305)	TSchemaInfo values indicate the type of schema information to retrieve.
	TSeekOption (see page 1308)	This is record ADODB.TSeekOption.
	TXactAttribute (see page 1308)	TXactAttribute values specify automated transaction behavior.

Types

Name	Description
_Command (see page 1310)	This is type ADODB._Command.
_Connection (see page 1311)	This is type ADODB._Connection.
_Parameter (see page 1311)	This is type ADODB._Parameter.
_Recordset (see page 1311)	This is type ADODB._Recordset.
Error (see page 1312)	This is type ADODB.Error.
Errors (see page 1312)	This is type ADODB.Errors.
Parameters (see page 1312)	This is type ADODB.Parameters.
Properties (see page 1313)	This is type ADODB.Properties.
Property_ (see page 1313)	This is type ADODB.Property_.
PVariantList (see page 1313)	This is type ADODB.PVariantList.
TBeginTransCompleteEvent (see page 1314)	This is type ADODB.TBeginTransCompleteEvent.
TConnectErrorEvent (see page 1314)	This is type ADODB.TConnectErrorEvent.
TCursorOptions (see page 1314)	This is type ADODB.TCursorOptions.
TDataType (see page 1315)	TDataType indicates the type of a parameter.
TDisconnectEvent (see page 1315)	This is type ADODB.TDisconnectEvent.
TEndOfRecordsetEvent (see page 1315)	This is type ADODB.TEndOfRecordsetEvent.
TExecuteCompleteEvent (see page 1316)	This is type ADODB.TExecuteCompleteEvent.
TExecuteOptions (see page 1316)	TExecuteOptions values specify the characteristics of an execution operation.
TFetchProgressEvent (see page 1317)	This is type ADODB.TFetchProgressEvent.
TFieldChangeCompleteEvent (see page 1317)	This is type ADODB.TFieldChangeCompleteEvent.
TInfoMessageEvent (see page 1317)	This is type ADODB.TInfoMessageEvent.

TObjectStates (see page 1318)	TObjectStates values indicate the current state of an ADO object.
TParameterAttributes (see page 1318)	TParameterAttributes values indicate the types of values a parameter can accept.
TPropList (see page 1319)	This is type ADODB.TPropList.
TRecordChangeCompleteEvent (see page 1319)	This is type ADODB.TRecordChangeCompleteEvent.
TRecordsetCreate (see page 1320)	This is type ADODB.TRecordsetCreate.
TRecordsetErrorEvent (see page 1320)	This is type ADODB.TRecordsetErrorEvent.
TRecordsetEvent (see page 1320)	This is type ADODB.TRecordsetEvent.
TRecordsetReasonEvent (see page 1321)	This is type ADODB.TRecordsetReasonEvent.
TRecordStatusSet (see page 1321)	TRecordStatusSet values indicate the status of the current record relative to batch updates or other bulk operations
TVariantList (see page 1322)	This is type ADODB.TVariantList.
TWillChangeFieldEvent (see page 1322)	This is type ADODB.TWillChangeFieldEvent.
TWillChangeRecordEvent (see page 1323)	This is type ADODB.TWillChangeRecordEvent.
TWillConnectEvent (see page 1323)	This is type ADODB.TWillConnectEvent.
TWillExecuteEvent (see page 1323)	This is type ADODB.TWillExecuteEvent.
TXactAttributes (see page 1324)	This is type ADODB.TXactAttributes.

8.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EADOError (see page 978)	EADOError is the exception class for database errors generated by ADO databases.
	TADOBlobStream (see page 979)	TADOBlobStream is a stream object that reads from and writes to field objects that represent Binary large object (BLOB) fields.
	TADOCommand (see page 984)	TADOCommand represents an ADO command object.
	TADOConnection (see page 1001)	TADOConnection connects to an ADO data store.
	TADODataSet (see page 1041)	TADODataSet represents a dataset retrieved from an ADO data store.
	TADOQuery (see page 1084)	TADOQuery provides the means for issuing SQL (see page 1105) against an ADO data store.
	TADOStoredProc (see page 1106)	TADOStoredProc encapsulates a stored procedure in an ADO database.
	TADOTable (see page 1125)	TADOTable is a dataset component that encapsulates a table accessed through an ADO data store.
	TCustomADODataSet (see page 1145)	TCustomADODataSet is the virtualized base class for all ADO dataset components.
	TParameter (see page 1250)	TParameter represents a parameter.
	TParameters (see page 1261)	TParameters manages a collection of parameters.
	TRDSConnection (see page 1274)	TRDSConnection implements an RDS DataSpace object.

8.1.1 EADOError Class

EADOError is the exception class for database errors generated by ADO databases.

Class Hierarchy

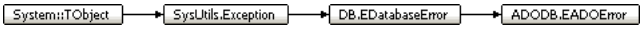

Pascal

```
EADOError = class(EDatabaseError);
```

C++

```
class EADOError : public EDatabaseError;
```

File

ADODB

Description

An EADOError exception is raised when a component detects a database error. Use EADOError in an exception handling block or to create a database exception.

See Also

TDataSet (see page 6894)

DatabaseError

EDatabaseError (see page 6731)

8.1.2 TADOBlobStream Class

TADOBlobStream is a stream object that reads from and writes to field objects that represent Binary large object (BLOB) fields.

Class Hierarchy

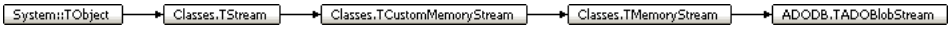

Pascal

```
TADOBlobStream = class(TMemoryStream);
```

C++

```
class TADOBlobStream : public TMemoryStream;
```

File

ADODB

Description

Use TADOBlobStream to access or modify the value of a BLOB or memo field in an ADO dataset. BLOB fields are represented by TBlobField objects and descendants of TBlobField such as TGraphicField and TMemoField.

TADOBlobStream allows objects that have no specialized knowledge of how data is stored in a BLOB field (raw binary data) to read or write such data by employing the uniform stream interface.

To use an ADO BLOB stream, create an instance of TADOBlobStream, use the methods of the stream to read or write the data, and then free the BLOB stream. Do not use the same instance of TADOBlobStream to access data from more than one record. Instead, create a new TADOBlobStream object every time you need to read or write BLOB data on a new record.

Use the Read method to copy the data from BLOB field object to a null-terminated string (or comparable) buffer. Use the Write (see page 983) method to copy from buffer to BLOB field object. When reading data from the BLOB field object, use the Size property to determine the volume of the field's contents and to allocate the correct amount of space in memory for the receiving buffer.

See Also

TStringStream (see page 3088)

TStream (see page 2973)

TStream (see page 2973)

TCustomMemoryStream (see page 2815)

TADOBlobStream

PChar

8.1.2.1 TADOBlobStream Members

The following tables list the members exposed by TADOBlobStream.

Protected Methods

	Name	Description
	GetSize (see page 2982)	This is GetSize, a member of class TStream.
	ReadBlobData (see page 982)	This is ReadBlobData, a member of class TADOBlobStream.

	Realloc (see page 982)	This is Realloc, a member of class TADOBlobStream.
	SetPointer (see page 2823)	Replaces the memory buffer associated with the memory stream.
	SetSize (see page 2989)	This is the overview for the SetSize method overload.

Protected Properties

	Name	Description
	Capacity (see page 2912)	This is Capacity, a member of class TMemoryStream.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2909)	Frees the memory buffer, discarding all data associated with the memory stream.
	CopyFrom (see page 2979)	Copies a specified number of bytes from one stream to another.
	Create (see page 981)	Creates a new TADOBlobStream (see page 979) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 981)	Destroys an instance of TADOBlobStream (see page 979).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FixupResourceHeader (see page 2981)	Patches the resource header for a resource that has been written to the stream.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LoadFromFile (see page 2909)	Loads the entire contents of a file into the memory buffer.
	LoadFromStream (see page 2909)	Loads the entire contents of a stream into the memory buffer.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Read (see page 2817)	Reads up to Count bytes from the memory stream into Buffer and advances the current position of the stream by the number of bytes read.
	Read (see page 2982)	Introduces an abstract or pure virtual method responsible for reading from the stream.
	ReadBuffer (see page 2983)	Reads bytes from the stream into Buffer.
	ReadComponent (see page 2983)	Initiates streaming of components and their properties.
	ReadComponentRes (see page 2987)	Reads components and their properties from a stream in a resource file format.
	ReadResHeader (see page 2987)	Reads a resource-file header from the stream.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 2818)	Writes the entire contents of the memory stream to the file with a given file name.

	SaveToStream (see page 2818)	Writes the entire contents of the memory stream to the stream object specified by Stream.
V	Seek (see page 2819)	Moves the current position of the stream by Offset bytes, relative to the origin specified by Origin.
V	Seek (see page 2988)	This is the overview for the Seek method overload.
V	SetSize (see page 2910)	Sets the Size (see page 2997) property of the memory stream.
	Truncate (see page 983)	Discards all data in the BLOB field from the current position on.
V	Write (see page 983)	Writes data from a memory buffer to a BLOB field object.
	WriteBuffer (see page 2990)	Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.
	WriteComponent (see page 2990)	Initiates the writing of components and their properties to a stream.
	WriteComponentRes (see page 2994)	Writes components and their properties to a stream from a resource file format.
	WriteDescendent (see page 2994)	Streams components and their properties in inherited forms.
	WriteDescendentRes (see page 2995)	Streams components and their properties in inherited forms.
	WriteResourceHeader (see page 2995)	Writes a resource-file header to the stream.

Public Properties

	Name	Description
R	Memory (see page 2823)	Provides direct access to the memory pool allocated for the memory stream.
	Position (see page 2996)	Indicates the current offset into the stream for reading and writing.
	Size (see page 2997)	Indicates the size in bytes of the stream.

8.1.2.2 TADOBlobStream.Create Constructor

Creates a new TADOBlobStream (see page 979) object.

Pascal

```
constructor Create(Field: TBlobField; Mode: TBlobStreamMode);
```

C++

```
__fastcall TADOBlobStream(TBlobField Field, TBlobStreamMode Mode);
```

Description

Call Create to obtain an instance of TADOBlobStream (see page 979) for reading from or writing to a specific BLOB field object.

Create links the TADOBlobStream (see page 979) to the field object specified by the Field parameter. Mode specifies whether the BLOB stream will be used to read data (bmRead), write data (bmWrite) or modify data (bmReadWrite).

See Also

Destroy_d

8.1.2.3 TADOBlobStream.Destroy Destructor

Destroys an instance of TADOBlobStream (see page 979).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TADOBlobStream();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TADOBlobStream (see page 979)

reference is not nil before calling Destroy.

Destroy a TADOBlobStream (see page 979) object when it is no longer needed to store or write data. Destroy calls Clear to free the memory buffer before calling the inherited destructor.

Destroy performs the following tasks:

Sets the field's data.

Changes the field object's Modified property to true.

Triggers a field change event in the field's dataset object.

Calls the inherited destructor.

See Also

Create_d

Free (see page 22300)

Modified (see page 6820)

8.1.2.4 TADOBlobStream Methods

The methods of the TADOBlobStream class are listed here.

Protected Methods

	Name	Description
	ReadBlobData (see page 982)	This is ReadBlobData, a member of class TADOBlobStream.
V	Realloc (see page 982)	This is Realloc, a member of class TADOBlobStream.

Public Methods

	Name	Description
	Truncate (see page 983)	Discards all data in the BLOB field from the current position on.
V	Write (see page 983)	Writes data from a memory buffer to a BLOB field object.

8.1.2.4.1 TADOBlobStream.ReadBlobData Method

Pascal

```
procedure ReadBlobData;
```

C++

```
__fastcall ReadBlobData();
```

Description

This is ReadBlobData, a member of class TADOBlobStream.

8.1.2.4.2 TADOBlobStream.Realloc Method

Pascal

```
function Realloc(var NewCapacity: Longint): Pointer; override;
```

C++

```
virtual __fastcall void * Realloc(Longint NewCapacity);
```

Description

This is Realloc, a member of class TADOBlobStream.

8.1.2.4.3 TADOBlobStream.Truncate Method

Discards all data in the BLOB field from the current position on.

Pascal

```
procedure Truncate;
```

C++

```
__fastcall Truncate();
```

Description

Use Truncate to limit the size of the data in the BLOB field. Calling Truncate when the current position is 0 will clear the contents of the BLOB field. If Position is other than zero, data is cleared from the point to the end of the field's data for the current record.

Note: Do not call Truncate when the TADOBlobStream (see page 979) was created in bmRead mode.

See Also

Position (see page 2996)

8.1.2.4.4 TADOBlobStream.Write Method

Writes data from a memory buffer to a BLOB field object.

Pascal

```
function Write(const Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  Buffer, Longint Count);
```

Description

Use Write to copy data from a null-terminated string (or comparable) buffer in memory to the BLOB field object. Write copies data from the buffer to the current position in the BLOB field object (current position indicated by the Position property). Write copies the number of bytes specified in Count or the number of bytes remaining in the memory buffer (counting from the current position to the end), whichever is less. Write can be called multiple times during the lifetime of a TADOBlobStream (see page 979), each call copying less than the total number of bytes to be copied.

Count represents the maximum number of bytes to copy in a single call to Write. Count may be less than, the same as, or greater than the number of bytes actually contained in the memory buffer. If Count is greater than the actual number of bytes present or remaining in the memory buffer, copying stops at the last byte. No penalty is incurred on exceptions generated by specifying a number greater than the actual size of the contents.

Write updates the Size property to Position + Count, and sets the Position property to the new value of Size. Thus, any data that was stored in the memory stream in the Count bytes after the current position is lost when calling Write.

After successful execution, Write returns the number of bytes copied to the BLOB field object. If the return value is less than Count, the most likely reason is that the end of the buffer was reached prior to copying the specified number of bytes.

```
var
  P: PChar;
  S: Integer;
  BS:TADOBlobStream;
begin
```

```
if not (ADOTable1.State in [dsEdit, dsInsert]) then
    ADOTable1.Edit;
  BS := TADOBlobStream.Create(TMemoField(ADOTable1.Fields[1]), bmWrite);
try
    S := Memo1.GetTextLen;
    Inc(S);
    P := AllocMem(S);
    FillChar(P^, S, #0);
    Memo1.GetTextBuf(P, S);
    BS.Write(P^, S);
finally
    BS.Free;
    FreeMem(P, S);
end;
end;
int S;
ADOTable1->Edit();
TADOBlobStream *BS = new TADOBlobStream(ADOTable1->Fields->Fields[1], bmWrite);
try
{
  S = Memo1->GetTextLen() + 1;
char *P = new char[S];
  Memo1->GetTextBuf(P, S);
  BS->Write(P.c_str(), P.Length());
}
__finally
{
delete BS;
}
```

See Also

Memory (see page 2823)

Position (see page 2996)

Size (see page 2997)

WriteBuffer (see page 2990)

WriteComponent (see page 2990)

Position (see page 2996)

8.1.3 TADOCommand Class

TADOCommand represents an ADO command object.

Class Hierarchy

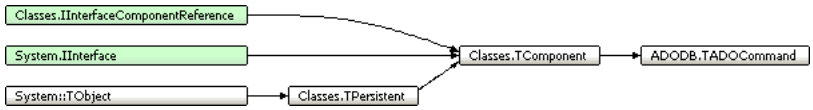

Pascal

```
TADOCommand = class(TComponent);
```

C++

```
class TADOCommand : public TComponent;
```

File

ADODB

Description

The TADOCommand represents the ADO Command object. Use TADOCommand for issuing commands against a data store accessed through an ADO (ActiveX Data Objects) provider.

The TADOCommand component executes the command specified in its CommandText (see page 995) property. One command may be executed at a time. Parameters (see page 999), if the command includes them, are specified in its Parameters (see page 999) property. The command is executed by a call to the Execute (see page 992) method.

TADOCommand can either use a TADOConnection (see page 1001) object to connect to a data store (through its Connection (see page 997) property) or connect directly to the data store if the connection information is specified in the ConnectionString (see page 998) property.

TADOCommand is most often used for executing data definition language (DDL) SQL commands or to execute a stored procedure that does not return a result set. For SQL statements that return a result set, TADODataSet (see page 1041), TADOQuery (see page 1084), or TADOStoredProc (see page 1106) is better suited. The Execute (see page 992) method of TADOCommand is, however, capable of returning a recordset. To use that recordset, however, you will need a separate ADO dataset component.

As a close representation of the ADO Command object, many of the TADOCommand properties and methods have the same name as those of the ADO Command object and serve the same purpose. For example, the CommandType (see page 997) property and Execute (see page 992) methods are the same in both the TADOCommand component and ADO Command object. Note that subsequent versions of ADO may introduce changes in the underlying ADO command object, which will change the behavior of TADOCommand. As a result, you may want to view the Microsoft Data Store SDK documentation for additional information about the behavior of TADOCommand.

See Also

Command

TADOQuery (see page 1084)

TADOTable (see page 1125)

TADODataSet (see page 1041)

TADOStoredProc (see page 1106)

Accessing the Connection's Datasets

Overview of ADO Components

Using Command Objects

8.1.3.1 TADOCommand Members

The following tables list the members exposed by TADOCommand.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.

	_Release (see page 2769)	Called when an application releases a component interface.
	AssignCommandText (see page 990)	This is AssignCommandText, a member of class TADOCommand.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckCommandText (see page 991)	This is CheckCommandText, a member of class TADOCommand.
	ClearActiveConnection (see page 991)	This is ClearActiveConnection, a member of class TADOCommand.
	ComponentLoading (see page 992)	This is ComponentLoading, a member of class TADOCommand.
	ConnectionStateChange (see page 992)	This is ConnectionStateChange, a member of class TADOCommand.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Initialize (see page 993)	This is Initialize, a member of class TADOCommand.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	OpenConnection (see page 993)	This is OpenConnection, a member of class TADOCommand.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetCommandText (see page 994)	This is SetCommandText, a member of class TADOCommand.
	SetConnectionFlag (see page 994)	This is SetConnectionFlag, a member of class TADOCommand.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveConnection (see page 995)	This is ActiveConnection, a member of class TADOCommand.
	CommandTextAlias (see page 996)	This is CommandTextAlias, a member of class TADOCommand.
	ComponentRef (see page 997)	This is ComponentRef, a member of class TADOCommand.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 990)	Makes one ADO command component a duplicate of another.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	Cancel (see page 991)	Halts execution of an ADO command.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 988)	Constructs an instance of an ADO command component.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 989)	Destroys the command component.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 992)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	CommandObject (see page 995)	Provides direct access to the ADO command object.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Properties (see page 1000)	This is Properties, a member of class TADOCommand.
R	States (see page 1001)	Indicates the current state of the ADO command component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	CommandText (see page 995)	Specifies command to execute.
	CommandTimeout (see page 996)	Specifies the amount of time to attempt execution of the command.
	CommandType (see page 997)	Specifies the type of command to execute.
	Connection (see page 997)	Specifies ADO connection component to use.
	ConnectionString (see page 998)	Specifies the connection information for the database.
	ExecuteOptions (see page 999)	Specifies the characteristics of a command execution.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParamCheck (see page 999)	Specifies whether the parameter list for an ADO command component is regenerated if the SQL changes at runtime.
	Parameters (see page 999)	Contains the parameters for an ADO command object's SQL statement.
	Prepared (see page 1000)	Specifies whether the command is prepared before execution.
	Tag (see page 2808)	Stores an integer value as part of a component.

8.1.3.2 TADOCommand.Create Constructor

Constructs an instance of an ADO command component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TADOCommand(TComponent * AOwner);
```

Description

Use Create to programmatically instantiate an ADO command component. Command components added in the form designer are created automatically.

Create

Calls its inherited Create method.

Initializes the CommandObject (see page 995) property.

Initializes the Parameters (see page 999) property.

Sets the ParamCheck (see page 999) property to true.

See Also

Create_d

Destroy_d

8.1.3.3 TADOCommand.Destroy Destructor

Destroys the command component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TADOCommand();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the command component is not nil, and only then calls Destroy.

Destroy performs the following tasks:

Destroys the command object in the CommandObject (see page 995) property.

Destroys the parameter collection in the Parameters (see page 999) property.

Calls the inherited destructor.

See Also

Free (see page 22300)

8.1.3.4 TADOCommand Methods

The methods of the TADOCommand class are listed here.

Protected Methods

	Name	Description
	AssignCommandText (see page 990)	This is AssignCommandText, a member of class TADOCommand.
	CheckCommandText (see page 991)	This is CheckCommandText, a member of class TADOCommand.
	ClearActiveConnection (see page 991)	This is ClearActiveConnection, a member of class TADOCommand.

	ComponentLoading (see page 992)	This is ComponentLoading, a member of class TADOCommand.
	ConnectionStateChange (see page 992)	This is ConnectionStateChange, a member of class TADOCommand.
V	Initialize (see page 993)	This is Initialize, a member of class TADOCommand.
V	OpenConnection (see page 993)	This is OpenConnection, a member of class TADOCommand.
V	SetCommandText (see page 994)	This is SetCommandText, a member of class TADOCommand.
V	SetConnectionFlag (see page 994)	This is SetConnectionFlag, a member of class TADOCommand.

Public Methods

	Name	Description
V	Assign (see page 990)	Makes one ADO command component a duplicate of another.
	Cancel (see page 991)	Halts execution of an ADO command.
	Execute (see page 992)	This is the overview for the Execute method overload.

8.1.3.4.1 TADOCommand.Assign Method

Makes one ADO command component a duplicate of another.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the property values from one ADO command object to the properties of the current ADO command object.

Source is the ADO command object from which property values are copied. Source must be another TADOCommand (see page 984) object or an instance of a descendant class.

```
ADOCommand1.Assign(ADOCommand2);
ADOCommand1->Assign(ADOCommand2);
```

Assign copies the values of the following properties specific to ADO command objects:

Connection (see page 997), if assigned; otherwise the ConnectionString (see page 998) property is copied.

CommandText (see page 995).

CommandTimeout (see page 996).

CommandType (see page 997).

Prepared (see page 1000).

Parameters (see page 999).

8.1.3.4.2 TADOCommand.AssignCommandText Method

Pascal

```
procedure AssignCommandText(const Value: WideString; Loading: Boolean = False);
```

C++

```
__fastcall AssignCommandText(const BSTR Value, Boolean Loading = False);
```

Description

This is AssignCommandText, a member of class TADOCommand.

8.1.3.4.3 TADOCommand.Cancel Method

Halts execution of an ADO command.

Pascal

```
procedure Cancel;
```

C++

```
__fastcall Cancel();
```

Description

Call Cancel abort the execution of a command executed by the command component. To use Cancel to abort a command, the command by have been executed asynchronously (the eoAsyncExecute TExecuteOption (see page 1299) must have been used). If the command is not asynchronous an exception is raised. Use the command component's ExecuteOptions (see page 999) to specify that a command is executed asynchronously.

Note: To successfully cancel a command, Cancel must be called before the number of seconds specified in CommandTimeout (see page 996) have expired.

See Also

Execute (see page 992)

ExecuteOptions (see page 999)

TExecuteOptions

CommandTimeout (see page 996)

Canceling Commands

8.1.3.4.4 TADOCommand.CheckCommandText Method

Pascal

```
procedure CheckCommandText;
```

C++

```
__fastcall CheckCommandText();
```

Description

This is CheckCommandText, a member of class TADOCommand.

8.1.3.4.5 TADOCommand.ClearActiveConnection Method

Pascal

```
procedure ClearActiveConnection;
```

C++

```
__fastcall ClearActiveConnection();
```

Description

This is ClearActiveConnection, a member of class TADOCommand.

8.1.3.4.6 TADOCommand.ComponentLoading Method

Pascal

```
function ComponentLoading: Boolean;
```

C++

```
__fastcall Boolean ComponentLoading();
```

Description

This is ComponentLoading, a member of class TADOCommand.

8.1.3.4.7 TADOCommand.ConnectionStateChange Method

Pascal

```
procedure ConnectionStateChange(Sender: TObject; Connecting: Boolean);
```

C++

```
__fastcall ConnectionStateChange(TObject * Sender, Boolean Connecting);
```

Description

This is ConnectionStateChange, a member of class TADOCommand.

8.1.3.4.8 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
	TADOCommand.Execute () (see page 992)	Causes the ADO command object's command to be executed.
	TADOCommand.Execute (Integer, OleVariant) (see page 992)	Causes the ADO command object's command to be executed.
	TADOCommand.Execute (OleVariant) (see page 992)	Causes the ADO command object's command to be executed.

8.1.3.4.8.1 TADOCommand.Execute Method ()

Causes the ADO command object's command to be executed.

Pascal

```
function Execute: _Recordset; overload;
function Execute(const Parameters: OleVariant): _Recordset; overload;
function Execute(var RecordsAffected: Integer; const Parameters: OleVariant): _Recordset;
overload;
```

C++

```
__fastcall _Recordset Execute();
__fastcall _Recordset Execute(const OleVariant Parameters);
__fastcall _Recordset Execute(int RecordsAffected, const OleVariant Parameters);
```

Description

Call Execute to immediately execute the command specified in the CommandText (see page 995) property.

RecordsAffected indicates the number of records, if the command operates on data, that are affected by the command after execution.

Parameters (see page 999) is a collection of parameters for the command, unnecessary if the command does not use any parameters.

When a command is executed that creates a recordset, Execute returns the recordset and it must be accessed through an ADO dataset component. To do this, assign the return value of Execute directly to the recordset property of an ADO data set component. For example:

```
ADODataSet1.Recordset := ADOCommand1.Execute;
ADODataSet1->Recordset = ADOCommand1->Execute();
```

See Also

TParameters (see page 1261)

Cancel (see page 991)

CommandObject (see page 995)

CommandType (see page 997)

CommandText (see page 995)

Canceling Commands

Handling Command Parameters

Retrieving Result Sets with Commands

Using the Execute Method

8.1.3.4.9 TADOCommand.Initialize Method

Pascal

```
procedure Initialize(DoAppend: Boolean = True); virtual;
```

C++

```
virtual __fastcall Initialize(Boolean DoAppend = True);
```

Description

This is Initialize, a member of class TADOCommand.

8.1.3.4.10 TADOCommand.OpenConnection Method

Pascal

```
procedure OpenConnection; virtual;
```

C++

```
virtual __fastcall OpenConnection();
```

Description

This is OpenConnection, a member of class TADOCommand.

8.1.3.4.11 TADOCommand.SetCommandText Method

Pascal

```
procedure SetCommandText(const Value: WideString); virtual;
```

C++

```
virtual __fastcall SetCommandText(const BSTR Value);
```

Description

This is SetCommandText, a member of class TADOCommand.

8.1.3.4.12 TADOCommand.SetConnectionFlag Method

Pascal

```
function SetConnectionFlag(Flag: Integer; Value: Boolean): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean SetConnectionFlag(int Flag, Boolean Value);
```

Description

This is SetConnectionFlag, a member of class TADOCommand.

8.1.3.5 TADOCommand Properties

The properties of the TADOCommand class are listed here.

Protected Properties

	Name	Description
R	ActiveConnection (see page 995)	This is ActiveConnection, a member of class TADOCommand.
	CommandTextAlias (see page 996)	This is CommandTextAlias, a member of class TADOCommand.
	ComponentRef (see page 997)	This is ComponentRef, a member of class TADOCommand.

Public Properties

	Name	Description
R	CommandObject (see page 995)	Provides direct access to the ADO command object.
R	Properties (see page 1000)	This is Properties, a member of class TADOCommand.
R	States (see page 1001)	Indicates the current state of the ADO command component.

Published Properties

	Name	Description
	CommandText (see page 995)	Specifies command to execute.
	CommandTimeout (see page 996)	Specifies the amount of time to attempt execution of the command.
	CommandType (see page 997)	Specifies the type of command to execute.
	Connection (see page 997)	Specifies ADO connection component to use.
	ConnectionString (see page 998)	Specifies the connection information for the database.
	ExecuteOptions (see page 999)	Specifies the characteristics of a command execution.
	ParamCheck (see page 999)	Specifies whether the parameter list for an ADO command component is regenerated if the SQL changes at runtime.
	Parameters (see page 999)	Contains the parameters for an ADO command object's SQL statement.
	Prepared (see page 1000)	Specifies whether the command is prepared before execution.

8.1.3.5.1 TADOCommand.ActiveConnection Property

Pascal

```
property ActiveConnection: _Connection;
```

C++

```
__property _Connection ActiveConnection;
```

Description

This is ActiveConnection, a member of class TADOCommand.

8.1.3.5.2 TADOCommand.CommandObject Property

Provides direct access to the ADO command object.

Pascal

```
property CommandObject: _Command;
```

C++

```
__property _Command CommandObject;
```

Description

Use CommandObject to get direct access to the ADO command object the command component represents. Accessing the underlying command object is especially useful for utilizing properties and methods not surfaced in TADOCommand (see page 984).

Note: Use of CommandObject to directly access the underlying ADO command object requires a good working knowledge of ADO objects in general and the ADO command object in specific. It is not recommended that you use the command object directly unless familiar with command object operations. Consult the Microsoft Data Store SDK help for specific information on using ADO command objects.

See Also

CommandText (see page 995)

CommandTimeout (see page 996)

CommandType (see page 997)

Execute (see page 992)

8.1.3.5.3 TADOCommand.CommandText Property

Specifies command to execute.

Pascal

```
property CommandText: WideString;
```

C++

```
__property BSTR CommandText;
```

Description

Use CommandText to specify the command to execute using the ADO command component. CommandText is a textual representation of the command such as an SQL statement, a table name, or the name of a stored procedure to execute.

```
ADOCommand1.CommandText := 'DROP TABLE Employee1986';
```

```
ADOCommand1->CommandText = "DROP TABLE Employee1986";
```

If the command includes parameters, as might be the case with an SQL statement or stored procedure, access the them through the Parameters (see page 999) property.

See Also

CommandObject (see page 995)

CommandTimeout (see page 996)

CommandType (see page 997)

Execute (see page 992)

Parameters (see page 999)

Specifying the Command

8.1.3.5.4 TADOCommand.CommandTextAlias Property

Pascal

```
property CommandTextAlias: string;
```

C++

```
__property AnsiString CommandTextAlias;
```

Description

This is CommandTextAlias, a member of class TADOCommand.

8.1.3.5.5 TADOCommand.CommandTimeout Property

Specifies the amount of time to attempt execution of the command.

Pascal

```
property CommandTimeout: Integer;
```

C++

```
__property int CommandTimeout;
```

Description

Use CommandTimeout to indicate the time, in seconds, that can elapse before an attempt to execute the command is considered unsuccessful. The default value for CommandTimeout is 30.

If a command is successfully executed prior to the expiration of the seconds specified, CommandTimeout has no effect.

See Also

CommandText (see page 995)

CommandObject (see page 995)

CommandType (see page 997)

Execute (see page 992)

Cancel (see page 991)

Canceling Commands

8.1.3.5.6 TADOCommand.CommandType Property

Specifies the type of command to execute.

Pascal

```
property CommandType: TCommandType;
```

C++

```
__property TCommandType CommandType;
```

Description

Set CommandType to indicate the type of command that is contained in the CommandText (see page 995) property. The value in CommandType should agree with that of CommandText (see page 995). For instance, if CommandText (see page 995) is the name of a table, CommandType should be cmdTable or cmdTableDirect.

The default value of CommandType is cmdUnknown. While CommandType may be left as cmdUnknown for all commands, setting CommandType to explicitly indicate the type of the command results in better performance. When CommandType is cmdUnknown, ADO must evaluate the command to determine its type, which slows the operation.

Note: The TCommandType (see page 1291) constants cmdTable, cmdTableDirect, and cmdOpenFile should not be used with commands executed from a TADOCommand (see page 984) component.

See Also

CommandText (see page 995)

CommandObject (see page 995)

CommandTimeout (see page 996)

Execute (see page 992)

CommandType

Specifying the Command

8.1.3.5.7 TADOCommand.ComponentRef Property

Pascal

```
property ComponentRef: TComponent;
```

C++

```
__property TComponent * ComponentRef;
```

Description

This is ComponentRef, a member of class TADOCommand.

8.1.3.5.8 TADOCommand.Connection Property

Specifies ADO connection component to use.

Pascal

```
property Connection: TADOConnection;
```

C++

```
__property TADOConnection Connection;
```

Description

Set Connection to specify the TADOConnection (see page 1001) the command component uses to connect to the data store.

```
ADOCommand1.Connection := ADOConnection1;
ADOCommand1->Connection = ADOConnection1;
```

Note: the connection of an ADO command component to a data store can either be through a TADOConnection (see page 1001) specified in the Connection property or directly through the connection string specified in ConnectionString (see page 998). These two properties are mutually exclusive.

See Also

ConnectionString (see page 998)

Connecting to a Data Store Using TADOConnection

8.1.3.5.9 TADOCommand.ConnectionString Property

Specifies the connection information for the database.

Pascal

```
property ConnectionString: WideString;
```

C++

```
__property BSTR ConnectionString;
```

Description

Set ConnectionString to specify the information needed to connect the command component to the data store. The value used for ConnectionString consists of one or more arguments ADO uses to establish the connection. Specify multiple arguments as a list with individual arguments separated by semicolons.

```
ADOCommand1.ConnectionString := 'Provider=YourProviderName;Remote Server=YourServerName';
ADOCommand1->ConnectionString = "Provider=YourProviderName;Remote Server=YourServerName";
```

At design-time, select from the list of available ADO database connections invoked from the Object Inspector.

A connection string may be saved to file for later use. Specify only the name of such a file in ConnectionString to reuse a saved connection string.

The connection string may also contain login information such as user ID and password, for automated logins.

ADO supports the following four arguments for connection strings. Any other arguments (such as a user ID and password) are not processed by ADO and simply passed on to the provider.

Argument	Meaning
Provider (see page 18419)	The name of the provider to use for the connection.
File name	The name of a file containing connection information.
Remote Provider (see page 18419)	The name of the provider to use for a client-side connection.
Remote Server	The path name of the server to use for a client-side connection.

Note: the connection of an ADO command component to a database can either be through a TADOConnection (see page 1001) specified in the Connection (see page 997) property or directly through the connection string specified in ConnectionString. These two means for connecting to a database are mutually exclusive.

Note: After opening a connection, the contents of ConnectionString may be changed by ADO. One example of this behavior is when ADO-defined arguments are mapped to their provider equivalents.

See Also

Connection (see page 997)

8.1.3.5.10 TADOCommand.ExecuteOptions Property

Specifies the characteristics of a command execution.

Pascal

```
property ExecuteOptions: TExecuteOptions;
```

C++

```
__property TExecuteOptions ExecuteOptions;
```

Description

Set ExecuteOptions to control the characteristics of a command executed from the ADO dataset component.

8.1.3.5.11 TADOCommand.ParamCheck Property

Specifies whether the parameter list for an ADO command component is regenerated if the SQL changes at runtime.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

Set ParamCheck to specify whether or not the Parameters (see page 999) property is initialized using ":Param" style parameters in an SQL statement specified in the CommandText (see page 995) property. If the SQL statement in CommandText (see page 995) only contains "?" style parameters, the Parameters (see page 999) property will only be initialized if the OLE DB (see page 6725) provider returns parameter information for an SQL command. Otherwise, the parameters must be explicitly created using the CreateParameter method (TParameters (see page 1261)).

When ParamCheck is true, a parameter is generated for each ":Param" style parameter in the SQL statement.

The default value of ParamCheck is true.

This property is useful for data definition language (DDL) statements that contain parameters as part of the DDL statement and that are not parameters for the ADO command component. For example, the DDL statement to create a stored procedure may contain parameter statements that are part of the stored procedure. Set ParamCheck to false to prevent these parameters from being mistaken for parameters of the ADO command component executing the DDL statement.

See Also

Parameters (see page 999)

CommandText (see page 995)

CreateParameter (see page 1266)

8.1.3.5.12 TADOCommand.Parameters Property

Contains the parameters for an ADO command object's SQL statement.

Pascal

```
property Parameters: TParameters;
```

C++

```
__property TParameters Parameters;
```

Description

Access Parameters at runtime to view and set parameter names, values, and data types dynamically (at design time use the collection editor for the Parameters property to set parameter information). Parameters is a zero-based array of TParameter (see page 1250) parameter records. Index specifies the array element to access.

Note: Parameters is only applicable when the ADO command object issues an SQL statement or executes a stored procedure (the CommandType (see page 997) is cmdText or cmdStoredProc) and parameters are actually used in the SQL statement or stored procedure.

See Also

Prepared (see page 1000)

Handling Command Parameters

8.1.3.5.13 TADOCommand.Prepared Property

Specifies whether the command is prepared before execution.

Pascal

```
property Prepared: WordBool;
```

C++

```
__property WordBool Prepared;
```

Description

Set Prepared before calling the Execute (see page 992) method to specify whether ADO prepares the command. If Prepared is set to true and the command component is connected to a data store, ADO prepares the command before executing it. If Prepared is set to false, ADO does not prepare the command.

The default value of Prepared is false.

See Also

Parameters (see page 999)

Execute (see page 992)

8.1.3.5.14 TADOCommand.Properties Property

Pascal

```
property Properties: Properties;
```

C++

```
__property Properties Properties;
```

Description

This is Properties, a member of class TADOCommand.

8.1.3.5.15 TADOCommand.States Property

Indicates the current state of the ADO command component.

Pascal

```
property States: TObjectStates;
```

C++

```
__property TObjectStates States;
```

Description

Read States at runtime to determine the activity being performed, if any, by the command component at a specific point in time. Activity status includes the command component connecting to a data store, executing a command, fetching rows for a recordset, sitting idle but open, and sitting idle and closed.

See Also

Connection (see page 997)

Execute (see page 992)

8.1.4 TADOConnection Class

TADOConnection connects to an ADO data store.

Class Hierarchy

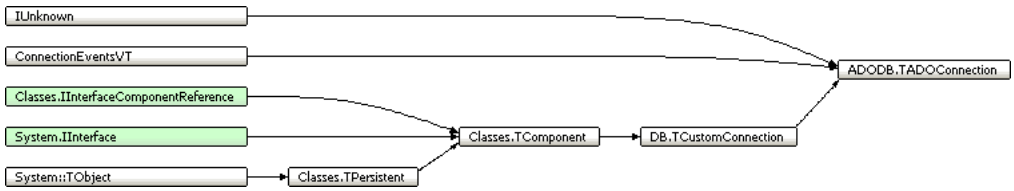

Pascal

```
TADOConnection = class(TCustomConnection, IUnknown, ConnectionEventsVT);
```

C++

```
class TADOConnection : public TCustomConnection, public IUnknown, public ConnectionEventsVT;
```

File

ADODB

Description

TADOConnection encapsulates the ADO connection object. Use TADOConnection for connecting to ADO data stores. The connection provided by a single TADOConnection component can be shared by multiple ADO command and dataset components through their Connection properties.

TADOConnection allows you to control the attributes and conditions of a connection to a data store. Use the properties of TADOConnection to control such attributes as record locking scheme (optimistic versus pessimistic), cursor type, cursor location, isolation level, and connection time-out. Methods are also provided for implementing transactions and retrieving metadata about the database to which this component connects.

See Also

TADOCOmmand

TCustomADONETConnector

TADODataSet (see page 1041)

TADOTable (see page 1125)

TADOQuery (see page 1084)

TADOStoredProc (see page 1106)

Connection

Connecting to Databases: Overview

Connecting to a Data Store Using TADOConnection

Connecting to ADO Data Stores

Overview of ADO Components

Connecting Directly to a Database Server

Building a VCL Forms ADO.NET Database Application

8.1.4.1 TADOConnection Members

The following tables list the members exposed by TADOConnection.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BeginTransComplete (see page 1008)	This is BeginTransComplete, a member of class TADOConnection.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckActive (see page 1009)	This is CheckActive, a member of class TADOConnection.
V	CheckDisconnect (see page 1009)	This is CheckDisconnect, a member of class TADOConnection.
	CheckInactive (see page 1010)	This is CheckInactive, a member of class TADOConnection.
	CommitTransComplete (see page 1010)	This is CommitTransComplete, a member of class TADOConnection.
	ConnectComplete (see page 1010)	This is ConnectComplete, a member of class TADOConnection.
	ConnectionPoint (see page 1011)	This is ConnectionPoint, a member of class TADOConnection.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	Disconnect (see page 1011)	This is Disconnect, a member of class TADOConnection.
V	DoConnect (see page 1011)	Provides the interface for a method that opens a connection.
V	DoDisconnect (see page 1011)	Provides the interface for a method that terminates the connection.
	ExecuteComplete (see page 1013)	This is ExecuteComplete, a member of class TADOConnection.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.

	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetConnected (see page 1013)	Returns the value of the Connected property.
	GetDataSet (see page 6865)	Returns a dataset, given its index in the DataSets (see page 6870) property array.
	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.
	GetErrors (see page 1013)	This is GetErrors, a member of class TADOConnection.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InfoMessage (see page 1016)	This is InfoMessage, a member of class TADOConnection.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 1016)	Initializes the connection component after all objects in the form or data module have been loaded.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RegisterClient (see page 1017)	Registers an object that uses the connection component.
	RollbackTransComplete (see page 1018)	This is RollbackTransComplete, a member of class TADOConnection.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnected (see page 6867)	Changes the value of the Connected (see page 6868) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	UnRegisterClient (see page 1019)	Removes a registered client so that it is no longer associated with the connection component.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WaitForConnectComplete (see page 1019)	This is WaitForConnectComplete, a member of class TADOConnection.
	WillConnect (see page 1019)	This is WillConnect, a member of class TADOConnection.
	WillExecute (see page 1020)	This is WillExecute, a member of class TADOConnection.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginTrans (see page 1008)	Initiates a new transaction in the associated database.
	Cancel (see page 1009)	Aborts an attempt to connect to a data store.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 6863)	Closes the connection.
	CommitTrans (see page 1010)	Commits an open transaction.
V	Create (see page 1006)	Constructs an instance of the button control and initializes its values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 1006)	Destroys the bitmap button object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 1012)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldNames (see page 1014)	This is the overview for the GetFieldNames method overload.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProcedureNames (see page 1014)	This is the overview for the GetProcedureNames method overload.
	GetTableNames (see page 1015)	This is the overview for the GetTableNames method overload.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 1016)	Initiates a connection to a database.
	OpenSchema (see page 1017)	Retrieves schema information from the connection's provider.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RollbackTrans (see page 1018)	Rolls back an active transaction.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	CommandCount (see page 1021)	Indicates the number of command components associated with the connection.
R	Commands (see page 1021)	Lists all active commands for the connection component.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	ConnectionObject (see page 1023)	Provides direct access to the ADO connection object.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 1027)	Provides an indexed array of all active datasets for a connection component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Errors (see page 1028)	Errors is the implementation of the Errors Collection.
R	InTransaction (see page 1028)	Indicates whether a transaction is in progress.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Properties (see page 1032)	This is Properties, a member of class TADOConnection.
R	State (see page 1032)	Indicates the current state of the ADO connection.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Version (see page 1033)	Indicates the version of ADO used.

Published Events

	Name	Description
	AfterConnect (see page 1034)	Occurs after a connection is established.
	AfterDisconnect (see page 1034)	Occurs after the connection closes.
	BeforeConnect (see page 1034)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 1035)	Occurs immediately before the connection closes.
	OnBeginTransComplete (see page 1035)	Occurs after beginning a transaction.
	OnCommitTransComplete (see page 1036)	Occurs after committing a transaction.
	OnConnectComplete (see page 1036)	Occurs after starting a connection.
	OnDisconnect (see page 1037)	Occurs after a connection is terminated.
	OnExecuteComplete (see page 1038)	Occurs after a command is executed.
	OnInfoMessage (see page 1038)	Occurs when information messages are received from the data store connection.

	OnRollbackTransComplete (see page 1039)	Occurs after rolling back a transaction.
	OnWillConnect (see page 1039)	Occurs after a request to start a connection.
	OnWillExecute (see page 1040)	Occurs after a database server signals acceptance of a command execution.

Published Properties

	Name	Description
	Attributes (see page 1020)	Specifies automated transaction behavior.
	CommandTimeout (see page 1022)	Specifies amount of time to attempt execution of a command.
	Connected (see page 1022)	Specifies whether the connection is active.
	ConnectionString (see page 1024)	Specifies the connection information for the data store.
	ConnectionTimeout (see page 1025)	Specifies amount of time to attempt a connection.
	ConnectOptions (see page 1026)	Specifies whether a connection is synchronous or asynchronous.
	CursorLocation (see page 1026)	Specifies whether cursors for the connection is client-side or server-side.
	DefaultDatabase (see page 1027)	Indicates the database the ADO connection uses by default.
	IsolationLevel (see page 1029)	Specifies the transaction isolation level for transactions.
	KeepConnection (see page 1029)	Specifies whether an application remains connected to a database even if no datasets are open.
	LoginPrompt (see page 1029)	Specifies whether a login dialog appears immediately before opening a new connection.
	Mode (see page 1031)	Indicate the persmissions available to a connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OnLogin (see page 1031)	Occurs when a communication channel opens to the server.
	Provider (see page 1032)	Specifies the provider for the ADO connection.
	Tag (see page 2808)	Stores an integer value as part of a component.

8.1.4.2 TADOConnection.Create Constructor

Constructs an instance of the button control and initializes its values.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TADOConnection(TComponent * AOwner);
```

Description

Use Create to programmatically instantiate an ADO connection. ADO connections added in the form designer are created automatically.

Create

Calls its inherited Create method.

Initializes its connection interface.

Initializes is list of associated datasets.

See Also

Destroy_d

8.1.4.3 TADOConnection.Destroy Destructor

Destroys the bitmap button object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TADOConnection();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the ADO connection is not nil, and only then calls Destroy.

Destroy performs the following tasks:

Closes the connection.

Destroys its list of associated datasets.

Checks for unprocessed connection events.

Destroys its connection interface.

Calls the inherited destructor.

See Also

Create_d

Free (see page 22300)

8.1.4.4 TADOConnection Methods

The methods of the TADOConnection class are listed here.

Protected Methods

	Name	Description
	BeginTransComplete (see page 1008)	This is BeginTransComplete, a member of class TADOConnection.
	CheckActive (see page 1009)	This is CheckActive, a member of class TADOConnection.
V	CheckDisconnect (see page 1009)	This is CheckDisconnect, a member of class TADOConnection.
	CheckInactive (see page 1010)	This is CheckInactive, a member of class TADOConnection.
	CommitTransComplete (see page 1010)	This is CommitTransComplete, a member of class TADOConnection.
	ConnectComplete (see page 1010)	This is ConnectComplete, a member of class TADOConnection.
	ConnectionPoint (see page 1011)	This is ConnectionPoint, a member of class TADOConnection.
	Disconnect (see page 1011)	This is Disconnect, a member of class TADOConnection.
V	DoConnect (see page 1011)	Provides the interface for a method that opens a connection.
V	DoDisconnect (see page 1011)	Provides the interface for a method that terminates the connection.
	ExecuteComplete (see page 1013)	This is ExecuteComplete, a member of class TADOConnection.
V	GetConnected (see page 1013)	Returns the value of the Connected property.
	GetErrors (see page 1013)	This is GetErrors, a member of class TADOConnection.
	InfoMessage (see page 1016)	This is InfoMessage, a member of class TADOConnection.
V	Loaded (see page 1016)	Initializes the connection component after all objects in the form or data module have been loaded.
V	RegisterClient (see page 1017)	Registers an object that uses the connection component.
	RollbackTransComplete (see page 1018)	This is RollbackTransComplete, a member of class TADOConnection.
V	UnRegisterClient (see page 1019)	Removes a registered client so that it is no longer associated with the connection component.
V	WaitForConnectComplete (see page 1019)	This is WaitForConnectComplete, a member of class TADOConnection.
	WillConnect (see page 1019)	This is WillConnect, a member of class TADOConnection.
	WillExecute (see page 1020)	This is WillExecute, a member of class TADOConnection.

Public Methods

	Name	Description
	BeginTrans (see page 1008)	Initiates a new transaction in the associated database.
	Cancel (see page 1009)	Aborts an attempt to connect to a data store.
	CommitTrans (see page 1010)	Commits an open transaction.
	Execute (see page 1012)	This is the overview for the Execute method overload.
	GetFieldNames (see page 1014)	This is the overview for the GetFieldNames method overload.
	GetProcedureNames (see page 1014)	This is the overview for the GetProcedureNames method overload.
	GetTableNames (see page 1015)	This is the overview for the GetTableNames method overload.
	Open (see page 1016)	Initiates a connection to a database.
	OpenSchema (see page 1017)	Retrieves schema information from the connection's provider.
	RollbackTrans (see page 1018)	Rolls back an active transaction.

8.1.4.4.1 TADOConnection.BeginTrans Method

Initiates a new transaction in the associated database.

Pascal

```
function BeginTrans: Integer;
```

C++

```
__fastcall int BeginTrans();
```

Description

Call BeginTrans to start a new transaction in the data store the ADO connection component is connected to.

BeginTrans returns a value of type integer, indicating the nesting level of the new transaction.

A successful execution of BeginTrans triggers an OnBeginTransComplete (see page 1035) event and sets the InTransaction (see page 1028) property to true.

Note: The ADO connection object must have an active connection before BeginTrans can be used.

See Also

CommitTrans (see page 1010)

RollbackTrans (see page 1018)

Connected (see page 1022)

Open (see page 1016)

OnBeginTransComplete (see page 1035)

InTransaction (see page 1028)

8.1.4.4.2 TADOConnection.BeginTransComplete Method

Pascal

```
procedure BeginTransComplete(TransactionLevel: Integer; const pError: Error; var adStatus:
EventStatusEnum; const pConnection: _Connection); safecall;
```

C++

```
__fastcall BeginTransComplete(int TransactionLevel, const Error pError, EventStatusEnum
adStatus, const _Connection pConnection);
```

Description

This is BeginTransComplete, a member of class TADOConnection.

8.1.4.4.3 TADOConnection.Cancel Method

Aborts an attempt to connect to a data store.

Pascal

```
procedure Cancel;
```

C++

```
__fastcall Cancel();
```

Description

Call Cancel to abort an attempt to connect to an ADO data store. To use cancel, the connection attempted must be asynchronous (the ConnectOptions (see page 1026) property must be set to coAsyncConnect).

Cancel must be used after an attempt to connect is initiated by calling the Open (see page 1016) method (or setting Connected (see page 1022) to true) and before the connection has been successfully made or has timed out.

See Also

Open (see page 1016)

Connected (see page 1022)

ConnectionTimeout (see page 1025)

ConnectOptions (see page 1026)

8.1.4.4.4 TADOConnection.CheckActive Method

Pascal

```
procedure CheckActive;
```

C++

```
__fastcall CheckActive();
```

Description

This is CheckActive, a member of class TADOConnection.

8.1.4.4.5 TADOConnection.CheckDisconnect Method

Pascal

```
procedure CheckDisconnect; virtual;
```

C++

```
virtual __fastcall CheckDisconnect();
```

Description

This is CheckDisconnect, a member of class TADOConnection.

8.1.4.4.6 TADOConnection.CheckInactive Method

Pascal

```
procedure CheckInactive;
```

C++

```
__fastcall CheckInactive();
```

Description

This is CheckInactive, a member of class TADOConnection.

8.1.4.4.7 TADOConnection.CommitTrans Method

Commits an open transaction.

Pascal

```
procedure CommitTrans;
```

C++

```
__fastcall CommitTrans();
```

Description

Call CommitTrans to save any changes made during the current transaction and to end the transaction.

A successful execution of CommitTrans triggers an OnCommitTransComplete (see page 1036) event and sets the InTransaction (see page 1028) property to false.

See Also

BeginTrans (see page 1008)

RollbackTrans (see page 1018)

OnCommitTransComplete (see page 1036)

InTransaction (see page 1028)

8.1.4.4.8 TADOConnection.CommitTransComplete Method

Pascal

```
procedure CommitTransComplete(const pError: Error; var adStatus: EventStatusEnum; const
pConnection: _Connection); safecall;
```

C++

```
__fastcall CommitTransComplete(const Error pError, EventStatusEnum adStatus, const _Connection
pConnection);
```

Description

This is CommitTransComplete, a member of class TADOConnection.

8.1.4.4.9 TADOConnection.ConnectComplete Method

Pascal

```
procedure ConnectComplete(const pError: Error; var adStatus: EventStatusEnum; const
pConnection: _Connection); safecall;
```

C++

```
__fastcall ConnectComplete(const Error pError, EventStatusEnum adStatus, const _Connection
pConnection);
```

Description

This is ConnectComplete, a member of class TADOConnection.

8.1.4.4.10 TADOConnection.ConnectionPoint Method

Pascal

```
function ConnectionPoint: IConnectionPoint;
```

C++

```
__fastcall IConnectionPoint ConnectionPoint();
```

Description

This is ConnectionPoint, a member of class TADOConnection.

8.1.4.4.11 TADOConnection.Disconnect Method

Pascal

```
procedure Disconnect(var adStatus: EventStatusEnum; const pConnection: _Connection); safecall;
```

C++

```
__fastcall Disconnect(EventStatusEnum adStatus, const _Connection pConnection);
```

Description

This is Disconnect, a member of class TADOConnection.

8.1.4.4.12 TADOConnection.DoConnect Method

Provides the interface for a method that opens a connection.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

The Connected property uses DoConnect to establish a connection. Descendant classes override the DoConnect method to establish their connection as appropriate. As implemented in TCustomConnection, DoConnect does nothing.

See Also

Connected (see page 6868)

DoDisconnect (see page 6864)

8.1.4.4.13 TADOConnection.DoDisconnect Method

Provides the interface for a method that terminates the connection.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

The Connected property uses DoDisconnect to close a connection. Descendant classes override the DoDisconnect method to drop a connection. As implemented in TCustomConnection, DoDisconnect does nothing.

See Also

Connected (see page 6868)

DoConnect (see page 6864)

8.1.4.4.14 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
	TADOConnection.Execute (WideString, Integer, TExecuteOptions) (see page 1012)	Executes a command.
	TADOConnection.Execute (WideString, TCommandType, TExecuteOptions) (see page 1012)	Executes a command.

8.1.4.4.14.1 TADOConnection.Execute Method (WideString, Integer, TExecuteOptions)

Executes a command.

Pascal

```
procedure Execute(const CommandText: WideString; var RecordsAffected: Integer; const
ExecuteOptions: TExecuteOptions = [eoExecuteNoRecords]); overload;
function Execute(const CommandText: WideString; const CommandType: TCommandType = cmdText;
const ExecuteOptions: TExecuteOptions = []): _Recordset; overload;
```

C++

```
__fastcall Execute(const BSTR CommandText, int RecordsAffected, const TExecuteOptions
ExecuteOptions = [eoExecuteNoRecords]);
__fastcall _Recordset Execute(const BSTR CommandText, const TCommandType CommandType =
cmdText, const TExecuteOptions ExecuteOptions = []);
```

Description

Call Execute to execute a command using the ADO connection object.

CommandText is the command to execute.

ExecuteOptions is a TExecuteOptions (see page 1316) value that specifies the characteristics of the command execution.

Execute returns a recordset if the command executed is one that generates a recordset.

RecordsAffected indicates the number of records, if the command operates on data, that are affected by the command after execution.

See Also

GetTableNames (see page 1015)

OnWillExecute (see page 1040)

OnExecuteComplete (see page 1038)

8.1.4.4.15 TADOConnection.ExecuteComplete Method

Pascal

```
procedure ExecuteComplete(RecordsAffected: Integer; const pError: Error; var adStatus:
EventStatusEnum; const pCommand: _Command; const pRecordset: _Recordset; const pConnection:
_Connection); safecall;
```

C++

```
__fastcall ExecuteComplete(int RecordsAffected, const Error pError, EventStatusEnum adStatus,
const _Command pCommand, const _Recordset pRecordset, const _Connection pConnection);
```

Description

This is ExecuteComplete, a member of class TADOConnection.

8.1.4.4.16 TADOConnection.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected read implementation of the Connected property. Descendant classes override this method to return true if a connection is open and false otherwise. As defined in TCustomConnection, GetConnected always returns false.

See Also

Connected (see page 6868)

SetConnected (see page 6867)

8.1.4.4.17 TADOConnection.GetErrors Method

Pascal

```
function GetErrors: Errors;
```

C++

```
__fastcall Errors GetErrors();
```

Description

This is GetErrors, a member of class TADOConnection.

8.1.4.4.18 GetFieldNames Method

This is the overview for the GetFieldNames method overload.

Overload List

	Name	Description
	TADOConnection.GetFieldNames (string, TStrings) (see page 1014)	Populates a string list with the names of fields in a table.
	TADOConnection.GetFieldNames (WideString, TWideStrings) (see page 1014)	Populates a string list with the names of fields in a table.

8.1.4.4.18.1 TADOConnection.GetFieldNames Method (WideString, TWideStrings)

Populates a string list with the names of fields in a table.

Pascal

```
procedure GetFieldNames(const TableName: WideString; List: TWideStrings); overload;
procedure GetFieldNames(const TableName: string; List: TStrings); overload;
```

C++

```
__fastcall GetFieldNames(const BSTR TableName, TWideStrings List);
__fastcall GetFieldNames(const AnsiString TableName, TStrings List);
```

Description

Call GetFieldNames to retrieve a list of fields in a table. The names of the fields are put into the already-existing string list object specified in the List parameter. Specify the table for which to retrieve the names of fields in the TableName property.

```
ADOConnection1.GetFieldNames('Employee', ListBox1.Items);
ADOConnection1->GetFieldNames("Employee", ListBox1->Items);
```

See Also

TStrings (see page 3029)

GetProcedureNames (see page 1014)

GetTableNames (see page 1015)

8.1.4.4.19 GetProcedureNames Method

This is the overview for the GetProcedureNames method overload.

Overload List

	Name	Description
	TADOConnection.GetProcedureNames (TStrings) (see page 1014)	Populates a string list with the names of stored procedures in the database.
	TADOConnection.GetProcedureNames (TWideStrings) (see page 1014)	Populates a string list with the names of stored procedures in the database.

8.1.4.4.19.1 TADOConnection.GetProcedureNames Method (TStrings)

Populates a string list with the names of stored procedures in the database.

Pascal

```
procedure GetProcedureNames(List: TWideStrings); overload;
procedure GetProcedureNames(List: TStrings); overload;
```

C++

```
__fastcall GetProcedureNames(TWideStrings List);
__fastcall GetProcedureNames(TStrings List);
```

Description

Call GetProcedureNames to retrieve a list of stored procedures in the associated database. The names of the stored procedures are put into the already-existing string list object specified in the List parameter.

```
ADOConnection1.GetProcedureNames(ListBox1.Items);
ADOConnection1->GetProcedureNames(ListBox1->Items);
```

Note: Any contents already in the target string list object are eliminated and overwritten by the data produces by GetProcedureNames.

See Also

TStrings (see page 3029)

GetTableNames (see page 1015)

GetFieldNames (see page 1014)

8.1.4.4.20 GetTableNames Method

This is the overview for the GetTableNames method overload.

Overload List

	Name	Description
	TADOConnection.GetTableNames (TStrings, Boolean) (see page 1015)	Populates a string list with the names of tables in the database.
	TADOConnection.GetTableNames (TWideStrings, Boolean) (see page 1015)	Populates a string list with the names of tables in the database.

8.1.4.4.20.1 TADOConnection.GetTableNames Method (TStrings, Boolean)

Populates a string list with the names of tables in the database.

Pascal

```
procedure GetTableNames(List: TWideStrings; SystemTables: Boolean = False); overload;
procedure GetTableNames(List: TStrings; SystemTables: Boolean = False); overload;
```

C++

```
__fastcall GetTableNames(TWideStrings List, Boolean SystemTables = False);
__fastcall GetTableNames(TStrings List, Boolean SystemTables = False);
```

Description

Call GetTableNames to retrieve a list of tables in the associated database.

List is the already-existing string list object into which the tables names are put.

Set SystemTables to indicate whether the list of table names should include the database's system tables.

```
ADOConnection1.GetTableNames(ListBox2.Items, False);
ADOConnection1->GetTableNames(ListBox2->Items, false);
```

Note: Any contents already in the target string list object are eliminated and overwritten by the data produces by GetTableNames.

See Also

GetFieldNames (see page 1014)

8.1.4.4.21 TADOConnection.InfoMessage Method

Pascal

```
procedure InfoMessage(const pError: Error; var adStatus: EventStatusEnum; const pConnection:
_Connection); safecall;
```

C++

```
__fastcall InfoMessage(const Error pError, EventStatusEnum adStatus, const _Connection
pConnection);
```

Description

This is InfoMessage, a member of class TADOConnection.

8.1.4.4.22 TADOConnection.Loaded Method

Initializes the connection component after all objects in the form or data module have been loaded.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. It is called automatically after the form or data module that contains the connection component has loaded all its owned objects. TCustomConnection overrides the inherited method to establish a connection if the Connected property is true.

See Also

StreamedConnected (see page 6874)

Loaded (see page 2782)

8.1.4.4.23 TADOConnection.Open Method

Initiates a connection to a database.

Pascal

```
procedure Open(const UserID: WideString; const Password: WideString); overload;
```

C++

```
__fastcall Open(const BSTR UserID, const BSTR Password);
```

Description

Call Open to initiate a connection to the database specified in the ConnectionString (see page 1024) property.

UserID and Password parameters for Open can be optionally used to pass the user ID and login password to the database server at the same time the request is made to connect.

```
ADOConnection1.Open('Joe Sixpack', 'SecretWord');
ADOConnection1->Open("Joe Sixpack", "SecretWord");
```

Alternately, this login information may be passed in the value in ConnectionString (see page 1024).

When explicitly passing login information through either the Open method or the ConnectionString (see page 1024), the LoginPrompt (see page 1029) property should be set to false to prevent an unnecessary login dialog.

See Also

Connected (see page 1022)

ConnectionString (see page 1024)

ConnectionObject (see page 1023)

OnWillConnect (see page 1039)

OnConnectComplete (see page 1036)

8.1.4.4.24 TADOConnection.OpenSchema Method

Retrieves schema information from the connection's provider.

Pascal

```
procedure OpenSchema(const Schema: TSchemaInfo; const Restrictions: OleVariant; const
SchemaID: OleVariant; DataSet: TADODataSet);
```

C++

```
__fastcall OpenSchema(const TSchemaInfo Schema, const OleVariant Restrictions, const
OleVariant SchemaID, TADODataSet DataSet);
```

Description

Call OpenSchema to open a connection and retrieve schema information from the associated provider. OpenSchema retrieves the schema information in the form of a recordset that can then be accessed through an ADO dataset component.

The Schema parameter is the type of schema information to retrieve, or in ADO terms, the query type.

The Restrictions parameter is an array of query constraints for the specified type of schema information. This lists the type of information requested for the entity specified by Schema. In ADO terms, the elements of the array list the criteria for the query type.

SchemaID is the GUID for a provider-schema schema query not defined by the OLE DB (see page 6725) specification. This parameter is required if the Schema parameter is set to siProviderSpecific, otherwise it is not used.

DataSet is the ADO dataset component (such as a TADOTable (see page 1125)) into the Recordset property of which the schema information is retrieved and through which the information is subsequently accessed. After OpenSchema loads the ADO dataset component with the schema information, use the standard navigation methods like First and Next to traverse the rows of data. Use the Fields property or the FieldByName method of the dataset component to read the value of any one column. Alternately, the schema data may be displayed using visual data-aware controls and a TDataSource.

See Also

Open (see page 1016)

Close (see page 6863)

8.1.4.4.25 TADOConnection.RegisterClient Method

Registers an object that uses the connection component.

Pascal

```
procedure RegisterClient(Client: TObject; Event: TConnectChangeEvent = nil); override;
```

C++

```
virtual __fastcall RegisterClient(TObject * Client, TConnectChangeEvent Event = nil);
```

Description

RegisterClient is used internally to associate objects with the connection component. Registered components can supply an event handler that responds when the connection is created or terminated.

Client is the object to register. If Client is a dataset, it is added to the DataSets property.

Event is an event handler that is called after the connection is established or before it is terminated.

Datasets call this method when they start using the connection component. For BDE-enabled datasets, ADO datasets, and dbExpress datasets, this is when they are activated. For client datasets, this is when the RemoteServer property is assigned.

See Also

DataSets (see page 6870)

UnRegisterClient (see page 6868)

8.1.4.4.26 TADOConnection.RollbackTrans Method

Rolls back an active transaction.

Pascal

```
procedure RollbackTrans;
```

C++

```
__fastcall RollbackTrans();
```

Description

Call RollbackTrans to cancel any changes made during the current transaction and to end the transaction.

A successful execution of RollbackTrans triggers an OnRollbackTransComplete (see page 1039) event and sets the InTransaction (see page 1028) property to true.

See Also

BeginTrans (see page 1008)

CommitTrans (see page 1010)

OnRollbackTransComplete (see page 1039)

InTransaction (see page 1028)

8.1.4.4.27 TADOConnection.RollbackTransComplete Method

Pascal

```
procedure RollbackTransComplete(const pError: Error; var adStatus: EventStatusEnum; const
pConnection: _Connection); safecall;
```

C++

```
__fastcall RollbackTransComplete(const Error pError, EventStatusEnum adStatus, const
_Connection pConnection);
```

Description

This is RollbackTransComplete, a member of class TADOConnection.

8.1.4.4.28 TADOConnection.UnRegisterClient Method

Removes a registered client so that it is no longer associated with the connection component.

Pascal

```
procedure UnRegisterClient(Client: TObject); override;
```

C++

```
virtual __fastcall UnRegisterClient(TObject * Client);
```

Description

UnRegisterClient is used internally to remove objects from the internal list of objects that are informed when a connection is created or terminated.

Client is the object that is removed from the list. It should be an object that was previously registered by a call to RegisterClient. If Client is a dataset, it is removed from the DataSets property list.

Datasets call this method when they stop using the connection component. For BDE-enabled datasets, ADO datasets, and dbExpress datasets, this is when they are deactivated. For client datasets, this is when the RemoteServer property is changed.

See Also

DataSets (see page 6870)

RegisterClient (see page 6866)

8.1.4.4.29 TADOConnection.WaitForConnectComplete Method

Pascal

```
procedure WaitForConnectComplete; virtual;
```

C++

```
virtual __fastcall WaitForConnectComplete();
```

Description

This is WaitForConnectComplete, a member of class TADOConnection.

8.1.4.4.30 TADOConnection.WillConnect Method

Pascal

```
procedure WillConnect(var ConnectionString: WideString; var UserID: WideString; var Password:
WideString; var Options: Integer; var adStatus: EventStatusEnum; const pConnection:
_Connection); safecall;
```

C++

```
__fastcall WillConnect(BSTR ConnectionString, BSTR UserID, BSTR Password, int Options,
EventStatusEnum adStatus, const _Connection pConnection);
```

Description

This is WillConnect, a member of class TADOConnection.

8.1.4.4.31 TADOConnection.WillExecute Method

Pascal

```
procedure WillExecute(var Source: WideString; var CursorType: CursorTypeEnum; var LockType:
LockTypeEnum; var Options: Integer; var adStatus: EventStatusEnum; const pCommand: _Command;
const pRecordset: _Recordset; const pConnection: _Connection); safecall;
```

C++

```
__fastcall WillExecute(BSTR Source, CursorTypeEnum CursorType, LockTypeEnum LockType, int
Options, EventStatusEnum adStatus, const _Command pCommand, const _Recordset pRecordset, const
_Connection pConnection);
```

Description

This is WillExecute, a member of class TADOConnection.

8.1.4.5 TADOConnection Properties

The properties of the TADOConnection class are listed here.

Public Properties

	Name	Description
R	CommandCount (see page 1021)	Indicates the number of command components associated with the connection.
R	Commands (see page 1021)	Lists all active commands for the connection component.
	ConnectionObject (see page 1023)	Provides direct access to the ADO connection object.
R	DataSets (see page 1027)	Provides an indexed array of all active datasets for a connection component.
R	Errors (see page 1028)	Errors is the implementation of the Errors Collection.
R	InTransaction (see page 1028)	Indicates whether a transaction is in progress.
R	Properties (see page 1032)	This is Properties, a member of class TADOConnection.
R	State (see page 1032)	Indicates the current state of the ADO connection.
R	Version (see page 1033)	Indicates the version of ADO used.

Published Properties

	Name	Description
	Attributes (see page 1020)	Specifies automated transaction behavior.
	CommandTimeout (see page 1022)	Specifies amount of time to attempt execution of a command.
	Connected (see page 1022)	Specifies whether the connection is active.
	ConnectionString (see page 1024)	Specifies the connection information for the data store.
	ConnectionTimeout (see page 1025)	Specifies amount of time to attempt a connection.
	ConnectOptions (see page 1026)	Specifies whether a connection is synchronous or asynchronous.
	CursorLocation (see page 1026)	Specifies whether cursors for the connection is client-side or server-side.
	DefaultDatabase (see page 1027)	Indicates the database the ADO connection uses by default.
	IsolationLevel (see page 1029)	Specifies the transaction isolation level for transactions.
	KeepConnection (see page 1029)	Specifies whether an application remains connected to a database even if no datasets are open.
	LoginPrompt (see page 1029)	Specifies whether a login dialog appears immediately before opening a new connection.
	Mode (see page 1031)	Indicate the persmissions available to a connection.
	OnLogin (see page 1031)	Occurs when a communication channel opens to the server.
	Provider (see page 1032)	Specifies the provider for the ADO connection.

8.1.4.5.1 TADOConnection.Attributes Property

Specifies automated transaction behavior.

Pascal

```
property Attributes: TXactAttributes;
```

C++

```
__property TXactAttributes Attributes;
```

Description

Set Attributes to specify whether a connection object performs retaining commits or retaining aborts. The two behaviors are independent of each other.

Attributes can contain one of the two TXactAttribute (see page 1308) values (xaCommitRetaining and xaAbortRetaining), both values, or neither value.

```
ADOConnection1.Attributes := [xaCommitRetaining, xaAbortRetaining];
ADOConnection1->Attributes = TXactAttributes() << xaCommitRetaining << xaAbortRetaining;
```

See Also

Specifying Whether the Connection Automatically Initiates Transactions

8.1.4.5.2 TADOConnection.CommandCount Property

Indicates the number of command components associated with the connection.

Pascal

```
property CommandCount: Integer;
```

C++

```
__property int CommandCount;
```

Description

Read CommandCount to determine the number of command components associated with the connection component. A command component is associated with a connection component if its Connection property is set to the connection component.

CommandCount can be used as the basis for a loop to access each command component using the Commands (see page 1021) property. For example, to execute all of the TADOCommand (see page 984) components connected through the TADOConnection (see page 1001) component ADOConnection1:

```
var
  i: Integer;
begin
for i := 0 to (ADOConnection1.CommandCount – 1) do
    ADOConnection1.Commands[i].Execute;
end;
for (int i = 0; i < ADOConnection1->CommandCount; i++)
  ADOConnection1->Commands[i]->Execute();
```

See Also

DataSetCount

DataSets (see page 1027)

Commands (see page 1021)

Accessing the Connection's Datasets

8.1.4.5.3 TADOConnection.Commands Property

Lists all active commands for the connection component.

Pascal

```
property Commands [Index: Integer]: TADOCommand;
```

C++

```
__property TADOCommand Commands[int Index];
```

Description

Use Commands to access active command components associated with a connection component. An active command component is one that is currently open.

Commands connected through an ADO connection object can be accessed in sequence through the Commands property in a loop based on CommandCount (see page 1021). For example, to execute all of the TADOCommand (see page 984) components connected through the TADOConnection (see page 1001) component ADOConnection1:

```
var
  i: Integer;
begin
for i := 0 to (ADOConnection1.CommandCount – 1) do
    ADOConnection1.Commands[i].Execute;
end;
for (int i = 0; i < ADOConnection1->CommandCount; i++)
  ADOConnection1->Commands[i]->Execute();
```

See Also

DataSetCount

DataSets (see page 1027)

CommandCount (see page 1021)

Accessing the Connections Datasets

8.1.4.5.4 TADOConnection.CommandTimeout Property

Specifies amount of time to attempt execution of a command.

Pascal

```
property CommandTimeout: Integer;
```

C++

```
__property int CommandTimeout;
```

Description

Use CommandTimeout to specify the amount of time, in seconds, that that can expire before an attempt to execute a command is considered unsuccessful. The default value is 30 seconds.

If a command successfully executed prior to the expiration of the seconds specified in CommandTimeout, this property has no effect. If execution of a command has not been successfully completed before the time has elapsed, the command is canceled.

See Also

ConnectionTimeout (see page 1025)

Controlling Timeouts

8.1.4.5.5 TADOConnection.Connected Property

Specifies whether the connection is active.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Set Connected to trueto establish a connection to an ADO data store without opening a dataset. Set Connected to false to close a connection. The default value for Connected is false.

An application can check Connected to determine the current status of a connection. If Connected is true, the connection is active; if false, and the KeepConnection (see page 1029) property is also false, then the connection is inactive.

Connected can also be used in a program to determine the success of a call to the Open (see page 1016) method (a true value in Connected) or the Close method (a false value).

```
with ADOConnection1 do begin
  Open;
if Connected then
    { connection successful }
else
    { connection unsuccessful };
end;
ADOConnection1->Open();
if (ADOConnection1->Connected)
  // connection successful
else
  // connection unsuccessful
```

See Also

Open (see page 1016)

Close (see page 6863)

ConnectionTimeout (see page 1025)

ConnectionString (see page 1024)

ConnectionObject (see page 1023)

ConnectOptions (see page 1026)

OnWillConnect (see page 1039)

OnConnectComplete (see page 1036)

8.1.4.5.6 TADOConnection.ConnectionObject Property

Provides direct access to the ADO connection object.

Pascal

```
property ConnectionObject: _Connection;
```

C++

```
__property _Connection ConnectionObject;
```

Description

Use ConnectionObject to get direct access to the ADO connection object the connection component represents. This direct access reference allows an application to use properties and methods of the underlying connection object. Accessing the underlying connection object is especially useful for utilizing properties and methods of the connection object not surfaced in

TADOConnection (see page 1001).

Note: Use of ConnectionObject to directly access the underlying ADO connection object requires a good working knowledge of ADO objects in general and the ADO connection object in specific. Using the connection object directly is not recommended unless you are familiar with connection object operations. Consult the Microsoft Data Store SDK help for specific information on using ADO connection objects.

See Also

Open (see page 1016)

Close (see page 6863)

ConnectionTimeout (see page 1025)

ConnectionString (see page 1024)

Connected (see page 1022)

ConnectOptions (see page 1026)

DefaultDatabase (see page 1027)

Accessing the Connection Object

8.1.4.5.7 TADOConnection.ConnectionString Property

Specifies the connection information for the data store.

Pascal

```
property ConnectionString: WideString;
```

C++

```
__property BSTR ConnectionString;
```

Description

Set ConnectionString to specify the information needed to connect the ADO connection component to the data store. The value used for ConnectionString consists of one or more arguments ADO uses to establish the connection. Specify multiple arguments as a list with individual arguments separated by semicolons.

```
ADOConnection1.ConnectionString := 'Provider=ProviderRef;Remote Server=ServerRef';
ADOConnection1->ConnectionString = "Provider=ProviderRef;Remote Server=ServerRef";
```

At design-time, select from the list of available ADO database connections invoked from the Object Inspector.

A connection string may be saved to file for later use. Specify only the name of such a file in ConnectionString to reuse a saved connection string.

The connection string may also contain login information such as user ID and password, for automated logins.

When explicitly passing login information through either the Open (see page 1016) method or the ConnectionString, the LoginPrompt (see page 1029) property should be set to false to prevent an unnecessary login dialog.

ADO supports the following four arguments for connection strings. Any other arguments (such as a user ID and password) are not processed by ADO and simply passed on to the provider.

Argument	Meaning
Provider (see page 1032)	The name of the provider to use for the connection.
File name	The name of a file containing connection information.

Remote Provider (see page 1032)	The name of the provider to use for a client-side connection.
Remote Server	The path name of the server to use for a client-side connection.

Note: After opening a connection, the contents of ConnectionString may be changed by ADO. One example of this behavior is when ADO-defined arguments are mapped to their provider equivalents.

See Also

Open (see page 1016)

Close (see page 6863)

ConnectionTimeout (see page 1025)

Connected (see page 1022)

ConnectionObject (see page 1023)

ConnectOptions (see page 1026)

DefaultDatabase (see page 1027)

Connecting to a Data Store Using TADOConnection

8.1.4.5.8 TADOConnection.ConnectionTimeout Property

Specifies amount of time to attempt a connection.

Pascal

```
property ConnectionTimeout: Integer;
```

C++

```
__property int ConnectionTimeout;
```

Description

Use ConnectionTimeout to specify the amount of time, in seconds, that can expire before an attempt to make a connection is considered unsuccessful. The default value is 15 seconds.

If a connection is successfully made prior to the expiration of the seconds specified or the Cancel (see page 1009) method is called, ConnectionTimeout has no effect. If the specified time expires and a connection has not been successfully made, the attempt is terminated and an exception is raised.

See Also

Open (see page 1016)

Close (see page 6863)

Connected (see page 1022)

ConnectionString (see page 1024)

ConnectionObject (see page 1023)

ConnectOptions (see page 1026)

Cancel (see page 1009)

Controlling Timeouts

8.1.4.5.9 TADOConnection.ConnectOptions Property

Specifies whether a connection is synchronous or asynchronous.

Pascal

```
property ConnectOptions: TConnectOption;
```

C++

```
__property TConnectOption ConnectOptions;
```

Description

Set ConnectOptions to specify whether the connection established by the TADOConnection (see page 1001) is synchronous or asynchronous. The default value for ConnectOptions is coConnectUnspecified.

An application will typically use the default of a synchronous connection (coConnectUnspecified). Typically an asynchronous is only really needed to compensate for a slow server.

See Also

Open (see page 1016)

Close (see page 6863)

Connected (see page 1022)

ConnectionString (see page 1024)

ConnectionObject (see page 1023)

Cancel (see page 1009)

ConnectionTimeout (see page 1025)

Forcing Asynchronous Connections

8.1.4.5.10 TADOConnection.CursorLocation Property

Specifies whether cursors for the connection is client-side or server-side.

Pascal

```
property CursorLocation: TCursorLocation;
```

C++

```
__property TCursorLocation CursorLocation;
```

Description

Use CursorLocation to indicate whether the cursors that use the connection object to connect to the ADO datastore use a client-side or server-side cursor library. CursorLocation only affects connections opened after the property is set. The default value for CursorLocation is clUseClient.

A client-side cursor offers more flexibility. All data is retrieved to the local machine and then operated on there, allowing operations not normally supported by servers like sorting and resorting the data and additional filtering. SQL statements are executed at the server, so for statements that restrict the result set with a WHERE clause, only the reduced result set is retrieved to a local cursor.

A server-side cursor offers less flexibility, but may be more advantageous (or necessary) for large result sets. Using a server-side cursor becomes necessary when the sheer size of a result set exceeds the available disk space that would be needed to create the client-side cursor. Also, many servers only support unidirectional cursors. This would preclude moving the

record pointer in the dataset backward (even one record) through the result set.

8.1.4.5.11 TADOConnection.DataSets Property

Provides an indexed array of all active datasets for a connection component.

Pascal

```
property DataSets [Index: Integer]: TCustomADODataSet;
```

C++

```
__property TCustomADODataSet DataSets[int Index];
```

Description

Use DataSets to access active datasets associated with a connection component. An active dataset is one that is currently open.

Datasets connected through an ADO connection object can be accessed in sequence through the DataSets property in a loop based on DataSetCount.

```
var
  i: Integer;
begin
for i := 0 to (ADOConnection1.DataSetCount – 1) do
    ListBox1.Items.Add(ADOConnection1.DataSets[i].Name)
end;
for (int i = 0; i < ADOConnection1->DataSetCount; i++)
  ListBox1->Items->Add(ADOConnection1->DataSets[i]->Name);
```

See Also

DataSetCount

CommandCount (see page 1021)

Commands (see page 1021)

8.1.4.5.12 TADOConnection.DefaultDatabase Property

Indicates the database the ADO connection uses by default.

Pascal

```
property DefaultDatabase: WideString;
```

C++

```
__property BSTR DefaultDatabase;
```

Description

Set DefaultDatabase to indicate the database the ADO connection objects uses if the connection to the database specified in ConnectionString (see page 1024) is unavailable, the connection cannot be made, or the database is not specified in the connection string.

Note: If a default database is specified in the ConnectionString (see page 1024) property, it has precedence over the value in DefaultDatabase. The DefaultDatabase value is automatically changed to the database specified in ConnectionString (see page 1024) when the connection component is opened.

See Also

ConnectionString (see page 1024)

Open (see page 1016)

ConnectionObject (see page 1023)

Connecting to a Data Store Using TADOConnection

8.1.4.5.13 TADOConnection.Errors Property

Errors is the implementation of the Errors Collection.

Pascal

```
property Errors: Errors;
```

C++

```
__property Errors Errors;
```

Description

Use Errors to get direct access to the ADO Errors Collection for the ADO Connection object the TADOConnection (see page 1001) component represents. This direct access reference allows an application to use properties and methods of the underlying Errors Collection object. Accessing the underlying connection object is especially useful for utilizing properties and methods of the Errors Collection not surfaced in TADOConnection (see page 1001).

The Errors Collection object of a Connection object contains the errors generated by the provider. For each provider error, one or more ADO Error (see page 1312) objects are placed in the Errors Collection. Error (see page 1312) objects represent individual errors from the provider and are not ADO-specific. On a subsequent error, the Errors Collection is cleared of any Error (see page 1312) objects and new Error (see page 1312) objects are inserted for the current error.

Note: Use of Errors to directly access the underlying ADO Errors Collection object requires a good working knowledge of ADO objects in general and ADO Errors Collection objects in specific. Using the Errors Collection object directly is not recommended unless you are familiar with connection object operations. Consult the Microsoft Data Store SDK help for specific information on using ADO Errors Collection objects.

8.1.4.5.14 TADOConnection.InTransaction Property

Indicates whether a transaction is in progress.

Pascal

```
property InTransaction: Boolean;
```

C++

```
__property Boolean InTransaction;
```

Description

Read InTransaction at runtime to determine whether a transaction is currently active for the ADO connection component. If the connection component is currently in a transaction, InTransaction contains a true value. If no transaction is currently active it contains false.

A transaction is will be active if the BeginTrans (see page 1008) method has been called to initiate a transaction but that transaction has not either been consummated by calling CommitTrans (see page 1010) or canceled by calling the RollbackTrans (see page 1018) method. Use the InTransaction property to prevent calling BeginTrans (see page 1008) when a transaction has already been initiated for the connection component.

See Also

BeginTrans (see page 1008)

CommitTrans (see page 1010)

RollbackTrans (see page 1018)

8.1.4.5.15 TADOConnection.IsolationLevel Property

Specifies the transaction isolation level for transactions.

Pascal

```
property IsolationLevel: TIsolationLevel;
```

C++

```
__property TIsolationLevel IsolationLevel;
```

Description

Use IsolationLevel to specify the transaction isolation level for a connection. The transaction isolation level determines how a transaction interacts with other simultaneous transactions when they work with the same tables, and how much a transaction sees of the work performed by other transactions. The default value for IsolationLevel is ilCursorStability.

Set IsolationLevel to a valid TIsolationLevel (see page 1300) value prior to starting a new transaction. Read IsolationLevel after the transaction has been activated to determine the actual transaction isolation level used. It is possible that a server will force an isolation level other than that requested if the level requested is not supported.

Below are four of the supported transaction levels and the transaction phenomena that may occur under each. For more information on all supported transaction levels, see the topic for the TIsolationLevel (see page 1300) type.

Isolation Level	Dirty Reads	Nonrepeatable Reads	Phantoms

Dirty reads occur when a transaction reads data that has not yet been committed. Nonrepeatable reads occur when a transaction reads the same row twice and the data is different each time. Phantoms are rows that match the criteria used to compose a result set, but that are not initially visible.

8.1.4.5.16 TADOConnection.KeepConnection Property

Specifies whether an application remains connected to a database even if no datasets are open.

Pascal

```
property KeepConnection: Boolean;
```

C++

```
__property Boolean KeepConnection;
```

Description

Use KeepConnection to specify whether an application remains connected to a database even if no associated dataset components are currently active. When KeepConnection is true (the default) the connection is maintained. For connections to remote database servers, or for applications that frequently open and close datasets, set KeepConnection to true to reduce network traffic, speed up applications, and avoid logging in to the server each time the connection is reestablished.

When KeepConnection is false a connection is dropped when there are no open datasets. Dropping a connection releases system resources allocated to the connection, but if a dataset is later opened that uses the database, the connection must be reestablished and initialized.

8.1.4.5.17 TADOConnection.LoginPrompt Property

Specifies whether a login dialog appears immediately before opening a new connection.

Pascal

```
property LoginPrompt: Boolean;
```

C++

```
__property Boolean LoginPrompt;
```

Description

Set LoginPrompt to true to provide login support when establishing a connection. LoginPrompt controls two things: the occurrence of the OnLogin event, and the appearance of a default login dialog that prompts users for a name and password when you include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi). When the OnLogin event occurs and when the default login dialog appears depend on the type of connection component:

For TDatabase, TIBDatabase, and TSQLConnection, the dialog appears after the BeforeConnect event and before the AfterConnect event, unless you supply an OnLogin event handler. If there is an OnLogin event handler, that event occurs in place of the login dialog, and there is no need to include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi). If correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler, the connection fails. The OnLogin event does not fire unless LoginPrompt is set to true.

For TADOConnection components, the dialog appears after the OnWillConnect event and before the BeforeConnect event. If there is an OnLogin event handler, that event occurs after the login dialog. If you do not include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi), the OnLogin event occurs but there is no default login dialog. If correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler, the connection fails. The OnLogin event does not fire unless LoginPrompt is set to true.

For DataSnap SOAP connection components, the dialog appears after the OnGetUsername event and before the BeforeConnect, AfterConnect, and OnLogin events. If the user cancels from the login dialog, no attempt is made to open a connection.

When LoginPrompt is false, the application must supply user name and password values programmatically:

For TDatabase, the user name and password can be supplied as USER NAME and PASSWORD parameters in the Params property.

For TADOConnection, the user name and password can be supplied as the ConnectionString property.

For TSQLConnection, the user name, password, and database can be supplied as UserName, Password, and Database parameters in the Params property, or provided as connection parameters associated with the connection name.

For DataSnap SOAP connection components, there is no built-in use for the user name and password supplied by the login dialog. (The UserName and Password properties of TWebConnection are unrelated).

Warning: Storing hard-coded user name and password entries as property values or in code for an OnLogin event handler can compromise server security.

See Also

OnLogin (see page 6873)

BeforeConnect (see page 6875)

AfterConnect (see page 6874)

OnGetUserName (see page 8533)

OnLogin (see page 20259)

OnLogin (see page 9994)

Controlling Server Login

8.1.4.5.18 TADOConnection.Mode Property

Indicate the persmissions available to a connection.

Pascal

```
property Mode: TConnectMode;
```

C++

```
__property TConnectMode Mode;
```

Description

Read Mode to determine the permissions available to the connection after the connection has been activated. The permissions expressed in Mode govern the types of operations (such as reading and writing) can be performed through the connection. They also control how these operations are affected by and affect other concurrent users.

See Also

Indicating the Types of Operations the Connection Supports

8.1.4.5.19 TADOConnection.OnLogin Property

Occurs when a communication channel opens to the server.

Pascal

```
property OnLogin: TLoginEvent;
```

C++

```
__property TLoginEvent OnLogin;
```

Description

OnLogin occurs when connections are opened if the LoginPrompt property is true.

For DataSnap SOAP connection components, the following sequence of events occurs when opening a connection to an application server:

1. An OnGetUsername event occurs, where the application can supply an initial user name for the login dialog.
2. The login dialog appears, where the user can supply a user name and password.
3. A BeforeConnect event occurs, the connection is opened, and an AfterConnect event occurs.
4. The OnLogin event occurs, where you can write code to log in to the application server using the username and password that were provided in the remote login dialog.

   Note: Do not use the OnLogin event if the connection component provides a different scheme for logging in. For example, TWebConnection uses the Username and Password properties instead.

   For ADO connection components, the following sequence of events occurs when opening a connection to an application server:

1. A BeforeConnect event occurs.

2. An OnWillConnect event occurs, where the application can supply an initial user name and password for the login dialog, as well as the connection options and status.

3. The login dialog appears, where the user can supply a user name and password.

4. The OnLogin event occurs, where you can write code to respond to the user-supplied values. (For example, you could convert an application-specific user name and password to database user name and password).

5. The connection is opened, and an AfterConnect event occurs.

Note: Some

C connection components that do not connect to an application server in a multitiered database application replace the OnLogin event defined in TCustomConnection with a different event that occurs before a connection is established.

See Also

Connected (see page 6868)

Open (see page 6866)

OnGetUserName (see page 8533)

BeforeConnect (see page 6875)

AfterConnect (see page 6874)

LoginPrompt (see page 6872)

OnWillConnect (see page 1039)

8.1.4.5.20 TADOConnection.Properties Property

Pascal

```
property Properties: Properties;
```

C++

```
__property Properties Properties;
```

Description

This is Properties, a member of class TADOConnection.

8.1.4.5.21 TADOConnection.Provider Property

Specifies the provider for the ADO connection.

Pascal

```
property Provider: WideString;
```

C++

```
__property BSTR Provider;
```

Description

Set Provider to indicate the provider the ADO connection object uses.

The provider a connection is to use can also be specified in the string entered in ConnectionString (see page 1024). If Provider is left blank and the provider specified through ConnectionString (see page 1024), when the connection is activated the Provider property is automatically set indicating the provider used.

See Also

Connecting to a Data Store Using TADOConnection

8.1.4.5.22 TADOConnection.State Property

Indicates the current state of the ADO connection.

Pascal

```
property State: TObjectStates;
```

C++

```
__property TObjectStates State;
```

Description

Read State at runtime to determine the activity being performed, if any, by the connection at a specific point in time. Activity status includes the command component connecting to a data store, executing a connection, fetching rows for a recordset, sitting idle but open, and sitting idle and closed.

See Also

Open (see page 1016)

Connected (see page 1022)

Close (see page 6863)

8.1.4.5.23 TADOConnection.Version Property

Indicates the version of ADO used.

Pascal

```
property Version: WideString;
```

C++

```
__property BSTR Version;
```

Description

Read Version at runtime to determine the version of ADO used. Some versions of ADO do not support all possible features, so querying Version is important when using features only supported in later versions.

8.1.4.6 TADOConnection Events

The events of the TADOConnection class are listed here.

Published Events

	Name	Description
	AfterConnect (see page 1034)	Occurs after a connection is established.
	AfterDisconnect (see page 1034)	Occurs after the connection closes.
	BeforeConnect (see page 1034)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 1035)	Occurs immediately before the connection closes.
	OnBeginTransComplete (see page 1035)	Occurs after beginning a transaction.
	OnCommitTransComplete (see page 1036)	Occurs after committing a transaction.
	OnConnectComplete (see page 1036)	Occurs after starting a connection.
	OnDisconnect (see page 1037)	Occurs after a connection is terminated.
	OnExecuteComplete (see page 1038)	Occurs after a command is executed.
	OnInfoMessage (see page 1038)	Occurs when information messages are received from the data store connection.
	OnRollbackTransComplete (see page 1039)	Occurs after rolling back a transaction.
	OnWillConnect (see page 1039)	Occurs after a request to start a connection.
	OnWillExecute (see page 1040)	Occurs after a database server signals acceptance of a command execution.

8.1.4.6.1 TADOConnection.AfterConnect Event

Occurs after a connection is established.

Pascal

```
property AfterConnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterConnect;
```

Description

Write an AfterConnect event handler to take application-specific actions immediately after the connection component opens a connection to the remote source of database information.

See Also

BeforeConnect (see page 6875)

Connected (see page 6868)

Connecting to a Database Server

8.1.4.6.2 TADOConnection.AfterDisconnect Event

Occurs after the connection closes.

Pascal

```
property AfterDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterDisconnect;
```

Description

Write an AfterDisconnect event handler to take application-specific actions after the connection component drops a connection.

See Also

BeforeDisconnect (see page 6875)

Connected (see page 6868)

Disconnecting from a Database Server

8.1.4.6.3 TADOConnection.BeforeConnect Event

Occurs immediately before establishing a connection.

Pascal

```
property BeforeConnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeConnect;
```

Description

Write a BeforeConnect event handler to take application-specific actions before the connection component opens a connection to the remote source of database information.

See Also

AfterConnect (see page 6874)

Connected (see page 6868)

Connecting to a Database Server

8.1.4.6.4 TADOConnection.BeforeDisconnect Event

Occurs immediately before the connection closes.

Pascal

```
property BeforeDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeDisconnect;
```

Description

Write a BeforeDisconnect event handler to take application-specific actions before dropping a connection.

Delphi Examples:

```
{
This BeforeDisconnect event handler makes sure that the
client dataset has applied all its pending updates before
terminating the connection to the server.
}
procedure TForm1.RemoteServer1BeforeDisconnect(Sender: TObject);
begin
  if (ClientDataSet1.ChangeCount > 0) then
    ClientDataSet1.ApplyUpdates(-1);
end;
```

See Also

AfterDisconnect (see page 6875)

Connected (see page 6868)

Disconnecting from a Database Server

8.1.4.6.5 TADOConnection.OnBeginTransComplete Event

Occurs after beginning a transaction.

Pascal

```
property OnBeginTransComplete: TBeginTransCompleteEvent;
```

C++

```
__property TBeginTransCompleteEvent OnBeginTransComplete;
```

Description

Write an OnBeginTransComplete event handler to take specific action after an attempt to begin a transaction has successfully completed. The OnBeginTransComplete event fires after the corresponding operation (initiation of a transaction) has executed at the ADO Connection object. Calling the BeginTrans (see page 1008) method will lead to this event occurring.

TransactionLevel is the new transaction level from the call to the BeginTrans (see page 1008) method.

Error (see page 1312) is an ADO Error (see page 1312) object. See the Microsoft Data Access SDK for more information on

using ADO Error (see page 1312) objects. The Error (see page 1312) parameter is only set if an error occurred during execution of the operation that triggered the event. In this case, the EventStatus parameter will contain the indicating value esErrorsOccured.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event.

See Also

OnCommitTransComplete (see page 1036)

OnRollbackTransComplete (see page 1039)

BeginTrans (see page 1008)

BeginTrans (see page 1008)

ADO Connection Events

8.1.4.6.6 TADOConnection.OnCommitTransComplete Event

Occurs after committing a transaction.

Pascal

```
property OnCommitTransComplete: TConnectErrorEvent;
```

C++

```
__property TConnectErrorEvent OnCommitTransComplete;
```

Description

Write an OnCommitTransComplete event handler to take specific action after an attempt to commit a transaction has successfully completed. The OnCommitTransComplete event fires after the corresponding operation (committing a transaction) has executed at the ADO Connection object. Calling the CommitTrans (see page 1010) method will lead to this event occurring.

Error (see page 1312) is an ADO Error (see page 1312) object. See the Microsoft Data Access SDK for more information on using ADO Error (see page 1312) objects. The Error (see page 1312) parameter is only set if an error occurred during execution of the operation that triggered the event. In this case, the EventStatus parameter will contain the indicating value esErrorsOccured.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event.

See Also

OnBeginTransComplete (see page 1035)

OnRollbackTransComplete (see page 1039)

CommitTrans (see page 1010)

CommitTrans (see page 1010)

ADO Connection Events

8.1.4.6.7 TADOConnection.OnConnectComplete Event

Occurs after starting a connection.

Pascal

```
property OnConnectComplete: TConnectErrorEvent;
```

C++

```
__property TConnectErrorEvent OnConnectComplete;
```

Description

Write an OnConnectComplete event handler to take specific action after an attempt to connect to a database has successfully completed and the connection has started. The OnConnectComplete event fires after the corresponding operation (establishing a connection) has executed at the ADO Connection object. Calling the Open (see page 1016) method or setting the Connected (see page 1022) property to true will lead to this event occurring.

Error (see page 1312) is an ADO Error (see page 1312) object. See the Microsoft Data Access SDK for more information on using ADO Error (see page 1312) objects. The Error (see page 1312) parameter is only set if an error occurred during execution of the operation that triggered the event. In this case, the EventStatus parameter will contain the indicating value esErrorsOccured.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event.

See Also

Open (see page 1016)

Connected (see page 1022)

ADO Connection Events

8.1.4.6.8 TADOConnection.OnDisconnect Event

Occurs after a connection is terminated.

Pascal

```
property OnDisconnect: TDisconnectEvent;
```

C++

```
__property TDisconnectEvent OnDisconnect;
```

Description

Write an OnDisconnect event handler to take specific action after a connection to a database is terminated. The OnDisconnect event fires after the corresponding operation (ending a connection) has executed at the ADO Connection object. Calling the Close method or setting the Connected (see page 1022) property to false will lead to this event occurring.

Error (see page 1312) is an ADO Error (see page 1312) object. See the Microsoft Data Access SDK for more information on using ADO Error (see page 1312) objects. The Error (see page 1312) parameter is only set if an error occurred during execution of the operation that triggered the event. In this case, the EventStatus parameter will contain the indicating value esErrorsOccured.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event.

See Also

Close (see page 6863)

Connected (see page 1022)

ADO Connection Events

8.1.4.6.9 TADOConnection.OnExecuteComplete Event

Occurs after a command is executed.

Pascal

```
property OnExecuteComplete: TExecuteCompleteEvent;
```

C++

```
__property TExecuteCompleteEvent OnExecuteComplete;
```

Description

Write an OnExecuteComplete event handler to take specific action after the ADO connection object has successfully executed a command. The OnExecuteComplete event fires after the corresponding operation (executing an ADO command) has completed execution at the ADO Connection object. Calling the Execute (see page 1012) method will lead to this event occurring.

Connection indicates the TADOConnection (see page 1001) component executing the ADO command.

RecordsAffected indicated the number of rows in the table that were affected by the execution of the command. For instance, if the command deletes rows in a table, RowsAfftected indicates how many rows were deleted (possibly less than all rows if a WHERE clause is used in the command).

Error (see page 1312) is an ADO Error (see page 1312) object. See the Microsoft Data Access SDK for more information on using ADO Error (see page 1312) objects. The Error (see page 1312) parameter is only set if an error occurred during execution of the operation that triggered the event. In this case, the EventStatus parameter will contain the indicating value esErrorsOccured.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event.

Command is the ADO Command object executing the command. If no Command object was used, this parameter contains a nil (Delphi) or NULL (C++) value.

Recordset is an ADO Recordset object containing the result set for the command. Recordset only contains an ADO Recordset object if the command was a SELECT statement, otherwise it contains a nil (Delphi) or NULL (C++) value.

See Also

Execute (see page 1012)

ADO Connection Events

8.1.4.6.10 TADOConnection.OnInfoMessage Event

Occurs when information messages are received from the data store connection.

Pascal

```
property OnInfoMessage: TInfoMessageEvent;
```

C++

```
__property TInfoMessageEvent OnInfoMessage;
```

Description

Write an OnInfoMessage event handler to take specific action after an information message is received from a database connection. This event occurs when a connection event successfully completes and additional information is returned by the provider.

Error (see page 1312) is an ADO Error (see page 1312) object. See the Microsoft Data Access SDK for more information on

using ADO Error (see page 1312) objects. The Error (see page 1312) parameter is only set if an error occurred during execution of the operation that triggered the event. In this case, the EventStatus parameter will contain the indicating value esErrorsOccured.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event.

See Also

ADO Connection Events

8.1.4.6.11 TADOConnection.OnRollbackTransComplete Event

Occurs after rolling back a transaction.

Pascal

```
property OnRollbackTransComplete: TConnectErrorEvent;
```

C++

```
__property TConnectErrorEvent OnRollbackTransComplete;
```

Description

Write an OnRollbackTransComplete event handler to take specific action after an attempt to roll back a transaction has successfully completed. The OnRollbackTransComplete event fires after the corresponding operation (rolling back a transaction) has executed at the ADO Connection object. Calling the RollbackTrans (see page 1018) method will lead to this event occurring.

Error (see page 1312) is an ADO Error (see page 1312) object. See the Microsoft Data Access SDK for more information on using ADO Error (see page 1312) objects. The Error (see page 1312) parameter is only set if an error occurred during execution of the operation that triggered the event. In this case, the EventStatus parameter will contain the indicating value esErrorsOccured.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event.

See Also

OnCommitTransComplete (see page 1036)

OnBeginTransComplete (see page 1035)

RollbackTrans (see page 1018)

RollbackTrans (see page 1018)

ADO Connection EventsADO Connection EventsADO Connection EventsADO Connection EventsADO Connection EventsADO Connection Events

8.1.4.6.12 TADOConnection.OnWillConnect Event

Occurs after a request to start a connection.

Pascal

```
property OnWillConnect: TWillConnectEvent;
```

C++

```
__property TWillConnectEvent OnWillConnect;
```

Description

Write an OnWillConnect event handler to take specific action before a connection starts and after the provider indicates the connection attempt can be made. The OnWillConnect event fires after the corresponding operation (requesting a connection attempt) has executed at the ADO Connection object. Calling the Open (see page 1016) method or setting the Connected (see page 1022) property to true will lead to this event occurring.

Connection indicates the TADOConnection (see page 1001) component attempting the connection.

ConnectionString (see page 1024) contains the connection information for the pending connection.

UserID contains the user identification for the pending connection.

Password contains the password for the pending connection.

ConnectionOptions contains the TConnectOption (see page 1293) value that describes how the provider is to evaluate the connection string for the pending connection.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event. In a handler for this event, set this parameter to esCancel to request cancellation of the pending connection operation.

See Also

Open (see page 1016)

Connected (see page 1022)

ADO Connection Events

8.1.4.6.13 TADOConnection.OnWillExecute Event

Occurs after a database server signals acceptance of a command execution.

Pascal

```
property OnWillExecute: TWillExecuteEvent;
```

C++

```
__property TWillExecuteEvent OnWillExecute;
```

Description

Write an OnWillExecute event handler to take specific action before a command is executed. The OnWillExecute event fires after the corresponding operation (requesting a command execution) has executed at the ADO Connection object. Calling the Execute (see page 1012) method will lead to this event occurring.

Error (see page 1312) is an ADO Error (see page 1312) object. See the Microsoft Data Access SDK for more information on using ADO Error (see page 1312) objects. The Error (see page 1312) parameter is only set if an error occurred during execution of the operation that triggered the event. In this case, the EventStatus parameter will contain the indicating value esErrorsOccured.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event. In a handler for this event, set this parameter to esCancel to request cancellation of the pending command execution.

See Also

Execute (see page 1012)

8.1.5 TADODataSet Class

TADODataSet represents a dataset retrieved from an ADO data store.

Class Hierarchy

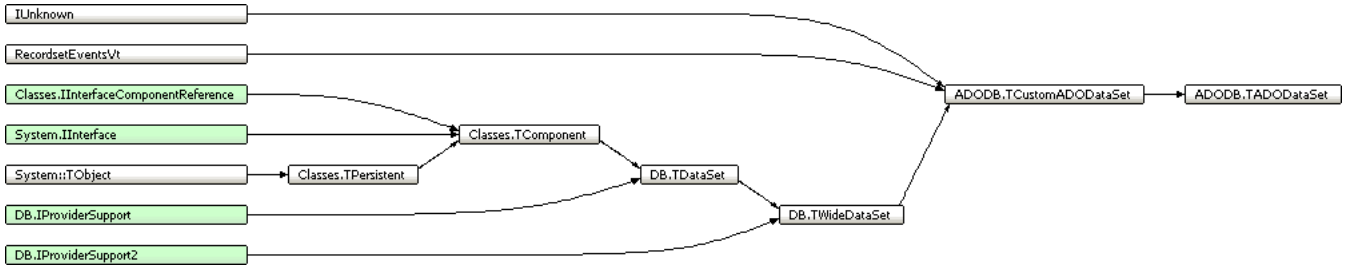

Pascal

```
TADODataSet = class(TCustomADODataSet);
```

C++

```
class TADODataSet : public TCustomADODataSet;
```

File

ADODB

Description

TADODataSet is the most generic of the ADO dataset components. TADODataSet is capable of retrieving a result set from one or more tables in an ADO data store. The retrieval can be either directly from a table or from one or more tables through an SQL statement.

Connect a TADODataSet to a data store before using it to retrieve a dataset. Do this with a connection string in its ConnectionString (see page 1206) property or with a connection already established by a TADOConnection (see page 1001) component specified in the Connection (see page 1205) property.

Use the TADODataSet component's CommandText (see page 1057) property to retrieve the dataset, specifying either a table name or an SQL statement (SELECT only). TADODataSet is not capable of issuing Data Manipulation Language (DML) SQL statements that do not return result sets (like DELETE, INSERT, and UPDATE). For this use a component like TADOCommand (see page 984) or TADOQuery (see page 1084).

See Also

TCustomADONETConnector

TADOConnection (see page 1001)

TADOTable (see page 1125)

TADOQuery (see page 1084)

TADOStoredProc (see page 1106)

Types of Datasets

Overview of ADO Components

Building a VCL Forms ADO.NET Database Application

8.1.5.1 TADODataSet Members

The following tables list the members exposed by TADODataSet.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	ActivateTextFilter (see page 1160)	This is ActivateTextFilter, a member of class TCustomADODataSet.
V	AllocRecordBuffer (see page 1160)	This is AllocRecordBuffer, a member of class TCustomADODataSet.
V	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CheckActive (see page 1162)	This is CheckActive, a member of class TCustomADODataSet.
V	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
V	CheckFieldCompatibility (see page 1162)	This is CheckFieldCompatibility, a member of class TCustomADODataSet.
V	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
V	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
V	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
V	ClearCalcFields (see page 1162)	This is ClearCalcFields, a member of class TCustomADODataSet.
V	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
V	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
V	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
	ConnectionPoint (see page 1163)	This is ConnectionPoint, a member of class TCustomADODataSet.
V	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
V	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
V	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
V	DataEvent (see page 1055)	This is DataEvent, a member of class TADODataSet.
	DeactivateFilters (see page 1164)	This is DeactivateFilters, a member of class TCustomADODataSet.
V	DefChanged (see page 1165)	This is DefChanged, a member of class TCustomADODataSet.
V	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
V	DefineProperties (see page 1165)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
V	DestroyLookupCursor (see page 1166)	This is DestroyLookupCursor, a member of class TCustomADODataSet.
V	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
V	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
V	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
V	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.

	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 1166)	This is DoOnNewRecord, a member of class TCustomADODataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	EnableEvents (see page 1166)	This is EnableEvents, a member of class TCustomADODataSet.
	EndOfRecordset (see page 1167)	This is EndOfRecordset, a member of class TCustomADODataSet.
	FetchComplete (see page 1167)	This is FetchComplete, a member of class TCustomADODataSet.
	FetchProgress (see page 1167)	This is FetchProgress, a member of class TCustomADODataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FieldChangeComplete (see page 1167)	This is FieldChangeComplete, a member of class TCustomADODataSet.
	FindRecord (see page 1168)	This is FindRecord, a member of class TCustomADODataSet.
	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeRecordBuffer (see page 1168)	This is FreeRecordBuffer, a member of class TCustomADODataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
	GetActiveRecBuf (see page 1168)	This is GetActiveRecBuf, a member of class TCustomADODataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
	GetBookmarkData (see page 1169)	This is GetBookmarkData, a member of class TCustomADODataSet.
	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
	GetBookmarkFlag (see page 1169)	This is GetBookmarkFlag, a member of class TCustomADODataSet.
	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 1169)	This is GetCanModify, a member of class TCustomADODataSet.
	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
	GetCommandText (see page 1170)	This is GetCommandText, a member of class TCustomADODataSet.
	GetDataSource (see page 1170)	This is GetDataSource, a member of class TCustomADODataSet.
	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.

V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
V	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
V	GetRecNo (see page 1171)	This is GetRecNo, a member of class TCustomADODataSet.
V	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
V	GetRecord (see page 1172)	This is GetRecord, a member of class TCustomADODataSet.
A	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
V	GetRecordCount (see page 1172)	This is GetRecordCount, a member of class TCustomADODataSet.
V	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
V	GetRecordSize (see page 1172)	This is GetRecordSize, a member of class TCustomADODataSet.
V	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
V	GetStateFieldValue (see page 1172)	This is GetStateFieldValue, a member of class TCustomADODataSet.
V	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
	InitOleDBAccess (see page 1173)	This is InitOleDBAccess, a member of class TCustomADODataSet.
V	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
V	InternalAddRecord (see page 1173)	This is InternalAddRecord, a member of class TCustomADODataSet.
V	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
V	InternalCancel (see page 1173)	This is InternalCancel, a member of class TCustomADODataSet.
V	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
V	InternalClose (see page 1173)	This is InternalClose, a member of class TCustomADODataSet.
A	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.
V	InternalDelete (see page 1173)	This is InternalDelete, a member of class TCustomADODataSet.
V	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
V	InternalEdit (see page 1174)	This is InternalEdit, a member of class TCustomADODataSet.
V	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
V	InternalFirst (see page 1174)	This is InternalFirst, a member of class TCustomADODataSet.
V	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
	InternalGetRecord (see page 1174)	This is InternalGetRecord, a member of class TCustomADODataSet.
V	InternalGotoBookmark (see page 1174)	This is InternalGotoBookmark, a member of class TCustomADODataSet.
V	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
V	InternalHandleException (see page 1175)	This is InternalHandleException, a member of class TCustomADODataSet.
A	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
V	InternalInitFieldDefs (see page 1175)	This is InternalInitFieldDefs, a member of class TCustomADODataSet.
A	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
V	InternalInitRecord (see page 1175)	This is InternalInitRecord, a member of class TCustomADODataSet.
V	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
V	InternalInsert (see page 1175)	This is InternalInsert, a member of class TCustomADODataSet.
V	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
V	InternalLast (see page 1175)	This is InternalLast, a member of class TCustomADODataSet.
V	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
V	InternalOpen (see page 1176)	This is InternalOpen, a member of class TCustomADODataSet.
A	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
V	InternalPost (see page 1176)	This is InternalPost, a member of class TCustomADODataSet.
V	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
V	InternalRefresh (see page 1176)	This is InternalRefresh, a member of class TCustomADODataSet.
V	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
	InternalRequery (see page 1176)	This is InternalRequery, a member of class TCustomADODataSet.
	InternalSetSort (see page 1177)	This is InternalSetSort, a member of class TCustomADODataSet.
V	InternalSetToRecord (see page 1177)	This is InternalSetToRecord, a member of class TCustomADODataSet.

	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsCursorOpen (see page 1177)	This is IsCursorOpen, a member of class TCustomADODataSet.
	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
	Loaded (see page 1177)	Initializes the component after the form file has been read into memory.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1179)	This is LocateRecord, a member of class TCustomADODataSet.
	MasterChanged (see page 1180)	This is MasterChanged, a member of class TCustomADODataSet.
	MasterDisabled (see page 1180)	This is MasterDisabled, a member of class TCustomADODataSet.
	MoveComplete (see page 1180)	This is MoveComplete, a member of class TCustomADODataSet.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	OpenCursor (see page 1056)	This is OpenCursor, a member of class TADODataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PrepareCursor (see page 1181)	This is PrepareCursor, a member of class TCustomADODataSet.
	PSEndTransaction (see page 1181)	Ends a transaction that was started by a call to PSStartTransaction.
	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 1182)	Executes the SQL command associated with the dataset.
	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 1182)	Executes a specified SQL command.
	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSExecuteStatement (see page 7535)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 1183)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 1183)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 6982)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 7537)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandTextW (see page 7537)	This is PSGetCommandTextW, a member of class TWideDataSet.
	PSGetCommandType (see page 1184)	Gets the command type to be executed by PSExecute.
	PSGetCommandType (see page 6982)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 1184)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 1185)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7538)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 1185)	This is PSGetKeyFieldsW, a member of class TCustomADODataSet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetKeyFieldsW (see page 7538)	This is PSGetKeyFieldsW, a member of class TWideDataSet.
	PSGetParams (see page 1185)	Returns the current parameter values of the dataset.
	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7538)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 1186)	This is PSGetQuoteCharW, a member of class TCustomADODataSet.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetQuoteCharW (see page 7539)	This is PSGetQuoteCharW, a member of class TWideDataSet.

	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7539)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 1186)	This is PSGetTableNameW, a member of class TCustomADODataSet.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetTableNameW (see page 7539)	This is PSGetTableNameW, a member of class TWideDataSet.
	PSGetUpdateException (see page 1186)	Generates an EUpdateError object based on another exception object.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 1187)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 1187)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 1188)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 1188)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 1188)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetCommandText (see page 7540)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 1189)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 1189)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 1190)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RecordChangeComplete (see page 1190)	This is RecordChangeComplete, a member of class TCustomADODataSet.
	RecordsetChangeComplete (see page 1190)	This is RecordsetChangeComplete, a member of class TCustomADODataSet.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	RefreshParams (see page 1191)	This is RefreshParams, a member of class TCustomADODataSet.
	ReleaseLock (see page 1191)	This is ReleaseLock, a member of class TCustomADODataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBlockReadSize (see page 1194)	This is SetBlockReadSize, a member of class TCustomADODataSet.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBookmarkData (see page 1194)	This is SetBookmarkData, a member of class TCustomADODataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkFlag (see page 1194)	This is SetBookmarkFlag, a member of class TCustomADODataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCommandText (see page 1194)	This is SetCommandText, a member of class TCustomADODataSet.
	SetConnection (see page 1056)	This is SetConnection, a member of class TADODataSet.
	SetConnectionFlag (see page 1195)	This is SetConnectionFlag, a member of class TCustomADODataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetDataSetField (see page 1056)	This is SetDataSetField, a member of class TADODataSet.
	SetDataSource (see page 1195)	This is SetDataSource, a member of class TCustomADODataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDetailFilter (see page 1195)	This is SetDetailFilter, a member of class TCustomADODataSet.
V	SetFieldData (see page 1195)	This is the overview for the SetFieldData method overload.
V	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
V	SetFiltered (see page 1196)	This is SetFiltered, a member of class TCustomADODataSet.
V	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
V	SetFilterOptions (see page 1196)	This is SetFilterOptions, a member of class TCustomADODataSet.
V	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
V	SetFilterText (see page 1196)	This is SetFilterText, a member of class TCustomADODataSet.
V	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
V	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetParamsFromCursor (see page 1197)	This is SetParamsFromCursor, a member of class TCustomADODataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetRecNo (see page 1197)	This is SetRecNo, a member of class TCustomADODataSet.
V	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
V	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateIndexDefs (see page 1198)	This is UpdateIndexDefs, a member of class TCustomADODataSet.
V	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.
	UpdateRecordSetPosition (see page 1198)	This is UpdateRecordSetPosition, a member of class TCustomADODataSet.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WillChangeField (see page 1199)	This is WillChangeField, a member of class TCustomADODataSet.
	WillChangeRecord (see page 1199)	This is WillChangeRecord, a member of class TCustomADODataSet.
	WillChangeRecordset (see page 1200)	This is WillChangeRecordset, a member of class TCustomADODataSet.
	WillMove (see page 1200)	This is WillMove, a member of class TCustomADODataSet.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.

R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
R	Command (see page 1204)	Provides access to the internal command object that executes commands for the dataset.
	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
R	MasterDataLink (see page 1218)	Manages the relationship between the ADO dataset and a master table.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
	Reserved (see page 7035)	For internal use only.

Public Events

	Name	Description
	AfterRefresh (see page 7047)	Occurs after an application refreshes the data in the dataset.
	BeforeRefresh (see page 7052)	Occurs immediately before an application refreshes the data in the dataset.
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 1160)	Tests the validity of a specified bookmark.
V	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
V	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
	CancelBatch (see page 1161)	Cancels a pending batch update.
	CancelUpdates (see page 1162)	Cancels pending cached updates.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Clone (see page 1163)	Duplicates another dataset in the calling component
	Close (see page 6920)	Closes a dataset.
V	CompareBookmarks (see page 1163)	Compares two bookmarks.
V	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
V	Create (see page 1156)	Creates an instance of a TCustomADONETConnector component.

	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 6904)	Creates an instance of a TDataSet (see page 6894) component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateBlobStream (see page 1164)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
	CreateDataSet (see page 1055)	Creates the recordset for an ADO dataset.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
	DeleteRecords (see page 1165)	Deletes one or more records in the dataset.
	Destroy (see page 1157)	Destroys the TCustomADONETConnector instance.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 6904)	Destroys the TDataSet (see page 6894) instance.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FilterOnBookmarks (see page 1168)	Filters a data set to just rows with pre-defined bookmarks.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetBlobFieldData (see page 1169)	Retrieves the current value of a field into a buffer.
	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
	GetDetailLinkFields (see page 1170)	Lists the field components that link this dataset as a detail of a master dataset.
	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldData (see page 1171)	This is the overview for the GetFieldData method overload.

	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetIndexNames (see page 1055)	Retrieves a list of available indexes for a table.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
	IsSequenced (see page 1177)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	Last (see page 6968)	Moves to the last record in the dataset.
	LoadFromFile (see page 1178)	Loads a recordset from a file.
	Locate (see page 1179)	Searches the dataset for a specified record and makes that record the current record.
	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Lookup (see page 1180)	Retrieves field values from a row that matches specified search values.
	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	NextRecordset (see page 1181)	Provides access to a secondary recordset.
	Open (see page 6976)	Opens the dataset.
	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSEndTransaction (see page 7589)	Ends a transaction that was started by a call to PSStartTransaction (see page 7597).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecute (see page 7590)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSExecuteStatement (see page 7590)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 7591)	Returns information that is stored in data packets as Name/Value pairs.

	PSGetCommandText (see page 7579)	Gets the SQL command to be executed by PSExecute (see page 7578).
	PSGetCommandText (see page 7591)	Gets the SQL command to be executed by PSExecute (see page 7590).
	PSGetCommandType (see page 7580)	Gets the command type to be executed by PSExecute (see page 7578).
	PSGetCommandType (see page 7591)	Gets the command type to be executed by PSExecute (see page 7590).
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 7592)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 7592)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7593)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetParams (see page 7593)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7594)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7594)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSGetUpdateException (see page 7594)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 7595)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 7595)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 7596)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 7596)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7585)	Assigns a specified SQL command to be executed by PSExecute (see page 7578).
	PSSetCommandText (see page 7596)	Assigns a specified SQL command to be executed by PSExecute (see page 7590).
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 7597)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).
	PSStartTransaction (see page 7597)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7589).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 7598)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Requery (see page 1191)	Refreshes the recordset.
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.

V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 1192)	Saves a recordset to a file.
	Seek (see page 1192)	Searches for a record using the current index.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	Supports (see page 1197)	Indicates whether a dataset supports certain types of operations.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateBatch (see page 1197)	Writes pending batch updates to disk.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 1199)	Indicates the update status of the current record.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.

Public Properties

	Name	Description
	Active (see page 7004)	Specifies whether or not a dataset is open.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignerData (see page 1208)	Stores custom data.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.
	FilterGroup (see page 1214)	Filters a recordset based on row update status.
	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
	IndexDefs (see page 1062)	Contains the definitions of the indexes for the current table.
R	IndexFieldCount (see page 1216)	Indicates the number of fields that comprise the current key.
	IndexFields (see page 1217)	Retrieves or sets a field for an index.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	Modified (see page 7029)	Indicates whether the active record is modified.
	ObjectView (see page 7031)	Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.

R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Properties (see page 1221)	This is Properties, a member of class TCustomADODataSet.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
	Recordset (see page 1221)	This is Recordset, a member of class TCustomADODataSet.
R	RecordsetState (see page 1222)	This is RecordsetState, a member of class TCustomADODataSet.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
R	RecordStatus (see page 1222)	Indicates the status of the current row.
	Sort (see page 1222)	Specifies the sort order of the recordset.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterCancel (see page 1067)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 1069)	Occurs after an application closes a dataset.
	AfterDelete (see page 1071)	Occurs after an application deletes a record.
	AfterEdit (see page 1073)	Occurs after an application starts editing a record.
	AfterInsert (see page 1074)	Occurs after an application inserts a new record.
	AfterOpen (see page 1075)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 1076)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 1233)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 1076)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 1077)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 1077)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 1078)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 1078)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 1079)	Occurs before an application enters insert mode.
	BeforeOpen (see page 1080)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 1081)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 1239)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 1082)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 1082)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 1083)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 1083)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnEndOfRecordset (see page 1241)	Occurs when moving a recordset's row pointer past the last row.
	OnFetchComplete (see page 1242)	Occurs after records have been retrieved into the recordset.
	OnFetchProgress (see page 1242)	Occurs periodically during an asynchronous data retrieval operation.
	OnFieldChangeComplete (see page 1243)	Occurs after a field has changed.
	OnFilterRecord (see page 1244)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnMoveComplete (see page 1244)	Occurs after the row pointer in a recordset has moved.
	OnNewRecord (see page 1084)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 1084)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnRecordChangeComplete (see page 1246)	Occurs after one or more records in the recordset change.
	OnRecordsetChangeComplete (see page 1247)	Occurs after the recordset has changed.
	OnRecordsetCreate (see page 1247)	Occurs when the recordset has been initialized and is ready to use.

	OnWillChangeField (see page 1248)	Occurs before a field is changed.
	OnWillChangeRecord (see page 1248)	Occurs before a record in the recordset is changed.
	OnWillChangeRecordset (see page 1249)	Occurs before the recordset has changed.
	OnWillMove (see page 1250)	Occurs before the row pointer in a recordset is moved.

Published Properties

	Name	Description
	Active (see page 1201)	Specifies whether or not a dataset is open.
	AutoCalcFields (see page 1203)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	CacheSize (see page 1203)	Specifies the size of the cache for the dataset.
	CommandText (see page 1057)	This is CommandText, a member of class TADODataSet.
	CommandTimeout (see page 1057)	Specifies amount of time to attempt execution of a command.
	CommandType (see page 1057)	Specifies the type of command to execute.
	Connection (see page 1205)	Specifies the ADO connection component to use.
	ConnectionString (see page 1206)	Specifies the connection information for the data store.
	CursorLocation (see page 1207)	Specifies whether the cursor for the connection is client-side or server-side.
	CursorType (see page 1208)	Specifies type of cursor an ADO dataset uses.
	DataSetField (see page 1058)	Indicates the persistent TDataSetField object that owns a nested dataset.
	DataSource (see page 1059)	Represents the data source of another dataset that supplies values to the dataset.
	EnableBCD (see page 1059)	Specifies whether to treat numeric fields as floating-point or BCD.
	ExecuteOptions (see page 1209)	Specifies the characteristics of a command execution.
	FieldDefs (see page 1060)	Points to the list of field definitions for the dataset.
	Filter (see page 1212)	Specifies the text of the current filter for a recordset.
	Filtered (see page 1214)	Specifies whether or not filtering is active for a dataset.
	IndexFieldNames (see page 1063)	Lists the columns to use for a persistent sort of the recordset.
	IndexName (see page 1063)	Specifies the currently active index.
	LockType (see page 1218)	Specifies the lock type to use when opening a dataset.
	MarshalOptions (see page 1218)	Specifies which records are marshaled back to the server.
	MasterFields (see page 1064)	Specifies field or fields on which a Master-Detail link is based.
	MaxRecords (see page 1219)	Specifies how many rows to return in a result set.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParamCheck (see page 1065)	Specifies whether the parameter list for an ADO dataset component is regenerated if the SQL changes at runtime.
	Parameters (see page 1065)	Contains the collection of parameters for an SQL statement.
	Prepared (see page 1066)	Specifies whether the command is prepared before execution.
	RDSConnection (see page 1066)	Specifies the RDS connection component to use.
	StoreDefs (see page 1066)	Specifies whether field definitions are stored with the ADO dataset.
	Tag (see page 2808)	Stores an integer value as part of a component.

8.1.5.2 TADODataSet Methods

The methods of the TADODataSet class are listed here.

Protected Methods

	Name	Description
	DataEvent (see page 1055)	This is DataEvent, a member of class TADODataSet.
	OpenCursor (see page 1056)	This is OpenCursor, a member of class TADODataSet.
	SetConnection (see page 1056)	This is SetConnection, a member of class TADODataSet.
	SetDataSetField (see page 1056)	This is SetDataSetField, a member of class TADODataSet.

Public Methods

	Name	Description
	CreateDataSet (see page 1055)	Creates the recordset for an ADO dataset.
	GetIndexNames (see page 1055)	Retrieves a list of available indexes for a table.

8.1.5.2.1 TADODataSet.CreateDataSet Method

Creates the recordset for an ADO dataset.

Pascal

```
procedure CreateDataSet;
```

C++

```
__fastcall CreateDataSet();
```

Description

Applications should not need to call CreateDataSet themselves. CreateDataSet is called automatically by an ADO dataset to create its recordset.

8.1.5.2.2 TADODataSet.DataEvent Method

Pascal

```
procedure DataEvent(Event: TDataEvent; Info: Longint); override;
```

C++

```
virtual __fastcall DataEvent(TDataEvent Event, Longint Info);
```

Description

This is DataEvent, a member of class TADODataSet.

8.1.5.2.3 TADODataSet.GetIndexNames Method

Retrieves a list of available indexes for a table.

Pascal

```
procedure GetIndexNames(List: TStrings);
```

C++

```
__fastcall GetIndexNames(TStrings List);
```

Description

Call GetIndexNames to retrieve a list of all available indexes for a table. GetIndexNames fills a string list object (such as a TStringList) with the names of the index for the table represented by the TADODataSet (see page 1041) component (and only that table). Any previous contents in the string list object are completely overwritten by the new information.

List is the string list object, created and maintained by the application, into which the index names are retrieved. This string list object must be created prior to the call to GetIndexNames.

The command line below fills the Items property of a TListBox with the names of indexes for the table represented by ADODataSet1.

```
ADODataSet1.GetIndexNames(ListBox1.Items);
ADODataSet1->GetIndexNames(ListBox1->Items);
```

See Also

GetTableNames (see page 1015)

GetProcedureNames (see page 1014)

Obtaining Information About Indexes

8.1.5.2.4 TADODataSet.OpenCursor Method

Pascal

```
procedure OpenCursor(InfoQuery: Boolean); override;
```

C++

```
virtual __fastcall OpenCursor(Boolean InfoQuery);
```

Description

This is OpenCursor, a member of class TADODataSet.

8.1.5.2.5 TADODataSet.SetConnection Method

Pascal

```
procedure SetConnection(const Value: TADOConnection); override;
```

C++

```
virtual __fastcall SetConnection(const TADOConnection Value);
```

Description

This is SetConnection, a member of class TADODataSet.

8.1.5.2.6 TADODataSet.SetDataSetField Method

Pascal

```
procedure SetDataSetField(const Value: TDataSetField); override;
```

C++

```
virtual __fastcall SetDataSetField(const TDataSetField Value);
```

Description

This is SetDataSetField, a member of class TADODataSet.

8.1.5.3 TADODataSet Properties

The properties of the TADODataSet class are listed here.

Public Properties

	Name	Description
	IndexDefs (see page 1062)	Contains the definitions of the indexes for the current table.

Published Properties

	Name	Description
	CommandText (see page 1057)	This is CommandText, a member of class TADODataSet.

	CommandTimeout (see page 1057)	Specifies amount of time to attempt execution of a command.
	CommandType (see page 1057)	Specifies the type of command to execute.
	DataSetField (see page 1058)	Indicates the persistent TDataSetField object that owns a nested dataset.
	DataSource (see page 1059)	Represents the data source of another dataset that supplies values to the dataset.
	EnableBCD (see page 1059)	Specifies whether to treat numeric fields as floating-point or BCD.
	FieldDefs (see page 1060)	Points to the list of field definitions for the dataset.
	IndexFieldNames (see page 1063)	Lists the columns to use for a persistent sort of the recordset.
	IndexName (see page 1063)	Specifies the currently active index.
	MasterFields (see page 1064)	Specifies field or fields on which a Master-Detail link is based.
	ParamCheck (see page 1065)	Specifies whether the parameter list for an ADO dataset component is regenerated if the SQL changes at runtime.
	Parameters (see page 1065)	Contains the collection of parameters for an SQL statement.
	Prepared (see page 1066)	Specifies whether the command is prepared before execution.
	RDSConnection (see page 1066)	Specifies the RDS connection component to use.
	StoreDefs (see page 1066)	Specifies whether field definitions are stored with the ADO dataset.

8.1.5.3.1 TADODataSet.CommandText Property

Pascal

```
property CommandText: WideString;
```

C++

```
__property BSTR CommandText;
```

Description

This is CommandText, a member of class TADODataSet.

8.1.5.3.2 TADODataSet.CommandTimeout Property

Specifies amount of time to attempt execution of a command.

Pascal

```
property CommandTimeout: Integer;
```

C++

```
__property int CommandTimeout;
```

Description

Use CommandTimeout to specify the amount of time that expires before an attempt to execute a command is considered unsuccessful. Measured in seconds, the default value for CommandTimeout is 30.

If a command is successfully executed prior to the expiration of the seconds specified, CommandTimeout has no effect.

See Also

commandtext (see page 1204)

CommandType (see page 1205)

8.1.5.3.3 TADODataSet.CommandType Property

Specifies the type of command to execute.

Pascal

```
property CommandType: TCommandType;
```

C++

```
__property TCommandType CommandType;
```

Description

Set CommandType to indicate the type of command that is contained in the CommandText property. The value in CommandType should agree with that of CommandText. For instance, if CommandText is the name of a table, CommandType should be cmdTable or cmdTableDirect.

The default value of CommandType is cmdUnknown. While CommandType may be left as cmdUnknown for all commands, better performance will be realized if CommandType is used to explicitly indicate the type of the command. When the type cmdUnknown is used, ADO must evaluate the command to determine its type, which slows the operation.

```
with ADODataSet1 do begin
  CommandType := cmdText;
  CommandText := 'SELECT * FROM CustomerTable';
  Open;
end;
ADODataSet1->CommandType = cmdText;
ADODataSet1->CommandText = "SELECT * FROM CustomerTable";
ADODataSet1->Open();
```

See Also

commandtext (see page 1204)

Open (see page 6976)

CommandTimeout (see page 1204)

Using TADODataSet

8.1.5.3.4 TADODataSet.DataSetField Property

Indicates the persistent TDataSetField object that owns a nested dataset.

Pascal

```
property DataSetField: TDataSetField;
```

C++

```
__property TDataSetField DataSetField;
```

Description

Use DataSetField when the dataset is the nested detail set of another dataset, such as an Oracle8 nested dataset. Read DataSetField to access the data set field component in the master dataset that includes this dataset as a detail. Set DataSetField to a field component in the master dataset to assign this dataset as the value of that field.

Note: When setting the DataSetField property, the dataset must be of (or a descendant of) the class specified by the master dataset's NestedDataSetClass property. Otherwise, setting DataSetField raises an exception. Further, setting DataSetField to a TDataSetField that is contained in the same dataset raises an exception (A dataset can't be nested in itself).

See Also

TNestedTable (see page 10040)

NestedDataSet (see page 7070)

NestedDataSetClass (see page 7030)

Using Nested Detail Tables

Working with DataSet Fields

Working with Reference Fields

8.1.5.3.5 TADODataSet.DataSource Property

Represents the data source of another dataset that supplies values to the dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;__property TDataSource DataSource;
```

Description

As implemented in TDataSet, the DataSource property is always nil (Delphi) or NULL (C++).

Descendant dataset classes that support specifying a data source redeclare and implement methods for getting and setting the DataSource property.

See Also

Active (see page 7004)

IsLinkedTo (see page 6967)

Establishing Master/detail Relationships Using Parameters

8.1.5.3.6 TADODataSet.EnableBCD Property

Specifies whether to treat numeric fields as floating-point or BCD.

Pascal

```
property EnableBCD: Boolean;
```

C++

```
__property Boolean EnableBCD;
```

Description

Use the EnableBCD property to specify how fields are mapped to field classes. If EnableBCD is true, (the default value) adDecimal and adNumeric fields are mapped to the TBCDField class when field objects are created. If EnableBCD is false, the fields are mapped to the TFloatField class.

EnableBCD determines whether numeric and decimal fields are translated as floating-point values or binary coded decimal (BCD) values. BCD values eliminate the rounding errors associated with floating point math (such as a 3 * (2/3) resulting in 2.00000000001).

If persistent field objects are used (such as those created using the Fields Editor), the field class associated with a given field can contradict the setting of EnableBCD. Three numeric fields in the same table could, for instance, each be represented by a TFloatField, a TBCDField, and a TVariantField – regardless of the value in EnableBCD. Select field classes on a field-by-field basis for greater flexibility.

Note: For numeric values with more than 4 digits to the right of the decimal place, use of TFloatField is generally better. This is because TBCDField uses the currency data type that has a fixed scale of 4 decimal places.

Note: For fields with very large numbers of more than 19 significant digits, you can use TVariantField type persistent field objects. The TFloatField and TBCDField classes lack sufficient capacity for fields of this size. TVariantField allows the getting and setting of the field data as strings, preventing without loss of data due to insufficient capacity. However, arithmetic operations cannot be performed on numbers accessed through a TVariantField object.

See Also

TBCDField (see page 6776)

TFloatField (see page 7229)

8.1.5.3.7 TADODataSet.FieldDefs Property

Points to the list of field definitions for the dataset.

Pascal

```
property FieldDefs: TFieldDefs;
```

C++

```
__property TFieldDefs FieldDefs;
```

Description

FieldDefs lists the field definitions for a dataset. While an application can examine FieldDefs to explore the field definitions for a dataset, it should not change these definitions unless creating a new table with CreateTable or CreateDataSet.

To access fields and field values in a dataset, use the Fields, AggFields, and FieldValues properties, and the FieldsByName method.

Note: If the dataset includes object field descendants, FieldDefs represents a hierarchical view of the data, meaning that the definitions include object field definitions. To determine the definitions in a flattened view, use FieldDefList instead.

C++ Examples:

```
/*
The following code creates and activates a client dataset in
the form's OnCreate event handler.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TFieldDefs *pDefs = CDS2->FieldDefs;
  TFieldDef *pDef = pDefs->AddFieldDef();
  pDef->DataType = ftInteger;
  pDef->Name = "Field1";

  pDef = pDefs->AddFieldDef();
  pDef->DataType = ftString;
  pDef->Size = 10;
  pDef->Name = "Field2";

  TIndexDef *pIDef = CDS2->IndexDefs->AddIndexDef();
  pIDef->Fields = "Field1";
  pIDef->Name = "IntIndex";

  CDS2->CreateDataSet();
}
```

Delphi Examples:

```
{
The following code creates and activates a client dataset in
the form's OnCreate event handler.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  with CDS2 do
  begin
```

```
      with FieldDefs.AddFieldDef do
      begin
        DataType := ftInteger;
        Name := 'Field1';
      end;
      with FieldDefs.AddFieldDef do
      begin
        DataType := ftString;
        Size := 10;
        Name := 'Field2';
      end;
      with IndexDefs.AddIndexDef do
      begin
        Fields := 'Field1';
        Name := 'IntIndex';
      end;
      CreateDataSet;
    end;
  end;
  { The following example shows how to create a table. }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Table1.Close;
    Table1.DeleteTable;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    Table1:= TTable.Create(Form1);
    with Table1 do
    begin
      DatabaseName := 'DBDEMOS';
      TableType := ttParadox;
      TableName := 'CustInfo';
      Table1.Active := False;
      { Don't overwrite an existing table }
      if Table1.Exists then
        MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
      else
      begin
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
        { Next, describe the fields in the table }
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
```

```
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.Active:= True;
  end;
```

See Also

CreateTable (see page 10189)

Fields (see page 7023)

FieldValues (see page 7025)

ClearFields (see page 6919)

GetFieldList (see page 6953)

GetFieldNames (see page 6954)

FieldByName (see page 6936)

FindField (see page 6938)

SetFields (see page 6997)

ObjectView (see page 7031)

AggFields (see page 7006)

CreateDataSet (see page 8401)

Creating and Deleting Tables

8.1.5.3.8 TADODataSet.IndexDefs Property

Contains the definitions of the indexes for the current table.

Pascal

```
property IndexDefs: TIndexDefs;
```

C++

```
__property TIndexDefs IndexDefs;
```

Description

IndexDefs contains definitions for each index the current table has. IndexDefs is a collection of TIndexDef objects. Each TIndexDef object contains the definition for one index.

Inspect IndexDefs at runtime to get information on a given index. Use an index with IndexDefs to address a specific TIndexDef object. Use properties of TIndexDef to evaluate various index attributes. The example below retrieves the fields list for the third index of the table accessed through the component ADOTable1.

```
ADOTable1.IndexDefs.Update;
StrVar := ADOTable1.IndexDefs[2].Fields;
ADOTable1->IndexDefs->Update();
StrVar = ADOTable1->IndexDefs[2] ->Fields;
```

Call the Update method (TIndexDefs) prior to inspecting the property to ensure it contains the most current index information for the table.

See Also

TIndexDef (see page 7283)

TIndexDefs (see page 7292)

Update (see page 7299)

8.1.5.3.9 TADODataSet.IndexFieldNames Property

Lists the columns to use for a persistent sort of the recordset.

Pascal

```
property IndexFieldNames: string;
```

C++

```
__property AnsiString IndexFieldNames;
```

Description

Use IndexFieldNames as an alternative method of specifying the sort order of a recordset. IndexFieldNames is a direct implementation of the Sort property of the ADO recordset object, with the field list from IndexFieldNames being used for the Sort property. The actual sorting of data occurs on the OLE DB side and use of indexes for the sort depend on the particular provider.

To use IndexFieldNames to establish a recordset's sort order, the recordset must have been opened using a client-side cursor (the dataset component's CursorLocation property must be clUseClient). An exception is raised on attempts to use IndexFieldNames with server-side cursors.

IndexFieldNames may be used to set the sort order for the detail table in a master-detail link, increasing the performance of the link. However, use of IndexFieldNames is not required for master-detail links.

See Also

IndexName (see page 1217)

Creating an Index with IndexFieldNames

8.1.5.3.10 TADODataSet.IndexName Property

Specifies the currently active index.

Pascal

```
property IndexName: WideString;
```

C++

```
__property BSTR IndexName;
```

Description

Use IndexName to activate an index and cause it to actively order the dataset's rows. At runtime, set IndexName to a string containing the name of the index. At design-time, select the desired index from the drop-down list in the Object Inspector.

```
ADOTable1.IndexName := 'LastName';
ADOTable1->IndexName = "LastName";
```

Set IndexName to an empty string to have no index actively ordering the dataset. The rows then appear in natural order. In some database systems, the primary index of a table does not have a name. In these cases, setting IndexName to an empty string activates the primary index.

An index must be active for certain index dependent operations to be used, such as the Seek method.

Note: Many providers do not support changing indexes while the dataset is active, raising an exception when an attempt is made to do this. Ensure that the dataset is closed before changing the index through the IndexName property and reopen the dataset after specifying a new index.

See Also

Seek (see page 1192)

IndexDefs (see page 1215)

IndexFieldNames (see page 1216)

IndexFieldCount (see page 1216)

8.1.5.3.11 TADODataSet.MasterFields Property

Specifies field or fields on which a Master-Detail link is based.

Pascal

```
property MasterFields: WideString;
```

C++

```
__property BSTR MasterFields;
```

Description

Use MasterFields after setting the MasterSource or DataSource property to specify the names of one or more fields in another, master, dataset that are used to establish a Master-Detail relationship between this dataset and the master dataset. The master dataset is specified by assigning its data source to the MasterSource or DataSource property.

MasterFields is a string containing one or more field names in the master dataset. When the datasets are related based on two fields, separate field names with semicolons:

```
ADOTable2.MasterSource := DataSource1;
ADOTable2.MasterFields := 'CustID;SaleDate'
ADOTable2->MasterSource = DataSource1;
ADOTable2->MasterFields = "CustID;SaleDate"
```

Each time the current record in the master dataset changes, the new values in those fields are used to select corresponding records in this table for display.

Note: At design time, use the Field Link designer to establish the master-detail relationship between two tables.

See Also

MasterSource (see page 1144)

DataSource (see page 7014)

Making the Table a Detail of Another Dataset

8.1.5.3.12 TADODataSet.ParamCheck Property

Specifies whether the parameter list for an ADO dataset component is regenerated if the SQL changes at runtime.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

Set ParamCheck to specify whether or not the Parameters property is initialized using ":Param" style parameters in an SQL statement specified in the CommandText property. If the SQL statement in CommandText only contains "?" style parameters, the Parameters property will only be initialized if the OLE DB provider returns parameter information for an SQL command. Otherwise, the parameters must be explicitly created using the CreateParameter method (TParameter).

When ParamCheck is true, a parameter is generated for each ":Param" style parameter in the SQL statement.

The default value of ParamCheck is true.

This property is useful for data definition language (DDL) statements that contain parameters as part of the DDL statement and that are not parameters for the ADO dataset component. For example, the DDL statement to create a stored procedure may contain parameter statements that are part of the stored procedure. Set ParamCheck to false to prevent these parameters from being mistaken for parameters of the ADO dataset component executing the DDL statement.

See Also

TADOQuery (see page 1084)

TADOStoredProc (see page 1106)

Parameters (see page 1220)

Open (see page 6976)

commandtext (see page 1204)

8.1.5.3.13 TADODataSet.Parameters Property

Contains the collection of parameters for an SQL statement.

Pascal

```
property Parameters: TParameters;
```

C++

```
__property TParameters Parameters;
```

Description

At design-time, use Parameters to access the parameters for the SQL statement used by the ADO component. Use the properties and methods of Parameters to set and get individual parameter values or to set or inspect parameter attributes.

At runtime, use the Parameters property editor to access the collection of parameters.

See Also

SQL (see page 1105)

ExecSQL (see page 1100)

8.1.5.3.14 TADODataSet.Prepared Property

Specifies whether the command is prepared before execution.

Pascal

```
property Prepared: Boolean;
```

C++

```
__property Boolean Prepared;
```

Description

Set Prepared before calling the Open method to specify whether ADO prepares the command used to create the dataset's recordset. If Prepared is set to true and the dataset component is connected to a data store, ADO prepares the command before executing it. If Prepared is set to false, ADO does not prepare the command.

The default value of Prepared is false.

See Also

Parameters (see page 1220)

ExecSQL (see page 1100)

Open (see page 6976)

8.1.5.3.15 TADODataSet.RDSConnection Property

Specifies the RDS connection component to use.

Pascal

```
property RDSConnection: TRDSConnection;
```

C++

```
__property TRDSConnection RDSConnection;
```

Description

Use RDSConnection to specify the RDS connection component to use to obtain the recordset when the TADODataSet (see page 1041) is opened. This property is mutually exclusive of the Connection (see page 1205) property. If the TADODataSet (see page 1041) uses RDSConnection to connect to the default RDS DataFactory, then the ConnectionString (see page 1206) property should also be set with the connection information required by the DataFactory. The CommandText (see page 1057) property should either specify the name of a property on the associated business object or be an SQL query.

See Also

TRDSConnection (see page 1274)

Using TADODataSet

8.1.5.3.16 TADODataSet.StoreDefs Property

Specifies whether field definitions are stored with the ADO dataset.

Pascal

```
property StoreDefs: Boolean;
```

C++

```
__property Boolean StoreDefs;
```

Description

StoreDefs controls whether the FieldDefs property value is saved with the ADO dataset when it is saved to a form (.dfm) file. By default, StoreDefs is false unless you explicitly change any of the field definitions.

See Also

FieldDefs (see page 7019)

8.1.5.4 TADODataSet Events

The events of the TADODataSet class are listed here.

Published Events

	Name	Description
	AfterCancel (see page 1067)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 1069)	Occurs after an application closes a dataset.
	AfterDelete (see page 1071)	Occurs after an application deletes a record.
	AfterEdit (see page 1073)	Occurs after an application starts editing a record.
	AfterInsert (see page 1074)	Occurs after an application inserts a new record.
	AfterOpen (see page 1075)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 1076)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterScroll (see page 1076)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 1077)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 1077)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 1078)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 1078)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 1079)	Occurs before an application enters insert mode.
	BeforeOpen (see page 1080)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 1081)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeScroll (see page 1082)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 1082)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 1083)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 1083)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnNewRecord (see page 1084)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 1084)	Occurs when an application attempts to modify or insert a record and an exception is raised.

8.1.5.4.1 TADODataSet.AfterCancel Event

Occurs after an application completes a request to cancel modifications to the active record.

Pascal

```
property AfterCancel: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterCancel;
```

Description

Write an AfterCancel event handler to take specific action after an application cancels changes to the active record. AfterCancel is called by the Cancel method after it updates the current position, releases the lock on the active record if necessary, and sets the dataset state to dsBrowse. If an application requires additional processing before returning control to a user after a Cancel event, code it in the AfterCancel event.

Delphi Examples:

```
{
This example updates the form's status bar with a message
when an AfterCancel event occurs.  Set the StatusBar
SimplePanel property to True.
}
procedure TForm1.Table1AfterCancel(DataSet: TDataSet);
begin
  StatusBar1.SimpleText := 'Record changes cancelled for ' + DataSet.Name;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Table1.Cancel;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  Table1.Post
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Name := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
    begin
        Table1.Close;
        Table1.DeleteTable;
    end;
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
```

```
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.AfterCancel:= Table1AfterCancel;
    Table1.Active:= True;
  end;
```

See Also

BeforeCancel (see page 7048)

Cancel (see page 6916)

State (see page 7036)

8.1.5.4.2 TADODataSet.AfterClose Event

Occurs after an application closes a dataset.

Pascal

```
property AfterClose: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterClose;
```

Description

Write an AfterClose event handler to take specific action immediately after an application closes a dataset. For example, as a security measure, an application might clear a PASSWORD entry from the Params property of a TSQLConnection component when the dataset is closed.

AfterClose is called after a dataset is closed and the dataset state is set to dsInactive.

Delphi Examples:

```
{
This example adds an entry to a memo with a message when an
AfterOpen event or an AfterClose event occurs.  Populate the
ClientDataSet events with CheckButtonActive.
}
procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  if (Button1.Enabled) then
    Memo2.Lines.Add('Now that the dataset is open, record information is available.')
  else
    Memo2.Lines.Add('Now that the dataset is closed, record information is not available.');
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
  RemoteServer1.Connected := False;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].DisableControls;
  end;
end;

procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
```

See Also

Active (see page 7004)

State (see page 7036)

BeforeClose (see page 7048)

Close (see page 6920)

Params (see page 20253)

Opening and Closing Datasets

8.1.5.4.3 TADODataSet.AfterDelete Event

Occurs after an application deletes a record.

Pascal

```
property AfterDelete: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterDelete;
```

Description

Write an AfterDelete event handler to take specific action immediately after an application deletes the active record in a dataset. AfterDelete is called by Delete after it deletes the record, sets the dataset state to dsBrowse, and repositions the current record.

C++ Examples:

```
/*
This example displays a message on the form's status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
*/
void __fastcall TForm1::Table1AfterDelete(TDataSet *DataSet)
{
  StatusBar1->SimpleText = Format(
    "There are now %d records in the table",
    ARRAYOFCONST(((int)DataSet->RecordCount)));
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Table1->Delete();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Table1 = new TTable(Form1);
  Table1->Active = false; // The Table component must not be active
  Table1->DatabaseName = "DBDEMOS";
  Table1->TableType = ttParadox;
  Table1->TableName = "CustInfo";
  Table1->Active = False;
  if (!Table1->Exists) // Don't overwrite an existing table
  {
    Table1->Close();
    Table1->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Table1->FieldDefs->Clear();
  TFieldDef *pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Table1->IndexDefs->Clear();
```

```
  /* the 1st index has no name because it is a Paradox primary key */
  Table1->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Table1->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Table1->CreateTable();
  Table1->Active = True;
  for (int i = 1; i <= 20; i++)
    Table1->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  DS2->DataSet = Table1;
  DBGrid2->DataSource->DataSet = Table1;
  Table1->AfterDelete = Table1AfterDelete;
  Table1->Active = True;
}
```

Delphi Examples:

```
{
This example displays a message on the form’s status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
}
procedure TForm1.Table1AfterDelete(DataSet: TDataSet);
begin
  StatusBar1.SimpleText := Format('There are now %d records in the table',
[DataSet.RecordCount]);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Delete;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Name := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
    begin
        Table1.Close;
        Table1.DeleteTable;
    end;
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
```

```
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.AfterDelete:= Table1AfterDelete;
    Table1.Active:= True;
  end;
```

See Also

State (see page 7036)

BeforeDelete (see page 7049)

Delete (see page 6924)

8.1.5.4.4 TADODataSet.AfterEdit Event

Occurs after an application starts editing a record.

Pascal

```
property AfterEdit: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterEdit;
```

Description

Write an AfterEdit event handler to take specific action immediately after dataset enters edit mode. AfterEdit is called by Edit after it enables editing of a record, recalculates calculated fields, and calls the data event handler to process a record change.

Delphi Examples:

```
{
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
```

```
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

BeforeEdit (see page 7049)

Edit (see page 6932)

8.1.5.4.5 TADODataSet.AfterInsert Event

Occurs after an application inserts a new record.

Pascal

```
property AfterInsert: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterInsert;
```

Description

Write an AfterInsert event handler to take specific action immediately after an application inserts a record. The Insert and Append methods generate an AfterInsert event after inserting or appending a new record.

Delphi Examples:

```
{
This example updates a memo with a message when an
AfterInsert event or a BeforeInsert event occurs.
}
procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After inserting a record');
end;

procedure TForm1.ClientDataSet1BeforeInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before inserting a record');
end;
```

See Also

BeforeInsert (see page 7050)

Append (see page 6910)

Insert (see page 6959)

8.1.5.4.6 TADODataSet.AfterOpen Event

Occurs after an application completes opening a dataset and before any data access occurs.

Pascal

```
property AfterOpen: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterOpen;
```

Description

Write an AfterOpen event handler to take specific action immediately after an application opens the dataset. AfterOpen is called after the dataset establishes access to its data and the dataset is put into dsBrowse state. For example, an AfterOpen event handler might check an ini file to determine the last record touched in the dataset the previous time the application ran, and position the dataset at that record.

Delphi Examples:

```
{
This example adds an entry to a memo with a message when an
AfterOpen event or an AfterClose event occurs.  Populate the
ClientDataSet events with CheckButtonActive.
}
procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  if (Button1.Enabled) then
    Memo2.Lines.Add('Now that the dataset is open, record information is available.')
  else
    Memo2.Lines.Add('Now that the dataset is closed, record information is not available.');
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
  RemoteServer1.Connected := False;
  with RemoteServer1 do
  begin
```

```
    for I := 0 to DataSetCount - 1 do
      DataSets[I].DisableControls;
  end;
end;

procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
```

See Also

Active (see page 7004)

State (see page 7036)

BeforeOpen (see page 7051)

Open (see page 6976)

Opening and Closing Datasets

8.1.5.4.7 TADODataSet.AfterPost Event

Occurs after an application writes the active record to the database or change log and returns to browse state.

Pascal

```
property AfterPost: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterPost;
```

Description

Write an AfterPost event handler to take specific action immediately after an application posts a change to the active record. AfterPost is called after a modification or insertion is made to a record.

Delphi Examples:

```
{
This example updates the form's status bar with a message
when an AfterPost event occurs.
}
procedure TForm1.ClientDataSet1AfterPost(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before posting a record');
end;
```

See Also

BeforePost (see page 7052)

Post (see page 6977)

8.1.5.4.8 TADODataSet.AfterScroll Event

Occurs after an application scrolls from one record to another.

Pascal

```
property AfterScroll: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterScroll;
```

Description

Write an AfterScroll event handler to take specific action immediately after an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. AfterScroll is called after all other events triggered by these methods and any other methods that switch from record to record in the dataset.

See Also

BeforeScroll (see page 7053)

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Prior (see page 6979)

Navigating Datasets

8.1.5.4.9 TADODataSet.BeforeCancel Event

Occurs before an application executes a request to cancel changes to the active record.

Pascal

```
property BeforeCancel: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeCancel;
```

Description

Write a BeforeCancel event to take specific action before an application carries out a request to cancel changes. BeforeCancel is called by the Cancel method before it cancels a dataset operation such as Edit, Insert, or Delete.

An application might use the BeforeCancel event to record a user's changes in an undo buffer.

See Also

AfterCancel (see page 7038)

Delete (see page 6924)

Edit (see page 6932)

Insert (see page 6959)

8.1.5.4.10 TADODataSet.BeforeClose Event

Occurs immediately before the dataset closes.

Pascal

```
property BeforeClose: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeClose;
```

Description

Write a BeforeClose event to take specific action before an application closes a dataset. Calling Close or setting the Active property to false results in a call to the BeforeClose event handler.

See Also

Active (see page 7004)

AfterClose (see page 7040)

Close (see page 6920)

Opening and Closing Datasets

Posting Data

8.1.5.4.11 TADODataSet.BeforeDelete Event

Occurs before an application attempts to delete the active record.

Pascal

```
property BeforeDelete: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeDelete;
```

Description

Write a BeforeDelete event handler to take specific action before an application deletes the active record. BeforeDelete is called by Delete before it actually deletes a record.

In Delphi, making use of this event an application might, for example, display a dialog box asking for confirmation before deleting the record. On denial of confirmation, the application could abort the deletion by calling the Abort procedure.

See Also

AfterDelete (see page 7041)

Delete (see page 6924)

Abort (see page 22786)

8.1.5.4.12 TADODataSet.BeforeEdit Event

Occurs before an application enters edit mode for the active record.

Pascal

```
property BeforeEdit: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeEdit;
```

Description

Write a BeforeEdit event handler to take specific action before an application enables editing of the active record. For example, an application that keeps a log of database edits could use the BeforeEdit event to record the edit request, time, and user before entering edit state.

Delphi Examples:

```
{
```

```
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

State (see page 7036)

AfterEdit (see page 7044)

Edit (see page 6932)

8.1.5.4.13 TADODataSet.BeforeInsert Event

Occurs before an application enters insert mode.

Pascal

```
property BeforeInsert: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeInsert;
```

Description

Write a BeforeInsert event handler to take specific action before an application inserts or appends a new record. The Insert or Append method generates a BeforeInsert method before it sets the dataset into dsInsert state.

Delphi Examples:

```
{
This example updates a memo with a message when an
AfterInsert event or a BeforeInsert event occurs.
}
procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After inserting a record');
end;

procedure TForm1.ClientDataSet1BeforeInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before inserting a record');
end;
{
This example uses the BeforeInsert event to do data
validation. If the StrToInt function raises an exception,
the edit control's contents are set to a valid value so
that the assignment to the INTEGER field in the dataset
```

```
will succeed.  Continuing after the exception causes a
'0' to be placed in the PORTA field.
DataSet design:
This example requires a ClientDataSet, a DBGrid, a
DataSource and a DBNavigator.  Name the ClientDataSet CDS.
Set the DataSet property in the DataSource to CDS and name
that DataSource DS.  Set the DataSource in the DBNavigator
to DS the DataSource in the DBGrid to DS.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
type to Integer.  The file used to load CDS must have an
attribute name of HOST with an attribute type of string and
an attribute name of PORTA with an attribute type of integer.
Add DB, DBCtrls, Grids, DBGrids and DBClient to the uses clause.
}
{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button3Click(Sender: TObject);
begin
  CDS.Insert;
  CDS.FieldByName('PORTA').AsInteger := StrToInt(Edit1.Text);
  CDS.Post;
end;

procedure TForm1.CDSBeforeInsert(DataSet: TDataSet);
begin
  try
  { Make sure edit field can be converted to integer --
   this will raise an exception if it can't }
    StrToInt(Edit1.Text);
  except
    Edit1.Text := '0';
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;
```

See Also

State (see page 7036)

AfterInsert (see page 7044)

Append (see page 6910)

Insert (see page 6959)

8.1.5.4.14 TADODataSet.BeforeOpen Event

Occurs before an application executes a request to open a dataset.

Pascal

```
property BeforeOpen: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeOpen;
```

Description

Write a BeforeOpen event handler to take specific action before an application opens a dataset for viewing or editing. BeforeOpen is triggered when an application sets the Active property to true for a dataset or an application calls Open.

See Also

Active (see page 7004)

AfterOpen (see page 7045)

Open (see page 6976)

Opening and Closing Datasets

8.1.5.4.15 TADODataSet.BeforePost Event

Occurs before an application posts changes for the active record to the database or change log.

Pascal

```
property BeforePost: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforePost;
```

Description

Write a BeforePost event handler to take specific action before an application posts dataset changes. BeforePost is triggered when an application calls the Post method. Post checks to make sure all required fields are present, then calls BeforePost before posting the record.

An application might use BeforePost to perform validity checks on data changes before committing them. If it encountered a validity problem, it could call Abort to cancel the Post operation (Delphi) or throw an exception (C++).

Delphi Examples:

```
{
This example checks for a valid entry in a TDBEdit control
and calls the Abort procedure if the control is empty; Abort
cancels the post before it happens.
}
procedure TForm1.ClientDataSet1BeforePost(DataSet: TDataSet);
begin
  if DBEdit1.Text = '' then
    Abort;
  DataSet.Fields[2].AsString := DBEdit1.Text;
end;
```

See Also

AfterPost (see page 7046)

Post (see page 6977)

Abort (see page 22786)

Working with Field Component Methods at Runtime

8.1.5.4.16 TADODataSet.BeforeScroll Event

Occurs before an application scrolls from one record to another.

Pascal

```
property BeforeScroll: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeScroll;
```

Description

Write a BeforeScroll event handler to take specific action immediately before an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. BeforeScroll is called before all other events triggered by these methods and any other methods that switch from record to record in the dataset.

See Also

AfterScroll (see page 7047)

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Prior (see page 6979)

Navigating Datasets

8.1.5.4.17 TADODataSet.OnCalcFields Event

Occurs when an application recalculates calculated fields.

Pascal

```
property OnCalcFields: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent OnCalcFields;
```

Description

Write an OnCalcFields event handler to take specific action when an application recalculates calculated fields. A calculated field is one that derives its value from the values in one or more fields in the dataset, sometimes with additional processing.

OnCalcFields is triggered when:

A dataset is opened.

A dataset is put into dsEdit state.

A record is retrieved from a database.

When the AutoCalcFields property is true, OnCalcFields is also triggered when:

Focus moves from one visual control to another, or from one column to another is a data-aware grid control and modifications were made to the record.

Note: When the AutoCalcFields property is true, an OnCalcFields event handler should not modify the dataset (or a linked

dataset if it is part of a master-detail relationship), because such modifications retrigger the OnCalcField event, leading to infinite recursion.

If an application permits users to change data, OnCalcFields is frequently triggered. To reduce the frequency with which OnCalcFields occurs, set AutoCalcFields to false.

Warning: When the dataset is the master table of a master-detail relationship, OnCalcFields occurs before detail sets have been synchronized with the master table.

See Also

Active (see page 7004)

AutoCalcFields (see page 7006)

Open (see page 6976)

Post (see page 6977)

Calculating Fields

Using Internally Calculated Fields in Client Datasets

Defining a Calculated Field

8.1.5.4.18 TADODataSet.OnDeleteError Event

Occurs when an application attempts to delete a record and an exception is raised.

Pascal

```
property OnDeleteError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnDeleteError;
```

Description

Write an OnDeleteError event handler to handle exceptions that occur when an attempt to delete a record fails.

See Also

OnEditError (see page 7054)

OnPostError (see page 7056)

Delete (see page 6924)

8.1.5.4.19 TADODataSet.OnEditError Event

Occurs when an application attempts to modify or insert a record and an exception is raised.

Pascal

```
property OnEditError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnEditError;
```

Description

Write an OnEditError event handler to handle exceptions that occur when an attempt to edit a record fails.

See Also

OnDeleteError (see page 7054)

OnPostError (see page 7056)

Edit (see page 6932)

Insert (see page 6959)

8.1.5.4.20 TADODataSet.OnNewRecord Event

Occurs when an application inserts or appends a new dataset record.

Pascal

```
property OnNewRecord: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent OnNewRecord;
```

Description

Write an OnNewRecord event handler to take specific actions as an application inserts or appends a new record. OnNewRecord is called as part of the insert or append process. An application might use the OnNewRecord event to set initial values for a record or as a way of implementing cascading insertions in related datasets.

See Also

AfterInsert (see page 7044)

BeforeInsert (see page 7050)

8.1.5.4.21 TADODataSet.OnPostError Event

Occurs when an application attempts to modify or insert a record and an exception is raised.

Pascal

```
property OnPostError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnPostError;
```

Description

Write an OnPostError event handler to handle exceptions that occur when an attempt to post a record fails.

See Also

OnDeleteError (see page 7054)

OnEditError (see page 7054)

Post (see page 6977)

8.1.6 TADOQuery Class

TADOQuery provides the means for issuing SQL (see page 1105) against an ADO data store.

Class Hierarchy

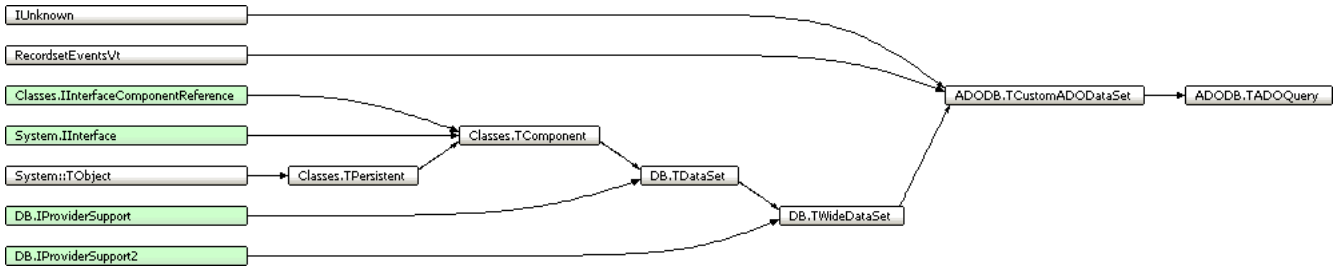

Pascal

```
TADOQuery = class(TCustomADODataSet);
```

C++

```
class TADOQuery : public TCustomADODataSet;
```

File

ADODB

Description

Use TADOQuery to access one or more tables in a data store using SQL (see page 1105) statements.

Retrieve data from tables in an ADO data store using SELECT statements. Perform actions on tables and other metadata objects in an ADO data store with statements like INSERT, DELETE, UPDATE, ALTER TABLE, and CREATE TABLE. Execute stored procedures.

See Also

TDataSet (see page 6894)

TADODataSet (see page 1041)

TADOConnection (see page 1001)

TADOTable (see page 1125)

TADOStoredProc (see page 1106)

Types of Datasets

Overview of ADO Components

8.1.6.1 TADOQuery Members

The following tables list the members exposed by TADOQuery.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.

	ActivateTextFilter (see page 1160)	This is ActivateTextFilter, a member of class TCustomADODataSet.
	AllocRecordBuffer (see page 1160)	This is AllocRecordBuffer, a member of class TCustomADODataSet.
	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckActive (see page 1162)	This is CheckActive, a member of class TCustomADODataSet.
	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
	CheckFieldCompatibility (see page 1162)	This is CheckFieldCompatibility, a member of class TCustomADODataSet.
	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
	ClearCalcFields (see page 1162)	This is ClearCalcFields, a member of class TCustomADODataSet.
	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
	ConnectionPoint (see page 1163)	This is ConnectionPoint, a member of class TCustomADODataSet.
	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
	DataEvent (see page 1164)	This is DataEvent, a member of class TCustomADODataSet.
	DataEvent (see page 6924)	This is DataEvent, a member of class TDataSet.
	DeactivateFilters (see page 1164)	This is DeactivateFilters, a member of class TCustomADODataSet.
	DefChanged (see page 1165)	This is DefChanged, a member of class TCustomADODataSet.
	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
	DefineProperties (see page 1165)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
	DestroyLookupCursor (see page 1166)	This is DestroyLookupCursor, a member of class TCustomADODataSet.
	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.

	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 1166)	This is DoOnNewRecord, a member of class TCustomADODataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	EnableEvents (see page 1166)	This is EnableEvents, a member of class TCustomADODataSet.
	EndOfRecordset (see page 1167)	This is EndOfRecordset, a member of class TCustomADODataSet.
	FetchComplete (see page 1167)	This is FetchComplete, a member of class TCustomADODataSet.
	FetchProgress (see page 1167)	This is FetchProgress, a member of class TCustomADODataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FieldChangeComplete (see page 1167)	This is FieldChangeComplete, a member of class TCustomADODataSet.
	FindRecord (see page 1168)	This is FindRecord, a member of class TCustomADODataSet.
	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeRecordBuffer (see page 1168)	This is FreeRecordBuffer, a member of class TCustomADODataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
	GetActiveRecBuf (see page 1168)	This is GetActiveRecBuf, a member of class TCustomADODataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
	GetBookmarkData (see page 1169)	This is GetBookmarkData, a member of class TCustomADODataSet.
	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
	GetBookmarkFlag (see page 1169)	This is GetBookmarkFlag, a member of class TCustomADODataSet.
	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 1169)	This is GetCanModify, a member of class TCustomADODataSet.
	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
	GetCommandText (see page 1170)	This is GetCommandText, a member of class TCustomADODataSet.
	GetDataSource (see page 1170)	This is GetDataSource, a member of class TCustomADODataSet.
	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
	GetRecNo (see page 1171)	This is GetRecNo, a member of class TCustomADODataSet.
	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
	GetRecord (see page 1172)	This is GetRecord, a member of class TCustomADODataSet.
	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
	GetRecordCount (see page 1172)	This is GetRecordCount, a member of class TCustomADODataSet.
	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
	GetRecordSize (see page 1172)	This is GetRecordSize, a member of class TCustomADODataSet.
	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
	GetStateFieldValue (see page 1172)	This is GetStateFieldValue, a member of class TCustomADODataSet.
	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.

	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
	InitOleDBAccess (see page 1173)	This is InitOleDBAccess, a member of class TCustomADODataSet.
V	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
V	InternalAddRecord (see page 1173)	This is InternalAddRecord, a member of class TCustomADODataSet.
V	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
V	InternalCancel (see page 1173)	This is InternalCancel, a member of class TCustomADODataSet.
V	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
V	InternalClose (see page 1173)	This is InternalClose, a member of class TCustomADODataSet.
A	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.
V	InternalDelete (see page 1173)	This is InternalDelete, a member of class TCustomADODataSet.
V	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
V	InternalEdit (see page 1174)	This is InternalEdit, a member of class TCustomADODataSet.
V	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
V	InternalFirst (see page 1174)	This is InternalFirst, a member of class TCustomADODataSet.
V	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
	InternalGetRecord (see page 1174)	This is InternalGetRecord, a member of class TCustomADODataSet.
V	InternalGotoBookmark (see page 1174)	This is InternalGotoBookmark, a member of class TCustomADODataSet.
V	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
V	InternalHandleException (see page 1175)	This is InternalHandleException, a member of class TCustomADODataSet.
A	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
V	InternalInitFieldDefs (see page 1175)	This is InternalInitFieldDefs, a member of class TCustomADODataSet.
A	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
V	InternalInitRecord (see page 1175)	This is InternalInitRecord, a member of class TCustomADODataSet.
V	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
V	InternalInsert (see page 1175)	This is InternalInsert, a member of class TCustomADODataSet.
V	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
V	InternalLast (see page 1175)	This is InternalLast, a member of class TCustomADODataSet.
V	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
V	InternalOpen (see page 1176)	This is InternalOpen, a member of class TCustomADODataSet.
A	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
V	InternalPost (see page 1176)	This is InternalPost, a member of class TCustomADODataSet.
V	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
V	InternalRefresh (see page 1176)	This is InternalRefresh, a member of class TCustomADODataSet.
V	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
	InternalRequery (see page 1176)	This is InternalRequery, a member of class TCustomADODataSet.
	InternalSetSort (see page 1177)	This is InternalSetSort, a member of class TCustomADODataSet.
V	InternalSetToRecord (see page 1177)	This is InternalSetToRecord, a member of class TCustomADODataSet.
V	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsCursorOpen (see page 1177)	This is IsCursorOpen, a member of class TCustomADODataSet.
A	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
V	Loaded (see page 1177)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1179)	This is LocateRecord, a member of class TCustomADODataSet.
V	MasterChanged (see page 1180)	This is MasterChanged, a member of class TCustomADODataSet.
V	MasterDisabled (see page 1180)	This is MasterDisabled, a member of class TCustomADODataSet.
	MoveComplete (see page 1180)	This is MoveComplete, a member of class TCustomADODataSet.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	OpenCursor (see page 1181)	This is OpenCursor, a member of class TCustomADODataSet.

	OpenCursor (see page 6976)	This is OpenCursor, a member of class TDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PrepareCursor (see page 1181)	This is PrepareCursor, a member of class TCustomADODataSet.
	PSEndTransaction (see page 1181)	Ends a transaction that was started by a call to PSStartTransaction.
	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 1182)	Executes the SQL command associated with the dataset.
	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 1182)	Executes a specified SQL command.
	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSExecuteStatement (see page 7535)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 1183)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 1183)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 6982)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 7537)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandTextW (see page 7537)	This is PSGetCommandTextW, a member of class TWideDataSet.
	PSGetCommandType (see page 1184)	Gets the command type to be executed by PSExecute.
	PSGetCommandType (see page 6982)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 1184)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 1185)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7538)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 1185)	This is PSGetKeyFieldsW, a member of class TCustomADODataSet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetKeyFieldsW (see page 7538)	This is PSGetKeyFieldsW, a member of class TWideDataSet.
	PSGetParams (see page 1185)	Returns the current parameter values of the dataset.
	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7538)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 1186)	This is PSGetQuoteCharW, a member of class TCustomADODataSet.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetQuoteCharW (see page 7539)	This is PSGetQuoteCharW, a member of class TWideDataSet.
	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7539)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 1186)	This is PSGetTableNameW, a member of class TCustomADODataSet.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetTableNameW (see page 7539)	This is PSGetTableNameW, a member of class TWideDataSet.
	PSGetUpdateException (see page 1186)	Generates an EUpdateError object based on another exception object.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 1187)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 1187)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 1188)	Indicates whether the dataset can execute SQL statements.

	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 1188)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 1188)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetCommandText (see page 7540)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 1189)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 1189)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 1190)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	QueryChanged (see page 1100)	This is QueryChanged, a member of class TADOQuery.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RecordChangeComplete (see page 1190)	This is RecordChangeComplete, a member of class TCustomADODataSet.
	RecordsetChangeComplete (see page 1190)	This is RecordsetChangeComplete, a member of class TCustomADODataSet.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	RefreshParams (see page 1191)	This is RefreshParams, a member of class TCustomADODataSet.
	ReleaseLock (see page 1191)	This is ReleaseLock, a member of class TCustomADODataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBlockReadSize (see page 1194)	This is SetBlockReadSize, a member of class TCustomADODataSet.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBookmarkData (see page 1194)	This is SetBookmarkData, a member of class TCustomADODataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkFlag (see page 1194)	This is SetBookmarkFlag, a member of class TCustomADODataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCommandText (see page 1194)	This is SetCommandText, a member of class TCustomADODataSet.
	SetConnection (see page 1194)	This is SetConnection, a member of class TCustomADODataSet.
	SetConnectionFlag (see page 1195)	This is SetConnectionFlag, a member of class TCustomADODataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDataSource (see page 1195)	This is SetDataSource, a member of class TCustomADODataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDetailFilter (see page 1195)	This is SetDetailFilter, a member of class TCustomADODataSet.
	SetFieldData (see page 1195)	This is the overview for the SetFieldData method overload.
	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
	SetFiltered (see page 1196)	This is SetFiltered, a member of class TCustomADODataSet.
	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
	SetFilterOptions (see page 1196)	This is SetFilterOptions, a member of class TCustomADODataSet.
	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
	SetFilterText (see page 1196)	This is SetFilterText, a member of class TCustomADODataSet.
	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property

	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
V	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetParamsFromCursor (see page 1197)	This is SetParamsFromCursor, a member of class TCustomADODataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetRecNo (see page 1197)	This is SetRecNo, a member of class TCustomADODataSet.
V	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
V	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateIndexDefs (see page 1198)	This is UpdateIndexDefs, a member of class TCustomADODataSet.
V	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.
	UpdateRecordSetPosition (see page 1198)	This is UpdateRecordSetPosition, a member of class TCustomADODataSet.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WillChangeField (see page 1199)	This is WillChangeField, a member of class TCustomADODataSet.
	WillChangeRecord (see page 1199)	This is WillChangeRecord, a member of class TCustomADODataSet.
	WillChangeRecordset (see page 1200)	This is WillChangeRecordset, a member of class TCustomADODataSet.
	WillMove (see page 1200)	This is WillMove, a member of class TCustomADODataSet.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
R	Command (see page 1204)	Provides access to the internal command object that executes commands for the dataset.
	CommandText (see page 1204)	This is CommandText, a member of class TCustomADODataSet.
	CommandType (see page 1205)	Specifies the type of command to execute.
	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
	FieldDefs (see page 1210)	Points to the list of field definitions for the dataset.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
	IndexDefs (see page 1215)	Contains the definitions of the indexes for the current table.

	IndexFieldNames (see page 1216)	Lists the columns to use for a persistent sort of the recordset.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
R	MasterDataLink (see page 1218)	Manages the relationship between the ADO dataset and a master table.
	MasterFields (see page 1219)	Specifies field or fields on which a Master-Detail link is based.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
	Reserved (see page 7035)	For internal use only.
	StoreDefs (see page 1223)	Specifies whether field definitions are stored with the ADO dataset.

Public Events

	Name	Description
	AfterCancel (see page 7038)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 7040)	Occurs after an application closes a dataset.
	AfterDelete (see page 7041)	Occurs after an application deletes a record.
	AfterEdit (see page 7044)	Occurs after an application starts editing a record.
	AfterInsert (see page 7044)	Occurs after an application inserts a new record.
	AfterOpen (see page 7045)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 7046)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 7047)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 7047)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 7048)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 7048)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 7049)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 7049)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 7050)	Occurs before an application enters insert mode.
	BeforeOpen (see page 7051)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 7052)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 7052)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 7053)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 7053)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 7054)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 7054)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 7056)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 7056)	Occurs when an application attempts to modify or insert a record and an exception is raised.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 1160)	Tests the validity of a specified bookmark.

	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
	CancelBatch (see page 1161)	Cancels a pending batch update.
	CancelUpdates (see page 1162)	Cancels pending cached updates.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Clone (see page 1163)	Duplicates another dataset in the calling component
	Close (see page 6920)	Closes a dataset.
	CompareBookmarks (see page 1163)	Compares two bookmarks.
	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
	Create (see page 1099)	Creates an instance of a TADOQuery (see page 1084) component.
	CreateBlobStream (see page 1164)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
	DeleteRecords (see page 1165)	Deletes one or more records in the dataset.
	Destroy (see page 1099)	Destroys the TADOQuery (see page 1084) instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	ExecSQL (see page 1100)	Executes the SQL (see page 1105) statement for the ADO query.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FilterOnBookmarks (see page 1168)	Filters a data set to just rows with pre-defined bookmarks.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetBlobFieldData (see page 1169)	Retrieves the current value of a field into a buffer.
V	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
V	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
V	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
V	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
V	GetDetailLinkFields (see page 1170)	Lists the field components that link this dataset as a detail of a master dataset.
V	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetFieldData (see page 1171)	This is the overview for the GetFieldData method overload.
V	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
V	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
V	IsSequenced (see page 1177)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
V	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	Last (see page 6968)	Moves to the last record in the dataset.
	LoadFromFile (see page 1178)	Loads a recordset from a file.
V	Locate (see page 1179)	Searches the dataset for a specified record and makes that record the current record.
V	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Lookup (see page 1180)	Retrieves field values from a row that matches specified search values.
V	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	NextRecordset (see page 1181)	Provides access to a secondary recordset.

	Open (see page 6976)	Opens the dataset.
V	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSEndTransaction (see page 7589)	Ends a transaction that was started by a call to PSStartTransaction (see page 7597).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecute (see page 7590)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSExecuteStatement (see page 7590)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 7591)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 7579)	Gets the SQL command to be executed by PSExecute (see page 7578).
	PSGetCommandText (see page 7591)	Gets the SQL command to be executed by PSExecute (see page 7590).
	PSGetCommandType (see page 7580)	Gets the command type to be executed by PSExecute (see page 7578).
	PSGetCommandType (see page 7591)	Gets the command type to be executed by PSExecute (see page 7590).
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 7592)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 7592)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7593)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetParams (see page 7593)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7594)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7594)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSGetUpdateException (see page 7594)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 7595)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 7595)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 7596)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 7596)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7585)	Assigns a specified SQL command to be executed by PSExecute (see page 7578).
	PSSetCommandText (see page 7596)	Assigns a specified SQL command to be executed by PSExecute (see page 7590).
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 7597)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).

	PSStartTransaction (see page 7597)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7589).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 7598)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Requery (see page 1191)	Refreshes the recordset.
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 1192)	Saves a recordset to a file.
	Seek (see page 1192)	Searches for a record using the current index.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	Supports (see page 1197)	Indicates whether a dataset supports certain types of operations.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateBatch (see page 1197)	Writes pending batch updates to disk.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 1199)	Indicates the update status of the current record.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.

Public Properties

	Name	Description
	Active (see page 7004)	Specifies whether or not a dataset is open.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignerData (see page 1208)	Stores custom data.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.

R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
	FieldDefs (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.
	FilterGroup (see page 1214)	Filters a recordset based on row update status.
	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
R	IndexFieldCount (see page 1216)	Indicates the number of fields that comprise the current key.
	IndexFields (see page 1217)	Retrieves or sets a field for an index.
	IndexName (see page 1217)	Specifies the currently active index.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	Modified (see page 7029)	Indicates whether the active record is modified.
	ObjectView (see page 7031)	Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Properties (see page 1221)	This is Properties, a member of class TCustomADODataSet.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
	Recordset (see page 1221)	This is Recordset, a member of class TCustomADODataSet.
R	RecordsetState (see page 1222)	This is RecordsetState, a member of class TCustomADODataSet.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
R	RecordStatus (see page 1222)	Indicates the status of the current row.
R	RowsAffected (see page 1105)	Returns the number of rows operated upon by the latest query execution.
	Sort (see page 1222)	Specifies the sort order of the recordset.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterCancel (see page 1224)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 1226)	Occurs after an application closes a dataset.
	AfterDelete (see page 1227)	Occurs after an application deletes a record.
	AfterEdit (see page 1230)	Occurs after an application starts editing a record.
	AfterInsert (see page 1231)	Occurs after an application inserts a new record.
	AfterOpen (see page 1231)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 1233)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 1233)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 1233)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 1234)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 1234)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 1235)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 1235)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 1236)	Occurs before an application enters insert mode.
	BeforeOpen (see page 1237)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 1238)	Occurs before an application posts changes for the active record to the database or change log.

	BeforeRefresh (see page 1239)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 1239)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 1239)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 1240)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 1241)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnEndOfRecordset (see page 1241)	Occurs when moving a recordset's row pointer past the last row.
	OnFetchComplete (see page 1242)	Occurs after records have been retrieved into the recordset.
	OnFetchProgress (see page 1242)	Occurs periodically during an asynchronous data retrieval operation.
	OnFieldChangeComplete (see page 1243)	Occurs after a field has changed.
	OnFilterRecord (see page 1244)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnMoveComplete (see page 1244)	Occurs after the row pointer in a recordset has moved.
	OnNewRecord (see page 1245)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 1245)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnRecordChangeComplete (see page 1246)	Occurs after one or more records in the recordset change.
	OnRecordsetChangeComplete (see page 1247)	Occurs after the recordset has changed.
	OnRecordsetCreate (see page 1247)	Occurs when the recordset has been initialized and is ready to use.
	OnWillChangeField (see page 1248)	Occurs before a field is changed.
	OnWillChangeRecord (see page 1248)	Occurs before a record in the recordset is changed.
	OnWillChangeRecordset (see page 1249)	Occurs before the recordset has changed.
	OnWillMove (see page 1250)	Occurs before the row pointer in a recordset is moved.

Published Properties

	Name	Description
	Active (see page 1201)	Specifies whether or not a dataset is open.
	AutoCalcFields (see page 1203)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	CacheSize (see page 1203)	Specifies the size of the cache for the dataset.
	CommandTimeout (see page 1101)	Specifies amount of time to attempt execution of a command.
	Connection (see page 1205)	Specifies the ADO connection component to use.
	ConnectionString (see page 1206)	Specifies the connection information for the data store.
	CursorLocation (see page 1207)	Specifies whether the cursor for the connection is client-side or server-side.
	CursorType (see page 1208)	Specifies type of cursor an ADO dataset uses.
	DataSource (see page 1101)	Specifies the data source component from which to extract current field values to use in same-name parameters in the ADO query's SQL (see page 1105) statement.
	EnableBCD (see page 1103)	Specifies whether to treat numeric fields as floating-point or BCD.
	ExecuteOptions (see page 1209)	Specifies the characteristics of a command execution.
	Filter (see page 1212)	Specifies the text of the current filter for a recordset.
	Filtered (see page 1214)	Specifies whether or not filtering is active for a dataset.
	LockType (see page 1218)	Specifies the lock type to use when opening a dataset.
	MarshalOptions (see page 1218)	Specifies which records are marshaled back to the server.
	MaxRecords (see page 1219)	Specifies how many rows to return in a result set.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParamCheck (see page 1103)	Specifies whether the parameter list for an ADO dataset component is regenerated if the SQL changes at runtime.
	Parameters (see page 1104)	Contains the collection of parameters for an SQL statement.
	Prepared (see page 1104)	Specifies whether the command is prepared before execution.
	SQL (see page 1105)	Contains the text of the SQL statement to execute for the ADO query.
	Tag (see page 2808)	Stores an integer value as part of a component.

8.1.6.2 TADOQuery.Create Constructor

Creates an instance of a TADOQuery (see page 1084) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TADOQuery(TComponent * AOwner);
```

Description

Call Create to instantiate an ADO query component at runtime. Query components placed in forms or data modules at design time are created automatically.

See Also

TADOStoredProc (see page 1106)

TADOTable (see page 1125)

Destroy (see page 1099)

8.1.6.3 TADOQuery.Destroy Destructor

Destroys the TADOQuery (see page 1084) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TADOQuery();
```

Description

Do not call Destroy directly in an application. Usually destruction of ADO query components is handled automatically by Delphi. If an application creates its own instances of a query, and does not assign an Owner that is responsible for freeing the dataset, then the application should call Free, which checks that the query is not nil before calling Destroy.

See Also

Designer (see page 7015)

FieldDefs (see page 7019)

Create_d

Free (see page 22300)

8.1.6.4 TADOQuery Methods

The methods of the TADOQuery class are listed here.

Protected Methods

	Name	Description
	QueryChanged (see page 1100)	This is QueryChanged, a member of class TADOQuery.

Public Methods

	Name	Description
	ExecSQL (see page 1100)	Executes the SQL (see page 1105) statement for the ADO query.

8.1.6.4.1 TADOQuery.ExecSQL Method

Executes the SQL (see page 1105) statement for the ADO query.

Pascal

```
function ExecSQL: Integer;
```

C++

```
__fastcall int ExecSQL();
```

Description

Call ExecSQL to execute the SQL (see page 1105) statement currently assigned to the SQL (see page 1105) property. Use ExecSQL to execute queries that do not return a cursor to data (such as INSERT, UPDATE, DELETE, and CREATE TABLE).

ExecSQL returns an integer value reflecting the number of rows affected by the executed SQL (see page 1105) statement.

Note: For SELECT statements, call Open instead of ExecSQL or set the Active (see page 1201) property to true.

To speed performance, an application should ordinarily prepare the query by setting the Prepared (see page 1104) property to true before calling ExecSQL for the first time.

See Also

SQL (see page 1105)

Parameters

SQL (see page 1105)

Active (see page 7004)

Open (see page 6976)

Prepared

ParamByName (see page 1269)

8.1.6.4.2 TADOQuery.QueryChanged Method

Pascal

```
procedure QueryChanged(Sender: TObject);
```

C++

```
__fastcall QueryChanged(TObject * Sender);
```

Description

This is QueryChanged, a member of class TADOQuery.

8.1.6.5 TADOQuery Properties

The properties of the TADOQuery class are listed here.

Public Properties

	Name	Description
R	RowsAffected (see page 1105)	Returns the number of rows operated upon by the latest query execution.

Published Properties

	Name	Description
	CommandTimeout (see page 1101)	Specifies amount of time to attempt execution of a command.
	DataSource (see page 1101)	Specifies the data source component from which to extract current field values to use in same-name parameters in the ADO query's SQL (see page 1105) statement.
	EnableBCD (see page 1103)	Specifies whether to treat numeric fields as floating-point or BCD.
	ParamCheck (see page 1103)	Specifies whether the parameter list for an ADO dataset component is regenerated if the SQL changes at runtime.
	Parameters (see page 1104)	Contains the collection of parameters for an SQL statement.
	Prepared (see page 1104)	Specifies whether the command is prepared before execution.
	SQL (see page 1105)	Contains the text of the SQL statement to execute for the ADO query.

8.1.6.5.1 TADOQuery.CommandTimeout Property

Specifies amount of time to attempt execution of a command.

Pascal

```
property CommandTimeout: Integer;
```

C++

```
__property int CommandTimeout;
```

Description

Use CommandTimeout to specify the amount of time that expires before an attempt to execute a command is considered unsuccessful. Measured in seconds, the default value for CommandTimeout is 30.

If a command is successfully executed prior to the expiration of the seconds specified, CommandTimeout has no effect.

See Also

commandtext (see page 1204)

CommandType (see page 1205)

8.1.6.5.2 TADOQuery.DataSource Property

Specifies the data source component from which to extract current field values to use in same-name parameters in the ADO query's SQL (see page 1105) statement.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;__property TDataSource DataSource;
```

Description

Set DataSource to automatically fill parameters in a query with fields values from another dataset. Parameters (see page 1104) that have the same name as fields in the other dataset are filled with the field values. Parameters (see page 1104) with names that are not the same as fields in the other dataset do not automatically get values, and must be programmatically set. For example, if the SQL (see page 1105) property of the TADOQuery (see page 1084) contains the SQL (see page 1105) statement below and the dataset referenced through DataSource has a CustNo field, the value from the current record in that

other dataset is used in the CustNo parameter:

```
SELECT *
FROM Orders O
WHERE (O.CustNo = :CustNo)
```

DataSource must point to a TDataSource component linked to another dataset component; it cannot point to this query component. The dataset specified in DataSource must be created, populated, and opened before attempting to bind parameters. Parameters (see page 1104) are bound by setting the ADO query's Prepared (see page 1104) property to true prior to executing the query. If the SQL (see page 1105) statement used by a query does not contain parameters, or all parameters are bound by the application using the Parameters (see page 1104) property or the ParamByName method, DataSource need not be assigned. The example below shows setting the DataSource property of ADOQuery2 to the data source for ADOQuery1, preparing ADOQuery2, and activating ADOQuery2:

```
with ADOQuery2 do begin
  DataSource := DataSource1;
  Prepared := True;
  Open;
end;
ADOQuery2->DataSource = DataSource1;
ADOQuery2->Prepared = true;
ADOQuery2->Open();
```

If the SQL (see page 1105) statement in the TADOQuery (see page 1084) is a SELECT query, the query is executed using the new field values each time the record pointer in the other dataset is changed. It is not necessary to call the Open method of the TADOQuery (see page 1084) each time. This makes using the DataSource property to dynamically filter a query result set useful for establishing Master-Detail relationships. Set the DataSource property in the Detail query to the TDataSource component for the Master dataset.

If the SQL (see page 1105) statement uses other than a SELECT query (such as INSERT or UPDATE), the parameters with the same name as fields in the other dataset still get values, but the query must be explicitly executed each time the other dataset's record pointer moves. For example, the SQL (see page 1105) statement below uses the INSERT statement and has the parameters CustNo and CompanyName:

INSERT INTO Customer

(CustNo, Company)

VALUES (:CustNo, :CompanyName)

Another dataset, ADOQuery1 and DataSource1, has a CustNo field but no CompanyName field. If this dataset is used through the DataSource property, the CompanyName parameter must be programmatically assigned a value. Because ADOQuery1 has a CustNo field and ADOQuery1 is referenced through the DataSource property, the CustNo parameter automatically receives a value.

```
with ADOQuery2 do begin
```

DataSource := DataSource1;

```
  ParamByName ('CompanyName').AsString := Edit1.Text;
```

Prepared (see page 1104) := True;

ExecSQL (see page 1100);

```
end;
ADOQuery2->DataSource = DataSource1;
ADOQuery2->ParamByName("CompanyName")->AsString = Edit1->Text;
ADOQuery2->Prepared = true;
ADOQuery2->ExecSQL();
```

If the SQL (see page 1105) statement contains parameters with the same name as fields in the other dataset, do not manually set values for these parameters. Any values programmatically set, such as by using the Parameters (see page 1104) property

or the ParamByName method, will be overridden with automatic values. Parameters (see page 1104) of other names must be programmatically given values. These parameters are unaffected by setting DataSource.

DataSource can be set at runtime or at design-time using the Object Inspector. At design-time, select the desired TDataSource from the drop-down list or type in the name.

See Also

Parameters

Prepared

ExecSQL (see page 1100)

8.1.6.5.3 TADOQuery.EnableBCD Property

Specifies whether to treat numeric fields as floating-point or BCD.

Pascal

```
property EnableBCD: Boolean;
```

C++

```
__property Boolean EnableBCD;
```

Description

Use the EnableBCD property to specify how fields are mapped to field classes. If EnableBCD is true, (the default value) adDecimal and adNumeric fields are mapped to the TBCDField class when field objects are created. If EnableBCD is false, the fields are mapped to the TFloatField class.

EnableBCD determines whether numeric and decimal fields are translated as floating-point values or binary coded decimal (BCD) values. BCD values eliminate the rounding errors associated with floating point math (such as a 3 * (2/3) resulting in 2.00000000001).

If persistent field objects are used (such as those created using the Fields Editor), the field class associated with a given field can contradict the setting of EnableBCD. Three numeric fields in the same table could, for instance, each be represented by a TFloatField, a TBCDField, and a TVariantField – regardless of the value in EnableBCD. Select field classes on a field-by-field basis for greater flexibility.

Note: For numeric values with more than 4 digits to the right of the decimal place, use of TFloatField is generally better. This is because TBCDField uses the currency data type that has a fixed scale of 4 decimal places.

Note: For fields with very large numbers of more than 19 significant digits, you can use TVariantField type persistent field objects. The TFloatField and TBCDField classes lack sufficient capacity for fields of this size. TVariantField allows the getting and setting of the field data as strings, preventing without loss of data due to insufficient capacity. However, arithmetic operations cannot be performed on numbers accessed through a TVariantField object.

See Also

TBCDField (see page 6776)

TFloatField (see page 7229)

8.1.6.5.4 TADOQuery.ParamCheck Property

Specifies whether the parameter list for an ADO dataset component is regenerated if the SQL changes at runtime.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

Set ParamCheck to specify whether or not the Parameters property is initialized using ":Param" style parameters in an SQL statement specified in the CommandText property. If the SQL statement in CommandText only contains "?" style parameters, the Parameters property will only be initialized if the OLE DB provider returns parameter information for an SQL command. Otherwise, the parameters must be explicitly created using the CreateParameter method (TParameter).

When ParamCheck is true, a parameter is generated for each ":Param" style parameter in the SQL statement.

The default value of ParamCheck is true.

This property is useful for data definition language (DDL) statements that contain parameters as part of the DDL statement and that are not parameters for the ADO dataset component. For example, the DDL statement to create a stored procedure may contain parameter statements that are part of the stored procedure. Set ParamCheck to false to prevent these parameters from being mistaken for parameters of the ADO dataset component executing the DDL statement.

See Also

TADOQuery (see page 1084)

TADOStoredProc (see page 1106)

Parameters (see page 1220)

Open (see page 6976)

commandtext (see page 1204)

8.1.6.5.5 TADOQuery.Parameters Property

Contains the collection of parameters for an SQL statement.

Pascal

```
property Parameters: TParameters;
```

C++

```
__property TParameters Parameters;
```

Description

At design-time, use Parameters to access the parameters for the SQL statement used by the ADO component. Use the properties and methods of Parameters to set and get individual parameter values or to set or inspect parameter attributes.

At runtime, use the Parameters property editor to access the collection of parameters.

See Also

SQL (see page 1105)

ExecSQL (see page 1100)

8.1.6.5.6 TADOQuery.Prepared Property

Specifies whether the command is prepared before execution.

Pascal

```
property Prepared: Boolean;
```

C++

```
__property Boolean Prepared;
```

Description

Set Prepared before calling the Open method to specify whether ADO prepares the command used to create the dataset's recordset. If Prepared is set to true and the dataset component is connected to a data store, ADO prepares the command before executing it. If Prepared is set to false, ADO does not prepare the command.

The default value of Prepared is false.

See Also

Parameters (see page 1220)

ExecSQL (see page 1100)

Open (see page 6976)

8.1.6.5.7 TADOQuery.RowsAffected Property

Returns the number of rows operated upon by the latest query execution.

Pascal

```
property RowsAffected: Integer;
```

C++

```
__property int RowsAffected;
```

Description

Inspect RowsAffected to determine how many rows were updated or deleted by the last query operation. If no rows were updated or deleted, RowsAffected has a value of zero. RowsAffected will have a value of –1 if the execution of the SQL (see page 1105) statement could not be executed due to an error condition. This latter situation would typically follow the raising of an exception.

See Also

ExecSQL (see page 1100)

8.1.6.5.8 TADOQuery.SQL Property

Contains the text of the SQL statement to execute for the ADO query.

Pascal

```
property SQL: TWideStrings;
```

C++

```
__property TWideStrings SQL;
```

Description

Use SQL to provide the SQL statement that an ADO query component executes when its ExecSQL (see page 1100) or Open method is called.

The SQL statement provided to the SQL property may contain replaceable parameters, following standard ADO syntax conventions. Parameters (see page 1104) are created and stored in the Parameters (see page 1104) property.

At design-time, edit the SQL statement using the property editor invoked by clicking the ellipsis button of the property in the Object Inspector. At runtime, use properties and methods of string list objects to clear the current contents, add new contents, or

to modify existing contents.

```
with ADOQuery1 do begin
  with SQL do begin
    Clear;
    Add('SELECT EmpNo, LastName, FirstName, HireDate');
    Add('FROM Employee');
  end;
  Open;
end;
ADOQuery1->Close();
ADOQuery1->SQL->Clear();
ADOQuery1->SQL->Add("SELECT EmpNo, LastName, FirstName, HireDate");
ADOQuery1->SQL->Add("FROM Employee");
ADOQuery1->Open();
```

Note: The library does not evaluate the SQL sent to the database via a TADOQuery (see page 1084) component. The SQL used must be valid for the particular database type accessed via ADO. Any error messages passed back to the application will have originated at the ADO or database level, and so will have error codes and messages specific to those systems.

See Also

Parameters

Prepared

ExecSQL (see page 1100)

ParamCheck

Open (see page 6976)

Active (see page 7004)

8.1.7 TADOStoredProc Class

TADOStoredProc encapsulates a stored procedure in an ADO database.

Class Hierarchy

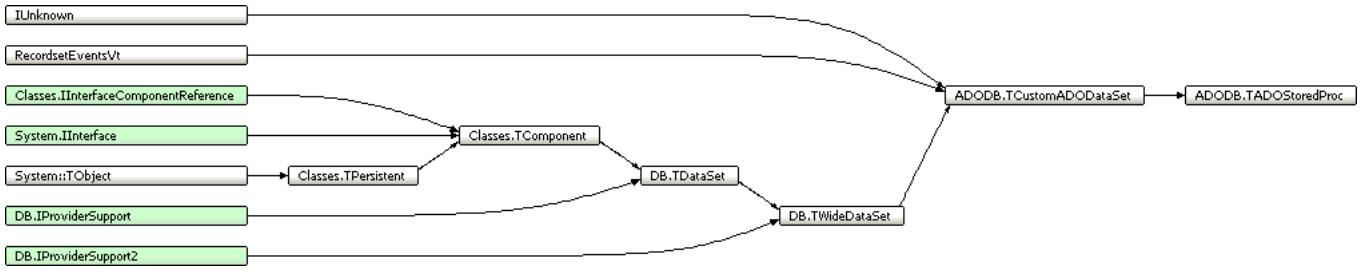

Pascal

```
TADOStoredProc = class(TCustomADODataSet);
```

C++

```
class TADOStoredProc : public TCustomADODataSet;
```

File

ADODB

Description

Use a TADOStoredProc object when a client application must use a stored procedure in a database. A stored procedure is a grouped set of statements, stored as part of a database's metadata (just like tables, indexes, and domains), that performs a frequently repeated, database-related task on the server and passes results to the client.

Note: Not all database servers support stored procedures. See a specific server's documentation to determine if it supports stored procedures.

Many stored procedures require a series of input arguments, or parameters, that are used during processing. TADOStoredProc provides a Parameters (see page 1123) property that enables an application to set these parameters before executing the stored procedure.

TADOStoredProc reuses the Parameters (see page 1123) property to hold the results returned by a stored procedure. Parameters (see page 1123) is a collection of TParameter (see page 1250) objects, each representing an individual parameter for the stored procedure. Depending on server implementation, a stored procedure can return either a single set of values, or a result set similar to the result set returned by a query.

See Also

TDataSet (see page 6894)

TADODataSet (see page 1041)

TADOConnection (see page 1001)

TADOTable (see page 1125)

TADOQuery (see page 1084)

Types of Datasets

Overview of ADO Components

8.1.7.1 TADOStoredProc Members

The following tables list the members exposed by TADOStoredProc.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	ActivateTextFilter (see page 1160)	This is ActivateTextFilter, a member of class TCustomADODataSet.
V	AllocRecordBuffer (see page 1160)	This is AllocRecordBuffer, a member of class TCustomADODataSet.
V	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CheckActive (see page 1162)	This is CheckActive, a member of class TCustomADODataSet.

	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
	CheckFieldCompatibility (see page 1162)	This is CheckFieldCompatibility, a member of class TCustomADODataSet.
	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
	ClearCalcFields (see page 1162)	This is ClearCalcFields, a member of class TCustomADODataSet.
	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
	ConnectionPoint (see page 1163)	This is ConnectionPoint, a member of class TCustomADODataSet.
	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
	DataEvent (see page 1164)	This is DataEvent, a member of class TCustomADODataSet.
	DataEvent (see page 6924)	This is DataEvent, a member of class TDataSet.
	DeactivateFilters (see page 1164)	This is DeactivateFilters, a member of class TCustomADODataSet.
	DefChanged (see page 1165)	This is DefChanged, a member of class TCustomADODataSet.
	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
	DefineProperties (see page 1165)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
	DestroyLookupCursor (see page 1166)	This is DestroyLookupCursor, a member of class TCustomADODataSet.
	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 1166)	This is DoOnNewRecord, a member of class TCustomADODataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	EnableEvents (see page 1166)	This is EnableEvents, a member of class TCustomADODataSet.
	EndOfRecordset (see page 1167)	This is EndOfRecordset, a member of class TCustomADODataSet.
	FetchComplete (see page 1167)	This is FetchComplete, a member of class TCustomADODataSet.
	FetchProgress (see page 1167)	This is FetchProgress, a member of class TCustomADODataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FieldChangeComplete (see page 1167)	This is FieldChangeComplete, a member of class TCustomADODataSet.
	FindRecord (see page 1168)	This is FindRecord, a member of class TCustomADODataSet.

	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeRecordBuffer (see page 1168)	This is FreeRecordBuffer, a member of class TCustomADODataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
	GetActiveRecBuf (see page 1168)	This is GetActiveRecBuf, a member of class TCustomADODataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
	GetBookmarkData (see page 1169)	This is GetBookmarkData, a member of class TCustomADODataSet.
	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
	GetBookmarkFlag (see page 1169)	This is GetBookmarkFlag, a member of class TCustomADODataSet.
	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 1169)	This is GetCanModify, a member of class TCustomADODataSet.
	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
	GetCommandText (see page 1170)	This is GetCommandText, a member of class TCustomADODataSet.
	GetDataSource (see page 1170)	This is GetDataSource, a member of class TCustomADODataSet.
	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
	GetRecNo (see page 1171)	This is GetRecNo, a member of class TCustomADODataSet.
	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
	GetRecord (see page 1172)	This is GetRecord, a member of class TCustomADODataSet.
	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
	GetRecordCount (see page 1172)	This is GetRecordCount, a member of class TCustomADODataSet.
	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
	GetRecordSize (see page 1172)	This is GetRecordSize, a member of class TCustomADODataSet.
	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
	GetStateFieldValue (see page 1172)	This is GetStateFieldValue, a member of class TCustomADODataSet.
	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
	InitOleDBAccess (see page 1173)	This is InitOleDBAccess, a member of class TCustomADODataSet.
	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
	InternalAddRecord (see page 1173)	This is InternalAddRecord, a member of class TCustomADODataSet.
	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
	InternalCancel (see page 1173)	This is InternalCancel, a member of class TCustomADODataSet.

	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
	InternalClose (see page 1173)	This is InternalClose, a member of class TCustomADODataSet.
	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.
	InternalDelete (see page 1173)	This is InternalDelete, a member of class TCustomADODataSet.
	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
	InternalEdit (see page 1174)	This is InternalEdit, a member of class TCustomADODataSet.
	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
	InternalFirst (see page 1174)	This is InternalFirst, a member of class TCustomADODataSet.
	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
	InternalGetRecord (see page 1174)	This is InternalGetRecord, a member of class TCustomADODataSet.
	InternalGotoBookmark (see page 1174)	This is InternalGotoBookmark, a member of class TCustomADODataSet.
	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
	InternalHandleException (see page 1175)	This is InternalHandleException, a member of class TCustomADODataSet.
	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
	InternalInitFieldDefs (see page 1175)	This is InternalInitFieldDefs, a member of class TCustomADODataSet.
	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
	InternalInitRecord (see page 1175)	This is InternalInitRecord, a member of class TCustomADODataSet.
	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
	InternalInsert (see page 1175)	This is InternalInsert, a member of class TCustomADODataSet.
	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
	InternalLast (see page 1175)	This is InternalLast, a member of class TCustomADODataSet.
	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
	InternalOpen (see page 1176)	This is InternalOpen, a member of class TCustomADODataSet.
	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
	InternalPost (see page 1176)	This is InternalPost, a member of class TCustomADODataSet.
	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
	InternalRefresh (see page 1176)	This is InternalRefresh, a member of class TCustomADODataSet.
	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
	InternalRequery (see page 1176)	This is InternalRequery, a member of class TCustomADODataSet.
	InternalSetSort (see page 1177)	This is InternalSetSort, a member of class TCustomADODataSet.
	InternalSetToRecord (see page 1177)	This is InternalSetToRecord, a member of class TCustomADODataSet.
	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsCursorOpen (see page 1177)	This is IsCursorOpen, a member of class TCustomADODataSet.
	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
	Loaded (see page 1177)	Initializes the component after the form file has been read into memory.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1179)	This is LocateRecord, a member of class TCustomADODataSet.
	MasterChanged (see page 1180)	This is MasterChanged, a member of class TCustomADODataSet.
	MasterDisabled (see page 1180)	This is MasterDisabled, a member of class TCustomADODataSet.
	MoveComplete (see page 1180)	This is MoveComplete, a member of class TCustomADODataSet.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	OpenCursor (see page 1181)	This is OpenCursor, a member of class TCustomADODataSet.
	OpenCursor (see page 6976)	This is OpenCursor, a member of class TDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PrepareCursor (see page 1181)	This is PrepareCursor, a member of class TCustomADODataSet.
	PSEndTransaction (see page 1181)	Ends a transaction that was started by a call to PSStartTransaction.
	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 1182)	Executes the SQL command associated with the dataset.
	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 1182)	Executes a specified SQL command.

	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSExecuteStatement (see page 7535)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 1183)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 1183)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 6982)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 7537)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandTextW (see page 7537)	This is PSGetCommandTextW, a member of class TWideDataSet.
	PSGetCommandType (see page 1184)	Gets the command type to be executed by PSExecute.
	PSGetCommandType (see page 6982)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 1184)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 1185)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7538)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 1185)	This is PSGetKeyFieldsW, a member of class TCustomADODataSet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetKeyFieldsW (see page 7538)	This is PSGetKeyFieldsW, a member of class TWideDataSet.
	PSGetParams (see page 1185)	Returns the current parameter values of the dataset.
	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7538)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 1186)	This is PSGetQuoteCharW, a member of class TCustomADODataSet.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetQuoteCharW (see page 7539)	This is PSGetQuoteCharW, a member of class TWideDataSet.
	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7539)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 1186)	This is PSGetTableNameW, a member of class TCustomADODataSet.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetTableNameW (see page 7539)	This is PSGetTableNameW, a member of class TWideDataSet.
	PSGetUpdateException (see page 1186)	Generates an EUpdateError object based on another exception object.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 1187)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 1187)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 1188)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 1188)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 1188)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetCommandText (see page 7540)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 1189)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 1189)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.

	PSUpdateRecord (see page 1190)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RecordChangeComplete (see page 1190)	This is RecordChangeComplete, a member of class TCustomADODataSet.
	RecordsetChangeComplete (see page 1190)	This is RecordsetChangeComplete, a member of class TCustomADODataSet.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	RefreshParams (see page 1191)	This is RefreshParams, a member of class TCustomADODataSet.
	ReleaseLock (see page 1191)	This is ReleaseLock, a member of class TCustomADODataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBlockReadSize (see page 1194)	This is SetBlockReadSize, a member of class TCustomADODataSet.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBookmarkData (see page 1194)	This is SetBookmarkData, a member of class TCustomADODataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkFlag (see page 1194)	This is SetBookmarkFlag, a member of class TCustomADODataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCommandText (see page 1194)	This is SetCommandText, a member of class TCustomADODataSet.
	SetConnection (see page 1194)	This is SetConnection, a member of class TCustomADODataSet.
	SetConnectionFlag (see page 1195)	This is SetConnectionFlag, a member of class TCustomADODataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDataSource (see page 1195)	This is SetDataSource, a member of class TCustomADODataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDetailFilter (see page 1195)	This is SetDetailFilter, a member of class TCustomADODataSet.
	SetFieldData (see page 1195)	This is the overview for the SetFieldData method overload.
	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
	SetFiltered (see page 1196)	This is SetFiltered, a member of class TCustomADODataSet.
	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
	SetFilterOptions (see page 1196)	This is SetFilterOptions, a member of class TCustomADODataSet.
	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
	SetFilterText (see page 1196)	This is SetFilterText, a member of class TCustomADODataSet.
	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetParamsFromCursor (see page 1197)	This is SetParamsFromCursor, a member of class TCustomADODataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetRecNo (see page 1197)	This is SetRecNo, a member of class TCustomADODataSet.
	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.

	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateIndexDefs (see page 1198)	This is UpdateIndexDefs, a member of class TCustomADODataSet.
V	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.
	UpdateRecordSetPosition (see page 1198)	This is UpdateRecordSetPosition, a member of class TCustomADODataSet.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WillChangeField (see page 1199)	This is WillChangeField, a member of class TCustomADODataSet.
	WillChangeRecord (see page 1199)	This is WillChangeRecord, a member of class TCustomADODataSet.
	WillChangeRecordset (see page 1200)	This is WillChangeRecordset, a member of class TCustomADODataSet.
	WillMove (see page 1200)	This is WillMove, a member of class TCustomADODataSet.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
R	Command (see page 1204)	Provides access to the internal command object that executes commands for the dataset.
	CommandText (see page 1204)	This is CommandText, a member of class TCustomADODataSet.
	CommandType (see page 1205)	Specifies the type of command to execute.
	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
	FieldDefs (see page 1210)	Points to the list of field definitions for the dataset.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
	IndexDefs (see page 1215)	Contains the definitions of the indexes for the current table.
	IndexFieldNames (see page 1216)	Lists the columns to use for a persistent sort of the recordset.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
R	MasterDataLink (see page 1218)	Manages the relationship between the ADO dataset and a master table.
	MasterFields (see page 1219)	Specifies field or fields on which a Master-Detail link is based.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
	ParamCheck (see page 1220)	Specifies whether the parameter list for an ADO dataset component is regenerated if the SQL changes at runtime.
	Reserved (see page 7035)	For internal use only.
	StoreDefs (see page 1223)	Specifies whether field definitions are stored with the ADO dataset.

Public Events

	Name	Description
	AfterCancel (see page 7038)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 7040)	Occurs after an application closes a dataset.
	AfterDelete (see page 7041)	Occurs after an application deletes a record.
	AfterEdit (see page 7044)	Occurs after an application starts editing a record.
	AfterInsert (see page 7044)	Occurs after an application inserts a new record.
	AfterOpen (see page 7045)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 7046)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 7047)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 7047)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 7048)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 7048)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 7049)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 7049)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 7050)	Occurs before an application enters insert mode.
	BeforeOpen (see page 7051)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 7052)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 7052)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 7053)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 7053)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 7054)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 7054)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 7056)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 7056)	Occurs when an application attempts to modify or insert a record and an exception is raised.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 1160)	Tests the validity of a specified bookmark.
V	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
V	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
	CancelBatch (see page 1161)	Cancels a pending batch update.
	CancelUpdates (see page 1162)	Cancels pending cached updates.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.

	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Clone (see page 1163)	Duplicates another dataset in the calling component
	Close (see page 6920)	Closes a dataset.
V	CompareBookmarks (see page 1163)	Compares two bookmarks.
V	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
V	Create (see page 1120)	Creates an instance of a TADOStoredProc (see page 1106) component.
V	CreateBlobStream (see page 1164)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
V	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
	DeleteRecords (see page 1165)	Deletes one or more records in the dataset.
V	Destroy (see page 1157)	Destroys the TCustomADONETConnector instance.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 6904)	Destroys the TDataSet (see page 6894) instance.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	ExecProc (see page 1121)	Executes the stored procedure on the server.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FilterOnBookmarks (see page 1168)	Filters a data set to just rows with pre-defined bookmarks.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetBlobFieldData (see page 1169)	Retrieves the current value of a field into a buffer.
V	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.

	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
	GetDetailLinkFields (see page 1170)	Lists the field components that link this dataset as a detail of a master dataset.
	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldData (see page 1171)	This is the overview for the GetFieldData method overload.
	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
	IsSequenced (see page 1177)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	Last (see page 6968)	Moves to the last record in the dataset.
	LoadFromFile (see page 1178)	Loads a recordset from a file.
	Locate (see page 1179)	Searches the dataset for a specified record and makes that record the current record.
	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Lookup (see page 1180)	Retrieves field values from a row that matches specified search values.
	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	NextRecordset (see page 1181)	Provides access to a secondary recordset.
	Open (see page 6976)	Opens the dataset.
	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
	Prior (see page 6979)	Moves to the previous record in the dataset.

	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSEndTransaction (see page 7589)	Ends a transaction that was started by a call to PSStartTransaction (see page 7597).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecute (see page 7590)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSExecuteStatement (see page 7590)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 7591)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 7579)	Gets the SQL command to be executed by PSExecute (see page 7578).
	PSGetCommandText (see page 7591)	Gets the SQL command to be executed by PSExecute (see page 7590).
	PSGetCommandType (see page 7580)	Gets the command type to be executed by PSExecute (see page 7578).
	PSGetCommandType (see page 7591)	Gets the command type to be executed by PSExecute (see page 7590).
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 7592)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 7592)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7593)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetParams (see page 7593)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7594)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7594)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSGetUpdateException (see page 7594)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 7595)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 7595)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 7596)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 7596)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7585)	Assigns a specified SQL command to be executed by PSExecute (see page 7578).
	PSSetCommandText (see page 7596)	Assigns a specified SQL command to be executed by PSExecute (see page 7590).
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 7597)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).
	PSStartTransaction (see page 7597)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7589).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 7598)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Requery (see page 1191)	Refreshes the recordset.
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 1192)	Saves a recordset to a file.
	Seek (see page 1192)	Searches for a record using the current index.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	Supports (see page 1197)	Indicates whether a dataset supports certain types of operations.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateBatch (see page 1197)	Writes pending batch updates to disk.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 1199)	Indicates the update status of the current record.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.

Public Properties

	Name	Description
	Active (see page 7004)	Specifies whether or not a dataset is open.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignerData (see page 1208)	Stores custom data.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
	FieldDefs (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset

	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.
	FilterGroup (see page 1214)	Filters a recordset based on row update status.
	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
R	IndexFieldCount (see page 1216)	Indicates the number of fields that comprise the current key.
	IndexFields (see page 1217)	Retrieves or sets a field for an index.
	IndexName (see page 1217)	Specifies the currently active index.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	Modified (see page 7029)	Indicates whether the active record is modified.
	ObjectView (see page 7031)	Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Properties (see page 1221)	This is Properties, a member of class TCustomADODataSet.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
	Recordset (see page 1221)	This is Recordset, a member of class TCustomADODataSet.
R	RecordsetState (see page 1222)	This is RecordsetState, a member of class TCustomADODataSet.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
R	RecordStatus (see page 1222)	Indicates the status of the current row.
	Sort (see page 1222)	Specifies the sort order of the recordset.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterCancel (see page 1224)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 1226)	Occurs after an application closes a dataset.
	AfterDelete (see page 1227)	Occurs after an application deletes a record.
	AfterEdit (see page 1230)	Occurs after an application starts editing a record.
	AfterInsert (see page 1231)	Occurs after an application inserts a new record.
	AfterOpen (see page 1231)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 1233)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 1233)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 1233)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 1234)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 1234)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 1235)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 1235)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 1236)	Occurs before an application enters insert mode.
	BeforeOpen (see page 1237)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 1238)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 1239)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 1239)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 1239)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 1240)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 1241)	Occurs when an application attempts to modify or insert a record and an exception is raised.

	OnEndOfRecordset (see page 1241)	Occurs when moving a recordset's row pointer past the last row.
	OnFetchComplete (see page 1242)	Occurs after records have been retrieved into the recordset.
	OnFetchProgress (see page 1242)	Occurs periodically during an asynchronous data retrieval operation.
	OnFieldChangeComplete (see page 1243)	Occurs after a field has changed.
	OnFilterRecord (see page 1244)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnMoveComplete (see page 1244)	Occurs after the row pointer in a recordset has moved.
	OnNewRecord (see page 1245)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 1245)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnRecordChangeComplete (see page 1246)	Occurs after one or more records in the recordset change.
	OnRecordsetChangeComplete (see page 1247)	Occurs after the recordset has changed.
	OnRecordsetCreate (see page 1247)	Occurs when the recordset has been initialized and is ready to use.
	OnWillChangeField (see page 1248)	Occurs before a field is changed.
	OnWillChangeRecord (see page 1248)	Occurs before a record in the recordset is changed.
	OnWillChangeRecordset (see page 1249)	Occurs before the recordset has changed.
	OnWillMove (see page 1250)	Occurs before the row pointer in a recordset is moved.

Published Properties

	Name	Description
	Active (see page 1201)	Specifies whether or not a dataset is open.
	AutoCalcFields (see page 1203)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	CacheSize (see page 1203)	Specifies the size of the cache for the dataset.
	CommandTimeout (see page 1122)	Specifies amount of time to attempt execution of a command.
	Connection (see page 1205)	Specifies the ADO connection component to use.
	ConnectionString (see page 1206)	Specifies the connection information for the data store.
	CursorLocation (see page 1207)	Specifies whether the cursor for the connection is client-side or server-side.
	CursorType (see page 1208)	Specifies type of cursor an ADO dataset uses.
	DataSource (see page 1122)	Represents the data source of another dataset that supplies values to the dataset.
	EnableBCD (see page 1123)	Specifies whether to treat numeric fields as floating-point or BCD.
	ExecuteOptions (see page 1209)	Specifies the characteristics of a command execution.
	Filter (see page 1212)	Specifies the text of the current filter for a recordset.
	Filtered (see page 1214)	Specifies whether or not filtering is active for a dataset.
	LockType (see page 1218)	Specifies the lock type to use when opening a dataset.
	MarshalOptions (see page 1218)	Specifies which records are marshaled back to the server.
	MaxRecords (see page 1219)	Specifies how many rows to return in a result set.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Parameters (see page 1123)	Contains the collection of parameters for an SQL statement.
	Prepared (see page 1124)	Specifies whether the command is prepared before execution.
	ProcedureName (see page 1124)	Indicates the stored procedure used by the TADOStoredProc (see page 1106).
	Tag (see page 2808)	Stores an integer value as part of a component.

8.1.7.2 TADOStoredProc.Create Constructor

Creates an instance of a TADOStoredProc (see page 1106) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TADOStoredProc(TComponent * AOwner);
```

Description

Call Create to instantiate an ADO stored procedure component at runtime. Stored procedure components placed in forms or data modules at design time are created automatically.

Create

Calls its inherited Create method.

Sets the CommandType (see page 1205) property to cmdStoredProc.

See Also

TADODataSet (see page 1041)

TADOQuery (see page 1084)

TADOTable (see page 1125)

8.1.7.3 TADOStoredProc Methods

The methods of the TADOStoredProc class are listed here.

Public Methods

	Name	Description
	ExecProc (see page 1121)	Executes the stored procedure on the server.

8.1.7.3.1 TADOStoredProc.ExecProc Method

Executes the stored procedure on the server.

Pascal

```
procedure ExecProc;
```

C++

```
__fastcall ExecProc();
```

Description

Call ExecProc to execute a stored procedure. Before calling ExecProc:

1. Provide any input parameters in the Parameters (see page 1123) property. At design-time, a developer can provide parameters using the Parameters (see page 1123) Editor. At runtime an application must access the Parameters (see page 1123) property directly.
2. Set the Prepared (see page 1124) property to true to bind the parameters.

If a stored procedure returns output parameters, they are stored in the Parameters (see page 1123) property when ExecProc returns control to the application. An application can access the output parameters using the Parameters (see page 1123) property and identifying the particular parameter by its number or by using the ParamByName method and identifying the parameter by its name.

Tip: If an application is only interested in the result set returned by a stored procedure, call Open method for the TADOStoredProc (see page 1106) component or set its Active (see page 1201) property to true.

See Also

Active (see page 7004)

Open (see page 6976)

Parameters

Prepared

Connection

8.1.7.4 TADOStoredProc Properties

The properties of the TADOStoredProc class are listed here.

Published Properties

	Name	Description
	CommandTimeout (see page 1122)	Specifies amount of time to attempt execution of a command.
	DataSource (see page 1122)	Represents the data source of another dataset that supplies values to the dataset.
	EnableBCD (see page 1123)	Specifies whether to treat numeric fields as floating-point or BCD.
	Parameters (see page 1123)	Contains the collection of parameters for an SQL statement.
	Prepared (see page 1124)	Specifies whether the command is prepared before execution.
	ProcedureName (see page 1124)	Indicates the stored procedure used by the TADOStoredProc (see page 1106).

8.1.7.4.1 TADOStoredProc.CommandTimeout Property

Specifies amount of time to attempt execution of a command.

Pascal

```
property CommandTimeout: Integer;
```

C++

```
__property int CommandTimeout;
```

Description

Use CommandTimeout to specify the amount of time that expires before an attempt to execute a command is considered unsuccessful. Measured in seconds, the default value for CommandTimeout is 30.

If a command is successfully executed prior to the expiration of the seconds specified, CommandTimeout has no effect.

See Also

commandtext (see page 1204)

CommandType (see page 1205)

8.1.7.4.2 TADOStoredProc.DataSource Property

Represents the data source of another dataset that supplies values to the dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;__property TDataSource DataSource;
```

Description

As implemented in TDataSet, the DataSource property is always nil (Delphi) or NULL (C++).

Descendant dataset classes that support specifying a data source redeclare and implement methods for getting and setting the

DataSource property.

See Also

Active (see page 7004)

IsLinkedTo (see page 6967)

Establishing Master/detail Relationships Using Parameters

8.1.7.4.3 TADOStoredProc.EnableBCD Property

Specifies whether to treat numeric fields as floating-point or BCD.

Pascal

```
property EnableBCD: Boolean;
```

C++

```
__property Boolean EnableBCD;
```

Description

Use the EnableBCD property to specify how fields are mapped to field classes. If EnableBCD is true, (the default value) adDecimal and adNumeric fields are mapped to the TBCDField class when field objects are created. If EnableBCD is false, the fields are mapped to the TFloatField class.

EnableBCD determines whether numeric and decimal fields are translated as floating-point values or binary coded decimal (BCD) values. BCD values eliminate the rounding errors associated with floating point math (such as a 3 * (2/3) resulting in 2.00000000001).

If persistent field objects are used (such as those created using the Fields Editor), the field class associated with a given field can contradict the setting of EnableBCD. Three numeric fields in the same table could, for instance, each be represented by a TFloatField, a TBCDField, and a TVariantField – regardless of the value in EnableBCD. Select field classes on a field-by-field basis for greater flexibility.

Note: For numeric values with more than 4 digits to the right of the decimal place, use of TFloatField is generally better. This is because TBCDField uses the currency data type that has a fixed scale of 4 decimal places.

Note: For fields with very large numbers of more than 19 significant digits, you can use TVariantField type persistent field objects. The TFloatField and TBCDField classes lack sufficient capacity for fields of this size. TVariantField allows the getting and setting of the field data as strings, preventing without loss of data due to insufficient capacity. However, arithmetic operations cannot be performed on numbers accessed through a TVariantField object.

See Also

TBCDField (see page 6776)

TFloatField (see page 7229)

8.1.7.4.4 TADOStoredProc.Parameters Property

Contains the collection of parameters for an SQL statement.

Pascal

```
property Parameters: TParameters;
```

C++

```
__property TParameters Parameters;
```

Description

At design-time, use Parameters to access the parameters for the SQL statement used by the ADO component. Use the properties and methods of Parameters to set and get individual parameter values or to set or inspect parameter attributes.

At runtime, use the Parameters property editor to access the collection of parameters.

See Also

SQL (see page 1105)

ExecSQL (see page 1100)

8.1.7.4.5 TADOStoredProc.Prepared Property

Specifies whether the command is prepared before execution.

Pascal

```
property Prepared: Boolean;
```

C++

```
__property Boolean Prepared;
```

Description

Set Prepared before calling the Open method to specify whether ADO prepares the command used to create the dataset's recordset. If Prepared is set to true and the dataset component is connected to a data store, ADO prepares the command before executing it. If Prepared is set to false, ADO does not prepare the command.

The default value of Prepared is false.

See Also

Parameters (see page 1220)

ExecSQL (see page 1100)

Open (see page 6976)

8.1.7.4.6 TADOStoredProc.ProcedureName Property

Indicates the stored procedure used by the TADOStoredProc (see page 1106).

Pascal

```
property ProcedureName: WideString;
```

C++

```
__property BSTR ProcedureName;
```

Description

Set ProcedureName to specify the stored procedure in an ADO database with which the TADOStoredProc (see page 1106) is associated.

At design-time, the property editor displays a list of available stored procedures. This list is only possible when a valid connection can be established between the stored procedure component and a database.

At runtime, set the ProcedureName property using a WideString (or comparable) value. The value used must be a valid name for a stored procedure in the associated database.

See Also

GetProcedureNames (see page 1014)

Connection (see page 1205)

8.1.8 TADOTable Class

TADOTable is a dataset component that encapsulates a table accessed through an ADO data store.

Class Hierarchy

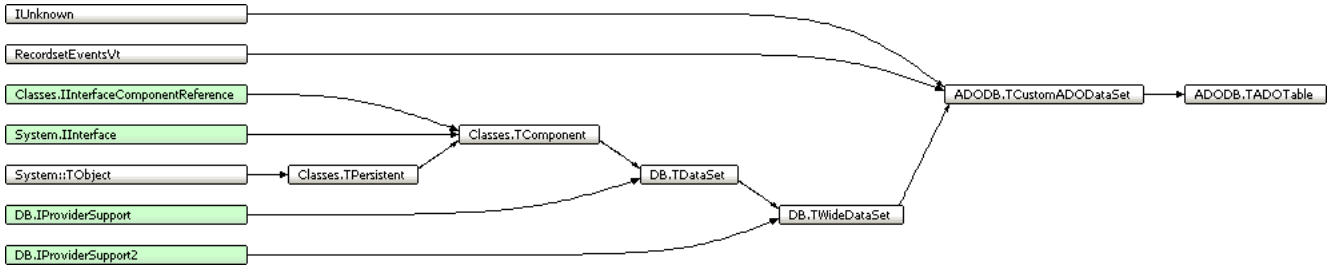

Pascal

```
TADOTable = class(TCustomADODataSet);
```

C++

```
class TADOTable : public TCustomADODataSet;
```

File

ADODB

Description

Use TADOTable to access data in a single database table using ADO. TADOTable provides direct access to every record and field in an underlying database table. An ADO table component can also work with a subset of records within a database table using ranges and filters.

See Also

TDataSet (see page 6894)

TADODataSet (see page 1041)

TADOConnection (see page 1001)

TADOQuery (see page 1084)

TADOStoredProc (see page 1106)

Types of Datasets

Overview of ADO Components

8.1.8.1 TADOTable Members

The following tables list the members exposed by TADOTable.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	ActivateTextFilter (see page 1160)	This is ActivateTextFilter, a member of class TCustomADODataSet.
V	AllocRecordBuffer (see page 1160)	This is AllocRecordBuffer, a member of class TCustomADODataSet.
V	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CheckActive (see page 1162)	This is CheckActive, a member of class TCustomADODataSet.
V	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
V	CheckFieldCompatibility (see page 1162)	This is CheckFieldCompatibility, a member of class TCustomADODataSet.
V	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
V	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
V	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
V	ClearCalcFields (see page 1162)	This is ClearCalcFields, a member of class TCustomADODataSet.
V	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
V	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
V	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
	ConnectionPoint (see page 1163)	This is ConnectionPoint, a member of class TCustomADODataSet.
V	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
V	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
V	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
V	DataEvent (see page 1164)	This is DataEvent, a member of class TCustomADODataSet.
V	DataEvent (see page 6924)	This is DataEvent, a member of class TDataSet.
	DeactivateFilters (see page 1164)	This is DeactivateFilters, a member of class TCustomADODataSet.
V	DefChanged (see page 1165)	This is DefChanged, a member of class TCustomADODataSet.
V	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
V	DefineProperties (see page 1165)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
V	DestroyLookupCursor (see page 1166)	This is DestroyLookupCursor, a member of class TCustomADODataSet.
V	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
V	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
V	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
V	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
V	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
V	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
V	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
V	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.

	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 1166)	This is DoOnNewRecord, a member of class TCustomADODataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	EnableEvents (see page 1166)	This is EnableEvents, a member of class TCustomADODataSet.
	EndOfRecordset (see page 1167)	This is EndOfRecordset, a member of class TCustomADODataSet.
	FetchComplete (see page 1167)	This is FetchComplete, a member of class TCustomADODataSet.
	FetchProgress (see page 1167)	This is FetchProgress, a member of class TCustomADODataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FieldChangeComplete (see page 1167)	This is FieldChangeComplete, a member of class TCustomADODataSet.
	FindRecord (see page 1168)	This is FindRecord, a member of class TCustomADODataSet.
	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeRecordBuffer (see page 1168)	This is FreeRecordBuffer, a member of class TCustomADODataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
	GetActiveRecBuf (see page 1168)	This is GetActiveRecBuf, a member of class TCustomADODataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
	GetBookmarkData (see page 1169)	This is GetBookmarkData, a member of class TCustomADODataSet.
	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
	GetBookmarkFlag (see page 1169)	This is GetBookmarkFlag, a member of class TCustomADODataSet.
	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 1169)	This is GetCanModify, a member of class TCustomADODataSet.
	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
	GetCommandText (see page 1170)	This is GetCommandText, a member of class TCustomADODataSet.
	GetDataSource (see page 1170)	This is GetDataSource, a member of class TCustomADODataSet.
	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.

	GetRecNo (see page 1171)	This is GetRecNo, a member of class TCustomADODataSet.
	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
	GetRecord (see page 1172)	This is GetRecord, a member of class TCustomADODataSet.
	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
	GetRecordCount (see page 1172)	This is GetRecordCount, a member of class TCustomADODataSet.
	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
	GetRecordSize (see page 1172)	This is GetRecordSize, a member of class TCustomADODataSet.
	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
	GetStateFieldValue (see page 1172)	This is GetStateFieldValue, a member of class TCustomADODataSet.
	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
	InitOleDBAccess (see page 1173)	This is InitOleDBAccess, a member of class TCustomADODataSet.
	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
	InternalAddRecord (see page 1173)	This is InternalAddRecord, a member of class TCustomADODataSet.
	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
	InternalCancel (see page 1173)	This is InternalCancel, a member of class TCustomADODataSet.
	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
	InternalClose (see page 1173)	This is InternalClose, a member of class TCustomADODataSet.
	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.
	InternalDelete (see page 1173)	This is InternalDelete, a member of class TCustomADODataSet.
	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
	InternalEdit (see page 1174)	This is InternalEdit, a member of class TCustomADODataSet.
	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
	InternalFirst (see page 1174)	This is InternalFirst, a member of class TCustomADODataSet.
	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
	InternalGetRecord (see page 1174)	This is InternalGetRecord, a member of class TCustomADODataSet.
	InternalGotoBookmark (see page 1174)	This is InternalGotoBookmark, a member of class TCustomADODataSet.
	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
	InternalHandleException (see page 1175)	This is InternalHandleException, a member of class TCustomADODataSet.
	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
	InternalInitFieldDefs (see page 1175)	This is InternalInitFieldDefs, a member of class TCustomADODataSet.
	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
	InternalInitRecord (see page 1175)	This is InternalInitRecord, a member of class TCustomADODataSet.
	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
	InternalInsert (see page 1175)	This is InternalInsert, a member of class TCustomADODataSet.
	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
	InternalLast (see page 1175)	This is InternalLast, a member of class TCustomADODataSet.
	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
	InternalOpen (see page 1176)	This is InternalOpen, a member of class TCustomADODataSet.
	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
	InternalPost (see page 1176)	This is InternalPost, a member of class TCustomADODataSet.
	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
	InternalRefresh (see page 1176)	This is InternalRefresh, a member of class TCustomADODataSet.
	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
	InternalRequery (see page 1176)	This is InternalRequery, a member of class TCustomADODataSet.
	InternalSetSort (see page 1177)	This is InternalSetSort, a member of class TCustomADODataSet.
	InternalSetToRecord (see page 1177)	This is InternalSetToRecord, a member of class TCustomADODataSet.
	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsCursorOpen (see page 1177)	This is IsCursorOpen, a member of class TCustomADODataSet.

	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
	Loaded (see page 1177)	Initializes the component after the form file has been read into memory.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1179)	This is LocateRecord, a member of class TCustomADODataSet.
	MasterChanged (see page 1180)	This is MasterChanged, a member of class TCustomADODataSet.
	MasterDisabled (see page 1180)	This is MasterDisabled, a member of class TCustomADODataSet.
	MoveComplete (see page 1180)	This is MoveComplete, a member of class TCustomADODataSet.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	OpenCursor (see page 1181)	This is OpenCursor, a member of class TCustomADODataSet.
	OpenCursor (see page 6976)	This is OpenCursor, a member of class TDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PrepareCursor (see page 1181)	This is PrepareCursor, a member of class TCustomADODataSet.
	PSEndTransaction (see page 1181)	Ends a transaction that was started by a call to PSStartTransaction.
	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 1182)	Executes the SQL command associated with the dataset.
	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 1182)	Executes a specified SQL command.
	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSExecuteStatement (see page 7535)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 1183)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 1183)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 6982)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 7537)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandTextW (see page 7537)	This is PSGetCommandTextW, a member of class TWideDataSet.
	PSGetCommandType (see page 1184)	Gets the command type to be executed by PSExecute.
	PSGetCommandType (see page 6982)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 1184)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 1185)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7538)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 1185)	This is PSGetKeyFieldsW, a member of class TCustomADODataSet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetKeyFieldsW (see page 7538)	This is PSGetKeyFieldsW, a member of class TWideDataSet.
	PSGetParams (see page 1185)	Returns the current parameter values of the dataset.
	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7538)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 1186)	This is PSGetQuoteCharW, a member of class TCustomADODataSet.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetQuoteCharW (see page 7539)	This is PSGetQuoteCharW, a member of class TWideDataSet.
	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7539)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 1186)	This is PSGetTableNameW, a member of class TCustomADODataSet.

	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetTableNameW (see page 7539)	This is PSGetTableNameW, a member of class TWideDataSet.
	PSGetUpdateException (see page 1186)	Generates an EUpdateError object based on another exception object.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 1187)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 1187)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 1188)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 1188)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 1188)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetCommandText (see page 7540)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 1189)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 1189)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 1190)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RecordChangeComplete (see page 1190)	This is RecordChangeComplete, a member of class TCustomADODataSet.
	RecordsetChangeComplete (see page 1190)	This is RecordsetChangeComplete, a member of class TCustomADODataSet.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	RefreshParams (see page 1191)	This is RefreshParams, a member of class TCustomADODataSet.
	ReleaseLock (see page 1191)	This is ReleaseLock, a member of class TCustomADODataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBlockReadSize (see page 1194)	This is SetBlockReadSize, a member of class TCustomADODataSet.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBookmarkData (see page 1194)	This is SetBookmarkData, a member of class TCustomADODataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkFlag (see page 1194)	This is SetBookmarkFlag, a member of class TCustomADODataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCommandText (see page 1194)	This is SetCommandText, a member of class TCustomADODataSet.
	SetConnection (see page 1194)	This is SetConnection, a member of class TCustomADODataSet.
	SetConnectionFlag (see page 1195)	This is SetConnectionFlag, a member of class TCustomADODataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDataSource (see page 1195)	This is SetDataSource, a member of class TCustomADODataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDetailFilter (see page 1195)	This is SetDetailFilter, a member of class TCustomADODataSet.
	SetFieldData (see page 1195)	This is the overview for the SetFieldData method overload.

V	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
V	SetFiltered (see page 1196)	This is SetFiltered, a member of class TCustomADODataSet.
V	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
V	SetFilterOptions (see page 1196)	This is SetFilterOptions, a member of class TCustomADODataSet.
V	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
V	SetFilterText (see page 1196)	This is SetFilterText, a member of class TCustomADODataSet.
V	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
V	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetParamsFromCursor (see page 1197)	This is SetParamsFromCursor, a member of class TCustomADODataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetRecNo (see page 1197)	This is SetRecNo, a member of class TCustomADODataSet.
V	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
V	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateIndexDefs (see page 1198)	This is UpdateIndexDefs, a member of class TCustomADODataSet.
V	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.
	UpdateRecordSetPosition (see page 1198)	This is UpdateRecordSetPosition, a member of class TCustomADODataSet.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WillChangeField (see page 1199)	This is WillChangeField, a member of class TCustomADODataSet.
	WillChangeRecord (see page 1199)	This is WillChangeRecord, a member of class TCustomADODataSet.
	WillChangeRecordset (see page 1200)	This is WillChangeRecordset, a member of class TCustomADODataSet.
	WillMove (see page 1200)	This is WillMove, a member of class TCustomADODataSet.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.

R	Command (see page 1204)	Provides access to the internal command object that executes commands for the dataset.
	CommandText (see page 1204)	This is CommandText, a member of class TCustomADODataSet.
	CommandType (see page 1205)	Specifies the type of command to execute.
	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
	DataSource (see page 1208)	Represents the data source of another dataset that supplies values to the dataset.
	FieldDefs (see page 1210)	Points to the list of field definitions for the dataset.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
R	MasterDataLink (see page 1218)	Manages the relationship between the ADO dataset and a master table.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
	ParamCheck (see page 1220)	Specifies whether the parameter list for an ADO dataset component is regenerated if the SQL changes at runtime.
	Parameters (see page 1220)	Contains the collection of parameters for an SQL statement.
	Prepared (see page 1221)	Specifies whether the command is prepared before execution.
	Reserved (see page 7035)	For internal use only.
	StoreDefs (see page 1223)	Specifies whether field definitions are stored with the ADO dataset.

Public Events

	Name	Description
	AfterCancel (see page 7038)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 7040)	Occurs after an application closes a dataset.
	AfterDelete (see page 7041)	Occurs after an application deletes a record.
	AfterEdit (see page 7044)	Occurs after an application starts editing a record.
	AfterInsert (see page 7044)	Occurs after an application inserts a new record.
	AfterOpen (see page 7045)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 7046)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 7047)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 7047)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 7048)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 7048)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 7049)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 7049)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 7050)	Occurs before an application enters insert mode.
	BeforeOpen (see page 7051)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 7052)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 7052)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 7053)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 7053)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 7054)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 7054)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 7056)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 7056)	Occurs when an application attempts to modify or insert a record and an exception is raised.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 1160)	Tests the validity of a specified bookmark.
V	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
V	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
	CancelBatch (see page 1161)	Cancels a pending batch update.
	CancelUpdates (see page 1162)	Cancels pending cached updates.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Clone (see page 1163)	Duplicates another dataset in the calling component
	Close (see page 6920)	Closes a dataset.
V	CompareBookmarks (see page 1163)	Compares two bookmarks.
V	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
V	Create (see page 1139)	Creates an instance of a TADOTable (see page 1125) component.
V	CreateBlobStream (see page 1164)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
V	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
	DeleteRecords (see page 1165)	Deletes one or more records in the dataset.
V	Destroy (see page 1157)	Destroys the TCustomADONETConnector instance.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 6904)	Destroys the TDataSet (see page 6894) instance.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FilterOnBookmarks (see page 1168)	Filters a data set to just rows with pre-defined bookmarks.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetBlobFieldData (see page 1169)	Retrieves the current value of a field into a buffer.
V	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
V	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
V	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
V	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
V	GetDetailLinkFields (see page 1170)	Lists the field components that link this dataset as a detail of a master dataset.
V	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetFieldData (see page 1171)	This is the overview for the GetFieldData method overload.
V	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
V	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetIndexNames (see page 1140)	Retrieves a list of available indexes for a table.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.

	IsSequenced (see page 1177)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	Last (see page 6968)	Moves to the last record in the dataset.
	LoadFromFile (see page 1178)	Loads a recordset from a file.
	Locate (see page 1179)	Searches the dataset for a specified record and makes that record the current record.
	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Lookup (see page 1180)	Retrieves field values from a row that matches specified search values.
	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	NextRecordset (see page 1181)	Provides access to a secondary recordset.
	Open (see page 6976)	Opens the dataset.
	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSEndTransaction (see page 7589)	Ends a transaction that was started by a call to PSStartTransaction (see page 7597).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecute (see page 7590)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSExecuteStatement (see page 7590)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 7591)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 7579)	Gets the SQL command to be executed by PSExecute (see page 7578).
	PSGetCommandText (see page 7591)	Gets the SQL command to be executed by PSExecute (see page 7590).
	PSGetCommandType (see page 7580)	Gets the command type to be executed by PSExecute (see page 7578).
	PSGetCommandType (see page 7591)	Gets the command type to be executed by PSExecute (see page 7590).
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 7592)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 7592)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7593)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetParams (see page 7593)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7594)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7594)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.

	PSGetUpdateException (see page 7594)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 7595)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 7595)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 7596)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 7596)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7585)	Assigns a specified SQL command to be executed by PSExecute (see page 7578).
	PSSetCommandText (see page 7596)	Assigns a specified SQL command to be executed by PSExecute (see page 7590).
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 7597)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).
	PSStartTransaction (see page 7597)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7589).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 7598)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Requery (see page 1191)	Refreshes the recordset.
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 1192)	Saves a recordset to a file.
	Seek (see page 1192)	Searches for a record using the current index.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	Supports (see page 1197)	Indicates whether a dataset supports certain types of operations.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateBatch (see page 1197)	Writes pending batch updates to disk.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 1199)	Indicates the update status of the current record.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.

Public Properties

	Name	Description
	Active (see page 7004)	Specifies whether or not a dataset is open.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.

	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
R	DataSource (see page 7014)	Represents the data source of another dataset that supplies values to the dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignerData (see page 1208)	Stores custom data.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
	FieldDefs (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.
	FilterGroup (see page 1214)	Filters a recordset based on row update status.
	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
	IndexDefs (see page 1141)	Contains the definitions of the indexes for the current table.
R	IndexFieldCount (see page 1216)	Indicates the number of fields that comprise the current key.
	IndexFields (see page 1217)	Retrieves or sets a field for an index.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	Modified (see page 7029)	Indicates whether the active record is modified.
	ObjectView (see page 7031)	Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Properties (see page 1221)	This is Properties, a member of class TCustomADODataSet.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
	Recordset (see page 1221)	This is Recordset, a member of class TCustomADODataSet.
R	RecordsetState (see page 1222)	This is RecordsetState, a member of class TCustomADODataSet.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
R	RecordStatus (see page 1222)	Indicates the status of the current row.
	Sort (see page 1222)	Specifies the sort order of the recordset.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterCancel (see page 1224)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 1226)	Occurs after an application closes a dataset.
	AfterDelete (see page 1227)	Occurs after an application deletes a record.
	AfterEdit (see page 1230)	Occurs after an application starts editing a record.
	AfterInsert (see page 1231)	Occurs after an application inserts a new record.
	AfterOpen (see page 1231)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 1233)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 1233)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 1233)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 1234)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 1234)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 1235)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 1235)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 1236)	Occurs before an application enters insert mode.
	BeforeOpen (see page 1237)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 1238)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 1239)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 1239)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 1239)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 1240)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 1241)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnEndOfRecordset (see page 1241)	Occurs when moving a recordset's row pointer past the last row.
	OnFetchComplete (see page 1242)	Occurs after records have been retrieved into the recordset.
	OnFetchProgress (see page 1242)	Occurs periodically during an asynchronous data retrieval operation.
	OnFieldChangeComplete (see page 1243)	Occurs after a field has changed.
	OnFilterRecord (see page 1244)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnMoveComplete (see page 1244)	Occurs after the row pointer in a recordset has moved.
	OnNewRecord (see page 1245)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 1245)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnRecordChangeComplete (see page 1246)	Occurs after one or more records in the recordset change.
	OnRecordsetChangeComplete (see page 1247)	Occurs after the recordset has changed.
	OnRecordsetCreate (see page 1247)	Occurs when the recordset has been initialized and is ready to use.
	OnWillChangeField (see page 1248)	Occurs before a field is changed.
	OnWillChangeRecord (see page 1248)	Occurs before a record in the recordset is changed.
	OnWillChangeRecordset (see page 1249)	Occurs before the recordset has changed.
	OnWillMove (see page 1250)	Occurs before the row pointer in a recordset is moved.

Published Properties

	Name	Description
	Active (see page 1201)	Specifies whether or not a dataset is open.
	AutoCalcFields (see page 1203)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	CacheSize (see page 1203)	Specifies the size of the cache for the dataset.
	CommandTimeout (see page 1140)	Specifies amount of time to attempt execution of a command.
	Connection (see page 1205)	Specifies the ADO connection component to use.
	ConnectionString (see page 1206)	Specifies the connection information for the data store.

	CursorLocation (see page 1207)	Specifies whether the cursor for the connection is client-side or server-side.
	CursorType (see page 1208)	Specifies type of cursor an ADO dataset uses.
	EnableBCD (see page 1141)	Specifies whether to treat numeric fields as floating-point or BCD.
	ExecuteOptions (see page 1209)	Specifies the characteristics of a command execution.
	Filter (see page 1212)	Specifies the text of the current filter for a recordset.
	Filtered (see page 1214)	Specifies whether or not filtering is active for a dataset.
	IndexFieldNames (see page 1142)	Lists the columns to use for a persistent sort of the recordset.
	IndexName (see page 1143)	Specifies the currently active index.
	LockType (see page 1218)	Specifies the lock type to use when opening a dataset.
	MarshalOptions (see page 1218)	Specifies which records are marshaled back to the server.
	MasterFields (see page 1143)	Specifies field or fields on which a Master-Detail link is based.
	MasterSource (see page 1144)	Specifies the data source component used for a Master-Detail relationship.
	MaxRecords (see page 1219)	Specifies how many rows to return in a result set.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ReadOnly (see page 1144)	Specifies whether the data retrieved is read-only.
	TableDirect (see page 1145)	Specifies how the table is referenced.
	TableName (see page 1145)	Indicates the base table operated on.
	Tag (see page 2808)	Stores an integer value as part of a component.

8.1.8.2 TADOTable.Create Constructor

Creates an instance of a TADOTable (see page 1125) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TADOTable(TComponent * AOwner);
```

Description

Call Create to instantiate an ADO table component at runtime. Table components placed in forms or data modules at design time are created automatically.

Create

Calls its inherited Create method.

Sets the CommandType (see page 1205) property to cmdTable.

See Also

TADOStoredProc (see page 1106)

TADOTable (see page 1125)

Destroy

8.1.8.3 TADOTable Methods

The methods of the TADOTable class are listed here.

Public Methods

	Name	Description
	GetIndexNames (see page 1140)	Retrieves a list of available indexes for a table.

8.1.8.3.1 TADOTable.GetIndexNames Method

Retrieves a list of available indexes for a table.

Pascal

```
procedure GetIndexNames(List: TStrings);
```

C++

```
__fastcall GetIndexNames(TStrings List);
```

Description

Call GetIndexNames to retrieve a list of all available indexes for a table. GetIndexNames fills a string list object (such as a TStringList) with the names of the index for the table represented by the TADOTable (see page 1125) component (and only that table). Any previous contents in the string list object are completely overwritten by the new information.

List is the string list object, created and maintained by the application, into which the index names are retrieved. This string list object must be created prior to the call to GetIndexNames.

The command line below fills the Items property of a TListBox with the names of indexes for the table represented by ADOTable1.

```
ADOTable1.GetIndexNames(ListBox1.Items);
ADOTable1->GetIndexNames(ListBox1->Items);
```

See Also

Obtaining Information About Indexes

8.1.8.4 TADOTable Properties

The properties of the TADOTable class are listed here.

Public Properties

	Name	Description
	IndexDefs (see page 1141)	Contains the definitions of the indexes for the current table.

Published Properties

	Name	Description
	CommandTimeout (see page 1140)	Specifies amount of time to attempt execution of a command.
	EnableBCD (see page 1141)	Specifies whether to treat numeric fields as floating-point or BCD.
	IndexFieldNames (see page 1142)	Lists the columns to use for a persistent sort of the recordset.
	IndexName (see page 1143)	Specifies the currently active index.
	MasterFields (see page 1143)	Specifies field or fields on which a Master-Detail link is based.
	MasterSource (see page 1144)	Specifies the data source component used for a Master-Detail relationship.
	ReadOnly (see page 1144)	Specifies whether the data retrieved is read-only.
	TableDirect (see page 1145)	Specifies how the table is referenced.
	TableName (see page 1145)	Indicates the base table operated on.

8.1.8.4.1 TADOTable.CommandTimeout Property

Specifies amount of time to attempt execution of a command.

Pascal

```
property CommandTimeout: Integer;
```

C++

```
__property int CommandTimeout;
```

Description

Use CommandTimeout to specify the amount of time that expires before an attempt to execute a command is considered unsuccessful. Measured in seconds, the default value for CommandTimeout is 30.

If a command is successfully executed prior to the expiration of the seconds specified, CommandTimeout has no effect.

See Also

commandtext (see page 1204)

CommandType (see page 1205)

8.1.8.4.2 TADOTable.EnableBCD Property

Specifies whether to treat numeric fields as floating-point or BCD.

Pascal

```
property EnableBCD: Boolean;
```

C++

```
__property Boolean EnableBCD;
```

Description

Use the EnableBCD property to specify how fields are mapped to field classes. If EnableBCD is true, (the default value) adDecimal and adNumeric fields are mapped to the TBCDField class when field objects are created. If EnableBCD is false, the fields are mapped to the TFloatField class.

EnableBCD determines whether numeric and decimal fields are translated as floating-point values or binary coded decimal (BCD) values. BCD values eliminate the rounding errors associated with floating point math (such as a 3 * (2/3) resulting in 2.00000000001).

If persistent field objects are used (such as those created using the Fields Editor), the field class associated with a given field can contradict the setting of EnableBCD. Three numeric fields in the same table could, for instance, each be represented by a TFloatField, a TBCDField, and a TVariantField – regardless of the value in EnableBCD. Select field classes on a field-by-field basis for greater flexibility.

Note: For numeric values with more than 4 digits to the right of the decimal place, use of TFloatField is generally better. This is because TBCDField uses the currency data type that has a fixed scale of 4 decimal places.

Note: For fields with very large numbers of more than 19 significant digits, you can use TVariantField type persistent field objects. The TFloatField and TBCDField classes lack sufficient capacity for fields of this size. TVariantField allows the getting and setting of the field data as strings, preventing without loss of data due to insufficient capacity. However, arithmetic operations cannot be performed on numbers accessed through a TVariantField object.

See Also

TBCDField (see page 6776)

TFloatField (see page 7229)

8.1.8.4.3 TADOTable.IndexDefs Property

Contains the definitions of the indexes for the current table.

Pascal

```
property IndexDefs: TIndexDefs;
```

C++

```
__property TIndexDefs IndexDefs;
```

Description

IndexDefs contains definitions for each index the current table has. IndexDefs is a collection of TIndexDef objects. Each TIndexDef object contains the definition for one index.

Inspect IndexDefs at runtime to get information on a given index. Use an index with IndexDefs to address a specific TIndexDef object. Use properties of TIndexDef to evaluate various index attributes. The example below retrieves the fields list for the third index of the table accessed through the component ADOTable1.

```
ADOTable1.IndexDefs.Update;
StrVar := ADOTable1.IndexDefs[2].Fields;
ADOTable1->IndexDefs->Update();
StrVar = ADOTable1->IndexDefs[2] ->Fields;
```

Call the Update method (TIndexDefs) prior to inspecting the property to ensure it contains the most current index information for the table.

See Also

TIndexDef (see page 7283)

TIndexDefs (see page 7292)

Update (see page 7299)

8.1.8.4.4 TADOTable.IndexFieldNames Property

Lists the columns to use for a persistent sort of the recordset.

Pascal

```
property IndexFieldNames: string;
```

C++

```
__property AnsiString IndexFieldNames;
```

Description

Use IndexFieldNames as an alternative method of specifying the sort order of a recordset. IndexFieldNames is a direct implementation of the Sort property of the ADO recordset object, with the field list from IndexFieldNames being used for the Sort property. The actual sorting of data occurs on the OLE DB side and use of indexes for the sort depend on the particular provider.

To use IndexFieldNames to establish a recordset's sort order, the recordset must have been opened using a client-side cursor (the dataset component's CursorLocation property must be clUseClient). An exception is raised on attempts to use IndexFieldNames with server-side cursors.

IndexFieldNames may be used to set the sort order for the detail table in a master-detail link, increasing the performance of the link. However, use of IndexFieldNames is not required for master-detail links.

See Also

IndexName (see page 1217)

Creating an Index with IndexFieldNames

8.1.8.4.5 TADOTable.IndexName Property

Specifies the currently active index.

Pascal

```
property IndexName: WideString;
```

C++

```
__property BSTR IndexName;
```

Description

Use IndexName to activate an index and cause it to actively order the dataset's rows. At runtime, set IndexName to a string containing the name of the index. At design-time, select the desired index from the drop-down list in the Object Inspector.

```
ADOTable1.IndexName := 'LastName';
ADOTable1->IndexName = "LastName";
```

Set IndexName to an empty string to have no index actively ordering the dataset. The rows then appear in natural order. In some database systems, the primary index of a table does not have a name. In these cases, setting IndexName to an empty string activates the primary index.

An index must be active for certain index dependent operations to be used, such as the Seek method.

Note: Many providers do not support changing indexes while the dataset is active, raising an exception when an attempt is made to do this. Ensure that the dataset is closed before changing the index through the IndexName property and reopen the dataset after specifying a new index.

See Also

Seek (see page 1192)

IndexDefs (see page 1215)

IndexFieldNames (see page 1216)

IndexFieldCount (see page 1216)

8.1.8.4.6 TADOTable.MasterFields Property

Specifies field or fields on which a Master-Detail link is based.

Pascal

```
property MasterFields: WideString;
```

C++

```
__property BSTR MasterFields;
```

Description

Use MasterFields after setting the MasterSource or DataSource property to specify the names of one or more fields in another, master, dataset that are used to establish a Master-Detail relationship between this dataset and the master dataset. The master dataset is specified by assigning its data source to the MasterSource or DataSource property.

MasterFields is a string containing one or more field names in the master dataset. When the datasets are related based on two fields, separate field names with semicolons:

```
ADOTable2.MasterSource := DataSource1;
ADOTable2.MasterFields := 'CustID;SaleDate'
ADOTable2->MasterSource = DataSource1;
ADOTable2->MasterFields = "CustID;SaleDate"
```

Each time the current record in the master dataset changes, the new values in those fields are used to select corresponding records in this table for display.

Note: At design time, use the Field Link designer to establish the master-detail relationship between two tables.

See Also

MasterSource (see page 1144)

DataSource (see page 7014)

Making the Table a Detail of Another Dataset

8.1.8.4.7 TADOTable.MasterSource Property

Specifies the data source component used for a Master-Detail relationship.

Pascal

```
property MasterSource: TDataSource;
```

C++

```
__property TDataSource MasterSource;
```

Description

Use MasterSource to specify the name of the data source component whose DataSet property identifies a dataset to use as a Master table in establishing a Master-Detail relationship between this table and another one.

After setting the MasterSource property, specify which fields to use in the master table by setting the MasterFields (see page 1143) property. At runtime each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this table for display.

```
ADOTable2.MasterSource := DataSource1;
ADOTable2.MasterFields := 'CustID';
ADOTable2->MasterSource = DataSource1;
ADOTable2->MasterFields = "CustID";
```

Note: At design time choose an available data source from the MasterSource property's drop-down list in the Object Inspector.

See Also

TDataSource (see page 7070)

MasterFields

Making the Table a Detail of Another Dataset

8.1.8.4.8 TADOTable.ReadOnly Property

Specifies whether the data retrieved is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use the ReadOnly property to prevent users from updating, inserting, or deleting data through the TADOTable (see page 1125). By default, ReadOnly is False, meaning users can alter a table's data.

See Also

CanModify

8.1.8.4.9 TADOTable.TableDirect Property

Specifies how the table is referenced.

Pascal

```
property TableDirect: Boolean;
```

C++

```
__property Boolean TableDirect;
```

Description

Set TableDirect to specify whether the table is accessed simply via its name or whether a background SQL statement is used. Not all providers support accessing a table by its name and require that this access be done with a SELECT statement. If TableDirect is true the data retrieval request uses a background SQL statement and all columns are simply returned. If it is false, the TADOTable (see page 1125) component creates a SELECT statement to retrieve the table's data. The default value of TableDirect is false.

8.1.8.4.10 TADOTable.TableName Property

Indicates the base table operated on.

Pascal

```
property TableName: WideString;
```

C++

```
__property BSTR TableName;
```

Description

Use TableName to specify the base table in a database on which the ADO table component operates. Data is retrieved from and changes written to this base table.

Before setting or changing the value of TableName, you should deactivate the ADO table component by calling its Close method or setting its Active (see page 1201) property to false.

At design-time, if a valid connection to a database can be established, the Object Inspector provides a drop-down list of names of available tables.

See Also

Close (see page 6920)

Active (see page 7004)

ConnectionString

GetTableNames (see page 1015)

8.1.9 TCustomADODataSet Class

TCustomADODataSet is the virtualized base class for all ADO dataset components.

Class Hierarchy

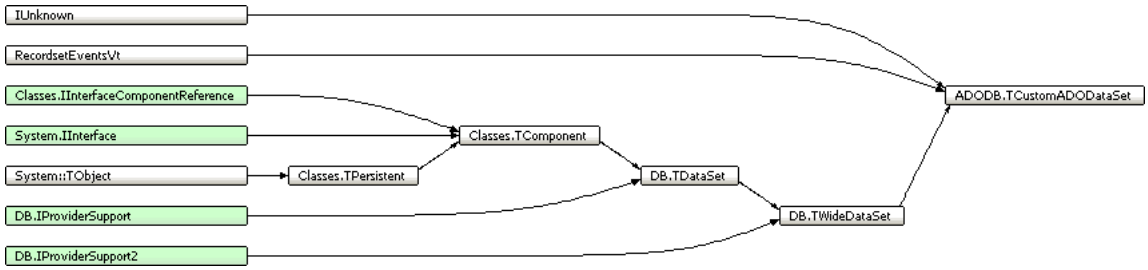

Pascal

```
TCustomADODataSet = class(TWideDataSet, IUnknown, RecordsetEventsVt);
```

C++

```
class TCustomADODataSet : public TWideDataSet, public IUnknown, public RecordsetEventsVt;
```

File

ADODB

Description

TCustomADODataSet encapsulates a set of properties, events, and methods for working with data accessed through an ADO (ActiveX Data Objects) provider. TCustomADODataSet is the base class from which all ADO dataset components descend. Applications should never instantiate TCustomADODataSet objects. Instead, instantiate an object from one of the descendant classes: TADODataSet (see page 1041), TADOTable (see page 1125), TADOQuery (see page 1084), or TADOStoredProc (see page 1106).

TCustomADODataSet cannot be instantiated. Developers must use or derive descendants of TCustomADODataSet that declare and implement methods with no implementation in TCustomADODataSet. Many of the other TCustomADODataSet methods are declared and implemented in TCustomADODataSet as virtual methods, but the implementations are merely stubs that are reimplemented in descendants such as TADODataSet (see page 1041), TADOQuery (see page 1084), TADOStoredProc (see page 1106), and TADOTable (see page 1125).

See Also

TDataSet (see page 6894)

TADODataSet (see page 1041)

TADOTable (see page 1125)

TADOQuery (see page 1084)

TADOStoredProc (see page 1106)

8.1.9.1 TCustomADODataSet Members

The following tables list the members exposed by TCustomADODataSet.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	ActivateTextFilter (see page 1160)	This is ActivateTextFilter, a member of class TCustomADODataSet.
V	AllocRecordBuffer (see page 1160)	This is AllocRecordBuffer, a member of class TCustomADODataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CheckActive (see page 1162)	This is CheckActive, a member of class TCustomADODataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
V	CheckFieldCompatibility (see page 1162)	This is CheckFieldCompatibility, a member of class TCustomADODataSet.
V	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
V	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
V	ClearCalcFields (see page 1162)	This is ClearCalcFields, a member of class TCustomADODataSet.
V	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
V	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
	ConnectionPoint (see page 1163)	This is ConnectionPoint, a member of class TCustomADODataSet.
V	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
V	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
V	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
V	DataEvent (see page 1164)	This is DataEvent, a member of class TCustomADODataSet.
	DeactivateFilters (see page 1164)	This is DeactivateFilters, a member of class TCustomADODataSet.
V	DefChanged (see page 1165)	This is DefChanged, a member of class TCustomADODataSet.
V	DefineProperties (see page 1165)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
V	DestroyLookupCursor (see page 1166)	This is DestroyLookupCursor, a member of class TCustomADODataSet.
V	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
V	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
V	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
V	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
V	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
V	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
V	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
V	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
V	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
V	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
V	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
V	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
V	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
V	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
V	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
V	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
V	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
V	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
V	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
V	DoOnNewRecord (see page 1166)	This is DoOnNewRecord, a member of class TCustomADODataSet.
	EnableEvents (see page 1166)	This is EnableEvents, a member of class TCustomADODataSet.

	EndOfRecordset (see page 1167)	This is EndOfRecordset, a member of class TCustomADODataSet.
	FetchComplete (see page 1167)	This is FetchComplete, a member of class TCustomADODataSet.
	FetchProgress (see page 1167)	This is FetchProgress, a member of class TCustomADODataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FieldChangeComplete (see page 1167)	This is FieldChangeComplete, a member of class TCustomADODataSet.
V	FindRecord (see page 1168)	This is FindRecord, a member of class TCustomADODataSet.
V	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
V	FreeRecordBuffer (see page 1168)	This is FreeRecordBuffer, a member of class TCustomADODataSet.
	GetActiveRecBuf (see page 1168)	This is GetActiveRecBuf, a member of class TCustomADODataSet.
V	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
V	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
V	GetBookmarkData (see page 1169)	This is GetBookmarkData, a member of class TCustomADODataSet.
V	GetBookmarkFlag (see page 1169)	This is GetBookmarkFlag, a member of class TCustomADODataSet.
V	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
V	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
V	GetCanModify (see page 1169)	This is GetCanModify, a member of class TCustomADODataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
	GetCommandText (see page 1170)	This is GetCommandText, a member of class TCustomADODataSet.
V	GetDataSource (see page 1170)	This is GetDataSource, a member of class TCustomADODataSet.
V	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
V	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
V	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
V	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
V	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
V	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
V	GetRecNo (see page 1171)	This is GetRecNo, a member of class TCustomADODataSet.
V	GetRecord (see page 1172)	This is GetRecord, a member of class TCustomADODataSet.
V	GetRecordCount (see page 1172)	This is GetRecordCount, a member of class TCustomADODataSet.
V	GetRecordSize (see page 1172)	This is GetRecordSize, a member of class TCustomADODataSet.
V	GetStateFieldValue (see page 1172)	This is GetStateFieldValue, a member of class TCustomADODataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
	InitOleDBAccess (see page 1173)	This is InitOleDBAccess, a member of class TCustomADODataSet.
V	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
V	InternalAddRecord (see page 1173)	This is InternalAddRecord, a member of class TCustomADODataSet.
V	InternalCancel (see page 1173)	This is InternalCancel, a member of class TCustomADODataSet.
V	InternalClose (see page 1173)	This is InternalClose, a member of class TCustomADODataSet.
V	InternalDelete (see page 1173)	This is InternalDelete, a member of class TCustomADODataSet.
V	InternalEdit (see page 1174)	This is InternalEdit, a member of class TCustomADODataSet.
V	InternalFirst (see page 1174)	This is InternalFirst, a member of class TCustomADODataSet.
	InternalGetRecord (see page 1174)	This is InternalGetRecord, a member of class TCustomADODataSet.
V	InternalGotoBookmark (see page 1174)	This is InternalGotoBookmark, a member of class TCustomADODataSet.

	InternalHandleException (see page 1175)	This is InternalHandleException, a member of class TCustomADODataSet.
	InternalInitFieldDefs (see page 1175)	This is InternalInitFieldDefs, a member of class TCustomADODataSet.
	InternalInitRecord (see page 1175)	This is InternalInitRecord, a member of class TCustomADODataSet.
	InternalInsert (see page 1175)	This is InternalInsert, a member of class TCustomADODataSet.
	InternalLast (see page 1175)	This is InternalLast, a member of class TCustomADODataSet.
	InternalOpen (see page 1176)	This is InternalOpen, a member of class TCustomADODataSet.
	InternalPost (see page 1176)	This is InternalPost, a member of class TCustomADODataSet.
	InternalRefresh (see page 1176)	This is InternalRefresh, a member of class TCustomADODataSet.
	InternalRequery (see page 1176)	This is InternalRequery, a member of class TCustomADODataSet.
	InternalSetSort (see page 1177)	This is InternalSetSort, a member of class TCustomADODataSet.
	InternalSetToRecord (see page 1177)	This is InternalSetToRecord, a member of class TCustomADODataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsCursorOpen (see page 1177)	This is IsCursorOpen, a member of class TCustomADODataSet.
	Loaded (see page 1177)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1179)	This is LocateRecord, a member of class TCustomADODataSet.
	MasterChanged (see page 1180)	This is MasterChanged, a member of class TCustomADODataSet.
	MasterDisabled (see page 1180)	This is MasterDisabled, a member of class TCustomADODataSet.
	MoveComplete (see page 1180)	This is MoveComplete, a member of class TCustomADODataSet.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	OpenCursor (see page 1181)	This is OpenCursor, a member of class TCustomADODataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PrepareCursor (see page 1181)	This is PrepareCursor, a member of class TCustomADODataSet.
	PSEndTransaction (see page 1181)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 1182)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 1182)	Executes a specified SQL command.
	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSExecuteStatement (see page 7535)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 1183)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 1183)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandTextW (see page 7537)	This is PSGetCommandTextW, a member of class TWideDataSet.
	PSGetCommandType (see page 1184)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 1184)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 1185)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7538)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 1185)	This is PSGetKeyFieldsW, a member of class TCustomADODataSet.
	PSGetParams (see page 1185)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7538)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 1186)	This is PSGetQuoteCharW, a member of class TCustomADODataSet.
	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7539)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 1186)	This is PSGetTableNameW, a member of class TCustomADODataSet.
	PSGetUpdateException (see page 1186)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 1187)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 1187)	Indicates whether the dataset includes native SQL support.

	PSIsSQLSupported (see page 1188)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 1188)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 1188)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetCommandText (see page 7540)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 1189)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 1189)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 1190)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RecordChangeComplete (see page 1190)	This is RecordChangeComplete, a member of class TCustomADODataSet.
	RecordsetChangeComplete (see page 1190)	This is RecordsetChangeComplete, a member of class TCustomADODataSet.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	RefreshParams (see page 1191)	This is RefreshParams, a member of class TCustomADODataSet.
	ReleaseLock (see page 1191)	This is ReleaseLock, a member of class TCustomADODataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBlockReadSize (see page 1194)	This is SetBlockReadSize, a member of class TCustomADODataSet.
	SetBookmarkData (see page 1194)	This is SetBookmarkData, a member of class TCustomADODataSet.
	SetBookmarkFlag (see page 1194)	This is SetBookmarkFlag, a member of class TCustomADODataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCommandText (see page 1194)	This is SetCommandText, a member of class TCustomADODataSet.
	SetConnection (see page 1194)	This is SetConnection, a member of class TCustomADODataSet.
	SetConnectionFlag (see page 1195)	This is SetConnectionFlag, a member of class TCustomADODataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDataSource (see page 1195)	This is SetDataSource, a member of class TCustomADODataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDetailFilter (see page 1195)	This is SetDetailFilter, a member of class TCustomADODataSet.
	SetFieldData (see page 1195)	This is the overview for the SetFieldData method overload.
	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
	SetFiltered (see page 1196)	This is SetFiltered, a member of class TCustomADODataSet.
	SetFilterOptions (see page 1196)	This is SetFilterOptions, a member of class TCustomADODataSet.
	SetFilterText (see page 1196)	This is SetFilterText, a member of class TCustomADODataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetParamsFromCursor (see page 1197)	This is SetParamsFromCursor, a member of class TCustomADODataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetRecNo (see page 1197)	This is SetRecNo, a member of class TCustomADODataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.

	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateIndexDefs (see page 1198)	This is UpdateIndexDefs, a member of class TCustomADODataSet.
	UpdateRecordSetPosition (see page 1198)	This is UpdateRecordSetPosition, a member of class TCustomADODataSet.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WillChangeField (see page 1199)	This is WillChangeField, a member of class TCustomADODataSet.
	WillChangeRecord (see page 1199)	This is WillChangeRecord, a member of class TCustomADODataSet.
	WillChangeRecordset (see page 1200)	This is WillChangeRecordset, a member of class TCustomADODataSet.
	WillMove (see page 1200)	This is WillMove, a member of class TCustomADODataSet.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
R	Command (see page 1204)	Provides access to the internal command object that executes commands for the dataset.
	CommandText (see page 1204)	This is CommandText, a member of class TCustomADODataSet.
	CommandTimeout (see page 1204)	Specifies amount of time to attempt execution of a command.
	CommandType (see page 1205)	Specifies the type of command to execute.
	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
	DataSource (see page 1208)	Represents the data source of another dataset that supplies values to the dataset.
	FieldDefs (see page 1210)	Points to the list of field definitions for the dataset.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
	IndexDefs (see page 1215)	Contains the definitions of the indexes for the current table.
	IndexFieldNames (see page 1216)	Lists the columns to use for a persistent sort of the recordset.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
R	MasterDataLink (see page 1218)	Manages the relationship between the ADO dataset and a master table.
	MasterFields (see page 1219)	Specifies field or fields on which a Master-Detail link is based.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
	ParamCheck (see page 1220)	Specifies whether the parameter list for an ADO dataset component is regenerated if the SQL changes at runtime.
	Parameters (see page 1220)	Contains the collection of parameters for an SQL statement.
	Prepared (see page 1221)	Specifies whether the command is prepared before execution.
	Reserved (see page 7035)	For internal use only.

	StoreDefs (see page 1223)	Specifies whether field definitions are stored with the ADO dataset.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 1160)	Tests the validity of a specified bookmark.
V	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
	CancelBatch (see page 1161)	Cancels a pending batch update.
	CancelUpdates (see page 1162)	Cancels pending cached updates.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Clone (see page 1163)	Duplicates another dataset in the calling component
	Close (see page 6920)	Closes a dataset.
V	CompareBookmarks (see page 1163)	Compares two bookmarks.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
V	Create (see page 1156)	Creates an instance of a TCustomADONETConnector component.
V	CreateBlobStream (see page 1164)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
	DeleteRecords (see page 1165)	Deletes one or more records in the dataset.
V	Destroy (see page 1157)	Destroys the TCustomADONETConnector instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FilterOnBookmarks (see page 1168)	Filters a data set to just rows with pre-defined bookmarks.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.

	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetBlobFieldData (see page 1169)	Retrieves the current value of a field into a buffer.
V	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
V	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
V	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
V	GetDetailLinkFields (see page 1170)	Lists the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetFieldData (see page 1171)	This is the overview for the GetFieldData method overload.
V	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
V	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
V	IsSequenced (see page 1177)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	Last (see page 6968)	Moves to the last record in the dataset.
	LoadFromFile (see page 1178)	Loads a recordset from a file.
V	Locate (see page 1179)	Searches the dataset for a specified record and makes that record the current record.
V	Lookup (see page 1180)	Retrieves field values from a row that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	Next (see page 6973)	Moves to the next record in the dataset.
	NextRecordset (see page 1181)	Provides access to a secondary recordset.
	Open (see page 6976)	Opens the dataset.
V	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7593)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7594)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7594)	Returns the name of the dataset table that appears in generated SQL statements.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Requery (see page 1191)	Refreshes the recordset.
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 1192)	Saves a recordset to a file.
	Seek (see page 1192)	Searches for a record using the current index.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	Supports (see page 1197)	Indicates whether a dataset supports certain types of operations.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateBatch (see page 1197)	Writes pending batch updates to disk.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 1199)	Indicates the update status of the current record.

Public Properties

	Name	Description
R	AggFields (see page 7006)	Contains any aggregate fields.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.

R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignerData (see page 1208)	Stores custom data.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EnableBCD (see page 1209)	Specifies whether to treat numeric fields as floating-point or BCD.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
	FilterGroup (see page 1214)	Filters a recordset based on row update status.
	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
R	IndexFieldCount (see page 1216)	Indicates the number of fields that comprise the current key.
	IndexFields (see page 1217)	Retrieves or sets a field for an index.
	IndexName (see page 1217)	Specifies the currently active index.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	Modified (see page 7029)	Indicates whether the active record is modified.
	ObjectView (see page 7031)	Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Properties (see page 1221)	This is Properties, a member of class TCustomADODataSet.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
	Recordset (see page 1221)	This is Recordset, a member of class TCustomADODataSet.
R	RecordsetState (see page 1222)	This is RecordsetState, a member of class TCustomADODataSet.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
R	RecordStatus (see page 1222)	Indicates the status of the current row.
	Sort (see page 1222)	Specifies the sort order of the recordset.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterCancel (see page 1224)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 1226)	Occurs after an application closes a dataset.
	AfterDelete (see page 1227)	Occurs after an application deletes a record.
	AfterEdit (see page 1230)	Occurs after an application starts editing a record.
	AfterInsert (see page 1231)	Occurs after an application inserts a new record.
	AfterOpen (see page 1231)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 1233)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 1233)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 1233)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 1234)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 1234)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 1235)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 1235)	Occurs before an application enters edit mode for the active record.

	BeforeInsert (see page 1236)	Occurs before an application enters insert mode.
	BeforeOpen (see page 1237)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 1238)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 1239)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 1239)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 1239)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 1240)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 1241)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnEndOfRecordset (see page 1241)	Occurs when moving a recordset's row pointer past the last row.
	OnFetchComplete (see page 1242)	Occurs after records have been retrieved into the recordset.
	OnFetchProgress (see page 1242)	Occurs periodically during an asynchronous data retrieval operation.
	OnFieldChangeComplete (see page 1243)	Occurs after a field has changed.
	OnFilterRecord (see page 1244)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnMoveComplete (see page 1244)	Occurs after the row pointer in a recordset has moved.
	OnNewRecord (see page 1245)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 1245)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnRecordChangeComplete (see page 1246)	Occurs after one or more records in the recordset change.
	OnRecordsetChangeComplete (see page 1247)	Occurs after the recordset has changed.
	OnRecordsetCreate (see page 1247)	Occurs when the recordset has been initialized and is ready to use.
	OnWillChangeField (see page 1248)	Occurs before a field is changed.
	OnWillChangeRecord (see page 1248)	Occurs before a record in the recordset is changed.
	OnWillChangeRecordset (see page 1249)	Occurs before the recordset has changed.
	OnWillMove (see page 1250)	Occurs before the row pointer in a recordset is moved.

Published Properties

	Name	Description
	Active (see page 1201)	Specifies whether or not a dataset is open.
	AutoCalcFields (see page 1203)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	CacheSize (see page 1203)	Specifies the size of the cache for the dataset.
	Connection (see page 1205)	Specifies the ADO connection component to use.
	ConnectionString (see page 1206)	Specifies the connection information for the data store.
	CursorLocation (see page 1207)	Specifies whether the cursor for the connection is client-side or server-side.
	CursorType (see page 1208)	Specifies type of cursor an ADO dataset uses.
	ExecuteOptions (see page 1209)	Specifies the characteristics of a command execution.
	Filter (see page 1212)	Specifies the text of the current filter for a recordset.
	Filtered (see page 1214)	Specifies whether or not filtering is active for a dataset.
	LockType (see page 1218)	Specifies the lock type to use when opening a dataset.
	MarshalOptions (see page 1218)	Specifies which records are marshaled back to the server.
	MaxRecords (see page 1219)	Specifies how many rows to return in a result set.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

8.1.9.2 TCustomADODataSet.Create Constructor

Creates an instance of a TCustomADONETConnector component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomADODataSet(TComponent * AOwner);
```

Description

Call Create to instantiate an ADO dataset component at runtime. TCustomADONETConnector contains abstract methods, and is not intended to be used in applications. Instead, applications instantiate dataset descendants such as TADODataSet (see page 1041), TADOTable (see page 1125), TADOQuery (see page 1084), or TADOStoredProc (see page 1106).

AOwner is the component that becomes the value of the Owner property. Typically, this is the form or data module that contains the dataset.

See Also

TADOStoredProc (see page 1106)

TADOTable (see page 1125)

Owner (see page 2807)

8.1.9.3 TCustomADODataSet.Destroy Destructor

Destroys the TCustomADONETConnector instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomADODataSet();
```

Description

Do not call Destroy directly in an application. Usually destruction of datasets is handled automatically by Delphi. If an application creates its own instances of a dataset, however, and does not assign an Owner that is responsible for freeing the dataset, then the application should call Free, which verifies that the dataset reference is not nil before calling Destroy.

Destroy performs the following tasks:

Closes the dataset.

Disconnects from the associated database.

Calls the inherited destructor.

See Also

Designer (see page 7015)

FieldDefs (see page 7019)

ADONETDb.TCustomADONETConnector.Create

Free (see page 22300)

8.1.9.4 TCustomADODataSet Methods

The methods of the TCustomADODataSet class are listed here.

Protected Methods

	Name	Description
	ActivateTextFilter (see page 1160)	This is ActivateTextFilter, a member of class TCustomADODataSet.
V	AllocRecordBuffer (see page 1160)	This is AllocRecordBuffer, a member of class TCustomADODataSet.
V	CheckActive (see page 1162)	This is CheckActive, a member of class TCustomADODataSet.
V	CheckFieldCompatibility (see page 1162)	This is CheckFieldCompatibility, a member of class TCustomADODataSet.
V	ClearCalcFields (see page 1162)	This is ClearCalcFields, a member of class TCustomADODataSet.
	ConnectionPoint (see page 1163)	This is ConnectionPoint, a member of class TCustomADODataSet.
V	DataEvent (see page 1164)	This is DataEvent, a member of class TCustomADODataSet.
	DeactivateFilters (see page 1164)	This is DeactivateFilters, a member of class TCustomADODataSet.
V	DefChanged (see page 1165)	This is DefChanged, a member of class TCustomADODataSet.
V	DefineProperties (see page 1165)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DestroyLookupCursor (see page 1166)	This is DestroyLookupCursor, a member of class TCustomADODataSet.
V	DoOnNewRecord (see page 1166)	This is DoOnNewRecord, a member of class TCustomADODataSet.
	EnableEvents (see page 1166)	This is EnableEvents, a member of class TCustomADODataSet.
	EndOfRecordset (see page 1167)	This is EndOfRecordset, a member of class TCustomADODataSet.
	FetchComplete (see page 1167)	This is FetchComplete, a member of class TCustomADODataSet.
	FetchProgress (see page 1167)	This is FetchProgress, a member of class TCustomADODataSet.
	FieldChangeComplete (see page 1167)	This is FieldChangeComplete, a member of class TCustomADODataSet.
V	FindRecord (see page 1168)	This is FindRecord, a member of class TCustomADODataSet.
V	FreeRecordBuffer (see page 1168)	This is FreeRecordBuffer, a member of class TCustomADODataSet.
	GetActiveRecBuf (see page 1168)	This is GetActiveRecBuf, a member of class TCustomADODataSet.
V	GetBookmarkData (see page 1169)	This is GetBookmarkData, a member of class TCustomADODataSet.
V	GetBookmarkFlag (see page 1169)	This is GetBookmarkFlag, a member of class TCustomADODataSet.
V	GetCanModify (see page 1169)	This is GetCanModify, a member of class TCustomADODataSet.
	GetCommandText (see page 1170)	This is GetCommandText, a member of class TCustomADODataSet.
V	GetDataSource (see page 1170)	This is GetDataSource, a member of class TCustomADODataSet.
V	GetRecNo (see page 1171)	This is GetRecNo, a member of class TCustomADODataSet.
V	GetRecord (see page 1172)	This is GetRecord, a member of class TCustomADODataSet.
V	GetRecordCount (see page 1172)	This is GetRecordCount, a member of class TCustomADODataSet.
V	GetRecordSize (see page 1172)	This is GetRecordSize, a member of class TCustomADODataSet.
V	GetStateFieldValue (see page 1172)	This is GetStateFieldValue, a member of class TCustomADODataSet.
	InitOleDBAccess (see page 1173)	This is InitOleDBAccess, a member of class TCustomADODataSet.
V	InternalAddRecord (see page 1173)	This is InternalAddRecord, a member of class TCustomADODataSet.
V	InternalCancel (see page 1173)	This is InternalCancel, a member of class TCustomADODataSet.
V	InternalClose (see page 1173)	This is InternalClose, a member of class TCustomADODataSet.
V	InternalDelete (see page 1173)	This is InternalDelete, a member of class TCustomADODataSet.
V	InternalEdit (see page 1174)	This is InternalEdit, a member of class TCustomADODataSet.
V	InternalFirst (see page 1174)	This is InternalFirst, a member of class TCustomADODataSet.
	InternalGetRecord (see page 1174)	This is InternalGetRecord, a member of class TCustomADODataSet.
V	InternalGotoBookmark (see page 1174)	This is InternalGotoBookmark, a member of class TCustomADODataSet.
V	InternalHandleException (see page 1175)	This is InternalHandleException, a member of class TCustomADODataSet.
V	InternalInitFieldDefs (see page 1175)	This is InternalInitFieldDefs, a member of class TCustomADODataSet.
V	InternalInitRecord (see page 1175)	This is InternalInitRecord, a member of class TCustomADODataSet.
V	InternalInsert (see page 1175)	This is InternalInsert, a member of class TCustomADODataSet.
V	InternalLast (see page 1175)	This is InternalLast, a member of class TCustomADODataSet.
V	InternalOpen (see page 1176)	This is InternalOpen, a member of class TCustomADODataSet.
V	InternalPost (see page 1176)	This is InternalPost, a member of class TCustomADODataSet.
V	InternalRefresh (see page 1176)	This is InternalRefresh, a member of class TCustomADODataSet.
	InternalRequery (see page 1176)	This is InternalRequery, a member of class TCustomADODataSet.
	InternalSetSort (see page 1177)	This is InternalSetSort, a member of class TCustomADODataSet.
V	InternalSetToRecord (see page 1177)	This is InternalSetToRecord, a member of class TCustomADODataSet.
V	IsCursorOpen (see page 1177)	This is IsCursorOpen, a member of class TCustomADODataSet.

	Loaded (see page 1177)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1179)	This is LocateRecord, a member of class TCustomADODataSet.
	MasterChanged (see page 1180)	This is MasterChanged, a member of class TCustomADODataSet.
	MasterDisabled (see page 1180)	This is MasterDisabled, a member of class TCustomADODataSet.
	MoveComplete (see page 1180)	This is MoveComplete, a member of class TCustomADODataSet.
	OpenCursor (see page 1181)	This is OpenCursor, a member of class TCustomADODataSet.
	PrepareCursor (see page 1181)	This is PrepareCursor, a member of class TCustomADODataSet.
	PSEndTransaction (see page 1181)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 1182)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 1182)	Executes a specified SQL command.
	PSGetAttributes (see page 1183)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 1183)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandType (see page 1184)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 1184)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 1185)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFieldsW (see page 1185)	This is PSGetKeyFieldsW, a member of class TCustomADODataSet.
	PSGetParams (see page 1185)	Returns the current parameter values of the dataset.
	PSGetQuoteCharW (see page 1186)	This is PSGetQuoteCharW, a member of class TCustomADODataSet.
	PSGetTableNameW (see page 1186)	This is PSGetTableNameW, a member of class TCustomADODataSet.
	PSGetUpdateException (see page 1186)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 1187)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 1187)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 1188)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 1188)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 1188)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetParams (see page 1189)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 1189)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 1190)	Applies a single update to the underlying database table or tables.
	RecordChangeComplete (see page 1190)	This is RecordChangeComplete, a member of class TCustomADODataSet.
	RecordsetChangeComplete (see page 1190)	This is RecordsetChangeComplete, a member of class TCustomADODataSet.
	RefreshParams (see page 1191)	This is RefreshParams, a member of class TCustomADODataSet.
	ReleaseLock (see page 1191)	This is ReleaseLock, a member of class TCustomADODataSet.
	SetBlockReadSize (see page 1194)	This is SetBlockReadSize, a member of class TCustomADODataSet.
	SetBookmarkData (see page 1194)	This is SetBookmarkData, a member of class TCustomADODataSet.
	SetBookmarkFlag (see page 1194)	This is SetBookmarkFlag, a member of class TCustomADODataSet.
	SetCommandText (see page 1194)	This is SetCommandText, a member of class TCustomADODataSet.
	SetConnection (see page 1194)	This is SetConnection, a member of class TCustomADODataSet.
	SetConnectionFlag (see page 1195)	This is SetConnectionFlag, a member of class TCustomADODataSet.
	SetDataSource (see page 1195)	This is SetDataSource, a member of class TCustomADODataSet.
	SetDetailFilter (see page 1195)	This is SetDetailFilter, a member of class TCustomADODataSet.
	SetFieldData (see page 1195)	This is the overview for the SetFieldData method overload.
	SetFiltered (see page 1196)	This is SetFiltered, a member of class TCustomADODataSet.
	SetFilterOptions (see page 1196)	This is SetFilterOptions, a member of class TCustomADODataSet.
	SetFilterText (see page 1196)	This is SetFilterText, a member of class TCustomADODataSet.
	SetParamsFromCursor (see page 1197)	This is SetParamsFromCursor, a member of class TCustomADODataSet.
	SetRecNo (see page 1197)	This is SetRecNo, a member of class TCustomADODataSet.
	UpdateIndexDefs (see page 1198)	This is UpdateIndexDefs, a member of class TCustomADODataSet.
	UpdateRecordSetPosition (see page 1198)	This is UpdateRecordSetPosition, a member of class TCustomADODataSet.
	WillChangeField (see page 1199)	This is WillChangeField, a member of class TCustomADODataSet.
	WillChangeRecord (see page 1199)	This is WillChangeRecord, a member of class TCustomADODataSet.
	WillChangeRecordset (see page 1200)	This is WillChangeRecordset, a member of class TCustomADODataSet.
	WillMove (see page 1200)	This is WillMove, a member of class TCustomADODataSet.

Public Methods

	Name	Description
	BookmarkValid (see page 1160)	Tests the validity of a specified bookmark.
	CancelBatch (see page 1161)	Cancels a pending batch update.
	CancelUpdates (see page 1162)	Cancels pending cached updates.
	Clone (see page 1163)	Duplicates another dataset in the calling component
	CompareBookmarks (see page 1163)	Compares two bookmarks.
	CreateBlobStream (see page 1164)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	DeleteRecords (see page 1165)	Deletes one or more records in the dataset.
	FilterOnBookmarks (see page 1168)	Filters a data set to just rows with pre-defined bookmarks.
	GetBlobFieldData (see page 1169)	Retrieves the current value of a field into a buffer.
	GetDetailLinkFields (see page 1170)	Lists the field components that link this dataset as a detail of a master dataset.
	GetFieldData (see page 1171)	This is the overview for the GetFieldData method overload.
	IsSequenced (see page 1177)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	LoadFromFile (see page 1178)	Loads a recordset from a file.
	Locate (see page 1179)	Searches the dataset for a specified record and makes that record the current record.
	Lookup (see page 1180)	Retrieves field values from a row that matches specified search values.
	NextRecordset (see page 1181)	Provides access to a secondary recordset.
	Requery (see page 1191)	Refreshes the recordset.
	SaveToFile (see page 1192)	Saves a recordset to a file.
	Seek (see page 1192)	Searches for a record using the current index.
	Supports (see page 1197)	Indicates whether a dataset supports certain types of operations.
	UpdateBatch (see page 1197)	Writes pending batch updates to disk.
	UpdateStatus (see page 1199)	Indicates the update status of the current record.

8.1.9.4.1 TCustomADODataSet.ActivateTextFilter Method

Pascal

```
procedure ActivateTextFilter(const FilterText: string);
```

C++

```
__fastcall ActivateTextFilter(const AnsiString FilterText);
```

Description

This is ActivateTextFilter, a member of class TCustomADODataSet.

8.1.9.4.2 TCustomADODataSet.AllocRecordBuffer Method

Pascal

```
function AllocRecordBuffer: PChar; override;
```

C++

```
virtual __fastcall const char * AllocRecordBuffer();
```

Description

This is AllocRecordBuffer, a member of class TCustomADODataSet.

8.1.9.4.3 TCustomADODataSet.BookmarkValid Method

Tests the validity of a specified bookmark.

Pascal

```
function BookmarkValid(Bookmark: TBookmark): Boolean; override;
```

C++

```
virtual __fastcall Boolean BookmarkValid(TBookmark Bookmark);
```

Description

Call BookmarkValid to determine if a specified bookmark is currently assigned a value. BookmarkValid returns true if a bookmark is valid. Otherwise, it returns false.

Bookmark specifies the bookmark to test.

See Also

CompareBookmarks (see page 1163)

GetBookmark (see page 6948)

8.1.9.4.4 TCustomADODataSet.CancelBatch Method

Cancels a pending batch update.

Pascal

```
procedure CancelBatch(AffectRecords: TAffectRecords = arAll);
```

C++

```
__fastcall CancelBatch(TAffectRecords AffectRecords = arAll);
```

Description

Call CancelBatch to cancel any pending updates in a dataset that has been opened in batch update mode. CancelBatch may be called without error when there are no pending updates, but this has no effect.

CancelBatch is not applicable to datasets in immediate update mode. Calling CancelBatch for such datasets without the arCurrent constant raises an exception.

The default AffectsRecords parameter value is arAll.

The AffectRecords parameter may be any one of the following TAffectRecords (see page 1290) values:

Option	Meaning
arCurrent	Only the updates for the current row are applied.
arFiltered	Only updates for rows matching the current filter are applied.
arAll	Updates for all rows are applied.
arAllChapters	Update affects all chapters (ADO chapters)

Note: : To use batch updating, the CursorType (see page 1208) property of the dataset component must be either ctKeySet (the default) or ctStatic and the LockType (see page 1218) property must be ltBatchOptimistic.

See Also

LockType (see page 1218)

Filtered (see page 1214)

RecordStatus (see page 1222)

FilterGroup (see page 1214)

UpdateBatch (see page 1197)

CursorType (see page 1208),

8.1.9.4.5 TCustomADODataSet.CancelUpdates Method

Cancels pending cached updates.

Pascal

```
procedure CancelUpdates;
```

C++

```
__fastcall CancelUpdates();
```

Description

Call CancelUpdates to cancel any pending cached updates and revert field values to their original values (the values present since the last applied update, if any).

Note: CancelUpdates does the same thing as CancelBatch (see page 1161)(arAll).

See Also

CancelBatch (see page 1161)

8.1.9.4.6 TCustomADODataSet.CheckActive Method

Pascal

```
procedure CheckActive; override;
```

C++

```
virtual __fastcall CheckActive();
```

Description

This is CheckActive, a member of class TCustomADODataSet.

8.1.9.4.7 TCustomADODataSet.CheckFieldCompatibility Method

Pascal

```
procedure CheckFieldCompatibility(Field: TField; FieldDef: TFieldDef); override;
```

C++

```
virtual __fastcall CheckFieldCompatibility(TField Field, TFieldDef FieldDef);
```

Description

This is CheckFieldCompatibility, a member of class TCustomADODataSet.

8.1.9.4.8 TCustomADODataSet.ClearCalcFields Method

Pascal

```
procedure ClearCalcFields(Buffer: PChar); override;
```

C++

```
virtual __fastcall ClearCalcFields(const char * Buffer);
```

Description

This is ClearCalcFields, a member of class TCustomADODataSet.

8.1.9.4.9 TCustomADODataSet.Clone Method

Duplicates another dataset in the calling component

Pascal

```
procedure Clone(Source: TCustomADODataSet; LockType: TADOLockType = ltUnspecified);
```

C++

```
__fastcall Clone(TCustomADODataSet Source, TADOLockType LockType = ltUnspecified);
```

Description

Call Clone to make the recordset of the calling ADO dataset component a duplicate of the recordset active in another TCustomADONETConnector descendant component. If the attempt to clone a dataset is successful, the current ADO dataset component's recordset will be the same as that in the other dataset. If unsuccessful, the recordset is set to nil (Delphi) or NULL (C++) and an EADOError (see page 978) exception raised.

LockType (see page 1218) specifies the lock type to use when opening the clone recordset. By default, LockType (see page 1218) is ltUnspecified.

See Also

LockType

8.1.9.4.10 TCustomADODataSet.CompareBookmarks Method

Compares two bookmarks.

Pascal

```
function CompareBookmarks(Bookmark1: TBookmark; Bookmark2: TBookmark): Integer; override;
```

C++

```
virtual __fastcall int CompareBookmarks(TBookmark Bookmark1, TBookmark Bookmark2);
```

Description

Call CompareBookmarks to compare one bookmark to another. CompareBookmarks returns an integer value indicating the difference between the two bookmarks. CompareBookmarks returns a 0 (zero) if the two bookmarks are the same, a –1 (negative one) if bookmark one is less than bookmark two, or a 1 (positive one) if bookmark one is greater than bookmark two. CompareBookmarks also returns a zero if one of the two bookmarks is nil (Delphi) or NULL (C++).

Bookmark1 and Bookmark2 are the two TBookmark values to compare.

See Also

BookmarkValid (see page 1160)

GetBookmark (see page 6948)

8.1.9.4.11 TCustomADODataSet.ConnectionPoint Method

Pascal

```
function ConnectionPoint: IConnectionPoint;
```

C++

```
__fastcall IConnectionPoint ConnectionPoint();
```

Description

This is ConnectionPoint, a member of class TCustomADODataSet.

8.1.9.4.12 TCustomADODataSet.CreateBlobStream Method

Creates a blob stream for a Binary large object (BLOB) field in the dataset.

Pascal

```
function CreateBlobStream(Field: TField; Mode: TBlobStreamMode): TStream; override;
```

C++

```
virtual __fastcall TStream CreateBlobStream(TField Field, TBlobStreamMode Mode);
```

Description

Call CreateBlobStream to obtain a stream for reading and writing the value of the field specified by the Field parameter.

Field is a string containing the name of the BLOB (or comparable type) field in the dataset.

Mode indicates whether the stream will be used for reading the field's value (bmRead), writing the field's value (bmWrite), or both (bmReadWrite).

See Also

TBlobStream (see page 9958)

TBlobField (see page 6801)

TClientBlobStream (see page 8285)

8.1.9.4.13 TCustomADODataSet.DataEvent Method

Pascal

```
procedure DataEvent(Event: TDataEvent; Info: Longint); override;
```

C++

```
virtual __fastcall DataEvent(TDataEvent Event, Longint Info);
```

Description

This is DataEvent, a member of class TCustomADODataSet.

8.1.9.4.14 TCustomADODataSet.DeactivateFilters Method

Pascal

```
procedure DeactivateFilters;
```

C++

```
__fastcall DeactivateFilters();
```

Description

This is DeactivateFilters, a member of class TCustomADODataSet.

8.1.9.4.15 TCustomADODataSet.DefChanged Method

Pascal

```
procedure DefChanged(Sender: TObject); override;
```

C++

```
virtual __fastcall DefChanged(TObject * Sender);
```

Description

This is DefChanged, a member of class TCustomADODataSet.

8.1.9.4.16 TCustomADODataSet.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor (see page 2839)

DefineBinaryProperty (see page 2837)

DefineProperties (see page 2931)

DefineProperty (see page 2838)

8.1.9.4.17 TCustomADODataSet.DeleteRecords Method

Deletes one or more records in the dataset.

Pascal

```
procedure DeleteRecords(AffectRecords: TAffectRecords = arAll);
```

C++

```
__fastcall DeleteRecords(TAffectRecords AffectRecords = arAll);
```

Description

Use DeleteRecords to delete one or more records from the recordset the ADO dataset component represents.

If the recordset does not support record deletions, an exception is raised. Use the Supports (see page 1197) method to determine whether the recordset supports deleting records.

If the dataset was opened in batch update mode, the records are only marked for deletion. Apply the batch deletions by calling the UpdateBatch (see page 1197) method. If the dataset is not in batch update mode, the records are immediately removed from the base table.

AffectRecords is a TAffectRecords (see page 1290) constant that determines which records are deleted by DeleteRecords. The default value for the AffectRecords parameter is arAll, which causes all records in the recordset to be deleted.

Option	Meaning
arCurrent	Only the deletes the current record.
arFiltered	Only deletes records accessible through the current filter.
arAll	Deletes all records in the recordset.
arAllChapters	Deletes records in all chapters (ADO chapters)

See Also

Supports (see page 1197)

UpdateBatch (see page 1197)

8.1.9.4.18 TCustomADODataSet.DestroyLookupCursor Method

Pascal

```
procedure DestroyLookupCursor; virtual;
```

C++

```
virtual __fastcall DestroyLookupCursor();
```

Description

This is DestroyLookupCursor, a member of class TCustomADODataSet.

8.1.9.4.19 TCustomADODataSet.DoOnNewRecord Method

Pascal

```
procedure DoOnNewRecord; override;
```

C++

```
virtual __fastcall DoOnNewRecord();
```

Description

This is DoOnNewRecord, a member of class TCustomADODataSet.

8.1.9.4.20 TCustomADODataSet.EnableEvents Method

Pascal

```
procedure EnableEvents;
```

C++

```
__fastcall EnableEvents();
```

Description

This is EnableEvents, a member of class TCustomADODataSet.

8.1.9.4.21 TCustomADODataSet.EndOfRecordset Method

Pascal

```
procedure EndOfRecordset(var fMoreData: WordBool; var adStatus: EventStatusEnum; const
pRecordset: _Recordset); safecall;
```

C++

```
__fastcall EndOfRecordset(WordBool fMoreData, EventStatusEnum adStatus, const _Recordset
pRecordset);
```

Description

This is EndOfRecordset, a member of class TCustomADODataSet.

8.1.9.4.22 TCustomADODataSet.FetchComplete Method

Pascal

```
procedure FetchComplete(const pError: Error; var adStatus: EventStatusEnum; const pRecordset:
_Recordset); safecall;
```

C++

```
__fastcall FetchComplete(const Error pError, EventStatusEnum adStatus, const _Recordset
pRecordset);
```

Description

This is FetchComplete, a member of class TCustomADODataSet.

8.1.9.4.23 TCustomADODataSet.FetchProgress Method

Pascal

```
procedure FetchProgress(Progress: Integer; MaxProgress: Integer; var adStatus:
EventStatusEnum; const pRecordset: _Recordset); safecall;
```

C++

```
__fastcall FetchProgress(int Progress, int MaxProgress, EventStatusEnum adStatus, const
_Recordset pRecordset);
```

Description

This is FetchProgress, a member of class TCustomADODataSet.

8.1.9.4.24 TCustomADODataSet.FieldChangeComplete Method

Pascal

```
procedure FieldChangeComplete(cFields: Integer; Fields: OleVariant; const pError: Error; var
adStatus: EventStatusEnum; const pRecordset: _Recordset); safecall;
```

C++

```
__fastcall FieldChangeComplete(int cFields, OleVariant Fields, const Error pError,
EventStatusEnum adStatus, const _Recordset pRecordset);
```

Description

This is FieldChangeComplete, a member of class TCustomADODataSet.

8.1.9.4.25 TCustomADODataSet.FilterOnBookmarks Method

Filters a data set to just rows with pre-defined bookmarks.

Pascal

```
procedure FilterOnBookmarks(Bookmarks: array of const);
```

C++

```
__fastcall FilterOnBookmarks(array of const Bookmarks);
```

Description

Use FilterOnBookmarks to filter an ADO dataset to display only those data rows for which a bookmark has been defined. To successfully use FilterOnBookmarks, you must have already defined at least one valid bookmark in the dataset.

When FilterOnBookmarks is called, the Filter (see page 1212) property is automatically cleared, the FilterGroup (see page 1214) property is set to fgUnassigned, and the Filtered (see page 1214) property is set to true.

See Also

Filter (see page 1212)

Filtered (see page 1214)

FilterGroup (see page 1214)

8.1.9.4.26 TCustomADODataSet.FindRecord Method

Pascal

```
function FindRecord(Restart: Boolean; GoForward: Boolean): Boolean; override;
```

C++

```
virtual __fastcall Boolean FindRecord(Boolean Restart, Boolean GoForward);
```

Description

This is FindRecord, a member of class TCustomADODataSet.

8.1.9.4.27 TCustomADODataSet.FreeRecordBuffer Method

Pascal

```
procedure FreeRecordBuffer(var Buffer: PChar); override;
```

C++

```
virtual __fastcall FreeRecordBuffer(const char * Buffer);
```

Description

This is FreeRecordBuffer, a member of class TCustomADODataSet.

8.1.9.4.28 TCustomADODataSet.GetActiveRecBuf Method

Pascal

```
function GetActiveRecBuf(var RecBuf: PChar): Boolean;
```

C++

```
__fastcall Boolean GetActiveRecBuf(const char * RecBuf);
```

Description

This is GetActiveRecBuf, a member of class TCustomADODataSet.

8.1.9.4.29 TCustomADODataSet.GetBlobFieldData Method

Retrieves the current value of a field into a buffer.

Pascal

```
function GetBlobFieldData(FieldNo: Integer; var Buffer: TBlobByteData): Integer; override;
```

C++

```
virtual __fastcall int GetBlobFieldData(int FieldNo, TBlobByteData Buffer);
```

Description

Most applications do not need to call GetBlobFieldData, this method being used internally by descendants of TCustomADODataSet (see page 1145). GetBlobFieldData reads blob data from the field specified by FieldNo into a Buffer. The buffer is a dynamic array of bytes, so that it can grow to accommodate the size of the BLOB data. GetBlobFieldData returns the size of the buffer.

See Also

GetFieldData (see page 6952)

GetFieldData (see page 1171)

GetData (see page 7131)

8.1.9.4.30 TCustomADODataSet.GetBookmarkData Method

Pascal

```
procedure GetBookmarkData(Buffer: PChar; Data: Pointer); override;
```

C++

```
virtual __fastcall GetBookmarkData(const char * Buffer, void * Data);
```

Description

This is GetBookmarkData, a member of class TCustomADODataSet.

8.1.9.4.31 TCustomADODataSet.GetBookmarkFlag Method

Pascal

```
function GetBookmarkFlag(Buffer: PChar): TBookmarkFlag; override;
```

C++

```
virtual __fastcall TBookmarkFlag GetBookmarkFlag(const char * Buffer);
```

Description

This is GetBookmarkFlag, a member of class TCustomADODataSet.

8.1.9.4.32 TCustomADODataSet.GetCanModify Method

Pascal

```
function GetCanModify: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetCanModify();
```

Description

This is GetCanModify, a member of class TCustomADODataSet.

8.1.9.4.33 TCustomADODataSet.GetCommandText Method

Pascal

```
function GetCommandText: WideString;
```

C++

```
__fastcall BSTR GetCommandText();
```

Description

This is GetCommandText, a member of class TCustomADODataSet.

8.1.9.4.34 TCustomADODataSet.GetDataSource Method

Pascal

```
function GetDataSource: TDataSource; override;
```

C++

```
virtual __fastcall TDataSource GetDataSource();
```

Description

This is GetDataSource, a member of class TCustomADODataSet.

8.1.9.4.35 TCustomADODataSet.GetDetailLinkFields Method

Lists the field components that link this dataset as a detail of a master dataset.

Pascal

```
procedure GetDetailLinkFields(MasterFields: TList; DetailFields: TList); override;
```

C++

```
virtual __fastcall GetDetailLinkFields(TList MasterFields, TList DetailFields);
```

Description

GetDetailLInkFIelds fills the two lists with field components that define a master-detail relationship between this dataset and another (master) dataset. The MasterFields (see page 1219) list is filled with fields from the master table whose values must equal the values of the fields in the DetailFields list. The DetailFields list is filled with fields from this dataset.

Applications seldom call GetDetailLinkFields. It is used internally by the Field Link designer when setting up master/detail relationships.

See Also

DataSource (see page 7014)

MasterSource (see page 1144)

MasterFields (see page 1219)

8.1.9.4.36 GetFieldData Method

This is the overview for the GetFieldData method overload.

Overload List

	Name	Description
	TCustomADODataSet.GetFieldData (Integer, Pointer) (see page 1171)	Retrieves the current value of a field into a buffer.
	TCustomADODataSet.GetFieldData (TField, Pointer) (see page 1171)	Retrieves the current value of a field into a buffer.
	TCustomADODataSet.GetFieldData (TField, Pointer, Boolean) (see page 1171)	Retrieves the current value of a field into a buffer.

8.1.9.4.36.1 TCustomADODataSet.GetFieldData Method (Integer, Pointer)

Retrieves the current value of a field into a buffer.

Pascal

```
function GetFieldData(Field: TField; Buffer: Pointer): Boolean; override;
function GetFieldData(Field: TField; Buffer: Pointer; NativeFormat: Boolean): Boolean;
override;
function GetFieldData(FieldNo: Integer; Buffer: Pointer): Boolean; override; overload;
```

C++

```
virtual __fastcall Boolean GetFieldData(TField Field, void * Buffer);
virtual __fastcall Boolean GetFieldData(TField Field, void * Buffer, Boolean NativeFormat);
virtual __fastcall Boolean GetFieldData(int FieldNo, void * Buffer);
```

Description

Most applications do not need to call GetFieldData. TField objects call this method to implement their GetData method.

The Field or FieldNo parameter indicates the field whose data should be fetched. Field specifies the component itself, while FieldNo indicates its field number.

Buffer is a memory buffer with sufficient space to accept the value of the field as it exists in the database (unformatted and untranslated). GetFieldData returns a value that indicates whether the data was successfully fetched.

GetFieldData returns a boolean value indicating whether the operation was successful.

See Also

GetBlobFieldData (see page 6947)

GetBlobFieldData (see page 1169)

GetData (see page 7131)

8.1.9.4.37 TCustomADODataSet.GetRecNo Method

Pascal

```
function GetRecNo: Integer; override;
```

C++

```
virtual __fastcall int GetRecNo();
```

Description

This is GetRecNo, a member of class TCustomADODataSet.

8.1.9.4.38 TCustomADODataSet.GetRecord Method

Pascal

```
function GetRecord(Buffer: PChar; GetMode: TGetMode; DoCheck: Boolean): TGetResult; override;
```

C++

```
virtual __fastcall TGetResult GetRecord(const char * Buffer, TGetMode GetMode, Boolean
DoCheck);
```

Description

This is GetRecord, a member of class TCustomADODataSet.

8.1.9.4.39 TCustomADODataSet.GetRecordCount Method

Pascal

```
function GetRecordCount: Integer; override;
```

C++

```
virtual __fastcall int GetRecordCount();
```

Description

This is GetRecordCount, a member of class TCustomADODataSet.

8.1.9.4.40 TCustomADODataSet.GetRecordSize Method

Pascal

```
function GetRecordSize: Word; override;
```

C++

```
virtual __fastcall Word GetRecordSize();
```

Description

This is GetRecordSize, a member of class TCustomADODataSet.

8.1.9.4.41 TCustomADODataSet.GetStateFieldValue Method

Pascal

```
function GetStateFieldValue(State: TDataSetState; Field: TField): Variant; override;
```

C++

```
virtual __fastcall Variant GetStateFieldValue(TDataSetState State, TField Field);
```

Description

This is GetStateFieldValue, a member of class TCustomADODataSet.

8.1.9.4.42 TCustomADODataSet.InitOleDBAccess Method

Pascal

```
procedure InitOleDBAccess(Initializing: Boolean);
```

C++

```
__fastcall InitOleDBAccess(Boolean Initializing);
```

Description

This is InitOleDBAccess, a member of class TCustomADODataSet.

8.1.9.4.43 TCustomADODataSet.InternalAddRecord Method

Pascal

```
procedure InternalAddRecord(Buffer: Pointer; Append: Boolean); override;
```

C++

```
virtual __fastcall InternalAddRecord(void * Buffer, Boolean Append);
```

Description

This is InternalAddRecord, a member of class TCustomADODataSet.

8.1.9.4.44 TCustomADODataSet.InternalCancel Method

Pascal

```
procedure InternalCancel; override;
```

C++

```
virtual __fastcall InternalCancel();
```

Description

This is InternalCancel, a member of class TCustomADODataSet.

8.1.9.4.45 TCustomADODataSet.InternalClose Method

Pascal

```
procedure InternalClose; override;
```

C++

```
virtual __fastcall InternalClose();
```

Description

This is InternalClose, a member of class TCustomADODataSet.

8.1.9.4.46 TCustomADODataSet.InternalDelete Method

Pascal

```
procedure InternalDelete; override;
```

C++

```
virtual __fastcall InternalDelete();
```

Description

This is InternalDelete, a member of class TCustomADODataSet.

8.1.9.4.47 TCustomADODataSet.InternalEdit Method

Pascal

```
procedure InternalEdit; override;
```

C++

```
virtual __fastcall InternalEdit();
```

Description

This is InternalEdit, a member of class TCustomADODataSet.

8.1.9.4.48 TCustomADODataSet.InternalFirst Method

Pascal

```
procedure InternalFirst; override;
```

C++

```
virtual __fastcall InternalFirst();
```

Description

This is InternalFirst, a member of class TCustomADODataSet.

8.1.9.4.49 TCustomADODataSet.InternalGetRecord Method

Pascal

```
function InternalGetRecord(Buffer: PChar; GetMode: TGetMode; DoCheck: Boolean): TGetResult;
```

C++

```
__fastcall TGetResult InternalGetRecord(const char * Buffer, TGetMode GetMode, Boolean
DoCheck);
```

Description

This is InternalGetRecord, a member of class TCustomADODataSet.

8.1.9.4.50 TCustomADODataSet.InternalGotoBookmark Method

Pascal

```
procedure InternalGotoBookmark(Bookmark: Pointer); override;
```

C++

```
virtual __fastcall InternalGotoBookmark(void * Bookmark);
```

Description

This is InternalGotoBookmark, a member of class TCustomADODataSet.

8.1.9.4.51 TCustomADODataSet.InternalHandleException Method

Pascal

```
procedure InternalHandleException; override;
```

C++

```
virtual __fastcall InternalHandleException();
```

Description

This is InternalHandleException, a member of class TCustomADODataSet.

8.1.9.4.52 TCustomADODataSet.InternalInitFieldDefs Method

Pascal

```
procedure InternalInitFieldDefs; override;
```

C++

```
virtual __fastcall InternalInitFieldDefs();
```

Description

This is InternalInitFieldDefs, a member of class TCustomADODataSet.

8.1.9.4.53 TCustomADODataSet.InternalInitRecord Method

Pascal

```
procedure InternalInitRecord(Buffer: PChar); override;
```

C++

```
virtual __fastcall InternalInitRecord(const char * Buffer);
```

Description

This is InternalInitRecord, a member of class TCustomADODataSet.

8.1.9.4.54 TCustomADODataSet.InternalInsert Method

Pascal

```
procedure InternalInsert; override;
```

C++

```
virtual __fastcall InternalInsert();
```

Description

This is InternalInsert, a member of class TCustomADODataSet.

8.1.9.4.55 TCustomADODataSet.InternalLast Method

Pascal

```
procedure InternalLast; override;
```

C++

```
virtual __fastcall InternalLast();
```

Description

This is InternalLast, a member of class TCustomADODataSet.

8.1.9.4.56 TCustomADODataSet.InternalOpen Method

Pascal

```
procedure InternalOpen; override;
```

C++

```
virtual __fastcall InternalOpen();
```

Description

This is InternalOpen, a member of class TCustomADODataSet.

8.1.9.4.57 TCustomADODataSet.InternalPost Method

Pascal

```
procedure InternalPost; override;
```

C++

```
virtual __fastcall InternalPost();
```

Description

This is InternalPost, a member of class TCustomADODataSet.

8.1.9.4.58 TCustomADODataSet.InternalRefresh Method

Pascal

```
procedure InternalRefresh; override;
```

C++

```
virtual __fastcall InternalRefresh();
```

Description

This is InternalRefresh, a member of class TCustomADODataSet.

8.1.9.4.59 TCustomADODataSet.InternalRequery Method

Pascal

```
procedure InternalRequery(Options: TExecuteOptions = []);
```

C++

```
__fastcall InternalRequery(TExecuteOptions Options = []);
```

Description

This is InternalRequery, a member of class TCustomADODataSet.

8.1.9.4.60 TCustomADODataSet.InternalSetSort Method

Pascal

```
procedure InternalSetSort(Value: WideString);
```

C++

```
__fastcall InternalSetSort(BSTR Value);
```

Description

This is InternalSetSort, a member of class TCustomADODataSet.

8.1.9.4.61 TCustomADODataSet.InternalSetToRecord Method

Pascal

```
procedure InternalSetToRecord(Buffer: PChar); override;
```

C++

```
virtual __fastcall InternalSetToRecord(const char * Buffer);
```

Description

This is InternalSetToRecord, a member of class TCustomADODataSet.

8.1.9.4.62 TCustomADODataSet.IsCursorOpen Method

Pascal

```
function IsCursorOpen: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCursorOpen();
```

Description

This is IsCursorOpen, a member of class TCustomADODataSet.

8.1.9.4.63 TCustomADODataSet.IsSequenced Method

Indicates whether the underlying database table uses record numbers to indicate the order of records.

Pascal

```
function IsSequenced: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsSequenced();
```

Description

Use IsSequenced to determine whether the underlying database table supports sequence numbers, or whether these are computed by the dataset component. When IsSequenced returns true, applications can safely use the RecNo property to navigate to records in the dataset.

8.1.9.4.64 TCustomADODataSet.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState (see page 2804)

Create (see page 2766)

TReader (see page 2933)

8.1.9.4.65 TCustomADODataSet.LoadFromFile Method

Loads a recordset from a file.

Pascal

```
procedure LoadFromFile(const FileName: WideString);
```

C++

```
__fastcall LoadFromFile(const BSTR FileName);
```

Description

Call LoadFromFile to load the recordset for the calling ADO dataset component from a file. If the loading operation fails, the current recordset is neutralized (set to nil (Delphi) or NULL (C++)), the dataset component remains inactive, and an EOleException exception is raised. If the attempt to load data from a file is successful, the ADO dataset component is automatically activated and the data made available.

FileName is a string containing the name of the file.

LoadFromFile closes the dataset component before loading the recordset from the file specified in FileName.

See Also

EADOError (see page 978)

SaveToFile (see page 1192),

8.1.9.4.66 TCustomADODataSet.Locate Method

Searches the dataset for a specified record and makes that record the current record.

Pascal

```
function Locate(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions):
Boolean; override;
```

C++

```
virtual __fastcall Boolean Locate(const AnsiString KeyFields, const Variant KeyValues,
TLocateOptions Options);
```

Description

Call Locate to search a dataset for a specific row and make it the current row.

KeyFields is a string containing a semicolon-delimited list of field names on which to search.

KeyValues is a variant that specifies the values to match in the key fields. If KeyFields lists a single field, KeyValues specifies the value for that field on the desired row. To specify multiple search values, pass a variant array as KeyValues, or construct a variant array on the fly using the VarArrayOf routine. For example:

```
with ADOTable1 do
  Locate('Company;Contact;Phone', VarArrayOf(['Sight Diver', 'P', '408-431-1000']),
[loPartialKey]);
TLocateOptions Opts;
Opts.Clear();
Opts << loPartialKey;
Variant locvalues[2];
locvalues[0] = Variant("Sight Diver");
locvalues[1] = Variant("P");
CustTable->Locate("Company;Contact", VarArrayOf(locvalues, 1), Opts);
```

Options is a set that optionally specifies additional search latitude when searching on string fields. If Options contains the loCaseInsensitive setting, then Locate ignores case when matching fields. If Options contains the loPartialKey setting, then Locate allows partial-string matching on strings in KeyValues. If Options is an empty set, or if KeyFields does not include any string fields, Options is ignored.

Locate returns true if it finds a matching row, and makes that row the current one. Otherwise Locate returns false.

See Also

FindField (see page 6938)

ADONETDb.TCustomADONETConnector.Lookup

8.1.9.4.67 TCustomADODataSet.LocateRecord Method

Pascal

```
function LocateRecord(const KeyFields: string; const KeyValues: OleVariant; Options:
TLocateOptions; SyncCursor: Boolean): Boolean;
```

C++

```
__fastcall Boolean LocateRecord(const AnsiString KeyFields, const OleVariant KeyValues,
TLocateOptions Options, Boolean SyncCursor);
```

Description

This is LocateRecord, a member of class TCustomADODataSet.

8.1.9.4.68 TCustomADODataSet.Lookup Method

Retrieves field values from a row that matches specified search values.

Pascal

```
function Lookup(const KeyFields: string; const KeyValues: Variant; const ResultFields:
string): Variant; override;
```

C++

```
virtual __fastcall Variant Lookup(const AnsiString KeyFields, const Variant KeyValues, const
AnsiString ResultFields);
```

Description

Call Lookup to retrieve values for specified fields from a row that matches search criteria. KeyFields is a string containing a semicolon-delimited list of field names on which to search.

KeyValues is a variant array containing the values to match in the key fields. To specify multiple search values, pass KeyValues as a variant array as an argument, or construct a variant array on the fly using the VarArrayOf routine.

ResultFields is a string containing a semicolon-delimited list of field names whose values should be returned from the matching row.

Lookup returns a variant array containing the values from the fields specified in ResultFields.

See Also

ADONETDb.TCustomADONETConnector.Locate

8.1.9.4.69 TCustomADODataSet.MasterChanged Method

Pascal

```
procedure MasterChanged(Sender: TObject); virtual;
```

C++

```
virtual __fastcall MasterChanged(TObject * Sender);
```

Description

This is MasterChanged, a member of class TCustomADODataSet.

8.1.9.4.70 TCustomADODataSet.MasterDisabled Method

Pascal

```
procedure MasterDisabled(Sender: TObject); virtual;
```

C++

```
virtual __fastcall MasterDisabled(TObject * Sender);
```

Description

This is MasterDisabled, a member of class TCustomADODataSet.

8.1.9.4.71 TCustomADODataSet.MoveComplete Method

Pascal

```
procedure MoveComplete(adReason: EventReasonEnum; const pError: Error; var adStatus:
```

```
EventStatusEnum; const pRecordset: _Recordset); safecall;
```

C++

```
__fastcall MoveComplete(EventReasonEnum adReason, const Error pError, EventStatusEnum
adStatus, const _Recordset pRecordset);
```

Description

This is MoveComplete, a member of class TCustomADODataSet.

8.1.9.4.72 TCustomADODataSet.NextRecordset Method

Provides access to a secondary recordset.

Pascal

```
function NextRecordset(var RecordsAffected: Integer): _Recordset;
```

C++

```
__fastcall _Recordset NextRecordset(int RecordsAffected);
```

Description

Call NextRecordset in situations where multiple recordsets are returned to provide access to the second (and subsequent) recordsets. The ADO Recordset (see page 1221) object returned by NextRecordset must be assigned to the Recordset (see page 1221) property of an ADO dataset component, such as TADODataSet (see page 1041), to be accessible.

```
ADODataSet1.Recordset := ADOQuery1.NextRecordset
ADODataSet1->Recordset = ADOQuery1->NextRecordset;
```

RecordsAffected is the number of rows in the returned recordset.

8.1.9.4.73 TCustomADODataSet.OpenCursor Method

Pascal

```
procedure OpenCursor(InfoQuery: Boolean); override;
```

C++

```
virtual __fastcall OpenCursor(Boolean InfoQuery);
```

Description

This is OpenCursor, a member of class TCustomADODataSet.

8.1.9.4.74 TCustomADODataSet.PrepareCursor Method

Pascal

```
procedure PrepareCursor; virtual;
```

C++

```
virtual __fastcall PrepareCursor();
```

Description

This is PrepareCursor, a member of class TCustomADODataSet.

8.1.9.4.75 TCustomADODataSet.PSEndTransaction Method

Ends a transaction that was started by a call to PSStartTransaction.

Pascal

```
procedure PSEndTransaction(Commit: Boolean); override;
```

C++

```
virtual __fastcall PSEndTransaction(Boolean Commit);
```

Description

The provider component applies updates within a transaction, if possible. To do so, it uses PSEndTransaction in conjunction with the PSStartTransaction method.

The Commit parameter indicates whether the dataset should commit the current transaction (Commit is true), or roll it back (Commit is false).

Note: TDataSetProvider only starts a transaction (Using PSStartTransaction) if the PSInTransaction method indicates that there is no transaction already established. That is, it does not assume nested transaction support is available. Thus, there is no need to ensure that PSEndTransaction does not terminate a transaction that was not started by the provider.

See Also

PSStartTransaction (see page 7586)

PSInTransaction (see page 7583)

applyupdates (see page 18459)

8.1.9.4.76 TCustomADODataSet.PSExecute Method

Executes the SQL command associated with the dataset.

Pascal

```
procedure PSExecute; override;
```

C++

```
virtual __fastcall PSExecute();
```

Description

The provider component calls PSExecute to implement its Execute method. If the dataset represents a query or stored procedure that does not return a cursor, this method executes the query or stored procedure. If the provider Options includes poAllowCommandText and the dataset supports the PSSetCommandText method, the statement specified by PSSetCommandText may be executed instead.

If there is not statement to execute, PSExecute raises an EDatabaseError exception.

See Also

PSSetCommandText (see page 7585)

Execute (see page 18466)

Options (see page 18440)

8.1.9.4.77 TCustomADODataSet.PSExecuteStatement Method

Executes a specified SQL command.

Pascal

```
function PSExecuteStatement(const ASQL: WideString; AParams: TParams; ResultSet: Pointer =
nil): Integer; override;
```

C++

```
virtual __fastcall int PSExecuteStatement(const BSTR ASQL, TParams AParams, void * ResultSet =
nil);
```

Description

If the value of its ResolveToDataSet property is false, a provider component calls PSExecuteStatement (indirectly) to execute SQL statements that it generates for applying updates. Usually, the dataset passes these statements on to its database server for execution.

The ASQL parameter specifies the SQL command to execute.

AParams contains any parameter values that should be applied to ASQL before execution.

ResultSet allows PSExecuteStatement to return a dataset component that contains the results of a SELECT statement. If ResultSet is not nil (Delphi) or NULL (C++), PSExecuteStatement creates a new dataset component as the target of this pointer and fills it with the result set. The caller is responsible for freeing the dataset.

PSExecuteStatement returns the number of rows affected by executing ASQL.

See Also

PSUpdateRecord (see page 7598)

applyupdates (see page 18459)

ResolveToDataSet (see page 18531)

8.1.9.4.78 TCustomADODataSet.PSGetAttributes Method

Returns information that is stored in data packets as Name/Value pairs.

Pascal

```
procedure PSGetAttributes(List: TList); override;
```

C++

```
virtual __fastcall PSGetAttributes(TList List);
```

Description

The provider component calls PSGetAttributes to add information about the dataset to the metadata stored with generated data packets. This information is stored as Name/Value pairs that are added to the list specified by the List parameter.

See Also

DoGetProviderAttributes (see page 18521)

8.1.9.4.79 TCustomADODataSet.PSGetCommandText Method

Gets the SQL command to be executed by PSExecute.

Pascal

```
function PSGetCommandText: string; override;
```

C++

```
virtual __fastcall AnsiString PSGetCommandText();
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed

by a call to PSExecute.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

PSSetCommandText (see page 7585)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

8.1.9.4.80 TCustomADODataSet.PSGetCommandType Method

Gets the command type to be executed by PSExecute.

Pascal

```
function PSGetCommandType: TPSCommandType; override;
```

C++

```
virtual __fastcall TPSCommandType PSGetCommandType();
```

Description

PSGetCommandType gets the command type executed by PSExecute. The type may be a type of SQL command or other command types, such as stored procedures.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

PSGetCommandText (see page 7579)

Execute (see page 18466)

Options (see page 18440)

8.1.9.4.81 TCustomADODataSet.PSGetDefaultOrder Method

Returns the definition of an index that imposes the default order on the data included in data packets.

Pascal

```
function PSGetDefaultOrder: TIndexDef; override;
```

C++

```
virtual __fastcall TIndexDef PSGetDefaultOrder();
```

Description

The provider component calls PSGetDefaultOrder to generate a default index, called DEFAULT_ORDER, for the records that appear in data packets. This index defines the default sort order for records when they appear in the client dataset on the client.

PSGetDefaultOrder creates a TIndexDef object to describe the default index. The provider that calls PSGetDefaultOrder subsequently frees this TIndexDef object.

See Also

CreateDataPacket (see page 18518)

8.1.9.4.82 TCustomADODataSet.PSGetIndexDefs Method

Returns the definitions of all specified indexes defined for the dataset.

Pascal

```
function PSGetIndexDefs(IndexTypes: TIndexOptions = [ixPrimary..ixNonMaintained]): TIndexDefs;
override;
```

C++

```
virtual __fastcall TIndexDefs PSGetIndexDefs(TIndexOptions IndexTypes =
[ixPrimary..ixNonMaintained]);
```

Description

The provider component calls PSGetIndexDefs to locate an index that contains information it needs. For example, it uses PSGetIndexDefs to locate an index that contains the key fields returned by PSGetKeyFields. This information allows the provider to locate the records that need to change when it applies updates. The provider also calls this method to locate indexes that it adds to the metadata of data packets.

IndexTypes indicates the options required of the returned indexes. PSGetIndexDefs returns all indexes whose Options property includes the specified flags.

PSGetIndexDefs returns a TIndexDefs object that contains all the indexes of the dataset that match the IndexTypes parameter.

See Also

Options (see page 7291)

PSGetKeyFields (see page 7581)

PSGetDefaultOrder (see page 7580)

8.1.9.4.83 TCustomADODataSet.PSGetKeyFieldsW Method

Pascal

```
function PSGetKeyFieldsW: WideString; override;
```

C++

```
virtual __fastcall BSTR PSGetKeyFieldsW();
```

Description

This is PSGetKeyFieldsW, a member of class TCustomADODataSet.

8.1.9.4.84 TCustomADODataSet.PSGetParams Method

Returns the current parameter values of the dataset.

Pascal

```
function PSGetParams: TParams; override;
```

C++

```
virtual __fastcall TParams PSGetParams();
```

Description

The provider component calls PSGetParams to implement its GetParams method.

PSGetParams returns a TParams object that describes the current values of all parameters of the dataset. If the dataset does not have any parameters, PSGetParams returns nil (Delphi) or NULL (C++).

Note: The PSGetParams method should not create a new instance of TParams for the return value. The provider that calls PSGetParams does not free the returned value.

See Also

GetParams (see page 18467)

8.1.9.4.85 TCustomADODataSet.PSGetQuoteCharW Method

Pascal

```
function PSGetQuoteCharW: WideString; override;
```

C++

```
virtual __fastcall BSTR PSGetQuoteCharW();
```

Description

This is PSGetQuoteCharW, a member of class TCustomADODataSet.

8.1.9.4.86 TCustomADODataSet.PSGetTableNameW Method

Pascal

```
function PSGetTableNameW: WideString; override;
```

C++

```
virtual __fastcall BSTR PSGetTableNameW();
```

Description

This is PSGetTableNameW, a member of class TCustomADODataSet.

8.1.9.4.87 TCustomADODataSet.PSGetUpdateException Method

Generates an EUpdateError object based on another exception object.

Pascal

```
function PSGetUpdateException(E: Exception; Prev: EUpdateError): EUpdateError; override;
```

C++

```
virtual __fastcall EUpdateError PSGetUpdateException(Exception E, EUpdateError Prev);
```

Description

The provider component calls PSGetUpdateException when the exception E occurs while applying updates. This method allows the provider to create a single EUpdateError object that encapsulates all the exceptions that occur in an update operation.

PSGetUpdateError determines the error message, context string, and error code from the exception specified by E and creates a new EUpdateError exception that includes that information as well as the information about a previous error as specified by Prev. The context string and error code for E may be specific to the Exception class associated with the dataset. By default, TDataSet generates an EUpdateError with an error code of 1 and an empty context string.

See Also

Create (see page 6733)

8.1.9.4.88 TCustomADODataSet.PSInTransaction Method

Indicates whether actions against the dataset occur in the context of a transaction.

Pascal

```
function PSInTransaction: Boolean; override;
```

C++

```
virtual __fastcall Boolean PSInTransaction();
```

Description

The provider component applies updates within a transaction, if possible. To do so, it first checks whether a transaction is already underway, and if not, generates one using PSStartTransaction. To determine whether a transaction is already underway, the provider calls PSInTransaction.

PSInTransaction returns true if there is a transaction underway, and false otherwise.

See Also

PSStartTransaction (see page 7586)

PSEndTransaction (see page 7578)

8.1.9.4.89 TCustomADODataSet.PSIsSQLBased Method

Indicates whether the dataset includes native SQL support.

Pascal

```
function PSIsSQLBased: Boolean; override;
```

C++

```
virtual __fastcall Boolean PSIsSQLBased();
```

Description

The provider calls PSIsSQLBased to determine whether the dataset supports some form of native SQL. This information is used when generating SQL commands that are executed by PSExecuteStatement.

Even if the dataset is not SQL-based, it may still allow the provider to execute SQL statements for applying updates. For example, BDE-enabled local databases such as Paradox and DBASE provide SQL support via the BDE's local SQL engine, but are not SQL-based. However, table and field names are not specified in the same way for the local SQL engine as they are for native SQL. The provider uses PSIsSQLBased to determine how these names should appear in generated SQL.

Note: The PSIsSQLSupported method indicates whether the dataset can execute SQL commands at all, regardless of whether the dataset is SQL-based.

See Also

PSIsSQLSupported (see page 7584)

PSExecuteStatement (see page 7579)

8.1.9.4.90 TCustomADODataSet.PSIsSQLSupported Method

Indicates whether the dataset can execute SQL statements.

Pascal

```
function PSIsSQLSupported: Boolean; override;
```

C++

```
virtual __fastcall Boolean PSIsSQLSupported();
```

Description

PSIsSQLSupported indicates whether the provider can call PSExecuteStatement to execute an SQL statement. When PSIsSQLSupported returns false, calling PSExecuteStatement generates an exception. The provider can still work with a dataset that returns false from PSIsSQLSupported, but it must set the ResolveToDataSet property to true, or provide data on a read-only basis.

See Also

PSIsSQLBased (see page 7584)

PSExecuteStatement (see page 7579)

ResolveToDataSet (see page 18531)

8.1.9.4.91 TCustomADODataSet.PSReset Method

Resets the dataset so that it is positioned on the first record.

Pascal

```
procedure PSReset; override;
```

C++

```
virtual __fastcall PSReset();
```

Description

The provider calls PSReset to reposition the dataset on the first record before fetching records.

See Also

GetRecords (see page 18468)

AS_GetRecords (see page 16455)

8.1.9.4.92 TCustomADODataSet.PSSetCommandText Method

Assigns a specified SQL command to be executed by PSExecute.

Pascal

```
procedure PSSetCommandText(const CommandText: WideString); override;
```

C++

```
virtual __fastcall PSSetCommandText(const BSTR CommandText);
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed

by a call to PSExecute.

The value of the CommandText parameter replaces the dataset's SQL statement (if it has one), or replaces the name of an underlying database table or stored procedure.

See Also

CommandText (see page 8465)

PSExecute (see page 7590)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

8.1.9.4.93 TCustomADODataSet.PSSetParams Method

Assigns the specified parameter values to the dataset.

Pascal

```
procedure PSSetParams(AParams: TParams); override;
```

C++

```
virtual __fastcall PSSetParams(TParams AParams);
```

Description

PSSetParams assigns the parameter values specified by AParams to the parameters of the dataset.

See Also

Params (see page 8489)

PSGetParams (see page 7582)

8.1.9.4.94 TCustomADODataSet.PSStartTransaction Method

Starts a transaction that encapsulates all activity until a call to PSEndTransaction.

Pascal

```
procedure PSStartTransaction; override;
```

C++

```
virtual __fastcall PSStartTransaction();
```

Description

The provider component applies updates within a transaction, if possible. To do so, it calls PSStartTransaction to start a transaction before applying updates, and calls PSEndTransaction when all updates are applied or too many errors are encountered.

Note: TDataSetProvider only starts a transaction (Using PSStartTransaction) if the PSInTransaction method indicates that there is no transaction already established. That is, it does not assume nested transaction support is available.

See Also

PSEndTransaction (see page 7578)

PSInTransaction (see page 7583)

applyupdates (see page 18459)

8.1.9.4.95 TCustomADODataSet.PSUpdateRecord Method

Applies a single update to the underlying database table or tables.

Pascal

```
function PSUpdateRecord(UpdateKind: TUpdateKind; Delta: TDataSet): Boolean; override;
```

C++

```
virtual __fastcall Boolean PSUpdateRecord(TUpdateKind UpdateKind, TDataSet Delta);
```

Description

Before the provider generates SQL statements to apply an update, it calls PSUpdateRecord to let the dataset apply the update in some other fashion. For example, BDE-enabled datasets generate an OnUpdateRecord to allow the application to apply the update and, if the OnUpdateRecord event handler does not apply the update, uses any associated TUpdateSQL object to apply the update.

UpdateKind indicates whether the update is an insertion (ukInsert), deletion (ukDelete), or modification (ukModify).

Delta is the delta datapacket. The current record represents the record to be updated.

PSUpdateRecord returns true if it updates the record, false otherwise. If PSUpdateRecord returns false, the provider generates dynamic SQL to perform the update and calls PSExecuteStatement.

Note: PSUpdateRecord is only called if the provider's ResolveToDataSet property is false.

See Also

OnUpdateRecord (see page 9957)

TUpdateSQL (see page 10229)

applyupdates (see page 18459)

ResolveToDataSet (see page 18531)

PSExecuteStatement (see page 7579)

8.1.9.4.96 TCustomADODataSet.RecordChangeComplete Method

Pascal

```
procedure RecordChangeComplete(adReason: EventReasonEnum; cRecords: Integer; const pError:
Error; var adStatus: EventStatusEnum; const pRecordset: _Recordset); safecall;
```

C++

```
__fastcall RecordChangeComplete(EventReasonEnum adReason, int cRecords, const Error pError,
EventStatusEnum adStatus, const _Recordset pRecordset);
```

Description

This is RecordChangeComplete, a member of class TCustomADODataSet.

8.1.9.4.97 TCustomADODataSet.RecordsetChangeComplete Method

Pascal

```
procedure RecordsetChangeComplete(adReason: EventReasonEnum; const pError: Error; var
adStatus: EventStatusEnum; const pRecordset: _Recordset); safecall;
```

C++

```
__fastcall RecordsetChangeComplete(EventReasonEnum adReason, const Error pError,
EventStatusEnum adStatus, const _Recordset pRecordset);
```

Description

This is RecordsetChangeComplete, a member of class TCustomADODataSet.

8.1.9.4.98 TCustomADODataSet.RefreshParams Method

Pascal

```
procedure RefreshParams;
```

C++

```
__fastcall RefreshParams();
```

Description

This is RefreshParams, a member of class TCustomADODataSet.

8.1.9.4.99 TCustomADODataSet.ReleaseLock Method

Pascal

```
procedure ReleaseLock;
```

C++

```
__fastcall ReleaseLock();
```

Description

This is ReleaseLock, a member of class TCustomADODataSet.

8.1.9.4.100 TCustomADODataSet.Requery Method

Refreshes the recordset.

Pascal

```
procedure Requery(Options: TExecuteOptions = []);
```

C++

```
__fastcall Requery(TExecuteOptions Options = []);
```

Description

Call Requery to refresh the recordset. Requery updates the dataset by re-executing the original command or SQL statement that produced the recordset.

The effects of Requery are the same as calling the dataset's Close method and then its Open method. However, there are circumstances when one means of refreshing the recordset is better than the other. The values in properties like CursorLocation (see page 1207), CursorType (see page 1208), and LockType (see page 1218) cannot be changed while the dataset is open, so Requery can only refresh the recordset using the existing values in these properties. To refresh the recordset using different values for these properties, effect the refresh using the Close and Open methods.

Options is a TExecuteOptions (see page 1316) value that specifies the characteristics of the command execution affected by the requery that produces the recordset for the dataset.

See Also

ADONETDb.TCustomADONETConnector.LoadFromFile

Close (see page 6920)

Open (see page 6976)

CursorLocation

CursorType

LockType

8.1.9.4.101 TCustomADODataSet.SaveToFile Method

Saves a recordset to a file.

Pascal

```
procedure SaveToFile(const FileName: WideString = ''; Format: TPersistFormat = pfADTG);
```

C++

```
__fastcall SaveToFile(const BSTR FileName = '', TPersistFormat Format = pfADTG);
```

Description

Call SaveToFile to save the current recordset to a file. If the destination file already exists, it is overwritten.

FileName is a string containing the name of the destination file. This file remains open from the first call to SaveToFile until the dataset is closed. This file can be read by other applications while it is open, but they cannot write to the file.

Format specifies the file format for the saved recordset. By default, Format is pfADTG (Advanced Data TableGram format).

Note: Microsoft recommends using a client-side cursor (the dataset is opened with a CursorLocation (see page 1207) property value of clUseClient). This way, if the provider used does not support saving the recordset the client cursor will provide the necessary functionality.

See Also

LoadFromFile (see page 1178)

CursorLocation (see page 1207)

8.1.9.4.102 TCustomADODataSet.Seek Method

Searches for a record using the current index.

Pascal

```
function Seek(const KeyValues: Variant; SeekOption: TSeekOption = soFirstEQ): Boolean;
```

C++

```
__fastcall Boolean Seek(const Variant KeyValues, TSeekOption SeekOption = soFirstEQ);
```

Description

Call Seek to find a record with a specified value (or values) in the field (or fields) on which the current index is based. An index must be active to use Seek. The field used for the search is that of the currently active index. Seek moved the record pointer of the dataset based on the success of the search. If the search finds a record, the record pointer is moved to that record. If Seek does not find a record matching the value in KeyValues using the comparison method specified in SeekOption, the record pointer is positioned to the end of the dataset. Seek returns a boolean value reflecting the success of the search: true if a record

was found or false if no record was found.

```
SuccessVar := ADODataSet1.Seek('Jones', soFirstEQ);
SuccessVar = ADODataSet1->Seek("Jones", soFirstEQ);
```

If the current index is based on more than one field, search values supplied in the Seek method must be for all fields sequentially from left to right in the index expression. That is, if the index is based on three fields, the search can be on the first field, the first and second fields, or on the first and second and third fields. It cannot be based only on the third field or on a combination of the first and third fields. To search based on multiple fields, pass a variant array to the Seek method.

KeyValues is the value to search for or multiple values if the index is based on multiple fields.

```
ADODataSet1.Seek(VarArrayOf([90030, 90020]), soFirstEQ);
ADODataSet1->Seek(VarArrayOf(OPENARRAY(Variant, (90030, 90020))), soFirstEQ);
```

SeekOption specifies the type of comparison made in determining whether the search was successful. SeekOption can be one of the six constants:

Seek Option	Meaning
soFirstEQ	Record pointer positioned at the first matching record, if one is found, or at the end of the dataset if one is not found.
soLastEQ	Record pointer positioned at the last matching record, if one is found, or at the end of the dataset if one is not found.
soAfterEQ	Record pointer positioned at matching record, if found, or just after where that matching record would have been found.
soAfter	Record pointer positioned just after where a matching record would have been found.
soBeforeEQ	Record pointer positioned at matching record, if found, or just before where that matching record would have been found.
soBefore	Record pointer positioned just before where a matching record would have been found.

Using the Seek method depends on the settings of a few properties. IndexName (see page 1217) must be set to activate the index to use, CommandType (see page 1205) must be cmdTableDirect (for TADODataSet (see page 1041), set TableDirect to true for a TADOTable (see page 1125)), CursorLocation (see page 1207) must be clUseServer, and CursorType (see page 1208) must be ctKeySet.

Note: The VCL Seek method is a direct implementation of the Seek method for the ADO Recordset (see page 1221) object. At the time of this writing, this method is only supported for use with Microsoft Access2000 and the Jet 4 provider.

See Also

ADONETDb.TCustomADONETConnector.Locate

ADONETDb.TCustomADONETConnector.IndexFields

IndexName

CommandType

TableDirect (see page 1145)

CursorLocation

CursorType

Using Indexes to Search for Records

8.1.9.4.103 TCustomADODataSet.SetBlockReadSize Method

Pascal

```
procedure SetBlockReadSize(Value: Integer); override;
```

C++

```
virtual __fastcall SetBlockReadSize(int Value);
```

Description

This is SetBlockReadSize, a member of class TCustomADODataSet.

8.1.9.4.104 TCustomADODataSet.SetBookmarkData Method

Pascal

```
procedure SetBookmarkData(Buffer: PChar; Data: Pointer); override;
```

C++

```
virtual __fastcall SetBookmarkData(const char * Buffer, void * Data);
```

Description

This is SetBookmarkData, a member of class TCustomADODataSet.

8.1.9.4.105 TCustomADODataSet.SetBookmarkFlag Method

Pascal

```
procedure SetBookmarkFlag(Buffer: PChar; Value: TBookmarkFlag); override;
```

C++

```
virtual __fastcall SetBookmarkFlag(const char * Buffer, TBookmarkFlag Value);
```

Description

This is SetBookmarkFlag, a member of class TCustomADODataSet.

8.1.9.4.106 TCustomADODataSet.SetCommandText Method

Pascal

```
procedure SetCommandText(const Value: WideString);
```

C++

```
__fastcall SetCommandText(const BSTR Value);
```

Description

This is SetCommandText, a member of class TCustomADODataSet.

8.1.9.4.107 TCustomADODataSet.SetConnection Method

Pascal

```
procedure SetConnection(const Value: TADOConnection); virtual;
```

C++

```
virtual __fastcall SetConnection(const TADOConnection Value);
```

Description

This is SetConnection, a member of class TCustomADODataSet.

8.1.9.4.108 TCustomADODataSet.SetConnectionFlag Method

Pascal

```
function SetConnectionFlag(Flag: Integer; Value: Boolean): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean SetConnectionFlag(int Flag, Boolean Value);
```

Description

This is SetConnectionFlag, a member of class TCustomADODataSet.

8.1.9.4.109 TCustomADODataSet.SetDataSource Method

Pascal

```
procedure SetDataSource(const Value: TDataSource); virtual;
```

C++

```
virtual __fastcall SetDataSource(const TDataSource Value);
```

Description

This is SetDataSource, a member of class TCustomADODataSet.

8.1.9.4.110 TCustomADODataSet.SetDetailFilter Method

Pascal

```
function SetDetailFilter: Boolean;
```

C++

```
__fastcall Boolean SetDetailFilter();
```

Description

This is SetDetailFilter, a member of class TCustomADODataSet.

8.1.9.4.111 SetFieldData Method

This is the overview for the SetFieldData method overload.

Overload List

	Name	Description
	TCustomADODataSet.SetFieldData (TField, Pointer) (see page 1196)	This is SetFieldData, a member of class TCustomADODataSet.
	TCustomADODataSet.SetFieldData (TField, Pointer, Boolean) (see page 1196)	This is SetFieldData, a member of class TCustomADODataSet.

8.1.9.4.111.1 TCustomADODataSet.SetFieldData Method (TField, Pointer)

Pascal

```
procedure SetFieldData(Field: TField; Buffer: Pointer); override;
```

C++

```
virtual __fastcall SetFieldData(TField Field, void * Buffer);
```

Description

This is SetFieldData, a member of class TCustomADODataSet.

8.1.9.4.111.2 TCustomADODataSet.SetFieldData Method (TField, Pointer, Boolean)

Pascal

```
procedure SetFieldData(Field: TField; Buffer: Pointer; NativeFormat: Boolean); override;
```

C++

```
virtual __fastcall SetFieldData(TField Field, void * Buffer, Boolean NativeFormat);
```

Description

This is SetFieldData, a member of class TCustomADODataSet.

8.1.9.4.112 TCustomADODataSet.SetFiltered Method

Pascal

```
procedure SetFiltered(Value: Boolean); override;
```

C++

```
virtual __fastcall SetFiltered(Boolean Value);
```

Description

This is SetFiltered, a member of class TCustomADODataSet.

8.1.9.4.113 TCustomADODataSet.SetFilterOptions Method

Pascal

```
procedure SetFilterOptions(Value: TFilterOptions); override;
```

C++

```
virtual __fastcall SetFilterOptions(TFilterOptions Value);
```

Description

This is SetFilterOptions, a member of class TCustomADODataSet.

8.1.9.4.114 TCustomADODataSet.SetFilterText Method

Pascal

```
procedure SetFilterText(const Value: string); override;
```

C++

```
virtual __fastcall SetFilterText(const AnsiString Value);
```

Description

This is SetFilterText, a member of class TCustomADODataSet.

8.1.9.4.115 TCustomADODataSet.SetParamsFromCursor Method

Pascal

```
procedure SetParamsFromCursor;
```

C++

```
__fastcall SetParamsFromCursor();
```

Description

This is SetParamsFromCursor, a member of class TCustomADODataSet.

8.1.9.4.116 TCustomADODataSet.SetRecNo Method

Pascal

```
procedure SetRecNo(Value: Integer); override;
```

C++

```
virtual __fastcall SetRecNo(int Value);
```

Description

This is SetRecNo, a member of class TCustomADODataSet.

8.1.9.4.117 TCustomADODataSet.Supports Method

Indicates whether a dataset supports certain types of operations.

Pascal

```
function Supports(CursorOptions: TCursorOptions): Boolean;
```

C++

```
__fastcall Boolean Supports(TCursorOptions CursorOptions);
```

Description

Call Supports to determine whether the active recordset supports the cursor operations specified in CursorOptions. Supports will indicate the active recordset, as defined, will support such cursor functionality as adding new rows, using ADO bookmarks, deleting rows, and updating rows. Other cursor functionality can be specified.

CursorOptions specifies one or more TCursorOptions (see page 1314) cursor operations.

If the recordset is active Supports returns a boolean whether the specified cursor operations are supported by the recordset. If all specified cursor operations are supported, Supports returns true, otherwise Supports returns false (one or more of the specified cursor operations are not supported by the recordset).

8.1.9.4.118 TCustomADODataSet.UpdateBatch Method

Writes pending batch updates to disk.

Pascal

```
procedure UpdateBatch(AffectRecords: TAffectRecords = arAll);
```

C++

```
__fastcall UpdateBatch(TAffectRecords AffectRecords = arAll);
```

Description

Call UpdateBatch to write any pending updates to disk for a dataset that is in batch update mode to the associated database.

The AffectRecords parameter may be any one of the following TAffectRecords (see page 1290) values:

Option	Meaning
arCurrent	Only the updates for the current row are applied.
arFiltered	Only updates for rows matching the current filter are applied.
arAll	Updates for all rows are applied.
arAllChapters	Update affects all chapters (ADO chapters)

Note: : To use batch updating, the CursorType (see page 1208) property of the dataset component must be either ctKeySet (the default) or ctStatic and the LockType (see page 1218) property must be ltBatchOptimistic.

See Also

LockType (see page 1218)

Filtered (see page 1214)

RecordStatus (see page 1222)

FilterGroup (see page 1214)

CancelBatch (see page 1161)

CursorType (see page 1208),

8.1.9.4.119 TCustomADODataSet.UpdateIndexDefs Method

Pascal

```
procedure UpdateIndexDefs; override;
```

C++

```
virtual __fastcall UpdateIndexDefs();
```

Description

This is UpdateIndexDefs, a member of class TCustomADODataSet.

8.1.9.4.120 TCustomADODataSet.UpdateRecordSetPosition Method

Pascal

```
procedure UpdateRecordSetPosition(Buffer: PChar);
```

C++

```
__fastcall UpdateRecordSetPosition(const char * Buffer);
```

Description

This is UpdateRecordSetPosition, a member of class TCustomADODataSet.

8.1.9.4.121 TCustomADODataSet.UpdateStatus Method

Indicates the update status of the current record.

Pascal

```
function UpdateStatus: TUpdateStatus; override;
```

C++

```
virtual __fastcall TUpdateStatus UpdateStatus();
```

Description

Call UpdateStatus for datasets opened in batch update mode to determine the update status of the currently active record. UpdateStatus only provides useful information when the dataset has been opened in batch update mode. Otherwise there cannot be any unapplied updates.

Note: : To use batch updating, the CursorType (see page 1208) property of the dataset component must be either ctKeySet (the default) or ctStatic and the LockType (see page 1218) property must be ltBatchOptimistic.

See Also

LockType (see page 1218)

RecordStatus (see page 1222)

DeleteRecords (see page 1165)

UpdateBatch (see page 1197)

CancelBatch (see page 1161)

CursorType (see page 1208),

8.1.9.4.122 TCustomADODataSet.WillChangeField Method

Pascal

```
procedure WillChangeField(cFields: Integer; Fields: OleVariant; var adStatus: EventStatusEnum;
const pRecordset: _Recordset); safecall;
```

C++

```
__fastcall WillChangeField(int cFields, OleVariant Fields, EventStatusEnum adStatus, const
_Recordset pRecordset);
```

Description

This is WillChangeField, a member of class TCustomADODataSet.

8.1.9.4.123 TCustomADODataSet.WillChangeRecord Method

Pascal

```
procedure WillChangeRecord(adReason: EventReasonEnum; cRecords: Integer; var adStatus:
EventStatusEnum; const pRecordset: _Recordset); safecall;
```

C++

```
__fastcall WillChangeRecord(EventReasonEnum adReason, int cRecords, EventStatusEnum adStatus,
const _Recordset pRecordset);
```

Description

This is WillChangeRecord, a member of class TCustomADODataSet.

8.1.9.4.124 TCustomADODataSet.WillChangeRecordset Method

Pascal

```
procedure WillChangeRecordset(adReason: EventReasonEnum; var adStatus: EventStatusEnum; const
pRecordset: _Recordset); safecall;
```

C++

```
__fastcall WillChangeRecordset(EventReasonEnum adReason, EventStatusEnum adStatus, const
_Recordset pRecordset);
```

Description

This is WillChangeRecordset, a member of class TCustomADODataSet.

8.1.9.4.125 TCustomADODataSet.WillMove Method

Pascal

```
procedure WillMove(adReason: EventReasonEnum; var adStatus: EventStatusEnum; const pRecordset:
_Recordset); safecall;
```

C++

```
__fastcall WillMove(EventReasonEnum adReason, EventStatusEnum adStatus, const _Recordset
pRecordset);
```

Description

This is WillMove, a member of class TCustomADODataSet.

8.1.9.5 TCustomADODataSet Properties

The properties of the TCustomADODataSet class are listed here.

Protected Properties

	Name	Description
R	Command (see page 1204)	Provides access to the internal command object that executes commands for the dataset.
	CommandText (see page 1204)	This is CommandText, a member of class TCustomADODataSet.
	CommandTimeout (see page 1204)	Specifies amount of time to attempt execution of a command.
	CommandType (see page 1205)	Specifies the type of command to execute.
	DataSource (see page 1208)	Represents the data source of another dataset that supplies values to the dataset.
	FieldDefs (see page 1210)	Points to the list of field definitions for the dataset.
	IndexDefs (see page 1215)	Contains the definitions of the indexes for the current table.
	IndexFieldNames (see page 1216)	Lists the columns to use for a persistent sort of the recordset.
R	MasterDataLink (see page 1218)	Manages the relationship between the ADO dataset and a master table.
	MasterFields (see page 1219)	Specifies field or fields on which a Master-Detail link is based.
	ParamCheck (see page 1220)	Specifies whether the parameter list for an ADO dataset component is regenerated if the SQL changes at runtime.
	Parameters (see page 1220)	Contains the collection of parameters for an SQL statement.
	Prepared (see page 1221)	Specifies whether the command is prepared before execution.
	StoreDefs (see page 1223)	Specifies whether field definitions are stored with the ADO dataset.

Public Properties

	Name	Description
	DesignerData (see page 1208)	Stores custom data.

	EnableBCD (see page 1209)	Specifies whether to treat numeric fields as floating-point or BCD.
	FilterGroup (see page 1214)	Filters a recordset based on row update status.
R	IndexFieldCount (see page 1216)	Indicates the number of fields that comprise the current key.
	IndexFields (see page 1217)	Retrieves or sets a field for an index.
	IndexName (see page 1217)	Specifies the currently active index.
R	Properties (see page 1221)	This is Properties, a member of class TCustomADODataSet.
	Recordset (see page 1221)	This is Recordset, a member of class TCustomADODataSet.
R	RecordsetState (see page 1222)	This is RecordsetState, a member of class TCustomADODataSet.
R	RecordStatus (see page 1222)	Indicates the status of the current row.
	Sort (see page 1222)	Specifies the sort order of the recordset.

Published Properties

	Name	Description
	Active (see page 1201)	Specifies whether or not a dataset is open.
	AutoCalcFields (see page 1203)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	CacheSize (see page 1203)	Specifies the size of the cache for the dataset.
	Connection (see page 1205)	Specifies the ADO connection component to use.
	ConnectionString (see page 1206)	Specifies the connection information for the data store.
	CursorLocation (see page 1207)	Specifies whether the cursor for the connection is client-side or server-side.
	CursorType (see page 1208)	Specifies type of cursor an ADO dataset uses.
	ExecuteOptions (see page 1209)	Specifies the characteristics of a command execution.
	Filter (see page 1212)	Specifies the text of the current filter for a recordset.
	Filtered (see page 1214)	Specifies whether or not filtering is active for a dataset.
	LockType (see page 1218)	Specifies the lock type to use when opening a dataset.
	MarshalOptions (see page 1218)	Specifies which records are marshaled back to the server.
	MaxRecords (see page 1219)	Specifies how many rows to return in a result set.

8.1.9.5.1 TCustomADODataSet.Active Property

Specifies whether or not a dataset is open.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Use Active to determine or set whether a dataset is populated with data. When Active is false, the dataset is closed; the dataset cannot read or write data and data-aware controls can not use it to fetch data or post edits. When Active is true, the dataset can be populated with data. It can read data from a database or other source (such as a provider). Depending on the CanModify property, active datasets can post changes.

Setting Active to true:

1. Generates a BeforeOpen event.
2. Sets the dataset state to dsBrowse.
3. Establishes a way to fetch data (typically by opening a cursor).
4. Generates an AfterOpen event.

If an error occurs while opening the dataset, dataset state is set to dsInactive, and any cursor is closed.

Setting Active to false:

1. Triggers a BeforeClose event.

2. Sets the State property to dsInactive.
3. Closes the cursor.
4. Triggers an AfterClose event.

An application must set Active to false before changing other properties that affect the status of a database or the controls that display data in an application.

Note: Calling the Open method sets Active to true; calling the Close method sets Active to false.

Delphi Examples:

```
{
This example requires a database with the following name or
in the form of a *.DB file in the following path.  The table
should be closed when changing the database.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Customers.Active := False;
  try
  { First try to use an alias }
    Customers.DatabaseName := 'DBDEMOS';
    Customers.Active := True;
  except
    on EDatabaseError do
    begin
      { If that fails, try to use the drive and directory }
      Customers.TableName := 'Flights';
      Customers.DatabaseName :=
        'c:\Documents and Settings\All Users\Documents\RAD
Studio\5.0\Demos\IntraWeb\Win32\FlightInformation\Data';
      Customers.Active := True;
    end;
  end;
end;
```

See Also

DataSource (see page 7014)

State (see page 7036)

AfterOpen (see page 7045)

BeforeOpen (see page 7051)

Close (see page 6920)

Open (see page 6976)

CanModify (see page 7012)

BeforeClose (see page 7048)

AfterClose (see page 7040)

Using Datasets with Decision Support Components

Making the Table a Detail of Another Dataset

Opening and Closing Datasets

Fetching the Data

8.1.9.5.2 TCustomADODataSet.AutoCalcFields Property

Determines when the OnCalcFields event is triggered and when lookup field values are calculated.

Pascal

```
property AutoCalcFields: Boolean;
```

C++

```
__property Boolean AutoCalcFields;
```

Description

Set AutoCalcFields to control when the OnCalcFields event is triggered to update calculated fields and when lookup fields are calculated.

A calculated field is one that derives its value from the values of one or more fields in the active record, sometimes with additional processing. Lookup fields are fields whose values come from a secondary dataset or lookup cache.

Note: Unidirectional datasets support calculated fields but not unidirectional fields. For unidirectional datasets, OnCalcFields only controls when calculated fields are updated.

When AutoCalcFields is true (the default), Lookup fields are recalculated and OnCalcFields is triggered when:

The dataset is opened.

The dataset is put into dsEdit state.

Focus moves from one visual control to another, or from one column to another in a data-aware grid and modifications have been made to the record.

When AutoCalcFields is false, Lookup fields are recalculated and the OnCalcFields event occurs only when

The dataset is opened.

The dataset is put into dsEdit state.

A record is retrieved from a database.

If an application permits users to change data, OnCalcFields is frequently triggered. In these cases an application may set AutoCalcFields to false to reduce the frequency with which the OnCalcFields event occurs and with which lookup values are fetched.

See Also

OnCalcFields (see page 7053)

Calculating Fields

8.1.9.5.3 TCustomADODataSet.CacheSize Property

Specifies the size of the cache for the dataset.

Pascal

```
property CacheSize: Integer;
```

C++

```
__property int CacheSize;
```

Description

Set CacheSize to control how many rows the ADO dataset's provider keeps cached in its buffer and how many to retrieve at one

time into local memory. Default value of CacheSize is 1 and the minimum allowed value is 1.

For example, if CacheSize is set to 20, when the dataset is first activated the associated provider retrieves the first 20 rows into local memory. As the row pointer is moved through this recordset, the provider retrieves the data from the local memory buffer. If the dataset's row pointer moves beyond the last of these 20 buffered rows, the provider retrieves the next 20.

CacheSize represents the maximum number of rows to retrieve at one time. If the number of rows remaining to be retrieved is less than CacheSize, the remaining rows are retrieved and no exception is raised. The number of rows that can be returned by the provider can be less than all of the rows if the ADO dataset component's MaxRecords (see page 1219) property is set to other than the default (all rows).

Note: The rows retrieved from the cache do not reflect any changes made by other concurrent users of the data.

See Also

BlockReadSize

MaxRecords (see page 1219)

8.1.9.5.4 TCustomADODataSet.Command Property

Provides access to the internal command object that executes commands for the dataset.

Pascal

```
property Command: TADOCommand;
```

C++

```
__property TADOCommand Command;
```

Description

Applications can't use the protected Command property. Command is an internal TADOCommand (see page 984) component that the dataset uses to execute commands against the data store. For example, when the dataset is opened, the ADO dataset uses Command to execute the command that returns the records in the dataset.

See Also

Open (see page 6976)

commandtext (see page 1204)

8.1.9.5.5 TCustomADODataSet.CommandText Property

Pascal

```
property CommandText: WideString;
```

C++

```
__property BSTR CommandText;
```

Description

This is CommandText, a member of class TCustomADODataSet.

8.1.9.5.6 TCustomADODataSet.CommandTimeout Property

Specifies amount of time to attempt execution of a command.

Pascal

```
property CommandTimeout: Integer;
```

C++

```
__property int CommandTimeout;
```

Description

Use CommandTimeout to specify the amount of time that expires before an attempt to execute a command is considered unsuccessful. Measured in seconds, the default value for CommandTimeout is 30.

If a command is successfully executed prior to the expiration of the seconds specified, CommandTimeout has no effect.

See Also

commandtext (see page 1204)

CommandType (see page 1205)

8.1.9.5.7 TCustomADODataSet.CommandType Property

Specifies the type of command to execute.

Pascal

```
property CommandType: TCommandType;
```

C++

```
__property TCommandType CommandType;
```

Description

Set CommandType to indicate the type of command that is contained in the CommandText (see page 1204) property. The value in CommandType should agree with that of CommandText (see page 1204). For instance, if CommandText (see page 1204) is the name of a table, CommandType should be cmdTable or cmdTableDirect.

The default value of CommandType is cmdUnknown. While CommandType may be left as cmdUnknown for all commands, better performance will be realized if CommandType is used to explicitly indicate the type of the command. When the type cmdUnknown is used, ADO must evaluate the command to determine its type, which slows the operation.

```
with ADODataSet1 do begin
  CommandType := cmdText;
  CommandText := 'SELECT * FROM CustomerTable';
  Open;
end;
ADODataSet1->CommandType = cmdText;
ADODataSet1->CommandText = "SELECT * FROM CustomerTable";
ADODataSet1->Open();
```

See Also

commandtext (see page 1204)

Open (see page 6976)

CommandTimeout (see page 1204)

Using TADODataSet

8.1.9.5.8 TCustomADODataSet.Connection Property

Specifies the ADO connection component to use.

Pascal

```
property Connection: TADOConnection;
```

C++

```
__property TADOConnection Connection;
```

Description

Use Connection to specify an ADO connection object to use to connect to an ADO data store.

At design-time, select from the list of available TADOConnection (see page 1001) objects (if any have been added to the application) invoked from the Object Inspector.

At runtime, set Connection to reference an existing TADOConnection (see page 1001) object.

ADOTable1.Connection := ADOConnection1;

ADOTable1->Connection = ADOConnection1;

Note: a connection to an ADO database can be established either using a separate TADOConnection (see page 1001) object (configured to connect to the database) or directly with a valid entry in the ConnectionString (see page 1206) property.

See Also

ConnectionString (see page 1206)

Using Query-type Datasets

Using Stored Procedure-type Datasets

Using Table Type Datasets

Connecting to a Data Store Using TADOConnection

Connecting an ADO Dataset to a Data Store

Using TADODataSet

Connecting Directly to a Database Server

8.1.9.5.9 TCustomADODataSet.ConnectionString Property

Specifies the connection information for the data store.

Pascal

```
property ConnectionString: WideString;
```

C++

```
__property BSTR ConnectionString;
```

Description

Set ConnectionString to specify the information needed to connect the ADO dataset component to the data store. The value used for ConnectionString consists of one or more arguments ADO uses to establish the connection. Specify multiple arguments as a list with individual arguments separated by semicolons.

```
ADOQuery1.ConnectionString := 'Provider=ProviderRef;Remote Server=ServerRef';
ADOQuery1->ConnectionString = "Provider=ProviderRef;Remote Server=ServerRef";
```

At design-time, select from the list of available ADO database connections invoked from the Object Inspector.

A connection string may be saved to file for later use. Specify only the name of such a file in ConnectionString to reuse a saved connection string.

The connection string may also contain login information such as user ID and password, for automated logins.

ADO supports the following four arguments for connection strings. Any other arguments (such as a user ID and password) are

not processed by ADO and simply passed on to the provider.

Argument	Meaning
Provider (see page 18419)	The name of the provider to use for the connection.
File name	The name of a file containing connection information.
Remote Provider (see page 18419)	The name of the provider to use for a client-side connection.
Remote Server	The path name of the server to use for a client-side connection.

Note: After opening a connection, the contents of ConnectionString may be changed by ADO. One example of this behavior is when ADO-defined arguments are mapped to their provider equivalents.

See Also

TADOConnection (see page 1001)

Connection (see page 1205)

ConnectionString (see page 1024)

Using Implicit Connections

Connecting an ADO Dataset to a Data Store

Using TADODataSet

Connecting Directly to a Database Server

8.1.9.5.10 TCustomADODataSet.CursorLocation Property

Specifies whether the cursor for the connection is client-side or server-side.

Pascal

```
property CursorLocation: TCursorLocation;
```

C++

```
__property TCursorLocation CursorLocation;
```

Description

Use CursorLocation to indicate whether the cursors that use the connection object to connect to the ADO datastore use a client-side or server-side cursor library. CursorLocation only affects connections opened after the property is set. The default value for CursorLocation is clUseClient.

A client-side cursor offers more flexibility. All data is retrieved to the local machine and then operated on there, allowing operations not normally supported by servers like sorting and resorting the data and additional filtering. SQL statements are executed at the server, so for statements that restrict the result set with a WHERE clause, only the reduced result set is retrieved to a local cursor.

A server-side cursor offers less flexibility, but may be more advantageous (or necessary) for large result sets. Using a server-side cursor becomes necessary when the sheer size of a result set exceeds the available disk space that would be needed to create the client-side cursor. Also, many servers only support unidirectional cursors. This would preclude moving the record pointer in the dataset backward (even one record) through the result set.

See Also

CursorType (see page 1208)

8.1.9.5.11 TCustomADODataSet.CursorType Property

Specifies type of cursor an ADO dataset uses.

Pascal

```
property CursorType: TCursorType;
```

C++

```
__property TCursorType CursorType;
```

Description

Set CursorType to indicate the type of cursor the ADO dataset uses for the recordset when it is opened. CursorType must be set prior to activating the dataset component. Among other cursor aspects, CursorType affects directional scrolling through a recordset and the visibility of changes made by other users.

The default value of CursorType is ctKeyset.

See Also

CursorLocation (see page 1207)

Opening the Dataset in Batch Update Mode

8.1.9.5.12 TCustomADODataSet.DataSource Property

Represents the data source of another dataset that supplies values to the dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

As implemented in TDataSet, the DataSource property is always nil (Delphi) or NULL (C++).

Descendant dataset classes that support specifying a data source redeclare and implement methods for getting and setting the DataSource property.

See Also

Active (see page 7004)

IsLinkedTo (see page 6967)

Establishing Master/detail Relationships Using Parameters

8.1.9.5.13 TCustomADODataSet.DesignerData Property

Stores custom data.

Pascal

```
property DesignerData: string;
```

C++

```
__property AnsiString DesignerData;
```

Description

Component writers use this property for storing designer-specific information with the TCustomADODataSet (see page 1145) descendent component. The VCL system does not use this property or its value at all. Use must be implemented by the programmer.

8.1.9.5.14 TCustomADODataSet.EnableBCD Property

Specifies whether to treat numeric fields as floating-point or BCD.

Pascal

```
property EnableBCD: Boolean;
```

C++

```
__property Boolean EnableBCD;
```

Description

Use the EnableBCD property to specify how fields are mapped to field classes. If EnableBCD is true, (the default value) adDecimal and adNumeric fields are mapped to the TBCDField class when field objects are created. If EnableBCD is false, the fields are mapped to the TFloatField class.

EnableBCD determines whether numeric and decimal fields are translated as floating-point values or binary coded decimal (BCD) values. BCD values eliminate the rounding errors associated with floating point math (such as a 3 * (2/3) resulting in 2.00000000001).

If persistent field objects are used (such as those created using the Fields Editor), the field class associated with a given field can contradict the setting of EnableBCD. Three numeric fields in the same table could, for instance, each be represented by a TFloatField, a TBCDField, and a TVariantField – regardless of the value in EnableBCD. Select field classes on a field-by-field basis for greater flexibility.

Note: For numeric values with more than 4 digits to the right of the decimal place, use of TFloatField is generally better. This is because TBCDField uses the currency data type that has a fixed scale of 4 decimal places.

Note: For fields with very large numbers of more than 19 significant digits, you can use TVariantField type persistent field objects. The TFloatField and TBCDField classes lack sufficient capacity for fields of this size. TVariantField allows the getting and setting of the field data as strings, preventing without loss of data due to insufficient capacity. However, arithmetic operations cannot be performed on numbers accessed through a TVariantField object.

See Also

TBCDField (see page 6776)

TFloatField (see page 7229)

8.1.9.5.15 TCustomADODataSet.ExecuteOptions Property

Specifies the characteristics of a command execution.

Pascal

```
property ExecuteOptions: TExecuteOptions;
```

C++

```
__property TExecuteOptions ExecuteOptions;
```

Description

Set ExecuteOptions to control the characteristics of a command (a data retrieval operation) executed from the ADO dataset

component.

See Also

Fetching Records Asynchronously

8.1.9.5.16 TCustomADODataSet.FieldDefs Property

Points to the list of field definitions for the dataset.

Pascal

```
property FieldDefs: TFieldDefs;
```

C++

```
__property TFieldDefs FieldDefs;
```

Description

FieldDefs lists the field definitions for a dataset. While an application can examine FieldDefs to explore the field definitions for a dataset, it should not change these definitions unless creating a new table with CreateTable or CreateDataSet.

To access fields and field values in a dataset, use the Fields, AggFields, and FieldValues properties, and the FieldsByName method.

Note: If the dataset includes object field descendants, FieldDefs represents a hierarchical view of the data, meaning that the definitions include object field definitions. To determine the definitions in a flattened view, use FieldDefList instead.

C++ Examples:

```
/*
The following code creates and activates a client dataset in
the form's OnCreate event handler.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TFieldDefs *pDefs = CDS2->FieldDefs;
  TFieldDef *pDef = pDefs->AddFieldDef();
  pDef->DataType = ftInteger;
  pDef->Name = "Field1";

  pDef = pDefs->AddFieldDef();
  pDef->DataType = ftString;
  pDef->Size = 10;
  pDef->Name = "Field2";

  TIndexDef *pIDef = CDS2->IndexDefs->AddIndexDef();
  pIDef->Fields = "Field1";
  pIDef->Name = "IntIndex";

  CDS2->CreateDataSet();
}
```

Delphi Examples:

```
{
The following code creates and activates a client dataset in
the form's OnCreate event handler.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  with CDS2 do
  begin
```

```
      with FieldDefs.AddFieldDef do
      begin
        DataType := ftInteger;
        Name := 'Field1';
      end;
      with FieldDefs.AddFieldDef do
      begin
        DataType := ftString;
        Size := 10;
        Name := 'Field2';
      end;
      with IndexDefs.AddIndexDef do
      begin
        Fields := 'Field1';
        Name := 'IntIndex';
      end;
      CreateDataSet;
    end;
  end;
  { The following example shows how to create a table. }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Table1.Close;
    Table1.DeleteTable;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    Table1:= TTable.Create(Form1);
    with Table1 do
    begin
      DatabaseName := 'DBDEMOS';
      TableType := ttParadox;
      TableName := 'CustInfo';
      Table1.Active := False;
      { Don't overwrite an existing table }
      if Table1.Exists then
        MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
      else
      begin
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
        { Next, describe the fields in the table }
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
```

```
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.Active:= True;
  end;
```

See Also

CreateTable (see page 10189)

Fields (see page 7023)

FieldValues (see page 7025)

ClearFields (see page 6919)

GetFieldList (see page 6953)

GetFieldNames (see page 6954)

FieldByName (see page 6936)

FindField (see page 6938)

SetFields (see page 6997)

ObjectView (see page 7031)

AggFields (see page 7006)

CreateDataSet (see page 8401)

Creating and Deleting Tables

8.1.9.5.17 TCustomADODataSet.Filter Property

Specifies the text of the current filter for a recordset.

Pascal

```
property Filter: string;
```

C++

```
__property AnsiString Filter;
```

Description

Use Filter to specify a recordset filter. When filtering is applied to a recordset by setting the Filtered (see page 1214) property to true, only those records that meet a filter's conditions are available to the application. Filter contains the expression used to evaluate the rows in the recordset. At design-time, enter the filter expression into the editing cell for the Filter property within the Object Inspector. At runtime, assign a string containing the filter expression to the Filter property.

The filter expression below displays only those rows where the State field is "CA" or "MA":

```
State = 'CA' OR State = 'MA'
```

The following runtime example shows how to assign that filter expression to the Filter property and activate the filtering.

```
with ADODataSet1 do begin
  Filtered := False;
  Filter := 'State = ' + QuotedStr('CA') + ' OR ' +
    'State = ' + QuotedStr('CA');
  Filtered := True;
end;
ADODataSet1->Filtered = false;
ADODataSet1->Filter = "State = " + QuotedStr("CA") + " OR " + "State = " + QuotedStr("CA");
ADODataSet1->Filtered = true;
```

When a filter is set, Blank records do not appear unless explicitly included in the filter. To include rows based on blank, (or "NULL") values in table columns, compare the column to NULL in the expression. For example:

```
with ADODataSet1 do begin
  Filtered := False;
  Filter := 'State = ' + QuotedStr('CA') + ' OR ' +
    'State = NULL';
  Filtered := True;
end;
ADODataSet1->Filtered = false;
ADODataSet1->Filter = "State = " + QuotedStr("CA") + " OR " + "State = NULL";
ADODataSet1->Filtered = true;
```

When a field name contains spaces, you must enclose the field name in brackets. For example:

```
[Home State] = 'CA' or [Home State] = 'MA'
```

Filter expressions on remote SQL tables and on client datasets support field comparisons. For example:

```
Field1 > Field2
```

The FilterOptions property controls case sensitivity and filtering on partial comparisons.

Tip: Applications can set Filter at runtime to change the filtering condition for a dataset at (for example, in response to user input).

Note: : include a space between comparison values and comparison operators in filter expressions. For instance, ensure that there is a space after the field name and before the operator.

See Also

Filtered (see page 1214)

FilterOptions (see page 7027)

OnFilterRecord (see page 7055)

Locate (see page 1179)

Lookup (see page 1180)

8.1.9.5.18 TCustomADODataSet.Filtered Property

Specifies whether or not filtering is active for a dataset.

Pascal

```
property Filtered: Boolean;
```

C++

```
__property Boolean Filtered;
```

Description

Check Filtered to determine whether or not dataset filtering is in effect. If Filtered is true, then filtering is active.

To apply filter conditions specified in the Filter (see page 1212) property or the OnFilterRecord (see page 1244) event handler, set Filtered to true.

Note: When filtering is enabled, user edits to a record may mean that the record no longer meets a filter's test condition. The next time an attempt is made to the read the record from the dataset while the filter is in effect, the record is not visible. If that happens, the next record that passes the filter condition becomes the active record.

See Also

Filter (see page 1212)

FilterOptions (see page 7027)

OnFilterRecord (see page 7055)

Locate (see page 1179)

Lookup (see page 1180)

FilterGroup (see page 1214)

8.1.9.5.19 TCustomADODataSet.FilterGroup Property

Filters a recordset based on row update status.

Pascal

```
property FilterGroup: TFilterGroup;
```

C++

```
__property TFilterGroup FilterGroup;
```

Description

Use FilterGroup to filter the recordset displayed with an ADO dataset component. The filter requires the dataset be in update batch mode and the rows are filtered based on the update status of individual rows. Set FilterGroup to a TFilterGroup (see page 1299) constant to filter the dataset to show just the rows that match that batch update status. For instance, set FilterGroup to fgPendingRecords to show just the rows that have been modified since the last update application.

```
if (ADODataSet1.LockType = ltBatchOptimistic) then begin
  ADODataSet1.Filtered := True;
  ADODataSet1.FilterGroup := fgPendingRecords;
end;
if (ADODataSet1->LockType == ltBatchOptimistic)
{
  ADODataSet1->Filtered = true;
  ADODataSet1->FilterGroup = fgPendingRecords;
};
```

To use FilterGroup to filter based on the update status of rows, ensure that the LockType (see page 1218) property is set to ltBatchOptimistic (a prerequisite for batch updating) and the Filtered (see page 1214) property is set to true. In a filtered dataset, check the update status of individual rows using the RecordStatus (see page 1222) property.

The TFilterGroup (see page 1299) type consists of the following constants:

Filter (see page 1212) group type	Meaning
fgUnassigned	Specifies that no filtering is in effect. This constant is used internally by the ADO dataset component.
fgNone	Removes any current filtering and all rows are visible. Can also be done by setting the Filtered (see page 1214) property to false.
fgPendingRecords	Filters to show just the rows that have been changed and the changes have not been applied (UpdateBatch (see page 1197) method) or canceled (CancelBatch (see page 1161)).
fgAffectedRecords	Filters to show just the rows affected by the last update.
fgFetchedRecords	Filters to show just the rows in the current update cache. This is the results of the last call to retrieve rows from the database.
fgPredicate	Filters to show just deleted rows.
fgConflictingRecords	Filters to show just the rows that had changes made to them, but could not be applied due to errors on the apply attempt.

See Recordset.Filter (see page 1212) property

See Also

LockType (see page 1218)

Filtered (see page 1214)

RecordStatus (see page 1222)

UpdateBatch (see page 1197)

CancelBatch (see page 1161)

Filtering Multiple Rows Based On Update Status

8.1.9.5.20 TCustomADODataSet.IndexDefs Property

Contains the definitions of the indexes for the current table.

Pascal

```
property IndexDefs: TIndexDefs;
```

C++

```
__property TIndexDefs IndexDefs;
```

Description

IndexDefs contains definitions for each index the current table has. IndexDefs is a collection of TIndexDef objects. Each TIndexDef object contains the definition for one index.

Inspect IndexDefs at runtime to get information on a given index. Use an index with IndexDefs to address a specific TIndexDef object. Use properties of TIndexDef to evaluate various index attributes. The example below retrieves the fields list for the third index of the table accessed through the component ADOTable1.

```
ADOTable1.IndexDefs.Update;
```

```
StrVar := ADOTable1.IndexDefs[2].Fields;
ADOTable1->IndexDefs->Update();
StrVar = ADOTable1->IndexDefs[2] ->Fields;
```

Call the Update method (TIndexDefs) prior to inspecting the property to ensure it contains the most current index information for the table.

See Also

TIndexDef (see page 7283)

TIndexDefs (see page 7292)

Update (see page 7299)

8.1.9.5.21 TCustomADODataSet.IndexFieldCount Property

Indicates the number of fields that comprise the current key.

Pascal

```
property IndexFieldCount: Integer;
```

C++

```
__property int IndexFieldCount;
```

Description

Examine IndexFieldCount to determine the number of fields that comprise the current key. For indexes based on a single column, IndexFieldCount returns 1. For multi-column indexes, IndexFieldCount indicates the number of fields upon which the index is based.

See Also

IndexFields (see page 1217)

IndexName (see page 1217)

8.1.9.5.22 TCustomADODataSet.IndexFieldNames Property

Lists the columns to use for a persistent sort of the recordset.

Pascal

```
property IndexFieldNames: string;
```

C++

```
__property AnsiString IndexFieldNames;
```

Description

Use IndexFieldNames as an alternative method of specifying the sort order of a recordset. IndexFieldNames is a direct implementation of the Sort (see page 1222) property of the ADO recordset object, with the field list from IndexFieldNames being used for the Sort (see page 1222) property. The actual sorting of data occurs on the OLE DB (see page 6725) side and use of indexes for the sort depend on the particular provider.

To use IndexFieldNames to establish a recordset's sort order, the recordset must have been opened using a client-side cursor (the dataset component's CursorLocation (see page 1207) property must be clUseClient). An exception is raised on attempts to use IndexFieldNames with server-side cursors.

IndexFieldNames may be used to set the sort order for the detail table in a master-detail link, increasing the performance of the link. However, use of IndexFieldNames is not required for master-detail links.

See Also

IndexName (see page 1217)

Creating an Index with IndexFieldNames

8.1.9.5.23 TCustomADODataSet.IndexFields Property

Retrieves or sets a field for an index.

Pascal

```
property IndexFields [Index: Integer]: TField;
```

C++

```
__property TField IndexFields[int Index];
```

Description

IndexFields provides a zero-based array of field objects, each of which corresponds to a field in the current index. Index is an ordinal value indicating the position of a field in the index. The first field in the index is IndexFields[0], the second is IndexFields[1], and so on.

See Also

IndexFieldCount (see page 1216)

IndexName (see page 1217)

8.1.9.5.24 TCustomADODataSet.IndexName Property

Specifies the currently active index.

Pascal

```
property IndexName: WideString;
```

C++

```
__property BSTR IndexName;
```

Description

Use IndexName to activate an index and cause it to actively order the dataset's rows. At runtime, set IndexName to a string containing the name of the index. At design-time, select the desired index from the drop-down list in the Object Inspector.

```
ADOTable1.IndexName := 'LastName';
ADOTable1->IndexName = "LastName";
```

Set IndexName to an empty string to have no index actively ordering the dataset. The rows then appear in natural order. In some database systems, the primary index of a table does not have a name. In these cases, setting IndexName to an empty string activates the primary index.

An index must be active for certain index dependent operations to be used, such as the Seek (see page 1192) method.

Note: Many providers do not support changing indexes while the dataset is active, raising an exception when an attempt is made to do this. Ensure that the dataset is closed before changing the index through the IndexName property and reopen the dataset after specifying a new index.

See Also

Seek (see page 1192)

IndexDefs (see page 1215)

IndexFieldNames (see page 1216)

IndexFieldCount (see page 1216)

8.1.9.5.25 TCustomADODataSet.LockType Property

Specifies the lock type to use when opening a dataset.

Pascal

```
property LockType: TADOLockType;
```

C++

```
__property TADOLockType LockType;
```

Description

Use LockType to specify the lock type to use when opening a dataset. LockType determines whether the row locking scheme for a recordset will be pessimistic, optimistic, or read-only. To be effective, LockType must be set prior to activating the ADO dataset component.

The default value of LockType is ltOptimistic.

Use the LockType ltBatchOptimistic to open a dataset in batch update mode.

See Also

FilterGroup (see page 1214)

8.1.9.5.26 TCustomADODataSet.MarshalOptions Property

Specifies which records are marshaled back to the server.

Pascal

```
property MarshalOptions: TMarshalOption;
```

C++

```
__property TMarshalOption MarshalOptions;
```

Description

Use MarshalOptions when the dataset component uses a client-side cursor and data changes are to be sent back to the server. MarshalOptions specifies which records in the local recordset are sent to the middle tier or server. A value of moMarshalAll results in all records in the local recordset being packaged and sent back. A value of moMarshalModifiedOnly sends back only those records that have actually been changed in the local recordset The default value for MarshalOptions is moMarshalAll.

8.1.9.5.27 TCustomADODataSet.MasterDataLink Property

Manages the relationship between the ADO dataset and a master table.

Pascal

```
property MasterDataLink: TMasterDataLink;
```

C++

```
__property TMasterDataLink MasterDataLink;
```

Description

Applications can't use the protected MasterDataLink property. It is used internally to handle the coordination of this dataset with the dataset that is associated with the DataSource (see page 1208) property.

See Also

MasterSource (see page 1144)

DataSource (see page 7014)

MasterFields (see page 1219)

8.1.9.5.28 TCustomADODataSet.MasterFields Property

Specifies field or fields on which a Master-Detail link is based.

Pascal

```
property MasterFields: WideString;
```

C++

```
__property BSTR MasterFields;
```

Description

Use MasterFields after setting the MasterSource or DataSource (see page 1208) property to specify the names of one or more fields in another, master, dataset that are used to establish a Master-Detail relationship between this dataset and the master dataset. The master dataset is specified by assigning its data source to the MasterSource or DataSource (see page 1208) property.

MasterFields is a string containing one or more field names in the master dataset. When the datasets are related based on two fields, separate field names with semicolons:

```
ADOTable2.MasterSource := DataSource1;
ADOTable2.MasterFields := 'CustID;SaleDate'
ADOTable2->MasterSource = DataSource1;
ADOTable2->MasterFields = "CustID;SaleDate"
```

Each time the current record in the master dataset changes, the new values in those fields are used to select corresponding records in this table for display.

Note: At design time, use the Field Link designer to establish the master-detail relationship between two tables.

See Also

MasterSource (see page 1144)

DataSource (see page 7014)

Making the Table a Detail of Another Dataset

8.1.9.5.29 TCustomADODataSet.MaxRecords Property

Specifies how many rows to return in a result set.

Pascal

```
property MaxRecords: Integer;
```

C++

```
__property int MaxRecords;
```

Description

Use MaxRecords to control the number of rows the provider for the ADO dataset component returns from the data source. Set MaxRecords to indicate the maximum number of rows. The default value for MaxRecords is 0 (zero), which places no limits on the result set.

See Also

CacheSize (see page 1203)

8.1.9.5.30 TCustomADODataSet.ParamCheck Property

Specifies whether the parameter list for an ADO dataset component is regenerated if the SQL changes at runtime.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

Set ParamCheck to specify whether or not the Parameters (see page 1220) property is initialized using ":Param" style parameters in an SQL statement specified in the CommandText (see page 1204) property. If the SQL statement in CommandText (see page 1204) only contains "?" style parameters, the Parameters (see page 1220) property will only be initialized if the OLE DB (see page 6725) provider returns parameter information for an SQL command. Otherwise, the parameters must be explicitly created using the CreateParameter method (TParameter (see page 1250)).

When ParamCheck is true, a parameter is generated for each ":Param" style parameter in the SQL statement.

The default value of ParamCheck is true.

This property is useful for data definition language (DDL) statements that contain parameters as part of the DDL statement and that are not parameters for the ADO dataset component. For example, the DDL statement to create a stored procedure may contain parameter statements that are part of the stored procedure. Set ParamCheck to false to prevent these parameters from being mistaken for parameters of the ADO dataset component executing the DDL statement.

See Also

TADOQuery (see page 1084)

TADOStoredProc (see page 1106)

Parameters (see page 1220)

Open (see page 6976)

commandtext (see page 1204)

8.1.9.5.31 TCustomADODataSet.Parameters Property

Contains the collection of parameters for an SQL statement.

Pascal

```
property Parameters: TParameters;
```

C++

```
__property TParameters Parameters;
```

Description

At design-time, use Parameters to access the parameters for the SQL statement used by the ADO component. Use the properties and methods of Parameters to set and get individual parameter values or to set or inspect parameter attributes.

At runtime, use the Parameters property editor to access the collection of parameters.

See Also

SQL (see page 1105)

ExecSQL (see page 1100)

8.1.9.5.32 TCustomADODataSet.Prepared Property

Specifies whether the command is prepared before execution.

Pascal

```
property Prepared: Boolean;
```

C++

```
__property Boolean Prepared;
```

Description

Set Prepared before calling the Open method to specify whether ADO prepares the command used to create the dataset's recordset. If Prepared is set to true and the dataset component is connected to a data store, ADO prepares the command before executing it. If Prepared is set to false, ADO does not prepare the command.

The default value of Prepared is false.

See Also

Parameters (see page 1220)

ExecSQL (see page 1100)

Open (see page 6976)

8.1.9.5.33 TCustomADODataSet.Properties Property

Pascal

```
property Properties: Properties;
```

C++

```
__property Properties Properties;
```

Description

This is Properties, a member of class TCustomADODataSet.

8.1.9.5.34 TCustomADODataSet.Recordset Property

Pascal

```
property Recordset: _Recordset;
```

C++

```
__property _Recordset Recordset;
```

Description

This is Recordset, a member of class TCustomADODataSet.

8.1.9.5.35 TCustomADODataSet.RecordsetState Property

Pascal

```
property RecordsetState: TObjectStates;
```

C++

```
__property TObjectStates RecordsetState;
```

Description

This is RecordsetState, a member of class TCustomADODataSet.

8.1.9.5.36 TCustomADODataSet.RecordStatus Property

Indicates the status of the current row.

Pascal

```
property RecordStatus: TRecordStatusSet;
```

C++

```
__property TRecordStatusSet RecordStatus;
```

Description

Read RecordStatus to determine the status of the current row relative to batch updates or other bulk operations. Among other record state aspects, RecordStatus indicates whether the current record is new, modified, deleted, or unmodified. RecordStatus will also give an indication why a row is not saved when modifying, adding, or deleting data.

Note: RecordStatus only provides meaningful information if the dataset is opened in batch update mode.

See Also

LockType (see page 1218)

Filtered (see page 1214)

FilterGroup (see page 1214)

UpdateBatch (see page 1197)

CancelBatch (see page 1161)

Inspecting the Update Status of Individual Rows

8.1.9.5.37 TCustomADODataSet.Sort Property

Specifies the sort order of the recordset.

Pascal

```
property Sort: WideString;
```

C++

```
__property BSTR Sort;
```

Description

Set Sort to establish or change the list of fields on which the recordset is sorted. Set sort to the name of a single field or to a comma-separated list of fields. Each field may optionally be followed by the keyword ASCENDING or DESCENDING to specify a sort direction for the field. If one of these keywords is not used, the default sort direction for the field is ascending. Set Sort to an

empty string to reset the recordset to the sort order originally used when the recordset's data was first retrieved.

```
ADOQuery1.Sort := 'LastName ASC, DateDue DESC'
ADOQuery1->Sort = "LastName ASC, DateDue DESC"
```

Read Sort to determine the field (or fields) on which the recordset is sorted.

Note: If the cursor is client-side (the dataset component's CursorLocation (see page 1207) property or that of an associated TADOConnection (see page 1001) component is clUseClient) and no index already exists matching the requested field sort order, a temporary index is created. Resetting the sort order by setting Sort to an empty string automatically deletes the temporary index.

See Also

CursorLocation

CursorLocation (see page 1026)

Using TADODataSet

8.1.9.5.38 TCustomADODataSet.StoreDefs Property

Specifies whether field definitions are stored with the ADO dataset.

Pascal

```
property StoreDefs: Boolean;
```

C++

```
__property Boolean StoreDefs;
```

Description

StoreDefs controls whether the FieldDefs (see page 1210) property value is saved with the ADO dataset when it is saved to a form (.dfm) file. By default, StoreDefs is false unless you explicitly change any of the field definitions.

See Also

FieldDefs (see page 7019)

8.1.9.6 TCustomADODataSet Events

The events of the TCustomADODataSet class are listed here.

Published Events

	Name	Description
	AfterCancel (see page 1224)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 1226)	Occurs after an application closes a dataset.
	AfterDelete (see page 1227)	Occurs after an application deletes a record.
	AfterEdit (see page 1230)	Occurs after an application starts editing a record.
	AfterInsert (see page 1231)	Occurs after an application inserts a new record.
	AfterOpen (see page 1231)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 1233)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 1233)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 1233)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 1234)	Occurs before an application executes a request to cancel changes to the active record.

	BeforeClose (see page 1234)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 1235)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 1235)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 1236)	Occurs before an application enters insert mode.
	BeforeOpen (see page 1237)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 1238)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 1239)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 1239)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 1239)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 1240)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 1241)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnEndOfRecordset (see page 1241)	Occurs when moving a recordset's row pointer past the last row.
	OnFetchComplete (see page 1242)	Occurs after records have been retrieved into the recordset.
	OnFetchProgress (see page 1242)	Occurs periodically during an asynchronous data retrieval operation.
	OnFieldChangeComplete (see page 1243)	Occurs after a field has changed.
	OnFilterRecord (see page 1244)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnMoveComplete (see page 1244)	Occurs after the row pointer in a recordset has moved.
	OnNewRecord (see page 1245)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 1245)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnRecordChangeComplete (see page 1246)	Occurs after one or more records in the recordset change.
	OnRecordsetChangeComplete (see page 1247)	Occurs after the recordset has changed.
	OnRecordsetCreate (see page 1247)	Occurs when the recordset has been initialized and is ready to use.
	OnWillChangeField (see page 1248)	Occurs before a field is changed.
	OnWillChangeRecord (see page 1248)	Occurs before a record in the recordset is changed.
	OnWillChangeRecordset (see page 1249)	Occurs before the recordset has changed.
	OnWillMove (see page 1250)	Occurs before the row pointer in a recordset is moved.

8.1.9.6.1 TCustomADODataSet.AfterCancel Event

Occurs after an application completes a request to cancel modifications to the active record.

Pascal

```
property AfterCancel: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterCancel;
```

Description

Write an AfterCancel event handler to take specific action after an application cancels changes to the active record. AfterCancel is called by the Cancel method after it updates the current position, releases the lock on the active record if necessary, and sets the dataset state to dsBrowse. If an application requires additional processing before returning control to a user after a Cancel event, code it in the AfterCancel event.

Delphi Examples:

```
{
This example updates the form's status bar with a message
when an AfterCancel event occurs.  Set the StatusBar
SimplePanel property to True.
}
procedure TForm1.Table1AfterCancel(DataSet: TDataSet);
begin
  StatusBar1.SimpleText := 'Record changes cancelled for ' + DataSet.Name;
```

```
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Table1.Cancel;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  Table1.Post
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Name := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
    begin
        Table1.Close;
        Table1.DeleteTable;
    end;
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
```

```
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.AfterCancel:= Table1AfterCancel;
    Table1.Active:= True;
  end;
```

See Also

BeforeCancel (see page 7048)

Cancel (see page 6916)

State (see page 7036)

8.1.9.6.2 TCustomADODataSet.AfterClose Event

Occurs after an application closes a dataset.

Pascal

```
property AfterClose: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterClose;
```

Description

Write an AfterClose event handler to take specific action immediately after an application closes a dataset. For example, as a security measure, an application might clear a PASSWORD entry from the Params property of a TSQLConnection component when the dataset is closed.

AfterClose is called after a dataset is closed and the dataset state is set to dsInactive.

Delphi Examples:

```
{
This example adds an entry to a memo with a message when an
AfterOpen event or an AfterClose event occurs.  Populate the
ClientDataSet events with CheckButtonActive.
}
procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  if (Button1.Enabled) then
    Memo2.Lines.Add('Now that the dataset is open, record information is available.')
  else
    Memo2.Lines.Add('Now that the dataset is closed, record information is not available.');
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
  RemoteServer1.Connected := False;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].DisableControls;
  end;
end;

procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
```

See Also

Active (see page 7004)

State (see page 7036)

BeforeClose (see page 7048)

Close (see page 6920)

Params (see page 20253)

Opening and Closing Datasets

8.1.9.6.3 TCustomADODataSet.AfterDelete Event

Occurs after an application deletes a record.

Pascal

```
property AfterDelete: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterDelete;
```

Description

Write an AfterDelete event handler to take specific action immediately after an application deletes the active record in a dataset. AfterDelete is called by Delete after it deletes the record, sets the dataset state to dsBrowse, and repositions the current record.

C++ Examples:

```
/*
This example displays a message on the form's status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
*/
void __fastcall TForm1::Table1AfterDelete(TDataSet *DataSet)
{
  StatusBar1->SimpleText = Format(
    "There are now %d records in the table",
    ARRAYOFCONST(((int)DataSet->RecordCount)));
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Table1->Delete();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Table1 = new TTable(Form1);
  Table1->Active = false; // The Table component must not be active
  Table1->DatabaseName = "DBDEMOS";
  Table1->TableType = ttParadox;
  Table1->TableName = "CustInfo";
  Table1->Active = False;
  if (!Table1->Exists) // Don't overwrite an existing table
  {
    Table1->Close();
    Table1->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Table1->FieldDefs->Clear();
  TFieldDef *pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Table1->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Table1->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Table1->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Table1->CreateTable();
  Table1->Active = True;
  for (int i = 1; i <= 20; i++)
    Table1->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  DS2->DataSet = Table1;
  DBGrid2->DataSource->DataSet = Table1;
  Table1->AfterDelete = Table1AfterDelete;
  Table1->Active = True;
}
```

Delphi Examples:

```
{
This example displays a message on the form's status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
```

```
}
procedure TForm1.Table1AfterDelete(DataSet: TDataSet);
begin
  StatusBar1.SimpleText := Format('There are now %d records in the table',
[DataSet.RecordCount]);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Delete;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Name := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
    begin
        Table1.Close;
        Table1.DeleteTable;
    end;
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
```

```
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.AfterDelete:= Table1AfterDelete;
    Table1.Active:= True;
  end;
```

See Also

State (see page 7036)

BeforeDelete (see page 7049)

Delete (see page 6924)

8.1.9.6.4 TCustomADODataSet.AfterEdit Event

Occurs after an application starts editing a record.

Pascal

```
property AfterEdit: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterEdit;
```

Description

Write an AfterEdit event handler to take specific action immediately after dataset enters edit mode. AfterEdit is called by Edit after it enables editing of a record, recalculates calculated fields, and calls the data event handler to process a record change.

Delphi Examples:

```
{
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

BeforeEdit (see page 7049)

Edit (see page 6932)

8.1.9.6.5 TCustomADODataSet.AfterInsert Event

Occurs after an application inserts a new record.

Pascal

```
property AfterInsert: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterInsert;
```

Description

Write an AfterInsert event handler to take specific action immediately after an application inserts a record. The Insert and Append methods generate an AfterInsert event after inserting or appending a new record.

Delphi Examples:

```
{
This example updates a memo with a message when an
AfterInsert event or a BeforeInsert event occurs.
}
procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After inserting a record');
end;

procedure TForm1.ClientDataSet1BeforeInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before inserting a record');
end;
```

See Also

BeforeInsert (see page 7050)

Append (see page 6910)

Insert (see page 6959)

8.1.9.6.6 TCustomADODataSet.AfterOpen Event

Occurs after an application completes opening a dataset and before any data access occurs.

Pascal

```
property AfterOpen: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterOpen;
```

Description

Write an AfterOpen event handler to take specific action immediately after an application opens the dataset. AfterOpen is called after the dataset establishes access to its data and the dataset is put into dsBrowse state. For example, an AfterOpen event handler might check an ini file to determine the last record touched in the dataset the previous time the application ran, and position the dataset at that record.

Delphi Examples:

```
{
This example adds an entry to a memo with a message when an
AfterOpen event or an AfterClose event occurs.  Populate the
ClientDataSet events with CheckButtonActive.
}
procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  if (Button1.Enabled) then
    Memo2.Lines.Add('Now that the dataset is open, record information is available.')
  else
    Memo2.Lines.Add('Now that the dataset is closed, record information is not available.');
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
  RemoteServer1.Connected := False;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].DisableControls;
  end;
end;

procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
```

See Also

Active (see page 7004)

State (see page 7036)

BeforeOpen (see page 7051)

Open (see page 6976)

Opening and Closing Datasets

8.1.9.6.7 TCustomADODataSet.AfterPost Event

Occurs after an application writes the active record to the database or change log and returns to browse state.

Pascal

```
property AfterPost: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterPost;
```

Description

Write an AfterPost event handler to take specific action immediately after an application posts a change to the active record. AfterPost is called after a modification or insertion is made to a record.

Delphi Examples:

```
{
This example updates the form's status bar with a message
when an AfterPost event occurs.
}
procedure TForm1.ClientDataSet1AfterPost(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before posting a record');
end;
```

See Also

BeforePost (see page 7052)

Post (see page 6977)

8.1.9.6.8 TCustomADODataSet.AfterRefresh Event

Occurs after an application refreshes the data in the dataset.

Pascal

```
property AfterRefresh: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterRefresh;
```

Description

Write an AfterRefresh event handler to take specific action immediately after an application has updated the records in the dataset. AfterRefresh is generated by calls to the Refresh method.

See Also

BeforeRefresh (see page 7052)

Refresh (see page 6991)

8.1.9.6.9 TCustomADODataSet.AfterScroll Event

Occurs after an application scrolls from one record to another.

Pascal

```
property AfterScroll: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterScroll;
```

Description

Write an AfterScroll event handler to take specific action immediately after an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. AfterScroll is called after all other events triggered by these methods and any other methods that switch from record to record in the dataset.

See Also

BeforeScroll (see page 7053)

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Prior (see page 6979)

Navigating Datasets

8.1.9.6.10 TCustomADODataSet.BeforeCancel Event

Occurs before an application executes a request to cancel changes to the active record.

Pascal

```
property BeforeCancel: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeCancel;
```

Description

Write a BeforeCancel event to take specific action before an application carries out a request to cancel changes. BeforeCancel is called by the Cancel method before it cancels a dataset operation such as Edit, Insert, or Delete.

An application might use the BeforeCancel event to record a user's changes in an undo buffer.

See Also

AfterCancel (see page 7038)

Delete (see page 6924)

Edit (see page 6932)

Insert (see page 6959)

8.1.9.6.11 TCustomADODataSet.BeforeClose Event

Occurs immediately before the dataset closes.

Pascal

```
property BeforeClose: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeClose;
```

Description

Write a BeforeClose event to take specific action before an application closes a dataset. Calling Close or setting the Active property to false results in a call to the BeforeClose event handler.

See Also

Active (see page 7004)

AfterClose (see page 7040)

Close (see page 6920)

Opening and Closing Datasets

Posting Data

8.1.9.6.12 TCustomADODataSet.BeforeDelete Event

Occurs before an application attempts to delete the active record.

Pascal

```
property BeforeDelete: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeDelete;
```

Description

Write a BeforeDelete event handler to take specific action before an application deletes the active record. BeforeDelete is called by Delete before it actually deletes a record.

In Delphi, making use of this event an application might, for example, display a dialog box asking for confirmation before deleting the record. On denial of confirmation, the application could abort the deletion by calling the Abort procedure.

See Also

AfterDelete (see page 7041)

Delete (see page 6924)

Abort (see page 22786)

8.1.9.6.13 TCustomADODataSet.BeforeEdit Event

Occurs before an application enters edit mode for the active record.

Pascal

```
property BeforeEdit: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeEdit;
```

Description

Write a BeforeEdit event handler to take specific action before an application enables editing of the active record. For example, an application that keeps a log of database edits could use the BeforeEdit event to record the edit request, time, and user before entering edit state.

Delphi Examples:

```
{
```

```
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

State (see page 7036)

AfterEdit (see page 7044)

Edit (see page 6932)

8.1.9.6.14 TCustomADODataSet.BeforeInsert Event

Occurs before an application enters insert mode.

Pascal

```
property BeforeInsert: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeInsert;
```

Description

Write a BeforeInsert event handler to take specific action before an application inserts or appends a new record. The Insert or Append method generates a BeforeInsert method before it sets the dataset into dsInsert state.

Delphi Examples:

```
{
This example updates a memo with a message when an
AfterInsert event or a BeforeInsert event occurs.
}
procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After inserting a record');
end;

procedure TForm1.ClientDataSet1BeforeInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before inserting a record');
end;
{
This example uses the BeforeInsert event to do data
validation. If the StrToInt function raises an exception,
the edit control's contents are set to a valid value so
that the assignment to the INTEGER field in the dataset
```

```
will succeed.  Continuing after the exception causes a
'0' to be placed in the PORTA field.
DataSet design:
This example requires a ClientDataSet, a DBGrid, a
DataSource and a DBNavigator.  Name the ClientDataSet CDS.
Set the DataSet property in the DataSource to CDS and name
that DataSource DS.  Set the DataSource in the DBNavigator
to DS the DataSource in the DBGrid to DS.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
type to Integer.  The file used to load CDS must have an
attribute name of HOST with an attribute type of string and
an attribute name of PORTA with an attribute type of integer.
Add DB, DBCtrls, Grids, DBGrids and DBClient to the uses clause.
}
{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button3Click(Sender: TObject);
begin
  CDS.Insert;
  CDS.FieldByName('PORTA').AsInteger := StrToInt(Edit1.Text);
  CDS.Post;
end;

procedure TForm1.CDSBeforeInsert(DataSet: TDataSet);
begin
  try
  { Make sure edit field can be converted to integer --
   this will raise an exception if it can't }
    StrToInt(Edit1.Text);
  except
    Edit1.Text := '0';
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;
```

See Also

State (see page 7036)

AfterInsert (see page 7044)

Append (see page 6910)

Insert (see page 6959)

8.1.9.6.15 TCustomADODataSet.BeforeOpen Event

Occurs before an application executes a request to open a dataset.

Pascal

```
property BeforeOpen: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeOpen;
```

Description

Write a BeforeOpen event handler to take specific action before an application opens a dataset for viewing or editing. BeforeOpen is triggered when an application sets the Active property to true for a dataset or an application calls Open.

See Also

Active (see page 7004)

AfterOpen (see page 7045)

Open (see page 6976)

Opening and Closing Datasets

8.1.9.6.16 TCustomADODataSet.BeforePost Event

Occurs before an application posts changes for the active record to the database or change log.

Pascal

```
property BeforePost: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforePost;
```

Description

Write a BeforePost event handler to take specific action before an application posts dataset changes. BeforePost is triggered when an application calls the Post method. Post checks to make sure all required fields are present, then calls BeforePost before posting the record.

An application might use BeforePost to perform validity checks on data changes before committing them. If it encountered a validity problem, it could call Abort to cancel the Post operation (Delphi) or throw an exception (C++).

Delphi Examples:

```
{
This example checks for a valid entry in a TDBEdit control
and calls the Abort procedure if the control is empty; Abort
cancels the post before it happens.
}
procedure TForm1.ClientDataSet1BeforePost(DataSet: TDataSet);
begin
  if DBEdit1.Text = '' then
    Abort;
  DataSet.Fields[2].AsString := DBEdit1.Text;
end;
```

See Also

AfterPost (see page 7046)

Post (see page 6977)

Abort (see page 22786)

Working with Field Component Methods at Runtime

8.1.9.6.17 TCustomADODataSet.BeforeRefresh Event

Occurs immediately before an application refreshes the data in the dataset.

Pascal

```
property BeforeRefresh: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeRefresh;
```

Description

Write a BeforeRefresh event handler to take specific action immediately before an application updates the records in the dataset. BeforeRefresh is generated by calls to the Refresh method.

See Also

AfterRefresh (see page 7047)

Refresh (see page 6991)

8.1.9.6.18 TCustomADODataSet.BeforeScroll Event

Occurs before an application scrolls from one record to another.

Pascal

```
property BeforeScroll: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeScroll;
```

Description

Write a BeforeScroll event handler to take specific action immediately before an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. BeforeScroll is called before all other events triggered by these methods and any other methods that switch from record to record in the dataset.

See Also

AfterScroll (see page 7047)

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Prior (see page 6979)

Navigating Datasets

8.1.9.6.19 TCustomADODataSet.OnCalcFields Event

Occurs when an application recalculates calculated fields.

Pascal

```
property OnCalcFields: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent OnCalcFields;
```

Description

Write an OnCalcFields event handler to take specific action when an application recalculates calculated fields. A calculated field is one that derives its value from the values in one or more fields in the dataset, sometimes with additional processing.

OnCalcFields is triggered when:

A dataset is opened.

A dataset is put into dsEdit state.

A record is retrieved from a database.

When the AutoCalcFields property is true, OnCalcFields is also triggered when:

Focus moves from one visual control to another, or from one column to another is a data-aware grid control and modifications were made to the record.

Note: When the AutoCalcFields property is true, an OnCalcFields event handler should not modify the dataset (or a linked dataset if it is part of a master-detail relationship), because such modifications retrigger the OnCalcField event, leading to infinite recursion.

If an application permits users to change data, OnCalcFields is frequently triggered. To reduce the frequency with which OnCalcFields occurs, set AutoCalcFields to false.

Warning: When the dataset is the master table of a master-detail relationship, OnCalcFields occurs before detail sets have been synchronized with the master table.

See Also

Active (see page 7004)

AutoCalcFields (see page 7006)

Open (see page 6976)

Post (see page 6977)

Calculating Fields

Using Internally Calculated Fields in Client Datasets

Defining a Calculated Field

8.1.9.6.20 TCustomADODataSet.OnDeleteError Event

Occurs when an application attempts to delete a record and an exception is raised.

Pascal

```
property OnDeleteError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnDeleteError;
```

Description

Write an OnDeleteError event handler to handle exceptions that occur when an attempt to delete a record fails.

See Also

OnEditError (see page 7054)

OnPostError (see page 7056)

Delete (see page 6924)

8.1.9.6.21 TCustomADODataSet.OnEditError Event

Occurs when an application attempts to modify or insert a record and an exception is raised.

Pascal

```
property OnEditError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnEditError;
```

Description

Write an OnEditError event handler to handle exceptions that occur when an attempt to edit a record fails.

See Also

OnDeleteError (see page 7054)

OnPostError (see page 7056)

Edit (see page 6932)

Insert (see page 6959)

8.1.9.6.22 TCustomADODataSet.OnEndOfRecordset Event

Occurs when moving a recordset's row pointer past the last row.

Pascal

```
property OnEndOfRecordset: TEndOfRecordsetEvent;
```

C++

```
__property TEndOfRecordsetEvent OnEndOfRecordset;
```

Description

Write an OnEndOfRecordset event handler to take specific action after an attempt to move to a row past the end of the recordset. No row navigation methods should trigger this event due to the effects of the Eof property. This event should only be triggered by internal recordset navigation operations or by direct access to the Recordset (see page 1221) object (through the dataset component's Recordset (see page 1221) property).

DataSet is the ADO dataset component that triggered the OnEndOfRecordset event. This dataset component also contains the recordset in question.

MoreData is a boolean value indicating whether there are more unfetched rows beyond the last row fetched prior to the OnEndOfRecordset event.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event.

Note: : A Recordset (see page 1221) object's row pointer is distinct from the record pointer of its ADO dataset component. It is possible that the ADO Recordset (see page 1221) object will generate row movement events without the ADO dataset component generating a BeforeScroll (see page 1239) or AfterScroll (see page 1233) event. A couple reasons for this are background updating and resynchronizing of the records as data is changed. The OnEndOfRecordset event is purely a Recordset (see page 1221) object event and independent of any dataset component record navigation events.

8.1.9.6.23 TCustomADODataSet.OnFetchComplete Event

Occurs after records have been retrieved into the recordset.

Pascal

```
property OnFetchComplete: TRecordsetEvent;
```

C++

```
__property TRecordsetEvent OnFetchComplete;
```

Description

Write an OnFetchComplete event handler to take specific action after all of the rows from an asynchronous data retrieval operation have been returned to the recordset. Calling the dataset component's Open method or setting its Active (see page 1201) property to true will lead to this event occurring.

DataSet is the ADO dataset component that triggered the OnFetchComplete event. This dataset component also contains the recordset in question.

Error (see page 1312) is an ADO Error (see page 1312) object. See the Microsoft Data Access SDK for more information on using ADO Error (see page 1312) objects. The Error (see page 1312) parameter is only set if an error occurred during execution of the operation that triggered the event. In this case, the EventStatus parameter will contain the value esErrorsOccured.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event.

See Also

OnFetchProgress (see page 1242)

8.1.9.6.24 TCustomADODataSet.OnFetchProgress Event

Occurs periodically during an asynchronous data retrieval operation.

Pascal

```
property OnFetchProgress: TFetchProgressEvent;
```

C++

```
__property TFetchProgressEvent OnFetchProgress;
```

Description

Write an OnFetchProgress event handler to take specific action during an asynchronous data retrieval operation. The OnFetchProgress event fires periodically during the data retrieval to provide indications of its progress. Create (see page 1156) a handler for this event to react to this periodic notification, such as providing the user with a visual indication of the progress of the data retrieval.

DataSet is the ADO dataset component that triggered the OnFetchProgress event. This dataset component also contains the recordset in question.

Progress is the number of records that have been received since the data fetching operation began.

MaxProgress is the total number of records to be retrieved by the operation.

Progress and MaxProgress used together to get percent complete. For example, Progress divided by MaxProgress and multiplied by 100 yields the percent completion of the data fetching.

```
procedure TForm1.ADODataSet1FetchProgress(DataSet: TCustomADODataSet; Progress, MaxProgress:
```

```
Integer; var EventStatus: TEventStatus);
begin
  Caption := 'Percent complete: ' +
    IntToStr(Trunc(Progress / MaxProgress * 100)) + '%';
  Application.ProcessMessages;
end;
void __fastcall TForm1::ADODataSet1FetchProgress(TCustomADODataSet *DataSet, int Progress, int
MaxProgress, TEventStatus &EventStatus)
{
  Caption = "Percent complete: " +
    IntToStr(Trunc(Progress / MaxProgress * 100)) + "%";
  Application->ProcessMessages;
}
```

See Also

OnFetchComplete (see page 1242)

8.1.9.6.25 TCustomADODataSet.OnFieldChangeComplete Event

Occurs after a field has changed.

Pascal

```
property OnFieldChangeComplete: TFieldChangeCompleteEvent;
```

C++

```
__property TFieldChangeCompleteEvent OnFieldChangeComplete;
```

Description

Write an OnFieldChangeComplete event handler to take specific action before a pending operation changes the value of one or more ADO Field objects in the Recordset (see page 1221) object. (The term "Field object" as used here refers to the ADO Field object, not VCL persistent field objects that descend from TField.)

DataSet is the ADO dataset component that triggered the OnFieldChangeComplete event. This dataset component also contains the recordset in question.

FieldCount is the number of ADO Field objects in the Fields parameter.

Fields is an array of Variants (see page 23567) representing the ADO Field objects about to be changed by the operation. See the Microsoft Data Access SDK for more information on using ADO Field objects.

Error (see page 1312) is an ADO Error (see page 1312) object. See the Microsoft Data Access SDK for more information on using ADO Error (see page 1312) objects. The Error (see page 1312) parameter is only set if an error occurred during execution of the operation that triggered the event. In this case, the EventStatus parameter will contain the value esErrorsOccured.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event.

Note: A Recordset (see page 1221) object's Field objects are distinct from the VCL field objects of its ADO dataset component. The OnFieldChangeComplete event is purely a Recordset (see page 1221) object event and independent of any VCL field change events.

See Also

OnWillChangeField (see page 1248)

OnRecordChangeComplete (see page 1246)

OnRecordsetChangeComplete (see page 1247)

8.1.9.6.26 TCustomADODataSet.OnFilterRecord Event

Occurs each time a different record in the dataset becomes the active record and filtering is enabled.

Pascal

```
property OnFilterRecord: TFilterRecordEvent;
```

C++

```
__property TFilterRecordEvent OnFilterRecord;
```

Description

Write an OnFilterRecord event handler to specify for each record in a dataset whether it should be visible to the application. To indicate that a record passes the filter condition, the OnFilterRecord event handler must set the Accept parameter to true. To exclude a record, set the Accept parameter to false. Most TDataSet descendants (such as ADO datasets or BDE-based datasets) initialize Accept to true before calling the OnFilterRecord event handler.

Warning: Do not supply an OnFilterRecord event handler to a unidirectional dataset. Unidirectional datasets do not support filters, and assigning an OnFilterRecord event handler causes the dataset to raise an exception.

Filtering is enabled if the Filtered property is true. When an application is processing a filter, the State property for the dataset is dsFilter.

Use an OnFilterRecord event handler to filter records using a criterion that can't be implemented using the Filter property. For example, using the Filter property, field comparisons are not supported against local tables (Paradox, dBASE, Access, FoxPro), but an OnFilterRecord event handler can implement any criterion at all.

Tip: Be sure that any interactions between the Filter property and the OnFilterRecord event handler do not result in an empty filter set when they are used simultaneously in an application.

See Also

Filter (see page 7026)

Filtered (see page 7027)

FilterOptions (see page 7027)

State (see page 7036)

Creating Filters

Navigating Records in a Filtered Dataset

Displaying and Editing a Subset of Data Using Filters

Writing an OnFilterRecord Event Handler

8.1.9.6.27 TCustomADODataSet.OnMoveComplete Event

Occurs after the row pointer in a recordset has moved.

Pascal

```
property OnMoveComplete: TRecordsetErrorEvent;
```

C++

```
__property TRecordsetErrorEvent OnMoveComplete;
```

Description

Write an OnMoveComplete event handler to take specific action after the row pointer has moved in the dataset component's

Recordset (see page 1221) object. Calls to the First, Last, Next, Prior, MoveBy, Insert, Delete, Post, Requery (see page 1191), and Resynch methods can lead to the firing of this event. Setting or getting a bookmark can also trigger the event.

DataSet is the ADO dataset component that triggered the OnMoveComplete event. This dataset component also contains the recordset in question.

Reason indicates the type of move that triggered the event. While the TEventReason (see page 1296) type contains more constants, the value of Reason for an OnMoveComplete event is limited to: erRequery, erMove, erMoveFirst, erMoveNext, erMovePrevious, and erMoveLast.

Error (see page 1312) is an ADO Error (see page 1312) object. See the Microsoft Data Access SDK for more information on using ADO Error (see page 1312) objects. The Error (see page 1312) parameter is only set if an error occurred during execution of the operation that triggered the event. In this case, the EventStatus parameter will contain the value esErrorsOccured.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event.

Note: : A Recordset (see page 1221) object's row pointer is distinct from the record pointer of its ADO dataset component. It is possible that the ADO Recordset (see page 1221) object will generate row movement events without the ADO dataset component generating a BeforeScroll (see page 1239) or AfterScroll (see page 1233) event. A couple reasons for this are background updating and resynchronizing of the records as data is changed. The OnMoveComplete event is purely a Recordset (see page 1221) object event and independent of any dataset component record navigation events.

See Also

OnWillMove (see page 1250)

8.1.9.6.28 TCustomADODataSet.OnNewRecord Event

Occurs when an application inserts or appends a new dataset record.

Pascal

```
property OnNewRecord: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent OnNewRecord;
```

Description

Write an OnNewRecord event handler to take specific actions as an application inserts or appends a new record. OnNewRecord is called as part of the insert or append process. An application might use the OnNewRecord event to set initial values for a record or as a way of implementing cascading insertions in related datasets.

See Also

AfterInsert (see page 7044)

BeforeInsert (see page 7050)

8.1.9.6.29 TCustomADODataSet.OnPostError Event

Occurs when an application attempts to modify or insert a record and an exception is raised.

Pascal

```
property OnPostError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnPostError;
```

Description

Write an OnPostError event handler to handle exceptions that occur when an attempt to post a record fails.

See Also

OnDeleteError (see page 7054)

OnEditError (see page 7054)

Post (see page 6977)

8.1.9.6.30 TCustomADODataSet.OnRecordChangeComplete Event

Occurs after one or more records in the recordset change.

Pascal

```
property OnRecordChangeComplete: TRecordChangeCompleteEvent;
```

C++

```
__property TRecordChangeCompleteEvent OnRecordChangeComplete;
```

Description

Write an OnRecordChangeComplete event handler to take specific action after one or more rows in the dataset component's Recordset (see page 1221) object have changed.

DataSet is the ADO dataset component that triggered the OnRecordChangeComplete event. This dataset component also contains the recordset in question.

Reason indicates the type of change that triggered the event. While the TEventReason (see page 1296) type contains more constants, the value of Reason for an OnRecordChangeComplete event is limited to: erAddNew, erDelete, erUpdate, erUndoUpdate, erUndoAddNew, erUndoDelete, and erFirstChange.

RecordCount is the number of records that were changed.

Error (see page 1312) is an ADO Error (see page 1312) object. See the Microsoft Data Access SDK for more information on using ADO Error (see page 1312) objects. The Error (see page 1312) parameter is only set if an error occurred during execution of the operation that triggered the event. In this case, the EventStatus parameter will contain the value esErrorsOccured.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event.

Note: : The OnRecordChangeComplete event is purely a Recordset (see page 1221) object event and independent of any VCL data change events.

See Also

OnWillChangeRecord (see page 1248)

OnFieldChangeComplete (see page 1243)

OnRecordsetChangeComplete (see page 1247)

8.1.9.6.31 TCustomADODataSet.OnRecordsetChangeComplete Event

Occurs after the recordset has changed.

Pascal

```
property OnRecordsetChangeComplete: TRecordsetErrorEvent;
```

C++

```
__property TRecordsetErrorEvent OnRecordsetChangeComplete;
```

Description

Write an OnRecordsetChangeComplete event handler to take specific action after the dataset component's Recordset (see page 1221) object has changed.

DataSet is the ADO dataset component that triggered the OnRecordsetChangeComplete event. This dataset component also contains the recordset in question.

Reason indicates the type of change that triggered the event. While the TEventReason (see page 1296) type contains more constants, the value of Reason for an OnRecordsetChangeComplete event is limited to: erAddNew, erDelete, erUpdate, erUndoUpdate, erUndoAddNew, erUndoDelete, and erFirstChange.

Error (see page 1312) is an ADO Error (see page 1312) object. See the Microsoft Data Access SDK for more information on using ADO Error (see page 1312) objects. The Error (see page 1312) parameter is only set if an error occurred during execution of the operation that triggered the event. In this case, the EventStatus parameter will contain the value esErrorsOccured.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event. EventStatus will be esOK if the operation that triggered the event executed without error, esErrorsOccured if error occurred during execution of that operation, or esCancel if the operation was canceled.

Note: : The OnRecordsetChangeComplete event is purely a Recordset (see page 1221) object event and independent of any VCL data change events.

See Also

OnWillChangeRecordset (see page 1249)

OnRecordChangeComplete (see page 1246)

OnFieldChangeComplete (see page 1243)

8.1.9.6.32 TCustomADODataSet.OnRecordsetCreate Event

Occurs when the recordset has been initialized and is ready to use.

Pascal

```
property OnRecordsetCreate: TRecordsetCreate;
```

C++

```
__property TRecordsetCreate OnRecordsetCreate;
```

Description

Write an OnRecordsetCreate event handler to take specific action when the Recordset (see page 1221) property is first available for use. When the ADO dataset is first opened, it initializes the Recordset (see page 1221) property to the interface that it uses for accessing with data. OnRecordsetCreate occurs when Recordset (see page 1221) is fully initialized, and can be used by an application.

DataSet is the ADO dataset component whose Recordset (see page 1221) property has just been initialized to the interface for an ADO recordset.

Recordset (see page 1221) is the interface to the ADO recordset. It is the value of the Recordset (see page 1221) property.

See Also

RecordSet (see page 1221)

Open (see page 6976)

8.1.9.6.33 TCustomADODataSet.OnWillChangeField Event

Occurs before a field is changed.

Pascal

```
property OnWillChangeField: TWillChangeFieldEvent;
```

C++

```
__property TWillChangeFieldEvent OnWillChangeField;
```

Description

Write an OnWillChangeField event handler to take specific action before a pending operation changes the value of one or more ADO Field objects in the Recordset (see page 1221) object. (The term "Field object" as used here refers to the ADO Field object, not VCL persistent field objects that descend from TField.)

DataSet is the ADO dataset component that triggered the OnWillChangeField event. This dataset component also contains the recordset in question.

FieldCount is the number of ADO Field objects in the Fields parameter.

Fields is an array of Variants (see page 23567) representing the ADO Field objects about to be changed by the operation. See the Microsoft Data Access SDK for more information on using ADO Field objects.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event.

Note: : A Recordset (see page 1221) object's Field objects are distinct from the VCL field objects of its ADO dataset component. The OnWillChangeField event is purely a Recordset (see page 1221) object event and independent of any VCL field change events.

See Also

OnFieldChangeComplete (see page 1243)

OnWillChangeRecord (see page 1248)

OnWillChangeRecordset (see page 1249)

8.1.9.6.34 TCustomADODataSet.OnWillChangeRecord Event

Occurs before a record in the recordset is changed.

Pascal

```
property OnWillChangeRecord: TWillChangeRecordEvent;
```

C++

```
__property TWillChangeRecordEvent OnWillChangeRecord;
```

Description

Write an OnWillChangeRecord event handler to take specific action just before a pending operation changes a row in the dataset component's recordset object.

DataSet is the ADO dataset component that triggered the OnWillChangeRecord event. This dataset component also contains the recordset in question.

Reason indicates the type of change that triggered the event. While the TEventReason (see page 1296) type contains more constants, the value of Reason for an OnWillChangeRecord event is limited to: erAddNew, erDelete, erUpdate, erUndoUpdate, erUndoAddNew, erUndoDelete, and erFirstChange.

RecordCount is the number of records that were changed.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event.

Note: : The OnWillChangeRecord event is purely a Recordset (see page 1221) object event and independent of any VCL data change events.

See Also

OnRecordChangeComplete (see page 1246)

OnWillChangeField (see page 1248)

OnWillChangeRecordset (see page 1249)

8.1.9.6.35 TCustomADODataSet.OnWillChangeRecordset Event

Occurs before the recordset has changed.

Pascal

```
property OnWillChangeRecordset: TRecordsetReasonEvent;
```

C++

```
__property TRecordsetReasonEvent OnWillChangeRecordset;
```

Description

Write an OnWillChangeRecordset event handler to take specific action before the dataset component's Recordset (see page 1221) object is changed.

DataSet is the ADO dataset component that triggered the OnWillChangeRecordset event. This dataset component also contains the recordset in question.

Reason indicates the type of change that triggered the event. While the TEventReason (see page 1296) type contains more constants, the value of Reason for an OnWillChangeRecordset event is limited to: erAddNew, erDelete, erUpdate, erUndoUpdate, erUndoAddNew, erUndoDelete, and erFirstChange.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event. EventStatus will be esOK if the operation that triggered the event executed without error, esErrorsOccured if error occurred during execution of that operation, or esCancel if the operation was canceled.

Note: : The OnWillChangeRecordset event is purely a Recordset (see page 1221) object event and independent of any VCL data change events.

See Also

RecordSet (see page 1221)

OnWillChangeRecord (see page 1248)

OnWillChangeField (see page 1248)

8.1.9.6.36 TCustomADODataSet.OnWillMove Event

Occurs before the row pointer in a recordset is moved.

Pascal

```
property OnWillMove: TRecordsetReasonEvent;
```

C++

```
__property TRecordsetReasonEvent OnWillMove;
```

Description

Write an OnWillMove event handler to take specific action before an attempt to move the row pointer in the dataset component's Recordset (see page 1221) object. Calls to the First, Last, Next, Prior, MoveBy, Insert, Delete, Post, Requery (see page 1191), and Resynch methods can lead to the firing of this event. Setting or getting a bookmark can also trigger the event.

DataSet is the ADO dataset component that triggered the OnWillMove event. This dataset component also contains the recordset in question.

Reason indicates the type of move that triggered the event. While the TEventReason (see page 1296) type contains more constants, the value of Reason for an OnWillMove event is limited to: erRequery, erMove, erMoveFirst, erMoveNext, erMovePrevious, and erMoveLast.

The EventStatus value (ADO EventStatusEnum) indicates the success or failure of the execution of the operation that triggered the event.

Note: : A Recordset (see page 1221) object's row pointer is distinct from the record pointer of its ADO dataset component. It is possible that the ADO Recordset (see page 1221) object will generate row movement events without the ADO dataset component generating a BeforeScroll (see page 1239) or AfterScroll (see page 1233) event. A couple reasons for this are background updating and resynchronizing of the records as data is changed. The OnWillMove event is purely a Recordset (see page 1221) object event, and independent of any dataset component record navigation events.

See Also

OnMoveComplete (see page 1244)

8.1.10 TParameter Class

TParameter represents a parameter.

Class Hierarchy

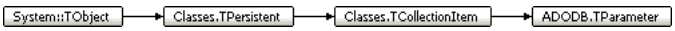

Pascal

```
TParameter = class(TCollectionItem);
```

C++

```
class TParameter : public TCollectionItem;
```

File

ADODB

Description

Use the properties and methods of a TParameter to set or get the value of a parameter used in an ADO query, stored procedure, dataset, or command component. Objects that use parameters create TParameter objects to represent those parameters and use a TParameters (see page 1261) collection to organize and operate on the TParameter objects. For example, TParameters (see page 1261) are used by TADOQuery (see page 1084) objects and TADOStoredProc (see page 1106) components (in their Parameters (see page 1259) properties).

See Also

TParameters (see page 1261)

TParameters (see page 1261)

TParameters (see page 1261)

Parameters

Using Parameters in Queries

8.1.10.1 TParameter Members

The following tables list the members exposed by TParameter.

Protected Methods

	Name	Description
	AssignTo (see page 1254)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 1255)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	IsEqual (see page 1255)	This is IsEqual, a member of class TParameter.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AppendChunk (see page 1253)	Adds data to a parameter.
	Assign (see page 1253)	Copies the properties and value of one TParameter (see page 1250) object to another.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2757)	Creates and initializes a TCollectionItem (see page 2755) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.

V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LoadFromFile (see page 1255)	Loads the value of a parameter from a file.
	LoadFromStream (see page 1255)	Loads the value of a parameter from a memory stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
R	ParameterObject (see page 1259)	Provides direct access to the ADO Parameter object.
R	Parameters (see page 1259)	Provides a reference to the TParameters (see page 1261) container.
R	Properties (see page 1260)	This is Properties, a member of class TParameter.

Published Properties

	Name	Description
	Attributes (see page 1257)	Specifies the attributes of a parameter.
	DataType (see page 1257)	Indicates the type of value the parameter represents.
	Direction (see page 1257)	Indicates the type of parameter.
	Name (see page 1258)	Identifies the TParameter (see page 1250) object.
	NumericScale (see page 1258)	Specifies the number of decimal places for the parameter.
	Precision (see page 1259)	Specifies the number of digits allowed for a numeric parameter.
	Size (see page 1260)	Specifies the size of a string type parameter.
	Value (see page 1260)	Represents the value of the parameter as a Variant.

8.1.10.2 TParameter Methods

The methods of the TParameter class are listed here.

Protected Methods

	Name	Description
V	AssignTo (see page 1254)	Copies the properties of an object to a destination object.

	GetDisplayName (see page 1255)	Returns the name of the collection item as it appears in the collection editor.
	IsEqual (see page 1255)	This is IsEqual, a member of class TParameter.

Public Methods

	Name	Description
	AppendChunk (see page 1253)	Adds data to a parameter.
	Assign (see page 1253)	Copies the properties and value of one TParameter (see page 1250) object to another.
	LoadFromFile (see page 1255)	Loads the value of a parameter from a file.
	LoadFromStream (see page 1255)	Loads the value of a parameter from a memory stream.

8.1.10.2.1 TParameter.AppendChunk Method

Adds data to a parameter.

Pascal

```
procedure AppendChunk(Val: OleVariant);
```

C++

```
__fastcall AppendChunk(OleVariant Val);
```

Description

Call AppendChunk to put data into or add data to a parameter's value. Use AppendChunk to insert binary or text data into a parameter.

AppendChunk may be used to pass all of the data at one time or in pieces using multiple calls to the method. The first call to AppendChunk overwrites any existing data in the parameter. If the data is being passed in its entirety using one call to AppendChunk, no further action takes place. If data is passed in multiple calls, the data passed in Val in each subsequent call to AppendChunk is appended to data previously passed.

Val is the data to pass to the parameter.

Note: : The Attributes (see page 1257) property must contain paLong to use AppendChunk.

See Also

Attributes (see page 1257)

8.1.10.2.2 TParameter.Assign Method

Copies the properties and value of one TParameter (see page 1250) object to another.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to make one TParameter (see page 1250) an exact clone of another. Assign copies the values of the Attributes (see page 1257), DataType (see page 1257), Direction (see page 1257), Name (see page 1258), NumericScale (see page 1258), Precision (see page 1259), Size (see page 1260), and Value (see page 1260) properties from the parameter object specified in Source to the calling parameter object.

```
ADOStoredProc1.Parameters[0].Assign(ADOStoredProc2.Parameters.Items[0]);
ADOStoredProc1->Parameters[0]->Assign(ADOStoredProc2->Parameters.Items[0]);
```

Assign may be used with various types of sources: another TParameter (see page 1250), a field in a table, a string list object

(most often for a memo type parameter), or a TBitmap (for BLOB type fields), a TPicture (for BLOB type fields). In the example below, Assign is used to give a value for a memo type parameter from a string list object (the Lines property of a TMemo).

```
ADOStoredProc1.Parameters[1].Assign(Memo1.Lines);
ADOStoredProc1->Parameters[1]->Assign(Memo1->Lines);
```

See Also

Attributes (see page 1257)

Direction (see page 1257)

Name (see page 1258)

NumericScale (see page 1258)

Precision (see page 1259)

Size (see page 1260)

Value (see page 1260)

DataType (see page 1257)

8.1.10.2.3 TParameter.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign (see page 2930)

EConvertError (see page 22724)

8.1.10.2.4 TParameter.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName (see page 2761)

GetNamePath (see page 2759)

8.1.10.2.5 TParameter.IsEqual Method

Pascal

```
function IsEqual(Value: TParameter): Boolean;
```

C++

```
__fastcall Boolean IsEqual(TParameter Value);
```

Description

This is IsEqual, a member of class TParameter.

8.1.10.2.6 TParameter.LoadFromFile Method

Loads the value of a parameter from a file.

Pascal

```
procedure LoadFromFile(const FileName: string; DataType: TDataType);
```

C++

```
__fastcall LoadFromFile(const AnsiString FileName, TDataType DataType);
```

Description

Call LoadFromFile to copy the contents of a file into a parameter.

FileName contains the name of the file from which to copy data.

DataType (see page 1257) is the field data type the loaded data is to be treated as.

See Also

LoadFromStream (see page 1255)

8.1.10.2.7 TParameter.LoadFromStream Method

Loads the value of a parameter from a memory stream.

Pascal

```
procedure LoadFromStream(Stream: TStream; DataType: TDataType);
```

C++

```
__fastcall LoadFromStream(TStream Stream, TDataType DataType);
```

Description

Call LoadFromStream to copy the contents of a memory stream into a parameter.

Stream is a reference to the memory stream from which to copy data.

DataType (see page 1257) is the field data type the loaded data is to be treated as.

```
var
  MS: TMemoryStream;
begin
  MS := TMemoryStream.Create;
try
    ... fill memory stream with data ...
    ADOQuery1.Parameters[4].LoadFromStream(MS);
finally
if assigned(MS) then
    MS.Free;
end;
end;
TMemoryStream *MS = new(TMemoryStream);
try
{
  Memo1->Lines->SaveToStream(MS);
  MS->Seek(0, soFromBeginning);
  ADOQuery1->Parameters[4]->LoadFromStream(MS);
}
__finally
{
delete MS;
}
```

See Also

LoadFromFile (see page 1255)

8.1.10.3 TParameter Properties

The properties of the TParameter class are listed here.

Public Properties

	Name	Description
R	ParameterObject (see page 1259)	Provides direct access to the ADO Parameter object.
R	Parameters (see page 1259)	Provides a reference to the TParameters (see page 1261) container.
R	Properties (see page 1260)	This is Properties, a member of class TParameter.

Published Properties

	Name	Description
	Attributes (see page 1257)	Specifies the attributes of a parameter.
	DataType (see page 1257)	Indicates the type of value the parameter represents.
	Direction (see page 1257)	Indicates the type of parameter.
	Name (see page 1258)	Identifies the TParameter (see page 1250) object.
	NumericScale (see page 1258)	Specifies the number of decimal places for the parameter.
	Precision (see page 1259)	Specifies the number of digits allowed for a numeric parameter.

	Size (see page 1260)	Specifies the size of a string type parameter.
	Value (see page 1260)	Represents the value of the parameter as a Variant.

8.1.10.3.1 TParameter.Attributes Property

Specifies the attributes of a parameter.

Pascal

```
property Attributes: TParameterAttributes;
```

C++

```
__property TParameterAttributes Attributes;
```

Description

Set Attributes to specify the attributes of a parameter, thus controlling the type of values a parameter will accept. By default, Attributes is assigned no value at all. Attributes may be set to any one or a combination of the three value TParameter (see page 1250) Attributes constants: paSigned, paNullable, and paLong.

See Also

Working with Stored Procedure Parameters

Supplying Parameters at Design Time

8.1.10.3.2 TParameter.DataType Property

Indicates the type of value the parameter represents.

Pascal

```
property DataType: TDataType;
```

C++

```
__property TDataType DataType;
```

Description

Set DataType to reflect the data type of the value the parameter represents. The default value of DataType is ftUnknown.

Read DataType to discover the type of data that was assigned to the parameter. Each possible value of DataType corresponds to a type of table field.

```
ADOStoredProc1.Parameters.Items[0].DataType := ftString;
ADOStoredProc1->Parameters->Items[0]->DataType = ftString;
```

See Also

Working with Stored Procedure Parameters

Supplying Parameters at Design Time

8.1.10.3.3 TParameter.Direction Property

Indicates the type of parameter.

Pascal

```
property Direction: TParameterDirection;
```

C++

```
__property TParameterDirection Direction;
```

Description

Set Direction to indicate the type of parameter the TParameter (see page 1250) object represents. A parameter can be one of the types: input, output, input-output, or return value. Assign (see page 1253) a type of pdUnknown when the actual type of the parameter is not known at the time. The default value of Direction is pdInput.

See Also

Working with Stored Procedure Parameters

Supplying Parameters at Design Time

8.1.10.3.4 TParameter.Name Property

Identifies the TParameter (see page 1250) object.

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

Use Name to identify a particular parameter within a TParameters (see page 1261) object.

Note: Name is the name of the TParameter (see page 1250) object, not the name of the parameter it represents in a query or stored procedure.

See Also

Working with Stored Procedure Parameters

8.1.10.3.5 TParameter.NumericScale Property

Specifies the number of decimal places for the parameter.

Pascal

```
property NumericScale: Byte;
```

C++

```
__property Byte NumericScale;
```

Description

Set NumericScale to indicate the maximum number of characters that can appear to the right of the decimal place in the parameter's value (the scale). (The Precision (see page 1259) property controls the total number of digits allowed.) By default, NumericScale is zero.

See Also

Precision (see page 1259)

Size (see page 1260)

Working with Stored Procedure Parameters

Supplying Parameters at Design Time

8.1.10.3.6 TParameter.ParameterObject Property

Provides direct access to the ADO Parameter object.

Pascal

```
property ParameterObject: _Parameter;
```

C++

```
__property _Parameter ParameterObject;
```

Description

Use ParameterObject to gain direct access to the ADO Parameter object that corresponds to the TParameter (see page 1250) object. Using this reference, you can access properties and call methods of the ADO Parameter object.

Under ordinary circumstances it should not be necessary to use the ADO Parameter directly in an application. This property is provided for the instances where a property or method of the ADO Parameter is needed but not implemented in the TParameter (see page 1250) object. You should be familiar with ADO objects before attempting to use them directly. Consult the Microsoft Data Access SDK for more specific information on ADO objects and their properties and methods.

8.1.10.3.7 TParameter.Parameters Property

Provides a reference to the TParameters (see page 1261) container.

Pascal

```
property Parameters: TParameters;
```

C++

```
__property TParameters Parameters;
```

Description

Use Parameters to get a reference to the TParameters (see page 1261) object in which the TParameter (see page 1250) object is contained.

See Also

TParameters (see page 1261)

8.1.10.3.8 TParameter.Precision Property

Specifies the number of digits allowed for a numeric parameter.

Pascal

```
property Precision: Byte;
```

C++

```
__property Byte Precision;
```

Description

Set Precision to indicate the maximum number of digits allowed in the parameter, including both significant and fractional (the precision). (The NumericScale (see page 1258) property controls the total number of decimal digits allowed.) By default, Precision is zero.

See Also

NumericScale (see page 1258)

Working with Stored Procedure Parameters

Supplying Parameters at Design Time

8.1.10.3.9 TParameter.Properties Property

Pascal

```
property Properties: Properties;
```

C++

```
__property Properties Properties;
```

Description

This is Properties, a member of class TParameter.

8.1.10.3.10 TParameter.Size Property

Specifies the size of a string type parameter.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

Set Size to indicate the maximum number of characters the parameter may contain. By default, Size is zero.

Note: Size only pertains to string type parameters (like CHAR and VARCHAR). For numeric type parameters, use the NumericScale (see page 1258) and Precision (see page 1259) properties. For other column types, Size should remain set to zero.

See Also

Precision (see page 1259)

NumericScale (see page 1258)

Working with Stored Procedure Parameters

Supplying Parameters at Design Time

8.1.10.3.11 TParameter.Value Property

Represents the value of the parameter as a Variant.

Pascal

```
property Value: Variant;
```

C++

```
__property Variant Value;
```

Description

Use Value in generic code that manipulates the values of parameters without needing to know the field type the parameters represent.

See Also

Working with Stored Procedure Parameters

Supplying Parameters at Design Time

8.1.11 TParameters Class

TParameters manages a collection of parameters.

Class Hierarchy

Pascal

```
TParameters = class(TOwnedCollection);
```

C++

```
class TParameters : public TOwnedCollection;
```

File

ADODB

Description

Use TParameters to manage a collection of TParameter (see page 1250) objects for an object that uses field parameters. For example, TADOStoredProc (see page 1106) objects and TADOQuery (see page 1084) objects use TParameters objects (in properties named Parameters (see page 1312)) to create and access their parameters.

Use the properties and methods of TParameters to:

Access a specific parameter.

Add or delete field parameters from the list.

Get or set the values of individual parameters.

Compare collections of parameters.

Iterate through all parameters.

Copy a collection of parameters to another parameter collection.

See Also

Parameters

TParameter (see page 1250)

8.1.11.1 TParameters Members

The following tables list the members exposed by TParameters.

Protected Methods

	Name	Description
	Added (see page 2736)	Responds when items are added to the collection.
	AppendParameters (see page 1264)	This is AppendParameters, a member of class TParameters.

V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
	Create_Parameter (see page 1265)	This is Create_Parameter, a member of class TParameters.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
V	GetAttr (see page 1267)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
V	GetAttrCount (see page 1267)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
V	GetItemAttr (see page 1268)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2915)	Returns the owner of an object.
	GetOwner (see page 2932)	Returns the owner of an object.
	InternalRefresh (see page 1269)	This is InternalRefresh, a member of class TParameters.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 1271)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	Modified (see page 1272)	This is Modified, a member of class TParameters.
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 2736)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	AddParameter (see page 1264)	Adds a parameter to the collection.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
	AssignValues (see page 1265)	Assigns parameter values from another parameters collection.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 2734)	Creates and initializes a collection.
	Create (see page 2914)	Creates and initializes a TOwnedCollection (see page 2912) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParameter (see page 1266)	Adds a parameter to the collection.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.

V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	FindParam (see page 1266)	Locates the parameter with a given name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParamList (see page 1268)	Fills a TList object with names of available parameters.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEqual (see page 1269)	Indicates whether two TParameters (see page 1261) objects have identical contents.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
	ParamByName (see page 1269)	Locates the parameter with a given name.
	ParseSQL (see page 1270)	Parses an SQL statement for parameters and TParameter (see page 1250) objects.
	Refresh (see page 1271)	Refreshes a parameter collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Command (see page 1272)	Indicates the associated ADO command object.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 1272)	Contains the collection of parameters.
R	ParameterCollection (see page 1273)	Encapsulates the ADO Parameters (see page 1312) Collection object.
	ParamValues (see page 1273)	Retrieves the value of one or more parameters.

8.1.11.2 TParameters Methods

The methods of the TParameters class are listed here.

Protected Methods

	Name	Description
	AppendParameters (see page 1264)	This is AppendParameters, a member of class TParameters.
	Create_Parameter (see page 1265)	This is Create_Parameter, a member of class TParameters.
V	GetAttr (see page 1267)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.

	GetAttrCount (see page 1267)	Returns the number of custom attributes associated with items in the collection.
	GetItemAttr (see page 1268)	Returns the value of a custom attribute assigned to one of the collection's items.
	InternalRefresh (see page 1269)	This is InternalRefresh, a member of class TParameters.
	Update (see page 1271)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	AddParameter (see page 1264)	Adds a parameter to the collection.
	AssignValues (see page 1265)	Assigns parameter values from another parameters collection.
	CreateParameter (see page 1266)	Adds a parameter to the collection.
	FindParam (see page 1266)	Locates the parameter with a given name.
	GetParamList (see page 1268)	Fills a TList object with names of available parameters.
	IsEqual (see page 1269)	Indicates whether two TParameters (see page 1261) objects have identical contents.
	ParamByName (see page 1269)	Locates the parameter with a given name.
	ParseSQL (see page 1270)	Parses an SQL statement for parameters and TParameter (see page 1250) objects.
	Refresh (see page 1271)	Refreshes a parameter collection.

8.1.11.2.1 TParameters.AddParameter Method

Adds a parameter to the collection.

Pascal

```
function AddParameter: TParameter;
```

C++

```
__fastcall TParameter AddParameter();
```

Description

Call AddParameter to create a single empty TParameter (see page 1250) object and add it to the TParameters (see page 1261) collection. All properties of the newly added TParameter (see page 1250) object are either at their default value or are empty.

```
with ADOStoredProc1.AddParameter do begin
  DataType := ftString;
..Direction := pdInput;
  Value := 'malathion';
end;
int PCount;
ADOStoredProc1->AddParameter;
PCount = ADOStoredProc1->Parameters->Count - 1;
ADOStoredProc1->Parameters[PCount]->DataType = ftString;
ADOStoredProc1->Parameters[PCount]->Direction = pdInput;
ADOStoredProc1->Parameters[PCount]->Value = "malathion";
```

See Also

TParameter (see page 1250)

Parameters (see page 1259)

CreateParameter (see page 1266)

8.1.11.2.2 TParameters.AppendParameters Method

Pascal

```
procedure AppendParameters;
```

C++

```
__fastcall AppendParameters();
```

Description

This is AppendParameters, a member of class TParameters.

8.1.11.2.3 TParameters.AssignValues Method

Assigns parameter values from another parameters collection.

Pascal

```
procedure AssignValues(Value: TParameters);
```

C++

```
__fastcall AssignValues(TParameters Value);
```

Description

Call AssignValues to assign the values of parameters in the calling TParameters (see page 1261) object using the values in another TParameters (see page 1261).

```
ADOQuery1.Parameters.AssignValues(ADOQuery2.Parameters);
ADOQuery1->Parameters->AssignValues(ADOQuery2->Parameters);
```

Each parameter with the same name as a parameter in the source TParameters (see page 1261) specified in Value is assigned the value from the source parameter. Parameters (see page 1312) in the calling TParameters (see page 1261) object that do not exist in the source TParameters (see page 1261) object are unaffected. Parameters (see page 1312) that only exist in the source TParameters (see page 1261) are not subsequently created in the calling TParameters (see page 1261) object. For example if the calling TParameters (see page 1261) has parameters named Parm1 and Parm2 and the source TParameters (see page 1261) has Parm2 and Parm3:

Parm1 in the calling TParameters (see page 1261) is unaffected.

Parm2 gets the value of Parm2 in the source TParameters (see page 1261).

No Parm3 is created in the calling TParameters (see page 1261).

See Also

TParameter (see page 1250)

Value (see page 1260)

8.1.11.2.4 TParameters.Create_Parameter Method

Pascal

```
function Create_Parameter(const Name: WideString; DataType: TDataType; Direction:
TParameterDirection = pdInput; Size: Integer = 0): _Parameter;
```

C++

```
__fastcall _Parameter Create_Parameter(const BSTR Name, TDataType DataType,
TParameterDirection Direction = pdInput, int Size = 0);
```

Description

This is Create_Parameter, a member of class TParameters.

8.1.11.2.5 TParameters.CreateParameter Method

Adds a parameter to the collection.

Pascal

```
function CreateParameter(const Name: WideString; DataType: TDataType; Direction:
TParameterDirection; Size: Integer; Value: OleVariant): TParameter;
```

C++

```
__fastcall TParameter CreateParameter(const BSTR Name, TDataType DataType, TParameterDirection
Direction, int Size, OleVariant Value);
```

Description

Call CreateParameter to create a single TParameter (see page 1250) object and add it to the TParameters (see page 1261) collection. Property values for the newly added TParameter (see page 1250) object are supplied via CreateParameter parameters.

Name is a WideString value containing the name of the newly created TParameter (see page 1250) object.

DataType is a TDataType (see page 1315) value indicating the data type of the parameter the new TParameter (see page 1250) object represents.

Direction is a TParameterDirection (see page 1303) value indicating the parameter type. A parameter represented by a TParameter (see page 1250) may be any one of: input, output, input-output, return value. Designate a parameter as unknown if the parameter type is not know at the time the TParameter (see page 1250) is created.

Size indicates the maximum size of the parameter's value.

Value is an OleVariant containing the parameter's value. A value need not be passed for Value when the TParameter (see page 1250) is created. It can be assigned later. If no value is assigned at this time, pass a nil (Delphi) or NULL (C++) for Value.

```
ADOStoredProc1.CreateParameter('Parm1', ftString, pdInput, 10, nil);
```

See Also

TParameter (see page 1250)

Parameters (see page 1259)

AddParameter (see page 1264)

8.1.11.2.6 TParameters.FindParam Method

Locates the parameter with a given name.

Pascal

```
function FindParam(const Value: WideString): TParameter;
```

C++

```
__fastcall TParameter FindParam(const BSTR Value);
```

Description

Use FindParam method to find a parameter with the name specified in Value. If a match is found, FindParam returns a reference to the TParameter (see page 1250) object (assignment-compatible with TParam objects) for the parameter of the specified name. Otherwise, it returns nil (Delphi) or NULL (C++). Use this method rather than a direct reference to the Items (see page 1272) property to avoid depending on the order of the entries.

To locate more than one parameter at a time, by name, use the GetParamList (see page 1268) method instead. To get only

the value of a named parameter, use the ParamValues (see page 1273) property.

Note: FindParam operates with the name of a parameter as it appears in an SQL statement or stored procedure, not the Name property of a TParameter (see page 1250) object. The two ways of referring to a parameter should not be confused.

See Also

TParameter (see page 1250)

TParam (see page 7386)

Items (see page 1272)

Name (see page 1258)

ParamByName (see page 1269)

GetParamList (see page 1268)

ParamValues (see page 1273)

8.1.11.2.7 TParameters.GetAttr Method

Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.

Pascal

```
function GetAttr(Index: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetAttr(int Index);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. The GetAttr method returns the name of an attribute.

Index identifies the attribute whose name is requested. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

As implemented by TCollection, GetAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttrCount (see page 2744)

GetItemAttr (see page 2745)

8.1.11.2.8 TParameters.GetAttrCount Method

Returns the number of custom attributes associated with items in the collection.

Pascal

```
function GetAttrCount: Integer; override;
```

C++

```
virtual __fastcall int GetAttrCount();
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetAttrCount returns the number of distinct attributes assigned to each item in the collection.

As implemented in TCollection, GetAttrCount always returns 0, because TCollection defines no custom attributes.

See Also

GetAttr (see page 2743)

GetItemAttr (see page 2745)

8.1.11.2.9 TParameters.GetItemAttr Method

Returns the value of a custom attribute assigned to one of the collection's items.

Pascal

```
function GetItemAttr(Index: Integer; ItemIndex: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetItemAttr(int Index, int ItemIndex);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetItemAttr returns the value of one of these attributes for a specified item in the collection.

Index identifies which attribute's value is desired. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

ItemIndex identifies the item whose attribute value is desired. This is an index into the Items property array.

As implemented in TCollection, GetItemAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttr (see page 2743)

GetAttrCount (see page 2744)

8.1.11.2.10 TParameters.GetParamList Method

Fills a TList object with names of available parameters.

Pascal

```
procedure GetParamList(List: TList; const ParamNames: WideString);
```

C++

```
__fastcall GetParamList(TList List, const BSTR ParamNames);
```

Description

Use GetParamList to obtain a subset of the parameters in the TParameters (see page 1261) collection that have been identified by name.

ParamNames is a string giving the names of all field parameters of interest. Multiple parameter names are separated by semicolons (;).

List is the TList object into which the parameters names are placed. GetParamList adds the field parameters to List in the order they appear in ParamNames. If ParamNames includes a name that does not match any of the field parameters in Items (see page 1272), an exception is raised.

Note: GetParamList operates with the names of parameters as they appear in an SQL statement or stored procedure, not the Name property of TParameter (see page 1250) objects. The two ways of referring to a parameter should not be confused.

See Also

TParameter (see page 1250)

Name (see page 1258)

ParamByName (see page 1269)

FindParam (see page 1266)

ParamValues (see page 1273)

8.1.11.2.11 TParameters.InternalRefresh Method

Pascal

```
function InternalRefresh: Boolean;
```

C++

```
__fastcall Boolean InternalRefresh();
```

Description

This is InternalRefresh, a member of class TParameters.

8.1.11.2.12 TParameters.IsEqual Method

Indicates whether two TParameters (see page 1261) objects have identical contents.

Pascal

```
function IsEqual(Value: TParameters): Boolean;
```

C++

```
__fastcall Boolean IsEqual(TParameters Value);
```

Description

Use IsEqual to compare the contents of two TParameters (see page 1261) objects. IsEqual returns true if the parameter collection specified in Value exactly matches the calling parameter collection. To match, each parameter collection must have the same number of TParameter (see page 1250) objects, in the same order, which match on name, datatype, parameter type, and value.

```
if (ADOQuery1.Parameters.IsEqual(ADOQuery2.Parameters) then
  { do one thing }
else
  { do something else };
if (ADOQuery1->Parameters->IsEqual(ADOQuery2->Parameters)
  // do one thing
else
  // do something else
```

See Also

TParameter (see page 1250)

8.1.11.2.13 TParameters.ParamByName Method

Locates the parameter with a given name.

Pascal

```
function ParamByName(const Value: WideString): TParameter;
```

C++

```
__fastcall TParameter ParamByName(const BSTR Value);
```

Description

Use ParamByName method to find a parameter with the name specified in Value. If a match is found, ParamByName returns a reference to the corresponding TParameter (see page 1250) object. Otherwise, the TParameter (see page 1250) reference is nil (Delphi) or NULL (C++) and an exception is raised. Use this method rather than a direct reference to the Items (see page 1272) property to avoid depending on the order of the entries.

```
ADOQuery1.Parameters.ParamByName('Verb').Value := 'propitiate';
ADOQuery1.Open;
ADOQuery1->Parameters->ParamByName("Verb")->Value = "propitiate";
ADOQuery1->Open();
```

To locate a parameter by name without raising an exception if the parameter is not found, use the FindParam (see page 1266) method.

To locate more than one parameter at a time, by name, use the GetParamList (see page 1268) method instead. To get only the value of a named parameter, use the ParamValues (see page 1273) property.

Note: ParamByName operates with the name of a parameter as it appears in an SQL statement or stored procedure, not the Name property of a TParameter (see page 1250) object. The two ways of referring to a parameter should not be confused.

See Also

Items (see page 1272)

Name (see page 1258)

GetParamList (see page 1268)

FindParam (see page 1266)

ParamValues (see page 1273)

8.1.11.2.14 TParameters.ParseSQL Method

Parses an SQL statement for parameters and TParameter (see page 1250) objects.

Pascal

```
function ParseSQL(SQL: WideString; DoCreate: Boolean): WideString;
```

C++

```
__fastcall BSTR ParseSQL(BSTR SQL, Boolean DoCreate);
```

Description

Call ParseSQL to parse an SQL statement for parameters. For each parameter found in the SQL statement, a TParameter (see page 1250) object is added to the calling TParameters (see page 1261) object using the name of the parameter in the SQL statement.

SQL contains the SQL statement to parse.

DoCreate indicates whether to clear all existing parameter definitions before parsing the SQL statement.

ParseSQL returns a parsed version of the SQL statement specified in the SQL parameter.

```
ADOQuery1.Parameters.ParseSQL(ADOQuery1.Text, True);
ADOQuery1->Parameters->ParseSQL(ADOQuery1->Text, true);
```

8.1.11.2.15 TParameters.Refresh Method

Refreshes a parameter collection.

Pascal

```
function Refresh: Boolean;
```

C++

```
__fastcall Boolean Refresh();
```

Description

Call Refresh to ensure a parameter collection has the most up to date information on available parameters.

See Also

TParameter (see page 1250)

Parameters (see page 1259)

8.1.11.2.16 TParameters.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed (see page 2740)

Changed (see page 2758)

8.1.11.3 TParameters Properties

The properties of the TParameters class are listed here.

Protected Properties

	Name	Description
R	Modified (see page 1272)	This is Modified, a member of class TParameters.

Public Properties

	Name	Description
R	Command (see page 1272)	Indicates the associated ADO command object.

	Items (see page 1272)	Contains the collection of parameters.
R	ParameterCollection (see page 1273)	Encapsulates the ADO Parameters (see page 1312) Collection object.
	ParamValues (see page 1273)	Retrieves the value of one or more parameters.

8.1.11.3.1 TParameters.Command Property

Indicates the associated ADO command object.

Pascal

```
property Command: TADOCommand;
```

C++

```
__property TADOCommand Command;
```

Description

Read Command to get a reference to the ADO command object associated with the ADO dataset object that contains the TParameters (see page 1261) object.

8.1.11.3.2 TParameters.Items Property

Contains the collection of parameters.

Pascal

```
property Items [Index: Integer]: TParameter;
```

C++

```
__property TParameter Items[int Index];
```

Description

Items is a collection of TParameter (see page 1250) objects. Each TParameter (see page 1250) object in the collection represents an individual parameter. Use Items to access a particular parameter.

Index indicates the specific parameter to access. Index identifies the parameter's position in the collection of parameters, in the range 0 to Count - 1.

See Also

TParameter (see page 1250)

TADOQuery (see page 1084)

TADOStoredProc (see page 1106)

Count (see page 2749)

ExecSQL (see page 1100)

operator_[]

8.1.11.3.3 TParameters.Modified Property

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

This is Modified, a member of class TParameters.

8.1.11.3.4 TParameters.ParameterCollection Property

Encapsulates the ADO Parameters (see page 1312) Collection object.

Pascal

```
property ParameterCollection: Parameters;
```

C++

```
__property Parameters ParameterCollection;
```

Description

Use Properties (see page 1313) to get direct access to the ADO Parameters (see page 1312) Collection for the ADO Recordset object with which the TParameters (see page 1261) is associated. A Parameters (see page 1312) Collection object contains al of the ADO Parameters (see page 1312) objects for a command. A Parameters (see page 1312) Collection contains one or more ADO Parameter objects.

Note: Use of ParameterCollection to directly access the underlying ADO Parameters (see page 1312) Collection object requires a good working knowledge of ADO objects in general and ADO Parameters (see page 1312) Collection objects in specific. Consult the Microsoft Data Store SDK help for specific information on using ADO Parameters (see page 1312) Collection objects.

See Also

TParameter (see page 1250)

8.1.11.3.5 TParameters.ParamValues Property

Retrieves the value of one or more parameters.

Pascal

```
property ParamValues [const ParamName: WideString]: Variant;
```

C++

```
__property Variant ParamValues[BSTR const ParamName];
```

Description

Use ParamValues to get or set the individual values of parameters identified by name.

ParamName contains the names of the individual parameters of interest. If ParamValues is used to access more than one parameter, the names are separated by semicolons (;).

Setting ParamValues sets the Value property for each parameter listed in the ParamName string. Specify the values as Variants (see page 23567), in order, in a variant array.

Getting ParamValues retrieves a variant array of variants, each of which represents the value of one of the named parameters.

If ParamName includes a name that does not match any of the field parameters in Items (see page 1272), an exception is raised.

Note: ParamValues operates with the names of parameters as they appear in an SQL statement or stored procedure, not the Name property of TParameter (see page 1250) objects. The two should not be confused.

See Also

Items (see page 1272)

Name (see page 1258)

GetParamList (see page 1268)

FindParam (see page 1266)

ParamByName (see page 1269)

8.1.12 TRDSConnection Class

TRDSConnection implements an RDS DataSpace object.

Class Hierarchy

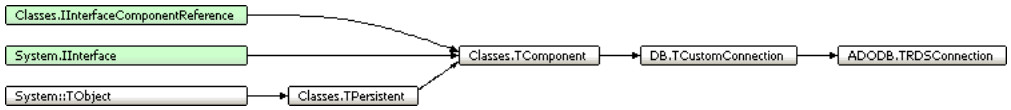

Pascal

```
TRDSConnection = class(TCustomConnection);
```

C++

```
class TRDSConnection : public TCustomConnection;
```

File

ADODB

Description

The TRDSConnection class exposes the functionality of the RDS DataSpace object. RDS DataSpace objects manage the marshaling of data when a Recordset object is passed from one process or machine to another. Use TRDSConnection when building multi-tier applications using ADO-based business objects (Application Servers).

Use TRDSConnection in the place of TADOConnection (see page 1001). Associate the TRDSComponent with a TADODataSet (see page 1041) component using the latter's RDSConnection property.

By default, an RDS connection component is configured to work with the RDS DataFactory object. The DataFactory object can be used when you do not require a specialized business object. When using a DataFactory object, the ConnectionString and CommandText properties of the TADODataSet (see page 1041) are passed to the Query method of the DataFactory object. When using a business object, the CommandText property should contain the name of a property of the business object which returns the desired Recordset for the TADODataSet (see page 1041). The A value for the ConnectionString property is not required when using a business object.

See Also

TADOConnection (see page 1001)

TADODataSet (see page 1041)

TADODataSet (see page 1041)

Using TADODataSet

Database Architecture

8.1.12.1 TRDSConnection Members

The following tables list the members exposed by TRDSConnection.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoConnect (see page 1279)	Provides the interface for a method that opens a connection.
V	DoDisconnect (see page 1279)	Provides the interface for a method that terminates the connection.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetConnected (see page 1280)	Returns the value of the Connected property.
V	GetDataSet (see page 6865)	Returns a dataset, given its index in the DataSets (see page 6870) property array.
V	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 6866)	Initializes the connection component after all objects in the form or data module have been loaded.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	RegisterClient (see page 6866)	Registers an object that uses the connection component.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetConnected (see page 6867)	Changes the value of the Connected (see page 6868) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.

	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	UnRegisterClient (see page 6868)	Removes a registered client so that it is no longer associated with the connection component.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 6863)	Closes the connection.
V	Create (see page 1278)	Creates an instance of TRDSConnection (see page 1274).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 1278)	Destroys the instance of a connection component.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetRecordset (see page 1280)	Retrieves a Recordset.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 6866)	Opens the connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AppServer (see page 1281)	Provides access to the application server communication interface.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 6870)	Lists the datasets currently using this connection component.
R	DataSpaceObject (see page 1282)	Provides access to the interface for the RDS DataSpace object.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	LoginPrompt (see page 6872)	Specifies whether a login dialog appears immediately before opening a new connection.
	OnLogin (see page 6873)	Occurs when a communication channel opens to the server.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterConnect (see page 1284)	Occurs after a connection is established.
	AfterDisconnect (see page 1284)	Occurs after the connection closes.

	BeforeConnect (see page 1284)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 1285)	Occurs immediately before the connection closes.

Published Properties

	Name	Description
	ComputerName (see page 1281)	Specifies the source of a business object.
	Connected (see page 1282)	Determines whether a connection has been established to the remote source of data.
	InternetTimeout (see page 1283)	Specifies the amount of time before a request times out.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ServerName (see page 1283)	Specifies the business object.
	Tag (see page 2808)	Stores an integer value as part of a component.

8.1.12.2 TRDSConnection.Create Constructor

Creates an instance of TRDSConnection (see page 1274).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TRDSConnection(TComponent * AOwner);
```

Description

Call Create to instantiate an RDS connection component at runtime. The Owner parameter specifies another component (usually a form or data module) that is responsible for freeing the RDS connection component.

Create

Calls its inherited Create method.

Associates the RDS connection component to the default DataFactory.

See Also

Owner (see page 2807)

8.1.12.3 TRDSConnection.Destroy Destructor

Destroys the instance of a connection component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRDSConnection();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which verifies that the TRDSConnection (see page 1274) reference is not nil before calling Destroy

Destroy closes all active datasets and closes its connection before freeing all internal resources.

Destroyperforms the following tasks:

Clears references.

Frees its data space object.

Calls the inherited destructor.

See Also

Create (see page 1278)

Free (see page 22300)

8.1.12.4 TRDSConnection Methods

The methods of the TRDSConnection class are listed here.

Protected Methods

	Name	Description
	DoConnect (see page 1279)	Provides the interface for a method that opens a connection.
	DoDisconnect (see page 1279)	Provides the interface for a method that terminates the connection.
	GetConnected (see page 1280)	Returns the value of the Connected property.

Public Methods

	Name	Description
	GetRecordset (see page 1280)	Retrieves a Recordset.

8.1.12.4.1 TRDSConnection.DoConnect Method

Provides the interface for a method that opens a connection.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

The Connected property uses DoConnect to establish a connection. Descendant classes override the DoConnect method to establish their connection as appropriate. As implemented in TCustomConnection, DoConnect does nothing.

See Also

Connected (see page 6868)

DoDisconnect (see page 6864)

8.1.12.4.2 TRDSConnection.DoDisconnect Method

Provides the interface for a method that terminates the connection.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

The Connected property uses DoDisconnect to close a connection. Descendant classes override the DoDisconnect method to

drop a connection. As implemented in TCustomConnection, DoDisconnect does nothing.

See Also

Connected (see page 6868)

DoConnect (see page 6864)

8.1.12.4.3 TRDSConnection.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected read implementation of the Connected property. Descendant classes override this method to return true if a connection is open and false otherwise. As defined in TCustomConnection, GetConnected always returns false.

See Also

Connected (see page 6868)

SetConnected (see page 6867)

8.1.12.4.4 TRDSConnection.GetRecordset Method

Retrieves a Recordset.

Pascal

```
function GetRecordset(const CommandText: WideString; ConnectionString: WideString = ''):
_Recordset;
```

C++

```
__fastcall _Recordset GetRecordset(const BSTR CommandText, BSTR ConnectionString = '');
```

Description

Call GetRecordset to retrieve a Recordset object from a business object. GetRecordset is called automatically by the TADODataSet (see page 1041) associated with the RDS connection component.

When using a business object, only the CommandText parameter should be passed a value. This is the Name property of the business object that returns the Recordset object. When GetRecordset is called by the TADODataSet (see page 1041) component, the value used for the CommandText parameter is the value in the CommandText property of the ADO dataset component.

When using a DataFactory, the CommandText parameter is a query command (SQL statement) to execute and the ConnectionString specifies the information needed to establish the connection. When GetRecordset is called by the ADO dataset component, the value used for the ConnectionString parameter is the value in the ConnectionString property of the ADO dataset component.

```
ADODataSet1.CommandText := 'SELECT * FROM Employee';
ADODataSet1.RecordSet := RDSConnection1.GetRecordset(ADODataSet1.CommandText, '');
ADODataSet1->CommandText := "SELECT * FROM Employee";
ADODataSet1->Recordset := RDSConnection1->GetRecordset(ADODataSet1.CommandText, "");
```

8.1.12.5 TRDSConnection Properties

The properties of the TRDSConnection class are listed here.

Public Properties

	Name	Description
R	AppServer (see page 1281)	Provides access to the application server communication interface.
R	DataSpaceObject (see page 1282)	Provides access to the interface for the RDS DataSpace object.

Published Properties

	Name	Description
	ComputerName (see page 1281)	Specifies the source of a business object.
	Connected (see page 1282)	Determines whether a connection has been established to the remote source of data.
	InternetTimeout (see page 1283)	Specifies the amount of time before a request times out.
	ServerName (see page 1283)	Specifies the business object.

8.1.12.5.1 TRDSConnection.AppServer Property

Provides access to the application server communication interface.

Pascal

```
property AppServer: OleVariant;
```

C++

```
__property OleVariant AppServer;
```

Description

Use AppServer to get direct access to the interface the ADO dataset uses to communicate with the application server. Use this property if you need to access additional properties or methods of the business object not related to Recordsets.

8.1.12.5.2 TRDSConnection.ComputerName Property

Specifies the source of a business object.

Pascal

```
property ComputerName: WideString;
```

C++

```
__property BSTR ComputerName;
```

Description

Use ComputerName to specify the source of a business object to use through the TRDSConnection (see page 1274) component. If left blank, the business object is loaded from the local machine.

Several protocols are supported for creating the business object: HTTP, HTTPS, and DCOM. For business objects created with HTTP and HTTPS, ComputerName is a string containing a Universal Resource Identifier (URL) identifying the Internet Information Server (IIS) Web server where an instance of the server business object is created. For DCOM, ComputerName is the name of the machine.

8.1.12.5.3 TRDSConnection.Connected Property

Determines whether a connection has been established to the remote source of data.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Set Connected to true to open the connection. Set Connected to false to terminate the connection.

Setting Connected to true generates a BeforeConnect event, calls the protected DoConnect method to establish the connection, and generates an AfterConnect event. In addition, when setting Connected to true, TCustomConnection may display a login dialog, depending on the value of LoginPrompt.

Setting Connected to false generates a BeforeDisconnect event, calls the protected DoConnect method to drop the connection, and generates an AfterDisconnect event.

When deriving custom connection components from TCustomConnection, override GetConnected to return true when a connection is established, and override DoConnect and DoDisconnect to create and drop the connection.

See Also

DataSets (see page 6870)

Open (see page 6866)

Close (see page 6863)

DoConnect (see page 6864)

DoDisconnect (see page 6864)

BeforeConnect (see page 6875)

AfterConnect (see page 6874)

BeforeDisconnect (see page 6875)

AfterDisconnect (see page 6875)

GetConnected (see page 6865)

LoginPrompt (see page 6872)

Disconnecting from a Database Server

Connecting to a Database Server

8.1.12.5.4 TRDSConnection.DataSpaceObject Property

Provides access to the interface for the RDS DataSpace object.

Pascal

```
property DataSpaceObject: DataSpace;
```

C++

```
__property DataSpace DataSpaceObject;
```

Description

Use DataSpaceObject to get direct access to the RDS DataSpace object the RDS connection component represents. This direct access reference allows an application to use properties and methods of the underlying DataSpace object. Accessing the underlying DataSpace object is especially useful for utilizing properties and methods of the DataSpace object not surfaced in TRDSConnection (see page 1274).

Note: Use of DataSpaceObject to directly access the underlying RDS DataSpace object requires a good working knowledge of ADO objects in general and the RDS DataSpace object in specific. It is not recommended that you use the DataSpace object directly unless familiar with DataSpace object operations. Consult the Microsoft Data Store SDK help for specific information on using RDS DataSpace objects.

8.1.12.5.5 TRDSConnection.InternetTimeout Property

Specifies the amount of time before a request times out.

Pascal

```
property InternetTimeout: Integer;
```

C++

```
__property int InternetTimeout;
```

Description

Use InternetTimeout to specify the amount of time (measured in milliseconds) after which a request times out. InternetTimeout only applies to requests sent with the HTTP and HTTPS protocols. The default value for InternetTimeout is zero.

In a multi-tier environment, requests for data may take several minutes to execute. InternetTimeout can be used to specify a longer time-out period for long running requests.

8.1.12.5.6 TRDSConnection.ServerName Property

Specifies the business object.

Pascal

```
property ServerName: WideString;
```

C++

```
__property BSTR ServerName;
```

Description

Use ServerName to specify the ProgID of the business object to instantiate. The default value is that in the RDSServer.DataFactory property. This is the ProgID of the RDS DataFactory object. The DataFactory allows for execution of queries against any datasource. See the Microsoft Data Store SDK help for more information on the DataFactory object.

See Also

Connecting to the Application Server using DataSnap Components

8.1.12.6 TRDSConnection Events

The events of the TRDSConnection class are listed here.

Published Events

	Name	Description
	AfterConnect (see page 1284)	Occurs after a connection is established.
	AfterDisconnect (see page 1284)	Occurs after the connection closes.
	BeforeConnect (see page 1284)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 1285)	Occurs immediately before the connection closes.

8.1.12.6.1 TRDSConnection.AfterConnect Event

Occurs after a connection is established.

Pascal

```
property AfterConnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterConnect;
```

Description

Write an AfterConnect event handler to take application-specific actions immediately after the connection component opens a connection to the remote source of database information.

See Also

BeforeConnect (see page 6875)

Connected (see page 6868)

Connecting to a Database Server

8.1.12.6.2 TRDSConnection.AfterDisconnect Event

Occurs after the connection closes.

Pascal

```
property AfterDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterDisconnect;
```

Description

Write an AfterDisconnect event handler to take application-specific actions after the connection component drops a connection.

See Also

BeforeDisconnect (see page 6875)

Connected (see page 6868)

Disconnecting from a Database Server

8.1.12.6.3 TRDSConnection.BeforeConnect Event

Occurs immediately before establishing a connection.

Pascal

```
property BeforeConnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeConnect;
```

Description

Write a BeforeConnect event handler to take application-specific actions before the connection component opens a connection to the remote source of database information.

See Also

AfterConnect (see page 6874)

Connected (see page 6868)

Connecting to a Database Server

8.1.12.6.4 TRDSConnection.BeforeDisconnect Event

Occurs immediately before the connection closes.

Pascal

```
property BeforeDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeDisconnect;
```

Description

Write a BeforeDisconnect event handler to take application-specific actions before dropping a connection.

Delphi Examples:

```
{
This BeforeDisconnect event handler makes sure that the
client dataset has applied all its pending updates before
terminating the connection to the server.
}
procedure TForm1.RemoteServer1BeforeDisconnect(Sender: TObject);
begin
  if (ClientDataSet1.ChangeCount > 0) then
    ClientDataSet1.ApplyUpdates(-1);
end;
```

See Also

AfterDisconnect (see page 6875)

Connected (see page 6868)

Disconnecting from a Database Server

8.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CreateUDLFile (see page 1286)	This is function ADODB.CreateUDLFile.
	DataLinkDir (see page 1286)	This is function ADODB.DataLinkDir.
	GetDataLinkFiles (see page 1287)	This is function ADODB.GetDataLinkFiles.
	GetDataLinkFiles (see page 1287)	This is function ADODB.GetDataLinkFiles.
	GetProviderNames (see page 1287)	This is function ADODB.GetProviderNames.
	GetProviderNames (see page 1288)	This is function ADODB.GetProviderNames.
	PromptDataLinkFile (see page 1288)	This is function ADODB.PromptDataLinkFile.
	PromptDataSource (see page 1288)	This is function ADODB.PromptDataSource.

8.2.1 ADODB.CreateUDLFile Function

Pascal

```
procedure CreateUDLFile(const FileName: WideString; const ProviderName: WideString; const
DataSourceName: WideString);
```

C++

```
CreateUDLFile(const BSTR FileName, const BSTR ProviderName, const BSTR DataSourceName);
```

File

ADODB

Description

This is function ADODB.CreateUDLFile.

8.2.2 ADODB.DataLinkDir Function

Pascal

```
function DataLinkDir: string;
```

C++

```
AnsiString DataLinkDir();
```

File

ADODB

Description

This is function ADODB.DataLinkDir.

8.2.3 ADODB.GetDataLinkFiles Function

Pascal

```
function GetDataLinkFiles(FileNames: TStrings; Directory: string = ''): Integer; overload;
```

C++

```
int GetDataLinkFiles(TStrings FileNames, AnsiString Directory = '');
```

File

ADODB

Description

This is function ADODB.GetDataLinkFiles.

8.2.4 ADODB.GetDataLinkFiles Function

Pascal

```
function GetDataLinkFiles(FileNames: TWideStrings; Directory: string = ''): Integer; overload;
```

C++

```
int GetDataLinkFiles(TWideStrings FileNames, AnsiString Directory = '');
```

File

ADODB

Description

This is function ADODB.GetDataLinkFiles.

8.2.5 ADODB.GetProviderNames Function

Pascal

```
procedure GetProviderNames(Names: TStrings); overload;
```

C++

```
GetProviderNames(TStrings Names);
```

File

ADODB

Description

This is function ADODB.GetProviderNames.

8.2.6 ADODB.GetProviderNames Function

Pascal

```
procedure GetProviderNames(Names: TWideStrings); overload;
```

C++

```
GetProviderNames(TWideStrings Names);
```

File

ADODB

Description

This is function ADODB.GetProviderNames.

8.2.7 ADODB.PromptDataLinkFile Function

Pascal

```
function PromptDataLinkFile(ParentHandle: THandle; InitialFile: WideString): WideString;
```

C++

```
BSTR PromptDataLinkFile(THandle ParentHandle, BSTR InitialFile);
```

File

ADODB

Description

This is function ADODB.PromptDataLinkFile.

8.2.8 ADODB.PromptDataSource Function

Pascal

```
function PromptDataSource(ParentHandle: THandle; InitialString: WideString): WideString;
```

C++

```
BSTR PromptDataSource(THandle ParentHandle, BSTR InitialString);
```

File

ADODB

Description

This is function ADODB.PromptDataSource.

8.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TADOLockType (see page 1289)	TADOLockType specifies the types of locks used.
	TAffectRecords (see page 1290)	This is record ADODB.TAffectRecords.
	TCommandType (see page 1291)	TCommandType values designate the type of a command.
	TConnectMode (see page 1292)	TConnectMode values indicate the persimissions available to a connection.
	TConnectOption (see page 1293)	TConnectOption indicates whether the connection to the ADO data store is synchronous or asynchronous.
	TCursorLocation (see page 1293)	TCursorLocation values specify whether to use client-side or server-side cursor library.
	TCursorOption (see page 1294)	TCursorOptions (see page 1314) values indicate the functionality a recordset provides.
	TCursorType (see page 1295)	TCursorType valies indicate the type of cursor to use for a recordset.
	TEventReason (see page 1296)	TEventReason values indicate the reason an event occurred.
	TEventStatus (see page 1298)	TEventStatus values indicate the success or failure of an operation.
	TExecuteOption (see page 1299)	This is record ADODB.TExecuteOption.
	TFilterGroup (see page 1299)	This is record ADODB.TFilterGroup.
	TIsolationLevel (see page 1300)	TIsolationLevel values specify the level of isolation for a connection.
	TMarshalOption (see page 1301)	This is record ADODB.TMarshalOption.
	TObjectState (see page 1302)	This is record ADODB.TObjectState.
	TParameterAttribute (see page 1302)	This is record ADODB.TParameterAttribute.
	TParameterDirection (see page 1303)	TParameterDirection specifies the direction of a parameter.
	TPersistFormat (see page 1303)	TPersistFormat values indicate the format for a saved recordset.
	TRecordStatus (see page 1304)	This is record ADODB.TRecordStatus.
	TSchemaInfo (see page 1305)	TSchemaInfo values indicate the type of schema information to retrieve.
	TSeekOption (see page 1308)	This is record ADODB.TSeekOption.
	TXactAttribute (see page 1308)	TXactAttribute values specify automated transaction behavior.

8.3.1 ADODB.TADOLockType Enumeration

TADOLockType specifies the types of locks used.

Pascal

```
TADOLockType = (
  ltUnspecified,
  ltReadOnly,
  ltPessimistic,
  ltOptimistic,
  ltBatchOptimistic
);
```

C++

```
enum TADOLockType {
  ltUnspecified,
  ltReadOnly,
  ltPessimistic,
  ltOptimistic,
```

```
  ltBatchOptimistic
};
```

File

ADODB

Description

Use TADOLockType with execution methods to specify the types of record locks used during editing. Lock type values allow you to specify whether the locking scheme used for a recordset is optimistic or pessimistic. To use a specific locking scheme, the provider and database type must support that locking scheme. Use ltUnspecified to have the provider use the default locking scheme. Use the LockType ltBatchOptimistic to open a recordset in batch update mode.

In addition to use as command execution method parameters, TADOLockType values are passed as var parameters to event handlers related to execution. This value indicates the lock type already in use when the execution was initiated. The LockType parameter of the TADOConnection (see page 1001). ::OnWillExecute event is an example of this.

The constants that make up the TADOLockType type correspond directly to the LockTypeEnum values with similar names. These values are discussed in the Microsoft Data Access SDK help, in the topic for the LockType property of the ADO Recordset object. See the SDK for additional information on the ADO constants and their effects.

TADOLockType consists of the five constants summarized in the following table:

Lock Type	Meaning
ltUnspecified	A specific lock type has not been specified or determined.
ltReadOnly	Read-only; the current execution cannot modify data.
ltPessimistic	Pessimistic locks; locks instantiated on a record-by-record basis when edit mode is applied to a record.
ltOptimistic	Optimistic locks; locked are initiated on a record-by-record basis, but only when updates are applied and a record is being updated.
ltBatchOptimistic	Optimistic batch updates; used for batch update mode rather than immediate update mode.

See Also

TCustomADONETConnector

TADOConnection

8.3.2 ADODB.TAffectRecords Enumeration

Pascal

```
TAffectRecords = (
  arCurrent,
  arFiltered,
  arAll,
  arAllChapters
);
```

C++

```
enum TAffectRecords {
  arCurrent,
  arFiltered,
  arAll,
  arAllChapters
};
```

File

ADODB

Description

This is record ADODB.TAffectRecords.

8.3.3 ADODB.TCommandType Enumeration

TCommandType values designate the type of a command.

Pascal

```
TCommandType = (
  cmdUnknown,
  cmdText,
  cmdTable,
  cmdStoredProc,
  cmdFile,
  cmdTableDirect
);
```

C++

```
enum TCommandType {
  cmdUnknown,
  cmdText,
  cmdTable,
  cmdStoredProc,
  cmdFile,
  cmdTableDirect
};
```

File

ADODB

Description

Use TCommandType values to specify the type for a command contained in the CommandText property of an ADO component capable of issuing commands.

The constants that make up the TCommandType type correspond directly to the ADO CommandTypeEnum constants. These constants are used in the CommandType property of ADO Command objects. For additional information on these ADO constants and their effects, see the Microsoft Data Access SDK help in the topic for the CommandType property of the ADO Command object.

TCommandType consists of the seven constants summarized in the following table:

Command type	Meaning
cmdUnknown	The type of command in the CommandText property is not known.
cmdText	CommandText is a textual definition of a command or stored procedure call; data returned by an internal SQL statement.
cmdTable	CommandText is the name of a table.
cmdStoredProc	CommandText is the name of a stored procedure.
cmdFile	CommandText is the name of a saved recordset file.
cmdTableDirect	CommandText is the name of a table; all columns are returned.

Note: The TCommandType constants cmdTable, cmdTableDirect, and cmdOpenFile are not for use with commands executed from a TADOCommand (see page 984) component.

See Also

TADOCommand

TCustomADONETConnector

8.3.4 ADODB.TConnectMode Enumeration

TConnectMode values indicate the persimissions available to a connection.

Pascal

```
TConnectMode = (
  cmUnknown,
  cmRead,
  cmWrite,
  cmReadWrite,
  cmShareDenyRead,
  cmShareDenyWrite,
  cmShareExclusive,
  cmShareDenyNone
);
```

C++

```
enum TConnectMode {
  cmUnknown,
  cmRead,
  cmWrite,
  cmReadWrite,
  cmShareDenyRead,
  cmShareDenyWrite,
  cmShareExclusive,
  cmShareDenyNone
};
```

File

ADODB

Description

Use TConnectMode values to determine the permissions available to a connection and thus what type of data operations can be performed through that connection.

TConnectMode constants correspond directly to the ConnectModeEnum values for the Mode property of the ADO Connection object. The TConnectMode constants have the same effect on a TADOConnection (see page 1001) component as the ConnectModeEnum values have on the Connection object. These values are discussed in the Microsoft Data Access SDK help.

TConnectMode consists of the eight constants summarized in the following table:

Connect Mode	Meaning
cmUnknown	The permissions have not yet been set for the connection or cannot be determined.
cmRead	Read-only permissions are available to the connection.
cmWrite	Write-only permissions are available to the connection.
cmReadWrite	Read/write permissions are available to the connection.

cmShareDenyRead	Prevents others from opening connections with read permissions.
cmShareDenyWrite	Prevents others from opening connection with write permissions.
cmShareExclusive	Prevents others from opening connection.
cmShareDenyNone	Prevents others from opening connection with any permissions.

See Also

TADOConnection

8.3.5 ADODB.TConnectOption Enumeration

TConnectOption indicates whether the connection to the ADO data store is synchronous or asynchronous.

Pascal

```
TConnectOption = (
  coConnectUnspecified,
  coAsyncConnect
);
```

C++

```
enum TConnectOption {
  coConnectUnspecified,
  coAsyncConnect
};
```

File

ADODB

Description

TConnectOption is the type of the TADOConnection (see page 1001) object's ConnectOptions property. It specifies whether the connection established by the TADOConnection (see page 1001) is synchronous or asynchronous. The following table lists the possible values:

Value	Meaning
coConnectUnspecified	The connection is formed synchronously.
coAsyncConnect	The connection is formed asynchronously. This is useful if the server is very slow, but the application must be aware that not all data values are necessarily available when the connection is first established.

See Also

TADOConnection

8.3.6 ADODB.TCursorLocation Enumeration

TCursorLocation values specify whether to use client-side or server-side cursor library.

Pascal

```
TCursorLocation = (
```

```
    clUseServer,
    clUseClient
  );
```

C++

```
  enum TCursorLocation {
    clUseServer,
    clUseClient
  };
```

File

ADODB

Description

Use TCursorLocation values when determining a cursor location to specify whether to use a client-side or server-side cursor library. The value clUseServer indicates a server-side cursor library is used and clUseClient that a client-side cursor is used.

The TCursorLocation constants correspond to the Long value in the CursorLocation property of ADO Connection and Recordset objects. The ADO constants are adUseNone, adUseClient, and adUseServer. The TCursorLocation values clUseServer and clUseClient have the same effect as the ADO values adUseClient and adUseServer. See the Microsoft Data Access SDK help for information on these values and their effects.

See Also

TCustomADONETConnector

TADOConnection

8.3.7 ADODB.TCursorOption Enumeration

TCursorOptions (see page 1314) values indicate the functionality a recordset provides.

Pascal

```
  TCursorOption = (
    coHoldRecords,
    coMovePrevious,
    coAddNew,
    coDelete,
    coUpdate,
    coBookmark,
    coApproxPosition,
    coUpdateBatch,
    coResync,
    coNotify,
    coFind,
    coSeek,
    coIndex
  );
```

C++

```
  enum TCursorOption {
    coHoldRecords,
    coMovePrevious,
    coAddNew,
    coDelete,
    coUpdate,
    coBookmark,
    coApproxPosition,
    coUpdateBatch,
```

```
  coResync,
  coNotify,
  coFind,
  coSeek,
  coIndex
};
```

File

ADODB

Description

TCursorOptions (see page 1314) values are individual options the Supports method checks for. Include one or more of these values as a set for the CursorOptions parameter of Supports.

The constants of the TCursorOptions (see page 1314) type correspond directly with the CursorOptions parameter of the Supports method of the ADO Recordset object. See the Microsoft Data Access SDK help for information on these values and their effects.

TCursorOptions (see page 1314) consists of the eleven constants summarized in the following table:

Cursor Option	Meaning
coHoldRecords	More records can be retrieved or the position of the next retrieve can be changed without committing pending changes.
coMovePrevious	The pointer in the ADO Recordset object can be moved backwards without using bookmarks.
coAddNew	The recordset supports adding new records.
coDelete	The recordset supports deleting existing records.
coUpdate	The recordset supports modifying existing records.
coBookmark	The Bookmark property of the Recordset object can be used to move to or access specific records.
coApproxPosition	The recordset supports use of the RecNo property of the ADO dataset component.
coUpdateBatch	Batch updating (using the UpdateBatch and CancelBatch methods) can be used to transmit changes to the provider in groups.
coResync	The data in the cursor can be updated with the Resync method.
coNotify	The recordset supports notifications and returns events.
coFind	The Locate method can be used with the ADO dataset component.
coSeek	The Seek method can be used to search for a row in a dataset.
coIndex	The IndexName property of the ADO dataset component can be used to specify an index.

See Also

TCustomADONETConnector

8.3.8 ADODB.TCursorType Enumeration

TCursorType valies indicate the type of cursor to use for a recordset.

Pascal

```
TCursorType = (
  ctUnspecified,
  ctOpenForwardOnly,
  ctKeyset,
```

```
    ctDynamic,
    ctStatic
  );
```

C++

```
  enum TCursorType {
    ctUnspecified,
    ctOpenForwardOnly,
    ctKeyset,
    ctDynamic,
    ctStatic
  };
```

File

ADODB

Description

Use TCursorType values to indicate the type of cursor to use when a recordset is opened.

The constants of TCursorType correspond with the CursorTypeEnum values for the CursorType property of the ADO recordset object. See the Microsoft Data Access SDK help for information on these values and their effects.

TCursorType consists of the five constants summarized in the following table:

Cursor Type	Meaning
ctUnspecified	A specific cursor type has not yet been specified.
ctOpenForwardOnly	Forward-only cursor in which you can only scroll forward; better performance because only requires single through recordset.
ctKeyset	Keyset cursor; records added by other users are not visible; records deleted by other users are inaccessible.
ctDynamic	Dynamic cursor; data modifications, deletions, and insertions by other users are visible; scrolling forward and back allowed.
ctStatic	Static cursor; static copy of records; data changes by other users not visible; used mostly for reporting.

Note: Only ctStatic is supported if the CursorLocation property of the ADO dataset component is set to clUseClient.

Note: If the requested cursor type is not supported by the provider, the provider may return another cursor type. This alternate cursor type is reflected in the CursorType property after the recordset is opened.

See Also

TCustomADONETConnector

8.3.9 ADODB.TEventReason Enumeration

TEventReason values indicate the reason an event occurred.

Pascal

```
  TEventReason = (
    erAddNew,
    erDelete,
    erUpdate,
    erUndoUpdate,
```

```
    erUndoAddNew,
    erUndoDelete,
    erRequery,
    erResynch,
    erClose,
    erMove,
    erFirstChange,
    erMoveFirst,
    erMoveNext,
    erMovePrevious,
    erMoveLast
  );
```

C++

```
  enum TEventReason {
    erAddNew,
    erDelete,
    erUpdate,
    erUndoUpdate,
    erUndoAddNew,
    erUndoDelete,
    erRequery,
    erResynch,
    erClose,
    erMove,
    erFirstChange,
    erMoveFirst,
    erMoveNext,
    erMovePrevious,
    erMoveLast
  };
```

File

ADODB

Description

Use TEventReason values to determine the reason an event has occurred.

TEventReason consists of the five constants summarized in the following table:

Execute Option	Meaning
erAddNew	A new row was added.
erDelete	An existing row was deleted.
erUpdate	An existing row was modified with new values.
erUndoUpdate	An update operation was canceled.
erUndoAddNew	A row insert operation was canceled.
erUndoDelete	A row delete operation was canceled.
erRequery	The recordset was refreshed with the Requery method.
erResynch	The recordset was resynchronized with the Resynch method.
erClose	The recordset was closed.
erMove	The recordset's row pointer moved.
erFirstChange	Equivalent to the ADO value ad.
erMoveFirst	The recordset's row pointer moved to the first row.
erMoveNext	The recordset's row pointer moved to the next row.
erMovePrevious	The recordset's row pointer moved to the previous row.

erMoveLast	The recordset's row pointer moved to the last row.

See Also

TCustomADONETConnector

TCustomADONETConnector

TCustomADONETConnector

TCustomADONETConnector

TCustomADONETConnector

TCustomADONETConnector

TCustomADONETConnector

8.3.10 ADODB.TEventStatus Enumeration

TEventStatus values indicate the success or failure of an operation.

Pascal

```
TEventStatus = (
  esOK,
  esErrorsOccured,
  esCantDeny,
  esCancel,
  esUnwantedEvent
);
```

C++

```
enum TEventStatus {
  esOK,
  esErrorsOccured,
  esCantDeny,
  esCancel,
  esUnwantedEvent
};
```

File

ADODB

Description

Read TEventStatus values to determine the success or failure status of an operation that triggers an event. A TEventStatus value is passed to the handler for every event of the TADOConnection (see page 1001) and TCustomADONETConnector descendant. Each TEventStatus is directly equivalent to the ADO value with a similar name.

In some cases, the event handler status parameter may be assigned a value to alter the outcome of the occurrence that triggered the event. These cases are documented in the help topics for individual connection and dataset component events.

The EventStatus parameter of associated event handlers surfaces the ADO adStatus parameter of its events, which is of ADO type EventStatusEnum. If the operation executed without problem, the TEventStatus value returned is esOK. If the operation did not execute without problem, one of the other values is returned. For instance, esErrorsOccured is returned when the operation failed because of one or more error conditions.

TEventStatus consists of the five constants summarized in the following table:

Execute Option	Meaning
esOK	Operation executed without problem.
esErrorsOccured	An error occurred during execution of the operation.
esCantDeny	A pending connection operation cannot be canceled. (Connection events only.).
esCancel	A pending connection has been canceled before it became active. (Connection events only.)
esUnwantedEvent	Set by the ADO method, prevents subsequent notification of the event.

See Also

TCustomADONETConnector

TADOConnection

8.3.11 ADODB.TExecuteOption Enumeration

Pascal

```
TExecuteOption = (
  eoAsyncExecute,
  eoAsyncFetch,
  eoAsyncFetchNonBlocking,
  eoExecuteNoRecords
);
```

C++

```
enum TExecuteOption {
  eoAsyncExecute,
  eoAsyncFetch,
  eoAsyncFetchNonBlocking,
  eoExecuteNoRecords
};
```

File

ADODB

Description

This is record ADODB.TExecuteOption.

8.3.12 ADODB.TFilterGroup Enumeration

Pascal

```
TFilterGroup = (
  fgUnassigned,
  fgNone,
  fgPendingRecords,
  fgAffectedRecords,
  fgFetchedRecords,
  fgPredicate,
  fgConflictingRecords
);
```

C++

```
enum TFilterGroup {
  fgUnassigned,
  fgNone,
  fgPendingRecords,
  fgAffectedRecords,
  fgFetchedRecords,
  fgPredicate,
  fgConflictingRecords
};
```

File

ADODB

Description

This is record ADODB.TFilterGroup.

8.3.13 ADODB.TIsolationLevel Enumeration

TIsolationLevel values specify the level of isolation for a connection.

Pascal

```
TIsolationLevel = (
  ilUnspecified,
  ilChaos,
  ilReadUncommitted,
  ilBrowse,
  ilCursorStability,
  ilReadCommitted,
  ilRepeatableRead,
  ilSerializable,
  ilIsolated
);
```

C++

```
enum TIsolationLevel {
  ilUnspecified,
  ilChaos,
  ilReadUncommitted,
  ilBrowse,
  ilCursorStability,
  ilReadCommitted,
  ilRepeatableRead,
  ilSerializable,
  ilIsolated
};
```

File

ADODB

Description

Use TIsolationLevel values to specify the transaction isolation level for a connection.

Note: it is possible that a server will force an isolation level other than that requested.

The constants that make up the TIsolationLevel type correspond directly to the ADO IsolationLevelEnum values with the same names. These values are discussed in the Microsoft Data Access SDK help, in the topic for the IsolationLevel property of the

ADO Connection object. See the SDK for additional information on the ADO constants and their effects.

The constants that make up the TADOLockType (see page 1289) type correspond directly to the LockTypeEnum values with similar names and have the same effects. These values are discussed in the Microsoft Data Access SDK help, in the topic for the LockType property of the ADO Recordset object. See the SDK for additional information on the ADO constants and their effects.

TIsolationLevel consists of the nine constants summarized in the following table:

Isolation Level	Meaning
ilUnspecified	Server is using a an isolation level other than what was requested and the specific isolation level cannot be determined.
ilChaos	Changes from more highly isolated transactions cannot be overwritten by the current connection.
ilReadUncommitted	Uncommitted changes in other transactions are visible.
ilBrowse	Uncommitted changes in other transactions are visible.
ilCursorStability	Changes from other transactions only visible after being committed.
ilReadCommitted	Changes from other transactions only visible after being committed.
ilRepeatableRead	Changes made in other transactions not visible, but requerying can retrieve new recordsets.
ilSerializable	Transactions conducted in isolation from other transactions.
ilIsolated	Transactions conducted in isolation from other transactions.

Note: The constants ilSerializable and ilIsolated have the same effects. These two constants parallel the ADO constants adXactSerializable and adXactIsolated, which have the same effects.

See Also

TADOConnection

8.3.14 ADODB.TMarshalOption Enumeration

Pascal

```
TMarshalOption = (
  moMarshalAll,
  moMarshalModifiedOnly
);
```

C++

```
enum TMarshalOption {
  moMarshalAll,
  moMarshalModifiedOnly
};
```

File

ADODB

Description

This is record ADODB.TMarshalOption.

8.3.15 ADODB.TObjectState Enumeration

Pascal

```
TObjectState = (
  stClosed,
  stOpen,
  stConnecting,
  stExecuting,
  stFetching
);
```

C++

```
enum TObjectState {
  stClosed,
  stOpen,
  stConnecting,
  stExecuting,
  stFetching
};
```

File

ADODB

Description

This is record ADODB.TObjectState.

8.3.16 ADODB.TParameterAttribute Enumeration

Pascal

```
TParameterAttribute = (
  paSigned,
  paNullable,
  paLong
);
```

C++

```
enum TParameterAttribute {
  paSigned,
  paNullable,
  paLong
};
```

File

ADODB

Description

This is record ADODB.TParameterAttribute.

8.3.17 ADODB.TParameterDirection Enumeration

TParameterDirection specifies the direction of a parameter.

Pascal

```
TParameterDirection = (
  pdUnknown,
  pdInput,
  pdOutput,
  pdInputOutput,
  pdReturnValue
);
```

C++

```
enum TParameterDirection {
  pdUnknown,
  pdInput,
  pdOutput,
  pdInputOutput,
  pdReturnValue
};
```

File

ADODB

Description

Use a TParameterDirection value to specify whether a parameter is an input parameter, an output parameter, both, or conveys a return value.

The constants that make up the TParameterDirection type correspond directly to the ADO ParameterDirectionEnum constants with similar names used for the Direction property of the ADO Parameter object. For instance, the constant pdInput corresponds to the ADO constant adParamInput. For additional information on these ADO constants and their effects, see the Microsoft Data Access SDK help in the topic for the Direction property of the ADO Parameter object.

TParameterDirection consists of the five constants summarized in the following table:

Parameter Direction	Meaning
pdUnknown	Parameter direction is unknown or cannot be determined.
pdInput	Parameter can only be used to convey data to a stored procedure or SQL statement.
pdOutput	Parameter can only be used to convey data from a stored procedure or SQL statement.
pdInputOutput	Parameter can be used to convey data to or from a stored procedure or SQL statement.
pdReturnValue	Parameter is used to convey a return value.

See Also

TParameter

8.3.18 ADODB.TPersistFormat Enumeration

TPersistFormat values indicate the format for a saved recordset.

Pascal

```
TPersistFormat = (
  pfADTG,
  pfXML
);
```

C++

```
enum TPersistFormat {
  pfADTG,
  pfXML
};
```

File

ADODB

Description

Use TPersistFormat values to specify the format in which recordsets are saved to file. The value pfADTG indicates the file is saved in ADTG (Advanced Data Tablegram) format. The value pfXML saves the recordset in XML format (requires ADO 2.1 and above).

The constants that make up the TPersistFormat type correspond directly to the ADO PersistFormatEnum constants adPersistADTG and adPersistXML. PersistFormatEnum constants are used in the PersistFormat parameter of the Save method of the ADO Recordset object. For additional information on these ADO constants and their effects, see the Microsoft Data Access SDK help in the topic for the Direction property of the ADO Parameter object.

See Also

TCustomADONETConnector

TCustomADONETConnector

8.3.19 ADODB.TRecordStatus Enumeration

Pascal

```
TRecordStatus = (
  rsOK,
  rsNew,
  rsModified,
  rsDeleted,
  rsUnmodified,
  rsInvalid,
  rsMultipleChanges,
  rsPendingChanges,
  rsCanceled,
  rsCantRelease,
  rsConcurrencyViolation,
  rsIntegrityViolation,
  rsMaxChangesExceeded,
  rsObjectOpen,
  rsOutOfMemory,
  rsPermissionDenied,
  rsSchemaViolation,
  rsDBDeleted
);
```

C++

```
enum TRecordStatus {
```

```
    rsOK,
    rsNew,
    rsModified,
    rsDeleted,
    rsUnmodified,
    rsInvalid,
    rsMultipleChanges,
    rsPendingChanges,
    rsCanceled,
    rsCantRelease,
    rsConcurrencyViolation,
    rsIntegrityViolation,
    rsMaxChangesExceeded,
    rsObjectOpen,
    rsOutOfMemory,
    rsPermissionDenied,
    rsSchemaViolation,
    rsDBDeleted
  };
```

File

ADODB

Description

This is record ADODB.TRecordStatus.

8.3.20 ADODB.TSchemaInfo Enumeration

TSchemaInfo values indicate the type of schema information to retrieve.

Pascal

```
  TSchemaInfo = (
    siAsserts,
    siCatalogs,
    siCharacterSets,
    siCollations,
    siColumns,
    siCheckConstraints,
    siConstraintColumnUsage,
    siConstraintTableUsage,
    siKeyColumnUsage,
    siReferentialConstraints,
    siTableConstraints,
    siColumnsDomainUsage,
    siIndexes,
    siColumnPrivileges,
    siTablePrivileges,
    siUsagePrivileges,
    siProcedures,
    siSchemata,
    siSQLLanguages,
    siStatistics,
    siTables,
    siTranslations,
    siProviderTypes,
    siViews,
    siViewColumnUsage,
    siViewTableUsage,
    siProcedureParameters,
    siForeignKeys,
```

```
    siPrimaryKeys,
    siProcedureColumns,
    siDBInfoKeywords,
    siDBInfoLiterals,
    siCubes,
    siDimensions,
    siHierarchies,
    siLevels,
    siMeasures,
    siProperties,
    siMembers,
    siProviderSpecific
  );
```

C++

```
  enum TSchemaInfo {
    siAsserts,
    siCatalogs,
    siCharacterSets,
    siCollations,
    siColumns,
    siCheckConstraints,
    siConstraintColumnUsage,
    siConstraintTableUsage,
    siKeyColumnUsage,
    siReferentialConstraints,
    siTableConstraints,
    siColumnsDomainUsage,
    siIndexes,
    siColumnPrivileges,
    siTablePrivileges,
    siUsagePrivileges,
    siProcedures,
    siSchemata,
    siSQLLanguages,
    siStatistics,
    siTables,
    siTranslations,
    siProviderTypes,
    siViews,
    siViewColumnUsage,
    siViewTableUsage,
    siProcedureParameters,
    siForeignKeys,
    siPrimaryKeys,
    siProcedureColumns,
    siDBInfoKeywords,
    siDBInfoLiterals,
    siCubes,
    siDimensions,
    siHierarchies,
    siLevels,
    siMeasures,
    siProperties,
    siMembers,
    siProviderSpecific
  };
```

File

ADODB

Description

Use TSchemaInfo values to specify the type of schema information to retrieve from a provider.

The constants that make up the TSchemaInfo type correspond directly to the ADO constants used for the QueryType parameter

for the OpenSchema method of the ADO Connection object. For additional information on these ADO constants and their effects, see the Microsoft Data Access SDK help in the topic for the OpenSchema method of the ADO Connection object.

A TSchemaInfo value may be any of the constants below:

Schema Parameter Value	ADO Criteria Values
siProviderSpecific	The provider defines its own nonstandard schema queries outside those listed below.
siAsserts	CONSTRAINT_CATALOG
siCatalogs	CATALOG_NAME
siCharacterSets	CHARACTER_SET_CATALOG
siCollations	COLLATION_CATALOG
siColumns	TABLE_CATALOG
siCheckConstraints	CONSTRAINT_CATALOG
siConstraintColumnUsage	TABLE_CATALOG
siConstraintTableUsage	TABLE_CATALOG
siKeyColumnUsage	CONSTRAINT_CATALOG
siReferentialConstraints	CONSTRAINT_CATALOG
siTableConstraints	CONSTRAINT_CATALOG
siColumnsDomainUsage	DOMAIN_CATALOG
siIndexes	TABLE_CATALOG
siColumnPrivileges	TABLE_CATALOG
siTablePrivileges	TABLE_CATALOG
siUsagePrivileges	OBJECT_CATALOG
siProcedures	PROCEDURE_CATALOG
siSchemata	CATALOG_NAME
siSQLLanguages	None.
siStatistics	TABLE_CATALOG
siTables	TABLE_CATALOG
siTranslations	TRANSLATION_CATALOG
siProviderTypes	DATA_TYPE
siViews	TABLE_CATALOG
siViewColumnUsage	VIEW_CATALOG
siViewTableUsage	VIEW_CATALOG
siProcedureParameters	PROCEDURE_CATALOG
siForeignKeys	PK_TABLE_CATALOG
siPrimaryKeys	PK_TABLE_CATALOG
siProcedureColumns	PROCEDURE_CATALOG
siDBInfoKeywords	(Defined in OLE DB (see page 6725), but not supported by ADO.)
siDBInfoLiterals	(Defined in OLE DB (see page 6725), but not supported by ADO.)
siCubes	(Defined in OLE DB (see page 6725), but not supported by ADO.)

siDimensions	(Defined in OLE DB (see page 6725), but not supported by ADO.)
siHierarchies	(Defined in OLE DB (see page 6725), but not supported by ADO.)
siLevels	(Defined in OLE DB (see page 6725), but not supported by ADO.)
siMeasures	(Defined in OLE DB (see page 6725), but not supported by ADO.)
siProperties	(Defined in OLE DB (see page 6725), but not supported by ADO.)
siMembers	(Defined in OLE DB (see page 6725), but not supported by ADO.)

See Also

TADOConnection

8.3.21 ADODB.TSeekOption Enumeration

Pascal

```
TSeekOption = (
  soFirstEQ,
  soLastEQ,
  soAfterEQ,
  soAfter,
  soBeforeEQ,
  soBefore
);
```

C++

```
enum TSeekOption {
  soFirstEQ,
  soLastEQ,
  soAfterEQ,
  soAfter,
  soBeforeEQ,
  soBefore
};
```

File

ADODB

Description

This is record ADODB.TSeekOption.

8.3.22 ADODB.TXactAttribute Enumeration

TXactAttribute values specify automated transaction behavior.

Pascal

```
TXactAttribute = (
  xaCommitRetaining,
  xaAbortRetaining
);
```

C++

```
enum TXactAttribute {
  xaCommitRetaining,
  xaAbortRetaining
};
```

File

ADODB

Description

Use TXactAttribute to specify how the committing and aborting of a connection component's transactions is handled. A property of type TXactAttribute may contain one TXactAttribute constant, both, or none.

An xaCommitRetaining value specifies that the connection performs retaining commits. Committing a transaction automatically starts a new transaction.

An xaAbortRetaining value specifies that the connection performs retaining aborts. Rolling back a transaction automatically starts a new transaction.

The constants that make up the TXactAttributes (see page 1324) type correspond directly to the ADO XactAttributeEnum constants adXactCommitRetaining and adXactAbortRetaining. These ADO constants are in the Attributes property of the ADO Connection object. For additional information on these ADO constants and their effects, see the Microsoft Data Access SDK help in the topic for the Attributes property of the ADO Connection object.

See Also

TADOConnection

8.4 Types

The following table lists types in this documentation.

Types

Name	Description
_Command (see page 1310)	This is type ADODB._Command.
_Connection (see page 1311)	This is type ADODB._Connection.
_Parameter (see page 1311)	This is type ADODB._Parameter.
_Recordset (see page 1311)	This is type ADODB._Recordset.
Error (see page 1312)	This is type ADODB.Error.
Errors (see page 1312)	This is type ADODB.Errors.
Parameters (see page 1312)	This is type ADODB.Parameters.
Properties (see page 1313)	This is type ADODB.Properties.
Property_ (see page 1313)	This is type ADODB.Property_.
PVariantList (see page 1313)	This is type ADODB.PVariantList.
TBeginTransCompleteEvent (see page 1314)	This is type ADODB.TBeginTransCompleteEvent.
TConnectErrorEvent (see page 1314)	This is type ADODB.TConnectErrorEvent.
TCursorOptions (see page 1314)	This is type ADODB.TCursorOptions.
TDataType (see page 1315)	TDataType indicates the type of a parameter.
TDisconnectEvent (see page 1315)	This is type ADODB.TDisconnectEvent.
TEndOfRecordsetEvent (see page 1315)	This is type ADODB.TEndOfRecordsetEvent.
TExecuteCompleteEvent (see page 1316)	This is type ADODB.TExecuteCompleteEvent.
TExecuteOptions (see page 1316)	TExecuteOptions values specify the characteristics of an execution operation.
TFetchProgressEvent (see page 1317)	This is type ADODB.TFetchProgressEvent.
TFieldChangeCompleteEvent (see page 1317)	This is type ADODB.TFieldChangeCompleteEvent.
TInfoMessageEvent (see page 1317)	This is type ADODB.TInfoMessageEvent.
TObjectStates (see page 1318)	TObjectStates values indicate the current state of an ADO object.
TParameterAttributes (see page 1318)	TParameterAttributes values indicate the types of values a parameter can accept.
TPropList (see page 1319)	This is type ADODB.TPropList.
TRecordChangeCompleteEvent (see page 1319)	This is type ADODB.TRecordChangeCompleteEvent.
TRecordsetCreate (see page 1320)	This is type ADODB.TRecordsetCreate.
TRecordsetErrorEvent (see page 1320)	This is type ADODB.TRecordsetErrorEvent.
TRecordsetEvent (see page 1320)	This is type ADODB.TRecordsetEvent.
TRecordsetReasonEvent (see page 1321)	This is type ADODB.TRecordsetReasonEvent.
TRecordStatusSet (see page 1321)	TRecordStatusSet values indicate the status of the current record relative to batch updates or other bulk operations
TVariantList (see page 1322)	This is type ADODB.TVariantList.
TWillChangeFieldEvent (see page 1322)	This is type ADODB.TWillChangeFieldEvent.
TWillChangeRecordEvent (see page 1323)	This is type ADODB.TWillChangeRecordEvent.
TWillConnectEvent (see page 1323)	This is type ADODB.TWillConnectEvent.
TWillExecuteEvent (see page 1323)	This is type ADODB.TWillExecuteEvent.
TXactAttributes (see page 1324)	This is type ADODB.TXactAttributes.

8.4.1 ADODB._Command Type

Pascal

```
_Command = ADOInt._Command;
```

C++

```
ADOInt._Command _Command;
```

File

ADODB

Description

This is type ADODB._Command.

8.4.2 ADODB._Connection Type

Pascal

```
_Connection = ADOInt._Connection;
```

C++

```
ADOInt._Connection _Connection;
```

File

ADODB

Description

This is type ADODB._Connection.

8.4.3 ADODB._Parameter Type

Pascal

```
_Parameter = ADOInt._Parameter;
```

C++

```
ADOInt._Parameter _Parameter;
```

File

ADODB

Description

This is type ADODB._Parameter.

8.4.4 ADODB._Recordset Type

Pascal

```
_Recordset = ADOInt.Recordset;
```

C++

```
ADOInt.Recordset _Recordset;
```

File

ADODB

Description

This is type ADODB._Recordset.

8.4.5 ADODB.Error Type

Pascal

```
Error = ADOInt.Error;
```

C++

```
ADOInt.Error Error;
```

File

ADODB

Description

This is type ADODB.Error.

8.4.6 ADODB.Errors Type

Pascal

```
Errors = ADOInt.Errors;
```

C++

```
ADOInt.Errors Errors;
```

File

ADODB

Description

This is type ADODB.Errors.

8.4.7 ADODB.Parameters Type

Pascal

```
Parameters = ADOInt.Parameters;
```

C++

```
ADOInt.Parameters Parameters;
```

File

ADODB

Description

This is type ADODB.Parameters.

8.4.8 ADODB.Properties Type

Pascal

```
Properties = ADOInt.Properties;
```

C++

```
ADOInt.Properties Properties;
```

File

ADODB

Description

This is type ADODB.Properties.

8.4.9 ADODB.Property_ Type

Pascal

```
Property_ = ADOInt.Property_;
```

C++

```
ADOInt.Property_ Property_;
```

File

ADODB

Description

This is type ADODB.Property_.

8.4.10 ADODB.PVariantList Type

Pascal

```
PVariantList = ^TVariantList;
```

C++

```
^TVariantList PVariantList;
```

File

ADODB

Description

This is type ADODB.PVariantList.

8.4.11 ADODB.TBeginTransCompleteEvent Type

Pascal

```
TBeginTransCompleteEvent = procedure (Connection: TADOConnection; TransactionLevel: Integer;
const Error: Error; var EventStatus: TEventStatus) of object;
```

C++

```
(Connection: TADOConnection; TransactionLevel: Integer; const Error: Error; var EventStatus:
TEventStatus) ( TBeginTransCompleteEvent)();
```

File

ADODB

Description

This is type ADODB.TBeginTransCompleteEvent.

8.4.12 ADODB.TConnectErrorEvent Type

Pascal

```
TConnectErrorEvent = procedure (Connection: TADOConnection; const Error: Error; var
EventStatus: TEventStatus) of object;
```

C++

```
(Connection: TADOConnection; const Error: Error; var EventStatus: TEventStatus) (
TConnectErrorEvent)();
```

File

ADODB

Description

This is type ADODB.TConnectErrorEvent.

8.4.13 ADODB.TCursorOptions Type

Pascal

```
TCursorOptions = set of TCursorOption;
```

C++

```
TCursorOption TCursorOptions;
```

File

ADODB

Description

This is type ADODB.TCursorOptions.

8.4.14 ADODB.TDataType Type

TDataType indicates the type of a parameter.

Pascal

```
TDataType = TFieldType;
```

C++

```
TFieldType TDataType;
```

File

ADODB

Description

TDataType represents a field type when it is applied to a parameter.

8.4.15 ADODB.TDisconnectEvent Type

Pascal

```
TDisconnectEvent = procedure (Connection: TADOConnection; var EventStatus: TEventStatus) of
object;
```

C++

```
(Connection: TADOConnection; var EventStatus: TEventStatus) ( TDisconnectEvent)();
```

File

ADODB

Description

This is type ADODB.TDisconnectEvent.

8.4.16 ADODB.TEndOfRecordsetEvent Type

Pascal

```
TEndOfRecordsetEvent = procedure (DataSet: TCustomADODataSet; var MoreData: WordBool; var
EventStatus: TEventStatus) of object;
```

C++

```
(DataSet: TCustomADODataSet; var MoreData: WordBool; var EventStatus: TEventStatus) (
TEndOfRecordsetEvent)();
```

File

ADODB

Description

This is type ADODB.TEndOfRecordsetEvent.

8.4.17 ADODB.TExecuteCompleteEvent Type

Pascal

```
TExecuteCompleteEvent = procedure (Connection: TADOConnection; RecordsAffected: Integer; const
Error: Error; var EventStatus: TEventStatus; const Command: _Command; const Recordset:
_Recordset) of object;
```

C++

```
(Connection: TADOConnection; RecordsAffected: Integer; const Error: Error; var EventStatus:
TEventStatus; const Command: _Command; const Recordset: _Recordset) ( TExecuteCompleteEvent)();
```

File

ADODB

Description

This is type ADODB.TExecuteCompleteEvent.

8.4.18 ADODB.TExecuteOptions Type

TExecuteOptions values specify the characteristics of an execution operation.

Pascal

```
TExecuteOptions = set of TExecuteOption;
```

C++

```
TExecuteOption TExecuteOptions;
```

File

ADODB

Description

When executing a command or connection using a method, use TExecuteOptions values to specify the characteristics of the execution operation. When reacting to a command or connection execution in an event handler, read the execute options parameter to determine the options already established for the connection triggering the event.

TExecuteOptions consists of the constants summarized in the following table:

Execute Option	Meaning
eoAsyncExecute	The command is executed asynchronously.
eoAsyncFetch	The command fetches remaining rows after the initial quantity specified in the Cache property asynchronously.
eoAsyncFetchNonBlocking	Execution is performed without blocking the thread.
eoExecuteNoRecords	A command or stored procedure that does not return. If any rows are retrieved, they are discarded and not returned.

See Also

TCustomADONETConnector

TADOCommand

8.4.19 ADODB.TFetchProgressEvent Type

Pascal

```
TFetchProgressEvent = procedure (DataSet: TCustomADODataSet; Progress, MaxProgress: Integer;
var EventStatus: TEventStatus) of object;
```

C++

```
(DataSet: TCustomADODataSet; Progress, MaxProgress: Integer; var EventStatus: TEventStatus) (
TFetchProgressEvent)();
```

File

ADODB

Description

This is type ADODB.TFetchProgressEvent.

8.4.20 ADODB.TFieldChangeCompleteEvent Type

Pascal

```
TFieldChangeCompleteEvent = procedure (DataSet: TCustomADODataSet; const FieldCount: Integer;
const Fields: OleVariant; const Error: Error; var EventStatus: TEventStatus) of object;
```

C++

```
(DataSet: TCustomADODataSet; const FieldCount: Integer; const Fields: OleVariant; const Error:
Error; var EventStatus: TEventStatus) ( TFieldChangeCompleteEvent)();
```

File

ADODB

Description

This is type ADODB.TFieldChangeCompleteEvent.

8.4.21 ADODB.TInfoMessageEvent Type

Pascal

```
TInfoMessageEvent = procedure (Connection: TADOConnection; const Error: Error; var
EventStatus: TEventStatus) of object;
```

C++

```
(Connection: TADOConnection; const Error: Error; var EventStatus: TEventStatus) (
TInfoMessageEvent)();
```

File

ADODB

Description

This is type ADODB.TInfoMessageEvent.

8.4.22 ADODB.TObjectStates Type

TObjectStates values indicate the current state of an ADO object.

Pascal

```
TObjectStates = set of TObjectState;
```

C++

```
TObjectState TObjectStates;
```

File

ADODB

Description

Use TObjectStates values to determine the current activity status of a connection or recordset.

The constants that make up the TObjectStates type correspond directly to the similarly named constants used for the State property of the ADO Connection, Command, and Recordset objects. For additional information on these ADO constants and their effects, see the Microsoft Data Access SDK help in the topic for the State property.

TObjectStates consists of the five constants summarized in the following table:

Object State	Meaning
stClosed	Connection object is inactive and not connected to a database.
stOpen	Connection object is inactive, but connected to a database.
stConnecting	Connection object is in the process of connecting to a database.
stExecuting	Connection object is currently executing.
stFetching	Connection object is retrieving data from the database.

See Also

TADOCommand

TADOConnection

TCustomADONETConnector

8.4.23 ADODB.TParameterAttributes Type

TParameterAttributes values indicate the types of values a parameter can accept.

Pascal

```
TParameterAttributes = set of TParameterAttribute;
```

C++

```
TParameterAttribute TParameterAttributes;
```

File

ADODB

Description

Use TParameterAttributes values to specify the characteristics of a parameter that control the types of values a parameter can accept.

The constants that make of the TParameterAttributes type correspond to the ADO ParameterAttributesEnum values of similar names. For instance, the TParameterAttributes type constant paSigned corresponds to the ADO constant adParamSigned. For additional information on these ADO constants and their effects, see the Microsoft Data Access SDK help in the topic for the Attributes property of the ADO Parameter object.

TParameterAttributes consists of the three constants summarized in the following table:

Parameter Attribute	Meaning
psSigned	The parameter accepts signed values.
psNullable	The parameter accepts NULL values.
psLong	The parameter accepts long binary data.

See Also

TParameter

8.4.24 ADODB.TPropList Type

Pascal

```
TPropList = array of PPropInfo;
```

C++

```
array of PPropInfo TPropList;
```

File

ADODB

Description

This is type ADODB.TPropList.

8.4.25 ADODB.TRecordChangeCompleteEvent Type

Pascal

```
TRecordChangeCompleteEvent = procedure (DataSet: TCustomADODataSet; const Reason:
TEventReason; const RecordCount: Integer; const Error: Error; var EventStatus: TEventStatus)
of object;
```

C++

```
(DataSet: TCustomADODataSet; const Reason: TEventReason; const RecordCount: Integer; const
Error: Error; var EventStatus: TEventStatus) ( TRecordChangeCompleteEvent)();
```

File

ADODB

Description

This is type ADODB.TRecordChangeCompleteEvent.

8.4.26 ADODB.TRecordsetCreate Type

Pascal

```
TRecordsetCreate = procedure (DataSet: TCustomADODataSet; const Recordset: _Recordset) of
object;
```

C++

```
(DataSet: TCustomADODataSet; const Recordset: _Recordset) ( TRecordsetCreate)();
```

File

ADODB

Description

This is type ADODB.TRecordsetCreate.

8.4.27 ADODB.TRecordsetErrorEvent Type

Pascal

```
TRecordsetErrorEvent = procedure (DataSet: TCustomADODataSet; const Reason: TEventReason;
const Error: Error; var EventStatus: TEventStatus) of object;
```

C++

```
(DataSet: TCustomADODataSet; const Reason: TEventReason; const Error: Error; var EventStatus:
TEventStatus) ( TRecordsetErrorEvent)();
```

File

ADODB

Description

This is type ADODB.TRecordsetErrorEvent.

8.4.28 ADODB.TRecordsetEvent Type

Pascal

```
TRecordsetEvent = procedure (DataSet: TCustomADODataSet; const Error: Error; var EventStatus:
TEventStatus) of object;
```

C++

```
(DataSet: TCustomADODataSet; const Error: Error; var EventStatus: TEventStatus) (
TRecordsetEvent)();
```

File

ADODB

Description

This is type ADODB.TRecordsetEvent.

8.4.29 ADODB.TRecordsetReasonEvent Type

Pascal

```
TRecordsetReasonEvent = procedure (DataSet: TCustomADODataSet; const Reason: TEventReason; var
EventStatus: TEventStatus) of object;
```

C++

```
(DataSet: TCustomADODataSet; const Reason: TEventReason; var EventStatus: TEventStatus) (
TRecordsetReasonEvent)();
```

File

ADODB

Description

This is type ADODB.TRecordsetReasonEvent.

8.4.30 ADODB.TRecordStatusSet Type

TRecordStatusSet values indicate the status of the current record relative to batch updates or other bulk operations

Pascal

```
TRecordStatusSet = set of TRecordStatus;
```

C++

```
TRecordStatus TRecordStatusSet;
```

File

ADODB

Description

Use TRecordStatusSet values to determine the status of the current row.

The constants that make up the TRecordStatus (see page 1304) type correspond directly to the ADO RecordStatusEnum values used for the Status property of the ADO Recordset object. For additional information on these ADO constants and their effects, see the Microsoft Data Access SDK help in the topic for the Status property of the ADO Recordset object.

A TRecordStatusSet value may contain any one or more of the following constants:

Record Status Set Constant	Meaning
rsOK	The row was successfully updated
rsNew	The row is a new insert.

rsModified	The row has been modified.
rsDeleted	The row has been deleted.
rsUnmodified	The row has not been modified since being retrieved.
rsInvalid	The row was not saved due because the bookmark was invalid.
rsMultipleChanges	The row was not saved because it would have affected multiple rows.
rsPendingChanges	The row was not saved because it refers to a pending insert.
rsCanceled	The row was not saved because the operation was canceled.
rsCantRelease	The new row was not saved because of existing row locks.
rsConcurrencyViolation	The row was not saved because optimistic concurrency was in use.
rsIntegrityViolation	The row was not saved because the user violated integrity constraints.
rsMaxChangesExceeded	The row was not saved because there were too many pending changes.
rsObjectOpen	The row was not saved because of a conflict with an open storage object.
rsOutOfMemory	The row was not saved because the computer has run out of memory.
rsPermissionDenied	The row was not saved because the user has insufficient permissions.
rsSchemaViolation	The row was not saved because it violates the structure of the underlying database.
rsDBDeleted	The row has already been deleted from the data source.

See Also

TCustomADONETConnector

8.4.31 ADODB.TVariantList Type

Pascal

```
TVariantList = array[0..0] of OleVariant;
```

C++

```
array[0..0] of OleVariant TVariantList;
```

File

ADODB

Description

This is type ADODB.TVariantList.

8.4.32 ADODB.TWillChangeFieldEvent Type

Pascal

```
TWillChangeFieldEvent = procedure (DataSet: TCustomADODataSet; const FieldCount: Integer;
const Fields: OleVariant; var EventStatus: TEventStatus) of object;
```

C++

```
(DataSet: TCustomADODataSet; const FieldCount: Integer; const Fields: OleVariant; var
```

```
EventStatus: TEventStatus) ( TWillChangeFieldEvent)();
```

File

ADODB

Description

This is type ADODB.TWillChangeFieldEvent.

8.4.33 ADODB.TWillChangeRecordEvent Type

Pascal

```
TWillChangeRecordEvent = procedure (DataSet: TCustomADODataSet; const Reason: TEventReason;
const RecordCount: Integer; var EventStatus: TEventStatus) of object;
```

C++

```
(DataSet: TCustomADODataSet; const Reason: TEventReason; const RecordCount: Integer; var
EventStatus: TEventStatus) ( TWillChangeRecordEvent)();
```

File

ADODB

Description

This is type ADODB.TWillChangeRecordEvent.

8.4.34 ADODB.TWillConnectEvent Type

Pascal

```
TWillConnectEvent = procedure (Connection: TADOConnection; var ConnectionString, UserID,
Password: WideString; var ConnectOptions: TConnectOption; var EventStatus: TEventStatus) of
object;
```

C++

```
(Connection: TADOConnection; var ConnectionString, UserID, Password: WideString; var
ConnectOptions: TConnectOption; var EventStatus: TEventStatus) ( TWillConnectEvent)();
```

File

ADODB

Description

This is type ADODB.TWillConnectEvent.

8.4.35 ADODB.TWillExecuteEvent Type

Pascal

```
TWillExecuteEvent = procedure (Connection: TADOConnection; var CommandText: WideString; var
CursorType: TCursorType; var LockType: TADOLockType; var CommandType: TCommandType; var
ExecuteOptions: TExecuteOptions; var EventStatus: TEventStatus; const Command: _Command; const
Recordset: _Recordset) of object;
```

C++

```
(Connection: TADOConnection; var CommandText: WideString; var CursorType: TCursorType; var
LockType: TADOLockType; var CommandType: TCommandType; var ExecuteOptions: TExecuteOptions;
var EventStatus: TEventStatus; const Command: _Command; const Recordset: _Recordset) (
TWillExecuteEvent)();
```

File

ADODB

Description

This is type ADODB.TWillExecuteEvent.

8.4.36 ADODB.TXactAttributes Type

Pascal

```
TXactAttributes = set of TXactAttribute;
```

C++

```
TXactAttribute TXactAttributes;
```

File

ADODB

Description

This is type ADODB.TXactAttributes.

9 AppEvnts Namespace

This is namespace AppEvnts.

Topics

Name	Description
Classes (see page 1326)	The following table lists classes in this documentation.

Classes

	Name	Description
	TApplicationEvents (see page 1326)	TApplicationEvents intercepts application-level events.
	TCustomApplicationEvents (see page 1347)	TCustomApplicationEvents intercepts application-level events.

9.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TApplicationEvents (see page 1326)	TApplicationEvents intercepts application-level events.
	TCustomApplicationEvents (see page 1347)	TCustomApplicationEvents intercepts application-level events.

9.1.1 TApplicationEvents Class

TApplicationEvents intercepts application-level events.

Class Hierarchy

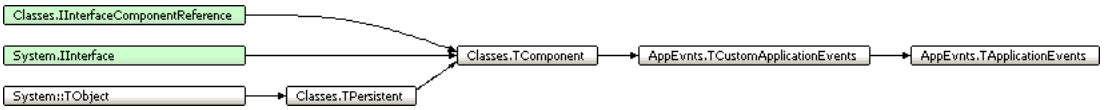

Pascal

```
TApplicationEvents = class(TCustomApplicationEvents);
```

C++

```
class TApplicationEvents : public TCustomApplicationEvents;
```

File

AppEvnts

Description

Use TApplicationEvents to intercept the events of the global Application object. When you add a TApplicationEvents object to a form, the Application object forwards all events to the TApplicationEvents object. Thus, each event of the TApplicationEvents object is the same as the event with the same name on the Application object.

Each form in an application can have its own TApplicationEvents object. Each application event occurs for all the TApplicationEvents objects in the project. To change the order in which the different TApplicationEvents objects receive events, use the Activate (see page 1351) method. To prevent other TApplicationEvents objects from receiving a specific event, use the CancelDispatch (see page 1351) method.

See Also

TApplication (see page 13660)

Application

9.1.1.1 TApplicationEvents Members

The following tables list the members exposed by TApplicationEvents.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.

	Activate (see page 1351)	Ensures that this is the first component to receive any application events.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CancelDispatch (see page 1351)	Prevents other TCustomApplicationEvents (see page 1347) objects from receiving the current event.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 1350)	Creates a new TCustomApplicationEvents (see page 1347) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnActionExecute (see page 1330)	Occurs when an action's Execute method is called and its action list has not already handled it.
	OnActionUpdate (see page 1331)	Occurs when an action's Update method is called and its action list has not already handled it.
	OnActivate (see page 1333)	Occurs when an application becomes active.
	OnDeactivate (see page 1334)	Occurs when an application becomes inactive.
	OnException (see page 1335)	Occurs when an unhandled exception occurs in the application.
	OnHelp (see page 1336)	Occurs when the application receives a request for help.
	OnHint (see page 1337)	Occurs when the mouse pointer moves over a control or menu item that can display a Help Hint.
	OnIdle (see page 1339)	Occurs when an application becomes idle.
	OnMessage (see page 1340)	Occurs when the application receives a Windows message.
	OnMinimize (see page 1341)	Occurs when an application is minimized.
	OnModalBegin (see page 1342)	This is OnModalBegin, a member of class TApplicationEvents.
	OnModalEnd (see page 1342)	This is OnModalEnd, a member of class TApplicationEvents.
	OnRestore (see page 1342)	Occurs when the previously minimized application is restored to its normal size.
	OnSettingChange (see page 1343)	Occurs when Windows notifies the application that a system-wide setting has changed.
	OnShortCut (see page 1344)	Occurs when the user presses a key (before the OnKeyDown event).
	OnShowHint (see page 1345)	Occurs when the application is about to display the hint window for a Help Hint.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

9.1.1.2 TApplicationEvents Events

The events of the TApplicationEvents class are listed here.

Published Events

	Name	Description
	OnActionExecute (see page 1330)	Occurs when an action's Execute method is called and its action list has not already handled it.
	OnActionUpdate (see page 1331)	Occurs when an action's Update method is called and its action list has not already handled it.
	OnActivate (see page 1333)	Occurs when an application becomes active.
	OnDeactivate (see page 1334)	Occurs when an application becomes inactive.
	OnException (see page 1335)	Occurs when an unhandled exception occurs in the application.
	OnHelp (see page 1336)	Occurs when the application receives a request for help.
	OnHint (see page 1337)	Occurs when the mouse pointer moves over a control or menu item that can display a Help Hint.
	OnIdle (see page 1339)	Occurs when an application becomes idle.
	OnMessage (see page 1340)	Occurs when the application receives a Windows message.
	OnMinimize (see page 1341)	Occurs when an application is minimized.
	OnModalBegin (see page 1342)	This is OnModalBegin, a member of class TApplicationEvents.
	OnModalEnd (see page 1342)	This is OnModalEnd, a member of class TApplicationEvents.
	OnRestore (see page 1342)	Occurs when the previously minimized application is restored to its normal size.
	OnSettingChange (see page 1343)	Occurs when Windows notifies the application that a system-wide setting has changed.
	OnShortCut (see page 1344)	Occurs when the user presses a key (before the OnKeyDown event).
	OnShowHint (see page 1345)	Occurs when the application is about to display the hint window for a Help Hint.

9.1.1.2.1 TApplicationEvents.OnActionExecute Event

Occurs when an action's Execute method is called and its action list has not already handled it.

Pascal

```
property OnActionExecute: TActionEvent;
```

C++

```
__property TActionEvent OnActionExecute;
```

Description

Use the OnExecute event handler to respond when a user invokes actions whose action lists do not have OnExecute event handlers.

If the action list that contains the action does not handle it in an OnExecute event handler, then the action is routed to the Application object's OnActionExecute event. The application events object hooks this event and responds with its own OnActionExecute event handler.

The Handled parameter of the event handler returns false by default. If the handler handles the event, it should change Handled to true, thereby preventing further attempts to handle the action. When the event handler exits with Handled set to false, the event first goes to any other application events objects, and then the action's OnExecute event occurs. If the action remains unhandled after that, the active control's ExecuteAction method is called to allow the action to execute with an identified target. Finally, the active form's ExecuteAction method is called if all other handlers do not handle the action.

Note: Call the CancelDispatch method from an OnActionExecute event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents, a TActionList
and a TListBox in the form.  Select the TApplicationEvents,
```

```
double click on the OnActionExecute event and add the
following code to the handler.  Create an TAction named
MyAction in a TActionList.  The TActionList must have no
OnExecute event handler, and the TAction has an OnExecute
event handler called MyActionExecute.  Then set an object's
(such as a button) Action to MyAction.  The Handled
parameter of the event handler returns false by default. If
the handler handles the event, it should change Handled to
true, thereby preventing further attempts to handle the
action.
*/
void __fastcall TAppEventForm::ApplicationEventsActionExecute(
      TBasicAction *Action, bool &Handled)
{
    lbOther->Items->Add("Event OnActionExecute");
}

void __fastcall TAppEventForm::ActionExecute(TObject *Sender)
{
    ShowMessage("Action executed");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents, a TActionList
and a TListBox in the form.  Select the TApplicationEvents,
double click on the OnActionExecute event and add the
following code to the handler.  Create an TAction named
MyAction in a TActionList.  The TActionList must have no
OnExecute event handler, and the TAction has an OnExecute
event handler called MyActionExecute.  Then set an object's
(such as a button) Action to MyAction.  The Handled
parameter of the event handler returns false by default. If
the handler handles the event, it should change Handled to
true, thereby preventing further attempts to handle the
action.
}
procedure TMainForm.ApplicationEventsActionExecute(Action: TBasicAction;
  var Handled: Boolean);
begin
  lbOther.Items.Add('Event OnActionExecute');
end;

procedure TMainForm.MyActionExecute(Sender: TObject);
begin
  ShowMessage('Action executed');
end;
```

See Also

OnActionUpdate (see page 1354)

ExecuteAction (see page 13673)

OnExecute (see page 2716)

OnExecute (see page 327)

ExecuteAction

CancelDispatch (see page 1351)

9.1.1.2.2 TApplicationEvents.OnActionUpdate Event

Occurs when an action's Update method is called and its action list has not already handled it.

Pascal

```
property OnActionUpdate: TActionEvent;
```

C++

```
__property TActionEvent OnActionUpdate;
```

Description

Use the OnActionUpdate event handler to update the properties of an action when the application is idle if the action's action list does not handle it in an OnUpdate event handler. OnActionUpdate occurs after the OnIdle event.

If the action list that contains the action does not update it in an OnUpdate event handler, the action is routed to the Application object's OnActionUpdate event. The application events object hooks this event and responds with its own OnActionUpdate event handler.

The Handled parameter of the event handler returns false by default. If the handler updates the event, it should change Handled to true, thereby ending the processing of the action. When the event handler exits with Handled set to false, the event first goes to any other application events objects, and then the action's OnUpdate event occurs. If the action is not updated after that, the active control's UpdateAction method is called to allow the target to update the action. Finally, the active form's UpdateAction method is called if all other handlers do not handle the action.

Note: Call the CancelDispatch method from an OnActionUpdate event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnActionUpdate event and add the following code to
the handler.  Create other TApplicationEvents the same way.
When these events update this event is triggered.  The
Handled parameter of the event handler returns false by
default. If the handler handles the event, it should change
Handled to true, thereby preventing further attempts to
*/
void __fastcall TAppEventForm::ApplicationEventsActionUpdate(
      TBasicAction *Action, bool &Handled)
{
     lbActionUpdate->Items->Add("Event OnActionUpdate");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnActionUpdate event and add the following code to
the handler.  Create other TApplicationEvents the same way.
When these events update this event is triggered.  The
Handled parameter of the event handler returns false by
default. If the handler handles the event, it should change
Handled to true, thereby preventing further attempts to
handle the action.
}
procedure TMainForm.ApplicationEventsActionUpdate(Action: TBasicAction;
  var Handled: Boolean);
begin
  lbActionUpdate.Items.Add('Event OnActionUpdate');
end;
```

See Also

OnActionExecute (see page 1352)

UpdateAction (see page 13701)

OnUpdate (see page 2718)

OnUpdate (see page 327)

UpdateAction

OnIdle (see page 1361)

CancelDispatch (see page 1351)

9.1.1.2.3 TApplicationEvents.OnActivate Event

Occurs when an application becomes active.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to perform special processing when the application becomes active.

An application becomes active when it is initially run or when focus moves from another Windows application back to any window of the application.

Note: Call the CancelDispatch method from an OnActivate event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnActivate event and add the following code
to the handler.  These events occur when the form gains
or loses focus.
*/
void __fastcall TAppEventForm::ApplicationEventsActivate(TObject *Sender)
{
     lbOther->Items->Add("Event OnActivate");
}

void __fastcall TAppEventForm::ApplicationEventsDeactivate(TObject *Sender)
{
    lbOther->Items->Add("Event OnDeactivate");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnActivate event and add the following code
to the handler.  These events occur when the form gains
or loses focus.
```

```
}
procedure TMainForm.ApplicationEventsActivate(Sender: TObject);
begin
  lbOther.Items.Add('Event OnActivate');
end;

procedure TMainForm.ApplicationEventsDeactivate(Sender: TObject);
begin
  lbOther.Items.Add('Event OnDeactivate');
end;
```

See Also

Active (see page 13704)

OnDeactivate (see page 1356)

CancelDispatch (see page 1351)

9.1.1.2.4 TApplicationEvents.OnDeactivate Event

Occurs when an application becomes inactive.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to perform any special processing that should occur immediately before the application is deactivated. The OnDeactivate event occurs when the user switches from the application to another Windows application.

Note: Call the CancelDispatch method from an OnDeactivate event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnActivate event and add the following code
to the handler.  These events occur when the form gains
or loses focus.
*/
void __fastcall TAppEventForm::ApplicationEventsActivate(TObject *Sender)
{
     lbOther->Items->Add("Event OnActivate");
}

void __fastcall TAppEventForm::ApplicationEventsDeactivate(TObject *Sender)
{
    lbOther->Items->Add("Event OnDeactivate");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnActivate event and add the following code
to the handler.  These events occur when the form gains
```

```
or loses focus.
}
procedure TMainForm.ApplicationEventsActivate(Sender: TObject);
begin
  lbOther.Items.Add('Event OnActivate');
end;

procedure TMainForm.ApplicationEventsDeactivate(Sender: TObject);
begin
  lbOther.Items.Add('Event OnDeactivate');
end;
```

See Also

Active (see page 13704)

OnActivate (see page 1355)

CancelDispatch (see page 1351)

9.1.1.2.5 TApplicationEvents.OnException Event

Occurs when an unhandled exception occurs in the application.

Pascal

```
property OnException: TExceptionEvent;
```

C++

```
__property TExceptionEvent OnException;
```

Description

Use OnException to change the default behavior that occurs when an exception is not handled by application code. The OnException event handler is called automatically in the application's HandleException method.

OnException only handles exceptions that occur during message processing. Exceptions that occur before or after the execution of the application's Run method do not generate OnException events.

If an exception passes through the try blocks in the application code, the application automatically calls the HandleException method. Unless the exception object is EAbort, HandleException calls the OnException handler, if one exists. Otherwise, it calls ShowException to display a message box indicating an error occurred.

Note: Call the CancelDispatch method from an OnException event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnException event and add the following code
to the handler.  Trigger this event by calling the
Exception Create constructor.
*/
void __fastcall TAppEventForm::ApplicationEventsException(TObject *Sender,
      Exception *E)
{
    lbOther->Items->Add("Event OnException: " + E->Message);
}

void __fastcall TAppEventForm::MenuExceptionClick(TObject *Sender)
{
```

```
    char *sExceptionRaised = "This is an exception";
    throw Exception(sExceptionRaised);
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnException event and add the following code
to the handler.  Trigger this event by calling the
Exception Create constructor.
}
procedure TMainForm.ApplicationEventsException(Sender: TObject;
  E: Exception);
begin
  lbOther.Items.Add('Event OnException: ' + E.Message);
end;

procedure TMainForm.MenuExceptionClick(Sender: TObject);
resourcestring
  sExceptionRaised = 'This is an exception';

begin
  raise Exception.Create(sExceptionRaised);
end;
```

See Also

HandleException (see page 13674)

ShowException (see page 13698)

CancelDispatch (see page 1351)

9.1.1.2.6 TApplicationEvents.OnHelp Event

Occurs when the application receives a request for help.

Pascal

```
property OnHelp: THelpEvent;
```

C++

```
__property THelpEvent OnHelp;
```

Description

Write an OnHelp event handler to perform special processing when the user requests help. The HelpContext and the HelpJump methods automatically trigger the OnHelp event.

Set CallHelp to true to have the application call WinHelp after the event. Set CallHelp to false to prevent the application from calling WinHelp. All application help methods go through OnHelp. The application calls WinHelp only if OnHelp's CallHelp parameter returns True, true or if no OnHelp event handler is assigned.

To find the possible values of the Command and Data parameters, search for WinHelp in the Win32 Developer's Reference Help (Win32.HLP) file, which explains the WinHelp API (application programming interface). The possible values for the Data parameter depend upon the value of the Command parameter.

The return value is true if the event handler succeeds, false if it fails.

Note: Call the CancelDispatch method from an OnHelp event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnHelp event and add the following code to the
handler.  Trigger this event by pressing the F1 key or
call the HelpContext method.  Set CallHelp to true if the
application should still invoke the help system after the
event.
*/
bool __fastcall TAppEventForm::ApplicationEventsHelp(WORD Command, int Data,
      bool &CallHelp)
{
    lbOther->Items->Add("Event OnHelp");
    return false;
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnHelp event and add the following code to the
handler.  Trigger this event by pressing the F1 key or
call the HelpContext method.  Set CallHelp to true if the
application should still invoke the help system after the
event.
}
function TMainForm.ApplicationEventsHelp(Command: Word; Data: Integer;
  var CallHelp: Boolean): Boolean;
begin
  lbOther.Items.Add('Event OnHelp');
  CallHelp := False;
end;
```

See Also

HelpCommand (see page 13678)

HelpContext (see page 13679)

HelpFile (see page 13711)

CancelDispatch (see page 1351)

HelpJump (see page 13679)

9.1.1.2.7 TApplicationEvents.OnHint Event

Occurs when the mouse pointer moves over a control or menu item that can display a Help Hint.

Pascal

```
property OnHint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnHint;
```

Description

Write an OnHint event handler to perform special processing when the mouse pauses over a control or menu item whose Hint property is not an empty string (' "" ').

A common use of the OnHint event is to display the value of a control or menu item's Hint property as the text of a status bar (TStatusBar). The Hint property of a control can specify both a short Help Hint and a (usually) longer hint that appears elsewhere

because of code in an OnHint event handler.

Note: Call the CancelDispatch method from an OnHint event handler to prevent the application from forwarding the event to any other application events objects.

Tip: Component writers can respond to the automatically-fired THintAction action in a component's ExecuteAction method rather than relying on an OnHint event handler.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnHint event and the OnShowHint event and add the
following code to the handlers.  OnHint is triggered when
the mouse passes over the control. OnShowHint is triggered
when the mouse passes over the control and the application
is about to display the hint window for a Help Hint
*/
void __fastcall TAppEventForm::ApplicationEventsHint(TObject *Sender)
{
    lbOther->Items->Add("Event OnHint");
}

void __fastcall TAppEventForm::ApplicationEventsShowHint(AnsiString &HintStr,
      bool &CanShow, THintInfo &HintInfo)
{
    lbOther->Items->Add("Event OnShowHint");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnHint event and the OnShowHint event and add the
following code to the handlers.  OnHint is triggered when
the mouse passes over the control. OnShowHint is triggered
when the mouse passes over the control and the application
is about to display the hint window for a Help Hint
}
procedure TMainForm.ApplicationEventsHint(Sender: TObject);
begin
  lbOther.Items.Add('Event OnHint');
end;

procedure TMainForm.ApplicationEventsShowHint(var HintStr: String;
  var CanShow: Boolean; var HintInfo: THintInfo);
begin
  lbOther.Items.Add('Event OnShowHint');
end;
```

See Also

THintAction (see page 20707)

GetLongHint (see page 6252)

GetShortHint (see page 6253)

CancelDispatch (see page 1351)

Hint (see page 13712)

ActivateHint (see page 13668)

9.1.1.2.8 TApplicationEvents.OnIdle Event

Occurs when an application becomes idle.

Pascal

```
property OnIdle: TIdleEvent;
```

C++

```
__property TIdleEvent OnIdle;
```

Description

Write an OnIdle event handler to perform special processing when an application is idle. An application is idle when it is not processing code. For example, an application is idle when it is waiting for input from the user.

OnIdle is called only once, as the application transitions into an idle state. It is not called continuously unless Done is set to false. Applications that set Done to false consume an inordinate amount of CPU time, which affects overall system performance.

Note: Call the CancelDispatch method from an OnIdle event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnIdle event and add the following code to the
handler.  The OnIdle event handler is used to perform
special processing when an application is idle. An
application is idle when it is not processing code.
*/
void __fastcall TAppEventForm::ApplicationEventsIdle(TObject *Sender,
      bool &Done)
{
    lbIdle->Items->Add("Event OnIdle");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnIdle event and add the following code to the
handler.  The OnIdle event handler is used to perform
special processing when an application is idle. An
application is idle when it is not processing code.
}
procedure TMainForm.ApplicationEventsIdle(Sender: TObject;
  var Done: Boolean);
begin
  lbIdle.Items.Add('Event OnIdle');
end;
```

See Also

OnActionUpdate (see page 1354)

CancelDispatch (see page 1351)

9.1.1.2.9 TApplicationEvents.OnMessage Event

Occurs when the application receives a Windows message.

Pascal

```
property OnMessage: TMessageEvent;
```

C++

```
__property TMessageEvent OnMessage;
```

Description

Use OnMessage to trap any or all Windows messages posted to all windows in the application. OnMessage occurs when an application receives a Windows message. OnMessage only receives messages that are posted to the message queue, not those sent directly with the Windows API SendMessage function.

An OnMessage event handler allows an application to respond to messages other than those declared in the events for TApplication. If the application doesn't have a specific handler for an incoming message, the message is dispatched to the window for which it was intended, and Windows handles the message.

Warning: Caution:Thousands of messages per second flow though this event. Be careful when coding the handler, because it can affect the performance of the entire application.

Tip: Call the CancelDispatch method from an OnMessage event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnMessage event and add the following code to the
handler.  OnMessage is triggered when the application
receives a Windows message.
*/
void __fastcall TAppEventForm::ApplicationEventsMessage(tagMSG &Msg,
      bool &Handled)
{
    lbOnMessage->Items->Add("X:=" + IntToStr(Msg.pt.x) + "|Y:=" + IntToStr(Msg.pt.y));
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnMessage event and add the following code to the
handler.  OnMessage is triggered when the application
receives a Windows message.
}
procedure TMainForm.ApplicationEventsMessage(var Msg: tagMSG;
  var Handled: Boolean);
begin
  lbOnMessage.Items.Add('X:=' + IntToStr(Msg.pt.x) + '|Y:=' + IntToStr(Msg.pt.y));
end;
```

See Also

HandleMessage (see page 13674)

CancelDispatch (see page 1351)

ProcessMessages (see page 13691)

9.1.1.2.10 TApplicationEvents.OnMinimize Event

Occurs when an application is minimized.

Pascal

```
property OnMinimize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMinimize;
```

Description

Write an OnMinimize event handler to perform special processing when the application is minimized. The application is minimized, either because the user minimizes the main window, or because of a call to the Minimize method. The application's Icon property determines the icon that represents the minimized application.

Note: Call the CancelDispatch method from an OnMinimize event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnMinimize event and add the following code to the
handler.  OnMinimize is triggered when the form is
minimized.
*/
void __fastcall TAppEventForm::ApplicationEventsMinimize(TObject *Sender)
{
  lbOther->Items->Add("Event OnMinimize");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnMinimize event and add the following code to the
handler.  OnMinimize is triggered when the form is
minimized.
}
procedure TMainForm.ApplicationEventsMinimize(Sender: TObject);
begin
  lbOther.Items.Add('Event OnMinimize');
end;
```

See Also

Icon (see page 13717)

Minimize (see page 13688)

CancelDispatch (see page 1351)

OnRestore (see page 1364)

9.1.1.2.11 TApplicationEvents.OnModalBegin Event

Pascal

```
property OnModalBegin: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnModalBegin;
```

Description

This is OnModalBegin, a member of class TApplicationEvents.

9.1.1.2.12 TApplicationEvents.OnModalEnd Event

Pascal

```
property OnModalEnd: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnModalEnd;
```

Description

This is OnModalEnd, a member of class TApplicationEvents.

9.1.1.2.13 TApplicationEvents.OnRestore Event

Occurs when the previously minimized application is restored to its normal size.

Pascal

```
property OnRestore: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnRestore;
```

Description

Write an OnRestore event handler to perform special processing when the application is restored from the minimized state in which it appears as an icon. An application is restored either because the user restores the application, or because the application calls the Restore method.

Note: Don't confuse restoring an application with restoring a form or window to its original size. To minimize, maximize, and restore a window or form, change the value of the WindowState property.

Note: Call the CancelDispatch method from an OnRestore event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnRestore event and add the following code to the
handler.  OnRestore is triggered when a minimized
application is restored.
*/
```

```
void __fastcall TAppEventForm::ApplicationEventsRestore(TObject *Sender)
{

  lbOther->Items->Add("Event OnRestore");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnRestore event and add the following code to the
handler.  OnRestore is triggered when a minimized
application is restored.
}
procedure TMainForm.ApplicationEventsRestore(Sender: TObject);
begin
  lbOther.Items.Add('Event OnRestore');
end;
```

See Also

CancelDispatch (see page 1351)

Minimize (see page 13688)

Restore (see page 13694)

9.1.1.2.14 TApplicationEvents.OnSettingChange Event

Occurs when Windows notifies the application that a system-wide setting has changed.

Pascal

```
property OnSettingChange: TSettingChangeEvent;
```

C++

```
__property TSettingChangeEvent OnSettingChange;
```

Description

Use OnSettingChange to respond when Windows informs the application that a system-wide setting or policy has changed.

C++ Examples:

```
/*
This example requires that a memo and a TApplicationEvent be
added to the form.  Change the Windows Locale by using the
Regional and Language Options in the Windows Control Panel.
*/
void __fastcall TForm1::ApplicationEvents1SettingChange(TObject *Sender, int Flag, const
AnsiString Section, int &Result)
{
  Memo1->Lines->BeginUpdate();
  try
  {
//    char buffer[64];
    AnsiString buffer;
    buffer.sprintf("Section = %s", Section.c_str());
    Memo1->Lines->Add(buffer);
    buffer.sprintf("Flag = %8x", Flag);
    Memo1->Lines->Add(buffer);
    if (strcmp(Section.c_str(), "intl") == 0)
    {
      buffer.sprintf("DefaultLCID = %8x", SysLocale.DefaultLCID);
      Memo1->Lines->Add(buffer);
      buffer.sprintf("PriLangID = %8x", SysLocale.PriLangID);
```

```
        Memo1->Lines->Add(buffer);
        buffer.sprintf("SubLangID = %8x", SysLocale.SubLangID);
        Memo1->Lines->Add(buffer);
        buffer.sprintf("FarEast = %s", BoolToStr(SysLocale.FarEast, true));
        Memo1->Lines->Add(buffer);
        buffer.sprintf("MiddleEast = %s", BoolToStr(SysLocale.MiddleEast, true));
        Memo1->Lines->Add(buffer);
        Memo1->Lines->Add("");
      }
    }
    __finally
    {
      Memo1->Lines->EndUpdate();
    }
  }
```

Delphi Examples:

```
{
This example requires that a memo and a TApplicationEvent be
added to the form.  Change the Windows Locale by using the
Regional and Language Options in the Windows Control Panel.
}
procedure TForm1.ApplicationEvents1SettingChange(Sender: TObject; Flag: Integer; const
Section: string; var Result: Integer);
begin
  Memo1.Lines.BeginUpdate;
  try
    Memo1.Lines.Add(Format('Section  = %s', [Section]));
    Memo1.Lines.Add(Format('Flags  = %.8x', [Flag]));
    if AnsiSameStr(Section, 'intl') then
    with SysLocale do
    begin
      Memo1.Lines.Add(Format('DefaultLCID  = %.8x', [DefaultLCID]));
      Memo1.Lines.Add(Format('PriLangID  = %.8x', [PriLangID]));
      Memo1.Lines.Add(Format('SubLangID  = %.8x', [SubLangID]));
      Memo1.Lines.Add(Format('FarEast  = %s', [BoolToStr(FarEast, True)]));
      Memo1.Lines.Add(Format('MiddleEast  = %s', [BoolToStr(MiddleEast, True)]));
    end;
    Memo1.Lines.Add('');
  finally
    Memo1.Lines.EndUpdate;
  end;
//  Result:= 0;
end;
```

See Also

OnSettingChange (see page 13741)

variables

9.1.1.2.15 TApplicationEvents.OnShortCut Event

Occurs when the user presses a key (before the OnKeyDown event).

Pascal

```
property OnShortCut: TShortCutEvent;
```

C++

```
__property TShortCutEvent OnShortCut;
```

Description

Use OnShortCut to dispatch shortcut keystrokes before the form or its controls handle them. When the user presses a key, the

application can dispatch it as a shortcut key instead of allowing the standard keystroke processing (OnKeyDown, OnKeyPress, and OnKeyUp). Built-in shortcut processing is provided for menu shortcuts and actions associated with the form. OnShortCut lets the application implement additional shortcuts.

If the OnShortCut event handler responds to the keystroke, set its Handled parameter to true. This prevents the keystroke from being passed on to menus or actions associated with the application. It also prevents the standard keystroke processing in the same way that a menu or action shortcut does.

Note: Call the CancelDispatch method from an OnShortCut event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnShortCut event and add the following code to the
handler.  Trigger this event by pressing the CTRL key or
any function key.
*/
void __fastcall TAppEventForm::ApplicationEventsShortCut(TWMKey &Msg,
      bool &Handled)
{
  lbOther->Items->Add("Event OnShortCut");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnShortCut event and add the following code to the
handler.  Trigger this event by pressing the CTRL key or
any function key.
}
procedure TMainForm.ApplicationEventsShortCut(var Msg: TWMKey;
  var Handled: Boolean);
begin
  lbOther.Items.Add('Event OnShortCut');
end;
```

See Also

OnShortCut (see page 13977)

OnKeyDown (see page 6225)

CancelDispatch (see page 1351)

9.1.1.2.16 TApplicationEvents.OnShowHint Event

Occurs when the application is about to display the hint window for a Help Hint.

Pascal

```
property OnShowHint: TShowHintEvent;
```

C++

```
__property TShowHintEvent OnShowHint;
```

Description

Write an OnShowHint event handler to change the appearance and behavior of Help Hints.

The HintStr parameter sets the text of the Help Hint. To obtain the text of a hint for a particular control, call the GetLongHint or GetShortHint function, assigning the result to HintStr. To change the text, change the value of this string.

Use the CanShow parameter to permit or prevent the Help Hint from displaying. If CanShow is true, the Help Hint displays. If it is false, the Help Hint does not appear.

Note: Call the CancelDispatch method from an OnShowHint event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnHint event and the OnShowHint event and add the
following code to the handlers.  OnHint is triggered when
the mouse passes over the control. OnShowHint is triggered
when the mouse passes over the control and the application
is about to display the hint window for a Help Hint
*/
void __fastcall TAppEventForm::ApplicationEventsHint(TObject *Sender)
{
    lbOther->Items->Add("Event OnHint");
}

void __fastcall TAppEventForm::ApplicationEventsShowHint(AnsiString &HintStr,
      bool &CanShow, THintInfo &HintInfo)
{
    lbOther->Items->Add("Event OnShowHint");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnHint event and the OnShowHint event and add the
following code to the handlers.  OnHint is triggered when
the mouse passes over the control. OnShowHint is triggered
when the mouse passes over the control and the application
is about to display the hint window for a Help Hint
}
procedure TMainForm.ApplicationEventsHint(Sender: TObject);
begin
  lbOther.Items.Add('Event OnHint');
end;

procedure TMainForm.ApplicationEventsShowHint(var HintStr: String;
  var CanShow: Boolean; var HintInfo: THintInfo);
begin
  lbOther.Items.Add('Event OnShowHint');
end;
```

See Also

GetLongHint (see page 6252)

GetShortHint (see page 6253)

HintHidePause (see page 13715)

OnHint (see page 1359)

CancelDispatch (see page 1351)

9.1.2 TCustomApplicationEvents Class

TCustomApplicationEvents intercepts application-level events.

Class Hierarchy

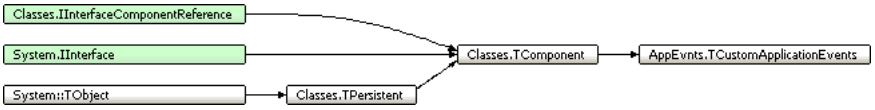

Pascal

```
TCustomApplicationEvents = class(TComponent);
```

C++

```
class TCustomApplicationEvents : public TComponent;
```

File

AppEvnts

Description

Use TCustomApplicationEvents as a base class for objects that intercept the events of the global Application object. Each event of the TApplicationEvents (see page 1326) object is the same as the event with the same name on the Application object.

Do not create instances of TCustomApplicationEvents. Instead, use a descendant of TCustomApplicationEvents such a TApplicationEvents (see page 1326).

See Also

TApplication (see page 13660)

Application

TApplicationEvents (see page 1326)

9.1.2.1 TCustomApplicationEvents Members

The following tables list the members exposed by TCustomApplicationEvents.

Protected Events

	Name	Description
	OnActionExecute (see page 1352)	Occurs when an action's Execute method is called and its action list has not already handled it.
	OnActionUpdate (see page 1354)	Occurs when an action's Update method is called and its action list has not already handled it.
	OnActivate (see page 1355)	Occurs when an application becomes active.
	OnDeactivate (see page 1356)	Occurs when an application becomes inactive.
	OnException (see page 1357)	Occurs when an unhandled exception occurs in the application.
	OnHelp (see page 1358)	Occurs when the application receives a request for help.
	OnHint (see page 1359)	Occurs when the mouse pointer moves over a control or menu item that can display a Help Hint.
	OnIdle (see page 1361)	Occurs when an application becomes idle.
	OnMessage (see page 1362)	Occurs when the application receives a Windows message.
	OnMinimize (see page 1363)	Occurs when an application is minimized.

	OnModalBegin (see page 1364)	This is OnModalBegin, a member of class TCustomApplicationEvents.
	OnModalEnd (see page 1364)	This is OnModalEnd, a member of class TCustomApplicationEvents.
	OnRestore (see page 1364)	Occurs when the previously minimized application is restored to its normal size.
	OnSettingChange (see page 1365)	Occurs when Windows notifies the application that a system-wide setting has changed.
	OnShortCut (see page 1366)	Occurs when the user presses a key (before the OnKeyDown event).
	OnShowHint (see page 1367)	Occurs when the application is about to display the hint window for a Help Hint.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.

	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Activate (see page 1351)	Ensures that this is the first component to receive any application events.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CancelDispatch (see page 1351)	Prevents other TCustomApplicationEvents (see page 1347) objects from receiving the current event.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1350)	Creates a new TCustomApplicationEvents (see page 1347) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

9.1.2.2 TCustomApplicationEvents.Create Constructor

Creates a new TCustomApplicationEvents (see page 1347) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomApplicationEvents(TComponent * AOwner);
```

Description

Call Create to instantiate TCustomApplicationEvents (see page 1347) at runtime. Components added to forms at design time are created automatically.

Create adds the component to the internal list of components that respond to the application's events. Once the component is created, it responds to any of the application events unless another TCustomApplicationEvents (see page 1347) component prevents it by calling CancelDispatch (see page 1351).

See Also

CancelDispatch (see page 1351)

9.1.2.3 TCustomApplicationEvents Methods

The methods of the TCustomApplicationEvents class are listed here.

Public Methods

	Name	Description
	Activate (see page 1351)	Ensures that this is the first component to receive any application events.
	CancelDispatch (see page 1351)	Prevents other TCustomApplicationEvents (see page 1347) objects from receiving the current event.

9.1.2.3.1 TCustomApplicationEvents.Activate Method

Ensures that this is the first component to receive any application events.

Pascal

```
procedure Activate;
```

C++

```
__fastcall Activate();
```

Description

Call Activate to move the TCustomApplicationEvents (see page 1347) descendant to the front of the internal list of components that respond to the application's events. There is no need to call Activate unless the project includes more than one TCustomApplicationEvents (see page 1347) descendant.

Typically, each form that contains a TCustomApplicationEvents (see page 1347) descendant calls that component's Activate method from its OnActivate (see page 1355) event handler. This has the effect of switching the first event handler to receive an event every time the user changes the active form.

Activate does not prevent other TCustomApplicationEvents (see page 1347) descendants from receiving events, it merely places them in the queue behind the active TCustomApplicationEvents (see page 1347) descendant. To prevent other components from receiving the events, use the CancelDispatch (see page 1351) method.

Warning: Do not call Activate from an event handler that is responding to an application-level event. Such a call will cause an exception to be raised.

See Also

OnActivate (see page 13943)

CancelDispatch (see page 1351)

9.1.2.3.2 TCustomApplicationEvents.CancelDispatch Method

Prevents other TCustomApplicationEvents (see page 1347) objects from receiving the current event.

Pascal

```
procedure CancelDispatch;
```

C++

```
__fastcall CancelDispatch();
```

Description

Call CancelDispatch from one of the TCustomApplicationEvents (see page 1347) descendant's event handlers when it handles an event. CancelDispatch prevents other TCustomApplicationEvents (see page 1347) descendant's from receiving

the event after the current event handler exits.

CancelDispatch has no effect unless it is called from one of the component's event handlers.

9.1.2.4 TCustomApplicationEvents Events

The events of the TCustomApplicationEvents class are listed here.

Protected Events

	Name	Description
	OnActionExecute (see page 1352)	Occurs when an action's Execute method is called and its action list has not already handled it.
	OnActionUpdate (see page 1354)	Occurs when an action's Update method is called and its action list has not already handled it.
	OnActivate (see page 1355)	Occurs when an application becomes active.
	OnDeactivate (see page 1356)	Occurs when an application becomes inactive.
	OnException (see page 1357)	Occurs when an unhandled exception occurs in the application.
	OnHelp (see page 1358)	Occurs when the application receives a request for help.
	OnHint (see page 1359)	Occurs when the mouse pointer moves over a control or menu item that can display a Help Hint.
	OnIdle (see page 1361)	Occurs when an application becomes idle.
	OnMessage (see page 1362)	Occurs when the application receives a Windows message.
	OnMinimize (see page 1363)	Occurs when an application is minimized.
	OnModalBegin (see page 1364)	This is OnModalBegin, a member of class TCustomApplicationEvents.
	OnModalEnd (see page 1364)	This is OnModalEnd, a member of class TCustomApplicationEvents.
	OnRestore (see page 1364)	Occurs when the previously minimized application is restored to its normal size.
	OnSettingChange (see page 1365)	Occurs when Windows notifies the application that a system-wide setting has changed.
	OnShortCut (see page 1366)	Occurs when the user presses a key (before the OnKeyDown event).
	OnShowHint (see page 1367)	Occurs when the application is about to display the hint window for a Help Hint.

9.1.2.4.1 TCustomApplicationEvents.OnActionExecute Event

Occurs when an action's Execute method is called and its action list has not already handled it.

Pascal

```
property OnActionExecute: TActionEvent;
```

C++

```
__property TActionEvent OnActionExecute;
```

Description

Use the OnExecute event handler to respond when a user invokes actions whose action lists do not have OnExecute event handlers.

If the action list that contains the action does not handle it in an OnExecute event handler, then the action is routed to the Application object's OnActionExecute event. The application events object hooks this event and responds with its own OnActionExecute event handler.

The Handled parameter of the event handler returns false by default. If the handler handles the event, it should change Handled to true, thereby preventing further attempts to handle the action. When the event handler exits with Handled set to false, the event first goes to any other application events objects, and then the action's OnExecute event occurs. If the action remains unhandled after that, the active control's ExecuteAction method is called to allow the action to execute with an identified target. Finally, the active form's ExecuteAction method is called if all other handlers do not handle the action.

Note: Call the CancelDispatch (see page 1351) method from an OnActionExecute event handler to prevent the application

from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents, a TActionList
and a TListBox in the form.  Select the TApplicationEvents,
double click on the OnActionExecute event and add the
following code to the handler.  Create an TAction named
MyAction in a TActionList.  The TActionList must have no
OnExecute event handler, and the TAction has an OnExecute
event handler called MyActionExecute.  Then set an object's
(such as a button) Action to MyAction.  The Handled
parameter of the event handler returns false by default. If
the handler handles the event, it should change Handled to
true, thereby preventing further attempts to handle the
action.
*/
void __fastcall TAppEventForm::ApplicationEventsActionExecute(
      TBasicAction *Action, bool &Handled)
{
    lbOther->Items->Add("Event OnActionExecute");
}

void __fastcall TAppEventForm::ActionExecute(TObject *Sender)
{
    ShowMessage("Action executed");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents, a TActionList
and a TListBox in the form.  Select the TApplicationEvents,
double click on the OnActionExecute event and add the
following code to the handler.  Create an TAction named
MyAction in a TActionList.  The TActionList must have no
OnExecute event handler, and the TAction has an OnExecute
event handler called MyActionExecute.  Then set an object's
(such as a button) Action to MyAction.  The Handled
parameter of the event handler returns false by default. If
the handler handles the event, it should change Handled to
true, thereby preventing further attempts to handle the
action.
}
procedure TMainForm.ApplicationEventsActionExecute(Action: TBasicAction;
  var Handled: Boolean);
begin
  lbOther.Items.Add('Event OnActionExecute');
end;

procedure TMainForm.MyActionExecute(Sender: TObject);
begin
  ShowMessage('Action executed');
end;
```

See Also

OnActionUpdate (see page 1354)

ExecuteAction (see page 13673)

OnExecute (see page 2716)

OnExecute (see page 327)

ExecuteAction

CancelDispatch (see page 1351)

9.1.2.4.2 TCustomApplicationEvents.OnActionUpdate Event

Occurs when an action's Update method is called and its action list has not already handled it.

Pascal

```
property OnActionUpdate: TActionEvent;
```

C++

```
__property TActionEvent OnActionUpdate;
```

Description

Use the OnActionUpdate event handler to update the properties of an action when the application is idle if the action's action list does not handle it in an OnUpdate event handler. OnActionUpdate occurs after the OnIdle (see page 1361) event.

If the action list that contains the action does not update it in an OnUpdate event handler, the action is routed to the Application object's OnActionUpdate event. The application events object hooks this event and responds with its own OnActionUpdate event handler.

The Handled parameter of the event handler returns false by default. If the handler updates the event, it should change Handled to true, thereby ending the processing of the action. When the event handler exits with Handled set to false, the event first goes to any other application events objects, and then the action's OnUpdate event occurs. If the action is not updated after that, the active control's UpdateAction method is called to allow the target to update the action. Finally, the active form's UpdateAction method is called if all other handlers do not handle the action.

Note: Call the CancelDispatch (see page 1351) method from an OnActionUpdate event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnActionUpdate event and add the following code to
the handler.  Create other TApplicationEvents the same way.
When these events update this event is triggered.  The
Handled parameter of the event handler returns false by
default. If the handler handles the event, it should change
Handled to true, thereby preventing further attempts to
*/
void __fastcall TAppEventForm::ApplicationEventsActionUpdate(
      TBasicAction *Action, bool &Handled)
{
     lbActionUpdate->Items->Add("Event OnActionUpdate");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnActionUpdate event and add the following code to
the handler.  Create other TApplicationEvents the same way.
When these events update this event is triggered.  The
Handled parameter of the event handler returns false by
default. If the handler handles the event, it should change
Handled to true, thereby preventing further attempts to
```

```
  handle the action.
  }
  procedure TMainForm.ApplicationEventsActionUpdate(Action: TBasicAction;
    var Handled: Boolean);
  begin
    lbActionUpdate.Items.Add('Event OnActionUpdate');
  end;
```

See Also

OnActionExecute (see page 1352)

UpdateAction (see page 13701)

OnUpdate (see page 2718)

OnUpdate (see page 327)

UpdateAction

OnIdle (see page 1361)

CancelDispatch (see page 1351)

9.1.2.4.3 TCustomApplicationEvents.OnActivate Event

Occurs when an application becomes active.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to perform special processing when the application becomes active.

An application becomes active when it is initially run or when focus moves from another Windows application back to any window of the application.

Note: Call the CancelDispatch (see page 1351) method from an OnActivate event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnActivate event and add the following code
to the handler.  These events occur when the form gains
or loses focus.
*/
void __fastcall TAppEventForm::ApplicationEventsActivate(TObject *Sender)
{
     lbOther->Items->Add("Event OnActivate");
}

void __fastcall TAppEventForm::ApplicationEventsDeactivate(TObject *Sender)
{
    lbOther->Items->Add("Event OnDeactivate");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnActivate event and add the following code
to the handler.  These events occur when the form gains
or loses focus.
}
procedure TMainForm.ApplicationEventsActivate(Sender: TObject);
begin
  lbOther.Items.Add('Event OnActivate');
end;

procedure TMainForm.ApplicationEventsDeactivate(Sender: TObject);
begin
  lbOther.Items.Add('Event OnDeactivate');
end;
```

See Also

Active (see page 13704)

OnDeactivate (see page 1356)

CancelDispatch (see page 1351)

9.1.2.4.4 TCustomApplicationEvents.OnDeactivate Event

Occurs when an application becomes inactive.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to perform any special processing that should occur immediately before the application is deactivated. The OnDeactivate event occurs when the user switches from the application to another Windows application.

Note: Call the CancelDispatch (see page 1351) method from an OnDeactivate event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnActivate event and add the following code
to the handler.  These events occur when the form gains
or loses focus.
*/
void __fastcall TAppEventForm::ApplicationEventsActivate(TObject *Sender)
{
     lbOther->Items->Add("Event OnActivate");
}

void __fastcall TAppEventForm::ApplicationEventsDeactivate(TObject *Sender)
{
    lbOther->Items->Add("Event OnDeactivate");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnActivate event and add the following code
to the handler.  These events occur when the form gains
or loses focus.
}
procedure TMainForm.ApplicationEventsActivate(Sender: TObject);
begin
  lbOther.Items.Add('Event OnActivate');
end;

procedure TMainForm.ApplicationEventsDeactivate(Sender: TObject);
begin
  lbOther.Items.Add('Event OnDeactivate');
end;
```

See Also

Active (see page 13704)

OnActivate (see page 1355)

CancelDispatch (see page 1351)

9.1.2.4.5 TCustomApplicationEvents.OnException Event

Occurs when an unhandled exception occurs in the application.

Pascal

```
property OnException: TExceptionEvent;
```

C++

```
__property TExceptionEvent OnException;
```

Description

Use OnException to change the default behavior that occurs when an exception is not handled by application code. The OnException event handler is called automatically in the application's HandleException method.

OnException only handles exceptions that occur during message processing. Exceptions that occur before or after the execution of the application's Run method do not generate OnException events.

If an exception passes through the try blocks in the application code, the application automatically calls the HandleException method. Unless the exception object is EAbort, HandleException calls the OnException handler, if one exists. Otherwise, it calls ShowException to display a message box indicating an error occurred.

Note: Call the CancelDispatch (see page 1351) method from an OnException event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnException event and add the following code
to the handler.  Trigger this event by calling the
Exception Create constructor.
*/
void __fastcall TAppEventForm::ApplicationEventsException(TObject *Sender,
```

```
      Exception *E)
{
    lbOther->Items->Add("Event OnException: " + E->Message);
}

void __fastcall TAppEventForm::MenuExceptionClick(TObject *Sender)
{
    char *sExceptionRaised = "This is an exception";
    throw Exception(sExceptionRaised);
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnException event and add the following code
to the handler.  Trigger this event by calling the
Exception Create constructor.
}
procedure TMainForm.ApplicationEventsException(Sender: TObject;
  E: Exception);
begin
  lbOther.Items.Add('Event OnException: ' + E.Message);
end;

procedure TMainForm.MenuExceptionClick(Sender: TObject);
resourcestring
  sExceptionRaised = 'This is an exception';

begin
  raise Exception.Create(sExceptionRaised);
end;
```

See Also

HandleException (see page 13674)

ShowException (see page 13698)

CancelDispatch (see page 1351)

9.1.2.4.6 TCustomApplicationEvents.OnHelp Event

Occurs when the application receives a request for help.

Pascal

```
property OnHelp: THelpEvent;
```

C++

```
__property THelpEvent OnHelp;
```

Description

Write an OnHelp event handler to perform special processing when the user requests help. The HelpContext and the HelpJump methods automatically trigger the OnHelp event.

Set CallHelp to true to have the application call WinHelp after the event. Set CallHelp to false to prevent the application from calling WinHelp. All application help methods go through OnHelp. The application calls WinHelp only if OnHelp's CallHelp parameter returns True, true or if no OnHelp event handler is assigned.

To find the possible values of the Command and Data parameters, search for WinHelp in the Win32 Developer's Reference Help (Win32.HLP) file, which explains the WinHelp API (application programming interface). The possible values for the Data parameter depend upon the value of the Command parameter.

The return value is true if the event handler succeeds, false if it fails.

Note: Call the CancelDispatch (see page 1351) method from an OnHelp event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnHelp event and add the following code to the
handler.  Trigger this event by pressing the F1 key or
call the HelpContext method.  Set CallHelp to true if the
application should still invoke the help system after the
event.
*/
bool __fastcall TAppEventForm::ApplicationEventsHelp(WORD Command, int Data,
      bool &CallHelp)
{
    lbOther->Items->Add("Event OnHelp");
    return false;
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnHelp event and add the following code to the
handler.  Trigger this event by pressing the F1 key or
call the HelpContext method.  Set CallHelp to true if the
application should still invoke the help system after the
event.
}
function TMainForm.ApplicationEventsHelp(Command: Word; Data: Integer;
  var CallHelp: Boolean): Boolean;
begin
  lbOther.Items.Add('Event OnHelp');
  CallHelp := False;
end;
```

See Also

HelpCommand (see page 13678)

HelpContext (see page 13679)

HelpFile (see page 13711)

CancelDispatch (see page 1351)

HelpJump (see page 13679)

9.1.2.4.7 TCustomApplicationEvents.OnHint Event

Occurs when the mouse pointer moves over a control or menu item that can display a Help Hint.

Pascal

```
property OnHint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnHint;
```

Description

Write an OnHint event handler to perform special processing when the mouse pauses over a control or menu item whose Hint property is not an empty string (' "" ').

A common use of the OnHint event is to display the value of a control or menu item's Hint property as the text of a status bar (TStatusBar). The Hint property of a control can specify both a short Help Hint and a (usually) longer hint that appears elsewhere because of code in an OnHint event handler.

Note: Call the CancelDispatch (see page 1351) method from an OnHint event handler to prevent the application from forwarding the event to any other application events objects.

Tip: Component writers can respond to the automatically-fired THintAction action in a component's ExecuteAction method rather than relying on an OnHint event handler.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnHint event and the OnShowHint event and add the
following code to the handlers.  OnHint is triggered when
the mouse passes over the control. OnShowHint is triggered
when the mouse passes over the control and the application
is about to display the hint window for a Help Hint
*/
void __fastcall TAppEventForm::ApplicationEventsHint(TObject *Sender)
{
    lbOther->Items->Add("Event OnHint");
}

void __fastcall TAppEventForm::ApplicationEventsShowHint(AnsiString &HintStr,
      bool &CanShow, THintInfo &HintInfo)
{
    lbOther->Items->Add("Event OnShowHint");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnHint event and the OnShowHint event and add the
following code to the handlers.  OnHint is triggered when
the mouse passes over the control. OnShowHint is triggered
when the mouse passes over the control and the application
is about to display the hint window for a Help Hint
}
procedure TMainForm.ApplicationEventsHint(Sender: TObject);
begin
  lbOther.Items.Add('Event OnHint');
end;

procedure TMainForm.ApplicationEventsShowHint(var HintStr: String;
  var CanShow: Boolean; var HintInfo: THintInfo);
begin
  lbOther.Items.Add('Event OnShowHint');
end;
```

See Also

THintAction (see page 20707)

GetLongHint (see page 6252)

GetShortHint (see page 6253)

CancelDispatch (see page 1351)

Hint (see page 13712)

ActivateHint (see page 13668)

9.1.2.4.8 TCustomApplicationEvents.OnIdle Event

Occurs when an application becomes idle.

Pascal

```
property OnIdle: TIdleEvent;
```

C++

```
__property TIdleEvent OnIdle;
```

Description

Write an OnIdle event handler to perform special processing when an application is idle. An application is idle when it is not processing code. For example, an application is idle when it is waiting for input from the user.

OnIdle is called only once, as the application transitions into an idle state. It is not called continuously unless Done is set to false. Applications that set Done to false consume an inordinate amount of CPU time, which affects overall system performance.

Note: Call the CancelDispatch (see page 1351) method from an OnIdle event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnIdle event and add the following code to the
handler.  The OnIdle event handler is used to perform
special processing when an application is idle. An
application is idle when it is not processing code.
*/
void __fastcall TAppEventForm::ApplicationEventsIdle(TObject *Sender,
      bool &Done)
{
    lbIdle->Items->Add("Event OnIdle");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnIdle event and add the following code to the
handler.  The OnIdle event handler is used to perform
special processing when an application is idle. An
application is idle when it is not processing code.
}
procedure TMainForm.ApplicationEventsIdle(Sender: TObject;
  var Done: Boolean);
begin
  lbIdle.Items.Add('Event OnIdle');
end;
```

See Also

OnActionUpdate (see page 1354)

CancelDispatch (see page 1351)

9.1.2.4.9 TCustomApplicationEvents.OnMessage Event

Occurs when the application receives a Windows message.

Pascal

```
property OnMessage: TMessageEvent;
```

C++

```
__property TMessageEvent OnMessage;
```

Description

Use OnMessage to trap any or all Windows messages posted to all windows in the application. OnMessage occurs when an application receives a Windows message. OnMessage only receives messages that are posted to the message queue, not those sent directly with the Windows API SendMessage function.

An OnMessage event handler allows an application to respond to messages other than those declared in the events for TApplication. If the application doesn't have a specific handler for an incoming message, the message is dispatched to the window for which it was intended, and Windows handles the message.

Warning: Caution:Thousands of messages per second flow though this event. Be careful when coding the handler, because it can affect the performance of the entire application.

Tip: Call the CancelDispatch (see page 1351) method from an OnMessage event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnMessage event and add the following code to the
handler.  OnMessage is triggered when the application
receives a Windows message.
*/
void __fastcall TAppEventForm::ApplicationEventsMessage(tagMSG &Msg,
      bool &Handled)
{
    lbOnMessage->Items->Add("X:=" + IntToStr(Msg.pt.x) + "|Y:=" + IntToStr(Msg.pt.y));
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnMessage event and add the following code to the
handler.  OnMessage is triggered when the application
receives a Windows message.
}
procedure TMainForm.ApplicationEventsMessage(var Msg: tagMSG;
  var Handled: Boolean);
begin
  lbOnMessage.Items.Add('X:=' + IntToStr(Msg.pt.x) + '|Y:=' + IntToStr(Msg.pt.y));
end;
```

See Also

HandleMessage (see page 13674)

CancelDispatch (see page 1351)

ProcessMessages (see page 13691)

9.1.2.4.10 TCustomApplicationEvents.OnMinimize Event

Occurs when an application is minimized.

Pascal

```
property OnMinimize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMinimize;
```

Description

Write an OnMinimize event handler to perform special processing when the application is minimized. The application is minimized, either because the user minimizes the main window, or because of a call to the Minimize method. The application's Icon property determines the icon that represents the minimized application.

Note: Call the CancelDispatch (see page 1351) method from an OnMinimize event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnMinimize event and add the following code to the
handler.  OnMinimize is triggered when the form is
minimized.
*/
void __fastcall TAppEventForm::ApplicationEventsMinimize(TObject *Sender)
{
  lbOther->Items->Add("Event OnMinimize");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnMinimize event and add the following code to the
handler.  OnMinimize is triggered when the form is
minimized.
}
procedure TMainForm.ApplicationEventsMinimize(Sender: TObject);
begin
  lbOther.Items.Add('Event OnMinimize');
end;
```

See Also

Icon (see page 13717)

Minimize (see page 13688)

CancelDispatch (see page 1351)

OnRestore (see page 1364)

9.1.2.4.11 TCustomApplicationEvents.OnModalBegin Event

Pascal

```
property OnModalBegin: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnModalBegin;
```

Description

This is OnModalBegin, a member of class TCustomApplicationEvents.

9.1.2.4.12 TCustomApplicationEvents.OnModalEnd Event

Pascal

```
property OnModalEnd: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnModalEnd;
```

Description

This is OnModalEnd, a member of class TCustomApplicationEvents.

9.1.2.4.13 TCustomApplicationEvents.OnRestore Event

Occurs when the previously minimized application is restored to its normal size.

Pascal

```
property OnRestore: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnRestore;
```

Description

Write an OnRestore event handler to perform special processing when the application is restored from the minimized state in which it appears as an icon. An application is restored either because the user restores the application, or because the application calls the Restore method.

Note: Don't confuse restoring an application with restoring a form or window to its original size. To minimize, maximize, and restore a window or form, change the value of the WindowState property.

Note: Call the CancelDispatch (see page 1351) method from an OnRestore event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnRestore event and add the following code to the
handler.  OnRestore is triggered when a minimized
```

```
application is restored.
*/
void __fastcall TAppEventForm::ApplicationEventsRestore(TObject *Sender)
{

  lbOther->Items->Add("Event OnRestore");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnRestore event and add the following code to the
handler.  OnRestore is triggered when a minimized
application is restored.
}
procedure TMainForm.ApplicationEventsRestore(Sender: TObject);
begin
  lbOther.Items.Add('Event OnRestore');
end;
```

See Also

CancelDispatch (see page 1351)

Minimize (see page 13688)

Restore (see page 13694)

9.1.2.4.14 TCustomApplicationEvents.OnSettingChange Event

Occurs when Windows notifies the application that a system-wide setting has changed.

Pascal

```
property OnSettingChange: TSettingChangeEvent;
```

C++

```
__property TSettingChangeEvent OnSettingChange;
```

Description

Use OnSettingChange to respond when Windows informs the application that a system-wide setting or policy has changed.

C++ Examples:

```
/*
This example requires that a memo and a TApplicationEvent be
added to the form.  Change the Windows Locale by using the
Regional and Language Options in the Windows Control Panel.
*/
void __fastcall TForm1::ApplicationEvents1SettingChange(TObject *Sender, int Flag, const
AnsiString Section, int &Result)
{
  Memo1->Lines->BeginUpdate();
  try
  {
//    char buffer[64];
    AnsiString buffer;
    buffer.sprintf("Section = %s", Section.c_str());
    Memo1->Lines->Add(buffer);
    buffer.sprintf("Flag = %8x", Flag);
    Memo1->Lines->Add(buffer);
    if (strcmp(Section.c_str(), "intl") == 0)
    {
      buffer.sprintf("DefaultLCID = %8x", SysLocale.DefaultLCID);
```

```
        Memo1->Lines->Add(buffer);
        buffer.sprintf("PriLangID = %8x", SysLocale.PriLangID);
        Memo1->Lines->Add(buffer);
        buffer.sprintf("SubLangID = %8x", SysLocale.SubLangID);
        Memo1->Lines->Add(buffer);
        buffer.sprintf("FarEast = %s", BoolToStr(SysLocale.FarEast, true));
        Memo1->Lines->Add(buffer);
        buffer.sprintf("MiddleEast = %s", BoolToStr(SysLocale.MiddleEast, true));
        Memo1->Lines->Add(buffer);
        Memo1->Lines->Add("");
      }
    }
    __finally
    {
      Memo1->Lines->EndUpdate();
    }
  }
```

Delphi Examples:

```
{
This example requires that a memo and a TApplicationEvent be
added to the form.  Change the Windows Locale by using the
Regional and Language Options in the Windows Control Panel.
}
procedure TForm1.ApplicationEvents1SettingChange(Sender: TObject; Flag: Integer; const
Section: string; var Result: Integer);
begin
  Memo1.Lines.BeginUpdate;
  try
    Memo1.Lines.Add(Format('Section  = %s', [Section]));
    Memo1.Lines.Add(Format('Flags  = %.8x', [Flag]));
    if AnsiSameStr(Section, 'intl') then
    with SysLocale do
    begin
      Memo1.Lines.Add(Format('DefaultLCID  = %.8x', [DefaultLCID]));
      Memo1.Lines.Add(Format('PriLangID  = %.8x', [PriLangID]));
      Memo1.Lines.Add(Format('SubLangID  = %.8x', [SubLangID]));
      Memo1.Lines.Add(Format('FarEast  = %s', [BoolToStr(FarEast, True)]));
      Memo1.Lines.Add(Format('MiddleEast  = %s', [BoolToStr(MiddleEast, True)]));
    end;
    Memo1.Lines.Add('');
  finally
    Memo1.Lines.EndUpdate;
  end;
//  Result:= 0;
end;
```

See Also

OnSettingChange (see page 13741)

variables

9.1.2.4.15 TCustomApplicationEvents.OnShortCut Event

Occurs when the user presses a key (before the OnKeyDown event).

Pascal

```
property OnShortCut: TShortCutEvent;
```

C++

```
__property TShortCutEvent OnShortCut;
```

Description

Use OnShortCut to dispatch shortcut keystrokes before the form or its controls handle them. When the user presses a key, the application can dispatch it as a shortcut key instead of allowing the standard keystroke processing (OnKeyDown, OnKeyPress, and OnKeyUp). Built-in shortcut processing is provided for menu shortcuts and actions associated with the form. OnShortCut lets the application implement additional shortcuts.

If the OnShortCut event handler responds to the keystroke, set its Handled parameter to true. This prevents the keystroke from being passed on to menus or actions associated with the application. It also prevents the standard keystroke processing in the same way that a menu or action shortcut does.

Note: Call the CancelDispatch (see page 1351) method from an OnShortCut event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnShortCut event and add the following code to the
handler.  Trigger this event by pressing the CTRL key or
any function key.
*/
void __fastcall TAppEventForm::ApplicationEventsShortCut(TWMKey &Msg,
      bool &Handled)
{
  lbOther->Items->Add("Event OnShortCut");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnShortCut event and add the following code to the
handler.  Trigger this event by pressing the CTRL key or
any function key.
}
procedure TMainForm.ApplicationEventsShortCut(var Msg: TWMKey;
  var Handled: Boolean);
begin
  lbOther.Items.Add('Event OnShortCut');
end;
```

See Also

OnShortCut (see page 13977)

OnKeyDown (see page 6225)

CancelDispatch (see page 1351)

9.1.2.4.16 TCustomApplicationEvents.OnShowHint Event

Occurs when the application is about to display the hint window for a Help Hint.

Pascal

```
property OnShowHint: TShowHintEvent;
```

C++

```
__property TShowHintEvent OnShowHint;
```

Description

Write an OnShowHint event handler to change the appearance and behavior of Help Hints.

The HintStr parameter sets the text of the Help Hint. To obtain the text of a hint for a particular control, call the GetLongHint or GetShortHint function, assigning the result to HintStr. To change the text, change the value of this string.

Use the CanShow parameter to permit or prevent the Help Hint from displaying. If CanShow is true, the Help Hint displays. If it is false, the Help Hint does not appear.

Note: Call the CancelDispatch (see page 1351) method from an OnShowHint event handler to prevent the application from forwarding the event to any other application events objects.

C++ Examples:

```
/*
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnHint event and the OnShowHint event and add the
following code to the handlers.  OnHint is triggered when
the mouse passes over the control. OnShowHint is triggered
when the mouse passes over the control and the application
is about to display the hint window for a Help Hint
*/
void __fastcall TAppEventForm::ApplicationEventsHint(TObject *Sender)
{
    lbOther->Items->Add("Event OnHint");
}

void __fastcall TAppEventForm::ApplicationEventsShowHint(AnsiString &HintStr,
      bool &CanShow, THintInfo &HintInfo)
{
    lbOther->Items->Add("Event OnShowHint");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents and a TListBox
in the form.  Select the TApplicationEvents, double click
on the OnHint event and the OnShowHint event and add the
following code to the handlers.  OnHint is triggered when
the mouse passes over the control. OnShowHint is triggered
when the mouse passes over the control and the application
is about to display the hint window for a Help Hint
}
procedure TMainForm.ApplicationEventsHint(Sender: TObject);
begin
  lbOther.Items.Add('Event OnHint');
end;

procedure TMainForm.ApplicationEventsShowHint(var HintStr: String;
  var CanShow: Boolean; var HintInfo: THintInfo);
begin
  lbOther.Items.Add('Event OnShowHint');
end;
```

See Also

GetLongHint (see page 6252)

GetShortHint (see page 6253)

HintHidePause (see page 13715)

OnHint (see page 1359)

CancelDispatch (see page 1351)

10 AutoAdap Namespace

This is namespace AutoAdap.

Topics

Name	Description
Classes (see page 1373)	The following table lists classes in this documentation.
Interfaces (see page 1434)	The following table lists interfaces in this documentation.
Types (see page 1475)	The following table lists types in this documentation.

Classes

	Name	Description
	TActionLink (see page 1373)	This is class AutoAdap.TActionLink.
	TAdapterActionArrayStub (see page 1376)	This is class AutoAdap.TAdapterActionArrayStub.
	TAdapterActionStub (see page 1379)	This is class AutoAdap.TAdapterActionStub.
	TAdapterComponentsStub (see page 1383)	This is class AutoAdap.TAdapterComponentsStub.
	TAdapterErrorsStub (see page 1384)	This is class AutoAdap.TAdapterErrorsStub.
	TAdapterErrorStub (see page 1387)	This is class AutoAdap.TAdapterErrorStub.
	TAdapterFieldStub (see page 1390)	This is class AutoAdap.TAdapterFieldStub.
	TAdapterFieldValuesStub (see page 1398)	This is class AutoAdap.TAdapterFieldValuesStub.
	TAdapterHiddenFieldsStub (see page 1400)	This is class AutoAdap.TAdapterHiddenFieldsStub.
	TAdapterHiddenFieldStub (see page 1403)	This is class AutoAdap.TAdapterHiddenFieldStub.
	TAdapterImageStub (see page 1405)	This is class AutoAdap.TAdapterImageStub.
	TAdapterRecordsStub (see page 1407)	This is class AutoAdap.TAdapterRecordsStub.
	TAdapterRecordStub (see page 1409)	This is class AutoAdap.TAdapterRecordStub.
	TAdapterStub (see page 1412)	This is class AutoAdap.TAdapterStub.
	TAdapterSubComponentStub (see page 1416)	This is class AutoAdap.TAdapterSubComponentStub.
	TAdapterValuesListStub (see page 1419)	This is class AutoAdap.TAdapterValuesListStub.
	TBaseAdapterStub (see page 1422)	This is class AutoAdap.TBaseAdapterStub.
	TBaseRecordsAdapterStub (see page 1425)	This is class AutoAdap.TBaseRecordsAdapterStub.
	TWebApplicationStub (see page 1427)	This is class AutoAdap.TWebApplicationStub.
	TWebEndUserStub (see page 1430)	This is class AutoAdap.TWebEndUserStub.

Interfaces

	Name	Description
	IAdapterErrorsStub (see page 1434)	This is class AutoAdap.IAdapterErrorsStub.
	IAdapterErrorStub (see page 1435)	This is class AutoAdap.IAdapterErrorStub.

	IAdapterFieldStub (see page 1438)	This is class AutoAdap.IAdapterFieldStub.
	IAdapterFieldValuesStub (see page 1448)	This is class AutoAdap.IAdapterFieldValuesStub.
	IAdapterHiddenFieldsStub (see page 1450)	This is class AutoAdap.IAdapterHiddenFieldsStub.
	IAdapterHiddenFieldStub (see page 1451)	This is class AutoAdap.IAdapterHiddenFieldStub.
	IAdapterImageStub (see page 1453)	This is class AutoAdap.IAdapterImageStub.
	IAdapterRecordStub (see page 1454)	This is class AutoAdap.IAdapterRecordStub.
	IAdapterStub (see page 1455)	This is class AutoAdap.IAdapterStub.
	IAdapterValuesListStub (see page 1459)	This is class AutoAdap.IAdapterValuesListStub.
	IApplicationStub (see page 1462)	This is class AutoAdap.IApplicationStub.
	IBaseAdapterStub (see page 1466)	This is class AutoAdap.IBaseAdapterStub.
	IBaseRecordsAdapterStub (see page 1468)	This is class AutoAdap.IBaseRecordsAdapterStub.
	IEndUserStub (see page 1470)	This is class AutoAdap.IEndUserStub.
	IImageProducerStub (see page 1473)	This is class AutoAdap.IImageProducerStub.

Types

Name	Description
TAdapterStubClass (see page 1475)	This is type AutoAdap.TAdapterStubClass.

10.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TActionLink (see page 1373)	This is class AutoAdap.TActionLink.
	TAdapterActionArrayStub (see page 1376)	This is class AutoAdap.TAdapterActionArrayStub.
	TAdapterActionStub (see page 1379)	This is class AutoAdap.TAdapterActionStub.
	TAdapterComponentsStub (see page 1383)	This is class AutoAdap.TAdapterComponentsStub.
	TAdapterErrorsStub (see page 1384)	This is class AutoAdap.TAdapterErrorsStub.
	TAdapterErrorStub (see page 1387)	This is class AutoAdap.TAdapterErrorStub.
	TAdapterFieldStub (see page 1390)	This is class AutoAdap.TAdapterFieldStub.
	TAdapterFieldValuesStub (see page 1398)	This is class AutoAdap.TAdapterFieldValuesStub.
	TAdapterHiddenFieldsStub (see page 1400)	This is class AutoAdap.TAdapterHiddenFieldsStub.
	TAdapterHiddenFieldStub (see page 1403)	This is class AutoAdap.TAdapterHiddenFieldStub.
	TAdapterImageStub (see page 1405)	This is class AutoAdap.TAdapterImageStub.
	TAdapterRecordsStub (see page 1407)	This is class AutoAdap.TAdapterRecordsStub.
	TAdapterRecordStub (see page 1409)	This is class AutoAdap.TAdapterRecordStub.
	TAdapterStub (see page 1412)	This is class AutoAdap.TAdapterStub.
	TAdapterSubComponentStub (see page 1416)	This is class AutoAdap.TAdapterSubComponentStub.
	TAdapterValuesListStub (see page 1419)	This is class AutoAdap.TAdapterValuesListStub.
	TBaseAdapterStub (see page 1422)	This is class AutoAdap.TBaseAdapterStub.
	TBaseRecordsAdapterStub (see page 1425)	This is class AutoAdap.TBaseRecordsAdapterStub.
	TWebApplicationStub (see page 1427)	This is class AutoAdap.TWebApplicationStub.
	TWebEndUserStub (see page 1430)	This is class AutoAdap.TWebEndUserStub.

10.1.1 TActionLink Class

Class Hierarchy

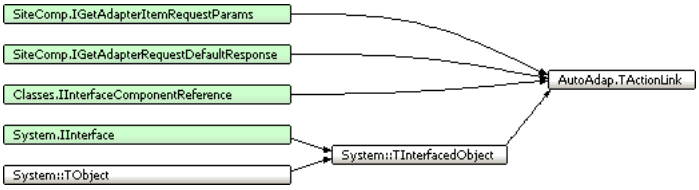

Pascal

```
TActionLink = class(TInterfacedObject, IGetAdapterItemRequestParams,
IGetAdapterRequestDefaultResponse, IInterfaceComponentReference);
```

C++

```
class TActionLink : public TInterfacedObject, public IGetAdapterItemRequestParams, public
IGetAdapterRequestDefaultResponse, public IInterfaceComponentReference;
```

File

AutoAdap

Description

This is class AutoAdap.TActionLink.

10.1.1.1 TActionLink Members

The following tables list the members exposed by TActionLink.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetAdapterItemRequestParams (see page 1375)	This is GetAdapterItemRequestParams, a member of class TActionLink.
	GetComponent (see page 1376)	Returns a reference to the component associated by the implementing class.
	GetFailurePage (see page 1376)	This is GetFailurePage, a member of class TActionLink.
	GetSuccessPage (see page 1376)	This is GetSuccessPage, a member of class TActionLink.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1375)	This is Create, a member of class TActionLink.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	FailurePage (see page 19694)	This is FailurePage, a member of class IGetAdapterRequestDefaultResponse.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	SuccessPage (see page 19694)	This is SuccessPage, a member of class IGetAdapterRequestDefaultResponse.

10.1.1.2 TActionLink.Create Constructor

Pascal

```
constructor Create(AAction: IInterface; const APage: string; const AErrorPage: string);
```

C++

```
__fastcall TActionLink(IInterface AAction, const AnsiString APage, const AnsiString
AErrorPage);
```

Description

This is Create, a member of class TActionLink.

10.1.1.3 TActionLink Methods

The methods of the TActionLink class are listed here.

Protected Methods

	Name	Description
	GetAdapterItemRequestParams (see page 1375)	This is GetAdapterItemRequestParams, a member of class TActionLink.
	GetComponent (see page 1376)	Returns a reference to the component associated by the implementing class.
	GetFailurePage (see page 1376)	This is GetFailurePage, a member of class TActionLink.
	GetSuccessPage (see page 1376)	This is GetSuccessPage, a member of class TActionLink.

10.1.1.3.1 TActionLink.GetAdapterItemRequestParams Method

Pascal

```
procedure GetAdapterItemRequestParams(var AIdentifier: string; AParams:
IAdapterItemRequestParams);
```

C++

```
__fastcall GetAdapterItemRequestParams(AnsiString AIdentifier, IAdapterItemRequestParams
AParams);
```

Description

This is GetAdapterItemRequestParams, a member of class TActionLink.

10.1.1.3.2 TActionLink.GetComponent Method

Returns a reference to the component associated by the implementing class.

Pascal

```
function GetComponent: TComponent;
```

C++

```
__fastcall TComponent * GetComponent();
```

Description

GetComponent returns a TComponent value that is associated with the implementing class. For example, the TComponent class itself implements the GetComponent to return a pointer to itself.

See Also

TComponent (see page 2762)

10.1.1.3.3 TActionLink.GetFailurePage Method

Pascal

```
function GetFailurePage: string;
```

C++

```
__fastcall AnsiString GetFailurePage();
```

Description

This is GetFailurePage, a member of class TActionLink.

10.1.1.3.4 TActionLink.GetSuccessPage Method

Pascal

```
function GetSuccessPage: string;
```

C++

```
__fastcall AnsiString GetSuccessPage();
```

Description

This is GetSuccessPage, a member of class TActionLink.

10.1.2 TAdapterActionArrayStub Class

Class Hierarchy

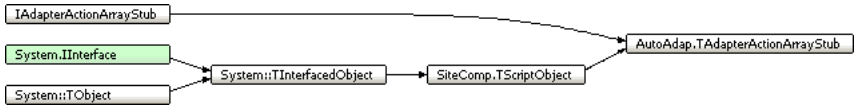

Pascal

```
TAdapterActionArrayStub = class(TScriptObject, IAdapterActionArrayStub);
```

C++

```
class TAdapterActionArrayStub : public TScriptObject, public IAdapterActionArrayStub;
```

File

AutoAdap

Description

This is class AutoAdap.TAdapterActionArrayStub.

10.1.2.1 TAdapterActionArrayStub Members

The following tables list the members exposed by TAdapterActionArrayStub.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1378)	This is Create, a member of class TAdapterActionArrayStub.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Action (see page 1378)	This is Action, a member of class TAdapterActionArrayStub.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

10.1.2.2 TAdapterActionArrayStub.Create Constructor

Pascal

```
constructor Create(AAction: IInterfaceComponentReference);
```

C++

```
__fastcall TAdapterActionArrayStub(IInterfaceComponentReference AAction);
```

Description

This is Create, a member of class TAdapterActionArrayStub.

10.1.2.3 TAdapterActionArrayStub Properties

The properties of the TAdapterActionArrayStub class are listed here.

Public Properties

	Name	Description
R	Action (see page 1378)	This is Action, a member of class TAdapterActionArrayStub.

10.1.2.3.1 TAdapterActionArrayStub.Action Property

Pascal

```
property Action: IInterfaceComponentReference;
```

C++

```
__property IInterfaceComponentReference Action;
```

Description

This is Action, a member of class TAdapterActionArrayStub.

10.1.3 TAdapterActionStub Class

Class Hierarchy

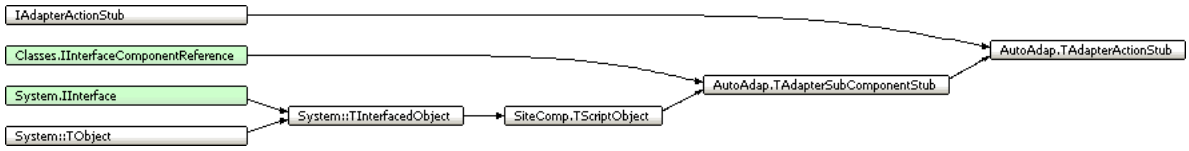

Pascal

```
TAdapterActionStub = class(TAdapterSubComponentStub, IAdapterActionStub);
```

C++

```
class TAdapterActionStub : public TAdapterSubComponentStub, public IAdapterActionStub;
```

File

AutoAdap

Description

This is class AutoAdap.TAdapterActionStub.

10.1.3.1 TAdapterActionStub Members

The following tables list the members exposed by TAdapterActionStub.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
A	CreateActionArrayStub (see page 1381)	This is CreateActionArrayStub, a member of class TAdapterActionStub.
A	CreateActionStub (see page 1381)	This is CreateActionStub, a member of class TAdapterActionStub.
	Get_Array_ (see page 1381)	This is Get_Array_, a member of class TAdapterActionStub.
	Get_AsFieldValue (see page 1382)	This is Get_AsFieldValue, a member of class TAdapterActionStub.
	Get_AsHREF (see page 1382)	This is Get_AsHREF, a member of class TAdapterActionStub.
	Get_CanExecute (see page 1382)	This is Get_CanExecute, a member of class TAdapterActionStub.
	Get_DisplayLabel (see page 1382)	This is Get_DisplayLabel, a member of class TAdapterActionStub.
	Get_DisplayStyle (see page 1382)	This is Get_DisplayStyle, a member of class TAdapterActionStub.
	Get_Enabled (see page 1383)	This is Get_Enabled, a member of class TAdapterActionStub.
	Get_Name (see page 1418)	This is Get_Name, a member of class TAdapterSubComponentStub.
	Get_Visible (see page 1383)	This is Get_Visible, a member of class TAdapterActionStub.
	GetComponent (see page 1418)	Returns a reference to the component associated by the implementing class.
	LinkToPage (see page 1383)	This is LinkToPage, a member of class TAdapterActionStub.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 1417)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ObjectIntf (see page 1419)	This is ObjectIntf, a member of class TAdapterSubComponentStub.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

10.1.3.2 TAdapterActionStub Methods

The methods of the TAdapterActionStub class are listed here.

Protected Methods

	Name	Description
	CreateActionArrayStub (see page 1381)	This is CreateActionArrayStub, a member of class TAdapterActionStub.
	CreateActionStub (see page 1381)	This is CreateActionStub, a member of class TAdapterActionStub.
	Get_Array_ (see page 1381)	This is Get_Array_, a member of class TAdapterActionStub.
	Get_AsFieldValue (see page 1382)	This is Get_AsFieldValue, a member of class TAdapterActionStub.
	Get_AsHREF (see page 1382)	This is Get_AsHREF, a member of class TAdapterActionStub.
	Get_CanExecute (see page 1382)	This is Get_CanExecute, a member of class TAdapterActionStub.
	Get_DisplayLabel (see page 1382)	This is Get_DisplayLabel, a member of class TAdapterActionStub.
	Get_DisplayStyle (see page 1382)	This is Get_DisplayStyle, a member of class TAdapterActionStub.
	Get_Enabled (see page 1383)	This is Get_Enabled, a member of class TAdapterActionStub.
	Get_Visible (see page 1383)	This is Get_Visible, a member of class TAdapterActionStub.
	LinkToPage (see page 1383)	This is LinkToPage, a member of class TAdapterActionStub.

10.1.3.2.1 TAdapterActionStub.CreateActionArrayStub Method

Pascal

```
function CreateActionArrayStub(O: IInterfaceComponentReference): IInterface; virtual; abstract;
```

C++

```
virtual __fastcall IInterface CreateActionArrayStub(IInterfaceComponentReference O) = 0;
```

Description

This is CreateActionArrayStub, a member of class TAdapterActionStub.

10.1.3.2.2 TAdapterActionStub.CreateActionStub Method

Pascal

```
function CreateActionStub(O: IInterfaceComponentReference): IInterface; virtual; abstract;
overload;
```

C++

```
virtual __fastcall IInterface CreateActionStub(IInterfaceComponentReference O) = 0;
```

Description

This is CreateActionStub, a member of class TAdapterActionStub.

10.1.3.2.3 TAdapterActionStub.Get_Array_ Method

Pascal

```
function Get_Array_: IInterface;
```

C++

```
__fastcall IInterface Get_Array_();
```

Description

This is Get_Array_, a member of class TAdapterActionStub.

10.1.3.2.4 TAdapterActionStub.Get_AsFieldValue Method

Pascal

```
function Get_AsFieldValue: string;
```

C++

```
__fastcall AnsiString Get_AsFieldValue();
```

Description

This is Get_AsFieldValue, a member of class TAdapterActionStub.

10.1.3.2.5 TAdapterActionStub.Get_AsHREF Method

Pascal

```
function Get_AsHREF: string;
```

C++

```
__fastcall AnsiString Get_AsHREF();
```

Description

This is Get_AsHREF, a member of class TAdapterActionStub.

10.1.3.2.6 TAdapterActionStub.Get_CanExecute Method

Pascal

```
function Get_CanExecute: Boolean;
```

C++

```
__fastcall Boolean Get_CanExecute();
```

Description

This is Get_CanExecute, a member of class TAdapterActionStub.

10.1.3.2.7 TAdapterActionStub.Get_DisplayLabel Method

Pascal

```
function Get_DisplayLabel: string;
```

C++

```
__fastcall AnsiString Get_DisplayLabel();
```

Description

This is Get_DisplayLabel, a member of class TAdapterActionStub.

10.1.3.2.8 TAdapterActionStub.Get_DisplayStyle Method

Pascal

```
function Get_DisplayStyle: string;
```

C++

```
__fastcall AnsiString Get_DisplayStyle();
```

Description

This is Get_DisplayStyle, a member of class TAdapterActionStub.

10.1.3.2.9 TAdapterActionStub.Get_Enabled Method

Pascal

```
function Get_Enabled: Boolean;
```

C++

```
__fastcall Boolean Get_Enabled();
```

Description

This is Get_Enabled, a member of class TAdapterActionStub.

10.1.3.2.10 TAdapterActionStub.Get_Visible Method

Pascal

```
function Get_Visible: Boolean;
```

C++

```
__fastcall Boolean Get_Visible();
```

Description

This is Get_Visible, a member of class TAdapterActionStub.

10.1.3.2.11 TAdapterActionStub.LinkToPage Method

Pascal

```
function LinkToPage(const PageSuccess: string; const PageFail: string): IInterface;
```

C++

```
__fastcall IInterface LinkToPage(const AnsiString PageSuccess, const AnsiString PageFail);
```

Description

This is LinkToPage, a member of class TAdapterActionStub.

10.1.4 TAdapterComponentsStub Class

Class Hierarchy

Pascal

```
TAdapterComponentsStub = class(TWebComponentsStub, IComponentsStub);
```

C++

```
class TAdapterComponentsStub : public TWebComponentsStub, public IComponentsStub;
```

File

AutoAdap

Description

This is class AutoAdap.TAdapterComponentsStub.

10.1.4.1 TAdapterComponentsStub Members

The following tables list the members exposed by TAdapterComponentsStub.

Protected Methods

	Name	Description
	FindObject (see page 1384)	This is FindObject, a member of class TAdapterComponentsStub.

10.1.4.2 TAdapterComponentsStub Methods

The methods of the TAdapterComponentsStub class are listed here.

Protected Methods

	Name	Description
	FindObject (see page 1384)	This is FindObject, a member of class TAdapterComponentsStub.

10.1.4.2.1 TAdapterComponentsStub.FindObject Method

Pascal

```
function FindObject(const AName: string): TObject; override;
```

C++

```
virtual __fastcall TObject * FindObject(const AnsiString AName);
```

Description

This is FindObject, a member of class TAdapterComponentsStub.

10.1.5 TAdapterErrorsStub Class

Class Hierarchy

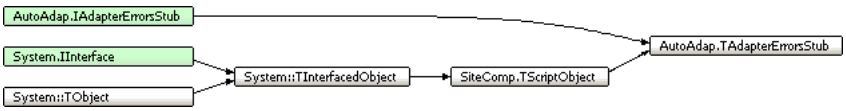

Pascal

```
TAdapterErrorsStub = class(TScriptObject, IAdapterErrorsStub);
```

C++

```
class TAdapterErrorsStub : public TScriptObject, public IAdapterErrorsStub;
```

File

AutoAdap

Description

This is class AutoAdap.TAdapterErrorsStub.

10.1.5.1 TAdapterErrorsStub Members

The following tables list the members exposed by TAdapterErrorsStub.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
A	CreateErrorsStub (see page 1386)	This is CreateErrorsStub, a member of class TAdapterErrorsStub.
	DefineLabel (see page 1387)	This is DefineLabel, a member of class TAdapterErrorsStub.
	FieldErrors (see page 1387)	This is FieldErrors, a member of class TAdapterErrorsStub.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1386)	This is Create, a member of class TAdapterErrorsStub.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Errors (see page 1387)	This is Errors, a member of class TAdapterErrorsStub.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

10.1.5.2 TAdapterErrorsStub.Create Constructor

Pascal

```
constructor Create(AErrors: TObject);
```

C++

```
__fastcall TAdapterErrorsStub(TObject * AErrors);
```

Description

This is Create, a member of class TAdapterErrorsStub.

10.1.5.3 TAdapterErrorsStub Methods

The methods of the TAdapterErrorsStub class are listed here.

Protected Methods

	Name	Description
A	CreateErrorsStub (see page 1386)	This is CreateErrorsStub, a member of class TAdapterErrorsStub.
	DefineLabel (see page 1387)	This is DefineLabel, a member of class TAdapterErrorsStub.
	FieldErrors (see page 1387)	This is FieldErrors, a member of class TAdapterErrorsStub.

10.1.5.3.1 TAdapterErrorsStub.CreateErrorsStub Method

Pascal

```
function CreateErrorsStub(O: TObject): IInterface; virtual; abstract;
```

C++

```
virtual __fastcall IInterface CreateErrorsStub(TObject * O) = 0;
```

Description

This is CreateErrorsStub, a member of class TAdapterErrorsStub.

10.1.5.3.2 TAdapterErrorsStub.DefineLabel Method

Pascal

```
procedure DefineLabel(const Object_: IUnknown; const Value: string);
```

C++

```
__fastcall DefineLabel(const IUnknown Object_, const AnsiString Value);
```

Description

This is DefineLabel, a member of class TAdapterErrorsStub.

10.1.5.3.3 TAdapterErrorsStub.FieldErrors Method

Pascal

```
function FieldErrors(const Object_: IUnknown): IInterface;
```

C++

```
__fastcall IInterface FieldErrors(const IUnknown Object_);
```

Description

This is FieldErrors, a member of class TAdapterErrorsStub.

10.1.5.4 TAdapterErrorsStub Properties

The properties of the TAdapterErrorsStub class are listed here.

Public Properties

	Name	Description
	Errors (see page 1387)	This is Errors, a member of class TAdapterErrorsStub.

10.1.5.4.1 TAdapterErrorsStub.Errors Property

Pascal

```
property Errors: TObject;
```

C++

```
__property TObject * Errors;
```

Description

This is Errors, a member of class TAdapterErrorsStub.

10.1.6 TAdapterErrorStub Class

Class Hierarchy

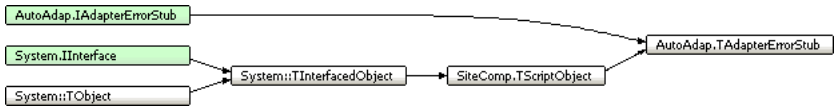

Pascal

```
TAdapterErrorStub = class(TScriptObject, IAdapterErrorStub);
```

C++

```
class TAdapterErrorStub : public TScriptObject, public IAdapterErrorStub;
```

File

AutoAdap

Description

This is class AutoAdap.TAdapterErrorStub.

10.1.6.1 TAdapterErrorStub Members

The following tables list the members exposed by TAdapterErrorStub.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	Get_Field (see page 1389)	This is Get_Field, a member of class TAdapterErrorStub.
	Get_ID (see page 1390)	This is Get_ID, a member of class TAdapterErrorStub.
	Get_Message (see page 1390)	This is Get_Message, a member of class TAdapterErrorStub.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1389)	This is Create, a member of class TAdapterErrorStub.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Field (see page 1437)	This is Field, a member of class IAdapterErrorStub.
R	ID (see page 1437)	This is ID, a member of class IAdapterErrorStub.
R	Message (see page 1437)	This is Message, a member of class IAdapterErrorStub.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

10.1.6.2 TAdapterErrorStub.Create Constructor

Pascal

```
constructor Create(const AError: IAdapterError);
```

C++

```
__fastcall TAdapterErrorStub(const IAdapterError AError);
```

Description

This is Create, a member of class TAdapterErrorStub.

10.1.6.3 TAdapterErrorStub Methods

The methods of the TAdapterErrorStub class are listed here.

Protected Methods

	Name	Description
	Get_Field (see page 1389)	This is Get_Field, a member of class TAdapterErrorStub.
	Get_ID (see page 1390)	This is Get_ID, a member of class TAdapterErrorStub.
	Get_Message (see page 1390)	This is Get_Message, a member of class TAdapterErrorStub.

10.1.6.3.1 TAdapterErrorStub.Get_Field Method

Pascal

```
function Get_Field: IInterface;
```

C++

```
__fastcall IInterface Get_Field();
```

Description

This is Get_Field, a member of class TAdapterErrorStub.

10.1.6.3.2 TAdapterErrorStub.Get_ID Method

Pascal

```
function Get_ID: Integer;
```

C++

```
__fastcall int Get_ID();
```

Description

This is Get_ID, a member of class TAdapterErrorStub.

10.1.6.3.3 TAdapterErrorStub.Get_Message Method

Pascal

```
function Get_Message: string;
```

C++

```
__fastcall AnsiString Get_Message();
```

Description

This is Get_Message, a member of class TAdapterErrorStub.

10.1.7 TAdapterFieldStub Class

Class Hierarchy

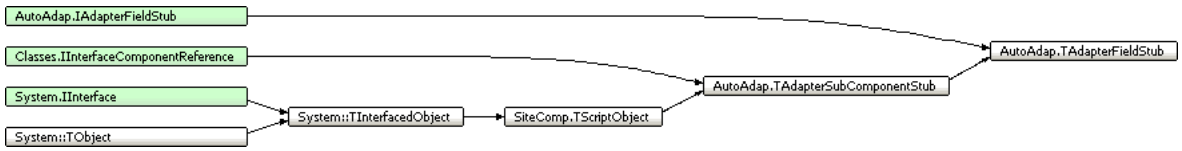

Pascal

```
TAdapterFieldStub = class(TAdapterSubComponentStub, IAdapterFieldStub);
```

C++

```
class TAdapterFieldStub : public TAdapterSubComponentStub, public IAdapterFieldStub;
```

File

AutoAdap

Description

This is class AutoAdap.TAdapterFieldStub.

10.1.7.1 TAdapterFieldStub Members

The following tables list the members exposed by TAdapterFieldStub.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
A	CreateFieldValuesStub (see page 1393)	This is CreateFieldValuesStub, a member of class TAdapterFieldStub.
A	CreateImageStub (see page 1393)	This is CreateImageStub, a member of class TAdapterFieldStub.
A	CreateValuesListStub (see page 1394)	This is CreateValuesListStub, a member of class TAdapterFieldStub.
	Get_CanModify (see page 1394)	This is Get_CanModify, a member of class TAdapterFieldStub.
	Get_CanView (see page 1394)	This is Get_CanView, a member of class TAdapterFieldStub.
	Get_DisplayLabel (see page 1394)	This is Get_DisplayLabel, a member of class TAdapterFieldStub.
	Get_DisplayStyle (see page 1394)	This is Get_DisplayStyle, a member of class TAdapterFieldStub.
	Get_DisplayText (see page 1395)	This is Get_DisplayText, a member of class TAdapterFieldStub.
	Get_DisplayWidth (see page 1395)	This is Get_DisplayWidth, a member of class TAdapterFieldStub.
	Get_EditText (see page 1395)	This is Get_EditText, a member of class TAdapterFieldStub.
	Get_Image (see page 1395)	This is Get_Image, a member of class TAdapterFieldStub.
	Get_InputName (see page 1396)	This is Get_InputName, a member of class TAdapterFieldStub.
	Get_InputStyle (see page 1396)	This is Get_InputStyle, a member of class TAdapterFieldStub.
	Get_MaxLength (see page 1396)	This is Get_MaxLength, a member of class TAdapterFieldStub.
	Get_Name (see page 1418)	This is Get_Name, a member of class TAdapterSubComponentStub.
	Get_Required (see page 1396)	This is Get_Required, a member of class TAdapterFieldStub.
	Get_Value (see page 1396)	This is Get_Value, a member of class TAdapterFieldStub.
	Get_Values (see page 1397)	This is Get_Values, a member of class TAdapterFieldStub.
	Get_ValuesList (see page 1397)	This is Get_ValuesList, a member of class TAdapterFieldStub.
	Get_ViewMode (see page 1397)	This is Get_ViewMode, a member of class TAdapterFieldStub.
	Get_Visible (see page 1397)	This is Get_Visible, a member of class TAdapterFieldStub.
	GetComponent (see page 1418)	Returns a reference to the component associated by the implementing class.
	IsEqual (see page 1398)	This is IsEqual, a member of class TAdapterFieldStub.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 1417)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	Get_Name (see page 1442)	This is Get_Name, a member of class IAdapterFieldStub.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	CanModify (see page 1444)	This is CanModify, a member of class IAdapterFieldStub.
R	CanView (see page 1444)	This is CanView, a member of class IAdapterFieldStub.
R	DisplayLabel (see page 1445)	This is DisplayLabel, a member of class IAdapterFieldStub.
R	DisplayStyle (see page 1445)	This is DisplayStyle, a member of class IAdapterFieldStub.
R	DisplayText (see page 1445)	This is DisplayText, a member of class IAdapterFieldStub.
R	DisplayWidth (see page 1445)	This is DisplayWidth, a member of class IAdapterFieldStub.
R	EditText (see page 1445)	This is EditText, a member of class IAdapterFieldStub.
R	Image (see page 1446)	This is Image, a member of class IAdapterFieldStub.
R	InputName (see page 1446)	This is InputName, a member of class IAdapterFieldStub.
R	InputStyle (see page 1446)	This is InputStyle, a member of class IAdapterFieldStub.
R	MaxLength (see page 1446)	This is MaxLength, a member of class IAdapterFieldStub.
R	Name (see page 1447)	This is Name, a member of class IAdapterFieldStub.
R	ObjectIntf (see page 1419)	This is ObjectIntf, a member of class TAdapterSubComponentStub.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	Required (see page 1447)	This is Required, a member of class IAdapterFieldStub.
R	Value (see page 1447)	This is Value, a member of class IAdapterFieldStub.
R	Values (see page 1447)	This is Values, a member of class IAdapterFieldStub.
R	ValuesList (see page 1447)	This is ValuesList, a member of class IAdapterFieldStub.
R	ViewMode (see page 1448)	This is ViewMode, a member of class IAdapterFieldStub.
R	Visible (see page 1448)	This is Visible, a member of class IAdapterFieldStub.

10.1.7.2 TAdapterFieldStub Methods

The methods of the TAdapterFieldStub class are listed here.

Protected Methods

	Name	Description
	CreateFieldValuesStub (see page 1393)	This is CreateFieldValuesStub, a member of class TAdapterFieldStub.
	CreateImageStub (see page 1393)	This is CreateImageStub, a member of class TAdapterFieldStub.
	CreateValuesListStub (see page 1394)	This is CreateValuesListStub, a member of class TAdapterFieldStub.
	Get_CanModify (see page 1394)	This is Get_CanModify, a member of class TAdapterFieldStub.
	Get_CanView (see page 1394)	This is Get_CanView, a member of class TAdapterFieldStub.
	Get_DisplayLabel (see page 1394)	This is Get_DisplayLabel, a member of class TAdapterFieldStub.
	Get_DisplayStyle (see page 1394)	This is Get_DisplayStyle, a member of class TAdapterFieldStub.
	Get_DisplayText (see page 1395)	This is Get_DisplayText, a member of class TAdapterFieldStub.
	Get_DisplayWidth (see page 1395)	This is Get_DisplayWidth, a member of class TAdapterFieldStub.
	Get_EditText (see page 1395)	This is Get_EditText, a member of class TAdapterFieldStub.
	Get_Image (see page 1395)	This is Get_Image, a member of class TAdapterFieldStub.
	Get_InputName (see page 1396)	This is Get_InputName, a member of class TAdapterFieldStub.
	Get_InputStyle (see page 1396)	This is Get_InputStyle, a member of class TAdapterFieldStub.
	Get_MaxLength (see page 1396)	This is Get_MaxLength, a member of class TAdapterFieldStub.
	Get_Required (see page 1396)	This is Get_Required, a member of class TAdapterFieldStub.
	Get_Value (see page 1396)	This is Get_Value, a member of class TAdapterFieldStub.
	Get_Values (see page 1397)	This is Get_Values, a member of class TAdapterFieldStub.
	Get_ValuesList (see page 1397)	This is Get_ValuesList, a member of class TAdapterFieldStub.
	Get_ViewMode (see page 1397)	This is Get_ViewMode, a member of class TAdapterFieldStub.
	Get_Visible (see page 1397)	This is Get_Visible, a member of class TAdapterFieldStub.
	IsEqual (see page 1398)	This is IsEqual, a member of class TAdapterFieldStub.

10.1.7.2.1 TAdapterFieldStub.CreateFieldValuesStub Method

Pascal

```
function CreateFieldValuesStub(O: TComponent): IInterface; virtual; abstract;
```

C++

```
virtual __fastcall IInterface CreateFieldValuesStub(TComponent * O) = 0;
```

Description

This is CreateFieldValuesStub, a member of class TAdapterFieldStub.

10.1.7.2.2 TAdapterFieldStub.CreateImageStub Method

Pascal

```
function CreateImageStub(AdapterImage: IInterface): IInterface; virtual; abstract;
```

C++

```
virtual __fastcall IInterface CreateImageStub(IInterface AdapterImage) = 0;
```

Description

This is CreateImageStub, a member of class TAdapterFieldStub.

10.1.7.2.3 TAdapterFieldStub.CreateValuesListStub Method

Pascal

```
function CreateValuesListStub(O: TComponent): IInterface; virtual; abstract;
```

C++

```
virtual __fastcall IInterface CreateValuesListStub(TComponent * O) = 0;
```

Description

This is CreateValuesListStub, a member of class TAdapterFieldStub.

10.1.7.2.4 TAdapterFieldStub.Get_CanModify Method

Pascal

```
function Get_CanModify: Boolean;
```

C++

```
__fastcall Boolean Get_CanModify();
```

Description

This is Get_CanModify, a member of class TAdapterFieldStub.

10.1.7.2.5 TAdapterFieldStub.Get_CanView Method

Pascal

```
function Get_CanView: Boolean;
```

C++

```
__fastcall Boolean Get_CanView();
```

Description

This is Get_CanView, a member of class TAdapterFieldStub.

10.1.7.2.6 TAdapterFieldStub.Get_DisplayLabel Method

Pascal

```
function Get_DisplayLabel: string;
```

C++

```
__fastcall AnsiString Get_DisplayLabel();
```

Description

This is Get_DisplayLabel, a member of class TAdapterFieldStub.

10.1.7.2.7 TAdapterFieldStub.Get_DisplayStyle Method

Pascal

```
function Get_DisplayStyle: string;
```

C++

```
__fastcall AnsiString Get_DisplayStyle();
```

Description

This is Get_DisplayStyle, a member of class TAdapterFieldStub.

10.1.7.2.8 TAdapterFieldStub.Get_DisplayText Method

Pascal

```
function Get_DisplayText: string;
```

C++

```
__fastcall AnsiString Get_DisplayText();
```

Description

This is Get_DisplayText, a member of class TAdapterFieldStub.

10.1.7.2.9 TAdapterFieldStub.Get_DisplayWidth Method

Pascal

```
function Get_DisplayWidth: Integer;
```

C++

```
__fastcall int Get_DisplayWidth();
```

Description

This is Get_DisplayWidth, a member of class TAdapterFieldStub.

10.1.7.2.10 TAdapterFieldStub.Get_EditText Method

Pascal

```
function Get_EditText: string;
```

C++

```
__fastcall AnsiString Get_EditText();
```

Description

This is Get_EditText, a member of class TAdapterFieldStub.

10.1.7.2.11 TAdapterFieldStub.Get_Image Method

Pascal

```
function Get_Image: IInterface;
```

C++

```
__fastcall IInterface Get_Image();
```

Description

This is Get_Image, a member of class TAdapterFieldStub.

10.1.7.2.12 TAdapterFieldStub.Get_InputName Method

Pascal

```
function Get_InputName: string;
```

C++

```
__fastcall AnsiString Get_InputName();
```

Description

This is Get_InputName, a member of class TAdapterFieldStub.

10.1.7.2.13 TAdapterFieldStub.Get_InputStyle Method

Pascal

```
function Get_InputStyle: string;
```

C++

```
__fastcall AnsiString Get_InputStyle();
```

Description

This is Get_InputStyle, a member of class TAdapterFieldStub.

10.1.7.2.14 TAdapterFieldStub.Get_MaxLength Method

Pascal

```
function Get_MaxLength: Integer;
```

C++

```
__fastcall int Get_MaxLength();
```

Description

This is Get_MaxLength, a member of class TAdapterFieldStub.

10.1.7.2.15 TAdapterFieldStub.Get_Required Method

Pascal

```
function Get_Required: Boolean;
```

C++

```
__fastcall Boolean Get_Required();
```

Description

This is Get_Required, a member of class TAdapterFieldStub.

10.1.7.2.16 TAdapterFieldStub.Get_Value Method

Pascal

```
function Get_Value: Variant;
```

C++

```
__fastcall Variant Get_Value();
```

Description

This is Get_Value, a member of class TAdapterFieldStub.

10.1.7.2.17 TAdapterFieldStub.Get_Values Method

Pascal

```
function Get_Values: IInterface;
```

C++

```
__fastcall IInterface Get_Values();
```

Description

This is Get_Values, a member of class TAdapterFieldStub.

10.1.7.2.18 TAdapterFieldStub.Get_ValuesList Method

Pascal

```
function Get_ValuesList: IInterface;
```

C++

```
__fastcall IInterface Get_ValuesList();
```

Description

This is Get_ValuesList, a member of class TAdapterFieldStub.

10.1.7.2.19 TAdapterFieldStub.Get_ViewMode Method

Pascal

```
function Get_ViewMode: string;
```

C++

```
__fastcall AnsiString Get_ViewMode();
```

Description

This is Get_ViewMode, a member of class TAdapterFieldStub.

10.1.7.2.20 TAdapterFieldStub.Get_Visible Method

Pascal

```
function Get_Visible: Boolean;
```

C++

```
__fastcall Boolean Get_Visible();
```

Description

This is Get_Visible, a member of class TAdapterFieldStub.

10.1.7.2.21 TAdapterFieldStub.IsEqual Method

Pascal

```
function IsEqual(Value: Variant): Boolean;
```

C++

```
__fastcall Boolean IsEqual(Variant Value);
```

Description

This is IsEqual, a member of class TAdapterFieldStub.

10.1.8 TAdapterFieldValuesStub Class

Class Hierarchy

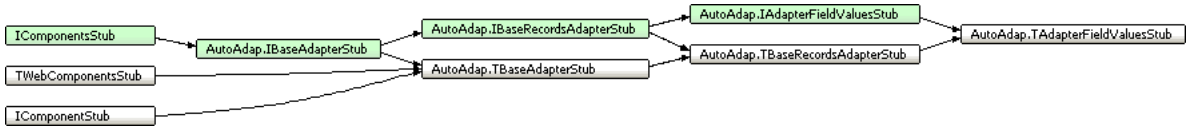

Pascal

```
TAdapterFieldValuesStub = class(TBaseRecordsAdapterStub, IAdapterFieldValuesStub);
```

C++

```
class TAdapterFieldValuesStub : public TBaseRecordsAdapterStub, public IAdapterFieldValuesStub;
```

File

AutoAdap

Description

This is class AutoAdap.TAdapterFieldValuesStub.

10.1.8.1 TAdapterFieldValuesStub Members

The following tables list the members exposed by TAdapterFieldValuesStub.

Protected Fields

	Name	Description
	FActions (see page 1423)	This is FActions, a member of class TBaseAdapterStub.
	FFields (see page 1424)	This is FFields, a member of class TBaseAdapterStub.

Protected Methods

	Name	Description
	CreateComponentsStub (see page 1424)	This is CreateComponentsStub, a member of class TBaseAdapterStub.
	CreateRecordsStub (see page 1427)	This is CreateRecordsStub, a member of class TBaseRecordsAdapterStub.
	FindObject (see page 1399)	This is FindObject, a member of class TAdapterFieldValuesStub.
	Get_Actions (see page 1424)	This is Get_Actions, a member of class TBaseAdapterStub.
	Get_ClassName_ (see page 1425)	This is Get_ClassName_, a member of class TBaseAdapterStub.
	Get_Fields (see page 1425)	This is Get_Fields, a member of class TBaseAdapterStub.
	Get_Name_ (see page 1425)	This is Get_Name_, a member of class TBaseAdapterStub.

	Get_Records (see page 1399)	This is Get_Records, a member of class TAdapterFieldValuesStub.
	Get_Value (see page 1400)	This is Get_Value, a member of class TAdapterFieldValuesStub.
	HasValue (see page 1400)	This is HasValue, a member of class TAdapterFieldValuesStub.

Public Methods

	Name	Description
	Get_Actions (see page 1467)	This is Get_Actions, a member of class IBaseAdapterStub.
	Get_ClassName_ (see page 1467)	This is Get_ClassName_, a member of class IBaseAdapterStub.
	Get_Fields (see page 1467)	This is Get_Fields, a member of class IBaseAdapterStub.
	Get_Name_ (see page 1467)	This is Get_Name_, a member of class IBaseAdapterStub.

Public Properties

	Name	Description
R	Actions (see page 1468)	This is Actions, a member of class IBaseAdapterStub.
R	Fields (see page 1468)	This is Fields, a member of class IBaseAdapterStub.
R	Records (see page 1469)	This is Records, a member of class IBaseRecordsAdapterStub.
R	Value (see page 1450)	This is Value, a member of class IAdapterFieldValuesStub.

10.1.8.2 TAdapterFieldValuesStub Methods

The methods of the TAdapterFieldValuesStub class are listed here.

Protected Methods

	Name	Description
V	FindObject (see page 1399)	This is FindObject, a member of class TAdapterFieldValuesStub.
	Get_Records (see page 1399)	This is Get_Records, a member of class TAdapterFieldValuesStub.
	Get_Value (see page 1400)	This is Get_Value, a member of class TAdapterFieldValuesStub.
	HasValue (see page 1400)	This is HasValue, a member of class TAdapterFieldValuesStub.

10.1.8.2.1 TAdapterFieldValuesStub.FindObject Method

Pascal

```
function FindObject(const AName: string): TObject; override;
```

C++

```
virtual __fastcall TObject * FindObject(const AnsiString AName);
```

Description

This is FindObject, a member of class TAdapterFieldValuesStub.

10.1.8.2.2 TAdapterFieldValuesStub.Get_Records Method

Pascal

```
function Get_Records: IInterface;
```

C++

```
__fastcall IInterface Get_Records();
```

Description

This is Get_Records, a member of class TAdapterFieldValuesStub.

10.1.8.2.3 TAdapterFieldValuesStub.Get_Value Method

Pascal

```
function Get_Value: Variant;
```

C++

```
__fastcall Variant Get_Value();
```

Description

This is Get_Value, a member of class TAdapterFieldValuesStub.

10.1.8.2.4 TAdapterFieldValuesStub.HasValue Method

Pascal

```
function HasValue(Value: Variant): Boolean;
```

C++

```
__fastcall Boolean HasValue(Variant Value);
```

Description

This is HasValue, a member of class TAdapterFieldValuesStub.

10.1.9 TAdapterHiddenFieldsStub Class

Class Hierarchy

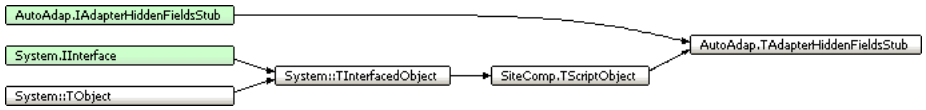

Pascal

```
TAdapterHiddenFieldsStub = class(TScriptObject, IAdapterHiddenFieldsStub);
```

C++

```
class TAdapterHiddenFieldsStub : public TScriptObject, public IAdapterHiddenFieldsStub;
```

File

AutoAdap

Description

This is class AutoAdap.TAdapterHiddenFieldsStub.

10.1.9.1 TAdapterHiddenFieldsStub Members

The following tables list the members exposed by TAdapterHiddenFieldsStub.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	WriteFields (see page 1402)	This is WriteFields, a member of class TAdapterHiddenFieldsStub.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1402)	This is Create, a member of class TAdapterHiddenFieldsStub.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	HiddenFields (see page 1402)	This is HiddenFields, a member of class TAdapterHiddenFieldsStub.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

10.1.9.2 TAdapterHiddenFieldsStub.Create Constructor

Pascal

```
constructor Create(AHiddenFields: TObject);
```

C++

```
__fastcall TAdapterHiddenFieldsStub(TObject * AHiddenFields);
```

Description

This is Create, a member of class TAdapterHiddenFieldsStub.

10.1.9.3 TAdapterHiddenFieldsStub Methods

The methods of the TAdapterHiddenFieldsStub class are listed here.

Protected Methods

	Name	Description
	WriteFields (see page 1402)	This is WriteFields, a member of class TAdapterHiddenFieldsStub.

10.1.9.3.1 TAdapterHiddenFieldsStub.WriteFields Method

Pascal

```
procedure WriteFields(const Response: IInterface);
```

C++

```
__fastcall WriteFields(const IInterface Response);
```

Description

This is WriteFields, a member of class TAdapterHiddenFieldsStub.

10.1.9.4 TAdapterHiddenFieldsStub Properties

The properties of the TAdapterHiddenFieldsStub class are listed here.

Public Properties

	Name	Description
R	HiddenFields (see page 1402)	This is HiddenFields, a member of class TAdapterHiddenFieldsStub.

10.1.9.4.1 TAdapterHiddenFieldsStub.HiddenFields Property

Pascal

```
property HiddenFields: TObject;
```

C++

```
__property TObject * HiddenFields;
```

Description

This is HiddenFields, a member of class TAdapterHiddenFieldsStub.

10.1.10 TAdapterHiddenFieldStub Class

Class Hierarchy

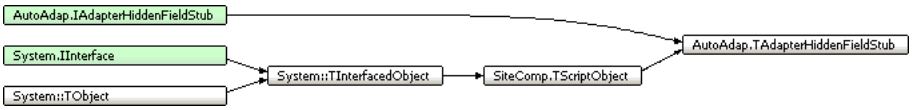

Pascal

```
TAdapterHiddenFieldStub = class(TScriptObject, IAdapterHiddenFieldStub);
```

C++

```
class TAdapterHiddenFieldStub : public TScriptObject, public IAdapterHiddenFieldStub;
```

File

AutoAdap

Description

This is class AutoAdap.TAdapterHiddenFieldStub.

10.1.10.1 TAdapterHiddenFieldStub Members

The following tables list the members exposed by TAdapterHiddenFieldStub.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	Get_Name (see page 1405)	This is Get_Name, a member of class TAdapterHiddenFieldStub.
	Get_Value (see page 1405)	This is Get_Value, a member of class TAdapterHiddenFieldStub.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.

	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1404)	This is Create, a member of class TAdapterHiddenFieldStub.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Name (see page 1452)	This is Name, a member of class IAdapterHiddenFieldStub.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	Value (see page 1453)	This is Value, a member of class IAdapterHiddenFieldStub.

10.1.10.2 TAdapterHiddenFieldStub.Create Constructor

Pascal

```
constructor Create(const AParam: IAdapterHiddenField);
```

C++

```
__fastcall TAdapterHiddenFieldStub(const IAdapterHiddenField AParam);
```

Description

This is Create, a member of class TAdapterHiddenFieldStub.

10.1.10.3 TAdapterHiddenFieldStub Methods

The methods of the TAdapterHiddenFieldStub class are listed here.

Protected Methods

	Name	Description
	Get_Name (see page 1405)	This is Get_Name, a member of class TAdapterHiddenFieldStub.
	Get_Value (see page 1405)	This is Get_Value, a member of class TAdapterHiddenFieldStub.

10.1.10.3.1 TAdapterHiddenFieldStub.Get_Name Method

Pascal

```
function Get_Name: string;
```

C++

```
__fastcall AnsiString Get_Name();
```

Description

This is Get_Name, a member of class TAdapterHiddenFieldStub.

10.1.10.3.2 TAdapterHiddenFieldStub.Get_Value Method

Pascal

```
function Get_Value: string;
```

C++

```
__fastcall AnsiString Get_Value();
```

Description

This is Get_Value, a member of class TAdapterHiddenFieldStub.

10.1.11 TAdapterImageStub Class

Class Hierarchy

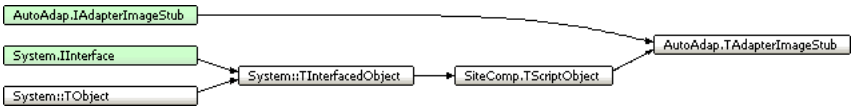

Pascal

```
TAdapterImageStub = class(TScriptObject, IAdapterImageStub);
```

C++

```
class TAdapterImageStub : public TScriptObject, public IAdapterImageStub;
```

File

AutoAdap

Description

This is class AutoAdap.TAdapterImageStub.

10.1.11.1 TAdapterImageStub Members

The following tables list the members exposed by TAdapterImageStub.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	Get_AsHREF (see page 1407)	This is Get_AsHREF, a member of class TAdapterImageStub.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1407)	This is Create, a member of class TAdapterImageStub.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	AsHREF (see page 1454)	This is AsHREF, a member of class IAdapterImageStub.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

10.1.11.2 TAdapterImageStub.Create Constructor

Pascal

```
constructor Create(AIntf: IInterface);
```

C++

```
__fastcall TAdapterImageStub(IInterface AIntf);
```

Description

This is Create, a member of class TAdapterImageStub.

10.1.11.3 TAdapterImageStub Methods

The methods of the TAdapterImageStub class are listed here.

Protected Methods

	Name	Description
	Get_AsHREF (see page 1407)	This is Get_AsHREF, a member of class TAdapterImageStub.

10.1.11.3.1 TAdapterImageStub.Get_AsHREF Method

Pascal

```
function Get_AsHREF: string;
```

C++

```
__fastcall AnsiString Get_AsHREF();
```

Description

This is Get_AsHREF, a member of class TAdapterImageStub.

10.1.12 TAdapterRecordsStub Class

Class Hierarchy

Pascal

```
TAdapterRecordsStub = class(TScriptObject);
```

C++

```
class TAdapterRecordsStub : public TScriptObject;
```

File

AutoAdap

Description

This is class AutoAdap.TAdapterRecordsStub.

10.1.12.1 TAdapterRecordsStub Members

The following tables list the members exposed by TAdapterRecordsStub.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1409)	This is Create, a member of class TAdapterRecordsStub.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Adapter (see page 1409)	This is Adapter, a member of class TAdapterRecordsStub.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

10.1.12.2 TAdapterRecordsStub.Create Constructor

Pascal

```
constructor Create(AAdapter: TComponent);
```

C++

```
__fastcall TAdapterRecordsStub(TComponent * AAdapter);
```

Description

This is Create, a member of class TAdapterRecordsStub.

10.1.12.3 TAdapterRecordsStub Properties

The properties of the TAdapterRecordsStub class are listed here.

Public Properties

	Name	Description
R	Adapter (see page 1409)	This is Adapter, a member of class TAdapterRecordsStub.

10.1.12.3.1 TAdapterRecordsStub.Adapter Property

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

This is Adapter, a member of class TAdapterRecordsStub.

10.1.13 TAdapterRecordStub Class

Class Hierarchy

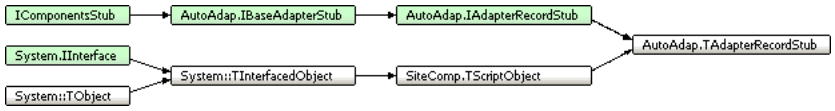

Pascal

```
TAdapterRecordStub = class(TScriptObject, IAdapterRecordStub);
```

C++

```
class TAdapterRecordStub : public TScriptObject, public IAdapterRecordStub;
```

File

AutoAdap

Description

This is class AutoAdap.TAdapterRecordStub.

10.1.13.1 TAdapterRecordStub Members

The following tables list the members exposed by TAdapterRecordStub.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1411)	This is Create, a member of class TAdapterRecordStub.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	Get_Actions (see page 1467)	This is Get_Actions, a member of class IBaseAdapterStub.
	Get_ClassName_ (see page 1467)	This is Get_ClassName_, a member of class IBaseAdapterStub.
	Get_Fields (see page 1467)	This is Get_Fields, a member of class IBaseAdapterStub.

	Get_Name_ (see page 1467)	This is Get_Name_, a member of class IBaseAdapterStub.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Actions (see page 1468)	This is Actions, a member of class IBaseAdapterStub.
R	Fields (see page 1468)	This is Fields, a member of class IBaseAdapterStub.
R	Impl (see page 1411)	This is Impl, a member of class TAdapterRecordStub.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

10.1.13.2 TAdapterRecordStub.Create Constructor

Pascal

```
constructor Create(AAdapter: TComponent; AAdapterStubClass: TAdapterStubClass);
```

C++

```
__fastcall TAdapterRecordStub(TComponent * AAdapter, TAdapterStubClass AAdapterStubClass);
```

Description

This is Create, a member of class TAdapterRecordStub.

10.1.13.3 TAdapterRecordStub Properties

The properties of the TAdapterRecordStub class are listed here.

Public Properties

	Name	Description
R	Impl (see page 1411)	This is Impl, a member of class TAdapterRecordStub.

10.1.13.3.1 TAdapterRecordStub.Impl Property

Pascal

```
property Impl: TBaseAdapterStub;
```

C++

```
__property TBaseAdapterStub Impl;
```

Description

This is Impl, a member of class TAdapterRecordStub.

10.1.14 TAdapterStub Class

Class Hierarchy

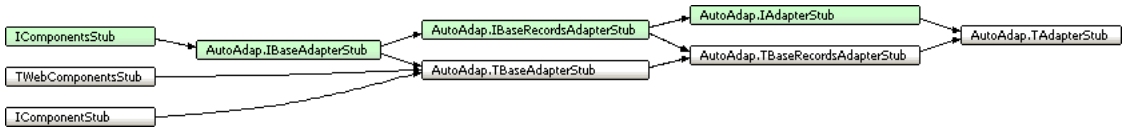

Pascal

```
TAdapterStub = class(TBaseRecordsAdapterStub, IAdapterStub);
```

C++

```
class TAdapterStub : public TBaseRecordsAdapterStub, public IAdapterStub;
```

File

AutoAdap

Description

This is class AutoAdap.TAdapterStub.

10.1.14.1 TAdapterStub Members

The following tables list the members exposed by TAdapterStub.

Protected Fields

	Name	Description
	FActions (see page 1423)	This is FActions, a member of class TBaseAdapterStub.
	FFields (see page 1424)	This is FFields, a member of class TBaseAdapterStub.

Protected Methods

	Name	Description
A	CreateComponentsStub (see page 1424)	This is CreateComponentsStub, a member of class TBaseAdapterStub.
A	CreateErrorsStub (see page 1413)	This is CreateErrorsStub, a member of class TAdapterStub.
A	CreateHiddenFieldsStub (see page 1414)	This is CreateHiddenFieldsStub, a member of class TAdapterStub.
A	CreateRecordsStub (see page 1427)	This is CreateRecordsStub, a member of class TBaseRecordsAdapterStub.
V	FindObject (see page 1414)	This is FindObject, a member of class TAdapterStub.
	Get_Actions (see page 1424)	This is Get_Actions, a member of class TBaseAdapterStub.
	Get_CanModify (see page 1414)	This is Get_CanModify, a member of class TAdapterStub.
	Get_CanView (see page 1414)	This is Get_CanView, a member of class TAdapterStub.
	Get_ClassName_ (see page 1425)	This is Get_ClassName_, a member of class TBaseAdapterStub.
	Get_Errors (see page 1414)	This is Get_Errors, a member of class TAdapterStub.
	Get_Fields (see page 1425)	This is Get_Fields, a member of class TBaseAdapterStub.
	Get_HiddenFields (see page 1415)	This is Get_HiddenFields, a member of class TAdapterStub.
	Get_HiddenRecordFields (see page 1415)	This is Get_HiddenRecordFields, a member of class TAdapterStub.
	Get_Mode (see page 1415)	This is Get_Mode, a member of class TAdapterStub.
	Get_Name_ (see page 1425)	This is Get_Name_, a member of class TBaseAdapterStub.
	Get_Records (see page 1427)	This is Get_Records, a member of class TBaseRecordsAdapterStub.
	Set_Mode (see page 1415)	This is Set_Mode, a member of class TAdapterStub.

Public Methods

	Name	Description
	Get_Actions (see page 1467)	This is Get_Actions, a member of class IBaseAdapterStub.
	Get_ClassName_ (see page 1467)	This is Get_ClassName_, a member of class IBaseAdapterStub.
	Get_Fields (see page 1467)	This is Get_Fields, a member of class IBaseAdapterStub.
	Get_Name_ (see page 1467)	This is Get_Name_, a member of class IBaseAdapterStub.
	Get_Records (see page 1469)	This is Get_Records, a member of class IBaseRecordsAdapterStub.

Public Properties

	Name	Description
R	Actions (see page 1468)	This is Actions, a member of class IBaseAdapterStub.
R	CanModify (see page 1458)	This is CanModify, a member of class IAdapterStub.
R	CanView (see page 1458)	This is CanView, a member of class IAdapterStub.
R	Errors (see page 1458)	This is Errors, a member of class IAdapterStub.
R	Fields (see page 1468)	This is Fields, a member of class IBaseAdapterStub.
R	HiddenFields (see page 1458)	This is HiddenFields, a member of class IAdapterStub.
R	HiddenRecordFields (see page 1459)	This is HiddenRecordFields, a member of class IAdapterStub.
	Mode (see page 1459)	This is Mode, a member of class IAdapterStub.
R	Records (see page 1469)	This is Records, a member of class IBaseRecordsAdapterStub.

10.1.14.2 TAdapterStub Methods

The methods of the TAdapterStub class are listed here.

Protected Methods

	Name	Description
A	CreateErrorsStub (see page 1413)	This is CreateErrorsStub, a member of class TAdapterStub.
A	CreateHiddenFieldsStub (see page 1414)	This is CreateHiddenFieldsStub, a member of class TAdapterStub.
V	FindObject (see page 1414)	This is FindObject, a member of class TAdapterStub.
	Get_CanModify (see page 1414)	This is Get_CanModify, a member of class TAdapterStub.
	Get_CanView (see page 1414)	This is Get_CanView, a member of class TAdapterStub.
	Get_Errors (see page 1414)	This is Get_Errors, a member of class TAdapterStub.
	Get_HiddenFields (see page 1415)	This is Get_HiddenFields, a member of class TAdapterStub.
	Get_HiddenRecordFields (see page 1415)	This is Get_HiddenRecordFields, a member of class TAdapterStub.
	Get_Mode (see page 1415)	This is Get_Mode, a member of class TAdapterStub.
	Set_Mode (see page 1415)	This is Set_Mode, a member of class TAdapterStub.

10.1.14.2.1 TAdapterStub.CreateErrorsStub Method

Pascal

```
function CreateErrorsStub(O: TObject): IInterface; virtual; abstract;
```

C++

```
virtual __fastcall IInterface CreateErrorsStub(TObject * O) = 0;
```

Description

This is CreateErrorsStub, a member of class TAdapterStub.

10.1.14.2.2 TAdapterStub.CreateHiddenFieldsStub Method

Pascal

```
function CreateHiddenFieldsStub(O: TObject): IInterface; virtual; abstract;
```

C++

```
virtual __fastcall IInterface CreateHiddenFieldsStub(TObject * O) = 0;
```

Description

This is CreateHiddenFieldsStub, a member of class TAdapterStub.

10.1.14.2.3 TAdapterStub.FindObject Method

Pascal

```
function FindObject(const AName: string): TObject; override;
```

C++

```
virtual __fastcall TObject * FindObject(const AnsiString AName);
```

Description

This is FindObject, a member of class TAdapterStub.

10.1.14.2.4 TAdapterStub.Get_CanModify Method

Pascal

```
function Get_CanModify: Boolean;
```

C++

```
__fastcall Boolean Get_CanModify();
```

Description

This is Get_CanModify, a member of class TAdapterStub.

10.1.14.2.5 TAdapterStub.Get_CanView Method

Pascal

```
function Get_CanView: Boolean;
```

C++

```
__fastcall Boolean Get_CanView();
```

Description

This is Get_CanView, a member of class TAdapterStub.

10.1.14.2.6 TAdapterStub.Get_Errors Method

Pascal

```
function Get_Errors: IInterface;
```

C++

```
__fastcall IInterface Get_Errors();
```

Description

This is Get_Errors, a member of class TAdapterStub.

10.1.14.2.7 TAdapterStub.Get_HiddenFields Method

Pascal

```
function Get_HiddenFields: IInterface;
```

C++

```
__fastcall IInterface Get_HiddenFields();
```

Description

This is Get_HiddenFields, a member of class TAdapterStub.

10.1.14.2.8 TAdapterStub.Get_HiddenRecordFields Method

Pascal

```
function Get_HiddenRecordFields: IInterface;
```

C++

```
__fastcall IInterface Get_HiddenRecordFields();
```

Description

This is Get_HiddenRecordFields, a member of class TAdapterStub.

10.1.14.2.9 TAdapterStub.Get_Mode Method

Pascal

```
function Get_Mode: string;
```

C++

```
__fastcall AnsiString Get_Mode();
```

Description

This is Get_Mode, a member of class TAdapterStub.

10.1.14.2.10 TAdapterStub.Set_Mode Method

Pascal

```
procedure Set_Mode(const Value: string);
```

C++

```
__fastcall Set_Mode(const AnsiString Value);
```

Description

This is Set_Mode, a member of class TAdapterStub.

10.1.15 TAdapterSubComponentStub Class

Class Hierarchy

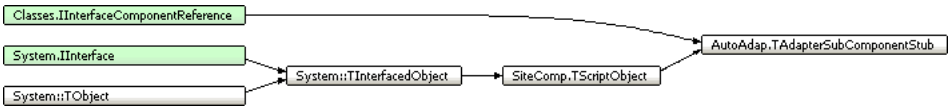

Pascal

```
TAdapterSubComponentStub = class(TScriptObject, IInterfaceComponentReference);
```

C++

```
class TAdapterSubComponentStub : public TScriptObject, public IInterfaceComponentReference;
```

File

AutoAdap

Description

This is class AutoAdap.TAdapterSubComponentStub.

10.1.15.1 TAdapterSubComponentStub Members

The following tables list the members exposed by TAdapterSubComponentStub.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	Get_Name (see page 1418)	This is Get_Name, a member of class TAdapterSubComponentStub.
	GetComponent (see page 1418)	Returns a reference to the component associated by the implementing class.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.

	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1417)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ObjectIntf (see page 1419)	This is ObjectIntf, a member of class TAdapterSubComponentStub.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

10.1.15.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TAdapterSubComponentStub.Create (IInterfaceComponentReference) (see page 1417)	This is Create, a member of class TAdapterSubComponentStub.
	TAdapterSubComponentStub.Create (TComponent) (see page 1418)	This is Create, a member of class TAdapterSubComponentStub.

10.1.15.2.1 TAdapterSubComponentStub.Create Constructor (IInterfaceComponentReference)

Pascal

```
constructor Create(AObject: IInterfaceComponentReference); overload;
```

C++

```
__fastcall TAdapterSubComponentStub(IInterfaceComponentReference AObject);
```

Description

This is Create, a member of class TAdapterSubComponentStub.

10.1.15.2.2 TAdapterSubComponentStub.Create Constructor (TComponent)

Pascal

```
constructor Create(AObject: TComponent); overload;
```

C++

```
__fastcall TAdapterSubComponentStub(TComponent * AObject);
```

Description

This is Create, a member of class TAdapterSubComponentStub.

10.1.15.3 TAdapterSubComponentStub Methods

The methods of the TAdapterSubComponentStub class are listed here.

Protected Methods

	Name	Description
	Get_Name (see page 1418)	This is Get_Name, a member of class TAdapterSubComponentStub.
	GetComponent (see page 1418)	Returns a reference to the component associated by the implementing class.

10.1.15.3.1 TAdapterSubComponentStub.Get_Name Method

Pascal

```
function Get_Name: Variant;
```

C++

```
__fastcall Variant Get_Name();
```

Description

This is Get_Name, a member of class TAdapterSubComponentStub.

10.1.15.3.2 TAdapterSubComponentStub.GetComponent Method

Returns a reference to the component associated by the implementing class.

Pascal

```
function GetComponent: TComponent;
```

C++

```
__fastcall TComponent * GetComponent();
```

Description

GetComponent returns a TComponent value that is associated with the implementing class. For example, the TComponent class itself implements the GetComponent to return a pointer to itself.

See Also

TComponent (see page 2762)

10.1.15.4 TAdapterSubComponentStub Properties

The properties of the TAdapterSubComponentStub class are listed here.

Public Properties

	Name	Description
R	ObjectIntf (see page 1419)	This is ObjectIntf, a member of class TAdapterSubComponentStub.

10.1.15.4.1 TAdapterSubComponentStub.ObjectIntf Property

Pascal

```
property ObjectIntf: IInterfaceComponentReference;
```

C++

```
__property IInterfaceComponentReference ObjectIntf;
```

Description

This is ObjectIntf, a member of class TAdapterSubComponentStub.

10.1.16 TAdapterValuesListStub Class

Class Hierarchy

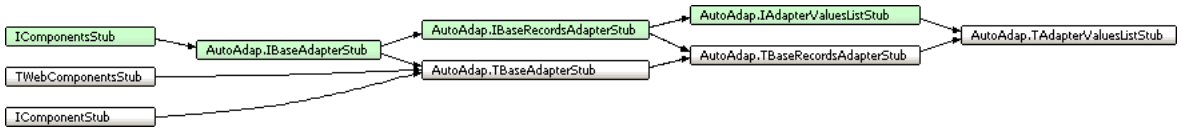

Pascal

```
TAdapterValuesListStub = class(TBaseRecordsAdapterStub, IAdapterValuesListStub);
```

C++

```
class TAdapterValuesListStub : public TBaseRecordsAdapterStub, public IAdapterValuesListStub;
```

File

AutoAdap

Description

This is class AutoAdap.TAdapterValuesListStub.

10.1.16.1 TAdapterValuesListStub Members

The following tables list the members exposed by TAdapterValuesListStub.

Protected Fields

	Name	Description
	FActions (see page 1423)	This is FActions, a member of class TBaseAdapterStub.
	FFields (see page 1424)	This is FFields, a member of class TBaseAdapterStub.

Protected Methods

	Name	Description
A	CreateComponentsStub (see page 1424)	This is CreateComponentsStub, a member of class TBaseAdapterStub.
A	CreateImageStub (see page 1421)	This is CreateImageStub, a member of class TAdapterValuesListStub.
A	CreateRecordsStub (see page 1427)	This is CreateRecordsStub, a member of class TBaseRecordsAdapterStub.
V	FindObject (see page 1421)	This is FindObject, a member of class TAdapterValuesListStub.
	Get_Actions (see page 1424)	This is Get_Actions, a member of class TBaseAdapterStub.
	Get_ClassName_ (see page 1425)	This is Get_ClassName_, a member of class TBaseAdapterStub.
	Get_Fields (see page 1425)	This is Get_Fields, a member of class TBaseAdapterStub.
	Get_Image (see page 1421)	This is Get_Image, a member of class TAdapterValuesListStub.
	Get_Name_ (see page 1425)	This is Get_Name_, a member of class TBaseAdapterStub.
	Get_Records (see page 1421)	This is Get_Records, a member of class TAdapterValuesListStub.
	Get_Value (see page 1421)	This is Get_Value, a member of class TAdapterValuesListStub.
	Get_ValueName (see page 1422)	This is Get_ValueName, a member of class TAdapterValuesListStub.
	ImageOfValue (see page 1422)	This is ImageOfValue, a member of class TAdapterValuesListStub.
	NameOfValue (see page 1422)	This is NameOfValue, a member of class TAdapterValuesListStub.

Public Methods

	Name	Description
	Get_Actions (see page 1467)	This is Get_Actions, a member of class IBaseAdapterStub.
	Get_ClassName_ (see page 1467)	This is Get_ClassName_, a member of class IBaseAdapterStub.
	Get_Fields (see page 1467)	This is Get_Fields, a member of class IBaseAdapterStub.
	Get_Name_ (see page 1467)	This is Get_Name_, a member of class IBaseAdapterStub.

Public Properties

	Name	Description
R	Actions (see page 1468)	This is Actions, a member of class IBaseAdapterStub.
R	Fields (see page 1468)	This is Fields, a member of class IBaseAdapterStub.
R	Image (see page 1461)	This is Image, a member of class IAdapterValuesListStub.
R	Records (see page 1469)	This is Records, a member of class IBaseRecordsAdapterStub.
R	Value (see page 1462)	This is Value, a member of class IAdapterValuesListStub.
R	ValueName (see page 1462)	This is ValueName, a member of class IAdapterValuesListStub.

10.1.16.2 TAdapterValuesListStub Methods

The methods of the TAdapterValuesListStub class are listed here.

Protected Methods

	Name	Description
A	CreateImageStub (see page 1421)	This is CreateImageStub, a member of class TAdapterValuesListStub.
V	FindObject (see page 1421)	This is FindObject, a member of class TAdapterValuesListStub.
	Get_Image (see page 1421)	This is Get_Image, a member of class TAdapterValuesListStub.
	Get_Records (see page 1421)	This is Get_Records, a member of class TAdapterValuesListStub.
	Get_Value (see page 1421)	This is Get_Value, a member of class TAdapterValuesListStub.
	Get_ValueName (see page 1422)	This is Get_ValueName, a member of class TAdapterValuesListStub.
	ImageOfValue (see page 1422)	This is ImageOfValue, a member of class TAdapterValuesListStub.
	NameOfValue (see page 1422)	This is NameOfValue, a member of class TAdapterValuesListStub.

10.1.16.2.1 TAdapterValuesListStub.CreateImageStub Method

Pascal

```
function CreateImageStub(AdapterImage: IInterface): IInterface; virtual; abstract;
```

C++

```
virtual __fastcall IInterface CreateImageStub(IInterface AdapterImage) = 0;
```

Description

This is CreateImageStub, a member of class TAdapterValuesListStub.

10.1.16.2.2 TAdapterValuesListStub.FindObject Method

Pascal

```
function FindObject(const AName: string): TObject; override;
```

C++

```
virtual __fastcall TObject * FindObject(const AnsiString AName);
```

Description

This is FindObject, a member of class TAdapterValuesListStub.

10.1.16.2.3 TAdapterValuesListStub.Get_Image Method

Pascal

```
function Get_Image: IInterface;
```

C++

```
__fastcall IInterface Get_Image();
```

Description

This is Get_Image, a member of class TAdapterValuesListStub.

10.1.16.2.4 TAdapterValuesListStub.Get_Records Method

Pascal

```
function Get_Records: IInterface;
```

C++

```
__fastcall IInterface Get_Records();
```

Description

This is Get_Records, a member of class TAdapterValuesListStub.

10.1.16.2.5 TAdapterValuesListStub.Get_Value Method

Pascal

```
function Get_Value: Variant;
```

C++

```
__fastcall Variant Get_Value();
```

Description

This is Get_Value, a member of class TAdapterValuesListStub.

10.1.16.2.6 TAdapterValuesListStub.Get_ValueName Method

Pascal

```
function Get_ValueName: string;
```

C++

```
__fastcall AnsiString Get_ValueName();
```

Description

This is Get_ValueName, a member of class TAdapterValuesListStub.

10.1.16.2.7 TAdapterValuesListStub.ImageOfValue Method

Pascal

```
function ImageOfValue(Value: Variant): IInterface;
```

C++

```
__fastcall IInterface ImageOfValue(Variant Value);
```

Description

This is ImageOfValue, a member of class TAdapterValuesListStub.

10.1.16.2.8 TAdapterValuesListStub.NameOfValue Method

Pascal

```
function NameOfValue(Value: Variant): string;
```

C++

```
__fastcall AnsiString NameOfValue(Variant Value);
```

Description

This is NameOfValue, a member of class TAdapterValuesListStub.

10.1.17 TBaseAdapterStub Class

Class Hierarchy

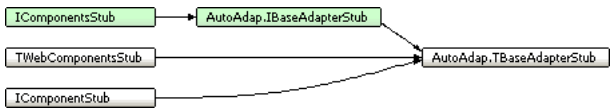

Pascal

```
TBaseAdapterStub = class(TWebComponentsStub, IComponentStub, IBaseAdapterStub);
```

C++

```
class TBaseAdapterStub : public TWebComponentsStub, public IComponentStub, public
```

```
IBaseAdapterStub;
```

File

AutoAdap

Description

This is class AutoAdap.TBaseAdapterStub.

10.1.17.1 TBaseAdapterStub Members

The following tables list the members exposed by TBaseAdapterStub.

Protected Fields

	Name	Description
	FActions (see page 1423)	This is FActions, a member of class TBaseAdapterStub.
	FFields (see page 1424)	This is FFields, a member of class TBaseAdapterStub.

Protected Methods

	Name	Description
A	CreateComponentsStub (see page 1424)	This is CreateComponentsStub, a member of class TBaseAdapterStub.
V	FindObject (see page 1424)	This is FindObject, a member of class TBaseAdapterStub.
	Get_Actions (see page 1424)	This is Get_Actions, a member of class TBaseAdapterStub.
	Get_ClassName_ (see page 1425)	This is Get_ClassName_, a member of class TBaseAdapterStub.
	Get_Fields (see page 1425)	This is Get_Fields, a member of class TBaseAdapterStub.
	Get_Name_ (see page 1425)	This is Get_Name_, a member of class TBaseAdapterStub.

Public Properties

	Name	Description
R	Actions (see page 1468)	This is Actions, a member of class IBaseAdapterStub.
R	Fields (see page 1468)	This is Fields, a member of class IBaseAdapterStub.

10.1.17.2 TBaseAdapterStub Fields

The fields of the TBaseAdapterStub class are listed here.

Protected Fields

	Name	Description
	FActions (see page 1423)	This is FActions, a member of class TBaseAdapterStub.
	FFields (see page 1424)	This is FFields, a member of class TBaseAdapterStub.

10.1.17.2.1 TBaseAdapterStub.FActions Field

Pascal

```
FActions: IInterface;
```

C++

```
IInterface FActions;
```

Description

This is FActions, a member of class TBaseAdapterStub.

10.1.17.2.2 TBaseAdapterStub.FFields Field

Pascal

```
FFields: IInterface;
```

C++

```
IInterface FFields;
```

Description

This is FFields, a member of class TBaseAdapterStub.

10.1.17.3 TBaseAdapterStub Methods

The methods of the TBaseAdapterStub class are listed here.

Protected Methods

	Name	Description
	CreateComponentsStub (see page 1424)	This is CreateComponentsStub, a member of class TBaseAdapterStub.
	FindObject (see page 1424)	This is FindObject, a member of class TBaseAdapterStub.
	Get_Actions (see page 1424)	This is Get_Actions, a member of class TBaseAdapterStub.
	Get_ClassName_ (see page 1425)	This is Get_ClassName_, a member of class TBaseAdapterStub.
	Get_Fields (see page 1425)	This is Get_Fields, a member of class TBaseAdapterStub.
	Get_Name_ (see page 1425)	This is Get_Name_, a member of class TBaseAdapterStub.

10.1.17.3.1 TBaseAdapterStub.CreateComponentsStub Method

Pascal

```
function CreateComponentsStub(O: TComponent): IInterface; virtual; abstract;
```

C++

```
virtual __fastcall IInterface CreateComponentsStub(TComponent * O) = 0;
```

Description

This is CreateComponentsStub, a member of class TBaseAdapterStub.

10.1.17.3.2 TBaseAdapterStub.FindObject Method

Pascal

```
function FindObject(const AName: string): TObject; override;
```

C++

```
virtual __fastcall TObject * FindObject(const AnsiString AName);
```

Description

This is FindObject, a member of class TBaseAdapterStub.

10.1.17.3.3 TBaseAdapterStub.Get_Actions Method

Pascal

```
function Get_Actions: IInterface;
```

C++

```
__fastcall IInterface Get_Actions();
```

Description

This is Get_Actions, a member of class TBaseAdapterStub.

10.1.17.3.4 TBaseAdapterStub.Get_ClassName_ Method

Pascal

```
function Get_ClassName_: string;
```

C++

```
__fastcall AnsiString Get_ClassName_();
```

Description

This is Get_ClassName_, a member of class TBaseAdapterStub.

10.1.17.3.5 TBaseAdapterStub.Get_Fields Method

Pascal

```
function Get_Fields: IInterface;
```

C++

```
__fastcall IInterface Get_Fields();
```

Description

This is Get_Fields, a member of class TBaseAdapterStub.

10.1.17.3.6 TBaseAdapterStub.Get_Name_ Method

Pascal

```
function Get_Name_: string;
```

C++

```
__fastcall AnsiString Get_Name_();
```

Description

This is Get_Name_, a member of class TBaseAdapterStub.

10.1.18 TBaseRecordsAdapterStub Class

Class Hierarchy

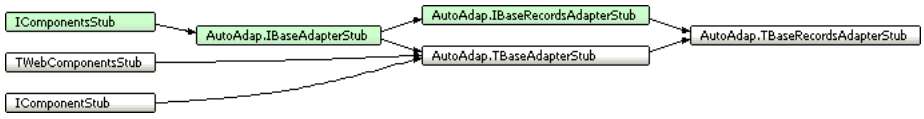

Pascal

```
TBaseRecordsAdapterStub = class(TBaseAdapterStub, IBaseRecordsAdapterStub);
```

C++

```
class TBaseRecordsAdapterStub : public TBaseAdapterStub, public IBaseRecordsAdapterStub;
```

File

AutoAdap

Description

This is class AutoAdap.TBaseRecordsAdapterStub.

10.1.18.1 TBaseRecordsAdapterStub Members

The following tables list the members exposed by TBaseRecordsAdapterStub.

Protected Fields

	Name	Description
	FActions (see page 1423)	This is FActions, a member of class TBaseAdapterStub.
	FFields (see page 1424)	This is FFields, a member of class TBaseAdapterStub.

Protected Methods

	Name	Description
A	CreateComponentsStub (see page 1424)	This is CreateComponentsStub, a member of class TBaseAdapterStub.
A	CreateRecordsStub (see page 1427)	This is CreateRecordsStub, a member of class TBaseRecordsAdapterStub.
V	FindObject (see page 1424)	This is FindObject, a member of class TBaseAdapterStub.
	Get_Actions (see page 1424)	This is Get_Actions, a member of class TBaseAdapterStub.
	Get_ClassName_ (see page 1425)	This is Get_ClassName_, a member of class TBaseAdapterStub.
	Get_Fields (see page 1425)	This is Get_Fields, a member of class TBaseAdapterStub.
	Get_Name_ (see page 1425)	This is Get_Name_, a member of class TBaseAdapterStub.
	Get_Records (see page 1427)	This is Get_Records, a member of class TBaseRecordsAdapterStub.

Public Methods

	Name	Description
	Get_Actions (see page 1467)	This is Get_Actions, a member of class IBaseAdapterStub.
	Get_ClassName_ (see page 1467)	This is Get_ClassName_, a member of class IBaseAdapterStub.
	Get_Fields (see page 1467)	This is Get_Fields, a member of class IBaseAdapterStub.
	Get_Name_ (see page 1467)	This is Get_Name_, a member of class IBaseAdapterStub.

Public Properties

	Name	Description
R	Actions (see page 1468)	This is Actions, a member of class IBaseAdapterStub.
R	Fields (see page 1468)	This is Fields, a member of class IBaseAdapterStub.
R	Records (see page 1469)	This is Records, a member of class IBaseRecordsAdapterStub.

10.1.18.2 TBaseRecordsAdapterStub Methods

The methods of the TBaseRecordsAdapterStub class are listed here.

Protected Methods

	Name	Description
A	CreateRecordsStub (see page 1427)	This is CreateRecordsStub, a member of class TBaseRecordsAdapterStub.
	Get_Records (see page 1427)	This is Get_Records, a member of class TBaseRecordsAdapterStub.

10.1.18.2.1 TBaseRecordsAdapterStub.CreateRecordsStub Method

Pascal

```
function CreateRecordsStub(O: TComponent): IInterface; virtual; abstract;
```

C++

```
virtual __fastcall IInterface CreateRecordsStub(TComponent * O) = 0;
```

Description

This is CreateRecordsStub, a member of class TBaseRecordsAdapterStub.

10.1.18.2.2 TBaseRecordsAdapterStub.Get_Records Method

Pascal

```
function Get_Records: IInterface;
```

C++

```
__fastcall IInterface Get_Records();
```

Description

This is Get_Records, a member of class TBaseRecordsAdapterStub.

10.1.19 TWebApplicationStub Class

Class Hierarchy

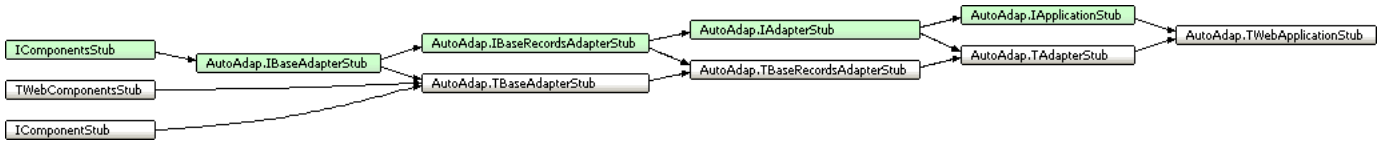

Pascal

```
TWebApplicationStub = class(TAdapterStub, IApplicationStub);
```

C++

```
class TWebApplicationStub : public TAdapterStub, public IApplicationStub;
```

File

AutoAdap

Description

This is class AutoAdap.TWebApplicationStub.

10.1.19.1 TWebApplicationStub Members

The following tables list the members exposed by TWebApplicationStub.

Protected Fields

	Name	Description
	FActions (see page 1423)	This is FActions, a member of class TBaseAdapterStub.
	FFields (see page 1424)	This is FFields, a member of class TBaseAdapterStub.

Protected Methods

	Name	Description
A	CreateComponentsStub (see page 1424)	This is CreateComponentsStub, a member of class TBaseAdapterStub.
A	CreateErrorsStub (see page 1413)	This is CreateErrorsStub, a member of class TAdapterStub.
A	CreateHiddenFieldsStub (see page 1414)	This is CreateHiddenFieldsStub, a member of class TAdapterStub.
A	CreateRecordsStub (see page 1427)	This is CreateRecordsStub, a member of class TBaseRecordsAdapterStub.
V	FindObject (see page 1414)	This is FindObject, a member of class TAdapterStub.
V	FindObject (see page 1424)	This is FindObject, a member of class TBaseAdapterStub.
	Get_Actions (see page 1424)	This is Get_Actions, a member of class TBaseAdapterStub.
	Get_CanModify (see page 1414)	This is Get_CanModify, a member of class TAdapterStub.
	Get_CanView (see page 1414)	This is Get_CanView, a member of class TAdapterStub.
	Get_ClassName_ (see page 1425)	This is Get_ClassName_, a member of class TBaseAdapterStub.
	Get_Designing (see page 1429)	This is Get_Designing, a member of class TWebApplicationStub.
	Get_Errors (see page 1414)	This is Get_Errors, a member of class TAdapterStub.
	Get_Fields (see page 1425)	This is Get_Fields, a member of class TBaseAdapterStub.
	Get_HiddenFields (see page 1415)	This is Get_HiddenFields, a member of class TAdapterStub.
	Get_HiddenRecordFields (see page 1415)	This is Get_HiddenRecordFields, a member of class TAdapterStub.
	Get_Mode (see page 1415)	This is Get_Mode, a member of class TAdapterStub.
	Get_ModuleFileName (see page 1429)	This is Get_ModuleFileName, a member of class TWebApplicationStub.
	Get_ModulePath (see page 1430)	This is Get_ModulePath, a member of class TWebApplicationStub.
	Get_Name_ (see page 1425)	This is Get_Name_, a member of class TBaseAdapterStub.
	Get_Records (see page 1427)	This is Get_Records, a member of class TBaseRecordsAdapterStub.
	Get_Title (see page 1430)	This is Get_Title, a member of class TWebApplicationStub.
	QualifyFileName (see page 1430)	This is QualifyFileName, a member of class TWebApplicationStub.
	Set_Mode (see page 1415)	This is Set_Mode, a member of class TAdapterStub.

Public Methods

	Name	Description
	Create (see page 1429)	This is Create, a member of class TWebApplicationStub.
	Get_Actions (see page 1467)	This is Get_Actions, a member of class IBaseAdapterStub.
	Get_CanModify (see page 1456)	This is Get_CanModify, a member of class IAdapterStub.
	Get_CanView (see page 1456)	This is Get_CanView, a member of class IAdapterStub.
	Get_ClassName_ (see page 1467)	This is Get_ClassName_, a member of class IBaseAdapterStub.
	Get_Errors (see page 1456)	This is Get_Errors, a member of class IAdapterStub.
	Get_Fields (see page 1467)	This is Get_Fields, a member of class IBaseAdapterStub.
	Get_HiddenFields (see page 1457)	This is Get_HiddenFields, a member of class IAdapterStub.
	Get_HiddenRecordFields (see page 1457)	This is Get_HiddenRecordFields, a member of class IAdapterStub.
	Get_Mode (see page 1457)	This is Get_Mode, a member of class IAdapterStub.
	Get_Name_ (see page 1467)	This is Get_Name_, a member of class IBaseAdapterStub.
	Get_Records (see page 1469)	This is Get_Records, a member of class IBaseRecordsAdapterStub.
	Set_Mode (see page 1457)	This is Set_Mode, a member of class IAdapterStub.

Public Properties

	Name	Description
R	Actions (see page 1468)	This is Actions, a member of class IBaseAdapterStub.
R	CanModify (see page 1458)	This is CanModify, a member of class IAdapterStub.
R	CanView (see page 1458)	This is CanView, a member of class IAdapterStub.
R	Designing (see page 1465)	This is Designing, a member of class IApplicationStub.
R	Errors (see page 1458)	This is Errors, a member of class IAdapterStub.

R	Fields (see page 1468)	This is Fields, a member of class IBaseAdapterStub.
R	HiddenFields (see page 1458)	This is HiddenFields, a member of class IAdapterStub.
R	HiddenRecordFields (see page 1459)	This is HiddenRecordFields, a member of class IAdapterStub.
	Mode (see page 1459)	This is Mode, a member of class IAdapterStub.
R	ModuleFileName (see page 1465)	This is ModuleFileName, a member of class IApplicationStub.
R	ModulePath (see page 1465)	This is ModulePath, a member of class IApplicationStub.
R	Records (see page 1469)	This is Records, a member of class IBaseRecordsAdapterStub.
R	Title (see page 1465)	This is Title, a member of class IApplicationStub.

10.1.19.2 TWebApplicationStub.Create Constructor

Pascal

```
constructor Create(AContext: IScriptContext); reintroduce;
```

C++

```
__fastcall TWebApplicationStub(IScriptContext AContext);
```

Description

This is Create, a member of class TWebApplicationStub.

10.1.19.3 TWebApplicationStub Methods

The methods of the TWebApplicationStub class are listed here.

Protected Methods

	Name	Description
	Get_Designing (see page 1429)	This is Get_Designing, a member of class TWebApplicationStub.
	Get_ModuleFileName (see page 1429)	This is Get_ModuleFileName, a member of class TWebApplicationStub.
	Get_ModulePath (see page 1430)	This is Get_ModulePath, a member of class TWebApplicationStub.
	Get_Title (see page 1430)	This is Get_Title, a member of class TWebApplicationStub.
	QualifyFileName (see page 1430)	This is QualifyFileName, a member of class TWebApplicationStub.

10.1.19.3.1 TWebApplicationStub.Get_Designing Method

Pascal

```
function Get_Designing: Boolean;
```

C++

```
__fastcall Boolean Get_Designing();
```

Description

This is Get_Designing, a member of class TWebApplicationStub.

10.1.19.3.2 TWebApplicationStub.Get_ModuleFileName Method

Pascal

```
function Get_ModuleFileName: string;
```

C++

```
__fastcall AnsiString Get_ModuleFileName();
```

Description

This is Get_ModuleFileName, a member of class TWebApplicationStub.

10.1.19.3.3 TWebApplicationStub.Get_ModulePath Method

Pascal

```
function Get_ModulePath: string;
```

C++

```
__fastcall AnsiString Get_ModulePath();
```

Description

This is Get_ModulePath, a member of class TWebApplicationStub.

10.1.19.3.4 TWebApplicationStub.Get_Title Method

Pascal

```
function Get_Title: string;
```

C++

```
__fastcall AnsiString Get_Title();
```

Description

This is Get_Title, a member of class TWebApplicationStub.

10.1.19.3.5 TWebApplicationStub.QualifyFileName Method

Pascal

```
function QualifyFileName(const FileName: PChar): string;
```

C++

```
__fastcall AnsiString QualifyFileName(const const char * FileName);
```

Description

This is QualifyFileName, a member of class TWebApplicationStub.

10.1.20 TWebEndUserStub Class

Class Hierarchy

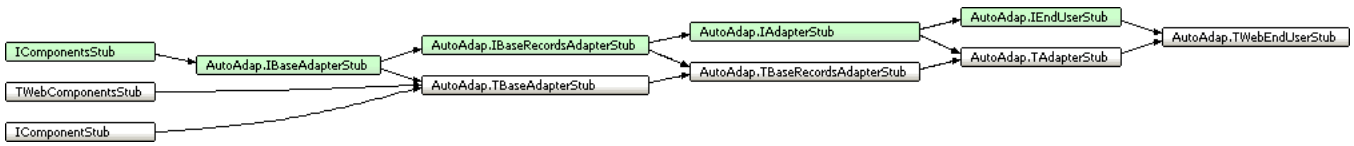

Pascal

```
TWebEndUserStub = class(TAdapterStub, IEndUserStub);
```

C++

```
class TWebEndUserStub : public TAdapterStub, public IEndUserStub;
```

File

AutoAdap

Description

This is class AutoAdap.TWebEndUserStub.

10.1.20.1 TWebEndUserStub Members

The following tables list the members exposed by TWebEndUserStub.

Protected Fields

	Name	Description
	FActions (see page 1423)	This is FActions, a member of class TBaseAdapterStub.
	FFields (see page 1424)	This is FFields, a member of class TBaseAdapterStub.

Protected Methods

	Name	Description
	CreateActionStub (see page 1432)	This is CreateActionStub, a member of class TWebEndUserStub.
	CreateComponentsStub (see page 1424)	This is CreateComponentsStub, a member of class TBaseAdapterStub.
	CreateErrorsStub (see page 1413)	This is CreateErrorsStub, a member of class TAdapterStub.
	CreateHiddenFieldsStub (see page 1414)	This is CreateHiddenFieldsStub, a member of class TAdapterStub.
	CreateRecordsStub (see page 1427)	This is CreateRecordsStub, a member of class TBaseRecordsAdapterStub.
	FindObject (see page 1414)	This is FindObject, a member of class TAdapterStub.
	FindObject (see page 1424)	This is FindObject, a member of class TBaseAdapterStub.
	Get_Actions (see page 1424)	This is Get_Actions, a member of class TBaseAdapterStub.
	Get_CanModify (see page 1414)	This is Get_CanModify, a member of class TAdapterStub.
	Get_CanView (see page 1414)	This is Get_CanView, a member of class TAdapterStub.
	Get_ClassName_ (see page 1425)	This is Get_ClassName_, a member of class TBaseAdapterStub.
	Get_DisplayName (see page 1433)	This is Get_DisplayName, a member of class TWebEndUserStub.
	Get_Errors (see page 1414)	This is Get_Errors, a member of class TAdapterStub.
	Get_Fields (see page 1425)	This is Get_Fields, a member of class TBaseAdapterStub.
	Get_HiddenFields (see page 1415)	This is Get_HiddenFields, a member of class TAdapterStub.
	Get_HiddenRecordFields (see page 1415)	This is Get_HiddenRecordFields, a member of class TAdapterStub.
	Get_LoggedIn (see page 1433)	This is Get_LoggedIn, a member of class TWebEndUserStub.
	Get_LoginFormAction (see page 1433)	This is Get_LoginFormAction, a member of class TWebEndUserStub.
	Get_LogoutAction (see page 1433)	This is Get_LogoutAction, a member of class TWebEndUserStub.
	Get_Mode (see page 1415)	This is Get_Mode, a member of class TAdapterStub.
	Get_Name_ (see page 1425)	This is Get_Name_, a member of class TBaseAdapterStub.
	Get_Records (see page 1427)	This is Get_Records, a member of class TBaseRecordsAdapterStub.
	Set_Mode (see page 1415)	This is Set_Mode, a member of class TAdapterStub.

Public Methods

	Name	Description
	Create (see page 1432)	This is Create, a member of class TWebEndUserStub.
	Get_Actions (see page 1467)	This is Get_Actions, a member of class IBaseAdapterStub.
	Get_CanModify (see page 1456)	This is Get_CanModify, a member of class IAdapterStub.
	Get_CanView (see page 1456)	This is Get_CanView, a member of class IAdapterStub.
	Get_ClassName_ (see page 1467)	This is Get_ClassName_, a member of class IBaseAdapterStub.
	Get_Errors (see page 1456)	This is Get_Errors, a member of class IAdapterStub.
	Get_Fields (see page 1467)	This is Get_Fields, a member of class IBaseAdapterStub.

	Get_HiddenFields (see page 1457)	This is Get_HiddenFields, a member of class IAdapterStub.
	Get_HiddenRecordFields (see page 1457)	This is Get_HiddenRecordFields, a member of class IAdapterStub.
	Get_Mode (see page 1457)	This is Get_Mode, a member of class IAdapterStub.
	Get_Name_ (see page 1467)	This is Get_Name_, a member of class IBaseAdapterStub.
	Get_Records (see page 1469)	This is Get_Records, a member of class IBaseRecordsAdapterStub.
	Set_Mode (see page 1457)	This is Set_Mode, a member of class IAdapterStub.

Public Properties

	Name	Description
R	Actions (see page 1468)	This is Actions, a member of class IBaseAdapterStub.
R	CanModify (see page 1458)	This is CanModify, a member of class IAdapterStub.
R	CanView (see page 1458)	This is CanView, a member of class IAdapterStub.
R	DisplayName (see page 1472)	This is DisplayName, a member of class IEndUserStub.
R	Errors (see page 1458)	This is Errors, a member of class IAdapterStub.
R	Fields (see page 1468)	This is Fields, a member of class IBaseAdapterStub.
R	HiddenFields (see page 1458)	This is HiddenFields, a member of class IAdapterStub.
R	HiddenRecordFields (see page 1459)	This is HiddenRecordFields, a member of class IAdapterStub.
R	LoggedIn (see page 1472)	This is LoggedIn, a member of class IEndUserStub.
R	LoginFormAction (see page 1472)	This is LoginFormAction, a member of class IEndUserStub.
R	LogoutAction (see page 1473)	This is LogoutAction, a member of class IEndUserStub.
	Mode (see page 1459)	This is Mode, a member of class IAdapterStub.
R	Records (see page 1469)	This is Records, a member of class IBaseRecordsAdapterStub.

10.1.20.2 TWebEndUserStub.Create Constructor

Pascal

```
constructor Create(AContext: IScriptContext); reintroduce;
```

C++

```
__fastcall TWebEndUserStub(IScriptContext AContext);
```

Description

This is Create, a member of class TWebEndUserStub.

10.1.20.3 TWebEndUserStub Methods

The methods of the TWebEndUserStub class are listed here.

Protected Methods

	Name	Description
A	CreateActionStub (see page 1432)	This is CreateActionStub, a member of class TWebEndUserStub.
	Get_DisplayName (see page 1433)	This is Get_DisplayName, a member of class TWebEndUserStub.
	Get_LoggedIn (see page 1433)	This is Get_LoggedIn, a member of class TWebEndUserStub.
	Get_LoginFormAction (see page 1433)	This is Get_LoginFormAction, a member of class TWebEndUserStub.
	Get_LogoutAction (see page 1433)	This is Get_LogoutAction, a member of class TWebEndUserStub.

10.1.20.3.1 TWebEndUserStub.CreateActionStub Method

Pascal

```
function CreateActionStub(O: TComponent): IInterface; virtual; abstract; overload;
```

C++

```
virtual __fastcall IInterface CreateActionStub(TComponent * O) = 0;
```

Description

This is CreateActionStub, a member of class TWebEndUserStub.

10.1.20.3.2 TWebEndUserStub.Get_DisplayName Method

Pascal

```
function Get_DisplayName: string;
```

C++

```
__fastcall AnsiString Get_DisplayName();
```

Description

This is Get_DisplayName, a member of class TWebEndUserStub.

10.1.20.3.3 TWebEndUserStub.Get_LoggedIn Method

Pascal

```
function Get_LoggedIn: Boolean;
```

C++

```
__fastcall Boolean Get_LoggedIn();
```

Description

This is Get_LoggedIn, a member of class TWebEndUserStub.

10.1.20.3.4 TWebEndUserStub.Get_LoginFormAction Method

Pascal

```
function Get_LoginFormAction: IInterface;
```

C++

```
__fastcall IInterface Get_LoginFormAction();
```

Description

This is Get_LoginFormAction, a member of class TWebEndUserStub.

10.1.20.3.5 TWebEndUserStub.Get_LogoutAction Method

Pascal

```
function Get_LogoutAction: IInterface;
```

C++

```
__fastcall IInterface Get_LogoutAction();
```

Description

This is Get_LogoutAction, a member of class TWebEndUserStub.

10.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAdapterErrorsStub (see page 1434)	This is class AutoAdap.IAdapterErrorsStub.
	IAdapterErrorStub (see page 1435)	This is class AutoAdap.IAdapterErrorStub.
	IAdapterFieldStub (see page 1438)	This is class AutoAdap.IAdapterFieldStub.
	IAdapterFieldValuesStub (see page 1448)	This is class AutoAdap.IAdapterFieldValuesStub.
	IAdapterHiddenFieldsStub (see page 1450)	This is class AutoAdap.IAdapterHiddenFieldsStub.
	IAdapterHiddenFieldStub (see page 1451)	This is class AutoAdap.IAdapterHiddenFieldStub.
	IAdapterImageStub (see page 1453)	This is class AutoAdap.IAdapterImageStub.
	IAdapterRecordStub (see page 1454)	This is class AutoAdap.IAdapterRecordStub.
	IAdapterStub (see page 1455)	This is class AutoAdap.IAdapterStub.
	IAdapterValuesListStub (see page 1459)	This is class AutoAdap.IAdapterValuesListStub.
	IApplicationStub (see page 1462)	This is class AutoAdap.IApplicationStub.
	IBaseAdapterStub (see page 1466)	This is class AutoAdap.IBaseAdapterStub.
	IBaseRecordsAdapterStub (see page 1468)	This is class AutoAdap.IBaseRecordsAdapterStub.
	IEndUserStub (see page 1470)	This is class AutoAdap.IEndUserStub.
	IImageProducerStub (see page 1473)	This is class AutoAdap.IImageProducerStub.

10.2.1 IAdapterErrorsStub Interface

Class Hierarchy

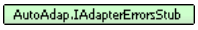

Pascal

```
IAdapterErrorsStub = interface;
```

C++

```
__interface IAdapterErrorsStub;
```

File

AutoAdap

Description

This is class AutoAdap.IAdapterErrorsStub.

10.2.1.1 IAdapterErrorsStub Members

The following tables list the members exposed by IAdapterErrorsStub.

Public Methods

	Name	Description
	DefineLabel (see page 1435)	This is DefineLabel, a member of class IAdapterErrorsStub.

	FieldErrors (see page 1435)	This is FieldErrors, a member of class IAdapterErrorsStub.

10.2.1.2 IAdapterErrorsStub Methods

The methods of the IAdapterErrorsStub class are listed here.

Public Methods

	Name	Description
	DefineLabel (see page 1435)	This is DefineLabel, a member of class IAdapterErrorsStub.
	FieldErrors (see page 1435)	This is FieldErrors, a member of class IAdapterErrorsStub.

10.2.1.2.1 IAdapterErrorsStub.DefineLabel Method

Pascal

```
procedure DefineLabel(const Object_: IUnknown; const Value: string);
```

C++

```
__fastcall DefineLabel(const IUnknown Object_, const AnsiString Value);
```

Description

This is DefineLabel, a member of class IAdapterErrorsStub.

10.2.1.2.2 IAdapterErrorsStub.FieldErrors Method

Pascal

```
function FieldErrors(const Object_: IUnknown): IInterface;
```

C++

```
__fastcall IInterface FieldErrors(const IUnknown Object_);
```

Description

This is FieldErrors, a member of class IAdapterErrorsStub.

10.2.2 IAdapterErrorStub Interface

Class Hierarchy

AutoAdap.IAdapterErrorStub

Pascal

```
IAdapterErrorStub = interface;
```

C++

```
__interface IAdapterErrorStub;
```

File

AutoAdap

Description

This is class AutoAdap.IAdapterErrorStub.

10.2.2.1 IAdapterErrorStub Members

The following tables list the members exposed by IAdapterErrorStub.

Public Methods

	Name	Description
	Get_Field (see page 1436)	This is Get_Field, a member of class IAdapterErrorStub.
	Get_ID (see page 1436)	This is Get_ID, a member of class IAdapterErrorStub.
	Get_Message (see page 1437)	This is Get_Message, a member of class IAdapterErrorStub.

Public Properties

	Name	Description
R	Field (see page 1437)	This is Field, a member of class IAdapterErrorStub.
R	ID (see page 1437)	This is ID, a member of class IAdapterErrorStub.
R	Message (see page 1437)	This is Message, a member of class IAdapterErrorStub.

10.2.2.2 IAdapterErrorStub Methods

The methods of the IAdapterErrorStub class are listed here.

Public Methods

	Name	Description
	Get_Field (see page 1436)	This is Get_Field, a member of class IAdapterErrorStub.
	Get_ID (see page 1436)	This is Get_ID, a member of class IAdapterErrorStub.
	Get_Message (see page 1437)	This is Get_Message, a member of class IAdapterErrorStub.

10.2.2.2.1 IAdapterErrorStub.Get_Field Method

Pascal

```
function Get_Field: IInterface;
```

C++

```
__fastcall IInterface Get_Field();
```

Description

This is Get_Field, a member of class IAdapterErrorStub.

10.2.2.2.2 IAdapterErrorStub.Get_ID Method

Pascal

```
function Get_ID: Integer;
```

C++

```
__fastcall int Get_ID();
```

Description

This is Get_ID, a member of class IAdapterErrorStub.

10.2.2.2.3 IAdapterErrorStub.Get_Message Method

Pascal

```
function Get_Message: string;
```

C++

```
__fastcall AnsiString Get_Message();
```

Description

This is Get_Message, a member of class IAdapterErrorStub.

10.2.2.3 IAdapterErrorStub Properties

The properties of the IAdapterErrorStub class are listed here.

Public Properties

	Name	Description
R	Field (see page 1437)	This is Field, a member of class IAdapterErrorStub.
R	ID (see page 1437)	This is ID, a member of class IAdapterErrorStub.
R	Message (see page 1437)	This is Message, a member of class IAdapterErrorStub.

10.2.2.3.1 IAdapterErrorStub.Field Property

Pascal

```
property Field: IInterface;
```

C++

```
__property IInterface Field;
```

Description

This is Field, a member of class IAdapterErrorStub.

10.2.2.3.2 IAdapterErrorStub.ID Property

Pascal

```
property ID: Integer;
```

C++

```
__property int ID;
```

Description

This is ID, a member of class IAdapterErrorStub.

10.2.2.3.3 IAdapterErrorStub.Message Property

Pascal

```
property Message: string;
```

C++

```
__property AnsiString Message;
```

Description

This is Message, a member of class IAdapterErrorStub.

10.2.3 IAdapterFieldStub Interface

Class Hierarchy

AutoAdap.IAdapterFieldStub

Pascal

```
IAdapterFieldStub = interface;
```

C++

```
__interface IAdapterFieldStub;
```

File

AutoAdap

Description

This is class AutoAdap.IAdapterFieldStub.

10.2.3.1 IAdapterFieldStub Members

The following tables list the members exposed by IAdapterFieldStub.

Public Methods

	Name	Description
	Get_CanModify (see page 1439)	This is Get_CanModify, a member of class IAdapterFieldStub.
	Get_CanView (see page 1440)	This is Get_CanView, a member of class IAdapterFieldStub.
	Get_DisplayLabel (see page 1440)	This is Get_DisplayLabel, a member of class IAdapterFieldStub.
	Get_DisplayStyle (see page 1440)	This is Get_DisplayStyle, a member of class IAdapterFieldStub.
	Get_DisplayText (see page 1440)	This is Get_DisplayText, a member of class IAdapterFieldStub.
	Get_DisplayWidth (see page 1441)	This is Get_DisplayWidth, a member of class IAdapterFieldStub.
	Get_EditText (see page 1441)	This is Get_EditText, a member of class IAdapterFieldStub.
	Get_Image (see page 1441)	This is Get_Image, a member of class IAdapterFieldStub.
	Get_InputName (see page 1441)	This is Get_InputName, a member of class IAdapterFieldStub.
	Get_InputStyle (see page 1441)	This is Get_InputStyle, a member of class IAdapterFieldStub.
	Get_MaxLength (see page 1442)	This is Get_MaxLength, a member of class IAdapterFieldStub.
	Get_Name (see page 1442)	This is Get_Name, a member of class IAdapterFieldStub.
	Get_Required (see page 1442)	This is Get_Required, a member of class IAdapterFieldStub.
	Get_Value (see page 1442)	This is Get_Value, a member of class IAdapterFieldStub.
	Get_Values (see page 1443)	This is Get_Values, a member of class IAdapterFieldStub.
	Get_ValuesList (see page 1443)	This is Get_ValuesList, a member of class IAdapterFieldStub.
	Get_ViewMode (see page 1443)	This is Get_ViewMode, a member of class IAdapterFieldStub.
	Get_Visible (see page 1443)	This is Get_Visible, a member of class IAdapterFieldStub.
	IsEqual (see page 1443)	This is IsEqual, a member of class IAdapterFieldStub.

Public Properties

	Name	Description
R	CanModify (see page 1444)	This is CanModify, a member of class IAdapterFieldStub.

R	CanView (see page 1444)	This is CanView, a member of class IAdapterFieldStub.
R	DisplayLabel (see page 1445)	This is DisplayLabel, a member of class IAdapterFieldStub.
R	DisplayStyle (see page 1445)	This is DisplayStyle, a member of class IAdapterFieldStub.
R	DisplayText (see page 1445)	This is DisplayText, a member of class IAdapterFieldStub.
R	DisplayWidth (see page 1445)	This is DisplayWidth, a member of class IAdapterFieldStub.
R	EditText (see page 1445)	This is EditText, a member of class IAdapterFieldStub.
R	Image (see page 1446)	This is Image, a member of class IAdapterFieldStub.
R	InputName (see page 1446)	This is InputName, a member of class IAdapterFieldStub.
R	InputStyle (see page 1446)	This is InputStyle, a member of class IAdapterFieldStub.
R	MaxLength (see page 1446)	This is MaxLength, a member of class IAdapterFieldStub.
R	Name (see page 1447)	This is Name, a member of class IAdapterFieldStub.
R	Required (see page 1447)	This is Required, a member of class IAdapterFieldStub.
R	Value (see page 1447)	This is Value, a member of class IAdapterFieldStub.
R	Values (see page 1447)	This is Values, a member of class IAdapterFieldStub.
R	ValuesList (see page 1447)	This is ValuesList, a member of class IAdapterFieldStub.
R	ViewMode (see page 1448)	This is ViewMode, a member of class IAdapterFieldStub.
R	Visible (see page 1448)	This is Visible, a member of class IAdapterFieldStub.

10.2.3.2 IAdapterFieldStub Methods

The methods of the IAdapterFieldStub class are listed here.

Public Methods

	Name	Description
	Get_CanModify (see page 1439)	This is Get_CanModify, a member of class IAdapterFieldStub.
	Get_CanView (see page 1440)	This is Get_CanView, a member of class IAdapterFieldStub.
	Get_DisplayLabel (see page 1440)	This is Get_DisplayLabel, a member of class IAdapterFieldStub.
	Get_DisplayStyle (see page 1440)	This is Get_DisplayStyle, a member of class IAdapterFieldStub.
	Get_DisplayText (see page 1440)	This is Get_DisplayText, a member of class IAdapterFieldStub.
	Get_DisplayWidth (see page 1441)	This is Get_DisplayWidth, a member of class IAdapterFieldStub.
	Get_EditText (see page 1441)	This is Get_EditText, a member of class IAdapterFieldStub.
	Get_Image (see page 1441)	This is Get_Image, a member of class IAdapterFieldStub.
	Get_InputName (see page 1441)	This is Get_InputName, a member of class IAdapterFieldStub.
	Get_InputStyle (see page 1441)	This is Get_InputStyle, a member of class IAdapterFieldStub.
	Get_MaxLength (see page 1442)	This is Get_MaxLength, a member of class IAdapterFieldStub.
	Get_Name (see page 1442)	This is Get_Name, a member of class IAdapterFieldStub.
	Get_Required (see page 1442)	This is Get_Required, a member of class IAdapterFieldStub.
	Get_Value (see page 1442)	This is Get_Value, a member of class IAdapterFieldStub.
	Get_Values (see page 1443)	This is Get_Values, a member of class IAdapterFieldStub.
	Get_ValuesList (see page 1443)	This is Get_ValuesList, a member of class IAdapterFieldStub.
	Get_ViewMode (see page 1443)	This is Get_ViewMode, a member of class IAdapterFieldStub.
	Get_Visible (see page 1443)	This is Get_Visible, a member of class IAdapterFieldStub.
	IsEqual (see page 1443)	This is IsEqual, a member of class IAdapterFieldStub.

10.2.3.2.1 IAdapterFieldStub.Get_CanModify Method

Pascal

```
function Get_CanModify: Boolean;
```

C++

```
__fastcall Boolean Get_CanModify();
```

Description

This is Get_CanModify, a member of class IAdapterFieldStub.

10.2.3.2.2 IAdapterFieldStub.Get_CanView Method

Pascal

```
function Get_CanView: Boolean;
```

C++

```
__fastcall Boolean Get_CanView();
```

Description

This is Get_CanView, a member of class IAdapterFieldStub.

10.2.3.2.3 IAdapterFieldStub.Get_DisplayLabel Method

Pascal

```
function Get_DisplayLabel: string;
```

C++

```
__fastcall AnsiString Get_DisplayLabel();
```

Description

This is Get_DisplayLabel, a member of class IAdapterFieldStub.

10.2.3.2.4 IAdapterFieldStub.Get_DisplayStyle Method

Pascal

```
function Get_DisplayStyle: string;
```

C++

```
__fastcall AnsiString Get_DisplayStyle();
```

Description

This is Get_DisplayStyle, a member of class IAdapterFieldStub.

10.2.3.2.5 IAdapterFieldStub.Get_DisplayText Method

Pascal

```
function Get_DisplayText: string;
```

C++

```
__fastcall AnsiString Get_DisplayText();
```

Description

This is Get_DisplayText, a member of class IAdapterFieldStub.

10.2.3.2.6 IAdapterFieldStub.Get_DisplayWidth Method

Pascal

```
function Get_DisplayWidth: Integer;
```

C++

```
__fastcall int Get_DisplayWidth();
```

Description

This is Get_DisplayWidth, a member of class IAdapterFieldStub.

10.2.3.2.7 IAdapterFieldStub.Get_EditText Method

Pascal

```
function Get_EditText: string;
```

C++

```
__fastcall AnsiString Get_EditText();
```

Description

This is Get_EditText, a member of class IAdapterFieldStub.

10.2.3.2.8 IAdapterFieldStub.Get_Image Method

Pascal

```
function Get_Image: IInterface;
```

C++

```
__fastcall IInterface Get_Image();
```

Description

This is Get_Image, a member of class IAdapterFieldStub.

10.2.3.2.9 IAdapterFieldStub.Get_InputName Method

Pascal

```
function Get_InputName: string;
```

C++

```
__fastcall AnsiString Get_InputName();
```

Description

This is Get_InputName, a member of class IAdapterFieldStub.

10.2.3.2.10 IAdapterFieldStub.Get_InputStyle Method

Pascal

```
function Get_InputStyle: string;
```

C++

```
__fastcall AnsiString Get_InputStyle();
```

Description

This is Get_InputStyle, a member of class IAdapterFieldStub.

10.2.3.2.11 IAdapterFieldStub.Get_MaxLength Method

Pascal

```
function Get_MaxLength: Integer;
```

C++

```
__fastcall int Get_MaxLength();
```

Description

This is Get_MaxLength, a member of class IAdapterFieldStub.

10.2.3.2.12 IAdapterFieldStub.Get_Name Method

Pascal

```
function Get_Name: Variant;
```

C++

```
__fastcall Variant Get_Name();
```

Description

This is Get_Name, a member of class IAdapterFieldStub.

10.2.3.2.13 IAdapterFieldStub.Get_Required Method

Pascal

```
function Get_Required: Boolean;
```

C++

```
__fastcall Boolean Get_Required();
```

Description

This is Get_Required, a member of class IAdapterFieldStub.

10.2.3.2.14 IAdapterFieldStub.Get_Value Method

Pascal

```
function Get_Value: Variant;
```

C++

```
__fastcall Variant Get_Value();
```

Description

This is Get_Value, a member of class IAdapterFieldStub.

10.2.3.2.15 IAdapterFieldStub.Get_Values Method

Pascal

```
function Get_Values: IInterface;
```

C++

```
__fastcall IInterface Get_Values();
```

Description

This is Get_Values, a member of class IAdapterFieldStub.

10.2.3.2.16 IAdapterFieldStub.Get_ValuesList Method

Pascal

```
function Get_ValuesList: IInterface;
```

C++

```
__fastcall IInterface Get_ValuesList();
```

Description

This is Get_ValuesList, a member of class IAdapterFieldStub.

10.2.3.2.17 IAdapterFieldStub.Get_ViewMode Method

Pascal

```
function Get_ViewMode: string;
```

C++

```
__fastcall AnsiString Get_ViewMode();
```

Description

This is Get_ViewMode, a member of class IAdapterFieldStub.

10.2.3.2.18 IAdapterFieldStub.Get_Visible Method

Pascal

```
function Get_Visible: Boolean;
```

C++

```
__fastcall Boolean Get_Visible();
```

Description

This is Get_Visible, a member of class IAdapterFieldStub.

10.2.3.2.19 IAdapterFieldStub.IsEqual Method

Pascal

```
function IsEqual(Value: Variant): Boolean;
```

C++

```
__fastcall Boolean IsEqual(Variant Value);
```

Description

This is IsEqual, a member of class IAdapterFieldStub.

10.2.3.3 IAdapterFieldStub Properties

The properties of the IAdapterFieldStub class are listed here.

Public Properties

	Name	Description
R	CanModify (see page 1444)	This is CanModify, a member of class IAdapterFieldStub.
R	CanView (see page 1444)	This is CanView, a member of class IAdapterFieldStub.
R	DisplayLabel (see page 1445)	This is DisplayLabel, a member of class IAdapterFieldStub.
R	DisplayStyle (see page 1445)	This is DisplayStyle, a member of class IAdapterFieldStub.
R	DisplayText (see page 1445)	This is DisplayText, a member of class IAdapterFieldStub.
R	DisplayWidth (see page 1445)	This is DisplayWidth, a member of class IAdapterFieldStub.
R	EditText (see page 1445)	This is EditText, a member of class IAdapterFieldStub.
R	Image (see page 1446)	This is Image, a member of class IAdapterFieldStub.
R	InputName (see page 1446)	This is InputName, a member of class IAdapterFieldStub.
R	InputStyle (see page 1446)	This is InputStyle, a member of class IAdapterFieldStub.
R	MaxLength (see page 1446)	This is MaxLength, a member of class IAdapterFieldStub.
R	Name (see page 1447)	This is Name, a member of class IAdapterFieldStub.
R	Required (see page 1447)	This is Required, a member of class IAdapterFieldStub.
R	Value (see page 1447)	This is Value, a member of class IAdapterFieldStub.
R	Values (see page 1447)	This is Values, a member of class IAdapterFieldStub.
R	ValuesList (see page 1447)	This is ValuesList, a member of class IAdapterFieldStub.
R	ViewMode (see page 1448)	This is ViewMode, a member of class IAdapterFieldStub.
R	Visible (see page 1448)	This is Visible, a member of class IAdapterFieldStub.

10.2.3.3.1 IAdapterFieldStub.CanModify Property

Pascal

```
property CanModify: Boolean;
```

C++

```
__property Boolean CanModify;
```

Description

This is CanModify, a member of class IAdapterFieldStub.

10.2.3.3.2 IAdapterFieldStub.CanView Property

Pascal

```
property CanView: Boolean;
```

C++

```
__property Boolean CanView;
```

Description

This is CanView, a member of class IAdapterFieldStub.

10.2.3.3.3 IAdapterFieldStub.DisplayLabel Property

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

This is DisplayLabel, a member of class IAdapterFieldStub.

10.2.3.3.4 IAdapterFieldStub.DisplayStyle Property

Pascal

```
property DisplayStyle: string;
```

C++

```
__property AnsiString DisplayStyle;
```

Description

This is DisplayStyle, a member of class IAdapterFieldStub.

10.2.3.3.5 IAdapterFieldStub.DisplayText Property

Pascal

```
property DisplayText: string;
```

C++

```
__property AnsiString DisplayText;
```

Description

This is DisplayText, a member of class IAdapterFieldStub.

10.2.3.3.6 IAdapterFieldStub.DisplayWidth Property

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

This is DisplayWidth, a member of class IAdapterFieldStub.

10.2.3.3.7 IAdapterFieldStub.EditText Property

Pascal

```
property EditText: string;
```

C++

```
__property AnsiString EditText;
```

Description

This is EditText, a member of class IAdapterFieldStub.

10.2.3.3.8 IAdapterFieldStub.Image Property

Pascal

```
property Image: IInterface;
```

C++

```
__property IInterface Image;
```

Description

This is Image, a member of class IAdapterFieldStub.

10.2.3.3.9 IAdapterFieldStub.InputName Property

Pascal

```
property InputName: string;
```

C++

```
__property AnsiString InputName;
```

Description

This is InputName, a member of class IAdapterFieldStub.

10.2.3.3.10 IAdapterFieldStub.InputStyle Property

Pascal

```
property InputStyle: string;
```

C++

```
__property AnsiString InputStyle;
```

Description

This is InputStyle, a member of class IAdapterFieldStub.

10.2.3.3.11 IAdapterFieldStub.MaxLength Property

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

This is MaxLength, a member of class IAdapterFieldStub.

10.2.3.3.12 IAdapterFieldStub.Name Property

Pascal

```
property Name: Variant;
```

C++

```
__property Variant Name;
```

Description

This is Name, a member of class IAdapterFieldStub.

10.2.3.3.13 IAdapterFieldStub.Required Property

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;
```

Description

This is Required, a member of class IAdapterFieldStub.

10.2.3.3.14 IAdapterFieldStub.Value Property

Pascal

```
property Value: Variant;
```

C++

```
__property Variant Value;
```

Description

This is Value, a member of class IAdapterFieldStub.

10.2.3.3.15 IAdapterFieldStub.Values Property

Pascal

```
property Values: IInterface;
```

C++

```
__property IInterface Values;
```

Description

This is Values, a member of class IAdapterFieldStub.

10.2.3.3.16 IAdapterFieldStub.ValuesList Property

Pascal

```
property ValuesList: IInterface;
```

C++

```
__property IInterface ValuesList;
```

Description

This is ValuesList, a member of class IAdapterFieldStub.

10.2.3.3.17 IAdapterFieldStub.ViewMode Property

Pascal

```
property ViewMode: string;
```

C++

```
__property AnsiString ViewMode;
```

Description

This is ViewMode, a member of class IAdapterFieldStub.

10.2.3.3.18 IAdapterFieldStub.Visible Property

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

This is Visible, a member of class IAdapterFieldStub.

10.2.4 IAdapterFieldValuesStub Interface

Class Hierarchy

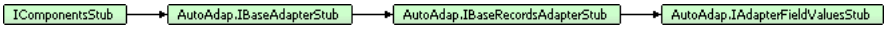

Pascal

```
IAdapterFieldValuesStub = interface(IBaseRecordsAdapterStub);
```

C++

```
__interface IAdapterFieldValuesStub : public IBaseRecordsAdapterStub;
```

File

AutoAdap

Description

This is class AutoAdap.IAdapterFieldValuesStub.

10.2.4.1 IAdapterFieldValuesStub Members

The following tables list the members exposed by IAdapterFieldValuesStub.

Public Methods

	Name	Description
	Get_Actions (see page 1467)	This is Get_Actions, a member of class IBaseAdapterStub.
	Get_ClassName_ (see page 1467)	This is Get_ClassName_, a member of class IBaseAdapterStub.
	Get_Fields (see page 1467)	This is Get_Fields, a member of class IBaseAdapterStub.
	Get_Name_ (see page 1467)	This is Get_Name_, a member of class IBaseAdapterStub.
	Get_Records (see page 1469)	This is Get_Records, a member of class IBaseRecordsAdapterStub.
	Get_Value (see page 1449)	This is Get_Value, a member of class IAdapterFieldValuesStub.
	HasValue (see page 1449)	This is HasValue, a member of class IAdapterFieldValuesStub.

Public Properties

	Name	Description
R	Actions (see page 1468)	This is Actions, a member of class IBaseAdapterStub.
R	Fields (see page 1468)	This is Fields, a member of class IBaseAdapterStub.
R	Records (see page 1469)	This is Records, a member of class IBaseRecordsAdapterStub.
R	Value (see page 1450)	This is Value, a member of class IAdapterFieldValuesStub.

10.2.4.2 IAdapterFieldValuesStub Methods

The methods of the IAdapterFieldValuesStub class are listed here.

Public Methods

	Name	Description
	Get_Value (see page 1449)	This is Get_Value, a member of class IAdapterFieldValuesStub.
	HasValue (see page 1449)	This is HasValue, a member of class IAdapterFieldValuesStub.

10.2.4.2.1 IAdapterFieldValuesStub.Get_Value Method

Pascal

```
function Get_Value: Variant;
```

C++

```
__fastcall Variant Get_Value();
```

Description

This is Get_Value, a member of class IAdapterFieldValuesStub.

10.2.4.2.2 IAdapterFieldValuesStub.HasValue Method

Pascal

```
function HasValue(Value: Variant): Boolean;
```

C++

```
__fastcall Boolean HasValue(Variant Value);
```

Description

This is HasValue, a member of class IAdapterFieldValuesStub.

10.2.4.3 IAdapterFieldValuesStub Properties

The properties of the IAdapterFieldValuesStub class are listed here.

Public Properties

	Name	Description
	Value (see page 1450)	This is Value, a member of class IAdapterFieldValuesStub.

10.2.4.3.1 IAdapterFieldValuesStub.Value Property

Pascal

```
property Value: Variant;
```

C++

```
__property Variant Value;
```

Description

This is Value, a member of class IAdapterFieldValuesStub.

10.2.5 IAdapterHiddenFieldsStub Interface

Class Hierarchy

AutoAdap.IAdapterHiddenFieldsStub

Pascal

```
IAdapterHiddenFieldsStub = interface;
```

C++

```
__interface IAdapterHiddenFieldsStub;
```

File

AutoAdap

Description

This is class AutoAdap.IAdapterHiddenFieldsStub.

10.2.5.1 IAdapterHiddenFieldsStub Members

The following tables list the members exposed by IAdapterHiddenFieldsStub.

Public Methods

	Name	Description
	WriteFields (see page 1451)	This is WriteFields, a member of class IAdapterHiddenFieldsStub.

10.2.5.2 IAdapterHiddenFieldsStub Methods

The methods of the IAdapterHiddenFieldsStub class are listed here.

Public Methods

	Name	Description
	WriteFields (see page 1451)	This is WriteFields, a member of class IAdapterHiddenFieldsStub.

10.2.5.2.1 IAdapterHiddenFieldsStub.WriteFields Method

Pascal

```
procedure WriteFields(const Response: IInterface);
```

C++

```
__fastcall WriteFields(const IInterface Response);
```

Description

This is WriteFields, a member of class IAdapterHiddenFieldsStub.

10.2.6 IAdapterHiddenFieldStub Interface

Class Hierarchy

AutoAdap.IAdapterHiddenFieldStub

Pascal

```
IAdapterHiddenFieldStub = interface;
```

C++

```
__interface IAdapterHiddenFieldStub;
```

File

AutoAdap

Description

This is class AutoAdap.IAdapterHiddenFieldStub.

10.2.6.1 IAdapterHiddenFieldStub Members

The following tables list the members exposed by IAdapterHiddenFieldStub.

Public Methods

	Name	Description
	Get_Name (see page 1452)	This is Get_Name, a member of class IAdapterHiddenFieldStub.
	Get_Value (see page 1452)	This is Get_Value, a member of class IAdapterHiddenFieldStub.

Public Properties

	Name	Description
R	Name (see page 1452)	This is Name, a member of class IAdapterHiddenFieldStub.
R	Value (see page 1453)	This is Value, a member of class IAdapterHiddenFieldStub.

10.2.6.2 IAdapterHiddenFieldStub Methods

The methods of the IAdapterHiddenFieldStub class are listed here.

Public Methods

	Name	Description
	Get_Name (see page 1452)	This is Get_Name, a member of class IAdapterHiddenFieldStub.
	Get_Value (see page 1452)	This is Get_Value, a member of class IAdapterHiddenFieldStub.

10.2.6.2.1 IAdapterHiddenFieldStub.Get_Name Method

Pascal

```
function Get_Name: string;
```

C++

```
__fastcall AnsiString Get_Name();
```

Description

This is Get_Name, a member of class IAdapterHiddenFieldStub.

10.2.6.2.2 IAdapterHiddenFieldStub.Get_Value Method

Pascal

```
function Get_Value: string;
```

C++

```
__fastcall AnsiString Get_Value();
```

Description

This is Get_Value, a member of class IAdapterHiddenFieldStub.

10.2.6.3 IAdapterHiddenFieldStub Properties

The properties of the IAdapterHiddenFieldStub class are listed here.

Public Properties

	Name	Description
R	Name (see page 1452)	This is Name, a member of class IAdapterHiddenFieldStub.
R	Value (see page 1453)	This is Value, a member of class IAdapterHiddenFieldStub.

10.2.6.3.1 IAdapterHiddenFieldStub.Name Property

Pascal

```
property Name: string;
```

C++

```
__property AnsiString Name;
```

Description

This is Name, a member of class IAdapterHiddenFieldStub.

10.2.6.3.2 IAdapterHiddenFieldStub.Value Property

Pascal

```
property Value: string;
```

C++

```
__property AnsiString Value;
```

Description

This is Value, a member of class IAdapterHiddenFieldStub.

10.2.7 IAdapterImageStub Interface

Class Hierarchy

AutoAdap.IAdapterImageStub

Pascal

```
IAdapterImageStub = interface;
```

C++

```
__interface IAdapterImageStub;
```

File

AutoAdap

Description

This is class AutoAdap.IAdapterImageStub.

10.2.7.1 IAdapterImageStub Members

The following tables list the members exposed by IAdapterImageStub.

Public Methods

	Name	Description
	Get_AsHREF (see page 1454)	This is Get_AsHREF, a member of class IAdapterImageStub.

Public Properties

	Name	Description
R	AsHREF (see page 1454)	This is AsHREF, a member of class IAdapterImageStub.

10.2.7.2 IAdapterImageStub Methods

The methods of the IAdapterImageStub class are listed here.

Public Methods

	Name	Description
	Get_AsHREF (see page 1454)	This is Get_AsHREF, a member of class IAdapterImageStub.

10.2.7.2.1 IAdapterImageStub.Get_AsHREF Method

Pascal

```
function Get_AsHREF: string;
```

C++

```
__fastcall AnsiString Get_AsHREF();
```

Description

This is Get_AsHREF, a member of class IAdapterImageStub.

10.2.7.3 IAdapterImageStub Properties

The properties of the IAdapterImageStub class are listed here.

Public Properties

	Name	Description
	AsHREF (see page 1454)	This is AsHREF, a member of class IAdapterImageStub.

10.2.7.3.1 IAdapterImageStub.AsHREF Property

Pascal

```
property AsHREF: string;
```

C++

```
__property AnsiString AsHREF;
```

Description

This is AsHREF, a member of class IAdapterImageStub.

10.2.8 IAdapterRecordStub Interface

Class Hierarchy

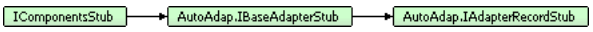

Pascal

```
IAdapterRecordStub = interface(IBaseAdapterStub);
```

C++

```
__interface IAdapterRecordStub : public IBaseAdapterStub;
```

File

AutoAdap

Description

This is class AutoAdap.IAdapterRecordStub.

10.2.9 IAdapterStub Interface

Class Hierarchy

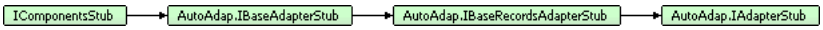

Pascal

```
IAdapterStub = interface(IBaseRecordsAdapterStub);
```

C++

```
__interface IAdapterStub : public IBaseRecordsAdapterStub;
```

File

AutoAdap

Description

This is class AutoAdap.IAdapterStub.

10.2.9.1 IAdapterStub Members

The following tables list the members exposed by IAdapterStub.

Public Methods

	Name	Description
	Get_Actions (see page 1467)	This is Get_Actions, a member of class IBaseAdapterStub.
	Get_CanModify (see page 1456)	This is Get_CanModify, a member of class IAdapterStub.
	Get_CanView (see page 1456)	This is Get_CanView, a member of class IAdapterStub.
	Get_ClassName_ (see page 1467)	This is Get_ClassName_, a member of class IBaseAdapterStub.
	Get_Errors (see page 1456)	This is Get_Errors, a member of class IAdapterStub.
	Get_Fields (see page 1467)	This is Get_Fields, a member of class IBaseAdapterStub.
	Get_HiddenFields (see page 1457)	This is Get_HiddenFields, a member of class IAdapterStub.
	Get_HiddenRecordFields (see page 1457)	This is Get_HiddenRecordFields, a member of class IAdapterStub.
	Get_Mode (see page 1457)	This is Get_Mode, a member of class IAdapterStub.
	Get_Name_ (see page 1467)	This is Get_Name_, a member of class IBaseAdapterStub.
	Get_Records (see page 1469)	This is Get_Records, a member of class IBaseRecordsAdapterStub.
	Set_Mode (see page 1457)	This is Set_Mode, a member of class IAdapterStub.

Public Properties

	Name	Description
R	Actions (see page 1468)	This is Actions, a member of class IBaseAdapterStub.
R	CanModify (see page 1458)	This is CanModify, a member of class IAdapterStub.
R	CanView (see page 1458)	This is CanView, a member of class IAdapterStub.
R	Errors (see page 1458)	This is Errors, a member of class IAdapterStub.
R	Fields (see page 1468)	This is Fields, a member of class IBaseAdapterStub.
R	HiddenFields (see page 1458)	This is HiddenFields, a member of class IAdapterStub.
R	HiddenRecordFields (see page 1459)	This is HiddenRecordFields, a member of class IAdapterStub.
	Mode (see page 1459)	This is Mode, a member of class IAdapterStub.
R	Records (see page 1469)	This is Records, a member of class IBaseRecordsAdapterStub.

10.2.9.2 IAdapterStub Methods

The methods of the IAdapterStub class are listed here.

Public Methods

	Name	Description
	Get_CanModify (see page 1456)	This is Get_CanModify, a member of class IAdapterStub.
	Get_CanView (see page 1456)	This is Get_CanView, a member of class IAdapterStub.
	Get_Errors (see page 1456)	This is Get_Errors, a member of class IAdapterStub.
	Get_HiddenFields (see page 1457)	This is Get_HiddenFields, a member of class IAdapterStub.
	Get_HiddenRecordFields (see page 1457)	This is Get_HiddenRecordFields, a member of class IAdapterStub.
	Get_Mode (see page 1457)	This is Get_Mode, a member of class IAdapterStub.
	Set_Mode (see page 1457)	This is Set_Mode, a member of class IAdapterStub.

10.2.9.2.1 IAdapterStub.Get_CanModify Method

Pascal

```
function Get_CanModify: Boolean;
```

C++

```
__fastcall Boolean Get_CanModify();
```

Description

This is Get_CanModify, a member of class IAdapterStub.

10.2.9.2.2 IAdapterStub.Get_CanView Method

Pascal

```
function Get_CanView: Boolean;
```

C++

```
__fastcall Boolean Get_CanView();
```

Description

This is Get_CanView, a member of class IAdapterStub.

10.2.9.2.3 IAdapterStub.Get_Errors Method

Pascal

```
function Get_Errors: IInterface;
```

C++

```
__fastcall IInterface Get_Errors();
```

Description

This is Get_Errors, a member of class IAdapterStub.

10.2.9.2.4 IAdapterStub.Get_HiddenFields Method

Pascal

```
function Get_HiddenFields: IInterface;
```

C++

```
__fastcall IInterface Get_HiddenFields();
```

Description

This is Get_HiddenFields, a member of class IAdapterStub.

10.2.9.2.5 IAdapterStub.Get_HiddenRecordFields Method

Pascal

```
function Get_HiddenRecordFields: IInterface;
```

C++

```
__fastcall IInterface Get_HiddenRecordFields();
```

Description

This is Get_HiddenRecordFields, a member of class IAdapterStub.

10.2.9.2.6 IAdapterStub.Get_Mode Method

Pascal

```
function Get_Mode: string;
```

C++

```
__fastcall AnsiString Get_Mode();
```

Description

This is Get_Mode, a member of class IAdapterStub.

10.2.9.2.7 IAdapterStub.Set_Mode Method

Pascal

```
procedure Set_Mode(const Value: string);
```

C++

```
__fastcall Set_Mode(const AnsiString Value);
```

Description

This is Set_Mode, a member of class IAdapterStub.

10.2.9.3 IAdapterStub Properties

The properties of the IAdapterStub class are listed here.

Public Properties

	Name	Description
R	CanModify (see page 1458)	This is CanModify, a member of class IAdapterStub.
R	CanView (see page 1458)	This is CanView, a member of class IAdapterStub.
R	Errors (see page 1458)	This is Errors, a member of class IAdapterStub.
R	HiddenFields (see page 1458)	This is HiddenFields, a member of class IAdapterStub.
R	HiddenRecordFields (see page 1459)	This is HiddenRecordFields, a member of class IAdapterStub.
	Mode (see page 1459)	This is Mode, a member of class IAdapterStub.

10.2.9.3.1 IAdapterStub.CanModify Property

Pascal

```
property CanModify: Boolean;
```

C++

```
__property Boolean CanModify;
```

Description

This is CanModify, a member of class IAdapterStub.

10.2.9.3.2 IAdapterStub.CanView Property

Pascal

```
property CanView: Boolean;
```

C++

```
__property Boolean CanView;
```

Description

This is CanView, a member of class IAdapterStub.

10.2.9.3.3 IAdapterStub.Errors Property

Pascal

```
property Errors: IInterface;
```

C++

```
__property IInterface Errors;
```

Description

This is Errors, a member of class IAdapterStub.

10.2.9.3.4 IAdapterStub.HiddenFields Property

Pascal

```
property HiddenFields: IInterface;
```

C++

```
__property IInterface HiddenFields;
```

Description

This is HiddenFields, a member of class IAdapterStub.

10.2.9.3.5 IAdapterStub.HiddenRecordFields Property

Pascal

```
property HiddenRecordFields: IInterface;
```

C++

```
__property IInterface HiddenRecordFields;
```

Description

This is HiddenRecordFields, a member of class IAdapterStub.

10.2.9.3.6 IAdapterStub.Mode Property

Pascal

```
property Mode: string;
```

C++

```
__property AnsiString Mode;
```

Description

This is Mode, a member of class IAdapterStub.

10.2.10 IAdapterValuesListStub Interface

Class Hierarchy

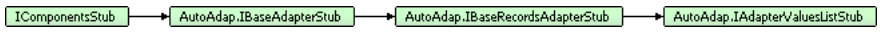

Pascal

```
IAdapterValuesListStub = interface(IBaseRecordsAdapterStub);
```

C++

```
__interface IAdapterValuesListStub : public IBaseRecordsAdapterStub;
```

File

AutoAdap

Description

This is class AutoAdap.IAdapterValuesListStub.

10.2.10.1 IAdapterValuesListStub Members

The following tables list the members exposed by IAdapterValuesListStub.

Public Methods

	Name	Description
	Get_Actions (see page 1467)	This is Get_Actions, a member of class IBaseAdapterStub.
	Get_ClassName_ (see page 1467)	This is Get_ClassName_, a member of class IBaseAdapterStub.
	Get_Fields (see page 1467)	This is Get_Fields, a member of class IBaseAdapterStub.
	Get_Image (see page 1460)	This is Get_Image, a member of class IAdapterValuesListStub.
	Get_Name_ (see page 1467)	This is Get_Name_, a member of class IBaseAdapterStub.
	Get_Records (see page 1469)	This is Get_Records, a member of class IBaseRecordsAdapterStub.
	Get_Value (see page 1460)	This is Get_Value, a member of class IAdapterValuesListStub.
	Get_ValueName (see page 1461)	This is Get_ValueName, a member of class IAdapterValuesListStub.
	ImageOfValue (see page 1461)	This is ImageOfValue, a member of class IAdapterValuesListStub.
	NameOfValue (see page 1461)	This is NameOfValue, a member of class IAdapterValuesListStub.

Public Properties

	Name	Description
R	Actions (see page 1468)	This is Actions, a member of class IBaseAdapterStub.
R	Fields (see page 1468)	This is Fields, a member of class IBaseAdapterStub.
R	Image (see page 1461)	This is Image, a member of class IAdapterValuesListStub.
R	Records (see page 1469)	This is Records, a member of class IBaseRecordsAdapterStub.
R	Value (see page 1462)	This is Value, a member of class IAdapterValuesListStub.
R	ValueName (see page 1462)	This is ValueName, a member of class IAdapterValuesListStub.

10.2.10.2 IAdapterValuesListStub Methods

The methods of the IAdapterValuesListStub class are listed here.

Public Methods

	Name	Description
	Get_Image (see page 1460)	This is Get_Image, a member of class IAdapterValuesListStub.
	Get_Value (see page 1460)	This is Get_Value, a member of class IAdapterValuesListStub.
	Get_ValueName (see page 1461)	This is Get_ValueName, a member of class IAdapterValuesListStub.
	ImageOfValue (see page 1461)	This is ImageOfValue, a member of class IAdapterValuesListStub.
	NameOfValue (see page 1461)	This is NameOfValue, a member of class IAdapterValuesListStub.

10.2.10.2.1 IAdapterValuesListStub.Get_Image Method

Pascal

```
function Get_Image: IInterface;
```

C++

```
__fastcall IInterface Get_Image();
```

Description

This is Get_Image, a member of class IAdapterValuesListStub.

10.2.10.2.2 IAdapterValuesListStub.Get_Value Method

Pascal

```
function Get_Value: Variant;
```

C++

```
__fastcall Variant Get_Value();
```

Description

This is Get_Value, a member of class IAdapterValuesListStub.

10.2.10.2.3 IAdapterValuesListStub.Get_ValueName Method

Pascal

```
function Get_ValueName: string;
```

C++

```
__fastcall AnsiString Get_ValueName();
```

Description

This is Get_ValueName, a member of class IAdapterValuesListStub.

10.2.10.2.4 IAdapterValuesListStub.ImageOfValue Method

Pascal

```
function ImageOfValue(Value: Variant): IInterface;
```

C++

```
__fastcall IInterface ImageOfValue(Variant Value);
```

Description

This is ImageOfValue, a member of class IAdapterValuesListStub.

10.2.10.2.5 IAdapterValuesListStub.NameOfValue Method

Pascal

```
function NameOfValue(Value: Variant): string;
```

C++

```
__fastcall AnsiString NameOfValue(Variant Value);
```

Description

This is NameOfValue, a member of class IAdapterValuesListStub.

10.2.10.3 IAdapterValuesListStub Properties

The properties of the IAdapterValuesListStub class are listed here.

Public Properties

	Name	Description
R	Image (see page 1461)	This is Image, a member of class IAdapterValuesListStub.
R	Value (see page 1462)	This is Value, a member of class IAdapterValuesListStub.
R	ValueName (see page 1462)	This is ValueName, a member of class IAdapterValuesListStub.

10.2.10.3.1 IAdapterValuesListStub.Image Property

Pascal

```
property Image: IInterface;
```

C++

```
__property IInterface Image;
```

Description

This is Image, a member of class IAdapterValuesListStub.

10.2.10.3.2 IAdapterValuesListStub.Value Property

Pascal

```
property Value: Variant;
```

C++

```
__property Variant Value;
```

Description

This is Value, a member of class IAdapterValuesListStub.

10.2.10.3.3 IAdapterValuesListStub.ValueName Property

Pascal

```
property ValueName: string;
```

C++

```
__property AnsiString ValueName;
```

Description

This is ValueName, a member of class IAdapterValuesListStub.

10.2.11 IApplicationStub Interface

Class Hierarchy

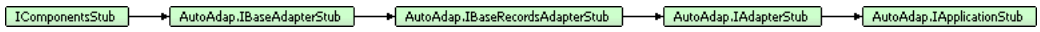

Pascal

```
IApplicationStub = interface(IAdapterStub);
```

C++

```
__interface IApplicationStub : public IAdapterStub;
```

File

AutoAdap

Description

This is class AutoAdap.IApplicationStub.

10.2.11.1 IApplicationStub Members

The following tables list the members exposed by IApplicationStub.

Public Methods

	Name	Description
	Get_Actions (see page 1467)	This is Get_Actions, a member of class IBaseAdapterStub.
	Get_CanModify (see page 1456)	This is Get_CanModify, a member of class IAdapterStub.
	Get_CanView (see page 1456)	This is Get_CanView, a member of class IAdapterStub.
	Get_ClassName_ (see page 1467)	This is Get_ClassName_, a member of class IBaseAdapterStub.
	Get_Designing (see page 1463)	This is Get_Designing, a member of class IApplicationStub.
	Get_Errors (see page 1456)	This is Get_Errors, a member of class IAdapterStub.
	Get_Fields (see page 1467)	This is Get_Fields, a member of class IBaseAdapterStub.
	Get_HiddenFields (see page 1457)	This is Get_HiddenFields, a member of class IAdapterStub.
	Get_HiddenRecordFields (see page 1457)	This is Get_HiddenRecordFields, a member of class IAdapterStub.
	Get_Mode (see page 1457)	This is Get_Mode, a member of class IAdapterStub.
	Get_ModuleFileName (see page 1464)	This is Get_ModuleFileName, a member of class IApplicationStub.
	Get_ModulePath (see page 1464)	This is Get_ModulePath, a member of class IApplicationStub.
	Get_Name_ (see page 1467)	This is Get_Name_, a member of class IBaseAdapterStub.
	Get_Records (see page 1469)	This is Get_Records, a member of class IBaseRecordsAdapterStub.
	Get_Title (see page 1464)	This is Get_Title, a member of class IApplicationStub.
	QualifyFileName (see page 1464)	This is QualifyFileName, a member of class IApplicationStub.
	Set_Mode (see page 1457)	This is Set_Mode, a member of class IAdapterStub.

Public Properties

	Name	Description
R	Actions (see page 1468)	This is Actions, a member of class IBaseAdapterStub.
R	CanModify (see page 1458)	This is CanModify, a member of class IAdapterStub.
R	CanView (see page 1458)	This is CanView, a member of class IAdapterStub.
R	Designing (see page 1465)	This is Designing, a member of class IApplicationStub.
R	Errors (see page 1458)	This is Errors, a member of class IAdapterStub.
R	Fields (see page 1468)	This is Fields, a member of class IBaseAdapterStub.
R	HiddenFields (see page 1458)	This is HiddenFields, a member of class IAdapterStub.
R	HiddenRecordFields (see page 1459)	This is HiddenRecordFields, a member of class IAdapterStub.
	Mode (see page 1459)	This is Mode, a member of class IAdapterStub.
R	ModuleFileName (see page 1465)	This is ModuleFileName, a member of class IApplicationStub.
R	ModulePath (see page 1465)	This is ModulePath, a member of class IApplicationStub.
R	Records (see page 1469)	This is Records, a member of class IBaseRecordsAdapterStub.
R	Title (see page 1465)	This is Title, a member of class IApplicationStub.

10.2.11.2 IApplicationStub Methods

The methods of the IApplicationStub class are listed here.

Public Methods

	Name	Description
	Get_Designing (see page 1463)	This is Get_Designing, a member of class IApplicationStub.
	Get_ModuleFileName (see page 1464)	This is Get_ModuleFileName, a member of class IApplicationStub.
	Get_ModulePath (see page 1464)	This is Get_ModulePath, a member of class IApplicationStub.
	Get_Title (see page 1464)	This is Get_Title, a member of class IApplicationStub.
	QualifyFileName (see page 1464)	This is QualifyFileName, a member of class IApplicationStub.

10.2.11.2.1 IApplicationStub.Get_Designing Method

Pascal

```
function Get_Designing: Boolean;
```

C++

```
__fastcall Boolean Get_Designing();
```

Description

This is Get_Designing, a member of class IApplicationStub.

10.2.11.2.2 IApplicationStub.Get_ModuleFileName Method

Pascal

```
function Get_ModuleFileName: string;
```

C++

```
__fastcall AnsiString Get_ModuleFileName();
```

Description

This is Get_ModuleFileName, a member of class IApplicationStub.

10.2.11.2.3 IApplicationStub.Get_ModulePath Method

Pascal

```
function Get_ModulePath: string;
```

C++

```
__fastcall AnsiString Get_ModulePath();
```

Description

This is Get_ModulePath, a member of class IApplicationStub.

10.2.11.2.4 IApplicationStub.Get_Title Method

Pascal

```
function Get_Title: string;
```

C++

```
__fastcall AnsiString Get_Title();
```

Description

This is Get_Title, a member of class IApplicationStub.

10.2.11.2.5 IApplicationStub.QualifyFileName Method

Pascal

```
function QualifyFileName(const FileName: PChar): string;
```

C++

```
__fastcall AnsiString QualifyFileName(const const char * FileName);
```

Description

This is QualifyFileName, a member of class IApplicationStub.

10.2.11.3 IApplicationStub Properties

The properties of the IApplicationStub class are listed here.

Public Properties

	Name	Description
R	Designing (see page 1465)	This is Designing, a member of class IApplicationStub.
R	ModuleFileName (see page 1465)	This is ModuleFileName, a member of class IApplicationStub.
R	ModulePath (see page 1465)	This is ModulePath, a member of class IApplicationStub.
R	Title (see page 1465)	This is Title, a member of class IApplicationStub.

10.2.11.3.1 IApplicationStub.Designing Property

Pascal

```
property Designing: Boolean;
```

C++

```
__property Boolean Designing;
```

Description

This is Designing, a member of class IApplicationStub.

10.2.11.3.2 IApplicationStub.ModuleFileName Property

Pascal

```
property ModuleFileName: string;
```

C++

```
__property AnsiString ModuleFileName;
```

Description

This is ModuleFileName, a member of class IApplicationStub.

10.2.11.3.3 IApplicationStub.ModulePath Property

Pascal

```
property ModulePath: string;
```

C++

```
__property AnsiString ModulePath;
```

Description

This is ModulePath, a member of class IApplicationStub.

10.2.11.3.4 IApplicationStub.Title Property

Pascal

```
property Title: string;
```

C++

```
__property AnsiString Title;
```

Description

This is Title, a member of class IApplicationStub.

10.2.12 IBaseAdapterStub Interface

Class Hierarchy

Pascal

```
IBaseAdapterStub = interface(IComponentsStub);
```

C++

```
__interface IBaseAdapterStub : public IComponentsStub;
```

File

AutoAdap

Description

This is class AutoAdap.IBaseAdapterStub.

10.2.12.1 IBaseAdapterStub Members

The following tables list the members exposed by IBaseAdapterStub.

Public Methods

	Name	Description
	Get_Actions (see page 1467)	This is Get_Actions, a member of class IBaseAdapterStub.
	Get_ClassName_ (see page 1467)	This is Get_ClassName_, a member of class IBaseAdapterStub.
	Get_Fields (see page 1467)	This is Get_Fields, a member of class IBaseAdapterStub.
	Get_Name_ (see page 1467)	This is Get_Name_, a member of class IBaseAdapterStub.

Public Properties

	Name	Description
R	Actions (see page 1468)	This is Actions, a member of class IBaseAdapterStub.
R	Fields (see page 1468)	This is Fields, a member of class IBaseAdapterStub.

10.2.12.2 IBaseAdapterStub Methods

The methods of the IBaseAdapterStub class are listed here.

Public Methods

	Name	Description
	Get_Actions (see page 1467)	This is Get_Actions, a member of class IBaseAdapterStub.
	Get_ClassName_ (see page 1467)	This is Get_ClassName_, a member of class IBaseAdapterStub.
	Get_Fields (see page 1467)	This is Get_Fields, a member of class IBaseAdapterStub.

	Get_Name_ (see page 1467)	This is Get_Name_, a member of class IBaseAdapterStub.

10.2.12.2.1 IBaseAdapterStub.Get_Actions Method

Pascal

```
function Get_Actions: IInterface;
```

C++

```
__fastcall IInterface Get_Actions();
```

Description

This is Get_Actions, a member of class IBaseAdapterStub.

10.2.12.2.2 IBaseAdapterStub.Get_ClassName_ Method

Pascal

```
function Get_ClassName_: string;
```

C++

```
__fastcall AnsiString Get_ClassName_();
```

Description

This is Get_ClassName_, a member of class IBaseAdapterStub.

10.2.12.2.3 IBaseAdapterStub.Get_Fields Method

Pascal

```
function Get_Fields: IInterface;
```

C++

```
__fastcall IInterface Get_Fields();
```

Description

This is Get_Fields, a member of class IBaseAdapterStub.

10.2.12.2.4 IBaseAdapterStub.Get_Name_ Method

Pascal

```
function Get_Name_: string;
```

C++

```
__fastcall AnsiString Get_Name_();
```

Description

This is Get_Name_, a member of class IBaseAdapterStub.

10.2.12.3 IBaseAdapterStub Properties

The properties of the IBaseAdapterStub class are listed here.

Public Properties

	Name	Description
R	Actions (see page 1468)	This is Actions, a member of class IBaseAdapterStub.
R	Fields (see page 1468)	This is Fields, a member of class IBaseAdapterStub.

10.2.12.3.1 IBaseAdapterStub.Actions Property

Pascal

```
property Actions: IInterface;
```

C++

```
__property IInterface Actions;
```

Description

This is Actions, a member of class IBaseAdapterStub.

10.2.12.3.2 IBaseAdapterStub.Fields Property

Pascal

```
property Fields: IInterface;
```

C++

```
__property IInterface Fields;
```

Description

This is Fields, a member of class IBaseAdapterStub.

10.2.13 IBaseRecordsAdapterStub Interface

Class Hierarchy

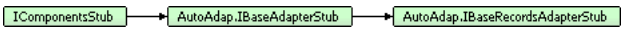

Pascal

```
IBaseRecordsAdapterStub = interface(IBaseAdapterStub);
```

C++

```
__interface IBaseRecordsAdapterStub : public IBaseAdapterStub;
```

File

AutoAdap

Description

This is class AutoAdap.IBaseRecordsAdapterStub.

10.2.13.1 IBaseRecordsAdapterStub Members

The following tables list the members exposed by IBaseRecordsAdapterStub.

Public Methods

	Name	Description
	Get_Actions (see page 1467)	This is Get_Actions, a member of class IBaseAdapterStub.
	Get_ClassName_ (see page 1467)	This is Get_ClassName_, a member of class IBaseAdapterStub.
	Get_Fields (see page 1467)	This is Get_Fields, a member of class IBaseAdapterStub.
	Get_Name_ (see page 1467)	This is Get_Name_, a member of class IBaseAdapterStub.
	Get_Records (see page 1469)	This is Get_Records, a member of class IBaseRecordsAdapterStub.

Public Properties

	Name	Description
R	Actions (see page 1468)	This is Actions, a member of class IBaseAdapterStub.
R	Fields (see page 1468)	This is Fields, a member of class IBaseAdapterStub.
R	Records (see page 1469)	This is Records, a member of class IBaseRecordsAdapterStub.

10.2.13.2 IBaseRecordsAdapterStub Methods

The methods of the IBaseRecordsAdapterStub class are listed here.

Public Methods

	Name	Description
	Get_Records (see page 1469)	This is Get_Records, a member of class IBaseRecordsAdapterStub.

10.2.13.2.1 IBaseRecordsAdapterStub.Get_Records Method

Pascal

```
function Get_Records: IInterface;
```

C++

```
__fastcall IInterface Get_Records();
```

Description

This is Get_Records, a member of class IBaseRecordsAdapterStub.

10.2.13.3 IBaseRecordsAdapterStub Properties

The properties of the IBaseRecordsAdapterStub class are listed here.

Public Properties

	Name	Description
R	Records (see page 1469)	This is Records, a member of class IBaseRecordsAdapterStub.

10.2.13.3.1 IBaseRecordsAdapterStub.Records Property

Pascal

```
property Records: IInterface;
```

C++

```
__property IInterface Records;
```

Description

This is Records, a member of class IBaseRecordsAdapterStub.

10.2.14 IEndUserStub Interface

Class Hierarchy

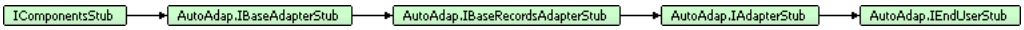

Pascal

```
IEndUserStub = interface(IAdapterStub);
```

C++

```
__interface IEndUserStub : public IAdapterStub;
```

File

AutoAdap

Description

This is class AutoAdap.IEndUserStub.

10.2.14.1 IEndUserStub Members

The following tables list the members exposed by IEndUserStub.

Public Methods

	Name	Description
	Get_Actions (see page 1467)	This is Get_Actions, a member of class IBaseAdapterStub.
	Get_CanModify (see page 1456)	This is Get_CanModify, a member of class IAdapterStub.
	Get_CanView (see page 1456)	This is Get_CanView, a member of class IAdapterStub.
	Get_ClassName_ (see page 1467)	This is Get_ClassName_, a member of class IBaseAdapterStub.
	Get_DisplayName (see page 1471)	This is Get_DisplayName, a member of class IEndUserStub.
	Get_Errors (see page 1456)	This is Get_Errors, a member of class IAdapterStub.
	Get_Fields (see page 1467)	This is Get_Fields, a member of class IBaseAdapterStub.
	Get_HiddenFields (see page 1457)	This is Get_HiddenFields, a member of class IAdapterStub.
	Get_HiddenRecordFields (see page 1457)	This is Get_HiddenRecordFields, a member of class IAdapterStub.
	Get_LoggedIn (see page 1471)	This is Get_LoggedIn, a member of class IEndUserStub.
	Get_LoginFormAction (see page 1471)	This is Get_LoginFormAction, a member of class IEndUserStub.
	Get_LogoutAction (see page 1472)	This is Get_LogoutAction, a member of class IEndUserStub.
	Get_Mode (see page 1457)	This is Get_Mode, a member of class IAdapterStub.
	Get_Name_ (see page 1467)	This is Get_Name_, a member of class IBaseAdapterStub.
	Get_Records (see page 1469)	This is Get_Records, a member of class IBaseRecordsAdapterStub.
	Set_Mode (see page 1457)	This is Set_Mode, a member of class IAdapterStub.

Public Properties

	Name	Description
R	Actions (see page 1468)	This is Actions, a member of class IBaseAdapterStub.
R	CanModify (see page 1458)	This is CanModify, a member of class IAdapterStub.
R	CanView (see page 1458)	This is CanView, a member of class IAdapterStub.
R	DisplayName (see page 1472)	This is DisplayName, a member of class IEndUserStub.
R	Errors (see page 1458)	This is Errors, a member of class IAdapterStub.
R	Fields (see page 1468)	This is Fields, a member of class IBaseAdapterStub.
R	HiddenFields (see page 1458)	This is HiddenFields, a member of class IAdapterStub.
R	HiddenRecordFields (see page 1459)	This is HiddenRecordFields, a member of class IAdapterStub.

R	LoggedIn (see page 1472)	This is LoggedIn, a member of class IEndUserStub.
R	LoginFormAction (see page 1472)	This is LoginFormAction, a member of class IEndUserStub.
R	LogoutAction (see page 1473)	This is LogoutAction, a member of class IEndUserStub.
	Mode (see page 1459)	This is Mode, a member of class IAdapterStub.
R	Records (see page 1469)	This is Records, a member of class IBaseRecordsAdapterStub.

10.2.14.2 IEndUserStub Methods

The methods of the IEndUserStub class are listed here.

Public Methods

	Name	Description
	Get_DisplayName (see page 1471)	This is Get_DisplayName, a member of class IEndUserStub.
	Get_LoggedIn (see page 1471)	This is Get_LoggedIn, a member of class IEndUserStub.
	Get_LoginFormAction (see page 1471)	This is Get_LoginFormAction, a member of class IEndUserStub.
	Get_LogoutAction (see page 1472)	This is Get_LogoutAction, a member of class IEndUserStub.

10.2.14.2.1 IEndUserStub.Get_DisplayName Method

Pascal

```
function Get_DisplayName: string;
```

C++

```
__fastcall AnsiString Get_DisplayName();
```

Description

This is Get_DisplayName, a member of class IEndUserStub.

10.2.14.2.2 IEndUserStub.Get_LoggedIn Method

Pascal

```
function Get_LoggedIn: Boolean;
```

C++

```
__fastcall Boolean Get_LoggedIn();
```

Description

This is Get_LoggedIn, a member of class IEndUserStub.

10.2.14.2.3 IEndUserStub.Get_LoginFormAction Method

Pascal

```
function Get_LoginFormAction: IInterface;
```

C++

```
__fastcall IInterface Get_LoginFormAction();
```

Description

This is Get_LoginFormAction, a member of class IEndUserStub.

10.2.14.2.4 IEndUserStub.Get_LogoutAction Method

Pascal

```
function Get_LogoutAction: IInterface;
```

C++

```
__fastcall IInterface Get_LogoutAction();
```

Description

This is Get_LogoutAction, a member of class IEndUserStub.

10.2.14.3 IEndUserStub Properties

The properties of the IEndUserStub class are listed here.

Public Properties

	Name	Description
R	DisplayName (see page 1472)	This is DisplayName, a member of class IEndUserStub.
R	LoggedIn (see page 1472)	This is LoggedIn, a member of class IEndUserStub.
R	LoginFormAction (see page 1472)	This is LoginFormAction, a member of class IEndUserStub.
R	LogoutAction (see page 1473)	This is LogoutAction, a member of class IEndUserStub.

10.2.14.3.1 IEndUserStub.DisplayName Property

Pascal

```
property DisplayName: string;
```

C++

```
__property AnsiString DisplayName;
```

Description

This is DisplayName, a member of class IEndUserStub.

10.2.14.3.2 IEndUserStub.LoggedIn Property

Pascal

```
property LoggedIn: Boolean;
```

C++

```
__property Boolean LoggedIn;
```

Description

This is LoggedIn, a member of class IEndUserStub.

10.2.14.3.3 IEndUserStub.LoginFormAction Property

Pascal

```
property LoginFormAction: IInterface;
```

C++

```
__property IInterface LoginFormAction;
```

Description

This is LoginFormAction, a member of class IEndUserStub.

10.2.14.3.4 IEndUserStub.LogoutAction Property

Pascal

```
property LogoutAction: IInterface;
```

C++

```
__property IInterface LogoutAction;
```

Description

This is LogoutAction, a member of class IEndUserStub.

10.2.15 IImageProducerStub Interface

Class Hierarchy

AutoAdap.IImageProducerStub

Pascal

```
IImageProducerStub = interface;
```

C++

```
__interface IImageProducerStub;
```

File

AutoAdap

Description

This is class AutoAdap.IImageProducerStub.

10.2.15.1 IImageProducerStub Members

The following tables list the members exposed by IImageProducerStub.

Public Methods

	Name	Description
	GetDisplayStyle (see page 1474)	This is GetDisplayStyle, a member of class IImageProducerStub.
	GetEventImage (see page 1474)	This is GetEventImage, a member of class IImageProducerStub.
	GetImage (see page 1474)	This is GetImage, a member of class IImageProducerStub.

10.2.15.2 IImageProducerStub Methods

The methods of the IImageProducerStub class are listed here.

Public Methods

	Name	Description
	GetDisplayStyle (see page 1474)	This is GetDisplayStyle, a member of class IImageProducerStub.
	GetEventImage (see page 1474)	This is GetEventImage, a member of class IImageProducerStub.
	GetImage (see page 1474)	This is GetImage, a member of class IImageProducerStub.

10.2.15.2.1 IImageProducerStub.GetDisplayStyle Method

Pascal

```
function GetDisplayStyle(const Object_: IUnknown): string;
```

C++

```
__fastcall AnsiString GetDisplayStyle(const IUnknown Object_);
```

Description

This is GetDisplayStyle, a member of class IImageProducerStub.

10.2.15.2.2 IImageProducerStub.GetEventImage Method

Pascal

```
function GetEventImage(const Object_: IUnknown; const Event: string; const Caption: string):
IInterface;
```

C++

```
__fastcall IInterface GetEventImage(const IUnknown Object_, const AnsiString Event, const
AnsiString Caption);
```

Description

This is GetEventImage, a member of class IImageProducerStub.

10.2.15.2.3 IImageProducerStub.GetImage Method

Pascal

```
function GetImage(const Object_: IUnknown; const Caption: string): IInterface;
```

C++

```
__fastcall IInterface GetImage(const IUnknown Object_, const AnsiString Caption);
```

Description

This is GetImage, a member of class IImageProducerStub.

10.3 Types

The following table lists types in this documentation.

Types

Name	Description
TAdapterStubClass (see page 1475)	This is type AutoAdap.TAdapterStubClass.

10.3.1 AutoAdap.TAdapterStubClass Type

Pascal

```
TAdapterStubClass = class of TBaseAdapterStub;
```

C++

```
TBaseAdapterStub TAdapterStubClass;
```

File

AutoAdap

Description

This is type AutoAdap.TAdapterStubClass.

11 AutoDisp Namespace

This is namespace AutoDisp.

Topics

Name	Description
Classes (see page 1478)	The following table lists classes in this documentation.

Classes

	Name	Description
	TWebDispatch (see page 1478)	TWebDispatch describes the HTTP messages to which an auto-dispatching component responds.
	TWebPathInfo (see page 1483)	TWebPathInfo describes the HTTP messages to which an auto-dispatching component responds.

11.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TWebDispatch (see page 1478)	TWebDispatch describes the HTTP messages to which an auto-dispatching component responds.
	TWebPathInfo (see page 1483)	TWebPathInfo describes the HTTP messages to which an auto-dispatching component responds.

11.1.1 TWebDispatch Class

TWebDispatch describes the HTTP messages to which an auto-dispatching component responds.

Class Hierarchy

Pascal

```
TWebDispatch = class(TPersistent);
```

C++

```
class TWebDispatch : public TPersistent;
```

File

AutoDisp

Description

TWebDispatch implements the WebDispatch property of an auto-dispatching component such as TXMLBroker, THTTPSoapDispatcher, or TWSDLHTMLPublish. It specifies the path info portion of the URL for messages directed at the component, the type of messages to which the component can respond, and allows an application to disable the auto-dispatching so that it does not respond directly to HTTP messages from a Web browser.

See Also

TXMLBroker (see page 26049)

THTTPSoapDispatcher (see page 24628)

TWSDLHTMLPublish (see page 26024)

11.1.1.1 TWebDispatch Members

The following tables list the members exposed by TWebDispatch.

Protected Methods

	Name	Description
	AssignTo (see page 1480)	Copies the properties of an object to a destination object.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPathInfo (see page 1481)	This is GetPathInfo, a member of class TWebDispatch.
	SetPathInfo (see page 1481)	This is SetPathInfo, a member of class TWebDispatch.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1480)	Creates and initializes an instance of TWebDispatch (see page 1478).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 1480)	Destroys an instance of TWebDispatch (see page 1478).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Mask (see page 1482)	Provides a mask object for comparing the PathInfo (see page 1483) property with another path.

Published Properties

	Name	Description
	Enabled (see page 1482)	Indicates whether the Web dispatcher passes requests to the object using the TWebDispatch (see page 1478) object.
	MethodType (see page 1483)	This is MethodType, a member of class TWebDispatch.
	PathInfo (see page 1483)	Indicates the path portion of the Uniform Resource Identifier (URI) for which the auto-dispatching component handles requests.

11.1.1.2 TWebDispatch.Create Constructor

Creates and initializes an instance of TWebDispatch (see page 1478).

Pascal

```
constructor Create(AComponent: TComponent);
```

C++

```
__fastcall TWebDispatch(TComponent * AComponent);
```

Description

The TWebDispatch (see page 1478) constructor is called from the constructor of the auto-dispatching component that uses it to implement its WebDispatch property. The AComponent parameter specifies the auto-dispatching component that uses the TWebDispatch (see page 1478) instance.

11.1.1.3 TWebDispatch.Destroy Destructor

Destroys an instance of TWebDispatch (see page 1478).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebDispatch();
```

Description

Do not call Destroy. The TWebDispatch (see page 1478) instance is freed by the auto-dispatching component that uses it.

11.1.1.4 TWebDispatch Methods

The methods of the TWebDispatch class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 1480)	Copies the properties of an object to a destination object.
	GetPathInfo (see page 1481)	This is GetPathInfo, a member of class TWebDispatch.
	SetPathInfo (see page 1481)	This is SetPathInfo, a member of class TWebDispatch.

11.1.1.4.1 TWebDispatch.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should

be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign (see page 2930)

EConvertError (see page 22724)

11.1.1.4.2 TWebDispatch.GetPathInfo Method

Pascal

```
function GetPathInfo: string;
```

C++

```
__fastcall AnsiString GetPathInfo();
```

Description

This is GetPathInfo, a member of class TWebDispatch.

11.1.1.4.3 TWebDispatch.SetPathInfo Method

Pascal

```
procedure SetPathInfo(const Value: string);
```

C++

```
__fastcall SetPathInfo(const AnsiString Value);
```

Description

This is SetPathInfo, a member of class TWebDispatch.

11.1.1.5 TWebDispatch Properties

The properties of the TWebDispatch class are listed here.

Public Properties

	Name	Description
R	Mask (see page 1482)	Provides a mask object for comparing the PathInfo (see page 1483) property with another path.

Published Properties

	Name	Description
	Enabled (see page 1482)	Indicates whether the Web dispatcher passes requests to the object using the TWebDispatch (see page 1478) object.
	MethodType (see page 1483)	This is MethodType, a member of class TWebDispatch.
	PathInfo (see page 1483)	Indicates the path portion of the Uniform Resource Identifier (URI) for which the auto-dispatching component handles requests.

11.1.1.5.1 TWebDispatch.Enabled Property

Indicates whether the Web dispatcher passes requests to the object using the TWebDispatch (see page 1478) object.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Set Enabled to indicate whether the object using the Web dispatcher responds directly to HTTP request messages. By default, Enabled is true, but an application may disable the auto-dispatching of a component because

The application responds to request messages using an action item on the Web dispatcher rather than using the auto-dispatch feature of the component that uses TWebDispatch (see page 1478) as its WebDispatch property. Action items give you greater control over the order in which the dispatcher passes a request message on for handling, and let you embed the default action of the auto-dispatching component with other actions performed by the action item.

The application uses other features of the auto-dispatching component but does not want it to respond to HTTP request messages. For example, a Web client may include an XML broker to provide data packets that appear in the generated HTML pages, but not allow Web browsers to apply updates. When Enabled is false, the Web dispatcher does not pass update messages on to the XML broker.

See Also

TWebActionItem (see page 15130)

Actions (see page 15122)

11.1.1.5.2 TWebDispatch.Mask Property

Provides a mask object for comparing the PathInfo (see page 1483) property with another path.

Pascal

```
property Mask: TMask;
```

C++

```
__property TMask Mask;
```

Description

Use Mask to compare the PathInfo (see page 1483) property with the path portion of a URL. Mask is a TMask object that represents the string specified by PathInfo (see page 1483).

See Also

PathInfo (see page 1483)

11.1.1.5.3 TWebDispatch.MethodType Property

Pascal

```
property MethodType: TMethodType;
```

C++

```
__property TMethodType MethodType;
```

Description

This is MethodType, a member of class TWebDispatch.

11.1.1.5.4 TWebDispatch.PathInfo Property

Indicates the path portion of the Uniform Resource Identifier (URI) for which the auto-dispatching component handles requests.

Pascal

```
property PathInfo: string;
```

C++

```
__property AnsiString PathInfo;
```

Description

Set PathInfo to the path portion of the URI for the target of any request messages serviced by the auto-dispatching component that uses TWebDispatch (see page 1478). The Web dispatcher matches components to request messages by comparing PathInfo with the path portion of the URI specified as the target of the request. If the PathInfo properties match, and the MethodType (see page 1483) matches, the Web dispatcher invokes the auto-dispatching object so that it can respond to the message.

See Also

Mask (see page 1482)

MethodType (see page 1483)

PathInfo (see page 15185)

11.1.2 TWebPathInfo Class

TWebPathInfo describes the HTTP messages to which an auto-dispatching component responds.

Class Hierarchy

AutoDisp.TWebPathInfo

Pascal

```
TWebPathInfo = class;
```

C++

```
class TWebPathInfo;
```

File

AutoDisp

Description

TWebPageInfo implements the PathInfo (see page 1485) property of an auto-dispatching component such as TWebDispatch (see page 1478). It specifies the path info portion of the URL for messages directed at the component.

11.1.2.1 TWebPathInfo Members

The following tables list the members exposed by TWebPathInfo.

Public Methods

	Name	Description
	Create (see page 1484)	This is Create, a member of class TWebPathInfo.
V	Destroy (see page 1484)	This is Destroy, a member of class TWebPathInfo.

Public Properties

	Name	Description
R	Mask (see page 1485)	Provides a mask object for comparing the PathInfo (see page 1485) property with another path.
	PathInfo (see page 1485)	Indicates the path portion of the Uniform Resource Identifier (URI) for which the auto-dispatching component handles requests.

11.1.2.2 TWebPathInfo.Create Constructor

Pascal

```
constructor Create(AOwner: TComponent);
```

C++

```
__fastcall TWebPathInfo(TComponent * AOwner);
```

Description

This is Create, a member of class TWebPathInfo.

11.1.2.3 TWebPathInfo.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebPathInfo();
```

Description

This is Destroy, a member of class TWebPathInfo.

11.1.2.4 TWebPathInfo Properties

The properties of the TWebPathInfo class are listed here.

Public Properties

	Name	Description
R	Mask (see page 1485)	Provides a mask object for comparing the PathInfo (see page 1485) property with another path.
	PathInfo (see page 1485)	Indicates the path portion of the Uniform Resource Identifier (URI) for which the auto-dispatching component handles requests.

11.1.2.4.1 TWebPathInfo.Mask Property

Provides a mask object for comparing the PathInfo (see page 1485) property with another path.

Pascal

```
property Mask: TMask;
```

C++

```
__property TMask Mask;
```

Description

Use Mask to compare the PathInfo (see page 1485) property with the path portion of a URL. Mask is a TMask object that represents the string specified by PathInfo (see page 1485).

See Also

PathInfo (see page 1485)

11.1.2.4.2 TWebPathInfo.PathInfo Property

Indicates the path portion of the Uniform Resource Identifier (URI) for which the auto-dispatching component handles requests.

Pascal

```
property PathInfo: string;
```

C++

```
__property AnsiString PathInfo;
```

Description

Set PathInfo to the path portion of the URI for the target of any request messages serviced by the auto-dispatching component that uses PathInfo. The Web dispatcher matches components to request messages by comparing PathInfo with the path portion of the URI specified as the target of the request. If the PathInfo properties match, and the MethodType matches, the Web dispatcher invokes the auto-dispatching object so that it can respond to the message.

See Also

Mask (see page 1485)

12 AxCtrls Namespace

This is namespace AxCtrls.

Topics

Name	Description
Classes (see page 1489)	The following table lists classes in this documentation.
Interfaces (see page 1702)	The following table lists interfaces in this documentation.
Functions (see page 1709)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 1714)	The following table lists structs, records, enums in this documentation.
Types (see page 1715)	The following table lists types in this documentation.
Constants (see page 1717)	The following table lists constants in this documentation.

Classes

	Name	Description
	TActiveForm (see page 1489)	TActiveForm is the base class for a VCL Form exposed as an ActiveX control. TActiveForm represents a form that is created by a class factory (in Delphi) or using the Active Template Library (C++) and used as an ActiveX control in ActiveX host applications.
	TActiveFormControl (see page 1507)	A COM object which provides the interfaces necessary to support an ActiveX control.
	TActiveFormFactory (see page 1514)	The ActiveFormFactory object is the class factory for TActiveForm (see page 1489).
	TActiveXControl (see page 1517)	TActiveXControl is an abstract class that provides the functionality to embed a VCL control into an ActiveX Container window.
	TActiveXControlFactory (see page 1554)	TActiveXControlFactory is the factory class for ActiveX control objects.
	TActiveXPropertyPage (see page 1561)	TActiveXPropertyPage represents a COM object that provides the interfaces for supporting an ActiveX property page.
	TActiveXPropertyPageFactory (see page 1564)	TActiveXPropertyPageFactory implements a COM factory for an ActiveX property page.
	TAdapterNotifier (see page 1567)	TAdapterNotifier provides the binding between a VCL type object and comparable OLE type object.
	TConnectionPoint (see page 1570)	TConnectionPoint implements the IConnectionPoint interface for a single outgoing interface.
	TConnectionPoints (see page 1575)	TConnectionPoints implements the IConnectionPointContainer interface for a COM object that generates events on clients.
	TCustomAdapter (see page 1578)	TCustomAdapter is a base class for objects that bind VCL objects to COM interfaces.
	TFontAdapter (see page 1583)	TFontAdapter is a helper class used to connect a TFont object to an OLE font object.
	TOleGraphic (see page 1587)	TOleGraphic is a wrapper for an OLE graphical interface.

	TOleStream (see page 1595)	TOleStream reads and writes information over a streaming interface that is provided by an OLE object.
	TPictureAdapter (see page 1599)	TPictureAdapter connects a native TPicture object to an OLE picture object.
	TPropertyPage (see page 1603)	TPropertyPage is the base class for controls that represent an ActiveX control's property page.
	TPropertyPageImpl (see page 1679)	TPropertyPageImpl implements the IPropertyPage interface to provide the interfaces needed to support an ActiveX property page.
	TReflectorWindow (see page 1684)	TReflectorWindow ensures that ActiveX controls respond properly to Windows messages.
	TStringsAdapter (see page 1696)	TStringsAdapter is a helper class used to allow a TStrings object to implement an IStrings interface for use with OLE.

Constants

Name	Description
Class_DColorPropPage (see page 1717)	This is constant AxCtrls.Class_DColorPropPage.
Class_DFontPropPage (see page 1717)	This is constant AxCtrls.Class_DFontPropPage.
Class_DPicturePropPage (see page 1718)	This is constant AxCtrls.Class_DPicturePropPage.
Class_DStringPropPage (see page 1718)	This is constant AxCtrls.Class_DStringPropPage.

Functions

	Name	Description
	EnumDispatchProperties (see page 1709)	Fills a TStringList will all the property names and DispIDs in a specified IDispatch interface.
	GetOleFont (see page 1710)	Creates an OLE font object that is directly mapped to a native TFont.
	GetOlePicture (see page 1710)	Creates an OLE picture object that is directly mapped to a native TPicture.
	GetOleStrings (see page 1711)	Implements a TStrings object as an IStrings interface for use by OLE objects.
	ParkingWindow (see page 1711)	Provides a temporary parent window for ActiveX controls when the container is not ready to act as a parent.
	SetOleFont (see page 1712)	Connects an OLE font object to a TFont object and copies the properties of the OLE font object to the TFont.
	SetOlePicture (see page 1712)	Connects an OLE picture object to a TPicture object, and copies the properties of the OLE picture object to the TPicture.
	SetOleStrings (see page 1713)	Uses an IStrings interface to assign the contents of a TStrings object.

Interfaces

	Name	Description
	IAmbientDispatch (see page 1702)	This is class AxCtrls.IAmbientDispatch.
	IFontAccess (see page 1706)	This is class AxCtrls.IFontAccess.
	IPictureAccess (see page 1707)	This is class AxCtrls.IPictureAccess.

Structs, Records, Enums

	Name	Description
	TConnectionKind (see page 1714)	This is record AxCtrls.TConnectionKind.

Types

Name	Description
TActiveFormClass (see page 1715)	This is type AxCtrls.TActiveFormClass.
TActiveXControlClass (see page 1715)	This is type AxCtrls.TActiveXControlClass.
TDefinePropertyPage (see page 1716)	This is type AxCtrls.TDefinePropertyPage.
TPropertyPageClass (see page 1716)	This is type AxCtrls.TPropertyPageClass.

12.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TActiveForm (see page 1489)	TActiveForm is the base class for a VCL Form exposed as an ActiveX control. TActiveForm represents a form that is created by a class factory (in Delphi) or using the Active Template Library (C++) and used as an ActiveX control in ActiveX host applications.
	TActiveFormControl (see page 1507)	A COM object which provides the interfaces necessary to support an ActiveX control.
	TActiveFormFactory (see page 1514)	The ActiveFormFactory object is the class factory for TActiveForm (see page 1489).
	TActiveXControl (see page 1517)	TActiveXControl is an abstract class that provides the functionality to embed a VCL control into an ActiveX Container window.
	TActiveXControlFactory (see page 1554)	TActiveXControlFactory is the factory class for ActiveX control objects.
	TActiveXPropertyPage (see page 1561)	TActiveXPropertyPage represents a COM object that provides the interfaces for supporting an ActiveX property page.
	TActiveXPropertyPageFactory (see page 1564)	TActiveXPropertyPageFactory implements a COM factory for an ActiveX property page.
	TAdapterNotifier (see page 1567)	TAdapterNotifier provides the binding between a VCL type object and comparable OLE type object.
	TConnectionPoint (see page 1570)	TConnectionPoint implements the IConnectionPoint interface for a single outgoing interface.
	TConnectionPoints (see page 1575)	TConnectionPoints implements the IConnectionPointContainer interface for a COM object that generates events on clients.
	TCustomAdapter (see page 1578)	TCustomAdapter is a base class for objects that bind VCL objects to COM interfaces.
	TFontAdapter (see page 1583)	TFontAdapter is a helper class used to connect a TFont object to an OLE font object.
	TOleGraphic (see page 1587)	TOleGraphic is a wrapper for an OLE graphical interface.
	TOleStream (see page 1595)	TOleStream reads and writes information over a streaming interface that is provided by an OLE object.
	TPictureAdapter (see page 1599)	TPictureAdapter connects a native TPicture object to an OLE picture object.
	TPropertyPage (see page 1603)	TPropertyPage is the base class for controls that represent an ActiveX control's property page.
	TPropertyPageImpl (see page 1679)	TPropertyPageImpl implements the IPropertyPage interface to provide the interfaces needed to support an ActiveX property page.
	TReflectorWindow (see page 1684)	TReflectorWindow ensures that ActiveX controls respond properly to Windows messages.
	TStringsAdapter (see page 1696)	TStringsAdapter is a helper class used to allow a TStrings object to implement an IStrings interface for use with OLE.

12.1.1 TActiveForm Class

TActiveForm is the base class for a VCL Form exposed as an ActiveX control. TActiveForm represents a form that is created by a class factory (in Delphi) or using the Active Template Library (C++) and used as an ActiveX control in ActiveX host applications.

Class Hierarchy

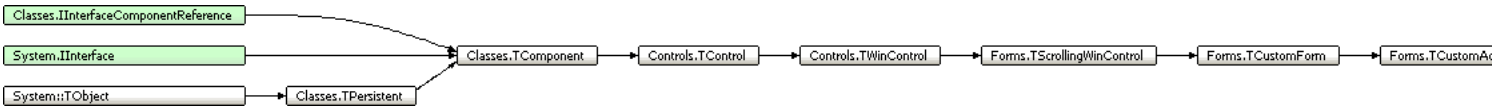

Pascal

```
TActiveForm = class(TCustomActiveForm);
```

C++

```
class TActiveForm : public TCustomActiveForm;
```

File

AxCtrls

Description

A TActiveForm object is used to create a form ActiveX control that can be displayed and run from within Web browsers.

Note: In Delphi applications, this object is created by TActiveFormControl (see page 1507). It is a descendant of this class that you design in the Form Designer when you create an ActiveForm.

See Also

TActiveFormControl (see page 1507)

TActiveFormFactory (see page 1514)

Tform

TPropertyPage (see page 1603)

Elements of an ActiveX Control

Generating an ActiveX Control Based On a VCL Form

12.1.1.1 TActiveForm Members

The following tables list the members exposed by TActiveForm.

Protected Events

	Name	Description
	OnActivate (see page 13943)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnCanResize (see page 13947)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnClick (see page 13947)	Occurs when the user clicks the control.
	OnClose (see page 13949)	Occurs when the form closes.
	OnCloseQuery (see page 13950)	Occurs when close is attempted.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 13951)	Occurs when the form is created.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

	OnDblClick (see page 13953)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 13953)	Occurs when the form loses focus.
	OnDestroy (see page 13953)	Occurs when the form is destroyed.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragDrop (see page 13954)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDragOver (see page 13957)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnHelp (see page 13959)	Occurs when the form receives a request for help.
	OnHide (see page 13960)	Occurs when the form is hidden (that is, when its Visible (see page 13940) property is set to false).
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyDown (see page 13962)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyPress (see page 13964)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnKeyUp (see page 13966)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseActivate (see page 13967)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseDown (see page 13968)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseMove (see page 13971)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseUp (see page 13973)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnPaint (see page 13975)	Occurs when the form is redrawn.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnResize (see page 13976)	Occurs immediately after the control is resized.
	OnShortCut (see page 13977)	Occurs when the user presses a key (before the OnKeyDown (see page 13962) event).
	OnShow (see page 13977)	Occurs when the form is shown (that is, when its Visible (see page 13940) property is set to true).
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFormState (see page 13863)	This is FFormState, a member of class TCustomForm.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	Activate (see page 13865)	Activation event dispatcher.
	ActiveChanged (see page 13865)	Active (see page 13896) state event dispatcher.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 14324)	Adjusts the ClientRect property to exclude the scroll bars.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 13866)	Aligns all child controls of the form that fall within a specified rectangle.
V	AlignControls (see page 14325)	Aligns the controls in the scrolling windowed control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
V	AutoScrollEnabled (see page 14325)	Indicates whether automatic scrolling is enabled.
V	AutoScrollInView (see page 14325)	Scrolls a control into the visible area of the scrolling windowed control.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
V	BeginAutoDrag (see page 13867)	Responds when the user starts dragging the form if DragMode is dmAutomatic.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.

	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	ChangeScale (see page 13867)	Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.
	ChangeScale (see page 14326)	Resizes the scrolling windowed control by a specific ratio.
	Click (see page 5743)	Respond to user click.
	CloseModal (see page 13868)	This is CloseModal, a member of class TCustomForm.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 13773)	Updates the active form's creation parameters according to the value of the AxBorderStyle (see page 13777) property.
	CreateParams (see page 13868)	Initializes the window-creation parameter record when the form window is created.
	CreateParams (see page 14326)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWindowHandle (see page 13869)	Creates a form window given a set of window-creation parameters.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 13869)	Creates the form window.
	CreateWnd (see page 14327)	Creates a Windows control corresponding to the TScrollingWinControl (see page 14312) object.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	Deactivate (see page 13870)	Generates an OnDeactivate (see page 13953) event.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DefineProperties (see page 13871)	Designates methods for storing a form's unpublished data on a stream such as a form file.
	DefinePropertyPages (see page 1505)	This is DefinePropertyPages, a member of class TActiveForm.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyHandle (see page 13871)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWindowHandle (see page 13872)	Destroys the window created in the CreateWindowHandle (see page 13869) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoClose (see page 13873)	Form closing event dispatcher.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoCreate (see page 13873)	Form creation event dispatcher.
	DoDestroy (see page 1505)	Form destruction event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.

	DoDock (see page 13874)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoFlipChildren (see page 14327)	Flips the control's children.
	DoHide (see page 13874)	Form hide event dispatcher.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoShow (see page 13875)	Form show event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	EventSinkChanged (see page 1506)	This is EventSinkChanged, a member of class TActiveForm.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorderIconStyles (see page 13875)	This is GetBorderIconStyles, a member of class TCustomForm.
	GetBorderStyles (see page 13876)	This is GetBorderStyles, a member of class TCustomForm.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetChildren (see page 13876)	Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetClientRect (see page 13876)	Returns the value of the ClientRect property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.

	GetFloating (see page 13877)	Returns the value of the Floating property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetOwnerWindow (see page 13878)	This is GetOwnerWindow, a member of class TCustomForm.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 13878)	Handles exceptions that are raised by the OnCreate (see page 13951) event handler.
	Initialize (see page 1506)	This is Initialize, a member of class TActiveForm.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	Loaded (see page 13880)	Performs fix-ups when the form is first loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	Notification (see page 13882)	Responds to notifications that objects are about to be inserted or removed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 13882)	Generates an OnPaint (see page 13975) event.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaintWindow (see page 13882)	Renders the image of the form.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteChanged (see page 13883)	Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.

	QueryInterface (see page 13884)	Returns a reference to a specified interface if that interface is supported by the form.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	ReadState (see page 13884)	Reads the form's published properties, stored data, and owned components from a reader object.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	RequestAlign (see page 13885)	Instructs the form's parent (if any) to reposition the form, enforcing its Align property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	Resizing (see page 13885)	Makes any required adjustments when the form changes size.
	Resizing (see page 14328)	This is Resizing, a member of class TScrollingWinControl.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetChildOrder (see page 13886)	Changes the order in which Child appears when GetChildren (see page 13876) is called.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParent (see page 13887)	Sets the Parent property.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentBiDiMode (see page 13887)	Sets the ParentBiDiMode property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.

V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
V	UpdateActions (see page 13891)	Updates all actions associated with the form.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	UpdateWindowState (see page 13892)	Updates the WindowState (see page 13942) property to reflect the current state of the form.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	ValidateRename (see page 13892)	Determines if a form can rename one of its owned components.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	VisibleChanging (see page 13893)	Responds just before the form changes the value of its Visible (see page 13940) property.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WndProc (see page 13893)	Provides specific message responses for the form.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
R	ActiveMDIChild (see page 13897)	Specifies the MDI child that has focus.
	AlphaBlend (see page 13898)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 13898)	Specifies the degree of translucency on a translucent form.
	AutoScroll (see page 13899)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoScroll (see page 14329)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderIcons (see page 13899)	Specifies which icons appear on the title bar of the form.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
R	ClientHandle (see page 13904)	Provides access to the handle of the form's internal MDI client window.
W	ClientHeight (see page 13905)	Specifies the height (in pixels) of the form's client area.
W	ClientWidth (see page 13905)	Specifies the width (in pixels) of the form's client area.
	Color (see page 5812)	Specifies the background color of the control.

	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Ctl3D (see page 13909)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultMonitor (see page 13909)	Specifies the monitor on which the form appears.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	FormStyle (see page 13912)	Determines the form's style.
	Height (see page 13915)	Specifies the vertical size of the control in pixels.
	HorzScrollBar (see page 13916)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 13917)	Specifies the icon that appears when the form is minimized.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	MDIChildCount (see page 13921)	Specifies the number of open MDI child forms.
R	MDIChildren (see page 13922)	Provides indexed access to all MDI child forms.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ObjectMenuItem (see page 13926)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 13926)	Specifies when OnCreate (see page 13951) and OnDestroy (see page 13953) events occur.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentFont (see page 13927)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PixelsPerInch (see page 13928)	Represents the proportion of the font on the system on which the form was designed.
R	PopupChildren (see page 13928)	This is PopupChildren, a member of class TCustomForm.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	PopupMenu (see page 13928)	Identifies the pop-up menu associated with the control.
	Position (see page 13933)	Represents the size and placement of the form.
	PrintScale (see page 13934)	Represents the proportions of a printed form.
	Scaled (see page 13935)	Specifies whether the form is sized according to the value of the PixelsPerInch (see page 13928) property.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	TileMode (see page 13936)	Represents how MDI child forms are arranged when the Tile method is called.
	TransparentColor (see page 13938)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 13939)	Indicates the color on the form that appears transparent when TransparentColor (see page 13938) is true.
	VertScrollBar (see page 13939)	Represents the vertical scroll bar for the scrolling windowed control.
W	Visible (see page 13940)	Indicates whether the form is visible.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	Width (see page 13941)	Specifies the horizontal size of the control or form in pixels.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowMenu (see page 13941)	Specifies the Window menu for an MDI parent form.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 13866)	Construction event dispatcher.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 13866)	Destruction event dispatcher.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Close (see page 13867)	Closes the form.
V	CloseQuery (see page 13868)	Close (see page 13867) attempt event dispatcher.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 13773)	Creates a new TCustomActiveForm (see page 13759) object.
V	Create (see page 13861)	Creates and initializes a new TForm (see page 13992) object.
V	Create (see page 14323)	Creates and initializes a new TScrollingWinControl (see page 14312) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	CreateNew (see page 13862)	Creates and initializes a new form.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 13870)	Provides message handling for all messages that the form does not fully process.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DefocusControl (see page 13871)	Removes focus from a control on the form.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 13862)	Removes the form from memory.
V	Destroy (see page 14323)	Removes the TScrollingWinControl (see page 14312) object from memory.
V	Destroy (see page 22287)	Disposes of an object instance.

	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	DisableAutoRange (see page 14327)	Disables automatic scrolling.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
V	Dock (see page 13872)	Handles automatic docking of the form into other windows.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EnableAutoRange (see page 14328)	Reenables automatic scrolling.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	FocusControl (see page 13875)	Sets focus to a control on the form.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFormImage (see page 13877)	Returns a bitmap of the form.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	Hide (see page 13879)	Hides the form.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	IsShortCut (see page 13880)	Processes shortcut keys when the form has focus.
	MakeFullyVisible (see page 13881)	Ensures that the form is fully visible on a specified monitor.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	MouseWheelHandler (see page 13881)	Directs mouse wheel messages to the appropriate control.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	Print (see page 13883)	Prints the form.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	RecreateAsPopup (see page 13884)	This is RecreateAsPopup, a member of class TCustomForm.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	Release (see page 13885)	Destroys the form and frees its associated memory.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollInView (see page 14328)	Scrolls a control into the visible area of the scrolling windowed control.
	SendCancelMode (see page 13886)	Cancels modes on the form.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetDesignVisible (see page 6178)	Makes a control visible at design time.
	SetFocus (see page 6179)	Gives the input focus to the control.

V	SetFocus (see page 13886)	Sets focus to the form.
V	SetFocusedControl (see page 13887)	Sets focus to a control on the form.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Show (see page 13888)	Shows the form.
V	ShowModal (see page 13890)	Shows a form as a modal dialog.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UpdateGlassFrame (see page 13892)	This is UpdateGlassFrame, a member of class TCustomForm.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.
V	WantChildKey (see page 13774)	Indicates whether the form processes keyboard input for a control it owns.
V	WantChildKey (see page 13893)	Indicates whether the form processes keyboard input for a control it owns.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Action (see page 13895)	Designates the action associated with the control.
R	Active (see page 13896)	Specifies whether the form has focus.
	ActiveControl (see page 13896)	Specifies the control that has focus on the form.
R	ActiveFormControl (see page 1506)	Pointer to the ActiveFormControl object.
	ActiveOleControl (see page 13897)	Specifies the OLE control on the form that reacts to changes in focus.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 13900)	Specifies the appearance and behavior of the form border.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 13900)	Provides access to the drawing area of the form.
	Caption (see page 13902)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 13908)	Specifies the background color of the control.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designer (see page 13910)	Specifies the designer interface for the form.

	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropTarget (see page 13910)	Specifies whether the form is the target of a drag-and-drop operation.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Font (see page 13910)	Controls the attributes of text written on or in the control.
R	FormState (see page 13911)	Indicates transitional state information about the form.
	GlassFrame (see page 13914)	This is GlassFrame, a member of class TCustomForm.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HelpFile (see page 13916)	Specifies the name of the file the form uses to display Help.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	KeyPreview (see page 13918)	Specifies whether the form should receive keyboard events before the active control.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Menu (see page 13923)	Specifies the form's main menu.
	ModalResult (see page 13924)	Represents the return value of a form that is used as a modal dialog.
R	Monitor (see page 13926)	Provides access to the monitor on which the form appears.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	OleFormObject (see page 13927)	Specifies the IOleForm (see page 14340) interface for an in-place OLE object contained in the form.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PopupMode (see page 13933)	This is PopupMode, a member of class TCustomForm.
	PopupParent (see page 13933)	Sets an order for stacked forms that users cannot change.
	ScreenSnap (see page 13935)	Specifies whether form snaps to edge of screen.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	SnapBuffer (see page 13936)	Specifies distance for screen snap.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
	Visible (see page 13797)	Indicates whether the form is visible.

R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.
	WindowState (see page 13942)	Represents how the form appears on the screen.

Published Events

	Name	Description
	OnActivate (see page 13799)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnClick (see page 13803)	Occurs when the user clicks the control.
	OnContextPopup (see page 13805)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 13805)	Occurs when the form is created.
	OnDblClick (see page 13807)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 13807)	Occurs when the form loses focus.
	OnDestroy (see page 13808)	Occurs when the form is destroyed.
	OnDragDrop (see page 13809)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13811)	Occurs when the user drags an object over a control.
	OnKeyDown (see page 13814)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 13816)	Occurs when a key is pressed.
	OnKeyUp (see page 13817)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 13819)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 13820)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 13822)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 13823)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 13823)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 13825)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 13827)	Occurs when the form is redrawn.

Published Properties

	Name	Description
	ActiveControl (see page 13775)	Specifies the control that has focus on the form.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 13776)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 13777)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 13777)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AxBorderStyle (see page 13777)	Sets the active form's border style.
	BorderWidth (see page 13778)	Specifies the width of the control's border.
	Caption (see page 13778)	Specifies a text string that identifies the control to the user.
	Color (see page 13781)	Specifies the background color of the control.
	Constraints (see page 13782)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Font (see page 13782)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	Height (see page 13783)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.

	HorzScrollBar (see page 13784)	Represents the horizontal scroll bar for the scrolling windowed control.
	HorzScrollBar (see page 14329)	Represents the horizontal scroll bar for the scrolling windowed control.
	KeyPreview (see page 13785)	Specifies whether the form should receive keyboard events before the active control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Left (see page 13920)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 13788)	Specifies when OnCreate and OnDestroy events occur.
	Padding (see page 13789)	Specifies the padding of a control.
	PixelsPerInch (see page 13789)	Represents the proportion of the font on the system on which the form was designed.
	PopupMenu (see page 13789)	Identifies the pop-up menu associated with the control.
	PrintScale (see page 13794)	Represents the proportions of a printed form.
	Scaled (see page 13794)	Specifies whether the form is sized according to the value of the PixelsPerInch property.
	ShowHint (see page 13795)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Top (see page 13937)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	VertScrollBar (see page 13796)	Represents the vertical scroll bar for the scrolling windowed control.
	VertScrollBar (see page 14330)	Represents the vertical scroll bar for the scrolling windowed control.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	Width (see page 13798)	Specifies the horizontal size of the control or form in pixels.

12.1.1.2 TActiveForm Methods

The methods of the TActiveForm class are listed here.

Protected Methods

	Name	Description
	DefinePropertyPages (see page 1505)	This is DefinePropertyPages, a member of class TActiveForm.
	DoDestroy (see page 1505)	Form destruction event dispatcher.
	EventSinkChanged (see page 1506)	This is EventSinkChanged, a member of class TActiveForm.
	Initialize (see page 1506)	This is Initialize, a member of class TActiveForm.

12.1.1.2.1 TActiveForm.DefinePropertyPages Method

Pascal

```
procedure DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage); virtual;
```

C++

```
virtual __fastcall DefinePropertyPages(TDefinePropertyPage DefinePropertyPage);
```

Description

This is DefinePropertyPages, a member of class TActiveForm.

12.1.1.2.2 TActiveForm.DoDestroy Method

Form destruction event dispatcher.

Pascal

```
procedure DoDestroy; override;
```

C++

```
virtual __fastcall DoDestroy();
```

Description

DoDestroy is called automatically when the TCustomForm object is about to be deallocated. The precise way DoDestroy is called can be modified by setting the OldCreateOrder property.

As implemented in TCustomForm, DoDestroy simply calls the OnDestroy event handler.

See Also

OnDestroy (see page 13953)

OldCreateOrder (see page 13926)

12.1.1.2.3 TActiveForm.EventSinkChanged Method

Pascal

```
procedure EventSinkChanged(const EventSink: IUnknown); virtual;
```

C++

```
virtual __fastcall EventSinkChanged(const IUnknown EventSink);
```

Description

This is EventSinkChanged, a member of class TActiveForm.

12.1.1.2.4 TActiveForm.Initialize Method

Pascal

```
procedure Initialize; virtual;
```

C++

```
virtual __fastcall Initialize();
```

Description

This is Initialize, a member of class TActiveForm.

12.1.1.3 TActiveForm Properties

The properties of the TActiveForm class are listed here.

Public Properties

	Name	Description
R	ActiveFormControl (see page 1506)	Pointer to the ActiveFormControl object.

12.1.1.3.1 TActiveForm.ActiveFormControl Property

Pointer to the ActiveFormControl object.

Pascal

```
property ActiveFormControl: TActiveFormControl;
```

C++

```
__property TActiveFormControl ActiveFormControl;
```

Description

Use ActiveFormControl to access the object that implements all the ActiveX interfaces for this ActiveForm. ActiveFormControl creates the TActiveForm (see page 1489) object.

12.1.2 TActiveFormControl Class

A COM object which provides the interfaces necessary to support an ActiveX control.

Class Hierarchy

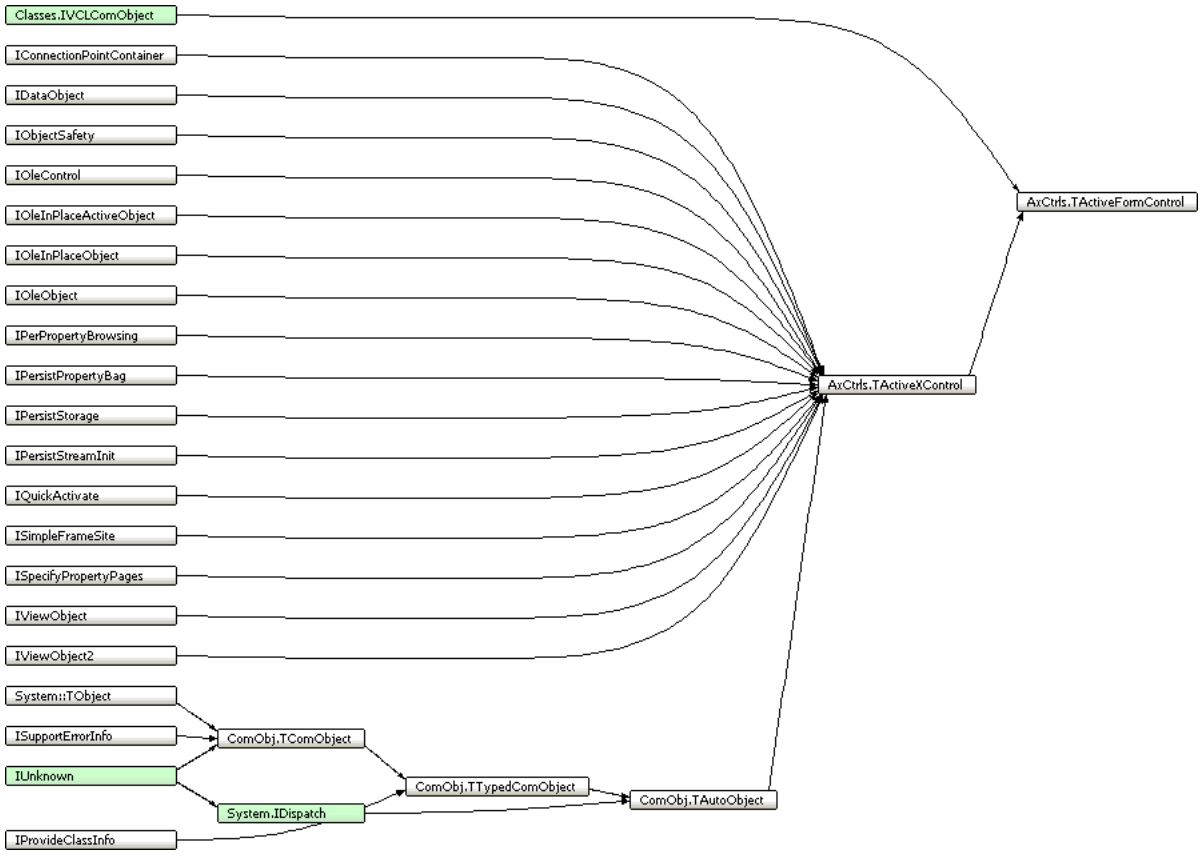

Pascal

```
TActiveFormControl = class(TActiveXControl, IVCLComObject);
```

C++

```
class TActiveFormControl : public TActiveXControl, public IVCLComObject;
```

File

AxCtrls

Description

TActiveFormControl acts as the bridge between a Delphi C++Builder form and the ActiveX control API. It is a descendant of TActiveXControl (see page 1517) which implements the necessary interfaces for an ActiveX control. It forwards the appropriate information to the ActiveForm.

This object is used internally by the DAX framework. Normally, you would not create an instance of this class.

See Also

TActiveForm (see page 1489)

Elements of an ActiveX Control

Generating an ActiveX Control Based On a VCL Form

12.1.2.1 TActiveFormControl Members

The following tables list the members exposed by TActiveFormControl.

Protected Methods

	Name	Description
	_AddRef (see page 5558)	This is _AddRef, a member of class TComObject.
	_Release (see page 5558)	This is _Release, a member of class TComObject.
	Advise (see page 1525)	This is Advise, a member of class TActiveXControl.
	Close (see page 1525)	This is Close, a member of class TActiveXControl.
	ContextSensitiveHelp (see page 1525)	This is ContextSensitiveHelp, a member of class TActiveXControl.
	DAdvise (see page 1525)	This is DAdvise, a member of class TActiveXControl.
V	DefinePropertyPages (see page 1512)	This is DefinePropertyPages, a member of class TActiveFormControl.
	DoVerb (see page 1526)	This is DoVerb, a member of class TActiveXControl.
	Draw (see page 1526)	This is Draw, a member of class TActiveXControl.
	DUnadvise (see page 1526)	This is DUnadvise, a member of class TActiveXControl.
	EnableModeless (see page 1527)	This is EnableModeless, a member of class TActiveXControl.
	EnumAdvise (see page 1527)	This is EnumAdvise, a member of class TActiveXControl.
	EnumDAdvise (see page 1527)	This is EnumDAdvise, a member of class TActiveXControl.
	EnumFormatEtc (see page 1527)	This is EnumFormatEtc, a member of class TActiveXControl.
	EnumVerbs (see page 1527)	This is EnumVerbs, a member of class TActiveXControl.
	EventConnect (see page 5542)	This is EventConnect, a member of class TAutoObject.
V	EventSinkChanged (see page 1512)	This is EventSinkChanged, a member of class TActiveFormControl.
	Freeze (see page 1528)	This is Freeze, a member of class TActiveXControl.
	FreezeEvents (see page 1528)	This is FreezeEvents, a member of class TActiveXControl.
	Get_BackColor (see page 1528)	This is Get_BackColor, a member of class TActiveXControl.
	Get_Caption (see page 1528)	This is Get_Caption, a member of class TActiveXControl.
	Get_Enabled (see page 1529)	This is Get_Enabled, a member of class TActiveXControl.
	Get_Font (see page 1529)	This is Get_Font, a member of class TActiveXControl.
	Get_ForeColor (see page 1529)	This is Get_ForeColor, a member of class TActiveXControl.
	Get_HWnd (see page 1529)	This is Get_HWnd, a member of class TActiveXControl.
	Get_TabStop (see page 1529)	This is Get_TabStop, a member of class TActiveXControl.
	Get_Text (see page 1530)	This is Get_Text, a member of class TActiveXControl.
	GetAdvise (see page 1530)	This is GetAdvise, a member of class TActiveXControl.
	GetCanonicalFormatEtc (see page 1530)	This is GetCanonicalFormatEtc, a member of class TActiveXControl.
	GetClassID (see page 1530)	This is GetClassID, a member of class TActiveXControl.
	GetClassInfo (see page 5583)	Returns a pointer to a TTypedComObject (see page 5581)'s type information.
	GetClientSite (see page 1531)	This is GetClientSite, a member of class TActiveXControl.
	GetClipboardData (see page 1531)	This is GetClipboardData, a member of class TActiveXControl.

	GetColorSet (see page 1531)	This is GetColorSet, a member of class TActiveXControl.
	GetContentExtent (see page 1531)	This is GetContentExtent, a member of class TActiveXControl.
	GetControlInfo (see page 1531)	This is GetControlInfo, a member of class TActiveXControl.
	GetData (see page 1532)	This is GetData, a member of class TActiveXControl.
	GetDataHere (see page 1532)	This is GetDataHere, a member of class TActiveXControl.
	GetDisplayString (see page 1532)	This is GetDisplayString, a member of class TActiveXControl.
	GetExtent (see page 1532)	This is GetExtent, a member of class TActiveXControl.
V	GetIDsOfNames (see page 5543)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetInterfaceSafetyOptions (see page 1533)	This is GetInterfaceSafetyOptions, a member of class TActiveXControl.
	GetMiscStatus (see page 1533)	This is GetMiscStatus, a member of class TActiveXControl.
	GetMoniker (see page 1533)	This is GetMoniker, a member of class TActiveXControl.
	GetPages (see page 1533)	This is GetPages, a member of class TActiveXControl.
	GetPredefinedStrings (see page 1533)	This is GetPredefinedStrings, a member of class TActiveXControl.
	GetPredefinedValue (see page 1534)	This is GetPredefinedValue, a member of class TActiveXControl.
V	GetPropertyString (see page 1534)	This is GetPropertyString, a member of class TActiveXControl.
V	GetPropertyStrings (see page 1534)	This is GetPropertyStrings, a member of class TActiveXControl.
V	GetPropertyValue (see page 1534)	This is GetPropertyValue, a member of class TActiveXControl.
V	GetPropFromBag (see page 1535)	This is GetPropFromBag, a member of class TActiveXControl.
	GetSizeMax (see page 1535)	This is GetSizeMax, a member of class TActiveXControl.
V	GetTypeInfo (see page 5543)	Retrieves the type information for an object.
V	GetTypeInfoCount (see page 5544)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetUserClassID (see page 1535)	This is GetUserClassID, a member of class TActiveXControl.
	GetUserType (see page 1535)	This is GetUserType, a member of class TActiveXControl.
	GetWindow (see page 1536)	This is GetWindow, a member of class TActiveXControl.
	HandsOffStorage (see page 1536)	This is HandsOffStorage, a member of class TActiveXControl.
	InitFromData (see page 1536)	This is InitFromData, a member of class TActiveXControl.
	InitNew (see page 1537)	This is InitNew, a member of class TActiveXControl.
	InPlaceActivate (see page 1537)	This is InPlaceActivate, a member of class TActiveXControl.
	InPlaceDeactivate (see page 1537)	This is InPlaceDeactivate, a member of class TActiveXControl.
	InterfaceSupportsErrorInfo (see page 5559)	This is InterfaceSupportsErrorInfo, a member of class TComObject.
	IsDirty (see page 1537)	This is IsDirty, a member of class TActiveXControl.
	IsUpToDate (see page 1538)	This is IsUpToDate, a member of class TActiveXControl.
V	LoadFromStream (see page 1538)	This is LoadFromStream, a member of class TActiveXControl.
	MapPropertyToPage (see page 1538)	This is MapPropertyToPage, a member of class TActiveXControl.
	OnAmbientPropertyChange (see page 1539)	This is OnAmbientPropertyChange, a member of class TActiveXControl.
	OnDocWindowActivate (see page 1539)	This is OnDocWindowActivate, a member of class TActiveXControl.
	OnFrameWindowActivate (see page 1539)	This is OnFrameWindowActivate, a member of class TActiveXControl.
	OnMnemonic (see page 1539)	This is OnMnemonic, a member of class TActiveXControl.
V	PerformVerb (see page 1540)	This is PerformVerb, a member of class TActiveXControl.
	PersistPropBagInitNew (see page 1540)	This is PersistPropBagInitNew, a member of class TActiveXControl.
	PersistPropBagLoad (see page 1540)	This is PersistPropBagLoad, a member of class TActiveXControl.
	PersistPropBagSave (see page 1540)	This is PersistPropBagSave, a member of class TActiveXControl.
	PersistStorageInitNew (see page 1541)	This is PersistStorageInitNew, a member of class TActiveXControl.
	PersistStorageLoad (see page 1541)	This is PersistStorageLoad, a member of class TActiveXControl.
	PersistStorageSave (see page 1541)	This is PersistStorageSave, a member of class TActiveXControl.
	PersistStreamLoad (see page 1541)	This is PersistStreamLoad, a member of class TActiveXControl.
	PersistStreamSave (see page 1541)	This is PersistStreamSave, a member of class TActiveXControl.
	PostMessageFilter (see page 1542)	This is PostMessageFilter, a member of class TActiveXControl.
	PreMessageFilter (see page 1542)	This is PreMessageFilter, a member of class TActiveXControl.
V	PutPropInBag (see page 1543)	This is PutPropInBag, a member of class TActiveXControl.
	QueryGetData (see page 1544)	This is QueryGetData, a member of class TActiveXControl.
	QueryInterface (see page 5560)	This is QueryInterface, a member of class TComObject.
	QuickActivate (see page 1544)	This is QuickActivate, a member of class TActiveXControl.

	ReactivateAndUndo (see page 1544)	This is ReactivateAndUndo, a member of class TActiveXControl.
	ResizeBorder (see page 1544)	This is ResizeBorder, a member of class TActiveXControl.
	SaveCompleted (see page 1544)	This is SaveCompleted, a member of class TActiveXControl.
V	SaveToStream (see page 1545)	This is SaveToStream, a member of class TActiveXControl.
	Set_BackColor (see page 1545)	This is Set_BackColor, a member of class TActiveXControl.
	Set_Caption (see page 1545)	This is Set_Caption, a member of class TActiveXControl.
	Set_Enabled (see page 1545)	This is Set_Enabled, a member of class TActiveXControl.
	Set_Font (see page 1546)	This is Set_Font, a member of class TActiveXControl.
	Set_ForeColor (see page 1546)	This is Set_ForeColor, a member of class TActiveXControl.
	Set_TabStop (see page 1546)	This is Set_TabStop, a member of class TActiveXControl.
	Set_Text (see page 1546)	This is Set_Text, a member of class TActiveXControl.
	SetAdvise (see page 1546)	This is SetAdvise, a member of class TActiveXControl.
	SetClientSite (see page 1547)	This is SetClientSite, a member of class TActiveXControl.
	SetColorScheme (see page 1547)	This is SetColorScheme, a member of class TActiveXControl.
	SetContentExtent (see page 1547)	This is SetContentExtent, a member of class TActiveXControl.
	SetData (see page 1547)	This is SetData, a member of class TActiveXControl.
	SetExtent (see page 1548)	This is SetExtent, a member of class TActiveXControl.
	SetHostNames (see page 1548)	This is SetHostNames, a member of class TActiveXControl.
	SetInPlaceSite (see page 1548)	This is SetInPlaceSite, a member of class TActiveXControl.
V	SetInterfaceSafetyOptions (see page 1548)	This is SetInterfaceSafetyOptions, a member of class TActiveXControl.
	SetMoniker (see page 1548)	This is SetMoniker, a member of class TActiveXControl.
	SetObjectRects (see page 1549)	This is SetObjectRects, a member of class TActiveXControl.
	ShowPropertyDialog (see page 1549)	This is ShowPropertyDialog, a member of class TActiveXControl.
	StdClickEvent (see page 1549)	This is StdClickEvent, a member of class TActiveXControl.
	StdDblClickEvent (see page 1549)	This is StdDblClickEvent, a member of class TActiveXControl.
	StdKeyDownEvent (see page 1550)	This is StdKeyDownEvent, a member of class TActiveXControl.
	StdKeyPressEvent (see page 1550)	This is StdKeyPressEvent, a member of class TActiveXControl.
	StdKeyUpEvent (see page 1550)	This is StdKeyUpEvent, a member of class TActiveXControl.
	StdMouseDownEvent (see page 1550)	This is StdMouseDownEvent, a member of class TActiveXControl.
	StdMouseMoveEvent (see page 1550)	This is StdMouseMoveEvent, a member of class TActiveXControl.
	StdMouseUpEvent (see page 1551)	This is StdMouseUpEvent, a member of class TActiveXControl.
	TranslateAccelerator (see page 1551)	This is TranslateAccelerator, a member of class TActiveXControl.
	UIDeactivate (see page 1551)	This is UIDeactivate, a member of class TActiveXControl.
	Unadvise (see page 1551)	This is Unadvise, a member of class TActiveXControl.
	Unfreeze (see page 1552)	This is Unfreeze, a member of class TActiveXControl.
	Update (see page 1552)	This is Update, a member of class TActiveXControl.
	ViewObjectGetExtent (see page 1552)	This is ViewObjectGetExtent, a member of class TActiveXControl.
V	WndProc (see page 1552)	This is WndProc, a member of class TActiveXControl.

Protected Properties

	Name	Description
R	AutoFactory (see page 5545)	The class factory that created this TAutoObject (see page 5540).
R	ConnectionPoints (see page 1553)	This is ConnectionPoints, a member of class TActiveXControl.
	EventSink (see page 5545)	This is EventSink, a member of class TAutoObject.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5555)	Instantiates a COM object that is not part of an aggregate.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateAggregated (see page 5556)	Instantiates a COM object that is part of an aggregate.
	CreateFromFactory (see page 5556)	Instantiates a COM object and initializes its values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 1522)	Destroys an instance of TActiveXControl (see page 1517).
V	Destroy (see page 5557)	Disposes of an instance of a COM object.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeOnRelease (see page 1512)	Supports the IVCLComObject interface.
	GetIDsOfNames (see page 3212)	This is GetIDsOfNames, a member of class IVCLComObject.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetTypeInfo (see page 3213)	This is GetTypeInfo, a member of class IVCLComObject.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 3213)	This is GetTypeInfoCount, a member of class IVCLComObject.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 1536)	Creates the class factory and connection object for the TActiveXControl (see page 1517).
V	Initialize (see page 5544)	Provides an interface to a method that initializes the Automation object.
V	Initialize (see page 5558)	Provides an interface to a method that initializes the COM object.
V	InitializeControl (see page 1513)	Creates InitializeControl creates the TActiveForm (see page 1489) instance and establishes a link between itself and the TActiveForm (see page 1489) referenced by the control properties.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invoke (see page 1513)	Invoke implements the OLE Automation IDispatch Invoke function.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	ObjAddRef (see page 5559)	Implements the AddRef method of the IUnknown interface.
V	ObjQueryInterface (see page 1513)	The ObjQueryInterface method takes an interface ID (the IID parameter) and returns the requested COM interface on the object.
V	ObjRelease (see page 5559)	Implements the Release method of the IUnknown interface.
	PropChanged (see page 1542)	This is the overview for the PropChanged method overload.
	PropRequestEdit (see page 1543)	This is the overview for the PropRequestEdit method overload.
	SafeCallException (see page 3213)	This is SafeCallException, a member of class IVCLComObject.
V	SafeCallException (see page 5560)	Handles OLE exceptions.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ClientSite (see page 1553)	Contains a pointer to the client site for the container on which the ActiveX control resides.
R	Control (see page 1553)	Indicates the VCL implementation of the ActiveX control.

R	Controller (see page 5561)	Specifies the controlling IUnknown interface that controls interfaces within an aggregate object.
R	Factory (see page 5561)	Specifies the class factory object for the TComObject (see page 5553) class.
R	InPlaceSite (see page 1554)	Contains a pointer to the site interface for the container on which the ActiveX control resides.
R	RefCount (see page 5562)	Indicates the number of references to the COM object that are currently in use.
	ServerExceptionHandler (see page 5562)	Provides access to the SafeCall exception notification process.

12.1.2.2 TActiveFormControl Methods

The methods of the TActiveFormControl class are listed here.

Protected Methods

	Name	Description
V	DefinePropertyPages (see page 1512)	This is DefinePropertyPages, a member of class TActiveFormControl.
V	EventSinkChanged (see page 1512)	This is EventSinkChanged, a member of class TActiveFormControl.

Public Methods

	Name	Description
	FreeOnRelease (see page 1512)	Supports the IVCLComObject interface.
V	InitializeControl (see page 1513)	Creates InitializeControl creates the TActiveForm (see page 1489) instance and establishes a link between itself and the TActiveForm (see page 1489) referenced by the control properties.
V	Invoke (see page 1513)	Invoke implements the OLE Automation IDispatch Invoke function.
V	ObjQueryInterface (see page 1513)	The ObjQueryInterface method takes an interface ID (the IID parameter) and returns the requested COM interface on the object.

12.1.2.2.1 TActiveFormControl.DefinePropertyPages Method

Pascal

```
procedure DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage); override;
```

C++

```
virtual __fastcall DefinePropertyPages(TDefinePropertyPage DefinePropertyPage);
```

Description

This is DefinePropertyPages, a member of class TActiveFormControl.

12.1.2.2.2 TActiveFormControl.EventSinkChanged Method

Pascal

```
procedure EventSinkChanged(const EventSink: IUnknown); override;
```

C++

```
virtual __fastcall EventSinkChanged(const IUnknown EventSink);
```

Description

This is EventSinkChanged, a member of class TActiveFormControl.

12.1.2.2.3 TActiveFormControl.FreeOnRelease Method

Supports the IVCLComObject interface.

Pascal

```
procedure FreeOnRelease;
```

C++

```
__fastcall FreeOnRelease();
```

Description

FreeOnRelease exists solely to support the IVCLComObject interface, and has an empty implementation.

12.1.2.2.4 TActiveFormControl.InitializeControl Method

Creates InitializeControl creates the TActiveForm (see page 1489) instance and establishes a link between itself and the TActiveForm (see page 1489) referenced by the control properties.

Pascal

```
procedure InitializeControl; override;
```

C++

```
virtual __fastcall InitializeControl();
```

Description

InitializeControl also instructs the TActiveForm (see page 1489) object to perform any of its own initialization.

12.1.2.2.5 TActiveFormControl.Invoke Method

Invoke implements the OLE Automation IDispatch Invoke function.

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; override;
```

C++

```
virtual __fastcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word Flags,
Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

Call Invoke to access the property or method specified by the DispID parameter. The Flags parameter indicates whether the DispID refers to method (DISPATCH_METHOD), a property value that is being read (DISPATCH_PROPERTYGET), or a property value that is being set (DISPATCH_PROPERTYPUT or DISPATCH_PROPERTYPUTREF). The Params and ExcepInfo parameters hold parameters and exception information, respectively, and depend on the property or method identified by the DispID parameter.

Invoke returns S_OK if it is successful. Otherwise, it returns an error code defined by the IDispatch interface.

12.1.2.2.6 TActiveFormControl.ObjQueryInterface Method

The ObjQueryInterface method takes an interface ID (the IID parameter) and returns the requested COM interface on the object.

Pascal

```
function ObjQueryInterface(const IID: TGUID; out Obj): HResult; override;
```

C++

```
virtual __fastcall HRESULT ObjQueryInterface(const TGUID IID,  Obj);
```

Description

The ObjQueryInterface method forwards the call first to the associated TActiveForm (see page 1489) object if it supports the requested interface, otherwise it queries its own interface.

See Also

TActiveForm (see page 1489)

TActiveFormFactory (see page 1514)

12.1.3 TActiveFormFactory Class

The ActiveFormFactory object is the class factory for TActiveForm (see page 1489).

Class Hierarchy

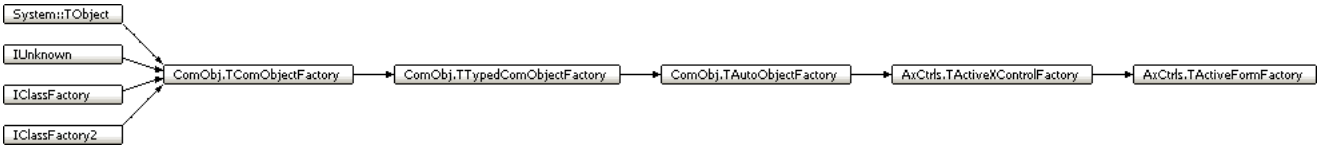

Pascal

```
TActiveFormFactory = class(TActiveXControlFactory);
```

C++

```
class TActiveFormFactory : public TActiveXControlFactory;
```

File

AxCtrls

Description

The ActiveFormFactory object is the class factory responsible for creating instances of the TActiveFormControl (see page 1507) object and therefore TActiveForm (see page 1489) object.

The ActiveForm wizard adds an ActiveForm factory object in the initialization section of the unit that defines the ActiveForm.

See Also

TActiveForm (see page 1489)

TActiveFormControl (see page 1507)

12.1.3.1 TActiveFormFactory Members

The following tables list the members exposed by TActiveFormFactory.

Protected Methods

	Name	Description
	_AddRef (see page 5567)	This is _AddRef, a member of class TComObjectFactory.
	_Release (see page 5567)	This is _Release, a member of class TComObjectFactory.
	CreateInstance (see page 5568)	This is CreateInstance, a member of class TComObjectFactory.
	CreateInstanceLic (see page 5568)	This is CreateInstanceLic, a member of class TComObjectFactory.
V	GetLicenseFileName (see page 1558)	This is GetLicenseFileName, a member of class TActiveXControlFactory.

	GetLicenseString (see page 5568)	This is GetLicenseString, a member of class TComObjectFactory.
	GetLicInfo (see page 5568)	This is GetLicInfo, a member of class TComObjectFactory.
	GetProgID (see page 5569)	This is GetProgID, a member of class TComObjectFactory.
	HasMachineLicense (see page 1558)	This is HasMachineLicense, a member of class TActiveXControlFactory.
	HasMachineLicense (see page 5569)	This is HasMachineLicense, a member of class TComObjectFactory.
	LockServer (see page 5569)	This is LockServer, a member of class TComObjectFactory.
	QueryInterface (see page 5569)	This is QueryInterface, a member of class TComObjectFactory.
	RequestLicKey (see page 5570)	This is RequestLicKey, a member of class TComObjectFactory.
	ValidateUserLicense (see page 5571)	This is ValidateUserLicense, a member of class TComObjectFactory.

Public Methods

	Name	Description
	AddVerb (see page 1557)	AddVerb adds a verb to the object's list of supported verbs.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1556)	Creates and initializes a new Create object.
	Create (see page 5548)	This is Create, a member of class TAutoObjectFactory.
	Create (see page 5565)	Instantiates a class factory object.
	Create (see page 5585)	Instantiates a TTypedComObjectFactory (see page 5583) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateComObject (see page 5567)	Instantiates a TComObject (see page 5553) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 1557)	Frees the instance of the Destroy.
	Destroy (see page 5566)	Disposes of an instance of a class factory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetInterfaceTypeInfo (see page 5586)	Retrieves the type information for the typed COM object created by the factory.
	GetIntfEntry (see page 1516)	Retrieves a pointer to the dispatch interface.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RegisterClassObject (see page 5569)	Registers the COM class associated with the factory object.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	UpdateRegistry (see page 1559)	UpdateRegistry adds or removes registry entries for the object.
	UpdateRegistry (see page 5570)	Registers COM objects created by the factory.

V	UpdateRegistry (see page 5586)	Registers the typed COM object class.

Public Properties

	Name	Description
R	ClassID (see page 5571)	This is ClassID, a member of class TComObjectFactory.
R	ClassInfo (see page 5587)	Specifies an interface reference that provides type information without having to load a type library.
R	ClassName (see page 5572)	Specifies the name of the class that the class factory instantiates.
R	ComClass (see page 5572)	This is ComClass, a member of class TComObjectFactory.
R	ComServer (see page 5572)	This is ComServer, a member of class TComObjectFactory.
R	Description (see page 5572)	Describes the class object.
R	DispIntfEntry (see page 5549)	Contains the entry for the dispatch interface implemented by the OLE Automation object created by the factory.
R	DispTypeInfo (see page 5549)	Returns an ITypeInfo interface reference to the type information for the dispinterface implemented by the OLE Automation object.
	ErrorIID (see page 5573)	Indicates the interface of the CoClass in which an error occurred.
R	EventIID (see page 5549)	This is EventIID, a member of class TAutoObjectFactory.
R	EventTypeInfo (see page 5550)	This is EventTypeInfo, a member of class TAutoObjectFactory.
R	Instancing (see page 5573)	Indicates the process space of COM object instances.
	LicString (see page 5573)	Contains licensing information about the object.
R	MiscStatus (see page 1559)	Represents a set of status bits that are stored in the registry.
R	ProgID (see page 5574)	Specifies the name of the class factory.
	ShowErrors (see page 5574)	Indicates whether the COM object factory displays a message box when exceptions are raised.
	SupportsLicensing (see page 5574)	Indicates whether the object being instantiated supports licensing.
R	ThreadingModel (see page 5575)	Indicates the type of thread support provided by objects that the factory creates.
R	ToolboxBitmapID (see page 1560)	Designates the resource identifier of the ActiveX control.
R	WinControlClass (see page 1560)	WinControlClass is the class of the VCL control that implements the Windows behavior for the ActiveX control.

12.1.3.2 TActiveFormFactory Methods

The methods of the TActiveFormFactory class are listed here.

Public Methods

	Name	Description
V	GetIntfEntry (see page 1516)	Retrieves a pointer to the dispatch interface.

12.1.3.2.1 TActiveFormFactory.GetIntfEntry Method

Retrieves a pointer to the dispatch interface.

Pascal

```
function GetIntfEntry(Guid: TGUID): PInterfaceEntry; override;
```

C++

```
virtual __fastcall PInterfaceEntry GetIntfEntry(TGUID Guid);
```

Description

The GetIntfEntry method obtains an interface, if it exists, from the TActiveForm (see page 1489) class. This is an internal method of the DAX framework. retrieves a pointer to the dispatch interface from the associated TActiveForm (see page 1489) object.

See Also

TActiveForm (see page 1489)

12.1.4 TActiveXControl Class

TActiveXControl is an abstract class that provides the functionality to embed a VCL control into an ActiveX Container window.

Class Hierarchy

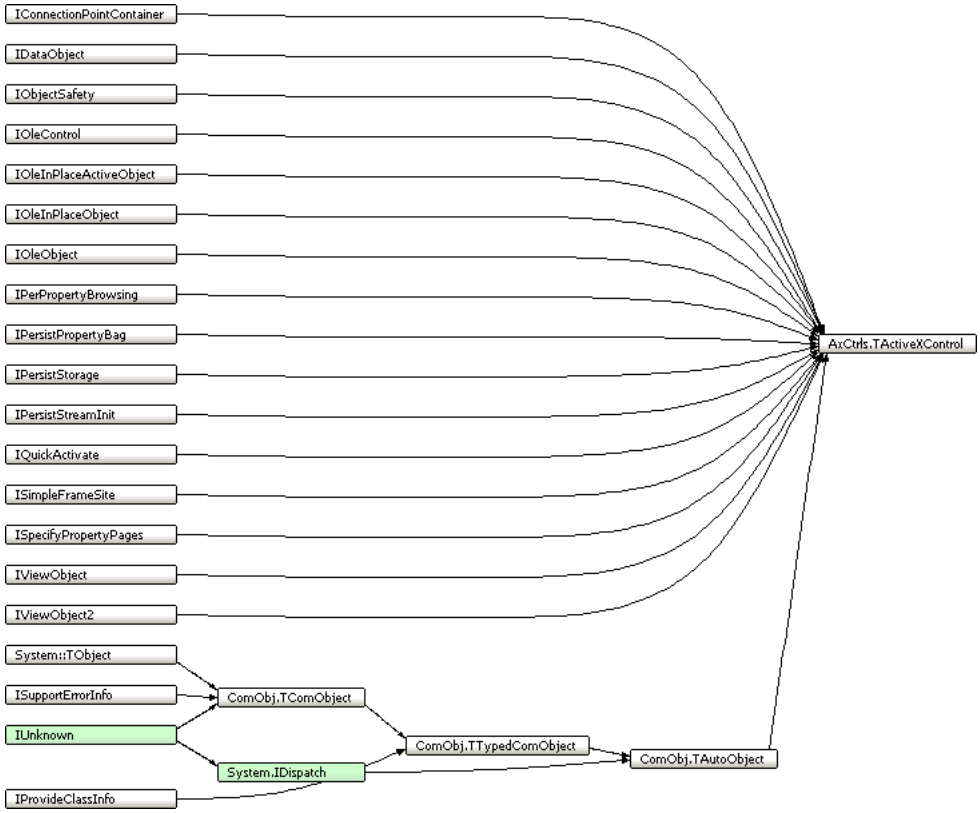

Pascal

```
TActiveXControl = class(TAutoObject, IConnectionPointContainer, IDataObject, IObjectSafety,
IOleControl, IOleInPlaceActiveObject, IOleInPlaceObject, IOleObject, IPerPropertyBrowsing,
IPersistPropertyBag, IPersistStorage, IPersistStreamInit, IQuickActivate, ISimpleFrameSite,
ISpecifyPropertyPages, IViewObject, IViewObject2);
```

C++

```
class TActiveXControl : public TAutoObject, public IConnectionPointContainer, public
IDataObject, public IObjectSafety, public IOleControl, public IOleInPlaceActiveObject, public
IOleInPlaceObject, public IOleObject, public IPerPropertyBrowsing, public IPersistPropertyBag,
public IPersistStorage, public IPersistStreamInit, public IQuickActivate, public
ISimpleFrameSite, public ISpecifyPropertyPages, public IViewObject, public IViewObject2;
```

File

AxCtrls

Description

TActiveXControl defines the core behavior and interfaces required of an ActiveX control and connects that behavior to any VCL control derived from TWinControl. This allows the control to be embedded in any ActiveX container, including Internet Explorer, Visual Basic, PowerBuilder, Paradox, CodeGear C++, IntraBuilder and, of course, Delphi.

To create a new ActiveX control class, use the ActiveX control wizard. The steps for building an ActiveX control are covered in the Developer's Guide or in the online help document, Developing COM-based applications.

The TActiveXControl class is derived from TAutoObject, therefore the object resulting is a COM object that supports COM-based Automation, a type library, and has a factory. In addition, TActiveXControl implements the following behaviors:

embedding in a container (site)

in-place activation

persistence

events

property pages

property browsing

TActiveXControl is an abstract class, and so cannot be instantiated. Instead, you must derive a new class from TActiveXControl to implement an ActiveX control. This derived class is called an implementation class. An implementation class derived from TActiveXControl normally implements an automation interface as well, to specify the object's properties and methods.

Ordinarily, you do not create an instance of an implementation class directly. Instead, you create a factory object in the initialization section of the unit that contains the class declaration. The factory object will then create the ActiveX control object when asked to by the system.

In addition to the interfaces inherited from TAutoObject, TActiveXControl implements the following interfaces:

Note: See the DelCtrls demo in ..\ActiveX\Delctrls for an example of using VCL components as ActiveX controls.

See Also

CreateOleObject

Elements of an ActiveX Control

Generating an ActiveX Control from a VCL Control

Implementing COM Objects with Wizards

12.1.4.1 TActiveXControl Members

The following tables list the members exposed by TActiveXControl.

Protected Methods

	Name	Description
	_AddRef (see page 5558)	This is _AddRef, a member of class TComObject.
	_Release (see page 5558)	This is _Release, a member of class TComObject.
	Advise (see page 1525)	This is Advise, a member of class TActiveXControl.
	Close (see page 1525)	This is Close, a member of class TActiveXControl.
	ContextSensitiveHelp (see page 1525)	This is ContextSensitiveHelp, a member of class TActiveXControl.
	DAdvise (see page 1525)	This is DAdvise, a member of class TActiveXControl.
V	DefinePropertyPages (see page 1526)	This is DefinePropertyPages, a member of class TActiveXControl.
	DoVerb (see page 1526)	This is DoVerb, a member of class TActiveXControl.
	Draw (see page 1526)	This is Draw, a member of class TActiveXControl.
	DUnadvise (see page 1526)	This is DUnadvise, a member of class TActiveXControl.
	EnableModeless (see page 1527)	This is EnableModeless, a member of class TActiveXControl.
	EnumAdvise (see page 1527)	This is EnumAdvise, a member of class TActiveXControl.
	EnumDAdvise (see page 1527)	This is EnumDAdvise, a member of class TActiveXControl.
	EnumFormatEtc (see page 1527)	This is EnumFormatEtc, a member of class TActiveXControl.

	EnumVerbs (see page 1527)	This is EnumVerbs, a member of class TActiveXControl.
	EventConnect (see page 5542)	This is EventConnect, a member of class TAutoObject.
V	EventSinkChanged (see page 5543)	This is EventSinkChanged, a member of class TAutoObject.
	Freeze (see page 1528)	This is Freeze, a member of class TActiveXControl.
	FreezeEvents (see page 1528)	This is FreezeEvents, a member of class TActiveXControl.
	Get_BackColor (see page 1528)	This is Get_BackColor, a member of class TActiveXControl.
	Get_Caption (see page 1528)	This is Get_Caption, a member of class TActiveXControl.
	Get_Enabled (see page 1529)	This is Get_Enabled, a member of class TActiveXControl.
	Get_Font (see page 1529)	This is Get_Font, a member of class TActiveXControl.
	Get_ForeColor (see page 1529)	This is Get_ForeColor, a member of class TActiveXControl.
	Get_HWnd (see page 1529)	This is Get_HWnd, a member of class TActiveXControl.
	Get_TabStop (see page 1529)	This is Get_TabStop, a member of class TActiveXControl.
	Get_Text (see page 1530)	This is Get_Text, a member of class TActiveXControl.
	GetAdvise (see page 1530)	This is GetAdvise, a member of class TActiveXControl.
	GetCanonicalFormatEtc (see page 1530)	This is GetCanonicalFormatEtc, a member of class TActiveXControl.
	GetClassID (see page 1530)	This is GetClassID, a member of class TActiveXControl.
	GetClassInfo (see page 5583)	Returns a pointer to a TTypedComObject (see page 5581)'s type information.
	GetClientSite (see page 1531)	This is GetClientSite, a member of class TActiveXControl.
	GetClipboardData (see page 1531)	This is GetClipboardData, a member of class TActiveXControl.
	GetColorSet (see page 1531)	This is GetColorSet, a member of class TActiveXControl.
	GetContentExtent (see page 1531)	This is GetContentExtent, a member of class TActiveXControl.
	GetControlInfo (see page 1531)	This is GetControlInfo, a member of class TActiveXControl.
	GetData (see page 1532)	This is GetData, a member of class TActiveXControl.
	GetDataHere (see page 1532)	This is GetDataHere, a member of class TActiveXControl.
	GetDisplayString (see page 1532)	This is GetDisplayString, a member of class TActiveXControl.
	GetExtent (see page 1532)	This is GetExtent, a member of class TActiveXControl.
V	GetIDsOfNames (see page 5543)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetInterfaceSafetyOptions (see page 1533)	This is GetInterfaceSafetyOptions, a member of class TActiveXControl.
	GetMiscStatus (see page 1533)	This is GetMiscStatus, a member of class TActiveXControl.
	GetMoniker (see page 1533)	This is GetMoniker, a member of class TActiveXControl.
	GetPages (see page 1533)	This is GetPages, a member of class TActiveXControl.
	GetPredefinedStrings (see page 1533)	This is GetPredefinedStrings, a member of class TActiveXControl.
	GetPredefinedValue (see page 1534)	This is GetPredefinedValue, a member of class TActiveXControl.
V	GetPropertyString (see page 1534)	This is GetPropertyString, a member of class TActiveXControl.
V	GetPropertyStrings (see page 1534)	This is GetPropertyStrings, a member of class TActiveXControl.
V	GetPropertyValue (see page 1534)	This is GetPropertyValue, a member of class TActiveXControl.
V	GetPropFromBag (see page 1535)	This is GetPropFromBag, a member of class TActiveXControl.
	GetSizeMax (see page 1535)	This is GetSizeMax, a member of class TActiveXControl.
V	GetTypeInfo (see page 5543)	Retrieves the type information for an object.
V	GetTypeInfoCount (see page 5544)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetUserClassID (see page 1535)	This is GetUserClassID, a member of class TActiveXControl.
	GetUserType (see page 1535)	This is GetUserType, a member of class TActiveXControl.
	GetWindow (see page 1536)	This is GetWindow, a member of class TActiveXControl.
	HandsOffStorage (see page 1536)	This is HandsOffStorage, a member of class TActiveXControl.
	InitFromData (see page 1536)	This is InitFromData, a member of class TActiveXControl.
V	InitializeControl (see page 1537)	This is InitializeControl, a member of class TActiveXControl.
	InitNew (see page 1537)	This is InitNew, a member of class TActiveXControl.
	InPlaceActivate (see page 1537)	This is InPlaceActivate, a member of class TActiveXControl.
	InPlaceDeactivate (see page 1537)	This is InPlaceDeactivate, a member of class TActiveXControl.
	InterfaceSupportsErrorInfo (see page 5559)	This is InterfaceSupportsErrorInfo, a member of class TComObject.
V	Invoke (see page 5544)	Provides access to Automation properties and methods exposed by an object that supports the IDispatch interface.
	IsDirty (see page 1537)	This is IsDirty, a member of class TActiveXControl.

	IsUpToDate (see page 1538)	This is IsUpToDate, a member of class TActiveXControl.
V	LoadFromStream (see page 1538)	This is LoadFromStream, a member of class TActiveXControl.
	MapPropertyToPage (see page 1538)	This is MapPropertyToPage, a member of class TActiveXControl.
	OnAmbientPropertyChange (see page 1539)	This is OnAmbientPropertyChange, a member of class TActiveXControl.
	OnDocWindowActivate (see page 1539)	This is OnDocWindowActivate, a member of class TActiveXControl.
	OnFrameWindowActivate (see page 1539)	This is OnFrameWindowActivate, a member of class TActiveXControl.
	OnMnemonic (see page 1539)	This is OnMnemonic, a member of class TActiveXControl.
V	PerformVerb (see page 1540)	This is PerformVerb, a member of class TActiveXControl.
	PersistPropBagInitNew (see page 1540)	This is PersistPropBagInitNew, a member of class TActiveXControl.
	PersistPropBagLoad (see page 1540)	This is PersistPropBagLoad, a member of class TActiveXControl.
	PersistPropBagSave (see page 1540)	This is PersistPropBagSave, a member of class TActiveXControl.
	PersistStorageInitNew (see page 1541)	This is PersistStorageInitNew, a member of class TActiveXControl.
	PersistStorageLoad (see page 1541)	This is PersistStorageLoad, a member of class TActiveXControl.
	PersistStorageSave (see page 1541)	This is PersistStorageSave, a member of class TActiveXControl.
	PersistStreamLoad (see page 1541)	This is PersistStreamLoad, a member of class TActiveXControl.
	PersistStreamSave (see page 1541)	This is PersistStreamSave, a member of class TActiveXControl.
	PostMessageFilter (see page 1542)	This is PostMessageFilter, a member of class TActiveXControl.
	PreMessageFilter (see page 1542)	This is PreMessageFilter, a member of class TActiveXControl.
V	PutPropInBag (see page 1543)	This is PutPropInBag, a member of class TActiveXControl.
	QueryGetData (see page 1544)	This is QueryGetData, a member of class TActiveXControl.
	QueryInterface (see page 5560)	This is QueryInterface, a member of class TComObject.
	QuickActivate (see page 1544)	This is QuickActivate, a member of class TActiveXControl.
	ReactivateAndUndo (see page 1544)	This is ReactivateAndUndo, a member of class TActiveXControl.
	ResizeBorder (see page 1544)	This is ResizeBorder, a member of class TActiveXControl.
	SaveCompleted (see page 1544)	This is SaveCompleted, a member of class TActiveXControl.
V	SaveToStream (see page 1545)	This is SaveToStream, a member of class TActiveXControl.
	Set_BackColor (see page 1545)	This is Set_BackColor, a member of class TActiveXControl.
	Set_Caption (see page 1545)	This is Set_Caption, a member of class TActiveXControl.
	Set_Enabled (see page 1545)	This is Set_Enabled, a member of class TActiveXControl.
	Set_Font (see page 1546)	This is Set_Font, a member of class TActiveXControl.
	Set_ForeColor (see page 1546)	This is Set_ForeColor, a member of class TActiveXControl.
	Set_TabStop (see page 1546)	This is Set_TabStop, a member of class TActiveXControl.
	Set_Text (see page 1546)	This is Set_Text, a member of class TActiveXControl.
	SetAdvise (see page 1546)	This is SetAdvise, a member of class TActiveXControl.
	SetClientSite (see page 1547)	This is SetClientSite, a member of class TActiveXControl.
	SetColorScheme (see page 1547)	This is SetColorScheme, a member of class TActiveXControl.
	SetContentExtent (see page 1547)	This is SetContentExtent, a member of class TActiveXControl.
	SetData (see page 1547)	This is SetData, a member of class TActiveXControl.
	SetExtent (see page 1548)	This is SetExtent, a member of class TActiveXControl.
	SetHostNames (see page 1548)	This is SetHostNames, a member of class TActiveXControl.
	SetInPlaceSite (see page 1548)	This is SetInPlaceSite, a member of class TActiveXControl.
V	SetInterfaceSafetyOptions (see page 1548)	This is SetInterfaceSafetyOptions, a member of class TActiveXControl.
	SetMoniker (see page 1548)	This is SetMoniker, a member of class TActiveXControl.
	SetObjectRects (see page 1549)	This is SetObjectRects, a member of class TActiveXControl.
	ShowPropertyDialog (see page 1549)	This is ShowPropertyDialog, a member of class TActiveXControl.
	StdClickEvent (see page 1549)	This is StdClickEvent, a member of class TActiveXControl.
	StdDblClickEvent (see page 1549)	This is StdDblClickEvent, a member of class TActiveXControl.
	StdKeyDownEvent (see page 1550)	This is StdKeyDownEvent, a member of class TActiveXControl.
	StdKeyPressEvent (see page 1550)	This is StdKeyPressEvent, a member of class TActiveXControl.
	StdKeyUpEvent (see page 1550)	This is StdKeyUpEvent, a member of class TActiveXControl.
	StdMouseDownEvent (see page 1550)	This is StdMouseDownEvent, a member of class TActiveXControl.
	StdMouseMoveEvent (see page 1550)	This is StdMouseMoveEvent, a member of class TActiveXControl.
	StdMouseUpEvent (see page 1551)	This is StdMouseUpEvent, a member of class TActiveXControl.
	TranslateAccelerator (see page 1551)	This is TranslateAccelerator, a member of class TActiveXControl.

	UIDeactivate (see page 1551)	This is UIDeactivate, a member of class TActiveXControl.
	Unadvise (see page 1551)	This is Unadvise, a member of class TActiveXControl.
	Unfreeze (see page 1552)	This is Unfreeze, a member of class TActiveXControl.
	Update (see page 1552)	This is Update, a member of class TActiveXControl.
	ViewObjectGetExtent (see page 1552)	This is ViewObjectGetExtent, a member of class TActiveXControl.
V	WndProc (see page 1552)	This is WndProc, a member of class TActiveXControl.

Protected Properties

	Name	Description
R	AutoFactory (see page 5545)	The class factory that created this TAutoObject (see page 5540).
R	ConnectionPoints (see page 1553)	This is ConnectionPoints, a member of class TActiveXControl.
	EventSink (see page 5545)	This is EventSink, a member of class TAutoObject.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5555)	Instantiates a COM object that is not part of an aggregate.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateAggregated (see page 5556)	Instantiates a COM object that is part of an aggregate.
	CreateFromFactory (see page 5556)	Instantiates a COM object and initializes its values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 1522)	Destroys an instance of TActiveXControl (see page 1517).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 1536)	Creates the class factory and connection object for the TActiveXControl (see page 1517).
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	ObjAddRef (see page 5559)	Implements the AddRef method of the IUnknown interface.
V	ObjQueryInterface (see page 1538)	Returns a requested COM interface on the object.

V	ObjRelease (see page 5559)	Implements the Release method of the IUnknown interface.
	PropChanged (see page 1542)	This is the overview for the PropChanged method overload.
	PropRequestEdit (see page 1543)	This is the overview for the PropRequestEdit method overload.
V	SafeCallException (see page 5560)	Handles OLE exceptions.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ClientSite (see page 1553)	Contains a pointer to the client site for the container on which the ActiveX control resides.
R	Control (see page 1553)	Indicates the VCL implementation of the ActiveX control.
R	Controller (see page 5561)	Specifies the controlling IUnknown interface that controls interfaces within an aggregate object.
R	Factory (see page 5561)	Specifies the class factory object for the TComObject (see page 5553) class.
R	InPlaceSite (see page 1554)	Contains a pointer to the site interface for the container on which the ActiveX control resides.
R	RefCount (see page 5562)	Indicates the number of references to the COM object that are currently in use.
	ServerExceptionHandler (see page 5562)	Provides access to the SafeCall exception notification process.

12.1.4.2 TActiveXControl.Destroy Destructor

Destroys an instance of TActiveXControl (see page 1517).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TActiveXControl();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the ActiveX control reference is not nil before calling Destroy. Destroy frees the resources used to turn the Control (see page 1553) into an active X control. It does not destroy the object referenced by the Control (see page 1553) property.

See Also

Control (see page 1553)

Free (see page 22300)

12.1.4.3 TActiveXControl Methods

The methods of the TActiveXControl class are listed here.

Protected Methods

	Name	Description
	Advise (see page 1525)	This is Advise, a member of class TActiveXControl.
	Close (see page 1525)	This is Close, a member of class TActiveXControl.
	ContextSensitiveHelp (see page 1525)	This is ContextSensitiveHelp, a member of class TActiveXControl.
	DAdvise (see page 1525)	This is DAdvise, a member of class TActiveXControl.
V	DefinePropertyPages (see page 1526)	This is DefinePropertyPages, a member of class TActiveXControl.
	DoVerb (see page 1526)	This is DoVerb, a member of class TActiveXControl.
	Draw (see page 1526)	This is Draw, a member of class TActiveXControl.
	DUnadvise (see page 1526)	This is DUnadvise, a member of class TActiveXControl.

	EnableModeless (see page 1527)	This is EnableModeless, a member of class TActiveXControl.
	EnumAdvise (see page 1527)	This is EnumAdvise, a member of class TActiveXControl.
	EnumDAdvise (see page 1527)	This is EnumDAdvise, a member of class TActiveXControl.
	EnumFormatEtc (see page 1527)	This is EnumFormatEtc, a member of class TActiveXControl.
	EnumVerbs (see page 1527)	This is EnumVerbs, a member of class TActiveXControl.
	Freeze (see page 1528)	This is Freeze, a member of class TActiveXControl.
	FreezeEvents (see page 1528)	This is FreezeEvents, a member of class TActiveXControl.
	Get_BackColor (see page 1528)	This is Get_BackColor, a member of class TActiveXControl.
	Get_Caption (see page 1528)	This is Get_Caption, a member of class TActiveXControl.
	Get_Enabled (see page 1529)	This is Get_Enabled, a member of class TActiveXControl.
	Get_Font (see page 1529)	This is Get_Font, a member of class TActiveXControl.
	Get_ForeColor (see page 1529)	This is Get_ForeColor, a member of class TActiveXControl.
	Get_HWnd (see page 1529)	This is Get_HWnd, a member of class TActiveXControl.
	Get_TabStop (see page 1529)	This is Get_TabStop, a member of class TActiveXControl.
	Get_Text (see page 1530)	This is Get_Text, a member of class TActiveXControl.
	GetAdvise (see page 1530)	This is GetAdvise, a member of class TActiveXControl.
	GetCanonicalFormatEtc (see page 1530)	This is GetCanonicalFormatEtc, a member of class TActiveXControl.
	GetClassID (see page 1530)	This is GetClassID, a member of class TActiveXControl.
	GetClientSite (see page 1531)	This is GetClientSite, a member of class TActiveXControl.
	GetClipboardData (see page 1531)	This is GetClipboardData, a member of class TActiveXControl.
	GetColorSet (see page 1531)	This is GetColorSet, a member of class TActiveXControl.
	GetContentExtent (see page 1531)	This is GetContentExtent, a member of class TActiveXControl.
	GetControlInfo (see page 1531)	This is GetControlInfo, a member of class TActiveXControl.
	GetData (see page 1532)	This is GetData, a member of class TActiveXControl.
	GetDataHere (see page 1532)	This is GetDataHere, a member of class TActiveXControl.
	GetDisplayString (see page 1532)	This is GetDisplayString, a member of class TActiveXControl.
	GetExtent (see page 1532)	This is GetExtent, a member of class TActiveXControl.
V	GetInterfaceSafetyOptions (see page 1533)	This is GetInterfaceSafetyOptions, a member of class TActiveXControl.
	GetMiscStatus (see page 1533)	This is GetMiscStatus, a member of class TActiveXControl.
	GetMoniker (see page 1533)	This is GetMoniker, a member of class TActiveXControl.
	GetPages (see page 1533)	This is GetPages, a member of class TActiveXControl.
	GetPredefinedStrings (see page 1533)	This is GetPredefinedStrings, a member of class TActiveXControl.
	GetPredefinedValue (see page 1534)	This is GetPredefinedValue, a member of class TActiveXControl.
V	GetPropertyString (see page 1534)	This is GetPropertyString, a member of class TActiveXControl.
V	GetPropertyStrings (see page 1534)	This is GetPropertyStrings, a member of class TActiveXControl.
V	GetPropertyValue (see page 1534)	This is GetPropertyValue, a member of class TActiveXControl.
V	GetPropFromBag (see page 1535)	This is GetPropFromBag, a member of class TActiveXControl.
	GetSizeMax (see page 1535)	This is GetSizeMax, a member of class TActiveXControl.
	GetUserClassID (see page 1535)	This is GetUserClassID, a member of class TActiveXControl.
	GetUserType (see page 1535)	This is GetUserType, a member of class TActiveXControl.
	GetWindow (see page 1536)	This is GetWindow, a member of class TActiveXControl.
	HandsOffStorage (see page 1536)	This is HandsOffStorage, a member of class TActiveXControl.
	InitFromData (see page 1536)	This is InitFromData, a member of class TActiveXControl.
V	InitializeControl (see page 1537)	This is InitializeControl, a member of class TActiveXControl.
	InitNew (see page 1537)	This is InitNew, a member of class TActiveXControl.
	InPlaceActivate (see page 1537)	This is InPlaceActivate, a member of class TActiveXControl.
	InPlaceDeactivate (see page 1537)	This is InPlaceDeactivate, a member of class TActiveXControl.
	IsDirty (see page 1537)	This is IsDirty, a member of class TActiveXControl.
	IsUpToDate (see page 1538)	This is IsUpToDate, a member of class TActiveXControl.
V	LoadFromStream (see page 1538)	This is LoadFromStream, a member of class TActiveXControl.
	MapPropertyToPage (see page 1538)	This is MapPropertyToPage, a member of class TActiveXControl.
	OnAmbientPropertyChange (see page 1539)	This is OnAmbientPropertyChange, a member of class TActiveXControl.
	OnDocWindowActivate (see page 1539)	This is OnDocWindowActivate, a member of class TActiveXControl.
	OnFrameWindowActivate (see page 1539)	This is OnFrameWindowActivate, a member of class TActiveXControl.

	OnMnemonic (see page 1539)	This is OnMnemonic, a member of class TActiveXControl.
V	PerformVerb (see page 1540)	This is PerformVerb, a member of class TActiveXControl.
	PersistPropBagInitNew (see page 1540)	This is PersistPropBagInitNew, a member of class TActiveXControl.
	PersistPropBagLoad (see page 1540)	This is PersistPropBagLoad, a member of class TActiveXControl.
	PersistPropBagSave (see page 1540)	This is PersistPropBagSave, a member of class TActiveXControl.
	PersistStorageInitNew (see page 1541)	This is PersistStorageInitNew, a member of class TActiveXControl.
	PersistStorageLoad (see page 1541)	This is PersistStorageLoad, a member of class TActiveXControl.
	PersistStorageSave (see page 1541)	This is PersistStorageSave, a member of class TActiveXControl.
	PersistStreamLoad (see page 1541)	This is PersistStreamLoad, a member of class TActiveXControl.
	PersistStreamSave (see page 1541)	This is PersistStreamSave, a member of class TActiveXControl.
	PostMessageFilter (see page 1542)	This is PostMessageFilter, a member of class TActiveXControl.
	PreMessageFilter (see page 1542)	This is PreMessageFilter, a member of class TActiveXControl.
V	PutPropInBag (see page 1543)	This is PutPropInBag, a member of class TActiveXControl.
	QueryGetData (see page 1544)	This is QueryGetData, a member of class TActiveXControl.
	QuickActivate (see page 1544)	This is QuickActivate, a member of class TActiveXControl.
	ReactivateAndUndo (see page 1544)	This is ReactivateAndUndo, a member of class TActiveXControl.
	ResizeBorder (see page 1544)	This is ResizeBorder, a member of class TActiveXControl.
	SaveCompleted (see page 1544)	This is SaveCompleted, a member of class TActiveXControl.
V	SaveToStream (see page 1545)	This is SaveToStream, a member of class TActiveXControl.
	Set_BackColor (see page 1545)	This is Set_BackColor, a member of class TActiveXControl.
	Set_Caption (see page 1545)	This is Set_Caption, a member of class TActiveXControl.
	Set_Enabled (see page 1545)	This is Set_Enabled, a member of class TActiveXControl.
	Set_Font (see page 1546)	This is Set_Font, a member of class TActiveXControl.
	Set_ForeColor (see page 1546)	This is Set_ForeColor, a member of class TActiveXControl.
	Set_TabStop (see page 1546)	This is Set_TabStop, a member of class TActiveXControl.
	Set_Text (see page 1546)	This is Set_Text, a member of class TActiveXControl.
	SetAdvise (see page 1546)	This is SetAdvise, a member of class TActiveXControl.
	SetClientSite (see page 1547)	This is SetClientSite, a member of class TActiveXControl.
	SetColorScheme (see page 1547)	This is SetColorScheme, a member of class TActiveXControl.
	SetContentExtent (see page 1547)	This is SetContentExtent, a member of class TActiveXControl.
	SetData (see page 1547)	This is SetData, a member of class TActiveXControl.
	SetExtent (see page 1548)	This is SetExtent, a member of class TActiveXControl.
	SetHostNames (see page 1548)	This is SetHostNames, a member of class TActiveXControl.
	SetInPlaceSite (see page 1548)	This is SetInPlaceSite, a member of class TActiveXControl.
V	SetInterfaceSafetyOptions (see page 1548)	This is SetInterfaceSafetyOptions, a member of class TActiveXControl.
	SetMoniker (see page 1548)	This is SetMoniker, a member of class TActiveXControl.
	SetObjectRects (see page 1549)	This is SetObjectRects, a member of class TActiveXControl.
	ShowPropertyDialog (see page 1549)	This is ShowPropertyDialog, a member of class TActiveXControl.
	StdClickEvent (see page 1549)	This is StdClickEvent, a member of class TActiveXControl.
	StdDblClickEvent (see page 1549)	This is StdDblClickEvent, a member of class TActiveXControl.
	StdKeyDownEvent (see page 1550)	This is StdKeyDownEvent, a member of class TActiveXControl.
	StdKeyPressEvent (see page 1550)	This is StdKeyPressEvent, a member of class TActiveXControl.
	StdKeyUpEvent (see page 1550)	This is StdKeyUpEvent, a member of class TActiveXControl.
	StdMouseDownEvent (see page 1550)	This is StdMouseDownEvent, a member of class TActiveXControl.
	StdMouseMoveEvent (see page 1550)	This is StdMouseMoveEvent, a member of class TActiveXControl.
	StdMouseUpEvent (see page 1551)	This is StdMouseUpEvent, a member of class TActiveXControl.
	TranslateAccelerator (see page 1551)	This is TranslateAccelerator, a member of class TActiveXControl.
	UIDeactivate (see page 1551)	This is UIDeactivate, a member of class TActiveXControl.
	Unadvise (see page 1551)	This is Unadvise, a member of class TActiveXControl.
	Unfreeze (see page 1552)	This is Unfreeze, a member of class TActiveXControl.
	Update (see page 1552)	This is Update, a member of class TActiveXControl.
	ViewObjectGetExtent (see page 1552)	This is ViewObjectGetExtent, a member of class TActiveXControl.
V	WndProc (see page 1552)	This is WndProc, a member of class TActiveXControl.

Public Methods

	Name	Description
	Initialize (see page 1536)	Creates the class factory and connection object for the TActiveXControl (see page 1517).
	ObjQueryInterface (see page 1538)	Returns a requested COM interface on the object.
	PropChanged (see page 1542)	This is the overview for the PropChanged method overload.
	PropRequestEdit (see page 1543)	This is the overview for the PropRequestEdit method overload.

12.1.4.3.1 TActiveXControl.Advise Method

Pascal

```
function Advise(const advSink: IAdviseSink; out dwConnection: Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Advise(const IAdviseSink advSink, Longint dwConnection);
```

Description

This is Advise, a member of class TActiveXControl.

12.1.4.3.2 TActiveXControl.Close Method

Pascal

```
function Close(dwSaveOption: Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Close(Longint dwSaveOption);
```

Description

This is Close, a member of class TActiveXControl.

12.1.4.3.3 TActiveXControl.ContextSensitiveHelp Method

Pascal

```
function ContextSensitiveHelp(fEnterMode: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT ContextSensitiveHelp(BOOL fEnterMode);
```

Description

This is ContextSensitiveHelp, a member of class TActiveXControl.

12.1.4.3.4 TActiveXControl.DAdvise Method

Pascal

```
function DAdvise(const formatetc: TFormatEtc; advf: Longint; const advSink: IAdviseSink; out
dwConnection: Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT DAdvise(const TFormatEtc formatetc, Longint advf, const
IAdviseSink advSink, Longint dwConnection);
```

Description

This is DAdvise, a member of class TActiveXControl.

12.1.4.3.5 TActiveXControl.DefinePropertyPages Method

Pascal

```
procedure DefinePropertyPages(DefinePropertyPage: TDefinePropertyPage); virtual;
```

C++

```
virtual __fastcall DefinePropertyPages(TDefinePropertyPage DefinePropertyPage);
```

Description

This is DefinePropertyPages, a member of class TActiveXControl.

12.1.4.3.6 TActiveXControl.DoVerb Method

Pascal

```
function DoVerb(iVerb: Longint; msg: PMsg; const activeSite: IOleClientSite; lindex: Longint;
hwndParent: HWND; const posRect: TRect): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT DoVerb(Longint iVerb, PMsg msg, const IOleClientSite activeSite,
Longint lindex, HWND hwndParent, const TRect posRect);
```

Description

This is DoVerb, a member of class TActiveXControl.

12.1.4.3.7 TActiveXControl.Draw Method

Pascal

```
function Draw(dwDrawAspect: Longint; lindex: Longint; pvAspect: Pointer; ptd: PDVTargetDevice;
hicTargetDev: HDC; hdcDraw: HDC; prcBounds: PRect; prcWBounds: PRect; fnContinue:
TContinueFunc; dwContinue: Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Draw(Longint dwDrawAspect, Longint lindex, void * pvAspect,
PDVTargetDevice ptd, HDC hicTargetDev, HDC hdcDraw, PRect prcBounds, PRect prcWBounds,
TContinueFunc fnContinue, Longint dwContinue);
```

Description

This is Draw, a member of class TActiveXControl.

12.1.4.3.8 TActiveXControl.DUnadvise Method

Pascal

```
function DUnadvise(dwConnection: Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT DUnadvise(Longint dwConnection);
```

Description

This is DUnadvise, a member of class TActiveXControl.

12.1.4.3.9 TActiveXControl.EnableModeless Method

Pascal

```
function EnableModeless(fEnable: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT EnableModeless(BOOL fEnable);
```

Description

This is EnableModeless, a member of class TActiveXControl.

12.1.4.3.10 TActiveXControl.EnumAdvise Method

Pascal

```
function EnumAdvise(out enumAdvise: IEnumStatData): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT EnumAdvise(IEnumStatData enumAdvise);
```

Description

This is EnumAdvise, a member of class TActiveXControl.

12.1.4.3.11 TActiveXControl.EnumDAdvise Method

Pascal

```
function EnumDAdvise(out enumAdvise: IEnumStatData): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT EnumDAdvise(IEnumStatData enumAdvise);
```

Description

This is EnumDAdvise, a member of class TActiveXControl.

12.1.4.3.12 TActiveXControl.EnumFormatEtc Method

Pascal

```
function EnumFormatEtc(dwDirection: Longint; out enumFormatEtc: IEnumFormatEtc): HResult;
stdcall;
```

C++

```
__fastcall __stdcall HRESULT EnumFormatEtc(Longint dwDirection, IEnumFormatEtc enumFormatEtc);
```

Description

This is EnumFormatEtc, a member of class TActiveXControl.

12.1.4.3.13 TActiveXControl.EnumVerbs Method

Pascal

```
function EnumVerbs(out enumOleVerb: IEnumOleVerb): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT EnumVerbs(IEnumOleVerb enumOleVerb);
```

Description

This is EnumVerbs, a member of class TActiveXControl.

12.1.4.3.14 TActiveXControl.Freeze Method

Pascal

```
function Freeze(dwDrawAspect: Longint; lindex: Longint; pvAspect: Pointer; out dwFreeze:
Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Freeze(Longint dwDrawAspect, Longint lindex, void * pvAspect,
Longint dwFreeze);
```

Description

This is Freeze, a member of class TActiveXControl.

12.1.4.3.15 TActiveXControl.FreezeEvents Method

Pascal

```
function FreezeEvents(bFreeze: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT FreezeEvents(BOOL bFreeze);
```

Description

This is FreezeEvents, a member of class TActiveXControl.

12.1.4.3.16 TActiveXControl.Get_BackColor Method

Pascal

```
function Get_BackColor: Integer; safecall;
```

C++

```
__fastcall int Get_BackColor();
```

Description

This is Get_BackColor, a member of class TActiveXControl.

12.1.4.3.17 TActiveXControl.Get_Caption Method

Pascal

```
function Get_Caption: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Caption();
```

Description

This is Get_Caption, a member of class TActiveXControl.

12.1.4.3.18 TActiveXControl.Get_Enabled Method

Pascal

```
function Get_Enabled: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_Enabled();
```

Description

This is Get_Enabled, a member of class TActiveXControl.

12.1.4.3.19 TActiveXControl.Get_Font Method

Pascal

```
function Get_Font: Font; safecall;
```

C++

```
__fastcall Font Get_Font();
```

Description

This is Get_Font, a member of class TActiveXControl.

12.1.4.3.20 TActiveXControl.Get_ForeColor Method

Pascal

```
function Get_ForeColor: Integer; safecall;
```

C++

```
__fastcall int Get_ForeColor();
```

Description

This is Get_ForeColor, a member of class TActiveXControl.

12.1.4.3.21 TActiveXControl.Get_HWnd Method

Pascal

```
function Get_HWnd: Integer; safecall;
```

C++

```
__fastcall int Get_HWnd();
```

Description

This is Get_HWnd, a member of class TActiveXControl.

12.1.4.3.22 TActiveXControl.Get_TabStop Method

Pascal

```
function Get_TabStop: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_TabStop();
```

Description

This is Get_TabStop, a member of class TActiveXControl.

12.1.4.3.23 TActiveXControl.Get_Text Method

Pascal

```
function Get_Text: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Text();
```

Description

This is Get_Text, a member of class TActiveXControl.

12.1.4.3.24 TActiveXControl.GetAdvise Method

Pascal

```
function GetAdvise(pAspects: PLongint; pAdvf: PLONGINT; out advSink: IAdviseSink): HResult;
stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetAdvise(PLongint pAspects, PLONGINT pAdvf, IAdviseSink advSink);
```

Description

This is GetAdvise, a member of class TActiveXControl.

12.1.4.3.25 TActiveXControl.GetCanonicalFormatEtc Method

Pascal

```
function GetCanonicalFormatEtc(const formatetc: TFormatEtc; out formatetcOut: TFormatEtc):
HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetCanonicalFormatEtc(const TFormatEtc formatetc, TFormatEtc
formatetcOut);
```

Description

This is GetCanonicalFormatEtc, a member of class TActiveXControl.

12.1.4.3.26 TActiveXControl.GetClassID Method

Pascal

```
function GetClassID(out classID: TCLSID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetClassID(TCLSID classID);
```

Description

This is GetClassID, a member of class TActiveXControl.

12.1.4.3.27 TActiveXControl.GetClientSite Method

Pascal

```
function GetClientSite(out clientSite: IOleClientSite): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetClientSite(IOleClientSite clientSite);
```

Description

This is GetClientSite, a member of class TActiveXControl.

12.1.4.3.28 TActiveXControl.GetClipboardData Method

Pascal

```
function GetClipboardData(dwReserved: Longint; out dataObject: IDataObject): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetClipboardData(Longint dwReserved, IDataObject dataObject);
```

Description

This is GetClipboardData, a member of class TActiveXControl.

12.1.4.3.29 TActiveXControl.GetColorSet Method

Pascal

```
function GetColorSet(dwDrawAspect: Longint; lindex: Longint; pvAspect: Pointer; ptd:
PDVTargetDevice; hicTargetDev: HDC; out colorSet: PLogPalette): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetColorSet(Longint dwDrawAspect, Longint lindex, void *
pvAspect, PDVTargetDevice ptd, HDC hicTargetDev, PLogPalette colorSet);
```

Description

This is GetColorSet, a member of class TActiveXControl.

12.1.4.3.30 TActiveXControl.GetContentExtent Method

Pascal

```
function GetContentExtent(out sizel: TPoint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetContentExtent(TPoint sizel);
```

Description

This is GetContentExtent, a member of class TActiveXControl.

12.1.4.3.31 TActiveXControl.GetControlInfo Method

Pascal

```
function GetControlInfo(var ci: TControlInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetControlInfo(TControlInfo ci);
```

Description

This is GetControlInfo, a member of class TActiveXControl.

12.1.4.3.32 TActiveXControl.GetData Method

Pascal

```
function GetData(const formatetcIn: TFormatEtc; out medium: TStgMedium): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetData(const TFormatEtc formatetcIn, TStgMedium medium);
```

Description

This is GetData, a member of class TActiveXControl.

12.1.4.3.33 TActiveXControl.GetDataHere Method

Pascal

```
function GetDataHere(const formatetc: TFormatEtc; out medium: TStgMedium): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetDataHere(const TFormatEtc formatetc, TStgMedium medium);
```

Description

This is GetDataHere, a member of class TActiveXControl.

12.1.4.3.34 TActiveXControl.GetDisplayString Method

Pascal

```
function GetDisplayString(dispid: TDispID; out bstr: WideString): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetDisplayString(TDispID dispid, BSTR bstr);
```

Description

This is GetDisplayString, a member of class TActiveXControl.

12.1.4.3.35 TActiveXControl.GetExtent Method

Pascal

```
function GetExtent(dwDrawAspect: Longint; out size: TPoint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetExtent(Longint dwDrawAspect, TPoint size);
```

Description

This is GetExtent, a member of class TActiveXControl.

12.1.4.3.36 TActiveXControl.GetInterfaceSafetyOptions Method

Pascal

```
function GetInterfaceSafetyOptions(const IID: TIID; pdwSupportedOptions: PDWORD;
pdwEnabledOptions: PDWORD): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT GetInterfaceSafetyOptions(const TIID IID, PDWORD
pdwSupportedOptions, PDWORD pdwEnabledOptions);
```

Description

This is GetInterfaceSafetyOptions, a member of class TActiveXControl.

12.1.4.3.37 TActiveXControl.GetMiscStatus Method

Pascal

```
function GetMiscStatus(dwAspect: Longint; out dwStatus: Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetMiscStatus(Longint dwAspect, Longint dwStatus);
```

Description

This is GetMiscStatus, a member of class TActiveXControl.

12.1.4.3.38 TActiveXControl.GetMoniker Method

Pascal

```
function GetMoniker(dwAssign: Longint; dwWhichMoniker: Longint; out mk: IMoniker): HResult;
stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetMoniker(Longint dwAssign, Longint dwWhichMoniker, IMoniker mk);
```

Description

This is GetMoniker, a member of class TActiveXControl.

12.1.4.3.39 TActiveXControl.GetPages Method

Pascal

```
function GetPages(out pages: TCAGUID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetPages(TCAGUID pages);
```

Description

This is GetPages, a member of class TActiveXControl.

12.1.4.3.40 TActiveXControl.GetPredefinedStrings Method

Pascal

```
function GetPredefinedStrings(dispid: TDispID; out caStringsOut: TCAPOleStr; out caCookiesOut:
```

```
TCALongint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetPredefinedStrings(TDispID dispid, TCAPOleStr caStringsOut,
TCALongint caCookiesOut);
```

Description

This is GetPredefinedStrings, a member of class TActiveXControl.

12.1.4.3.41 TActiveXControl.GetPredefinedValue Method

Pascal

```
function GetPredefinedValue(dispid: TDispID; dwCookie: Longint; out varOut: OleVariant):
HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetPredefinedValue(TDispID dispid, Longint dwCookie, OleVariant
varOut);
```

Description

This is GetPredefinedValue, a member of class TActiveXControl.

12.1.4.3.42 TActiveXControl.GetPropertyString Method

Pascal

```
function GetPropertyString(DispID: Integer; var S: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetPropertyString(int DispID, AnsiString S);
```

Description

This is GetPropertyString, a member of class TActiveXControl.

12.1.4.3.43 TActiveXControl.GetPropertyStrings Method

Pascal

```
function GetPropertyStrings(DispID: Integer; Strings: TStrings): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetPropertyStrings(int DispID, TStrings Strings);
```

Description

This is GetPropertyStrings, a member of class TActiveXControl.

12.1.4.3.44 TActiveXControl.GetPropertyValue Method

Pascal

```
procedure GetPropertyValue(DispID: Integer; Cookie: Integer; var Value: OleVariant); virtual;
```

C++

```
virtual __fastcall GetPropertyValue(int DispID, int Cookie, OleVariant Value);
```

Description

This is GetPropertyValue, a member of class TActiveXControl.

12.1.4.3.45 TActiveXControl.GetPropFromBag Method

Pascal

```
procedure GetPropFromBag(const PropName: WideString; DispatchID: Integer; PropBag:
IPropertyBag; ErrorLog: IErrorLog); virtual;
```

C++

```
virtual __fastcall GetPropFromBag(const BSTR PropName, int DispatchID, IPropertyBag PropBag,
IErrorLog ErrorLog);
```

Description

This is GetPropFromBag, a member of class TActiveXControl.

12.1.4.3.46 TActiveXControl.GetSizeMax Method

Pascal

```
function GetSizeMax(out cbSize: Largeint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetSizeMax(Largeint cbSize);
```

Description

This is GetSizeMax, a member of class TActiveXControl.

12.1.4.3.47 TActiveXControl.GetUserClassID Method

Pascal

```
function GetUserClassID(out clsid: TCLSID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetUserClassID(TCLSID clsid);
```

Description

This is GetUserClassID, a member of class TActiveXControl.

12.1.4.3.48 TActiveXControl.GetUserType Method

Pascal

```
function GetUserType(dwFormOfType: Longint; out pszUserType: POleStr): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetUserType(Longint dwFormOfType, POleStr pszUserType);
```

Description

This is GetUserType, a member of class TActiveXControl.

12.1.4.3.49 TActiveXControl.GetWindow Method

Pascal

```
function GetWindow(out wnd: HWnd): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetWindow(HWnd wnd);
```

Description

This is GetWindow, a member of class TActiveXControl.

12.1.4.3.50 TActiveXControl.HandsOffStorage Method

Pascal

```
function HandsOffStorage: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT HandsOffStorage();
```

Description

This is HandsOffStorage, a member of class TActiveXControl.

12.1.4.3.51 TActiveXControl.InitFromData Method

Pascal

```
function InitFromData(const dataObject: IDataObject; fCreation: BOOL; dwReserved: Longint):
HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT InitFromData(const IDataObject dataObject, BOOL fCreation,
Longint dwReserved);
```

Description

This is InitFromData, a member of class TActiveXControl.

12.1.4.3.52 TActiveXControl.Initialize Method

Creates the class factory and connection object for the TActiveXControl (see page 1517).

Pascal

```
procedure Initialize; override;
```

C++

```
virtual __fastcall Initialize();
```

Description

The Initialize method creates the class factory object and creates and encapsulates the VCL control.

See Also

InitializeControl (see page 1513)

12.1.4.3.53 TActiveXControl.InitializeControl Method

Pascal

```
procedure InitializeControl; virtual;
```

C++

```
virtual __fastcall InitializeControl();
```

Description

This is InitializeControl, a member of class TActiveXControl.

12.1.4.3.54 TActiveXControl.InitNew Method

Pascal

```
function InitNew: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT InitNew();
```

Description

This is InitNew, a member of class TActiveXControl.

12.1.4.3.55 TActiveXControl.InPlaceActivate Method

Pascal

```
function InPlaceActivate(ActivateUI: Boolean): HResult;
```

C++

```
__fastcall HRESULT InPlaceActivate(Boolean ActivateUI);
```

Description

This is InPlaceActivate, a member of class TActiveXControl.

12.1.4.3.56 TActiveXControl.InPlaceDeactivate Method

Pascal

```
function InPlaceDeactivate: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT InPlaceDeactivate();
```

Description

This is InPlaceDeactivate, a member of class TActiveXControl.

12.1.4.3.57 TActiveXControl.IsDirty Method

Pascal

```
function IsDirty: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT IsDirty();
```

Description

This is IsDirty, a member of class TActiveXControl.

12.1.4.3.58 TActiveXControl.IsUpToDate Method

Pascal

```
function IsUpToDate: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT IsUpToDate();
```

Description

This is IsUpToDate, a member of class TActiveXControl.

12.1.4.3.59 TActiveXControl.LoadFromStream Method

Pascal

```
procedure LoadFromStream(const Stream: IStream); virtual;
```

C++

```
virtual __fastcall LoadFromStream(const IStream Stream);
```

Description

This is LoadFromStream, a member of class TActiveXControl.

12.1.4.3.60 TActiveXControl.MapPropertyToPage Method

Pascal

```
function MapPropertyToPage(dispid: TDispID; out clsid: TCLSID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT MapPropertyToPage(TDispID dispid, TCLSID clsid);
```

Description

This is MapPropertyToPage, a member of class TActiveXControl.

12.1.4.3.61 TActiveXControl.ObjQueryInterface Method

Returns a requested COM interface on the object.

Pascal

```
function ObjQueryInterface(const IID: TGUID; out Obj): HResult; override;
```

C++

```
virtual __fastcall HRESULT ObjQueryInterface(const TGUID IID,  Obj);
```

Description

ObjQueryInterface attempts to locate the interface specified by the IID parameter, and returns it in the Obj parameter. If the requested interface is found and returned, ObjQueryInterface returns S_OK.

See Also

TActiveForm (see page 1489)

TActiveFormFactory (see page 1514)

12.1.4.3.62 TActiveXControl.OnAmbientPropertyChange Method

Pascal

```
function OnAmbientPropertyChange(dispid: TDispID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnAmbientPropertyChange(TDispID dispid);
```

Description

This is OnAmbientPropertyChange, a member of class TActiveXControl.

12.1.4.3.63 TActiveXControl.OnDocWindowActivate Method

Pascal

```
function OnDocWindowActivate(fActivate: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnDocWindowActivate(BOOL fActivate);
```

Description

This is OnDocWindowActivate, a member of class TActiveXControl.

12.1.4.3.64 TActiveXControl.OnFrameWindowActivate Method

Pascal

```
function OnFrameWindowActivate(fActivate: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnFrameWindowActivate(BOOL fActivate);
```

Description

This is OnFrameWindowActivate, a member of class TActiveXControl.

12.1.4.3.65 TActiveXControl.OnMnemonic Method

Pascal

```
function OnMnemonic(msg: PMsg): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnMnemonic(PMsg msg);
```

Description

This is OnMnemonic, a member of class TActiveXControl.

12.1.4.3.66 TActiveXControl.PerformVerb Method

Pascal

```
procedure PerformVerb(Verb: Integer); virtual;
```

C++

```
virtual __fastcall PerformVerb(int Verb);
```

Description

This is PerformVerb, a member of class TActiveXControl.

12.1.4.3.67 TActiveXControl.PersistPropBagInitNew Method

Pascal

```
function PersistPropBagInitNew: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT PersistPropBagInitNew();
```

Description

This is PersistPropBagInitNew, a member of class TActiveXControl.

12.1.4.3.68 TActiveXControl.PersistPropBagLoad Method

Pascal

```
function PersistPropBagLoad(const pPropBag: IPropertyBag; const pErrorLog: IErrorLog):
HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT PersistPropBagLoad(const IPropertyBag pPropBag, const IErrorLog
pErrorLog);
```

Description

This is PersistPropBagLoad, a member of class TActiveXControl.

12.1.4.3.69 TActiveXControl.PersistPropBagSave Method

Pascal

```
function PersistPropBagSave(const pPropBag: IPropertyBag; fClearDirty: BOOL;
fSaveAllProperties: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT PersistPropBagSave(const IPropertyBag pPropBag, BOOL fClearDirty,
BOOL fSaveAllProperties);
```

Description

This is PersistPropBagSave, a member of class TActiveXControl.

12.1.4.3.70 TActiveXControl.PersistStorageInitNew Method

Pascal

```
function PersistStorageInitNew(const stg: IStorage): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT PersistStorageInitNew(const IStorage stg);
```

Description

This is PersistStorageInitNew, a member of class TActiveXControl.

12.1.4.3.71 TActiveXControl.PersistStorageLoad Method

Pascal

```
function PersistStorageLoad(const stg: IStorage): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT PersistStorageLoad(const IStorage stg);
```

Description

This is PersistStorageLoad, a member of class TActiveXControl.

12.1.4.3.72 TActiveXControl.PersistStorageSave Method

Pascal

```
function PersistStorageSave(const stgSave: IStorage; fSameAsLoad: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT PersistStorageSave(const IStorage stgSave, BOOL fSameAsLoad);
```

Description

This is PersistStorageSave, a member of class TActiveXControl.

12.1.4.3.73 TActiveXControl.PersistStreamLoad Method

Pascal

```
function PersistStreamLoad(const stm: IStream): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT PersistStreamLoad(const IStream stm);
```

Description

This is PersistStreamLoad, a member of class TActiveXControl.

12.1.4.3.74 TActiveXControl.PersistStreamSave Method

Pascal

```
function PersistStreamSave(const stm: IStream; fClearDirty: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT PersistStreamSave(const IStream stm, BOOL fClearDirty);
```

Description

This is PersistStreamSave, a member of class TActiveXControl.

12.1.4.3.75 TActiveXControl.PostMessageFilter Method

Pascal

```
function PostMessageFilter(wnd: HWnd; msg: Integer; wp: Integer; lp: Integer; out res:
Integer; Cookie: Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT PostMessageFilter(HWnd wnd, int msg, int wp, int lp, int res,
Longint Cookie);
```

Description

This is PostMessageFilter, a member of class TActiveXControl.

12.1.4.3.76 TActiveXControl.PreMessageFilter Method

Pascal

```
function PreMessageFilter(wnd: HWnd; msg: Integer; wp: Integer; lp: Integer; out res: Integer;
out Cookie: Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT PreMessageFilter(HWnd wnd, int msg, int wp, int lp, int res,
Longint Cookie);
```

Description

This is PreMessageFilter, a member of class TActiveXControl.

12.1.4.3.77 PropChanged Method

This is the overview for the PropChanged method overload.

Overload List

	Name	Description
	TActiveXControl.PropChanged (TDispID) (see page 1542)	Generates an OnChanged event.
	TActiveXControl.PropChanged (WideString) (see page 1542)	Generates an OnChanged event.

12.1.4.3.77.1 TActiveXControl.PropChanged Method (TDispID)

Generates an OnChanged event.

Pascal

```
procedure PropChanged(const PropertyName: WideString); overload;
procedure PropChanged(DispID: TDispID); overload;
```

C++

```
__fastcall PropChanged(const BSTR PropertyName);
__fastcall PropChanged(TDispID DispID);
```

Description

Call PropChanged when the property specified by the DispID (or PropertyName) parameter changes. PropChanged generates an OnChanged event using the IPropertyNotifySink interface.

When possible, call this procedure with the DispID rather than the PropertyName because it is more efficient. Calling PropChanged with the PropertyName first looks up the DispID from the given PropertyName before generating the event.

12.1.4.3.78 PropRequestEdit Method

This is the overview for the PropRequestEdit method overload.

Overload List

	Name	Description
	TActiveXControl.PropRequestEdit (TDispID) (see page 1543)	Indicates whether a specified property can be changed.
	TActiveXControl.PropRequestEdit (WideString) (see page 1543)	Indicates whether a specified property can be changed.

12.1.4.3.78.1 TActiveXControl.PropRequestEdit Method (TDispID)

Indicates whether a specified property can be changed.

Pascal

```
function PropRequestEdit(const PropertyName: WideString): Boolean; overload;
function PropRequestEdit(DispID: TDispID): Boolean; overload;
```

C++

```
__fastcall Boolean PropRequestEdit(const BSTR PropertyName);
__fastcall Boolean PropRequestEdit(TDispID DispID);
```

Description

Call PropRequestEdit to determine whether the property specified by the DispID (or PropertyName) parameter can be edited. PropRequestEdit uses the IPropertyNotifySink interface to determine whether the property can be changed, and returns True if it can.

When possible, call this function with the DispID rather than the PropertyName because it is more efficient. Calling PropRequestEdit with the PropertyName first looks up the DispID from the given PropertyName before generating the event.

12.1.4.3.79 TActiveXControl.PutPropInBag Method

Pascal

```
procedure PutPropInBag(const PropName: WideString; DispatchID: Integer; PropBag:
IPropertyBag); virtual;
```

C++

```
virtual __fastcall PutPropInBag(const BSTR PropName, int DispatchID, IPropertyBag PropBag);
```

Description

This is PutPropInBag, a member of class TActiveXControl.

12.1.4.3.80 TActiveXControl.QueryGetData Method

Pascal

```
function QueryGetData(const formatetc: TFormatEtc): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT QueryGetData(const TFormatEtc formatetc);
```

Description

This is QueryGetData, a member of class TActiveXControl.

12.1.4.3.81 TActiveXControl.QuickActivate Method

Pascal

```
function QuickActivate(var qaCont: tagQACONTAINER; var qaCtrl: tagQACONTROL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT QuickActivate(tagQACONTAINER qaCont, tagQACONTROL qaCtrl);
```

Description

This is QuickActivate, a member of class TActiveXControl.

12.1.4.3.82 TActiveXControl.ReactivateAndUndo Method

Pascal

```
function ReactivateAndUndo: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT ReactivateAndUndo();
```

Description

This is ReactivateAndUndo, a member of class TActiveXControl.

12.1.4.3.83 TActiveXControl.ResizeBorder Method

Pascal

```
function ResizeBorder(const rcBorder: TRect; const uiWindow: IOleInPlaceUIWindow;
fFrameWindow: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT ResizeBorder(const TRect rcBorder, const IOleInPlaceUIWindow
uiWindow, BOOL fFrameWindow);
```

Description

This is ResizeBorder, a member of class TActiveXControl.

12.1.4.3.84 TActiveXControl.SaveCompleted Method

Pascal

```
function SaveCompleted(const stgNew: IStorage): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SaveCompleted(const IStorage stgNew);
```

Description

This is SaveCompleted, a member of class TActiveXControl.

12.1.4.3.85 TActiveXControl.SaveToStream Method

Pascal

```
procedure SaveToStream(const Stream: IStream); virtual;
```

C++

```
virtual __fastcall SaveToStream(const IStream Stream);
```

Description

This is SaveToStream, a member of class TActiveXControl.

12.1.4.3.86 TActiveXControl.Set_BackColor Method

Pascal

```
procedure Set_BackColor(Value: Integer); safecall;
```

C++

```
__fastcall Set_BackColor(int Value);
```

Description

This is Set_BackColor, a member of class TActiveXControl.

12.1.4.3.87 TActiveXControl.Set_Caption Method

Pascal

```
procedure Set_Caption(const Value: WideString); safecall;
```

C++

```
__fastcall Set_Caption(const BSTR Value);
```

Description

This is Set_Caption, a member of class TActiveXControl.

12.1.4.3.88 TActiveXControl.Set_Enabled Method

Pascal

```
procedure Set_Enabled(Value: WordBool); safecall;
```

C++

```
__fastcall Set_Enabled(WordBool Value);
```

Description

This is Set_Enabled, a member of class TActiveXControl.

12.1.4.3.89 TActiveXControl.Set_Font Method

Pascal

```
procedure Set_Font(const Value: Font); safecall;
```

C++

```
__fastcall Set_Font(const Font Value);
```

Description

This is Set_Font, a member of class TActiveXControl.

12.1.4.3.90 TActiveXControl.Set_ForeColor Method

Pascal

```
procedure Set_ForeColor(Value: Integer); safecall;
```

C++

```
__fastcall Set_ForeColor(int Value);
```

Description

This is Set_ForeColor, a member of class TActiveXControl.

12.1.4.3.91 TActiveXControl.Set_TabStop Method

Pascal

```
procedure Set_TabStop(Value: WordBool); safecall;
```

C++

```
__fastcall Set_TabStop(WordBool Value);
```

Description

This is Set_TabStop, a member of class TActiveXControl.

12.1.4.3.92 TActiveXControl.Set_Text Method

Pascal

```
procedure Set_Text(const Value: WideString); safecall;
```

C++

```
__fastcall Set_Text(const BSTR Value);
```

Description

This is Set_Text, a member of class TActiveXControl.

12.1.4.3.93 TActiveXControl.SetAdvise Method

Pascal

```
function SetAdvise(aspects: Longint; advf: Longint; const advSink: IAdviseSink): HResult;
stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetAdvise(Longint aspects, Longint advf, const IAdviseSink
advSink);
```

Description

This is SetAdvise, a member of class TActiveXControl.

12.1.4.3.94 TActiveXControl.SetClientSite Method

Pascal

```
function SetClientSite(const clientSite: IOleClientSite): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetClientSite(const IOleClientSite clientSite);
```

Description

This is SetClientSite, a member of class TActiveXControl.

12.1.4.3.95 TActiveXControl.SetColorScheme Method

Pascal

```
function SetColorScheme(const logpal: TLogPalette): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetColorScheme(const TLogPalette logpal);
```

Description

This is SetColorScheme, a member of class TActiveXControl.

12.1.4.3.96 TActiveXControl.SetContentExtent Method

Pascal

```
function SetContentExtent(const sizel: TPoint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetContentExtent(const TPoint sizel);
```

Description

This is SetContentExtent, a member of class TActiveXControl.

12.1.4.3.97 TActiveXControl.SetData Method

Pascal

```
function SetData(const formatetc: TFormatEtc; var medium: TStgMedium; fRelease: BOOL):
HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetData(const TFormatEtc formatetc, TStgMedium medium, BOOL
fRelease);
```

Description

This is SetData, a member of class TActiveXControl.

12.1.4.3.98 TActiveXControl.SetExtent Method

Pascal

```
function SetExtent(dwDrawAspect: Longint; const size: TPoint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetExtent(Longint dwDrawAspect, const TPoint size);
```

Description

This is SetExtent, a member of class TActiveXControl.

12.1.4.3.99 TActiveXControl.SetHostNames Method

Pascal

```
function SetHostNames(szContainerApp: POleStr; szContainerObj: POleStr): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetHostNames(POleStr szContainerApp, POleStr szContainerObj);
```

Description

This is SetHostNames, a member of class TActiveXControl.

12.1.4.3.100 TActiveXControl.SetInPlaceSite Method

Pascal

```
procedure SetInPlaceSite(const NewInPlaceSite: IOleInPlaceSite);
```

C++

```
__fastcall SetInPlaceSite(const IOleInPlaceSite NewInPlaceSite);
```

Description

This is SetInPlaceSite, a member of class TActiveXControl.

12.1.4.3.101 TActiveXControl.SetInterfaceSafetyOptions Method

Pascal

```
function SetInterfaceSafetyOptions(const IID: TIID; dwOptionSetMask: DWORD; dwEnabledOptions:
DWORD): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT SetInterfaceSafetyOptions(const TIID IID, DWORD
dwOptionSetMask, DWORD dwEnabledOptions);
```

Description

This is SetInterfaceSafetyOptions, a member of class TActiveXControl.

12.1.4.3.102 TActiveXControl.SetMoniker Method

Pascal

```
function SetMoniker(dwWhichMoniker: Longint; const mk: IMoniker): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetMoniker(Longint dwWhichMoniker, const IMoniker mk);
```

Description

This is SetMoniker, a member of class TActiveXControl.

12.1.4.3.103 TActiveXControl.SetObjectRects Method

Pascal

```
function SetObjectRects(const rcPosRect: TRect; const rcClipRect: TRect): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetObjectRects(const TRect rcPosRect, const TRect rcClipRect);
```

Description

This is SetObjectRects, a member of class TActiveXControl.

12.1.4.3.104 TActiveXControl.ShowPropertyDialog Method

Pascal

```
procedure ShowPropertyDialog;
```

C++

```
__fastcall ShowPropertyDialog();
```

Description

This is ShowPropertyDialog, a member of class TActiveXControl.

12.1.4.3.105 TActiveXControl.StdClickEvent Method

Pascal

```
procedure StdClickEvent(Sender: TObject);
```

C++

```
__fastcall StdClickEvent(TObject * Sender);
```

Description

This is StdClickEvent, a member of class TActiveXControl.

12.1.4.3.106 TActiveXControl.StdDblClickEvent Method

Pascal

```
procedure StdDblClickEvent(Sender: TObject);
```

C++

```
__fastcall StdDblClickEvent(TObject * Sender);
```

Description

This is StdDblClickEvent, a member of class TActiveXControl.

12.1.4.3.107 TActiveXControl.StdKeyDownEvent Method

Pascal

```
procedure StdKeyDownEvent(Sender: TObject; var Key: Word; Shift: TShiftState);
```

C++

```
__fastcall StdKeyDownEvent(TObject * Sender, Word Key, TShiftState Shift);
```

Description

This is StdKeyDownEvent, a member of class TActiveXControl.

12.1.4.3.108 TActiveXControl.StdKeyPressEvent Method

Pascal

```
procedure StdKeyPressEvent(Sender: TObject; var Key: Char);
```

C++

```
__fastcall StdKeyPressEvent(TObject * Sender, Char Key);
```

Description

This is StdKeyPressEvent, a member of class TActiveXControl.

12.1.4.3.109 TActiveXControl.StdKeyUpEvent Method

Pascal

```
procedure StdKeyUpEvent(Sender: TObject; var Key: Word; Shift: TShiftState);
```

C++

```
__fastcall StdKeyUpEvent(TObject * Sender, Word Key, TShiftState Shift);
```

Description

This is StdKeyUpEvent, a member of class TActiveXControl.

12.1.4.3.110 TActiveXControl.StdMouseDownEvent Method

Pascal

```
procedure StdMouseDownEvent(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X:
Integer; Y: Integer);
```

C++

```
__fastcall StdMouseDownEvent(TObject * Sender, TMouseButton Button, TShiftState Shift, int X,
int Y);
```

Description

This is StdMouseDownEvent, a member of class TActiveXControl.

12.1.4.3.111 TActiveXControl.StdMouseMoveEvent Method

Pascal

```
procedure StdMouseMoveEvent(Sender: TObject; Shift: TShiftState; X: Integer; Y: Integer);
```

C++

```
__fastcall StdMouseMoveEvent(TObject * Sender, TShiftState Shift, int X, int Y);
```

Description

This is StdMouseMoveEvent, a member of class TActiveXControl.

12.1.4.3.112 TActiveXControl.StdMouseUpEvent Method

Pascal

```
procedure StdMouseUpEvent(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X:
Integer; Y: Integer);
```

C++

```
__fastcall StdMouseUpEvent(TObject * Sender, TMouseButton Button, TShiftState Shift, int X,
int Y);
```

Description

This is StdMouseUpEvent, a member of class TActiveXControl.

12.1.4.3.113 TActiveXControl.TranslateAccelerator Method

Pascal

```
function TranslateAccelerator(var msg: TMsg): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT TranslateAccelerator(TMsg msg);
```

Description

This is TranslateAccelerator, a member of class TActiveXControl.

12.1.4.3.114 TActiveXControl.UIDeactivate Method

Pascal

```
function UIDeactivate: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT UIDeactivate();
```

Description

This is UIDeactivate, a member of class TActiveXControl.

12.1.4.3.115 TActiveXControl.Unadvise Method

Pascal

```
function Unadvise(dwConnection: Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Unadvise(Longint dwConnection);
```

Description

This is Unadvise, a member of class TActiveXControl.

12.1.4.3.116 TActiveXControl.Unfreeze Method

Pascal

```
function Unfreeze(dwFreeze: Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Unfreeze(Longint dwFreeze);
```

Description

This is Unfreeze, a member of class TActiveXControl.

12.1.4.3.117 TActiveXControl.Update Method

Pascal

```
function Update: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Update();
```

Description

This is Update, a member of class TActiveXControl.

12.1.4.3.118 TActiveXControl.ViewObjectGetExtent Method

Pascal

```
function ViewObjectGetExtent(dwDrawAspect: Longint; lindex: Longint; ptd: PDVTargetDevice; out
size: TPoint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT ViewObjectGetExtent(Longint dwDrawAspect, Longint lindex,
PDVTargetDevice ptd, TPoint size);
```

Description

This is ViewObjectGetExtent, a member of class TActiveXControl.

12.1.4.3.119 TActiveXControl.WndProc Method

Pascal

```
procedure WndProc(var Message: TMessage); virtual;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

This is WndProc, a member of class TActiveXControl.

12.1.4.4 TActiveXControl Properties

The properties of the TActiveXControl class are listed here.

Protected Properties

	Name	Description
R	ConnectionPoints (see page 1553)	This is ConnectionPoints, a member of class TActiveXControl.

Public Properties

	Name	Description
R	ClientSite (see page 1553)	Contains a pointer to the client site for the container on which the ActiveX control resides.
R	Control (see page 1553)	Indicates the VCL implementation of the ActiveX control.
R	InPlaceSite (see page 1554)	Contains a pointer to the site interface for the container on which the ActiveX control resides.

12.1.4.4.1 TActiveXControl.ClientSite Property

Contains a pointer to the client site for the container on which the ActiveX control resides.

Pascal

```
property ClientSite: IOleClientSite;
```

C++

```
__property IOleClientSite ClientSite;
```

Description

ClientSite contains an interface pointer to the object's client site. Within a compound document, each embedded object has its own client site. The client site is the place where it is displayed and through which it retrieves information about its storage, user interface, and other resources.

Use this property when you need to get client site information from the container.

See Also

InPlaceSite (see page 1554)

12.1.4.4.2 TActiveXControl.ConnectionPoints Property

Pascal

```
property ConnectionPoints: TConnectionPoints;
```

C++

```
__property TConnectionPoints ConnectionPoints;
```

Description

This is ConnectionPoints, a member of class TActiveXControl.

12.1.4.4.3 TActiveXControl.Control Property

Indicates the VCL implementation of the ActiveX control.

Pascal

```
property Control: TWinControl;
```

C++

```
__property TWinControl * Control;
```

Description

Control is the VCL control derived from TWinControl that implements the ActiveX control.

12.1.4.4.4 TActiveXControl.InPlaceSite Property

Contains a pointer to the site interface for the container on which the ActiveX control resides.

Pascal

```
property InPlaceSite: IOleInPlaceSite;
```

C++

```
__property IOleInPlaceSite InPlaceSite;
```

Description

InPlaceSite contains an interface pointer to the container's IOleInPlaceSite interface when the container supports embedding and in-place activation. Use this interface to access the windows that contain the activeX object (both the Frame window and the immediate container, or site, window).

See Also

ClientSite (see page 1553)

12.1.5 TActiveXControlFactory Class

TActiveXControlFactory is the factory class for ActiveX control objects.

Class Hierarchy

Pascal

```
TActiveXControlFactory = class(TAutoObjectFactory);
```

C++

```
class TActiveXControlFactory : public TAutoObjectFactory;
```

File

AxCtrls

Description

An ActiveX control is a COM object, which means the module it is implemented in must provide a factory object for the system to be able to create it. The TActiveXControlFactory class is a factory for ActiveX controls, which are implemented using the TActiveXControl (see page 1517) abstract class.

For more information about COM factories, see TComObjectFactory.

The ActiveX Control wizard adds To add an ActiveX control factory to your program, create the factory object in the initialization section of the unit that defines the TActiveXControl (see page 1517) object.

See Also

TActiveXControl (see page 1517)

TComObjectFactory (see page 5563)

12.1.5.1 TActiveXControlFactory Members

The following tables list the members exposed by TActiveXControlFactory.

Protected Methods

	Name	Description
	_AddRef (see page 5567)	This is _AddRef, a member of class TComObjectFactory.
	_Release (see page 5567)	This is _Release, a member of class TComObjectFactory.
	CreateInstance (see page 5568)	This is CreateInstance, a member of class TComObjectFactory.
	CreateInstanceLic (see page 5568)	This is CreateInstanceLic, a member of class TComObjectFactory.
V	GetLicenseFileName (see page 1558)	This is GetLicenseFileName, a member of class TActiveXControlFactory.
V	GetLicenseString (see page 5568)	This is GetLicenseString, a member of class TComObjectFactory.
	GetLicInfo (see page 5568)	This is GetLicInfo, a member of class TComObjectFactory.
V	GetProgID (see page 5569)	This is GetProgID, a member of class TComObjectFactory.
V	HasMachineLicense (see page 1558)	This is HasMachineLicense, a member of class TActiveXControlFactory.
	LockServer (see page 5569)	This is LockServer, a member of class TComObjectFactory.
	QueryInterface (see page 5569)	This is QueryInterface, a member of class TComObjectFactory.
	RequestLicKey (see page 5570)	This is RequestLicKey, a member of class TComObjectFactory.
V	ValidateUserLicense (see page 5571)	This is ValidateUserLicense, a member of class TComObjectFactory.

Public Methods

	Name	Description
	AddVerb (see page 1557)	AddVerb adds a verb to the object's list of supported verbs.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1556)	Creates and initializes a new Create object.
V	CreateComObject (see page 5567)	Instantiates a TComObject (see page 5553) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 1557)	Frees the instance of the Destroy.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetInterfaceTypeInfo (see page 5586)	Retrieves the type information for the typed COM object created by the factory.
V	GetIntfEntry (see page 5548)	Returns the entry for a specific interface implemented in a class.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RegisterClassObject (see page 5569)	Registers the COM class associated with the factory object.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	UpdateRegistry (see page 1559)	UpdateRegistry adds or removes registry entries for the object.

Public Properties

	Name	Description
R	ClassID (see page 5571)	This is ClassID, a member of class TComObjectFactory.
R	ClassInfo (see page 5587)	Specifies an interface reference that provides type information without having to load a type library.
R	ClassName (see page 5572)	Specifies the name of the class that the class factory instantiates.
R	ComClass (see page 5572)	This is ComClass, a member of class TComObjectFactory.
R	ComServer (see page 5572)	This is ComServer, a member of class TComObjectFactory.
R	Description (see page 5572)	Describes the class object.
R	DispIntfEntry (see page 5549)	Contains the entry for the dispatch interface implemented by the OLE Automation object created by the factory.
R	DispTypeInfo (see page 5549)	Returns an ITypeInfo interface reference to the type information for the dispinterface implemented by the OLE Automation object.
	ErrorIID (see page 5573)	Indicates the interface of the CoClass in which an error occurred.
R	EventIID (see page 5549)	This is EventIID, a member of class TAutoObjectFactory.
R	EventTypeInfo (see page 5550)	This is EventTypeInfo, a member of class TAutoObjectFactory.
R	Instancing (see page 5573)	Indicates the process space of COM object instances.
	LicString (see page 5573)	Contains licensing information about the object.
R	MiscStatus (see page 1559)	Represents a set of status bits that are stored in the registry.
R	ProgID (see page 5574)	Specifies the name of the class factory.
	ShowErrors (see page 5574)	Indicates whether the COM object factory displays a message box when exceptions are raised.
	SupportsLicensing (see page 5574)	Indicates whether the object being instantiated supports licensing.
R	ThreadingModel (see page 5575)	Indicates the type of thread support provided by objects that the factory creates.
R	ToolboxBitmapID (see page 1560)	Designates the resource identifier of the ActiveX control.
R	WinControlClass (see page 1560)	WinControlClass is the class of the VCL control that implements the Windows behavior for the ActiveX control.

12.1.5.2 TActiveXControlFactory.Create Constructor

Creates and initializes a new Create object.

Pascal

```
constructor Create(ComServer: TComServerObject; ActiveXControlClass: TActiveXControlClass;
WinControlClass: TWinControlClass; const ClassID: TGUID; ToolboxBitmapID: Integer; const
LicStr: string; MiscStatus: Integer; ThreadingModel: TThreadingModel = tmSingle);
```

C++

```
__fastcall TActiveXControlFactory(TComServerObject ComServer, TActiveXControlClass
ActiveXControlClass, TWinControlClass WinControlClass, const TGUID ClassID, int
ToolboxBitmapID, const AnsiString LicStr, int MiscStatus, TThreadingModel ThreadingModel =
tmSingle);
```

Description

The ComServer parameter is a pointer to the COM server object for the module containing the property page factory. This is

normally a global variable, ComServer, declared in the ComServ (see page 5621) unit.

The ActiveXControlClass parameter is the class object for the controller object that implements the ActiveX control. This object should be derived from TActiveXControl (see page 1517).

The WinControlClass (see page 1560) parameter is the class object for the VCL control class that implements the Windows behavior of the control. This object should be derived from TWinControl.

The ClassID parameter is the GUID of the ActiveX control. Each COM object should have a unique GUID.

LicStr is the license string for controls created with this factory. LicStr is empty if controls created with this factory are unlicensed. MiscStatus (see page 1559) are the status bits that are stored in the registry.

ThreadingModel specifies how calls to the created ActiveX control are serialized. This determines the value of the ThreadingModel property.

See Also

ToolboxBitmapID (see page 1560)

MiscStatus (see page 1559)

12.1.5.3 TActiveXControlFactory.Destroy Destructor

Frees the instance of the Destroy.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TActiveXControlFactory();
```

Description

Do not call Destroy ~Destroy directly in an application. Factory objects are destroyed automatically when the server shuts down.

See Also

Freedelete_keyword

12.1.5.4 TActiveXControlFactory Methods

The methods of the TActiveXControlFactory class are listed here.

Protected Methods

	Name	Description
	GetLicenseFileName (see page 1558)	This is GetLicenseFileName, a member of class TActiveXControlFactory.
	HasMachineLicense (see page 1558)	This is HasMachineLicense, a member of class TActiveXControlFactory.

Public Methods

	Name	Description
	AddVerb (see page 1557)	AddVerb adds a verb to the object's list of supported verbs.
	UpdateRegistry (see page 1559)	UpdateRegistry adds or removes registry entries for the object.

12.1.5.4.1 TActiveXControlFactory.AddVerb Method

AddVerb adds a verb to the object's list of supported verbs.

Pascal

```
procedure AddVerb(Verb: Integer; const VerbName: string);
```

C++

```
__fastcall AddVerb(int Verb, const AnsiString VerbName);
```

Description

Use AddVerb to add a verb that host applications can apply to the ActiveX control. In the Verb parameter, pass a number to identify the verb. User-defined verbs are normally small positive numbers.

In the VerbName parameter, pass the text to display when presenting the user with a choice of verbs. In some ActiveX (formerly called OLE) containers, the verb text is displayed in a popup menu when the user right-clicks on a control.

Placing an ampersand (&) before a character in this string causes the character to be displayed underlined in a pop-up menu.

ActiveX defines the following commonly used verbs:

VerbName	Verb
OLEIVERB_PRIMARY	0
OLEIVERB_SHOW	–1
OLEIVERB_OPEN	–2
OLEIVERB_HIDE	–3
OLEIVERB_UIACTIVATE	–4
OLEIVERB_INPLACEACTIVATE	–5
OLEIVERB_DISCARDUNDOSTATE	–6
OLEIVERB_PROPERTIES	–7

See Also

UpdateRegistry (see page 1559)

12.1.5.4.2 TActiveXControlFactory.GetLicenseFileName Method

Pascal

```
function GetLicenseFileName: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetLicenseFileName();
```

Description

This is GetLicenseFileName, a member of class TActiveXControlFactory.

12.1.5.4.3 TActiveXControlFactory.HasMachineLicense Method

Pascal

```
function HasMachineLicense: Boolean; override;
```

C++

```
virtual __fastcall Boolean HasMachineLicense();
```

Description

This is HasMachineLicense, a member of class TActiveXControlFactory.

12.1.5.4.4 TActiveXControlFactory.UpdateRegistry Method

UpdateRegistry adds or removes registry entries for the object.

Pascal

```
procedure UpdateRegistry(Register: Boolean); override;
```

C++

```
virtual __fastcall UpdateRegistry(Boolean Register);
```

Description

This method is called to add or remove the control from the system registry. The default behavior of this method is to add normal ActiveX control registry keys, server file information, and type library information to the registry. It also adds the verbs added with the AddVerb (see page 1557) method.

If the Register parameter is True, true, the control should add its keys to the system registry. If it is False, false, the control should remove its keys from the registry.

Override this method if you want your control to store more information in the registry than the base implementation stores. If you do override this method, be sure to call the inherited implementation in the correct order. When adding registry entries (that is, Register is True), true), call the inherited method before adding custom entries. When removing registry entries (that is, Register is False), false), call the inherited method after removing the custom entries.

See Also

AddVerb (see page 1557)

12.1.5.5 TActiveXControlFactory Properties

The properties of the TActiveXControlFactory class are listed here.

Public Properties

	Name	Description
R	MiscStatus (see page 1559)	Represents a set of status bits that are stored in the registry.
R	ToolboxBitmapID (see page 1560)	Designates the resource identifier of the ActiveX control.
R	WinControlClass (see page 1560)	WinControlClass is the class of the VCL control that implements the Windows behavior for the ActiveX control.

12.1.5.5.1 TActiveXControlFactory.MiscStatus Property

Represents a set of status bits that are stored in the registry.

Pascal

```
property MiscStatus: Integer;
```

C++

```
__property int MiscStatus;
```

Description

MiscStatus is stored in a registry entry under the CLSID entry for the class. This entry provides information about the ActiveX server to a container. The entry contains a set of binary flags for which ActiveX defines specific meanings. This entry is not

required.

The following table lists the status bits that ActiveX control containers must recognize and support. For each status bit, it indicates whether container support is required for the feature and includes a brief description.

Status bit	Container support	Description
ACTIVATEWHENVISIBLE	yes	Indicates that the object should be activated when it becomes visible, even if it isn't UI-active.
IGNOREACTIVATEWHENVISIBLE	no	Needed for inactive and windowless control support.
INSIDEOUT	no	Indicates that the object should be activated inside-out (it can be in-place active as well as UI-active.) Usually used with embedded compound documents rather than with controls.
INVISIBLEATRUNTIME	yes	Designates a control that should be visible at design-time, but not at run-time.
ALWAYSRUN	yes	Indicates that the object should always be in the running state and that the default handler should not delay the loading of the object until the last possible moment.
ACTSLIKEBUTTON	no	Indicates that the object has button-like behavior. This allows the container to ask the control to draw itself as the default button or as a normal button.
ACTSLIKELABEL	no	Indicates that the object behaves like a label. This allows the container to handle the control properly when its label gets focus.
NOUIACTIVATE	yes	Indicates that the control has no user interface to activate.
ALIGNABLE	no	Indicates that the control can be aligned by the user.
SIMPLEFRAME	no	Indicates that the control is a simple holder for other controls.
SETCLIENTSITEFIRST	no	Indicates that the control needs to access the client site very early in its creation.
IMEMODE	no	Indicates that the control supports an Input Method Editor (IME) for entering extended characters. Typically, a container that supports IME provides the control with an extended IMEMode property.

12.1.5.5.2 TActiveXControlFactory.ToolboxBitmapID Property

Designates the resource identifier of the ActiveX control.

Pascal

```
property ToolboxBitmapID: Integer;
```

C++

```
__property int ToolboxBitmapID;
```

Description

Use ToolboxBitmapID to obtain the resource ID of the bitmap that is used to represent the control on a design palette.

See Also

Create (see page 1556)

12.1.5.5.3 TActiveXControlFactory.WinControlClass Property

WinControlClass is the class of the VCL control that implements the Windows behavior for the ActiveX control.

Pascal

```
property WinControlClass: TWinControlClass;
```

C++

```
__property TWinControlClass WinControlClass;
```

Description

The factory object keeps a reference to the VCL control's class information so that it can ask the class object to manufacture a VCL control when the ActiveX control is created.

See Also

TWinControl (see page 6112)

Control (see page 1553)

12.1.6 TActiveXPropertyPage Class

TActiveXPropertyPage represents a COM object that provides the interfaces for supporting an ActiveX property page.

Class Hierarchy

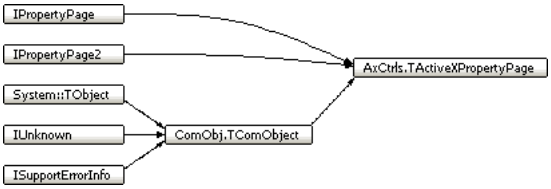

Pascal

```
TActiveXPropertyPage = class(TComObject, IPropertyPage, IPropertyPage2);
```

C++

```
class TActiveXPropertyPage : public TComObject, public IPropertyPage, public IPropertyPage2;
```

File

AxCtrls

Description

This class is used internally by the TPropertyPage (see page 1603) control to provide a bridge between a Delphi C++Builder form and the ActiveX property page API.

See Also

TPropertyPage (see page 1603)

12.1.6.1 TActiveXPropertyPage Members

The following tables list the members exposed by TActiveXPropertyPage.

Protected Methods

	Name	Description
	_AddRef (see page 5558)	This is _AddRef, a member of class TComObject.

	_Release (see page 5558)	This is _Release, a member of class TComObject.
	InterfaceSupportsErrorInfo (see page 5559)	This is InterfaceSupportsErrorInfo, a member of class TComObject.
	QueryInterface (see page 5560)	This is QueryInterface, a member of class TComObject.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5555)	Instantiates a COM object that is not part of an aggregate.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateAggregated (see page 5556)	Instantiates a COM object that is part of an aggregate.
	CreateFromFactory (see page 5556)	Instantiates a COM object and initializes its values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 1563)	Frees the instance of the Destroy.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 1563)	Initializes the instance of a Initialize descendant.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	ObjAddRef (see page 5559)	Implements the AddRef method of the IUnknown interface.
V	ObjQueryInterface (see page 5559)	Implements the QueryInterface (see page 5560) method of the IUnknown interface.
V	ObjRelease (see page 5559)	Implements the Release method of the IUnknown interface.
V	SafeCallException (see page 5560)	Handles OLE exceptions.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Controller (see page 5561)	Specifies the controlling IUnknown interface that controls interfaces within an aggregate object.
R	Factory (see page 5561)	Specifies the class factory object for the TComObject (see page 5553) class.
R	PropertyPageImpl (see page 1564)	The implementation of the COM server portion of the COM object.
R	RefCount (see page 5562)	Indicates the number of references to the COM object that are currently in use.
	ServerExceptionHandler (see page 5562)	Provides access to the SafeCall exception notification process.

12.1.6.2 TActiveXPropertyPage.Destroy Destructor

Frees the instance of the Destroy.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TActiveXPropertyPage();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TPropertyPage (see page 1603) reference is not nil before calling Destroy. ~Destroy::.

See Also

Free (see page 22300)

12.1.6.3 TActiveXPropertyPage Methods

The methods of the TActiveXPropertyPage class are listed here.

Public Methods

	Name	Description
	Initialize (see page 1563)	Initializes the instance of a Initialize descendant.

12.1.6.3.1 TActiveXPropertyPage.Initialize Method

Initializes the instance of a Initialize descendant.

Pascal

```
procedure Initialize; override;
```

C++

```
virtual __fastcall Initialize();
```

Description

Override the Initialize method to initialize an instance of a TPropertyPage (see page 1603) descendant before it is used.

See Also

Create (see page 5555)

12.1.6.4 TActiveXPropertyPage Properties

The properties of the TActiveXPropertyPage class are listed here.

Public Properties

	Name	Description
	PropertyPageImpl (see page 1564)	The implementation of the COM server portion of the COM object.

12.1.6.4.1 TActiveXPropertyPage.PropertyPageImpl Property

The implementation of the COM server portion of the COM object.

Pascal

```
property PropertyPageImpl: TPropertyPageImpl;
```

C++

```
__property TPropertyPageImpl PropertyPageImpl;
```

Description

PropertyPageImpl is the COM server portion of the implementation of the IPropertyPage interface. Applications do not create the TPropertyPageImpl (see page 1679) objects; they are created automatically when the user creates an ActiveX control object by using the ActiveX control or ActiveForms wizard.

TActiveXPropertyPage (see page 1561) implements the COM server portion of the COM object while TPropertyPageImpl (see page 1679) implements the property page portion.

12.1.7 TActiveXPropertyPageFactory Class

TActiveXPropertyPageFactory implements a COM factory for an ActiveX property page.

Class Hierarchy

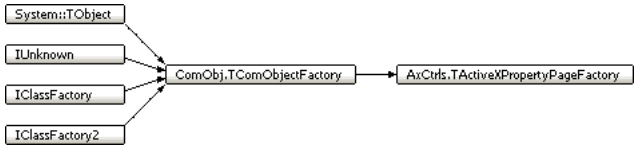

Pascal

```
TActiveXPropertyPageFactory = class(TComObjectFactory);
```

C++

```
class TActiveXPropertyPageFactory : public TComObjectFactory;
```

File

AxCtrls

Description

A property page is a COM object, which means the module it is implemented in must provide a factory so that the system can create the object. TActiveXPropertyPageFactory is a factory for ActiveX property pages, which are implemented by subclassing TPropertyPage (see page 1603).

For more information about COM factories, see TComObjectFactory.

To add a property page factory to a program, create the factory object in the initialization section of the unit that defines the property page form.

See Also

TPropertyPage (see page 1603)

TComServerObject (see page 5575)

12.1.7.1 TActiveXPropertyPageFactory Members

The following tables list the members exposed by TActiveXPropertyPageFactory.

Protected Methods

	Name	Description
	_AddRef (see page 5567)	This is _AddRef, a member of class TComObjectFactory.
	_Release (see page 5567)	This is _Release, a member of class TComObjectFactory.
	CreateInstance (see page 5568)	This is CreateInstance, a member of class TComObjectFactory.
	CreateInstanceLic (see page 5568)	This is CreateInstanceLic, a member of class TComObjectFactory.
V	GetLicenseString (see page 5568)	This is GetLicenseString, a member of class TComObjectFactory.
	GetLicInfo (see page 5568)	This is GetLicInfo, a member of class TComObjectFactory.
V	GetProgID (see page 5569)	This is GetProgID, a member of class TComObjectFactory.
V	HasMachineLicense (see page 5569)	This is HasMachineLicense, a member of class TComObjectFactory.
	LockServer (see page 5569)	This is LockServer, a member of class TComObjectFactory.
	QueryInterface (see page 5569)	This is QueryInterface, a member of class TComObjectFactory.
	RequestLicKey (see page 5570)	This is RequestLicKey, a member of class TComObjectFactory.
V	ValidateUserLicense (see page 5571)	This is ValidateUserLicense, a member of class TComObjectFactory.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1566)	Creates a property page factory and binds that factory to property page class.
V	CreateComObject (see page 1567)	Instantiates a TComObject object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5566)	Disposes of an instance of a class factory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RegisterClassObject (see page 5569)	Registers the COM class associated with the factory object.

V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	UpdateRegistry (see page 5570)	Registers COM objects created by the factory.

Public Properties

	Name	Description
R	ClassID (see page 5571)	This is ClassID, a member of class TComObjectFactory.
R	ClassName (see page 5572)	Specifies the name of the class that the class factory instantiates.
R	ComClass (see page 5572)	This is ComClass, a member of class TComObjectFactory.
R	ComServer (see page 5572)	This is ComServer, a member of class TComObjectFactory.
R	Description (see page 5572)	Describes the class object.
	ErrorIID (see page 5573)	Indicates the interface of the CoClass in which an error occurred.
R	Instancing (see page 5573)	Indicates the process space of COM object instances.
	LicString (see page 5573)	Contains licensing information about the object.
R	ProgID (see page 5574)	Specifies the name of the class factory.
	ShowErrors (see page 5574)	Indicates whether the COM object factory displays a message box when exceptions are raised.
	SupportsLicensing (see page 5574)	Indicates whether the object being instantiated supports licensing.
R	ThreadingModel (see page 5575)	Indicates the type of thread support provided by objects that the factory creates.

12.1.7.2 TActiveXPropertyPageFactory.Create Constructor

Creates a property page factory and binds that factory to property page class.

Pascal

```
constructor Create(ComServer: TComServerObject; PropertyPageClass: TPropertyPageClass; const
ClassID: TGUID);
```

C++

```
__fastcall TActiveXPropertyPageFactory(TComServerObject ComServer, TPropertyPageClass
PropertyPageClass, const TGUID ClassID);
```

Description

The ComServer parameter is a pointer to the COM server object for the ActiveX library. This is normally the global variable, ComServer, declared in the ComServ (see page 5621) unit.

The PropertyPageClass parameter is the class object of the property page's implementing class. The implementing class must be derived form TPropertyPage (see page 1603). This class implements the form that is to appear in the property dialog.

The ClassID parameter is a GUID that identifies the property page's class. Each property page should have its own GUID. See the Developer's Programmer's Guide for information on how to create a GUID.

See Also

Create (see page 5565)

12.1.7.3 TActiveXPropertyPageFactory Methods

The methods of the TActiveXPropertyPageFactory class are listed here.

Public Methods

	Name	Description
V	CreateComObject (see page 1567)	Instantiates a TComObject object.

12.1.7.3.1 TActiveXPropertyPageFactory.CreateComObject Method

Instantiates a TComObject object.

Pascal

```
function CreateComObject(const Controller: IUnknown): TComObject; override;
```

C++

```
virtual __fastcall TComObject CreateComObject(const IUnknown Controller);
```

Description

CreateComObject is the method that instantiates the class type for which TComObjectFactory is the class factory. CreateComObject calls the CreateFromFactory constructor of the TComObject class to create a COM object. CreateComObject passes itself as the Factory parameter to CreateFromFactory.

CreateComClass creates a COM object based upon the value of the ComClass property. ComClass is the class identifier (CLSID) used by other applications to identify the COM class for which they request an object instance.

See Also

comclass (see page 5572)

CreateFromFactory (see page 5556)

TComObject (see page 5553)

12.1.8 TAdapterNotifier Class

TAdapterNotifier provides the binding between a VCL type object and comparable OLE type object.

Class Hierarchy

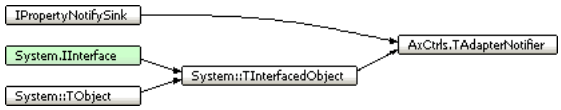

Pascal

```
TAdapterNotifier = class(TInterfacedObject, IPropertyNotifySink);
```

C++

```
class TAdapterNotifier : public TInterfacedObject, public IPropertyNotifySink;
```

File

AxCtrls

Description

Descendants of TCustomAdapter (see page 1578) use TAdapterNotifier to bind VCL utility objects (such as TFont) with corresponding OLE objects. For example, TFontAdapter (see page 1583) uses TAdapterNotifier to bind a TFont object with an OLE font object.

See Also

TCustomAdapter

TFontAdapter (see page 1583)

TPictureAdapter (see page 1599)

12.1.8.1 TAdapterNotifier Members

The following tables list the members exposed by TAdapterNotifier.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	OnChanged (see page 1569)	This is OnChanged, a member of class TAdapterNotifier.
	OnRequestEdit (see page 1570)	This is OnRequestEdit, a member of class TAdapterNotifier.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1569)	Creates an instance of a TAdapterNotifier (see page 1567) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

12.1.8.2 TAdapterNotifier.Create Constructor

Creates an instance of a TAdapterNotifier (see page 1567) object.

Pascal

```
constructor Create(Adapter: TCustomAdapter);
```

C++

```
__fastcall TAdapterNotifier(TCustomAdapter Adapter);
```

Description

You would normally never create an object of this class. TAdapterNotifier (see page 1567) objects are created by and bound to a descendant of TCustomAdapter (see page 1578).

See Also

Create (see page 22287)

12.1.8.3 TAdapterNotifier Methods

The methods of the TAdapterNotifier class are listed here.

Protected Methods

	Name	Description
	OnChanged (see page 1569)	This is OnChanged, a member of class TAdapterNotifier.
	OnRequestEdit (see page 1570)	This is OnRequestEdit, a member of class TAdapterNotifier.

12.1.8.3.1 TAdapterNotifier.OnChanged Method

Pascal

```
function OnChanged(dispid: TDispID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnChanged(TDispID dispid);
```

Description

This is OnChanged, a member of class TAdapterNotifier.

12.1.8.3.2 TAdapterNotifier.OnRequestEdit Method

Pascal

```
function OnRequestEdit(dispid: TDispID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnRequestEdit(TDispID dispid);
```

Description

This is OnRequestEdit, a member of class TAdapterNotifier.

12.1.9 TConnectionPoint Class

TConnectionPoint implements the IConnectionPoint interface for a single outgoing interface.

Class Hierarchy

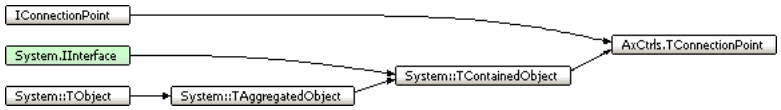

Pascal

```
TConnectionPoint = class(TContainedObject, IConnectionPoint);
```

C++

```
class TConnectionPoint : public TContainedObject, public IConnectionPoint;
```

File

AxCtrls

Description

TConnectionPoint implements the IConnectionPoint interface, which client applications call to register an interest in a specific outgoing interface. The corresponding outgoing interface defines event handlers that the client implements. The client uses IConnectionPoint to supply its implementation of this interface (called an event sink), so that it can be called when events occur.

Wizards that generate event support code automatically create a class member of type TConnectionPoint, which maintains a list of event sinks supplied by clients.

TConnectionPoint objects support the following features:

Outgoing interfaces, such as event sets.

The ability to enumerate the IIDs of the outgoing interfaces.

The ability to connect and disconnect sinks to the object for those outgoing IIDs.

The ability to enumerate the connections that exist to a particular outgoing interface.

The IConnectionPoint and IConnectionPointContainer interfaces are used for OLE event handling. Because the wizard automatically sets up events for a control, users only need to use these interfaces directly when they wish to modify the standard VCL implementation or to add events to a COM server whose wizard does not generate event support code.

GetConnectionInterface (see page 1573), GetConnectionPointContainer (see page 1574), Advise (see page 1573), Unadvise (see page 1574), EnumConnections (see page 1573)

See Also

TConnectionPoints (see page 1575)

12.1.9.1 TConnectionPoint Members

The following tables list the members exposed by TConnectionPoint.

Protected Methods

	Name	Description
	_AddRef (see page 22250)	This is _AddRef, a member of class TAggregatedObject.
	_Release (see page 22250)	This is _Release, a member of class TAggregatedObject.
	Advise (see page 1573)	This is Advise, a member of class TConnectionPoint.
	EnumConnections (see page 1573)	This is EnumConnections, a member of class TConnectionPoint.
	GetConnectionInterface (see page 1573)	This is GetConnectionInterface, a member of class TConnectionPoint.
	GetConnectionPointContainer (see page 1574)	This is GetConnectionPointContainer, a member of class TConnectionPoint.
	QueryInterface (see page 22250)	This is QueryInterface, a member of class TAggregatedObject.
V	QueryInterface (see page 22255)	Returns a reference to a specified interface if the object supports that interface.
	Unadvise (see page 1574)	This is Unadvise, a member of class TConnectionPoint.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1572)	Instantiates a connection point object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 1572)	Destroys the instance of a connection point object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Controller (see page 22251)	Provides access to the controlling interface of the aggregate.
R	SinkList (see page 1574)	Lists the sinks for the outgoing interfaces maintained by the connection point.

12.1.9.2 TConnectionPoint.Create Constructor

Instantiates a connection point object.

Pascal

```
constructor Create(Container: TConnectionPoints; const IID: TGUID; Kind: TConnectionKind;
OnConnect: TConnectEvent);
```

C++

```
__fastcall TConnectionPoint(TConnectionPoints Container, const TGUID IID, TConnectionKind
Kind, TConnectEvent OnConnect);
```

Description

Create is called automatically by the connection point container's CreateConnectionPoint method, to instantiate a connection point object. This occurs when an ActiveX control is initialized.

Create allocates memory for a connection point object and calls the inherited Create, passing it the Container's controlling IUnknown. Container is a connection point container object. Create adds the connection point object to the Container. Then it sets the

IID as the GUID or interface identifier of the outgoing interface managed by this connection point.

IKind, indicating whether it is a single or multiple connection to that point.

OnConnect as the connection event used in the IConnectionPoint Advise (see page 1573) method.

TConnectEvent points to a procedure for the OnConnect event of the connection point. TConnectionKind (see page 1714) is a type indicating whether there are multiple connections to the connection point.

See Also

Create

Initialize (see page 1536)

Destroy (see page 1572)

CreateConnectionPoint (see page 1577)

12.1.9.3 TConnectionPoint.Destroy Destructor

Destroys the instance of a connection point object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TConnectionPoint();
```

Description

It is not necessary to explicitly free the connection point object. Connection point objects are automatically destroyed. If the connection point container object is not nil, Destroy calls on it to remove the connection point object from it.

See Also

TConnectionPoints (see page 1575)

Free (see page 22300)

12.1.9.4 TConnectionPoint Methods

The methods of the TConnectionPoint class are listed here.

Protected Methods

	Name	Description
	Advise (see page 1573)	This is Advise, a member of class TConnectionPoint.
	EnumConnections (see page 1573)	This is EnumConnections, a member of class TConnectionPoint.
	GetConnectionInterface (see page 1573)	This is GetConnectionInterface, a member of class TConnectionPoint.
	GetConnectionPointContainer (see page 1574)	This is GetConnectionPointContainer, a member of class TConnectionPoint.
	Unadvise (see page 1574)	This is Unadvise, a member of class TConnectionPoint.

12.1.9.4.1 TConnectionPoint.Advise Method

Pascal

```
function Advise(const unkSink: IUnknown; out dwCookie: Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Advise(const IUnknown unkSink, Longint dwCookie);
```

Description

This is Advise, a member of class TConnectionPoint.

12.1.9.4.2 TConnectionPoint.EnumConnections Method

Pascal

```
function EnumConnections(out enumconn: IEnumConnections): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT EnumConnections(IEnumConnections enumconn);
```

Description

This is EnumConnections, a member of class TConnectionPoint.

12.1.9.4.3 TConnectionPoint.GetConnectionInterface Method

Pascal

```
function GetConnectionInterface(out iid: TIID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetConnectionInterface(TIID iid);
```

Description

This is GetConnectionInterface, a member of class TConnectionPoint.

12.1.9.4.4 TConnectionPoint.GetConnectionPointContainer Method

Pascal

```
function GetConnectionPointContainer(out cpc: IConnectionPointContainer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetConnectionPointContainer(IConnectionPointContainer cpc);
```

Description

This is GetConnectionPointContainer, a member of class TConnectionPoint.

12.1.9.4.5 TConnectionPoint.Unadvise Method

Pascal

```
function Unadvise(dwCookie: Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Unadvise(Longint dwCookie);
```

Description

This is Unadvise, a member of class TConnectionPoint.

12.1.9.5 TConnectionPoint Properties

The properties of the TConnectionPoint class are listed here.

Public Properties

	Name	Description
	SinkList (see page 1574)	Lists the sinks for the outgoing interfaces maintained by the connection point.

12.1.9.5.1 TConnectionPoint.SinkList Property

Lists the sinks for the outgoing interfaces maintained by the connection point.

Pascal

```
property SinkList: TList;
```

C++

```
__property TList SinkList;
```

Description

SinkList lists the sinks managed by the connection point. These are the interfaces that clients implement for responding to server events. When a client registers an interest in the connection point, it supplies an interface pointer that is added to this list.

Each item in the list can be cast to an IUnknown and used to access the outgoing interface.

12.1.10 TConnectionPoints Class

TConnectionPoints implements the IConnectionPointContainer interface for a COM object that generates events on clients.

Class Hierarchy

AxCtrls.TConnectionPoints

Pascal

```
TConnectionPoints = class;
```

C++

```
class TConnectionPoints;
```

File

AxCtrls

Description

TConnectionPoints lets an object indicate the existence of outgoing interfaces (event interfaces). TConnectionPoints implements the IConnectionPointContainer interface, which clients use to register an interest in event notifications. COM wizards add a TConnectionPoints object as a class member to automatically generated classes that fire events on clients.

The IConnectionPoint interface provides access to an enumerator sub-object with the IEnumConnectionPoints interface. It also provides access to all the connection point sub-objects, each of which implements the IConnectionPoint interface. The IConnectionPoint interface provides access to an enumerator sub-object with the IEnumConnections interface.

TConnectionPoints objects are used to obtain access to:

An enumerator sub-object with the IEnumConnectionPoints interface. The IEnumConnectionPoints interface can be used to enumerate connection points for each outgoing IID.

Connection point sub-objects with the IConnectionPoint interface for each outgoing IID. Through the IConnectionPoint interface, a client starts or terminates an advisory loop with the connectable object and the client's own sink. The client can also use the IConnectionPoint interface to obtain an enumerator object with the IEnumConnections interface to enumerate the connections that it knows about.

Because the ActiveX wizard automatically sets up events for a control, users only need to use these interfaces directly when they wish to modify the standard VCL implementation or to add events to a non-ActiveX control automation server.

IConnectionPointContainer

See Also

TConnectionPoint (see page 1570)

12.1.10.1 TConnectionPoints Members

The following tables list the members exposed by TConnectionPoints.

Protected Methods

	Name	Description
	EnumConnectionPoints (see page 1577)	This is EnumConnectionPoints, a member of class TConnectionPoints.
	FindConnectionPoint (see page 1578)	This is FindConnectionPoint, a member of class TConnectionPoints.

Public Methods

	Name	Description
	Create (see page 1576)	Instantiates a connection points object.
	CreateConnectionPoint (see page 1577)	Instantiates a connection point object.
V	Destroy (see page 1576)	Destroys an instance of a connection points object.

Public Properties

	Name	Description
R	Controller (see page 1578)	Provides access to the IUnknown of the connection points.

12.1.10.2 TConnectionPoints.Create Constructor

Instantiates a connection points object.

Pascal

```
constructor Create(const AController: IUnknown);
```

C++

```
__fastcall TConnectionPoints(const IUnknown AController);
```

Description

It is not necessary to call Create directly. It is used internally by other objects to create the connection points object.

Create allocates memory for a connection points object and initializes the Controller (see page 1578) property using the AController parameter. Then it creates a list for the individual connection point objects that it will contain.

See Also

Controller (see page 1578)

12.1.10.3 TConnectionPoints.Destroy Destructor

Destroys an instance of a connection points object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TConnectionPoints();
```

Description

Destroy is called automatically. It is not necessary for the user to dispose of the connection points object. Destroy frees the list of connection point objects that was created in the constructor.

See Also

Free (see page 22300)

12.1.10.4 TConnectionPoints Methods

The methods of the TConnectionPoints class are listed here.

Protected Methods

	Name	Description
	EnumConnectionPoints (see page 1577)	This is EnumConnectionPoints, a member of class TConnectionPoints.
	FindConnectionPoint (see page 1578)	This is FindConnectionPoint, a member of class TConnectionPoints.

Public Methods

	Name	Description
	CreateConnectionPoint (see page 1577)	Instantiates a connection point object.

12.1.10.4.1 TConnectionPoints.CreateConnectionPoint Method

Instantiates a connection point object.

Pascal

```
function CreateConnectionPoint(const IID: TGUID; Kind: TConnectionKind; OnConnect:
TConnectEvent): TConnectionPoint;
```

C++

```
__fastcall TConnectionPoint CreateConnectionPoint(const TGUID IID, TConnectionKind Kind,
TConnectEvent OnConnect);
```

Description

CreateConnectionPoint is automatically called when an ActiveX control is initialized. CreateConnectionPoint creates and returns a connection point object. It passes to the connection point constructor:

Itself as the connection point container.

IID as the GUID or interface identifier of the outgoing interface managed by this connection point.

OnConnect as the connection event used in the IConnectionPoint Advise method.

TConnectEvent is a pointer to a procedure for the OnConnect event of the connection point object being created. TConnectionKind (see page 1714) is a type indicating whether there are multiple connections to that connection point.

See Also

Advise

Initialize (see page 1536)

Create (see page 1572)

12.1.10.4.2 TConnectionPoints.EnumConnectionPoints Method

Pascal

```
function EnumConnectionPoints(out enumconn: IEnumConnectionPoints): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT EnumConnectionPoints(IEnumConnectionPoints enumconn);
```

Description

This is EnumConnectionPoints, a member of class TConnectionPoints.

12.1.10.4.3 TConnectionPoints.FindConnectionPoint Method

Pascal

```
function FindConnectionPoint(const iid: TIID; out cp: IConnectionPoint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT FindConnectionPoint(const TIID iid, IConnectionPoint cp);
```

Description

This is FindConnectionPoint, a member of class TConnectionPoints.

12.1.10.5 TConnectionPoints Properties

The properties of the TConnectionPoints class are listed here.

Public Properties

	Name	Description
	Controller (see page 1578)	Provides access to the IUnknown of the connection points.

12.1.10.5.1 TConnectionPoints.Controller Property

Provides access to the IUnknown of the connection points.

Pascal

```
property Controller: IUnknown;
```

C++

```
__property IUnknown Controller;
```

Description

Controller is the IUnknown interface reference of the connection points object. TConnectionPoints (see page 1575) aggregates several sub-objects that share this interface as a single controlling IUnknown. The IUnknown methods implemented in the sub-objects delegate to this Controller.

When the TConnectionPoints (see page 1575) object is created, its constructor is passed a controlling IUnknown. Controller is initialized to this IUnknown in the constructor.

See Also

Create (see page 1576)

12.1.11 TCustomAdapter Class

TCustomAdapter is a base class for objects that bind VCL objects to COM interfaces.

Class Hierarchy

Pascal

```
TCustomAdapter = class(TInterfacedObject);
```

C++

```
class TCustomAdapter : public TInterfacedObject;
```

File

AxCtrls

Description

Do not create objects of this class. Instead, call the GetOleFont (see page 1710) procedure to create the binding between a TFont and an OLE font object or GetOlePicture (see page 1710) to create the binding between a TPicture and an OLE picture object.

See Also

TFontAdapter (see page 1583)

TPictureAdapter (see page 1599)

GetOleFont

GetOlePicture

12.1.11.1 TCustomAdapter Members

The following tables list the members exposed by TCustomAdapter.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Fields

	Name	Description
	Updating (see page 1581)	This is Updating, a member of class TCustomAdapter.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
V	Changed (see page 1582)	Responds when the VCL representation of the OLE object changes.
	ConnectOleObject (see page 1582)	Associates the adapter with an OLE interface.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	ReleaseOleObject (see page 1582)	Releases the associated OLE interface, allowing it to decrease its reference count.
A	Update (see page 1583)	Provides the interface for a method that responds to changes in the OLE object.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1580)	Constructs an instance of the custom adapter control.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 1581)	Destroys the custom adapter object.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

12.1.11.2 TCustomAdapter.Create Constructor

Constructs an instance of the custom adapter control.

Pascal

```
constructor Create;
```

C++

```
__fastcall TCustomAdapter();
```

Description

Never create an object of this class, only its descendants, such as TFontAdapter (see page 1583) or TPictureAdapter (see page 1599).

See Also

Destroy

12.1.11.3 TCustomAdapter.Destroy Destructor

Destroys the custom adapter object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapter();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks that the adapter is not nil before calling Destroy.

Destroy frees and disconnects the VCL object from an OLE object.

See Also

Create

Free (see page 22300)

12.1.11.4 TCustomAdapter Fields

The fields of the TCustomAdapter class are listed here.

Protected Fields

	Name	Description
	Updating (see page 1581)	This is Updating, a member of class TCustomAdapter.

12.1.11.4.1 TCustomAdapter.Updating Field

Pascal

```
Updating: Boolean;
```

C++

```
Boolean Updating;
```

Description

This is Updating, a member of class TCustomAdapter.

12.1.11.5 TCustomAdapter Methods

The methods of the TCustomAdapter class are listed here.

Protected Methods

	Name	Description
V	Changed (see page 1582)	Responds when the VCL representation of the OLE object changes.
	ConnectOleObject (see page 1582)	Associates the adapter with an OLE interface.
	ReleaseOleObject (see page 1582)	Releases the associated OLE interface, allowing it to decrease its reference count.
A	Update (see page 1583)	Provides the interface for a method that responds to changes in the OLE object.

12.1.11.5.1 TCustomAdapter.Changed Method

Responds when the VCL representation of the OLE object changes.

Pascal

```
procedure Changed; virtual;
```

C++

```
virtual __fastcall Changed();
```

Description

Changed is called automatically so that the associated OLE object can be updated to reflect changes made to the corresponding VCL object. In TCustomAdapter (see page 1578), Changed calls ReleaseOleObject (see page 1582) so that the OLE object can be re-acquired to reflect the changes in the VCL object. Descendants of TCustomAdapter (see page 1578) override Changed to use the OLE interface to make appropriate changes.

See Also

ReleaseOleObject (see page 1582)

12.1.11.5.2 TCustomAdapter.ConnectOleObject Method

Associates the adapter with an OLE interface.

Pascal

```
procedure ConnectOleObject(OleObject: IUnknown);
```

C++

```
__fastcall ConnectOleObject(IUnknown OleObject);
```

Description

ConnectOleObject is used internally to associate the adapter with the OLE object it represents.

See Also

ReleaseOleObject (see page 1582)

12.1.11.5.3 TCustomAdapter.ReleaseOleObject Method

Releases the associated OLE interface, allowing it to decrease its reference count.

Pascal

```
procedure ReleaseOleObject;
```

C++

```
__fastcall ReleaseOleObject();
```

Description

When implementing a descendant of TCustomAdapter (see page 1578), call ReleaseOleObject when the adapter no longer needs the OLE object.

See Also

ConnectOleObject (see page 1582)

12.1.11.5.4 TCustomAdapter.Update Method

Provides the interface for a method that responds to changes in the OLE object.

Pascal

```
procedure Update; virtual; abstract;
```

C++

```
virtual __fastcall Update() = 0;
```

Description

Descendants of TCustomAdapter (see page 1578) override Update to modify the associated VCL object so that it reflects changes in the corresponding OLE object. It is the converse of the Changed (see page 1582) method.

See Also

Changed (see page 1582)

12.1.12 TFontAdapter Class

TFontAdapter is a helper class used to connect a TFont object to an OLE font object.

Class Hierarchy

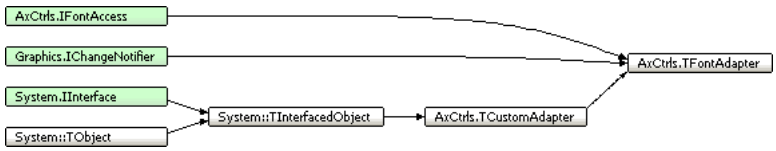

Pascal

```
TFontAdapter = class(TCustomAdapter, IChangeNotifier, IFontAccess);
```

C++

```
class TFontAdapter : public TCustomAdapter, public IChangeNotifier, public IFontAccess;
```

File

AxCtrls

Description

To create an instance of a TFontAdapter, call the global GetOleFont (see page 1586) procedure. GetOleFont (see page 1586) creates the binding between a TFont object and an OLE font interface.

See Also

TPictureAdapter (see page 1599)

TFont (see page 14497)

GetOleFont

SetOleFont

12.1.12.1 TFontAdapter Members

The following tables list the members exposed by TFontAdapter.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Fields

	Name	Description
	Updating (see page 1581)	This is Updating, a member of class TCustomAdapter.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
V	Changed (see page 1585)	Updates the associated IFont interface with the new properties of the font object.
	ConnectOleObject (see page 1582)	Associates the adapter with an OLE interface.
	GetOleFont (see page 1586)	Obtains the IFontDisp interface from the associated OLE object.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	ReleaseOleObject (see page 1582)	Releases the associated OLE interface, allowing it to decrease its reference count.
	SetOleFont (see page 1586)	Associates the adapter with the OLE object represented by a specified interface.
V	Update (see page 1587)	Updates the associated TFont object to reflect changes in the associated OLE font.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1585)	Creates an instance of the font adapter control.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 1581)	Destroys the custom adapter object.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

12.1.12.2 TFontAdapter.Create Constructor

Creates an instance of the font adapter control.

Pascal

```
constructor Create(Font: TFont);
```

C++

```
__fastcall TFontAdapter(TFont Font);
```

Description

Applications do not normally create objects of this class. Instead, they call the GetOleFont (see page 1586) procedure, which creates the binding between a TFont and an OLE font object.

See Also

GetOleFont

12.1.12.3 TFontAdapter Methods

The methods of the TFontAdapter class are listed here.

Protected Methods

	Name	Description
V	Changed (see page 1585)	Updates the associated IFont interface with the new properties of the font object.
	GetOleFont (see page 1586)	Obtains the IFontDisp interface from the associated OLE object.
	SetOleFont (see page 1586)	Associates the adapter with the OLE object represented by a specified interface.
V	Update (see page 1587)	Updates the associated TFont object to reflect changes in the associated OLE font.

12.1.12.3.1 TFontAdapter.Changed Method

Updates the associated IFont interface with the new properties of the font object.

Pascal

```
procedure Changed; override;
```

C++

```
virtual __fastcall Changed();
```

Description

Changed is called automatically when the TFont object changes. It ensures that the associated OLE interface is updated.

See Also

SetOleFont (see page 1586)

Update (see page 1587)

SetOleFont

12.1.12.3.2 TFontAdapter.GetOleFont Method

Obtains the IFontDisp interface from the associated OLE object.

Pascal

```
procedure GetOleFont(var OleFont: IFontDisp);
```

C++

```
__fastcall GetOleFont(IFontDisp OleFont);
```

Description

ActiveX controls that use TFontAdapter (see page 1583) call GetOleFont to obtain the interface that corresponds to the adapter's TFont object. If the OLE object for the font does not exist, GetOleFont creates it.

The IFontDisp interface is returned as the OleFont parameter.

See Also

SetOleFont (see page 1586)

ConnectOleObject (see page 1582)

12.1.12.3.3 TFontAdapter.SetOleFont Method

Associates the adapter with the OLE object represented by a specified interface.

Pascal

```
procedure SetOleFont(const OleFont: IFontDisp);
```

C++

```
__fastcall SetOleFont(const IFontDisp OleFont);
```

Description

ActiveX controls that use TFontAdapter (see page 1583) call SetOleFont to associate the adapter with the interface passed as the OleFont parameter.

See Also

GetOleFont (see page 1586)

12.1.12.3.4 TFontAdapter.Update Method

Updates the associated TFont object to reflect changes in the associated OLE font.

Pascal

```
procedure Update; override;
```

C++

```
virtual __fastcall Update();
```

Description

Update is called automatically when the associated OLE object changes. It ensures that the corresponding TFont object reflects those changes.

See Also

SetOleFont

12.1.13 TOleGraphic Class

TOleGraphic is a wrapper for an OLE graphical interface.

Class Hierarchy

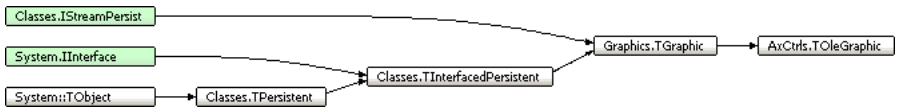

Pascal

```
TOleGraphic = class(TGraphic);
```

C++

```
class TOleGraphic : public TGraphic;
```

File

AxCtrls

Description

TOleGraphic is a graphical container for icons, metafiles and bitmaps used in ActiveX controls. TOleGraphic is used internally to expose an IPicture interface as the image of the TPicture property of an ActiveX control. Because TOleGraphic is a wrapper for OLE interfaces, it cannot be used to load or save information to a file.

See Also

TActiveXControl (see page 1517)

TGraphic (see page 14516)

12.1.13.1 TOleGraphic Members

The following tables list the members exposed by TOleGraphic.

Protected Methods

	Name	Description
	_AddRef (see page 2853)	Increments the reference count for the interfaced object's interface.
	_Release (see page 2854)	Decrements the reference count for the interfaced object's interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Changed (see page 1590)	This is Changed, a member of class TOleGraphic.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 14520)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Draw (see page 1590)	This is Draw, a member of class TOleGraphic.
V	Equals (see page 14521)	This is Equals, a member of class TGraphic.
V	GetEmpty (see page 1591)	This is GetEmpty, a member of class TOleGraphic.
V	GetHeight (see page 1591)	This is GetHeight, a member of class TOleGraphic.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetPalette (see page 1591)	This is GetPalette, a member of class TOleGraphic.
V	GetTransparent (see page 1591)	This is GetTransparent, a member of class TOleGraphic.
V	GetWidth (see page 1591)	This is GetWidth, a member of class TOleGraphic.
	Progress (see page 14524)	This is Progress, a member of class TGraphic.
V	ReadData (see page 14524)	This is ReadData, a member of class TGraphic.
V	SetHeight (see page 1593)	This is SetHeight, a member of class TOleGraphic.
V	SetPalette (see page 1594)	This is SetPalette, a member of class TOleGraphic.
V	SetTransparent (see page 14527)	This is SetTransparent, a member of class TGraphic.
V	SetWidth (see page 1594)	This is SetWidth, a member of class TOleGraphic.
V	WriteData (see page 14527)	This is WriteData, a member of class TGraphic.

Public Events

	Name	Description
	OnChange (see page 14531)	Occurs whenever a graphical image changes.
	OnProgress (see page 14531)	Occurs when a graphical image is in the process of changing.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 2854)	Obtains a reference to the Owner's interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 1590)	Copies an IPicture to the OLE graphic object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 14519)	Constructs an object and initializes its data before the object is first used.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	LoadFromClipboardFormat (see page 1592)	Loads an OLE graphic from the Clipboard into the OLE graphic object.
V	LoadFromFile (see page 1592)	Prevents OleGraphics from trying to load from a file.
V	LoadFromStream (see page 1592)	Loads an IPicture interface from a stream into the OLE graphic object.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	QueryInterface (see page 2854)	Returns a reference to a specified interface if the object supports that interface.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SaveToClipboardFormat (see page 1593)	Introduces an abstract method that assigns the graphic to a variable in clipboard format.
V	SaveToFile (see page 14525)	Saves a graphics image to a file.
V	SaveToStream (see page 1593)	Saves the OLE graphic to a stream.
V	SetSize (see page 14527)	Specifies the size of the graphic.

Public Properties

	Name	Description
R	Empty (see page 14528)	Indicates whether the graphics object contains a graphic.
	Height (see page 14528)	Specifies the vertical size of the graphic in pixels.
R	MMHeight (see page 1594)	Indicates the height of the OLE graphic in 0.01 millimeter units.
R	MMWidth (see page 1595)	Indicates the width of the OLE graphic in 0.01 millimeter units.
	Modified (see page 14528)	Indicates whether the graphics object has been changed or edited.
	Palette (see page 14529)	Indicates the color palette of the graphical image.
	PaletteModified (see page 14529)	Indicates whether the palette has changed.
	Picture (see page 1595)	Specifies the interface reference for which the OLE graphic is a container.
	Transparent (see page 14529)	Indicates whether the image covers its rectangular area.
	Width (see page 14530)	Determines the maximum width of the graphics object in pixels.

12.1.13.2 TOleGraphic Methods

The methods of the TOleGraphic class are listed here.

Protected Methods

	Name	Description
V	Changed (see page 1590)	This is Changed, a member of class TOleGraphic.
V	Draw (see page 1590)	This is Draw, a member of class TOleGraphic.
V	GetEmpty (see page 1591)	This is GetEmpty, a member of class TOleGraphic.
V	GetHeight (see page 1591)	This is GetHeight, a member of class TOleGraphic.
V	GetPalette (see page 1591)	This is GetPalette, a member of class TOleGraphic.
V	GetTransparent (see page 1591)	This is GetTransparent, a member of class TOleGraphic.
V	GetWidth (see page 1591)	This is GetWidth, a member of class TOleGraphic.
V	SetHeight (see page 1593)	This is SetHeight, a member of class TOleGraphic.

	SetPalette (see page 1594)	This is SetPalette, a member of class TOleGraphic.
	SetWidth (see page 1594)	This is SetWidth, a member of class TOleGraphic.

Public Methods

	Name	Description
	Assign (see page 1590)	Copies an IPicture to the OLE graphic object.
	LoadFromClipboardFormat (see page 1592)	Loads an OLE graphic from the Clipboard into the OLE graphic object.
	LoadFromFile (see page 1592)	Prevents OleGraphics from trying to load from a file.
	LoadFromStream (see page 1592)	Loads an IPicture interface from a stream into the OLE graphic object.
	SaveToClipboardFormat (see page 1593)	Introduces an abstract method that assigns the graphic to a variable in clipboard format.
	SaveToStream (see page 1593)	Saves the OLE graphic to a stream.

12.1.13.2.1 TOleGraphic.Assign Method

Copies an IPicture to the OLE graphic object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Assign copies the IPicture contained in Source to the OLE graphic object. Assign then calls the inherited Assign. The IPicture can be a bitmap, metafile or icon image.

See Also

Assign (see page 2930)

12.1.13.2.2 TOleGraphic.Changed Method

Pascal

```
procedure Changed(Sender: TObject); override;
```

C++

```
virtual __fastcall Changed(TObject * Sender);
```

Description

This is Changed, a member of class TOleGraphic.

12.1.13.2.3 TOleGraphic.Draw Method

Pascal

```
procedure Draw(ACanvas: TCanvas; const Rect: TRect); override;
```

C++

```
virtual __fastcall Draw(TCanvas ACanvas, const TRect Rect);
```

Description

This is Draw, a member of class TOleGraphic.

12.1.13.2.4 TOleGraphic.GetEmpty Method

Pascal

```
function GetEmpty: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetEmpty();
```

Description

This is GetEmpty, a member of class TOleGraphic.

12.1.13.2.5 TOleGraphic.GetHeight Method

Pascal

```
function GetHeight: Integer; override;
```

C++

```
virtual __fastcall int GetHeight();
```

Description

This is GetHeight, a member of class TOleGraphic.

12.1.13.2.6 TOleGraphic.GetPalette Method

Pascal

```
function GetPalette: HPALETTE; override;
```

C++

```
virtual __fastcall HPALETTE GetPalette();
```

Description

This is GetPalette, a member of class TOleGraphic.

12.1.13.2.7 TOleGraphic.GetTransparent Method

Pascal

```
function GetTransparent: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetTransparent();
```

Description

This is GetTransparent, a member of class TOleGraphic.

12.1.13.2.8 TOleGraphic.GetWidth Method

Pascal

```
function GetWidth: Integer; override;
```

C++

```
virtual __fastcall int GetWidth();
```

Description

This is GetWidth, a member of class TOleGraphic.

12.1.13.2.9 TOleGraphic.LoadFromClipboardFormat Method

Loads an OLE graphic from the Clipboard into the OLE graphic object.

Pascal

```
procedure LoadFromClipboardFormat(AFormat: Word; AData: THandle; APalette: HPALETTE); override;
```

C++

```
virtual __fastcall LoadFromClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette);
```

Description

LoadFromClipboardFormat is called if the OLE graphic is registered with the TPicture object using the RegisterClipboardFormat method.

LoadFromClipboardFormat replaces the current image with the data pointed to by the AData parameter. The palette for the OLE graphic is specified by the APalette parameter.

See Also

TClipboard (see page 3328)

TPicture (see page 14584)

RegisterClipboardFormat (see page 14592)

12.1.13.2.10 TOleGraphic.LoadFromFile Method

Prevents OleGraphics from trying to load from a file.

Pascal

```
procedure LoadFromFile(const Filename: string); override;
```

C++

```
virtual __fastcall LoadFromFile(const AnsiString Filename);
```

Description

LoadFromFile is overridden in TOleGraphic (see page 1587) so that it will not load a graphic from a file. Attempting to do so is not appropriate for an IPicture.

See Also

LoadFromStream (see page 1592)

SaveToFile (see page 14525)

12.1.13.2.11 TOleGraphic.LoadFromStream Method

Loads an IPicture interface from a stream into the OLE graphic object.

Pascal

```
procedure LoadFromStream(Stream: TStream); override;
```

C++

```
virtual __fastcall LoadFromStream(TStream Stream);
```

Description

To use LoadFromStream specify the stream from which the OLE graphic is to be loaded as the value of Stream.

See Also

TStream (see page 2973)

12.1.13.2.12 TOleGraphic.SaveToClipboardFormat Method

Introduces an abstract method that assigns the graphic to a variable in clipboard format.

Pascal

```
procedure SaveToClipboardFormat(var AFormat: Word; var AData: THandle; var APalette:
HPALETTE); override;
```

C++

```
virtual __fastcall SaveToClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette);
```

Description

Each descendant graphic object defines a SaveToClipboardFormat method that saves that particular graphic image type to a Clipboard format. An application must have registered the format with the TGraphic object using the RegisterClipboardFormat method.

See Also

TClipboard (see page 3328)

RegisterClipboardFormat (see page 14592)

12.1.13.2.13 TOleGraphic.SaveToStream Method

Saves the OLE graphic to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream); override;
```

C++

```
virtual __fastcall SaveToStream(TStream Stream);
```

Description

Use SaveToStream to save the OLE graphic to the stream specified by the Stream parameter. SaveToStream uses the IPersistStream interface of the graphic write it to the stream.

See Also

TStream (see page 2973)

12.1.13.2.14 TOleGraphic.SetHeight Method

Pascal

```
procedure SetHeight(Value: Integer); override;
```

C++

```
virtual __fastcall SetHeight(int Value);
```

Description

This is SetHeight, a member of class TOleGraphic.

12.1.13.2.15 TOleGraphic.SetPalette Method

Pascal

```
procedure SetPalette(Value: HPALETTE); override;
```

C++

```
virtual __fastcall SetPalette(HPALETTE Value);
```

Description

This is SetPalette, a member of class TOleGraphic.

12.1.13.2.16 TOleGraphic.SetWidth Method

Pascal

```
procedure SetWidth(Value: Integer); override;
```

C++

```
virtual __fastcall SetWidth(int Value);
```

Description

This is SetWidth, a member of class TOleGraphic.

12.1.13.3 TOleGraphic Properties

The properties of the TOleGraphic class are listed here.

Public Properties

	Name	Description
R	MMHeight (see page 1594)	Indicates the height of the OLE graphic in 0.01 millimeter units.
R	MMWidth (see page 1595)	Indicates the width of the OLE graphic in 0.01 millimeter units.
	Picture (see page 1595)	Specifies the interface reference for which the OLE graphic is a container.

12.1.13.3.1 TOleGraphic.MMHeight Property

Indicates the height of the OLE graphic in 0.01 millimeter units.

Pascal

```
property MMHeight: Integer;
```

C++

```
__property int MMHeight;
```

Description

MMHeight is used for a more accurate reading of the vertical size of the graphic. The Height property, by contrast, is always in screen device pixel units; to avoid loss of precision in converting between device pixels and millimeters, set or read the

dimensions in millimeters with MMHeight.

See Also

Height

MMWidth (see page 1595)

12.1.13.3.2 TOleGraphic.MMWidth Property

Indicates the width of the OLE graphic in 0.01 millimeter units.

Pascal

```
property MMWidth: Integer;
```

C++

```
__property int MMWidth;
```

Description

MMWidth is used for a more accurate reading of the horizontal size of the graphic. The Width property, by contrast, is always in screen device pixel units; to avoid loss of precision in converting between device pixels and millimeters, set or read the dimensions in millimeters with MMWidth.

See Also

MMHeight (see page 1594)

Width

12.1.13.3.3 TOleGraphic.Picture Property

Specifies the interface reference for which the OLE graphic is a container.

Pascal

```
property Picture: IPicture;
```

C++

```
__property IPicture Picture;
```

Description

Picture is for internal use only.

12.1.14 TOleStream Class

TOleStream reads and writes information over a streaming interface that is provided by an OLE object.

Class Hierarchy

Pascal

```
TOleStream = class(TStream);
```

C++

```
class TOleStream : public TStream;
```

File

AxCtrls

Description

Use TOleStream to create a TStream wrapper for an OLE stream interface. TOleStream uses the streaming interface of the OLE stream to read and write information. TActiveXControl (see page 1517) objects use TOleStream to load themselves from a stream or save themselves to a stream.

See Also

TActiveXControl (see page 1517)

TStreamAdapter (see page 2997)

IStream

12.1.14.1 TOleStream Members

The following tables list the members exposed by TOleStream.

Protected Methods

	Name	Description
	GetIStream (see page 1598)	This is GetIStream, a member of class TOleStream.
V	GetSize (see page 2982)	This is GetSize, a member of class TStream.
V	SetSize (see page 2989)	This is the overview for the SetSize method overload.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CopyFrom (see page 2979)	Copies a specified number of bytes from one stream to another.
	Create (see page 1597)	Creates a TOleStream (see page 1595) object to represent a stream interface.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FixupResourceHeader (see page 2981)	Patches the resource header for a resource that has been written to the stream.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Read (see page 1598)	Reads up to Count bytes from the current position into Buffer.
	ReadBuffer (see page 2983)	Reads bytes from the stream into Buffer.
	ReadComponent (see page 2983)	Initiates streaming of components and their properties.
	ReadComponentRes (see page 2987)	Reads components and their properties from a stream in a resource file format.
	ReadResHeader (see page 2987)	Reads a resource-file header from the stream.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Seek (see page 1598)	Resets the current position of the stream.
V	Seek (see page 2988)	This is the overview for the Seek method overload.
V	Write (see page 1599)	Writes Count bytes from Buffer to the current position in the Ole stream and updates the current position by Count bytes.
	WriteBuffer (see page 2990)	Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.
	WriteComponent (see page 2990)	Initiates the writing of components and their properties to a stream.
	WriteComponentRes (see page 2994)	Writes components and their properties to a stream from a resource file format.
	WriteDescendent (see page 2994)	Streams components and their properties in inherited forms.
	WriteDescendentRes (see page 2995)	Streams components and their properties in inherited forms.
	WriteResourceHeader (see page 2995)	Writes a resource-file header to the stream.

Public Properties

	Name	Description
	Position (see page 2996)	Indicates the current offset into the stream for reading and writing.
	Size (see page 2997)	Indicates the size in bytes of the stream.

12.1.14.2 TOleStream.Create Constructor

Creates a TOleStream (see page 1595) object to represent a stream interface.

Pascal

```
constructor Create(const Stream: IStream);
```

C++

```
__fastcall TOleStream(const IStream Stream);
```

Description

Call Create to convert a stream interface into a TOleStream (see page 1595) object. The Stream parameter is the stream interface that the TOleStream (see page 1595) object uses for reading and writing.

12.1.14.3 TOleStream Methods

The methods of the TOleStream class are listed here.

Protected Methods

	Name	Description
	GetIStream (see page 1598)	This is GetIStream, a member of class TOleStream.

Public Methods

	Name	Description
V	Read (see page 1598)	Reads up to Count bytes from the current position into Buffer.

	Seek (see page 1598)	Resets the current position of the stream.
	Write (see page 1599)	Writes Count bytes from Buffer to the current position in the Ole stream and updates the current position by Count bytes.

12.1.14.3.1 TOleStream.GetIStream Method

Pascal

```
function GetIStream: IStream;
```

C++

```
__fastcall IStream GetIStream();
```

Description

This is GetIStream, a member of class TOleStream.

12.1.14.3.2 TOleStream.Read Method

Reads up to Count bytes from the current position into Buffer.

Pascal

```
function Read(var Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Read( Buffer, Longint Count);
```

Description

Call Read to read data from the Ole stream when it is not known how many bytes are available. Buffer must have at least Count bytes allocated to hold the data that was read from the stream.

Read uses the stream interface to transfer up to Count bytes into Buffer, starting at the current position, and then advances the current position by the number of bytes actually transferred. Read returns the number of bytes actually transferred (which may be less than the number requested in Count.)

All the other data-reading methods of an Ole stream (ReadBuffer, ReadComponent) call Read to do their actual reading.

See Also

Seek (see page 1598)

Write (see page 1599)

12.1.14.3.3 TOleStream.Seek Method

Resets the current position of the stream.

Pascal

```
function Seek(Offset: Longint; Origin: Word): Longint; override;
```

C++

```
virtual __fastcall Longint Seek(Longint Offset, Word Origin);
```

Description

Use Seek to move the current position by the indicated offset. Seek uses the stream interface to change the current position to the value indicated by the Offset and Origin parameters.

The Origin parameter indicates how to interpret the Offset parameter. Origin should be one of the following values:

Value	Meaning
soFromBeginning	Offset is from the beginning of the Ole stream. Seek moves to the position Offset. Offset must be >= 0.
soFromCurrent	Offset is from the current position in the Ole stream. Seek moves to Position + Offset.
soFromEnd	Offset is from the end of the Ole stream. Offset must be <= 0 to indicate a number of bytes before the end of the stream.

Seek returns the new value of the Position property.

See Also

Read (see page 1598)

Write (see page 1599)

12.1.14.3.4 TOleStream.Write Method

Writes Count bytes from Buffer to the current position in the Ole stream and updates the current position by Count bytes.

Pascal

```
function Write(const Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  Buffer, Longint Count);
```

Description

Use Write to write Count bytes to the Ole stream, starting at the current position. Write uses the stream interface to transfer the information from Buffer.

All the other data-writing methods of an Ole stream (WriteBuffer, WriteComponent) call Write to do their actual writing.

See Also

Seek (see page 1598)

Read (see page 1598)

12.1.15 TPictureAdapter Class

TPictureAdapter connects a native TPicture object to an OLE picture object.

Class Hierarchy

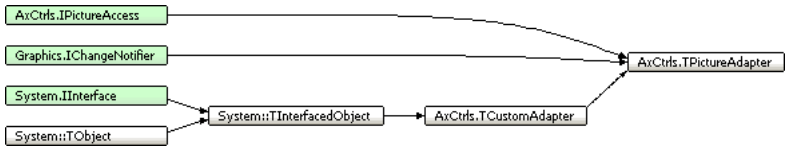

Pascal

```
TPictureAdapter = class(TCustomAdapter, IChangeNotifier, IPictureAccess);
```

C++

```
class TPictureAdapter : public TCustomAdapter, public IChangeNotifier, public IPictureAccess;
```

File

AxCtrls

Description

To create an instance of a TPictureAdapter, call the GetOlePicture (see page 1602) procedure, which creates the binding between a TPicture and an OLE picture object.

See Also

GetOlePicture

SetOlePicture

TFontAdapter (see page 1583)

TPicture (see page 14584)

12.1.15.1 TPictureAdapter Members

The following tables list the members exposed by TPictureAdapter.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Fields

	Name	Description
	Updating (see page 1581)	This is Updating, a member of class TCustomAdapter.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
V	Changed (see page 1582)	Responds when the VCL representation of the OLE object changes.
	ConnectOleObject (see page 1582)	Associates the adapter with an OLE interface.
	GetOlePicture (see page 1602)	This is GetOlePicture, a member of class TPictureAdapter.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	ReleaseOleObject (see page 1582)	Releases the associated OLE interface, allowing it to decrease its reference count.
	SetOlePicture (see page 1602)	This is SetOlePicture, a member of class TPictureAdapter.
V	Update (see page 1602)	Provides the interface for a method that responds to changes in the OLE object.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	Changed (see page 14607)	Called when a change occurs in the implementing class.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1601)	Create constructs an instance of the picture adapter control.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 1581)	Destroys the custom adapter object.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

12.1.15.2 TPictureAdapter.Create Constructor

Create constructs an instance of the picture adapter control.

Pascal

```
constructor Create(Picture: TPicture);
```

C++

```
__fastcall TPictureAdapter(TPicture Picture);
```

Description

Do not call the constructor for this class directly. Instead, call the GetOlePicture (see page 1602) procedure to create the binding between a TPicture and an OLE picture.

See Also

Destroy

GetOlePicture

12.1.15.3 TPictureAdapter Methods

The methods of the TPictureAdapter class are listed here.

Protected Methods

	Name	Description
	GetOlePicture (see page 1602)	This is GetOlePicture, a member of class TPictureAdapter.
	SetOlePicture (see page 1602)	This is SetOlePicture, a member of class TPictureAdapter.
V	Update (see page 1602)	Provides the interface for a method that responds to changes in the OLE object.

12.1.15.3.1 TPictureAdapter.GetOlePicture Method

Pascal

```
procedure GetOlePicture(var OlePicture: IPictureDisp);
```

C++

```
__fastcall GetOlePicture(IPictureDisp OlePicture);
```

Description

This is GetOlePicture, a member of class TPictureAdapter.

12.1.15.3.2 TPictureAdapter.SetOlePicture Method

Pascal

```
procedure SetOlePicture(const OlePicture: IPictureDisp);
```

C++

```
__fastcall SetOlePicture(const IPictureDisp OlePicture);
```

Description

This is SetOlePicture, a member of class TPictureAdapter.

12.1.15.3.3 TPictureAdapter.Update Method

Provides the interface for a method that responds to changes in the OLE object.

Pascal

```
procedure Update; override;
```

C++

```
virtual __fastcall Update();
```

Description

Descendants of TCustomAdapter override Update to modify the associated VCL object so that it reflects changes in the corresponding OLE object. It is the converse of the Changed method.

See Also

Changed (see page 1582)

12.1.16 TPropertyPage Class

TPropertyPage is the base class for controls that represent an ActiveX control's property page.

Class Hierarchy

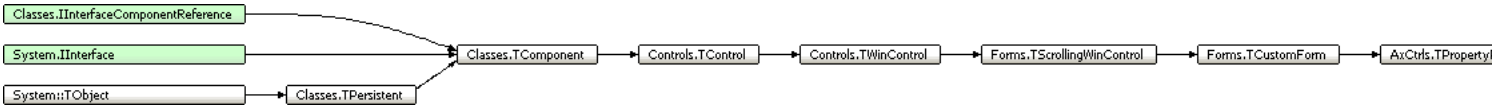

Pascal

```
TPropertyPage = class(TCustomForm);
```

C++

```
class TPropertyPage : public TCustomForm;
```

File

AxCtrls

Description

TPropertyPage extends TCustomForm by adding the functionality needed to embed the form in an ActiveX property page dialog. It also adds methods to copy property values from the object to the form's controls, and vice versa.

To implement your own property page for an ActiveX object, derive a new class from TPropertyPage, design the form, then override the UpdateObject (see page 1618) and UpdatePropertyPage (see page 1619) methods. The IDE provides an expert for creating a skeleton property page. See Creating a new property page for more information.

TPropertyPage provides a helper method, Modified (see page 1618), that your form can call when any of the controls on the form are changed by the user. Calling the Modified (see page 1618) method notifies the property dialog that the user has modified the object's properties. Normally this will enable the Apply button on the property dialog.

The property page maintains a connection to the object it is editing, using the OleObject (see page 1635) or OleObjects (see page 1635) property. This property is a Variant and can be used to read and write the object's properties.

See Also

TActiveXPropertyPageFactory (see page 1564)

Creating a New Property Page

12.1.16.1 TPropertyPage Members

The following tables list the members exposed by TPropertyPage.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnCanResize (see page 13947)	Occurs when an attempt is made to resize the control.
	OnCloseQuery (see page 13950)	Occurs when close is attempted.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.

	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnHelp (see page 13959)	Occurs when the form receives a request for help.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseActivate (see page 13967)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnShortCut (see page 13977)	Occurs when the user presses a key (before the OnKeyDown (see page 13962) event).
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFormState (see page 13863)	This is FFormState, a member of class TCustomForm.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	Activate (see page 13865)	Activation event dispatcher.
	ActiveChanged (see page 13865)	Active (see page 13896) state event dispatcher.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 14324)	Adjusts the ClientRect property to exclude the scroll bars.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.

	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	AlignControls (see page 13866)	Aligns all child controls of the form that fall within a specified rectangle.
	AlignControls (see page 14325)	Aligns the controls in the scrolling windowed control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	AutoScrollEnabled (see page 14325)	Indicates whether automatic scrolling is enabled.
	AutoScrollInView (see page 14325)	Scrolls a control into the visible area of the scrolling windowed control.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	BeginAutoDrag (see page 13867)	Responds when the user starts dragging the form if DragMode is dmAutomatic.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	ChangeScale (see page 13867)	Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.
	ChangeScale (see page 14326)	Resizes the scrolling windowed control by a specific ratio.
	Click (see page 5743)	Respond to user click.
	CloseModal (see page 13868)	This is CloseModal, a member of class TCustomForm.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 13868)	Initializes the window-creation parameter record when the form window is created.
	CreateParams (see page 14326)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWindowHandle (see page 13869)	Creates a form window given a set of window-creation parameters.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 13869)	Creates the form window.
	CreateWnd (see page 14327)	Creates a Windows control corresponding to the TScrollingWinControl (see page 14312) object.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	Deactivate (see page 13870)	Generates an OnDeactivate (see page 13953) event.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.

	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DefineProperties (see page 13871)	Designates methods for storing a form's unpublished data on a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyHandle (see page 13871)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWindowHandle (see page 13872)	Destroys the window created in the CreateWindowHandle (see page 13869) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoClose (see page 13873)	Form closing event dispatcher.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoCreate (see page 13873)	Form creation event dispatcher.
	DoDestroy (see page 13873)	Form destruction event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDock (see page 13874)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoFlipChildren (see page 14327)	Flips the control's children.
	DoHide (see page 13874)	Form hide event dispatcher.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoShow (see page 13875)	Form show event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorderIconStyles (see page 13875)	This is GetBorderIconStyles, a member of class TCustomForm.
	GetBorderStyles (see page 13876)	This is GetBorderStyles, a member of class TCustomForm.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.

	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetChildren (see page 13876)	Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetClientRect (see page 13876)	Returns the value of the ClientRect property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloating (see page 13877)	Returns the value of the Floating property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetOwnerWindow (see page 13878)	This is GetOwnerWindow, a member of class TCustomForm.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 13878)	Handles exceptions that are raised by the OnCreate (see page 13951) event handler.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 13880)	Performs fix-ups when the form is first loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.

	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	Notification (see page 13882)	Responds to notifications that objects are about to be inserted or removed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 13882)	Generates an OnPaint (see page 13975) event.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaintWindow (see page 13882)	Renders the image of the form.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteChanged (see page 13883)	Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	QueryInterface (see page 13884)	Returns a reference to a specified interface if that interface is supported by the form.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	ReadState (see page 13884)	Reads the form's published properties, stored data, and owned components from a reader object.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	RequestAlign (see page 13885)	Instructs the form's parent (if any) to reposition the form, enforcing its Align property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	Resizing (see page 13885)	Makes any required adjustments when the form changes size.
	Resizing (see page 14328)	This is Resizing, a member of class TScrollingWinControl.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetChildOrder (see page 13886)	Changes the order in which Child appears when GetChildren (see page 13876) is called.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.

	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParent (see page 13887)	Sets the Parent property.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
V	SetParentBiDiMode (see page 13887)	Sets the ParentBiDiMode property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
V	UpdateActions (see page 13891)	Updates all actions associated with the form.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	UpdateWindowState (see page 13892)	Updates the WindowState (see page 13942) property to reflect the current state of the form.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	ValidateRename (see page 13892)	Determines if a form can rename one of its owned components.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	VisibleChanging (see page 13893)	Responds just before the form changes the value of its Visible (see page 13940) property.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WndProc (see page 13893)	Provides specific message responses for the form.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
R	ActiveMDIChild (see page 13897)	Specifies the MDI child that has focus.
	AlphaBlend (see page 13898)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 13898)	Specifies the degree of translucency on a translucent form.

	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderIcons (see page 13899)	Specifies which icons appear on the title bar of the form.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	ClientHandle (see page 13904)	Provides access to the handle of the form's internal MDI client window.
	DefaultMonitor (see page 13909)	Specifies the monitor on which the form appears.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	FormStyle (see page 13912)	Determines the form's style.
	Icon (see page 13917)	Specifies the icon that appears when the form is minimized.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	MDIChildCount (see page 13921)	Specifies the number of open MDI child forms.
R	MDIChildren (see page 13922)	Provides indexed access to all MDI child forms.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ObjectMenuItem (see page 13926)	Represents an OLE object menu item that reacts to selections of OLE objects.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
R	PopupChildren (see page 13928)	This is PopupChildren, a member of class TCustomForm.
	Position (see page 13933)	Represents the size and placement of the form.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	TileMode (see page 13936)	Represents how MDI child forms are arranged when the Tile method is called.
	TransparentColor (see page 13938)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 13939)	Indicates the color on the form that appears transparent when TransparentColor (see page 13938) is true.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowMenu (see page 13941)	Specifies the Window menu for an MDI parent form.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 13866)	Construction event dispatcher.

	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 13866)	Destruction event dispatcher.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Close (see page 13867)	Closes the form.
	CloseQuery (see page 13868)	Close (see page 13867) attempt event dispatcher.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 1616)	Creates an instance of TPropertyPage (see page 1603).
	CreateNew (see page 13862)	Creates and initializes a new form.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 13870)	Provides message handling for all messages that the form does not fully process.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DefocusControl (see page 13871)	Removes focus from a control on the form.
	Destroy (see page 1617)	Frees an instance of TPropertyPage (see page 1603).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	DisableAutoRange (see page 14327)	Disables automatic scrolling.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	Dock (see page 13872)	Handles automatic docking of the form into other windows.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EnableAutoRange (see page 14328)	Reenables automatic scrolling.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	EnumCtlProps (see page 1617)	Looks for a PropType type property in a control on the form.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	FocusControl (see page 13875)	Sets focus to a control on the form.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFormImage (see page 13877)	Returns a bitmap of the form.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	Hide (see page 13879)	Hides the form.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	IsShortCut (see page 13880)	Processes shortcut keys when the form has focus.
	MakeFullyVisible (see page 13881)	Ensures that the form is fully visible on a specified monitor.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	Modified (see page 1618)	Notifies the property page's container that the user has edited one of the controls on the page.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MouseWheelHandler (see page 13881)	Directs mouse wheel messages to the appropriate control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	Print (see page 13883)	Prints the form.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	RecreateAsPopup (see page 13884)	This is RecreateAsPopup, a member of class TCustomForm.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	Release (see page 13885)	Destroys the form and frees its associated memory.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollInView (see page 14328)	Scrolls a control into the visible area of the scrolling windowed control.
	SendCancelMode (see page 13886)	Cancels modes on the form.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
V	SetFocus (see page 13886)	Sets focus to the form.
V	SetFocusedControl (see page 13887)	Sets focus to a control on the form.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Show (see page 13888)	Shows the form.
V	ShowModal (see page 13890)	Shows a form as a modal dialog.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UpdateGlassFrame (see page 13892)	This is UpdateGlassFrame, a member of class TCustomForm.
V	UpdateObject (see page 1618)	UpdateObject is called to copy property values from the controls on the form to the ActiveX Object.
V	UpdatePropertyPage (see page 1619)	Copies the property values from the ActiveX object to controls on the property page.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.
V	WantChildKey (see page 13893)	Indicates whether the form processes keyboard input for a control it owns.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Action (see page 13895)	Designates the action associated with the control.
R	Active (see page 13896)	Specifies whether the form has focus.
	ActiveOleControl (see page 13897)	Specifies the OLE control on the form that reacts to changes in focus.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 13900)	Specifies the appearance and behavior of the form border.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 13900)	Provides access to the drawing area of the form.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designer (see page 13910)	Specifies the designer interface for the form.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropTarget (see page 13910)	Specifies whether the form is the target of a drag-and-drop operation.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	FormState (see page 13911)	Indicates transitional state information about the form.
	GlassFrame (see page 13914)	This is GlassFrame, a member of class TCustomForm.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HelpFile (see page 13916)	Specifies the name of the file the form uses to display Help.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.

	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Menu (see page 13923)	Specifies the form's main menu.
	ModalResult (see page 13924)	Represents the return value of a form that is used as a modal dialog.
R	Monitor (see page 13926)	Provides access to the monitor on which the form appears.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	OleFormObject (see page 13927)	Specifies the IOleForm (see page 14340) interface for an in-place OLE object contained in the form.
R	OleObject (see page 1635)	Contains a reference to the first item in the array of ActiveX controls being edited by the page.
	OleObjects (see page 1635)	An array of Dispatch IDs for each ActiveX control being edited by the page.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PopupMode (see page 13933)	This is PopupMode, a member of class TCustomForm.
	PopupParent (see page 13933)	Sets an order for stacked forms that users cannot change.
	ScreenSnap (see page 13935)	Specifies whether form snaps to edge of screen.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	SnapBuffer (see page 13936)	Specifies distance for screen snap.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.
	WindowState (see page 13942)	Represents how the form appears on the screen.

Published Events

	Name	Description
	OnActivate (see page 1646)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnClick (see page 1650)	Occurs when the user clicks the control.
	OnClose (see page 1652)	Occurs when the form closes.
	OnContextPopup (see page 1653)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 1654)	Occurs when the form is created.
	OnDblClick (see page 1655)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 1656)	Occurs when the form loses focus.
	OnDestroy (see page 1656)	Occurs when the form is destroyed.
	OnDragDrop (see page 1657)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1659)	Occurs when the user drags an object over a control.
	OnHide (see page 1662)	Occurs when the form is hidden (that is, when its Visible property is set to false).
	OnKeyDown (see page 1664)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 1667)	Occurs when a key is pressed.
	OnKeyUp (see page 1668)	Occurs when the user releases a key that has been pressed.
	OnMouseDown (see page 1669)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 1672)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1674)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 1676)	Occurs when the form is redrawn.

	OnResize (see page 1677)	Occurs immediately after the control is resized.
	OnShow (see page 1678)	Occurs when the form is shown (that is, when its Visible property is set to true).

Published Properties

	Name	Description
	ActiveControl (see page 1620)	Specifies the control that has focus on the form.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AutoScroll (see page 1621)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	Caption (see page 1621)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 1624)	Specifies the height (in pixels) of the form's client area.
	ClientWidth (see page 1624)	Specifies the width (in pixels) of the form's client area.
	Color (see page 1626)	Specifies the background color of the control.
	Ctl3D (see page 1627)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Enabled (see page 1628)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1628)	Controls the attributes of text written on or in the control.
	Height (see page 1629)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HorzScrollBar (see page 1630)	Represents the horizontal scroll bar for the scrolling windowed control.
	KeyPreview (see page 1631)	Specifies whether the form should receive keyboard events before the active control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Left (see page 13920)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 1634)	Specifies when OnCreate and OnDestroy events occur.
	ParentFont (see page 1636)	Determines where a control looks for its font information.
	PixelsPerInch (see page 1636)	Represents the proportion of the font on the system on which the form was designed.
	PopupMenu (see page 1636)	Identifies the pop-up menu associated with the control.
	PrintScale (see page 1641)	Represents the proportions of a printed form.
	Scaled (see page 1641)	Specifies whether the form is sized according to the value of the PixelsPerInch property.
	ShowHint (see page 1642)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Top (see page 13937)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	VertScrollBar (see page 1643)	Represents the vertical scroll bar for the scrolling windowed control.
	Visible (see page 1644)	Indicates whether the form is visible.
	Width (see page 1645)	Specifies the horizontal size of the control or form in pixels.

12.1.16.2 TPropertyPage.Create Constructor

Creates an instance of TPropertyPage (see page 1603).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TPropertyPage(TComponent * AOwner);
```

Description

Call Create to instantiate a property page at runtime. Property pages added at design time are created automatically. Pass a single Component as an argument to assign the property page an owner that is responsible for freeing the property page.

Create instantiates the helper objects used by the property page.

12.1.16.3 TPropertyPage.Destroy Destructor

Frees an instance of TPropertyPage (see page 1603).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPropertyPage();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which verifies that the property page reference is not nil, and only then calls Destroy.

Destroy frees any associated OLE objects before freeing the object instance.

See Also

Free (see page 22300)

12.1.16.4 TPropertyPage Methods

The methods of the TPropertyPage class are listed here.

Public Methods

	Name	Description
	EnumCtlProps (see page 1617)	Looks for a PropType type property in a control on the form.
	Modified (see page 1618)	Notifies the property page's container that the user has edited one of the controls on the page.
V	UpdateObject (see page 1618)	UpdateObject is called to copy property values from the controls on the form to the ActiveX Object.
V	UpdatePropertyPage (see page 1619)	Copies the property values from the ActiveX object to controls on the property page.

12.1.16.4.1 TPropertyPage.EnumCtlProps Method

Looks for a PropType type property in a control on the form.

Pascal

```
procedure EnumCtlProps(PropType: TGUID; PropNames: TStrings);
```

C++

```
__fastcall EnumCtlProps(TGUID PropType, TStrings PropNames);
```

Description

Use EnumCtlProps to locate all properties of the ActiveX object specified by OleObject (see page 1635) that use the GUID specified by the PropType parameter. The names of all properties of the specified type are added to the PropNames list. The property names can be used subsequently to read or write property values using the IDispatch interface of the ActiveX object.

Note: EnumCtlProps does not clear the PropNames list before adding property names. Applications should clear the string list before calling EnumCtlProps to avoid mistaking entries that were previously in the list for properties of the ActiveX object.

See Also

UpdateObject (see page 1618)

UpdatePropertyPage (see page 1619)

OleObject (see page 1635)

12.1.16.4.2 TPropertyPage.Modified Method

Notifies the property page's container that the user has edited one of the controls on the page.

Pascal

```
procedure Modified;
```

C++

```
__fastcall Modified();
```

Description

Call Modified from the controls on the property page when their values change. The property dialog will enable the Apply button in response to this method call.

12.1.16.4.3 TPropertyPage.UpdateObject Method

UpdateObject is called to copy property values from the controls on the form to the ActiveX Object.

Pascal

```
procedure UpdateObject; virtual;
```

C++

```
virtual __fastcall UpdateObject();
```

Description

Override this method to copy values from the controls on the property page to properties of the ActiveX Object. Cast the value of the OleObject (see page 1635) property to an IDispatch interface to set the property values.

UpdateObject is the reverse of UpdatePropertyPage (see page 1619), which copies the property values from the ActiveX Object to controls on the form.

See Also

UpdatePropertyPage (see page 1619)

OleObject (see page 1635)

12.1.16.4.4 TPropertyPage.UpdatePropertyPage Method

Copies the property values from the ActiveX object to controls on the property page.

Pascal

```
procedure UpdatePropertyPage; virtual;
```

C++

```
virtual __fastcall UpdatePropertyPage();
```

Description

This method is normally called when the property page is first shown, or when the user undoes changes to the property dialog box. The UpdatePropertyPage method implemented by TPropertyPage (see page 1603) does nothing. Override this method to copy the property values from the ActiveX object to the controls on the property page.

UpdatePropertyPage is the reverse of UpdateObject (see page 1618), which copies the property values from controls on the property page to the ActiveX object.

See Also

UpdateObject (see page 1618)

OleObject (see page 1635)

12.1.16.5 TPropertyPage Properties

The properties of the TPropertyPage class are listed here.

Public Properties

	Name	Description
R	OleObject (see page 1635)	Contains a reference to the first item in the array of ActiveX controls being edited by the page.
	OleObjects (see page 1635)	An array of Dispatch IDs for each ActiveX control being edited by the page.

Published Properties

	Name	Description
	ActiveControl (see page 1620)	Specifies the control that has focus on the form.
	AutoScroll (see page 1621)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	Caption (see page 1621)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 1624)	Specifies the height (in pixels) of the form's client area.
	ClientWidth (see page 1624)	Specifies the width (in pixels) of the form's client area.
	Color (see page 1626)	Specifies the background color of the control.
	Ctl3D (see page 1627)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Enabled (see page 1628)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 1628)	Controls the attributes of text written on or in the control.
	Height (see page 1629)	Specifies the vertical size of the control in pixels.
	HorzScrollBar (see page 1630)	Represents the horizontal scroll bar for the scrolling windowed control.
	KeyPreview (see page 1631)	Specifies whether the form should receive keyboard events before the active control.
	OldCreateOrder (see page 1634)	Specifies when OnCreate and OnDestroy events occur.
	ParentFont (see page 1636)	Determines where a control looks for its font information.
	PixelsPerInch (see page 1636)	Represents the proportion of the font on the system on which the form was designed.
	PopupMenu (see page 1636)	Identifies the pop-up menu associated with the control.

	PrintScale (see page 1641)	Represents the proportions of a printed form.
	Scaled (see page 1641)	Specifies whether the form is sized according to the value of the PixelsPerInch property.
	ShowHint (see page 1642)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	VertScrollBar (see page 1643)	Represents the vertical scroll bar for the scrolling windowed control.
	Visible (see page 1644)	Indicates whether the form is visible.
	Width (see page 1645)	Specifies the horizontal size of the control or form in pixels.

12.1.16.5.1 TPropertyPage.ActiveControl Property

Specifies the control that has focus on the form.

Pascal

```
property ActiveControl: TWinControl;
```

C++

```
__property TWinControl * ActiveControl;
```

Description

Use ActiveControl to get or set the control that has focus on the form. Only one control can have focus at a given time in an application.

If the form does not have focus, ActiveControl is the control on the form that will receive focus when the form receives focus.

Note: When focus shifts to another control, ActiveControl is updated before the OnExit event occurs.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Timer1->Interval = 100;
  if (ActiveControl)
    ActiveControl->Left = ActiveControl->Left + 1;
}
```

Delphi Examples:

```
{
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  if ActiveControl <> nil then
    ActiveControl.Left := ActiveControl.Left + 1;
end;
```

See Also

ActiveOleControl (see page 13897)

OnExit (see page 6221)

OnEnter (see page 6219)

Active (see page 13896)

12.1.16.5.2 TPropertyPage.AutoScroll Property

Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

If AutoScroll is true, the scroll bars appear automatically when necessary. For example, if the user resizes the control so that some of its controls are partially obscured, scroll bars appear.

If AutoScroll is false, scroll bars don't appear automatically. In this case, use the HorzScrollBar and VertScrollBar properties to make scroll bars appear.

See Also

AutoScrollInView (see page 14325)

HorzScrollBar (see page 14329)

VertScrollBar (see page 14330)

12.1.16.5.3 TPropertyPage.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
```

```
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
```

```
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;
```

```
procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

12.1.16.5.4 TPropertyPage.ClientHeight Property

Specifies the height (in pixels) of the form's client area.

Pascal

```
property ClientHeight: Integer;
```

C++

```
__property int ClientHeight;
```

Description

Use ClientHeight to determine the height (in pixels) of the form's client area. The client area is the usable area inside the form's border, excluding the title bar, scroll bars, and so on.

Set ClientHeight to change the height of the form's window based on the desired client area. To change the height of the form's window based on the total size of the window (including the border, menu, status bar and so on), use the Height property instead.

Delphi Examples:

```
{
This example reduces the height of the form's client area by
half when the user clicks the button on the form:
Note that only the client area is halved, not the entire form.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ClientHeight := ClientHeight div 2;
end;
```

See Also

ClientRect

ClientWidth (see page 13905)

Height (see page 5825)

12.1.16.5.5 TPropertyPage.ClientWidth Property

Specifies the width (in pixels) of the form's client area.

Pascal

```
property ClientWidth: Integer;
```

C++

```
__property int ClientWidth;
```

Description

Use ClientWidth to determine the width (in pixels) of the form's client area. The client area is the usable area inside the form's border. Set ClientWidth to change the width of the form's window based on the desired client area. To change the width of the form's window based on the total size of the window (including the border, status bar and so on), use the Width property instead.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    ClientWidth := 300;
    with HorzScrollBar do
    begin
      { Set the range to twice the ClientWidth of the form }
      { This means that the form's logical size is twice as big }
      { as the physical window. }
      { Note that Range must always be larger than the ClientWidth }
      Range := 600;
      Position := Range - ClientWidth; { start form out fully scrolled }
      Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
      Visible := True;  { Show the scrollbar }
    end;
  end;
  {
  This example shows how to use a splitter to divide a form
  into resizable panes.  All objects on the form are created
  dynamically in the OnCreate event handler of the form. To
  run this example, add ExtCtrls and FileCtrl to the uses
  clause of the form's implementation section.
  Note that when creating a splitter dynamically at runtime,
  it is important to set its position to the appropriate side
  of the control it will resize.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    Split: TSplitter;
    Files: TFileListBox;
    Dirs: TDirectoryListBox;
  begin
    { first add a directory list box to the form }
    Dirs := TDirectoryListBox.Create(Form1);
    { line it up on the left of the form }
    Dirs.Parent := Form1;
    Dirs.Align := alLeft;
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
```

See Also

ClientHeight (see page 13905)

ClientRect

Width (see page 5849)

12.1.16.5.6 TPropertyPage.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

12.1.16.5.7 TPropertyPage.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

12.1.16.5.8 TPropertyPage.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

12.1.16.5.9 TPropertyPage.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
```

```
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

12.1.16.5.10 TPropertyPage.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
```

```
  observe the effect on the selected text.
  */
  #include <ComCtrls.hpp>

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    for (int I = 0; I <= 2; I++)
    {
      TRichEdit *pRich = new TRichEdit(this);
      pRich->Parent = this;
      pRich->Top = pRich->Height * I;
      pRich->Text = "This text will take the focus.";
      pRich->SelStart = 0;
      pRich->SelLength = pRich->Text.Length();
      if (I == 1)
        pRich->HideSelection = false;
    }
  }
```

Delphi Examples:

```
  {
  HideSelection, Height example
  This example needs only a blank form. Three Rich Edit
  controls are placed vertically on the form, each contain a
  line of text, and the second form has HideSelection set to
  false.  Try tabbing between the three Rich Edit controls and
  observe the effect on the selected text.
  }
  uses ComCtrls;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    I: Integer;
  begin
    for I := 0 to 2 do
      with TRichEdit.Create(Self) do
      begin
        Parent := Self;
        Top := Height * I;
        Text := 'This text will take the focus.';
        SelStart := 0;
        SelLength := Length(Text);
        if (I = 1) then
          HideSelection := False;
      end;
  end;
```

See Also

ClientHeight (see page 5808)

Width (see page 5849)

Using TDBListBox and TDBComboBox

12.1.16.5.11 TPropertyPage.HorzScrollBar Property

Represents the horizontal scroll bar for the scrolling windowed control.

Pascal

```
  property HorzScrollBar: TControlScrollBar;
```

C++

```
  __property TControlScrollBar HorzScrollBar;
```

Description

Use HorzScrollBar to hide, show, or manipulate the horizontal scroll bar for the scrolling windowed control.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form's logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
    Range := 600;
    Position := Range - ClientWidth; { start form out fully scrolled }
    Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
    Visible := True;  { Show the scrollbar }
  end;
end;
```

See Also

AutoScroll (see page 14329)

VertScrollBar (see page 14330)

TControlScrollBar (see page 13744)

12.1.16.5.12 TPropertyPage.KeyPreview Property

Specifies whether the form should receive keyboard events before the active control.

Pascal

```
property KeyPreview: Boolean;
```

C++

```
__property Boolean KeyPreview;
```

Description

If KeyPreview is true, keyboard events occur on the form before they occur on the active control. (The active control is specified by the ActiveControl property.)

If KeyPreview is false, keyboard events occur only on the active control.

Navigation keys (Tab, BackTab, the arrow keys, and so on) are unaffected by KeyPreview because they do not generate keyboard events. Similarly, when a button has focus or when its Default property is true, the Enter key is unaffected by KeyPreview because it does not generate a keyboard events.

KeyPreview is false by default.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
```

```
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

ActiveControl (see page 13896)

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

OnKeyPress (see page 6227)

12.1.16.5.13 TPropertyPage.OldCreateOrder Property

Specifies when OnCreate and OnDestroy events occur.

Pascal

```
property OldCreateOrder: Boolean;
```

C++

```
__property Boolean OldCreateOrder;
```

Description

When OldCreateOrder is false (the default) the OnCreate event occurs after all constructors are finished and the OnDestroy event occurs before any destructors are called.

This differs from earlier versions of the library (distributed with Delphi version 3 and previous and C++Builder 1), where the OnCreate event occurred when the TCustomForm constructor executed and the OnDestroy event occurred when the TCustomForm destructor executed. Applications that require the OnCreate event and OnDestroy events to occur from the constructor and destructor of the form can set OldCreateOrder to true.

See Also

OnCreate (see page 13951)

OnDestroy (see page 13953)

BeforeDestruction (see page 13866)

AfterConstruction (see page 13866)

12.1.16.5.14 TPropertyPage.OleObject Property

Contains a reference to the first item in the array of ActiveX controls being edited by the page.

Pascal

```
property OleObject: OleVariant;
```

C++

```
__property OleVariant OleObject;
```

Description

Property pages have the ability to be "hooked up to" multiple ActiveX controls, usually by multi-selecting the ActiveX controls in the form designer, and right clicking to invoke the property pages. OleObject points to the first control in the array. To access other ActiveX controls, use the property, OleObjects (see page 1635).

Cast the value of OleObject to the appropriate interface, such as IUnknown or IDispatch, and use the methods of the interface to access the property values and methods of the control.

See Also

UpdatePropertyPage (see page 1619)

UpdateObject (see page 1618)

OleObjects (see page 1635)

Updating the Object

Updating the Property Page

12.1.16.5.15 TPropertyPage.OleObjects Property

An array of Dispatch IDs for each ActiveX control being edited by the page.

Pascal

```
property OleObjects: TInterfaceList;
```

C++

```
__property TInterfaceList OleObjects;
```

Description

Property pages have the ability to be "hooked up to" multiple ActiveX controls, usually by multi-selecting the ActiveX controls in the form designer, and right clicking to invoke the property pages. Cast the value of an entry from the OleObjects Items property to the appropriate interface, such as IUnknown or IDispatch, and use the methods of the interface to access the property values and methods of the control.

See Also

UpdatePropertyPage (see page 1619)

UpdateObject (see page 1618)

OleObject (see page 1635)

Items (see page 2866)

12.1.16.5.16 TPropertyPage.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

12.1.16.5.17 TPropertyPage.PixelsPerInch Property

Represents the proportion of the font on the system on which the form was designed.

Pascal

```
property PixelsPerInch: Integer;
```

C++

```
__property int PixelsPerInch;
```

Description

Use PixelsPerInch at runtime to change how the form is scaled to the current screen resolution. If PixelsPerInch is changed from its default value, the form will not have the same proportion on every screen resolution.

At design time, PixelsPerInch is set automatically when the form is saved. Changes to PixelsPerInch made in the Object Inspector only affect the form at runtime.

The form's Scaled property must be true for PixelsPerInch to have an effect.

See Also

Scaled (see page 13935)

12.1.16.5.18 TPropertyPage.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
```

```
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      const int num_items = 4;
      for (int index = 0; index < num_items; ++index)
      {
          TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
          PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
          NewItem->Caption = "Menu Item " + IntToStr(index);
          NewItem->Tag = index;
          NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
          TNotifyEvent();
      }
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
          {
              case 0:
              {
                  ShowMessage("first item clicked");
                  break;
              }
              case 1:
              {
                  ShowMessage("second item clicked");
                  break;
              }
              case 2:
              {
                  ShowMessage("third item clicked");
                  break;
              }
              case 3:
              {
                  ShowMessage("fourth item clicked");
                  break;
              }
          }
      }
  }

  /*
```

```
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
```

```
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
```

```
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

12.1.16.5.19 TPropertyPage.PrintScale Property

Represents the proportions of a printed form.

Pascal

```
property PrintScale: TPrintScale;
```

C++

```
__property TPrintScale PrintScale;
```

Description

Use PrintScale to get or set the proportions of the printed form. PrintScale can have one of the following TPrintScale values:

Value	Meaning
poNone	No special scaling occurs; therefore, the printed form and how the form appears onscreen may appear squished or stretched.
poProportional	The form is printed so that the printed image is approximately the same visible size as on the screen (WYSIWYG). The form image is scaled so that the printed image is approximately the same visible size as on the screen.
poPrintToFit	The form is printed using the same screen proportions, but in a size that just fits the printed page.

See Also

Print (see page 13883)

12.1.16.5.20 TPropertyPage.Scaled Property

Specifies whether the form is sized according to the value of the PixelsPerInch property.

Pascal

```
property Scaled: Boolean;
```

C++

```
__property Boolean Scaled;
```

Description

Scaled determines whether the form adjusts itself to differences between the font used at design time and the current system font used by a system at runtime. If Scaled is true, the form resizes itself and all its child controls to maintain the relationship between the size of controls and the height of text displayed in the default font. If Scaled is false, no such resizing occurs.

By setting Scaled to true, the form can adjust to fonts that are not scalable (that is, fonts that only supply discrete point sizes) or to differences such as forms designed using small fonts at design time that are run on systems that use large fonts.

The degree of resizing is determined by the PixelsPerInch property, which measures the proportion of the font on the system used when designing the form.

See Also

PixelsPerInch (see page 13928)

Considerations When Dynamically Resizing Forms and Controls

12.1.16.5.21 TPropertyPage.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;
```

```
    char string[10];
    char index[3];
    for(int i = 1; i <= 10; i++)
    {
      strcpy(string, "Item");
      itoa(i, index, 10);
      strcat(string, index);
      ListBox1->Items->Add(string);
    }
    ListBox1->Hint = "Select an item";
    ListBox1->ShowHint = true;
    Application->ShowHint = true;
  }
  /*
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  */
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

12.1.16.5.22 TPropertyPage.VertScrollBar Property

Represents the vertical scroll bar for the scrolling windowed control.

Pascal

```
property VertScrollBar: TControlScrollBar;
```

C++

```
__property TControlScrollBar VertScrollBar;
```

Description

Use VertScrollBar to hide, show, or manipulate the vertical scroll bar for the scrolling windowed control.

C++ Examples:

```
/*
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  const int iPageDelta = 10;

  if (Key == VK_NEXT)
    VertScrollBar->Position += iPageDelta;
  else if (Key == VK_PRIOR)
    VertScrollBar->Position -= iPageDelta;
}
```

Delphi Examples:

```
{
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
}
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
const
  PageDelta = 10;
begin
  With VertScrollbar do
    if Key = VK_NEXT then
      Position := Position + PageDelta
    else if Key = VK_PRIOR then
      Position := Position - PageDelta;
end;
```

See Also

AutoScroll (see page 14329)

HorzScrollBar (see page 14329)

TControlScrollBar (see page 13744)

12.1.16.5.23 TPropertyPage.Visible Property

Indicates whether the form is visible.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use Visible to determine whether the form is visible.

If Visible is true, the form is visible unless it is completely obscured by other forms. If Visible is false, the form is not visible.

The Show and ShowModal methods set Visible to true and bring the form to the front of all open windows.

C++ Examples:

```
/*
The following code uses two forms. Form1 has a button on it. The
second form is used as a tool palette. This code makes the palette
form visible, and ensures it is the top form by bringing it to the
front.  To run this example, you must include Unit2.h in this
module.
*/
#include "Unit2.h"

void __fastcall TForm1::ShowPaletteButtonClick(TObject *Sender)
{
  if (!Form2->Visible)
  {
    Form2->Visible = true;
    Form2->BringToFront();
  }
}
```

Delphi Examples:

```
{
The following code uses two forms. Form1 has a button on it. The
second form is used as a tool palette. This code makes the palette
form visible, and ensures it is the top form by bringing it to the
front.  To run this example, you must add Unit2 in the uses clause
of your unit.
}
procedure TForm1.ShowPaletteButtonClick(Sender: TObject);
begin
if Form2.Visible = False then Form2.Visible := True;
Form2.BringToFront;
end;
```

See Also

BringToFront (see page 5740)

Show (see page 13888)

ShowModal (see page 13890)

12.1.16.5.24 TPropertyPage.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

See Also

ClientWidth (see page 5811)

Height (see page 5825)

Left (see page 5830)

12.1.16.6 TPropertyPage Events

The events of the TPropertyPage class are listed here.

Published Events

	Name	Description
	OnActivate (see page 1646)	Occurs when the form becomes active.
	OnClick (see page 1650)	Occurs when the user clicks the control.
	OnClose (see page 1652)	Occurs when the form closes.
	OnContextPopup (see page 1653)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 1654)	Occurs when the form is created.
	OnDblClick (see page 1655)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 1656)	Occurs when the form loses focus.
	OnDestroy (see page 1656)	Occurs when the form is destroyed.
	OnDragDrop (see page 1657)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 1659)	Occurs when the user drags an object over a control.
	OnHide (see page 1662)	Occurs when the form is hidden (that is, when its Visible property is set to false).
	OnKeyDown (see page 1664)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 1667)	Occurs when a key is pressed.
	OnKeyUp (see page 1668)	Occurs when the user releases a key that has been pressed.
	OnMouseDown (see page 1669)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 1672)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 1674)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 1676)	Occurs when the form is redrawn.
	OnResize (see page 1677)	Occurs immediately after the control is resized.
	OnShow (see page 1678)	Occurs when the form is shown (that is, when its Visible property is set to true).

12.1.16.6.1 TPropertyPage.OnActivate Event

Occurs when the form becomes active.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Use OnActivate to perform special processing when the form receives focus. A form becomes active when focus is transferred to it (when the user clicks on the form, for example).

Note: The OnActivate event of the application (TApplication), not the form, occurs when Windows switches control from another application.

Delphi Examples:

```
{
Here is the code on Form1 that fills the list box.
```

```
}
var
  Form1: TForm1;
  Customers: TTable;
  Blobs: TTable;
  Form1Done : Boolean;
  procedure CreateMyTable;
  procedure CreateCustSource;
  procedure CreateBlobSource;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  CreateCustSource;
  CreateBlobSource;
  for I := 0 to ComponentCount - 1 do
  begin
    if (Components[I] is TDataSource) then
      ListBox1.Items.AddObject(Components[I].Name,
        Components[I] as TObject);
  end;
end;

procedure CreateCustSource();
begin
  Customers:= TTable.Create(Form1);
  with Customers do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'MyCustInfo';
    ReadOnly:= False;

//    Don't overwrite an existing table
    if (not Customers.Exists) then CreateMyTable
    else
    begin
      if (Customers.Exists AND
        (MessageDlg('MyCustInfo table already exists.  Do you want to rebuild it?',
mtConfirmation, [mbYes, mbNo], 0) = mrYes)) then
      begin
        Customers.Close;
        Customers.DeleteTable;
        CreateMyTable;
      end;
    end;
  end;
  Form1.CustomerDS.DataSet:= Customers;
  Customers.Active:= True;
end;

procedure CreateMyTable();
var
  i: Integer;
begin
//      The Table component must not be active
//      First, describe the type of table and give
//      it a name
//      Next, describe the fields in the table
  with Customers do
  begin
```

```
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
          with AddFieldDef do
          begin
            Name := 'Field3';
            DataType := ftString;
            Size := 30;
          end;
          with AddFieldDef do
          begin
            Name := 'Field4';
            DataType := ftString;
            Size := 30;
          end;
        end;

//      Take out this part to remove indexing
//      Next, describe any indexes
        with IndexDefs do
        begin
          Clear;
//          The 1st index has no name because it is
//          a Paradox primary key
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;

//        Call the CreateTable method to create the table
        CreateTable;
        Customers.Active:= True;
        for i := 1 to 20 do
          Customers.AppendRecord([i*111, i*222, i*333, i*444]);
      end;
  end;

  procedure CreateBlobSource;
  var
    i: Integer;
  begin
    Blobs:= TTable.Create(Form1);
    with Blobs do
    begin
      DatabaseName := 'DBDEMOS';
```

```
      TableType := ttParadox;
      TableName := 'MyBlobInfo';
      Blobs.Active := False;
      { Don't overwrite an existing table }
//    if Blobs.Exists then
//      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
//    else
      begin
        if (Blobs.Exists) then
        begin
          Blobs.Close;
          Blobs.DeleteTable;
        end;
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
        { Next, describe the fields in the table }
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftBlob;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
{
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
}
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Blobs.Active:= True;
        for i := 1 to 20 do
          Blobs.AppendRecord([i*111, IntToStr(i*222)]);
      end;
    end;
    Form1.BlobDS.DataSet:= Blobs;
    Blobs.Active:= True;
  end;

  procedure TForm1.ListBox1MouseDown(Sender: TObject; Button: TMouseButton;
```

```
    Shift: TShiftState; X, Y: Integer);
  begin
    Form1Done := True;
  end;

  initialization

  Form1Done := False;
  {
  The following code sets the DataSource for a DBGrid
  component in Form2 to the currently selected data source in
  a list provided in Form1. This allows users to specify the
  desired information in Form1, and then move to Form2 to view
  the information. Select an item in the listbox and then
  activate Form2 by clicking in it.
  }
  procedure TForm2.FormActivate(Sender: TObject);
  begin
    if (Form1Done) then
    begin
      DBGrid2.DataSource :=
        Form1.ListBox1.Items.Objects[Form1.ListBox1.ItemIndex] as TDataSource;
      DBNavigator2.DataSource :=
        Form1.ListBox1.Items.Objects[Form1.ListBox1.ItemIndex] as TDataSource;
    end;
  end;
```

See Also

FormStyle (see page 13912)

OnDeactivate (see page 13953)

OnActivate (see page 13726)

12.1.16.6.2 TPropertyPage.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.

- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
```

```
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

12.1.16.6.3 TPropertyPage.OnClose Event

Occurs when the form closes.

Pascal

```
property OnClose: TCloseEvent;
```

C++

```
__property TCloseEvent OnClose;
```

Description

Use OnClose to perform special processing when the form closes. The OnClose event specifies which event handler to call when a form is about to close. The handler specified by OnClose might, for example, test to make sure all fields in a data-entry form have valid contents before allowing the form to close.

A form is closed by the Close method or when the user chooses Close from the form's system menu.

The TCloseEvent type points to a method that handles the closing of a form. The value of the Action parameter determines if the form actually closes. These are the possible values of Action:

Value	Meaning
caNone	The form is not allowed to close, so nothing happens.
caHide	The form is not closed, but just hidden. Your application can still access a hidden form.
caFree	The form is closed and all allocated memory for the form is freed.
caMinimize	The form is minimized, rather than closed. This is the default action for MDI child forms.

If a form is an MDI child form, and its BorderIcons property is biMinimize, then the default Action is caMinimize. If a MDI child form does not have these settings, the default Action is caNone, meaning that nothing happens when the user attempts to close the form.

If a form is an SDI child form, Action defaults to caHide.

To close the form and free it in an OnClose event, set Action to caFree.

Note: When the application shuts down, the main form receives an OnClose event, but any child forms do not receive the OnClose event.

C++ Examples:

```
/*
This example displays a message dialog box when the user
attempts to close the form. If the user clicks the Yes
button, the form closes; otherwise, the form only minimizes.
TCustomForm.Close does not allow an action of caMinimize
for a form closing itself.  Close Form2 from Form1 (from
a button?) and implement the OnClose in Form2.  Place
"Application.CreateForm(TForm2, Form2);" in the project dpr
file to open Form2 and set the Form2 Visible property to
True.
*/
void __fastcall TForm2::FormClose(TObject *Sender, TCloseAction &Action)
{
  if (MessageDlg("Close application ?", mtConfirmation, TMsgDlgButtons() << mbYes << mbNo,0)
== mrYes)
    Action = caFree;
  else
    Action = caMinimize;
}
```

See Also

OnCreate (see page 13951)

OnCloseQuery (see page 13950)

OnDestroy (see page 13953)

12.1.16.6.4 TPropertyPage.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the

default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

12.1.16.6.5 TPropertyPage.OnCreate Event

Occurs when the form is created.

Pascal

```
property OnCreate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCreate;
```

Description

Use OnCreate to perform special processing when the form is created and is invoked by TCustomForm's constructor. Either implement this event or override the constructor of the form; do not do both. Any objects created in the OnCreate event should be freed by the OnDestroy event.

When a form is being created and its Visible property is true, the following events occur in the order listed:

1. OnCreate2. OnShow3. OnActivate4. OnPaint

Note: Use of the OnCreate event is discouraged in C++ code because it can interact badly with the form's constructor (see OldCreateOrder). It is recommended that you override the form constructor instead.

C++ Examples:

```
/*
This example requires a new TPageControl, as-is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
evaluates as true.  If Button is not btNext, the previous Tab Page
is selected.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
const
  TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *page = new TTabSheet(PageControl1);
    page->PageControl = PageControl1;
    page->Caption = AnsiString("Page") + IntToStr(i);
    page->Brush->Color = colorPalette[i];
  }
```

```
}

void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  PageControl1->SelectNextPage(Button == btNext);
}
```

Delphi Examples:

```
{
This example requires a new TPageControl, as-is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
evaluates as true.  If Button is not btNext, the previous Tab Page
is selected.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
const
  colorPalette: Array[0..11] of TColor =
    (clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet #' + IntToStr(i);
      Brush.Color := colorPalette[i];
    end;
end;

procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
begin
  PageControl1.SelectNextPage(Button = btNext);
end;
```

See Also

OnDestroy (see page 13953)

OnActivate (see page 13943)

OnPaint (see page 13975)

OnShow (see page 13977)

OldCreateOrder (see page 13926)

12.1.16.6.6 TPropertyPage.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

12.1.16.6.7 TPropertyPage.OnDeactivate Event

Occurs when the form loses focus.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Use OnDeactivate to perform special processing when the form transitions from being the active form to another form in the same application becoming the active form. If activation goes to another application, this event is not triggered. To determine if another application has become active, Use the TApplication object's OnDeactivate event.

See Also

OnActivate (see page 13943)

OnDeactivate (see page 13727)

12.1.16.6.8 TPropertyPage.OnDestroy Event

Occurs when the form is destroyed.

Pascal

```
property OnDestroy: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDestroy;
```

Description

Use OnDestroy to perform special processing when the form is destroyed. Either implement this event or override the destructor of the class, but not both. This event should destroy any objects created in the OnCreate event.

C++ Examples:

```
/*
The following code allocates a TBitmap object that is used
by the application when the main form is first created. The
OnDestroy method frees the bitmap as the application shuts
down.
*/
Graphics::TBitmap *LogoBmp;
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  LogoBmp = new Graphics::TBitmap();
  LogoBmp->LoadFromFile("../FACTORY.BMP");
}
```

```
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete LogoBmp;
}

void __fastcall TForm1::FormPaint(TObject *Sender)
{
  Canvas->Brush->Bitmap = LogoBmp;
  Canvas->FillRect(Rect(0,0,100,100));
}
```

Delphi Examples:

```
{
The following code explicitly allocates memory for a pointer
in the OnCreate event of Form1, then releases the memory in
the OnDestroy event. Assume that MyPtr is a Pointer type
field of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  New(MyPtr);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Dispose(MyPtr);
end;
```

See Also

OnCreate (see page 13951)

Free (see page 22300)

Destroy (see page 13862)

OldCreateOrder (see page 13926)

12.1.16.6.9 TPropertyPage.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/
```

```
// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

12.1.16.6.10 TPropertyPage.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

12.1.16.6.11 TPropertyPage.OnHide Event

Occurs when the form is hidden (that is, when its Visible property is set to false).

Pascal

```
property OnHide: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnHide;
```

Description

Use OnHide to perform special processing when the form is hidden (that is, when the form's Visible property is set to false).

C++ Examples:

```
/*
This example uses two forms, each with a label and a button.
When the user clicks a button, the other form appears and
the current form disappears. Also, a message appears in the
label of the form that is showing, indicating that the other
form is hidden. Set the OnHide event handle for each form to
FormHide.This is the implementation section of Unit1:
*/
#include "TFormOnHide.h"
#include "Unit2.h"  // We must include unit2's header file too!

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
  Hide();
}

void __fastcall TForm1::FormHide(TObject *Sender)
{
  Form2->Label1->Caption = "Form1 is hiding";
```

```
}
/*
This is the implementation of Unit2:
*/
#include "Unit2.h"
#include "TFormOnHide.h"  // We must include unit1's header file too!

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm2 *Form2;

__fastcall TForm2::TForm2(TComponent* Owner)
    : TForm(Owner)
{
}

void __fastcall TForm2::Button1Click(TObject *Sender)
{
  Form1->Show();
  Hide();
}

void __fastcall TForm2::FormHide(TObject *Sender)
{
  Form1->Label1->Caption = "Form2 is hiding";
}
```

Delphi Examples:

```
{
This example uses two forms, each with a label and a button.
When the user clicks a button, the other form appears and
the current form disappears. Also, a message appears in the
label of the form that is showing, indicating that the other
form is hidden. Set the OnHide event handle for each form to
FormHide.This is the implementation section of Unit1:
}

implementation

{$R *.dfm}

uses Unit2;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
  Hide;
end;

procedure TForm1.FormHide(Sender: TObject);
begin
  Form2.Label1.Caption := 'Form1 is hiding';
end;
{
This is the implementation section of Unit2:
}
implementation

{$R *.dfm}

uses TFormOnHide;

procedure TForm2.Button1Click(Sender: TObject);
begin
  Form1.Show;
  Hide;
```

```
end;

procedure TForm2.FormHide(Sender: TObject);
begin
  Form1.Label1.Caption := 'Form2 is hiding';
end;
```

See Also

OnShow (see page 13977)

12.1.16.6.12 TPropertyPage.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
```

```
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
```

```
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

12.1.16.6.13 TPropertyPage.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

12.1.16.6.14 TPropertyPage.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

12.1.16.6.15 TPropertyPage.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

12.1.16.6.16 TPropertyPage.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
```

```
      }
      else
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
```

```
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

12.1.16.6.17 TPropertyPage.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;
```

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

12.1.16.6.18 TPropertyPage.OnPaint Event

Occurs when the form is redrawn.

Pascal

```
property OnPaint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPaint;
```

Description

Use OnPaint to perform special processing when the form is redrawn. Any special painting on the form should be done in this event. OnPaint occurs before any controls on the form are painted.

To determine which portions of the form's canvas need to be repainted, use the ClipRect property of the canvas.

If you use the form's Canvas property outside the OnPaint event, it will be erased and drawn-over by the next OnPaint event.

C++ Examples:

```
/*
The following code loads a background bitmap onto the Canvas
of the main form in the OnPaint event handler.
*/
Graphics::TBitmap *TheGraphic;

void __fastcall TForm1::FormPaint(TObject *Sender)// OnPaint event handler
{
  Form1->Canvas->Draw(0, 0, TheGraphic); // Draw the graphic on the Canvas
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TheGraphic = new Graphics::TBitmap(); // Create the bitmap object
  TheGraphic->LoadFromFile("..\\bigf.bmp"); // Load the bitmap from a file
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete TheGraphic;
}
```

Delphi Examples:

```
{
The following code loads a background bitmap onto the Canvas
of the main form in the OnPaint event handler.
}
procedure TForm1.FormPaint(Sender: TObject); { OnPaint event handler}
begin
  Form1.Canvas.Draw(0, 0, TheGraphic); { Draw the graphic on the Canvas }
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  TheGraphic := TBitmap.Create; { Create the bitmap object }
  TheGraphic.LoadFromFile('bigf.bmp'); { Load the bitmap from a file}
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  (TheGraphic as TObject).Free;
end;
```

See Also

Canvas (see page 13900)

ClipRect (see page 14486)

12.1.16.6.19 TPropertyPage.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
```

```
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

12.1.16.6.20 TPropertyPage.OnShow Event

Occurs when the form is shown (that is, when its Visible property is set to true).

Pascal

```
property OnShow: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnShow;
```

Description

Use OnShow to perform special processing when the form is shown (that is, when the form's Visible property is set to true).

C++ Examples:

```
/*
This project requires a Page Control.  You will need to
populate the Page Control by right clicking and selecting
New Page from the context menu. As the tabs are selected by
the user, the Page Control OnChange event fires, which
causes the form caption to display the captions of each of
the TTabSheets attached to the Page Control.
Note the use of the Form’s OnShow event handler to set the
caption when the form first appears.  This occurs before
any changes trigger the OnChange event of the Page Control.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  Caption = AnsiString("Now working on tab: " + PageControl1->ActivePage->Caption);
}

void __fastcall TForm1::FormShow(TObject *Sender)
{
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This project requires a Page Control.  You will need to
populate the Page Control by right clicking and selecting
New Page from the context menu. As the tabs are selected by
the user, the Page Control OnChange event fires, which
causes the form caption to display the captions of each of
the TTabSheets attached to the Page Control.
Note the use of the Form’s OnShow event handler to set the
caption when the form first appears.  This occurs before
any changes trigger the OnChange event of the Page Control.
}
```

```
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  Caption := ' Now working on tab:  ' + PageControl1.ActivePage.Caption;
end;

procedure TForm1.FormShow(Sender: TObject);
begin
  PageControl1Change(Sender);
end;
```

See Also

OnHide (see page 13960)

12.1.17 TPropertyPageImpl Class

TPropertyPageImpl implements the IPropertyPage interface to provide the interfaces needed to support an ActiveX property page.

Class Hierarchy

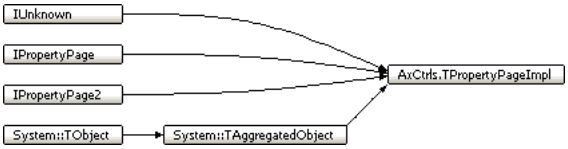

Pascal

```
TPropertyPageImpl = class(TAggregatedObject, IUnknown, IPropertyPage, IPropertyPage2);
```

C++

```
class TPropertyPageImpl : public TAggregatedObject, public IUnknown, public IPropertyPage,
public IPropertyPage2;
```

File

AxCtrls

Description

Applications do not create TPropertyPageImpl objects; they are created automatically when the user creates an ActiveX control object by using the ActiveX control or ActiveForms wizard.

This class is used internally by TPropertyPage (see page 1603) to provide a bridge between a Delphi form and the ActiveX property page API. TPropertyPageImpl implements the property page portion of the COM object while TActiveXPropertyPage (see page 1561) implements the COM server portion.

12.1.17.1 TPropertyPageImpl Members

The following tables list the members exposed by TPropertyPageImpl.

Protected Methods

	Name	Description
	_AddRef (see page 22250)	This is _AddRef, a member of class TAggregatedObject.
	_Release (see page 22250)	This is _Release, a member of class TAggregatedObject.
	Activate (see page 1681)	This is Activate, a member of class TPropertyPageImpl.

	Apply (see page 1681)	This is Apply, a member of class TPropertyPageImpl.
	Deactivate (see page 1681)	This is Deactivate, a member of class TPropertyPageImpl.
	EditProperty (see page 1682)	This is EditProperty, a member of class TPropertyPageImpl.
	GetPageInfo (see page 1682)	This is GetPageInfo, a member of class TPropertyPageImpl.
	Help (see page 1682)	This is Help, a member of class TPropertyPageImpl.
	IsPageDirty (see page 1683)	This is IsPageDirty, a member of class TPropertyPageImpl.
	Move (see page 1683)	This is Move, a member of class TPropertyPageImpl.
	QueryInterface (see page 22250)	This is QueryInterface, a member of class TAggregatedObject.
	SetObjects (see page 1683)	This is SetObjects, a member of class TPropertyPageImpl.
	SetPageSite (see page 1683)	This is SetPageSite, a member of class TPropertyPageImpl.
	Show (see page 1684)	This is Show, a member of class TPropertyPageImpl.
	TranslateAccelerator (see page 1684)	This is TranslateAccelerator, a member of class TPropertyPageImpl.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22249)	Instantiates an aggregated object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	InitPropertyPage (see page 1682)	Initializes the associated property page.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Controller (see page 22251)	Provides access to the controlling interface of the aggregate.
	PropertyPage (see page 1684)	This is PropertyPage, a member of class TPropertyPageImpl.

12.1.17.2 TPropertyPageImpl Methods

The methods of the TPropertyPageImpl class are listed here.

Protected Methods

	Name	Description
	Activate (see page 1681)	This is Activate, a member of class TPropertyPageImpl.
	Apply (see page 1681)	This is Apply, a member of class TPropertyPageImpl.
	Deactivate (see page 1681)	This is Deactivate, a member of class TPropertyPageImpl.
	EditProperty (see page 1682)	This is EditProperty, a member of class TPropertyPageImpl.
	GetPageInfo (see page 1682)	This is GetPageInfo, a member of class TPropertyPageImpl.
	Help (see page 1682)	This is Help, a member of class TPropertyPageImpl.
	IsPageDirty (see page 1683)	This is IsPageDirty, a member of class TPropertyPageImpl.
	Move (see page 1683)	This is Move, a member of class TPropertyPageImpl.
	SetObjects (see page 1683)	This is SetObjects, a member of class TPropertyPageImpl.
	SetPageSite (see page 1683)	This is SetPageSite, a member of class TPropertyPageImpl.
	Show (see page 1684)	This is Show, a member of class TPropertyPageImpl.
	TranslateAccelerator (see page 1684)	This is TranslateAccelerator, a member of class TPropertyPageImpl.

Public Methods

	Name	Description
	InitPropertyPage (see page 1682)	Initializes the associated property page.

12.1.17.2.1 TPropertyPageImpl.Activate Method

Pascal

```
function Activate(hwndParent: HWnd; const rc: TRect; bModal: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Activate(HWnd hwndParent, const TRect rc, BOOL bModal);
```

Description

This is Activate, a member of class TPropertyPageImpl.

12.1.17.2.2 TPropertyPageImpl.Apply Method

Pascal

```
function Apply: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Apply();
```

Description

This is Apply, a member of class TPropertyPageImpl.

12.1.17.2.3 TPropertyPageImpl.Deactivate Method

Pascal

```
function Deactivate: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Deactivate();
```

Description

This is Deactivate, a member of class TPropertyPageImpl.

12.1.17.2.4 TPropertyPageImpl.EditProperty Method

Pascal

```
function EditProperty(dispid: TDispID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT EditProperty(TDispID dispid);
```

Description

This is EditProperty, a member of class TPropertyPageImpl.

12.1.17.2.5 TPropertyPageImpl.GetPageInfo Method

Pascal

```
function GetPageInfo(out pageInfo: TPropPageInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetPageInfo(TPropPageInfo pageInfo);
```

Description

This is GetPageInfo, a member of class TPropertyPageImpl.

12.1.17.2.6 TPropertyPageImpl.Help Method

Pascal

```
function Help(pszHelpDir: POleStr): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Help(POleStr pszHelpDir);
```

Description

This is Help, a member of class TPropertyPageImpl.

12.1.17.2.7 TPropertyPageImpl.InitPropertyPage Method

Initializes the associated property page.

Pascal

```
procedure InitPropertyPage; virtual;
```

C++

```
virtual __fastcall InitPropertyPage();
```

Description

Do not call InitPropertyPage, TActiveXPropertyPage (see page 1561) calls this method after creating both the TPropertyPage (see page 1603) and TPropertyPageImpl (see page 1679) objects. It sets the TPropertyPageImpl (see page 1679)'s

PropertyPage (see page 1684) property, and then calls InitPropertyPage. InitPropertyPage sets an internal field of TPropertyPage (see page 1603) so that the TPropertyPage (see page 1603) can locate its associated TPropertyPageImpl (see page 1679). In addition, InitPropertyPage initializes some properties on the property page.

See Also

TPropertyPage (see page 1603)

TActiveXPropertyPage (see page 1561)

PropertyPage

12.1.17.2.8 TPropertyPageImpl.IsPageDirty Method

Pascal

```
function IsPageDirty: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT IsPageDirty();
```

Description

This is IsPageDirty, a member of class TPropertyPageImpl.

12.1.17.2.9 TPropertyPageImpl.Move Method

Pascal

```
function Move(const rect: TRect): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Move(const TRect rect);
```

Description

This is Move, a member of class TPropertyPageImpl.

12.1.17.2.10 TPropertyPageImpl.SetObjects Method

Pascal

```
function SetObjects(cObjects: Longint; pUnkList: PUnknownList): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetObjects(Longint cObjects, PUnknownList pUnkList);
```

Description

This is SetObjects, a member of class TPropertyPageImpl.

12.1.17.2.11 TPropertyPageImpl.SetPageSite Method

Pascal

```
function SetPageSite(const pageSite: IPropertyPageSite): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetPageSite(const IPropertyPageSite pageSite);
```

Description

This is SetPageSite, a member of class TPropertyPageImpl.

12.1.17.2.12 TPropertyPageImpl.Show Method

Pascal

```
function Show(nCmdShow: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Show(int nCmdShow);
```

Description

This is Show, a member of class TPropertyPageImpl.

12.1.17.2.13 TPropertyPageImpl.TranslateAccelerator Method

Pascal

```
function TranslateAccelerator(msg: PMsg): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT TranslateAccelerator(PMsg msg);
```

Description

This is TranslateAccelerator, a member of class TPropertyPageImpl.

12.1.17.3 TPropertyPageImpl Properties

The properties of the TPropertyPageImpl class are listed here.

Public Properties

	Name	Description
	PropertyPage (see page 1684)	This is PropertyPage, a member of class TPropertyPageImpl.

12.1.17.3.1 TPropertyPageImpl.PropertyPage Property

Pascal

```
property PropertyPage: TPropertyPage;
```

C++

```
__property TPropertyPage PropertyPage;
```

Description

This is PropertyPage, a member of class TPropertyPageImpl.

12.1.18 TReflectorWindow Class

TReflectorWindow ensures that ActiveX controls respond properly to Windows messages.

Class Hierarchy

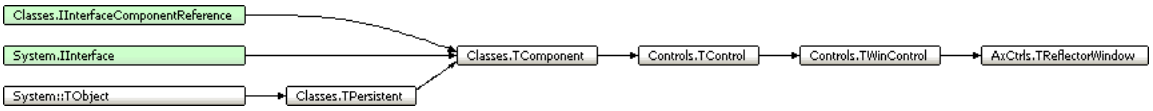

Pascal

```
TReflectorWindow = class(TWinControl);
```

C++

```
class TReflectorWindow : public TWinControl;
```

File

AxCtrls

Description

TReflectorWindow is for internal use only. Applications should never instantiate this class. It is used to reflect certain messages to an ActiveX control.

12.1.18.1 TReflectorWindow Members

The following tables list the members exposed by TReflectorWindow.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.

	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.

	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 1695)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.

	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.

	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.

	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).

	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 1695)	Creates an instance of TReflectorWindow (see page 1684).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.

	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.

	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.

	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.

	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

12.1.18.2 TReflectorWindow.Create Constructor

Creates an instance of TReflectorWindow (see page 1684).

Pascal

```
constructor Create(ParentWindow: HWND; Control: TControl); reintroduce;
```

C++

```
__fastcall TReflectorWindow(HWND ParentWindow, TControl * Control);
```

Description

Applications never have to instantiate TReflectorWindow (see page 1684).

See Also

TActiveXControl (see page 1517)

12.1.18.3 TReflectorWindow Methods

The methods of the TReflectorWindow class are listed here.

Protected Methods

	Name	Description
V	CreateParams (see page 1695)	Initializes a window-creation parameter data structure.

12.1.18.3.1 TReflectorWindow.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

12.1.19 TStringsAdapter Class

TStringsAdapter is a helper class used to allow a TStrings object to implement an IStrings interface for use with OLE.

Class Hierarchy

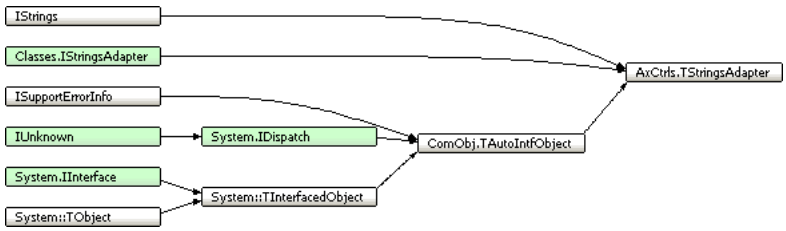

Pascal

```
TStringsAdapter = class(TAutoIntfObject, IStrings, IStringsAdapter);
```

C++

```
class TStringsAdapter : public TAutoIntfObject, public IStrings, public IStringsAdapter;
```

File

AxCtrls

Description

Use a strings adapter to adapt a TStrings object so that it supports the IStrings interface. To do so, create the strings adapter by calling GetOleStrings (see page 1711), passing the TStrings object as a parameter. Once the strings adapter is instantiated for a TStrings object, the StringsAdapter property of the TStrings object can be used to interact with the TStrings object through an IStrings interface.

See Also

TStrings (see page 3029)

GetOleStrings (see page 1711)

12.1.19.1 TStringsAdapter Members

The following tables list the members exposed by TStringsAdapter.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_NewEnum (see page 1699)	This is _NewEnum, a member of class TStringsAdapter.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	Add (see page 1699)	This is Add, a member of class TStringsAdapter.
	Clear (see page 1699)	This is Clear, a member of class TStringsAdapter.
	Count (see page 1700)	This is Count, a member of class TStringsAdapter.
	Get_ControlDefault (see page 1700)	This is Get_ControlDefault, a member of class TStringsAdapter.
	Get_Item (see page 1700)	This is Get_Item, a member of class TStringsAdapter.
	GetIDsOfNames (see page 5537)	This is GetIDsOfNames, a member of class TAutoIntfObject.
	GetTypeInfo (see page 5537)	This is GetTypeInfo, a member of class TAutoIntfObject.
	GetTypeInfoCount (see page 5537)	This is GetTypeInfoCount, a member of class TAutoIntfObject.
	InterfaceSupportsErrorInfo (see page 5538)	This is InterfaceSupportsErrorInfo, a member of class TAutoIntfObject.
	Invoke (see page 5538)	This is Invoke, a member of class TAutoIntfObject.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	ReferenceStrings (see page 1700)	This is ReferenceStrings, a member of class TStringsAdapter.
	ReleaseStrings (see page 1700)	This is ReleaseStrings, a member of class TStringsAdapter.
	Remove (see page 1701)	This is Remove, a member of class TStringsAdapter.
	Set_ControlDefault (see page 1701)	This is Set_ControlDefault, a member of class TStringsAdapter.
	Set_Item (see page 1701)	This is Set_Item, a member of class TStringsAdapter.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

	Create (see page 1698)	Creates an instance of the string adapter control.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 5538)	SafeCallException is used for OLE exception handling.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	DispIID (see page 5539)	DispIID is the dispatch identifier.
R	DispIntfEntry (see page 5539)	DispIntfEntry contains the entry for the dispatch interface implemented by the automation object.
R	DispTypeInfo (see page 5539)	DispTypeInfo returns an ITypeInfo interface reference to the type information for the dispinterface implemented by the automation object.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

12.1.19.2 TStringsAdapter.Create Constructor

Creates an instance of the string adapter control.

Pascal

```
constructor Create(Strings: TStrings);
```

C++

```
__fastcall TStringsAdapter(TStrings Strings);
```

Description

Do not call the Create method directly. Instead, call the GetOleStrings (see page 1711) procedure to adapt a TStrings object so that it supports the IStrings interface.

See Also

GetOleStrings

12.1.19.3 TStringsAdapter Methods

The methods of the TStringsAdapter class are listed here.

Protected Methods

	Name	Description
	_NewEnum (see page 1699)	This is _NewEnum, a member of class TStringsAdapter.
	Add (see page 1699)	This is Add, a member of class TStringsAdapter.
	Clear (see page 1699)	This is Clear, a member of class TStringsAdapter.
	Count (see page 1700)	This is Count, a member of class TStringsAdapter.
	Get_ControlDefault (see page 1700)	This is Get_ControlDefault, a member of class TStringsAdapter.
	Get_Item (see page 1700)	This is Get_Item, a member of class TStringsAdapter.
	ReferenceStrings (see page 1700)	This is ReferenceStrings, a member of class TStringsAdapter.
	ReleaseStrings (see page 1700)	This is ReleaseStrings, a member of class TStringsAdapter.
	Remove (see page 1701)	This is Remove, a member of class TStringsAdapter.
	Set_ControlDefault (see page 1701)	This is Set_ControlDefault, a member of class TStringsAdapter.
	Set_Item (see page 1701)	This is Set_Item, a member of class TStringsAdapter.

12.1.19.3.1 TStringsAdapter._NewEnum Method

Pascal

```
function _NewEnum: IUnknown; safecall;
```

C++

```
__fastcall IUnknown _NewEnum();
```

Description

This is _NewEnum, a member of class TStringsAdapter.

12.1.19.3.2 TStringsAdapter.Add Method

Pascal

```
function Add(Item: OleVariant): Integer; safecall;
```

C++

```
__fastcall int Add(OleVariant Item);
```

Description

This is Add, a member of class TStringsAdapter.

12.1.19.3.3 TStringsAdapter.Clear Method

Pascal

```
procedure Clear; safecall;
```

C++

```
__fastcall Clear();
```

Description

This is Clear, a member of class TStringsAdapter.

12.1.19.3.4 TStringsAdapter.Count Method

Pascal

```
function Count: Integer; safecall;
```

C++

```
__fastcall int Count();
```

Description

This is Count, a member of class TStringsAdapter.

12.1.19.3.5 TStringsAdapter.Get_ControlDefault Method

Pascal

```
function Get_ControlDefault(Index: Integer): OleVariant; safecall;
```

C++

```
__fastcall OleVariant Get_ControlDefault(int Index);
```

Description

This is Get_ControlDefault, a member of class TStringsAdapter.

12.1.19.3.6 TStringsAdapter.Get_Item Method

Pascal

```
function Get_Item(Index: Integer): OleVariant; safecall;
```

C++

```
__fastcall OleVariant Get_Item(int Index);
```

Description

This is Get_Item, a member of class TStringsAdapter.

12.1.19.3.7 TStringsAdapter.ReferenceStrings Method

Pascal

```
procedure ReferenceStrings(S: TStrings);
```

C++

```
__fastcall ReferenceStrings(TStrings S);
```

Description

This is ReferenceStrings, a member of class TStringsAdapter.

12.1.19.3.8 TStringsAdapter.ReleaseStrings Method

Pascal

```
procedure ReleaseStrings;
```

C++

```
__fastcall ReleaseStrings();
```

Description

This is ReleaseStrings, a member of class TStringsAdapter.

12.1.19.3.9 TStringsAdapter.Remove Method

Pascal

```
procedure Remove(Index: Integer); safecall;
```

C++

```
__fastcall Remove(int Index);
```

Description

This is Remove, a member of class TStringsAdapter.

12.1.19.3.10 TStringsAdapter.Set_ControlDefault Method

Pascal

```
procedure Set_ControlDefault(Index: Integer; Value: OleVariant); safecall;
```

C++

```
__fastcall Set_ControlDefault(int Index, OleVariant Value);
```

Description

This is Set_ControlDefault, a member of class TStringsAdapter.

12.1.19.3.11 TStringsAdapter.Set_Item Method

Pascal

```
procedure Set_Item(Index: Integer; Value: OleVariant); safecall;
```

C++

```
__fastcall Set_Item(int Index, OleVariant Value);
```

Description

This is Set_Item, a member of class TStringsAdapter.

12.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAmbientDispatch (see page 1702)	This is class AxCtrls.IAmbientDispatch.
	IFontAccess (see page 1706)	This is class AxCtrls.IFontAccess.
	IPictureAccess (see page 1707)	This is class AxCtrls.IPictureAccess.

12.2.1 IAmbientDispatch Interface

Class Hierarchy

AxCtrls.IAmbientDispatch

Pascal

```
IAmbientDispatch = dispinterface;
```

C++

```
class IAmbientDispatch;
```

File

AxCtrls

Description

This is class AxCtrls.IAmbientDispatch.

12.2.1.1 IAmbientDispatch Members

The following tables list the members exposed by IAmbientDispatch.

Public Properties

	Name	Description
	AutoClip (see page 1703)	This is AutoClip, a member of class IAmbientDispatch.
	BackColor (see page 1703)	This is BackColor, a member of class IAmbientDispatch.
	DisplayAsDefault (see page 1703)	This is DisplayAsDefault, a member of class IAmbientDispatch.
	DisplayName (see page 1704)	This is DisplayName, a member of class IAmbientDispatch.
	Font (see page 1704)	This is Font, a member of class IAmbientDispatch.
	ForeColor (see page 1704)	This is ForeColor, a member of class IAmbientDispatch.
	LocaleID (see page 1704)	This is LocaleID, a member of class IAmbientDispatch.
	MessageReflect (see page 1705)	This is MessageReflect, a member of class IAmbientDispatch.
	ScaleUnits (see page 1705)	This is ScaleUnits, a member of class IAmbientDispatch.
	ShowGrabHandles (see page 1705)	This is ShowGrabHandles, a member of class IAmbientDispatch.
	ShowHatching (see page 1705)	This is ShowHatching, a member of class IAmbientDispatch.
	SupportsMnemonics (see page 1705)	This is SupportsMnemonics, a member of class IAmbientDispatch.
	TextAlign (see page 1706)	This is TextAlign, a member of class IAmbientDispatch.

	UIDead (see page 1706)	This is UIDead, a member of class IAmbientDispatch.
	UserMode (see page 1706)	This is UserMode, a member of class IAmbientDispatch.

12.2.1.2 IAmbientDispatch Properties

The properties of the IAmbientDispatch class are listed here.

Public Properties

	Name	Description
	AutoClip (see page 1703)	This is AutoClip, a member of class IAmbientDispatch.
	BackColor (see page 1703)	This is BackColor, a member of class IAmbientDispatch.
	DisplayAsDefault (see page 1703)	This is DisplayAsDefault, a member of class IAmbientDispatch.
	DisplayName (see page 1704)	This is DisplayName, a member of class IAmbientDispatch.
	Font (see page 1704)	This is Font, a member of class IAmbientDispatch.
	ForeColor (see page 1704)	This is ForeColor, a member of class IAmbientDispatch.
	LocaleID (see page 1704)	This is LocaleID, a member of class IAmbientDispatch.
	MessageReflect (see page 1705)	This is MessageReflect, a member of class IAmbientDispatch.
	ScaleUnits (see page 1705)	This is ScaleUnits, a member of class IAmbientDispatch.
	ShowGrabHandles (see page 1705)	This is ShowGrabHandles, a member of class IAmbientDispatch.
	ShowHatching (see page 1705)	This is ShowHatching, a member of class IAmbientDispatch.
	SupportsMnemonics (see page 1705)	This is SupportsMnemonics, a member of class IAmbientDispatch.
	TextAlign (see page 1706)	This is TextAlign, a member of class IAmbientDispatch.
	UIDead (see page 1706)	This is UIDead, a member of class IAmbientDispatch.
	UserMode (see page 1706)	This is UserMode, a member of class IAmbientDispatch.

12.2.1.2.1 IAmbientDispatch.AutoClip Property

Pascal

```
property AutoClip: WordBool;
```

C++

```
__property WordBool AutoClip;
```

Description

This is AutoClip, a member of class IAmbientDispatch.

12.2.1.2.2 IAmbientDispatch.BackColor Property

Pascal

```
property BackColor: Integer;
```

C++

```
__property int BackColor;
```

Description

This is BackColor, a member of class IAmbientDispatch.

12.2.1.2.3 IAmbientDispatch.DisplayAsDefault Property

Pascal

```
property DisplayAsDefault: WordBool;
```

C++

```
__property WordBool DisplayAsDefault;
```

Description

This is DisplayAsDefault, a member of class IAmbientDispatch.

12.2.1.2.4 IAmbientDispatch.DisplayName Property

Pascal

```
property DisplayName: WideString;
```

C++

```
__property BSTR DisplayName;
```

Description

This is DisplayName, a member of class IAmbientDispatch.

12.2.1.2.5 IAmbientDispatch.Font Property

Pascal

```
property Font: IFontDisp;
```

C++

```
__property IFontDisp Font;
```

Description

This is Font, a member of class IAmbientDispatch.

12.2.1.2.6 IAmbientDispatch.ForeColor Property

Pascal

```
property ForeColor: Integer;
```

C++

```
__property int ForeColor;
```

Description

This is ForeColor, a member of class IAmbientDispatch.

12.2.1.2.7 IAmbientDispatch.LocaleID Property

Pascal

```
property LocaleID: Integer;
```

C++

```
__property int LocaleID;
```

Description

This is LocaleID, a member of class IAmbientDispatch.

12.2.1.2.8 IAmbientDispatch.MessageReflect Property

Pascal

```
property MessageReflect: WordBool;
```

C++

```
__property WordBool MessageReflect;
```

Description

This is MessageReflect, a member of class IAmbientDispatch.

12.2.1.2.9 IAmbientDispatch.ScaleUnits Property

Pascal

```
property ScaleUnits: WideString;
```

C++

```
__property BSTR ScaleUnits;
```

Description

This is ScaleUnits, a member of class IAmbientDispatch.

12.2.1.2.10 IAmbientDispatch.ShowGrabHandles Property

Pascal

```
property ShowGrabHandles: WordBool;
```

C++

```
__property WordBool ShowGrabHandles;
```

Description

This is ShowGrabHandles, a member of class IAmbientDispatch.

12.2.1.2.11 IAmbientDispatch.ShowHatching Property

Pascal

```
property ShowHatching: WordBool;
```

C++

```
__property WordBool ShowHatching;
```

Description

This is ShowHatching, a member of class IAmbientDispatch.

12.2.1.2.12 IAmbientDispatch.SupportsMnemonics Property

Pascal

```
property SupportsMnemonics: WordBool;
```

C++

```
__property WordBool SupportsMnemonics;
```

Description

This is SupportsMnemonics, a member of class IAmbientDispatch.

12.2.1.2.13 IAmbientDispatch.TextAlign Property

Pascal

```
property TextAlign: Smallint;
```

C++

```
__property Smallint TextAlign;
```

Description

This is TextAlign, a member of class IAmbientDispatch.

12.2.1.2.14 IAmbientDispatch.UIDead Property

Pascal

```
property UIDead: WordBool;
```

C++

```
__property WordBool UIDead;
```

Description

This is UIDead, a member of class IAmbientDispatch.

12.2.1.2.15 IAmbientDispatch.UserMode Property

Pascal

```
property UserMode: WordBool;
```

C++

```
__property WordBool UserMode;
```

Description

This is UserMode, a member of class IAmbientDispatch.

12.2.2 IFontAccess Interface

Class Hierarchy

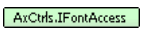

Pascal

```
IFontAccess = interface;
```

C++

```
__interface IFontAccess;
```

File

AxCtrls

Description

This is class AxCtrls.IFontAccess.

12.2.2.1 IFontAccess Members

The following tables list the members exposed by IFontAccess.

Public Methods

	Name	Description
	GetOleFont (see page 1707)	This is GetOleFont, a member of class IFontAccess.
	SetOleFont (see page 1707)	This is SetOleFont, a member of class IFontAccess.

12.2.2.2 IFontAccess Methods

The methods of the IFontAccess class are listed here.

Public Methods

	Name	Description
	GetOleFont (see page 1707)	This is GetOleFont, a member of class IFontAccess.
	SetOleFont (see page 1707)	This is SetOleFont, a member of class IFontAccess.

12.2.2.2.1 IFontAccess.GetOleFont Method

Pascal

```
procedure GetOleFont(var OleFont: IFontDisp);
```

C++

```
__fastcall GetOleFont(IFontDisp OleFont);
```

Description

This is GetOleFont, a member of class IFontAccess.

12.2.2.2.2 IFontAccess.SetOleFont Method

Pascal

```
procedure SetOleFont(const OleFont: IFontDisp);
```

C++

```
__fastcall SetOleFont(const IFontDisp OleFont);
```

Description

This is SetOleFont, a member of class IFontAccess.

12.2.3 IPictureAccess Interface

Class Hierarchy

AxCtrls.IPictureAccess

Pascal

```
IPictureAccess = interface;
```

C++

```
__interface IPictureAccess;
```

File

AxCtrls

Description

This is class AxCtrls.IPictureAccess.

12.2.3.1 IPictureAccess Members

The following tables list the members exposed by IPictureAccess.

Public Methods

	Name	Description
	GetOlePicture (see page 1708)	This is GetOlePicture, a member of class IPictureAccess.
	SetOlePicture (see page 1708)	This is SetOlePicture, a member of class IPictureAccess.

12.2.3.2 IPictureAccess Methods

The methods of the IPictureAccess class are listed here.

Public Methods

	Name	Description
	GetOlePicture (see page 1708)	This is GetOlePicture, a member of class IPictureAccess.
	SetOlePicture (see page 1708)	This is SetOlePicture, a member of class IPictureAccess.

12.2.3.2.1 IPictureAccess.GetOlePicture Method

Pascal

```
procedure GetOlePicture(var OlePicture: IPictureDisp);
```

C++

```
__fastcall GetOlePicture(IPictureDisp OlePicture);
```

Description

This is GetOlePicture, a member of class IPictureAccess.

12.2.3.2.2 IPictureAccess.SetOlePicture Method

Pascal

```
procedure SetOlePicture(const OlePicture: IPictureDisp);
```

C++

```
__fastcall SetOlePicture(const IPictureDisp OlePicture);
```

Description

This is SetOlePicture, a member of class IPictureAccess.

12.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	EnumDispatchProperties (see page 1709)	Fills a TStringList will all the property names and DispIDs in a specified IDispatch interface.
	GetOleFont (see page 1710)	Creates an OLE font object that is directly mapped to a native TFont.
	GetOlePicture (see page 1710)	Creates an OLE picture object that is directly mapped to a native TPicture.
	GetOleStrings (see page 1711)	Implements a TStrings object as an IStrings interface for use by OLE objects.
	ParkingWindow (see page 1711)	Provides a temporary parent window for ActiveX controls when the container is not ready to act as a parent.
	SetOleFont (see page 1712)	Connects an OLE font object to a TFont object and copies the properties of the OLE font object to the TFont.
	SetOlePicture (see page 1712)	Connects an OLE picture object to a TPicture object, and copies the properties of the OLE picture object to the TPicture.
	SetOleStrings (see page 1713)	Uses an IStrings interface to assign the contents of a TStrings object.

12.3.1 AxCtrls.EnumDispatchProperties Function

Fills a TStringList will all the property names and DispIDs in a specified IDispatch interface.

Pascal

```
procedure EnumDispatchProperties(Dispatch: IDispatch; PropType: TGUID; VTCode: Integer;
PropList: TStrings);
```

C++

```
EnumDispatchProperties(IDispatch Dispatch, TGUID PropType, int VTCode, TStrings PropList);
```

File

AxCtrls

Description

Use EnumDispatchProperties to get the names of the properties supported by an IDispatch interface. The Dispatch parameter is an instance of the IDispatch interface. The PropType and VTCode parameters allow the caller to limit the list to only those properties that match a particular GUID and type identifier. To see all properties of the IDispatch interface, set PropType to GUID_NULL and VTCode to VT_EMPTY.

The property names that the IDispatch interface supports are added to the list passed in as the PropList parameter.

See Also

TStringList

SetDispatchPropValue

12.3.2 AxCtrls.GetOleFont Function

Creates an OLE font object that is directly mapped to a native TFont.

Pascal

```
procedure GetOleFont(Font: TFont; var OleFont: IFontDisp);
```

C++

```
GetOleFont(TFont Font, IFontDisp OleFont);
```

File

AxCtrls

Description

OLE conventions implement fonts as automation objects that contain the properties of the font. This routine creates an adapter object that maps the properties of a TFont object into an OLE font object, thereby allowing an OLE client to access the font properties of a VCL control. Call this function when you implement a method that retrieves a TFont value from a VCL object and returns it as an OLE font.

The Font parameter is a pointer to the native TFont. This can be the Font property of a VCL control. The OLE font is returned in the OleFont parameter. The resulting OLE font object is a direct connection to the native VCL font, so changing the OLE font's properties will change the TFont's properties, and vice-versa.

See Also

SetOleFont (see page 1712)

TFontAdapter (see page 1583)

12.3.3 AxCtrls.GetOlePicture Function

Creates an OLE picture object that is directly mapped to a native TPicture.

Pascal

```
procedure GetOlePicture(Picture: TPicture; var OlePicture: IPictureDisp);
```

C++

```
GetOlePicture(TPicture Picture, IPictureDisp OlePicture);
```

File

AxCtrls

Description

OLE conventions implement pictures as automation objects that contain the properties of the picture. This routine creates an adapter object that maps the properties of a native VCL TPicture into an OLE picture object, thereby allowing an OLE client to access the picture properties of a VCL control. Call this function when you implement a method that retrieves a TPicture value from a VCL object and returns it as an OLE picture.

The Picture parameter is a pointer to the native TPicture. This can be the Picture property of a VCL control. The OLE picture is returned in the OlePicture parameter. The resulting OLE picture object is a direct connection to the native VCL picture, so changing the OLE picture's properties will change the TPicture's properties, and vice-versa.

See Also

SetOlePicture (see page 1712)

TPictureAdapter (see page 1599)

Marshaling Data

12.3.4 AxCtrls.GetOleStrings Function

Implements a TStrings object as an IStrings interface for use by OLE objects.

Pascal

```
procedure GetOleStrings(Strings: TStrings; var OleStrings: IStrings);
```

C++

```
GetOleStrings(TStrings Strings, IStrings OleStrings);
```

File

AxCtrls

Description

When the GetOleStrings function is called, the Strings parameter is implemented as an IStrings and assigned to the StringsAdapter (TStrings) property. The IStrings interface is returned as the OleStrings parameter.

See Also

TStringsAdapter (see page 1696)

TStrings

SetOleStrings (see page 1713)

Marshaling Data

12.3.5 AxCtrls.ParkingWindow Function

Provides a temporary parent window for ActiveX controls when the container is not ready to act as a parent.

Pascal

```
function ParkingWindow: HWND;
```

C++

```
HWND ParkingWindow();
```

File

AxCtrls

Description

Call ParkingWindow to get the window handle of the parking window. This can be used as a temporary parent to activeX controls until the host application's container is ready.

See Also

TWinControl

TWinControl

12.3.6 AxCtrls.SetOleFont Function

Connects an OLE font object to a TFont object and copies the properties of the OLE font object to the TFont.

Pascal

```
procedure SetOleFont(Font: TFont; OleFont: IFontDisp);
```

C++

```
SetOleFont(TFont Font, IFontDisp OleFont);
```

File

AxCtrls

Description

OLE conventions implement fonts as automation objects that contain the properties of the font. This routine associates an OLE font object (the OleFont parameter) with a native VCL font (the Font parameter), allowing the font to be accessed by both OLE programs and VCL programs. The TFont object is initialized to have the same properties as OleFont.

Call this method when you implement an automation method that takes an OLE font as a parameter, and which needs to pass the font along as a native TFont object.

See Also

GetOleFont (see page 1710)

TFontAdapter (see page 1583)

12.3.7 AxCtrls.SetOlePicture Function

Connects an OLE picture object to a TPicture object, and copies the properties of the OLE picture object to the TPicture.

Pascal

```
procedure SetOlePicture(Picture: TPicture; OlePicture: IPictureDisp);
```

C++

```
SetOlePicture(TPicture Picture, IPictureDisp OlePicture);
```

File

AxCtrls

Description

OLE conventions implement pictures as automation objects that contain the properties of the picture. This routine associates an OLE picture object (the OlePicture parameter) with a native VCL picture (the Picture parameter), allowing the picture to be accessed by both OLE programs and VCL programs. The TPicture object is initialized to have the same properties as OlePicture.

Call this method when you implement an automation method that takes an OLE picture as a parameter, and which needs to pass

the picture along as a native TPicture object.

See Also

GetOlePicture (see page 1710)

TPictureAdapter (see page 1599)

12.3.8 AxCtrls.SetOleStrings Function

Uses an IStrings interface to assign the contents of a TStrings object.

Pascal

```
procedure SetOleStrings(Strings: TStrings; OleStrings: IStrings);
```

C++

```
SetOleStrings(TStrings Strings, IStrings OleStrings);
```

File

AxCtrls

Description

Use SetOleStrings to update a TStrings object so that it matches the strings supplied by an IStrings interface. The Strings parameter is the TStrings object to update. The OleStrings parameter is the interface used as a source.

See Also

TStringsAdapter (see page 1696)

TStrings

GetOleStrings (see page 1711)

12.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TConnectionKind (see page 1714)	This is record AxCtrls.TConnectionKind.

12.4.1 AxCtrls.TConnectionKind Enumeration

Pascal

```
TConnectionKind = (
  ckSingle,
  ckMulti
);
```

C++

```
enum TConnectionKind {
  ckSingle,
  ckMulti
};
```

File

AxCtrls

Description

This is record AxCtrls.TConnectionKind.

12.5 Types

The following table lists types in this documentation.

Types

Name	Description
TActiveFormClass (see page 1715)	This is type AxCtrls.TActiveFormClass.
TActiveXControlClass (see page 1715)	This is type AxCtrls.TActiveXControlClass.
TDefinePropertyPage (see page 1716)	This is type AxCtrls.TDefinePropertyPage.
TPropertyPageClass (see page 1716)	This is type AxCtrls.TPropertyPageClass.

12.5.1 AxCtrls.TActiveFormClass Type

Pascal

```
TActiveFormClass = class of TActiveForm;
```

C++

```
TActiveForm TActiveFormClass;
```

File

AxCtrls

Description

This is type AxCtrls.TActiveFormClass.

12.5.2 AxCtrls.TActiveXControlClass Type

Pascal

```
TActiveXControlClass = class of TActiveXControl;
```

C++

```
TActiveXControl TActiveXControlClass;
```

File

AxCtrls

Description

This is type AxCtrls.TActiveXControlClass.

12.5.3 AxCtrls.TDefinePropertyPage Type

Pascal

```
TDefinePropertyPage = procedure (const GUID: TGUID) of object;
```

C++

```
(const GUID: TGUID) ( TDefinePropertyPage)();
```

File

AxCtrls

Description

This is type AxCtrls.TDefinePropertyPage.

12.5.4 AxCtrls.TPropertyPageClass Type

Pascal

```
TPropertyPageClass = class of TPropertyPage;
```

C++

```
TPropertyPage TPropertyPageClass;
```

File

AxCtrls

Description

This is type AxCtrls.TPropertyPageClass.

12.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
Class_DColorPropPage (see page 1717)	This is constant AxCtrls.Class_DColorPropPage.
Class_DFontPropPage (see page 1717)	This is constant AxCtrls.Class_DFontPropPage.
Class_DPicturePropPage (see page 1718)	This is constant AxCtrls.Class_DPicturePropPage.
Class_DStringPropPage (see page 1718)	This is constant AxCtrls.Class_DStringPropPage.

12.6.1 AxCtrls.Class_DColorPropPage Constant

Pascal

```
Class_DColorPropPage: TGUID = '{5CFF5D59-5946-11D0-BDEF-00A024D1875C}';
```

C++

```
TGUID Class_DColorPropPage = '{5CFF5D59-5946-11D0-BDEF-00A024D1875C}';
```

File

AxCtrls

Description

This is constant AxCtrls.Class_DColorPropPage.

12.6.2 AxCtrls.Class_DFontPropPage Constant

Pascal

```
Class_DFontPropPage: TGUID = '{5CFF5D5B-5946-11D0-BDEF-00A024D1875C}';
```

C++

```
TGUID Class_DFontPropPage = '{5CFF5D5B-5946-11D0-BDEF-00A024D1875C}';
```

File

AxCtrls

Description

This is constant AxCtrls.Class_DFontPropPage.

12.6.3 AxCtrls.Class_DPicturePropPage Constant

Pascal

```
Class_DPicturePropPage: TGUID = '{5CFF5D5A-5946-11D0-BDEF-00A024D1875C}';
```

C++

```
TGUID Class_DPicturePropPage = '{5CFF5D5A-5946-11D0-BDEF-00A024D1875C}';
```

File

AxCtrls

Description

This is constant AxCtrls.Class_DPicturePropPage.

12.6.4 AxCtrls.Class_DStringPropPage Constant

Pascal

```
Class_DStringPropPage: TGUID = '{F42D677E-754B-11D0-BDFB-00A024D1875C}';
```

C++

```
TGUID Class_DStringPropPage = '{F42D677E-754B-11D0-BDFB-00A024D1875C}';
```

File

AxCtrls

Description

This is constant AxCtrls.Class_DStringPropPage.

13 BandActn Namespace

This is namespace BandActn.

Topics

Name	Description
Classes (see page 1720)	The following table lists classes in this documentation.

Classes

	Name	Description
	TCustomizeActionBars (see page 1720)	TCustomizeActionBars is an action that causes the ActionBands customization dialog to appear.

13.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCustomizeActionBars (see page 1720)	TCustomizeActionBars is an action that causes the ActionBands customization dialog to appear.

13.1.1 TCustomizeActionBars Class

TCustomizeActionBars is an action that causes the ActionBands customization dialog to appear.

Class Hierarchy

Pascal

```
TCustomizeActionBars = class(TAction);
```

C++

```
class TCustomizeActionBars : public TAction;
```

File

BandActn

Description

TCustomizeActionBars is an action that causes the ActionBands customization dialog to appear. This dialog allows users to customize the ActionBands components (such as the main menu component TActionMainMenuBar) as they see fit. This action works only with the ActionBands action manager (TActionManager); it will not work with action lists by themselves.

See Also

TActionManager (see page 432)

TCustomizeDlg (see page 6528)

TActionMainMenuBar (see page 585)

TActionToolBar (see page 38)

13.1.1.1 TCustomizeActionBars Members

The following tables list the members exposed by TCustomizeActionBars.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.

	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1725)	Creates and initializes a TCustomizeActionBars (see page 1720) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".

	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 1725)	Creates the actual ActionBands customization dialog object.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 1726)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.

	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	ActionManager (see page 1726)	Refers to the application's action manager.
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
R	CustomizeDlg (see page 1726)	Provides access to the dialog that implements the customize dialog.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.

	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

13.1.1.2 TCustomizeActionBars.Create Constructor

Creates and initializes a TCustomizeActionBars (see page 1720) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomizeActionBars(TComponent * AOwner);
```

Description

Applications do not usually create action objects explicitly. Actions are created automatically when you add them to an action manager or action list component at design time.

See Also

ActionList (see page 295)

13.1.1.3 TCustomizeActionBars Methods

The methods of the TCustomizeActionBars class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1725)	Creates the actual ActionBands customization dialog object.
	HandlesTarget (see page 1726)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.

13.1.1.3.1 TCustomizeActionBars.ExecuteTarget Method

Creates the actual ActionBands customization dialog object.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

ExecuteTarget creates an instance of TCustomizeDlg, the ActionBands customization dialog, prepares it for use and makes it visible to the user.

See Also

ExecuteTarget (see page 2712)

HandlesTarget (see page 2713)

13.1.1.3.2 TCustomizeActionBars.HandlesTarget Method

Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

HandlesTarget was introduced in TBasicAction so that descendants can override it to check the type and state of a target. That was done in TCustomizeActionBars (see page 1720), but since Target is never really the target (the true target is always the ActionBands customization dialog, whose location is stored internally), HandlesTarget always returns true.

See Also

ExecuteTarget (see page 1725)

HandlesTarget (see page 2713)

13.1.1.4 TCustomizeActionBars Properties

The properties of the TCustomizeActionBars class are listed here.

Published Properties

	Name	Description
	ActionManager (see page 1726)	Refers to the application's action manager.
R	CustomizeDlg (see page 1726)	Provides access to the dialog that implements the customize dialog.

13.1.1.4.1 TCustomizeActionBars.ActionManager Property

Refers to the application's action manager.

Pascal

```
property ActionManager: TCustomActionManager;
```

C++

```
__property TCustomActionManager ActionManager;
```

Description

Use ActionManager to refer to the application's action manager object.

See Also

TActionManager (see page 432)

TCustomActionManager (see page 537)

13.1.1.4.2 TCustomizeActionBars.CustomizeDlg Property

Provides access to the dialog that implements the customize dialog.

Pascal

```
property CustomizeDlg: TCustomizeDlg;
```

C++

```
__property TCustomizeDlg CustomizeDlg;
```

Description

Read CustomizeDlg to access the dialog that TCustomizeActionBars (see page 1720) displays when you call the Show method.

See Also

TCustomizeDlg (see page 6528)

14 BDEReg Namespace

This is namespace BDEReg.

Topics

Name	Description
Classes (see page 1731)	The following table lists classes in this documentation.
Functions (see page 1756)	The following table lists functions in this documentation.
Constants (see page 1758)	The following table lists constants in this documentation.

Classes

	Name	Description
	TBatchMoveSprig (see page 1731)	This is class BDEReg.TBatchMoveSprig.
	TBDEDataSetSprig (see page 1732)	This is class BDEReg.TBDEDataSetSprig.
	TDatabaseSprig (see page 1733)	This is class BDEReg.TDatabaseSprig.
	TDBDataSetSprig (see page 1736)	This is class BDEReg.TDBDataSetSprig.
	TDefaultSessionSprig (see page 1738)	This is class BDEReg.TDefaultSessionSprig.
	TImpliedDatabaseSprig (see page 1739)	This is class BDEReg.TImpliedDatabaseSprig.
	TImpliedSessionSprig (see page 1742)	This is class BDEReg.TImpliedSessionSprig.
	TNestedTableSprig (see page 1743)	This is class BDEReg.TNestedTableSprig.
	TQuerySprig (see page 1744)	This is class BDEReg.TQuerySprig.
	TSessionSprig (see page 1745)	This is class BDEReg.TSessionSprig.
	TStoredProcSprig (see page 1747)	This is class BDEReg.TStoredProcSprig.
	TTableFieldLinkProperty (see page 1748)	This is class BDEReg.TTableFieldLinkProperty.
	TTableSprig (see page 1753)	This is class BDEReg.TTableSprig.
	TUpdateSQLSprig (see page 1754)	This is class BDEReg.TUpdateSQLSprig.

Constants

Name	Description
cDefaultSessionSprigName (see page 1758)	This is constant BDEReg.cDefaultSessionSprigName.
cImpliedDatabaseSprigPrefix (see page 1758)	This is constant BDEReg.cImpliedDatabaseSprigPrefix.
cImpliedSessionSprigPrefix (see page 1758)	This is constant BDEReg.cImpliedSessionSprigPrefix.

Functions

	Name	Description
	Register (see page 1756)	This is function BDEReg.Register.
	SprigBDEImpliedDatabaseName (see page 1756)	This is function BDEReg.SprigBDEImpliedDatabaseName.

	SprigBDEImpliedSessionName (see page 1757)	This is function BDEReg.SprigBDEImpliedSessionName.
	SprigBDESessionName (see page 1757)	This is function BDEReg.SprigBDESessionName.

14.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TBatchMoveSprig (see page 1731)	This is class BDEReg.TBatchMoveSprig.
	TBDEDataSetSprig (see page 1732)	This is class BDEReg.TBDEDataSetSprig.
	TDatabaseSprig (see page 1733)	This is class BDEReg.TDatabaseSprig.
	TDBDataSetSprig (see page 1736)	This is class BDEReg.TDBDataSetSprig.
	TDefaultSessionSprig (see page 1738)	This is class BDEReg.TDefaultSessionSprig.
	TImpliedDatabaseSprig (see page 1739)	This is class BDEReg.TImpliedDatabaseSprig.
	TImpliedSessionSprig (see page 1742)	This is class BDEReg.TImpliedSessionSprig.
	TNestedTableSprig (see page 1743)	This is class BDEReg.TNestedTableSprig.
	TQuerySprig (see page 1744)	This is class BDEReg.TQuerySprig.
	TSessionSprig (see page 1745)	This is class BDEReg.TSessionSprig.
	TStoredProcSprig (see page 1747)	This is class BDEReg.TStoredProcSprig.
	TTableFieldLinkProperty (see page 1748)	This is class BDEReg.TTableFieldLinkProperty.
	TTableSprig (see page 1753)	This is class BDEReg.TTableSprig.
	TUpdateSQLSprig (see page 1754)	This is class BDEReg.TUpdateSQLSprig.

14.1.1 TBatchMoveSprig Class

Class Hierarchy

Pascal

```
TBatchMoveSprig = class(TComponentSprig);
```

C++

```
class TBatchMoveSprig : public TComponentSprig;
```

File

BDEReg

Description

This is class BDEReg.TBatchMoveSprig.

14.1.1.1 TBatchMoveSprig Members

The following tables list the members exposed by TBatchMoveSprig.

Public Methods

	Name	Description
	AnyProblems (see page 1732)	This is AnyProblems, a member of class TBatchMoveSprig.
	Caption (see page 1732)	This is Caption, a member of class TBatchMoveSprig.

14.1.1.2 TBatchMoveSprig Methods

The methods of the TBatchMoveSprig class are listed here.

Public Methods

	Name	Description
	AnyProblems (see page 1732)	This is AnyProblems, a member of class TBatchMoveSprig.
	Caption (see page 1732)	This is Caption, a member of class TBatchMoveSprig.

14.1.1.2.1 TBatchMoveSprig.AnyProblems Method

Pascal

```
function AnyProblems: Boolean; override;
```

C++

```
virtual __fastcall Boolean AnyProblems();
```

Description

This is AnyProblems, a member of class TBatchMoveSprig.

14.1.1.2.2 TBatchMoveSprig.Caption Method

Pascal

```
function Caption: string; override;
```

C++

```
virtual __fastcall AnsiString Caption();
```

Description

This is Caption, a member of class TBatchMoveSprig.

14.1.2 TBDEDataSetSprig Class

Class Hierarchy

Pascal

```
TBDEDataSetSprig = class(TDataSetSprig);
```

C++

```
class TBDEDataSetSprig : public TDataSetSprig;
```

File

BDEReg

Description

This is class BDEReg.TBDEDataSetSprig.

14.1.2.1 TBDEDataSetSprig Members

The following tables list the members exposed by TBDEDataSetSprig.

Public Methods

	Name	Description
	GetDSDesignerClass (see page 1733)	This is GetDSDesignerClass, a member of class TBDEDataSetSprig.

14.1.2.2 TBDEDataSetSprig Methods

The methods of the TBDEDataSetSprig class are listed here.

Public Methods

	Name	Description
	GetDSDesignerClass (see page 1733)	This is GetDSDesignerClass, a member of class TBDEDataSetSprig.

14.1.2.2.1 TBDEDataSetSprig.GetDSDesignerClass Method

Pascal

```
function GetDSDesignerClass: TDSDesignerClass; override;
```

C++

```
virtual __fastcall TDSDesignerClass GetDSDesignerClass();
```

Description

This is GetDSDesignerClass, a member of class TBDEDataSetSprig.

14.1.3 TDatabaseSprig Class

Class Hierarchy

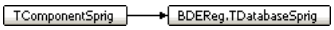

Pascal

```
TDatabaseSprig = class(TComponentSprig);
```

C++

```
class TDatabaseSprig : public TComponentSprig;
```

File

BDEReg

Description

This is class BDEReg.TDatabaseSprig.

14.1.3.1 TDatabaseSprig Members

The following tables list the members exposed by TDatabaseSprig.

Public Methods

	Name	Description
	AnyProblems (see page 1734)	This is AnyProblems, a member of class TDatabaseSprig.
	Caption (see page 1734)	This is Caption, a member of class TDatabaseSprig.
	DragDropTo (see page 1734)	This is DragDropTo, a member of class TDatabaseSprig.
	DragOverTo (see page 1735)	This is DragOverTo, a member of class TDatabaseSprig.
	FigureParent (see page 1735)	This is FigureParent, a member of class TDatabaseSprig.
	Name (see page 1735)	This is Name, a member of class TDatabaseSprig.
	PaletteOverTo (see page 1735)	This is PaletteOverTo, a member of class TDatabaseSprig.

14.1.3.2 TDatabaseSprig Methods

The methods of the TDatabaseSprig class are listed here.

Public Methods

	Name	Description
	AnyProblems (see page 1734)	This is AnyProblems, a member of class TDatabaseSprig.
	Caption (see page 1734)	This is Caption, a member of class TDatabaseSprig.
	DragDropTo (see page 1734)	This is DragDropTo, a member of class TDatabaseSprig.
	DragOverTo (see page 1735)	This is DragOverTo, a member of class TDatabaseSprig.
	FigureParent (see page 1735)	This is FigureParent, a member of class TDatabaseSprig.
	Name (see page 1735)	This is Name, a member of class TDatabaseSprig.
	PaletteOverTo (see page 1735)	This is PaletteOverTo, a member of class TDatabaseSprig.

14.1.3.2.1 TDatabaseSprig.AnyProblems Method

Pascal

```
function AnyProblems: Boolean; override;
```

C++

```
virtual __fastcall Boolean AnyProblems();
```

Description

This is AnyProblems, a member of class TDatabaseSprig.

14.1.3.2.2 TDatabaseSprig.Caption Method

Pascal

```
function Caption: string; override;
```

C++

```
virtual __fastcall AnsiString Caption();
```

Description

This is Caption, a member of class TDatabaseSprig.

14.1.3.2.3 TDatabaseSprig.DragDropTo Method

Pascal

```
function DragDropTo(AItem: TSprig): Boolean; override;
```

C++

```
virtual __fastcall Boolean DragDropTo(TSprig AItem);
```

Description

This is DragDropTo, a member of class TDatabaseSprig.

14.1.3.2.4 TDatabaseSprig.DragOverTo Method

Pascal

```
function DragOverTo(AItem: TSprig): Boolean; override;
```

C++

```
virtual __fastcall Boolean DragOverTo(TSprig AItem);
```

Description

This is DragOverTo, a member of class TDatabaseSprig.

14.1.3.2.5 TDatabaseSprig.FigureParent Method

Pascal

```
procedure FigureParent; override;
```

C++

```
virtual __fastcall FigureParent();
```

Description

This is FigureParent, a member of class TDatabaseSprig.

14.1.3.2.6 TDatabaseSprig.Name Method

Pascal

```
function Name: string; override;
```

C++

```
virtual __fastcall AnsiString Name();
```

Description

This is Name, a member of class TDatabaseSprig.

14.1.3.2.7 TDatabaseSprig.PaletteOverTo Method

Pascal

```
class function PaletteOverTo(AParent: TSprig; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean PaletteOverTo(TSprig AParent, TClass AClass);
```

Description

This is PaletteOverTo, a member of class TDatabaseSprig.

14.1.4 TDBDataSetSprig Class

Class Hierarchy

Pascal

```
TDBDataSetSprig = class(TBDEDataSetSprig);
```

C++

```
class TDBDataSetSprig : public TBDEDataSetSprig;
```

File

BDEReg

Description

This is class BDEReg.TDBDataSetSprig.

14.1.4.1 TDBDataSetSprig Members

The following tables list the members exposed by TDBDataSetSprig.

Public Methods

	Name	Description
	AnyProblems (see page 1736)	This is AnyProblems, a member of class TDBDataSetSprig.
	DragDropTo (see page 1737)	This is DragDropTo, a member of class TDBDataSetSprig.
	DragOverTo (see page 1737)	This is DragOverTo, a member of class TDBDataSetSprig.
	FigureParent (see page 1737)	This is FigureParent, a member of class TDBDataSetSprig.
	GetDSDesignerClass (see page 1733)	This is GetDSDesignerClass, a member of class TBDEDataSetSprig.
	PaletteOverTo (see page 1737)	This is PaletteOverTo, a member of class TDBDataSetSprig.
	Reparent (see page 1738)	This is Reparent, a member of class TDBDataSetSprig.

14.1.4.2 TDBDataSetSprig Methods

The methods of the TDBDataSetSprig class are listed here.

Public Methods

	Name	Description
	AnyProblems (see page 1736)	This is AnyProblems, a member of class TDBDataSetSprig.
	DragDropTo (see page 1737)	This is DragDropTo, a member of class TDBDataSetSprig.
	DragOverTo (see page 1737)	This is DragOverTo, a member of class TDBDataSetSprig.
	FigureParent (see page 1737)	This is FigureParent, a member of class TDBDataSetSprig.
	PaletteOverTo (see page 1737)	This is PaletteOverTo, a member of class TDBDataSetSprig.
	Reparent (see page 1738)	This is Reparent, a member of class TDBDataSetSprig.

14.1.4.2.1 TDBDataSetSprig.AnyProblems Method

Pascal

```
function AnyProblems: Boolean; override;
```

C++

```
virtual __fastcall Boolean AnyProblems();
```

Description

This is AnyProblems, a member of class TDBDataSetSprig.

14.1.4.2.2 TDBDataSetSprig.DragDropTo Method

Pascal

```
function DragDropTo(AItem: TSprig): Boolean; override;
```

C++

```
virtual __fastcall Boolean DragDropTo(TSprig AItem);
```

Description

This is DragDropTo, a member of class TDBDataSetSprig.

14.1.4.2.3 TDBDataSetSprig.DragOverTo Method

Pascal

```
function DragOverTo(AItem: TSprig): Boolean; override;
```

C++

```
virtual __fastcall Boolean DragOverTo(TSprig AItem);
```

Description

This is DragOverTo, a member of class TDBDataSetSprig.

14.1.4.2.4 TDBDataSetSprig.FigureParent Method

Pascal

```
procedure FigureParent; override;
```

C++

```
virtual __fastcall FigureParent();
```

Description

This is FigureParent, a member of class TDBDataSetSprig.

14.1.4.2.5 TDBDataSetSprig.PaletteOverTo Method

Pascal

```
class function PaletteOverTo(AParent: TSprig; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean PaletteOverTo(TSprig AParent, TClass AClass);
```

Description

This is PaletteOverTo, a member of class TDBDataSetSprig.

14.1.4.2.6 TDBDataSetSprig.Reparent Method

Pascal

```
procedure Reparent; override;
```

C++

```
virtual __fastcall Reparent();
```

Description

This is Reparent, a member of class TDBDataSetSprig.

14.1.5 TDefaultSessionSprig Class

Class Hierarchy

Pascal

```
TDefaultSessionSprig = class(TTransientSprig);
```

C++

```
class TDefaultSessionSprig : public TTransientSprig;
```

File

BDEReg

Description

This is class BDEReg.TDefaultSessionSprig.

14.1.5.1 TDefaultSessionSprig Members

The following tables list the members exposed by TDefaultSessionSprig.

Public Methods

	Name	Description
	Caption (see page 1739)	This is Caption, a member of class TDefaultSessionSprig.
	ItemClass (see page 1739)	This is ItemClass, a member of class TDefaultSessionSprig.
	UniqueName (see page 1739)	This is UniqueName, a member of class TDefaultSessionSprig.

14.1.5.2 TDefaultSessionSprig Methods

The methods of the TDefaultSessionSprig class are listed here.

Public Methods

	Name	Description
	Caption (see page 1739)	This is Caption, a member of class TDefaultSessionSprig.
	ItemClass (see page 1739)	This is ItemClass, a member of class TDefaultSessionSprig.
	UniqueName (see page 1739)	This is UniqueName, a member of class TDefaultSessionSprig.

14.1.5.2.1 TDefaultSessionSprig.Caption Method

Pascal

```
function Caption: string; override;
```

C++

```
virtual __fastcall AnsiString Caption();
```

Description

This is Caption, a member of class TDefaultSessionSprig.

14.1.5.2.2 TDefaultSessionSprig.ItemClass Method

Pascal

```
function ItemClass: TClass; override;
```

C++

```
virtual __fastcall TClass ItemClass();
```

Description

This is ItemClass, a member of class TDefaultSessionSprig.

14.1.5.2.3 TDefaultSessionSprig.UniqueName Method

Pascal

```
function UniqueName: string; override;
```

C++

```
virtual __fastcall AnsiString UniqueName();
```

Description

This is UniqueName, a member of class TDefaultSessionSprig.

14.1.6 TImpliedDatabaseSprig Class

Class Hierarchy

Pascal

```
TImpliedDatabaseSprig = class(TTransientSprig);
```

C++

```
class TImpliedDatabaseSprig : public TTransientSprig;
```

File

BDEReg

Description

This is class BDEReg.TImpliedDatabaseSprig.

14.1.6.1 TImpliedDatabaseSprig Members

The following tables list the members exposed by TImpliedDatabaseSprig.

Public Methods

	Name	Description
	AnyProblems (see page 1740)	This is AnyProblems, a member of class TImpliedDatabaseSprig.
	Caption (see page 1740)	This is Caption, a member of class TImpliedDatabaseSprig.
	DragDropTo (see page 1741)	This is DragDropTo, a member of class TImpliedDatabaseSprig.
	DragOverTo (see page 1741)	This is DragOverTo, a member of class TImpliedDatabaseSprig.
	FigureParent (see page 1741)	This is FigureParent, a member of class TImpliedDatabaseSprig.
	ItemClass (see page 1741)	This is ItemClass, a member of class TImpliedDatabaseSprig.
	UniqueName (see page 1741)	This is UniqueName, a member of class TImpliedDatabaseSprig.

14.1.6.2 TImpliedDatabaseSprig Methods

The methods of the TImpliedDatabaseSprig class are listed here.

Public Methods

	Name	Description
	AnyProblems (see page 1740)	This is AnyProblems, a member of class TImpliedDatabaseSprig.
	Caption (see page 1740)	This is Caption, a member of class TImpliedDatabaseSprig.
	DragDropTo (see page 1741)	This is DragDropTo, a member of class TImpliedDatabaseSprig.
	DragOverTo (see page 1741)	This is DragOverTo, a member of class TImpliedDatabaseSprig.
	FigureParent (see page 1741)	This is FigureParent, a member of class TImpliedDatabaseSprig.
	ItemClass (see page 1741)	This is ItemClass, a member of class TImpliedDatabaseSprig.
	UniqueName (see page 1741)	This is UniqueName, a member of class TImpliedDatabaseSprig.

14.1.6.2.1 TImpliedDatabaseSprig.AnyProblems Method

Pascal

```
function AnyProblems: Boolean; override;
```

C++

```
virtual __fastcall Boolean AnyProblems();
```

Description

This is AnyProblems, a member of class TImpliedDatabaseSprig.

14.1.6.2.2 TImpliedDatabaseSprig.Caption Method

Pascal

```
function Caption: string; override;
```

C++

```
virtual __fastcall AnsiString Caption();
```

Description

This is Caption, a member of class TImpliedDatabaseSprig.

14.1.6.2.3 TImpliedDatabaseSprig.DragDropTo Method

Pascal

```
function DragDropTo(AItem: TSprig): Boolean; override;
```

C++

```
virtual __fastcall Boolean DragDropTo(TSprig AItem);
```

Description

This is DragDropTo, a member of class TImpliedDatabaseSprig.

14.1.6.2.4 TImpliedDatabaseSprig.DragOverTo Method

Pascal

```
function DragOverTo(AItem: TSprig): Boolean; override;
```

C++

```
virtual __fastcall Boolean DragOverTo(TSprig AItem);
```

Description

This is DragOverTo, a member of class TImpliedDatabaseSprig.

14.1.6.2.5 TImpliedDatabaseSprig.FigureParent Method

Pascal

```
procedure FigureParent; override;
```

C++

```
virtual __fastcall FigureParent();
```

Description

This is FigureParent, a member of class TImpliedDatabaseSprig.

14.1.6.2.6 TImpliedDatabaseSprig.ItemClass Method

Pascal

```
function ItemClass: TClass; override;
```

C++

```
virtual __fastcall TClass ItemClass();
```

Description

This is ItemClass, a member of class TImpliedDatabaseSprig.

14.1.6.2.7 TImpliedDatabaseSprig.UniqueName Method

Pascal

```
function UniqueName: string; override;
```

C++

```
virtual __fastcall AnsiString UniqueName();
```

Description

This is UniqueName, a member of class TImpliedDatabaseSprig.

14.1.7 TImpliedSessionSprig Class

Class Hierarchy

Pascal

```
TImpliedSessionSprig = class(TTransientSprig);
```

C++

```
class TImpliedSessionSprig : public TTransientSprig;
```

File

BDEReg

Description

This is class BDEReg.TImpliedSessionSprig.

14.1.7.1 TImpliedSessionSprig Members

The following tables list the members exposed by TImpliedSessionSprig.

Public Methods

	Name	Description
	Caption (see page 1742)	This is Caption, a member of class TImpliedSessionSprig.
	ItemClass (see page 1743)	This is ItemClass, a member of class TImpliedSessionSprig.
	UniqueName (see page 1743)	This is UniqueName, a member of class TImpliedSessionSprig.

14.1.7.2 TImpliedSessionSprig Methods

The methods of the TImpliedSessionSprig class are listed here.

Public Methods

	Name	Description
	Caption (see page 1742)	This is Caption, a member of class TImpliedSessionSprig.
	ItemClass (see page 1743)	This is ItemClass, a member of class TImpliedSessionSprig.
	UniqueName (see page 1743)	This is UniqueName, a member of class TImpliedSessionSprig.

14.1.7.2.1 TImpliedSessionSprig.Caption Method

Pascal

```
function Caption: string; override;
```

C++

```
virtual __fastcall AnsiString Caption();
```

Description

This is Caption, a member of class TImpliedSessionSprig.

14.1.7.2.2 TImpliedSessionSprig.ItemClass Method

Pascal

```
function ItemClass: TClass; override;
```

C++

```
virtual __fastcall TClass ItemClass();
```

Description

This is ItemClass, a member of class TImpliedSessionSprig.

14.1.7.2.3 TImpliedSessionSprig.UniqueName Method

Pascal

```
function UniqueName: string; override;
```

C++

```
virtual __fastcall AnsiString UniqueName();
```

Description

This is UniqueName, a member of class TImpliedSessionSprig.

14.1.8 TNestedTableSprig Class

Class Hierarchy

Pascal

```
TNestedTableSprig = class(TBDEDataSetSprig);
```

C++

```
class TNestedTableSprig : public TBDEDataSetSprig;
```

File

BDEReg

Description

This is class BDEReg.TNestedTableSprig.

14.1.8.1 TNestedTableSprig Members

The following tables list the members exposed by TNestedTableSprig.

Public Methods

	Name	Description
	GetDSDesignerClass (see page 1733)	This is GetDSDesignerClass, a member of class TBDEDataSetSprig.
	ParentProperty (see page 1744)	This is ParentProperty, a member of class TNestedTableSprig.

14.1.8.2 TNestedTableSprig Methods

The methods of the TNestedTableSprig class are listed here.

Public Methods

	Name	Description
	ParentProperty (see page 1744)	This is ParentProperty, a member of class TNestedTableSprig.

14.1.8.2.1 TNestedTableSprig.ParentProperty Method

Pascal

```
class function ParentProperty: string; override;
```

C++

```
virtual __fastcall AnsiString ParentProperty();
```

Description

This is ParentProperty, a member of class TNestedTableSprig.

14.1.9 TQuerySprig Class

Class Hierarchy

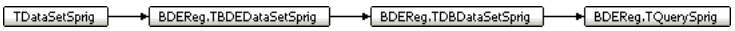

Pascal

```
TQuerySprig = class(TDBDataSetSprig);
```

C++

```
class TQuerySprig : public TDBDataSetSprig;
```

File

BDEReg

Description

This is class BDEReg.TQuerySprig.

14.1.9.1 TQuerySprig Members

The following tables list the members exposed by TQuerySprig.

Public Methods

	Name	Description
	AnyProblems (see page 1745)	This is AnyProblems, a member of class TQuerySprig.

	DragDropTo (see page 1737)	This is DragDropTo, a member of class TDBDataSetSprig.
	DragOverTo (see page 1737)	This is DragOverTo, a member of class TDBDataSetSprig.
	FigureParent (see page 1737)	This is FigureParent, a member of class TDBDataSetSprig.
	GetDSDesignerClass (see page 1733)	This is GetDSDesignerClass, a member of class TBDEDataSetSprig.
	PaletteOverTo (see page 1737)	This is PaletteOverTo, a member of class TDBDataSetSprig.
	Reparent (see page 1738)	This is Reparent, a member of class TDBDataSetSprig.

14.1.9.2 TQuerySprig Methods

The methods of the TQuerySprig class are listed here.

Public Methods

	Name	Description
	AnyProblems (see page 1745)	This is AnyProblems, a member of class TQuerySprig.

14.1.9.2.1 TQuerySprig.AnyProblems Method

Pascal

```
function AnyProblems: Boolean; override;
```

C++

```
virtual __fastcall Boolean AnyProblems();
```

Description

This is AnyProblems, a member of class TQuerySprig.

14.1.10 TSessionSprig Class

Class Hierarchy

Pascal

```
TSessionSprig = class(TComponentSprig);
```

C++

```
class TSessionSprig : public TComponentSprig;
```

File

BDEReg

Description

This is class BDEReg.TSessionSprig.

14.1.10.1 TSessionSprig Members

The following tables list the members exposed by TSessionSprig.

Public Methods

	Name	Description
	AnyProblems (see page 1746)	This is AnyProblems, a member of class TSessionSprig.
	Caption (see page 1746)	This is Caption, a member of class TSessionSprig.
	Name (see page 1746)	This is Name, a member of class TSessionSprig.

14.1.10.2 TSessionSprig Methods

The methods of the TSessionSprig class are listed here.

Public Methods

	Name	Description
	AnyProblems (see page 1746)	This is AnyProblems, a member of class TSessionSprig.
	Caption (see page 1746)	This is Caption, a member of class TSessionSprig.
	Name (see page 1746)	This is Name, a member of class TSessionSprig.

14.1.10.2.1 TSessionSprig.AnyProblems Method

Pascal

```
function AnyProblems: Boolean; override;
```

C++

```
virtual __fastcall Boolean AnyProblems();
```

Description

This is AnyProblems, a member of class TSessionSprig.

14.1.10.2.2 TSessionSprig.Caption Method

Pascal

```
function Caption: string; override;
```

C++

```
virtual __fastcall AnsiString Caption();
```

Description

This is Caption, a member of class TSessionSprig.

14.1.10.2.3 TSessionSprig.Name Method

Pascal

```
function Name: string; override;
```

C++

```
virtual __fastcall AnsiString Name();
```

Description

This is Name, a member of class TSessionSprig.

14.1.11 TStoredProcSprig Class

Class Hierarchy

Pascal

```
TStoredProcSprig = class(TDBDataSetSprig);
```

C++

```
class TStoredProcSprig : public TDBDataSetSprig;
```

File

BDEReg

Description

This is class BDEReg.TStoredProcSprig.

14.1.11.1 TStoredProcSprig Members

The following tables list the members exposed by TStoredProcSprig.

Public Methods

	Name	Description
	AnyProblems (see page 1747)	This is AnyProblems, a member of class TStoredProcSprig.
	Caption (see page 1748)	This is Caption, a member of class TStoredProcSprig.
	DragDropTo (see page 1737)	This is DragDropTo, a member of class TDBDataSetSprig.
	DragOverTo (see page 1737)	This is DragOverTo, a member of class TDBDataSetSprig.
	FigureParent (see page 1737)	This is FigureParent, a member of class TDBDataSetSprig.
	GetDSDesignerClass (see page 1733)	This is GetDSDesignerClass, a member of class TBDEDataSetSprig.
	PaletteOverTo (see page 1737)	This is PaletteOverTo, a member of class TDBDataSetSprig.
	Reparent (see page 1738)	This is Reparent, a member of class TDBDataSetSprig.

14.1.11.2 TStoredProcSprig Methods

The methods of the TStoredProcSprig class are listed here.

Public Methods

	Name	Description
	AnyProblems (see page 1747)	This is AnyProblems, a member of class TStoredProcSprig.
	Caption (see page 1748)	This is Caption, a member of class TStoredProcSprig.

14.1.11.2.1 TStoredProcSprig.AnyProblems Method

Pascal

```
function AnyProblems: Boolean; override;
```

C++

```
virtual __fastcall Boolean AnyProblems();
```

Description

This is AnyProblems, a member of class TStoredProcSprig.

14.1.11.2.2 TStoredProcSprig.Caption Method

Pascal

```
function Caption: string; override;
```

C++

```
virtual __fastcall AnsiString Caption();
```

Description

This is Caption, a member of class TStoredProcSprig.

14.1.12 TTableFieldLinkProperty Class

Class Hierarchy

Pascal

```
TTableFieldLinkProperty = class(TFieldLinkProperty);
```

C++

```
class TTableFieldLinkProperty : public TFieldLinkProperty;
```

File

BDEReg

Description

This is class BDEReg.TTableFieldLinkProperty.

14.1.12.1 TTableFieldLinkProperty Members

The following tables list the members exposed by TTableFieldLinkProperty.

Protected Methods

	Name	Description
	GetFieldNamesForIndex (see page 1749)	This is GetFieldNamesForIndex, a member of class TTableFieldLinkProperty.
	GetIndexBased (see page 1750)	This is GetIndexBased, a member of class TTableFieldLinkProperty.
	GetIndexDefs (see page 1750)	This is GetIndexDefs, a member of class TTableFieldLinkProperty.
	GetIndexFieldNames (see page 1750)	This is GetIndexFieldNames, a member of class TTableFieldLinkProperty.
	GetIndexName (see page 1750)	This is GetIndexName, a member of class TTableFieldLinkProperty.
	GetMasterFields (see page 1751)	This is GetMasterFields, a member of class TTableFieldLinkProperty.
	SetIndexFieldNames (see page 1751)	This is SetIndexFieldNames, a member of class TTableFieldLinkProperty.
	SetIndexName (see page 1751)	This is SetIndexName, a member of class TTableFieldLinkProperty.
	SetMasterFields (see page 1751)	This is SetMasterFields, a member of class TTableFieldLinkProperty.

Public Methods

	Name	Description
	Edit (see page 1749)	This is Edit, a member of class TTableFieldLinkProperty.

Public Properties

	Name	Description
	IndexBased (see page 1752)	This is IndexBased, a member of class TTableFieldLinkProperty.
	IndexDefs (see page 1752)	This is IndexDefs, a member of class TTableFieldLinkProperty.
	IndexFieldNames (see page 1752)	This is IndexFieldNames, a member of class TTableFieldLinkProperty.
	IndexName (see page 1752)	This is IndexName, a member of class TTableFieldLinkProperty.
	MasterFields (see page 1753)	This is MasterFields, a member of class TTableFieldLinkProperty.

14.1.12.2 TTableFieldLinkProperty Methods

The methods of the TTableFieldLinkProperty class are listed here.

Protected Methods

	Name	Description
	GetFieldNamesForIndex (see page 1749)	This is GetFieldNamesForIndex, a member of class TTableFieldLinkProperty.
	GetIndexBased (see page 1750)	This is GetIndexBased, a member of class TTableFieldLinkProperty.
	GetIndexDefs (see page 1750)	This is GetIndexDefs, a member of class TTableFieldLinkProperty.
	GetIndexFieldNames (see page 1750)	This is GetIndexFieldNames, a member of class TTableFieldLinkProperty.
	GetIndexName (see page 1750)	This is GetIndexName, a member of class TTableFieldLinkProperty.
	GetMasterFields (see page 1751)	This is GetMasterFields, a member of class TTableFieldLinkProperty.
	SetIndexFieldNames (see page 1751)	This is SetIndexFieldNames, a member of class TTableFieldLinkProperty.
	SetIndexName (see page 1751)	This is SetIndexName, a member of class TTableFieldLinkProperty.
	SetMasterFields (see page 1751)	This is SetMasterFields, a member of class TTableFieldLinkProperty.

Public Methods

	Name	Description
	Edit (see page 1749)	This is Edit, a member of class TTableFieldLinkProperty.

14.1.12.2.1 TTableFieldLinkProperty.Edit Method

Pascal

```
procedure Edit; override;
```

C++

```
virtual __fastcall Edit();
```

Description

This is Edit, a member of class TTableFieldLinkProperty.

14.1.12.2.2 TTableFieldLinkProperty.GetFieldNamesForIndex Method

Pascal

```
procedure GetFieldNamesForIndex(List: TStrings); override;
```

C++

```
virtual __fastcall GetFieldNamesForIndex(TStrings List);
```

Description

This is GetFieldNamesForIndex, a member of class TTableFieldLinkProperty.

14.1.12.2.3 TTableFieldLinkProperty.GetIndexBased Method

Pascal

```
function GetIndexBased: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIndexBased();
```

Description

This is GetIndexBased, a member of class TTableFieldLinkProperty.

14.1.12.2.4 TTableFieldLinkProperty.GetIndexDefs Method

Pascal

```
function GetIndexDefs: TIndexDefs; override;
```

C++

```
virtual __fastcall TIndexDefs GetIndexDefs();
```

Description

This is GetIndexDefs, a member of class TTableFieldLinkProperty.

14.1.12.2.5 TTableFieldLinkProperty.GetIndexFieldNames Method

Pascal

```
function GetIndexFieldNames: string; override;
```

C++

```
virtual __fastcall AnsiString GetIndexFieldNames();
```

Description

This is GetIndexFieldNames, a member of class TTableFieldLinkProperty.

14.1.12.2.6 TTableFieldLinkProperty.GetIndexName Method

Pascal

```
function GetIndexName: string; override;
```

C++

```
virtual __fastcall AnsiString GetIndexName();
```

Description

This is GetIndexName, a member of class TTableFieldLinkProperty.

14.1.12.2.7 TTableFieldLinkProperty.GetMasterFields Method

Pascal

```
function GetMasterFields: string; override;
```

C++

```
virtual __fastcall AnsiString GetMasterFields();
```

Description

This is GetMasterFields, a member of class TTableFieldLinkProperty.

14.1.12.2.8 TTableFieldLinkProperty.SetIndexFieldNames Method

Pascal

```
procedure SetIndexFieldNames(const Value: string); override;
```

C++

```
virtual __fastcall SetIndexFieldNames(const AnsiString Value);
```

Description

This is SetIndexFieldNames, a member of class TTableFieldLinkProperty.

14.1.12.2.9 TTableFieldLinkProperty.SetIndexName Method

Pascal

```
procedure SetIndexName(const Value: string); override;
```

C++

```
virtual __fastcall SetIndexName(const AnsiString Value);
```

Description

This is SetIndexName, a member of class TTableFieldLinkProperty.

14.1.12.2.10 TTableFieldLinkProperty.SetMasterFields Method

Pascal

```
procedure SetMasterFields(const Value: string); override;
```

C++

```
virtual __fastcall SetMasterFields(const AnsiString Value);
```

Description

This is SetMasterFields, a member of class TTableFieldLinkProperty.

14.1.12.3 TTableFieldLinkProperty Properties

The properties of the TTableFieldLinkProperty class are listed here.

Public Properties

	Name	Description
R	IndexBased (see page 1752)	This is IndexBased, a member of class TTableFieldLinkProperty.
R	IndexDefs (see page 1752)	This is IndexDefs, a member of class TTableFieldLinkProperty.
	IndexFieldNames (see page 1752)	This is IndexFieldNames, a member of class TTableFieldLinkProperty.
	IndexName (see page 1752)	This is IndexName, a member of class TTableFieldLinkProperty.
	MasterFields (see page 1753)	This is MasterFields, a member of class TTableFieldLinkProperty.

14.1.12.3.1 TTableFieldLinkProperty.IndexBased Property

Pascal

```
property IndexBased: Boolean;
```

C++

```
__property Boolean IndexBased;
```

Description

This is IndexBased, a member of class TTableFieldLinkProperty.

14.1.12.3.2 TTableFieldLinkProperty.IndexDefs Property

Pascal

```
property IndexDefs: TIndexDefs;
```

C++

```
__property TIndexDefs IndexDefs;
```

Description

This is IndexDefs, a member of class TTableFieldLinkProperty.

14.1.12.3.3 TTableFieldLinkProperty.IndexFieldNames Property

Pascal

```
property IndexFieldNames: string;
```

C++

```
__property AnsiString IndexFieldNames;
```

Description

This is IndexFieldNames, a member of class TTableFieldLinkProperty.

14.1.12.3.4 TTableFieldLinkProperty.IndexName Property

Pascal

```
property IndexName: string;
```

C++

```
__property AnsiString IndexName;
```

Description

This is IndexName, a member of class TTableFieldLinkProperty.

14.1.12.3.5 TTableFieldLinkProperty.MasterFields Property

Pascal

```
property MasterFields: string;
```

C++

```
__property AnsiString MasterFields;
```

Description

This is MasterFields, a member of class TTableFieldLinkProperty.

14.1.13 TTableSprig Class

Class Hierarchy

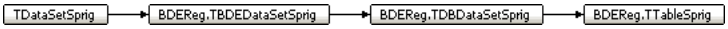

Pascal

```
TTableSprig = class(TDBDataSetSprig);
```

C++

```
class TTableSprig : public TDBDataSetSprig;
```

File

BDEReg

Description

This is class BDEReg.TTableSprig.

14.1.13.1 TTableSprig Members

The following tables list the members exposed by TTableSprig.

Public Methods

	Name	Description
	AnyProblems (see page 1754)	This is AnyProblems, a member of class TTableSprig.
	Caption (see page 1754)	This is Caption, a member of class TTableSprig.
	DragDropTo (see page 1737)	This is DragDropTo, a member of class TDBDataSetSprig.
	DragOverTo (see page 1737)	This is DragOverTo, a member of class TDBDataSetSprig.
	FigureParent (see page 1737)	This is FigureParent, a member of class TDBDataSetSprig.
	GetDSDesignerClass (see page 1733)	This is GetDSDesignerClass, a member of class TBDEDataSetSprig.
	PaletteOverTo (see page 1737)	This is PaletteOverTo, a member of class TDBDataSetSprig.
	Reparent (see page 1738)	This is Reparent, a member of class TDBDataSetSprig.

14.1.13.2 TTableSprig Methods

The methods of the TTableSprig class are listed here.

Public Methods

	Name	Description
	AnyProblems (see page 1754)	This is AnyProblems, a member of class TTableSprig.
	Caption (see page 1754)	This is Caption, a member of class TTableSprig.

14.1.13.2.1 TTableSprig.AnyProblems Method

Pascal

```
function AnyProblems: Boolean; override;
```

C++

```
virtual __fastcall Boolean AnyProblems();
```

Description

This is AnyProblems, a member of class TTableSprig.

14.1.13.2.2 TTableSprig.Caption Method

Pascal

```
function Caption: string; override;
```

C++

```
virtual __fastcall AnsiString Caption();
```

Description

This is Caption, a member of class TTableSprig.

14.1.14 TUpdateSQLSprig Class

Class Hierarchy

Pascal

```
TUpdateSQLSprig = class(TComponentSprig);
```

C++

```
class TUpdateSQLSprig : public TComponentSprig;
```

File

BDEReg

Description

This is class BDEReg.TUpdateSQLSprig.

14.1.14.1 TUpdateSQLSprig Members

The following tables list the members exposed by TUpdateSQLSprig.

Public Methods

	Name	Description
	AnyProblems (see page 1755)	This is AnyProblems, a member of class TUpdateSQLSprig.

14.1.14.2 TUpdateSQLSprig Methods

The methods of the TUpdateSQLSprig class are listed here.

Public Methods

	Name	Description
	AnyProblems (see page 1755)	This is AnyProblems, a member of class TUpdateSQLSprig.

14.1.14.2.1 TUpdateSQLSprig.AnyProblems Method

Pascal

```
function AnyProblems: Boolean; override;
```

C++

```
virtual __fastcall Boolean AnyProblems();
```

Description

This is AnyProblems, a member of class TUpdateSQLSprig.

14.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	Register (see page 1756)	This is function BDEReg.Register.
	SprigBDEImpliedDatabaseName (see page 1756)	This is function BDEReg.SprigBDEImpliedDatabaseName.
	SprigBDEImpliedSessionName (see page 1757)	This is function BDEReg.SprigBDEImpliedSessionName.
	SprigBDESessionName (see page 1757)	This is function BDEReg.SprigBDESessionName.

14.2.1 BDEReg.Register Function

Pascal

```
procedure Register;
```

C++

```
Register();
```

File

BDEReg

Description

This is function BDEReg.Register.

14.2.2 BDEReg.SprigBDEImpliedDatabaseName Function

Pascal

```
function SprigBDEImpliedDatabaseName(const AName: string): string;
```

C++

```
AnsiString SprigBDEImpliedDatabaseName(const AnsiString AName);
```

File

BDEReg

Description

This is function BDEReg.SprigBDEImpliedDatabaseName.

14.2.3 BDEReg.SprigBDEImpliedSessionName Function

Pascal

```
function SprigBDEImpliedSessionName(const AName: string): string;
```

C++

```
AnsiString SprigBDEImpliedSessionName(const AnsiString AName);
```

File

BDEReg

Description

This is function BDEReg.SprigBDEImpliedSessionName.

14.2.4 BDEReg.SprigBDESessionName Function

Pascal

```
function SprigBDESessionName(const AName: string): string;
```

C++

```
AnsiString SprigBDESessionName(const AnsiString AName);
```

File

BDEReg

Description

This is function BDEReg.SprigBDESessionName.

14.3 Constants

The following table lists constants in this documentation.

Constants

Name	Description
cDefaultSessionSprigName (see page 1758)	This is constant BDEReg.cDefaultSessionSprigName.
cImpliedDatabaseSprigPrefix (see page 1758)	This is constant BDEReg.cImpliedDatabaseSprigPrefix.
cImpliedSessionSprigPrefix (see page 1758)	This is constant BDEReg.cImpliedSessionSprigPrefix.

14.3.1 BDEReg.cDefaultSessionSprigName Constant

Pascal

```
cDefaultSessionSprigName = '<DefaultSession>';
```

C++

```
cDefaultSessionSprigName = '<DefaultSession>';
```

File

BDEReg

Description

This is constant BDEReg.cDefaultSessionSprigName.

14.3.2 BDEReg.cImpliedDatabaseSprigPrefix Constant

Pascal

```
cImpliedDatabaseSprigPrefix = '<ImpliedDatabase>';
```

C++

```
cImpliedDatabaseSprigPrefix = '<ImpliedDatabase>';
```

File

BDEReg

Description

This is constant BDEReg.cImpliedDatabaseSprigPrefix.

14.3.3 BDEReg.cImpliedSessionSprigPrefix Constant

Pascal

```
cImpliedSessionSprigPrefix = '<ImpliedSession>';
```

C++

```
cImpliedSessionSprigPrefix = '<ImpliedSession>';
```

File

BDEReg

Description

This is constant BDEReg.cImpliedSessionSprigPrefix.

15 Borland Namespace

This is namespace Borland.

Namespaces

Name	Description
Data (see page 1762)	This is namespace Borland.Data.

15.1 Borland.Data Namespace

This is namespace Borland.Data.

Namespaces

Name	Description
AdoDbxClientProvider (see page 1762)	This is namespace Borland.Data.AdoDbxClientProvider.
DataStore (see page 1892)	This is namespace Borland.Data.DataStore.
MetaData (see page 2130)	This is namespace Borland.Data.MetaData.

15.1.1 Borland.Data.AdoDbxClientProvider Namespace

This is namespace Borland.Data.AdoDbxClientProvider.

Topics

Name	Description
Classes (see page 1763)	The following table lists classes in this documentation.
Types (see page 1890)	The following table lists types in this documentation.
Constants (see page 1891)	The following table lists constants in this documentation.

Classes

	Name	Description
	TAdoDbxCommand (see page 1763)	Represents a SQL statement or stored procedure to execute against a data source.
	TAdoDbxCommandBuilder (see page 1776)	Generates single-table commands as part of the operation of the TAdoDbxDataAdapter (see page 1814).
	TAdoDbxConnection (see page 1779)	Represents a connection to a database.
	TAdoDbxConnectionStringBuilder (see page 1790)	Class for strongly typed connection string builder.
	TAdoDbxDataAdapter (see page 1814)	Acts as a bridge between a DataSet and the underlying database.
	TAdoDbxDataReader (see page 1827)	Class to read rows forward-only from a data source.
	TAdoDbxDataSourceEnumerator (see page 1845)	Provides enumerator for finding all data sources on local network.
	TAdoDbxException (see page 1845)	Base class for all data source exceptions.
	TAdoDbxExtension (see page 1846)	Class to get information about a connection.
	TAdoDbxMetaDataColumnNames (see page 1848)	Column headers for Restrictions DataTable.
	TAdoDbxParameter (see page 1850)	Represents a parameter that is passed to or from a command.
	TAdoDbxParameterCollection (see page 1865)	Collects parameters into a .NET collection object that can be read and manipulated.
	TAdoDbxProviderFactory (see page 1874)	Base class for a provider's implementation of data source classes.
	TAdoDbxRowUpdatedEventArgs (see page 1880)	This is class Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatedEventArgs.
	TAdoDbxRowUpdatingEventArgs (see page 1881)	This is class Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatingEventArgs.
	TAdoDbxSchemaTableColumnNames (see page 1883)	This is class Borland.Data.AdoDbxClientProvider.TAdoDbxSchemaTableColumnNames.
	TAdoDbxTransaction (see page 1884)	Represents an instance of the TAdoDbxTransaction object.
	TConnectionStringCategories (see page 1888)	Categories used for connection string builder UI.

Constants

Name	Description
SCommandAlreadyPrepared (see page 1891)	This is constant Borland.Data.AdoDbxClientProvider.SCommandAlreadyPrepared.
SInvalidFieldType (see page 1892)	This is constant Borland.Data.AdoDbxClientProvider.SInvalidFieldType.
SUnSupported (see page 1892)	This is constant Borland.Data.AdoDbxClientProvider.SUnSupported.

Types

Name	Description
TAdoDbxRowUpdatedEventHandler (see page 1890)	Event handler for TAdoDbxDataAdapter.RowUpdated event.
TAdoDbxRowUpdatingEventHandler (see page 1890)	Event handler for TAdoDbxDataAdapter.RowUpdating event.
TObjectArray (see page 1891)	This is type Borland.Data.AdoDbxClientProvider.TObjectArray.

15.1.1.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TAdoDbxCommand (see page 1763)	Represents a SQL statement or stored procedure to execute against a data source.
	TAdoDbxCommandBuilder (see page 1776)	Generates single-table commands as part of the operation of the TAdoDbxDataAdapter (see page 1814).
	TAdoDbxConnection (see page 1779)	Represents a connection to a database.
	TAdoDbxConnectionStringBuilder (see page 1790)	Class for strongly typed connection string builder.
	TAdoDbxDataAdapter (see page 1814)	Acts as a bridge between a DataSet and the underlying database.
	TAdoDbxDataReader (see page 1827)	Class to read rows forward-only from a data source.
	TAdoDbxDataSourceEnumerator (see page 1845)	Provides enumerator for finding all data sources on local network.
	TAdoDbxException (see page 1845)	Base class for all data source exceptions.
	TAdoDbxExtension (see page 1846)	Class to get information about a connection.
	TAdoDbxMetaDataColumnNames (see page 1848)	Column headers for Restrictions DataTable.
	TAdoDbxParameter (see page 1850)	Represents a parameter that is passed to or from a command.
	TAdoDbxParameterCollection (see page 1865)	Collects parameters into a .NET collection object that can be read and manipulated.
	TAdoDbxProviderFactory (see page 1874)	Base class for a provider's implementation of data source classes.
	TAdoDbxRowUpdatedEventArgs (see page 1880)	This is class Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatedEventArgs.
	TAdoDbxRowUpdatingEventArgs (see page 1881)	This is class Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatingEventArgs.
	TAdoDbxSchemaTableColumnNames (see page 1883)	This is class Borland.Data.AdoDbxClientProvider.TAdoDbxSchemaTableColumnNames.
	TAdoDbxTransaction (see page 1884)	Represents an instance of the TAdoDbxTransaction object.
	TConnectionStringCategories (see page 1888)	Categories used for connection string builder UI.

15.1.1.1.1 TAdoDbxCommand Class

Represents a SQL statement or stored procedure to execute against a data source.

Class Hierarchy

Pascal

```
TAdoDbxCommand = class(DbCommand, ICloneable);
```

C++

```
class TAdoDbxCommand : public DbCommand, public ICloneable;
```

File

Borland.Data.AdoDbxClientProvider

Description

Use the TAdoDbxCommand constructor to create a new instance of a TAdoDbxCommand object. The TAdoDbxCommand object is created automatically in your project when you create a TAdoDbxDataAdapter (see page 1814) object. The TAdoDbxCommand object provides the means for executing SQL statements and stored procedures.

TAdoDbxCommand is an implementation of the .NET DbCommand class.

The CommandType (see page 1773) property specifies whether a SQL statement, a table name, or a stored procedure name is being used in the CommandText (see page 1772) property.

- The Connection property should be set to a valid TAdoDbxConnection (see page 1779).
- The Transaction (see page 1774) property must be set to a valid TAdoDbxTransaction (see page 1884) to execute a SQL statement in the context of a transaction.

Depending on the type of SQL statement being executed, you can use one of the following methods:

- ExecuteReader. Used for Select statements and stored procedures that return a reader or multiple readers. A new TAdoDbxDataReader (see page 1827) is returned if ExecuteReader successfully processes the SQL statement. You cannot update data when using a TAdoDbxDataReader (see page 1827) object. To update data, use the TAdoDbxDataAdapter (see page 1814) object.
- ExecuteScalar (see page 1768). Similar to ExecuteReader but returns only the first column of the first record as an object and is primarily used to execute SQL to return aggregate values.
- ExecuteNonQuery (see page 1768). Executes DDL or non-Select DML statements or stored procedures that return no dataset. On successful execution of a DML statement, ExecuteNonQuery (see page 1768) returns the number of rows affected in the database.

To execute parameterized SQL, you can specify the CommandText (see page 1772) values with parameter markers. The **?** character is used as a parameter marker. Currently, there is no support for named parameters. During preparation, the database-specific provider implementation takes care of converting the **?** parameter markers to database-specific parameter markers and also takes care of generating the appropriate SQL for calling a stored procedure.

When executing the same SQL repeatedly, call the Prepare (see page 1770) method once and bind parameters. You specify parameters by adding a TAdoDbxParameter (see page 1850) instance to the TAdoDbxCommand.Parameters (see page 1774) property, which is a TAdoDbxParameterCollection (see page 1865). Preparing a parameterized SQL statement on most databases creates an execution plan on the server that is then used during subsequent execution of the same SQL with different parameter values.

Once the TAdoDbxCommand is complete, calling Dispose (or Free in Delphi) frees all of the statement resources allocated by the provider.

See Also

DbCommand (MSDN)

Dispose (MSDN)

ADO.NET Overview

TAdoDbxConnection (see page 1779)

TAdoDbxDataAdapter (see page 1814)

TAdoDbxParameter (see page 1850)

TAdoDbxParameterCollection (see page 1865)

TAdoDbxTransaction (see page 1884)

CommandText (see page 1772)

CommandType (see page 1773)

Connection

Parameters (see page 1774)

Transaction (see page 1774)

ExecuteNonQuery (see page 1768)

ExecuteReader

ExecuteScalar (see page 1768)

Prepare (see page 1770)

15.1.1.1.1.1 TAdoDbxCommand Members

The following tables list the members exposed by TAdoDbxCommand.

Public Methods

	Name	Description
V	Cancel (see page 1767)	Attempts to cancel a TAdoDbxCommand (see page 1763).
	Clone (see page 1767)	This is Clone, a member of class TAdoDbxCommand.
V	Destroy (see page 1766)	Destroys a TAdoDbxCommand (see page 1763) instance.
V	ExecuteNonQuery (see page 1768)	Executes a SQL statement against a connection object.
V	ExecuteScalar (see page 1768)	Execute a SQL query and return the first column of the first row of data.
V	get_CommandText (see page 1768)	This is get_CommandText, a member of class TAdoDbxCommand.
V	get_CommandTimeout (see page 1769)	This is get_CommandTimeout, a member of class TAdoDbxCommand.
V	get_CommandType (see page 1769)	This is get_CommandType, a member of class TAdoDbxCommand.
V	get_DesignTimeVisible (see page 1770)	This is get_DesignTimeVisible, a member of class TAdoDbxCommand.
V	get_UpdatedRowSource (see page 1770)	This is get_UpdatedRowSource, a member of class TAdoDbxCommand.
V	Prepare (see page 1770)	Create a compiled version of a command on the data source.
V	set_CommandText (see page 1770)	This is set_CommandText, a member of class TAdoDbxCommand.
V	set_CommandTimeout (see page 1771)	This is set_CommandTimeout, a member of class TAdoDbxCommand.
V	set_CommandType (see page 1771)	This is set_CommandType, a member of class TAdoDbxCommand.
V	set_DesignTimeVisible (see page 1771)	This is set_DesignTimeVisible, a member of class TAdoDbxCommand.
V	set_UpdatedRowSource (see page 1772)	This is set_UpdatedRowSource, a member of class TAdoDbxCommand.

Public Properties

	Name	Description
	CommandText (see page 1772)	Gets or sets the SQL statement, table name, or stored procedure to execute at the data source.
	CommandTimeout (see page 1773)	Gets or sets the time, in seconds, before terminating a long-running SQL command.
	CommandType (see page 1773)	Gets or sets the CommandType that indicates how to interpret the CommandText (see page 1772) property.
	DesignTimeVisible (see page 1774)	Determines whether or not a command is visible at design time.
R	Parameters (see page 1774)	Returns the TAdoDbxParameterCollection (see page 1865).
	Transaction (see page 1774)	Provides the transaction in which the TAdoDbxCommand (see page 1763) executes.
	UpdateRowSource (see page 1775)	This is UpdateRowSource, a member of class TAdoDbxCommand.

Strict protected Methods

	Name	Description
	CreateDbParameter (see page 1767)	This is CreateDbParameter, a member of class TAdoDbxCommand.
	ExecuteDbDataReader (see page 1767)	This is ExecuteDbDataReader, a member of class TAdoDbxCommand.
	get_DbConnection (see page 1769)	This is get_DbConnection, a member of class TAdoDbxCommand.
	get_DbParameterCollection (see page 1769)	This is get_DbParameterCollection, a member of class TAdoDbxCommand.
	get_DbTransaction (see page 1769)	This is get_DbTransaction, a member of class TAdoDbxCommand.
	set_DbConnection (see page 1771)	This is set_DbConnection, a member of class TAdoDbxCommand.
	set_DbTransaction (see page 1771)	This is set_DbTransaction, a member of class TAdoDbxCommand.

15.1.1.1.1.2 TAdoDbxCommand.Destroy Destructor

Destroys a TAdoDbxCommand (see page 1763) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdoDbxCommand();
```

Description

This method destroys a TAdoDbxCommand (see page 1763) instance, freeing all its resources, by calling the **Close** method.

Note: Developers should rarely call Destroy directly; calling Dispose (or Free in Delphi) maps the call to the destructor.

See Also

Dispose (MSDN)

15.1.1.1.1.3 TAdoDbxCommand Methods

The methods of the TAdoDbxCommand class are listed here.

Public Methods

	Name	Description
	Cancel (see page 1767)	Attempts to cancel a TAdoDbxCommand (see page 1763).
	Clone (see page 1767)	This is Clone, a member of class TAdoDbxCommand.
	ExecuteNonQuery (see page 1768)	Executes a SQL statement against a connection object.
	ExecuteScalar (see page 1768)	Execute a SQL query and return the first column of the first row of data.
	get_CommandText (see page 1768)	This is get_CommandText, a member of class TAdoDbxCommand.
	get_CommandTimeout (see page 1769)	This is get_CommandTimeout, a member of class TAdoDbxCommand.
	get_CommandType (see page 1769)	This is get_CommandType, a member of class TAdoDbxCommand.
	get_DesignTimeVisible (see page 1770)	This is get_DesignTimeVisible, a member of class TAdoDbxCommand.
	get_UpdatedRowSource (see page 1770)	This is get_UpdatedRowSource, a member of class TAdoDbxCommand.
	Prepare (see page 1770)	Create a compiled version of a command on the data source.
	set_CommandText (see page 1770)	This is set_CommandText, a member of class TAdoDbxCommand.
	set_CommandTimeout (see page 1771)	This is set_CommandTimeout, a member of class TAdoDbxCommand.
	set_CommandType (see page 1771)	This is set_CommandType, a member of class TAdoDbxCommand.
	set_DesignTimeVisible (see page 1771)	This is set_DesignTimeVisible, a member of class TAdoDbxCommand.
	set_UpdatedRowSource (see page 1772)	This is set_UpdatedRowSource, a member of class TAdoDbxCommand.

Strict protected Methods

	Name	Description
	CreateDbParameter (see page 1767)	This is CreateDbParameter, a member of class TAdoDbxCommand.
	ExecuteDbDataReader (see page 1767)	This is ExecuteDbDataReader, a member of class TAdoDbxCommand.
	get_DbConnection (see page 1769)	This is get_DbConnection, a member of class TAdoDbxCommand.

	get_DbParameterCollection (see page 1769)	This is get_DbParameterCollection, a member of class TAdoDbxCommand.
	get_DbTransaction (see page 1769)	This is get_DbTransaction, a member of class TAdoDbxCommand.
	set_DbConnection (see page 1771)	This is set_DbConnection, a member of class TAdoDbxCommand.
	set_DbTransaction (see page 1771)	This is set_DbTransaction, a member of class TAdoDbxCommand.

15.1.1.1.1.3.1 TAdoDbxCommand.Cancel Method

Attempts to cancel a TAdoDbxCommand (see page 1763).

Pascal

```
procedure Cancel; override;
```

C++

```
virtual __fastcall Cancel();
```

Description

Cancel attempts to cancel a TAdoDbxCommand (see page 1763). If a command is in progress, it is cancelled. No exception is generated if the cancel fails. If no command is in progress, nothing happens.

See Also

Cancel (MSDN)

15.1.1.1.1.3.2 TAdoDbxCommand.Clone Method

Pascal

```
function Clone: TObject;
```

C++

```
__fastcall TObject * Clone();
```

Description

This is Clone, a member of class TAdoDbxCommand.

15.1.1.1.1.3.3 TAdoDbxCommand.CreateDbParameter Method

Pascal

```
function CreateDbParameter: DbParameter; override;
```

C++

```
virtual __fastcall DbParameter CreateDbParameter();
```

Description

This is CreateDbParameter, a member of class TAdoDbxCommand.

15.1.1.1.1.3.4 TAdoDbxCommand.ExecuteDbDataReader Method

Pascal

```
function ExecuteDbDataReader(Behavior: CommandBehavior): DbDataReader; override;
```

C++

```
virtual __fastcall DbDataReader ExecuteDbDataReader(CommandBehavior Behavior);
```

Description

This is ExecuteDbDataReader, a member of class TAdoDbxCommand.

15.1.1.1.1.3.5 TAdoDbxCommand.ExecuteNonQuery Method

Executes a SQL statement against a connection object.

Pascal

```
function ExecuteNonQuery: Integer; override;
```

C++

```
virtual __fastcall int ExecuteNonQuery();
```

Description

ExecuteNonQuery does not return data, but does allow you to perform SQL UPDATE, INSERT, or DELETE statements to modify a database. ExecuteNonQuery returns the number of rows affected for executing UPDATE, INSERT, or DELETE statements and -1 otherwise.

See Also

ExecuteNonQuery (MSDN)

ExecuteReader

ExecuteScalar (see page 1768)

15.1.1.1.1.3.6 TAdoDbxCommand.ExecuteScalar Method

Execute a SQL query and return the first column of the first row of data.

Pascal

```
function ExecuteScalar: System; override;
```

C++

```
virtual __fastcall System ExecuteScalar();
```

Description

ExecuteScalar returns only the first column of the first row of data from a SQL query.

Note: ExecuteScalar is useful for running aggregate SQL commands, such as `select sum(field) from table`.

See Also

ExecuteScalar (MSDN)

ExecuteNonQuery (see page 1768)

ExecuteReader

15.1.1.1.1.3.7 TAdoDbxCommand.get_CommandText Method

Pascal

```
function get_CommandText: String; override;
```

C++

```
virtual __fastcall AnsiString get_CommandText();
```

Description

This is get_CommandText, a member of class TAdoDbxCommand.

15.1.1.1.1.3.8 TAdoDbxCommand.get_CommandTimeout Method

Pascal

```
function get_CommandTimeout: Integer; override;
```

C++

```
virtual __fastcall int get_CommandTimeout();
```

Description

This is get_CommandTimeout, a member of class TAdoDbxCommand.

15.1.1.1.1.3.9 TAdoDbxCommand.get_CommandType Method

Pascal

```
function get_CommandType: CommandType; override;
```

C++

```
virtual __fastcall CommandType get_CommandType();
```

Description

This is get_CommandType, a member of class TAdoDbxCommand.

15.1.1.1.1.3.10 TAdoDbxCommand.get_DbConnection Method

Pascal

```
function get_DbConnection: DbConnection; override;
```

C++

```
virtual __fastcall DbConnection get_DbConnection();
```

Description

This is get_DbConnection, a member of class TAdoDbxCommand.

15.1.1.1.1.3.11 TAdoDbxCommand.get_DbParameterCollection Method

Pascal

```
function get_DbParameterCollection: DbParameterCollection; override;
```

C++

```
virtual __fastcall DbParameterCollection get_DbParameterCollection();
```

Description

This is get_DbParameterCollection, a member of class TAdoDbxCommand.

15.1.1.1.1.3.12 TAdoDbxCommand.get_DbTransaction Method

Pascal

```
function get_DbTransaction: DbTransaction; override;
```

C++

```
virtual __fastcall DbTransaction get_DbTransaction();
```

Description

This is get_DbTransaction, a member of class TAdoDbxCommand.

15.1.1.1.1.3.13 TAdoDbxCommand.get_DesignTimeVisible Method

Pascal

```
function get_DesignTimeVisible: Boolean; override;
```

C++

```
virtual __fastcall Boolean get_DesignTimeVisible();
```

Description

This is get_DesignTimeVisible, a member of class TAdoDbxCommand.

15.1.1.1.1.3.14 TAdoDbxCommand.get_UpdatedRowSource Method

Pascal

```
function get_UpdatedRowSource: UpdateRowSource; override;
```

C++

```
virtual __fastcall UpdateRowSource get_UpdatedRowSource();
```

Description

This is get_UpdatedRowSource, a member of class TAdoDbxCommand.

15.1.1.1.1.3.15 TAdoDbxCommand.Prepare Method

Create a compiled version of a command on the data source.

Pascal

```
procedure Prepare; override;
```

C++

```
virtual __fastcall Prepare();
```

Description

Prepare creates a compiled version of a SQL command on the data source. If the CommandType (see page 1773) property is **TableDirect**, Prepare does nothing.

See Also

Prepare (MSDN)

CommandType (see page 1773)

15.1.1.1.1.3.16 TAdoDbxCommand.set_CommandText Method

Pascal

```
procedure set_CommandText(CommandText: String); override;
```

C++

```
virtual __fastcall set_CommandText(AnsiString CommandText);
```

Description

This is set_CommandText, a member of class TAdoDbxCommand.

15.1.1.1.1.3.17 TAdoDbxCommand.set_CommandTimeout Method

Pascal

```
procedure set_CommandTimeout(Timeout: Integer); override;
```

C++

```
virtual __fastcall set_CommandTimeout(int Timeout);
```

Description

This is set_CommandTimeout, a member of class TAdoDbxCommand.

15.1.1.1.1.3.18 TAdoDbxCommand.set_CommandType Method

Pascal

```
procedure set_CommandType(CommandType: CommandType); override;
```

C++

```
virtual __fastcall set_CommandType(CommandType CommandType);
```

Description

This is set_CommandType, a member of class TAdoDbxCommand.

15.1.1.1.1.3.19 TAdoDbxCommand.set_DbConnection Method

Pascal

```
procedure set_DbConnection(Connection: DbConnection); override;
```

C++

```
virtual __fastcall set_DbConnection(DbConnection Connection);
```

Description

This is set_DbConnection, a member of class TAdoDbxCommand.

15.1.1.1.1.3.20 TAdoDbxCommand.set_DbTransaction Method

Pascal

```
procedure set_DbTransaction(Transaction: DbTransaction); override;
```

C++

```
virtual __fastcall set_DbTransaction(DbTransaction Transaction);
```

Description

This is set_DbTransaction, a member of class TAdoDbxCommand.

15.1.1.1.1.3.21 TAdoDbxCommand.set_DesignTimeVisible Method

Pascal

```
procedure set_DesignTimeVisible(DesignTimeVisible: Boolean); override;
```

C++

```
virtual __fastcall set_DesignTimeVisible(Boolean DesignTimeVisible);
```

Description

This is set_DesignTimeVisible, a member of class TAdoDbxCommand.

15.1.1.1.1.3.22 TAdoDbxCommand.set_UpdatedRowSource Method

Pascal

```
procedure set_UpdatedRowSource(UpdatedRowSource: UpdateRowSource); override;
```

C++

```
virtual __fastcall set_UpdatedRowSource(UpdateRowSource UpdatedRowSource);
```

Description

This is set_UpdatedRowSource, a member of class TAdoDbxCommand.

15.1.1.1.1.4 TAdoDbxCommand Properties

The properties of the TAdoDbxCommand class are listed here.

Public Properties

	Name	Description
	CommandText (see page 1772)	Gets or sets the SQL statement, table name, or stored procedure to execute at the data source.
	CommandTimeout (see page 1773)	Gets or sets the time, in seconds, before terminating a long-running SQL command.
	CommandType (see page 1773)	Gets or sets the CommandType that indicates how to interpret the CommandText (see page 1772) property.
	DesignTimeVisible (see page 1774)	Determines whether or not a command is visible at design time.
R	Parameters (see page 1774)	Returns the TAdoDbxParameterCollection (see page 1865).
	Transaction (see page 1774)	Provides the transaction in which the TAdoDbxCommand (see page 1763) executes.
	UpdateRowSource (see page 1775)	This is UpdateRowSource, a member of class TAdoDbxCommand.

15.1.1.1.1.4.1 TAdoDbxCommand.CommandText Property

Gets or sets the SQL statement, table name, or stored procedure to execute at the data source.

Pascal

```
property CommandText: String;
```

C++

```
__property AnsiString CommandText;
```

Description

When you set the CommandType (see page 1773) property to *StoredProcedure*, you set the CommandText property to the name of the stored procedure. You might need to use escape character syntax if your stored procedure name contains special characters, such as underscores. The command executes this stored procedure when you call either the ExecuteReader, ExecuteScalar (see page 1768), or ExecuteNonQuery (see page 1768) methods. When you set the CommandType (see page 1773) property to *TableDirect*, you set the CommandText property to the name of the table you want to access. You might need to use escape character syntax if any of the tables contain special characters.

See Also

CommandText (MSDN)

Building a VCL Forms dbExpress.NET Database Application

CommandType (see page 1773)

ExecuteNonQuery (see page 1768)

ExecuteReader

ExecuteScalar (see page 1768)

15.1.1.1.1.4.2 TAdoDbxCommand.CommandTimeout Property

Gets or sets the time, in seconds, before terminating a long-running SQL command.

Pascal

```
property CommandTimeout: Integer;
```

C++

```
__property int CommandTimeout;
```

Description

CommandTimeout gets or set the wait time in seconds before terminating a command and generating an error.

See Also

CommandTimeout (MSDN)

15.1.1.1.1.4.3 TAdoDbxCommand.CommandType Property

Gets or sets the CommandType that indicates how to interpret the CommandText (see page 1772) property.

Pascal

```
property CommandType: CommandType;
```

C++

```
__property CommandType CommandType;
```

Description

When you set the CommandType property to **StoredProcedure**, you should set the CommandText (see page 1772) property to the name of the stored procedure. The command executes this stored procedure when you call the ExecuteScalar (see page 1768), ExecuteReader, or ExecuteNonQuery (see page 1768) methods. The Connection, CommandType and CommandText (see page 1772) properties cannot be set if the current connection is performing an execute or fetch operation.

AdoDbxClient supports the ? symbol as a placeholder for passing parameters to a SQL statement or a stored procedure called by a TAdoDbxCommand (see page 1763) when CommandType is set to **Text**.

Type	Description
Text	A SQL command. Default.
TableDirect	If you set the CommandType property to **TableDirect**, set the CommandText (see page 1772) property to the name of the table you want to access. You might need to use escape characters or other qualifying characters if the tables contain special characters. When you call one of the Execute methods, the query returns all rows and columns.
StoredProcedure	The name of a stored procedure.

See Also

CommandType (MSDN)

CommandText (see page 1772)

ExecuteNonQuery (see page 1768)

ExecuteReader

ExecuteScalar (see page 1768)

15.1.1.1.1.4.4 TAdoDbxCommand.DesignTimeVisible Property

Determines whether or not a command is visible at design time.

Pascal

```
property DesignTimeVisible: Boolean;
```

C++

```
__property Boolean DesignTimeVisible;
```

Description

A boolean that can be set on a TAdoDbxCommand (see page 1763) to determine the command's visibility during design time.

Note: DesignTimeVisible is generally not of interest to end users, but to component writers extending ADO.NET providers.

See Also

DesignTimeVisible (MSDN)

15.1.1.1.1.4.5 TAdoDbxCommand.Parameters Property

Returns the TAdoDbxParameterCollection (see page 1865).

Pascal

```
property Parameters: DbParameterCollection;
```

C++

```
__property DbParameterCollection Parameters;
```

Description

The parameters of the SQL statement or stored procedure. The default is an empty collection.

You cannot use named parameters for passing parameters to a SQL Statement or a stored procedure called by a TAdoDbxCommand (see page 1763) when CommandType (see page 1773) is set to **Text**. In this case, the question mark (?) placeholder must be used. For example: `SELECT * FROM Employee WHERE EmployeeID = ?`. The order in which TAdoDbxParameter (see page 1850) objects are added to the TAdoDbxParameterCollection (see page 1865) must directly correspond to the position of the question mark placeholder for the parameter.

If the parameters in the collection do not match the requirements of the query to be executed, you may get an error.

See Also

Connection (MSDN)

TAdoDbxCommand (see page 1763)

TAdoDbxParameter (see page 1850)

TAdoDbxParameterCollection (see page 1865)

CommandType (see page 1773)

15.1.1.1.1.4.6 TAdoDbxCommand.Transaction Property

Provides the transaction in which the TAdoDbxCommand (see page 1763) executes.

Pascal

```
property Transaction: DbTransaction;
```

C++

```
__property DbTransaction Transaction;
```

Description

Transaction provides the transaction the command executes within. The transaction must already have been created prior to naming it with the Transaction property on the TAdoDbxCommand (see page 1763).

Warning: You cannot set the Transaction property if it is already set to a specific value and the command is in the process of executing. If you set the transaction property to a TAdoDbxTransaction (see page 1884) object that is not connected to the same TAdoDbxConnection (see page 1779) as the TAdoDbxCommand (see page 1763) object, an exception is thrown the next time you attempt to execute a statement.

```
connection.Open();
TAdoDbxTransaction transaction = connection.BeginTransaction();
string textSQL = "INSERT INTO Employee (...) VALUES (...)";
TAdoDbxCommand command = new TAdoDbxCommand(textSQL, connection, transaction);
int intRecordsAffected = command.ExecuteNonQuery();
if (intRecordsAffected == 1)
{
    MessageBox.Show("Row updated.");
    transaction.Commit();
}
else
{
    // No records affected
    MessageBox.Show("No row updated.");
    transaction.Rollback();
}
connection.Open();
TTrans: TTransaction.BeginTransaction(connection);
string textSQL = "INSERT INTO Employee (...) VALUES (...)";
TAdoDbxCommand command = new TAdoDbxCommand(textSQL, connection, TTrans);
int intRecordsAffected = command.ExecuteNonQuery();
if (intRecordsAffected == 1)
{
    MessageBox.Show("Row updated.");
    TTrans.Commit();
}
else
{
    // No records affected
    MessageBox("No row updated.", [smbOK]);
    TTrans.Rollback();
}
```

See Also

Transaction (MSDN)

TAdoDbxCommand (see page 1763)

TAdoDbxConnection (see page 1779)

TAdoDbxTransaction (see page 1884)

15.1.1.1.1.4.7 TAdoDbxCommand.UpdateRowSource Property

Pascal

```
property UpdateRowSource: UpdateRowSource;
```

C++

```
__property UpdateRowSource UpdateRowSource;
```

Description

This is UpdateRowSource, a member of class TAdoDbxCommand.

15.1.1.1.2 TAdoDbxCommandBuilder Class

Generates single-table commands as part of the operation of the TAdoDbxDataAdapter (see page 1814).

Class Hierarchy

Pascal

```
TAdoDbxCommandBuilder = class(DbCommandBuilder);
```

C++

```
class TAdoDbxCommandBuilder : public DbCommandBuilder;
```

File

Borland.Data.AdoDbxClientProvider

Description

The TAdoDbxCommandBuilder class is used to automatically generate single-table commands to reconcile changes made to a DataSet with its underlying database.

TAdoDbxCommandBuilder implements the ADO.NET DbCommandBuilder class.

You can generate SQL commands based on the SelectCommand property, which you must set prior to executing any of the other TAdoDbxCommandBuilder methods. The results of the SelectCommand are used to return metadata that is then used to construct the arguments to the InsertCommand, UpdateCommand, or DeleteCommand properties of TAdoDbxDataAdapter (see page 1814). If you want to achieve greater performance, you might want to consider using direct commands, rather than using the TAdoDbxCommandBuilder.

If you review the SQL statements generated for the UpdateCommand and DeleteCommand, you see that they each contain a `WHERE` clause that specifies that the row can only be updated if it contains all original values and hasn't been deleted from the database. Records are not locked and therefore the TAdoDbxCommandBuilder operates on the basis of optimistic concurrency. If you try to update or delete a record that has been changed by another user, the operation throws a DBConcurrencyException.

Note: You can only use the TAdoDbxCommandBuilder on standalone tables. If you try to update a column that participates in a foreign-key constraint, you might encounter a failure.

Warning: SelectCommand must return at least one primary key or unique column.

See Also

DbCommandBuilder (MSDN)

TAdoDbxDataAdapter (see page 1814)

DeleteCommand (see page 1823)

InsertCommand (see page 1824)

SelectCommand (see page 1825)

UpdateCommand (see page 1825)

15.1.1.1.2.1 TAdoDbxCommandBuilder Members

The following tables list the members exposed by TAdoDbxCommandBuilder.

Public Methods

	Name	Description
	Create (see page 1777)	This is the overview for the Create constructor overload.

Strict protected Methods

	Name	Description
	AdoDbxRowUpdatingHandler (see page 1778)	This is AdoDbxRowUpdatingHandler, a member of class TAdoDbxCommandBuilder.
V	ApplyParameterInfo (see page 1778)	This is ApplyParameterInfo, a member of class TAdoDbxCommandBuilder.
V	GetParameterName (see page 1778)	This is the overview for the GetParameterName method overload.
V	GetParameterPlaceholder (see page 1779)	This is GetParameterPlaceholder, a member of class TAdoDbxCommandBuilder.
V	SetRowUpdatingHandler (see page 1779)	This is SetRowUpdatingHandler, a member of class TAdoDbxCommandBuilder.

15.1.1.1.2.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TAdoDbxCommandBuilder.Create () (see page 1777)	Creates an instance of TAdoDbxCommandBuilder (see page 1776)
	TAdoDbxCommandBuilder.Create (TAdoDbxDataAdapter) (see page 1777)	Creates an instance of TAdoDbxCommandBuilder (see page 1776)

15.1.1.1.2.2.1 TAdoDbxCommandBuilder.Create Constructor ()

Creates an instance of TAdoDbxCommandBuilder (see page 1776)

Pascal

```
constructor Create; overload;
constructor Create(Adapter: TAdoDbxDataAdapter); overload;
```

C++

```
__fastcall TAdoDbxCommandBuilder();
__fastcall TAdoDbxCommandBuilder(TAdoDbxDataAdapter Adapter);
```

Description

This constructor creates and initializes an instance of TAdoDbxCommandBuilder (see page 1776).

See Also

Create (MSDN)

15.1.1.1.2.3 TAdoDbxCommandBuilder Methods

The methods of the TAdoDbxCommandBuilder class are listed here.

Strict protected Methods

	Name	Description
	AdoDbxRowUpdatingHandler (see page 1778)	This is AdoDbxRowUpdatingHandler, a member of class TAdoDbxCommandBuilder.

	ApplyParameterInfo (see page 1778)	This is ApplyParameterInfo, a member of class TAdoDbxCommandBuilder.
	GetParameterName (see page 1778)	This is the overview for the GetParameterName method overload.
	GetParameterPlaceholder (see page 1779)	This is GetParameterPlaceholder, a member of class TAdoDbxCommandBuilder.
	SetRowUpdatingHandler (see page 1779)	This is SetRowUpdatingHandler, a member of class TAdoDbxCommandBuilder.

15.1.1.1.2.3.1 TAdoDbxCommandBuilder.AdoDbxRowUpdatingHandler Method

Pascal

```
procedure AdoDbxRowUpdatingHandler(Sender: System.Object; e: TAdoDbxRowUpdatingEventArgs);
```

C++

```
__fastcall AdoDbxRowUpdatingHandler(System.Object Sender, TAdoDbxRowUpdatingEventArgs e);
```

Description

This is AdoDbxRowUpdatingHandler, a member of class TAdoDbxCommandBuilder.

15.1.1.1.2.3.2 TAdoDbxCommandBuilder.ApplyParameterInfo Method

Pascal

```
procedure ApplyParameterInfo(parameter: DbParameter; row: DataRow; statementType:
StatementType; whereClause: Boolean); override;
```

C++

```
virtual __fastcall ApplyParameterInfo(DbParameter parameter, DataRow row, StatementType
statementType, Boolean whereClause);
```

Description

This is ApplyParameterInfo, a member of class TAdoDbxCommandBuilder.

15.1.1.1.2.3.3 GetParameterName Method

This is the overview for the GetParameterName method overload.

Overload List

	Name	Description
	TAdoDbxCommandBuilder.GetParameterName (Integer) (see page 1778)	This is GetParameterName, a member of class TAdoDbxCommandBuilder.
	TAdoDbxCommandBuilder.GetParameterName (string) (see page 1779)	This is GetParameterName, a member of class TAdoDbxCommandBuilder.

15.1.1.1.2.3.3.1 TAdoDbxCommandBuilder.GetParameterName Method (Integer)

Pascal

```
function GetParameterName(parameterOrdinal: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetParameterName(int parameterOrdinal);
```

Description

This is GetParameterName, a member of class TAdoDbxCommandBuilder.

15.1.1.1.2.3.3.2 TAdoDbxCommandBuilder.GetParameterName Method (string)

Pascal

```
function GetParameterName(parameterName: string): string; override;
```

C++

```
virtual __fastcall AnsiString GetParameterName(AnsiString parameterName);
```

Description

This is GetParameterName, a member of class TAdoDbxCommandBuilder.

15.1.1.1.2.3.4 TAdoDbxCommandBuilder.GetParameterPlaceholder Method

Pascal

```
function GetParameterPlaceholder(parameterOrdinal: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetParameterPlaceholder(int parameterOrdinal);
```

Description

This is GetParameterPlaceholder, a member of class TAdoDbxCommandBuilder.

15.1.1.1.2.3.5 TAdoDbxCommandBuilder.SetRowUpdatingHandler Method

Pascal

```
procedure SetRowUpdatingHandler(adapter: DbDataAdapter); override;
```

C++

```
virtual __fastcall SetRowUpdatingHandler(DbDataAdapter adapter);
```

Description

This is SetRowUpdatingHandler, a member of class TAdoDbxCommandBuilder.

15.1.1.1.3 TAdoDbxConnection Class

Represents a connection to a database.

Class Hierarchy

Pascal

```
TAdoDbxConnection = class(DbConnection, ICloneable);
```

C++

```
class TAdoDbxConnection : public DbConnection, public ICloneable;
```

File

Borland.Data.AdoDbxClientProvider

Description

TAdoDbxConnection represents a connection to a database. Upon creation of the object instance, all properties are set to their initial values. You must explicitly close any open connection using the Close (see page 1785) method. There is no notion of an implicit disconnection in the .NET environment.

TAdoDbxConnection is an implementation of the ADO.NET DbConnection class.

To establish a database connection, you must create a new TAdoDbxConnection object and you must set its ConnectionString (see page 1788) property. The ConnectionString (see page 1788) is a name-value pair of all the parameters needed to connect to a particular database.

TAdoDbxConnection has Open (see page 1787) and Close (see page 1785) methods to connect and disconnect from a database. The Open (see page 1787) method uses the connection parameters specified in the ConnectionString (see page 1788) property to establish a connection to a database. For every Open (see page 1787) method, you should include a corresponding Close (see page 1785) method. Otherwise, the connection might not be closed properly and any subsequent connection attempt will fail with an exception.

A valid connection must be opened before you can use the TAdoDbxConnection to create a new TAdoDbxCommand (see page 1763) or call BeginTransaction to create a new TAdoDbxTransaction (see page 1884) instance.

The following Delphi code shows the ConnectionString (see page 1788) needed to connect to InterBase.

```
// Here is a simple way to connect to an “IBCONNECTION” connection from the dbxconnections.ini file:

    Factory := System.Data.Common.DbProviderFactories.GetFactory('Borland.Data.AdoDbxClient');
    Connection := Factory.CreateConnection();
    Connection.ConnectionString := 'ConnectionName=' + ‘IBCONNECTION’;
    Connection.Open;
```

See Also

DbConnection (MSDN)

ADO.NET Overview

Building an ASP.NET Database Application

TAdoDbxCommand (see page 1763)

TAdoDbxTransaction (see page 1884)

ConnectionString (see page 1788)

BeginTransaction

Close (see page 1785)

Open (see page 1787)

15.1.1.1.3.1 TAdoDbxConnection Members

The following tables list the members exposed by TAdoDbxConnection.

Public Methods

	Name	Description
	ChangeDatabase (see page 1784)	Changes the current database for a connection.
	Clone (see page 1785)	This is Clone, a member of class TAdoDbxConnection.
	Close (see page 1785)	Close database connection.
	Create (see page 1781)	This is the overview for the Create constructor overload.
	Destroy (see page 1782)	Destroys a TAdoDbxConnection (see page 1779) instance.
	get_ConnectionString (see page 1785)	This is get_ConnectionString, a member of class TAdoDbxConnection.
	get_ConnectionTimeout (see page 1786)	This is get_ConnectionTimeout, a member of class TAdoDbxConnection.
	get_Database (see page 1786)	This is get_Database, a member of class TAdoDbxConnection.
	get_DataSource (see page 1786)	This is get_DataSource, a member of class TAdoDbxConnection.
	get_ServerVersion (see page 1786)	This is get_ServerVersion, a member of class TAdoDbxConnection.
	get_State (see page 1786)	This is get_State, a member of class TAdoDbxConnection.

V	GetSchema (see page 1787)	This is the overview for the GetSchema method overload.
V	Open (see page 1787)	Opens a connection to a database using the connection string parameters.
V	set_ConnectionString (see page 1788)	This is set_ConnectionString, a member of class TAdoDbxConnection.

Public Properties

	Name	Description
	ConnectionString (see page 1788)	Contains the property settings necessary to connect to a database.
R	ConnectionTimeout (see page 1789)	Returns the time to wait while trying to establish a connection.
R	Database (see page 1789)	Contains the name of the current database or the database to be connected.
R	DataSource (see page 1789)	Name of database server to which to connect.
R	Dbx (see page 1790)	This is Dbx, a member of class TAdoDbxConnection.
R	ServerVersion (see page 1790)	Server version of connected database
R	State (see page 1790)	Gets or sets the current state of the connection.

Strict private Fields

	Name	Description
	FConnectionString (see page 1782)	This is FConnectionString, a member of class TAdoDbxConnection.
	FConnectionTimeout (see page 1783)	This is FConnectionTimeout, a member of class TAdoDbxConnection.
	FDbxExtension (see page 1783)	This is FDbxExtension, a member of class TAdoDbxConnection.
	FDriverName (see page 1783)	This is FDriverName, a member of class TAdoDbxConnection.
	FSchemaItems (see page 1783)	This is FSchemaItems, a member of class TAdoDbxConnection.

Strict private Methods

	Name	Description
	AdoDbxIsolation (see page 1784)	This is AdoDbxIsolation, a member of class TAdoDbxConnection.

Strict protected Methods

	Name	Description
V	BeginDbTransaction (see page 1784)	This is BeginDbTransaction, a member of class TAdoDbxConnection.
V	CreateDbCommand (see page 1785)	This is CreateDbCommand, a member of class TAdoDbxConnection.

15.1.1.1.3.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TAdoDbxConnection.Create () (see page 1781)	Creates a new instance of a TAdoDbxConnection (see page 1779) object, which represents an open connection to a specified data source.
	TAdoDbxConnection.Create (string) (see page 1781)	Creates a new instance of a TAdoDbxConnection (see page 1779) object, which represents an open connection to a specified data source.

15.1.1.1.3.2.1 TAdoDbxConnection.Create Constructor ()

Creates a new instance of a TAdoDbxConnection (see page 1779) object, which represents an open connection to a specified data source.

Pascal

```
constructor Create; overload;
constructor Create(const ConnectionString: string); overload;
```

C++

```
__fastcall TAdoDbxConnection();
__fastcall TAdoDbxConnection(const AnsiString ConnectionString);
```

Description

The TAdoDbxConnection (see page 1779) object represents a connection to a specified data source. The TAdoDbxConnection (see page 1779) object contains the connection string, along with other properties. You need to explicitly open and close connections using the Open (see page 1787) and Close (see page 1785) methods, respectively. There is no notion in the .NET framework of an implicit open or close operation.

See Also

Create (MSDN)

Close (see page 1785)

Open (see page 1787)

15.1.1.1.3.3 TAdoDbxConnection.Destroy Destructor

Destroys a TAdoDbxConnection (see page 1779) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdoDbxConnection();
```

Description

This method destroys a TAdoDbxConnection (see page 1779) instance, freeing all its resources.

Note: This method should not be called directly. Call either the Close (see page 1785) or Free method instead.

See Also

Close (see page 1785)

Create (see page 1781)

15.1.1.1.3.4 TAdoDbxConnection Fields

The fields of the TAdoDbxConnection class are listed here.

Strict private Fields

	Name	Description
	FConnectionString (see page 1782)	This is FConnectionString, a member of class TAdoDbxConnection.
	FConnectionTimeout (see page 1783)	This is FConnectionTimeout, a member of class TAdoDbxConnection.
	FDbxExtension (see page 1783)	This is FDbxExtension, a member of class TAdoDbxConnection.
	FDriverName (see page 1783)	This is FDriverName, a member of class TAdoDbxConnection.
	FSchemaItems (see page 1783)	This is FSchemaItems, a member of class TAdoDbxConnection.

15.1.1.1.3.4.1 TAdoDbxConnection.FConnectionString Field

Pascal

```
class var FConnectionString: String;
```

C++

```
AnsiString FConnectionString;
```

Description

This is FConnectionString, a member of class TAdoDbxConnection.

15.1.1.1.3.4.2 TAdoDbxConnection.FConnectionTimeout Field

Pascal

```
class var FConnectionTimeout: Integer;
```

C++

```
int FConnectionTimeout;
```

Description

This is FConnectionTimeout, a member of class TAdoDbxConnection.

15.1.1.1.3.4.3 TAdoDbxConnection.FDbxExtension Field

Pascal

```
class var FDbxExtension: TAdoDbxExtension;
```

C++

```
TAdoDbxExtension FDbxExtension;
```

Description

This is FDbxExtension, a member of class TAdoDbxConnection.

15.1.1.1.3.4.4 TAdoDbxConnection.FDriverName Field

Pascal

```
class var FDriverName: String;
```

C++

```
AnsiString FDriverName;
```

Description

This is FDriverName, a member of class TAdoDbxConnection.

15.1.1.1.3.4.5 TAdoDbxConnection.FSchemaItems Field

Pascal

```
class var FSchemaItems: TStringList;
```

C++

```
TStringList FSchemaItems;
```

Description

This is FSchemaItems, a member of class TAdoDbxConnection.

15.1.1.1.3.5 TAdoDbxConnection Methods

The methods of the TAdoDbxConnection class are listed here.

Public Methods

	Name	Description
	ChangeDatabase (see page 1784)	Changes the current database for a connection.

	Clone (see page 1785)	This is Clone, a member of class TAdoDbxConnection.
	Close (see page 1785)	Close database connection.
	get_ConnectionString (see page 1785)	This is get_ConnectionString, a member of class TAdoDbxConnection.
	get_ConnectionTimeout (see page 1786)	This is get_ConnectionTimeout, a member of class TAdoDbxConnection.
	get_Database (see page 1786)	This is get_Database, a member of class TAdoDbxConnection.
	get_DataSource (see page 1786)	This is get_DataSource, a member of class TAdoDbxConnection.
	get_ServerVersion (see page 1786)	This is get_ServerVersion, a member of class TAdoDbxConnection.
	get_State (see page 1786)	This is get_State, a member of class TAdoDbxConnection.
	GetSchema (see page 1787)	This is the overview for the GetSchema method overload.
	Open (see page 1787)	Opens a connection to a database using the connection string parameters.
	set_ConnectionString (see page 1788)	This is set_ConnectionString, a member of class TAdoDbxConnection.

Strict private Methods

	Name	Description
	AdoDbxIsolation (see page 1784)	This is AdoDbxIsolation, a member of class TAdoDbxConnection.

Strict protected Methods

	Name	Description
	BeginDbTransaction (see page 1784)	This is BeginDbTransaction, a member of class TAdoDbxConnection.
	CreateDbCommand (see page 1785)	This is CreateDbCommand, a member of class TAdoDbxConnection.

15.1.1.1.3.5.1 TAdoDbxConnection.AdoDbxIsolation Method

Pascal

```
function AdoDbxIsolation(Isolation: IsolationLevel): TDBXIsolation;
```

C++

```
__fastcall TDBXIsolation AdoDbxIsolation(IsolationLevel Isolation);
```

Description

This is AdoDbxIsolation, a member of class TAdoDbxConnection.

15.1.1.1.3.5.2 TAdoDbxConnection.BeginDbTransaction Method

Pascal

```
function BeginDbTransaction(Isolation: IsolationLevel): DbTransaction; override;
```

C++

```
virtual __fastcall DbTransaction BeginDbTransaction(IsolationLevel Isolation);
```

Description

This is BeginDbTransaction, a member of class TAdoDbxConnection.

15.1.1.1.3.5.3 TAdoDbxConnection.ChangeDatabase Method

Changes the current database for a connection.

Pascal

```
procedure ChangeDatabase(DatabaseName: String); override;
```

C++

```
virtual __fastcall ChangeDatabase(AnsiString DatabaseName);
```

Description

ChangeDatabase switches the connection to a specified database.

See Also

ChangeDatabase (MSDN)

15.1.1.1.3.5.4 TAdoDbxConnection.Clone Method

Pascal

```
function Clone: TObject;
```

C++

```
__fastcall TObject * Clone();
```

Description

This is Clone, a member of class TAdoDbxConnection.

15.1.1.1.3.5.5 TAdoDbxConnection.Close Method

Close database connection.

Pascal

```
procedure Close; override;
```

C++

```
virtual __fastcall Close();
```

Description

Close rolls back any pending transactions and releases the connection to the connection pool or closes the connection if there is no pool. Calling Close more than once does not raise an exception.

See Also

Close (MSDN)

15.1.1.1.3.5.6 TAdoDbxConnection.CreateDbCommand Method

Pascal

```
function CreateDbCommand: DbCommand; override;
```

C++

```
virtual __fastcall DbCommand CreateDbCommand();
```

Description

This is CreateDbCommand, a member of class TAdoDbxConnection.

15.1.1.1.3.5.7 TAdoDbxConnection.get_ConnectionString Method

Pascal

```
function get_ConnectionString: String; override;
```

C++

```
virtual __fastcall AnsiString get_ConnectionString();
```

Description

This is get_ConnectionString, a member of class TAdoDbxConnection.

15.1.1.1.3.5.8 TAdoDbxConnection.get_ConnectionTimeout Method

Pascal

```
function get_ConnectionTimeout: Integer; override;
```

C++

```
virtual __fastcall int get_ConnectionTimeout();
```

Description

This is get_ConnectionTimeout, a member of class TAdoDbxConnection.

15.1.1.1.3.5.9 TAdoDbxConnection.get_Database Method

Pascal

```
function get_Database: String; override;
```

C++

```
virtual __fastcall AnsiString get_Database();
```

Description

This is get_Database, a member of class TAdoDbxConnection.

15.1.1.1.3.5.10 TAdoDbxConnection.get_DataSource Method

Pascal

```
function get_DataSource: String; override;
```

C++

```
virtual __fastcall AnsiString get_DataSource();
```

Description

This is get_DataSource, a member of class TAdoDbxConnection.

15.1.1.1.3.5.11 TAdoDbxConnection.get_ServerVersion Method

Pascal

```
function get_ServerVersion: String; override;
```

C++

```
virtual __fastcall AnsiString get_ServerVersion();
```

Description

This is get_ServerVersion, a member of class TAdoDbxConnection.

15.1.1.1.3.5.12 TAdoDbxConnection.get_State Method

Pascal

```
function get_State: ConnectionState; override;
```

C++

```
virtual __fastcall ConnectionState get_State();
```

Description

This is get_State, a member of class TAdoDbxConnection.

15.1.1.1.3.5.13 GetSchema Method

This is the overview for the GetSchema method overload.

Overload List

	Name	Description
	TAdoDbxConnection.GetSchema () (see page 1787)	Gets schema information for the connected data source.
	TAdoDbxConnection.GetSchema (String) (see page 1787)	Gets schema information for the connected data source.
	TAdoDbxConnection.GetSchema (String, array of String) (see page 1787)	Gets schema information for the connected data source.

15.1.1.1.3.5.13.1 TAdoDbxConnection.GetSchema Method ()

Gets schema information for the connected data source.

Pascal

```
function GetSchema: DataTable; override;
function GetSchema(Name: String): DataTable; override;
function GetSchema(Name: String; Restrictions: array of String): DataTable; override;
```

C++

```
virtual __fastcall DataTable GetSchema();
virtual __fastcall DataTable GetSchema(AnsiString Name);
virtual __fastcall DataTable GetSchema(AnsiString Name, array of String Restrictions);
```

Description

GetSchema obtains a DataTable with schema information for the connected data source.

The Name parameter specifies the kind of metadata to get. The standard names supported by DbConnection.GetSchema are supported.

In addition, you can specify one of the name constants in the DBXMetaDataNames (see page 10685) class TDBXMetaDataCollectionName to get a particular metadata collection. You would then use the corresponding class in the DBXMetaDataNames (see page 10685) namespace to get the column information you want for that metadata collection.

For instance, to get procedure source code in a database, use the constant ProcedureSources to get a DataTable with procedure source information. Use the DBXMetaDataNames (see page 10685) classes TDBXProcedureSourcesColumns and TDBXProcedureSourcesIndex with the returned DataTable to access the procedure source information by name or ordinal.

See Also

GetSchema (MSDN)

TDBXMetaDataCollectionName (see page 10758)

TDBXProcedureSourcesColumns (see page 10807)

TDBXProcedureSourcesIndex (see page 10810)

15.1.1.1.3.5.14 TAdoDbxConnection.Open Method

Opens a connection to a database using the connection string parameters.

Pascal

```
procedure Open; override;
```

C++

```
virtual __fastcall Open();
```

Description

Open opens a connection to a database using the ConnectionString (see page 1788) parameters. The ConnectionString (see page 1788) property must be set before opening a connection. You must separate keyword value pairs by semicolons (;). Names are not case sensitive. Most applications only need to set the ConnectionName name property, which identifies a collection of properties stored in a dbxconnections.ini file. Alternatively, an application can specify all properties from the dbxconnections.ini and dbxdriver.ini file for the database being connected to.

See Also

Open (MSDN)

ConnectionString (see page 1788)

15.1.1.1.3.5.15 TAdoDbxConnection.set_ConnectionString Method

Pascal

```
procedure set_ConnectionString(ConnectionString: String); override;
```

C++

```
virtual __fastcall set_ConnectionString(AnsiString ConnectionString);
```

Description

This is set_ConnectionString, a member of class TAdoDbxConnection.

15.1.1.1.3.6 TAdoDbxConnection Properties

The properties of the TAdoDbxConnection class are listed here.

Public Properties

	Name	Description
	ConnectionString (see page 1788)	Contains the property settings necessary to connect to a database.
R	ConnectionTimeout (see page 1789)	Returns the time to wait while trying to establish a connection.
R	Database (see page 1789)	Contains the name of the current database or the database to be connected.
R	DataSource (see page 1789)	Name of database server to which to connect.
R	Dbx (see page 1790)	This is Dbx, a member of class TAdoDbxConnection.
R	ServerVersion (see page 1790)	Server version of connected database
R	State (see page 1790)	Gets or sets the current state of the connection.

15.1.1.1.3.6.1 TAdoDbxConnection.ConnectionString Property

Contains the property settings necessary to connect to a database.

Pascal

```
property ConnectionString: String;
```

C++

```
__property AnsiString ConnectionString;
```

Description

The ConnectionString property must be set before opening a connection. These keyword-value pairs must be separated by semicolons (;). You can delimit values with single or double quotes. Names are not case sensitive.

Most applications only need to set the ConnectionName property, which identifies a collection of properties stored in a

dbxconnections.ini file. Alternatively, an application can specify all properties from the dbxconnections.ini and dbxdriver.ini file for the database being connected to.

Many of the connection string values have corresponding read-only properties.

See Also

ConnectionString (MSDN)

ConnectionName (see page 1806)

15.1.1.1.3.6.2 TAdoDbxConnection.ConnectionTimeout Property

Returns the time to wait while trying to establish a connection.

Pascal

```
property ConnectionTimeout: Integer;
```

C++

```
__property int ConnectionTimeout;
```

Description

ConnectionTimeout contains the number of seconds to wait while trying to establish a connection before terminating the attempt and generating an error.

See Also

ConnectionTimeout (MSDN)

15.1.1.1.3.6.3 TAdoDbxConnection.Database Property

Contains the name of the current database or the database to be connected.

Pascal

```
property Database: String;
```

C++

```
__property AnsiString Database;
```

Description

Database contains the name of the current database or the database to be connected. You can get or set the database name.

See Also

Database (MSDN)

15.1.1.1.3.6.4 TAdoDbxConnection.DataSource Property

Name of database server to which to connect.

Pascal

```
property DataSource: String;
```

C++

```
__property AnsiString DataSource;
```

Description

DataSource holds the name of database server to which to connect.

See Also

DataSource (MSDN)

15.1.1.1.3.6.5 TAdoDbxConnection.Dbx Property

Pascal

```
property Dbx: TAdoDbxExtension;
```

C++

```
__property TAdoDbxExtension Dbx;
```

Description

This is Dbx, a member of class TAdoDbxConnection.

15.1.1.1.3.6.6 TAdoDbxConnection.ServerVersion Property

Server version of connected database

Pascal

```
property ServerVersion: String;
```

C++

```
__property AnsiString ServerVersion;
```

Description

ServerVersion contains a string representing the server version of connected database.

See Also

ServerVersion (MSDN)

15.1.1.1.3.6.7 TAdoDbxConnection.State Property

Gets or sets the current state of the connection.

Pascal

```
property State: ConnectionState;
```

C++

```
__property ConnectionState State;
```

Description

State contains the current connection state value. The format of the string returned depends on the specific type of connection you are using.

See Also

State (MSDN)

15.1.1.1.4 TAdoDbxConnectionStringBuilder Class

Class for strongly typed connection string builder.

Class Hierarchy

Pascal

```
TAdoDbxConnectionStringBuilder = class(DbConnectionStringBuilder);
```

C++

```
class TAdoDbxConnectionStringBuilder : public DbConnectionStringBuilder;
```

File

Borland.Data.AdoDbxClientProvider

Description

The TAdoDbxConnectionStringBuilder class provides a mechanism for building connection strings in a strongly typed manner. Connection string builders create and edit syntactically correct connection strings.

TAdoDbxConnectionStringBuilder is an implementation of the ADO.NET DbConnectionStringBuilder class.

Users can derive classes from this base class to assign arbitrary key-value pairs that can be tailored for a particular provider. Applications that need to provide connection strings can use TAdoDbxConnectionStringBuilder or one of its derived classes to create and modify connection strings.

This class makes no checks on strings, so it is possible to build invalid connection strings.

The properties in TAdoDbxConnectionStringBuilder correspond to the constants in DBXCommon.TDBXPropertyNames (see page 10566).

See Also

DbConnectionStringBuilder (MSDN)

TDBXPropertyNames (see page 10566)

15.1.1.1.4.1 TAdoDbxConnectionStringBuilder Members

The following tables list the members exposed by TAdoDbxConnectionStringBuilder.

Public Methods

	Name	Description
	Create (see page 1793)	Creates an instance of TAdoDbxConnectionStringBuilder (see page 1790)
	get_ConnectionName (see page 1795)	This is get_ConnectionName, a member of class TAdoDbxConnectionStringBuilder.
	get_Database (see page 1795)	This is get_Database, a member of class TAdoDbxConnectionStringBuilder.
	get_DelegateConnection (see page 1795)	This is get_DelegateConnection, a member of class TAdoDbxConnectionStringBuilder.
	get_DelegateDriver (see page 1795)	This is get_DelegateDriver, a member of class TAdoDbxConnectionStringBuilder.
	get_DriverAssembly (see page 1795)	This is get_DriverAssembly, a member of class TAdoDbxConnectionStringBuilder.
	get_DriverAssemblyLoader (see page 1796)	This is get_DriverAssemblyLoader, a member of class TAdoDbxConnectionStringBuilder.
	get_DriverName (see page 1796)	This is get_DriverName, a member of class TAdoDbxConnectionStringBuilder.
	get_DriverPackage (see page 1796)	This is get_DriverPackage, a member of class TAdoDbxConnectionStringBuilder.
	get_DriverPackageLoader (see page 1796)	This is get_DriverPackageLoader, a member of class TAdoDbxConnectionStringBuilder.
	get_DriverUnit (see page 1796)	This is get_DriverUnit, a member of class TAdoDbxConnectionStringBuilder.
	get_ErrorResourceFile (see page 1797)	This is get_ErrorResourceFile, a member of class TAdoDbxConnectionStringBuilder.
	get_GetDriverFunc (see page 1797)	This is get_GetDriverFunc, a member of class TAdoDbxConnectionStringBuilder.
	get_HostName (see page 1797)	This is get_HostName, a member of class TAdoDbxConnectionStringBuilder.
	get_IsolationLevel (see page 1797)	This is get_IsolationLevel, a member of class TAdoDbxConnectionStringBuilder.
V	get_Item (see page 1797)	This is get_Item, a member of class TAdoDbxConnectionStringBuilder.
	get_LibraryName (see page 1798)	This is get_LibraryName, a member of class TAdoDbxConnectionStringBuilder.

	get_MaxBlobSize (see page 1798)	This is get_MaxBlobSize, a member of class TAdoDbxConnectionStringBuilder.
	get_MetaDataAssemblyLoader (see page 1798)	This is get_MetaDataAssemblyLoader, a member of class TAdoDbxConnectionStringBuilder.
	get_MetaDataPackageLoader (see page 1798)	This is get_MetaDataPackageLoader, a member of class TAdoDbxConnectionStringBuilder.
	get_Password (see page 1798)	This is get_Password, a member of class TAdoDbxConnectionStringBuilder.
	get_Port (see page 1799)	This is get_Port, a member of class TAdoDbxConnectionStringBuilder.
	get_ProductVersion (see page 1799)	This is get_ProductVersion, a member of class TAdoDbxConnectionStringBuilder.
	get_Role (see page 1799)	This is get_Role, a member of class TAdoDbxConnectionStringBuilder.
	get_SchemaOverride (see page 1799)	This is get_SchemaOverride, a member of class TAdoDbxConnectionStringBuilder.
	get_User_Name (see page 1799)	This is get_User_Name, a member of class TAdoDbxConnectionStringBuilder.
	get_VendorLib (see page 1800)	This is get_VendorLib, a member of class TAdoDbxConnectionStringBuilder.
	set_ConnectionName (see page 1800)	This is set_ConnectionName, a member of class TAdoDbxConnectionStringBuilder.
	set_Database (see page 1800)	This is set_Database, a member of class TAdoDbxConnectionStringBuilder.
	set_DelegateConnection (see page 1800)	This is set_DelegateConnection, a member of class TAdoDbxConnectionStringBuilder.
	set_DelegateDriver (see page 1800)	This is set_DelegateDriver, a member of class TAdoDbxConnectionStringBuilder.
	set_DriverAssembly (see page 1801)	This is set_DriverAssembly, a member of class TAdoDbxConnectionStringBuilder.
	set_DriverAssemblyLoader (see page 1801)	This is set_DriverAssemblyLoader, a member of class TAdoDbxConnectionStringBuilder.
	set_DriverName (see page 1801)	This is set_DriverName, a member of class TAdoDbxConnectionStringBuilder.
	set_DriverPackage (see page 1801)	This is set_DriverPackage, a member of class TAdoDbxConnectionStringBuilder.
	set_DriverPackageLoader (see page 1801)	This is set_DriverPackageLoader, a member of class TAdoDbxConnectionStringBuilder.
	set_DriverUnit (see page 1802)	This is set_DriverUnit, a member of class TAdoDbxConnectionStringBuilder.
	set_ErrorResourceFile (see page 1802)	This is set_ErrorResourceFile, a member of class TAdoDbxConnectionStringBuilder.
	set_GetDriverFunc (see page 1802)	This is set_GetDriverFunc, a member of class TAdoDbxConnectionStringBuilder.
	set_HostName (see page 1802)	This is set_HostName, a member of class TAdoDbxConnectionStringBuilder.
	set_IsolationLevel (see page 1802)	This is set_IsolationLevel, a member of class TAdoDbxConnectionStringBuilder.
	set_Item (see page 1803)	This is set_Item, a member of class TAdoDbxConnectionStringBuilder.
	set_LibraryName (see page 1803)	This is set_LibraryName, a member of class TAdoDbxConnectionStringBuilder.
	set_MaxBlobSize (see page 1803)	This is set_MaxBlobSize, a member of class TAdoDbxConnectionStringBuilder.
	set_MetaDataAssemblyLoader (see page 1803)	This is set_MetaDataAssemblyLoader, a member of class TAdoDbxConnectionStringBuilder.
	set_MetaDataPackageLoader (see page 1803)	This is set_MetaDataPackageLoader, a member of class TAdoDbxConnectionStringBuilder.
	set_Password (see page 1804)	This is set_Password, a member of class TAdoDbxConnectionStringBuilder.
	set_Port (see page 1804)	This is set_Port, a member of class TAdoDbxConnectionStringBuilder.
	set_ProductVersion (see page 1804)	This is set_ProductVersion, a member of class TAdoDbxConnectionStringBuilder.
	set_Role (see page 1804)	This is set_Role, a member of class TAdoDbxConnectionStringBuilder.
	set_SchemaOverride (see page 1804)	This is set_SchemaOverride, a member of class TAdoDbxConnectionStringBuilder.
	set_User_Name (see page 1805)	This is set_User_Name, a member of class TAdoDbxConnectionStringBuilder.
	set_VendorLib (see page 1805)	This is set_VendorLib, a member of class TAdoDbxConnectionStringBuilder.
	TryGetValue (see page 1805)	Gets key value.

Public Properties

	Name	Description
	ConnectionName (see page 1806)	Name of connection.
	DelegateDriver (see page 1807)	Driver is a delegate.
	DriverName (see page 1808)	Unique name for a dbExpress driver.
	ErrorResourceFile (see page 1809)	Specify a localized error message file for Linux.

Published Properties

	Name	Description
	Database (see page 1806)	Name of a database to connect to.
	DelegateConnection (see page 1807)	Delegate connection used by this connection.
	DriverAssembly (see page 1807)	Used to load a managed driver dynamically.
	DriverAssemblyLoader (see page 1808)	Used to load a managed driver dynamically from an assembly.
	DriverPackage (see page 1808)	Used to load a native driver dynamically.
	DriverPackageLoader (see page 1809)	Native Driver loader class used to load a native driver dynamically from a package.
	DriverUnit (see page 1809)	Used to provide informative error message when driver cannot be loaded.
	GetDriverFunc (see page 1809)	Exported entry point for loading a driver prior to dbExpress 4.
	HostName (see page 1810)	Host name of a database server.
	IsolationLevel (see page 1810)	Initial transaction isolation to use when connecting to a database.
	LibraryName (see page 1811)	Name of the DBXDynalink DLL.
	MaxBlobSize (see page 1811)	Maximum BLOB size.
	MetaDataAssemblyLoader (see page 1811)	Used to load a managed driver dynamically from an assembly.
	MetaDataPackageLoader (see page 1811)	Class used to load a native driver dynamically from a package.
	Password (see page 1812)	Login password name.
	Port (see page 1812)	Port number of a database server.
	ProductVersion (see page 1812)	Driver product version.
	Role (see page 1813)	Login role.
	SchemaOverride (see page 1813)	VCL components use this setting to scope metadata requests.
	User_Name (see page 1814)	This is User_Name, a member of class TAdoDbxConnectionStringBuilder.
	VendorLib (see page 1814)	Name of a vendor specific client library.

15.1.1.1.4.2 TAdoDbxConnectionStringBuilder.Create Constructor

Creates an instance of TAdoDbxConnectionStringBuilder (see page 1790)

Pascal

```
constructor Create;
```

C++

```
__fastcall TAdoDbxConnectionStringBuilder();
```

Description

This constructor creates and initializes an instance of TAdoDbxConnectionStringBuilder (see page 1790).

See Also

Create (MSDN)

15.1.1.1.4.3 TAdoDbxConnectionStringBuilder Methods

The methods of the TAdoDbxConnectionStringBuilder class are listed here.

Public Methods

	Name	Description
	get_ConnectionName (see page 1795)	This is get_ConnectionName, a member of class TAdoDbxConnectionStringBuilder.
	get_Database (see page 1795)	This is get_Database, a member of class TAdoDbxConnectionStringBuilder.
	get_DelegateConnection (see page 1795)	This is get_DelegateConnection, a member of class TAdoDbxConnectionStringBuilder.
	get_DelegateDriver (see page 1795)	This is get_DelegateDriver, a member of class TAdoDbxConnectionStringBuilder.
	get_DriverAssembly (see page 1795)	This is get_DriverAssembly, a member of class TAdoDbxConnectionStringBuilder.

	get_DriverAssemblyLoader (see page 1796)	This is get_DriverAssemblyLoader, a member of class TAdoDbxConnectionStringBuilder.
	get_DriverName (see page 1796)	This is get_DriverName, a member of class TAdoDbxConnectionStringBuilder.
	get_DriverPackage (see page 1796)	This is get_DriverPackage, a member of class TAdoDbxConnectionStringBuilder.
	get_DriverPackageLoader (see page 1796)	This is get_DriverPackageLoader, a member of class TAdoDbxConnectionStringBuilder.
	get_DriverUnit (see page 1796)	This is get_DriverUnit, a member of class TAdoDbxConnectionStringBuilder.
	get_ErrorResourceFile (see page 1797)	This is get_ErrorResourceFile, a member of class TAdoDbxConnectionStringBuilder.
	get_GetDriverFunc (see page 1797)	This is get_GetDriverFunc, a member of class TAdoDbxConnectionStringBuilder.
	get_HostName (see page 1797)	This is get_HostName, a member of class TAdoDbxConnectionStringBuilder.
	get_IsolationLevel (see page 1797)	This is get_IsolationLevel, a member of class TAdoDbxConnectionStringBuilder.
	get_Item (see page 1797)	This is get_Item, a member of class TAdoDbxConnectionStringBuilder.
	get_LibraryName (see page 1798)	This is get_LibraryName, a member of class TAdoDbxConnectionStringBuilder.
	get_MaxBlobSize (see page 1798)	This is get_MaxBlobSize, a member of class TAdoDbxConnectionStringBuilder.
	get_MetaDataAssemblyLoader (see page 1798)	This is get_MetaDataAssemblyLoader, a member of class TAdoDbxConnectionStringBuilder.
	get_MetaDataPackageLoader (see page 1798)	This is get_MetaDataPackageLoader, a member of class TAdoDbxConnectionStringBuilder.
	get_Password (see page 1798)	This is get_Password, a member of class TAdoDbxConnectionStringBuilder.
	get_Port (see page 1799)	This is get_Port, a member of class TAdoDbxConnectionStringBuilder.
	get_ProductVersion (see page 1799)	This is get_ProductVersion, a member of class TAdoDbxConnectionStringBuilder.
	get_Role (see page 1799)	This is get_Role, a member of class TAdoDbxConnectionStringBuilder.
	get_SchemaOverride (see page 1799)	This is get_SchemaOverride, a member of class TAdoDbxConnectionStringBuilder.
	get_User_Name (see page 1799)	This is get_User_Name, a member of class TAdoDbxConnectionStringBuilder.
	get_VendorLib (see page 1800)	This is get_VendorLib, a member of class TAdoDbxConnectionStringBuilder.
	set_ConnectionName (see page 1800)	This is set_ConnectionName, a member of class TAdoDbxConnectionStringBuilder.
	set_Database (see page 1800)	This is set_Database, a member of class TAdoDbxConnectionStringBuilder.
	set_DelegateConnection (see page 1800)	This is set_DelegateConnection, a member of class TAdoDbxConnectionStringBuilder.
	set_DelegateDriver (see page 1800)	This is set_DelegateDriver, a member of class TAdoDbxConnectionStringBuilder.
	set_DriverAssembly (see page 1801)	This is set_DriverAssembly, a member of class TAdoDbxConnectionStringBuilder.
	set_DriverAssemblyLoader (see page 1801)	This is set_DriverAssemblyLoader, a member of class TAdoDbxConnectionStringBuilder.
	set_DriverName (see page 1801)	This is set_DriverName, a member of class TAdoDbxConnectionStringBuilder.
	set_DriverPackage (see page 1801)	This is set_DriverPackage, a member of class TAdoDbxConnectionStringBuilder.
	set_DriverPackageLoader (see page 1801)	This is set_DriverPackageLoader, a member of class TAdoDbxConnectionStringBuilder.
	set_DriverUnit (see page 1802)	This is set_DriverUnit, a member of class TAdoDbxConnectionStringBuilder.
	set_ErrorResourceFile (see page 1802)	This is set_ErrorResourceFile, a member of class TAdoDbxConnectionStringBuilder.
	set_GetDriverFunc (see page 1802)	This is set_GetDriverFunc, a member of class TAdoDbxConnectionStringBuilder.
	set_HostName (see page 1802)	This is set_HostName, a member of class TAdoDbxConnectionStringBuilder.
	set_IsolationLevel (see page 1802)	This is set_IsolationLevel, a member of class TAdoDbxConnectionStringBuilder.
	set_Item (see page 1803)	This is set_Item, a member of class TAdoDbxConnectionStringBuilder.
	set_LibraryName (see page 1803)	This is set_LibraryName, a member of class TAdoDbxConnectionStringBuilder.
	set_MaxBlobSize (see page 1803)	This is set_MaxBlobSize, a member of class TAdoDbxConnectionStringBuilder.
	set_MetaDataAssemblyLoader (see page 1803)	This is set_MetaDataAssemblyLoader, a member of class TAdoDbxConnectionStringBuilder.
	set_MetaDataPackageLoader (see page 1803)	This is set_MetaDataPackageLoader, a member of class TAdoDbxConnectionStringBuilder.
	set_Password (see page 1804)	This is set_Password, a member of class TAdoDbxConnectionStringBuilder.
	set_Port (see page 1804)	This is set_Port, a member of class TAdoDbxConnectionStringBuilder.
	set_ProductVersion (see page 1804)	This is set_ProductVersion, a member of class TAdoDbxConnectionStringBuilder.
	set_Role (see page 1804)	This is set_Role, a member of class TAdoDbxConnectionStringBuilder.
	set_SchemaOverride (see page 1804)	This is set_SchemaOverride, a member of class TAdoDbxConnectionStringBuilder.

	set_User_Name (see page 1805)	This is set_User_Name, a member of class TAdoDbxConnectionStringBuilder.
	set_VendorLib (see page 1805)	This is set_VendorLib, a member of class TAdoDbxConnectionStringBuilder.
	TryGetValue (see page 1805)	Gets key value.

15.1.1.1.4.3.1 TAdoDbxConnectionStringBuilder.get_ConnectionName Method

Pascal

```
function get_ConnectionName: string;
```

C++

```
__fastcall AnsiString get_ConnectionName();
```

Description

This is get_ConnectionName, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.2 TAdoDbxConnectionStringBuilder.get_Database Method

Pascal

```
function get_Database: string;
```

C++

```
__fastcall AnsiString get_Database();
```

Description

This is get_Database, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.3 TAdoDbxConnectionStringBuilder.get_DelegateConnection Method

Pascal

```
function get_DelegateConnection: string;
```

C++

```
__fastcall AnsiString get_DelegateConnection();
```

Description

This is get_DelegateConnection, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.4 TAdoDbxConnectionStringBuilder.get_DelegateDriver Method

Pascal

```
function get_DelegateDriver: string;
```

C++

```
__fastcall AnsiString get_DelegateDriver();
```

Description

This is get_DelegateDriver, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.5 TAdoDbxConnectionStringBuilder.get_DriverAssembly Method

Pascal

```
function get_DriverAssembly: string;
```

C++

```
__fastcall AnsiString get_DriverAssembly();
```

Description

This is get_DriverAssembly, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.6 TAdoDbxConnectionStringBuilder.get_DriverAssemblyLoader Method

Pascal

```
function get_DriverAssemblyLoader: string;
```

C++

```
__fastcall AnsiString get_DriverAssemblyLoader();
```

Description

This is get_DriverAssemblyLoader, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.7 TAdoDbxConnectionStringBuilder.get_DriverName Method

Pascal

```
function get_DriverName: string;
```

C++

```
__fastcall AnsiString get_DriverName();
```

Description

This is get_DriverName, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.8 TAdoDbxConnectionStringBuilder.get_DriverPackage Method

Pascal

```
function get_DriverPackage: string;
```

C++

```
__fastcall AnsiString get_DriverPackage();
```

Description

This is get_DriverPackage, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.9 TAdoDbxConnectionStringBuilder.get_DriverPackageLoader Method

Pascal

```
function get_DriverPackageLoader: string;
```

C++

```
__fastcall AnsiString get_DriverPackageLoader();
```

Description

This is get_DriverPackageLoader, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.10 TAdoDbxConnectionStringBuilder.get_DriverUnit Method

Pascal

```
function get_DriverUnit: string;
```

C++

```
__fastcall AnsiString get_DriverUnit();
```

Description

This is get_DriverUnit, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.11 TAdoDbxConnectionStringBuilder.get_ErrorResourceFile Method

Pascal

```
function get_ErrorResourceFile: string;
```

C++

```
__fastcall AnsiString get_ErrorResourceFile();
```

Description

This is get_ErrorResourceFile, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.12 TAdoDbxConnectionStringBuilder.get_GetDriverFunc Method

Pascal

```
function get_GetDriverFunc: string;
```

C++

```
__fastcall AnsiString get_GetDriverFunc();
```

Description

This is get_GetDriverFunc, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.13 TAdoDbxConnectionStringBuilder.get_HostName Method

Pascal

```
function get_HostName: string;
```

C++

```
__fastcall AnsiString get_HostName();
```

Description

This is get_HostName, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.14 TAdoDbxConnectionStringBuilder.get_IsolationLevel Method

Pascal

```
function get_IsolationLevel: Integer;
```

C++

```
__fastcall int get_IsolationLevel();
```

Description

This is get_IsolationLevel, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.15 TAdoDbxConnectionStringBuilder.get_Item Method

Pascal

```
function get_Item(keyword: string): TObject; override;
```

C++

```
virtual __fastcall TObject * get_Item(AnsiString keyword);
```

Description

This is get_Item, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.16 TAdoDbxConnectionStringBuilder.get_LibraryName Method

Pascal

```
function get_LibraryName: string;
```

C++

```
__fastcall AnsiString get_LibraryName();
```

Description

This is get_LibraryName, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.17 TAdoDbxConnectionStringBuilder.get_MaxBlobSize Method

Pascal

```
function get_MaxBlobSize: Integer;
```

C++

```
__fastcall int get_MaxBlobSize();
```

Description

This is get_MaxBlobSize, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.18 TAdoDbxConnectionStringBuilder.get_MetaDataAssemblyLoader Method

Pascal

```
function get_MetaDataAssemblyLoader: string;
```

C++

```
__fastcall AnsiString get_MetaDataAssemblyLoader();
```

Description

This is get_MetaDataAssemblyLoader, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.19 TAdoDbxConnectionStringBuilder.get_MetaDataPackageLoader Method

Pascal

```
function get_MetaDataPackageLoader: string;
```

C++

```
__fastcall AnsiString get_MetaDataPackageLoader();
```

Description

This is get_MetaDataPackageLoader, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.20 TAdoDbxConnectionStringBuilder.get_Password Method

Pascal

```
function get_Password: string;
```

C++

```
__fastcall AnsiString get_Password();
```

Description

This is get_Password, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.21 TAdoDbxConnectionStringBuilder.get_Port Method

Pascal

```
function get_Port: Integer;
```

C++

```
__fastcall int get_Port();
```

Description

This is get_Port, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.22 TAdoDbxConnectionStringBuilder.get_ProductVersion Method

Pascal

```
function get_ProductVersion: string;
```

C++

```
__fastcall AnsiString get_ProductVersion();
```

Description

This is get_ProductVersion, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.23 TAdoDbxConnectionStringBuilder.get_Role Method

Pascal

```
function get_Role: string;
```

C++

```
__fastcall AnsiString get_Role();
```

Description

This is get_Role, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.24 TAdoDbxConnectionStringBuilder.get_SchemaOverride Method

Pascal

```
function get_SchemaOverride: string;
```

C++

```
__fastcall AnsiString get_SchemaOverride();
```

Description

This is get_SchemaOverride, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.25 TAdoDbxConnectionStringBuilder.get_User_Name Method

Pascal

```
function get_User_Name: string;
```

C++

```
__fastcall AnsiString get_User_Name();
```

Description

This is get_User_Name, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.26 TAdoDbxConnectionStringBuilder.get_VendorLib Method

Pascal

```
function get_VendorLib: string;
```

C++

```
__fastcall AnsiString get_VendorLib();
```

Description

This is get_VendorLib, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.27 TAdoDbxConnectionStringBuilder.set_ConnectionName Method

Pascal

```
procedure set_ConnectionName(const Value: string);
```

C++

```
__fastcall set_ConnectionName(const AnsiString Value);
```

Description

This is set_ConnectionName, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.28 TAdoDbxConnectionStringBuilder.set_Database Method

Pascal

```
procedure set_Database(const Value: string);
```

C++

```
__fastcall set_Database(const AnsiString Value);
```

Description

This is set_Database, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.29 TAdoDbxConnectionStringBuilder.set_DelegateConnection Method

Pascal

```
procedure set_DelegateConnection(const Value: string);
```

C++

```
__fastcall set_DelegateConnection(const AnsiString Value);
```

Description

This is set_DelegateConnection, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.30 TAdoDbxConnectionStringBuilder.set_DelegateDriver Method

Pascal

```
procedure set_DelegateDriver(const Value: string);
```

C++

```
__fastcall set_DelegateDriver(const AnsiString Value);
```

Description

This is set_DelegateDriver, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.31 TAdoDbxConnectionStringBuilder.set_DriverAssembly Method

Pascal

```
procedure set_DriverAssembly(const Value: string);
```

C++

```
__fastcall set_DriverAssembly(const AnsiString Value);
```

Description

This is set_DriverAssembly, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.32 TAdoDbxConnectionStringBuilder.set_DriverAssemblyLoader Method

Pascal

```
procedure set_DriverAssemblyLoader(const Value: string);
```

C++

```
__fastcall set_DriverAssemblyLoader(const AnsiString Value);
```

Description

This is set_DriverAssemblyLoader, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.33 TAdoDbxConnectionStringBuilder.set_DriverName Method

Pascal

```
procedure set_DriverName(const Value: string);
```

C++

```
__fastcall set_DriverName(const AnsiString Value);
```

Description

This is set_DriverName, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.34 TAdoDbxConnectionStringBuilder.set_DriverPackage Method

Pascal

```
procedure set_DriverPackage(const Value: string);
```

C++

```
__fastcall set_DriverPackage(const AnsiString Value);
```

Description

This is set_DriverPackage, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.35 TAdoDbxConnectionStringBuilder.set_DriverPackageLoader Method

Pascal

```
procedure set_DriverPackageLoader(const Value: string);
```

C++

```
__fastcall set_DriverPackageLoader(const AnsiString Value);
```

Description

This is set_DriverPackageLoader, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.36 TAdoDbxConnectionStringBuilder.set_DriverUnit Method

Pascal

```
procedure set_DriverUnit(const Value: string);
```

C++

```
__fastcall set_DriverUnit(const AnsiString Value);
```

Description

This is set_DriverUnit, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.37 TAdoDbxConnectionStringBuilder.set_ErrorResourceFile Method

Pascal

```
procedure set_ErrorResourceFile(const Value: string);
```

C++

```
__fastcall set_ErrorResourceFile(const AnsiString Value);
```

Description

This is set_ErrorResourceFile, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.38 TAdoDbxConnectionStringBuilder.set_GetDriverFunc Method

Pascal

```
procedure set_GetDriverFunc(const Value: string);
```

C++

```
__fastcall set_GetDriverFunc(const AnsiString Value);
```

Description

This is set_GetDriverFunc, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.39 TAdoDbxConnectionStringBuilder.set_HostName Method

Pascal

```
procedure set_HostName(const Value: string);
```

C++

```
__fastcall set_HostName(const AnsiString Value);
```

Description

This is set_HostName, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.40 TAdoDbxConnectionStringBuilder.set_IsolationLevel Method

Pascal

```
procedure set_IsolationLevel(const Value: Integer);
```

C++

```
__fastcall set_IsolationLevel(const int Value);
```

Description

This is set_IsolationLevel, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.41 TAdoDbxConnectionStringBuilder.set_Item Method

Pascal

```
procedure set_Item(keyword: string; value: TObject); override;
```

C++

```
virtual __fastcall set_Item(AnsiString keyword, TObject * value);
```

Description

This is set_Item, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.42 TAdoDbxConnectionStringBuilder.set_LibraryName Method

Pascal

```
procedure set_LibraryName(const Value: string);
```

C++

```
__fastcall set_LibraryName(const AnsiString Value);
```

Description

This is set_LibraryName, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.43 TAdoDbxConnectionStringBuilder.set_MaxBlobSize Method

Pascal

```
procedure set_MaxBlobSize(const Value: Integer);
```

C++

```
__fastcall set_MaxBlobSize(const int Value);
```

Description

This is set_MaxBlobSize, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.44 TAdoDbxConnectionStringBuilder.set_MetaDataAssemblyLoader Method

Pascal

```
procedure set_MetaDataAssemblyLoader(const Value: string);
```

C++

```
__fastcall set_MetaDataAssemblyLoader(const AnsiString Value);
```

Description

This is set_MetaDataAssemblyLoader, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.45 TAdoDbxConnectionStringBuilder.set_MetaDataPackageLoader Method

Pascal

```
procedure set_MetaDataPackageLoader(const Value: string);
```

C++

```
__fastcall set_MetaDataPackageLoader(const AnsiString Value);
```

Description

This is set_MetaDataPackageLoader, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.46 TAdoDbxConnectionStringBuilder.set_Password Method

Pascal

```
procedure set_Password(const Value: string);
```

C++

```
__fastcall set_Password(const AnsiString Value);
```

Description

This is set_Password, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.47 TAdoDbxConnectionStringBuilder.set_Port Method

Pascal

```
procedure set_Port(const Value: Integer);
```

C++

```
__fastcall set_Port(const int Value);
```

Description

This is set_Port, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.48 TAdoDbxConnectionStringBuilder.set_ProductVersion Method

Pascal

```
procedure set_ProductVersion(const Value: string);
```

C++

```
__fastcall set_ProductVersion(const AnsiString Value);
```

Description

This is set_ProductVersion, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.49 TAdoDbxConnectionStringBuilder.set_Role Method

Pascal

```
procedure set_Role(const Value: string);
```

C++

```
__fastcall set_Role(const AnsiString Value);
```

Description

This is set_Role, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.50 TAdoDbxConnectionStringBuilder.set_SchemaOverride Method

Pascal

```
procedure set_SchemaOverride(const Value: string);
```

C++

```
__fastcall set_SchemaOverride(const AnsiString Value);
```

Description

This is set_SchemaOverride, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.51 TAdoDbxConnectionStringBuilder.set_User_Name Method

Pascal

```
procedure set_User_Name(const Value: string);
```

C++

```
__fastcall set_User_Name(const AnsiString Value);
```

Description

This is set_User_Name, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.52 TAdoDbxConnectionStringBuilder.set_VendorLib Method

Pascal

```
procedure set_VendorLib(const Value: string);
```

C++

```
__fastcall set_VendorLib(const AnsiString Value);
```

Description

This is set_VendorLib, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.3.53 TAdoDbxConnectionStringBuilder.TryGetValue Method

Gets key value.

Pascal

```
function TryGetValue(keyword: string; out value: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean TryGetValue(AnsiString keyword, TObject * value);
```

Description

TryGetValue gets a value corresponding to a given key.

See Also

TryGetValue (MSDN)

15.1.1.1.4.4 TAdoDbxConnectionStringBuilder Properties

The properties of the TAdoDbxConnectionStringBuilder class are listed here.

Public Properties

	Name	Description
	ConnectionName (see page 1806)	Name of connection.
	DelegateDriver (see page 1807)	Driver is a delegate.
	DriverName (see page 1808)	Unique name for a dbExpress driver.
	ErrorResourceFile (see page 1809)	Specify a localized error message file for Linux.

Published Properties

	Name	Description
	Database (see page 1806)	Name of a database to connect to.
	DelegateConnection (see page 1807)	Delegate connection used by this connection.
	DriverAssembly (see page 1807)	Used to load a managed driver dynamically.
	DriverAssemblyLoader (see page 1808)	Used to load a managed driver dynamically from an assembly.
	DriverPackage (see page 1808)	Used to load a native driver dynamically.
	DriverPackageLoader (see page 1809)	Native Driver loader class used to load a native driver dynamically from a package.
	DriverUnit (see page 1809)	Used to provide informative error message when driver cannot be loaded.
	GetDriverFunc (see page 1809)	Exported entry point for loading a driver prior to dbExpress 4.
	HostName (see page 1810)	Host name of a database server.
	IsolationLevel (see page 1810)	Initial transaction isolation to use when connecting to a database.
	LibraryName (see page 1811)	Name of the DBXDynalink DLL.
	MaxBlobSize (see page 1811)	Maximum BLOB size.
	MetaDataAssemblyLoader (see page 1811)	Used to load a managed driver dynamically from an assembly.
	MetaDataPackageLoader (see page 1811)	Class used to load a native driver dynamically from a package.
	Password (see page 1812)	Login password name.
	Port (see page 1812)	Port number of a database server.
	ProductVersion (see page 1812)	Driver product version.
	Role (see page 1813)	Login role.
	SchemaOverride (see page 1813)	VCL components use this setting to scope metadata requests.
	User_Name (see page 1814)	This is User_Name, a member of class TAdoDbxConnectionStringBuilder.
	VendorLib (see page 1814)	Name of a vendor specific client library.

15.1.1.1.4.4.1 TAdoDbxConnectionStringBuilder.ConnectionName Property

Name of connection.

Pascal

```
property ConnectionName: string;
```

C++

```
__property AnsiString ConnectionName;
```

Description

Name of connection. If connection properties are read from the dbxconnections.ini file, this is the ini file section name. AdoDbx Client can use this property setting for a simple one property connection string. Connection names can also be used to load a connection from a TDBXConnectionFactory instance.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.2 TAdoDbxConnectionStringBuilder.Database Property

Name of a database to connect to.

Pascal

```
[Category(TConnectionStringCategories.Database)]
property Database: string;
```

C++

```
[Category(TConnectionStringCategories.Database)]
__property AnsiString Database;
```

Description

Name of a database to connect to for this connection.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.3 TAdoDbxConnectionStringBuilder.DelegateConnection Property

Delegate connection used by this connection.

Pascal

```
[Category(TConnectionStringCategories.DriverConfiguration)]
property DelegateConnection: string;
```

C++

```
[Category(TConnectionStringCategories.DriverConfiguration)]
__property AnsiString DelegateConnection;
```

Description

Delegate connection used by this connection. Delegate connections implement the dbExpress 4 driver framework. They can perform pre or post processing of all public and protected methods before delegating to the real connection or another delegate connection.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.4 TAdoDbxConnectionStringBuilder.DelegateDriver Property

Driver is a delegate.

Pascal

```
property DelegateDriver: string;
```

C++

```
__property AnsiString DelegateDriver;
```

Description

Indicates that this driver is a delegate and can only be used in conjunction with a non-delegate driver.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.5 TAdoDbxConnectionStringBuilder.DriverAssembly Property

Used to load a managed driver dynamically.

Pascal

```
[Category(TConnectionStringCategories.DriverConfiguration)]
property DriverAssembly: string;
```

C++

```
[Category(TConnectionStringCategories.DriverConfiguration)]
__property AnsiString DriverAssembly;
```

Description

Used to load a managed driver dynamically for this connection.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.6 TAdoDbxConnectionStringBuilder.DriverAssemblyLoader Property

Used to load a managed driver dynamically from an assembly.

Pascal

```
[Category(TConnectionStringCategories.DriverConfiguration)]
property DriverAssemblyLoader: string;
```

C++

```
[Category(TConnectionStringCategories.DriverConfiguration)]
__property AnsiString DriverAssemblyLoader;
```

Description

Managed Driver loader class used to load a managed driver dynamically from an assembly.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.7 TAdoDbxConnectionStringBuilder.DriverName Property

Unique name for a dbExpress driver.

Pascal

```
property DriverName: string;
```

C++

```
__property AnsiString DriverName;
```

Description

Unique name for a dbExpress driver. Used by the dbxconnections.ini file to refer to a connections driver in the dbxdrivers.ini file.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.8 TAdoDbxConnectionStringBuilder.DriverPackage Property

Used to load a native driver dynamically.

Pascal

```
[Category(TConnectionStringCategories.DriverConfiguration)]
property DriverPackage: string;
```

C++

```
[Category(TConnectionStringCategories.DriverConfiguration)]
__property AnsiString DriverPackage;
```

Description

Used to load a native driver dynamically for this connection.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.9 TAdoDbxConnectionStringBuilder.DriverPackageLoader Property

Native Driver loader class used to load a native driver dynamically from a package.

Pascal

```
[Category(TConnectionStringCategories.DriverConfiguration)]
property DriverPackageLoader: string;
```

C++

```
[Category(TConnectionStringCategories.DriverConfiguration)]
__property AnsiString DriverPackageLoader;
```

Description

Native Driver loader class used to load a native driver dynamically from a package for this connection.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.10 TAdoDbxConnectionStringBuilder.DriverUnit Property

Used to provide informative error message when driver cannot be loaded.

Pascal

```
[Category(TConnectionStringCategories.DriverConfiguration)]
property DriverUnit: string;
```

C++

```
[Category(TConnectionStringCategories.DriverConfiguration)]
__property AnsiString DriverUnit;
```

Description

Used to provide informative error message when driver cannot be loaded for this connection.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.11 TAdoDbxConnectionStringBuilder.ErrorResourceFile Property

Specify a localized error message file for Linux.

Pascal

```
property ErrorResourceFile: string;
```

C++

```
__property AnsiString ErrorResourceFile;
```

Description

Used only by Linux versions of dbExpress to specify a localized error message file.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.12 TAdoDbxConnectionStringBuilder.GetDriverFunc Property

Exported entry point for loading a driver prior to dbExpress 4.

Pascal

```
[Category(TConnectionStringCategories.Dynalink)]
property GetDriverFunc: string;
```

C++

```
[Category(TConnectionStringCategories.Dynalink)]
__property AnsiString GetDriverFunc;
```

Description

Used by DBXDynalink drivers to specify exported entry point for loading a driver prior to dbExpress 4.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.13 TAdoDbxConnectionStringBuilder.HostName Property

Host name of a database server.

Pascal

```
[Category(TConnectionStringCategories.Database)]
property HostName: string;
```

C++

```
[Category(TConnectionStringCategories.Database)]
__property AnsiString HostName;
```

Description

Host name of a database server that a driver connects to.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.14 TAdoDbxConnectionStringBuilder.IsolationLevel Property

Initial transaction isolation to use when connecting to a database.

Pascal

```
[Category(TConnectionStringCategories.Database)]
property IsolationLevel: Integer;
```

C++

```
[Category(TConnectionStringCategories.Database)]
__property int IsolationLevel;
```

Description

Initial transaction isolation to use when connecting to a database. Transaction isolation can also be specified when a transaction is started.

See Also

IsolationLevel (MSDN)

IsolationLevel enumeration (MSDN)

TDBXPropertyNames (see page 10566)

IsolationLevel

IsolationLevel

15.1.1.1.4.4.15 TAdoDbxConnectionStringBuilder.LibraryName Property

Name of the DBXDynalink DLL.

Pascal

```
[Category(TConnectionStringCategories.Dynalink)]
property LibraryName: string;
```

C++

```
[Category(TConnectionStringCategories.Dynalink)]
__property AnsiString LibraryName;
```

Description

Used by DBXDynalink drivers to specify the name of the DBXDynalink DLL.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.16 TAdoDbxConnectionStringBuilder.MaxBlobSize Property

Maximum BLOB size.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

Maximum BLOB size. Set to –1 to specify no limit.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.17 TAdoDbxConnectionStringBuilder.MetaDataAssemblyLoader Property

Used to load a managed driver dynamically from an assembly.

Pascal

```
[Category(TConnectionStringCategories.DriverConfiguration)]
property MetaDataAssemblyLoader: string;
```

C++

```
[Category(TConnectionStringCategories.DriverConfiguration)]
__property AnsiString MetaDataAssemblyLoader;
```

Description

Managed MetaData (see page 2130) loader class used to load a managed driver dynamically from an assembly.

See Also

TDBXPropertyNamesEx (see page 10574)

15.1.1.1.4.4.18 TAdoDbxConnectionStringBuilder.MetaDataPackageLoader Property

Class used to load a native driver dynamically from a package.

Pascal

```
[Category(TConnectionStringCategories.DriverConfiguration)]
property MetaDataPackageLoader: string;
```

C++

```
[Category(TConnectionStringCategories.DriverConfiguration)]
__property AnsiString MetaDataPackageLoader;
```

Description

Native MetaData (see page 2130) loader class used to load a native driver dynamically from a package.

See Also

TDBXPropertyNamesEx (see page 10574)

15.1.1.1.4.4.19 TAdoDbxConnectionStringBuilder.Password Property

Login password name.

Pascal

```
[PasswordPropertyText(true)]
[Category(TConnectionStringCategories.Database)]
property Password: string;
```

C++

```
[PasswordPropertyText(true)]
[Category(TConnectionStringCategories.Database)]
__property AnsiString Password;
```

Description

Login password name used to connect to a database.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.20 TAdoDbxConnectionStringBuilder.Port Property

Port number of a database server.

Pascal

```
[Category(TConnectionStringCategories.Database)]
property Port: Integer;
```

C++

```
[Category(TConnectionStringCategories.Database)]
__property int Port;
```

Description

Port number of a database server that a driver connects to.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.21 TAdoDbxConnectionStringBuilder.ProductVersion Property

Driver product version.

Pascal

```
[Category(TConnectionStringCategories.DriverConfiguration)]
property ProductVersion: string;
```

C++

```
[Category(TConnectionStringCategories.DriverConfiguration)]
__property AnsiString ProductVersion;
```

Description

Can be specified as a driver property as an override of the TDBXConnectionEx.ProductVersion property. This may be necessary for drivers that do not have the ability to report the product version. The ProductVersion is a number in the format nn.nn.nnnn.

See Also

TDBXPropertyNamesEx (see page 10574)

15.1.1.1.4.4.22 TAdoDbxConnectionStringBuilder.Role Property

Login role.

Pascal

```
[Category(TConnectionStringCategories.Database)]
property Role: string;
```

C++

```
[Category(TConnectionStringCategories.Database)]
__property AnsiString Role;
```

Description

Login role used to connect to a database.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.23 TAdoDbxConnectionStringBuilder.SchemaOverride Property

VCL components use this setting to scope metadata requests.

Pascal

```
property SchemaOverride: string;
```

C++

```
__property AnsiString SchemaOverride;
```

Description

Most applications should not need to use this constant. It is provided for backwards compatibility of the dbExpress VCL. VCL components use this setting to scope metadata requests. This property should be set to **user-name.override-schema-name**. For example, tables created by the MSSQL sa user are placed in the dbo schema. In this case, SchemaOverride can be set to 'sa.dbo'. This causes VCL to specify the 'dbo' schema for metadata requests when a connection is made using the 'sa' user. An SQL `'%'` like pattern can also be used for the user-name and the override-schema-name. For example, `'%.%'` causes the schema to not be specified at all for all users.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.4.4.24 TAdoDbxConnectionStringBuilder.User_Name Property

Pascal

```
[Category(TConnectionStringCategories.Database)]
property User_Name: string;
```

C++

```
[Category(TConnectionStringCategories.Database)]
__property AnsiString User_Name;
```

Description

This is User_Name, a member of class TAdoDbxConnectionStringBuilder.

15.1.1.1.4.4.25 TAdoDbxConnectionStringBuilder.VendorLib Property

Name of a vendor specific client library.

Pascal

```
[Category(TConnectionStringCategories.DriverConfiguration)]
property VendorLib: string;
```

C++

```
[Category(TConnectionStringCategories.DriverConfiguration)]
__property AnsiString VendorLib;
```

Description

Used by DBXDynalink drivers. Name of a vendor specific client library.

See Also

TDBXPropertyNames (see page 10566)

15.1.1.1.5 TAdoDbxDataAdapter Class

Acts as a bridge between a DataSet and the underlying database.

Class Hierarchy

Pascal

```
TAdoDbxDataAdapter = class(DbDataAdapter);
```

C++

```
class TAdoDbxDataAdapter : public DbDataAdapter;
```

File

Borland.Data.AdoDbxClientProvider

Description

The TAdoDbxDataAdapter object is a bridge between the DataSet and the underlying database. It represents a set of data commands and a database connection that are used to fill the DataSet and update the data source. The TAdoDbxDataAdapter object exposes methods to issue SQL commands against the underlying database, as well as others to manage the DataSet object.

The TAdoDbxDataAdapter class is an implementation of the ADO.NET DbDataAdapter class.

You must set the SelectCommand (see page 1825) property with a valid SQL statement before you call the Fill method. This

establishes the structure of the data to be returned from the database to the DataSet. If you choose to update your database with changes in the DataSet, you need to set the appropriate properties, either InsertCommand (see page 1824), UpdateCommand (see page 1825), or DeleteCommand (see page 1823), before you issue the update. For example, if you delete rows from the DataSet and you want this change to be reflected in the database, you need to first set the DeleteCommand (see page 1823) property before starting your update of the database. Otherwise, if you do not set the command, there is no valid SQL command to issue against the database for the update and the update fails. In that case, the database throws an exception, which you can trap.

When the Active property on the TAdoDbxDataAdapter is set to **True**, the DataSet is automatically filled with data. You can see this behavior during design-time if you set the Active property to **True**, then connect a DataGrid component to the DataSet.

The TAdoDbxConnection (see page 1779) associated with the SelectCommand (see page 1825) is used to execute the command specified in the SelectCommand.CommandText. If the TAdoDbxConnection (see page 1779) is already opened, it is used; otherwise, Fill takes care of opening the connection, executing the SQL statement, retrieving the result set, and closing the connection. The TAdoDbxDataAdapter provides InsertCommand (see page 1824), UpdateCommand (see page 1825) and DeleteCommand (see page 1823) properties for resolving DataSet changes back into the data source. You must specify a valid parameterized SQL statement for each of these commands before calling the Update method. Based on the DataSet changes, the Update method executes `INSERT`, `UPDATE`, and `DELETE` commands and persists all the changes to the data source.

For resolving data from a stored procedure or a complex SQL such as joins, the TAdoDbxDataAdapter class's DeleteCommand (see page 1823), UpdateCommand (see page 1825) and InsertCommand (see page 1824) should be explicitly specified and you should use the Update method. Like the GetSchemaTable method in TAdoDbxDataReader (see page 1827), TAdoDbxDataAdapter has a FillSchema method that creates a DataTable and configures the metadata to match the database. Consequently, to enforce integrity constraints in the DataSet, such as primary key and unique key, you must call FillSchema before calling Fill. The TableMappings property allows column names to be mapped from the data source to more meaningful names.

The following C# code shows how to provide and resolve data with a TAdoDbxDataAdapter.

```
static void FillDataAdapter ( TAdoDbxConnection Conn )
 {
      int Rows = 0;

TAdoDbxTransaction Trans = (TAdoDbxTransaction) Conn.BeginTransaction();

    TAdoDbxDataAdapter adapter = new TAdoDbxDataAdapter();
TAdoDbxCommand Comm = new TAdoDbxCommand("SELECT * FROM TESTTABLE", Conn);
    adapter.SelectCommand = Comm;

    DataTableMapping dataTabMap = adapter.TableMappings.Add("Table1","TESTTABLE");
    DataSet ds = new DataSet();
    adapter.FillSchema(ds, SchemaType.Source, "TESTTABLE");
    Rows = adapter.Fill(ds, "TESTTABLE");

    InsertRecord(Conn, adapter, ds.Tables["TESTTABLE"]);
    adapter.Update(ds,"TESTTABLE");
    ds.AcceptChanges();

    Trans.Commit();

   }

static void InsertRecord(TAdoDbxConnection Conn, TAdoDbxDataAdapter adapter, DataTable
dataTable )
  {
    TAdoDbxCommand CommIns = new TAdoDbxCommand("INSERT INTO TESTTABLE VALUES(?)", Conn);
    TAdoDbxParameter param1 = CommIns.Parameters.Add("FCHAR",DbType.StringFixedLength, 10);
    param1.SourceColumn = "FCHAR";
```

```
    adapter.InsertCommand = CommIns;

    //Insert 10 records
    for ( int i=0; i < 10; i++)
    {
        DataRow newRow = dataTable.NewRow();
        newRow["FCHAR"] = "VINA" + i;
        dataTable.Rows.Add(newRow);
    }

  }
static void FillDataAdapter ( TAdoDbxConnection Conn )
 {
      int Rows = 0;

TAdoDbxTransaction Trans = (TAdoDbxTransaction) Conn.BeginTransaction();

    TAdoDbxDataAdapter adapter = new TAdoDbxDataAdapter();
TAdoDbxCommand Comm = new TAdoDbxCommand("SELECT * FROM TESTTABLE", Conn);
    adapter.SelectCommand = Comm;

    DataTableMapping dataTabMap = adapter.TableMappings.Add("Table1","TESTTABLE");
    DataSet ds = new DataSet();
    adapter.FillSchema(ds, SchemaType.Source, "TESTTABLE");
    Rows = adapter.Fill(ds, "TESTTABLE");

    InsertRecord(Conn, adapter, ds.Tables["TESTTABLE"]);
    adapter.Update(ds,"TESTTABLE");
    ds.AcceptChanges();

    Trans.Commit();

   }

static void InsertRecord(TAdoDbxConnection Conn, TAdoDbxDataAdapter adapter, DataTable
dataTable )
  {
TAdoDbxCommand CommIns = new TAdoDbxCommand("INSERT INTO TESTTABLE VALUES(?)", Conn);
TAdoDbxParameter param1 = CommIns.Parameters.Add("FCHAR",DbType.StringFixedLength, 10);
    param1.SourceColumn = "FCHAR";

    adapter.InsertCommand = CommIns;

    //Insert 10 records
    for ( int i=0; i < 10; i++)
    {
        DataRow newRow = dataTable.NewRow();
        newRow["FCHAR"] = "VINA" + i;
        dataTable.Rows.Add(newRow);
    }

  }
```

See Also

DbDataAdapter (MSDN)

ADO.NET Overview

Building an ASP.NET Database Application

Creating a Briefcase Application with DB Web Controls

Building an Application with DB Web Controls

Creating an XML File for DB Web Controls

Adding Aggregate Values with DBWebAggregateControl

TAdoDbxConnection (see page 1779)

TAdoDbxDataReader (see page 1827)

CommandText (see page 1772)

DeleteCommand (see page 1823)

InsertCommand (see page 1824)

SelectCommand (see page 1825)

TableMappings

UpdateCommand (see page 1825)

GetSchemaTable (see page 1840)

Fill

FillSchema

Update

15.1.1.1.5.1 TAdoDbxDataAdapter Members

The following tables list the members exposed by TAdoDbxDataAdapter.

Public Events

	Name	Description
	RowUpdated (see page 1826)	Row has been updated.
	RowUpdating (see page 1826)	Row is being updated.

Public Methods

	Name	Description
	add_RowUpdated (see page 1819)	Add RowUpdated (see page 1826) event.
	add_RowUpdating (see page 1820)	Add RowUpdating (see page 1826) event.
	Create (see page 1818)	This is the overview for the Create constructor overload.
	get_DeleteCommand (see page 1820)	This is get_DeleteCommand, a member of class TAdoDbxDataAdapter.
	get_InsertCommand (see page 1820)	This is get_InsertCommand, a member of class TAdoDbxDataAdapter.
	get_SelectCommand (see page 1821)	This is get_SelectCommand, a member of class TAdoDbxDataAdapter.
	get_UpdateCommand (see page 1821)	This is get_UpdateCommand, a member of class TAdoDbxDataAdapter.
	remove_RowUpdated (see page 1821)	Remove RowUpdated (see page 1826) event.
	remove_RowUpdating (see page 1822)	Remove RowUpdating (see page 1826) event.
	set_DeleteCommand (see page 1822)	This is set_DeleteCommand, a member of class TAdoDbxDataAdapter.
	set_InsertCommand (see page 1822)	This is set_InsertCommand, a member of class TAdoDbxDataAdapter.
	set_SelectCommand (see page 1823)	This is set_SelectCommand, a member of class TAdoDbxDataAdapter.
	set_UpdateCommand (see page 1823)	This is set_UpdateCommand, a member of class TAdoDbxDataAdapter.

Public Properties

	Name	Description
	DeleteCommand (see page 1823)	A TAdoDbxCommand (see page 1763) used during Update to delete records in the data source that correspond to deleted rows in the DataSet.
	InsertCommand (see page 1824)	A TAdoDbxCommand (see page 1763) used during Update to insert records into the database that correspond to new rows in the DataSet.
	SelectCommand (see page 1825)	A TAdoDbxCommand (see page 1763) used during Fill to select records from the database for placement in the DataSet.
	UpdateCommand (see page 1825)	A TAdoDbxCommand (see page 1763) used during Update to update records in the data source that correspond to modified rows in the DataSet.

Strict private Fields

	Name	Description
	FEventRowUpdated (see page 1819)	This is FEventRowUpdated, a member of class TAdoDbxDataAdapter.
	FEventRowUpdating (see page 1819)	This is FEventRowUpdating, a member of class TAdoDbxDataAdapter.

Strict protected Methods

	Name	Description
	OnRowUpdated (see page 1821)	This is OnRowUpdated, a member of class TAdoDbxDataAdapter.
	OnRowUpdating (see page 1821)	This is OnRowUpdating, a member of class TAdoDbxDataAdapter.

15.1.1.1.5.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TAdoDbxDataAdapter.Create () (see page 1818)	Creates an instance of TAdoDbxDataAdapter (see page 1814)
	TAdoDbxDataAdapter.Create (string, string) (see page 1818)	Creates an instance of TAdoDbxDataAdapter (see page 1814)
	TAdoDbxDataAdapter.Create (string, TAdoDbxConnection) (see page 1818)	Creates an instance of TAdoDbxDataAdapter (see page 1814)
	TAdoDbxDataAdapter.Create (TAdoDbxCommand) (see page 1818)	Creates an instance of TAdoDbxDataAdapter (see page 1814)

15.1.1.1.5.2.1 TAdoDbxDataAdapter.Create Constructor ()

Creates an instance of TAdoDbxDataAdapter (see page 1814)

Pascal

```
class constructor Create; overload;
constructor Create(Command: TAdoDbxCommand); overload;
constructor Create(const selectCommandText: string; const selectConnectionString: string);
overload;
constructor Create(const selectCommandText: string; selectConnection: TAdoDbxConnection);
overload;
```

C++

```
__fastcall TAdoDbxDataAdapter();
__fastcall TAdoDbxDataAdapter(TAdoDbxCommand Command);
__fastcall TAdoDbxDataAdapter(const AnsiString selectCommandText, const AnsiString
selectConnectionString);
__fastcall TAdoDbxDataAdapter(const AnsiString selectCommandText, TAdoDbxConnection
selectConnection);
```

Description

This constructor creates and initializes an instance of TAdoDbxDataAdapter (see page 1814).

See Also

Create (MSDN)

15.1.1.1.5.3 TAdoDbxDataAdapter Fields

The fields of the TAdoDbxDataAdapter class are listed here.

Strict private Fields

	Name	Description
	FEventRowUpdated (see page 1819)	This is FEventRowUpdated, a member of class TAdoDbxDataAdapter.
	FEventRowUpdating (see page 1819)	This is FEventRowUpdating, a member of class TAdoDbxDataAdapter.

15.1.1.1.5.3.1 TAdoDbxDataAdapter.FEventRowUpdated Field

Pascal

```
class var FEventRowUpdated: System.Object;
```

C++

```
System.Object FEventRowUpdated;
```

Description

This is FEventRowUpdated, a member of class TAdoDbxDataAdapter.

15.1.1.1.5.3.2 TAdoDbxDataAdapter.FEventRowUpdating Field

Pascal

```
class var FEventRowUpdating: System.Object;
```

C++

```
System.Object FEventRowUpdating;
```

Description

This is FEventRowUpdating, a member of class TAdoDbxDataAdapter.

15.1.1.1.5.4 TAdoDbxDataAdapter Methods

The methods of the TAdoDbxDataAdapter class are listed here.

Public Methods

	Name	Description
	add_RowUpdated (see page 1819)	Add RowUpdated (see page 1826) event.
	add_RowUpdating (see page 1820)	Add RowUpdating (see page 1826) event.
	get_DeleteCommand (see page 1820)	This is get_DeleteCommand, a member of class TAdoDbxDataAdapter.
	get_InsertCommand (see page 1820)	This is get_InsertCommand, a member of class TAdoDbxDataAdapter.
	get_SelectCommand (see page 1821)	This is get_SelectCommand, a member of class TAdoDbxDataAdapter.
	get_UpdateCommand (see page 1821)	This is get_UpdateCommand, a member of class TAdoDbxDataAdapter.
	remove_RowUpdated (see page 1821)	Remove RowUpdated (see page 1826) event.
	remove_RowUpdating (see page 1822)	Remove RowUpdating (see page 1826) event.
	set_DeleteCommand (see page 1822)	This is set_DeleteCommand, a member of class TAdoDbxDataAdapter.
	set_InsertCommand (see page 1822)	This is set_InsertCommand, a member of class TAdoDbxDataAdapter.
	set_SelectCommand (see page 1823)	This is set_SelectCommand, a member of class TAdoDbxDataAdapter.
	set_UpdateCommand (see page 1823)	This is set_UpdateCommand, a member of class TAdoDbxDataAdapter.

Strict protected Methods

	Name	Description
	OnRowUpdated (see page 1821)	This is OnRowUpdated, a member of class TAdoDbxDataAdapter.
	OnRowUpdating (see page 1821)	This is OnRowUpdating, a member of class TAdoDbxDataAdapter.

15.1.1.1.5.4.1 TAdoDbxDataAdapter.add_RowUpdated Method

Add RowUpdated (see page 1826) event.

Pascal

```
procedure add_RowUpdated(const Value: TAdoDbxRowUpdatedEventHandler);
```

C++

```
__fastcall add_RowUpdated(const TAdoDbxRowUpdatedEventHandler Value);
```

Description

add_RowUpdated adds a RowUpdated (see page 1826) event. This is a multicast event that can trigger multiple event handlers.

See Also

remove_RowUpdated (see page 1821)

RowUpdated (see page 1826)

TAdoDbxRowUpdatedEventHandler (see page 1890)

15.1.1.1.5.4.2 TAdoDbxDataAdapter.add_RowUpdating Method

Add RowUpdating (see page 1826) event.

Pascal

```
procedure add_RowUpdating(const Value: TAdoDbxRowUpdatingEventHandler);
```

C++

```
__fastcall add_RowUpdating(const TAdoDbxRowUpdatingEventHandler Value);
```

Description

add_RowUpdating adds a RowUpdating (see page 1826) event. This is a multicast event that can trigger multiple event handlers.

See Also

remove_RowUpdating (see page 1822)

RowUpdating (see page 1826)

TAdoDbxRowUpdatingEventHandler (see page 1890)

15.1.1.1.5.4.3 TAdoDbxDataAdapter.get_DeleteCommand Method

Pascal

```
function get_DeleteCommand: TAdoDbxCommand;
```

C++

```
__fastcall TAdoDbxCommand get_DeleteCommand();
```

Description

This is get_DeleteCommand, a member of class TAdoDbxDataAdapter.

15.1.1.1.5.4.4 TAdoDbxDataAdapter.get_InsertCommand Method

Pascal

```
function get_InsertCommand: TAdoDbxCommand;
```

C++

```
__fastcall TAdoDbxCommand get_InsertCommand();
```

Description

This is get_InsertCommand, a member of class TAdoDbxDataAdapter.

15.1.1.1.5.4.5 TAdoDbxDataAdapter.get_SelectCommand Method

Pascal

```
function get_SelectCommand: TAdoDbxCommand;
```

C++

```
__fastcall TAdoDbxCommand get_SelectCommand();
```

Description

This is get_SelectCommand, a member of class TAdoDbxDataAdapter.

15.1.1.1.5.4.6 TAdoDbxDataAdapter.get_UpdateCommand Method

Pascal

```
function get_UpdateCommand: TAdoDbxCommand;
```

C++

```
__fastcall TAdoDbxCommand get_UpdateCommand();
```

Description

This is get_UpdateCommand, a member of class TAdoDbxDataAdapter.

15.1.1.1.5.4.7 TAdoDbxDataAdapter.OnRowUpdated Method

Pascal

```
procedure OnRowUpdated(value: RowUpdatedEventArgs); override;
```

C++

```
virtual __fastcall OnRowUpdated(RowUpdatedEventArgs value);
```

Description

This is OnRowUpdated, a member of class TAdoDbxDataAdapter.

15.1.1.1.5.4.8 TAdoDbxDataAdapter.OnRowUpdating Method

Pascal

```
procedure OnRowUpdating(value: RowUpdatingEventArgs); override;
```

C++

```
virtual __fastcall OnRowUpdating(RowUpdatingEventArgs value);
```

Description

This is OnRowUpdating, a member of class TAdoDbxDataAdapter.

15.1.1.1.5.4.9 TAdoDbxDataAdapter.remove_RowUpdated Method

Remove RowUpdated (see page 1826) event.

Pascal

```
procedure remove_RowUpdated(const Value: TAdoDbxRowUpdatedEventHandler);
```

C++

```
__fastcall remove_RowUpdated(const TAdoDbxRowUpdatedEventHandler Value);
```

Description

remove_RowUpdated removes a RowUpdated (see page 1826) event. This is a multicast event that can trigger multiple event handlers.

See Also

add_RowUpdated (see page 1819)

RowUpdated (see page 1826)

TAdoDbxRowUpdatedEventHandler (see page 1890)

15.1.1.1.5.4.10 TAdoDbxDataAdapter.remove_RowUpdating Method

Remove RowUpdating (see page 1826) event.

Pascal

```
procedure remove_RowUpdating(const Value: TAdoDbxRowUpdatingEventHandler);
```

C++

```
__fastcall remove_RowUpdating(const TAdoDbxRowUpdatingEventHandler Value);
```

Description

remove_RowUpdating removes a RowUpdating (see page 1826) event. This is a multicast event that can trigger multiple event handlers.

See Also

add_RowUpdating (see page 1820)

RowUpdating (see page 1826)

TAdoDbxRowUpdatingEventHandler (see page 1890)

15.1.1.1.5.4.11 TAdoDbxDataAdapter.set_DeleteCommand Method

Pascal

```
procedure set_DeleteCommand(const Value: TAdoDbxCommand);
```

C++

```
__fastcall set_DeleteCommand(const TAdoDbxCommand Value);
```

Description

This is set_DeleteCommand, a member of class TAdoDbxDataAdapter.

15.1.1.1.5.4.12 TAdoDbxDataAdapter.set_InsertCommand Method

Pascal

```
procedure set_InsertCommand(const Value: TAdoDbxCommand);
```

C++

```
__fastcall set_InsertCommand(const TAdoDbxCommand Value);
```

Description

This is set_InsertCommand, a member of class TAdoDbxDataAdapter.

15.1.1.1.5.4.13 TAdoDbxDataAdapter.set_SelectCommand Method

Pascal

```
procedure set_SelectCommand(const Value: TAdoDbxCommand);
```

C++

```
__fastcall set_SelectCommand(const TAdoDbxCommand Value);
```

Description

This is set_SelectCommand, a member of class TAdoDbxDataAdapter.

15.1.1.1.5.4.14 TAdoDbxDataAdapter.set_UpdateCommand Method

Pascal

```
procedure set_UpdateCommand(const Value: TAdoDbxCommand);
```

C++

```
__fastcall set_UpdateCommand(const TAdoDbxCommand Value);
```

Description

This is set_UpdateCommand, a member of class TAdoDbxDataAdapter.

15.1.1.1.5.5 TAdoDbxDataAdapter Properties

The properties of the TAdoDbxDataAdapter class are listed here.

Public Properties

	Name	Description
	DeleteCommand (see page 1823)	A TAdoDbxCommand (see page 1763) used during Update to delete records in the data source that correspond to deleted rows in the DataSet.
	InsertCommand (see page 1824)	A TAdoDbxCommand (see page 1763) used during Update to insert records into the database that correspond to new rows in the DataSet.
	SelectCommand (see page 1825)	A TAdoDbxCommand (see page 1763) used during Fill to select records from the database for placement in the DataSet.
	UpdateCommand (see page 1825)	A TAdoDbxCommand (see page 1763) used during Update to update records in the data source that correspond to modified rows in the DataSet.

15.1.1.1.5.5.1 TAdoDbxDataAdapter.DeleteCommand Property

A TAdoDbxCommand (see page 1763) used during Update to delete records in the data source that correspond to deleted rows in the DataSet.

Pascal

```
property DeleteCommand: TAdoDbxCommand;
```

C++

```
__property TAdoDbxCommand DeleteCommand;
```

Description

Use this property to store a SQL DELETE statement that you intend to pass to the TAdoDbxDataAdapter (see page 1814) multiple times. This property serves as a shorthand method for storing and calling a command that you intend to use multiple times.

When you assign DeleteCommand to a previously created TAdoDbxCommand (see page 1763), the TAdoDbxCommand (see page 1763) is not cloned. The DeleteCommand maintains a reference to the previously created TAdoDbxCommand (see page 1763) object.

Note: During Update, if you don't set this property and primary key information is present in the DataSet, you can generate the DeleteCommand automatically by setting the SelectCommand (see page 1825) property and using the TAdoDbxCommandBuilder (see page 1776). Additional commands that you do not set are generated by the TAdoDbxCommandBuilder (see page 1776). This generation logic requires key column information to be present in the DataSet. For more information, see the Microsoft .NET SDK online help.

See Also

DeleteCommand (MSDN)

TAdoDbxCommand (see page 1763)

TAdoDbxCommandBuilder (see page 1776)

TAdoDbxDataAdapter (see page 1814)

DataSet

15.1.1.1.5.5.2 TAdoDbxDataAdapter.InsertCommand Property

A TAdoDbxCommand (see page 1763) used during Update to insert records into the database that correspond to new rows in the DataSet.

Pascal

```
property InsertCommand: TAdoDbxCommand;
```

C++

```
__property TAdoDbxCommand InsertCommand;
```

Description

This property contains a SQL INSERT statement that you specify. This property acts as a form of shorthand, allowing you to store an INSERT statement that you expect to use multiple times.

During Update, if you don't set this property and if primary key information is present in the DataSet, you can generate the InsertCommand automatically by setting the SelectCommand (see page 1825) property and using the TAdoDbxCommandBuilder (see page 1776). Then, any additional commands that you do not set are generated by the TAdoDbxCommandBuilder (see page 1776). This generation logic requires key column information to be present in the DataSet.

If execution of this command returns rows, they can be added to the DataSet depending on how you set the UpdatedRowSource property of the TAdoDbxCommand (see page 1763) object.

Note: When you assign InsertCommand to a previously created TAdoDbxCommand (see page 1763), the TAdoDbxCommand (see page 1763) is not cloned. The InsertCommand maintains a reference to the previously created TAdoDbxCommand (see page 1763) object.

See Also

InsertCommand (MSDN)

TAdoDbxCommand (see page 1763)

TAdoDbxCommandBuilder (see page 1776)

DataSet

SelectCommand (see page 1825)

UpdatedRowSource

15.1.1.1.5.5.3 TAdoDbxDataAdapter.SelectCommand Property

A TAdoDbxCommand (see page 1763) used during Fill to select records from the database for placement in the DataSet.

Pascal

```
property SelectCommand: TAdoDbxCommand;
```

C++

```
__property TAdoDbxCommand SelectCommand;
```

Description

This property contains the SQL SELECT statement you assign to it. The property serves as a shorthand method for referring to a SQL statement that you expect to use many times.

Note: When you assign the SelectCommand to a previously created TAdoDbxCommand (see page 1763), the TAdoDbxCommand (see page 1763) is not cloned. The SelectCommand maintains a reference to the previously created TAdoDbxCommand (see page 1763) object. If the SelectCommand does not return any rows, no tables are added to the DataSet, and no exception is raised.

See Also

SelectCommand (MSDN)

TAdoDbxCommand (see page 1763)

Fill

DataSet

15.1.1.1.5.5.4 TAdoDbxDataAdapter.UpdateCommand Property

A TAdoDbxCommand (see page 1763) used during Update to update records in the data source that correspond to modified rows in the DataSet.

Pascal

```
property UpdateCommand: TAdoDbxCommand;
```

C++

```
__property TAdoDbxCommand UpdateCommand;
```

Description

Use this property to get or set the SQL UPDATE statement. You can use this property as a form of shorthand to define a SQL UPDATE statement that you intend to use multiple times.

When UpdateCommand is assigned to a previously created TAdoDbxCommand (see page 1763), the TAdoDbxCommand (see page 1763) is not cloned. The UpdateCommand maintains a reference to the previously created TAdoDbxCommand (see page 1763) object. If execution of this command returns rows, these rows can be merged with the DataSet depending on how you set the UpdatedRowSource property of the TAdoDbxCommand (see page 1763) object.

Note: During Update, if you don't set this property and primary key information is present in the DataSet, the UpdateCommand can be generated automatically if you set the SelectCommand (see page 1825) property and use the TAdoDbxCommandBuilder (see page 1776). Then, any additional commands that you don't set are generated by the TAdoDbxCommandBuilder (see page 1776). This generation logic requires key column information to be present in the DataSet.

See Also

UpdateCommand (MSDN)

TAdoDbxCommand (see page 1763)

TAdoDbxCommandBuilder (see page 1776)

DataSet

SelectCommand (see page 1825)

UpdatedRowSource

15.1.1.1.5.6 TAdoDbxDataAdapter Events

The events of the TAdoDbxDataAdapter class are listed here.

Public Events

	Name	Description
	RowUpdated (see page 1826)	Row has been updated.
	RowUpdating (see page 1826)	Row is being updated.

15.1.1.1.5.6.1 TAdoDbxDataAdapter.RowUpdated Event

Row has been updated.

Pascal

```
property RowUpdated: TAdoDbxRowUpdatedEventHandler;
```

C++

```
__property TAdoDbxRowUpdatedEventHandler RowUpdated;
```

Description

RowUpdated indicates that a row has been updated. This event occurs just before a row is updated. This is a multicast event that can trigger multiple event handlers.

See Also

add_RowUpdated (see page 1819)

remove_RowUpdated (see page 1821)

RowUpdating (see page 1826)

TAdoDbxRowUpdatedEventHandler (see page 1890)

15.1.1.1.5.6.2 TAdoDbxDataAdapter.RowUpdating Event

Row is being updated.

Pascal

```
property RowUpdating: TAdoDbxRowUpdatingEventHandler;
```

C++

```
__property TAdoDbxRowUpdatingEventHandler RowUpdating;
```

Description

RowUpdated (see page 1826) indicates that a row is being updated. This event occurs just before a row begins to be updated. This is a multicast event that can trigger multiple event handlers.

See Also

RowUpdated (see page 1826)

TAdoDbxRowUpdatingEventHandler (see page 1890)

15.1.1.1.6 TAdoDbxDataReader Class

Class to read rows forward-only from a data source.

Class Hierarchy

Pascal

```
TAdoDbxDataReader = class(DbDataReader);
```

C++

```
class TAdoDbxDataReader : public DbDataReader;
```

File

Borland.Data.AdoDbxClientProvider

Description

A TAdoDbxDataReader is returned as a result of a `SELECT` or stored procedure execution from a call to TAdoDbxCommand.ExecuteReader. Because there is no public constructor, you cannot directly instantiate a TAdoDbxDataReader. Instead, obtain the TAdoDbxDataReader through the ExecuteReader method of the TAdoDbxCommand (see page 1763) object.

The TAdoDbxDataReader class is an implementation of the ADO.NET DbDataReader class.

The TAdoDbxDataReader provides a forward-only reader and the associated metadata. TAdoDbxDataReader methods such as GetName (see page 1840), GetDataTypeName (see page 1837), GetFieldType (see page 1838) and GetDataTypeName (see page 1837) provide the metadata.

For all of these methods, you must pass the ordinal of the column, which is zero-based, in the ordinal parameter. Given a column name, GetOrdinal (see page 1840) returns the column ordinal or position in the select list. GetName (see page 1840), GetDataTypeName (see page 1837) and GetFieldType (see page 1838) return the name, SQL datatype name, and the .NET Framework System.Type, respectively, for a particular column. GetSchemaTable (see page 1840) also can be used to retrieve the metadata of a column as a DataTable.

You can call TAdoDbxDataReader.Read (see page 1842) to fetch records one after the other until a **false** value is returned, which indicates that you have hit the EOF. Before accessing individual column values, you can check to see if the data is `NULL` by calling IsDBNull (see page 1841). Then, depending on the datatype, you can call one of the field accessor methods, such as GetInt16 (see page 1839), GetInt32 (see page 1839) or GetFloat (see page 1838).

You can access BLOB data as a byte array or a character array by calling GetBytes (see page 1835) or GetChars (see page 1836). A null buffer passed to these methods returns the size of the BLOB data available. The current implementation of TAdoDbxDataReader does not support fetching BLOB data by specifying offsets.

The initial state of the TAdoDbxDataReader returned from a call to TAdoDbxCommand.ExecuteReader is **open**. Once all of the records have been fetched, you can call the Close (see page 1832) method to free all of the column-related resources. To find out if a TAdoDbxDataReader is closed, you can check the IsClosed (see page 1844) property. If the TAdoDbxDataReader is closed, it returns **True**, otherwise **False**. If the CommandBehavior parameter is **CloseConnection** in ExecuteReader, the TAdoDbxConnection (see page 1779) used for executing the SQL is also closed when the TAdoDbxDataReader is closed. The NextResult (see page 1842) method returns **True** if more results are available from a stored procedure execution.

The following C# code shows how to retrieve data using a TAdoDbxDataReader, assuming that a you already have a valid connection and command.

```
public static void ExecuteCommand ( TAdoDbxCommand Comm )
```

```
{
  Comm.CommandText  =  " SELECT * FROM EMPLOYEE";
   Comm.Prepare();
   TAdoDbxDataReader Reader = Comm.ExecuteReader();
   if ( Reader != null )
   {
      while (Reader.Read())
      {
         ShowData(Reader);
      }
      Reader.Close();
   }
   Comm.Close();
}

public static void ShowData(TAdoDbxDataReader Reader)
{
   long retVal = 0, startIndex = 0;
   int  buffSize = 0;
   char []buffer = null;

   for (Int32 index = 0; index < Reader.FieldCount; index++)
   {
      //Check for NULL
      if ( Reader.IsDBNull(index) )
      {
         Console.Write("NULL");
      }
      else
      {
         Type t =  Reader.GetFieldType(index);
         if (t == typeof(Int16) )
            Console.Write(Reader.GetInt16(index));
         else if ( t == typeof(Int32) )
            Console.Write(Reader.GetInt32(index));
         else if ( t == typeof(String) )
            Console.Write(Reader.GetString(index));
         else if ( t == typeof(float) )
            Console.Write(Reader.GetFloat(index));
         else if ( t == typeof(double) )
            Console.Write(Reader.GetDouble(index));
         else if ( t == typeof(DateTime) )
            Console.Write(Reader.GetDateTime(index));
         else if ( t == typeof(Decimal) )
            Console.Write(Reader.GetDecimal(index));
         else if ( (t == typeof(Byte[])) || (t == typeof(Char[])) )
         {
            String DataType =  Reader.GetDataTypeName(index);
            if (DataType == "Blob")
            {
               retVal = Reader.GetChars(index, 0, null, 0, 0);
               Console.Write("Blob Size = " + retVal);
               buffSize = (int) retVal;

               //Display only character blob data
               if ( retVal > 0 && (t== typeof(Char[])) )
               {
                  buffer = new char[buffSize];
                  startIndex = 0;
                  retVal = Reader.GetChars(index, startIndex, buffer, 0, buffSize);
                  for (int i = 0; i < buffer.Length; i++ )
                     Console.Write(buffer[i]);
             }

          }
```

```
            }
        }
        if (index < Reader.FieldCount -1)
          Console.Write(", ");
    }
    Console.WriteLine();
```

See Also

DbDataReader (MSDN)

CommandBehavior (MSDN)

System.Type type (MSDN)

TAdoDbxCommand (see page 1763)

TAdoDbxConnection (see page 1779)

IsClosed (see page 1844)

ExecuteReader

GetBytes (see page 1835)

GetChars (see page 1836)

GetDataTypeName (see page 1837)

GetFieldType (see page 1838)

GetFloat (see page 1838)

GetInt16 (see page 1839)

GetInt32 (see page 1839)

GetName (see page 1840)

GetOrdinal (see page 1840)

GetSchemaTable (see page 1840)

IsDBNull (see page 1841)

NextResult (see page 1842)

Read (see page 1842)

15.1.1.1.6.1 TAdoDbxDataReader Members

The following tables list the members exposed by TAdoDbxDataReader.

Public Methods

	Name	Description
	Close (see page 1832)	Closes the TAdoDbxDataReader (see page 1827).
	Destroy (see page 1830)	Destroys a TAdoDbxDataReader (see page 1827) instance.
	get_Depth (see page 1833)	This is get_Depth, a member of class TAdoDbxDataReader.
	get_FieldCount (see page 1833)	This is get_FieldCount, a member of class TAdoDbxDataReader.
	get_HasRows (see page 1833)	This is get_HasRows, a member of class TAdoDbxDataReader.
	get_IsClosed (see page 1833)	This is get_IsClosed, a member of class TAdoDbxDataReader.
	get_Item (see page 1834)	This is the overview for the get_Item method overload.

	get_RecordsAffected (see page 1834)	This is get_RecordsAffected, a member of class TAdoDbxDataReader.
	GetBoolean (see page 1834)	Gets the specified column value as a Boolean.
	GetByte (see page 1835)	Gets the specified column value as a byte.
	GetBytes (see page 1835)	Gets column value into an array of bytes.
	GetChar (see page 1835)	Gets the specified column value as a single character.
	GetChars (see page 1836)	Gets column value into an array of chars.
	GetData (see page 1836)	Return a data reader for the column.
	GetDataTypeName (see page 1837)	Gets name of data type for column.
	GetDateTime (see page 1837)	Gets the specified column value as a DateTime.
	GetDecimal (see page 1837)	Gets the specified column value as a Decimal.
	GetDouble (see page 1837)	Gets the specified column value as a double-precision floating point number.
	GetEnumerator (see page 1838)	Returns an IEnumerator that can iterate through rows.
	GetFieldType (see page 1838)	Gets the data type of a specified column.
	GetFloat (see page 1838)	Gets the specified column value as a single-precision floating point number.
	GetGuid (see page 1839)	Gets the specified column value as a GUID.
	GetInt16 (see page 1839)	Gets the specified column value as a 16-bit signed integer.
	GetInt32 (see page 1839)	Gets the specified column value as a 32-bit signed integer.
	GetInt64 (see page 1839)	Gets the specified column value as a 64-bit signed integer.
	GetName (see page 1840)	Gets the specified column name.
	GetOrdinal (see page 1840)	Gets the column ordinal from the name.
	GetSchemaTable (see page 1840)	Returns column metadata of the table.
	GetString (see page 1841)	Gets the specified column value as a String.
	GetValue (see page 1841)	Gets the specified column value as an Object.
	GetValues (see page 1841)	Gets all column values as an array of Objects.
	IsDBNull (see page 1841)	Indicates whether specified column value has a value.
	NextResult (see page 1842)	Advances reader to next result.
	Read (see page 1842)	Advances reader to next row in result set.

Public Properties

	Name	Description
	Depth (see page 1843)	Gets or sets the depth of nesting for the current row.
	FieldCount (see page 1843)	Number of columns in this TAdoDbxDataReader (see page 1827).
	HasRows (see page 1843)	Indicates whether TAdoDbxDataReader (see page 1827) has rows.
	IsClosed (see page 1844)	Returns a Boolean indicating whether the data reader is closed or open.
	Item (see page 1844)	Gets or sets the value of a column in its native format.
	RecordsAffected (see page 1844)	Gets the number of rows changed, inserted, or deleted by execution of SQL statement.

Strict private Fields

	Name	Description
	FCommandBehavior (see page 1831)	This is FCommandBehavior, a member of class TAdoDbxDataReader.
	FDbxCommand (see page 1831)	This is FDbxCommand, a member of class TAdoDbxDataReader.
	FDbxReader (see page 1831)	This is FDbxReader, a member of class TAdoDbxDataReader.
	FPeekAheadRead (see page 1832)	This is FPeekAheadRead, a member of class TAdoDbxDataReader.

15.1.1.1.6.2 TAdoDbxDataReader.Destroy Destructor

Destroys a TAdoDbxDataReader (see page 1827) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdoDbxDataReader();
```

Description

This method destroys a TAdoDbxDataReader (see page 1827) instance, freeing all its resources.

Note: Destroy should not be called directly. Call the Close (see page 1832) or Free method instead.

See Also

Close (see page 1832)

15.1.1.1.6.3 TAdoDbxDataReader Fields

The fields of the TAdoDbxDataReader class are listed here.

Strict private Fields

	Name	Description
	FCommandBehavior (see page 1831)	This is FCommandBehavior, a member of class TAdoDbxDataReader.
	FDbxCommand (see page 1831)	This is FDbxCommand, a member of class TAdoDbxDataReader.
	FDbxReader (see page 1831)	This is FDbxReader, a member of class TAdoDbxDataReader.
	FPeekAheadRead (see page 1832)	This is FPeekAheadRead, a member of class TAdoDbxDataReader.

15.1.1.1.6.3.1 TAdoDbxDataReader.FCommandBehavior Field

Pascal

```
FCommandBehavior: CommandBehavior;
```

C++

```
CommandBehavior FCommandBehavior;
```

Description

This is FCommandBehavior, a member of class TAdoDbxDataReader.

15.1.1.1.6.3.2 TAdoDbxDataReader.FDbxCommand Field

Pascal

```
FDbxCommand: TDBXCommand;
```

C++

```
TDBXCommand FDbxCommand;
```

Description

This is FDbxCommand, a member of class TAdoDbxDataReader.

15.1.1.1.6.3.3 TAdoDbxDataReader.FDbxReader Field

Pascal

```
FDbxReader: TDBXReader;
```

C++

```
TDBXReader FDbxReader;
```

Description

This is FDbxReader, a member of class TAdoDbxDataReader.

15.1.1.1.6.3.4 TAdoDbxDataReader.FPeekAheadRead Field

Pascal

```
FPeekAheadRead: Boolean;
```

C++

```
Boolean FPeekAheadRead;
```

Description

This is FPeekAheadRead, a member of class TAdoDbxDataReader.

15.1.1.1.6.4 TAdoDbxDataReader Methods

The methods of the TAdoDbxDataReader class are listed here.

Public Methods

	Name	Description
	Close (see page 1832)	Closes the TAdoDbxDataReader (see page 1827).
	get_Depth (see page 1833)	This is get_Depth, a member of class TAdoDbxDataReader.
	get_FieldCount (see page 1833)	This is get_FieldCount, a member of class TAdoDbxDataReader.
	get_HasRows (see page 1833)	This is get_HasRows, a member of class TAdoDbxDataReader.
	get_IsClosed (see page 1833)	This is get_IsClosed, a member of class TAdoDbxDataReader.
	get_Item (see page 1834)	This is the overview for the get_Item method overload.
	get_RecordsAffected (see page 1834)	This is get_RecordsAffected, a member of class TAdoDbxDataReader.
	GetBoolean (see page 1834)	Gets the specified column value as a Boolean.
	GetByte (see page 1835)	Gets the specified column value as a byte.
	GetBytes (see page 1835)	Gets column value into an array of bytes.
	GetChar (see page 1835)	Gets the specified column value as a single character.
	GetChars (see page 1836)	Gets column value into an array of chars.
	GetData (see page 1836)	Return a data reader for the column.
	GetDataTypeName (see page 1837)	Gets name of data type for column.
	GetDateTime (see page 1837)	Gets the specified column value as a DateTime.
	GetDecimal (see page 1837)	Gets the specified column value as a Decimal.
	GetDouble (see page 1837)	Gets the specified column value as a double-precision floating point number.
	GetEnumerator (see page 1838)	Returns an IEnumerator that can iterate through rows.
	GetFieldType (see page 1838)	Gets the data type of a specified column.
	GetFloat (see page 1838)	Gets the specified column value as a single-precision floating point number.
	GetGuid (see page 1839)	Gets the specified column value as a GUID.
	GetInt16 (see page 1839)	Gets the specified column value as a 16-bit signed integer.
	GetInt32 (see page 1839)	Gets the specified column value as a 32-bit signed integer.
	GetInt64 (see page 1839)	Gets the specified column value as a 64-bit signed integer.
	GetName (see page 1840)	Gets the specified column name.
	GetOrdinal (see page 1840)	Gets the column ordinal from the name.
	GetSchemaTable (see page 1840)	Returns column metadata of the table.
	GetString (see page 1841)	Gets the specified column value as a String.
	GetValue (see page 1841)	Gets the specified column value as an Object.
	GetValues (see page 1841)	Gets all column values as an array of Objects.
	IsDBNull (see page 1841)	Indicates whether specified column value has a value.
	NextResult (see page 1842)	Advances reader to next result.
	Read (see page 1842)	Advances reader to next row in result set.

15.1.1.1.6.4.1 TAdoDbxDataReader.Close Method

Closes the TAdoDbxDataReader (see page 1827).

Pascal

```
procedure Close; override;
```

C++

```
virtual __fastcall Close();
```

Description

Close closes the TAdoDbxDataReader (see page 1827).

Note: Close should be called as soon as the TAdoDbxDataReader (see page 1827) is no longer needed to free up resources.

See Also

Close (MSDN)

15.1.1.1.6.4.2 TAdoDbxDataReader.get_Depth Method

Pascal

```
function get_Depth: Integer; override;
```

C++

```
virtual __fastcall int get_Depth();
```

Description

This is get_Depth, a member of class TAdoDbxDataReader.

15.1.1.1.6.4.3 TAdoDbxDataReader.get_FieldCount Method

Pascal

```
function get_FieldCount: Integer; override;
```

C++

```
virtual __fastcall int get_FieldCount();
```

Description

This is get_FieldCount, a member of class TAdoDbxDataReader.

15.1.1.1.6.4.4 TAdoDbxDataReader.get_HasRows Method

Pascal

```
function get_HasRows: Boolean; override;
```

C++

```
virtual __fastcall Boolean get_HasRows();
```

Description

This is get_HasRows, a member of class TAdoDbxDataReader.

15.1.1.1.6.4.5 TAdoDbxDataReader.get_IsClosed Method

Pascal

```
function get_IsClosed: Boolean; override;
```

C++

```
virtual __fastcall Boolean get_IsClosed();
```

Description

This is get_IsClosed, a member of class TAdoDbxDataReader.

15.1.1.1.6.4.6 get_Item Method

This is the overview for the get_Item method overload.

Overload List

	Name	Description
	TAdoDbxDataReader.get_Item (Integer) (see page 1834)	This is get_Item, a member of class TAdoDbxDataReader.
	TAdoDbxDataReader.get_Item (String) (see page 1834)	This is get_Item, a member of class TAdoDbxDataReader.

15.1.1.1.6.4.6.1 TAdoDbxDataReader.get_Item Method (Integer)

Pascal

```
function get_Item(Ordinal: Integer): TObject; override; overload;
```

C++

```
virtual __fastcall TObject * get_Item(int Ordinal);
```

Description

This is get_Item, a member of class TAdoDbxDataReader.

15.1.1.1.6.4.6.2 TAdoDbxDataReader.get_Item Method (String)

Pascal

```
function get_Item(Name: String): TObject; override; overload;
```

C++

```
virtual __fastcall TObject * get_Item(AnsiString Name);
```

Description

This is get_Item, a member of class TAdoDbxDataReader.

15.1.1.1.6.4.7 TAdoDbxDataReader.get_RecordsAffected Method

Pascal

```
function get_RecordsAffected: Integer; override;
```

C++

```
virtual __fastcall int get_RecordsAffected();
```

Description

This is get_RecordsAffected, a member of class TAdoDbxDataReader.

15.1.1.1.6.4.8 TAdoDbxDataReader.GetBoolean Method

Gets the specified column value as a Boolean.

Pascal

```
function GetBoolean(Ordinal: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean GetBoolean(int Ordinal);
```

Description

GetBoolean gets the specified zero-based column value as a Boolean.

See Also

GetBoolean (MSDN)

15.1.1.1.6.4.9 TAdoDbxDataReader.GetByte Method

Gets the specified column value as a byte.

Pascal

```
function GetByte(Ordinal: Integer): Byte; override;
```

C++

```
virtual __fastcall Byte GetByte(int Ordinal);
```

Description

GetByte gets the specified zero-based column value as a byte value.

See Also

GetByte (MSDN)

15.1.1.1.6.4.10 TAdoDbxDataReader.GetBytes Method

Gets column value into an array of bytes.

Pascal

```
function GetBytes(Ordinal: Integer; DataOffset: Int64; Bytes: array of byte; BufferOffset:
Integer; Length: Integer): Int64; override;
```

C++

```
virtual __fastcall Int64 GetBytes(int Ordinal, Int64 DataOffset, array of byte Bytes, int
BufferOffset, int Length);
```

Description

GetBytes reads a column value into an array of bytes. It returns the number of bytes read. The parameters are:

Parameter	Description
Ordinal	Zero based column ordinal to read from.
DataOffset	Offset into the data to start reading from.
Bytes	Byte array to read into.
BufferOffset	Byte offset into buffer at which to write data.
Length	Maximum number of bytes to read.

See Also

GetBytes (MSDN)

15.1.1.1.6.4.11 TAdoDbxDataReader.GetChar Method

Gets the specified column value as a single character.

Pascal

```
function GetChar(Ordinal: Integer): Char; override;
```

C++

```
virtual __fastcall Char GetChar(int Ordinal);
```

Description

GetChar gets the specified zero-based column value as a single character.

See Also

GetChar (MSDN)

15.1.1.1.6.4.12 TAdoDbxDataReader.GetChars Method

Gets column value into an array of chars.

Pascal

```
function GetChars(Ordinal: Integer; DataOffset: Int64; Bytes: array of char; BufferOffset:
Integer; Length: Integer): Int64; override;
```

C++

```
virtual __fastcall Int64 GetChars(int Ordinal, Int64 DataOffset, array of char Bytes, int
BufferOffset, int Length);
```

Description

GetChars reads a column value into an array of chars. It returns the number of characters read. The parameters are:

Parameter	Description
Ordinal	Zero based column ordinal to read from.
DataOffset	Offset into the data to start reading from.
Bytes	Char array to read into.
BufferOffset	Byte offset into buffer at which to write data.
Length	Maximum number of chars to read.

See Also

GetChars (MSDN)

15.1.1.1.6.4.13 TAdoDbxDataReader.GetData Method

Return a data reader for the column.

Pascal

```
function GetData(Index: Integer): IDataReader;
```

C++

```
__fastcall IDataReader GetData(int Index);
```

Description

GetData returns an TAdoDbxDataReader (see page 1827) for the zero-based ordinal.

See Also

TAdoDbxDataReader (see page 1827)

GetData (MSDN)

15.1.1.1.6.4.14 TAdoDbxDataReader.GetDataTypeName Method

Gets name of data type for column.

Pascal

```
function GetDataTypeName(Ordinal: Integer): String; override;
```

C++

```
virtual __fastcall AnsiString GetDataTypeName(int Ordinal);
```

Description

GetDataTypeName returns a string representing the data type for a zero-based column number.

See Also

GetDataTypeName (MSDN)

15.1.1.1.6.4.15 TAdoDbxDataReader.GetDateTime Method

Gets the specified column value as a DateTime.

Pascal

```
function GetDateTime(Ordinal: Integer): DateTime; override;
```

C++

```
virtual __fastcall DateTime GetDateTime(int Ordinal);
```

Description

GetDateTime gets the specified zero-based column value as a DateTime object.

See Also

GetDateTime (MSDN)

15.1.1.1.6.4.16 TAdoDbxDataReader.GetDecimal Method

Gets the specified column value as a Decimal.

Pascal

```
function GetDecimal(Ordinal: Integer): Decimal; override;
```

C++

```
virtual __fastcall Decimal GetDecimal(int Ordinal);
```

Description

GetDecimal gets the specified zero-based column value as a Decimal object.

See Also

GetDecimal (MSDN)

15.1.1.1.6.4.17 TAdoDbxDataReader.GetDouble Method

Gets the specified column value as a double-precision floating point number.

Pascal

```
function GetDouble(Ordinal: Integer): double; override;
```

C++

```
virtual __fastcall double GetDouble(int Ordinal);
```

Description

GetDouble gets the specified zero-based column value as a double-precision floating point number.

See Also

GetDouble (MSDN)

15.1.1.1.6.4.18 TAdoDbxDataReader.GetEnumerator Method

Returns an IEnumerator that can iterate through rows.

Pascal

```
function GetEnumerator: IEnumerator; override;
```

C++

```
virtual __fastcall IEnumerator GetEnumerator();
```

Description

GetEnumerator returns an IEnumerator that can iterate through rows of a data reader.

See Also

GetEnumerator (MSDN)

IEnumerator interface (MSDN)

15.1.1.1.6.4.19 TAdoDbxDataReader.GetFieldType Method

Gets the data type of a specified column.

Pascal

```
function GetFieldType(Ordinal: Integer): System; override;
```

C++

```
virtual __fastcall System GetFieldType(int Ordinal);
```

Description

GetFieldType gets the System.Type data type of a specified zero-based column number.

See Also

GetFieldType (MSDN)

System.Type type (MSDN)

15.1.1.1.6.4.20 TAdoDbxDataReader.GetFloat Method

Gets the specified column value as a single-precision floating point number.

Pascal

```
function GetFloat(Ordinal: Integer): single; override;
```

C++

```
virtual __fastcall single GetFloat(int Ordinal);
```

Description

GetFloat gets the specified zero-based column value as a single-precision floating point number.

See Also

GetFloat (MSDN)

15.1.1.1.6.4.21 TAdoDbxDataReader.GetGuid Method

Gets the specified column value as a GUID.

Pascal

```
function GetGuid(Ordinal: Integer): Guid; override;
```

C++

```
virtual __fastcall Guid GetGuid(int Ordinal);
```

Description

GetGuid gets the specified zero-based column value as a globally-unique identifier (GUID).

See Also

GetGuid (MSDN)

15.1.1.1.6.4.22 TAdoDbxDataReader.GetInt16 Method

Gets the specified column value as a 16-bit signed integer.

Pascal

```
function GetInt16(Ordinal: Integer): Smallint; override;
```

C++

```
virtual __fastcall Smallint GetInt16(int Ordinal);
```

Description

GetInt16 gets the specified zero-based column value as a 16-bit signed integer.

See Also

GetInt16 (MSDN)

15.1.1.1.6.4.23 TAdoDbxDataReader.GetInt32 Method

Gets the specified column value as a 32-bit signed integer.

Pascal

```
function GetInt32(Ordinal: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetInt32(int Ordinal);
```

Description

GetInt32 gets the specified zero-based column value as a 32-bit signed integer.

See Also

GetInt32 (MSDN)

15.1.1.1.6.4.24 TAdoDbxDataReader.GetInt64 Method

Gets the specified column value as a 64-bit signed integer.

Pascal

```
function GetInt64(Ordinal: Integer): Int64; override;
```

C++

```
virtual __fastcall Int64 GetInt64(int Ordinal);
```

Description

GetInt64 gets the specified zero-based column value as a 64-bit signed integer.

See Also

GetInt64 (MSDN)

15.1.1.1.6.4.25 TAdoDbxDataReader.GetName Method

Gets the specified column name.

Pascal

```
function GetName(Ordinal: Integer): String; override;
```

C++

```
virtual __fastcall AnsiString GetName(int Ordinal);
```

Description

GetName gets the specified zero-based column's name.

See Also

GetName (MSDN)

15.1.1.1.6.4.26 TAdoDbxDataReader.GetOrdinal Method

Gets the column ordinal from the name.

Pascal

```
function GetOrdinal(Name: String): Integer; override;
```

C++

```
virtual __fastcall int GetOrdinal(AnsiString Name);
```

Description

GetOrdinal gets the column ordinal from its name.

See Also

GetOrdinal (MSDN)

15.1.1.1.6.4.27 TAdoDbxDataReader.GetSchemaTable Method

Returns column metadata of the table.

Pascal

```
function GetSchemaTable: DataTable; override;
```

C++

```
virtual __fastcall DataTable GetSchemaTable();
```

Description

GetSchemaTable returns a DataTable describing the column metadata of the table.

See Also

GetSchemaTable (MSDN)

15.1.1.1.6.4.28 TAdoDbxDataReader.GetString Method

Gets the specified column value as a String.

Pascal

```
function GetString(Ordinal: Integer): String; override;
```

C++

```
virtual __fastcall AnsiString GetString(int Ordinal);
```

Description

GetString gets the specified zero-based column value as a String instance.

See Also

GetString (MSDN)

15.1.1.1.6.4.29 TAdoDbxDataReader.GetValue Method

Gets the specified column value as an Object.

Pascal

```
function GetValue(ordinal: Integer): TObject; override;
```

C++

```
virtual __fastcall TObject * GetValue(int ordinal);
```

Description

GetValue gets the specified zero-based column value in the current row as an Object instance.

See Also

GetValue (MSDN)

15.1.1.1.6.4.30 TAdoDbxDataReader.GetValues Method

Gets all column values as an array of Objects.

Pascal

```
function GetValues(Values: TObjectArray): Integer; override;
```

C++

```
virtual __fastcall int GetValues(TObjectArray Values);
```

Description

GetValues gets the current row's column values as an array of Object instances in the given Object array. Returns the number of array Objects.

See Also

GetValues (MSDN)

15.1.1.1.6.4.31 TAdoDbxDataReader.IsDBNull Method

Indicates whether specified column value has a value.

Pascal

```
function IsDBNull(Ordinal: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean IsDBNull(int Ordinal);
```

Description

For the specified zero-based column, IsDBNull returns true if the object is equivalent to DBNull and false otherwise.

See Also

IsDBNull (MSDN)

DBNull (MSDN)

15.1.1.1.6.4.32 TAdoDbxDataReader.NextResult Method

Advances reader to next result.

Pascal

```
function NextResult: Boolean; override;
```

C++

```
virtual __fastcall Boolean NextResult();
```

Description

NextResult advances to the next result when a provider has returned multiple result sets, such as several rows. Returns true if there is more data; false otherwise.

See Also

NextResult (MSDN)

15.1.1.1.6.4.33 TAdoDbxDataReader.Read Method

Advances reader to next row in result set.

Pascal

```
function Read: Boolean; override;
```

C++

```
virtual __fastcall Boolean Read();
```

Description

Read advances to the next row in the result set. It returns true if there are more rows and false otherwise.

Note: The starting position of a reader is before the first row, so you must call Read to start getting data.

See Also

Read (MSDN)

15.1.1.1.6.5 TAdoDbxDataReader Properties

The properties of the TAdoDbxDataReader class are listed here.

Public Properties

	Name	Description
R	Depth (see page 1843)	Gets or sets the depth of nesting for the current row.

R	FieldCount (see page 1843)	Number of columns in this TAdoDbxDataReader (see page 1827).
R	HasRows (see page 1843)	Indicates whether TAdoDbxDataReader (see page 1827) has rows.
R	IsClosed (see page 1844)	Returns a Boolean indicating whether the data reader is closed or open.
R	Item (see page 1844)	Gets or sets the value of a column in its native format.
R	RecordsAffected (see page 1844)	Gets the number of rows changed, inserted, or deleted by execution of SQL statement.

15.1.1.1.6.5.1 TAdoDbxDataReader.Depth Property

Gets or sets the depth of nesting for the current row.

Pascal

```
property Depth: Integer;
```

C++

```
__property int Depth;
```

Description

Depth returns the nesting depth of the current row in the current table. The outermost table has a depth of 0.

See Also

Depth (MSDN)

15.1.1.1.6.5.2 TAdoDbxDataReader.FieldCount Property

Number of columns in this TAdoDbxDataReader (see page 1827).

Pascal

```
property FieldCount: Integer;
```

C++

```
__property int FieldCount;
```

Description

FieldCount contains the number of columns in the TAdoDbxDataReader (see page 1827).

See Also

FieldCount (MSDN)

15.1.1.1.6.5.3 TAdoDbxDataReader.HasRows Property

Indicates whether TAdoDbxDataReader (see page 1827) has rows.

Pascal

```
property HasRows: Boolean;
```

C++

```
__property Boolean HasRows;
```

Description

HasRows is true when the TAdoDbxDataReader (see page 1827) has one or more rows; false otherwise.

See Also

HasRows (MSDN)

15.1.1.1.6.5.4 TAdoDbxDataReader.IsClosed Property

Returns a Boolean indicating whether the data reader is closed or open.

Pascal

```
property IsClosed: Boolean;
```

C++

```
__property Boolean IsClosed;
```

Description

Returns **True** if the DataReader is closed, **False** if open. The IsClosed and RecordsAffected (see page 1844) properties are the only ones that you can access after the TAdoDbxDataReader (see page 1827) instance is closed.

See Also

IsClosed (MSDN)

RecordsAffected (see page 1844)

15.1.1.1.6.5.5 TAdoDbxDataReader.Item Property

Gets or sets the value of a column in its native format.

Pascal

```
property Item [Name: String]: TObject;
```

C++

```
__property TObject * Item[AnsiString Name];
```

Description

The indexer for the TAdoDbxDataReader (see page 1827) class. You can specify either the string name of the column or the column ordinal. Using the ordinal is more efficient than using the name.

15.1.1.1.6.5.6 TAdoDbxDataReader.RecordsAffected Property

Gets the number of rows changed, inserted, or deleted by execution of SQL statement.

Pascal

```
property RecordsAffected: Integer;
```

C++

```
__property int RecordsAffected;
```

Description

The RecordsAffected property contains the number of rows changed, inserted, or deleted by SQL statements. It contains zero (0) if no rows were affected or the statement failed, and -1 for SELECT statements. The RecordsAffected property is not set until all rows are read and you close the TAdoDbxDataReader (see page 1827). The value of this property is cumulative. For example, if two records are inserted in batch mode, the value of RecordsAffected is two.

The IsClosed (see page 1844) and RecordsAffected properties are the only ones that you can call after the TAdoDbxDataReader (see page 1827) is closed.

Returns	When
0	No rows were affected or the statement failed.
-1	A SELECT statement executed.

Other numbers	The number of rows affected. The value is cumulative and reflects actual records affected by batch mode operations.

See Also

RecordsAffected (MSDN)

IsClosed (see page 1844)

15.1.1.1.7 TAdoDbxDataSourceEnumerator Class

Provides enumerator for finding all data sources on local network.

Class Hierarchy

Pascal

```
TAdoDbxDataSourceEnumerator = class(DbDataSourceEnumerator);
```

C++

```
class TAdoDbxDataSourceEnumerator : public DbDataSourceEnumerator;
```

File

Borland.Data.AdoDbxClientProvider

Description

TAdoDbxDataSourceEnumerator provides an enumerator for all available data sources on the local network. This class also provides a base for enumerating local strongly typed data sources.

The TAdoDbxDataSourceEnumerator class is an implementation of the ADO.NET DbDataSourceEnumerator class.

See Also

DbDataSourceEnumerator (MSDN)

15.1.1.1.8 TAdoDbxException Class

Base class for all data source exceptions.

Class Hierarchy

Pascal

```
TAdoDbxException = class(DbException);
```

C++

```
class TAdoDbxException : public DbException;
```

File

Borland.Data.AdoDbxClientProvider

Description

TAdoDbxException serves as a base class for provider-specific exceptions.

The TAdoDbxException class is an implementation of the ADO.NET DbException class.

See Also

DbException (MSDN)

15.1.1.1.8.1 TAdoDbxException Members

The following tables list the members exposed by TAdoDbxException.

Protected Methods

	Name	Description
	Create (see page 1846)	This is the overview for the Create constructor overload.

15.1.1.1.8.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TAdoDbxException.Create (String) (see page 1846)	This is Create, a member of class TAdoDbxException.
	TAdoDbxException.Create (TDBXError) (see page 1846)	This is Create, a member of class TAdoDbxException.

15.1.1.1.8.2.1 TAdoDbxException.Create Constructor (String)

Pascal

```
constructor Create(Message: String); overload;
```

C++

```
__fastcall TAdoDbxException(AnsiString Message);
```

Description

This is Create, a member of class TAdoDbxException.

15.1.1.1.8.2.2 TAdoDbxException.Create Constructor (TDBXError)

Pascal

```
constructor Create(DBXError: TDBXError); overload;
```

C++

```
__fastcall TAdoDbxException(TDBXError DBXError);
```

Description

This is Create, a member of class TAdoDbxException.

15.1.1.1.9 TAdoDbxExtension Class

Class to get information about a connection.

Class Hierarchy

Borland.Data.AdoDbxClientProvider.TAdoDbxExtension

Pascal

```
TAdoDbxExtension = class;
```

C++

```
class TAdoDbxExtension;
```

File

Borland.Data.AdoDbxClientProvider

Description

TAdoDbxExtension gets information about a TAdoDbxConnection (see page 1779).

See Also

TAdoDbxConnection (see page 1779)

15.1.1.1.9.1 TAdoDbxExtension Members

The following tables list the members exposed by TAdoDbxExtension.

Public Methods

	Name	Description
	Create (see page 1847)	Creates an instance of TAdoDbxExtension (see page 1846)
	GetVendorProperty (see page 1848)	Retrieves vendor specific information.

Strict private Fields

	Name	Description
	FConnection (see page 1847)	This is FConnection, a member of class TAdoDbxExtension.

15.1.1.1.9.2 TAdoDbxExtension.Create Constructor

Creates an instance of TAdoDbxExtension (see page 1846)

Pascal

```
constructor Create(Connection: TAdoDbxConnection);
```

C++

```
__fastcall TAdoDbxExtension(TAdoDbxConnection Connection);
```

Description

This constructor creates and initializes an instance of TAdoDbxExtension (see page 1846).

15.1.1.1.9.3 TAdoDbxExtension Fields

The fields of the TAdoDbxExtension class are listed here.

Strict private Fields

	Name	Description
	FConnection (see page 1847)	This is FConnection, a member of class TAdoDbxExtension.

15.1.1.1.9.3.1 TAdoDbxExtension.FConnection Field

Pascal

```
FConnection: TAdoDbxConnection;
```

C++

```
TAdoDbxConnection FConnection;
```

Description

This is FConnection, a member of class TAdoDbxExtension.

15.1.1.1.9.4 TAdoDbxExtension Methods

The methods of the TAdoDbxExtension class are listed here.

Public Methods

	Name	Description
	GetVendorProperty (see page 1848)	Retrieves vendor specific information.

15.1.1.1.9.4.1 TAdoDbxExtension.GetVendorProperty Method

Retrieves vendor specific information.

Pascal

```
function GetVendorProperty(const Name: WideString): WideString;
```

C++

```
__fastcall BSTR GetVendorProperty(const BSTR Name);
```

Description

GetVendorProperty retrieves vendor specific information. An empty string is returned if the property is not supported.

15.1.1.1.10 TAdoDbxMetaDataColumnNames Class

Column headers for Restrictions DataTable.

Class Hierarchy

Borland.Data.AdoDbxClientProvider.TAdoDbxMetaDataColumnNames

Pascal

```
TAdoDbxMetaDataColumnNames = class;
```

C++

```
class TAdoDbxMetaDataColumnNames;
```

File

Borland.Data.AdoDbxClientProvider

Description

TAdoDbxMetaDataColumnNames are the column headers of the DataTable returned when TAdoDbxConnection.GetSchema is called with DbMetaDataCollectionNames.Restrictions as the first parameter—the Restrictions DataTable.

This table contains the metadata column names.

Constant	Description
RestrictionName (see page 1849)	Name of the restriction.
RestrictionNumber (see page 1849)	Order in which the restriction should be specified in the restrictions array.

RestrictionDefault (see page 1849)	Default value if nil is specified for the restriction.

See Also

TAdoDbxConnection (see page 1779)

GetSchema (see page 1787)

15.1.1.1.10.1 TAdoDbxMetaDataColumnNames Members

The following tables list the members exposed by TAdoDbxMetaDataColumnNames.

Public Constants

	Name	Description
	RestrictionDefault (see page 1849)	This is RestrictionDefault, a member of class TAdoDbxMetaDataColumnNames.
	RestrictionName (see page 1849)	This is RestrictionName, a member of class TAdoDbxMetaDataColumnNames.
	RestrictionNumber (see page 1849)	This is RestrictionNumber, a member of class TAdoDbxMetaDataColumnNames.

15.1.1.1.10.2 TAdoDbxMetaDataColumnNames Constants

The constants of the TAdoDbxMetaDataColumnNames class are listed here.

Public Constants

	Name	Description
	RestrictionDefault (see page 1849)	This is RestrictionDefault, a member of class TAdoDbxMetaDataColumnNames.
	RestrictionName (see page 1849)	This is RestrictionName, a member of class TAdoDbxMetaDataColumnNames.
	RestrictionNumber (see page 1849)	This is RestrictionNumber, a member of class TAdoDbxMetaDataColumnNames.

15.1.1.1.10.2.1 TAdoDbxMetaDataColumnNames.RestrictionDefault Constant

Pascal

```
const RestrictionDefault = 'RestrictionDefault';
```

C++

```
RestrictionDefault = 'RestrictionDefault';
```

Description

This is RestrictionDefault, a member of class TAdoDbxMetaDataColumnNames.

15.1.1.1.10.2.2 TAdoDbxMetaDataColumnNames.RestrictionName Constant

Pascal

```
const RestrictionName = 'RestrictionName';
```

C++

```
RestrictionName = 'RestrictionName';
```

Description

This is RestrictionName, a member of class TAdoDbxMetaDataColumnNames.

15.1.1.1.10.2.3 TAdoDbxMetaDataColumnNames.RestrictionNumber Constant

Pascal

```
const RestrictionNumber = 'RestrictionNumber';
```

C++

```
RestrictionNumber = 'RestrictionNumber';
```

Description

This is RestrictionNumber, a member of class TAdoDbxMetaDataColumnNames.

15.1.1.1.11 TAdoDbxParameter Class

Represents a parameter that is passed to or from a command.

Class Hierarchy

Pascal

```
TAdoDbxParameter = class(DbParameter, ICloneable);
```

C++

```
class TAdoDbxParameter : public DbParameter, public ICloneable;
```

File

Borland.Data.AdoDbxClientProvider

Description

If you are using parameterized SQL or stored procedures, you can pass parameters to them using the TAdoDbxParameter class. The Direction (see page 1861) property determines whether a parameter is an input parameter, an output parameter, or a return value. You can create a new instance of a TAdoDbxParameter object using the simple constructor syntax. You can also specify a number of overloaded values in the constructor.

The TAdoDbxParameter class is an implementation of the ADO.NET DbParameter class.

To pass runtime parameters for a parameterized SQL statement or stored procedure, use the TAdoDbxParameterCollection (see page 1865) class. An empty TAdoDbxParameterCollection (see page 1865) is returned by the TAdoDbxCommand.Parameters property. After successfully preparing the command, parameters are added to the TAdoDbxParameterCollection (see page 1865) by calling the Add method and passing the parameter information such as name, datatype, precision, scale, size, and so on. You can use one of the overloaded Add methods or you can set individual TAdoDbxParameter properties such as Direction (see page 1861), Precision (see page 1862), Scale (see page 1863), DbType (see page 1860) and Size (see page 1863).

Parameter names must be unique, and the parameters must be added to the TAdoDbxParameterCollection (see page 1865) in the same order in which parameter markers appear in the SQL. The Direction (see page 1861) property, by default, is set to **ParameterDirection**.**Input**. In the case of stored procedures, it can be set to **Output**, **InputOutput**, or **ReturnValue** . If the inout parameter is expected to return more data than the input, you should specify the Precision (see page 1862) as a size large enough to hold the output data.

Note: Not all databases support all the different parameter directions.

While specifying the parameter datatype, you can use either System.Type or the ADO.NET logical type and the subtype. You should set the Value (see page 1865) property with the runtime value for all parameters before executing the command. After successful execution, output data is available in the Value (see page 1865) property.

See Also

DbParameter (MSDN)

System.Type type (MSDN)

TAdoDbxCommand (see page 1763)

TAdoDbxDataAdapter (see page 1814)

TAdoDbxParameterCollection (see page 1865)

DbType (see page 1860)

Direction (see page 1861)

Precision (see page 1862)

Scale (see page 1863)

Size (see page 1863)

Value (see page 1865)

15.1.1.1.11.1 TAdoDbxParameter Members

The following tables list the members exposed by TAdoDbxParameter.

Public Methods

	Name	Description
	Clone (see page 1855)	This is Clone, a member of class TAdoDbxParameter.
	Create (see page 1852)	This is the overview for the Create constructor overload.
	Destroy (see page 1853)	Destroys a TAdoDbxParameter (see page 1850) instance.
	get_DbType (see page 1855)	This is get_DbType, a member of class TAdoDbxParameter.
	get_Direction (see page 1855)	This is get_Direction, a member of class TAdoDbxParameter.
	get_IsNullable (see page 1855)	This is get_IsNullable, a member of class TAdoDbxParameter.
	get_ParameterName (see page 1856)	This is get_ParameterName, a member of class TAdoDbxParameter.
	get_Precision (see page 1856)	This is get_Precision, a member of class TAdoDbxParameter.
	get_Scale (see page 1856)	This is get_Scale, a member of class TAdoDbxParameter.
	get_Size (see page 1856)	This is get_Size, a member of class TAdoDbxParameter.
	get_SourceColumn (see page 1856)	This is get_SourceColumn, a member of class TAdoDbxParameter.
	get_SourceColumnNullMapping (see page 1857)	This is get_SourceColumnNullMapping, a member of class TAdoDbxParameter.
	get_SourceVersion (see page 1857)	This is get_SourceVersion, a member of class TAdoDbxParameter.
	get_Value (see page 1857)	This is get_Value, a member of class TAdoDbxParameter.
	ResetDbType (see page 1857)	Resets the DBType property to original settings.
	set_DbType (see page 1857)	This is set_DbType, a member of class TAdoDbxParameter.
	set_Direction (see page 1858)	This is set_Direction, a member of class TAdoDbxParameter.
	set_IsNullable (see page 1858)	This is set_IsNullable, a member of class TAdoDbxParameter.
	set_ParameterName (see page 1858)	This is set_ParameterName, a member of class TAdoDbxParameter.
	set_Precision (see page 1858)	This is set_Precision, a member of class TAdoDbxParameter.
	set_Scale (see page 1858)	This is set_Scale, a member of class TAdoDbxParameter.
	set_Size (see page 1859)	This is set_Size, a member of class TAdoDbxParameter.
	set_SourceColumn (see page 1859)	This is set_SourceColumn, a member of class TAdoDbxParameter.
	set_SourceColumnNullMapping (see page 1859)	This is set_SourceColumnNullMapping, a member of class TAdoDbxParameter.
	set_SourceVersion (see page 1859)	This is set_SourceVersion, a member of class TAdoDbxParameter.
	set_Value (see page 1859)	This is set_Value, a member of class TAdoDbxParameter.

Published Properties

	Name	Description
	DbType (see page 1860)	Gets or sets the data type of the parameter.
	DbxDataType (see page 1860)	TDBXDataTypes data type
	DbxSubType (see page 1861)	TDBXDataTypessub type

	Direction (see page 1861)	Gets or sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.
	IsNullable (see page 1862)	Gets or sets the value that determines whether or not a parameter value can be null.
	ParameterName (see page 1862)	Gets and sets the parameter name of the given parameter in a parameter collection.
	Precision (see page 1862)	Gets or sets the maximum number of digits used to represent the Value (see page 1865) property.
	Scale (see page 1863)	Gets and sets the scale of the Value (see page 1865) property, which contains either a numeric or decimal value.
	Size (see page 1863)	Gets and sets the size of the parameter value in bytes.
	SourceColumn (see page 1863)	The name of the source column mapped to the DataSet. The default is an empty string.
	SourceColumnNullMapping (see page 1864)	Sets or gets a value indicating whether the column can be null.
	SourceVersion (see page 1864)	Gets or sets the DataRowVersion to use when loading Value (see page 1865).
	Value (see page 1865)	Gets or sets the value of the parameter.

Strict private Fields

	Name	Description
	FConnection (see page 1853)	This is FConnection, a member of class TAdoDbxParameter.
	FDbxContext (see page 1853)	This is FDbxContext, a member of class TAdoDbxParameter.
	FSourceColumn (see page 1854)	This is FSourceColumn, a member of class TAdoDbxParameter.
	FSourceColumnNullMapping (see page 1854)	This is FSourceColumnNullMapping, a member of class TAdoDbxParameter.
	FSourceVersion (see page 1854)	This is FSourceVersion, a member of class TAdoDbxParameter.

15.1.1.1.11.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TAdoDbxParameter.Create (TDBXContext) (see page 1852)	Creates a new instance of the TAdoDbxParameter (see page 1850).
	TAdoDbxParameter.Create (TDBXParameterEx, TAdoDbxConnection) (see page 1852)	Creates a new instance of the TAdoDbxParameter (see page 1850).

15.1.1.1.11.2.1 TAdoDbxParameter.Create Constructor (TDBXContext)

Creates a new instance of the TAdoDbxParameter (see page 1850).

Pascal

```
constructor Create(DbxContext: TDBXContext); overload;
constructor Create(DbxParameter: TDBXParameterEx; Connection: TAdoDbxConnection); overload;
```

C++

```
__fastcall TAdoDbxParameter(TDBXContext DbxContext);
__fastcall TAdoDbxParameter(TDBXParameterEx DbxParameter, TAdoDbxConnection Connection);
```

Description

Creates a new instance of the TAdoDbxParameter (see page 1850) class, which represents a parameter that is passed to or from a command.

If you are using stored procedures, you can pass parameters to them using the TAdoDbxParameter (see page 1850) class. You can also retrieve values from a database column to use as a parameter to a command. The Direction (see page 1861) property determines whether a parameter is an input parameter, an output parameter, or a return value.

See Also

Create (MSDN)

Direction (see page 1861)

15.1.1.1.11.3 TAdoDbxParameter.Destroy Destructor

Destroys a TAdoDbxParameter (see page 1850) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdoDbxParameter();
```

Description

This method destroys a TAdoDbxParameter (see page 1850) instance, freeing all its resources.

See Also

Create (see page 1852)

15.1.1.1.11.4 TAdoDbxParameter Fields

The fields of the TAdoDbxParameter class are listed here.

Strict private Fields

	Name	Description
	FConnection (see page 1853)	This is FConnection, a member of class TAdoDbxParameter.
	FDbxContext (see page 1853)	This is FDbxContext, a member of class TAdoDbxParameter.
	FSourceColumn (see page 1854)	This is FSourceColumn, a member of class TAdoDbxParameter.
	FSourceColumnNullMapping (see page 1854)	This is FSourceColumnNullMapping, a member of class TAdoDbxParameter.
	FSourceVersion (see page 1854)	This is FSourceVersion, a member of class TAdoDbxParameter.

15.1.1.1.11.4.1 TAdoDbxParameter.FConnection Field

Pascal

```
FConnection: TAdoDbxConnection;
```

C++

```
TAdoDbxConnection FConnection;
```

Description

This is FConnection, a member of class TAdoDbxParameter.

15.1.1.1.11.4.2 TAdoDbxParameter.FDbxContext Field

Pascal

```
FDbxContext: TDbxContext;
```

C++

```
TDbxContext FDbxContext;
```

Description

This is FDbxContext, a member of class TAdoDbxParameter.

15.1.1.1.11.4.3 TAdoDbxParameter.FSourceColumn Field

Pascal

```
FSourceColumn: String;
```

C++

```
AnsiString FSourceColumn;
```

Description

This is FSourceColumn, a member of class TAdoDbxParameter.

15.1.1.1.11.4.4 TAdoDbxParameter.FSourceColumnNullMapping Field

Pascal

```
FSourceColumnNullMapping: Boolean;
```

C++

```
Boolean FSourceColumnNullMapping;
```

Description

This is FSourceColumnNullMapping, a member of class TAdoDbxParameter.

15.1.1.1.11.4.5 TAdoDbxParameter.FSourceVersion Field

Pascal

```
FSourceVersion: DataRowVersion;
```

C++

```
DataRowVersion FSourceVersion;
```

Description

This is FSourceVersion, a member of class TAdoDbxParameter.

15.1.1.1.11.5 TAdoDbxParameter Methods

The methods of the TAdoDbxParameter class are listed here.

Public Methods

	Name	Description
	Clone (see page 1855)	This is Clone, a member of class TAdoDbxParameter.
	get_DbType (see page 1855)	This is get_DbType, a member of class TAdoDbxParameter.
	get_Direction (see page 1855)	This is get_Direction, a member of class TAdoDbxParameter.
	get_IsNullable (see page 1855)	This is get_IsNullable, a member of class TAdoDbxParameter.
	get_ParameterName (see page 1856)	This is get_ParameterName, a member of class TAdoDbxParameter.
	get_Precision (see page 1856)	This is get_Precision, a member of class TAdoDbxParameter.
	get_Scale (see page 1856)	This is get_Scale, a member of class TAdoDbxParameter.
	get_Size (see page 1856)	This is get_Size, a member of class TAdoDbxParameter.
	get_SourceColumn (see page 1856)	This is get_SourceColumn, a member of class TAdoDbxParameter.
	get_SourceColumnNullMapping (see page 1857)	This is get_SourceColumnNullMapping, a member of class TAdoDbxParameter.
	get_SourceVersion (see page 1857)	This is get_SourceVersion, a member of class TAdoDbxParameter.
	get_Value (see page 1857)	This is get_Value, a member of class TAdoDbxParameter.
	ResetDbType (see page 1857)	Resets the DBType property to original settings.
	set_DbType (see page 1857)	This is set_DbType, a member of class TAdoDbxParameter.
	set_Direction (see page 1858)	This is set_Direction, a member of class TAdoDbxParameter.

	set_IsNullable (see page 1858)	This is set_IsNullable, a member of class TAdoDbxParameter.
	set_ParameterName (see page 1858)	This is set_ParameterName, a member of class TAdoDbxParameter.
	set_Precision (see page 1858)	This is set_Precision, a member of class TAdoDbxParameter.
	set_Scale (see page 1858)	This is set_Scale, a member of class TAdoDbxParameter.
	set_Size (see page 1859)	This is set_Size, a member of class TAdoDbxParameter.
	set_SourceColumn (see page 1859)	This is set_SourceColumn, a member of class TAdoDbxParameter.
	set_SourceColumnNullMapping (see page 1859)	This is set_SourceColumnNullMapping, a member of class TAdoDbxParameter.
	set_SourceVersion (see page 1859)	This is set_SourceVersion, a member of class TAdoDbxParameter.
	set_Value (see page 1859)	This is set_Value, a member of class TAdoDbxParameter.

15.1.1.1.11.5.1 TAdoDbxParameter.Clone Method

Pascal

```
function Clone: TObject; virtual;
```

C++

```
virtual __fastcall TObject * Clone();
```

Description

This is Clone, a member of class TAdoDbxParameter.

15.1.1.1.11.5.2 TAdoDbxParameter.get_DbType Method

Pascal

```
function get_DbType: DbType; override;
```

C++

```
virtual __fastcall DbType get_DbType();
```

Description

This is get_DbType, a member of class TAdoDbxParameter.

15.1.1.1.11.5.3 TAdoDbxParameter.get_Direction Method

Pascal

```
function get_Direction: ParameterDirection; override;
```

C++

```
virtual __fastcall ParameterDirection get_Direction();
```

Description

This is get_Direction, a member of class TAdoDbxParameter.

15.1.1.1.11.5.4 TAdoDbxParameter.get_IsNullable Method

Pascal

```
function get_IsNullable: Boolean; override;
```

C++

```
virtual __fastcall Boolean get_IsNullable();
```

Description

This is get_IsNullable, a member of class TAdoDbxParameter.

15.1.1.1.11.5.5 TAdoDbxParameter.get_ParameterName Method

Pascal

```
function get_ParameterName: String; override;
```

C++

```
virtual __fastcall AnsiString get_ParameterName();
```

Description

This is get_ParameterName, a member of class TAdoDbxParameter.

15.1.1.1.11.5.6 TAdoDbxParameter.get_Precision Method

Pascal

```
function get_Precision: Byte;
```

C++

```
__fastcall Byte get_Precision();
```

Description

This is get_Precision, a member of class TAdoDbxParameter.

15.1.1.1.11.5.7 TAdoDbxParameter.get_Scale Method

Pascal

```
function get_Scale: Byte;
```

C++

```
__fastcall Byte get_Scale();
```

Description

This is get_Scale, a member of class TAdoDbxParameter.

15.1.1.1.11.5.8 TAdoDbxParameter.get_Size Method

Pascal

```
function get_Size: Integer; override;
```

C++

```
virtual __fastcall int get_Size();
```

Description

This is get_Size, a member of class TAdoDbxParameter.

15.1.1.1.11.5.9 TAdoDbxParameter.get_SourceColumn Method

Pascal

```
function get_SourceColumn: String; override;
```

C++

```
virtual __fastcall AnsiString get_SourceColumn();
```

Description

This is get_SourceColumn, a member of class TAdoDbxParameter.

15.1.1.1.11.5.10 TAdoDbxParameter.get_SourceColumnNullMapping Method

Pascal

```
function get_SourceColumnNullMapping: Boolean; override;
```

C++

```
virtual __fastcall Boolean get_SourceColumnNullMapping();
```

Description

This is get_SourceColumnNullMapping, a member of class TAdoDbxParameter.

15.1.1.1.11.5.11 TAdoDbxParameter.get_SourceVersion Method

Pascal

```
function get_SourceVersion: DataRowVersion; override;
```

C++

```
virtual __fastcall DataRowVersion get_SourceVersion();
```

Description

This is get_SourceVersion, a member of class TAdoDbxParameter.

15.1.1.1.11.5.12 TAdoDbxParameter.get_Value Method

Pascal

```
function get_Value: TObject; override;
```

C++

```
virtual __fastcall TObject * get_Value();
```

Description

This is get_Value, a member of class TAdoDbxParameter.

15.1.1.1.11.5.13 TAdoDbxParameter.ResetDbType Method

Resets the DBType property to original settings.

Pascal

```
procedure ResetDbType; override;
```

C++

```
virtual __fastcall ResetDbType();
```

Description

ResetDbType resets the DBType property to original settings.

See Also

ResetDbType (MSDN)

DbType (see page 1860)

15.1.1.1.11.5.14 TAdoDbxParameter.set_DbType Method

Pascal

```
procedure set_DbType(DbType: DbType); override;
```

C++

```
virtual __fastcall set_DbType(DbType DbType);
```

Description

This is set_DbType, a member of class TAdoDbxParameter.

15.1.1.1.11.5.15 TAdoDbxParameter.set_Direction Method

Pascal

```
procedure set_Direction(Direction: ParameterDirection); override;
```

C++

```
virtual __fastcall set_Direction(ParameterDirection Direction);
```

Description

This is set_Direction, a member of class TAdoDbxParameter.

15.1.1.1.11.5.16 TAdoDbxParameter.set_IsNullable Method

Pascal

```
procedure set_IsNullable(IsNullable: Boolean); override;
```

C++

```
virtual __fastcall set_IsNullable(Boolean IsNullable);
```

Description

This is set_IsNullable, a member of class TAdoDbxParameter.

15.1.1.1.11.5.17 TAdoDbxParameter.set_ParameterName Method

Pascal

```
procedure set_ParameterName(ParameterName: String); override;
```

C++

```
virtual __fastcall set_ParameterName(AnsiString ParameterName);
```

Description

This is set_ParameterName, a member of class TAdoDbxParameter.

15.1.1.1.11.5.18 TAdoDbxParameter.set_Precision Method

Pascal

```
procedure set_Precision(Precision: Byte);
```

C++

```
__fastcall set_Precision(Byte Precision);
```

Description

This is set_Precision, a member of class TAdoDbxParameter.

15.1.1.1.11.5.19 TAdoDbxParameter.set_Scale Method

Pascal

```
procedure set_Scale(Scale: Byte);
```

C++

```
__fastcall set_Scale(Byte Scale);
```

Description

This is set_Scale, a member of class TAdoDbxParameter.

15.1.1.1.11.5.20 TAdoDbxParameter.set_Size Method

Pascal

```
procedure set_Size(Size: Integer); override;
```

C++

```
virtual __fastcall set_Size(int Size);
```

Description

This is set_Size, a member of class TAdoDbxParameter.

15.1.1.1.11.5.21 TAdoDbxParameter.set_SourceColumn Method

Pascal

```
procedure set_SourceColumn(SourceColumn: String); override;
```

C++

```
virtual __fastcall set_SourceColumn(AnsiString SourceColumn);
```

Description

This is set_SourceColumn, a member of class TAdoDbxParameter.

15.1.1.1.11.5.22 TAdoDbxParameter.set_SourceColumnNullMapping Method

Pascal

```
procedure set_SourceColumnNullMapping(SourceColumnNullMapping: Boolean); override;
```

C++

```
virtual __fastcall set_SourceColumnNullMapping(Boolean SourceColumnNullMapping);
```

Description

This is set_SourceColumnNullMapping, a member of class TAdoDbxParameter.

15.1.1.1.11.5.23 TAdoDbxParameter.set_SourceVersion Method

Pascal

```
procedure set_SourceVersion(SourceVersion: DataRowVersion); override;
```

C++

```
virtual __fastcall set_SourceVersion(DataRowVersion SourceVersion);
```

Description

This is set_SourceVersion, a member of class TAdoDbxParameter.

15.1.1.1.11.5.24 TAdoDbxParameter.set_Value Method

Pascal

```
procedure set_Value(Value: TObject); override;
```

C++

```
virtual __fastcall set_Value(TObject * Value);
```

Description

This is set_Value, a member of class TAdoDbxParameter.

15.1.1.1.11.6 TAdoDbxParameter Properties

The properties of the TAdoDbxParameter class are listed here.

Published Properties

	Name	Description
	DbType (see page 1860)	Gets or sets the data type of the parameter.
	DbxDataType (see page 1860)	TDBXDataTypes data type
	DbxSubType (see page 1861)	TDBXDataTypessub type
	Direction (see page 1861)	Gets or sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.
	IsNullable (see page 1862)	Gets or sets the value that determines whether or not a parameter value can be null.
	ParameterName (see page 1862)	Gets and sets the parameter name of the given parameter in a parameter collection.
	Precision (see page 1862)	Gets or sets the maximum number of digits used to represent the Value (see page 1865) property.
	Scale (see page 1863)	Gets and sets the scale of the Value (see page 1865) property, which contains either a numeric or decimal value.
	Size (see page 1863)	Gets and sets the size of the parameter value in bytes.
	SourceColumn (see page 1863)	The name of the source column mapped to the DataSet. The default is an empty string.
	SourceColumnNullMapping (see page 1864)	Sets or gets a value indicating whether the column can be null.
	SourceVersion (see page 1864)	Gets or sets the DataRowVersion to use when loading Value (see page 1865).
	Value (see page 1865)	Gets or sets the value of the parameter.

15.1.1.1.11.6.1 TAdoDbxParameter.DbType Property

Gets or sets the data type of the parameter.

Pascal

```
property DbType: DbType;
```

C++

```
__property DbType DbType;
```

Description

DbType is the parameter type, specified by a DbType enumeration. If the type is not specified, you can infer the .NET data provider type of the parameter from the Value (see page 1865) property of the TAdoDbxParameter (see page 1850) object though it is not explicitly set.

See Also

DbType (MSDN)

DbType enumeration (MSDN)

15.1.1.1.11.6.2 TAdoDbxParameter.DbxDataType Property

TDBXDataTypes data type

Pascal

```
property DbxDataType: Integer;
```

C++

```
__property int DbxDataType;
```

Description

DbxDataType gets and sets the TDBXDataTypes data type.

See Also

TDBXDataTypes (see page 10447)

15.1.1.1.11.6.3 TAdoDbxParameter.DbxSubType Property

TDBXDataTypessub type

Pascal

```
property DbxSubType: Integer;
```

C++

```
__property int DbxSubType;
```

Description

DbxSubType gets and sets the TDBXDataTypes sub type.

See Also

TDBXDataTypes (see page 10447)

15.1.1.1.11.6.4 TAdoDbxParameter.Direction Property

Gets or sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.

Pascal

```
property Direction: ParameterDirection;
```

C++

```
__property ParameterDirection Direction;
```

Description

Direction is one of the ParameterDirection enumeration values.

If you want to use output or return parameters with stored procedures, you need to set this property. If the direction is output, and execution of the associated TAdoDbxCommand (see page 1763) does not return a value, the TAdoDbxParameter (see page 1850) contains a null value. After the last row from the last result set is read, *Output*, *InputOutput*, and *ReturnValue* parameters are updated. The default is *Input*.

Constant	Description
Input	The parameter is an input parameter.
InputOutput	The parameter can accept both input and output.
Output	The parameter is an output parameter.
ReturnValue	The parameter represents a return value from an operation such as a stored procedure, built-in function, or user-defined function.

See Also

Direction (MSDN)

ParameterDirection (MSDN)

15.1.1.1.11.6.5 TAdoDbxParameter.IsNullable Property

Gets or sets the value that determines whether or not a parameter value can be null.

Pascal

```
property IsNullable: Boolean;
```

C++

```
__property Boolean IsNullable;
```

Description

Set to **True** if null values are accepted, otherwise, set to **False**. The default is **False**.

See Also

IsNullable (MSDN)

15.1.1.1.11.6.6 TAdoDbxParameter.ParameterName Property

Gets and sets the parameter name of the given parameter in a parameter collection.

Pascal

```
property ParameterName: String;
```

C++

```
__property AnsiString ParameterName;
```

Description

ParameterName defaults to an empty string.

See Also

ParameterName (MSDN)

15.1.1.1.11.6.7 TAdoDbxParameter.Precision Property

Gets or sets the maximum number of digits used to represent the Value (see page 1865) property.

Pascal

```
property Precision: Byte;
```

C++

```
__property Byte Precision;
```

Description

Gets or sets the maximum number of digits used to represent the Value (see page 1865) property. The default is 0, which indicates that the data provider sets the precision for the Value (see page 1865) property.

See Also

Value (see page 1865)

15.1.1.1.11.6.8 TAdoDbxParameter.Scale Property

Gets and sets the scale of the Value (see page 1865) property, which contains either a numeric or decimal value.

Pascal

```
property Scale: Byte;
```

C++

```
__property Byte Scale;
```

Description

Gets and sets the number of digits to the right of the decimal for the numeric or decimal value of the Value (see page 1865) property. The default is 0.

See Also

Value (see page 1865)

15.1.1.1.11.6.9 TAdoDbxParameter.Size Property

Gets and sets the size of the parameter value in bytes.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

Size is an Int32 type used for binary and string types. Size is ignored for fixed length data types. For Unicode, Size is the number of characters, not bytes. Size does not include any string terminating characters.

Size and DbType (see page 1860) can be inferred if Value (see page 1865) is set, but they are not explicitly set by setting Value (see page 1865).

Warning: If a parameter's Direction (see page 1861) is InputOutput

, **Output**, or **ReturnValue**, you must set Size.

Warning: If the size of a parameter's Value (see page 1865) exceedsSize, Value (see page 1865) data is truncated to Size.

See Also

Size (MSDN)

DbType (see page 1860)

Direction (see page 1861)

Value (see page 1865)

15.1.1.1.11.6.10 TAdoDbxParameter.SourceColumn Property

The name of the source column mapped to the DataSet. The default is an empty string.

Pascal

```
property SourceColumn: String;
```

C++

```
__property AnsiString SourceColumn;
```

Description

Use this property and the SourceVersion (see page 1864) property when you are submitting pending changes to the database and you want to validate the version of the data in the target row. To return a value from a particular column in the database, set the SourceColumn property to the name of the column. If the Direction (see page 1861) property is set to *Input*, the value is taken from the DataSet instead of the database. If the Direction (see page 1861) property is set to *Output*, the value is taken from the database.

See Also

SourceColumn (MSDN)

Direction (see page 1861)

SourceVersion (see page 1864)

15.1.1.1.11.6.11 TAdoDbxParameter.SourceColumnNullMapping Property

Sets or gets a value indicating whether the column can be null.

Pascal

```
property SourceColumnNullMapping: Boolean;
```

C++

```
__property Boolean SourceColumnNullMapping;
```

Description

SourceColumnNullMapping sets or gets a value indicating whether the column can be set to null. This value allows TAdoDbxCommandBuilder (see page 1776) to generate correct Update statements for nullable columns.

See Also

SourceColumnNullMapping (MSDN)

15.1.1.1.11.6.12 TAdoDbxParameter.SourceVersion Property

Gets or sets the DataRowVersion to use when loading Value (see page 1865).

Pascal

```
property SourceVersion: DataRowVersion;
```

C++

```
__property DataRowVersion SourceVersion;
```

Description

Use this property and the SourceColumn (see page 1863) property when you are submitting pending changes to the database and you want to validate the version of the data in the target row. Used by UpdateCommand during an Update operation to determine whether the parameter value is set to **Current** or **Original**. This allows primary keys to be updated. This property is ignored by the InsertCommand and DeleteCommand properties. This property is set to the version of the DataRow used by the Item property or the GetChildRows method of the DataRow object.

The following table lists the allowable SourceVersion values, which are DataRowVersion values.

Member	Description
Current	Current value of the column. Default.
Default	Uses the DefaultValue of the column.
Original	Original value of the column.

Proposed	Uses a Proposed value.

See Also

SourceVersion (MSDN)

Item (MSDN)

GetChildRows (MSDN)

DataRow (MSDN)

DataRowVersion enumeration (MSDN)

UpdateCommand (see page 1825)

Value (see page 1865)

Update

15.1.1.1.11.6.13 TAdoDbxParameter.Value Property

Gets or sets the value of the parameter.

Pascal

```
[TypeConverter(typeof(StringConverter))]
property Value: TObject;
```

C++

```
[TypeConverter(typeof(StringConverter))]
__property TObject * Value;
```

Description

Value contains an Object data type. You need to convert the value if you want to store it in another data type. For input parameters, the value is bound to the TAdoDbxCommand (see page 1763) that is sent to the server. If the parameter's Direction (see page 1861) is **Output** or **ReturnValue**, the value is set on completion of the TAdoDbxCommand (see page 1763) and after the TAdoDbxDataReader (see page 1827) is closed. When sending a null parameter value to the server, you must specify DBNull, not null. The null value in the system is an empty object that has no value. DBNull is used to represent null values.

If the application specifies the DbType (see page 1860), the value is converted to that type when the provider sends the data to the server. DbType (see page 1860) can be inferred from Value though it is not explicitly set by setting Value.

See Also

Value (MSDN)

DBNull (MSDN)

TAdoDbxCommand (see page 1763)

TAdoDbxDataReader (see page 1827)

DbType (see page 1860)

Direction (see page 1861)

15.1.1.1.12 TAdoDbxParameterCollection Class

Collects parameters into a .NET collection object that can be read and manipulated.

Class Hierarchy

Pascal

```
TAdoDbxParameterCollection = class(DbParameterCollection);
```

C++

```
class TAdoDbxParameterCollection : public DbParameterCollection;
```

File

Borland.Data.AdoDbxClientProvider

Description

Serves as a logical storage and management structure for a group of parameters. Collections, which can be organized as lists, arrays, and hashtables, can be read from and sometimes written to, depending on the properties of the collection. You can navigate the collection tree to locate parameters or to add parameters.

The TAdoDbxParameterCollection class is an implementation of the ADO.NET DbParameterCollection class.

See Also

DbParameterCollection (MSDN)

15.1.1.1.12.1 TAdoDbxParameterCollection Members

The following tables list the members exposed by TAdoDbxParameterCollection.

Public Methods

	Name	Description
	Add (see page 1868)	Adds a TAdoDbxParameter (see page 1850) parameter object to the parameter collection.
	AddRange (see page 1868)	Copies an array of parameters to a collection.
	Clear (see page 1869)	Removes all parameters from collection.
	Contains (see page 1869)	This is the overview for the Contains method overload.
	CopyTo (see page 1869)	Copies an array of items to collection.
	Destroy (see page 1867)	Destroys a TAdoDbxParameterCollection (see page 1865) instance.
	get_Count (see page 1870)	This is get_Count, a member of class TAdoDbxParameterCollection.
	get_IsFixedSize (see page 1870)	This is get_IsFixedSize, a member of class TAdoDbxParameterCollection.
	get_IsReadOnly (see page 1870)	This is get_IsReadOnly, a member of class TAdoDbxParameterCollection.
	get_IsSynchronized (see page 1870)	This is get_IsSynchronized, a member of class TAdoDbxParameterCollection.
	get_SyncRoot (see page 1871)	This is get_SyncRoot, a member of class TAdoDbxParameterCollection.
	GetEnumerator (see page 1871)	Returns an enumerator for the collection.
	IndexOf (see page 1872)	This is the overview for the IndexOf method overload.
	Insert (see page 1872)	Inserts a parameter in collection.
	Remove (see page 1873)	Removes a specified parameter from the collection.
	RemoveAt (see page 1873)	This is the overview for the RemoveAt method overload.

Strict private Fields

	Name	Description
	FDBXParameters (see page 1867)	This is FDBXParameters, a member of class TAdoDbxParameterCollection.
	FParameters (see page 1867)	This is FParameters, a member of class TAdoDbxParameterCollection.

Strict protected Methods

	Name	Description
	GetParameter (see page 1871)	This is the overview for the GetParameter method overload.

	SetParameter (see page 1874)	This is the overview for the SetParameter method overload.

15.1.1.1.12.2 TAdoDbxParameterCollection.Destroy Destructor

Destroys a TAdoDbxParameterCollection (see page 1865) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdoDbxParameterCollection();
```

Description

This method destroys a TAdoDbxParameterCollection (see page 1865) instance, freeing all its resources.

15.1.1.1.12.3 TAdoDbxParameterCollection Fields

The fields of the TAdoDbxParameterCollection class are listed here.

Strict private Fields

	Name	Description
	FDBXParameters (see page 1867)	This is FDBXParameters, a member of class TAdoDbxParameterCollection.
	FParameters (see page 1867)	This is FParameters, a member of class TAdoDbxParameterCollection.

15.1.1.1.12.3.1 TAdoDbxParameterCollection.FDBXParameters Field

Pascal

```
FDBXParameters: TDBXParameterList;
```

C++

```
TDBXParameterList FDBXParameters;
```

Description

This is FDBXParameters, a member of class TAdoDbxParameterCollection.

15.1.1.1.12.3.2 TAdoDbxParameterCollection.FParameters Field

Pascal

```
FParameters: ArrayList;
```

C++

```
ArrayList FParameters;
```

Description

This is FParameters, a member of class TAdoDbxParameterCollection.

15.1.1.1.12.4 TAdoDbxParameterCollection Methods

The methods of the TAdoDbxParameterCollection class are listed here.

Public Methods

	Name	Description
	Add (see page 1868)	Adds a TAdoDbxParameter (see page 1850) parameter object to the parameter collection.
	AddRange (see page 1868)	Copies an array of parameters to a collection.

	Clear (see page 1869)	Removes all parameters from collection.
	Contains (see page 1869)	This is the overview for the Contains method overload.
	CopyTo (see page 1869)	Copies an array of items to collection.
	get_Count (see page 1870)	This is get_Count, a member of class TAdoDbxParameterCollection.
	get_IsFixedSize (see page 1870)	This is get_IsFixedSize, a member of class TAdoDbxParameterCollection.
	get_IsReadOnly (see page 1870)	This is get_IsReadOnly, a member of class TAdoDbxParameterCollection.
	get_IsSynchronized (see page 1870)	This is get_IsSynchronized, a member of class TAdoDbxParameterCollection.
	get_SyncRoot (see page 1871)	This is get_SyncRoot, a member of class TAdoDbxParameterCollection.
	GetEnumerator (see page 1871)	Returns an enumerator for the collection.
	IndexOf (see page 1872)	This is the overview for the IndexOf method overload.
	Insert (see page 1872)	Inserts a parameter in collection.
	Remove (see page 1873)	Removes a specified parameter from the collection.
	RemoveAt (see page 1873)	This is the overview for the RemoveAt method overload.

Strict protected Methods

	Name	Description
	GetParameter (see page 1871)	This is the overview for the GetParameter method overload.
	SetParameter (see page 1874)	This is the overview for the SetParameter method overload.

15.1.1.1.12.4.1 TAdoDbxParameterCollection.Add Method

Adds a TAdoDbxParameter (see page 1850) parameter object to the parameter collection.

Pascal

```
function Add(Value: TObject): Integer; override;
```

C++

```
virtual __fastcall int Add(TObject * Value);
```

Description

The TAdoDbxParameterCollection (see page 1865) is a collection object that can hold a number of objects of the same type in a hierarchical list. You can derive parameters from your database columns, and add them to the TAdoDbxParameterCollection (see page 1865) in order to access them when building SQL statements using the TAdoDbxCommand (see page 1763) object or the TAdoDbxCommandBuilder (see page 1776) object.

See Also

Add (MSDN)

15.1.1.1.12.4.2 TAdoDbxParameterCollection.AddRange Method

Copies an array of parameters to a collection.

Pascal

```
procedure AddRange(Values: System.Array); override;
```

C++

```
virtual __fastcall AddRange(System.Array Values);
```

Description

AddRange adds an array of TAdoDbxParameter (see page 1850) objects to the parameter collection.

If you call AddRange multiple times in the same session, each call copies only the most recently named parameter collection.

See Also

AddRange (MSDN)

Count

15.1.1.1.12.4.3 TAdoDbxParameterCollection.Clear Method

Removes all parameters from collection.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Clear removes all TAdoDbxParameter (see page 1850) parameter objects from the parameter collection.

See Also

Clear (MSDN)

15.1.1.1.12.4.4 Contains Method

This is the overview for the Contains method overload.

Overload List

	Name	Description
	TAdoDbxParameterCollection.Contains (string) (see page 1869)	Indicates whether parameter with specified property in collection.
	TAdoDbxParameterCollection.Contains (TObject) (see page 1869)	Indicates whether parameter with specified property in collection.

15.1.1.1.12.4.4.1 TAdoDbxParameterCollection.Contains Method (TObject)

Indicates whether parameter with specified property in collection.

Pascal

```
function Contains(Value: TObject): Boolean; override;
function Contains(Value: string): Boolean; override;
```

C++

```
virtual __fastcall Boolean Contains(TObject * Value);
virtual __fastcall Boolean Contains(AnsiString Value);
```

Description

Contains Indicates whether a TAdoDbxParameter (see page 1850) parameter with the specified property is already in the collection.

See Also

Contains (MSDN)

15.1.1.1.12.4.5 TAdoDbxParameterCollection.CopyTo Method

Copies an array of items to collection.

Pascal

```
procedure CopyTo(Values: System.Array; Index: Integer); override;
```

C++

```
virtual __fastcall CopyTo(System.Array Values, int Index);
```

Description

CopyTo copies an array of items to the collection starting at a specified index.

Parameter	Description
Values	Array of items to copy.
Index	Index in the collection at which to start copying.

See Also

CopyTo (MSDN)

15.1.1.1.12.4.6 TAdoDbxParameterCollection.get_Count Method

Pascal

```
function get_Count: Integer; override;
```

C++

```
virtual __fastcall int get_Count();
```

Description

This is get_Count, a member of class TAdoDbxParameterCollection.

15.1.1.1.12.4.7 TAdoDbxParameterCollection.get_IsFixedSize Method

Pascal

```
function get_IsFixedSize: Boolean; override;
```

C++

```
virtual __fastcall Boolean get_IsFixedSize();
```

Description

This is get_IsFixedSize, a member of class TAdoDbxParameterCollection.

15.1.1.1.12.4.8 TAdoDbxParameterCollection.get_IsReadOnly Method

Pascal

```
function get_IsReadOnly: Boolean; override;
```

C++

```
virtual __fastcall Boolean get_IsReadOnly();
```

Description

This is get_IsReadOnly, a member of class TAdoDbxParameterCollection.

15.1.1.1.12.4.9 TAdoDbxParameterCollection.get_IsSynchronized Method

Pascal

```
function get_IsSynchronized: Boolean; override;
```

C++

```
virtual __fastcall Boolean get_IsSynchronized();
```

Description

This is get_IsSynchronized, a member of class TAdoDbxParameterCollection.

15.1.1.1.12.4.10 TAdoDbxParameterCollection.get_SyncRoot Method

Pascal

```
function get_SyncRoot: TObject; override;
```

C++

```
virtual __fastcall TObject * get_SyncRoot();
```

Description

This is get_SyncRoot, a member of class TAdoDbxParameterCollection.

15.1.1.1.12.4.11 TAdoDbxParameterCollection.GetEnumerator Method

Returns an enumerator for the collection.

Pascal

```
function GetEnumerator: IEnumerator; override;
```

C++

```
virtual __fastcall IEnumerator GetEnumerator();
```

Description

GetEnumerator returns an enumerator, which supports simple iteration over a collection. The returned value supports the IEnumerator Interface.

See Also

GetEnumerator (MSDN)

IEnumerator (MSDN)

15.1.1.1.12.4.12 GetParameter Method

This is the overview for the GetParameter method overload.

Overload List

	Name	Description
	TAdoDbxParameterCollection.GetParameter (Integer) (see page 1871)	This is GetParameter, a member of class TAdoDbxParameterCollection.
	TAdoDbxParameterCollection.GetParameter (String) (see page 1872)	This is GetParameter, a member of class TAdoDbxParameterCollection.

15.1.1.1.12.4.12.1 TAdoDbxParameterCollection.GetParameter Method (Integer)

Pascal

```
function GetParameter(Ordinal: Integer): DbParameter; override;
```

C++

```
virtual __fastcall DbParameter GetParameter(int Ordinal);
```

Description

This is GetParameter, a member of class TAdoDbxParameterCollection.

15.1.1.1.12.4.12.2 TAdoDbxParameterCollection.GetParameter Method (String)

Pascal

```
function GetParameter(Name: String): DbParameter; override;
```

C++

```
virtual __fastcall DbParameter GetParameter(AnsiString Name);
```

Description

This is GetParameter, a member of class TAdoDbxParameterCollection.

15.1.1.1.12.4.13 IndexOf Method

This is the overview for the IndexOf method overload.

Overload List

	Name	Description
	TAdoDbxParameterCollection.IndexOf (String) (see page 1872)	Get index of parameter in collection.
	TAdoDbxParameterCollection.IndexOf (TObject) (see page 1872)	Get index of parameter in collection.

15.1.1.1.12.4.13.1 TAdoDbxParameterCollection.IndexOf Method (String)

Get index of parameter in collection.

Pascal

```
function IndexOf(Parameter: TObject): Integer; override;
function IndexOf(Name: String): Integer; override;
```

C++

```
virtual __fastcall int IndexOf(TObject * Parameter);
virtual __fastcall int IndexOf(AnsiString Name);
```

Description

IndexOf returns the index in the collection of a given TAdoDbxParameter (see page 1850) instance.

See Also

IndexOf (MSDN)

15.1.1.1.12.4.14 TAdoDbxParameterCollection.Insert Method

Inserts a parameter in collection.

Pascal

```
procedure Insert(Ordinal: Integer; Parameter: TObject); override;
```

C++

```
virtual __fastcall Insert(int Ordinal, TObject * Parameter);
```

Description

Insert a specifed TAdoDbxParameter (see page 1850) instance into a collection at the specified index.

Parameter	Description
Ordinal	Index in collection at which to insert TAdoDbxParameter (see page 1850) instance.
Parameter	TAdoDbxParameter (see page 1850) instance to insert.

See Also

Insert (MSDN)

15.1.1.1.12.4.15 TAdoDbxParameterCollection.Remove Method

Removes a specified parameter from the collection.

Pascal

```
procedure Remove(Parameter: TObject); override;
```

C++

```
virtual __fastcall Remove(TObject * Parameter);
```

Description

Remove removes the specified TAdoDbxParameter (see page 1850) instance from the collection.

See Also

Remove (MSDN)

15.1.1.1.12.4.16 RemoveAt Method

This is the overview for the RemoveAt method overload.

Overload List

	Name	Description
	TAdoDbxParameterCollection.RemoveAt (Integer) (see page 1873)	Removes a specified parameter from the collection.
	TAdoDbxParameterCollection.RemoveAt (String) (see page 1873)	Removes a specified parameter from the collection.

15.1.1.1.12.4.16.1 TAdoDbxParameterCollection.RemoveAt Method (Integer)

Removes a specified parameter from the collection.

Pascal

```
procedure RemoveAt(Ordinal: Integer); override;
procedure RemoveAt(Name: String); override;
```

C++

```
virtual __fastcall RemoveAt(int Ordinal);
virtual __fastcall RemoveAt(AnsiString Name);
```

Description

RemoveAt removes a specified TAdoDbxParameter (see page 1850) instance from the collection. The parameter may be specified by either a String or an index into the collection.

See Also

RemoveAt (MSDN)

15.1.1.1.12.4.17 SetParameter Method

This is the overview for the SetParameter method overload.

Overload List

	Name	Description
	TAdoDbxParameterCollection.SetParameter (Integer, DbParameter) (see page 1874)	This is SetParameter, a member of class TAdoDbxParameterCollection.
	TAdoDbxParameterCollection.SetParameter (String, DbParameter) (see page 1874)	This is SetParameter, a member of class TAdoDbxParameterCollection.

15.1.1.1.12.4.17.1 TAdoDbxParameterCollection.SetParameter Method (Integer, DbParameter)

Pascal

```
procedure SetParameter(Ordinal: Integer; Parameter: DbParameter); override;
```

C++

```
virtual __fastcall SetParameter(int Ordinal, DbParameter Parameter);
```

Description

This is SetParameter, a member of class TAdoDbxParameterCollection.

15.1.1.1.12.4.17.2 TAdoDbxParameterCollection.SetParameter Method (String, DbParameter)

Pascal

```
procedure SetParameter(Name: String; Parameter: DbParameter); override;
```

C++

```
virtual __fastcall SetParameter(AnsiString Name, DbParameter Parameter);
```

Description

This is SetParameter, a member of class TAdoDbxParameterCollection.

15.1.1.1.13 TAdoDbxProviderFactory Class

Base class for a provider's implementation of data source classes.

Class Hierarchy

Pascal

```
TAdoDbxProviderFactory = class(DbProviderFactory);
```

C++

```
class TAdoDbxProviderFactory : public DbProviderFactory;
```

File

Borland.Data.AdoDbxClientProvider

Description

TAdoDbxProviderFactory furnishes a set of methods to get instances of a provider's implementation of the data source classes, such as TAdoDbxCommand (see page 1763) or TAdoDbxConnection (see page 1779). Use this factory rather than

instantiate these classes directly in your application.

The TAdoDbxProviderFactory class is an implementation of the ADO.NET DbProviderFactory class.

See Also

DbProviderFactory (MSDN)

15.1.1.1.13.1 TAdoDbxProviderFactory Members

The following tables list the members exposed by TAdoDbxProviderFactory.

Public Fields

	Name	Description
	Instance (see page 1876)	This is Instance, a member of class TAdoDbxProviderFactory.

Public Methods

	Name	Description
	AdoDbxException (see page 1877)	Raise an exception for a given error.
V	CreateCommand (see page 1877)	Gets instance of command class for provider.
V	CreateCommandBuilder (see page 1877)	Gets instance of command builder class for provider.
V	CreateConnection (see page 1878)	Gets instance of connection class for provider.
V	CreateConnectionStringBuilder (see page 1878)	Gets instance of connection string builder class for provider.
V	CreateDataAdapter (see page 1878)	Gets instance of data adapter class for provider.
V	CreateDataSourceEnumerator (see page 1879)	Gets instance of data source enumerator class for provider.
V	CreateParameter (see page 1879)	Gets instance of parameter class for provider.
	DbxToAdoType (see page 1879)	This is DbxToAdoType, a member of class TAdoDbxProviderFactory.
V	Destroy (see page 1876)	Destroys a TAdoDbxProviderFactory (see page 1874) instance.
	ReadCanCreateDataSourceEnumerator (see page 1879)	Indicates whether data source enumerator can be created.

Public Properties

	Name	Description
R	CanCreateDataSourceEnumerator (see page 1880)	Indicates whether data source enumerator can be created.

Strict private Methods

	Name	Description
	Create (see page 1875)	This is the overview for the Create constructor overload.

15.1.1.1.13.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TAdoDbxProviderFactory.Create () (see page 1875)	This is Create, a member of class TAdoDbxProviderFactory.
	TAdoDbxProviderFactory.Create (TDBXContext) (see page 1876)	This is Create, a member of class TAdoDbxProviderFactory.

15.1.1.1.13.2.1 TAdoDbxProviderFactory.Create Constructor ()

Pascal

```
class constructor Create;
```

C++

```
__fastcall TAdoDbxProviderFactory();
```

Description

This is Create, a member of class TAdoDbxProviderFactory.

15.1.1.1.13.2.2 TAdoDbxProviderFactory.Create Constructor (TDBXContext)

Pascal

```
constructor Create(DbxContext: TDBXContext);
```

C++

```
__fastcall TAdoDbxProviderFactory(TDBXContext DbxContext);
```

Description

This is Create, a member of class TAdoDbxProviderFactory.

15.1.1.1.13.3 TAdoDbxProviderFactory.Destroy Destructor

Destroys a TAdoDbxProviderFactory (see page 1874) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdoDbxProviderFactory();
```

Description

This method destroys a TAdoDbxProviderFactory (see page 1874) instance, freeing all its resources.

15.1.1.1.13.4 TAdoDbxProviderFactory Fields

The fields of the TAdoDbxProviderFactory class are listed here.

Public Fields

	Name	Description
	Instance (see page 1876)	This is Instance, a member of class TAdoDbxProviderFactory.

15.1.1.1.13.4.1 TAdoDbxProviderFactory.Instance Field

Pascal

```
class var Instance: TAdoDbxProviderFactory;
```

C++

```
TAdoDbxProviderFactory Instance;
```

Description

This is Instance, a member of class TAdoDbxProviderFactory.

15.1.1.1.13.5 TAdoDbxProviderFactory Methods

The methods of the TAdoDbxProviderFactory class are listed here.

Public Methods

	Name	Description
	AdoDbxException (see page 1877)	Raise an exception for a given error.
V	CreateCommand (see page 1877)	Gets instance of command class for provider.
V	CreateCommandBuilder (see page 1877)	Gets instance of command builder class for provider.
V	CreateConnection (see page 1878)	Gets instance of connection class for provider.
V	CreateConnectionStringBuilder (see page 1878)	Gets instance of connection string builder class for provider.
V	CreateDataAdapter (see page 1878)	Gets instance of data adapter class for provider.
V	CreateDataSourceEnumerator (see page 1879)	Gets instance of data source enumerator class for provider.
V	CreateParameter (see page 1879)	Gets instance of parameter class for provider.
	DbxToAdoType (see page 1879)	This is DbxToAdoType, a member of class TAdoDbxProviderFactory.
	ReadCanCreateDataSourceEnumerator (see page 1879)	Indicates whether data source enumerator can be created.

15.1.1.1.13.5.1 TAdoDbxProviderFactory.AdoDbxException Method

Raise an exception for a given error.

Pascal

```
procedure AdoDbxException(DBXError: TDBXError);
```

C++

```
__fastcall AdoDbxException(TDBXError DBXError);
```

Description

AdoDbxException raises a TAdoDbxException (see page 1845) for the given TDBXError instance.

See Also

TAdoDbxException (see page 1845)

TDBXError (see page 10486)

15.1.1.1.13.5.2 TAdoDbxProviderFactory.CreateCommand Method

Gets instance of command class for provider.

Pascal

```
function CreateCommand: DbCommand; override;
```

C++

```
virtual __fastcall DbCommand CreateCommand();
```

Description

CreateCommand returns an instance of the provider's class that implements TAdoDbxCommand (see page 1763).

See Also

CreateCommand (MSDN)

15.1.1.1.13.5.3 TAdoDbxProviderFactory.CreateCommandBuilder Method

Gets instance of command builder class for provider.

Pascal

```
function CreateCommandBuilder: DbCommandBuilder; override;
```

C++

```
virtual __fastcall DbCommandBuilder CreateCommandBuilder();
```

Description

CreateCommandBuilder returns an instance of the provider's class that implements TAdoDbxCommandBuilder (see page 1776).

See Also

CreateCommandBuilder (MSDN)

15.1.1.1.13.5.4 TAdoDbxProviderFactory.CreateConnection Method

Gets instance of connection class for provider.

Pascal

```
function CreateConnection: DbConnection; override;
```

C++

```
virtual __fastcall DbConnection CreateConnection();
```

Description

CreateConnection returns an instance of the provider's class that implements TAdoDbxConnection (see page 1779).

See Also

CreateConnection (MSDN)

15.1.1.1.13.5.5 TAdoDbxProviderFactory.CreateConnectionStringBuilder Method

Gets instance of connection string builder class for provider.

Pascal

```
function CreateConnectionStringBuilder: DbConnectionStringBuilder; override;
```

C++

```
virtual __fastcall DbConnectionStringBuilder CreateConnectionStringBuilder();
```

Description

CreateConnectionStringBuilder returns an instance of the provider's class that implements TAdoDbxConnectionStringBuilder (see page 1790) .

See Also

CreateConnectionStringBuilder (MSDN)

15.1.1.1.13.5.6 TAdoDbxProviderFactory.CreateDataAdapter Method

Gets instance of data adapter class for provider.

Pascal

```
function CreateDataAdapter: DbDataAdapter; override;
```

C++

```
virtual __fastcall DbDataAdapter CreateDataAdapter();
```

Description

CreateDataAdapter returns an instance of the provider's class that implements TAdoDbxDataAdapter (see page 1814).

See Also

CreateDataAdapter (MSDN)

15.1.1.1.13.5.7 TAdoDbxProviderFactory.CreateDataSourceEnumerator Method

Gets instance of data source enumerator class for provider.

Pascal

```
function CreateDataSourceEnumerator: DbDataSourceEnumerator; override;
```

C++

```
virtual __fastcall DbDataSourceEnumerator CreateDataSourceEnumerator();
```

Description

CreateDataSourceEnumerator returns an instance of the provider's class that implements TAdoDbxDataSourceEnumerator (see page 1845).

See Also

CreateDataSourceEnumerator (MSDN)

15.1.1.1.13.5.8 TAdoDbxProviderFactory.CreateParameter Method

Gets instance of parameter class for provider.

Pascal

```
function CreateParameter: DbParameter; override;
```

C++

```
virtual __fastcall DbParameter CreateParameter();
```

Description

CreateParameter returns an instance of the provider's class that implements TAdoDbxParameter (see page 1850).

See Also

CreateParameter (MSDN)

15.1.1.1.13.5.9 TAdoDbxProviderFactory.DbxToAdoType Method

Pascal

```
class function DbxToAdoType(DbxType: TDBXType): DbType; static;
```

C++

```
static __fastcall DbType DbxToAdoType(TDBXType DbxType);
```

Description

This is DbxToAdoType, a member of class TAdoDbxProviderFactory.

15.1.1.1.13.5.10 TAdoDbxProviderFactory.ReadCanCreateDataSourceEnumerator Method

Indicates whether data source enumerator can be created.

Pascal

```
function ReadCanCreateDataSourceEnumerator: Boolean;
```

C++

```
__fastcall Boolean ReadCanCreateDataSourceEnumerator();
```

Description

ReadCanCreateDataSourceEnumerator returns **True** if a TAdoDbxDataSourceEnumerator (see page 1845) can be created

and **False** otherwise. It reads the CanCreateDataSourceEnumerator (see page 1880) property.

See Also

TAdoDbxDataSourceEnumerator (see page 1845)

CanCreateDataSourceEnumerator (see page 1880)

15.1.1.1.13.6 TAdoDbxProviderFactory Properties

The properties of the TAdoDbxProviderFactory class are listed here.

Public Properties

	Name	Description
R	CanCreateDataSourceEnumerator (see page 1880)	Indicates whether data source enumerator can be created.

15.1.1.1.13.6.1 TAdoDbxProviderFactory.CanCreateDataSourceEnumerator Property

Indicates whether data source enumerator can be created.

Pascal

```
property CanCreateDataSourceEnumerator: Boolean;
```

C++

```
__property Boolean CanCreateDataSourceEnumerator;
```

Description

CanCreateDataSourceEnumerator is **True** if a TAdoDbxDataSourceEnumerator (see page 1845) can be created and **False** otherwise. The ReadCanCreateDataSourceEnumerator (see page 1879) method returns this property.

See Also

TAdoDbxDataSourceEnumerator (see page 1845)

ReadCanCreateDataSourceEnumerator (see page 1879)

15.1.1.1.14 TAdoDbxRowUpdatedEventArgs Class

Class Hierarchy

Pascal

```
TAdoDbxRowUpdatedEventArgs = class(RowUpdatedEventArgs);
```

C++

```
class TAdoDbxRowUpdatedEventArgs : public RowUpdatedEventArgs;
```

File

Borland.Data.AdoDbxClientProvider

Description

This is class Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatedEventArgs.

15.1.1.1.14.1 TAdoDbxRowUpdatedEventArgs Members

The following tables list the members exposed by TAdoDbxRowUpdatedEventArgs.

Public Methods

	Name	Description
	get_Command (see page 1881)	This is get_Command, a member of class TAdoDbxRowUpdatedEventArgs.

Public Properties

	Name	Description
R	Command (see page 1881)	This is Command, a member of class TAdoDbxRowUpdatedEventArgs.

15.1.1.1.14.2 TAdoDbxRowUpdatedEventArgs Methods

The methods of the TAdoDbxRowUpdatedEventArgs class are listed here.

Public Methods

	Name	Description
	get_Command (see page 1881)	This is get_Command, a member of class TAdoDbxRowUpdatedEventArgs.

15.1.1.1.14.2.1 TAdoDbxRowUpdatedEventArgs.get_Command Method

Pascal

```
function get_Command: TAdoDbxCommand;
```

C++

```
__fastcall TAdoDbxCommand get_Command();
```

Description

This is get_Command, a member of class TAdoDbxRowUpdatedEventArgs.

15.1.1.1.14.3 TAdoDbxRowUpdatedEventArgs Properties

The properties of the TAdoDbxRowUpdatedEventArgs class are listed here.

Public Properties

	Name	Description
R	Command (see page 1881)	This is Command, a member of class TAdoDbxRowUpdatedEventArgs.

15.1.1.1.14.3.1 TAdoDbxRowUpdatedEventArgs.Command Property

Pascal

```
property Command: TAdoDbxCommand;
```

C++

```
__property TAdoDbxCommand Command;
```

Description

This is Command, a member of class TAdoDbxRowUpdatedEventArgs.

15.1.1.1.15 TAdoDbxRowUpdatingEventArgs Class

Class Hierarchy

Pascal

```
TAdoDbxRowUpdatingEventArgs = class(RowUpdatingEventArgs);
```

C++

```
class TAdoDbxRowUpdatingEventArgs : public RowUpdatingEventArgs;
```

File

Borland.Data.AdoDbxClientProvider

Description

This is class Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatingEventArgs.

15.1.1.1.15.1 TAdoDbxRowUpdatingEventArgs Members

The following tables list the members exposed by TAdoDbxRowUpdatingEventArgs.

Public Methods

	Name	Description
	get_Command (see page 1882)	This is get_Command, a member of class TAdoDbxRowUpdatingEventArgs.
	set_Command (see page 1882)	This is set_Command, a member of class TAdoDbxRowUpdatingEventArgs.

Public Properties

	Name	Description
	Command (see page 1883)	This is Command, a member of class TAdoDbxRowUpdatingEventArgs.

15.1.1.1.15.2 TAdoDbxRowUpdatingEventArgs Methods

The methods of the TAdoDbxRowUpdatingEventArgs class are listed here.

Public Methods

	Name	Description
	get_Command (see page 1882)	This is get_Command, a member of class TAdoDbxRowUpdatingEventArgs.
	set_Command (see page 1882)	This is set_Command, a member of class TAdoDbxRowUpdatingEventArgs.

15.1.1.1.15.2.1 TAdoDbxRowUpdatingEventArgs.get_Command Method

Pascal

```
function get_Command: TAdoDbxCommand;
```

C++

```
__fastcall TAdoDbxCommand get_Command();
```

Description

This is get_Command, a member of class TAdoDbxRowUpdatingEventArgs.

15.1.1.1.15.2.2 TAdoDbxRowUpdatingEventArgs.set_Command Method

Pascal

```
procedure set_Command(const Value: TAdoDbxCommand);
```

C++

```
__fastcall set_Command(const TAdoDbxCommand Value);
```

Description

This is set_Command, a member of class TAdoDbxRowUpdatingEventArgs.

15.1.1.1.15.3 TAdoDbxRowUpdatingEventArgs Properties

The properties of the TAdoDbxRowUpdatingEventArgs class are listed here.

Public Properties

	Name	Description
	Command (see page 1883)	This is Command, a member of class TAdoDbxRowUpdatingEventArgs.

15.1.1.1.15.3.1 TAdoDbxRowUpdatingEventArgs.Command Property

Pascal

```
property Command: TAdoDbxCommand;
```

C++

```
__property TAdoDbxCommand Command;
```

Description

This is Command, a member of class TAdoDbxRowUpdatingEventArgs.

15.1.1.1.16 TAdoDbxSchemaTableColumnNames Class

Class Hierarchy

Borland.Data.AdoDbxClientProvider.TAdoDbxSchemaTableColumnNames

Pascal

```
TAdoDbxSchemaTableColumnNames = class;
```

C++

```
class TAdoDbxSchemaTableColumnNames;
```

File

Borland.Data.AdoDbxClientProvider

Description

This is class Borland.Data.AdoDbxClientProvider.TAdoDbxSchemaTableColumnNames.

15.1.1.1.16.1 TAdoDbxSchemaTableColumnNames Members

The following tables list the members exposed by TAdoDbxSchemaTableColumnNames.

Public Constants

	Name	Description
	DbxDataType (see page 1884)	This is DbxDataType, a member of class TAdoDbxSchemaTableColumnNames.
	DbxSubType (see page 1884)	This is DbxSubType, a member of class TAdoDbxSchemaTableColumnNames.

15.1.1.1.16.2 TAdoDbxSchemaTableColumnNames Constants

The constants of the TAdoDbxSchemaTableColumnNames class are listed here.

Public Constants

	Name	Description
	DbxDataType (see page 1884)	This is DbxDataType, a member of class TAdoDbxSchemaTableColumnNames.
	DbxSubType (see page 1884)	This is DbxSubType, a member of class TAdoDbxSchemaTableColumnNames.

15.1.1.1.16.2.1 TAdoDbxSchemaTableColumnNames.DbxDataType Constant

Pascal

```
const DbxDataType = 'DbxDataType';
```

C++

```
DbxDataType = 'DbxDataType';
```

Description

This is DbxDataType, a member of class TAdoDbxSchemaTableColumnNames.

15.1.1.1.16.2.2 TAdoDbxSchemaTableColumnNames.DbxSubType Constant

Pascal

```
const DbxSubType = 'DbxSubType';
```

C++

```
DbxSubType = 'DbxSubType';
```

Description

This is DbxSubType, a member of class TAdoDbxSchemaTableColumnNames.

15.1.1.1.17 TAdoDbxTransaction Class

Represents an instance of the TAdoDbxTransaction object.

Class Hierarchy

Pascal

```
TAdoDbxTransaction = class(DbTransaction);
```

C++

```
class TAdoDbxTransaction : public DbTransaction;
```

File

Borland.Data.AdoDbxClientProvider

Description

Use the BeginTransaction method in TAdoDbxConnection (see page 1779) to create a new instance of the TAdoDbxTransaction object. You use this method within the context of a TAdoDbxConnection (see page 1779) object. Each transaction is directly related to a specific connection. The connection must be open before calling BeginTransaction.

TAdoDbxTransaction implements the ADO.NET class DbTransaction.

TAdoDbxTransaction takes care of controlling a database transaction with respect to a connection. Calling the TAdoDbxConnection.BeginTransaction method on an opened connection returns a new TAdoDbxTransaction object. The Commit (see page 1887) and Rollback (see page 1888) methods take care of committing or rolling back a transaction. For setting different isolation levels, the IsolationLevel property can be used, and the default isolation is `ReadCommitted` . The current implementation supports nested multiple transactions on a single connection, but not overlapped multiple transactions on a single connection.

The following c# code shows runtime parameter binding and execution of a stored procedure in the context of a transaction.

```
static void BindParameter ( TAdoDbxConnection Conn, Int32 Times )
```

```
{
    Int32 Count = 0, Rows = 0;
    TAdoDbxTransaction Trans = (TAdoDbxTransaction) Conn.BeginTransaction();

    TAdoDbxCommand Comm = (TAdoDbxCommand) Conn.CreateCommand();
    Comm.Connection   = Conn;
    Comm.Transaction  = Trans;

    Comm.CommandType = CommandType.StoredProcedure;
    Comm.CommandText  = "MYTESTPROC";

    Comm.Prepare();

    TAdoDbxParameter param1 = Comm.Parameters.Add("P1",
      DbType.StringFixedLength, 10);
    TAdoDbxParameter param2 = Comm.Parameters.Add("P2",
      DbType.String, 5);
    param2.Direction = ParameterDirection.InputOutput;
    param2.Precision = 25;
    TAdoDbxParameter param3 = Comm.Parameters.Add("P3",
      DbType.Decimal);
    param3.Direction = ParameterDirection.Output;


    while ( Count < Times )
   {
       param1.Value = "Record" + Count;
       param2.Value = "Hello";
       param3.Value = null;

       Rows = Comm.ExecuteNonQuery();
       Console.WriteLine("Output param from  MYTESTPROC=  " + param2.Value );
       Console.WriteLine("InputOutput param from  MYTESTPROC=  " + param3.Value );
       Count ++;

    }

    Comm.Close();

    Trans.Commit();
}
```

See Also

DbTransaction (MSDN)

Modifying Database Connections

TAdoDbxConnection (see page 1779)

IsolationLevel

BeginTransaction

Commit (see page 1887)

Rollback (see page 1888)

15.1.1.1.17.1 TAdoDbxTransaction Members

The following tables list the members exposed by TAdoDbxTransaction.

Public Methods

	Name	Description
	Commit (see page 1887)	Commits a transaction.

	Destroy (see page 1886)	Destroys a TAdoDbxTransaction (see page 1884) instance.
	get_IsolationLevel (see page 1888)	This is get_IsolationLevel, a member of class TAdoDbxTransaction.
	Rollback (see page 1888)	Discards the actions performed in a pending transaction.

Strict private Fields

	Name	Description
	FConnection (see page 1886)	This is FConnection, a member of class TAdoDbxTransaction.
	FIsolation (see page 1886)	This is FIsolation, a member of class TAdoDbxTransaction.
	FTransaction (see page 1887)	This is FTransaction, a member of class TAdoDbxTransaction.

Strict protected Methods

	Name	Description
	get_DbConnection (see page 1887)	This is get_DbConnection, a member of class TAdoDbxTransaction.

15.1.1.1.17.2 TAdoDbxTransaction.Destroy Destructor

Destroys a TAdoDbxTransaction (see page 1884) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdoDbxTransaction();
```

Description

This method destroys a TAdoDbxTransaction (see page 1884) instance, freeing all its resources.

15.1.1.1.17.3 TAdoDbxTransaction Fields

The fields of the TAdoDbxTransaction class are listed here.

Strict private Fields

	Name	Description
	FConnection (see page 1886)	This is FConnection, a member of class TAdoDbxTransaction.
	FIsolation (see page 1886)	This is FIsolation, a member of class TAdoDbxTransaction.
	FTransaction (see page 1887)	This is FTransaction, a member of class TAdoDbxTransaction.

15.1.1.1.17.3.1 TAdoDbxTransaction.FConnection Field

Pascal

```
FConnection: TAdoDbxConnection;
```

C++

```
TAdoDbxConnection FConnection;
```

Description

This is FConnection, a member of class TAdoDbxTransaction.

15.1.1.1.17.3.2 TAdoDbxTransaction.FIsolation Field

Pascal

```
FIsolation: IsolationLevel;
```

C++

```
IsolationLevel FIsolation;
```

Description

This is FIsolation, a member of class TAdoDbxTransaction.

15.1.1.1.17.3.3 TAdoDbxTransaction.FTransaction Field

Pascal

```
FTransaction: TDBXTransaction;
```

C++

```
TDBXTransaction FTransaction;
```

Description

This is FTransaction, a member of class TAdoDbxTransaction.

15.1.1.1.17.4 TAdoDbxTransaction Methods

The methods of the TAdoDbxTransaction class are listed here.

Public Methods

	Name	Description
	Commit (see page 1887)	Commits a transaction.
	get_IsolationLevel (see page 1888)	This is get_IsolationLevel, a member of class TAdoDbxTransaction.
	Rollback (see page 1888)	Discards the actions performed in a pending transaction.

Strict protected Methods

	Name	Description
	get_DbConnection (see page 1887)	This is get_DbConnection, a member of class TAdoDbxTransaction.

15.1.1.1.17.4.1 TAdoDbxTransaction.Commit Method

Commits a transaction.

Pascal

```
procedure Commit; override;
```

C++

```
virtual __fastcall Commit();
```

Description

If the commit fails, this method throws an exception, typically indicating that the transaction has been committed already, does not exist, or that the connection is broken.

See Also

Commit (MSDN)

15.1.1.1.17.4.2 TAdoDbxTransaction.get_DbConnection Method

Pascal

```
function get_DbConnection: DbConnection; override;
```

C++

```
virtual __fastcall DbConnection get_DbConnection();
```

Description

This is get_DbConnection, a member of class TAdoDbxTransaction.

15.1.1.1.17.4.3 TAdoDbxTransaction.get_IsolationLevel Method

Pascal

```
function get_IsolationLevel: IsolationLevel; override;
```

C++

```
virtual __fastcall IsolationLevel get_IsolationLevel();
```

Description

This is get_IsolationLevel, a member of class TAdoDbxTransaction.

15.1.1.1.17.4.4 TAdoDbxTransaction.Rollback Method

Discards the actions performed in a pending transaction.

Pascal

```
procedure Rollback; override;
```

C++

```
virtual __fastcall Rollback();
```

Description

The transaction can only be rolled back from a pending state, after BeginTransaction has been called, but before Commit (see page 1887) is called.

See Also

Rollback (MSDN)

TAdoDbxConnection (see page 1779)

BeginTransaction

Commit (see page 1887)

15.1.1.1.18 TConnectionStringCategories Class

Categories used for connection string builder UI.

Class Hierarchy

Pascal

```
TConnectionStringCategories = class;
```

C++

```
class TConnectionStringCategories;
```

File

Borland.Data.AdoDbxClientProvider

Description

TConnectionStringCategories lists categories used for the connection string builder user interface.

This table contains the categories.

Constant	Description
Dynalink (see page 1889)	Dynalink (see page 1889) related properties.
DriverConfiguration (see page 1889)	Driver configuration related properties.
DataBase (see page 1889)	Database related properties.

15.1.1.1.18.1 TConnectionStringCategories Members

The following tables list the members exposed by TConnectionStringCategories.

Public Constants

	Name	Description
	DataBase (see page 1889)	This is DataBase, a member of class TConnectionStringCategories.
	DriverConfiguration (see page 1889)	This is DriverConfiguration, a member of class TConnectionStringCategories.
	Dynalink (see page 1889)	This is Dynalink, a member of class TConnectionStringCategories.

15.1.1.1.18.2 TConnectionStringCategories Constants

The constants of the TConnectionStringCategories class are listed here.

Public Constants

	Name	Description
	DataBase (see page 1889)	This is DataBase, a member of class TConnectionStringCategories.
	DriverConfiguration (see page 1889)	This is DriverConfiguration, a member of class TConnectionStringCategories.
	Dynalink (see page 1889)	This is Dynalink, a member of class TConnectionStringCategories.

15.1.1.1.18.2.1 TConnectionStringCategories.DataBase Constant

Pascal

```
const DataBase = 'Database';
```

C++

```
DataBase = 'Database';
```

Description

This is DataBase, a member of class TConnectionStringCategories.

15.1.1.1.18.2.2 TConnectionStringCategories.DriverConfiguration Constant

Pascal

```
const DriverConfiguration = 'Driver Configuration';
```

C++

```
DriverConfiguration = 'Driver Configuration';
```

Description

This is DriverConfiguration, a member of class TConnectionStringCategories.

15.1.1.1.18.2.3 TConnectionStringCategories.Dynalink Constant

Pascal

```
const Dynalink = 'Dynalink';
```

C++

```
Dynalink = 'Dynalink';
```

Description

This is Dynalink, a member of class TConnectionStringCategories.

15.1.1.2 Types

The following table lists types in this documentation.

Types

Name	Description
TAdoDbxRowUpdatedEventHandler (see page 1890)	Event handler for TAdoDbxDataAdapter.RowUpdated event.
TAdoDbxRowUpdatingEventHandler (see page 1890)	Event handler for TAdoDbxDataAdapter.RowUpdating event.
TObjectArray (see page 1891)	This is type Borland.Data.AdoDbxClientProvider.TObjectArray.

15.1.1.2.1 Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatedEventHandler Type

Event handler for TAdoDbxDataAdapter.RowUpdated event.

Pascal

```
TAdoDbxRowUpdatedEventHandler = procedure (Sender: System.Object; e:
TAdoDbxRowUpdatedEventArgs) of object;
```

C++

```
(Sender: System.Object; e: TAdoDbxRowUpdatedEventArgs) ( TAdoDbxRowUpdatedEventHandler)();
```

File

Borland.Data.AdoDbxClientProvider

Description

TAdoDbxRowUpdatedEventHandler is an event handler that can be set on the TAdoDbxDataAdapter.RowUpdated event.

See Also

RowUpdated (see page 1826)

RowUpdating (see page 1826)

TAdoDbxRowUpdatingEventHandler (see page 1890)

15.1.1.2.2 Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatingEventHandler Type

Event handler for TAdoDbxDataAdapter.RowUpdating event.

Pascal

```
TAdoDbxRowUpdatingEventHandler = procedure (Sender: System.Object; e:
TAdoDbxRowUpdatingEventArgs) of object;
```

C++

```
(Sender: System.Object; e: TAdoDbxRowUpdatingEventArgs) ( TAdoDbxRowUpdatingEventHandler)();
```

File

Borland.Data.AdoDbxClientProvider

Description

TAdoDbxRowUpdatingEventHandler is an event handler that can be set on the TAdoDbxDataAdapter.RowUpdating event.

See Also

RowUpdated (see page 1826)

RowUpdating (see page 1826)

TAdoDbxRowUpdatedEventHandler (see page 1890)

15.1.1.2.3 Borland.Data.AdoDbxClientProvider.TObjectArray Type

Pascal

```
TObjectArray = array of System.Object;
```

C++

```
array of System.Object TObjectArray;
```

File

Borland.Data.AdoDbxClientProvider

Description

This is type Borland.Data.AdoDbxClientProvider.TObjectArray.

15.1.1.3 Constants

The following table lists constants in this documentation.

Constants

Name	Description
SCommandAlreadyPrepared (see page 1891)	This is constant Borland.Data.AdoDbxClientProvider.SCommandAlreadyPrepared.
SInvalidFieldType (see page 1892)	This is constant Borland.Data.AdoDbxClientProvider.SInvalidFieldType.
SUnSupported (see page 1892)	This is constant Borland.Data.AdoDbxClientProvider.SUnSupported.

15.1.1.3.1 Borland.Data.AdoDbxClientProvider.SCommandAlreadyPrepared Constant

Pascal

```
SCommandAlreadyPrepared = 'Command already prepared';
```

C++

```
SCommandAlreadyPrepared = 'Command already prepared';
```

File

Borland.Data.AdoDbxClientProvider

Description

This is constant Borland.Data.AdoDbxClientProvider.SCommandAlreadyPrepared.

15.1.1.3.2 Borland.Data.AdoDbxClientProvider.SInvalidFieldType Constant

Pascal

```
SInvalidFieldType = 'Invalid field type:  %s';
```

C++

```
SInvalidFieldType = 'Invalid field type:  %s';
```

File

Borland.Data.AdoDbxClientProvider

Description

This is constant Borland.Data.AdoDbxClientProvider.SInvalidFieldType.

15.1.1.3.3 Borland.Data.AdoDbxClientProvider.SUnSupported Constant

Pascal

```
SUnSupported = 'Unsupported Operation';
```

C++

```
SUnSupported = 'Unsupported Operation';
```

File

Borland.Data.AdoDbxClientProvider

Description

This is constant Borland.Data.AdoDbxClientProvider.SUnSupported.

15.1.2 Borland.Data.DataStore Namespace

This is namespace Borland.Data.DataStore.

Topics

Name	Description
Classes (see page 1893)	The following table lists classes in this documentation.
Types (see page 2129)	The following table lists types in this documentation.

Classes

	Name	Description
	ConColumns (see page 1894)	Table returned by the builtin stored procedure DB_ADMIN.GET_CONNECTIONS(). Provides a table of information about open connections for a BlackfishSQL server.
	ConnectionProperties (see page 1897)	Names of connection properties supported by Blackfish SQL driver.
	DatabaseColumns (see page 1906)	Column names for result table returned from DB_ADMIN.GET_DATABASE_PROPS.

	DatabaseStatusTable (see page 1910)	Provides the column names for the result table returned from the builtin stored procedure DB_ADMIN.GET_DATABASE_STATUS(). The DB_ADMIN.GET_DATABASE_STATUS() stored procedure provides a table of information about the status of the connected database.
	DataStoreCommand (see page 1912)	Provides execution of SQL statements and execution of stored procedures.
	DataStoreConnection (see page 1923)	Provides a connection to a Blackfish SQL database.
	DataStoreDataAdapter (see page 1933)	Fills a DataSet (see page 1937) and updates a Blackfish SQL database.
	DataStoreErrorCode (see page 1939)	Symbolic error codes for Blackfish SQL database.
	DataStoreParameter (see page 2033)	Specifies parameter for DataStoreCommand (see page 1912).
	DataStoreParameterCollection (see page 2041)	Collection of parameters for DataStoreCommand (see page 1912).
	DataStoreRowUpdatedEventArgs (see page 2053)	Holds data for RowUpdated event of DataStoreDataAdapter (see page 1933).
	DataStoreRowUpdatingEventArgs (see page 2054)	Holds data for RowUpdating event of DataStoreDataAdapter (see page 1933).
	DataStoreTransaction (see page 2056)	This is class Borland.Data.DataStore.DataStoreTransaction.
	LicenseColumns (see page 2059)	Column names for result table returned from DB_ADMIN.GET_LICENSE and DB_ADMIN.GET_ALL_LICENSES.
	LockColumns (see page 2062)	Provides column names for result table from built-in stored procedure DB_ADMIN.GET_LOCKS.
	LogFilter (see page 2065)	Provides names of events to be logged.
	MirrorStatusColumns (see page 2069)	Provides columns of result table from built-in stored procedure DB_ADMIN.GET_MIRRORS.
	MirrorTypes (see page 2072)	Contains the valid values for the TYPE column from DB_ADMIN.GET_MIRRORS stored procedure.
	ProtocolValue (see page 2074)	Contains constant strings for the protocol connection property.
	ScheduleColumns (see page 2076)	Extends SysMirrors (see page 2089) with additional status information.
	SchedulePeriod (see page 2078)	Possible values for SysSchedule.PERIOD column.
	SqlStateCode (see page 2079)	Provides the possible values of the SQLState property of a SQLException raised by Blackfish SQL.
	StatusLogColumns (see page 2087)	Provides column names for result table from built-in stored procedure DB_ADMIN.GET_STATUS_LOGS.
	SysMirrors (see page 2089)	Provides the column names of the system table **SYS.MIRRORS**.
	SysSchedule (see page 2095)	Provides column names from system table **SYS.MIRROR_SCHEDULE**.
	SystemProperties (see page 2097)	Specifies names of system properties that should be applied to all databases open in a process.
	TriggerContext (see page 2101)	An instance of the TriggerContext class is passed to each trigger.
	TriggerReadRow (see page 2103)	Provides read access to a row of data.
	TriggerReadWriteRow (see page 2117)	Provides write access to a row of data.

Types

Name	Description
DataStoreRowUpdatedEventHandler (see page 2129)	This is type Borland.Data.DataStore.DataStoreRowUpdatedEventHandler.
DataStoreRowUpdatingEventHandler (see page 2129)	This is type Borland.Data.DataStore.DataStoreRowUpdatingEventHandler.

15.1.2.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ConColumns (see page 1894)	Table returned by the builtin stored procedure DB_ADMIN.GET_CONNECTIONS(). Provides a table of information about open connections for a BlackfishSQL server.
	ConnectionProperties (see page 1897)	Names of connection properties supported by Blackfish SQL driver.
	DatabaseColumns (see page 1906)	Column names for result table returned from DB_ADMIN.GET_DATABASE_PROPS.
	DatabaseStatusTable (see page 1910)	Provides the column names for the result table returned from the builtin stored procedure DB_ADMIN.GET_DATABASE_STATUS(). The DB_ADMIN.GET_DATABASE_STATUS() stored procedure provides a table of information about the status of the connected database.

	DataStoreCommand (see page 1912)	Provides execution of SQL statements and execution of stored procedures.
	DataStoreConnection (see page 1923)	Provides a connection to a Blackfish SQL database.
	DataStoreDataAdapter (see page 1933)	Fills a DataSet (see page 1937) and updates a Blackfish SQL database.
	DataStoreErrorCode (see page 1939)	Symbolic error codes for Blackfish SQL database.
	DataStoreParameter (see page 2033)	Specifies parameter for DataStoreCommand (see page 1912).
	DataStoreParameterCollection (see page 2041)	Collection of parameters for DataStoreCommand (see page 1912).
	DataStoreRowUpdatedEventArgs (see page 2053)	Holds data for RowUpdated event of DataStoreDataAdapter (see page 1933).
	DataStoreRowUpdatingEventArgs (see page 2054)	Holds data for RowUpdating event of DataStoreDataAdapter (see page 1933).
	DataStoreTransaction (see page 2056)	This is class Borland.Data.DataStore.DataStoreTransaction.
	LicenseColumns (see page 2059)	Column names for result table returned from DB_ADMIN.GET_LICENSE and DB_ADMIN.GET_ALL_LICENSES.
	LockColumns (see page 2062)	Provides column names for result table from built-in stored procedure DB_ADMIN.GET_LOCKS.
	LogFilter (see page 2065)	Provides names of events to be logged.
	MirrorStatusColumns (see page 2069)	Provides columns of result table from built-in stored procedure DB_ADMIN.GET_MIRRORS.
	MirrorTypes (see page 2072)	Contains the valid values for the TYPE column from DB_ADMIN.GET_MIRRORS stored procedure.
	ProtocolValue (see page 2074)	Contains constant strings for the protocol connection property.
	ScheduleColumns (see page 2076)	Extends SysMirrors (see page 2089) with additional status information.
	SchedulePeriod (see page 2078)	Possible values for SysSchedule.PERIOD column.
	SqlStateCode (see page 2079)	Provides the possible values of the SQLState property of a SQLException raised by Blackfish SQL.
	StatusLogColumns (see page 2087)	Provides column names for result table from built-in stored procedure DB_ADMIN.GET_STATUS_LOGS.
	SysMirrors (see page 2089)	Provides the column names of the system table **SYS.MIRRORS**.
	SysSchedule (see page 2095)	Provides column names from system table **SYS.MIRROR_SCHEDULE**.
	SystemProperties (see page 2097)	Specifies names of system properties that should be applied to all databases open in a process.
	TriggerContext (see page 2101)	An instance of the TriggerContext class is passed to each trigger.
	TriggerReadRow (see page 2103)	Provides read access to a row of data.
	TriggerReadWriteRow (see page 2117)	Provides write access to a row of data.

15.1.2.1.1 ConColumns Class

Table returned by the builtin stored procedure DB_ADMIN.GET_CONNECTIONS(). Provides a table of information about open connections for a BlackfishSQL server.

Class Hierarchy

Borland.Data.DataStore.ConColumns

Pascal

```
ConColumns = class;
```

C++

```
class ConColumns;
```

File

ConColumns.cs

Description

The table returned by the builtin stored procedure DB_ADMIN.GET_CONNECTIONS(). Provides a table of information about open connections for a BlackfishSQL server.

The columns of the result table are:

Constant	Description
Id (see page 1896)	Unique id for the connection. Note that this id is reused if the connection is closed.
User (see page 1897)	Name of the user that created this connection.
Ip (see page 1896)	IP address for this connection.
Host (see page 1896)	Host (see page 1896) name for this connection
FileName (see page 1896)	File name of database connected to.
Remarks (see page 1897)	Optional remarks that explain the reason for this connection.
Birth (see page 1896)	Timestamp of the point in time the connection was created.
RemotePort (see page 1897)	Remote port that this connection is using.

15.1.2.1.1.1 ConColumns Members

The following tables list the members exposed by ConColumns.

Public Fields

	Name	Description
	Birth (see page 1896)	This is Birth, a member of class ConColumns.
	FileName (see page 1896)	This is FileName, a member of class ConColumns.
	Host (see page 1896)	This is Host, a member of class ConColumns.
	Id (see page 1896)	This is Id, a member of class ConColumns.
	Ip (see page 1896)	This is Ip, a member of class ConColumns.
	Remarks (see page 1897)	This is Remarks, a member of class ConColumns.
	RemotePort (see page 1897)	This is RemotePort, a member of class ConColumns.
	User (see page 1897)	This is User, a member of class ConColumns.

15.1.2.1.1.2 ConColumns Fields

The fields of the ConColumns class are listed here.

Public Fields

	Name	Description
	Birth (see page 1896)	This is Birth, a member of class ConColumns.
	FileName (see page 1896)	This is FileName, a member of class ConColumns.
	Host (see page 1896)	This is Host, a member of class ConColumns.
	Id (see page 1896)	This is Id, a member of class ConColumns.
	Ip (see page 1896)	This is Ip, a member of class ConColumns.
	Remarks (see page 1897)	This is Remarks, a member of class ConColumns.
	RemotePort (see page 1897)	This is RemotePort, a member of class ConColumns.
	User (see page 1897)	This is User, a member of class ConColumns.

15.1.2.1.1.2.1 ConColumns.Birth Field

Pascal

```
const Birth: string = "BIRTH";
```

C++

```
string Birth = "BIRTH";
```

Description

This is Birth, a member of class ConColumns.

15.1.2.1.1.2.2 ConColumns.FileName Field

Pascal

```
const FileName: string = "FILE_NAME";
```

C++

```
string FileName = "FILE_NAME";
```

Description

This is FileName, a member of class ConColumns.

15.1.2.1.1.2.3 ConColumns.Host Field

Pascal

```
const Host: string = "HOST";
```

C++

```
string Host = "HOST";
```

Description

This is Host, a member of class ConColumns.

15.1.2.1.1.2.4 ConColumns.Id Field

Pascal

```
const Id: string = "ID";
```

C++

```
string Id = "ID";
```

Description

This is Id, a member of class ConColumns.

15.1.2.1.1.2.5 ConColumns.Ip Field

Pascal

```
const Ip: string = "IP";
```

C++

```
string Ip = "IP";
```

Description

This is Ip, a member of class ConColumns.

15.1.2.1.1.2.6 ConColumns.Remarks Field

Pascal

```
const Remarks: string = "REMARKS";
```

C++

```
string Remarks = "REMARKS";
```

Description

This is Remarks, a member of class ConColumns.

15.1.2.1.1.2.7 ConColumns.RemotePort Field

Pascal

```
const RemotePort: string = "REMOTE_PORT";
```

C++

```
string RemotePort = "REMOTE_PORT";
```

Description

This is RemotePort, a member of class ConColumns.

15.1.2.1.1.2.8 ConColumns.User Field

Pascal

```
const User: string = "USER";
```

C++

```
string User = "USER";
```

Description

This is User, a member of class ConColumns.

15.1.2.1.2 ConnectionProperties Class

Names of connection properties supported by Blackfish SQL driver.

Class Hierarchy

Pascal

```
ConnectionProperties = class;
```

C++

```
class ConnectionProperties;
```

File

ConnectionProperties.cs

Description

This class specifies the names of connection properties supported by the Blackfish SQL driver. These constants can be used to ensure that an application is using the correct name for a property. Connection properties should be combined with semicolons into a string, which is used for the ConnectionString property.

This table contains connection property names.

Name	Description
User (see page 1906)	User (see page 1906) name for Connection.
Password (see page 1903)	Password (see page 1903) for Connection.
LockWaitTime (see page 1902)	Amount of time in milliseconds to wait for a lock before timing out. Default is 30,000 milliseconds.
ReadOnlyTxDelay (see page 1905)	Specifies maximum amount of time in milliseconds the readOnlyTx connection's view of the data can be shared with other read-only transactions. Default is 5,000 milliseconds. This enables the system to optimize by allowing multiple connections to share the same view (cached data). This also controls how "out of date" the readOnly view of the data is for a readOnly transaction. A value of 0 indicates there should be no delay or sharing of the cached view.
ReadOnlyTx (see page 1905)	Create a connections with read-only transactions. When set to true, only read access is allowed for this connection. Only committed data is seen. Other connections can write to the same database without being blocked by reads from this connection. Default is **false**.
ReadOnly (see page 1905)	Open database in read-only mode. When set, only read access is allowed. Unlike **ReadOnlyTx (see page 1905)**, this property prevents any connection from performing any write operation to the database. This setting is ideal for databases that reside on read-only devices such as a CD-ROM.
TableLockTables (see page 1906)	Disable row locks for the specified tables. List of comma separated names of tables that the lock manager should use table locks instead of row locks. A value of "*" disables row locks for all tables.
LogFilters (see page 1903)	Enable logging for the specified events. This property can be set to one of two types of values: 1) List of comma separated LogFilter (see page 2065) **String** values. 2) The string representation of the one or more LogFilterCodes that have been ORed together before converted to a **String** value.
MaxRowLocks (see page 1903)	Set the maximal number of row locks per table. If the number of acquired row locks exceeds this number for a table, the lock manager escalates to a table lock. Default is 50.
CreateDb (see page 1901)	Set to **true** to create the database that is being opened if it does not already exist. Default is **false**.
HeuristicCompletion (see page 1902)	This property is currently only supported for Java (see page 15681) applications. This property allows you to control the behavior when one or more databases fail during a two-phase commit. When XA transactions are prepared but not finished (no commit or rollback), you can specify one of three possible string settings for this property: • **commit**: causes the transaction to be heuristically committed when DataStore (see page 1892) reopens the database. • **rollback**: causes the transaction to be heuristically rolled back when DataStore (see page 1892) reopens the database. • **none**: causes DataStore (see page 1892) to keep the transaction state when reopening a database. When this option is used, the locks that were held when the transaction was prepared are reacquired and held until the transaction is committed or rolled back by a JTA /JTS-compliant transaction manager.
ShutdownServer (see page 1905)	If **true**, causes a Blackfish SQL Server to close all open databases and shutdown.

PingServer (see page 1904)	If **true**, pings the server without creating a connection.
Fileio (see page 1902)	This property only affects Java (see page 15681) servers file IO. This is used to specify the file IO APIs used for database and log files. This property can be set to **new** or **native**. Setting this property can improve performance, but **new** works only for JDK 1.4 platforms and **native** is currently only implemented for Win32 platforms. See com.borland.datastore.cons.FileIO for more details.
Database (see page 1901)	Specifies file name of database to open.
Host (see page 1902)	Specify host server to connect to.
Port (see page 1904)	Specify port to communicate with a remote server. Default is 2508.
ExtendedMetadata (see page 1902)	Add extra columns to metadata requests. This property only affects Java (see page 15681) drivers. When this boolean property is set to **true**, additional columns of information for java.sql.DatabaseMetaData methods like view source for views and additional metadata columns for tables are provided.
Protocol (see page 1904)	Specify network protocol to access server. The possible values are the strings: **TCP** and **InProcess**. Default is **InProcess**.
Remarks (see page 1905)	Optional remarks explaining reason for this connection.
CommitOnClose (see page 1901)	Commit active transaction when connection is closed. Default is **true**.
Pooling (see page 1904)	Create a connection pool. Set this to **false** if no connection pool is wanted. Default is **true**.
MaxPoolSize (see page 1903)	Maximum number of connections in connection pool. Default is 20.
MinPoolSize (see page 1903)	Initial number of connections created in a new connection pool. Default is 1.
ConnectionTimeout (see page 1901)	Maximum time (in seconds) to wait for a free connection from the pool. Default is 1.
PriorFileName (see page 1904)	File name of a prior version of database. This can be used to open the current database. Currently this is only used to open encrypted databases that have been damaged from the crash recovery of an encrypted database. There is also a DataStoreConnection.setPriorFileName. Note that the prior database is only used to decrypt the user table that is not supposed to be encrypted in the first place. If user connecting does not have startup rights in both the prior and current database, the prior database is not used for the open operation. Once the database affected by this defect has been opened, it should be copied or packed into a new database file.

15.1.2.1.2.1 ConnectionProperties Members

The following tables list the members exposed by ConnectionProperties.

Public Fields

	Name	Description
	ClientDataDirectory (see page 1901)	This is ClientDataDirectory, a member of class ConnectionProperties.
	CommitOnClose (see page 1901)	This is CommitOnClose, a member of class ConnectionProperties.
	ConnectionTimeout (see page 1901)	This is ConnectionTimeout, a member of class ConnectionProperties.
	CreateDb (see page 1901)	This is CreateDb, a member of class ConnectionProperties.
	Database (see page 1901)	This is Database, a member of class ConnectionProperties.
	ExtendedMetadata (see page 1902)	This is ExtendedMetadata, a member of class ConnectionProperties.

	Fileio (see page 1902)	This is Fileio, a member of class ConnectionProperties.
	HeuristicCompletion (see page 1902)	This is HeuristicCompletion, a member of class ConnectionProperties.
	Host (see page 1902)	This is Host, a member of class ConnectionProperties.
	LockWaitTime (see page 1902)	This is LockWaitTime, a member of class ConnectionProperties.
	LogFilters (see page 1903)	This is LogFilters, a member of class ConnectionProperties.
	MaxPoolSize (see page 1903)	This is MaxPoolSize, a member of class ConnectionProperties.
	MaxRowLocks (see page 1903)	This is MaxRowLocks, a member of class ConnectionProperties.
	MinPoolSize (see page 1903)	This is MinPoolSize, a member of class ConnectionProperties.
	Password (see page 1903)	This is Password, a member of class ConnectionProperties.
	PingServer (see page 1904)	This is PingServer, a member of class ConnectionProperties.
	Pooling (see page 1904)	This is Pooling, a member of class ConnectionProperties.
	Port (see page 1904)	This is Port, a member of class ConnectionProperties.
	PriorFileName (see page 1904)	This is PriorFileName, a member of class ConnectionProperties.
	Protocol (see page 1904)	This is Protocol, a member of class ConnectionProperties.
	ReadOnly (see page 1905)	This is ReadOnly, a member of class ConnectionProperties.
	ReadOnlyTx (see page 1905)	This is ReadOnlyTx, a member of class ConnectionProperties.
	ReadOnlyTxDelay (see page 1905)	This is ReadOnlyTxDelay, a member of class ConnectionProperties.
	Remarks (see page 1905)	This is Remarks, a member of class ConnectionProperties.
	ShutdownServer (see page 1905)	This is ShutdownServer, a member of class ConnectionProperties.
	TableLockTables (see page 1906)	This is TableLockTables, a member of class ConnectionProperties.
	User (see page 1906)	This is User, a member of class ConnectionProperties.

15.1.2.1.2.2 ConnectionProperties Fields

The fields of the ConnectionProperties class are listed here.

Public Fields

	Name	Description
	ClientDataDirectory (see page 1901)	This is ClientDataDirectory, a member of class ConnectionProperties.
	CommitOnClose (see page 1901)	This is CommitOnClose, a member of class ConnectionProperties.
	ConnectionTimeout (see page 1901)	This is ConnectionTimeout, a member of class ConnectionProperties.
	CreateDb (see page 1901)	This is CreateDb, a member of class ConnectionProperties.
	Database (see page 1901)	This is Database, a member of class ConnectionProperties.
	ExtendedMetadata (see page 1902)	This is ExtendedMetadata, a member of class ConnectionProperties.
	Fileio (see page 1902)	This is Fileio, a member of class ConnectionProperties.
	HeuristicCompletion (see page 1902)	This is HeuristicCompletion, a member of class ConnectionProperties.
	Host (see page 1902)	This is Host, a member of class ConnectionProperties.
	LockWaitTime (see page 1902)	This is LockWaitTime, a member of class ConnectionProperties.
	LogFilters (see page 1903)	This is LogFilters, a member of class ConnectionProperties.
	MaxPoolSize (see page 1903)	This is MaxPoolSize, a member of class ConnectionProperties.
	MaxRowLocks (see page 1903)	This is MaxRowLocks, a member of class ConnectionProperties.
	MinPoolSize (see page 1903)	This is MinPoolSize, a member of class ConnectionProperties.
	Password (see page 1903)	This is Password, a member of class ConnectionProperties.
	PingServer (see page 1904)	This is PingServer, a member of class ConnectionProperties.
	Pooling (see page 1904)	This is Pooling, a member of class ConnectionProperties.
	Port (see page 1904)	This is Port, a member of class ConnectionProperties.
	PriorFileName (see page 1904)	This is PriorFileName, a member of class ConnectionProperties.
	Protocol (see page 1904)	This is Protocol, a member of class ConnectionProperties.
	ReadOnly (see page 1905)	This is ReadOnly, a member of class ConnectionProperties.
	ReadOnlyTx (see page 1905)	This is ReadOnlyTx, a member of class ConnectionProperties.
	ReadOnlyTxDelay (see page 1905)	This is ReadOnlyTxDelay, a member of class ConnectionProperties.
	Remarks (see page 1905)	This is Remarks, a member of class ConnectionProperties.
	ShutdownServer (see page 1905)	This is ShutdownServer, a member of class ConnectionProperties.
	TableLockTables (see page 1906)	This is TableLockTables, a member of class ConnectionProperties.
	User (see page 1906)	This is User, a member of class ConnectionProperties.

15.1.2.1.2.2.1 ConnectionProperties.ClientDataDirectory Field

Pascal

```
const ClientDataDirectory: string = "clientDataDirectory";
```

C++

```
string ClientDataDirectory = "clientDataDirectory";
```

Description

This is ClientDataDirectory, a member of class ConnectionProperties.

15.1.2.1.2.2.2 ConnectionProperties.CommitOnClose Field

Pascal

```
const CommitOnClose: string = "commitOnClose";
```

C++

```
string CommitOnClose = "commitOnClose";
```

Description

This is CommitOnClose, a member of class ConnectionProperties.

15.1.2.1.2.2.3 ConnectionProperties.ConnectionTimeout Field

Pascal

```
const ConnectionTimeout: string = "connectionTimeout";
```

C++

```
string ConnectionTimeout = "connectionTimeout";
```

Description

This is ConnectionTimeout, a member of class ConnectionProperties.

15.1.2.1.2.2.4 ConnectionProperties.CreateDb Field

Pascal

```
const CreateDb: string = "create";
```

C++

```
string CreateDb = "create";
```

Description

This is CreateDb, a member of class ConnectionProperties.

15.1.2.1.2.2.5 ConnectionProperties.Database Field

Pascal

```
const Database: string = "database";
```

C++

```
string Database = "database";
```

Description

This is Database, a member of class ConnectionProperties.

15.1.2.1.2.2.6 ConnectionProperties.ExtendedMetadata Field

Pascal

```
const ExtendedMetadata: string = "extendedMetaData";
```

C++

```
string ExtendedMetadata = "extendedMetaData";
```

Description

This is ExtendedMetadata, a member of class ConnectionProperties.

15.1.2.1.2.2.7 ConnectionProperties.Fileio Field

Pascal

```
const Fileio: string = "fileio";
```

C++

```
string Fileio = "fileio";
```

Description

This is Fileio, a member of class ConnectionProperties.

15.1.2.1.2.2.8 ConnectionProperties.HeuristicCompletion Field

Pascal

```
const HeuristicCompletion: string = "heuristicCompletion";
```

C++

```
string HeuristicCompletion = "heuristicCompletion";
```

Description

This is HeuristicCompletion, a member of class ConnectionProperties.

15.1.2.1.2.2.9 ConnectionProperties.Host Field

Pascal

```
const Host: string = "host";
```

C++

```
string Host = "host";
```

Description

This is Host, a member of class ConnectionProperties.

15.1.2.1.2.2.10 ConnectionProperties.LockWaitTime Field

Pascal

```
const LockWaitTime: string = "lockWaitTime";
```

C++

```
string LockWaitTime = "lockWaitTime";
```

Description

This is LockWaitTime, a member of class ConnectionProperties.

15.1.2.1.2.2.11 ConnectionProperties.LogFilters Field

Pascal

```
const LogFilters: string = "logFilters";
```

C++

```
string LogFilters = "logFilters";
```

Description

This is LogFilters, a member of class ConnectionProperties.

15.1.2.1.2.2.12 ConnectionProperties.MaxPoolSize Field

Pascal

```
const MaxPoolSize: string = "maxPoolSize";
```

C++

```
string MaxPoolSize = "maxPoolSize";
```

Description

This is MaxPoolSize, a member of class ConnectionProperties.

15.1.2.1.2.2.13 ConnectionProperties.MaxRowLocks Field

Pascal

```
const MaxRowLocks: string = "maxRowLocks";
```

C++

```
string MaxRowLocks = "maxRowLocks";
```

Description

This is MaxRowLocks, a member of class ConnectionProperties.

15.1.2.1.2.2.14 ConnectionProperties.MinPoolSize Field

Pascal

```
const MinPoolSize: string = "minPoolSize";
```

C++

```
string MinPoolSize = "minPoolSize";
```

Description

This is MinPoolSize, a member of class ConnectionProperties.

15.1.2.1.2.2.15 ConnectionProperties.Password Field

Pascal

```
const Password: string = "password";
```

C++

```
string Password = "password";
```

Description

This is Password, a member of class ConnectionProperties.

15.1.2.1.2.2.16 ConnectionProperties.PingServer Field

Pascal

```
const PingServer: string = "pingServer";
```

C++

```
string PingServer = "pingServer";
```

Description

This is PingServer, a member of class ConnectionProperties.

15.1.2.1.2.2.17 ConnectionProperties.Pooling Field

Pascal

```
const Pooling: string = "pooling";
```

C++

```
string Pooling = "pooling";
```

Description

This is Pooling, a member of class ConnectionProperties.

15.1.2.1.2.2.18 ConnectionProperties.Port Field

Pascal

```
const Port: string = Borland.Data.DataStore.SystemProperties.Port;
```

C++

```
string Port = Borland.Data.DataStore.SystemProperties.Port;
```

Description

This is Port, a member of class ConnectionProperties.

15.1.2.1.2.2.19 ConnectionProperties.PriorFileName Field

Pascal

```
const PriorFileName: string = "priorFileName";
```

C++

```
string PriorFileName = "priorFileName";
```

Description

This is PriorFileName, a member of class ConnectionProperties.

15.1.2.1.2.2.20 ConnectionProperties.Protocol Field

Pascal

```
const Protocol: string = "protocol";
```

C++

```
string Protocol = "protocol";
```

Description

This is Protocol, a member of class ConnectionProperties.

15.1.2.1.2.2.21 ConnectionProperties.ReadOnly Field

Pascal

```
const ReadOnly: string = "readonly";
```

C++

```
string ReadOnly = "readonly";
```

Description

This is ReadOnly, a member of class ConnectionProperties.

15.1.2.1.2.2.22 ConnectionProperties.ReadOnlyTx Field

Pascal

```
const ReadOnlyTx: string = "readOnlyTx";
```

C++

```
string ReadOnlyTx = "readOnlyTx";
```

Description

This is ReadOnlyTx, a member of class ConnectionProperties.

15.1.2.1.2.2.23 ConnectionProperties.ReadOnlyTxDelay Field

Pascal

```
const ReadOnlyTxDelay: string = "readOnlyTxDelay";
```

C++

```
string ReadOnlyTxDelay = "readOnlyTxDelay";
```

Description

This is ReadOnlyTxDelay, a member of class ConnectionProperties.

15.1.2.1.2.2.24 ConnectionProperties.Remarks Field

Pascal

```
const Remarks: string = "remarks";
```

C++

```
string Remarks = "remarks";
```

Description

This is Remarks, a member of class ConnectionProperties.

15.1.2.1.2.2.25 ConnectionProperties.ShutdownServer Field

Pascal

```
const ShutdownServer: string = "shutdownServer";
```

C++

```
string ShutdownServer = "shutdownServer";
```

Description

This is ShutdownServer, a member of class ConnectionProperties.

15.1.2.1.2.2.26 ConnectionProperties.TableLockTables Field

Pascal

```
const TableLockTables: string = "tableLockTables";
```

C++

```
string TableLockTables = "tableLockTables";
```

Description

This is TableLockTables, a member of class ConnectionProperties.

15.1.2.1.2.2.27 ConnectionProperties.User Field

Pascal

```
const User: string = "user";
```

C++

```
string User = "user";
```

Description

This is User, a member of class ConnectionProperties.

15.1.2.1.3 DatabaseColumns Class

Column names for result table returned from DB_ADMIN.GET_DATABASE_PROPS.

Class Hierarchy

Borland.Data.DataStore.DatabaseColumns

Pascal

```
DatabaseColumns = class;
```

C++

```
class DatabaseColumns;
```

File

DatabaseColumns.cs

Description

Column names for result table returned from the built-in stored procedure DB_ADMIN.GET_DATABASE_PROPS. The result table contains one row with the following columns:

Column	Description
BLOCK_SIZE	Database block size. The size in bytes of each read/write operation for the database file. A row of a table must fit in a block.
STORE_VERSION	Database file version. One of the values from **StoreVersion (see page 1910)**.
LOG_BLOCK_SIZE	Block size used for log files.
A_LOG_DIR	Directory for log files.
MAX_OPEN_LOGS	Maximum log files open at one time.
MAX_LOG_SIZE	Maximum log file size in bytes.
RECORD_STATUS	Send log output to the status log files.

LOG_ENABLED	Transaction logging enabled.
LOG_PREALLOCATE_COUNT	Pre-allocation log file count.
SOFT_COMMIT	Soft commit enabled.
CHECKPOINT_FREQUENCY	Checkpoint frequency.
DEFAULT_SCHEMA_NAME	The default schema for the current user. If the user has not created his/her own schema this is **DEFAULT_SCHEMA**. If a user has created his/her own schema this will be the current user name.

See Also

DB_ADMIN

GET_DATABASE_PROPS

15.1.2.1.3.1 DatabaseColumns Members

The following tables list the members exposed by DatabaseColumns.

Public Fields

	Name	Description
	ALogDir (see page 1908)	This is ALogDir, a member of class DatabaseColumns.
	BlockSize (see page 1908)	This is BlockSize, a member of class DatabaseColumns.
	CheckpointFrequency (see page 1908)	This is CheckpointFrequency, a member of class DatabaseColumns.
	DefaultSchemaName (see page 1908)	This is DefaultSchemaName, a member of class DatabaseColumns.
	LogBlockSize (see page 1908)	This is LogBlockSize, a member of class DatabaseColumns.
	LogEnabled (see page 1909)	This is LogEnabled, a member of class DatabaseColumns.
	LogPreallocateCount (see page 1909)	This is LogPreallocateCount, a member of class DatabaseColumns.
	MaxLogSize (see page 1909)	This is MaxLogSize, a member of class DatabaseColumns.
	MaxOpenLogs (see page 1909)	This is MaxOpenLogs, a member of class DatabaseColumns.
	RecordStatus (see page 1909)	This is RecordStatus, a member of class DatabaseColumns.
	SoftCommit (see page 1910)	This is SoftCommit, a member of class DatabaseColumns.
	StoreVersion (see page 1910)	This is StoreVersion, a member of class DatabaseColumns.

15.1.2.1.3.2 DatabaseColumns Fields

The fields of the DatabaseColumns class are listed here.

Public Fields

	Name	Description
	ALogDir (see page 1908)	This is ALogDir, a member of class DatabaseColumns.
	BlockSize (see page 1908)	This is BlockSize, a member of class DatabaseColumns.
	CheckpointFrequency (see page 1908)	This is CheckpointFrequency, a member of class DatabaseColumns.
	DefaultSchemaName (see page 1908)	This is DefaultSchemaName, a member of class DatabaseColumns.
	LogBlockSize (see page 1908)	This is LogBlockSize, a member of class DatabaseColumns.
	LogEnabled (see page 1909)	This is LogEnabled, a member of class DatabaseColumns.
	LogPreallocateCount (see page 1909)	This is LogPreallocateCount, a member of class DatabaseColumns.
	MaxLogSize (see page 1909)	This is MaxLogSize, a member of class DatabaseColumns.
	MaxOpenLogs (see page 1909)	This is MaxOpenLogs, a member of class DatabaseColumns.
	RecordStatus (see page 1909)	This is RecordStatus, a member of class DatabaseColumns.
	SoftCommit (see page 1910)	This is SoftCommit, a member of class DatabaseColumns.
	StoreVersion (see page 1910)	This is StoreVersion, a member of class DatabaseColumns.

15.1.2.1.3.2.1 DatabaseColumns.ALogDir Field

Pascal

```
const ALogDir: string = "A_LOG_DIR";
```

C++

```
string ALogDir = "A_LOG_DIR";
```

Description

This is ALogDir, a member of class DatabaseColumns.

15.1.2.1.3.2.2 DatabaseColumns.BlockSize Field

Pascal

```
const BlockSize: string = "BLOCK_SIZE";
```

C++

```
string BlockSize = "BLOCK_SIZE";
```

Description

This is BlockSize, a member of class DatabaseColumns.

15.1.2.1.3.2.3 DatabaseColumns.CheckpointFrequency Field

Pascal

```
const CheckpointFrequency: string = "CHECKPOINT_FREQUENCY";
```

C++

```
string CheckpointFrequency = "CHECKPOINT_FREQUENCY";
```

Description

This is CheckpointFrequency, a member of class DatabaseColumns.

15.1.2.1.3.2.4 DatabaseColumns.DefaultSchemaName Field

Pascal

```
const DefaultSchemaName: string = "DEFAULT_SCHEMA_NAME";
```

C++

```
string DefaultSchemaName = "DEFAULT_SCHEMA_NAME";
```

Description

This is DefaultSchemaName, a member of class DatabaseColumns.

15.1.2.1.3.2.5 DatabaseColumns.LogBlockSize Field

Pascal

```
const LogBlockSize: string = "LOG_BLOCK_SIZE";
```

C++

```
string LogBlockSize = "LOG_BLOCK_SIZE";
```

Description

This is LogBlockSize, a member of class DatabaseColumns.

15.1.2.1.3.2.6 DatabaseColumns.LogEnabled Field

Pascal

```
const LogEnabled: string = "LOG_ENABLED";
```

C++

```
string LogEnabled = "LOG_ENABLED";
```

Description

This is LogEnabled, a member of class DatabaseColumns.

15.1.2.1.3.2.7 DatabaseColumns.LogPreallocateCount Field

Pascal

```
const LogPreallocateCount: string = "LOG_PREALLOCATE_COUNT";
```

C++

```
string LogPreallocateCount = "LOG_PREALLOCATE_COUNT";
```

Description

This is LogPreallocateCount, a member of class DatabaseColumns.

15.1.2.1.3.2.8 DatabaseColumns.MaxLogSize Field

Pascal

```
const MaxLogSize: string = "MAX_LOG_SIZE";
```

C++

```
string MaxLogSize = "MAX_LOG_SIZE";
```

Description

This is MaxLogSize, a member of class DatabaseColumns.

15.1.2.1.3.2.9 DatabaseColumns.MaxOpenLogs Field

Pascal

```
const MaxOpenLogs: string = "MAX_OPEN_LOGS";
```

C++

```
string MaxOpenLogs = "MAX_OPEN_LOGS";
```

Description

This is MaxOpenLogs, a member of class DatabaseColumns.

15.1.2.1.3.2.10 DatabaseColumns.RecordStatus Field

Pascal

```
const RecordStatus: string = "RECORD_STATUS";
```

C++

```
string RecordStatus = "RECORD_STATUS";
```

Description

This is RecordStatus, a member of class DatabaseColumns.

15.1.2.1.3.2.11 DatabaseColumns.SoftCommit Field

Pascal

```
const SoftCommit: string = "SOFT_COMMIT";
```

C++

```
string SoftCommit = "SOFT_COMMIT";
```

Description

This is SoftCommit, a member of class DatabaseColumns.

15.1.2.1.3.2.12 DatabaseColumns.StoreVersion Field

Pascal

```
const StoreVersion: string = "STORE_VERSION";
```

C++

```
string StoreVersion = "STORE_VERSION";
```

Description

This is StoreVersion, a member of class DatabaseColumns.

15.1.2.1.4 DatabaseStatusTable Class

Provides the column names for the result table returned from the builtin stored procedure DB_ADMIN.GET_DATABASE_STATUS(). The DB_ADMIN.GET_DATABASE_STATUS() stored procedure provides a table of information about the status of the connected database.

Class Hierarchy

Pascal

```
DatabaseStatusTable = class;
```

C++

```
class DatabaseStatusTable;
```

File

DatabaseStatusColumns.cs

Description

Provides the column names for the result table returned from the builtin stored procedure DB_ADMIN.GET_DATABASE_STATUS(). The DB_ADMIN.GET_DATABASE_STATUS() stored procedure provides a table of information about the status of the connected database.

The columns of the result table are:

Constant	Description
LowestLsn (see page 1911)	Lowest log sequence number.
LowestLsnRemarks (see page 1912)	Lowest log sequence number remarks.
HighestLsn (see page 1911)	Highest log sequence number.
LastCheckpointLsn (see page 1911)	Last checkpoint log sequence number.

15.1.2.1.4.1 DatabaseStatusTable Members

The following tables list the members exposed by DatabaseStatusTable.

Public Fields

	Name	Description
	HighestLsn (see page 1911)	This is HighestLsn, a member of class DatabaseStatusTable.
	LastCheckpointLsn (see page 1911)	This is LastCheckpointLsn, a member of class DatabaseStatusTable.
	LowestLsn (see page 1911)	This is LowestLsn, a member of class DatabaseStatusTable.
	LowestLsnRemarks (see page 1912)	This is LowestLsnRemarks, a member of class DatabaseStatusTable.

15.1.2.1.4.2 DatabaseStatusTable Fields

The fields of the DatabaseStatusTable class are listed here.

Public Fields

	Name	Description
	HighestLsn (see page 1911)	This is HighestLsn, a member of class DatabaseStatusTable.
	LastCheckpointLsn (see page 1911)	This is LastCheckpointLsn, a member of class DatabaseStatusTable.
	LowestLsn (see page 1911)	This is LowestLsn, a member of class DatabaseStatusTable.
	LowestLsnRemarks (see page 1912)	This is LowestLsnRemarks, a member of class DatabaseStatusTable.

15.1.2.1.4.2.1 DatabaseStatusTable.HighestLsn Field

Pascal

```
const HighestLsn: string = "HIGHEST_LSN";
```

C++

```
string HighestLsn = "HIGHEST_LSN";
```

Description

This is HighestLsn, a member of class DatabaseStatusTable.

15.1.2.1.4.2.2 DatabaseStatusTable.LastCheckpointLsn Field

Pascal

```
const LastCheckpointLsn: string = "LAST_CHECKPOINT_LSN";
```

C++

```
string LastCheckpointLsn = "LAST_CHECKPOINT_LSN";
```

Description

This is LastCheckpointLsn, a member of class DatabaseStatusTable.

15.1.2.1.4.2.3 DatabaseStatusTable.LowestLsn Field

Pascal

```
const LowestLsn: string = "LOWEST_LSN";
```

C++

```
string LowestLsn = "LOWEST_LSN";
```

Description

This is LowestLsn, a member of class DatabaseStatusTable.

15.1.2.1.4.2.4 DatabaseStatusTable.LowestLsnRemarks Field

Pascal

```
const LowestLsnRemarks: string = "LOWEST_LSN_REMARKS";
```

C++

```
string LowestLsnRemarks = "LOWEST_LSN_REMARKS";
```

Description

This is LowestLsnRemarks, a member of class DatabaseStatusTable.

15.1.2.1.5 DataStoreCommand Class

Provides execution of SQL statements and execution of stored procedures.

Class Hierarchy

Pascal

```
DataStoreCommand = class(DbCommand, IDbCommand, ICloneable, IDisposable);
```

C++

```
class DataStoreCommand : public DbCommand, public IDbCommand, public ICloneable, public
IDisposable;
```

File

Ado.Net20.cs

Description

DataStoreCommand provides execution of SQL statements and execution of stored procedures.

DataStoreCommand is an implementation of the .NET DbCommand class.

After associating a command object with a connection object, the CommandText (see page 1919) property should be set to a SQL statement or a stored procedure name. If a stored procedure name is used, the CommandType (see page 1920) property should be set to CommandType (see page 1920).**StoredProcedure**. Then the command can be executed using one of these methods:

- ExecuteReader (see page 1918): Executes a command that returns rows. Most **SELECT** statements should use this method. If the actual command has no rows, the resulting DataStoreDataReader returns **false** from the DataStoreDataReader.Read method and the DataStoreDataReader.FieldCount property yields 0.
- ExecuteNonQuery (see page 1917): Executes a command that does not return rows, such as **INSERT**, **UPDATE**, **DELETE** and **DDL** statements. If the actual command has result rows, they are discarded at the server.
- ExecuteScalar (see page 1918): Retrieves a single value from a resulting row. CommandText (see page 1919) can be assigned a different value after executing one statement.

See Also

DbCommand (MSDN)

DataStoreDataReader

CommandText (see page 1919)

CommandType (see page 1920)

FieldCount

ExecuteNonQuery (see page 1917)

ExecuteReader (see page 1918)

ExecuteScalar (see page 1918)

Read

15.1.2.1.5.1 DataStoreCommand Members

The following tables list the members exposed by DataStoreCommand.

Internal Methods

	Name	Description
	DataReaderClosed (see page 1916)	This is DataReaderClosed, a member of class DataStoreCommand.
	FailIfNotOpen (see page 1919)	This is FailIfNotOpen, a member of class DataStoreCommand.

Protected Methods

	Name	Description
	CreateDbParameter (see page 1916)	This is CreateDbParameter, a member of class DataStoreCommand.
	ExecuteDbDataReader (see page 1917)	This is ExecuteDbDataReader, a member of class DataStoreCommand.

Protected Properties

	Name	Description
	DbConnection (see page 1921)	This is DbConnection, a member of class DataStoreCommand.
	DbParameterCollection (see page 1921)	This is DbParameterCollection, a member of class DataStoreCommand.
	DbTransaction (see page 1922)	This is DbTransaction, a member of class DataStoreCommand.

Public Methods

	Name	Description
	Cancel (see page 1915)	Attempts to cancels execution of a DataStoreCommand (see page 1912).
	Clone (see page 1915)	Creates a new object that is a copy of the current instance.
	Close (see page 1916)	Release resources held by command.
	CreateParameter (see page 1916)	Creates and returns a DataStoreParameter (see page 2033) that can be added to the Parameters (see page 1922) of this command.
	DataStoreCommand (see page 1914)	This is the overview for the DataStoreCommand constructor overload.
	Dispose (see page 1917)	This is the overview for the Dispose method overload.
	ExecuteNonQuery (see page 1917)	Executes a SQL statement and returns the number of rows affected.
	ExecuteReader (see page 1918)	This is the overview for the ExecuteReader method overload.
	ExecuteScalar (see page 1918)	Executes query and returns first column of first row in result set.
	Prepare (see page 1919)	Prepares statement for execution on server.

Public Properties

	Name	Description
	CommandText (see page 1919)	Gets or sets Blackfish SQL SQL statement or stored procedure name.
	CommandTimeout (see page 1920)	Gets or sets wait time before terminating the execution of command.
	CommandType (see page 1920)	Gets or sets a value specifying how the CommandText (see page 1919) property should be used.
	Connection (see page 1921)	Gets or sets the DataStoreConnection (see page 1923) used by this command.
	DesignTimeVisible (see page 1922)	Command object should be visible in customized interface control.
	Parameters (see page 1922)	Gets the DataStoreParameterCollection (see page 2041) for this command.
	RecordsAffected (see page 1922)	This is RecordsAffected, a member of class DataStoreCommand.
	Transaction (see page 1923)	Get transaction of command.

	UpdatedRowSource (see page 1923)	How command results are applied to DataRow when used by Update method of DataStoreDataAdapter (see page 1933).

15.1.2.1.5.2 DataStoreCommand Constructor

This is the overview for the DataStoreCommand constructor overload.

Overload List

	Name	Description
	DataStoreCommand.DataStoreCommand () (see page 1914)	Initializes a new instance of the DataStoreCommand class.
	DataStoreCommand.DataStoreCommand (string) (see page 1914)	Initializes a new instance of the DataStoreCommand class.
	DataStoreCommand.DataStoreCommand (string, DataStoreConnection) (see page 1914)	Initializes a new instance of the DataStoreCommand class.
	DataStoreCommand.DataStoreCommand (string, DataStoreConnection, DataStoreTransaction) (see page 1914)	Initializes a new instance of the DataStoreCommand class.

15.1.2.1.5.2.1 DataStoreCommand.DataStoreCommand Constructor ()

Initializes a new instance of the DataStoreCommand class.

Pascal

```
constructor Create;
constructor Create(cmdText: string);
constructor Create(cmdText: string; connection: DataStoreConnection);
constructor Create(cmdText: string; connection: DataStoreConnection; transaction:
DataStoreTransaction);
```

C++

```
DataStoreCommand();
DataStoreCommand(string cmdText);
DataStoreCommand(string cmdText, DataStoreConnection connection);
DataStoreCommand(string cmdText, DataStoreConnection connection, DataStoreTransaction
transaction);
```

Description

DataStoreCommand initializes a new instance of the DataStoreCommand class, which is an implementation of the DbCommand class.

When a new instance of DataStoreCommand is created, the command properties Connection (see page 1921) and CommandText (see page 1919) should be set.

One form of this constructor sets the initial value for CommandText (see page 1919).

One form of this constructor sets the initial value for the properties CommandText (see page 1919) and Connection (see page 1921).

See Also

DbCommand (MSDN)

CommandText (see page 1919)

Connection (see page 1921)

15.1.2.1.5.3 DataStoreCommand Methods

The methods of the DataStoreCommand class are listed here.

Internal Methods

	Name	Description
	DataReaderClosed (see page 1916)	This is DataReaderClosed, a member of class DataStoreCommand.
	FailIfNotOpen (see page 1919)	This is FailIfNotOpen, a member of class DataStoreCommand.

Protected Methods

	Name	Description
	CreateDbParameter (see page 1916)	This is CreateDbParameter, a member of class DataStoreCommand.
	ExecuteDbDataReader (see page 1917)	This is ExecuteDbDataReader, a member of class DataStoreCommand.

Public Methods

	Name	Description
	Cancel (see page 1915)	Attempts to cancels execution of a DataStoreCommand (see page 1912).
	Clone (see page 1915)	Creates a new object that is a copy of the current instance.
	Close (see page 1916)	Release resources held by command.
	CreateParameter (see page 1916)	Creates and returns a DataStoreParameter (see page 2033) that can be added to the Parameters (see page 1922) of this command.
	Dispose (see page 1917)	This is the overview for the Dispose method overload.
	ExecuteNonQuery (see page 1917)	Executes a SQL statement and returns the number of rows affected.
	ExecuteReader (see page 1918)	This is the overview for the ExecuteReader method overload.
	ExecuteScalar (see page 1918)	Executes query and returns first column of first row in result set.
	Prepare (see page 1919)	Prepares statement for execution on server.

15.1.2.1.5.3.1 DataStoreCommand.Cancel Method

Attempts to cancels execution of a DataStoreCommand (see page 1912).

Pascal

```
function Cancel: void;
```

C++

```
void Cancel();
```

Description

Attempts to cancels the execution of a DataStoreCommand (see page 1912).

See Also

Cancel (MSDN)

15.1.2.1.5.3.2 DataStoreCommand.Clone Method

Creates a new object that is a copy of the current instance.

Pascal

```
function Clone: DataStoreCommand;
```

C++

```
DataStoreCommand Clone();
```

Description

Clone creates a new object that is a copy of the current instance.

See Also

Clone (MSDN)

15.1.2.1.5.3.3 DataStoreCommand.Close Method

Release resources held by command.

Pascal

```
function Close: void;
```

C++

```
void Close();
```

Description

Close releases the resources held by this command.

15.1.2.1.5.3.4 DataStoreCommand.CreateDbParameter Method

Pascal

```
function CreateDbParameter: DbParameter;
```

C++

```
DbParameter CreateDbParameter();
```

Description

This is CreateDbParameter, a member of class DataStoreCommand.

15.1.2.1.5.3.5 DataStoreCommand.CreateParameter Method

Creates and returns a DataStoreParameter (see page 2033) that can be added to the Parameters (see page 1922) of this command.

Pascal

```
function CreateParameter: DataStoreParameter;
```

C++

```
DataStoreParameter CreateParameter();
```

Description

CreateParameter creates and returns a DataStoreParameter (see page 2033) that can be added to the Parameters (see page 1922) of this command.

See Also

CreateParameter (MSDN)

15.1.2.1.5.3.6 DataStoreCommand.DataReaderClosed Method

Pascal

```
function DataReaderClosed(behavior: CommandBehavior): void;
```

C++

```
void DataReaderClosed(CommandBehavior behavior);
```

Description

This is DataReaderClosed, a member of class DataStoreCommand.

15.1.2.1.5.3.7 Dispose Method

This is the overview for the Dispose method overload.

Overload List

	Name	Description
	DataStoreCommand.Dispose () (see page 1917)	Releases the resources used by the DataStoreCommand (see page 1912).
	DataStoreCommand.Dispose (bool) (see page 1917)	Releases the resources used by the DataStoreCommand (see page 1912).

15.1.2.1.5.3.7.1 DataStoreCommand.Dispose Method ()

Releases the resources used by the DataStoreCommand (see page 1912).

Pascal

```
function Dispose: void;
function Dispose(disposing: bool): void;
```

C++

```
void Dispose();
void Dispose(bool disposing);
```

Description

Dispose releases the resources used by the DataStoreCommand (see page 1912).

See Also

Dispose (MSDN)

15.1.2.1.5.3.8 DataStoreCommand.ExecuteDbDataReader Method

Pascal

```
function ExecuteDbDataReader(behavior: CommandBehavior): DbDataReader;
```

C++

```
DbDataReader ExecuteDbDataReader(CommandBehavior behavior);
```

Description

This is ExecuteDbDataReader, a member of class DataStoreCommand.

15.1.2.1.5.3.9 DataStoreCommand.ExecuteNonQuery Method

Executes a SQL statement and returns the number of rows affected.

Pascal

```
function ExecuteNonQuery: Integer;
```

C++

```
int ExecuteNonQuery();
```

Description

ExecuteNonQuery executes a SQL statement and returns the number of rows affected by executing this command. Use this method to execute SQL statements that do not generate a result table.

See Also

ExecuteNonQuery (MSDN)

15.1.2.1.5.3.10 ExecuteReader Method

This is the overview for the ExecuteReader method overload.

Overload List

	Name	Description
	DataStoreCommand.ExecuteReader () (see page 1918)	Returns a DataStoreDataReader instance.
	DataStoreCommand.ExecuteReader (CommandBehavior) (see page 1918)	Returns a DataStoreDataReader instance.

15.1.2.1.5.3.10.1 DataStoreCommand.ExecuteReader Method ()

Returns a DataStoreDataReader instance.

Pascal

```
function ExecuteReader: DataStoreDataReader;
function ExecuteReader(behavior: CommandBehavior): DataStoreDataReader;
```

C++

```
DataStoreDataReader ExecuteReader();
DataStoreDataReader ExecuteReader(CommandBehavior behavior);
```

Description

ExecuteReader returns a DataStoreDataReader instance.

The parameter **behavior** is one of the **CommandBehavior** constants. Only **CommandBehavior**.**CloseConnection** is currently supported.

See Also

ExecuteReader (MSDN)

15.1.2.1.5.3.11 DataStoreCommand.ExecuteScalar Method

Executes query and returns first column of first row in result set.

Pascal

```
function ExecuteScalar: object;
```

C++

```
object ExecuteScalar();
```

Description

ExecuteScalar executes the query and returns the first column of the first row in the result set returned by the query. Extra columns or rows are ignored. Returns the value of the first column of the first row or DBNull.Value if no such value exists.

See Also

ExecuteScalar (MSDN)

15.1.2.1.5.3.12 DataStoreCommand.FailIfNotOpen Method

Pascal

```
function FailIfNotOpen: void;
```

C++

```
void FailIfNotOpen();
```

Description

This is FailIfNotOpen, a member of class DataStoreCommand.

15.1.2.1.5.3.13 DataStoreCommand.Prepare Method

Prepares statement for execution on server.

Pascal

```
function Prepare: void;
```

C++

```
void Prepare();
```

Description

This method prepares the statement for execution on the server.

See Also

Prepare (MSDN)

15.1.2.1.5.4 DataStoreCommand Properties

The properties of the DataStoreCommand class are listed here.

Protected Properties

	Name	Description
	DbConnection (see page 1921)	This is DbConnection, a member of class DataStoreCommand.
	DbParameterCollection (see page 1921)	This is DbParameterCollection, a member of class DataStoreCommand.
	DbTransaction (see page 1922)	This is DbTransaction, a member of class DataStoreCommand.

Public Properties

	Name	Description
	CommandText (see page 1919)	Gets or sets Blackfish SQL SQL statement or stored procedure name.
	CommandTimeout (see page 1920)	Gets or sets wait time before terminating the execution of command.
	CommandType (see page 1920)	Gets or sets a value specifying how the CommandText (see page 1919) property should be used.
	Connection (see page 1921)	Gets or sets the DataStoreConnection (see page 1923) used by this command.
	DesignTimeVisible (see page 1922)	Command object should be visible in customized interface control.
	Parameters (see page 1922)	Gets the DataStoreParameterCollection (see page 2041) for this command.
	RecordsAffected (see page 1922)	This is RecordsAffected, a member of class DataStoreCommand.
	Transaction (see page 1923)	Get transaction of command.
	UpdatedRowSource (see page 1923)	How command results are applied to DataRow when used by Update method of DataStoreDataAdapter (see page 1933).

15.1.2.1.5.4.1 DataStoreCommand.CommandText Property

Gets or sets Blackfish SQL SQL statement or stored procedure name.

Pascal

```
property CommandText: string;
```

C++

```
__property string CommandText;
```

Description

CommandText holds the Blackfish SQL SQL statement or stored procedure name to execute.

When the CommandType (see page 1920) property is set to CommandType (see page 1920).**Text**, this property should be set to a SQL statement. Parameters (see page 1922) can be marked with either a question mark (?) or named parameter markers. For example:

```
SELECT * FROM EMPLOYEE WHERE EMP_NO=?
SELECT * FROM EMPLOYEE WHERE EMP_NO=EMP
```

When question marks (?) are used, the order of the parameters in Parameters (see page 1922) must match the order in which they appear in the SQL statement. When named parameters are used the names must match, but the order doesn't matter.

When the CommandType (see page 1920) property is set to CommandType (see page 1920).**StoredProcedure**, this property should be set to a stored procedure name. The parameters are implicit. For example:

```
DB_UTIL.SIN
MYSCHEMA.MYPROCEDURE
```

The parameters are given by the name of the parameters of the stored procedure. You can find the names of the stored procedure from ISQL or from the built-in stored procedure DB_ADMIN.GET_PROCEDURE_COLUMNS.

See Also

CommandText (MSDN)

DB_ADMIN

GET_PROCEDURE_COLUMNS

15.1.2.1.5.4.2 DataStoreCommand.CommandTimeout Property

Gets or sets wait time before terminating the execution of command.

Pascal

```
property CommandTimeout: Integer;
```

C++

```
__property int CommandTimeout;
```

Description

CommandTimeout gets or sets the wait time before terminating the execution a command and generating an error. Zero indicates an indefinite time-out period.

See Also

CommandTimeout (MSDN)

15.1.2.1.5.4.3 DataStoreCommand.CommandType Property

Gets or sets a value specifying how the CommandText (see page 1919) property should be used.

Pascal

```
property CommandType: CommandType;
```

C++

```
__property CommandType CommandType;
```

Description

CommandType gets or sets a value specifying how the CommandText (see page 1919) property should be used. Must be one of CommandType.**Text**, CommandType.**StoredProcedure**, CommandType.**Table** values. Default is CommandType.**Text**.

See Also

CommandType (MSDN)

CommandText (see page 1919)

15.1.2.1.5.4.4 DataStoreCommand.Connection Property

Gets or sets the DataStoreConnection (see page 1923) used by this command.

Pascal

```
property Connection: DataStoreConnection;
```

C++

```
__property DataStoreConnection Connection;
```

Description

Connection gets or sets the DataStoreConnection (see page 1923) used by this command.

See Also

Connection (MSDN)

DataStoreConnection (see page 1923)

15.1.2.1.5.4.5 DataStoreCommand.DbConnection Property

Pascal

```
property DbConnection: DbConnection;
```

C++

```
__property DbConnection DbConnection;
```

Description

This is DbConnection, a member of class DataStoreCommand.

15.1.2.1.5.4.6 DataStoreCommand.DbParameterCollection Property

Pascal

```
property DbParameterCollection: DbParameterCollection;
```

C++

```
__property DbParameterCollection DbParameterCollection;
```

Description

This is DbParameterCollection, a member of class DataStoreCommand.

15.1.2.1.5.4.7 DataStoreCommand.DbTransaction Property

Pascal

```
property DbTransaction: DbTransaction;
```

C++

```
__property DbTransaction DbTransaction;
```

Description

This is DbTransaction, a member of class DataStoreCommand.

15.1.2.1.5.4.8 DataStoreCommand.DesignTimeVisible Property

Command object should be visible in customized interface control.

Pascal

```
property DesignTimeVisible: bool;
```

C++

```
__property bool DesignTimeVisible;
```

Description

DesignTimeVisible gets or sets a value indicating whether the command object should be visible in a customized interface control.

See Also

DesignTimeVisible (MSDN)

15.1.2.1.5.4.9 DataStoreCommand.Parameters Property

Gets the DataStoreParameterCollection (see page 2041) for this command.

Pascal

```
property Parameters: DataStoreParameterCollection;
```

C++

```
__property DataStoreParameterCollection Parameters;
```

Description

Parameters gets the DataStoreParameterCollection (see page 2041) for this command. See CommandText (see page 1919) property for information on how parameter markers are specified.

See Also

Parameters (MSDN)

DataStoreParameterCollection (see page 2041)

CommandText (see page 1919)

15.1.2.1.5.4.10 DataStoreCommand.RecordsAffected Property

Pascal

```
property RecordsAffected: Integer;
```

C++

```
__property int RecordsAffected;
```

Description

This is RecordsAffected, a member of class DataStoreCommand.

15.1.2.1.5.4.11 DataStoreCommand.Transaction Property

Get transaction of command.

Pascal

```
property Transaction: DataStoreTransaction;
```

C++

```
__property DataStoreTransaction Transaction;
```

Description

Transactiongets the transaction of the command.

See Also

Transaction (MSDN)

15.1.2.1.5.4.12 DataStoreCommand.UpdatedRowSource Property

How command results are applied to DataRow when used by Update method of DataStoreDataAdapter (see page 1933).

Pascal

```
property UpdatedRowSource: UpdateRowSource;
```

C++

```
__property UpdateRowSource UpdatedRowSource;
```

Description

UpdatedRowSource gets or sets how command results are applied to the DataRow when used by the Update method of the DataStoreDataAdapter (see page 1933)

See Also

UpdatedRowSource (MSDN)

Update (MSDN)

DataStoreDataAdapter (see page 1933)

15.1.2.1.6 DataStoreConnection Class

Provides a connection to a Blackfish SQL database.

Class Hierarchy

Pascal

```
DataStoreConnection = class(DbConnection, IDbConnection, ICloneable, IDisposable);
```

C++

```
class DataStoreConnection : public DbConnection, public IDbConnection, public ICloneable,
public IDisposable;
```

File

Ado.Net20.cs

Description

DataStoreConnection provides a connection to a Blackfish SQL database.

DataStoreConnection is an implementation of the .NET DbConnection class.

Each connection provides a separate transactional context. The following example creates an in-process connection:

```
DataStoreConnection con = new DataStoreConnection();
con.ConnectionString = "database=c:\\tmp\\x.jds;user=USERNAME;password=PASSWORD";
con.Open();
```

It is possible to create the database by including `create=true` in the ConnectionString (see page 1930).

See Also

DbConnection (MSDN)

ConnectionString (see page 1930)

15.1.2.1.6.1 DataStoreConnection Members

The following tables list the members exposed by DataStoreConnection.

Internal Methods

	Name	Description
	EndTransaction (see page 1929)	This is EndTransaction, a member of class DataStoreConnection.
	FailIfNotOpen (see page 1929)	This is FailIfNotOpen, a member of class DataStoreConnection.

Internal Properties

	Name	Description
	InternalState (see page 1932)	This is InternalState, a member of class DataStoreConnection.
	MetaDataGenerator (see page 1932)	This is MetaDataGenerator, a member of class DataStoreConnection.

Protected Methods

	Name	Description
	BeginDbTransaction (see page 1926)	This is BeginDbTransaction, a member of class DataStoreConnection.
	CreateDbCommand (see page 1928)	This is CreateDbCommand, a member of class DataStoreConnection.

Public Methods

	Name	Description
	BeginTransaction (see page 1926)	This is the overview for the BeginTransaction method overload.
	ChangeDatabase (see page 1927)	Changes current database for open connection.
	Clone (see page 1927)	Creates a new object that is a copy of the current instance.
	Close (see page 1927)	Closes the connection to the database.
	CreateCommand (see page 1928)	Creates and returns a DataStoreCommand (see page 1912) that uses current connection.
	DataStoreConnection (see page 1925)	This is the overview for the DataStoreConnection constructor overload.
	Dispose (see page 1928)	This is the overview for the Dispose method overload.
	GetSchema (see page 1929)	This is the overview for the GetSchema method overload.
	Open (see page 1930)	Opens a database connection with the property settings specified by the ConnectionString (see page 1930).

Public Properties

	Name	Description
	ConnectionString (see page 1930)	String used to open a Blackfish SQL database.
	ConnectionTimeout (see page 1931)	Wait time before while establishing connection before terminating attempt.
	Database (see page 1931)	Database name of connection.
	DataSource (see page 1931)	Database (see page 1931) server name of the connection.
	ServerVersion (see page 1932)	Version of server if connected.
	State (see page 1932)	Current state of connection.

15.1.2.1.6.2 DataStoreConnection Constructor

This is the overview for the DataStoreConnection constructor overload.

Overload List

	Name	Description
	DataStoreConnection.DataStoreConnection () (see page 1925)	Initializes a new instance of the DataStoreConnection class.
	DataStoreConnection.DataStoreConnection (string) (see page 1925)	Initializes a new instance of the DataStoreConnection class.

15.1.2.1.6.2.1 DataStoreConnection.DataStoreConnection Constructor ()

Initializes a new instance of the DataStoreConnection class.

Pascal

```
constructor Create;
constructor Create(initialConnectionString: string);
```

C++

```
DataStoreConnection();
DataStoreConnection(string initialConnectionString);
```

Description

DataStoreConnection initializes a new instance of the DataStoreConnection class, which is an implementation of the DbConnection class.

When a new instance is created, the connection properties should be set with the ConnectionString (see page 1930) property.

One form of this constructor allows you to specify an initial ConnectionString (see page 1930).

See Also

DbConnection (MSDN)

ConnectionString (see page 1930)

15.1.2.1.6.3 DataStoreConnection Methods

The methods of the DataStoreConnection class are listed here.

Internal Methods

	Name	Description
	EndTransaction (see page 1929)	This is EndTransaction, a member of class DataStoreConnection.
	FailIfNotOpen (see page 1929)	This is FailIfNotOpen, a member of class DataStoreConnection.

Protected Methods

	Name	Description
	BeginDbTransaction (see page 1926)	This is BeginDbTransaction, a member of class DataStoreConnection.
	CreateDbCommand (see page 1928)	This is CreateDbCommand, a member of class DataStoreConnection.

Public Methods

	Name	Description
	BeginTransaction (see page 1926)	This is the overview for the BeginTransaction method overload.
	ChangeDatabase (see page 1927)	Changes current database for open connection.
	Clone (see page 1927)	Creates a new object that is a copy of the current instance.
	Close (see page 1927)	Closes the connection to the database.
	CreateCommand (see page 1928)	Creates and returns a DataStoreCommand (see page 1912) that uses current connection.
	Dispose (see page 1928)	This is the overview for the Dispose method overload.
	GetSchema (see page 1929)	This is the overview for the GetSchema method overload.
	Open (see page 1930)	Opens a database connection with the property settings specified by the ConnectionString (see page 1930).

15.1.2.1.6.3.1 DataStoreConnection.BeginDbTransaction Method

Pascal

```
function BeginDbTransaction(isolationLevel: IsolationLevel): DbTransaction;
```

C++

```
DbTransaction BeginDbTransaction(IsolationLevel isolationLevel);
```

Description

This is BeginDbTransaction, a member of class DataStoreConnection.

15.1.2.1.6.3.2 BeginTransaction Method

This is the overview for the BeginTransaction method overload.

Overload List

	Name	Description
	DataStoreConnection.BeginTransaction () (see page 1926)	Begins database transaction.
	DataStoreConnection.BeginTransaction (IsolationLevel) (see page 1926)	Begins database transaction.

15.1.2.1.6.3.2.1 DataStoreConnection.BeginTransaction Method ()

Begins database transaction.

Pascal

```
function BeginTransaction: DataStoreTransaction;
function BeginTransaction(level: IsolationLevel): DataStoreTransaction;
```

C++

```
DataStoreTransaction BeginTransaction();
DataStoreTransaction BeginTransaction(IsolationLevel level);
```

Description

BeginTransaction begins a database transaction. By default, the connection operates in auto-commit mode. After a transaction is

started, all changes made are pending until Commit is called on the transaction object. After the commit, the connection operates in auto-commit mode again.

Returns an DataStoreTransaction (see page 2056) instance representing the transaction.

The transaction isolation level is set to the passed level for the remainder of this transaction.

See Also

BeginTransaction (MSDN)

DataStoreTransaction (see page 2056)

Commit (see page 2057)

15.1.2.1.6.3.3 DataStoreConnection.ChangeDatabase Method

Changes current database for open connection.

Pascal

```
function ChangeDatabase(database: string): void;
```

C++

```
void ChangeDatabase(string database);
```

Description

ChangeDatabase changes the current database for an open connection.

See Also

ChangeDatabase (MSDN)

15.1.2.1.6.3.4 DataStoreConnection.Clone Method

Creates a new object that is a copy of the current instance.

Pascal

```
function Clone: DataStoreConnection;
```

C++

```
DataStoreConnection Clone();
```

Description

Clone creates a new object that is a copy of the current instance.

See Also

Clone (MSDN)

15.1.2.1.6.3.5 DataStoreConnection.Close Method

Closes the connection to the database.

Pascal

```
function Close: void;
```

C++

```
void Close();
```

Description

This method closes the connection to the database.

See Also

Close (MSDN)

15.1.2.1.6.3.6 DataStoreConnection.CreateCommand Method

Creates and returns a DataStoreCommand (see page 1912) that uses current connection.

Pascal

```
function CreateCommand: DataStoreCommand;
```

C++

```
DataStoreCommand CreateCommand();
```

Description

CreateCommand creates and returns a DataStoreCommand (see page 1912) that uses the current connection.

See Also

CreateCommand (MSDN)

15.1.2.1.6.3.7 DataStoreConnection.CreateDbCommand Method

Pascal

```
function CreateDbCommand: DbCommand;
```

C++

```
DbCommand CreateDbCommand();
```

Description

This is CreateDbCommand, a member of class DataStoreConnection.

15.1.2.1.6.3.8 Dispose Method

This is the overview for the Dispose method overload.

Overload List

	Name	Description
	DataStoreConnection.Dispose () (see page 1928)	Releases resources used by object.
	DataStoreConnection.Dispose (bool) (see page 1928)	Releases resources used by object.

15.1.2.1.6.3.8.1 DataStoreConnection.Dispose Method ()

Releases resources used by object.

Pascal

```
function Dispose: void;
function Dispose(disposing: bool): void;
```

C++

```
void Dispose();
void Dispose(bool disposing);
```

Description

Dispose releases the resources used by the object.

See Also

Dispose (MSDN)

15.1.2.1.6.3.9 DataStoreConnection.EndTransaction Method

Pascal

```
function EndTransaction(outerMost: bool; commitWork: bool): void;
```

C++

```
void EndTransaction(bool outerMost, bool commitWork);
```

Description

This is EndTransaction, a member of class DataStoreConnection.

15.1.2.1.6.3.10 DataStoreConnection.FailIfNotOpen Method

Pascal

```
function FailIfNotOpen: void;
```

C++

```
void FailIfNotOpen();
```

Description

This is FailIfNotOpen, a member of class DataStoreConnection.

15.1.2.1.6.3.11 GetSchema Method

This is the overview for the GetSchema method overload.

Overload List

	Name	Description
	DataStoreConnection.GetSchema () (see page 1929)	Returns schema information for data source of DataStoreConnection (see page 1923).
	DataStoreConnection.GetSchema (string) (see page 1929)	Returns schema information for data source of DataStoreConnection (see page 1923).
	DataStoreConnection.GetSchema (string, string[]) (see page 1929)	Returns schema information for data source of DataStoreConnection (see page 1923).

15.1.2.1.6.3.11.1 DataStoreConnection.GetSchema Method ()

Returns schema information for data source of DataStoreConnection (see page 1923).

Pascal

```
function GetSchema: DataTable;
function GetSchema(collectionName: string): DataTable;
function GetSchema(collectionName: string; restrictionValues: string[]): DataTable;
```

C++

```
DataTable GetSchema();
DataTable GetSchema(string collectionName);
```

```
DataTable GetSchema(string collectionName, string[] restrictionValues);
```

Description

GetSchema returns schema information for the data source of DataStoreConnection (see page 1923).

See Also

GetSchema (MSDN)

15.1.2.1.6.3.12 DataStoreConnection.Open Method

Opens a database connection with the property settings specified by the ConnectionString (see page 1930).

Pascal

```
function Open: void;
```

C++

```
void Open();
```

Description

This method opens a database connection with the property settings specified by the ConnectionString (see page 1930).

See Also

Open (MSDN)

15.1.2.1.6.4 DataStoreConnection Properties

The properties of the DataStoreConnection class are listed here.

Internal Properties

	Name	Description
	InternalState (see page 1932)	This is InternalState, a member of class DataStoreConnection.
	MetaDataGenerator (see page 1932)	This is MetaDataGenerator, a member of class DataStoreConnection.

Public Properties

	Name	Description
	ConnectionString (see page 1930)	String used to open a Blackfish SQL database.
	ConnectionTimeout (see page 1931)	Wait time before while establishing connection before terminating attempt.
	Database (see page 1931)	Database name of connection.
	DataSource (see page 1931)	Database (see page 1931) server name of the connection.
	ServerVersion (see page 1932)	Version of server if connected.
	State (see page 1932)	Current state of connection.

15.1.2.1.6.4.1 DataStoreConnection.ConnectionString Property

String used to open a Blackfish SQL database.

Pascal

```
property ConnectionString: string;
```

C++

```
__property string ConnectionString;
```

Description

ConnectionString gets or sets the string used to open a Blackfish SQL database. It specifies which database to connect to and miscellaneous other connection settings. Here is a typical connection string:

```
"database=c:\\db\\hr.jds;user=USER;password=PASSWORD"
```

By default, the database is started in-process with your client code. If the database is started out of process, include the following options in the connection string: `protocol=TCP;host=MYSERVERNAME`. The "protocol" option is only valid for the Borland.Ds.Server assembly. The assembly Borland.DS.Driver can only connect out of process. The assembly Borland.DS.Mobile can only connect in-process. For a complete list of connection options, see **ConnectionProperties (see page 1897)**.

See Also

ConnectionString (MSDN)

15.1.2.1.6.4.2 DataStoreConnection.ConnectionTimeout Property

Wait time before while establishing connection before terminating attempt.

Pascal

```
property ConnectionTimeout: Integer;
```

C++

```
__property int ConnectionTimeout;
```

Description

ConnectionTimeout gets or sets the wait time while trying to establish a connection before terminating the attempt and generating an error.

See Also

ConnectionTimeout (MSDN)

15.1.2.1.6.4.3 DataStoreConnection.Database Property

Database name of connection.

Pascal

```
property Database: string;
```

C++

```
__property string Database;
```

Description

Database gets the database name of the connection. Use ConnectionString (see page 1930) to set this value.

See Also

Database (MSDN)

ConnectionString (see page 1930)

15.1.2.1.6.4.4 DataStoreConnection.DataSource Property

Database (see page 1931) server name of the connection.

Pascal

```
property DataSource: string;
```

C++

```
__property string DataSource;
```

Description

DataSource gets the database server name of the connection. Use ConnectionString (see page 1930) to set this value.

See Also

DataSource (MSDN)

ConnectionString (see page 1930)

15.1.2.1.6.4.5 DataStoreConnection.InternalState Property

Pascal

```
property InternalState: ConnectionState;
```

C++

```
__property ConnectionState InternalState;
```

Description

This is InternalState, a member of class DataStoreConnection.

15.1.2.1.6.4.6 DataStoreConnection.MetaDataGenerator Property

Pascal

```
property MetaDataGenerator: MetaDataGenerator;
```

C++

```
__property MetaDataGenerator MetaDataGenerator;
```

Description

This is MetaDataGenerator, a member of class DataStoreConnection.

15.1.2.1.6.4.7 DataStoreConnection.ServerVersion Property

Version of server if connected.

Pascal

```
property ServerVersion: string;
```

C++

```
__property string ServerVersion;
```

Description

ServerVersion gets the version of the server if connected. Use ConnectionString (see page 1930) to set this value.

See Also

ServerVersion (MSDN)

ConnectionString (see page 1930)

15.1.2.1.6.4.8 DataStoreConnection.State Property

Current state of connection.

Pascal

```
property State: ConnectionState;
```

C++

```
__property ConnectionState State;
```

Description

State gets the current state of the connection.

See Also

State (MSDN)

15.1.2.1.7 DataStoreDataAdapter Class

Fills a DataSet (see page 1937) and updates a Blackfish SQL database.

Class Hierarchy

Pascal

```
DataStoreDataAdapter = class(DbDataAdapter, IDbDataAdapter, ICloneable, ISupportInitialize);
```

C++

```
class DataStoreDataAdapter : public DbDataAdapter, public IDbDataAdapter, public ICloneable,
public ISupportInitialize;
```

File

Ado.Net20.cs

Description

DataStoreDataAdapter specifies an object that can be used to fill a DataSet (see page 1937) and update a Blackfish SQL database. It contains a DataStoreConnection (see page 1923) instance and up to 4 DataStoreCommand (see page 1912) instances.

DataStoreCommand (see page 1912) is an implementation of the .NET DbDataAdapter class.

See Also

DbDataAdapter (MSDN)

DataStoreCommand (see page 1912)

DataStoreConnection (see page 1923)

15.1.2.1.7.1 DataStoreDataAdapter Members

The following tables list the members exposed by DataStoreDataAdapter.

Protected Methods

	Name	Description
	CreateRowUpdatedEvent (see page 1935)	This is CreateRowUpdatedEvent, a member of class DataStoreDataAdapter.
	CreateRowUpdatingEvent (see page 1936)	This is CreateRowUpdatingEvent, a member of class DataStoreDataAdapter.
	OnRowUpdated (see page 1936)	This is OnRowUpdated, a member of class DataStoreDataAdapter.
	OnRowUpdating (see page 1936)	This is OnRowUpdating, a member of class DataStoreDataAdapter.

Public Events

	Name	Description
	RowUpdated (see page 1939)	Occurs during Update after a command is executed against data source.
	RowUpdating (see page 1939)	Occurs during Update before a command is executed against the data source.

Public Methods

	Name	Description
	BeginInit (see page 1935)	Processing that occurs before initialization.
	DataStoreDataAdapter (see page 1934)	This is the overview for the DataStoreDataAdapter constructor overload.
	EndInit (see page 1936)	Processing that occurs after initialization.

Public Properties

	Name	Description
	DataSet (see page 1937)	Live Data (see page 1762) for Design???.
	DeleteCommand (see page 1937)	DataStoreCommand (see page 1912) used to delete records from Blackfish SQL database.
	InsertCommand (see page 1937)	DataStoreCommand (see page 1912) used to insert new records into Blackfish SQL database.
	SelectCommand (see page 1938)	DataStoreCommand (see page 1912) used to select records from Blackfish SQL database into a DataSet (see page 1937).
	UpdateCommand (see page 1938)	DataStoreCommand (see page 1912) used to update records from DataSet (see page 1937) into Blackfish SQL database.

15.1.2.1.7.2 DataStoreDataAdapter Constructor

This is the overview for the DataStoreDataAdapter constructor overload.

Overload List

	Name	Description
	DataStoreDataAdapter.DataStoreDataAdapter () (see page 1934)	Initializes a new instance of the DataStoreDataAdapter class.
	DataStoreDataAdapter.DataStoreDataAdapter (DataStoreCommand) (see page 1934)	Initializes a new instance of the DataStoreDataAdapter class.
	DataStoreDataAdapter.DataStoreDataAdapter (string, DataStoreConnection) (see page 1934)	Initializes a new instance of the DataStoreDataAdapter class.
	DataStoreDataAdapter.DataStoreDataAdapter (string, string) (see page 1934)	Initializes a new instance of the DataStoreDataAdapter class.

15.1.2.1.7.2.1 DataStoreDataAdapter.DataStoreDataAdapter Constructor ()

Initializes a new instance of the DataStoreDataAdapter class.

Pascal

```
constructor Create;
constructor Create(selectCommand: DataStoreCommand);
constructor Create(selectQuery: string; connectionString: string);
constructor Create(selectQuery: string; connection: DataStoreConnection);
```

C++

```
DataStoreDataAdapter();
DataStoreDataAdapter(DataStoreCommand selectCommand);
DataStoreDataAdapter(string selectQuery, string connectionString);
DataStoreDataAdapter(string selectQuery, DataStoreConnection connection);
```

Description

DataStoreDataAdapter initializes a new instance of the DataStoreDataAdapter class, which is an implementation of the DbDataAdapter class.

See Also

DbDataAdapter (MSDN)

15.1.2.1.7.3 DataStoreDataAdapter Methods

The methods of the DataStoreDataAdapter class are listed here.

Protected Methods

	Name	Description
	CreateRowUpdatedEvent (see page 1935)	This is CreateRowUpdatedEvent, a member of class DataStoreDataAdapter.
	CreateRowUpdatingEvent (see page 1936)	This is CreateRowUpdatingEvent, a member of class DataStoreDataAdapter.
	OnRowUpdated (see page 1936)	This is OnRowUpdated, a member of class DataStoreDataAdapter.
	OnRowUpdating (see page 1936)	This is OnRowUpdating, a member of class DataStoreDataAdapter.

Public Methods

	Name	Description
	BeginInit (see page 1935)	Processing that occurs before initialization.
	EndInit (see page 1936)	Processing that occurs after initialization.

15.1.2.1.7.3.1 DataStoreDataAdapter.BeginInit Method

Processing that occurs before initialization.

Pascal

```
function BeginInit: void;
```

C++

```
void BeginInit();
```

Description

BeginInit specifies any processing that occurs before initialization. Enter any code you wish to execute prior to initialization of the DataStoreDataAdapter (see page 1933).

See Also

EndInit (see page 1936)

15.1.2.1.7.3.2 DataStoreDataAdapter.CreateRowUpdatedEvent Method

Pascal

```
function CreateRowUpdatedEvent(dataRow: DataRow; command: IDbCommand; statementType:
StatementType; tableMapping: DataTableMapping): RowUpdatedEventArgs;
```

C++

```
RowUpdatedEventArgs CreateRowUpdatedEvent(DataRow dataRow, IDbCommand command, StatementType
statementType, DataTableMapping tableMapping);
```

Description

This is CreateRowUpdatedEvent, a member of class DataStoreDataAdapter.

15.1.2.1.7.3.3 DataStoreDataAdapter.CreateRowUpdatingEvent Method

Pascal

```
function CreateRowUpdatingEvent(dataRow: DataRow; command: IDbCommand; statementType:
StatementType; tableMapping: DataTableMapping): RowUpdatingEventArgs;
```

C++

```
RowUpdatingEventArgs CreateRowUpdatingEvent(DataRow dataRow, IDbCommand command, StatementType
statementType, DataTableMapping tableMapping);
```

Description

This is CreateRowUpdatingEvent, a member of class DataStoreDataAdapter.

15.1.2.1.7.3.4 DataStoreDataAdapter.EndInit Method

Processing that occurs after initialization.

Pascal

```
function EndInit: void;
```

C++

```
void EndInit();
```

Description

BeginInit (see page 1935) specifies any processing that occurs after initialization. Enter any code you wish to execute after initialization of the DataStoreDataAdapter (see page 1933).

See Also

BeginInit (see page 1935)

15.1.2.1.7.3.5 DataStoreDataAdapter.OnRowUpdated Method

Pascal

```
function OnRowUpdated(value: RowUpdatedEventArgs): void;
```

C++

```
void OnRowUpdated(RowUpdatedEventArgs value);
```

Description

This is OnRowUpdated, a member of class DataStoreDataAdapter.

15.1.2.1.7.3.6 DataStoreDataAdapter.OnRowUpdating Method

Pascal

```
function OnRowUpdating(value: RowUpdatingEventArgs): void;
```

C++

```
void OnRowUpdating(RowUpdatingEventArgs value);
```

Description

This is OnRowUpdating, a member of class DataStoreDataAdapter.

15.1.2.1.7.4 DataStoreDataAdapter Properties

The properties of the DataStoreDataAdapter class are listed here.

Public Properties

	Name	Description
	DataSet (see page 1937)	Live Data (see page 1762) for Design???.
	DeleteCommand (see page 1937)	DataStoreCommand (see page 1912) used to delete records from Blackfish SQL database.
	InsertCommand (see page 1937)	DataStoreCommand (see page 1912) used to insert new records into Blackfish SQL database.
	SelectCommand (see page 1938)	DataStoreCommand (see page 1912) used to select records from Blackfish SQL database into a DataSet (see page 1937).
	UpdateCommand (see page 1938)	DataStoreCommand (see page 1912) used to update records from DataSet (see page 1937) into Blackfish SQL database.

15.1.2.1.7.4.1 DataStoreDataAdapter.DataSet Property

Live Data (see page 1762) for Design???.

Pascal

```
property DataSet: DataSet;
```

C++

```
__property DataSet DataSet;
```

Description

Live Data (see page 1762) for Design:.

15.1.2.1.7.4.2 DataStoreDataAdapter.DeleteCommand Property

DataStoreCommand (see page 1912) used to delete records from Blackfish SQL database.

Pascal

```
property DeleteCommand: DataStoreCommand;
```

C++

```
__property DataStoreCommand DeleteCommand;
```

Description

DeleteCommand gets or sets the DataStoreCommand (see page 1912) used to delete records identified by a DataSet (see page 1937) from a Blackfish SQL database.

See Also

DeleteCommand (MSDN)

InsertCommand (see page 1937)

SelectCommand (see page 1938)

UpdateCommand (see page 1938)

15.1.2.1.7.4.3 DataStoreDataAdapter.InsertCommand Property

DataStoreCommand (see page 1912) used to insert new records into Blackfish SQL database.

Pascal

```
property InsertCommand: DataStoreCommand;
```

C++

```
__property DataStoreCommand InsertCommand;
```

Description

InsertCommand gets or sets the DataStoreCommand (see page 1912) used to insert new records from a DataSet (see page 1937) into a Blackfish SQL database.

See Also

InsertCommand (MSDN)

DeleteCommand (see page 1937)

SelectCommand (see page 1938)

UpdateCommand (see page 1938)

15.1.2.1.7.4.4 DataStoreDataAdapter.SelectCommand Property

DataStoreCommand (see page 1912) used to select records from Blackfish SQL database into a DataSet (see page 1937).

Pascal

```
property SelectCommand: DataStoreCommand;
```

C++

```
__property DataStoreCommand SelectCommand;
```

Description

SelectCommand gets or sets the DataStoreCommand (see page 1912) used to select records from a Blackfish SQL database into a DataSet (see page 1937).

See Also

SelectCommand (MSDN)

DeleteCommand (see page 1937)

InsertCommand (see page 1937)

UpdateCommand (see page 1938)

15.1.2.1.7.4.5 DataStoreDataAdapter.UpdateCommand Property

DataStoreCommand (see page 1912) used to update records from DataSet (see page 1937) into Blackfish SQL database.

Pascal

```
property UpdateCommand: DataStoreCommand;
```

C++

```
__property DataStoreCommand UpdateCommand;
```

Description

UpdateCommand gets or sets the DataStoreCommand (see page 1912) used to update records from a DataSet (see page 1937) into a Blackfish SQL database.

See Also

UpdateCommand (MSDN)

DeleteCommand (see page 1937)

InsertCommand (see page 1937)

SelectCommand (see page 1938)

15.1.2.1.7.5 DataStoreDataAdapter Events

The events of the DataStoreDataAdapter class are listed here.

Public Events

	Name	Description
	RowUpdated (see page 1939)	Occurs during Update after a command is executed against data source.
	RowUpdating (see page 1939)	Occurs during Update before a command is executed against the data source.

15.1.2.1.7.5.1 DataStoreDataAdapter.RowUpdated Event

Occurs during Update after a command is executed against data source.

Pascal

```
property RowUpdated: DataStoreRowUpdatedEventHandler;
```

C++

```
__property DataStoreRowUpdatedEventHandler RowUpdated;
```

Description

RowUpdated occurs during Update after a command is executed against the data source. The attempt to update is made, so the event fires.

See Also

Update (MSDN)

15.1.2.1.7.5.2 DataStoreDataAdapter.RowUpdating Event

Occurs during Update before a command is executed against the data source.

Pascal

```
property RowUpdating: DataStoreRowUpdatingEventHandler;
```

C++

```
__property DataStoreRowUpdatingEventHandler RowUpdating;
```

Description

RowUpdating occurs during Update before a command is executed against the data source. The attempt to update is made, so the event fires.

See Also

Update (MSDN)

15.1.2.1.8 DataStoreErrorCode Class

Symbolic error codes for Blackfish SQL database.

Class Hierarchy

Borland.Data.DataStore.DataStoreErrorCode

Pascal

```
DataStoreErrorCode = class;
```

C++

```
class DataStoreErrorCode;
```

File

DataStoreErrorCode.cs

Description

DataStoreErrorCode provides symbolic error codes for the Blackfish SQL database.

This table contains error codes.

Constant	Description
GenericError (see page 1987)	Undefined error.
NoUpdatableColumns (see page 2013)	Could not find any updatable columns when saving dataset data.
UnknownColumnName (see page 2029)	Cannot find column with given column name.
ColumnTypeConflict (see page 1975)	Attempt to add column with same name, but different data type as existing **DataColumn**.
InvalidColumnPosition (see page 1993)	Column position is out of range.
ColumnAlreadyBound (see page 1974)	Column already bound to a DataSet. Use `(Column)Column.clone()` to add this column.
CannotChangeColumnDataType (see page 1971)	Cannot change column data type because associated column already has data.
UnrecognizedDataType (see page 2030)	Unknown data type.
DatasetHasNoTables (see page 1979)	Cannot be saved. No updatable columns have a table name.
PartialSearchForString (see page 2016)	Partial search option can only be used when last column searched on is of **String** type".
NeedLocateStartOption (see page 2006)	Must specify one of **First**, **Last**, **Next** or **Prior Locate** for locate operations.
LinkColumnsError (see page 2000)	The number and data type of columns in the DataSet **MasterLinkColumns** and **DetailLinkColumns** must match.
DatasetCorrupt (see page 1978)	Operation failed. DataSet structurally corrupt.
DuplicateColumnName (see page 1984)	Duplicate column name.
NotUpdateable (see page 2013)	DataSet has no unique row identifiers and is not updatable.
DatasetOpen (see page 1979)	Operation cannot be performed on an open DataSet. Close the DataSet first.
DatasetNotOpen (see page 1979)	Operation failed. DataSet is not open.
ExceptionChain (see page 1985)	Chained exception. To access the lower level exception(s) call `getExceptionChain`. Also, `printStackTrace` displays all exceptions in the chain.
InvalidSortColumn (see page 1997)	Operation failed. Attempt to sort on non sortable column, i.e. BinaryColumn.

NullColumnName (see page 2014)	Trying to perform an operation that requires a column name without having specified one.
ResolveInProgress (see page 2020)	DataSet changes are currently being saved. Retry operation later.
InvalidColumnFormat (see page 1992)	Column formatting error for default, minimum, maximum values using edit mask or display format.
OnepassInputStream (see page 2015)	Attempting to read twice from a java.io.InputStream implementation that throws an exception when `InputStream.reset` is called and `InputStream.available` returns 0 bytes. To correct this, use an InputStream implementation that supports `reset`.
InsufficientRowid (see page 1990)	`DataSet.refetchRow` was called with insufficient row id.
NonExistentRowid (see page 2009)	`DataSet.refetchRow` was called with non existent row id
RefreshrowNotSupported (see page 2020)	Current DataSet does not refetchRow
IoError (see page 1999)	Operation failed due to an IO error.
FieldPostError (see page 1986)	Occurs when a DataSet requests that a data aware control post field values being edited and the field value cannot be posted (most likely due to a validation error). Unfortunately controls do not currently provide any information on why a post failed.
ClassNotFoundError (see page 1974)	Class could not be found during deserialization process.
ReadOnlyStore (see page 2019)	Store property setting is read-only.
ProcedureFailed (see page 2017)	Execution of procedure failed.
InvalidStoreName (see page 1997)	The **storeName** (storage name) property must be one or more characters in length.
NeedsRecalc (see page 2007)	Prevents write access if **StorageDataSet.CalcListener** or **StorageDataSet.AggCalcListener** have not been set on a **StorageDataSet** component. Should only occur with **StorageDataSets** that use DataStore (see page 1892) for a backing store. DataStore (see page 1892) persists the class names of calc listeners. Thus if a **StorageDataSet** component opens without specifying a **StorageDataSet.CalcListener** or **StorageDataSet.AggCalcListener** that were used previously, the **StorageDataSet** is marked read-only. To correct, 1. add the appropriate listeners, or 2. call **StorageDataSet.recalc**
DuplicatePrimary (see page 1984)	Attempt to specify different primary key when one already exists.
NoPrimaryKey (see page 2010)	Attempt to specify primary Sort without specifying any sortKey columns.
InvalidSortAsInserted (see page 1997)	Sort as inserted not supported with descending sort orders.
InvalidSort (see page 1997)	No keys specified to create index for **SortDescriptor**
TooManyAutoIncrements (see page 2026)	There can only be one autoincrement column in a table.

InvalidConstraintName (see page 1993)	This constraint name does not exist.
ReadOnlyColumn (see page 2018)	Attempting to assign a value to a read-only column.
ReadOnlyDataset (see page 2019)	Attempting to assign a value to a read-only table.
LessThanMin (see page 1999)	Value entered for column is less than the column **min** property.
GreaterThanMax (see page 1987)	Value entered for column is greater than the column **max** property.
InvalidPrecision (see page 1996)	String values for this column cannot exceed the precision length set in the **Column.precision** property. This property can be implicitly set when data is retrieved from a provider. If data is being provided from a JDBC driver, the precision specified by the JDBC result set is propagated to the **Column.precision** property. This propagation can be overridden by explicitly setting the **Column.precision** property.
InvalidFormat (see page 1994)	Cannot parse value or generic format error.
NoRowsToDelete (see page 2011)	No rows to delete.
InsertNotAllowed (see page 1990)	Row insertion not allowed.
UpdateNotAllowed (see page 2031)	Row editing not allowed.
DeleteNotAllowed (see page 1982)	Row deleting not allowed.
DuplicateKey (see page 1984)	The key value is a duplicate.
ForeignKeyViolation (see page 1987)	Violation of foreign key constraint has occurred.
ForeignKeyError (see page 1986)	Violation of foreign key constraint definition error.
ConstraintNameExists (see page 1977)	Could not add constraint because a unique or foreign key constraint already exists with that name.
StoreNameNotSet (see page 2024)	Operation failed. **StoreName** property for Table component not set.
InternalDuplicateKey (see page 1990)	Operation failed. Attempt to add duplicate key value. This is different than ValidationException.DUPLICATE_KEY error. This error is an internal unexpected key violation, not unique constraint.
RowTooWide (see page 2023)	Max row size exceeded. Too many columns and/or high **Column.MaxInline** properties used. There are two ways to remedy this problem: 1) set **Column.MaxInline** property for columns with data types of **Variant.STRING**, **Variant.INPUTSTREAM** or **Variant.OBJECT** to a value less than 16. By default, the maximum inline storage for such fields does not go below 16 unless the **Column.MaxInline** property is set lower. 2) Create your DataStore (see page 1892) with a larger block size by setting the **DataStore.BlockSize** property.
DatasetAlreadyOpen (see page 1978)	Attempt to open table with a StorageDataSet that has a different set of columns.
CannotRestructure (see page 1972)	Cannot restructure when table is opened by more than one StorageDataSet.

ColumnNeedsRestructure (see page 1975)	Column is new or its data type has changed. Restructure the DataSet with `DataSet.setRestructure(false);`
DatastoreNotOpen (see page 1981)	Operation failed. DataStore (see page 1892) not open.
DatastoreNotFound (see page 1980)	DataStore (see page 1892) not found by **DataStore.storeExists** method or **DataStoreConnection.FileName** property is null.
DatastoreInvalid (see page 1980)	File specified by **DataStore.FileName** property does not appear to be a valid DataStore (see page 1892) file. File signature or size is invalid.
InternalUpdateFailed (see page 1991)	Previous update failed. Shut down and reopen the DataStore (see page 1892).
StreamOpenTwice (see page 2025)	Unexpected internal condition. call **DataStore.shutDown** and then **DataStore.open**.
NameNotUnique (see page 2006)	Name of storage file or table being added, undeleted, or renamed is not unique.
StoreOperationUnsupported (see page 2025)	Unexpected internal condition. Operation not supported on this stream.
DatastoreExists (see page 1980)	**ResponseEvent**occurred. If **DataStore.ResponseListener** does not exist or file is not deleted by ResponseListener, an IOException is thrown. This element is deprecated.
DatastoreOpen (see page 1981)	Operation failed. Operation cannot be performed on an opened DataStore (see page 1892).
WriteBlockError (see page 2032)	Error writing to DataStore (see page 1892). Unexpected cache block contents.
ReadBlockError (see page 2018)	Error reading from DataStore (see page 1892). Unexpected block contents.
AggOperatorNotFound (see page 1966)	AggOperator class could not be loaded. The class is probably not in the classpath.
OperationCanceled (see page 2015)	Operation canceled. Used for canceling long running operations such as sorting or restructuring.
RestructureDataLoss (see page 2021)	Restructure operation is converting from one data type to another. Old values are not converted to new data type.
RestructurePrecisionLoss (see page 2021)	Restructure operation is converting from one data type to another. Type conversion may result in precision loss when values of old data type are converted to values of new data type.
RestructureParseError (see page 2021)	Restructure operation is converting from one data type to another. One or more parse errors occurred converting a **String** data type to a non **String** data type.
BigdecimalPrecisionError (see page 1970)	Precision of BigDecimal exceeded.
DatastoreAlreadyOpen (see page 1980)	DataStore (see page 1892) is already opened by another process.
StreamClosed (see page 2025)	InputStream from a DataStore (see page 1892) has been closed and is no longer accessible.
CantCreateOpenStream (see page 1974)	Can't create or delete a stream that is still open.
CantCreateOpenFile (see page 1974)	Use **CANT_CREATE_OPEN_STREAM** This element is deprecated.
StreamNotFound (see page 2025)	Stream not found in DataStore (see page 1892).

FileNotFound (see page 1986)	See **STREAM_NOT_FOUN**D This element is deprecated.
NewerVersion (see page 2008)	DataStore (see page 1892) cannot be opened because it was created by a newer a version of the DataStore (see page 1892) component.
DatasetExists (see page 1979)	Cannot create a file stream with the same name as an existing table stream.
FileExists (see page 1986)	Cannot create a DataSet stream with the same name as an existing file.
RowNotFound (see page 2023)	Unexpected condition. Internal row not found.
TooManyErrors (see page 2026)	Too many errors detected by the stream verifier.
ReadOnly (see page 2018)	Write operation failed. The DataStore (see page 1892) is read-only.
OlderVersion (see page 2014)	Cannot perform operation on DataStore (see page 1892) because it was created with an older version of DataStore (see page 1892). To upgrade the DataStore (see page 1892) file, use `DataStoreConnection.copyStreams`.
InlineTooSmall (see page 1989)	**Column.MaxInline** property setting is too small.
DataHasDuplicates (see page 1978)	An attempt to create a unique index failed, because the table had duplicate row values for the given column combination.
InvalidPattern (see page 1996)	Invalid escape sequence in **String** pattern at character offset. Escape character of '\' can only precede '*', '?' or '\' characters.
IncompatibleBlockSize (see page 1988)	Incompatible **DataStore.BlockSize** property for copyStream operation. If copying between two DataStores, both DataStores must currently have the same **DataStore.BlockSize** property setting.
CannotOpen (see page 1972)	Could not create a lock dir for DataStore (see page 1892) file. Make sure the DataStore (see page 1892) file exists and that you have permission to create a subdirectory.
CannotLockDir (see page 1971)	Same as **CannotOpen (see page 1972)**.
InvalidDirectoryAttributes (see page 1994)	Cannot open DataSet because it has invalid directory attributes.
DatastoreIsUnlicensed (see page 1980)	DataStore (see page 1892) is unlicensed.
ConnectionCountExceeded (see page 1976)	Connection count exceeded for DataStore (see page 1892) license.
SecurityError (see page 2023)	Security error.
InvalidSavepoint (see page 1996)	Invalid Savepoint. Savepoint is out of date or was created for a different connection.
CreateErrorMethodExists (see page 1977)	An attempt was made to create a Java (see page 15681) Method that already exists.
MethodNonexistent (see page 2003)	An attempt was made to drop or reference a Java (see page 15681) Method that doesn't exist.
MethodClassMissing (see page 2002)	The class of a Java (see page 15681) Method is missing from the classpath.
MethodNameMissing (see page 2002)	The class of a Java (see page 15681) Method does not contain any methods matching its definition.
ConflictingMethodDef (see page 1976)	This class of a Java (see page 15681) Method has conflicting definitions for this method.

DropErrorDependentView (see page 1983)	This table or view cannot be dropped because it has a dependent view.
DropErrorForeignKey (see page 1983)	This table cannot be dropped because it appears in a foreign key.
DropSchemaError (see page 1984)	This schema cannot be dropped because it still has tables, views, or methods.
RevokeErrorDependentView (see page 2022)	This privilege or role revoke cannot be performed because a dependent view exists.
RevokeErrorForeignKey (see page 2022)	This privilege or role revoke cannot be performed because a dependent foreign key exists.
RevokeErrorDependentPrivilege (see page 2022)	This privilege or role revoke cannot be performed because a dependent privilege exists.
TableNonexistent (see page 2026)	An attempt was made to reference a table that doesn't exist.
DropErrorRoleGrant (see page 1984)	This role cannot be dropped because it is granted to a user or a role.
MirrorError (see page 2003)	An error occurred in the management of fail safe mirrors.
CannotStartServer (see page 1973)	DataStore (see page 1892) server could not be started. This usually occurs because the port is in use by another process.
NewerVersionForOperation (see page 2008)	The functionality invoked can only be used in database files. Consult the documentation for alternative functionality.
CyclesInViewDefinition (see page 1978)	An **ALTER VIEW** would result in a cyclic view definition where the view depends on it own definition. This is illegal.
DbAdminError (see page 1981)	An Error occurred processing a `DB_ADMIN` stored procedure call.
ReadonlyTimeout (see page 2019)	Unexpected time-out constructing a transactionally consistent read only view of the database.
DesktopLicense (see page 1982)	Desktop license can only connect to localhost or 127.0.0.1 host name.
AfterLast (see page 1966)	The ResultSet is positioned after the last result row. It is an error to access columns in this state.
AggNotValidHere (see page 1966)	A SQL aggregate function such as **COUNT**, **SUM**, **AVG**, **MIN** or **MAX** was found in an improper context. These functions may only appear in certain expressions in a **SELECT** expression: In the select list and in a **HAVING** clause.
AmbiguousType (see page 1967)	The type of a SQL parameter marker cannot be identified by the surrounding operators. Use the `CAST` function to identify the type, eg. `CAST ( ? AS DATE )`.
BadColumnIndex (see page 1967)	A column index is out of range. Note: In JDBC, these indices are 1 based.
BadDirection (see page 1968)	An invalid **FETCH** direction was given.
BadFetchsize (see page 1968)	An invalid **FETCH** size was specified.
BadMaxfieldsize (see page 1968)	An invalid maximum field size was specified. This number must be non negative.
BadMaxrows (see page 1968)	An invalid number for maximum wanted rows in a ResultSet was specified. This number must be non negative.

BadOrdinal (see page 1969)	An invalid ordinal was specified in an **ORDER BY** clause or **GROUP BY** of a **SELECT** statement. `ORDER BY 1` means sort in the order of the first output column etc.
BadUrl (see page 1969)	An invalid connection URL was given for the database.
BeforeFirst (see page 1969)	The ResultSet is positioned before the first result row. It is an error to access columns in this state.
CannotAssignTypes (see page 1970)	An assignment was attempted with a value of the wrong type. This can be a **DEFAULT** value of a column, or an **INSERT** or **UPDATE** statement.
CannotCast (see page 1970)	A cast from one type to another type was impossible.
CannotUseInternalrow (see page 1973)	An INTERNALROW column cannot be set from an **INSERT** or **UPDATE** statement. This column is a pseudo column and is maintained by the database system.
CannotUseViewHere (see page 1973)	A VIEW cannot be altered in an **ALTER** statement. Instead **DROP** the view and redefine it.
ColumnNotInLeftRef (see page 1975)	A column name listed in a **USING** clause of a table **JOIN** does not exist in the left table expression.
ColumnNotInRightRef (see page 1975)	A column name listed in a **USING** clause of a table **JOIN** does not exist in the right table expression.
ColumnSpecifiedTwice (see page 1975)	A column name is specified twice in the column list for the **INSERT** statement, i.e. two values are given for the same column.
ConnectionClosed (see page 1976)	An attempt has been made to use a connection that was closed earlier.
ConnectionError (see page 1976)	A connection error was encountered. Read the message string for details.
DifferentDegree (see page 1982)	The table expression cannot be evaluated because the two tables involved have a different column count.
DifferentSymbol (see page 1982)	The SQL statement could not be parsed. Check the syntax and correct the error. Notice the column information in the error message.
DivideByZero (see page 1983)	A divide by zero occurred during execution of SQL statement.
EofExpected (see page 1985)	The SQL statement could not be parsed. Note only one statement can be can be parsed and executed at a time.
FeatureNotSupp (see page 1985)	This feature is not supported by the current driver.
FloatPrecision (see page 1986)	The maximum **FLOAT** precision supported is 52 bits.
ForwardOnly (see page 1987)	This is a **FORWARD_ONLY** cursor. Therefore **next** is the only valid navigation method for this cursor.
HasResultset (see page 1987)	The **executeUpdate** method was called to execute a SQL query, which produced a ResultSet. Use **executeQuery** instead, or if the nature of the query is unknown, use **execute**.
IllegalCatalog (see page 1988)	Catalogs are not supported.
IllegalCharacter (see page 1988)	This character cannot be part of a SQL syntax unless it appears in quotes.
IllegalExprStart (see page 1988)	Syntax error in SQL. The parser expects an expression, but does not recognize this symbol as the start of an expression.

IncompatibleTypes (see page 1989)	The table expression cannot be evaluated because the two tables involved have a different column type for the column index mentioned in error message.
IncompatibleTypesForOp (see page 1989)	A type mismatch was found for the given operator.
IndexAlreadyExists (see page 1989)	An index with this name has already been created.
InternalError (see page 1991)	An internal error has happened. Please submit the exception and, if possible, a reproducible test case to CodeGear.
InvalidAggExpr (see page 1991)	A SQL aggregate function such as **COUNT**, **SUM**, **AVG**, **MIN** or **MAX** was found with an invalid expression argument. If such an argument expression has multiple column references, those column references must all be part of the same **SELECT** expression. It is an illegal aggregate expression if there is a mix of inner and outer column references.
InvalidBinaryString (see page 1992)	The binary string contains non binary characters. The valid repertoires are: B'01' and X'0123456789abcdef' or X'0123456789ABCDEF'.
InvalidColumnname (see page 1992)	The named column was not found in the current scope.
InvalidColumnRef (see page 1993)	The dot "." separated column syntax was used more than 4 levels deep. That syntax is only recognized for 4 levels: catalog.schema.table.column.
InvalidEscapeCharInLike (see page 1994)	The string for the escape character in a **LIKE** operator evaluated to a string that was not exactly one character long.
InvalidExtractField (see page 1994)	The extract field for the **EXTRACT** function was incorrect. Check the syntax.
InvalidIndexname (see page 1995)	The specified index does not exist on the specified table.
InvalidLikePattern (see page 1995)	The pattern string in a **LIKE** operator did not evaluate to a valid pattern string.
InvalidTableRef (see page 1998)	The dot "." separated column syntax was used more than 3 levels deep. That syntax for a table only recognizes 3 levels: catalog.schema.table.
InvalidTablename (see page 1998)	The named table was not found as a **VIEW** or **TABLE**.
InvalidTrimChar (see page 1998)	The string for the trim character of the **TRIM** function evaluated to a string that was not exactly one character long.
InvalidViewDefinition (see page 1998)	The view was invalidated by changes to a parent table.
InvalidJoin (see page 1995)	An invalid **JOIN** expression was given. A **JOIN** expression must have at least one of the following specifiers: **NATURAL**, **UNION**, **CROSS**, **ON** or **USING**.
IoNetError (see page 1999)	An connection related problem occurred during the execution of this operation.
IoProtocolError (see page 1999)	A protocol error was found during execution of this operation. The connection between client and server may be unstable.
IsInsertRow (see page 1999)	Delete and update operations cannot happen while editing the special driver insert row.
MalformedDateOrTime (see page 2001)	The given date or time string did not represent a date or a time.
MalformedDecimal (see page 2001)	The given decimal value is not properly formatted.
MalformedFloating (see page 2001)	The given floating point value is not properly formatted.

MethodArgMismatch (see page 2002)	A parameter mismatch was found for the stored procedure.
MethodNotFound (see page 2003)	No such method was found in the database.
MismatchInsertColumnCount (see page 2004)	The number of insert values doesn't match the number needed for the **INSERT** statement.
MismatchColumnlistCount (see page 2003)	The number of columns in the column list doesn't match the number of columns in the table expression.
MismatchIntoParamCount (see page 2004)	The number of output parameters doesn't match the number of output columns in this single row **SELECT** query.
MoreThanOneRow (see page 2005)	A scalar subquery resulted in a ResultSet with more than one row.
MustBeSingleColumn (see page 2006)	This subquery resulted in a ResultSet with more than one column, which is not valid in current context.
NegativeSqrtArgument (see page 2007)	The argument to the **SQRT** function is negative.
NegativeSubstringLength (see page 2007)	The length argument to the **SUBSTRING** function is negative
NestedAggregators (see page 2008)	Nested aggregator functions such as **COUNT**, **SUM**, **AVG**, **MIN** or **MAX** is not valid.
NoGroupingColumn (see page 2009)	This column must be in the **GROUP BY** clause or appear inside an aggregate function.
NoReadonly (see page 2010)	The connection should not be opened in read only mode, because it is already open in read-write mode.
NoResultsetProduced (see page 2011)	The query that was executed with **executeQuery** did not yield a ResultSet.
NoServerFound (see page 2011)	A remote connection could not be made because database server was not found.
NoSuchParameter (see page 2012)	An attempt was made to set or get from a parameter that doesn't exist in the query.
NodefaultForBinary (see page 2009)	No DEFAULT value is accepted for binary columns.
NonPrimitiveRequired (see page 2010)	A NULL value was passed for a parameter that does not accept NULL values. Correct the parameter or change the store procedure to accept NULL values by using non primitive data types in Java (see page 15681), i.e. use java.lang.Integer instead of int, and so on.
NonTransactional (see page 2010)	The data store is not transactional, and therefore cannot be opened through a SQL based driver.
NotAnOutputParameter (see page 2012)	An attempt was made to register this parameter as an output parameter, however the parameter does not represent an output parameter in the SQL query.
NotInsertRow (see page 2012)	Inserts in driver can only happen from special driver insert row.
ResultNotUpdateable (see page 2022)	An attempt to update current ResultSet was made. However the query that was run was not an updatable query.
NowReadonly (see page 2014)	The connection should not be opened in read- write mode, because it is already open in read-only mode.
NumericOverflow (see page 2014)	A floating point operation yielded a numeric overflow value.

NumericUnderflow (see page 2014)	A floating point operation yielded a numeric underflow value.
OutParameterRequired (see page 2015)	An output parameter is required at this position, because the stored procedure is an in-out parameter.
ParameterValueNotSet (see page 2016)	This input parameter was not set to a value prior to executing the query.
PrivilegeMix (see page 2016)	Cannot mix system and object privileges.
ProtocolVersion (see page 2017)	An incompatible version of driver client and server was used.
ColumnIsReadonly (see page 1974)	An attempt was made to update a read-only column.
RenamedTwice (see page 2020)	A column was renamed twice in the same **ALTER TABLE** statement.
ResultsetClosed (see page 2022)	An attempt was made to access a closed ResultSet.
SingleRowHasMoreRows (see page 2024)	A single row SELECT statement yielded more than one1 row.
StatementClosed (see page 2024)	An attempt was made to access a closed statement.
StringTooLong (see page 2025)	A string was too large to transport over the wire from the client to the database server.
TableAlreadyExists (see page 2026)	A table or a view already exists with that name.
Toomanyspecs (see page 2027)	An invalid **JOIN** expression was given. A **JOIN** expression cannot have more than one of the following specifiers: **NATURAL**, **UNION**, **CROSS**, **ON** or **USING**.
TypenameExpected (see page 2028)	An unrecognized data type was found in the SQL query.
UnclosedComment (see page 2028)	An unclosed comment was found.
UnclosedIdentifier (see page 2028)	An unclosed quoted identifier was found.
UnclosedStringLiteral (see page 2029)	An unclosed string literal was found.
UnknownColumn (see page 2029)	A column name was not found in the ResultSet.
UnrecognizedEscape (see page 2030)	An unrecognized escape sequence or escape function was found.
ViewNotUpdatable (see page 2032)	An attempt to update a non updatable **VIEW** was made.
WithCheckOptionViolation (see page 2032)	An insert or an update was made to a row in a view, which caused that row to be filtered out of that view.
IllegalMethodCall (see page 1988)	A method call was attempted that caused an IllegalAccessException. Check if the method is public and static and is the class public.
IntegerRequired (see page 1990)	A function that takes an integer argument was given a numeric parameter with significant decimal digits.
OverloadedDiffParams (see page 2015)	A function is overloaded with different parameter names, so it cannot be called using named parameters.

WrongSchemaname (see page 2033)	A create schema must be run by the owner of that schema.
WrongCreateschemaname (see page 2032)	If present, the schema name of an object created in a **CREATE SCHEMA** statement must match that schema.
WrongTableType (see page 2033)	A view operation was attempted on a table, or a table operation was attempted on a view.
SyntaxError (see page 2026)	An error occurred during syntax or semantic check of query.
RuntimeError (see page 2023)	An error happened during run of query.
FailoverError (see page 1985)	An error occurred attempting to fail over to another server.
CannotMixNamedParameters (see page 1972)	The SQL parameters must be accessed by either name or ordinal. Mixing these access methods is not supported.
UnknownParameter (see page 2030)	A reference to an unknown SQL parameter name was made.
ConnectionTimeout (see page 1977)	Connection time-out waiting for an available connection in the connection pool.
ConnectionPoolClosed (see page 1977)	The connection pool was closed while waiting for an available connection.
UnsupportedDbxType (see page 2031)	Unspported dbExpress data type from **TDBXDataTypes** class.
MaxResolveFailed (see page 2002)	If the **StorageDataSet.MaxResolveErrors** property is set, a ResolutionException with this error code is thrown when the maximum errors are encountered. The resolution process is aborted (i.e. changes are rolled back).
ResolvePartial (see page 2021)	If the **StorageDataSet.MaxResolveErrors** property is set, a ResolutionException with this error code is thrown when all rows have been processed. Any successfully processed rows will be committed.
LogExists (see page 2001)	Existing log files and/or log anchor file found in the directory where the system is attempting to create new log files.
NeedALogdir (see page 2006)	'B' log directory specified without specifying 'A' log directory. The 'A' log directory must be specified. 'B' log directory is an optional duplexing of the 'B' log.
LogdirMismatch (see page 2000)	'A' and 'B' log directories specified for DataStore (see page 1892), but one of them does not exist. The log files must both exist or both not exist. This ensures that the duplexed logs are in sync.
AmbiguousLogdir (see page 1966)	Persistent DataStore (see page 1892) log directory setting contradicts **TxManager** log directory setting and the persistent DataStore (see page 1892) log directory still contains log files. If the log files were copied to the new directory specified by the **TxManager**, the log files in the old location (persisted in the DataStore (see page 1892) file) must be renamed or deleted.
InvalidAnchorLength (see page 1992)	Anchor file has an invalid length. It is probably corrupt, but this should be a rare error, since the previous anchor is copied to at least two separate files. The system redundantly copies this anchor inside the DataStore (see page 1892) file itself, so this could be a recoverable failure. Log duplexing is another way to recover from such a failure. If the log is being duplexed (by setting the **TxManager.BLogDir** property), the A and B anchors are examined and the newest, valid anchor is used.
InvalidAnchorFile (see page 1991)	Anchor file could not be read or has an invalid contents. It is probably corrupt, but this should be a rare error, since the previous anchor is copied to at least two separate files. The system redundantly copies this anchor inside the DataStore (see page 1892) file itself, so this could be a recoverable failure. Log duplexing is another way to recover from such a failure. If the log is being duplexed (by setting the **TxManager.BLogDir** property), The A and B anchors are examined and the newest, valid anchor is used.

BirthstampMismatch (see page 1970)	Birthstamp for "A" anchor file and "B" anchor file do not match. This probably means that the log files were not created for the same DataStore (see page 1892) file.
AnchorTooOld (see page 1967)	Anchor timestamp is older then timestamp of DataStore (see page 1892) file. This is probably because the log files are out of date with the DataStore (see page 1892) file.
TooManyOpenFiles (see page 2027)	Too many simultaneous open files by log manager. This should not occur, but if it does, shut down and restart the system.
InvalidUserName (see page 1998)	Invalid user name for **DataStoreConnection.UserName** property. Typically caused when the property has never been set.
NotTransactional (see page 2013)	Attempted transactional operation when its associated **DataStore.TransacitonMan** property is null.
InvalidLsn (see page 1995)	Unexpected internal condition. Invalid address into a log file.
MissingLogFile (see page 2004)	Missing log file.
MissingCheckpointFile (see page 2004)	Missing checkpoint log file.
TxManagerInuse (see page 2027)	Attempt to use a **TxManager** component instance for two different DataStores.
TxRequired (see page 2028)	This connection cannot be opened because the DataStore (see page 1892) is not transactional or read-only. This error can be corrected by setting the **DataStore.ReadOnly (see page 2018)** property to **true** or by setting the **DataStore.TxManager** property to a non-null **TxManager** instance. the DataStore (see page 1892) JDBC driver can only open connections to transactional or read-only DataStores. Once a DataStore (see page 1892) component is opened with the **TxManager** property set, the DataStore (see page 1892) file is marked transactional and all **TxManager** properties are persisted in the DataStore (see page 1892) file. This operation only needs to be performed once.
TxNotAllowed (see page 2028)	Can't make this DataStore (see page 1892) transactional because it is an older file format.
RecoveryDenied (see page 2019)	Recovery denied by ResponseListener event handler. To prevent this exception and allow recovery to complete, do not deny this event in your application's ResponseListener handler.
ActionComplete (see page 1965)	Do not document or make public. Used internally during recovery.
LockTimeout (see page 2000)	Time-out waiting for lock. This can be caused by a deadlock scenario or because a connection with a lock is taking a long time to commit the transaction that has acquired a lock.
OldLog (see page 2015)	Log file appears to be older than the DataStore (see page 1892) it is associated with. This may be caused by copying old log files to the directory of a DataStore (see page 1892) database file that has more recent changes. You can open the DataStore (see page 1892) in read-only mode (see **DataStoreConnection.setReadOnly**) and copy its contents (see **DataStoreConnection.copyStreams**) to set up new log files.
RowLockTimeout (see page 2023)	Time-out waiting for row lock. This can be caused by a deadlock scenario or because a connection with a lock is taking a long time to commit the transaction that has acquired a lock.
XaActive (see page 2033)	Prepared XA transaction needs to be committed or rolled back.

Deadlock (see page 1981)	Two or more connections have transactions that are acquiring two or more record and/or table locks in an opposite order. Here are some typical scenarios. Read/write access when updating a row. When a transaction reads a row, a share lock is acquired. If the transaction writes to the same row, an exclusive lock must be acquired to update the same row. Share locks can be held by more than one transaction for the same row, but exclusive locks can only be held by one transaction. So if transaction t1 and t2 first acquire a share lock for row 3 and then both try to acquire an exclusive row lock for the same row, a dead lock occurs, because neither transaction can acquire an exclusive lock for the row that they both have share locks on. The most typical solution for this problem is to use the **FOR UPDATE** clause when executing SQL **SELECT** statements for rows that are modified or deleted. Table lock escalation. If more than one thread acquires more than the maximum number of row locks for a table, a deadlock can occur. This is because a shared table lock that prevents exclusive table locks is acquired before row locks are acquired. When the lock manager decides to acquire an exclusive table lock because maximum row locks setting has been exceeded, a dead lock can occur if other transactions are also attempting the same escalation. This is because more than one transaction has a shared lock for the same table they are also trying to get exclusive locks for. Possible solutions include: 1) catch the deadlock exception and retry the operation. These exceptions are typically detected very quickly and automatically rollback the transaction. 2) Use a higher setting for the **maxRowLocks** property setting. Use short-duration write transactions. Connections should try to use short-duration transactions in high concurrency environments. However, in low- or no-concurrency environments, a long-duration transaction can provide better throughput, since fewer commit requests are made. There is a significant overhead to the commit operation, because it must guarantee the durability of a transaction. Using read-only transactions for transactions that do not perform write operations. Read-only transactions aren't blocked by writers or other readers, and because they don't acquire locks, they never block other transactions. To make JDBC connections use read-only transactions, set the **readOnly** property of the java.sql.Connection object (returned by the java.sql.DriverManager.getConnection and com.borland.dx.sql.dataset.Database.getJdbcConnection methods) to **true**. When using DataStoreConnection (see page 1923) objects, set the **readOnlyTx** property to **true** before opening the connection. Read-only transactions work by simulating a snapshot of the database. The snapshot sees only data from transactions that are committed at the point the read-only transaction starts; otherwise, the connection would have to see if there were pending changes and roll them back whenever it accessed the data. A snapshot begins when the DataStoreConnection (see page 1923) opens. It is refreshed every time the commit method is called. See also com.borland.datastore.driver.cons.ExtendedProperties.MAX_ROW_LOCKS.
AppendFailed (see page 1967)	Prepared XA transaction needs to be committed or rolled back.
TxAlreadyStarted (see page 2027)	Operation cannot be completed because transaction already started.
PhysUndoNeeded (see page 2016)	Physical undo needed. Try `DataStoreConnection.setTableLockTables(new String[])` first.
ReadonlytxStartFailed (see page 2019)	Read-only transaction could not start because connection with modifications to the database has not committed.

15.1.2.1.8.1 DataStoreErrorCode Members

The following tables list the members exposed by DataStoreErrorCode.

Internal Fields

	Name	Description
	ActionComplete (see page 1965)	This is ActionComplete, a member of class DataStoreErrorCode.

Public Fields

	Name	Description
	AfterLast (see page 1966)	This is AfterLast, a member of class DataStoreErrorCode.
	AggNotValidHere (see page 1966)	This is AggNotValidHere, a member of class DataStoreErrorCode.
	AggOperatorNotFound (see page 1966)	This is AggOperatorNotFound, a member of class DataStoreErrorCode.
	AlreadyLoading (see page 1966)	This is AlreadyLoading, a member of class DataStoreErrorCode.
	AmbiguousLogdir (see page 1966)	This is AmbiguousLogdir, a member of class DataStoreErrorCode.
	AmbiguousType (see page 1967)	This is AmbiguousType, a member of class DataStoreErrorCode.
	AnchorTooOld (see page 1967)	This is AnchorTooOld, a member of class DataStoreErrorCode.
	AppendFailed (see page 1967)	This is AppendFailed, a member of class DataStoreErrorCode.
	ApplicationError (see page 1967)	This is ApplicationError, a member of class DataStoreErrorCode.
	BadColumnIndex (see page 1967)	This is BadColumnIndex, a member of class DataStoreErrorCode.
	BadDirection (see page 1968)	This is BadDirection, a member of class DataStoreErrorCode.
	BadFetchsize (see page 1968)	This is BadFetchsize, a member of class DataStoreErrorCode.
	BadFieldlist (see page 1968)	This is BadFieldlist, a member of class DataStoreErrorCode.
	BadMaxfieldsize (see page 1968)	This is BadMaxfieldsize, a member of class DataStoreErrorCode.
	BadMaxrows (see page 1968)	This is BadMaxrows, a member of class DataStoreErrorCode.
	BadOrdinal (see page 1969)	This is BadOrdinal, a member of class DataStoreErrorCode.
	BadProcedureProperties (see page 1969)	This is BadProcedureProperties, a member of class DataStoreErrorCode.
	BadQueryProperties (see page 1969)	This is BadQueryProperties, a member of class DataStoreErrorCode.
	BadUrl (see page 1969)	This is BadUrl, a member of class DataStoreErrorCode.
	BeforeFirst (see page 1969)	This is BeforeFirst, a member of class DataStoreErrorCode.
	BigdecimalPrecisionError (see page 1970)	This is BigdecimalPrecisionError, a member of class DataStoreErrorCode.
	BirthstampMismatch (see page 1970)	This is BirthstampMismatch, a member of class DataStoreErrorCode.
	CannotAssignTypes (see page 1970)	This is CannotAssignTypes, a member of class DataStoreErrorCode.
	CannotCast (see page 1970)	This is CannotCast, a member of class DataStoreErrorCode.
	CannotChangeColumn (see page 1970)	This is CannotChangeColumn, a member of class DataStoreErrorCode.
	CannotChangeColumnDataType (see page 1971)	This is CannotChangeColumnDataType, a member of class DataStoreErrorCode.
	CannotDittoExisting (see page 1971)	This is CannotDittoExisting, a member of class DataStoreErrorCode.
	CannotFindTableName (see page 1971)	This is CannotFindTableName, a member of class DataStoreErrorCode.
	CannotImportNullDataset (see page 1971)	This is CannotImportNullDataset, a member of class DataStoreErrorCode.
	CannotLockDir (see page 1971)	This is CannotLockDir, a member of class DataStoreErrorCode.
	CannotMixNamedParameters (see page 1972)	This is CannotMixNamedParameters, a member of class DataStoreErrorCode.
	CannotOpen (see page 1972)	This is CannotOpen, a member of class DataStoreErrorCode.
	CannotOrphanDetails (see page 1972)	This is CannotOrphanDetails, a member of class DataStoreErrorCode.
	CannotRefresh (see page 1972)	This is CannotRefresh, a member of class DataStoreErrorCode.
	CannotRestructure (see page 1972)	This is CannotRestructure, a member of class DataStoreErrorCode.
	CannotSaveChanges (see page 1973)	This is CannotSaveChanges, a member of class DataStoreErrorCode.
	CannotStartServer (see page 1973)	This is CannotStartServer, a member of class DataStoreErrorCode.
	CannotUpdateScopedDataRow (see page 1973)	This is CannotUpdateScopedDataRow, a member of class DataStoreErrorCode.
	CannotUseInternalrow (see page 1973)	This is CannotUseInternalrow, a member of class DataStoreErrorCode.
	CannotUseViewHere (see page 1973)	This is CannotUseViewHere, a member of class DataStoreErrorCode.
	CantCreateOpenFile (see page 1974)	This is CantCreateOpenFile, a member of class DataStoreErrorCode.
	CantCreateOpenStream (see page 1974)	This is CantCreateOpenStream, a member of class DataStoreErrorCode.
	ClassNotFoundError (see page 1974)	This is ClassNotFoundError, a member of class DataStoreErrorCode.
	ColumnAlreadyBound (see page 1974)	This is ColumnAlreadyBound, a member of class DataStoreErrorCode.
	ColumnIsReadonly (see page 1974)	This is ColumnIsReadonly, a member of class DataStoreErrorCode.
	ColumnNeedsRestructure (see page 1975)	This is ColumnNeedsRestructure, a member of class DataStoreErrorCode.
	ColumnNotInLeftRef (see page 1975)	This is ColumnNotInLeftRef, a member of class DataStoreErrorCode.
	ColumnNotInRightRef (see page 1975)	This is ColumnNotInRightRef, a member of class DataStoreErrorCode.

	ColumnSpecifiedTwice (see page 1975)	This is ColumnSpecifiedTwice, a member of class DataStoreErrorCode.
	ColumnTypeConflict (see page 1975)	This is ColumnTypeConflict, a member of class DataStoreErrorCode.
	ConflictingMethodDef (see page 1976)	This is ConflictingMethodDef, a member of class DataStoreErrorCode.
	ConnectionClosed (see page 1976)	This is ConnectionClosed, a member of class DataStoreErrorCode.
	ConnectionCountExceeded (see page 1976)	This is ConnectionCountExceeded, a member of class DataStoreErrorCode.
	ConnectionDescriptorNotSet (see page 1976)	This is ConnectionDescriptorNotSet, a member of class DataStoreErrorCode.
	ConnectionError (see page 1976)	This is ConnectionError, a member of class DataStoreErrorCode.
	ConnectionNotClosed (see page 1977)	This is ConnectionNotClosed, a member of class DataStoreErrorCode.
	ConnectionPoolClosed (see page 1977)	This is ConnectionPoolClosed, a member of class DataStoreErrorCode.
	ConnectionTimeout (see page 1977)	This is ConnectionTimeout, a member of class DataStoreErrorCode.
	ConstraintNameExists (see page 1977)	This is ConstraintNameExists, a member of class DataStoreErrorCode.
	CreateErrorMethodExists (see page 1977)	This is CreateErrorMethodExists, a member of class DataStoreErrorCode.
	CyclesInViewDefinition (see page 1978)	This is CyclesInViewDefinition, a member of class DataStoreErrorCode.
	DataFileLoadFailed (see page 1978)	This is DataFileLoadFailed, a member of class DataStoreErrorCode.
	DataHasDuplicates (see page 1978)	This is DataHasDuplicates, a member of class DataStoreErrorCode.
	DatasetAlreadyOpen (see page 1978)	This is DatasetAlreadyOpen, a member of class DataStoreErrorCode.
	DatasetCorrupt (see page 1978)	This is DatasetCorrupt, a member of class DataStoreErrorCode.
	DatasetExists (see page 1979)	This is DatasetExists, a member of class DataStoreErrorCode.
	DatasetHasNoRows (see page 1979)	This is DatasetHasNoRows, a member of class DataStoreErrorCode.
	DatasetHasNoTables (see page 1979)	This is DatasetHasNoTables, a member of class DataStoreErrorCode.
	DatasetNotOpen (see page 1979)	This is DatasetNotOpen, a member of class DataStoreErrorCode.
	DatasetOpen (see page 1979)	This is DatasetOpen, a member of class DataStoreErrorCode.
	DatastoreAlreadyOpen (see page 1980)	This is DatastoreAlreadyOpen, a member of class DataStoreErrorCode.
	DatastoreExists (see page 1980)	This is DatastoreExists, a member of class DataStoreErrorCode.
	DatastoreInvalid (see page 1980)	This is DatastoreInvalid, a member of class DataStoreErrorCode.
	DatastoreIsUnlicensed (see page 1980)	This is DatastoreIsUnlicensed, a member of class DataStoreErrorCode.
	DatastoreNotFound (see page 1980)	This is DatastoreNotFound, a member of class DataStoreErrorCode.
	DatastoreNotOpen (see page 1981)	This is DatastoreNotOpen, a member of class DataStoreErrorCode.
	DatastoreOpen (see page 1981)	This is DatastoreOpen, a member of class DataStoreErrorCode.
	DbAdminError (see page 1981)	This is DbAdminError, a member of class DataStoreErrorCode.
	Deadlock (see page 1981)	This is Deadlock, a member of class DataStoreErrorCode.
	DeleteDuplicates (see page 1981)	This is DeleteDuplicates, a member of class DataStoreErrorCode.
	DeleteFailed (see page 1982)	This is DeleteFailed, a member of class DataStoreErrorCode.
	DeleteNotAllowed (see page 1982)	This is DeleteNotAllowed, a member of class DataStoreErrorCode.
	DesktopLicense (see page 1982)	This is DesktopLicense, a member of class DataStoreErrorCode.
	DifferentDegree (see page 1982)	This is DifferentDegree, a member of class DataStoreErrorCode.
	DifferentSymbol (see page 1982)	This is DifferentSymbol, a member of class DataStoreErrorCode.
	DivideByZero (see page 1983)	This is DivideByZero, a member of class DataStoreErrorCode.
	DriverNotLoadedAtRuntime (see page 1983)	This is DriverNotLoadedAtRuntime, a member of class DataStoreErrorCode.
	DriverNotLoadedInDesign (see page 1983)	This is DriverNotLoadedInDesign, a member of class DataStoreErrorCode.
	DropErrorDependentView (see page 1983)	This is DropErrorDependentView, a member of class DataStoreErrorCode.
	DropErrorForeignKey (see page 1983)	This is DropErrorForeignKey, a member of class DataStoreErrorCode.
	DropErrorRoleGrant (see page 1984)	This is DropErrorRoleGrant, a member of class DataStoreErrorCode.
	DropSchemaError (see page 1984)	This is DropSchemaError, a member of class DataStoreErrorCode.
	DuplicateColumnName (see page 1984)	This is DuplicateColumnName, a member of class DataStoreErrorCode.
	DuplicateKey (see page 1984)	This is DuplicateKey, a member of class DataStoreErrorCode.
	DuplicatePrimary (see page 1984)	This is DuplicatePrimary, a member of class DataStoreErrorCode.
	EmptyColumnNames (see page 1985)	This is EmptyColumnNames, a member of class DataStoreErrorCode.
	EofExpected (see page 1985)	This is EofExpected, a member of class DataStoreErrorCode.
	ExceptionChain (see page 1985)	This is ExceptionChain, a member of class DataStoreErrorCode.
	FailoverError (see page 1985)	This is FailoverError, a member of class DataStoreErrorCode.
	FeatureNotSupp (see page 1985)	This is FeatureNotSupp, a member of class DataStoreErrorCode.
	FieldPostError (see page 1986)	This is FieldPostError, a member of class DataStoreErrorCode.
	FileExists (see page 1986)	This is FileExists, a member of class DataStoreErrorCode.

	FileNotFound (see page 1986)	This is FileNotFound, a member of class DataStoreErrorCode.
	FloatPrecision (see page 1986)	This is FloatPrecision, a member of class DataStoreErrorCode.
	ForeignKeyError (see page 1986)	This is ForeignKeyError, a member of class DataStoreErrorCode.
	ForeignKeyViolation (see page 1987)	This is ForeignKeyViolation, a member of class DataStoreErrorCode.
	ForwardOnly (see page 1987)	This is ForwardOnly, a member of class DataStoreErrorCode.
	GenericError (see page 1987)	This is GenericError, a member of class DataStoreErrorCode.
	GreaterThanMax (see page 1987)	This is GreaterThanMax, a member of class DataStoreErrorCode.
	HasResultset (see page 1987)	This is HasResultset, a member of class DataStoreErrorCode.
	IllegalCatalog (see page 1988)	This is IllegalCatalog, a member of class DataStoreErrorCode.
	IllegalCharacter (see page 1988)	This is IllegalCharacter, a member of class DataStoreErrorCode.
	IllegalExprStart (see page 1988)	This is IllegalExprStart, a member of class DataStoreErrorCode.
	IllegalMethodCall (see page 1988)	This is IllegalMethodCall, a member of class DataStoreErrorCode.
	IncompatibleBlockSize (see page 1988)	This is IncompatibleBlockSize, a member of class DataStoreErrorCode.
	IncompatibleDataRow (see page 1989)	This is IncompatibleDataRow, a member of class DataStoreErrorCode.
	IncompatibleTypes (see page 1989)	This is IncompatibleTypes, a member of class DataStoreErrorCode.
	IncompatibleTypesForOp (see page 1989)	This is IncompatibleTypesForOp, a member of class DataStoreErrorCode.
	IndexAlreadyExists (see page 1989)	This is IndexAlreadyExists, a member of class DataStoreErrorCode.
	InlineTooSmall (see page 1989)	This is InlineTooSmall, a member of class DataStoreErrorCode.
	InsertFailed (see page 1990)	This is InsertFailed, a member of class DataStoreErrorCode.
	InsertNotAllowed (see page 1990)	This is InsertNotAllowed, a member of class DataStoreErrorCode.
	InsufficientRowid (see page 1990)	This is InsufficientRowid, a member of class DataStoreErrorCode.
	IntegerRequired (see page 1990)	This is IntegerRequired, a member of class DataStoreErrorCode.
	InternalDuplicateKey (see page 1990)	This is InternalDuplicateKey, a member of class DataStoreErrorCode.
	InternalError (see page 1991)	This is InternalError, a member of class DataStoreErrorCode.
	InternalUpdateFailed (see page 1991)	This is InternalUpdateFailed, a member of class DataStoreErrorCode.
	InvalidAggDescriptor (see page 1991)	This is InvalidAggDescriptor, a member of class DataStoreErrorCode.
	InvalidAggExpr (see page 1991)	This is InvalidAggExpr, a member of class DataStoreErrorCode.
	InvalidAnchorFile (see page 1991)	This is InvalidAnchorFile, a member of class DataStoreErrorCode.
	InvalidAnchorLength (see page 1992)	This is InvalidAnchorLength, a member of class DataStoreErrorCode.
	InvalidBinaryString (see page 1992)	This is InvalidBinaryString, a member of class DataStoreErrorCode.
	InvalidClass (see page 1992)	This is InvalidClass, a member of class DataStoreErrorCode.
	InvalidColumnFormat (see page 1992)	This is InvalidColumnFormat, a member of class DataStoreErrorCode.
	InvalidColumnname (see page 1992)	This is InvalidColumnname, a member of class DataStoreErrorCode.
	InvalidColumnPosition (see page 1993)	This is InvalidColumnPosition, a member of class DataStoreErrorCode.
	InvalidColumnRef (see page 1993)	This is InvalidColumnRef, a member of class DataStoreErrorCode.
	InvalidColumnType (see page 1993)	This is InvalidColumnType, a member of class DataStoreErrorCode.
	InvalidColumnValue (see page 1993)	This is InvalidColumnValue, a member of class DataStoreErrorCode.
	InvalidConstraintName (see page 1993)	This is InvalidConstraintName, a member of class DataStoreErrorCode.
	InvalidDataFileFormat (see page 1994)	This is InvalidDataFileFormat, a member of class DataStoreErrorCode.
	InvalidDirectoryAttributes (see page 1994)	This is InvalidDirectoryAttributes, a member of class DataStoreErrorCode.
	InvalidEscapeCharInLike (see page 1994)	This is InvalidEscapeCharInLike, a member of class DataStoreErrorCode.
	InvalidExtractField (see page 1994)	This is InvalidExtractField, a member of class DataStoreErrorCode.
	InvalidFormat (see page 1994)	This is InvalidFormat, a member of class DataStoreErrorCode.
	InvalidIndexname (see page 1995)	This is InvalidIndexname, a member of class DataStoreErrorCode.
	InvalidIteratorUse (see page 1995)	This is InvalidIteratorUse, a member of class DataStoreErrorCode.
	InvalidJoin (see page 1995)	This is InvalidJoin, a member of class DataStoreErrorCode.
	InvalidLikePattern (see page 1995)	This is InvalidLikePattern, a member of class DataStoreErrorCode.
	InvalidLsn (see page 1995)	This is InvalidLsn, a member of class DataStoreErrorCode.
	InvalidPattern (see page 1996)	This is InvalidPattern, a member of class DataStoreErrorCode.
	InvalidPrecision (see page 1996)	This is InvalidPrecision, a member of class DataStoreErrorCode.
	InvalidRowValues (see page 1996)	This is InvalidRowValues, a member of class DataStoreErrorCode.
	InvalidSavepoint (see page 1996)	This is InvalidSavepoint, a member of class DataStoreErrorCode.
	InvalidSchemaFile (see page 1996)	This is InvalidSchemaFile, a member of class DataStoreErrorCode.
	InvalidSort (see page 1997)	This is InvalidSort, a member of class DataStoreErrorCode.

	InvalidSortAsInserted (see page 1997)	This is InvalidSortAsInserted, a member of class DataStoreErrorCode.
	InvalidSortColumn (see page 1997)	This is InvalidSortColumn, a member of class DataStoreErrorCode.
	InvalidStoreClass (see page 1997)	This is InvalidStoreClass, a member of class DataStoreErrorCode.
	InvalidStoreName (see page 1997)	This is InvalidStoreName, a member of class DataStoreErrorCode.
	InvalidTablename (see page 1998)	This is InvalidTablename, a member of class DataStoreErrorCode.
	InvalidTableRef (see page 1998)	This is InvalidTableRef, a member of class DataStoreErrorCode.
	InvalidTrimChar (see page 1998)	This is InvalidTrimChar, a member of class DataStoreErrorCode.
	InvalidUserName (see page 1998)	This is InvalidUserName, a member of class DataStoreErrorCode.
	InvalidViewDefinition (see page 1998)	This is InvalidViewDefinition, a member of class DataStoreErrorCode.
	IoError (see page 1999)	This is IoError, a member of class DataStoreErrorCode.
	IoNetError (see page 1999)	This is IoNetError, a member of class DataStoreErrorCode.
	IoProtocolError (see page 1999)	This is IoProtocolError, a member of class DataStoreErrorCode.
	IsInsertRow (see page 1999)	This is IsInsertRow, a member of class DataStoreErrorCode.
	LessThanMin (see page 1999)	This is LessThanMin, a member of class DataStoreErrorCode.
	LinkColumnsError (see page 2000)	This is LinkColumnsError, a member of class DataStoreErrorCode.
	LinkfieldInUserparameters (see page 2000)	This is LinkfieldInUserparameters, a member of class DataStoreErrorCode.
	LoadingNotStarted (see page 2000)	This is LoadingNotStarted, a member of class DataStoreErrorCode.
	LockTimeout (see page 2000)	This is LockTimeout, a member of class DataStoreErrorCode.
	LogdirMismatch (see page 2000)	This is LogdirMismatch, a member of class DataStoreErrorCode.
	LogExists (see page 2001)	This is LogExists, a member of class DataStoreErrorCode.
	MalformedDateOrTime (see page 2001)	This is MalformedDateOrTime, a member of class DataStoreErrorCode.
	MalformedDecimal (see page 2001)	This is MalformedDecimal, a member of class DataStoreErrorCode.
	MalformedFloating (see page 2001)	This is MalformedFloating, a member of class DataStoreErrorCode.
	MasterDetailViewError (see page 2001)	This is MasterDetailViewError, a member of class DataStoreErrorCode.
	MasterNavigationError (see page 2002)	This is MasterNavigationError, a member of class DataStoreErrorCode.
	MaxResolveFailed (see page 2002)	This is MaxResolveFailed, a member of class DataStoreErrorCode.
	MethodArgMismatch (see page 2002)	This is MethodArgMismatch, a member of class DataStoreErrorCode.
	MethodClassMissing (see page 2002)	This is MethodClassMissing, a member of class DataStoreErrorCode.
	MethodNameMissing (see page 2002)	This is MethodNameMissing, a member of class DataStoreErrorCode.
	MethodNonexistent (see page 2003)	This is MethodNonexistent, a member of class DataStoreErrorCode.
	MethodNotFound (see page 2003)	This is MethodNotFound, a member of class DataStoreErrorCode.
	MirrorError (see page 2003)	This is MirrorError, a member of class DataStoreErrorCode.
	MismatchColumnlistCount (see page 2003)	This is MismatchColumnlistCount, a member of class DataStoreErrorCode.
	MismatchedParameterFormat (see page 2003)	This is MismatchedParameterFormat, a member of class DataStoreErrorCode.
	MismatchInsertColumnCount (see page 2004)	This is MismatchInsertColumnCount, a member of class DataStoreErrorCode.
	MismatchIntoParamCount (see page 2004)	This is MismatchIntoParamCount, a member of class DataStoreErrorCode.
	MismatchParamResult (see page 2004)	This is MismatchParamResult, a member of class DataStoreErrorCode.
	MissingCheckpointFile (see page 2004)	This is MissingCheckpointFile, a member of class DataStoreErrorCode.
	MissingLogFile (see page 2004)	This is MissingLogFile, a member of class DataStoreErrorCode.
	MissingMasterDataset (see page 2005)	This is MissingMasterDataset, a member of class DataStoreErrorCode.
	MissingReplacestorerow (see page 2005)	This is MissingReplacestorerow, a member of class DataStoreErrorCode.
	MissingResolver (see page 2005)	This is MissingResolver, a member of class DataStoreErrorCode.
	MoreThanOneRow (see page 2005)	This is MoreThanOneRow, a member of class DataStoreErrorCode.
	MultipleRowsAffected (see page 2005)	This is MultipleRowsAffected, a member of class DataStoreErrorCode.
	MustBeSingleColumn (see page 2006)	This is MustBeSingleColumn, a member of class DataStoreErrorCode.
	NameNotUnique (see page 2006)	This is NameNotUnique, a member of class DataStoreErrorCode.
	NeedALogdir (see page 2006)	This is NeedALogdir, a member of class DataStoreErrorCode.
	NeedLocateStartOption (see page 2006)	This is NeedLocateStartOption, a member of class DataStoreErrorCode.
	NeedProcedureprovider (see page 2006)	This is NeedProcedureprovider, a member of class DataStoreErrorCode.
	NeedQueryprovider (see page 2007)	This is NeedQueryprovider, a member of class DataStoreErrorCode.
	NeedsRecalc (see page 2007)	This is NeedsRecalc, a member of class DataStoreErrorCode.
	NeedStoragedataset (see page 2007)	This is NeedStoragedataset, a member of class DataStoreErrorCode.
	NegativeSqrtArgument (see page 2007)	This is NegativeSqrtArgument, a member of class DataStoreErrorCode.
	NegativeSubstringLength (see page 2007)	This is NegativeSubstringLength, a member of class DataStoreErrorCode.

	NestedAggregators (see page 2008)	This is NestedAggregators, a member of class DataStoreErrorCode.
	NewerVersion (see page 2008)	This is NewerVersion, a member of class DataStoreErrorCode.
	NewerVersionForOperation (see page 2008)	This is NewerVersionForOperation, a member of class DataStoreErrorCode.
	NoCalcAggFields (see page 2008)	This is NoCalcAggFields, a member of class DataStoreErrorCode.
	NoCalcFields (see page 2008)	This is NoCalcFields, a member of class DataStoreErrorCode.
	NoDatabaseToResolve (see page 2009)	This is NoDatabaseToResolve, a member of class DataStoreErrorCode.
	NodefaultForBinary (see page 2009)	This is NodefaultForBinary, a member of class DataStoreErrorCode.
	NoGroupingColumn (see page 2009)	This is NoGroupingColumn, a member of class DataStoreErrorCode.
	NonExistentRowid (see page 2009)	This is NonExistentRowid, a member of class DataStoreErrorCode.
	NoNonBlobColumns (see page 2009)	This is NoNonBlobColumns, a member of class DataStoreErrorCode.
	NonPrimitiveRequired (see page 2010)	This is NonPrimitiveRequired, a member of class DataStoreErrorCode.
	NonTransactional (see page 2010)	This is NonTransactional, a member of class DataStoreErrorCode.
	NoPrimaryKey (see page 2010)	This is NoPrimaryKey, a member of class DataStoreErrorCode.
	NoPriorOriginalRow (see page 2010)	This is NoPriorOriginalRow, a member of class DataStoreErrorCode.
	NoReadonly (see page 2010)	This is NoReadonly, a member of class DataStoreErrorCode.
	NoResultSet (see page 2011)	This is NoResultSet, a member of class DataStoreErrorCode.
	NoResultsetProduced (see page 2011)	This is NoResultsetProduced, a member of class DataStoreErrorCode.
	NoRowsAffected (see page 2011)	This is NoRowsAffected, a member of class DataStoreErrorCode.
	NoRowsToDelete (see page 2011)	This is NoRowsToDelete, a member of class DataStoreErrorCode.
	NoServerFound (see page 2011)	This is NoServerFound, a member of class DataStoreErrorCode.
	NoSuchParameter (see page 2012)	This is NoSuchParameter, a member of class DataStoreErrorCode.
	NotAnOutputParameter (see page 2012)	This is NotAnOutputParameter, a member of class DataStoreErrorCode.
	NotDatabaseResolver (see page 2012)	This is NotDatabaseResolver, a member of class DataStoreErrorCode.
	NotImplemented (see page 2012)	This is NotImplemented, a member of class DataStoreErrorCode.
	NotInsertRow (see page 2012)	This is NotInsertRow, a member of class DataStoreErrorCode.
	NotSelectQuery (see page 2013)	This is NotSelectQuery, a member of class DataStoreErrorCode.
	NotTransactional (see page 2013)	This is NotTransactional, a member of class DataStoreErrorCode.
	NotUpdateable (see page 2013)	This is NotUpdateable, a member of class DataStoreErrorCode.
	NoUpdatableColumns (see page 2013)	This is NoUpdatableColumns, a member of class DataStoreErrorCode.
	NoWhereClause (see page 2013)	This is NoWhereClause, a member of class DataStoreErrorCode.
	NowReadonly (see page 2014)	This is NowReadonly, a member of class DataStoreErrorCode.
	NullColumnName (see page 2014)	This is NullColumnName, a member of class DataStoreErrorCode.
	NumericOverflow (see page 2014)	This is NumericOverflow, a member of class DataStoreErrorCode.
	NumericUnderflow (see page 2014)	This is NumericUnderflow, a member of class DataStoreErrorCode.
	OlderVersion (see page 2014)	This is OlderVersion, a member of class DataStoreErrorCode.
	OldLog (see page 2015)	This is OldLog, a member of class DataStoreErrorCode.
	OnepassInputStream (see page 2015)	This is OnepassInputStream, a member of class DataStoreErrorCode.
	OperationCanceled (see page 2015)	This is OperationCanceled, a member of class DataStoreErrorCode.
	OutParameterRequired (see page 2015)	This is OutParameterRequired, a member of class DataStoreErrorCode.
	OverloadedDiffParams (see page 2015)	This is OverloadedDiffParams, a member of class DataStoreErrorCode.
	ParameterCountMismatch (see page 2016)	This is ParameterCountMismatch, a member of class DataStoreErrorCode.
	ParameterValueNotSet (see page 2016)	This is ParameterValueNotSet, a member of class DataStoreErrorCode.
	PartialSearchForString (see page 2016)	This is PartialSearchForString, a member of class DataStoreErrorCode.
	PhysUndoNeeded (see page 2016)	This is PhysUndoNeeded, a member of class DataStoreErrorCode.
	PrivilegeMix (see page 2016)	This is PrivilegeMix, a member of class DataStoreErrorCode.
	ProcedureFailed (see page 2017)	This is ProcedureFailed, a member of class DataStoreErrorCode.
	ProcedureInProcess (see page 2017)	This is ProcedureInProcess, a member of class DataStoreErrorCode.
	ProtocolVersion (see page 2017)	This is ProtocolVersion, a member of class DataStoreErrorCode.
	ProviderFailed (see page 2017)	This is ProviderFailed, a member of class DataStoreErrorCode.
	ProviderOwned (see page 2017)	This is ProviderOwned, a member of class DataStoreErrorCode.
	QueryFailed (see page 2018)	This is QueryFailed, a member of class DataStoreErrorCode.
	QueryInProcess (see page 2018)	This is QueryInProcess, a member of class DataStoreErrorCode.
	ReadBlockError (see page 2018)	This is ReadBlockError, a member of class DataStoreErrorCode.
	ReadOnly (see page 2018)	This is ReadOnly, a member of class DataStoreErrorCode.

	ReadOnlyColumn (see page 2018)	This is ReadOnlyColumn, a member of class DataStoreErrorCode.
	ReadOnlyDataset (see page 2019)	This is ReadOnlyDataset, a member of class DataStoreErrorCode.
	ReadOnlyStore (see page 2019)	This is ReadOnlyStore, a member of class DataStoreErrorCode.
	ReadonlyTimeout (see page 2019)	This is ReadonlyTimeout, a member of class DataStoreErrorCode.
	ReadonlytxStartFailed (see page 2019)	This is ReadonlytxStartFailed, a member of class DataStoreErrorCode.
	RecoveryDenied (see page 2019)	This is RecoveryDenied, a member of class DataStoreErrorCode.
	RefreshrowNotSupported (see page 2020)	This is RefreshrowNotSupported, a member of class DataStoreErrorCode.
	RenamedTwice (see page 2020)	This is RenamedTwice, a member of class DataStoreErrorCode.
	ReopenFailure (see page 2020)	This is ReopenFailure, a member of class DataStoreErrorCode.
	ResolveFailed (see page 2020)	This is ResolveFailed, a member of class DataStoreErrorCode.
	ResolveInProgress (see page 2020)	This is ResolveInProgress, a member of class DataStoreErrorCode.
	ResolvePartial (see page 2021)	This is ResolvePartial, a member of class DataStoreErrorCode.
	RestructureDataLoss (see page 2021)	This is RestructureDataLoss, a member of class DataStoreErrorCode.
	RestructureInProgress (see page 2021)	This is RestructureInProgress, a member of class DataStoreErrorCode.
	RestructureParseError (see page 2021)	This is RestructureParseError, a member of class DataStoreErrorCode.
	RestructurePrecisionLoss (see page 2021)	This is RestructurePrecisionLoss, a member of class DataStoreErrorCode.
	ResultNotUpdateable (see page 2022)	This is ResultNotUpdateable, a member of class DataStoreErrorCode.
	ResultsetClosed (see page 2022)	This is ResultsetClosed, a member of class DataStoreErrorCode.
	RevokeErrorDependentPrivilege (see page 2022)	This is RevokeErrorDependentPrivilege, a member of class DataStoreErrorCode.
	RevokeErrorDependentView (see page 2022)	This is RevokeErrorDependentView, a member of class DataStoreErrorCode.
	RevokeErrorForeignKey (see page 2022)	This is RevokeErrorForeignKey, a member of class DataStoreErrorCode.
	RowLockTimeout (see page 2023)	This is RowLockTimeout, a member of class DataStoreErrorCode.
	RowNotFound (see page 2023)	This is RowNotFound, a member of class DataStoreErrorCode.
	RowTooWide (see page 2023)	This is RowTooWide, a member of class DataStoreErrorCode.
	RuntimeError (see page 2023)	This is RuntimeError, a member of class DataStoreErrorCode.
	SecurityError (see page 2023)	This is SecurityError, a member of class DataStoreErrorCode.
	SetCalculatedFailure (see page 2024)	This is SetCalculatedFailure, a member of class DataStoreErrorCode.
	SingleRowHasMoreRows (see page 2024)	This is SingleRowHasMoreRows, a member of class DataStoreErrorCode.
	SqlError (see page 2024)	This is SqlError, a member of class DataStoreErrorCode.
	StatementClosed (see page 2024)	This is StatementClosed, a member of class DataStoreErrorCode.
	StoreNameNotSet (see page 2024)	This is StoreNameNotSet, a member of class DataStoreErrorCode.
	StoreOperationUnsupported (see page 2025)	This is StoreOperationUnsupported, a member of class DataStoreErrorCode.
	StreamClosed (see page 2025)	This is StreamClosed, a member of class DataStoreErrorCode.
	StreamNotFound (see page 2025)	This is StreamNotFound, a member of class DataStoreErrorCode.
	StreamOpenTwice (see page 2025)	This is StreamOpenTwice, a member of class DataStoreErrorCode.
	StringTooLong (see page 2025)	This is StringTooLong, a member of class DataStoreErrorCode.
	SyntaxError (see page 2026)	This is SyntaxError, a member of class DataStoreErrorCode.
	TableAlreadyExists (see page 2026)	This is TableAlreadyExists, a member of class DataStoreErrorCode.
	TableNonexistent (see page 2026)	This is TableNonexistent, a member of class DataStoreErrorCode.
	TooManyAutoIncrements (see page 2026)	This is TooManyAutoIncrements, a member of class DataStoreErrorCode.
	TooManyErrors (see page 2026)	This is TooManyErrors, a member of class DataStoreErrorCode.
	TooManyOpenFiles (see page 2027)	This is TooManyOpenFiles, a member of class DataStoreErrorCode.
	Toomanyspecs (see page 2027)	This is Toomanyspecs, a member of class DataStoreErrorCode.
	TransactionIsolationLevelNotSupported (see page 2027)	This is TransactionIsolationLevelNotSupported, a member of class DataStoreErrorCode.
	TxAlreadyStarted (see page 2027)	This is TxAlreadyStarted, a member of class DataStoreErrorCode.
	TxManagerInuse (see page 2027)	This is TxManagerInuse, a member of class DataStoreErrorCode.
	TxNotAllowed (see page 2028)	This is TxNotAllowed, a member of class DataStoreErrorCode.
	TxRequired (see page 2028)	This is TxRequired, a member of class DataStoreErrorCode.
	TypenameExpected (see page 2028)	This is TypenameExpected, a member of class DataStoreErrorCode.
	UnclosedComment (see page 2028)	This is UnclosedComment, a member of class DataStoreErrorCode.
	UnclosedIdentifier (see page 2028)	This is UnclosedIdentifier, a member of class DataStoreErrorCode.
	UnclosedStringLiteral (see page 2029)	This is UnclosedStringLiteral, a member of class DataStoreErrorCode.
	UnexpectedEndOfQuery (see page 2029)	This is UnexpectedEndOfQuery, a member of class DataStoreErrorCode.
	UnknownColumn (see page 2029)	This is UnknownColumn, a member of class DataStoreErrorCode.

	UnknownColumnName (see page 2029)	This is UnknownColumnName, a member of class DataStoreErrorCode.
	UnknownDetailName (see page 2029)	This is UnknownDetailName, a member of class DataStoreErrorCode.
	UnknownParameter (see page 2030)	This is UnknownParameter, a member of class DataStoreErrorCode.
	UnknownParamName (see page 2030)	This is UnknownParamName, a member of class DataStoreErrorCode.
	UnrecognizedDataType (see page 2030)	This is UnrecognizedDataType, a member of class DataStoreErrorCode.
	UnrecognizedEscape (see page 2030)	This is UnrecognizedEscape, a member of class DataStoreErrorCode.
	UnsupportedColumntype (see page 2030)	This is UnsupportedColumntype, a member of class DataStoreErrorCode.
	UnsupportedDbxType (see page 2031)	This is UnsupportedDbxType, a member of class DataStoreErrorCode.
	UpdateFailed (see page 2031)	This is UpdateFailed, a member of class DataStoreErrorCode.
	UpdateNotAllowed (see page 2031)	This is UpdateNotAllowed, a member of class DataStoreErrorCode.
	UrlNotFound (see page 2031)	This is UrlNotFound, a member of class DataStoreErrorCode.
	UrlNotFoundInDesign (see page 2031)	This is UrlNotFoundInDesign, a member of class DataStoreErrorCode.
	ViewNotUpdatable (see page 2032)	This is ViewNotUpdatable, a member of class DataStoreErrorCode.
	WithCheckOptionViolation (see page 2032)	This is WithCheckOptionViolation, a member of class DataStoreErrorCode.
	WriteBlockError (see page 2032)	This is WriteBlockError, a member of class DataStoreErrorCode.
	WrongCreateschemaname (see page 2032)	This is WrongCreateschemaname, a member of class DataStoreErrorCode.
	WrongDatabase (see page 2032)	This is WrongDatabase, a member of class DataStoreErrorCode.
	WrongSchemaname (see page 2033)	This is WrongSchemaname, a member of class DataStoreErrorCode.
	WrongTableType (see page 2033)	This is WrongTableType, a member of class DataStoreErrorCode.
	XaActive (see page 2033)	This is XaActive, a member of class DataStoreErrorCode.

15.1.2.1.8.2 DataStoreErrorCode Fields

The fields of the DataStoreErrorCode class are listed here.

Internal Fields

	Name	Description
	ActionComplete (see page 1965)	This is ActionComplete, a member of class DataStoreErrorCode.

Public Fields

	Name	Description
	AfterLast (see page 1966)	This is AfterLast, a member of class DataStoreErrorCode.
	AggNotValidHere (see page 1966)	This is AggNotValidHere, a member of class DataStoreErrorCode.
	AggOperatorNotFound (see page 1966)	This is AggOperatorNotFound, a member of class DataStoreErrorCode.
	AlreadyLoading (see page 1966)	This is AlreadyLoading, a member of class DataStoreErrorCode.
	AmbiguousLogdir (see page 1966)	This is AmbiguousLogdir, a member of class DataStoreErrorCode.
	AmbiguousType (see page 1967)	This is AmbiguousType, a member of class DataStoreErrorCode.
	AnchorTooOld (see page 1967)	This is AnchorTooOld, a member of class DataStoreErrorCode.
	AppendFailed (see page 1967)	This is AppendFailed, a member of class DataStoreErrorCode.
	ApplicationError (see page 1967)	This is ApplicationError, a member of class DataStoreErrorCode.
	BadColumnIndex (see page 1967)	This is BadColumnIndex, a member of class DataStoreErrorCode.
	BadDirection (see page 1968)	This is BadDirection, a member of class DataStoreErrorCode.
	BadFetchsize (see page 1968)	This is BadFetchsize, a member of class DataStoreErrorCode.
	BadFieldlist (see page 1968)	This is BadFieldlist, a member of class DataStoreErrorCode.
	BadMaxfieldsize (see page 1968)	This is BadMaxfieldsize, a member of class DataStoreErrorCode.
	BadMaxrows (see page 1968)	This is BadMaxrows, a member of class DataStoreErrorCode.
	BadOrdinal (see page 1969)	This is BadOrdinal, a member of class DataStoreErrorCode.
	BadProcedureProperties (see page 1969)	This is BadProcedureProperties, a member of class DataStoreErrorCode.
	BadQueryProperties (see page 1969)	This is BadQueryProperties, a member of class DataStoreErrorCode.
	BadUrl (see page 1969)	This is BadUrl, a member of class DataStoreErrorCode.
	BeforeFirst (see page 1969)	This is BeforeFirst, a member of class DataStoreErrorCode.
	BigdecimalPrecisionError (see page 1970)	This is BigdecimalPrecisionError, a member of class DataStoreErrorCode.
	BirthstampMismatch (see page 1970)	This is BirthstampMismatch, a member of class DataStoreErrorCode.
	CannotAssignTypes (see page 1970)	This is CannotAssignTypes, a member of class DataStoreErrorCode.
	CannotCast (see page 1970)	This is CannotCast, a member of class DataStoreErrorCode.

	CannotChangeColumn (see page 1970)	This is CannotChangeColumn, a member of class DataStoreErrorCode.
	CannotChangeColumnDataType (see page 1971)	This is CannotChangeColumnDataType, a member of class DataStoreErrorCode.
	CannotDittoExisting (see page 1971)	This is CannotDittoExisting, a member of class DataStoreErrorCode.
	CannotFindTableName (see page 1971)	This is CannotFindTableName, a member of class DataStoreErrorCode.
	CannotImportNullDataset (see page 1971)	This is CannotImportNullDataset, a member of class DataStoreErrorCode.
	CannotLockDir (see page 1971)	This is CannotLockDir, a member of class DataStoreErrorCode.
	CannotMixNamedParameters (see page 1972)	This is CannotMixNamedParameters, a member of class DataStoreErrorCode.
	CannotOpen (see page 1972)	This is CannotOpen, a member of class DataStoreErrorCode.
	CannotOrphanDetails (see page 1972)	This is CannotOrphanDetails, a member of class DataStoreErrorCode.
	CannotRefresh (see page 1972)	This is CannotRefresh, a member of class DataStoreErrorCode.
	CannotRestructure (see page 1972)	This is CannotRestructure, a member of class DataStoreErrorCode.
	CannotSaveChanges (see page 1973)	This is CannotSaveChanges, a member of class DataStoreErrorCode.
	CannotStartServer (see page 1973)	This is CannotStartServer, a member of class DataStoreErrorCode.
	CannotUpdateScopedDataRow (see page 1973)	This is CannotUpdateScopedDataRow, a member of class DataStoreErrorCode.
	CannotUseInternalrow (see page 1973)	This is CannotUseInternalrow, a member of class DataStoreErrorCode.
	CannotUseViewHere (see page 1973)	This is CannotUseViewHere, a member of class DataStoreErrorCode.
	CantCreateOpenFile (see page 1974)	This is CantCreateOpenFile, a member of class DataStoreErrorCode.
	CantCreateOpenStream (see page 1974)	This is CantCreateOpenStream, a member of class DataStoreErrorCode.
	ClassNotFoundError (see page 1974)	This is ClassNotFoundError, a member of class DataStoreErrorCode.
	ColumnAlreadyBound (see page 1974)	This is ColumnAlreadyBound, a member of class DataStoreErrorCode.
	ColumnIsReadonly (see page 1974)	This is ColumnIsReadonly, a member of class DataStoreErrorCode.
	ColumnNeedsRestructure (see page 1975)	This is ColumnNeedsRestructure, a member of class DataStoreErrorCode.
	ColumnNotInLeftRef (see page 1975)	This is ColumnNotInLeftRef, a member of class DataStoreErrorCode.
	ColumnNotInRightRef (see page 1975)	This is ColumnNotInRightRef, a member of class DataStoreErrorCode.
	ColumnSpecifiedTwice (see page 1975)	This is ColumnSpecifiedTwice, a member of class DataStoreErrorCode.
	ColumnTypeConflict (see page 1975)	This is ColumnTypeConflict, a member of class DataStoreErrorCode.
	ConflictingMethodDef (see page 1976)	This is ConflictingMethodDef, a member of class DataStoreErrorCode.
	ConnectionClosed (see page 1976)	This is ConnectionClosed, a member of class DataStoreErrorCode.
	ConnectionCountExceeded (see page 1976)	This is ConnectionCountExceeded, a member of class DataStoreErrorCode.
	ConnectionDescriptorNotSet (see page 1976)	This is ConnectionDescriptorNotSet, a member of class DataStoreErrorCode.
	ConnectionError (see page 1976)	This is ConnectionError, a member of class DataStoreErrorCode.
	ConnectionNotClosed (see page 1977)	This is ConnectionNotClosed, a member of class DataStoreErrorCode.
	ConnectionPoolClosed (see page 1977)	This is ConnectionPoolClosed, a member of class DataStoreErrorCode.
	ConnectionTimeout (see page 1977)	This is ConnectionTimeout, a member of class DataStoreErrorCode.
	ConstraintNameExists (see page 1977)	This is ConstraintNameExists, a member of class DataStoreErrorCode.
	CreateErrorMethodExists (see page 1977)	This is CreateErrorMethodExists, a member of class DataStoreErrorCode.
	CyclesInViewDefinition (see page 1978)	This is CyclesInViewDefinition, a member of class DataStoreErrorCode.
	DataFileLoadFailed (see page 1978)	This is DataFileLoadFailed, a member of class DataStoreErrorCode.
	DataHasDuplicates (see page 1978)	This is DataHasDuplicates, a member of class DataStoreErrorCode.
	DatasetAlreadyOpen (see page 1978)	This is DatasetAlreadyOpen, a member of class DataStoreErrorCode.
	DatasetCorrupt (see page 1978)	This is DatasetCorrupt, a member of class DataStoreErrorCode.
	DatasetExists (see page 1979)	This is DatasetExists, a member of class DataStoreErrorCode.
	DatasetHasNoRows (see page 1979)	This is DatasetHasNoRows, a member of class DataStoreErrorCode.
	DatasetHasNoTables (see page 1979)	This is DatasetHasNoTables, a member of class DataStoreErrorCode.
	DatasetNotOpen (see page 1979)	This is DatasetNotOpen, a member of class DataStoreErrorCode.
	DatasetOpen (see page 1979)	This is DatasetOpen, a member of class DataStoreErrorCode.
	DatastoreAlreadyOpen (see page 1980)	This is DatastoreAlreadyOpen, a member of class DataStoreErrorCode.
	DatastoreExists (see page 1980)	This is DatastoreExists, a member of class DataStoreErrorCode.
	DatastoreInvalid (see page 1980)	This is DatastoreInvalid, a member of class DataStoreErrorCode.
	DatastoreIsUnlicensed (see page 1980)	This is DatastoreIsUnlicensed, a member of class DataStoreErrorCode.
	DatastoreNotFound (see page 1980)	This is DatastoreNotFound, a member of class DataStoreErrorCode.
	DatastoreNotOpen (see page 1981)	This is DatastoreNotOpen, a member of class DataStoreErrorCode.
	DatastoreOpen (see page 1981)	This is DatastoreOpen, a member of class DataStoreErrorCode.
	DbAdminError (see page 1981)	This is DbAdminError, a member of class DataStoreErrorCode.

	Deadlock (see page 1981)	This is Deadlock, a member of class DataStoreErrorCode.
	DeleteDuplicates (see page 1981)	This is DeleteDuplicates, a member of class DataStoreErrorCode.
	DeleteFailed (see page 1982)	This is DeleteFailed, a member of class DataStoreErrorCode.
	DeleteNotAllowed (see page 1982)	This is DeleteNotAllowed, a member of class DataStoreErrorCode.
	DesktopLicense (see page 1982)	This is DesktopLicense, a member of class DataStoreErrorCode.
	DifferentDegree (see page 1982)	This is DifferentDegree, a member of class DataStoreErrorCode.
	DifferentSymbol (see page 1982)	This is DifferentSymbol, a member of class DataStoreErrorCode.
	DivideByZero (see page 1983)	This is DivideByZero, a member of class DataStoreErrorCode.
	DriverNotLoadedAtRuntime (see page 1983)	This is DriverNotLoadedAtRuntime, a member of class DataStoreErrorCode.
	DriverNotLoadedInDesign (see page 1983)	This is DriverNotLoadedInDesign, a member of class DataStoreErrorCode.
	DropErrorDependentView (see page 1983)	This is DropErrorDependentView, a member of class DataStoreErrorCode.
	DropErrorForeignKey (see page 1983)	This is DropErrorForeignKey, a member of class DataStoreErrorCode.
	DropErrorRoleGrant (see page 1984)	This is DropErrorRoleGrant, a member of class DataStoreErrorCode.
	DropSchemaError (see page 1984)	This is DropSchemaError, a member of class DataStoreErrorCode.
	DuplicateColumnName (see page 1984)	This is DuplicateColumnName, a member of class DataStoreErrorCode.
	DuplicateKey (see page 1984)	This is DuplicateKey, a member of class DataStoreErrorCode.
	DuplicatePrimary (see page 1984)	This is DuplicatePrimary, a member of class DataStoreErrorCode.
	EmptyColumnNames (see page 1985)	This is EmptyColumnNames, a member of class DataStoreErrorCode.
	EofExpected (see page 1985)	This is EofExpected, a member of class DataStoreErrorCode.
	ExceptionChain (see page 1985)	This is ExceptionChain, a member of class DataStoreErrorCode.
	FailoverError (see page 1985)	This is FailoverError, a member of class DataStoreErrorCode.
	FeatureNotSupp (see page 1985)	This is FeatureNotSupp, a member of class DataStoreErrorCode.
	FieldPostError (see page 1986)	This is FieldPostError, a member of class DataStoreErrorCode.
	FileExists (see page 1986)	This is FileExists, a member of class DataStoreErrorCode.
	FileNotFound (see page 1986)	This is FileNotFound, a member of class DataStoreErrorCode.
	FloatPrecision (see page 1986)	This is FloatPrecision, a member of class DataStoreErrorCode.
	ForeignKeyError (see page 1986)	This is ForeignKeyError, a member of class DataStoreErrorCode.
	ForeignKeyViolation (see page 1987)	This is ForeignKeyViolation, a member of class DataStoreErrorCode.
	ForwardOnly (see page 1987)	This is ForwardOnly, a member of class DataStoreErrorCode.
	GenericError (see page 1987)	This is GenericError, a member of class DataStoreErrorCode.
	GreaterThanMax (see page 1987)	This is GreaterThanMax, a member of class DataStoreErrorCode.
	HasResultset (see page 1987)	This is HasResultset, a member of class DataStoreErrorCode.
	IllegalCatalog (see page 1988)	This is IllegalCatalog, a member of class DataStoreErrorCode.
	IllegalCharacter (see page 1988)	This is IllegalCharacter, a member of class DataStoreErrorCode.
	IllegalExprStart (see page 1988)	This is IllegalExprStart, a member of class DataStoreErrorCode.
	IllegalMethodCall (see page 1988)	This is IllegalMethodCall, a member of class DataStoreErrorCode.
	IncompatibleBlockSize (see page 1988)	This is IncompatibleBlockSize, a member of class DataStoreErrorCode.
	IncompatibleDataRow (see page 1989)	This is IncompatibleDataRow, a member of class DataStoreErrorCode.
	IncompatibleTypes (see page 1989)	This is IncompatibleTypes, a member of class DataStoreErrorCode.
	IncompatibleTypesForOp (see page 1989)	This is IncompatibleTypesForOp, a member of class DataStoreErrorCode.
	IndexAlreadyExists (see page 1989)	This is IndexAlreadyExists, a member of class DataStoreErrorCode.
	InlineTooSmall (see page 1989)	This is InlineTooSmall, a member of class DataStoreErrorCode.
	InsertFailed (see page 1990)	This is InsertFailed, a member of class DataStoreErrorCode.
	InsertNotAllowed (see page 1990)	This is InsertNotAllowed, a member of class DataStoreErrorCode.
	InsufficientRowid (see page 1990)	This is InsufficientRowid, a member of class DataStoreErrorCode.
	IntegerRequired (see page 1990)	This is IntegerRequired, a member of class DataStoreErrorCode.
	InternalDuplicateKey (see page 1990)	This is InternalDuplicateKey, a member of class DataStoreErrorCode.
	InternalError (see page 1991)	This is InternalError, a member of class DataStoreErrorCode.
	InternalUpdateFailed (see page 1991)	This is InternalUpdateFailed, a member of class DataStoreErrorCode.
	InvalidAggDescriptor (see page 1991)	This is InvalidAggDescriptor, a member of class DataStoreErrorCode.
	InvalidAggExpr (see page 1991)	This is InvalidAggExpr, a member of class DataStoreErrorCode.
	InvalidAnchorFile (see page 1991)	This is InvalidAnchorFile, a member of class DataStoreErrorCode.
	InvalidAnchorLength (see page 1992)	This is InvalidAnchorLength, a member of class DataStoreErrorCode.
	InvalidBinaryString (see page 1992)	This is InvalidBinaryString, a member of class DataStoreErrorCode.

	InvalidClass (see page 1992)	This is InvalidClass, a member of class DataStoreErrorCode.
	InvalidColumnFormat (see page 1992)	This is InvalidColumnFormat, a member of class DataStoreErrorCode.
	InvalidColumnname (see page 1992)	This is InvalidColumnname, a member of class DataStoreErrorCode.
	InvalidColumnPosition (see page 1993)	This is InvalidColumnPosition, a member of class DataStoreErrorCode.
	InvalidColumnRef (see page 1993)	This is InvalidColumnRef, a member of class DataStoreErrorCode.
	InvalidColumnType (see page 1993)	This is InvalidColumnType, a member of class DataStoreErrorCode.
	InvalidColumnValue (see page 1993)	This is InvalidColumnValue, a member of class DataStoreErrorCode.
	InvalidConstraintName (see page 1993)	This is InvalidConstraintName, a member of class DataStoreErrorCode.
	InvalidDataFileFormat (see page 1994)	This is InvalidDataFileFormat, a member of class DataStoreErrorCode.
	InvalidDirectoryAttributes (see page 1994)	This is InvalidDirectoryAttributes, a member of class DataStoreErrorCode.
	InvalidEscapeCharInLike (see page 1994)	This is InvalidEscapeCharInLike, a member of class DataStoreErrorCode.
	InvalidExtractField (see page 1994)	This is InvalidExtractField, a member of class DataStoreErrorCode.
	InvalidFormat (see page 1994)	This is InvalidFormat, a member of class DataStoreErrorCode.
	InvalidIndexname (see page 1995)	This is InvalidIndexname, a member of class DataStoreErrorCode.
	InvalidIteratorUse (see page 1995)	This is InvalidIteratorUse, a member of class DataStoreErrorCode.
	InvalidJoin (see page 1995)	This is InvalidJoin, a member of class DataStoreErrorCode.
	InvalidLikePattern (see page 1995)	This is InvalidLikePattern, a member of class DataStoreErrorCode.
	InvalidLsn (see page 1995)	This is InvalidLsn, a member of class DataStoreErrorCode.
	InvalidPattern (see page 1996)	This is InvalidPattern, a member of class DataStoreErrorCode.
	InvalidPrecision (see page 1996)	This is InvalidPrecision, a member of class DataStoreErrorCode.
	InvalidRowValues (see page 1996)	This is InvalidRowValues, a member of class DataStoreErrorCode.
	InvalidSavepoint (see page 1996)	This is InvalidSavepoint, a member of class DataStoreErrorCode.
	InvalidSchemaFile (see page 1996)	This is InvalidSchemaFile, a member of class DataStoreErrorCode.
	InvalidSort (see page 1997)	This is InvalidSort, a member of class DataStoreErrorCode.
	InvalidSortAsInserted (see page 1997)	This is InvalidSortAsInserted, a member of class DataStoreErrorCode.
	InvalidSortColumn (see page 1997)	This is InvalidSortColumn, a member of class DataStoreErrorCode.
	InvalidStoreClass (see page 1997)	This is InvalidStoreClass, a member of class DataStoreErrorCode.
	InvalidStoreName (see page 1997)	This is InvalidStoreName, a member of class DataStoreErrorCode.
	InvalidTablename (see page 1998)	This is InvalidTablename, a member of class DataStoreErrorCode.
	InvalidTableRef (see page 1998)	This is InvalidTableRef, a member of class DataStoreErrorCode.
	InvalidTrimChar (see page 1998)	This is InvalidTrimChar, a member of class DataStoreErrorCode.
	InvalidUserName (see page 1998)	This is InvalidUserName, a member of class DataStoreErrorCode.
	InvalidViewDefinition (see page 1998)	This is InvalidViewDefinition, a member of class DataStoreErrorCode.
	IoError (see page 1999)	This is IoError, a member of class DataStoreErrorCode.
	IoNetError (see page 1999)	This is IoNetError, a member of class DataStoreErrorCode.
	IoProtocolError (see page 1999)	This is IoProtocolError, a member of class DataStoreErrorCode.
	IsInsertRow (see page 1999)	This is IsInsertRow, a member of class DataStoreErrorCode.
	LessThanMin (see page 1999)	This is LessThanMin, a member of class DataStoreErrorCode.
	LinkColumnsError (see page 2000)	This is LinkColumnsError, a member of class DataStoreErrorCode.
	LinkfieldInUserparameters (see page 2000)	This is LinkfieldInUserparameters, a member of class DataStoreErrorCode.
	LoadingNotStarted (see page 2000)	This is LoadingNotStarted, a member of class DataStoreErrorCode.
	LockTimeout (see page 2000)	This is LockTimeout, a member of class DataStoreErrorCode.
	LogdirMismatch (see page 2000)	This is LogdirMismatch, a member of class DataStoreErrorCode.
	LogExists (see page 2001)	This is LogExists, a member of class DataStoreErrorCode.
	MalformedDateOrTime (see page 2001)	This is MalformedDateOrTime, a member of class DataStoreErrorCode.
	MalformedDecimal (see page 2001)	This is MalformedDecimal, a member of class DataStoreErrorCode.
	MalformedFloating (see page 2001)	This is MalformedFloating, a member of class DataStoreErrorCode.
	MasterDetailViewError (see page 2001)	This is MasterDetailViewError, a member of class DataStoreErrorCode.
	MasterNavigationError (see page 2002)	This is MasterNavigationError, a member of class DataStoreErrorCode.
	MaxResolveFailed (see page 2002)	This is MaxResolveFailed, a member of class DataStoreErrorCode.
	MethodArgMismatch (see page 2002)	This is MethodArgMismatch, a member of class DataStoreErrorCode.
	MethodClassMissing (see page 2002)	This is MethodClassMissing, a member of class DataStoreErrorCode.
	MethodNameMissing (see page 2002)	This is MethodNameMissing, a member of class DataStoreErrorCode.
	MethodNonexistent (see page 2003)	This is MethodNonexistent, a member of class DataStoreErrorCode.

	MethodNotFound (see page 2003)	This is MethodNotFound, a member of class DataStoreErrorCode.
	MirrorError (see page 2003)	This is MirrorError, a member of class DataStoreErrorCode.
	MismatchColumnlistCount (see page 2003)	This is MismatchColumnlistCount, a member of class DataStoreErrorCode.
	MismatchedParameterFormat (see page 2003)	This is MismatchedParameterFormat, a member of class DataStoreErrorCode.
	MismatchInsertColumnCount (see page 2004)	This is MismatchInsertColumnCount, a member of class DataStoreErrorCode.
	MismatchIntoParamCount (see page 2004)	This is MismatchIntoParamCount, a member of class DataStoreErrorCode.
	MismatchParamResult (see page 2004)	This is MismatchParamResult, a member of class DataStoreErrorCode.
	MissingCheckpointFile (see page 2004)	This is MissingCheckpointFile, a member of class DataStoreErrorCode.
	MissingLogFile (see page 2004)	This is MissingLogFile, a member of class DataStoreErrorCode.
	MissingMasterDataset (see page 2005)	This is MissingMasterDataset, a member of class DataStoreErrorCode.
	MissingReplacestorerow (see page 2005)	This is MissingReplacestorerow, a member of class DataStoreErrorCode.
	MissingResolver (see page 2005)	This is MissingResolver, a member of class DataStoreErrorCode.
	MoreThanOneRow (see page 2005)	This is MoreThanOneRow, a member of class DataStoreErrorCode.
	MultipleRowsAffected (see page 2005)	This is MultipleRowsAffected, a member of class DataStoreErrorCode.
	MustBeSingleColumn (see page 2006)	This is MustBeSingleColumn, a member of class DataStoreErrorCode.
	NameNotUnique (see page 2006)	This is NameNotUnique, a member of class DataStoreErrorCode.
	NeedALogdir (see page 2006)	This is NeedALogdir, a member of class DataStoreErrorCode.
	NeedLocateStartOption (see page 2006)	This is NeedLocateStartOption, a member of class DataStoreErrorCode.
	NeedProcedureprovider (see page 2006)	This is NeedProcedureprovider, a member of class DataStoreErrorCode.
	NeedQueryprovider (see page 2007)	This is NeedQueryprovider, a member of class DataStoreErrorCode.
	NeedsRecalc (see page 2007)	This is NeedsRecalc, a member of class DataStoreErrorCode.
	NeedStoragedataset (see page 2007)	This is NeedStoragedataset, a member of class DataStoreErrorCode.
	NegativeSqrtArgument (see page 2007)	This is NegativeSqrtArgument, a member of class DataStoreErrorCode.
	NegativeSubstringLength (see page 2007)	This is NegativeSubstringLength, a member of class DataStoreErrorCode.
	NestedAggregators (see page 2008)	This is NestedAggregators, a member of class DataStoreErrorCode.
	NewerVersion (see page 2008)	This is NewerVersion, a member of class DataStoreErrorCode.
	NewerVersionForOperation (see page 2008)	This is NewerVersionForOperation, a member of class DataStoreErrorCode.
	NoCalcAggFields (see page 2008)	This is NoCalcAggFields, a member of class DataStoreErrorCode.
	NoCalcFields (see page 2008)	This is NoCalcFields, a member of class DataStoreErrorCode.
	NoDatabaseToResolve (see page 2009)	This is NoDatabaseToResolve, a member of class DataStoreErrorCode.
	NodefaultForBinary (see page 2009)	This is NodefaultForBinary, a member of class DataStoreErrorCode.
	NoGroupingColumn (see page 2009)	This is NoGroupingColumn, a member of class DataStoreErrorCode.
	NonExistentRowid (see page 2009)	This is NonExistentRowid, a member of class DataStoreErrorCode.
	NoNonBlobColumns (see page 2009)	This is NoNonBlobColumns, a member of class DataStoreErrorCode.
	NonPrimitiveRequired (see page 2010)	This is NonPrimitiveRequired, a member of class DataStoreErrorCode.
	NonTransactional (see page 2010)	This is NonTransactional, a member of class DataStoreErrorCode.
	NoPrimaryKey (see page 2010)	This is NoPrimaryKey, a member of class DataStoreErrorCode.
	NoPriorOriginalRow (see page 2010)	This is NoPriorOriginalRow, a member of class DataStoreErrorCode.
	NoReadonly (see page 2010)	This is NoReadonly, a member of class DataStoreErrorCode.
	NoResultSet (see page 2011)	This is NoResultSet, a member of class DataStoreErrorCode.
	NoResultsetProduced (see page 2011)	This is NoResultsetProduced, a member of class DataStoreErrorCode.
	NoRowsAffected (see page 2011)	This is NoRowsAffected, a member of class DataStoreErrorCode.
	NoRowsToDelete (see page 2011)	This is NoRowsToDelete, a member of class DataStoreErrorCode.
	NoServerFound (see page 2011)	This is NoServerFound, a member of class DataStoreErrorCode.
	NoSuchParameter (see page 2012)	This is NoSuchParameter, a member of class DataStoreErrorCode.
	NotAnOutputParameter (see page 2012)	This is NotAnOutputParameter, a member of class DataStoreErrorCode.
	NotDatabaseResolver (see page 2012)	This is NotDatabaseResolver, a member of class DataStoreErrorCode.
	NotImplemented (see page 2012)	This is NotImplemented, a member of class DataStoreErrorCode.
	NotInsertRow (see page 2012)	This is NotInsertRow, a member of class DataStoreErrorCode.
	NotSelectQuery (see page 2013)	This is NotSelectQuery, a member of class DataStoreErrorCode.
	NotTransactional (see page 2013)	This is NotTransactional, a member of class DataStoreErrorCode.
	NotUpdateable (see page 2013)	This is NotUpdateable, a member of class DataStoreErrorCode.
	NoUpdatableColumns (see page 2013)	This is NoUpdatableColumns, a member of class DataStoreErrorCode.
	NoWhereClause (see page 2013)	This is NoWhereClause, a member of class DataStoreErrorCode.

	NowReadonly (see page 2014)	This is NowReadonly, a member of class DataStoreErrorCode.
	NullColumnName (see page 2014)	This is NullColumnName, a member of class DataStoreErrorCode.
	NumericOverflow (see page 2014)	This is NumericOverflow, a member of class DataStoreErrorCode.
	NumericUnderflow (see page 2014)	This is NumericUnderflow, a member of class DataStoreErrorCode.
	OlderVersion (see page 2014)	This is OlderVersion, a member of class DataStoreErrorCode.
	OldLog (see page 2015)	This is OldLog, a member of class DataStoreErrorCode.
	OnepassInputStream (see page 2015)	This is OnepassInputStream, a member of class DataStoreErrorCode.
	OperationCanceled (see page 2015)	This is OperationCanceled, a member of class DataStoreErrorCode.
	OutParameterRequired (see page 2015)	This is OutParameterRequired, a member of class DataStoreErrorCode.
	OverloadedDiffParams (see page 2015)	This is OverloadedDiffParams, a member of class DataStoreErrorCode.
	ParameterCountMismatch (see page 2016)	This is ParameterCountMismatch, a member of class DataStoreErrorCode.
	ParameterValueNotSet (see page 2016)	This is ParameterValueNotSet, a member of class DataStoreErrorCode.
	PartialSearchForString (see page 2016)	This is PartialSearchForString, a member of class DataStoreErrorCode.
	PhysUndoNeeded (see page 2016)	This is PhysUndoNeeded, a member of class DataStoreErrorCode.
	PrivilegeMix (see page 2016)	This is PrivilegeMix, a member of class DataStoreErrorCode.
	ProcedureFailed (see page 2017)	This is ProcedureFailed, a member of class DataStoreErrorCode.
	ProcedureInProcess (see page 2017)	This is ProcedureInProcess, a member of class DataStoreErrorCode.
	ProtocolVersion (see page 2017)	This is ProtocolVersion, a member of class DataStoreErrorCode.
	ProviderFailed (see page 2017)	This is ProviderFailed, a member of class DataStoreErrorCode.
	ProviderOwned (see page 2017)	This is ProviderOwned, a member of class DataStoreErrorCode.
	QueryFailed (see page 2018)	This is QueryFailed, a member of class DataStoreErrorCode.
	QueryInProcess (see page 2018)	This is QueryInProcess, a member of class DataStoreErrorCode.
	ReadBlockError (see page 2018)	This is ReadBlockError, a member of class DataStoreErrorCode.
	ReadOnly (see page 2018)	This is ReadOnly, a member of class DataStoreErrorCode.
	ReadOnlyColumn (see page 2018)	This is ReadOnlyColumn, a member of class DataStoreErrorCode.
	ReadOnlyDataset (see page 2019)	This is ReadOnlyDataset, a member of class DataStoreErrorCode.
	ReadOnlyStore (see page 2019)	This is ReadOnlyStore, a member of class DataStoreErrorCode.
	ReadonlyTimeout (see page 2019)	This is ReadonlyTimeout, a member of class DataStoreErrorCode.
	ReadonlytxStartFailed (see page 2019)	This is ReadonlytxStartFailed, a member of class DataStoreErrorCode.
	RecoveryDenied (see page 2019)	This is RecoveryDenied, a member of class DataStoreErrorCode.
	RefreshrowNotSupported (see page 2020)	This is RefreshrowNotSupported, a member of class DataStoreErrorCode.
	RenamedTwice (see page 2020)	This is RenamedTwice, a member of class DataStoreErrorCode.
	ReopenFailure (see page 2020)	This is ReopenFailure, a member of class DataStoreErrorCode.
	ResolveFailed (see page 2020)	This is ResolveFailed, a member of class DataStoreErrorCode.
	ResolveInProgress (see page 2020)	This is ResolveInProgress, a member of class DataStoreErrorCode.
	ResolvePartial (see page 2021)	This is ResolvePartial, a member of class DataStoreErrorCode.
	RestructureDataLoss (see page 2021)	This is RestructureDataLoss, a member of class DataStoreErrorCode.
	RestructureInProgress (see page 2021)	This is RestructureInProgress, a member of class DataStoreErrorCode.
	RestructureParseError (see page 2021)	This is RestructureParseError, a member of class DataStoreErrorCode.
	RestructurePrecisionLoss (see page 2021)	This is RestructurePrecisionLoss, a member of class DataStoreErrorCode.
	ResultNotUpdateable (see page 2022)	This is ResultNotUpdateable, a member of class DataStoreErrorCode.
	ResultsetClosed (see page 2022)	This is ResultsetClosed, a member of class DataStoreErrorCode.
	RevokeErrorDependentPrivilege (see page 2022)	This is RevokeErrorDependentPrivilege, a member of class DataStoreErrorCode.
	RevokeErrorDependentView (see page 2022)	This is RevokeErrorDependentView, a member of class DataStoreErrorCode.
	RevokeErrorForeignKey (see page 2022)	This is RevokeErrorForeignKey, a member of class DataStoreErrorCode.
	RowLockTimeout (see page 2023)	This is RowLockTimeout, a member of class DataStoreErrorCode.
	RowNotFound (see page 2023)	This is RowNotFound, a member of class DataStoreErrorCode.
	RowTooWide (see page 2023)	This is RowTooWide, a member of class DataStoreErrorCode.
	RuntimeError (see page 2023)	This is RuntimeError, a member of class DataStoreErrorCode.
	SecurityError (see page 2023)	This is SecurityError, a member of class DataStoreErrorCode.
	SetCalculatedFailure (see page 2024)	This is SetCalculatedFailure, a member of class DataStoreErrorCode.
	SingleRowHasMoreRows (see page 2024)	This is SingleRowHasMoreRows, a member of class DataStoreErrorCode.
	SqlError (see page 2024)	This is SqlError, a member of class DataStoreErrorCode.
	StatementClosed (see page 2024)	This is StatementClosed, a member of class DataStoreErrorCode.

	StoreNameNotSet (see page 2024)	This is StoreNameNotSet, a member of class DataStoreErrorCode.
	StoreOperationUnsupported (see page 2025)	This is StoreOperationUnsupported, a member of class DataStoreErrorCode.
	StreamClosed (see page 2025)	This is StreamClosed, a member of class DataStoreErrorCode.
	StreamNotFound (see page 2025)	This is StreamNotFound, a member of class DataStoreErrorCode.
	StreamOpenTwice (see page 2025)	This is StreamOpenTwice, a member of class DataStoreErrorCode.
	StringTooLong (see page 2025)	This is StringTooLong, a member of class DataStoreErrorCode.
	SyntaxError (see page 2026)	This is SyntaxError, a member of class DataStoreErrorCode.
	TableAlreadyExists (see page 2026)	This is TableAlreadyExists, a member of class DataStoreErrorCode.
	TableNonexistent (see page 2026)	This is TableNonexistent, a member of class DataStoreErrorCode.
	TooManyAutoIncrements (see page 2026)	This is TooManyAutoIncrements, a member of class DataStoreErrorCode.
	TooManyErrors (see page 2026)	This is TooManyErrors, a member of class DataStoreErrorCode.
	TooManyOpenFiles (see page 2027)	This is TooManyOpenFiles, a member of class DataStoreErrorCode.
	Toomanyspecs (see page 2027)	This is Toomanyspecs, a member of class DataStoreErrorCode.
	TransactionIsolationLevelNotSupported (see page 2027)	This is TransactionIsolationLevelNotSupported, a member of class DataStoreErrorCode.
	TxAlreadyStarted (see page 2027)	This is TxAlreadyStarted, a member of class DataStoreErrorCode.
	TxManagerInuse (see page 2027)	This is TxManagerInuse, a member of class DataStoreErrorCode.
	TxNotAllowed (see page 2028)	This is TxNotAllowed, a member of class DataStoreErrorCode.
	TxRequired (see page 2028)	This is TxRequired, a member of class DataStoreErrorCode.
	TypenameExpected (see page 2028)	This is TypenameExpected, a member of class DataStoreErrorCode.
	UnclosedComment (see page 2028)	This is UnclosedComment, a member of class DataStoreErrorCode.
	UnclosedIdentifier (see page 2028)	This is UnclosedIdentifier, a member of class DataStoreErrorCode.
	UnclosedStringLiteral (see page 2029)	This is UnclosedStringLiteral, a member of class DataStoreErrorCode.
	UnexpectedEndOfQuery (see page 2029)	This is UnexpectedEndOfQuery, a member of class DataStoreErrorCode.
	UnknownColumn (see page 2029)	This is UnknownColumn, a member of class DataStoreErrorCode.
	UnknownColumnName (see page 2029)	This is UnknownColumnName, a member of class DataStoreErrorCode.
	UnknownDetailName (see page 2029)	This is UnknownDetailName, a member of class DataStoreErrorCode.
	UnknownParameter (see page 2030)	This is UnknownParameter, a member of class DataStoreErrorCode.
	UnknownParamName (see page 2030)	This is UnknownParamName, a member of class DataStoreErrorCode.
	UnrecognizedDataType (see page 2030)	This is UnrecognizedDataType, a member of class DataStoreErrorCode.
	UnrecognizedEscape (see page 2030)	This is UnrecognizedEscape, a member of class DataStoreErrorCode.
	UnsupportedColumntype (see page 2030)	This is UnsupportedColumntype, a member of class DataStoreErrorCode.
	UnsupportedDbxType (see page 2031)	This is UnsupportedDbxType, a member of class DataStoreErrorCode.
	UpdateFailed (see page 2031)	This is UpdateFailed, a member of class DataStoreErrorCode.
	UpdateNotAllowed (see page 2031)	This is UpdateNotAllowed, a member of class DataStoreErrorCode.
	UrlNotFound (see page 2031)	This is UrlNotFound, a member of class DataStoreErrorCode.
	UrlNotFoundInDesign (see page 2031)	This is UrlNotFoundInDesign, a member of class DataStoreErrorCode.
	ViewNotUpdatable (see page 2032)	This is ViewNotUpdatable, a member of class DataStoreErrorCode.
	WithCheckOptionViolation (see page 2032)	This is WithCheckOptionViolation, a member of class DataStoreErrorCode.
	WriteBlockError (see page 2032)	This is WriteBlockError, a member of class DataStoreErrorCode.
	WrongCreateschemaname (see page 2032)	This is WrongCreateschemaname, a member of class DataStoreErrorCode.
	WrongDatabase (see page 2032)	This is WrongDatabase, a member of class DataStoreErrorCode.
	WrongSchemaname (see page 2033)	This is WrongSchemaname, a member of class DataStoreErrorCode.
	WrongTableType (see page 2033)	This is WrongTableType, a member of class DataStoreErrorCode.
	XaActive (see page 2033)	This is XaActive, a member of class DataStoreErrorCode.

15.1.2.1.8.2.1 DataStoreErrorCode.ActionComplete Field

Pascal

```
const ActionComplete: Integer = 6000 + 23;
```

C++

```
int ActionComplete = 6000 + 23;
```

Description

This is ActionComplete, a member of class DataStoreErrorCode.

15.1.2.1.8.2.2 DataStoreErrorCode.AfterLast Field

Pascal

```
const AfterLast: Integer = 3000 + 1;
```

C++

```
int AfterLast = 3000 + 1;
```

Description

This is AfterLast, a member of class DataStoreErrorCode.

15.1.2.1.8.2.3 DataStoreErrorCode.AggNotValidHere Field

Pascal

```
const AggNotValidHere: Integer = 3000 + 2;
```

C++

```
int AggNotValidHere = 3000 + 2;
```

Description

This is AggNotValidHere, a member of class DataStoreErrorCode.

15.1.2.1.8.2.4 DataStoreErrorCode.AggOperatorNotFound Field

Pascal

```
const AggOperatorNotFound: Integer = 2000 + 24;
```

C++

```
int AggOperatorNotFound = 2000 + 24;
```

Description

This is AggOperatorNotFound, a member of class DataStoreErrorCode.

15.1.2.1.8.2.5 DataStoreErrorCode.AlreadyLoading Field

Pascal

```
const AlreadyLoading: Integer = 0 + 28;
```

C++

```
int AlreadyLoading = 0 + 28;
```

Description

This is AlreadyLoading, a member of class DataStoreErrorCode.

15.1.2.1.8.2.6 DataStoreErrorCode.AmbiguousLogdir Field

Pascal

```
const AmbiguousLogdir: Integer = 6000 + 5;
```

C++

```
int AmbiguousLogdir = 6000 + 5;
```

Description

This is AmbiguousLogdir, a member of class DataStoreErrorCode.

15.1.2.1.8.2.7 DataStoreErrorCode.AmbiguousType Field

Pascal

```
const AmbiguousType: Integer = 3000 + 4;
```

C++

```
int AmbiguousType = 3000 + 4;
```

Description

This is AmbiguousType, a member of class DataStoreErrorCode.

15.1.2.1.8.2.8 DataStoreErrorCode.AnchorTooOld Field

Pascal

```
const AnchorTooOld: Integer = 6000 + 9;
```

C++

```
int AnchorTooOld = 6000 + 9;
```

Description

This is AnchorTooOld, a member of class DataStoreErrorCode.

15.1.2.1.8.2.9 DataStoreErrorCode.AppendFailed Field

Pascal

```
const AppendFailed: Integer = 6000 + 29;
```

C++

```
int AppendFailed = 6000 + 29;
```

Description

This is AppendFailed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.10 DataStoreErrorCode.ApplicationError Field

Pascal

```
const ApplicationError: Integer = 1000 + 12;
```

C++

```
int ApplicationError = 1000 + 12;
```

Description

This is ApplicationError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.11 DataStoreErrorCode.BadColumnIndex Field

Pascal

```
const BadColumnIndex: Integer = 3000 + 5;
```

C++

```
int BadColumnIndex = 3000 + 5;
```

Description

This is BadColumnIndex, a member of class DataStoreErrorCode.

15.1.2.1.8.2.12 DataStoreErrorCode.BadDirection Field

Pascal

```
const BadDirection: Integer = 3000 + 6;
```

C++

```
int BadDirection = 3000 + 6;
```

Description

This is BadDirection, a member of class DataStoreErrorCode.

15.1.2.1.8.2.13 DataStoreErrorCode.BadFetchsize Field

Pascal

```
const BadFetchsize: Integer = 3000 + 7;
```

C++

```
int BadFetchsize = 3000 + 7;
```

Description

This is BadFetchsize, a member of class DataStoreErrorCode.

15.1.2.1.8.2.14 DataStoreErrorCode.BadFieldlist Field

Pascal

```
const BadFieldlist: Integer = 7000 + 2;
```

C++

```
int BadFieldlist = 7000 + 2;
```

Description

This is BadFieldlist, a member of class DataStoreErrorCode.

15.1.2.1.8.2.15 DataStoreErrorCode.BadMaxfieldsize Field

Pascal

```
const BadMaxfieldsize: Integer = 3000 + 8;
```

C++

```
int BadMaxfieldsize = 3000 + 8;
```

Description

This is BadMaxfieldsize, a member of class DataStoreErrorCode.

15.1.2.1.8.2.16 DataStoreErrorCode.BadMaxrows Field

Pascal

```
const BadMaxrows: Integer = 3000 + 9;
```

C++

```
int BadMaxrows = 3000 + 9;
```

Description

This is BadMaxrows, a member of class DataStoreErrorCode.

15.1.2.1.8.2.17 DataStoreErrorCode.BadOrdinal Field

Pascal

```
const BadOrdinal: Integer = 3000 + 10;
```

C++

```
int BadOrdinal = 3000 + 10;
```

Description

This is BadOrdinal, a member of class DataStoreErrorCode.

15.1.2.1.8.2.18 DataStoreErrorCode.BadProcedureProperties Field

Pascal

```
const BadProcedureProperties: Integer = 0 + 89;
```

C++

```
int BadProcedureProperties = 0 + 89;
```

Description

This is BadProcedureProperties, a member of class DataStoreErrorCode.

15.1.2.1.8.2.19 DataStoreErrorCode.BadQueryProperties Field

Pascal

```
const BadQueryProperties: Integer = 0 + 1;
```

C++

```
int BadQueryProperties = 0 + 1;
```

Description

This is BadQueryProperties, a member of class DataStoreErrorCode.

15.1.2.1.8.2.20 DataStoreErrorCode.BadUrl Field

Pascal

```
const BadUrl: Integer = 3000 + 11;
```

C++

```
int BadUrl = 3000 + 11;
```

Description

This is BadUrl, a member of class DataStoreErrorCode.

15.1.2.1.8.2.21 DataStoreErrorCode.BeforeFirst Field

Pascal

```
const BeforeFirst: Integer = 3000 + 12;
```

C++

```
int BeforeFirst = 3000 + 12;
```

Description

This is BeforeFirst, a member of class DataStoreErrorCode.

15.1.2.1.8.2.22 DataStoreErrorCode.BigdecimalPrecisionError Field

Pascal

```
const BigdecimalPrecisionError: Integer = 2000 + 29;
```

C++

```
int BigdecimalPrecisionError = 2000 + 29;
```

Description

This is BigdecimalPrecisionError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.23 DataStoreErrorCode.BirthstampMismatch Field

Pascal

```
const BirthstampMismatch: Integer = 6000 + 8;
```

C++

```
int BirthstampMismatch = 6000 + 8;
```

Description

This is BirthstampMismatch, a member of class DataStoreErrorCode.

15.1.2.1.8.2.24 DataStoreErrorCode.CannotAssignTypes Field

Pascal

```
const CannotAssignTypes: Integer = 3000 + 14;
```

C++

```
int CannotAssignTypes = 3000 + 14;
```

Description

This is CannotAssignTypes, a member of class DataStoreErrorCode.

15.1.2.1.8.2.25 DataStoreErrorCode.CannotCast Field

Pascal

```
const CannotCast: Integer = 3000 + 15;
```

C++

```
int CannotCast = 3000 + 15;
```

Description

This is CannotCast, a member of class DataStoreErrorCode.

15.1.2.1.8.2.26 DataStoreErrorCode.CannotChangeColumn Field

Pascal

```
const CannotChangeColumn: Integer = 0 + 70;
```

C++

```
int CannotChangeColumn = 0 + 70;
```

Description

This is CannotChangeColumn, a member of class DataStoreErrorCode.

15.1.2.1.8.2.27 DataStoreErrorCode.CannotChangeColumnDataType Field

Pascal

```
const CannotChangeColumnDataType: Integer = 0 + 14;
```

C++

```
int CannotChangeColumnDataType = 0 + 14;
```

Description

This is CannotChangeColumnDataType, a member of class DataStoreErrorCode.

15.1.2.1.8.2.28 DataStoreErrorCode.CannotDittoExisting Field

Pascal

```
const CannotDittoExisting: Integer = 1000 + 10;
```

C++

```
int CannotDittoExisting = 1000 + 10;
```

Description

This is CannotDittoExisting, a member of class DataStoreErrorCode.

15.1.2.1.8.2.29 DataStoreErrorCode.CannotFindTableName Field

Pascal

```
const CannotFindTableName: Integer = 0 + 7;
```

C++

```
int CannotFindTableName = 0 + 7;
```

Description

This is CannotFindTableName, a member of class DataStoreErrorCode.

15.1.2.1.8.2.30 DataStoreErrorCode.CannotImportNullDataset Field

Pascal

```
const CannotImportNullDataset: Integer = 0 + 9;
```

C++

```
int CannotImportNullDataset = 0 + 9;
```

Description

This is CannotImportNullDataset, a member of class DataStoreErrorCode.

15.1.2.1.8.2.31 DataStoreErrorCode.CannotLockDir Field

Pascal

```
const CannotLockDir: Integer = CannotOpen;
```

C++

```
int CannotLockDir = CannotOpen;
```

Description

This is CannotLockDir, a member of class DataStoreErrorCode.

15.1.2.1.8.2.32 DataStoreErrorCode.CannotMixNamedParameters Field

Pascal

```
const CannotMixNamedParameters: Integer = 3000 + 116;
```

C++

```
int CannotMixNamedParameters = 3000 + 116;
```

Description

This is CannotMixNamedParameters, a member of class DataStoreErrorCode.

15.1.2.1.8.2.33 DataStoreErrorCode.CannotOpen Field

Pascal

```
const CannotOpen: Integer = 2000 + 47;
```

C++

```
int CannotOpen = 2000 + 47;
```

Description

This is CannotOpen, a member of class DataStoreErrorCode.

15.1.2.1.8.2.34 DataStoreErrorCode.CannotOrphanDetails Field

Pascal

```
const CannotOrphanDetails: Integer = 1000 + 3;
```

C++

```
int CannotOrphanDetails = 1000 + 3;
```

Description

This is CannotOrphanDetails, a member of class DataStoreErrorCode.

15.1.2.1.8.2.35 DataStoreErrorCode.CannotRefresh Field

Pascal

```
const CannotRefresh: Integer = 0 + 77;
```

C++

```
int CannotRefresh = 0 + 77;
```

Description

This is CannotRefresh, a member of class DataStoreErrorCode.

15.1.2.1.8.2.36 DataStoreErrorCode.CannotRestructure Field

Pascal

```
const CannotRestructure: Integer = 2000 + 8;
```

C++

```
int CannotRestructure = 2000 + 8;
```

Description

This is CannotRestructure, a member of class DataStoreErrorCode.

15.1.2.1.8.2.37 DataStoreErrorCode.CannotSaveChanges Field

Pascal

```
const CannotSaveChanges: Integer = 0 + 76;
```

C++

```
int CannotSaveChanges = 0 + 76;
```

Description

This is CannotSaveChanges, a member of class DataStoreErrorCode.

15.1.2.1.8.2.38 DataStoreErrorCode.CannotStartServer Field

Pascal

```
const CannotStartServer: Integer = 2000 + 67;
```

C++

```
int CannotStartServer = 2000 + 67;
```

Description

This is CannotStartServer, a member of class DataStoreErrorCode.

15.1.2.1.8.2.39 DataStoreErrorCode.CannotUpdateScopedDataRow Field

Pascal

```
const CannotUpdateScopedDataRow: Integer = 0 + 30;
```

C++

```
int CannotUpdateScopedDataRow = 0 + 30;
```

Description

This is CannotUpdateScopedDataRow, a member of class DataStoreErrorCode.

15.1.2.1.8.2.40 DataStoreErrorCode.CannotUseInternalrow Field

Pascal

```
const CannotUseInternalrow: Integer = 3000 + 17;
```

C++

```
int CannotUseInternalrow = 3000 + 17;
```

Description

This is CannotUseInternalrow, a member of class DataStoreErrorCode.

15.1.2.1.8.2.41 DataStoreErrorCode.CannotUseViewHere Field

Pascal

```
const CannotUseViewHere: Integer = 3000 + 18;
```

C++

```
int CannotUseViewHere = 3000 + 18;
```

Description

This is CannotUseViewHere, a member of class DataStoreErrorCode.

15.1.2.1.8.2.42 DataStoreErrorCode.CantCreateOpenFile Field

Pascal

```
const CantCreateOpenFile: Integer = CantCreateOpenStream;
```

C++

```
int CantCreateOpenFile = CantCreateOpenStream;
```

Description

This is CantCreateOpenFile, a member of class DataStoreErrorCode.

15.1.2.1.8.2.43 DataStoreErrorCode.CantCreateOpenStream Field

Pascal

```
const CantCreateOpenStream: Integer = 2000 + 32;
```

C++

```
int CantCreateOpenStream = 2000 + 32;
```

Description

This is CantCreateOpenStream, a member of class DataStoreErrorCode.

15.1.2.1.8.2.44 DataStoreErrorCode.ClassNotFoundError Field

Pascal

```
const ClassNotFoundError: Integer = 0 + 99;
```

C++

```
int ClassNotFoundError = 0 + 99;
```

Description

This is ClassNotFoundError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.45 DataStoreErrorCode.ColumnAlreadyBound Field

Pascal

```
const ColumnAlreadyBound: Integer = 0 + 13;
```

C++

```
int ColumnAlreadyBound = 0 + 13;
```

Description

This is ColumnAlreadyBound, a member of class DataStoreErrorCode.

15.1.2.1.8.2.46 DataStoreErrorCode.ColumnIsReadonly Field

Pascal

```
const ColumnIsReadonly: Integer = 3000 + 91;
```

C++

```
int ColumnIsReadonly = 3000 + 91;
```

Description

This is ColumnIsReadonly, a member of class DataStoreErrorCode.

15.1.2.1.8.2.47 DataStoreErrorCode.ColumnNeedsRestructure Field

Pascal

```
const ColumnNeedsRestructure: Integer = 2000 + 9;
```

C++

```
int ColumnNeedsRestructure = 2000 + 9;
```

Description

This is ColumnNeedsRestructure, a member of class DataStoreErrorCode.

15.1.2.1.8.2.48 DataStoreErrorCode.ColumnNotInLeftRef Field

Pascal

```
const ColumnNotInLeftRef: Integer = 3000 + 19;
```

C++

```
int ColumnNotInLeftRef = 3000 + 19;
```

Description

This is ColumnNotInLeftRef, a member of class DataStoreErrorCode.

15.1.2.1.8.2.49 DataStoreErrorCode.ColumnNotInRightRef Field

Pascal

```
const ColumnNotInRightRef: Integer = 3000 + 20;
```

C++

```
int ColumnNotInRightRef = 3000 + 20;
```

Description

This is ColumnNotInRightRef, a member of class DataStoreErrorCode.

15.1.2.1.8.2.50 DataStoreErrorCode.ColumnSpecifiedTwice Field

Pascal

```
const ColumnSpecifiedTwice: Integer = 3000 + 21;
```

C++

```
int ColumnSpecifiedTwice = 3000 + 21;
```

Description

This is ColumnSpecifiedTwice, a member of class DataStoreErrorCode.

15.1.2.1.8.2.51 DataStoreErrorCode.ColumnTypeConflict Field

Pascal

```
const ColumnTypeConflict: Integer = 0 + 11;
```

C++

```
int ColumnTypeConflict = 0 + 11;
```

Description

This is ColumnTypeConflict, a member of class DataStoreErrorCode.

15.1.2.1.8.2.52 DataStoreErrorCode.ConflictingMethodDef Field

Pascal

```
const ConflictingMethodDef: Integer = 2000 + 57;
```

C++

```
int ConflictingMethodDef = 2000 + 57;
```

Description

This is ConflictingMethodDef, a member of class DataStoreErrorCode.

15.1.2.1.8.2.53 DataStoreErrorCode.ConnectionClosed Field

Pascal

```
const ConnectionClosed: Integer = 3000 + 22;
```

C++

```
int ConnectionClosed = 3000 + 22;
```

Description

This is ConnectionClosed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.54 DataStoreErrorCode.ConnectionCountExceeded Field

Pascal

```
const ConnectionCountExceeded: Integer = 2000 + 50;
```

C++

```
int ConnectionCountExceeded = 2000 + 50;
```

Description

This is ConnectionCountExceeded, a member of class DataStoreErrorCode.

15.1.2.1.8.2.55 DataStoreErrorCode.ConnectionDescriptorNotSet Field

Pascal

```
const ConnectionDescriptorNotSet: Integer = 0 + 45;
```

C++

```
int ConnectionDescriptorNotSet = 0 + 45;
```

Description

This is ConnectionDescriptorNotSet, a member of class DataStoreErrorCode.

15.1.2.1.8.2.56 DataStoreErrorCode.ConnectionError Field

Pascal

```
const ConnectionError: Integer = 3000 + 23;
```

C++

```
int ConnectionError = 3000 + 23;
```

Description

This is ConnectionError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.57 DataStoreErrorCode.ConnectionNotClosed Field

Pascal

```
const ConnectionNotClosed: Integer = 0 + 80;
```

C++

```
int ConnectionNotClosed = 0 + 80;
```

Description

This is ConnectionNotClosed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.58 DataStoreErrorCode.ConnectionPoolClosed Field

Pascal

```
const ConnectionPoolClosed: Integer = 3000 + 119;
```

C++

```
int ConnectionPoolClosed = 3000 + 119;
```

Description

This is ConnectionPoolClosed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.59 DataStoreErrorCode.ConnectionTimeout Field

Pascal

```
const ConnectionTimeout: Integer = 3000 + 118;
```

C++

```
int ConnectionTimeout = 3000 + 118;
```

Description

This is ConnectionTimeout, a member of class DataStoreErrorCode.

15.1.2.1.8.2.60 DataStoreErrorCode.ConstraintNameExists Field

Pascal

```
const ConstraintNameExists: Integer = 1000 + 20;
```

C++

```
int ConstraintNameExists = 1000 + 20;
```

Description

This is ConstraintNameExists, a member of class DataStoreErrorCode.

15.1.2.1.8.2.61 DataStoreErrorCode.CreateErrorMethodExists Field

Pascal

```
const CreateErrorMethodExists: Integer = 2000 + 53;
```

C++

```
int CreateErrorMethodExists = 2000 + 53;
```

Description

This is CreateErrorMethodExists, a member of class DataStoreErrorCode.

15.1.2.1.8.2.62 DataStoreErrorCode.CyclesInViewDefinition Field

Pascal

```
const CyclesInViewDefinition: Integer = 2000 + 69;
```

C++

```
int CyclesInViewDefinition = 2000 + 69;
```

Description

This is CyclesInViewDefinition, a member of class DataStoreErrorCode.

15.1.2.1.8.2.63 DataStoreErrorCode.DataFileLoadFailed Field

Pascal

```
const DataFileLoadFailed: Integer = 0 + 60;
```

C++

```
int DataFileLoadFailed = 0 + 60;
```

Description

This is DataFileLoadFailed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.64 DataStoreErrorCode.DataHasDuplicates Field

Pascal

```
const DataHasDuplicates: Integer = 2000 + 44;
```

C++

```
int DataHasDuplicates = 2000 + 44;
```

Description

This is DataHasDuplicates, a member of class DataStoreErrorCode.

15.1.2.1.8.2.65 DataStoreErrorCode.DatasetAlreadyOpen Field

Pascal

```
const DatasetAlreadyOpen: Integer = 2000 + 7;
```

C++

```
int DatasetAlreadyOpen = 2000 + 7;
```

Description

This is DatasetAlreadyOpen, a member of class DataStoreErrorCode.

15.1.2.1.8.2.66 DataStoreErrorCode.DatasetCorrupt Field

Pascal

```
const DatasetCorrupt: Integer = 0 + 35;
```

C++

```
int DatasetCorrupt = 0 + 35;
```

Description

This is DatasetCorrupt, a member of class DataStoreErrorCode.

15.1.2.1.8.2.67 DataStoreErrorCode.DatasetExists Field

Pascal

```
const DatasetExists: Integer = 2000 + 35;
```

C++

```
int DatasetExists = 2000 + 35;
```

Description

This is DatasetExists, a member of class DataStoreErrorCode.

15.1.2.1.8.2.68 DataStoreErrorCode.DatasetHasNoRows Field

Pascal

```
const DatasetHasNoRows: Integer = 0 + 41;
```

C++

```
int DatasetHasNoRows = 0 + 41;
```

Description

This is DatasetHasNoRows, a member of class DataStoreErrorCode.

15.1.2.1.8.2.69 DataStoreErrorCode.DatasetHasNoTables Field

Pascal

```
const DatasetHasNoTables: Integer = 0 + 18;
```

C++

```
int DatasetHasNoTables = 0 + 18;
```

Description

This is DatasetHasNoTables, a member of class DataStoreErrorCode.

15.1.2.1.8.2.70 DataStoreErrorCode.DatasetNotOpen Field

Pascal

```
const DatasetNotOpen: Integer = 0 + 42;
```

C++

```
int DatasetNotOpen = 0 + 42;
```

Description

This is DatasetNotOpen, a member of class DataStoreErrorCode.

15.1.2.1.8.2.71 DataStoreErrorCode.DatasetOpen Field

Pascal

```
const DatasetOpen: Integer = 0 + 39;
```

C++

```
int DatasetOpen = 0 + 39;
```

Description

This is DatasetOpen, a member of class DataStoreErrorCode.

15.1.2.1.8.2.72 DataStoreErrorCode.DatastoreAlreadyOpen Field

Pascal

```
const DatastoreAlreadyOpen: Integer = 2000 + 30;
```

C++

```
int DatastoreAlreadyOpen = 2000 + 30;
```

Description

This is DatastoreAlreadyOpen, a member of class DataStoreErrorCode.

15.1.2.1.8.2.73 DataStoreErrorCode.DatastoreExists Field

Pascal

```
const DatastoreExists: Integer = 2000 + 17;
```

C++

```
int DatastoreExists = 2000 + 17;
```

Description

This is DatastoreExists, a member of class DataStoreErrorCode.

15.1.2.1.8.2.74 DataStoreErrorCode.DatastoreInvalid Field

Pascal

```
const DatastoreInvalid: Integer = 2000 + 12;
```

C++

```
int DatastoreInvalid = 2000 + 12;
```

Description

This is DatastoreInvalid, a member of class DataStoreErrorCode.

15.1.2.1.8.2.75 DataStoreErrorCode.DatastoreIsUnlicensed Field

Pascal

```
const DatastoreIsUnlicensed: Integer = 2000 + 49;
```

C++

```
int DatastoreIsUnlicensed = 2000 + 49;
```

Description

This is DatastoreIsUnlicensed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.76 DataStoreErrorCode.DatastoreNotFound Field

Pascal

```
const DatastoreNotFound: Integer = 2000 + 11;
```

C++

```
int DatastoreNotFound = 2000 + 11;
```

Description

This is DatastoreNotFound, a member of class DataStoreErrorCode.

15.1.2.1.8.2.77 DataStoreErrorCode.DatastoreNotOpen Field

Pascal

```
const DatastoreNotOpen: Integer = 2000 + 10;
```

C++

```
int DatastoreNotOpen = 2000 + 10;
```

Description

This is DatastoreNotOpen, a member of class DataStoreErrorCode.

15.1.2.1.8.2.78 DataStoreErrorCode.DatastoreOpen Field

Pascal

```
const DatastoreOpen: Integer = 2000 + 19;
```

C++

```
int DatastoreOpen = 2000 + 19;
```

Description

This is DatastoreOpen, a member of class DataStoreErrorCode.

15.1.2.1.8.2.79 DataStoreErrorCode.DbAdminError Field

Pascal

```
const DbAdminError: Integer = 2000 + 70;
```

C++

```
int DbAdminError = 2000 + 70;
```

Description

This is DbAdminError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.80 DataStoreErrorCode.Deadlock Field

Pascal

```
const Deadlock: Integer = 6000 + 28;
```

C++

```
int Deadlock = 6000 + 28;
```

Description

This is Deadlock, a member of class DataStoreErrorCode.

15.1.2.1.8.2.81 DataStoreErrorCode.DeleteDuplicates Field

Pascal

```
const DeleteDuplicates: Integer = 0 + 105;
```

C++

```
int DeleteDuplicates = 0 + 105;
```

Description

This is DeleteDuplicates, a member of class DataStoreErrorCode.

15.1.2.1.8.2.82 DataStoreErrorCode.DeleteFailed Field

Pascal

```
const DeleteFailed: Integer = 5000 + 2;
```

C++

```
int DeleteFailed = 5000 + 2;
```

Description

This is DeleteFailed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.83 DataStoreErrorCode.DeleteNotAllowed Field

Pascal

```
const DeleteNotAllowed: Integer = 1000 + 16;
```

C++

```
int DeleteNotAllowed = 1000 + 16;
```

Description

This is DeleteNotAllowed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.84 DataStoreErrorCode.DesktopLicense Field

Pascal

```
const DesktopLicense: Integer = 2000 + 72;
```

C++

```
int DesktopLicense = 2000 + 72;
```

Description

This is DesktopLicense, a member of class DataStoreErrorCode.

15.1.2.1.8.2.85 DataStoreErrorCode.DifferentDegree Field

Pascal

```
const DifferentDegree: Integer = 3000 + 24;
```

C++

```
int DifferentDegree = 3000 + 24;
```

Description

This is DifferentDegree, a member of class DataStoreErrorCode.

15.1.2.1.8.2.86 DataStoreErrorCode.DifferentSymbol Field

Pascal

```
const DifferentSymbol: Integer = 3000 + 25;
```

C++

```
int DifferentSymbol = 3000 + 25;
```

Description

This is DifferentSymbol, a member of class DataStoreErrorCode.

15.1.2.1.8.2.87 DataStoreErrorCode.DivideByZero Field

Pascal

```
const DivideByZero: Integer = 3000 + 26;
```

C++

```
int DivideByZero = 3000 + 26;
```

Description

This is DivideByZero, a member of class DataStoreErrorCode.

15.1.2.1.8.2.88 DataStoreErrorCode.DriverNotLoadedAtRuntime Field

Pascal

```
const DriverNotLoadedAtRuntime: Integer = 0 + 83;
```

C++

```
int DriverNotLoadedAtRuntime = 0 + 83;
```

Description

This is DriverNotLoadedAtRuntime, a member of class DataStoreErrorCode.

15.1.2.1.8.2.89 DataStoreErrorCode.DriverNotLoadedInDesign Field

Pascal

```
const DriverNotLoadedInDesign: Integer = 0 + 82;
```

C++

```
int DriverNotLoadedInDesign = 0 + 82;
```

Description

This is DriverNotLoadedInDesign, a member of class DataStoreErrorCode.

15.1.2.1.8.2.90 DataStoreErrorCode.DropErrorDependentView Field

Pascal

```
const DropErrorDependentView: Integer = 2000 + 58;
```

C++

```
int DropErrorDependentView = 2000 + 58;
```

Description

This is DropErrorDependentView, a member of class DataStoreErrorCode.

15.1.2.1.8.2.91 DataStoreErrorCode.DropErrorForeignKey Field

Pascal

```
const DropErrorForeignKey: Integer = 2000 + 59;
```

C++

```
int DropErrorForeignKey = 2000 + 59;
```

Description

This is DropErrorForeignKey, a member of class DataStoreErrorCode.

15.1.2.1.8.2.92 DataStoreErrorCode.DropErrorRoleGrant Field

Pascal

```
const DropErrorRoleGrant: Integer = 2000 + 65;
```

C++

```
int DropErrorRoleGrant = 2000 + 65;
```

Description

This is DropErrorRoleGrant, a member of class DataStoreErrorCode.

15.1.2.1.8.2.93 DataStoreErrorCode.DropSchemaError Field

Pascal

```
const DropSchemaError: Integer = 2000 + 60;
```

C++

```
int DropSchemaError = 2000 + 60;
```

Description

This is DropSchemaError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.94 DataStoreErrorCode.DuplicateColumnName Field

Pascal

```
const DuplicateColumnName: Integer = 0 + 36;
```

C++

```
int DuplicateColumnName = 0 + 36;
```

Description

This is DuplicateColumnName, a member of class DataStoreErrorCode.

15.1.2.1.8.2.95 DataStoreErrorCode.DuplicateKey Field

Pascal

```
const DuplicateKey: Integer = 1000 + 17;
```

C++

```
int DuplicateKey = 1000 + 17;
```

Description

This is DuplicateKey, a member of class DataStoreErrorCode.

15.1.2.1.8.2.96 DataStoreErrorCode.DuplicatePrimary Field

Pascal

```
const DuplicatePrimary: Integer = 0 + 110;
```

C++

```
int DuplicatePrimary = 0 + 110;
```

Description

This is DuplicatePrimary, a member of class DataStoreErrorCode.

15.1.2.1.8.2.97 DataStoreErrorCode.EmptyColumnNames Field

Pascal

```
const EmptyColumnNames: Integer = 0 + 20;
```

C++

```
int EmptyColumnNames = 0 + 20;
```

Description

This is EmptyColumnNames, a member of class DataStoreErrorCode.

15.1.2.1.8.2.98 DataStoreErrorCode.EofExpected Field

Pascal

```
const EofExpected: Integer = 3000 + 28;
```

C++

```
int EofExpected = 3000 + 28;
```

Description

This is EofExpected, a member of class DataStoreErrorCode.

15.1.2.1.8.2.99 DataStoreErrorCode.ExceptionChain Field

Pascal

```
const ExceptionChain: Integer = 0 + 47;
```

C++

```
int ExceptionChain = 0 + 47;
```

Description

This is ExceptionChain, a member of class DataStoreErrorCode.

15.1.2.1.8.2.100 DataStoreErrorCode.FailoverError Field

Pascal

```
const FailoverError: Integer = 3000 + 115;
```

C++

```
int FailoverError = 3000 + 115;
```

Description

This is FailoverError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.101 DataStoreErrorCode.FeatureNotSupp Field

Pascal

```
const FeatureNotSupp: Integer = 3000 + 29;
```

C++

```
int FeatureNotSupp = 3000 + 29;
```

Description

This is FeatureNotSupp, a member of class DataStoreErrorCode.

15.1.2.1.8.2.102 DataStoreErrorCode.FieldPostError Field

Pascal

```
const FieldPostError: Integer = 0 + 81;
```

C++

```
int FieldPostError = 0 + 81;
```

Description

This is FieldPostError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.103 DataStoreErrorCode.FileExists Field

Pascal

```
const FileExists: Integer = 2000 + 36;
```

C++

```
int FileExists = 2000 + 36;
```

Description

This is FileExists, a member of class DataStoreErrorCode.

15.1.2.1.8.2.104 DataStoreErrorCode.FileNotFound Field

Pascal

```
const FileNotFound: Integer = StreamNotFound;
```

C++

```
int FileNotFound = StreamNotFound;
```

Description

This is FileNotFound, a member of class DataStoreErrorCode.

15.1.2.1.8.2.105 DataStoreErrorCode.FloatPrecision Field

Pascal

```
const FloatPrecision: Integer = 3000 + 30;
```

C++

```
int FloatPrecision = 3000 + 30;
```

Description

This is FloatPrecision, a member of class DataStoreErrorCode.

15.1.2.1.8.2.106 DataStoreErrorCode.ForeignKeyError Field

Pascal

```
const ForeignKeyError: Integer = 1000 + 19;
```

C++

```
int ForeignKeyError = 1000 + 19;
```

Description

This is ForeignKeyError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.107 DataStoreErrorCode.ForeignKeyViolation Field

Pascal

```
const ForeignKeyViolation: Integer = 1000 + 18;
```

C++

```
int ForeignKeyViolation = 1000 + 18;
```

Description

This is ForeignKeyViolation, a member of class DataStoreErrorCode.

15.1.2.1.8.2.108 DataStoreErrorCode.ForwardOnly Field

Pascal

```
const ForwardOnly: Integer = 3000 + 31;
```

C++

```
int ForwardOnly = 3000 + 31;
```

Description

This is ForwardOnly, a member of class DataStoreErrorCode.

15.1.2.1.8.2.109 DataStoreErrorCode.GenericError Field

Pascal

```
const GenericError: Integer = 0 + 0;
```

C++

```
int GenericError = 0 + 0;
```

Description

This is GenericError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.110 DataStoreErrorCode.GreaterThanMax Field

Pascal

```
const GreaterThanMax: Integer = 1000 + 8;
```

C++

```
int GreaterThanMax = 1000 + 8;
```

Description

This is GreaterThanMax, a member of class DataStoreErrorCode.

15.1.2.1.8.2.111 DataStoreErrorCode.HasResultset Field

Pascal

```
const HasResultset: Integer = 3000 + 34;
```

C++

```
int HasResultset = 3000 + 34;
```

Description

This is HasResultset, a member of class DataStoreErrorCode.

15.1.2.1.8.2.112 DataStoreErrorCode.IllegalCatalog Field

Pascal

```
const IllegalCatalog: Integer = 3000 + 35;
```

C++

```
int IllegalCatalog = 3000 + 35;
```

Description

This is IllegalCatalog, a member of class DataStoreErrorCode.

15.1.2.1.8.2.113 DataStoreErrorCode.IllegalCharacter Field

Pascal

```
const IllegalCharacter: Integer = 3000 + 36;
```

C++

```
int IllegalCharacter = 3000 + 36;
```

Description

This is IllegalCharacter, a member of class DataStoreErrorCode.

15.1.2.1.8.2.114 DataStoreErrorCode.IllegalExprStart Field

Pascal

```
const IllegalExprStart: Integer = 3000 + 37;
```

C++

```
int IllegalExprStart = 3000 + 37;
```

Description

This is IllegalExprStart, a member of class DataStoreErrorCode.

15.1.2.1.8.2.115 DataStoreErrorCode.IllegalMethodCall Field

Pascal

```
const IllegalMethodCall: Integer = 3000 + 107;
```

C++

```
int IllegalMethodCall = 3000 + 107;
```

Description

This is IllegalMethodCall, a member of class DataStoreErrorCode.

15.1.2.1.8.2.116 DataStoreErrorCode.IncompatibleBlockSize Field

Pascal

```
const IncompatibleBlockSize: Integer = 2000 + 46;
```

C++

```
int IncompatibleBlockSize = 2000 + 46;
```

Description

This is IncompatibleBlockSize, a member of class DataStoreErrorCode.

15.1.2.1.8.2.117 DataStoreErrorCode.IncompatibleDataRow Field

Pascal

```
const IncompatibleDataRow: Integer = 0 + 31;
```

C++

```
int IncompatibleDataRow = 0 + 31;
```

Description

This is IncompatibleDataRow, a member of class DataStoreErrorCode.

15.1.2.1.8.2.118 DataStoreErrorCode.IncompatibleTypes Field

Pascal

```
const IncompatibleTypes: Integer = 3000 + 38;
```

C++

```
int IncompatibleTypes = 3000 + 38;
```

Description

This is IncompatibleTypes, a member of class DataStoreErrorCode.

15.1.2.1.8.2.119 DataStoreErrorCode.IncompatibleTypesForOp Field

Pascal

```
const IncompatibleTypesForOp: Integer = 3000 + 39;
```

C++

```
int IncompatibleTypesForOp = 3000 + 39;
```

Description

This is IncompatibleTypesForOp, a member of class DataStoreErrorCode.

15.1.2.1.8.2.120 DataStoreErrorCode.IndexAlreadyExists Field

Pascal

```
const IndexAlreadyExists: Integer = 3000 + 40;
```

C++

```
int IndexAlreadyExists = 3000 + 40;
```

Description

This is IndexAlreadyExists, a member of class DataStoreErrorCode.

15.1.2.1.8.2.121 DataStoreErrorCode.InlineTooSmall Field

Pascal

```
const InlineTooSmall: Integer = 2000 + 42;
```

C++

```
int InlineTooSmall = 2000 + 42;
```

Description

This is InlineTooSmall, a member of class DataStoreErrorCode.

15.1.2.1.8.2.122 DataStoreErrorCode.InsertFailed Field

Pascal

```
const InsertFailed: Integer = 5000 + 1;
```

C++

```
int InsertFailed = 5000 + 1;
```

Description

This is InsertFailed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.123 DataStoreErrorCode.InsertNotAllowed Field

Pascal

```
const InsertNotAllowed: Integer = 1000 + 14;
```

C++

```
int InsertNotAllowed = 1000 + 14;
```

Description

This is InsertNotAllowed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.124 DataStoreErrorCode.InsufficientRowid Field

Pascal

```
const InsufficientRowid: Integer = 0 + 73;
```

C++

```
int InsufficientRowid = 0 + 73;
```

Description

This is InsufficientRowid, a member of class DataStoreErrorCode.

15.1.2.1.8.2.125 DataStoreErrorCode.IntegerRequired Field

Pascal

```
const IntegerRequired: Integer = 3000 + 108;
```

C++

```
int IntegerRequired = 3000 + 108;
```

Description

This is IntegerRequired, a member of class DataStoreErrorCode.

15.1.2.1.8.2.126 DataStoreErrorCode.InternalDuplicateKey Field

Pascal

```
const InternalDuplicateKey: Integer = 2000 + 4;
```

C++

```
int InternalDuplicateKey = 2000 + 4;
```

Description

This is InternalDuplicateKey, a member of class DataStoreErrorCode.

15.1.2.1.8.2.127 DataStoreErrorCode.InternalError Field

Pascal

```
const InternalError: Integer = 3000 + 41;
```

C++

```
int InternalError = 3000 + 41;
```

Description

This is InternalError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.128 DataStoreErrorCode.InternalUpdateFailed Field

Pascal

```
const InternalUpdateFailed: Integer = 2000 + 13;
```

C++

```
int InternalUpdateFailed = 2000 + 13;
```

Description

This is InternalUpdateFailed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.129 DataStoreErrorCode.InvalidAggDescriptor Field

Pascal

```
const InvalidAggDescriptor: Integer = 0 + 46;
```

C++

```
int InvalidAggDescriptor = 0 + 46;
```

Description

This is InvalidAggDescriptor, a member of class DataStoreErrorCode.

15.1.2.1.8.2.130 DataStoreErrorCode.InvalidAggExpr Field

Pascal

```
const InvalidAggExpr: Integer = 3000 + 42;
```

C++

```
int InvalidAggExpr = 3000 + 42;
```

Description

This is InvalidAggExpr, a member of class DataStoreErrorCode.

15.1.2.1.8.2.131 DataStoreErrorCode.InvalidAnchorFile Field

Pascal

```
const InvalidAnchorFile: Integer = 6000 + 7;
```

C++

```
int InvalidAnchorFile = 6000 + 7;
```

Description

This is InvalidAnchorFile, a member of class DataStoreErrorCode.

15.1.2.1.8.2.132 DataStoreErrorCode.InvalidAnchorLength Field

Pascal

```
const InvalidAnchorLength: Integer = 6000 + 6;
```

C++

```
int InvalidAnchorLength = 6000 + 6;
```

Description

This is InvalidAnchorLength, a member of class DataStoreErrorCode.

15.1.2.1.8.2.133 DataStoreErrorCode.InvalidBinaryString Field

Pascal

```
const InvalidBinaryString: Integer = 3000 + 43;
```

C++

```
int InvalidBinaryString = 3000 + 43;
```

Description

This is InvalidBinaryString, a member of class DataStoreErrorCode.

15.1.2.1.8.2.134 DataStoreErrorCode.InvalidClass Field

Pascal

```
const InvalidClass: Integer = 0 + 106;
```

C++

```
int InvalidClass = 0 + 106;
```

Description

This is InvalidClass, a member of class DataStoreErrorCode.

15.1.2.1.8.2.135 DataStoreErrorCode.InvalidColumnFormat Field

Pascal

```
const InvalidColumnFormat: Integer = 0 + 71;
```

C++

```
int InvalidColumnFormat = 0 + 71;
```

Description

This is InvalidColumnFormat, a member of class DataStoreErrorCode.

15.1.2.1.8.2.136 DataStoreErrorCode.InvalidColumnname Field

Pascal

```
const InvalidColumnname: Integer = 3000 + 46;
```

C++

```
int InvalidColumnname = 3000 + 46;
```

Description

This is InvalidColumnname, a member of class DataStoreErrorCode.

15.1.2.1.8.2.137 DataStoreErrorCode.InvalidColumnPosition Field

Pascal

```
const InvalidColumnPosition: Integer = 0 + 12;
```

C++

```
int InvalidColumnPosition = 0 + 12;
```

Description

This is InvalidColumnPosition, a member of class DataStoreErrorCode.

15.1.2.1.8.2.138 DataStoreErrorCode.InvalidColumnRef Field

Pascal

```
const InvalidColumnRef: Integer = 3000 + 45;
```

C++

```
int InvalidColumnRef = 3000 + 45;
```

Description

This is InvalidColumnRef, a member of class DataStoreErrorCode.

15.1.2.1.8.2.139 DataStoreErrorCode.InvalidColumnType Field

Pascal

```
const InvalidColumnType: Integer = 0 + 49;
```

C++

```
int InvalidColumnType = 0 + 49;
```

Description

This is InvalidColumnType, a member of class DataStoreErrorCode.

15.1.2.1.8.2.140 DataStoreErrorCode.InvalidColumnValue Field

Pascal

```
const InvalidColumnValue: Integer = 1000 + 4;
```

C++

```
int InvalidColumnValue = 1000 + 4;
```

Description

This is InvalidColumnValue, a member of class DataStoreErrorCode.

15.1.2.1.8.2.141 DataStoreErrorCode.InvalidConstraintName Field

Pascal

```
const InvalidConstraintName: Integer = 0 + 115;
```

C++

```
int InvalidConstraintName = 0 + 115;
```

Description

This is InvalidConstraintName, a member of class DataStoreErrorCode.

15.1.2.1.8.2.142 DataStoreErrorCode.InvalidDataFileFormat Field

Pascal

```
const InvalidDataFileFormat: Integer = 0 + 32;
```

C++

```
int InvalidDataFileFormat = 0 + 32;
```

Description

This is InvalidDataFileFormat, a member of class DataStoreErrorCode.

15.1.2.1.8.2.143 DataStoreErrorCode.InvalidDirectoryAttributes Field

Pascal

```
const InvalidDirectoryAttributes: Integer = 2000 + 48;
```

C++

```
int InvalidDirectoryAttributes = 2000 + 48;
```

Description

This is InvalidDirectoryAttributes, a member of class DataStoreErrorCode.

15.1.2.1.8.2.144 DataStoreErrorCode.InvalidEscapeCharInLike Field

Pascal

```
const InvalidEscapeCharInLike: Integer = 3000 + 48;
```

C++

```
int InvalidEscapeCharInLike = 3000 + 48;
```

Description

This is InvalidEscapeCharInLike, a member of class DataStoreErrorCode.

15.1.2.1.8.2.145 DataStoreErrorCode.InvalidExtractField Field

Pascal

```
const InvalidExtractField: Integer = 3000 + 49;
```

C++

```
int InvalidExtractField = 3000 + 49;
```

Description

This is InvalidExtractField, a member of class DataStoreErrorCode.

15.1.2.1.8.2.146 DataStoreErrorCode.InvalidFormat Field

Pascal

```
const InvalidFormat: Integer = 1000 + 11;
```

C++

```
int InvalidFormat = 1000 + 11;
```

Description

This is InvalidFormat, a member of class DataStoreErrorCode.

15.1.2.1.8.2.147 DataStoreErrorCode.InvalidIndexname Field

Pascal

```
const InvalidIndexname: Integer = 3000 + 50;
```

C++

```
int InvalidIndexname = 3000 + 50;
```

Description

This is InvalidIndexname, a member of class DataStoreErrorCode.

15.1.2.1.8.2.148 DataStoreErrorCode.InvalidIteratorUse Field

Pascal

```
const InvalidIteratorUse: Integer = 0 + 109;
```

C++

```
int InvalidIteratorUse = 0 + 109;
```

Description

This is InvalidIteratorUse, a member of class DataStoreErrorCode.

15.1.2.1.8.2.149 DataStoreErrorCode.InvalidJoin Field

Pascal

```
const InvalidJoin: Integer = 3000 + 56;
```

C++

```
int InvalidJoin = 3000 + 56;
```

Description

This is InvalidJoin, a member of class DataStoreErrorCode.

15.1.2.1.8.2.150 DataStoreErrorCode.InvalidLikePattern Field

Pascal

```
const InvalidLikePattern: Integer = 3000 + 51;
```

C++

```
int InvalidLikePattern = 3000 + 51;
```

Description

This is InvalidLikePattern, a member of class DataStoreErrorCode.

15.1.2.1.8.2.151 DataStoreErrorCode.InvalidLsn Field

Pascal

```
const InvalidLsn: Integer = 6000 + 15;
```

C++

```
int InvalidLsn = 6000 + 15;
```

Description

This is InvalidLsn, a member of class DataStoreErrorCode.

15.1.2.1.8.2.152 DataStoreErrorCode.InvalidPattern Field

Pascal

```
const InvalidPattern: Integer = 2000 + 45;
```

C++

```
int InvalidPattern = 2000 + 45;
```

Description

This is InvalidPattern, a member of class DataStoreErrorCode.

15.1.2.1.8.2.153 DataStoreErrorCode.InvalidPrecision Field

Pascal

```
const InvalidPrecision: Integer = 1000 + 9;
```

C++

```
int InvalidPrecision = 1000 + 9;
```

Description

This is InvalidPrecision, a member of class DataStoreErrorCode.

15.1.2.1.8.2.154 DataStoreErrorCode.InvalidRowValues Field

Pascal

```
const InvalidRowValues: Integer = 1000 + 5;
```

C++

```
int InvalidRowValues = 1000 + 5;
```

Description

This is InvalidRowValues, a member of class DataStoreErrorCode.

15.1.2.1.8.2.155 DataStoreErrorCode.InvalidSavepoint Field

Pascal

```
const InvalidSavepoint: Integer = 2000 + 52;
```

C++

```
int InvalidSavepoint = 2000 + 52;
```

Description

This is InvalidSavepoint, a member of class DataStoreErrorCode.

15.1.2.1.8.2.156 DataStoreErrorCode.InvalidSchemaFile Field

Pascal

```
const InvalidSchemaFile: Integer = 0 + 50;
```

C++

```
int InvalidSchemaFile = 0 + 50;
```

Description

This is InvalidSchemaFile, a member of class DataStoreErrorCode.

15.1.2.1.8.2.157 DataStoreErrorCode.InvalidSort Field

Pascal

```
const InvalidSort: Integer = 0 + 113;
```

C++

```
int InvalidSort = 0 + 113;
```

Description

This is InvalidSort, a member of class DataStoreErrorCode.

15.1.2.1.8.2.158 DataStoreErrorCode.InvalidSortAsInserted Field

Pascal

```
const InvalidSortAsInserted: Integer = 0 + 112;
```

C++

```
int InvalidSortAsInserted = 0 + 112;
```

Description

This is InvalidSortAsInserted, a member of class DataStoreErrorCode.

15.1.2.1.8.2.159 DataStoreErrorCode.InvalidSortColumn Field

Pascal

```
const InvalidSortColumn: Integer = 0 + 48;
```

C++

```
int InvalidSortColumn = 0 + 48;
```

Description

This is InvalidSortColumn, a member of class DataStoreErrorCode.

15.1.2.1.8.2.160 DataStoreErrorCode.InvalidStoreClass Field

Pascal

```
const InvalidStoreClass: Integer = 0 + 108;
```

C++

```
int InvalidStoreClass = 0 + 108;
```

Description

This is InvalidStoreClass, a member of class DataStoreErrorCode.

15.1.2.1.8.2.161 DataStoreErrorCode.InvalidStoreName Field

Pascal

```
const InvalidStoreName: Integer = 0 + 104;
```

C++

```
int InvalidStoreName = 0 + 104;
```

Description

This is InvalidStoreName, a member of class DataStoreErrorCode.

15.1.2.1.8.2.162 DataStoreErrorCode.InvalidTablename Field

Pascal

```
const InvalidTablename: Integer = 3000 + 53;
```

C++

```
int InvalidTablename = 3000 + 53;
```

Description

This is InvalidTablename, a member of class DataStoreErrorCode.

15.1.2.1.8.2.163 DataStoreErrorCode.InvalidTableRef Field

Pascal

```
const InvalidTableRef: Integer = 3000 + 52;
```

C++

```
int InvalidTableRef = 3000 + 52;
```

Description

This is InvalidTableRef, a member of class DataStoreErrorCode.

15.1.2.1.8.2.164 DataStoreErrorCode.InvalidTrimChar Field

Pascal

```
const InvalidTrimChar: Integer = 3000 + 54;
```

C++

```
int InvalidTrimChar = 3000 + 54;
```

Description

This is InvalidTrimChar, a member of class DataStoreErrorCode.

15.1.2.1.8.2.165 DataStoreErrorCode.InvalidUserName Field

Pascal

```
const InvalidUserName: Integer = 6000 + 11;
```

C++

```
int InvalidUserName = 6000 + 11;
```

Description

This is InvalidUserName, a member of class DataStoreErrorCode.

15.1.2.1.8.2.166 DataStoreErrorCode.InvalidViewDefinition Field

Pascal

```
const InvalidViewDefinition: Integer = 3000 + 55;
```

C++

```
int InvalidViewDefinition = 3000 + 55;
```

Description

This is InvalidViewDefinition, a member of class DataStoreErrorCode.

15.1.2.1.8.2.167 DataStoreErrorCode.IoError Field

Pascal

```
const IoError: Integer = 0 + 79;
```

C++

```
int IoError = 0 + 79;
```

Description

This is IoError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.168 DataStoreErrorCode.IoNetError Field

Pascal

```
const IoNetError: Integer = 3000 + 57;
```

C++

```
int IoNetError = 3000 + 57;
```

Description

This is IoNetError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.169 DataStoreErrorCode.IoProtocolError Field

Pascal

```
const IoProtocolError: Integer = 3000 + 58;
```

C++

```
int IoProtocolError = 3000 + 58;
```

Description

This is IoProtocolError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.170 DataStoreErrorCode.IsInsertRow Field

Pascal

```
const IsInsertRow: Integer = 3000 + 59;
```

C++

```
int IsInsertRow = 3000 + 59;
```

Description

This is IsInsertRow, a member of class DataStoreErrorCode.

15.1.2.1.8.2.171 DataStoreErrorCode.LessThanMin Field

Pascal

```
const LessThanMin: Integer = 1000 + 7;
```

C++

```
int LessThanMin = 1000 + 7;
```

Description

This is LessThanMin, a member of class DataStoreErrorCode.

15.1.2.1.8.2.172 DataStoreErrorCode.LinkColumnsError Field

Pascal

```
const LinkColumnsError: Integer = 0 + 33;
```

C++

```
int LinkColumnsError = 0 + 33;
```

Description

This is LinkColumnsError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.173 DataStoreErrorCode.LinkfieldInUserparameters Field

Pascal

```
const LinkfieldInUserparameters: Integer = 0 + 86;
```

C++

```
int LinkfieldInUserparameters = 0 + 86;
```

Description

This is LinkfieldInUserparameters, a member of class DataStoreErrorCode.

15.1.2.1.8.2.174 DataStoreErrorCode.LoadingNotStarted Field

Pascal

```
const LoadingNotStarted: Integer = 0 + 29;
```

C++

```
int LoadingNotStarted = 0 + 29;
```

Description

This is LoadingNotStarted, a member of class DataStoreErrorCode.

15.1.2.1.8.2.175 DataStoreErrorCode.LockTimeout Field

Pascal

```
const LockTimeout: Integer = 6000 + 24;
```

C++

```
int LockTimeout = 6000 + 24;
```

Description

This is LockTimeout, a member of class DataStoreErrorCode.

15.1.2.1.8.2.176 DataStoreErrorCode.LogdirMismatch Field

Pascal

```
const LogdirMismatch: Integer = 6000 + 4;
```

C++

```
int LogdirMismatch = 6000 + 4;
```

Description

This is LogdirMismatch, a member of class DataStoreErrorCode.

15.1.2.1.8.2.177 DataStoreErrorCode.LogExists Field

Pascal

```
const LogExists: Integer = 6000 + 0;
```

C++

```
int LogExists = 6000 + 0;
```

Description

This is LogExists, a member of class DataStoreErrorCode.

15.1.2.1.8.2.178 DataStoreErrorCode.MalformedDateOrTime Field

Pascal

```
const MalformedDateOrTime: Integer = 3000 + 60;
```

C++

```
int MalformedDateOrTime = 3000 + 60;
```

Description

This is MalformedDateOrTime, a member of class DataStoreErrorCode.

15.1.2.1.8.2.179 DataStoreErrorCode.MalformedDecimal Field

Pascal

```
const MalformedDecimal: Integer = 3000 + 61;
```

C++

```
int MalformedDecimal = 3000 + 61;
```

Description

This is MalformedDecimal, a member of class DataStoreErrorCode.

15.1.2.1.8.2.180 DataStoreErrorCode.MalformedFloating Field

Pascal

```
const MalformedFloating: Integer = 3000 + 62;
```

C++

```
int MalformedFloating = 3000 + 62;
```

Description

This is MalformedFloating, a member of class DataStoreErrorCode.

15.1.2.1.8.2.181 DataStoreErrorCode.MasterDetailViewError Field

Pascal

```
const MasterDetailViewError: Integer = 0 + 34;
```

C++

```
int MasterDetailViewError = 0 + 34;
```

Description

This is MasterDetailViewError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.182 DataStoreErrorCode.MasterNavigationError Field

Pascal

```
const MasterNavigationError: Integer = 0 + 43;
```

C++

```
int MasterNavigationError = 0 + 43;
```

Description

This is MasterNavigationError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.183 DataStoreErrorCode.MaxResolveFailed Field

Pascal

```
const MaxResolveFailed: Integer = 5000 + 4;
```

C++

```
int MaxResolveFailed = 5000 + 4;
```

Description

This is MaxResolveFailed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.184 DataStoreErrorCode.MethodArgMismatch Field

Pascal

```
const MethodArgMismatch: Integer = 3000 + 63;
```

C++

```
int MethodArgMismatch = 3000 + 63;
```

Description

This is MethodArgMismatch, a member of class DataStoreErrorCode.

15.1.2.1.8.2.185 DataStoreErrorCode.MethodClassMissing Field

Pascal

```
const MethodClassMissing: Integer = 2000 + 55;
```

C++

```
int MethodClassMissing = 2000 + 55;
```

Description

This is MethodClassMissing, a member of class DataStoreErrorCode.

15.1.2.1.8.2.186 DataStoreErrorCode.MethodNameMissing Field

Pascal

```
const MethodNameMissing: Integer = 2000 + 56;
```

C++

```
int MethodNameMissing = 2000 + 56;
```

Description

This is MethodNameMissing, a member of class DataStoreErrorCode.

15.1.2.1.8.2.187 DataStoreErrorCode.MethodNonexistent Field

Pascal

```
const MethodNonexistent: Integer = 2000 + 54;
```

C++

```
int MethodNonexistent = 2000 + 54;
```

Description

This is MethodNonexistent, a member of class DataStoreErrorCode.

15.1.2.1.8.2.188 DataStoreErrorCode.MethodNotFound Field

Pascal

```
const MethodNotFound: Integer = 3000 + 64;
```

C++

```
int MethodNotFound = 3000 + 64;
```

Description

This is MethodNotFound, a member of class DataStoreErrorCode.

15.1.2.1.8.2.189 DataStoreErrorCode.MirrorError Field

Pascal

```
const MirrorError: Integer = 2000 + 66;
```

C++

```
int MirrorError = 2000 + 66;
```

Description

This is MirrorError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.190 DataStoreErrorCode.MismatchColumnlistCount Field

Pascal

```
const MismatchColumnlistCount: Integer = 3000 + 66;
```

C++

```
int MismatchColumnlistCount = 3000 + 66;
```

Description

This is MismatchColumnlistCount, a member of class DataStoreErrorCode.

15.1.2.1.8.2.191 DataStoreErrorCode.MismatchedParameterFormat Field

Pascal

```
const MismatchedParameterFormat: Integer = 0 + 5;
```

C++

```
int MismatchedParameterFormat = 0 + 5;
```

Description

This is MismatchedParameterFormat, a member of class DataStoreErrorCode.

15.1.2.1.8.2.192 DataStoreErrorCode.MismatchInsertColumnCount Field

Pascal

```
const MismatchInsertColumnCount: Integer = 3000 + 65;
```

C++

```
int MismatchInsertColumnCount = 3000 + 65;
```

Description

This is MismatchInsertColumnCount, a member of class DataStoreErrorCode.

15.1.2.1.8.2.193 DataStoreErrorCode.MismatchIntoParamCount Field

Pascal

```
const MismatchIntoParamCount: Integer = 3000 + 67;
```

C++

```
int MismatchIntoParamCount = 3000 + 67;
```

Description

This is MismatchIntoParamCount, a member of class DataStoreErrorCode.

15.1.2.1.8.2.194 DataStoreErrorCode.MismatchParamResult Field

Pascal

```
const MismatchParamResult: Integer = 0 + 93;
```

C++

```
int MismatchParamResult = 0 + 93;
```

Description

This is MismatchParamResult, a member of class DataStoreErrorCode.

15.1.2.1.8.2.195 DataStoreErrorCode.MissingCheckpointFile Field

Pascal

```
const MissingCheckpointFile: Integer = 6000 + 17;
```

C++

```
int MissingCheckpointFile = 6000 + 17;
```

Description

This is MissingCheckpointFile, a member of class DataStoreErrorCode.

15.1.2.1.8.2.196 DataStoreErrorCode.MissingLogFile Field

Pascal

```
const MissingLogFile: Integer = 6000 + 16;
```

C++

```
int MissingLogFile = 6000 + 16;
```

Description

This is MissingLogFile, a member of class DataStoreErrorCode.

15.1.2.1.8.2.197 DataStoreErrorCode.MissingMasterDataset Field

Pascal

```
const MissingMasterDataset: Integer = 0 + 65;
```

C++

```
int MissingMasterDataset = 0 + 65;
```

Description

This is MissingMasterDataset, a member of class DataStoreErrorCode.

15.1.2.1.8.2.198 DataStoreErrorCode.MissingReplacestorerow Field

Pascal

```
const MissingReplacestorerow: Integer = 0 + 102;
```

C++

```
int MissingReplacestorerow = 0 + 102;
```

Description

This is MissingReplacestorerow, a member of class DataStoreErrorCode.

15.1.2.1.8.2.199 DataStoreErrorCode.MissingResolver Field

Pascal

```
const MissingResolver: Integer = 0 + 64;
```

C++

```
int MissingResolver = 0 + 64;
```

Description

This is MissingResolver, a member of class DataStoreErrorCode.

15.1.2.1.8.2.200 DataStoreErrorCode.MoreThanOneRow Field

Pascal

```
const MoreThanOneRow: Integer = 3000 + 68;
```

C++

```
int MoreThanOneRow = 3000 + 68;
```

Description

This is MoreThanOneRow, a member of class DataStoreErrorCode.

15.1.2.1.8.2.201 DataStoreErrorCode.MultipleRowsAffected Field

Pascal

```
const MultipleRowsAffected: Integer = 0 + 38;
```

C++

```
int MultipleRowsAffected = 0 + 38;
```

Description

This is MultipleRowsAffected, a member of class DataStoreErrorCode.

15.1.2.1.8.2.202 DataStoreErrorCode.MustBeSingleColumn Field

Pascal

```
const MustBeSingleColumn: Integer = 3000 + 69;
```

C++

```
int MustBeSingleColumn = 3000 + 69;
```

Description

This is MustBeSingleColumn, a member of class DataStoreErrorCode.

15.1.2.1.8.2.203 DataStoreErrorCode.NameNotUnique Field

Pascal

```
const NameNotUnique: Integer = 2000 + 15;
```

C++

```
int NameNotUnique = 2000 + 15;
```

Description

This is NameNotUnique, a member of class DataStoreErrorCode.

15.1.2.1.8.2.204 DataStoreErrorCode.NeedALogdir Field

Pascal

```
const NeedALogdir: Integer = 6000 + 3;
```

C++

```
int NeedALogdir = 6000 + 3;
```

Description

This is NeedALogdir, a member of class DataStoreErrorCode.

15.1.2.1.8.2.205 DataStoreErrorCode.NeedLocateStartOption Field

Pascal

```
const NeedLocateStartOption: Integer = 0 + 24;
```

C++

```
int NeedLocateStartOption = 0 + 24;
```

Description

This is NeedLocateStartOption, a member of class DataStoreErrorCode.

15.1.2.1.8.2.206 DataStoreErrorCode.NeedProcedureprovider Field

Pascal

```
const NeedProcedureprovider: Integer = 0 + 96;
```

C++

```
int NeedProcedureprovider = 0 + 96;
```

Description

This is NeedProcedureprovider, a member of class DataStoreErrorCode.

15.1.2.1.8.2.207 DataStoreErrorCode.NeedQueryprovider Field

Pascal

```
const NeedQueryprovider: Integer = 0 + 95;
```

C++

```
int NeedQueryprovider = 0 + 95;
```

Description

This is NeedQueryprovider, a member of class DataStoreErrorCode.

15.1.2.1.8.2.208 DataStoreErrorCode.NeedsRecalc Field

Pascal

```
const NeedsRecalc: Integer = 0 + 107;
```

C++

```
int NeedsRecalc = 0 + 107;
```

Description

This is NeedsRecalc, a member of class DataStoreErrorCode.

15.1.2.1.8.2.209 DataStoreErrorCode.NeedStoragedataset Field

Pascal

```
const NeedStoragedataset: Integer = 0 + 90;
```

C++

```
int NeedStoragedataset = 0 + 90;
```

Description

This is NeedStoragedataset, a member of class DataStoreErrorCode.

15.1.2.1.8.2.210 DataStoreErrorCode.NegativeSqrtArgument Field

Pascal

```
const NegativeSqrtArgument: Integer = 3000 + 70;
```

C++

```
int NegativeSqrtArgument = 3000 + 70;
```

Description

This is NegativeSqrtArgument, a member of class DataStoreErrorCode.

15.1.2.1.8.2.211 DataStoreErrorCode.NegativeSubstringLength Field

Pascal

```
const NegativeSubstringLength: Integer = 3000 + 71;
```

C++

```
int NegativeSubstringLength = 3000 + 71;
```

Description

This is NegativeSubstringLength, a member of class DataStoreErrorCode.

15.1.2.1.8.2.212 DataStoreErrorCode.NestedAggregators Field

Pascal

```
const NestedAggregators: Integer = 3000 + 72;
```

C++

```
int NestedAggregators = 3000 + 72;
```

Description

This is NestedAggregators, a member of class DataStoreErrorCode.

15.1.2.1.8.2.213 DataStoreErrorCode.NewerVersion Field

Pascal

```
const NewerVersion: Integer = 2000 + 34;
```

C++

```
int NewerVersion = 2000 + 34;
```

Description

This is NewerVersion, a member of class DataStoreErrorCode.

15.1.2.1.8.2.214 DataStoreErrorCode.NewerVersionForOperation Field

Pascal

```
const NewerVersionForOperation: Integer = 2000 + 68;
```

C++

```
int NewerVersionForOperation = 2000 + 68;
```

Description

This is NewerVersionForOperation, a member of class DataStoreErrorCode.

15.1.2.1.8.2.215 DataStoreErrorCode.NoCalcAggFields Field

Pascal

```
const NoCalcAggFields: Integer = 0 + 92;
```

C++

```
int NoCalcAggFields = 0 + 92;
```

Description

This is NoCalcAggFields, a member of class DataStoreErrorCode.

15.1.2.1.8.2.216 DataStoreErrorCode.NoCalcFields Field

Pascal

```
const NoCalcFields: Integer = 0 + 44;
```

C++

```
int NoCalcFields = 0 + 44;
```

Description

This is NoCalcFields, a member of class DataStoreErrorCode.

15.1.2.1.8.2.217 DataStoreErrorCode.NoDatabaseToResolve Field

Pascal

```
const NoDatabaseToResolve: Integer = 0 + 97;
```

C++

```
int NoDatabaseToResolve = 0 + 97;
```

Description

This is NoDatabaseToResolve, a member of class DataStoreErrorCode.

15.1.2.1.8.2.218 DataStoreErrorCode.NodefaultForBinary Field

Pascal

```
const NodefaultForBinary: Integer = 3000 + 78;
```

C++

```
int NodefaultForBinary = 3000 + 78;
```

Description

This is NodefaultForBinary, a member of class DataStoreErrorCode.

15.1.2.1.8.2.219 DataStoreErrorCode.NoGroupingColumn Field

Pascal

```
const NoGroupingColumn: Integer = 3000 + 73;
```

C++

```
int NoGroupingColumn = 3000 + 73;
```

Description

This is NoGroupingColumn, a member of class DataStoreErrorCode.

15.1.2.1.8.2.220 DataStoreErrorCode.NonExistentRowid Field

Pascal

```
const NonExistentRowid: Integer = 0 + 74;
```

C++

```
int NonExistentRowid = 0 + 74;
```

Description

This is NonExistentRowid, a member of class DataStoreErrorCode.

15.1.2.1.8.2.221 DataStoreErrorCode.NoNonBlobColumns Field

Pascal

```
const NoNonBlobColumns: Integer = 0 + 21;
```

C++

```
int NoNonBlobColumns = 0 + 21;
```

Description

This is NoNonBlobColumns, a member of class DataStoreErrorCode.

15.1.2.1.8.2.222 DataStoreErrorCode.NonPrimitiveRequired Field

Pascal

```
const NonPrimitiveRequired: Integer = 3000 + 79;
```

C++

```
int NonPrimitiveRequired = 3000 + 79;
```

Description

This is NonPrimitiveRequired, a member of class DataStoreErrorCode.

15.1.2.1.8.2.223 DataStoreErrorCode.NonTransactional Field

Pascal

```
const NonTransactional: Integer = 3000 + 80;
```

C++

```
int NonTransactional = 3000 + 80;
```

Description

This is NonTransactional, a member of class DataStoreErrorCode.

15.1.2.1.8.2.224 DataStoreErrorCode.NoPrimaryKey Field

Pascal

```
const NoPrimaryKey: Integer = 0 + 111;
```

C++

```
int NoPrimaryKey = 0 + 111;
```

Description

This is NoPrimaryKey, a member of class DataStoreErrorCode.

15.1.2.1.8.2.225 DataStoreErrorCode.NoPriorOriginalRow Field

Pascal

```
const NoPriorOriginalRow: Integer = 0 + 100;
```

C++

```
int NoPriorOriginalRow = 0 + 100;
```

Description

This is NoPriorOriginalRow, a member of class DataStoreErrorCode.

15.1.2.1.8.2.226 DataStoreErrorCode.NoReadonly Field

Pascal

```
const NoReadonly: Integer = 3000 + 74;
```

C++

```
int NoReadonly = 3000 + 74;
```

Description

This is NoReadonly, a member of class DataStoreErrorCode.

15.1.2.1.8.2.227 DataStoreErrorCode.NoResultSet Field

Pascal

```
const NoResultSet: Integer = 0 + 78;
```

C++

```
int NoResultSet = 0 + 78;
```

Description

This is NoResultSet, a member of class DataStoreErrorCode.

15.1.2.1.8.2.228 DataStoreErrorCode.NoResultsetProduced Field

Pascal

```
const NoResultsetProduced: Integer = 3000 + 75;
```

C++

```
int NoResultsetProduced = 3000 + 75;
```

Description

This is NoResultsetProduced, a member of class DataStoreErrorCode.

15.1.2.1.8.2.229 DataStoreErrorCode.NoRowsAffected Field

Pascal

```
const NoRowsAffected: Integer = 0 + 51;
```

C++

```
int NoRowsAffected = 0 + 51;
```

Description

This is NoRowsAffected, a member of class DataStoreErrorCode.

15.1.2.1.8.2.230 DataStoreErrorCode.NoRowsToDelete Field

Pascal

```
const NoRowsToDelete: Integer = 1000 + 13;
```

C++

```
int NoRowsToDelete = 1000 + 13;
```

Description

This is NoRowsToDelete, a member of class DataStoreErrorCode.

15.1.2.1.8.2.231 DataStoreErrorCode.NoServerFound Field

Pascal

```
const NoServerFound: Integer = 3000 + 76;
```

C++

```
int NoServerFound = 3000 + 76;
```

Description

This is NoServerFound, a member of class DataStoreErrorCode.

15.1.2.1.8.2.232 DataStoreErrorCode.NoSuchParameter Field

Pascal

```
const NoSuchParameter: Integer = 3000 + 77;
```

C++

```
int NoSuchParameter = 3000 + 77;
```

Description

This is NoSuchParameter, a member of class DataStoreErrorCode.

15.1.2.1.8.2.233 DataStoreErrorCode.NotAnOutputParameter Field

Pascal

```
const NotAnOutputParameter: Integer = 3000 + 81;
```

C++

```
int NotAnOutputParameter = 3000 + 81;
```

Description

This is NotAnOutputParameter, a member of class DataStoreErrorCode.

15.1.2.1.8.2.234 DataStoreErrorCode.NotDatabaseResolver Field

Pascal

```
const NotDatabaseResolver: Integer = 0 + 67;
```

C++

```
int NotDatabaseResolver = 0 + 67;
```

Description

This is NotDatabaseResolver, a member of class DataStoreErrorCode.

15.1.2.1.8.2.235 DataStoreErrorCode.NotImplemented Field

Pascal

```
const NotImplemented: Integer = 7000 + 3;
```

C++

```
int NotImplemented = 7000 + 3;
```

Description

This is NotImplemented, a member of class DataStoreErrorCode.

15.1.2.1.8.2.236 DataStoreErrorCode.NotInsertRow Field

Pascal

```
const NotInsertRow: Integer = 3000 + 82;
```

C++

```
int NotInsertRow = 3000 + 82;
```

Description

This is NotInsertRow, a member of class DataStoreErrorCode.

15.1.2.1.8.2.237 DataStoreErrorCode.NotSelectQuery Field

Pascal

```
const NotSelectQuery: Integer = 0 + 2;
```

C++

```
int NotSelectQuery = 0 + 2;
```

Description

This is NotSelectQuery, a member of class DataStoreErrorCode.

15.1.2.1.8.2.238 DataStoreErrorCode.NotTransactional Field

Pascal

```
const NotTransactional: Integer = 6000 + 14;
```

C++

```
int NotTransactional = 6000 + 14;
```

Description

This is NotTransactional, a member of class DataStoreErrorCode.

15.1.2.1.8.2.239 DataStoreErrorCode.NotUpdateable Field

Pascal

```
const NotUpdateable: Integer = 0 + 37;
```

C++

```
int NotUpdateable = 0 + 37;
```

Description

This is NotUpdateable, a member of class DataStoreErrorCode.

15.1.2.1.8.2.240 DataStoreErrorCode.NoUpdatableColumns Field

Pascal

```
const NoUpdatableColumns: Integer = 0 + 8;
```

C++

```
int NoUpdatableColumns = 0 + 8;
```

Description

This is NoUpdatableColumns, a member of class DataStoreErrorCode.

15.1.2.1.8.2.241 DataStoreErrorCode.NoWhereClause Field

Pascal

```
const NoWhereClause: Integer = 0 + 40;
```

C++

```
int NoWhereClause = 0 + 40;
```

Description

This is NoWhereClause, a member of class DataStoreErrorCode.

15.1.2.1.8.2.242 DataStoreErrorCode.NowReadonly Field

Pascal

```
const NowReadonly: Integer = 3000 + 84;
```

C++

```
int NowReadonly = 3000 + 84;
```

Description

This is NowReadonly, a member of class DataStoreErrorCode.

15.1.2.1.8.2.243 DataStoreErrorCode.NullColumnName Field

Pascal

```
const NullColumnName: Integer = 0 + 68;
```

C++

```
int NullColumnName = 0 + 68;
```

Description

This is NullColumnName, a member of class DataStoreErrorCode.

15.1.2.1.8.2.244 DataStoreErrorCode.NumericOverflow Field

Pascal

```
const NumericOverflow: Integer = 3000 + 85;
```

C++

```
int NumericOverflow = 3000 + 85;
```

Description

This is NumericOverflow, a member of class DataStoreErrorCode.

15.1.2.1.8.2.245 DataStoreErrorCode.NumericUnderflow Field

Pascal

```
const NumericUnderflow: Integer = 3000 + 86;
```

C++

```
int NumericUnderflow = 3000 + 86;
```

Description

This is NumericUnderflow, a member of class DataStoreErrorCode.

15.1.2.1.8.2.246 DataStoreErrorCode.OlderVersion Field

Pascal

```
const OlderVersion: Integer = 2000 + 41;
```

C++

```
int OlderVersion = 2000 + 41;
```

Description

This is OlderVersion, a member of class DataStoreErrorCode.

15.1.2.1.8.2.247 DataStoreErrorCode.OldLog Field

Pascal

```
const OldLog: Integer = 6000 + 25;
```

C++

```
int OldLog = 6000 + 25;
```

Description

This is OldLog, a member of class DataStoreErrorCode.

15.1.2.1.8.2.248 DataStoreErrorCode.OnepassInputStream Field

Pascal

```
const OnepassInputStream: Integer = 0 + 72;
```

C++

```
int OnepassInputStream = 0 + 72;
```

Description

This is OnepassInputStream, a member of class DataStoreErrorCode.

15.1.2.1.8.2.249 DataStoreErrorCode.OperationCanceled Field

Pascal

```
const OperationCanceled: Integer = 2000 + 25;
```

C++

```
int OperationCanceled = 2000 + 25;
```

Description

This is OperationCanceled, a member of class DataStoreErrorCode.

15.1.2.1.8.2.250 DataStoreErrorCode.OutParameterRequired Field

Pascal

```
const OutParameterRequired: Integer = 3000 + 87;
```

C++

```
int OutParameterRequired = 3000 + 87;
```

Description

This is OutParameterRequired, a member of class DataStoreErrorCode.

15.1.2.1.8.2.251 DataStoreErrorCode.OverloadedDiffParams Field

Pascal

```
const OverloadedDiffParams: Integer = 3000 + 109;
```

C++

```
int OverloadedDiffParams = 3000 + 109;
```

Description

This is OverloadedDiffParams, a member of class DataStoreErrorCode.

15.1.2.1.8.2.252 DataStoreErrorCode.ParameterCountMismatch Field

Pascal

```
const ParameterCountMismatch: Integer = 0 + 6;
```

C++

```
int ParameterCountMismatch = 0 + 6;
```

Description

This is ParameterCountMismatch, a member of class DataStoreErrorCode.

15.1.2.1.8.2.253 DataStoreErrorCode.ParameterValueNotSet Field

Pascal

```
const ParameterValueNotSet: Integer = 3000 + 88;
```

C++

```
int ParameterValueNotSet = 3000 + 88;
```

Description

This is ParameterValueNotSet, a member of class DataStoreErrorCode.

15.1.2.1.8.2.254 DataStoreErrorCode.PartialSearchForString Field

Pascal

```
const PartialSearchForString: Integer = 0 + 23;
```

C++

```
int PartialSearchForString = 0 + 23;
```

Description

This is PartialSearchForString, a member of class DataStoreErrorCode.

15.1.2.1.8.2.255 DataStoreErrorCode.PhysUndoNeeded Field

Pascal

```
const PhysUndoNeeded: Integer = 6000 + 32;
```

C++

```
int PhysUndoNeeded = 6000 + 32;
```

Description

This is PhysUndoNeeded, a member of class DataStoreErrorCode.

15.1.2.1.8.2.256 DataStoreErrorCode.PrivilegeMix Field

Pascal

```
const PrivilegeMix: Integer = 3000 + 89;
```

C++

```
int PrivilegeMix = 3000 + 89;
```

Description

This is PrivilegeMix, a member of class DataStoreErrorCode.

15.1.2.1.8.2.257 DataStoreErrorCode.ProcedureFailed Field

Pascal

```
const ProcedureFailed: Integer = 0 + 103;
```

C++

```
int ProcedureFailed = 0 + 103;
```

Description

This is ProcedureFailed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.258 DataStoreErrorCode.ProcedureInProcess Field

Pascal

```
const ProcedureInProcess: Integer = 0 + 94;
```

C++

```
int ProcedureInProcess = 0 + 94;
```

Description

This is ProcedureInProcess, a member of class DataStoreErrorCode.

15.1.2.1.8.2.259 DataStoreErrorCode.ProtocolVersion Field

Pascal

```
const ProtocolVersion: Integer = 3000 + 90;
```

C++

```
int ProtocolVersion = 3000 + 90;
```

Description

This is ProtocolVersion, a member of class DataStoreErrorCode.

15.1.2.1.8.2.260 DataStoreErrorCode.ProviderFailed Field

Pascal

```
const ProviderFailed: Integer = 0 + 87;
```

C++

```
int ProviderFailed = 0 + 87;
```

Description

This is ProviderFailed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.261 DataStoreErrorCode.ProviderOwned Field

Pascal

```
const ProviderOwned: Integer = 0 + 88;
```

C++

```
int ProviderOwned = 0 + 88;
```

Description

This is ProviderOwned, a member of class DataStoreErrorCode.

15.1.2.1.8.2.262 DataStoreErrorCode.QueryFailed Field

Pascal

```
const QueryFailed: Integer = 0 + 62;
```

C++

```
int QueryFailed = 0 + 62;
```

Description

This is QueryFailed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.263 DataStoreErrorCode.QueryInProcess Field

Pascal

```
const QueryInProcess: Integer = 0 + 26;
```

C++

```
int QueryInProcess = 0 + 26;
```

Description

This is QueryInProcess, a member of class DataStoreErrorCode.

15.1.2.1.8.2.264 DataStoreErrorCode.ReadBlockError Field

Pascal

```
const ReadBlockError: Integer = 2000 + 23;
```

C++

```
int ReadBlockError = 2000 + 23;
```

Description

This is ReadBlockError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.265 DataStoreErrorCode.ReadOnly Field

Pascal

```
const ReadOnly: Integer = 2000 + 40;
```

C++

```
int ReadOnly = 2000 + 40;
```

Description

This is ReadOnly, a member of class DataStoreErrorCode.

15.1.2.1.8.2.266 DataStoreErrorCode.ReadOnlyColumn Field

Pascal

```
const ReadOnlyColumn: Integer = 1000 + 1;
```

C++

```
int ReadOnlyColumn = 1000 + 1;
```

Description

This is ReadOnlyColumn, a member of class DataStoreErrorCode.

15.1.2.1.8.2.267 DataStoreErrorCode.ReadOnlyDataset Field

Pascal

```
const ReadOnlyDataset: Integer = 1000 + 2;
```

C++

```
int ReadOnlyDataset = 1000 + 2;
```

Description

This is ReadOnlyDataset, a member of class DataStoreErrorCode.

15.1.2.1.8.2.268 DataStoreErrorCode.ReadOnlyStore Field

Pascal

```
const ReadOnlyStore: Integer = 0 + 101;
```

C++

```
int ReadOnlyStore = 0 + 101;
```

Description

This is ReadOnlyStore, a member of class DataStoreErrorCode.

15.1.2.1.8.2.269 DataStoreErrorCode.ReadonlyTimeout Field

Pascal

```
const ReadonlyTimeout: Integer = 2000 + 71;
```

C++

```
int ReadonlyTimeout = 2000 + 71;
```

Description

This is ReadonlyTimeout, a member of class DataStoreErrorCode.

15.1.2.1.8.2.270 DataStoreErrorCode.ReadonlytxStartFailed Field

Pascal

```
const ReadonlytxStartFailed: Integer = 6000 + 33;
```

C++

```
int ReadonlytxStartFailed = 6000 + 33;
```

Description

This is ReadonlytxStartFailed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.271 DataStoreErrorCode.RecoveryDenied Field

Pascal

```
const RecoveryDenied: Integer = 6000 + 22;
```

C++

```
int RecoveryDenied = 6000 + 22;
```

Description

This is RecoveryDenied, a member of class DataStoreErrorCode.

15.1.2.1.8.2.272 DataStoreErrorCode.RefreshrowNotSupported Field

Pascal

```
const RefreshrowNotSupported: Integer = 0 + 75;
```

C++

```
int RefreshrowNotSupported = 0 + 75;
```

Description

This is RefreshrowNotSupported, a member of class DataStoreErrorCode.

15.1.2.1.8.2.273 DataStoreErrorCode.RenamedTwice Field

Pascal

```
const RenamedTwice: Integer = 3000 + 92;
```

C++

```
int RenamedTwice = 3000 + 92;
```

Description

This is RenamedTwice, a member of class DataStoreErrorCode.

15.1.2.1.8.2.274 DataStoreErrorCode.ReopenFailure Field

Pascal

```
const ReopenFailure: Integer = 0 + 61;
```

C++

```
int ReopenFailure = 0 + 61;
```

Description

This is ReopenFailure, a member of class DataStoreErrorCode.

15.1.2.1.8.2.275 DataStoreErrorCode.ResolveFailed Field

Pascal

```
const ResolveFailed: Integer = 0 + 63;
```

C++

```
int ResolveFailed = 0 + 63;
```

Description

This is ResolveFailed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.276 DataStoreErrorCode.ResolveInProgress Field

Pascal

```
const ResolveInProgress: Integer = 0 + 69;
```

C++

```
int ResolveInProgress = 0 + 69;
```

Description

This is ResolveInProgress, a member of class DataStoreErrorCode.

15.1.2.1.8.2.277 DataStoreErrorCode.ResolvePartial Field

Pascal

```
const ResolvePartial: Integer = 5000 + 5;
```

C++

```
int ResolvePartial = 5000 + 5;
```

Description

This is ResolvePartial, a member of class DataStoreErrorCode.

15.1.2.1.8.2.278 DataStoreErrorCode.RestructureDataLoss Field

Pascal

```
const RestructureDataLoss: Integer = 2000 + 26;
```

C++

```
int RestructureDataLoss = 2000 + 26;
```

Description

This is RestructureDataLoss, a member of class DataStoreErrorCode.

15.1.2.1.8.2.279 DataStoreErrorCode.RestructureInProgress Field

Pascal

```
const RestructureInProgress: Integer = 0 + 27;
```

C++

```
int RestructureInProgress = 0 + 27;
```

Description

This is RestructureInProgress, a member of class DataStoreErrorCode.

15.1.2.1.8.2.280 DataStoreErrorCode.RestructureParseError Field

Pascal

```
const RestructureParseError: Integer = 2000 + 28;
```

C++

```
int RestructureParseError = 2000 + 28;
```

Description

This is RestructureParseError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.281 DataStoreErrorCode.RestructurePrecisionLoss Field

Pascal

```
const RestructurePrecisionLoss: Integer = 2000 + 27;
```

C++

```
int RestructurePrecisionLoss = 2000 + 27;
```

Description

This is RestructurePrecisionLoss, a member of class DataStoreErrorCode.

15.1.2.1.8.2.282 DataStoreErrorCode.ResultNotUpdateable Field

Pascal

```
const ResultNotUpdateable: Integer = 3000 + 83;
```

C++

```
int ResultNotUpdateable = 3000 + 83;
```

Description

This is ResultNotUpdateable, a member of class DataStoreErrorCode.

15.1.2.1.8.2.283 DataStoreErrorCode.ResultsetClosed Field

Pascal

```
const ResultsetClosed: Integer = 3000 + 93;
```

C++

```
int ResultsetClosed = 3000 + 93;
```

Description

This is ResultsetClosed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.284 DataStoreErrorCode.RevokeErrorDependentPrivilege Field

Pascal

```
const RevokeErrorDependentPrivilege: Integer = 2000 + 63;
```

C++

```
int RevokeErrorDependentPrivilege = 2000 + 63;
```

Description

This is RevokeErrorDependentPrivilege, a member of class DataStoreErrorCode.

15.1.2.1.8.2.285 DataStoreErrorCode.RevokeErrorDependentView Field

Pascal

```
const RevokeErrorDependentView: Integer = 2000 + 61;
```

C++

```
int RevokeErrorDependentView = 2000 + 61;
```

Description

This is RevokeErrorDependentView, a member of class DataStoreErrorCode.

15.1.2.1.8.2.286 DataStoreErrorCode.RevokeErrorForeignKey Field

Pascal

```
const RevokeErrorForeignKey: Integer = 2000 + 62;
```

C++

```
int RevokeErrorForeignKey = 2000 + 62;
```

Description

This is RevokeErrorForeignKey, a member of class DataStoreErrorCode.

15.1.2.1.8.2.287 DataStoreErrorCode.RowLockTimeout Field

Pascal

```
const RowLockTimeout: Integer = 6000 + 26;
```

C++

```
int RowLockTimeout = 6000 + 26;
```

Description

This is RowLockTimeout, a member of class DataStoreErrorCode.

15.1.2.1.8.2.288 DataStoreErrorCode.RowNotFound Field

Pascal

```
const RowNotFound: Integer = 2000 + 38;
```

C++

```
int RowNotFound = 2000 + 38;
```

Description

This is RowNotFound, a member of class DataStoreErrorCode.

15.1.2.1.8.2.289 DataStoreErrorCode.RowTooWide Field

Pascal

```
const RowTooWide: Integer = 2000 + 6;
```

C++

```
int RowTooWide = 2000 + 6;
```

Description

This is RowTooWide, a member of class DataStoreErrorCode.

15.1.2.1.8.2.290 DataStoreErrorCode.RuntimeError Field

Pascal

```
const RuntimeError: Integer = 3000 + 114;
```

C++

```
int RuntimeError = 3000 + 114;
```

Description

This is RuntimeError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.291 DataStoreErrorCode.SecurityError Field

Pascal

```
const SecurityError: Integer = 2000 + 51;
```

C++

```
int SecurityError = 2000 + 51;
```

Description

This is SecurityError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.292 DataStoreErrorCode.SetCalculatedFailure Field

Pascal

```
const SetCalculatedFailure: Integer = 0 + 15;
```

C++

```
int SetCalculatedFailure = 0 + 15;
```

Description

This is SetCalculatedFailure, a member of class DataStoreErrorCode.

15.1.2.1.8.2.293 DataStoreErrorCode.SingleRowHasMoreRows Field

Pascal

```
const SingleRowHasMoreRows: Integer = 3000 + 94;
```

C++

```
int SingleRowHasMoreRows = 3000 + 94;
```

Description

This is SingleRowHasMoreRows, a member of class DataStoreErrorCode.

15.1.2.1.8.2.294 DataStoreErrorCode.SqlError Field

Pascal

```
const SqlError: Integer = 0 + 66;
```

C++

```
int SqlError = 0 + 66;
```

Description

This is SqlError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.295 DataStoreErrorCode.StatementClosed Field

Pascal

```
const StatementClosed: Integer = 3000 + 95;
```

C++

```
int StatementClosed = 3000 + 95;
```

Description

This is StatementClosed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.296 DataStoreErrorCode.StoreNameNotSet Field

Pascal

```
const StoreNameNotSet: Integer = 2000 + 1;
```

C++

```
int StoreNameNotSet = 2000 + 1;
```

Description

This is StoreNameNotSet, a member of class DataStoreErrorCode.

15.1.2.1.8.2.297 DataStoreErrorCode.StoreOperationUnsupported Field

Pascal

```
const StoreOperationUnsupported: Integer = 2000 + 16;
```

C++

```
int StoreOperationUnsupported = 2000 + 16;
```

Description

This is StoreOperationUnsupported, a member of class DataStoreErrorCode.

15.1.2.1.8.2.298 DataStoreErrorCode.StreamClosed Field

Pascal

```
const StreamClosed: Integer = 2000 + 31;
```

C++

```
int StreamClosed = 2000 + 31;
```

Description

This is StreamClosed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.299 DataStoreErrorCode.StreamNotFound Field

Pascal

```
const StreamNotFound: Integer = 2000 + 33;
```

C++

```
int StreamNotFound = 2000 + 33;
```

Description

This is StreamNotFound, a member of class DataStoreErrorCode.

15.1.2.1.8.2.300 DataStoreErrorCode.StreamOpenTwice Field

Pascal

```
const StreamOpenTwice: Integer = 2000 + 14;
```

C++

```
int StreamOpenTwice = 2000 + 14;
```

Description

This is StreamOpenTwice, a member of class DataStoreErrorCode.

15.1.2.1.8.2.301 DataStoreErrorCode.StringTooLong Field

Pascal

```
const StringTooLong: Integer = 3000 + 96;
```

C++

```
int StringTooLong = 3000 + 96;
```

Description

This is StringTooLong, a member of class DataStoreErrorCode.

15.1.2.1.8.2.302 DataStoreErrorCode.SyntaxError Field

Pascal

```
const SyntaxError: Integer = 3000 + 113;
```

C++

```
int SyntaxError = 3000 + 113;
```

Description

This is SyntaxError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.303 DataStoreErrorCode.TableAlreadyExists Field

Pascal

```
const TableAlreadyExists: Integer = 3000 + 97;
```

C++

```
int TableAlreadyExists = 3000 + 97;
```

Description

This is TableAlreadyExists, a member of class DataStoreErrorCode.

15.1.2.1.8.2.304 DataStoreErrorCode.TableNonexistent Field

Pascal

```
const TableNonexistent: Integer = 2000 + 64;
```

C++

```
int TableNonexistent = 2000 + 64;
```

Description

This is TableNonexistent, a member of class DataStoreErrorCode.

15.1.2.1.8.2.305 DataStoreErrorCode.TooManyAutoIncrements Field

Pascal

```
const TooManyAutoIncrements: Integer = 0 + 114;
```

C++

```
int TooManyAutoIncrements = 0 + 114;
```

Description

This is TooManyAutoIncrements, a member of class DataStoreErrorCode.

15.1.2.1.8.2.306 DataStoreErrorCode.TooManyErrors Field

Pascal

```
const TooManyErrors: Integer = 2000 + 39;
```

C++

```
int TooManyErrors = 2000 + 39;
```

Description

This is TooManyErrors, a member of class DataStoreErrorCode.

15.1.2.1.8.2.307 DataStoreErrorCode.TooManyOpenFiles Field

Pascal

```
const TooManyOpenFiles: Integer = 6000 + 10;
```

C++

```
int TooManyOpenFiles = 6000 + 10;
```

Description

This is TooManyOpenFiles, a member of class DataStoreErrorCode.

15.1.2.1.8.2.308 DataStoreErrorCode.Toomanyspecs Field

Pascal

```
const Toomanyspecs: Integer = 3000 + 98;
```

C++

```
int Toomanyspecs = 3000 + 98;
```

Description

This is Toomanyspecs, a member of class DataStoreErrorCode.

15.1.2.1.8.2.309 DataStoreErrorCode.TransactionIsolationLevelNotSupported Field

Pascal

```
const TransactionIsolationLevelNotSupported: Integer = 0 + 22;
```

C++

```
int TransactionIsolationLevelNotSupported = 0 + 22;
```

Description

This is TransactionIsolationLevelNotSupported, a member of class DataStoreErrorCode.

15.1.2.1.8.2.310 DataStoreErrorCode.TxAlreadyStarted Field

Pascal

```
const TxAlreadyStarted: Integer = 6000 + 30;
```

C++

```
int TxAlreadyStarted = 6000 + 30;
```

Description

This is TxAlreadyStarted, a member of class DataStoreErrorCode.

15.1.2.1.8.2.311 DataStoreErrorCode.TxManagerInuse Field

Pascal

```
const TxManagerInuse: Integer = 6000 + 18;
```

C++

```
int TxManagerInuse = 6000 + 18;
```

Description

This is TxManagerInuse, a member of class DataStoreErrorCode.

15.1.2.1.8.2.312 DataStoreErrorCode.TxNotAllowed Field

Pascal

```
const TxNotAllowed: Integer = 6000 + 21;
```

C++

```
int TxNotAllowed = 6000 + 21;
```

Description

This is TxNotAllowed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.313 DataStoreErrorCode.TxRequired Field

Pascal

```
const TxRequired: Integer = 6000 + 20;
```

C++

```
int TxRequired = 6000 + 20;
```

Description

This is TxRequired, a member of class DataStoreErrorCode.

15.1.2.1.8.2.314 DataStoreErrorCode.TypenameExpected Field

Pascal

```
const TypenameExpected: Integer = 3000 + 99;
```

C++

```
int TypenameExpected = 3000 + 99;
```

Description

This is TypenameExpected, a member of class DataStoreErrorCode.

15.1.2.1.8.2.315 DataStoreErrorCode.UnclosedComment Field

Pascal

```
const UnclosedComment: Integer = 3000 + 100;
```

C++

```
int UnclosedComment = 3000 + 100;
```

Description

This is UnclosedComment, a member of class DataStoreErrorCode.

15.1.2.1.8.2.316 DataStoreErrorCode.UnclosedIdentifier Field

Pascal

```
const UnclosedIdentifier: Integer = 3000 + 101;
```

C++

```
int UnclosedIdentifier = 3000 + 101;
```

Description

This is UnclosedIdentifier, a member of class DataStoreErrorCode.

15.1.2.1.8.2.317 DataStoreErrorCode.UnclosedStringLiteral Field

Pascal

```
const UnclosedStringLiteral: Integer = 3000 + 102;
```

C++

```
int UnclosedStringLiteral = 3000 + 102;
```

Description

This is UnclosedStringLiteral, a member of class DataStoreErrorCode.

15.1.2.1.8.2.318 DataStoreErrorCode.UnexpectedEndOfQuery Field

Pascal

```
const UnexpectedEndOfQuery: Integer = 0 + 3;
```

C++

```
int UnexpectedEndOfQuery = 0 + 3;
```

Description

This is UnexpectedEndOfQuery, a member of class DataStoreErrorCode.

15.1.2.1.8.2.319 DataStoreErrorCode.UnknownColumn Field

Pascal

```
const UnknownColumn: Integer = 3000 + 103;
```

C++

```
int UnknownColumn = 3000 + 103;
```

Description

This is UnknownColumn, a member of class DataStoreErrorCode.

15.1.2.1.8.2.320 DataStoreErrorCode.UnknownColumnName Field

Pascal

```
const UnknownColumnName: Integer = 0 + 10;
```

C++

```
int UnknownColumnName = 0 + 10;
```

Description

This is UnknownColumnName, a member of class DataStoreErrorCode.

15.1.2.1.8.2.321 DataStoreErrorCode.UnknownDetailName Field

Pascal

```
const UnknownDetailName: Integer = 0 + 98;
```

C++

```
int UnknownDetailName = 0 + 98;
```

Description

This is UnknownDetailName, a member of class DataStoreErrorCode.

15.1.2.1.8.2.322 DataStoreErrorCode.UnknownParameter Field

Pascal

```
const UnknownParameter: Integer = 3000 + 117;
```

C++

```
int UnknownParameter = 3000 + 117;
```

Description

This is UnknownParameter, a member of class DataStoreErrorCode.

15.1.2.1.8.2.323 DataStoreErrorCode.UnknownParamName Field

Pascal

```
const UnknownParamName: Integer = 0 + 4;
```

C++

```
int UnknownParamName = 0 + 4;
```

Description

This is UnknownParamName, a member of class DataStoreErrorCode.

15.1.2.1.8.2.324 DataStoreErrorCode.UnrecognizedDataType Field

Pascal

```
const UnrecognizedDataType: Integer = 0 + 16;
```

C++

```
int UnrecognizedDataType = 0 + 16;
```

Description

This is UnrecognizedDataType, a member of class DataStoreErrorCode.

15.1.2.1.8.2.325 DataStoreErrorCode.UnrecognizedEscape Field

Pascal

```
const UnrecognizedEscape: Integer = 3000 + 104;
```

C++

```
int UnrecognizedEscape = 3000 + 104;
```

Description

This is UnrecognizedEscape, a member of class DataStoreErrorCode.

15.1.2.1.8.2.326 DataStoreErrorCode.UnsupportedColumntype Field

Pascal

```
const UnsupportedColumntype: Integer = 7000 + 1;
```

C++

```
int UnsupportedColumntype = 7000 + 1;
```

Description

This is UnsupportedColumntype, a member of class DataStoreErrorCode.

15.1.2.1.8.2.327 DataStoreErrorCode.UnsupportedDbxType Field

Pascal

```
const UnsupportedDbxType: Integer = 3000 + 120;
```

C++

```
int UnsupportedDbxType = 3000 + 120;
```

Description

This is UnsupportedDbxType, a member of class DataStoreErrorCode.

15.1.2.1.8.2.328 DataStoreErrorCode.UpdateFailed Field

Pascal

```
const UpdateFailed: Integer = 5000 + 3;
```

C++

```
int UpdateFailed = 5000 + 3;
```

Description

This is UpdateFailed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.329 DataStoreErrorCode.UpdateNotAllowed Field

Pascal

```
const UpdateNotAllowed: Integer = 1000 + 15;
```

C++

```
int UpdateNotAllowed = 1000 + 15;
```

Description

This is UpdateNotAllowed, a member of class DataStoreErrorCode.

15.1.2.1.8.2.330 DataStoreErrorCode.UrlNotFound Field

Pascal

```
const UrlNotFound: Integer = 0 + 84;
```

C++

```
int UrlNotFound = 0 + 84;
```

Description

This is UrlNotFound, a member of class DataStoreErrorCode.

15.1.2.1.8.2.331 DataStoreErrorCode.UrlNotFoundInDesign Field

Pascal

```
const UrlNotFoundInDesign: Integer = 0 + 85;
```

C++

```
int UrlNotFoundInDesign = 0 + 85;
```

Description

This is UrlNotFoundInDesign, a member of class DataStoreErrorCode.

15.1.2.1.8.2.332 DataStoreErrorCode.ViewNotUpdatable Field

Pascal

```
const ViewNotUpdatable: Integer = 3000 + 105;
```

C++

```
int ViewNotUpdatable = 3000 + 105;
```

Description

This is ViewNotUpdatable, a member of class DataStoreErrorCode.

15.1.2.1.8.2.333 DataStoreErrorCode.WithCheckOptionViolation Field

Pascal

```
const WithCheckOptionViolation: Integer = 3000 + 106;
```

C++

```
int WithCheckOptionViolation = 3000 + 106;
```

Description

This is WithCheckOptionViolation, a member of class DataStoreErrorCode.

15.1.2.1.8.2.334 DataStoreErrorCode.WriteBlockError Field

Pascal

```
const WriteBlockError: Integer = 2000 + 22;
```

C++

```
int WriteBlockError = 2000 + 22;
```

Description

This is WriteBlockError, a member of class DataStoreErrorCode.

15.1.2.1.8.2.335 DataStoreErrorCode.WrongCreateschemaname Field

Pascal

```
const WrongCreateschemaname: Integer = 3000 + 111;
```

C++

```
int WrongCreateschemaname = 3000 + 111;
```

Description

This is WrongCreateschemaname, a member of class DataStoreErrorCode.

15.1.2.1.8.2.336 DataStoreErrorCode.WrongDatabase Field

Pascal

```
const WrongDatabase: Integer = 0 + 91;
```

C++

```
int WrongDatabase = 0 + 91;
```

Description

This is WrongDatabase, a member of class DataStoreErrorCode.

15.1.2.1.8.2.337 DataStoreErrorCode.WrongSchemaname Field

Pascal

```
const WrongSchemaname: Integer = 3000 + 110;
```

C++

```
int WrongSchemaname = 3000 + 110;
```

Description

This is WrongSchemaname, a member of class DataStoreErrorCode.

15.1.2.1.8.2.338 DataStoreErrorCode.WrongTableType Field

Pascal

```
const WrongTableType: Integer = 3000 + 112;
```

C++

```
int WrongTableType = 3000 + 112;
```

Description

This is WrongTableType, a member of class DataStoreErrorCode.

15.1.2.1.8.2.339 DataStoreErrorCode.XaActive Field

Pascal

```
const XaActive: Integer = 6000 + 27;
```

C++

```
int XaActive = 6000 + 27;
```

Description

This is XaActive, a member of class DataStoreErrorCode.

15.1.2.1.9 DataStoreParameter Class

Specifies parameter for DataStoreCommand (see page 1912).

Class Hierarchy

Pascal

```
DataStoreParameter = class(DbParameter, IDbDataParameter, ICloneable);
```

C++

```
class DataStoreParameter : public DbParameter, public IDbDataParameter, public ICloneable;
```

File

Ado.Net20.cs

Description

DataStoreParameter specifies a parameter for a DataStoreCommand (see page 1912). Parameter names are case sensitive.

DataStoreParameter is an implementation of the .NET DbParameter class.

See Also

DbParameter (MSDN)

DataStoreCommand (see page 1912)

15.1.2.1.9.1 DataStoreParameter Members

The following tables list the members exposed by DataStoreParameter.

Internal Properties

	Name	Description
	WasPrepared (see page 2041)	This is WasPrepared, a member of class DataStoreParameter.

Public Methods

	Name	Description
	Clone (see page 2036)	Creates a new object that is a copy of the current instance.
	DataStoreParameter (see page 2034)	This is the overview for the DataStoreParameter constructor overload.
	ResetDbType (see page 2036)	Resets DbType (see page 2037) property to original settings.

Public Properties

	Name	Description
	DbType (see page 2037)	Gets and sets expected data type of parameter.
	Direction (see page 2037)	Gets or sets direction of parameter.
	IsNullable (see page 2037)	Indicates whether parameter accepts null values.
	JDSType (see page 2038)	Gets and sets expected JDSType of parameter.
	ParameterName (see page 2038)	Gets or sets parameter name.
	Precision (see page 2038)	Gets or sets maximum number of digits used to represent Value (see page 2041) property.
	Scale (see page 2039)	Gets and sets scale of Value (see page 2041) property, which contains either numeric or decimal value.
	Size (see page 2039)	Gets and sets the size of the parameter value in bytes.
	SourceColumn (see page 2039)	The name of the source column mapped to the DataSet. The default is an empty string.
	SourceColumnNullMapping (see page 2040)	Sets or gets a value indicating whether the column can be null.
	SourceVersion (see page 2040)	Gets or sets the DataRowVersion to use when loading Value (see page 2041).
	Value (see page 2041)	Gets or sets value of parameter.

15.1.2.1.9.2 DataStoreParameter Constructor

This is the overview for the DataStoreParameter constructor overload.

Overload List

	Name	Description
	DataStoreParameter.DataStoreParameter () (see page 2035)	Creates a new instance of a DataStoreCommand (see page 1912) class.
	DataStoreParameter.DataStoreParameter (string, DbType) (see page 2035)	Creates a new instance of a DataStoreCommand (see page 1912) class.

	DataStoreParameter.DataStoreParameter (string, DbType, ParameterDirection, string, DataRowVersion) (see page 2035)	Creates a new instance of a DataStoreCommand (see page 1912) class.
	DataStoreParameter.DataStoreParameter (string, JDSType) (see page 2035)	Creates a new instance of a DataStoreCommand (see page 1912) class.
	DataStoreParameter.DataStoreParameter (string, JDSType, ParameterDirection, string, DataRowVersion) (see page 2035)	Creates a new instance of a DataStoreCommand (see page 1912) class.

15.1.2.1.9.2.1 DataStoreParameter.DataStoreParameter Constructor ()

Creates a new instance of a DataStoreCommand (see page 1912) class.

Pascal

```
constructor Create;
constructor Create(parameterName: string; type: DbType);
constructor Create(parameterName: string; type: JDSType);
constructor Create(parameterName: string; type: DbType; paramDirection: ParameterDirection;
sourceColumn: string; sourceVersion: DataRowVersion);
constructor Create(parameterName: string; type: JDSType; paramDirection: ParameterDirection;
sourceColumn: string; sourceVersion: DataRowVersion);
```

C++

```
DataStoreParameter();
DataStoreParameter(string parameterName, DbType type);
DataStoreParameter(string parameterName, JDSType type);
DataStoreParameter(string parameterName, DbType type, ParameterDirection paramDirection,
string sourceColumn, DataRowVersion sourceVersion);
DataStoreParameter(string parameterName, JDSType type, ParameterDirection paramDirection,
string sourceColumn, DataRowVersion sourceVersion);
```

Description

Creates a new instance of a DataStoreCommand (see page 1912) class, which is an implementation of the .NET DbParameter class.

There are several forms of this constructor:

- Creates a new instance of a DataStoreCommand (see page 1912) class.
- Creates a new instance of a DataStoreCommand (see page 1912) class with a given name and expected data type. The parameter **parameterName** is the name of this parameter. The parameter **type** is the type of the parameter.
- Creates a new instance of a DataStoreCommand (see page 1912) class with a given name and expected data type. The parameter **parameterName** is the name of this parameter. The parameter **type** is the JDSType (see page 2038) of the parameter.
- Creates a new instance of a DataStoreCommand (see page 1912) class with a given name, expected data type, parameter direction, source column name, and source version. The parameter **parameterName** is the name of this parameter. The parameter **type** is the type of this parameter. The parameter **paramDirection** is the direction: Input, Output, InputOutput, ReturnValue. The parameter **sourceColumn** is the name of the source column that is mapped to a DataSet to get and set the value. The parameter **sourceVersion** is the version of the value to get from the DataSet.
- Creates a new instance of a DataStoreCommand (see page 1912) class with a given name, expected data type, parameter direction, source column name, and source version. The parameter **parameterName** is the name of this parameter. The parameter **type** is the JDSType (see page 2038) of this parameter. The parameter **paramDirection** is the direction: Input, Output, InputOutput, ReturnValue. The parameter **sourceColumn** is the name of the source column that is mapped to a DataSet to get and set the value. The parameter **sourceVersion** is specified the version of the value to get from the DataSet.

See Also

DbParameter constructor (MSDN)

DbParameter class (MSDN)

15.1.2.1.9.3 DataStoreParameter Methods

The methods of the DataStoreParameter class are listed here.

Public Methods

	Name	Description
	Clone (see page 2036)	Creates a new object that is a copy of the current instance.
	ResetDbType (see page 2036)	Resets DbType (see page 2037) property to original settings.

15.1.2.1.9.3.1 DataStoreParameter.Clone Method

Creates a new object that is a copy of the current instance.

Pascal

```
function Clone: DataStoreParameter;
```

C++

```
DataStoreParameter Clone();
```

Description

Clone creates a new object that is a copy of the current instance.

See Also

Clone (MSDN)

15.1.2.1.9.3.2 DataStoreParameter.ResetDbType Method

Resets DbType (see page 2037) property to original settings.

Pascal

```
function ResetDbType: void;
```

C++

```
void ResetDbType();
```

Description

ResetDbType resets the DbType (see page 2037) property to its original settings.

See Also

ResetDbType (MSDN)

DbType (see page 2037)

15.1.2.1.9.4 DataStoreParameter Properties

The properties of the DataStoreParameter class are listed here.

Internal Properties

	Name	Description
	WasPrepared (see page 2041)	This is WasPrepared, a member of class DataStoreParameter.

Public Properties

	Name	Description
	DbType (see page 2037)	Gets and sets expected data type of parameter.

	Direction (see page 2037)	Gets or sets direction of parameter.
	IsNullable (see page 2037)	Indicates whether parameter accepts null values.
	JDSType (see page 2038)	Gets and sets expected JDSType of parameter.
	ParameterName (see page 2038)	Gets or sets parameter name.
	Precision (see page 2038)	Gets or sets maximum number of digits used to represent Value (see page 2041) property.
	Scale (see page 2039)	Gets and sets scale of Value (see page 2041) property, which contains either numeric or decimal value.
	Size (see page 2039)	Gets and sets the size of the parameter value in bytes.
	SourceColumn (see page 2039)	The name of the source column mapped to the DataSet. The default is an empty string.
	SourceColumnNullMapping (see page 2040)	Sets or gets a value indicating whether the column can be null.
	SourceVersion (see page 2040)	Gets or sets the DataRowVersion to use when loading Value (see page 2041).
	Value (see page 2041)	Gets or sets value of parameter.

15.1.2.1.9.4.1 DataStoreParameter.DbType Property

Gets and sets expected data type of parameter.

Pascal

```
property DbType: DbType;
```

C++

```
__property DbType DbType;
```

Description

DbType gets and sets the expected data type of the parameter.

See Also

DbType (MSDN)

JDSType (see page 2038)

15.1.2.1.9.4.2 DataStoreParameter.Direction Property

Gets or sets direction of parameter.

Pascal

```
property Direction: ParameterDirection;
```

C++

```
__property ParameterDirection Direction;
```

Description

Direction gets or sets the direction of this parameter.

See Also

Direction (MSDN)

15.1.2.1.9.4.3 DataStoreParameter.IsNullable Property

Indicates whether parameter accepts null values.

Pascal

```
property IsNullable: Boolean;
```

C++

```
__property Boolean IsNullable;
```

Description

IsNullable gets or sets a value that indicates whether the parameter accepts null values.

See Also

IsNullable (MSDN)

15.1.2.1.9.4.4 DataStoreParameter.JDSType Property

Gets and sets expected JDSType of parameter.

Pascal

```
property JDSType: JDSType;
```

C++

```
__property JDSType JDSType;
```

Description

JDSType gets and sets the expected JDSType of the parameter.

See Also

DbType (MSDN)

DbType (see page 2037)

15.1.2.1.9.4.5 DataStoreParameter.ParameterName Property

Gets or sets parameter name.

Pascal

```
property ParameterName: String;
```

C++

```
__property AnsiString ParameterName;
```

Description

ParameterName gets or sets the name of this parameter.

See Also

ParameterName (MSDN)

15.1.2.1.9.4.6 DataStoreParameter.Precision Property

Gets or sets maximum number of digits used to represent Value (see page 2041) property.

Pascal

```
property Precision: byte;
```

C++

```
__property byte Precision;
```

Description

Precision gets or sets the maximum number of digits used to represent the Value (see page 2041) property. The default is 0, which indicates that the data provider sets the precision for the Value (see page 2041) property.

See Also

Value (see page 2041)

15.1.2.1.9.4.7 DataStoreParameter.Scale Property

Gets and sets scale of Value (see page 2041) property, which contains either numeric or decimal value.

Pascal

```
property Scale: byte;
```

C++

```
__property byte Scale;
```

Description

Scale gets and sets the number of digits to the right of the decimal for the numeric or decimal value of the Value (see page 2041) property. The default is 0.

See Also

Value (see page 2041)

15.1.2.1.9.4.8 DataStoreParameter.Size Property

Gets and sets the size of the parameter value in bytes.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

Size is used for binary and string types. Size is ignored for fixed length data types. For Unicode, Size is the number of characters, not bytes. Size does not include any string terminating characters.

Size and DbType (see page 2037) can be inferred if Value (see page 2041) is set, but they are not explicitly set by setting Value (see page 2041).

See Also

Size (MSDN)

DbType (see page 1860)

Value (see page 1865)

15.1.2.1.9.4.9 DataStoreParameter.SourceColumn Property

The name of the source column mapped to the DataSet. The default is an empty string.

Pascal

```
property SourceColumn: String;
```

C++

```
__property AnsiString SourceColumn;
```

Description

Use this property and the SourceColumn property when you are submitting pending changes to the database and you want to validate the version of the data in the target row. To return a value from a particular column in the database, set the SourceColumn property to the name of the column. If the Direction (see page 2037) property is set to *Input*, the value is taken from the DataSet instead of the database. If the Direction (see page 2037) property is set to *Output*, the value is taken from the database.

See Also

SourceColumn (MSDN)

Direction (see page 2037)

15.1.2.1.9.4.10 DataStoreParameter.SourceColumnNullMapping Property

Sets or gets a value indicating whether the column can be null.

Pascal

```
property SourceColumnNullMapping: Boolean;
```

C++

```
__property Boolean SourceColumnNullMapping;
```

Description

SourceColumnNullMapping sets or gets a value indicating whether the column can be set to null. This value allows DataStoreCommandBuilder to generate correct Update statements for nullable columns.

See Also

SourceColumnNullMapping (MSDN)

DataStoreCommandBuilder

15.1.2.1.9.4.11 DataStoreParameter.SourceVersion Property

Gets or sets the DataRowVersion to use when loading Value (see page 2041).

Pascal

```
property SourceVersion: DataRowVersion;
```

C++

```
__property DataRowVersion SourceVersion;
```

Description

Use this property and the SourceColumn (see page 2039) property when you are submitting pending changes to the database and you want to validate the version of the data in the target row. Used by UpdateCommand during an Update operation to determine whether the parameter value is set to **Current** or **Original**. This allows primary keys to be updated. This property is ignored by the InsertCommand and DeleteCommand properties. This property is set to the version of the DataRow used by the Item property or the GetChildRows method of the DataRow object.

The following table lists the allowable SourceVersion values, which are DataRowVersion values.

Member	Description
Current	Current value of the column. Default.
Default	Uses the DefaultValue of the column.
Original	Original value of the column.
Proposed	Uses a Proposed value.

See Also

SourceVersion (MSDN)

Item (MSDN)

GetChildRows (MSDN)

DataRow (MSDN)

DataRowVersion enumeration (MSDN)

SourceColumn (see page 2039)

UpdateCommand (see page 1938)

Value (see page 2041)

Update

15.1.2.1.9.4.12 DataStoreParameter.Value Property

Gets or sets value of parameter.

Pascal

```
property Value: object;
```

C++

```
__property object Value;
```

Description

Value gets or sets the value of this parameter. Null values are accepted as either null or DBNull. If the parameter's Direction (see page 2037) is **Output** or **ReturnValue**, any null values received are DBNull values. The input type is inferred from the value itself. Binary data can be accepted as a byte[] or as an instance of a System.IO.Stream class. Binary output will be a System.IO.Stream instance.

See Also

Value (MSDN)

DBNull (MSDN)

Direction (see page 2037)

15.1.2.1.9.4.13 DataStoreParameter.WasPrepared Property

Pascal

```
property WasPrepared: bool;
```

C++

```
__property bool WasPrepared;
```

Description

This is WasPrepared, a member of class DataStoreParameter.

15.1.2.1.10 DataStoreParameterCollection Class

Collection of parameters for DataStoreCommand (see page 1912).

Class Hierarchy

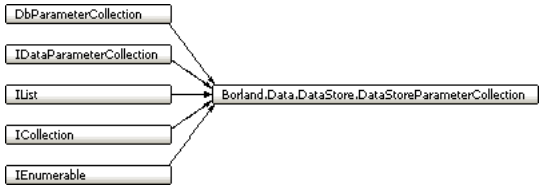

Pascal

```
DataStoreParameterCollection = class(DbParameterCollection, IDataParameterCollection, IList,
ICollection, IEnumerable);
```

C++

```
class DataStoreParameterCollection : public DbParameterCollection, public
IDataParameterCollection, public IList, public ICollection, public IEnumerable;
```

File

Ado.Net20.cs

Description

DataStoreParameterCollection specifies a collection of DataStoreParameters for a DataStoreCommand (see page 1912).

DataStoreParameterCollection is an implementation of the .NET DbParameterCollection class.

See Also

DbParameterCollection (MSDN)

DataStoreCommand (see page 1912)

DataStoreParameter (see page 2033)

15.1.2.1.10.1 DataStoreParameterCollection Members

The following tables list the members exposed by DataStoreParameterCollection.

Internal Methods

	Name	Description
	InvalidatePreparation (see page 2048)	This is InvalidatePreparation, a member of class DataStoreParameterCollection.

Protected Methods

	Name	Description
	GetParameter (see page 2047)	This is the overview for the GetParameter method overload.
	SetParameter (see page 2050)	This is the overview for the SetParameter method overload.

Public Methods

	Name	Description
	Add (see page 2044)	This is the overview for the Add method overload.
	AddRange (see page 2045)	Adds array of items with specified values to DataStoreParameterCollection (see page 2041).
	Clear (see page 2045)	Removes all DataStoreParameter (see page 2033) values from DataStoreParameterCollection (see page 2041).
	Contains (see page 2045)	This is the overview for the Contains method overload.
	CopyTo (see page 2046)	Copies an array of items to collection.
	DataStoreParameterCollection (see page 2043)	Initializes a new instance of the DataStoreParameterCollection class.
	GetEnumerator (see page 2046)	Returns an enumerator for the collection.
	IndexOf (see page 2047)	This is the overview for the IndexOf method overload.
	Insert (see page 2048)	Inserts a parameter in collection.
	Remove (see page 2049)	Removes specified parameter from collection.
	RemoveAt (see page 2049)	This is the overview for the RemoveAt method overload.
	this (see page 2050)	This is the overview for the this indexer overload.

Public Properties

	Name	Description
	Count (see page 2051)	Returns a count of the number of parameters in the collection.

	IsFixedSize (see page 2051)	Returns a Boolean indicating whether the collection object has a fixed size or whether it is automatically expanding to accommodate addition of new items.
	IsReadOnly (see page 2052)	Returns a Boolean indicating whether the collection object is read-only or read-write.
	IsSynchronized (see page 2052)	Returns a Boolean indicating whether or not the collection is synchronized and therefore thread-safe.
	SyncRoot (see page 2052)	Returns the SyncRoot of the collection.

15.1.2.1.10.2 DataStoreParameterCollection.DataStoreParameterCollection Constructor

Initializes a new instance of the DataStoreParameterCollection class.

Pascal

```
constructor Create;
```

C++

```
DataStoreParameterCollection();
```

Description

DataStoreParameterCollection initializes a new instance of the DataStoreParameterCollection class, which is an implementation of the DbParameterCollection class.

See Also

DbParameterCollection constructor (MSDN)

DbParameterCollection class (MSDN)

15.1.2.1.10.3 DataStoreParameterCollection Methods

The methods of the DataStoreParameterCollection class are listed here.

Internal Methods

	Name	Description
	InvalidatePreparation (see page 2048)	This is InvalidatePreparation, a member of class DataStoreParameterCollection.

Protected Methods

	Name	Description
	GetParameter (see page 2047)	This is the overview for the GetParameter method overload.
	SetParameter (see page 2050)	This is the overview for the SetParameter method overload.

Public Methods

	Name	Description
	Add (see page 2044)	This is the overview for the Add method overload.
	AddRange (see page 2045)	Adds array of items with specified values to DataStoreParameterCollection (see page 2041).
	Clear (see page 2045)	Removes all DataStoreParameter (see page 2033) values from DataStoreParameterCollection (see page 2041).
	Contains (see page 2045)	This is the overview for the Contains method overload.
	CopyTo (see page 2046)	Copies an array of items to collection.
	GetEnumerator (see page 2046)	Returns an enumerator for the collection.
	IndexOf (see page 2047)	This is the overview for the IndexOf method overload.
	Insert (see page 2048)	Inserts a parameter in collection.
	Remove (see page 2049)	Removes specified parameter from collection.
	RemoveAt (see page 2049)	This is the overview for the RemoveAt method overload.
	this (see page 2050)	This is the overview for the this indexer overload.

15.1.2.1.10.3.1 Add Method

This is the overview for the Add method overload.

Overload List

	Name	Description
	DataStoreParameterCollection.Add (DataStoreParameter) (see page 2044)	Adds DataStoreParameter (see page 2033) object to collection.
	DataStoreParameterCollection.Add (object) (see page 2044)	Adds DataStoreParameter (see page 2033) object to collection.
	DataStoreParameterCollection.Add (string, DbType) (see page 2044)	Adds DataStoreParameter (see page 2033) object to collection.
	DataStoreParameterCollection.Add (string, DbType, ParameterDirection, string, DataRowVersion) (see page 2044)	Adds DataStoreParameter (see page 2033) object to collection.
	DataStoreParameterCollection.Add (string, JDSType) (see page 2044)	Adds DataStoreParameter (see page 2033) object to collection.
	DataStoreParameterCollection.Add (string, JDSType, ParameterDirection, string, DataRowVersion) (see page 2044)	Adds DataStoreParameter (see page 2033) object to collection.

15.1.2.1.10.3.1.1 DataStoreParameterCollection.Add Method (DataStoreParameter)

Adds DataStoreParameter (see page 2033) object to collection.

Pascal

```
function Add(value: object): Integer;
function Add(value: DataStoreParameter): Integer;
function Add(parameterName: string; type: DbType): DataStoreParameter;
function Add(parameterName: string; type: JDSType): DataStoreParameter;
function Add(parameterName: string; dbType: DbType; direction: ParameterDirection;
sourceColumn: string; version: DataRowVersion): DataStoreParameter;
function Add(parameterName: string; type: JDSType; direction: ParameterDirection;
sourceColumn: string; version: DataRowVersion): DataStoreParameter;
```

C++

```
int Add(object value);
int Add(DataStoreParameter value);
DataStoreParameter Add(string parameterName, DbType type);
DataStoreParameter Add(string parameterName, JDSType type);
DataStoreParameter Add(string parameterName, DbType dbType, ParameterDirection direction,
string sourceColumn, DataRowVersion version);
DataStoreParameter Add(string parameterName, JDSType type, ParameterDirection direction,
string sourceColumn, DataRowVersion version);
```

Description

Add adds a DataStoreParameter (see page 2033) to the DataStoreParameterCollection (see page 2041). It has several forms:

- Adds the specified DataStoreParameter (see page 2033) object to the collection.
- Adds the specified DataStoreParameter (see page 2033) object to the collection. Returns the index of the parameter in the collection.
- Adds a DataStoreParameter (see page 2033) with the specified name and type to the collection. Returns the DataStoreParameter (see page 2033) added.

- Adds a DataStoreParameter (see page 2033) with the specified name and JDSType to the collection. Returns the DataStoreParameter (see page 2033) added.
- Adds a DataStoreParameter (see page 2033) with the specified name, type, direction, source column, and source version to the collection. Returns the DataStoreParameter (see page 2033) added.
- Adds a DataStoreParameter (see page 2033) with the specified name, JDSType, direction, source column, and source version to the collection. Returns the DataStoreParameter (see page 2033) added.

See Also

Add (MSDN)

DataStoreParameter (see page 2033)

15.1.2.1.10.3.2 DataStoreParameterCollection.AddRange Method

Adds array of items with specified values to DataStoreParameterCollection (see page 2041).

Pascal

```
function AddRange(values: Array): void;
```

C++

```
void AddRange(Array values);
```

Description

AddRange adds an array of items with the specified values to the DataStoreParameterCollection (see page 2041).

See Also

AddRange (MSDN)

15.1.2.1.10.3.3 DataStoreParameterCollection.Clear Method

Removes all DataStoreParameter (see page 2033) values from DataStoreParameterCollection (see page 2041).

Pascal

```
function Clear: void;
```

C++

```
void Clear();
```

Description

Clear removes all DataStoreParameter (see page 2033) values from the DataStoreParameterCollection (see page 2041).

See Also

Clear (MSDN)

DataStoreParameter (see page 2033)

15.1.2.1.10.3.4 Contains Method

This is the overview for the Contains method overload.

Overload List

	Name	Description
	DataStoreParameterCollection.Contains (object) (see page 2046)	Checks if parameter exists in collection.

	DataStoreParameterCollection.Contains (string) (see page 2046)	Checks if parameter exists in collection.

15.1.2.1.10.3.4.1 DataStoreParameterCollection.Contains Method (object)

Checks if parameter exists in collection.

Pascal

```
function Contains(value: object): bool;
function Contains(parameterName: string): bool;
```

C++

```
bool Contains(object value);
bool Contains(string parameterName);
```

Description

There are several forms of this method:

- Checks if a specific parameter exists in the collection. The parameter **value** is value of the parameter. Returns **true** if the parameter is present.
- Checks if a parameter with the specified name exists in the collection. The parameter **parameterName** is name of the parameter. Returns **true** if the parameter is present.

See Also

Contains (MSDN)

15.1.2.1.10.3.5 DataStoreParameterCollection.CopyTo Method

Copies an array of items to collection.

Pascal

```
function CopyTo(array: Array; index: Integer): void;
```

C++

```
void CopyTo(Array array, int index);
```

Description

CopyTo copies an array of items to the collection starting at a specified index.

Parameter	Description
Values	Array of items to copy.
Index	Index in the collection at which to start copying.

See Also

CopyTo (MSDN)

15.1.2.1.10.3.6 DataStoreParameterCollection.GetEnumerator Method

Returns an enumerator for the collection.

Pascal

```
function GetEnumerator: IEnumerator;
```

C++

```
IEnumerator GetEnumerator();
```

Description

GetEnumerator returns an enumerator, which supports simple iteration over a collection. The returned value supports the IEnumerator Interface.

See Also

GetEnumerator (MSDN)

IEnumerator (MSDN)

15.1.2.1.10.3.7 GetParameter Method

This is the overview for the GetParameter method overload.

Overload List

	Name	Description
	DataStoreParameterCollection.GetParameter (int) (see page 2047)	This is GetParameter, a member of class DataStoreParameterCollection.
	DataStoreParameterCollection.GetParameter (string) (see page 2047)	This is GetParameter, a member of class DataStoreParameterCollection.

15.1.2.1.10.3.7.1 DataStoreParameterCollection.GetParameter Method (int)

Pascal

```
function GetParameter(index: Integer): DbParameter;
```

C++

```
DbParameter GetParameter(int index);
```

Description

This is GetParameter, a member of class DataStoreParameterCollection.

15.1.2.1.10.3.7.2 DataStoreParameterCollection.GetParameter Method (string)

Pascal

```
function GetParameter(parameterName: string): DbParameter;
```

C++

```
DbParameter GetParameter(string parameterName);
```

Description

This is GetParameter, a member of class DataStoreParameterCollection.

15.1.2.1.10.3.8 IndexOf Method

This is the overview for the IndexOf method overload.

Overload List

	Name	Description
	DataStoreParameterCollection.IndexOf (object) (see page 2048)	Returns index in collection of parameter.
	DataStoreParameterCollection.IndexOf (string) (see page 2048)	Returns index in collection of parameter.

15.1.2.1.10.3.8.1 DataStoreParameterCollection.IndexOf Method (object)

Returns index in collection of parameter.

Pascal

```
function IndexOf(value: object): Integer;
function IndexOf(parameterName: string): Integer;
```

C++

```
int IndexOf(object value);
int IndexOf(string parameterName);
```

Description

This method has several forms:

- Returns the index in the collection of a parameter value. The parameter **value** is name of the parameter.
- Returns the index in the collection of a parameter with the specified name. The parameter **parameterName** is name of the parameter.

See Also

IndexOf (MSDN)

15.1.2.1.10.3.9 DataStoreParameterCollection.Insert Method

Inserts a parameter in collection.

Pascal

```
function Insert(index: Integer; value: object): void;
```

C++

```
void Insert(int index, object value);
```

Description

Insert a specified DataStoreParameter (see page 2033) instance into a collection at the specified index.

Parameter	Description
Ordinal	Index in collection at which to insert DataStoreParameter (see page 2033) instance.
Parameter	DataStoreParameter (see page 2033) instance to insert.

See Also

Insert (MSDN)

15.1.2.1.10.3.10 DataStoreParameterCollection.InvalidatePreparation Method

Pascal

```
function InvalidatePreparation: void;
```

C++

```
void InvalidatePreparation();
```

Description

This is InvalidatePreparation, a member of class DataStoreParameterCollection.

15.1.2.1.10.3.11 DataStoreParameterCollection.Remove Method

Removes specified parameter from collection.

Pascal

```
function Remove(value: object): void;
```

C++

```
void Remove(object value);
```

Description

Remove removes the specified DataStoreParameter (see page 2033) instance from the collection.

See Also

Remove (MSDN)

15.1.2.1.10.3.12 RemoveAt Method

This is the overview for the RemoveAt method overload.

Overload List

	Name	Description
	DataStoreParameterCollection.RemoveAt (int) (see page 2049)	Removes a specified parameter from the collection.
	DataStoreParameterCollection.RemoveAt (string) (see page 2049)	Removes a specified parameter from the collection.

15.1.2.1.10.3.12.1 DataStoreParameterCollection.RemoveAt Method (int)

Removes a specified parameter from the collection.

Pascal

```
function RemoveAt(index: Integer): void;
function RemoveAt(parameterName: string): void;
```

C++

```
void RemoveAt(int index);
void RemoveAt(string parameterName);
```

Description

RemoveAt removes a specified DataStoreParameter (see page 2033) instance from the collection. The parameter may be specified by either a String or an index into the collection.

See Also

RemoveAt (MSDN)

15.1.2.1.10.3.13 SetParameter Method

This is the overview for the SetParameter method overload.

Overload List

	Name	Description
	DataStoreParameterCollection.SetParameter (int, DbParameter) (see page 2050)	This is SetParameter, a member of class DataStoreParameterCollection.
	DataStoreParameterCollection.SetParameter (string, DbParameter) (see page 2050)	This is SetParameter, a member of class DataStoreParameterCollection.

15.1.2.1.10.3.13.1 DataStoreParameterCollection.SetParameter Method (int, DbParameter)

Pascal

```
function SetParameter(index: Integer; value: DbParameter): void;
```

C++

```
void SetParameter(int index, DbParameter value);
```

Description

This is SetParameter, a member of class DataStoreParameterCollection.

15.1.2.1.10.3.13.2 DataStoreParameterCollection.SetParameter Method (string, DbParameter)

Pascal

```
function SetParameter(parameterName: string; value: DbParameter): void;
```

C++

```
void SetParameter(string parameterName, DbParameter value);
```

Description

This is SetParameter, a member of class DataStoreParameterCollection.

15.1.2.1.10.3.14 this Indexer

This is the overview for the this indexer overload.

Overload List

	Name	Description
	DataStoreParameterCollection.this (int) (see page 2050)	This is this, a member of class DataStoreParameterCollection.
	DataStoreParameterCollection.this (string) (see page 2051)	This is this, a member of class DataStoreParameterCollection.

15.1.2.1.10.3.14.1 DataStoreParameterCollection.this Indexer (int)

Pascal

```
this(index: Integer);
```

C++

```
DataStoreParameter this(int index);
```

Description

This is this, a member of class DataStoreParameterCollection.

15.1.2.1.10.3.14.2 DataStoreParameterCollection.this Indexer (string)

Pascal

```
this(index: string);
```

C++

```
DataStoreParameter this(string index);
```

Description

This is this, a member of class DataStoreParameterCollection.

15.1.2.1.10.4 DataStoreParameterCollection Properties

The properties of the DataStoreParameterCollection class are listed here.

Public Properties

	Name	Description
	Count (see page 2051)	Returns a count of the number of parameters in the collection.
	IsFixedSize (see page 2051)	Returns a Boolean indicating whether the collection object has a fixed size or whether it is automatically expanding to accommodate addition of new items.
	IsReadOnly (see page 2052)	Returns a Boolean indicating whether the collection object is read-only or read-write.
	IsSynchronized (see page 2052)	Returns a Boolean indicating whether or not the collection is synchronized and therefore thread-safe.
	SyncRoot (see page 2052)	Returns the SyncRoot of the collection.

15.1.2.1.10.4.1 DataStoreParameterCollection.Count Property

Returns a count of the number of parameters in the collection.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use this method to retrieve a count of the number of parameters in the collection. You might, for example, use the method to set a count parameter to control the number of times you must loop through the list of parameters.

See Also

Count (MSDN)

15.1.2.1.10.4.2 DataStoreParameterCollection.IsFixedSize Property

Returns a Boolean indicating whether the collection object has a fixed size or whether it is automatically expanding to accommodate addition of new items.

Pascal

```
property IsFixedSize: bool;
```

C++

```
__property bool IsFixedSize;
```

Description

By default, .NET collection objects are dynamically allocated and automatically expand to accommodate new members. You can set a limit on the size, which can improve performance. The Boolean returned by IsFixedSize is **True** if the parameter is a fixed size and **False** if it is variable.

See Also

IsFixedSize (MSDN)

15.1.2.1.10.4.3 DataStoreParameterCollection.IsReadOnly Property

Returns a Boolean indicating whether the collection object is read-only or read-write.

Pascal

```
property IsReadOnly: bool;
```

C++

```
__property bool IsReadOnly;
```

Description

If the collection object is read-only you can read and copy the parameter values, but you are unable to update the values as the result of an event. If the collection is read-write, you can modify the value of the parameters in the collection. The Boolean returned by IsReadOnly is **True** if the parameter collection is read-only and **False** if it is read-write.

See Also

IsReadOnly (MSDN)

15.1.2.1.10.4.4 DataStoreParameterCollection.IsSynchronized Property

Returns a Boolean indicating whether or not the collection is synchronized and therefore thread-safe.

Pascal

```
property IsSynchronized: bool;
```

C++

```
__property bool IsSynchronized;
```

Description

Synchronization of a collection, list, array, or hashtable object ensures that the object is thread-safe, meaning that it is safe from being modified by another thread. The Boolean returned by IsSynchronized is **True** if the parameter collection is synchronized and **False** if it is not.

See Also

IsSynchronized (MSDN)

15.1.2.1.10.4.5 DataStoreParameterCollection.SyncRoot Property

Returns the SyncRoot of the collection.

Pascal

```
property SyncRoot: object;
```

C++

```
__property object SyncRoot;
```

Description

Each collection, whether a list, array, or hashtable, has a SyncRoot. Each collection contains a single root and multiple nodes or branches below that root. When multiple threads access a collection, they might update values in the collection. When you want to use the values of the collection, your thread might not have an accurate and current snapshot of the collection that has been modified numerous times since you first retrieved it.

Perform the synchronization on the SyncRoot, not on the collection directly. Synchronization, starting at the SyncRoot, ripples the updates throughout the tree to give you an accurate collection. Nonetheless, even when the collection is synchronized, other threads can modify the collection, which then causes the enumerator to throw an exception. You might consider trapping the exception to make sure that you know when your collection is modified by another thread.

See Also

SyncRoot (MSDN)

15.1.2.1.11 DataStoreRowUpdatedEventArgs Class

Holds data for RowUpdated event of DataStoreDataAdapter (see page 1933).

Class Hierarchy

Pascal

```
DataStoreRowUpdatedEventArgs = class(RowUpdatedEventArgs);
```

C++

```
class DataStoreRowUpdatedEventArgs : public RowUpdatedEventArgs;
```

File

Ado.Net20.cs

Description

DataStoreRowUpdatedEventArgs provides an object to hold the data for the RowUpdated event of a DataStoreDataAdapter (see page 1933).

See Also

DataStoreDataAdapter (see page 1933)

RowUpdated (see page 1939)

15.1.2.1.11.1 DataStoreRowUpdatedEventArgs Members

The following tables list the members exposed by DataStoreRowUpdatedEventArgs.

Public Methods

	Name	Description
	DataStoreRowUpdatedEventArgs (see page 2054)	This is DataStoreRowUpdatedEventArgs, a member of class DataStoreRowUpdatedEventArgs.

Public Properties

	Name	Description
	Command (see page 2054)	DataStoreCommand (see page 1912) that triggered event.

15.1.2.1.11.2 DataStoreRowUpdatedEventArgs.DataStoreRowUpdatedEventArgs Constructor

Pascal

```
constructor Create(row: DataRow; command: IDbCommand; statementType: StatementType;
tableMapping: DataTableMapping);
```

C++

```
DataStoreRowUpdatedEventArgs(DataRow row, IDbCommand command, StatementType statementType,
DataTableMapping tableMapping);
```

Description

This is DataStoreRowUpdatedEventArgs, a member of class DataStoreRowUpdatedEventArgs.

15.1.2.1.11.3 DataStoreRowUpdatedEventArgs Properties

The properties of the DataStoreRowUpdatedEventArgs class are listed here.

Public Properties

	Name	Description
	Command (see page 2054)	DataStoreCommand (see page 1912) that triggered event.

15.1.2.1.11.3.1 DataStoreRowUpdatedEventArgs.Command Property

DataStoreCommand (see page 1912) that triggered event.

Pascal

```
property Command: DataStoreCommand;
```

C++

```
__property DataStoreCommand Command;
```

Description

Command returns the DataStoreCommand (see page 1912) that triggered the row updated event.

See Also

DataStoreCommand (see page 1912)

15.1.2.1.12 DataStoreRowUpdatingEventArgs Class

Holds data for RowUpdating event of DataStoreDataAdapter (see page 1933).

Class Hierarchy

Pascal

```
DataStoreRowUpdatingEventArgs = class(RowUpdatingEventArgs);
```

C++

```
class DataStoreRowUpdatingEventArgs : public RowUpdatingEventArgs;
```

File

Ado.Net20.cs

Description

DataStoreRowUpdatingEventArgs provides an object to hold the data for the RowUpdating event of a DataStoreDataAdapter (see page 1933).

See Also

DataStoreDataAdapter (see page 1933)

RowUpdating (see page 1939)

15.1.2.1.12.1 DataStoreRowUpdatingEventArgs Members

The following tables list the members exposed by DataStoreRowUpdatingEventArgs.

Public Methods

	Name	Description
	DataStoreRowUpdatingEventArgs (see page 2055)	This is DataStoreRowUpdatingEventArgs, a member of class DataStoreRowUpdatingEventArgs.

Public Properties

	Name	Description
	Command (see page 2055)	DataStoreCommand (see page 1912) that triggered event.

15.1.2.1.12.2 DataStoreRowUpdatingEventArgs.DataStoreRowUpdatingEventArgs Constructor

Pascal

```
constructor Create(row: DataRow; command: IDbCommand; statementType: StatementType;
tableMapping: DataTableMapping);
```

C++

```
DataStoreRowUpdatingEventArgs(DataRow row, IDbCommand command, StatementType statementType,
DataTableMapping tableMapping);
```

Description

This is DataStoreRowUpdatingEventArgs, a member of class DataStoreRowUpdatingEventArgs.

15.1.2.1.12.3 DataStoreRowUpdatingEventArgs Properties

The properties of the DataStoreRowUpdatingEventArgs class are listed here.

Public Properties

	Name	Description
	Command (see page 2055)	DataStoreCommand (see page 1912) that triggered event.

15.1.2.1.12.3.1 DataStoreRowUpdatingEventArgs.Command Property

DataStoreCommand (see page 1912) that triggered event.

Pascal

```
property Command: DataStoreCommand;
```

C++

```
__property DataStoreCommand Command;
```

Description

Command gets and sets the DataStoreCommand (see page 1912) that triggered the row updating event.

See Also

DataStoreCommand (see page 1912)

15.1.2.1.13 DataStoreTransaction Class

Class Hierarchy

Pascal

```
DataStoreTransaction = class(DbTransaction, IDbTransaction, IDisposable);
```

C++

```
class DataStoreTransaction : public DbTransaction, public IDbTransaction, public IDisposable;
```

File

Ado.Net20.cs

Description

This is class Borland.Data.DataStore.DataStoreTransaction.

15.1.2.1.13.1 DataStoreTransaction Members

The following tables list the members exposed by DataStoreTransaction.

Internal Methods

	Name	Description
	DataStoreTransaction (see page 2056)	This is DataStoreTransaction, a member of class DataStoreTransaction.

Protected Properties

	Name	Description
	DbConnection (see page 2058)	This is DbConnection, a member of class DataStoreTransaction.

Public Methods

	Name	Description
	Commit (see page 2057)	Commits the current transaction..
	Dispose (see page 2057)	This is the overview for the Dispose method overload.
	Rollback (see page 2058)	Rolls back the current transaction..

Public Properties

	Name	Description
	Connection (see page 2058)	Gets the connection on which the transaction was started.
	IsolationLevel (see page 2058)	Gets the transaction isolation level for this transaction.

15.1.2.1.13.2 DataStoreTransaction.DataStoreTransaction Constructor

Pascal

```
constructor Create(connection: DataStoreConnection; outerMost: bool);
```

C++

```
DataStoreTransaction(DataStoreConnection connection, bool outerMost);
```

Description

This is DataStoreTransaction, a member of class DataStoreTransaction.

15.1.2.1.13.3 DataStoreTransaction Methods

The methods of the DataStoreTransaction class are listed here.

Public Methods

	Name	Description
	Commit (see page 2057)	Commits the current transaction..
	Dispose (see page 2057)	This is the overview for the Dispose method overload.
	Rollback (see page 2058)	Rolls back the current transaction..

15.1.2.1.13.3.1 DataStoreTransaction.Commit Method

Commits the current transaction..

Pascal

```
function Commit: void;
```

C++

```
void Commit();
```

Description

Commit commits the current transaction.

15.1.2.1.13.3.2 Dispose Method

This is the overview for the Dispose method overload.

Overload List

	Name	Description
	DataStoreTransaction.Dispose () (see page 2057)	Disposes this object.
	DataStoreTransaction.Dispose (bool) (see page 2057)	Disposes this object.

15.1.2.1.13.3.2.1 DataStoreTransaction.Dispose Method ()

Disposes this object.

Pascal

```
function Dispose: void;
function Dispose(disposing: bool): void;
```

C++

```
void Dispose();
void Dispose(bool disposing);
```

Description

Dispose disposes this object.

15.1.2.1.13.3.3 DataStoreTransaction.Rollback Method

Rolls back the current transaction..

Pascal

```
function Rollback: void;
```

C++

```
void Rollback();
```

Description

Rollback rolls back the current transaction.

15.1.2.1.13.4 DataStoreTransaction Properties

The properties of the DataStoreTransaction class are listed here.

Protected Properties

	Name	Description
	DbConnection (see page 2058)	This is DbConnection, a member of class DataStoreTransaction.

Public Properties

	Name	Description
	Connection (see page 2058)	Gets the connection on which the transaction was started.
	IsolationLevel (see page 2058)	Gets the transaction isolation level for this transaction.

15.1.2.1.13.4.1 DataStoreTransaction.Connection Property

Gets the connection on which the transaction was started.

Pascal

```
property Connection: DataStoreConnection;
```

C++

```
__property DataStoreConnection Connection;
```

Description

Connection gets the connection on which the transaction was started.

15.1.2.1.13.4.2 DataStoreTransaction.DbConnection Property

Pascal

```
property DbConnection: DbConnection;
```

C++

```
__property DbConnection DbConnection;
```

Description

This is DbConnection, a member of class DataStoreTransaction.

15.1.2.1.13.4.3 DataStoreTransaction.IsolationLevel Property

Gets the transaction isolation level for this transaction.

Pascal

```
property IsolationLevel: IsolationLevel;
```

C++

```
__property IsolationLevel IsolationLevel;
```

Description

IsolationLevel gets the transaction isolation level for this transaction.

15.1.2.1.14 LicenseColumns Class

Column names for result table returned from DB_ADMIN.GET_LICENSE and DB_ADMIN.GET_ALL_LICENSES.

Class Hierarchy

Borland.Data.DataStore.LicenseColumns

Pascal

```
LicenseColumns = class;
```

C++

```
class LicenseColumns;
```

File

LicenseColumns.cs

Description

LicenseColumns contains column names for result table returned from DB_ADMIN.GET_LICENSE and DB_ADMIN.GET_ALL_LICENSES

The columns of the result table are:

Property Names	Description
USER_LIMIT	Maximum number of database users allowed across all databases.
FILE_SIZE_LIMIT	Maximum database file size in megabytes. Once this size is exceeded, the database can be read, but no longer allows any write operations.
DESKTOP	If this is **true**, applications can only connect to databases on the local desktop computer. If a remote connection is being used, that connection can only connect to the localhost host.
DEPLOYMENT_LIMIT	Maximum times a database application can be deployed.
EXPIRED	Location of the license file used for this deployment.
SERIAL_NUMBER	Serial number.
PRODUCT_ID	Product ID.
SKU_ID	SKU ID.
FILE_NAME	Location of the license file used for this deployment.

15.1.2.1.14.1 LicenseColumns Members

The following tables list the members exposed by LicenseColumns.

Public Fields

	Name	Description
	CacheSizeLimit (see page 2060)	This is CacheSizeLimit, a member of class LicenseColumns.
	DeploymentLimit (see page 2060)	This is DeploymentLimit, a member of class LicenseColumns.
	Desktop (see page 2060)	This is Desktop, a member of class LicenseColumns.

	Expired (see page 2061)	This is Expired, a member of class LicenseColumns.
	FileName (see page 2061)	This is FileName, a member of class LicenseColumns.
	FileSizeLimit (see page 2061)	This is FileSizeLimit, a member of class LicenseColumns.
	ProductId (see page 2061)	This is ProductId, a member of class LicenseColumns.
	SerialNumber (see page 2061)	This is SerialNumber, a member of class LicenseColumns.
	SkuId (see page 2062)	This is SkuId, a member of class LicenseColumns.
	UserLimit (see page 2062)	This is UserLimit, a member of class LicenseColumns.

15.1.2.1.14.2 LicenseColumns Fields

The fields of the LicenseColumns class are listed here.

Public Fields

	Name	Description
	CacheSizeLimit (see page 2060)	This is CacheSizeLimit, a member of class LicenseColumns.
	DeploymentLimit (see page 2060)	This is DeploymentLimit, a member of class LicenseColumns.
	Desktop (see page 2060)	This is Desktop, a member of class LicenseColumns.
	Expired (see page 2061)	This is Expired, a member of class LicenseColumns.
	FileName (see page 2061)	This is FileName, a member of class LicenseColumns.
	FileSizeLimit (see page 2061)	This is FileSizeLimit, a member of class LicenseColumns.
	ProductId (see page 2061)	This is ProductId, a member of class LicenseColumns.
	SerialNumber (see page 2061)	This is SerialNumber, a member of class LicenseColumns.
	SkuId (see page 2062)	This is SkuId, a member of class LicenseColumns.
	UserLimit (see page 2062)	This is UserLimit, a member of class LicenseColumns.

15.1.2.1.14.2.1 LicenseColumns.CacheSizeLimit Field

Pascal

```
const CacheSizeLimit: string = "CacheSizeLimit";
```

C++

```
string CacheSizeLimit = "CacheSizeLimit";
```

Description

This is CacheSizeLimit, a member of class LicenseColumns.

15.1.2.1.14.2.2 LicenseColumns.DeploymentLimit Field

Pascal

```
const DeploymentLimit: string = "DeploymentLimit";
```

C++

```
string DeploymentLimit = "DeploymentLimit";
```

Description

This is DeploymentLimit, a member of class LicenseColumns.

15.1.2.1.14.2.3 LicenseColumns.Desktop Field

Pascal

```
const Desktop: string = "Desktop";
```

C++

```
string Desktop = "Desktop";
```

Description

This is Desktop, a member of class LicenseColumns.

15.1.2.1.14.2.4 LicenseColumns.Expired Field

Pascal

```
const Expired: string = "Expired";
```

C++

```
string Expired = "Expired";
```

Description

This is Expired, a member of class LicenseColumns.

15.1.2.1.14.2.5 LicenseColumns.FileName Field

Pascal

```
const FileName: string = "FileName";
```

C++

```
string FileName = "FileName";
```

Description

This is FileName, a member of class LicenseColumns.

15.1.2.1.14.2.6 LicenseColumns.FileSizeLimit Field

Pascal

```
const FileSizeLimit: string = "FileSizeLimit";
```

C++

```
string FileSizeLimit = "FileSizeLimit";
```

Description

This is FileSizeLimit, a member of class LicenseColumns.

15.1.2.1.14.2.7 LicenseColumns.ProductId Field

Pascal

```
const ProductId: string = "ProductId";
```

C++

```
string ProductId = "ProductId";
```

Description

This is ProductId, a member of class LicenseColumns.

15.1.2.1.14.2.8 LicenseColumns.SerialNumber Field

Pascal

```
const SerialNumber: string = "SerialNumber";
```

C++

```
string SerialNumber = "SerialNumber";
```

Description

This is SerialNumber, a member of class LicenseColumns.

15.1.2.1.14.2.9 LicenseColumns.SkuId Field

Pascal

```
const SkuId: string = "SKUId";
```

C++

```
string SkuId = "SKUId";
```

Description

This is SkuId, a member of class LicenseColumns.

15.1.2.1.14.2.10 LicenseColumns.UserLimit Field

Pascal

```
const UserLimit: string = "UserLimit";
```

C++

```
string UserLimit = "UserLimit";
```

Description

This is UserLimit, a member of class LicenseColumns.

15.1.2.1.15 LockColumns Class

Provides column names for result table from built-in stored procedure DB_ADMIN.GET_LOCKS.

Class Hierarchy

Borland.Data.DataStore.LockColumns

Pascal

```
LockColumns = class;
```

C++

```
class LockColumns;
```

File

LockColumns.cs

Description

LockColumns provides the column names for the result table from the built-in stored procedure DB_ADMIN.GET_LOCKS. The columns of the result table are:

Constant	Description
NAME	Name (see page 2064) of object that lock applies to. Typically this is the name of a table.
TYPE	The type of lock, which can be table or row.
GRANTED	Indicates whether or not the lock has been granted.
SHARED	Indicates whether or not the lock can be shared with another shared lock. If **false**, this is an exclusive lock that cannot share.

USER	User (see page 2065) that holds the lock.
ROW	If the **TYPE** column is set to row, this is the row that the lock applies to. Otherwise this is set to null.
COMP_MASK	If the **TYPE** column is set to **TABLE**, this is set to a collection of bits that this table lock is compatible with.
LOCK_MASK	If the TYPE column is set to **TABLE**, this will be set to a collection of bits that this table lock is locking.
ROW_LOCK	Row (see page 2064) lock.
TABLE_LOCK	Table lock.

See Also

DB_ADMIN

GET_LOCKS

15.1.2.1.15.1 LockColumns Members

The following tables list the members exposed by LockColumns.

Public Fields

	Name	Description
	CompMask (see page 2063)	This is CompMask, a member of class LockColumns.
	Granted (see page 2064)	This is Granted, a member of class LockColumns.
	LockMask (see page 2064)	This is LockMask, a member of class LockColumns.
	Name (see page 2064)	This is Name, a member of class LockColumns.
	Row (see page 2064)	This is Row, a member of class LockColumns.
	RowLock (see page 2064)	This is RowLock, a member of class LockColumns.
	Shared (see page 2065)	This is Shared, a member of class LockColumns.
	TableLock (see page 2065)	This is TableLock, a member of class LockColumns.
	Type (see page 2065)	This is Type, a member of class LockColumns.
	User (see page 2065)	This is User, a member of class LockColumns.

15.1.2.1.15.2 LockColumns Fields

The fields of the LockColumns class are listed here.

Public Fields

	Name	Description
	CompMask (see page 2063)	This is CompMask, a member of class LockColumns.
	Granted (see page 2064)	This is Granted, a member of class LockColumns.
	LockMask (see page 2064)	This is LockMask, a member of class LockColumns.
	Name (see page 2064)	This is Name, a member of class LockColumns.
	Row (see page 2064)	This is Row, a member of class LockColumns.
	RowLock (see page 2064)	This is RowLock, a member of class LockColumns.
	Shared (see page 2065)	This is Shared, a member of class LockColumns.
	TableLock (see page 2065)	This is TableLock, a member of class LockColumns.
	Type (see page 2065)	This is Type, a member of class LockColumns.
	User (see page 2065)	This is User, a member of class LockColumns.

15.1.2.1.15.2.1 LockColumns.CompMask Field

Pascal

```
const CompMask: string = "COMP_MASK";
```

C++

```
string CompMask = "COMP_MASK";
```

Description

This is CompMask, a member of class LockColumns.

15.1.2.1.15.2.2 LockColumns.Granted Field

Pascal

```
const Granted: string = "GRANTED";
```

C++

```
string Granted = "GRANTED";
```

Description

This is Granted, a member of class LockColumns.

15.1.2.1.15.2.3 LockColumns.LockMask Field

Pascal

```
const LockMask: string = "LOCK_MASK";
```

C++

```
string LockMask = "LOCK_MASK";
```

Description

This is LockMask, a member of class LockColumns.

15.1.2.1.15.2.4 LockColumns.Name Field

Pascal

```
const Name: string = "NAME";
```

C++

```
string Name = "NAME";
```

Description

This is Name, a member of class LockColumns.

15.1.2.1.15.2.5 LockColumns.Row Field

Pascal

```
const Row: string = "ROW";
```

C++

```
string Row = "ROW";
```

Description

This is Row, a member of class LockColumns.

15.1.2.1.15.2.6 LockColumns.RowLock Field

Pascal

```
const RowLock: string = "ROW_LOCK";
```

C++

```
string RowLock = "ROW_LOCK";
```

Description

This is RowLock, a member of class LockColumns.

15.1.2.1.15.2.7 LockColumns.Shared Field

Pascal

```
const Shared: string = "SHARED";
```

C++

```
string Shared = "SHARED";
```

Description

This is Shared, a member of class LockColumns.

15.1.2.1.15.2.8 LockColumns.TableLock Field

Pascal

```
const TableLock: string = "TABLE_LOCK";
```

C++

```
string TableLock = "TABLE_LOCK";
```

Description

This is TableLock, a member of class LockColumns.

15.1.2.1.15.2.9 LockColumns.Type Field

Pascal

```
const Type: string = "TYPE";
```

C++

```
string Type = "TYPE";
```

Description

This is Type, a member of class LockColumns.

15.1.2.1.15.2.10 LockColumns.User Field

Pascal

```
const User: string = "USER";
```

C++

```
string User = "USER";
```

Description

This is User, a member of class LockColumns.

15.1.2.1.16 LogFilter Class

Provides names of events to be logged.

Class Hierarchy

Borland.Data.DataStore.LogFilter

Pascal

```
LogFilter = class;
```

C++

```
class LogFilter;
```

File

LogFilter.cs

Description

LogFilter provides the names of events to be logged. Certain database events can be logged to a status text file placed in the same directory as the database.

This table contains events that can be logged.

Event	Description
Init (see page 2068)	The initial property settings for a new connection.
Connect (see page 2067)	Connect (see page 2067) and disconnect events.
Prepare (see page 2068)	SQL Statement prepare.
SetParameter (see page 2068)	SQL parameter settings.
Execute (see page 2067)	SQL Statement execution.
Transactions (see page 2069)	Transaction commit and rollback operations.
Errors (see page 2067)	Exceptions encountered.
LockErrors (see page 2068)	Lock errors like time-outs and deadlocks.
Xa (see page 2069)	Distributed computing XA related operations.
Mirror (see page 2068)	Database mirroring operations.
System (see page 2069)	System (see page 2069) operations, such as database startup/shutdown, log file creation and truncation.

15.1.2.1.16.1 LogFilter Members

The following tables list the members exposed by LogFilter.

Public Fields

	Name	Description
	Connect (see page 2067)	This is Connect, a member of class LogFilter.
	Errors (see page 2067)	This is Errors, a member of class LogFilter.
	Execute (see page 2067)	This is Execute, a member of class LogFilter.
	Init (see page 2068)	This is Init, a member of class LogFilter.

	LockErrors (see page 2068)	This is LockErrors, a member of class LogFilter.
	Mirror (see page 2068)	This is Mirror, a member of class LogFilter.
	Prepare (see page 2068)	This is Prepare, a member of class LogFilter.
	SetParameter (see page 2068)	This is SetParameter, a member of class LogFilter.
	System (see page 2069)	This is System, a member of class LogFilter.
	Transactions (see page 2069)	This is Transactions, a member of class LogFilter.
	Xa (see page 2069)	This is Xa, a member of class LogFilter.

15.1.2.1.16.2 LogFilter Fields

The fields of the LogFilter class are listed here.

Public Fields

	Name	Description
	Connect (see page 2067)	This is Connect, a member of class LogFilter.
	Errors (see page 2067)	This is Errors, a member of class LogFilter.
	Execute (see page 2067)	This is Execute, a member of class LogFilter.
	Init (see page 2068)	This is Init, a member of class LogFilter.
	LockErrors (see page 2068)	This is LockErrors, a member of class LogFilter.
	Mirror (see page 2068)	This is Mirror, a member of class LogFilter.
	Prepare (see page 2068)	This is Prepare, a member of class LogFilter.
	SetParameter (see page 2068)	This is SetParameter, a member of class LogFilter.
	System (see page 2069)	This is System, a member of class LogFilter.
	Transactions (see page 2069)	This is Transactions, a member of class LogFilter.
	Xa (see page 2069)	This is Xa, a member of class LogFilter.

15.1.2.1.16.2.1 LogFilter.Connect Field

Pascal

```
const Connect: string = "CONNECT";
```

C++

```
string Connect = "CONNECT";
```

Description

This is Connect, a member of class LogFilter.

15.1.2.1.16.2.2 LogFilter.Errors Field

Pascal

```
const Errors: string = "ERRORS";
```

C++

```
string Errors = "ERRORS";
```

Description

This is Errors, a member of class LogFilter.

15.1.2.1.16.2.3 LogFilter.Execute Field

Pascal

```
const Execute: string = "EXECUTE";
```

C++

```
string Execute = "EXECUTE";
```

Description

This is Execute, a member of class LogFilter.

15.1.2.1.16.2.4 LogFilter.Init Field

Pascal

```
const Init: string = "INIT";
```

C++

```
string Init = "INIT";
```

Description

This is Init, a member of class LogFilter.

15.1.2.1.16.2.5 LogFilter.LockErrors Field

Pascal

```
const LockErrors: string = "LOCK_ERRORS";
```

C++

```
string LockErrors = "LOCK_ERRORS";
```

Description

This is LockErrors, a member of class LogFilter.

15.1.2.1.16.2.6 LogFilter.Mirror Field

Pascal

```
const Mirror: string = "MIRROR";
```

C++

```
string Mirror = "MIRROR";
```

Description

This is Mirror, a member of class LogFilter.

15.1.2.1.16.2.7 LogFilter.Prepare Field

Pascal

```
const Prepare: string = "PREPARE";
```

C++

```
string Prepare = "PREPARE";
```

Description

This is Prepare, a member of class LogFilter.

15.1.2.1.16.2.8 LogFilter.SetParameter Field

Pascal

```
const SetParameter: string = "SET_PARAMETER";
```

C++

```
string SetParameter = "SET_PARAMETER";
```

Description

This is SetParameter, a member of class LogFilter.

15.1.2.1.16.2.9 LogFilter.System Field

Pascal

```
const System: string = "SYSTEM";
```

C++

```
string System = "SYSTEM";
```

Description

This is System, a member of class LogFilter.

15.1.2.1.16.2.10 LogFilter.Transactions Field

Pascal

```
const Transactions: string = "TRANSACTIONS";
```

C++

```
string Transactions = "TRANSACTIONS";
```

Description

This is Transactions, a member of class LogFilter.

15.1.2.1.16.2.11 LogFilter.Xa Field

Pascal

```
const Xa: string = "XA";
```

C++

```
string Xa = "XA";
```

Description

This is Xa, a member of class LogFilter.

15.1.2.1.17 MirrorStatusColumns Class

Provides columns of result table from built-in stored procedure DB_ADMIN.GET_MIRRORS.

Class Hierarchy

Pascal

```
MirrorStatusColumns = class(Borland.Data.DataStore.SysMirrors);
```

C++

```
class MirrorStatusColumns : public Borland.Data.DataStore.SysMirrors;
```

File

MirrorStatusColumns.cs

Description

The DB_ADMIN.GET_MIRRORS function provides a result table consisting of columns from SysMirrors (see page 2089) and

the status columns specified below. The checkStatus parameter of DB_ADMIN.GET_MIRRORS must be true for any data to be returned in the status columns.

The status columns are:

Constant	Description
MIRROR_REPLAY_STAMP	A stamp that marks the last time the log files were replayed against this mirror.
ATTACHED_AUTO_FAIL_OVER_COUNT	The number of mirrors configured for automatic fail over that are attached to by the primary mirror. The majority of all automatic fail over mirrors are attached for automatic fail over to be enabled.
TOTAL_AUTO_FAIL_OVER_COUNT	The total number of all mirrors configured for automatic fail over. The majority of all automatic fail over mirrors must be attached for automatic fail over to be enabled.
ATTACHED_INSTANT_SYNCH_COUNT	The number of mirrors configured for instant synchronization with this mirror that are attached to this mirror.
TOTAL_INSTANT_SYNCH_COUNT	The number of mirrors configured for instant synchronization with this mirror.
CONNECT_ERROR	Exception Object encountered when trying to retrieve status on this mirror.
HIGHEST_REDO_STAMP	A stamp that marks the most recent redo record applied to the database.

See Also

DB_ADMIN

GET_MIRRORS

15.1.2.1.17.1 MirrorStatusColumns Members

The following tables list the members exposed by MirrorStatusColumns.

Public Fields

	Name	Description
	AttachedAutoFailOverCount (see page 2071)	This is AttachedAutoFailOverCount, a member of class MirrorStatusColumns.
	AttachedInstantSynchCount (see page 2071)	This is AttachedInstantSynchCount, a member of class MirrorStatusColumns.
	AutoFailOver (see page 2092)	This is AutoFailOver, a member of class SysMirrors.
	BirthStamp (see page 2092)	This is BirthStamp, a member of class SysMirrors.
	ClientHashCode (see page 2092)	This is ClientHashCode, a member of class SysMirrors.
	ConnectError (see page 2071)	This is ConnectError, a member of class MirrorStatusColumns.
	FailOverPriority (see page 2092)	This is FailOverPriority, a member of class SysMirrors.
	FileName (see page 2093)	This is FileName, a member of class SysMirrors.
	HighestRedoStamp (see page 2071)	This is HighestRedoStamp, a member of class MirrorStatusColumns.
	Host (see page 2093)	This is Host, a member of class SysMirrors.
	Id (see page 2093)	This is Id, a member of class SysMirrors.
	InstantSynch (see page 2093)	This is InstantSynch, a member of class SysMirrors.
	LastKnownReplay (see page 2093)	This is LastKnownReplay, a member of class SysMirrors.
	MirrorReplayStamp (see page 2072)	This is MirrorReplayStamp, a member of class MirrorStatusColumns.
	Name (see page 2094)	This is Name, a member of class SysMirrors.
	Port (see page 2094)	This is Port, a member of class SysMirrors.
	SynchMirrorName (see page 2094)	This is SynchMirrorName, a member of class SysMirrors.
	TotalAutoFailOverCount (see page 2072)	This is TotalAutoFailOverCount, a member of class MirrorStatusColumns.
	TotalInstantSynchCount (see page 2072)	This is TotalInstantSynchCount, a member of class MirrorStatusColumns.
	Type (see page 2094)	This is Type, a member of class SysMirrors.
	ValidatedPrimary (see page 2094)	This is ValidatedPrimary, a member of class SysMirrors.

15.1.2.1.17.2 MirrorStatusColumns Fields

The fields of the MirrorStatusColumns class are listed here.

Public Fields

	Name	Description
	AttachedAutoFailOverCount (see page 2071)	This is AttachedAutoFailOverCount, a member of class MirrorStatusColumns.
	AttachedInstantSynchCount (see page 2071)	This is AttachedInstantSynchCount, a member of class MirrorStatusColumns.
	ConnectError (see page 2071)	This is ConnectError, a member of class MirrorStatusColumns.
	HighestRedoStamp (see page 2071)	This is HighestRedoStamp, a member of class MirrorStatusColumns.
	MirrorReplayStamp (see page 2072)	This is MirrorReplayStamp, a member of class MirrorStatusColumns.
	TotalAutoFailOverCount (see page 2072)	This is TotalAutoFailOverCount, a member of class MirrorStatusColumns.
	TotalInstantSynchCount (see page 2072)	This is TotalInstantSynchCount, a member of class MirrorStatusColumns.

15.1.2.1.17.2.1 MirrorStatusColumns.AttachedAutoFailOverCount Field

Pascal

```
const AttachedAutoFailOverCount: string = "ATTACHED_AUTO_FAIL_OVER_COUNT";
```

C++

```
string AttachedAutoFailOverCount = "ATTACHED_AUTO_FAIL_OVER_COUNT";
```

Description

This is AttachedAutoFailOverCount, a member of class MirrorStatusColumns.

15.1.2.1.17.2.2 MirrorStatusColumns.AttachedInstantSynchCount Field

Pascal

```
const AttachedInstantSynchCount: string = "ATTACHED_INSTANT_SYNCH_COUNT";
```

C++

```
string AttachedInstantSynchCount = "ATTACHED_INSTANT_SYNCH_COUNT";
```

Description

This is AttachedInstantSynchCount, a member of class MirrorStatusColumns.

15.1.2.1.17.2.3 MirrorStatusColumns.ConnectError Field

Pascal

```
const ConnectError: string = "CONNECT_ERROR";
```

C++

```
string ConnectError = "CONNECT_ERROR";
```

Description

This is ConnectError, a member of class MirrorStatusColumns.

15.1.2.1.17.2.4 MirrorStatusColumns.HighestRedoStamp Field

Pascal

```
const HighestRedoStamp: string = "HIGHEST_REDO_STAMP";
```

C++

```
string HighestRedoStamp = "HIGHEST_REDO_STAMP";
```

Description

This is HighestRedoStamp, a member of class MirrorStatusColumns.

15.1.2.1.17.2.5 MirrorStatusColumns.MirrorReplayStamp Field

Pascal

```
const MirrorReplayStamp: string = "MIRROR_REPLAY_STAMP";
```

C++

```
string MirrorReplayStamp = "MIRROR_REPLAY_STAMP";
```

Description

This is MirrorReplayStamp, a member of class MirrorStatusColumns.

15.1.2.1.17.2.6 MirrorStatusColumns.TotalAutoFailOverCount Field

Pascal

```
const TotalAutoFailOverCount: string = "TOTAL_AUTO_FAIL_OVER_COUNT";
```

C++

```
string TotalAutoFailOverCount = "TOTAL_AUTO_FAIL_OVER_COUNT";
```

Description

This is TotalAutoFailOverCount, a member of class MirrorStatusColumns.

15.1.2.1.17.2.7 MirrorStatusColumns.TotalInstantSynchCount Field

Pascal

```
const TotalInstantSynchCount: string = "TOTAL_INSTANT_SYNCH_COUNT";
```

C++

```
string TotalInstantSynchCount = "TOTAL_INSTANT_SYNCH_COUNT";
```

Description

This is TotalInstantSynchCount, a member of class MirrorStatusColumns.

15.1.2.1.18 MirrorTypes Class

Contains the valid values for the TYPE column from DB_ADMIN.GET_MIRRORS stored procedure.

Class Hierarchy

Pascal

```
MirrorTypes = class;
```

C++

```
class MirrorTypes;
```

File

MirrorTypes.cs

Description

MirrorTypes contains the valid values for the TYPE column from DB_ADMIN.GET_MIRRORS stored procedure. These values

are also found in the system table SYS.SYSMIRRORS.

The valid values are:

Constant	Description
NONE	Mirroring not enabled.
PRIMARY	There is only one primary mirror. It is the only mirror that supports any form of write operation on the database.
READONLY	There can be one or more **READONLY** MIRRORS. **READONLY** mirrors support read operations, but not write operations. **READONLY** mirrors provide a view of the PRIMARY database at the point in time they were last synchronized with the **PRIMARY**.
DIRECTORY	There can be one or more **DIRECTORY** MIRRORS. **DIRECTORY** mirrors support read operations, but not write operations. A **DIRECTORY** mirror mirrors the **SYS.MIRRORS** table, but not the data itself. Connection requests to a **DIRECTORY** mirror are redirected to the **PRIMARY** mirror if a writable connection is needed. Otherwise a load balancing algorithm is used to decide which **READONLY** mirror should be used for the connection.

See Also

SysMirrors (see page 2089)

15.1.2.1.18.1 MirrorTypes Members

The following tables list the members exposed by MirrorTypes.

Public Fields

	Name	Description
	Directory (see page 2073)	This is Directory, a member of class MirrorTypes.
	None (see page 2074)	This is None, a member of class MirrorTypes.
	Primary (see page 2074)	This is Primary, a member of class MirrorTypes.
	Readonly (see page 2074)	This is Readonly, a member of class MirrorTypes.

15.1.2.1.18.2 MirrorTypes Fields

The fields of the MirrorTypes class are listed here.

Public Fields

	Name	Description
	Directory (see page 2073)	This is Directory, a member of class MirrorTypes.
	None (see page 2074)	This is None, a member of class MirrorTypes.
	Primary (see page 2074)	This is Primary, a member of class MirrorTypes.
	Readonly (see page 2074)	This is Readonly, a member of class MirrorTypes.

15.1.2.1.18.2.1 MirrorTypes.Directory Field

Pascal

```
const Directory: string = "DIRECTORY";
```

C++

```
string Directory = "DIRECTORY";
```

Description

This is Directory, a member of class MirrorTypes.

15.1.2.1.18.2.2 MirrorTypes.None Field

Pascal

```
const None: string = "NONE";
```

C++

```
string None = "NONE";
```

Description

This is None, a member of class MirrorTypes.

15.1.2.1.18.2.3 MirrorTypes.Primary Field

Pascal

```
const Primary: string = "PRIMARY";
```

C++

```
string Primary = "PRIMARY";
```

Description

This is Primary, a member of class MirrorTypes.

15.1.2.1.18.2.4 MirrorTypes.Readonly Field

Pascal

```
const Readonly: string = "READONLY";
```

C++

```
string Readonly = "READONLY";
```

Description

This is Readonly, a member of class MirrorTypes.

15.1.2.1.19 ProtocolValue Class

Contains constant strings for the protocol connection property.

Class Hierarchy

Borland.Data.DataStore.ProtocolValue

Pascal

```
ProtocolValue = class;
```

C++

```
class ProtocolValue;
```

File

ProtocolValue.cs

Description

The ProtocolValue class contains constant strings for the protocol connection property.

The columns of the result table are:

Constant	Description
InProcess (see page 2075)	Specify this protocol when the server should be run in process with the client code. Note that a database file can only be accessed by one server process at any given time.
Tcp (see page 2075)	Specify this protocol for using the DataStore (see page 1892) TCP/IP protocol.

See Also

Borland.Data.DataStore.ConnectionPropertiesProtocol

15.1.2.1.19.1 ProtocolValue Members

The following tables list the members exposed by ProtocolValue.

Public Fields

	Name	Description
	InProcess (see page 2075)	This is InProcess, a member of class ProtocolValue.
	Tcp (see page 2075)	This is Tcp, a member of class ProtocolValue.

15.1.2.1.19.2 ProtocolValue Fields

The fields of the ProtocolValue class are listed here.

Public Fields

	Name	Description
	InProcess (see page 2075)	This is InProcess, a member of class ProtocolValue.
	Tcp (see page 2075)	This is Tcp, a member of class ProtocolValue.

15.1.2.1.19.2.1 ProtocolValue.InProcess Field

Pascal

```
const InProcess: string = "InProcess";
```

C++

```
string InProcess = "InProcess";
```

Description

This is InProcess, a member of class ProtocolValue.

15.1.2.1.19.2.2 ProtocolValue.Tcp Field

Pascal

```
const Tcp: string = "TCP";
```

C++

```
string Tcp = "TCP";
```

Description

This is Tcp, a member of class ProtocolValue.

15.1.2.1.20 ScheduleColumns Class

Extends SysMirrors (see page 2089) with additional status information.

Class Hierarchy

Borland.Data.DataStore.ScheduleColumns

Pascal

```
ScheduleColumns = class;
```

C++

```
class ScheduleColumns;
```

File

ScheduleColumns.cs

Description

ScheduleColumns extends SysMirrors (see page 2089) with additional status information on if a mirror is accessible, how recently it was synchronized, and whether enough mirrors are accessible to enable automatic fail over.

This table contains scheduling constants.

Constant	Description
JAVA_METHOD	String name of Java (see page 15681) method to call for the period specified in **PERIOD**, **DAY**, **TIME** and **MILLIS** columns.
PERIOD	A String that is one of the values in the SchedulePeriod (see page 2078) interface.
DAY	Only used when **PERIOD** is set to **MONTH** or **WEEK**.
TIME	Only used when **PERIOD** is set to **MONTH**, **WEEK**, or **DAY**.
MILLIS	Only used when **PERIOD** is set to **MILLIS**.

See Also

SchedulePeriod (see page 2078)

SysMirrors (see page 2089)

15.1.2.1.20.1 ScheduleColumns Members

The following tables list the members exposed by ScheduleColumns.

Public Fields

	Name	Description
	Day (see page 2077)	This is Day, a member of class ScheduleColumns.
	JavaMethod (see page 2077)	This is JavaMethod, a member of class ScheduleColumns.
	Millis (see page 2077)	This is Millis, a member of class ScheduleColumns.
	Period (see page 2077)	This is Period, a member of class ScheduleColumns.
	Time (see page 2078)	This is Time, a member of class ScheduleColumns.

15.1.2.1.20.2 ScheduleColumns Fields

The fields of the ScheduleColumns class are listed here.

Public Fields

	Name	Description
	Day (see page 2077)	This is Day, a member of class ScheduleColumns.
	JavaMethod (see page 2077)	This is JavaMethod, a member of class ScheduleColumns.
	Millis (see page 2077)	This is Millis, a member of class ScheduleColumns.
	Period (see page 2077)	This is Period, a member of class ScheduleColumns.
	Time (see page 2078)	This is Time, a member of class ScheduleColumns.

15.1.2.1.20.2.1 ScheduleColumns.Day Field

Pascal

```
const Day: string = "DAY";
```

C++

```
string Day = "DAY";
```

Description

This is Day, a member of class ScheduleColumns.

15.1.2.1.20.2.2 ScheduleColumns.JavaMethod Field

Pascal

```
const JavaMethod: string = "JAVA_METHOD";
```

C++

```
string JavaMethod = "JAVA_METHOD";
```

Description

This is JavaMethod, a member of class ScheduleColumns.

15.1.2.1.20.2.3 ScheduleColumns.Millis Field

Pascal

```
const Millis: string = "MILLIS";
```

C++

```
string Millis = "MILLIS";
```

Description

This is Millis, a member of class ScheduleColumns.

15.1.2.1.20.2.4 ScheduleColumns.Period Field

Pascal

```
const Period: string = "PERIOD";
```

C++

```
string Period = "PERIOD";
```

Description

This is Period, a member of class ScheduleColumns.

15.1.2.1.20.2.5 ScheduleColumns.Time Field

Pascal

```
const Time: string = "TIME";
```

C++

```
string Time = "TIME";
```

Description

This is Time, a member of class ScheduleColumns.

15.1.2.1.21 SchedulePeriod Class

Possible values for SysSchedule.PERIOD column.

Class Hierarchy

Borland.Data.DataStore.SchedulePeriod

Pascal

```
SchedulePeriod = class;
```

C++

```
class SchedulePeriod;
```

File

SchedulePeriod.cs

Description

SchedulePeriod contains possible values for the SysSchedule.PERIOD column.

Constant	Description
WEEK	Schedule a weekly call to a Java (see page 15681) method.
DAY	Schedule a daily call to a Java (see page 15681) method.
MILLIS	Schedule a call to a Java (see page 15681) method on a millisecond interval.

See Also

SysSchedule (see page 2095)

15.1.2.1.21.1 SchedulePeriod Members

The following tables list the members exposed by SchedulePeriod.

Public Fields

	Name	Description
	Day (see page 2079)	This is Day, a member of class SchedulePeriod.
	Millis (see page 2079)	This is Millis, a member of class SchedulePeriod.
	Week (see page 2079)	This is Week, a member of class SchedulePeriod.

15.1.2.1.21.2 SchedulePeriod Fields

The fields of the SchedulePeriod class are listed here.

Public Fields

	Name	Description
	Day (see page 2079)	This is Day, a member of class SchedulePeriod.
	Millis (see page 2079)	This is Millis, a member of class SchedulePeriod.
	Week (see page 2079)	This is Week, a member of class SchedulePeriod.

15.1.2.1.21.2.1 SchedulePeriod.Day Field

Pascal

```
const Day: string = "DAY";
```

C++

```
string Day = "DAY";
```

Description

This is Day, a member of class SchedulePeriod.

15.1.2.1.21.2.2 SchedulePeriod.Millis Field

Pascal

```
const Millis: string = "MILLIS";
```

C++

```
string Millis = "MILLIS";
```

Description

This is Millis, a member of class SchedulePeriod.

15.1.2.1.21.2.3 SchedulePeriod.Week Field

Pascal

```
const Week: string = "WEEK";
```

C++

```
string Week = "WEEK";
```

Description

This is Week, a member of class SchedulePeriod.

15.1.2.1.22 SqlStateCode Class

Provides the possible values of the SQLState property of a SQLException raised by Blackfish SQL.

Class Hierarchy

Pascal

```
SqlStateCode = class;
```

C++

```
class SqlStateCode;
```

File

SqlStateCode.cs

Description

Provides the possible values of the SQLState property of a SQLException raised by Blackfish SQL.

Each value is a 5 character error code defined in the SQL92 standard. The first 2 characters are an error class value, and the latter 3 characters are a subclass error value.

This table lists the possible error class values (the first two characters in the error code):

Error Class	Description
02	No data.
08	Connection error.
0A	Feature not supported.
21	Cardinality violation.
22	Data (see page 1762) exception.
24	Invalid cursor state.
42	Syntax or access violation.
44	Check option violation.
HZ	Remote database access error.
IM	Non standard error class.

This table lists the 5–character SQL error codes:

Constant	Description
Sql02000 (see page 2082)	Query has no result set.
Sql08001 (see page 2082)	SQL connection could not be made.
Sql08003 (see page 2082)	SQL connection was never made or was previously closed.
Sql08004 (see page 2083)	SQL connection rejected by server.
Sql08006 (see page 2083)	General connection error.
Sql0A000 (see page 2083)	Feature not supported.
Sql21000 (see page 2083)	Cardinality error.
Sql22000 (see page 2083)	General data exception.
Sql22001 (see page 2084)	String data with too many chars/bytes.
Sql22003 (see page 2084)	Numeric value out of range.
Sql22005 (see page 2084)	Value could not be casted to target type.
Sql22007 (see page 2084)	Malformed date/time value.
Sql22011 (see page 2084)	SUBSTRING error.
Sql22012 (see page 2085)	Divide by zero error.
Sql22019 (see page 2085)	Invalid escape character in LIKE operator.
Sql22023 (see page 2085)	Invalid parameter.
Sql22025 (see page 2085)	Invalid LIKE pattern.
Sql22027 (see page 2085)	TRIM error.
Sql23000 (see page 2086)	Integrity constraint violation.
Sql24000 (see page 2086)	Invalid cursor state.

Sql42000 (see page 2086)	Syntax or access violation.
Sql44000 (see page 2086)	With check option violation.
SqlHz309 (see page 2086)	Unknown protocol request encountered.
SqlHz316 (see page 2087)	Network protocol error.
SqlHz320 (see page 2087)	Network protocol version mismatch.
SqlIm000 (see page 2087)	General error category.

15.1.2.1.22.1 SqlStateCode Members

The following tables list the members exposed by SqlStateCode.

Public Fields

	Name	Description
	Sql02000 (see page 2082)	This is Sql02000, a member of class SqlStateCode.
	Sql08001 (see page 2082)	This is Sql08001, a member of class SqlStateCode.
	Sql08003 (see page 2082)	This is Sql08003, a member of class SqlStateCode.
	Sql08004 (see page 2083)	This is Sql08004, a member of class SqlStateCode.
	Sql08006 (see page 2083)	This is Sql08006, a member of class SqlStateCode.
	Sql0A000 (see page 2083)	This is Sql0A000, a member of class SqlStateCode.
	Sql21000 (see page 2083)	This is Sql21000, a member of class SqlStateCode.
	Sql22000 (see page 2083)	This is Sql22000, a member of class SqlStateCode.
	Sql22001 (see page 2084)	This is Sql22001, a member of class SqlStateCode.
	Sql22003 (see page 2084)	This is Sql22003, a member of class SqlStateCode.
	Sql22005 (see page 2084)	This is Sql22005, a member of class SqlStateCode.
	Sql22007 (see page 2084)	This is Sql22007, a member of class SqlStateCode.
	Sql22011 (see page 2084)	This is Sql22011, a member of class SqlStateCode.
	Sql22012 (see page 2085)	This is Sql22012, a member of class SqlStateCode.
	Sql22019 (see page 2085)	This is Sql22019, a member of class SqlStateCode.
	Sql22023 (see page 2085)	This is Sql22023, a member of class SqlStateCode.
	Sql22025 (see page 2085)	This is Sql22025, a member of class SqlStateCode.
	Sql22027 (see page 2085)	This is Sql22027, a member of class SqlStateCode.
	Sql23000 (see page 2086)	This is Sql23000, a member of class SqlStateCode.
	Sql24000 (see page 2086)	This is Sql24000, a member of class SqlStateCode.
	Sql42000 (see page 2086)	This is Sql42000, a member of class SqlStateCode.
	Sql44000 (see page 2086)	This is Sql44000, a member of class SqlStateCode.
	SqlHz309 (see page 2086)	This is SqlHz309, a member of class SqlStateCode.
	SqlHz316 (see page 2087)	This is SqlHz316, a member of class SqlStateCode.
	SqlHz320 (see page 2087)	This is SqlHz320, a member of class SqlStateCode.
	SqlIm000 (see page 2087)	This is SqlIm000, a member of class SqlStateCode.

15.1.2.1.22.2 SqlStateCode Fields

The fields of the SqlStateCode class are listed here.

Public Fields

	Name	Description
	Sql02000 (see page 2082)	This is Sql02000, a member of class SqlStateCode.
	Sql08001 (see page 2082)	This is Sql08001, a member of class SqlStateCode.
	Sql08003 (see page 2082)	This is Sql08003, a member of class SqlStateCode.
	Sql08004 (see page 2083)	This is Sql08004, a member of class SqlStateCode.
	Sql08006 (see page 2083)	This is Sql08006, a member of class SqlStateCode.

	Sql0A000 (see page 2083)	This is Sql0A000, a member of class SqlStateCode.
	Sql21000 (see page 2083)	This is Sql21000, a member of class SqlStateCode.
	Sql22000 (see page 2083)	This is Sql22000, a member of class SqlStateCode.
	Sql22001 (see page 2084)	This is Sql22001, a member of class SqlStateCode.
	Sql22003 (see page 2084)	This is Sql22003, a member of class SqlStateCode.
	Sql22005 (see page 2084)	This is Sql22005, a member of class SqlStateCode.
	Sql22007 (see page 2084)	This is Sql22007, a member of class SqlStateCode.
	Sql22011 (see page 2084)	This is Sql22011, a member of class SqlStateCode.
	Sql22012 (see page 2085)	This is Sql22012, a member of class SqlStateCode.
	Sql22019 (see page 2085)	This is Sql22019, a member of class SqlStateCode.
	Sql22023 (see page 2085)	This is Sql22023, a member of class SqlStateCode.
	Sql22025 (see page 2085)	This is Sql22025, a member of class SqlStateCode.
	Sql22027 (see page 2085)	This is Sql22027, a member of class SqlStateCode.
	Sql23000 (see page 2086)	This is Sql23000, a member of class SqlStateCode.
	Sql24000 (see page 2086)	This is Sql24000, a member of class SqlStateCode.
	Sql42000 (see page 2086)	This is Sql42000, a member of class SqlStateCode.
	Sql44000 (see page 2086)	This is Sql44000, a member of class SqlStateCode.
	SqlHz309 (see page 2086)	This is SqlHz309, a member of class SqlStateCode.
	SqlHz316 (see page 2087)	This is SqlHz316, a member of class SqlStateCode.
	SqlHz320 (see page 2087)	This is SqlHz320, a member of class SqlStateCode.
	SqlIm000 (see page 2087)	This is SqlIm000, a member of class SqlStateCode.

15.1.2.1.22.2.1 SqlStateCode.Sql02000 Field

Pascal

```
const Sql02000: string = "02000";
```

C++

```
string Sql02000 = "02000";
```

Description

This is Sql02000, a member of class SqlStateCode.

15.1.2.1.22.2.2 SqlStateCode.Sql08001 Field

Pascal

```
const Sql08001: string = "08001";
```

C++

```
string Sql08001 = "08001";
```

Description

This is Sql08001, a member of class SqlStateCode.

15.1.2.1.22.2.3 SqlStateCode.Sql08003 Field

Pascal

```
const Sql08003: string = "08003";
```

C++

```
string Sql08003 = "08003";
```

Description

This is Sql08003, a member of class SqlStateCode.

15.1.2.1.22.2.4 SqlStateCode.Sql08004 Field

Pascal

```
const Sql08004: string = "08004";
```

C++

```
string Sql08004 = "08004";
```

Description

This is Sql08004, a member of class SqlStateCode.

15.1.2.1.22.2.5 SqlStateCode.Sql08006 Field

Pascal

```
const Sql08006: string = "08006";
```

C++

```
string Sql08006 = "08006";
```

Description

This is Sql08006, a member of class SqlStateCode.

15.1.2.1.22.2.6 SqlStateCode.Sql0A000 Field

Pascal

```
const Sql0A000: string = "0A000";
```

C++

```
string Sql0A000 = "0A000";
```

Description

This is Sql0A000, a member of class SqlStateCode.

15.1.2.1.22.2.7 SqlStateCode.Sql21000 Field

Pascal

```
const Sql21000: string = "21000";
```

C++

```
string Sql21000 = "21000";
```

Description

This is Sql21000, a member of class SqlStateCode.

15.1.2.1.22.2.8 SqlStateCode.Sql22000 Field

Pascal

```
const Sql22000: string = "22000";
```

C++

```
string Sql22000 = "22000";
```

Description

This is Sql22000, a member of class SqlStateCode.

15.1.2.1.22.2.9 SqlStateCode.Sql22001 Field

Pascal

```
const Sql22001: string = "22001";
```

C++

```
string Sql22001 = "22001";
```

Description

This is Sql22001, a member of class SqlStateCode.

15.1.2.1.22.2.10 SqlStateCode.Sql22003 Field

Pascal

```
const Sql22003: string = "22003";
```

C++

```
string Sql22003 = "22003";
```

Description

This is Sql22003, a member of class SqlStateCode.

15.1.2.1.22.2.11 SqlStateCode.Sql22005 Field

Pascal

```
const Sql22005: string = "22005";
```

C++

```
string Sql22005 = "22005";
```

Description

This is Sql22005, a member of class SqlStateCode.

15.1.2.1.22.2.12 SqlStateCode.Sql22007 Field

Pascal

```
const Sql22007: string = "22007";
```

C++

```
string Sql22007 = "22007";
```

Description

This is Sql22007, a member of class SqlStateCode.

15.1.2.1.22.2.13 SqlStateCode.Sql22011 Field

Pascal

```
const Sql22011: string = "22011";
```

C++

```
string Sql22011 = "22011";
```

Description

This is Sql22011, a member of class SqlStateCode.

15.1.2.1.22.2.14 SqlStateCode.Sql22012 Field

Pascal

```
const Sql22012: string = "22012";
```

C++

```
string Sql22012 = "22012";
```

Description

This is Sql22012, a member of class SqlStateCode.

15.1.2.1.22.2.15 SqlStateCode.Sql22019 Field

Pascal

```
const Sql22019: string = "22019";
```

C++

```
string Sql22019 = "22019";
```

Description

This is Sql22019, a member of class SqlStateCode.

15.1.2.1.22.2.16 SqlStateCode.Sql22023 Field

Pascal

```
const Sql22023: string = "22023";
```

C++

```
string Sql22023 = "22023";
```

Description

This is Sql22023, a member of class SqlStateCode.

15.1.2.1.22.2.17 SqlStateCode.Sql22025 Field

Pascal

```
const Sql22025: string = "22025";
```

C++

```
string Sql22025 = "22025";
```

Description

This is Sql22025, a member of class SqlStateCode.

15.1.2.1.22.2.18 SqlStateCode.Sql22027 Field

Pascal

```
const Sql22027: string = "22027";
```

C++

```
string Sql22027 = "22027";
```

Description

This is Sql22027, a member of class SqlStateCode.

15.1.2.1.22.2.19 SqlStateCode.Sql23000 Field

Pascal

```
const Sql23000: string = "23000";
```

C++

```
string Sql23000 = "23000";
```

Description

This is Sql23000, a member of class SqlStateCode.

15.1.2.1.22.2.20 SqlStateCode.Sql24000 Field

Pascal

```
const Sql24000: string = "24000";
```

C++

```
string Sql24000 = "24000";
```

Description

This is Sql24000, a member of class SqlStateCode.

15.1.2.1.22.2.21 SqlStateCode.Sql42000 Field

Pascal

```
const Sql42000: string = "42000";
```

C++

```
string Sql42000 = "42000";
```

Description

This is Sql42000, a member of class SqlStateCode.

15.1.2.1.22.2.22 SqlStateCode.Sql44000 Field

Pascal

```
const Sql44000: string = "44000";
```

C++

```
string Sql44000 = "44000";
```

Description

This is Sql44000, a member of class SqlStateCode.

15.1.2.1.22.2.23 SqlStateCode.SqlHz309 Field

Pascal

```
const SqlHz309: string = "HZ309";
```

C++

```
string SqlHz309 = "HZ309";
```

Description

This is SqlHz309, a member of class SqlStateCode.

15.1.2.1.22.2.24 SqlStateCode.SqlHz316 Field

Pascal

```
const SqlHz316: string = "HZ316";
```

C++

```
string SqlHz316 = "HZ316";
```

Description

This is SqlHz316, a member of class SqlStateCode.

15.1.2.1.22.2.25 SqlStateCode.SqlHz320 Field

Pascal

```
const SqlHz320: string = "HZ320";
```

C++

```
string SqlHz320 = "HZ320";
```

Description

This is SqlHz320, a member of class SqlStateCode.

15.1.2.1.22.2.26 SqlStateCode.SqlIm000 Field

Pascal

```
const SqlIm000: string = "IM000";
```

C++

```
string SqlIm000 = "IM000";
```

Description

This is SqlIm000, a member of class SqlStateCode.

15.1.2.1.23 StatusLogColumns Class

Provides column names for result table from built-in stored procedure DB_ADMIN.GET_STATUS_LOGS.

Class Hierarchy

Borland.Data.DataStore.StatusLogColumns

Pascal

```
StatusLogColumns = class;
```

C++

```
class StatusLogColumns;
```

File

StatusLogColumns.cs

Description

StatusLogColumns provides the column names for the result table from the built-in stored procedure DB_ADMIN.GET_STATUS_LOGS. Each row in the result table describes an existing status log file for the current database. The columns of the result table are:

Column	Description
ID	A name of a column in the result table from DB_ADMIN.GET_STATUS_LOGS representing the ID of a status log file.
NAME	A name of a column in the result table from DB_ADMIN.GET_STATUS_LOGS representing the name of the status log file.
LAST_MODIFIED	A name of a column in the result table from DB_ADMIN.GET_STATUS_LOGS representing the time the status log file was last changed.
LENGTH	A name of a column in the result table from DB_ADMIN.GET_STATUS_LOGS representing the length of the status log file.

See Also

DB_ADMIN

GET_STATUS_LOGS

15.1.2.1.23.1 StatusLogColumns Members

The following tables list the members exposed by StatusLogColumns.

Public Fields

	Name	Description
	Id (see page 2088)	This is Id, a member of class StatusLogColumns.
	LastModified (see page 2088)	This is LastModified, a member of class StatusLogColumns.
	Length (see page 2089)	This is Length, a member of class StatusLogColumns.
	Name (see page 2089)	This is Name, a member of class StatusLogColumns.

15.1.2.1.23.2 StatusLogColumns Fields

The fields of the StatusLogColumns class are listed here.

Public Fields

	Name	Description
	Id (see page 2088)	This is Id, a member of class StatusLogColumns.
	LastModified (see page 2088)	This is LastModified, a member of class StatusLogColumns.
	Length (see page 2089)	This is Length, a member of class StatusLogColumns.
	Name (see page 2089)	This is Name, a member of class StatusLogColumns.

15.1.2.1.23.2.1 StatusLogColumns.Id Field

Pascal

```
const Id: string = "ID";
```

C++

```
string Id = "ID";
```

Description

This is Id, a member of class StatusLogColumns.

15.1.2.1.23.2.2 StatusLogColumns.LastModified Field

Pascal

```
const LastModified: string = "LAST_MODIFIED";
```

C++

```
string LastModified = "LAST_MODIFIED";
```

Description

This is LastModified, a member of class StatusLogColumns.

15.1.2.1.23.2.3 StatusLogColumns.Length Field

Pascal

```
const Length: string = "LENGTH";
```

C++

```
string Length = "LENGTH";
```

Description

This is Length, a member of class StatusLogColumns.

15.1.2.1.23.2.4 StatusLogColumns.Name Field

Pascal

```
const Name: string = "NAME";
```

C++

```
string Name = "NAME";
```

Description

This is Name, a member of class StatusLogColumns.

15.1.2.1.24 SysMirrors Class

Provides the column names of the system table **SYS.MIRRORS**.

Class Hierarchy

Borland.Data.DataStore.SysMirrors

Pascal

```
SysMirrors = class;
```

C++

```
class SysMirrors;
```

File

SysMirrors.cs

Description

Provides the column names of the system table **SYS.MIRRORS**. In addition to providing the column names for the system table **SYS.MIRRORS**, the same column names should be used to create a property string for the built-in stored procedures DB_ADMIN.CREATE_MIRROR and DB_ADMIN.ALTER_MIRROR. The columns of the system table **SYS.MIRRORS** are:

Column	Description
ID	This is assigned a unique value when a mirror is first created. Although a mirror's name can be changed, its ID can never be changed. If a mirror definition is deleted, its ID is not reused by mirrors created afterwards.

NAME	Unique name for this mirror.
TYPE	Type (see page 2094) of mirror. This can only be set to one of the values in **MirrorTypes (see page 2072)**.
HOST	The host computer name that this mirror can be accessed from. This is needed for remotely connecting to this mirror.
PORT	The port of computer through which this mirror can be accessed. This is needed for remotely connecting to this mirror. Defaults to **DataStoreServer.DEFAULT_PORT**.
FILE_NAME	The file name for this mirror located on the HOST computer
AUTO_FAIL_OVER	This boolean property can be set to true for **READONLY** and **PRIMARY** mirrors. When a **READONLY** mirror has this property set to **true**, all write transactions are guaranteed to be durable on this mirror and the PRIMARY mirror. This allows one of the READONLY mirrors with this property setting to automatically become the PRIMARY mirror if the existing PRIMARY mirror should fail. It is useful to set this property to true for a PRIMARY mirror, so that if the PRIMARY mirror fails and another mirror automatically becomes the PRIMARY, the old PRIMARY knows to assume the role of a READONLY mirror with the **AUTO_FAIL_OVER** option enabled. To prevent more than one PRIMARY mirror existing at one time, a majority of the mirrors that have the **AUTO_FAIL_OVER** column set to **true** must be attachable (connected to). This implies that 3 or more mirrors must be configured with the **AUTO_FAIL_OVER** column set to true. In addition, a PRIMARY mirror does not allow transactions to start or commit if the log records for such a transaction cannot be durably written to a majority of **AUTO_FAIL_OVER** mirrors. The combination of these two rules are needed to prevent automatic fail over from allowing two primary mirrors to simultaneously accept write transactions. Automatic fail over will also be disabled if a majority of **AUTO_FAIL_OVER** mirrors do not have the same contents for the **SYS.MIRRORS** table. This can happen even if the **AUTO_FAIL_OVER** mirrors have the most recent log records from the PRIMARY mirror. Whenever the contents of the **SYS.MIRRORS** table are changed, an attempt is made to synchronize all mirrors. If a mirror is off line it will not be synchronized and the DB_ADMIN.SYNCH_MIRROR method needs to be called when that mirror is brought back on online. Note that in the situation where a majority of **AUTO_FAIL_OVER** mirrors cannot be attached to by the PRIMARY mirror, the PRIMARY mirror can be made writable by setting the **AUTO_FAIL_OVER** column to **false** for the PRIMARY mirror. This disables automatic fail over for the PRIMARY mirror. When a client's connection begins to encounter network errors indicating that the PRIMARY server has failed or there is a network failure, the client initiates an automatic fail over process when its failed transaction is rolled back. This process starts off by connecting to one of the READONLY mirrors with **AUTO_FAIL_OVER** set to **true** that has the lowest value in its FAIL_OVER_PRIORITY column. The highest priority mirror is asked to become the new PRIMARY mirror. If two mirrors have the same priority, then the ID column is used as the tie breaker. Again, this only succeeds if the PRIMARY candidate can connect to a majority of other **AUTO_FAIL_OVER** mirrors that have also lost their connection with the existing PRIMARY mirror.
FAIL_OVER_PRIORITY	Mirrors with a higher priority are chosen for fail over before mirrors with a lower priority setting.
INSTANT_SYNCH	This causes a mirror to have all log records of committed transactions. These mirrors can slow down the write performance of applications, since no transaction can commit until all of the **INSTANT_SYNCH** mirrors have made the transaction durable. **INSTANT_SYNCH** is automatically set to **true** if **AUTO_FAIL_OVER** is set to **true**. However, **INSTANT_SYNCH** can be enabled without enabling **AUTO_FAIL_OVER**. Note that this setting only causes the mirror's log files to be updated with the committed log records from the primary mirror. The database does not reflect these changes until these log records have been replayed by calling the DB_ADMIN.SYNCH_MIRROR method.

SYNCH_MIRROR_NAME	The name of another mirror that this mirror is synchronized from. This can only be set to one of the **NAME** column values of another mirror in the **SYS.MIRRORS** table. Mirrors with **AUTO_FAIL_OVER** do not need to specify a value for this column, since they always synchronize with the primary mirror. Mirrors with either **INSTANT_SYNCH** or **AUTO_FAIL_OVER** set to **true** have their log files kept in sync with their sync mirror on all commit/rollback operations. However the log files are only replayed when DB_ADMIN.SYNCH_MIRROR method is called.
BIRTH_STAMP	Birth stamp used by the primary mirror that this mirror is owned by.
LAST_KNOWN_REPLAY	Last known replay address for this mirror. Used internally to determine if old log files can be dropped.
CLIENT_HASH_CODE	Unique client side hash code.
VALIDATED_PRIMARY	Only meaningful for mirror configurations that have enabled automatic fail over. When set to **true**, this indicates that this mirror has been verified as the primary mirror and can accept write transactions.

See Also

DB_ADMIN

ALTER_MIRROR

CREATE_MIRROR

SYNCH_MIRROR

15.1.2.1.24.1 SysMirrors Members

The following tables list the members exposed by SysMirrors.

Public Fields

	Name	Description
	AutoFailOver (see page 2092)	This is AutoFailOver, a member of class SysMirrors.
	BirthStamp (see page 2092)	This is BirthStamp, a member of class SysMirrors.
	ClientHashCode (see page 2092)	This is ClientHashCode, a member of class SysMirrors.
	FailOverPriority (see page 2092)	This is FailOverPriority, a member of class SysMirrors.
	FileName (see page 2093)	This is FileName, a member of class SysMirrors.
	Host (see page 2093)	This is Host, a member of class SysMirrors.
	Id (see page 2093)	This is Id, a member of class SysMirrors.
	InstantSynch (see page 2093)	This is InstantSynch, a member of class SysMirrors.
	LastKnownReplay (see page 2093)	This is LastKnownReplay, a member of class SysMirrors.
	Name (see page 2094)	This is Name, a member of class SysMirrors.
	Port (see page 2094)	This is Port, a member of class SysMirrors.
	SynchMirrorName (see page 2094)	This is SynchMirrorName, a member of class SysMirrors.
	Type (see page 2094)	This is Type, a member of class SysMirrors.
	ValidatedPrimary (see page 2094)	This is ValidatedPrimary, a member of class SysMirrors.

15.1.2.1.24.2 SysMirrors Fields

The fields of the SysMirrors class are listed here.

Public Fields

	Name	Description
	AutoFailOver (see page 2092)	This is AutoFailOver, a member of class SysMirrors.
	BirthStamp (see page 2092)	This is BirthStamp, a member of class SysMirrors.
	ClientHashCode (see page 2092)	This is ClientHashCode, a member of class SysMirrors.
	FailOverPriority (see page 2092)	This is FailOverPriority, a member of class SysMirrors.

	FileName (see page 2093)	This is FileName, a member of class SysMirrors.
	Host (see page 2093)	This is Host, a member of class SysMirrors.
	Id (see page 2093)	This is Id, a member of class SysMirrors.
	InstantSynch (see page 2093)	This is InstantSynch, a member of class SysMirrors.
	LastKnownReplay (see page 2093)	This is LastKnownReplay, a member of class SysMirrors.
	Name (see page 2094)	This is Name, a member of class SysMirrors.
	Port (see page 2094)	This is Port, a member of class SysMirrors.
	SynchMirrorName (see page 2094)	This is SynchMirrorName, a member of class SysMirrors.
	Type (see page 2094)	This is Type, a member of class SysMirrors.
	ValidatedPrimary (see page 2094)	This is ValidatedPrimary, a member of class SysMirrors.

15.1.2.1.24.2.1 SysMirrors.AutoFailOver Field

Pascal

```
const AutoFailOver: string = "AUTO_FAIL_OVER";
```

C++

```
string AutoFailOver = "AUTO_FAIL_OVER";
```

Description

This is AutoFailOver, a member of class SysMirrors.

15.1.2.1.24.2.2 SysMirrors.BirthStamp Field

Pascal

```
const BirthStamp: string = "BIRTH_STAMP";
```

C++

```
string BirthStamp = "BIRTH_STAMP";
```

Description

This is BirthStamp, a member of class SysMirrors.

15.1.2.1.24.2.3 SysMirrors.ClientHashCode Field

Pascal

```
const ClientHashCode: string = "CLIENT_HASH_CODE";
```

C++

```
string ClientHashCode = "CLIENT_HASH_CODE";
```

Description

This is ClientHashCode, a member of class SysMirrors.

15.1.2.1.24.2.4 SysMirrors.FailOverPriority Field

Pascal

```
const FailOverPriority: string = "FAIL_OVER_PRIORITY";
```

C++

```
string FailOverPriority = "FAIL_OVER_PRIORITY";
```

Description

This is FailOverPriority, a member of class SysMirrors.

15.1.2.1.24.2.5 SysMirrors.FileName Field

Pascal

```
const FileName: string = "FILE_NAME";
```

C++

```
string FileName = "FILE_NAME";
```

Description

This is FileName, a member of class SysMirrors.

15.1.2.1.24.2.6 SysMirrors.Host Field

Pascal

```
const Host: string = "HOST";
```

C++

```
string Host = "HOST";
```

Description

This is Host, a member of class SysMirrors.

15.1.2.1.24.2.7 SysMirrors.Id Field

Pascal

```
const Id: string = "ID";
```

C++

```
string Id = "ID";
```

Description

This is Id, a member of class SysMirrors.

15.1.2.1.24.2.8 SysMirrors.InstantSynch Field

Pascal

```
const InstantSynch: string = "INSTANT_SYNCH";
```

C++

```
string InstantSynch = "INSTANT_SYNCH";
```

Description

This is InstantSynch, a member of class SysMirrors.

15.1.2.1.24.2.9 SysMirrors.LastKnownReplay Field

Pascal

```
const LastKnownReplay: string = "LAST_KNOWN_REPLAY";
```

C++

```
string LastKnownReplay = "LAST_KNOWN_REPLAY";
```

Description

This is LastKnownReplay, a member of class SysMirrors.

15.1.2.1.24.2.10 SysMirrors.Name Field

Pascal

```
const Name: string = "NAME";
```

C++

```
string Name = "NAME";
```

Description

This is Name, a member of class SysMirrors.

15.1.2.1.24.2.11 SysMirrors.Port Field

Pascal

```
const Port: string = "PORT";
```

C++

```
string Port = "PORT";
```

Description

This is Port, a member of class SysMirrors.

15.1.2.1.24.2.12 SysMirrors.SynchMirrorName Field

Pascal

```
const SynchMirrorName: string = "SYNCH_MIRROR_NAME";
```

C++

```
string SynchMirrorName = "SYNCH_MIRROR_NAME";
```

Description

This is SynchMirrorName, a member of class SysMirrors.

15.1.2.1.24.2.13 SysMirrors.Type Field

Pascal

```
const Type: string = "TYPE";
```

C++

```
string Type = "TYPE";
```

Description

This is Type, a member of class SysMirrors.

15.1.2.1.24.2.14 SysMirrors.ValidatedPrimary Field

Pascal

```
const ValidatedPrimary: string = "VALIDATED_PRIMARY";
```

C++

```
string ValidatedPrimary = "VALIDATED_PRIMARY";
```

Description

This is ValidatedPrimary, a member of class SysMirrors.

15.1.2.1.25 SysSchedule Class

Provides column names from system table **SYS.MIRROR_SCHEDULE**.

Class Hierarchy

Borland.Data.DataStore.SysSchedule

Pascal

```
SysSchedule = class;
```

C++

```
class SysSchedule;
```

File

SysSchedule.cs

Description

SysSchedule provides the column names from the system table **SYS.MIRROR_SCHEDULE**. In addition to being the column of the system table **SYS.MIRROR_SCHEDULE**, the same column names should be used to construct a property string for the built-in stored procedure DB_ADMIN.CREATE_MIRROR_SCHEDULE??? and DB_ADMIN.ALTER_MIRROR_SCHEDULE. The columns are:

Constant	Description
ID	Unique identifier **long int** for this schedule item.
REF	**long int** reference to an object that this schedule item applies to.
SCHEDULE_ITEM_CLASS	String that is name of class that extends the **ScheduleItem**. The execute method of this class is called at the scheduled time interval.
PERIOD	String that is one of the values in the SchedulePeriod (see page 2078) class.
DAY	Only used when **PERIOD** is set to **WEEK**.
TIME	Only used when **PERIOD** is set to **WEEK** or **DAY**.
MILLIS	Only used when **PERIOD** is set to **MILLIS**.

See Also

DB_ADMIN

SchedulePeriod (see page 2078)

ALTER_MIRROR_SCHEDULE

CREATE_MIRROR_SCHEDULE

15.1.2.1.25.1 SysSchedule Members

The following tables list the members exposed by SysSchedule.

Public Fields

	Name	Description
	Day (see page 2096)	This is Day, a member of class SysSchedule.
	Id (see page 2096)	This is Id, a member of class SysSchedule.
	Millis (see page 2096)	This is Millis, a member of class SysSchedule.
	Period (see page 2096)	This is Period, a member of class SysSchedule.

	Ref (see page 2097)	This is Ref, a member of class SysSchedule.
	ScheduleItemClass (see page 2097)	This is ScheduleItemClass, a member of class SysSchedule.
	Time (see page 2097)	This is Time, a member of class SysSchedule.

15.1.2.1.25.2 SysSchedule Fields

The fields of the SysSchedule class are listed here.

Public Fields

	Name	Description
	Day (see page 2096)	This is Day, a member of class SysSchedule.
	Id (see page 2096)	This is Id, a member of class SysSchedule.
	Millis (see page 2096)	This is Millis, a member of class SysSchedule.
	Period (see page 2096)	This is Period, a member of class SysSchedule.
	Ref (see page 2097)	This is Ref, a member of class SysSchedule.
	ScheduleItemClass (see page 2097)	This is ScheduleItemClass, a member of class SysSchedule.
	Time (see page 2097)	This is Time, a member of class SysSchedule.

15.1.2.1.25.2.1 SysSchedule.Day Field

Pascal

```
const Day: string = "DAY";
```

C++

```
string Day = "DAY";
```

Description

This is Day, a member of class SysSchedule.

15.1.2.1.25.2.2 SysSchedule.Id Field

Pascal

```
const Id: string = "ID";
```

C++

```
string Id = "ID";
```

Description

This is Id, a member of class SysSchedule.

15.1.2.1.25.2.3 SysSchedule.Millis Field

Pascal

```
const Millis: string = "MILLIS";
```

C++

```
string Millis = "MILLIS";
```

Description

This is Millis, a member of class SysSchedule.

15.1.2.1.25.2.4 SysSchedule.Period Field

Pascal

```
const Period: string = "PERIOD";
```

C++

```
string Period = "PERIOD";
```

Description

This is Period, a member of class SysSchedule.

15.1.2.1.25.2.5 SysSchedule.Ref Field

Pascal

```
const Ref: string = "REF";
```

C++

```
string Ref = "REF";
```

Description

This is Ref, a member of class SysSchedule.

15.1.2.1.25.2.6 SysSchedule.ScheduleItemClass Field

Pascal

```
const ScheduleItemClass: string = "SCHEDULE_ITEM_CLASS";
```

C++

```
string ScheduleItemClass = "SCHEDULE_ITEM_CLASS";
```

Description

This is ScheduleItemClass, a member of class SysSchedule.

15.1.2.1.25.2.7 SysSchedule.Time Field

Pascal

```
const Time: string = "TIME";
```

C++

```
string Time = "TIME";
```

Description

This is Time, a member of class SysSchedule.

15.1.2.1.26 SystemProperties Class

Specifies names of system properties that should be applied to all databases open in a process.

Class Hierarchy

Pascal

```
SystemProperties = class;
```

C++

```
class SystemProperties;
```

File

SystemProperties.cs

Description

This class specifies the names of system properties that should be applied to all databases open in a process. Some of these properties can also be specified as a connection property as well.

Blackfish SQL for Windows calls the `AppDomain.CurrentDomain.GetData` method with the string "blackfishsql." prepended to the property name to obtain system property settings. Here is how to set the default server port:

```
AppDomain.CurrentDomain.SetData("blackfishsql.port", "2508");
```

These settings can also be specified in the launcher config files as:

```
<appSettings> <add key="ServiceName" value="BlackfishSQL" />
            <add key="blackfishsql.port" value="2508" /> <add
            key="blackfishsql.tempDir" value="" /> </appSettings>
```

Blackfish SQL for Java (see page 15681) calls the `java.lang.System.getProperty` method with the string "blackfishsql." prepended to the property name to obtain system property settings. Here is how to set the default server port:

```
System.setProperty("blackfishsql.port", "2508");
```

These settings can also be specified in the launcher config files as:

```
vmparam -Dblackfishsql.port=2508
```

SystemProperties can be specified in the BSQLServer.exe.config file. Here is an example after a clean install:

```
<?xml version="1.0" encoding="utf-8" ?> <configuration>
            <startup> <supportedRuntime version="v2.0.50727" />
            <supportedRuntime version="v1.1.4322" /> <supportedRuntime
            version="v1.0.3705" /> </startup> <appSettings> <add
            key="ServiceName" value="BlackfishSQL" /> <!— Blackfish SQL system properties are
            documented in the Borland.Data.DataStore.SystemProperties class. Empty or missing
values
            cause default settings to be used. --> <add key="blackfishsql.minCacheSize"
            value="" /> <add key="blackfishsql.maxSortBuffer" value="" /> <add
            key="blackfishsql.tempDir" value="" /> <add key="blackfishsql.licenseDirectory"
            value="" /> <add key="blackfishsql.port" value="2508" /> <add
            key="blackfishsql.dataDirectory" value="" /> </appSettings>
            </configuration>
```

This table contains system property names.

System (see page 22079) Property Name	**Description**
MinCacheSize (see page 2101)	Sets the minimum number of cache blocks in the cache. Note that all databases opened in the same process share the same cache.
MaxSortBuffer (see page 2101)	Maximum sort buffer size in bytes. When sorting operations are performed for operations like index building, this is the maximum memory in bytes that is used for a sort buffer. Increasing this number may speed up very large sorts. Cannot be set to less than 32K (32*1024). Default is 12M (12*1024*1024).
TempDir (see page 2101)	Specify directory for temp files. Used for operations like large sorts. If this is not set, the directory of the **DATABASE** property is used for temporary files.
Port (see page 2101)	Specify the port to communicate with a remote server. Default is 2508.

LicenseDirectory (see page 2100)	If this directory location is specified, it is the only directory searched for licenses.
DataDirectory (see page 2100)	If this directory location is specified, this is used to replace the macro `|DataDirectory (see page 2100)|` at the beginning of a database file name specification.

15.1.2.1.26.1 SystemProperties Members

The following tables list the members exposed by SystemProperties.

Public Fields

	Name	Description
	AssemblyPath (see page 2099)	This is AssemblyPath, a member of class SystemProperties.
	BlackfishSqlNamespace (see page 2100)	This is BlackfishSqlNamespace, a member of class SystemProperties.
	DataDirectory (see page 2100)	This is DataDirectory, a member of class SystemProperties.
	LicenseDirectory (see page 2100)	This is LicenseDirectory, a member of class SystemProperties.
	LogFile (see page 2100)	This is LogFile, a member of class SystemProperties.
	LogFilters (see page 2100)	This is LogFilters, a member of class SystemProperties.
	MaxSortBuffer (see page 2101)	This is MaxSortBuffer, a member of class SystemProperties.
	MinCacheSize (see page 2101)	This is MinCacheSize, a member of class SystemProperties.
	Port (see page 2101)	This is Port, a member of class SystemProperties.
	TempDir (see page 2101)	This is TempDir, a member of class SystemProperties.

15.1.2.1.26.2 SystemProperties Fields

The fields of the SystemProperties class are listed here.

Public Fields

	Name	Description
	AssemblyPath (see page 2099)	This is AssemblyPath, a member of class SystemProperties.
	BlackfishSqlNamespace (see page 2100)	This is BlackfishSqlNamespace, a member of class SystemProperties.
	DataDirectory (see page 2100)	This is DataDirectory, a member of class SystemProperties.
	LicenseDirectory (see page 2100)	This is LicenseDirectory, a member of class SystemProperties.
	LogFile (see page 2100)	This is LogFile, a member of class SystemProperties.
	LogFilters (see page 2100)	This is LogFilters, a member of class SystemProperties.
	MaxSortBuffer (see page 2101)	This is MaxSortBuffer, a member of class SystemProperties.
	MinCacheSize (see page 2101)	This is MinCacheSize, a member of class SystemProperties.
	Port (see page 2101)	This is Port, a member of class SystemProperties.
	TempDir (see page 2101)	This is TempDir, a member of class SystemProperties.

15.1.2.1.26.2.1 SystemProperties.AssemblyPath Field

Pascal

```
const AssemblyPath: string = "assemblyPath";
```

C++

```
string AssemblyPath = "assemblyPath";
```

Description

This is AssemblyPath, a member of class SystemProperties.

15.1.2.1.26.2.2 SystemProperties.BlackfishSqlNamespace Field

Pascal

```
const BlackfishSqlNamespace: string = "blackfishsql";
```

C++

```
string BlackfishSqlNamespace = "blackfishsql";
```

Description

This is BlackfishSqlNamespace, a member of class SystemProperties.

15.1.2.1.26.2.3 SystemProperties.DataDirectory Field

Pascal

```
const DataDirectory: string = "dataDirectory";
```

C++

```
string DataDirectory = "dataDirectory";
```

Description

This is DataDirectory, a member of class SystemProperties.

15.1.2.1.26.2.4 SystemProperties.LicenseDirectory Field

Pascal

```
const LicenseDirectory: string = "licenseDirectory";
```

C++

```
string LicenseDirectory = "licenseDirectory";
```

Description

This is LicenseDirectory, a member of class SystemProperties.

15.1.2.1.26.2.5 SystemProperties.LogFile Field

Pascal

```
const LogFile: string = "logFile";
```

C++

```
string LogFile = "logFile";
```

Description

This is LogFile, a member of class SystemProperties.

15.1.2.1.26.2.6 SystemProperties.LogFilters Field

Pascal

```
const LogFilters: string = "logFilters";
```

C++

```
string LogFilters = "logFilters";
```

Description

This is LogFilters, a member of class SystemProperties.

15.1.2.1.26.2.7 SystemProperties.MaxSortBuffer Field

Pascal

```
const MaxSortBuffer: string = "maxSortBuffer";
```

C++

```
string MaxSortBuffer = "maxSortBuffer";
```

Description

This is MaxSortBuffer, a member of class SystemProperties.

15.1.2.1.26.2.8 SystemProperties.MinCacheSize Field

Pascal

```
const MinCacheSize: string = "minCacheSize";
```

C++

```
string MinCacheSize = "minCacheSize";
```

Description

This is MinCacheSize, a member of class SystemProperties.

15.1.2.1.26.2.9 SystemProperties.Port Field

Pascal

```
const Port: string = "port";
```

C++

```
string Port = "port";
```

Description

This is Port, a member of class SystemProperties.

15.1.2.1.26.2.10 SystemProperties.TempDir Field

Pascal

```
const TempDir: string = "tempDir";
```

C++

```
string TempDir = "tempDir";
```

Description

This is TempDir, a member of class SystemProperties.

15.1.2.1.27 TriggerContext Class

An instance of the TriggerContext class is passed to each trigger.

Class Hierarchy

Pascal

```
TriggerContext = class;
```

C++

```
class TriggerContext;
```

File

TriggerContext.cs

Description

An instance of the TriggerContext class is passed to each trigger. From the context, it is possible to get access to a connection instance, and access to the new and old rows.

See Also

TriggerReadRow (see page 2103)

TriggerReadWriteRow (see page 2117)

15.1.2.1.27.1 TriggerContext Members

The following tables list the members exposed by TriggerContext.

Internal Fields

	Name	Description
	method (see page 2102)	This is method, a member of class TriggerContext.

Public Properties

	Name	Description
	Connection (see page 2103)	Current connection performing insert/update/delete.
	NewRow (see page 2103)	New row being inserted or values being updated.
	OldRow (see page 2103)	Old row being deleted or original values before update.

15.1.2.1.27.2 TriggerContext Fields

The fields of the TriggerContext class are listed here.

Internal Fields

	Name	Description
	method (see page 2102)	This is method, a member of class TriggerContext.

15.1.2.1.27.2.1 TriggerContext.method Field

Pascal

```
method: TriggerMethod;
```

C++

```
TriggerMethod method;
```

Description

This is method, a member of class TriggerContext.

15.1.2.1.27.3 TriggerContext Properties

The properties of the TriggerContext class are listed here.

Public Properties

	Name	Description
	Connection (see page 2103)	Current connection performing insert/update/delete.

	NewRow (see page 2103)	New row being inserted or values being updated.
	OldRow (see page 2103)	Old row being deleted or original values before update.

15.1.2.1.27.3.1 TriggerContext.Connection Property

Current connection performing insert/update/delete.

Pascal

```
property Connection: DataStoreConnection;
```

C++

```
__property DataStoreConnection Connection;
```

Description

Connection gets the current connection which is performing the insert/update/delete.

15.1.2.1.27.3.2 TriggerContext.NewRow Property

New row being inserted or values being updated.

Pascal

```
property NewRow: TriggerReadWriteRow;
```

C++

```
__property TriggerReadWriteRow NewRow;
```

Description

NewRow gets access to the new row being inserted or the values being updated.

See Also

OldRow (see page 2103)

15.1.2.1.27.3.3 TriggerContext.OldRow Property

Old row being deleted or original values before update.

Pascal

```
property OldRow: TriggerReadRow;
```

C++

```
__property TriggerReadRow OldRow;
```

Description

OldRow gets access to the old row being deleted or the original values before the update.

See Also

NewRow (see page 2103)

15.1.2.1.28 TriggerReadRow Class

Provides read access to a row of data.

Class Hierarchy

Borland.Data.DataStore.TriggerReadRow

Pascal

```
TriggerReadRow = class;
```

C++

```
class TriggerReadRow;
```

File

TriggerReadRow.cs

Description

The TriggerReadRow class provides read access to a row of data. It has methods to read values from each column.

See Also

TriggerContext (see page 2101)

TriggerReadWriteRow (see page 2117)

15.1.2.1.28.1 TriggerReadRow Members

The following tables list the members exposed by TriggerReadRow.

Internal Fields

	Name	Description
	method (see page 2105)	This is method, a member of class TriggerReadRow.

Public Methods

	Name	Description
	FindModified (see page 2105)	Returns first column ordinal starting with and including **startOrdinal** whose value changed.
	GetArrayLength (see page 2106)	Returns the length of the byte[] value of a given column.
	GetBoolean (see page 2106)	This is the overview for the GetBoolean method overload.
	GetByte (see page 2106)	This is the overview for the GetByte method overload.
	GetBytes (see page 2107)	This is the overview for the GetBytes method overload.
	GetColumnNames (see page 2108)	Returns an array containing names of first **columnCount** column names.
	GetDateTime (see page 2108)	This is the overview for the GetDateTime method overload.
	GetDecimal (see page 2109)	This is the overview for the GetDecimal method overload.
	GetDouble (see page 2109)	This is the overview for the GetDouble method overload.
	GetFloat (see page 2110)	This is the overview for the GetFloat method overload.
	GetInt16 (see page 2110)	This is the overview for the GetInt16 method overload.
	GetInt32 (see page 2111)	This is the overview for the GetInt32 method overload.
	GetInt64 (see page 2112)	This is the overview for the GetInt64 method overload.
	GetObject (see page 2112)	This is the overview for the GetObject method overload.
	GetOrdinal (see page 2113)	Returns ordinal in row of given column.
	GetString (see page 2113)	This is the overview for the GetString method overload.
	IsAssignedNull (see page 2114)	This is the overview for the IsAssignedNull method overload.
	IsModified (see page 2114)	This is the overview for the IsModified method overload.
	IsNull (see page 2115)	This is the overview for the IsNull method overload.
	IsUnassignedNull (see page 2116)	This is the overview for the IsUnassignedNull method overload.
	ToString (see page 2116)	Returns String representation of TriggerReadRow (see page 2103) object.

Public Properties

	Name	Description
	FieldCount (see page 2117)	Gets number of columns in row.

15.1.2.1.28.2 TriggerReadRow Fields

The fields of the TriggerReadRow class are listed here.

Internal Fields

	Name	Description
	method (see page 2105)	This is method, a member of class TriggerReadRow.

15.1.2.1.28.2.1 TriggerReadRow.method Field

Pascal

```
method: TriggerMethod;
```

C++

```
TriggerMethod method;
```

Description

This is method, a member of class TriggerReadRow.

15.1.2.1.28.3 TriggerReadRow Methods

The methods of the TriggerReadRow class are listed here.

Public Methods

	Name	Description
	FindModified (see page 2105)	Returns first column ordinal starting with and including **startOrdinal** whose value changed.
	GetArrayLength (see page 2106)	Returns the length of the byte[] value of a given column.
	GetBoolean (see page 2106)	This is the overview for the GetBoolean method overload.
	GetByte (see page 2106)	This is the overview for the GetByte method overload.
	GetBytes (see page 2107)	This is the overview for the GetBytes method overload.
	GetColumnNames (see page 2108)	Returns an array containing names of first **columnCount** column names.
	GetDateTime (see page 2108)	This is the overview for the GetDateTime method overload.
	GetDecimal (see page 2109)	This is the overview for the GetDecimal method overload.
	GetDouble (see page 2109)	This is the overview for the GetDouble method overload.
	GetFloat (see page 2110)	This is the overview for the GetFloat method overload.
	GetInt16 (see page 2110)	This is the overview for the GetInt16 method overload.
	GetInt32 (see page 2111)	This is the overview for the GetInt32 method overload.
	GetInt64 (see page 2112)	This is the overview for the GetInt64 method overload.
	GetObject (see page 2112)	This is the overview for the GetObject method overload.
	GetOrdinal (see page 2113)	Returns ordinal in row of given column.
	GetString (see page 2113)	This is the overview for the GetString method overload.
	IsAssignedNull (see page 2114)	This is the overview for the IsAssignedNull method overload.
	IsModified (see page 2114)	This is the overview for the IsModified method overload.
	IsNull (see page 2115)	This is the overview for the IsNull method overload.
	IsUnassignedNull (see page 2116)	This is the overview for the IsUnassignedNull method overload.
	ToString (see page 2116)	Returns String representation of TriggerReadRow (see page 2103) object.

15.1.2.1.28.3.1 TriggerReadRow.FindModified Method

Returns first column ordinal starting with and including **startOrdinal** whose value changed.

Pascal

```
function FindModified(startOrdinal: Integer): Integer;
```

C++

```
int FindModified(int startOrdinal);
```

Description

FindModified returns the first column ordinal starting with and including the **startOrdinal** whose value has been changed. The parameter **startOrdinal** is the first ordinal to check.

15.1.2.1.28.3.2 TriggerReadRow.GetArrayLength Method

Returns the length of the byte[] value of a given column.

Pascal

```
function GetArrayLength(columnName: string): Integer;
```

C++

```
int GetArrayLength(string columnName);
```

Description

GetArrayLength returns the length of the byte[] value of a given column whose name is in the parameter **columnName**.

15.1.2.1.28.3.3 GetBoolean Method

This is the overview for the GetBoolean method overload.

Overload List

	Name	Description
	TriggerReadRow.GetBoolean (int) (see page 2106)	Returns boolean value of given column.
	TriggerReadRow.GetBoolean (string) (see page 2106)	Returns boolean value of given column.

15.1.2.1.28.3.3.1 TriggerReadRow.GetBoolean Method (int)

Returns boolean value of given column.

Pascal

```
function GetBoolean(columnName: string): bool;
function GetBoolean(ordinal: Integer): bool;
```

C++

```
bool GetBoolean(string columnName);
bool GetBoolean(int ordinal);
```

Description

GetBoolean has several forms, depending on the input parameter:

- Returns the boolean value of a given column. The parameter **columnName** is the name of the column containing the wanted value.
- Returns the boolean value of a given column. The parameter **ordinal** is the ordinal of the column containing the wanted value.

15.1.2.1.28.3.4 GetByte Method

This is the overview for the GetByte method overload.

Overload List

	Name	Description
	TriggerReadRow.GetByte (int) (see page 2107)	Returns the byte value of a given column.
	TriggerReadRow.GetByte (string) (see page 2107)	Returns the byte value of a given column.

15.1.2.1.28.3.4.1 TriggerReadRow.GetByte Method (int)

Returns the byte value of a given column.

Pascal

```
function GetByte(columnName: string): byte;
function GetByte(ordinal: Integer): byte;
```

C++

```
byte GetByte(string columnName);
byte GetByte(int ordinal);
```

Description

GetByte has several forms, depending on the input parameter:

- Returns the value in the Column named **columnName** as a byte. The parameter **columnName** is the name of the column containing the wanted value.
- Returns the value in the column indicated by its ordinal as a byte. The parameter **ordinal** is the ordinal of the column containing the wanted value.

See Also

GetBytes (see page 2107)

15.1.2.1.28.3.5 GetBytes Method

This is the overview for the GetBytes method overload.

Overload List

	Name	Description
	TriggerReadRow.GetBytes (int) (see page 2107)	Returns the byte[] value of a given column.
	TriggerReadRow.GetBytes (string) (see page 2107)	Returns the byte[] value of a given column.

15.1.2.1.28.3.5.1 TriggerReadRow.GetBytes Method (int)

Returns the byte[] value of a given column.

Pascal

```
function GetBytes(columnName: string): byte[];
function GetBytes(ordinal: Integer): byte[];
```

C++

```
byte[] GetBytes(string columnName);
byte[] GetBytes(int ordinal);
```

Description

GetBytes has several forms, depending on the input parameter:

- Returns the value in the column named **columnName** as a byte[]. The parameter **columnName** is the name of the column containing the wanted value.
- Returns the value in the column indicated by its ordinal as a byte[]. The parameter **ordinal** is the ordinal of the column containing the wanted value.

See Also

GetByte (see page 2107)

15.1.2.1.28.3.6 TriggerReadRow.GetColumnNames Method

Returns an array containing names of first **columnCount** column names.

Pascal

```
function GetColumnNames(columnCount: Integer): string[];
```

C++

```
string[] GetColumnNames(int columnCount);
```

Description

GetColumnNames returns an array containing the names of the first **columnCount** column names. The parameter **columnCount** is the maximum number of column names requested.

15.1.2.1.28.3.7 GetDateTime Method

This is the overview for the GetDateTime method overload.

Overload List

	Name	Description
	TriggerReadRow.GetDateTime (int) (see page 2108)	Returns DateTime value of given column.
	TriggerReadRow.GetDateTime (string) (see page 2108)	Returns DateTime value of given column.

15.1.2.1.28.3.7.1 TriggerReadRow.GetDateTime Method (int)

Returns DateTime value of given column.

Pascal

```
function GetDateTime(columnName: string): DateTime;
function GetDateTime(ordinal: Integer): DateTime;
```

C++

```
DateTime GetDateTime(string columnName);
DateTime GetDateTime(int ordinal);
```

Description

GetDateTime has several forms, depending on the input parameter:

- Returns the DateTime value of a given column. The parameter **columnName** is the name of the column containing the wanted value.
- Returns the DateTime value of a given column. The parameter **ordinal** is the ordinal of the column containing the wanted

value.

15.1.2.1.28.3.8 GetDecimal Method

This is the overview for the GetDecimal method overload.

Overload List

	Name	Description
	TriggerReadRow.GetDecimal (int) (see page 2109)	Returns the Decimal value of a given column.
	TriggerReadRow.GetDecimal (string) (see page 2109)	Returns the Decimal value of a given column.

15.1.2.1.28.3.8.1 TriggerReadRow.GetDecimal Method (int)

Returns the Decimal value of a given column.

Pascal

```
function GetDecimal(columnName: string): Decimal;
function GetDecimal(ordinal: Integer): Decimal;
```

C++

```
Decimal GetDecimal(string columnName);
Decimal GetDecimal(int ordinal);
```

Description

GetDecimal has several forms, depending on the input parameter:

- Returns the Decimal value of a given column. The parameter **columnName** is the name of the column containing the wanted value.
- Returns the Decimal value of a given column. The parameter **ordinal** is the ordinal of the column containing the wanted value.

See Also

GetDouble (see page 2109)

GetFloat (see page 2110)

15.1.2.1.28.3.9 GetDouble Method

This is the overview for the GetDouble method overload.

Overload List

	Name	Description
	TriggerReadRow.GetDouble (int) (see page 2109)	Returns the double value of a given column.
	TriggerReadRow.GetDouble (string) (see page 2109)	Returns the double value of a given column.

15.1.2.1.28.3.9.1 TriggerReadRow.GetDouble Method (int)

Returns the double value of a given column.

Pascal

```
function GetDouble(columnName: string): double;
```

```
function GetDouble(ordinal: Integer): double;
```

C++

```
double GetDouble(string columnName);
double GetDouble(int ordinal);
```

Description

GetDouble has several forms, depending on the input parameter:

- Returns the double value of a given column. The parameter **columnName** is the name of the column containing the wanted value.
- Returns the double value of a given column. The parameter **ordinal** is the ordinal of the column containing the wanted value.

See Also

GetDecimal (see page 2109)

GetFloat (see page 2110)

15.1.2.1.28.3.10 GetFloat Method

This is the overview for the GetFloat method overload.

Overload List

	Name	Description
	TriggerReadRow.GetFloat (int) (see page 2110)	Returns float value of given column.
	TriggerReadRow.GetFloat (string) (see page 2110)	Returns float value of given column.

15.1.2.1.28.3.10.1 TriggerReadRow.GetFloat Method (int)

Returns float value of given column.

Pascal

```
function GetFloat(columnName: string): float;
function GetFloat(ordinal: Integer): float;
```

C++

```
float GetFloat(string columnName);
float GetFloat(int ordinal);
```

Description

GetFloat has several forms, depending on the input parameter:

- Returns the float value of a given column. The parameter **columnName** is the name of the column containing the wanted value.
- Returns the float value of a given column. The parameter **ordinal** is the ordinal of the column containing the wanted value.

See Also

GetDecimal (see page 2109)

GetDouble (see page 2109)

15.1.2.1.28.3.11 GetInt16 Method

This is the overview for the GetInt16 method overload.

Overload List

	Name	Description
	TriggerReadRow.GetInt16 (int) (see page 2111)	Returns short value of given column.
	TriggerReadRow.GetInt16 (string) (see page 2111)	Returns short value of given column.

15.1.2.1.28.3.11.1 TriggerReadRow.GetInt16 Method (int)

Returns short value of given column.

Pascal

```
function GetInt16(columnName: string): short;
function GetInt16(ordinal: Integer): short;
```

C++

```
short GetInt16(string columnName);
short GetInt16(int ordinal);
```

Description

GetInt16 has several forms, depending on the input parameter:

- Returns the short value of a given column. The parameter **columnName** is the name of the column containing the wanted value.
- Returns the short value of a given column. The parameter **ordinal** is the ordinal of the column containing the wanted value.

See Also

GetInt32 (see page 2111)

GetInt64 (see page 2112)

15.1.2.1.28.3.12 GetInt32 Method

This is the overview for the GetInt32 method overload.

Overload List

	Name	Description
	TriggerReadRow.GetInt32 (int) (see page 2111)	Returns int value of given column.
	TriggerReadRow.GetInt32 (string) (see page 2111)	Returns int value of given column.

15.1.2.1.28.3.12.1 TriggerReadRow.GetInt32 Method (int)

Returns int value of given column.

Pascal

```
function GetInt32(columnName: string): Integer;
function GetInt32(ordinal: Integer): Integer;
```

C++

```
int GetInt32(string columnName);
int GetInt32(int ordinal);
```

Description

GetInt32 has several forms, depending on the input parameter:

- Returns the int value of a given column. The parameter **columnName** is the name of the column containing the wanted value.
- Returns the int value of a given column. The parameter **ordinal** is the ordinal of the column containing the wanted value.

See Also

GetInt16 (see page 2111)

GetInt64 (see page 2112)

15.1.2.1.28.3.13 GetInt64 Method

This is the overview for the GetInt64 method overload.

Overload List

	Name	Description
	TriggerReadRow.GetInt64 (int) (see page 2112)	Returns long value of given column.
	TriggerReadRow.GetInt64 (string) (see page 2112)	Returns long value of given column.

15.1.2.1.28.3.13.1 TriggerReadRow.GetInt64 Method (int)

Returns long value of given column.

Pascal

```
function GetInt64(columnName: string): long;
function GetInt64(ordinal: Integer): long;
```

C++

```
long GetInt64(string columnName);
long GetInt64(int ordinal);
```

Description

GetInt64 has several forms, depending on the input parameter:

- Returns the long value of a given column. The parameter **columnName** is the name of the column containing the wanted value.
- Returns the long value of a given column. The parameter **ordinal** is the ordinal of the column containing the wanted value.

See Also

GetInt16 (see page 2111)

GetInt32 (see page 2111)

15.1.2.1.28.3.14 GetObject Method

This is the overview for the GetObject method overload.

Overload List

	Name	Description
	TriggerReadRow.GetObject (int) (see page 2113)	Returns value of given column.

	TriggerReadRow.GetObject (string) (see page 2113)	Returns value of given column.

15.1.2.1.28.3.14.1 TriggerReadRow.GetObject Method (int)

Returns value of given column.

Pascal

```
function GetObject(columnName: string): object;
function GetObject(ordinal: Integer): object;
```

C++

```
object GetObject(string columnName);
object GetObject(int ordinal);
```

Description

GetObject has several forms, depending on the input parameter:

- Returns the value of a given column. The parameter **columnName** is the name of the column containing the wanted value.
- Returns the value of a given column. The parameter **ordinal** is the ordinal of the column containing the wanted value.

15.1.2.1.28.3.15 TriggerReadRow.GetOrdinal Method

Returns ordinal in row of given column.

Pascal

```
function GetOrdinal(columnName: string): Integer;
```

C++

```
int GetOrdinal(string columnName);
```

Description

GetOrdinal returns the ordinal in the row of a given column. The parameter **columnName** is the name of the wanted column.

15.1.2.1.28.3.16 GetString Method

This is the overview for the GetString method overload.

Overload List

	Name	Description
	TriggerReadRow.GetString (int) (see page 2113)	Returns the string value of a given column.
	TriggerReadRow.GetString (string) (see page 2113)	Returns the string value of a given column.

15.1.2.1.28.3.16.1 TriggerReadRow.GetString Method (int)

Returns the string value of a given column.

Pascal

```
function GetString(columnName: string): string;
function GetString(ordinal: Integer): string;
```

C++

```
string GetString(string columnName);
string GetString(int ordinal);
```

Description

GetString has several forms, depending on the input parameter:

- Returns the string value of a given column. The parameter **columnName** is the name of the column containing the wanted value.
- Returns the string value of a given column. The parameter **ordinal** is the ordinal of the column containing the wanted value.

See Also

ToString (see page 2116)

15.1.2.1.28.3.17 IsAssignedNull Method

This is the overview for the IsAssignedNull method overload.

Overload List

	Name	Description
	TriggerReadRow.IsAssignedNull (int) (see page 2114)	Returns **true** if value of given column is assigned null.
	TriggerReadRow.IsAssignedNull (string) (see page 2114)	Returns **true** if value of given column is assigned null.

15.1.2.1.28.3.17.1 TriggerReadRow.IsAssignedNull Method (int)

Returns **true** if value of given column is assigned null.

Pascal

```
function IsAssignedNull(columnName: string): bool;
function IsAssignedNull(ordinal: Integer): bool;
```

C++

```
bool IsAssignedNull(string columnName);
bool IsAssignedNull(int ordinal);
```

Description

IsAssignedNull has several forms, depending on the input parameter:

- Returns **true** if the value in the column named **columnName** is assigned null. The parameter **columnName** is the name of the column containing the wanted value.
- Returns **true** if the value in the column indicated by its ordinal is assigned null. The parameter **ordinal** is the ordinal of the column containing the wanted value.

See Also

IsNull (see page 2115)

IsUnassignedNull (see page 2116)

15.1.2.1.28.3.18 IsModified Method

This is the overview for the IsModified method overload.

Overload List

	Name	Description
	TriggerReadRow.IsModified (int) (see page 2115)	Returns **true** if value of given column has been changed.
	TriggerReadRow.IsModified (string) (see page 2115)	Returns **true** if value of given column has been changed.

15.1.2.1.28.3.18.1 TriggerReadRow.IsModified Method (int)

Returns **true** if value of given column has been changed.

Pascal

```
function IsModified(ordinal: Integer): bool;
function IsModified(columnName: string): bool;
```

C++

```
bool IsModified(int ordinal);
bool IsModified(string columnName);
```

Description

IsModified has several forms, depending on the input parameter:

- Returns **true** if the value of a given column has been changed. The parameter **columnName** is the name of the column in question.
- Returns **true** if the value of a given column has been changed. The parameter **ordinal** is the ordinal of the column in question.

15.1.2.1.28.3.19 IsNull Method

This is the overview for the IsNull method overload.

Overload List

	Name	Description
	TriggerReadRow.IsNull (int) (see page 2115)	Returns **true** if the value of a given column is null.
	TriggerReadRow.IsNull (string) (see page 2115)	Returns **true** if the value of a given column is null.

15.1.2.1.28.3.19.1 TriggerReadRow.IsNull Method (int)

Returns **true** if the value of a given column is null.

Pascal

```
function IsNull(columnName: string): bool;
function IsNull(ordinal: Integer): bool;
```

C++

```
bool IsNull(string columnName);
bool IsNull(int ordinal);
```

Description

IsNull has several forms, depending on the input parameter:

- Returns **true** if the value of a given column is null. The parameter **columnName** is the name of the column containing the wanted value.
- Returns **true** if the value of a given column is NULL. The parameter **ordinal** is the ordinal of the column containing the wanted

value.

See Also

IsAssignedNull (see page 2114)

IsUnassignedNull (see page 2116)

15.1.2.1.28.3.20 IsUnassignedNull Method

This is the overview for the IsUnassignedNull method overload.

Overload List

	Name	Description
	TriggerReadRow.IsUnassignedNull (int) (see page 2116)	Returns **true**if value of given column is unassigned null.
	TriggerReadRow.IsUnassignedNull (string) (see page 2116)	Returns **true**if value of given column is unassigned null.

15.1.2.1.28.3.20.1 TriggerReadRow.IsUnassignedNull Method (int)

Returns **true**if value of given column is unassigned null.

Pascal

```
function IsUnassignedNull(columnName: string): bool;
function IsUnassignedNull(ordinal: Integer): bool;
```

C++

```
bool IsUnassignedNull(string columnName);
bool IsUnassignedNull(int ordinal);
```

Description

IsUnassignedNull has several forms, depending on the input parameter:

- Returns **true** if the value of a given column is an unassigned null. The parameter **columnName** is the name of the column containing the wanted value.
- Returns **true** if the value of a given column is unassigned null. The parameter **ordinal** is the ordinal of the column containing the wanted value.

See Also

IsAssignedNull (see page 2114)

IsNull (see page 2115)

15.1.2.1.28.3.21 TriggerReadRow.ToString Method

Returns String representation of TriggerReadRow (see page 2103) object.

Pascal

```
function ToString: string;
```

C++

```
string ToString();
```

Description

ToString returns a String representation of this TriggerReadRow (see page 2103) object.

See Also

GetString (see page 2113)

15.1.2.1.28.4 TriggerReadRow Properties

The properties of the TriggerReadRow class are listed here.

Public Properties

	Name	Description
	FieldCount (see page 2117)	Gets number of columns in row.

15.1.2.1.28.4.1 TriggerReadRow.FieldCount Property

Gets number of columns in row.

Pascal

```
property FieldCount: Integer;
```

C++

```
__property int FieldCount;
```

Description

FieldCount gets the number of columns in the row.

15.1.2.1.29 TriggerReadWriteRow Class

Provides write access to a row of data.

Class Hierarchy

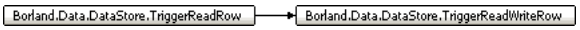

Pascal

```
TriggerReadWriteRow = class(TriggerReadRow);
```

C++

```
class TriggerReadWriteRow : public TriggerReadRow;
```

File

TriggerReadWriteRow.cs

Description

The TriggerReadWriteRow class provides write access to a row of data. It has methods to write values into each column.

See Also

TriggerContext (see page 2101)

TriggerReadRow (see page 2103)

15.1.2.1.29.1 TriggerReadWriteRow Members

The following tables list the members exposed by TriggerReadWriteRow.

Internal Fields

	Name	Description
	method (see page 2105)	This is method, a member of class TriggerReadRow.

Public Methods

	Name	Description
	ClearValues (see page 2119)	Nulls out all values of row.
	FindModified (see page 2105)	Returns first column ordinal starting with and including **startOrdinal** whose value changed.
	GetArrayLength (see page 2106)	Returns the length of the byte[] value of a given column.
	GetBoolean (see page 2106)	This is the overview for the GetBoolean method overload.
	GetByte (see page 2106)	This is the overview for the GetByte method overload.
	GetBytes (see page 2107)	This is the overview for the GetBytes method overload.
	GetColumnNames (see page 2108)	Returns an array containing names of first **columnCount** column names.
	GetDateTime (see page 2108)	This is the overview for the GetDateTime method overload.
	GetDecimal (see page 2109)	This is the overview for the GetDecimal method overload.
	GetDouble (see page 2109)	This is the overview for the GetDouble method overload.
	GetFloat (see page 2110)	This is the overview for the GetFloat method overload.
	GetInt16 (see page 2110)	This is the overview for the GetInt16 method overload.
	GetInt32 (see page 2111)	This is the overview for the GetInt32 method overload.
	GetInt64 (see page 2112)	This is the overview for the GetInt64 method overload.
	GetObject (see page 2112)	This is the overview for the GetObject method overload.
	GetOrdinal (see page 2113)	Returns ordinal in row of given column.
	GetString (see page 2113)	This is the overview for the GetString method overload.
	IsAssignedNull (see page 2114)	This is the overview for the IsAssignedNull method overload.
	IsModified (see page 2114)	This is the overview for the IsModified method overload.
	IsNull (see page 2115)	This is the overview for the IsNull method overload.
	IsUnassignedNull (see page 2116)	This is the overview for the IsUnassignedNull method overload.
	SetAssignedNull (see page 2119)	This is the overview for the SetAssignedNull method overload.
	SetBoolean (see page 2120)	This is the overview for the SetBoolean method overload.
	SetByte (see page 2120)	This is the overview for the SetByte method overload.
	SetBytes (see page 2121)	This is the overview for the SetBytes method overload.
	SetDateTime (see page 2122)	This is the overview for the SetDateTime method overload.
	SetDecimal (see page 2122)	This is the overview for the SetDecimal method overload.
	SetDefault (see page 2123)	This is the overview for the SetDefault method overload.
	SetDefaultValues (see page 2123)	Sets values of all columns to default values.
	SetDouble (see page 2124)	This is the overview for the SetDouble method overload.
	SetFloat (see page 2124)	This is the overview for the SetFloat method overload.
	SetInt16 (see page 2125)	This is the overview for the SetInt16 method overload.
	SetInt32 (see page 2126)	This is the overview for the SetInt32 method overload.
	SetInt64 (see page 2126)	This is the overview for the SetInt64 method overload.
	SetObject (see page 2127)	This is the overview for the SetObject method overload.
	SetString (see page 2128)	This is the overview for the SetString method overload.
	SetUnassignedNull (see page 2128)	This is the overview for the SetUnassignedNull method overload.
	ToString (see page 2116)	Returns String representation of TriggerReadRow (see page 2103) object.

Public Properties

	Name	Description
	FieldCount (see page 2117)	Gets number of columns in row.

15.1.2.1.29.2 TriggerReadWriteRow Methods

The methods of the TriggerReadWriteRow class are listed here.

Public Methods

	Name	Description
	ClearValues (see page 2119)	Nulls out all values of row.
	SetAssignedNull (see page 2119)	This is the overview for the SetAssignedNull method overload.
	SetBoolean (see page 2120)	This is the overview for the SetBoolean method overload.
	SetByte (see page 2120)	This is the overview for the SetByte method overload.
	SetBytes (see page 2121)	This is the overview for the SetBytes method overload.
	SetDateTime (see page 2122)	This is the overview for the SetDateTime method overload.
	SetDecimal (see page 2122)	This is the overview for the SetDecimal method overload.
	SetDefault (see page 2123)	This is the overview for the SetDefault method overload.
V	SetDefaultValues (see page 2123)	Sets values of all columns to default values.
	SetDouble (see page 2124)	This is the overview for the SetDouble method overload.
	SetFloat (see page 2124)	This is the overview for the SetFloat method overload.
	SetInt16 (see page 2125)	This is the overview for the SetInt16 method overload.
	SetInt32 (see page 2126)	This is the overview for the SetInt32 method overload.
	SetInt64 (see page 2126)	This is the overview for the SetInt64 method overload.
	SetObject (see page 2127)	This is the overview for the SetObject method overload.
	SetString (see page 2128)	This is the overview for the SetString method overload.
	SetUnassignedNull (see page 2128)	This is the overview for the SetUnassignedNull method overload.

15.1.2.1.29.2.1 TriggerReadWriteRow.ClearValues Method

Nulls out all values of row.

Pascal

```
function ClearValues: void;
```

C++

```
void ClearValues();
```

Description

ClearValues nulls out all values of the row. It sets them to unassigned nulls.

15.1.2.1.29.2.2 SetAssignedNull Method

This is the overview for the SetAssignedNull method overload.

Overload List

	Name	Description
	TriggerReadWriteRow.SetAssignedNull (int) (see page 2119)	Sets the value of a given column to assigned null.
	TriggerReadWriteRow.SetAssignedNull (string) (see page 2119)	Sets the value of a given column to assigned null.

15.1.2.1.29.2.2.1 TriggerReadWriteRow.SetAssignedNull Method (int)

Sets the value of a given column to assigned null.

Pascal

```
function SetAssignedNull(columnName: string): void;
function SetAssignedNull(ordinal: Integer): void;
```

C++

```
void SetAssignedNull(string columnName);
```

```
void SetAssignedNull(int ordinal);
```

Description

SetAssignedNull has several forms, depending on the input parameter:

- Sets the value of a given column to assigned null. The parameter **columnName** is the name of the column containing the value to be changed.
- Sets the value of a given column to assigned null. The parameter **ordinal** is the ordinal of the column containing the value to be changed.

See Also

SetUnassignedNull (see page 2129)

15.1.2.1.29.2.3 SetBoolean Method

This is the overview for the SetBoolean method overload.

Overload List

	Name	Description
	TriggerReadWriteRow.SetBoolean (int, bool) (see page 2120)	Sets the boolean value of a given column.
	TriggerReadWriteRow.SetBoolean (string, bool) (see page 2120)	Sets the boolean value of a given column.

15.1.2.1.29.2.3.1 TriggerReadWriteRow.SetBoolean Method (int, bool)

Sets the boolean value of a given column.

Pascal

```
function SetBoolean(columnName: string; value: bool): void;
function SetBoolean(ordinal: Integer; value: bool): void;
```

C++

```
void SetBoolean(string columnName, bool value);
void SetBoolean(int ordinal, bool value);
```

Description

SetBoolean has several forms, depending on the input parameter:

- Sets the boolean value of a given column. The parameter **columnName** is the name of the column containing the value to be changed. The parameter **value** is the new value as a boolean.
- Sets the boolean value of a given column. The parameter **ordinal** is the ordinal of the column containing the value to be changed. The parameter **value** is the new value as a boolean.

15.1.2.1.29.2.4 SetByte Method

This is the overview for the SetByte method overload.

Overload List

	Name	Description
	TriggerReadWriteRow.SetByte (int, byte) (see page 2121)	Sets byte value of given column.
	TriggerReadWriteRow.SetByte (string, byte) (see page 2121)	Sets byte value of given column.

15.1.2.1.29.2.4.1 TriggerReadWriteRow.SetByte Method (int, byte)

Sets byte value of given column.

Pascal

```
function SetByte(columnName: string; value: byte): void;
function SetByte(ordinal: Integer; value: byte): void;
```

C++

```
void SetByte(string columnName, byte value);
void SetByte(int ordinal, byte value);
```

Description

SetByte has several forms, depending on the input parameter:

- Sets the byte value of a given column. The parameter **columnName** is the name of the column containing the value to be changed. The parameter **value** is the new value as a byte.
- Sets the byte value of a given column. The parameter **ordinal** is the ordinal of the column containing the value to be changed. The parameter **value** is the new value as a byte.

See Also

SetBytes (see page 2121)

15.1.2.1.29.2.5 SetBytes Method

This is the overview for the SetBytes method overload.

Overload List

	Name	Description
	TriggerReadWriteRow.SetBytes (int, byte[], int) (see page 2121)	Sets the byte[] value of a given column.
	TriggerReadWriteRow.SetBytes (string, byte[], int) (see page 2121)	Sets the byte[] value of a given column.

15.1.2.1.29.2.5.1 TriggerReadWriteRow.SetBytes Method (int, byte[], int)

Sets the byte[] value of a given column.

Pascal

```
function SetBytes(columnName: string; value: byte[]; length: Integer): void;
function SetBytes(ordinal: Integer; value: byte[]; length: Integer): void;
```

C++

```
void SetBytes(string columnName, byte[] value, int length);
void SetBytes(int ordinal, byte[] value, int length);
```

Description

SetBytes has several forms, depending on the input parameters:

- Sets the byte[] value of a given column. The parameter **columnName** is the name of the column containing the value to be changed. The parameter value is the new value as a byte[]. The parameter **length** is the length of the byte[].
- Sets the byte[] value of a given column. The parameter **ordinal** is the ordinal of the column containing the value to be changed. The parameter **value** is the new value as a byte[]. The parameter **length** is the length of the byte[].

See Also

SetByte (see page 2121)

15.1.2.1.29.2.6 SetDateTime Method

This is the overview for the SetDateTime method overload.

Overload List

	Name	Description
	TriggerReadWriteRow.SetDateTime (int, DateTime) (see page 2122)	Sets DateTime value of given column.
	TriggerReadWriteRow.SetDateTime (string, DateTime) (see page 2122)	Sets DateTime value of given column.

15.1.2.1.29.2.6.1 TriggerReadWriteRow.SetDateTime Method (int, DateTime)

Sets DateTime value of given column.

Pascal

```
function SetDateTime(columnName: string; value: DateTime): void;
function SetDateTime(ordinal: Integer; value: DateTime): void;
```

C++

```
void SetDateTime(string columnName, DateTime value);
void SetDateTime(int ordinal, DateTime value);
```

Description

SetDateTime has several forms, depending on the input parameter:

- Sets the DateTime value of a given column. The parameter **columnName** is the name of the column containing the value to be changed. The parameter **value** is the new value as a DateTime.
- Sets the DateTime value of a given column. The parameter **ordinal** is the ordinal of the column containing the value to be changed. The parameter **value** is the new value as a DateTime.

15.1.2.1.29.2.7 SetDecimal Method

This is the overview for the SetDecimal method overload.

Overload List

	Name	Description
	TriggerReadWriteRow.SetDecimal (int, Decimal) (see page 2122)	Sets the Decimal value of a given column.
	TriggerReadWriteRow.SetDecimal (string, Decimal) (see page 2122)	Sets the Decimal value of a given column.

15.1.2.1.29.2.7.1 TriggerReadWriteRow.SetDecimal Method (int, Decimal)

Sets the Decimal value of a given column.

Pascal

```
function SetDecimal(columnName: string; value: Decimal): void;
function SetDecimal(ordinal: Integer; value: Decimal): void;
```

C++

```
void SetDecimal(string columnName, Decimal value);
void SetDecimal(int ordinal, Decimal value);
```

Description

SetDecimal has several forms, depending on the input parameters:

- Sets the Decimal value of a given column. The parameter **columnName** is the name of the column containing the value to be changed. The parameter **value** is the new value as a Decimal.
- Sets the Decimal value of a given column. The parameter **ordinal** is the ordinal of the column containing the value to be changed. The parameter **value** is the new value as a Decimal.

See Also

SetDouble (see page 2124)

SetFloat (see page 2125)

15.1.2.1.29.2.8 SetDefault Method

This is the overview for the SetDefault method overload.

Overload List

	Name	Description
	TriggerReadWriteRow.SetDefault (int) (see page 2123)	Sets the value of a given column to its default value.
	TriggerReadWriteRow.SetDefault (string) (see page 2123)	Sets the value of a given column to its default value.

15.1.2.1.29.2.8.1 TriggerReadWriteRow.SetDefault Method (int)

Sets the value of a given column to its default value.

Pascal

```
function SetDefault(columnName: string): void;
function SetDefault(ordinal: Integer): void;
```

C++

```
void SetDefault(string columnName);
void SetDefault(int ordinal);
```

Description

SetDefault has several forms, depending on the input parameter:

- Sets the value of a given column to its default value. The parameter **columnName** is the name of the column containing the value to be changed.
- Sets the value of a given column to its default value. The parameter **ordinal** is the ordinal of the column containing the value to be changed.

See Also

SetDefaultValues (see page 2123)

15.1.2.1.29.2.9 TriggerReadWriteRow.SetDefaultValues Method

Sets values of all columns to default values.

Pascal

```
function SetDefaultValues: void;
```

C++

```
void SetDefaultValues();
```

Description

SetDefaultValues sets the values of all columns to their default values.

See Also

SetDefault (see page 2123)

15.1.2.1.29.2.10 SetDouble Method

This is the overview for the SetDouble method overload.

Overload List

	Name	Description
	TriggerReadWriteRow.SetDouble (int, double) (see page 2124)	Sets double value of given column.
	TriggerReadWriteRow.SetDouble (string, double) (see page 2124)	Sets double value of given column.

15.1.2.1.29.2.10.1 TriggerReadWriteRow.SetDouble Method (int, double)

Sets double value of given column.

Pascal

```
function SetDouble(columnName: string; value: double): void;
function SetDouble(ordinal: Integer; value: double): void;
```

C++

```
void SetDouble(string columnName, double value);
void SetDouble(int ordinal, double value);
```

Description

SetDouble has several forms, depending on the input parameters:

- Sets the double value of a given column. The parameter **columnName** is the name of the column containing the value to be changed. The parameter **value** is the new value as a double.
- Sets the double value of a given column. The parameter **ordinal** is the ordinal of the column containing the value to be changed. The parameter **value** is the new value as a double.

See Also

SetDecimal (see page 2122)

SetFloat (see page 2125)

15.1.2.1.29.2.11 SetFloat Method

This is the overview for the SetFloat method overload.

Overload List

	Name	Description
	TriggerReadWriteRow.SetFloat (int, float) (see page 2125)	Sets float value of given column.
	TriggerReadWriteRow.SetFloat (string, float) (see page 2125)	Sets float value of given column.

15.1.2.1.29.2.11.1 TriggerReadWriteRow.SetFloat Method (int, float)

Sets float value of given column.

Pascal

```
function SetFloat(columnName: string; value: float): void;
function SetFloat(ordinal: Integer; value: float): void;
```

C++

```
void SetFloat(string columnName, float value);
void SetFloat(int ordinal, float value);
```

Description

SetFloat has several forms, depending on the input parameters:

- Sets the float value of a given column. The parameter **columnName** is the name of the column containing the value to be changed. The parameter **value** is the new value as a float.
- Sets the float value of a given column. The parameter **ordinal** is the ordinal of the column containing the value to be changed. The parameter **value** is the new value as a float.

See Also

SetDecimal (see page 2122)

SetDouble (see page 2124)

15.1.2.1.29.2.12 SetInt16 Method

This is the overview for the SetInt16 method overload.

Overload List

	Name	Description
	TriggerReadWriteRow.SetInt16 (int, short) (see page 2125)	Sets short value of given column.
	TriggerReadWriteRow.SetInt16 (string, short) (see page 2125)	Sets short value of given column.

15.1.2.1.29.2.12.1 TriggerReadWriteRow.SetInt16 Method (int, short)

Sets short value of given column.

Pascal

```
function SetInt16(columnName: string; value: short): void;
function SetInt16(ordinal: Integer; value: short): void;
```

C++

```
void SetInt16(string columnName, short value);
```

```
void SetInt16(int ordinal, short value);
```

Description

SetInt16 has several forms, depending on the input parameters:

- Sets the short value of a given column. The parameter **columnName** is the name of the column containing the value to be changed. The parameter **value** is the new value as a short.
- Sets the short value of a given column. The parameter **ordinal** is the ordinal of the column containing the value to be changed. The parameter **value** is the new value as a short.

See Also

SetInt32 (see page 2126)

SetInt64 (see page 2127)

15.1.2.1.29.2.13 SetInt32 Method

This is the overview for the SetInt32 method overload.

Overload List

	Name	Description
	TriggerReadWriteRow.SetInt32 (int, int) (see page 2126)	Sets int value of given column.
	TriggerReadWriteRow.SetInt32 (string, int) (see page 2126)	Sets int value of given column.

15.1.2.1.29.2.13.1 TriggerReadWriteRow.SetInt32 Method (int, int)

Sets int value of given column.

Pascal

```
function SetInt32(columnName: string; value: Integer): void;
function SetInt32(ordinal: Integer; value: Integer): void;
```

C++

```
void SetInt32(string columnName, int value);
void SetInt32(int ordinal, int value);
```

Description

SetInt32 has several forms, depending on the input parameters:

- Sets the int value of a given column. The parameter **columnName** is the name of the column containing the value to be changed. The parameter **value** is the new value as an int.
- Sets the int value of a given column. The parameter **ordinal** is the ordinal of the column containing the value to be changed. The parameter **value** is the new value as an int.

See Also

SetInt16 (see page 2125)

SetInt64 (see page 2127)

15.1.2.1.29.2.14 SetInt64 Method

This is the overview for the SetInt64 method overload.

Overload List

	Name	Description
	TriggerReadWriteRow.SetInt64 (int, long) (see page 2127)	Sets long value of given column.
	TriggerReadWriteRow.SetInt64 (string, long) (see page 2127)	Sets long value of given column.

15.1.2.1.29.2.14.1 TriggerReadWriteRow.SetInt64 Method (int, long)

Sets long value of given column.

Pascal

```
function SetInt64(columnName: string; value: long): void;
function SetInt64(ordinal: Integer; value: long): void;
```

C++

```
void SetInt64(string columnName, long value);
void SetInt64(int ordinal, long value);
```

Description

SetInt32 (see page 2126) has several forms, depending on the input parameters:

- Sets the long value of a given column. The parameter **columnName** is the name of the column containing the value to be changed. The parameter **value** is the new value as a long.
- Sets the long value of a given column. The parameter **ordinal** is the ordinal of the column containing the value to be changed. The parameter **value** is the new value as a long.

See Also

SetInt16 (see page 2125)

SetInt32 (see page 2126)

15.1.2.1.29.2.15 SetObject Method

This is the overview for the SetObject method overload.

Overload List

	Name	Description
	TriggerReadWriteRow.SetObject (int, object) (see page 2127)	Sets value of given column.
	TriggerReadWriteRow.SetObject (string, object) (see page 2127)	Sets value of given column.

15.1.2.1.29.2.15.1 TriggerReadWriteRow.SetObject Method (int, object)

Sets value of given column.

Pascal

```
function SetObject(columnName: string; value: object): void;
function SetObject(ordinal: Integer; value: object): void;
```

C++

```
void SetObject(string columnName, object value);
```

```
void SetObject(int ordinal, object value);
```

Description

SetObject has several forms, depending on the input parameters:

- Sets the value of a given column. The parameter **columnName** is the name of the column containing the value to be changed. The parameter **value** is the new value.
- Sets the value of a given column. The parameter **ordinal** is the ordinal of the column containing the value to be changed. The parameter **value** is the new value.

15.1.2.1.29.2.16 SetString Method

This is the overview for the SetString method overload.

Overload List

	Name	Description
	TriggerReadWriteRow.SetString (int, string) (see page 2128)	Sets string value of given column.
	TriggerReadWriteRow.SetString (string, string) (see page 2128)	Sets string value of given column.

15.1.2.1.29.2.16.1 TriggerReadWriteRow.SetString Method (int, string)

Sets string value of given column.

Pascal

```
function SetString(columnName: string; value: string): void;
function SetString(ordinal: Integer; value: string): void;
```

C++

```
void SetString(string columnName, string value);
void SetString(int ordinal, string value);
```

Description

SetString has several forms, depending on the input parameters:

- Sets the string value of a given column. The parameter **columnName** is the name of the column containing the value to be changed. The parameter **value** is the new value as a string.
- Sets the string value of a given column. The parameter **ordinal** is the ordinal of the column containing the value to be changed. The parameter **value** is the new value as a string.

15.1.2.1.29.2.17 SetUnassignedNull Method

This is the overview for the SetUnassignedNull method overload.

Overload List

	Name	Description
	TriggerReadWriteRow.SetUnassignedNull (int) (see page 2129)	Sets the value of a given column to unassigned null.
	TriggerReadWriteRow.SetUnassignedNull (string) (see page 2129)	Sets the value of a given column to unassigned null.

15.1.2.1.29.2.17.1 TriggerReadWriteRow.SetUnassignedNull Method (int)

Sets the value of a given column to unassigned null.

Pascal

```
function SetUnassignedNull(columnName: string): void;
function SetUnassignedNull(ordinal: Integer): void;
```

C++

```
void SetUnassignedNull(string columnName);
void SetUnassignedNull(int ordinal);
```

Description

SetUnassignedNull has several forms, depending on the input parameters:

- Sets the value of a given column to unassigned null. The parameter **columnName** is the name of the column containing the value to be changed.
- Sets the value of a given column to unassigned null. The parameter **ordinal** is the ordinal of the column containing the value to be changed.

See Also

SetAssignedNull (see page 2119)

15.1.2.2 Types

The following table lists types in this documentation.

Types

Name	Description
DataStoreRowUpdatedEventHandler (see page 2129)	This is type Borland.Data.DataStore.DataStoreRowUpdatedEventHandler.
DataStoreRowUpdatingEventHandler (see page 2129)	This is type Borland.Data.DataStore.DataStoreRowUpdatingEventHandler.

15.1.2.2.1 Borland.Data.DataStore.DataStoreRowUpdatedEventHandler Type

Pascal

```
DataStoreRowUpdatedEventHandler = ;
```

C++

```
delegate void  DataStoreRowUpdatedEventHandler();
```

File

Ado.Net20.cs

Description

This is type Borland.Data.DataStore.DataStoreRowUpdatedEventHandler.

15.1.2.2.2 Borland.Data.DataStore.DataStoreRowUpdatingEventHandler Type

Pascal

```
DataStoreRowUpdatingEventHandler = ;
```

C++

```
delegate void  DataStoreRowUpdatingEventHandler();
```

File

Ado.Net20.cs

Description

This is type Borland.Data.DataStore.DataStoreRowUpdatingEventHandler.

15.1.3 Borland.Data.MetaData Namespace

This is namespace Borland.Data.MetaData.

Topics

Name	Description
Classes (see page 2131)	The following table lists classes in this documentation.

Classes

	Name	Description
	CatalogsColumns (see page 2132)	Reserved for future use.
	ColumnConstraintsColumns (see page 2133)	Reserved for future use.
	ColumnsColumns (see page 2135)	Column names for result table from built-in stored procedure DB_ADMIN.GET_COLUMNS and DataStoreConnection.GetSchema(MetaDataCollectionName.Columns)
	DataTypesColumns (see page 2140)	Column names for result table from the built-in stored procedure DB_ADMIN.GET_DATATYPES. and DataStoreConnection.GetSchema(MetaDataCollectionName.DataTypes).
	ForeignKeyColumnsColumns (see page 2149)	Column names for result table from the built-in stored procedure DB_ADMIN.GET_FOREIGN_KEY_COLUMNS and DataStoreConnection.GetSchema(MetaDataCollectionName.ForeignKeyColumns).
	ForeignKeysColumns (see page 2153)	Column names for result table from the built-in stored procedure DB_ADMIN.GET_FOREIGN_KEYS and DataStoreConnection.GetSchema(MetaDataCollectionName.ForeignKeys).
	IndexColumnsColumns (see page 2154)	Column names for result table from the built-in stored procedure DB_ADMIN.GET_INDEX_COLUMNS and DataStoreConnection.GetSchema(MetaDataCollectionName.IndexColumns).
	IndexesColumns (see page 2157)	Provides the column names for the result table from the built-in stored procedure DB_ADMIN.GET_INDEXES and DataStoreConnection.GetSchema(MetaDataCollectionName.IndexColumns).
	MetaDataCollectionName (see page 2160)	Collection names that can be passed to DataStoreConnection.GetSchema to receive metadata collections.
	ProcedureParametersColumns (see page 2175)	Column names for result table from the built-in stored procedure DB_ADMIN.GET_PROCEDURE_COLUMNS and DataStoreConnection.GetSchema(MetaDataCollectionName.ProcedureParamete rs).
	ProceduresColumns (see page 2180)	Provides the column names for the result table from the built-in stored procedure DB_ADMIN.GET_PROCEDURES and DataStoreConnection.GetSchema(MetaDataCollectionName.Procedures).
	ProcedureSourcesColumns (see page 2182)	Column names for the result table from the built-in stored procedure DB_ADMIN.GET_PROCEDURES and DataStoreConnection.GetSchema(MetaDataCollectionName.ProcedureSources).
	ReservedWordsColumns (see page 2184)	Column names for result table from the built-in stored procedure DB_ADMIN.GET_KEYWORDS and DataStoreConnection.GetSchema(MetaDataCollectionName.ReservedWords).
	RolesColumns (see page 2185)	Column names for the result table from the built-in stored procedure DB_ADMIN.GET_ROLES and DataStoreConnection.GetSchema(MetaDataCollectionName.Roles).
	SchemasColumns (see page 2187)	Column names for the result table from the built-in stored procedure DB_ADMIN.GET_SCHEMAS and DataStoreConnection.GetSchema(MetaDataCollectionName.Schemas).
	SynonymsColumns (see page 2188)	Reserved for future use.

	TablesColumns (see page 2190)	Column names for the result table from the built-in stored procedure DB_ADMIN.GET_TABLES and DataStoreConnection.GetSchema(MetaDataCollectionName.Tables).
	UsersColumns (see page 2192)	Column names for the result table from the built-in stored procedure DB_ADMIN.GET_USERS and DataStoreConnection.GetSchema(MetaDataCollectionName.Users).
	ViewsColumns (see page 2193)	Column names for the result table from the built-in stored procedure DB_ADMIN.GET_VIEWS DataStoreConnection.GetSchema(MetaDataCollectionName.Views).

15.1.3.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	CatalogsColumns (see page 2132)	Reserved for future use.
	ColumnConstraintsColumns (see page 2133)	Reserved for future use.
	ColumnsColumns (see page 2135)	Column names for result table from built-in stored procedure DB_ADMIN.GET_COLUMNS and DataStoreConnection.GetSchema(MetaDataCollectionName.Columns)
	DataTypesColumns (see page 2140)	Column names for result table from the built-in stored procedure DB_ADMIN.GET_DATATYPES. and DataStoreConnection.GetSchema(MetaDataCollectionName.DataTypes).
	ForeignKeyColumnsColumns (see page 2149)	Column names for result table from the built-in stored procedure DB_ADMIN.GET_FOREIGN_KEY_COLUMNS and DataStoreConnection.GetSchema(MetaDataCollectionName.ForeignKeyColumns).
	ForeignKeysColumns (see page 2153)	Column names for result table from the built-in stored procedure DB_ADMIN.GET_FOREIGN_KEYS and DataStoreConnection.GetSchema(MetaDataCollectionName.ForeignKeys).
	IndexColumnsColumns (see page 2154)	Column names for result table from the built-in stored procedure DB_ADMIN.GET_INDEX_COLUMNS and DataStoreConnection.GetSchema(MetaDataCollectionName.IndexColumns).
	IndexesColumns (see page 2157)	Provides the column names for the result table from the built-in stored procedure DB_ADMIN.GET_INDEXES and DataStoreConnection.GetSchema(MetaDataCollectionName.IndexColumns).
	MetaDataCollectionName (see page 2160)	Collection names that can be passed to DataStoreConnection.GetSchema to receive metadata collections.
	ProcedureParametersColumns (see page 2175)	Column names for result table from the built-in stored procedure DB_ADMIN.GET_PROCEDURE_COLUMNS and DataStoreConnection.GetSchema(MetaDataCollectionName.ProcedureParamete rs).
	ProceduresColumns (see page 2180)	Provides the column names for the result table from the built-in stored procedure DB_ADMIN.GET_PROCEDURES and DataStoreConnection.GetSchema(MetaDataCollectionName.Procedures).
	ProcedureSourcesColumns (see page 2182)	Column names for the result table from the built-in stored procedure DB_ADMIN.GET_PROCEDURES and DataStoreConnection.GetSchema(MetaDataCollectionName.ProcedureSources).
	ReservedWordsColumns (see page 2184)	Column names for result table from the built-in stored procedure DB_ADMIN.GET_KEYWORDS and DataStoreConnection.GetSchema(MetaDataCollectionName.ReservedWords).
	RolesColumns (see page 2185)	Column names for the result table from the built-in stored procedure DB_ADMIN.GET_ROLES and DataStoreConnection.GetSchema(MetaDataCollectionName.Roles).
	SchemasColumns (see page 2187)	Column names for the result table from the built-in stored procedure DB_ADMIN.GET_SCHEMAS and DataStoreConnection.GetSchema(MetaDataCollectionName.Schemas).
	SynonymsColumns (see page 2188)	Reserved for future use.
	TablesColumns (see page 2190)	Column names for the result table from the built-in stored procedure DB_ADMIN.GET_TABLES and DataStoreConnection.GetSchema(MetaDataCollectionName.Tables).

	UsersColumns (see page 2192)	Column names for the result table from the built-in stored procedure DB_ADMIN.GET_USERS and DataStoreConnection.GetSchema(MetaDataCollectionName.Users).
	ViewsColumns (see page 2193)	Column names for the result table from the built-in stored procedure DB_ADMIN.GET_VIEWS DataStoreConnection.GetSchema(MetaDataCollectionName.Views).

15.1.3.1.1 CatalogsColumns Class

Reserved for future use.

Class Hierarchy

Borland.Data.MetaData.CatalogsColumns

Pascal

```
CatalogsColumns = class;
```

C++

```
class CatalogsColumns;
```

File

CatalogsColumns.cs

Description

Reserved for future use.

The columns of the result table are:

Constant	Description
CatalogName (see page 2133)	Used to access the CatalogName (see page 2133) column in the Catalogs metadata collection. The data in this column specifies the catalog name.

See Also

Blackfish SQL Developer's Guide.

15.1.3.1.1.1 CatalogsColumns Members

The following tables list the members exposed by CatalogsColumns.

Public Fields

	Name	Description
	CatalogName (see page 2133)	This is CatalogName, a member of class CatalogsColumns.

15.1.3.1.1.2 CatalogsColumns Fields

The fields of the CatalogsColumns class are listed here.

Public Fields

	Name	Description
	CatalogName (see page 2133)	This is CatalogName, a member of class CatalogsColumns.

15.1.3.1.1.2.1 CatalogsColumns.CatalogName Field

Pascal

```
const CatalogName: string = "CatalogName";
```

C++

```
string CatalogName = "CatalogName";
```

Description

This is CatalogName, a member of class CatalogsColumns.

15.1.3.1.2 ColumnConstraintsColumns Class

Reserved for future use.

Class Hierarchy

Borland.Data.MetaData.ColumnConstraintsColumns

Pascal

```
ColumnConstraintsColumns = class;
```

C++

```
class ColumnConstraints Columns;
```

File

ColumnConstraintsColumns.cs

Description

Reserved for future use.

See Also

Blackfish SQL Developer's Guide.

15.1.3.1.2.1 ColumnConstraintsColumns Members

The following tables list the members exposed by ColumnConstraintsColumns.

Public Fields

	Name	Description
	CatalogName (see page 2134)	This is CatalogName, a member of class ColumnConstraintsColumns.
	ColumnName (see page 2134)	This is ColumnName, a member of class ColumnConstraintsColumns.
	ConstraintName (see page 2134)	This is ConstraintName, a member of class ColumnConstraintsColumns.
	SchemaName (see page 2134)	This is SchemaName, a member of class ColumnConstraintsColumns.
	TableName (see page 2134)	This is TableName, a member of class ColumnConstraintsColumns.

15.1.3.1.2.2 ColumnConstraintsColumns Fields

The fields of the ColumnConstraintsColumns class are listed here.

Public Fields

	Name	Description
	CatalogName (see page 2134)	This is CatalogName, a member of class ColumnConstraintsColumns.
	ColumnName (see page 2134)	This is ColumnName, a member of class ColumnConstraintsColumns.
	ConstraintName (see page 2134)	This is ConstraintName, a member of class ColumnConstraintsColumns.

	SchemaName (see page 2134)	This is SchemaName, a member of class ColumnConstraintsColumns.
	TableName (see page 2134)	This is TableName, a member of class ColumnConstraintsColumns.

15.1.3.1.2.2.1 ColumnConstraintsColumns.CatalogName Field

Pascal

```
const CatalogName: string = "CatalogName";
```

C++

```
string CatalogName = "CatalogName";
```

Description

This is CatalogName, a member of class ColumnConstraintsColumns.

15.1.3.1.2.2.2 ColumnConstraintsColumns.ColumnName Field

Pascal

```
const ColumnName: string = "ColumnName";
```

C++

```
string ColumnName = "ColumnName";
```

Description

This is ColumnName, a member of class ColumnConstraintsColumns.

15.1.3.1.2.2.3 ColumnConstraintsColumns.ConstraintName Field

Pascal

```
const ConstraintName: string = "ConstraintName";
```

C++

```
string ConstraintName = "ConstraintName";
```

Description

This is ConstraintName, a member of class ColumnConstraintsColumns.

15.1.3.1.2.2.4 ColumnConstraintsColumns.SchemaName Field

Pascal

```
const SchemaName: string = "SchemaName";
```

C++

```
string SchemaName = "SchemaName";
```

Description

This is SchemaName, a member of class ColumnConstraintsColumns.

15.1.3.1.2.2.5 ColumnConstraintsColumns.TableName Field

Pascal

```
const TableName: string = "TableName";
```

C++

```
string TableName = "TableName";
```

Description

This is TableName, a member of class ColumnConstraintsColumns.

15.1.3.1.3 ColumnsColumns Class

Column names for result table from built-in stored procedure DB_ADMIN.GET_COLUMNS and DataStoreConnection.GetSchema(MetaDataCollectionName.Columns)

Class Hierarchy

Borland.Data.MetaData.ColumnsColumns

Pascal

```
ColumnsColumns = class;
```

C++

```
class ColumnsColumns;
```

File

ColumnsColumns.cs

Description

ColumnsColumns provides the column names for the result table from the built-in stored procedure DB_ADMIN.GET_COLUMNS and DataStoreConnection.GetSchema(MetaDataCollectionName.Columns). Each row in the result table of DB_ADMIN.GET_COLUMNS describes a column from a table in a database.

The columns of the result table are:

Constant	Description
CatalogName (see page 2137)	Used to access the CatalogName (see page 2137) column in the Columns metadata collection. The data in this column specifies the catalog name.
SchemaName (see page 2140)	Used to access the SchemaName (see page 2140) column in the Columns metadata collection. The data in this column specifies the schema name.
TableName (see page 2140)	Used to access the TableName (see page 2140) column in the Columns metadata collection. The data in this column specifies the table name.
ColumnName (see page 2137)	Used to access the ColumnName (see page 2137) column in the Columns metadata collection. The data in this column specifies the column name.
TypeName (see page 2140)	Used to access the TypeName (see page 2140) column in the Columns metadata collection. The data in this column specifies the database specific data type name.
Precision (see page 2139)	Used to access the Precision (see page 2139) column in the Columns metadata collection.The data in this column specifies the precision of this type. For a string data type, this is the maximum number of characters. For a decimal types, this is the maximum number digits including digits from the scale.
Scale (see page 2140)	Used to access the Scale (see page 2140) column in the Columns metadata collection. The data in this column specifies the scale of a decimal type.
Ordinal (see page 2139)	Used to access the Ordinal (see page 2139) column in the Columns metadata collection. The data in this column specifies the position in the columns of the table starting with 1.

DefaultValue (see page 2138)	Used to access the DefaultValue (see page 2138) column in the Columns metadata collection. The data in this column specifies the default value.
IsNullable (see page 2138)	Used to access the IsNullable (see page 2138) column in the Columns metadata collection. The data in this column specifies that the column can accept NULL values.
IsAutoIncrement (see page 2138)	Used to access the IsAutoIncrement (see page 2138) column in the Columns metadata collection. The data in this column specifies that the column is an autoincrement value.
MaxInline (see page 2139)	Used to access the MaxInline (see page 2139) column in the Columns metadata collection. The data in this column specifies the maximal number of bytes to write into row. The rest of the data is handled as a blob (Blackfish SQL only).
DbxDataType (see page 2137)	Used to access the DbxDataType (see page 2137) column in the Columns metadata collection. The data in this column specifies the DBX data type.
IsFixedLength (see page 2138)	Used to access the IsFixedLength (see page 2138) column in the EndColumns metadata collection.The data in this column specifies that the column holds fixed length data.
IsUnicode (see page 2139)	Used to access the IsUnicode (see page 2139) column in the Columns metadata collection. The data in this column specifies that the column can hold unicode characters.
IsLong (see page 2138)	Used to access the IsLong (see page 2138) column in the Columns metadata collection. The data in this column specifies that the column is a blob data type.
IsUnsigned (see page 2139)	Used to access the IsUnsigned (see page 2139) column in the Columns metadata collection. The data in this column specifies that the column is an unsigned numeric type.

See Also

Blackfish SQL Developer's Guide.

15.1.3.1.3.1 ColumnsColumns Members

The following tables list the members exposed by ColumnsColumns.

Public Fields

	Name	Description
	CatalogName (see page 2137)	This is CatalogName, a member of class ColumnsColumns.
	ColumnName (see page 2137)	This is ColumnName, a member of class ColumnsColumns.
	DbxDataType (see page 2137)	This is DbxDataType, a member of class ColumnsColumns.
	DefaultValue (see page 2138)	This is DefaultValue, a member of class ColumnsColumns.
	IsAutoIncrement (see page 2138)	This is IsAutoIncrement, a member of class ColumnsColumns.
	IsFixedLength (see page 2138)	This is IsFixedLength, a member of class ColumnsColumns.
	IsLong (see page 2138)	This is IsLong, a member of class ColumnsColumns.
	IsNullable (see page 2138)	This is IsNullable, a member of class ColumnsColumns.
	IsUnicode (see page 2139)	This is IsUnicode, a member of class ColumnsColumns.
	IsUnsigned (see page 2139)	This is IsUnsigned, a member of class ColumnsColumns.
	MaxInline (see page 2139)	This is MaxInline, a member of class ColumnsColumns.
	Ordinal (see page 2139)	This is Ordinal, a member of class ColumnsColumns.
	Precision (see page 2139)	This is Precision, a member of class ColumnsColumns.
	Scale (see page 2140)	This is Scale, a member of class ColumnsColumns.
	SchemaName (see page 2140)	This is SchemaName, a member of class ColumnsColumns.
	TableName (see page 2140)	This is TableName, a member of class ColumnsColumns.
	TypeName (see page 2140)	This is TypeName, a member of class ColumnsColumns.

15.1.3.1.3.2 ColumnsColumns Fields

The fields of the ColumnsColumns class are listed here.

Public Fields

	Name	Description
	CatalogName (see page 2137)	This is CatalogName, a member of class ColumnsColumns.
	ColumnName (see page 2137)	This is ColumnName, a member of class ColumnsColumns.
	DbxDataType (see page 2137)	This is DbxDataType, a member of class ColumnsColumns.
	DefaultValue (see page 2138)	This is DefaultValue, a member of class ColumnsColumns.
	IsAutoIncrement (see page 2138)	This is IsAutoIncrement, a member of class ColumnsColumns.
	IsFixedLength (see page 2138)	This is IsFixedLength, a member of class ColumnsColumns.
	IsLong (see page 2138)	This is IsLong, a member of class ColumnsColumns.
	IsNullable (see page 2138)	This is IsNullable, a member of class ColumnsColumns.
	IsUnicode (see page 2139)	This is IsUnicode, a member of class ColumnsColumns.
	IsUnsigned (see page 2139)	This is IsUnsigned, a member of class ColumnsColumns.
	MaxInline (see page 2139)	This is MaxInline, a member of class ColumnsColumns.
	Ordinal (see page 2139)	This is Ordinal, a member of class ColumnsColumns.
	Precision (see page 2139)	This is Precision, a member of class ColumnsColumns.
	Scale (see page 2140)	This is Scale, a member of class ColumnsColumns.
	SchemaName (see page 2140)	This is SchemaName, a member of class ColumnsColumns.
	TableName (see page 2140)	This is TableName, a member of class ColumnsColumns.
	TypeName (see page 2140)	This is TypeName, a member of class ColumnsColumns.

15.1.3.1.3.2.1 ColumnsColumns.CatalogName Field

Pascal

```
const CatalogName: string = "CatalogName";
```

C++

```
string CatalogName = "CatalogName";
```

Description

This is CatalogName, a member of class ColumnsColumns.

15.1.3.1.3.2.2 ColumnsColumns.ColumnName Field

Pascal

```
const ColumnName: string = "ColumnName";
```

C++

```
string ColumnName = "ColumnName";
```

Description

This is ColumnName, a member of class ColumnsColumns.

15.1.3.1.3.2.3 ColumnsColumns.DbxDataType Field

Pascal

```
const DbxDataType: string = "DbxDataType";
```

C++

```
string DbxDataType = "DbxDataType";
```

Description

This is DbxDataType, a member of class ColumnsColumns.

15.1.3.1.3.2.4 ColumnsColumns.DefaultValue Field

Pascal

```
const DefaultValue: string = "DefaultValue";
```

C++

```
string DefaultValue = "DefaultValue";
```

Description

This is DefaultValue, a member of class ColumnsColumns.

15.1.3.1.3.2.5 ColumnsColumns.IsAutoIncrement Field

Pascal

```
const IsAutoIncrement: string = "IsAutoIncrement";
```

C++

```
string IsAutoIncrement = "IsAutoIncrement";
```

Description

This is IsAutoIncrement, a member of class ColumnsColumns.

15.1.3.1.3.2.6 ColumnsColumns.IsFixedLength Field

Pascal

```
const IsFixedLength: string = "IsFixedLength";
```

C++

```
string IsFixedLength = "IsFixedLength";
```

Description

This is IsFixedLength, a member of class ColumnsColumns.

15.1.3.1.3.2.7 ColumnsColumns.IsLong Field

Pascal

```
const IsLong: string = "IsLong";
```

C++

```
string IsLong = "IsLong";
```

Description

This is IsLong, a member of class ColumnsColumns.

15.1.3.1.3.2.8 ColumnsColumns.IsNullable Field

Pascal

```
const IsNullable: string = "IsNullable";
```

C++

```
string IsNullable = "IsNullable";
```

Description

This is IsNullable, a member of class ColumnsColumns.

15.1.3.1.3.2.9 ColumnsColumns.IsUnicode Field

Pascal

```
const IsUnicode: string = "IsUnicode";
```

C++

```
string IsUnicode = "IsUnicode";
```

Description

This is IsUnicode, a member of class ColumnsColumns.

15.1.3.1.3.2.10 ColumnsColumns.IsUnsigned Field

Pascal

```
const IsUnsigned: string = "IsUnsigned";
```

C++

```
string IsUnsigned = "IsUnsigned";
```

Description

This is IsUnsigned, a member of class ColumnsColumns.

15.1.3.1.3.2.11 ColumnsColumns.MaxInline Field

Pascal

```
const MaxInline: string = "MaxInline";
```

C++

```
string MaxInline = "MaxInline";
```

Description

This is MaxInline, a member of class ColumnsColumns.

15.1.3.1.3.2.12 ColumnsColumns.Ordinal Field

Pascal

```
const Ordinal: string = "Ordinal";
```

C++

```
string Ordinal = "Ordinal";
```

Description

This is Ordinal, a member of class ColumnsColumns.

15.1.3.1.3.2.13 ColumnsColumns.Precision Field

Pascal

```
const Precision: string = "Precision";
```

C++

```
string Precision = "Precision";
```

Description

This is Precision, a member of class ColumnsColumns.

15.1.3.1.3.2.14 ColumnsColumns.Scale Field

Pascal

```
const Scale: string = "Scale";
```

C++

```
string Scale = "Scale";
```

Description

This is Scale, a member of class ColumnsColumns.

15.1.3.1.3.2.15 ColumnsColumns.SchemaName Field

Pascal

```
const SchemaName: string = "SchemaName";
```

C++

```
string SchemaName = "SchemaName";
```

Description

This is SchemaName, a member of class ColumnsColumns.

15.1.3.1.3.2.16 ColumnsColumns.TableName Field

Pascal

```
const TableName: string = "TableName";
```

C++

```
string TableName = "TableName";
```

Description

This is TableName, a member of class ColumnsColumns.

15.1.3.1.3.2.17 ColumnsColumns.TypeName Field

Pascal

```
const TypeName: string = "TypeName";
```

C++

```
string TypeName = "TypeName";
```

Description

This is TypeName, a member of class ColumnsColumns.

15.1.3.1.4 DataTypesColumns Class

Column names for result table from the built-in stored procedure DB_ADMIN.GET_DATATYPES. and DataStoreConnection.GetSchema(MetaDataCollectionName.DataTypes).

Class Hierarchy

Borland.Data.MetaData.DataTypesColumns

Pascal

```
DataTypesColumns = class;
```

C++

```
class DataTypesColumns;
```

File

DataTypesColumns.cs

Description

DataTypesColumns provides the column names for the result table from the built-in stored procedure DB_ADMIN.GET_DATATYPES and DataStoreConnection.GetSchema(MetaDataCollectionName.DataTypes).

Each row in the result table of DB_ADMIN.GET_DATATYPES describes a data type. The columns of the result table are:

Constant	Description
TypeName (see page 2148)	Used to access the TypeName (see page 2148) column in the DataTypes metadata collection. The data in this column specifies the database specific data type name.
DbxDataType (see page 2144)	Used to access the DbxDataType (see page 2144) column in the DataTypes metadata collection. The data in this column specifies the DBX data type.
ColumnSize (see page 2143)	Used to access the ColumnSize (see page 2143) column in the DataTypes metadata collection. The data in this column specifies the length of this type. For a string data type, this is the maximum number of characters. For a floating point number, this is the number of significant bits in the mantissa. For other numeric types, this is the maximum number of significant digits. For a date, time, and timestamp types, this is the length of the string representation.
CreateFormat (see page 2144)	Used to access the CreateFormat (see page 2144) column in the DataTypes metadata collection. The data in this column specifies the create format as used in CREATE TABLE.
CreateParameters (see page 2144)	Used to access the CreateParameters (see page 2144) column in the DataTypes metadata collection. The data in this column specifies the parameters for the create format.
DataType (see page 2144)	Used to access the DataType (see page 2144) column in the DataTypes metadata collection. The data in this column specifies the name of the language type representing this data type.
IsAutoIncrementable (see page 2144)	Used to access the IsAutoIncrementable (see page 2144) column in the DataTypes metadata collection. The data in this column specifies that the data type can be used as for an autoincrement column.
IsBestMatch (see page 2145)	Used to access the IsBestMatch (see page 2145) column in the DataTypes metadata collection. The data in this column specifies that the data type is the best match for the language type.
IsCaseSensitive (see page 2145)	Used to access the IsCaseSensitive (see page 2145) column in the DataTypes metadata collection. The data in this column specifies that the data type is case sensitive.
IsFixedLength (see page 2145)	Used to access the IsFixedLength (see page 2145) column in the DataTypes metadata collection. The data in this column specifies that the data type is fixed length.
IsFixedPrecisionScale (see page 2145)	Used to access the IsFixedPrecisionScale (see page 2145) column in the DataTypes metadata collection. The data in this column specifies that the data type is an exact numeric.
IsLong (see page 2146)	Used to access the IsLong (see page 2146) column in the DataTypes metadata collection. The data in this column specifies that the data type is a blob type.
IsNullable (see page 2146)	Used to access the IsNullable (see page 2146) column in the DataTypes metadata collection. The data in this column specifies that the data type can be assigned a value of NULL.

IsSearchable (see page 2146)	Used to access the IsSearchable (see page 2146) column in the DataTypes metadata collection. The data in this column specifies that the data type can be searched in the SQL database.
IsSearchableWithLike (see page 2146)	Used to access the IsSearchableWithLike (see page 2146) column in the DataTypes metadata collection. The data in this column specifies that the data type can be search with the SQL LIKE operator.
IsUnsigned (see page 2147)	Used to access the IsUnsigned (see page 2147) column in the DataTypes metadata collection. The data in this column specifies that the data type is an unsigned numeric.
MaximumScale (see page 2147)	Used to access the MaximumScale (see page 2147) column in the DataTypes metadata collection. The data in this column specifies the maximum possible scale value.
MinimumScale (see page 2148)	Used to access the MinimumScale (see page 2148) column in the DataTypes metadata collection. The data in this column specifies the minimum possible scale value.
IsConcurrencyType (see page 2145)	Used to access the IsConcurrencyType (see page 2145) column in the DataTypes metadata collection. The data in this column specifies that the data type has special update semantics.
MaximumVersion (see page 2148)	Used to access the MaximumVersion (see page 2148) column in the DataTypes metadata collection. The data in this column specifies the last database product version where this type was avaliable.
MinimumVersion (see page 2148)	Used to access the MinimumVersion (see page 2148) column in the DataTypes metadata collection. The data in this column specifies the first database product version where this type was available.
IsLiteralSupported (see page 2146)	Used to access the IsLiteralSupported (see page 2146) column in the DataTypes metadata collection. The data in this column specifies that the data type can be represented as a SQL literal.
LiteralPrefix (see page 2147)	Used to access the LiteralPrefix (see page 2147) column in the DataTypes metadata collection. The data in this column specifies the prefix of a SQL literal.
LiteralSuffix (see page 2147)	Used to access the LiteralSuffix (see page 2147) column in the DataTypes metadata collection. The data in this column specifies the suffix of a SQL literal.
IsUnicode (see page 2147)	Used to access the IsUnicode (see page 2147) column in the DataTypes metadata collection. The data in this column specifies that the data type can hold unicode character data.
ProviderDbType (see page 2148)	Used to access the ProviderDbType (see page 2148) column in the DataTypes metadata collection. The data in this column specifies a SQLDataStore data type from the constants in Borland.Data.DataStore.JdsType. Reserved for .NET applications.

See Also

Blackfish SQL Developer's Guide.

15.1.3.1.4.1 DataTypesColumns Members

The following tables list the members exposed by DataTypesColumns.

Public Fields

	Name	Description
	ColumnSize (see page 2143)	This is ColumnSize, a member of class DataTypesColumns.
	CreateFormat (see page 2144)	This is CreateFormat, a member of class DataTypesColumns.
	CreateParameters (see page 2144)	This is CreateParameters, a member of class DataTypesColumns.
	DataType (see page 2144)	This is DataType, a member of class DataTypesColumns.
	DbxDataType (see page 2144)	This is DbxDataType, a member of class DataTypesColumns.
	IsAutoIncrementable (see page 2144)	This is IsAutoIncrementable, a member of class DataTypesColumns.
	IsBestMatch (see page 2145)	This is IsBestMatch, a member of class DataTypesColumns.
	IsCaseSensitive (see page 2145)	This is IsCaseSensitive, a member of class DataTypesColumns.
	IsConcurrencyType (see page 2145)	This is IsConcurrencyType, a member of class DataTypesColumns.
	IsFixedLength (see page 2145)	This is IsFixedLength, a member of class DataTypesColumns.
	IsFixedPrecisionScale (see page 2145)	This is IsFixedPrecisionScale, a member of class DataTypesColumns.
	IsLiteralSupported (see page 2146)	This is IsLiteralSupported, a member of class DataTypesColumns.

	IsLong (see page 2146)	This is IsLong, a member of class DataTypesColumns.
	IsNullable (see page 2146)	This is IsNullable, a member of class DataTypesColumns.
	IsSearchable (see page 2146)	This is IsSearchable, a member of class DataTypesColumns.
	IsSearchableWithLike (see page 2146)	This is IsSearchableWithLike, a member of class DataTypesColumns.
	IsUnicode (see page 2147)	This is IsUnicode, a member of class DataTypesColumns.
	IsUnsigned (see page 2147)	This is IsUnsigned, a member of class DataTypesColumns.
	LiteralPrefix (see page 2147)	This is LiteralPrefix, a member of class DataTypesColumns.
	LiteralSuffix (see page 2147)	This is LiteralSuffix, a member of class DataTypesColumns.
	MaximumScale (see page 2147)	This is MaximumScale, a member of class DataTypesColumns.
	MaximumVersion (see page 2148)	This is MaximumVersion, a member of class DataTypesColumns.
	MinimumScale (see page 2148)	This is MinimumScale, a member of class DataTypesColumns.
	MinimumVersion (see page 2148)	This is MinimumVersion, a member of class DataTypesColumns.
	ProviderDbType (see page 2148)	This is ProviderDbType, a member of class DataTypesColumns.
	TypeName (see page 2148)	This is TypeName, a member of class DataTypesColumns.

15.1.3.1.4.2 DataTypesColumns Fields

The fields of the DataTypesColumns class are listed here.

Public Fields

	Name	Description
	ColumnSize (see page 2143)	This is ColumnSize, a member of class DataTypesColumns.
	CreateFormat (see page 2144)	This is CreateFormat, a member of class DataTypesColumns.
	CreateParameters (see page 2144)	This is CreateParameters, a member of class DataTypesColumns.
	DataType (see page 2144)	This is DataType, a member of class DataTypesColumns.
	DbxDataType (see page 2144)	This is DbxDataType, a member of class DataTypesColumns.
	IsAutoIncrementable (see page 2144)	This is IsAutoIncrementable, a member of class DataTypesColumns.
	IsBestMatch (see page 2145)	This is IsBestMatch, a member of class DataTypesColumns.
	IsCaseSensitive (see page 2145)	This is IsCaseSensitive, a member of class DataTypesColumns.
	IsConcurrencyType (see page 2145)	This is IsConcurrencyType, a member of class DataTypesColumns.
	IsFixedLength (see page 2145)	This is IsFixedLength, a member of class DataTypesColumns.
	IsFixedPrecisionScale (see page 2145)	This is IsFixedPrecisionScale, a member of class DataTypesColumns.
	IsLiteralSupported (see page 2146)	This is IsLiteralSupported, a member of class DataTypesColumns.
	IsLong (see page 2146)	This is IsLong, a member of class DataTypesColumns.
	IsNullable (see page 2146)	This is IsNullable, a member of class DataTypesColumns.
	IsSearchable (see page 2146)	This is IsSearchable, a member of class DataTypesColumns.
	IsSearchableWithLike (see page 2146)	This is IsSearchableWithLike, a member of class DataTypesColumns.
	IsUnicode (see page 2147)	This is IsUnicode, a member of class DataTypesColumns.
	IsUnsigned (see page 2147)	This is IsUnsigned, a member of class DataTypesColumns.
	LiteralPrefix (see page 2147)	This is LiteralPrefix, a member of class DataTypesColumns.
	LiteralSuffix (see page 2147)	This is LiteralSuffix, a member of class DataTypesColumns.
	MaximumScale (see page 2147)	This is MaximumScale, a member of class DataTypesColumns.
	MaximumVersion (see page 2148)	This is MaximumVersion, a member of class DataTypesColumns.
	MinimumScale (see page 2148)	This is MinimumScale, a member of class DataTypesColumns.
	MinimumVersion (see page 2148)	This is MinimumVersion, a member of class DataTypesColumns.
	ProviderDbType (see page 2148)	This is ProviderDbType, a member of class DataTypesColumns.
	TypeName (see page 2148)	This is TypeName, a member of class DataTypesColumns.

15.1.3.1.4.2.1 DataTypesColumns.ColumnSize Field

Pascal

```
const ColumnSize: string = "ColumnSize";
```

C++

```
string ColumnSize = "ColumnSize";
```

Description

This is ColumnSize, a member of class DataTypesColumns.

15.1.3.1.4.2.2 DataTypesColumns.CreateFormat Field

Pascal

```
const CreateFormat: string = "CreateFormat";
```

C++

```
string CreateFormat = "CreateFormat";
```

Description

This is CreateFormat, a member of class DataTypesColumns.

15.1.3.1.4.2.3 DataTypesColumns.CreateParameters Field

Pascal

```
const CreateParameters: string = "CreateParameters";
```

C++

```
string CreateParameters = "CreateParameters";
```

Description

This is CreateParameters, a member of class DataTypesColumns.

15.1.3.1.4.2.4 DataTypesColumns.DataType Field

Pascal

```
const DataType: string = "DataType";
```

C++

```
string DataType = "DataType";
```

Description

This is DataType, a member of class DataTypesColumns.

15.1.3.1.4.2.5 DataTypesColumns.DbxDataType Field

Pascal

```
const DbxDataType: string = "DbxDataType";
```

C++

```
string DbxDataType = "DbxDataType";
```

Description

This is DbxDataType, a member of class DataTypesColumns.

15.1.3.1.4.2.6 DataTypesColumns.IsAutoIncrementable Field

Pascal

```
const IsAutoIncrementable: string = "IsAutoIncrementable";
```

C++

```
string IsAutoIncrementable = "IsAutoIncrementable";
```

Description

This is IsAutoIncrementable, a member of class DataTypesColumns.

15.1.3.1.4.2.7 DataTypesColumns.IsBestMatch Field

Pascal

```
const IsBestMatch: string = "IsBestMatch";
```

C++

```
string IsBestMatch = "IsBestMatch";
```

Description

This is IsBestMatch, a member of class DataTypesColumns.

15.1.3.1.4.2.8 DataTypesColumns.IsCaseSensitive Field

Pascal

```
const IsCaseSensitive: string = "IsCaseSensitive";
```

C++

```
string IsCaseSensitive = "IsCaseSensitive";
```

Description

This is IsCaseSensitive, a member of class DataTypesColumns.

15.1.3.1.4.2.9 DataTypesColumns.IsConcurrencyType Field

Pascal

```
const IsConcurrencyType: string = "IsConcurrencyType";
```

C++

```
string IsConcurrencyType = "IsConcurrencyType";
```

Description

This is IsConcurrencyType, a member of class DataTypesColumns.

15.1.3.1.4.2.10 DataTypesColumns.IsFixedLength Field

Pascal

```
const IsFixedLength: string = "IsFixedLength";
```

C++

```
string IsFixedLength = "IsFixedLength";
```

Description

This is IsFixedLength, a member of class DataTypesColumns.

15.1.3.1.4.2.11 DataTypesColumns.IsFixedPrecisionScale Field

Pascal

```
const IsFixedPrecisionScale: string = "IsFixedPrecisionScale";
```

C++

```
string IsFixedPrecisionScale = "IsFixedPrecisionScale";
```

Description

This is IsFixedPrecisionScale, a member of class DataTypesColumns.

15.1.3.1.4.2.12 DataTypesColumns.IsLiteralSupported Field

Pascal

```
const IsLiteralSupported: string = "IsLiteralSupported";
```

C++

```
string IsLiteralSupported = "IsLiteralSupported";
```

Description

This is IsLiteralSupported, a member of class DataTypesColumns.

15.1.3.1.4.2.13 DataTypesColumns.IsLong Field

Pascal

```
const IsLong: string = "IsLong";
```

C++

```
string IsLong = "IsLong";
```

Description

This is IsLong, a member of class DataTypesColumns.

15.1.3.1.4.2.14 DataTypesColumns.IsNullable Field

Pascal

```
const IsNullable: string = "IsNullable";
```

C++

```
string IsNullable = "IsNullable";
```

Description

This is IsNullable, a member of class DataTypesColumns.

15.1.3.1.4.2.15 DataTypesColumns.IsSearchable Field

Pascal

```
const IsSearchable: string = "IsSearchable";
```

C++

```
string IsSearchable = "IsSearchable";
```

Description

This is IsSearchable, a member of class DataTypesColumns.

15.1.3.1.4.2.16 DataTypesColumns.IsSearchableWithLike Field

Pascal

```
const IsSearchableWithLike: string = "IsSearchableWithLike";
```

C++

```
string IsSearchableWithLike = "IsSearchableWithLike";
```

Description

This is IsSearchableWithLike, a member of class DataTypesColumns.

15.1.3.1.4.2.17 DataTypesColumns.IsUnicode Field

Pascal

```
const IsUnicode: string = "IsUnicode";
```

C++

```
string IsUnicode = "IsUnicode";
```

Description

This is IsUnicode, a member of class DataTypesColumns.

15.1.3.1.4.2.18 DataTypesColumns.IsUnsigned Field

Pascal

```
const IsUnsigned: string = "IsUnsigned";
```

C++

```
string IsUnsigned = "IsUnsigned";
```

Description

This is IsUnsigned, a member of class DataTypesColumns.

15.1.3.1.4.2.19 DataTypesColumns.LiteralPrefix Field

Pascal

```
const LiteralPrefix: string = "LiteralPrefix";
```

C++

```
string LiteralPrefix = "LiteralPrefix";
```

Description

This is LiteralPrefix, a member of class DataTypesColumns.

15.1.3.1.4.2.20 DataTypesColumns.LiteralSuffix Field

Pascal

```
const LiteralSuffix: string = "LiteralSuffix";
```

C++

```
string LiteralSuffix = "LiteralSuffix";
```

Description

This is LiteralSuffix, a member of class DataTypesColumns.

15.1.3.1.4.2.21 DataTypesColumns.MaximumScale Field

Pascal

```
const MaximumScale: string = "MaximumScale";
```

C++

```
string MaximumScale = "MaximumScale";
```

Description

This is MaximumScale, a member of class DataTypesColumns.

15.1.3.1.4.2.22 DataTypesColumns.MaximumVersion Field

Pascal

```
const MaximumVersion: string = "MaximumVersion";
```

C++

```
string MaximumVersion = "MaximumVersion";
```

Description

This is MaximumVersion, a member of class DataTypesColumns.

15.1.3.1.4.2.23 DataTypesColumns.MinimumScale Field

Pascal

```
const MinimumScale: string = "MinimumScale";
```

C++

```
string MinimumScale = "MinimumScale";
```

Description

This is MinimumScale, a member of class DataTypesColumns.

15.1.3.1.4.2.24 DataTypesColumns.MinimumVersion Field

Pascal

```
const MinimumVersion: string = "MinimumVersion";
```

C++

```
string MinimumVersion = "MinimumVersion";
```

Description

This is MinimumVersion, a member of class DataTypesColumns.

15.1.3.1.4.2.25 DataTypesColumns.ProviderDbType Field

Pascal

```
const ProviderDbType: string = "ProviderDbType";
```

C++

```
string ProviderDbType = "ProviderDbType";
```

Description

This is ProviderDbType, a member of class DataTypesColumns.

15.1.3.1.4.2.26 DataTypesColumns.TypeName Field

Pascal

```
const TypeName: string = "TypeName";
```

C++

```
string TypeName = "TypeName";
```

Description

This is TypeName, a member of class DataTypesColumns.

15.1.3.1.5 ForeignKeyColumnsColumns Class

Column names for result table from the built-in stored procedure DB_ADMIN.GET_FOREIGN_KEY_COLUMNS and DataStoreConnection.GetSchema(MetaDataCollectionName.ForeignKeyColumns).

Class Hierarchy

Borland.Data.MetaData.ForeignKeyColumnsColumns

Pascal

```
ForeignKeyColumnsColumns = class;
```

C++

```
class ForeignKeyColumnsColumns;
```

File

ForeignKeyColumnsColumns.cs

Description

Column names for result table from the built-in stored procedure DB_ADMIN.GET_FOREIGN_KEY_COLUMNS and DataStoreConnection.GetSchema(MetaDataCollectionName.ForeignKeyColumns).

The columns of the result table are:

Constant	Description
CatalogName (see page 2150)	Used to access the CatalogName (see page 2150) column in the ForeignKeyColumns metadata collection. The data in this column specifies the catalog name of the table with the foreign key.
SchemaName (see page 2152)	Used to access the SchemaName (see page 2152) column in the ForeignKeyColumns metadata collection. The data in this column specifies the schema name of the table with the foreign key.
TableName (see page 2152)	Used to access the TableName (see page 2152) column in the ForeignKeyColumns metadata collection. The data in this column specifies the name of the table with the foreign key.
ForeignKeyName (see page 2151)	Used to access the ForeignKeyName (see page 2151) column in the ForeignKeyColumns metadata collection. The data in this column specifies the foreign key name.
ColumnName (see page 2151)	Used to access the ColumnName (see page 2151) column in the ForeignKeyColumns metadata collection. The data in this column specifies a column which refers to a column of in another table.
PrimaryCatalogName (see page 2151)	Used to access the PrimaryCatalogName (see page 2151) column in the ForeignKeyColumns metadata collection. The data in this column specifies the catalog name of the table being referred to.
PrimarySchemaName (see page 2152)	Used to access the PrimarySchemaName (see page 2152) column in the ForeignKeyColumns metadata collection. The data in this column specifies the schema name of the table being referred to.
PrimaryTableName (see page 2152)	Used to access the PrimaryTableName (see page 2152) column in the ForeignKeyColumns metadata collection. The data in this column specifies the name of the table being referred to.
PrimaryKeyName (see page 2152)	Used to access the PrimaryKeyName (see page 2152) column in the ForeignKeyColumns metadata collection. The data in this column specifies the name of the key being referred to. This is usually the primary key.

PrimaryColumnName (see page 2151)	Used to access the PrimaryColumnName (see page 2151) column in the ForeignKeyColumns metadata collection. The data in this column specifies the column which is being referred to.
Ordinal (see page 2151)	Used to access the Ordinal (see page 2151) column in the ForeignKeyColumns metadata collection. The data in this column specifies the position of the column reference in the foreign key.

See Also

Blackfish SQL Developer's Guide.

15.1.3.1.5.1 ForeignKeyColumnsColumns Members

The following tables list the members exposed by ForeignKeyColumnsColumns.

Public Fields

	Name	Description
	CatalogName (see page 2150)	This is CatalogName, a member of class ForeignKeyColumnsColumns.
	ColumnName (see page 2151)	This is ColumnName, a member of class ForeignKeyColumnsColumns.
	ForeignKeyName (see page 2151)	This is ForeignKeyName, a member of class ForeignKeyColumnsColumns.
	Ordinal (see page 2151)	This is Ordinal, a member of class ForeignKeyColumnsColumns.
	PrimaryCatalogName (see page 2151)	This is PrimaryCatalogName, a member of class ForeignKeyColumnsColumns.
	PrimaryColumnName (see page 2151)	This is PrimaryColumnName, a member of class ForeignKeyColumnsColumns.
	PrimaryKeyName (see page 2152)	This is PrimaryKeyName, a member of class ForeignKeyColumnsColumns.
	PrimarySchemaName (see page 2152)	This is PrimarySchemaName, a member of class ForeignKeyColumnsColumns.
	PrimaryTableName (see page 2152)	This is PrimaryTableName, a member of class ForeignKeyColumnsColumns.
	SchemaName (see page 2152)	This is SchemaName, a member of class ForeignKeyColumnsColumns.
	TableName (see page 2152)	This is TableName, a member of class ForeignKeyColumnsColumns.

15.1.3.1.5.2 ForeignKeyColumnsColumns Fields

The fields of the ForeignKeyColumnsColumns class are listed here.

Public Fields

	Name	Description
	CatalogName (see page 2150)	This is CatalogName, a member of class ForeignKeyColumnsColumns.
	ColumnName (see page 2151)	This is ColumnName, a member of class ForeignKeyColumnsColumns.
	ForeignKeyName (see page 2151)	This is ForeignKeyName, a member of class ForeignKeyColumnsColumns.
	Ordinal (see page 2151)	This is Ordinal, a member of class ForeignKeyColumnsColumns.
	PrimaryCatalogName (see page 2151)	This is PrimaryCatalogName, a member of class ForeignKeyColumnsColumns.
	PrimaryColumnName (see page 2151)	This is PrimaryColumnName, a member of class ForeignKeyColumnsColumns.
	PrimaryKeyName (see page 2152)	This is PrimaryKeyName, a member of class ForeignKeyColumnsColumns.
	PrimarySchemaName (see page 2152)	This is PrimarySchemaName, a member of class ForeignKeyColumnsColumns.
	PrimaryTableName (see page 2152)	This is PrimaryTableName, a member of class ForeignKeyColumnsColumns.
	SchemaName (see page 2152)	This is SchemaName, a member of class ForeignKeyColumnsColumns.
	TableName (see page 2152)	This is TableName, a member of class ForeignKeyColumnsColumns.

15.1.3.1.5.2.1 ForeignKeyColumnsColumns.CatalogName Field

Pascal

```
const CatalogName: string = "CatalogName";
```

C++

```
string CatalogName = "CatalogName";
```

Description

This is CatalogName, a member of class ForeignKeyColumnsColumns.

15.1.3.1.5.2.2 ForeignKeyColumnsColumns.ColumnName Field

Pascal

```
const ColumnName: string = "ColumnName";
```

C++

```
string ColumnName = "ColumnName";
```

Description

This is ColumnName, a member of class ForeignKeyColumnsColumns.

15.1.3.1.5.2.3 ForeignKeyColumnsColumns.ForeignKeyName Field

Pascal

```
const ForeignKeyName: string = "ForeignKeyName";
```

C++

```
string ForeignKeyName = "ForeignKeyName";
```

Description

This is ForeignKeyName, a member of class ForeignKeyColumnsColumns.

15.1.3.1.5.2.4 ForeignKeyColumnsColumns.Ordinal Field

Pascal

```
const Ordinal: string = "Ordinal";
```

C++

```
string Ordinal = "Ordinal";
```

Description

This is Ordinal, a member of class ForeignKeyColumnsColumns.

15.1.3.1.5.2.5 ForeignKeyColumnsColumns.PrimaryCatalogName Field

Pascal

```
const PrimaryCatalogName: string = "PrimaryCatalogName";
```

C++

```
string PrimaryCatalogName = "PrimaryCatalogName";
```

Description

This is PrimaryCatalogName, a member of class ForeignKeyColumnsColumns.

15.1.3.1.5.2.6 ForeignKeyColumnsColumns.PrimaryColumnName Field

Pascal

```
const PrimaryColumnName: string = "PrimaryColumnName";
```

C++

```
string PrimaryColumnName = "PrimaryColumnName";
```

Description

This is PrimaryColumnName, a member of class ForeignKeyColumnsColumns.

15.1.3.1.5.2.7 ForeignKeyColumnsColumns.PrimaryKeyName Field

Pascal

```
const PrimaryKeyName: string = "PrimaryKeyName";
```

C++

```
string PrimaryKeyName = "PrimaryKeyName";
```

Description

This is PrimaryKeyName, a member of class ForeignKeyColumnsColumns.

15.1.3.1.5.2.8 ForeignKeyColumnsColumns.PrimarySchemaName Field

Pascal

```
const PrimarySchemaName: string = "PrimarySchemaName";
```

C++

```
string PrimarySchemaName = "PrimarySchemaName";
```

Description

This is PrimarySchemaName, a member of class ForeignKeyColumnsColumns.

15.1.3.1.5.2.9 ForeignKeyColumnsColumns.PrimaryTableName Field

Pascal

```
const PrimaryTableName: string = "PrimaryTableName";
```

C++

```
string PrimaryTableName = "PrimaryTableName";
```

Description

This is PrimaryTableName, a member of class ForeignKeyColumnsColumns.

15.1.3.1.5.2.10 ForeignKeyColumnsColumns.SchemaName Field

Pascal

```
const SchemaName: string = "SchemaName";
```

C++

```
string SchemaName = "SchemaName";
```

Description

This is SchemaName, a member of class ForeignKeyColumnsColumns.

15.1.3.1.5.2.11 ForeignKeyColumnsColumns.TableName Field

Pascal

```
const TableName: string = "TableName";
```

C++

```
string TableName = "TableName";
```

Description

This is TableName, a member of class ForeignKeyColumnsColumns.

15.1.3.1.6 ForeignKeysColumns Class

Column names for result table from the built-in stored procedure DB_ADMIN.GET_FOREIGN_KEYS and DataStoreConnection.GetSchema(MetaDataCollectionName.ForeignKeys).

Class Hierarchy

Borland.Data.MetaData.ForeignKeysColumns

Pascal

```
ForeignKeysColumns = class;
```

C++

```
class ForeignKeysColumns;
```

File

ForeignKeysColumns.cs

Description

Column names for result table from the built-in stored procedure DB_ADMIN.GET_FOREIGN_KEYS and DataStoreConnection.GetSchema(MetaDataCollectionName.ForeignKeys).

The columns of the result table are:

Constant	Description
CatalogName (see page 2154)	Used to access the CatalogName (see page 2154) column in the ForeignKeys metadata collection. The data in this column specifies the catalog name.
SchemaName (see page 2154)	Used to access the SchemaName (see page 2154) column in the ForeignKeys metadata collection. The data in this column specifies the schema name.
TableName (see page 2154)	Used to access the TableName (see page 2154) column in the ForeignKeys metadata collection. The data in this column specifies the table name.
ForeignKeyName (see page 2154)	Used to access the ForeignKeyName (see page 2154) column in the ForeignKeys metadata collection. The data in this column specifies the foreign key name.

15.1.3.1.6.1 ForeignKeysColumns Members

The following tables list the members exposed by ForeignKeysColumns.

Public Fields

	Name	Description
	CatalogName (see page 2154)	This is CatalogName, a member of class ForeignKeysColumns.
	ForeignKeyName (see page 2154)	This is ForeignKeyName, a member of class ForeignKeysColumns.
	SchemaName (see page 2154)	This is SchemaName, a member of class ForeignKeysColumns.
	TableName (see page 2154)	This is TableName, a member of class ForeignKeysColumns.

15.1.3.1.6.2 ForeignKeysColumns Fields

The fields of the ForeignKeysColumns class are listed here.

Public Fields

	Name	Description
	CatalogName (see page 2154)	This is CatalogName, a member of class ForeignKeysColumns.
	ForeignKeyName (see page 2154)	This is ForeignKeyName, a member of class ForeignKeysColumns.
	SchemaName (see page 2154)	This is SchemaName, a member of class ForeignKeysColumns.
	TableName (see page 2154)	This is TableName, a member of class ForeignKeysColumns.

15.1.3.1.6.2.1 ForeignKeysColumns.CatalogName Field

Pascal

```
const CatalogName: string = "CatalogName";
```

C++

```
string CatalogName = "CatalogName";
```

Description

This is CatalogName, a member of class ForeignKeysColumns.

15.1.3.1.6.2.2 ForeignKeysColumns.ForeignKeyName Field

Pascal

```
const ForeignKeyName: string = "ForeignKeyName";
```

C++

```
string ForeignKeyName = "ForeignKeyName";
```

Description

This is ForeignKeyName, a member of class ForeignKeysColumns.

15.1.3.1.6.2.3 ForeignKeysColumns.SchemaName Field

Pascal

```
const SchemaName: string = "SchemaName";
```

C++

```
string SchemaName = "SchemaName";
```

Description

This is SchemaName, a member of class ForeignKeysColumns.

15.1.3.1.6.2.4 ForeignKeysColumns.TableName Field

Pascal

```
const TableName: string = "TableName";
```

C++

```
string TableName = "TableName";
```

Description

This is TableName, a member of class ForeignKeysColumns.

15.1.3.1.7 IndexColumnsColumns Class

Column names for result table from the built-in stored procedure DB_ADMIN.GET_INDEX_COLUMNS and

DataStoreConnection.GetSchema(MetaDataCollectionName.IndexColumns).

Class Hierarchy

Borland.Data.MetaData.IndexColumnsColumns

Pascal

```
IndexColumnsColumns = class;
```

C++

```
class IndexColumnsColumns;
```

File

IndexColumnsColumns.cs

Description

Column names for result table from the built-in stored procedure DB_ADMIN.GET_INDEX_COLUMNS and DataStoreConnection.GetSchema(MetaDataCollectionName.IndexColumns).

The columns of the result table are:

Constant	Description
CatalogName (see page 2156)	Used to access the CatalogName (see page 2156) column in the IndexColumns metadata collection. The data in this column specifies the catalog name.
SchemaName (see page 2157)	Used to access the SchemaName (see page 2157) column in the IndexColumns metadata collection. The data in this column specifies the schema name.
TableName (see page 2157)	Used to access the TableName (see page 2157) column in the IndexColumns metadata collection. The data in this column specifies the table name.
IndexName (see page 2156)	Used to access the IndexName (see page 2156) column in the IndexColumns metadata collection. data in this column specifies the index name.
ColumnName (see page 2156)	Used to access the ColumnName (see page 2156) column in the IndexColumns metadata collection. The data in this column specifies a column in the index.
Ordinal (see page 2157)	Used to access the Ordinal (see page 2157) column in the IndexColumns metadata collection. The data in this column specifies the position of the column in the index starting with 1.
IsAscending (see page 2157)	Used to access the IsAscending (see page 2157) column in the IndexColumns metadata collection. The data in this column specifies that the index column is a descending index column.

See Also

Blackfish SQL Developer's Guide.

15.1.3.1.7.1 IndexColumnsColumns Members

The following tables list the members exposed by IndexColumnsColumns.

Public Fields

	Name	Description
	CatalogName (see page 2156)	This is CatalogName, a member of class IndexColumnsColumns.

	ColumnName (see page 2156)	This is ColumnName, a member of class IndexColumnsColumns.
	IndexName (see page 2156)	This is IndexName, a member of class IndexColumnsColumns.
	IsAscending (see page 2157)	This is IsAscending, a member of class IndexColumnsColumns.
	Ordinal (see page 2157)	This is Ordinal, a member of class IndexColumnsColumns.
	SchemaName (see page 2157)	This is SchemaName, a member of class IndexColumnsColumns.
	TableName (see page 2157)	This is TableName, a member of class IndexColumnsColumns.

15.1.3.1.7.2 IndexColumnsColumns Fields

The fields of the IndexColumnsColumns class are listed here.

Public Fields

	Name	Description
	CatalogName (see page 2156)	This is CatalogName, a member of class IndexColumnsColumns.
	ColumnName (see page 2156)	This is ColumnName, a member of class IndexColumnsColumns.
	IndexName (see page 2156)	This is IndexName, a member of class IndexColumnsColumns.
	IsAscending (see page 2157)	This is IsAscending, a member of class IndexColumnsColumns.
	Ordinal (see page 2157)	This is Ordinal, a member of class IndexColumnsColumns.
	SchemaName (see page 2157)	This is SchemaName, a member of class IndexColumnsColumns.
	TableName (see page 2157)	This is TableName, a member of class IndexColumnsColumns.

15.1.3.1.7.2.1 IndexColumnsColumns.CatalogName Field

Pascal

```
const CatalogName: string = "CatalogName";
```

C++

```
string CatalogName = "CatalogName";
```

Description

This is CatalogName, a member of class IndexColumnsColumns.

15.1.3.1.7.2.2 IndexColumnsColumns.ColumnName Field

Pascal

```
const ColumnName: string = "ColumnName";
```

C++

```
string ColumnName = "ColumnName";
```

Description

This is ColumnName, a member of class IndexColumnsColumns.

15.1.3.1.7.2.3 IndexColumnsColumns.IndexName Field

Pascal

```
const IndexName: string = "IndexName";
```

C++

```
string IndexName = "IndexName";
```

Description

This is IndexName, a member of class IndexColumnsColumns.

15.1.3.1.7.2.4 IndexColumnsColumns.IsAscending Field

Pascal

```
const IsAscending: string = "IsAscending";
```

C++

```
string IsAscending = "IsAscending";
```

Description

This is IsAscending, a member of class IndexColumnsColumns.

15.1.3.1.7.2.5 IndexColumnsColumns.Ordinal Field

Pascal

```
const Ordinal: string = "Ordinal";
```

C++

```
string Ordinal = "Ordinal";
```

Description

This is Ordinal, a member of class IndexColumnsColumns.

15.1.3.1.7.2.6 IndexColumnsColumns.SchemaName Field

Pascal

```
const SchemaName: string = "SchemaName";
```

C++

```
string SchemaName = "SchemaName";
```

Description

This is SchemaName, a member of class IndexColumnsColumns.

15.1.3.1.7.2.7 IndexColumnsColumns.TableName Field

Pascal

```
const TableName: string = "TableName";
```

C++

```
string TableName = "TableName";
```

Description

This is TableName, a member of class IndexColumnsColumns.

15.1.3.1.8 IndexesColumns Class

Provides the column names for the result table from the built-in stored procedure DB_ADMIN.GET_INDEXES and DataStoreConnection.GetSchema(MetaDataCollectionName.IndexColumns).

Class Hierarchy

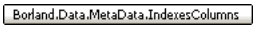

Pascal

```
IndexesColumns = class;
```

C++

```
class IndexesColumns;
```

File

IndexesColumns.cs

Description

IndexesColumns provides the column names for the result table from the built-in stored procedure DB_ADMIN.GET_INDEXES and DataStoreConnection.GetSchema(MetaDataCollectionName.IndexColumns). Each row in the result table of DB_ADMIN.GET_INDEXES describes a column in an index in a database.

The columns of the result table are:

Constant	Description
CatalogName (see page 2159)	Used to access the CatalogName (see page 2159) column in the Indexes metadata collection. The data in this column specifies the catalog name.
SchemaName (see page 2160)	Used to access the SchemaName (see page 2160) column in the Indexes metadata collection. The data in this column specifies the schema name.
TableName (see page 2160)	Used to access the TableName (see page 2160) column in the Indexes metadata collection. The data in this column specifies the table name.
IndexName (see page 2159)	Used to access the IndexName (see page 2159) column in the Indexes metadata collection. The data in this column specifies the index name.
ConstraintName (see page 2159)	Used to access the ConstraintName (see page 2159) column in the Indexes metadata collection. The data in this column specifies the constraint name, may be NULL.
IsPrimary (see page 2160)	Used to access the IsPrimary (see page 2160) column in the Indexes metadata collection. The data in this column specifies that the index is the primary key of the table.
IsUnique (see page 2160)	Used to access the IsUnique (see page 2160) column in the Indexes metadata collection. The data in this column specifies that the index is an unique index of the table.
IsAscending (see page 2159)	Used to access the IsAscending (see page 2159) column in the Indexes metadata collection. The data in this column specifies that the index is a descending index (Interbase only).

See Also

Blackfish SQL Developer's Guide.

15.1.3.1.8.1 IndexesColumns Members

The following tables list the members exposed by IndexesColumns.

Public Fields

	Name	Description
	CatalogName (see page 2159)	This is CatalogName, a member of class IndexesColumns.
	ConstraintName (see page 2159)	This is ConstraintName, a member of class IndexesColumns.
	IndexName (see page 2159)	This is IndexName, a member of class IndexesColumns.
	IsAscending (see page 2159)	This is IsAscending, a member of class IndexesColumns.
	IsPrimary (see page 2160)	This is IsPrimary, a member of class IndexesColumns.
	IsUnique (see page 2160)	This is IsUnique, a member of class IndexesColumns.
	SchemaName (see page 2160)	This is SchemaName, a member of class IndexesColumns.

	TableName (see page 2160)	This is TableName, a member of class IndexesColumns.

15.1.3.1.8.2 IndexesColumns Fields

The fields of the IndexesColumns class are listed here.

Public Fields

	Name	Description
	CatalogName (see page 2159)	This is CatalogName, a member of class IndexesColumns.
	ConstraintName (see page 2159)	This is ConstraintName, a member of class IndexesColumns.
	IndexName (see page 2159)	This is IndexName, a member of class IndexesColumns.
	IsAscending (see page 2159)	This is IsAscending, a member of class IndexesColumns.
	IsPrimary (see page 2160)	This is IsPrimary, a member of class IndexesColumns.
	IsUnique (see page 2160)	This is IsUnique, a member of class IndexesColumns.
	SchemaName (see page 2160)	This is SchemaName, a member of class IndexesColumns.
	TableName (see page 2160)	This is TableName, a member of class IndexesColumns.

15.1.3.1.8.2.1 IndexesColumns.CatalogName Field

Pascal

```
const CatalogName: string = "CatalogName";
```

C++

```
string CatalogName = "CatalogName";
```

Description

This is CatalogName, a member of class IndexesColumns.

15.1.3.1.8.2.2 IndexesColumns.ConstraintName Field

Pascal

```
const ConstraintName: string = "ConstraintName";
```

C++

```
string ConstraintName = "ConstraintName";
```

Description

This is ConstraintName, a member of class IndexesColumns.

15.1.3.1.8.2.3 IndexesColumns.IndexName Field

Pascal

```
const IndexName: string = "IndexName";
```

C++

```
string IndexName = "IndexName";
```

Description

This is IndexName, a member of class IndexesColumns.

15.1.3.1.8.2.4 IndexesColumns.IsAscending Field

Pascal

```
const IsAscending: string = "IsAscending";
```

C++

```
string IsAscending = "IsAscending";
```

Description

This is IsAscending, a member of class IndexesColumns.

15.1.3.1.8.2.5 IndexesColumns.IsPrimary Field

Pascal

```
const IsPrimary: string = "IsPrimary";
```

C++

```
string IsPrimary = "IsPrimary";
```

Description

This is IsPrimary, a member of class IndexesColumns.

15.1.3.1.8.2.6 IndexesColumns.IsUnique Field

Pascal

```
const IsUnique: string = "IsUnique";
```

C++

```
string IsUnique = "IsUnique";
```

Description

This is IsUnique, a member of class IndexesColumns.

15.1.3.1.8.2.7 IndexesColumns.SchemaName Field

Pascal

```
const SchemaName: string = "SchemaName";
```

C++

```
string SchemaName = "SchemaName";
```

Description

This is SchemaName, a member of class IndexesColumns.

15.1.3.1.8.2.8 IndexesColumns.TableName Field

Pascal

```
const TableName: string = "TableName";
```

C++

```
string TableName = "TableName";
```

Description

This is TableName, a member of class IndexesColumns.

15.1.3.1.9 MetaDataCollectionName Class

Collection names that can be passed to DataStoreConnection.GetSchema to receive metadata collections.

Class Hierarchy

Borland.Data.MetaData.MetaDataCollectionName

Pascal

```
MetaDataCollectionName = class;
```

C++

```
class MetaDataCollectionName;
```

File

MetaDataCollectionName.cs

Description

Collection names that can be passed to DataStoreConnection.GetSchema to receive metadata collections.

The collection names are:

Catalogs (see page 2163)
ColumnConstraints (see page 2163)
Columns (see page 2163)
DataTypes (see page 2164)
ForeignKeyColumns (see page 2165)
ForeignKeys (see page 2166)
IndexColumns (see page 2167)
Indexes (see page 2167)
PackageProcedureParameters (see page 2168)
PackageProcedures (see page 2168)
Packages (see page 2169)
PackageSources (see page 2169)
ProcedureParameters (see page 2170)
Procedures (see page 2171)
ProcedureSources (see page 2171)
ReservedWords (see page 2172)
Roles (see page 2172)
Schemas (see page 2172)
Synonyms (see page 2173)
Tables (see page 2173)
Users (see page 2174)
Views (see page 2174)

15.1.3.1.9.1 MetaDataCollectionName Members

The following tables list the members exposed by MetaDataCollectionName.

Public Fields

	Name	Description
	Catalogs (see page 2163)	This metadata collection represents catalogs in a database.
	ColumnConstraints (see page 2163)	This metadata collection is used internally.
	Columns (see page 2163)	This metadata collection represents table columns in a database.
	DataTypes (see page 2164)	This metadata collection represents data types in a database.
	ForeignKeyColumns (see page 2165)	This metadata collection represents foreign key columns in a database.
	ForeignKeys (see page 2166)	This metadata collection represents foreign keys in a database.
	IndexColumns (see page 2167)	This metadata collection represents index columns in a database.
	Indexes (see page 2167)	This metadata collection represents table indexes in a database.
	PackageProcedureParameters (see page 2168)	This metadata collection represents package procedure parameters in a database.
	PackageProcedures (see page 2168)	This metadata collection represents package procedures in a database.
	Packages (see page 2169)	This metadata collection represents packages in a database.
	PackageSources (see page 2169)	This metadata collection represents package source code in a database.
	ProcedureParameters (see page 2170)	This metadata collection represents procedure parameters in a database.
	Procedures (see page 2171)	This metadata collection represents procedures in a database.
	ProcedureSources (see page 2171)	This metadata collection represents procedure source code in a database.
	ReservedWords (see page 2172)	This metadata collection represents reserved words in a database.
	Roles (see page 2172)	This metadata collection represents roles in a database.
	Schemas (see page 2172)	This metadata collection represents schemas in a database.
	Synonyms (see page 2173)	This metadata collection represents synonyms in a database.
	Tables (see page 2173)	This metadata collection represents tables in a database.
	Users (see page 2174)	This metadata collection represents users in a database.
	Views (see page 2174)	This metadata collection represents views in a database.

15.1.3.1.9.2 MetaDataCollectionName Fields

The fields of the MetaDataCollectionName class are listed here.

Public Fields

	Name	Description
	Catalogs (see page 2163)	This metadata collection represents catalogs in a database.
	ColumnConstraints (see page 2163)	This metadata collection is used internally.
	Columns (see page 2163)	This metadata collection represents table columns in a database.
	DataTypes (see page 2164)	This metadata collection represents data types in a database.
	ForeignKeyColumns (see page 2165)	This metadata collection represents foreign key columns in a database.
	ForeignKeys (see page 2166)	This metadata collection represents foreign keys in a database.
	IndexColumns (see page 2167)	This metadata collection represents index columns in a database.
	Indexes (see page 2167)	This metadata collection represents table indexes in a database.
	PackageProcedureParameters (see page 2168)	This metadata collection represents package procedure parameters in a database.
	PackageProcedures (see page 2168)	This metadata collection represents package procedures in a database.
	Packages (see page 2169)	This metadata collection represents packages in a database.
	PackageSources (see page 2169)	This metadata collection represents package source code in a database.
	ProcedureParameters (see page 2170)	This metadata collection represents procedure parameters in a database.
	Procedures (see page 2171)	This metadata collection represents procedures in a database.
	ProcedureSources (see page 2171)	This metadata collection represents procedure source code in a database.
	ReservedWords (see page 2172)	This metadata collection represents reserved words in a database.
	Roles (see page 2172)	This metadata collection represents roles in a database.
	Schemas (see page 2172)	This metadata collection represents schemas in a database.
	Synonyms (see page 2173)	This metadata collection represents synonyms in a database.
	Tables (see page 2173)	This metadata collection represents tables in a database.
	Users (see page 2174)	This metadata collection represents users in a database.

	Views (see page 2174)	This metadata collection represents views in a database.

15.1.3.1.9.2.1 MetaDataCollectionName.Catalogs Field

This metadata collection represents catalogs in a database.

Pascal

```
const Catalogs: string = "Catalogs";
```

C++

```
string Catalogs = "Catalogs";
```

Description

This metadata collection represents catalogs in a database. The columns are:

Column Name	Data (see page 1762) Type	Description
CatalogName	String	The catalog name.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.2 MetaDataCollectionName.ColumnConstraints Field

This metadata collection is used internally.

Pascal

```
const ColumnConstraints: string = "ColumnConstraints";
```

C++

```
string ColumnConstraints = "ColumnConstraints";
```

Description

This metadata collection represents catalogs in a database is used internally.

The columns are:

Column Name	Data (see page 1762) Type	Description
CatalogName	String	The catalog name.
SchemaName	String	The schema name.
TableName	String	The table name.
ConstraintName	String	The constraint name.
ColumnName	String	The column name.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.3 MetaDataCollectionName.Columns Field

This metadata collection represents table columns in a database.

Pascal

```
const Columns: string = "Columns";
```

C++

```
string Columns = "Columns";
```

Description

This metadata collection represents table columns in a database. The columns are:

Column Name	Data (see page 1762) Type	Description
CatalogName	String	The catalog name.
SchemaName	String	The schema name.
TableName	String	The table name.
ColumnName	String	The column name.
TypeName	String	The database specific data type name.
Precision	Int32	The precision of this type. For a string data type, this is the maximum number of characters. For a decimal types, this is the maximum number digits including digits from the scale.
Scale	Int32	The scale of a decimal type.
Ordinal	Int32	The position in the rows of the table starting with 1.
DefaultValue	String	The default value.
IsNullable	Boolean	Can accept NULL values.
IsAutoIncrement	Boolean	Is an autoincrement value.
MaxInline	Int32	The maximal number of bytes to write into row. The rest of the data is handled as a blob (Blackfish SQL only).
DbxDataType	Int32	The DBX data type.
IsFixedLength	Boolean	Holds fixed length data.
IsUnicode	Boolean	Can hold unicode characters.
IsLong	Boolean	Is a blob data type.
IsUnsigned	Boolean	Is an unsigned numeric type.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.4 MetaDataCollectionName.DataTypes Field

This metadata collection represents data types in a database.

Pascal

```
const DataTypes: string = "DataTypes";
```

C++

```
string DataTypes = "DataTypes";
```

Description

This metadata collection represents the data types supported by a database.

The columns are:

Column Name	Data (see page 1762) Type	Description
The database specific data type name.	The DBX data type.	The database specific data type name. The length of this type. For a string data type, this is the maximum number of characters. For a floating point number, this is the number of significant bits in the mantissa. For other numeric types, this is the maximum number of significant digits. For a date, time, and timestamp types, this is the length of the string representation.
CreateFormat	String	The create format as used in CREATE TABLE.
CreateParameters	String	The parameters for the create format.
DataType	String	The name of the language type representing this data type.
IsAutoIncrementable	Boolean	Can be used as for an autoincrement column.
IsBestMatch	Boolean	Is the best match for the language type.
IsCaseSensitive	Boolean	Is case sensitive.
IsFixedLength	Boolean	Is fixed length.
IsFixedPrecisionScale	Boolean	Is an exact numeric.
IsLong	Boolean	Is a blob type.
IsNullable	Boolean	Can be assigned a value of NULL.
IsSearchable	Boolean	Can be searched in the SQL database.
IsSearchableWithLike	Boolean	Can be searched with the SQL LIKE operator.
IsUnsigned	Boolean	Is an unsigned numeric.
MaximumScale	Int16	The maximum possible scale value.
MinimumScale	Int16	The minimum possible scale value.
IsConcurrencyType	Boolean	Has special update semantics .
MaximumVersion	String	The last database product version where this type was available.
MinimumVersion	String	The first database product version where this type was available.
IsLiteralSupported	Boolean	Can be represented as a SQL literal.
LiteralPrefix	String	The prefix of a SQL literal.
LiteralSuffix	String	The suffix of a SQL literal.
IsUnicode	Boolean	Can hold unicode character data.
ProviderDb	Int32	The same as the DbxDataType column. Reserved for .NET applications.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.5 MetaDataCollectionName.ForeignKeyColumns Field

This metadata collection represents foreign key columns in a database.

Pascal

```
const ForeignKeyColumns: string = "ForeignKeyColumns";
```

C++

```
string ForeignKeyColumns = "ForeignKeyColumns";
```

Description

This metadata collection represents foreign key columns in a database.

The columns are:

Column Name	Data (see page 1762) Type	Description
CatalogName	String	The catalog name.
SchemaName	String	The schema name of the table with the foreign key.
TableName	String	The table name of the table with the foreign key.
ForeignKeyName	String	The foreign key name.
ColumnName	String	A column which refers to a column of in another table.
PrimaryCatalogName	String	The catalog name of the table being referred to.
PrimarySchemaName	String	The schema name of the table being referred to.
PrimaryTableName	String	The name of the table being referred to.
PrimaryKeyName	String	The name of the key being referred to. This is usually the primary key.
PrimaryColumnName	String	The column which is being referred to.
Ordinal	Int32	The position of the column reference in the foreign key.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.6 MetaDataCollectionName.ForeignKeys Field

This metadata collection represents foreign keys in a database.

Pascal

```
const ForeignKeys: string = "ForeignKeys";
```

C++

```
string ForeignKeys = "ForeignKeys";
```

Description

This metadata collection represents foreign keys in a database.

The columns are:

Column Name	Data (see page 1762) Type	Description
CatalogName	String	The catalog name.
SchemaName	String	The schema name.
TableName	String	The table name.
ForeignKeyName	String	The foreign key name.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.7 MetaDataCollectionName.IndexColumns Field

This metadata collection represents index columns in a database.

Pascal

```
const IndexColumns: string = "IndexColumns";
```

C++

```
string IndexColumns = "IndexColumns";
```

Description

This metadata collection represents index columns in a database.

The columns are:

Column Name	Data (see page 1762) Type	Description
CatalogName	String	The catalog name.
SchemaName	String	The schema name.
TableName	String	The table name.
IndexName	String	The index name.
ColumnName	String	A column in the index.
Ordinal	Int32	The position of the column in the index starting with 1.
IsAscending	Boolean	A descending index column.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.8 MetaDataCollectionName.Indexes Field

This metadata collection represents table indexes in a database.

Pascal

```
const Indexes: string = "Indexes";
```

C++

```
string Indexes = "Indexes";
```

Description

This metadata collection represents table indexes in a database.

The columns are:

Column Name	Data (see page 1762) Type	Description
CatalogName	String	The catalog name.
SchemaName	String	The schema name.
TableName	String	The table name.
IndexName	String	The index name.
ConstraintName	String	The constraint name, may be NULL.
IsPrimary	Boolean	The primary index of the table.

IsUnique	Boolean	A unique index of the table.
IsAscending	Boolean	A descending index (Interbase only).

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.9 MetaDataCollectionName.PackageProcedureParameters Field

This metadata collection represents package procedure parameters in a database.

Pascal

```
const PackageProcedureParameters: string = "PackageProcedureParameters";
```

C++

```
string PackageProcedureParameters = "PackageProcedureParameters";
```

Description

This metadata collection represents package procedure parameters in a database.

The columns are:

Column Name	Data (see page 1762) Type	Description
CatalogName	String	The catalog name.
SchemaName	String	The schema name.
PackageName	String	The package name.
ProcedureName	String	The procedure name from the package.
ParameterName	String	The parameter mode: IN, OUT, INOUT, or RESULT.
TypeName	String	The database specific data type name.
Precision	Int32	The precision of this type. For a string data type, this is the maximum number of characters. For a decimal types, this is the maximum number digits including digits from the scale.
Scale	Int32	The scale of a decimal type.
Ordinal	Int32	The position in the rows of the table starting with 1.
IsNullable	Boolean	Can accept NULL values.
DbxDataType	Int32	The DBX data type.
IsFixedLength	Boolean	Holds fixed length data.
IsUnicode	Boolean	Can hold UNICODE characters.
IsLong	Boolean	Is blob data type.
IsUnsigned	Boolean	Is unsigned numeric type.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.10 MetaDataCollectionName.PackageProcedures Field

This metadata collection represents package procedures in a database.

Pascal

```
const PackageProcedures: string = "PackageProcedures";
```

C++

```
string PackageProcedures = "PackageProcedures";
```

Description

This metadata collection represents package procedures in a database. The columns are:

Column Name	Data (see page 1762) Type	Description
CatalogName	String	The catalog name.
SchemaName	String	The schema name.
PackageName	String	The package name.
ProcedureName	String	The procedure name from the package.
ProcedureType	String	The procedure type: FUNCTION, PROCEDURE.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.11 MetaDataCollectionName.Packages Field

This metadata collection represents packages in a database.

Pascal

```
const Packages: string = "Packages";
```

C++

```
string Packages = "Packages";
```

Description

This metadata collection represents packages in a database. The columns are:

Column Name	Data (see page 1762) Type	Description
CatalogName	String	The catalog name.
SchemaName	String	The schema name.
PackageName	String	The package name.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.12 MetaDataCollectionName.PackageSources Field

This metadata collection represents package source code in a database.

Pascal

```
const PackageSources: string = "PackageSources";
```

C++

```
string PackageSources = "PackageSources";
```

Description

This metadata collection represents package source code in a database.

The columns are:

Column Name	Data (see page 1762) Type	Description
CatalogName	String	The catalog name.
PackageName	String	The package name.
Definition	String	The package source code.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.13 MetaDataCollectionName.ProcedureParameters Field

This metadata collection represents procedure parameters in a database.

Pascal

```
const ProcedureParameters: string = "ProcedureParameters";
```

C++

```
string ProcedureParameters = "ProcedureParameters";
```

Description

This metadata collection represents procedure parameters in a database.

The columns are:

Column Name	Data (see page 1762) Type	Description
CatalogName	String	The catalog name.
SchemaName	String	The schema name.
ProcedureName	String	The procedure name.
ParameterName	String	The parameter name.
ParameterMode	String	The parameter mode: IN, OUT, INOUT, or RESULT.
TypeName	String	The database specific data type name.
Precision	Int32	The precision of this type. For a string data type, this is the maximum number of characters. For a decimal types, this is the maximum number digits including digits from the scale.
Scale	Int32	The scale of a decimal type.
Ordinal	Int32	The position in the rows of the table starting with 1.
IsNullable	Boolean	Can accept NULL values.
DbxDataType	Int32	The DBX data type.
IsFixedLength	Boolean	Holds fixed length data.
IsUnicode	Boolean	Can hold UNICODE characters.
IsLong	Boolean	Blob data type.

IsUnsigned	Boolean	Unsigned numeric type.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.14 MetaDataCollectionName.Procedures Field

This metadata collection represents procedures in a database.

Pascal

```
const Procedures: string = "Procedures";
```

C++

```
string Procedures = "Procedures";
```

Description

This metadata collection represents procedures in a database.

The columns are:

Column Name	Data (see page 1762) Type	Description
CatalogName	String	The catalog name.
SchemaName	String	The schema name.
ProcedureName	String	The procedure name.
ProcedureType	String	The procedure type: FUNCTION or PROCEDURE.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.15 MetaDataCollectionName.ProcedureSources Field

This metadata collection represents procedure source code in a database.

Pascal

```
const ProcedureSources: string = "ProcedureSources";
```

C++

```
string ProcedureSources = "ProcedureSources";
```

Description

This metadata collection represents procedure source code in a database. The columns are:

Column Name	Data (see page 1762) Type	Description
CatalogName	String	The catalog name.
SchemaName	String	The schema name.
ProcedureName	String	The procedure name.
ProcedureType	String	The procedure type: FUNCTION or PROCEDURE.
Definition	String	The source of the SQL procedure.
ExternalDefinition	String	A reference to an procedure external to the database.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.16 MetaDataCollectionName.ReservedWords Field

This metadata collection represents reserved words in a database.

Pascal

```
const ReservedWords: string = "ReservedWords";
```

C++

```
string ReservedWords = "ReservedWords";
```

Description

This metadata collection represents reserved words in a database. The columns are:

Column Name	Data (see page 1762) Type	Description
ReservedWord	String	An identifier reserved for the SQL syntax. These identifiers should not be used as object names such as tables and columns. Some databases allows their use if they are quoted.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.17 MetaDataCollectionName.Roles Field

This metadata collection represents roles in a database.

Pascal

```
const Roles: string = "Roles";
```

C++

```
string Roles = "Roles";
```

Description

This metadata collection represents roles in a database.

The columns are:

Column Name	Data (see page 1762) Type	Description
RoleName	String	A role name.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.18 MetaDataCollectionName.Schemas Field

This metadata collection represents schemas in a database.

Pascal

```
const Schemas: string = "Schemas";
```

C++

```
string Schemas = "Schemas";
```

Description

This metadata collection represents schemas in a database. The columns are:

Column Name	Data (see page 1762) Type	Description
CatalogName	String	The catalog name.
SchemaName	String	The schema name.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.19 MetaDataCollectionName.Synonyms Field

This metadata collection represents synonyms in a database.

Pascal

```
const Synonyms: string = "Synonyms";
```

C++

```
string Synonyms = "Synonyms";
```

Description

This metadata collection represents synonyms in a database.

The columns are:

Column Name	Data (see page 1762) Type	Description
CatalogName	String	The catalog name.
SchemaName	String	The schema name.
SynonymName	String	The catalog name of the table to which this synonym refers.
TableCatalogName	String	The catalog name of the table to which this synonym refers.
TableSchemaName	String .	The schema name of the table to which this synonym refers.
TableName	String	The name of the table to which this synonym refers.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.20 MetaDataCollectionName.Tables Field

This metadata collection represents tables in a database.

Pascal

```
const Tables: string = "Tables";
```

C++

```
string Tables = "Tables";
```

Description

This metadata collection represents tables in a database.

The columns are:

Column Name	Data (see page 1762) Type	Description
CatalogName	String	The catalog name.
SchemaName	String	The schema name.
TableName	String	The table name.
TableType	String	The table type: TABLE, VIEW, SYNONYM, or SYSTEM_TABLE.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.21 MetaDataCollectionName.Users Field

This metadata collection represents users in a database.

Pascal

```
const Users: string = "Users";
```

C++

```
string Users = "Users";
```

Description

This metadata collection represents users in a database. The columns are:

Column Name	Data (see page 1762) Type	Description
UserName	String	A user name.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.9.2.22 MetaDataCollectionName.Views Field

This metadata collection represents views in a database.

Pascal

```
const Views: string = "Views";
```

C++

```
string Views = "Views";
```

Description

This metadata collection represents views in a database.

The columns are:

Column Name	Data (see page 1762) Type	Description
CatalogName	String	The catalog name.

SchemaName	String	The schema name.
ViewName	String	The view name.
Definition	String	The SQL that was used to create the view.

See Also

MetaDataCollectionName (see page 2160)

15.1.3.1.10 ProcedureParametersColumns Class

Column names for result table from the built-in stored procedure DB_ADMIN.GET_PROCEDURE_COLUMNS and DataStoreConnection.GetSchema(MetaDataCollectionName.ProcedureParameters).

Class Hierarchy

Borland.Data.MetaData.ProcedureParametersColumns

Pascal

```
ProcedureParametersColumns = class;
```

C++

```
class ProcedureParametersColumns;
```

File

ProcedureParametersColumns.cs

Description

Column names for result table from the built-in stored procedure DB_ADMIN.GET_PROCEDURE_COLUMNS and DataStoreConnection.GetSchema(MetaDataCollectionName.ProcedureParameters).

The columns of the result table are:

Constant	Description
CatalogName (see page 2177)	Used to access the CatalogName (see page 2177) column in the ProcedureParameters metadata collection.The data in this column specifies the catalog name.
SchemaName (see page 2179)	Used to access the SchemaName (see page 2179) column in the ProcedureParameters metadata collection. The data in this column specifies the schema name.
ProcedureName (see page 2179)	Used to access the ProcedureName (see page 2179) column in the ProcedureParameters metadata collection. The data in this column specifies the procedure name.
ParameterName (see page 2179)	Used to access the ParameterName (see page 2179) column in the ProcedureParameters metadata collection. The data in this column specifies the parameter name.
ParameterMode (see page 2178)	Used to access the ParameterMode (see page 2178) column in the ProcedureParameters metadata collection. The data in this column specifies the parameter mode: IN, OUT, INOUT, RESULT.
TypeName (see page 2180)	Used to access the TypeName (see page 2180) column in the ProcedureParameters metadata collection. The data in this column specifies the database specific data type name.
Precision (see page 2179)	Used to access the Precision (see page 2179) column in the ProcedureParameters metadata collection. The data in this column specifies the precision of this type. For a string data type, this is the maximum number of characters. For a decimal types, this is the maximum number digits including digits from the scale.

Scale (see page 2179)	Used to access the Scale (see page 2179) column in the ProcedureParameters metadata collection. The data in this column specifies the scale of a decimal type.
Ordinal (see page 2178)	Used to access the Ordinal (see page 2178) column in the ProcedureParameters metadata collection. The data in this column specifies the parameter number starting with 1.
IsNullable (see page 2178)	Used to access the IsNullable (see page 2178) column in the ProcedureParameters metadata collection. The data in this column specifies that the procedure parameter can accept NULL values.
DbxDataType (see page 2177)	Used to access the DbxDataType (see page 2177) column in the ProcedureParameters metadata collection. The data in this column specifies the DBX data type.
IsFixedLength (see page 2177)	Used to access the IsFixedLength (see page 2177) column in the ProcedureParameters metadata collection. The data in this column specifies that the procedure parameter holds fixed length data.
IsUnicode (see page 2178)	Used to access the IsUnicode (see page 2178) column in the ProcedureParameters metadata collection. The data in this column specifies that the procedure parameter can hold unicode characters.
IsLong (see page 2177)	Used to access the IsLong (see page 2177) column in the ProcedureParameters metadata collection. The data in this column specifies that the procedure parameter is blob data type.
IsUnsigned (see page 2178)	Used to access the IsUnsigned (see page 2178) column in the ProcedureParameters metadata collection. The data in this column specifies that the procedure parameter is unsigned numeric type.

See Also

Blackfish SQL Developer's Guide.

15.1.3.1.10.1 ProcedureParametersColumns Members

The following tables list the members exposed by ProcedureParametersColumns.

Public Fields

	Name	Description
	CatalogName (see page 2177)	This is CatalogName, a member of class ProcedureParametersColumns.
	DbxDataType (see page 2177)	This is DbxDataType, a member of class ProcedureParametersColumns.
	IsFixedLength (see page 2177)	This is IsFixedLength, a member of class ProcedureParametersColumns.
	IsLong (see page 2177)	This is IsLong, a member of class ProcedureParametersColumns.
	IsNullable (see page 2178)	This is IsNullable, a member of class ProcedureParametersColumns.
	IsUnicode (see page 2178)	This is IsUnicode, a member of class ProcedureParametersColumns.
	IsUnsigned (see page 2178)	This is IsUnsigned, a member of class ProcedureParametersColumns.
	Ordinal (see page 2178)	This is Ordinal, a member of class ProcedureParametersColumns.
	ParameterMode (see page 2178)	This is ParameterMode, a member of class ProcedureParametersColumns.
	ParameterName (see page 2179)	This is ParameterName, a member of class ProcedureParametersColumns.
	Precision (see page 2179)	This is Precision, a member of class ProcedureParametersColumns.
	ProcedureName (see page 2179)	This is ProcedureName, a member of class ProcedureParametersColumns.
	Scale (see page 2179)	This is Scale, a member of class ProcedureParametersColumns.
	SchemaName (see page 2179)	This is SchemaName, a member of class ProcedureParametersColumns.
	TypeName (see page 2180)	This is TypeName, a member of class ProcedureParametersColumns.

15.1.3.1.10.2 ProcedureParametersColumns Fields

The fields of the ProcedureParametersColumns class are listed here.

Public Fields

	Name	Description
	CatalogName (see page 2177)	This is CatalogName, a member of class ProcedureParametersColumns.

	DbxDataType (see page 2177)	This is DbxDataType, a member of class ProcedureParametersColumns.
	IsFixedLength (see page 2177)	This is IsFixedLength, a member of class ProcedureParametersColumns.
	IsLong (see page 2177)	This is IsLong, a member of class ProcedureParametersColumns.
	IsNullable (see page 2178)	This is IsNullable, a member of class ProcedureParametersColumns.
	IsUnicode (see page 2178)	This is IsUnicode, a member of class ProcedureParametersColumns.
	IsUnsigned (see page 2178)	This is IsUnsigned, a member of class ProcedureParametersColumns.
	Ordinal (see page 2178)	This is Ordinal, a member of class ProcedureParametersColumns.
	ParameterMode (see page 2178)	This is ParameterMode, a member of class ProcedureParametersColumns.
	ParameterName (see page 2179)	This is ParameterName, a member of class ProcedureParametersColumns.
	Precision (see page 2179)	This is Precision, a member of class ProcedureParametersColumns.
	ProcedureName (see page 2179)	This is ProcedureName, a member of class ProcedureParametersColumns.
	Scale (see page 2179)	This is Scale, a member of class ProcedureParametersColumns.
	SchemaName (see page 2179)	This is SchemaName, a member of class ProcedureParametersColumns.
	TypeName (see page 2180)	This is TypeName, a member of class ProcedureParametersColumns.

15.1.3.1.10.2.1 ProcedureParametersColumns.CatalogName Field

Pascal

```
const CatalogName: string = "CatalogName";
```

C++

```
string CatalogName = "CatalogName";
```

Description

This is CatalogName, a member of class ProcedureParametersColumns.

15.1.3.1.10.2.2 ProcedureParametersColumns.DbxDataType Field

Pascal

```
const DbxDataType: string = "DbxDataType";
```

C++

```
string DbxDataType = "DbxDataType";
```

Description

This is DbxDataType, a member of class ProcedureParametersColumns.

15.1.3.1.10.2.3 ProcedureParametersColumns.IsFixedLength Field

Pascal

```
const IsFixedLength: string = "IsFixedLength";
```

C++

```
string IsFixedLength = "IsFixedLength";
```

Description

This is IsFixedLength, a member of class ProcedureParametersColumns.

15.1.3.1.10.2.4 ProcedureParametersColumns.IsLong Field

Pascal

```
const IsLong: string = "IsLong";
```

C++

```
string IsLong = "IsLong";
```

Description

This is IsLong, a member of class ProcedureParametersColumns.

15.1.3.1.10.2.5 ProcedureParametersColumns.IsNullable Field

Pascal

```
const IsNullable: string = "IsNullable";
```

C++

```
string IsNullable = "IsNullable";
```

Description

This is IsNullable, a member of class ProcedureParametersColumns.

15.1.3.1.10.2.6 ProcedureParametersColumns.IsUnicode Field

Pascal

```
const IsUnicode: string = "IsUnicode";
```

C++

```
string IsUnicode = "IsUnicode";
```

Description

This is IsUnicode, a member of class ProcedureParametersColumns.

15.1.3.1.10.2.7 ProcedureParametersColumns.IsUnsigned Field

Pascal

```
const IsUnsigned: string = "IsUnsigned";
```

C++

```
string IsUnsigned = "IsUnsigned";
```

Description

This is IsUnsigned, a member of class ProcedureParametersColumns.

15.1.3.1.10.2.8 ProcedureParametersColumns.Ordinal Field

Pascal

```
const Ordinal: string = "Ordinal";
```

C++

```
string Ordinal = "Ordinal";
```

Description

This is Ordinal, a member of class ProcedureParametersColumns.

15.1.3.1.10.2.9 ProcedureParametersColumns.ParameterMode Field

Pascal

```
const ParameterMode: string = "ParameterMode";
```

C++

```
string ParameterMode = "ParameterMode";
```

Description

This is ParameterMode, a member of class ProcedureParametersColumns.

15.1.3.1.10.2.10 ProcedureParametersColumns.ParameterName Field

Pascal

```
const ParameterName: string = "ParameterName";
```

C++

```
string ParameterName = "ParameterName";
```

Description

This is ParameterName, a member of class ProcedureParametersColumns.

15.1.3.1.10.2.11 ProcedureParametersColumns.Precision Field

Pascal

```
const Precision: string = "Precision";
```

C++

```
string Precision = "Precision";
```

Description

This is Precision, a member of class ProcedureParametersColumns.

15.1.3.1.10.2.12 ProcedureParametersColumns.ProcedureName Field

Pascal

```
const ProcedureName: string = "ProcedureName";
```

C++

```
string ProcedureName = "ProcedureName";
```

Description

This is ProcedureName, a member of class ProcedureParametersColumns.

15.1.3.1.10.2.13 ProcedureParametersColumns.Scale Field

Pascal

```
const Scale: string = "Scale";
```

C++

```
string Scale = "Scale";
```

Description

This is Scale, a member of class ProcedureParametersColumns.

15.1.3.1.10.2.14 ProcedureParametersColumns.SchemaName Field

Pascal

```
const SchemaName: string = "SchemaName";
```

C++

```
string SchemaName = "SchemaName";
```

Description

This is SchemaName, a member of class ProcedureParametersColumns.

15.1.3.1.10.2.15 ProcedureParametersColumns.TypeName Field

Pascal

```
const TypeName: string = "TypeName";
```

C++

```
string TypeName = "TypeName";
```

Description

This is TypeName, a member of class ProcedureParametersColumns.

15.1.3.1.11 ProceduresColumns Class

Provides the column names for the result table from the built-in stored procedure DB_ADMIN.GET_PROCEDURES and DataStoreConnection.GetSchema(MetaDataCollectionName.Procedures).

Class Hierarchy

Pascal

```
ProceduresColumns = class;
```

C++

```
class ProceduresColumns;
```

File

ProceduresColumns.cs

Description

Provides the column names for the result table from the built-in stored procedure DB_ADMIN.GET_PROCEDURES and DataStoreConnection.GetSchema(MetaDataCollectionName.Procedures).

The columns of the result table are:

Constant	Description
CatalogName (see page 2181)	Used to access the CatalogName (see page 2181) column in the Procedures metadata collection. The data in this column specifies the catalog name.
SchemaName (see page 2182)	Used to access the SchemaName (see page 2182) column in the Procedures metadata collection. The data in this column specifies the schema name.
ProcedureName (see page 2181)	Used to access the ProcedureName (see page 2181) column in the Procedures metadata collection. The data in this column specifies the procedure name.
ProcedureType (see page 2181)	Used to access the ProcedureType (see page 2181) column in the Procedures metadata collection. The data in this column specifies the procedure type: FUNCTION or PROCEDURE.

See Also

Blackfish SQL Developer's Guide.

15.1.3.1.11.1 ProceduresColumns Members

The following tables list the members exposed by ProceduresColumns.

Public Fields

	Name	Description
	CatalogName (see page 2181)	This is CatalogName, a member of class ProceduresColumns.
	ProcedureName (see page 2181)	This is ProcedureName, a member of class ProceduresColumns.
	ProcedureType (see page 2181)	This is ProcedureType, a member of class ProceduresColumns.
	SchemaName (see page 2182)	This is SchemaName, a member of class ProceduresColumns.

15.1.3.1.11.2 ProceduresColumns Fields

The fields of the ProceduresColumns class are listed here.

Public Fields

	Name	Description
	CatalogName (see page 2181)	This is CatalogName, a member of class ProceduresColumns.
	ProcedureName (see page 2181)	This is ProcedureName, a member of class ProceduresColumns.
	ProcedureType (see page 2181)	This is ProcedureType, a member of class ProceduresColumns.
	SchemaName (see page 2182)	This is SchemaName, a member of class ProceduresColumns.

15.1.3.1.11.2.1 ProceduresColumns.CatalogName Field

Pascal

```
const CatalogName: string = "CatalogName";
```

C++

```
string CatalogName = "CatalogName";
```

Description

This is CatalogName, a member of class ProceduresColumns.

15.1.3.1.11.2.2 ProceduresColumns.ProcedureName Field

Pascal

```
const ProcedureName: string = "ProcedureName";
```

C++

```
string ProcedureName = "ProcedureName";
```

Description

This is ProcedureName, a member of class ProceduresColumns.

15.1.3.1.11.2.3 ProceduresColumns.ProcedureType Field

Pascal

```
const ProcedureType: string = "ProcedureType";
```

C++

```
string ProcedureType = "ProcedureType";
```

Description

This is ProcedureType, a member of class ProceduresColumns.

15.1.3.1.11.2.4 ProceduresColumns.SchemaName Field

Pascal

```
const SchemaName: string = "SchemaName";
```

C++

```
string SchemaName = "SchemaName";
```

Description

This is SchemaName, a member of class ProceduresColumns.

15.1.3.1.12 ProcedureSourcesColumns Class

Column names for the result table from the built-in stored procedure DB_ADMIN.GET_PROCEDURES and DataStoreConnection.GetSchema(MetaDataCollectionName.ProcedureSources).

Class Hierarchy

Pascal

```
ProcedureSourcesColumns = class;
```

C++

```
class ProcedureSourcesColumns;
```

File

ProcedureSourcesColumns.cs

Description

Column names for the result table from the built-in stored procedure DB_ADMIN.GET_PROCEDURES and DataStoreConnection.GetSchema(MetaDataCollectionName.ProcedureSources).

The columns of the result table are:

Constant	Description
CatalogName (see page 2183)	Used to access the CatalogName (see page 2183) column in the ProcedureSources metadata collection. The data in this column specifies the catalog name.
SchemaName (see page 2184)	Used to access the SchemaName (see page 2184) column in the ProcedureSources metadata collection. The data in this column specifies the schema name.
ProcedureName (see page 2184)	Used to access the ProcedureName (see page 2184) column in the ProcedureSources metadata collection. The data in this column specifies the procedure name.
ProcedureType (see page 2184)	Used to access the ProcedureType (see page 2184) column in the ProcedureSources metadata collection. The data in this column specifies the procedure type: FUNCTION or PROCEDURE.
Definition (see page 2183)	Used to access the Definition (see page 2183) column in the ProcedureSources metadata collection. The data in this column specifies the source of the SQL procedure.

ExternalDefinition (see page 2184)	Used to access the ExternalDefinition (see page 2184) column in the ProcedureSources metadata collection. The data in this column specifies a reference to an procedure external to the database.

See Also

Blackfish SQL Developer's Guide.

15.1.3.1.12.1 ProcedureSourcesColumns Members

The following tables list the members exposed by ProcedureSourcesColumns.

Public Fields

	Name	Description
	CatalogName (see page 2183)	This is CatalogName, a member of class ProcedureSourcesColumns.
	Definition (see page 2183)	This is Definition, a member of class ProcedureSourcesColumns.
	ExternalDefinition (see page 2184)	This is ExternalDefinition, a member of class ProcedureSourcesColumns.
	ProcedureName (see page 2184)	This is ProcedureName, a member of class ProcedureSourcesColumns.
	ProcedureType (see page 2184)	This is ProcedureType, a member of class ProcedureSourcesColumns.
	SchemaName (see page 2184)	This is SchemaName, a member of class ProcedureSourcesColumns.

15.1.3.1.12.2 ProcedureSourcesColumns Fields

The fields of the ProcedureSourcesColumns class are listed here.

Public Fields

	Name	Description
	CatalogName (see page 2183)	This is CatalogName, a member of class ProcedureSourcesColumns.
	Definition (see page 2183)	This is Definition, a member of class ProcedureSourcesColumns.
	ExternalDefinition (see page 2184)	This is ExternalDefinition, a member of class ProcedureSourcesColumns.
	ProcedureName (see page 2184)	This is ProcedureName, a member of class ProcedureSourcesColumns.
	ProcedureType (see page 2184)	This is ProcedureType, a member of class ProcedureSourcesColumns.
	SchemaName (see page 2184)	This is SchemaName, a member of class ProcedureSourcesColumns.

15.1.3.1.12.2.1 ProcedureSourcesColumns.CatalogName Field

Pascal

```
const CatalogName: string = "CatalogName";
```

C++

```
string CatalogName = "CatalogName";
```

Description

This is CatalogName, a member of class ProcedureSourcesColumns.

15.1.3.1.12.2.2 ProcedureSourcesColumns.Definition Field

Pascal

```
const Definition: string = "Definition";
```

C++

```
string Definition = "Definition";
```

Description

This is Definition, a member of class ProcedureSourcesColumns.

15.1.3.1.12.2.3 ProcedureSourcesColumns.ExternalDefinition Field

Pascal

```
const ExternalDefinition: string = "ExternalDefinition";
```

C++

```
string ExternalDefinition = "ExternalDefinition";
```

Description

This is ExternalDefinition, a member of class ProcedureSourcesColumns.

15.1.3.1.12.2.4 ProcedureSourcesColumns.ProcedureName Field

Pascal

```
const ProcedureName: string = "ProcedureName";
```

C++

```
string ProcedureName = "ProcedureName";
```

Description

This is ProcedureName, a member of class ProcedureSourcesColumns.

15.1.3.1.12.2.5 ProcedureSourcesColumns.ProcedureType Field

Pascal

```
const ProcedureType: string = "ProcedureType";
```

C++

```
string ProcedureType = "ProcedureType";
```

Description

This is ProcedureType, a member of class ProcedureSourcesColumns.

15.1.3.1.12.2.6 ProcedureSourcesColumns.SchemaName Field

Pascal

```
const SchemaName: string = "SchemaName";
```

C++

```
string SchemaName = "SchemaName";
```

Description

This is SchemaName, a member of class ProcedureSourcesColumns.

15.1.3.1.13 ReservedWordsColumns Class

Column names for result table from the built-in stored procedure DB_ADMIN.GET_KEYWORDS and DataStoreConnection.GetSchema(MetaDataCollectionName.ReservedWords).

Class Hierarchy

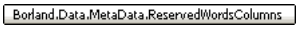

Pascal

```
ReservedWordsColumns = class;
```

C++

```
class ReservedWordsColumns;
```

File

ReservedWordsColumns.cs

Description

Column names for result table from the built-in stored procedure DB_ADMIN.GET_KEYWORDS and DataStoreConnection.GetSchema(MetaDataCollectionName.ReservedWords).

The columns of the result table are:

Constant	Description
ReservedWord (see page 2185)	Used to access the ReservedWord (see page 2185) column in the ReservedWords metadata collection. The data in this column specifies an identifier reserved for the SQL syntax. These identifiers should not be used as object names such as tables and columns. Some databases allow their use if they are quoted.

See Also

Blackfish SQL Developer's Guide.

15.1.3.1.13.1 ReservedWordsColumns Members

The following tables list the members exposed by ReservedWordsColumns.

Public Fields

	Name	Description
	ReservedWord (see page 2185)	This is ReservedWord, a member of class ReservedWordsColumns.

15.1.3.1.13.2 ReservedWordsColumns Fields

The fields of the ReservedWordsColumns class are listed here.

Public Fields

	Name	Description
	ReservedWord (see page 2185)	This is ReservedWord, a member of class ReservedWordsColumns.

15.1.3.1.13.2.1 ReservedWordsColumns.ReservedWord Field

Pascal

```
const ReservedWord: string = "ReservedWord";
```

C++

```
string ReservedWord = "ReservedWord";
```

Description

This is ReservedWord, a member of class ReservedWordsColumns.

15.1.3.1.14 RolesColumns Class

Column names for the result table from the built-in stored procedure DB_ADMIN.GET_ROLES and DataStoreConnection.GetSchema(MetaDataCollectionName.Roles).

Class Hierarchy

Borland.Data.MetaData.RolesColumns

Pascal

```
RolesColumns = class;
```

C++

```
class RolesColumns;
```

File

RolesColumns.cs

Description

Column names for the result table from the built-in stored procedure DB_ADMIN.GET_ROLES and DataStoreConnection.GetSchema(MetaDataCollectionName.Roles).

The columns of the result table are:

Constant	Description
RoleName (see page 2186)	Used to access the RoleName (see page 2186) column in the Roles metadata collection. The data in this column specifies a role name.

See Also

Blackfish SQL Developer's Guide.

15.1.3.1.14.1 RolesColumns Members

The following tables list the members exposed by RolesColumns.

Public Fields

	Name	Description
	RoleName (see page 2186)	This is RoleName, a member of class RolesColumns.

15.1.3.1.14.2 RolesColumns Fields

The fields of the RolesColumns class are listed here.

Public Fields

	Name	Description
	RoleName (see page 2186)	This is RoleName, a member of class RolesColumns.

15.1.3.1.14.2.1 RolesColumns.RoleName Field

Pascal

```
const RoleName: string = "RoleName";
```

C++

```
string RoleName = "RoleName";
```

Description

This is RoleName, a member of class RolesColumns.

15.1.3.1.15 SchemasColumns Class

Column names for the result table from the built-in stored procedure DB_ADMIN.GET_SCHEMAS and DataStoreConnection.GetSchema(MetaDataCollectionName.Schemas).

Class Hierarchy

Borland.Data.MetaData.SchemasColumns

Pascal

```
SchemasColumns = class;
```

C++

```
class SchemasColumns;
```

File

SchemasColumns.cs

Description

Column names for the result table from the built-in stored procedure DB_ADMIN.GET_SCHEMAS and DataStoreConnection.GetSchema(MetaDataCollectionName.Schemas).

The columns of the result table are:

Constant	Description
CatalogName (see page 2188)	Used to access the CatalogName (see page 2188) column in the Schemas metadata collection. The data in this column specifies the catalog name.
SchemaName (see page 2188)	Used to access the SchemaName (see page 2188) column in the Schemas metadata collection. The data in this column specifies the schema name.

See Also

Blackfish SQL Developer's Guide.

15.1.3.1.15.1 SchemasColumns Members

The following tables list the members exposed by SchemasColumns.

Public Fields

	Name	Description
	CatalogName (see page 2188)	This is CatalogName, a member of class SchemasColumns.
	SchemaName (see page 2188)	This is SchemaName, a member of class SchemasColumns.

15.1.3.1.15.2 SchemasColumns Fields

The fields of the SchemasColumns class are listed here.

Public Fields

	Name	Description
	CatalogName (see page 2188)	This is CatalogName, a member of class SchemasColumns.
	SchemaName (see page 2188)	This is SchemaName, a member of class SchemasColumns.

15.1.3.1.15.2.1 SchemasColumns.CatalogName Field

Pascal

```
const CatalogName: string = "CatalogName";
```

C++

```
string CatalogName = "CatalogName";
```

Description

This is CatalogName, a member of class SchemasColumns.

15.1.3.1.15.2.2 SchemasColumns.SchemaName Field

Pascal

```
const SchemaName: string = "SchemaName";
```

C++

```
string SchemaName = "SchemaName";
```

Description

This is SchemaName, a member of class SchemasColumns.

15.1.3.1.16 SynonymsColumns Class

Reserved for future use.

Class Hierarchy

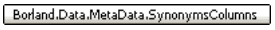

Pascal

```
SynonymsColumns = class;
```

C++

```
class SynonymsColumns;
```

File

SynonymsColumns.cs

Description

Reserved for future use.

15.1.3.1.16.1 SynonymsColumns Members

The following tables list the members exposed by SynonymsColumns.

Public Fields

	Name	Description
	CatalogName (see page 2189)	This is CatalogName, a member of class SynonymsColumns.
	SchemaName (see page 2189)	This is SchemaName, a member of class SynonymsColumns.
	SynonymName (see page 2189)	This is SynonymName, a member of class SynonymsColumns.
	TableCatalogName (see page 2189)	This is TableCatalogName, a member of class SynonymsColumns.
	TableName (see page 2190)	This is TableName, a member of class SynonymsColumns.
	TableSchemaName (see page 2190)	This is TableSchemaName, a member of class SynonymsColumns.

15.1.3.1.16.2 SynonymsColumns Fields

The fields of the SynonymsColumns class are listed here.

Public Fields

	Name	Description
	CatalogName (see page 2189)	This is CatalogName, a member of class SynonymsColumns.
	SchemaName (see page 2189)	This is SchemaName, a member of class SynonymsColumns.
	SynonymName (see page 2189)	This is SynonymName, a member of class SynonymsColumns.
	TableCatalogName (see page 2189)	This is TableCatalogName, a member of class SynonymsColumns.
	TableName (see page 2190)	This is TableName, a member of class SynonymsColumns.
	TableSchemaName (see page 2190)	This is TableSchemaName, a member of class SynonymsColumns.

15.1.3.1.16.2.1 SynonymsColumns.CatalogName Field

Pascal

```
const CatalogName: string = "CatalogName";
```

C++

```
string CatalogName = "CatalogName";
```

Description

This is CatalogName, a member of class SynonymsColumns.

15.1.3.1.16.2.2 SynonymsColumns.SchemaName Field

Pascal

```
const SchemaName: string = "SchemaName";
```

C++

```
string SchemaName = "SchemaName";
```

Description

This is SchemaName, a member of class SynonymsColumns.

15.1.3.1.16.2.3 SynonymsColumns.SynonymName Field

Pascal

```
const SynonymName: string = "SynonymName";
```

C++

```
string SynonymName = "SynonymName";
```

Description

This is SynonymName, a member of class SynonymsColumns.

15.1.3.1.16.2.4 SynonymsColumns.TableCatalogName Field

Pascal

```
const TableCatalogName: string = "TableCatalogName";
```

C++

```
string TableCatalogName = "TableCatalogName";
```

Description

This is TableCatalogName, a member of class SynonymsColumns.

15.1.3.1.16.2.5 SynonymsColumns.TableName Field

Pascal

```
const TableName: string = "TableName";
```

C++

```
string TableName = "TableName";
```

Description

This is TableName, a member of class SynonymsColumns.

15.1.3.1.16.2.6 SynonymsColumns.TableSchemaName Field

Pascal

```
const TableSchemaName: string = "TableSchemaName";
```

C++

```
string TableSchemaName = "TableSchemaName";
```

Description

This is TableSchemaName, a member of class SynonymsColumns.

15.1.3.1.17 TablesColumns Class

Column names for the result table from the built-in stored procedure DB_ADMIN.GET_TABLES and DataStoreConnection.GetSchema(MetaDataCollectionName.Tables).

Class Hierarchy

Pascal

```
TablesColumns = class;
```

C++

```
class TablesColumns;
```

File

TablesColumns.cs

Description

Column names for the result table from the built-in stored procedure DB_ADMIN.GET_TABLES and DataStoreConnection.GetSchema(MetaDataCollectionName.Tables).

The columns of the result table are:

Constant	Description
CatalogName (see page 2191)	Used to access the CatalogName (see page 2191) column in the Tables metadata collection. The data in this column specifies the catalog name.

TableName (see page 2192)	Used to access the TableName (see page 2192) column in the Tables metadata collection. The data in this column specifies the table name.
SchemaName (see page 2191)	Used to access the SchemaName (see page 2191) column in the Tables metadata collection. The data in this column specifies the schema name.
TableType (see page 2192)	Used to access the TableType (see page 2192) column in the Tables metadata collection. The data in this column specifies the table type: TABLE, VIEW, SYNONYM, or SYSTEM_TABLE.

See Also

Blackfish SQL Developer's Guide.

15.1.3.1.17.1 TablesColumns Members

The following tables list the members exposed by TablesColumns.

Public Fields

	Name	Description
	CatalogName (see page 2191)	This is CatalogName, a member of class TablesColumns.
	SchemaName (see page 2191)	This is SchemaName, a member of class TablesColumns.
	TableName (see page 2192)	This is TableName, a member of class TablesColumns.
	TableType (see page 2192)	This is TableType, a member of class TablesColumns.

15.1.3.1.17.2 TablesColumns Fields

The fields of the TablesColumns class are listed here.

Public Fields

	Name	Description
	CatalogName (see page 2191)	This is CatalogName, a member of class TablesColumns.
	SchemaName (see page 2191)	This is SchemaName, a member of class TablesColumns.
	TableName (see page 2192)	This is TableName, a member of class TablesColumns.
	TableType (see page 2192)	This is TableType, a member of class TablesColumns.

15.1.3.1.17.2.1 TablesColumns.CatalogName Field

Pascal

```
const CatalogName: string = "CatalogName";
```

C++

```
string CatalogName = "CatalogName";
```

Description

This is CatalogName, a member of class TablesColumns.

15.1.3.1.17.2.2 TablesColumns.SchemaName Field

Pascal

```
const SchemaName: string = "SchemaName";
```

C++

```
string SchemaName = "SchemaName";
```

Description

This is SchemaName, a member of class TablesColumns.

15.1.3.1.17.2.3 TablesColumns.TableName Field

Pascal

```
const TableName: string = "TableName";
```

C++

```
string TableName = "TableName";
```

Description

This is TableName, a member of class TablesColumns.

15.1.3.1.17.2.4 TablesColumns.TableType Field

Pascal

```
const TableType: string = "TableType";
```

C++

```
string TableType = "TableType";
```

Description

This is TableType, a member of class TablesColumns.

15.1.3.1.18 UsersColumns Class

Column names for the result table from the built-in stored procedure DB_ADMIN.GET_USERS and DataStoreConnection.GetSchema(MetaDataCollectionName.Users).

Class Hierarchy

Borland.Data.MetaData.UsersColumns

Pascal

```
UsersColumns = class;
```

C++

```
class UsersColumns;
```

File

UsersColumns.cs

Description

Column names for the result table from the built-in stored procedure DB_ADMIN.GET_USERS and DataStoreConnection.GetSchema(MetaDataCollectionName.Users).

The columns of the result table are:

Constant	Description
UserName (see page 2193)	Used to access the UserName (see page 2193) column in the Users metadata collection. The data in this column specifies a user name.

See Also

Blackfish SQL Developer's Guide.

15.1.3.1.18.1 UsersColumns Members

The following tables list the members exposed by UsersColumns.

Public Fields

	Name	Description
	UserName (see page 2193)	This is UserName, a member of class UsersColumns.

15.1.3.1.18.2 UsersColumns Fields

The fields of the UsersColumns class are listed here.

Public Fields

	Name	Description
	UserName (see page 2193)	This is UserName, a member of class UsersColumns.

15.1.3.1.18.2.1 UsersColumns.UserName Field

Pascal

```
const UserName: string = "UserName";
```

C++

```
string UserName = "UserName";
```

Description

This is UserName, a member of class UsersColumns.

15.1.3.1.19 ViewsColumns Class

Column names for the result table from the built-in stored procedure DB_ADMIN.GET_VIEWS DataStoreConnection.GetSchema(MetaDataCollectionName.Views).

Class Hierarchy

Borland.Data.MetaData.ViewsColumns

Pascal

```
ViewsColumns = class;
```

C++

```
class ViewsColumns;
```

File

ViewsColumns.cs

Description

Column names for the result table from the built-in stored procedure DB_ADMIN.GET_VIEWS DataStoreConnection.GetSchema(MetaDataCollectionName.Views).

The columns of the result table are:

Constant	Description
CatalogName (see page 2194)	Used to access the CatalogName (see page 2194) column in the Views metadata collection. The data in this column specifies the catalog name.
SchemaName (see page 2195)	Used to access the SchemaName (see page 2195) column in the Views metadata collection. The data in this column specifies the schema name.
ViewName (see page 2195)	Used to access the ViewName (see page 2195) column in the Views metadata collection. The data in this column specifies the view name.
Definition (see page 2195)	Used to access the Definition (see page 2195) column in the Views metadata collection. The data in this column specifies the SQL which was used to create the view.

See Also

Blackfish SQL Developer's Guide.

15.1.3.1.19.1 ViewsColumns Members

The following tables list the members exposed by ViewsColumns.

Public Fields

	Name	Description
	CatalogName (see page 2194)	This is CatalogName, a member of class ViewsColumns.
	Definition (see page 2195)	This is Definition, a member of class ViewsColumns.
	SchemaName (see page 2195)	This is SchemaName, a member of class ViewsColumns.
	ViewName (see page 2195)	This is ViewName, a member of class ViewsColumns.

15.1.3.1.19.2 ViewsColumns Fields

The fields of the ViewsColumns class are listed here.

Public Fields

	Name	Description
	CatalogName (see page 2194)	This is CatalogName, a member of class ViewsColumns.
	Definition (see page 2195)	This is Definition, a member of class ViewsColumns.
	SchemaName (see page 2195)	This is SchemaName, a member of class ViewsColumns.
	ViewName (see page 2195)	This is ViewName, a member of class ViewsColumns.

15.1.3.1.19.2.1 ViewsColumns.CatalogName Field

Pascal

```
const CatalogName: string = "CatalogName";
```

C++

```
string CatalogName = "CatalogName";
```

Description

This is CatalogName, a member of class ViewsColumns.

15.1.3.1.19.2.2 ViewsColumns.Definition Field

Pascal

```
const Definition: string = "Definition";
```

C++

```
string Definition = "Definition";
```

Description

This is Definition, a member of class ViewsColumns.

15.1.3.1.19.2.3 ViewsColumns.SchemaName Field

Pascal

```
const SchemaName: string = "SchemaName";
```

C++

```
string SchemaName = "SchemaName";
```

Description

This is SchemaName, a member of class ViewsColumns.

15.1.3.1.19.2.4 ViewsColumns.ViewName Field

Pascal

```
const ViewName: string = "ViewName";
```

C++

```
string ViewName = "ViewName";
```

Description

This is ViewName, a member of class ViewsColumns.

16 ButtonGroup Namespace

This is namespace ButtonGroup.

Topics

Name	Description
Classes (see page 2198)	The following table lists classes in this documentation.
Types (see page 2284)	The following table lists types in this documentation.

Classes

	Name	Description
	TButtonGroup (see page 2198)	TButtonGroup displays a group of buttons in a container.
	TGrpButtonItem (see page 2275)	TGrpButtonItem is an individual button of a TButtonGroup (see page 2198) control.
	TGrpButtonItems (see page 2278)	TGrpButtonItems maintains the collection of buttons that appear in a TButtonGroup (see page 2198) control.

Types

Name	Description
TGrpButtonDrawEvent (see page 2284)	TGrpButtonDrawEvent is the type of Button Group draw event handler.
TGrpButtonDrawIconEvent (see page 2285)	TGrpButtonDrawIconEvent is the type of Button Group OnDrawIcon event handler.
TGrpButtonEvent (see page 2285)	TGrpButtonEvent is a type of Button Group event handler.
TGrpButtonItemClass (see page 2286)	Metaclass of TGrpButtonItem (see page 2275).
TGrpButtonItemsClass (see page 2286)	Metaclass of TGrpButtonItems (see page 2278).
TGrpButtonOptions (see page 2286)	Options for button groups.
TGrpButtonReorderEvent (see page 2287)	TGrpButtonReorderEvent is a type of Button Group reorder event handler.

16.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TButtonGroup (see page 2198)	TButtonGroup displays a group of buttons in a container.
	TGrpButtonItem (see page 2275)	TGrpButtonItem is an individual button of a TButtonGroup (see page 2198) control.
	TGrpButtonItems (see page 2278)	TGrpButtonItems maintains the collection of buttons that appear in a TButtonGroup (see page 2198) control.

16.1.1 TButtonGroup Class

TButtonGroup displays a group of buttons in a container.

Class Hierarchy

Pascal

```
TButtonGroup = class(TCustomControl);
```

C++

```
class TButtonGroup : public TCustomControl;
```

File

ButtonGroup

Description

You can use a TButtonGroup to create a related set of buttons. For example in the RAD Studio IDE, a button group control is used to display **Tool Palette** categories. The button group appears when you click the **Categories** button in the IDE.

16.1.1.1 TButtonGroup Members

The following tables list the members exposed by TButtonGroup.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnResize (see page 5870)	Occurs immediately after the control is resized.

	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
V	CreateButton (see page 2211)	This is CreateButton, a member of class TButtonGroup.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 2211)	Creates underlying screen object.
V	CreateParams (see page 2211)	Initializes a window-creation parameter data structure.

	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
V	DoEndDrag (see page 2212)	OnEndDrag event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoHotButton (see page 2212)	This is DoHotButton, a member of class TButtonGroup.
V	DoItemClicked (see page 2212)	This is DoItemClicked, a member of class TButtonGroup.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
V	DoMouseWheelDown (see page 2213)	OnMouseWheelDown event dispatcher.
V	DoMouseWheelUp (see page 2213)	OnMouseWheelUp event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoReorderButton (see page 2214)	This is DoReorderButton, a member of class TButtonGroup.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
V	DoStartDrag (see page 2214)	OnStartDrag event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 2215)	OnDragOver event dispatcher.
V	DrawButton (see page 2215)	This is DrawButton, a member of class TButtonGroup.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.

	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetButtonClass (see page 2215)	This is GetButtonClass, a member of class TButtonGroup.
	GetButtonsClass (see page 2216)	This is GetButtonsClass, a member of class TButtonGroup.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 2216)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 2216)	OnMouseDown event dispatcher.
	MouseMove (see page 2217)	Respond to mouse moving over control area..
	MouseUp (see page 2217)	OnMouseUp event dispatcher.
	Notification (see page 2218)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 2218)	Renders the image of a custom control.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.

	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 2219)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.

	UpdateAllButtons (see page 2220)	This is UpdateAllButtons, a member of class TButtonGroup.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateButton (see page 2220)	This is UpdateButton, a member of class TButtonGroup.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2210)	Assigns values to buttons and categories.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2209)	Constructor for the object.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2209)	Destructor of the object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
V	DragDrop (see page 2214)	OnDragDrop event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetButtonRect (see page 2215)	Retrieves the bounding rectangle of a button, given the rectangle's index number.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	IndexOfButtonAt (see page 2216)	Retrieves the index of a button, given x and y coordinates.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveInsertionPoints (see page 2219)	Removes the insertion points added by SetInsertionPoints (see page 2220).
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollIntoView (see page 2219)	Scrolls buttons to make the the buttons visible in the group container.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetInsertionPoints (see page 2220)	Draws an insert line for inserting at InsertionIndex. You can choose to show or hide insertion points.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	TargetIndexAt (see page 2220)	Provides the target insertion index of a given coordinate (x,y). If the target is somewhere in or above the top half of a button, this method inserts an object above the button. If the target is somewhere in or below the bottom half of a button, the method inserts an object after the button.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	Canvas (see page 2229)	Specifies the TCanvas object that presents a drawing surface for the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.

R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	DragImageList (see page 2232)	DragImageList defines the list of images shown during drag operations.
	DragIndex (see page 2233)	DragIndex defines the index of the button during drag operations.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAfterDrawButton (see page 2248)	Occurs after a button is drawn.
	OnAlignInsertBefore (see page 2248)	Occurs when an object is aligned.
	OnAlignPosition (see page 2249)	Occurs when a control is aligned to a new position.
	OnBeforeDrawButton (see page 2249)	Occurs just before a button is drawn.
	OnButtonClicked (see page 2249)	Occurs when a button is clicked, if the OnClick (see page 2249) event is not present.

	OnClick (see page 2249)	Occurs when the user clicks the control.
	OnContextPopup (see page 2251)	Occurs when a user right-clicks on the dock site to display the popup menu.
	OnDockDrop (see page 2251)	Occurs when the control is dropped by a release of the mouse button.
	OnDockOver (see page 2251)	Occurs when the control is being moved over another control, but has not yet been docked.
	OnDragDrop (see page 2252)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2254)	Occurs when the user drags an object over a control.
	OnDrawButton (see page 2256)	Occurs when a button is redrawn.
	OnDrawIcon (see page 2256)	Occurs when an icon is redrawn.
	OnEndDock (see page 2257)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2257)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2258)	Occurs when the control gets focus.
	OnExit (see page 2258)	Occurs when the control loses focus, typically in tabbing between controls.
	OnHotButton (see page 2258)	Fired when the mouse pointer is over the button.
	OnKeyDown (see page 2259)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2261)	Occurs when a key is pressed.
	OnKeyUp (see page 2262)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2264)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2265)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2267)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2268)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2268)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2270)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 2273)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 2273)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 2273)	Occurs when the mouse wheel is rotated upward.
	OnReorderButton (see page 2274)	Fired when buttons are reordered in the button group.
	OnStartDock (see page 2274)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2274)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 2221)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 2225)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 2225)	Specifies which edges of the control are beveled.
	BevelInner (see page 2226)	Specifies the cut of the inner bevel.
	BevelKind (see page 2226)	Specifies the control's bevel style.
	BevelOuter (see page 2227)	Specifies the cut of the outer bevel.
	BevelWidth (see page 2227)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 2227)	BorderStyle specifies the appearance of the button group border.
	BorderWidth (see page 2228)	Specifies the width of the control's border.
	ButtonHeight (see page 2228)	ButtonHeight specifies the height of the buttons in the group.
	ButtonOptions (see page 2228)	Specifies options for button display and behavior.
	ButtonWidth (see page 2229)	ButtonWidth specifies the width of the buttons in the group.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DockSite (see page 2231)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 2232)	Indicates the image used to represent the mouse pointer when the control is being dragged.

	DragKind (see page 2233)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2233)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2235)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2235)	Controls the attributes of text written on or in the control.
	Height (see page 2237)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Images (see page 2238)	This is Images, a member of class TButtonGroup.
	ItemIndex (see page 2238)	Specifies the index of the currently selected item.
	Items (see page 2238)	Button items in this group.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	PopupMenu (see page 2238)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2243)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2244)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2245)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 2245)	Determines whether the component appears onscreen.
	Width (see page 2247)	Specifies the horizontal size of the control or form in pixels.

16.1.1.2 TButtonGroup.Create Constructor

Constructor for the object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TButtonGroup(TComponent * AOwner);
```

16.1.1.3 TButtonGroup.Destroy Destructor

Destructor of the object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TButtonGroup();
```

Description

Releases memory and performs garbage collection for the object.

16.1.1.4 TButtonGroup Methods

The methods of the TButtonGroup class are listed here.

Protected Methods

	Name	Description
	CreateButton (see page 2211)	This is CreateButton, a member of class TButtonGroup.
	CreateHandle (see page 2211)	Creates underlying screen object.
	CreateParams (see page 2211)	Initializes a window-creation parameter data structure.
	DoEndDrag (see page 2212)	OnEndDrag event dispatcher.
	DoHotButton (see page 2212)	This is DoHotButton, a member of class TButtonGroup.
	DoItemClicked (see page 2212)	This is DoItemClicked, a member of class TButtonGroup.
	DoMouseWheelDown (see page 2213)	OnMouseWheelDown event dispatcher.
	DoMouseWheelUp (see page 2213)	OnMouseWheelUp event dispatcher.
	DoReorderButton (see page 2214)	This is DoReorderButton, a member of class TButtonGroup.
	DoStartDrag (see page 2214)	OnStartDrag event dispatcher.
	DragOver (see page 2215)	OnDragOver event dispatcher.
	DrawButton (see page 2215)	This is DrawButton, a member of class TButtonGroup.
	GetButtonClass (see page 2215)	This is GetButtonClass, a member of class TButtonGroup.
	GetButtonsClass (see page 2216)	This is GetButtonsClass, a member of class TButtonGroup.
	GetDragImages (see page 2216)	Returns the control's image list containing the image to be displayed while dragging.
	MouseDown (see page 2216)	OnMouseDown event dispatcher.
	MouseMove (see page 2217)	Respond to mouse moving over control area..
	MouseUp (see page 2217)	OnMouseUp event dispatcher.
	Notification (see page 2218)	Responds to notifications that components are being created or destroyed.
	Paint (see page 2218)	Renders the image of a custom control.
	Resize (see page 2219)	Respond to control resize.
	UpdateAllButtons (see page 2220)	This is UpdateAllButtons, a member of class TButtonGroup.
	UpdateButton (see page 2220)	This is UpdateButton, a member of class TButtonGroup.

Public Methods

	Name	Description
	Assign (see page 2210)	Assigns values to buttons and categories.
	DragDrop (see page 2214)	OnDragDrop event dispatcher.
	GetButtonRect (see page 2215)	Retrieves the bounding rectangle of a button, given the rectangle's index number.
	IndexOfButtonAt (see page 2216)	Retrieves the index of a button, given x and y coordinates.
	RemoveInsertionPoints (see page 2219)	Removes the insertion points added by SetInsertionPoints (see page 2220).
	ScrollIntoView (see page 2219)	Scrolls buttons to make the the buttons visible in the group container.
	SetInsertionPoints (see page 2220)	Draws an insert line for inserting at InsertionIndex. You can choose to show or hide insertion points.
	TargetIndexAt (see page 2220)	Provides the target insertion index of a given coordinate (x,y). If the target is somewhere in or above the top half of a button, this method inserts an object above the button. If the target is somewhere in or below the bottom half of a button, the method inserts an object after the button.

16.1.1.4.1 TButtonGroup.Assign Method

Assigns values to buttons and categories.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

16.1.1.4.2 TButtonGroup.CreateButton Method

Pascal

```
function CreateButton: TGrpButtonItem; virtual;
```

C++

```
virtual __fastcall TGrpButtonItem CreateButton();
```

Description

This is CreateButton, a member of class TButtonGroup.

16.1.1.4.3 TButtonGroup.CreateHandle Method

Creates underlying screen object.

Pascal

```
procedure CreateHandle; override;
```

C++

```
virtual __fastcall CreateHandle();
```

Description

CreateHandle creates the Microsoft Windows screen object that underlies the control. if it does not already exist. The new window handle becomes the value of the Handle property.

CreateHandle creates or locates the parent control's window before it creates a window for the control itself.

Note: If the Parent property is nil (Delphi) or NULL (C++), CreateHandle raises an exception.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

Handle (see page 6207)

Parent (see page 5832)

16.1.1.4.4 TButtonGroup.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

16.1.1.4.5 TButtonGroup.DoEndDrag Method

OnEndDrag event dispatcher.

Pascal

```
procedure DoEndDrag(Target: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DoEndDrag(TObject * Target, int X, int Y);
```

Description

DoEndDrag calls any event handler attached to the OnEndDrag event. When creating a component that descends from TControl, override DoEndDrag to provide other responses in addition to the inherited event-handler call when the user drops an object or cancels a drag-and-drop operation.

See Also

DoStartDrag (see page 5753)

OnEndDrag (see page 5860)

16.1.1.4.6 TButtonGroup.DoHotButton Method

Pascal

```
procedure DoHotButton; dynamic;
```

C++

```
__fastcall DoHotButton();
```

Description

This is DoHotButton, a member of class TButtonGroup.

16.1.1.4.7 TButtonGroup.DoItemClicked Method

Pascal

```
procedure DoItemClicked(const Index: Integer); virtual;
```

C++

```
virtual __fastcall DoItemClicked(const int Index);
```

Description

This is DoItemClicked, a member of class TButtonGroup.

16.1.1.4.8 TButtonGroup.DoMouseWheelDown Method

OnMouseWheelDown event dispatcher.

Pascal

```
function DoMouseWheelDown(Shift: TShiftState; MousePos: TPoint): Boolean; override;
```

C++

```
virtual __fastcall Boolean DoMouseWheelDown(TShiftState Shift, TPoint MousePos);
```

Description

The DoMouseWheel method calls DoMouseWheelDown to generate an OnMouseWheelDown event.

The Shift parameter indicates the state of the Shift, Alt, and Ctrl keys.

The MousePos parameter indicates the position of the mouse pointer.

DoMouseWheelDown returns true if an event handler indicates that it handled the message, and false otherwise.

Override DoMouseWheelDown to perform class-specific tasks in addition to calling the event handler when the mouse wheel is rotated downward.

See Also

OnMouseWheelDown (see page 5869)

DoMouseWheel (see page 5751)

16.1.1.4.9 TButtonGroup.DoMouseWheelUp Method

OnMouseWheelUp event dispatcher.

Pascal

```
function DoMouseWheelUp(Shift: TShiftState; MousePos: TPoint): Boolean; override;
```

C++

```
virtual __fastcall Boolean DoMouseWheelUp(TShiftState Shift, TPoint MousePos);
```

Description

The DoMouseWheel method calls DoMouseWheelUp to generate an OnMouseWheelUp event.

The Shift parameter indicates the state of the Shift, Alt, and Ctrl keys.

The MousePos parameter indicates the position of the mouse pointer.

DoMouseWheelUp returns true if an event handler indicates that it handled the message, and false otherwise.

Override DoMouseWheelUp to perform class-specific tasks in addition to calling the event handler when the mouse wheel is rotated upward.

See Also

OnMouseWheelUp (see page 5870)

DoMouseWheel (see page 5751)

16.1.1.4.10 TButtonGroup.DoReorderButton Method

Pascal

```
procedure DoReorderButton(const OldIndex: Integer; const NewIndex: Integer);
```

C++

```
__fastcall DoReorderButton(const int OldIndex, const int NewIndex);
```

Description

This is DoReorderButton, a member of class TButtonGroup.

16.1.1.4.11 TButtonGroup.DoStartDrag Method

OnStartDrag event dispatcher.

Pascal

```
procedure DoStartDrag(var DragObject: TDragObject); override;
```

C++

```
virtual __fastcall DoStartDrag(TDragObject DragObject);
```

Description

DoStartDrag calls any event handler attached to the OnStartDrag event. When creating a component that descends from TControl, override DoStartDrag to provide other responses in addition to the inherited event-handler call when the user starts a drag-and-drop operation.

See Also

DoEndDrag (see page 5751)

OnStartDrag (see page 5871)

16.1.1.4.12 TButtonGroup.DragDrop Method

OnDragDrop event dispatcher.

Pascal

```
procedure DragDrop(Source: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DragDrop(TObject * Source, int X, int Y);
```

Description

When creating a component that descends from TControl, override DragDrop to add additional code that executes before the OnDragDrop event handler is called.

The Source parameter is the object that was dropped onto the control. The X and Y parameters are the mouse coordinates where the object was dropped.

See Also

DragCanceled (see page 5754)

OnDragDrop (see page 5855)

16.1.1.4.13 TButtonGroup.DragOver Method

OnDragOver event dispatcher.

Pascal

```
procedure DragOver(Source: TObject; X: Integer; Y: Integer; State: TDragState; var Accept:
Boolean); override;
```

C++

```
virtual __fastcall DragOver(TObject * Source, int X, int Y, TDragState State, Boolean Accept);
```

Description

Override DragOver to add additional code that executes before the OnDragOver event handler is called.

DragOver sets the Accept parameter to true to indicate that the user can drop the dragged object on the control. It sets Accept to false to indicate that the user cannot drop the dragged object on the control.

The Source parameter is the object being dragged. The State parameter indicates how the dragged object is moving in relation to the control. X and Y indicate the current position of the mouse.

See Also

DragDrop (see page 5754)

OnDragOver (see page 5857)

16.1.1.4.14 TButtonGroup.DrawButton Method

Pascal

```
procedure DrawButton(Index: Integer; Canvas: TCanvas; Rect: TRect; State: TButtonDrawState);
virtual;
```

C++

```
virtual __fastcall DrawButton(int Index, TCanvas Canvas, TRect Rect, TButtonDrawState State);
```

Description

This is DrawButton, a member of class TButtonGroup.

16.1.1.4.15 TButtonGroup.GetButtonClass Method

Pascal

```
function GetButtonClass: TGrpButtonItemClass; virtual;
```

C++

```
virtual __fastcall TGrpButtonItemClass GetButtonClass();
```

Description

This is GetButtonClass, a member of class TButtonGroup.

16.1.1.4.16 TButtonGroup.GetButtonRect Method

Retrieves the bounding rectangle of a button, given the rectangle's index number.

Pascal

```
function GetButtonRect(const Index: Integer): TRect;
```

C++

```
__fastcall TRect GetButtonRect(const int Index);
```

16.1.1.4.17 TButtonGroup.GetButtonsClass Method

Pascal

```
function GetButtonsClass: TGrpButtonItemsClass; virtual;
```

C++

```
virtual __fastcall TGrpButtonItemsClass GetButtonsClass();
```

Description

This is GetButtonsClass, a member of class TButtonGroup.

16.1.1.4.18 TButtonGroup.GetDragImages Method

Returns the control's image list containing the image to be displayed while dragging.

Pascal

```
function GetDragImages: TDragImageList; override;
```

C++

```
virtual __fastcall TDragImageList GetDragImages();
```

Description

Override GetDragImages to provide a custom image list for the control. As implemented in TControl, GetDragImages returns nil (Delphi) or NULL (C++).

See Also

DragCursor (see page 5818)

OnStartDrag (see page 5871)

16.1.1.4.19 TButtonGroup.IndexOfButtonAt Method

Retrieves the index of a button, given x and y coordinates.

Pascal

```
function IndexOfButtonAt(const X: Integer; const Y: Integer): Integer;
```

C++

```
__fastcall int IndexOfButtonAt(const int X, const int Y);
```

16.1.1.4.20 TButtonGroup.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove (see page 5773)

MouseUp (see page 5774)

OnMouseDown (see page 5862)

16.1.1.4.21 TButtonGroup.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown (see page 5773)

MouseUp (see page 5774)

OnMouseMove (see page 5865)

16.1.1.4.22 TButtonGroup.MouseUp Method

OnMouseUp event dispatcher.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user releases a previously pressed mouse button while the cursor's hotspot is over the control.

A control calls MouseUp in response to any of the Windows mouse-up messages (WM_LBUTTONUP, WM_MBUTTONUP, WM_RBUTTONUP), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

See Also

MouseDown (see page 5773)

MouseMove (see page 5773)

OnMouseUp (see page 5867)

16.1.1.4.23 TButtonGroup.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

16.1.1.4.24 TButtonGroup.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas (see page 5921)

PaintWindow (see page 5920)

GetDeviceContext (see page 6155)

ClipRect (see page 14486)

16.1.1.4.25 TButtonGroup.RemoveInsertionPoints Method

Removes the insertion points added by SetInsertionPoints (see page 2220).

Pascal

```
procedure RemoveInsertionPoints;
```

C++

```
__fastcall RemoveInsertionPoints();
```

16.1.1.4.26 TButtonGroup.Resize Method

Respond to control resize.

Pascal

```
procedure Resize; override;
```

C++

```
virtual __fastcall Resize();
```

Description

Resize is called automatically immediately after the control's dimensions change. As implemented in TControl, Resize calls the control's OnResize event handler if it is assigned.

See Also

OnResize (see page 5870)

16.1.1.4.27 TButtonGroup.ScrollIntoView Method

Scrolls buttons to make the the buttons visible in the group container.

Pascal

```
procedure ScrollIntoView(const Index: Integer);
```

C++

```
__fastcall ScrollIntoView(const int Index);
```

16.1.1.4.28 TButtonGroup.SetInsertionPoints Method

Draws an insert line for inserting at InsertionIndex. You can choose to show or hide insertion points.

Pascal

```
procedure SetInsertionPoints(const InsertionIndex: Integer);
```

C++

```
__fastcall SetInsertionPoints(const int InsertionIndex);
```

16.1.1.4.29 TButtonGroup.TargetIndexAt Method

Provides the target insertion index of a given coordinate (x,y). If the target is somewhere in or above the top half of a button, this method inserts an object above the button. If the target is somewhere in or below the bottom half of a button, the method inserts an object after the button.

Pascal

```
function TargetIndexAt(const X: Integer; const Y: Integer): Integer;
```

C++

```
__fastcall int TargetIndexAt(const int X, const int Y);
```

16.1.1.4.30 TButtonGroup.UpdateAllButtons Method

Pascal

```
procedure UpdateAllButtons;
```

C++

```
__fastcall UpdateAllButtons();
```

Description

This is UpdateAllButtons, a member of class TButtonGroup.

16.1.1.4.31 TButtonGroup.UpdateButton Method

Pascal

```
procedure UpdateButton(const Index: Integer);
```

C++

```
__fastcall UpdateButton(const int Index);
```

Description

This is UpdateButton, a member of class TButtonGroup.

16.1.1.5 TButtonGroup Properties

The properties of the TButtonGroup class are listed here.

Public Properties

	Name	Description
	Canvas (see page 2229)	Specifies the TCanvas object that presents a drawing surface for the control.
R	DragImageList (see page 2232)	DragImageList defines the list of images shown during drag operations.
	DragIndex (see page 2233)	DragIndex defines the index of the button during drag operations.

Published Properties

	Name	Description
	Align (see page 2221)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2225)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 2225)	Specifies which edges of the control are beveled.
	BevelInner (see page 2226)	Specifies the cut of the inner bevel.
	BevelKind (see page 2226)	Specifies the control's bevel style.
	BevelOuter (see page 2227)	Specifies the cut of the outer bevel.
	BevelWidth (see page 2227)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 2227)	BorderStyle specifies the appearance of the button group border.
	BorderWidth (see page 2228)	Specifies the width of the control's border.
	ButtonHeight (see page 2228)	ButtonHeight specifies the height of the buttons in the group.
	ButtonOptions (see page 2228)	Specifies options for button display and behavior.
	ButtonWidth (see page 2229)	ButtonWidth specifies the width of the buttons in the group.
	DockSite (see page 2231)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 2232)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2233)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2233)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2235)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2235)	Controls the attributes of text written on or in the control.
	Height (see page 2237)	Specifies the vertical size of the control in pixels.
	Images (see page 2238)	This is Images, a member of class TButtonGroup.
	ItemIndex (see page 2238)	Specifies the index of the currently selected item.
	Items (see page 2238)	Button items in this group.
	PopupMenu (see page 2238)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2243)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2244)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2245)	Determines if the user can tab to a control.
	Visible (see page 2245)	Determines whether the component appears onscreen.
	Width (see page 2247)	Specifies the horizontal size of the control or form in pixels.

16.1.1.5.1 TButtonGroup.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The

value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
```

```
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;
```

```
procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

16.1.1.5.2 TButtonGroup.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

16.1.1.5.3 TButtonGroup.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties

determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

16.1.1.5.4 TButtonGroup.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

16.1.1.5.5 TButtonGroup.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

16.1.1.5.6 TButtonGroup.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

16.1.1.5.7 TButtonGroup.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

16.1.1.5.8 TButtonGroup.BorderStyle Property

BorderStyle specifies the appearance of the button group border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to get or set the appearance of the button group border. BorderStyle can be any of the following values:

Value	Meaning
bsNone	Has no border.
bsSingle	Has a border, whose width is defined by BorderWidth (see page 2228).

See Also

BorderWidth (see page 2228)

16.1.1.5.9 TButtonGroup.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics or text drawn by the control is clipped to the area within the border.

See Also

ClientRect

BevelOuter (see page 6194)

16.1.1.5.10 TButtonGroup.ButtonHeight Property

ButtonHeight specifies the height of the buttons in the group.

Pascal

```
property ButtonHeight: Integer;
```

C++

```
__property int ButtonHeight;
```

Description

Use ButtonHeight to set the height of all of the buttons contained within the button group.

See Also

ButtonWidth (see page 2229)

16.1.1.5.11 TButtonGroup.ButtonOptions Property

Specifies options for button display and behavior.

Pascal

```
property ButtonOptions: TGrpButtonOptions;
```

C++

```
__property TGrpButtonOptions ButtonOptions;
```

Description

Set the following options on your button group:

Option	Description
gboAllowReorder	Allows the user to reorder buttons within the group at runtime.
gboFullSize	Sets the width of the buttons in the group to the maximum size, which is the entire width of the container.
gboGroupStyle	Specifies that the buttons should inherit the group style that is set on the container.
gboShowCaptions	Toggles the captions that you set in the Items (see page 2238) property.

16.1.1.5.12 TButtonGroup.ButtonWidth Property

ButtonWidth specifies the width of the buttons in the group.

Pascal

```
property ButtonWidth: Integer;
```

C++

```
__property int ButtonWidth;
```

Description

Use ButtonWidth to set the width of all of the buttons contained within the button group.

See Also

ButtonHeight (see page 2228)

16.1.1.5.13 TButtonGroup.Canvas Property

Specifies the TCanvas object that presents a drawing surface for the control.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control. Canvas encapsulates a Windows device context, providing all the tools and methods needed for drawing and painting.

Canvas is a protected property that is usually redeclared as public in descendants of TCustomControl.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
```

```
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas,Rect.Left,Rect.Top,index);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(index) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TStringGrid *myStringGrid;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TStringGrid")
  {
    myStringGrid = (TStringGrid *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myStringGrid->Canvas->MoveTo(myStringGrid->ClientRect.Left, myStringGrid->ClientRect.Top);
    myStringGrid->Canvas->LineTo(myStringGrid->ClientRect.Right,
myStringGrid->ClientRect.Bottom);
    myStringGrid = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
```

```
      DrawGrid1.Canvas.DrawFocusRect(Rect);
      Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
    end;
    ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  end;

  procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  begin
    CanSelect:= True;
  end;
  {
  The following code uses ClientRect to find and draw a line
  from the top left to the bottom right of the current
  control.  Select a TControl in the ListBox list.  This
  example assumes that the current object is a descendent of
  TControl and has a public Canvas property.
  }
    if (ListBox1.Items.Objects[Index] is TStringGrid) then
    begin
      myStringGrid:= (ListBox1.Items.Objects[Index] as TStringGrid);
      with myStringGrid.ClientRect do
      begin // make sure we are using the TControl's canvas and not the form's!
        myStringGrid.Canvas.MoveTo(Left, Top);
        myStringGrid.Canvas.LineTo(Right, Bottom);
      end;
    myStringGrid:= nil;
    end;
```

See Also

Paint (see page 5920)

16.1.1.5.14 TButtonGroup.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite: Boolean;
```

C++

```
__property Boolean DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
```

```
    if (Client->ClassNameIs("TDockableForm"))
      dynamic_cast<TDockableForm *>(Client)->DockSite = True;
    if ((DockClientCount == 2) && (NewTarget != this))        // Self?
      PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
    UpdateCaption(Client);
  }
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

16.1.1.5.15 TButtonGroup.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

16.1.1.5.16 TButtonGroup.DragImageList Property

DragImageList defines the list of images shown during drag operations.

Pascal

```
property DragImageList: TDragImageList;
```

C++

```
__property TDragImageList DragImageList;
```

Description

TDragImageList is used to manage images that are displayed during drag and drop or drag and dock operations. Use DragImageList to add images and control attributes.

See Also

TDragImageList (see page 6021)

16.1.1.5.17 TButtonGroup.DragIndex Property

DragIndex defines the index of the button during drag operations.

Pascal

```
property DragIndex: Integer;
```

C++

```
__property int DragIndex;
```

Description

If a drag operation is coming from this control, it is because they are dragging the item at DragIndex. Set DragIndex to control which item is being dragged before manually calling BeginDrag.

See Also

BeginDrag

DragImageList (see page 2232)

16.1.1.5.18 TButtonGroup.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

16.1.1.5.19 TButtonGroup.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
```

```
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

16.1.1.5.20 TButtonGroup.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

16.1.1.5.21 TButtonGroup.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

16.1.1.5.22 TButtonGroup.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
HideSelection, Height example
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
```

```
        HideSelection := False;
    end;
end;
```

See Also

ClientHeight (see page 5808)

Width (see page 5849)

Using TDBListBox and TDBComboBox

16.1.1.5.23 TButtonGroup.Images Property

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

This is Images, a member of class TButtonGroup.

16.1.1.5.24 TButtonGroup.ItemIndex Property

Specifies the index of the currently selected item.

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

16.1.1.5.25 TButtonGroup.Items Property

Button items in this group.

Pascal

```
property Items: TGrpButtonItems;
```

C++

```
__property TGrpButtonItems Items;
```

16.1.1.5.26 TButtonGroup.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
```

```
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}
```

```
void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
```

```
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
```

```
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

16.1.1.5.27 TButtonGroup.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
```

```
      itoa(i, index, 10);
      strcat(string, index);
      ListBox1->Items->Add(string);
    }
    ListBox1->Hint = "Select an item";
    ListBox1->ShowHint = true;
    Application->ShowHint = true;
  }
  /*
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  */
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

16.1.1.5.28 TButtonGroup.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder

value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

16.1.1.5.29 TButtonGroup.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

16.1.1.5.30 TButtonGroup.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false,

the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
```

```
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

16.1.1.5.31 TButtonGroup.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

See Also

ClientWidth (see page 5811)

Height (see page 5825)

Left (see page 5830)

16.1.1.6 TButtonGroup Events

The events of the TButtonGroup class are listed here.

Published Events

	Name	Description
	OnAfterDrawButton (see page 2248)	Occurs after a button is drawn.
	OnAlignInsertBefore (see page 2248)	Occurs when an object is aligned.
	OnAlignPosition (see page 2249)	Occurs when a control is aligned to a new position.
	OnBeforeDrawButton (see page 2249)	Occurs just before a button is drawn.
	OnButtonClicked (see page 2249)	Occurs when a button is clicked, if the OnClick (see page 2249) event is not present.
	OnClick (see page 2249)	Occurs when the user clicks the control.
	OnContextPopup (see page 2251)	Occurs when a user right-clicks on the dock site to display the popup menu.
	OnDockDrop (see page 2251)	Occurs when the control is dropped by a release of the mouse button.
	OnDockOver (see page 2251)	Occurs when the control is being moved over another control, but has not yet been docked.

	OnDragDrop (see page 2252)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2254)	Occurs when the user drags an object over a control.
	OnDrawButton (see page 2256)	Occurs when a button is redrawn.
	OnDrawIcon (see page 2256)	Occurs when an icon is redrawn.
	OnEndDock (see page 2257)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2257)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2258)	Occurs when the control gets focus.
	OnExit (see page 2258)	Occurs when the control loses focus, typically in tabbing between controls.
	OnHotButton (see page 2258)	Fired when the mouse pointer is over the button.
	OnKeyDown (see page 2259)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2261)	Occurs when a key is pressed.
	OnKeyUp (see page 2262)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2264)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2265)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2267)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2268)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2268)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2270)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 2273)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 2273)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 2273)	Occurs when the mouse wheel is rotated upward.
	OnReorderButton (see page 2274)	Fired when buttons are reordered in the button group.
	OnStartDock (see page 2274)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2274)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

16.1.1.6.1 TButtonGroup.OnAfterDrawButton Event

Occurs after a button is drawn.

Pascal

```
property OnAfterDrawButton: TGrpButtonDrawEvent;
```

C++

```
__property TGrpButtonDrawEvent OnAfterDrawButton;
```

16.1.1.6.2 TButtonGroup.OnAlignInsertBefore Event

Occurs when an object is aligned.

Pascal

```
property OnAlignInsertBefore: TAlignInsertBeforeEvent;
```

C++

```
__property TAlignInsertBeforeEvent OnAlignInsertBefore;
```

Description

When the object is aligned with other objects, it is inserted before other objects.

16.1.1.6.3 TButtonGroup.OnAlignPosition Event

Occurs when a control is aligned to a new position.

Pascal

```
property OnAlignPosition: TAlignPositionEvent;
```

C++

```
__property TAlignPositionEvent OnAlignPosition;
```

16.1.1.6.4 TButtonGroup.OnBeforeDrawButton Event

Occurs just before a button is drawn.

Pascal

```
property OnBeforeDrawButton: TGrpButtonDrawEvent;
```

C++

```
__property TGrpButtonDrawEvent OnBeforeDrawButton;
```

16.1.1.6.5 TButtonGroup.OnButtonClicked Event

Occurs when a button is clicked, if the OnClick (see page 2249) event is not present.

Pascal

```
property OnButtonClicked: TGrpButtonEvent;
```

C++

```
__property TGrpButtonEvent OnButtonClicked;
```

16.1.1.6.6 TButtonGroup.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B.

However, focus does not move to the control in these instances.

- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
```

```
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

16.1.1.6.7 TButtonGroup.OnContextPopup Event

Occurs when a user right-clicks on the dock site to display the popup menu.

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

Any action you add to this event may occur prior to the popup menu being displayed. You can test this yourself by adding a ShowMessage command to the event handler and running the application.

16.1.1.6.8 TButtonGroup.OnDockDrop Event

Occurs when the control is dropped by a release of the mouse button.

Pascal

```
property OnDockDrop: TDockDropEvent;
```

C++

```
__property TDockDropEvent OnDockDrop;
```

16.1.1.6.9 TButtonGroup.OnDockOver Event

Occurs when the control is being moved over another control, but has not yet been docked.

Pascal

```
property OnDockOver: TDockOverEvent;
```

C++

```
__property TDockOverEvent OnDockOver;
```

16.1.1.6.10 TButtonGroup.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
```

```
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

16.1.1.6.11 TButtonGroup.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
```

```
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
```

```
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

16.1.1.6.12 TButtonGroup.OnDrawButton Event

Occurs when a button is redrawn.

Pascal

```
property OnDrawButton: TGrpButtonDrawEvent;
```

C++

```
__property TGrpButtonDrawEvent OnDrawButton;
```

16.1.1.6.13 TButtonGroup.OnDrawIcon Event

Occurs when an icon is redrawn.

Pascal

```
property OnDrawIcon: TGrpButtonDrawIconEvent;
```

C++

```
__property TGrpButtonDrawIconEvent OnDrawIcon;
```

16.1.1.6.14 TButtonGroup.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

16.1.1.6.15 TButtonGroup.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
```

```
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

16.1.1.6.16 TButtonGroup.OnEnter Event

Occurs when the control gets focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

16.1.1.6.17 TButtonGroup.OnExit Event

Occurs when the control loses focus, typically in tabbing between controls.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

16.1.1.6.18 TButtonGroup.OnHotButton Event

Fired when the mouse pointer is over the button.

Pascal

```
property OnHotButton: TGrpButtonEvent;
```

C++

```
__property TGrpButtonEvent OnHotButton;
```

16.1.1.6.19 TButtonGroup.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
```

```
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}
```

```
uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

16.1.1.6.20 TButtonGroup.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

16.1.1.6.21 TButtonGroup.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
```

```
  event to FormKeyUp.  Set the KeyPreview on the form to True.
  }
  var
    FormColor: TColor;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    KeyPreview := True;
    FormColor := Form1.Color;
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

16.1.1.6.22 TButtonGroup.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

16.1.1.6.23 TButtonGroup.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
```

```
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
```

```
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

16.1.1.6.24 TButtonGroup.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

16.1.1.6.25 TButtonGroup.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

16.1.1.6.26 TButtonGroup.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
```

```
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
```

```
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

16.1.1.6.27 TButtonGroup.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
```

```
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

16.1.1.6.28 TButtonGroup.OnMouseWheel Event

Occurs when the mouse wheel is rotated.

Pascal

```
property OnMouseWheel: TMouseWheelEvent;
```

C++

```
__property TMouseWheelEvent OnMouseWheel;
```

Description

Write code in the OnMouseWheel event handler to respond to mouse wheel messages. If there is no OnMouseWheel event handler, or if the mouse wheel message is not handled in the OnMouseWheel event handler, then an OnMouseWheelUp or OnMouseWheelDown event occurs, depending on the direction the mouse wheel was rotated.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheelUp (see page 5870)

16.1.1.6.29 TButtonGroup.OnMouseWheelDown Event

Occurs when the mouse wheel is rotated downward.

Pascal

```
property OnMouseWheelDown: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelDown;
```

Description

Write code in the OnMouseWheelDown event handler to perform actions when the mouse wheel is rotated downward. OnMouseWheelDown occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheel (see page 5869)

OnMouseWheelUp (see page 5870)

16.1.1.6.30 TButtonGroup.OnMouseWheelUp Event

Occurs when the mouse wheel is rotated upward.

Pascal

```
property OnMouseWheelUp: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelUp;
```

Description

Write code in the OnMouseWheelUp event handler to perform actions when the mouse wheel is rotated upward. OnMouseWheelUp occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheel (see page 5869)

16.1.1.6.31 TButtonGroup.OnReorderButton Event

Fired when buttons are reordered in the button group.

Pascal

```
property OnReorderButton: TGrpButtonReorderEvent;
```

C++

```
__property TGrpButtonReorderEvent OnReorderButton;
```

16.1.1.6.32 TButtonGroup.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

16.1.1.6.33 TButtonGroup.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

16.1.2 TGrpButtonItem Class

TGrpButtonItem is an individual button of a TButtonGroup (see page 2198) control.

Class Hierarchy

Pascal

```
TGrpButtonItem = class(TBaseButtonItem);
```

C++

```
class TGrpButtonItem : public TBaseButtonItem;
```

File

ButtonGroup

16.1.2.1 TGrpButtonItem Members

The following tables list the members exposed by TGrpButtonItem.

Protected Methods

	Name	Description
	ActionChange (see page 2420)	This is ActionChange, a member of class TBaseButtonItem.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetActionLinkClass (see page 2421)	This is GetActionLinkClass, a member of class TBaseButtonItem.
	GetDisplayName (see page 2421)	Returns the name of the collection item as it appears in the collection editor.
	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
	GetNotifyTarget (see page 2277)	This is GetNotifyTarget, a member of class TGrpButtonItem.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Protected Properties

	Name	Description
	ActionLink (see page 2422)	This is ActionLink, a member of class TBaseButtonItem.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2420)	Copies the contents of another, similar object.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2419)	Creates and initializes a TCollectionItem instance.
	Create (see page 2757)	Creates and initializes a TCollectionItem (see page 2755) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	ScrollIntoView (see page 2278)	Scroll buttons to make the item visible

Public Properties

	Name	Description
	Collection (see page 2278)	The name of the collection that the button group items are contained in.
	Data (see page 2423)	This is Data, a member of class TBaseButtonItem.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Events

	Name	Description
	OnClick (see page 2423)	This is OnClick, a member of class TBaseButtonItem.

Published Properties

	Name	Description
	Action (see page 2422)	Specifies the action the button executes when clicked.
R	ButtonGroup (see page 2278)	The name of the button group that the button item belongs to.
	Caption (see page 2422)	This is Caption, a member of class TBaseButtonItem.
	Hint (see page 2423)	This is Hint, a member of class TBaseButtonItem.
	ImageIndex (see page 2423)	The index of a specified image.

16.1.2.2 TGrpButtonItem Methods

The methods of the TGrpButtonItem class are listed here.

Protected Methods

	Name	Description
V	GetNotifyTarget (see page 2277)	This is GetNotifyTarget, a member of class TGrpButtonItem.

Public Methods

	Name	Description
V	ScrollIntoView (see page 2278)	Scroll buttons to make the item visible

16.1.2.2.1 TGrpButtonItem.GetNotifyTarget Method

Pascal

```
function GetNotifyTarget: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetNotifyTarget();
```

Description

This is GetNotifyTarget, a member of class TGrpButtonItem.

16.1.2.2.2 TGrpButtonItem.ScrollIntoView Method

Scroll buttons to make the item visible

Pascal

```
procedure ScrollIntoView; override;
```

C++

```
virtual __fastcall ScrollIntoView();
```

16.1.2.3 TGrpButtonItem Properties

The properties of the TGrpButtonItem class are listed here.

Public Properties

	Name	Description
	Collection (see page 2278)	The name of the collection that the button group items are contained in.

Published Properties

	Name	Description
R	ButtonGroup (see page 2278)	The name of the button group that the button item belongs to.

16.1.2.3.1 TGrpButtonItem.ButtonGroup Property

The name of the button group that the button item belongs to.

Pascal

```
property ButtonGroup: TButtonGroup;
```

C++

```
__property TButtonGroup ButtonGroup;
```

16.1.2.3.2 TGrpButtonItem.Collection Property

The name of the collection that the button group items are contained in.

Pascal

```
property Collection: TGrpButtonItems;
```

C++

```
__property TGrpButtonItems Collection;
```

16.1.3 TGrpButtonItems Class

TGrpButtonItems maintains the collection of buttons that appear in a TButtonGroup (see page 2198) control.

Class Hierarchy

Pascal

```
TGrpButtonItems = class(TCollection);
```

C++

```
class TGrpButtonItems : public TCollection;
```

File

ButtonGroup

16.1.3.1 TGrpButtonItems Members

The following tables list the members exposed by TGrpButtonItems.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2281)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2282)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 2281)	Adds an item to the button group at the end of the list of buttons.
	AddItem (see page 2281)	Adds a button to the button group at a specified index position.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2281)	Updates the IDs of buttons in the button group when a button is removed, moved, or added.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 2280)	Constructor for the group button items.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2282)	Inserts a button item at a specified index.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ButtonGroup (see page 2283)	The button group that this item belongs to.
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 2283)	Lists the items in the collection.

16.1.3.2 TGrpButtonItems.Create Constructor

Constructor for the group button items.

Pascal

```
constructor Create(const ButtonGroup: TButtonGroup);
```

C++

```
__fastcall TGrpButtonItems(const TButtonGroup ButtonGroup);
```

16.1.3.3 TGrpButtonItems Methods

The methods of the TGrpButtonItems class are listed here.

Protected Methods

	Name	Description
	GetOwner (see page 2281)	Returns the owner of an object.
	Update (see page 2282)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 2281)	Adds an item to the button group at the end of the list of buttons.
	AddItem (see page 2281)	Adds a button to the button group at a specified index position.
	BeginUpdate (see page 2281)	Updates the IDs of buttons in the button group when a button is removed, moved, or added.
	Insert (see page 2282)	Inserts a button item at a specified index.

16.1.3.3.1 TGrpButtonItems.Add Method

Adds an item to the button group at the end of the list of buttons.

Pascal

```
function Add: TGrpButtonItem;
```

C++

```
__fastcall TGrpButtonItem Add();
```

16.1.3.3.2 TGrpButtonItems.AddItem Method

Adds a button to the button group at a specified index position.

Pascal

```
function AddItem(Item: TGrpButtonItem; Index: Integer): TGrpButtonItem;
```

C++

```
__fastcall TGrpButtonItem AddItem(TGrpButtonItem Item, int Index);
```

16.1.3.3.3 TGrpButtonItems.BeginUpdate Method

Updates the IDs of buttons in the button group when a button is removed, moved, or added.

Pascal

```
procedure BeginUpdate; override;
```

C++

```
virtual __fastcall BeginUpdate();
```

16.1.3.3.4 TGrpButtonItems.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

16.1.3.3.5 TGrpButtonItems.Insert Method

Inserts a button item at a specified index.

Pascal

```
function Insert(Index: Integer): TGrpButtonItem;
```

C++

```
__fastcall TGrpButtonItem Insert(int Index);
```

16.1.3.3.6 TGrpButtonItems.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed (see page 2740)

Changed (see page 2758)

16.1.3.4 TGrpButtonItems Properties

The properties of the TGrpButtonItems class are listed here.

Public Properties

	Name	Description
R	ButtonGroup (see page 2283)	The button group that this item belongs to.
	Items (see page 2283)	Lists the items in the collection.

16.1.3.4.1 TGrpButtonItems.ButtonGroup Property

The button group that this item belongs to.

Pascal

```
property ButtonGroup: TButtonGroup;
```

C++

```
__property TButtonGroup ButtonGroup;
```

16.1.3.4.2 TGrpButtonItems.Items Property

Lists the items in the collection.

Pascal

```
property Items [Index: Integer]: TGrpButtonItem;
```

C++

```
__property TGrpButtonItem Items[int Index];
```

Description

Use Items to access individual items in the collection. The value of the Index parameter corresponds to the Index property of TCollectionItem. It represents the position of the item in the collection.

See Also

Count (see page 2749)

ItemClass (see page 2751)

Index (see page 2762)

Add (see page 2736)

Insert (see page 2746)

16.2 Types

The following table lists types in this documentation.

Types

Name	Description
TGrpButtonDrawEvent (see page 2284)	TGrpButtonDrawEvent is the type of Button Group draw event handler.
TGrpButtonDrawIconEvent (see page 2285)	TGrpButtonDrawIconEvent is the type of Button Group OnDrawIcon event handler.
TGrpButtonEvent (see page 2285)	TGrpButtonEvent is a type of Button Group event handler.
TGrpButtonItemClass (see page 2286)	Metaclass of TGrpButtonItem (see page 2275).
TGrpButtonItemsClass (see page 2286)	Metaclass of TGrpButtonItems (see page 2278).
TGrpButtonOptions (see page 2286)	Options for button groups.
TGrpButtonReorderEvent (see page 2287)	TGrpButtonReorderEvent is a type of Button Group reorder event handler.

16.2.1 ButtonGroup.TGrpButtonDrawEvent Type

TGrpButtonDrawEvent is the type of Button Group draw event handler.

Pascal

```
TGrpButtonDrawEvent = procedure (Sender: TObject; Index: Integer; Canvas: TCanvas; Rect:
TRect; State: TButtonDrawState) of object;
```

C++

```
(Sender: TObject; Index: Integer; Canvas: TCanvas; Rect: TRect; State: TButtonDrawState) (
TGrpButtonDrawEvent)();
```

File

ButtonGroup

Description

TGrpButtonDrawEvent defines the type of method procedure that you must define in order to implement the OnAfterDrawButton, OnBeforeDrawButton and OnDrawButton events of the TButtonGroup (see page 2198) class.

Each of these events is invoked once per button in the button group.

Sender is a reference to the TButtonGroup (see page 2198) invoking the event.

Index is the index of the button in the button group.

Canvas is the canvas on which the drawing takes place.

Rect defines the rectangle in which the user drawing must be made.

State defines state of the button to be drawn.

See Also

OnAfterDrawButton (see page 2248)

OnBeforeDrawButton (see page 2249)

OnDrawButton (see page 2256)

TButtonDrawState (see page 2545)

16.2.2 ButtonGroup.TGrpButtonDrawIconEvent Type

TGrpButtonDrawIconEvent is the type of Button Group OnDrawIcon event handler.

Pascal

```
TGrpButtonDrawIconEvent = procedure (Sender: TObject; Index: Integer; Canvas: TCanvas; Rect:
TRect; State: TButtonDrawState; var TextOffset: Integer) of object;
```

C++

```
(Sender: TObject; Index: Integer; Canvas: TCanvas; Rect: TRect; State: TButtonDrawState; var
TextOffset: Integer) ( TGrpButtonDrawIconEvent)();
```

File

ButtonGroup

Description

TGrpButtonDrawEvent (see page 2284) defines the type of method procedure that you must define in order to implement the OnDrawIcon event which is called when a button icon is drawn or redrawn.

This event is invoked once per button icon in the button group.

Sender is a reference to the TButtonGroup (see page 2198) invoking the event.

Index is the index of the button in the button group.

Canvas is the canvas on which the drawing takes place.

State defines state of the button to be drawn.

TextOffset defines the offset of the button text in pixels.

See Also

OnDrawIcon (see page 2256)

TButtonDrawState (see page 2545)

16.2.3 ButtonGroup.TGrpButtonEvent Type

TGrpButtonEvent is a type of Button Group event handler.

Pascal

```
TGrpButtonEvent = procedure (Sender: TObject; Index: Integer) of object;
```

C++

```
(Sender: TObject; Index: Integer) ( TGrpButtonEvent)();
```

File

ButtonGroup

Description

TGrpButtonEvent defines the type of method procedure that you must define in order to implement the OnButtonClicked and

OnHotButton events.

This event is invoked for a button in the button group.

Sender is a reference to the TButtonGroup (see page 2198) invoking the event.

Index is the index of the button in the button group.

See Also

OnButtonClicked (see page 2249)

OnHotButton (see page 2258)

16.2.4 ButtonGroup.TGrpButtonItemClass Type

Metaclass of TGrpButtonItem (see page 2275).

Pascal

```
TGrpButtonItemClass = class of TGrpButtonItem;
```

C++

```
TGrpButtonItem TGrpButtonItemClass;
```

File

ButtonGroup

16.2.5 ButtonGroup.TGrpButtonItemsClass Type

Metaclass of TGrpButtonItems (see page 2278).

Pascal

```
TGrpButtonItemsClass = class of TGrpButtonItems;
```

C++

```
TGrpButtonItems TGrpButtonItemsClass;
```

File

ButtonGroup

16.2.6 ButtonGroup.TGrpButtonOptions Type

Options for button groups.

Pascal

```
TGrpButtonOptions = set of (gboAllowReorder, gboFullSize, gboGroupStyle, gboShowCaptions);
```

C++

```
(gboAllowReorder, gboFullSize, gboGroupStyle, gboShowCaptions) TGrpButtonOptions;
```

File

ButtonGroup

Description

The following options are available:

Option	Description
gboAllowReorder	Allows the user to reorder buttons within the group at runtime by dragging buttons to new locations within the group.
gboFullSize	Resizes the buttons to take up the entire width of the button group container.
gboGroupStyle	Shows selected button as greyed out or darker than unselected buttons.
gboShowCaptions	Displays captions on each button.

16.2.7 ButtonGroup.TGrpButtonReorderEvent Type

TGrpButtonReorderEvent is a type of Button Group reorder event handler.

Pascal

```
TGrpButtonReorderEvent = procedure (Sender: TObject; OldIndex, NewIndex: Integer) of object;
```

C++

```
(Sender: TObject; OldIndex, NewIndex: Integer) ( TGrpButtonReorderEvent)();
```

File

ButtonGroup

Description

TGrpButtonEvent (see page 2285) defines the type of method procedure that you must define in order to implement the OnReorderButton and event.

This event is invoked when buttons are reordered in the button group.

Sender is a reference to the TButtonGroup (see page 2198) invoking the event.

OldIndex is the index of the original button position in the button group.

NewIndex is the index of the relocated button position in the button group.

See Also

OnReorderButton (see page 2274)

17 Buttons Namespace

This is namespace Buttons.

Topics

Name	Description
Classes (see page 2290)	The following table lists classes in this documentation.
Functions (see page 2391)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 2392)	The following table lists structs, records, enums in this documentation.
Types (see page 2396)	The following table lists types in this documentation.

Classes

	Name	Description
	TBitBtn (see page 2290)	TBitBtn is a push button control that can include a bitmap on its face.
	TBitBtnActionLink (see page 2330)	TBitBtnActionLink is an action link designed specifically for button control clients.
	TSpeedButton (see page 2335)	TSpeedButton is a button that is used to execute commands or set modes.
	TSpeedButtonActionLink (see page 2382)	TSpeedButtonActionLink is an action link designed specifically for button control clients.

Functions

	Name	Description
	DrawButtonFace (see page 2391)	Draws a push button.

Structs, Records, Enums

	Name	Description
	TBitBtnKind (see page 2392)	TBitBtnKind identifies the appearance and use of a TBitBtn (see page 2290) control.
	TButtonLayout (see page 2393)	TButtonLayout defines values for the Layout property of buttons.
	TButtonState (see page 2394)	TButtonState describes the current state of a button.
	TButtonStyle (see page 2395)	TButtonStyle specifies the basic look of a TBitBtn (see page 2290) control.

Types

Name	Description
TNumGlyphs (see page 2396)	This is type Buttons.TNumGlyphs.

17.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TBitBtn (see page 2290)	TBitBtn is a push button control that can include a bitmap on its face.
	TBitBtnActionLink (see page 2330)	TBitBtnActionLink is an action link designed specifically for button control clients.
	TSpeedButton (see page 2335)	TSpeedButton is a button that is used to execute commands or set modes.
	TSpeedButtonActionLink (see page 2382)	TSpeedButtonActionLink is an action link designed specifically for button control clients.

17.1.1 TBitBtn Class

TBitBtn is a push button control that can include a bitmap on its face.

Class Hierarchy

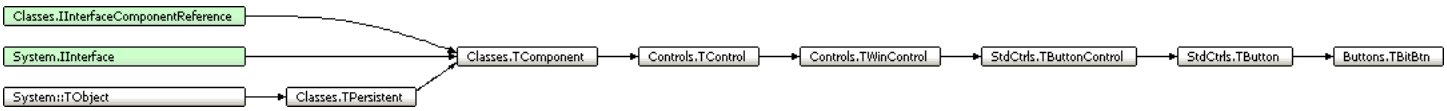

Pascal

```
TBitBtn = class(TButton);
```

C++

```
class TBitBtn : public TButton;
```

File

Buttons

Description

Bitmap buttons exhibit the same behavior as button controls. Use them to initiate actions from forms and dialog boxes.

Bitmap buttons implement properties that specify the bitmap images, along with their appearance and placement on the button. You can choose from predefined bitmap buttons styles or use your own customized bitmap for the button. Although the button can be associated with only one bitmap, the bitmap (glyph property) can be subdivided into four equal-sized portions, which are displayed based on the state of the button: up, down, disabled, and clicked.

The Kind (see page 2317) property of TBitBtn provides commonly used buttons, such as OK, Cancel (see page 2312), Help, and so on. These predefined button types have corresponding graphical images and default behaviors, so you can easily add them to your application with little or no coding necessary.

The recommended way to implement the response of other button kinds to user clicks is to assign an action from an action list as the value of the Action (see page 2307) property. By setting the Action (see page 2307) property, you make the button a client of the action, and the action handles updating the button's properties and responding when the user clicks the button.

If you are not using the built-in response to a specific kind of button or an action to respond when the user clicks the button, then you can specify the button's response by writing an OnClick event handler.

See Also

TButton (see page 20909)

TSpeedButton (see page 2335)

Bitmap Buttons

Buttons and Similar Controls

17.1.1.1 TBitBtn Members

The following tables list the members exposed by TBitBtn.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActionChange (see page 2303)	Updates the button control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CopyImage (see page 2304)	This is CopyImage, a member of class TBitBtn.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 2304)	Creates underlying screen object.
V	CreateParams (see page 2304)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CreateWnd (see page 20922)	Creates the underlying window.

	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 2305)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 20993)	Returns whether the button control is checked.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.

	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 2305)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.

	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetButtonStyle (see page 2306)	This is SetButtonStyle, a member of class TBitBtn.
	SetChecked (see page 20994)	Sets the Checked (see page 20995) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.

	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WndProc (see page 20994)	Handles Windows messages.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Checked (see page 20995)	Specifies whether the button control is checked.
	ClicksDisabled (see page 20996)	This is ClicksDisabled, a member of class TButtonControl.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

	WordWrap (see page 20996)	Specifies whether the button text wraps to fit the width of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Click (see page 2303)	Simulates a mouse click, as if the user had clicked the button.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 2302)	Constructs an instance of the button control and initializes its values.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2302)	Destroys the bitmap button object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
V	UseRightToLeftAlignment (see page 20923)	Specifies whether the control's alignment is in a right-to-left mode.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.

R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 20949)	Occurs when the user clicks the control.
	OnContextPopup (see page 20951)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 20951)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 20954)	Occurs when the user drags an object over a control.
	OnEndDock (see page 20956)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 20956)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2326)	Occurs when a control receives the input focus.
	OnExit (see page 2328)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 20961)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 20964)	Occurs when a key is pressed.
	OnKeyUp (see page 20965)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 20967)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 20968)	Occurs when the user presses a mouse button with the mouse pointer over a control.

	OnMouseEnter (see page 20970)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 20970)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 20971)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 20973)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 20975)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 20976)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Action (see page 2307)	Designates the action associated with the control.
	Align (see page 2307)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 2310)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2311)	Specifies the bi-directional mode for the control.
	Cancel (see page 2312)	Determines whether the button's OnClick event handler executes when the Escape key is pressed.
	Caption (see page 2312)	Specifies a text string that identifies the control to the user.
	Constraints (see page 2315)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Default (see page 2315)	Determines whether the button's OnClick event handler executes when the Enter key is pressed.
	DragCursor (see page 20933)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 20934)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 20934)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2316)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 20936)	Controls the attributes of text written on or in the control.
	Glyph (see page 2317)	Specifies the bitmap that appears on the bitmap button.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Kind (see page 2317)	Specifies the kind of bitmap button.
	Layout (see page 2318)	Specifies where the image appears on the bitmap button.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margin (see page 2318)	Specifies the number of pixels between the edge of the image (specified in the Glyph (see page 2317) property) and the edge of the button.
	Margins (see page 5831)	Specifies the margins for the control.
	ModalResult (see page 2319)	Determines whether and how the button closes its (modal) parent form.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NumGlyphs (see page 2320)	Indicates the number of images that are in the graphic specified in the Glyph (see page 2317) property.
	ParentBiDiMode (see page 2320)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 20938)	Determines where a control looks for its font information.
	ParentShowHint (see page 2320)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 20939)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2321)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Spacing (see page 2322)	Determines where the image and text appear on a bitmap or speed button.
	Style (see page 2323)	Determines the appearance of a bitmap button.

	TabOrder (see page 2323)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2324)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 2324)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	WordWrap (see page 20948)	Specifies whether the button text wraps to fit the width of the control.

17.1.1.2 TBitBtn.Create Constructor

Constructs an instance of the button control and initializes its values.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBitBtn(TComponent * AOwner);
```

Description

Use Create to programmatically instantiate a button. Buttons (see page 2289) added using the form designer are created automatically.

The AOwner parameter specifies the component, typically a form, that is responsible for freeing the button.

See Also

Owner (see page 2807)

Destroy (see page 2302)

17.1.1.3 TBitBtn.Destroy Destructor

Destroys the bitmap button object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBitBtn();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the bitmap button is not nil, and only then calls Destroy.

See Also

delete

Free (see page 22300)

17.1.1.4 TBitBtn Methods

The methods of the TBitBtn class are listed here.

Protected Methods

	Name	Description
	ActionChange (see page 2303)	Updates the button control to reflect changes in its associated action.
	CopyImage (see page 2304)	This is CopyImage, a member of class TBitBtn.
	CreateHandle (see page 2304)	Creates underlying screen object.
	CreateParams (see page 2304)	Initializes a window-creation parameter data structure.
	GetActionLinkClass (see page 2305)	Returns the appropriate action link class that is associated with button controls.
	GetPalette (see page 2305)	Returns the handle to a palette for use by the control.
	SetButtonStyle (see page 2306)	This is SetButtonStyle, a member of class TBitBtn.

Public Methods

	Name	Description
	Click (see page 2303)	Simulates a mouse click, as if the user had clicked the button.

17.1.1.4.1 TBitBtn.ActionChange Method

Updates the button control to reflect changes in its associated action.

Pascal

```
procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); override;
```

C++

```
virtual __fastcall ActionChange(TObject * Sender, Boolean CheckDefaults);
```

Description

ActionChange is called automatically when the associated action changes. It updates the button control to reflect the action's current Checked state. Then it calls the inherited method, which checks the associated properties for that client type.

The Sender parameter is the associated action. The CheckDefaults parameter specifies whether properties and event handlers should be changed if they differ from the default values. When CheckDefaults is true, properties and event handlers are only changed when the current values are the default values (that is, when the values have not been previously changed). When CheckDefaults is false, properties and event handlers are changed, regardless of their current values.

See Also

TCustomAction (see page 297)

17.1.1.4.2 TBitBtn.Click Method

Simulates a mouse click, as if the user had clicked the button.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Use Click to simulate a mouse click. Click performs the following tasks:

1. If the value of the Kind (see page 2317) property is bkClose, Click closes the form. If the value of Kind (see page 2317) is bkHelp, Click displays the Help screen assigned with the HelpContext property.
2. Any associated action fires, or, if there is no associated action, any code attached to the OnClick event is executed.

See Also

Kind (see page 2317)

Action (see page 5797)

OnClick (see page 5851)

17.1.1.4.3 TBitBtn.CopyImage Method

Pascal

```
procedure CopyImage(ImageList: TCustomImageList; Index: Integer);
```

C++

```
__fastcall CopyImage(TCustomImageList ImageList, int Index);
```

Description

This is CopyImage, a member of class TBitBtn.

17.1.1.4.4 TBitBtn.CreateHandle Method

Creates underlying screen object.

Pascal

```
procedure CreateHandle; override;
```

C++

```
virtual __fastcall CreateHandle();
```

Description

CreateHandle creates the Microsoft Windows screen object that underlies the control. if it does not already exist. The new window handle becomes the value of the Handle property.

CreateHandle creates or locates the parent control's window before it creates a window for the control itself.

Note: If the Parent property is nil (Delphi) or NULL (C++), CreateHandle raises an exception.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

Handle (see page 6207)

Parent (see page 5832)

17.1.1.4.5 TBitBtn.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

17.1.1.4.6 TBitBtn.GetActionLinkClass Method

Returns the appropriate action link class that is associated with button controls.

Pascal

```
function GetActionLinkClass: TControlActionLinkClass; override;
```

C++

```
virtual __fastcall TControlActionLinkClass GetActionLinkClass();
```

Description

GetActionLinkClass is used to get the appropriate class type for the action link used with button controls. GetActionLinkClass is used internally to create action link objects for clients when a button control has an associated action. The action link is used to link the action to the button control client. Each Action link class type is designed to link specific actions to clients based upon the types of properties the client supports. For button controls, GetActionLinkClass returns the TButtonActionLink class type.

See Also

TButtonActionLink (see page 20976)

17.1.1.4.7 TBitBtn.GetPalette Method

Returns the handle to a palette for use by the control.

Pascal

```
function GetPalette: HPALETTE; override;
```

C++

```
virtual __fastcall HPALETTE GetPalette();
```

Description

As implemented in TControl, GetPalette returns a null handle, indicating that the control does not have a palette. Override GetPalette to implement derived objects that use a palette. Create a palette handle for the control by a call to the Windows API function CreatePalette. GetPalette should return the palette handle for the control.

Specifying the palette for a control tells the application that the control's palette needs to be realized and designates the palette

to use for realization.

GetPalette is called only if the run-time video mode requires color palette support, for example, in 256-color mode.

See Also

PaletteChanged (see page 5775)

17.1.1.4.8 TBitBtn.SetButtonStyle Method

Pascal

```
procedure SetButtonStyle(ADefault: Boolean); override;
```

C++

```
virtual __fastcall SetButtonStyle(Boolean ADefault);
```

Description

This is SetButtonStyle, a member of class TBitBtn.

17.1.1.5 TBitBtn Properties

The properties of the TBitBtn class are listed here.

Published Properties

	Name	Description
	Action (see page 2307)	Designates the action associated with the control.
	Align (see page 2307)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2310)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2311)	Specifies the bi-directional mode for the control.
	Cancel (see page 2312)	Determines whether the button's OnClick event handler executes when the Escape key is pressed.
	Caption (see page 2312)	Specifies a text string that identifies the control to the user.
	Constraints (see page 2315)	Specifies the size constraints for the control.
	Default (see page 2315)	Determines whether the button's OnClick event handler executes when the Enter key is pressed.
	Enabled (see page 2316)	Controls whether the control responds to mouse, keyboard, and timer events.
	Glyph (see page 2317)	Specifies the bitmap that appears on the bitmap button.
	Kind (see page 2317)	Specifies the kind of bitmap button.
	Layout (see page 2318)	Specifies where the image appears on the bitmap button.
	Margin (see page 2318)	Specifies the number of pixels between the edge of the image (specified in the Glyph (see page 2317) property) and the edge of the button.
	ModalResult (see page 2319)	Determines whether and how the button closes its (modal) parent form.
	NumGlyphs (see page 2320)	Indicates the number of images that are in the graphic specified in the Glyph (see page 2317) property.
	ParentBiDiMode (see page 2320)	Specifies whether the control uses its parent's BiDiMode.
	ParentShowHint (see page 2320)	Determines where a control looks to find out if its Help Hint should be shown.
	ShowHint (see page 2321)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Spacing (see page 2322)	Determines where the image and text appear on a bitmap or speed button.
	Style (see page 2323)	Determines the appearance of a bitmap button.
	TabOrder (see page 2323)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2324)	Determines if the user can tab to a control.
	Visible (see page 2324)	Determines whether the component appears onscreen.

17.1.1.5.1 TBitBtn.Action Property

Designates the action associated with the control.

Pascal

```
property Action: TBasicAction;
```

C++

```
__property TBasicAction Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

17.1.1.5.2 TBitBtn.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
```

```
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
```

```
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

17.1.1.5.3 TBitBtn.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

17.1.1.5.4 TBitBtn.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

17.1.1.5.5 TBitBtn.Cancel Property

Determines whether the button's OnClick event handler executes when the Escape key is pressed.

Pascal

```
property Cancel: Boolean;
```

C++

```
__property Boolean Cancel;
```

Description

If Cancel is true, the button's OnClick event handler executes when the user presses Esc. Although an application can have more than one Cancel button, the form calls the OnClick event handler only for the first visible button in the tab order.

See Also

Default (see page 20932)

OnClick (see page 5851)

17.1.1.5.6 TBitBtn.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
```

```
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

17.1.1.5.7 TBitBtn.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

17.1.1.5.8 TBitBtn.Default Property

Determines whether the button's OnClick event handler executes when the Enter key is pressed.

Pascal

```
property Default: Boolean;
```

C++

```
__property Boolean Default;
```

Description

If Default is true, the button's OnClick event handler executes when the user presses Enter.

Although an application can have more than one Default button, the form calls the OnClick event handler only for the first visible button in the tab order. Moreover, any button that has focus becomes the Default button temporarily; hence, if the user selects another button before pressing Enter, the selected button's OnClick event handler executes instead.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
```

```
and enters a path or filename in the edit box and presses
the button, all the controls update:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FileListBox1->FileEdit = Edit1;
  FilterComboBox1->FileList = FileListBox1;
  DirectoryListBox1->FileList = FileListBox1;
  DirectoryListBox1->DirLabel = Label1;
  DriveComboBox1->DirList = DirectoryListBox1;
  Button1->Default = true;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FileListBox1->ApplyFilePath(Edit1->Text);
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FileListBox1.ApplyFilePath(Edit1.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileEdit := Edit1;
  FilterComboBox1.FileList := FileListBox1;
  DirectoryListBox1.FileList := FileListBox1;
  DirectoryListBox1.DirLabel := Label1;
  DriveComboBox1.DirList := DirectoryListBox1;
  Button1.Default := True;
end;
```

See Also

Cancel (see page 20929)

OnClick (see page 5851)

17.1.1.5.9 TBitBtn.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

17.1.1.5.10 TBitBtn.Glyph Property

Specifies the bitmap that appears on the bitmap button.

Pascal

```
property Glyph: TBitmap;
```

C++

```
__property TBitmap Glyph;
```

Description

Use the Open dialog box that appears as an editor in the Object Inspector to choose a bitmap file (with a .BMP extension) to use on the button, or specify a TBitmap object at runtime.

You can provide up to four images within a single bitmap. All images must be the same size and next to each other in a row. Bit buttons display one of these images depending on their state.

Image position	Button state	Description
First	Up	This image appears when the button is up (unselected). This image is also used when the button has focus (for example, if the user tabs to it); in this case, a focus rectangle is drawn around the button. If no other images exist in the bitmap, bit buttons also use this image for all other states.
Second	Disabled	This image usually appears dimmed to indicate that the button can't be selected.
Third	Clicked	This image appears when the button is clicked. The Up image reappears when the user releases the mouse button.
Fourth	Down	This image appears when the button stays down (indicating that it remains selected).

If only one image is present, bit buttons attempt to represent the other states by altering the image slightly for the different states, although the Down state is always the same as the Up state. If you aren't satisfied with the results, you can provide additional images in the bitmap.

If you have multiple images in a bitmap, you must specify the number of images that are in the bitmap with the Num Glyphs property.

Note: The lower left pixel of the bitmap is reserved for the "transparent" color. Any pixel in the bitmap which matches that lower left pixel will be transparent.

See Also

Kind (see page 2317)

NumGlyphs (see page 2320)

17.1.1.5.11 TBitBtn.Kind Property

Specifies the kind of bitmap button.

Pascal

```
property Kind: TBitBtnKind;
```

C++

```
__property TBitBtnKind Kind;
```

Description

Use Kind to specify the appearance of the TBitBtn (see page 2290) control and its response when the user clicks the button. A standard set of buttons are included and operate as expected without the addition of code. However, you can write additional code in the button events, if you want to. This does not include the Custom button kind, which can be programmed in any way you like.

The images (such as the green check mark on the OK button) appear on the button when using this version of TBitBtn (see page 2290).

See Also

ModalResult (see page 20937)

Glyph (see page 2317)

Style (see page 2323)

17.1.1.5.12 TBitBtn.Layout Property

Specifies where the image appears on the bitmap button.

Pascal

```
property Layout: TButtonLayout;
```

C++

```
__property TButtonLayout Layout;
```

Description

Layout indicates whether the text appears on the left of the button (blGlyphLeft), the right of the button (blGlyphRight), the top (blGlyphTop) or the bottom (blGlyphBottom).

See Also

Glyph (see page 2317)

Kind (see page 2317)

Margin (see page 2318)

17.1.1.5.13 TBitBtn.Margin Property

Specifies the number of pixels between the edge of the image (specified in the Glyph (see page 2317) property) and the edge of the button.

Pascal

```
property Margin: Integer;
```

C++

```
__property int Margin;
```

Description

The margin is the space between the image edge and the button edge. The format of the edges depends on the layout of the image and text (specified in the Layout (see page 2318) property). For example, if Layout (see page 2318) is blGlyphLeft, the margin appears between the left edge of the image and the left edge of the button. If Margin is 3, three pixels separate the image and the button edges. If Margin is 0, there is no space between the image and the button edges.

If Margin is -1 (the default value), the image and text (specified in the Caption (see page 2312) property) are centered. The number of pixels between the image and button edge is equal to the number of pixels between the opposite edge of the button and the text.

See Also

Glyph (see page 2317)

Layout (see page 2318)

Spacing (see page 2322)

Caption (see page 5806)

17.1.1.5.14 TBitBtn.ModalResult Property

Determines whether and how the button closes its (modal) parent form.

Pascal

```
property ModalResult: TModalResult;
```

C++

```
__property TModalResult ModalResult;
```

Description

Setting the TButton component's ModalResult property is an easy way to make clicking the button close a modal form. When a button is clicked, the ModalResult property of its parent form is set to the same value as the button's ModalResult property.

For example, if a dialog box has OK and Cancel buttons, their ModalResult properties could be set at design time to mrOk and mrCancel, respectively. At runtime, clicking the OK button then changes the dialog's ModalResult property to mrOk, and clicking the Cancel button changes the dialog's ModalResult property to mrCancel. Unless further processing is required, no OnClick event handlers are required for the buttons.

The following constants are defined for the ModalResult property:

Constant	Value	Meaning
mrNone	0	None. Used as a default value before the user exits.
mrOk	idOK	The user exited with OK button.
mrCancel	idCancel	The user exited with the CANCEL button.
mrAbort	idAbort	The user exited with the ABORT button.
mrRetry	idRetry	The user exited with the RETRY button.
mrIgnore	idIgnore	The user exited with the IGNORE button.
mrYes	idYes	The user exited with the YES button.
mrNo	idNo	The user exited with the NO button.
mrAll	mrNo + 1	The user exited with the ALL button.
mrNoToAll	mrAll + 1	The user exited with the NO TO ALL button.

mrYesToAll	mrNoToAll + 1	The user exited with the YES TO ALL button.

17.1.1.5.15 TBitBtn.NumGlyphs Property

Indicates the number of images that are in the graphic specified in the Glyph (see page 2317) property.

Pascal

```
property NumGlyphs: TNumGlyphs;
```

C++

```
__property TNumGlyphs NumGlyphs;
```

Description

If you have multiple images in a bitmap, you must specify the number of images that are in the bitmap with the NumGlyphs property. The default value is 1.

See Also

Glyph (see page 2317)

17.1.1.5.16 TBitBtn.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

17.1.1.5.17 TBitBtn.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

17.1.1.5.18 TBitBtn.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;
```

```
    char string[10];
    char index[3];
    for(int i = 1; i <= 10; i++)
    {
      strcpy(string, "Item");
      itoa(i, index, 10);
      strcat(string, index);
      ListBox1->Items->Add(string);
    }
    ListBox1->Hint = "Select an item";
    ListBox1->ShowHint = true;
    Application->ShowHint = true;
  }
  /*
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  */
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

17.1.1.5.19 TBitBtn.Spacing Property

Determines where the image and text appear on a bitmap or speed button.

Pascal

```
property Spacing: Integer;
```

C++

```
__property int Spacing;
```

Description

Spacing determines the number of pixels between the image (specified in the Glyph (see page 2317) property) and the text (specified in the Caption (see page 2312) property). The default value is 4 pixels.

If Spacing is a positive number, its value is the number of pixels between the image and text. If Spacing is 0, no pixels will be between the image and text. If Spacing is -1, the text appears centered between the image and the button edge. The number of pixels between the image and text is equal to the number of pixels between the text and the button edge opposite the glyph.

See Also

Glyph (see page 2317)

Layout (see page 2318)

Margin (see page 2318)

Caption (see page 5806)

17.1.1.5.20 TBitBtn.Style Property

Determines the appearance of a bitmap button.

Pascal

```
property Style: TButtonStyle;
```

C++

```
__property TButtonStyle Style;
```

Description

Style indicates whether the TBitBtn (see page 2290) control uses the modern button look, or reflects the look of a 16-bit Windows pushbutton.

See Also

Kind (see page 2317)

17.1.1.5.21 TBitBtn.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the

same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

17.1.1.5.22 TBitBtn.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

17.1.1.5.23 TBitBtn.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
```

```
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
```

```
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

17.1.1.6 TBitBtn Events

The events of the TBitBtn class are listed here.

Published Events

	Name	Description
	OnEnter (see page 2326)	Occurs when a control receives the input focus.
	OnExit (see page 2328)	Occurs when the input focus shifts away from one control to another.

17.1.1.6.1 TBitBtn.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
```

```
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
```

```
    FontDialog1.Execute;
  end;

  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    myActiveControl := ActiveControl;
  end;

  procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
  begin
    if myActiveControl is TEdit then
      with myActiveControl as TEdit do
         Font.Assign(TFontDialog(Sender).Font)
    else if myActiveControl is TRichEdit then
      with myActiveControl as TRichEdit do
        SelAttributes.Assign(TFontDialog(Sender).Font)
    else
      Beep;
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

17.1.1.6.2 TBitBtn.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
```

```
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

17.1.2 TBitBtnActionLink Class

TBitBtnActionLink is an action link designed specifically for button control clients.

Class Hierarchy

Pascal

```
TBitBtnActionLink = class(TControlActionLink);
```

C++

```
class TBitBtnActionLink : public TControlActionLink;
```

File

Buttons

Description

TBitBtnActionLink is designed to specifically link actions to bit buttton control clients. TBitBtnActionLink overrides methods to determine whether the action and client are linked, and to ensure that action events are appropriate for properties of bit button control clients.

TBitBtnActionLink inherits, from its ancestors, support for linking actions with the properties of components, generic controls, and windowed controls. TBitBtnActionLink adds support for linking the checked property to the appropriate property of a bit button control.

17.1.2.1 TBitBtnActionLink Members

The following tables list the members exposed by TBitBtnActionLink.

Protected Fields

	Name	Description
	FAction (see page 2722)	This is FAction, a member of class TBasicActionLink.
	FClient (see page 2333)	This is FClient, a member of class TBitBtnActionLink.
	FImageIndex (see page 2333)	This is FImageIndex, a member of class TBitBtnActionLink.

Protected Methods

	Name	Description
	AssignClient (see page 2334)	Assigns a client control that is linked to the associated action.
	Change (see page 2723)	This is Change, a member of class TBasicActionLink.
	DoShowHint (see page 5896)	Indicates whether the hint is shown, and modified.
	IsCaptionLinked (see page 268)	Indicates whether the Caption properties for the action and client are linked.
	IsCaptionLinked (see page 5896)	Indicates whether the caption for the action and the client are linked.
	IsCheckedLinked (see page 269)	Indicates whether the Checked properties for the action and client are linked.
	IsDropdownMenuLinked (see page 5896)	This is IsDropdownMenuLinked, a member of class TControlActionLink.
	IsEnabledLinked (see page 269)	Indicates whether the Enabled properties for the action and client are linked.
	IsEnabledLinked (see page 5897)	Indicates whether the enabled state for the action and the client are linked.
	IsEnableDropdownLinked (see page 5897)	This is IsEnableDropdownLinked, a member of class TControlActionLink.
	IsGroupIndexLinked (see page 270)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsHelpContextLinked (see page 270)	Indicates whether the HelpContext properties for the action and client controls are linked.
	IsHelpLinked (see page 271)	Indicates whether the help properties for the action and client are linked.
	IsHelpLinked (see page 5897)	Indicates whether the help properties for the action and the client are linked.
	IsHintLinked (see page 271)	Indicates whether the Hint properties for the action and client are linked.
	IsHintLinked (see page 5898)	Indicates whether the Hint property for the action and the client are linked.
	IsImageIndexLinked (see page 2334)	Indicates whether the image index for the action and the client are linked.
	IsOnExecuteLinked (see page 2723)	This is IsOnExecuteLinked, a member of class TBasicActionLink.
	IsOnExecuteLinked (see page 5898)	Indicates whether the OnExecute event handler for the action and the client are linked.
	IsPopupMenuLinked (see page 5898)	This is IsPopupMenuLinked, a member of class TControlActionLink.
	IsShortCutLinked (see page 272)	Indicates whether the ShortCut properties for the action and client are linked.
	IsVisibleLinked (see page 273)	Indicates whether the Visible properties for the action and client are linked.
	IsVisibleLinked (see page 5899)	Indicates whether the visible state for the action and the client are linked.
	SetAction (see page 2724)	This is SetAction, a member of class TBasicActionLink.
	SetAutoCheck (see page 274)	Introduces an interface for setting the AutoCheck property of a client.
	SetCaption (see page 274)	Introduces an interface for setting the caption on a client.
	SetCaption (see page 5899)	Sets the caption of a control if the client is linked to the action for this event.
	SetChecked (see page 274)	Introduces an interface for setting the checked state of a client.
	SetDropdownMenu (see page 5899)	This is SetDropdownMenu, a member of class TControlActionLink.
	SetEnabled (see page 275)	Introduces an interface for setting the enabled state of a client.
	SetEnabled (see page 5900)	Sets the enabled state of a control if the client is linked to the action for this event.
	SetEnableDropdown (see page 5900)	This is SetEnableDropdown, a member of class TControlActionLink.
	SetGroupIndex (see page 275)	Introduces an interface for setting the GroupIndex property of a client.
	SetHelpContext (see page 276)	Introduces an interface for setting the help context of a client.
	SetHelpContext (see page 5900)	Sets the HelpContext of a control if the client is linked to the action help properties.
	SetHelpKeyword (see page 276)	Introduces an interface for setting the help keyword of a client.
	SetHelpKeyword (see page 5901)	Sets the HelpKeyword of a control if the client is linked to the action help properties.
	SetHelpType (see page 277)	Introduces an interface for setting the HelpType property of a client.
	SetHelpType (see page 5901)	Sets the HelpType of a control if the client is linked to the action help properties.
	SetHint (see page 278)	Introduces an interface for setting the hint for a client.
	SetHint (see page 5901)	Sets the hint of a control if the client is linked to the action for this event.
	SetImageIndex (see page 2335)	Introduces an interface for setting the image index for a client.
	SetOnExecute (see page 2724)	This is SetOnExecute, a member of class TBasicActionLink.

	SetOnExecute (see page 5902)	Sets the execution event handler for a control if the client is linked to the action for this event.
	SetPopupMenu (see page 5902)	This is SetPopupMenu, a member of class TControlActionLink.
	SetShortCut (see page 279)	Introduces an interface for setting the shortcut for a client.
	SetVisible (see page 279)	Introduces an interface for setting the visible state of a client.
	SetVisible (see page 5902)	Sets the visible state of a control if the client is linked to the action for this event.

Public Events

	Name	Description
	OnChange (see page 2725)	Occurs when an OnChange event is triggered on the action.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2332)	Instantiates and initializes a TBasicActionLink object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2722)	Destroys an instance of an action link object.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 2723)	Invokes the action's Execute method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Update (see page 2724)	Invokes the action's Update method.

Public Properties

	Name	Description
	Action (see page 2725)	Indicates which action the link connects to its client.

17.1.2.2 TBitBtnActionLink.Create Constructor

Instantiates and initializes a TBasicActionLink object.

Pascal

```
constructor Create(AClient: TObject); override;
```

C++

```
virtual __fastcall TBitBtnActionLink(TObject * AClient);
```

Description

Applications do not need to call the constructor directly. Instances of TBasicActionLink are created automatically when you connect an action with a client in the action list editor. The AClient parameter is the client for the link. The link connects this client to its action.

Create ensures that the client is of the appropriate type for the particular type of link. Descendant classes can progressively specify more customized client types. For example, there are action link classes for controls, for windowed controls, and for button controls.

See Also

Action (see page 5797)

ActionLink (see page 5798)

17.1.2.3 TBitBtnActionLink Fields

The fields of the TBitBtnActionLink class are listed here.

Protected Fields

	Name	Description
	FClient (see page 2333)	This is FClient, a member of class TBitBtnActionLink.
	FImageIndex (see page 2333)	This is FImageIndex, a member of class TBitBtnActionLink.

17.1.2.3.1 TBitBtnActionLink.FClient Field

Pascal

```
FClient: TBitBtn;
```

C++

```
TBitBtn FClient;
```

Description

This is FClient, a member of class TBitBtnActionLink.

17.1.2.3.2 TBitBtnActionLink.FImageIndex Field

Pascal

```
FImageIndex: Integer;
```

C++

```
int FImageIndex;
```

Description

This is FImageIndex, a member of class TBitBtnActionLink.

17.1.2.4 TBitBtnActionLink Methods

The methods of the TBitBtnActionLink class are listed here.

Protected Methods

	Name	Description
	AssignClient (see page 2334)	Assigns a client control that is linked to the associated action.
	IsImageIndexLinked (see page 2334)	Indicates whether the image index for the action and the client are linked.
	SetImageIndex (see page 2335)	Introduces an interface for setting the image index for a client.

17.1.2.4.1 TBitBtnActionLink.AssignClient Method

Assigns a client control that is linked to the associated action.

Pascal

```
procedure AssignClient(AClient: TObject); override;
```

C++

```
virtual __fastcall AssignClient(TObject * AClient);
```

Description

TControlActionLink overrides AssignClient to ensure that the client for the action descends from TControl. Descendants of TControlActionLink can further limit the type of client associated with the link.

See Also

Action (see page 5797)

17.1.2.4.2 TBitBtnActionLink.IsImageIndexLinked Method

Indicates whether the image index for the action and the client are linked.

Pascal

```
function IsImageIndexLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsImageIndexLinked();
```

Description

TActionLink descendants call IsImageIndexLinked to determine whether they need to propagate changes to the ImageIndex property of the action so that they are reflected by the client object. When IsImageIndexLinked returns true, the action controls the ImageIndex property: Changes to the ImageIndex property of the action are reflected by changes to the corresponding property of the client object. When IsImageIndexLinked returns false, the client object does not reflect the ImageIndex property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink, IsImageIndexLinked merely checks that the Action property specifies a descendant of TCustomAction (that is, it is an action that has an ImageIndex property). Descendant classes override IsImageIndexLinked to determine whether the ImageIndex property is linked with the appropriate client property.

See Also

ImageIndex (see page 313)

ImageIndex (see page 16148)

SetImageIndex (see page 278)

Action (see page 2725)

17.1.2.4.3 TBitBtnActionLink.SetImageIndex Method

Introduces an interface for setting the image index for a client.

Pascal

```
procedure SetImageIndex(Value: Integer); override;
```

C++

```
virtual __fastcall SetImageIndex(int Value);
```

Description

The associated action calls SetImageIndex when its ImageIndex property changes so that the action link can propagate the new ImageIndex value to the client object.

Value specifies the new value of the ImageIndex property.

As implemented in TActionLink, SetImageIndex does nothing. Descendant classes override SetImageIndex to set the client property that corresponds to the action's ImageIndex property if the IsImageIndexLinked method returns true.

See Also

ImageIndex (see page 313)

ImageIndex (see page 16148)

Action (see page 2725)

IsImageIndexLinked (see page 272)

17.1.3 TSpeedButton Class

TSpeedButton is a button that is used to execute commands or set modes.

Class Hierarchy

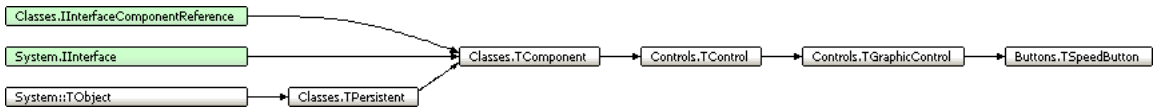

Pascal

```
TSpeedButton = class(TGraphicControl);
```

C++

```
class TSpeedButton : public TGraphicControl;
```

File

Buttons

Description

Use TSpeedButton to add a button to a group of buttons in a form. TSpeedButton introduces properties that can be used to set graphical images that represent the different button states (selected, unselected, disabled and so on). Use other properties to

specify multiple images or to rearrange the images and text on the button. TSpeedButton also introduces properties that allow speed buttons to work together as a group. Speed buttons are commonly grouped in panels to create specialized tool bars and tool palettes.

The recommended way to implement the response of the button when the user clicks on it is to assign an action from an action list as the value of the Action (see page 2350) property. By setting the Action (see page 2350) property, you make the button a client of the action, and the action handles updating the button's properties and responding when the user clicks the button.

If you are not using an action to respond when the user clicks the button, then you can specify the button's response by writing an OnClick (see page 2372) event handler.

Tip: To create a speed button with a client-aligned glyph that covers the button's edges, add a 1-pixel margin around the button's top and left edges and a 3-pixel margin around its bottom and right edges. Do this by manually editing the bitmap specified by the Glyph (see page 2359) property.

See Also

TBitBtn (see page 2290)

TButton (see page 20909)

TToolButton (see page 5107)

Buttons and Similar Controls

Speed Buttons

Adding a Cool Bar Component

Adding a Speed Button to a Panel

Adding a Toolbar Using the Toolbar Component

Adding a Tool Button

Adding Hidden Toolbars

Allowing Toggle Buttons

Allowing Toggled Tool Buttons

Assigning a Menu to a Tool Button

Assigning a Speed Button's Glyph

Assigning Images to Tool Buttons

Creating a Group of Speed Buttons

Creating Groups of Tool Buttons

Hiding and Showing Toolbars

Responding to Clicks

Setting the Initial Condition of a Speed Button

Setting Tool Button Appearance and Initial Conditions

VCL Overview

17.1.3.1 TSpeedButton Members

The following tables list the members exposed by TSpeedButton.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FState (see page 2345)	This is FState, a member of class TSpeedButton.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActionChange (see page 2345)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.

	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	CopyImage (see page 2346)	This is CopyImage, a member of class TSpeedButton.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
V	GetActionLinkClass (see page 2346)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetPalette (see page 2347)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2347)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 2348)	OnMouseDown event dispatcher.
V	MouseMove (see page 2348)	Respond to mouse moving over control area..

	MouseUp (see page 2349)	OnMouseUp event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Paint (see page 2349)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.

	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	MouseInControl (see page 2361)	This is MouseInControl, a member of class TSpeedButton.
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Click (see page 2346)	Generates an OnClick (see page 2372) event or triggers the associated action.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 2344)	Creates and initializes an instance of TSpeedButton (see page 2335).
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2344)	Destroys an instance of TSpeedButton (see page 2335).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.

	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.

V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnClick (see page 2372)	Occurs when the user clicks the control.
	OnDblClick (see page 2374)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnMouseActivate (see page 2374)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2375)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2377)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2378)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2378)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2380)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Published Properties

	Name	Description
	Action (see page 2350)	Designates the action associated with the control.
	Align (see page 2350)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AllowAllUp (see page 2354)	Specifies whether all speed buttons in the group that contains this speed button can be unselected at the same time.
	Anchors (see page 2355)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2355)	Specifies the bi-directional mode for the control.
	Caption (see page 2356)	Specifies the caption, or label, that appears on the speed button.
	Constraints (see page 2356)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Down (see page 2357)	Specifies whether the button is selected (down) or unselected (up).
	Enabled (see page 2357)	Controls whether the control responds to mouse, keyboard, and timer events.
	Flat (see page 2357)	Determines whether the button has a 3D border that provides a raised or lowered look.
	Font (see page 2358)	Controls the attributes of text written on or in the control.
	Glyph (see page 2359)	Specifies the bitmap that appears on the speed button.
	GroupIndex (see page 2360)	Allows speed buttons to work together as a group.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Layout (see page 2360)	Determines where the image or text appears on the speed button.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margin (see page 2361)	Specifies the number of pixels between the edge of the button and the image or caption drawn on its surface.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NumGlyphs (see page 2361)	Specifies the number of images included in the Glyph (see page 2359) property.
	ParentBiDiMode (see page 2362)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 2362)	Determines where a control looks for its font information.
	ParentShowHint (see page 2363)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2363)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2368)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

	Spacing (see page 2369)	Determines where the image and text appear on a speed button.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Transparent (see page 2370)	Specifies whether the background of the button is transparent.
	Visible (see page 2370)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

17.1.3.2 TSpeedButton.Create Constructor

Creates and initializes an instance of TSpeedButton (see page 2335).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSpeedButton(TComponent * AOwner);
```

Description

Call Create to create a speed button at runtime. Speed buttons placed on forms at design time are created automatically.

AOwner is the component, typically a form, that is responsible for freeing the speed button.

See Also

Owner (see page 2807)

17.1.3.3 TSpeedButton.Destroy Destructor

Destroys an instance of TSpeedButton (see page 2335).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSpeedButton();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the speed button is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

17.1.3.4 TSpeedButton Fields

The fields of the TSpeedButton class are listed here.

Protected Fields

	Name	Description
	FState (see page 2345)	This is FState, a member of class TSpeedButton.

17.1.3.4.1 TSpeedButton.FState Field

Pascal

```
FState: TButtonState;
```

C++

```
TButtonState FState;
```

Description

This is FState, a member of class TSpeedButton.

17.1.3.5 TSpeedButton Methods

The methods of the TSpeedButton class are listed here.

Protected Methods

	Name	Description
	ActionChange (see page 2345)	Responds when the action associated with the control changes.
	CopyImage (see page 2346)	This is CopyImage, a member of class TSpeedButton.
	GetActionLinkClass (see page 2346)	Returns the appropriate action link class that is associated with the control class.
	GetPalette (see page 2347)	Returns the handle to a palette for use by the control.
	Loaded (see page 2347)	Initializes the control after it is loaded from a stream.
	MouseDown (see page 2348)	OnMouseDown event dispatcher.
	MouseMove (see page 2348)	Respond to mouse moving over control area..
	MouseUp (see page 2349)	OnMouseUp event dispatcher.
	Paint (see page 2349)	Render control surface.

Public Methods

	Name	Description
	Click (see page 2346)	Generates an OnClick (see page 2372) event or triggers the associated action.

17.1.3.5.1 TSpeedButton.ActionChange Method

Responds when the action associated with the control changes.

Pascal

```
procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); override;
```

C++

```
virtual __fastcall ActionChange(TObject * Sender, Boolean CheckDefaults);
```

Description

ActionChange is called automatically when the action object specified by the Action property, or any of its properties, changes. By default, the control's action link's OnChange event handler calls ActionChange.

ActionChange updates the control properties that correspond to properties of the associated action. These properties include the Caption, Enabled, Hint, and Visible properties and the OnClick event handler.

The Sender parameter is the action object that changed. The CheckDefaults parameter indicates whether the control's properties should all be updated or only those that have not been explicitly changed from their default values. When CheckDefaults is true, only those properties and event handlers that have not been changed from the default values are updated.

See Also

Action (see page 5797)

OnChange (see page 2725)

17.1.3.5.2 TSpeedButton.Click Method

Generates an OnClick (see page 2372) event or triggers the associated action.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Call Click to generate an OnClick (see page 2372) event. If the speed button has an OnClick (see page 2372) event handler, or if it is linked to an action, Click allows an application to simulate a click on the button.

See Also

OnClick (see page 5851)

17.1.3.5.3 TSpeedButton.CopyImage Method

Pascal

```
procedure CopyImage(ImageList: TCustomImageList; Index: Integer);
```

C++

```
__fastcall CopyImage(TCustomImageList ImageList, int Index);
```

Description

This is CopyImage, a member of class TSpeedButton.

17.1.3.5.4 TSpeedButton.GetActionLinkClass Method

Returns the appropriate action link class that is associated with the control class.

Pascal

```
function GetActionLinkClass: TControlActionLinkClass; override;
```

C++

```
virtual __fastcall TControlActionLinkClass GetActionLinkClass();
```

Description

GetActionLinkClass returns the appropriate class type for the action link used with a control. This class is used internally to create an action link object for the control when it has an associated action. The action link links the action to the control client. Each Action link class is designed to link specific properties and event handlers of the action to its client, based on the types of properties the client supports.

As implemented in TControl, GetActionLinkClass returns the TControlActionLink class type, which associates the ShowHint, Caption, Enabled and Visible properties and the OnClick event handler. Descendants of TControl override this method to specify a descendant of TControlActionLink that handles their configuration of properties and events.

See Also

TControlActionLink (see page 5892)

ActionLink (see page 5798)

Action (see page 5797)

17.1.3.5.5 TSpeedButton.GetPalette Method

Returns the handle to a palette for use by the control.

Pascal

```
function GetPalette: HPALETTE; override;
```

C++

```
virtual __fastcall HPALETTE GetPalette();
```

Description

As implemented in TControl, GetPalette returns a null handle, indicating that the control does not have a palette. Override GetPalette to implement derived objects that use a palette. Create a palette handle for the control by a call to the Windows API function CreatePalette. GetPalette should return the palette handle for the control.

Specifying the palette for a control tells the application that the control's palette needs to be realized and designates the palette to use for realization.

GetPalette is called only if the run-time video mode requires color palette support, for example, in 256-color mode.

See Also

PaletteChanged (see page 5775)

17.1.3.5.6 TSpeedButton.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded (see page 2782)

Action (see page 5797)

ActionChange (see page 5737)

17.1.3.5.7 TSpeedButton.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove (see page 5773)

MouseUp (see page 5774)

OnMouseDown (see page 5862)

17.1.3.5.8 TSpeedButton.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown (see page 5773)

MouseUp (see page 5774)

OnMouseMove (see page 5865)

17.1.3.5.9 TSpeedButton.MouseUp Method

OnMouseUp event dispatcher.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user releases a previously pressed mouse button while the cursor's hotspot is over the control.

A control calls MouseUp in response to any of the Windows mouse-up messages (WM_LBUTTONUP, WM_MBUTTONUP, WM_RBUTTONUP), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

See Also

MouseDown (see page 5773)

MouseMove (see page 5773)

OnMouseUp (see page 5867)

17.1.3.5.10 TSpeedButton.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas (see page 6053)

17.1.3.6 TSpeedButton Properties

The properties of the TSpeedButton class are listed here.

Protected Properties

	Name	Description
R	MouseInControl (see page 2361)	This is MouseInControl, a member of class TSpeedButton.

Published Properties

	Name	Description
	Action (see page 2350)	Designates the action associated with the control.
	Align (see page 2350)	Determines how the control aligns within its container (parent control).
	AllowAllUp (see page 2354)	Specifies whether all speed buttons in the group that contains this speed button can be unselected at the same time.
	Anchors (see page 2355)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2355)	Specifies the bi-directional mode for the control.
	Caption (see page 2356)	Specifies the caption, or label, that appears on the speed button.
	Constraints (see page 2356)	Specifies the size constraints for the control.
	Down (see page 2357)	Specifies whether the button is selected (down) or unselected (up).
	Enabled (see page 2357)	Controls whether the control responds to mouse, keyboard, and timer events.
	Flat (see page 2357)	Determines whether the button has a 3D border that provides a raised or lowered look.
	Font (see page 2358)	Controls the attributes of text written on or in the control.
	Glyph (see page 2359)	Specifies the bitmap that appears on the speed button.
	GroupIndex (see page 2360)	Allows speed buttons to work together as a group.
	Layout (see page 2360)	Determines where the image or text appears on the speed button.
	Margin (see page 2361)	Specifies the number of pixels between the edge of the button and the image or caption drawn on its surface.
	NumGlyphs (see page 2361)	Specifies the number of images included in the Glyph (see page 2359) property.
	ParentBiDiMode (see page 2362)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 2362)	Determines where a control looks for its font information.
	ParentShowHint (see page 2363)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2363)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2368)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Spacing (see page 2369)	Determines where the image and text appear on a speed button.
	Transparent (see page 2370)	Specifies whether the background of the button is transparent.
	Visible (see page 2370)	Determines whether the component appears onscreen.

17.1.3.6.1 TSpeedButton.Action Property

Designates the action associated with the control.

Pascal

```
property Action: TBasicAction;
```

C++

```
__property TBasicAction Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

17.1.3.6.2 TSpeedButton.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;
```

```
  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
```

```
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
```

```
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

17.1.3.6.3 TSpeedButton.AllowAllUp Property

Specifies whether all speed buttons in the group that contains this speed button can be unselected at the same time.

Pascal

```
property AllowAllUp: Boolean;
```

C++

```
__property Boolean AllowAllUp;
```

Description

Use AllowAllUp with a group of buttons that belong to the same group. Speed buttons belong to the same group if they have the same value on their GroupIndex (see page 2360) property. Buttons (see page 2289) in the same group behave in a mutually exclusive fashion: when one button in the group is selected (down), all other buttons in the group become unselected (up).

If AllowAllUp is true, all of the speed buttons in a group can be unselected at the same time. Clicking the single selected button in the group will deselect that button without selecting another. If AllowAllUp is false, exactly one of the buttons in the group must be selected at any time. Clicking a down button won't return the button to its up state.

Changing the value of the AllowAllUp property for one speed button in a group changes the AllowAllUp value for all buttons in the group.

Note: If GroupIndex (see page 2360) is zero, AllowAllUp has no effect.

See Also

Down (see page 2357)

GroupIndex (see page 2360)

17.1.3.6.4 TSpeedButton.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

17.1.3.6.5 TSpeedButton.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

17.1.3.6.6 TSpeedButton.Caption Property

Specifies the caption, or label, that appears on the speed button.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the caption that appears on the speed button. If the speed button is the client of an action, the Caption property can be set by that action.

See Also

Glyph (see page 2359)

Layout (see page 2360)

17.1.3.6.7 TSpeedButton.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

17.1.3.6.8 TSpeedButton.Down Property

Specifies whether the button is selected (down) or unselected (up).

Pascal

```
property Down: Boolean;
```

C++

```
__property Boolean Down;
```

Description

Read Down to determine whether a speed button is selected. The Down property only applies if the GroupIndex (see page 2360) property of the button is nonzero.

When GroupIndex (see page 2360) is greater than 0, set Down to true to select a button. When the user clicks on a button in the unselected (up) state, the button is selected and Down is set to true. When the user clicks on a button in the selected (down) state, if Caption (see page 2356) is true, the button becomes unselected and Down is set to false.

At design time, specify which button in a group is the initially selected button by setting the Down property of the selected button to true.

Note: When GroupIndex (see page 2360) is 0, buttons do not remain in the selected state when clicked.

See Also

Caption (see page 5806)

GroupIndex (see page 2360)

17.1.3.6.9 TSpeedButton.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

17.1.3.6.10 TSpeedButton.Flat Property

Determines whether the button has a 3D border that provides a raised or lowered look.

Pascal

```
property Flat: Boolean;
```

C++

```
__property Boolean Flat;
```

Description

Set Flat to true to remove the raised border when the button is unselected and the lowered border when the button is clicked or selected. When Flat is true, use separate bitmaps for the different button states to provide visual feedback to the user about the button state.

See Also

Glyph (see page 2359)

NumGlyphs (see page 2361)

17.1.3.6.11 TSpeedButton.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

17.1.3.6.12 TSpeedButton.Glyph Property

Specifies the bitmap that appears on the speed button.

Pascal

```
property Glyph: TBitmap;
```

C++

```
__property TBitmap Glyph;
```

Description

Set Glyph to a bitmap object that contains the image that should appear on the face of the button. Bring up the Open dialog box from the Object Inspector to choose a bitmap file (with a .BMP extension), or specify a bitmap file at runtime.

Glyph can provide up to four images within a single bitmap. All images must be the same size and next to each other in a horizontal row. TSpeedButton (see page 2335) displays one of these images depending on the state of the button.

Image position	Button state	Description
First	Up	This image appears when the button is unselected. If no other images exist in the bitmap, this image is used for all states.
Second	Disabled	This image usually appears dimmed to indicate that the button can't be selected.
Third	Clicked	This image appears when the button is clicked. If GroupIndex (see page 2360) is 0, the Up image reappears when the user releases the mouse button.
Fourth	Down (see page 2357)	This image appears when the button stays down indicating that it remains selected.

If only one image is present, TSpeedButton (see page 2335) attempts to represent the other states by altering the image slightly for each state, although the Down (see page 2357) state is always the same as the Up state.

If the bitmap contains multiple images, specify the number of images in the bitmap with the NumGlyphs (see page 2361) property.

Note: The lower left pixel of the bitmap is reserved for the "transparent" color. Any pixel in the bitmap that matches the lower left pixel will be transparent.

See Also

Down (see page 2357)

GroupIndex (see page 2360)

Layout (see page 2360)

NumGlyphs (see page 2361)

17.1.3.6.13 TSpeedButton.GroupIndex Property

Allows speed buttons to work together as a group.

Pascal

```
property GroupIndex: Integer;
```

C++

```
__property int GroupIndex;
```

Description

Set GroupIndex to determine how the button behaves when clicked.

When GroupIndex is 0, the button behaves independently of all other buttons on the form. When the user clicks such a speed button, the button appears pressed (in its clicked state) and then returns to its normal up state when the user releases the mouse button.

When GroupIndex is greater than 0, the button remains selected (in its down state) when clicked by the user. When the user clicks a selected button, it returns to the up state, unless AllowAllUp (see page 2354) is false. Setting the GroupIndex property of a single speed button to a value greater than 0 causes the button to behave as a two-state button when AllowAllUp (see page 2354) is true.

Speed buttons with the same GroupIndex property value (other than 0), work together as a group. When the user clicks one of these buttons, it remains selected until the user clicks another speed button belonging to the same group. Speed buttons used in this way can present mutually exclusive choices to the user.

See Also

Caption (see page 5806)

Down (see page 2357)

Layout (see page 2360)

17.1.3.6.14 TSpeedButton.Layout Property

Determines where the image or text appears on the speed button.

Pascal

```
property Layout: TButtonLayout;
```

C++

```
__property TButtonLayout Layout;
```

Description

Set Layout to determine the position of the bitmap specified by the Glyph (see page 2359) property or the text specified by the Caption (see page 2356) property.

See Also

Caption (see page 5806)

Glyph (see page 2359)

Margin (see page 2361)

Spacing (see page 2369)

17.1.3.6.15 TSpeedButton.Margin Property

Specifies the number of pixels between the edge of the button and the image or caption drawn on its surface.

Pascal

```
property Margin: Integer;
```

C++

```
__property int Margin;
```

Description

Use Margin to specify the indentation of the image specified by the Glyph (see page 2359) property or the text specified by the Caption (see page 2356) property. The edges that Margin separates depends on the Layout (see page 2360) property. If Layout (see page 2360) is blGlyphLeft, the margin appears between the left edge of the image or caption and the left edge of the button. If Layout (see page 2360) is blGlyphRight, the margin separates the right edges. If Layout (see page 2360) is blGlyphTop, the margin separates the top edges, and if Layout (see page 2360) is blGlyphBottom, the margin separates the bottom edges.

If Margin is –1, the image or text is centered on the button.

See Also

Caption (see page 5806)

Glyph (see page 2359)

Spacing (see page 2369)

17.1.3.6.16 TSpeedButton.MouseInControl Property

Pascal

```
property MouseInControl: Boolean;
```

C++

```
__property Boolean MouseInControl;
```

Description

This is MouseInControl, a member of class TSpeedButton.

17.1.3.6.17 TSpeedButton.NumGlyphs Property

Specifies the number of images included in the Glyph (see page 2359) property.

Pascal

```
property NumGlyphs: TNumGlyphs;
```

C++

```
__property TNumGlyphs NumGlyphs;
```

Description

Set NumGlyphs to the number of images provided by the bitmap assigned to the Glyph (see page 2359) property. All images must be the same size and next to each other in a row. The Glyph (see page 2359) property can provide up to four images.

See Also

Glyph (see page 2359)

17.1.3.6.18 TSpeedButton.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

17.1.3.6.19 TSpeedButton.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

17.1.3.6.20 TSpeedButton.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

17.1.3.6.21 TSpeedButton.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
```

```
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
```

```
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
```

```
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

17.1.3.6.22 TSpeedButton.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
```

```
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

17.1.3.6.23 TSpeedButton.Spacing Property

Determines where the image and text appear on a speed button.

Pascal

```
property Spacing: Integer;
```

C++

```
__property int Spacing;
```

Description

Set Spacing to the number of pixels that should appear between the image specified in the Glyph (see page 2359) property and the text specified in the Caption (see page 2356) property.

If Spacing is a positive number, its value is the number of pixels between the image and text. If Spacing is 0, the image and text appear flush with each other. If Spacing is -1, the text appears centered between the image and the button edge.

See Also

Caption (see page 5806)

Glyph (see page 2359)

Margin (see page 2361)

Layout (see page 2360)

17.1.3.6.24 TSpeedButton.Transparent Property

Specifies whether the background of the button is transparent.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Use Transparent to specify whether the background of the button is transparent.

See Also

Glyph (see page 2359)

NumGlyphs (see page 2361)

17.1.3.6.25 TSpeedButton.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
```

```
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

17.1.3.7 TSpeedButton Events

The events of the TSpeedButton class are listed here.

Published Events

	Name	Description
	OnClick (see page 2372)	Occurs when the user clicks the control.
	OnDblClick (see page 2374)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnMouseActivate (see page 2374)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2375)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2377)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2378)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2378)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2380)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

17.1.3.7.1 TSpeedButton.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
```

```
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
```

```
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

17.1.3.7.2 TSpeedButton.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

17.1.3.7.3 TSpeedButton.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no

control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

17.1.3.7.4 TSpeedButton.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
```

```
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
```

```
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

17.1.3.7.5 TSpeedButton.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

17.1.3.7.6 TSpeedButton.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

17.1.3.7.7 TSpeedButton.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}
```

```
// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
```

```
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

17.1.3.7.8 TSpeedButton.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
```

```
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

17.1.4 TSpeedButtonActionLink Class

TSpeedButtonActionLink is an action link designed specifically for button control clients.

Class Hierarchy

Pascal

```
TSpeedButtonActionLink = class(TControlActionLink);
```

C++

```
class TSpeedButtonActionLink : public TControlActionLink;
```

File

Buttons

Description

TSpeedButtonActionLink is designed to specifically link actions to speed buttton control clients. TSpeedButtonActionLink overrides methods to determine whether the action and client are linked, and to ensure that action events are appropriate for properties of speed button control clients.

TSpeedButtonActionLink inherits, from its ancestors, support for linking actions with the properties of components, generic controls, and windowed controls. TSpeedButtonActionLink adds support for linking the checked property to the appropriate property of a speed button control.

17.1.4.1 TSpeedButtonActionLink Members

The following tables list the members exposed by TSpeedButtonActionLink.

Protected Fields

	Name	Description
	FAction (see page 2722)	This is FAction, a member of class TBasicActionLink.
	FClient (see page 2386)	This is FClient, a member of class TSpeedButtonActionLink.
	FImageIndex (see page 2386)	This is FImageIndex, a member of class TSpeedButtonActionLink.

Protected Methods

	Name	Description
	AssignClient (see page 2386)	Assigns a client control that is linked to the associated action.
	Change (see page 2723)	This is Change, a member of class TBasicActionLink.
	DoShowHint (see page 5896)	Indicates whether the hint is shown, and modified.
	IsCaptionLinked (see page 268)	Indicates whether the Caption properties for the action and client are linked.
	IsCaptionLinked (see page 5896)	Indicates whether the caption for the action and the client are linked.
	IsCheckedLinked (see page 2387)	Indicates whether the Checked properties for the action and client are linked.
	IsDropdownMenuLinked (see page 5896)	This is IsDropdownMenuLinked, a member of class TControlActionLink.
	IsEnabledLinked (see page 269)	Indicates whether the Enabled properties for the action and client are linked.
	IsEnabledLinked (see page 5897)	Indicates whether the enabled state for the action and the client are linked.
	IsEnableDropdownLinked (see page 5897)	This is IsEnableDropdownLinked, a member of class TControlActionLink.
	IsGroupIndexLinked (see page 2387)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsHelpContextLinked (see page 270)	Indicates whether the HelpContext properties for the action and client controls are linked.
	IsHelpLinked (see page 271)	Indicates whether the help properties for the action and client are linked.
	IsHelpLinked (see page 5897)	Indicates whether the help properties for the action and the client are linked.
	IsHintLinked (see page 271)	Indicates whether the Hint properties for the action and client are linked.
	IsHintLinked (see page 5898)	Indicates whether the Hint property for the action and the client are linked.
	IsImageIndexLinked (see page 2388)	Indicates whether the image index for the action and the client are linked.
	IsOnExecuteLinked (see page 2723)	This is IsOnExecuteLinked, a member of class TBasicActionLink.
	IsOnExecuteLinked (see page 5898)	Indicates whether the OnExecute event handler for the action and the client are linked.
	IsPopupMenuLinked (see page 5898)	This is IsPopupMenuLinked, a member of class TControlActionLink.
	IsShortCutLinked (see page 272)	Indicates whether the ShortCut properties for the action and client are linked.

	IsVisibleLinked (see page 273)	Indicates whether the Visible properties for the action and client are linked.
	IsVisibleLinked (see page 5899)	Indicates whether the visible state for the action and the client are linked.
	SetAction (see page 2724)	This is SetAction, a member of class TBasicActionLink.
	SetAutoCheck (see page 274)	Introduces an interface for setting the AutoCheck property of a client.
	SetCaption (see page 274)	Introduces an interface for setting the caption on a client.
	SetCaption (see page 5899)	Sets the caption of a control if the client is linked to the action for this event.
	SetChecked (see page 2388)	Introduces an interface for setting the checked state of a client.
	SetDropdownMenu (see page 5899)	This is SetDropdownMenu, a member of class TControlActionLink.
	SetEnabled (see page 275)	Introduces an interface for setting the enabled state of a client.
	SetEnabled (see page 5900)	Sets the enabled state of a control if the client is linked to the action for this event.
	SetEnableDropdown (see page 5900)	This is SetEnableDropdown, a member of class TControlActionLink.
	SetGroupIndex (see page 2389)	Introduces an interface for setting the GroupIndex property of a client.
	SetHelpContext (see page 276)	Introduces an interface for setting the help context of a client.
	SetHelpContext (see page 5900)	Sets the HelpContext of a control if the client is linked to the action help properties.
	SetHelpKeyword (see page 276)	Introduces an interface for setting the help keyword of a client.
	SetHelpKeyword (see page 5901)	Sets the HelpKeyword of a control if the client is linked to the action help properties.
	SetHelpType (see page 277)	Introduces an interface for setting the HelpType property of a client.
	SetHelpType (see page 5901)	Sets the HelpType of a control if the client is linked to the action help properties.
	SetHint (see page 278)	Introduces an interface for setting the hint for a client.
	SetHint (see page 5901)	Sets the hint of a control if the client is linked to the action for this event.
	SetImageIndex (see page 2389)	Introduces an interface for setting the image index for a client.
	SetOnExecute (see page 2724)	This is SetOnExecute, a member of class TBasicActionLink.
	SetOnExecute (see page 5902)	Sets the execution event handler for a control if the client is linked to the action for this event.
	SetPopupMenu (see page 5902)	This is SetPopupMenu, a member of class TControlActionLink.
	SetShortCut (see page 279)	Introduces an interface for setting the shortcut for a client.
	SetVisible (see page 279)	Introduces an interface for setting the visible state of a client.
	SetVisible (see page 5902)	Sets the visible state of a control if the client is linked to the action for this event.

Public Events

	Name	Description
	OnChange (see page 2725)	Occurs when an OnChange event is triggered on the action.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2385)	Instantiates and initializes a TBasicActionLink object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2722)	Destroys an instance of an action link object.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 2723)	Invokes the action's Execute method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Update (see page 2724)	Invokes the action's Update method.

Public Properties

	Name	Description
	Action (see page 2725)	Indicates which action the link connects to its client.

17.1.4.2 TSpeedButtonActionLink.Create Constructor

Instantiates and initializes a TBasicActionLink object.

Pascal

```
constructor Create(AClient: TObject); override;
```

C++

```
virtual __fastcall TSpeedButtonActionLink(TObject * AClient);
```

Description

Applications do not need to call the constructor directly. Instances of TBasicActionLink are created automatically when you connect an action with a client in the action list editor. The AClient parameter is the client for the link. The link connects this client to its action.

Create ensures that the client is of the appropriate type for the particular type of link. Descendant classes can progressively specify more customized client types. For example, there are action link classes for controls, for windowed controls, and for button controls.

See Also

Action (see page 5797)

ActionLink (see page 5798)

17.1.4.3 TSpeedButtonActionLink Fields

The fields of the TSpeedButtonActionLink class are listed here.

Protected Fields

	Name	Description
	FClient (see page 2386)	This is FClient, a member of class TSpeedButtonActionLink.
	FImageIndex (see page 2386)	This is FImageIndex, a member of class TSpeedButtonActionLink.

17.1.4.3.1 TSpeedButtonActionLink.FClient Field

Pascal

```
FClient: TSpeedButton;
```

C++

```
TSpeedButton FClient;
```

Description

This is FClient, a member of class TSpeedButtonActionLink.

17.1.4.3.2 TSpeedButtonActionLink.FImageIndex Field

Pascal

```
FImageIndex: Integer;
```

C++

```
int FImageIndex;
```

Description

This is FImageIndex, a member of class TSpeedButtonActionLink.

17.1.4.4 TSpeedButtonActionLink Methods

The methods of the TSpeedButtonActionLink class are listed here.

Protected Methods

	Name	Description
	AssignClient (see page 2386)	Assigns a client control that is linked to the associated action.
	IsCheckedLinked (see page 2387)	Indicates whether the Checked properties for the action and client are linked.
	IsGroupIndexLinked (see page 2387)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsImageIndexLinked (see page 2388)	Indicates whether the image index for the action and the client are linked.
	SetChecked (see page 2388)	Introduces an interface for setting the checked state of a client.
	SetGroupIndex (see page 2389)	Introduces an interface for setting the GroupIndex property of a client.
	SetImageIndex (see page 2389)	Introduces an interface for setting the image index for a client.

17.1.4.4.1 TSpeedButtonActionLink.AssignClient Method

Assigns a client control that is linked to the associated action.

Pascal

```
procedure AssignClient(AClient: TObject); override;
```

C++

```
virtual __fastcall AssignClient(TObject * AClient);
```

Description

TControlActionLink overrides AssignClient to ensure that the client for the action descends from TControl. Descendants of TControlActionLink can further limit the type of client associated with the link.

See Also

Action (see page 5797)

17.1.4.4.2 TSpeedButtonActionLink.IsCheckedLinked Method

Indicates whether the Checked properties for the action and client are linked.

Pascal

```
function IsCheckedLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCheckedLinked();
```

Description

TActionLink descendants call IsCheckedLinked to determine whether they need to propagate changes to the Checked property of the action so that they are reflected by the client object. When IsCheckedLinked returns true, the action controls the Checked property: Changes to the Checked property of the action are reflected by changes to the corresponding property of the client object. When IsCheckedLinked returns false, the client object does not reflect the Checked property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink, IsCheckedLinked merely checks that the Action property specifies a descendant of TCustomAction (that is, it is an action that has a Checked property). Descendant classes override IsCheckedLinked to determine whether the Checked property is linked with the appropriate client property.

See Also

Checked (see page 308)

SetChecked (see page 274)

Action (see page 2725)

17.1.4.4.3 TSpeedButtonActionLink.IsGroupIndexLinked Method

Indicates whether the GroupIndex property for the action controls client behavior.

Pascal

```
function IsGroupIndexLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsGroupIndexLinked();
```

Description

TActionLink descendants can call IsGroupIndexLinked to determine whether they need to propagate changes to the GroupIndex property of the action so that they are reflected by the client object. When IsGroupIndexLinked returns true, the action controls the GroupIndex property: Changes to the GroupIndex property of the action are reflected in the client object. When IsGroupIndexLinked returns false, the client object is not influenced by the GroupIndex property of its action.

As implemented in TActionLink, IsGroupIndexLinked merely checks that the Action property specifies a descendant of TCustomAction, and if so, returns true.

See Also

GroupIndex (see page 311)

SetGroupIndex (see page 275)

Action (see page 2725)

17.1.4.4.4 TSpeedButtonActionLink.IsImageIndexLinked Method

Indicates whether the image index for the action and the client are linked.

Pascal

```
function IsImageIndexLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsImageIndexLinked();
```

Description

TActionLink descendants call IsImageIndexLinked to determine whether they need to propagate changes to the ImageIndex property of the action so that they are reflected by the client object. When IsImageIndexLinked returns true, the action controls the ImageIndex property: Changes to the ImageIndex property of the action are reflected by changes to the corresponding property of the client object. When IsImageIndexLinked returns false, the client object does not reflect the ImageIndex property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink, IsImageIndexLinked merely checks that the Action property specifies a descendant of TCustomAction (that is, it is an action that has an ImageIndex property). Descendant classes override IsImageIndexLinked to determine whether the ImageIndex property is linked with the appropriate client property.

See Also

ImageIndex (see page 313)

ImageIndex (see page 16148)

SetImageIndex (see page 278)

Action (see page 2725)

17.1.4.4.5 TSpeedButtonActionLink.SetChecked Method

Introduces an interface for setting the checked state of a client.

Pascal

```
procedure SetChecked(Value: Boolean); override;
```

C++

```
virtual __fastcall SetChecked(Boolean Value);
```

Description

The associated action calls SetChecked when its Checked property changes so that the action link can propagate the new Checked value to the client object.

Value specifies the new value of the Checked property.

As implemented in TActionLink, SetChecked does nothing. Descendant classes override SetChecked to set the client property that corresponds to the action's Checked property if the IsCheckedLinked method returns true.

See Also

Checked (see page 308)

Action (see page 2725)

IsCheckedLinked (see page 269)

17.1.4.4.6 TSpeedButtonActionLink.SetGroupIndex Method

Introduces an interface for setting the GroupIndex property of a client.

Pascal

```
procedure SetGroupIndex(Value: Integer); override;
```

C++

```
virtual __fastcall SetGroupIndex(int Value);
```

Description

The associated action calls SetGroupIndex when its GroupIndex property changes so that the action link can propagate the new GroupIndex value to the client object.

Value specifies the new value of the GroupIndex property.

As implemented in TActionLink, SetGroupIndex does nothing. It is not strictly necessary for descendant classes to override SetGroupIndex to set the client property that corresponds to the action's GroupIndex property, because the action itself implements the GroupIndex feature. That is, when an action's Checked property is set to true, it unchecks other actions in the same action list if they have the same GroupIndex value.

See Also

GroupIndex (see page 311)

GroupIndex (see page 16146)

Action (see page 2725)

GroupIndex (see page 2360)

Checked (see page 308)

IsGroupIndexLinked (see page 270)

17.1.4.4.7 TSpeedButtonActionLink.SetImageIndex Method

Introduces an interface for setting the image index for a client.

Pascal

```
procedure SetImageIndex(Value: Integer); override;
```

C++

```
virtual __fastcall SetImageIndex(int Value);
```

Description

The associated action calls SetImageIndex when its ImageIndex property changes so that the action link can propagate the new ImageIndex value to the client object.

Value specifies the new value of the ImageIndex property.

As implemented in TActionLink, SetImageIndex does nothing. Descendant classes override SetImageIndex to set the client property that corresponds to the action's ImageIndex property if the IsImageIndexLinked method returns true.

See Also

ImageIndex (see page 313)

ImageIndex (see page 16148)

Action (see page 2725)

IsImageIndexLinked (see page 272)

17.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DrawButtonFace (see page 2391)	Draws a push button.

17.2.1 Buttons.DrawButtonFace Function

Draws a push button.

Pascal

```
function DrawButtonFace(Canvas: TCanvas; const Client: TRect; BevelWidth: Integer; Style:
TButtonStyle; IsRounded: Boolean; IsDown: Boolean; IsFocused: Boolean): TRect;
```

C++

```
TRect DrawButtonFace(TCanvas Canvas, const TRect Client, int BevelWidth, TButtonStyle Style,
Boolean IsRounded, Boolean IsDown, Boolean IsFocused);
```

File

Buttons

Description

Use DrawButtonFace to draw a standard push button. DrawButtonFace draws the borders and background of the button. DrawButtonFace can be used by button controls that do not descend from TButtonControl, which must implement all their own rendering.

Canvas is the drawing surface on which to render the button.

Client indicates the boundaries of the button's client area on Canvas.

BevelWidth is the width of the beveled edge on the outside of the button. This parameter is ignored.

IsDown indicates whether the button is depressed.

IsFocused indicates whether the button has keyboard focus.

Flat indicates whether the button has a beveled border.

FillColor is the color to use for the surface of the button.

FillStyle is the style to use for the brush that paints the surface of the button.

DrawButtonFace returns the dimensions of the remaining usable area on the surface of the button after the edges and bevels have been drawn.

See Also

TButtonControl

17.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TBitBtnKind (see page 2392)	TBitBtnKind identifies the appearance and use of a TBitBtn (see page 2290) control.
	TButtonLayout (see page 2393)	TButtonLayout defines values for the Layout property of buttons.
	TButtonState (see page 2394)	TButtonState describes the current state of a button.
	TButtonStyle (see page 2395)	TButtonStyle specifies the basic look of a TBitBtn (see page 2290) control.

17.3.1 Buttons.TBitBtnKind Enumeration

TBitBtnKind identifies the appearance and use of a TBitBtn (see page 2290) control.

Pascal

```
TBitBtnKind = (
  bkCustom,
  bkOK,
  bkCancel,
  bkHelp,
  bkYes,
  bkNo,
  bkClose,
  bkAbort,
  bkRetry,
  bkIgnore,
  bkAll
);
```

C++

```
enum TBitBtnKind {
  bkCustom,
  bkOK,
  bkCancel,
  bkHelp,
  bkYes,
  bkNo,
  bkClose,
  bkAbort,
  bkRetry,
  bkIgnore,
  bkAll
};
```

File

Buttons

Description

TBitBtnKind specifies how a TBitBtn (see page 2290) control is rendered and how it behaves when clicked. The following table lists the possible values:

Value	Meaning
bkCustom	The button uses the Glyph property to determine the image that appears. There is no associated ModalResult value. Instead, you must explicitly select a ModalResult for the button or supply the code to respond using an action or an OnClick event.
bkOK	The text "OK" appears on the button face (possibly accompanied by a green check mark). The button becomes the default button (the Default property is automatically set to true). When the user chooses the button, the dialog box closes. The resulting ModalResult value of the button is mrOk.
bkCancel	The text "Cancel" appears on the button face (possibly accompanied by a red X). The button becomes the Cancel button (the Cancel property is automatically set to true). When the user chooses the button, the dialog box closes. The resulting ModalResult value of the bitmap button is mrCancel.
bkYes	The text "Yes" appears on the button face (possibly accompanied by a green check mark). The button becomes the default button (the Default property is automatically set to true). When the user chooses the button, any changes the user made in the dialog box are accepted and the dialog box closes. The resulting ModalResult value of the bitmap button is mrYes.
bkNo	The text "No" appears on the button face (possibly accompanied by a red no symbol). The button becomes the Cancel button (the Cancel property is automatically set to true). When the user chooses the button, any changes the user made in the dialog box are canceled and the dialog box closes. The resulting ModalResult value of the bitmap button is mrNo.
bkHelp	The text "Help" appears on the button face (possibly accompanied by a cyan question mark). When the user chooses the button, the help topic associated with the button is automatically displayed.
bkClose	The text "Close" appears on the button face (possibly accompanied by a door with a green exit sign over it). When the user chooses the button, the form closes. The Default property of the button is true.
bkAbort	The text "Abort" appears on the button face (possibly accompanied by a red X).
bkRetry	The text "Retry" appears on the button face (possibly accompanied by a green circular arrow).
bkIgnore	The text "Ignore" appears on the button face (possibly accompanied by a green man walking away). Use it to allow the user to continue after an error has occurred.
bkAll	The text "Yes to All" appears on the button face (possibly accompanied by a double green check mark).

See Also

CustomAlignInsertBefore

CustomAlignPosition

17.3.2 Buttons.TButtonLayout Enumeration

TButtonLayout defines values for the Layout property of buttons.

Pascal

```
TButtonLayout = (
  blGlyphLeft,
  blGlyphRight,
  blGlyphTop,
  blGlyphBottom
);
```

C++

```
enum TButtonLayout {
  blGlyphLeft,
  blGlyphRight,
  blGlyphTop,
```

```
  blGlyphBottom
};
```

File

Buttons

Description

TButtonLayout defines the location of an image on a button. The following table lists the values for TButtonLayout:

Value	Meaning
blGlyphLeft	The image appears near the left side of the button.
blGlyphRight	The image appears near the right side of the button.
blGlyphTop	The image appears near the top of the button.
blGlyphBottom	The image appears near the bottom of the button.

See Also

TBitBtn

TSpeedButton

17.3.3 Buttons.TButtonState Enumeration

TButtonState describes the current state of a button.

Pascal

```
TButtonState = (
  bsUp,
  bsDisabled,
  bsDown,
  bsExclusive
);
```

C++

```
enum TButtonState {
  bsUp,
  bsDisabled,
  bsDown,
  bsExclusive
};
```

File

Buttons

Description

Use TButtonState as an indicator of the state of the button control. The possible values are as follows:

Value	State Description
bsUp	The button control is up, or unselected.
bsDisabled	The button is not enabled, which generally means it is greyed out. It cannot be selected by the user.
bsDown	The button control is down, or selected.

bsExclusive	The button is a "radio button." It is one of a group of buttons, exactly one of which must be selected at any given time. Selecting another button will cause the previously selected button to become deselected.

17.3.4 Buttons.TButtonStyle Enumeration

TButtonStyle specifies the basic look of a TBitBtn (see page 2290) control.

Pascal

```
TButtonStyle = (
  bsAutoDetect,
  bsWin31,
  bsNew
);
```

C++

```
enum TButtonStyle {
  bsAutoDetect,
  bsWin31,
  bsNew
};
```

File

Buttons

Description

The following are possible values of TButtonStyle:

Value	Meaning
bsAutoDetect	When using Windows 3.x, the bitmap button uses the standard Windows 3.x look. When using a later version of Windows, the bitmap button uses a newer look.
bsWin31	Uses the standard Windows 3.1 look, regardless of which version of Windows is running.
bsNew	Uses a new bitmap button look, regardless of which version of Windows is running.

See Also

TBitBtn (see page 2290)

17.4 Types

The following table lists types in this documentation.

Types

Name	Description
TNumGlyphs (see page 2396)	This is type Buttons.TNumGlyphs.

17.4.1 Buttons.TNumGlyphs Type

Pascal

```
TNumGlyphs = 1..4;
```

C++

```
1..4 TNumGlyphs;
```

File

Buttons

Description

This is type Buttons.TNumGlyphs.

18 CaptionedDockTree Namespace

This is namespace CaptionedDockTree.

Topics

Name	Description
Classes (see page 2398)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 2409)	The following table lists structs, records, enums in this documentation.
Types (see page 2411)	The following table lists types in this documentation.
Constants (see page 2413)	The following table lists constants in this documentation.

Classes

	Name	Description
	TCaptionedDockTree (see page 2398)	Defines a new appearance of the docking mechanism for components that can be docked.
	TDockCaptionDrawer (see page 2405)	Creates an instance of the TDockCaptionDrawer.

Constants

Name	Description
dchtCaption (see page 2413)	This is constant CaptionedDockTree.dchtCaption.
dchtClose (see page 2413)	This is constant CaptionedDockTree.dchtClose.
dchtCustom (see page 2414)	This is constant CaptionedDockTree.dchtCustom.
dchtNone (see page 2414)	This is constant CaptionedDockTree.dchtNone.
dchtPin (see page 2414)	This is constant CaptionedDockTree.dchtPin.

Structs, Records, Enums

	Name	Description
	TDockCaptionOrientation (see page 2409)	Defines the values used to determine caption orientation.
	TDockCaptionPinButton (see page 2409)	The pin button styles.
	TParentFormState (see page 2410)	Stores information about the parent dock form for use in drawing the dock site.

Types

Name	Description
TCaptionedDockTreeClass (see page 2411)	This is type CaptionedDockTree.TCaptionedDockTreeClass.
TDockCaptionDrawerClass (see page 2411)	This is type CaptionedDockTree.TDockCaptionDrawerClass.
TDockCaptionHitTest (see page 2411)	Defines the values available for use in a hit test.

18.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCaptionedDockTree (see page 2398)	Defines a new appearance of the docking mechanism for components that can be docked.
	TDockCaptionDrawer (see page 2405)	Creates an instance of the TDockCaptionDrawer.

18.1.1 TCaptionedDockTree Class

Defines a new appearance of the docking mechanism for components that can be docked.

Class Hierarchy

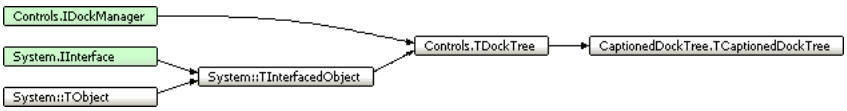

Pascal

```
TCaptionedDockTree = class(TDockTree);
```

C++

```
class TCaptionedDockTree : public TDockTree;
```

File

CaptionedDockTree

Description

Include this class in your project to use the more modern appearance of RAD Studio. If you do not add this class, your docking items will appear as they did in Delphi 7.

18.1.1.1 TCaptionedDockTree Members

The following tables list the members exposed by TCaptionedDockTree.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
V	AdjustCaptionRect (see page 2402)	This is AdjustCaptionRect, a member of class TCaptionedDockTree.

V	AdjustDockRect (see page 2402)	Adjusts the boundaries of the docking zone where a specified control is docked.
V	AdjustFrameRect (see page 5988)	This is AdjustFrameRect, a member of class TDockTree.
	BeginUpdate (see page 5988)	Responds when the docked controls are about to change.
	EndUpdate (see page 5988)	Responds when the docked controls have finished changing.
	FindControlAtPos (see page 5989)	This is FindControlAtPos, a member of class TDockTree.
	GetControlBounds (see page 5989)	Returns the dimensions allotted to a control that is already docked.
V	HitTest (see page 5990)	Indicates which docked control is at a specified mouse position.
V	InsertControl (see page 5990)	Handles the docking of a control into the associated DockSite (see page 5996).
	InternalCaptionHitTest (see page 2403)	This is InternalCaptionHitTest, a member of class TCaptionedDockTree.
V	LoadFromStream (see page 5991)	Creates docking zones for a set of docked controls read from a stream.
V	MouseDown (see page 5991)	This is MouseDown, a member of class TDockTree.
V	MouseMove (see page 5991)	This is MouseMove, a member of class TDockTree.
V	MouseUp (see page 5992)	This is MouseUp, a member of class TDockTree.
V	PaintDockFrame (see page 2404)	Paints the elements of a docking zone that surround a docked control.
V	PositionDockRect (see page 5993)	Computes the dimensions of a new docking zone.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
V	ReferenceFromOrient (see page 5993)	This is ReferenceFromOrient, a member of class TDockTree.
V	RemoveControl (see page 5993)	Handles the undocking of a control from the associated DockSite (see page 5996).
V	ResetBounds (see page 5994)	Recomputes the layout of docked controls.
V	SaveToStream (see page 5994)	Saves a list of the set of docked controls and their positions to a stream.
	SetReplacingControl (see page 5995)	Prepares the dock tree for replacing one docked control with another.
V	ShowHint (see page 5995)	This is ShowHint, a member of class TDockTree.
	UpdateAll (see page 5995)	Makes any internal adjustments to reflect the current state of the docked controls.
V	WndProc (see page 2404)	This is WndProc, a member of class TCaptionedDockTree.
V	ZoneCaptionHitTest (see page 2404)	This is ZoneCaptionHitTest, a member of class TCaptionedDockTree.

Protected Properties

	Name	Description
R	DockCaptionDrawer (see page 2405)	This is DockCaptionDrawer, a member of class TCaptionedDockTree.
R	DockCaptionOrientation (see page 2405)	This is DockCaptionOrientation, a member of class TCaptionedDockTree.
	DockSite (see page 5996)	Specifies the windowed control that uses this dock tree to manage docking operations.
	RelativeSizes (see page 5997)	This is RelativeSizes, a member of class TDockTree.
R	TopZone (see page 5997)	This is TopZone, a member of class TDockTree.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 6239)	Responds when the docked controls are about to change.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2400)	This is the overview for the Create constructor overload.

V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2401)	Frees the memory associated with the TDockTree object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 6240)	Responds when the docked controls have finished changing.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetControlBounds (see page 6240)	Returns the dimensions allotted to a control that is already docked.
V	GetDockCaptionDrawer (see page 2403)	This is GetDockCaptionDrawer, a member of class TCaptionedDockTree.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetParentFormState (see page 2403)	Returns the state of the form that contains the control.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertControl (see page 6240)	Handles the docking of a control into the associated container control.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LoadFromStream (see page 6241)	Lays out a set of docked controls read from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintSite (see page 6241)	Paints the elements that surround all docked controls.
V	PaintSite (see page 5992)	Paints the elements that surround all docked controls.
	PositionDockRect (see page 6242)	Computes the dimensions of a new docking zone.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RemoveControl (see page 6242)	Handles the undocking of a control from the associated container control.
	ResetBounds (see page 6243)	Recomputes the layout of docked controls.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToStream (see page 6245)	Saves a list of the set of docked controls and their positions to a stream.
	SetReplacingControl (see page 6245)	This is SetReplacingControl, a member of class IDockManager.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

18.1.1.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
V	TCaptionedDockTree.Create (TWinControl) (see page 2401)	Creates and initializes a TDockTree object.
	TCaptionedDockTree.Create (TWinControl, TDockCaptionOrientation) (see page 2401)	This is Create, a member of class TCaptionedDockTree.

18.1.1.2.1 TCaptionedDockTree.Create Constructor (TWinControl)

Creates and initializes a TDockTree object.

Pascal

```
constructor Create(DockSite: TWinControl); override; overload;
```

C++

```
virtual __fastcall TCaptionedDockTree(TWinControl * DockSite);
```

Description

Createinstantiates a new TDockTree object to manage docking operations for the control specified by the DockSite parameter. Typically, applications do not create TDockTree objects. Instead, the windowed control that is the value of the DockSite parameter creates the dock tree when an application sets its UseDockManager property to true.

DockSite is the windowed control that uses the dock tree to manage the docking of child controls. It becomes the value of the DockSite property.

Createinitializes the new dock tree to reflect any controls currently docked in DockSite. It then replaces the WindowProc property of DockSite with a method that handles all docking-related messages, passing any other messages on to DockSite's previous window procedure.

See Also

UseDockManager (see page 6212)

DockManager (see page 6202)

DefaultDockTreeClass (see page 6321)

DockSite (see page 5996)

WindowProc (see page 5872)

18.1.1.2.2 TCaptionedDockTree.Create Constructor (TWinControl, TDockCaptionOrientation)

Pascal

```
constructor Create(DockSite: TWinControl; ADockCaptionOrientation: TDockCaptionOrientation);
overload; reintroduce;
```

C++

```
__fastcall TCaptionedDockTree(TWinControl * DockSite, TDockCaptionOrientation
ADockCaptionOrientation);
```

Description

This is Create, a member of class TCaptionedDockTree.

18.1.1.3 TCaptionedDockTree.Destroy Destructor

Frees the memory associated with the TDockTree object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCaptionedDockTree();
```

Description

Do not explicitly destroy TDockTree objects. As interfaced objects, dock trees are destroyed automatically when the reference count of their IDockManager interface drops to zero.

Destroy detaches the dock tree from the windowed control specified by the DockSite property, restoring its WindowProc property to the value it had when it called the TDockTree constructor.

See Also

RefCount (see page 22281)

Create (see page 5985)

DockSite (see page 5996)

WindowProc (see page 5872)

18.1.1.4 TCaptionedDockTree Methods

The methods of the TCaptionedDockTree class are listed here.

Protected Methods

	Name	Description
	AdjustCaptionRect (see page 2402)	This is AdjustCaptionRect, a member of class TCaptionedDockTree.
	AdjustDockRect (see page 2402)	Adjusts the boundaries of the docking zone where a specified control is docked.
	InternalCaptionHitTest (see page 2403)	This is InternalCaptionHitTest, a member of class TCaptionedDockTree.
	PaintDockFrame (see page 2404)	Paints the elements of a docking zone that surround a docked control.
	WndProc (see page 2404)	This is WndProc, a member of class TCaptionedDockTree.
	ZoneCaptionHitTest (see page 2404)	This is ZoneCaptionHitTest, a member of class TCaptionedDockTree.

Public Methods

	Name	Description
	GetDockCaptionDrawer (see page 2403)	This is GetDockCaptionDrawer, a member of class TCaptionedDockTree.
	GetParentFormState (see page 2403)	Returns the state of the form that contains the control.

18.1.1.4.1 TCaptionedDockTree.AdjustCaptionRect Method

Pascal

```
function AdjustCaptionRect(const ARect: TRect): TRect; virtual;
```

C++

```
virtual __fastcall TRect AdjustCaptionRect(const TRect ARect);
```

Description

This is AdjustCaptionRect, a member of class TCaptionedDockTree.

18.1.1.4.2 TCaptionedDockTree.AdjustDockRect Method

Adjusts the boundaries of the docking zone where a specified control is docked.

Pascal

```
procedure AdjustDockRect(Control: TControl; var ARect: TRect); override;
```

C++

```
virtual __fastcall AdjustDockRect(TControl * Control, TRect ARect);
```

Description

Applications can't call the protected AdjustDockRect method. It is called automatically to make any final adjustments to the rectangle where a client control is docked.

Control is the control that is docked.

On entry, ARect indicates the boundaries of Control when it is docked to the windowed control specified by the DockSite property. AdjustDockRect changes this to make any final adjustments to the space in which Control is docked.

In TDockTree, AdjustDockRect expands the ARect parameter to make room for the "grab region" around a docked control. (The grab region is indicated by a double bevel, and is where the user can click with the mouse to start an undocking operation.) AdjustDockRect checks the Align property of the dock site to determine whether the grab region should appear on the side or the top of the docked control.

See Also

Align (see page 5798)

DockSite (see page 5996)

18.1.1.4.3 TCaptionedDockTree.GetDockCaptionDrawer Method

Pascal

```
class function GetDockCaptionDrawer: TDockCaptionDrawerClass; virtual;
```

C++

```
virtual __fastcall TDockCaptionDrawerClass GetDockCaptionDrawer();
```

Description

This is GetDockCaptionDrawer, a member of class TCaptionedDockTree.

18.1.1.4.4 TCaptionedDockTree.GetParentFormState Method

Returns the state of the form that contains the control.

Pascal

```
class function GetParentFormState(const Control: TControl): TParentFormState; virtual;
```

C++

```
virtual __fastcall TParentFormState GetParentFormState(const TControl * Control);
```

18.1.1.4.5 TCaptionedDockTree.InternalCaptionHitTest Method

Pascal

```
function InternalCaptionHitTest(const Zone: TDockZone; const MousePos: TPoint):
TDockCaptionHitTest;
```

C++

```
__fastcall TDockCaptionHitTest InternalCaptionHitTest(const TDockZone Zone, const TPoint
MousePos);
```

Description

This is InternalCaptionHitTest, a member of class TCaptionedDockTree.

18.1.1.4.6 TCaptionedDockTree.PaintDockFrame Method

Paints the elements of a docking zone that surround a docked control.

Pascal

```
procedure PaintDockFrame(Canvas: TCanvas; Control: TControl; const ARect: TRect); override;
```

C++

```
virtual __fastcall PaintDockFrame(TCanvas Canvas, TControl * Control, const TRect ARect);
```

Description

Application's can't call the protected PaintDockFrame method. It is used by the PaintSite method to render the grab lines and close buttons that make up the grab region of a docking zone.

Canvas is the drawing surface on which to paint.

Control is the control that is docked in the zone being rendered.

ARect indicates the boundaries of the entire docking zone.

Note: Descendants classes can override this method to change the way a dock tree paints docking zones. The descendant class should only paint in the portions of ARect that are not occupied by Control.

See Also

PaintSite (see page 5992)

18.1.1.4.7 TCaptionedDockTree.WndProc Method

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

This is WndProc, a member of class TCaptionedDockTree.

18.1.1.4.8 TCaptionedDockTree.ZoneCaptionHitTest Method

Pascal

```
function ZoneCaptionHitTest(const Zone: TDockZone; const MousePos: TPoint; var HTFlag:
Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ZoneCaptionHitTest(const TDockZone Zone, const TPoint MousePos, int
HTFlag);
```

Description

This is ZoneCaptionHitTest, a member of class TCaptionedDockTree.

18.1.1.5 TCaptionedDockTree Properties

The properties of the TCaptionedDockTree class are listed here.

Protected Properties

	Name	Description
R	DockCaptionDrawer (see page 2405)	This is DockCaptionDrawer, a member of class TCaptionedDockTree.
R	DockCaptionOrientation (see page 2405)	This is DockCaptionOrientation, a member of class TCaptionedDockTree.

18.1.1.5.1 TCaptionedDockTree.DockCaptionDrawer Property

Pascal

```
property DockCaptionDrawer: TDockCaptionDrawer;
```

C++

```
__property TDockCaptionDrawer DockCaptionDrawer;
```

Description

This is DockCaptionDrawer, a member of class TCaptionedDockTree.

18.1.1.5.2 TCaptionedDockTree.DockCaptionOrientation Property

Pascal

```
property DockCaptionOrientation: TDockCaptionOrientation;
```

C++

```
__property TDockCaptionOrientation DockCaptionOrientation;
```

Description

This is DockCaptionOrientation, a member of class TCaptionedDockTree.

18.1.2 TDockCaptionDrawer Class

Creates an instance of the TDockCaptionDrawer.

Class Hierarchy

Pascal

```
TDockCaptionDrawer = class(TObject);
```

C++

```
class TDockCaptionDrawer : public TObject;
```

File

CaptionedDockTree

Description

TDockCaptionDrawer is defined as **virtual** so the call to TCaptionedDockTree (see page 2398). GetDockCaptionDrawer.

Create (see page 2406) is called on the correct type.

18.1.2.1 TDockCaptionDrawer Members

The following tables list the members exposed by TDockCaptionDrawer.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2406)	This is Create, a member of class TDockCaptionDrawer.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DockCaptionHitTest (see page 2407)	Indicates which docked control is at a specified mouse position.
	DrawDockCaption (see page 2407)	Draws the caption on the control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	DockCaptionPinButton (see page 2408)	The pin button style to draw, if any.

18.1.2.2 TDockCaptionDrawer.Create Constructor

Pascal

```
constructor Create(DockCaptionOrientation: TDockCaptionOrientation); virtual;
```

C++

```
virtual __fastcall TDockCaptionDrawer(TDockCaptionOrientation DockCaptionOrientation);
```

Description

This is Create, a member of class TDockCaptionDrawer.

18.1.2.3 TDockCaptionDrawer Methods

The methods of the TDockCaptionDrawer class are listed here.

Public Methods

	Name	Description
	DockCaptionHitTest (see page 2407)	Indicates which docked control is at a specified mouse position.
	DrawDockCaption (see page 2407)	Draws the caption on the control.

18.1.2.3.1 TDockCaptionDrawer.DockCaptionHitTest Method

Indicates which docked control is at a specified mouse position.

Pascal

```
function DockCaptionHitTest(const CaptionRect: TRect; const MousePos: TPoint):
TDockCaptionHitTest; virtual;
```

C++

```
virtual __fastcall TDockCaptionHitTest DockCaptionHitTest(const TRect CaptionRect, const
TPoint MousePos);
```

Description

Provides hit tests for the caption. You can return your own hit test results for your own drawer, using custom values.

18.1.2.3.2 TDockCaptionDrawer.DrawDockCaption Method

Draws the caption on the control.

Pascal

```
procedure DrawDockCaption(const Canvas: TCanvas; CaptionRect: TRect; State: TParentFormState);
virtual;
```

C++

```
virtual __fastcall DrawDockCaption(const TCanvas Canvas, TRect CaptionRect, TParentFormState
State);
```

Description

The following parameters are passed in the DrawDockCaption procedure.

Parameter	Description
Canvas	The canvas object upon which to draw the caption.
CaptionRect	The defined rectangle in which the caption is to be drawn.
State	The current state of the parent form.

18.1.2.4 TDockCaptionDrawer Properties

The properties of the TDockCaptionDrawer class are listed here.

Public Properties

	Name	Description
	DockCaptionPinButton (see page 2408)	The pin button style to draw, if any.

18.1.2.4.1 TDockCaptionDrawer.DockCaptionPinButton Property

The pin button style to draw, if any.

Pascal

```
property DockCaptionPinButton: TDockCaptionPinButton;
```

C++

```
__property TDockCaptionPinButton DockCaptionPinButton;
```

Description

The valid values are:

Style	Description
dcpbNone	No style set.
dcpbUp	Shows the push pin in an upward facing direction.
dcpbDown	Shows the push pin in a downward facing direction.

18.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TDockCaptionOrientation (see page 2409)	Defines the values used to determine caption orientation.
	TDockCaptionPinButton (see page 2409)	The pin button styles.

Records

	Name	Description
	TParentFormState (see page 2410)	Stores information about the parent dock form for use in drawing the dock site.

18.2.1 CaptionedDockTree.TDockCaptionOrientation Enumeration

Defines the values used to determine caption orientation.

Pascal

```
TDockCaptionOrientation = (
  dcoHorizontal,
  dcoVertical
);
```

C++

```
enum TDockCaptionOrientation {
  dcoHorizontal,
  dcoVertical
};
```

File

CaptionedDockTree

Description

The following values allow you to set caption orientation:

Value	Description
dcoHorizontal	Positions the caption horizontally.
dcoVertical	Positions the caption vertically.

18.2.2 CaptionedDockTree.TDockCaptionPinButton Enumeration

The pin button styles.

Pascal

```
TDockCaptionPinButton = (
  dcpbNone,
  dcpbUp,
  dcpbDown
);
```

C++

```
enum TDockCaptionPinButton {
  dcpbNone,
  dcpbUp,
  dcpbDown
};
```

File

CaptionedDockTree

Description

The valid values are:

Style	Description
dcpbNone	No style set.
dcpbUp	Shows the push pin in an upward facing direction.
dcpbDown	Shows the push pin in a downward facing direction.

18.2.3 CaptionedDockTree.TParentFormState Record

Stores information about the parent dock form for use in drawing the dock site.

Pascal

```
TParentFormState = record
  Caption: string;
  EndColor: TColor;
  FontColor: TColor;
  StartColor: TColor;
  Focused: Boolean;
  Icon: TIcon;
end;
```

C++

```
struct TParentFormState {
  AnsiString Caption;
  TColor EndColor;
  TColor FontColor;
  TColor StartColor;
  Boolean Focused;
  TIcon Icon;
};
```

File

CaptionedDockTree

18.3 Types

The following table lists types in this documentation.

Types

Name	Description
TCaptionedDockTreeClass (see page 2411)	This is type CaptionedDockTree.TCaptionedDockTreeClass.
TDockCaptionDrawerClass (see page 2411)	This is type CaptionedDockTree.TDockCaptionDrawerClass.
TDockCaptionHitTest (see page 2411)	Defines the values available for use in a hit test.

18.3.1 CaptionedDockTree.TCaptionedDockTreeClass Type

Pascal

```
TCaptionedDockTreeClass = class of TCaptionedDockTree;
```

C++

```
TCaptionedDockTree TCaptionedDockTreeClass;
```

File

CaptionedDockTree

Description

This is type CaptionedDockTree.TCaptionedDockTreeClass.

18.3.2 CaptionedDockTree.TDockCaptionDrawerClass Type

Pascal

```
TDockCaptionDrawerClass = class of TDockCaptionDrawer;
```

C++

```
TDockCaptionDrawer TDockCaptionDrawerClass;
```

File

CaptionedDockTree

Description

This is type CaptionedDockTree.TDockCaptionDrawerClass.

18.3.3 CaptionedDockTree.TDockCaptionHitTest Type

Defines the values available for use in a hit test.

Pascal

```
TDockCaptionHitTest = Cardinal;
```

C++

```
unsigned TDockCaptionHitTest;
```

File

CaptionedDockTree

Description

The following values are used in hit tests:

Constant	Value
dchtNone (see page 2414)	0
dchtCaption (see page 2413)	1
dchtClose (see page 2413)	2
dchtPin (see page 2414)	3
dchtCustom (see page 2414)	10 or above

You can create your own hit tests and use **dchtCustom (see page 2414)** to set values. Values 4 through 9 are reserved. The default value of **dchtCustom (see page 2414)** may change.

18.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
dchtCaption (see page 2413)	This is constant CaptionedDockTree.dchtCaption.
dchtClose (see page 2413)	This is constant CaptionedDockTree.dchtClose.
dchtCustom (see page 2414)	This is constant CaptionedDockTree.dchtCustom.
dchtNone (see page 2414)	This is constant CaptionedDockTree.dchtNone.
dchtPin (see page 2414)	This is constant CaptionedDockTree.dchtPin.

18.4.1 CaptionedDockTree.dchtCaption Constant

Pascal

```
dchtCaption = 1;
```

C++

```
dchtCaption = 1;
```

File

CaptionedDockTree

Description

This is constant CaptionedDockTree.dchtCaption.

18.4.2 CaptionedDockTree.dchtClose Constant

Pascal

```
dchtClose = 2;
```

C++

```
dchtClose = 2;
```

File

CaptionedDockTree

Description

This is constant CaptionedDockTree.dchtClose.

18.4.3 CaptionedDockTree.dchtCustom Constant

Pascal

```
dchtCustom = 10;
```

C++

```
dchtCustom = 10;
```

File

CaptionedDockTree

Description

This is constant CaptionedDockTree.dchtCustom.

18.4.4 CaptionedDockTree.dchtNone Constant

Pascal

```
dchtNone = 0;
```

C++

```
dchtNone = 0;
```

File

CaptionedDockTree

Description

This is constant CaptionedDockTree.dchtNone.

18.4.5 CaptionedDockTree.dchtPin Constant

Pascal

```
dchtPin = 3;
```

C++

```
dchtPin = 3;
```

File

CaptionedDockTree

Description

This is constant CaptionedDockTree.dchtPin.

19 CategoryButtons Namespace

This is namespace CategoryButtons.

Topics

Name	Description
Classes (see page 2417)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 2543)	The following table lists structs, records, enums in this documentation.
Types (see page 2544)	The following table lists types in this documentation.
Constants (see page 2551)	The following table lists constants in this documentation.

Classes

	Name	Description
	TBaseButtonItem (see page 2417)	TBaseButtonItem is the base class for button items used by TButtonGroup and TCategoryButtons (see page 2452).
	TButtonCategories (see page 2424)	TButtonCategories maintains the collection of button categories that appear in a TCategoryButtons (see page 2452) control.
	TButtonCategory (see page 2429)	TButtonCategory is an individual button category of a TCategoryButtons (see page 2452) control.
	TButtonCollection (see page 2436)	Defines a collection of buttons that appear in a TButtonCategory (see page 2429).
	TButtonItem (see page 2441)	An instance of a button in a TCategoryButtons (see page 2452) control.
	TButtonItemActionLink (see page 2445)	TButtonItemActionLink is an action link designed specifically for controls using TButtonItem (see page 2441).
	TCategoryButtons (see page 2452)	Defines a group of categorized buttons.

Constants

Name	Description
crDragCopy (see page 2551)	This is constant CategoryButtons.crDragCopy.

Structs, Records, Enums

	Name	Description
	TCatButtonFlow (see page 2543)	Defines the ordering style of category buttons.

Types

Name	Description
TBaseButtonItemClass (see page 2544)	Defines button items that are to be included in a container, such as in button groups or category buttons.
TButtonCategoriesClass (see page 2544)	Metaclass of the TButtonCategories (see page 2424) class.

TButtonCategoryClass (see page 2545)	Metaclass of TButtonCategory (see page 2429).
TButtonDrawState (see page 2545)	Values indicating the drawing state of buttons.
TButtonItemActionLinkClass (see page 2546)	Metaclass of TButtonItemActionLink (see page 2445).
TButtonItemClass (see page 2546)	Defines the button item class.
TCatButtonCopyEvent (see page 2546)	This is type CategoryButtons.TCatButtonCopyEvent.
TCatButtonDrawEvent (see page 2547)	This is type CategoryButtons.TCatButtonDrawEvent.
TCatButtonDrawIconEvent (see page 2547)	This is type CategoryButtons.TCatButtonDrawIconEvent.
TCatButtonEvent (see page 2547)	This is type CategoryButtons.TCatButtonEvent.
TCatButtonGetHint (see page 2548)	This is type CategoryButtons.TCatButtonGetHint.
TCatButtonOptions (see page 2548)	Defines button options for category buttons.
TCatButtonReorderEvent (see page 2549)	This is type CategoryButtons.TCatButtonReorderEvent.
TCategoryCollapseEvent (see page 2549)	This is type CategoryButtons.TCategoryCollapseEvent.
TCategoryReorderEvent (see page 2549)	This is type CategoryButtons.TCategoryReorderEvent.

19.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TBaseButtonItem (see page 2417)	TBaseButtonItem is the base class for button items used by TButtonGroup and TCategoryButtons (see page 2452).
	TButtonCategories (see page 2424)	TButtonCategories maintains the collection of button categories that appear in a TCategoryButtons (see page 2452) control.
	TButtonCategory (see page 2429)	TButtonCategory is an individual button category of a TCategoryButtons (see page 2452) control.
	TButtonCollection (see page 2436)	Defines a collection of buttons that appear in a TButtonCategory (see page 2429).
	TButtonItem (see page 2441)	An instance of a button in a TCategoryButtons (see page 2452) control.
	TButtonItemActionLink (see page 2445)	TButtonItemActionLink is an action link designed specifically for controls using TButtonItem (see page 2441).
	TCategoryButtons (see page 2452)	Defines a group of categorized buttons.

19.1.1 TBaseButtonItem Class

TBaseButtonItem is the base class for button items used by TButtonGroup and TCategoryButtons (see page 2452).

Class Hierarchy

Pascal

```
TBaseButtonItem = class(TCollectionItem);
```

C++

```
class TBaseButtonItem : public TCollectionItem;
```

File

CategoryButtons

19.1.1.1 TBaseButtonItem Members

The following tables list the members exposed by TBaseButtonItem.

Protected Methods

	Name	Description
	ActionChange (see page 2420)	This is ActionChange, a member of class TBaseButtonItem.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetActionLinkClass (see page 2421)	This is GetActionLinkClass, a member of class TBaseButtonItem.
	GetDisplayName (see page 2421)	Returns the name of the collection item as it appears in the collection editor.

	GetNotifyTarget (see page 2421)	This is GetNotifyTarget, a member of class TBaseButtonItem.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Protected Properties

	Name	Description
	ActionLink (see page 2422)	This is ActionLink, a member of class TBaseButtonItem.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2420)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2419)	Creates and initializes a TCollectionItem instance.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScrollIntoView (see page 2421)	Scrolls a button to make it visible.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	Data (see page 2423)	This is Data, a member of class TBaseButtonItem.

	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Events

	Name	Description
	OnClick (see page 2423)	This is OnClick, a member of class TBaseButtonItem.

Published Properties

	Name	Description
	Action (see page 2422)	Specifies the action the button executes when clicked.
	Caption (see page 2422)	This is Caption, a member of class TBaseButtonItem.
	Hint (see page 2423)	This is Hint, a member of class TBaseButtonItem.
	ImageIndex (see page 2423)	The index of a specified image.

19.1.1.2 TBaseButtonItem.Create Constructor

Creates and initializes a TCollectionItem instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TBaseButtonItem(TCollection Collection);
```

Description

Do not call Create to instantiate a collection item. Instead, call the Add method of the collection to which the item should belong.

Collection identifies the TCollection instance to which the new item belongs.

See Also

Add (see page 2736)

Collection (see page 2761)

19.1.1.3 TBaseButtonItem Methods

The methods of the TBaseButtonItem class are listed here.

Protected Methods

	Name	Description
V	ActionChange (see page 2420)	This is ActionChange, a member of class TBaseButtonItem.
V	GetActionLinkClass (see page 2421)	This is GetActionLinkClass, a member of class TBaseButtonItem.
V	GetDisplayName (see page 2421)	Returns the name of the collection item as it appears in the collection editor.
A	GetNotifyTarget (see page 2421)	This is GetNotifyTarget, a member of class TBaseButtonItem.

Public Methods

	Name	Description
V	Assign (see page 2420)	Copies the contents of another, similar object.
A	ScrollIntoView (see page 2421)	Scrolls a button to make it visible.

19.1.1.3.1 TBaseButtonItem.ActionChange Method

Pascal

```
procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); virtual;
```

C++

```
virtual __fastcall ActionChange(TObject * Sender, Boolean CheckDefaults);
```

Description

This is ActionChange, a member of class TBaseButtonItem.

19.1.1.3.2 TBaseButtonItem.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo (see page 2931)

EConvertError (see page 22724)

19.1.1.3.3 TBaseButtonItem.GetActionLinkClass Method

Pascal

```
function GetActionLinkClass: TButtonItemActionLinkClass; virtual;
```

C++

```
virtual __fastcall TButtonItemActionLinkClass GetActionLinkClass();
```

Description

This is GetActionLinkClass, a member of class TBaseButtonItem.

19.1.1.3.4 TBaseButtonItem.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName (see page 2761)

GetNamePath (see page 2759)

19.1.1.3.5 TBaseButtonItem.GetNotifyTarget Method

Pascal

```
function GetNotifyTarget: TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * GetNotifyTarget() = 0;
```

Description

This is GetNotifyTarget, a member of class TBaseButtonItem.

19.1.1.3.6 TBaseButtonItem.ScrollIntoView Method

Scrolls a button to make it visible.

Pascal

```
procedure ScrollIntoView; virtual; abstract;
```

C++

```
virtual __fastcall ScrollIntoView() = 0;
```

Description

Supply the following value for this method:

Parameter	Description
Index	An Integer representing the button you want scrolled into view.

19.1.1.4 TBaseButtonItem Properties

The properties of the TBaseButtonItem class are listed here.

Protected Properties

	Name	Description
	ActionLink (see page 2422)	This is ActionLink, a member of class TBaseButtonItem.

Public Properties

	Name	Description
	Data (see page 2423)	This is Data, a member of class TBaseButtonItem.

Published Properties

	Name	Description
	Action (see page 2422)	Specifies the action the button executes when clicked.
	Caption (see page 2422)	This is Caption, a member of class TBaseButtonItem.
	Hint (see page 2423)	This is Hint, a member of class TBaseButtonItem.
	ImageIndex (see page 2423)	The index of a specified image.

19.1.1.4.1 TBaseButtonItem.Action Property

Specifies the action the button executes when clicked.

Pascal

```
property Action: TBasicAction;
```

C++

```
__property TBasicAction Action;
```

19.1.1.4.2 TBaseButtonItem.ActionLink Property

Pascal

```
property ActionLink: TButtonItemActionLink;
```

C++

```
__property TButtonItemActionLink ActionLink;
```

Description

This is ActionLink, a member of class TBaseButtonItem.

19.1.1.4.3 TBaseButtonItem.Caption Property

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

This is Caption, a member of class TBaseButtonItem.

19.1.1.4.4 TBaseButtonItem.Data Property

Pascal

```
property Data: Pointer;
```

C++

```
__property void * Data;
```

Description

This is Data, a member of class TBaseButtonItem.

19.1.1.4.5 TBaseButtonItem.Hint Property

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

This is Hint, a member of class TBaseButtonItem.

19.1.1.4.6 TBaseButtonItem.ImageIndex Property

The index of a specified image.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

19.1.1.5 TBaseButtonItem Events

The events of the TBaseButtonItem class are listed here.

Published Events

	Name	Description
	OnClick (see page 2423)	This is OnClick, a member of class TBaseButtonItem.

19.1.1.5.1 TBaseButtonItem.OnClick Event

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

This is OnClick, a member of class TBaseButtonItem.

19.1.2 TButtonCategories Class

TButtonCategories maintains the collection of button categories that appear in a TCategoryButtons (see page 2452) control.

Class Hierarchy

System::TObject → Classes.TPersistent → Classes.TCollection → CategoryButtons.TButtonCategories

Pascal

```
TButtonCategories = class(TCollection);
```

C++

```
class TButtonCategories : public TCollection;
```

File

CategoryButtons

19.1.2.1 TButtonCategories Members

The following tables list the members exposed by TButtonCategories.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2427)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2428)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 2426)	Adds a new item at the end of the collection.
	AddItem (see page 2426)	Adds an item to the category at the specified index.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2427)	Begins the update procedure to update the button list whenever the categories or buttons within the categories are modified.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 2426)	This is Create, a member of class TButtonCategories.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	IndexOf (see page 2427)	This is IndexOf, a member of class TButtonCategories.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2428)	Inserts a new item at the specified index.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ButtonGroup (see page 2428)	A specified button group within the category button container.
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 2429)	The number of categories in the category button container.

19.1.2.2 TButtonCategories.Create Constructor

Pascal

```
constructor Create(const ButtonGroup: TCategoryButtons);
```

C++

```
__fastcall TButtonCategories(const TCategoryButtons ButtonGroup);
```

Description

This is Create, a member of class TButtonCategories.

19.1.2.3 TButtonCategories Methods

The methods of the TButtonCategories class are listed here.

Protected Methods

	Name	Description
V	GetOwner (see page 2427)	Returns the owner of an object.
V	Update (see page 2428)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 2426)	Adds a new item at the end of the collection.
	AddItem (see page 2426)	Adds an item to the category at the specified index.
V	BeginUpdate (see page 2427)	Begins the update procedure to update the button list whenever the categories or buttons within the categories are modified.
	IndexOf (see page 2427)	This is IndexOf, a member of class TButtonCategories.
	Insert (see page 2428)	Inserts a new item at the specified index.

19.1.2.3.1 TButtonCategories.Add Method

Adds a new item at the end of the collection.

Pascal

```
function Add: TButtonCategory;
```

C++

```
__fastcall TButtonCategory Add();
```

19.1.2.3.2 TButtonCategories.AddItem Method

Adds an item to the category at the specified index.

Pascal

```
function AddItem(Item: TButtonCategory; Index: Integer): TButtonCategory;
```

C++

```
__fastcall TButtonCategory AddItem(TButtonCategory Item, int Index);
```

Description

The scope determines whether the method adds a new category or a new button.

19.1.2.3.3 TButtonCategories.BeginUpdate Method

Begins the update procedure to update the button list whenever the categories or buttons within the categories are modified.

Pascal

```
procedure BeginUpdate; override;
```

C++

```
virtual __fastcall BeginUpdate();
```

19.1.2.3.4 TButtonCategories.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

19.1.2.3.5 TButtonCategories.IndexOf Method

Pascal

```
function IndexOf(const Caption: string): Integer;
```

C++

```
__fastcall int IndexOf(const AnsiString Caption);
```

Description

This is IndexOf, a member of class TButtonCategories.

19.1.2.3.6 TButtonCategories.Insert Method

Inserts a new item at the specified index.

Pascal

```
function Insert(Index: Integer): TButtonCategory;
```

C++

```
__fastcall TButtonCategory Insert(int Index);
```

19.1.2.3.7 TButtonCategories.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed (see page 2740)

Changed (see page 2758)

19.1.2.4 TButtonCategories Properties

The properties of the TButtonCategories class are listed here.

Public Properties

	Name	Description
R	ButtonGroup (see page 2428)	A specified button group within the category button container.
	Items (see page 2429)	The number of categories in the category button container.

19.1.2.4.1 TButtonCategories.ButtonGroup Property

A specified button group within the category button container.

Pascal

```
property ButtonGroup: TCategoryButtons;
```

C++

```
__property TCategoryButtons ButtonGroup;
```

19.1.2.4.2 TButtonCategories.Items Property

The number of categories in the category button container.

Pascal

```
property Items [Index: Integer]: TButtonCategory;
```

C++

```
__property TButtonCategory Items[int Index];
```

19.1.3 TButtonCategory Class

TButtonCategory is an individual button category of a TCategoryButtons (see page 2452) control.

Class Hierarchy

Pascal

```
TButtonCategory = class(TCollectionItem);
```

C++

```
class TButtonCategory : public TCollectionItem;
```

File

CategoryButtons

19.1.3.1 TButtonCategory Members

The following tables list the members exposed by TButtonCategory.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2433)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2432)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2431)	Creates and initializes a TCollectionItem instance.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2431)	Destroys the TCollectionItem instance and frees its memory.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	IndexOf (see page 2432)	This is IndexOf, a member of class TButtonCategory.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScrollIntoView (see page 2433)	Scrolls a button to make it visible within a group or category.

Public Properties

	Name	Description
R	Categories (see page 2434)	Retrieves the owning ButtonCategories object.
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	Data (see page 2434)	This is Data, a member of class TButtonCategory.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
	InterfaceData (see page 2435)	Interface data for the button category object.

Published Properties

	Name	Description
	Caption (see page 2434)	Specifies the caption for the button category.
	Collapsed (see page 2434)	Specifies whether or not the button in the collection is collapsed.
	Color (see page 2434)	Specifies the color of the category item.
	GradientColor (see page 2435)	Specifies the colors used for a gradient fill on the category label in a category button object.
	Items (see page 2435)	Specify the button items in this category.
	TextColor (see page 2435)	Specify the color used for the font in category captions.

19.1.3.2 TButtonCategory.Create Constructor

Creates and initializes a TCollectionItem instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TButtonCategory(TCollection Collection);
```

Description

Do not call Create to instantiate a collection item. Instead, call the Add method of the collection to which the item should belong.

Collection identifies the TCollection instance to which the new item belongs.

See Also

Add (see page 2736)

Collection (see page 2761)

19.1.3.3 TButtonCategory.Destroy Destructor

Destroys the TCollectionItem instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TButtonCategory();
```

Description

Destroy is called indirectly by TCollection's Clear or Delete method.

See Also

Clear (see page 2741)

Delete (see page 2741)

Free (see page 22300)

19.1.3.4 TButtonCategory Methods

The methods of the TButtonCategory class are listed here.

Protected Methods

	Name	Description
	SetIndex (see page 2433)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	Assign (see page 2432)	Copies the contents of another, similar object.
	IndexOf (see page 2432)	This is IndexOf, a member of class TButtonCategory.
	ScrollIntoView (see page 2433)	Scrolls a button to make it visible within a group or category.

19.1.3.4.1 TButtonCategory.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo (see page 2931)

EConvertError (see page 22724)

19.1.3.4.2 TButtonCategory.IndexOf Method

Pascal

```
function IndexOf(const Caption: string): Integer;
```

C++

```
__fastcall int IndexOf(const AnsiString Caption);
```

Description

This is IndexOf, a member of class TButtonCategory.

19.1.3.4.3 TButtonCategory.ScrollIntoView Method

Scrolls a button to make it visible within a group or category.

Pascal

```
procedure ScrollIntoView;
```

C++

```
__fastcall ScrollIntoView();
```

Description

Supply the following value for this method:

Parameter	Description
Index	An Integer representing the button you want scrolled into view.

19.1.3.4.4 TButtonCategory.SetIndex Method

Moves the collection item to a specified position in the parent collection.

Pascal

```
procedure SetIndex(Value: Integer); override;
```

C++

```
virtual __fastcall SetIndex(int Value);
```

Description

SetIndex is the protected write implementation of the Index property. The Value parameter specifies the new (0-based) position for the item in its parent collection.

See Also

Index (see page 2762)

19.1.3.5 TButtonCategory Properties

The properties of the TButtonCategory class are listed here.

Public Properties

	Name	Description
R	Categories (see page 2434)	Retrieves the owning ButtonCategories object.
	Data (see page 2434)	This is Data, a member of class TButtonCategory.
	InterfaceData (see page 2435)	Interface data for the button category object.

Published Properties

	Name	Description
	Caption (see page 2434)	Specifies the caption for the button category.
	Collapsed (see page 2434)	Specifies whether or not the button in the collection is collapsed.
	Color (see page 2434)	Specifies the color of the category item.
	GradientColor (see page 2435)	Specifies the colors used for a gradient fill on the category label in a category button object.
	Items (see page 2435)	Specify the button items in this category.
	TextColor (see page 2435)	Specify the color used for the font in category captions.

19.1.3.5.1 TButtonCategory.Caption Property

Specifies the caption for the button category.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

19.1.3.5.2 TButtonCategory.Categories Property

Retrieves the owning ButtonCategories object.

Pascal

```
property Categories: TButtonCategories;
```

C++

```
__property TButtonCategories Categories;
```

19.1.3.5.3 TButtonCategory.Collapsed Property

Specifies whether or not the button in the collection is collapsed.

Pascal

```
property Collapsed: Boolean;
```

C++

```
__property Boolean Collapsed;
```

Description

If set to **True**, the button is collapsed, otherwise, when set to **False**, the button is expanded.

19.1.3.5.4 TButtonCategory.Color Property

Specifies the color of the category item.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

The category appears as either a vertical or horizontal bar within the category button container. This property applies to the background of the container.

19.1.3.5.5 TButtonCategory.Data Property

Pascal

```
property Data: Pointer;
```

C++

```
__property void * Data;
```

Description

This is Data, a member of class TButtonCategory.

19.1.3.5.6 TButtonCategory.GradientColor Property

Specifies the colors used for a gradient fill on the category label in a category button object.

Pascal

```
property GradientColor: TColor;
```

C++

```
__property TColor GradientColor;
```

Description

Inherits from TColor.

19.1.3.5.7 TButtonCategory.InterfaceData Property

Interface data for the button category object.

Pascal

```
property InterfaceData: IInterface;
```

C++

```
__property IInterface InterfaceData;
```

19.1.3.5.8 TButtonCategory.Items Property

Specify the button items in this category.

Pascal

```
property Items: TButtonCollection;
```

C++

```
__property TButtonCollection Items;
```

Description

You define items from within the context of the **Categories (see page 2434) Editor**. When the **Categories (see page 2434) Editor** is displayed, the **Object Inspector** changes to reflect what is in the **Categories (see page 2434) Editor**. In the **Object Inspector**, you can click the Items property to display the **Items Editor**. Use the **Items Editor** to add buttons to the category.

19.1.3.5.9 TButtonCategory.TextColor Property

Specify the color used for the font in category captions.

Pascal

```
property TextColor: TColor;
```

C++

```
__property TColor TextColor;
```

19.1.4 TButtonCollection Class

Defines a collection of buttons that appear in a TButtonCategory (see page 2429).

Class Hierarchy

System::TObject → Classes.TPersistent → Classes.TCollection → CategoryButtons.TButtonCollection

Pascal

```
TButtonCollection = class(TCollection);
```

C++

```
class TButtonCollection : public TCollection;
```

File

CategoryButtons

19.1.4.1 TButtonCollection Members

The following tables list the members exposed by TButtonCollection.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2438)	Returns the owner of an object.
V	Notify (see page 2439)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2440)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 2438)	Adds a button to the end of the collection.
	AddItem (see page 2438)	Adds a button to a collection at a specified index.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 2438)	The constructor for the collection.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2439)	Inserts a button in the collection at the specified index.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Category (see page 2440)	The name of the current category.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 2440)	Defines the button items in a category.

19.1.4.2 TButtonCollection.Create Constructor

The constructor for the collection.

Pascal

```
constructor Create(const ACategory: TButtonCategory);
```

C++

```
__fastcall TButtonCollection(const TButtonCategory ACategory);
```

19.1.4.3 TButtonCollection Methods

The methods of the TButtonCollection class are listed here.

Protected Methods

	Name	Description
	GetOwner (see page 2438)	Returns the owner of an object.
	Notify (see page 2439)	Responds when items are added to or removed from the collection.
	Update (see page 2440)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 2438)	Adds a button to the end of the collection.
	AddItem (see page 2438)	Adds a button to a collection at a specified index.
	Insert (see page 2439)	Inserts a button in the collection at the specified index.

19.1.4.3.1 TButtonCollection.Add Method

Adds a button to the end of the collection.

Pascal

```
function Add: TButtonItem;
```

C++

```
__fastcall TButtonItem Add();
```

19.1.4.3.2 TButtonCollection.AddItem Method

Adds a button to a collection at a specified index.

Pascal

```
function AddItem(Item: TButtonItem; Index: Integer): TButtonItem;
```

C++

```
__fastcall TButtonItem AddItem(TButtonItem Item, int Index);
```

19.1.4.3.3 TButtonCollection.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

19.1.4.3.4 TButtonCollection.Insert Method

Inserts a button in the collection at the specified index.

Pascal

```
function Insert(Index: Integer): TButtonItem;
```

C++

```
__fastcall TButtonItem Insert(int Index);
```

19.1.4.3.5 TButtonCollection.Notify Method

Responds when items are added to or removed from the collection.

Pascal

```
procedure Notify(Item: TCollectionItem; Action: TCollectionNotification); override;
```

C++

```
virtual __fastcall Notify(TCollectionItem Item, TCollectionNotification Action);
```

Description

Notify is called automatically when the items in the collection change.

Item is the item that was just added to or that is about to be removed from the collection.

Action indicates whether item was added, is about to be removed, or is about to be deleted:

As implemented in TCollection, Notify calls Added when Action is cnAdded and calls Deleting when Action is cnDeleting. TCollection ignores the cnExtracting action. Descendant classes can override Notify to modify this behavior.

See Also

Added (see page 2736)

Deleting (see page 2741)

19.1.4.3.6 TButtonCollection.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed (see page 2740)

Changed (see page 2758)

19.1.4.4 TButtonCollection Properties

The properties of the TButtonCollection class are listed here.

Public Properties

	Name	Description
R	Category (see page 2440)	The name of the current category.
	Items (see page 2440)	Defines the button items in a category.

19.1.4.4.1 TButtonCollection.Category Property

The name of the current category.

Pascal

```
property Category: TButtonCategory;
```

C++

```
__property TButtonCategory Category;
```

19.1.4.4.2 TButtonCollection.Items Property

Defines the button items in a category.

Pascal

```
property Items [Index: Integer]: TButtonItem;
```

C++

```
__property TButtonItem Items[int Index];
```

19.1.5 TButtonItem Class

An instance of a button in a TCategoryButtons (see page 2452) control.

Class Hierarchy

System::TObject → Classes.TPersistent → Classes.TCollectionItem → CategoryButtons.TBaseButtonItem → CategoryButtons.TButtonItem

Pascal

```
TButtonItem = class(TBaseButtonItem);
```

C++

```
class TButtonItem : public TBaseButtonItem;
```

File

CategoryButtons

Description

You add button items to the category by using the **Items Editor**. The **Items Editor** can be invoked from the Items property in the **Object Inspector**, only when the **Categories Editor** is displayed and a category is selected.

19.1.5.1 TButtonItem Members

The following tables list the members exposed by TButtonItem.

Protected Methods

	Name	Description
	ActionChange (see page 2420)	This is ActionChange, a member of class TBaseButtonItem.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetActionLinkClass (see page 2421)	This is GetActionLinkClass, a member of class TBaseButtonItem.
	GetDisplayName (see page 2421)	Returns the name of the collection item as it appears in the collection editor.
	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
	GetNotifyTarget (see page 2444)	This is GetNotifyTarget, a member of class TButtonItem.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Protected Properties

	Name	Description
	ActionLink (see page 2422)	This is ActionLink, a member of class TBaseButtonItem.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2443)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2419)	Creates and initializes a TCollectionItem instance.
V	Create (see page 2757)	Creates and initializes a TCollectionItem (see page 2755) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	ScrollIntoView (see page 2444)	Scrolls a button to make it visible within a button group.

Public Properties

	Name	Description
R	Category (see page 2445)	The parent category for the given button item.
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	Data (see page 2423)	This is Data, a member of class TBaseButtonItem.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
	InterfaceData (see page 2445)	Interface data for the button item.

Published Events

	Name	Description
	OnClick (see page 2423)	This is OnClick, a member of class TBaseButtonItem.

Published Properties

	Name	Description
	Action (see page 2422)	Specifies the action the button executes when clicked.
R	ButtonGroup (see page 2444)	The parent button group of the given button item.

	Caption (see page 2422)	This is Caption, a member of class TBaseButtonItem.
	Hint (see page 2423)	This is Hint, a member of class TBaseButtonItem.
	ImageIndex (see page 2423)	The index of a specified image.

19.1.5.2 TButtonItem Methods

The methods of the TButtonItem class are listed here.

Protected Methods

	Name	Description
	GetNotifyTarget (see page 2444)	This is GetNotifyTarget, a member of class TButtonItem.

Public Methods

	Name	Description
	Assign (see page 2443)	Copies the contents of another, similar object.
	ScrollIntoView (see page 2444)	Scrolls a button to make it visible within a button group.

19.1.5.2.1 TButtonItem.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo (see page 2931)

EConvertError (see page 22724)

19.1.5.2.2 TButtonItem.GetNotifyTarget Method

Pascal

```
function GetNotifyTarget: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetNotifyTarget();
```

Description

This is GetNotifyTarget, a member of class TButtonItem.

19.1.5.2.3 TButtonItem.ScrollIntoView Method

Scrolls a button to make it visible within a button group.

Pascal

```
procedure ScrollIntoView; override;
```

C++

```
virtual __fastcall ScrollIntoView();
```

Description

Supply the following value for this method:

Parameter	Description
Index	An Integer representing the button you want scrolled into view.

19.1.5.3 TButtonItem Properties

The properties of the TButtonItem class are listed here.

Public Properties

	Name	Description
R	Category (see page 2445)	The parent category for the given button item.
	InterfaceData (see page 2445)	Interface data for the button item.

Published Properties

	Name	Description
R	ButtonGroup (see page 2444)	The parent button group of the given button item.

19.1.5.3.1 TButtonItem.ButtonGroup Property

The parent button group of the given button item.

Pascal

```
property ButtonGroup: TCategoryButtons;
```

C++

```
__property TCategoryButtons ButtonGroup;
```

19.1.5.3.2 TButtonItem.Category Property

The parent category for the given button item.

Pascal

```
property Category: TButtonCategory;
```

C++

```
__property TButtonCategory Category;
```

19.1.5.3.3 TButtonItem.InterfaceData Property

Interface data for the button item.

Pascal

```
property InterfaceData: IInterface;
```

C++

```
__property IInterface InterfaceData;
```

19.1.6 TButtonItemActionLink Class

TButtonItemActionLink is an action link designed specifically for controls using TButtonItem (see page 2441).

Class Hierarchy

Pascal

```
TButtonItemActionLink = class(TActionLink);
```

C++

```
class TButtonItemActionLink : public TActionLink;
```

File

CategoryButtons

19.1.6.1 TButtonItemActionLink Members

The following tables list the members exposed by TButtonItemActionLink.

Protected Fields

	Name	Description
	FAction (see page 2722)	This is FAction, a member of class TBasicActionLink.
	FClient (see page 2447)	This is FClient, a member of class TButtonItemActionLink.

Protected Methods

	Name	Description
	AssignClient (see page 2448)	This is AssignClient, a member of class TButtonItemActionLink.

	Change (see page 2723)	This is Change, a member of class TBasicActionLink.
	IsCaptionLinked (see page 2448)	Indicates whether the Caption properties for the action and client are linked.
	IsCheckedLinked (see page 269)	Indicates whether the Checked properties for the action and client are linked.
	IsEnabledLinked (see page 269)	Indicates whether the Enabled properties for the action and client are linked.
	IsGroupIndexLinked (see page 270)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsHelpContextLinked (see page 270)	Indicates whether the HelpContext properties for the action and client controls are linked.
	IsHelpLinked (see page 271)	Indicates whether the help properties for the action and client are linked.
	IsHintLinked (see page 2449)	Indicates whether the Hint properties for the action and client are linked.
	IsImageIndexLinked (see page 2449)	Indicates whether the image index for the action and the client are linked.
	IsOnExecuteLinked (see page 2450)	This is IsOnExecuteLinked, a member of class TButtonItemActionLink.
	IsShortCutLinked (see page 272)	Indicates whether the ShortCut properties for the action and client are linked.
	IsVisibleLinked (see page 273)	Indicates whether the Visible properties for the action and client are linked.
	SetAction (see page 2724)	This is SetAction, a member of class TBasicActionLink.
	SetAutoCheck (see page 274)	Introduces an interface for setting the AutoCheck property of a client.
	SetCaption (see page 2450)	Introduces an interface for setting the caption on a client.
	SetChecked (see page 274)	Introduces an interface for setting the checked state of a client.
	SetEnabled (see page 275)	Introduces an interface for setting the enabled state of a client.
	SetGroupIndex (see page 275)	Introduces an interface for setting the GroupIndex property of a client.
	SetHelpContext (see page 276)	Introduces an interface for setting the help context of a client.
	SetHelpKeyword (see page 276)	Introduces an interface for setting the help keyword of a client.
	SetHelpType (see page 277)	Introduces an interface for setting the HelpType property of a client.
	SetHint (see page 2450)	Introduces an interface for setting the hint for a client.
	SetImageIndex (see page 2451)	Introduces an interface for setting the image index for a client.
	SetOnExecute (see page 2452)	This is SetOnExecute, a member of class TButtonItemActionLink.
	SetShortCut (see page 279)	Introduces an interface for setting the shortcut for a client.
	SetVisible (see page 279)	Introduces an interface for setting the visible state of a client.

Public Events

	Name	Description
	OnChange (see page 2725)	Occurs when an OnChange event is triggered on the action.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2721)	Instantiates and initializes a TBasicActionLink (see page 2720) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2722)	Destroys an instance of an action link object.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoShowHint (see page 2448)	This is DoShowHint, a member of class TButtonItemActionLink.
	Execute (see page 2723)	Invokes the action's Execute method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Update (see page 2724)	Invokes the action's Update method.

Public Properties

	Name	Description
	Action (see page 2725)	Indicates which action the link connects to its client.

19.1.6.2 TButtonItemActionLink Fields

The fields of the TButtonItemActionLink class are listed here.

Protected Fields

	Name	Description
	FClient (see page 2447)	This is FClient, a member of class TButtonItemActionLink.

19.1.6.2.1 TButtonItemActionLink.FClient Field

Pascal

```
FClient: TBaseButtonItem;
```

C++

```
TBaseButtonItem FClient;
```

Description

This is FClient, a member of class TButtonItemActionLink.

19.1.6.3 TButtonItemActionLink Methods

The methods of the TButtonItemActionLink class are listed here.

Protected Methods

	Name	Description
	AssignClient (see page 2448)	This is AssignClient, a member of class TButtonItemActionLink.
	IsCaptionLinked (see page 2448)	Indicates whether the Caption properties for the action and client are linked.
	IsHintLinked (see page 2449)	Indicates whether the Hint properties for the action and client are linked.
	IsImageIndexLinked (see page 2449)	Indicates whether the image index for the action and the client are linked.
	IsOnExecuteLinked (see page 2450)	This is IsOnExecuteLinked, a member of class TButtonItemActionLink.
	SetCaption (see page 2450)	Introduces an interface for setting the caption on a client.
	SetHint (see page 2450)	Introduces an interface for setting the hint for a client.

	SetImageIndex (see page 2451)	Introduces an interface for setting the image index for a client.
	SetOnExecute (see page 2452)	This is SetOnExecute, a member of class TButtonItemActionLink.

Public Methods

	Name	Description
	DoShowHint (see page 2448)	This is DoShowHint, a member of class TButtonItemActionLink.

19.1.6.3.1 TButtonItemActionLink.AssignClient Method

Pascal

```
procedure AssignClient(AClient: TObject); override;
```

C++

```
virtual __fastcall AssignClient(TObject * AClient);
```

Description

This is AssignClient, a member of class TButtonItemActionLink.

19.1.6.3.2 TButtonItemActionLink.DoShowHint Method

Pascal

```
function DoShowHint(var HintStr: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoShowHint(AnsiString HintStr);
```

Description

This is DoShowHint, a member of class TButtonItemActionLink.

19.1.6.3.3 TButtonItemActionLink.IsCaptionLinked Method

Indicates whether the Caption properties for the action and client are linked.

Pascal

```
function IsCaptionLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCaptionLinked();
```

Description

TActionLink descendants call IsCaptionLinked to determine whether they need to propagate changes to the Caption property of the action so that they are reflected by the client object. When IsCaptionLinked returns true, the action controls the Caption property: Changes to the Caption property of the action are reflected by changes to the corresponding property of the client object. When IsCaptionLinked returns false, the client object does not reflect the Caption property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink, IsCaptionLinked merely checks that the Action property specifies a descendant of TCustomAction (that is, it is an action that has a Caption property). Descendant classes override IsCaptionLinked to determine whether the Caption property is linked with the appropriate client property.

See Also

Caption (see page 308)

Caption (see page 5806)

SetCaption (see page 274)

Action (see page 2725)

19.1.6.3.4 TButtonItemActionLink.IsHintLinked Method

Indicates whether the Hint properties for the action and client are linked.

Pascal

```
function IsHintLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsHintLinked();
```

Description

TActionLink descendants call IsHintLinked to determine whether they need to propagate changes to the Hint property of the action so that they are reflected by the client object. When IsHintLinked returns true, the action controls the Hint property: Changes to the Hint property of the action are reflected by changes to the corresponding property of the client object. When IsHintLinked returns false, the client object does not reflect the Hint property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink, IsHintLinked merely checks that the Action property specifies a descendant of TCustomAction (that is, it is an action that has a Hint property). Descendant classes override IsHintLinked to determine whether the Hint property is linked with the appropriate client property.

See Also

Hint (see page 312)

Hint (see page 5827)

Hint (see page 16148)

SetHint (see page 278)

Action (see page 2725)

19.1.6.3.5 TButtonItemActionLink.IsImageIndexLinked Method

Indicates whether the image index for the action and the client are linked.

Pascal

```
function IsImageIndexLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsImageIndexLinked();
```

Description

TActionLink descendants call IsImageIndexLinked to determine whether they need to propagate changes to the ImageIndex property of the action so that they are reflected by the client object. When IsImageIndexLinked returns true, the action controls the ImageIndex property: Changes to the ImageIndex property of the action are reflected by changes to the corresponding property of the client object. When IsImageIndexLinked returns false, the client object does not reflect the ImageIndex property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink, IsImageIndexLinked merely checks that the Action property specifies a descendant of

TCustomAction (that is, it is an action that has an ImageIndex property). Descendant classes override IsImageIndexLinked to determine whether the ImageIndex property is linked with the appropriate client property.

See Also

ImageIndex (see page 313)

ImageIndex (see page 16148)

SetImageIndex (see page 278)

Action (see page 2725)

19.1.6.3.6 TButtonItemActionLink.IsOnExecuteLinked Method

Pascal

```
function IsOnExecuteLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsOnExecuteLinked();
```

Description

This is IsOnExecuteLinked, a member of class TButtonItemActionLink.

19.1.6.3.7 TButtonItemActionLink.SetCaption Method

Introduces an interface for setting the caption on a client.

Pascal

```
procedure SetCaption(const Value: string); override;
```

C++

```
virtual __fastcall SetCaption(const AnsiString Value);
```

Description

The associated action calls SetCaption when its Caption property changes so that the action link can propagate the new Caption value to the client object.

Value specifies the string to which the caption should be set.

As implemented in TActionLink, SetCaption does nothing. Descendant classes override SetCaption to set the client property that corresponds to the action's Caption property if the IsCaptionLinked method returns true.

See Also

Caption (see page 308)

Caption (see page 5806)

IsCaptionLinked (see page 268)

Action (see page 2725)

19.1.6.3.8 TButtonItemActionLink.SetHint Method

Introduces an interface for setting the hint for a client.

Pascal

```
procedure SetHint(const Value: string); override;
```

C++

```
virtual __fastcall SetHint(const AnsiString Value);
```

Description

The associated action calls SetHint when its Hint property changes so that the action link can propagate the new Hint value to the client object.

Value specifies the new value of the Hint property.

As implemented in TActionLink, SetHint does nothing. Descendant classes override SetHint to set the client property that corresponds to the action's Hint property if the IsHintLinked method returns true.

See Also

Hint (see page 312)

Hint (see page 5827)

Hint (see page 16148)

Action (see page 2725)

IsHintLinked (see page 271)

19.1.6.3.9 TButtonItemActionLink.SetImageIndex Method

Introduces an interface for setting the image index for a client.

Pascal

```
procedure SetImageIndex(Value: Integer); override;
```

C++

```
virtual __fastcall SetImageIndex(int Value);
```

Description

The associated action calls SetImageIndex when its ImageIndex property changes so that the action link can propagate the new ImageIndex value to the client object.

Value specifies the new value of the ImageIndex property.

As implemented in TActionLink, SetImageIndex does nothing. Descendant classes override SetImageIndex to set the client property that corresponds to the action's ImageIndex property if the IsImageIndexLinked method returns true.

See Also

ImageIndex (see page 313)

ImageIndex (see page 16148)

Action (see page 2725)

IsImageIndexLinked (see page 272)

19.1.6.3.10 TButtonItemActionLink.SetOnExecute Method

Pascal

```
procedure SetOnExecute(Value: TNotifyEvent); override;
```

C++

```
virtual __fastcall SetOnExecute(TNotifyEvent Value);
```

Description

This is SetOnExecute, a member of class TButtonItemActionLink.

19.1.7 TCategoryButtons Class

Defines a group of categorized buttons.

Class Hierarchy

Pascal

```
TCategoryButtons = class(TCustomControl);
```

C++

```
class TCategoryButtons : public TCustomControl;
```

File

CategoryButtons

Description

TCategoryButtons is used for creating groups of buttons that can be categorized, as on a tool palette. It provides more flexibility than a tool bar.

The TCategoryButtons object can contain TButtonCategory (see page 2429) objects, each representing a set of buttons in the same category. Each TButtonCategory (see page 2429) object can contain TButtonItem (see page 2441) objects. The TButtonCategory (see page 2429) and TButtonItem (see page 2441) objects can be dynamically reordered. For instance, a button may be dragged or copied from one category to another. Categories (see page 2486) can be expanded to show all buttons or collapsed to hide them.

See Also

TButtonCategory (see page 2429)

TButtonItem (see page 2441)

19.1.7.1 TCategoryButtons Members

The following tables list the members exposed by TCategoryButtons.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
V	BeginAutoDrag (see page 2466)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.

	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	CMFontchanged (see page 2466)	This is CMFontchanged, a member of class TCategoryButtons.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 2466)	Creates underlying screen object.
	CreateParams (see page 2467)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DoBeginDrag (see page 2467)	This is DoBeginDrag, a member of class TCategoryButtons.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoCopyButton (see page 2468)	This is DoCopyButton, a member of class TCategoryButtons.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 2468)	OnEndDrag event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoHotButton (see page 2468)	This is DoHotButton, a member of class TCategoryButtons.
	DoItemClicked (see page 2469)	This is DoItemClicked, a member of class TCategoryButtons.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 2469)	OnMouseWheelDown event dispatcher.
	DoMouseWheelUp (see page 2469)	OnMouseWheelUp event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoReorderButton (see page 2470)	This is DoReorderButton, a member of class TCategoryButtons.
	DoReorderCategory (see page 2470)	This is DoReorderCategory, a member of class TCategoryButtons.

	DoSelectedItemChanged (see page 2470)	This is DoSelectedItemChanged, a member of class TCategoryButtons.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
V	DoStartDrag (see page 2470)	OnStartDrag event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 2471)	OnDragOver event dispatcher.
V	DrawButton (see page 2472)	This is DrawButton, a member of class TCategoryButtons.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetAllowReorder (see page 2472)	This is GetAllowReorder, a member of class TCategoryButtons.
V	GetButtonCategoriesClass (see page 2473)	This is GetButtonCategoriesClass, a member of class TCategoryButtons.
V	GetButtonCategoryClass (see page 2473)	This is GetButtonCategoryClass, a member of class TCategoryButtons.
V	GetButtonItemClass (see page 2473)	This is GetButtonItemClass, a member of class TCategoryButtons.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 2474)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetScrollPos (see page 2474)	This is GetScrollPos, a member of class TCategoryButtons.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.

	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 2475)	OnMouseDown event dispatcher.
V	MouseMove (see page 2475)	Respond to mouse moving over control area..
V	MouseUp (see page 2476)	OnMouseUp event dispatcher.
V	Notification (see page 2476)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 2477)	Renders the image of a custom control.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
V	Resize (see page 2477)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	ScrollRectIntoView (see page 2478)	This is ScrollRectIntoView, a member of class TCategoryButtons.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.

	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetScrollPos (see page 2478)	This is SetScrollPos, a member of class TCategoryButtons.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.

	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2465)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 2464)	Creates instance of TCategoryButtons (see page 2452).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2464)	Destroys instance of TCategoryButtons (see page 2452).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.

	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
V	DragDrop (see page 2471)	Handles user dragging button.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GenerateDragImage (see page 2472)	Allows drag images to be generated from outside the control.
	GetButtonAt (see page 2472)	Retrieves button at specified position.
	GetButtonRect (see page 2473)	Retrieves specified button's bounding rectangle.
	GetCategoryAt (see page 2473)	Retrieves category at specified position.
	GetCategoryRect (see page 2474)	Retrieves specified button's bounding rectangle.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTargetAt (see page 2474)	Gives you the target insertion point in the given group. TargetButton may be nil while the TargetCategory is not, or both may be nil.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveInsertionPoints (see page 2477)	Removes insertion points in the container.
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollIntoView (see page 2478)	This is the overview for the ScrollIntoView method overload.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetInsertionButton (see page 2478)	Sets the current insertion mark.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateAllButtons (see page 2479)	Repaints or refreshes all buttons.
	UpdateButton (see page 2479)	Repaints or refreshes the button.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.

	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragButton (see page 2490)	Button being dragged.
	DragCategory (see page 2490)	Category being dragged.
R	DragImageList (see page 2491)	List of images used during drag or dock operations.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	FocusedItem (see page 2494)	Specifies the button that currently has focus.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.

	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelectedItem (see page 2502)	Returns the selected button.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAfterDrawButton (see page 2508)	Fired after a button is drawn
	OnAlignInsertBefore (see page 2508)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2508)	Occurs when an object with custom alignment is aligned.
	OnBeforeDrawButton (see page 2509)	Fired before a button is drawn
	OnButtonClicked (see page 2509)	Fired when a button is clicked, if OnClick (see page 2510) event is not present
	OnCategoryCollapase (see page 2509)	Fired when a category is collapsed
	OnClick (see page 2510)	Occurs when mouse button goes up.
	OnContextPopup (see page 2510)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCopyButton (see page 2511)	Fired when a button is copied
	OnDockDrop (see page 2511)	Occurs when another control is docked to the control.
	OnDockOver (see page 2513)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 2515)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2517)	Occurs when the user drags an object over a control.
	OnDrawButton (see page 2519)	Fired when a button needs to be redrawn; it prevents default drawing.
	OnDrawIcon (see page 2520)	Fired when an icon needs to be redrawn; it prevents default drawing.
	OnDrawText (see page 2520)	Fired when a caption needs to be redrawn; it prevents default drawing.
	OnEndDock (see page 2520)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2520)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2521)	Occurs when a control receives the input focus.
	OnExit (see page 2524)	Occurs when the input focus shifts away from one control to another.
	OnGetHint (see page 2525)	Fired when a hint string is being retrieved
	OnHotButton (see page 2526)	Fired when the "hot" button changes
	OnKeyDown (see page 2526)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2529)	Occurs when a key is pressed.
	OnKeyUp (see page 2530)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2531)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2532)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2535)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2535)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2535)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2537)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 2540)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 2540)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 2540)	Occurs when the mouse wheel is rotated upward.
	OnReorderButton (see page 2541)	Fired when buttons are reordered
	OnReorderCategory (see page 2541)	Fired when categories are reordered
	OnSelectedItemChange (see page 2541)	Fired when the selected item changes

	OnStartDock (see page 2541)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2542)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 2480)	Determines how control aligns within its parent control.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 2480)	Specifies how the control is anchored to its parent.
	BackgroundGradientColor (see page 2481)	Specifies the color for a gradient fill in the background of a category.
	BackgroundGradientDirection (see page 2481)	Direction of background gradient fill.
	BevelEdges (see page 2481)	Specifies which edges of the control are beveled.
	BevelInner (see page 2482)	Determines the style of the inner bevel of a control.
	BevelKind (see page 2482)	Specifies the control's bevel style.
	BevelOuter (see page 2483)	Determines the style of the outer bevel of a control.
	BevelWidth (see page 2483)	Width (see page 2506) of outer and inner bevels.
	BorderStyle (see page 2484)	Determines whether control has single line border or none.
	BorderWidth (see page 2484)	Width (see page 2506) of border.
	ButtonFlow (see page 2484)	Specifies the direction of buttons in a category.
	ButtonHeight (see page 2485)	Height (see page 2496) of buttons in a category.
	ButtonOptions (see page 2485)	Specifies options for button display and behavior.
	ButtonWidth (see page 2486)	Width (see page 2506) of buttons in a category.
	Categories (see page 2486)	Specifies the categories represented in the category buttons object.
	Color (see page 2486)	Category buttons background color
	Cursor (see page 2486)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DockSite (see page 2489)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 2491)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2492)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2492)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2494)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2494)	Controls the attributes of text written on or in the control.
	GradientDirection (see page 2495)	Specifies the direction of gradient fill for the category object.
	Height (see page 2496)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HotButtonColor (see page 2497)	Specifies the color of a hot button, if necessary.
	Images (see page 2497)	List of button images.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	PopupMenu (see page 2497)	Identifies the pop-up menu associated with the control.
	RegularButtonColor (see page 2502)	Specifies the color of a button in the normal, unselected state.
	SelectedButtonColor (see page 2502)	Specifies the color of the button when a user selects it.
	ShowHint (see page 2502)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2504)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2504)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.

	Visible (see page 2505)	Determines whether the component appears onscreen.
	Width (see page 2506)	Specifies the horizontal size of the control or form in pixels.

19.1.7.2 TCategoryButtons.Create Constructor

Creates instance of TCategoryButtons (see page 2452).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCategoryButtons(TComponent * AOwner);
```

Description

This method creates an instance of TCategoryButtons (see page 2452). Create sets default values for a variety of attributes, including width, height, control style, button options, and colors.

See Also

Destroy (see page 2464)

19.1.7.3 TCategoryButtons.Destroy Destructor

Destroys instance of TCategoryButtons (see page 2452).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCategoryButtons();
```

Description

This method destroys an instance of TCategoryButtons (see page 2452), freeing all it resources.

See Also

Create (see page 2464)

19.1.7.4 TCategoryButtons Methods

The methods of the TCategoryButtons class are listed here.

Protected Methods

	Name	Description
	BeginAutoDrag (see page 2466)	Begin Automatic Drag.
	CMFontchanged (see page 2466)	This is CMFontchanged, a member of class TCategoryButtons.
	CreateHandle (see page 2466)	Creates underlying screen object.
	CreateParams (see page 2467)	Initializes a window-creation parameter data structure.
	DoBeginDrag (see page 2467)	This is DoBeginDrag, a member of class TCategoryButtons.
	DoCopyButton (see page 2468)	This is DoCopyButton, a member of class TCategoryButtons.
	DoEndDrag (see page 2468)	OnEndDrag event dispatcher.
	DoHotButton (see page 2468)	This is DoHotButton, a member of class TCategoryButtons.
	DoItemClicked (see page 2469)	This is DoItemClicked, a member of class TCategoryButtons.

	DoMouseWheelDown (see page 2469)	OnMouseWheelDown event dispatcher.
	DoMouseWheelUp (see page 2469)	OnMouseWheelUp event dispatcher.
	DoReorderButton (see page 2470)	This is DoReorderButton, a member of class TCategoryButtons.
	DoReorderCategory (see page 2470)	This is DoReorderCategory, a member of class TCategoryButtons.
	DoSelectedItemChanged (see page 2470)	This is DoSelectedItemChanged, a member of class TCategoryButtons.
	DoStartDrag (see page 2470)	OnStartDrag event dispatcher.
	DragOver (see page 2471)	OnDragOver event dispatcher.
	DrawButton (see page 2472)	This is DrawButton, a member of class TCategoryButtons.
	GetAllowReorder (see page 2472)	This is GetAllowReorder, a member of class TCategoryButtons.
	GetButtonCategoriesClass (see page 2473)	This is GetButtonCategoriesClass, a member of class TCategoryButtons.
	GetButtonCategoryClass (see page 2473)	This is GetButtonCategoryClass, a member of class TCategoryButtons.
	GetButtonItemClass (see page 2473)	This is GetButtonItemClass, a member of class TCategoryButtons.
	GetDragImages (see page 2474)	Returns the control's image list containing the image to be displayed while dragging.
	GetScrollPos (see page 2474)	This is GetScrollPos, a member of class TCategoryButtons.
	MouseDown (see page 2475)	OnMouseDown event dispatcher.
	MouseMove (see page 2475)	Respond to mouse moving over control area..
	MouseUp (see page 2476)	OnMouseUp event dispatcher.
	Notification (see page 2476)	Responds to notifications that components are being created or destroyed.
	Paint (see page 2477)	Renders the image of a custom control.
	Resize (see page 2477)	Respond to control resize.
	ScrollRectIntoView (see page 2478)	This is ScrollRectIntoView, a member of class TCategoryButtons.
	SetScrollPos (see page 2478)	This is SetScrollPos, a member of class TCategoryButtons.

Public Methods

	Name	Description
	Assign (see page 2465)	Copies the contents of another, similar object.
	DragDrop (see page 2471)	Handles user dragging button.
	GenerateDragImage (see page 2472)	Allows drag images to be generated from outside the control.
	GetButtonAt (see page 2472)	Retrieves button at specified position.
	GetButtonRect (see page 2473)	Retrieves specified button's bounding rectangle.
	GetCategoryAt (see page 2473)	Retrieves category at specified position.
	GetCategoryRect (see page 2474)	Retrieves specified button's bounding rectangle.
	GetTargetAt (see page 2474)	Gives you the target insertion point in the given group. TargetButton may be nil while the TargetCategory is not, or both may be nil.
	RemoveInsertionPoints (see page 2477)	Removes insertion points in the container.
	ScrollIntoView (see page 2478)	This is the overview for the ScrollIntoView method overload.
	SetInsertionButton (see page 2478)	Sets the current insertion mark.
	UpdateAllButtons (see page 2479)	Repaints or refreshes all buttons.
	UpdateButton (see page 2479)	Repaints or refreshes the button.

19.1.7.4.1 TCategoryButtons.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. In particular, copy the Categories (see page 2486), ButtonHeight (see page 2485), ButtonWidth (see page 2486), ButtonOptions (see page 2485) properties.

See Also

ButtonHeight (see page 2485)

ButtonOptions (see page 2485)

ButtonWidth (see page 2486)

Categories (see page 2486)

Assign (see page 2930)

19.1.7.4.2 TCategoryButtons.BeginAutoDrag Method

Begin Automatic Drag.

Pascal

```
procedure BeginAutoDrag; override;
```

C++

```
virtual __fastcall BeginAutoDrag();
```

Description

BeginAutoDrag is called automatically when a user drags a control whose DragMode is dmAutomatic. Do not call BeginAutoDrag in application code.

As implemented in TControl, BeginAutoDrag calls BeginDrag, passing in the global Mouse variable's DragImmediate and DragThreshold properties.

See Also

DragKind (see page 5819)

DragMode (see page 5819)

EndDrag (see page 5757)

Mouse (see page 6321)

DragImmediate (see page 6100)

DragThreshold (see page 6101)

Mouse (see page 6321)

19.1.7.4.3 TCategoryButtons.CMFontchanged Method

Pascal

```
procedure CMFontchanged(var Message: TMessage); message CM_FONTCHANGED;
```

C++

```
__fastcall CMFontchanged(TMessage Message);
```

Description

This is CMFontchanged, a member of class TCategoryButtons.

19.1.7.4.4 TCategoryButtons.CreateHandle Method

Creates underlying screen object.

Pascal

```
procedure CreateHandle; override;
```

C++

```
virtual __fastcall CreateHandle();
```

Description

CreateHandle creates the Microsoft Windows screen object that underlies the control. if it does not already exist. The new window handle becomes the value of the Handle property.

CreateHandle creates or locates the parent control's window before it creates a window for the control itself.

Note: If the Parent property is nil (Delphi) or NULL (C++), CreateHandle raises an exception.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

Handle (see page 6207)

Parent (see page 5832)

19.1.7.4.5 TCategoryButtons.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

19.1.7.4.6 TCategoryButtons.DoBeginDrag Method

Pascal

```
procedure DoBeginDrag(Immediate: Boolean; Threshold: Integer); virtual;
```

C++

```
virtual __fastcall DoBeginDrag(Boolean Immediate, int Threshold);
```

Description

This is DoBeginDrag, a member of class TCategoryButtons.

19.1.7.4.7 TCategoryButtons.DoCopyButton Method

Pascal

```
procedure DoCopyButton(const Button: TButtonItem; const TargetCategory: TButtonCategory; const
TargetButton: TButtonItem); dynamic;
```

C++

```
__fastcall DoCopyButton(const TButtonItem Button, const TButtonCategory TargetCategory, const
TButtonItem TargetButton);
```

Description

This is DoCopyButton, a member of class TCategoryButtons.

19.1.7.4.8 TCategoryButtons.DoEndDrag Method

OnEndDrag event dispatcher.

Pascal

```
procedure DoEndDrag(Target: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DoEndDrag(TObject * Target, int X, int Y);
```

Description

DoEndDrag calls any event handler attached to the OnEndDrag event. When creating a component that descends from TControl, override DoEndDrag to provide other responses in addition to the inherited event-handler call when the user drops an object or cancels a drag-and-drop operation.

See Also

DoStartDrag (see page 5753)

OnEndDrag (see page 5860)

19.1.7.4.9 TCategoryButtons.DoHotButton Method

Pascal

```
procedure DoHotButton; dynamic;
```

C++

```
__fastcall DoHotButton();
```

Description

This is DoHotButton, a member of class TCategoryButtons.

19.1.7.4.10 TCategoryButtons.DoItemClicked Method

Pascal

```
procedure DoItemClicked(const Button: TButtonItem); dynamic;
```

C++

```
__fastcall DoItemClicked(const TButtonItem Button);
```

Description

This is DoItemClicked, a member of class TCategoryButtons.

19.1.7.4.11 TCategoryButtons.DoMouseWheelDown Method

OnMouseWheelDown event dispatcher.

Pascal

```
function DoMouseWheelDown(Shift: TShiftState; MousePos: TPoint): Boolean; override;
```

C++

```
virtual __fastcall Boolean DoMouseWheelDown(TShiftState Shift, TPoint MousePos);
```

Description

The DoMouseWheel method calls DoMouseWheelDown to generate an OnMouseWheelDown event.

The Shift parameter indicates the state of the Shift, Alt, and Ctrl keys.

The MousePos parameter indicates the position of the mouse pointer.

DoMouseWheelDown returns true if an event handler indicates that it handled the message, and false otherwise.

Override DoMouseWheelDown to perform class-specific tasks in addition to calling the event handler when the mouse wheel is rotated downward.

See Also

OnMouseWheelDown (see page 5869)

DoMouseWheel (see page 5751)

19.1.7.4.12 TCategoryButtons.DoMouseWheelUp Method

OnMouseWheelUp event dispatcher.

Pascal

```
function DoMouseWheelUp(Shift: TShiftState; MousePos: TPoint): Boolean; override;
```

C++

```
virtual __fastcall Boolean DoMouseWheelUp(TShiftState Shift, TPoint MousePos);
```

Description

The DoMouseWheel method calls DoMouseWheelUp to generate an OnMouseWheelUp event.

The Shift parameter indicates the state of the Shift, Alt, and Ctrl keys.

The MousePos parameter indicates the position of the mouse pointer.

DoMouseWheelUp returns true if an event handler indicates that it handled the message, and false otherwise.

Override DoMouseWheelUp to perform class-specific tasks in addition to calling the event handler when the mouse wheel is rotated upward.

See Also

OnMouseWheelUp (see page 5870)

DoMouseWheel (see page 5751)

19.1.7.4.13 TCategoryButtons.DoReorderButton Method

Pascal

```
procedure DoReorderButton(const Button: TButtonItem; const TargetCategory: TButtonCategory;
const TargetButton: TButtonItem); dynamic;
```

C++

```
__fastcall DoReorderButton(const TButtonItem Button, const TButtonCategory TargetCategory,
const TButtonItem TargetButton);
```

Description

This is DoReorderButton, a member of class TCategoryButtons.

19.1.7.4.14 TCategoryButtons.DoReorderCategory Method

Pascal

```
procedure DoReorderCategory(const SourceCategory: TButtonCategory; const TargetCategory:
TButtonCategory); dynamic;
```

C++

```
__fastcall DoReorderCategory(const TButtonCategory SourceCategory, const TButtonCategory
TargetCategory);
```

Description

This is DoReorderCategory, a member of class TCategoryButtons.

19.1.7.4.15 TCategoryButtons.DoSelectedItemChanged Method

Pascal

```
procedure DoSelectedItemChanged(const Button: TButtonItem); dynamic;
```

C++

```
__fastcall DoSelectedItemChanged(const TButtonItem Button);
```

Description

This is DoSelectedItemChanged, a member of class TCategoryButtons.

19.1.7.4.16 TCategoryButtons.DoStartDrag Method

OnStartDrag event dispatcher.

Pascal

```
procedure DoStartDrag(var DragObject: TDragObject); override;
```

C++

```
virtual __fastcall DoStartDrag(TDragObject DragObject);
```

Description

DoStartDrag calls any event handler attached to the OnStartDrag event. When creating a component that descends from TControl, override DoStartDrag to provide other responses in addition to the inherited event-handler call when the user starts a drag-and-drop operation.

See Also

DoEndDrag (see page 5751)

OnStartDrag (see page 5871)

19.1.7.4.17 TCategoryButtons.DragDrop Method

Handles user dragging button.

Pascal

```
procedure DragDrop(Source: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DragDrop(TObject * Source, int X, int Y);
```

Description

The **Source** parameter is the object that was dropped onto the button group. The **X** and Y parameters are the mouse coordinates where the object was dropped.

This method handles reordering TButtonCategory (see page 2429) and TButtonItem (see page 2441) objects in a TCategoryButtons (see page 2452) object in response to user drag drop operations. Users can drag TButtonCategory (see page 2429) objects to reorder them. They can also drag TCategoryButtons (see page 2452) objects to reorder them or move them to another category. Pressing `Ctrl` while dragging copies the button. This method calls RemoveInsertionPoints (see page 2477) to remove insertion points.

See Also

TButtonCategory (see page 2429)

TButtonItem (see page 2441)

RemoveInsertionPoints (see page 2477)

19.1.7.4.18 TCategoryButtons.DragOver Method

OnDragOver event dispatcher.

Pascal

```
procedure DragOver(Source: TObject; X: Integer; Y: Integer; State: TDragState; var Accept:
Boolean); override;
```

C++

```
virtual __fastcall DragOver(TObject * Source, int X, int Y, TDragState State, Boolean Accept);
```

Description

Override DragOver to add additional code that executes before the OnDragOver event handler is called.

DragOver sets the Accept parameter to true to indicate that the user can drop the dragged object on the control. It sets Accept to

false to indicate that the user cannot drop the dragged object on the control.

The Source parameter is the object being dragged. The State parameter indicates how the dragged object is moving in relation to the control. X and Y indicate the current position of the mouse.

See Also

DragDrop (see page 5754)

OnDragOver (see page 5857)

19.1.7.4.19 TCategoryButtons.DrawButton Method

Pascal

```
procedure DrawButton(const Button: TButtonItem; Canvas: TCanvas; Rect: TRect; State:
TButtonDrawState); virtual;
```

C++

```
virtual __fastcall DrawButton(const TButtonItem Button, TCanvas Canvas, TRect Rect,
TButtonDrawState State);
```

Description

This is DrawButton, a member of class TCategoryButtons.

19.1.7.4.20 TCategoryButtons.GenerateDragImage Method

Allows drag images to be generated from outside the control.

Pascal

```
procedure GenerateDragImage;
```

C++

```
__fastcall GenerateDragImage();
```

Description

Use the GenerateDragImage method to allow drag images to be generated from outside the control.

19.1.7.4.21 TCategoryButtons.GetAllowReorder Method

Pascal

```
function GetAllowReorder: Boolean; dynamic;
```

C++

```
__fastcall Boolean GetAllowReorder();
```

Description

This is GetAllowReorder, a member of class TCategoryButtons.

19.1.7.4.22 TCategoryButtons.GetButtonAt Method

Retrieves button at specified position.

Pascal

```
function GetButtonAt(X: Integer; Y: Integer; Category: TButtonCategory = nil): TButtonItem;
```

C++

```
__fastcall TButtonItem GetButtonAt(int X, int Y, TButtonCategory Category = nil);
```

19.1.7.4.23 TCategoryButtons.GetButtonCategoriesClass Method

Pascal

```
function GetButtonCategoriesClass: TButtonCategoriesClass; virtual;
```

C++

```
virtual __fastcall TButtonCategoriesClass GetButtonCategoriesClass();
```

Description

This is GetButtonCategoriesClass, a member of class TCategoryButtons.

19.1.7.4.24 TCategoryButtons.GetButtonCategoryClass Method

Pascal

```
function GetButtonCategoryClass: TButtonCategoryClass; virtual;
```

C++

```
virtual __fastcall TButtonCategoryClass GetButtonCategoryClass();
```

Description

This is GetButtonCategoryClass, a member of class TCategoryButtons.

19.1.7.4.25 TCategoryButtons.GetButtonItemClass Method

Pascal

```
function GetButtonItemClass: TButtonItemClass; virtual;
```

C++

```
virtual __fastcall TButtonItemClass GetButtonItemClass();
```

Description

This is GetButtonItemClass, a member of class TCategoryButtons.

19.1.7.4.26 TCategoryButtons.GetButtonRect Method

Retrieves specified button's bounding rectangle.

Pascal

```
function GetButtonRect(const Button: TButtonItem): TRect;
```

C++

```
__fastcall TRect GetButtonRect(const TButtonItem Button);
```

19.1.7.4.27 TCategoryButtons.GetCategoryAt Method

Retrieves category at specified position.

Pascal

```
function GetCategoryAt(X: Integer; Y: Integer): TButtonCategory;
```

C++

```
__fastcall TButtonCategory GetCategoryAt(int X, int Y);
```

19.1.7.4.28 TCategoryButtons.GetCategoryRect Method

Retrieves specified button's bounding rectangle.

Pascal

```
function GetCategoryRect(const Category: TButtonCategory): TRect;
```

C++

```
__fastcall TRect GetCategoryRect(const TButtonCategory Category);
```

19.1.7.4.29 TCategoryButtons.GetDragImages Method

Returns the control's image list containing the image to be displayed while dragging.

Pascal

```
function GetDragImages: TDragImageList; override;
```

C++

```
virtual __fastcall TDragImageList GetDragImages();
```

Description

Override GetDragImages to provide a custom image list for the control. As implemented in TControl, GetDragImages returns nil (Delphi) or NULL (C++).

See Also

DragCursor (see page 5818)

OnStartDrag (see page 5871)

19.1.7.4.30 TCategoryButtons.GetScrollPos Method

Pascal

```
function GetScrollPos: Integer;
```

C++

```
__fastcall int GetScrollPos();
```

Description

This is GetScrollPos, a member of class TCategoryButtons.

19.1.7.4.31 TCategoryButtons.GetTargetAt Method

Gives you the target insertion point in the given group. TargetButton may be nil while the TargetCategory is not, or both may be nil.

Pascal

```
procedure GetTargetAt(X: Integer; Y: Integer; var TargetButton: TButtonItem; var
TargetCategory: TButtonCategory);
```

C++

```
__fastcall GetTargetAt(int X, int Y, TButtonItem TargetButton, TButtonCategory TargetCategory);
```

19.1.7.4.32 TCategoryButtons.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove (see page 5773)

MouseUp (see page 5774)

OnMouseDown (see page 5862)

19.1.7.4.33 TCategoryButtons.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown (see page 5773)

MouseUp (see page 5774)

OnMouseMove (see page 5865)

19.1.7.4.34 TCategoryButtons.MouseUp Method

OnMouseUp event dispatcher.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user releases a previously pressed mouse button while the cursor's hotspot is over the control.

A control calls MouseUp in response to any of the Windows mouse-up messages (WM_LBUTTONUP, WM_MBUTTONUP, WM_RBUTTONUP), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

See Also

MouseDown (see page 5773)

MouseMove (see page 5773)

OnMouseUp (see page 5867)

19.1.7.4.35 TCategoryButtons.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

19.1.7.4.36 TCategoryButtons.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas (see page 5921)

PaintWindow (see page 5920)

GetDeviceContext (see page 6155)

ClipRect (see page 14486)

19.1.7.4.37 TCategoryButtons.RemoveInsertionPoints Method

Removes insertion points in the container.

Pascal

```
procedure RemoveInsertionPoints;
```

C++

```
__fastcall RemoveInsertionPoints();
```

19.1.7.4.38 TCategoryButtons.Resize Method

Respond to control resize.

Pascal

```
procedure Resize; override;
```

C++

```
virtual __fastcall Resize();
```

Description

Resize is called automatically immediately after the control's dimensions change. As implemented in TControl, Resize calls the control's OnResize event handler if it is assigned.

See Also

OnResize (see page 5870)

19.1.7.4.39 ScrollIntoView Method

This is the overview for the ScrollIntoView method overload.

Overload List

	Name	Description
	TCategoryButtons.ScrollIntoView (TButtonCategory) (see page 2478)	Scrolls a button to make it visible in the container.
	TCategoryButtons.ScrollIntoView (TButtonItem) (see page 2478)	Scrolls a button to make it visible in the container.

19.1.7.4.39.1 TCategoryButtons.ScrollIntoView Method (TButtonCategory)

Scrolls a button to make it visible in the container.

Pascal

```
procedure ScrollIntoView(const Button: TButtonItem); overload;
procedure ScrollIntoView(const Category: TButtonCategory); overload;
```

C++

```
__fastcall ScrollIntoView(const TButtonItem Button);
__fastcall ScrollIntoView(const TButtonCategory Category);
```

19.1.7.4.40 TCategoryButtons.ScrollRectIntoView Method

Pascal

```
procedure ScrollRectIntoView(const Rect: TRect);
```

C++

```
__fastcall ScrollRectIntoView(const TRect Rect);
```

Description

This is ScrollRectIntoView, a member of class TCategoryButtons.

19.1.7.4.41 TCategoryButtons.SetInsertionButton Method

Sets the current insertion mark.

Pascal

```
procedure SetInsertionButton(InsertionButton: TButtonItem; InsertionCategory: TButtonCategory);
```

C++

```
__fastcall SetInsertionButton(TButtonItem InsertionButton, TButtonCategory InsertionCategory);
```

19.1.7.4.42 TCategoryButtons.SetScrollPos Method

Pascal

```
procedure SetScrollPos(const Value: Integer);
```

C++

```
__fastcall SetScrollPos(const int Value);
```

Description

This is SetScrollPos, a member of class TCategoryButtons.

19.1.7.4.43 TCategoryButtons.UpdateAllButtons Method

Repaints or refreshes all buttons.

Pascal

```
procedure UpdateAllButtons;
```

C++

```
__fastcall UpdateAllButtons();
```

19.1.7.4.44 TCategoryButtons.UpdateButton Method

Repaints or refreshes the button.

Pascal

```
procedure UpdateButton(const Button: TButtonItem);
```

C++

```
__fastcall UpdateButton(const TButtonItem Button);
```

19.1.7.5 TCategoryButtons Properties

The properties of the TCategoryButtons class are listed here.

Public Properties

	Name	Description
	DragButton (see page 2490)	Button being dragged.
	DragCategory (see page 2490)	Category being dragged.
R	DragImageList (see page 2491)	List of images used during drag or dock operations.
	FocusedItem (see page 2494)	Specifies the button that currently has focus.
	SelectedItem (see page 2502)	Returns the selected button.

Published Properties

	Name	Description
	Align (see page 2480)	Determines how control aligns within its parent control.
	Anchors (see page 2480)	Specifies how the control is anchored to its parent.
	BackgroundGradientColor (see page 2481)	Specifies the color for a gradient fill in the background of a category.
	BackgroundGradientDirection (see page 2481)	Direction of background gradient fill.
	BevelEdges (see page 2481)	Specifies which edges of the control are beveled.
	BevelInner (see page 2482)	Determines the style of the inner bevel of a control.
	BevelKind (see page 2482)	Specifies the control's bevel style.
	BevelOuter (see page 2483)	Determines the style of the outer bevel of a control.
	BevelWidth (see page 2483)	Width (see page 2506) of outer and inner bevels.
	BorderStyle (see page 2484)	Determines whether control has single line border or none.
	BorderWidth (see page 2484)	Width (see page 2506) of border.
	ButtonFlow (see page 2484)	Specifies the direction of buttons in a category.
	ButtonHeight (see page 2485)	Height (see page 2496) of buttons in a category.
	ButtonOptions (see page 2485)	Specifies options for button display and behavior.
	ButtonWidth (see page 2486)	Width (see page 2506) of buttons in a category.
	Categories (see page 2486)	Specifies the categories represented in the category buttons object.

	Color (see page 2486)	Category buttons background color
	Cursor (see page 2486)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DockSite (see page 2489)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 2491)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2492)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2492)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2494)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2494)	Controls the attributes of text written on or in the control.
	GradientDirection (see page 2495)	Specifies the direction of gradient fill for the category object.
	Height (see page 2496)	Specifies the vertical size of the control in pixels.
	HotButtonColor (see page 2497)	Specifies the color of a hot button, if necessary.
	Images (see page 2497)	List of button images.
	PopupMenu (see page 2497)	Identifies the pop-up menu associated with the control.
	RegularButtonColor (see page 2502)	Specifies the color of a button in the normal, unselected state.
	SelectedButtonColor (see page 2502)	Specifies the color of the button when a user selects it.
	ShowHint (see page 2502)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2504)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2504)	Determines if the user can tab to a control.
	Visible (see page 2505)	Determines whether the component appears onscreen.
	Width (see page 2506)	Specifies the horizontal size of the control or form in pixels.

19.1.7.5.1 TCategoryButtons.Align Property

Determines how control aligns within its parent control.

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

TCategoryButtons (see page 2452) uses the Align property inherited from TControl.

See Also

TControl (see page 5724)

Align (see page 5798)

19.1.7.5.2 TCategoryButtons.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

TCategoryButtons (see page 2452) uses the Anchors property inherited from TControl.

See Also

TControl (see page 5724)

Anchors (see page 5802)

19.1.7.5.3 TCategoryButtons.BackgroundGradientColor Property

Specifies the color for a gradient fill in the background of a category.

Pascal

```
property BackgroundGradientColor: TColor;
```

C++

```
__property TColor BackgroundGradientColor;
```

19.1.7.5.4 TCategoryButtons.BackgroundGradientDirection Property

Direction of background gradient fill.

Pascal

```
property BackgroundGradientDirection: TGradientDirection;
```

C++

```
__property TGradientDirection BackgroundGradientDirection;
```

Description

The following values are available:

Parameter	Description
gdHorizontal	The gradient appears horizontally on the background of the container. Default.
gdVertical	The gradient appears vertically on the background of the container.

19.1.7.5.5 TCategoryButtons.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner (see page 2482), BevelOuter (see page 2483), BevelWidth (see page 2483) and BevelKind (see page 2482) properties determine the appearance of the specified edges.

See Also

BevelInner (see page 2482)

BevelKind (see page 2482)

BevelOuter (see page 2483)

BevelWidth (see page 2483)

19.1.7.5.6 TCategoryButtons.BevelInner Property

Determines the style of the inner bevel of a control.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to provide the control with a three-dimensional beveled look. A control has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel BevelOuter (see page 2483), separated by the number of pixels specified in the BorderWidth (see page 2484) property. The width of the bevels, if they are drawn, is the BevelWidth (see page 2483) property.

See Also

BorderWidth (see page 2484)

BevelEdges (see page 2481)

BevelKind (see page 2482)

BevelOuter (see page 2483)

BevelWidth (see page 2483)

19.1.7.5.7 TCategoryButtons.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth (see page 2483) and the cut of the bevel specified by BevelInner (see page 2482) or BevelOuter (see page 2483), can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 2481)

BevelInner (see page 2482)

BevelOuter (see page 2483)

BevelWidth (see page 2483)

19.1.7.5.8 TCategoryButtons.BevelOuter Property

Determines the style of the outer bevel of a control.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelOuter to provide the control with a three-dimensional beveled look. A control has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel BevelOuter, separated by the number of pixels specified in the BorderWidth (see page 2484) property. The width of the bevels, if they are drawn, is the BevelWidth (see page 2483) property.

See Also

BorderWidth (see page 2484)

BevelEdges (see page 2481)

BevelInner (see page 2482)

BevelKind (see page 2482)

BevelWidth (see page 2483)

19.1.7.5.9 TCategoryButtons.BevelWidth Property

Width (see page 2506) of outer and inner bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelInner (see page 2482) and BeveOuter to provide the control with a three-dimensional beveled look. A control has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel BevelOuter (see page 2483), separated by the number of pixels specified in the BorderWidth (see page 2484) property. The width of the bevels, if they are drawn, is the BevelWidth property.

See Also

BorderWidth (see page 2484)

BevelEdges (see page 2481)

BevelInner (see page 2482)

BevelKind (see page 2482)

BevelOuter (see page 2483)

19.1.7.5.10 TCategoryButtons.BorderStyle Property

Determines whether control has single line border or none.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the control stands out. BorderStyle can have a value of either **bsSingle** or **bsNone**. If BorderStyle is **bsSingle**, the control has a single-line border around the client area. If BorderStyle is **bsNone**, there is no border.

See Also

BorderWidth (see page 2484)

19.1.7.5.11 TCategoryButtons.BorderWidth Property

Width (see page 2506) of border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

A control has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel BevelOuter (see page 2483), separated by the number of pixels specified in the BorderWidth property.

See Also

BevelInner (see page 2482)

BevelOuter (see page 2483)

19.1.7.5.12 TCategoryButtons.ButtonFlow Property

Specifies the direction of buttons in a category.

Pascal

```
property ButtonFlow: TCatButtonFlow;
```

C++

```
__property TCatButtonFlow ButtonFlow;
```

Description

Use one of the following values:

Parameter	Description
cbfVertical	Organizes the buttons in the category vertically.
cbfHorizontal	Organizes the buttons in the category horizontally.

19.1.7.5.13 TCategoryButtons.ButtonHeight Property

Height (see page 2496) of buttons in a category.

Pascal

```
property ButtonHeight: Integer;
```

C++

```
__property int ButtonHeight;
```

Description

ButtonHeight specifies the height of buttons in each of the TButtonCategory (see page 2429) objects.

See Also

TButtonCategory (see page 2429)

TButtonItem (see page 2441)

ButtonWidth (see page 2486)

19.1.7.5.14 TCategoryButtons.ButtonOptions Property

Specifies options for button display and behavior.

Pascal

```
property ButtonOptions: TCatButtonOptions;
```

C++

```
__property TCatButtonOptions ButtonOptions;
```

Description

The following options are available:

Option	Description
boAllowReorder	Allows the user to reorder buttons within a category at runtime.
boAllowCopyingButtons	Allows the user to copy buttons at runtime.
boFullSize	Sets the width of the buttons in a category to the maximum size, which is the entire width of the TButtonCategory (see page 2429) object. In this case, each button appears on its own line.
boGradientFill	The color of the TButtonCategory (see page 2429) button is shaded along a gradient. Otherwise, the color is solid.
boShowCaptions	Toggles the button captions. The category captions remain displayed.
boVerticalCategoryCaptions	The caption for a category appears on the side vertically when the group is expanded. Otherwise, it appears above the group.
boBoldCaptions	Make category captions bold.
boUsePlusMinus	Use plus/minus signs for expanding or collapsing a category. Otherwise, use arrows.
boCaptionOnlyBorder	When a category is collapsed so that only its caption button is displayed, draw a border around this button.

See Also

TButtonCategory (see page 2429)

19.1.7.5.15 TCategoryButtons.ButtonWidth Property

Width (see page 2506) of buttons in a category.

Pascal

```
property ButtonWidth: Integer;
```

C++

```
__property int ButtonWidth;
```

Description

ButtonWidth specifies the width of buttons in each of the TButtonCategory (see page 2429) objects.

See Also

TButtonCategory (see page 2429)

TButtonItem (see page 2441)

ButtonHeight (see page 2485)

19.1.7.5.16 TCategoryButtons.Categories Property

Specifies the categories represented in the category buttons object.

Pascal

```
property Categories: TButtonCategories;
```

C++

```
__property TButtonCategories Categories;
```

19.1.7.5.17 TCategoryButtons.Color Property

Category buttons background color

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Color specifies the background color of the TCategoryButtons (see page 2452) object.

19.1.7.5.18 TCategoryButtons.Cursor Property

Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.

Pascal

```
property Cursor: TCursor;
```

C++

```
__property TCursor Cursor;
```

Description

Change the value of Cursor to provide feedback to the user when the mouse pointer enters the control. The value of Cursor is

the index of the cursor in the list of cursors maintained by the global variable, Screen. In addition to the built-in cursors provided by TScreen, applications can add custom cursors to the list.

C++ Examples:

```
/*
This example shows how to add custom cursors to an
application.  The following code makes this cursor available
to the application via the constant crMyCursor, and sets it
as the global cursor to the application.
*/
Graphics::TBitmap *bmpMask, *bmpColor;
TIconInfo *iconInfo;

const crMyCursor = 5;

  void __fastcall TForm1::Button2Click(TObject *Sender)
{
  bmpMask = new Graphics::TBitmap();
  bmpColor = new Graphics::TBitmap();
  iconInfo = new TIconInfo();

  bmpMask->LoadFromFile("../SquareMask.bmp");
  bmpColor->LoadFromFile("../Square.bmp");

  iconInfo->fIcon = false;
  iconInfo->xHotspot = 15;
  iconInfo->yHotspot = 15;
  iconInfo->hbmMask = bmpMask->Handle;
  iconInfo->hbmColor = bmpColor->Handle;

  Screen->Cursors[crMyCursor] = CreateIconIndirect(iconInfo);

  Screen->Cursor = crMyCursor;

  delete bmpMask;
  delete bmpColor;
  delete iconInfo;
}
```

Delphi Examples:

```
{
This example uses a splitter and two image controls to
create a "slide show" effect.  The example displays an image
(the current "slide"), which can be hidden behind a screen
that looks like a venetian blind. When the blind is pulled
down to completely cover the current slide, the slide changes
to a different image.  When the blind is raised, the new
image appears.
To prepare this example, follow these steps:
1. Place an image control on the form.
Set the picture property to a bitmap of a venetian blind,
Set the Stretch property to True.
Set the Align property to alTop.
2. Add a splitter control to the form.
Set the Align property to alTop.
Set the Cursor property to crVSplit.
Set the OnMoved event handler to the code that appears below.
Set the MinSize property to 1.
3. Add a second image control to the form.
Set the Align property to alClient.
Set the Stretch property to True.
4. Select the form and add the OnCreate and OnDestroy event
   handlers that appear below.
5. Add the global variables below to the unit.
}
```

```
var
  CurImage: Integer;
  BMPs: Array[1..5] of TBitmap;

procedure TForm1.Splitter1Moved(Sender: TObject);
begin
  { check if the blind is closed }
  if Image2.Height <= 1 then
  begin
    { the blind is closed, change the current slide }
    CurImage := CurImage + 1;
    if CurImage = 6 then
       CurImage := 1;
    Image2.Picture.Bitmap := BMPs[CurImage];
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { Initialize the array of slides.
    These must be freed in the OnDestroy event handler. }
  BMPs[1] := TBitmap.Create;
  BMPs[1].LoadFromFile('Factory.bmp');
  BMPs[2] := TBitmap.Create;
  BMPs[2].LoadFromFile('Rhododendron.bmp');
  BMPs[3] := TBitmap.Create;
  BMPs[3].LoadFromFile('littleB_64.bmp');
  BMPs[4] := TBitmap.Create;
  BMPs[4].LoadFromFile('Soap bubbles.bmp');
  BMPs[5] := TBitmap.Create;
  BMPs[5].LoadFromFile('Gone Fishing.bmp');
  CurImage := 1;
  Image2.Picture.Bitmap := BMPs[1];
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  { clean up }
  for I := 1 to 5 do
    BMPs[I].Free;
end;
{
This example shows how to add custom cursors to an
application.  The following code makes this cursor available
to the application via the constant crMyCursor, and sets it
as the global cursor to the application.
}
var
  bmpMask : TBitmap;
  bmpColor : TBitmap;
  iconInfo : TIconInfo;
const
  crMyCursor = 5;

procedure TForm1.Button2Click(Sender: TObject);
begin
  bmpMask := TBitmap.Create;
  bmpColor := TBitmap.Create;

  bmpMask.LoadFromFile('SquareMask.bmp');
  bmpColor.LoadFromFile('Square.bmp');

  with iconInfo do
  begin
```

```
      fIcon := false;
      xHotspot := 15;
      yHotspot := 15;
      hbmMask := bmpMask.Handle;
      hbmColor := bmpColor.Handle;
    end;

    Screen.Cursors[crMyCursor] := CreateIconIndirect(iconInfo);

    Screen.Cursor := crMyCursor;

    bmpMask.Free;
    bmpColor.Free;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    DestroyIcon(Screen.Cursors[crMyCursor]);
  end;
```

See Also

DragCursor (see page 5818)

Cursors (see page 14212)

Screen (see page 14385)

ProcessMessages (see page 13691)

19.1.7.5.19 TCategoryButtons.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite: Boolean;
```

C++

```
__property Boolean DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
```

```
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

19.1.7.5.20 TCategoryButtons.DragButton Property

Button being dragged.

Pascal

```
property DragButton: TButtonItem;
```

C++

```
__property TButtonItem DragButton;
```

Description

DragButton specifies the button being dragged. You can write to this to control the button dragged.

See Also

DragCategory (see page 2490)

DragCursor (see page 2491)

DragImageList (see page 2491)

DragKind (see page 2492)

DragMode (see page 2492)

19.1.7.5.21 TCategoryButtons.DragCategory Property

Category being dragged.

Pascal

```
property DragCategory: TButtonCategory;
```

C++

```
__property TButtonCategory DragCategory;
```

Description

DragCategory specifies the category being dragged. You can write to this to control the category dragged.

See Also

TButtonCategory (see page 2429)

DragButton (see page 2490)

DragCursor (see page 2491)

DragImageList (see page 2491)

DragKind (see page 2492)

DragMode (see page 2492)

19.1.7.5.22 TCategoryButtons.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

19.1.7.5.23 TCategoryButtons.DragImageList Property

List of images used during drag or dock operations.

Pascal

```
property DragImageList: TDragImageList;
```

C++

```
__property TDragImageList DragImageList;
```

Description

DragImageList holds a list of images that are displayed during drag and drop or drag and dock operations. Categories (see page 2486) and buttons may be dragged in a CategoryButtons (see page 2415) object.

See Also

TDragImageList (see page 6021)

DragButton (see page 2490)

DragCategory (see page 2490)

DragCursor (see page 2491)

DragKind (see page 2492)

DragMode (see page 2492)

19.1.7.5.24 TCategoryButtons.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

19.1.7.5.25 TCategoryButtons.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
```

```
    else if (HT.Contains(htNowhere))
      AttachMode = naAdd;
    else if (HT.Contains(htOnIndent))
      AttachMode = naInsert;
    else
      return;
    TreeView1->Selected->MoveTo(Item, AttachMode);
  }

  void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
        int X, int Y, TDragState State, bool &Accept)
  {
    Accept = (Source == (TObject *)TreeView1);
  }
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

19.1.7.5.26 TCategoryButtons.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

19.1.7.5.27 TCategoryButtons.FocusedItem Property

Specifies the button that currently has focus.

Pascal

```
property FocusedItem: TButtonItem;
```

C++

```
__property TButtonItem FocusedItem;
```

19.1.7.5.28 TCategoryButtons.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
```

```
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

19.1.7.5.29 TCategoryButtons.GradientDirection Property

Specifies the direction of gradient fill for the category object.

Pascal

```
property GradientDirection: TGradientDirection;
```

C++

```
__property TGradientDirection GradientDirection;
```

Description

There are three primary objects that take color in the category buttons control. The first object is the container itself, which takes a background gradient. The second object is the category, which appears as a rounded rectangle within the container. The category also takes a gradient. The third object is an individual button, which appears as a rectangle in the category. The button takes a solid color.

GradientDirection applies to the category.

19.1.7.5.30 TCategoryButtons.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
HideSelection, Height example
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
```

```
        HideSelection := False;
    end;
end;
```

See Also

ClientHeight (see page 5808)

Width (see page 5849)

Using TDBListBox and TDBComboBox

19.1.7.5.31 TCategoryButtons.HotButtonColor Property

Specifies the color of a hot button, if necessary.

Pascal

```
property HotButtonColor: TColor;
```

C++

```
__property TColor HotButtonColor;
```

19.1.7.5.32 TCategoryButtons.Images Property

List of button images.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Images is a TCustomImageList object containing the button images for the CategoryButtons (see page 2415). Each button has an index into the image list.

See Also

TCustomImageList (see page 15361)

19.1.7.5.33 TCategoryButtons.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
```

```
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
```

```
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
```

```
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}
```

```
type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;
```

```
procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

19.1.7.5.34 TCategoryButtons.RegularButtonColor Property

Specifies the color of a button in the normal, unselected state.

Pascal

```
property RegularButtonColor: TColor;
```

C++

```
__property TColor RegularButtonColor;
```

19.1.7.5.35 TCategoryButtons.SelectedButtonColor Property

Specifies the color of the button when a user selects it.

Pascal

```
property SelectedButtonColor: TColor;
```

C++

```
__property TColor SelectedButtonColor;
```

19.1.7.5.36 TCategoryButtons.SelectedItem Property

Returns the selected button.

Pascal

```
property SelectedItem: TButtonItem;
```

C++

```
__property TButtonItem SelectedItem;
```

19.1.7.5.37 TCategoryButtons.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine

whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
```

```
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

19.1.7.5.38 TCategoryButtons.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

19.1.7.5.39 TCategoryButtons.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

19.1.7.5.40 TCategoryButtons.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
```

```
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

19.1.7.5.41 TCategoryButtons.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

See Also

ClientWidth (see page 5811)

Height (see page 5825)

Left (see page 5830)

19.1.7.6 TCategoryButtons Events

The events of the TCategoryButtons class are listed here.

Published Events

	Name	Description
	OnAfterDrawButton (see page 2508)	Fired after a button is drawn
	OnAlignInsertBefore (see page 2508)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2508)	Occurs when an object with custom alignment is aligned.
	OnBeforeDrawButton (see page 2509)	Fired before a button is drawn
	OnButtonClicked (see page 2509)	Fired when a button is clicked, if OnClick (see page 2510) event is not present
	OnCategoryCollapase (see page 2509)	Fired when a category is collapsed
	OnClick (see page 2510)	Occurs when mouse button goes up.
	OnContextPopup (see page 2510)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCopyButton (see page 2511)	Fired when a button is copied
	OnDockDrop (see page 2511)	Occurs when another control is docked to the control.
	OnDockOver (see page 2513)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 2515)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2517)	Occurs when the user drags an object over a control.
	OnDrawButton (see page 2519)	Fired when a button needs to be redrawn; it prevents default drawing.
	OnDrawIcon (see page 2520)	Fired when an icon needs to be redrawn; it prevents default drawing.
	OnDrawText (see page 2520)	Fired when a caption needs to be redrawn; it prevents default drawing.
	OnEndDock (see page 2520)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2520)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2521)	Occurs when a control receives the input focus.
	OnExit (see page 2524)	Occurs when the input focus shifts away from one control to another.
	OnGetHint (see page 2525)	Fired when a hint string is being retrieved
	OnHotButton (see page 2526)	Fired when the "hot" button changes
	OnKeyDown (see page 2526)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2529)	Occurs when a key is pressed.
	OnKeyUp (see page 2530)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2531)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2532)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2535)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2535)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2535)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2537)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 2540)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 2540)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 2540)	Occurs when the mouse wheel is rotated upward.
	OnReorderButton (see page 2541)	Fired when buttons are reordered

	OnReorderCategory (see page 2541)	Fired when categories are reordered
	OnSelectedItemChange (see page 2541)	Fired when the selected item changes
	OnStartDock (see page 2541)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2542)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

19.1.7.6.1 TCategoryButtons.OnAfterDrawButton Event

Fired after a button is drawn

Pascal

```
property OnAfterDrawButton: TCatButtonDrawEvent;
```

C++

```
__property TCatButtonDrawEvent OnAfterDrawButton;
```

19.1.7.6.2 TCategoryButtons.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore: TAlignInsertBeforeEvent;
```

C++

```
__property TAlignInsertBeforeEvent OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore (see page 6137)

Align (see page 5798)

TAlignInsertBeforeEvent (see page 6293)

19.1.7.6.3 TCategoryButtons.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition: TAlignPositionEvent;
```

C++

```
__property TAlignPositionEvent OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment

parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition (see page 6138)

Align (see page 5798)

Anchors (see page 5802)

TAlignInfo (see page 6269)

TAlignPositionEvent (see page 6293)

19.1.7.6.4 TCategoryButtons.OnBeforeDrawButton Event

Fired before a button is drawn

Pascal

```
property OnBeforeDrawButton: TCatButtonDrawEvent;
```

C++

```
__property TCatButtonDrawEvent OnBeforeDrawButton;
```

19.1.7.6.5 TCategoryButtons.OnButtonClicked Event

Fired when a button is clicked, if OnClick (see page 2510) event is not present

Pascal

```
property OnButtonClicked: TCatButtonEvent;
```

C++

```
__property TCatButtonEvent OnButtonClicked;
```

19.1.7.6.6 TCategoryButtons.OnCategoryCollapase Event

Fired when a category is collapsed

Pascal

```
property OnCategoryCollapase: TCategoryCollapseEvent;
```

C++

```
__property TCategoryCollapseEvent OnCategoryCollapase;
```

19.1.7.6.7 TCategoryButtons.OnClick Event

Occurs when mouse button goes up.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

OnClick occurs when the mouse button goes up. If there is no handler for this event, the OnButtonClicked (see page 2509) event is triggered.

See Also

OnButtonClicked

19.1.7.6.8 TCategoryButtons.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

19.1.7.6.9 TCategoryButtons.OnCopyButton Event

Fired when a button is copied

Pascal

```
property OnCopyButton: TCatButtonCopyEvent;
```

C++

```
__property TCatButtonCopyEvent OnCopyButton;
```

19.1.7.6.10 TCategoryButtons.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop: TDockDropEvent;
```

C++

```
__property TDockDropEvent OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
```

```
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
```

```
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockOver (see page 6216)

19.1.7.6.11 TCategoryButtons.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver: TDockOverEvent;
```

C++

```
__property TDockOverEvent OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
```

```
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
```

```
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnGetSiteInfo (see page 6223)

TDragDockObject (see page 6009)

19.1.7.6.12 TCategoryButtons.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
```

```
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

19.1.7.6.13 TCategoryButtons.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
```

```
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
```

```
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

19.1.7.6.14 TCategoryButtons.OnDrawButton Event

Fired when a button needs to be redrawn; it prevents default drawing.

Pascal

```
property OnDrawButton: TCatButtonDrawEvent;
```

C++

```
__property TCatButtonDrawEvent OnDrawButton;
```

19.1.7.6.15 TCategoryButtons.OnDrawIcon Event

Fired when an icon needs to be redrawn; it prevents default drawing.

Pascal

```
property OnDrawIcon: TCatButtonDrawIconEvent;
```

C++

```
__property TCatButtonDrawIconEvent OnDrawIcon;
```

19.1.7.6.16 TCategoryButtons.OnDrawText Event

Fired when a caption needs to be redrawn; it prevents default drawing.

Pascal

```
property OnDrawText: TCatButtonDrawEvent;
```

C++

```
__property TCatButtonDrawEvent OnDrawText;
```

19.1.7.6.17 TCategoryButtons.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

19.1.7.6.18 TCategoryButtons.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

19.1.7.6.19 TCategoryButtons.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
```

```
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
```

```
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

19.1.7.6.20 TCategoryButtons.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
```

```
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

19.1.7.6.21 TCategoryButtons.OnGetHint Event

Fired when a hint string is being retrieved

Pascal

```
property OnGetHint: TCatButtonGetHint;
```

C++

```
__property TCatButtonGetHint OnGetHint;
```

19.1.7.6.22 TCategoryButtons.OnHotButton Event

Fired when the "hot" button changes

Pascal

```
property OnHotButton: TCatButtonEvent;
```

C++

```
__property TCatButtonEvent OnHotButton;
```

19.1.7.6.23 TCategoryButtons.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
```

```
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
```

```
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

19.1.7.6.24 TCategoryButtons.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

19.1.7.6.25 TCategoryButtons.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

19.1.7.6.26 TCategoryButtons.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

19.1.7.6.27 TCategoryButtons.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
```

```
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

19.1.7.6.28 TCategoryButtons.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

19.1.7.6.29 TCategoryButtons.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

19.1.7.6.30 TCategoryButtons.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
```

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

19.1.7.6.31 TCategoryButtons.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

19.1.7.6.32 TCategoryButtons.OnMouseWheel Event

Occurs when the mouse wheel is rotated.

Pascal

```
property OnMouseWheel: TMouseWheelEvent;
```

C++

```
__property TMouseWheelEvent OnMouseWheel;
```

Description

Write code in the OnMouseWheel event handler to respond to mouse wheel messages. If there is no OnMouseWheel event handler, or if the mouse wheel message is not handled in the OnMouseWheel event handler, then an OnMouseWheelUp or OnMouseWheelDown event occurs, depending on the direction the mouse wheel was rotated.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheelUp (see page 5870)

19.1.7.6.33 TCategoryButtons.OnMouseWheelDown Event

Occurs when the mouse wheel is rotated downward.

Pascal

```
property OnMouseWheelDown: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelDown;
```

Description

Write code in the OnMouseWheelDown event handler to perform actions when the mouse wheel is rotated downward. OnMouseWheelDown occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheel (see page 5869)

OnMouseWheelUp (see page 5870)

19.1.7.6.34 TCategoryButtons.OnMouseWheelUp Event

Occurs when the mouse wheel is rotated upward.

Pascal

```
property OnMouseWheelUp: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelUp;
```

Description

Write code in the OnMouseWheelUp event handler to perform actions when the mouse wheel is rotated upward. OnMouseWheelUp occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheel (see page 5869)

19.1.7.6.35 TCategoryButtons.OnReorderButton Event

Fired when buttons are reordered

Pascal

```
property OnReorderButton: TCatButtonReorderEvent;
```

C++

```
__property TCatButtonReorderEvent OnReorderButton;
```

19.1.7.6.36 TCategoryButtons.OnReorderCategory Event

Fired when categories are reordered

Pascal

```
property OnReorderCategory: TCategoryReorderEvent;
```

C++

```
__property TCategoryReorderEvent OnReorderCategory;
```

19.1.7.6.37 TCategoryButtons.OnSelectedItemChange Event

Fired when the selected item changes

Pascal

```
property OnSelectedItemChange: TCatButtonEvent;
```

C++

```
__property TCatButtonEvent OnSelectedItemChange;
```

19.1.7.6.38 TCategoryButtons.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is

automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

19.1.7.6.39 TCategoryButtons.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

19.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCatButtonFlow (see page 2543)	Defines the ordering style of category buttons.

19.2.1 CategoryButtons.TCatButtonFlow Enumeration

Defines the ordering style of category buttons.

Pascal

```
TCatButtonFlow = (
  cbfVertical,
  cbfHorizontal
);
```

C++

```
enum TCatButtonFlow {
  cbfVertical,
  cbfHorizontal
};
```

File

CategoryButtons

Description

The following values are available for ordering the category buttons:

Value	Description
cbfVertical	Stacks the buttons vertically.
cbfHorizontal	Arranges the buttons horizontally and sequentially.

19.3 Types

The following table lists types in this documentation.

Types

Name	Description
TBaseButtonItemClass (see page 2544)	Defines button items that are to be included in a container, such as in button groups or category buttons.
TButtonCategoriesClass (see page 2544)	Metaclass of the TButtonCategories (see page 2424) class.
TButtonCategoryClass (see page 2545)	Metaclass of TButtonCategory (see page 2429).
TButtonDrawState (see page 2545)	Values indicating the drawing state of buttons.
TButtonItemActionLinkClass (see page 2546)	Metaclass of TButtonItemActionLink (see page 2445).
TButtonItemClass (see page 2546)	Defines the button item class.
TCatButtonCopyEvent (see page 2546)	This is type CategoryButtons.TCatButtonCopyEvent.
TCatButtonDrawEvent (see page 2547)	This is type CategoryButtons.TCatButtonDrawEvent.
TCatButtonDrawIconEvent (see page 2547)	This is type CategoryButtons.TCatButtonDrawIconEvent.
TCatButtonEvent (see page 2547)	This is type CategoryButtons.TCatButtonEvent.
TCatButtonGetHint (see page 2548)	This is type CategoryButtons.TCatButtonGetHint.
TCatButtonOptions (see page 2548)	Defines button options for category buttons.
TCatButtonReorderEvent (see page 2549)	This is type CategoryButtons.TCatButtonReorderEvent.
TCategoryCollapseEvent (see page 2549)	This is type CategoryButtons.TCategoryCollapseEvent.
TCategoryReorderEvent (see page 2549)	This is type CategoryButtons.TCategoryReorderEvent.

19.3.1 CategoryButtons.TBaseButtonItemClass Type

Defines button items that are to be included in a container, such as in button groups or category buttons.

Pascal

```
TBaseButtonItemClass = class of TBaseButtonItem;
```

C++

```
TBaseButtonItem TBaseButtonItemClass;
```

File

CategoryButtons

19.3.2 CategoryButtons.TButtonCategoriesClass Type

Metaclass of the TButtonCategories (see page 2424) class.

Pascal

```
TButtonCategoriesClass = class of TButtonCategories;
```

C++

```
TButtonCategories TButtonCategoriesClass;
```

File

CategoryButtons

19.3.3 CategoryButtons.TButtonCategoryClass Type

Metaclass of TButtonCategory (see page 2429).

Pascal

```
TButtonCategoryClass = class of TButtonCategory;
```

C++

```
TButtonCategory TButtonCategoryClass;
```

File

CategoryButtons

Description

You can pass this type in a function at runtime when you do not necessarily know the type.

19.3.4 CategoryButtons.TButtonDrawState Type

Values indicating the drawing state of buttons.

Pascal

```
TButtonDrawState = set of (bdsSelected, bdsHot, bdsFocused, bdsDown, bdsDragged,
bdsInsertLeft, bdsInsertTop, bdsInsertRight, bdsInsertBottom);
```

C++

```
(bdsSelected, bdsHot, bdsFocused, bdsDown, bdsDragged, bdsInsertLeft, bdsInsertTop,
bdsInsertRight, bdsInsertBottom) TButtonDrawState;
```

File

CategoryButtons

Description

You can use the following parameters to set the drawing state of the buttons:

Parameter	Description
bdsSelected	Item is selected and in the down state.
bdsHot	Item has the mouse over it
bdsFocused	Item should show focus.
bdsDown	Item is being pressed by the user.
bdsDragged	The drag image for that button is being drawn.
bdsInsertLeft	Shows that an item can be inserted to the left of this item.
bdsInsertTop	Shows that an item can be inserted above this item.
bdsInsertRight	Shows that an item can be inserted to the right of this item.

bdsInsertBottom	Show that an item can be inserted below this item.

19.3.5 CategoryButtons.TButtonItemActionLinkClass Type

Metaclass of TButtonItemActionLink (see page 2445).

Pascal

```
TButtonItemActionLinkClass = class of TButtonItemActionLink;
```

C++

```
TButtonItemActionLink TButtonItemActionLinkClass;
```

File

CategoryButtons

19.3.6 CategoryButtons.TButtonItemClass Type

Defines the button item class.

Pascal

```
TButtonItemClass = class of TButtonItem;
```

C++

```
TButtonItem TButtonItemClass;
```

File

CategoryButtons

Description

Inherits from the TBaseButtonItemClass (see page 2544).

19.3.7 CategoryButtons.TCatButtonCopyEvent Type

Pascal

```
TCatButtonCopyEvent = procedure (Sender: TObject; const SourceButton, CopiedButton:
TButtonItem) of object;
```

C++

```
(Sender: TObject; const SourceButton, CopiedButton: TButtonItem) ( TCatButtonCopyEvent)();
```

File

CategoryButtons

Description

This is type CategoryButtons.TCatButtonCopyEvent.

19.3.8 CategoryButtons.TCatButtonDrawEvent Type

Pascal

```
TCatButtonDrawEvent = procedure (Sender: TObject; const Button: TButtonItem; Canvas: TCanvas;
Rect: TRect; State: TButtonDrawState) of object;
```

C++

```
(Sender: TObject; const Button: TButtonItem; Canvas: TCanvas; Rect: TRect; State:
TButtonDrawState) ( TCatButtonDrawEvent)();
```

File

CategoryButtons

Description

This is type CategoryButtons.TCatButtonDrawEvent.

19.3.9 CategoryButtons.TCatButtonDrawIconEvent Type

Pascal

```
TCatButtonDrawIconEvent = procedure (Sender: TObject; const Button: TButtonItem; Canvas:
TCanvas; Rect: TRect; State: TButtonDrawState; var TextOffset: Integer) of object;
```

C++

```
(Sender: TObject; const Button: TButtonItem; Canvas: TCanvas; Rect: TRect; State:
TButtonDrawState; var TextOffset: Integer) ( TCatButtonDrawIconEvent)();
```

File

CategoryButtons

Description

This is type CategoryButtons.TCatButtonDrawIconEvent.

19.3.10 CategoryButtons.TCatButtonEvent Type

Pascal

```
TCatButtonEvent = procedure (Sender: TObject; const Button: TButtonItem) of object;
```

C++

```
(Sender: TObject; const Button: TButtonItem) ( TCatButtonEvent)();
```

File

CategoryButtons

Description

This is type CategoryButtons.TCatButtonEvent.

19.3.11 CategoryButtons.TCatButtonGetHint Type

Pascal

```
TCatButtonGetHint = procedure (Sender: TObject; const Button: TButtonItem; const Category:
TButtonCategory; var HintStr: string; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; const Button: TButtonItem; const Category: TButtonCategory; var HintStr:
string; var Handled: Boolean) ( TCatButtonGetHint)();
```

File

CategoryButtons

Description

This is type CategoryButtons.TCatButtonGetHint.

19.3.12 CategoryButtons.TCatButtonOptions Type

Defines button options for category buttons.

Pascal

```
TCatButtonOptions = set of (boAllowReorder, boAllowCopyingButtons, boFullSize, boGradientFill,
boShowCaptions, boVerticalCategoryCaptions, boBoldCaptions, boUsePlusMinus,
boCaptionOnlyBorder);
```

C++

```
(boAllowReorder, boAllowCopyingButtons, boFullSize, boGradientFill, boShowCaptions,
boVerticalCategoryCaptions, boBoldCaptions, boUsePlusMinus, boCaptionOnlyBorder)
TCatButtonOptions;
```

File

CategoryButtons

Description

The following options are supported:

Option	Description
boAllowReorder	Allows the user to reorder buttons at runtime.
boAllowCopyingButtons	Allows the user to copy buttons, typically with a drag operation.
boFullSize	Sets the buttons to full size, meaning they are stacked vertically and their horizontal length takes up the entire width of the container.
boGradientFill	Sets a gradiant fill.
boShowCaptions	Displays the captions on buttons.
boVerticalCategoryCaptions	Sets category captions to a vertical orientation.
boBoldCaptions	Sets captions to bold.
boUsePlusMinus	Sets a plus and minus collapse control on the left-hand side of the category.

boCaptionOnlyBorder	When category is expanded, shows only the caption in a slight section of the category gradient. When set to **False**, the border shows the gradient around the entire group of buttons in the category.

19.3.13 CategoryButtons.TCatButtonReorderEvent Type

Pascal

```
TCatButtonReorderEvent = procedure (Sender: TObject; const Button: TButtonItem; const
SourceCategory, TargetCategory: TButtonCategory) of object;
```

C++

```
(Sender: TObject; const Button: TButtonItem; const SourceCategory, TargetCategory:
TButtonCategory) ( TCatButtonReorderEvent)();
```

File

CategoryButtons

Description

This is type CategoryButtons.TCatButtonReorderEvent.

19.3.14 CategoryButtons.TCategoryCollapseEvent Type

Pascal

```
TCategoryCollapseEvent = procedure (Sender: TObject; const Category: TButtonCategory) of
object;
```

C++

```
(Sender: TObject; const Category: TButtonCategory) ( TCategoryCollapseEvent)();
```

File

CategoryButtons

Description

This is type CategoryButtons.TCategoryCollapseEvent.

19.3.15 CategoryButtons.TCategoryReorderEvent Type

Pascal

```
TCategoryReorderEvent = procedure (Sender: TObject; const SourceCategory, TargetCategory:
TButtonCategory) of object;
```

C++

```
(Sender: TObject; const SourceCategory, TargetCategory: TButtonCategory) (
TCategoryReorderEvent)();
```

File

CategoryButtons

Description

This is type CategoryButtons.TCategoryReorderEvent.

19.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
crDragCopy (see page 2551)	This is constant CategoryButtons.crDragCopy.

19.4.1 CategoryButtons.crDragCopy Constant

Pascal

```
crDragCopy = TCursor(-23);
```

C++

```
crDragCopy = TCursor(-23);
```

File

CategoryButtons

Description

This is constant CategoryButtons.crDragCopy.

20 CGIApp Namespace

This is namespace CGIApp.

Topics

Name	Description
Classes (see page 2554)	The following table lists classes in this documentation.

Classes

	Name	Description
	TCGIApplication (see page 2554)	TCGIApplication encapsulates a CGI server application.
	TCGIFactory (see page 2559)	

20.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCGIApplication (see page 2554)	TCGIApplication encapsulates a CGI server application.
	TCGIFactory (see page 2559)	

20.1.1 TCGIApplication Class

TCGIApplication encapsulates a CGI server application.

Class Hierarchy

Pascal

```
TCGIApplication = class(TWebApplication);
```

C++

```
class TCGIApplication : public TWebApplication;
```

File

CGIApp

Description

TCGIApplication provides the fundamental behavior of a CGI server application. TCGIApplication starts running when the CGI server receives an HTTP request message. It creates TCGIRequest and TCGIResponse objects to represent the request message and any response that should be sent in return, passes these to the dispatcher so that the response can be filled in, and sends the response if it was not already sent by a TWebActionItem.

Each CGI server project automatically declares an Application variable of type TCGIApplication as the instance of the application. This declaration overrides the Application variable declared in the Forms (see page 13657) unit if the CGIApp (see page 2553) unit is declared last in the uses list of the project file. Every CGI server application must include the CGIApp (see page 2553) unit.

To work with an Apache server, use the TApacheApplication object. To work with an ISAPI or NSAPI server, use the TISAPIApplication object instead.

See Also

TISAPIApplication (see page 15648)

20.1.1.1 TCGIApplication Members

The following tables list the members exposed by TCGIApplication.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActivateWebModules (see page 25453)	Activates a Web module for the application and returns it.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DeactivateWebModules (see page 25453)	Deactivates the Web module of the application.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleRequest (see page 25454)	Responds to an HTTP request message received by the server.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.

	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddWebModuleFactory (see page 25453)	Adds a Web module factory to the request handler.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24622)	Creates an instance of TWebApplication (see page 24618).
V	Create (see page 25452)	Creates an instance of TWebRequestHandler (see page 25448).
V	CreateForm (see page 24623)	Registers the single Web module with the Web application.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2557)	Frees the memory associated with the TCGIApplication (see page 2554) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandleException (see page 25454)	Responds to exceptions related to TWebRequestHandler (see page 25448) objects.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

V	Initialize (see page 24623)	Introduces an interface for a method that initializes the application.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Run (see page 2558)	Executes the CGI application in response to an HTTP request message.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActiveCount (see page 25455)	Indicates the number of active connections currently being serviced by the application.
	CacheConnections (see page 25455)	Determines whether the Web modules used by the application are cached for re-use.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	InactiveCount (see page 25456)	Indicates the number of Web modules cached for later reuse.
	MaxConnections (see page 25457)	Sets or returns the MaxConnections used by the socket.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Title (see page 24624)	Specifies a title for the Web application.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	WebModuleClass (see page 25457)	Contains a reference to a Web module.
R	WebModuleFactory (see page 25457)	A list of Web module factory objects.
R	WebModuleFactoryCount (see page 25458)	The number of Web module factories in use by the request handler.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

20.1.1.2 TCGIApplication.Destroy Destructor

Frees the memory associated with the TCGIApplication (see page 2554) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCGIApplication();
```

Description

Do not call Destroy. The global Application variable is freed when the application shuts down.

20.1.1.3 TCGIApplication Methods

The methods of the TCGIApplication class are listed here.

Public Methods

	Name	Description
	Run (see page 2558)	Executes the CGI application in response to an HTTP request message.

20.1.1.3.1 TCGIApplication.Run Method

Executes the CGI application in response to an HTTP request message.

Pascal

```
procedure Run; override;
```

C++

```
virtual __fastcall Run();
```

Description

When creating a new project, the IDE automatically creates a main program block in the project file that calls the Run method. When the application is started in response to an HTTP request message, the CGI application's Run method is called.

After calling the inherited Run method to set up the default error handler and shut down sequence for the application, TCGIApplication (see page 2554)'s Run method creates the appropriate request and response objects for the HTTP request message that started the CGI application. Run calls the HandleRequest method to pass the request and response on to the dispatcher where the response is filled out and sent. Finally, Run frees the request and response objects.

See Also

TCGIRequest (see page 2564)

TCGIResponse (see page 2572)

TWebDispatcher (see page 15147)

TWinCGIRequest (see page 2583)

TWinCGIResponse (see page 2590)

HandleRequest (see page 25454)

20.1.2 TCGIFactory Class

Class Hierarchy

System::TObject → CGIApp.TCGIFactory

Pascal

```
TCGIFactory = class(TObject);
```

C++

```
class TCGIFactory : public TObject;
```

File

CGIApp

See Also

TCGIApplication (see page 2554)

20.1.2.1 TCGIFactory Members

The following tables list the members exposed by TCGIFactory.

Protected Methods

	Name	Description
	NewRequest (see page 2560)	
	NewResponse (see page 2561)	

Protected Properties

	Name	Description
	OutputFileName (see page 2561)	This is OutputFileName, a member of class TCGIFactory.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2560)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

20.1.2.2 TCGIFactory.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TCGIFactory();
```

Description

Do not create instances of Create. Instead, construct descendants of Create using the new keyword, passing any arguments for the descendant object's constructor.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

Destroy (see page 22287)

new

20.1.2.3 TCGIFactory Methods

The methods of the TCGIFactory class are listed here.

Protected Methods

	Name	Description
V	NewRequest (see page 2560)	
V	NewResponse (see page 2561)	

20.1.2.3.1 TCGIFactory.NewRequest Method

Pascal

```
function NewRequest: TCGIRequest; virtual;
```

C++

```
virtual __fastcall TCGIRequest NewRequest();
```

See Also

TCGIResponse (see page 2572)

NewResponse (see page 2561)

20.1.2.3.2 TCGIFactory.NewResponse Method

Pascal

```
function NewResponse(CGIRequest: TCGIRequest): TCGIResponse; virtual;
```

C++

```
virtual __fastcall TCGIResponse NewResponse(TCGIRequest CGIRequest);
```

See Also

TCGIResponse (see page 2572)

NewRequest (see page 2560)

Request_Object

20.1.2.4 TCGIFactory Properties

The properties of the TCGIFactory class are listed here.

Protected Properties

	Name	Description
	OutputFileName (see page 2561)	This is OutputFileName, a member of class TCGIFactory.

20.1.2.4.1 TCGIFactory.OutputFileName Property

Pascal

```
property OutputFileName: string;
```

C++

```
__property AnsiString OutputFileName;
```

Description

This is OutputFileName, a member of class TCGIFactory.

21 CGIHTTP Namespace

This is namespace CGIHTTP.

Topics

Name	Description
Classes (see page 2564)	The following table lists classes in this documentation.
Types (see page 2591)	The following table lists types in this documentation.

Classes

	Name	Description
	TCGIRequest (see page 2564)	TCGIRequest represents client request information that is passed to a CGI Web server application.
	TCGIResponse (see page 2572)	TCGIResponse represents HTTP messages sent by a CGI application in response to an HTTP request message.
	TWinCGIRequest (see page 2583)	TWinCGIRequest represents client request information that is passed to a Windows hosted CGI Web server application.
	TWinCGIResponse (see page 2590)	TWinCGIResponse represents HTTP messages sent by a Windows hosted CGI application in response to an HTTP request message.

Types

Name	Description
TCGIRequestClass (see page 2591)	This is type CGIHTTP.TCGIRequestClass.
TCGIResponseClass (see page 2591)	This is type CGIHTTP.TCGIResponseClass.
TWinCGIRequestClass (see page 2592)	This is type CGIHTTP.TWinCGIRequestClass.
TWinCGIResponseClass (see page 2592)	This is type CGIHTTP.TWinCGIResponseClass.

21.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCGIRequest (see page 2564)	TCGIRequest represents client request information that is passed to a CGI Web server application.
	TCGIResponse (see page 2572)	TCGIResponse represents HTTP messages sent by a CGI application in response to an HTTP request message.
	TWinCGIRequest (see page 2583)	TWinCGIRequest represents client request information that is passed to a Windows hosted CGI Web server application.
	TWinCGIResponse (see page 2590)	TWinCGIResponse represents HTTP messages sent by a Windows hosted CGI application in response to an HTTP request message.

21.1.1 TCGIRequest Class

TCGIRequest represents client request information that is passed to a CGI Web server application.

Class Hierarchy

Pascal

```
TCGIRequest = class(TWebRequest);
```

C++

```
class TCGIRequest : public TWebRequest;
```

File

CGIHTTP

Description

Use TCGIRequest to interpret an HTTP request to a CGI server application.

CGI application objects create TCGIRequest objects to capture client requests. The properties of TCGIRequest are obtained from the Web server. The CGI application uses TCGIRequest objects to create corresponding TCGIResponse (see page 2572) objects. The TWebDispatcher for the application then invokes the TWebActionItem that is associated with the path portion of the URL of the TCGIRequest object.

See Also

TCGIApplication (see page 2554)

TCGIResponse (see page 2572)

TWebActionItem (see page 15130)

TWebDispatcher (see page 15147)

TWinCGIRequest (see page 2583)

Accessing Client Request Information

21.1.1.1 TCGIRequest Members

The following tables list the members exposed by TCGIRequest.

Protected Methods

	Name	Description
	GetDateVariable (see page 2567)	Provides the abstract (pure virtual) prototype for a method to extract a date value from an indexed table of HTTP header variables.
	GetIntegerVariable (see page 2568)	Provides the abstract (pure virtual) prototype for a method to extract an integer value from an indexed table of HTTP header variables.
	GetInternalPathInfo (see page 15169)	This is GetInternalPathInfo, a member of class TWebRequest.
	GetInternalScriptName (see page 15169)	This is GetInternalScriptName, a member of class TWebRequest.
	GetStringVariable (see page 2569)	Provides the abstract (pure virtual) prototype for a method to extract a string value from an indexed table of HTTP header variables.
	UpdateMethodType (see page 15172)	Provides the abstract prototype for a method to write a pure virtual method for updating the client method type.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2567)	Creates an instance of TCGIRequest (see page 2564).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 15165)	Destroys an instance of TWebRequest (see page 15161).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExtractContentFields (see page 15166)	Parses the Content (see page 15176) property into its constituent fields.
	ExtractCookieFields (see page 15166)	Parses the Cookie (see page 15180) property into its constituent fields.
	ExtractFields (see page 15167)	Parses a multi-valued string into its constituent fields.
	ExtractQueryFields (see page 15167)	Parses the Query (see page 15186) property into its constituent fields.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetFieldByName (see page 2568)	Provides a method that returns the value of the HTTP header variable specified by the Name parameter.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ReadClient (see page 2570)	Provides a method to read from the HTTP request into a buffer.
	ReadString (see page 2570)	Provides the method to read Count bytes from the HTTP request into a string.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	TranslateURI (see page 2570)	Provides a method to translate a Uniform Resource Identifier (URI) into an internal representation of a location on the server.
	WriteClient (see page 2571)	Provides a method to write Count bytes from Buffer back to the HTTP client.
	WriteHeaders (see page 2571)	Writes an HTTP header string.
	WriteString (see page 2572)	Provides a method for writingthe string AString back to the HTTP client.

Public Properties

	Name	Description
	Accept (see page 15174)	Reports the value of the Accept header of the HTTP request message.
	Authorization (see page 15175)	Reports the value of the Authorization header of the HTTP request message.
	CacheControl (see page 15175)	Reports the value of the Cache-Control header of the HTTP request message.
	Connection (see page 15176)	Reports the value of the Connection header of the HTTP request message.
	Content (see page 15176)	Represents the unparsed contents of the HTTP request message.
	ContentEncoding (see page 15177)	Reports the value of the Content (see page 15176)-Encoding header of the HTTP request message.
	ContentFields (see page 15177)	Reports the parsed content of a POST request message.
	ContentLength (see page 15178)	Reports the value of the Content (see page 15176)-Length header of the HTTP request message.
	ContentParser (see page 15178)	Refers to the object's content parser object.
	ContentType (see page 15179)	Reports the value of the Content (see page 15176)-Type header of the HTTP request message.
	ContentVersion (see page 15179)	Reports the value of the Content (see page 15176)-Version header of the HTTP request message.
	Cookie (see page 15180)	Reports the value of the Cookie header of the HTTP request message.
	CookieFields (see page 15180)	Represents the parsed contents of the Cookie (see page 15180) header of an HTTP request message.
	Date (see page 15181)	Reports the value of the Date header of the HTTP request message.
	DerivedFrom (see page 15181)	Reports the value of the Derived-From (see page 15182) header of the HTTP request message.
	Expires (see page 15182)	Reports the value of the Expires header of the HTTP request message.
	Files (see page 15182)	Provides access to the files uploaded in the HTTP request.
	From (see page 15182)	Reports the value of the From header of the HTTP request message.
	Host (see page 15183)	Reports the value of the Host header of the HTTP request message.
	IfModifiedSince (see page 15183)	Reports the value of the If-Modified-Since header of the HTTP request message.
	InternalPathInfo (see page 15184)	Returns the path info portion of the URL (see page 15190).
	InternalScriptName (see page 15184)	Returns the ScriptName (see page 15189) part of the URL (see page 15190).
	Method (see page 15185)	This is Method, a member of class TWebRequest.
	MethodType (see page 15185)	This is MethodType, a member of class TWebRequest.
	PathInfo (see page 15185)	Reports the value of the path information (if any) of the URL (see page 15190) specified in the HTTP request message.
	PathTranslated (see page 15186)	Represents a translation of the PathInfo (see page 15185) property to a fully qualified path on the Web server.
	ProtocolVersion (see page 15186)	Reports the version number of the HTTP protocol specified in the initial header of the HTTP request message.
	Query (see page 15186)	Indicates the value of the query information (if any) of the URL (see page 15190) specified in the HTTP request message.
	QueryFields (see page 15187)	Represents the parsed contents of the Query (see page 15186) in the URL (see page 15190) of the HTTP request message.
	Referer (see page 15188)	Reports the value of the Referer header of the HTTP request message.
	RemoteAddr (see page 15188)	Indicates the remote IP address of the client associated with the HTTP request message.
	RemoteHost (see page 15188)	Reports the fully qualified reverse domain lookup of the client associated with the HTTP request message.

R	ScriptName (see page 15189)	Indicates the value of the script information (if any) of the URL (see page 15190) specified in the HTTP request message.
R	ServerPort (see page 15189)	Indicates the port number of the port on the Web server that receives the HTTP request message.
R	Title (see page 15190)	Reports the value of the Title header of the HTTP request message.
R	URL (see page 15190)	Reports the Uniform Resource Identifier (URI) specified by the HTTP request message to identify the target of the request.
R	UserAgent (see page 15191)	Reports the value of the User-Agent header of the HTTP request message.

21.1.1.2 TCGIRequest.Create Constructor

Creates an instance of TCGIRequest (see page 2564).

Pascal

```
constructor Create;
```

C++

```
__fastcall TCGIRequest();
```

Description

Applications shouldn't need to directly instantiate a CGI request object. The TCGIApplication object creates a TCGIRequest (see page 2564) object whenever it receives a client request.

See Also

TCGIApplication (see page 2554)

21.1.1.3 TCGIRequest Methods

The methods of the TCGIRequest class are listed here.

Protected Methods

	Name	Description
V	GetDateVariable (see page 2567)	Provides the abstract (pure virtual) prototype for a method to extract a date value from an indexed table of HTTP header variables.
V	GetIntegerVariable (see page 2568)	Provides the abstract (pure virtual) prototype for a method to extract an integer value from an indexed table of HTTP header variables.
V	GetStringVariable (see page 2569)	Provides the abstract (pure virtual) prototype for a method to extract a string value from an indexed table of HTTP header variables.

Public Methods

	Name	Description
V	GetFieldByName (see page 2568)	Provides a method that returns the value of the HTTP header variable specified by the Name parameter.
V	ReadClient (see page 2570)	Provides a method to read from the HTTP request into a buffer.
V	ReadString (see page 2570)	Provides the method to read Count bytes from the HTTP request into a string.
V	TranslateURI (see page 2570)	Provides a method to translate a Uniform Resource Identifier (URI) into an internal representation of a location on the server.
V	WriteClient (see page 2571)	Provides a method to write Count bytes from Buffer back to the HTTP client.
V	WriteHeaders (see page 2571)	Writes an HTTP header string.
V	WriteString (see page 2572)	Provides a method for writingthe string AString back to the HTTP client.

21.1.1.3.1 TCGIRequest.GetDateVariable Method

Provides the abstract (pure virtual) prototype for a method to extract a date value from an indexed table of HTTP header

variables.

Pascal

```
function GetDateVariable(Index: Integer): TDateTime; override;
```

C++

```
virtual __fastcall TDateTime GetDateVariable(int Index);
```

Description

Descendants of TWebRequest implement GetDateVariable to provide a method by which the date-time values are read from the header strings of the HTTP request message. The request object uses GetDateVariable to access the properties that represent dates from the HTTP request headers.

See Also

Date (see page 15181)

Expires (see page 15182)

IfModifiedSince (see page 15183)

21.1.1.3.2 TCGIRequest.GetFieldByName Method

Provides a method that returns the value of the HTTP header variable specified by the Name parameter.

Pascal

```
function GetFieldByName(const Name: string): string; override;
```

C++

```
virtual __fastcall AnsiString GetFieldByName(const AnsiString Name);
```

Description

Use GetFieldByName to read the value of an HTTP header variable that is not surfaced as one of the properties of TCGIRequest (see page 2564). For information on possible HTTP request header variables, see the current Internet standards documents.

See Also

GetStringVariable (see page 15169)

21.1.1.3.3 TCGIRequest.GetIntegerVariable Method

Provides the abstract (pure virtual) prototype for a method to extract an integer value from an indexed table of HTTP header variables.

Pascal

```
function GetIntegerVariable(Index: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetIntegerVariable(int Index);
```

Description

Descendants of TWebRequest implement GetIntegerVariable to provide a method by which the integer values are read from the header strings of the HTTP request message. The request object uses GetIntegerVariable to access the properties that represent integers from the HTTP request headers.

See Also

ContentLength (see page 15178)

ServerPort (see page 15189)

21.1.1.3.4 TCGIRequest.GetStringVariable Method

Provides the abstract (pure virtual) prototype for a method to extract a string value from an indexed table of HTTP header variables.

Pascal

```
function GetStringVariable(Index: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetStringVariable(int Index);
```

Description

Descendants of TWebRequest implement GetStringVariable to provide a method by which the string values are read from the header strings of the HTTP request message. The request object uses GetStringVariable to access the properties that represent strings from the HTTP request headers.

See Also

Accept (see page 15174)

CacheControl (see page 15175)

Connection (see page 15176)

Content (see page 15176)

ContentEncoding (see page 15177)

ContentType (see page 15179)

ContentVersion (see page 15179)

Cookie (see page 15180)

DerivedFrom (see page 15181)

From (see page 15182)

Host (see page 15183)

TWebRequest (see page 15161)

PathInfo (see page 15185)

PathTranslated (see page 15186)

ProtocolVersion (see page 15186)

Query (see page 15186)

Referer (see page 15188)

RemoteAddr (see page 15188)

RemoteHost (see page 15188)

ScriptName (see page 15189)

Title (see page 15190)

URL (see page 15190)

UserAgent (see page 15191)

21.1.1.3.5 TCGIRequest.ReadClient Method

Provides a method to read from the HTTP request into a buffer.

Pascal

```
function ReadClient(var Buffer; Count: Integer): Integer; override;
```

C++

```
virtual __fastcall int ReadClient( Buffer, int Count);
```

Description

Use ReadClient to get the next chunk of information when the content of the request message is too large for the Content property.

Use Buffer to provide a buffer to contain the content information. Use Count to specify the number of bytes to read. ReadClient returns the number of bytes actually read.

See Also

Content (see page 15176)

ReadString (see page 2570)

WriteClient (see page 2571)

21.1.1.3.6 TCGIRequest.ReadString Method

Provides the method to read Count bytes from the HTTP request into a string.

Pascal

```
function ReadString(Count: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString ReadString(int Count);
```

Description

ReadString is used to load content information from the HTTP request into a string. Use ReadString to get the next chunk of information when the content of the request message is too large for the Content property.

See Also

Content (see page 15176)

ReadClient (see page 2570)

WriteString (see page 2572)

21.1.1.3.7 TCGIRequest.TranslateURI Method

Provides a method to translate a Uniform Resource Identifier (URI) into an internal representation of a location on the server.

Pascal

```
function TranslateURI(const URI: string): string; override;
```

C++

```
virtual __fastcall AnsiString TranslateURI(const AnsiString URI);
```

Description

Descendants of TCGIRequest (see page 2564) implement TranslateURI. TranslateURI is used to translate the URI specified in a client request into a format that can be used by the server application. TranslateURI is not relevant to CGI applications.

See Also

PathTranslated (see page 15186)

URL (see page 15190)

21.1.1.3.8 TCGIRequest.WriteClient Method

Provides a method to write Count bytes from Buffer back to the HTTP client.

Pascal

```
function WriteClient(var Buffer; Count: Integer): Integer; override;
```

C++

```
virtual __fastcall int WriteClient( Buffer, int Count);
```

Description

WriteClient is used by TCGIResponse (see page 2572) to send the contents of a response to the CGI client.

See Also

TWebResponse (see page 15191)

ReadClient (see page 2570)

WriteString (see page 2572)

WriteHeaders (see page 2571)

21.1.1.3.9 TCGIRequest.WriteHeaders Method

Writes an HTTP header string.

Pascal

```
function WriteHeaders(StatusCode: Integer; const StatusString: string; const Headers: string):
Boolean; override;
```

C++

```
virtual __fastcall Boolean WriteHeaders(int StatusCode, const AnsiString StatusString, const
AnsiString Headers);
```

Description

WriteHeaders writes the HTTP headers of the response message before a call to the WriteClient (see page 2571) method. These headers are used on the HTTP message that the WriteClient (see page 2571) method sends in response to this request.

StatusCode is the status code of the response. This informs the client of the status of the request. For a list of status code values, see the StatusCode property of TWebResponse.

StatusString is a text string that describes the status that StatusCode indicates.

Headers is a formatted string that includes any other HTTP headers to add. It represents each header as a string of the form 'Headername: Value', where name/value pairs are separated by line breaks.

WriteHeaders returns true if it succeeds in writing the specified headers.

See Also

WriteClient (see page 2571)

WriteString (see page 2572)

StatusCode (see page 15215)

21.1.1.3.10 TCGIRequest.WriteString Method

Provides a method for writingthe string AString back to the HTTP client.

Pascal

```
function WriteString(const AString: string): Boolean; override;
```

C++

```
virtual __fastcall Boolean WriteString(const AnsiString AString);
```

Description

WriteString is used by TCGIResponse (see page 2572) to send the contents of a response to the CGI client.

See Also

ReadString (see page 2570)

TCGIResponse (see page 2572)

WriteClient (see page 2571)

21.1.2 TCGIResponse Class

TCGIResponse represents HTTP messages sent by a CGI application in response to an HTTP request message.

Class Hierarchy

Pascal

```
TCGIResponse = class(TWebResponse);
```

C++

```
class TCGIResponse : public TWebResponse;
```

File

CGIHTTP

Description

Use TCGIResponse to specify the response to an HTTP request message in a CGI application.

The CGI application creates a TCGIResponse object based on the TCGIRequest (see page 2564) object for an incoming web client request message. The TWebDispatcher for the application then passes the TCGIResponse object to the TWebActionItem associated with the TCGIRequest (see page 2564) object. The TWebActionItem sets the properties of the TCGIResponse object.

See Also

TCGIApplication (see page 2554)

TCGIRequest (see page 2564)

TWebActionItem (see page 15130)

TWebDispatcher (see page 15147)

TWinCGIResponse (see page 2590)

Creating HTTP Response Messages

21.1.2.1 TCGIResponse Members

The following tables list the members exposed by TCGIResponse.

Protected Fields

	Name	Description
	FCustomHeaders (see page 15195)	This is FCustomHeaders, a member of class TWebResponse.
	FHTTPRequest (see page 15196)	This is FHTTPRequest, a member of class TWebResponse.

Protected Methods

	Name	Description
	AddCustomHeaders (see page 15197)	Appends the custom headers specified by the CustomHeaders (see page 15211) property to the Headers parameter.
	FormatAuthenticate (see page 15197)	Lists of one or more comma-separated challenges provided when the request fails with StatusCode (see page 15215) 401 (Unauthorized).
V	GetContent (see page 2576)	Provides the abstract (pure virtual) prototype of a method to access the Content property.
V	GetDateVariable (see page 2576)	Provides the abstract (pure virtual) prototype of a method to extract a date value from an indexed table of HTTP header variables.
V	GetIntegerVariable (see page 2577)	Provides the abstract (in C++ terminology, pure vitrual) prototype of a method to extract an integer value from an indexed table of HTTP header variables.
V	GetLogMessage (see page 2577)	Provides the abstract (pure virtual) prototype of a method to access the LogMessage property.
V	GetStatusCode (see page 2578)	Provides the abstract (pure virtual) prototype of a method to access the StatusCode property.
V	GetStringVariable (see page 2578)	Provides the prototype of a method to extract a string value from an indexed table of HTTP header variables.
V	InitResponse (see page 2579)	This is InitResponse, a member of class TCGIResponse.
V	SetContent (see page 2580)	Provides the abstract (pure virtual) prototype of a method to set the Content property.
V	SetContentStream (see page 15203)	Sets the value of the ContentStream (see page 15209) property to the Value parameter.
V	SetDateVariable (see page 2581)	Provides the abstract (pure virtual) prototype of a method to set a date-time valued HTTP header property.
V	SetIntegerVariable (see page 2581)	Provides the abstract (pure virtual) prototype of a method to set an integer valued HTTP header property.
V	SetLogMessage (see page 2582)	Provides the abstract (pure virtual) prototype of a method to set the value of the LogMessage property.
V	SetStatusCode (see page 2582)	Provides the abstract (pure virtual) prototype of a method to set the value of the StatusCode property.
V	SetStringVariable (see page 2582)	Provides the abstract (pure virtual) prototype of a method to set a string valued HTTP header property.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2575)	Creates an instance of TCGIResponse (see page 2572) to represent the response to the HTTPRequest parameter.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 15195)	Disposes of an object instance.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetCustomHeader (see page 15198)	Returns the value of the custom header with the specified name.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SendRedirect (see page 2579)	Sends the response with a StatusCode of 301 (Moved Permanently) and a Location of URI.
V	SendResponse (see page 2579)	Sends the HTTP response message to the web client.
V	SendStream (see page 2580)	Sends the HTTP response message with the contents set to the contents of AStream.
V	Sent (see page 2580)	Indicates whether the HTTP response message has already been sent.
	SetCookieField (see page 15203)	Adds a cookie header to the response message.
	SetCustomHeader (see page 15203)	Replaces the value of a specified custom header.

Public Properties

	Name	Description
	Allow (see page 15207)	Lists the HTTP request methods that the application can service.
	Content (see page 15207)	Contains the information sent to the web client in response to an HTTP request message.
	ContentEncoding (see page 15208)	Lists the mechanisms, if any, that are used to encode the content of the HTTP response message.
	ContentLength (see page 15208)	Specifies the length, in bytes, of the Content (see page 15207) property.
	ContentStream (see page 15209)	Specifies the stream object for writing the contents of the HTTP response message to the web client.
	ContentType (see page 15209)	Indicates the media type of the Content (see page 15207) of the HTTP response message.
	ContentVersion (see page 15210)	Represents the version associated with the Content (see page 15207) of the HTTP response message.
R	Cookies (see page 15210)	Lists all cookie headers that should be sent with the response.
	CustomHeaders (see page 15211)	Lists message header strings that are added to the response message before it is sent.
	Date (see page 15211)	Indicates the time and date when the HTTP response message is sent to the web client.
	DerivedFrom (see page 15212)	Specifies the URI of the resource where the Content (see page 15207) of the HTTP response originated.

	Expires (see page 15212)	Specifies the date and time after which the Content (see page 15207) of the HTTP response message should be considered invalid.
R	HTTPRequest (see page 15213)	Represents the HTTP request message for which this is a response.
	LastModified (see page 15213)	Indicates the date and time that the value of the Content (see page 15207) property was last updated.
	Location (see page 15214)	Specifies the URI of a resource the web client should point to for the completion of its request.
	LogMessage (see page 15214)	Describes the HTTP response message for event logging purposes.
	Realm (see page 15214)	Specifies the set of protected-access URIs that includes the URL specified by the HTTP request message.
	ReasonString (see page 15215)	Indicates the status of the response to the web client request as a human-readable string.
	Server (see page 15215)	Provides information about the software used by the server application to handle the request.
	StatusCode (see page 15215)	Indicates the status of the response to the HTTP request message as a 3-digit integer.
	Title (see page 15217)	Specifies a title for the Content (see page 15207) of the HTTP response message.
	Version (see page 15217)	Specifies the HTTP protocol version number that the web client is using.
	WWWAuthenticate (see page 15218)	Lists of one or more comma-separated challenges provided when the request fails with StatusCode (see page 15215) 401 (Unauthorized).

21.1.2.2 TCGIResponse.Create Constructor

Creates an instance of TCGIResponse (see page 2572) to represent the response to the HTTPRequest parameter.

Pascal

```
constructor Create(HTTPRequest: TWebRequest);
```

C++

```
__fastcall TCGIResponse(TWebRequest HTTPRequest);
```

Description

Applications shouldn't need to call Create to instantiate a CGI response object. The TCGIApplication object creates a TCGIResponse (see page 2572) for each TCGIRequest (see page 2564) it creates.

See Also

TCGIApplication (see page 2554)

TCGIRequest (see page 2564)

21.1.2.3 TCGIResponse Methods

The methods of the TCGIResponse class are listed here.

Protected Methods

	Name	Description
V	GetContent (see page 2576)	Provides the abstract (pure virtual) prototype of a method to access the Content property.
V	GetDateVariable (see page 2576)	Provides the abstract (pure virtual) prototype of a method to extract a date value from an indexed table of HTTP header variables.
V	GetIntegerVariable (see page 2577)	Provides the abstract (in C++ terminology, pure vitrual) prototype of a method to extract an integer value from an indexed table of HTTP header variables.
V	GetLogMessage (see page 2577)	Provides the abstract (pure virtual) prototype of a method to access the LogMessage property.
V	GetStatusCode (see page 2578)	Provides the abstract (pure virtual) prototype of a method to access the StatusCode property.

	GetStringVariable (see page 2578)	Provides the prototype of a method to extract a string value from an indexed table of HTTP header variables.
	InitResponse (see page 2579)	This is InitResponse, a member of class TCGIResponse.
	SetContent (see page 2580)	Provides the abstract (pure virtual) prototype of a method to set the Content property.
	SetDateVariable (see page 2581)	Provides the abstract (pure virtual) prototype of a method to set a date-time valued HTTP header property.
	SetIntegerVariable (see page 2581)	Provides the abstract (pure virtual) prototype of a method to set an integer valued HTTP header property.
	SetLogMessage (see page 2582)	Provides the abstract (pure virtual) prototype of a method to set the value of the LogMessage property.
	SetStatusCode (see page 2582)	Provides the abstract (pure virtual) prototype of a method to set the value of the StatusCode property.
	SetStringVariable (see page 2582)	Provides the abstract (pure virtual) prototype of a method to set a string valued HTTP header property.

Public Methods

	Name	Description
	SendRedirect (see page 2579)	Sends the response with a StatusCode of 301 (Moved Permanently) and a Location of URI.
	SendResponse (see page 2579)	Sends the HTTP response message to the web client.
	SendStream (see page 2580)	Sends the HTTP response message with the contents set to the contents of AStream.
	Sent (see page 2580)	Indicates whether the HTTP response message has already been sent.

21.1.2.3.1 TCGIResponse.GetContent Method

Provides the abstract (pure virtual) prototype of a method to access the Content property.

Pascal

```
function GetContent: string; override;
```

C++

```
virtual __fastcall AnsiString GetContent();
```

Description

Descendants of TWebResponse implement GetContent to provide an access method for their internal representation of the Content property. This method is abstract (pure virtual) because response objects for different types of HTTP applications store the value of the Content property differently.

See Also

Content (see page 15207)

SetContent (see page 15202)

21.1.2.3.2 TCGIResponse.GetDateVariable Method

Provides the abstract (pure virtual) prototype of a method to extract a date value from an indexed table of HTTP header variables.

Pascal

```
function GetDateVariable(Index: Integer): TDateTime; override;
```

C++

```
virtual __fastcall TDateTime GetDateVariable(int Index);
```

Description

Descendants of TWebResponse implement GetDateVariable to provide a method by which date-time values are read from the

internal table used to store HTTP header variables. The response object uses GetDateVariable to access the properties that represent dates.

See Also

Date (see page 15211)

Expires (see page 15212)

LastModified (see page 15213)

SetDateVariable (see page 15204)

21.1.2.3.3 TCGIResponse.GetIntegerVariable Method

Provides the abstract (in C++ terminology, pure vitrual) prototype of a method to extract an integer value from an indexed table of HTTP header variables.

Pascal

```
function GetIntegerVariable(Index: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetIntegerVariable(int Index);
```

Description

Descendants of TWebResponse implement GetIntegerVariable to provide a method by which integer values are read from the internal table used to store HTTP header variables. The response object uses GetIntegerVariable to access the properties that represent integers.

See Also

ContentLength (see page 15208)

SetIntegerVariable (see page 15204)

21.1.2.3.4 TCGIResponse.GetLogMessage Method

Provides the abstract (pure virtual) prototype of a method to access the LogMessage property.

Pascal

```
function GetLogMessage: string; override;
```

C++

```
virtual __fastcall AnsiString GetLogMessage();
```

Description

Descendants of TWebResponse implement GetLogMessage to provide an access method for internal representation of the LogMessage property. This method is abstract (pure virtual) because response objects for different types of HTTP applications store the value of the LogMessage property differently.

See Also

LogMessage (see page 15214)

SetLogMessage (see page 15205)

21.1.2.3.5 TCGIResponse.GetStatusCode Method

Provides the abstract (pure virtual) prototype of a method to access the StatusCode property.

Pascal

```
function GetStatusCode: Integer; override;
```

C++

```
virtual __fastcall int GetStatusCode();
```

Description

Descendants of TWebResponse implement GetStatusCode to provide an access method for internal representation of the StatusCode property. This method is abstract (pure virtual) because response objects for different types of HTTP applications store the value of the StatusCode property differently.

See Also

StatusCode (see page 15215)

SetStatusCode (see page 15205)

21.1.2.3.6 TCGIResponse.GetStringVariable Method

Provides the prototype of a method to extract a string value from an indexed table of HTTP header variables.

Pascal

```
function GetStringVariable(Index: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetStringVariable(int Index);
```

Description

GetStringVariable provides the abstract (pure virtual) prototype of a method to extract a string value from an indexed table of HTTP header variables. Descendants of TWebResponse implement GetStringVariable to provide a method by which the string values are read from the internal table used to store HTTP header variables. The response object uses GetStringVariable to access the properties that represent strings.

See Also

Allow (see page 15207)

ContentEncoding (see page 15208)

ContentType (see page 15209)

ContentVersion (see page 15210)

DerivedFrom (see page 15212)

Location (see page 15214)

Realm (see page 15214)

ReasonString (see page 15215)

Server (see page 15215)

SetStringVariable (see page 15205)

Title (see page 15217)

Version (see page 15217)

WWWAuthenticate (see page 15218)

21.1.2.3.7 TCGIResponse.InitResponse Method

Pascal

```
procedure InitResponse; virtual;
```

C++

```
virtual __fastcall InitResponse();
```

Description

This is InitResponse, a member of class TCGIResponse.

21.1.2.3.8 TCGIResponse.SendRedirect Method

Sends the response with a StatusCode of 301 (Moved Permanently) and a Location of URI.

Pascal

```
procedure SendRedirect(const URI: string); override;
```

C++

```
virtual __fastcall SendRedirect(const AnsiString URI);
```

Description

Call SendRedirect to redirect the web client request to another URI that is the proper target of the request. SendRedirect does the same thing as setting the StatusCode to 301, the Location to URI, and calling SendResponse (see page 2579).

See Also

Location (see page 15214)

SendResponse (see page 2579)

StatusCode (see page 15215)

21.1.2.3.9 TCGIResponse.SendResponse Method

Sends the HTTP response message to the web client.

Pascal

```
procedure SendResponse; override;
```

C++

```
virtual __fastcall SendResponse();
```

Description

Call SendResponse after setting the properties of the HTTP response object to specify the response that should be made to the web client. If the request should be redirected to another URI, it is more efficient to call SendRedirect (see page 2579) instead.

See Also

SendRedirect (see page 2579)

21.1.2.3.10 TCGIResponse.SendStream Method

Sends the HTTP response message with the contents set to the contents of AStream.

Pascal

```
procedure SendStream(AStream: TStream); override;
```

C++

```
virtual __fastcall SendStream(TStream AStream);
```

Description

Only call SendStream after first calling SendResponse (see page 2579). Unlike SendStream, SendResponse (see page 2579) fills the header of the HTTP response message, based on the properties of the TCGIResponse (see page 2572) object. If SendResponse (see page 2579) has not been called first to set the header of the HTTP response message, set the ContentStream property to AStream, and call SendResponse (see page 2579).

See Also

ContentStream (see page 15209)

SendResponse (see page 2579)

Sent (see page 2580)

21.1.2.3.11 TCGIResponse.Sent Method

Indicates whether the HTTP response message has already been sent.

Pascal

```
function Sent: Boolean; override;
```

C++

```
virtual __fastcall Boolean Sent();
```

Description

TCGIResponse (see page 2572) checks whether the message was sent and returns True or False true or false as appropriate.

TCGIApplication objects read Sent and send unsent responses to the web client.

See Also

SendRedirect (see page 2579)

SendStream (see page 2580)

TCGIApplication (see page 2554)

21.1.2.3.12 TCGIResponse.SetContent Method

Provides the abstract (pure virtual) prototype of a method to set the Content property.

Pascal

```
procedure SetContent(const Value: string); override;
```

C++

```
virtual __fastcall SetContent(const AnsiString Value);
```

Description

Descendants of TWebResponse implement SetContent to provide a method for setting the Content property. This method is abstract (pure virtual) because response objects for different types of HTTP applications can store the value of the Content property differently.

See Also

Content (see page 15207)

GetContent (see page 15198)

21.1.2.3.13 TCGIResponse.SetDateVariable Method

Provides the abstract (pure virtual) prototype of a method to set a date-time valued HTTP header property.

Pascal

```
procedure SetDateVariable(Index: Integer; const Value: TDateTime); override;
```

C++

```
virtual __fastcall SetDateVariable(int Index, const TDateTime Value);
```

Description

Descendants of TWebResponse implement SetDateVariable to provide a method by which date-time values are written to the internal table used to store HTTP header variables. The response object uses SetDateVariable to set the properties that represent dates.

See Also

Date (see page 15211)

Expires (see page 15212)

GetDateVariable (see page 15198)

LastModified (see page 15213)

21.1.2.3.14 TCGIResponse.SetIntegerVariable Method

Provides the abstract (pure virtual) prototype of a method to set an integer valued HTTP header property.

Pascal

```
procedure SetIntegerVariable(Index: Integer; Value: Integer); override;
```

C++

```
virtual __fastcall SetIntegerVariable(int Index, int Value);
```

Description

Descendants of TWebResponse implement SetIntegerVariable to provide a method by which integer values are written to the internal table used to store HTTP header variables. The response object uses SetIntegerVariable to set the properties that represent integers.

See Also

ContentLength (see page 15208)

GetIntegerVariable (see page 15199)

21.1.2.3.15 TCGIResponse.SetLogMessage Method

Provides the abstract (pure virtual) prototype of a method to set the value of the LogMessage property.

Pascal

```
procedure SetLogMessage(const Value: string); override;
```

C++

```
virtual __fastcall SetLogMessage(const AnsiString Value);
```

Description

Descendants of TWebResponse implement SetLogMessage to provide a method to set the internal representation of the LogMessage property. This method is abstract (pure virtual) because response objects for different types of HTTP applications store the value of the LogMessage property differently.

See Also

GetLogMessage (see page 15199)

LogMessage (see page 15214)

21.1.2.3.16 TCGIResponse.SetStatusCode Method

Provides the abstract (pure virtual) prototype of a method to set the value of the StatusCode property.

Pascal

```
procedure SetStatusCode(Value: Integer); override;
```

C++

```
virtual __fastcall SetStatusCode(int Value);
```

Description

Descendants of TWebResponse implement SetStatusCode to provide a method to set the internal representation of the StatusCode property. This method is abstract (pure virtual) because response objects for different types of HTTP applications store the value of the StatusCode property differently.

See Also

GetStatusCode (see page 15199)

StatusCode (see page 15215)

21.1.2.3.17 TCGIResponse.SetStringVariable Method

Provides the abstract (pure virtual) prototype of a method to set a string valued HTTP header property.

Pascal

```
procedure SetStringVariable(Index: Integer; const Value: string); override;
```

C++

```
virtual __fastcall SetStringVariable(int Index, const AnsiString Value);
```

Description

Descendants of TWebResponse implement SetStringVariable to provide a method by which string values are written to the internal table used to store HTTP header variables. The response object uses SetStringVariable to set the properties that represent strings.

See Also

Allow (see page 15207)

ContentEncoding (see page 15208)

ContentType (see page 15209)

ContentVersion (see page 15210)

DerivedFrom (see page 15212)

GetStringVariable (see page 15200)

Location (see page 15214)

Realm (see page 15214)

ReasonString (see page 15215)

Server (see page 15215)

Title (see page 15217)

Version (see page 15217)

WWWAuthenticate (see page 15218)

21.1.3 TWinCGIRequest Class

TWinCGIRequest represents client request information that is passed to a Windows hosted CGI Web server application.

Class Hierarchy

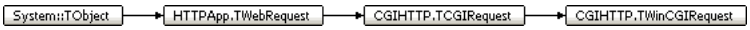

Pascal

```
TWinCGIRequest = class(TCGIRequest);
```

C++

```
class TWinCGIRequest : public TCGIRequest;
```

File

CGIHTTP

Description

Use TWinCGIRequest to interpret a client request to a Windows hosted CGI Web server application.

CGI application objects create TWinCGIRequest objects to capture client requests. The properties of the TWinCGIRequest objects are obtained from the Web server. The CGI application uses TWinCGIRequest objects to create corresponding TWinCGIResponse (see page 2590) objects. The TWebDispatcher object for the application then invokes the TWebActionItem that is associated with the path portion of the URL of the TWinCGIRequest object.

See Also

TCGIApplication (see page 2554)

TWebActionItem (see page 15130)

TWebDispatcher (see page 15147)

TWinCGIResponse (see page 2590)

21.1.3.1 TWinCGIRequest Members

The following tables list the members exposed by TWinCGIRequest.

Protected Methods

	Name	Description
V	GetDateVariable (see page 2567)	Provides the abstract (pure virtual) prototype for a method to extract a date value from an indexed table of HTTP header variables.
A	GetDateVariable (see page 15168)	Provides the abstract (pure virtual) prototype for a method to extract a date value from an indexed table of HTTP header variables.
V	GetIntegerVariable (see page 2568)	Provides the abstract (pure virtual) prototype for a method to extract an integer value from an indexed table of HTTP header variables.
A	GetIntegerVariable (see page 15169)	Provides the abstract (pure virtual) prototype for a method to extract an integer value from an indexed table of HTTP header variables.
V	GetInternalPathInfo (see page 15169)	This is GetInternalPathInfo, a member of class TWebRequest.
V	GetInternalScriptName (see page 15169)	This is GetInternalScriptName, a member of class TWebRequest.
V	GetStringVariable (see page 2587)	Provides the abstract (pure virtual) prototype for a method to extract a string value from an indexed table of HTTP header variables.
	UpdateMethodType (see page 15172)	Provides the abstract prototype for a method to write a pure virtual method for updating the client method type.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2586)	Creates an instance of TWinCGIRequest (see page 2583).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2586)	Destroys an instance of TWebRequest.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExtractContentFields (see page 15166)	Parses the Content (see page 15176) property into its constituent fields.
	ExtractCookieFields (see page 15166)	Parses the Cookie (see page 15180) property into its constituent fields.
	ExtractFields (see page 15167)	Parses a multi-valued string into its constituent fields.
	ExtractQueryFields (see page 15167)	Parses the Query (see page 15186) property into its constituent fields.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
V	GetFieldByName (see page 2587)	A specific GetFieldByName method for the TWinCGIRequest (see page 2583) object.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	ReadClient (see page 2588)	A specific method for reading from the HTTP request into a buffer.
V	ReadString (see page 2589)	A specific method for reading Count bytes from the HTTP request into a string.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	TranslateURI (see page 2570)	Provides a method to translate a Uniform Resource Identifier (URI) into an internal representation of a location on the server.
A	TranslateURI (see page 15171)	Provides the abstract (pure virtual) prototype for a method to translate a Uniform Resource Identifier (URI) into an internal representation of a location on the server.
V	WriteClient (see page 2589)	A specific method for writing Count bytes from Buffer back to the HTTP client.
V	WriteHeaders (see page 2571)	Writes an HTTP header string.
A	WriteHeaders (see page 15172)	Provides the abstract (pure virtual) prototype for a method to write an HTTP header string.
V	WriteString (see page 2589)	A specific method for writing the string AString back to the HTTP client.

Public Properties

	Name	Description
R	Accept (see page 15174)	Reports the value of the Accept header of the HTTP request message.
R	Authorization (see page 15175)	Reports the value of the Authorization header of the HTTP request message.
R	CacheControl (see page 15175)	Reports the value of the Cache-Control header of the HTTP request message.
R	Connection (see page 15176)	Reports the value of the Connection header of the HTTP request message.
R	Content (see page 15176)	Represents the unparsed contents of the HTTP request message.
R	ContentEncoding (see page 15177)	Reports the value of the Content (see page 15176)-Encoding header of the HTTP request message.
R	ContentFields (see page 15177)	Reports the parsed content of a POST request message.
R	ContentLength (see page 15178)	Reports the value of the Content (see page 15176)-Length header of the HTTP request message.
R	ContentParser (see page 15178)	Refers to the object's content parser object.
R	ContentType (see page 15179)	Reports the value of the Content (see page 15176)-Type header of the HTTP request message.
R	ContentVersion (see page 15179)	Reports the value of the Content (see page 15176)-Version header of the HTTP request message.
R	Cookie (see page 15180)	Reports the value of the Cookie header of the HTTP request message.
R	CookieFields (see page 15180)	Represents the parsed contents of the Cookie (see page 15180) header of an HTTP request message.
R	Date (see page 15181)	Reports the value of the Date header of the HTTP request message.
R	DerivedFrom (see page 15181)	Reports the value of the Derived-From (see page 15182) header of the HTTP request message.
R	Expires (see page 15182)	Reports the value of the Expires header of the HTTP request message.
R	Files (see page 15182)	Provides access to the files uploaded in the HTTP request.
R	From (see page 15182)	Reports the value of the From header of the HTTP request message.
R	Host (see page 15183)	Reports the value of the Host header of the HTTP request message.
R	IfModifiedSince (see page 15183)	Reports the value of the If-Modified-Since header of the HTTP request message.
R	InternalPathInfo (see page 15184)	Returns the path info portion of the URL (see page 15190).
R	InternalScriptName (see page 15184)	Returns the ScriptName (see page 15189) part of the URL (see page 15190).
R	Method (see page 15185)	This is Method, a member of class TWebRequest.
R	MethodType (see page 15185)	This is MethodType, a member of class TWebRequest.
R	PathInfo (see page 15185)	Reports the value of the path information (if any) of the URL (see page 15190) specified in the HTTP request message.
R	PathTranslated (see page 15186)	Represents a translation of the PathInfo (see page 15185) property to a fully qualified path on the Web server.
R	ProtocolVersion (see page 15186)	Reports the version number of the HTTP protocol specified in the initial header of the HTTP request message.

R	Query (see page 15186)	Indicates the value of the query information (if any) of the URL (see page 15190) specified in the HTTP request message.
R	QueryFields (see page 15187)	Represents the parsed contents of the Query (see page 15186) in the URL (see page 15190) of the HTTP request message.
R	Referer (see page 15188)	Reports the value of the Referer header of the HTTP request message.
R	RemoteAddr (see page 15188)	Indicates the remote IP address of the client associated with the HTTP request message.
R	RemoteHost (see page 15188)	Reports the fully qualified reverse domain lookup of the client associated with the HTTP request message.
R	ScriptName (see page 15189)	Indicates the value of the script information (if any) of the URL (see page 15190) specified in the HTTP request message.
R	ServerPort (see page 15189)	Indicates the port number of the port on the Web server that receives the HTTP request message.
R	Title (see page 15190)	Reports the value of the Title header of the HTTP request message.
R	URL (see page 15190)	Reports the Uniform Resource Identifier (URI) specified by the HTTP request message to identify the target of the request.
R	UserAgent (see page 15191)	Reports the value of the User-Agent header of the HTTP request message.

21.1.3.2 TWinCGIRequest.Create Constructor

Creates an instance of TWinCGIRequest (see page 2583).

Pascal

```
constructor Create(IniFileName: string; ContentFile: string; OutputFile: string);
```

C++

```
__fastcall TWinCGIRequest(AnsiString IniFileName, AnsiString ContentFile, AnsiString
OutputFile);
```

Description

Applications shouldn't need to call Create to instantiate a CGI request object. The TCGIApplication object creates a TWinCGIRequest (see page 2583) object whenever it receives a client request.

See Also

TCGIApplication (see page 2554)

21.1.3.3 TWinCGIRequest.Destroy Destructor

Destroys an instance of TWebRequest.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWinCGIRequest();
```

Description

Do not call Destroy. When the request message is fully handled, TWebApplication destroys the request object by calling its Free method.

Destroy frees the TStrings objects used to represent the ContentFields, CookieFields, and QueryFields properties.

See Also

ContentFields (see page 15177)

CookieFields (see page 15180)

Free (see page 22300)

QueryFields (see page 15187)

TWebApplication (see page 24618)

21.1.3.4 TWinCGIRequest Methods

The methods of the TWinCGIRequest class are listed here.

Protected Methods

	Name	Description
	GetStringVariable (see page 2587)	Provides the abstract (pure virtual) prototype for a method to extract a string value from an indexed table of HTTP header variables.

Public Methods

	Name	Description
	GetFieldByName (see page 2587)	A specific GetFieldByName method for the TWinCGIRequest (see page 2583) object.
	ReadClient (see page 2588)	A specific method for reading from the HTTP request into a buffer.
	ReadString (see page 2589)	A specific method for reading Count bytes from the HTTP request into a string.
	WriteClient (see page 2589)	A specific method for writing Count bytes from Buffer back to the HTTP client.
	WriteString (see page 2589)	A specific method for writing the string AString back to the HTTP client.

21.1.3.4.1 TWinCGIRequest.GetFieldByName Method

A specific GetFieldByName method for the TWinCGIRequest (see page 2583) object.

Pascal

```
function GetFieldByName(const Name: string): string; override;
```

C++

```
virtual __fastcall AnsiString GetFieldByName(const AnsiString Name);
```

Description

GetFieldByName returns the Name field found in the Extra Headers section of the initialization file specified by the application.

See Also

GetFieldByName (see page 15168)

21.1.3.4.2 TWinCGIRequest.GetStringVariable Method

Provides the abstract (pure virtual) prototype for a method to extract a string value from an indexed table of HTTP header variables.

Pascal

```
function GetStringVariable(Index: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetStringVariable(int Index);
```

Description

Descendants of TWebRequest implement GetStringVariable to provide a method by which the string values are read from the header strings of the HTTP request message. The request object uses GetStringVariable to access the properties that represent

strings from the HTTP request headers.

See Also

Accept (see page 15174)

CacheControl (see page 15175)

Connection (see page 15176)

Content (see page 15176)

ContentEncoding (see page 15177)

ContentType (see page 15179)

ContentVersion (see page 15179)

Cookie (see page 15180)

DerivedFrom (see page 15181)

From (see page 15182)

Host (see page 15183)

TWebRequest (see page 15161)

PathInfo (see page 15185)

PathTranslated (see page 15186)

ProtocolVersion (see page 15186)

Query (see page 15186)

Referer (see page 15188)

RemoteAddr (see page 15188)

RemoteHost (see page 15188)

ScriptName (see page 15189)

Title (see page 15190)

URL (see page 15190)

UserAgent (see page 15191)

21.1.3.4.3 TWinCGIRequest.ReadClient Method

A specific method for reading from the HTTP request into a buffer.

Pascal

```
function ReadClient(var Buffer; Count: Integer): Integer; override;
```

C++

```
virtual __fastcall int ReadClient( Buffer, int Count);
```

Description

ReadClient is a specific instance of the method ReadClient (inherited from TWebRequest) for the class TWinCGIRequest (see page 2583). See the documentation for the inherited method for more information.

See Also

ReadClient (see page 15170)

WriteClient (see page 2589)

21.1.3.4.4 TWinCGIRequest.ReadString Method

A specific method for reading Count bytes from the HTTP request into a string.

Pascal

```
function ReadString(Count: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString ReadString(int Count);
```

Description

ReadString is a specific instance of the method ReadString (inherited from TWebRequest) for the class TWinCGIRequest (see page 2583). See the documentation for the inherited method for more information.

See Also

ReadString (see page 15171)

WriteString (see page 15173)

21.1.3.4.5 TWinCGIRequest.WriteClient Method

A specific method for writing Count bytes from Buffer back to the HTTP client.

Pascal

```
function WriteClient(var Buffer; Count: Integer): Integer; override;
```

C++

```
virtual __fastcall int WriteClient( Buffer, int Count);
```

Description

WriteClient is a specific instance of the method WriteClient (inherited from TWebRequest) for the class TWinCGIRequest (see page 2583). See the documentation for the inherited method for more information.

See Also

WriteClient (see page 15172)

ReadClient (see page 2588)

21.1.3.4.6 TWinCGIRequest.WriteString Method

A specific method for writing the string AString back to the HTTP client.

Pascal

```
function WriteString(const AString: string): Boolean; override;
```

C++

```
virtual __fastcall Boolean WriteString(const AnsiString AString);
```

Description

WriteString is a specific instance of the method WriteString (inherited from TWebRequest) for the class TWinCGIRequest (see page 2583). See the documentation for the inherited method for more information.

See Also

WriteString (see page 15173)

ReadString (see page 2589)

21.1.4 TWinCGIResponse Class

TWinCGIResponse represents HTTP messages sent by a Windows hosted CGI application in response to an HTTP request message.

Class Hierarchy

Pascal

```
TWinCGIResponse = class(TCGIResponse);
```

C++

```
class TWinCGIResponse : public TCGIResponse;
```

File

CGIHTTP

Description

Use TWinCGIResponse to specify the response to an HTTP request message.

A CGI application creates a TWinCGIResponse object based on the TWinCGIRequest (see page 2583) object for an incoming web client request message. The Web dispatcher for the application then passes the TWinCGIResponse object to the TWebActionItem associated with the TWinCGIRequest (see page 2583) object. The TWebActionItem sets the properties of the TWinCGIResponse object.

See Also

TCGIApplication (see page 2554)

TWebActionItem (see page 15130)

TWebDispatcher (see page 15147)

TWinCGIRequest (see page 2583)

21.2 Types

The following table lists types in this documentation.

Types

Name	Description
TCGIRequestClass (see page 2591)	This is type CGIHTTP.TCGIRequestClass.
TCGIResponseClass (see page 2591)	This is type CGIHTTP.TCGIResponseClass.
TWinCGIRequestClass (see page 2592)	This is type CGIHTTP.TWinCGIRequestClass.
TWinCGIResponseClass (see page 2592)	This is type CGIHTTP.TWinCGIResponseClass.

21.2.1 CGIHTTP.TCGIRequestClass Type

Pascal

```
TCGIRequestClass = class of TCGIRequest;
```

C++

```
TCGIRequest TCGIRequestClass;
```

File

CGIHTTP

Description

This is type CGIHTTP.TCGIRequestClass.

21.2.2 CGIHTTP.TCGIResponseClass Type

Pascal

```
TCGIResponseClass = class of TCGIResponse;
```

C++

```
TCGIResponse TCGIResponseClass;
```

File

CGIHTTP

Description

This is type CGIHTTP.TCGIResponseClass.

21.2.3 CGIHTTP.TWinCGIRequestClass Type

Pascal

```
TWinCGIRequestClass = class of TWinCGIRequest;
```

C++

```
TWinCGIRequest TWinCGIRequestClass;
```

File

CGIHTTP

Description

This is type CGIHTTP.TWinCGIRequestClass.

21.2.4 CGIHTTP.TWinCGIResponseClass Type

Pascal

```
TWinCGIResponseClass = class of TWinCGIResponse;
```

C++

```
TWinCGIResponse TWinCGIResponseClass;
```

File

CGIHTTP

Description

This is type CGIHTTP.TWinCGIResponseClass.

22 CheckLst Namespace

This is namespace CheckLst.

Topics

Name	Description
Classes (see page 2594)	The following table lists classes in this documentation.

Classes

	Name	Description
	TCheckListBox (see page 2594)	TCheckListBox displays a scrollable list with check boxes next to each item.

22.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCheckListBox (see page 2594)	TCheckListBox displays a scrollable list with check boxes next to each item.

22.1.1 TCheckListBox Class

TCheckListBox displays a scrollable list with check boxes next to each item.

Class Hierarchy

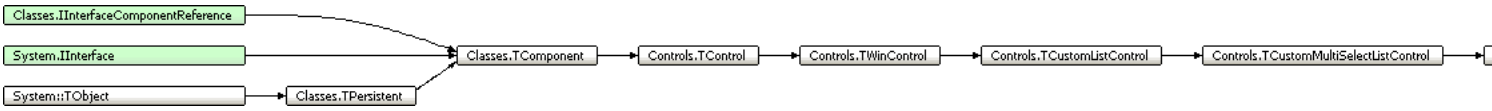

Pascal

```
TCheckListBox = class(TCustomListBox);
```

C++

```
class TCheckListBox : public TCustomListBox;
```

File

CheckLst

Description

TCheckListBox is similar to TListBox, except that each item has a check box next to it. Users can check or uncheck items in the list.

See Also

TCheckBox (see page 20996)

TCustomCheckBox (see page 21138)

TCustomListBox (see page 21308)

TListBox (see page 21598)

List Controls

List Boxes and Check-list Boxes

22.1.1.1 TCheckListBox Members

The following tables list the members exposed by TCheckListBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FMoving (see page 21321)	This is FMoving, a member of class TCustomListBox.
	FMultiSelect (see page 5956)	This is FMultiSelect, a member of class TCustomMultiSelectListControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.

	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ClickCheck (see page 2607)	This is ClickCheck, a member of class TCheckListBox.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 2607)	Initializes the window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 2608)	Creates a Windows control corresponding to the list box component.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DeleteString (see page 2609)	Deletes a string from the list box.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21326)	Saves the list box items in a string list before freeing the underlying Windows control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFindData (see page 21326)	Returns the index of a string in the list box.
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetData (see page 21327)	Returns a string in the list box given its index.
	DoGetDataObject (see page 21327)	Returns the object associated with an item in a virtual list box.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.

	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragCanceled (see page 21328)	Sends a mouse message canceling the drag-and-drop operation.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawItem (see page 2609)	Generates an OnDrawItem event.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetCheckWidth (see page 2609)	This is GetCheckWidth, a member of class TCheckListBox.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetCount (see page 21329)	Returns the value of the Count (see page 21346) property.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetItemData (see page 2610)	Returns a 32-bit value associated with the specified item in a list box.
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
V	GetItemIndex (see page 21329)	Returns the value of the ItemIndex property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
A	GetSelCount (see page 5956)	Returns the value of the SelCount (see page 5957) property.
V	GetSelCount (see page 21330)	Returns the number of selected items.

	GetSelected (see page 21330)	Indicates whether a particular item is selected.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InternalGetItemData (see page 2610)	Retrieves the data associated with an item in the list box.
	InternalSetItemData (see page 2610)	Associates an integer with a specified item in the list box.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 2611)	Responds when the user presses a key.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	LoadRecreateItems (see page 2611)	This is LoadRecreateItems, a member of class TCheckListBox.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MeasureItem (see page 21335)	Generates an OnMeasureItem (see page 21360) event.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 2611)	OnMouseDown event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetContent (see page 2612)	Removes all of the items from a list box.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	SaveRecreateItems (see page 2612)	This is SaveRecreateItems, a member of class TCheckListBox.

	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetItemData (see page 2612)	Sets the 32-bit value associated with the specified item in a list box.
A	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
V	SetItemIndex (see page 21336)	Gives focus to the item with a specified index.
A	SetMultiSelect (see page 5956)	Sets the value of the MultiSelect (see page 5957) property.
V	SetMultiSelect (see page 21337)	Sets the MultiSelect property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.

	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WndProc (see page 21337)	Handles Windows messages.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ExtendedSelect (see page 21346)	This is ExtendedSelect, a member of class TCustomListBox.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddItem (see page 5940)	Adds an item to the list control.
	AddItem (see page 21322)	Adds an item to the list box
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 5940)	Deletes all of the items from the list control.
	Clear (see page 21322)	Deletes all items from the list box.
	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.

V	ClearSelection (see page 21323)	Deselects all selected items in the list box, if possible.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
A	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
V	CopySelection (see page 21323)	Copies the selected items to another list-type control
V	Create (see page 2606)	Instantiates a list box control.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	DeleteSelected (see page 5941)	Deletes all the selected items.
V	DeleteSelected (see page 21325)	Deletes the selected items from the list box
V	Destroy (see page 2606)	Destroys the TCheckListBox (see page 2594) instance and frees its memory.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.

	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ItemAtPos (see page 21331)	Returns the index of the list box item indicated by the coordinates of a point on the control.
	ItemRect (see page 21333)	Returns the rectangle that surrounds the item specified in the Item parameter.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	MoveSelection (see page 5942)	Moves all selected items to another list control.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectAll (see page 5943)	Selects all items or all text in the selected item.

V	SelectAll (see page 21336)	Selects the entire block of text in the list box.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AutoCompleteDelay (see page 21342)	Specifies the delay between a key press and an attempt to autocomplete the field.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 21343)	Provides a drawing surface when implementing a handler for the OnDrawItem (see page 21358) event of an owner-draw list box.
	Checked (see page 2621)	Indicates which items are checked.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Count (see page 21346)	Indicates the number of items in the list box.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.

	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	Header (see page 2628)	Indicates which items act as headers.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemEnabled (see page 2631)	Enables or disables individual items in the list.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	MultiSelect (see page 5957)	Specifies whether the user can select more than one item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ScrollWidth (see page 21351)	Specifies the width, in pixels, by which the list box can scroll horizontally.
R	SelCount (see page 5957)	Indicates the number of selected items.
	Selected (see page 21351)	Indicates whether a particular item is selected.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	State (see page 2645)	Indicates which items are checked.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TopIndex (see page 21356)	Specifies the index number of the item that appears at the top of the list box.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 2649)	Occurs when the user clicks the control.
	OnClickCheck (see page 2651)	Occurs when the user selects or deselects an item's check box.
	OnContextPopup (see page 2652)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnData (see page 2652)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 2653)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 2653)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDblClick (see page 2654)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2654)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2656)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 2659)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnEndDock (see page 2661)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2661)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2662)	Occurs when a control receives the input focus.
	OnExit (see page 2665)	Occurs when the input focus shifts away from one control to another.

	OnKeyDown (see page 2666)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2669)	Occurs when a key is pressed.
	OnKeyUp (see page 2670)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 2672)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.
	OnMouseActivate (see page 2673)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2674)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2677)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2677)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2677)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2679)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2682)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2682)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 2614)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AllowGrayed (see page 2617)	Determines whether check box can be in a "grayed" state.
	Anchors (see page 2618)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 2618)	Determines whether the user can give focus to items by typing in the list.
	BevelEdges (see page 2619)	Specifies which edges of the control are beveled.
	BevelInner (see page 2619)	Specifies the cut of the inner bevel.
	BevelKind (see page 2619)	Specifies the control's bevel style.
	BevelOuter (see page 2620)	Specifies the cut of the outer bevel.
	BevelWidth (see page 2620)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 2621)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2621)	Determines whether the list box has a border.
	Color (see page 2622)	Specifies the background color of the control.
	Columns (see page 2623)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	Constraints (see page 2623)	Specifies the size constraints for the control.
	Ctl3D (see page 2624)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 2624)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2625)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2625)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2627)	Controls whether the control responds to mouse, keyboard, and timer events.
	Flat (see page 2627)	Determines whether the check boxes appear three-dimensional.
	Font (see page 2627)	Controls the attributes of text written on or in the control.
	HeaderBackgroundColor (see page 2629)	Specifies the background color for a header item in the listbox.
	HeaderColor (see page 2629)	Specifies the font color for a header item in the list box.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 2630)	Determines the behavior of the input method editor (IME).

	ImeName (see page 2630)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 2631)	Determines whether the list box displays the partial items.
	ItemHeight (see page 2632)	Specifies the height in pixels of an item in an owner-draw list box.
	Items (see page 2632)	Contains the strings that appear in the list box.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 2635)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2635)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2636)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 2636)	Determines where a control looks for its font information.
	ParentShowHint (see page 2637)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2637)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2642)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Sorted (see page 2643)	Specifies whether the items in a list box are arranged alphabetically.
	Style (see page 2645)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabOrder (see page 2646)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2647)	Determines if the user can tab to a control.
	TabWidth (see page 2647)	Specifies the size of the tabs in the list box.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 2647)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

22.1.1.2 TCheckListBox.Create Constructor

Instantiates a list box control.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCheckListBox(TComponent * AOwner);
```

Description

Call Create to instantiate a list box control at runtime. For list boxes created at design time, Create is called automatically.AOwner is the component that is responsible for freeing the list box. It becomes the value of the Owner property. If AOwner is nil, the caller is responsible for explicitly freeing the control.

See Also

Owner (see page 2807)

22.1.1.3 TCheckListBox.Destroy Destructor

Destroys the TCheckListBox (see page 2594) instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCheckListBox();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free verifies that the check list box is not nil before it calls Destroy.

See Also

Free (see page 22300)

22.1.1.4 TCheckListBox Methods

The methods of the TCheckListBox class are listed here.

Protected Methods

	Name	Description
	ClickCheck (see page 2607)	This is ClickCheck, a member of class TCheckListBox.
V	CreateParams (see page 2607)	Initializes the window-creation parameter record passed in the Params parameter.
V	CreateWnd (see page 2608)	Creates a Windows control corresponding to the list box component.
V	DeleteString (see page 2609)	Deletes a string from the list box.
V	DrawItem (see page 2609)	Generates an OnDrawItem event.
	GetCheckWidth (see page 2609)	This is GetCheckWidth, a member of class TCheckListBox.
V	GetItemData (see page 2610)	Returns a 32-bit value associated with the specified item in a list box.
V	InternalGetItemData (see page 2610)	Retrieves the data associated with an item in the list box.
V	InternalSetItemData (see page 2610)	Associates an integer with a specified item in the list box.
V	KeyPress (see page 2611)	Responds when the user presses a key.
V	LoadRecreateItems (see page 2611)	This is LoadRecreateItems, a member of class TCheckListBox.
V	MouseDown (see page 2611)	OnMouseDown event dispatcher.
V	ResetContent (see page 2612)	Removes all of the items from a list box.
V	SaveRecreateItems (see page 2612)	This is SaveRecreateItems, a member of class TCheckListBox.
V	SetItemData (see page 2612)	Sets the 32-bit value associated with the specified item in a list box.

22.1.1.4.1 TCheckListBox.ClickCheck Method

Pascal

```
procedure ClickCheck; dynamic;
```

C++

```
__fastcall ClickCheck();
```

Description

This is ClickCheck, a member of class TCheckListBox.

22.1.1.4.2 TCheckListBox.CreateParams Method

Initializes the window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

CreateParams is used to pass information, in its Params parameter, to CreateWindowHandle when the list box control is created.

TCustomListBox overrides CreateParams to customize the representation of the window. It uses information from the list box Columns, ExtendedSelect, IntegralHeight, MultiSelect, Sorted and Style properties and sets WS_HSCROLL, WS_VSCROLL, LBS_HASSTRINGS and LBS_NOTIFY Windows styles for creation of the list box.

When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

See Also

Columns (see page 21346)

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

extendedselect (see page 21346)

IntegralHeight (see page 21347)

MultiSelect

Sorted (see page 21353)

Style (see page 21354)

22.1.1.4.3 TCheckListBox.CreateWnd Method

Creates a Windows control corresponding to the list box component.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the list box must be displayed. CreateWnd calls the inherited CreateWnd method, which calls

CreateParams to initialize the window-creation parameters.

CreateWindowHandle to create the window handle for the control.

CreateWnd additionally sets up the list box position, tabs, items, and columns.

See Also

Columns (see page 21346)

CreateHandle (see page 6135)

CreateParams (see page 21324)

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

ItemIndex

Items (see page 21348)

TabWidth (see page 21355)

TopIndex (see page 21356)

22.1.1.4.4 TCheckListBox.DeleteString Method

Deletes a string from the list box.

Pascal

```
procedure DeleteString(Index: Integer); override;
```

C++

```
virtual __fastcall DeleteString(int Index);
```

Description

DeleteString deletes the string, specified by the Index parameter, from the list box control.

22.1.1.4.5 TCheckListBox.DrawItem Method

Generates an OnDrawItem event.

Pascal

```
procedure DrawItem(Index: Integer; Rect: TRect; State: TOwnerDrawState); override;
```

C++

```
virtual __fastcall DrawItem(int Index, TRect Rect, TOwnerDrawState State);
```

Description

DrawItem is called whenever a visual aspect of an owner-draw list box changes. The list box calls DrawItem for each visible item in its list, passing the index of the item in the Index parameter.

By default, the DrawItem method for a list box calls any event handler attached to the component's OnDrawItem event. If there is no handler attached, DrawItem fills the rectangle passed in Rect and draws any text associated with the indexed item. The default drawing ignores the State parameter, although State is passed along to attached event handlers.

See Also

ItemHeight (see page 21347)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

Style (see page 21354)

22.1.1.4.6 TCheckListBox.GetCheckWidth Method

Pascal

```
function GetCheckWidth: Integer;
```

C++

```
__fastcall int GetCheckWidth();
```

Description

This is GetCheckWidth, a member of class TCheckListBox.

22.1.1.4.7 TCheckListBox.GetItemData Method

Returns a 32-bit value associated with the specified item in a list box.

Pascal

```
function GetItemData(Index: Integer): LongInt; override;
```

C++

```
virtual __fastcall LongInt GetItemData(int Index);
```

Description

GetItemData retrieves the data associated with a particular item in the list box. This value can be displayed as that list box item, or can be a value (such as a pointer) associated with the list box item.

See Also

SetItemData (see page 21336)

22.1.1.4.8 TCheckListBox.InternalGetItemData Method

Retrieves the data associated with an item in the list box.

Pascal

```
function InternalGetItemData(Index: Integer): Longint; override;
```

C++

```
virtual __fastcall Longint InternalGetItemData(int Index);
```

Description

InternalGetItemData is the low-level protected method for fetching the data associated with an item in the list box. It retrieves a reference pointer to the object associated with the specified item, as an integer.

Index is the index of the item whose data is to be fetched.

See Also

GetItemData (see page 21329)

22.1.1.4.9 TCheckListBox.InternalSetItemData Method

Associates an integer with a specified item in the list box.

Pascal

```
procedure InternalSetItemData(Index: Integer; AData: Longint); override;
```

C++

```
virtual __fastcall InternalSetItemData(int Index, Longint AData);
```

Description

InternalSetItemData is the low-level protected method for associating data with an item in the list box. It associates the item that has the index specified by Index with the integer specified by AData.

See Also

SetItemData (see page 21336)

22.1.1.4.10 TCheckListBox.KeyPress Method

Responds when the user presses a key.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress is called automatically when the list box receives a key-press message (WM_CHAR). It generates an OnKeyPress event, and then, if the AutoComplete property is true, updates ItemIndex to reflect the next character that the user typed.

The Key parameter is the character represented by the key that is pressed, not a Windows virtual key code.

See Also

OnKeyPress (see page 6227)

22.1.1.4.11 TCheckListBox.LoadRecreateItems Method

Pascal

```
procedure LoadRecreateItems(RecreateItems: TStrings); override;
```

C++

```
virtual __fastcall LoadRecreateItems(TStrings RecreateItems);
```

Description

This is LoadRecreateItems, a member of class TCheckListBox.

22.1.1.4.12 TCheckListBox.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove (see page 5773)

MouseUp (see page 5774)

OnMouseDown (see page 5862)

22.1.1.4.13 TCheckListBox.ResetContent Method

Removes all of the items from a list box.

Pascal

```
procedure ResetContent; override;
```

C++

```
virtual __fastcall ResetContent();
```

Description

ResetContent is used to remove the items from the list box. ResetContent is used, for example, by the Clear method.

See Also

Clear (see page 21322)

22.1.1.4.14 TCheckListBox.SaveRecreateItems Method

Pascal

```
procedure SaveRecreateItems(RecreateItems: TStrings); override;
```

C++

```
virtual __fastcall SaveRecreateItems(TStrings RecreateItems);
```

Description

This is SaveRecreateItems, a member of class TCheckListBox.

22.1.1.4.15 TCheckListBox.SetItemData Method

Sets the 32-bit value associated with the specified item in a list box.

Pascal

```
procedure SetItemData(Index: Integer; AData: LongInt); override;
```

C++

```
virtual __fastcall SetItemData(int Index, LongInt AData);
```

Description

Use SetItemData to set the data associated with a particular item in a list box. This value can be displayed as that list box item, or it can be a value (such as a pointer) represented by the list box item.

See Also

GetItemData (see page 21329)

22.1.1.5 TCheckListBox Properties

The properties of the TCheckListBox class are listed here.

Public Properties

	Name	Description
	Checked (see page 2621)	Indicates which items are checked.
	Header (see page 2628)	Indicates which items act as headers.
	ItemEnabled (see page 2631)	Enables or disables individual items in the list.
	State (see page 2645)	Indicates which items are checked.

Published Properties

	Name	Description
	Align (see page 2614)	Determines how the control aligns within its container (parent control).
	AllowGrayed (see page 2617)	Determines whether check box can be in a "grayed" state.
	Anchors (see page 2618)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 2618)	Determines whether the user can give focus to items by typing in the list.
	BevelEdges (see page 2619)	Specifies which edges of the control are beveled.
	BevelInner (see page 2619)	Specifies the cut of the inner bevel.
	BevelKind (see page 2619)	Specifies the control's bevel style.
	BevelOuter (see page 2620)	Specifies the cut of the outer bevel.
	BevelWidth (see page 2620)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 2621)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2621)	Determines whether the list box has a border.
	Color (see page 2622)	Specifies the background color of the control.
	Columns (see page 2623)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	Constraints (see page 2623)	Specifies the size constraints for the control.
	Ctl3D (see page 2624)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 2624)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2625)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2625)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2627)	Controls whether the control responds to mouse, keyboard, and timer events.
	Flat (see page 2627)	Determines whether the check boxes appear three-dimensional.
	Font (see page 2627)	Controls the attributes of text written on or in the control.
	HeaderBackgroundColor (see page 2629)	Specifies the background color for a header item in the listbox.
	HeaderColor (see page 2629)	Specifies the font color for a header item in the list box.
	ImeMode (see page 2630)	Determines the behavior of the input method editor (IME).
	ImeName (see page 2630)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 2631)	Determines whether the list box displays the partial items.
	ItemHeight (see page 2632)	Specifies the height in pixels of an item in an owner-draw list box.
	Items (see page 2632)	Contains the strings that appear in the list box.
	ParentBiDiMode (see page 2635)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2635)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2636)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 2636)	Determines where a control looks for its font information.
	ParentShowHint (see page 2637)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2637)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2642)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Sorted (see page 2643)	Specifies whether the items in a list box are arranged alphabetically.

	Style (see page 2645)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabOrder (see page 2646)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2647)	Determines if the user can tab to a control.
	TabWidth (see page 2647)	Specifies the size of the tabs in the list box.
	Visible (see page 2647)	Determines whether the component appears onscreen.

22.1.1.5.1 TCheckListBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);
```

```
  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
```

```
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
```

```
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

22.1.1.5.2 TCheckListBox.AllowGrayed Property

Determines whether check box can be in a "grayed" state.

Pascal

```
property AllowGrayed: Boolean;
```

C++

```
__property Boolean AllowGrayed;
```

Description

If AllowGrayed is set to true, each item has three possible states: checked, unchecked, and grayed. If AllowGrayed is set to false, each item has only two possible states: checked and unchecked.

Note: It is also possible to disable individual items while still showing whether they are checked or unchecked. To disable items without using the grayed state, use the ItemEnabled (see page 2631) property.

See Also

Checked (see page 2621)

State (see page 2645)

ItemEnabled (see page 2631)

22.1.1.5.3 TCheckListBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

22.1.1.5.4 TCheckListBox.AutoComplete Property

Determines whether the user can give focus to items by typing in the list.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

Use AutoComplete to specify whether the list box automatically updates the ItemIndex property in response to user keystrokes. When AutoComplete is true, the item with focus changes to the first item that begins with the sequence of characters that the user typed. As the user types more characters, ItemIndex may change to a later item because the string to match becomes longer.

See Also

ItemIndex

22.1.1.5.5 TCheckListBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

22.1.1.5.6 TCheckListBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

22.1.1.5.7 TCheckListBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

22.1.1.5.8 TCheckListBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

22.1.1.5.9 TCheckListBox.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

22.1.1.5.10 TCheckListBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

22.1.1.5.11 TCheckListBox.BorderStyle Property

Determines whether the list box has a border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to change the appearance of the outer boundary of the list box control. A list box can have a single border (bsSingle) or no visible border (bsNone).

See Also

Ctl3D (see page 6198)

22.1.1.5.12 TCheckListBox.Checked Property

Indicates which items are checked.

Pascal

```
property Checked [Index: Integer]: Boolean;
```

C++

```
__property Boolean Checked[int Index];
```

Description

For each member of the Items (see page 2632) array, Checked is true if a check mark appears in the item's check box. Checked corresponds to the cbChecked state; that is, Checked is true when State (see page 2645) = cbChecked and false when State (see page 2645) = cbUnchecked or State (see page 2645) = cbGrayed.

Note: Items (see page 2632) that have been disabled using the ItemEnabled (see page 2631) property may still have a state of cbChecked and a Checked property of true.

See Also

AllowGrayed (see page 2617)

Items (see page 21348)

Selected (see page 21351)

State (see page 2645)

ItemEnabled (see page 2631)

22.1.1.5.13 TCheckListBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

22.1.1.5.14 TCheckListBox.Columns Property

Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.

Pascal

```
property Columns: Integer;
```

C++

```
__property int Columns;
```

Description

Use Columns to specify the number of columns, in a multi-column list box, that are visible without having to use the horizontal scroll bar.

Multi-column list boxes have a horizontal scroll bar that allows users to view multiple columns as they wrap. The default value for Columns is 0, meaning that the list box is not multi-column. That is, users can scroll only vertically and the list of items will not wrap.

For Columns values greater than 0, multiple columns accommodate the items as they wrap beyond the bottom of the list box. The Columns property specifies the number of columns that are visible without having to horizontally scroll the list box.

The width of each column depends upon both the Width property and the number of Columns.

See Also

ItemHeight (see page 21347)

Width (see page 5849)

22.1.1.5.15 TCheckListBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

22.1.1.5.16 TCheckListBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

22.1.1.5.17 TCheckListBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

22.1.1.5.18 TCheckListBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

22.1.1.5.19 TCheckListBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
```

```
      return;
    THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
    TNodeAttachMode AttachMode;
    TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
    if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
      AttachMode = naAddChild;
    else if (HT.Contains(htNowhere))
      AttachMode = naAdd;
    else if (HT.Contains(htOnIndent))
      AttachMode = naInsert;
    else
      return;
    TreeView1->Selected->MoveTo(Item, AttachMode);
  }

  void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
        int X, int Y, TDragState State, bool &Accept)
  {
    Accept = (Source == (TObject *)TreeView1);
  }
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

22.1.1.5.20 TCheckListBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

22.1.1.5.21 TCheckListBox.Flat Property

Determines whether the check boxes appear three-dimensional.

Pascal

```
property Flat: Boolean;
```

C++

```
__property Boolean Flat;
```

Description

When Flat is true (the default), the check boxes in the list box have flat borders. Setting Flat to false gives the check boxes raised borders, which create a three-dimensional look.

See Also

Ctl3D (see page 6198)

22.1.1.5.22 TCheckListBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

22.1.1.5.23 TCheckListBox.Header Property

Indicates which items act as headers.

Pascal

```
property Header [Index: Integer]: Boolean;
```

C++

```
__property Boolean Header[int Index];
```

Description

Set Header to true to specify that an item in the check listbox acts as a header rather than a selectable item. By setting Header to true for various items in the list, you can divide the check listbox into multiple sections, where each header labels the beginning of a section.

When Header is true for an item, the list box does not display a check box next to the item, but rather, it displays only the item's text. This item is drawn with a background color of HeaderBackgroundColor (see page 2629) and a font color of HeaderColor (see page 2629), which can be used to make it visually distinct from the checkable items in the list box.

Index is the index of the item.

See Also

HeaderColor (see page 2629)

HeaderBackgroundColor (see page 2629)

22.1.1.5.24 TCheckListBox.HeaderBackgroundColor Property

Specifies the background color for a header item in the listbox.

Pascal

```
property HeaderBackgroundColor: TColor;
```

C++

```
__property TColor HeaderBackgroundColor;
```

Description

Set HeaderBackgroundColor to indicate the color that appears as the background of header items. Header (see page 2628) items are items in the list box for which the Header (see page 2628) property is true. HeaderBackgroundColor can make a header item visually more distinct from the checkable items in the list box.

See Also

Header (see page 2628)

HeaderColor (see page 2629)

22.1.1.5.25 TCheckListBox.HeaderColor Property

Specifies the font color for a header item in the list box.

Pascal

```
property HeaderColor: TColor;
```

C++

```
__property TColor HeaderColor;
```

Description

Set HeaderColor to indicate the font color of header items. Header (see page 2628) items are items in the list box for which the Header (see page 2628) property is true. HeaderColor should be a color that provides contrast with the

HeaderBackgroundColor (see page 2629) property, which specifies the background color for header items.

See Also

Header (see page 2628)

HeaderBackgroundColor (see page 2629)

22.1.1.5.26 TCheckListBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

22.1.1.5.27 TCheckListBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently

installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

22.1.1.5.28 TCheckListBox.IntegralHeight Property

Determines whether the list box displays the partial items.

Pascal

```
property IntegralHeight: Boolean;
```

C++

```
__property Boolean IntegralHeight;
```

Description

IntegralHeight controls the height of the list box itself.

When IntegralHeight is true, and the Style property is lsOwnerDrawFixed, the list box height will always be a multiple of the ItemHeight property value. In this case, items will never be partially displayed.

If IntegralHeight is false, the list box height can be set to any value, but the last item may not be fully displayed.

If the Style property value of the list box is lbOwnerDrawVariable, setting the IntegralHeight property to true has no effect.

See Also

ItemHeight (see page 21347)

Items (see page 21348)

Style (see page 21354)

Using TDBListBox and TDBComboBox

22.1.1.5.29 TCheckListBox.ItemEnabled Property

Enables or disables individual items in the list.

Pascal

```
property ItemEnabled [Index: Integer]: Boolean;
```

C++

```
__property Boolean ItemEnabled[int Index];
```

Description

Use ItemEnabled to enable or disable individual list items without using the cbGrayed state. Using ItemEnabled allows the list box to display check marks for items with a state of cbChecked while still graying the item and preventing the user from changing its state.

For each member of the Items (see page 2632) array, ItemEnabled is true if the user can change the state of the item. Enabled (see page 2627) items have a state of cbChecked or cbUnchecked.

ItemEnabled is false when State (see page 2645) =cbGrayed or when the item has been explicitly disabled using this property.

See Also

State (see page 2645)

Checked (see page 2621)

22.1.1.5.30 TCheckListBox.ItemHeight Property

Specifies the height in pixels of an item in an owner-draw list box.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Use ItemHeight to control the height of an item in a list box when the Style property is lbOwnerDrawFixed. The ItemHeight property can also affect the height of the list box itself. When the IntegralHeight property is true, the height of the list box is always an increment of the ItemHeight value.

If the Style property is lbStandard or lbOwnerDrawVariable, the value of ItemHeight is ignored.

See Also

DrawItem (see page 21328)

IntegralHeight (see page 21347)

Items (see page 21348)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

Style (see page 21354)

Using TDBListBox and TDBComboBox

22.1.1.5.31 TCheckListBox.Items Property

Contains the strings that appear in the list box.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to add, insert, delete and move items. By default, the items in a list box are of type TStrings. Use this item type to access its methods or properties to manipulate the items in the list.

For example, the following code adds the text in the edit box to the list box as an item:

```
ListBox1.Items.Add(Edit1.Text); {Delphi }
ListBox1->Items->Add(Edit1->Text); // C++
```

C++ Examples:

```
/*
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Continue will continue listing
file sizes in a multiple selection that includes the deleted
file.  Break will stop.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
*/
#include <stdio.h>     // for FILE, fopen, fstat, fileno, fread and fclose
#include <sys\stat.h>  // for fstat and the stat struct

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < FileListBox1->Items->Count; i++)
  {
    if (FileListBox1->Selected[i])
    {
      if (!FileExists(FileListBox1->Items->Strings[i]))
      {
        MessageBeep(0);
        if (Application->MessageBox(
          AnsiString("File " + FileListBox1->Items->Strings[i] + "not found").c_str(),
          NULL,
          MB_OKCANCEL) == IDCANCEL)
          break;
        else
          continue;
      }
      FILE *F = fopen(FileListBox1->Items->Strings[i].c_str(),"r");
      struct stat statbuf;
      fstat(fileno(F), &statbuf);
      ListBox1->Items->Add(
          FileListBox1->Items->Strings[i] + ": " + IntToStr(statbuf.st_size));
      fclose(F);
    }
  }
}
/*
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
list of strings in the list box. When the application runs
and the button is clicked, the strings in the list box
become uppercase.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int I = 0; I < ListBox1->Items->Count; I++)
```

```
    ListBox1->Items->Strings[I] = UpperCase(ListBox1->Items->Strings[I]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");
}
```

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], 1);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode 1.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);

      AssignFile(F, FileListBox1.Items.Strings[i]);
      Reset(F, 1);
      ListBox1.Items.Add(
        FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
      CloseFile(F);
    end;
   finally
   { do something here }
   end;
  end;
end;
{
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
list of strings in the list box. When the application runs
and the button is clicked, the strings in the list box
become uppercase.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  I: Integer;
begin
  for I := 0 to ListBox1.Items.Count -1 do
    ListBox1.Items[I] := SysUtils.UpperCase(ListBox1.Items[I]);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  Listbox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;
```

See Also

Clear (see page 21322)

ItemIndex

OnDrawItem (see page 21358)

Selected (see page 21351)

TStrings (see page 3029)

22.1.1.5.32 TCheckListBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

22.1.1.5.33 TCheckListBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

22.1.1.5.34 TCheckListBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

22.1.1.5.35 TCheckListBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

22.1.1.5.36 TCheckListBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

22.1.1.5.37 TCheckListBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
```

```
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
```

```
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
```

```
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}
```

```
type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;
```

```
procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

22.1.1.5.38 TCheckListBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
```

```
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

22.1.1.5.39 TCheckListBox.Sorted Property

Specifies whether the items in a list box are arranged alphabetically.

Pascal

```
property Sorted: Boolean;
```

C++

```
__property Boolean Sorted;
```

Description

Use Sorted to sort the items by setting its value to true. If Sorted is false, the items are unsorted. When Sorted is true, items are automatically added or inserted into the list box in alphabetical order.

Note: Sorted has no effect if Listbox's style is lbVirtual or lbVirtualOwnerDraw

C++ Examples:

```
/*
This example uses an edit box, a list box, and two buttons
on a form. The buttons are named Add and Sort. When the user
clicks the Add button, the text in the edit box is added to
the list in the list box. Before the user clicks the sort
button, any new items are added to the end of the list.
After the user clicks the Sort button, the list in the list
box is sorted and remains sorted, even if additional strings
are added.  If the user clicks the sort button again, the
list box remains sorted, but any new strings are again added
to the end of the list.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");
}

void __fastcall TForm1::AddClick(TObject *Sender)
{
  ListBox1->Items->Add(Edit1->Text);
}

void __fastcall TForm1::SortClick(TObject *Sender)
{
  ListBox1->Sorted = True;
}
```

Delphi Examples:

```
{
This example uses an edit box, a list box, and two buttons
on a form. The buttons are named Add and Sort. When the user
clicks the Add button, the text in the edit box is added to
the list in the list box. Before the user clicks the sort
button, any new items are added to the end of the list.
After the user clicks the Sort button, the list in the list
box is sorted and remains sorted, even if additional strings
are added.  If the user clicks the sort button again, the
list box remains sorted, but any new strings are again added
to the end of the list.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;


procedure TForm1.AddClick(Sender: TObject);
begin
  ListBox1.Items.Add(Edit1.Text);
end;

procedure TForm1.SortClick(Sender: TObject);
begin
  ListBox1.Sorted := True;
```

```
end;
```

See Also

Items (see page 21348)

Style (see page 21354)

22.1.1.5.40 TCheckListBox.State Property

Indicates which items are checked.

Pascal

```
property State [Index: Integer]: TCheckBoxState;
```

C++

```
__property TCheckBoxState State[int Index];
```

Description

For each member of the Items (see page 2632) array, State indicates whether its check box is selected (cbChecked), deselected (cbUnchecked), or grayed (cbGrayed).

State combines the information provided by the Boolean Checked (see page 2621) property and the Boolean ItemEnabled (see page 2631) property:

When Checked (see page 2621) is true, State is cbChecked. (However, when Checked (see page 2621) is false, State may be either cbUnchecked or cbGrayed.)

When State is cbGrayed, ItemEnabled (see page 2631) is false. (However, ItemEnabled (see page 2631) may also be false when State is cbUnchecked or cbChecked.)

See Also

AllowGrayed (see page 2617)

Checked (see page 2621)

Items (see page 21348)

Selected (see page 21351)

State (see page 21153)

ItemEnabled (see page 2631)

22.1.1.5.41 TCheckListBox.Style Property

Determines whether the list box is standard or owner-draw and whether it is virtual.

Pascal

```
property Style: TListBoxStyle;
```

C++

```
__property TListBoxStyle Style;
```

Description

Use Style to specify whether the list box is a standard list box that displays text strings, or an owner-draw list box that can display graphical images. Owner-draw list boxes let you display items in some nonstandard way. In this case, you must write the code to paint items in the list box.

In addition, Style controls whether the list box is virtual, which means that items are supplied dynamically using event handlers. By default, Style is lbStandard, meaning that the list box is not virtual and it displays each item as a string.

See Also

DrawItem (see page 21328)

ItemHeight (see page 21347)

Items (see page 21348)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

OnData (see page 21357)

OnDataFind (see page 21357)

OnDataObject (see page 21358)

22.1.1.5.42 TCheckListBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

22.1.1.5.43 TCheckListBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

22.1.1.5.44 TCheckListBox.TabWidth Property

Specifies the size of the tabs in the list box.

Pascal

```
property TabWidth: Integer;
```

C++

```
__property int TabWidth;
```

Description

Set TabWidth to control the number of dialog base units displayed for each tab character that appears in the strings of a list box.

If TabWidth is set to 0, which is the default setting, the tabs default to the width specified by Windows, which is two dialog base units.

22.1.1.5.45 TCheckListBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
```

```
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
```

```
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

22.1.1.6 TCheckListBox Events

The events of the TCheckListBox class are listed here.

Published Events

	Name	Description
	OnClick (see page 2649)	Occurs when the user clicks the control.
	OnClickCheck (see page 2651)	Occurs when the user selects or deselects an item's check box.
	OnContextPopup (see page 2652)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnData (see page 2652)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 2653)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 2653)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDblClick (see page 2654)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2654)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2656)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 2659)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnEndDock (see page 2661)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2661)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2662)	Occurs when a control receives the input focus.
	OnExit (see page 2665)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2666)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2669)	Occurs when a key is pressed.
	OnKeyUp (see page 2670)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 2672)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.
	OnMouseActivate (see page 2673)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2674)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2677)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2677)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2677)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2679)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2682)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2682)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

22.1.1.6.1 TCheckListBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
```

```
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

22.1.1.6.2 TCheckListBox.OnClickCheck Event

Occurs when the user selects or deselects an item's check box.

Pascal

```
property OnClickCheck: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClickCheck;
```

Description

Write an OnClickCheck event handler to implement any special processing that should occur when the user checks or unchecks an item.

Note: This event does not occur for items that are grayed (State (see page 2645)=cbGrayed) or disabled (ItemEnabled (see page 2631)=false).

See Also

Checked (see page 2621)

State (see page 2645)

ItemEnabled (see page 2631)

22.1.1.6.3 TCheckListBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

22.1.1.6.4 TCheckListBox.OnData Event

Occurs when a virtual list box needs to supply the text of an item.

Pascal

```
property OnData: TLBGetDataEvent;
```

C++

```
__property TLBGetDataEvent OnData;
```

Description

You must supply an OnData event handler to dynamically supply the items in the list box when the Style property is lbVirtual or lbVirtualOwnerDraw.

See Also

OnDataFind (see page 21357)

OnDataObject (see page 21358)

Style (see page 21354)

Count (see page 21346)

22.1.1.6.5 TCheckListBox.OnDataFind Event

Occurs when a virtual list box needs to identify the index of an item given its text.

Pascal

```
property OnDataFind: TLBFindDataEvent;
```

C++

```
__property TLBFindDataEvent OnDataFind;
```

Description

You must supply an OnDataFind event handler to dynamically provide the mapping from list item to index when the Style property is lbVirtual or lbVirtualOwnerDraw.

See Also

OnData (see page 21357)

OnDataObject (see page 21358)

Style (see page 21354)

Count (see page 21346)

22.1.1.6.6 TCheckListBox.OnDataObject Event

Occurs when a virtual list box needs to supply the object associated with an item.

Pascal

```
property OnDataObject: TLBGetDataObjectEvent;
```

C++

```
__property TLBGetDataObjectEvent OnDataObject;
```

Description

Supply an OnDataObject event handler to dynamically provide objects associated with the items in a virtual list box.

See Also

OnData (see page 21357)

OnDataFind (see page 21357)

Style (see page 21354)

Count (see page 21346)

22.1.1.6.7 TCheckListBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

22.1.1.6.8 TCheckListBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
```

```
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
```

```
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

22.1.1.6.9 TCheckListBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
```

```
    TForm1(Source).Top := Y;
  end;

  procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code comes from an application that contains a list box
  and three labels, each with a different font and color.  The
  DragMode property for each of the labels is dmAutomatic.
  The user can select a label and drag it to a list box and
  drop it. When the label is dropped, the items in the list
  box assume the color and font of the dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

22.1.1.6.10 TCheckListBox.OnDrawItem Event

Occurs when an item in an owner-draw list box needs to be redisplayed.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

Use OnDrawItem to write a handler for drawing of the items in list boxes with the Style values lbOwnerDrawFixed, lbOwnerDrawVariable, or lbVirtualOwnerDraw. OnDrawItem occurs when the list box needs to display an item. OnDrawItem occurs only for owner-draw list boxes.

The size of the rectangle that contains the item is determined either by the ItemHeight property for fixed owner-draw list boxes or by the response to the OnMeasureItem event for variable owner-draw list boxes.

C++ Examples:

```
/*
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyList = new TStringList;
  bitmap0 = new Graphics::TBitmap;
  bitmap1 = new Graphics::TBitmap;
  bitmap2 = new Graphics::TBitmap;
//  MyList->Add("Animal");
//  MyList->Add("Flowers");
//  MyList->Add("Butterfly");
//  ListBox1.Items->AddStrings(MyList);

  ImageList1->GetBitmap(0, bitmap0);
  ListBox1->Items->AddObject("Flowers", bitmap0);
  ImageList1->GetBitmap(1, bitmap1);
  ListBox1->Items->AddObject("Animal", bitmap1);
  ImageList1->GetBitmap(2, bitmap2);
  ListBox1->Items->AddObject("Butterfly", bitmap2);
}

void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
      TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBitmap; // temporary variable for item's bitmap
```

```
  int     Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *pCanvas = ((TListBox *)Control)->Canvas;
  pCanvas->FillRect(Rect); // clear the rectangle
  pBitmap = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
  if (pBitmap)
  {
    pCanvas->BrushCopy(Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
pBitmap, Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
    Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  pCanvas->TextOut(Rect.Left + Offset, Rect.Top, ((TListBox *)Control)->Items->Strings[Index]);
}
```

Delphi Examples:

```
{
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  MyList := TStringList.Create;
  bitmap0 := TBitmap.Create;
  bitmap1 := TBitmap.Create;
  bitmap2 := TBitmap.Create;
//  MyList.Add('Animal');
//  MyList.Add('Flowers');
//  MyList.Add('Butterfly');
//  ListBox1.Items.AddStrings(MyList);

  ImageList1.GetBitmap(0, bitmap0);
  ListBox1.Items.AddObject('Flowers', bitmap0);
  ImageList1.GetBitmap(1, bitmap1);
  ListBox1.Items.AddObject('Animal', bitmap1);
  ImageList1.GetBitmap(2, bitmap2);
  ListBox1.Items.AddObject('Butterfly', bitmap2);
end;

procedure TForm1.ListBox1DrawItem(Control: TWinControl; Index: Integer; Rect:TRect; State:
TOwnerDrawState);
var
  Bitmap: TBitmap;      { temporary variable for the item's bitmap }
  Offset: Integer;      { text offset width }
begin
  Bitmap := TBitmap.Create;
  with (Control as TListBox).Canvas do  { draw on control canvas, not on the form }
  begin
    FillRect(Rect);       { clear the rectangle }
    Offset := 2;          { provide default offset }
    Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
    if Bitmap <> nil then
    begin
      BrushCopy(
        Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
        Bitmap,
        Bounds(0, 0, Bitmap.Width, Bitmap.Height),
        clRed);  {render bitmap}
      Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
    end;
```

```
    TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
text }
  end;
end;
```

See Also

DrawItem (see page 21328)

ItemHeight (see page 21347)

MeasureItem (see page 21335)

OnMeasureItem (see page 21360)

Style (see page 21354)

22.1.1.6.11 TCheckListBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

22.1.1.6.12 TCheckListBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
```

```
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

22.1.1.6.13 TCheckListBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
```

```
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

22.1.1.6.14 TCheckListBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
```

```
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

22.1.1.6.15 TCheckListBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
```

```
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
```

```
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

22.1.1.6.16 TCheckListBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

22.1.1.6.17 TCheckListBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
```

```
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
```

```
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

22.1.1.6.18 TCheckListBox.OnMeasureItem Event

Occurs when the application needs to redisplay an item in a variable height owner-draw list box.

Pascal

```
property OnMeasureItem: TMeasureItemEvent;
```

C++

```
__property TMeasureItemEvent OnMeasureItem;
```

Description

Use OnMeasureItem to write a handler to measure items to be drawn in a list box with a Style property value of lbOwnerDrawVariable.

OnMeasureItem is of type TMeasureItemEvent which contains three parameters describing the item to measure:

Control, specifying the control containing the item.

Index, specifying the index of the item in the control.

Height, specifying the height of the item.

The Index parameter identifies the position of the item in the list box.

The Height parameter should specify the height in pixels that the given item will occupy in the control. The Height parameter is passed by reference (a var parameter), which initially contains the default height of the item or the height of the item text in the control's font. The handler can set Height to a value appropriate to the contents of the item, such as the height of a graphical image to be displayed within the item.

After the OnMeasureItem event occurs, the OnDrawItem event occurs, rendering the item with the measured size.

C++ Examples:

```
/*
Here is a typical handler for an OnMeasureItem event. The
example assumes that a variable owner-draw list box already
has bitmaps associated with each of its strings. It sets the
height of the item to the height of the associated bitmap if
that height is greater than the default height
(ListBox1.ItemHeight).  Note that TWinControl can also be
used in place of TWidgetControl on Windows.
*/
void __fastcall TForm1::ListBox1MeasureItem(TWinControl *Control, int Index,
      int &Height)
{
  Graphics::TBitmap *bitmap = (Graphics::TBitmap *)((TListBox
*)Control)->Items->Objects[Index];
```

```
    if (bitmap && bitmap->Height > Height)
      Height = bitmap->Height;
  }

  void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
        TRect &Rect, TOwnerDrawState State)
  {
    Graphics::TBitmap *pBitmap; // temporary variable for item's bitmap
    int     Offset = 2;   // default text offset width
    // note that we draw on the listbox's canvas, not on the form
    TCanvas *pCanvas = ((TListBox *)Control)->Canvas;
    pCanvas->FillRect(Rect); // clear the rectangle
    pBitmap = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
    if (pBitmap)
    {
      pCanvas->BrushCopy(Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
  pBitmap, Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
      Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
    }
    // display the text
    pCanvas->TextOut(Rect.Left + Offset, Rect.Top, ((TListBox *)Control)->Items->Strings[Index]);
  }

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    Graphics::TBitmap *bitmap0 = new Graphics::TBitmap;
    Graphics::TBitmap *bitmap1 = new Graphics::TBitmap;
    Graphics::TBitmap *bitmap2 = new Graphics::TBitmap;
    bitmap0->LoadFromFile("../littleB_64.bmp");
  //  ImageList1->GetBitmap(0, bitmap0);
    ListBox1->Items->AddObject("Butterfly", bitmap0);
    ImageList1->GetBitmap(3, bitmap1);
    ListBox1->Items->AddObject("Animal", bitmap1); // no bitmap, see if it leaves this height at
  64.
    ImageList1->GetBitmap(4, bitmap2);
    ListBox1->Items->AddObject("Sunflower", bitmap2); // no bitmap, see if it leaves this height
  at 64.

  }
```

See Also

DrawItem (see page 21328)

ItemHeight (see page 21347)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

Style (see page 21354)

22.1.1.6.19 TCheckListBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse

button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

22.1.1.6.20 TCheckListBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area

of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

22.1.1.6.21 TCheckListBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

22.1.1.6.22 TCheckListBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

22.1.1.6.23 TCheckListBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
```

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

22.1.1.6.24 TCheckListBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

22.1.1.6.25 TCheckListBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

22.1.1.6.26 TCheckListBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

23 Classes Namespace

This is namespace Classes.

Topics

Name	Description
Classes (see page 2692)	The following table lists classes in this documentation.
Interfaces (see page 3195)	The following table lists interfaces in this documentation.
Functions (see page 3214)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 3268)	The following table lists structs, records, enums in this documentation.
Types (see page 3282)	The following table lists types in this documentation.
Variables (see page 3309)	The following table lists variables in this documentation.
Constants (see page 3316)	The following table lists constants in this documentation.

Classes

	Name	Description
	EBitsError (see page 2693)	EBitsError is the exception class for invalid attempts to access an array of boolean values.
	EClassNotFound (see page 2694)	EClassNotFound is the exception class for the failure to find a specified component when reading from a stream.
	EComponentError (see page 2694)	EComponentError is the exception class for registering and renaming components.
	EFCreateError (see page 2695)	EFCreateError is the exception class for streaming file-creation errors.
	EFilerError (see page 2696)	EFilerError is the exception class for component stream errors.
	EFileStreamError (see page 2697)	EFileStreamError is the exception class for general stream errors.
	EFOpenError (see page 2699)	EFOpenError is the exception class for file-opening errors during streaming.
	EInvalidImage (see page 2699)	EInvalidImage is the exception class for unsuccessful attempts to read resources.
	EInvalidOperation (see page 2700)	EInvalidOperation is the exception class for invalid operations on a component.
	EListError (see page 2701)	EListError is the exception class for list and string errors.
	EMethodNotFound (see page 2701)	Deprecated, please use EReadError (see page 2703).
	EOutOfResources (see page 2702)	EOutOfResources is the exception class for unsuccessful attempts to allocate handles.
	EParserError (see page 2702)	EParserError is the exception class for text-to-binary streaming conversion errors.
	EReadError (see page 2703)	EReadError is the exception class for unsuccessful attempts to read data from a stream.
	EResNotFound (see page 2704)	EResNotFound is the exception class for unlocatable resources.
	EStreamError (see page 2704)	EStreamError is the exception class for streaming errors.
	EStringListError (see page 2705)	EStringListError is the exception class for invalid listbox references.

	EThread (see page 2706)	EThread is the exception class for thread synchronization errors.
	EWriteError (see page 2706)	EWriteError is the exception class for unsuccessful attempts to write data to a stream.
	TBasicAction (see page 2707)	TBasicAction is the ancestor class for all action objects.
	TBasicActionLink (see page 2720)	TBasicActionLink is the base class for action link classes that handle the communication between actions and clients.
	TBits (see page 2725)	TBits stores an array of Boolean values.
	TClassFinder (see page 2728)	This is class Classes.TClassFinder.
	TCollection (see page 2731)	TCollection is a container for TCollectionItem (see page 2755) objects.
	TCollectionEnumerator (see page 2753)	Enumerates through the TCollectionItems in a TCollection (see page 2731) object.
	TCollectionItem (see page 2755)	TCollectionItem represents an item in a collection.
	TComponent (see page 2762)	TComponent is the common ancestor of all component classes.
	TComponentEnumerator (see page 2812)	TComponentEnumerator allows the components within a containing component to be discovered.
	TCustomMemoryStream (see page 2815)	TCustomMemoryStream is an abstract base class used as the common ancestor for memory streams.
	TDataModule (see page 2824)	TDataModule centralizes the handling of nonvisual components in an application.
	TFiler (see page 2834)	TFiler is the abstract base class for reader and writer objects, which are used for loading and saving components and their properties.
	TFileStream (see page 2841)	TFileStream enables applications to read from and write to a file on disk.
	THandleStream (see page 2845)	THandleStream enables applications to read from and write to communications resources identified by a handle.
	TInterfacedPersistent (see page 2851)	TInterfacedPersistent is an ancestor for persistent objects that implement interfaces.
	TInterfaceList (see page 2855)	TInterfaceList represents a list of interfaces.
	TInterfaceListEnumerator (see page 2866)	Enumerates through the IInterfaces in a TInterfaceList (see page 2855).
	TList (see page 2868)	TList stores an array of pointers.
	TListEnumerator (see page 2900)	Enumerates through the objects in a TList (see page 2868).
	TMemoryStream (see page 2902)	TMemoryStream is a stream that stores its data in dynamic memory.
	TOwnedCollection (see page 2912)	TOwnedCollection is a collection that maintains information about its owner.
	TParser (see page 2915)	TParser allows DFM files to be parsed.
	TPersistent (see page 2928)	TPersistent is the ancestor for all objects that have assignment and streaming capabilities.
	TReader (see page 2933)	TReader is a specialized filer that reads component data from an associated stream.
	TRecall (see page 2964)	TRecall stores the properties of a persistent object so that they can be restored at a later time.
	TResourceStream (see page 2968)	TResourceStream is a memory stream that provides access to the compiled resources in an application.
	TStream (see page 2973)	TStream is the base class type for stream objects that can read from or write to various kinds of storage media, such as disk files, dynamic memory, and so on.
	TStreamAdapter (see page 2997)	TStreamAdapter implements the IStream interface on a TStream (see page 2973) object.
	TStringList (see page 3006)	TStringList maintains a list of strings.
	TStrings (see page 3029)	TStrings is the base class for objects that represent a list of strings.
	TStringsEnumerator (see page 3086)	Enumerates through the strings of a TStrings (see page 3029) object.
	TStringStream (see page 3088)	TStringStream provides file-like access to information stored as a long string.
	TThread (see page 3108)	TThread is an abstract class that enables creation of separate threads of execution in an application.
	TThreadList (see page 3149)	TThreadList represents a thread-safe list.
	TWriter (see page 3175)	TWriter is a specialized filer object that writes data to its associated stream.

Constants

Name	Description
fmCreate (see page 3316)	Specifies a file to be created with a specific name.
MaxListSize (see page 3317)	Defines the maximum capacity of certain list classes and types.
scAlt (see page 3317)	Specifies that the Alt key was pressed in a short cut menu.
scCtrl (see page 3318)	Specifies that the Ctrl key was pressed in a short cut menu.

scNone (see page 3318)	Specifies that none of the Shift, Ctrl or Alt keys were pressed in a short cut menu.
scShift (see page 3319)	Specifies that the Shift was pressed in a short cut menu.
soFromBeginning (see page 3319)	Specifies that a stream seek operation should move relative to the stream beginning.
soFromCurrent (see page 3320)	Specifies that a stream seek operation should move relative to the current position.
soFromEnd (see page 3320)	Specifies that a stream seek operation should move relative to the stream end.
toEOF (see page 3321)	Specifies a TParser (see page 2915) End of file token.
toFloat (see page 3322)	Specifies a TParser (see page 2915) floating point number token.
toInteger (see page 3322)	Specifies a TParser (see page 2915) integer token.
toString (see page 3323)	Specifies a TParser (see page 2915) string token.
toSymbol (see page 3324)	Specifies a TParser (see page 2915) symbol token.
toWString (see page 3324)	Specifies a TParser (see page 2915) wide string token.

Functions

	Name	Description
	ActivateClassGroup (see page 3215)	Activates a group of classes that derive from TPersistent (see page 2928).
	ActiveClassGroup (see page 3216)	Activates a group of classes that derive from TPersistent (see page 2928).
	AllocateHWnd (see page 3217)	DeallocateHWnd (see page 3221) is used to destroy a hidden window created using AllocateHWnd.
	AncestorIsValid (see page 3217)	AncestorIsValid determines if one component is an ancestor of another.
	BeginGlobalLoading (see page 3218)	This is function Classes.BeginGlobalLoading.
	BinToHex (see page 3218)	Converts a binary value into its hexadecimal representation.
	Bounds (see page 3218)	Creates a TRect structure from the top-left coordinate, and width and height values of a rectangle.
	CheckSynchronize (see page 3219)	Allows background threads to synchronize their execution with the main thread.
	ClassGroupOf (see page 3220)	Returns the group of persistent classes a specified class or instance belongs to.
	ClassGroupOf (see page 3220)	Returns the group of persistent classes a specified class or instance belongs to.
	CollectionsEqual (see page 3220)	Compares the contents of two collections.
	CountGenerations (see page 3221)	Returns the number of intermediate classes between a derived class and its ancestor.
	DeallocateHWnd (see page 3221)	Frees a window that was created using AllocateHWnd (see page 3217).
	EndGlobalLoading (see page 3222)	This is function Classes.EndGlobalLoading.
	ExtractStrings (see page 3222)	Fills a string list with substrings parsed from a delimited list.
	FindClass (see page 3223)	Finds and returns a class that is derived from TPersistent (see page 2928).
	FindGlobalComponent (see page 3223)	Returns a top-level container component.
	FindIdentToInt (see page 3224)	Returns the TIdentToInt (see page 3295) conversion routine for a given data type.
	FindIntToIdent (see page 3224)	Returns the TIntToIdent (see page 3296) conversion routine for a given data type.
	FindNestedComponent (see page 3225)	FindNestedComponent returns a component contained within another.
	FindRootDesigner (see page 3225)	This is function Classes.FindRootDesigner.
	FreeObjectInstance (see page 3226)	This is function Classes.FreeObjectInstance.
	GetClass (see page 3226)	Returns a registered persistent class given its name.
	GetFixupInstanceNames (see page 3228)	This is function Classes.GetFixupInstanceNames.
	GetFixupReferenceNames (see page 3228)	This is function Classes.GetFixupReferenceNames.
	GetUltimateOwner (see page 3228)	Returns the top-level owner of a specified persistent object.
	GetUltimateOwner (see page 3228)	Returns the top-level owner of a specified persistent object.
	GetUltimateOwner (see page 3228)	Returns the top-level owner of a specified persistent object.
	GlobalFixupReferences (see page 3229)	This is function Classes.GlobalFixupReferences.
	GroupDescendentsWith (see page 3229)	Adds a specified class to the group in which another class appears.
	HexToBin (see page 3230)	Converts a string of hexadecimal digits to the corresponding binary value.
	IdentToInt (see page 3230)	Uses a mapping array to convert string identifiers into their corresponding integer values.
	InitComponentRes (see page 3231)	This is function Classes.InitComponentRes.
	InitInheritedComponent (see page 3231)	Initializes streaming of a form file for an inherited root class.

	IntToIdent (see page 3232)	Uses a mapping array to convert integers into their corresponding string identifiers.
	InvalidPoint (see page 3233)	Indicates whether a specified point is equal to (-1,-1).
	InvalidPoint (see page 3233)	Indicates whether a specified point is equal to (-1,-1).
	InvalidPoint (see page 3233)	Indicates whether a specified point is equal to (-1,-1).
	IsDefaultPropertyValue (see page 3233)	This is function Classes.IsDefaultPropertyValue.
	IsUniqueGlobalComponentName (see page 3234)	Indicates whether a proposed component name is unambiguous.
	LineStart (see page 3234)	Finds the end of the last whole line in a buffer.
	MakeObjectInstance (see page 3235)	This is function Classes.MakeObjectInstance.
	NotifyGlobalLoading (see page 3235)	This is function Classes.NotifyGlobalLoading.
	ObjectBinaryToText (see page 3235)	Converts the binary representation of an object into more readily understandable text.
	ObjectBinaryToText (see page 3235)	Converts the binary representation of an object into more readily understandable text.
	ObjectResourceToText (see page 3239)	Converts the binary representation of an object resource into more readily understandable text.
	ObjectResourceToText (see page 3239)	Converts the binary representation of an object resource into more readily understandable text.
	ObjectTextToBinary (see page 3240)	Converts a symbolic text representation of an object into the binary version that is used to save the object to files or memory streams.
	ObjectTextToBinary (see page 3240)	Converts a symbolic text representation of an object into the binary version that is used to save the object to files or memory streams.
	ObjectTextToResource (see page 3244)	Converts a symbolic text representation of an object into an internal binary representation.
	ObjectTextToResource (see page 3244)	Converts a symbolic text representation of an object into an internal binary representation.
	Point (see page 3244)	Creates a TPoint structure from a pair of coordinates.
	PointsEqual (see page 3245)	Indicates whether two points have the same coordinates.
	PointsEqual (see page 3245)	Indicates whether two points have the same coordinates.
	ReadComponentRes (see page 3245)	Reads components and their properties from a specified resource.
	ReadComponentResEx (see page 3246)	Reads a component from a resource.
	ReadComponentResFile (see page 3247)	Reads components and their properties from a specified Windows resource file.
	Rect (see page 3249)	Creates a TRect structure from a set of coordinates.
	Rect (see page 3249)	Creates a TRect structure from a set of coordinates.
	RedirectFixupReferences (see page 3250)	This is function Classes.RedirectFixupReferences.
	RegisterClass (see page 3250)	Registers a class of persistent object so that it's class type can be retrieved.
	RegisterClassAlias (see page 3254)	Registers a class that is identical to another class except for the name.
	RegisterClasses (see page 3255)	Registers a set of classes.
	RegisterComponents (see page 3256)	Registers a set of components so that they all appear on the same page of the component palette.
	RegisterFindGlobalComponentProc (see page 3259)	This is function Classes.RegisterFindGlobalComponentProc.
	RegisterIntegerConsts (see page 3259)	Registers conversion functions for string identifiers that represent type values.
	RegisterNoIcon (see page 3260)	Registers a set of components but does not add them to the component palette.
	RegisterNonActiveX (see page 3260)	Prevents a set of components from being made available to the ActiveX wizard.
	RemoveFixupReferences (see page 3261)	This is function Classes.RemoveFixupReferences.
	RemoveFixups (see page 3261)	This is function Classes.RemoveFixups.
	SmallPoint (see page 3262)	Creates a TSmallPoint structure from a pair of coordinates.
	StartClassGroup (see page 3262)	Starts a class group derived from TPersistent (see page 2928).
	TestStreamFormat (see page 3263)	Identifies the format of a form file when reading it from a stream.
	UnRegisterClass (see page 3263)	Unregisters an object class.
	UnRegisterClasses (see page 3264)	Unregisters a set of classes.
	UnregisterFindGlobalComponentProc (see page 3264)	This is function Classes.UnregisterFindGlobalComponentProc.
	UnregisterIntegerConsts (see page 3264)	Unregisters conversion functions for string identifiers that represent type values.
	UnRegisterModuleClasses (see page 3265)	Unregisters all classes defined in a specified module.
	WriteComponentResFile (see page 3266)	Writes components and their properties to a file using a resource file format.

Interfaces

	Name	Description
	IDesignerNotify (see page 3195)	IDesignerNotify is the interface for responding to notifications about changes to components in the designer.
	IInterfaceComponentReference (see page 3197)	IInterfaceComponentReference is the interface for responding to requests for retrieving components.
	IInterfaceList (see page 3198)	IInterfaceList provides access to a list of interfaces.
	IStreamPersist (see page 3207)	IStreamPersist is the interface for responding to requests to load/save an object from/to a persistent stream.
	IStringsAdapter (see page 3208)	This is class Classes.IStringsAdapter.
	IVarStreamable (see page 3210)	IVarStreamable is the interface for loading and saving the values of Variants (see page 23567).
	IVCLComObject (see page 3211)	This is class Classes.IVCLComObject.

Structs, Records, Enums

	Name	Description
	TActiveXRegType (see page 3268)	TActiveXRegType is used when excluding a component class from the ActiveX Wizard.
	TAlignment (see page 3269)	TAlignment specifies how text is aligned within a control.
	TBiDiMode (see page 3269)	TBiDiMode specifies the reading order (bidirectional mode) of a control.
	TCollectionNotification (see page 3270)	TCollectionNotification indicates the type of change that is made to the items in a collection.
	TDuplicates (see page 3271)	TDuplicates indicates the response when an application attempts to add a duplicate entry to a list.
	TFilerFlag (see page 3271)	TFilerFlag indicates information about how a filer should read or write a component and TFilerFlags (see page 3289) is a set of TFilerFlag values.
	THelpType (see page 3272)	THelpType indicates whether components invoke the help system using a context ID or a keyword.
	TIdentMapEntry (see page 3273)	TIdentMapEntry is used to perform ident string to integer value mapping.
	TListAssignOp (see page 3273)	TListAssignOp indicates how two lists should be merged.
	TListNotification (see page 3274)	TListNotification defines list processing action types.
	TOperation (see page 3275)	TOperation represents the types of operations whose occurrence is broadcast by the Notification method.
	TSeekOrigin (see page 3276)	TSeekOrigin indicates where to start a seek operation.
	TStreamOriginalFormat (see page 3276)	TStreamOriginalFormat indicates the format in which a form file is saved.
	TStreamOwnership (see page 3277)	TStreamOwnership indicates whether a stream object should be freed by the object that uses it.
	TStringItem (see page 3277)	TStringItem defines a TStringList (see page 3006) entry.
	TSynchronizeRecord (see page 3278)	This is record Classes.TSynchronizeRecord.
	TThreadPriority (see page 3278)	TThreadPriority indicates the scheduling priority of a thread object on Windows.
	TValueType (see page 3279)	TValueType defines the kinds of values written to and read from filer objects.
	TVerticalAlignment (see page 3280)	TVerticalAlignment specifies how text is vertically aligned within a control.

Types

Name	Description
PPointerList (see page 3283)	PPointerList is a pointer to a TPointerList (see page 3299) type.
PStringItem (see page 3284)	PStringItem is a TStringItem (see page 3277) pointer.
PStringItemList (see page 3284)	PStringItemList is a TStringItemList (see page 3304) pointer.
PSynchronizeRecord (see page 3285)	This is type Classes.PSynchronizeRecord.
TAncestorNotFoundEvent (see page 3285)	TAncestorNotFoundEvent is a procedure called by the TReader (see page 2933) class OnAncestorNotFound event.
TBasicActionClass (see page 3285)	TBasicActionClass defines the metaclass for TBasicAction (see page 2707).
TBasicActionLinkClass (see page 3286)	TBasicActionLinkClass defines the metaclass for TBasicActionLink (see page 2720).
TCollectionItemClass (see page 3286)	TCollectionItemClass defines the metaclass for TCollectionItem (see page 2755).
TComponentClass (see page 3287)	TComponentClass defines the metaclass for TComponent (see page 2762).
TComponentName (see page 3287)	TComponentName is the type for the Name property of all components.

TComponentState (see page 3287)	TComponentState is a set of flags that describe the current state of a component.
TComponentStyle (see page 3288)	TComponentStyle is a set of flags that describe the current Style of a component.
TCreateComponentEvent (see page 3289)	TCreateComponentEvent is a type of procedure called by the TReader (see page 2933) class OnCreateComponentEvent event.
TFilerFlags (see page 3289)	TFilerFlags is the set of TFilerFlag (see page 3271) values.
TFindAncestorEvent (see page 3290)	TFindAncestorEvent is a procedure called by the TWriter (see page 3175) class OnFindAncestor event.
TFindComponentClassEvent (see page 3290)	This is type Classes.TFindComponentClassEvent.
TFindComponentInstanceEvent (see page 3291)	This is type Classes.TFindComponentInstanceEvent.
TFindGlobalComponent (see page 3291)	This is type Classes.TFindGlobalComponent.
TFindMethodEvent (see page 3291)	TFindMethodEvent is a procedure called by the TReader (see page 2933) class OnFindMethod event.
TFindMethodInstanceEvent (see page 3292)	TFindMethodInstanceEvent is a procedure called by the TReader (see page 2933) class OnFindMethodInstance event.
TFindMethodNameEvent (see page 3292)	TFindMethodNameEvent is a procedure called by the TWriter (see page 3175) class OnFindMethodName event.
TGetChildProc (see page 3292)	TGetChildProc is a type of procedure called used internally by the TComponent (see page 2762) class.
TGetClass (see page 3293)	TGetClass is a callback procedure used by the TClassFinder (see page 2728) class.
TGetLookupInfoEvent (see page 3293)	This is type Classes.TGetLookupInfoEvent.
TGetModuleProc (see page 3294)	This is type Classes.TGetModuleProc.
TGetStrProc (see page 3294)	TGetStrProc is a general purpose type of callback procedure invoked for each of a set of strings.
THelpContext (see page 3295)	THelpContext is the range of values that a HelpContext value may have.
TIdentToInt (see page 3295)	TIdentToInt is a type of callback function used by the IdentToInt (see page 3230) routine.
TIntToIdent (see page 3296)	TIntToIdent is a type of callback function used by the IntToIdent (see page 3232) routine.
TIsUniqueGlobalComponentName (see page 3296)	TIsUniqueGlobalComponentName is a type of function used by IsUniqueGlobalComponentName (see page 3234).
TLeftRight (see page 3297)	TLeftRight is a subset of TAlignment (see page 3269) that allows left or right justification.
TListSortCompare (see page 3297)	TListSortCompare is a type of callback function called used internally by the TList (see page 2868) class.
TNotifyEvent (see page 3298)	TNotifyEvent is used for events that do not require parameters.
TParserErrorEvent (see page 3298)	TParserErrorEvent is a callback procedure used by the TClassFinder (see page 2728) class.
TPersistentClass (see page 3299)	TPersistentClass defines the metaclass for TPersistent (see page 2928).
TPointerList (see page 3299)	TPointerList is an array of pointers.
TReadComponentsProc (see page 3300)	TReadComponentsProc is a type of callback procedure called used internally by the TReader (see page 2933) class ReadComponents method.
TReaderError (see page 3300)	TReaderError is a type of callback procedure called used internally by the TReader (see page 2933) class ReadComponents method.
TReaderProc (see page 3301)	TReaderProc is a type of callback procedure called used by TFiler (see page 2834) and descendant DefineProperty methods.
TReferenceNameEvent (see page 3301)	TReferenceNameEvent is a procedure called by the TReader (see page 2933) class OnReferenceNameEvent event.
TSetNameEvent (see page 3302)	TSetNameEvent is a procedure called by the TReader (see page 2933) class OnSetNameEvent event.
TShiftState (see page 3302)	TShiftState indicates the state of the Alt, Ctrl, and Shift keys and the mouse buttons.
TShortCut (see page 3303)	TStringsDefined (see page 3306) defines an enumeration of string delimiter names.
TStreamProc (see page 3304)	TStreamProc is a type of callback procedure called used by TFiler (see page 2834) and descendant DefineBinaryProperty methods.
TStringItemList (see page 3304)	TStringItemList defines an array of TStringItems.
TStringListSortCompare (see page 3305)	TStringListSortCompare is a type of callback function called used internally by the TStringList (see page 3006) class.
TStringsDefined (see page 3306)	This is type Classes.TStringsDefined.
TThreadMethod (see page 3306)	This is type Classes.TThreadMethod.

TTopBottom (see page 3306)	TTopBottom is a subset of TVerticalAlignment (see page 3280) that allows top or bottom alignment.
TWndMethod (see page 3307)	TWndMethod is the type for Windows message procedures.
TWriterProc (see page 3307)	TWriterProc is a type of callback procedure called used by TFiler (see page 2834) and descendant DefineProperty methods.

Variables

Name	Description
AddDataModule (see page 3309)	Defines a procedure to be called when a TDataModule (see page 2824) class is created.
ApplicationHandleException (see page 3310)	This is variable Classes.ApplicationHandleException.
ApplicationShowException (see page 3310)	This is variable Classes.ApplicationShowException.
CreateVCLComObjectProc (see page 3310)	This is variable Classes.CreateVCLComObjectProc.
CurrentGroup (see page 3311)	Keeps track of which objects are associated with which packages.
GlobalNameSpace (see page 3311)	This is variable Classes.GlobalNameSpace.
IsUniqueGlobalComponentNameProc (see page 3311)	IsUniqueGlobalComponentNameProc is called by IsUniqueGlobalComponentName (see page 3234).
RegisterComponentsProc (see page 3312)	RegisterComponentsProc is called by RegisterComponents (see page 3256).
RegisterNoIconProc (see page 3313)	RegisterNoIconProc is called by RegisterNoIcon (see page 3260).
RegisterNonActiveXProc (see page 3313)	RegisterNonActiveXProc is called by RegisterNonActiveX (see page 3260).
RemoveDataModule (see page 3314)	Defines a procedure to be called when a TDataModule (see page 2824) class is destroyed.
WakeMainThread (see page 3314)	Represents a method (event handler) that is forced into the main thread's queue.

23.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EBitsError (see page 2693)	EBitsError is the exception class for invalid attempts to access an array of boolean values.
	EClassNotFound (see page 2694)	EClassNotFound is the exception class for the failure to find a specified component when reading from a stream.
	EComponentError (see page 2694)	EComponentError is the exception class for registering and renaming components.
	EFCreateError (see page 2695)	EFCreateError is the exception class for streaming file-creation errors.
	EFilerError (see page 2696)	EFilerError is the exception class for component stream errors.
	EFileStreamError (see page 2697)	EFileStreamError is the exception class for general stream errors.
	EFOpenError (see page 2699)	EFOpenError is the exception class for file-opening errors during streaming.
	EInvalidImage (see page 2699)	EInvalidImage is the exception class for unsuccessful attempts to read resources.
	EInvalidOperation (see page 2700)	EInvalidOperation is the exception class for invalid operations on a component.
	EListError (see page 2701)	EListError is the exception class for list and string errors.
	EMethodNotFound (see page 2701)	Deprecated, please use EReadError (see page 2703).
	EOutOfResources (see page 2702)	EOutOfResources is the exception class for unsuccessful attempts to allocate handles.
	EParserError (see page 2702)	EParserError is the exception class for text-to-binary streaming conversion errors.
	EReadError (see page 2703)	EReadError is the exception class for unsuccessful attempts to read data from a stream.
	EResNotFound (see page 2704)	EResNotFound is the exception class for unlocatable resources.
	EStreamError (see page 2704)	EStreamError is the exception class for streaming errors.
	EStringListError (see page 2705)	EStringListError is the exception class for invalid listbox references.
	EThread (see page 2706)	EThread is the exception class for thread synchronization errors.
	EWriteError (see page 2706)	EWriteError is the exception class for unsuccessful attempts to write data to a stream.
	TBasicAction (see page 2707)	TBasicAction is the ancestor class for all action objects.
	TBasicActionLink (see page 2720)	TBasicActionLink is the base class for action link classes that handle the communication between actions and clients.
	TBits (see page 2725)	TBits stores an array of Boolean values.
	TClassFinder (see page 2728)	This is class Classes.TClassFinder.
	TCollection (see page 2731)	TCollection is a container for TCollectionItem (see page 2755) objects.
	TCollectionEnumerator (see page 2753)	Enumerates through the TCollectionItems in a TCollection (see page 2731) object.
	TCollectionItem (see page 2755)	TCollectionItem represents an item in a collection.
	TComponent (see page 2762)	TComponent is the common ancestor of all component classes.
	TComponentEnumerator (see page 2812)	TComponentEnumerator allows the components within a containing component to be discovered.
	TCustomMemoryStream (see page 2815)	TCustomMemoryStream is an abstract base class used as the common ancestor for memory streams.
	TDataModule (see page 2824)	TDataModule centralizes the handling of nonvisual components in an application.
	TFiler (see page 2834)	TFiler is the abstract base class for reader and writer objects, which are used for loading and saving components and their properties.
	TFileStream (see page 2841)	TFileStream enables applications to read from and write to a file on disk.
	THandleStream (see page 2845)	THandleStream enables applications to read from and write to communications resources identified by a handle.
	TInterfacedPersistent (see page 2851)	TInterfacedPersistent is an ancestor for persistent objects that implement interfaces.
	TInterfaceList (see page 2855)	TInterfaceList represents a list of interfaces.

	TInterfaceListEnumerator (see page 2866)	Enumerates through the IInterfaces in a TInterfaceList (see page 2855).
	TList (see page 2868)	TList stores an array of pointers.
	TListEnumerator (see page 2900)	Enumerates through the objects in a TList (see page 2868).
	TMemoryStream (see page 2902)	TMemoryStream is a stream that stores its data in dynamic memory.
	TOwnedCollection (see page 2912)	TOwnedCollection is a collection that maintains information about its owner.
	TParser (see page 2915)	TParser allows DFM files to be parsed.
	TPersistent (see page 2928)	TPersistent is the ancestor for all objects that have assignment and streaming capabilities.
	TReader (see page 2933)	TReader is a specialized filer that reads component data from an associated stream.
	TRecall (see page 2964)	TRecall stores the properties of a persistent object so that they can be restored at a later time.
	TResourceStream (see page 2968)	TResourceStream is a memory stream that provides access to the compiled resources in an application.
	TStream (see page 2973)	TStream is the base class type for stream objects that can read from or write to various kinds of storage media, such as disk files, dynamic memory, and so on.
	TStreamAdapter (see page 2997)	TStreamAdapter implements the IStream interface on a TStream (see page 2973) object.
	TStringList (see page 3006)	TStringList maintains a list of strings.
	TStrings (see page 3029)	TStrings is the base class for objects that represent a list of strings.
	TStringsEnumerator (see page 3086)	Enumerates through the strings of a TStrings (see page 3029) object.
	TStringStream (see page 3088)	TStringStream provides file-like access to information stored as a long string.
	TThread (see page 3108)	TThread is an abstract class that enables creation of separate threads of execution in an application.
	TThreadList (see page 3149)	TThreadList represents a thread-safe list.
	TWriter (see page 3175)	TWriter is a specialized filer object that writes data to its associated stream.

23.1.1 EBitsError Class

EBitsError is the exception class for invalid attempts to access an array of boolean values.

Class Hierarchy

Pascal

```
EBitsError = class(Exception);
```

C++

```
class EBitsError : public Exception;
```

File

Classes

Description

EBitsError is raised when an application attempts to access a boolean array (an instance of the TBits (see page 2725) class) using an index value that is too large or too small. This can occur when the value of the Index parameter of the Bits property is

Less than zero.

Equal to or greater than the Size property.

See Also

TBits (see page 2725)

TBits (see page 2725)

TBits (see page 2725)

ERangeError

23.1.2 EClassNotFound Class

EClassNotFound is the exception class for the failure to find a specified component when reading from a stream.

Class Hierarchy

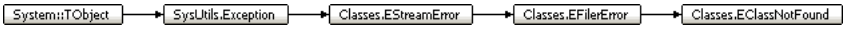

Pascal

```
EClassNotFound = class(EFilerError);
```

C++

```
class EClassNotFound : public EFilerError;
```

File

Classes

Description

EClassNotFound is raised if a class name that has not been linked into the current application is encountered when reading a component from a stream. This can happen when a component exists on a form, but has been removed from the type declaration.

See Also

FindClass (see page 3223)

TReader (see page 2933)

23.1.3 EComponentError Class

EComponentError is the exception class for registering and renaming components.

Class Hierarchy

Pascal

```
EComponentError = class(Exception);
```

C++

```
class EComponentError : public Exception;
```

File

Classes

Description

EComponentError is raised when:

An attempt to register a component fails.

An application cannot rename a component.

(Windows-only) A request is made to retrieve the COM interface of a component that doesn't support COM.

To register a component, write a procedure called "Register", declared in the interface section (Delphi) or namespace (C++) of the component's unit file. Note that unlike most Delphi procedure names, the name of the Register procedure is case-sensitive. Register must call RegisterComponents (see page 3256) for each component you want to register.

Failure to rename a component occurs when an application tries to rename the component at runtime, and the new name is the same as the name of another component with the same Owner or is not a valid identifier.

See Also

RegisterComponents (see page 3256)

RegisterNoIcon (see page 3260)

TComponent (see page 2762)

TComponent (see page 2762)

TComponent (see page 2762)

TCoolBar (see page 3603)

TDataSet (see page 6894)

TComponent (see page 2762)

23.1.4 EFCreateError Class

EFCreateError is the exception class for streaming file-creation errors.

Class Hierarchy

Pascal

```
EFCreateError = class(EFileStreamError);
```

C++

```
class EFCreateError : public EFileStreamError;
```

File

Classes

Description

EFCreateError is raised when an application unsuccessfully attempts to create a file. This can occur, for example, if a user specifies an invalid file name, or specifies the name of an existing file that cannot be overwritten because the user lacks appropriate access permission.

See Also

EFOpenError (see page 2699)

EFilerError (see page 2696)

TFileStream (see page 2841)

TStrings (see page 3029)

TCustomMemoryStream (see page 2815)

TMemoryStream (see page 2902)

ReadComponentResFile (see page 3247)

WriteComponentResFile (see page 3266)

23.1.5 EFilerError Class

EFilerError is the exception class for component stream errors.

Class Hierarchy

Pascal

```
EFilerError = class(EStreamError);
```

C++

```
class EFilerError : public EStreamError;
```

File

Classes

Description

EFilerError is raised when an error occurs while streaming a component. Some, but not all, component streaming errors raise descendants of EFilerError, as follows.

Exception	Description
EReadError (see page 2703)	Cannot read the specified number of bytes
EWriteError (see page 2706)	Cannot write the specified number of bytes
EClassNotFound (see page 2694)	Component is not linked to application
EInvalidImage (see page 2699)	Cannot read an resource file

EFilerError differs from EFileStreamError (see page 2697), which is raised by errors in general file streams.

See Also

RegisterClass (see page 3250)

EReadError (see page 2703)

EWriteError (see page 2706)

EClassNotFound (see page 2694)

EInvalidImage (see page 2699)

TStream (see page 2973)

TStream (see page 2973)

TStream (see page 2973)

23.1.6 EFileStreamError Class

EFileStreamError is the exception class for general stream errors.

Class Hierarchy

Pascal

```
EFileStreamError = class(EStreamError);
```

C++

```
class EFileStreamError : public EStreamError;
```

File

Classes

Description

EFileStreamError is raised when an error occurs while streaming data. Some, but not all, data streaming errors raise descendants of EFileStreamError, as follows.

Exception	Description
EFCreateError (see page 2695)	Error while creating a new file for streaming.
EFOpenError (see page 2699)	Error while opening an existing file for streaming.

EFileStreamError differs from EFilerError (see page 2696), which is raised by errors in component streams.

See Also

TFileStream (see page 2841)

23.1.6.1 EFileStreamError Members

The following tables list the members exposed by EFileStreamError.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2698)	Creates a new EFileStreamError (see page 2697) object.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.

	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.

23.1.6.2 EFileStreamError.Create Constructor

Creates a new EFileStreamError (see page 2697) object.

Pascal

```
constructor Create(ResStringRec: PResStringRec; const FileName: string);
```

C++

```
__fastcall EFileStreamError(PResStringRec ResStringRec, const AnsiString FileName);
```

Description

Create instantiates an exception object with a simple message string. Use the throw keyword with the constructor. For example,

```
throw EFileStreamError ("My error message");
```

Msg is the runtime error message that appears in the exception dialog box.

Args is an array of constants containing values to format according to format specifiers in Msg.

Args_Size is the index of the last constant in Args (one less than the number of entries).

AHelpContext is an integer that specifies the context-sensitive Help ID for the exception.

Ident is the unique ID of the string resource for the error message as specified in the RES file. If Ident is not a valid resource ID, the constructor creates an empty message string for the exception.

ResStringRec is a struct that describes a resourced string by giving its module and resource ID. It is used primarily with Delphi resourcestring values.

23.1.7 EFOpenError Class

EFOpenError is the exception class for file-opening errors during streaming.

Class Hierarchy

Pascal

```
EFOpenError = class(EFileStreamError);
```

C++

```
class EFOpenError : public EFileStreamError;
```

File

Classes

Description

EFOpenError is raised when an application cannot open a specified file. This can occur, for example, because the file doesn't exist or is not in the directory where the application is searching for it.

See Also

EFOpenError

EFilerError (see page 2696)

TFileStream (see page 2841)

TStrings (see page 3029)

TCustomMemoryStream (see page 2815)

TMemoryStream (see page 2902)

ReadComponentResFile

WriteComponentResFile

23.1.8 EInvalidImage Class

EInvalidImage is the exception class for unsuccessful attempts to read resources.

Class Hierarchy

Pascal

```
EInvalidImage = class(EFilerError);
```

C++

```
class EInvalidImage : public EFilerError;
```

File

Classes

Description

EInvalidImage is raised when an application cannot read a resource. This occurs most often because the specified file is not a resource file.

This error can occur if the TStream (see page 2973) object's read and write methods are used inconsistently. Always use ReadComponentRes (see page 3245) to read components written with WriteComponentRes, and ReadComponent to read components written with WriteComponent. Failure to use the corresponding read and write methods can result in EInvalidImage exceptions.

See Also

TStream (see page 2973)

ObjectResourceToText (see page 3239)

23.1.9 EInvalidOperation Class

EInvalidOperation is the exception class for invalid operations on a component.

Class Hierarchy

Pascal

```
EInvalidOperation = class(Exception);
```

C++

```
class EInvalidOperation : public Exception;
```

File

Classes

Description

EInvalidOperation is raised when an application attempts an operation that requires a window or widget handle on a component that does not have a parent (The Parent property is Nil in Delphi or NULL in C++). This exception can also occur if a drag-and-drop operation is attempted from a form (for example, when calling the BeginDrag method of a form).

Note: Do not confuse this object with EInvalidOp.

See Also

TWidgetControl

TWinControl (see page 6112)

BeginDrag

ValidParentForm

EInvalidOp (see page 22738)

23.1.10 EListError Class

EListError is the exception class for list and string errors.

Class Hierarchy

Pascal

```
EListError = class(Exception);
```

C++

```
class EListError : public Exception;
```

File

Classes

Description

EListError is raised when an error is made in a list, TStrings (see page 3029), or TStringList (see page 3006) object. This exception commonly occurs when an application refers to an item that is out of the list's range.

EListError also occurs if an application tries to add a duplicate string to a TStringList (see page 3006) object when the value of the Duplicates property is dupError.

An EListError exception is raised when an application attempts to insert a string into a sorted string list, since the insertion of a string at a specified position may put the list out of order.

See Also

TCollection (see page 2731)

TComponent (see page 2762)

TCustomListBox (see page 21308)

TStringList (see page 3006)

TStringList (see page 3006)

EStringListError (see page 2705)

23.1.11 EMethodNotFound Class

Deprecated, please use EReadError (see page 2703).

Class Hierarchy

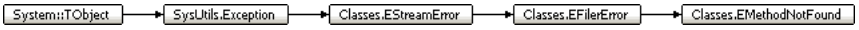

Pascal

```
EMethodNotFound = class(EFilerError);
```

C++

```
class EMethodNotFound : public EFilerError;
```

File

Classes

Description

The method EMethodNotFound has been deprecated, please use EReadError (see page 2703) in its place..

See Also

about EReadError

23.1.12 EOutOfResources Class

EOutOfResources is the exception class for unsuccessful attempts to allocate handles.

Class Hierarchy

Pascal

```
EOutOfResources = class(EOutOfMemory);
```

C++

```
class EOutOfResources : public EOutOfMemory;
```

File

Classes

Description

EOutOfResources is raised when an application attempts to create a Windows or widget handle and there are no more handles to allocate.

See Also

TWinControl (see page 6112)

TWinControl (see page 6112)

GetDeviceContext

TApplication (see page 13660)

CreateHandle

TTreeNodes (see page 5266)

InsertObject

EOutOfMemory (see page 22742)

23.1.13 EParserError Class

EParserError is the exception class for text-to-binary streaming conversion errors.

Class Hierarchy

System::TObject → SysUtils.Exception → Classes.EParserError

Pascal

```
EParserError = class(Exception);
```

C++

```
class EParserError : public Exception;
```

File

Classes

Description

EParserError is raised when an error occurs during text-to-binary conversion, usually due to a syntax error in the source text.

See Also

ObjectTextToBinary (see page 3240)

ObjectTextToResource (see page 3244)

23.1.14 EReadError Class

EReadError is the exception class for unsuccessful attempts to read data from a stream.

Class Hierarchy

System::TObject → SysUtils.Exception → Classes.EStreamError → Classes.EFilerError → Classes.EReadError

Pascal

```
EReadError = class(EFilerError);
```

C++

```
class EReadError : public EFilerError;
```

File

Classes

Description

EReadError is raised when an application attempts to read data from a stream but cannot read the specified number of bytes.

An EReadError exception may also be raised if a property cant be read while creating a form. This can occur because a component reads the form resource incorrectly, or because the resource is corrupt.

See Also

TReader (see page 2933)

TStream (see page 2973)

TStream (see page 2973)

TStream (see page 2973)

TStream (see page 2973)

TMemoryStream (see page 2902)

TMemoryStream (see page 2902)

23.1.15 EResNotFound Class

EResNotFound is the exception class for unlocatable resources.

Class Hierarchy

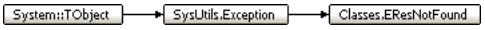

Pascal

```
EResNotFound = class(Exception);
```

C++

```
class EResNotFound : public Exception;
```

File

Classes

Description

EResNotFound is raised when a specified resource (such as a form or bitmap) can't be found in a form file or a resource file, or when the resource is not linked into the application.

If you receive this exception on startup of an application, or when you create a form, check the form's unit file for the compiler directive that links in the form file.

Note: In Delphi, this directive is {$R *.dfm} (for VCL forms)

In Delphi, this directive is {$R *.dfm} (for VCL forms)

Note: In C++, this directive #pragma resource #pragma resource "*.dfm" (for VCL forms)

If this directive is missing or commented out, the form file is not included in the executable and an EResNotFound exception is generated.

See Also

ReadComponentRes (see page 3245)

ReadComponentResEx (see page 3246)

TResourceStream (see page 2968)

23.1.16 EStreamError Class

EStreamError is the exception class for streaming errors.

Class Hierarchy

Pascal

```
EStreamError = class(Exception);
```

C++

```
class EStreamError : public Exception;
```

File

Classes

Description

EStreamError is the ancestor from which all streaming exception classes descend. EStreamError is itself raised when resources cannot be allocated for a memory stream.

See Also

TStream (see page 2973)

TMemoryStream (see page 2902)

TResourceStream (see page 2968)

EFCreateError (see page 2695)

EFOpenError (see page 2699)

EFilerError (see page 2696)

EClassNotFound (see page 2694)

EInvalidImage (see page 2699)

EReadError (see page 2703)

EWriteError (see page 2706)

EInOutError (see page 22732)

23.1.17 EStringListError Class

EStringListError is the exception class for invalid listbox references.

Class Hierarchy

Pascal

```
EStringListError = class(Exception);
```

C++

```
class EStringListError : public Exception;
```

File

Classes

Description

EStringListError is raised when an application attempts to access a list box (for example, to add an item) using an invalid index.

See Also

TCustomListBox (see page 21308)

EListError (see page 2701)

23.1.18 EThread Class

EThread is the exception class for thread synchronization errors.

Class Hierarchy

Pascal

```
EThread = class(Exception);
```

C++

```
class EThread : public Exception;
```

File

Classes

Description

EThread is never raised. It is defined for backward compatibility.

See Also

TThread (see page 3108)

TThread (see page 3108)

23.1.19 EWriteError Class

EWriteError is the exception class for unsuccessful attempts to write data to a stream.

Class Hierarchy

Pascal

```
EWriteError = class(EFilerError);
```

C++

```
class EWriteError : public EFilerError;
```

File

Classes

Description

EWriteError is raised when:

The WriteBuffer method of a stream object is unable to write the specified number of bytes.

The IDE is unable to write one of a component's properties to a stream. This can occur when you try to save a form at design time.

See Also

TStream (see page 2973)

TStream (see page 2973)

23.1.20 TBasicAction Class

TBasicAction is the ancestor class for all action objects.

Class Hierarchy

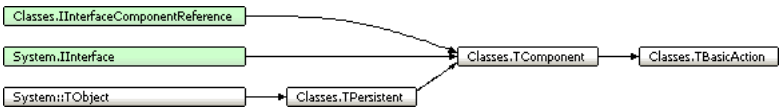

Pascal

```
TBasicAction = class(TComponent);
```

C++

```
class TBasicAction : public TComponent;
```

File

Classes

Description

TBasicAction introduces the fundamental behavior for an action. Descendants of TBasicAction add functionality for containment in an action list, for being categorized, and for specializing their behavior tailored to particular clients such as controls or menu items. Use TBasicAction if you want to create an action for an object that is neither a menu item nor a control.

23.1.20.1 TBasicAction Members

The following tables list the members exposed by TBasicAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.

	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.

	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)

V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

23.1.20.2 TBasicAction.Create Constructor

Instantiates and initializes a TBasicAction (see page 2707) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBasicAction(TComponent * AOwner);
```

Description

Applications do not need to instantiate TBasicAction (see page 2707) directly. Actions are created automatically when you choose New Action in the action list editor. If you want to create an action at runtime, call Create and assign a TActionList component to its ActionList property.

Create calls the inherited constructor and then creates its sub-objects, which includes a list of clients that are action links.

See Also

Create (see page 2766)

TActionList (see page 280)

ActionList (see page 295)

23.1.20.3 TBasicAction.Destroy Destructor

Disposes of an instance of a TBasicAction (see page 2707) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBasicAction();
```

Description

You do not need to call the destructor for an action. An action is a component and is automatically destroyed by its owner, which was passed to the constructor when it is created. If you must destroy an action manually, call Free instead, which safely invokes the destructor.

See Also

Free (see page 22300)

23.1.20.4 TBasicAction Fields

The fields of the TBasicAction class are listed here.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.

23.1.20.4.1 TBasicAction.FClients Field

Pascal

```
FClients: TList;
```

C++

```
TList FClients;
```

Description

This is FClients, a member of class TBasicAction.

23.1.20.5 TBasicAction Methods

The methods of the TBasicAction class are listed here.

Protected Methods

	Name	Description
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.

Public Methods

	Name	Description
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.

	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

23.1.20.5.1 TBasicAction.Change Method

Generates an OnChange (see page 2716) event.

Pascal

```
procedure Change; virtual;
```

C++

```
virtual __fastcall Change();
```

Description

Change is called automatically when the action's properties change. This method calls the OnChange (see page 2716) event handler, if one is assigned.

See Also

OnChange (see page 2716)

23.1.20.5.2 TBasicAction.Execute Method

Generates an OnExecute (see page 2716) event.

Pascal

```
function Execute: Boolean; dynamic;
```

C++

```
__fastcall Boolean Execute();
```

Description

Execute calls the OnExecute (see page 2716) event handler, if one is assigned, with the action as the Sender. Execute returns true if an event handler is called, false otherwise.

See Also

OnExecute (see page 2716)

23.1.20.5.3 TBasicAction.ExecuteTarget Method

Introduces an interface for invoking an action on a target client component or control.

Pascal

```
procedure ExecuteTarget(Target: TObject); virtual;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

ExecuteTarget does nothing in TBasicAction (see page 2707). ExecuteTarget was introduced in TBasicAction (see page 2707) so that descendants can override it to initiate the action on the target. For example, an edit action that performs copying might copy the contents of an edit control to the clipboard.

See Also

OnExecute (see page 2716)

HandlesTarget (see page 2713)

23.1.20.5.4 TBasicAction.HandlesTarget Method

Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

HandlesTarget does nothing in TBasicAction (see page 2707). HandlesTarget was introduced in TBasicAction (see page 2707) so that descendants can override it to check the type and state of a target. Handles target can verify any information about a target that is relevant for the action. HandlesTarget returns true if the target meets the specified criteria, false otherwise.

See Also

ExecuteTarget (see page 2712)

23.1.20.5.5 TBasicAction.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TBasicAction (see page 2707) overrides the Notification method to check whether the component that is the value of the ActionComponent (see page 2715) property is about to be freed. If so, it sets the ActionComponent (see page 2715) property to nil (Delphi) or NULL (C++).

See Also

ActionComponent (see page 2715)

23.1.20.5.6 TBasicAction.RegisterChanges Method

Associates the action with an action link.

Pascal

```
procedure RegisterChanges(Value: TBasicActionLink);
```

C++

```
__fastcall RegisterChanges(TBasicActionLink Value);
```

Description

RegisterChanges is called when the action and link are connected, for example, when a link sets a new action. Value is the action link with which the action associates itself. The link is added to the action's client list.

See Also

UnRegisterChanges (see page 2714)

23.1.20.5.7 TBasicAction.SetOnExecute Method

Assigns an OnExecute (see page 2716) event handler.

Pascal

```
procedure SetOnExecute(Value: TNotifyEvent); virtual;
```

C++

```
virtual __fastcall SetOnExecute(TNotifyEvent Value);
```

Description

SetOnExecute is the property write method for the OnExecute (see page 2716) event. In addition to assigning the value of OnExecute (see page 2716), SetOnExecute propagates the event handler to all client objects and generates an OnChange (see page 2716) event.

See Also

OnExecute (see page 2716)

OnChange (see page 2716)

23.1.20.5.8 TBasicAction.UnRegisterChanges Method

Breaks the association between the action and the action link.

Pascal

```
procedure UnRegisterChanges(Value: TBasicActionLink);
```

C++

```
__fastcall UnRegisterChanges(TBasicActionLink Value);
```

Description

UnRegisterChanges is called when the action and link should no longer be associated. UnRegisterChanges is called, for example, in the destructor or when a link sets a new action. Value is the action link with which the association is broken.

See Also

RegisterChanges (see page 2713)

23.1.20.5.9 TBasicAction.Update Method

Provides an opportunity to execute centralized code when an application is idle.

Pascal

```
function Update: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean Update();
```

Description

Update triggers the OnUpdate (see page 2718) event handler. Update returns true if a handler was found for that event, false otherwise. When the application is idle, the OnUpdate (see page 2718) event occurs for every action. This provides an opportunity for applications to execute centralized code for enabling and disabling, checking and unchecking, and so on.

See Also

OnUpdate (see page 2718)

23.1.20.5.10 TBasicAction.UpdateTarget Method

Introduces an interface for a method of notifying a client when the action updates itself.

Pascal

```
procedure UpdateTarget(Target: TObject); virtual;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

UpdateTarget does nothing in TBasicAction (see page 2707). UpdateTarget was introduced in TBasicAction (see page 2707) so that descendants can override it to correspondingly update a target when the action updates. For examples, see the TEditAction objects in the stdactns unit.

See Also

HandlesTarget (see page 2713)

OnUpdate (see page 2718)

23.1.20.6 TBasicAction Properties

The properties of the TBasicAction class are listed here.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.

23.1.20.6.1 TBasicAction.ActionComponent Property

Indicates the client component that caused this action to execute.

Pascal

```
property ActionComponent: TComponent;
```

C++

```
__property TComponent * ActionComponent;
```

Description

Use ActionComponent to discern which client component caused this action to execute. For example, examine ActionComponent from an OnExecute (see page 2716) event handler if you need to know what user action triggered this action.

When the user clicks a client control, that client sets ActionComponent before calling the action's Execute (see page 2712) method. After the action executes, the action resets ActionComponent to nil (Delphi) or NULL (C++).

See Also

Execute (see page 2712)

OnExecute (see page 2716)

23.1.20.7 TBasicAction Events

The events of the TBasicAction class are listed here.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

23.1.20.7.1 TBasicAction.OnChange Event

Occurs when one of the action's properties changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Applications can't use the protected OnChange event. It is used internally to manage the relationship between the properties of the action and the corresponding properties of the action's clients.

Component writers can use OnChange in descendant objects to respond when the action's properties change.

23.1.20.7.2 TBasicAction.OnExecute Event

Occurs when the client event that is linked to it fires.

Pascal

```
property OnExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExecute;
```

Description

Write an OnExecute event handler when you want to respond when the user triggers the client object's default event (typically an OnClick event). For most target clients, OnExecute is triggered on a Click method, and is associated with the OnClick event.

OnExecute also occurs when the user types the shortcut (or one of the secondary shortcuts) associated with the action or its client.

Warning: If you assign an OnExecute event handler to a predefined action, the default behavior of that action will not occur.

C++ Examples:

```
/*
This example requires a TApplicationEvents, a TActionList
and a TListBox in the form.  Select the TApplicationEvents,
double click on the OnActionExecute event and add the
following code to the handler.  Create an TAction named
MyAction in a TActionList.  The TActionList must have no
OnExecute event handler, and the TAction has an OnExecute
event handler called MyActionExecute.  Then set an object's
(such as a button) Action to MyAction.  The Handled
parameter of the event handler returns false by default. If
the handler handles the event, it should change Handled to
true, thereby preventing further attempts to handle the
action.
*/
void __fastcall TAppEventForm::ApplicationEventsActionExecute(
      TBasicAction *Action, bool &Handled)
{
    lbOther->Items->Add("Event OnActionExecute");
}

void __fastcall TAppEventForm::ActionExecute(TObject *Sender)
{
    ShowMessage("Action executed");
}
/*
When a client component or control is clicked, the OnExecute
event occurs for it’s associated action. For example, the
following code illustrates the OnExecute event handler for
an action that toggles the visibility of a toolbar when the
action is executed. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TActionList must have no OnExecute event handler, and the
TAction has an OnExecute event handler called Action1Execute.
Then set an object's (such as a button or menu item) Action
to Action1.  Set the TToolbar color to something that will
make it obvious when it is visible.  Also remember to uncheck
Project|Options|Applications|Enable runtime themes.
*/
void __fastcall TMainForm::Action1Execute(TObject *Sender)
{
  // Toggle Toolbar1's visibility
  ToolBar1->Visible = !ToolBar1->Visible;
  if (ToolBar1->Visible)
    lbOther->Items->Add("Event Action1Execute: toolbar visible");
  else
    lbOther->Items->Add("Event Action1Execute: toolbar not visible");
}
```

Delphi Examples:

```
{
```

```
This example requires a TApplicationEvents, a TActionList
and a TListBox in the form.  Select the TApplicationEvents,
double click on the OnActionExecute event and add the
following code to the handler.  Create an TAction named
MyAction in a TActionList.  The TActionList must have no
OnExecute event handler, and the TAction has an OnExecute
event handler called MyActionExecute.  Then set an object's
(such as a button) Action to MyAction.  The Handled
parameter of the event handler returns false by default. If
the handler handles the event, it should change Handled to
true, thereby preventing further attempts to handle the
action.
}
procedure TMainForm.ApplicationEventsActionExecute(Action: TBasicAction;
  var Handled: Boolean);
begin
  lbOther.Items.Add('Event OnActionExecute');
end;

procedure TMainForm.MyActionExecute(Sender: TObject);
begin
  ShowMessage('Action executed');
end;
{
When a client component or control is clicked, the OnExecute
event occurs for it’s associated action. For example, the
following code illustrates the OnExecute event handler for
an action that toggles the visibility of a toolbar when the
action is executed. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TActionList must have no OnExecute event handler, and the
TAction has an OnExecute event handler called Action1Execute.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Execute(Sender: TObject);
begin
    { Toggle Toolbar1’s visibility }
    ToolBar1.Visible := not ToolBar1.Visible;
  if ToolBar1.Visible then
    lbOther.Items.Add('Event Action1Execute: toolbar visible')
  else
    lbOther.Items.Add('Event Action1Execute: toolbar not visible')
end;
```

See Also

OnClick (see page 5851)

ShortCut (see page 314)

23.1.20.7.3 TBasicAction.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to

update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

23.1.21 TBasicActionLink Class

TBasicActionLink is the base class for action link classes that handle the communication between actions and clients.

Class Hierarchy

Pascal

```
TBasicActionLink = class(TObject);
```

C++

```
class TBasicActionLink : public TObject;
```

File

Classes

Description

TBasicActionLink is the base class from which all other action link classes ultimately descend. TBasicAction (see page 2707) link lays the foundation for the connection between action and the client. The client is a parameter to the TBasicActionLink constructor, and the action object is indicated by the Action (see page 2725) property. TBasicActionLink sets up the link between the action and client execution and updating events.

23.1.21.1 TBasicActionLink Members

The following tables list the members exposed by TBasicActionLink.

Protected Fields

	Name	Description
	FAction (see page 2722)	This is FAction, a member of class TBasicActionLink.

Protected Methods

	Name	Description
	AssignClient (see page 2723)	This is AssignClient, a member of class TBasicActionLink.
	Change (see page 2723)	This is Change, a member of class TBasicActionLink.
	IsOnExecuteLinked (see page 2723)	This is IsOnExecuteLinked, a member of class TBasicActionLink.
	SetAction (see page 2724)	This is SetAction, a member of class TBasicActionLink.
	SetOnExecute (see page 2724)	This is SetOnExecute, a member of class TBasicActionLink.

Public Events

	Name	Description
	OnChange (see page 2725)	Occurs when an OnChange event is triggered on the action.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.

	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2721)	Instantiates and initializes a TBasicActionLink (see page 2720) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2722)	Destroys an instance of an action link object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Execute (see page 2723)	Invokes the action's Execute method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Update (see page 2724)	Invokes the action's Update method.

Public Properties

	Name	Description
	Action (see page 2725)	Indicates which action the link connects to its client.

23.1.21.2 TBasicActionLink.Create Constructor

Instantiates and initializes a TBasicActionLink (see page 2720) object.

Pascal

```
constructor Create(AClient: TObject); virtual;
```

C++

```
virtual __fastcall TBasicActionLink(TObject * AClient);
```

Description

Applications do not need to call the constructor directly. Instances of TBasicActionLink (see page 2720) are created automatically when you connect an action with a client in the action list editor. The AClient parameter is the client for the link. The link connects this client to its action.

Create ensures that the client is of the appropriate type for the particular type of link. Descendant classes can progressively specify more customized client types. For example, there are action link classes for controls, for windowed controls, and for button controls.

See Also

Action (see page 5797)

ActionLink (see page 5798)

23.1.21.3 TBasicActionLink.Destroy Destructor

Destroys an instance of an action link object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBasicActionLink();
```

Description

You do not need to destroy the action link; it is automatically destroyed by the client that was passed to its constructor when it was created.

If you want to destroy the action link manually, do not call the destructor directly. Call Free instead. Free checks that the link is not nil, and only then invokes the destructor.

See Also

Free (see page 22300)

23.1.21.4 TBasicActionLink Fields

The fields of the TBasicActionLink class are listed here.

Protected Fields

	Name	Description
	FAction (see page 2722)	This is FAction, a member of class TBasicActionLink.

23.1.21.4.1 TBasicActionLink.FAction Field

Pascal

```
FAction: TBasicAction;
```

C++

```
TBasicAction FAction;
```

Description

This is FAction, a member of class TBasicActionLink.

23.1.21.5 TBasicActionLink Methods

The methods of the TBasicActionLink class are listed here.

Protected Methods

	Name	Description
	AssignClient (see page 2723)	This is AssignClient, a member of class TBasicActionLink.
	Change (see page 2723)	This is Change, a member of class TBasicActionLink.
	IsOnExecuteLinked (see page 2723)	This is IsOnExecuteLinked, a member of class TBasicActionLink.
	SetAction (see page 2724)	This is SetAction, a member of class TBasicActionLink.
	SetOnExecute (see page 2724)	This is SetOnExecute, a member of class TBasicActionLink.

Public Methods

	Name	Description
	Execute (see page 2723)	Invokes the action's Execute method.
	Update (see page 2724)	Invokes the action's Update method.

23.1.21.5.1 TBasicActionLink.AssignClient Method

Pascal

```
procedure AssignClient(AClient: TObject); virtual;
```

C++

```
virtual __fastcall AssignClient(TObject * AClient);
```

Description

This is AssignClient, a member of class TBasicActionLink.

23.1.21.5.2 TBasicActionLink.Change Method

Pascal

```
procedure Change; virtual;
```

C++

```
virtual __fastcall Change();
```

Description

This is Change, a member of class TBasicActionLink.

23.1.21.5.3 TBasicActionLink.Execute Method

Invokes the action's Execute method.

Pascal

```
function Execute(AComponent: TComponent = nil): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean Execute(TComponent * AComponent = nil);
```

Description

Execute calls the action's Execute method and returns true if the action has an assigned OnExecute event handler, false otherwise.

AComponent is the client component that the user clicked to execute the action.

See Also

Execute (see page 2712)

OnExecute (see page 2716)

23.1.21.5.4 TBasicActionLink.IsOnExecuteLinked Method

Pascal

```
function IsOnExecuteLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsOnExecuteLinked();
```

Description

This is IsOnExecuteLinked, a member of class TBasicActionLink.

23.1.21.5.5 TBasicActionLink.SetAction Method

Pascal

```
procedure SetAction(Value: TBasicAction); virtual;
```

C++

```
virtual __fastcall SetAction(TBasicAction Value);
```

Description

This is SetAction, a member of class TBasicActionLink.

23.1.21.5.6 TBasicActionLink.SetOnExecute Method

Pascal

```
procedure SetOnExecute(Value: TNotifyEvent); virtual;
```

C++

```
virtual __fastcall SetOnExecute(TNotifyEvent Value);
```

Description

This is SetOnExecute, a member of class TBasicActionLink.

23.1.21.5.7 TBasicActionLink.Update Method

Invokes the action's Update method.

Pascal

```
function Update: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean Update();
```

Description

Update calls the action's Update method and returns true if the action has an assigned OnUpdate event handler, false otherwise. The action link's Update method is called if the client is linked to the action for this event.

See Also

Update (see page 2714)

OnUpdate (see page 2718)

23.1.21.6 TBasicActionLink Properties

The properties of the TBasicActionLink class are listed here.

Public Properties

	Name	Description
	Action (see page 2725)	Indicates which action the link connects to its client.

23.1.21.6.1 TBasicActionLink.Action Property

Indicates which action the link connects to its client.

Pascal

```
property Action: TBasicAction;
```

C++

```
__property TBasicAction Action;
```

Description

Action is the action object that the link associates with its client.

See Also

Action (see page 5797)

ActionLink (see page 5798)

23.1.21.7 TBasicActionLink Events

The events of the TBasicActionLink class are listed here.

Public Events

	Name	Description
	OnChange (see page 2725)	Occurs when an OnChange event is triggered on the action.

23.1.21.7.1 TBasicActionLink.OnChange Event

Occurs when an OnChange event is triggered on the action.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler when you want to respond to a change that occurs when the associated action fires an OnChange event.

See Also

RegisterChanges (see page 2713)

23.1.22 TBits Class

TBits stores an array of Boolean values.

Class Hierarchy

Classes.TBits

Pascal

```
TBits = class;
```

C++

```
class TBits;
```

File

Classes

Description

Use TBits to store and access an indefinite number of boolean values. TBits can store as many of boolean values as can fit in available memory, automatically expanding its storage space as needed. If the number of boolean values is limited to 32, the same functionality can be achieved using a 32-bit integer with the bitwise AND (Delphi) or & (C++) and OR (Delphi) or | (C++) operators.

See Also

TList (see page 2868)

23.1.22.1 TBits Members

The following tables list the members exposed by TBits.

Public Methods

	Name	Description
	Destroy (see page 2726)	Destroys an instance of TBits (see page 2725).
	OpenBit (see page 2727)	Returns the index of the first false value.

Public Properties

	Name	Description
	Bits (see page 2727)	Provides indexed access to the array of Boolean values that the TBits (see page 2725) object represents.
	Size (see page 2728)	Represents the number of boolean values the TBits (see page 2725) object can hold.

23.1.22.2 TBits.Destroy Destructor

Destroys an instance of TBits (see page 2725).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBits();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TBits (see page 2725) object is not nil and only then calls Destroy.

Destroy frees the memory used to store the array of boolean values.

See Also

Bits (see page 2727)

Free (see page 22300)

23.1.22.3 TBits Methods

The methods of the TBits class are listed here.

Public Methods

	Name	Description
	OpenBit (see page 2727)	Returns the index of the first false value.

23.1.22.3.1 TBits.OpenBit Method

Returns the index of the first false value.

Pascal

```
function OpenBit: Integer;
```

C++

```
__fastcall int OpenBit();
```

Description

Use OpenBit to find the first bit in the array that has not been set to true.

See Also

Bits (see page 2727)

23.1.22.4 TBits Properties

The properties of the TBits class are listed here.

Public Properties

	Name	Description
	Bits (see page 2727)	Provides indexed access to the array of Boolean values that the TBits (see page 2725) object represents.
	Size (see page 2728)	Represents the number of boolean values the TBits (see page 2725) object can hold.

23.1.22.4.1 TBits.Bits Property

Provides indexed access to the array of Boolean values that the TBits (see page 2725) object represents.

Pascal

```
property Bits [Index: Integer]: Boolean;
```

C++

```
__property Boolean Bits[int Index];
```

Description

Use Bits to read or set a particular Boolean value, as indexed by the Index parameter. If Index is not in the range 0..Size (see

page 2728) - 1, an EBitsError (see page 2693) exception is raised.

Note: In Delphi, Bits is the default property for TBits (see page 2725). This means that the property name can be omitted in code. That is, the statement

MyBitsObject.Bits[0];

Can be written

MyBitsObject[0];

Note: In C++, Bits can be accessed using the [] operator, to achieve an affect similar to the default property in Delphi.

See Also

EBitsError (see page 2693)

Size (see page 2728)

operator_sb

23.1.22.4.2 TBits.Size Property

Represents the number of boolean values the TBits (see page 2725) object can hold.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

Set Size to expand or shrink the number of values in the array. When expanding the size, new entries are set to false. When shrinking the size, values whose indexes are less than or equal to the new size are preserved; entries whose indexes are greater than the new size are lost.

See Also

Bits (see page 2727)

23.1.23 TClassFinder Class

Class Hierarchy

Pascal

```
TClassFinder = class;
```

C++

```
class TClassFinder;
```

File

Classes

Description

This is class Classes.TClassFinder.

23.1.23.1 TClassFinder Members

The following tables list the members exposed by TClassFinder.

Public Methods

	Name	Description
	Create (see page 2729)	Creates a TClassFinder (see page 2728) class instance.
V	Destroy (see page 2729)	Frees the memory associated with the TClassFinder (see page 2728) object.
	GetClass (see page 2730)	Retrieves a registered, persistent class reference by name.
	GetClasses (see page 2731)	Retrieves the registered, persistent class references.

23.1.23.2 TClassFinder.Create Constructor

Creates a TClassFinder (see page 2728) class instance.

Pascal

```
constructor Create(AClass: TPersistentClass = nil; AIncludeActiveGroups: Boolean = False);
```

C++

```
__fastcall TClassFinder(TPersistentClass AClass = nil, Boolean AIncludeActiveGroups = False);
```

Description

Call Create to instantiate a TClassFinder (see page 2728) object at runtime.

The **AClass** optional parameter limits the scope of returned classes to the specified class and its ancestors. A nil value does not limit the returned classes.

If true, the **AIncludeActiveGroups** optional parameter will append to the list of classes those that are active in the current application.

See Also

TPersistentClass (see page 3299)

23.1.23.3 TClassFinder.Destroy Destructor

Frees the memory associated with the TClassFinder (see page 2728) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TClassFinder();
```

Description

Do not call Destroy directly. Instead, use the delete keyword on the object, which causes Destroy to be invoked automatically.

See Also

delete

23.1.23.4 TClassFinder Methods

The methods of the TClassFinder class are listed here.

Public Methods

	Name	Description
	GetClass (see page 2730)	Retrieves a registered, persistent class reference by name.
	GetClasses (see page 2731)	Retrieves the registered, persistent class references.

23.1.23.4.1 TClassFinder.GetClass Method

Retrieves a registered, persistent class reference by name.

Pascal

```
function GetClass(const AClassName: string): TPersistentClass;
```

C++

```
__fastcall TPersistentClass GetClass(const AnsiString AClassName);
```

Description

Call GetClass to retrieve a class reference of the specified **AClassName** name.

A nil value is returned if the specified class is not in the list of registered persistent classes.

C++ Examples:

```
/*
The following example allows users to type the name of a
graphics class into an edit control and when a button is
pressed, the name of the default extension for that class is
displayed in another edit control.  In the form's OnCreate
event handler (or a similar place) you must register the
graphics classes so that GetClass can find them.
*/
TMetaClass *MetaClass;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClassFinder *myClassFinder = new TClassFinder(MetaClass, False);
  TGraphicClass mygraphclass = (TGraphicClass) (myClassFinder->GetClass(Edit2->Text));
//  TGraphicClass mygraphclass = (TGraphicClass) GetClass(Edit2->Text);
  Edit1->Text = GraphicExtension(mygraphclass);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MetaClass = __classid(TIcon); // ico
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(Graphics::TBitmap); // bmp
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TMetafile); // emf
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TShape);  // nographic extension
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TImage); // nographic extension
  RegisterClasses(&MetaClass, 0);
}
```

Delphi Examples:

```
{
The following example allows users to type the name of a
graphics class into an edit control and when a button is
pressed, the name of the default extension for that class is
displayed in another edit control.  In the form's OnCreate
event handler (or a similar place) you must register the
graphics classes so that GetClass can find them.
}

uses ExtCtrls;

procedure TForm1.Button1Click(Sender: TObject);
var
  myClassFinder: TClassFinder;
  mygraphclass: TGraphicClass;
begin
  { make sure the classes are registered so that GetClass will work -- }
  { Usually, this goes in the initialization section where it is only executed once }
  RegisterClasses([TIcon, TBitmap, TButton, TShape, TImage, TMetafile]);
  try
    myClassFinder := TClassFinder.Create(TIcon, False);
    mygraphclass := TGraphicClass(myClassFinder.GetClass(Edit2.Text));
//    mygraphclass := TGraphicClass(GetClass(Edit2.Text));  // works as well
    Edit1.Text := GraphicExtension(mygraphclass);
  except
    on EAccessViolation do
      MessageDlg('Invalid class name.', mtError, [mbOK], 0)
  end;
end;
```

See Also

GetClasses (see page 2731)

23.1.23.4.2 TClassFinder.GetClasses Method

Retrieves the registered, persistent class references.

Pascal

```
procedure GetClasses(Proc: TGetClass);
```

C++

```
__fastcall GetClasses(TGetClass Proc);
```

Description

Call GetClasses to retrieve class references of all registered, persistent classes.

Proc specifies a call back procedure that is called for each class reference in the list. For example, you might declare such a procedure to add the returned class names to a list.

See Also

GetClass (see page 2730)

23.1.24 TCollection Class

TCollection is a container for TCollectionItem (see page 2755) objects.

Class Hierarchy

Pascal

```
TCollection = class(TPersistent);
```

C++

```
class TCollection : public TPersistent;
```

File

Classes

Description

Each TCollection holds a group of TCollectionItem (see page 2755) descendants. TCollection maintains an index of the collection items in its Items (see page 2751) array. The Count (see page 2749) property contains the number of items in the collection. Use the Add (see page 2736) and Delete (see page 2741) methods to add items to the collection and delete items from the collection.

Objects descended from TCollection can contain objects descended from TCollectionItem (see page 2755). Thus, for each TCollection descendant, there is a corresponding TCollectionItem (see page 2755) descendant. The following table lists some typical descendants of TCollection with the corresponding TCollectionItem (see page 2755) descendant and the component that uses each pair.

TCollection descendant	TCollectionItem (see page 2755) descendant	Component
TAggregates	TAggregate	TClientDataSet
TCookieCollection	TCookie	TWebResponse
TCoolBands	TCoolBand	TCoolBar
TDBGridColumns	TColumn	TDBGrid
TDependencies	TDependency	TService
TDisplayDims	TDisplayDim	TDecisionGrid
TFieldDefs	TFieldDef	TDataSet
THeaderSections	THeaderSection	THeaderControl
TIndexDefs	TIndexDef	TTable
TListColumns	TListColumn	TListView
TParams	TParam	many datasets
TStatusPanels	TStatusPanel	TStatusBar
TWorkAreas	TWorkArea	TListView

The controls that use TCollection and TCollectionItem (see page 2755) descendants have a published property that holds a collection. (For example, the Panels property of TStatusBar holds a TStatusPanels.) A standard property editor, referred to generically as the Collection editor, can be invoked from the Object Inspector to edit the items in the collection.

Note: When writing a TCollection descendant that is used by another control, be sure to override the protected GetOwner (see page 2932) method of the collection so that it can appear in the Object Inspector.

See Also

TCollectionItem (see page 2755)

Working with Lists

23.1.24.1 TCollection Members

The following tables list the members exposed by TCollection.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 2736)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 2734)	Creates and initializes a collection.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.

V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 2751)	Lists the items in the collection.

23.1.24.2 TCollection.Create Constructor

Creates and initializes a collection.

Pascal

```
constructor Create(ItemClass: TCollectionItemClass);
```

C++

```
__fastcall TCollection(TCollectionItemClass ItemClass);
```

Description

Call Create to instantiate a TCollection (see page 2731) object at runtime. Typically, TCollection (see page 2731) descendants are created by a component that uses the collection to implement a property.

ItemClass (see page 2751) identifies TCollectionItem (see page 2755) descendant that must be used to represent the items in the collection. The Add (see page 2736) method uses this class to create items of the appropriate type.

See Also

Add (see page 2736)

23.1.24.3 TCollection.Destroy Destructor

Destroys the collection and each item in it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCollection();
```

Description

Destroy calls the Clear (see page 2741) method to free each item referenced in the Items (see page 2751) array, then destroys the collection itself.

See Also

Clear (see page 2741)

Free (see page 22300)

23.1.24.4 TCollection Methods

The methods of the TCollection class are listed here.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 2736)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	Clear (see page 2741)	Deletes all items from the collection.
	Delete (see page 2741)	Deletes a single item from the collection.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.

	Owner (see page 2746)	Returns the Owner of the collection.

23.1.24.4.1 TCollection.Add Method

Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.

Pascal

```
function Add: TCollectionItem;
```

C++

```
__fastcall TCollectionItem Add();
```

Description

Call Add to create an item in the collection. The new item is placed at the end of the Items (see page 2751) array.

Add returns the new collection item.

See Also

Items (see page 2751)

Insert (see page 2746)

Create (see page 2734)

Delete (see page 2741)

23.1.24.4.2 TCollection.Added Method

Responds when items are added to the collection.

Pascal

```
procedure Added(var Item: TCollectionItem); virtual; deprecated;
```

C++

```
virtual __fastcall Added(TCollectionItem Item);
```

Description

Applications can't call the protected Added method. It is called automatically immediately after items are added to the collection.

Item is the item that was just added to the collection.

Added is a deprecated method. Descendant classes should override the Notify (see page 2746) method instead when responding to changes in the list of items.

See Also

Add (see page 2736)

Items (see page 2751)

Notify (see page 2746)

23.1.24.4.3 TCollection.Assign Method

Copies the contents of another collection to the object where the method is executed.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use Assign to copy the contents of one TCollection (see page 2731) instance to another. The Assign method deletes all items from the destination collection (the object where it is executed), then adds a copy of each item in the source collection's Items (see page 2751) array.

Source is another object (typically another collection) that contains the items that replace this collection's items.

Delphi Examples:

```
{
  This example saves params from a ClientDataSet, alters them
  and restores them.
}
procedure TForm1.SaveParamsClick(Sender: TObject);
begin
  { Save the parameters for the TDataSet }
  SavedParams.Assign(CDS.Params);
end;

procedure TForm1.AlterParamsClick(Sender: TObject);
var
  I : Integer;
  ListItem: string;
begin
  CDS.Params.CreateParam(ftString, 'stringParam', ptInput);  // you can also edit the params
in the CDS dynamically
  CDS.Params.CreateParam(ftSmallInt, 'smallIntParam', ptInput);
  CDS.Params.CreateParam(ftInteger, 'intParam', ptInput);
  CDS.Params.CreateParam(ftWord, 'wordParam', ptInput);
  CDS.Params.CreateParam(ftBoolean, 'boolParam', ptInput);
  CDS.Params.CreateParam(ftFloat, 'floatParam', ptInput);
  CDS.Params.CreateParam(ftCurrency, 'currencyParam', ptInput);
  CDS.Params.CreateParam(ftBCD, 'bcdParam', ptInput);
  CDS.Params.CreateParam(ftDate, 'dateParam', ptInput);
  CDS.Params.CreateParam(ftTime, 'timeParam', ptInput);
  CDS.Params.CreateParam(ftDateTime, 'dateTimeParam', ptInput);
  { Call FetchParams to ensure parameters reflect server metadata }
  CDS.FetchParams;
  with CDS.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
              Items[I].AsBoolean := True
            else
              Items[I].AsBoolean := False;
          end;
        ftFloat:
          Items[I].AsFloat := StrToFloat(ListItem);
```

```
          ftCurrency:
            Items[I].AsCurrency := StrToFloat(ListItem);
          ftBCD:
            Items[I].AsBCD := StrToCurr(ListItem);
          ftDate:
            Items[I].AsDate := StrToDate(ListItem);
          ftTime:
            Items[I].AsTime := StrToTime(ListItem);
          ftDateTime:
            Items[I].AsDateTime := StrToDateTime(ListItem);
        end;
      end;
    end;
  end;

  procedure TForm1.RestoreParamsClick(Sender: TObject);
  begin
    { Restore the parameters to TDataSet }
    CDS.Params.AssignValues(SavedParams);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    { Initialize SavedParams }
    SavedParams := TParams.Create;
    ListBox1.Items.Add('Here is a string');
    ListBox1.Items.Add('4');
    ListBox1.Items.Add('3456789');
    ListBox1.Items.Add('34');
    ListBox1.Items.Add('0');
    ListBox1.Items.Add('12.56');
    ListBox1.Items.Add('4.23');
    ListBox1.Items.Add('99.5');
    ListBox1.Items.Add('4/12/53');
    ListBox1.Items.Add('7:45:00');
    ListBox1.Items.Add('4/12/53 7:45:00');
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    SavedParams.Free;
  end;
  {
    This example saves params from a ClientDataSet, alters them
    and restores them.
  }
  procedure TForm1.SaveParamsClick(Sender: TObject);
  begin
    { Save the parameters for the TDataSet }
    SavedParams.Assign(CDS.Params);
  end;

  procedure TForm1.AlterParamsClick(Sender: TObject);
  var
    I : Integer;
  begin
    CDS.Params.CreateParam(ftInteger, 'StateParam', ptInput);
    CDS.Params.CreateParam(ftInteger, 'MyParam', ptInput);
    for I := 0 to CDS.Params.Count - 1 do
      if (CDS.Params.Items[I].IsNull) and
         (CDS.Params.Items[I].DataType = ftInteger) then
        { Items is the default property, so you can omit its name }
        CDS.Params[I].AsInteger := -1;
  end;

  procedure TForm1.RestoreParamsClick(Sender: TObject);
```

```
begin
  { Restore the parameters to TDataSet }
  CDS.Params.AssignValues(SavedParams);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { Initialize SavedParams }
  SavedParams := TParams.Create;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  SavedParams.Free;
end;
```

See Also

Assign (see page 2930)

TListColumns (see page 4245)

23.1.24.4.4 TCollection.BeginUpdate Method

Suspends screen repainting.

Pascal

```
procedure BeginUpdate; virtual;
```

C++

```
virtual __fastcall BeginUpdate();
```

Description

The BeginUpdate method suspends screen repainting until the EndUpdate (see page 2742) method is called. Use BeginUpdate to speed processing and avoid flicker while items are added to or deleted from a collection.

C++ Examples:

```
/*
The following example adds a panel to the status bar control
when the user clicks the button and adds a caption to the
panel. The code uses BeginUpdate and EndUpdate to prevent
repaints until the operation is complete.  A try...finally
block ensures EndUpdate is called even when an exception
occurs.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int PanelIndex;

  StatusBar1->Panels->BeginUpdate();
  PanelIndex = StatusBar1->Panels->Count - 1;
  try
  {
    StatusBar1->Panels->Add();
    PanelIndex++;
    StatusBar1->Panels->Items[PanelIndex]->Text = "Panel" + IntToStr(PanelIndex);
  }
  __finally
  {
    StatusBar1->Panels->EndUpdate();
  }
}
```

Delphi Examples:

```
{
The following example adds a panel to the status bar control
when the user clicks the button and adds a caption to the
panel. The code uses BeginUpdate and EndUpdate to prevent
repaints until the operation is complete.  A try...finally
block ensures EndUpdate is called even when an exception
occurs.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  PanelIndex : Integer;
begin
  with StatusBar1 do
  begin
    Panels.BeginUpdate;
    PanelIndex := StatusBar1.Panels.Count - 1;
    try
      Panels.Add;
      Inc(PanelIndex);
      Panels.Items[PanelIndex].Text :=
        'Panel' + IntToStr(PanelIndex);
    finally
      Panels.EndUpdate;
    end;
  end;
end;
```

See Also

EndUpdate (see page 2742)

23.1.24.4.5 TCollection.Changed Method

Responds when the collection or any of its items changes.

Pascal

```
procedure Changed;
```

C++

```
__fastcall Changed();
```

Description

Changed is called automatically when items in the collection change or when the EndUpdate (see page 2742) method signals that an update is complete. It checks the value of UpdateCount (see page 2753), and if it is 0, calls the Update (see page 2747) method, which performs any necessary updates.

When writing a TCollection (see page 2731) descendant, there is no need to call Changed. Instead, bracket any changes by calls to BeginUpdate (see page 2739) and EndUpdate (see page 2742).

See Also

Changed (see page 2758)

Update (see page 2747)

EndUpdate (see page 2742)

UpdateCount (see page 2753)

23.1.24.4.6 TCollection.Clear Method

Deletes all items from the collection.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Clear empties the Items (see page 2751) array and destroys each TCollectionItem (see page 2755).

See Also

Items (see page 2751)

TCollectionItem (see page 2755)

Delete (see page 2741)

23.1.24.4.7 TCollection.Delete Method

Deletes a single item from the collection.

Pascal

```
procedure Delete(Index: Integer);
```

C++

```
__fastcall Delete(int Index);
```

Description

Delete removes the specified collection item, moving up any items that come after that item in the Items (see page 2751) property array.

Index identifies the item to delete. This is the index of the item in the Items (see page 2751) property array. That is, 0 specifies the first item, 1 specifies the second item, and so on.

See Also

Items (see page 2751)

TCollectionItem (see page 2755)

Clear (see page 2741)

23.1.24.4.8 TCollection.Deleting Method

Responds when items are deleted from the collection.

Pascal

```
procedure Deleting(Item: TCollectionItem); virtual; deprecated;
```

C++

```
virtual __fastcall Deleting(TCollectionItem Item);
```

Description

Applications can't call the protected Deleting method. The Delete (see page 2741) method calls Deleting immediately before it

removes an item from the collection.

Item is the item that is about to be removed.

Deleting is a deprecated method. Descendant classes should override the Notify (see page 2746) method instead when responding to changes in the list of items.

See Also

Delete (see page 2741)

Notify (see page 2746)

23.1.24.4.9 TCollection.EndUpdate Method

Re-enables screen repainting.

Pascal

```
procedure EndUpdate; virtual;
```

C++

```
virtual __fastcall EndUpdate();
```

Description

Use EndUpdate to re-enable screen repainting that was turned off with the BeginUpdate (see page 2739) method.

C++ Examples:

```
/*
The following example adds a panel to the status bar control
when the user clicks the button and adds a caption to the
panel. The code uses BeginUpdate and EndUpdate to prevent
repaints until the operation is complete.  A try...finally
block ensures EndUpdate is called even when an exception
occurs.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int PanelIndex;

  StatusBar1->Panels->BeginUpdate();
  PanelIndex = StatusBar1->Panels->Count - 1;
  try
  {
    StatusBar1->Panels->Add();
    PanelIndex++;
    StatusBar1->Panels->Items[PanelIndex]->Text = "Panel" + IntToStr(PanelIndex);
  }
  __finally
  {
    StatusBar1->Panels->EndUpdate();
  }
}
```

Delphi Examples:

```
{
The following example adds a panel to the status bar control
when the user clicks the button and adds a caption to the
panel. The code uses BeginUpdate and EndUpdate to prevent
repaints until the operation is complete.  A try...finally
block ensures EndUpdate is called even when an exception
occurs.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  PanelIndex : Integer;
begin
  with StatusBar1 do
  begin
    Panels.BeginUpdate;
    PanelIndex := StatusBar1.Panels.Count - 1;
    try
      Panels.Add;
      Inc(PanelIndex);
      Panels.Items[PanelIndex].Text :=
        'Panel' + IntToStr(PanelIndex);
    finally
      Panels.EndUpdate;
    end;
  end;
end;
```

See Also

BeginUpdate (see page 2739)

23.1.24.4.10 TCollection.FindItemID Method

Returns the item with the specified ID.

Pascal

```
function FindItemID(ID: Integer): TCollectionItem;
```

C++

```
__fastcall TCollectionItem FindItemID(int ID);
```

Description

The FindItemID method returns the item in the collection whose ID property is passed to it as a parameter. If no item has the specified ID, FindItemID returns nil (Delphi) or NULL (C++).

See Also

ID (see page 2761)

23.1.24.4.11 TCollection.GetAttr Method

Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.

Pascal

```
function GetAttr(Index: Integer): string; dynamic;
```

C++

```
__fastcall AnsiString GetAttr(int Index);
```

Description

TCollection (see page 2731) descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. The GetAttr method returns the name of an attribute.

Index identifies the attribute whose name is requested. This is a value between 0 and n-1, where n is the value returned by GetAttrCount (see page 2744).

As implemented by TCollection (see page 2731), GetAttr always returns an empty string, because TCollection (see page 2731) defines no custom attributes.

See Also

GetAttrCount (see page 2744)

GetItemAttr (see page 2745)

23.1.24.4.12 TCollection.GetAttrCount Method

Returns the number of custom attributes associated with items in the collection.

Pascal

```
function GetAttrCount: Integer; dynamic;
```

C++

```
__fastcall int GetAttrCount();
```

Description

TCollection (see page 2731) descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetAttrCount returns the number of distinct attributes assigned to each item in the collection.

As implemented in TCollection (see page 2731), GetAttrCount always returns 0, because TCollection (see page 2731) defines no custom attributes.

See Also

GetAttr (see page 2743)

GetItemAttr (see page 2745)

23.1.24.4.13 TCollection.GetEnumerator Method

Returns a TCollection (see page 2731) enumerator.

Pascal

```
function GetEnumerator: TCollectionEnumerator;
```

C++

```
__fastcall TCollectionEnumerator GetEnumerator();
```

Description

GetEnumerator returns a TCollectionEnumerator (see page 2753) reference, which enumerates all items in the collection.

To do so, call the TCollectionEnumerator (see page 2753) GetCurrent method within a While MoveNext do loop.

See Also

TCollectionEnumerator (see page 2753)

23.1.24.4.14 TCollection.GetItem Method

Returns a specified item in the collection.

Pascal

```
function GetItem(Index: Integer): TCollectionItem;
```

C++

```
__fastcall TCollectionItem GetItem(int Index);
```

Description

GetItem is the protected read implementation of the Items (see page 2751) property.

See Also

Items (see page 2751)

23.1.24.4.15 TCollection.GetItemAttr Method

Returns the value of a custom attribute assigned to one of the collection's items.

Pascal

```
function GetItemAttr(Index: Integer; ItemIndex: Integer): string; dynamic;
```

C++

```
__fastcall AnsiString GetItemAttr(int Index, int ItemIndex);
```

Description

TCollection (see page 2731) descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetItemAttr returns the value of one of these attributes for a specified item in the collection.

Index identifies which attribute's value is desired. This is a value between 0 and n-1, where n is the value returned by GetAttrCount (see page 2744).

ItemIndex identifies the item whose attribute value is desired. This is an index into the Items (see page 2751) property array.

As implemented in TCollection (see page 2731), GetItemAttr always returns an empty string, because TCollection (see page 2731) defines no custom attributes.

See Also

GetAttr (see page 2743)

GetAttrCount (see page 2744)

23.1.24.4.16 TCollection.GetNamePath Method

Returns a string used by the Object Inspector.

Pascal

```
function GetNamePath: string; override;
```

C++

```
virtual __fastcall AnsiString GetNamePath();
```

Description

If the collection has no owner, GetNamePath returns the name of the collection's actual (runtime) type. If the collection is owned, GetNamePath returns the owner's name followed, if applicable, by a dot and the name of the owner's property that holds the collection. For example, GetNamePath might return "TreeView1.Items (see page 2751)".

Note: For a collection to have an owner, it must override the GetOwner (see page 2932) method.

See Also

GetOwner (see page 2932)

23.1.24.4.17 TCollection.Insert Method

Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.

Pascal

```
function Insert(Index: Integer): TCollectionItem;
```

C++

```
__fastcall TCollectionItem Insert(int Index);
```

Description

Call Insert to add a new item at a specified position in the collection. Existing items (starting from the specified position) are moved up in the Items (see page 2751) array.

Insert returns the new collection item.

See Also

Add (see page 2736)

Items (see page 2751)

23.1.24.4.18 TCollection.Notify Method

Responds when items are added to or removed from the collection.

Pascal

```
procedure Notify(Item: TCollectionItem; Action: TCollectionNotification); virtual;
```

C++

```
virtual __fastcall Notify(TCollectionItem Item, TCollectionNotification Action);
```

Description

Notify is called automatically when the items in the collection change.

Item is the item that was just added to or that is about to be removed from the collection.

Action indicates whether item was added, is about to be removed, or is about to be deleted:

As implemented in TCollection (see page 2731), Notify calls Added (see page 2736) when Action is cnAdded and calls Deleting (see page 2741) when Action is cnDeleting. TCollection (see page 2731) ignores the cnExtracting action. Descendant classes can override Notify to modify this behavior.

See Also

Added (see page 2736)

Deleting (see page 2741)

23.1.24.4.19 TCollection.Owner Method

Returns the Owner of the collection.

Pascal

```
function Owner: TPersistent;
```

C++

```
__fastcall TPersistent * Owner();
```

Description

Call Owner to obtain a reference to the object that owns this collection. Typically, the owner uses the collection to implement one of its properties.

23.1.24.4.20 TCollection.SetItem Method

Copies the properties of another item to a specified item in the collection.

Pascal

```
procedure SetItem(Index: Integer; Value: TCollectionItem);
```

C++

```
__fastcall SetItem(int Index, TCollectionItem Value);
```

Description

SetItem is the protected write implementation of the Items (see page 2751) property. It calls the Assign (see page 2736) method of the item specified by Index, so that the properties of the item specified by Value are copied to that item.

See Also

Items (see page 2751)

Assign (see page 2930)

23.1.24.4.21 TCollection.SetItemName Method

Initializes the name of a newly inserted collection item.

Pascal

```
procedure SetItemName(Item: TCollectionItem); virtual;
```

C++

```
virtual __fastcall SetItemName(TCollectionItem Item);
```

Description

The Insert (see page 2746) method calls SetItemName to initialize the Name property of items when it inserts them into the collection.

As implemented in TCollection (see page 2731), SetItemName does nothing. Some TCollection (see page 2731) descendants override this method to provide collection items with default names.

See Also

Insert (see page 2746)

Name (see page 2805)

23.1.24.4.22 TCollection.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); virtual;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection (see page 2731), Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed (see page 2740)

Changed (see page 2758)

23.1.24.5 TCollection Properties

The properties of the TCollection class are listed here.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 2751)	Lists the items in the collection.

23.1.24.5.1 TCollection.Capacity Property

Provides access to the internal TList (see page 2868) Capacity property.

Pascal

```
property Capacity: Integer;
```

C++

```
__property int Capacity;
```

Description

The Capacity property specifies the allocated size of the array of pointers maintained by the TList (see page 2868) object. This value is set to the number of pointers the list will need to contain.

23.1.24.5.2 TCollection.Count Property

Returns the number of items in the collection.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Count contains the number of items in the Items (see page 2751) array. Since Items (see page 2751) is indexed starting with 0, the value of Count is always one greater than the index of the last member of Items (see page 2751).

C++ Examples:

```
/*
The following example adds a panel to the status bar control
when the user clicks the button and adds a caption to the
panel. The code uses BeginUpdate and EndUpdate to prevent
repaints until the operation is complete.  A try...finally
block ensures EndUpdate is called even when an exception
occurs.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int PanelIndex;

  StatusBar1->Panels->BeginUpdate();
  PanelIndex = StatusBar1->Panels->Count - 1;
  try
  {
    StatusBar1->Panels->Add();
    PanelIndex++;
    StatusBar1->Panels->Items[PanelIndex]->Text = "Panel" + IntToStr(PanelIndex);
  }
  __finally
  {
    StatusBar1->Panels->EndUpdate();
  }
}
/*
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
*/
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2->Active = False;
  // Get the current available indices
  CDS2->IndexDefs->Update();
  // Find a field named "Field2"
  for (int I = 0; I < CDS2->IndexDefs->Count; I++)
    if (CDS2->IndexDefs->Items[I]->Fields == "Field2")
    {
      // set that index as the current index for the dataset}
      CDS2->IndexName = CDS2->IndexDefs->Items[I]->Name;
    }
  CDS2->Active = True;
}
/*
This example requires a TClientDataSet and a button on a form.
```

```
Put swSystem in the uses clause for gsAppPath.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
for (int I = 0; I < CDS->Params->Count; I++)
{
  if (CDS->Params->Items[I]->IsNull &&
      CDS->Params->Items[I]->DataType == ftInteger)
    CDS->Params->Items[I]->AsInteger = -1;
}}

#include <swSystem.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CDS->LoadFromFile(gsAppPath + "../CDS.XML");
  CDS->Params->CreateParam(ftInteger, "StateParam", ptInput);
  CDS->Params->CreateParam(ftInteger, "MyParam", ptInput);
}
```

Delphi Examples:

```
{
The following example adds a panel to the status bar control
when the user clicks the button and adds a caption to the
panel. The code uses BeginUpdate and EndUpdate to prevent
repaints until the operation is complete.  A try...finally
block ensures EndUpdate is called even when an exception
occurs.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  PanelIndex : Integer;
begin
  with StatusBar1 do
  begin
    Panels.BeginUpdate;
    PanelIndex := StatusBar1.Panels.Count - 1;
    try
      Panels.Add;
      Inc(PanelIndex);
      Panels.Items[PanelIndex].Text :=
        'Panel' + IntToStr(PanelIndex);
    finally
      Panels.EndUpdate;
    end;
  end;
end;
{
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
}
procedure TForm1.Button3Click(Sender: TObject);
var
  I : Integer;
begin
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2.Active := False;
  { Get the current available indices }
  CDS2.IndexDefs.Update;
  { Find a field named 'Field2' }
  for I := 0 to CDS2.IndexDefs.Count - 1 do
    if CDS2.IndexDefs.Items[I].Fields = 'Field2' then
    begin
      { set that index as the current index for the dataset}
      CDS2.IndexName := CDS2.IndexDefs.Items[I].Name;
```

```
      end;
    CDS2.Active := True;
  end;
  {
  This example requires a TClientDataSet and a button on a form.
  Put swSystem in the uses clause for gsAppPath.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    I : Integer;
  begin
  { Assign -1 to any integer parameter which does not have a value. }
  for I := 0 to CDS.Params.Count - 1 do
    if (CDS.Params.Items[I].IsNull) and
       (CDS.Params.Items[I].DataType = ftInteger) then
      { Items is the default property, so you can omit its name }
      CDS.Params[I].AsInteger := -1;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    CDS.LoadFromFile(gsAppPath + 'CDS.XML');
    CDS.Params.CreateParam(ftInteger, 'StateParam', ptInput);
    CDS.Params.CreateParam(ftInteger, 'MyParam', ptInput);
  end;
```

See Also

Items (see page 2751)

23.1.24.5.3 TCollection.ItemClass Property

Indicates the class to which the collection's items belong.

Pascal

```
property ItemClass: TCollectionItemClass;
```

C++

```
__property TCollectionItemClass ItemClass;
```

Description

ItemClass is the class (descended from TCollectionItem (see page 2755)) to which the items in the collection belong. For example, in an instance of the TCollection (see page 2731) descendant THeaderSections, the value of the ItemClass property is THeaderSection.

See Also

Items (see page 2751)

TCollectionItem (see page 2755)

23.1.24.5.4 TCollection.Items Property

Lists the items in the collection.

Pascal

```
property Items [Index: Integer]: TCollectionItem;
```

C++

```
__property TCollectionItem Items[int Index];
```

Description

Use Items to access individual items in the collection. The value of the Index parameter corresponds to the Index property of TCollectionItem (see page 2755). It represents the position of the item in the collection.

See Also

Count (see page 2749)

ItemClass (see page 2751)

Index (see page 2762)

Add (see page 2736)

Insert (see page 2746)

23.1.24.5.5 TCollection.NextID Property

Specifies a unique ID that can be assigned to the next added collection item.

Pascal

```
property NextID: Integer;
```

C++

```
__property int NextID;
```

Description

TCollection (see page 2731) uses NextID internally to assign unique identifiers to collection items. When a new item is added to the collection, it's ID property is given the value of NextID and NextID is incremented.

See Also

ID (see page 2761)

23.1.24.5.6 TCollection.PropName Property

Specifies the name of the property that the collection implements.

Pascal

```
property PropName: string;
```

C++

```
__property AnsiString PropName;
```

Description

The GetNamePath (see page 2745) method uses this protected property to assemble the name of the collection as it appears in the Object Inspector. It identifies the name of the property in the object returned by the protected GetOwner (see page 2932) method that is implemented using this collection object.

See Also

GetNamePath (see page 2745)

GetOwner (see page 2932)

23.1.24.5.7 TCollection.UpdateCount Property

Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Pascal

```
property UpdateCount: Integer;
```

C++

```
__property int UpdateCount;
```

Description

UpdateCount keeps track of calls to BeginUpdate (see page 2739) and EndUpdate (see page 2742) so that they can be nested. Every call to BeginUpdate (see page 2739) increments UpdateCount. Every call to EndUpdate (see page 2742) decrements it. When UpdateCount returns to 0, the collection updates itself to reflect all changes that occurred since the first call to BeginUpdate (see page 2739).

See Also

BeginUpdate (see page 2739)

EndUpdate (see page 2742)

23.1.25 TCollectionEnumerator Class

Enumerates through the TCollectionItems in a TCollection (see page 2731) object.

Class Hierarchy

Classes.TCollectionEnumerator

Pascal

```
TCollectionEnumerator = class;
```

C++

```
class TCollectionEnumerator;
```

File

Classes

Description

TCollectionEnumerator provides an enumerator for TCollection (see page 2731) objects.

23.1.25.1 TCollectionEnumerator Members

The following tables list the members exposed by TCollectionEnumerator.

Public Methods

	Name	Description
	Create (see page 2754)	Creates a TCollection (see page 2731) enumerator.
	GetCurrent (see page 2754)	Gets the current TCollectionItem (see page 2755).

	MoveNext (see page 2754)	Moves the enumerator to the next TCollectionItem (see page 2755) in the collection.

Public Properties

	Name	Description
	Current (see page 2755)	The current TCollection (see page 2731) object.

23.1.25.2 TCollectionEnumerator.Create Constructor

Creates a TCollection (see page 2731) enumerator.

Pascal

```
constructor Create(ACollection: TCollection);
```

C++

```
__fastcall TCollectionEnumerator(TCollection ACollection);
```

Description

Creates an instance of TCollectionEnumerator (see page 2753).

23.1.25.3 TCollectionEnumerator Methods

The methods of the TCollectionEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 2754)	Gets the current TCollectionItem (see page 2755).
	MoveNext (see page 2754)	Moves the enumerator to the next TCollectionItem (see page 2755) in the collection.

23.1.25.3.1 TCollectionEnumerator.GetCurrent Method

Gets the current TCollectionItem (see page 2755).

Pascal

```
function GetCurrent: TCollectionItem;
```

C++

```
__fastcall TCollectionItem GetCurrent();
```

Description

Use GetCurrent to get the current TCollectionItem (see page 2755) of the TCollection (see page 2731) object being enumerated.

23.1.25.3.2 TCollectionEnumerator.MoveNext Method

Moves the enumerator to the next TCollectionItem (see page 2755) in the collection.

Pascal

```
function MoveNext: Boolean;
```

C++

```
__fastcall Boolean MoveNext();
```

Description

Use MoveNext to move the TCollectionEnumerator (see page 2753) to the next TCollectionItem (see page 2755) in the TCollection (see page 2731) object.

23.1.25.4 TCollectionEnumerator Properties

The properties of the TCollectionEnumerator class are listed here.

Public Properties

	Name	Description
R	Current (see page 2755)	The current TCollection (see page 2731) object.

23.1.25.4.1 TCollectionEnumerator.Current Property

The current TCollection (see page 2731) object.

Pascal

```
property Current: TCollectionItem;
```

C++

```
__property TCollectionItem Current;
```

Description

Current contains the current TCollectionItem (see page 2755) in the TCollection (see page 2731) object.

23.1.26 TCollectionItem Class

TCollectionItem represents an item in a collection.

Class Hierarchy

Pascal

```
TCollectionItem = class(TPersistent);
```

C++

```
class TCollectionItem : public TPersistent;
```

File

Classes

Description

A TCollection (see page 2731) holds a group of TCollectionItem objects. TCollectionItems are created and destroyed by TCollection (see page 2731)'s Add and Clear methods. Each TCollectionItem has a Collection (see page 2761) property that points to the TCollection (see page 2731) object to which the item belongs.

Objects descended from TCollection (see page 2731) can contain objects descended from TCollectionItem. For example, a TDBGridColumns object contains TColumn objects; these two classes are used by TDBGrid to represent grid columns.

See Also

TCollection (see page 2731)

23.1.26.1 TCollectionItem Members

The following tables list the members exposed by TCollectionItem.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2757)	Creates and initializes a TCollectionItem (see page 2755) instance.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

23.1.26.2 TCollectionItem.Create Constructor

Creates and initializes a TCollectionItem (see page 2755) instance.

Pascal

```
constructor Create(Collection: TCollection); virtual;
```

C++

```
virtual __fastcall TCollectionItem(TCollection Collection);
```

Description

Do not call Create to instantiate a collection item. Instead, call the Add method of the collection to which the item should belong.

Collection (see page 2761) identifies the TCollection (see page 2731) instance to which the new item belongs.

See Also

Add (see page 2736)

Collection (see page 2761)

23.1.26.3 TCollectionItem.Destroy Destructor

Destroys the TCollectionItem (see page 2755) instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCollectionItem();
```

Description

Destroy is called indirectly by TCollection (see page 2731)'s Clear or Delete method.

See Also

Clear (see page 2741)

Delete (see page 2741)

Free (see page 22300)

23.1.26.4 TCollectionItem Methods

The methods of the TCollectionItem class are listed here.

Protected Methods

	Name	Description
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
V	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
V	GetOwner (see page 2759)	Returns the collection to which this item belongs.
V	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
V	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
V	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.

23.1.26.4.1 TCollectionItem.Changed Method

Updates the collection to reflect changes to the collection item.

Pascal

```
procedure Changed(AllItems: Boolean);
```

C++

```
__fastcall Changed(Boolean AllItems);
```

Description

Changed is called automatically when a collection item changes in a way that must be reflected by the collection to which the item belongs. For example, when an item's Index (see page 2762) property changes, the collection must update the indexes of all other items. Call this method in the implementation of a descendant class when making changes that require the collection to update.

The AllItems parameter indicates whether the change affects just the collection item (such as a change to its DisplayName (see page 2761) property) or whether it requires corresponding changes to other items in the collection (such as a change to the Index (see page 2762) property).

See Also

Index (see page 2762)

DisplayName (see page 2761)

Collection (see page 2761)

23.1.26.4.2 TCollectionItem.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName (see page 2761) property.

See Also

DisplayName (see page 2761)

GetNamePath (see page 2759)

23.1.26.4.3 TCollectionItem.GetNamePath Method

Returns a string used by the Object Inspector.

Pascal

```
function GetNamePath: string; override;
```

C++

```
virtual __fastcall AnsiString GetNamePath();
```

Description

GetNamePath determines the text that represents the collection in the Object Inspector. TCollectionItem (see page 2755) overrides the inherited method to return the name of the collection followed by the index of the item within that collection.

23.1.26.4.4 TCollectionItem.GetOwner Method

Returns the collection to which this item belongs.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath (see page 2759) method to find the collection that owns of the collection item, allowing the collection item to appear in the Object Inspector. It returns the value of the Collection (see page 2761) property.

Note: The Owner of a collection item is not synonymous with a component's Owner. It has no implications about the freeing of objects but merely serves to represent relationships to the Object Inspector.

See Also

GetNamePath (see page 2759)

Collection (see page 2761)

Owner (see page 2807)

23.1.26.4.5 TCollectionItem.SetCollection Method

Sets the value of the Collection (see page 2761) property.

Pascal

```
procedure SetCollection(Value: TCollection); virtual;
```

C++

```
virtual __fastcall SetCollection(TCollection Value);
```

Description

SetCollection is the protected write implementation of the Collection (see page 2761) property.

Value specifies the collection to which this item should be added.

SetCollection removes this item from the collection to which it currently belongs and adds it to the collection specified by Value.

See Also

Collection (see page 2761)

23.1.26.4.6 TCollectionItem.SetDisplayName Method

Sets the name of the collection item as it appears in the collection editor.

Pascal

```
procedure SetDisplayName(const Value: string); virtual;
```

C++

```
virtual __fastcall SetDisplayName(const AnsiString Value);
```

Description

SetDisplayName is the protected write implementation of the DisplayName (see page 2761) property.

Value specifies the name for the item that should appear in the collection editor.

See Also

DisplayName (see page 2761)

23.1.26.4.7 TCollectionItem.SetIndex Method

Moves the collection item to a specified position in the parent collection.

Pascal

```
procedure SetIndex(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetIndex(int Value);
```

Description

SetIndex is the protected write implementation of the Index (see page 2762) property. The Value parameter specifies the new (0-based) position for the item in its parent collection.

See Also

Index (see page 2762)

23.1.26.5 TCollectionItem Properties

The properties of the TCollectionItem class are listed here.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

23.1.26.5.1 TCollectionItem.Collection Property

Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.

Pascal

```
property Collection: TCollection;
```

C++

```
__property TCollection Collection;
```

Description

Each TCollectionItem (see page 2755) belongs to the TCollection (see page 2731) which creates it. The Collection property points to the collection object to which the item belongs.

23.1.26.5.2 TCollectionItem.DisplayName Property

The name displayed in the Collection (see page 2761) editor.

Pascal

```
property DisplayName: string;
```

C++

```
__property AnsiString DisplayName;
```

Description

DisplayName is the string that represents the item in the Collection (see page 2761) editor. When the item is selected in the Collection (see page 2761) editor, the DisplayName can also appear in the Object Inspector as the value of another property (for example, as the Text property of THeaderSection or TStatusPanel). By default, DisplayName is the name of the TCollectionItem (see page 2755) descendant class of which the item is an instance.

23.1.26.5.3 TCollectionItem.ID Property

A unique, permanent index for the item.

Pascal

```
property ID: Integer;
```

C++

```
__property int ID;
```

Description

In most cases, ID is the value of the item's Index (see page 2762) property when it was first added to the collection. But, unlike Index (see page 2762) values, ID values are never changed or reassigned. If an item is removed from a collection, its ID not used again.

An item's Index (see page 2762) may change when other items are deleted from the collection or when items are rearranged. ID provides a unique identifier for each item that is unchanged for the life of the collection.

See Also

Index (see page 2762)

23.1.26.5.4 TCollectionItem.Index Property

Returns the item's position in the Items array of TCollection (see page 2731).

Pascal

```
property Index: Integer;
```

C++

```
__property int Index;
```

Description

Each TCollectionItem (see page 2755) is indexed in the Items array of the TCollection (see page 2731) to which it belongs. The Index property of the TCollectionItem (see page 2755) contains the item's index value in that array. Read the value of Index to determine the collection item's position. Set the value of Index to move the collection item to a new position.

Items is a zero-based array. The first member of the collection has an index value of 0, the second member has an index value of 1, and so forth.

See Also

ID (see page 2761)

Items (see page 2751)

23.1.27 TComponent Class

TComponent is the common ancestor of all component classes.

Class Hierarchy

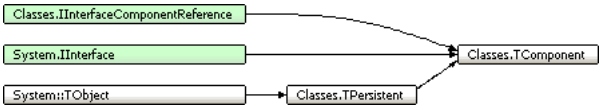

Pascal

```
TComponent = class(TPersistent, IInterface, IInterfaceComponentReference);
```

C++

```
class TComponent : public TPersistent, public IInterface, public IInterfaceComponentReference;
```

File

Classes

Description

TComponent is the base class for all components. TComponent implements the following features:

Components (see page 2799) are persistent objects that have the following capabilities:

IDE integration. The ability to appear on an IDE palette and be manipulated in a form designer.

Ownership. The ability to manage other components. If component A owns component B, then A is responsible for destroying B when A is destroyed.

Streaming and filing. Enhancements of the persistence features inherited from TPersistent (see page 2928).

COM support. Components (see page 2799) can be converted into ActiveX controls or other COM objects using wizards provided with Windows products. Components (see page 2799) can serve as wrappers for COM objects.

Note: COM features are present in all implementations of TComponent, including those provided with Linux development tools. However, these features are only useful in Windows applications, and are marked in this documentation as "Windows only". Do not use these features in cross-platform applications.

TComponent does not provide any user interface or display features. These features are provided by two classes that directly descend from TComponent.

TControl, in the QControls unit, is the base class for "visual" components in cross-platform applications.

TControl, in the Controls (see page 5711) unit, is the base class for "visual" components in Windows-only applications.

Note: The Controls (see page 5711) unit and other Windows-specific units are not provided with Linux development tools.

Components (see page 2799) that can be visible at runtime are sometimes called "visual components". Other components, which are never visible at runtime, are sometimes called "non-visual components". However it is more common to refer to "visual components" as "controls" and "non-visual components" simply as "components."

Do not create instances of TComponent. Use TComponent as a base class when declaring non-visual components that can appear on the component palette and be used in the form designer. Properties and methods of TComponent provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TControl (see page 5724)

TFiler (see page 2834)

RegisterComponents (see page 3256)

RegisterNonActiveX (see page 3260)

Using Streams to Read or Write Data

TComponent Branch

VCL Overview

23.1.27.1 TComponent Members

The following tables list the members exposed by TComponent.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.

	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

23.1.27.2 TComponent.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

```
constructor Create(AOwner: TComponent); virtual;
```

C++

```
virtual __fastcall TComponent(TComponent * AOwner);
```

Description

All objects have a Create method that constructs the object. TComponent (see page 2762) redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner (see page 2807), as indicated by the AOwner parameter

sets the ComponentStyle (see page 2805) property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner (see page 2807) parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent (see page 2762) constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components (see page 2799)

ComponentStyle (see page 2805)

Create (see page 22287)

Destroy (see page 2766)

Free (see page 22300)

Owner (see page 2807)

23.1.27.3 TComponent.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TComponent();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

Destroy (see page 22287)

DestroyComponents (see page 2771)

Destroying (see page 2772)

Free (see page 22300)

Notification (see page 2783)

Owner (see page 2807)

Release (see page 13885)

Components (see page 2799)

23.1.27.4 TComponent Fields

The fields of the TComponent class are listed here.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

23.1.27.4.1 TComponent.FComponentStyle Field

Pascal

```
FComponentStyle: TComponentStyle;
```

C++

```
TComponentStyle FComponentStyle;
```

Description

This is FComponentStyle, a member of class TComponent.

23.1.27.5 TComponent Methods

The methods of the TComponent class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

23.1.27.5.1 TComponent._AddRef Method

Called when an application uses a component interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef is a basic implementation of the IInterface method, _AddRef.

Note: If the component is a wrapper for a COM object, __AddRef calls the _AddRef method of that COM object, and returns the resulting reference count.

In all other cases, __AddRef simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __AddRef to implement reference counting.

See Also

Invoke (see page 2781)

QueryInterface (see page 2784)

_Release (see page 2769)

ComObject (see page 2793)

_AddRef (see page 22397)

23.1.27.5.2 TComponent._Release Method

Called when an application releases a component interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release is a basic implementation of the IInterface method, _Release.

__Release returns the resulting value of the reference count for the component's interface.

Note:

In all other cases, __Release simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __Release to implement reference counting.

See Also

Invoke (see page 2781)

QueryInterface (see page 2784)

_AddRef (see page 2769)

ComObject (see page 2793)

23.1.27.5.3 TComponent.BeforeDestruction Method

Performs any necessary actions before the first destructor is called.

Pascal

```
procedure BeforeDestruction; override;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

BeforeDestruction is called automatically immediately before the component's first destructor executes. Do not call it explicitly in your applications.

As implemented in TComponent (see page 2762), BeforeDestruction checks whether the Destroying (see page 2772) method has been called, and if not, calls it. Descendants that override this method to perform other actions before a component is destroyed should call the inherited method first to ensure that this check takes place.

See Also

Destroy (see page 2766)

Destroying (see page 2772)

AfterConstruction (see page 22289)

23.1.27.5.4 TComponent.ChangeName Method

Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.

Pascal

```
procedure ChangeName(const NewName: TComponentName);
```

C++

```
__fastcall ChangeName(const TComponentName NewName);
```

Description

Do not use ChangeName directly in an application. Instead, use the Name (see page 2805) property.

Note: The property setter for Name (see page 2805), SetName (see page 2789), uses ChangeName to change the component's name. ChangeName is not virtual; do not override it.

See Also

Name (see page 2805)

SetName (see page 2789)

ValidateRename (see page 2792)

23.1.27.5.5 TComponent.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent (see page 2762) overrides the DefineProperties method defined in TPersistent (see page 2928) to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor (see page 2839)

DefineBinaryProperty (see page 2837)

DefineProperties (see page 2931)

DefineProperty (see page 2838)

23.1.27.5.6 TComponent.DestroyComponents Method

Destroys all owned components.

Pascal

```
procedure DestroyComponents;
```

C++

```
__fastcall DestroyComponents();
```

Description

DestroyComponents iterates through the components owned by the component, removing each from the list of owned components and destroying it.

It is not necessary to call DestroyComponents directly. DestroyComponents is automatically called when the component is

destroyed.

See Also

Destroy (see page 2766)

Destroying (see page 2772)

Free (see page 22300)

Owner (see page 2807)

23.1.27.5.7 TComponent.Destroying Method

Indicates that the component and its owned components are about to be destroyed.

Pascal

```
procedure Destroying;
```

C++

```
__fastcall Destroying();
```

Description

Destroying sets the csDestroying flag in the ComponentState (see page 2804) property. It then calls the Destroying method for each owned component so that their csDestroying flags are also set. If csDestroying is already set, Destroying does nothing.

It is not necessary to call Destroying directly. Destroying is automatically called when the component is destroyed.

See Also

ComponentState (see page 2804)

Destroy (see page 2766)

DestroyComponents (see page 2771)

Free (see page 22300)

Owner (see page 2807)

23.1.27.5.8 TComponent.ExecuteAction Method

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; dynamic;
```

C++

```
__fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

This is ExecuteAction, a member of class TComponent.

23.1.27.5.9 TComponent.FindComponent Method

Indicates whether a given component is owned by the component.

Pascal

```
function FindComponent(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindComponent(const AnsiString AName);
```

Description

FindComponent returns the component in the Components (see page 2799) property array with the name that matches the string in the AName parameter. Use FindComponent to determine whether a given component is owned by another.

Component name matches are not case sensitive.

C++ Examples:

```
/*
This code resizes the active control to twice as wide and
half as high:
*/
void __fastcall TForm1::ButtonTSClick(TObject *Sender)
{
  TRect MyRect;
  if (myActiveControl == NULL) exit;
  MyRect = myActiveControl->BoundsRect;
  MyRect.Right = MyRect.Left + (MyRect.Right - MyRect.Left) / 2;
  MyRect.Bottom = MyRect.Top + 2 * (MyRect.Bottom - MyRect.Top);
  myActiveControl->BoundsRect = MyRect;
}
void __fastcall TForm1::ButtonSFClick(TObject *Sender)
{
  TRect MyRect;
  if (myActiveControl == NULL) exit;
  MyRect = myActiveControl->BoundsRect;
  MyRect.Right = MyRect.Left + 2 * (MyRect.Right - MyRect.Left);
  MyRect.Bottom = MyRect.Top + (MyRect.Bottom - MyRect.Top) / 2;
  myActiveControl->BoundsRect = MyRect;
}
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TComponent *Temp;
  TListItem *ListItem;
  ListView1->ViewStyle = vsList;
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Components: ";
  for (int I = ComponentCount - 1; I >= 0; I--)
  {
    Temp = Components[I];
    ListItem = ListView1->Items->Add();
    ListItem->Caption = Temp->Name;
  }
}
void __fastcall TForm1::ListView1SelectItem(TObject *Sender, TListItem *Item,
      bool Selected)
{
  TComponent *comp = FindComponent(Item->Caption);
  if (dynamic_cast<TWinControl *>(comp) != NULL)
    myActiveControl = (TWinControl *)comp;
}
/*
The following example creates 20 edit boxes, using
FindComponent with the edit box name to access each newly
created edit box.
*/
const int EditBoxCount = 20;
const int LeftCoordinate = 10;
TEdit* pe[20];
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const char* pszNamePrefix = "MyEdit";
```

```
    for (int i=0;i<EditBoxCount;i++)
    {
        pe[i] = new TEdit(this);
        pe[i]->Name = pszNamePrefix + IntToStr(i+1);
        pe[i]->Left = LeftCoordinate;
        pe[i]->Top = i*EditBoxCount;
        pe[i]->Parent = this;
    }
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<EditBoxCount;i++)
    delete pe[i];
}
```

Delphi Examples:

```
{
This code resizes the active control to twice as wide and
half as high:
}
procedure TForm1.ButtonSFClick(Sender: TObject);
var
  MyRect: TRect;
begin
  if (myActiveControl = nil) then exit;
  MyRect := myActiveControl.BoundsRect;
  MyRect.Right := MyRect.Left + (MyRect.Right - MyRect.Left) * 2;
  MyRect.Bottom := MyRect.Top + (MyRect.Bottom - MyRect.Top) div 2;
  myActiveControl.BoundsRect := MyRect;
end;

procedure TForm1.ButtonTSClick(Sender: TObject);
var
  MyRect: TRect;
begin
  if (myActiveControl = nil) then exit;
  MyRect := myActiveControl.BoundsRect;
  MyRect.Right := MyRect.Left + (MyRect.Right - MyRect.Left) div 2;
  MyRect.Bottom := MyRect.Top + (MyRect.Bottom - MyRect.Top) * 2;
  myActiveControl.BoundsRect := MyRect;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
  Temp: TComponent;
  ListItem: TListItem;
begin
  ListView1.ViewStyle := vsList;
  ListItem := ListView1.Items.Add;
  ListItem.Caption := 'Components: ';
  for I := ComponentCount - 1 downto 0 do
  begin
    Temp := Components[I];
    begin
      ListItem := ListView1.Items.Add;
      ListItem.Caption := Temp.Name;
    end;
  end;
end;

procedure TForm1.ListView1SelectItem(Sender: TObject; Item: TListItem;
  Selected: Boolean);
var comp: TComponent;
begin
```

```
    comp := FindComponent(Item.Caption);
    if comp is TWinControl then
      myActiveControl := TWinControl(comp);
  end;
  {
  The following example creates 20 edit boxes, using
  FindComponent with the edit box name to access each newly
  created edit box.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    i: Integer;
  const
    NamePrefix = 'MyEdit';
  begin
    for i := 1 to 20 do begin
      TEdit.Create(Self).Name := NamePrefix + IntToStr(i);
      with TEdit(FindComponent(NamePrefix + IntToStr(i))) do
      begin
        Left := 10;
        Top := i * 20;
        Parent := self;
      end;
    end;
  end;
```

See Also

Components (see page 2799)

Owner (see page 2807)

Name (see page 2805)

23.1.27.5.10 TComponent.FreeNotification Method

Ensures that AComponent is notified that the component is going to be destroyed.

Pascal

```
procedure FreeNotification(AComponent: TComponent);
```

C++

```
__fastcall FreeNotification(TComponent * AComponent);
```

Description

Use FreeNotification to register AComponent as a component that should be notified when the component is about to be destroyed. It is only necessary to register components this way when they are in a different form or have a different owner. For example, if AComponent is in another form and uses the component to implement a property, it must call FreeNotification so that its Notification (see page 2783) method is called when the component is destroyed.

For components with the same owner, the Notification (see page 2783) method is called automatically when an application explicitly frees the component. This notification is not sent out when components are freed implicitly, because the Owner (see page 2807) is already being freed.

See Also

Notification (see page 2783)

RemoveFreeNotification (see page 2787)

Owner (see page 2807)

23.1.27.5.11 TComponent.FreeOnRelease Method

Frees the interface reference for components that were created from COM classes.

Pascal

```
procedure FreeOnRelease;
```

C++

```
__fastcall FreeOnRelease();
```

Description

FreeOnRelease is called when an interface implemented by the component is released. FreeOnRelease is used internally and calls the corresponding interface method. It should not be necessary to call FreeOnRelease directly.

See Also

ComObject (see page 2793)

23.1.27.5.12 TComponent.GetChildOwner Method

Returns the owner of a child component being read from a stream.

Pascal

```
function GetChildOwner: TComponent; dynamic;
```

C++

```
__fastcall TComponent * GetChildOwner();
```

Description

GetChildOwner is used internally by the component streaming system. It is rarely necessary to call it directly.

In TComponent (see page 2762), GetChildOwner always returns nil (Delphi) or NULL (C++), indicating that the owner is the root component currently being read (usually a form or data module). The Owner (see page 2807) of a component is responsible for destroying it.

See Also

Create (see page 2766)

Owner (see page 2807)

23.1.27.5.13 TComponent.GetChildParent Method

Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.

Pascal

```
function GetChildParent: TComponent; dynamic;
```

C++

```
__fastcall TComponent * GetChildParent();
```

Description

GetChildParent is used internally in the component streaming system. It is not necessary to call it directly.

As implemented in TComponent (see page 2762), GetChildParent returns Self (Delphi) or this (C++). If GetChildParent returns nil (Delphi) or NULL (C++), the parent is assumed to be the root component currently being read (usually a form).

See Also

GetParentComponent (see page 2778)

23.1.27.5.14 TComponent.GetChildren Method

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); dynamic;
```

C++

```
__fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

This is GetChildren, a member of class TComponent.

23.1.27.5.15 TComponent.GetEnumerator Method

Returns a TComponent (see page 2762) enumerator.

Pascal

```
function GetEnumerator: TComponentEnumerator;
```

C++

```
__fastcall TComponentEnumerator GetEnumerator();
```

Description

GetEnumerator returns a TComponentEnumerator (see page 2812) reference, which enumerates the components contained within a specified containing component.

To process all these sub components, call the TComponentEnumerator (see page 2812) GetCurrent method within a While MoveNext do loop.

See Also

TComponentEnumerator (see page 2812)

23.1.27.5.16 TComponent.GetIDsOfNames Method

Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int NameCount, int
LocaleID, void * DispIDs);
```

Description

GetIDsOfNames implements the IDispatch interface GetIDsOfNames method. For components that support interfaces, GetIDsOfNames calls this method for the interface supported by the component, passing the specified parameters. The returned value can be used on subsequent calls to the Invoke (see page 2781) method.

See Also

GetTypeInfo (see page 2779)

GetTypeInfoCount (see page 2779)

Invoke (see page 2781)

23.1.27.5.17 TComponent.GetNamePath Method

Returns a string used by the Object Inspector.

Pascal

```
function GetNamePath: string; override;
```

C++

```
virtual __fastcall AnsiString GetNamePath();
```

Description

GetNamePath is used to determine the text to display in the Object Inspector for the name of the object being edited. GetNamePath is introduced in TPersistent (see page 2928) so descendants such as collections can appear in the Object Inspector. TComponent (see page 2762) overrides GetNamePath to return the component's name. Do not call GetNamePath directly.

See Also

GetOwner (see page 2778)

GetNamePath (see page 2932)

Name (see page 2805)

23.1.27.5.18 TComponent.GetOwner Method

Returns the owner of a component.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is called by GetNamePath (see page 2778) to find the owner of a component. GetNamePath (see page 2778) and GetOwner are introduced in TPersistent (see page 2928) so descendants such as collections can appear in the Object Inspector. For TComponent (see page 2762), GetOwner returns the value of the Owner (see page 2807) property.

See Also

GetNamePath (see page 2778)

Owner (see page 2807)

GetOwner (see page 2932)

23.1.27.5.19 TComponent.GetParentComponent Method

Returns the containing Component.

Pascal

```
function GetParentComponent: TComponent; dynamic;
```

C++

```
__fastcall TComponent * GetParentComponent();
```

Description

GetParentComponent returns a reference to the component that contains the current component, or nil of there is no container. Use HasParentComponent to determine if a component has a containing component.

For example, a button on a panel on a form will return the panel. The panel will return the form. And the form will return nil.

See Also

HasParentComponent

23.1.27.5.20 TComponent.GetTypeInfo Method

Retrieves the type information for an object.

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

GetTypeInfo implements the IDispatch interface GetTypeInfo method. For components that support interfaces, GetTypeInfo calls the GetTypeInfo method for the interface supported by the component, passing it the specified parameters. Use the returned value to get the type information for an interface implemented by the component.

See Also

GetIDsOfNames (see page 2777)

GetTypeInfoCount (see page 2779)

Invoke (see page 2781)

23.1.27.5.21 TComponent.GetTypeInfoCount Method

Returns the number of type information interfaces that an object provides (either 0 or 1).

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

GetTypeInfoCount implements the IDispatch interface GetTypeInfoCount method. For components that support interfaces, GetTypeInfoCount calls this method for the interface supported by the component. The Count parameter points to a location that receives the number of type information interfaces provided by the object. If the object provides type information, this number is 1; otherwise the number is 0.

See Also

GetIDsOfNames (see page 2777)

GetTypeInfo (see page 2779)

Invoke (see page 2781)

23.1.27.5.22 TComponent.HasParent Method

Pascal

```
function HasParent: Boolean; dynamic;
```

C++

```
__fastcall Boolean HasParent();
```

Description

This is HasParent, a member of class TComponent.

23.1.27.5.23 TComponent.InsertComponent Method

Establishes the component as the owner of a specified component.

Pascal

```
procedure InsertComponent(AComponent: TComponent);
```

C++

```
__fastcall InsertComponent(TComponent * AComponent);
```

Description

InsertComponent adds the component passed in the AComponent parameter to the end of the Components (see page 2799) array property. The inserted component must have no name (no specified Name (see page 2805) property value), or the name must be unique among all others in the Components (see page 2799) list.

When the owning component is destroyed, AComponent is destroyed also.

Components (see page 2799) are automatically inserted and removed when visually manipulating them in the form designer. Use InsertComponent when manually adding components to another Owner (see page 2807) component's Components (see page 2799) list.

C++ Examples:

```
/*
This code fragment moves any nonvisual components on the
form into a separate data module. Note that the components
are removed starting with the last component, so that the
unprocessed portion of the Components array does not change.
Note: This code does not save the form or data module to disk
after the nonvisual components are moved.  If executed at
runtime, the effect will not persist.
Add a second TForm and name it ComponentCount2.  Put
"#include ComponentCount2.cpp" in ComponentCount.h.
Place a TMemo in each form and a TButton in Form1.
*/

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int I;
  TComponent *Temp;
  Memo1->Lines->Add("Components removed: ");
  Form2->Memo1->Lines->Add("Components added: ");
  for (I = ComponentCount - 1; I >= 0; I--)
  {
    Temp = Components[I];
    // only move components that are not controls
```

```
      if (dynamic_cast<TControl *>(Temp) == NULL)
      {
        RemoveComponent(Temp);
        Memo1->Lines->Add(Temp->Name);
        Form2->InsertComponent(Temp);
        Form2->Memo1->Lines->Add(Temp->Name);
      }
    }
  }
```

Delphi Examples:

```
{
This code fragment moves any nonvisual components on the
form into a separate data module. Note that the components
are removed starting with the last component, so that the
unprocessed portion of the Components array does not change.
Note: This code does not save the form or data module to disk
after the nonvisual components are moved.  If executed at
runtime, the effect will not persist.
Add a second TForm and name it ComponentCount2.  Put
"ComponentCount2" at the end of the uses clause of Form1.
Place a TMemo in each form and a TButton in Form1.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
  Temp: TComponent;
begin
  Form1.Memo1.Lines.Add('Components removed: ');
  Form2.Memo1.Lines.Add('Components added: ');
  for I := ComponentCount - 1 downto 0 do
  begin
    Temp := Components[I];
    if not (Temp is TControl) then
    begin
      RemoveComponent(Temp);
      Form1.Memo1.Lines.Add(Temp.Name);
      Form2.InsertComponent(Temp);
      Form2.Memo1.Lines.Add(Temp.Name);
    end;
  end;
end;
```

See Also

Components (see page 2799)

Owner (see page 2807)

23.1.27.5.24 TComponent.Invoke Method

Provides access to Automation properties and methods when the component wraps an Automation object.

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word Flags,
Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

Invoke is the standard mechanism for accessing the exposed properties and methods of an Automation object. For components that wrap the IDispatch interface of an Automation object, Invoke calls the Invoke method for the interface supported by the component, passing it the parameters specified by the function.

See Also

GetIDsOfNames (see page 2777)

GetTypeInfo (see page 2779)

GetTypeInfoCount (see page 2779)

ComObject (see page 2793)

23.1.27.5.25 TComponent.IsImplementorOf Method

Indicates whether the component implements a specified interface.

Pascal

```
function IsImplementorOf(const I: IInterface): Boolean;
```

C++

```
__fastcall Boolean IsImplementorOf(const IInterface I);
```

Description

Call IsImplementorOf to determine whether the component (or, if the component aggregates its interface with other components, whether the controlling component) supports the interface specified by I. IsImplementorOf is similar to the QueryInterface (see page 2784) method, but it can handle a request for a nil (Delphi) or NULL (C++) interface, and it does not return an interface pointer.

The streaming system that loads and saves components uses IsImplementorOf to resolve property values that are interfaces.

See Also

QueryInterface (see page 2784)

23.1.27.5.26 TComponent.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; virtual;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling

pointers can be used after being streamed in.

As implemented in TComponent (see page 2762), Loaded clears the csLoading flag in the ComponentState (see page 2804) property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState (see page 2804)

Create (see page 2766)

TReader (see page 2933)

23.1.27.5.27 TComponent.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); virtual;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner (see page 2807) is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

23.1.27.5.28 TComponent.PaletteCreated Method

Responds when the component is created from the component palette.

Pascal

```
procedure PaletteCreated; dynamic;
```

C++

```
__fastcall PaletteCreated();
```

Description

PaletteCreated is called automatically at design time when the component has just been created from the component palette. Component writers can override this method to perform adjustments that are required only when the component is created from the component palette.

As implemented in TComponent (see page 2762), PaletteCreated does nothing.

See Also

Create (see page 2766)

Loaded (see page 2782)

23.1.27.5.29 TComponent.QueryInterface Method

Returns a reference to a specified interface if the object supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the component supports the interface specified by IID, and if so, returns a reference to that interface as the Obj parameter. If the component does not support the interface, the Obj parameter returns nil (Delphi) or NULL (C++).

Note: For components that act as COM object wrapper, QueryInterface calls the QueryInterface method of the internal COM object.

See Also

_AddRef (see page 2769)

_Release (see page 2769)

QueryInterface

QueryInterface (see page 22398)

Invoke (see page 2781)

ComObject (see page 2793)

23.1.27.5.30 TComponent.ReadState Method

Pascal

```
procedure ReadState(Reader: TReader); virtual;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

This is ReadState, a member of class TComponent.

23.1.27.5.31 TComponent.ReferenceInterface Method

Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.

Pascal

```
function ReferenceInterface(const I: IInterface; Operation: TOperation): Boolean;
```

C++

```
__fastcall Boolean ReferenceInterface(const IInterface I, TOperation Operation);
```

Description

Component writers use ReferenceInterface to ensure that properties whose values are interfaces are informed when the objects that implement those interfaces are destroyed. This notification must be in place for a property whose value is an interface to be saved with the component in a form file (that is, for such a property to persist as a published property).

I is an interface pointer that is the value of the published property of interest.

Operation indicates whether the notification link to the implementer of the interface should be established (opInsert) or removed (opRemove).

ReferenceInterface returns true if it is successful in establishing or removing the notification link. If ReferenceInterface returns false when called with Operation set to opInsert, the specified interface can't be stored as the value of a published property.

Note: A result of false does not necessarily indicate an error, merely that the interface can't be stored by the property streaming system. For example, ReferenceInterface returns false when the specified interface employs true reference counting, independent of component lifetimes.

See Also

FreeNotification (see page 2775)

RemoveFreeNotification (see page 2787)

23.1.27.5.32 TComponent.RemoveComponent Method

Removes a specified component specified from the component's Components (see page 2799) list.

Pascal

```
procedure RemoveComponent(AComponent: TComponent);
```

C++

```
__fastcall RemoveComponent(TComponent * AComponent);
```

Description

Components (see page 2799) are automatically inserted and removed when visually manipulating them in the form designer. Use RemoveComponent to programmatically delete the component specified by AComponent from its Owner (see page 2807) component.

C++ Examples:

```
/*
This code fragment moves any nonvisual components on the
form into a separate data module. Note that the components
are removed starting with the last component, so that the
unprocessed portion of the Components array does not change.
Note: This code does not save the form or data module to disk
after the nonvisual components are moved.  If executed at
```

```
runtime, the effect will not persist.
Add a second TForm and name it ComponentCount2.  Put
"#include ComponentCount2.cpp" in ComponentCount.h.
Place a TMemo in each form and a TButton in Form1.
*/

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int I;
  TComponent *Temp;
  Memo1->Lines->Add("Components removed: ");
  Form2->Memo1->Lines->Add("Components added: ");
  for (I = ComponentCount - 1; I >= 0; I--)
  {
    Temp = Components[I];
    // only move components that are not controls
    if (dynamic_cast<TControl *>(Temp) == NULL)
    {
      RemoveComponent(Temp);
      Memo1->Lines->Add(Temp->Name);
      Form2->InsertComponent(Temp);
      Form2->Memo1->Lines->Add(Temp->Name);
    }
  }
}
```

Delphi Examples:

```
{
This code fragment moves any nonvisual components on the
form into a separate data module. Note that the components
are removed starting with the last component, so that the
unprocessed portion of the Components array does not change.
Note: This code does not save the form or data module to disk
after the nonvisual components are moved.  If executed at
runtime, the effect will not persist.
Add a second TForm and name it ComponentCount2.  Put
"ComponentCount2" at the end of the uses clause of Form1.
Place a TMemo in each form and a TButton in Form1.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
  Temp: TComponent;
begin
  Form1.Memo1.Lines.Add('Components removed: ');
  Form2.Memo1.Lines.Add('Components added: ');
  for I := ComponentCount - 1 downto 0 do
  begin
    Temp := Components[I];
    if not (Temp is TControl) then
    begin
      RemoveComponent(Temp);
      Form1.Memo1.Lines.Add(Temp.Name);
      Form2.InsertComponent(Temp);
      Form2.Memo1.Lines.Add(Temp.Name);
    end;
  end;
end;
```

See Also

Components (see page 2799)

Notification (see page 2783)

Owner (see page 2807)

23.1.27.5.33 TComponent.RemoveFreeNotification Method

Disables destruction notification that was enabled by FreeNotification (see page 2775)

Pascal

```
procedure RemoveFreeNotification(AComponent: TComponent);
```

C++

```
__fastcall RemoveFreeNotification(TComponent * AComponent);
```

Description

RemoveFreeNotification removes the component specified by the AComponent parameter from the internal list of objects to be notified that the component is about to be destroyed. AComponent is added to this list by a previous call to the FreeNotification (see page 2775) method.

Most applications have no need to call RemoveFreeNotification. It is used by TComponent (see page 2762) to detect loops where two components are notifying each other of their impending destruction.

See Also

FreeNotification (see page 2775)

23.1.27.5.34 TComponent.SafeCallException Method

Handles exceptions in methods declared using the safecall calling convention.

Pascal

```
function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): HResult; override;
```

C++

```
virtual __fastcall HRESULT SafeCallException(TObject * ExceptObject, void * ExceptAddr);
```

Description

SafeCallException handles exceptions in methods that use the safecall calling convention. Some classes that implement interfaces override this method to handle errors that might occur. TComponent (see page 2762) calls the implementation of this method for the interface supported by the component, if it exists. If the component does not support interfaces, this method calls the SafeCallException method inherited from TObject, which returns E_UNEXPECTED. This is a default return value that is appropriate for classes that do not support any interfaces.

See Also

SafeCallException

23.1.27.5.35 TComponent.SetAncestor Method

Pascal

```
procedure SetAncestor(Value: Boolean);
```

C++

```
__fastcall SetAncestor(Boolean Value);
```

Description

This is SetAncestor, a member of class TComponent.

23.1.27.5.36 TComponent.SetChildOrder Method

Pascal

```
procedure SetChildOrder(Child: TComponent; Order: Integer); dynamic;
```

C++

```
__fastcall SetChildOrder(TComponent * Child, int Order);
```

Description

This is SetChildOrder, a member of class TComponent.

23.1.27.5.37 TComponent.SetDesigning Method

Ensures that components inserted at design time have their design-mode flag set.

Pascal

```
procedure SetDesigning(Value: Boolean; SetChildren: Boolean = True);
```

C++

```
__fastcall SetDesigning(Boolean Value, Boolean SetChildren = True);
```

Description

SetDesigning is used internally by the Form designer. Do not call SetDesigning directly.

SetDesigning sets the csDesigning flag in the ComponentState (see page 2804) property if Value is true; otherwise, it removes csDesigning.

If the SetChildren parameter is true, SetDesigning then calls the SetDesigning methods of any owned components, passing Value, so that the owned components' ComponentState (see page 2804) properties are synchronized with the owner's.

The InsertComponent (see page 2780) and RemoveComponent (see page 2785) methods call SetDesigning for inserted or removed components to ensure that their design-mode flags are set properly.

See Also

ComponentState (see page 2804)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

23.1.27.5.38 TComponent.SetDesignInstance Method

Ensures that components inserted at design time have their design-mode flag set.

Pascal

```
procedure SetDesignInstance(Value: Boolean);
```

C++

```
__fastcall SetDesignInstance(Boolean Value);
```

Description

SetDesignInstance is used internally by the Form designer to identify objects that act as a design surface. Do not call SetDesignInstance directly.

SetDesignInstance sets the csDesignInstance flag in the ComponentState (see page 2804) property if Value is true; otherwise,

it removes csDesignInstance.

See Also

ComponentState (see page 2804)

SetDesigning (see page 2788)

23.1.27.5.39 TComponent.SetInline Method

Sets the csInline bit of the component's ComponentState (see page 2804) property

Pascal

```
procedure SetInline(Value: Boolean);
```

C++

```
__fastcall SetInline(Boolean Value);
```

Description

SetInline is used internally to indicate whether the component can act as a root component in the designer but also be embedded in a form.

SetInline sets the csInline flag in the ComponentState (see page 2804) property if Value is true; otherwise, it removes csInline.

See Also

ComponentState (see page 2804)

23.1.27.5.40 TComponent.SetName Method

Sets the value of the Name (see page 2805) property.

Pascal

```
procedure SetName(const NewName: TComponentName); virtual;
```

C++

```
virtual __fastcall SetName(const TComponentName NewName);
```

Description

SetName is the virtual property setter for the Name (see page 2805) property. It calls the ChangeName (see page 2770) method, which performs the actual name change.

ChangeName (see page 2770) is not virtual; override SetName instead when modifying the behavior of the Name (see page 2805) property.

Warning: Using SetName to change the name of a component at runtime causes any references to the old name to become undefined. Any subsequent code that uses the old name will cause an exception.

See Also

ChangeName (see page 2770)

Name (see page 2805)

ValidateRename (see page 2792)

23.1.27.5.41 TComponent.SetParentComponent Method

Pascal

```
procedure SetParentComponent(Value: TComponent); dynamic;
```

C++

```
__fastcall SetParentComponent(TComponent * Value);
```

Description

This is SetParentComponent, a member of class TComponent.

23.1.27.5.42 TComponent.SetSubComponent Method

Identifies whether the component is a subcomponent.

Pascal

```
procedure SetSubComponent(IsSubComponent: Boolean);
```

C++

```
__fastcall SetSubComponent(Boolean IsSubComponent);
```

Description

Call SetSubComponent to indicate that this component is or is not a subcomponent. A subcomponent is a component whose Owner (see page 2807) is a component other than the form or data module in which it resides. Unless such a component calls SetSubComponent with IsSubComponent set to true, its published properties will not be saved to the form file.

IsSubComponent indicates whether the component is a subcomponent (true) or not (false).

SetSubComponent is called at design time either

From the constructor of a component that always acts as a subcomponent. In this case, the component calls its own SetSubComponent method from the constructor with IsSubComponent set to true.

Immediately after constructing an instance of the subcomponent. In this case, the Owner (see page 2807) calls the SetSubComponent method of a component it has just instantiated, with IsSubComponent set to true.

See Also

ComponentStyle (see page 2805)

Owner (see page 2807)

23.1.27.5.43 TComponent.UpdateAction Method

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; dynamic;
```

C++

```
__fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

This is UpdateAction, a member of class TComponent.

23.1.27.5.44 TComponent.Updated Method

Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

Pascal

```
procedure Updated; dynamic;
```

C++

```
__fastcall Updated();
```

Description

Do not call Updated directly. It is used internally to clear the csUpdating flag of the ComponentState (see page 2804) property. A call to Updated always follows a call to Updating (see page 2791), which sets the flag.

See Also

ComponentState (see page 2804)

Updating (see page 2791)

23.1.27.5.45 TComponent.UpdateRegistry Method

Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

Pascal

```
class procedure UpdateRegistry(Register: Boolean; const ClassID: string; const ProgID:
string); virtual;
```

C++

```
virtual __fastcall UpdateRegistry(Boolean Register, const AnsiString ClassID, const AnsiString
ProgID);
```

Description

Do not call UpdateRegistry directly. It is for internal use only.

23.1.27.5.46 TComponent.Updating Method

Sets the csUpdating state in the component's ComponentState (see page 2804) property.

Pascal

```
procedure Updating; dynamic;
```

C++

```
__fastcall Updating();
```

Description

Do not call Updating directly. It is used internally to indicate that the component is about to be updated. A call to Updating, which sets the csUpdating flag, is always followed by a call to Updated (see page 2791), which clears the flag.

See Also

ComponentState (see page 2804)

Updated (see page 2791)

23.1.27.5.47 TComponent.ValidateContainer Method

Determines whether an object can be inserted into a container.

Pascal

```
procedure ValidateContainer(AComponent: TComponent); dynamic;
```

C++

```
__fastcall ValidateContainer(TComponent * AComponent);
```

Description

ValidateContainer is called by a component when it is about to be inserted into a container object. By default, ValidateContainer calls the ValidateInsert (see page 2792) method of the component specified by the AComponent parameter.

Descendant components can override ValidateContainer to disallow a component from being inserted into specific containers. To disallow an insertion, raise an exception in the derived method.

See Also

ValidateInsert (see page 2792)

23.1.27.5.48 TComponent.ValidateInsert Method

Provides the interface for a method that validates an child component before it is inserted.

Pascal

```
procedure ValidateInsert(AComponent: TComponent); dynamic;
```

C++

```
__fastcall ValidateInsert(TComponent * AComponent);
```

Description

ValidateInsert does nothing in TComponent (see page 2762). Descendant classes can override it to disallow a component from accepting an object as a child. By default ValidateInsert allows any object to be inserted into the component.

If a component needs to validate only certain objects, descendant classes can override ValidateInsert to filter out those objects. To disallow an insertion, raise an exception in the derived method.

See Also

ValidateContainer (see page 2792)

23.1.27.5.49 TComponent.ValidateRename Method

Ensures that renaming an owned component does not create name conflict.

Pascal

```
procedure ValidateRename(AComponent: TComponent; const CurName: string; const NewName:
string); virtual;
```

C++

```
virtual __fastcall ValidateRename(TComponent * AComponent, const AnsiString CurName, const
AnsiString NewName);
```

Description

ValidateRename checks to see if a component can rename one of its owned components, passed in AComponent, from its

current name (CurName) to the string passed in NewName. If AComponent is nil (Delphi) or NULL (C++) or NewName is already the name of a component in the Components (see page 2799) list, ValidateRename raises an EComponentError (see page 2694) exception.

ValidateRename is used internally when the Name (see page 2805) property is modified. It is not necessary to call it directly.

See Also

ChangeName (see page 2770)

Name (see page 2805)

SetName (see page 2789)

23.1.27.5.50 TComponent.WriteState Method

Pascal

```
procedure WriteState(Writer: TWriter); virtual;
```

C++

```
virtual __fastcall WriteState(TWriter Writer);
```

Description

This is WriteState, a member of class TComponent.

23.1.27.6 TComponent Properties

The properties of the TComponent class are listed here.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

23.1.27.6.1 TComponent.ComObject Property

Specifies the interface reference implemented by the component.

Pascal

```
property ComObject: IUnknown;
```

C++

```
__property IUnknown ComObject;
```

Description

Use ComObject to assign a COM interface implemented by a component to an interface reference. This property is used by components that support COM interfaces.

If the component is not a wrapper for a COM component, trying to read ComObject causes TComponent (see page 2762) to raise an EComponentError (see page 2694) exception.

See Also

EComponentError (see page 2694)

23.1.27.6.2 TComponent.ComponentCount Property

Indicates the number of components owned by the component.

Pascal

```
property ComponentCount: Integer;
```

C++

```
__property int ComponentCount;
```

Description

Use the ComponentCount property to determine the number of components owned by a component, for example, when iterating through the components list to perform some action on all owned components. The ComponentCount property equals the number of items in the components list. This value is one more than the highest Components (see page 2799) index, because the first components index is 0.

C++ Examples:

```
/*
This code fragment moves any nonvisual components on the
form into a separate data module. Note that the components
are removed starting with the last component, so that the
unprocessed portion of the Components array does not change.
Note: This code does not save the form or data module to disk
after the nonvisual components are moved.  If executed at
runtime, the effect will not persist.
Add a second TForm and name it ComponentCount2.  Put
"#include ComponentCount2.cpp" in ComponentCount.h.
Place a TMemo in each form and a TButton in Form1.
*/

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int I;
  TComponent *Temp;
  Memo1->Lines->Add("Components removed: ");
  Form2->Memo1->Lines->Add("Components added: ");
  for (I = ComponentCount - 1; I >= 0; I--)
  {
    Temp = Components[I];
    // only move components that are not controls
    if (dynamic_cast<TControl *>(Temp) == NULL)
    {
      RemoveComponent(Temp);
      Memo1->Lines->Add(Temp->Name);
      Form2->InsertComponent(Temp);
      Form2->Memo1->Lines->Add(Temp->Name);
```

```
    }
  }
}
```

Delphi Examples:

```
{
This code fragment moves any nonvisual components on the
form into a separate data module. Note that the components
are removed starting with the last component, so that the
unprocessed portion of the Components array does not change.
Note: This code does not save the form or data module to disk
after the nonvisual components are moved.  If executed at
runtime, the effect will not persist.
Add a second TForm and name it ComponentCount2.  Put
"ComponentCount2" at the end of the uses clause of Form1.
Place a TMemo in each form and a TButton in Form1.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
  Temp: TComponent;
begin
  Form1.Memo1.Lines.Add('Components removed: ');
  Form2.Memo1.Lines.Add('Components added: ');
  for I := ComponentCount - 1 downto 0 do
  begin
    Temp := Components[I];
    if not (Temp is TControl) then
    begin
      RemoveComponent(Temp);
      Form1.Memo1.Lines.Add(Temp.Name);
      Form2.InsertComponent(Temp);
      Form2.Memo1.Lines.Add(Temp.Name);
    end;
  end;
end;
{
Here is the code on Form1 that fills the list box.
}
var
  Form1: TForm1;
  Customers: TTable;
  Blobs: TTable;
  Form1Done : Boolean;
  procedure CreateMyTable;
  procedure CreateCustSource;
  procedure CreateBlobSource;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  CreateCustSource;
  CreateBlobSource;
  for I := 0 to ComponentCount - 1 do
  begin
    if (Components[I] is TDataSource) then
      ListBox1.Items.AddObject(Components[I].Name,
        Components[I] as TObject);
  end;
end;
```

```
procedure CreateCustSource();
begin
  Customers:= TTable.Create(Form1);
  with Customers do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'MyCustInfo';
    ReadOnly:= False;

//    Don't overwrite an existing table
    if (not Customers.Exists) then CreateMyTable
    else
    begin
      if (Customers.Exists AND
        (MessageDlg('MyCustInfo table already exists.  Do you want to rebuild it?',
mtConfirmation, [mbYes, mbNo], 0) = mrYes)) then
      begin
        Customers.Close;
        Customers.DeleteTable;
        CreateMyTable;
      end;
    end;
  end;
  Form1.CustomerDS.DataSet:= Customers;
  Customers.Active:= True;
end;

procedure CreateMyTable();
var
  i: Integer;
begin
//      The Table component must not be active
//      First, describe the type of table and give
//      it a name
//      Next, describe the fields in the table
  with Customers do
  begin
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
        with AddFieldDef do
        begin
          Name := 'Field3';
          DataType := ftString;
          Size := 30;
        end;
        with AddFieldDef do
        begin
          Name := 'Field4';
          DataType := ftString;
          Size := 30;
        end;
```

```
      end;

//    Take out this part to remove indexing
//    Next, describe any indexes
      with IndexDefs do
      begin
        Clear;
//        The 1st index has no name because it is
//        a Paradox primary key
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
      end;

//      Call the CreateTable method to create the table
      CreateTable;
      Customers.Active:= True;
      for i := 1 to 20 do
        Customers.AppendRecord([i*111, i*222, i*333, i*444]);
    end;
end;

procedure CreateBlobSource;
var
  i: Integer;
begin
  Blobs:= TTable.Create(Form1);
  with Blobs do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'MyBlobInfo';
    Blobs.Active := False;
    { Don't overwrite an existing table }
//    if Blobs.Exists then
//      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
//    else
    begin
      if (Blobs.Exists) then
      begin
        Blobs.Close;
        Blobs.DeleteTable;
      end;
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
```

```
          begin
            Name := 'Field2';
            DataType := ftBlob;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
{
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
}
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Blobs.Active:= True;
        for i := 1 to 20 do
          Blobs.AppendRecord([i*111, IntToStr(i*222)]);
      end;
    end;
    Form1.BlobDS.DataSet:= Blobs;
    Blobs.Active:= True;
  end;

  procedure TForm1.ListBox1MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
  begin
    Form1Done := True;
  end;

  initialization

  Form1Done := False;
  {
  The following code sets the DataSource for a DBGrid
  component in Form2 to the currently selected data source in
  a list provided in Form1. This allows users to specify the
  desired information in Form1, and then move to Form2 to view
  the information. Select an item in the listbox and then
  activate Form2 by clicking in it.
  }
  procedure TForm2.FormActivate(Sender: TObject);
  begin
    if (Form1Done) then
    begin
      DBGrid2.DataSource :=
        Form1.ListBox1.Items.Objects[Form1.ListBox1.ItemIndex] as TDataSource;
      DBNavigator2.DataSource :=
        Form1.ListBox1.Items.Objects[Form1.ListBox1.ItemIndex] as TDataSource;
    end;
  end;
```

See Also

ComponentIndex (see page 2799)

Components (see page 2799)

23.1.27.6.3 TComponent.ComponentIndex Property

Indicates the position of the component in its owner's Components (see page 2799) property array.

Pascal

```
property ComponentIndex: Integer;
```

C++

```
__property int ComponentIndex;
```

Description

Use ComponentIndex when iterating through the Components (see page 2799) list of the component's owner to perform some action on owned components. It can be used in conjunction with ComponentCount (see page 2794). ComponentIndex is used internally for iterative assignment procedures.

Note: The first component in the list has a ComponentIndex value of 0, the second has a value of 1, and so on. Therefore, when using ComponentIndex with ComponentCount (see page 2794), note that ComponentCount (see page 2794) is always 1 more than the highest Components (see page 2799) index.

See Also

ComponentCount (see page 2794)

Components (see page 2799)

23.1.27.6.4 TComponent.Components Property

Lists all components owned by the component.

Pascal

```
property Components [Index: Integer]: TComponent;
```

C++

```
__property TComponent * Components[int Index];
```

Description

Use Components to access any of the components owned by this component, such as the components owned by a form. The Components property is most useful when referring to owned components by number rather than name. It is also used internally for iterative processing of all owned components.

Index ranges from 0 to ComponentIndex (see page 2799) minus 1.

C++ Examples:

```
/*
This code fragment moves any nonvisual components on the
form into a separate data module. Note that the components
are removed starting with the last component, so that the
unprocessed portion of the Components array does not change.
Note: This code does not save the form or data module to disk
after the nonvisual components are moved.  If executed at
runtime, the effect will not persist.
Add a second TForm and name it ComponentCount2.  Put
```

```
"#include ComponentCount2.cpp" in ComponentCount.h.
Place a TMemo in each form and a TButton in Form1.
*/

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int I;
  TComponent *Temp;
  Memo1->Lines->Add("Components removed: ");
  Form2->Memo1->Lines->Add("Components added: ");
  for (I = ComponentCount - 1; I >= 0; I--)
  {
    Temp = Components[I];
    // only move components that are not controls
    if (dynamic_cast<TControl *>(Temp) == NULL)
    {
      RemoveComponent(Temp);
      Memo1->Lines->Add(Temp->Name);
      Form2->InsertComponent(Temp);
      Form2->Memo1->Lines->Add(Temp->Name);
    }
  }
}
```

Delphi Examples:

```
{
This code fragment moves any nonvisual components on the
form into a separate data module. Note that the components
are removed starting with the last component, so that the
unprocessed portion of the Components array does not change.
Note: This code does not save the form or data module to disk
after the nonvisual components are moved.  If executed at
runtime, the effect will not persist.
Add a second TForm and name it ComponentCount2.  Put
"ComponentCount2" at the end of the uses clause of Form1.
Place a TMemo in each form and a TButton in Form1.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
  Temp: TComponent;
begin
  Form1.Memo1.Lines.Add('Components removed: ');
  Form2.Memo1.Lines.Add('Components added: ');
  for I := ComponentCount - 1 downto 0 do
  begin
    Temp := Components[I];
    if not (Temp is TControl) then
    begin
      RemoveComponent(Temp);
      Form1.Memo1.Lines.Add(Temp.Name);
      Form2.InsertComponent(Temp);
      Form2.Memo1.Lines.Add(Temp.Name);
    end;
  end;
end;
{
Here is the code on Form1 that fills the list box.
}
var
  Form1: TForm1;
  Customers: TTable;
  Blobs: TTable;
  Form1Done : Boolean;
  procedure CreateMyTable;
```

```
    procedure CreateCustSource;
    procedure CreateBlobSource;

  implementation

  {$R *.dfm}

  procedure TForm1.FormCreate(Sender: TObject);
  var
    I: Integer;
  begin
    CreateCustSource;
    CreateBlobSource;
    for I := 0 to ComponentCount - 1 do
    begin
      if (Components[I] is TDataSource) then
        ListBox1.Items.AddObject(Components[I].Name,
          Components[I] as TObject);
    end;
  end;

  procedure CreateCustSource();
  begin
    Customers:= TTable.Create(Form1);
    with Customers do
    begin
      DatabaseName := 'DBDEMOS';
      TableType := ttParadox;
      TableName := 'MyCustInfo';
      ReadOnly:= False;

  //    Don't overwrite an existing table
      if (not Customers.Exists) then CreateMyTable
      else
      begin
        if (Customers.Exists AND
          (MessageDlg('MyCustInfo table already exists.  Do you want to rebuild it?',
  mtConfirmation, [mbYes, mbNo], 0) = mrYes)) then
        begin
          Customers.Close;
          Customers.DeleteTable;
          CreateMyTable;
        end;
      end;
    end;
    Form1.CustomerDS.DataSet:= Customers;
    Customers.Active:= True;
  end;

  procedure CreateMyTable();
  var
    i: Integer;
  begin
  //      The Table component must not be active
  //      First, describe the type of table and give
  //      it a name
  //      Next, describe the fields in the table
    with Customers do
    begin
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
```

```
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
        with AddFieldDef do
        begin
          Name := 'Field3';
          DataType := ftString;
          Size := 30;
        end;
        with AddFieldDef do
        begin
          Name := 'Field4';
          DataType := ftString;
          Size := 30;
        end;
      end;

//    Take out this part to remove indexing
//    Next, describe any indexes
      with IndexDefs do
      begin
        Clear;
//        The 1st index has no name because it is
//        a Paradox primary key
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
      end;

//      Call the CreateTable method to create the table
      CreateTable;
      Customers.Active:= True;
      for i := 1 to 20 do
        Customers.AppendRecord([i*111, i*222, i*333, i*444]);
    end;
end;

procedure CreateBlobSource;
var
  i: Integer;
begin
  Blobs:= TTable.Create(Form1);
  with Blobs do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'MyBlobInfo';
    Blobs.Active := False;
    { Don't overwrite an existing table }
//    if Blobs.Exists then
//      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
//    else
```

```
    begin
      if (Blobs.Exists) then
      begin
        Blobs.Close;
        Blobs.DeleteTable;
      end;
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftBlob;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
{
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
}
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
      Blobs.Active:= True;
      for i := 1 to 20 do
        Blobs.AppendRecord([i*111, IntToStr(i*222)]);
    end;
  end;
  Form1.BlobDS.DataSet:= Blobs;
  Blobs.Active:= True;
end;

procedure TForm1.ListBox1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  Form1Done := True;
end;

initialization
```

```
Form1Done := False;
{
The following code sets the DataSource for a DBGrid
component in Form2 to the currently selected data source in
a list provided in Form1. This allows users to specify the
desired information in Form1, and then move to Form2 to view
the information. Select an item in the listbox and then
activate Form2 by clicking in it.
}
procedure TForm2.FormActivate(Sender: TObject);
begin
  if (Form1Done) then
  begin
    DBGrid2.DataSource :=
      Form1.ListBox1.Items.Objects[Form1.ListBox1.ItemIndex] as TDataSource;
    DBNavigator2.DataSource :=
      Form1.ListBox1.Items.Objects[Form1.ListBox1.ItemIndex] as TDataSource;
  end;
end;
```

See Also

ComponentCount (see page 2794)

ComponentIndex (see page 2799)

Owner (see page 2807)

23.1.27.6.5 TComponent.ComponentState Property

Describes the current state of the component, indicating when a component needs to avoid certain actions.

Pascal

```
property ComponentState: TComponentState;
```

C++

```
__property TComponentState ComponentState;
```

Description

Components (see page 2799) use the ComponentState property to detect states in which certain kinds of actions are allowed or disallowed. For example, if a component needs to avoid certain behaviors at design time that it performs at runtime, it can check for the csDesigning flag. ComponentState is read-only and its flags are set automatically when appropriate.

See Also

Destroying (see page 2772)

Loaded (see page 2782)

ReadState (see page 2784)

SetDesigning (see page 2788)

SetAncestor (see page 2787)

Updated (see page 2791)

Updating (see page 2791)

WriteState (see page 2793)

23.1.27.6.6 TComponent.ComponentStyle Property

Governs the behavior of the component.

Pascal

```
property ComponentStyle: TComponentStyle;
```

C++

```
__property TComponentStyle ComponentStyle;
```

Description

ComponentStyle governs how the component interacts with the streaming system and the Object Inspector. ComponentStyle is a read-only property. Typically, the value of the various component style flags are part of a component's definition, specified in a component's constructor. The one exception to this is the csSubComponent style, which can be set by calling the SetSubComponent (see page 2790) method.

See Also

SetSubComponent (see page 2790)

ComponentState (see page 2804)

23.1.27.6.7 TComponent.DesignInfo Property

Contains information used by the Form designer.

Pascal

```
property DesignInfo: Longint;
```

C++

```
__property Longint DesignInfo;
```

Description

DesignInfo is used internally. Do not use this property in applications.

See Also

IDesigner (see page 11149)

23.1.27.6.8 TComponent.Name Property

Specifies the name of the component as referenced in code.

Pascal

```
property Name: TComponentName;
```

C++

```
__property TComponentName Name;
```

Description

Use Name to change the name of a component to reflect its purpose in the current application. By default, the IDE assigns sequential names based on the type of the component, such as 'Button1', 'Button2', and so on.

Use Name to refer to the component in code.

Warning: Changing Name at runtime causes any references to the old name to become undefined. Any subsequent code that

uses the old name will cause an exception.

C++ Examples:

```
/*
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < ComponentCount; i++)
    ListBox1->Items->InsertObject(0,
                                  Components[i]->Name,
                                  (TObject *)Components[i]);
}

void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  if (Button == mbRight)
  {
    TClass ClassRef;
    int Index = ListBox1->ItemAtPos(Point(X,Y), true);
    // only components that are controls have a position
    // make sure the component is a control
    for (ClassRef = ListBox1->Items->Objects[Index]->ClassType();
         ClassRef != NULL;
         ClassRef = ClassRef->ClassParent())
      if (String(ClassRef->ClassName()) == "TControl")
      {
        TControl *TheObject = (TControl *)ListBox1->Items->Objects[Index];
        StatusBar1->SimpleText =
          TheObject->Name + " is at (" +
          IntToStr(TheObject->Left) + ", " +
          IntToStr(TheObject->Top) + ")";
         break;
      }
    if (ClassRef == NULL) // if it wasn't a control
      MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
```

```
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to Form1.ComponentCount-1 do
    ListBox1.Items.InsertObject(
      0, Form1.Components[I].Name,
      Form1.Components[I] as TObject);
end;

procedure TForm1.ListBox1MouseUp(
  Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X,Y: integer);
var
  APoint: TPoint;
  Index: integer;
  TheObject: TControl;
begin
  if Button = mbRight then
  begin
  APoint.X := X;
  APoint.Y := Y;
  Index := ListBox1.ItemAtPos(APoint, True);
  if (ListBox1.Items.Objects[Index] is TControl) then
  begin
    TheObject :=
      (ListBox1.Items.Objects[Index] as TControl);
    StatusBar1.SimpleText := TheObject.Name + ' is at (' +
                             IntToStr(TheObject.Left) + ', ' +
                             IntToStr(TheObject.Top) + ') ';
  end
  else
    Beep;
  end;
end;
```

See Also

ChangeName (see page 2770)

SetName (see page 2789)

ValidateRename (see page 2792)

Defining New Persistent Fields

Using TSQLQuery

Using TSQLStoredProc

Using TSQLTable

23.1.27.6.9 TComponent.Owner Property

Indicates the component that is responsible for streaming and freeing this component.

Pascal

```
property Owner: TComponent;
```

C++

```
__property TComponent * Owner;
```

Description

Use Owner to find the owner of a component. The Owner of a component is responsible for two things:

The memory for the owned component is freed when its owner's memory is freed. This means that when a form is destroyed, all the components on the form are also destroyed.

The Owner is responsible for loading and saving the published properties of its owned controls.

By default, a form owns all components that are on it. In turn, the form is owned by the application. Thus when the application shuts down and its memory is freed, the memory for all forms (and all their owned components) is also freed. When a form is loaded into memory, it loads all of the components that are on it.

The owner of a component is determined by the parameter passed to the constructor when the component is created. For components created in the form designer, the form is automatically assigned as the Owner.

Warning: If a component has an Owner other than a form or data module, it will not be saved or loaded with its Owner unless you identify it as a subcomponent. To identify a component as a subcomponent, call the SetSubComponent (see page 2790) method.

See Also

Components (see page 2799)

Create (see page 2766)

Destroy (see page 2766)

Free (see page 22300)

SetSubComponent (see page 2790)

Specifying a Provider

23.1.27.6.10 TComponent.Tag Property

Stores an integer value as part of a component.

Pascal

```
property Tag: Longint;
```

C++

```
__property Longint Tag;
```

Description

Tag has no predefined meaning. The Tag property is provided for the convenience of developers. It can be used for storing an additional integer value or it can be typecast to any 32-bit value such as a component reference or a pointer.

C++ Examples:

```
/*
The following example enables the user to move the current
selected cell in a db grid.  The Up and Down buttons have
their OnClick events assigned to the UpDownClick procedure.
The Left and Right buttons have their OnClick events
assigned to the LeftRightClick procedure.  The Up and Left
buttons have their Tag property set to -1, while the Down
and Right buttons have their Tag property set to 1.  MoveBy
does not move the cell in the db grid, it only moves the
```

```
selection.  The data set is not edited.  Notice that the
Up button and the Down button are separate TButtons, not
one TUpDown.  The tags must be set different.
*/
void __fastcall TForm1::UpDownClick(TObject *Sender)
{
  CDS2->MoveBy(((TComponent *)Sender)->Tag);
  DBGrid2->SetFocus();
}

void __fastcall TForm1::LeftRightClick(TObject *Sender)
{
  DBGrid2->SelectedIndex = DBGrid2->SelectedIndex + ((TComponent *)Sender)->Tag;
  DBGrid2->SetFocus();
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
```

```
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
The following example enables the user to move the current
selected cell in a db grid.  The Up and Down buttons have
their OnClick events assigned to the UpDownClick procedure.
The Left and Right buttons have their OnClick events
assigned to the LeftRightClick procedure.  The Up and Left
buttons have their Tag property set to -1, while the Down
and Right buttons have their Tag property set to 1.  MoveBy
does not move the cell in the db grid, it only moves the
selection.  The data set is not edited.  Notice that the
Up button and the Down button are separate TButtons, not
one TUpDown.  The tags must be set different.
}
procedure TForm1.UpDownClick(Sender: TObject);
begin
  CDS.MoveBy(TComponent(Sender).Tag);
  DBGrid1.SetFocus;
end;

procedure TForm1.LeftRightClick(Sender: TObject);
begin
  DBGrid1.SelectedIndex := DBGrid1.SelectedIndex + TComponent(Sender).Tag;
  DBGrid1.SetFocus;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
```

```
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
```

```
end;
```

23.1.27.6.11 TComponent.VCLComObject Property

Represents information used internally by components that support COM.

Pascal

```
property VCLComObject: Pointer;
```

C++

```
__property void * VCLComObject;
```

Description

VCLComObject is for internal use only.

See Also

ComObject (see page 2793)

23.1.28 TComponentEnumerator Class

TComponentEnumerator allows the components within a containing component to be discovered.

Class Hierarchy

Classes.TComponentEnumerator

Pascal

```
TComponentEnumerator = class;
```

C++

```
class TComponentEnumerator;
```

File

Classes

Description

The TComponentEnumerator returns references to the components contained within a specified containing component. This allows an application to inspect the contents of a component at run time.

It is useful, for example, for repeated sets of controls in panels on a form. When the user selects one panel (by button press within, for example), the code can determine the parent panel, and then use TComponentEnumerator to get the names of the controls on this particular panel.

See Also

TComponent (see page 2762)

23.1.28.1 TComponentEnumerator Members

The following tables list the members exposed by TComponentEnumerator.

Public Methods

	Name	Description
	Create (see page 2813)	Creates a TComponentEnumerator (see page 2812) class instance.
	GetCurrent (see page 2813)	Retrieves a reference to the current component.
	MoveNext (see page 2814)	Moves to the first, or next component within the containing component.

Public Properties

	Name	Description
R	Current (see page 2814)	Returns the currently selected component.

23.1.28.2 TComponentEnumerator.Create Constructor

Creates a TComponentEnumerator (see page 2812) class instance.

Pascal

```
constructor Create(AComponent: TComponent);
```

C++

```
__fastcall TComponentEnumerator(TComponent * AComponent);
```

Description

Call Create to instantiate a TComponentEnumerator (see page 2812) object at runtime.

The component to be enumerated is defined by **AComponent**.

See Also

TComponent (see page 2762)

23.1.28.3 TComponentEnumerator Methods

The methods of the TComponentEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 2813)	Retrieves a reference to the current component.
	MoveNext (see page 2814)	Moves to the first, or next component within the containing component.

23.1.28.3.1 TComponentEnumerator.GetCurrent Method

Retrieves a reference to the current component.

Pascal

```
function GetCurrent: TComponent;
```

C++

```
__fastcall TComponent * GetCurrent();
```

Description

Before you ever call GetCurrent, you must call MoveNext (see page 2814), successfully, or an exception will be thrown.

MoveNext (see page 2814) moves through the list of components within the parent component specified at class instantiation. GetCurrent returns a reference to the component currently referenced after the last MoveNext (see page 2814) operation.

The Current (see page 2814) property has the same effect as GetCurrent

See Also

Current (see page 2814)

MoveNext (see page 2814)

TComponent (see page 2762)

23.1.28.3.2 TComponentEnumerator.MoveNext Method

Moves to the first, or next component within the containing component.

Pascal

```
function MoveNext: Boolean;
```

C++

```
__fastcall Boolean MoveNext();
```

Description

MoveNext moves through the list of components within the parent component specified at class instantiation. A value of true is returned if a component was found, otherwise false. MoveNext is normally used in a while-true block.

Note: Before you ever call GetCurrent (see page 2813), you must call MoveNext successfully, or an exception will be thrown.

See Also

GetCurrent (see page 2813)

TComponent (see page 2762)

23.1.28.4 TComponentEnumerator Properties

The properties of the TComponentEnumerator class are listed here.

Public Properties

	Name	Description
R	Current (see page 2814)	Returns the currently selected component.

23.1.28.4.1 TComponentEnumerator.Current Property

Returns the currently selected component.

Pascal

```
property Current: TComponent;
```

C++

```
__property TComponent * Current;
```

Description

Before you ever use the Current, you must call MoveNext (see page 2814), successfully, or an exception will be thrown.

MoveNext (see page 2814) moves through the list of components within the parent component specified at class instantiation. Current returns a reference to the component currently referenced after the last MoveNext (see page 2814) operation.

The Current property has the same effect as GetCurrent (see page 2813)

See Also

GetCurrent (see page 2813)

TComponent (see page 2762)

23.1.29 TCustomMemoryStream Class

TCustomMemoryStream is an abstract base class used as the common ancestor for memory streams.

Class Hierarchy

Pascal

```
TCustomMemoryStream = class(TStream);
```

C++

```
class TCustomMemoryStream : public TStream;
```

File

Classes

Description

Use TCustomMemoryStream as a base class when defining a stream object that can transfer data that is stored in memory. Memory (see page 2823) streams are useful for providing file-like access to data that is stored in a less accessible medium. Data can be moved to an internal memory buffer when the memory stream is created. After manipulating the data in a memory stream, the data can be written out to its actual storage medium when the memory stream is destroyed.

Do not instantiate an instance of TCustomMemoryStream. It is an abstract class that implements behavior common to all memory streams. To work with an instance of a memory stream, use one of the descendants of TCustomMemoryStream, such as TMemoryStream (see page 2902) or TResourceStream (see page 2968).

See Also

TMemoryStream (see page 2902)

TResourceStream (see page 2968)

23.1.29.1 TCustomMemoryStream Members

The following tables list the members exposed by TCustomMemoryStream.

Protected Methods

	Name	Description
	GetSize (see page 2982)	This is GetSize, a member of class TStream.
	SetPointer (see page 2823)	Replaces the memory buffer associated with the memory stream.
	SetSize (see page 2989)	This is the overview for the SetSize method overload.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CopyFrom (see page 2979)	Copies a specified number of bytes from one stream to another.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FixupResourceHeader (see page 2981)	Patches the resource header for a resource that has been written to the stream.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Read (see page 2817)	Reads up to Count bytes from the memory stream into Buffer and advances the current position of the stream by the number of bytes read.
	ReadBuffer (see page 2983)	Reads bytes from the stream into Buffer.
	ReadComponent (see page 2983)	Initiates streaming of components and their properties.
	ReadComponentRes (see page 2987)	Reads components and their properties from a stream in a resource file format.
	ReadResHeader (see page 2987)	Reads a resource-file header from the stream.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 2818)	Writes the entire contents of the memory stream to the file with a given file name.
	SaveToStream (see page 2818)	Writes the entire contents of the memory stream to the stream object specified by Stream.
V	Seek (see page 2819)	Moves the current position of the stream by Offset bytes, relative to the origin specified by Origin.
V	Seek (see page 2988)	This is the overview for the Seek method overload.
A	Write (see page 2989)	Introduces a pure virtual method for writing to the stream.
	WriteBuffer (see page 2990)	Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.
	WriteComponent (see page 2990)	Initiates the writing of components and their properties to a stream.
	WriteComponentRes (see page 2994)	Writes components and their properties to a stream from a resource file format.
	WriteDescendent (see page 2994)	Streams components and their properties in inherited forms.
	WriteDescendentRes (see page 2995)	Streams components and their properties in inherited forms.
	WriteResourceHeader (see page 2995)	Writes a resource-file header to the stream.

Public Properties

	Name	Description
R	Memory (see page 2823)	Provides direct access to the memory pool allocated for the memory stream.
	Position (see page 2996)	Indicates the current offset into the stream for reading and writing.
	Size (see page 2997)	Indicates the size in bytes of the stream.

23.1.29.2 TCustomMemoryStream Methods

The methods of the TCustomMemoryStream class are listed here.

Protected Methods

	Name	Description
	SetPointer (see page 2823)	Replaces the memory buffer associated with the memory stream.

Public Methods

	Name	Description
V	Read (see page 2817)	Reads up to Count bytes from the memory stream into Buffer and advances the current position of the stream by the number of bytes read.
	SaveToFile (see page 2818)	Writes the entire contents of the memory stream to the file with a given file name.
	SaveToStream (see page 2818)	Writes the entire contents of the memory stream to the stream object specified by Stream.
V	Seek (see page 2819)	Moves the current position of the stream by Offset bytes, relative to the origin specified by Origin.

23.1.29.2.1 TCustomMemoryStream.Read Method

Reads up to Count bytes from the memory stream into Buffer and advances the current position of the stream by the number of bytes read.

Pascal

```
function Read(var Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Read( Buffer, Longint Count);
```

Description

Use Read to read the contents of the memory stream into a buffer, starting at the current position. Read will read up to Count bytes from the current position in Memory (see page 2823). If Count bytes extends beyond the end of the memory buffer, Read will only transfer the data up to the end of the associated memory buffer. Read returns the number of bytes actually transferred to Buffer, and advances the current position accordingly. If the return value is less than Count, it means that reading reached the end of the stream data.

All the other data-reading methods of the memory stream (ReadBuffer (see page 2983), ReadComponent (see page 2983)) call Read to do the actual reading.

Note: Read treats Count as an upper bound. The ReadBuffer (see page 2983) method, by contrast, raises an exception if Count bytes cannot be read.

See Also

Memory (see page 2823)

Position (see page 2996)

ReadBuffer (see page 2983)

ReadComponent (see page 2983)

Write (see page 2989)

23.1.29.2.2 TCustomMemoryStream.SaveToFile Method

Writes the entire contents of the memory stream to the file with a given file name.

Pascal

```
procedure SaveToFile(const FileName: string);
```

C++

```
__fastcall SaveToFile(const AnsiString FileName);
```

Description

Use SaveToFile to write the contents of Memory (see page 2823) to a file. SaveToFile allows an application to write out the contents of the memory stream without having to explicitly create and free a file stream object. If the named file cannot be created or opened, SaveToFile raises an EFCreateError (see page 2695) exception.

See Also

EFcreateError

Memory (see page 2823)

SaveToStream (see page 2818)

TFileStream (see page 2841)

23.1.29.2.3 TCustomMemoryStream.SaveToStream Method

Writes the entire contents of the memory stream to the stream object specified by Stream.

Pascal

```
procedure SaveToStream(Stream: TStream);
```

C++

```
__fastcall SaveToStream(TStream Stream);
```

Description

Use SaveToStream to copy data that is stored in memory into another storage medium. SaveToStream writes the entire contents of Memory (see page 2823) into the indicated stream object, starting at the current position in the stream that was passed as a parameter.

When the Stream parameter is a TFileStream (see page 2841) object, SaveToStream does much the same thing as the SaveToFile (see page 2818) method. However, SaveToStream writes to the current position in the target stream. Thus, for example, SaveToStream can be used to append the contents of Memory (see page 2823) to a file stream, rather than replace the contents of the file the way SaveToFile (see page 2818) does.

If the entire contents of the memory stream cannot be written to the target stream, SaveToStream raises an EWriteError (see page 2706) exception.

See Also

EWriteError (see page 2706)

Memory (see page 2823)

SaveToFile (see page 2818)

Position (see page 2996)

23.1.29.2.4 TCustomMemoryStream.Seek Method

Moves the current position of the stream by Offset bytes, relative to the origin specified by Origin.

Pascal

```
function Seek(Offset: Longint; Origin: Word): Longint; override;
```

C++

```
virtual __fastcall Longint Seek(Longint Offset, Word Origin);
```

Description

Use Seek to move the current position within the memory stream by the indicated offset. Seek allows an application to read from or write to a particular location within the Memory (see page 2823) associated with the memory stream.

If Offset is a negative number, the seek is backward from the specified origin. The following table shows the different values of Origin and their meanings for seeking.

Value	Meaning
soFromBeginning (see page 3319)	Offset is from the beginning of Memory (see page 2823). Seek moves to the position Offset. Offset must be >= 0.
soFromCurrent (see page 3320)	Offset is from the current position. Seek moves to Position (see page 2996) + Offset.
soFromEnd (see page 3320)	Offset is from the end of Memory (see page 2823). Offset must be <= 0 to indicate a number of bytes before the end of the memory buffer.

Seek returns the new value of the Position (see page 2996) property.

Note: Seek does no error checking on the value provided for Offset. Do not call seek with an offset that would move the current position less than 0 (before the start of Memory (see page 2823)) or greater than Size (see page 2997) (beyond the end of the memory buffer).

C++ Examples:

```
/*
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
*/
class MyScrollBar : public TScrollBar
{
__published:    // IDE-managed Components
private:    // User declarations
public:     // User declarations
    __fastcall MyScrollBar(TComponent* Owner);
};

__fastcall MyScrollBar::MyScrollBar(TComponent* Owner)
    : TScrollBar(Owner)
{
}

void RegisterClassesWithStreamingSystem(void)
{
  // Make sure that as part of the startup
  // code our streaming classes are registered
```

```
    // with the streaming system.

    #pragma startup RegisterClassesWithStreamingSystem

    Classes::RegisterClass(__classid(MyScrollBar));
  }

  MyScrollBar *ScrollBar1;

  AnsiString __fastcall ComponentToString(TComponent *c)
  {
   AnsiString as;
   TMemoryStream* pms = new TMemoryStream();
   TStringStream* pss = new TStringStream(as);

   try
   {
      pms->WriteComponent(c);
      pms->Seek(0, soFromBeginning);
      ObjectBinaryToText(pms, pss);
      pss->Seek(0, soFromBeginning);
      as = pss->DataString;
   }
   catch(...)
   {
      ShowMessage("Streaming error.");
   }

   delete pms;
   delete pss;
   return as;
  }

  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    Memo1->Text = ComponentToString(ScrollBar1);
  }

  TComponent *__fastcall StringToComponent(AnsiString as)

  {
   TMemoryStream* pms = new TMemoryStream();
   TStringStream* pss = new TStringStream(as);
   TComponent *pc;

   try
   {
      ObjectTextToBinary(pss, pms);
      pms->Seek(0, soFromBeginning);
   }
   catch(...)
   {
      ShowMessage("Streaming error.");
   }

   pc = pms->ReadComponent(NULL);
   delete pms;
   delete pss;
   return pc;
  }

  void __fastcall TForm1::Button2Click(TObject *Sender)
  {
    TComponent *temp = StringToComponent(Memo1->Text);
    ScrollBar1->Free();
```

```
  ScrollBar1 = dynamic_cast<MyScrollBar *>(temp);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ScrollBar1 = new MyScrollBar(Form1);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
  ScrollBar1->Top = 48;
  ScrollBar1->Left = 250;
  ScrollBar1->Name = "Ricksbar";
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete ScrollBar1;
}
```

Delphi Examples:

```
{
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
}
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  MyScrollBar = class(TScrollBar)
  end;

var
  Form1: TForm1;
  ScrollBar1: MyScrollBar;

implementation

{$R *.dfm}

function ComponentToStringProc(Component: TComponent): string;
var
  BinStream:TMemoryStream;
  StrStream: TStringStream;
  s: string;
begin
  BinStream := TMemoryStream.Create;
  try
    StrStream := TStringStream.Create(s);
    try
      BinStream.WriteComponent(Component);
      BinStream.Seek(0, soFromBeginning);
      ObjectBinaryToText(BinStream, StrStream);
      StrStream.Seek(0, soFromBeginning);
      Result:= StrStream.DataString;
```

```
      finally
        StrStream.Free;
      end;
    finally
      BinStream.Free
    end;
  end;

  function StringToComponentProc(Value: string): TComponent;
  var
    StrStream:TStringStream;
    BinStream: TMemoryStream;
  begin
    StrStream := TStringStream.Create(Value);
    try
      BinStream := TMemoryStream.Create;
      try
        ObjectTextToBinary(StrStream, BinStream);
        BinStream.Seek(0, soFromBeginning);
        Result:= BinStream.ReadComponent(nil);
      finally
        BinStream.Free;
      end;
    finally
      StrStream.Free;
    end;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Memo1.Text:= ComponentToStringProc(ScrollBar1);
  end;

  // Edit the scrollbar parameters in the memo, click this
  // button and see the scrollbar change.
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    ScrollBar1.Free;
    ScrollBar1:= (StringToComponentProc(Memo1.Text) as MyScrollBar);
    ScrollBar1.Parent:= Form1;
    ScrollBar1.Visible:= TRUE;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ScrollBar1:= MyScrollBar.Create(Form1);
    ScrollBar1.Parent:= Form1;
    ScrollBar1.Visible:= TRUE;
    ScrollBar1.Top:= 48;
    ScrollBar1.Left:= 250;
    ScrollBar1.Name:= 'Ricksbar';
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    ScrollBar1.Free;
  end;

  initialization

  RegisterClass(MyScrollBar);
```

See Also

Memory (see page 2823)

Position (see page 2996)

Read (see page 2817)

Size (see page 2997)

Write (see page 2989)

23.1.29.2.5 TCustomMemoryStream.SetPointer Method

Replaces the memory buffer associated with the memory stream.

Pascal

```
procedure SetPointer(Ptr: Pointer; Size: Longint);
```

C++

```
__fastcall SetPointer(void * Ptr, Longint Size);
```

Description

Use SetPointer to set the internal memory buffer, Memory (see page 2823), to be the value passed in by Ptr. Size (see page 2997) is the number of bytes Ptr points to.

Descendants of TCustomMemoryStream (see page 2815) should use SetPointer to associate the memory stream with the memory buffer that holds the data for the memory stream.

Note: SetPointer does not free the existing value of Memory (see page 2823), if any, when it replaces the memory buffer. Descendants of TCustomMemoryStream (see page 2815) that use SetPointer to replace the stream's memory pool should free the memory pointed to by the Memory (see page 2823) property before calling SetPointer to replace the memory buffer.

See Also

Memory (see page 2823)

Size (see page 2997)

23.1.29.3 TCustomMemoryStream Properties

The properties of the TCustomMemoryStream class are listed here.

Public Properties

	Name	Description
R	Memory (see page 2823)	Provides direct access to the memory pool allocated for the memory stream.

23.1.29.3.1 TCustomMemoryStream.Memory Property

Provides direct access to the memory pool allocated for the memory stream.

Pascal

```
property Memory: Pointer;
```

C++

```
__property void * Memory;
```

Description

Use Memory to get access to the memory for the stream. The memory for the stream holds the data that is being transferred by means of the memory stream. Size (see page 2997) is the number of bytes of Memory that were allocated, and Position (see page 2996) is the current position within Memory.

Note: Memory is a read-only property. Memory can be used to change the contents of the memory, but to set the actual memory the stream works with, descendants of TCustomMemoryStream (see page 2815) must assign a pointer to a memory buffer by calling the SetPointer (see page 2823) method.

See Also

Position (see page 2996)

SetPointer (see page 2823)

Size (see page 2997)

23.1.30 TDataModule Class

TDataModule centralizes the handling of nonvisual components in an application.

Class Hierarchy

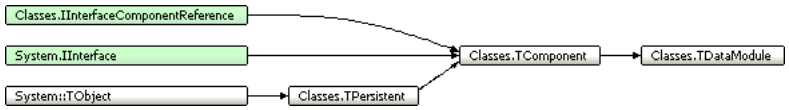

Pascal

```
TDataModule = class(TComponent);
```

C++

```
class TDataModule : public TComponent;
```

File

Classes

Description

Use a TDataModule object in an application to provide a location for centralized handling of nonvisual components. Typically these are data access components, such as TSQLDataSet, and TSQLConnection. DataModules are not limited to data access components, they can also contain other nonvisual components, such as TTimer, TOpenDialog, or TImageList).

At design time a TDataModule object provides a visual container into which a developer can place nonvisual components, set their properties, and write event handlers for them. To create a data module at design time, choose File | New Data Module.

In the unit file for the data module a developer may also place any business rules that are to be applied to the application.

To make the data module available to another unit in the application, select that unit, then choose File|Use Unit (when working in Delphi) or File|Include Unit Header (when working in C++) to add the data module to the uses clause for the unit.

See Also

TForm (see page 13992)

TFrame (see page 14114)

The Web Module

23.1.30.1 TDataModule Members

The following tables list the members exposed by TDataModule.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
V	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
V	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 2829)	Generates an OnCreate (see page 2833) event.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2830)	Generates an OnDestroy (see page 2834) event.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2827)	Creates an instance of a data module.
V	CreateNew (see page 2828)	Creates an instance of a data module and registers with the global screen object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2829)	Destroys the instance of the data module.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.
	Tag (see page 2808)	Stores an integer value as part of a component.

23.1.30.2 TDataModule.Create Constructor

Creates an instance of a data module.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDataModule(TComponent * AOwner);
```

Description

Call Create to instantiate a data module at runtime if it was not created at design time. Create calls CreateNew (see page 2828). If an error occurs, an exception is raised. Otherwise, if OldCreateOrder (see page 2833) is true, Create calls the OnCreate (see page 2833) event handler for the data module if one is assigned to it.

See Also

CreateNew (see page 2828)

Destroy (see page 2829)

OnCreate (see page 2833)

OldCreateOrder (see page 2833)

23.1.30.3 TDataModule.CreateNew Constructor

Creates an instance of a data module and registers with the global screen object.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); virtual;
```

C++

```
virtual __fastcall TDataModule(TComponent * AOwner, int Dummy = 0);
```

Description

Use CreateNew instead of Create (see page 2827) to create a data module without using the associated .form file to initialize it.

Warning: Using CreateNew instead of Create (see page 2827) can cause unpredictable results because most data modules are written assuming their controls will be created from the form file.

CreateNew calls the inherited constructor, and then informs the global screen object of the existence of the data module.

The first argument, AOwner, is the owner of the data module (usually the application). The second argument, Dummy, defaults to 0 and is not used by this constructor. The system passes 0 by default if this constructor is passed only the first argument, AOwner.

CreateNew bypasses the streaming in of the previously associated form file. Therefore, you must stream in an external form file to bind the visual components with their classes. You can stream in the previously associated form file by calling InitInheritedComponent (see page 3231). To stream in a different form file, bracket the call to CreateNew with calls to WriteComponentResFile (see page 3266) and ReadComponentResFile (see page 3247). The following code sequence

1. streams out an external form file.
2. creates a new data module disassociated from any form file;
3. streams in the external form file and binds it to this new data module.

```
WriteComponentResFile('Temp.xfm', DataModule1);
...
```

DataModule2 := TDataModule.CreateNew(Application);

```
ReadComponentResFile('Temp.xfm', DataModule2);
```

See Also

Create (see page 2827)

Destroy (see page 2829)

ReadComponentResFile

WriteComponentResFile

InitInheritedCOmponent

23.1.30.4 TDataModule.Destroy Destructor

Destroys the instance of the data module.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataModule();
```

Description

Do not call Destroy directly. Instead call Free, which checks that the data module reference is not nil before calling Destroy.

Before calling the inherited destructor, Destroy removes any fixup references, calls its OnDestroy (see page 2834) event handler (if one is defined and OldCreateOrder (see page 2833) is true), and informs the global screen variable that the data module is going away.

See Also

Create (see page 2827)

Free (see page 22300)

OnDestroy (see page 2834)

23.1.30.5 TDataModule Methods

The methods of the TDataModule class are listed here.

Protected Methods

	Name	Description
	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.
	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.

Public Methods

	Name	Description
	AfterConstruction (see page 2829)	Generates an OnCreate (see page 2833) event.
	BeforeDestruction (see page 2830)	Generates an OnDestroy (see page 2834) event.

23.1.30.5.1 TDataModule.AfterConstruction Method

Generates an OnCreate (see page 2833) event.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called after the data module's constructor has finished. Do not call it explicitly in applications.

TDataModule (see page 2824) overrides AfterConstruction to generate an OnCreate (see page 2833) event when OldCreateOrder (see page 2833) is false.

See Also

BeforeDestruction (see page 2830)

OnCreate (see page 2833)

OldCreateOrder (see page 2833)

23.1.30.5.2 TDataModule.BeforeDestruction Method

Generates an OnDestroy (see page 2834) event.

Pascal

```
procedure BeforeDestruction; override;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

BeforeDestruction is called immediately before the data module's destructor. Do not call it explicitly in applications.

TDataModule (see page 2824) overrides BeforeDestruction to generate an OnDestroy (see page 2834) event when OldCreateOrder (see page 2833) is false.

See Also

AfterConstruction (see page 2829)

OnDestroy (see page 2834)

OldCreateOrder (see page 2833)

23.1.30.5.3 TDataModule.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor (see page 2839)

DefineBinaryProperty (see page 2837)

DefineProperties (see page 2931)

DefineProperty (see page 2838)

23.1.30.5.4 TDataModule.DoCreate Method

Pascal

```
procedure DoCreate; virtual;
```

C++

```
virtual __fastcall DoCreate();
```

Description

This is DoCreate, a member of class TDataModule.

23.1.30.5.5 TDataModule.DoDestroy Method

Pascal

```
procedure DoDestroy; virtual;
```

C++

```
virtual __fastcall DoDestroy();
```

Description

This is DoDestroy, a member of class TDataModule.

23.1.30.5.6 TDataModule.GetChildren Method

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

This is GetChildren, a member of class TDataModule.

23.1.30.5.7 TDataModule.HandleCreateException Method

Pascal

```
function HandleCreateException: Boolean; dynamic;
```

C++

```
__fastcall Boolean HandleCreateException();
```

Description

This is HandleCreateException, a member of class TDataModule.

23.1.30.5.8 TDataModule.ReadState Method

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

This is ReadState, a member of class TDataModule.

23.1.30.6 TDataModule Properties

The properties of the TDataModule class are listed here.

Public Properties

	Name	Description
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.

Published Properties

	Name	Description
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.

23.1.30.6.1 TDataModule.DesignOffset Property

Specifies the design offset for the data module at design time.

Pascal

```
property DesignOffset: TPoint;
```

C++

```
__property TPoint DesignOffset;
```

Description

An application should never need to set this value. It is used to control the position of the data module at design time.

See Also

DesignSize (see page 2832)

23.1.30.6.2 TDataModule.DesignSize Property

Specifies the design size for the data module at design time.

Pascal

```
property DesignSize: TPoint;
```

C++

```
__property TPoint DesignSize;
```

Description

An application should never need to set this value. It controls the size of the data module window at design time.

See Also

DesignOffset (see page 2832)

23.1.30.6.3 TDataModule.OldCreateOrder Property

Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.

Pascal

```
property OldCreateOrder: Boolean;
```

C++

```
__property Boolean OldCreateOrder;
```

Description

When OldCreateOrder is false (the default) the OnCreate (see page 2833) event occurs after all constructors are finished (from the AfterConstruction (see page 2829) method) and the OnDestroy (see page 2834) event occurs before any destructors are called (from the BeforeDestruction (see page 2830) method).

When OldCreateOrder is true, OnCreate (see page 2833) is triggered from the data module constructor and OnDestroy (see page 2834) from the data module destructor.

Note: This timing differs from Delphi version 3 and previous and from C++Builder version 1, where the OnCreate (see page 2833) event occurred when the TDataModule (see page 2824) constructor executed and the OnDestroy (see page 2834) event occurred when the TDataModule (see page 2824) destructor executed. Applications that require the OnCreate (see page 2833) event and OnDestroy (see page 2834) events to occur from the constructor and destructor of the data module can set OldCreateOrder to true.

See Also

OnCreate (see page 2833)

AfterConstruction (see page 2829)

OnDestroy (see page 2834)

BeforeDestruction (see page 2830)

23.1.30.7 TDataModule Events

The events of the TDataModule class are listed here.

Published Events

	Name	Description
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.

23.1.30.7.1 TDataModule.OnCreate Event

Occurs when an application instantiates a data module.

Pascal

```
property OnCreate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCreate;
```

Description

Write an OnCreate event handler to take specific actions when an application instantiates a data module. For example, if a data module contains database and dataset components, an application may establish a database connection immediately. If the data module contains timers, the application may initialize them.

Note: Use of the OnCreate event is discouraged in C++ because it can interact badly with the data module's constructor (see OldCreateOrder (see page 2833)). It is recommended that you override the constructor instead.

See Also

OnDestroy (see page 2834)

OldCreateOrder (see page 2833)

23.1.30.7.2 TDataModule.OnDestroy Event

Occurs when the data module is about to be destroyed.

Pascal

```
property OnDestroy: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDestroy;
```

Description

Write an OnDestroy event handler to take specific actions when an application frees a data module. For example, if the unit code for the data module instantiates any objects of its own, such as string lists, the OnDestroy event handler can be used to free those objects.

See Also

OnCreate (see page 2833)

OldCreateOrder (see page 2833)

23.1.31 TFiler Class

TFiler is the abstract base class for reader and writer objects, which are used for loading and saving components and their properties.

Class Hierarchy

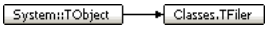

Pascal

```
TFiler = class(TObject);
```

C++

```
class TFiler : public TObject;
```

File

Classes

Description

TFiler works with stream and component objects. It handles the internal mechanics of the streaming system that reads and writes components and their associated data. This streaming system is the mechanism by which the IDE saves components to or reads components from disk (in form files) and by which it saves and restores temporary in-memory images of components.

TFiler has two descendants, TReader (see page 2933) and TWriter (see page 3175), that implement the methods introduced by TFiler. The filer type is used as a parameter to component methods that can require either a reader or writer object, depending on whether the data is read from or written to a stream. Every filer has an associated stream that is passed as a parameter to its constructor. Filer objects are usually created by the stream that becomes associated with the filer.

TFiler properties provide support for managing the streaming of owned components. Filer objects read and write the properties of components to and from a stream. Filer objects provide buffering of data to speed read and write operations.

TFiler introduces abstract methods that are customized by its descendant reader and writer objects. These include methods that enable components to load and save hidden or complex data as if it was the value of a published property.

Note: Most of the filer properties and methods have public visibility for convenience within the streaming system. The majority of them are never called directly by applications.

See Also

TComponent (see page 2762)

TReader (see page 2933)

TStream (see page 2973)

TWriter (see page 3175)

Using Streams to Read or Write Data

23.1.31.1 TFiler Members

The following tables list the members exposed by TFiler.

Protected Methods

	Name	Description
	SetRoot (see page 2838)	This is SetRoot, a member of class TFiler.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2836)	Creates a filer object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

A	DefineBinaryProperty (see page 2837)	Introduces an abstract method that allows descendants to read and write unformatted data like a property.
A	DefineProperty (see page 2838)	Introduces an abstract method that allows descendants to read or write data like a property.
V	Destroy (see page 2837)	Destroys a filer object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
A	FlushBuffer (see page 2838)	Introduces an abstract method that allows descendants to synchronize the filer's buffer with its stream.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Ancestor (see page 2839)	Determines whether to write the properties of inherited components.
	IgnoreChildren (see page 2839)	Enables a writer object to save a component without saving the components it owns.
R	LookupRoot (see page 2840)	Specifies the root component for resolving names while loading and saving components.
	Root (see page 2840)	Indicates which object is the root, or ultimate owner, of the objects read or written by the filer.

23.1.31.2 TFiler.Create Constructor

Creates a filer object.

Pascal

```
constructor Create(Stream: TStream; BufSize: Integer);
```

C++

```
__fastcall TFiler(TStream Stream, int BufSize);
```

Description

Call Create to instantiate a filer descendant at runtime, if necessary. It is seldom necessary to directly create a filer object because the methods and routines that use them automatically create filers.

Create allocates memory for a filer object, and associates it with the stream passed in the Stream parameter, with a buffer of size BufSize.

See Also

ReadComponent (see page 2983)

WriteDescendent (see page 2994)

23.1.31.3 TFiler.Destroy Destructor

Destroys a filer object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TFiler();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which checks that the filer reference is not nil before calling Destroy.

The Destroy method for filer objects~TFiler (see page 2834) frees memory associated with the buffer. Memory deallocation for the object instance is handled automatically when Destroy is called. TFiler (see page 2834)'s Destroy method is called by descendant objects in their respective destructors.

See Also

Destroy (see page 22287)

Free (see page 22300)

23.1.31.4 TFiler Methods

The methods of the TFiler class are listed here.

Protected Methods

	Name	Description
	SetRoot (see page 2838)	This is SetRoot, a member of class TFiler.

Public Methods

	Name	Description
	DefineBinaryProperty (see page 2837)	Introduces an abstract method that allows descendants to read and write unformatted data like a property.
	DefineProperty (see page 2838)	Introduces an abstract method that allows descendants to read or write data like a property.
	FlushBuffer (see page 2838)	Introduces an abstract method that allows descendants to synchronize the filer's buffer with its stream.

23.1.31.4.1 TFiler.DefineBinaryProperty Method

Introduces an abstract method that allows descendants to read and write unformatted data like a property.

Pascal

```
procedure DefineBinaryProperty(const Name: string; ReadData: TStreamProc; WriteData:
TStreamProc; HasData: Boolean); virtual; abstract;
```

C++

```
virtual __fastcall DefineBinaryProperty(const AnsiString Name, TStreamProc ReadData,
TStreamProc WriteData, Boolean HasData) = 0;
```

Description

Each descendant filer object declares a DefineBinaryProperty method that reads or writes binary data as if the data were a

property.

See Also

DefineProperties (see page 2771)

DefineProperty (see page 2838)

23.1.31.4.2 TFiler.DefineProperty Method

Introduces an abstract method that allows descendants to read or write data like a property.

Pascal

```
procedure DefineProperty(const Name: string; ReadData: TReaderProc; WriteData: TWriterProc;
HasData: Boolean); virtual; abstract;
```

C++

```
virtual __fastcall DefineProperty(const AnsiString Name, TReaderProc ReadData, TWriterProc
WriteData, Boolean HasData) = 0;
```

Description

Each descendant filer object declares a DefineProperty method that defines the data that the descendant object will read or write as if the data were a property.

See Also

DefineProperties (see page 2771)

DefineBinaryProperty (see page 2837)

23.1.31.4.3 TFiler.FlushBuffer Method

Introduces an abstract method that allows descendants to synchronize the filer's buffer with its stream.

Pascal

```
procedure FlushBuffer; virtual; abstract;
```

C++

```
virtual __fastcall FlushBuffer() = 0;
```

Description

Each descendant filer object defines a FlushBuffer method to provide practical implementations for synchronizing the filer object's buffer with the associated stream.

23.1.31.4.4 TFiler.SetRoot Method

Pascal

```
procedure SetRoot(Value: TComponent); virtual;
```

C++

```
virtual __fastcall SetRoot(TComponent * Value);
```

Description

This is SetRoot, a member of class TFiler.

23.1.31.5 TFiler Properties

The properties of the TFiler class are listed here.

Public Properties

	Name	Description
	Ancestor (see page 2839)	Determines whether to write the properties of inherited components.
	IgnoreChildren (see page 2839)	Enables a writer object to save a component without saving the components it owns.
R	LookupRoot (see page 2840)	Specifies the root component for resolving names while loading and saving components.
	Root (see page 2840)	Indicates which object is the root, or ultimate owner, of the objects read or written by the filer.

23.1.31.5.1 TFiler.Ancestor Property

Determines whether to write the properties of inherited components.

Pascal

```
property Ancestor: TPersistent;
```

C++

```
__property TPersistent * Ancestor;
```

Description

Do not use Ancestor directly. It is used internally by the methods of writer objects to write components in inherited forms.

In Visual Form Inheritance, a writer object only needs to write the values of properties that differ from those that are inherited. It tracks each component from which properties can be inherited in Ancestor and compares properties before writing. If Ancestor is nil (Delphi) or NULL (C++), there is no corresponding component from which to inherit, and the writer object writes the component out to the stream completely.

Ancestor is only non-nil (Delphi) or non-NULL (C++) when writing forms created with Visual Form Inheritance. The streaming system uses WriteDescendent and WriteDescendentRes to write such forms to the stream.

When writing or overriding a DefineProperties method, be aware that Ancestor might be set, and it might be necessary to write or omit properties, as appropriate. Normally, DefineProperty (see page 2838) read procedures ignore the value of Ancestor.

See Also

DefineProperties (see page 2771)

WriteComponent (see page 3183)

WriteDescendent (see page 2994)

WriteDescendent (see page 3184)

WriteDescendentRes (see page 2995)

23.1.31.5.2 TFiler.IgnoreChildren Property

Enables a writer object to save a component without saving the components it owns.

Pascal

```
property IgnoreChildren: Boolean;
```

C++

```
__property Boolean IgnoreChildren;
```

Description

Programs rarely use IgnoreChildren directly. Programs use IgnoreChildren internally to avoid writing owned components when the Root (see page 2840) is changed during the streaming process. A few components change Root (see page 2840) while streaming, but this is rare.

If IgnoreChildren is true, the writer writes a component without writing any child components it owns. If IgnoreChildren is false, the writer object writes all owned objects to the stream.

See Also

WriteComponent (see page 3183)

Root (see page 2840)

23.1.31.5.3 TFiler.LookupRoot Property

Specifies the root component for resolving names while loading and saving components.

Pascal

```
property LookupRoot: TComponent;
```

C++

```
__property TComponent * LookupRoot;
```

Description

The Filer sets LookupRoot when loading or saving components to resolve name references in nested frames. This allows the Writer to look up component names relative to a local root component, even when a frame is nested in another root component.

See Also

Root (see page 2840)

23.1.31.5.4 TFiler.Root Property

Indicates which object is the root, or ultimate owner, of the objects read or written by the filer.

Pascal

```
property Root: TComponent;
```

C++

```
__property TComponent * Root;
```

Description

Do not call Root directly. It is used internally to find the root object when streaming owned components. Root represents the ultimate owner of all the components being streamed during the current streaming process.

The ReadRootComponent and WriteRootComponent methods set Root before reading or writing their components and owned components.

See Also

Owner (see page 2959)

ReadRootComponent (see page 2952)

RootAncestor (see page 3192)

WriteRootComponent (see page 3188)

LookupRoot (see page 2840)

23.1.32 TFileStream Class

TFileStream enables applications to read from and write to a file on disk.

Class Hierarchy

Pascal

```
TFileStream = class(THandleStream);
```

C++

```
class TFileStream : public THandleStream;
```

File

Classes

Description

Use TFileStream to access the information in disk files. TFileStream will open a named file and provide methods to read from or write to it. If an application already has a handle to the file, opened in the appropriate mode, use THandleStream (see page 2845) instead.

See Also

ReadComponentResFile (see page 3247)

THandleStream (see page 2845)

WriteComponentResFile (see page 3266)

Setting the Response Content

23.1.32.1 TFileStream Members

The following tables list the members exposed by TFileStream.

Protected Fields

	Name	Description
	FHandle (see page 2848)	This is FHandle, a member of class THandleStream.

Protected Methods

	Name	Description
	GetSize (see page 2982)	This is GetSize, a member of class TStream.
	SetSize (see page 2849)	This is the overview for the SetSize method overload.
	SetSize (see page 2989)	This is the overview for the SetSize method overload.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CopyFrom (see page 2979)	Copies a specified number of bytes from one stream to another.
	Create (see page 2843)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2844)	Destroys an instance of TFileStream (see page 2841).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FixupResourceHeader (see page 2981)	Patches the resource header for a resource that has been written to the stream.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Read (see page 2848)	Reads up to Count bytes of data from the resource associated with the handle stream into Buffer.
A	Read (see page 2982)	Introduces an abstract or pure virtual method responsible for reading from the stream.
	ReadBuffer (see page 2983)	Reads bytes from the stream into Buffer.
	ReadComponent (see page 2983)	Initiates streaming of components and their properties.
	ReadComponentRes (see page 2987)	Reads components and their properties from a stream in a resource file format.
	ReadResHeader (see page 2987)	Reads a resource-file header from the stream.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Seek (see page 2849)	Resets the current position of the handle stream.
V	Seek (see page 2988)	This is the overview for the Seek method overload.
V	Write (see page 2850)	Writes Count bytes from the Buffer to the current position in the resource.
A	Write (see page 2989)	Introduces a pure virtual method for writing to the stream.
	WriteBuffer (see page 2990)	Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.
	WriteComponent (see page 2990)	Initiates the writing of components and their properties to a stream.
	WriteComponentRes (see page 2994)	Writes components and their properties to a stream from a resource file format.
	WriteDescendent (see page 2994)	Streams components and their properties in inherited forms.
	WriteDescendentRes (see page 2995)	Streams components and their properties in inherited forms.
	WriteResourceHeader (see page 2995)	Writes a resource-file header to the stream.

Public Properties

	Name	Description
R	FileName (see page 2845)	Preserves the file name of the file associated with the file stream.
R	Handle (see page 2850)	Specifies the handle for the communications resource the stream reads from and writes to.

	Position (see page 2996)	Indicates the current offset into the stream for reading and writing.
	Size (see page 2997)	Indicates the size in bytes of the stream.

Strict private Fields

	Name	Description
	FFileName (see page 2844)	This is FFileName, a member of class TFileStream.

23.1.32.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TFileStream.Create (string, Word) (see page 2843)	Creates an instance of TFileStream (see page 2841).
	TFileStream.Create (string, Word, Cardinal) (see page 2843)	Creates an instance of TFileStream (see page 2841).

23.1.32.2.1 TFileStream.Create Constructor (string, Word)

Creates an instance of TFileStream (see page 2841).

Pascal

```
constructor Create(const AFileName: string; Mode: Word); overload;
constructor Create(const AFileName: string; Mode: Word; Rights: Cardinal); overload;
```

C++

```
__fastcall TFileStream(const AnsiString AFileName, Word Mode);
__fastcall TFileStream(const AnsiString AFileName, Word Mode, unsigned Rights);
```

Description

Call Create to instantiate a file stream for reading from or writing to the named file. Specify the name of the file and the way the file should be opened as parameters.

The Mode parameter indicates how the file is to be opened. The Mode parameter consists of an open mode and (possibly) a share mode or'ed together. The open mode must be one of the following values:

Value	Meaning
fmCreate (see page 3316)	Create a file with the given name. If a file with the given name exists, open the file in write mode.
fmOpenRead	Open the file for reading only.
fmOpenWrite	Open the file for writing only. Writing to the file completely replaces the current contents.
fmOpenReadWrite	Open the file to modify the current contents rather than replace them.

The share mode must be one of the following values:

Value	Meaning
fmShareCompat	Sharing is compatible with the way FCBs are opened.
fmShareExclusive	Other applications can not open the file for any reason.
fmShareDenyWrite	Other applications can open the file for reading but not for writing.

fmShareDenyRead	Other applications can open the file for writing but not for reading.
fmShareDenyNone	No attempt is made to prevent other applications from reading from or writing to the file.

The Rights parameter indicates the permission bits with which to create the file on Linux when Mode is fmCreate (see page 3316). Rights is ignored when used on the Windows platform.

If the file can not be opened, Create raises an exception.

See Also

FileCreate (see page 22855)

FileOpen (see page 22867)

23.1.32.3 TFileStream.Destroy Destructor

Destroys an instance of TFileStream (see page 2841).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TFileStream();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which checks that the TFileStream (see page 2841) reference is not nil and only then calls Destroy.

Destroy closes the Handle (see page 2850) for the file stream before freeing the object.

See Also

FileClose (see page 22852)

Free (see page 22300)

Handle (see page 2850)

23.1.32.4 TFileStream Fields

The fields of the TFileStream class are listed here.

Strict private Fields

	Name	Description
	FFileName (see page 2844)	This is FFileName, a member of class TFileStream.

23.1.32.4.1 TFileStream.FFileName Field

Pascal

```
FFileName: string;
```

C++

```
AnsiString FFileName;
```

Description

This is FFileName, a member of class TFileStream.

23.1.32.5 TFileStream Properties

The properties of the TFileStream class are listed here.

Public Properties

	Name	Description
R	FileName (see page 2845)	Preserves the file name of the file associated with the file stream.

23.1.32.5.1 TFileStream.FileName Property

Preserves the file name of the file associated with the file stream.

Pascal

```
property FileName: string;
```

C++

```
__property AnsiString FileName;
```

Description

Use FileName to display the name of the file that TFileStream (see page 2841) has accessed from disk. This is a read only property.

23.1.33 THandleStream Class

THandleStream enables applications to read from and write to communications resources identified by a handle.

Class Hierarchy

Pascal

```
THandleStream = class(TStream);
```

C++

```
class THandleStream : public TStream;
```

File

Classes

Description

Use THandleStream to access files, sockets, named pipes, mailslots, or other communications resources that provide a handle when opened. For example, the FileOpen function provides a handle for a file on disk. THandleStream allows applications to use a uniform stream interface when performing I/O using a handle.

To avoid the overhead of managing file handles, use TFileStream (see page 2841) to work with disk files.

See Also

FileClose

FileCreate

FileOpen

TFileStream (see page 2841)

23.1.33.1 THandleStream Members

The following tables list the members exposed by THandleStream.

Protected Fields

	Name	Description
	FHandle (see page 2848)	This is FHandle, a member of class THandleStream.

Protected Methods

	Name	Description
V	GetSize (see page 2982)	This is GetSize, a member of class TStream.
V	SetSize (see page 2849)	This is the overview for the SetSize method overload.
V	SetSize (see page 2989)	This is the overview for the SetSize method overload.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CopyFrom (see page 2979)	Copies a specified number of bytes from one stream to another.
	Create (see page 2847)	Creates an instance of THandleStream (see page 2845).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FixupResourceHeader (see page 2981)	Patches the resource header for a resource that has been written to the stream.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Read (see page 2848)	Reads up to Count bytes of data from the resource associated with the handle stream into Buffer.

	ReadBuffer (see page 2983)	Reads bytes from the stream into Buffer.
	ReadComponent (see page 2983)	Initiates streaming of components and their properties.
	ReadComponentRes (see page 2987)	Reads components and their properties from a stream in a resource file format.
	ReadResHeader (see page 2987)	Reads a resource-file header from the stream.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Seek (see page 2849)	Resets the current position of the handle stream.
V	Seek (see page 2988)	This is the overview for the Seek method overload.
V	Write (see page 2850)	Writes Count bytes from the Buffer to the current position in the resource.
	WriteBuffer (see page 2990)	Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.
	WriteComponent (see page 2990)	Initiates the writing of components and their properties to a stream.
	WriteComponentRes (see page 2994)	Writes components and their properties to a stream from a resource file format.
	WriteDescendent (see page 2994)	Streams components and their properties in inherited forms.
	WriteDescendentRes (see page 2995)	Streams components and their properties in inherited forms.
	WriteResourceHeader (see page 2995)	Writes a resource-file header to the stream.

Public Properties

	Name	Description
R	Handle (see page 2850)	Specifies the handle for the communications resource the stream reads from and writes to.
	Position (see page 2996)	Indicates the current offset into the stream for reading and writing.
	Size (see page 2997)	Indicates the size in bytes of the stream.

23.1.33.2 THandleStream.Create Constructor

Creates an instance of THandleStream (see page 2845).

Pascal

```
constructor Create(AHandle: Integer);
```

C++

```
__fastcall THandleStream(int AHandle);
```

Description

Call Create to instantiate a THandleStream (see page 2845) for a given handle.

The handle must be obtained by opening or creating the resource in the appropriate mode. For example, to create a handle stream for reading from a file, obtain the file handle by opening the file with an fmOpenRead or fmOpenReadWrite mode. To create a handle stream for writing to a file, obtain the file handle by opening the file with an fmOpenWrite or fmOpenReadWrite mode.

See Also

FileCreate (see page 22855)

FileOpen (see page 22867)

Handle (see page 2850)

23.1.33.3 THandleStream Fields

The fields of the THandleStream class are listed here.

Protected Fields

	Name	Description
	FHandle (see page 2848)	This is FHandle, a member of class THandleStream.

23.1.33.3.1 THandleStream.FHandle Field

Pascal

```
FHandle: Integer;
```

C++

```
int FHandle;
```

Description

This is FHandle, a member of class THandleStream.

23.1.33.4 THandleStream Methods

The methods of the THandleStream class are listed here.

Protected Methods

	Name	Description
	SetSize (see page 2849)	This is the overview for the SetSize method overload.

Public Methods

	Name	Description
	Read (see page 2848)	Reads up to Count bytes of data from the resource associated with the handle stream into Buffer.
	Seek (see page 2849)	Resets the current position of the handle stream.
	Write (see page 2850)	Writes Count bytes from the Buffer to the current position in the resource.

23.1.33.4.1 THandleStream.Read Method

Reads up to Count bytes of data from the resource associated with the handle stream into Buffer.

Pascal

```
function Read(var Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Read( Buffer, Longint Count);
```

Description

Use Read to read data from the resource associated with the handle stream when the number of bytes in the file is not known. Read transfers up to Count bytes from the resource, starting at the current position, and then advances the current position in the resource by the number of bytes actually transferred. Read returns the number of bytes actually transferred, which may be less than Count if the end of file marker is encountered.

All other data-reading methods of a handle stream (ReadBuffer (see page 2983), ReadComponent (see page 2983)) call Read to do the actual reading.

See Also

Position (see page 2996)

ReadBuffer (see page 2983)

ReadComponent (see page 2983)

23.1.33.4.2 THandleStream.Seek Method

Resets the current position of the handle stream.

Pascal

```
function Seek(const Offset: Int64; Origin: TSeekOrigin): Int64; override;
```

C++

```
virtual __fastcall Int64 Seek(const Int64 Offset, TSeekOrigin Origin);
```

Description

Use Seek to move the current position within the resource associated with the handle stream by the indicated offset. Seek allows an application to read from or write to a particular location within the resource.

The Origin parameter indicates how to interpret the Offset parameter. Origin should be one of the following values:

Value	Meaning
soFromBeginning (see page 3319)	Offset is from the beginning of the resource. Seek moves to the position Offset. Offset must be >= 0.
soFromCurrent (see page 3320)	Offset is from the current position in the resource. Seek moves to Position (see page 2996) + Offset.
soFromEnd (see page 3320)	Offset is from the end of the resource. Offset must be <= 0 to indicate a number of bytes before the end of the file.

Seek returns the new value of the Position (see page 2996) property, the new current position in the resource.

See Also

Position (see page 2996)

Read (see page 2848)

Write (see page 2850)

23.1.33.4.3 SetSize Method

This is the overview for the SetSize method overload.

Overload List

	Name	Description
	THandleStream.SetSize (Int64) (see page 2849)	Sets the end of file marker to truncate the resource at the indicated position.
	THandleStream.SetSize (Longint) (see page 2849)	Sets the end of file marker to truncate the resource at the indicated position.

23.1.33.4.3.1 THandleStream.SetSize Method (Int64)

Sets the end of file marker to truncate the resource at the indicated position.

Pascal

```
procedure SetSize(NewSize: Longint); override;
procedure SetSize(const NewSize: Int64); override;
```

C++

```
virtual __fastcall SetSize(Longint NewSize);
virtual __fastcall SetSize(const Int64 NewSize);
```

Description

Call SetSize to set the size of the resource. SetSize calls Seek (see page 2849) to go to the indicated position, and then writes an end of file marker. If the size of the resource can not be changed, an exception is raised. For example, calling SetSize for a file handle that was opened in fmOpenRead mode will raise an exception.

See Also

Size (see page 2997)

23.1.33.4.4 THandleStream.Write Method

Writes Count bytes from the Buffer to the current position in the resource.

Pascal

```
function Write(const Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  Buffer, Longint Count);
```

Description

Use Write to write Count bytes to the resource associated with the handle stream, starting at the current position. After writing to the resource, Write advances the current position by the number bytes written, and returns the number of bytes written.

All other data-writing methods of a handle stream (WriteBuffer (see page 2990), WriteComponent (see page 2990)) call Write to do the actual writing.

See Also

Position (see page 2996)

WriteBuffer (see page 2990)

WriteComponent (see page 2990)

23.1.33.5 THandleStream Properties

The properties of the THandleStream class are listed here.

Public Properties

	Name	Description
R	Handle (see page 2850)	Specifies the handle for the communications resource the stream reads from and writes to.

23.1.33.5.1 THandleStream.Handle Property

Specifies the handle for the communications resource the stream reads from and writes to.

Pascal

```
property Handle: Integer;
```

C++

```
__property int Handle;
```

Description

Read (see page 2848) Handle to get the handle for file management functions. To read from or write to the resource, use the methods of the THandleStream (see page 2845) object.

Handle is a read-only property. The handle property cannot be changed to allow the handle stream to switch from reading to writing or vice versa. For example, to change from a file handle that is opened in read-only mode to one that is opened in write mode:

1Free the stream object.

2Call FileClose to close the file.

3Reopen the file in write mode, and use the handle to create a new instance of a handle stream. Alternately, open a TFileStream (see page 2841) object for the file, specifying a write mode for the stream.

Note: Do not call the FileClose function on the Handle until after the THandleStream (see page 2845) object has been destroyed.

See Also

TFileStream (see page 2841)

Create (see page 2847)

FileClose (see page 22852)

FileGetDate (see page 22866)

FileSetDate (see page 22878)

Free (see page 22300)

23.1.34 TInterfacedPersistent Class

TInterfacedPersistent is an ancestor for persistent objects that implement interfaces.

Class Hierarchy

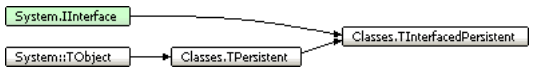

Pascal

```
TInterfacedPersistent = class(TPersistent, IInterface);
```

C++

```
class TInterfacedPersistent : public TPersistent, public IInterface;
```

File

Classes

Description

TInterfacedPersistent, like all persistent objects, supports the ability to read and write its properties to and from a stream. In addition, it supplies a default implementation of the IInterface methods (_AddRef (see page 2853), _Release (see page 2854), and QueryInterface (see page 2854)). This default implementation simply passes these calls on to the interface of the persistent object's Owner, if any.

Do not create instances of TInterfacedPersistent. Use TInterfacedPersistent as a base class when declaring objects that are not components, but that need to be saved to a stream or have their properties assigned to other objects, and that support interfaces.

See Also

TComponent (see page 2762)

TFiler (see page 2834)

TReader (see page 2933)

TStream (see page 2973)

TWriter (see page 3175)

23.1.34.1 TInterfacedPersistent Members

The following tables list the members exposed by TInterfacedPersistent.

Protected Methods

	Name	Description
	_AddRef (see page 2853)	Increments the reference count for the interfaced object's interface.
	_Release (see page 2854)	Decrements the reference count for the interfaced object's interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Methods

	Name	Description
V	AfterConstruction (see page 2854)	Obtains a reference to the Owner's interface.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	QueryInterface (see page 2854)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

23.1.34.2 TInterfacedPersistent Methods

The methods of the TInterfacedPersistent class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 2853)	Increments the reference count for the interfaced object's interface.
	_Release (see page 2854)	Decrements the reference count for the interfaced object's interface.

Public Methods

	Name	Description
V	AfterConstruction (see page 2854)	Obtains a reference to the Owner's interface.
V	QueryInterface (see page 2854)	Returns a reference to a specified interface if the object supports that interface.

23.1.34.2.1 TInterfacedPersistent._AddRef Method

Increments the reference count for the interfaced object's interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef implements the IInterface method, __AddRef, which increments the reference count of an interface and returns the current reference count.

If the interfaced persistent object has an Owner and that owner supports an interface, then __AddRef simply calls the __AddRef method of the Owner. Otherwise, it simply returns –1.

See Also

QueryInterface (see page 2854)

_Release (see page 2854)

23.1.34.2.2 TInterfacedPersistent._Release Method

Decrements the reference count for the interfaced object's interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

___Release implements the IInterface method, ___Release, which decrements the reference count of an interface and returns the current reference count. For some interfaces, when this causes the reference count to drop to zero, the object frees itself.

If the interfaced persistent object has an Owner and that owner supports an interface, then ___Release simply calls the ___Release method of the Owner. Otherwise, it simply returns –1.

See Also

QueryInterface (see page 2854)

_AddRef (see page 2853)

23.1.34.2.3 TInterfacedPersistent.AfterConstruction Method

Obtains a reference to the Owner's interface.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called automatically after the object's last constructor executes. Do not call it explicitly in your applications.

As implemented in TInterfacedPersistent (see page 2851), AfterConstruction calls the GetOwner (see page 2932) method to determine whether this object has an Owner. If so, it obtains a reference to the Owner's interface, to which it can delegate IInterface calls.

See Also

GetOwner (see page 2932)

23.1.34.2.4 TInterfacedPersistent.QueryInterface Method

Returns a reference to a specified interface if the object supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface implements the IInterface method, QueryInterface. This method checks whether the interfaced persistent object

supports the interface identified by IID. If so, it returns a reference as the Obj parameter, automatically incrementing the reference count for the interface. If the object does not support the specified interface, it returns a non-zero value.

See Also

_AddRef (see page 2853)

_Release (see page 2854)

23.1.35 TInterfaceList Class

TInterfaceList represents a list of interfaces.

Class Hierarchy

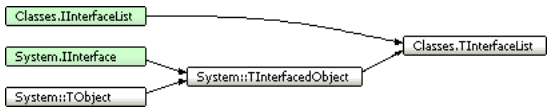

Pascal

```
TInterfaceList = class(TInterfacedObject, IInterfaceList);
```

C++

```
class TInterfaceList : public TInterfacedObject, public IInterfaceList;
```

File

Classes

Description

Use TInterfaceList to manage a list of interfaces.

See Also

Working with Lists

23.1.35.1 TInterfaceList Members

The following tables list the members exposed by TInterfaceList.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	Get (see page 2860)	Returns a specified interface from the list.
	GetCapacity (see page 2860)	Returns the value of the Capacity property.
	GetCount (see page 2861)	Returns the value of the Count property.
	Put (see page 2863)	Replaces an interface in the list with a specified interface.

	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetCapacity (see page 2864)	Sets the value of the Capacity property.
	SetCount (see page 2864)	Sets the value of the Count property.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Add (see page 2858)	Adds an interface to the list.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2858)	Clears the interface list.
	Create (see page 2857)	Creates an interface list.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2859)	Deletes an interface from the list.
V	Destroy (see page 2857)	Destroys the interface list.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Exchange (see page 2859)	Swaps the position of two interfaces in the list.
	Expand (see page 2859)	Increases the capacity of the interface list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	First (see page 2860)	Returns the first interface in the list.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2861)	Returns a TInterfaceList (see page 2855) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	IndexOf (see page 2861)	Returns the index of an interface.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2862)	Inserts an interface into the list at a specified position.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Last (see page 2862)	Returns the last interface in the list.
	Lock (see page 2862)	Prevents other threads from accessing the list.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Remove (see page 2863)	Removes an interface from the list.

	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Unlock (see page 2864)	Unlocks the list, allowing other threads to access it.

Public Properties

	Name	Description
	Capacity (see page 2865)	Specifies the list capacity.
	Count (see page 2865)	Specifies the number of interfaces in the list.
	Items (see page 2866)	Provides direct access to an interface in the list.
	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

23.1.35.2 TInterfaceList.Create Constructor

Creates an interface list.

Pascal

```
constructor Create;
```

C++

```
__fastcall TInterfaceList();
```

Description

Call Create to create an interface list.

23.1.35.3 TInterfaceList.Destroy Destructor

Destroys the interface list.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TInterfaceList();
```

Description

Do not call Destroy directly in an application. Instead, call the Free method.

See Also

Free (see page 22300)

23.1.35.4 TInterfaceList Methods

The methods of the TInterfaceList class are listed here.

Protected Methods

	Name	Description
	Get (see page 2860)	Returns a specified interface from the list.
	GetCapacity (see page 2860)	Returns the value of the Capacity property.
	GetCount (see page 2861)	Returns the value of the Count property.
	Put (see page 2863)	Replaces an interface in the list with a specified interface.
	SetCapacity (see page 2864)	Sets the value of the Capacity property.
	SetCount (see page 2864)	Sets the value of the Count property.

Public Methods

	Name	Description
	Add (see page 2858)	Adds an interface to the list.
	Clear (see page 2858)	Clears the interface list.
	Delete (see page 2859)	Deletes an interface from the list.
	Exchange (see page 2859)	Swaps the position of two interfaces in the list.
	Expand (see page 2859)	Increases the capacity of the interface list.
	First (see page 2860)	Returns the first interface in the list.
	GetEnumerator (see page 2861)	Returns a TInterfaceList (see page 2855) enumerator.
	IndexOf (see page 2861)	Returns the index of an interface.
	Insert (see page 2862)	Inserts an interface into the list at a specified position.
	Last (see page 2862)	Returns the last interface in the list.
	Lock (see page 2862)	Prevents other threads from accessing the list.
	Remove (see page 2863)	Removes an interface from the list.
	Unlock (see page 2864)	Unlocks the list, allowing other threads to access it.

23.1.35.4.1 TInterfaceList.Add Method

Adds an interface to the list.

Pascal

```
function Add(const Item: IInterface): Integer;
```

C++

```
__fastcall int Add(const IInterface Item);
```

Description

Call Add to add an interface to the list.

See Also

Count (see page 2865)

Add

Items (see page 2866)

Delete (see page 2859)

Capacity (see page 2865)

23.1.35.4.2 TInterfaceList.Clear Method

Clears the interface list.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to remove all interfaces from the list.

See Also

Items (see page 2866)

Delete (see page 2859)

23.1.35.4.3 TInterfaceList.Delete Method

Deletes an interface from the list.

Pascal

```
procedure Delete(Index: Integer);
```

C++

```
__fastcall Delete(int Index);
```

Description

Call Delete to delete an interface from the list.

See Also

Count (see page 2865)

Add (see page 2858)

Items (see page 2866)

Clear (see page 2858)

Remove (see page 2863)

23.1.35.4.4 TInterfaceList.Exchange Method

Swaps the position of two interfaces in the list.

Pascal

```
procedure Exchange(Index1: Integer; Index2: Integer);
```

C++

```
__fastcall Exchange(int Index1, int Index2);
```

Description

Call Exchange to move the interface at position Index1 so that it appears at position Index2 and move the interface at position Index2 so that it appears at position Index1. Index1 and Index2 identify two positions (zero offset) in the Items (see page 2866) property array.

See Also

Items (see page 2866)

23.1.35.4.5 TInterfaceList.Expand Method

Increases the capacity of the interface list.

Pascal

```
function Expand: TInterfaceList;
```

C++

```
__fastcall TInterfaceList Expand();
```

Description

If Capacity (see page 2865) is greater than 64, then Capacity (see page 2865) is increased by Capacity (see page 2865) / 4. If Capacity (see page 2865) is between eight and 64, Capacity (see page 2865) is increased by 16. If Capacity (see page 2865) is less than eight, Capacity (see page 2865) is increased by four.

See Also

Count (see page 2865)

Capacity (see page 2865)

23.1.35.4.6 TInterfaceList.First Method

Returns the first interface in the list.

Pascal

```
function First: IInterface;
```

C++

```
__fastcall IInterface First();
```

Description

Call First to obtain the first interface in the list.

See Also

Items (see page 2866)

Last (see page 2862)

23.1.35.4.7 TInterfaceList.Get Method

Returns a specified interface from the list.

Pascal

```
function Get(Index: Integer): IInterface;
```

C++

```
__fastcall IInterface Get(int Index);
```

Description

Get is the read implementation of the Items property.

Index identifies which interface to return, where 0 is the first interface, 1 is the second interface, and so on.

See Also

Items (see page 3206)

23.1.35.4.8 TInterfaceList.GetCapacity Method

Returns the value of the Capacity property.

Pascal

```
function GetCapacity: Integer;
```

C++

```
__fastcall int GetCapacity();
```

Description

GetCapacity is the read implementation of the Capacity property.

See Also

Capacity (see page 3205)

23.1.35.4.9 TInterfaceList.GetCount Method

Returns the value of the Count property.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the read implementation of the Count property.

See Also

Count (see page 3206)

23.1.35.4.10 TInterfaceList.GetEnumerator Method

Returns a TInterfaceList (see page 2855) enumerator.

Pascal

```
function GetEnumerator: TInterfaceListEnumerator;
```

C++

```
__fastcall TInterfaceListEnumerator GetEnumerator();
```

Description

GetEnumerator returns a TInterfaceListEnumerator (see page 2866) reference, which enumerates all IInterfaces in the Interface list.

To do so, call the TInterfaceListEnumerator (see page 2866) GetCurrent method within a While MoveNext do loop.

See Also

TInterfaceListEnumerator (see page 2866)

23.1.35.4.11 TInterfaceList.IndexOf Method

Returns the index of an interface.

Pascal

```
function IndexOf(const Item: IInterface): Integer;
```

C++

```
__fastcall int IndexOf(const IInterface Item);
```

Description

Call IndexOf to obtain the index of an interface.

See Also

Items (see page 2866)

23.1.35.4.12 TInterfaceList.Insert Method

Inserts an interface into the list at a specified position.

Pascal

```
procedure Insert(Index: Integer; const Item: IInterface);
```

C++

```
__fastcall Insert(int Index, const IInterface Item);
```

Description

Call Insert to insert an interface into the list. Item is the interface to insert, and Index indicates the position (zero-offset) where the interface should be added.

See Also

Count (see page 2865)

Add (see page 2858)

Items (see page 2866)

Delete (see page 2859)

Capacity (see page 2865)

23.1.35.4.13 TInterfaceList.Last Method

Returns the last interface in the list.

Pascal

```
function Last: IInterface;
```

C++

```
__fastcall IInterface Last();
```

Description

Call Last to obtain the last interface in the list.

See Also

First (see page 2860)

Items (see page 2866)

23.1.35.4.14 TInterfaceList.Lock Method

Prevents other threads from accessing the list.

Pascal

```
procedure Lock;
```

C++

```
__fastcall Lock();
```

Description

Call Lock to lock the list, preventing other threads from accessing the list.

See Also

Unlock (see page 2864)

23.1.35.4.15 TInterfaceList.Put Method

Replaces an interface in the list with a specified interface.

Pascal

```
procedure Put(Index: Integer; const Item: IInterface);
```

C++

```
__fastcall Put(int Index, const IInterface Item);
```

Description

Put is the write implementation of the Items property.

Index is the index of the interface to replace.

Item is the interface that replaces the interface identified by Index.

See Also

Items (see page 3206)

23.1.35.4.16 TInterfaceList.Remove Method

Removes an interface from the list.

Pascal

```
function Remove(const Item: IInterface): Integer;
```

C++

```
__fastcall int Remove(const IInterface Item);
```

Description

Call Remove to remove an interface from the list.

Remove returns the index of the removed interface, or –1 if the interface was not found.

See Also

Add (see page 2858)

Insert (see page 2862)

Delete (see page 2859)

Clear (see page 2858)

23.1.35.4.17 TInterfaceList.SetCapacity Method

Sets the value of the Capacity property.

Pascal

```
procedure SetCapacity(NewCapacity: Integer);
```

C++

```
__fastcall SetCapacity(int NewCapacity);
```

Description

SetCapacity is the write implementation of the Capacity property.

NewCapacity is the new capacity for the interface list.

See Also

Capacity (see page 3205)

23.1.35.4.18 TInterfaceList.SetCount Method

Sets the value of the Count property.

Pascal

```
procedure SetCount(NewCount: Integer);
```

C++

```
__fastcall SetCount(int NewCount);
```

Description

SetCount is the write implementation of the Count property.

NewCount is the new number of interfaces in the list.

See Also

Count (see page 3206)

23.1.35.4.19 TInterfaceList.Unlock Method

Unlocks the list, allowing other threads to access it.

Pascal

```
procedure Unlock;
```

C++

```
__fastcall Unlock();
```

Description

Call Unlock to unlock the list so that other threads can have access to the interfaces it contains.

See Also

Lock (see page 2862)

23.1.35.5 TInterfaceList Properties

The properties of the TInterfaceList class are listed here.

Public Properties

	Name	Description
	Capacity (see page 2865)	Specifies the list capacity.
	Count (see page 2865)	Specifies the number of interfaces in the list.
	Items (see page 2866)	Provides direct access to an interface in the list.

23.1.35.5.1 TInterfaceList.Capacity Property

Specifies the list capacity.

Pascal

```
property Capacity: Integer;
```

C++

```
__property int Capacity;
```

Description

Use Capacity to determine the list capacity. Before adding many interfaces to the list, you can improve performance by setting Capacity to the final number of interfaces. This prevents the interface list from having to reallocate the list every time an interface is added.

See Also

Count (see page 2865)

Add (see page 2858)

Items (see page 2866)

Expand (see page 2859)

23.1.35.5.2 TInterfaceList.Count Property

Specifies the number of interfaces in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use Count to determine the number of interfaces in the list.

See Also

Capacity (see page 2865)

Add (see page 2858)

Items (see page 2866)

Delete (see page 2859)

23.1.35.5.3 TInterfaceList.Items Property

Provides direct access to an interface in the list.

Pascal

```
property Items [Index: Integer]: IInterface;
```

C++

```
__property IInterface Items[int Index];
```

Description

Use Items to directly access an interface in the list. Index identifies each interface by its position in the list.

Note: In Delphi, Items is the default property of TInterfaceList (see page 2855). That means you can omit the property name. Thus, instead of writing

Note: InterfaceList1.Items[i]

Note: you can write

Note: InterfaceList1[i]

See Also

Count (see page 2865)

Add (see page 2858)

Delete (see page 2859)

operator_sb

23.1.36 TInterfaceListEnumerator Class

Enumerates through the IInterfaces in a TInterfaceList (see page 2855).

Class Hierarchy

Classes.TInterfaceListEnumerator

Pascal

```
TInterfaceListEnumerator = class;
```

C++

```
class TInterfaceListEnumerator;
```

File

Classes

Description

TInterfaceListEnumerator provides an enumerator for TInterfaceLists.

23.1.36.1 TInterfaceListEnumerator Members

The following tables list the members exposed by TInterfaceListEnumerator.

Public Methods

	Name	Description
	Create (see page 2867)	Creates a TInterfaceList (see page 2855) enumerator.
	GetCurrent (see page 2867)	Gets the current IInterface.
	MoveNext (see page 2868)	Moves the enumerator to the next IInterface in the collection.

Public Properties

	Name	Description
R	Current (see page 2868)	The current IInterface.

23.1.36.2 TInterfaceListEnumerator.Create Constructor

Creates a TInterfaceList (see page 2855) enumerator.

Pascal

```
constructor Create(AInterfaceList: TInterfaceList);
```

C++

```
__fastcall TInterfaceListEnumerator(TInterfaceList AInterfaceList);
```

Description

Creates an instance of TInterfaceListEnumerator (see page 2866).

23.1.36.3 TInterfaceListEnumerator Methods

The methods of the TInterfaceListEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 2867)	Gets the current IInterface.
	MoveNext (see page 2868)	Moves the enumerator to the next IInterface in the collection.

23.1.36.3.1 TInterfaceListEnumerator.GetCurrent Method

Gets the current IInterface.

Pascal

```
function GetCurrent: IInterface;
```

C++

```
__fastcall IInterface GetCurrent();
```

Description

Use GetCurrent to get the current IInterface of the TInterfaceList (see page 2855) object being enumerated.

23.1.36.3.2 TInterfaceListEnumerator.MoveNext Method

Moves the enumerator to the next IInterface in the collection.

Pascal

```
function MoveNext: Boolean;
```

C++

```
__fastcall Boolean MoveNext();
```

Description

Use MoveNext to move the TInterfaceListEnumerator (see page 2866) to the next IInterface in the TInterfaceList (see page 2855) object.

23.1.36.4 TInterfaceListEnumerator Properties

The properties of the TInterfaceListEnumerator class are listed here.

Public Properties

	Name	Description
R	Current (see page 2868)	The current IInterface.

23.1.36.4.1 TInterfaceListEnumerator.Current Property

The current IInterface.

Pascal

```
property Current: IInterface;
```

C++

```
__property IInterface Current;
```

Description

Current contains the current IInterface in the TInterfaceList (see page 2855) object being enumerated.

23.1.37 TList Class

TList stores an array of pointers.

Class Hierarchy

Pascal

```
TList = class(TObject);
```

C++

```
class TList : public TObject;
```

File

Classes

Description

TList, which stores an array of pointers, is often used to maintain lists of objects. TList introduces properties and methods to

Add (see page 2871) or delete the objects in the list.

Rearrange the objects in the list.

Locate and access objects in the list.

Sort (see page 2892) the objects in the list.

See Also

TCollection (see page 2731)

TStrings (see page 3029)

TComponent (see page 2762)

TObjectList (see page 5677)

TComponentList (see page 5653)

TClassList (see page 5646)

TStack (see page 5701)

Working with Lists

23.1.37.1 TList Members

The following tables list the members exposed by TList.

Protected Methods

	Name	Description
	Get (see page 2878)	This is Get, a member of class TList.
V	Grow (see page 2879)	This is Grow, a member of class TList.
V	Notify (see page 2884)	Responds when items are added, removed, or extracted from the list.
	Put (see page 2887)	This is Put, a member of class TList.
	SetCapacity (see page 2891)	This is SetCapacity, a member of class TList.
	SetCount (see page 2892)	This is SetCount, a member of class TList.

Public Methods

	Name	Description
	Add (see page 2871)	Inserts a new item at the end of the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2874)	Copies elements of one list to another.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 2874)	Deletes all items from the list.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

	Delete (see page 2874)	Removes the item at the position given by the Index parameter.
V	Destroy (see page 2870)	Destroys an instance of TList (see page 2868).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Error (see page 2875)	This is the overview for the Error method overload.
	Exchange (see page 2876)	Swaps the position of two items in the Items (see page 2898) array.
	Expand (see page 2877)	Increases the Capacity (see page 2895) of the list.
	Extract (see page 2877)	Removes a specified item from the list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	First (see page 2878)	Returns Items (see page 2898)[0].
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2878)	Returns a TList (see page 2868) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	IndexOf (see page 2879)	Returns the index of the first entry in the Items (see page 2898) array with a specified value.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2882)	Adds an object to the Items (see page 2898) array at the position specified by Index.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Last (see page 2882)	Returns Items (see page 2898)[Count (see page 2896) - 1].
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	Move (see page 2884)	Changes the position of an item in the Items (see page 2898) array.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Pack (see page 2885)	Deletes all nil (Delphi) or NULL (C++) items from the Items (see page 2898) array.
	Remove (see page 2887)	Deletes the first reference to the Item parameter from the Items (see page 2898) array.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Sort (see page 2892)	Performs a QuickSort on the list based on the comparison function Compare.

Public Properties

	Name	Description
	Capacity (see page 2895)	Specifies the allocated size of the array of pointers maintained by the TList (see page 2868) object.
	Count (see page 2896)	Indicates the number of entries in the list that are in use.
	Items (see page 2898)	Lists the object references.
R	List (see page 2900)	Specifies the array of pointers that make up the Items (see page 2898) array.

23.1.37.2 TList.Destroy Destructor

Destroys an instance of TList (see page 2868).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TList();
```

Description

Do not call Destroy directly. Instead, call Free. Free verifies that the TList (see page 2868) reference is not nil, and only then calls Destroy.

Destroy frees the memory used to store the list of items.

Note: Destroy does not free the memory pointed to by the elements of the list.

See Also

Free (see page 22300)

23.1.37.3 TList Methods

The methods of the TList class are listed here.

Protected Methods

	Name	Description
	Get (see page 2878)	This is Get, a member of class TList.
V	Grow (see page 2879)	This is Grow, a member of class TList.
V	Notify (see page 2884)	Responds when items are added, removed, or extracted from the list.
	Put (see page 2887)	This is Put, a member of class TList.
	SetCapacity (see page 2891)	This is SetCapacity, a member of class TList.
	SetCount (see page 2892)	This is SetCount, a member of class TList.

Public Methods

	Name	Description
	Add (see page 2871)	Inserts a new item at the end of the list.
	Assign (see page 2874)	Copies elements of one list to another.
V	Clear (see page 2874)	Deletes all items from the list.
	Delete (see page 2874)	Removes the item at the position given by the Index parameter.
	Error (see page 2875)	This is the overview for the Error method overload.
	Exchange (see page 2876)	Swaps the position of two items in the Items (see page 2898) array.
	Expand (see page 2877)	Increases the Capacity (see page 2895) of the list.
	Extract (see page 2877)	Removes a specified item from the list.
	First (see page 2878)	Returns Items (see page 2898)[0].
	GetEnumerator (see page 2878)	Returns a TList (see page 2868) enumerator.
	IndexOf (see page 2879)	Returns the index of the first entry in the Items (see page 2898) array with a specified value.
	Insert (see page 2882)	Adds an object to the Items (see page 2898) array at the position specified by Index.
	Last (see page 2882)	Returns Items (see page 2898)[Count (see page 2896) - 1].
	Move (see page 2884)	Changes the position of an item in the Items (see page 2898) array.
	Pack (see page 2885)	Deletes all nil (Delphi) or NULL (C++) items from the Items (see page 2898) array.
	Remove (see page 2887)	Deletes the first reference to the Item parameter from the Items (see page 2898) array.
	Sort (see page 2892)	Performs a QuickSort on the list based on the comparison function Compare.

23.1.37.3.1 TList.Add Method

Inserts a new item at the end of the list.

Pascal

```
function Add(Item: Pointer): Integer;
```

C++

```
__fastcall int Add(void * Item);
```

Description

Call Add to insert a new object at the end of the Items (see page 2898) array. Add returns the index of the new item, where the first item in the list has an index of 0.

Add increments Count (see page 2896) and, if necessary, allocates memory by increasing the value of Capacity (see page 2895).

Note: Add always inserts the Item pointer at the end of the Items (see page 2898) array, even if the Items (see page 2898) array contains nil (Delphi) or NULL (C++) pointers.

C++ Examples:

```
/*
This example creates a list object and inserts two records
into it. The value of the record fields are written on a
paintbox:
*/

typedef struct AList
{
  int I;
  char C;
} TAList;

typedef TAList* PAList;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PAList AStruct;
  TList *MyList = new TList;
  // fill the TList
  AStruct = new TAList;
  AStruct->I = 100;
  AStruct->C = 'Z';
  MyList->Add(AStruct);
  AStruct = new TAList;
  AStruct->I = 200;
  AStruct->C = 'X';
  MyList->Add(AStruct);

  // Go through the list, writing the elements to the
  // canvas of a paintbox component.
  int Y = 10; // position on canvas
  for (int i = 0; i < MyList->Count; i++)
  {
    AStruct = (PAList) MyList->Items[i];
    PaintBox1->Canvas->TextOut(10, Y, IntToStr(AStruct->I));
    Y += 30;  // Increment Y Value again
    PaintBox1->Canvas->TextOut(10, Y, AStruct->C);
    Y += 30;  //Increment Y Value
  }

  // Clean up - must free memory for the items as well as the list
  for (int i = 0; i < MyList->Count; i++)
  {
    AStruct = (PAList) MyList->Items[i];
    delete AStruct;
  }
  delete MyList;
```

```
}
```

Delphi Examples:

```
{
This example creates a list object and inserts two records
into it. The value of the record fields are written on a
paintbox:
}
procedure TForm1.Button1Click(Sender: TObject);
type
  PMyList = ^AList;
  AList = record
    I: Integer;
    C: Char;
  end;
var
  MyList: TList;
  ARecord: PMyList;
  B: Byte;
  Y: Word;
begin
  MyList := TList.Create;
  try
    New(ARecord);
    ARecord^.I := 100;
    ARecord^.C := 'Z';
    MyList.Add(ARecord); {Add integer 100 and character Z to list}
    New(ARecord);
    ARecord^.I := 200;
    ARecord^.C := 'X';
    MyList.Add(ARecord); {Add integer 200 and character X to list}

    { Now paint the items onto the paintbox}
    Y := 10;             {Variable used in TextOut function}
    for B := 0 to (MyList.Count - 1) do
    begin
      ARecord := MyList.Items[B];
      Canvas.TextOut(10, Y, IntToStr(ARecord^.I)); {Display I}
      Y := Y + 30;  {Increment Y Value again}
      Canvas.TextOut(10, Y, ARecord^.C);  {Display C}
      Y := Y + 30;  {Increment Y Value}
    end;

    { Cleanup: must free the list items as well as the list }
   for B := 0 to (MyList.Count - 1) do
   begin
     ARecord := MyList.Items[B];
     Dispose(ARecord);
   end;
  finally
    MyList.Free;
  end;
end;
```

See Also

Capacity (see page 2895)

Count (see page 2896)

Delete (see page 2874)

Insert (see page 2882)

Items (see page 2898)

Remove (see page 2887)

23.1.37.3.2 TList.Assign Method

Copies elements of one list to another.

Pascal

```
procedure Assign(ListA: TList; AOperator: TListAssignOp = laCopy; ListB: TList = nil);
```

C++

```
__fastcall Assign(TList ListA, TListAssignOp AOperator = laCopy, TList ListB = nil);
```

Description

Call Assign to assign the elements of another list to this one. Assign combines the source list with this one using the logical operator specified by the AOperator parameter.

If the ListB parameter is specified (Delphi) or not NULL (C++), then Assign first replaces all the elements of this list with those in ListA, and then merges ListB into this list using the operator specified by AOperator.

If the ListB parameter is not specified (Delphi) or NULL (C++), then Assign merges ListA into this list using the operator specified by AOperator.

See Also

Items (see page 2898)

23.1.37.3.3 TList.Clear Method

Deletes all items from the list.

Pascal

```
procedure Clear; virtual;
```

C++

```
virtual __fastcall Clear();
```

Description

Call Clear to empty the Items (see page 2898) array and set the Count (see page 2896) to 0. Clear also frees the memory used to store the Items (see page 2898) array and sets the Capacity (see page 2895) to 0.

See Also

Capacity (see page 2895)

Count (see page 2896)

Delete (see page 2874)

Items (see page 2898)

Remove (see page 2887)

23.1.37.3.4 TList.Delete Method

Removes the item at the position given by the Index parameter.

Pascal

```
procedure Delete(Index: Integer);
```

C++

```
__fastcall Delete(int Index);
```

Description

Call Delete to remove the item at a specific position from the list. The index is zero-based, so the first item has an Index value of 0, the second item has an Index value of 1, and so on. Calling Delete moves up all items in the Items (see page 2898) array that follow the deleted item, and reduces the Count (see page 2896).

To remove the reference to an item without deleting the entry from the Items (see page 2898) array and changing the Count (see page 2896), set the Items (see page 2898) property for Index to nil (Delphi) or NULL (C++).

Note: Delete does not free any memory associated with the item. To free the memory that was used to store a deleted item, set the Capacity (see page 2895) property.

See Also

Add (see page 2871)

Capacity (see page 2895)

Clear (see page 2874)

Count (see page 2896)

Insert (see page 2882)

Items (see page 2898)

Remove (see page 2887)

Extract (see page 2877)

TObjectList (see page 5677)

TComponentList (see page 5653)

23.1.37.3.5 Error Method

This is the overview for the Error method overload.

Overload List

	Name	Description
	TList.Error (PResStringRec, Integer) (see page 2875)	Raises an EListError (see page 2701) exception.
	TList.Error (string, Integer) (see page 2875)	Raises an EListError (see page 2701) exception.

23.1.37.3.5.1 TList.Error Method (PResStringRec, Integer)

Raises an EListError (see page 2701) exception.

Pascal

```
class procedure Error(const Msg: string; Data: Integer); virtual; overload;
class procedure Error(Msg: PResStringRec; Data: Integer); overload;
```

C++

```
virtual __fastcall Error(const AnsiString Msg, int Data);
__fastcall Error(PResStringRec Msg, int Data);
```

Description

Call Error to raise an exception when an error occurs working with a TList (see page 2868) object. Error assembles an error message from the format string (or resource string) passed as the Msg parameter and the data value passed as the Data parameter, and then raises an EListError (see page 2701) exception.

Call Error rather than adding a line such as

Note: raise EListError.CreateFmt(MyMsg, iBadValue);

Note: or

Note: resourcestring sBadValueMessage = '%s not a valid list value';

Note: ...

Note: EListError.Create(@sBadValueMessage, iBadValue);

Note: throw EListError (see page 2701)(MyMsg, iBadValue);

to reduce the code size of an application.

See Also

EListError (see page 2701)

Create (see page 22752)

CreateFmt (see page 22754)

23.1.37.3.6 TList.Exchange Method

Swaps the position of two items in the Items (see page 2898) array.

Pascal

```
procedure Exchange(Index1: Integer; Index2: Integer);
```

C++

```
__fastcall Exchange(int Index1, int Index2);
```

Description

Call Exchange to swap the positions of the items at positions Index1 and Index2 of the Items (see page 2898) array. The indexes are zero-based, so the first item in the list has an index value of 0, the second item has an index value of 1, and so on.

See Also

Items (see page 2898)

Move (see page 2884)

23.1.37.3.7 TList.Expand Method

Increases the Capacity (see page 2895) of the list.

Pascal

```
function Expand: TList;
```

C++

```
__fastcall TList Expand();
```

Description

Call Expand to create more space for adding new items to the list. Expand does nothing if the list is not already filled to Capacity (see page 2895).

If Count (see page 2896) = Capacity (see page 2895), Expand increases the Capacity (see page 2895) of the list as follows. If the value of Capacity (see page 2895) is greater than 8, Expand increases the Capacity (see page 2895) of the list by 16. If the value of Capacity (see page 2895) is greater than 4, but less than 9, the Capacity (see page 2895) of the list increases by 8. If the value of Capacity (see page 2895) is less than 4, the Capacity (see page 2895) of the list grows by 4.

The returned value is the expanded list object.

See Also

Capacity (see page 2895)

Count (see page 2896)

23.1.37.3.8 TList.Extract Method

Removes a specified item from the list.

Pascal

```
function Extract(Item: Pointer): Pointer;
```

C++

```
__fastcall void * Extract(void * Item);
```

Description

Call Extract to remove an item from the list. After the item is removed, all the objects that follow it are moved up in index position and Count (see page 2896) is decremented.

To remove the reference to an item without deleting the entry from the Items (see page 2898) array and changing the Count (see page 2896), set the Items (see page 2898) property for Index to nil (Delphi) or NULL (C++).

See Also

Add (see page 2871)

Capacity (see page 2895)

Clear (see page 2874)

Count (see page 2896)

Insert (see page 2882)

Items (see page 2898)

Remove (see page 2887)

Delete (see page 2874)

TObjectList (see page 5677)

TComponentList (see page 5653)

23.1.37.3.9 TList.First Method

Returns Items (see page 2898)[0].

Pascal

```
function First: Pointer;
```

C++

```
__fastcall void * First();
```

Description

Call First to get the first pointer in the Items (see page 2898) array.

See Also

Items (see page 2898)

Last (see page 2882)

23.1.37.3.10 TList.Get Method

Pascal

```
function Get(Index: Integer): Pointer;
```

C++

```
__fastcall void * Get(int Index);
```

Description

This is Get, a member of class TList.

23.1.37.3.11 TList.GetEnumerator Method

Returns a TList (see page 2868) enumerator.

Pascal

```
function GetEnumerator: TListEnumerator;
```

C++

```
__fastcall TListEnumerator GetEnumerator();
```

Description

GetEnumerator returns a TListEnumerator (see page 2900) reference, which enumerates all items in the list.

To do so, call the TListEnumerator (see page 2900) GetCurrent method within a While MoveNext do loop.

See Also

TListEnumerator (see page 2900)

23.1.37.3.12 TList.Grow Method

Pascal

```
procedure Grow; virtual;
```

C++

```
virtual __fastcall Grow();
```

Description

This is Grow, a member of class TList.

23.1.37.3.13 TList.IndexOf Method

Returns the index of the first entry in the Items (see page 2898) array with a specified value.

Pascal

```
function IndexOf(Item: Pointer): Integer;
```

C++

```
__fastcall int IndexOf(void * Item);
```

Description

Call IndexOf to get the index for a pointer in the Items (see page 2898) array. Specify the pointer as the Item parameter.

The first item in the array has index 0, the second item has index 1, and so on. If an item is not in the list, IndexOf returns -1. If a pointer appears more than once in the array, IndexOf returns the index of the first appearance.

C++ Examples:

```
/*
The following code adds an object to MyList if it isn't
already in the list.
*/
typedef struct AList
{
  int I;
  char C;
} TAList;

typedef AList* PAList;

TList *MyList;
PAList ARecord1, ARecord2;

void __fastcall DisplayTList(TList *theList)
{
  PAList AStruct;
  // Go through the list, writing the elements to the
  // canvas of a paintbox component.
  int Y = 10; // position on canvas
  for (int i = 0; i < theList->Count; i++)
  {
    AStruct = (PAList) theList->Items[i];
    Form1->PaintBox1->Canvas->TextOut(10, Y, IntToStr(AStruct->I));
    Y += 30;  // Increment Y Value again
    Form1->PaintBox1->Canvas->TextOut(10, Y, AStruct->C);
    Y += 30;  //Increment Y Value
  }
}
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (MyList->IndexOf(ARecord1) == -1) MyList->Add(ARecord1);
  DisplayTList(MyList);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  if (MyList->IndexOf(ARecord2) == -1) MyList->Add(ARecord2);
  DisplayTList(MyList);
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  if (MyList->Count != 0)
  {
    for (int B = MyList->Count - 1; B >= 0; B--)
    {
      PAList ARecord;
      ARecord = PAList(MyList->Items[B]);
      MyList->Remove(ARecord);
    }
  }
  Form1->PaintBox1->Repaint();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyList = new TList;
  ARecord1 = new TAList;
  ARecord1->I = 100;
  ARecord1->C = 'Z';
  ARecord2 = new TAList;
  ARecord2->I = 200;
  ARecord2->C = 'X';
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  // Clean up - must free memory for the items as well as the list
  for (int i = 0; i < MyList->Count; i++)
  {
    PAList ARecord = (PAList) MyList->Items[i];
    delete ARecord;
  }
  delete MyList;

}
```

Delphi Examples:

```
{
The following code adds an object to MyList if it isn't
already in the list.
}
type
  PMyList = ^AList;
  AList = record
    I: Integer;
    C: Char;
  end;
var
  Form1: TForm1;
  MyList: TList;
  ARecord1, ARecord2: PMyList;

implementation
```

```
{$R *.dfm}

procedure DisplayTList(TheList: TList);
var
  ARecord: PMyList;
  B: Byte;
  Y: Word;
begin
    { Now paint the items onto the paintbox}
    Y := 10;             {Variable used in TextOut function}
    for B := 0 to (TheList.Count - 1) do
    begin
      ARecord := TheList.Items[B];
      Form1.PaintBox1.Canvas.TextOut(10, Y, IntToStr(ARecord^.I)); {Display I}
      Y := Y + 30;  {Increment Y Value again}
      Form1.PaintBox1.Canvas.TextOut(10, Y, ARecord^.C);  {Display C}
      Y := Y + 30;  {Increment Y Value}
    end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  if (MyList.IndexOf(ARecord1) = -1) then MyList.Add(ARecord1);
  DisplayTList(MyList);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  if (MyList.IndexOf(ARecord2) = -1) then MyList.Add(ARecord2);
  DisplayTList(MyList);
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  ARecord: PMyList;
  B: Integer;
begin
  if (MyList.Count <> 0) then
  begin
    for B := (MyList.Count - 1) downto 0 do
    begin
      ARecord := MyList.Items[B];
      MyList.Remove(ARecord);
    end;
  end;
  Form1.PaintBox1.Repaint;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyList := TList.Create;
  New(ARecord1);
  ARecord1^.I := 100;
  ARecord1^.C := 'Z';
  New(ARecord2);
  ARecord2^.I := 200;
  ARecord2^.C := 'X';
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Dispose(ARecord1);
  Dispose(ARecord2);
  MyList.Free;
end;
```

See Also

Delete (see page 2874)

Items (see page 2898)

23.1.37.3.14 TList.Insert Method

Adds an object to the Items (see page 2898) array at the position specified by Index.

Pascal

```
procedure Insert(Index: Integer; Item: Pointer);
```

C++

```
__fastcall Insert(int Index, void * Item);
```

Description

Call Insert to add Item to the middle of the Items (see page 2898) array. The Index parameter is a zero-based index, so the first position in the array has an index of 0. Insert adds the item at the indicated position, shifting the item that previously occupied that position, and all subsequent items, up. Insert increments Count (see page 2896) and, if necessary, allocates memory by increasing the value of Capacity (see page 2895).

To replace a nil (Delphi) or NULL (C++) pointer in the array with a new item, without growing the Items (see page 2898) array, set the Items (see page 2898) property.

See Also

Add (see page 2871)

Capacity (see page 2895)

Count (see page 2896)

Delete (see page 2874)

Items (see page 2898)

Remove (see page 2887)

23.1.37.3.15 TList.Last Method

Returns Items (see page 2898)[Count (see page 2896) - 1].

Pascal

```
function Last: Pointer;
```

C++

```
__fastcall void * Last();
```

Description

Call Last to retrieve the last pointer in the Items (see page 2898) array.

C++ Examples:

```
/*
This example inserts two records into a list object and
displays the contents of the last record in the list on the form:
*/
typedef struct AList
{
```

```
  int I;
  char C;
} TAList;

typedef AList* PAList;

TList *MyList;
PAList ARecord1, ARecord2;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PAList ARecord = (AList *)(MyList->Last());
  PaintBox1->Canvas->TextOut(10, 10, IntToStr(ARecord->I)); // Display I
  PaintBox1->Canvas->TextOut(10, 40, ARecord->C); // Display C
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Refresh();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyList = new TList;
  ARecord1 = new TAList;
  ARecord1->I = 100;
  ARecord1->C = 'Z';
  MyList->Add(ARecord1);
  ARecord2 = new TAList;
  ARecord2->I = 200;
  ARecord2->C = 'X';
  MyList->Add(ARecord2);
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  // Clean up - must free memory for the items as well as the list
  for (int i = 0; i < MyList->Count; i++)
  {
    PAList ARecord = (PAList) MyList->Items[i];
    delete ARecord;
  }
  delete MyList;
}
```

Delphi Examples:

```
{
This example inserts two records into a list object and
displays the contents of the last record in the list on the form:
}
procedure TForm1.Button1Click(Sender: TObject);
type
  PMyList = ^AList;
  AList = record
    I: Integer;
    C: Char;
  end;
var
  MyList: TList;
  ARecord: PMyList;
  I: integer;
begin
  MyList := TList.Create;
  New(ARecord);
  ARecord^.I := 100;
  ARecord^.C := 'Z';
```

```
    MyList.Add(ARecord); {Add integer 100 and character Z to list}
    New(ARecord);
    ARecord^.I := 200;
    ARecord^.C := 'X';
    MyList.Add(ARecord); {Add integer 200 and character X to list}
    ARecord := MyList.Last;
    Canvas.TextOut(10, 10, IntToStr(ARecord^.I)); {Display I}
    Canvas.TextOut(10, 40, ARecord^.C);  {Display C}
    { Cleanup: must free the list items as well as the list }
    for I := 0 to (MyList.Count - 1) do
    begin
      ARecord := MyList.Items[I];
      Dispose(ARecord);
    end;
    MyList.Free;
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    Refresh;
  end;
```

See Also

Count (see page 2896)

First (see page 2878)

Items (see page 2898)

23.1.37.3.16 TList.Move Method

Changes the position of an item in the Items (see page 2898) array.

Pascal

```
procedure Move(CurIndex: Integer; NewIndex: Integer);
```

C++

```
__fastcall Move(int CurIndex, int NewIndex);
```

Description

Call Move to move the item at the position CurIndex so that it occupies the position NewIndex. CurIndex and NewIndex are zero-based indexes into the Items (see page 2898) array.

See Also

Exchange (see page 2876)

IndexOf (see page 2879)

Items (see page 2898)

23.1.37.3.17 TList.Notify Method

Responds when items are added, removed, or extracted from the list.

Pascal

```
procedure Notify(Ptr: Pointer; Action: TListNotification); virtual;
```

C++

```
virtual __fastcall Notify(void * Ptr, TListNotification Action);
```

Description

Notify is called automatically when the items in the collection change.

Ptr points to item affected by the change.

Action indicates the change.

As implemented in TList (see page 2868), Notify does nothing. Descendant classes can override Notify in order to implement an appropriate response to the change notification.

23.1.37.3.18 TList.Pack Method

Deletes all nil (Delphi) or NULL (C++) items from the Items (see page 2898) array.

Pascal

```
procedure Pack;
```

C++

```
__fastcall Pack();
```

Description

Call Pack to move all non-nil (Delphi) or non-NULL (C++) items to the front of the Items (see page 2898) array and reduce the Count (see page 2896) property to the number of items actually used. Pack does not free up the memory used to store the nil (Delphi) or NULL (C++) pointers. To free up the memory for the unused entries removed by Pack, set the Capacity (see page 2895) property to the new value of Count (see page 2896).

C++ Examples:

```
/*
This example requires two edit box controls and a button on
the form. The code creates a list object and adds some
strings to it. The forth string in the list is a null string.
The code counts the number of strings in the list and
displays the number in the Edit1 control. The code then
packs the list, removing the nil string, and counts the
strings in the list again. The second count displays in the
Edit2 control:
*/
void __fastcall  DisplayTList(TList *TheList)
{
    // Now paint the items onto the paintbox}
    short Y = 10;             // Variable used in TextOut function
    for (byte B = 0; B < TheList->Count; B++)
    {
      AnsiString str = PChar(TheList->Items[B]);
      Form1->Canvas->TextOut(10, Y, str);
      Y = Y + 30;  // Increment Y Value again
    }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyList = new TList; // Create a list
  MyList->Add(PChar("A string")); // Add a string
  MyList->Add(PChar("")); // Add an empty string
  MyList->Add(PChar("A third string")); // Add a string
  MyList->Add(NULL);              // Add nil
  MyList->Add(PChar("A fifth string")); // Add a string
  MyList->Add(PChar("")); // Add another empty string
  MyList->Add(PChar("A seventh string")); // Add a string
  Edit1->Text = IntToStr(MyList->Count); // Put count into Edit1
```

```
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  MyList->Pack();                          // Pack the list.
  Edit2->Text = IntToStr(MyList->Count); // Put count into Edit2
  Repaint();
}

void __fastcall TForm1::FormPaint(TObject *Sender)
{
  DisplayTList(MyList);
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete MyList;                          // Free memory for list
}
```

Delphi Examples:

```
{
This example requires two edit box controls and a button on
the form. The code creates a list object and adds some
strings to it. The second string in the list is a nil string.
The code counts the number of strings in the list and
displays the number in the Edit1 control. The code then
packs the list, removing the nil string, and counts the
strings in the list again. The second count displays in the
Edit2 control:
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  MyList := TList.Create;              {Create a list }
  MyList.Add(PChar('A string')); {Add a string}
  MyList.Add(PChar('')); { Add an empty string }
  MyList.Add(PChar('A third string')); {Add a string}
  MyList.Add(nil);              {Add nil }
  MyList.Add(PChar('A fifth string')); {Add a string}
  MyList.Add(PChar('')); { Add another empty string }
  MyList.Add(PChar('A seventh string')); {Add a string}
  Edit1.Text := IntToStr(MyList.Count); {Put count into Edit1}
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  MyList.Pack;                         {Pack the list.}
  Edit2.Text := IntToStr(MyList.Count); {Put count into Edit2}
  Repaint;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  MyList.Free;                         {Free memory for list}
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
  DisplayTList(MyList);
end;
```

See Also

Capacity (see page 2895)

Count (see page 2896)

Delete (see page 2874)

Items (see page 2898)

23.1.37.3.19 TList.Put Method

Pascal

```
procedure Put(Index: Integer; Item: Pointer);
```

C++

```
__fastcall Put(int Index, void * Item);
```

Description

This is Put, a member of class TList.

23.1.37.3.20 TList.Remove Method

Deletes the first reference to the Item parameter from the Items (see page 2898) array.

Pascal

```
function Remove(Item: Pointer): Integer;
```

C++

```
__fastcall int Remove(void * Item);
```

Description

Call Remove to remove a specific item from the Items (see page 2898) array when its index is unknown. The value returned is the index of the item in the Items (see page 2898) array before it was removed. After an item is removed, all the items that follow it are moved up in index position and the Count (see page 2896) is reduced by one.

If the Items (see page 2898) array contains more than one copy of the pointer, only the first copy is deleted.

C++ Examples:

```
/*
The following code adds an object to MyList if it isn't
already in the list.
*/
typedef struct AList
{
  int I;
  char C;
} TAList;

typedef AList* PAList;

TList *MyList;
PAList ARecord1, ARecord2;

void __fastcall DisplayTList(TList *theList)
{
  PAList AStruct;
  // Go through the list, writing the elements to the
  // canvas of a paintbox component.
  int Y = 10; // position on canvas
  for (int i = 0; i < theList->Count; i++)
  {
    AStruct = (PAList) theList->Items[i];
```

```
      Form1->PaintBox1->Canvas->TextOut(10, Y, IntToStr(AStruct->I));
      Y += 30;  // Increment Y Value again
      Form1->PaintBox1->Canvas->TextOut(10, Y, AStruct->C);
      Y += 30;  //Increment Y Value
    }
  }

  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    if (MyList->IndexOf(ARecord1) == -1) MyList->Add(ARecord1);
    DisplayTList(MyList);
  }

  void __fastcall TForm1::Button2Click(TObject *Sender)
  {
    if (MyList->IndexOf(ARecord2) == -1) MyList->Add(ARecord2);
    DisplayTList(MyList);
  }

  void __fastcall TForm1::Button3Click(TObject *Sender)
  {
    if (MyList->Count != 0)
    {
      for (int B = MyList->Count - 1; B >= 0; B--)
      {
        PAList ARecord;
        ARecord = PAList(MyList->Items[B]);
        MyList->Remove(ARecord);
      }
    }
    Form1->PaintBox1->Repaint();
  }

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    MyList = new TList;
    ARecord1 = new TAList;
    ARecord1->I = 100;
    ARecord1->C = 'Z';
    ARecord2 = new TAList;
    ARecord2->I = 200;
    ARecord2->C = 'X';
  }

  void __fastcall TForm1::FormDestroy(TObject *Sender)
  {
    // Clean up - must free memory for the items as well as the list
    for (int i = 0; i < MyList->Count; i++)
    {
      PAList ARecord = (PAList) MyList->Items[i];
      delete ARecord;
    }
    delete MyList;

  }
  /*
  The following code adds a new object to a list in a list
  object and then removes it:
  */
  class TMyClass : public TComponent
  {
  __published:    // IDE-managed Components
  private:    // User declarations
  public:     // User declarations
    AnsiString MyString;
      __fastcall TMyClass(TComponent* Owner, AnsiString mystr);
```

```
};

__fastcall TMyClass::TMyClass(TComponent* Owner, AnsiString mystr)
    : TComponent(Owner)
{
  MyString = mystr;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TList *list = new TList();
  TMyClass *TheObject = new TMyClass(Form1, "This is an object.");
  try
  {
    list->Add(TheObject); // add AnsiString instance to list
    MessageDlg("The list has " + IntToStr(list->Count) + " objects",
               mtInformation, TMsgDlgButtons() << mbOK, 0);
    list->Remove(TheObject);
    MessageDlg("The list has " + IntToStr(list->Count) + " objects",
               mtInformation, TMsgDlgButtons() << mbOK, 0);
  }
  __finally
  {
    delete list;
  }
}
```

Delphi Examples:

```
{
The following code adds an object to MyList if it isn't
already in the list.
}
type
  PMyList = ^AList;
  AList = record
    I: Integer;
    C: Char;
  end;
var
  Form1: TForm1;
  MyList: TList;
  ARecord1, ARecord2: PMyList;

implementation

{$R *.dfm}

procedure DisplayTList(TheList: TList);
var
  ARecord: PMyList;
  B: Byte;
  Y: Word;
begin
    { Now paint the items onto the paintbox}
    Y := 10;             {Variable used in TextOut function}
    for B := 0 to (TheList.Count - 1) do
    begin
      ARecord := TheList.Items[B];
      Form1.PaintBox1.Canvas.TextOut(10, Y, IntToStr(ARecord^.I)); {Display I}
      Y := Y + 30;  {Increment Y Value again}
      Form1.PaintBox1.Canvas.TextOut(10, Y, ARecord^.C);  {Display C}
      Y := Y + 30;  {Increment Y Value}
    end;
end;

procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  if (MyList.IndexOf(ARecord1) = -1) then MyList.Add(ARecord1);
  DisplayTList(MyList);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  if (MyList.IndexOf(ARecord2) = -1) then MyList.Add(ARecord2);
  DisplayTList(MyList);
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  ARecord: PMyList;
  B: Integer;
begin
  if (MyList.Count <> 0) then
  begin
    for B := (MyList.Count - 1) downto 0 do
    begin
      ARecord := MyList.Items[B];
      MyList.Remove(ARecord);
    end;
  end;
  Form1.PaintBox1.Repaint;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyList := TList.Create;
  New(ARecord1);
  ARecord1^.I := 100;
  ARecord1^.C := 'Z';
  New(ARecord2);
  ARecord2^.I := 200;
  ARecord2^.C := 'X';
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Dispose(ARecord1);
  Dispose(ARecord2);
  MyList.Free;
end;
{
The following code adds a new object to a list in a list
object and then removes it:
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  TMyClass = class
    MyString: string;
    constructor Create(S: string);
  end;

var
  Form1: TForm1;
  MyList: TList;
```

```
implementation

{$R *.dfm}

constructor TMyClass.Create(S: string);
begin
  MyString := S;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  MyObject: TMyClass;
begin
  MessageDlg('The list starts with ' + IntToStr(MyList.Count) + ' objects',
    mtInformation, [mbOk], 0);
  try
    MyObject := TMyClass.Create('Semper Fidelis!');  { create a class instance }
    try
      MyList.Add(MyObject);      { add instance to list }
      MessageDlg('The list has ' + IntToStr(MyList.Count) + ' objects',
                 mtInformation, [mbOk], 0);
      MyList.Remove(MyObject);
      MessageDlg('The list has ' + IntToStr(MyList.Count) + ' objects',
                 mtInformation, [mbOk], 0);
    finally
      MyObject.Free;
    end;                  { don't forget to clean up! }
  finally
    MyList.Free;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  MyList := TList.Create;              {Create a list }
  MyList.Add(PChar('A string')); {Add a string}
  MyList.Add(PChar('')); { Add an empty string }
  MyList.Add(PChar('A third string')); {Add a string}
  MyList.Add(nil);              {Add nil }
  MyList.Add(PChar('A fifth string')); {Add a string}
  MyList.Add(PChar('')); { Add another empty string }
  MyList.Add(PChar('A seventh string')); {Add a string}
end;
```

See Also

Add (see page 2871)

Count (see page 2896)

Delete (see page 2874)

Insert (see page 2882)

Items (see page 2898)

23.1.37.3.21 TList.SetCapacity Method

Pascal

```
procedure SetCapacity(NewCapacity: Integer);
```

C++

```
__fastcall SetCapacity(int NewCapacity);
```

Description

This is SetCapacity, a member of class TList.

23.1.37.3.22 TList.SetCount Method

Pascal

```
procedure SetCount(NewCount: Integer);
```

C++

```
__fastcall SetCount(int NewCount);
```

Description

This is SetCount, a member of class TList.

23.1.37.3.23 TList.Sort Method

Performs a QuickSort on the list based on the comparison function Compare.

Pascal

```
procedure Sort(Compare: TListSortCompare);
```

C++

```
__fastcall Sort(TListSortCompare Compare);
```

Description

Call Sort to sort the items in the Items (see page 2898) array. Compare is a comparison function that indicates how the items are to be ordered.

C++ Examples:

```
/*
The following code sorts the objects in a list in
alphabetical order based on their names. It assumes that the
list contains only TMyClass references.  The CompareNames
function performs the comparisons between objects in the list.
The list is sorted when the user clicks a button.
*/
class TMyClass : public TObject
{
__published:    // IDE-managed Components
private:    // User declarations
public:     // User declarations
  AnsiString MyString;
    __fastcall TMyClass(AnsiString mystr);
};

__fastcall TMyClass::TMyClass(AnsiString mystr)
    : TObject()
{
  MyString = mystr;
}

void __fastcall  DisplayTList(TList *TheList)
{
    // Now paint the items onto the paintbox}
```

```
    short Y = 10;             // Variable used in TextOut function
    for (byte B = 0; B < TheList->Count; B++)
    {
      TMyClass *temp = (TMyClass *)(TheList->Items[B]);
      Form1->Canvas->TextOut(10, Y, temp->MyString);
      Y = Y + 30;  // Increment Y Value again
    }
}

int __fastcall CompareNames(void *Item1, void *Item2)
{
  return CompareText(((TMyClass *)Item1)->MyString, ((TMyClass *)Item2)->MyString);
}
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  MyList->Sort(CompareNames);
  Refresh();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyList = new TList;           // Create a list
  for (int I = 0; I < ComponentCount; I++)
  {
    TMyClass *MyObject = new TMyClass(Components[I]->Name);  // create a class instance
    MyList->Add(MyObject);      // add instance to list
  }
}

void __fastcall TForm1::FormPaint(TObject *Sender)
{
  DisplayTList(MyList);
}
```

Delphi Examples:

```
{
The following code sorts the objects in a list in
alphabetical order based on their names. It assumes that the
list contains only TMyClass references.  The CompareNames
function performs the comparisons between objects in the list.
The list is sorted when the user clicks a button.
}
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Button1: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormPaint(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  TMyClass = class
    MyString: string;
    constructor Create(S: string);
  end;

var
  Form1: TForm1;
  MyList: TList;
```

```
implementation

{$R *.dfm}

constructor TMyClass.Create(S: string);
begin
  inherited Create;
  MyString := S;
end;

procedure DisplayTList(TheList: TList);
var
  B: Byte;
  Y: Word;
  str: string;
  Temp: TMyClass;
begin
    { Now paint the items onto the paintbox}
    Y := 10;             {Variable used in TextOut function}
    for B := 0 to (TheList.Count - 1) do
    begin
      Temp:= TheList.Items[B];
      if (Temp is TMyClass) then
      begin
        str:= Temp.MyString;   // clear string
        Form1.Canvas.TextOut(10, Y, str);
        Y := Y + 30;  {Increment Y Value again}
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
  MyObject: TMyClass;
begin
  MyList := TList.Create;              {Create a list }
  for I := 0 to ComponentCount - 1 do
    begin
      MyObject := TMyClass.Create(Components[I].Name);  { create a class instance }
      MyList.Add(MyObject);      { add instance to list }
    end;
end;

function CompareNames(Item1, Item2: Pointer): Integer;
begin
  Result := CompareText(TMyClass(Item1).MyString, TMyClass(Item2).MyString);
//  MessageDlg('Compare ' + TMyClass(Item1).MyString + ' to ' + TMyClass(Item2).MyString,
//                  mtInformation, [mbOk], 0);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  MyList.Sort(@CompareNames);
  Refresh;
end;
```

See Also

Items (see page 2898)

23.1.37.4 TList Properties

The properties of the TList class are listed here.

Public Properties

	Name	Description
	Capacity (see page 2895)	Specifies the allocated size of the array of pointers maintained by the TList (see page 2868) object.
	Count (see page 2896)	Indicates the number of entries in the list that are in use.
	Items (see page 2898)	Lists the object references.
R	List (see page 2900)	Specifies the array of pointers that make up the Items (see page 2898) array.

23.1.37.4.1 TList.Capacity Property

Specifies the allocated size of the array of pointers maintained by the TList (see page 2868) object.

Pascal

```
property Capacity: Integer;
```

C++

```
__property int Capacity;
```

Description

Set Capacity to the number of pointers the list will need to contain. When setting the Capacity property, an EOutOfMemory exception occurs if there is not enough memory to expand the list to its new size.

Read Capacity to learn number of objects the list can hold without reallocating memory. Do not confuse Capacity with the Count (see page 2896) property, which is the number of entries in the list that are in use. The value of Capacity is always greater than or equal to the value of Count (see page 2896). When Capacity is greater than Count (see page 2896), the unused memory can be reclaimed by setting Capacity to Count (see page 2896).

When an object is added to a list that is already filled to capacity, the Capacity property is automatically increased. Setting Capacity before adding objects can reduce the number of memory reallocations and thereby improve performance. For example,

Note: Delphi example:

```
List.Clear;
List.Capacity := Count;
for I := 1 to Count do List.Add(...);
```

Note: C++ example:

```
List->Clear();
List->Capacity = Count;
for (int I = 0; I < Count; I++)
  List->Add(...);
```

The assignment to Capacity before the for loop ensures that each of the following Add (see page 2871) operations doesn't cause the list to be reallocated. Avoiding reallocations on the calls to Add (see page 2871) improves performance and ensures that the Add (see page 2871) operations never raise an exception.

See Also

Add (see page 2871)

Count (see page 2896)

Expand (see page 2877)

Pack (see page 2885)

23.1.37.4.2 TList.Count Property

Indicates the number of entries in the list that are in use.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read Count to determine the number of entries in the Items (see page 2898) array.

Increasing the size of Count will add the necessary number of nil (Delphi) or NULL (C++) pointers to the end of the Items (see page 2898) array. Decreasing the size of Count will remove the necessary number of entries from the end of the Items (see page 2898) array.

Count is not always the same as the number of objects referenced in the list. Some of the entries in the Items (see page 2898) array may contain nil (Delphi) or NULL (C++) pointers. To remove the nil (Delphi) or NULL (C++) pointers and set Count to the number of entries that contain references to objects, call the Pack (see page 2885) method.

C++ Examples:

```
/*
This example creates a list object and inserts two records
into it. The value of the record fields are written on a
paintbox:
*/

typedef struct AList
{
  int I;
  char C;
} TAList;

typedef TAList* PAList;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PAList AStruct;
  TList *MyList = new TList;
  // fill the TList
  AStruct = new TAList;
  AStruct->I = 100;
  AStruct->C = 'Z';
  MyList->Add(AStruct);
  AStruct = new TAList;
  AStruct->I = 200;
  AStruct->C = 'X';
  MyList->Add(AStruct);

  // Go through the list, writing the elements to the
  // canvas of a paintbox component.
  int Y = 10; // position on canvas
  for (int i = 0; i < MyList->Count; i++)
  {
```

```
    AStruct = (PAList) MyList->Items[i];
    PaintBox1->Canvas->TextOut(10, Y, IntToStr(AStruct->I));
    Y += 30;  // Increment Y Value again
    PaintBox1->Canvas->TextOut(10, Y, AStruct->C);
    Y += 30;  //Increment Y Value
  }

  // Clean up - must free memory for the items as well as the list
  for (int i = 0; i < MyList->Count; i++)
  {
    AStruct = (PAList) MyList->Items[i];
    delete AStruct;
  }
  delete MyList;

}
```

Delphi Examples:

```
{
This example creates a list object and inserts two records
into it. The value of the record fields are written on a
paintbox:
}
procedure TForm1.Button1Click(Sender: TObject);
type
  PMyList = ^AList;
  AList = record
    I: Integer;
    C: Char;
  end;
var
  MyList: TList;
  ARecord: PMyList;
  B: Byte;
  Y: Word;
begin
  MyList := TList.Create;
  try
    New(ARecord);
    ARecord^.I := 100;
    ARecord^.C := 'Z';
    MyList.Add(ARecord); {Add integer 100 and character Z to list}
    New(ARecord);
    ARecord^.I := 200;
    ARecord^.C := 'X';
    MyList.Add(ARecord); {Add integer 200 and character X to list}

    { Now paint the items onto the paintbox}
    Y := 10;             {Variable used in TextOut function}
    for B := 0 to (MyList.Count - 1) do
    begin
      ARecord := MyList.Items[B];
      Canvas.TextOut(10, Y, IntToStr(ARecord^.I)); {Display I}
      Y := Y + 30;  {Increment Y Value again}
      Canvas.TextOut(10, Y, ARecord^.C);  {Display C}
      Y := Y + 30;  {Increment Y Value}
    end;

    { Cleanup: must free the list items as well as the list }
   for B := 0 to (MyList.Count - 1) do
   begin
     ARecord := MyList.Items[B];
     Dispose(ARecord);
   end;
  finally
    MyList.Free;
```

```
  end;
end;
```

See Also

Capacity (see page 2895)

Items (see page 2898)

Pack (see page 2885)

23.1.37.4.3 TList.Items Property

Lists the object references.

Pascal

```
property Items [Index: Integer]: Pointer;
```

C++

```
__property void * Items[int Index];
```

Description

Use Items to obtain a pointer to a specific object in the array. The Index parameter indicates the index of the object, where 0 is the index of the first object, 1 is the index of the second object, and so on. Set Items to change the reference at a specific location.

Use Items with the Count (see page 2896) property to iterate through all of the objects in the list.

Not all of the entries in the Items array need to contain references to objects. Some of the entries may be nil (Delphi) or NULL (C++) pointers. To remove the nil (Delphi) or NULL (C++) pointers and reduce the size of the Items array to the number of objects, call the Pack (see page 2885) method.

Note: Items is the default property for TList (see page 2868). This means you can omit the property name. Thus, instead of

Note: MyList.Items[i]

Note: you can write

Note: MyList[i]

C++ Examples:

```
/*
This example creates a list object and inserts two records
into it. The value of the record fields are written on a
paintbox:
*/

typedef struct AList
{
  int I;
  char C;
} TAList;

typedef TAList* PAList;
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PAList AStruct;
  TList *MyList = new TList;
  // fill the TList
  AStruct = new TAList;
  AStruct->I = 100;
  AStruct->C = 'Z';
  MyList->Add(AStruct);
  AStruct = new TAList;
  AStruct->I = 200;
  AStruct->C = 'X';
  MyList->Add(AStruct);

  // Go through the list, writing the elements to the
  // canvas of a paintbox component.
  int Y = 10; // position on canvas
  for (int i = 0; i < MyList->Count; i++)
  {
    AStruct = (PAList) MyList->Items[i];
    PaintBox1->Canvas->TextOut(10, Y, IntToStr(AStruct->I));
    Y += 30;  // Increment Y Value again
    PaintBox1->Canvas->TextOut(10, Y, AStruct->C);
    Y += 30;  //Increment Y Value
  }

  // Clean up - must free memory for the items as well as the list
  for (int i = 0; i < MyList->Count; i++)
  {
    AStruct = (PAList) MyList->Items[i];
    delete AStruct;
  }
  delete MyList;

}
```

Delphi Examples:

```
{
This example creates a list object and inserts two records
into it. The value of the record fields are written on a
paintbox:
}
procedure TForm1.Button1Click(Sender: TObject);
type
  PMyList = ^AList;
  AList = record
    I: Integer;
    C: Char;
  end;
var
  MyList: TList;
  ARecord: PMyList;
  B: Byte;
  Y: Word;
begin
  MyList := TList.Create;
  try
    New(ARecord);
    ARecord^.I := 100;
    ARecord^.C := 'Z';
    MyList.Add(ARecord); {Add integer 100 and character Z to list}
    New(ARecord);
    ARecord^.I := 200;
    ARecord^.C := 'X';
    MyList.Add(ARecord); {Add integer 200 and character X to list}
```

```
      { Now paint the items onto the paintbox}
      Y := 10;             {Variable used in TextOut function}
      for B := 0 to (MyList.Count - 1) do
      begin
        ARecord := MyList.Items[B];
        Canvas.TextOut(10, Y, IntToStr(ARecord^.I)); {Display I}
        Y := Y + 30;  {Increment Y Value again}
        Canvas.TextOut(10, Y, ARecord^.C);  {Display C}
        Y := Y + 30;  {Increment Y Value}
      end;

      { Cleanup: must free the list items as well as the list }
     for B := 0 to (MyList.Count - 1) do
     begin
       ARecord := MyList.Items[B];
       Dispose(ARecord);
     end;
    finally
      MyList.Free;
    end;
  end;
```

See Also

Count (see page 2896)

List (see page 2900)

Pack (see page 2885)

operator_sb

23.1.37.4.4 TList.List Property

Specifies the array of pointers that make up the Items (see page 2898) array.

Pascal

```
property List: PPointerList;
```

C++

```
__property PPointerList List;
```

Description

Use List to gain direct access to the Items (see page 2898) array.

See Also

Items (see page 2898)

23.1.38 TListEnumerator Class

Enumerates through the objects in a TList (see page 2868).

Class Hierarchy

Classes.TListEnumerator

Pascal

```
TListEnumerator = class;
```

C++

```
class TListEnumerator;
```

File

Classes

Description

TListEnumerator provides an enumerator for TList (see page 2868) objects.

23.1.38.1 TListEnumerator Members

The following tables list the members exposed by TListEnumerator.

Public Methods

	Name	Description
	Create (see page 2901)	Creates a TList (see page 2868) enumerator.
	GetCurrent (see page 2901)	Gets the current object.
	MoveNext (see page 2902)	Moves the enumerator to the next TList (see page 2868) item in the collection.

Public Properties

	Name	Description
R	Current (see page 2902)	The current TList (see page 2868) object.

23.1.38.2 TListEnumerator.Create Constructor

Creates a TList (see page 2868) enumerator.

Pascal

```
constructor Create(AList: TList);
```

C++

```
__fastcall TListEnumerator(TList AList);
```

Description

Creates an instance of TListEnumerator (see page 2900).

23.1.38.3 TListEnumerator Methods

The methods of the TListEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 2901)	Gets the current object.
	MoveNext (see page 2902)	Moves the enumerator to the next TList (see page 2868) item in the collection.

23.1.38.3.1 TListEnumerator.GetCurrent Method

Gets the current object.

Pascal

```
function GetCurrent: Pointer;
```

C++

```
__fastcall void * GetCurrent();
```

Description

Use GetCurrent to get the current object in the TList (see page 2868) being enumerated.

23.1.38.3.2 TListEnumerator.MoveNext Method

Moves the enumerator to the next TList (see page 2868) item in the collection.

Pascal

```
function MoveNext: Boolean;
```

C++

```
__fastcall Boolean MoveNext();
```

Description

Use MoveNext to move the TListEnumerator (see page 2900) to the next object in the TList (see page 2868).

23.1.38.4 TListEnumerator Properties

The properties of the TListEnumerator class are listed here.

Public Properties

	Name	Description
R	Current (see page 2902)	The current TList (see page 2868) object.

23.1.38.4.1 TListEnumerator.Current Property

The current TList (see page 2868) object.

Pascal

```
property Current: Pointer;
```

C++

```
__property void * Current;
```

Description

Current contains the current object in the TList (see page 2868) that is being enumerated.

23.1.39 TMemoryStream Class

TMemoryStream is a stream that stores its data in dynamic memory.

Class Hierarchy

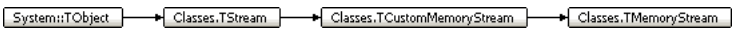

Pascal

```
TMemoryStream = class(TCustomMemoryStream);
```

C++

```
class TMemoryStream : public TCustomMemoryStream;
```

File

Classes

Description

Use TMemoryStream to store data in a dynamic memory buffer that is enhanced with file-like access capabilities. TMemoryStream provides the general I/O capabilities of a stream object while introducing methods and properties to manage a dynamic memory buffer.

Memory (see page 2823) streams are useful as intermediary objects that can hold information as well as read it from or write it to another storage medium. They provide a useful format for comparing the contents of streams, or for manipulating data that is stored in a less accessible medium.

C++ Examples:

```
/*
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
*/
class MyScrollBar : public TScrollBar
{
__published:    // IDE-managed Components
private:    // User declarations
public:     // User declarations
    __fastcall MyScrollBar(TComponent* Owner);
};

__fastcall MyScrollBar::MyScrollBar(TComponent* Owner)
    : TScrollBar(Owner)
{
}

void RegisterClassesWithStreamingSystem(void)
{
  // Make sure that as part of the startup
  // code our streaming classes are registered
  // with the streaming system.

  #pragma startup RegisterClassesWithStreamingSystem

  Classes::RegisterClass(__classid(MyScrollBar));
}

MyScrollBar *ScrollBar1;

AnsiString __fastcall ComponentToString(TComponent *c)
{
 AnsiString as;
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);

 try
 {
    pms->WriteComponent(c);
    pms->Seek(0, soFromBeginning);
    ObjectBinaryToText(pms, pss);
    pss->Seek(0, soFromBeginning);
```

```
    as = pss->DataString;
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 delete pms;
 delete pss;
 return as;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Memo1->Text = ComponentToString(ScrollBar1);
}

TComponent *__fastcall StringToComponent(AnsiString as)

{
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);
 TComponent *pc;

 try
 {
    ObjectTextToBinary(pss, pms);
    pms->Seek(0, soFromBeginning);
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 pc = pms->ReadComponent(NULL);
 delete pms;
 delete pss;
 return pc;
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  TComponent *temp = StringToComponent(Memo1->Text);
  ScrollBar1->Free();

  ScrollBar1 = dynamic_cast<MyScrollBar *>(temp);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ScrollBar1 = new MyScrollBar(Form1);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
  ScrollBar1->Top = 48;
  ScrollBar1->Left = 250;
  ScrollBar1->Name = "Ricksbar";
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete ScrollBar1;
}
```

Delphi Examples:

```
{
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
}
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  MyScrollBar = class(TScrollBar)
  end;

var
  Form1: TForm1;
  ScrollBar1: MyScrollBar;

implementation

{$R *.dfm}

function ComponentToStringProc(Component: TComponent): string;
var
  BinStream:TMemoryStream;
  StrStream: TStringStream;
  s: string;
begin
  BinStream := TMemoryStream.Create;
  try
    StrStream := TStringStream.Create(s);
    try
      BinStream.WriteComponent(Component);
      BinStream.Seek(0, soFromBeginning);
      ObjectBinaryToText(BinStream, StrStream);
      StrStream.Seek(0, soFromBeginning);
      Result:= StrStream.DataString;
    finally
      StrStream.Free;
    end;
  finally
    BinStream.Free
  end;
end;

function StringToComponentProc(Value: string): TComponent;
var
  StrStream:TStringStream;
  BinStream: TMemoryStream;
begin
  StrStream := TStringStream.Create(Value);
  try
    BinStream := TMemoryStream.Create;
    try
      ObjectTextToBinary(StrStream, BinStream);
      BinStream.Seek(0, soFromBeginning);
      Result:= BinStream.ReadComponent(nil);
    finally
```

```
      BinStream.Free;
    end;
  finally
    StrStream.Free;
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Memo1.Text:= ComponentToStringProc(ScrollBar1);
end;

// Edit the scrollbar parameters in the memo, click this
// button and see the scrollbar change.
procedure TForm1.Button2Click(Sender: TObject);
begin
  ScrollBar1.Free;
  ScrollBar1:= (StringToComponentProc(Memo1.Text) as MyScrollBar);
  ScrollBar1.Parent:= Form1;
  ScrollBar1.Visible:= TRUE;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ScrollBar1:= MyScrollBar.Create(Form1);
  ScrollBar1.Parent:= Form1;
  ScrollBar1.Visible:= TRUE;
  ScrollBar1.Top:= 48;
  ScrollBar1.Left:= 250;
  ScrollBar1.Name:= 'Ricksbar';
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  ScrollBar1.Free;
end;

initialization

RegisterClass(MyScrollBar);
```

See Also

TFiler (see page 2834)

TStringStream (see page 3088)

23.1.39.1 TMemoryStream Members

The following tables list the members exposed by TMemoryStream.

Protected Methods

	Name	Description
	GetSize (see page 2982)	This is GetSize, a member of class TStream.
	Realloc (see page 2910)	This is Realloc, a member of class TMemoryStream.
	SetPointer (see page 2823)	Replaces the memory buffer associated with the memory stream.
	SetSize (see page 2989)	This is the overview for the SetSize method overload.

Protected Properties

	Name	Description
	Capacity (see page 2912)	This is Capacity, a member of class TMemoryStream.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2909)	Frees the memory buffer, discarding all data associated with the memory stream.
	CopyFrom (see page 2979)	Copies a specified number of bytes from one stream to another.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2908)	Destroys an instance of TMemoryStream (see page 2902).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FixupResourceHeader (see page 2981)	Patches the resource header for a resource that has been written to the stream.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LoadFromFile (see page 2909)	Loads the entire contents of a file into the memory buffer.
	LoadFromStream (see page 2909)	Loads the entire contents of a stream into the memory buffer.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Read (see page 2817)	Reads up to Count bytes from the memory stream into Buffer and advances the current position of the stream by the number of bytes read.
A	Read (see page 2982)	Introduces an abstract or pure virtual method responsible for reading from the stream.
	ReadBuffer (see page 2983)	Reads bytes from the stream into Buffer.
	ReadComponent (see page 2983)	Initiates streaming of components and their properties.
	ReadComponentRes (see page 2987)	Reads components and their properties from a stream in a resource file format.
	ReadResHeader (see page 2987)	Reads a resource-file header from the stream.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 2818)	Writes the entire contents of the memory stream to the file with a given file name.
	SaveToStream (see page 2818)	Writes the entire contents of the memory stream to the stream object specified by Stream.
V	Seek (see page 2819)	Moves the current position of the stream by Offset bytes, relative to the origin specified by Origin.
V	Seek (see page 2988)	This is the overview for the Seek method overload.
V	SetSize (see page 2910)	Sets the Size (see page 2997) property of the memory stream.
V	Write (see page 2911)	Writes Count bytes from Buffer to the current position in the memory buffer and updates the current position by Count bytes.
	WriteBuffer (see page 2990)	Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.

	WriteComponent (see page 2990)	Initiates the writing of components and their properties to a stream.
	WriteComponentRes (see page 2994)	Writes components and their properties to a stream from a resource file format.
	WriteDescendent (see page 2994)	Streams components and their properties in inherited forms.
	WriteDescendentRes (see page 2995)	Streams components and their properties in inherited forms.
	WriteResourceHeader (see page 2995)	Writes a resource-file header to the stream.

Public Properties

	Name	Description
R	Memory (see page 2823)	Provides direct access to the memory pool allocated for the memory stream.
	Position (see page 2996)	Indicates the current offset into the stream for reading and writing.
	Size (see page 2997)	Indicates the size in bytes of the stream.

23.1.39.2 TMemoryStream.Destroy Destructor

Destroys an instance of TMemoryStream (see page 2902).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMemoryStream();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TMemoryStream (see page 2902) reference is not nil, and only then calls Destroy.

Destroy a TMemoryStream (see page 2902) object when it is no longer needed to store or write data. Destroy calls Clear (see page 2909) to free the memory buffer before calling the inherited destructor.

See Also

Clear (see page 2909)

Free (see page 22300)

23.1.39.3 TMemoryStream Methods

The methods of the TMemoryStream class are listed here.

Protected Methods

	Name	Description
V	Realloc (see page 2910)	This is Realloc, a member of class TMemoryStream.

Public Methods

	Name	Description
	Clear (see page 2909)	Frees the memory buffer, discarding all data associated with the memory stream.
	LoadFromFile (see page 2909)	Loads the entire contents of a file into the memory buffer.
	LoadFromStream (see page 2909)	Loads the entire contents of a stream into the memory buffer.
V	SetSize (see page 2910)	Sets the Size (see page 2997) property of the memory stream.
V	Write (see page 2911)	Writes Count bytes from Buffer to the current position in the memory buffer and updates the current position by Count bytes.

23.1.39.3.1 TMemoryStream.Clear Method

Frees the memory buffer, discarding all data associated with the memory stream.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Use Clear to empty the memory buffer for the memory stream and free all associated memory. In addition to freeing the memory associated with the memory buffer, Clear

Sets the Memory (see page 2823) property to nil (Delphi) or NULL (C++).

Sets the Position (see page 2996) property to 0.

Sets the Size (see page 2997) property to 0.

See Also

Memory (see page 2823)

Position (see page 2996)

Size (see page 2997)

23.1.39.3.2 TMemoryStream.LoadFromFile Method

Loads the entire contents of a file into the memory buffer.

Pascal

```
procedure LoadFromFile(const FileName: string);
```

C++

```
__fastcall LoadFromFile(const AnsiString FileName);
```

Description

Use LoadFromFile to fill the memory stream with the contents of a file. Pass the name of the file as the FileName parameter. LoadFromFile allows an application to read the contents of a file into the memory stream without having to explicitly create and free a file stream object.

LoadFromFile reallocates the memory buffer so that the contents of the file will exactly fit. It sets the Size (see page 2997) property accordingly, and then reads the entire contents of the file into the memory buffer. Thus, LoadFromFile will discard any pre-existing data stored in the memory stream.

See Also

LoadFromStream (see page 2909)

Memory (see page 2823)

Size (see page 2997)

23.1.39.3.3 TMemoryStream.LoadFromStream Method

Loads the entire contents of a stream into the memory buffer.

Pascal

```
procedure LoadFromStream(Stream: TStream);
```

C++

```
__fastcall LoadFromStream(TStream Stream);
```

Description

Use LoadFromStream to fill the memory stream with the contents of the stream specified by the Stream parameter. LoadFromStream always sets the Position (see page 2996) of the source stream to 0, before streaming in the number of bytes indicated by the source stream's Size (see page 2997) property.

LoadFromStream reallocates the memory buffer so that the contents of the source stream will exactly fit. It sets the Size (see page 2997) property accordingly, and then reads the entire contents of the source stream into the memory buffer. Thus, LoadFromStream will discard any pre-existing data stored in the memory stream.

If the source stream is a TFileStream (see page 2841) object, LoadFromStream does the same thing as LoadFromFile (see page 2909), except that the application must create and free the TFileStream (see page 2841) object. LoadFromStream also allows applications to fill a memory stream object from other types of stream objects.

See Also

LoadFromFile (see page 2909)

Memory (see page 2823)

TFileStream (see page 2841)

Position (see page 2996)

Size (see page 2997)

23.1.39.3.4 TMemoryStream.Realloc Method

Pascal

```
function Realloc(var NewCapacity: Longint): Pointer; virtual;
```

C++

```
virtual __fastcall void * Realloc(Longint NewCapacity);
```

Description

This is Realloc, a member of class TMemoryStream.

23.1.39.3.5 TMemoryStream.SetSize Method

Sets the Size (see page 2997) property of the memory stream.

Pascal

```
procedure SetSize(NewSize: Longint); override;
```

C++

```
virtual __fastcall SetSize(Longint NewSize);
```

Description

Use SetSize to set the Size (see page 2997) of a memory stream before filling it with data. SetSize allocates the memory buffer to hold NewSize bytes, preserving as much of the existing data as possible.

Use SetSize before filling the memory buffer with data from various sources, or from a portion of another stream. If the intended

contents of the memory stream is exactly the same as the contents of another stream or file, use LoadFromStream (see page 2909) or LoadFromFile (see page 2909) instead.

See Also

LoadFromFile (see page 2909)

LoadFromStream (see page 2909)

Memory (see page 2823)

Size (see page 2997)

23.1.39.3.6 TMemoryStream.Write Method

Writes Count bytes from Buffer to the current position in the memory buffer and updates the current position by Count bytes.

Pascal

```
function Write(const Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  Buffer, Longint Count);
```

Description

Use Write to insert Count bytes into the memory buffer of the memory stream, starting at the current position. Write will increase the size of the memory buffer, if necessary, to accommodate the data being written in. If the current position is not the end of the memory buffer, Write will overwrite the data following the current position.

Write updates the Size (see page 2997) property to Position (see page 2996) + Count, and sets the Position (see page 2996) property to the new value of Size (see page 2997). Thus, any data that was stored in the memory stream in the Count bytes after the current position is lost when calling Write.

Write always writes the Count bytes in the Buffer, unless there is a memory failure. Thus, for TMemoryStream (see page 2902), Write is equivalent to the WriteBuffer (see page 2990) method.

All other data-writing methods of a memory stream (WriteBuffer (see page 2990), WriteComponent (see page 2990)) call Write to do the actual writing.

See Also

Memory (see page 2823)

Position (see page 2996)

Size (see page 2997)

WriteBuffer (see page 2990)

WriteComponent (see page 2990)

23.1.39.4 TMemoryStream Properties

The properties of the TMemoryStream class are listed here.

Protected Properties

	Name	Description
	Capacity (see page 2912)	This is Capacity, a member of class TMemoryStream.

23.1.39.4.1 TMemoryStream.Capacity Property

Pascal

```
property Capacity: Longint;
```

C++

```
__property Longint Capacity;
```

Description

This is Capacity, a member of class TMemoryStream.

23.1.40 TOwnedCollection Class

TOwnedCollection is a collection that maintains information about its owner.

Class Hierarchy

Pascal

```
TOwnedCollection = class(TCollection);
```

C++

```
class TOwnedCollection : public TCollection;
```

File

Classes

Description

By maintaining information about its owner, TOwnedCollection lets the Object Inspector display the collection's name at design time.

Note: This information is not part of TOwnedCollection's public interface.

New collection classes should descend from TOwnedCollection. By descending from TOwnedCollection, the derived class does not need to add anything in order to appear in the Object Inspector. Classes (see page 2685) that descend directly from TCollection (see page 2731) must implement a GetOwner (see page 2915) method if they are to appear in the Object Inspector.

See Also

TCollection (see page 2731)

TCollectionItem (see page 2755)

TPersistent (see page 2928)

TPersistent (see page 2928)

23.1.40.1 TOwnedCollection Members

The following tables list the members exposed by TOwnedCollection.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2915)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 2736)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 2914)	Creates and initializes a TOwnedCollection (see page 2912) instance.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 2751)	Lists the items in the collection.

23.1.40.2 TOwnedCollection.Create Constructor

Creates and initializes a TOwnedCollection (see page 2912) instance.

Pascal

```
constructor Create(AOwner: TPersistent; ItemClass: TCollectionItemClass);
```

C++

```
__fastcall TOwnedCollection(TPersistent * AOwner, TCollectionItemClass ItemClass);
```

Description

The Create method takes two parameters: the name of an instance object descended from TPersistent (see page 2928) and the name of a TCollectionItem (see page 2755) descendant class. The first parameter is the owner of the TOwnedCollection (see page 2912) instance. The second parameter determines the class of the items created by the Add (see page 2736) method.

See Also

Add (see page 2736)

23.1.40.3 TOwnedCollection Methods

The methods of the TOwnedCollection class are listed here.

Protected Methods

	Name	Description
	GetOwner (see page 2915)	Returns the owner of an object.

23.1.40.3.1 TOwnedCollection.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

23.1.41 TParser Class

TParser allows DFM files to be parsed.

Class Hierarchy

Pascal

```
TParser = class(TObject);
```

C++

```
class TParser : public TObject;
```

File

Classes

Description

The TParser is principally used internally in RAD Studio to parse (tokenise) VCL Form files (which have a .dfm extension on Win32, and .nfm on .NET). However, it can potentially be used as a more general parser.

A VCL form file contains the description of the properties of the form and the components it owns. Each form file represents a single form, which usually corresponds to a window or dialog box in an application.

The parsing dissects the form text into a set of symbol strings, and value strings. Values may be string representations of strings, widestrings, integers, or floating point numbers.

Navigation is performed by repeatedly using the NextToken (see page 2922) method to retrieve a sequence of tokens (symobols and values). The value of each retrieved token is obtained using the TParser methods :

Method	Returns	Token (see page 2926)(s)
TokenFloat (see page 2923)	Float	toFloat (see page 3322)
TokenInteger	Integer	toInteger (see page 3322)
TokenString (see page 2924)	String	toFloat (see page 3322), toInteger (see page 3322), toString (see page 3323), toSymbol (see page 3324)
TokenWideString (see page 2925)	WideString	toWString (see page 3324)

toEOF (see page 3321) is the token returned when the form navigation is complete.

See Also

TForm (see page 13992)

23.1.41.1 TParser Members

The following tables list the members exposed by TParser.

Protected Methods

	Name	Description
	GetLinePos (see page 2921)	This is GetLinePos, a member of class TParser.

Public Events

	Name	Description
	OnError (see page 2927)	Occurs when a parse error is encountered.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckToken (see page 2919)	Checks if the current token in the file is a specific type.
	CheckTokenSymbol (see page 2919)	Checks if the current token in the file is a symbol with a specific value.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2917)	Creates a TParser (see page 2915) class instance.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2917)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Error (see page 2920)	Raise an EParseError error.
	ErrorFmt (see page 2920)	Raise an EParseError error with a specified error message.
	ErrorStr (see page 2921)	Raise an EParseError error with a specific message.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	HexToBinary (see page 2921)	Converts a sequence of hex values into binary.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextToken (see page 2922)	Moves to the next token in the file.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SourcePos (see page 2922)	Returns the position in the file being parsed.
	TokenComponentIdent (see page 2923)	Checks if the current token in the file is a valid component symbol.
	TokenFloat (see page 2923)	Returns a toFloat (see page 3322) token as a float.
	TokenInt (see page 2923)	Returns a toInteger (see page 3322) token as a float.
	TokenString (see page 2924)	Returns a string representation of the current token.
	TokenSymbolIs (see page 2924)	Checks if the next token in the file is a symbol with a specific value.
	TokenWideString (see page 2925)	Returns a toWString (see page 3324) token as a wide string value.

Public Properties

	Name	Description
R	FloatType (see page 2925)	Gives the type of floating pointer number.
R	LinePos (see page 2926)	Gives the position in the line currently being parsed.
R	SourceLine (see page 2926)	Gives the number of the line currently being parsed.
R	Token (see page 2926)	Gives the type of token at the current file position.

23.1.41.2 TParser.Create Constructor

Creates a TParser (see page 2915) class instance.

Pascal

```
constructor Create(Stream: TStream; AOnError: TParserErrorEvent = nil);
```

C++

```
__fastcall TParser(TStream Stream, TParserErrorEvent AOnError = nil);
```

Description

Call Create to instantiate a TParser (see page 2915) object at runtime.

The file to be parsed is defined by **Stream**. **AOnError** optionally defines an object method to be called when a parsing error is encountered. After this method returns, an Exception is raised anyway.

23.1.41.3 TParser.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TParser();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

23.1.41.4 TParser Methods

The methods of the TParser class are listed here.

Protected Methods

	Name	Description
	GetLinePos (see page 2921)	This is GetLinePos, a member of class TParser.

Public Methods

	Name	Description
	CheckToken (see page 2919)	Checks if the current token in the file is a specific type.
	CheckTokenSymbol (see page 2919)	Checks if the current token in the file is a symbol with a specific value.
	Error (see page 2920)	Raise an EParseError error.
	ErrorFmt (see page 2920)	Raise an EParseError error with a specified error message.
	ErrorStr (see page 2921)	Raise an EParseError error with a specific message.
	HexToBinary (see page 2921)	Converts a sequence of hex values into binary.
	NextToken (see page 2922)	Moves to the next token in the file.
	SourcePos (see page 2922)	Returns the position in the file being parsed.
	TokenComponentIdent (see page 2923)	Checks if the current token in the file is a valid component symbol.
	TokenFloat (see page 2923)	Returns a toFloat (see page 3322) token as a float.
	TokenInt (see page 2923)	Returns a toInteger (see page 3322) token as a float.
	TokenString (see page 2924)	Returns a string representation of the current token.
	TokenSymbolIs (see page 2924)	Checks if the next token in the file is a symbol with a specific value.
	TokenWideString (see page 2925)	Returns a toWString (see page 3324) token as a wide string value.

23.1.41.4.1 TParser.CheckToken Method

Checks if the current token in the file is a specific type.

Pascal

```
procedure CheckToken(T: Char);
```

C++

```
__fastcall CheckToken(Char T);
```

Description

Call CheckToken to check if the current token in the file is a specific type. If the token is not the specified type, then the EParseError exception is raised, otherwise, processing continues. You can respond to this error by specifying the OnError (see page 2927) event.

You pass the token type in the **T** value, passing one of the following values :

T value	Meaning
toEOF (see page 3321)	End of file has been reached
toFloat (see page 3322)	The next token is a floating (decimal) pointing value
toInteger (see page 3322)	The next token is an integer value
toString (see page 3323)	The next token is a text value
toSymbol (see page 3324)	The next token is a symbol
toWString (see page 3324)	The next token is a wide string value

See Also

CheckTokenSymbol (see page 2919)

EParseError

OnError

toEOF (see page 3321)

toFloat (see page 3322)

toInteger (see page 3322)

toString (see page 3323)

toSymbol (see page 3324)

toWString (see page 3324)

23.1.41.4.2 TParser.CheckTokenSymbol Method

Checks if the current token in the file is a symbol with a specific value.

Pascal

```
procedure CheckTokenSymbol(const S: string);
```

C++

```
__fastcall CheckTokenSymbol(const AnsiString S);
```

Description

Call CheckTokenSymbol to check if the current token in the file is symbol named **S**. If the token is not a symbol, or not a symbol with the specified value, then the EParseError exception is raised, otherwise, processing continues. You can respond to this error by specifying the OnError (see page 2927) event.

Note: Use TokenSymbolIs (see page 2924) if you want to perform this check without raising an exception

See Also

CheckToken (see page 2919)

EParseError

OnError

TokenSymbolIs (see page 2924)

toSymbol (see page 3324)

23.1.41.4.3 TParser.Error Method

Raise an EParseError error.

Pascal

```
procedure Error(const Ident: string);
```

C++

```
__fastcall Error(const AnsiString Ident);
```

Description

Call Error to force the raising of an EParseError exception. Use this to simulate an error encountered in the parsing of a file.

See Also

EParseError

ErrorFmt

ErrorStr

OnError

23.1.41.4.4 TParser.ErrorFmt Method

Raise an EParseError error with a specified error message.

Pascal

```
procedure ErrorFmt(const Ident: string; const Args: array of const);
```

C++

```
__fastcall ErrorFmt(const AnsiString Ident, const array of const Args);
```

Description

Call ErrorFmt to force the raising of an EParseError exception. **Ident** specifies the exception error message string. **Args** specify an array of constants that can be incorporated into the error message string - the Format routine is called to replace placeholder characters in the error message string with these values.

Use this to simulate an error encountered in the parsing of a file.

See Also

EParseError

ErrorFmt

ErrorStr

Format (see page 22897)

OnError

23.1.41.4.5 TParser.ErrorStr Method

Raise an EParseError error with a specific message.

Pascal

```
procedure ErrorStr(const Message: string);
```

C++

```
__fastcall ErrorStr(const AnsiString Message);
```

Description

Call ErrorStr to force the raising of an EParseError exception with an error message specified by **S**.

Use this to simulate an error encountered in the parsing of a file.

See Also

EParseError

Error

ErrorFmt

OnError

23.1.41.4.6 TParser.GetLinePos Method

Pascal

```
function GetLinePos: Integer;
```

C++

```
__fastcall int GetLinePos();
```

Description

This is GetLinePos, a member of class TParser.

23.1.41.4.7 TParser.HexToBinary Method

Converts a sequence of hex values into binary.

Pascal

```
procedure HexToBinary(Stream: TStream);
```

C++

```
__fastcall HexToBinary(TStream Stream);
```

Description

Call HexToBinary to convert a hex string value in a file to its binary equivalent. **Stream** specifies a writeable stream for the binary values to be written to.

It is principally used internally by TParser (see page 2915)

23.1.41.4.8 TParser.NextToken Method

Moves to the next token in the file.

Pascal

```
function NextToken: Char;
```

C++

```
__fastcall Char NextToken();
```

Description

Call NextToken to move to the next token in the file being parsed. This is normally called in a loop as the file is parsed, terminating when toEOF (see page 3321) is encountered. Use Token (see page 2926) to retrieve the type of the current token.

See Also

toEOF (see page 3321)

Token (see page 2926)

TokenFloat

TokenInt

TokenString

TokenSymbolIs

TokenSymbol

23.1.41.4.9 TParser.SourcePos Method

Returns the position in the file being parsed.

Pascal

```
function SourcePos: Longint;
```

C++

```
__fastcall Longint SourcePos();
```

Description

Call SourcePos to get the character position in the file being parsed..

See Also

LinePos (see page 2926)

SourceLine (see page 2926)

23.1.41.4.10 TParser.TokenComponentIdent Method

Checks if the current token in the file is a valid component symbol.

Pascal

```
function TokenComponentIdent: string;
```

C++

```
__fastcall AnsiString TokenComponentIdent();
```

Description

Call TokenComponentIdent to check if the current token in the file is symbol that legitimately represents a component. Namely, that it must start with a letter or number, and only contain alphanumeric and underscore characters. If it is valid, the string value of the token is returned. If not, a EParseError exception is raised.

See Also

EParseError

OnError

toSymbol (see page 3324)

23.1.41.4.11 TParser.TokenFloat Method

Returns a toFloat (see page 3322) token as a float.

Pascal

```
function TokenFloat: Extended;
```

C++

```
__fastcall Extended TokenFloat();
```

Description

Call TokenFloat to retrieve the current token value as a floating point number value. An exception will be raised of the token string value fails to convert to a floating point value.

See Also

EParseError

toFloat (see page 3322)

TokenInt (see page 2923)

TokenString (see page 2924)

TokenWideString (see page 2925)

23.1.41.4.12 TParser.TokenInt Method

Returns a toInteger (see page 3322) token as a float.

Pascal

```
function TokenInt: Int64;
```

C++

```
__fastcall Int64 TokenInt();
```

Description

Call TokenInt to retrieve the current token value as an integer number value. An exception will be raised of the token string value fails to convert to an integer value.

See Also

EParseError

TokenFloat (see page 2923)

TokenString (see page 2924)

TokenWideString (see page 2925)

toInteger (see page 3322)

23.1.41.4.13 TParser.TokenString Method

Returns a string representation of the current token.

Pascal

```
function TokenString: string;
```

C++

```
__fastcall AnsiString TokenString();
```

Description

Call TokenString to retrieve the current token value as a string, regardless of its type.

See Also

EParseError

TokenFloat (see page 2923)

TokenInt (see page 2923)

TokenWideString (see page 2925)

23.1.41.4.14 TParser.TokenSymbolIs Method

Checks if the next token in the file is a symbol with a specific value.

Pascal

```
function TokenSymbolIs(const S: string): Boolean;
```

C++

```
__fastcall Boolean TokenSymbolIs(const AnsiString S);
```

Description

Call TokenSymbolIs to check if the current token in the file is symbol named **S**. If the token is not a symbol, or not a symbol with the specified value, then false is returned, otherwise, true is returned.

See Also

CheckToken (see page 2919)

CheckTokenSymbol (see page 2919)

toSymbol (see page 3324)

23.1.41.4.15 TParser.TokenWideString Method

Returns a toWString (see page 3324) token as a wide string value.

Pascal

```
function TokenWideString: WideString;
```

C++

```
__fastcall BSTR TokenWideString();
```

Description

Call TokenWideString to retrieve the current token value as a wide string value.

See Also

EParseError

TokenFloat (see page 2923)

TokenInt (see page 2923)

TokenString (see page 2924)

toWString (see page 3324)

23.1.41.5 TParser Properties

The properties of the TParser class are listed here.

Public Properties

	Name	Description
R	FloatType (see page 2925)	Gives the type of floating pointer number.
R	LinePos (see page 2926)	Gives the position in the line currently being parsed.
R	SourceLine (see page 2926)	Gives the number of the line currently being parsed.
R	Token (see page 2926)	Gives the type of token at the current file position.

23.1.41.5.1 TParser.FloatType Property

Gives the type of floating pointer number.

Pascal

```
property FloatType: Char;
```

C++

```
__property Char FloatType;
```

Description

This read only property gives the type of floating point number for the current token. If the file value read for this token ends in one of : 'c', 'C', 'd', 'D', 's', 'S', then this is the value returned. Otherwise, #0 is returned.

See Also

TokenFloat (see page 2923)

toFloat (see page 3322)

23.1.41.5.2 TParser.LinePos Property

Gives the position in the line currently being parsed.

Pascal

```
property LinePos: Integer;
```

C++

```
__property int LinePos;
```

Description

This read only property gives the character position in the current line being parsed. The position in the file is determined by the NextToken (see page 2922) method.

See Also

NextToken (see page 2922)

SourceLine (see page 2926)

SourcePos (see page 2922)

23.1.41.5.3 TParser.SourceLine Property

Gives the number of the line currently being parsed.

Pascal

```
property SourceLine: Integer;
```

C++

```
__property int SourceLine;
```

Description

This read only property gives the line number of the current line being parsed. Lines start at 0. The position in the file is determined by the NextToken (see page 2922) method.

See Also

LinePos (see page 2926)

NextToken (see page 2922)

SourcePos (see page 2922)

23.1.41.5.4 TParser.Token Property

Gives the type of token at the current file position.

Pascal

```
property Token: Char;
```

C++

```
__property Char Token;
```

Description

Token is a read only property that describes the type of token at current file parse position. The NextToken (see page 2922) method is used to navigate through the file tokens. Token can then be used to determine what to do with each token

encountered.

The token character can have one of the following values :

Value	Meaning
toEOF (see page 3321)	End of file has been reached
toFloat (see page 3322)	The next token is a floating (decimal) pointing value
toInteger (see page 3322)	The next token is an integer value
toString (see page 3323)	The next token is a text value
toSymbol (see page 3324)	The next token is a symbol
toWString (see page 3324)	The next token is a wide string value

See Also

NextToken (see page 2922)

toEOF (see page 3321)

toFloat (see page 3322)

toInteger (see page 3322)

toString (see page 3323)

toSymbol (see page 3324)

toWString (see page 3324)

23.1.41.6 TParser Events

The events of the TParser class are listed here.

Public Events

	Name	Description
	OnError (see page 2927)	Occurs when a parse error is encountered.

23.1.41.6.1 TParser.OnError Event

Occurs when a parse error is encountered.

Pascal

```
property OnError: TParserErrorEvent;
```

C++

```
__property TParserErrorEvent OnError;
```

Description

OnError is used internally by TParser (see page 2915) to report errors.

See Also

TParseErrorEvent

23.1.42 TPersistent Class

TPersistent is the ancestor for all objects that have assignment and streaming capabilities.

Class Hierarchy

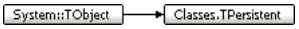

Pascal

```
TPersistent = class(TObject);
```

C++

```
class TPersistent : public TObject;
```

File

Classes

Description

TPersistent encapsulates the behavior common to all objects that can be assigned to other objects, and that can read and write their properties to and from a form file (.xfm or .dfm file). For this purpose TPersistent introduces methods that can be overridden to:

Define the procedure for loading and storing unpublished data to a stream.

Provide the means to assign values to properties.

Provide the means to assign the contents of one object to another.

Do not create instances of TPersistent. Use TPersistent as a base class when declaring objects that are not components, but that need to be saved to a stream or have their properties assigned to other objects.

See Also

TComponent (see page 2762)

TFiler (see page 2834)

TReader (see page 2933)

TStream (see page 2973)

TWriter (see page 3175)

TPersistent Branch

23.1.42.1 TPersistent Members

The following tables list the members exposed by TPersistent.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

23.1.42.2 TPersistent.Destroy Destructor

Destroys the TPersistent (see page 2928) instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPersistent();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

Free (see page 22300)

Destroy (see page 22287)

23.1.42.3 TPersistent Methods

The methods of the TPersistent class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Methods

	Name	Description
	Assign (see page 2930)	Copies the contents of another, similar object.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.

23.1.42.3.1 TPersistent.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); virtual;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent (see page 2928) calls the source object's AssignTo (see page 2931) method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo (see page 2931)

EConvertError (see page 22724)

23.1.42.3.2 TPersistent.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); virtual;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign (see page 2930) method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign (see page 2930) method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign (see page 2930) method of TPersistent (see page 2928) calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent (see page 2928) raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent (see page 2928) version of Assign (see page 2930), which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent (see page 2928) raises an exception.

See Also

Assign (see page 2930)

EConvertError (see page 22724)

23.1.42.3.3 TPersistent.DefineProperties Method

Provides an interface for a method that reads and writes otherwise unpublished data.

Pascal

```
procedure DefineProperties(Filer: TFiler); virtual;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

Descendants of TPersistent (see page 2928) override DefineProperties to designate a method for storing the object's unpublished data to a stream such as a form file. By default, writing an object to a stream writes the values of all its published properties, and reading the object in reads those values and assigns them to the properties. Objects can also specify methods

that read and write data other than published properties by overriding the DefineProperties method.

When overriding DefineProperties, consider including some or all of the following:

A call to the inherited method

Calls to the filer object's DefineProperty method

Calls to the filer object's DefineBinaryProperty method

DefineProperties is virtual, so descendant classes can override it as necessary but are not required to do so.

See Also

DefineProperties (see page 2771)

DefineBinaryProperty (see page 2837)

DefineProperty (see page 2838)

23.1.42.3.4 TPersistent.GetNamePath Method

Returns the name of the object as it appears in the Object Inspector.

Pascal

```
function GetNamePath: string; dynamic;
```

C++

```
__fastcall AnsiString GetNamePath();
```

Description

GetNamePath is for internal use only. It determines the text that the Object Inspector displays for the name of the object being edited. GetNamePath is introduced in TPersistent (see page 2928) so descendants such as collections can appear in the Object Inspector. Do not call GetNamePath directly.

For components, GetNamePath returns the component name. For TCollectionItem (see page 2755) objects it returns the name of the hosting component, the name of the property, and the index into the collection surrounded by brackets.

See Also

GetOwner (see page 2932)

DefineProperties (see page 2771)

DefineBinaryProperty (see page 2837)

DefineProperty (see page 2838)

23.1.42.3.5 TPersistent.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; dynamic;
```

C++

```
__fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath (see page 2932) method to find the owner of a persistent object. GetNamePath (see page 2932) and GetOwner are introduced in TPersistent (see page 2928) so descendants such as collections can appear

in the Object Inspector. As implemented in TPersistent (see page 2928), GetOwner returns nil (Delphi) or NULL (C++).

For TCollection (see page 2731), GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent (see page 2762) it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

23.1.43 TReader Class

TReader is a specialized filer that reads component data from an associated stream.

Class Hierarchy

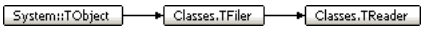

Pascal

```
TReader = class(TFiler);
```

C++

```
class TReader : public TFiler;
```

File

Classes

Description

TReader is used internally by the component streaming system to restore the state of a component that has been written to a stream. It handles the mechanics of reading the data associated with a component from the stream. The reader object, rather than the stream object, is responsible for handling the complexities of streaming components. These include methods for:

Creating component instances from class names read from the stream.

Assigning stream data to published component properties.

Enabling an object to store hidden or complex data as if they were published properties of the object.

Resolving references between components, such as when a property refers to another component.

Other methods and properties of TReader trigger the reader object's events and interaction with the stream.

Applications do not directly create reader objects. Readers are automatically created in a stream object's methods or in global routines that initiate the component streaming process. These include:

the global routine ObjectBinaryToText (see page 3235) procedure, which directly creates a reader object.

the global ReadComponentResFile (see page 3247) function, which creates a file stream that creates a reader object.

the ReadComponent (see page 2945) method of a stream object, which creates a reader object.

Once the streaming process is underway applications do not need to directly manipulate reader objects. The interaction between the reader, component, and stream objects happens automatically in methods of these objects that make calls to each other.

See Also

ObjectBinaryToText (see page 3235)

ReadComponentResFile (see page 3247)

TComponent (see page 2762)

TStream (see page 2973)

TWriter (see page 3175)

Using Streams to Read or Write Data

23.1.43.1 TReader Members

The following tables list the members exposed by TReader.

Protected Methods

	Name	Description
V	Error (see page 2942)	This is Error, a member of class TReader.
V	FindAncestorComponent (see page 2942)	This is FindAncestorComponent, a member of class TReader.
V	FindMethod (see page 2942)	This is FindMethod, a member of class TReader.
V	FindMethodInstance (see page 2942)	This is FindMethodInstance, a member of class TReader.
	PropertyError (see page 2944)	This is PropertyError, a member of class TReader.
	ReadData (see page 2947)	This is ReadData, a member of class TReader.
	ReadProperty (see page 2951)	This is ReadProperty, a member of class TReader.
	ReadPropValue (see page 2952)	This is ReadPropValue, a member of class TReader.
	ReadSet (see page 2952)	This is ReadSet, a member of class TReader.
V	ReferenceName (see page 2956)	This is ReferenceName, a member of class TReader.
V	SetName (see page 2956)	This is SetName, a member of class TReader.
	SetPosition (see page 2957)	This is SetPosition, a member of class TReader.
V	SetRoot (see page 2838)	This is SetRoot, a member of class TFiler.
	SkipBytes (see page 2957)	This is SkipBytes, a member of class TReader.
	SkipComponent (see page 2957)	This is SkipComponent, a member of class TReader.
	SkipProperty (see page 2957)	This is SkipProperty, a member of class TReader.
	SkipSetBody (see page 2958)	This is SkipSetBody, a member of class TReader.

Protected Properties

	Name	Description
R	CanHandleExceptions (see page 2958)	This is CanHandleExceptions, a member of class TReader.
R	PropName (see page 2960)	This is PropName, a member of class TReader.

Public Events

	Name	Description
	OnAncestorNotFound (see page 2961)	Occurs when the object from which a streamed object is derived cannot be found.
	OnCreateComponent (see page 2961)	Occurs when the reader has read a component's class but has not yet instantiated the component.
	OnError (see page 2961)	Occurs when a reader object encounters an error reading its data, such as reading the name of an undeclared property or an illegal value.
	OnFindComponentClass (see page 2962)	Occurs when a reader first reads the name of a component class.
	OnFindComponentInstance (see page 2962)	This is OnFindComponentInstance, a member of class TReader.
	OnFindMethod (see page 2962)	Occurs each time the reader object reads an event, which is a method-pointer property, for an object.
	OnFindMethodInstance (see page 2963)	This is OnFindMethodInstance, a member of class TReader.
	OnReferenceName (see page 2963)	Occurs when resolving references to other objects such as when one object is the property value of another.

	OnSetName (see page 2963)	Occurs just before the reader object sets the Name property of a component it reads from its stream.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginReferences (see page 2938)	Starts a block of commands that read components that might contain references to other components.
	CheckValue (see page 2939)	Checks that the next value to be read from the stream is of a specified type.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CopyValue (see page 2939)	Copies data while reading components.
	Create (see page 2836)	Creates a filer object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DefineBinaryProperty (see page 2939)	Defines binary data the reader object reads as if the data were a published property.
	DefineProperty (see page 2940)	Defines data the reader object reads as if it were a published property.
	Destroy (see page 2936)	Moves the current position of the stream back to the beginning before calling the inherited destructor.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndOfList (see page 2941)	Indicates the end of a group of items that are sequentially read from the stream.
	EndReferences (see page 2941)	Terminates a block of code that reads components from the reader object's stream that might contain references to each other.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FixupReferences (see page 2942)	Resolves references when streaming properties that are represented by another component.
	FlushBuffer (see page 2943)	Synchronizes the reader's buffer with its stream's buffer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextValue (see page 2943)	Returns the type of the next item in the reader object's stream without moving the position of the stream.
	Read (see page 2944)	Reads data from the associated stream.
	ReadBoolean (see page 2944)	Reads a boolean from the reader object's stream and returns that boolean value.
	ReadChar (see page 2945)	Reads a character from the reader object's stream and returns that character value.
	ReadCollection (see page 2945)	Reads a collection from the reader object's stream and returns that character value.

	ReadComponent (see page 2945)	Reads the component specified by Instance from the stream and returns the component.
	ReadComponents (see page 2946)	Reads a list of owned components from the reader object's associated stream.
	ReadCurrency (see page 2946)	Reads a Currency value from the reader object's stream and returns its value.
	ReadDate (see page 2947)	Reads a TDateTime value from the reader object's stream and returns its value.
	ReadDouble (see page 2947)	Reads a double type floating point number from the reader object's stream and returns its value.
	ReadFloat (see page 2948)	Reads a floating-point number from the reader object's stream and returns its value.
	ReadIdent (see page 2948)	Reads an identifier from the reader object's stream and returns the identifier.
	ReadInt64 (see page 2949)	Reads an 64-bit integer from the reader object's stream and returns its value.
	ReadInteger (see page 2949)	Reads an integer-type number from the reader object's stream and returns its value.
	ReadListBegin (see page 2950)	Reads a start-of-list marker from the reader object's associated stream.
	ReadListEnd (see page 2950)	Reads an end-of-list marker from the reader object's associated stream.
V	ReadPrefix (see page 2951)	Reads preliminary information concerning form inheritance for a component.
	ReadRootComponent (see page 2952)	Reads a component and all its owned components from the reader object's stream.
	ReadSignature (see page 2953)	Reads the filer signature from the reader object's associated stream.
	ReadSingle (see page 2953)	Reads a float value from the reader object's stream and returns its value.
	ReadStr (see page 2953)	Reads one character from the reader object's stream and returns its contents.
	ReadString (see page 2954)	Reads a tagged string value written by WriteString from the reader object's stream and returns its contents.
	ReadValue (see page 2954)	Reads the type of the next item on the reader object's stream and returns with the stream positioned after the value-type indicator.
	ReadVariant (see page 2955)	Reads a Variant value from the reader object's stream and returns its value.
	ReadWideChar (see page 2955)	Reads a wide character from the reader object's stream and returns that character value.
	ReadWideString (see page 2956)	Reads a tagged wide string value from the reader object's stream and returns its contents.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SkipValue (see page 2958)	Skips the next value read from the stream.

Public Properties

	Name	Description
	Ancestor (see page 2839)	Determines whether to write the properties of inherited components.
	IgnoreChildren (see page 2839)	Enables a writer object to save a component without saving the components it owns.
R	LookupRoot (see page 2840)	Specifies the root component for resolving names while loading and saving components.
	Owner (see page 2959)	Stores the component that is assigned as the Owner property of components read from the reader object's stream.
	Parent (see page 2959)	Stores the component that is assigned as the Parent property of controls read from the reader's stream.
	Position (see page 2960)	Represents the current reading position in the associated stream.
	Root (see page 2840)	Indicates which object is the root, or ultimate owner, of the objects read or written by the filer.

23.1.43.2 TReader.Destroy Destructor

Moves the current position of the stream back to the beginning before calling the inherited destructor.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TReader();
```

Description

Do not call Destroy directly. Reader objects are created indirectly through the methods or routines that use them. These methods and routines handle freeing the TReader (see page 2933) object.

See Also

Free (see page 22300)

23.1.43.3 TReader Methods

The methods of the TReader class are listed here.

Protected Methods

	Name	Description
	Error (see page 2942)	This is Error, a member of class TReader.
	FindAncestorComponent (see page 2942)	This is FindAncestorComponent, a member of class TReader.
	FindMethod (see page 2942)	This is FindMethod, a member of class TReader.
	FindMethodInstance (see page 2942)	This is FindMethodInstance, a member of class TReader.
	PropertyError (see page 2944)	This is PropertyError, a member of class TReader.
	ReadData (see page 2947)	This is ReadData, a member of class TReader.
	ReadProperty (see page 2951)	This is ReadProperty, a member of class TReader.
	ReadPropValue (see page 2952)	This is ReadPropValue, a member of class TReader.
	ReadSet (see page 2952)	This is ReadSet, a member of class TReader.
	ReferenceName (see page 2956)	This is ReferenceName, a member of class TReader.
	SetName (see page 2956)	This is SetName, a member of class TReader.
	SetPosition (see page 2957)	This is SetPosition, a member of class TReader.
	SkipBytes (see page 2957)	This is SkipBytes, a member of class TReader.
	SkipComponent (see page 2957)	This is SkipComponent, a member of class TReader.
	SkipProperty (see page 2957)	This is SkipProperty, a member of class TReader.
	SkipSetBody (see page 2958)	This is SkipSetBody, a member of class TReader.

Public Methods

	Name	Description
	BeginReferences (see page 2938)	Starts a block of commands that read components that might contain references to other components.
	CheckValue (see page 2939)	Checks that the next value to be read from the stream is of a specified type.
	CopyValue (see page 2939)	Copies data while reading components.
	DefineBinaryProperty (see page 2939)	Defines binary data the reader object reads as if the data were a published property.
	DefineProperty (see page 2940)	Defines data the reader object reads as if it were a published property.
	EndOfList (see page 2941)	Indicates the end of a group of items that are sequentially read from the stream.
	EndReferences (see page 2941)	Terminates a block of code that reads components from the reader object's stream that might contain references to each other.
	FixupReferences (see page 2942)	Resolves references when streaming properties that are represented by another component.
	FlushBuffer (see page 2943)	Synchronizes the reader's buffer with its stream's buffer.
	NextValue (see page 2943)	Returns the type of the next item in the reader object's stream without moving the position of the stream.
	Read (see page 2944)	Reads data from the associated stream.
	ReadBoolean (see page 2944)	Reads a boolean from the reader object's stream and returns that boolean value.
	ReadChar (see page 2945)	Reads a character from the reader object's stream and returns that character value.
	ReadCollection (see page 2945)	Reads a collection from the reader object's stream and returns that character value.
	ReadComponent (see page 2945)	Reads the component specified by Instance from the stream and returns the component.

	ReadComponents (see page 2946)	Reads a list of owned components from the reader object's associated stream.
	ReadCurrency (see page 2946)	Reads a Currency value from the reader object's stream and returns its value.
	ReadDate (see page 2947)	Reads a TDateTime value from the reader object's stream and returns its value.
	ReadDouble (see page 2947)	Reads a double type floating point number from the reader object's stream and returns its value.
	ReadFloat (see page 2948)	Reads a floating-point number from the reader object's stream and returns its value.
	ReadIdent (see page 2948)	Reads an identifier from the reader object's stream and returns the identifier.
	ReadInt64 (see page 2949)	Reads an 64-bit integer from the reader object's stream and returns its value.
	ReadInteger (see page 2949)	Reads an integer-type number from the reader object's stream and returns its value.
	ReadListBegin (see page 2950)	Reads a start-of-list marker from the reader object's associated stream.
	ReadListEnd (see page 2950)	Reads an end-of-list marker from the reader object's associated stream.
V	ReadPrefix (see page 2951)	Reads preliminary information concerning form inheritance for a component.
	ReadRootComponent (see page 2952)	Reads a component and all its owned components from the reader object's stream.
	ReadSignature (see page 2953)	Reads the filer signature from the reader object's associated stream.
	ReadSingle (see page 2953)	Reads a float value from the reader object's stream and returns its value.
	ReadStr (see page 2953)	Reads one character from the reader object's stream and returns its contents.
	ReadString (see page 2954)	Reads a tagged string value written by WriteString from the reader object's stream and returns its contents.
	ReadValue (see page 2954)	Reads the type of the next item on the reader object's stream and returns with the stream positioned after the value-type indicator.
	ReadVariant (see page 2955)	Reads a Variant value from the reader object's stream and returns its value.
	ReadWideChar (see page 2955)	Reads a wide character from the reader object's stream and returns that character value.
	ReadWideString (see page 2956)	Reads a tagged wide string value from the reader object's stream and returns its contents.
	SkipValue (see page 2958)	Skips the next value read from the stream.

23.1.43.3.1 TReader.BeginReferences Method

Starts a block of commands that read components that might contain references to other components.

Pascal

```
procedure BeginReferences;
```

C++

```
__fastcall BeginReferences();
```

Description

Do not call BeginReferences directly. BeginReferences is used internally for streaming properties that reference other components.

Following the call to BeginReferences, the reader object creates a list of all the objects read and their names. After all of the interdependent objects are read, FixupReferences (see page 2942) is called to resolve the named references from the stream into instance references. Finally, call EndReferences (see page 2941) to dispose of the fixup list.

BeginReferences is always used together with FixupReferences (see page 2942) and EndReferences (see page 2941) in a try...finally (Delphi) or try...__finally (C++) block.

See Also

EndReferences (see page 2941)

FixupReferences (see page 2942)

23.1.43.3.2 TReader.CheckValue Method

Checks that the next value to be read from the stream is of a specified type.

Pascal

```
procedure CheckValue(Value: TValueType);
```

C++

```
__fastcall CheckValue(TValueType Value);
```

Description

CheckValue ascertains that the next value to be read from the stream is of the type specified by Value. If it is not, CheckValue skips over that value, positioning the stream on the following value, and raises an EReadError (see page 2703) exception.

See Also

EReadError (see page 2703)

Position (see page 2960)

23.1.43.3.3 TReader.CopyValue Method

Copies data while reading components.

Pascal

```
procedure CopyValue(Writer: TWriter);
```

C++

```
__fastcall CopyValue(TWriter Writer);
```

Description

Do not call CopyValue, it is used internally.

23.1.43.3.4 TReader.DefineBinaryProperty Method

Defines binary data the reader object reads as if the data were a published property.

Pascal

```
procedure DefineBinaryProperty(const Name: string; ReadData: TStreamProc; WriteData:
TStreamProc; HasData: Boolean); override;
```

C++

```
virtual __fastcall DefineBinaryProperty(const AnsiString Name, TStreamProc ReadData,
TStreamProc WriteData, Boolean HasData);
```

Description

DefineBinaryProperty is called internally by the DefineProperties method of an object that has data it needs to store. Applications can call DefineBinaryProperty directly when overriding the DefineProperties method of a component with large custom data (like a bitmap). This is the only context in which DefineBinaryProperty (or DefineProperty (see page 2940)) should be used.

DefineProperties takes a generic filer object as its parameter. For reading binary data, DefineProperties takes a TReader (see page 2933) object and calls its DefineBinaryProperty method. DefineBinaryProperty then reads the property's name data and passes the data to the indicated ReadData (see page 2947) method.

The Name parameter specifies the name of the "fake" property to be read from the stream. A "fake" property is a property that is not published, and that exists only in the code for the DefineBinaryProperty method. The Top and Left properties of a non-visual

component are examples of "fake" properties. These are also called "defined properties" or "custom defined properties."

The ReadData (see page 2947) and WriteData parameters point to a procedure (defined in the component object) that reads or writes a binary representation of the object's data directly to or from the stream passed to them in the Stream parameter. For TReader (see page 2933) the WriteData parameter is ignored.

The HasData parameter determines at run time whether the "fake" property has data to store (write). For TReader (see page 2933) the HasData parameter is ignored.

Defined binary properties are rare. More commonly, objects use the DefineProperty (see page 2940) method. The difference between DefineBinaryProperty and DefineProperty (see page 2940) is that when using DefineBinaryProperty, the component reads the binary property directly from a memory stream, rather than going through the filer object. Binary data is also harder to edit as text.

Note: Streamable objects that are descended from TPersistent (see page 2928) inherit a DefineProperties method, however DefineProperties does nothing until TComponent (see page 2762). Therefore, it is a component's DefineProperties method that calls the reader's DefineBinaryProperty when reading binary data.

See Also

DefineProperty (see page 2940)

DefineProperties (see page 2771)

23.1.43.3.5 TReader.DefineProperty Method

Defines data the reader object reads as if it were a published property.

Pascal

```
procedure DefineProperty(const Name: string; ReadData: TReaderProc; WriteData: TWriterProc;
HasData: Boolean); override;
```

C++

```
virtual __fastcall DefineProperty(const AnsiString Name, TReaderProc ReadData, TWriterProc
WriteData, Boolean HasData);
```

Description

DefineProperty is called internally by the DefineProperties method of an object that has data it needs to store. DefineProperties takes a generic filer object as its parameter. For reading data DefineProperties takes a TReader (see page 2933) object and then calls the DefineProperties method of the reader object. DefineProperty then reads the property's name and its data.

The Name parameter specifies the name of the "fake" property to be read from the stream.

The ReadData (see page 2947) parameter points to a procedure (defined in the storing object) that reads the object's data which represents a property value to the reader object. For TReader (see page 2933) the WriteData parameter is ignored.

The HasData parameter determines at run time whether the "fake" property has data to store (write). For TReader (see page 2933) the HasData parameter is ignored.

The difference between DefineBinaryProperty (see page 2939) and DefineProperty is that the binary property is read directly from a stream, rather than going through a filer object, and binary data is harder to edit as text.

See Also

Ancestor (see page 2839)

DefineBinaryProperty (see page 2939)

DefineProperties (see page 2771)

23.1.43.3.6 TReader.EndOfList Method

Indicates the end of a group of items that are sequentially read from the stream.

Pascal

```
function EndOfList: Boolean;
```

C++

```
__fastcall Boolean EndOfList();
```

Description

EndOfList is used by other methods that iterate through a list of items when reading in data. It checks for a value type that signals the end of that type of data. EndOfList returns true if the reader object has read to the end of a list of items.

When reading a list of items, the reader object calls ReadListBegin (see page 2950) method, then the items are repeatedly read until EndOfList returns true. Finally, the reader calls ReadListEnd (see page 2950).

Note: Component writers can use this in their defined property ReadData (see page 2947) methods, as, for example, with TStrings (see page 3029).

See Also

ReadListBegin (see page 2950)

ReadListEnd (see page 2950)

DefineProperty (see page 2838)

23.1.43.3.7 TReader.EndReferences Method

Terminates a block of code that reads components from the reader object's stream that might contain references to each other.

Pascal

```
procedure EndReferences;
```

C++

```
__fastcall EndReferences();
```

Description

EndReferences is used for streaming properties that reference another component. Programmers should not need to call EndReferences directly.

The reader object calls EndReferences and creates a list of all the objects read and their names. After all the interdependent objects are read, FixupReferences (see page 2942) is called to resolve the named references from the stream into instance references. Finally, call EndReferences to dispose of the fixup list.

EndReferences is always used in the finally part of a try...finally (Delphi) or try...__finally (C++) block that also includes calls to BeginReferences (see page 2938) and FixupReferences (see page 2942).

See Also

BeginReferences (see page 2938)

FixupReferences (see page 2942)

23.1.43.3.8 TReader.Error Method

Pascal

```
function Error(const Message: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean Error(const AnsiString Message);
```

Description

This is Error, a member of class TReader.

23.1.43.3.9 TReader.FindAncestorComponent Method

Pascal

```
function FindAncestorComponent(const Name: string; ComponentClass: TPersistentClass):
TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * FindAncestorComponent(const AnsiString Name, TPersistentClass
ComponentClass);
```

Description

This is FindAncestorComponent, a member of class TReader.

23.1.43.3.10 TReader.FindMethod Method

Pascal

```
function FindMethod(Root: TComponent; const MethodName: string): Pointer; virtual;
```

C++

```
virtual __fastcall void * FindMethod(TComponent * Root, const AnsiString MethodName);
```

Description

This is FindMethod, a member of class TReader.

23.1.43.3.11 TReader.FindMethodInstance Method

Pascal

```
function FindMethodInstance(Root: TComponent; const MethodName: string): TMethod; virtual;
```

C++

```
virtual __fastcall TMethod FindMethodInstance(TComponent * Root, const AnsiString MethodName);
```

Description

This is FindMethodInstance, a member of class TReader.

23.1.43.3.12 TReader.FixupReferences Method

Resolves references when streaming properties that are represented by another component.

Pascal

```
procedure FixupReferences;
```

C++

```
__fastcall FixupReferences();
```

Description

FixupReferences is used internally by the streaming system to resolve the named references from the stream into instance references. Users would never need to call this method.

The reader object calls BeginReferences (see page 2938) and creates a list of all the objects read and their names. After all the interdependent objects are read, FixupReferences is called to resolve the references among various mutually dependent components read from the reader object's stream after all of the objects have been read. Finally, call EndReferences (see page 2941) to dispose of the fixup list.

FixupReferences is always used inside a try...finally (Delphi) or try...__finally (C++) block between calls to BeginReferences (see page 2938) and EndReferences (see page 2941).

See Also

BeginReferences (see page 2938)

EndReferences (see page 2941)

23.1.43.3.13 TReader.FlushBuffer Method

Synchronizes the reader's buffer with its stream's buffer.

Pascal

```
procedure FlushBuffer; override;
```

C++

```
virtual __fastcall FlushBuffer();
```

Description

FlushBuffer synchronizes the reader's buffer with the associated stream by setting the stream's Position (see page 2960) to match the reader's Position (see page 2960).

See Also

BeginReferences (see page 2938)

EndReferences (see page 2941)

23.1.43.3.14 TReader.NextValue Method

Returns the type of the next item in the reader object's stream without moving the position of the stream.

Pascal

```
function NextValue: TValueType;
```

C++

```
__fastcall TValueType NextValue();
```

Description

NextValue is used internally by reader methods to query the type of the next item in the stream. After identifying the type of the next item, NextValue returns with the stream position still before the value-type indicator.

See Also

ReadValue (see page 2954)

23.1.43.3.15 TReader.PropertyError Method

Pascal

```
procedure PropertyError(const Name: string);
```

C++

```
__fastcall PropertyError(const AnsiString Name);
```

Description

This is PropertyError, a member of class TReader.

23.1.43.3.16 TReader.Read Method

Reads data from the associated stream.

Pascal

```
procedure Read(var Buf; Count: Longint);
```

C++

```
__fastcall Read( Buf, Longint Count);
```

Description

Do not call read directly. Read is used internally to read data from the stream. Many other writer methods call Read, usually after setting pertinent values or verifying data types.

See Also

ReadChar (see page 2945)

ReadIdent (see page 2948)

ReadInteger (see page 2949)

ReadStr (see page 2953)

ReadString (see page 2954)

23.1.43.3.17 TReader.ReadBoolean Method

Reads a boolean from the reader object's stream and returns that boolean value.

Pascal

```
function ReadBoolean: Boolean;
```

C++

```
__fastcall Boolean ReadBoolean();
```

Description

ReadBoolean is a helper method used by other reader methods to read a tagged boolean value at the current reader Position (see page 2960). ReadBoolean uses ReadValue (see page 2954) to check the type for a vaTrue value and returns that result. Note that ReadBoolean will not verify that the value being read is a boolean, only whether or not the value is true.

See Also

ReadValue (see page 2954)

TValueType

23.1.43.3.18 TReader.ReadChar Method

Reads a character from the reader object's stream and returns that character value.

Pascal

```
function ReadChar: Char;
```

C++

```
__fastcall Char ReadChar();
```

Description

ReadChar is a helper method used by other reader methods to read a tagged character value at the current reader Position (see page 2960). ReadChar first checks the value type (TValueType (see page 3279)) by calling the ReadValue (see page 2954) method. If the item is a vaString, ReadString (see page 2954) checks the length. If the length indicates that the value is a char, it calls Read (see page 2944) and returns the character. Otherwise a EReadError (see page 2703) exception is raised.

Note: If the tagged data is a string with length = 1, it is char compatible. If length > 1, it is an error, however, the rest of the string data must be read to keep the reader in a consistent state. Each action that processes tagged data must leave the reader Position (see page 2960) at the start of the next unread tag.

See Also

EReadError (see page 2703)

Position (see page 2960)

Read (see page 2944)

TValueType

23.1.43.3.19 TReader.ReadCollection Method

Reads a collection from the reader object's stream and returns that character value.

Pascal

```
procedure ReadCollection(Collection: TCollection);
```

C++

```
__fastcall ReadCollection(TCollection Collection);
```

Description

ReadCollection is a helper method used by other reader methods to read a tagged collection value at the current reader Position (see page 2960). ReadCollection iterates through items in a collection so that each item in a collection property has an opportunity to stream out its properties. This method does not write out the number of items and the collection indices, because a user can add or delete items.

23.1.43.3.20 TReader.ReadComponent Method

Reads the component specified by Instance from the stream and returns the component.

Pascal

```
function ReadComponent(Component: TComponent): TComponent;
```

C++

```
__fastcall TComponent * ReadComponent(TComponent * Component);
```

Description

ReadComponent is called by ReadRootComponent (see page 2952) to read each of the components owned by Root (see page 2840). Users would not need to call ReadComponent directly.

ReadComponent handles the loading and initialization process for each component. After loading all the components in the list, ReadComponents (see page 2946) calls the Loaded method of each loaded component in the order it read them. ReadComponent manages the complexities of reading the component from the stream through calls to other reader methods and to the component's methods.

See Also

ReadComponents (see page 2946)

ReadRootComponent (see page 2952)

23.1.43.3.21 TReader.ReadComponents Method

Reads a list of owned components from the reader object's associated stream.

Pascal

```
procedure ReadComponents(AOwner: TComponent; AParent: TComponent; Proc: TReadComponentsProc);
```

C++

```
__fastcall ReadComponents(TComponent * AOwner, TComponent * AParent, TReadComponentsProc Proc);
```

Description

ReadComponents is used internally by the IDE for design-time streaming. Do not call ReadComponents directly.

ReadComponents first sets the reader object's Root (see page 2840) and Owner (see page 2959) properties to the component passed in the AOwner parameter and sets its Parent (see page 2959) property to AParent. Next ReadComponents reads components by calling the ReadComponent (see page 2945) method, passing each returned component to the method passed in Proc.

Note: ReadComponents is used at design-time for clipboard cut and paste of multiple components.

23.1.43.3.22 TReader.ReadCurrency Method

Reads a Currency value from the reader object's stream and returns its value.

Pascal

```
function ReadCurrency: Currency;
```

C++

```
__fastcall Currency ReadCurrency();
```

Description

ReadCurrency is a helper method used by other reader methods to read a tagged Currency data value at the current reader Position (see page 2960). ReadCurrency first checks the value type (TValueType (see page 3279)) by calling the ReadValue (see page 2954) method. If the item is a vaCurrency ReadCurrency calls Read (see page 2944) and returns the value. Otherwise it calls ReadInteger (see page 2949).

See Also

Position (see page 2960)

Read (see page 2944)

ReadInteger (see page 2949)

ReadValue (see page 2954)

TValueType

23.1.43.3.23 TReader.ReadData Method

Pascal

```
procedure ReadData(Instance: TComponent);
```

C++

```
__fastcall ReadData(TComponent * Instance);
```

Description

This is ReadData, a member of class TReader.

23.1.43.3.24 TReader.ReadDate Method

Reads a TDateTime value from the reader object's stream and returns its value.

Pascal

```
function ReadDate: TDateTime;
```

C++

```
__fastcall TDateTime ReadDate();
```

Description

ReadDate is a helper method used by other reader methods to read a tagged TDateTime data value at the current reader Position (see page 2960). ReadDate first checks the value type (TValueType (see page 3279)) by calling the ReadValue (see page 2954) method. If the item is a vaDate ReadDate calls Read (see page 2944) and returns the value. Otherwise it calls ReadInteger (see page 2949).

See Also

Position (see page 2960)

Read (see page 2944)

ReadInteger (see page 2949)

ReadValue (see page 2954)

TValueType

23.1.43.3.25 TReader.ReadDouble Method

Reads a double type floating point number from the reader object's stream and returns its value.

Pascal

```
function ReadDouble: Double;
```

C++

```
__fastcall Double ReadDouble();
```

Description

ReadDouble is a helper method used by other reader methods to read a tagged double type floating-point data value at the current reader Position (see page 2960). ReadFloat (see page 2948) first checks the value type (TValueType (see page 3279)) by calling the ReadValue (see page 2954) method. If the item is a vaDouble ReadDouble calls Read (see page 2944) and returns the double value. Otherwise it calls ReadInteger (see page 2949).

See Also

Position (see page 2960)

Read (see page 2944)

ReadInteger (see page 2949)

ReadValue (see page 2954)

TValueType

23.1.43.3.26 TReader.ReadFloat Method

Reads a floating-point number from the reader object's stream and returns its value.

Pascal

```
function ReadFloat: Extended;
```

C++

```
__fastcall Extended ReadFloat();
```

Description

ReadFloat is a helper method used by other reader methods to read a tagged floating-point data value at the current reader Position (see page 2960). ReadFloat first checks the value type (TValueType (see page 3279)) by calling the ReadValue (see page 2954) method. If the item is a vaExtended ReadFloat calls Read (see page 2944) and returns the float value. Otherwise it calls ReadInteger (see page 2949).

See Also

Position (see page 2960)

Read (see page 2944)

ReadInteger (see page 2949)

ReadValue (see page 2954)

TValueType

23.1.43.3.27 TReader.ReadIdent Method

Reads an identifier from the reader object's stream and returns the identifier.

Pascal

```
function ReadIdent: string;
```

C++

```
__fastcall AnsiString ReadIdent();
```

Description

ReadIndent is a helper method used by other reader methods to read an identifying type at the current reader Position (see page 2960). ReadIndent first checks the value type (TValueType (see page 3279)) by calling the ReadValue (see page 2954) method. If the item is:

A vaIdent type, ReadIndent calls Read (see page 2944) and returns the identifier string value.

A vaFalse type, ReadIdent returns the string 'False.'

A vaTrue type, ReadIdent returns the string 'True.'

A vaNil type, ReadIdent returns the string 'Nil.'

Otherwise it raises an EReadError (see page 2703) exception.

Note: The strings returned for vaFalse, vaTrue, and vaNil correspond to Delphi reserved words.

See Also

EReadError (see page 2703)

Position (see page 2960)

Read (see page 2944)

ReadValue (see page 2954)

TValueType

23.1.43.3.28 TReader.ReadInt64 Method

Reads an 64-bit integer from the reader object's stream and returns its value.

Pascal

```
function ReadInt64: Int64;
```

C++

```
__fastcall Int64 ReadInt64();
```

Description

ReadInt64 is a helper method used by other reader methods to read a tagged 64-bit integer value at the current reader Position (see page 2960). ReadInt64 first checks the value type (TValueType (see page 3279)) by calling the ReadValue (see page 2954) method. If the item is a vaInt64, ReadInt64 calls Read (see page 2944) and returns the integer value. Otherwise it calls ReadInteger (see page 2949), in case the value was saved in another integer format.

See Also

Position (see page 2960)

Read (see page 2944)

ReadValue (see page 2954)

TValueType

ReadInteger (see page 2949)

23.1.43.3.29 TReader.ReadInteger Method

Reads an integer-type number from the reader object's stream and returns its value.

Pascal

```
function ReadInteger: Longint;
```

C++

```
__fastcall Longint ReadInteger();
```

Description

ReadInteger is a helper method used by other reader methods to read a tagged integer value at the current reader Position (see page 2960). ReadInteger first checks the value type (TValueType (see page 3279)) by calling the ReadValue (see page 2954) method. If the item is a vaString or a vaInt8, vaInt16 or vaInt32 type, ReadInteger calls Read (see page 2944) and returns the integer value. Otherwise it raises an EReadError (see page 2703) exception.

See Also

EReadError (see page 2703)

Position (see page 2960)

Read (see page 2944)

ReadValue (see page 2954)

TValueType

23.1.43.3.30 TReader.ReadListBegin Method

Reads a start-of-list marker from the reader object's associated stream.

Pascal

```
procedure ReadListBegin;
```

C++

```
__fastcall ReadListBegin();
```

Description

ReadListBegin is used by other methods that iterate through a list of items to ensure that the type about to be read is a list. ReadListBegin is also used by component writers in defined property ReadData (see page 2947) methods.

If the next item in the stream is not a start-of list marker as written by the WriteListBegin method, ReadListBegin raises an EReadError (see page 2703) exception.

A call to ReadListBegin is generally followed by a reading loop that terminates when the EndOfList (see page 2941) method returns true, indicating that an end-of-list marker is next on the stream, at which point a call to ReadListEnd (see page 2950) is required.

See Also

EndOfList (see page 2941)

EReadError (see page 2703)

ReadListEnd (see page 2950)

WriteListBegin (see page 3187)

23.1.43.3.31 TReader.ReadListEnd Method

Reads an end-of-list marker from the reader object's associated stream.

Pascal

```
procedure ReadListEnd;
```

C++

```
__fastcall ReadListEnd();
```

Description

ReadListEnd is used by other methods that iterate through a list of items to signal the end of a group of items being read. ReadListEnd is also used by component writers in defined property ReadData (see page 2947) methods.

A call to ReadListEnd is generally preceded by a reading loop that terminates when the EndOfList (see page 2941) method returns true, indicating that an end-of-list marker is next on the stream.

If the item read is not an end-of-list marker, ReadListEnd raises an EReadError (see page 2703) exception. A call to ReadListEnd must correspond to a preceding call to ReadListBegin (see page 2950).

See Also

EndOfList (see page 2941)

EReadError (see page 2703)

ReadListBegin (see page 2950)

WriteListEnd (see page 3187)

23.1.43.3.32 TReader.ReadPrefix Method

Reads preliminary information concerning form inheritance for a component.

Pascal

```
procedure ReadPrefix(var Flags: TFilerFlags; var AChildPos: Integer); virtual;
```

C++

```
virtual __fastcall ReadPrefix(TFilerFlags Flags, int AChildPos);
```

Description

ReadPrefix is used internally by the component streaming system. It is called automatically by ReadComponent (see page 2945) to read flags that determine whether a component in an inherited form (ffInherited), whether its creation order in the form is important (ffChildPos), and whether it is a top-level component such as a form or data module (ffInline). It also reads the component's position in the ancestor form's creation order if the ffChildPos flag is set. When a writer object writes a component to its stream, it prefixes the component with these flags.

See Also

ReadComponent (see page 2945)

23.1.43.3.33 TReader.ReadProperty Method

Pascal

```
procedure ReadProperty(AInstance: TPersistent);
```

C++

```
__fastcall ReadProperty(TPersistent * AInstance);
```

Description

This is ReadProperty, a member of class TReader.

23.1.43.3.34 TReader.ReadPropValue Method

Pascal

```
procedure ReadPropValue(Instance: TPersistent; PropInfo: Pointer);
```

C++

```
__fastcall ReadPropValue(TPersistent * Instance, void * PropInfo);
```

Description

This is ReadPropValue, a member of class TReader.

23.1.43.3.35 TReader.ReadRootComponent Method

Reads a component and all its owned components from the reader object's stream.

Pascal

```
function ReadRootComponent(Root: TComponent): TComponent;
```

C++

```
__fastcall TComponent * ReadRootComponent(TComponent * Root);
```

Description

Do not call ReadRootComponent directly. It is used internally by the component streaming system. When the reader object is created by its stream in the stream's ReadComponent (see page 2945) method, it immediately calls the ReadRootComponent method, which initiates the streaming sequence of restoring an object.

ReadRootComponent first calls the ReadSignature (see page 2953) method to ensure that it is reading a proper component. ReadRootComponent also handles initializations and fixup references. For example, the component passed in Root (see page 2840) becomes the value of the Root (see page 2840) property for the reader object.

ReadRootComponent calls the component's ReadState method, which calls back to the reader's ReadData (see page 2947) method. The ReadState method of TComponent (see page 2762) is virtual, which allows component classes to override ReadState to prepare themselves for being loaded with the new data from the stream.

See Also

FixupReferences (see page 2942)

ReadComponents (see page 2946)

Read (see page 2944)

ReadSignature (see page 2953)

ReadState (see page 2784)

23.1.43.3.36 TReader.ReadSet Method

Pascal

```
function ReadSet(SetType: Pointer): Integer;
```

C++

```
__fastcall int ReadSet(void * SetType);
```

Description

This is ReadSet, a member of class TReader.

23.1.43.3.37 TReader.ReadSignature Method

Reads the filer signature from the reader object's associated stream.

Pascal

```
procedure ReadSignature;
```

C++

```
__fastcall ReadSignature();
```

Description

ReadSignature is used by several routines to ensure that the file about to be read is a valid component stream. For example, ReadRootComponent (see page 2952), calls ReadSignature before reading its component from the stream. By checking for the signature before loading objects, the reader object can guard against inadvertently reading invalid or corrupted data.

The signature is a four-character sequence.

See Also

ReadComponents (see page 2946)

ReadRootComponent (see page 2952)

23.1.43.3.38 TReader.ReadSingle Method

Reads a float value from the reader object's stream and returns its value.

Pascal

```
function ReadSingle: Single;
```

C++

```
__fastcall Single ReadSingle();
```

Description

ReadSingle is a helper method used by other reader methods to read a tagged Delphi Single data value at the current reader Position (see page 2960). (The Single data type corresponds to float in C++.) ReadSingle first checks the value type (TValueType (see page 3279)) by calling the ReadValue (see page 2954) method. If the item is a vaSingle, ReadSingle calls Read (see page 2944) and returns the value. Otherwise it calls ReadInteger (see page 2949).

See Also

Position (see page 2960)

Read (see page 2944)

ReadInteger (see page 2949)

ReadValue (see page 2954)

TValueType

23.1.43.3.39 TReader.ReadStr Method

Reads one character from the reader object's stream and returns its contents.

Pascal

```
function ReadStr: string;
```

C++

```
__fastcall AnsiString ReadStr();
```

Description

ReadStr is a helper method used by other reader methods to read one character from a string at the current reader Position (see page 2960). Unlike ReadChar (see page 2945), ReadStr will not throw an exception if the tagged value is not a single character.

Warning: This method is for Delphi internal use only.

See Also

EReadError (see page 2703)

Position (see page 2960)

ReadChar (see page 2945)

ReadString (see page 2954)

23.1.43.3.40 TReader.ReadString Method

Reads a tagged string value written by WriteString from the reader object's stream and returns its contents.

Pascal

```
function ReadString: string;
```

C++

```
__fastcall AnsiString ReadString();
```

Description

ReadString is a helper method used by other reader methods to read a string at the current reader Position (see page 2960). ReadString first checks the value type (TValueType (see page 3279)) by calling the ReadValue (see page 2954) method. If the item is a vaString or a vaLString type ReadString calls Read (see page 2944) and returns the string. Otherwise it raises an EReadError (see page 2703) exception.

Warning: Always use ReadString for reading component strings to streams.

See Also

EReadError (see page 2703)

Position (see page 2960)

Read (see page 2944)

ReadValue (see page 2954)

TValueType

23.1.43.3.41 TReader.ReadValue Method

Reads the type of the next item on the reader object's stream and returns with the stream positioned after the value-type indicator.

Pascal

```
function ReadValue: TValueType;
```

C++

```
__fastcall TValueType ReadValue();
```

Description

ReadValue is used by other reader methods when the subsequent processing depends upon a type check of the data about to be read.

See Also

NextValue (see page 2943)

23.1.43.3.42 TReader.ReadVariant Method

Reads a Variant value from the reader object's stream and returns its value.

Pascal

```
function ReadVariant: Variant;
```

C++

```
__fastcall Variant ReadVariant();
```

Description

ReadVariant is a helper method used by other reader methods to read a tagged Variant data value at the current reader Position (see page 2960). ReadVariant first checks the value type (TValueType (see page 3279)) by calling the ReadValue (see page 2954) method. If the item is a vaVariant ReadVariant calls Read (see page 2944) and returns the value. Otherwise it calls ReadInteger (see page 2949).

See Also

Position (see page 2960)

Read (see page 2944)

ReadInteger (see page 2949)

ReadValue (see page 2954)

TValueType

WriteVariant (see page 3190)

23.1.43.3.43 TReader.ReadWideChar Method

Reads a wide character from the reader object's stream and returns that character value.

Pascal

```
function ReadWideChar: WideChar;
```

C++

```
__fastcall WideChar ReadWideChar();
```

Description

ReadWideChar is a helper method used by other reader methods to read a tagged wide character value at the current reader Position (see page 2960). ReadWideChar invokes the ReadWideString (see page 2956), and return that string if it is 1 character long. Otherwise, an exception is thrown.

Note: If the tagged data is a widestring with length = 1, it is widechar compatible. If length > 1, it is an error, however, the rest of

the string data must be read to keep the reader in a consistent state. Each action that processes tagged data must leave the reader Position (see page 2960) at the start of the next unread tag.

See Also

EReadError (see page 2703)

Position (see page 2960)

Read (see page 2944)

ReadWideString (see page 2956)

TValueType

23.1.43.3.44 TReader.ReadWideString Method

Reads a tagged wide string value from the reader object's stream and returns its contents.

Pascal

```
function ReadWideString: WideString;
```

C++

```
__fastcall BSTR ReadWideString();
```

Description

ReadWideString is a helper method used by other reader methods to read a wide string at the current reader Position (see page 2960). ReadWideString first checks the value type (TValueType (see page 3279)) by calling the ReadValue (see page 2954) method. If the item is a vaWString type, ReadWideString calls Read (see page 2944) and returns the string. Otherwise it raises an EReadError (see page 2703) exception.

See Also

EReadError (see page 2703)

Position (see page 2960)

Read (see page 2944)

ReadValue (see page 2954)

TValueType

23.1.43.3.45 TReader.ReferenceName Method

Pascal

```
procedure ReferenceName(var Name: string); virtual;
```

C++

```
virtual __fastcall ReferenceName(AnsiString Name);
```

Description

This is ReferenceName, a member of class TReader.

23.1.43.3.46 TReader.SetName Method

Pascal

```
procedure SetName(Component: TComponent; var Name: string); virtual;
```

C++

```
virtual __fastcall SetName(TComponent * Component, AnsiString Name);
```

Description

This is SetName, a member of class TReader.

23.1.43.3.47 TReader.SetPosition Method

Pascal

```
procedure SetPosition(Value: Longint);
```

C++

```
__fastcall SetPosition(Longint Value);
```

Description

This is SetPosition, a member of class TReader.

23.1.43.3.48 TReader.SkipBytes Method

Pascal

```
procedure SkipBytes(Count: Integer);
```

C++

```
__fastcall SkipBytes(int Count);
```

Description

This is SkipBytes, a member of class TReader.

23.1.43.3.49 TReader.SkipComponent Method

Pascal

```
procedure SkipComponent(SkipHeader: Boolean);
```

C++

```
__fastcall SkipComponent(Boolean SkipHeader);
```

Description

This is SkipComponent, a member of class TReader.

23.1.43.3.50 TReader.SkipProperty Method

Pascal

```
procedure SkipProperty;
```

C++

```
__fastcall SkipProperty();
```

Description

This is SkipProperty, a member of class TReader.

23.1.43.3.51 TReader.SkipSetBody Method

Pascal

```
procedure SkipSetBody;
```

C++

```
__fastcall SkipSetBody();
```

Description

This is SkipSetBody, a member of class TReader.

23.1.43.3.52 TReader.SkipValue Method

Skips the next value read from the stream.

Pascal

```
procedure SkipValue;
```

C++

```
__fastcall SkipValue();
```

Description

Use SkipValue to skip over a value, positioning the stream on the following value.

See Also

EReadError (see page 2703)

Position (see page 2960)

23.1.43.4 TReader Properties

The properties of the TReader class are listed here.

Protected Properties

	Name	Description
R	CanHandleExceptions (see page 2958)	This is CanHandleExceptions, a member of class TReader.
R	PropName (see page 2960)	This is PropName, a member of class TReader.

Public Properties

	Name	Description
	Owner (see page 2959)	Stores the component that is assigned as the Owner property of components read from the reader object's stream.
	Parent (see page 2959)	Stores the component that is assigned as the Parent property of controls read from the reader's stream.
	Position (see page 2960)	Represents the current reading position in the associated stream.

23.1.43.4.1 TReader.CanHandleExceptions Property

Pascal

```
property CanHandleExceptions: Boolean;
```

C++

```
__property Boolean CanHandleExceptions;
```

Description

This is CanHandleExceptions, a member of class TReader.

23.1.43.4.2 TReader.Owner Property

Stores the component that is assigned as the Owner property of components read from the reader object's stream.

Pascal

```
property Owner: TComponent;
```

C++

```
__property TComponent * Owner;
```

Description

Owner is used internally by the component streaming system to determine which components to stream. The streaming process handles all components owned by the Root (see page 2840) component.

Owner is the Owner property of the component read from the stream. The component's Owner property is assigned its value from the reader's Owner property. If the value of the reader's Owner property is nil (Delphi) or NULL (C++), then the Root (see page 2840) property is assigned as the component's Owner property.

For most components, Owner is nil (Delphi) or NULL (C++). For components that can contain subcomponents that are not owned by the Root (see page 2840) form, such as the nodes of a tree view component, the Owner is not the same as the Root (see page 2840). The GetChildOwner method of the component determines the Owner.

See Also

Root (see page 2840)

Owner (see page 2807)

GetChildOwner (see page 2776)

Parent (see page 2959)

23.1.43.4.3 TReader.Parent Property

Stores the component that is assigned as the Parent property of controls read from the reader's stream.

Pascal

```
property Parent: TComponent;
```

C++

```
__property TComponent * Parent;
```

Description

Parent is used internally by the component streaming system. The value of the Parent property is determined by the order that components are read from the stream. The Parent relationship determines the visual containment of components.

See Also

GetChildren (see page 2777)

GetChildren (see page 6153)

GetChildren

Owner (see page 2959)

Parent (see page 5832)

23.1.43.4.4 TReader.Position Property

Represents the current reading position in the associated stream.

Pascal

```
property Position: Longint;
```

C++

```
__property Longint Position;
```

Description

Position is used to track the reader's position within the stream. The value of Position will always be inside the most recent buffer block read. Thus, for reading, Position will always be less than the stream's Position.

See Also

FlushBuffer (see page 2943)

Position (see page 2996)

23.1.43.4.5 TReader.PropName Property

Pascal

```
property PropName: string;
```

C++

```
__property AnsiString PropName;
```

Description

This is PropName, a member of class TReader.

23.1.43.5 TReader Events

The events of the TReader class are listed here.

Public Events

	Name	Description
	OnAncestorNotFound (see page 2961)	Occurs when the object from which a streamed object is derived cannot be found.
	OnCreateComponent (see page 2961)	Occurs when the reader has read a component's class but has not yet instantiated the component.
	OnError (see page 2961)	Occurs when a reader object encounters an error reading its data, such as reading the name of an undeclared property or an illegal value.
	OnFindComponentClass (see page 2962)	Occurs when a reader first reads the name of a component class.
	OnFindComponentInstance (see page 2962)	This is OnFindComponentInstance, a member of class TReader.
	OnFindMethod (see page 2962)	Occurs each time the reader object reads an event, which is a method-pointer property, for an object.
	OnFindMethodInstance (see page 2963)	This is OnFindMethodInstance, a member of class TReader.
	OnReferenceName (see page 2963)	Occurs when resolving references to other objects such as when one object is the property value of another.

	OnSetName (see page 2963)	Occurs just before the reader object sets the Name property of a component it reads from its stream.

23.1.43.5.1 TReader.OnAncestorNotFound Event

Occurs when the object from which a streamed object is derived cannot be found.

Pascal

```
property OnAncestorNotFound: TAncestorNotFoundEvent;
```

C++

```
__property TAncestorNotFoundEvent OnAncestorNotFound;
```

Description

OnAncestorNotFound is for internal use.

See Also

Ancestor (see page 2839)

23.1.43.5.2 TReader.OnCreateComponent Event

Occurs when the reader has read a component's class but has not yet instantiated the component.

Pascal

```
property OnCreateComponent: TCreateComponentEvent;
```

C++

```
__property TCreateComponentEvent OnCreateComponent;
```

Description

OnCreateComponent is used internally to provide an alternate mechanism for instantiating components. If OnCreateComponent returns a component as the value of the Component parameter, the Reader uses that instance instead of calling the component's constructor.

See Also

ReadComponent (see page 2945)

OnFindComponentClass (see page 2962)

23.1.43.5.3 TReader.OnError Event

Occurs when a reader object encounters an error reading its data, such as reading the name of an undeclared property or an illegal value.

Pascal

```
property OnError: TReaderError;
```

C++

```
__property TReaderError OnError;
```

Description

OnError is used internally by the IDE to report errors. It can also be used to write an event handler to selectively choose to process or ignore errors.

The last parameter passed to the OnError event handler, Handled, is passed by reference. By default, the Error (see page

2942) method passes false in Handled, but a handler that corrects the error or chooses to ignore the error can set Handled to true, which prevents further processing of the error. If the event handler returns with Handler still set to false, the reader object raises an EReadError (see page 2703) exception.

See Also

EReadError (see page 2703)

23.1.43.5.4 TReader.OnFindComponentClass Event

Occurs when a reader first reads the name of a component class.

Pascal

```
property OnFindComponentClass: TFindComponentClassEvent;
```

C++

```
__property TFindComponentClassEvent OnFindComponentClass;
```

Description

The ReadComponent (see page 2945) method generates an OnFindComponentClass event when it reads the class name for the component it is loading. An OnFindComponentClass event handler can replace the class reference with a different class to change the type of component that gets loaded.

See Also

ReadComponent (see page 2945)

OnCreateComponent (see page 2961)

23.1.43.5.5 TReader.OnFindComponentInstance Event

Pascal

```
property OnFindComponentInstance: TFindComponentInstanceEvent;
```

C++

```
__property TFindComponentInstanceEvent OnFindComponentInstance;
```

Description

This is OnFindComponentInstance, a member of class TReader.

23.1.43.5.6 TReader.OnFindMethod Event

Occurs each time the reader object reads an event, which is a method-pointer property, for an object.

Pascal

```
property OnFindMethod: TFindMethodEvent;
```

C++

```
__property TFindMethodEvent OnFindMethod;
```

Description

OnFindMethod is used internally by the IDE for reading events. It can also be used to write a handler to ask the user how to handle missing methods.

If the reader cannot locate the named method assigned to the method pointer, it sets the Error (see page 2942) parameter of the OnFindMethod handler to true. The handler can set Error (see page 2942) to false, which prevents FindMethod (see

page 2942) from raising an exception when the handler returns.

See Also

EReadError (see page 2703)

23.1.43.5.7 TReader.OnFindMethodInstance Event

Pascal

```
property OnFindMethodInstance: TFindMethodInstanceEvent;
```

C++

```
__property TFindMethodInstanceEvent OnFindMethodInstance;
```

Description

This is OnFindMethodInstance, a member of class TReader.

23.1.43.5.8 TReader.OnReferenceName Event

Occurs when resolving references to other objects such as when one object is the property value of another.

Pascal

```
property OnReferenceName: TReferenceNameEvent;
```

C++

```
__property TReferenceNameEvent OnReferenceName;
```

Description

OnReferenceName is for internal use only.

See Also

Name (see page 2805)

23.1.43.5.9 TReader.OnSetName Event

Occurs just before the reader object sets the Name property of a component it reads from its stream.

Pascal

```
property OnSetName: TSetNameEvent;
```

C++

```
__property TSetNameEvent OnSetName;
```

Description

OnSetName is used by internally by the IDE to handle duplicate names of components and forms. It can also be used to write an event handler to perform special processing just before the Name property is set for the component which has just been created and is about to have its properties read from the stream.

The Name parameter to the OnSetName event handler passed by reference, so the handler can change the name before the reader assigns it to the component. For example, OnSetName is useful for filtering all the component names in a form, to add or change part of the string.

See Also

Name (see page 2805)

23.1.44 TRecall Class

TRecall stores the properties of a persistent object so that they can be restored at a later time.

Class Hierarchy

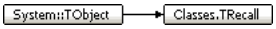

Pascal

```
TRecall = class(TObject);
```

C++

```
class TRecall : public TObject;
```

File

Classes

Description

TRecall acts as a temporary repository for the properties of a persistent object. To use TRecall

1. Create (see page 2965) an instance of TRecall, assigning an object to use for storing property values (the storage object), and an object whose property values it represents (the reference object). When you create an instance of TRecall, it automatically stores the current properties of the reference object.
2. Call the Store (see page 2967) method at any time to take a snapshot of the reference object's properties. TRecall updates the storage object so that it reflects only the property settings from the last time you called the Store (see page 2967) method (or, if Store (see page 2967) was never called, from the point when the TRecall object was created).
3. Delete the TRecall object to restore the reference object to the set of properties it had when you last called the Store (see page 2967) method.

If you do not want the TRecall object to restore the properties it is saving, call the Forget (see page 2967) method. After you call Forget (see page 2967), the TRecall object can't be used. It does not update the reference object when destroyed and can't save any more properties.

You can use TRecall to save the properties of any persistent object, using any explicitly specified storage object. In addition, TRecall has several descendants with their own built-in storage objects that work only with a specific class of reference object. These include TFontRecall, TPenRecall, and TBrushRecall.

See Also

TPersistent (see page 2928)

TBrushRecall (see page 14439)

TFontRecall (see page 14514)

TPenRecall (see page 14582)

23.1.44.1 TRecall Members

The following tables list the members exposed by TRecall.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2965)	Creates an instance of a TRecall (see page 2964) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2966)	Destroys the TRecall (see page 2964) instance, assigning the stored properties to the reference object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Forget (see page 2967)	Discards the saved property values and prevents the TRecall (see page 2964) object from updating the reference object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Store (see page 2967)	Saves the current property settings of the reference object.

Public Properties

	Name	Description
R	Reference (see page 2968)	Indicates the reference object that is updated when you destroy the TRecall (see page 2964) instance.

23.1.44.2 TRecall.Create Constructor

Creates an instance of a TRecall (see page 2964) object.

Pascal

```
constructor Create(AStorage: TPersistent; AReference: TPersistent);
```

C++

```
__fastcall TRecall(TPersistent * AStorage, TPersistent * AReference);
```

Description

Note: Call Create to instantiate a TRecall (see page 2964) object to save the properties of a specific reference object.

AStorage is an object that saves the properties of the reference object. Once you create the TRecall (see page 2964) instance, it is responsible for the memory of AStorage, and frees AStorage when it is destroyed. Do not free AStorage yourself after creating the TRecall (see page 2964) object.

AReference is the object whose properties are saved when you create the TRecall (see page 2964) object and, subsequently, any time you call the Store (see page 2967) method.

AStorage and AReference do not need to be the same type of object. However, TRecall (see page 2964) can only save and restore those properties that can be assigned from the reference object to the storage object and back again. That is, TRecall (see page 2964) only handles those properties that are copied by the Assign method of AStorage and AReference.

See Also

Destroy (see page 2966)

Reference (see page 2968)

Store (see page 2967)

Assign (see page 2930)

23.1.44.3 TRecall.Destroy Destructor

Destroys the TRecall (see page 2964) instance, assigning the stored properties to the reference object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRecall();
```

Description

Destroy a TRecall (see page 2964) instance to restore the properties of the reference object to the last values saved to the TRecall (see page 2964) object. The properties of the reference object are saved to the TRecall (see page 2964) instance when you first create it, and subsequently any time you call the Store (see page 2967) method. The TRecall (see page 2964) destructor assigns the currently saved property values to the reference object by calling the reference object's Assign method.

If you do not want to restore the currently saved property values, call the Forget (see page 2967) method before freeing the TRecall (see page 2964) object. Once you call Forget (see page 2967), TRecall (see page 2964) ceases to store any property information, and destroying the TRecall (see page 2964) leaves the properties of the reference object unchanged.

Note: Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

Free (see page 22300)

Reference (see page 2968)

Store (see page 2967)

Forget (see page 2967)

Assign (see page 2930)

23.1.44.4 TRecall Methods

The methods of the TRecall class are listed here.

Public Methods

	Name	Description
	Forget (see page 2967)	Discards the saved property values and prevents the TRecall (see page 2964) object from updating the reference object.
	Store (see page 2967)	Saves the current property settings of the reference object.

23.1.44.4.1 TRecall.Forget Method

Discards the saved property values and prevents the TRecall (see page 2964) object from updating the reference object.

Pascal

```
procedure Forget;
```

C++

```
__fastcall Forget();
```

Description

By default, the TRecall (see page 2964) destructor updates the reference object, assigning the most recently saved property values. To prevent this from happening, call Forget before destroying the TRecall (see page 2964) object.

Once you have called Forget, the TRecall (see page 2964) instance can not do anything. It ignores any further attempts to store property settings because it can't use them to update the reference object when it is destroyed.

See Also

Reference (see page 2968)

Destroy (see page 2966)

23.1.44.4.2 TRecall.Store Method

Saves the current property settings of the reference object.

Pascal

```
procedure Store;
```

C++

```
__fastcall Store();
```

Description

Call Store to update the TRecall (see page 2964) object's saved property values so that they reflect the current property values of the reference object. Each call to Store completely supersedes the previously stored property values.

When you destroy the TRecall (see page 2964) object, it assigns the most recently stored property values to the reference object.

See Also

Forget (see page 2967)

Reference (see page 2968)

Create (see page 2965)

23.1.44.5 TRecall Properties

The properties of the TRecall class are listed here.

Public Properties

	Name	Description
R	Reference (see page 2968)	Indicates the reference object that is updated when you destroy the TRecall (see page 2964) instance.

23.1.44.5.1 TRecall.Reference Property

Indicates the reference object that is updated when you destroy the TRecall (see page 2964) instance.

Pascal

```
property Reference: TPersistent;
```

C++

```
__property TPersistent * Reference;
```

Description

Read Reference to determine the object that TRecall (see page 2964) updates when it is destroyed. The value of this read-only property is assigned in the constructor.

See Also

Destroy (see page 2966)

23.1.45 TResourceStream Class

TResourceStream is a memory stream that provides access to the compiled resources in an application.

Class Hierarchy

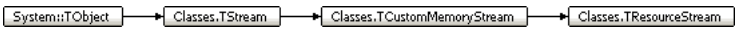

Pascal

```
TResourceStream = class(TCustomMemoryStream);
```

C++

```
class TResourceStream : public TCustomMemoryStream;
```

File

Classes

Description

Use TResourceStream to read the resources of an application. An instance of TResourceStream holds the value of a single resource in a memory buffer where it is accessible to the application.

The global ReadComponentRes (see page 3245) function uses TResourceStream to access the compiled resources used by the application.

See Also

ReadComponentRes (see page 3245)

TStream (see page 2973)

23.1.45.1 TResourceStream Members

The following tables list the members exposed by TResourceStream.

Protected Methods

	Name	Description
	GetSize (see page 2982)	This is GetSize, a member of class TStream.
	SetPointer (see page 2823)	Replaces the memory buffer associated with the memory stream.
	SetSize (see page 2989)	This is the overview for the SetSize method overload.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CopyFrom (see page 2979)	Copies a specified number of bytes from one stream to another.
	Create (see page 2970)	Creates an instance of TResourceStream (see page 2968) associated with a particular resource name and type.
	CreateFromID (see page 2971)	Creates an instance of TResourceStream (see page 2968) associated with a particular resource ID and type.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2972)	Destroys an instance of TResourceStream (see page 2968).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FixupResourceHeader (see page 2981)	Patches the resource header for a resource that has been written to the stream.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Read (see page 2817)	Reads up to Count bytes from the memory stream into Buffer and advances the current position of the stream by the number of bytes read.
	Read (see page 2982)	Introduces an abstract or pure virtual method responsible for reading from the stream.
	ReadBuffer (see page 2983)	Reads bytes from the stream into Buffer.
	ReadComponent (see page 2983)	Initiates streaming of components and their properties.

	ReadComponentRes (see page 2987)	Reads components and their properties from a stream in a resource file format.
	ReadResHeader (see page 2987)	Reads a resource-file header from the stream.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 2818)	Writes the entire contents of the memory stream to the file with a given file name.
	SaveToStream (see page 2818)	Writes the entire contents of the memory stream to the stream object specified by Stream.
V	Seek (see page 2819)	Moves the current position of the stream by Offset bytes, relative to the origin specified by Origin.
V	Seek (see page 2988)	This is the overview for the Seek method overload.
V	Write (see page 2973)	Overrides the inherited method to raise an exception when an attempt is made to write the resource back to the application.
	WriteBuffer (see page 2990)	Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.
	WriteComponent (see page 2990)	Initiates the writing of components and their properties to a stream.
	WriteComponentRes (see page 2994)	Writes components and their properties to a stream from a resource file format.
	WriteDescendent (see page 2994)	Streams components and their properties in inherited forms.
	WriteDescendentRes (see page 2995)	Streams components and their properties in inherited forms.
	WriteResourceHeader (see page 2995)	Writes a resource-file header to the stream.

Public Properties

	Name	Description
R	Memory (see page 2823)	Provides direct access to the memory pool allocated for the memory stream.
	Position (see page 2996)	Indicates the current offset into the stream for reading and writing.
	Size (see page 2997)	Indicates the size in bytes of the stream.

23.1.45.2 TResourceStream.Create Constructor

Creates an instance of TResourceStream (see page 2968) associated with a particular resource name and type.

Pascal

```
constructor Create(Instance: THandle; const ResName: string; ResType: PChar);
```

C++

```
__fastcall TResourceStream(THandle Instance, const AnsiString ResName, const char * ResType);
```

Description

Call Create to instantiate a TResourceStream (see page 2968), passing in parameters that identify the resource in a specified instance. TResourceStream (see page 2968) finds the resource data and loads it into the Memory (see page 2823) buffer for the TResourceStream (see page 2968).

The Instance parameter is the instance handle associated with the executable or shared library that contains the resource.

In the first constructor syntax, the ResName is the string associated with the resource in the .rc file that was compiled with the application. If the resource is associated with an integer ID rather than a string, use the string representation of the integer after a pound sign. Thus, for example, a resource with an integer identifier of 128 be identified by a ResName of #128.

Note: Specifying resources by ID requires less memory than specifying resources by name.

The ResType parameter is a string identifying the type of the resource. Applications can define their own resource types and identify them by name in the .rc file. In addition, there are a number of predefined resource types (which reflect Windows resource types). To identify a resource that is one of the predefined resource types, set ResType to the appropriate value from the following table:

ResType	Type of resource
RT_ACCELERATOR	Accelerator table

RT_BITMAP	Bitmap resource
RT_DIALOG	Dialog box
RT_FONT	Font resource
RT_FONTDIR	Font directory resource
RT_MENU	Menu resource
RT_RCDATA	Application-defined resource (raw data)
RT_STRING	String-table entry
RT_MESSAGETABLE	Message-table entry
RT_CURSOR	Hardware-dependent cursor resource
RT_GROUP_CURSOR	Hardware-independent cursor resource
RT_ICON	Hardware-dependent icon resource
RT_GROUP_ICON	Hardware-independent icon resource
RT_VERSION	Version resource

See Also

CreateFromID (see page 2971)

Memory (see page 2823)

23.1.45.3 TResourceStream.CreateFromID Constructor

Creates an instance of TResourceStream (see page 2968) associated with a particular resource ID and type.

Pascal

```
constructor CreateFromID(Instance: THandle; ResID: Integer; ResType: PChar);
```

C++

```
__fastcall TResourceStream(THandle Instance, int ResID, const char * ResType);
```

Description

Call CreateFromID to instantiate a TResourceStream (see page 2968) for a resource in the indicated instance that can be identified by an integer ID. Create (see page 2970) finds the resource data and loads it into the Memory (see page 2823) buffer for the TResourceStream (see page 2968).

The Instance parameter is the instance handle associated with the executable or shared library that contains the resource.

The ResID is the integer ID associated with the resource in the .rc file that was compiled with the application. If the Resource was identified by a string rather than an ID number, use the Create (see page 2970) method instead.

The ResType parameter is a string identifying the type of the resource. Applications can define their own resource types and identify them by name in the .rc file. In addition, there are a number of predefined resource types (which reflect Windows resource types). To identify a resource that is one of the predefined resource types, set ResType to the appropriate value from the following table:

ResType	Type of resource
RT_ACCELERATOR	Accelerator table
RT_BITMAP	Bitmap resource
RT_DIALOG	Dialog box

RT_FONT	Font resource
RT_FONTDIR	Font directory resource
RT_MENU	Menu resource
RT_RCDATA	Application-defined resource (raw data)
RT_STRING	String-table entry
RT_MESSAGETABLE	Message-table entry
RT_CURSOR	Hardware-dependent cursor resource
RT_GROUP_CURSOR	Hardware-independent cursor resource
RT_ICON	Hardware-dependent icon resource
RT_GROUP_ICON	Hardware-independent icon resource
RT_VERSION	Version resource

Note: Specifying resources by ID requires less memory than specifying resources by name.

See Also

Create (see page 2970)

Memory (see page 2823)

23.1.45.4 TResourceStream.Destroy Destructor

Destroys an instance of TResourceStream (see page 2968).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TResourceStream();
```

Description

Do not call Destroy directly in an application to destroy an instance of TResourceStream (see page 2968). Instead, call Free. Free verifies that the TResourceStream (see page 2968) reference is not nil, and only then calls Destroy.

Destroy frees the memory buffer that stores the resource before calling the inherited destructor.

See Also

Create (see page 2970)

Free (see page 22300)

Memory (see page 2823)

23.1.45.5 TResourceStream Methods

The methods of the TResourceStream class are listed here.

Public Methods

	Name	Description
	Write (see page 2973)	Overrides the inherited method to raise an exception when an attempt is made to write the resource back to the application.

23.1.45.5.1 TResourceStream.Write Method

Overrides the inherited method to raise an exception when an attempt is made to write the resource back to the application.

Pascal

```
function Write(const Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  Buffer, Longint Count);
```

Description

Applications should not use a TResourceStream (see page 2968) to write the resources of the running application. Write overrides the inherited method to raise an EStreamError (see page 2704) exception when an application tries to write to the application's resources.

As all other data-writing methods of TResourceStream (see page 2968) (WriteBuffer (see page 2990), WriteComponent (see page 2990)) call Write to do the actual writing, calling any of the data-writing methods of TResourceStream (see page 2968) will raise an exception.

See Also

EStreamError (see page 2704)

WriteBuffer (see page 2990)

WriteComponent (see page 2990)

23.1.46 TStream Class

TStream is the base class type for stream objects that can read from or write to various kinds of storage media, such as disk files, dynamic memory, and so on.

Class Hierarchy

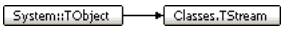

Pascal

```
TStream = class(TObject);
```

C++

```
class TStream : public TObject;
```

File

Classes

Description

Use specialized stream objects to read from, write to, or copy information stored in a particular medium. Each descendant of TStream implements methods for transferring information to and from a particular storage medium, such as a disk file, dynamic memory, and so on. In addition to methods for reading, writing, and copying bytes to and from the stream, stream objects permit

applications to seek to an arbitrary position in the stream. Properties of TStream provide information about the stream, such as its size and the current position in the stream.

TStream also introduces methods that work in conjunction with components and filers for loading and saving components in simple and inherited forms. These methods are called automatically by global routines that initiate component streaming. They can also be called directly to initiate the streaming process. Note, however, that component streaming always involves two additional objects:

A component object that is passed as a parameter to the stream's methods.

A filer object that is automatically created by the stream, and associated with the stream.

TStream is an abstract or, in C++ terminology, pure virtual class. It should not be instantiated; it relies on abstract or pure virtual methods that must be overridden in descendant classes. Descendant stream objects, such as memory and file streams used for component streaming, are created automatically by the global functions ReadComponentRes (see page 2987) and WriteComponentRes (see page 2994). For streaming other kinds of information, choose a descendant class according to the specific data and storage needs. These include

TFileStream (see page 2841) (for working with files)

TStringStream (see page 3088) (for manipulating in-memory strings)

TMemoryStream (see page 2902) (for working with a memory buffer)

TBlobStream (for working with BLOB fields)

TWinSocketStream (for reading and writing over a socket connection)

TOleStream (for using a COM interface to read and write)

C++ Examples:

```
/*
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
*/
class MyScrollBar : public TScrollBar
{
__published:    // IDE-managed Components
private:    // User declarations
public:     // User declarations
    __fastcall MyScrollBar(TComponent* Owner);
};

__fastcall MyScrollBar::MyScrollBar(TComponent* Owner)
    : TScrollBar(Owner)
{
}

void RegisterClassesWithStreamingSystem(void)
{
  // Make sure that as part of the startup
  // code our streaming classes are registered
  // with the streaming system.

  #pragma startup RegisterClassesWithStreamingSystem

  Classes::RegisterClass(__classid(MyScrollBar));
}

MyScrollBar *ScrollBar1;

AnsiString __fastcall ComponentToString(TComponent *c)
{
```

```
 AnsiString as;
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);

 try
 {
    pms->WriteComponent(c);
    pms->Seek(0, soFromBeginning);
    ObjectBinaryToText(pms, pss);
    pss->Seek(0, soFromBeginning);
    as = pss->DataString;
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 delete pms;
 delete pss;
 return as;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Memo1->Text = ComponentToString(ScrollBar1);
}

TComponent *__fastcall StringToComponent(AnsiString as)

{
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);
 TComponent *pc;

 try
 {
    ObjectTextToBinary(pss, pms);
    pms->Seek(0, soFromBeginning);
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 pc = pms->ReadComponent(NULL);
 delete pms;
 delete pss;
 return pc;
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  TComponent *temp = StringToComponent(Memo1->Text);
  ScrollBar1->Free();

  ScrollBar1 = dynamic_cast<MyScrollBar *>(temp);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ScrollBar1 = new MyScrollBar(Form1);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
  ScrollBar1->Top = 48;
```

```
  ScrollBar1->Left = 250;
  ScrollBar1->Name = "Ricksbar";
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete ScrollBar1;
}
```

Delphi Examples:

```
{
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
}
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  MyScrollBar = class(TScrollBar)
  end;

var
  Form1: TForm1;
  ScrollBar1: MyScrollBar;

implementation

{$R *.dfm}

function ComponentToStringProc(Component: TComponent): string;
var
  BinStream:TMemoryStream;
  StrStream: TStringStream;
  s: string;
begin
  BinStream := TMemoryStream.Create;
  try
    StrStream := TStringStream.Create(s);
    try
      BinStream.WriteComponent(Component);
      BinStream.Seek(0, soFromBeginning);
      ObjectBinaryToText(BinStream, StrStream);
      StrStream.Seek(0, soFromBeginning);
      Result:= StrStream.DataString;
    finally
      StrStream.Free;
    end;
  finally
    BinStream.Free
  end;
end;

function StringToComponentProc(Value: string): TComponent;
var
  StrStream:TStringStream;
```

```
    BinStream: TMemoryStream;
  begin
    StrStream := TStringStream.Create(Value);
    try
      BinStream := TMemoryStream.Create;
      try
        ObjectTextToBinary(StrStream, BinStream);
        BinStream.Seek(0, soFromBeginning);
        Result:= BinStream.ReadComponent(nil);
      finally
        BinStream.Free;
      end;
    finally
      StrStream.Free;
    end;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Memo1.Text:= ComponentToStringProc(ScrollBar1);
  end;

  // Edit the scrollbar parameters in the memo, click this
  // button and see the scrollbar change.
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    ScrollBar1.Free;
    ScrollBar1:= (StringToComponentProc(Memo1.Text) as MyScrollBar);
    ScrollBar1.Parent:= Form1;
    ScrollBar1.Visible:= TRUE;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ScrollBar1:= MyScrollBar.Create(Form1);
    ScrollBar1.Parent:= Form1;
    ScrollBar1.Visible:= TRUE;
    ScrollBar1.Top:= 48;
    ScrollBar1.Left:= 250;
    ScrollBar1.Name:= 'Ricksbar';
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    ScrollBar1.Free;
  end;

  initialization

  RegisterClass(MyScrollBar);
```

See Also

ReadComponentRes (see page 3245)

ReadComponentRes (see page 3245)

TComponent (see page 2762)

TFiler (see page 2834)

TStrings (see page 3029)

Using Streams

Using Streams to Read or Write Data

23.1.46.1 TStream Members

The following tables list the members exposed by TStream.

Protected Methods

	Name	Description
V	GetSize (see page 2982)	This is GetSize, a member of class TStream.
V	SetSize (see page 2989)	This is the overview for the SetSize method overload.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CopyFrom (see page 2979)	Copies a specified number of bytes from one stream to another.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FixupResourceHeader (see page 2981)	Patches the resource header for a resource that has been written to the stream.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
A	Read (see page 2982)	Introduces an abstract or pure virtual method responsible for reading from the stream.
	ReadBuffer (see page 2983)	Reads bytes from the stream into Buffer.
	ReadComponent (see page 2983)	Initiates streaming of components and their properties.
	ReadComponentRes (see page 2987)	Reads components and their properties from a stream in a resource file format.
	ReadResHeader (see page 2987)	Reads a resource-file header from the stream.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Seek (see page 2988)	This is the overview for the Seek method overload.
A	Write (see page 2989)	Introduces a pure virtual method for writing to the stream.
	WriteBuffer (see page 2990)	Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.

	WriteComponent (see page 2990)	Initiates the writing of components and their properties to a stream.
	WriteComponentRes (see page 2994)	Writes components and their properties to a stream from a resource file format.
	WriteDescendent (see page 2994)	Streams components and their properties in inherited forms.
	WriteDescendentRes (see page 2995)	Streams components and their properties in inherited forms.
	WriteResourceHeader (see page 2995)	Writes a resource-file header to the stream.

Public Properties

	Name	Description
	Position (see page 2996)	Indicates the current offset into the stream for reading and writing.
	Size (see page 2997)	Indicates the size in bytes of the stream.

23.1.46.2 TStream Methods

The methods of the TStream class are listed here.

Protected Methods

	Name	Description
V	GetSize (see page 2982)	This is GetSize, a member of class TStream.
V	SetSize (see page 2989)	This is the overview for the SetSize method overload.

Public Methods

	Name	Description
	CopyFrom (see page 2979)	Copies a specified number of bytes from one stream to another.
	FixupResourceHeader (see page 2981)	Patches the resource header for a resource that has been written to the stream.
A	Read (see page 2982)	Introduces an abstract or pure virtual method responsible for reading from the stream.
	ReadBuffer (see page 2983)	Reads bytes from the stream into Buffer.
	ReadComponent (see page 2983)	Initiates streaming of components and their properties.
	ReadComponentRes (see page 2987)	Reads components and their properties from a stream in a resource file format.
	ReadResHeader (see page 2987)	Reads a resource-file header from the stream.
V	Seek (see page 2988)	This is the overview for the Seek method overload.
A	Write (see page 2989)	Introduces a pure virtual method for writing to the stream.
	WriteBuffer (see page 2990)	Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.
	WriteComponent (see page 2990)	Initiates the writing of components and their properties to a stream.
	WriteComponentRes (see page 2994)	Writes components and their properties to a stream from a resource file format.
	WriteDescendent (see page 2994)	Streams components and their properties in inherited forms.
	WriteDescendentRes (see page 2995)	Streams components and their properties in inherited forms.
	WriteResourceHeader (see page 2995)	Writes a resource-file header to the stream.

23.1.46.2.1 TStream.CopyFrom Method

Copies a specified number of bytes from one stream to another.

Pascal

```
function CopyFrom(Source: TStream; Count: Int64): Int64;
```

C++

```
__fastcall Int64 CopyFrom(TStream Source, Int64 Count);
```

Description

Use CopyFrom to copy data to the stream from a different stream. Using CopyFrom eliminates the need for the user to create, read into, write from, and free a buffer when copying data.

CopyFrom copies Count bytes from the stream specified by Source into the stream. It then moves the current position by Count bytes, and returns the number of bytes copied.

If Count is 0, CopyFrom sets Source position to 0 before reading and then copies the entire contents of Source into the stream. If Count is greater than or less than 0, CopyFrom reads from the current position in Source.

C++ Examples:

```
/*
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Files
can be saved to and from the app directory by using relative
paths.
*/
void __fastcall TForm1::Save1Click(TObject *Sender)
{
  AnsiString NewFileName =
    ExtractFilePath(Application->ExeName) +
    ExtractFileName(Edit1->Text);
  AnsiString OldFileName =
    ExtractFilePath(Application->ExeName) +
    ExtractFileName(Edit2->Text);
  AnsiString Msg = Format("Copy %s to %s", ARRAYOFCONST((OldFileName, NewFileName)));
  if (MessageDlg(Msg, mtCustom, TMsgDlgButtons() << mbOK << mbCancel, 0) == mrOk)
  {
    TFileStream *OldFile = new TFileStream(OldFileName, fmOpenRead);
    try
    {
      TFileStream *NewFile = new TFileStream(NewFileName, fmCreate);
      try
      {
        NewFile->CopyFrom(OldFile, OldFile->Size);
      }
      __finally
      {
        FreeAndNil(&NewFile);
      }
    }
    __finally
    {
      FreeAndNil(&OldFile);
    }
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Label3->Caption = "Current directory: " + ExtractFilePath(Application->ExeName);
}
```

Delphi Examples:

```
{
The following example copies the data in the Notes field of
Table1 or SQLDataSet1 to the Remarks field of ClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Stream1: TBlobStream;
  Stream2: TStream;
  blobType : TBlobType;
begin
  blobType := Table1Notes.BlobType;
  Stream1 := TBlobStream.Create(Table1Notes, bmRead);
  try
```

```
      CDS2.Edit;
      { here's a different way to create a blob stream }
      Stream2 := CDS2.CreateBlobStream(CDS2.FieldByName('Remarks'), bmReadWrite);
      try
        Stream2.CopyFrom(Stream1, Stream1.Size);
//      CDS2.Post;
//      CDS2.Active := True;
      finally
        Stream2.Free;
      end;
    finally
      Stream1.Free;
    end;
  end;
  {
  This example copies a specified old file to a new file.  Add
  a TSaveDialog to the form.  Also add two TEdits, two TLabels
  and a TButton with the OnClick event named Save1Click. Files
  can be saved to and from the app directory by using relative
  paths.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Label3.Caption:= 'Current directory: ' + ExtractFilePath(Application.ExeName);
  end;

  procedure TForm1.Save1Click(Sender: TObject);
  var
    NewFileName, OldFileName: string;
    Msg: string;
    NewFile: TFileStream;
    OldFile: TFileStream;
  begin
    NewFileName :=
      ExtractFilePath(Application.ExeName) +
      ExtractFileName(Edit1.Text);
    OldFileName :=
      ExtractFilePath(Application.ExeName) +
      ExtractFileName(Edit2.Text);
    Msg := Format('Copy %s to %s?', [OldFileName, NewFileName]);
    if MessageDlg(Msg, mtCustom, mbOKCancel, 0) = mrOK then
    begin
      OldFile := TFileStream.Create(
        OldFileName, fmOpenRead or fmShareDenyWrite);
      try
        NewFile := TFileStream.Create(
          NewFileName, fmCreate or fmShareDenyRead);
        try
          NewFile.CopyFrom(OldFile, OldFile.Size);
        finally
          FreeAndNil(NewFile);
        end;
      finally
        FreeAndNil(OldFile);
      end;
    end;
  end;
```

23.1.46.2.2 TStream.FixupResourceHeader Method

Patches the resource header for a resource that has been written to the stream.

Pascal

```
procedure FixupResourceHeader(FixupInfo: Integer);
```

C++

```
__fastcall FixupResourceHeader(int FixupInfo);
```

Description

Do not call FixupResourceHeader, it is used internally. The WriteDescendentRes (see page 2995) method calls FixupResourceHeader after writing a component to the stream. This method then uses the current position to determine the size of the resource just written and adjust the resource header accordingly.

The FixupInfo parameter is the value returned by WriteResourceHeader (see page 2995) when it writes the resource header (before WriteDescendentRes (see page 2995) streams out the component).

See Also

WriteResourceHeader (see page 2995)

WriteDescendentRes (see page 2995)

Position (see page 2996)

23.1.46.2.3 TStream.GetSize Method

Pascal

```
function GetSize: Int64; virtual;
```

C++

```
virtual __fastcall Int64 GetSize();
```

Description

This is GetSize, a member of class TStream.

23.1.46.2.4 TStream.Read Method

Introduces an abstract or pure virtual method responsible for reading from the stream.

Pascal

```
function Read(var Buffer; Count: Longint): Longint; virtual; abstract;
```

C++

```
virtual __fastcall Longint Read( Buffer, Longint Count) = 0;
```

Description

Each descendant stream object defines a Read method that reads data from its particular storage medium (such as memory or a disk file).

Read is used in cases where the number of bytes to read from the stream is not necessarily fixed. It attempts to read up to Count bytes into buffer and returns the number of bytes actually read.

All the other data-reading methods of a stream (ReadBuffer (see page 2983), ReadComponent (see page 2983)) call Read to do their actual reading.

See Also

EReadError (see page 2703)

ReadBuffer (see page 2983)

ReadComponent (see page 2983)

23.1.46.2.5 TStream.ReadBuffer Method

Reads bytes from the stream into Buffer.

Pascal

```
procedure ReadBuffer(var Buffer; Count: Longint);
```

C++

```
__fastcall ReadBuffer( Buffer, Longint Count);
```

Description

Use ReadBuffer to read Count bytes from the stream into a buffer in cases where the number of bytes is known and fixed, for example when reading in structures. ReadBuffer is used internally for loading from a stream and copying from a stream.

ReadBuffer calls Read (see page 2982) to do the actual reading. If Count bytes cannot be read from the stream, an EReadError (see page 2703) exception is raised.

See Also

EReadError (see page 2703)

Read (see page 2982)

23.1.46.2.6 TStream.ReadComponent Method

Initiates streaming of components and their properties.

Pascal

```
function ReadComponent(Instance: TComponent): TComponent;
```

C++

```
__fastcall TComponent * ReadComponent(TComponent * Instance);
```

Description

ReadComponent is called indirectly by the global routine ReadComponentRes (see page 2987), by the ReadComponentRes (see page 2987) method, or it can be called directly to initiate component streaming.

ReadComponent reads data values from the stream and assigns them to Instance's properties. It then constructs a reader object and calls the reader's ReadRootComponent method to read Instance's property values and construct child objects defined in the stream as children of the Instance. ReadComponent returns the component.

If Instance is nil (Delphi) or NULL (C++), ReadComponent constructs a component based on the type information in the stream and returns the newly-constructed component.

C++ Examples:

```
/*
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
*/
class MyScrollBar : public TScrollBar
{
__published:    // IDE-managed Components
private:    // User declarations
public:     // User declarations
    __fastcall MyScrollBar(TComponent* Owner);
};
```

```
__fastcall MyScrollBar::MyScrollBar(TComponent* Owner)
    : TScrollBar(Owner)
{
}

void RegisterClassesWithStreamingSystem(void)
{
  // Make sure that as part of the startup
  // code our streaming classes are registered
  // with the streaming system.

  #pragma startup RegisterClassesWithStreamingSystem

  Classes::RegisterClass(__classid(MyScrollBar));
}

MyScrollBar *ScrollBar1;

AnsiString __fastcall ComponentToString(TComponent *c)
{
 AnsiString as;
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);

 try
 {
    pms->WriteComponent(c);
    pms->Seek(0, soFromBeginning);
    ObjectBinaryToText(pms, pss);
    pss->Seek(0, soFromBeginning);
    as = pss->DataString;
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 delete pms;
 delete pss;
 return as;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Memo1->Text = ComponentToString(ScrollBar1);
}

TComponent *__fastcall StringToComponent(AnsiString as)

{
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);
 TComponent *pc;

 try
 {
    ObjectTextToBinary(pss, pms);
    pms->Seek(0, soFromBeginning);
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 pc = pms->ReadComponent(NULL);
 delete pms;
```

```
 delete pss;
 return pc;
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  TComponent *temp = StringToComponent(Memo1->Text);
  ScrollBar1->Free();

  ScrollBar1 = dynamic_cast<MyScrollBar *>(temp);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ScrollBar1 = new MyScrollBar(Form1);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
  ScrollBar1->Top = 48;
  ScrollBar1->Left = 250;
  ScrollBar1->Name = "Ricksbar";
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete ScrollBar1;
}
```

Delphi Examples:

```
{
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
}
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  MyScrollBar = class(TScrollBar)
  end;

var
  Form1: TForm1;
  ScrollBar1: MyScrollBar;

implementation

{$R *.dfm}

function ComponentToStringProc(Component: TComponent): string;
var
  BinStream:TMemoryStream;
  StrStream: TStringStream;
  s: string;
begin
```

```
    BinStream := TMemoryStream.Create;
    try
      StrStream := TStringStream.Create(s);
      try
        BinStream.WriteComponent(Component);
        BinStream.Seek(0, soFromBeginning);
        ObjectBinaryToText(BinStream, StrStream);
        StrStream.Seek(0, soFromBeginning);
        Result:= StrStream.DataString;
      finally
        StrStream.Free;
      end;
    finally
      BinStream.Free
    end;
  end;

  function StringToComponentProc(Value: string): TComponent;
  var
    StrStream:TStringStream;
    BinStream: TMemoryStream;
  begin
    StrStream := TStringStream.Create(Value);
    try
      BinStream := TMemoryStream.Create;
      try
        ObjectTextToBinary(StrStream, BinStream);
        BinStream.Seek(0, soFromBeginning);
        Result:= BinStream.ReadComponent(nil);
      finally
        BinStream.Free;
      end;
    finally
      StrStream.Free;
    end;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Memo1.Text:= ComponentToStringProc(ScrollBar1);
  end;

  // Edit the scrollbar parameters in the memo, click this
  // button and see the scrollbar change.
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    ScrollBar1.Free;
    ScrollBar1:= (StringToComponentProc(Memo1.Text) as MyScrollBar);
    ScrollBar1.Parent:= Form1;
    ScrollBar1.Visible:= TRUE;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ScrollBar1:= MyScrollBar.Create(Form1);
    ScrollBar1.Parent:= Form1;
    ScrollBar1.Visible:= TRUE;
    ScrollBar1.Top:= 48;
    ScrollBar1.Left:= 250;
    ScrollBar1.Name:= 'Ricksbar';
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    ScrollBar1.Free;
  end;
```

```
initialization
RegisterClass(MyScrollBar);
```

See Also

Read (see page 2982)

ReadComponentRes

ReadComponentRes (see page 2987)

ReadRootComponent (see page 2952)

WriteComponent (see page 2990)

Using Streams to Read or Write Data

23.1.46.2.7 TStream.ReadComponentRes Method

Reads components and their properties from a stream in a resource file format.

Pascal

```
function ReadComponentRes(Instance: TComponent): TComponent;
```

C++

```
__fastcall TComponent * ReadComponentRes(TComponent * Instance);
```

Description

ReadComponentRes is called automatically by the global routine ReadComponentResFile (see page 3247). It can also be called directly if the current position of the stream points to a component written using the WriteComponentRes (see page 2994) method.

ReadComponentResFile (see page 3247) creates a file stream object, which then calls its ReadComponentRes method.

ReadComponentRes first calls the ReadResHeader (see page 2987) method to read a resource header from the stream. If the stream does not contain a resource header at the current position, ReadResHeader (see page 2987) will raise an EInvalidImage (see page 2699) exception. ReadComponentRes then calls ReadComponent (see page 2983) to read the properties that must be set on Instance.

See Also

EInvalidImage (see page 2699)

ReadComponentResFile

ReadResHeader (see page 2987)

TResourceStream (see page 2968)

WriteComponentRes (see page 2994)

23.1.46.2.8 TStream.ReadResHeader Method

Reads a resource-file header from the stream.

Pascal

```
procedure ReadResHeader;
```

C++

```
__fastcall ReadResHeader();
```

Description

Do not call ReadResHeader directly. It is called automatically by ReadComponentRes (see page 2987) before reading a component from a resource file.

After reading the resource-file header, ReadResHeader moves the current position of the stream to just beyond the header. If the stream does not contain a valid resource-file header, ReadResHeader raises an EInvalidImage (see page 2699) exception.

See Also

EInvalidImage (see page 2699)

ReadComponentRes (see page 2987)

23.1.46.2.9 Seek Method

This is the overview for the Seek method overload.

Overload List

	Name	Description
	TStream.Seek (Int64, TSeekOrigin) (see page 2988)	Moves to a specified position in the streamed resource.
	TStream.Seek (Longint, Word) (see page 2988)	Moves to a specified position in the streamed resource.

23.1.46.2.9.1 TStream.Seek Method (Int64, TSeekOrigin)

Moves to a specified position in the streamed resource.

Pascal

```
function Seek(Offset: Longint; Origin: Word): Longint; virtual; overload;
function Seek(const Offset: Int64; Origin: TSeekOrigin): Int64; virtual; overload;
```

C++

```
virtual __fastcall Longint Seek(Longint Offset, Word Origin);
virtual __fastcall Int64 Seek(const Int64 Offset, TSeekOrigin Origin);
```

Description

Call Seek to move the current position of the stream in its particular storage medium (such as memory or a disk file).

The Origin parameter indicates how to interpret the Offset parameter. Origin should be one of the following values:

Value	Meaning
soBeginning	Offset is from the beginning of the resource. Seek moves to the position Offset. Offset must be >= 0.
soCurrent	Offset is from the current position in the resource. Seek moves to Position (see page 2996) + Offset.
soEnd	Offset is from the end of the resource. Offset must be <= 0 to indicate a number of bytes before the end of the file.

Seek returns the new value of the Position (see page 2996) property.

Seek is called by the Position (see page 2996) and Size (see page 2997) properties.

Note: As implemented in TStream (see page 2973), the two versions (the 32-bit or 64-bit syntax) call each other. Descendant

stream classes must override at least one of these versions, and the override must not call the inherited default implementation.

See Also

Position (see page 2996)

23.1.46.2.10 SetSize Method

This is the overview for the SetSize method overload.

Overload List

	Name	Description
	TStream.SetSize (Int64) (see page 2989)	Provides a placeholder for a method to change the size of the streamed resource.
	TStream.SetSize (Longint) (see page 2989)	Provides a placeholder for a method to change the size of the streamed resource.

23.1.46.2.10.1 TStream.SetSize Method (Int64)

Provides a placeholder for a method to change the size of the streamed resource.

Pascal

```
procedure SetSize(NewSize: Longint); virtual; overload;
procedure SetSize(const NewSize: Int64); virtual; overload;
```

C++

```
virtual __fastcall SetSize(Longint NewSize);
virtual __fastcall SetSize(const Int64 NewSize);
```

Description

SetSize does nothing in TStream (see page 2973). Descendant stream classes can override this method to set the Size (see page 2997) property of their objects. SetSize is the write procedure for the Size (see page 2997) property.

Note: SetSize is introduced in TStream (see page 2973), even though it does nothing, because it is not possible to change the definition of a property in a descendant class. SetSize must therefore be introduced when Size (see page 2997) is introduced.

23.1.46.2.11 TStream.Write Method

Introduces a pure virtual method for writing to the stream.

Pascal

```
function Write(const Buffer; Count: Longint): Longint; virtual; abstract;
```

C++

```
virtual __fastcall Longint Write(const  Buffer, Longint Count) = 0;
```

Description

Each descendant stream object defines a Write method that writes data to its particular storage medium (such as memory or a disk file). Write attempts to write up to Count bytes from Buffer, and returns the number of bytes actually written.

All the other data-writing methods of a stream (WriteBuffer (see page 2990), WriteComponent (see page 2990)) call Write to do their actual writing.

See Also

WriteBuffer (see page 2990)

WriteComponent (see page 2990)

23.1.46.2.12 TStream.WriteBuffer Method

Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.

Pascal

```
procedure WriteBuffer(const Buffer; Count: Longint);
```

C++

```
__fastcall WriteBuffer(const  Buffer, Longint Count);
```

Description

Use WriteBuffer to save data to a stream. WriteBuffer and ReadBuffer (see page 2983) are used in cases where the number of bytes is known and required, for example when reading in structures. Use WriteBuffer for standard file I/O streaming.

WriteBuffer is used internally for writing to a stream and copying from a stream. It is used by other objects, such as strings and lists, for writing strings stored in a buffer.

WriteBuffer calls Write (see page 2989) to handle the actual writing. If the stream fails to write all the requested bytes, an EWriteError (see page 2706) exception is raised.

See Also

EWriteError (see page 2706)

ReadBuffer (see page 2983)

Write (see page 2989)

23.1.46.2.13 TStream.WriteComponent Method

Initiates the writing of components and their properties to a stream.

Pascal

```
procedure WriteComponent(Instance: TComponent);
```

C++

```
__fastcall WriteComponent(TComponent * Instance);
```

Description

WriteComponent is used internally in the component streaming system, but can also be called directly when writing components to memory streams or database blobs.

WriteComponent constructs a writer object and calls its WriteRootComponent method to write the component specified by Instance, and its owned objects, to the stream.

C++ Examples:

```
/*
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
*/
class MyScrollBar : public TScrollBar
```

```
{
__published:    // IDE-managed Components
private:    // User declarations
public:     // User declarations
    __fastcall MyScrollBar(TComponent* Owner);
};

__fastcall MyScrollBar::MyScrollBar(TComponent* Owner)
    : TScrollBar(Owner)
{
}

void RegisterClassesWithStreamingSystem(void)
{
  // Make sure that as part of the startup
  // code our streaming classes are registered
  // with the streaming system.

  #pragma startup RegisterClassesWithStreamingSystem

  Classes::RegisterClass(__classid(MyScrollBar));
}

MyScrollBar *ScrollBar1;

AnsiString __fastcall ComponentToString(TComponent *c)
{
 AnsiString as;
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);

 try
 {
    pms->WriteComponent(c);
    pms->Seek(0, soFromBeginning);
    ObjectBinaryToText(pms, pss);
    pss->Seek(0, soFromBeginning);
    as = pss->DataString;
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 delete pms;
 delete pss;
 return as;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Memo1->Text = ComponentToString(ScrollBar1);
}

TComponent *__fastcall StringToComponent(AnsiString as)

{
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);
 TComponent *pc;

 try
 {
    ObjectTextToBinary(pss, pms);
    pms->Seek(0, soFromBeginning);
 }
```

```
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 pc = pms->ReadComponent(NULL);
 delete pms;
 delete pss;
 return pc;
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  TComponent *temp = StringToComponent(Memo1->Text);
  ScrollBar1->Free();

  ScrollBar1 = dynamic_cast<MyScrollBar *>(temp);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ScrollBar1 = new MyScrollBar(Form1);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
  ScrollBar1->Top = 48;
  ScrollBar1->Left = 250;
  ScrollBar1->Name = "Ricksbar";
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete ScrollBar1;
}
```

Delphi Examples:

```
{
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
}
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  MyScrollBar = class(TScrollBar)
  end;

var
  Form1: TForm1;
  ScrollBar1: MyScrollBar;

implementation

{$R *.dfm}
```

```
function ComponentToStringProc(Component: TComponent): string;
var
  BinStream:TMemoryStream;
  StrStream: TStringStream;
  s: string;
begin
  BinStream := TMemoryStream.Create;
  try
    StrStream := TStringStream.Create(s);
    try
      BinStream.WriteComponent(Component);
      BinStream.Seek(0, soFromBeginning);
      ObjectBinaryToText(BinStream, StrStream);
      StrStream.Seek(0, soFromBeginning);
      Result:= StrStream.DataString;
    finally
      StrStream.Free;
    end;
  finally
    BinStream.Free
  end;
end;

function StringToComponentProc(Value: string): TComponent;
var
  StrStream:TStringStream;
  BinStream: TMemoryStream;
begin
  StrStream := TStringStream.Create(Value);
  try
    BinStream := TMemoryStream.Create;
    try
      ObjectTextToBinary(StrStream, BinStream);
      BinStream.Seek(0, soFromBeginning);
      Result:= BinStream.ReadComponent(nil);
    finally
      BinStream.Free;
    end;
  finally
    StrStream.Free;
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Memo1.Text:= ComponentToStringProc(ScrollBar1);
end;

// Edit the scrollbar parameters in the memo, click this
// button and see the scrollbar change.
procedure TForm1.Button2Click(Sender: TObject);
begin
  ScrollBar1.Free;
  ScrollBar1:= (StringToComponentProc(Memo1.Text) as MyScrollBar);
  ScrollBar1.Parent:= Form1;
  ScrollBar1.Visible:= TRUE;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ScrollBar1:= MyScrollBar.Create(Form1);
  ScrollBar1.Parent:= Form1;
  ScrollBar1.Visible:= TRUE;
  ScrollBar1.Top:= 48;
  ScrollBar1.Left:= 250;
```

```
  ScrollBar1.Name:= 'Ricksbar';
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  ScrollBar1.Free;
end;

initialization

RegisterClass(MyScrollBar);
```

See Also

Write (see page 2989)

WriteComponentRes (see page 2994)

WriteRootComponent (see page 3188)

WriteComponentResFile

Using Streams to Read or Write Data

23.1.46.2.14 TStream.WriteComponentRes Method

Writes components and their properties to a stream from a resource file format.

Pascal

```
procedure WriteComponentRes(const ResName: string; Instance: TComponent);
```

C++

```
__fastcall WriteComponentRes(const AnsiString ResName, TComponent * Instance);
```

Description

WriteComponentRes is used internally in the streaming system, but can also be called directly when sending data to other applications on disk. WriteComponentRes is used for streaming components that need data, such as a bitmap or icon to be stored in a resource-file format.

WriteComponentRes calls WriteDescendentRes (see page 2995), passing in nil (Delphi) or NULL (C++) as the Ancestor. Therefore, WriteDescendentRes (see page 2995) initiates the remainder of the streaming process for a component that, in this case, is not a descendant.

To read a component written with WriteComponentRes, call the ReadComponentRes (see page 2987) method.

See Also

Ancestor (see page 2839)

ReadComponentRes (see page 2987)

WriteComponent (see page 2990)

WriteDescendentRes (see page 2995)

23.1.46.2.15 TStream.WriteDescendent Method

Streams components and their properties in inherited forms.

Pascal

```
procedure WriteDescendent(Instance: TComponent; Ancestor: TComponent);
```

C++

```
__fastcall WriteDescendent(TComponent * Instance, TComponent * Ancestor);
```

Description

Do not call WriteDescendent directly. WriteDescendent is called automatically by WriteComponent (see page 2990).

WriteDescendent constructs a writer object, then calls the writer object's WriteDescendent method to write the component passed in Instance to the stream. Instance is either an inherited form descended from Ancestor or nil (Delphi) or NULL (C++). However, WriteDescendent is never used to write owned components, only to initiate streaming on the root component.

See Also

Ancestor (see page 2839)

23.1.46.2.16 TStream.WriteDescendentRes Method

Streams components and their properties in inherited forms.

Pascal

```
procedure WriteDescendentRes(const ResName: string; Instance: TComponent; Ancestor:
TComponent);
```

C++

```
__fastcall WriteDescendentRes(const AnsiString ResName, TComponent * Instance, TComponent *
Ancestor);
```

Description

Do not call WriteDescendentRes directly. WriteDescendentRes is called automatically by WriteComponentRes (see page 2994).

WriteDescendentRes writes a resource-file header to the stream, using the resource name passed in ResName as the name of the resource. It then calls WriteDescendent (see page 2994) to write Instance to the stream as a descendant of Ancestor.

See Also

Ancestor (see page 2839)

23.1.46.2.17 TStream.WriteResourceHeader Method

Writes a resource-file header to the stream.

Pascal

```
procedure WriteResourceHeader(const ResName: string; out FixupInfo: Integer);
```

C++

```
__fastcall WriteResourceHeader(const AnsiString ResName, int FixupInfo);
```

Description

Do not call WriteResourceHeader, it is used internally. The WriteDescendentRes (see page 2995) method calls WriteResourceHeader before writing a component to the stream. This method writes the resource-file header, using the value passed as ResName for the name of the resource. It returns a position in FixupInfo that must be used to adjust the header after the size of the resource is known. WriteDescendentRes (see page 2995) calls FixupResourceHeader (see page 2981) with the value returned as FixupInfo after streaming out the component.

See Also

FixupResourceHeader (see page 2981)

WriteDescendentRes (see page 2995)

23.1.46.3 TStream Properties

The properties of the TStream class are listed here.

Public Properties

	Name	Description
	Position (see page 2996)	Indicates the current offset into the stream for reading and writing.
	Size (see page 2997)	Indicates the size in bytes of the stream.

23.1.46.3.1 TStream.Position Property

Indicates the current offset into the stream for reading and writing.

Pascal

```
property Position: Int64;
```

C++

```
__property Int64 Position;
```

Description

Use Position to obtain the current position of the stream. This is the number of bytes from the beginning of the streamed data.

C++ Examples:

```
/*
This example requires TListBox, TMemo, TRichEdit, and
TButton controls placed on the form. The list box should
contain one or more items.  When the form becomes visible,
click on the button and the contents of the list box will be
transferred to a stream and then to the rich edit control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemoryStream* pms = new TMemoryStream();
  ListBox1->Items->SaveToStream(pms); // write list box contents to the
                                      // stream.
  pms->Position = 0;                  // reset to the beginning of the stream.
  RichEdit1->Lines->LoadFromStream(pms); // load stream contents into rich
                                         // edit control.
  delete pms;
}
```

Delphi Examples:

```
{
This example requires TListBox, TMemo, TRichEdit, and
TButton controls placed on the form. The list box should
contain one or more items.  When the form becomes visible,
click on the button and the contents of the list box will be
transferred to a stream and then to the rich edit control.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  TempStream : TMemoryStream;
begin
  TempStream := TMemoryStream.Create;
  ListBox1.Items.SaveToStream(TempStream);  // write list box contents to the
                                            // stream
  TempStream.Position := 0;      // reset to the beginning of the stream
```

```
    RichEdit1.Lines.LoadFromStream( TempStream); // load stream contents into rich
                                                 // edit control
    TempStream.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Apples');
    ListBox1.Items.Add('Oranges');
    ListBox1.Items.Add('Pears');
  end;
```

See Also

Seek (see page 2988)

Position (see page 2960)

Position (see page 3192)

23.1.46.3.2 TStream.Size Property

Indicates the size in bytes of the stream.

Pascal

```
property Size: Int64;
```

C++

```
__property Int64 Size;
```

Description

Use Size to find the size of the stream. Size is used internally in routines that read and write to and from the stream. Setting the Size property of TStream (see page 2973) does nothing. Some descendants of TStream (see page 2973) override this property to allow applications to change the size of the resource accessed using the stream.

See Also

Seek (see page 2988)

23.1.47 TStreamAdapter Class

TStreamAdapter implements the IStream interface on a TStream (see page 2973) object.

Class Hierarchy

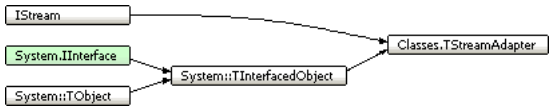

Pascal

```
TStreamAdapter = class(TInterfacedObject, IStream);
```

C++

```
class TStreamAdapter : public TInterfacedObject, public IStream;
```

File

Classes

Description

DescriptionTStreamAdapter is an interfaced object that implements the IStream interface. It is associated with a TStream (see page 2973) object in the constructor, which it uses to implement the IStream methods.

Note: On Windows, TStreamAdapter allows a TStream (see page 2973) object to be used as a COM-based IStream interface.

See Also

TCustomAdapter

TFontAdapter (see page 1583)

TPictureAdapter (see page 1599)

TOleStream (see page 1595)

IStream

23.1.47.1 TStreamAdapter Members

The following tables list the members exposed by TStreamAdapter.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clone (see page 3000)	Implements the IStream Clone method.
V	Commit (see page 3001)	Implements the IStream Commit method.
V	CopyTo (see page 3001)	Implements the IStream CopyTo method.
	Create (see page 2999)	Creates an instance of a TStreamAdapter (see page 2997) object.

	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 3000)	Frees the instance of TStreamAdapter (see page 2997).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LockRegion (see page 3002)	Implements the IStream LockRegion method.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Read (see page 3002)	Implements the IStream Read method.
	Revert (see page 3002)	Implements the IStream Revert method.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Seek (see page 3003)	Implements the IStream Seek method.
	SetSize (see page 3003)	Implements the IStream SetSize method.
	Stat (see page 3004)	Implements the IStream Stat method.
	UnlockRegion (see page 3004)	Implements the IStream UnlockRegion method.
	Write (see page 3004)	Implements the IStream Write method.

Public Properties

	Name	Description
	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Stream (see page 3005)	Identifies the associated TStream (see page 2973) object that TStreamAdapter (see page 2997) uses to implement the IStream interface.
	StreamOwnership (see page 3005)	Specifies whether the TStreamAdapter (see page 2997) should act as the Owner of the associated stream object.

23.1.47.2 TStreamAdapter.Create Constructor

Creates an instance of a TStreamAdapter (see page 2997) object.

Pascal

```
constructor Create(Stream: TStream; Ownership: TStreamOwnership = soReference);
```

C++

```
__fastcall TStreamAdapter(TStream Stream, TStreamOwnership Ownership = soReference);
```

Description

Call Create to instantiate a TStreamAdapter (see page 2997) object that implements the IStream interface using the stream object specified by Stream (see page 3005). Ownership initializes the StreamOwnership (see page 3005) property, indicating whether the TStreamAdapter (see page 2997) object is responsible for freeing the stream specified by Stream (see page 3005) from its own destructor.

See Also

Stream (see page 3005)

StreamOwnership (see page 3005)

23.1.47.3 TStreamAdapter.Destroy Destructor

Frees the instance of TStreamAdapter (see page 2997).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TStreamAdapter();
```

Description

Most applications do not need to explicitly free the TStreamAdapter (see page 2997) instance. Instead, it is freed automatically when its reference count drops to 0.

If you must explicitly destroy a TStreamAdapter (see page 2997) instance, use the Free method instead of Destoy. Free checks that the TStreamAdapter (see page 2997) reference is not nil before calling Destroy.

Destroy frees the associated stream object (Stream (see page 3005) property) if its StreamOwnership (see page 3005) property is True.

See Also

Stream (see page 3005)

StreamOwnership (see page 3005)

Free (see page 22300)

23.1.47.4 TStreamAdapter Methods

The methods of the TStreamAdapter class are listed here.

Public Methods

	Name	Description
	Clone (see page 3000)	Implements the IStream Clone method.
	Commit (see page 3001)	Implements the IStream Commit method.
	CopyTo (see page 3001)	Implements the IStream CopyTo method.
	LockRegion (see page 3002)	Implements the IStream LockRegion method.
	Read (see page 3002)	Implements the IStream Read method.
	Revert (see page 3002)	Implements the IStream Revert method.
	Seek (see page 3003)	Implements the IStream Seek method.
	SetSize (see page 3003)	Implements the IStream SetSize method.
	Stat (see page 3004)	Implements the IStream Stat method.
	UnlockRegion (see page 3004)	Implements the IStream UnlockRegion method.
	Write (see page 3004)	Implements the IStream Write method.

23.1.47.4.1 TStreamAdapter.Clone Method

Implements the IStream Clone method.

Pascal

```
function Clone(out stm: IStream): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT Clone(IStream stm);
```

Description

The Clone method defined by IStream returns a second IStream interface that accesses the same bytes as this IStream interface, but which operates independently (can have its own position).

TStreamAdapter (see page 2997) returns E_NOTIMPL to indicate that it does not support the Clone method.

23.1.47.4.2 TStreamAdapter.Commit Method

Implements the IStream Commit method.

Pascal

```
function Commit(grfCommitFlags: Longint): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT Commit(Longint grfCommitFlags);
```

Description

The Commit method defined by IStream commits any changes written to a stream that is participating in a transaction.

TStreamAdapter (see page 2997) does not implement any transaction support, but operates in direct mode. Because of this, Commit returns S_OK to indicate that all changes are already written, but does nothing else.

See Also

Revert (see page 3002)

23.1.47.4.3 TStreamAdapter.CopyTo Method

Implements the IStream CopyTo method.

Pascal

```
function CopyTo(stm: IStream; cb: Largeint; out cbRead: Largeint; out cbWritten: Largeint):
HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT CopyTo(IStream stm, Largeint cb, Largeint cbRead,
Largeint cbWritten);
```

Description

CopyTo copies up to cb bytes from the current position in the stream to the stream specified by stm. cbRead returns the number of bytes successfully read from the associated stream. cbWritten returns the number of bytes successfully written to stm.

CopyTo returns S_OK if it successfully copied all the bytes read from Stream (see page 3005) to the interface specified by stm. It returns E_FAIL if it could not write all the bytes read from Stream (see page 3005) to stm. It returns E_UNEXPECTED if it encountered an exception while attempting to perform the copy operation.

See Also

Stream (see page 3005)

CopyFrom (see page 2979)

Read (see page 3002)

23.1.47.4.4 TStreamAdapter.LockRegion Method

Implements the IStream LockRegion method.

Pascal

```
function LockRegion(libOffset: Largeint; cb: Largeint; dwLockType: Longint): HResult; virtual;
stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT LockRegion(Largeint libOffset, Largeint cb, Longint
dwLockType);
```

Description

The LockRegion method defined by IStream restricts access in the associated stream from the position specified by libOffset to the position specified by cb. dwLockType indicates the type of restriction imposed.

TStreamAdapter (see page 2997) does not support locking of regions, so LockRegion returns STG_E_INVALIDFUNCTION.

See Also

UnlockRegion (see page 3004)

23.1.47.4.5 TStreamAdapter.Read Method

Implements the IStream Read method.

Pascal

```
function Read(pv: Pointer; cb: Longint; pcbRead: PLongint): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT Read(void * pv, Longint cb, PLongint pcbRead);
```

Description

Read reads up to cb bytes from the current position in the stream into the buffer pointed to by pv. pcbRead returns the number of bytes actually read.

Read returns S_OK if it successfully reads pcbRead bytes into the buffer. It returns STG_E_INVALIDPOINTER it can't read because pv is nil (Delphi) or NULL (C++). It returns S_FALSE if the read operation fails for any other reason.

See Also

Write (see page 3004)

Read (see page 2982)

CopyTo (see page 3001)

23.1.47.4.6 TStreamAdapter.Revert Method

Implements the IStream Revert method.

Pascal

```
function Revert: HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT Revert();
```

Description

The Revert method defined by IStream rolls back any changes written to a stream that is participating in a transaction.

TStreamAdapter (see page 2997) does not implement any transaction support, but operates in direct mode. Because of this, Revert returns STG_E_REVERTED, but does nothing else.

See Also

Commit (see page 3001)

23.1.47.4.7 TStreamAdapter.Seek Method

Implements the IStream Seek method.

Pascal

```
function Seek(dlibMove: Largeint; dwOrigin: Longint; out libNewPosition: Largeint): HResult;
virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT Seek(Largeint dlibMove, Longint dwOrigin, Largeint
libNewPosition);
```

Description

Seek moves the current position of the stream to the position specified by dlibMove and dwOrigin. The dlibMove parameter indicates a number of bytes, and dwOrigin indicates from where the bytes are measured to locate the new position. The libNewPosition parameter returns the new position of the stream.

Seek returns S_OK if it successfully Seeks to the specified position. It returns STG_E_INVALIDFUNCTION if dlibMove is STREAM_SEEK_SET or STREAM_SEEK_END, because these types of seek operations are not supported by the TStream.Seek method. It returns STG_E_INVALIDPOINTER if the seek operation fails for any other reason.

See Also

Seek (see page 2988)

Stream (see page 3005)

23.1.47.4.8 TStreamAdapter.SetSize Method

Implements the IStream SetSize method.

Pascal

```
function SetSize(libNewSize: Largeint): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT SetSize(Largeint libNewSize);
```

Description

SetSize sets the size of the stream to the number of bytes specified by libNewSize.

SetSize returns S_OK if it successfully resizes the stream to libNewSize. It returns E_FAIL if it could not resize the stream and the current size of the stream is not libNewSize on exit. It returns E_UNEXPECTED if SetSize encounters an exception while trying to set the size of the stream.

See Also

Size (see page 2997)

Stream (see page 3005)

23.1.47.4.9 TStreamAdapter.Stat Method

Implements the IStream Stat method.

Pascal

```
function Stat(out statstg: TStatStg; grfStatFlag: Longint): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT Stat(TStatStg statstg, Longint grfStatFlag);
```

Description

Stat fills out a TStatStg record with information about the stream object The grfStatFlag parameter indicates what information in the record, if any, is not needed.

Stat ignores the grfStatFlag parameter. It always sets dwType to STGTY_STREAM, grfLocksSupported to LOCK_WRITE, and all of the TFileTime parameters to 0. It sets cbSize to the size of the stream, and leaves the remaining fields unchanged.

Stat returns S_OK if it does not encounter any exceptions filling out statstg, E_UNEXPECTED otherwise.

See Also

Size (see page 2997)

Stream (see page 3005)

23.1.47.4.10 TStreamAdapter.UnlockRegion Method

Implements the IStream UnlockRegion method.

Pascal

```
function UnlockRegion(libOffset: Largeint; cb: Largeint; dwLockType: Longint): HResult;
virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT UnlockRegion(Largeint libOffset, Largeint cb, Longint
dwLockType);
```

Description

The UnlockRegion method defined by IStream re-enables access in the associated stream that was previously restricted by a call to LockRegion (see page 3002). The region runs from the position specified by libOffset to the position specified by cb. dwLockType indicates the type of restriction to be released.

TStreamAdapter (see page 2997) does not support locking of regions, so UnlockRegion returns STG_E_INVALIDFUNCTION.

See Also

LockRegion (see page 3002)

23.1.47.4.11 TStreamAdapter.Write Method

Implements the IStream Write method.

Pascal

```
function Write(pv: Pointer; cb: Longint; pcbWritten: PLongint): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT Write(void * pv, Longint cb, PLongint pcbWritten);
```

Description

Write writes up to cb bytes from the buffer pointed to by pv into the stream, starting at the current position. pcbWritten returns the number of bytes actually written.

Write returns S_OK if it successfully writes pcbWritten bytes into the stream. It returns STG_E_INVALIDPOINTER it can't write because pv is nil. It returns STG_E_CANTSAVE if the write operation fails for any other reason.

See Also

Read (see page 3002)

Write (see page 2989)

23.1.47.5 TStreamAdapter Properties

The properties of the TStreamAdapter class are listed here.

Public Properties

	Name	Description
R	Stream (see page 3005)	Identifies the associated TStream (see page 2973) object that TStreamAdapter (see page 2997) uses to implement the IStream interface.
	StreamOwnership (see page 3005)	Specifies whether the TStreamAdapter (see page 2997) should act as the Owner of the associated stream object.

23.1.47.5.1 TStreamAdapter.Stream Property

Identifies the associated TStream (see page 2973) object that TStreamAdapter (see page 2997) uses to implement the IStream interface.

Pascal

```
property Stream: TStream;
```

C++

```
__property TStream Stream;
```

Description

Read (see page 3002) Stream to directly access the TStream (see page 2973) object that the stream adapter represents. This object is associated with the TStream (see page 2973) object in the TStreamAdapter (see page 2997) constructor.

See Also

Create (see page 2999)

StreamOwnership (see page 3005)

23.1.47.5.2 TStreamAdapter.StreamOwnership Property

Specifies whether the TStreamAdapter (see page 2997) should act as the Owner of the associated stream object.

Pascal

```
property StreamOwnership: TStreamOwnership;
```

C++

```
__property TStreamOwnership StreamOwnership;
```

Description

StreamOwnership indicates whether the TStreamAdapter (see page 2997) is responsible for freeing the associated stream object.

See Also

Create (see page 2999)

Stream (see page 3005)

Destroy (see page 3000)

23.1.48 TStringList Class

TStringList maintains a list of strings.

Class Hierarchy

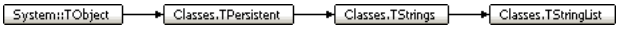

Pascal

```
TStringList = class(TStrings);
```

C++

```
class TStringList : public TStrings;
```

File

Classes

Description

Use a string list object to store and manipulate a list of strings. TStringList implements the abstract properties and methods introduced by TStrings (see page 3029), and introduces new properties, events, and methods to

Sort (see page 3023) the strings in the list.

Prohibit duplicate strings in sorted lists.

Respond to changes in the contents of the list.

Control whether strings are located, sorted, and identified as duplicates in a case-sensitive or case-insensitive manner.

See Also

TCollection (see page 2731)

TList (see page 2868)

Adding Graphical Objects to a String List

Adding Images to a String List

List Controls

Working with Lists

23.1.48.1 TStringList Members

The following tables list the members exposed by TStringList.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 3013)	This is Changed, a member of class TStringList.
	Changing (see page 3013)	This is Changing, a member of class TStringList.
	CompareStrings (see page 3014)	Compares two strings.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 3049)	Reads and writes the Strings (see page 3080) property as if it were published.
	Error (see page 3052)	This is the overview for the Error method overload.
	ExtractName (see page 3053)	Returns the name portion of a string that is a name value pair.
	Get (see page 3017)	Returns a string given its index.
	GetCapacity (see page 3017)	Returns the currently allocated size of the list of strings.
	GetCount (see page 3018)	Returns the number of strings in the list
	GetObject (see page 3018)	Returns the object associated with the string at a specified index.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTextStr (see page 3056)	Returns the value of the Text (see page 3084) property.
	InsertItem (see page 3020)	This is InsertItem, a member of class TStringList.
	Put (see page 3022)	Changes the value of the string with a specified index.
	PutObject (see page 3022)	Changes the object associated with the string at a specified index.
	SetCapacity (see page 3022)	Changes the amount of memory allocated to hold strings in the list.
	SetTextStr (see page 3069)	Sets the Text (see page 3084) property.
	SetUpdateState (see page 3023)	Performs internal adjustments before or after a series of updates.

Protected Properties

	Name	Description
	UpdateCount (see page 3085)	Indicates the number of calls to BeginUpdate (see page 3045) that have not been matched by a call to EndUpdate (see page 3050).

Public Events

	Name	Description
	OnChange (see page 3028)	Occurs immediately after the list of strings changes.
	OnChanging (see page 3028)	Occurs immediately before the list of strings changes.

Public Methods

	Name	Description
	Add (see page 3010)	Adds a new string to the list.
	AddObject (see page 3012)	Adds a string to the list, and associates an object with the string.
	AddStrings (see page 3039)	Adds a group of strings to the list.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 3041)	Adds a string to the list.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 3043)	Sets the strings in the list, and possibly associated objects, from a source object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 3045)	Enables the TStrings (see page 3029) object to track when the list of strings is changing.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.

	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 3013)	Deletes all the strings from the list.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	CustomSort (see page 3014)	Sorts the strings in the list in a customized order.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Delete (see page 3015)	Removes the string specified by the Index parameter.
V	Destroy (see page 3009)	Destroys an instance of TStringList (see page 3006).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 3050)	Enables the TStrings (see page 3029) object to keep track of when the list of strings has finished changing.
	Equals (see page 3051)	Compares the list of strings to the list from another TStrings (see page 3029) object and returns true if the two lists match.
V	Exchange (see page 3015)	Swaps the position of two strings in the list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
V	Find (see page 3016)	Locates the index for a string in a sorted list and indicates whether a string with that value already exists in the list.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 3054)	Returns a TStrings (see page 3029) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetText (see page 3055)	Allocates a text buffer and fills it with the value of the Text (see page 3084) property.
V	IndexOf (see page 3019)	Returns the position of a string in the list.
V	IndexOfName (see page 3057)	Returns the position of the first name-value pair with the specified name.
V	IndexOfObject (see page 3059)	Returns the index of the first string in the list associated with a given object.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Insert (see page 3019)	Inserts a string to the list at the position specified by Index.
V	InsertObject (see page 3020)	Inserts a string into the list at the specified position, and associates it with an object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	LoadFromFile (see page 3061)	Fills the list with the lines of text in a specified file.
V	LoadFromStream (see page 3062)	Fills the list with lines of text read from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	Move (see page 3063)	Changes the position of a string in the list.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SaveToFile (see page 3065)	Saves the strings in the list to the specified file.
V	SaveToStream (see page 3067)	Writes the value of the Text (see page 3084) property to a stream object.
V	SetText (see page 3068)	Sets the Text (see page 3084) property.
V	Sort (see page 3023)	Sorts the strings in the list in ascending order.

Public Properties

	Name	Description
	Capacity (see page 3070)	Indicates the number of strings the TStrings (see page 3029) object can hold.

	CaseSensitive (see page 3025)	Controls (see page 5711) whether strings are located, sorted, and identified as duplicates in a case-sensitive or case-insensitive manner.
	CommaText (see page 3070)	Lists the strings in the TStrings (see page 3029) object in system data format (SDF).
R	Count (see page 3071)	Introduces an abstract property to represent the number of strings in the list.
	DelimitedText (see page 3074)	Represents all the strings in the TStrings (see page 3029) object as a single delimited string.
	Delimiter (see page 3074)	Specifies the delimiter used by the DelimitedText (see page 3074) property.
	Duplicates (see page 3025)	Specifies whether duplicate strings can be added to sorted lists.
	LineBreak (see page 3075)	Defines the line break character(s).
R	Names (see page 3075)	Indicates the name part of strings that are name-value pairs.
	NameValueSeparator (see page 3077)	Indicates the character used to separate names from values.
	Objects (see page 3077)	Represents a set of objects that are associated one with each of the strings in the Strings (see page 3080) property.
	QuoteChar (see page 3080)	Specifies the quote character used by the DelimitedText (see page 3074) property.
	Sorted (see page 3026)	Specifies whether the strings in the list should be automatically sorted.
	StrictDelimiter (see page 3080)	Determines how the Delimiter (see page 3074) property is used.
	Strings (see page 3080)	References the strings in the list by their positions.
	StringsAdapter (see page 3084)	Implements an IStringsAdapter (see page 3208) interface for the TStrings (see page 3029) object.
	Text (see page 3084)	Lists the strings in the TStrings (see page 3029) object as a single string with the individual strings delimited by carriage returns and line feeds.
	ValueFromIndex (see page 3085)	Represents the value part of a string with a given index, on strings that are name-value pairs.
	Values (see page 3086)	Represents the value part of a string associated with a given name, on strings that are name-value pairs.

23.1.48.2 TStringList.Destroy Destructor

Destroys an instance of TStringList (see page 3006).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TStringList();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TStringList (see page 3006) reference is not nil and only then calls Destroy.

Destroy frees the memory allocated to hold the list of strings and object references before calling the inherited destructor.

See Also

Free (see page 22300)

23.1.48.3 TStringList Methods

The methods of the TStringList class are listed here.

Protected Methods

	Name	Description
V	Changed (see page 3013)	This is Changed, a member of class TStringList.
V	Changing (see page 3013)	This is Changing, a member of class TStringList.

	CompareStrings (see page 3014)	Compares two strings.
	Get (see page 3017)	Returns a string given its index.
	GetCapacity (see page 3017)	Returns the currently allocated size of the list of strings.
	GetCount (see page 3018)	Returns the number of strings in the list
	GetObject (see page 3018)	Returns the object associated with the string at a specified index.
	InsertItem (see page 3020)	This is InsertItem, a member of class TStringList.
	Put (see page 3022)	Changes the value of the string with a specified index.
	PutObject (see page 3022)	Changes the object associated with the string at a specified index.
	SetCapacity (see page 3022)	Changes the amount of memory allocated to hold strings in the list.
	SetUpdateState (see page 3023)	Performs internal adjustments before or after a series of updates.

Public Methods

	Name	Description
	Add (see page 3010)	Adds a new string to the list.
	AddObject (see page 3012)	Adds a string to the list, and associates an object with the string.
	Clear (see page 3013)	Deletes all the strings from the list.
	CustomSort (see page 3014)	Sorts the strings in the list in a customized order.
	Delete (see page 3015)	Removes the string specified by the Index parameter.
	Exchange (see page 3015)	Swaps the position of two strings in the list.
	Find (see page 3016)	Locates the index for a string in a sorted list and indicates whether a string with that value already exists in the list.
	IndexOf (see page 3019)	Returns the position of a string in the list.
	Insert (see page 3019)	Inserts a string to the list at the position specified by Index.
	InsertObject (see page 3020)	Inserts a string into the list at the specified position, and associates it with an object.
	Sort (see page 3023)	Sorts the strings in the list in ascending order.

23.1.48.3.1 TStringList.Add Method

Adds a new string to the list.

Pascal

```
function Add(const S: string): Integer; override;
```

C++

```
virtual __fastcall int Add(const AnsiString S);
```

Description

Call Add to add the string S to the list. If the list is sorted, S is added to the appropriate position in the sort order. If the list is not sorted, S is added to the end of the list. Add returns the position of the item in the list, where the first item in the list has a value of 0.

Note: For sorted lists, Add will raise an EListError (see page 2701) exception if the string S already appears in the list and Duplicates (see page 3025) is set to dupError. If Duplicates (see page 3025) is set to dupIgnore, trying to add a duplicate string causes Add to return the index of the existing entry.

C++ Examples:

```
/*
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
```

```
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  int Index;
  TStringList* MyList = new TStringList();
  try
  {
    MyList->Add("Animals");
    MyList->Add("Flowers");
    MyList->Add("Ocean");
    MyList->Sort(); // Find will only work on sorted lists
    if (MyList->Find("Flowers", Index))
    {
      ListBox1->Items->AddStrings(MyList);
      Label1->Caption = "Flowers has an index value of " + AnsiString(Index);
    }
  }
  __finally
  {
    delete MyList;
  }
}
/*
This example creates a stringlist, adds elements to it and
assigns the strings to items of a combo box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TStrings *StringList = new TStringList();
  try
  {
    StringList->Add("This example uses A string List. ");
    StringList->Add("It is the easiest way to add strings");
    StringList->Add("to a combobox's list of strings. ");
    StringList->Add("Always remember TStrings is abstract,");
    StringList->Add("So create a TStringList instead.");
    ComboBox1->Width = 210;
    ComboBox1->Items->Assign(StringList);
    ComboBox1->ItemIndex = 0;
  }
  __finally
  {
    delete StringList;
  }
}
```

Delphi Examples:

```
{
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  MyList: TStringList;
  Index: Integer;
begin
  MyList := TStringList.Create;
  try
    MyList.Add('Animals');
    MyList.Add('Flowers');
    MyList.Add('Cars');
```

```
      MyList.Sort;   { Find will only work on sorted lists! }
      if MyList.Find('Flowers', Index) then
      begin
        ListBox1.Items.AddStrings(MyList);
        Label1.Caption := 'Flowers has an index value of ' + IntToStr(Index);
      end;
    finally
      MyList.Free;
    end;
  end;
  {
  This example creates a stringlist, adds elements to it and
  assigns the strings to items of a combo box.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    StringList: TStringList;
  begin
    StringList := TStringList.Create;
    try
      with StringList do begin
        Add('This example uses A string List.');
        Add('It is the easiest way to add strings');
        Add('to a combobox''s list of strings.');
        Add('Always remember TStrings.Create method');
        Add('is abstract; So use TStringList.Create');
        Add('method instead.');
      end;
      with ComboBox1 do begin
        Width := 210;
        Items.Assign(StringList);
        ItemIndex := 0;
      end;
    finally
      StringList.free;
    end;
  end;
```

See Also

AddStrings (see page 3039)

Append (see page 3041)

Delete (see page 3015)

Duplicates (see page 3025)

Insert (see page 3019)

Sorted (see page 3026)

23.1.48.3.2 TStringList.AddObject Method

Adds a string to the list, and associates an object with the string.

Pascal

```
function AddObject(const S: string; AObject: TObject): Integer; override;
```

C++

```
virtual __fastcall int AddObject(const AnsiString S, TObject * AObject);
```

Description

Call AddObject to add a string and its associated object to the list. AddObject returns the index of the new string and object.

Note: The TStringList (see page 3006) object does not own the objects you add this way. Objects (see page 3077) added to the TStringList (see page 3006) object still exist even if the TStringList (see page 3006) instance is destroyed. They must be explicitly destroyed by the application.

Note: For sorted lists, AddObject raises an EListError (see page 2701) exception if the string S already appears in the list and Duplicates (see page 3025) is set to dupError. If Duplicates (see page 3025) is set to dupIgnore, trying to add a duplicate string causes AddObject to return the index of the existing entry.

See Also

Add (see page 3010)

AddStrings (see page 3039)

23.1.48.3.3 TStringList.Changed Method

Pascal

```
procedure Changed; virtual;
```

C++

```
virtual __fastcall Changed();
```

Description

This is Changed, a member of class TStringList.

23.1.48.3.4 TStringList.Changing Method

Pascal

```
procedure Changing; virtual;
```

C++

```
virtual __fastcall Changing();
```

Description

This is Changing, a member of class TStringList.

23.1.48.3.5 TStringList.Clear Method

Deletes all the strings from the list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call clear to empty the list of strings. All references to associated objects are also removed. However, the objects themselves are not freed.

See Also

Delete (see page 3015)

Objects

Strings

23.1.48.3.6 TStringList.CompareStrings Method

Compares two strings.

Pascal

```
function CompareStrings(const S1: string; const S2: string): Integer; override;
```

C++

```
virtual __fastcall int CompareStrings(const AnsiString S1, const AnsiString S2);
```

Description

TStrings uses CompareStrings internally to compare the values of strings that appear in the list. For example, the IndexOf and IndexOfName methods use CompareStrings to compare a specified string with the strings in the list.

S1 and S2 are the strings to compare.

CompareStrings returns a value less than 0 if S1 < S2, 0 if S1 == S2, and a value greater than 0 if S1 > S2.

As implemented in TStrings, CompareStrings uses the global AnsiCompareText function, which compares strings case insensitively. Some descendant classes override this method to change the way strings are compared (for example, to introduce case sensitivity).

See Also

AnsiCompareText (see page 22790)

IndexOf (see page 3056)

IndexOfName (see page 3057)

23.1.48.3.7 TStringList.CustomSort Method

Sorts the strings in the list in a customized order.

Pascal

```
procedure CustomSort(Compare: TStringListSortCompare); virtual;
```

C++

```
virtual __fastcall CustomSort(TStringListSortCompare Compare);
```

Description

Use CustomSort to sort the strings in the list, where the sort order is defined by the Compare parameter.

Supply a value for the Compare function that compares two strings in the string list. The List parameter provides access to the string list, while the Index1 and Index2 parameters identify the strings to be compared.

Do not pass nil (Delphi) or NULL (C++) as the value of the Compare parameter.

Note: You must explicitly call the CustomSort method. Setting the Sorted (see page 3026) property only sorts strings using ANSI (Windows) or UTF-8 (Linux) order, as implemented in the Sort (see page 3023) method.

See Also

Sort (see page 3023)

Strings

Sorted (see page 3026)

23.1.48.3.8 TStringList.Delete Method

Removes the string specified by the Index parameter.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Call Delete to remove a single string from the list. If an object is associated with the string, the reference to the object is removed as well. Index gives the position of the string, where 0 is the first string, 1 is the second string, and so on.

See Also

Add (see page 3010)

AddObject (see page 3012)

Clear (see page 3013)

IndexOf (see page 3019)

InsertObject (see page 3060)

Objects

Strings

23.1.48.3.9 TStringList.Exchange Method

Swaps the position of two strings in the list.

Pascal

```
procedure Exchange(Index1: Integer; Index2: Integer); override;
```

C++

```
virtual __fastcall Exchange(int Index1, int Index2);
```

Description

Call Exchange to rearrange the strings in the list. The strings are specified by their index values in the Index1 and Index2 parameters. Indexes are zero-based, so the first string in the list has an index value of 0, the second has an index value of 1, and so on.

If either string has an associated object, Exchange changes the index of the object as well.

Warning: Do not call Exchange on a sorted list except to swap two identical strings with different associated objects. Exchange does not check whether the list is sorted, and can destroy the sort order of a sorted list.

See Also

Move (see page 3063)

Objects

Sorted (see page 3026)

Strings

23.1.48.3.10 TStringList.Find Method

Locates the index for a string in a sorted list and indicates whether a string with that value already exists in the list.

Pascal

```
function Find(const S: string; var Index: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean Find(const AnsiString S, int Index);
```

Description

Use Find to obtain the index in a sorted list where the string S should be added. If the string S, or a string that differs from S only in case when CaseSensitive (see page 3025) is false, already exists in the list, Find returns true. If the list does not contain a string that matches S, Find returns false. The index where S should go is returned in the Index parameter. The value of Index is zero-based, where the first string has the index 0, the second string has the index 1, and so on.

Note: Only use Find with sorted lists. For unsorted lists, use the IndexOf (see page 3019) method instead.

C++ Examples:

```
/*
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  int Index;
  TStringList* MyList = new TStringList();
  try
  {
    MyList->Add("Animals");
    MyList->Add("Flowers");
    MyList->Add("Ocean");
    MyList->Sort(); // Find will only work on sorted lists
    if (MyList->Find("Flowers", Index))
    {
      ListBox1->Items->AddStrings(MyList);
      Label1->Caption = "Flowers has an index value of " + AnsiString(Index);
    }
  }
  __finally
  {
    delete MyList;
  }
}
```

Delphi Examples:

```
{
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
```

```
  added to the list box, and the index value of the 'Flowers'
  string appears in the caption of the label control.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    MyList: TStringList;
    Index: Integer;
  begin
    MyList := TStringList.Create;
    try
      MyList.Add('Animals');
      MyList.Add('Flowers');
      MyList.Add('Cars');
      MyList.Sort;   { Find will only work on sorted lists! }
      if MyList.Find('Flowers', Index) then
      begin
        ListBox1.Items.AddStrings(MyList);
        Label1.Caption := 'Flowers has an index value of ' + IntToStr(Index);
      end;
    finally
      MyList.Free;
    end;
  end;
```

See Also

Duplicates (see page 3025)

IndexOf (see page 3019)

Sorted (see page 3026)

Strings

23.1.48.3.11 TStringList.Get Method

Returns a string given its index.

Pascal

```
function Get(Index: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString Get(int Index);
```

Description

Get is the protected read implementation of the Strings property.

In TStrings Get is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to return the string with the specified index.

See Also

Strings (see page 3080)

23.1.48.3.12 TStringList.GetCapacity Method

Returns the currently allocated size of the list of strings.

Pascal

```
function GetCapacity: Integer; override;
```

C++

```
virtual __fastcall int GetCapacity();
```

Description

GetCapacity is the protected read implementation of the Capacity property. In TStrings, GetCapacity returns the value of the Count property. Descendants of TStrings can override this property to let a string list allocate memory for entries that have not been added to the list.

See Also

Capacity (see page 3070)

Count (see page 3071)

23.1.48.3.13 TStringList.GetCount Method

Returns the number of strings in the list

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCount is the protected read implementation of the Count property.

In TStrings GetCount is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to return the number of strings that have been added to the list.

See Also

Count (see page 3071)

23.1.48.3.14 TStringList.GetObject Method

Returns the object associated with the string at a specified index.

Pascal

```
function GetObject(Index: Integer): TObject; override;
```

C++

```
virtual __fastcall TObject * GetObject(int Index);
```

Description

GetObject is the protected read implementation of the Objects property.

Index is the index of the string with which the object is associated.

In TStrings, GetObject always returns nil (Delphi) or NULL (C++). This provides a default implementation for descendants that do not support associating objects with the strings in the list. Descendants that support this feature override GetObject to return the specified object.

See Also

Objects (see page 3077)

23.1.48.3.15 TStringList.IndexOf Method

Returns the position of a string in the list.

Pascal

```
function IndexOf(const S: string): Integer; override;
```

C++

```
virtual __fastcall int IndexOf(const AnsiString S);
```

Description

Call IndexOf to obtain the position of the first occurrence of a string that matches S. A string matches S if it is identical to S or, if CaseSensitive (see page 3025) is false, if it differs only in case.

IndexOf returns the 0-based index of the string. Thus, if S matches the first string in the list, IndexOf returns 0, if S is the second string, IndexOf returns 1, and so on. If the string does not have a match in the string list, IndexOf returns -1.

See Also

Find (see page 3016)

IndexOfName (see page 3057)

IndexOfObject (see page 3059)

Strings

23.1.48.3.16 TStringList.Insert Method

Inserts a string to the list at the position specified by Index.

Pascal

```
procedure Insert(Index: Integer; const S: string); override;
```

C++

```
virtual __fastcall Insert(int Index, const AnsiString S);
```

Description

Call Insert to add the string S to the list at the position specified by Index. If Index is 0, the string is inserted at the beginning of the list. If Index is 1, the string is put in the second position of the list, and so on.

If the string has an associated object, use the InsertObject (see page 3020) method instead.

Note: If the list is sorted, calling Insert or InsertObject (see page 3020) will raise an EListError (see page 2701) exception. Use Add (see page 3010) or AddObject (see page 3012) with sorted lists.

See Also

Add (see page 3010)

AddObject (see page 3012)

EListError (see page 2701)

InsertObject (see page 3020)

Sorted (see page 3026)

Strings

23.1.48.3.17 TStringList.InsertItem Method

Pascal

```
procedure InsertItem(Index: Integer; const S: string; AObject: TObject); virtual;
```

C++

```
virtual __fastcall InsertItem(int Index, const AnsiString S, TObject * AObject);
```

Description

This is InsertItem, a member of class TStringList.

23.1.48.3.18 TStringList.InsertObject Method

Inserts a string into the list at the specified position, and associates it with an object.

Pascal

```
procedure InsertObject(Index: Integer; const S: string; AObject: TObject); override;
```

C++

```
virtual __fastcall InsertObject(int Index, const AnsiString S, TObject * AObject);
```

Description

Call InsertObject to insert the string S into the list at the position identified by Index, and associate it with the object AObject. If Index is 0, the string is inserted at the beginning of the list. If Index is 1, the string is put in the second position of the list, and so on.

C++ Examples:

```
/*
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < ComponentCount; i++)
    ListBox1->Items->InsertObject(0,
                                  Components[i]->Name,
                                  (TObject *)Components[i]);
}

void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  if (Button == mbRight)
  {
    TClass ClassRef;
    int Index = ListBox1->ItemAtPos(Point(X,Y), true);
    // only components that are controls have a position
    // make sure the component is a control
```

```
    for (ClassRef = ListBox1->Items->Objects[Index]->ClassType();
         ClassRef != NULL;
         ClassRef = ClassRef->ClassParent())
      if (String(ClassRef->ClassName()) == "TControl")
      {
        TControl *TheObject = (TControl *)ListBox1->Items->Objects[Index];
        StatusBar1->SimpleText =
          TheObject->Name + " is at (" +
          IntToStr(TheObject->Left) + ", " +
          IntToStr(TheObject->Top) + ")";
         break;
      }
    if (ClassRef == NULL) // if it wasn't a control
      MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to Form1.ComponentCount-1 do
    ListBox1.Items.InsertObject(
      0, Form1.Components[I].Name,
      Form1.Components[I] as TObject);
end;

procedure TForm1.ListBox1MouseUp(
  Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X,Y: integer);
var
  APoint: TPoint;
  Index: integer;
  TheObject: TControl;
begin
  if Button = mbRight then
  begin
  APoint.X := X;
  APoint.Y := Y;
  Index := ListBox1.ItemAtPos(APoint, True);
  if (ListBox1.Items.Objects[Index] is TControl) then
  begin
    TheObject :=
      (ListBox1.Items.Objects[Index] as TControl);
    StatusBar1.SimpleText := TheObject.Name + ' is at (' +
                             IntToStr(TheObject.Left) + ', ' +
                             IntToStr(TheObject.Top) + ') ';
  end
  else
    Beep;
```

```
  end;
end;
```

See Also

Insert (see page 3019)

Objects (see page 3077)

23.1.48.3.19 TStringList.Put Method

Changes the value of the string with a specified index.

Pascal

```
procedure Put(Index: Integer; const S: string); override;
```

C++

```
virtual __fastcall Put(int Index, const AnsiString S);
```

Description

Put is the protected write implementation of the Strings property.

Put changes the value of the string with the index specified by Index to S. Put does not change the object at the specified position. That is, any object associated with the previous string becomes associated with the new string.

See Also

Strings (see page 3080)

Objects (see page 3077)

23.1.48.3.20 TStringList.PutObject Method

Changes the object associated with the string at a specified index.

Pascal

```
procedure PutObject(Index: Integer; AObject: TObject); override;
```

C++

```
virtual __fastcall PutObject(int Index, TObject * AObject);
```

Description

PutObject is the protected write implementation of the Objects property.

As implemented in TStrings, PutObject does nothing. This provides a default implementation for descendants that do not support associating objects with the strings in the list. Descendants that support this feature override PutObject to change the specified object.

See Also

Objects (see page 3077)

23.1.48.3.21 TStringList.SetCapacity Method

Changes the amount of memory allocated to hold strings in the list.

Pascal

```
procedure SetCapacity(NewCapacity: Integer); override;
```

C++

```
virtual __fastcall SetCapacity(int NewCapacity);
```

Description

SetCapacity is the protected read implementation of the Capacity property.

Integer is the number of strings the list can hold after the capacity has changed.

In TStrings, SetCapacity does nothing. Descendant classes override this method to change the number of strings that the list can hold. For descendant classes that implement SetCapacity, assigning a value smaller than Count removes strings from the end of the list. Assigning a value greater than Count allocates space for more strings to be added.

See Also

Capacity (see page 3070)

Count (see page 3071)

23.1.48.3.22 TStringList.SetUpdateState Method

Performs internal adjustments before or after a series of updates.

Pascal

```
procedure SetUpdateState(Updating: Boolean); override;
```

C++

```
virtual __fastcall SetUpdateState(Boolean Updating);
```

Description

SetUpdateState is called at the beginning or end of a series of updates. When the BeginUpdate method is first called and the TStrings object is not already in the middle of an update, TStrings calls SetUpdateState internally, with Updating set to true. When the EndUpdate method is called and it cancels out the last unmatched call to BeginUpdate, TStrings calls SetUpdateState internally, with Updating set to false.

As implemented in TStrings, SetUpdateState does nothing. Descendant classes can override this method to optimize the response to updates.

See Also

BeginUpdate (see page 3045)

EndUpdate (see page 3050)

23.1.48.3.23 TStringList.Sort Method

Sorts the strings in the list in ascending order.

Pascal

```
procedure Sort; virtual;
```

C++

```
virtual __fastcall Sort();
```

Description

Call Sort to sort the strings in a list that has the Sorted (see page 3026) property set to false. String lists with the Sorted (see page 3026) property set to true are automatically sorted.

Note: Sort uses AnsiCompareStr to sort the strings when CaseSensitive (see page 3025) is true and AnsiCompareText when CaseSensitive (see page 3025) is false. To provide your own comparison operator instead, use the CustomSort (see page 3014) method.

C++ Examples:

```
/*
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  int Index;
  TStringList* MyList = new TStringList();
  try
  {
    MyList->Add("Animals");
    MyList->Add("Flowers");
    MyList->Add("Ocean");
    MyList->Sort(); // Find will only work on sorted lists
    if (MyList->Find("Flowers", Index))
    {
      ListBox1->Items->AddStrings(MyList);
      Label1->Caption = "Flowers has an index value of " + AnsiString(Index);
    }
  }
  __finally
  {
    delete MyList;
  }
}
```

Delphi Examples:

```
{
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  MyList: TStringList;
  Index: Integer;
begin
  MyList := TStringList.Create;
  try
    MyList.Add('Animals');
    MyList.Add('Flowers');
    MyList.Add('Cars');
    MyList.Sort;   { Find will only work on sorted lists! }
    if MyList.Find('Flowers', Index) then
    begin
      ListBox1.Items.AddStrings(MyList);
      Label1.Caption := 'Flowers has an index value of ' + IntToStr(Index);
    end;
```

```
    finally
      MyList.Free;
    end;
  end;
```

See Also

Sorted (see page 3026)

Strings

AnsiCompareStr (see page 22789)

CustomSort (see page 3014)

AnsiCompareText (see page 22790)

23.1.48.4 TStringList Properties

The properties of the TStringList class are listed here.

Public Properties

	Name	Description
	CaseSensitive (see page 3025)	Controls (see page 5711) whether strings are located, sorted, and identified as duplicates in a case-sensitive or case-insensitive manner.
	Duplicates (see page 3025)	Specifies whether duplicate strings can be added to sorted lists.
	Sorted (see page 3026)	Specifies whether the strings in the list should be automatically sorted.

23.1.48.4.1 TStringList.CaseSensitive Property

Controls (see page 5711) whether strings are located, sorted, and identified as duplicates in a case-sensitive or case-insensitive manner.

Pascal

```
property CaseSensitive: Boolean;
```

C++

```
__property Boolean CaseSensitive;
```

Description

Use CaseSensitive to indicate whether strings in the list should be compared in a case-sensitive or case-insensitive manner. Set CaseSensitive to true to make the string list locate, check for duplicates, and sort its strings in a case-sensitive manner. Set CaseSensitive to false to make the string list perform these operations case-insensitively.

See Also

CustomSort (see page 3014)

Sort (see page 3023)

23.1.48.4.2 TStringList.Duplicates Property

Specifies whether duplicate strings can be added to sorted lists.

Pascal

```
property Duplicates: TDuplicates;
```

C++

```
__property TDuplicates Duplicates;
```

Description

Set Duplicates to specify what should happen when an attempt is made to add a duplicate string to a sorted list. The CaseSensitive (see page 3025) property controls whether two strings are considered duplicates if they are identical except for differences in case.

The value of Duplicates should be one of the following.

Value	Meaning
dupIgnore	Ignore attempts to add duplicate strings to the list.
dupError	raise an EStringListError (see page 2705) exception when an attempt is made to add duplicate strings to the sorted list.
dupAccept	Permit duplicate strings in the sorted list.

Set Duplicates before adding any strings to the list. Setting Duplicates to dupIgnore or dupError does nothing about duplicate strings that are already in the list.

Note: Duplicates does nothing if the list is not sorted.

See Also

Add (see page 3010)

EStringListError (see page 2705)

Sorted (see page 3026)

CaseSensitive (see page 3025)

23.1.48.4.3 TStringList.Sorted Property

Specifies whether the strings in the list should be automatically sorted.

Pascal

```
property Sorted: Boolean;
```

C++

```
__property Boolean Sorted;
```

Description

Set Sorted to true to cause the strings in the list to be automatically sorted in ascending order. Set Sorted to false to allow strings to remain where they are inserted. When Sorted is false, the strings in the list can be put in ascending order at any time by calling the Sort (see page 3023) method.

When Sorted is true, do not use Insert (see page 3019) to add strings to the list. Instead, use Add (see page 3010), which will insert the new strings in the appropriate position. When Sorted is false, use Insert (see page 3019) to add strings to an arbitrary position in the list, or Add (see page 3010) to add strings to the end of the list.

Note: The CaseSensitive (see page 3025) property controls whether the strings in the list are sorted based on a case-sensitive or case-insensitive comparison. The sort order takes into account the locale of the system on which the application is running.

C++ Examples:

```
/*
This example uses a list box on a form. When the application
runs, a string list object is created and three strings are
added to it. The strings are sorted and added to the list
box, where they appear in their sorted order:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TStringList *list = new TStringList();
  list->Add("Plants");
  list->Add("Animals");
  list->Add("Minerals");
  list->Sorted = true;
  ListBox1->Items->AddStrings(list);
  delete list;
}
```

Delphi Examples:

```
{
This example uses a list box on a form. When the application
runs, a string list object is created and three strings are
added to it. The strings are sorted and added to the list
box, where they appear in their sorted order:
}
procedure TForm1.FormCreate(Sender: TObject);
var
  MyList: TStringList;
begin
  MyList := TStringList.Create;
  MyList.Add('Plants');
  MyList.Add('Animals');
  MyList.Add('Minerals');
  MyList.Sorted := True;
  ListBox1.Items.AddStrings(MyList);
  MyList.Free;
end;
```

See Also

Add (see page 3010)

Duplicates (see page 3025)

Insert (see page 3019)

Sort (see page 3023)

AnsiCompareStr (see page 22789)

CaseSensitive (see page 3025)

23.1.48.5 TStringList Events

The events of the TStringList class are listed here.

Public Events

	Name	Description
	OnChange (see page 3028)	Occurs immediately after the list of strings changes.
	OnChanging (see page 3028)	Occurs immediately before the list of strings changes.

23.1.48.5.1 TStringList.OnChange Event

Occurs immediately after the list of strings changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to respond to changes in the list of strings. For example, if the string list is associated with a control, the OnChange event handler could tell the control to repaint itself whenever the content of the list changes.

Whenever strings in the list are added, deleted, moved, or modified, the following events take place:

1An OnChanging (see page 3028) event occurs before the change.

2The strings are added, deleted, moved, or modified.

3An OnChange event occurs.

Note: OnChange occurs for every change made to the list, regardless of whether the application calls BeginUpdate (see page 3045) and EndUpdate (see page 3050) around a series of changes.

See Also

BeginUpdate (see page 3045)

EndUpdate (see page 3050)

OnChanging (see page 3028)

23.1.48.5.2 TStringList.OnChanging Event

Occurs immediately before the list of strings changes.

Pascal

```
property OnChanging: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChanging;
```

Description

Write an OnChanging event handler to prepare for changes in the list of strings. For example, if the string list is associated with a control, the OnChanging event handler could tell the control to disable repaints until the OnChange (see page 3028) event when the list has finished changing.

Whenever strings in the list are added, deleted, moved, or modified, the following events take place:

1An OnChanging event occurs.

2The strings are added, deleted, moved, or modified.

3An OnChange (see page 3028) event occurs after the changes are complete.

Note: OnChanging occurs for every change made to the list, regardless of whether the application calls BeginUpdate (see page 3045) and EndUpdate (see page 3050) around a series of changes.

See Also

BeginUpdate (see page 3045)

EndUpdate (see page 3050)

OnChange (see page 3028)

23.1.49 TStrings Class

TStrings is the base class for objects that represent a list of strings.

Class Hierarchy

Pascal

```
TStrings = class(TPersistent);
```

C++

```
class TStrings : public TPersistent;
```

File

Classes

Description

Derive a class from TStrings to store and manipulate a list of strings. TStrings contains abstract or, in C++ terminology, pure virtual methods and should not be directly instantiated.

Descendants of TStrings can represent several individual strings, such as the individual lines that appear in a list box. Some objects use descendants of TStrings to represent one long body of text so that it can be manipulated in smaller chunks.

TStrings introduces many properties and methods to

Add (see page 3033) or delete strings at specified positions in the list.

Rearrange the strings in the list.

Access the string at a particular location.

Read the strings from or write the strings to a file or stream.

Associate an object with each string in the list.

Store and retrieve strings as name-value pairs.

For an explanation of name-value pairs, refer to the NameValueSeparator (see page 3077) property.

See Also

TStringList (see page 3006)

TList (see page 2868)

THashedStringList (see page 15450)

Specifying the HTML Template

23.1.49.1 TStrings Members

The following tables list the members exposed by TStrings.

Protected Methods

	Name	Description
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	CompareStrings (see page 3048)	Compares two strings.
V	DefineProperties (see page 3049)	Reads and writes the Strings (see page 3080) property as if it were published.
	Error (see page 3052)	This is the overview for the Error method overload.
	ExtractName (see page 3053)	Returns the name portion of a string that is a name value pair.
A	Get (see page 3053)	Returns a string given its index.
V	GetCapacity (see page 3054)	Returns the currently allocated size of the list of strings.
A	GetCount (see page 3054)	Returns the number of strings in the list
V	GetObject (see page 3055)	Returns the object associated with the string at a specified index.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetTextStr (see page 3056)	Returns the value of the Text (see page 3084) property.
V	Put (see page 3064)	Changes the value of the string with a specified index.
V	PutObject (see page 3065)	Changes the object associated with the string at a specified index.
V	SetCapacity (see page 3068)	Changes the amount of memory allocated to hold strings in the list.
V	SetTextStr (see page 3069)	Sets the Text (see page 3084) property.
V	SetUpdateState (see page 3069)	Performs internal adjustments before or after a series of updates.

Protected Properties

	Name	Description
R	UpdateCount (see page 3085)	Indicates the number of calls to BeginUpdate (see page 3045) that have not been matched by a call to EndUpdate (see page 3050).

Public Methods

	Name	Description
V	Add (see page 3033)	Adds a string at the end of the list.
V	AddObject (see page 3036)	Adds a string to the list, and associates an object with the string.
V	AddStrings (see page 3039)	Adds a group of strings to the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 3041)	Adds a string to the list.
V	Assign (see page 3043)	Sets the strings in the list, and possibly associated objects, from a source object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 3045)	Enables the TStrings (see page 3029) object to track when the list of strings is changing.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 3047)	Introduces an abstract (Delphi) or pure virtual (C++) method to empty the list and any associated objects.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	Delete (see page 3049)	Introduces an abstract (Delphi) or pure virtual (C++) method to delete a specified string from the list.
V	Destroy (see page 3032)	Destroys the TPersistent instance and frees its memory.

V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 3050)	Enables the TStrings (see page 3029) object to keep track of when the list of strings has finished changing.
	Equals (see page 3051)	Compares the list of strings to the list from another TStrings (see page 3029) object and returns true if the two lists match.
V	Exchange (see page 3053)	Swaps the position of two strings in the list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 3054)	Returns a TStrings (see page 3029) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetText (see page 3055)	Allocates a text buffer and fills it with the value of the Text (see page 3084) property.
V	IndexOf (see page 3056)	Returns the position of a string in the list.
V	IndexOfName (see page 3057)	Returns the position of the first name-value pair with the specified name.
V	IndexOfObject (see page 3059)	Returns the index of the first string in the list associated with a given object.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
A	Insert (see page 3059)	Introduces abstract (Delphi) or pure virtual (C++) method to insert a string at a specified position.
V	InsertObject (see page 3060)	Inserts a string into the list at the specified position, and associates it with an object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	LoadFromFile (see page 3061)	Fills the list with the lines of text in a specified file.
V	LoadFromStream (see page 3062)	Fills the list with lines of text read from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	Move (see page 3063)	Changes the position of a string in the list.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SaveToFile (see page 3065)	Saves the strings in the list to the specified file.
V	SaveToStream (see page 3067)	Writes the value of the Text (see page 3084) property to a stream object.
V	SetText (see page 3068)	Sets the Text (see page 3084) property.

Public Properties

	Name	Description
	Capacity (see page 3070)	Indicates the number of strings the TStrings (see page 3029) object can hold.
	CommaText (see page 3070)	Lists the strings in the TStrings (see page 3029) object in system data format (SDF).
R	Count (see page 3071)	Introduces an abstract property to represent the number of strings in the list.
	DelimitedText (see page 3074)	Represents all the strings in the TStrings (see page 3029) object as a single delimited string.
	Delimiter (see page 3074)	Specifies the delimiter used by the DelimitedText (see page 3074) property.
	LineBreak (see page 3075)	Defines the line break character(s).
R	Names (see page 3075)	Indicates the name part of strings that are name-value pairs.
	NameValueSeparator (see page 3077)	Indicates the character used to separate names from values.
	Objects (see page 3077)	Represents a set of objects that are associated one with each of the strings in the Strings (see page 3080) property.
	QuoteChar (see page 3080)	Specifies the quote character used by the DelimitedText (see page 3074) property.
	StrictDelimiter (see page 3080)	Determines how the Delimiter (see page 3074) property is used.

	Strings (see page 3080)	References the strings in the list by their positions.
	StringsAdapter (see page 3084)	Implements an IStringsAdapter (see page 3208) interface for the TStrings (see page 3029) object.
	Text (see page 3084)	Lists the strings in the TStrings (see page 3029) object as a single string with the individual strings delimited by carriage returns and line feeds.
	ValueFromIndex (see page 3085)	Represents the value part of a string with a given index, on strings that are name-value pairs.
	Values (see page 3086)	Represents the value part of a string associated with a given name, on strings that are name-value pairs.

23.1.49.2 TStrings.Destroy Destructor

Destroys the TPersistent instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TStrings();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

Free (see page 22300)

Destroy (see page 22287)

23.1.49.3 TStrings Methods

The methods of the TStrings class are listed here.

Protected Methods

	Name	Description
	CompareStrings (see page 3048)	Compares two strings.
	DefineProperties (see page 3049)	Reads and writes the Strings (see page 3080) property as if it were published.
	Error (see page 3052)	This is the overview for the Error method overload.
	ExtractName (see page 3053)	Returns the name portion of a string that is a name value pair.
	Get (see page 3053)	Returns a string given its index.
	GetCapacity (see page 3054)	Returns the currently allocated size of the list of strings.
	GetCount (see page 3054)	Returns the number of strings in the list
	GetObject (see page 3055)	Returns the object associated with the string at a specified index.
	GetTextStr (see page 3056)	Returns the value of the Text (see page 3084) property.
	Put (see page 3064)	Changes the value of the string with a specified index.
	PutObject (see page 3065)	Changes the object associated with the string at a specified index.
	SetCapacity (see page 3068)	Changes the amount of memory allocated to hold strings in the list.
	SetTextStr (see page 3069)	Sets the Text (see page 3084) property.
	SetUpdateState (see page 3069)	Performs internal adjustments before or after a series of updates.

Public Methods

	Name	Description
	Add (see page 3033)	Adds a string at the end of the list.
	AddObject (see page 3036)	Adds a string to the list, and associates an object with the string.
	AddStrings (see page 3039)	Adds a group of strings to the list.

	Append (see page 3041)	Adds a string to the list.
V	Assign (see page 3043)	Sets the strings in the list, and possibly associated objects, from a source object.
	BeginUpdate (see page 3045)	Enables the TStrings (see page 3029) object to track when the list of strings is changing.
A	Clear (see page 3047)	Introduces an abstract (Delphi) or pure virtual (C++) method to empty the list and any associated objects.
A	Delete (see page 3049)	Introduces an abstract (Delphi) or pure virtual (C++) method to delete a specified string from the list.
	EndUpdate (see page 3050)	Enables the TStrings (see page 3029) object to keep track of when the list of strings has finished changing.
	Equals (see page 3051)	Compares the list of strings to the list from another TStrings (see page 3029) object and returns true if the two lists match.
V	Exchange (see page 3053)	Swaps the position of two strings in the list.
	GetEnumerator (see page 3054)	Returns a TStrings (see page 3029) enumerator.
V	GetText (see page 3055)	Allocates a text buffer and fills it with the value of the Text (see page 3084) property.
V	IndexOf (see page 3056)	Returns the position of a string in the list.
V	IndexOfName (see page 3057)	Returns the position of the first name-value pair with the specified name.
V	IndexOfObject (see page 3059)	Returns the index of the first string in the list associated with a given object.
A	Insert (see page 3059)	Introduces abstract (Delphi) or pure virtual (C++) method to insert a string at a specified position.
V	InsertObject (see page 3060)	Inserts a string into the list at the specified position, and associates it with an object.
V	LoadFromFile (see page 3061)	Fills the list with the lines of text in a specified file.
V	LoadFromStream (see page 3062)	Fills the list with lines of text read from a stream.
V	Move (see page 3063)	Changes the position of a string in the list.
V	SaveToFile (see page 3065)	Saves the strings in the list to the specified file.
V	SaveToStream (see page 3067)	Writes the value of the Text (see page 3084) property to a stream object.
V	SetText (see page 3068)	Sets the Text (see page 3084) property.

23.1.49.3.1 TStrings.Add Method

Adds a string at the end of the list.

Pascal

```
function Add(const S: string): Integer; virtual;
```

C++

```
virtual __fastcall int Add(const AnsiString S);
```

Description

Call Add to add a string to the end of the list. Add returns the index of the new string.

C++ Examples:

```
/*
This example adds text to a TRichEdit and to a StatusBar.
This example requires no additional controls. All controls are
created dynamically within the form's OnCreate event handler.
Note: Be sure to add ComCtrls to your uses clause to support
TRichEdit.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TStatusBar *status = new TStatusBar(this);
  // Display the StatusBar control on the surface
  // of the form.
  status->Parent = this;
```

```
  // Allow a single line of text -- no panels.
  status->SimplePanel = true;
  // Provide sample text.
  // Create a new instance of TRichEdit, set its
  // properties, and display on the form.
  TRichEdit *rich = new TRichEdit(this);
  // Display the RichEdit control on the surface
  // of the form.
  rich->Parent = this;
  // The RichText control takes all space left
  // over from the StatusBar.
  rich->Align = alClient;
  // Add 50 lines to the RichText
  for (int i = 1; i <= 50; i++)
    rich->Lines->Add(AnsiString("This is line ") + IntToStr(i));
  rich->SelStart = 0; // set text cursor to first character
  status->SimpleText = IntToStr(rich->Lines->Count) + " Lines loaded";
}
/*
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
*/
void __fastcall TForm1::Add_a_fileClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    int index = TabControl1->Tabs->Add(OpenDialog1->FileName);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[index]);
  }
}

void __fastcall TForm1::Assign_filesClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TabControl1->Tabs->Assign(OpenDialog1->Files);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[TabControl1->TabIndex]);
  }
}
/*
Place the following code in the tab control’s OnChange event
handler:
*/
void __fastcall TForm1::TabControl1Change(TObject *Sender)
{
    Memo1->Lines->LoadFromFile(
      TabControl1->Tabs->Strings[TabControl1->TabIndex]);
}
```

Delphi Examples:

```
{
This example adds text to a TRichEdit and to a StatusBar.
This example requires no additional controls. All controls are
created dynamically within the form's OnCreate event handler.
Note: Be sure to add ComCtrls to your uses clause to support
TRichEdit.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  with TStatusBar.Create(Self) do
  begin
    // Display the StatusBar control on the surface
    // of the form.
    Parent := Self;
    // Allow a single line of text -- no panels.
    SimplePanel := True;
    // Provide sample text.
    // Create a new instance of TRichEdit, set its
    // properties, and display on the form.
    with TRichEdit.Create(Self) do
    begin
      // Display the RichEdit control on the surface
      // of the form.
      Parent := Self;
      // The RichText control takes all space left
      // over from the StatusBar.
      Align := alClient;
      // Add 50 lines to the RichText
      for i := 1 to 50 do
        Lines.Add('Adding line ' + IntToStr(i));
      SelStart := 0; // set text cursor to first character
      SimpleText := 'Number of Lines loaded to RichText: ' +
        IntToStr(Lines.Count);
    end;
  end;
end;
{
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
}
procedure TForm1.Add_a_fileClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    index:= TabControl1.Tabs.Add(OpenDialog1.FileName);
    Memo1.Lines.LoadFromFile(TabControl1.Tabs[index]);
  end;
end;

procedure TForm1.Assign_filesClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    TabControl1.Tabs.Assign(OpenDialog1.Files);
    Memo1.Lines.LoadFromFile(
      TabControl1.Tabs[TabControl1.TabIndex]);
```

```
    end;
  end;

  {
  Place the following code in the tab control's OnChange event
  handler:
  }
  procedure TForm1.TabControl1Change(Sender: TObject);
  begin
    with TabControl1 do
      Memo1.Lines.LoadFromFile(Tabs[TabIndex]);
  end;
```

See Also

AddStrings (see page 3039)

Append (see page 3041)

Count (see page 3071)

IndexOf (see page 3056)

Insert (see page 3059)

23.1.49.3.2 TStrings.AddObject Method

Adds a string to the list, and associates an object with the string.

Pascal

```
function AddObject(const S: string; AObject: TObject): Integer; virtual;
```

C++

```
virtual __fastcall int AddObject(const AnsiString S, TObject * AObject);
```

Description

Call AddObject to add a string and its associated object to the list. AddObject returns the index of the new string and object.

Note: The TStrings (see page 3029) object does not own the objects you add this way. Objects (see page 3077) added to the TStrings (see page 3029) object still exist even if the TStrings (see page 3029) instance is destroyed. They must be explicitly destroyed by the application.

C++ Examples:

```
/*
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
*/
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
```

```
  if (pCB)
    PageControl1->ActivePage = (TTabSheet *)(pCB->Items->Objects[pCB->ItemIndex]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
    ComboBox1->Items->AddObject(PageControl1->Pages[i]->Name, PageControl1->Pages[i]);
    ComboBox1->ItemIndex = 0;
  }
}
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", (TObject *)vsIcon);
  ComboBox1->Items->AddObject("vsList", (TObject *)vsList);
  ComboBox1->Items->AddObject("vsReport", (TObject *)vsReport);
  ComboBox1->Items->AddObject("vsSmallIcon", (TObject *)vsSmallIcon);
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
```

```
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
```

Delphi Examples:

```
{
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to PageControl1.PageCount - 1 do
    ComboBox1.Items.AddObject(PageControl1.Pages[i].Name,
      PageControl1.Pages[i]);
  ComboBox1.ItemIndex := 0;
end;

procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  if (Sender is TComboBox) then
    with (Sender as TComboBox) do
      PageControl1.ActivePage := TTabSheet(Items.Objects[ItemIndex]);
end;
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
```

```
      for I := 0 to ImageList1.Count - 1 do
      begin
        ListItem := Items.Add;
        Listitem.Caption := 'Image' + IntToStr(I);
        ListItem.ImageIndex := I;
        Listitem.SubItems.Add(Col2Array[I]);
      end;
      // Create two columns to show during viewing as vsReport
      NewColumn := Columns.Add;
      NewColumn.Caption := 'Column 1';
      NewColumn.Width := 200;
      NewColumn := Columns.Add;
      NewColumn.Caption := 'Column 2';
      NewColumn.Width := 200;
      // Add View styles and constants to the Combo Box
      ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
      ComboBox1.Items.AddObject('vsList', TObject(vsList));
      ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
      ComboBox1.Items.AddObject(
        'vsSmallIcon', TObject(vsSmallIcon));
      // Display first item in the Combo Box
      ComboBox1.ItemIndex := 0;
    end;
  end;

  procedure TComboForm.ComboBox1Click(Sender: TObject);
  begin
    with ComboBox1 do
      ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

  end;
```

See Also

Add (see page 3033)

AddStrings (see page 3039)

Count (see page 3071)

IndexOfObject (see page 3059)

Objects (see page 3077)

23.1.49.3.3 TStrings.AddStrings Method

Adds a group of strings to the list.

Pascal

```
procedure AddStrings(Strings: TStrings); virtual;
```

C++

```
virtual __fastcall AddStrings(TStrings Strings);
```

Description

Call AddStrings to add the strings from another TStrings (see page 3029) object to the list. If both the source and destination TStrings (see page 3029) objects support objects associated with their strings, references to the associated objects will be added as well.

C++ Examples:

```
/*
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
```

```
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  int Index;
  TStringList* MyList = new TStringList();
  try
  {
    MyList->Add("Animals");
    MyList->Add("Flowers");
    MyList->Add("Ocean");
    MyList->Sort(); // Find will only work on sorted lists
    if (MyList->Find("Flowers", Index))
    {
      ListBox1->Items->AddStrings(MyList);
      Label1->Caption = "Flowers has an index value of " + AnsiString(Index);
    }
  }
  __finally
  {
    delete MyList;
  }
}
/*
This example uses a list box on a form. When the application
runs, a string list object is created and three strings are
added to it. The strings are sorted and added to the list
box, where they appear in their sorted order:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TStringList *list = new TStringList();
  list->Add("Plants");
  list->Add("Animals");
  list->Add("Minerals");
  list->Sorted = true;
  ListBox1->Items->AddStrings(list);
  delete list;
}
```

Delphi Examples:

```
{
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  MyList: TStringList;
  Index: Integer;
begin
  MyList := TStringList.Create;
  try
    MyList.Add('Animals');
    MyList.Add('Flowers');
    MyList.Add('Cars');
    MyList.Sort;   { Find will only work on sorted lists! }
    if MyList.Find('Flowers', Index) then
```

```
      begin
        ListBox1.Items.AddStrings(MyList);
        Label1.Caption := 'Flowers has an index value of ' + IntToStr(Index);
      end;
    finally
      MyList.Free;
    end;
  end;
  {
  This example uses a list box on a form. When the application
  runs, a string list object is created and three strings are
  added to it. The strings are sorted and added to the list
  box, where they appear in their sorted order:
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    MyList: TStringList;
  begin
    MyList := TStringList.Create;
    MyList.Add('Plants');
    MyList.Add('Animals');
    MyList.Add('Minerals');
    MyList.Sorted := True;
    ListBox1.Items.AddStrings(MyList);
    MyList.Free;
  end;
```

See Also

Add (see page 3033)

AddObject (see page 3036)

Assign (see page 3043)

Count (see page 3071)

Objects (see page 3077)

23.1.49.3.4 TStrings.Append Method

Adds a string to the list.

Pascal

```
procedure Append(const S: string);
```

C++

```
__fastcall Append(const AnsiString S);
```

Description

Append is the same as the Add (see page 3033) method, except that it does not return a value. Use Append when there is no need to know the index of the string after it has been added, or with descendants of TStrings (see page 3029) for which the index returned is not meaningful.

For example, the TStrings (see page 3029) descendant used by memo objects uses an index to determine where to insert a string, but the inserted string does not necessarily end up as a single string in the list. Part of the inserted text may become part of the previous string, and part may be broken off into a subsequent string. The index returned by Add (see page 3033) is not meaningful in this case.

Use Append rather than Add (see page 3033) as a parameter for a function requiring a TGetStrProc (see page 3294).

C++ Examples:

```
/*
This example uses an Open dialog box, a memo, and a button
on a form. When the user clicks the button, the Open dialog
box appears. When the user selects files in the dialog box
and chooses the OK button, the first line from each of the
files is added to the memo. Choose multiple files using the
CNTL key or the SHIFT key.
*/

#include <stdio.h>     // for FILE

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FILE *stream;
  char FirstLine[512];

  OpenDialog1->Options.Clear();
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist;
  OpenDialog1->Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
  OpenDialog1->FilterIndex = 2; // start the dialog showing all files
  if (OpenDialog1->Execute())
  {
    for (int I = 0; I < OpenDialog1->Files->Count; I ++)
    {
      stream = fopen(OpenDialog1->Files->Strings[I].c_str(), "r");
      if (stream)
      {
        // read the first line from the file
        fgets(FirstLine, sizeof(FirstLine), stream);
        Memo1->Lines->Append(FirstLine);
        fclose(stream);
      }
    }
  }
}
```

Delphi Examples:

```
{
This example uses an Open dialog box, a memo, and a button
on a form. When the user clicks the button, the Open dialog
box appears. When the user selects files in the dialog box
and chooses the OK button, the first line from each of the
files is added to the memo. Choose multiple files using the
CNTL key or the SHIFT key.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: integer;
  F: TextFile;
  FirstLine: string;
begin
  OpenDialog1.Options := [ofAllowMultiSelect, ofFileMustExist];
  OpenDialog1.Filter := 'Text files (*.txt)|*.txt|All files (*.*)|*.*';
  OpenDialog1.FilterIndex := 2; { start the dialog showing all files }
  if OpenDialog1.Execute then
    with OpenDialog1.Files do
      for I := 0 to Count - 1 do
      begin
        AssignFile(F, Strings[I]);  { next file in Files property }
        Reset(F);
        Readln(F, FirstLine);  { Read the first line out of the file }
        Memo1.Lines.Append(FirstLine);  { Add the line to the memo }
        CloseFile(F);
      end;
end;
```

See Also

Add (see page 3033)

Lines (see page 21380)

TGetStrProc (see page 3294)

23.1.49.3.5 TStrings.Assign Method

Sets the strings in the list, and possibly associated objects, from a source object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use Assign to set the value of the TStrings (see page 3029) object from another object. If Source is of type TStrings (see page 3029), the list is set to the list of the source TStrings (see page 3029) object, and if associated objects are supported, any associated objects are copied from the Source as well.

If Source is not of type TStrings (see page 3029), the inherited Assign will set the value of the list from any object that supports TStrings (see page 3029) in its AssignTo (see page 2931) method.

C++ Examples:

```
/*
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
*/
void __fastcall TForm1::Add_a_fileClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    int index = TabControl1->Tabs->Add(OpenDialog1->FileName);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[index]);
  }
}

void __fastcall TForm1::Assign_filesClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TabControl1->Tabs->Assign(OpenDialog1->Files);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[TabControl1->TabIndex]);
  }
}
/*
Place the following code in the tab control's OnChange event
handler:
```

```
*/
void __fastcall TForm1::TabControl1Change(TObject *Sender)
{
    Memo1->Lines->LoadFromFile(
      TabControl1->Tabs->Strings[TabControl1->TabIndex]);
}
/*
This example creates a stringlist, adds elements to it and
assigns the strings to items of a combo box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TStrings *StringList = new TStringList();
  try
  {
    StringList->Add("This example uses A string List. ");
    StringList->Add("It is the easiest way to add strings");
    StringList->Add("to a combobox's list of strings. ");
    StringList->Add("Always remember TStrings is abstract,");
    StringList->Add("So create a TStringList instead.");
    ComboBox1->Width = 210;
    ComboBox1->Items->Assign(StringList);
    ComboBox1->ItemIndex = 0;
  }
  __finally
  {
    delete StringList;
  }
}
```

Delphi Examples:

```
{
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
}
procedure TForm1.Add_a_fileClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    index:= TabControl1.Tabs.Add(OpenDialog1.FileName);
    Memo1.Lines.LoadFromFile(TabControl1.Tabs[index]);
  end;
end;

procedure TForm1.Assign_filesClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    TabControl1.Tabs.Assign(OpenDialog1.Files);
    Memo1.Lines.LoadFromFile(
      TabControl1.Tabs[TabControl1.TabIndex]);
  end;
```

```
end;

{
Place the following code in the tab control's OnChange event
handler:
}
procedure TForm1.TabControl1Change(Sender: TObject);
begin
  with TabControl1 do
    Memo1.Lines.LoadFromFile(Tabs[TabIndex]);
end;
{
This example creates a stringlist, adds elements to it and
assigns the strings to items of a combo box.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  StringList: TStringList;
begin
  StringList := TStringList.Create;
  try
    with StringList do begin
      Add('This example uses A string List.');
      Add('It is the easiest way to add strings');
      Add('to a combobox''s list of strings.');
      Add('Always remember TStrings.Create method');
      Add('is abstract; So use TStringList.Create');
      Add('method instead.');
    end;
    with ComboBox1 do begin
      Width := 210;
      Items.Assign(StringList);
      ItemIndex := 0;
    end;
  finally
    StringList.free;
  end;
end;
```

See Also

AddStrings (see page 3039)

Objects (see page 3077)

Strings (see page 3080)

AssignTo (see page 2931)

23.1.49.3.6 TStrings.BeginUpdate Method

Enables the TStrings (see page 3029) object to track when the list of strings is changing.

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

BeginUpdate is called automatically by any property or method that changes the list of strings. Once the changes are complete, the property or method calls EndUpdate (see page 3050). Call BeginUpdate before directly modifying the strings in the list, and EndUpdate (see page 3050) after. When implementing properties or methods that change the list in descendants of

TStrings (see page 3029), call BeginUpdate before the changes are made, and EndUpdate (see page 3050) when the changes are complete.

TStrings (see page 3029) simply keeps track of when the list of strings is being changed. Some descendants of TStrings (see page 3029) use this information to perform certain actions, such as telling a control to repaint, when updates are complete.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if (((TForm1 *)Screen->CustomForms[i])->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
    end;
    Items.EndUpdate; {reenable painting }
```

```
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.Show;
end;
```

See Also

EndUpdate (see page 3050)

UpdateCount (see page 3085)

23.1.49.3.7 TStrings.Clear Method

Introduces an abstract (Delphi) or pure virtual (C++) method to empty the list and any associated objects.

Pascal

```
procedure Clear; virtual; abstract;
```

C++

```
virtual __fastcall Clear() = 0;
```

Description

Descendants of TStrings (see page 3029) implement a Clear method to delete all the strings in the list, and to remove any references to associated objects.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if (((TForm1 *)Screen->CustomForms[i])->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
```

```
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
    end;
    Items.EndUpdate; {reenable painting }
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.Show;
end;
```

See Also

Delete (see page 3049)

Objects (see page 3077)

Strings (see page 3080)

23.1.49.3.8 TStrings.CompareStrings Method

Compares two strings.

Pascal

```
function CompareStrings(const S1: string; const S2: string): Integer; virtual;
```

C++

```
virtual __fastcall int CompareStrings(const AnsiString S1, const AnsiString S2);
```

Description

TStrings (see page 3029) uses CompareStrings internally to compare the values of strings that appear in the list. For example, the IndexOf (see page 3056) and IndexOfName (see page 3057) methods use CompareStrings to compare a specified string with the strings in the list.

S1 and S2 are the strings to compare.

CompareStrings returns a value less than 0 if S1 < S2, 0 if S1 == S2, and a value greater than 0 if S1 > S2.

As implemented in TStrings (see page 3029), CompareStrings uses the global AnsiCompareText function, which compares strings case insensitively. Some descendant classes override this method to change the way strings are compared (for example, to introduce case sensitivity).

See Also

AnsiCompareText (see page 22790)

IndexOf (see page 3056)

IndexOfName (see page 3057)

23.1.49.3.9 TStrings.DefineProperties Method

Reads and writes the Strings (see page 3080) property as if it were published.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TStrings (see page 3029) overrides DefineProperties so that the strings in the list can be loaded and saved with a form file as if the Strings (see page 3080) property were published.

See Also

DefineProperties (see page 2931)

Strings (see page 3080)

23.1.49.3.10 TStrings.Delete Method

Introduces an abstract (Delphi) or pure virtual (C++) method to delete a specified string from the list.

Pascal

```
procedure Delete(Index: Integer); virtual; abstract;
```

C++

```
virtual __fastcall Delete(int Index) = 0;
```

Description

Descendants of TStrings (see page 3029) implement a Delete method to remove a specified string from the list. If an object is associated with the string, the reference to the object is removed as well. Index gives the position of the string, where 0 is the first string, 1 is the second string, and so on.

See Also

AddObject (see page 3036)

Clear (see page 3047)

IndexOf (see page 3056)

InsertObject (see page 3060)

Objects (see page 3077)

Strings (see page 3080)

23.1.49.3.11 TStrings.EndUpdate Method

Enables the TStrings (see page 3029) object to keep track of when the list of strings has finished changing.

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

EndUpdate is called automatically by any property or method that changes the list of strings. Call BeginUpdate (see page 3045) before directly modifying the strings in the list, and EndUpdate after. When implementing properties or methods that change the list in descendants of TStrings (see page 3029), call BeginUpdate (see page 3045) before the changes are made, and EndUpdate when the changes are complete.

TStrings (see page 3029) simply keeps track of when the list of strings is being changed. Some descendants of TStrings (see page 3029) use this information to perform certain actions, such as telling a control to repaint, when updates are complete.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if (((TForm1 *)Screen->CustomForms[i])->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
    end;
    Items.EndUpdate; {reenable painting }
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.Show;
end;
```

See Also

BeginUpdate (see page 3045)

UpdateCount (see page 3085)

23.1.49.3.12 TStrings.Equals Method

Compares the list of strings to the list from another TStrings (see page 3029) object and returns true if the two lists match.

Pascal

```
function Equals(Strings: TStrings): Boolean;
```

C++

```
__fastcall Boolean Equals(TStrings Strings);
```

Description

Call Equals to compare the lists in two TStrings (see page 3029) objects. Equals compares only the strings, not any references to associated objects. Equals returns true if the lists for both TStrings (see page 3029) objects have the same number of strings and the strings in each list match. Equals returns false if the lists are different in length, if they contain different strings, or if the order of the strings in the two lists differ.

Note: The Equals method uses an overridden Get (see page 3053) method to perform the compare, thus it does not provide a context-insensitive option.

See Also

Count (see page 3071)

Strings (see page 3080)

CompareStrings (see page 3048)

23.1.49.3.13 Error Method

This is the overview for the Error method overload.

Overload List

	Name	Description
	TStrings.Error (PResStringRec, Integer) (see page 3052)	raises an EStringListError (see page 2705) exception.
	TStrings.Error (string, Integer) (see page 3052)	raises an EStringListError (see page 2705) exception.

23.1.49.3.13.1 TStrings.Error Method (PResStringRec, Integer)

raises an EStringListError (see page 2705) exception.

Pascal

```
procedure Error(const Msg: string; Data: Integer); overload;
procedure Error(Msg: PResStringRec; Data: Integer); overload;
```

C++

```
__fastcall Error(const AnsiString Msg, int Data);
__fastcall Error(PResStringRec Msg, int Data);
```

Description

TStrings (see page 3029) calls Error internally to raise an EStringListError (see page 2705) exception when it encounters a problem.

Msg specifies the string with a single format specifier for an integer, that appears in the exception message box. It can be either a string, or a pointer to a record structure that indicates the module and resource identifier for a string.

Data is an integer value that is inserted into Msg.

See Also

EStringListError (see page 2705)

23.1.49.3.14 TStrings.Exchange Method

Swaps the position of two strings in the list.

Pascal

```
procedure Exchange(Index1: Integer; Index2: Integer); virtual;
```

C++

```
virtual __fastcall Exchange(int Index1, int Index2);
```

Description

Call Exchange to rearrange the strings in the list. The strings are specified by their index values in the Index1 and Index2 parameters. Indexes are zero-based, so the first string in the list has an index value of 0, the second has an index value of 1, and so on.

If either string has an associated object, Exchange changes the position of the object as well.

See Also

Move (see page 3063)

Objects (see page 3077)

Strings (see page 3080)

23.1.49.3.15 TStrings.ExtractName Method

Returns the name portion of a string that is a name value pair.

Pascal

```
function ExtractName(const S: string): string;
```

C++

```
__fastcall AnsiString ExtractName(const AnsiString S);
```

Description

TStrings (see page 3029) calls ExtractName internally to parse strings that are name-value pairs and return the name portion.

S is the string to parse.

If S is a name-value pair, ExtractName returns the name portion. Otherwise, ExtractName returns an empty string.

See Also

Names (see page 3075)

23.1.49.3.16 TStrings.Get Method

Returns a string given its index.

Pascal

```
function Get(Index: Integer): string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString Get(int Index) = 0;
```

Description

Get is the protected read implementation of the Strings (see page 3080) property.

In TStrings (see page 3029) Get is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to return the string with the specified index.

See Also

Strings (see page 3080)

23.1.49.3.17 TStrings.GetCapacity Method

Returns the currently allocated size of the list of strings.

Pascal

```
function GetCapacity: Integer; virtual;
```

C++

```
virtual __fastcall int GetCapacity();
```

Description

GetCapacity is the protected read implementation of the Capacity (see page 3070) property. In TStrings (see page 3029), GetCapacity returns the value of the Count (see page 3071) property. Descendants of TStrings (see page 3029) can override this property to let a string list allocate memory for entries that have not been added to the list.

See Also

Capacity (see page 3070)

Count (see page 3071)

23.1.49.3.18 TStrings.GetCount Method

Returns the number of strings in the list

Pascal

```
function GetCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetCount() = 0;
```

Description

GetCount is the protected read implementation of the Count (see page 3071) property.

In TStrings (see page 3029) GetCount is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to return the number of strings that have been added to the list.

See Also

Count (see page 3071)

23.1.49.3.19 TStrings.GetEnumerator Method

Returns a TStrings (see page 3029) enumerator.

Pascal

```
function GetEnumerator: TStringsEnumerator;
```

C++

```
__fastcall TStringsEnumerator GetEnumerator();
```

Description

GetEnumerator returns a TStringsEnumerator (see page 3086) reference, which enumerates the strings in a TStrings (see page 3029) object.

To process all these strings, call the TStringsEnumerator (see page 3086) GetCurrent method within a While MoveNext do loop.

See Also

TStringsEnumerator (see page 3086)

23.1.49.3.20 TStrings.GetObject Method

Returns the object associated with the string at a specified index.

Pascal

```
function GetObject(Index: Integer): TObject; virtual;
```

C++

```
virtual __fastcall TObject * GetObject(int Index);
```

Description

GetObject is the protected read implementation of the Objects (see page 3077) property.

Index is the index of the string with which the object is associated.

In TStrings (see page 3029), GetObject always returns nil (Delphi) or NULL (C++). This provides a default implementation for descendants that do not support associating objects with the strings in the list. Descendants that support this feature override GetObject to return the specified object.

See Also

Objects (see page 3077)

23.1.49.3.21 TStrings.GetText Method

Allocates a text buffer and fills it with the value of the Text (see page 3084) property.

Pascal

```
function GetText: PChar; virtual;
```

C++

```
virtual __fastcall const char * GetText();
```

Description

Call GetText to obtain a dynamically allocated character buffer containing all of the strings in the list. Individual strings are separated by a carriage return and (on Windows) line feed. The caller is responsible for freeing the returned value using the StrDispose procedure.

See Also

SetText (see page 3068)

Strings (see page 3080)

Text (see page 3084)

StrDispose (see page 22947)

23.1.49.3.22 TStrings.GetTextStr Method

Returns the value of the Text (see page 3084) property.

Pascal

```
function GetTextStr: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetTextStr();
```

Description

GetTextStr is the protected read implementation of the Text (see page 3084) property. It returns a string that lists all the strings in the list, with individual strings separated by a carriage return and (on Windows) line feed.

See Also

GetText (see page 3055)

Text (see page 3084)

23.1.49.3.23 TStrings.IndexOf Method

Returns the position of a string in the list.

Pascal

```
function IndexOf(const S: string): Integer; virtual;
```

C++

```
virtual __fastcall int IndexOf(const AnsiString S);
```

Description

Call IndexOf to obtain the position of the first occurrence of the string S, or of a string that differs from S only by case. IndexOf returns the 0-based index of the string. Thus, if S matches the first string in the list, IndexOf returns 0, if S is the second string, IndexOf returns 1, and so on. If the string is not in the string list, IndexOf returns -1.

Note: If the string appears in the list more than once, IndexOf returns the position of the first occurrence.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, and
a label on a form. When the user uses the directory list box
to change directories, a message appears and the color of
the form changes if the file AUTOEXEC.BAT is in the new
directory. The code is written in the OnChange event of the
directory list box:
*/
void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
  if (FileListBox1->Items->IndexOf("AUTOEXEC.BAT") > -1)
  {
```

```
    Color = clYellow;
    Label1->Caption = "You are in the root directory!";
  }
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, and
a label on a form. When the user uses the directory list box
to change directories, a message appears and the color of
the form changes if the file AUTOEXEC.BAT is in the new
directory. The code is written in the OnChange event of the
directory list box:
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
  if FileListBox1.Items.IndexOf('AUTOEXEC.BAT') > -1 then
  begin
    Color := clYellow;
    Label1.Caption := 'You are in the root directory!';
  end
  else
  begin
    Color := clBtnFace;
    Label1.Caption := '';
  end;
end;
```

See Also

IndexOfName (see page 3057)

IndexOfObject (see page 3059)

Strings (see page 3080)

23.1.49.3.24 TStrings.IndexOfName Method

Returns the position of the first name-value pair with the specified name.

Pascal

```
function IndexOfName(const Name: string): Integer; virtual;
```

C++

```
virtual __fastcall int IndexOfName(const AnsiString Name);
```

Description

Call IndexOfName to locate the first occurrence of a name-value pair where the name part is equal to the Name parameter or differs only in case. IndexOfName returns the 0-based index of the string. If no string in the list has the indicated name, IndexOfName returns -1.

Note: If there is more than one name-value pair with a name portion matching the Name parameter, IndexOfName returns the position of the first such string.

C++ Examples:

```
/*
The following example updates the strings in a list box
given the strings contained in another list box. If a string
in the source list box has the form Name=Value and the
```

```
destination list box contains a string with the same Name
part, the Value part in the destination list box will be
replaced by the source's value.  TListBox objects that are
not name-value pairs are not assigned a Name and IndexOfName
will return -1.  TStrings Values are referenced using the
name AnsiString as an index.
*/
void MergeStrings(TStrings *Dest, TStrings *Source)
{
  for (int i = 0; i < Source->Count; i++)
  {
    if (Source->Strings[i].Pos("=") > 1)
    {
      int DI = Dest->IndexOfName(Source->Names[i]);
      if (DI > -1)
        Dest->Values[Source->Names[i]] = Source->Values[Source->Names[i]];
    }
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  MergeStrings(ListBox1->Items,ListBox2->Items);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Plants = 10");
  ListBox1->Items->Add("Animals = 20");
  ListBox1->Items->Add("Minerals = 15");
  ListBox2->Items->Add("Animals = 4");
  ListBox2->Items->Add("Plants = 3");
  ListBox2->Items->Add("Minerals = 78");
}
```

Delphi Examples:

```
{
The following example updates the strings in a list box
given the strings contained in another list box. If a string
in the source list box has the form Name=Value and the
destination list box contains a string with the same Name
part, the Value part in the destination list box will be
replaced by the source's value.  TListBox objects that are
not name-value pairs are not assigned a Name and IndexOfName
will return -1.
}
procedure MergeStrings(Dest, Source: TStrings);
var
  I, DI: Integer;
  begin
  for I := 0 to Source.Count - 1 do
  begin
    if Pos ('=', Source[I]) > 1 then
    begin
      DI := Dest.IndexOfName(Source.Names[I]);
      if DI > -1 then Dest[DI] := Source[I];
    end;
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  MergeStrings(ListBox1.Items, ListBox2.Items);
end;

procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
  ListBox1.Items.Add('Plants = 10');
  ListBox1.Items.Add('Animals = 20');
  ListBox1.Items.Add('Minerals = 15');
  ListBox2.Items.Add('Animals = 4');
  ListBox2.Items.Add('Plants = 3');
  ListBox2.Items.Add('Minerals = 78');
end;
```

See Also

IndexOf (see page 3056)

Names (see page 3075)

Values (see page 3086)

23.1.49.3.25 TStrings.IndexOfObject Method

Returns the index of the first string in the list associated with a given object.

Pascal

```
function IndexOfObject(AObject: TObject): Integer; virtual;
```

C++

```
virtual __fastcall int IndexOfObject(TObject * AObject);
```

Description

Call IndexOfObject to locate the first string in the list associated with the object AObject. Specify the object you want to locate as the value of the AObject parameter. IndexOfObject returns the 0-based index of the string and object. If the object is not associated with any of the strings, IndexOfObject returns -1.

See Also

IndexOf (see page 3056)

Objects (see page 3077)

23.1.49.3.26 TStrings.Insert Method

Introduces abstract (Delphi) or pure virtual (C++) method to insert a string at a specified position.

Pascal

```
procedure Insert(Index: Integer; const S: string); virtual; abstract;
```

C++

```
virtual __fastcall Insert(int Index, const AnsiString S) = 0;
```

Description

Descendants of TStrings (see page 3029) implement an Insert method to add the string S to the list at the position specified by Index. If Index is 0, the string is inserted at the beginning of the list. If Index is 1, the string is put in the second position of the list, and so on.

All methods that add strings to the list use the Insert method to add the string.

If the string has an associated object, use the InsertObject (see page 3060) method instead.

See Also

Add (see page 3033)

AddObject (see page 3036)

AddStrings (see page 3039)

Append (see page 3041)

BeginUpdate (see page 3045)

EndUpdate (see page 3050)

InsertObject (see page 3060)

Strings (see page 3080)

23.1.49.3.27 TStrings.InsertObject Method

Inserts a string into the list at the specified position, and associates it with an object.

Pascal

```
procedure InsertObject(Index: Integer; const S: string; AObject: TObject); virtual;
```

C++

```
virtual __fastcall InsertObject(int Index, const AnsiString S, TObject * AObject);
```

Description

Call InsertObject to insert the string S into the list at the position identified by Index, and associate it with the object AObject. If Index is 0, the string is inserted at the beginning of the list. If Index is 1, the string is put in the second position of the list, and so on.

C++ Examples:

```
/*
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < ComponentCount; i++)
    ListBox1->Items->InsertObject(0,
                                  Components[i]->Name,
                                  (TObject *)Components[i]);
}

void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  if (Button == mbRight)
  {
    TClass ClassRef;
    int Index = ListBox1->ItemAtPos(Point(X,Y), true);
    // only components that are controls have a position
    // make sure the component is a control
```

```
    for (ClassRef = ListBox1->Items->Objects[Index]->ClassType();
         ClassRef != NULL;
         ClassRef = ClassRef->ClassParent())
      if (String(ClassRef->ClassName()) == "TControl")
      {
        TControl *TheObject = (TControl *)ListBox1->Items->Objects[Index];
        StatusBar1->SimpleText =
          TheObject->Name + " is at (" +
          IntToStr(TheObject->Left) + ", " +
          IntToStr(TheObject->Top) + ")";
         break;
      }
    if (ClassRef == NULL) // if it wasn't a control
      MessageBeep(0);
  }
}
```

See Also

Insert (see page 3059)

Objects (see page 3077)

23.1.49.3.28 TStrings.LoadFromFile Method

Fills the list with the lines of text in a specified file.

Pascal

```
procedure LoadFromFile(const FileName: string); virtual;
```

C++

```
virtual __fastcall LoadFromFile(const AnsiString FileName);
```

Description

Call LoadFromFile to fill the list of the TStrings (see page 3029) object from the file specified by FileName. LoadFromFile first clears any strings already in the list. Then, each line in the file, as indicated by carriage return or linefeed characters, is appended as a string in the list.

Note: LoadFromFile uses the Add (see page 3033) method to add the strings that are read from the file.

C++ Examples:

```
/*
This example requires two TRichEdit controls placed on the
form.  When the form becomes visible, the first Rich Edit
control will display the rich text in its raw form, including
formatting characters. The second will show the rich text in
its intended format.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
// you may need to change this path to suit your environment
  char const *Path = "..\\OVERVIEW.RTF";
  RichEdit1->PlainText = true;
  RichEdit1->Lines->LoadFromFile(Path);
  RichEdit2->PlainText = false;
  RichEdit2->Lines->LoadFromFile(Path);
}
```

Delphi Examples:

```
{
```

```
This example requires two TRichEdit controls placed on the
form.  When the form becomes visible, the first Rich Edit
control will display the rich text in its raw form, including
formatting characters. The second will show the rich text in
its intended format.
}

procedure TForm1.FormCreate(Sender: TObject);
const
  // you may need to change this path to suit your environment
  Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := True;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
  RichEdit2.PlainText := False;
  RichEdit2.Lines.LoadFromFile(Path);
  RichEdit2.ScrollBars := ssVertical;
end;
```

See Also

Add (see page 3033)

LoadFromStream (see page 3062)

SaveToFile (see page 3065)

Strings (see page 3080)

Text (see page 3084)

23.1.49.3.29 TStrings.LoadFromStream Method

Fills the list with lines of text read from a stream.

Pascal

```
procedure LoadFromStream(Stream: TStream); virtual;
```

C++

```
virtual __fastcall LoadFromStream(TStream Stream);
```

Description

Call LoadFromStream to fill the list of the TStrings (see page 3029) object from the stream specified by Stream. The text read from the stream is parsed into strings separated by carriage return or linefeed characters. Thus, LoadFromStream reads the value of the Text (see page 3084) property.

If the stream is a file stream, LoadFromStream does the same thing as LoadFromFile (see page 3061), except the application must create and destroy the file stream.

C++ Examples:

```
/*
This example requires TListBox, TMemo, TRichEdit, and
TButton controls placed on the form. The list box should
contain one or more items.  When the form becomes visible,
click on the button and the contents of the list box will be
transferred to a stream and then to the rich edit control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemoryStream* pms = new TMemoryStream();
  ListBox1->Items->SaveToStream(pms); // write list box contents to the
                                      // stream.
```

```
  pms->Position = 0;                  // reset to the beginning of the stream.
  RichEdit1->Lines->LoadFromStream(pms); // load stream contents into rich
                                         // edit control.
  delete pms;
}
```

Delphi Examples:

```
{
This example requires TListBox, TMemo, TRichEdit, and
TButton controls placed on the form. The list box should
contain one or more items.  When the form becomes visible,
click on the button and the contents of the list box will be
transferred to a stream and then to the rich edit control.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  TempStream : TMemoryStream;
begin
  TempStream := TMemoryStream.Create;
  ListBox1.Items.SaveToStream(TempStream);  // write list box contents to the
                                            // stream
  TempStream.Position := 0;      // reset to the beginning of the stream
  RichEdit1.Lines.LoadFromStream( TempStream); // load stream contents into rich
                                               // edit control
  TempStream.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Apples');
  ListBox1.Items.Add('Oranges');
  ListBox1.Items.Add('Pears');
end;
```

See Also

Add (see page 3033)

LoadFromFile (see page 3061)

SaveToStream (see page 3067)

Text (see page 3084)

23.1.49.3.30 TStrings.Move Method

Changes the position of a string in the list.

Pascal

```
procedure Move(CurIndex: Integer; NewIndex: Integer); virtual;
```

C++

```
virtual __fastcall Move(int CurIndex, int NewIndex);
```

Description

Use Move to move the string at position CurIndex so that it occupies the position NewIndex. The positions are specified as 0-based indexes. For example, the following lines of code move the string in the first position to the last position.

```
MyStringsObject.Move(0, MyStringsObject.Count
MyStringsObject->Move(0, MyStringsObject->Count - 1);
```

If the string has an associated object, the object remains associated with the string in its new position.

C++ Examples:

```
/*
This example uses a list box and a button on a form. The
list box contains items when the form appears. When the user
clicks the button, the selected item in the list box is
moved to the top of the list box:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->MultiSelect = false;
  Button1->Caption = "Move to Top";
  for (int i = 1; i <= 10; i++)
    ListBox1->Items->Add("Item " + IntToStr(i));
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ListBox1->ItemIndex >= 0 && ListBox1->ItemIndex <= 9)
    ListBox1->Items->Move(ListBox1->ItemIndex, 0);
}
```

Delphi Examples:

```
{
This example uses a list box and a button on a form. The
list box contains items when the form appears. When the user
clicks the button, the selected item in the list box is
moved to the top of the list box:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (ListBox1.ItemIndex in [0..9]) then
    ListBox1.Items.Move(ListBox1.ItemIndex, 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  ListBox1.MultiSelect := False;
  Button1.Caption := 'Move to Top';
  for I := 1 to 10 do
    ListBox1.Items.Add('Item ' + IntToStr(I));
end;
```

See Also

Count (see page 3071)

Objects (see page 3077)

Strings (see page 3080)

23.1.49.3.31 TStrings.Put Method

Changes the value of the string with a specified index.

Pascal

```
procedure Put(Index: Integer; const S: string); virtual;
```

C++

```
virtual __fastcall Put(int Index, const AnsiString S);
```

Description

Put is the protected write implementation of the Strings (see page 3080) property.

Put changes the value of the string with the index specified by Index to S. Put does not change the object at the specified position. That is, any object associated with the previous string becomes associated with the new string.

See Also

Strings (see page 3080)

Objects (see page 3077)

23.1.49.3.32 TStrings.PutObject Method

Changes the object associated with the string at a specified index.

Pascal

```
procedure PutObject(Index: Integer; AObject: TObject); virtual;
```

C++

```
virtual __fastcall PutObject(int Index, TObject * AObject);
```

Description

PutObject is the protected write implementation of the Objects (see page 3077) property.

As implemented in TStrings (see page 3029), PutObject does nothing. This provides a default implementation for descendants that do not support associating objects with the strings in the list. Descendants that support this feature override PutObject to change the specified object.

See Also

Objects (see page 3077)

23.1.49.3.33 TStrings.SaveToFile Method

Saves the strings in the list to the specified file.

Pascal

```
procedure SaveToFile(const FileName: string); virtual;
```

C++

```
virtual __fastcall SaveToFile(const AnsiString FileName);
```

Description

Call SaveToFile to save the strings in the list to the file specified by FileName. Each string in the list is written to a separate line in the file.

Note: In Linux, SaveToFile saves the strings as they appear in the list. If the last string does not have an end-of-line marker, the resulting file may be considered incomplete by some editors.

C++ Examples:

```
/*
This example displays a print dialog box with a Print to
File check box, and then prints the contents of a rich edit
control to the indicated destination:
Verify that the file is saved by the SaveDialog.
*/
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  PrintDialog1->Options << poPrintToFile;
```

```
    PrintDialog1->PrintToFile = true;
    if (PrintDialog1->Execute())
    {
      if (PrintDialog1->PrintToFile)
      {
        SaveDialog1->Title = "Print to File:";
        if (SaveDialog1->Execute())
          RichEdit1->Lines->SaveToFile(SaveDialog1->FileName);
        else
          RichEdit1->Print("");
      }
    }
  }

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
  // you may need to change this path to suit your environment
    char const *Path = "..\\overview.rtf";
    RichEdit1->PlainText = false;
    RichEdit1->Lines->LoadFromFile(Path);
    RichEdit1->ScrollBars = ssVertical;
  }
```

Delphi Examples:

```
  {
  This example displays a print dialog box with a Print to
  File check box, and then prints the contents of a rich edit
  control to the indicated destination:
  Verify that the file is saved by the SaveDialog.
  }
  procedure TForm1.BitBtn1Click(Sender: TObject);
  begin
  with PrintDialog1 do
    begin
    Options := [poPrintToFile];
    PrintToFile := True;
    if Execute then
      begin
      if PrintToFile then
        begin
        SaveDialog1.Title := 'Print to File: ';
        if SaveDialog1.Execute then
          RichEdit1.Lines.SaveToFile(SaveDialog1.FileName);
        end
      else
        RichEdit1.Print('');
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const
    // you may need to change this path to suit your environment
    Path = 'OverView.RTF';
  begin
    RichEdit1.PlainText := False;
    RichEdit1.Lines.LoadFromFile(Path);
    RichEdit1.ScrollBars := ssVertical;
  end;
```

See Also

LoadFromFile (see page 3061)

SaveToStream (see page 3067)

Strings (see page 3080)

Text (see page 3084)

23.1.49.3.34 TStrings.SaveToStream Method

Writes the value of the Text (see page 3084) property to a stream object.

Pascal

```
procedure SaveToStream(Stream: TStream); virtual;
```

C++

```
virtual __fastcall SaveToStream(TStream Stream);
```

Description

Call SaveToStream to save the strings in the list to the stream specified by the Stream parameter. SaveToStream writes the strings delimited by carriage return, line feed pairs. If the stream is a file stream, SaveToStream does the same thing as SaveToFile (see page 3065), except the application must create and destroy the file stream.

C++ Examples:

```
/*
This example requires TListBox, TMemo, TRichEdit, and
TButton controls placed on the form. The list box should
contain one or more items.  When the form becomes visible,
click on the button and the contents of the list box will be
transferred to a stream and then to the rich edit control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemoryStream* pms = new TMemoryStream();
  ListBox1->Items->SaveToStream(pms); // write list box contents to the
                                      // stream.
  pms->Position = 0;                  // reset to the beginning of the stream.
  RichEdit1->Lines->LoadFromStream(pms); // load stream contents into rich
                                         // edit control.
  delete pms;
}
```

Delphi Examples:

```
{
This example requires TListBox, TMemo, TRichEdit, and
TButton controls placed on the form. The list box should
contain one or more items.  When the form becomes visible,
click on the button and the contents of the list box will be
transferred to a stream and then to the rich edit control.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  TempStream : TMemoryStream;
begin
  TempStream := TMemoryStream.Create;
  ListBox1.Items.SaveToStream(TempStream);  // write list box contents to the
                                            // stream
  TempStream.Position := 0;      // reset to the beginning of the stream
  RichEdit1.Lines.LoadFromStream( TempStream); // load stream contents into rich
                                               // edit control
  TempStream.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
  ListBox1.Items.Add('Apples');
  ListBox1.Items.Add('Oranges');
  ListBox1.Items.Add('Pears');
end;
```

See Also

LoadFromStream (see page 3062)

SaveToFile (see page 3065)

Strings (see page 3080)

Text (see page 3084)

23.1.49.3.35 TStrings.SetCapacity Method

Changes the amount of memory allocated to hold strings in the list.

Pascal

```
procedure SetCapacity(NewCapacity: Integer); virtual;
```

C++

```
virtual __fastcall SetCapacity(int NewCapacity);
```

Description

SetCapacity is the protected read implementation of the Capacity (see page 3070) property.

Integer is the number of strings the list can hold after the capacity has changed.

In TStrings (see page 3029), SetCapacity does nothing. Descendant classes override this method to change the number of strings that the list can hold. For descendant classes that implement SetCapacity, assigning a value smaller than Count (see page 3071) removes strings from the end of the list. Assigning a value greater than Count (see page 3071) allocates space for more strings to be added.

See Also

Capacity (see page 3070)

Count (see page 3071)

23.1.49.3.36 TStrings.SetText Method

Sets the Text (see page 3084) property.

Pascal

```
procedure SetText(Text: PChar); virtual;
```

C++

```
virtual __fastcall SetText(const char * Text);
```

Description

Call SetText to replace the list with the strings specified by the Text (see page 3084) parameter. SetText adds strings one at a time to the list, using the carriage returns or linefeed characters in Text (see page 3084) as delimiters indicating when to add a new string.

See Also

Add (see page 3033)

Text (see page 3084)

23.1.49.3.37 TStrings.SetTextStr Method

Sets the Text (see page 3084) property.

Pascal

```
procedure SetTextStr(const Value: string); virtual;
```

C++

```
virtual __fastcall SetTextStr(const AnsiString Value);
```

Description

GetTextStr (see page 3056) is the protected write implementation of the Text (see page 3084) property. It replaces the list with the strings specified by the Value parameter. SetTextStr adds strings one at a time to the list, using the carriage returns or linefeed characters in Value as delimiters indicating when to add a new string.

See Also

SetText (see page 3068)

Text (see page 3084)

23.1.49.3.38 TStrings.SetUpdateState Method

Performs internal adjustments before or after a series of updates.

Pascal

```
procedure SetUpdateState(Updating: Boolean); virtual;
```

C++

```
virtual __fastcall SetUpdateState(Boolean Updating);
```

Description

SetUpdateState is called at the beginning or end of a series of updates. When the BeginUpdate (see page 3045) method is first called and the TStrings (see page 3029) object is not already in the middle of an update, TStrings (see page 3029) calls SetUpdateState internally, with Updating set to true. When the EndUpdate (see page 3050) method is called and it cancels out the last unmatched call to BeginUpdate (see page 3045), TStrings (see page 3029) calls SetUpdateState internally, with Updating set to false.

As implemented in TStrings (see page 3029), SetUpdateState does nothing. Descendant classes can override this method to optimize the response to updates.

See Also

BeginUpdate (see page 3045)

EndUpdate (see page 3050)

23.1.49.4 TStrings Properties

The properties of the TStrings class are listed here.

Protected Properties

	Name	Description
R	UpdateCount (see page 3085)	Indicates the number of calls to BeginUpdate (see page 3045) that have not been matched by a call to EndUpdate (see page 3050).

Public Properties

	Name	Description
	Capacity (see page 3070)	Indicates the number of strings the TStrings (see page 3029) object can hold.
	CommaText (see page 3070)	Lists the strings in the TStrings (see page 3029) object in system data format (SDF).
R	Count (see page 3071)	Introduces an abstract property to represent the number of strings in the list.
	DelimitedText (see page 3074)	Represents all the strings in the TStrings (see page 3029) object as a single delimited string.
	Delimiter (see page 3074)	Specifies the delimiter used by the DelimitedText (see page 3074) property.
	LineBreak (see page 3075)	Defines the line break character(s).
R	Names (see page 3075)	Indicates the name part of strings that are name-value pairs.
	NameValueSeparator (see page 3077)	Indicates the character used to separate names from values.
	Objects (see page 3077)	Represents a set of objects that are associated one with each of the strings in the Strings (see page 3080) property.
	QuoteChar (see page 3080)	Specifies the quote character used by the DelimitedText (see page 3074) property.
	StrictDelimiter (see page 3080)	Determines how the Delimiter (see page 3074) property is used.
	Strings (see page 3080)	References the strings in the list by their positions.
	StringsAdapter (see page 3084)	Implements an IStringsAdapter (see page 3208) interface for the TStrings (see page 3029) object.
	Text (see page 3084)	Lists the strings in the TStrings (see page 3029) object as a single string with the individual strings delimited by carriage returns and line feeds.
	ValueFromIndex (see page 3085)	Represents the value part of a string with a given index, on strings that are name-value pairs.
	Values (see page 3086)	Represents the value part of a string associated with a given name, on strings that are name-value pairs.

23.1.49.4.1 TStrings.Capacity Property

Indicates the number of strings the TStrings (see page 3029) object can hold.

Pascal

```
property Capacity: Integer;
```

C++

```
__property int Capacity;
```

Description

Read Capacity to determine the currently allocated size of the string list. For the TStrings (see page 3029) object, reading Capacity returns the Count (see page 3071) property, and setting Capacity does nothing. Descendants of TStrings (see page 3029) can override this property to allow a string list to allocate memory for entries that have not been added to the list.

See Also

Count (see page 3071)

23.1.49.4.2 TStrings.CommaText Property

Lists the strings in the TStrings (see page 3029) object in system data format (SDF).

Pascal

```
property CommaText: string;
```

C++

```
__property AnsiString CommaText;
```

Description

Use CommaText to get or set all the strings in the TStrings (see page 3029) object in a single comma-delimited string.

When retrieving CommaText, any string in the list that include spaces, commas or quotes will be contained in double quotes, and any double quotes in a string will be repeated. For example, if the list contains the following strings:

```
Stri,ng 1
Stri"ng 2
String 3
String4
```

CommaText will return:

```
"Stri,ng 1","Stri""ng 2","String 3",String4
```

When assigning CommaText, the value is parsed as SDF formatted text. For SDF format, strings are separated by commas or spaces, and optionally enclosed in double quotes. Double quote marks that are part of the string are repeated to distinguish them from the quotes that surround the string. Spaces and commas that are not contained within double quote marks are delimiters. Two commas next to each other will indicate an empty string, but spaces that appear next to another delimiter are ignored. For example, suppose CommaText is set to:

```
"Stri,ng 1", "Stri""ng 2" , String 3,String4
```

The list will then contain:

```
Stri,ng 1
Stri"ng 2
String
3
String4
```

Note: CommaText is the same as the DelimitedText (see page 3074) property with a delimiter of ',' and a quote character of "".

Including a trailing comma in the source string causes a blank item to be included in the string list. For example, if CommaText is set to

```
"String1, String 2, String 3,"
```

the string list will contain

```
String1
String2
String3
<Blank>
```

See Also

Add (see page 3033)

Strings (see page 3080)

Text (see page 3084)

DelimitedText (see page 3074)

23.1.49.4.3 TStrings.Count Property

Introduces an abstract property to represent the number of strings in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Descendants of TStrings (see page 3029) implement a Count property to indicate the number of strings in the list.

Use the Count property when iterating over all the strings in the list, or when trying to locate the position of a string relative to the last string in the list.

C++ Examples:

```
/*
The following example updates the strings in a list box
given the strings contained in another list box. If a string
in the source list box has the form Name=Value and the
destination list box contains a string with the same Name
part, the Value part in the destination list box will be
replaced by the source's value.  TListBox objects that are
not name-value pairs are not assigned a Name and IndexOfName
will return -1.  TStrings Values are referenced using the
name AnsiString as an index.
*/
void MergeStrings(TStrings *Dest, TStrings *Source)
{
  for (int i = 0; i < Source->Count; i++)
  {
    if (Source->Strings[i].Pos("=") > 1)
    {
      int DI = Dest->IndexOfName(Source->Names[i]);
      if (DI > -1)
        Dest->Values[Source->Names[i]] = Source->Values[Source->Names[i]];
    }
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  MergeStrings(ListBox1->Items,ListBox2->Items);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Plants = 10");
  ListBox1->Items->Add("Animals = 20");
  ListBox1->Items->Add("Minerals = 15");
  ListBox2->Items->Add("Animals = 4");
  ListBox2->Items->Add("Plants = 3");
  ListBox2->Items->Add("Minerals = 78");
}
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
```

```
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
The following example updates the strings in a list box
given the strings contained in another list box. If a string
in the source list box has the form Name=Value and the
destination list box contains a string with the same Name
part, the Value part in the destination list box will be
replaced by the source's value.  TListBox objects that are
not name-value pairs are not assigned a Name and IndexOfName
will return -1.
}
procedure MergeStrings(Dest, Source: TStrings);
var
  I, DI: Integer;
  begin
  for I := 0 to Source.Count - 1 do
  begin
    if Pos ('=', Source[I]) > 1 then
    begin
      DI := Dest.IndexOfName(Source.Names[I]);
      if DI > -1 then Dest[DI] := Source[I];
    end;
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  MergeStrings(ListBox1.Items, ListBox2.Items);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Plants = 10');
  ListBox1.Items.Add('Animals = 20');
  ListBox1.Items.Add('Minerals = 15');
  ListBox2.Items.Add('Animals = 4');
  ListBox2.Items.Add('Plants = 3');
  ListBox2.Items.Add('Minerals = 78');
end;
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

IndexOf (see page 3056)

Strings (see page 3080)

23.1.49.4.4 TStrings.DelimitedText Property

Represents all the strings in the TStrings (see page 3029) object as a single delimited string.

Pascal

```
property DelimitedText: string;
```

C++

```
__property AnsiString DelimitedText;
```

Description

Use DelimitedText to get or set all the strings in the TStrings (see page 3029) object in a single string.

When retrieving DelimitedText, the resulting value delimits individual strings in two ways: each string is surrounded (before and after) by the quote character specified by the QuoteChar (see page 3080) property. In addition, individual strings are separated by the character specified by the Delimiter (see page 3074) property.

When assigning DelimitedText, individual strings must be separated using QuoteChar (see page 3080) at both ends, using Delimiter (see page 3074) as a separator, or using both these methods.

A space is always interpreted as a delimiter, regardless of the value of Delimiter (see page 3074), unless the space occurs between quote characters.

Note: CommaText (see page 3070) is the same as the DelimitedText property when Delimiter (see page 3074) is ',' and QuoteChar (see page 3080) is "".

See Also

CommaText (see page 3070)

Delimiter (see page 3074)

Text (see page 3084)

QuoteChar (see page 3080)

23.1.49.4.5 TStrings.Delimiter Property

Specifies the delimiter used by the DelimitedText (see page 3074) property.

Pascal

```
property Delimiter: Char;
```

C++

```
__property Char Delimiter;
```

Description

Use Delimiter to get or set the delimiter used by the DelimitedText (see page 3074) property. DelimitedText (see page 3074) represents all of the strings in the TStrings (see page 3029) object as a single string, with individual strings separated by the character that is the value of Delimiter.

See Also

DelimitedText (see page 3074)

QuoteChar (see page 3080)

23.1.49.4.6 TStrings.LineBreak Property

Defines the line break character(s).

Pascal

```
property LineBreak: string;
```

C++

```
__property AnsiString LineBreak;
```

Description

LineBreak can be read and set. It is used internally in TStrings (see page 3029) to terminate strings in multi-string output operations.

For example, the GetText (see page 3055) method returns a long string containing all TStrings (see page 3029) strings, each of which is terminated by the LineBreak value.

Note: In Windows, the default LineBreak value is a carriage return and line feed combination (#13#10), whereas on Linux, it is just a carriage return (#13).

See Also

GetText (see page 3055)

23.1.49.4.7 TStrings.Names Property

Indicates the name part of strings that are name-value pairs.

Pascal

```
property Names [Index: Integer]: string;
```

C++

```
__property AnsiString Names[int Index];
```

Description

When the list of strings for the TStrings (see page 3029) object includes strings that are name-value pairs, read Names to access the name part of a string. Names is the name part of the string at Index, where 0 is the first string, 1 is the second string, and so on. If the string is not a name-value pair, Names contains an empty string.

C++ Examples:

```
/*
The following example updates the strings in a list box
given the strings contained in another list box. If a string
in the source list box has the form Name=Value and the
destination list box contains a string with the same Name
part, the Value part in the destination list box will be
replaced by the source's value.  TListBox objects that are
not name-value pairs are not assigned a Name and IndexOfName
will return -1.  TStrings Values are referenced using the
name AnsiString as an index.
*/
void MergeStrings(TStrings *Dest, TStrings *Source)
```

```
{
  for (int i = 0; i < Source->Count; i++)
  {
    if (Source->Strings[i].Pos("=") > 1)
    {
      int DI = Dest->IndexOfName(Source->Names[i]);
      if (DI > -1)
        Dest->Values[Source->Names[i]] = Source->Values[Source->Names[i]];
    }
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  MergeStrings(ListBox1->Items,ListBox2->Items);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Plants = 10");
  ListBox1->Items->Add("Animals = 20");
  ListBox1->Items->Add("Minerals = 15");
  ListBox2->Items->Add("Animals = 4");
  ListBox2->Items->Add("Plants = 3");
  ListBox2->Items->Add("Minerals = 78");
}
```

Delphi Examples:

```
{
The following example updates the strings in a list box
given the strings contained in another list box. If a string
in the source list box has the form Name=Value and the
destination list box contains a string with the same Name
part, the Value part in the destination list box will be
replaced by the source's value.  TListBox objects that are
not name-value pairs are not assigned a Name and IndexOfName
will return -1.
}
procedure MergeStrings(Dest, Source: TStrings);
var
  I, DI: Integer;
  begin
  for I := 0 to Source.Count - 1 do
  begin
    if Pos ('=', Source[I]) > 1 then
    begin
      DI := Dest.IndexOfName(Source.Names[I]);
      if DI > -1 then Dest[DI] := Source[I];
    end;
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  MergeStrings(ListBox1.Items, ListBox2.Items);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Plants = 10');
  ListBox1.Items.Add('Animals = 20');
  ListBox1.Items.Add('Minerals = 15');
  ListBox2.Items.Add('Animals = 4');
  ListBox2.Items.Add('Plants = 3');
  ListBox2.Items.Add('Minerals = 78');
end;
```

See Also

NameValueSeparator (see page 3077)

IndexOfName (see page 3057)

Strings (see page 3080)

Values (see page 3086)

23.1.49.4.8 TStrings.NameValueSeparator Property

Indicates the character used to separate names from values.

Pascal

```
property NameValueSeparator: Char;
```

C++

```
__property Char NameValueSeparator;
```

Description

Strings (see page 3080) that contain the NameValueSeparator character are considered name-value pairs. NameValueSeparator defaults to the equal sign (=). TStrings (see page 3029) defines various methods for accessing names and values and for searching for specific names.

Strings (see page 3080) that are name-value pairs consist of a name part, the separator character, and a value part. Any spaces around the separator character are part of the name or value. This convention corresponds to the format used in many initialization files. For example:

```
DisplayGrid=1
SnapToGrid=1
GridSizeX=8
GridSizeY=8
```

See Also

ValueFromIndex (see page 3085)

Names (see page 3075)

IndexOfName (see page 3057)

Strings (see page 3080)

Values (see page 3086)

23.1.49.4.9 TStrings.Objects Property

Represents a set of objects that are associated one with each of the strings in the Strings (see page 3080) property.

Pascal

```
property Objects [Index: Integer]: TObject;
```

C++

```
__property TObject * Objects[int Index];
```

Description

Setting the Objects property for TStrings (see page 3029) does nothing. Reading the Objects property for TStrings (see page 3029) returns nil (Delphi) or NULL (C++). Descendant classes can associate objects with the strings in the set by

implementing the Objects property.

Use the Objects property of a descendant of TStrings (see page 3029) to get or set the object associated with the string at the position indicated by Index. Index gives the position of the string associated with the object, where 0 is the first string, 1 is the second string, and so on. If a descendant of TStrings (see page 3029) does not support the Objects property, reading this property returns nil (Delphi) or NULL (C++)

Note: The TStrings (see page 3029) object does not own the objects in the Objects array. Objects added to the Objects array still exist even if the TStrings (see page 3029) object is destroyed. They must be explicitly destroyed by the application.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", (TObject *)vsIcon);
  ComboBox1->Items->AddObject("vsList", (TObject *)vsList);
  ComboBox1->Items->AddObject("vsReport", (TObject *)vsReport);
  ComboBox1->Items->AddObject("vsSmallIcon", (TObject *)vsSmallIcon);
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
```

```
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := 'Image' + IntToStr(I);
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
    NewColumn.Width := 200;
    // Add View styles and constants to the Combo Box
    ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
    ComboBox1.Items.AddObject('vsList', TObject(vsList));
    ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
    ComboBox1.Items.AddObject(
      'vsSmallIcon', TObject(vsSmallIcon));
    // Display first item in the Combo Box
    ComboBox1.ItemIndex := 0;
  end;
end;

procedure TComboForm.ComboBox1Click(Sender: TObject);
begin
  with ComboBox1 do
    ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

end;
```

See Also

AddObject (see page 3036)

IndexOfObject (see page 3059)

InsertObject (see page 3060)

Strings (see page 3080)

23.1.49.4.10 TStrings.QuoteChar Property

Specifies the quote character used by the DelimitedText (see page 3074) property.

Pascal

```
property QuoteChar: Char;
```

C++

```
__property Char QuoteChar;
```

Description

Use QuoteChar to get or set the quote character that is used to enclose individual strings in the DelimitedText (see page 3074) property.

See Also

Delimiter (see page 3074)

DelimitedText (see page 3074)

CommaText (see page 3070)

23.1.49.4.11 TStrings.StrictDelimiter Property

Determines how the Delimiter (see page 3074) property is used.

Pascal

```
property StrictDelimiter: Boolean;
```

C++

```
__property Boolean StrictDelimiter;
```

Description

Use this property to specify whether the Delimiter (see page 3074) is the only value used within the DelimitedText (see page 3074) property. If set to True, individual strings in DelimitedText (see page 3074) are separated only by the character that is the value of Delimiter (see page 3074). If set to False, individual strings in DelimitedText (see page 3074) can be separated by a space, a non-printable character, or the character that is the value of Delimiter (see page 3074).

23.1.49.4.12 TStrings.Strings Property

References the strings in the list by their positions.

Pascal

```
property Strings [Index: Integer]: string;
```

C++

```
__property AnsiString Strings[int Index];
```

Description

Descendants of TStrings (see page 3029) must implement an accessor function for the Strings property to return the string at the position indicated by Index. Index gives the position of the string, where 0 is the first string, 1 is the second string, and so on.

Use the Strings property to get or set the string at a particular position.

To store a string as a name-value pair, assign Strings a value that includes the NameValueSeparator (see page 3077) character. The name and value will then be accessible separately using the Names (see page 3075) and Values (see page 3086) properties.

Note: In Delphi, Strings is the default property of TStrings (see page 3029) objects. The Strings identifier can be omitted when accessing the Strings property of a descendant of TStrings (see page 3029). For example, the following two lines of code are both acceptable and do the same thing:

MyStrings.Strings[0] := 'This is the first string';

MyStrings[0] := 'This is the first string';

C++ Examples:

```
/*
This example uses an Open dialog box, a memo, and a button
on a form. When the user clicks the button, the Open dialog
box appears. When the user selects files in the dialog box
and chooses the OK button, the first line from each of the
files is added to the memo. Choose multiple files using the
CNTL key or the SHIFT key.
*/

#include <stdio.h>     // for FILE

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FILE *stream;
  char FirstLine[512];

  OpenDialog1->Options.Clear();
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist;
  OpenDialog1->Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
  OpenDialog1->FilterIndex = 2; // start the dialog showing all files
  if (OpenDialog1->Execute())
  {
    for (int I = 0; I < OpenDialog1->Files->Count; I ++)
    {
      stream = fopen(OpenDialog1->Files->Strings[I].c_str(), "r");
      if (stream)
      {
        // read the first line from the file
        fgets(FirstLine, sizeof(FirstLine), stream);
        Memo1->Lines->Append(FirstLine);
        fclose(stream);
      }
    }
  }
}
/*
The following example updates the strings in a list box
given the strings contained in another list box. If a string
in the source list box has the form Name=Value and the
destination list box contains a string with the same Name
part, the Value part in the destination list box will be
replaced by the source's value.  TListBox objects that are
not name-value pairs are not assigned a Name and IndexOfName
will return -1.  TStrings Values are referenced using the
```

```
name AnsiString as an index.
*/
void MergeStrings(TStrings *Dest, TStrings *Source)
{
  for (int i = 0; i < Source->Count; i++)
  {
    if (Source->Strings[i].Pos("=") > 1)
    {
      int DI = Dest->IndexOfName(Source->Names[i]);
      if (DI > -1)
        Dest->Values[Source->Names[i]] = Source->Values[Source->Names[i]];
    }
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  MergeStrings(ListBox1->Items,ListBox2->Items);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Plants = 10");
  ListBox1->Items->Add("Animals = 20");
  ListBox1->Items->Add("Minerals = 15");
  ListBox2->Items->Add("Animals = 4");
  ListBox2->Items->Add("Plants = 3");
  ListBox2->Items->Add("Minerals = 78");
}
/*
This example uses a button and a memo control on a form.
When the user clicks the button, the content of the specified
file is loaded into the memo, and the fifth line of the file
is written across the top of the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Memo1->Lines->LoadFromFile("../readme.txt");
  Caption = Memo1->Lines->Strings[4];
}
```

Delphi Examples:

```
{
This example uses an Open dialog box, a memo, and a button
on a form. When the user clicks the button, the Open dialog
box appears. When the user selects files in the dialog box
and chooses the OK button, the first line from each of the
files is added to the memo. Choose multiple files using the
CNTL key or the SHIFT key.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: integer;
  F: TextFile;
  FirstLine: string;
begin
  OpenDialog1.Options := [ofAllowMultiSelect, ofFileMustExist];
  OpenDialog1.Filter := 'Text files (*.txt)|*.txt|All files (*.*)|*.*';
  OpenDialog1.FilterIndex := 2; { start the dialog showing all files }
  if OpenDialog1.Execute then
    with OpenDialog1.Files do
      for I := 0 to Count - 1 do
      begin
        AssignFile(F, Strings[I]);  { next file in Files property }
        Reset(F);
        Readln(F, FirstLine);  { Read the first line out of the file }
```

```
        Memo1.Lines.Append(FirstLine);  { Add the line to the memo }
        CloseFile(F);
      end;
end;
{
The following example updates the strings in a list box
given the strings contained in another list box. If a string
in the source list box has the form Name=Value and the
destination list box contains a string with the same Name
part, the Value part in the destination list box will be
replaced by the source's value.  TListBox objects that are
not name-value pairs are not assigned a Name and IndexOfName
will return -1.
}
procedure MergeStrings(Dest, Source: TStrings);
var
  I, DI: Integer;
  begin
  for I := 0 to Source.Count - 1 do
  begin
    if Pos ('=', Source[I]) > 1 then
    begin
      DI := Dest.IndexOfName(Source.Names[I]);
      if DI > -1 then Dest[DI] := Source[I];
    end;
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  MergeStrings(ListBox1.Items, ListBox2.Items);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Plants = 10');
  ListBox1.Items.Add('Animals = 20');
  ListBox1.Items.Add('Minerals = 15');
  ListBox2.Items.Add('Animals = 4');
  ListBox2.Items.Add('Plants = 3');
  ListBox2.Items.Add('Minerals = 78');
end;
```

See Also

ValueFromIndex (see page 3085)

NameValueSeparator (see page 3077)

Names (see page 3075)

Values (see page 3086)

CommaText (see page 3070)

IndexOf (see page 3056)

Names (see page 3075)

Objects (see page 3077)

Text (see page 3084)

Values (see page 3086)

operator_sb

23.1.49.4.13 TStrings.StringsAdapter Property

Implements an IStringsAdapter (see page 3208) interface for the TStrings (see page 3029) object.

Pascal

```
property StringsAdapter: IStringsAdapter;
```

C++

```
__property IStringsAdapter StringsAdapter;
```

Description

Do not use the StringsAdapter property. StringsAdapter is used internally.

See Also

TStringsAdapter (see page 1696)

23.1.49.4.14 TStrings.Text Property

Lists the strings in the TStrings (see page 3029) object as a single string with the individual strings delimited by carriage returns and line feeds.

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

Use Text to get or set all the strings in the TStrings (see page 3029) object in a single string delimited by carriage return, line feed pairs.

When reading Text, the strings in the list will be separated by carriage return and (on Windows) line feed. If any of the strings in the list contain a carriage return (and line feed), the resulting value of Text will appear to contain more strings than is indicated by the Count (see page 3071) property.

When setting Text, the value will be parsed by separating into substrings whenever a carriage return or linefeed is encountered. (The two do not need to form pairs).

If the strings in the list contain carriage return or linefeed characters, a less ambiguous format for the strings is available through the CommaText (see page 3070) or DelimitedText (see page 3074) property.

See Also

Add (see page 3033)

CommaText (see page 3070)

Count (see page 3071)

Strings (see page 3080)

GetText (see page 3055)

SetText (see page 3068)

DelimitedText (see page 3074)

23.1.49.4.15 TStrings.UpdateCount Property

Indicates the number of calls to BeginUpdate (see page 3045) that have not been matched by a call to EndUpdate (see page 3050).

Pascal

```
property UpdateCount: Integer;
```

C++

```
__property int UpdateCount;
```

Description

TStrings (see page 3029) uses UpdateCount to keep track of calls to the BeginUpdate (see page 3045) and EndUpdate (see page 3050) methods. Every time a call is made to BeginUpdate (see page 3045), TStrings (see page 3029) increments the value of UpdateCount. Every call to EndUpdate (see page 3050) causes TStrings (see page 3029) to decrement UpdateCount.

When UpdateCount changes from 0 to 1, TStrings (see page 3029) calls the SetUpdateState (see page 3069) method with a parameter of true. When UpdateCount changes from 1 to 0, TStrings (see page 3029) calls the SetUpdateState (see page 3069) method with a parameter of false. This allows descendant classes to perform optimizations when handling multiple updates.

See Also

BeginUpdate (see page 3045)

EndUpdate (see page 3050)

SetUpdateState (see page 3069)

23.1.49.4.16 TStrings.ValueFromIndex Property

Represents the value part of a string with a given index, on strings that are name-value pairs.

Pascal

```
property ValueFromIndex [Index: Integer]: string;
```

C++

```
__property AnsiString ValueFromIndex[int Index];
```

Description

When the list of strings for the TStrings (see page 3029) object includes strings that are name-value pairs, use ValueFromIndex to get or set the value part of a string associated with an index.

For more information on name-value pairs, refer to the NameValueSeparator (see page 3077) property.

See Also

NameValueSeparator (see page 3077)

IndexOfName (see page 3057)

Strings (see page 3080)

Values (see page 3086)

23.1.49.4.17 TStrings.Values Property

Represents the value part of a string associated with a given name, on strings that are name-value pairs.

Pascal

```
property Values [const Name: string]: string;
```

C++

```
__property AnsiString Values[AnsiString const Name];
```

Description

When the list of strings for the TStrings (see page 3029) object includes strings that are name-value pairs, use Values to get or set the value part of a string associated with a specific name part.

For more information on name-value pairs, refer to the NameValueSeparator (see page 3077) property.

Note: The Name index is case-insensitive. That is, Values is the value part for the first occurrence of Name or an equivalent string that differs only in case.

See Also

ValueFromIndex (see page 3085)

NameValueSeparator (see page 3077)

IndexOfName (see page 3057)

Names (see page 3075)

Strings (see page 3080)

23.1.50 TStringsEnumerator Class

Enumerates through the strings of a TStrings (see page 3029) object.

Class Hierarchy

Pascal

```
TStringsEnumerator = class;
```

C++

```
class TStringsEnumerator;
```

File

Classes

Description

TStringsEnumerator provides an enumerator for TStrings (see page 3029) objects.

23.1.50.1 TStringsEnumerator Members

The following tables list the members exposed by TStringsEnumerator.

Public Methods

	Name	Description
	Create (see page 3087)	Creates a TStrings (see page 3029) enumerator.
	GetCurrent (see page 3087)	Gets the current string.
	MoveNext (see page 3087)	Moves the enumerator to the next string in the TStrings (see page 3029) collection.

Public Properties

	Name	Description
R	Current (see page 3088)	The current string.

23.1.50.2 TStringsEnumerator.Create Constructor

Creates a TStrings (see page 3029) enumerator.

Pascal

```
constructor Create(AStrings: TStrings);
```

C++

```
__fastcall TStringsEnumerator(TStrings AStrings);
```

Description

Creates an instance of TStringsEnumerator (see page 3086).

23.1.50.3 TStringsEnumerator Methods

The methods of the TStringsEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 3087)	Gets the current string.
	MoveNext (see page 3087)	Moves the enumerator to the next string in the TStrings (see page 3029) collection.

23.1.50.3.1 TStringsEnumerator.GetCurrent Method

Gets the current string.

Pascal

```
function GetCurrent: string;
```

C++

```
__fastcall AnsiString GetCurrent();
```

Description

Use GetCurrent to get the current string in the TStrings (see page 3029) object being enumerated.

23.1.50.3.2 TStringsEnumerator.MoveNext Method

Moves the enumerator to the next string in the TStrings (see page 3029) collection.

Pascal

```
function MoveNext: Boolean;
```

C++

```
__fastcall Boolean MoveNext();
```

Description

Use MoveNext to move the TStringsEnumerator (see page 3086) to the next string in the TStrings (see page 3029) object being enumerated.

23.1.50.4 TStringsEnumerator Properties

The properties of the TStringsEnumerator class are listed here.

Public Properties

	Name	Description
R	Current (see page 3088)	The current string.

23.1.50.4.1 TStringsEnumerator.Current Property

The current string.

Pascal

```
property Current: string;
```

C++

```
__property AnsiString Current;
```

Description

Current contains the current string in the TStrings (see page 3029) object being enumerated.

23.1.51 TStringStream Class

TStringStream provides file-like access to information stored as a long string.

Class Hierarchy

Pascal

```
TStringStream = class(TStream);
```

C++

```
class TStringStream : public TStream;
```

File

Classes

Description

Use TStringStream to store data as a long string enhanced with I/O capabilities. TStringStream is useful as an intermediary object that can hold text as well as read it from or write it to another storage medium. TStringStream provides a mechanism for

manipulating text that is obtained from a less accessible medium.

C++ Examples:

```
/*
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
*/
class MyScrollBar : public TScrollBar
{
__published:    // IDE-managed Components
private:    // User declarations
public:     // User declarations
    __fastcall MyScrollBar(TComponent* Owner);
};

__fastcall MyScrollBar::MyScrollBar(TComponent* Owner)
    : TScrollBar(Owner)
{
}

void RegisterClassesWithStreamingSystem(void)
{
  // Make sure that as part of the startup
  // code our streaming classes are registered
  // with the streaming system.

  #pragma startup RegisterClassesWithStreamingSystem

  Classes::RegisterClass(__classid(MyScrollBar));
}

MyScrollBar *ScrollBar1;

AnsiString __fastcall ComponentToString(TComponent *c)
{
 AnsiString as;
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);

 try
 {
    pms->WriteComponent(c);
    pms->Seek(0, soFromBeginning);
    ObjectBinaryToText(pms, pss);
    pss->Seek(0, soFromBeginning);
    as = pss->DataString;
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 delete pms;
 delete pss;
 return as;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Memo1->Text = ComponentToString(ScrollBar1);
}

TComponent *__fastcall StringToComponent(AnsiString as)
```

```
{
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);
 TComponent *pc;

 try
 {
    ObjectTextToBinary(pss, pms);
    pms->Seek(0, soFromBeginning);
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 pc = pms->ReadComponent(NULL);
 delete pms;
 delete pss;
 return pc;
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  TComponent *temp = StringToComponent(Memo1->Text);
  ScrollBar1->Free();

  ScrollBar1 = dynamic_cast<MyScrollBar *>(temp);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ScrollBar1 = new MyScrollBar(Form1);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
  ScrollBar1->Top = 48;
  ScrollBar1->Left = 250;
  ScrollBar1->Name = "Ricksbar";
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete ScrollBar1;
}
```

Delphi Examples:

```
{
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
}
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
    MyScrollBar = class(TScrollBar)
    end;

  var
    Form1: TForm1;
    ScrollBar1: MyScrollBar;

  implementation

  {$R *.dfm}

  function ComponentToStringProc(Component: TComponent): string;
  var
    BinStream:TMemoryStream;
    StrStream: TStringStream;
    s: string;
  begin
    BinStream := TMemoryStream.Create;
    try
      StrStream := TStringStream.Create(s);
      try
        BinStream.WriteComponent(Component);
        BinStream.Seek(0, soFromBeginning);
        ObjectBinaryToText(BinStream, StrStream);
        StrStream.Seek(0, soFromBeginning);
        Result:= StrStream.DataString;
      finally
        StrStream.Free;
      end;
    finally
      BinStream.Free
    end;
  end;

  function StringToComponentProc(Value: string): TComponent;
  var
    StrStream:TStringStream;
    BinStream: TMemoryStream;
  begin
    StrStream := TStringStream.Create(Value);
    try
      BinStream := TMemoryStream.Create;
      try
        ObjectTextToBinary(StrStream, BinStream);
        BinStream.Seek(0, soFromBeginning);
        Result:= BinStream.ReadComponent(nil);
      finally
        BinStream.Free;
      end;
    finally
      StrStream.Free;
    end;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Memo1.Text:= ComponentToStringProc(ScrollBar1);
  end;

  // Edit the scrollbar parameters in the memo, click this
  // button and see the scrollbar change.
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    ScrollBar1.Free;
    ScrollBar1:= (StringToComponentProc(Memo1.Text) as MyScrollBar);
    ScrollBar1.Parent:= Form1;
```

```
  ScrollBar1.Visible:= TRUE;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ScrollBar1:= MyScrollBar.Create(Form1);
  ScrollBar1.Parent:= Form1;
  ScrollBar1.Visible:= TRUE;
  ScrollBar1.Top:= 48;
  ScrollBar1.Left:= 250;
  ScrollBar1.Name:= 'Ricksbar';
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  ScrollBar1.Free;
end;

initialization

RegisterClass(MyScrollBar);
```

See Also

TPageProducer (see page 15309)

TMemoryStream (see page 2902)

23.1.51.1 TStringStream Members

The following tables list the members exposed by TStringStream.

Protected Methods

	Name	Description
	GetSize (see page 2982)	This is GetSize, a member of class TStream.
	SetSize (see page 3102)	This is SetSize, a member of class TStringStream.
	SetSize (see page 2989)	This is the overview for the SetSize method overload.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CopyFrom (see page 2979)	Copies a specified number of bytes from one stream to another.
	Create (see page 3093)	Creates an instance of TStringStream (see page 3088).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FixupResourceHeader (see page 2981)	Patches the resource header for a resource that has been written to the stream.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Read (see page 3097)	Reads up to Count bytes from the string stream into Buffer, and advances the current position of the stream by the number of bytes read.
	ReadBuffer (see page 2983)	Reads bytes from the stream into Buffer.
	ReadComponent (see page 2983)	Initiates streaming of components and their properties.
	ReadComponentRes (see page 2987)	Reads components and their properties from a stream in a resource file format.
	ReadResHeader (see page 2987)	Reads a resource-file header from the stream.
	ReadString (see page 3098)	Returns a string that consists of up to Count bytes from the current position in the string stream, and advances the current position of the stream by the number of bytes read.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Seek (see page 3098)	Moves the current position in the stream by Offset bytes, relative to a specified origin.
V	Seek (see page 2988)	This is the overview for the Seek method overload.
V	Write (see page 3103)	Write writes Count bytes from Buffer to the current position in the string stream and updates the current position by Count bytes.
	WriteBuffer (see page 2990)	Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.
	WriteComponent (see page 2990)	Initiates the writing of components and their properties to a stream.
	WriteComponentRes (see page 2994)	Writes components and their properties to a stream from a resource file format.
	WriteDescendent (see page 2994)	Streams components and their properties in inherited forms.
	WriteDescendentRes (see page 2995)	Streams components and their properties in inherited forms.
	WriteResourceHeader (see page 2995)	Writes a resource-file header to the stream.
	WriteString (see page 3103)	Concatenates a specified string to the current position in the string stream, and updates the current position accordingly.

Public Properties

	Name	Description
R	DataString (see page 3104)	Provides direct access to the string that stores the text represented by the TStringStream (see page 3088) object.
	Position (see page 2996)	Indicates the current offset into the stream for reading and writing.
	Size (see page 2997)	Indicates the size in bytes of the stream.

23.1.51.2 TStringStream.Create Constructor

Creates an instance of TStringStream (see page 3088).

Pascal

```
constructor Create(const AString: string);
```

C++

```
__fastcall TStringStream(const AnsiString AString);
```

Description

Call Create to instantiate a TStringStream (see page 3088) object. Create initializes the DataString (see page 3104)

property to the AString parameter.

C++ Examples:

```
/*
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
*/
class MyScrollBar : public TScrollBar
{
__published:    // IDE-managed Components
private:    // User declarations
public:     // User declarations
    __fastcall MyScrollBar(TComponent* Owner);
};

__fastcall MyScrollBar::MyScrollBar(TComponent* Owner)
    : TScrollBar(Owner)
{
}

void RegisterClassesWithStreamingSystem(void)
{
  // Make sure that as part of the startup
  // code our streaming classes are registered
  // with the streaming system.

  #pragma startup RegisterClassesWithStreamingSystem

  Classes::RegisterClass(__classid(MyScrollBar));
}

MyScrollBar *ScrollBar1;

AnsiString __fastcall ComponentToString(TComponent *c)
{
 AnsiString as;
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);

 try
 {
    pms->WriteComponent(c);
    pms->Seek(0, soFromBeginning);
    ObjectBinaryToText(pms, pss);
    pss->Seek(0, soFromBeginning);
    as = pss->DataString;
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 delete pms;
 delete pss;
 return as;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Memo1->Text = ComponentToString(ScrollBar1);
}

TComponent *__fastcall StringToComponent(AnsiString as)
```

```
{
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);
 TComponent *pc;

 try
 {
    ObjectTextToBinary(pss, pms);
    pms->Seek(0, soFromBeginning);
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 pc = pms->ReadComponent(NULL);
 delete pms;
 delete pss;
 return pc;
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  TComponent *temp = StringToComponent(Memo1->Text);
  ScrollBar1->Free();

  ScrollBar1 = dynamic_cast<MyScrollBar *>(temp);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ScrollBar1 = new MyScrollBar(Form1);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
  ScrollBar1->Top = 48;
  ScrollBar1->Left = 250;
  ScrollBar1->Name = "Ricksbar";
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete ScrollBar1;
}
```

Delphi Examples:

```
{
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
}
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
    MyScrollBar = class(TScrollBar)
    end;

  var
    Form1: TForm1;
    ScrollBar1: MyScrollBar;

  implementation

  {$R *.dfm}

  function ComponentToStringProc(Component: TComponent): string;
  var
    BinStream:TMemoryStream;
    StrStream: TStringStream;
    s: string;
  begin
    BinStream := TMemoryStream.Create;
    try
      StrStream := TStringStream.Create(s);
      try
        BinStream.WriteComponent(Component);
        BinStream.Seek(0, soFromBeginning);
        ObjectBinaryToText(BinStream, StrStream);
        StrStream.Seek(0, soFromBeginning);
        Result:= StrStream.DataString;
      finally
        StrStream.Free;
      end;
    finally
      BinStream.Free
    end;
  end;

  function StringToComponentProc(Value: string): TComponent;
  var
    StrStream:TStringStream;
    BinStream: TMemoryStream;
  begin
    StrStream := TStringStream.Create(Value);
    try
      BinStream := TMemoryStream.Create;
      try
        ObjectTextToBinary(StrStream, BinStream);
        BinStream.Seek(0, soFromBeginning);
        Result:= BinStream.ReadComponent(nil);
      finally
        BinStream.Free;
      end;
    finally
      StrStream.Free;
    end;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Memo1.Text:= ComponentToStringProc(ScrollBar1);
  end;

  // Edit the scrollbar parameters in the memo, click this
  // button and see the scrollbar change.
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    ScrollBar1.Free;
    ScrollBar1:= (StringToComponentProc(Memo1.Text) as MyScrollBar);
    ScrollBar1.Parent:= Form1;
```

```
  ScrollBar1.Visible:= TRUE;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ScrollBar1:= MyScrollBar.Create(Form1);
  ScrollBar1.Parent:= Form1;
  ScrollBar1.Visible:= TRUE;
  ScrollBar1.Top:= 48;
  ScrollBar1.Left:= 250;
  ScrollBar1.Name:= 'Ricksbar';
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  ScrollBar1.Free;
end;

initialization

RegisterClass(MyScrollBar);
```

See Also

DataString (see page 3104)

23.1.51.3 TStringStream Methods

The methods of the TStringStream class are listed here.

Protected Methods

	Name	Description
	SetSize (see page 3102)	This is SetSize, a member of class TStringStream.

Public Methods

	Name	Description
	Read (see page 3097)	Reads up to Count bytes from the string stream into Buffer, and advances the current position of the stream by the number of bytes read.
	ReadString (see page 3098)	Returns a string that consists of up to Count bytes from the current position in the string stream, and advances the current position of the stream by the number of bytes read.
	Seek (see page 3098)	Moves the current position in the stream by Offset bytes, relative to a specified origin.
	Write (see page 3103)	Write writes Count bytes from Buffer to the current position in the string stream and updates the current position by Count bytes.
	WriteString (see page 3103)	Concatenates a specified string to the current position in the string stream, and updates the current position accordingly.

23.1.51.3.1 TStringStream.Read Method

Reads up to Count bytes from the string stream into Buffer, and advances the current position of the stream by the number of bytes read.

Pascal

```
function Read(var Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Read( Buffer, Longint Count);
```

Description

Use Read to read the contents of the string stream into a buffer, starting at the current position. If Count extends beyond the end of the DataString (see page 3104), Read will only transfer the characters up to the end of the DataString (see page 3104), and advances the current position accordingly. If the return value is less than Count, it means that the end of the string was read.

The ReadBuffer (see page 2983) method and ReadComponent (see page 2983) method call Read to do the actual reading.

Note: Read treats Count as an upper bound. The ReadBuffer (see page 2983) method, by contrast, raises an exception if Count bytes cannot be read.

See Also

DataString (see page 3104)

Position (see page 2996)

ReadBuffer (see page 2983)

ReadComponent (see page 2983)

ReadString (see page 3098)

Write (see page 3103)

23.1.51.3.2 TStringStream.ReadString Method

Returns a string that consists of up to Count bytes from the current position in the string stream, and advances the current position of the stream by the number of bytes read.

Pascal

```
function ReadString(Count: Longint): string;
```

C++

```
__fastcall AnsiString ReadString(Longint Count);
```

Description

Use Read (see page 3097) to read the contents of the string stream into a string, starting at the current position. If Count extends beyond the end of the DataString (see page 3104), the returned string will only contain the characters up to the end of DataString (see page 3104), and the current position is advanced accordingly.

Note: ReadString does the same thing as Read (see page 3097), except that it returns a string rather than filling a buffer.

See Also

DataString (see page 3104)

Position (see page 2996)

Read (see page 3097)

WriteString (see page 3103)

23.1.51.3.3 TStringStream.Seek Method

Moves the current position in the stream by Offset bytes, relative to a specified origin.

Pascal

```
function Seek(Offset: Longint; Origin: Word): Longint; override;
```

C++

```
virtual __fastcall Longint Seek(Longint Offset, Word Origin);
```

Description

Use Seek to move the current position within the string stream by the indicated offset. Seek allows an application to read from or write to a particular location within the DataString (see page 3104).

If Offset is a negative number, the seek is backwards from the specified origin. The following table shows the different values of Origin and their meanings for seeking.

Value	Meaning
soFromBeginning (see page 3319)	Offset is from the beginning of DataString (see page 3104). Seek moves to the position Offset. Offset must be >= 0.
soFromCurrent (see page 3320)	Offset is from the current position. Seek moves to Position (see page 2996) + Offset.
soFromEnd (see page 3320)	Offset is from the end of Memory. Offset must be <= 0 to indicate a number of bytes before the end of the DataString (see page 3104).

Seek returns the new value of the Position (see page 2996) property.

C++ Examples:

```
/*
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
*/
class MyScrollBar : public TScrollBar
{
__published:    // IDE-managed Components
private:    // User declarations
public:     // User declarations
    __fastcall MyScrollBar(TComponent* Owner);
};

__fastcall MyScrollBar::MyScrollBar(TComponent* Owner)
    : TScrollBar(Owner)
{
}

void RegisterClassesWithStreamingSystem(void)
{
  // Make sure that as part of the startup
  // code our streaming classes are registered
  // with the streaming system.

  #pragma startup RegisterClassesWithStreamingSystem

  Classes::RegisterClass(__classid(MyScrollBar));
}

MyScrollBar *ScrollBar1;

AnsiString __fastcall ComponentToString(TComponent *c)
{
 AnsiString as;
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);
```

```
  try
  {
     pms->WriteComponent(c);
     pms->Seek(0, soFromBeginning);
     ObjectBinaryToText(pms, pss);
     pss->Seek(0, soFromBeginning);
     as = pss->DataString;
  }
  catch(...)
  {
     ShowMessage("Streaming error.");
  }

  delete pms;
  delete pss;
  return as;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Memo1->Text = ComponentToString(ScrollBar1);
}

TComponent *__fastcall StringToComponent(AnsiString as)

{
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);
 TComponent *pc;

 try
 {
    ObjectTextToBinary(pss, pms);
    pms->Seek(0, soFromBeginning);
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 pc = pms->ReadComponent(NULL);
 delete pms;
 delete pss;
 return pc;
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  TComponent *temp = StringToComponent(Memo1->Text);
  ScrollBar1->Free();

  ScrollBar1 = dynamic_cast<MyScrollBar *>(temp);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ScrollBar1 = new MyScrollBar(Form1);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
  ScrollBar1->Top = 48;
  ScrollBar1->Left = 250;
  ScrollBar1->Name = "Ricksbar";
}
```

```
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete ScrollBar1;
}
```

Delphi Examples:

```
{
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
}
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  MyScrollBar = class(TScrollBar)
  end;

var
  Form1: TForm1;
  ScrollBar1: MyScrollBar;

implementation

{$R *.dfm}

function ComponentToStringProc(Component: TComponent): string;
var
  BinStream:TMemoryStream;
  StrStream: TStringStream;
  s: string;
begin
  BinStream := TMemoryStream.Create;
  try
    StrStream := TStringStream.Create(s);
    try
      BinStream.WriteComponent(Component);
      BinStream.Seek(0, soFromBeginning);
      ObjectBinaryToText(BinStream, StrStream);
      StrStream.Seek(0, soFromBeginning);
      Result:= StrStream.DataString;
    finally
      StrStream.Free;
    end;
  finally
    BinStream.Free
  end;
end;

function StringToComponentProc(Value: string): TComponent;
var
  StrStream:TStringStream;
  BinStream: TMemoryStream;
begin
  StrStream := TStringStream.Create(Value);
  try
```

```
      BinStream := TMemoryStream.Create;
      try
        ObjectTextToBinary(StrStream, BinStream);
        BinStream.Seek(0, soFromBeginning);
        Result:= BinStream.ReadComponent(nil);
      finally
        BinStream.Free;
      end;
    finally
      StrStream.Free;
    end;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Memo1.Text:= ComponentToStringProc(ScrollBar1);
  end;

  // Edit the scrollbar parameters in the memo, click this
  // button and see the scrollbar change.
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    ScrollBar1.Free;
    ScrollBar1:= (StringToComponentProc(Memo1.Text) as MyScrollBar);
    ScrollBar1.Parent:= Form1;
    ScrollBar1.Visible:= TRUE;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ScrollBar1:= MyScrollBar.Create(Form1);
    ScrollBar1.Parent:= Form1;
    ScrollBar1.Visible:= TRUE;
    ScrollBar1.Top:= 48;
    ScrollBar1.Left:= 250;
    ScrollBar1.Name:= 'Ricksbar';
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    ScrollBar1.Free;
  end;

  initialization

  RegisterClass(MyScrollBar);
```

See Also

DataString (see page 3104)

Position (see page 2996)

Read (see page 3097)

Write (see page 3103)

23.1.51.3.4 TStringStream.SetSize Method

Pascal

```
procedure SetSize(NewSize: Longint); override;
```

C++

```
virtual __fastcall SetSize(Longint NewSize);
```

Description

This is SetSize, a member of class TStringStream.

23.1.51.3.5 TStringStream.Write Method

Write writes Count bytes from Buffer to the current position in the string stream and updates the current position by Count bytes.

Pascal

```
function Write(const Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  Buffer, Longint Count);
```

Description

Use Write to replace the end of the string from the current position on with the first Count bytes of Buffer. Write will change the Size (see page 2997) property to reflect the new length of the DataString (see page 3104).

Write always writes Count bytes from Buffer, even if a null character is included as one of the Count bytes. Thus, Write is equivalent to the WriteBuffer (see page 2990) method.

See Also

DataString (see page 3104)

Position (see page 2996)

Read (see page 3097)

Size

WriteBuffer (see page 2990)

WriteComponent (see page 2990)

WriteString (see page 3103)

23.1.51.3.6 TStringStream.WriteString Method

Concatenates a specified string to the current position in the string stream, and updates the current position accordingly.

Pascal

```
procedure WriteString(const AString: string);
```

C++

```
__fastcall WriteString(const AnsiString AString);
```

Description

Use WriteString to replace the end of the string from the current position on with the string specified by the AString parameter. WriteString changes the Size (see page 2997) property to reflect the new length of the DataString (see page 3104).

See Also

DataString (see page 3104)

Position (see page 2996)

ReadString (see page 3098)

Size (see page 2997)

Write (see page 3103)

23.1.51.4 TStringStream Properties

The properties of the TStringStream class are listed here.

Public Properties

	Name	Description
	DataString (see page 3104)	Provides direct access to the string that stores the text represented by the TStringStream (see page 3088) object.

23.1.51.4.1 TStringStream.DataString Property

Provides direct access to the string that stores the text represented by the TStringStream (see page 3088) object.

Pascal

```
property DataString: string;
```

C++

```
__property AnsiString DataString;
```

Description

Use DataString to get access to the text of the stream. The text represents the information that is being transferred by means of the string stream. Size (see page 2997) is the number of bytes in the string, and Position (see page 2996) is the current position within DataString.

Note: DataString is a read-only property. DataString can be used to change the contents of the string, but applications can't change the DataString itself.

C++ Examples:

```
/*
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
*/
class MyScrollBar : public TScrollBar
{
__published:    // IDE-managed Components
private:    // User declarations
public:     // User declarations
    __fastcall MyScrollBar(TComponent* Owner);
};

__fastcall MyScrollBar::MyScrollBar(TComponent* Owner)
    : TScrollBar(Owner)
{
}

void RegisterClassesWithStreamingSystem(void)
{
  // Make sure that as part of the startup
  // code our streaming classes are registered
  // with the streaming system.

  #pragma startup RegisterClassesWithStreamingSystem

  Classes::RegisterClass(__classid(MyScrollBar));
```

```
}

MyScrollBar *ScrollBar1;

AnsiString __fastcall ComponentToString(TComponent *c)
{
 AnsiString as;
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);

 try
 {
    pms->WriteComponent(c);
    pms->Seek(0, soFromBeginning);
    ObjectBinaryToText(pms, pss);
    pss->Seek(0, soFromBeginning);
    as = pss->DataString;
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 delete pms;
 delete pss;
 return as;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Memo1->Text = ComponentToString(ScrollBar1);
}

TComponent *__fastcall StringToComponent(AnsiString as)

{
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);
 TComponent *pc;

 try
 {
    ObjectTextToBinary(pss, pms);
    pms->Seek(0, soFromBeginning);
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 pc = pms->ReadComponent(NULL);
 delete pms;
 delete pss;
 return pc;
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  TComponent *temp = StringToComponent(Memo1->Text);
  ScrollBar1->Free();

  ScrollBar1 = dynamic_cast<MyScrollBar *>(temp);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ScrollBar1 = new MyScrollBar(Form1);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
  ScrollBar1->Top = 48;
  ScrollBar1->Left = 250;
  ScrollBar1->Name = "Ricksbar";
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete ScrollBar1;
}
```

Delphi Examples:

```
{
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
}
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  MyScrollBar = class(TScrollBar)
  end;

var
  Form1: TForm1;
  ScrollBar1: MyScrollBar;

implementation

{$R *.dfm}

function ComponentToStringProc(Component: TComponent): string;
var
  BinStream:TMemoryStream;
  StrStream: TStringStream;
  s: string;
begin
  BinStream := TMemoryStream.Create;
  try
    StrStream := TStringStream.Create(s);
    try
      BinStream.WriteComponent(Component);
      BinStream.Seek(0, soFromBeginning);
      ObjectBinaryToText(BinStream, StrStream);
      StrStream.Seek(0, soFromBeginning);
      Result:= StrStream.DataString;
    finally
      StrStream.Free;
    end;
  finally
    BinStream.Free
```

```
    end;
  end;

  function StringToComponentProc(Value: string): TComponent;
  var
    StrStream:TStringStream;
    BinStream: TMemoryStream;
  begin
    StrStream := TStringStream.Create(Value);
    try
      BinStream := TMemoryStream.Create;
      try
        ObjectTextToBinary(StrStream, BinStream);
        BinStream.Seek(0, soFromBeginning);
        Result:= BinStream.ReadComponent(nil);
      finally
        BinStream.Free;
      end;
    finally
      StrStream.Free;
    end;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Memo1.Text:= ComponentToStringProc(ScrollBar1);
  end;

  // Edit the scrollbar parameters in the memo, click this
  // button and see the scrollbar change.
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    ScrollBar1.Free;
    ScrollBar1:= (StringToComponentProc(Memo1.Text) as MyScrollBar);
    ScrollBar1.Parent:= Form1;
    ScrollBar1.Visible:= TRUE;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ScrollBar1:= MyScrollBar.Create(Form1);
    ScrollBar1.Parent:= Form1;
    ScrollBar1.Visible:= TRUE;
    ScrollBar1.Top:= 48;
    ScrollBar1.Left:= 250;
    ScrollBar1.Name:= 'Ricksbar';
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    ScrollBar1.Free;
  end;

  initialization

  RegisterClass(MyScrollBar);
```

See Also

Size

23.1.52 TThread Class

TThread is an abstract class that enables creation of separate threads of execution in an application.

Class Hierarchy

Classes.TThread

Pascal

```
TThread = class;
```

C++

```
class TThread;
```

File

Classes

Description

Create (see page 3110) a descendant of TThread to represent an execution thread in a multi-threaded application. Each new instance of a TThread descendant is a new thread of execution. Multiple instances of a TThread derived class make an application multi-threaded.

When an application is run, it is loaded into memory ready for execution. At this point it becomes a process containing one or more threads that contain the data, code and other system resources for the program. A thread executes one part of an application and is allocated CPU time by the operating system. All threads of a process share the same address space and can access the process's global variables.

Use threads to improve application performance by

- Managing input from several communication devices.
- Distinguishing among tasks of varying priority. For example, a high priority thread handles time critical tasks, and a low priority thread performs other tasks.

Following are issues and recommendations to be aware of when using threads:

- Keeping track of too many threads consumes CPU time; the recommended limit is 16 active threads per process on single processor systems.
- When multiple threads update the same resources, they must be synchronized to avoid conflicts.
- Most methods that access an object and update a form must only be called from within the main thread or use a synchronization object such as TMultiReadExclusiveWriteSynchronizer.

Define the thread object's Execute (see page 3115) method by inserting the code that should execute when the thread is executed.

Note: See the Threads demo for multi-threading program samples.

See Also

TCriticalSection (see page 22058)

TEvent (see page 22063)

TThreadList (see page 3149)

TMultiReadExclusiveWriteSynchronizer (see page 22766)

Service Threads

Defining Thread Objects

23.1.52.1 TThread Members

The following tables list the members exposed by TThread.

Protected Methods

	Name	Description
	CheckThreadError (see page 3114)	This is the overview for the CheckThreadError method overload.
V	DoTerminate (see page 3115)	Generates an OnTerminate (see page 3148) event.
A	Execute (see page 3115)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Protected Properties

	Name	Description
	ReturnValue (see page 3141)	Specifies the value returned to other waiting threads when the thread finishes executing.
R	Terminated (see page 3142)	Indicates whether the thread has been asked to terminate.

Public Events

	Name	Description
	OnTerminate (see page 3148)	Occurs after the thread's Execute (see page 3115) method has returned and before the thread is destroyed.

Public Methods

	Name	Description
V	AfterConstruction (see page 3114)	Executes the thread when the thread is created with CreateSuspended set to false.
	Create (see page 3110)	Creates an instance of a thread object.
V	Destroy (see page 3113)	Destroys the thread object and releases the memory allocated to it.
	Queue (see page 3119)	This is the overview for the Queue method overload.
	RemoveQueuedEvents (see page 3120)	Remove queued method calls.
	Resume (see page 3120)	Restarts the execution of a suspended thread.
	StaticQueue (see page 3124)	Asynchronously executes a method call within the main thread.
	StaticSynchronize (see page 3125)	Executes a method call within the main thread.
	Suspend (see page 3125)	Pauses a running thread.
	Synchronize (see page 3126)	This is the overview for the Synchronize method overload.
	Terminate (see page 3126)	Signals the thread to terminate by setting the Terminated (see page 3142) property to true.
	WaitFor (see page 3132)	Waits for the thread to terminate and then returns the value of the ReturnValue (see page 3141) property.

Public Properties

	Name	Description
R	FatalException (see page 3133)	Determines whether the thread object is automatically destroyed when the thread terminates.
	FreeOnTerminate (see page 3134)	Determines whether the thread object is automatically destroyed when the thread terminates.
R	Handle (see page 3137)	Indicates the thread's handle (Windows only).
	Priority (see page 3138)	Determines the thread's scheduling priority relative to other threads in the process.
	Suspended (see page 3142)	Indicates whether a thread is suspended.
R	ThreadID (see page 3148)	Identifies the thread throughout the system.

23.1.52.2 TThread.Create Constructor

Creates an instance of a thread object.

Pascal

```
constructor Create(CreateSuspended: Boolean);
```

C++

```
__fastcall TThread(Boolean CreateSuspended);
```

Description

Call Create to create a thread in an application. If CreateSuspended is False, Execute (see page 3115) is called immediately. If CreateSuspended is True, Execute (see page 3115) won't be called until after Resume (see page 3120) is called.

C++ Examples:

```
/*
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes. To
create the extension choose the
File | New | Other | Thread Object menu.
*/
bool mythreadRunning;
bool youthreadRunning;

class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
}

class TYouThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};

__fastcall TYouThread::TYouThread(bool suspended)
    : TThread(suspended)
{
}

void __fastcall TMyThread::Execute()
{
  Form1->Memo1->Lines->Add("Process has been running for this many seconds:");
  for (int I = 0; I <= 10; I++)
  {
```

```
    Form1->Memo1->Lines->Add("Low priority process " + IntToStr(I));
    Sleep(1000);
  }
  mythreadRunning = FALSE;
}

void __fastcall TYouThread::Execute()
{
  Form1->Memo3->Lines->Add("Second low priority process has been running for this many
seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo3->Lines->Add("Second low priority process " + IntToStr(I));
    Sleep(1000);
  }
  youthreadRunning = FALSE;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMyThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    SecondProcess = new TMyThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form1->Memo2->Lines->Add("Do some work in the main process for 10 seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo2->Lines->Add("Main process " + IntToStr(I));
    Sleep(1000);
  }
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  TYouThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    SecondProcess = new TYouThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  mythreadRunning = FALSE;
  youthreadRunning = FALSE;
}
```

Delphi Examples:

```
{
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes.  The
Execute procedure must be overridden in the extension class
of TThread or an "Abstract Error" will result. To create the
extension choose the File | New | Other | Thread Object menu.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (mythreadRunning = FALSE) then
  begin
    mythreadRunning:= TRUE;
    MyProcess := TMyThread.Create(True); { create suspended - secondprocess does not run yet }
    MyProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    MyProcess.Priority := tpLower;  // set the priority to lower than normal
    MyProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  if mythreadRunning then MyProcess.Terminate();
end;

procedure TMyThread.Execute;
var I: Integer;
begin
  Form1.Memo1.Lines.Add('Process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo1.Lines.Add('Low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
  mythreadRunning:= FALSE;
end;

procedure TYouThread.Execute;
var I: Integer;
begin
  Form1.Memo3.Lines.Add('Second low priority process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo3.Lines.Add('Second low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
    youthreadRunning:= FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
var   I: Integer;
begin
  Form1.Memo2.Lines.Add('Do some work in the main process for 10 seconds:');
  for I := 0 to 10 do
  begin
    Form1.Memo2.Lines.Add('Main process ' + InttoStr(I));
    Sleep(1000);
  end;
end;
```

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  if (youthreadRunning = FALSE) then
  begin
    youthreadRunning:= TRUE;
    YouProcess := TYouThread.Create(True); { create suspended – secondprocess does not run yet
}
    YouProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    YouProcess.Priority := tpLower;  // set the priority to lower than normal
    YouProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  if youthreadRunning then YouProcess.Terminate();
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  mythreadRunning:= FALSE;
  youthreadRunning:= FALSE;
end;
```

See Also

Destroy (see page 3113)

Execute (see page 3115)

FreeOnTerminate (see page 3134)

23.1.52.3 TThread.Destroy Destructor

Destroys the thread object and releases the memory allocated to it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TThread();
```

Description

Do not call Destroy in an application. Instead, use Free to destroy a thread when FreeOnTerminate (see page 3134) is False.

Destroy signals the thread to terminate and then waits for the thread to return before calling the inherited Destroy method.

See Also

Create (see page 3110)

Terminate (see page 3126)

WaitFor (see page 3132)

FreeOnTerminate (see page 3134)

Free (see page 22300)

23.1.52.4 TThread Methods

The methods of the TThread class are listed here.

Protected Methods

	Name	Description
	CheckThreadError (see page 3114)	This is the overview for the CheckThreadError method overload.
V	DoTerminate (see page 3115)	Generates an OnTerminate (see page 3148) event.
A	Execute (see page 3115)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Public Methods

	Name	Description
V	AfterConstruction (see page 3114)	Executes the thread when the thread is created with CreateSuspended set to false.
	Queue (see page 3119)	This is the overview for the Queue method overload.
	RemoveQueuedEvents (see page 3120)	Remove queued method calls.
	Resume (see page 3120)	Restarts the execution of a suspended thread.
	StaticQueue (see page 3124)	Asynchronously executes a method call within the main thread.
	StaticSynchronize (see page 3125)	Executes a method call within the main thread.
	Suspend (see page 3125)	Pauses a running thread.
	Synchronize (see page 3126)	This is the overview for the Synchronize method overload.
	Terminate (see page 3126)	Signals the thread to terminate by setting the Terminated (see page 3142) property to true.
	WaitFor (see page 3132)	Waits for the thread to terminate and then returns the value of the ReturnValue (see page 3141) property.

23.1.52.4.1 TThread.AfterConstruction Method

Executes the thread when the thread is created with CreateSuspended set to false.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction launches the Execute (see page 3115) method when the thread is not created in a suspended state. By calling Execute (see page 3115) from the AfterConstruction method rather than the constructor, TThread (see page 3108) avoids a race condition where the thread may free itself after Execute (see page 3115) finishes but before the main thread calls AfterConstruction.

See Also

Create (see page 3110)

Execute (see page 3115)

23.1.52.4.2 CheckThreadError Method

This is the overview for the CheckThreadError method overload.

Overload List

	Name	Description
	TThread.CheckThreadError (Boolean) (see page 3115)	This is CheckThreadError, a member of class TThread.
	TThread.CheckThreadError (Integer) (see page 3115)	This is CheckThreadError, a member of class TThread.

23.1.52.4.2.1 TThread.CheckThreadError Method (Boolean)

Pascal

```
procedure CheckThreadError(Success: Boolean); overload;
```

C++

```
__fastcall CheckThreadError(Boolean Success);
```

Description

This is CheckThreadError, a member of class TThread.

23.1.52.4.2.2 TThread.CheckThreadError Method (Integer)

Pascal

```
procedure CheckThreadError(ErrCode: Integer); overload;
```

C++

```
__fastcall CheckThreadError(int ErrCode);
```

Description

This is CheckThreadError, a member of class TThread.

23.1.52.4.3 TThread.DoTerminate Method

Generates an OnTerminate (see page 3148) event.

Pascal

```
procedure DoTerminate; virtual;
```

C++

```
virtual __fastcall DoTerminate();
```

Description

DoTerminate calls the OnTerminate (see page 3148) event handler, but does not terminate the thread.

See Also

OnTerminate (see page 3148)

23.1.52.4.4 TThread.Execute Method

Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Pascal

```
procedure Execute; virtual; abstract;
```

C++

```
virtual __fastcall Execute() = 0;
```

Description

Override Execute and insert the code that should be executed when the thread runs. Execute is responsible for checking the value of the Terminated (see page 3142) property to determine if the thread needs to exit.

A thread executes when Create (see page 3110) is called if CreateSuspended set to false, or when Resume (see page 3120) is first called after the thread is created if CreateSuspended set to true.

Note: Do not use the properties and methods of other objects directly in the Execute method of a thread. Instead, separate the use of other objects into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize (see page 3126) method.

C++ Examples:

```
/*
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes. To
create the extension choose the
File | New | Other | Thread Object menu.
*/
bool mythreadRunning;
bool youthreadRunning;

class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
}

class TYouThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};

__fastcall TYouThread::TYouThread(bool suspended)
    : TThread(suspended)
{
}

void __fastcall TMyThread::Execute()
{
  Form1->Memo1->Lines->Add("Process has been running for this many seconds:");
  for (int I = 0; I <= 10; I++)
  {
```

```
    Form1->Memo1->Lines->Add("Low priority process " + IntToStr(I));
    Sleep(1000);
  }
  mythreadRunning = FALSE;
}

void __fastcall TYouThread::Execute()
{
  Form1->Memo3->Lines->Add("Second low priority process has been running for this many
seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo3->Lines->Add("Second low priority process " + IntToStr(I));
    Sleep(1000);
  }
  youthreadRunning = FALSE;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMyThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    SecondProcess = new TMyThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form1->Memo2->Lines->Add("Do some work in the main process for 10 seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo2->Lines->Add("Main process " + IntToStr(I));
    Sleep(1000);
  }
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  TYouThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    SecondProcess = new TYouThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  mythreadRunning = FALSE;
  youthreadRunning = FALSE;
}
```

Delphi Examples:

```
{
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes.  The
Execute procedure must be overridden in the extension class
of TThread or an "Abstract Error" will result. To create the
extension choose the File | New | Other | Thread Object menu.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (mythreadRunning = FALSE) then
  begin
    mythreadRunning:= TRUE;
    MyProcess := TMyThread.Create(True); { create suspended - secondprocess does not run yet }
    MyProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    MyProcess.Priority := tpLower;  // set the priority to lower than normal
    MyProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  if mythreadRunning then MyProcess.Terminate();
end;

procedure TMyThread.Execute;
var I: Integer;
begin
  Form1.Memo1.Lines.Add('Process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo1.Lines.Add('Low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
  mythreadRunning:= FALSE;
end;

procedure TYouThread.Execute;
var I: Integer;
begin
  Form1.Memo3.Lines.Add('Second low priority process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo3.Lines.Add('Second low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
    youthreadRunning:= FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
var   I: Integer;
begin
  Form1.Memo2.Lines.Add('Do some work in the main process for 10 seconds:');
  for I := 0 to 10 do
  begin
    Form1.Memo2.Lines.Add('Main process ' + InttoStr(I));
    Sleep(1000);
  end;
end;
```

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  if (youthreadRunning = FALSE) then
  begin
    youthreadRunning:= TRUE;
    YouProcess := TYouThread.Create(True); { create suspended – secondprocess does not run yet
}
    YouProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    YouProcess.Priority := tpLower;  // set the priority to lower than normal
    YouProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  if youthreadRunning then YouProcess.Terminate();
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  mythreadRunning:= FALSE;
  youthreadRunning:= FALSE;
end;
```

See Also

Create (see page 3110)

Terminated (see page 3142)

Synchronize (see page 3126)

Checking for Termination by Other Threads

Waiting for a Thread to Finish Executing

Writing Clean-up Code

Writing the Thread Function

23.1.52.4.5 Queue Method

This is the overview for the Queue method overload.

Overload List

	Name	Description
	TThread.Queue (TThread, TThreadMethod) (see page 3119)	Asynchronously executes a method call within the main thread.
	TThread.Queue (TThreadMethod) (see page 3119)	Asynchronously executes a method call within the main thread.

23.1.52.4.5.1 TThread.Queue Method (TThread, TThreadMethod)

Asynchronously executes a method call within the main thread.

Pascal

```
procedure Queue(AMethod: TThreadMethod); overload;
class procedure Queue(AThread: TThread; AMethod: TThreadMethod); overload;
```

C++

```
__fastcall Queue(TThreadMethod AMethod);
__fastcall Queue(TThread AThread, TThreadMethod AMethod);
```

Description

Queue causes the call specified by **AMethod** to be executed using the main thread, thereby avoiding multi-thread conflicts. The current thread is passed in the **AThread** parameter.

If you are unsure whether a method call is thread-safe, call it from within the Synchronize (see page 3126) or Queue methods to ensure that it executes in the main thread.

Unlike Synchronize (see page 3126), execution of the current thread is allowed to continue. The main thread will eventually process all queued methods.

Warning: Do not call Queue from within the main thread. This can cause an infinite loop.

Note: You can also protect unsafe methods using critical sections or the multi-read exclusive-write synchronizer.

See Also

Execute (see page 3115)

Synchronize (see page 3126)

23.1.52.4.6 TThread.RemoveQueuedEvents Method

Remove queued method calls.

Pascal

```
class procedure RemoveQueuedEvents(AThread: TThread; AMethod: TThreadMethod);
```

C++

```
__fastcall RemoveQueuedEvents(TThread AThread, TThreadMethod AMethod);
```

Description

The Queue (see page 3119) method is used by a thread to queue a request for the main thread to execute a method. This is mostly done where the safety of the main thread is important.

RemoveQueuedEvents removes queued method calls. The operation proceeds in one of two ways. If **AThread** is specified, then all method calls queued by this thread are removed. If **AThread** is nil, then all calls to the specified **AMethod** method are removed, regardless of the thread that queued the call.

See Also

Queue (see page 3119)

23.1.52.4.7 TThread.Resume Method

Restarts the execution of a suspended thread.

Pascal

```
procedure Resume;
```

C++

```
__fastcall Resume();
```

Description

Call Resume to cause a suspended thread to start running again. Calls to Suspend (see page 3125) can be nested; Resume must be called the same number of times Suspend (see page 3125) was called before the thread will resume execution.

C++ Examples:

```
/*
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes. To
create the extension choose the
File | New | Other | Thread Object menu.
*/
bool mythreadRunning;
bool youthreadRunning;

class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
}

class TYouThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};

__fastcall TYouThread::TYouThread(bool suspended)
    : TThread(suspended)
{
}

void __fastcall TMyThread::Execute()
{
  Form1->Memo1->Lines->Add("Process has been running for this many seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo1->Lines->Add("Low priority process " + IntToStr(I));
    Sleep(1000);
  }
  mythreadRunning = FALSE;
}

void __fastcall TYouThread::Execute()
{
  Form1->Memo3->Lines->Add("Second low priority process has been running for this many
seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo3->Lines->Add("Second low priority process " + IntToStr(I));
```

```
    Sleep(1000);
  }
  youthreadRunning = FALSE;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMyThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    SecondProcess = new TMyThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form1->Memo2->Lines->Add("Do some work in the main process for 10 seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo2->Lines->Add("Main process " + IntToStr(I));
    Sleep(1000);
  }
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  TYouThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    SecondProcess = new TYouThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  mythreadRunning = FALSE;
  youthreadRunning = FALSE;
}
```

Delphi Examples:

```
{
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread’s FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes.  The
Execute procedure must be overridden in the extension class
of TThread or an "Abstract Error" will result. To create the
extension choose the File | New | Other | Thread Object menu.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (mythreadRunning = FALSE) then
```

```
  begin
    mythreadRunning:= TRUE;
    MyProcess := TMyThread.Create(True); { create suspended - secondprocess does not run yet }
    MyProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    MyProcess.Priority := tpLower;  // set the priority to lower than normal
    MyProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  if mythreadRunning then MyProcess.Terminate();
end;

procedure TMyThread.Execute;
var I: Integer;
begin
  Form1.Memo1.Lines.Add('Process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo1.Lines.Add('Low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
  mythreadRunning:= FALSE;
end;

procedure TYouThread.Execute;
var I: Integer;
begin
  Form1.Memo3.Lines.Add('Second low priority process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo3.Lines.Add('Second low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
    youthreadRunning:= FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
var   I: Integer;
begin
  Form1.Memo2.Lines.Add('Do some work in the main process for 10 seconds:');
  for I := 0 to 10 do
  begin
    Form1.Memo2.Lines.Add('Main process ' + InttoStr(I));
    Sleep(1000);
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  if (youthreadRunning = FALSE) then
  begin
    youthreadRunning:= TRUE;
    YouProcess := TYouThread.Create(True); { create suspended – secondprocess does not run yet
}
    YouProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    YouProcess.Priority := tpLower;  // set the priority to lower than normal
    YouProcess.Resume; { now run the thread }
  end
  else
```

```
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  if youthreadRunning then YouProcess.Terminate();
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  mythreadRunning:= FALSE;
  youthreadRunning:= FALSE;
end;
```

See Also

Suspend (see page 3125)

Executing Thread Objects

Starting and Stopping Threads

23.1.52.4.8 TThread.StaticQueue Method

Asynchronously executes a method call within the main thread.

Pascal

```
class procedure StaticQueue(AThread: TThread; AMethod: TThreadMethod);
```

C++

```
__fastcall StaticQueue(TThread AThread, TThreadMethod AMethod);
```

Description

StaticQueue causes the call specified by **AMethod** to be executed using the main thread, thereby avoiding multi-thread conflicts. The current thread is passed in the **AThread** parameter.

If you are unsure whether a method call is thread-safe, call it from within the Synchronize (see page 3126) or StaticQueue methods to ensure that it executes in the main thread.

Unlike Synchronize (see page 3126), execution of the current thread is allowed to continue. The main thread will eventually process all queued methods.

Warning: Do not call StaticQueue from within the main thread. This can cause an infinite loop.

Note: StaticQueue simply calls Queue (see page 3119) with the same parameters.

Note: You can also protect unsafe methods using critical sections or the multi-read exclusive-write synchronizer.

See Also

Execute (see page 3115)

Queue (see page 3119)

Synchronize (see page 3126)

23.1.52.4.9 TThread.StaticSynchronize Method

Executes a method call within the main thread.

Pascal

```
class procedure StaticSynchronize(AThread: TThread; AMethod: TThreadMethod);
```

C++

```
__fastcall StaticSynchronize(TThread AThread, TThreadMethod AMethod);
```

Description

StaticSynchronize causes the call specified by **AMethod** to be executed using the main thread, thereby avoiding multi-thread conflicts. The current thread is passed in the **AThread** parameter.

If you are unsure whether a method call is thread-safe, call it from within the StaticSynchronize method to ensure that it executes in the main thread.

Execution of the current thread is suspended while Method executes in the main thread.

Warning: Do not call StaticSynchronize from within the main thread. This can cause an infinite loop.

Note: StaticSynchronize simply invokes Synchronize (see page 3126) with the same parameters.

Note: You can also protect unsafe methods using critical sections or the multi-read exclusive-write synchronizer.

See Also

Execute (see page 3115)

Synchronize (see page 3126)

OnTerminate (see page 3148)

TCriticalSection (see page 22058)

TMultiReadExclusiveWriteSynchronizer (see page 22766)

CheckSynchronize

Using the Main VCL

23.1.52.4.10 TThread.Suspend Method

Pauses a running thread.

Pascal

```
procedure Suspend;
```

C++

```
__fastcall Suspend();
```

Description

Call Suspend to temporarily halt execution of the thread. To resume execution after a call to Suspend, call Resume (see page 3120). Calls to Suspend can be nested; Resume (see page 3120) must be called the same number of times Suspend was called before the thread will resume execution.

See Also

Resume (see page 3120)

Starting and Stopping Threads

23.1.52.4.11 Synchronize Method

This is the overview for the Synchronize method overload.

Overload List

	Name	Description
	TThread.Synchronize (TThread, TThreadMethod) (see page 3126)	This is Synchronize, a member of class TThread.
	TThread.Synchronize (TThreadMethod) (see page 3126)	This is Synchronize, a member of class TThread.

23.1.52.4.11.1 TThread.Synchronize Method (TThread, TThreadMethod)

Pascal

```
class procedure Synchronize(AThread: TThread; AMethod: TThreadMethod); overload;
```

C++

```
__fastcall Synchronize(TThread AThread, TThreadMethod AMethod);
```

Description

This is Synchronize, a member of class TThread.

23.1.52.4.11.2 TThread.Synchronize Method (TThreadMethod)

Pascal

```
procedure Synchronize(AMethod: TThreadMethod); overload;
```

C++

```
__fastcall Synchronize(TThreadMethod AMethod);
```

Description

This is Synchronize, a member of class TThread.

23.1.52.4.12 TThread.Terminate Method

Signals the thread to terminate by setting the Terminated (see page 3142) property to true.

Pascal

```
procedure Terminate;
```

C++

```
__fastcall Terminate();
```

Description

Terminate sets the thread's Terminated (see page 3142) property to true, signaling that the thread should be terminated as soon as possible.

For Terminate to work, the thread's Execute (see page 3115) method and any methods that Execute (see page 3115) calls should check Terminated (see page 3142) periodically and exit when it's true.

Note: Unlike the Windows API TerminateThread, which forces the thread to terminate immediately, the Terminate method merely requests that the thread terminate. This allows the thread to perform any cleanup before it shuts down.

C++ Examples:

```
/*
The following example shows how to use the LockList and
UnlockList methods to access a thread-safe version  of a
TList object:
*/
class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
}

class TYouThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};

__fastcall TYouThread::TYouThread(bool suspended)
    : TThread(suspended)
{
}

class TListThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TListThread(bool suspended);
};

__fastcall TListThread::TListThread(bool suspended)
    : TThread(suspended)
{
}

TThreadList *threadList1;
bool mythreadRunning, youthreadRunning, listthreadRunning;
int globalCount;
TMyThread *secondProcess;
TYouThread *otherSecondProcess;
TListThread *listProcess; // thread arecustom descendants of TThread
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    secondProcess = new TMyThread(True); // create suspended – secondprocess does not run yet
    secondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    secondProcess->Priority = tpLower;  // set the priority to lower than normal
    secondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TMyThread::Execute()
{
  for (int I = 0; I <= 20; I++)
  {
    if (Terminated) break;
    TRadioButton *myRadio = new TRadioButton(Form1);
    globalCount = globalCount + 1;
    myRadio->Name = "RadioButton" + IntToStr(globalCount);
    threadList1->Add(myRadio);
    Sleep(1000);
  }
  mythreadRunning = FALSE;
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    otherSecondProcess = new TYouThread(True); // create suspended – other secondprocess does
not run yet
    otherSecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    otherSecondProcess->Priority = tpLower;  // set the priority to lower than normal
    otherSecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TYouThread::Execute()
{
  for (int I = 0; I <= 10; I++)
  {
    if (Terminated) break;
    try
    {
      TList *list = threadList1->LockList();
      if (2*I < list->Count)
      {
        TControl *Temp = (TControl *)list->Items[2*I];
        threadList1->Remove(Temp);
      }
    }
    __finally
    {
      threadList1->UnlockList();
    }
    if (Terminated)
      MessageDlg("youThread has been asked to terminate, but is still running!",
```

```
        mtInformation, TMsgDlgButtons() << mbOK, 0);
    Sleep(3000);
  }
  youthreadRunning = FALSE;
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  if (listthreadRunning == FALSE)
  {
    listthreadRunning = TRUE;
    listProcess = new TListThread(True); // create suspended – secondprocess does not run yet
    listProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    listProcess->Priority = tpLower;  // set the priority to lower than normal
    listProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TListThread::Execute()
{
  while(True)
  {
    if (Terminated) break;
    Form1->ListBox1->Clear();
    TList *myList = threadList1->LockList();
    try
    {
      for (int I = 0; I < myList->Count; I++)
      {
        TControl *Temp = (TControl *)myList->Items[I];
        Form1->ListBox1->Items->Add(Temp->Name);
      }
    }
    __finally
    {
      threadList1->UnlockList();
    }
    Sleep(1000);
  }
}

void __fastcall TForm1::Button4Click(TObject *Sender)
{
  if (listProcess != NULL)
    listProcess->Terminate();
  if (secondProcess != NULL)
    secondProcess->Terminate();
  if (otherSecondProcess != NULL)
    otherSecondProcess->Terminate();
}
```

Delphi Examples:

```
{
The following example shows how to use the LockList and
UnlockList methods to access a thread-safe version  of a
TList object:
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button3: TButton;
    ListBox1: TListBox;
    Button2: TButton;
```

```
    Button4: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  TListThread = class(TThread)
  protected
    procedure Execute; override;
  end;
  TMyThread = class(TThread)
  protected
    procedure Execute; override;
  end;
  TYouThread = class(TThread)
  protected
    procedure Execute; override;
  end;

var
  Form1: TForm1;
  threadList1: TThreadList;
  mythreadRunning, youthreadRunning, listThreadRunning: Boolean;
  globalCount: Integer;
  listProcess: TListThread; { TListThread is a custom descendant of TThread }
  secondProcess: TMyThread; { TMyThread is a custom descendant of TThread }
  otherSecondProcess: TYouThread; { TMyThread is a custom descendant of TThread }

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
  if (mythreadRunning = FALSE) then
  begin
    mythreadRunning:= TRUE;
    secondProcess := TMyThread.Create(True); { create suspended – secondprocess does not run
yet }
    secondProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    secondProcess.Priority := tpLower;  // set the priority to lower than normal
    secondProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TMyThread.Execute;
var
  I: Integer;
  myRadio: TRadioButton;
begin
  for I := 0 to 20 do
  begin
    if (Terminated) then
    begin
      mythreadRunning:= FALSE;
      exit;
    end;
```

```
    myRadio:= TRadioButton.Create(Form1);
    globalCount:= globalCount + 1;
    myRadio.Name:= 'RadioButton' + IntToStr(globalCount);
    threadList1.Add(myRadio);
    Sleep(1000);
  end;
  mythreadRunning:= FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  if (listthreadRunning = FALSE) then
  begin
    listThreadRunning:= TRUE;
    listProcess := TListThread.Create(True); { create suspended – secondprocess does not run
yet }
    listProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    listProcess.Priority := tpLower;  // set the priority to lower than normal
    listProcess.Resume; { now run the thread }
  end;
end;

procedure TListThread.Execute;
var
  I: Integer;
  Temp: TControl;
  myList: TList;
begin
  while(True) do
  begin
    if (Terminated) then
    begin
      listthreadRunning:= FALSE;
      exit;
    end;
    Form1.ListBox1.Clear;
    myList:= threadList1.LockList;
    try
      for I := 0 to myList.Count-1 do
      begin
        Temp:= myList.Items[I];
        Form1.ListBox1.Items.Add(Temp.Name);
      end;
    finally
      threadList1.UnlockList;
    end;
    Sleep(1000);
  end;
  listthreadRunning:= FALSE;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  if (youthreadRunning = FALSE) then
  begin
    youthreadRunning:= TRUE;
    otherSecondProcess := TYouThread.Create(True); { create suspended – secondprocess does not
run yet }
    otherSecondProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    otherSecondProcess.Priority := tpLower;  // set the priority to lower than normal
    otherSecondProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;
```

```
procedure TForm1.Button4Click(Sender: TObject);
begin
  if (listProcess <> nil) then
    listProcess.Terminate;
  if (secondProcess <> nil) then
    secondProcess.Terminate;
  if (otherSecondProcess <> nil) then
    otherSecondProcess.Terminate;
end;

procedure TYouThread.Execute;
var
  I: Integer;
  Temp: TControl;
begin
  for I := 0 to 10 do
  begin
    if (Terminated) then
    begin
      youThreadRunning:= FALSE;
      exit;
    end;
    with threadList1.LockList do
    try
      if (2*I < Count) then
      begin
        Temp:= Items[2*I];
        threadList1.Remove(Temp);
      end;
    finally
      threadList1.UnlockList;
    end;
    if (Terminated) then
      MessageDlg('youThread has been asked to terminate, but is still running!',
        mtInformation, [mbOk], 0);
    Sleep(3000);
  end;
  youthreadRunning:= FALSE;
end;
```

See Also

Execute (see page 3115)

Terminated (see page 3142)

Checking for Termination by Other Threads

Starting and Stopping Threads

23.1.52.4.13 TThread.WaitFor Method

Waits for the thread to terminate and then returns the value of the ReturnValue (see page 3141) property.

Pascal

```
function WaitFor: LongWord;
```

C++

```
__fastcall LongWord WaitFor();
```

Description

Call WaitFor to obtain the value of ReturnValue (see page 3141) when the thread finishes executing. WaitFor doesn't return until the thread terminates, so the thread must exit either by finishing the Execute (see page 3115) method or by exiting when

the Terminated (see page 3142) property is true.

See Also

TThread (see page 3108)

Execute (see page 3115)

ReturnValue (see page 3141)

Synchronize (see page 3126)

Waiting for a Thread to Finish Executing

23.1.52.5 TThread Properties

The properties of the TThread class are listed here.

Protected Properties

	Name	Description
	ReturnValue (see page 3141)	Specifies the value returned to other waiting threads when the thread finishes executing.
R	Terminated (see page 3142)	Indicates whether the thread has been asked to terminate.

Public Properties

	Name	Description
R	FatalException (see page 3133)	Determines whether the thread object is automatically destroyed when the thread terminates.
	FreeOnTerminate (see page 3134)	Determines whether the thread object is automatically destroyed when the thread terminates.
R	Handle (see page 3137)	Indicates the thread's handle (Windows only).
	Priority (see page 3138)	Determines the thread's scheduling priority relative to other threads in the process.
	Suspended (see page 3142)	Indicates whether a thread is suspended.
R	ThreadID (see page 3148)	Identifies the thread throughout the system.

23.1.52.5.1 TThread.FatalException Property

Determines whether the thread object is automatically destroyed when the thread terminates.

Pascal

```
property FatalException: TObject;
```

C++

```
__property TObject * FatalException;
```

Description

If the Execute (see page 3115) method raises an exception that is not caught and handled within that method, the thread terminates and sets FatalException to the exception object for that exception. Applications can check FatalException from an OnTerminate (see page 3148) event handler to determine whether the thread terminated due to an exception.

See Also

ReturnValue (see page 3141)

OnTerminate (see page 3148)

Exception (see page 22750)

Execute (see page 3115)

23.1.52.5.2 TThread.FreeOnTerminate Property

Determines whether the thread object is automatically destroyed when the thread terminates.

Pascal

```
property FreeOnTerminate: Boolean;
```

C++

```
__property Boolean FreeOnTerminate;
```

Description

Set FreeOnTerminate to true if you don't want to explicitly destroy threads after they finish executing. When FreeOnTerminate is false, the thread object must be explicitly destroyed in application code.

Warning: When FreeOnTerminate is true, the Execute (see page 3115) method may run and then free the thread before your application can execute the next line of code. Thus, you should not call any methods of the thread object when FreeOnTerminate is true unless you create the thread in a suspended state.

C++ Examples:

```
/*
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes. To
create the extension choose the
File | New | Other | Thread Object menu.
*/
bool mythreadRunning;
bool youthreadRunning;

class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
}

class TYouThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};

__fastcall TYouThread::TYouThread(bool suspended)
    : TThread(suspended)
{
```

```
}

void __fastcall TMyThread::Execute()
{
  Form1->Memo1->Lines->Add("Process has been running for this many seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo1->Lines->Add("Low priority process " + IntToStr(I));
    Sleep(1000);
  }
  mythreadRunning = FALSE;
}

void __fastcall TYouThread::Execute()
{
  Form1->Memo3->Lines->Add("Second low priority process has been running for this many
seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo3->Lines->Add("Second low priority process " + IntToStr(I));
    Sleep(1000);
  }
  youthreadRunning = FALSE;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMyThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    SecondProcess = new TMyThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form1->Memo2->Lines->Add("Do some work in the main process for 10 seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo2->Lines->Add("Main process " + IntToStr(I));
    Sleep(1000);
  }
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  TYouThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    SecondProcess = new TYouThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  mythreadRunning = FALSE;
  youthreadRunning = FALSE;
}
```

Delphi Examples:

```
{
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes.  The
Execute procedure must be overridden in the extension class
of TThread or an "Abstract Error" will result. To create the
extension choose the File | New | Other | Thread Object menu.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (mythreadRunning = FALSE) then
  begin
    mythreadRunning:= TRUE;
    MyProcess := TMyThread.Create(True); { create suspended - secondprocess does not run yet }
    MyProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    MyProcess.Priority := tpLower;  // set the priority to lower than normal
    MyProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  if mythreadRunning then MyProcess.Terminate();
end;

procedure TMyThread.Execute;
var I: Integer;
begin
  Form1.Memo1.Lines.Add('Process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo1.Lines.Add('Low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
  mythreadRunning:= FALSE;
end;

procedure TYouThread.Execute;
var I: Integer;
begin
  Form1.Memo3.Lines.Add('Second low priority process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo3.Lines.Add('Second low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
    youthreadRunning:= FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
var   I: Integer;
begin
```

```
    Form1.Memo2.Lines.Add('Do some work in the main process for 10 seconds:');
    for I := 0 to 10 do
    begin
      Form1.Memo2.Lines.Add('Main process ' + InttoStr(I));
      Sleep(1000);
    end;
  end;

  procedure TForm1.Button3Click(Sender: TObject);
  begin
    if (youthreadRunning = FALSE) then
    begin
      youthreadRunning:= TRUE;
      YouProcess := TYouThread.Create(True); { create suspended – secondprocess does not run yet
  }
      YouProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
      YouProcess.Priority := tpLower;  // set the priority to lower than normal
      YouProcess.Resume; { now run the thread }
    end
    else
      MessageDlg('This thread is still running.  You are going to hurt yourself!',
        mtInformation, [mbOk], 0);
  end;

  procedure TForm1.Button5Click(Sender: TObject);
  begin
    if youthreadRunning then YouProcess.Terminate();
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    mythreadRunning:= FALSE;
    youthreadRunning:= FALSE;
  end;
```

See Also

Terminated (see page 3142)

Terminate (see page 3126)

OnTerminate (see page 3148)

Create (see page 3110)

Initializing the Thread

23.1.52.5.3 TThread.Handle Property

Indicates the thread's handle (Windows only).

Pascal

```
property Handle: THandle;
```

C++

```
__property THandle Handle;
```

Description

On Windows, use Handle for calling Win32 API functions for thread manipulation.

The Handle property is not available on Linux.

See Also

ThreadID (see page 3148)

23.1.52.5.4 TThread.Priority Property

Determines the thread's scheduling priority relative to other threads in the process.

Pascal

```
property Priority: TThreadPriority;
```

C++

```
__property TThreadPriority Priority;
```

Description

Priority indicates the priority used when scheduling the thread. Adjust the priority higher or lower as needed.

On Win32, Priority is a TThread (see page 3108) Priority value. The possible values are listed in a table within the topic “Initializing the Thread” (see the link at the end of this topic). The system schedules CPU cycles to each thread based on a priority scale; the Priority property adjusts a thread's priority higher or lower on the scale.

On Linux, Priority is an integer value. A higher value (numerically) implies that the thread has greater priority. The possible values for priority depend on the value of the Policy property.

Warning: Boosting the thread priority of a CPU intensive operation may "starve" the other threads in the application. Only apply priority boosts to threads that spend most of their time waiting for external events.

C++ Examples:

```
/*
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread’s FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes. To
create the extension choose the
File | New | Other | Thread Object menu.
*/
bool mythreadRunning;
bool youthreadRunning;

class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
}

class TYouThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};

__fastcall TYouThread::TYouThread(bool suspended)
```

```
    : TThread(suspended)
{
}

void __fastcall TMyThread::Execute()
{
  Form1->Memo1->Lines->Add("Process has been running for this many seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo1->Lines->Add("Low priority process " + IntToStr(I));
    Sleep(1000);
  }
  mythreadRunning = FALSE;
}

void __fastcall TYouThread::Execute()
{
  Form1->Memo3->Lines->Add("Second low priority process has been running for this many
seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo3->Lines->Add("Second low priority process " + IntToStr(I));
    Sleep(1000);
  }
  youthreadRunning = FALSE;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMyThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    SecondProcess = new TMyThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form1->Memo2->Lines->Add("Do some work in the main process for 10 seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo2->Lines->Add("Main process " + IntToStr(I));
    Sleep(1000);
  }
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  TYouThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    SecondProcess = new TYouThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
```

```
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  mythreadRunning = FALSE;
  youthreadRunning = FALSE;
}
```

Delphi Examples:

```
{
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes.  The
Execute procedure must be overridden in the extension class
of TThread or an "Abstract Error" will result. To create the
extension choose the File | New | Other | Thread Object menu.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (mythreadRunning = FALSE) then
  begin
    mythreadRunning:= TRUE;
    MyProcess := TMyThread.Create(True); { create suspended - secondprocess does not run yet }
    MyProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    MyProcess.Priority := tpLower;  // set the priority to lower than normal
    MyProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  if mythreadRunning then MyProcess.Terminate();
end;

procedure TMyThread.Execute;
var I: Integer;
begin
  Form1.Memo1.Lines.Add('Process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo1.Lines.Add('Low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
  mythreadRunning:= FALSE;
end;

procedure TYouThread.Execute;
var I: Integer;
begin
  Form1.Memo3.Lines.Add('Second low priority process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo3.Lines.Add('Second low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
    youthreadRunning:= FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
```

```
var   I: Integer;
begin
  Form1.Memo2.Lines.Add('Do some work in the main process for 10 seconds:');
  for I := 0 to 10 do
  begin
    Form1.Memo2.Lines.Add('Main process ' + InttoStr(I));
    Sleep(1000);
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  if (youthreadRunning = FALSE) then
  begin
    youthreadRunning:= TRUE;
    YouProcess := TYouThread.Create(True); { create suspended – secondprocess does not run yet
}
    YouProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    YouProcess.Priority := tpLower;  // set the priority to lower than normal
    YouProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  if youthreadRunning then YouProcess.Terminate();
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  mythreadRunning:= FALSE;
  youthreadRunning:= FALSE;
end;
```

See Also

Execute (see page 3115)

Policy

Initializing the Thread

23.1.52.5.5 TThread.ReturnValue Property

Specifies the value returned to other waiting threads when the thread finishes executing.

Pascal

```
property ReturnValue: Integer;
```

C++

```
__property int ReturnValue;
```

Description

Use ReturnValue to indicate success/failure or numeric result/output of the thread to the application or other threads. The WaitFor (see page 3132) method returns the value stored in ReturnValue.

See Also

Execute (see page 3115)

WaitFor (see page 3132)

Waiting for a Thread to Finish Executing

23.1.52.5.6 TThread.Suspended Property

Indicates whether a thread is suspended.

Pascal

```
property Suspended: Boolean;
```

C++

```
__property Boolean Suspended;
```

Description

Set Suspended to true to suspend a thread; set it to false to resume it. Suspended threads do not continue execution until they are resumed.

See Also

Resume (see page 3120)

Suspend (see page 3125)

23.1.52.5.7 TThread.Terminated Property

Indicates whether the thread has been asked to terminate.

Pascal

```
property Terminated: Boolean;
```

C++

```
__property Boolean Terminated;
```

Description

The thread's Execute (see page 3115) method and any methods that Execute (see page 3115) calls should check Terminated periodically and exit when it's true. The Terminate (see page 3126) method sets the Terminated property to true.

The Terminate (see page 3126) method is the polite way to abort the execution of a thread, but it requires cooperation from the thread's Execute (see page 3115) code. Using Terminate (see page 3126) is recommended over the TerminateThread Win32 API call.

C++ Examples:

```
/*
The following example shows how to use the LockList and
UnlockList methods to access a thread-safe version  of a
TList object:
*/
class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
```

```
}

class TYouThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};

__fastcall TYouThread::TYouThread(bool suspended)
    : TThread(suspended)
{
}

class TListThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TListThread(bool suspended);
};

__fastcall TListThread::TListThread(bool suspended)
    : TThread(suspended)
{
}

TThreadList *threadList1;
bool mythreadRunning, youthreadRunning, listthreadRunning;
int globalCount;
TMyThread *secondProcess;
TYouThread *otherSecondProcess;
TListThread *listProcess; // thread arecustom descendants of TThread

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    secondProcess = new TMyThread(True); // create suspended – secondprocess does not run yet
    secondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    secondProcess->Priority = tpLower;  // set the priority to lower than normal
    secondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TMyThread::Execute()
{
  for (int I = 0; I <= 20; I++)
  {
    if (Terminated) break;
    TRadioButton *myRadio = new TRadioButton(Form1);
    globalCount = globalCount + 1;
    myRadio->Name = "RadioButton" + IntToStr(globalCount);
    threadList1->Add(myRadio);
    Sleep(1000);
  }
  mythreadRunning = FALSE;
```

```
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    otherSecondProcess = new TYouThread(True); // create suspended – other secondprocess does
not run yet
    otherSecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    otherSecondProcess->Priority = tpLower;  // set the priority to lower than normal
    otherSecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TYouThread::Execute()
{
  for (int I = 0; I <= 10; I++)
  {
    if (Terminated) break;
    try
    {
      TList *list = threadList1->LockList();
      if (2*I < list->Count)
      {
        TControl *Temp = (TControl *)list->Items[2*I];
        threadList1->Remove(Temp);
      }
    }
    __finally
    {
      threadList1->UnlockList();
    }
    if (Terminated)
      MessageDlg("youThread has been asked to terminate, but is still running!",
        mtInformation, TMsgDlgButtons() << mbOK, 0);
    Sleep(3000);
  }
  youthreadRunning = FALSE;
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  if (listthreadRunning == FALSE)
  {
    listthreadRunning = TRUE;
    listProcess = new TListThread(True); // create suspended – secondprocess does not run yet
    listProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    listProcess->Priority = tpLower;  // set the priority to lower than normal
    listProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TListThread::Execute()
{
  while(True)
  {
    if (Terminated) break;
    Form1->ListBox1->Clear();
    TList *myList = threadList1->LockList();
```

```
    try
    {
      for (int I = 0; I < myList->Count; I++)
      {
        TControl *Temp = (TControl *)myList->Items[I];
        Form1->ListBox1->Items->Add(Temp->Name);
      }
    }
    __finally
    {
      threadList1->UnlockList();
    }
    Sleep(1000);
  }
}

void __fastcall TForm1::Button4Click(TObject *Sender)
{
  if (listProcess != NULL)
    listProcess->Terminate();
  if (secondProcess != NULL)
    secondProcess->Terminate();
  if (otherSecondProcess != NULL)
    otherSecondProcess->Terminate();
}
```

Delphi Examples:

```
{
The following example shows how to use the LockList and
UnlockList methods to access a thread-safe version  of a
TList object:
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button3: TButton;
    ListBox1: TListBox;
    Button2: TButton;
    Button4: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  TListThread = class(TThread)
  protected
    procedure Execute; override;
  end;
  TMyThread = class(TThread)
  protected
    procedure Execute; override;
  end;
  TYouThread = class(TThread)
  protected
    procedure Execute; override;
  end;

var
  Form1: TForm1;
  threadList1: TThreadList;
```

```
    mythreadRunning, youthreadRunning, listThreadRunning: Boolean;
    globalCount: Integer;
    listProcess: TListThread; { TListThread is a custom descendant of TThread }
    secondProcess: TMyThread; { TMyThread is a custom descendant of TThread }
    otherSecondProcess: TYouThread; { TMyThread is a custom descendant of TThread }

  implementation

  {$R *.dfm}

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    if (mythreadRunning = FALSE) then
    begin
      mythreadRunning:= TRUE;
      secondProcess := TMyThread.Create(True); { create suspended – secondprocess does not run
  yet }
      secondProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
      secondProcess.Priority := tpLower;  // set the priority to lower than normal
      secondProcess.Resume; { now run the thread }
    end
    else
      MessageDlg('This thread is still running.  You are going to hurt yourself!',
        mtInformation, [mbOk], 0);
  end;

  procedure TMyThread.Execute;
  var
    I: Integer;
    myRadio: TRadioButton;
  begin
    for I := 0 to 20 do
    begin
      if (Terminated) then
      begin
        mythreadRunning:= FALSE;
        exit;
      end;
      myRadio:= TRadioButton.Create(Form1);
      globalCount:= globalCount + 1;
      myRadio.Name:= 'RadioButton' + IntToStr(globalCount);
      threadList1.Add(myRadio);
      Sleep(1000);
    end;
    mythreadRunning:= FALSE;
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    if (listthreadRunning = FALSE) then
    begin
      listThreadRunning:= TRUE;
      listProcess := TListThread.Create(True); { create suspended – secondprocess does not run
  yet }
      listProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
      listProcess.Priority := tpLower;  // set the priority to lower than normal
      listProcess.Resume; { now run the thread }
    end;
  end;

  procedure TListThread.Execute;
  var
    I: Integer;
    Temp: TControl;
    myList: TList;
  begin
```

```
    while(True) do
    begin
      if (Terminated) then
      begin
        listthreadRunning:= FALSE;
        exit;
      end;
      Form1.ListBox1.Clear;
      myList:= threadList1.LockList;
      try
        for I := 0 to myList.Count-1 do
        begin
          Temp:= myList.Items[I];
          Form1.ListBox1.Items.Add(Temp.Name);
        end;
      finally
        threadList1.UnlockList;
      end;
      Sleep(1000);
    end;
    listthreadRunning:= FALSE;
  end;

  procedure TForm1.Button3Click(Sender: TObject);
  begin
    if (youthreadRunning = FALSE) then
    begin
      youthreadRunning:= TRUE;
      otherSecondProcess := TYouThread.Create(True); { create suspended – secondprocess does not
run yet }
      otherSecondProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
      otherSecondProcess.Priority := tpLower;  // set the priority to lower than normal
      otherSecondProcess.Resume; { now run the thread }
    end
    else
      MessageDlg('This thread is still running.  You are going to hurt yourself!',
        mtInformation, [mbOk], 0);
  end;

  procedure TForm1.Button4Click(Sender: TObject);
  begin
    if (listProcess <> nil) then
      listProcess.Terminate;
    if (secondProcess <> nil) then
      secondProcess.Terminate;
    if (otherSecondProcess <> nil) then
      otherSecondProcess.Terminate;
  end;

  procedure TYouThread.Execute;
  var
    I: Integer;
    Temp: TControl;
  begin
    for I := 0 to 10 do
    begin
      if (Terminated) then
      begin
        youThreadRunning:= FALSE;
        exit;
      end;
      with threadList1.LockList do
      try
        if (2*I < Count) then
        begin
          Temp:= Items[2*I];
```

```
          threadList1.Remove(Temp);
        end;
      finally
        threadList1.UnlockList;
      end;
      if (Terminated) then
        MessageDlg('youThread has been asked to terminate, but is still running!',
          mtInformation, [mbOk], 0);
      Sleep(3000);
    end;
    youthreadRunning:= FALSE;
  end;
```

See Also

Execute (see page 3115)

OnTerminate (see page 3148)

Terminate (see page 3126)

Checking for Termination by Other Threads

Starting and Stopping Threads

23.1.52.5.8 TThread.ThreadID Property

Identifies the thread throughout the system.

Pascal

```
property ThreadID: THandle;
```

C++

```
__property THandle ThreadID;
```

Description

Use ThreadID, during debugging, to identify the thread in the Threads status box. On Win32, ThreadID is also useful when calling Win32 API functions for manipulating the thread.

ThreadID is different than the thread's handle in the Handle (see page 3137) property.

See Also

Handle (see page 3137)

23.1.52.6 TThread Events

The events of the TThread class are listed here.

Public Events

	Name	Description
	OnTerminate (see page 3148)	Occurs after the thread's Execute (see page 3115) method has returned and before the thread is destroyed.

23.1.52.6.1 TThread.OnTerminate Event

Occurs after the thread's Execute (see page 3115) method has returned and before the thread is destroyed.

Pascal

```
property OnTerminate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnTerminate;
```

Description

Write an OnTerminate event handler to execute code after the thread finishes executing. The OnTerminate event handler is called in the context of the main thread, which means methods and properties can be called freely.

See Also

DoTerminate (see page 3115)

Terminated (see page 3142)

FatalException (see page 3133)

Waiting for a Task to Be Completed

Writing Clean-up Code

23.1.53 TThreadList Class

TThreadList represents a thread-safe list.

Class Hierarchy

Classes.TThreadList

Pascal

```
TThreadList = class;
```

C++

```
class TThreadList;
```

File

Classes

Description

A TThreadList object is a thread-safe list. Each TThreadList maintains a private TList (see page 2868) (a list of pointers to objects). You can add or remove items in a TThreadList from multiple threads without explicit locks.

To access the actual TList (see page 2868) object managed by the thread list, first lock the list by calling the LockList (see page 3157) method. Unlock the list by calling the Unlock method when done.

Tip: By default, TThreadList ignores attempts to add duplicate entries to the list. If the list is large, this default is computationally expensive. For better performance, you may want to change the Duplicates (see page 3175) property to dupAccept where possible.

See Also

TList (see page 2868)

Working with Lists

Locking Objects

Using the Main VCL

23.1.53.1 TThreadList Members

The following tables list the members exposed by TThreadList.

Public Methods

	Name	Description
	Add (see page 3151)	Adds an item to the list.
	Clear (see page 3157)	Deletes all items from the list.
	Create (see page 3150)	Creates and initializes a TThreadList (see page 3149) instance.
V	Destroy (see page 3150)	Destroys the TThreadList (see page 3149) and frees its memory.
	LockList (see page 3157)	Locks the thread-safe list and returns a TList (see page 2868) object.
	Remove (see page 3163)	Removes an item from the list.
	UnlockList (see page 3169)	Unlocks the thread-safe list.

Public Properties

	Name	Description
	Duplicates (see page 3175)	Indicates whether the thread list allows duplicate entries.

23.1.53.2 TThreadList.Create Constructor

Creates and initializes a TThreadList (see page 3149) instance.

Pascal

```
constructor Create;
```

C++

```
__fastcall TThreadList();
```

Description

The Create method generates a TThreadList (see page 3149) object.

23.1.53.3 TThreadList.Destroy Destructor

Destroys the TThreadList (see page 3149) and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TThreadList();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the thread list is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

23.1.53.4 TThreadList Methods

The methods of the TThreadList class are listed here.

Public Methods

	Name	Description
	Add (see page 3151)	Adds an item to the list.
	Clear (see page 3157)	Deletes all items from the list.
	LockList (see page 3157)	Locks the thread-safe list and returns a TList (see page 2868) object.
	Remove (see page 3163)	Removes an item from the list.
	UnlockList (see page 3169)	Unlocks the thread-safe list.

23.1.53.4.1 TThreadList.Add Method

Adds an item to the list.

Pascal

```
procedure Add(Item: Pointer);
```

C++

```
__fastcall Add(void * Item);
```

Description

Call Add to insert a new object at the end of the thread-safe list.

C++ Examples:

```
/*
The following example shows how to use the LockList and
UnlockList methods to access a thread-safe version  of a
TList object:
*/
class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
}

class TYouThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};

__fastcall TYouThread::TYouThread(bool suspended)
    : TThread(suspended)
```

```
{
}

class TListThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TListThread(bool suspended);
};

__fastcall TListThread::TListThread(bool suspended)
    : TThread(suspended)
{
}

TThreadList *threadList1;
bool mythreadRunning, youthreadRunning, listthreadRunning;
int globalCount;
TMyThread *secondProcess;
TYouThread *otherSecondProcess;
TListThread *listProcess; // thread arecustom descendants of TThread

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    secondProcess = new TMyThread(True); // create suspended – secondprocess does not run yet
    secondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    secondProcess->Priority = tpLower;  // set the priority to lower than normal
    secondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TMyThread::Execute()
{
  for (int I = 0; I <= 20; I++)
  {
    if (Terminated) break;
    TRadioButton *myRadio = new TRadioButton(Form1);
    globalCount = globalCount + 1;
    myRadio->Name = "RadioButton" + IntToStr(globalCount);
    threadList1->Add(myRadio);
    Sleep(1000);
  }
  mythreadRunning = FALSE;
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    otherSecondProcess = new TYouThread(True); // create suspended – other secondprocess does
not run yet
    otherSecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    otherSecondProcess->Priority = tpLower;  // set the priority to lower than normal
    otherSecondProcess->Resume(); // now run the thread
  }
  else
```

```
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TYouThread::Execute()
{
  for (int I = 0; I <= 10; I++)
  {
    if (Terminated) break;
    try
    {
      TList *list = threadList1->LockList();
      if (2*I < list->Count)
      {
        TControl *Temp = (TControl *)list->Items[2*I];
        threadList1->Remove(Temp);
      }
    }
    __finally
    {
      threadList1->UnlockList();
    }
    if (Terminated)
      MessageDlg("youThread has been asked to terminate, but is still running!",
        mtInformation, TMsgDlgButtons() << mbOK, 0);
    Sleep(3000);
  }
  youthreadRunning = FALSE;
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  if (listthreadRunning == FALSE)
  {
    listthreadRunning = TRUE;
    listProcess = new TListThread(True); // create suspended – secondprocess does not run yet
    listProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    listProcess->Priority = tpLower;  // set the priority to lower than normal
    listProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TListThread::Execute()
{
  while(True)
  {
    if (Terminated) break;
    Form1->ListBox1->Clear();
    TList *myList = threadList1->LockList();
    try
    {
      for (int I = 0; I < myList->Count; I++)
      {
        TControl *Temp = (TControl *)myList->Items[I];
        Form1->ListBox1->Items->Add(Temp->Name);
      }
    }
    __finally
    {
      threadList1->UnlockList();
    }
    Sleep(1000);
  }
```

```
}

void __fastcall TForm1::Button4Click(TObject *Sender)
{
  if (listProcess != NULL)
    listProcess->Terminate();
  if (secondProcess != NULL)
    secondProcess->Terminate();
  if (otherSecondProcess != NULL)
    otherSecondProcess->Terminate();
}
```

Delphi Examples:

```
{
The following example shows how to use the LockList and
UnlockList methods to access a thread-safe version  of a
TList object:
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button3: TButton;
    ListBox1: TListBox;
    Button2: TButton;
    Button4: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  TListThread = class(TThread)
  protected
    procedure Execute; override;
  end;
  TMyThread = class(TThread)
  protected
    procedure Execute; override;
  end;
  TYouThread = class(TThread)
  protected
    procedure Execute; override;
  end;

var
  Form1: TForm1;
  threadList1: TThreadList;
  mythreadRunning, youthreadRunning, listThreadRunning: Boolean;
  globalCount: Integer;
  listProcess: TListThread; { TListThread is a custom descendant of TThread }
  secondProcess: TMyThread; { TMyThread is a custom descendant of TThread }
  otherSecondProcess: TYouThread; { TMyThread is a custom descendant of TThread }

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
  if (mythreadRunning = FALSE) then
  begin
```

```
    mythreadRunning:= TRUE;
    secondProcess := TMyThread.Create(True); { create suspended – secondprocess does not run
yet }
    secondProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    secondProcess.Priority := tpLower;  // set the priority to lower than normal
    secondProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TMyThread.Execute;
var
  I: Integer;
  myRadio: TRadioButton;
begin
  for I := 0 to 20 do
  begin
    if (Terminated) then
    begin
      mythreadRunning:= FALSE;
      exit;
    end;
    myRadio:= TRadioButton.Create(Form1);
    globalCount:= globalCount + 1;
    myRadio.Name:= 'RadioButton' + IntToStr(globalCount);
    threadList1.Add(myRadio);
    Sleep(1000);
  end;
  mythreadRunning:= FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  if (listthreadRunning = FALSE) then
  begin
    listThreadRunning:= TRUE;
    listProcess := TListThread.Create(True); { create suspended – secondprocess does not run
yet }
    listProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    listProcess.Priority := tpLower;  // set the priority to lower than normal
    listProcess.Resume; { now run the thread }
  end;
end;

procedure TListThread.Execute;
var
  I: Integer;
  Temp: TControl;
  myList: TList;
begin
  while(True) do
  begin
    if (Terminated) then
    begin
      listthreadRunning:= FALSE;
      exit;
    end;
    Form1.ListBox1.Clear;
    myList:= threadList1.LockList;
    try
      for I := 0 to myList.Count-1 do
      begin
        Temp:= myList.Items[I];
        Form1.ListBox1.Items.Add(Temp.Name);
```

```
      end;
    finally
      threadList1.UnlockList;
    end;
    Sleep(1000);
  end;
  listthreadRunning:= FALSE;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  if (youthreadRunning = FALSE) then
  begin
    youthreadRunning:= TRUE;
    otherSecondProcess := TYouThread.Create(True); { create suspended – secondprocess does not
run yet }
    otherSecondProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    otherSecondProcess.Priority := tpLower;  // set the priority to lower than normal
    otherSecondProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  if (listProcess <> nil) then
    listProcess.Terminate;
  if (secondProcess <> nil) then
    secondProcess.Terminate;
  if (otherSecondProcess <> nil) then
    otherSecondProcess.Terminate;
end;

procedure TYouThread.Execute;
var
  I: Integer;
  Temp: TControl;
begin
  for I := 0 to 10 do
  begin
    if (Terminated) then
    begin
      youThreadRunning:= FALSE;
      exit;
    end;
    with threadList1.LockList do
    try
      if (2*I < Count) then
      begin
        Temp:= Items[2*I];
        threadList1.Remove(Temp);
      end;
    finally
      threadList1.UnlockList;
    end;
    if (Terminated) then
      MessageDlg('youThread has been asked to terminate, but is still running!',
        mtInformation, [mbOk], 0);
    Sleep(3000);
  end;
  youthreadRunning:= FALSE;
end;
```

23.1.53.4.2 TThreadList.Clear Method

Deletes all items from the list.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Clear locks the list, calls the Clear method in TList (see page 2868), then unlocks the list.

See Also

Clear (see page 2874)

23.1.53.4.3 TThreadList.LockList Method

Locks the thread-safe list and returns a TList (see page 2868) object.

Pascal

```
function LockList: TList;
```

C++

```
__fastcall TList LockList();
```

Description

Note: In Windows, LockList calls the Windows EnterCriticalSection function to lock the thread-safe list. It then returns the TList (see page 2868) object that contains the list.

Note: In Linux, LockList returns the TList (see page 2868) object that contains the list, after locking out all other threads.

C++ Examples:

```
/*
The following example shows how to use the LockList and
UnlockList methods to access a thread-safe version  of a
TList object:
*/
class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
}

class TYouThread : public TThread
```

```
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};

__fastcall TYouThread::TYouThread(bool suspended)
    : TThread(suspended)
{
}

class TListThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TListThread(bool suspended);
};

__fastcall TListThread::TListThread(bool suspended)
    : TThread(suspended)
{
}

TThreadList *threadList1;
bool mythreadRunning, youthreadRunning, listthreadRunning;
int globalCount;
TMyThread *secondProcess;
TYouThread *otherSecondProcess;
TListThread *listProcess; // thread arecustom descendants of TThread

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    secondProcess = new TMyThread(True); // create suspended – secondprocess does not run yet
    secondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    secondProcess->Priority = tpLower;  // set the priority to lower than normal
    secondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TMyThread::Execute()
{
  for (int I = 0; I <= 20; I++)
  {
    if (Terminated) break;
    TRadioButton *myRadio = new TRadioButton(Form1);
    globalCount = globalCount + 1;
    myRadio->Name = "RadioButton" + IntToStr(globalCount);
    threadList1->Add(myRadio);
    Sleep(1000);
  }
  mythreadRunning = FALSE;
}

void __fastcall TForm1::Button3Click(TObject *Sender)
```

```
{
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    otherSecondProcess = new TYouThread(True); // create suspended – other secondprocess does
not run yet
    otherSecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    otherSecondProcess->Priority = tpLower;  // set the priority to lower than normal
    otherSecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TYouThread::Execute()
{
  for (int I = 0; I <= 10; I++)
  {
    if (Terminated) break;
    try
    {
      TList *list = threadList1->LockList();
      if (2*I < list->Count)
      {
        TControl *Temp = (TControl *)list->Items[2*I];
        threadList1->Remove(Temp);
      }
    }
    __finally
    {
      threadList1->UnlockList();
    }
    if (Terminated)
      MessageDlg("youThread has been asked to terminate, but is still running!",
        mtInformation, TMsgDlgButtons() << mbOK, 0);
    Sleep(3000);
  }
  youthreadRunning = FALSE;
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  if (listthreadRunning == FALSE)
  {
    listthreadRunning = TRUE;
    listProcess = new TListThread(True); // create suspended – secondprocess does not run yet
    listProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    listProcess->Priority = tpLower;  // set the priority to lower than normal
    listProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TListThread::Execute()
{
  while(True)
  {
    if (Terminated) break;
    Form1->ListBox1->Clear();
    TList *myList = threadList1->LockList();
    try
    {
      for (int I = 0; I < myList->Count; I++)
```

```
      {
        TControl *Temp = (TControl *)myList->Items[I];
        Form1->ListBox1->Items->Add(Temp->Name);
      }
    }
    __finally
    {
      threadList1->UnlockList();
    }
    Sleep(1000);
  }
}

void __fastcall TForm1::Button4Click(TObject *Sender)
{
  if (listProcess != NULL)
    listProcess->Terminate();
  if (secondProcess != NULL)
    secondProcess->Terminate();
  if (otherSecondProcess != NULL)
    otherSecondProcess->Terminate();
}
```

Delphi Examples:

```
{
The following example shows how to use the LockList and
UnlockList methods to access a thread-safe version  of a
TList object:
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button3: TButton;
    ListBox1: TListBox;
    Button2: TButton;
    Button4: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  TListThread = class(TThread)
  protected
    procedure Execute; override;
  end;
  TMyThread = class(TThread)
  protected
    procedure Execute; override;
  end;
  TYouThread = class(TThread)
  protected
    procedure Execute; override;
  end;

var
  Form1: TForm1;
  threadList1: TThreadList;
  mythreadRunning, youthreadRunning, listThreadRunning: Boolean;
  globalCount: Integer;
  listProcess: TListThread; { TListThread is a custom descendant of TThread }
```

```
    secondProcess: TMyThread; { TMyThread is a custom descendant of TThread }
    otherSecondProcess: TYouThread; { TMyThread is a custom descendant of TThread }

  implementation

  {$R *.dfm}

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    if (mythreadRunning = FALSE) then
    begin
      mythreadRunning:= TRUE;
      secondProcess := TMyThread.Create(True); { create suspended – secondprocess does not run
  yet }
      secondProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
      secondProcess.Priority := tpLower;  // set the priority to lower than normal
      secondProcess.Resume; { now run the thread }
    end
    else
      MessageDlg('This thread is still running.  You are going to hurt yourself!',
        mtInformation, [mbOk], 0);
  end;

  procedure TMyThread.Execute;
  var
    I: Integer;
    myRadio: TRadioButton;
  begin
    for I := 0 to 20 do
    begin
      if (Terminated) then
      begin
        mythreadRunning:= FALSE;
        exit;
      end;
      myRadio:= TRadioButton.Create(Form1);
      globalCount:= globalCount + 1;
      myRadio.Name:= 'RadioButton' + IntToStr(globalCount);
      threadList1.Add(myRadio);
      Sleep(1000);
    end;
    mythreadRunning:= FALSE;
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    if (listthreadRunning = FALSE) then
    begin
      listThreadRunning:= TRUE;
      listProcess := TListThread.Create(True); { create suspended – secondprocess does not run
  yet }
      listProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
      listProcess.Priority := tpLower;  // set the priority to lower than normal
      listProcess.Resume; { now run the thread }
    end;
  end;

  procedure TListThread.Execute;
  var
    I: Integer;
    Temp: TControl;
    myList: TList;
  begin
    while(True) do
    begin
      if (Terminated) then
```

```
      begin
        listthreadRunning:= FALSE;
        exit;
      end;
      Form1.ListBox1.Clear;
      myList:= threadList1.LockList;
      try
        for I := 0 to myList.Count-1 do
        begin
          Temp:= myList.Items[I];
          Form1.ListBox1.Items.Add(Temp.Name);
        end;
      finally
        threadList1.UnlockList;
      end;
      Sleep(1000);
    end;
    listthreadRunning:= FALSE;
  end;

  procedure TForm1.Button3Click(Sender: TObject);
  begin
    if (youthreadRunning = FALSE) then
    begin
      youthreadRunning:= TRUE;
      otherSecondProcess := TYouThread.Create(True); { create suspended – secondprocess does not
run yet }
      otherSecondProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
      otherSecondProcess.Priority := tpLower;  // set the priority to lower than normal
      otherSecondProcess.Resume; { now run the thread }
    end
    else
      MessageDlg('This thread is still running.  You are going to hurt yourself!',
        mtInformation, [mbOk], 0);
  end;

  procedure TForm1.Button4Click(Sender: TObject);
  begin
    if (listProcess <> nil) then
      listProcess.Terminate;
    if (secondProcess <> nil) then
      secondProcess.Terminate;
    if (otherSecondProcess <> nil) then
      otherSecondProcess.Terminate;
  end;

  procedure TYouThread.Execute;
  var
    I: Integer;
    Temp: TControl;
  begin
    for I := 0 to 10 do
    begin
      if (Terminated) then
      begin
        youThreadRunning:= FALSE;
        exit;
      end;
      with threadList1.LockList do
      try
        if (2*I < Count) then
        begin
          Temp:= Items[2*I];
          threadList1.Remove(Temp);
        end;
      finally
```

```
      threadList1.UnlockList;
    end;
    if (Terminated) then
      MessageDlg('youThread has been asked to terminate, but is still running!',
        mtInformation, [mbOk], 0);
    Sleep(3000);
  end;
  youthreadRunning:= FALSE;
end;
```

See Also

TList (see page 2868)

UnlockList (see page 3169)

Locking Objects

23.1.53.4.4 TThreadList.Remove Method

Removes an item from the list.

Pascal

```
procedure Remove(Item: Pointer);
```

C++

```
__fastcall Remove(void * Item);
```

Description

Call Remove to remove a specific item from the list without having to determine its index. Remove locks the list, calls the Remove method in TList (see page 2868), then unlocks the list.

C++ Examples:

```
/*
The following example shows how to use the LockList and
UnlockList methods to access a thread-safe version  of a
TList object:
*/
class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
}

class TYouThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};
```

```
__fastcall TYouThread::TYouThread(bool suspended)
    : TThread(suspended)
{
}

class TListThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TListThread(bool suspended);
};

__fastcall TListThread::TListThread(bool suspended)
    : TThread(suspended)
{
}

TThreadList *threadList1;
bool mythreadRunning, youthreadRunning, listthreadRunning;
int globalCount;
TMyThread *secondProcess;
TYouThread *otherSecondProcess;
TListThread *listProcess; // thread arecustom descendants of TThread

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    secondProcess = new TMyThread(True); // create suspended – secondprocess does not run yet
    secondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    secondProcess->Priority = tpLower;  // set the priority to lower than normal
    secondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TMyThread::Execute()
{
  for (int I = 0; I <= 20; I++)
  {
    if (Terminated) break;
    TRadioButton *myRadio = new TRadioButton(Form1);
    globalCount = globalCount + 1;
    myRadio->Name = "RadioButton" + IntToStr(globalCount);
    threadList1->Add(myRadio);
    Sleep(1000);
  }
  mythreadRunning = FALSE;
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    otherSecondProcess = new TYouThread(True); // create suspended – other secondprocess does
not run yet
    otherSecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    otherSecondProcess->Priority = tpLower;  // set the priority to lower than normal
    otherSecondProcess->Resume(); // now run the thread
```

```
    }
    else
      MessageDlg("This thread is still running.  You are going to hurt yourself!",
        mtInformation, TMsgDlgButtons() << mbOK, 0);
  }

  void __fastcall TYouThread::Execute()
  {
    for (int I = 0; I <= 10; I++)
    {
      if (Terminated) break;
      try
      {
        TList *list = threadList1->LockList();
        if (2*I < list->Count)
        {
          TControl *Temp = (TControl *)list->Items[2*I];
          threadList1->Remove(Temp);
        }
      }
      __finally
      {
        threadList1->UnlockList();
      }
      if (Terminated)
        MessageDlg("youThread has been asked to terminate, but is still running!",
          mtInformation, TMsgDlgButtons() << mbOK, 0);
      Sleep(3000);
    }
    youthreadRunning = FALSE;
  }

  void __fastcall TForm1::Button2Click(TObject *Sender)
  {
    if (listthreadRunning == FALSE)
    {
      listthreadRunning = TRUE;
      listProcess = new TListThread(True); // create suspended – secondprocess does not run yet
      listProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
      listProcess->Priority = tpLower;  // set the priority to lower than normal
      listProcess->Resume(); // now run the thread
    }
    else
      MessageDlg("This thread is still running.  You are going to hurt yourself!",
        mtInformation, TMsgDlgButtons() << mbOK, 0);
  }

  void __fastcall TListThread::Execute()
  {
    while(True)
    {
      if (Terminated) break;
      Form1->ListBox1->Clear();
      TList *myList = threadList1->LockList();
      try
      {
        for (int I = 0; I < myList->Count; I++)
        {
          TControl *Temp = (TControl *)myList->Items[I];
          Form1->ListBox1->Items->Add(Temp->Name);
        }
      }
      __finally
      {
        threadList1->UnlockList();
      }
```

```
    Sleep(1000);
  }
}

void __fastcall TForm1::Button4Click(TObject *Sender)
{
  if (listProcess != NULL)
    listProcess->Terminate();
  if (secondProcess != NULL)
    secondProcess->Terminate();
  if (otherSecondProcess != NULL)
    otherSecondProcess->Terminate();
}
```

Delphi Examples:

```
{
The following example shows how to use the LockList and
UnlockList methods to access a thread-safe version  of a
TList object:
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button3: TButton;
    ListBox1: TListBox;
    Button2: TButton;
    Button4: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  TListThread = class(TThread)
  protected
    procedure Execute; override;
  end;
  TMyThread = class(TThread)
  protected
    procedure Execute; override;
  end;
  TYouThread = class(TThread)
  protected
    procedure Execute; override;
  end;

var
  Form1: TForm1;
  threadList1: TThreadList;
  mythreadRunning, youthreadRunning, listThreadRunning: Boolean;
  globalCount: Integer;
  listProcess: TListThread; { TListThread is a custom descendant of TThread }
  secondProcess: TMyThread; { TMyThread is a custom descendant of TThread }
  otherSecondProcess: TYouThread; { TMyThread is a custom descendant of TThread }

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
  if (mythreadRunning = FALSE) then
  begin
    mythreadRunning:= TRUE;
    secondProcess := TMyThread.Create(True); { create suspended – secondprocess does not run
yet }
    secondProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    secondProcess.Priority := tpLower;  // set the priority to lower than normal
    secondProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TMyThread.Execute;
var
  I: Integer;
  myRadio: TRadioButton;
begin
  for I := 0 to 20 do
  begin
    if (Terminated) then
    begin
      mythreadRunning:= FALSE;
      exit;
    end;
    myRadio:= TRadioButton.Create(Form1);
    globalCount:= globalCount + 1;
    myRadio.Name:= 'RadioButton' + IntToStr(globalCount);
    threadList1.Add(myRadio);
    Sleep(1000);
  end;
  mythreadRunning:= FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  if (listthreadRunning = FALSE) then
  begin
    listThreadRunning:= TRUE;
    listProcess := TListThread.Create(True); { create suspended – secondprocess does not run
yet }
    listProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    listProcess.Priority := tpLower;  // set the priority to lower than normal
    listProcess.Resume; { now run the thread }
  end;
end;

procedure TListThread.Execute;
var
  I: Integer;
  Temp: TControl;
  myList: TList;
begin
  while(True) do
  begin
    if (Terminated) then
    begin
      listthreadRunning:= FALSE;
      exit;
    end;
    Form1.ListBox1.Clear;
    myList:= threadList1.LockList;
    try
      for I := 0 to myList.Count-1 do
      begin
```

```
          Temp:= myList.Items[I];
          Form1.ListBox1.Items.Add(Temp.Name);
        end;
      finally
        threadList1.UnlockList;
      end;
      Sleep(1000);
    end;
    listthreadRunning:= FALSE;
  end;

  procedure TForm1.Button3Click(Sender: TObject);
  begin
    if (youthreadRunning = FALSE) then
    begin
      youthreadRunning:= TRUE;
      otherSecondProcess := TYouThread.Create(True); { create suspended – secondprocess does not
run yet }
      otherSecondProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
      otherSecondProcess.Priority := tpLower;  // set the priority to lower than normal
      otherSecondProcess.Resume; { now run the thread }
    end
    else
      MessageDlg('This thread is still running.  You are going to hurt yourself!',
        mtInformation, [mbOk], 0);
  end;

  procedure TForm1.Button4Click(Sender: TObject);
  begin
    if (listProcess <> nil) then
      listProcess.Terminate;
    if (secondProcess <> nil) then
      secondProcess.Terminate;
    if (otherSecondProcess <> nil) then
      otherSecondProcess.Terminate;
  end;

  procedure TYouThread.Execute;
  var
    I: Integer;
    Temp: TControl;
  begin
    for I := 0 to 10 do
    begin
      if (Terminated) then
      begin
        youThreadRunning:= FALSE;
        exit;
      end;
      with threadList1.LockList do
      try
        if (2*I < Count) then
        begin
          Temp:= Items[2*I];
          threadList1.Remove(Temp);
        end;
      finally
        threadList1.UnlockList;
      end;
      if (Terminated) then
        MessageDlg('youThread has been asked to terminate, but is still running!',
          mtInformation, [mbOk], 0);
      Sleep(3000);
    end;
    youthreadRunning:= FALSE;
  end;
```

See Also

Remove (see page 2887)

23.1.53.4.5 TThreadList.UnlockList Method

Unlocks the thread-safe list.

Pascal

```
procedure UnlockList;
```

C++

```
__fastcall UnlockList();
```

Description

UnlockList unlocks the list.

C++ Examples:

```
/*
The following example shows how to use the LockList and
UnlockList methods to access a thread-safe version  of a
TList object:
*/
class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
}

class TYouThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};

__fastcall TYouThread::TYouThread(bool suspended)
    : TThread(suspended)
{
}

class TListThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TListThread(bool suspended);
};
```

```
__fastcall TListThread::TListThread(bool suspended)
    : TThread(suspended)
{
}

TThreadList *threadList1;
bool mythreadRunning, youthreadRunning, listthreadRunning;
int globalCount;
TMyThread *secondProcess;
TYouThread *otherSecondProcess;
TListThread *listProcess; // thread arecustom descendants of TThread

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    secondProcess = new TMyThread(True); // create suspended – secondprocess does not run yet
    secondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    secondProcess->Priority = tpLower;  // set the priority to lower than normal
    secondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TMyThread::Execute()
{
  for (int I = 0; I <= 20; I++)
  {
    if (Terminated) break;
    TRadioButton *myRadio = new TRadioButton(Form1);
    globalCount = globalCount + 1;
    myRadio->Name = "RadioButton" + IntToStr(globalCount);
    threadList1->Add(myRadio);
    Sleep(1000);
  }
  mythreadRunning = FALSE;
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    otherSecondProcess = new TYouThread(True); // create suspended – other secondprocess does
not run yet
    otherSecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    otherSecondProcess->Priority = tpLower;  // set the priority to lower than normal
    otherSecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TYouThread::Execute()
{
  for (int I = 0; I <= 10; I++)
  {
    if (Terminated) break;
    try
    {
      TList *list = threadList1->LockList();
```

```
      if (2*I < list->Count)
      {
        TControl *Temp = (TControl *)list->Items[2*I];
        threadList1->Remove(Temp);
      }
    }
    __finally
    {
      threadList1->UnlockList();
    }
    if (Terminated)
      MessageDlg("youThread has been asked to terminate, but is still running!",
        mtInformation, TMsgDlgButtons() << mbOK, 0);
    Sleep(3000);
  }
  youthreadRunning = FALSE;
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  if (listthreadRunning == FALSE)
  {
    listthreadRunning = TRUE;
    listProcess = new TListThread(True); // create suspended – secondprocess does not run yet
    listProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    listProcess->Priority = tpLower;  // set the priority to lower than normal
    listProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TListThread::Execute()
{
  while(True)
  {
    if (Terminated) break;
    Form1->ListBox1->Clear();
    TList *myList = threadList1->LockList();
    try
    {
      for (int I = 0; I < myList->Count; I++)
      {
        TControl *Temp = (TControl *)myList->Items[I];
        Form1->ListBox1->Items->Add(Temp->Name);
      }
    }
    __finally
    {
      threadList1->UnlockList();
    }
    Sleep(1000);
  }
}

void __fastcall TForm1::Button4Click(TObject *Sender)
{
  if (listProcess != NULL)
    listProcess->Terminate();
  if (secondProcess != NULL)
    secondProcess->Terminate();
  if (otherSecondProcess != NULL)
    otherSecondProcess->Terminate();
}
```

Delphi Examples:

```
{
The following example shows how to use the LockList and
UnlockList methods to access a thread-safe version  of a
TList object:
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button3: TButton;
    ListBox1: TListBox;
    Button2: TButton;
    Button4: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  TListThread = class(TThread)
  protected
    procedure Execute; override;
  end;
  TMyThread = class(TThread)
  protected
    procedure Execute; override;
  end;
  TYouThread = class(TThread)
  protected
    procedure Execute; override;
  end;

var
  Form1: TForm1;
  threadList1: TThreadList;
  mythreadRunning, youthreadRunning, listThreadRunning: Boolean;
  globalCount: Integer;
  listProcess: TListThread; { TListThread is a custom descendant of TThread }
  secondProcess: TMyThread; { TMyThread is a custom descendant of TThread }
  otherSecondProcess: TYouThread; { TMyThread is a custom descendant of TThread }

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
  if (mythreadRunning = FALSE) then
  begin
    mythreadRunning:= TRUE;
    secondProcess := TMyThread.Create(True); { create suspended – secondprocess does not run
yet }
    secondProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    secondProcess.Priority := tpLower;  // set the priority to lower than normal
    secondProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TMyThread.Execute;
```

```
var
  I: Integer;
  myRadio: TRadioButton;
begin
  for I := 0 to 20 do
  begin
    if (Terminated) then
    begin
      mythreadRunning:= FALSE;
      exit;
    end;
    myRadio:= TRadioButton.Create(Form1);
    globalCount:= globalCount + 1;
    myRadio.Name:= 'RadioButton' + IntToStr(globalCount);
    threadList1.Add(myRadio);
    Sleep(1000);
  end;
  mythreadRunning:= FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  if (listthreadRunning = FALSE) then
  begin
    listThreadRunning:= TRUE;
    listProcess := TListThread.Create(True); { create suspended – secondprocess does not run
yet }
    listProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    listProcess.Priority := tpLower;  // set the priority to lower than normal
    listProcess.Resume; { now run the thread }
  end;
end;

procedure TListThread.Execute;
var
  I: Integer;
  Temp: TControl;
  myList: TList;
begin
  while(True) do
  begin
    if (Terminated) then
    begin
      listthreadRunning:= FALSE;
      exit;
    end;
    Form1.ListBox1.Clear;
    myList:= threadList1.LockList;
    try
      for I := 0 to myList.Count-1 do
      begin
        Temp:= myList.Items[I];
        Form1.ListBox1.Items.Add(Temp.Name);
      end;
    finally
      threadList1.UnlockList;
    end;
    Sleep(1000);
  end;
  listthreadRunning:= FALSE;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  if (youthreadRunning = FALSE) then
  begin
```

```
      youthreadRunning:= TRUE;
      otherSecondProcess := TYouThread.Create(True); { create suspended – secondprocess does not
run yet }
      otherSecondProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
      otherSecondProcess.Priority := tpLower;  // set the priority to lower than normal
      otherSecondProcess.Resume; { now run the thread }
    end
    else
      MessageDlg('This thread is still running.  You are going to hurt yourself!',
        mtInformation, [mbOk], 0);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  if (listProcess <> nil) then
    listProcess.Terminate;
  if (secondProcess <> nil) then
    secondProcess.Terminate;
  if (otherSecondProcess <> nil) then
    otherSecondProcess.Terminate;
end;

procedure TYouThread.Execute;
var
  I: Integer;
  Temp: TControl;
begin
  for I := 0 to 10 do
  begin
    if (Terminated) then
    begin
      youThreadRunning:= FALSE;
      exit;
    end;
    with threadList1.LockList do
    try
      if (2*I < Count) then
      begin
        Temp:= Items[2*I];
        threadList1.Remove(Temp);
      end;
    finally
      threadList1.UnlockList;
    end;
    if (Terminated) then
      MessageDlg('youThread has been asked to terminate, but is still running!',
        mtInformation, [mbOk], 0);
    Sleep(3000);
  end;
  youthreadRunning:= FALSE;
end;
```

See Also

LockList (see page 3157)

Locking Objects

23.1.53.5 TThreadList Properties

The properties of the TThreadList class are listed here.

Public Properties

	Name	Description
	Duplicates (see page 3175)	Indicates whether the thread list allows duplicate entries.

23.1.53.5.1 TThreadList.Duplicates Property

Indicates whether the thread list allows duplicate entries.

Pascal

```
property Duplicates: TDuplicates;
```

C++

```
__property TDuplicates Duplicates;
```

Description

Set Duplicates to specify what should happen when an attempt is made to add a duplicate item to the list. The value of Duplicates should be one of the following.

Value	Meaning
dupIgnore	Ignore attempts to add duplicate items to the list.
dupError	raise an EListError (see page 2701) exception when an attempt is made to add duplicate items to the list.
dupAccept	Permit duplicate items in the list.

Set Duplicates before adding any items to the list. Setting Duplicates to dupIgnore or dupError does nothing about duplicates that are already in the list.

Note: Setting duplicates to dupIgnore (the default) or dupError can be computationally expensive if the list has many items.

See Also

Clear (see page 2874)

23.1.54 TWriter Class

TWriter is a specialized filer object that writes data to its associated stream.

Class Hierarchy

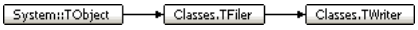

Pascal

```
TWriter = class(TFiler);
```

C++

```
class TWriter : public TFiler;
```

File

Classes

Description

TWriter is used internally by the component streaming system to write information associated with a component, such as

published properties of components or custom property data, to a stream. TWriter handles the mechanics of writing the data associated with a component to a stream. It is the writer object, rather than the stream, that is responsible for handling the complexities of streaming components. These include methods for:

Writing different kinds of items to the associated stream.

Writing property deltas used for streaming inherited forms and properties with default values.

Writing nested groups of items or collections to the stream.

Other methods and properties of TWriter are used for interacting with stream and component objects.

Do not directly create writer objects. Writers are automatically created in stream object methods or in global routines that initiate the streaming process. These include:

the global routine ObjectTextToBinary (see page 3240) procedure, which directly creates a writer.

the global WriteComponentResFile (see page 3266) function, which creates a file stream that creates a writer.

the WriteDescendent (see page 3184) method of TStream (see page 2973), which creates a writer object.

Once the streaming process is underway, programs do not need to directly manipulate writer objects. The interaction between the writer, component, and stream objects happens automatically in methods of these objects that make calls to each other.

See Also

ObjectTextToBinary (see page 3240)

TComponent (see page 2762)

TReader (see page 2933)

TStream (see page 2973)

TStream (see page 2973)

WriteComponentResFile (see page 3266)

Using Streams to Read or Write Data

23.1.54.1 TWriter Members

The following tables list the members exposed by TWriter.

Protected Methods

	Name	Description
	FindMethodName (see page 3181)	This is FindMethodName, a member of class TWriter.
	SetRoot (see page 3181)	This is SetRoot, a member of class TWriter.
	WriteBinary (see page 3182)	This is WriteBinary, a member of class TWriter.
	WritePrefix (see page 3187)	This is WritePrefix, a member of class TWriter.
	WriteProperties (see page 3188)	This is WriteProperties, a member of class TWriter.
	WriteProperty (see page 3188)	This is WriteProperty, a member of class TWriter.
	WritePropName (see page 3188)	This is WritePropName, a member of class TWriter.
	WriteValue (see page 3190)	This is WriteValue, a member of class TWriter.

Public Events

	Name	Description
	OnFindAncestor (see page 3193)	Occurs when a component is about to be written to a stream.
	OnFindMethodName (see page 3194)	This is OnFindMethodName, a member of class TWriter.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2836)	Creates a filer object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DefineBinaryProperty (see page 3179)	Treats unformatted data as if it were the value of a published property, and writes it directly to a memory stream.
	DefineProperty (see page 3180)	Defines data the writer object writes as if the data were a property.
	Destroy (see page 3178)	Destroys of an instance of a writer.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FlushBuffer (see page 3181)	Synchronizes the writer's buffer with the associated stream by writing the current buffer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Write (see page 3181)	Writes Count bytes from Buf to the writer object's associated stream.
	WriteBoolean (see page 3182)	Writes the Boolean value passed in Value to the writer object's stream.
	WriteChar (see page 3182)	Writes the character passed in Value to the writer object's stream.
	WriteCollection (see page 3183)	Writes the collection of objects passed in Value to the writer object's stream.
	WriteComponent (see page 3183)	Writes the component specified by Component to a stream.
	WriteCurrency (see page 3184)	Writes a currency value to the writer's stream.
	WriteDate (see page 3184)	Writes a datetime value to the writer's stream.
	WriteDescendent (see page 3184)	Triggers the writing of descendant components in inherited forms.
	WriteDouble (see page 3185)	Writes a value of type Double to the writer object’s stream.
	WriteFloat (see page 3185)	Writes the floating-point value passed in Value to the writer object's stream.
	WriteIdent (see page 3186)	Writes the identifier passed in Ident to the writer object's stream.
	WriteInteger (see page 3186)	This is the overview for the WriteInteger method overload.
	WriteListBegin (see page 3187)	Writes a start-of-list marker to the writer object's associated stream.
	WriteListEnd (see page 3187)	Writes an end-of-list marker to the writer object's associated stream.
	WriteRootComponent (see page 3188)	Calls the WriteDescendent (see page 3184) method.
	WriteSignature (see page 3189)	Writes a filer signature to the writer object's associated stream.

	WriteSingle (see page 3189)	Writes a value of type Single (Delphi) or float (C++) to the writer's stream.
	WriteStr (see page 3189)	Writes raw data to the writer object's stream.
	WriteString (see page 3190)	Writes tagged data to the writer object's stream.
	WriteVariant (see page 3190)	Writes a Variant value to the writer's stream.
	WriteWideChar (see page 3191)	Writes the wide character passed in Value to the writer object's stream.
	WriteWideString (see page 3191)	WriteWideString writes a wide string to the writer's stream.

Public Properties

	Name	Description
	Ancestor (see page 2839)	Determines whether to write the properties of inherited components.
	IgnoreChildren (see page 2839)	Enables a writer object to save a component without saving the components it owns.
R	LookupRoot (see page 2840)	Specifies the root component for resolving names while loading and saving components.
	Position (see page 3192)	Represents the current writing position in the associated stream.
	Root (see page 2840)	Indicates which object is the root, or ultimate owner, of the objects read or written by the filer.
	RootAncestor (see page 3192)	Represents the ancestor of the component in the Root (see page 2840) property.
	UseQualifiedNames (see page 3193)	Indicates whether all class names should be qualified by the name of the unit in which they are defined.

23.1.54.2 TWriter.Destroy Destructor

Destroys of an instance of a writer.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWriter();
```

Description

Do not call Destroy directly in an application. Writers are created indirectly by the global routines or stream methods that use them. When these routines are finished, they handle disposing of the writer.

Destroy writes the buffer before calling the inherited Destroy, which frees the buffer.

See Also

Destroy (see page 2837)

Free (see page 22300)

23.1.54.3 TWriter Methods

The methods of the TWriter class are listed here.

Protected Methods

	Name	Description
V	FindMethodName (see page 3181)	This is FindMethodName, a member of class TWriter.
V	SetRoot (see page 3181)	This is SetRoot, a member of class TWriter.
	WriteBinary (see page 3182)	This is WriteBinary, a member of class TWriter.
	WritePrefix (see page 3187)	This is WritePrefix, a member of class TWriter.
	WriteProperties (see page 3188)	This is WriteProperties, a member of class TWriter.
	WriteProperty (see page 3188)	This is WriteProperty, a member of class TWriter.

	WritePropName (see page 3188)	This is WritePropName, a member of class TWriter.
	WriteValue (see page 3190)	This is WriteValue, a member of class TWriter.

Public Methods

	Name	Description
	DefineBinaryProperty (see page 3179)	Treats unformatted data as if it were the value of a published property, and writes it directly to a memory stream.
	DefineProperty (see page 3180)	Defines data the writer object writes as if the data were a property.
	FlushBuffer (see page 3181)	Synchronizes the writer's buffer with the associated stream by writing the current buffer.
	Write (see page 3181)	Writes Count bytes from Buf to the writer object's associated stream.
	WriteBoolean (see page 3182)	Writes the Boolean value passed in Value to the writer object's stream.
	WriteChar (see page 3182)	Writes the character passed in Value to the writer object's stream.
	WriteCollection (see page 3183)	Writes the collection of objects passed in Value to the writer object's stream.
	WriteComponent (see page 3183)	Writes the component specified by Component to a stream.
	WriteCurrency (see page 3184)	Writes a currency value to the writer's stream.
	WriteDate (see page 3184)	Writes a datetime value to the writer's stream.
	WriteDescendent (see page 3184)	Triggers the writing of descendant components in inherited forms.
	WriteDouble (see page 3185)	Writes a value of type Double to the writer object's stream.
	WriteFloat (see page 3185)	Writes the floating-point value passed in Value to the writer object's stream.
	WriteIdent (see page 3186)	Writes the identifier passed in Ident to the writer object's stream.
	WriteInteger (see page 3186)	This is the overview for the WriteInteger method overload.
	WriteListBegin (see page 3187)	Writes a start-of-list marker to the writer object's associated stream.
	WriteListEnd (see page 3187)	Writes an end-of-list marker to the writer object's associated stream.
	WriteRootComponent (see page 3188)	Calls the WriteDescendent (see page 3184) method.
	WriteSignature (see page 3189)	Writes a filer signature to the writer object's associated stream.
	WriteSingle (see page 3189)	Writes a value of type Single (Delphi) or float (C++) to the writer's stream.
	WriteStr (see page 3189)	Writes raw data to the writer object's stream.
	WriteString (see page 3190)	Writes tagged data to the writer object's stream.
	WriteVariant (see page 3190)	Writes a Variant value to the writer's stream.
	WriteWideChar (see page 3191)	Writes the wide character passed in Value to the writer object's stream.
	WriteWideString (see page 3191)	WriteWideString writes a wide string to the writer's stream.

23.1.54.3.1 TWriter.DefineBinaryProperty Method

Treats unformatted data as if it were the value of a published property, and writes it directly to a memory stream.

Pascal

```
procedure DefineBinaryProperty(const Name: string; ReadData: TStreamProc; WriteData:
TStreamProc; HasData: Boolean); override;
```

C++

```
virtual __fastcall DefineBinaryProperty(const AnsiString Name, TStreamProc ReadData,
TStreamProc WriteData, Boolean HasData);
```

Description

DefineBinaryProperty is called internally by the DefineProperties method of an object that has data it needs to store. DefineProperties takes a generic filer object as its parameter.

For writing binary data DefineProperties takes a TWriter (see page 3175) object and then calls its DefineBinaryProperty method. DefineBinaryProperty writes the property's name and it's data, but only if the HasData parameter is true. Otherwise it does nothing.

The Name parameter specifies the name of the "fake" property to be written to the stream. A "fake" property is a property that is not published, and that exists only in the code for the DefineBinaryProperty method. The Top and Left properties of a non-visual component are examples of "fake" properties. These are also called "defined properties" or "custom defined properties."

The TStreamProc (see page 3304) type is the type of the ReadData and WriteData parameters. It is the method-pointer type that points to a procedure (defined in the storing object) that reads or writes a binary representation of the object's data directly to or from the stream passed as its Stream parameter. For TWriter (see page 3175) the ReadData parameter is ignored.

The HasData parameter determines at run time whether the "fake" property has data to store (write).

Defined binary properties are quite rare. Persistent objects that store graphics as binary data are the most common use of filer objects to store and retrieve that data. More commonly, objects use the DefineProperty (see page 3180) method. The difference between DefineBinaryProperty and DefineProperty (see page 3180) is that with DefineBinaryProperty the component reads or writes the binary property directly to or from a memory stream, rather than going through a filer object.

Note: Streamable objects that are descended from TPersistent (see page 2928) inherit a DefineProperties method, however DefineProperties does not do anything until TComponent (see page 2762). It is TComponent (see page 2762)'s DefineProperties method that calls the writer's DefineBinaryProperty when writing out binary data.

See Also

DefineProperty (see page 3180)

DefineProperties (see page 2771)

23.1.54.3.2 TWriter.DefineProperty Method

Defines data the writer object writes as if the data were a property.

Pascal

```
procedure DefineProperty(const Name: string; ReadData: TReaderProc; WriteData: TWriterProc;
HasData: Boolean); override;
```

C++

```
virtual __fastcall DefineProperty(const AnsiString Name, TReaderProc ReadData, TWriterProc
WriteData, Boolean HasData);
```

Description

DefineProperty is called internally by the DefineProperties method of an object that has data it needs to store. DefineProperties takes a generic filer object as its parameter. For writing data, DefineProperties takes a TWriter (see page 3175) object and calls its DefineProperty method. DefineProperty then writes the property's name and its data, but only if the HasData parameter is true. Otherwise it does nothing.

The Name parameter specifies the name of the "fake" property to be written to the stream.

The WriteData parameter points to a procedure (defined in the component object) that writes the object's data, which represents a property value, to the writer object. For TWriter (see page 3175) the ReadData parameter is ignored.

The HasData parameter determines at run time whether the "fake" property has data to store (write). Thus, writer objects only use HasData when writing data.

The difference between DefineBinaryProperty (see page 3179) and DefineProperty is that with DefineBinaryProperty (see page 3179), the binary data is written directly to a stream object, rather than going through a filer object.

When writing a component that has special (large) data storage requirements, the component's DefineProperties can be overridden. For each special or "fake" property item, call DefineProperty or DefineBinaryProperty (see page 3179). For the ReadData and WriteData parameters, pass in methods of the component that know how to handle that special data type. When reading, ReadData is called. When writing, WriteData is called.

Note: When defining properties, a component should be aware of the Ancestor (see page 2839) property, which, if non-nil (Delphi) or non-NULL (C++) indicates that the component should only write the values of properties that differ from those inherited from Ancestor (see page 2839).

See Also

Ancestor (see page 2839)

DefineBinaryProperty (see page 3179)

DefineProperties (see page 2771)

23.1.54.3.3 TWriter.FindMethodName Method

Pascal

```
function FindMethodName(Method: TMethod): string; virtual;
```

C++

```
virtual __fastcall AnsiString FindMethodName(TMethod Method);
```

Description

This is FindMethodName, a member of class TWriter.

23.1.54.3.4 TWriter.FlushBuffer Method

Synchronizes the writer's buffer with the associated stream by writing the current buffer.

Pascal

```
procedure FlushBuffer; override;
```

C++

```
virtual __fastcall FlushBuffer();
```

Description

FlushBuffer is not generally used except for internal implementation in the IDE.

FlushBuffer writes the current buffer to the stream, then resets an internal pointer to the correct position.

23.1.54.3.5 TWriter.SetRoot Method

Pascal

```
procedure SetRoot(Value: TComponent); override;
```

C++

```
virtual __fastcall SetRoot(TComponent * Value);
```

Description

This is SetRoot, a member of class TWriter.

23.1.54.3.6 TWriter.Write Method

Writes Count bytes from Buf to the writer object's associated stream.

Pascal

```
procedure Write(const Buf; Count: Longint);
```

C++

```
__fastcall Write(const  Buf, Longint Count);
```

Description

Do not call Write directly. It is used internally to write data to the stream. Many other writer methods call Write, usually after setting pertinent values or verifying data types.

See Also

ObjectBinaryToText

23.1.54.3.7 TWriter.WriteBinary Method

Pascal

```
procedure WriteBinary(WriteData: TStreamProc);
```

C++

```
__fastcall WriteBinary(TStreamProc WriteData);
```

Description

This is WriteBinary, a member of class TWriter.

23.1.54.3.8 TWriter.WriteBoolean Method

Writes the Boolean value passed in Value to the writer object's stream.

Pascal

```
procedure WriteBoolean(Value: Boolean);
```

C++

```
__fastcall WriteBoolean(Boolean Value);
```

Description

WriteBoolean is used internally by the component streaming system to write component properties to a stream. Use WriteBoolean for writing component boolean properties to streams.

WriteBoolean checks to ensure that Value is the correct type before calling Write (see page 3181) to write the boolean value and its value type to the stream.

See Also

Write (see page 3181)

23.1.54.3.9 TWriter.WriteChar Method

Writes the character passed in Value to the writer object's stream.

Pascal

```
procedure WriteChar(Value: Char);
```

C++

```
__fastcall WriteChar(Char Value);
```

Description

WriteChar is used internally by the component streaming system to write component properties to a stream. Use WriteChar for writing component character properties to streams.

WriteChar calls WriteString (see page 3190) which checks to ensure that Value is the correct type before calling Write (see

page 3181) to write the character and its value type to the stream.

Note: WriteChar calls StringChar to enable writing character properties that are upwardly compatible with string properties.

See Also

Write (see page 3181)

WriteString (see page 3190)

23.1.54.3.10 TWriter.WriteCollection Method

Writes the collection of objects passed in Value to the writer object's stream.

Pascal

```
procedure WriteCollection(Value: TCollection);
```

C++

```
__fastcall WriteCollection(TCollection Value);
```

Description

WriteCollection is used internally by the component streaming system to write component properties to a stream. Use WriteCollection for writing component properties that are collections to streams.

WriteCollection first writes the type identifier indicating a collection.

If the Value parameter is not nil (Delphi) or NULL (C++), WriteCollection next writes each element of the collection to the stream, bounding the elements with start-of-list and end-of-list markers. Otherwise, it skips the elements, assuming an empty collection.

See Also

Write (see page 3181)

WriteListBegin (see page 3187)

WriteListEnd (see page 3187)

23.1.54.3.11 TWriter.WriteComponent Method

Writes the component specified by Component to a stream.

Pascal

```
procedure WriteComponent(Component: TComponent);
```

C++

```
__fastcall WriteComponent(TComponent * Component);
```

Description

Never call WriteComponent directly. WriteComponent is recursively called for each owned component in Root (see page 2840).

WriteComponent sets the csWriting state in Component's ComponentState property before calling the WriteState method of Component and clears the csWriting flag when WriteState returns.

Note: The sequence of events is as follows: the writer's WriteComponent method calls the Component's WriteState method, which calls the writer's WriteData method. That method writes the properties for each child component, then calls WriteComponent.

Because the Component's WriteState is virtual, it is the Component's only opportunity to prepare itself to be streamed (i.e. to

consolidate its data.)

See Also

WriteDescendent (see page 3184)

UseQualifiedNames (see page 3193)

23.1.54.3.12 TWriter.WriteCurrency Method

Writes a currency value to the writer's stream.

Pascal

```
procedure WriteCurrency(const Value: Currency);
```

C++

```
__fastcall WriteCurrency(const Currency Value);
```

Description

WriteCurrency is used internally by the component streaming system to write component properties to a stream. Use WriteCurrency for writing component properties of type Currency to streams.

WriteCurrency first writes a type identifier before writing the currency value.

23.1.54.3.13 TWriter.WriteDate Method

Writes a datetime value to the writer's stream.

Pascal

```
procedure WriteDate(const Value: TDateTime);
```

C++

```
__fastcall WriteDate(const TDateTime Value);
```

Description

WriteDate is used internally by the component streaming system to write component properties to a stream. Use WriteDate for writing component properties of type TDateTime to streams.

WriteDate first writes a type identifier before writing the TDateTime value.

23.1.54.3.14 TWriter.WriteDescendent Method

Triggers the writing of descendant components in inherited forms.

Pascal

```
procedure WriteDescendent(Root: TComponent; AAncestor: TComponent);
```

C++

```
__fastcall WriteDescendent(TComponent * Root, TComponent * AAncestor);
```

Description

Never call WriteDescendent directly. It is part of a sequence of calls used for streaming descendant components in inherited forms.

WriteDescendent sets the writer object's Ancestor (see page 2839) and RootAncestor (see page 3192) properties to the value passed in AAncestor, sets the Root (see page 2840) property to the value passed in Root (see page 2840), then calls

WriteSignature (see page 3189) and WriteComponent (see page 3183) to write Root (see page 2840) and any components it owns to the writer object's stream.

WriteDescendent differs from WriteComponent (see page 3183) in that, by setting Ancestor (see page 2839), it uses Ancestor (see page 2839)'s property values as the defaults, rather than those defined by Root (see page 2840)'s type.

See Also

Ancestor (see page 2839)

Root (see page 2840)

WriteComponent (see page 3183)

23.1.54.3.15 TWriter.WriteDouble Method

Writes a value of type Double to the writer object’s stream.

Pascal

```
procedure WriteDouble(const Value: Double);
```

C++

```
__fastcall WriteDouble(const Double Value);
```

Description

WriteDouble is used internally by the component streaming system to write component properties to a stream. Use WriteDouble to write a value of type Double to the writer object’s stream.

WriteDouble checks to ensure that Value is the correct type before calling Write (see page 3181) to write the float value and its value type to the stream.

See Also

Write (see page 3181)

23.1.54.3.16 TWriter.WriteFloat Method

Writes the floating-point value passed in Value to the writer object's stream.

Pascal

```
procedure WriteFloat(const Value: Extended);
```

C++

```
__fastcall WriteFloat(const Extended Value);
```

Description

WriteFloat is used internally by the component streaming system to write component properties to a stream. Use WriteFloat for writing floating point data to streams.

WriteFloat checks to ensure that Value is the correct type before calling Write (see page 3181) to write the float value and its value type to the stream.

See Also

Write (see page 3181)

23.1.54.3.17 TWriter.WriteIdent Method

Writes the identifier passed in Ident to the writer object's stream.

Pascal

```
procedure WriteIdent(const Ident: string);
```

C++

```
__fastcall WriteIdent(const AnsiString Ident);
```

Description

WriteIdent is used internally by the component streaming system to write component properties to a stream. Use WriteIdent for writing component identifier properties to streams. Component names and enumerated type elements are examples of strings that must be restricted to identifier syntax.

WriteIdent checks for certain common identifiers (such as true, false, nil (Delphi), or NULL (C++)) and inserts a special token for these values. Otherwise, it writes the identifier as a string.

See Also

Write (see page 3181)

WriteStr (see page 3189)

23.1.54.3.18 WriteInteger Method

This is the overview for the WriteInteger method overload.

Overload List

	Name	Description
	TWriter.WriteInteger (Int64) (see page 3186)	Writes the integer value passed in Value to the writer object's stream.
	TWriter.WriteInteger (Longint) (see page 3186)	Writes the integer value passed in Value to the writer object's stream.

23.1.54.3.18.1 TWriter.WriteInteger Method (Int64)

Writes the integer value passed in Value to the writer object's stream.

Pascal

```
procedure WriteInteger(Value: Longint); overload;
procedure WriteInteger(Value: Int64); overload;
```

C++

```
__fastcall WriteInteger(Longint Value);
__fastcall WriteInteger(Int64 Value);
```

Description

WriteInteger is used internally by the component streaming system to write component properties to a stream. Use WriteInteger for writing component integer properties to streams.

WriteInteger checks to ensure that Value is the correct type before calling Write (see page 3181) to write the integer value and its value type to the stream. WriteInteger handles integers of different sizes.

See Also

Write (see page 3181)

23.1.54.3.19 TWriter.WriteListBegin Method

Writes a start-of-list marker to the writer object's associated stream.

Pascal

```
procedure WriteListBegin;
```

C++

```
__fastcall WriteListBegin();
```

Description

WriteListBegin is used by other methods that iterate through a group of items about to be written to the stream. WriteListBegin writes a start-of-list marker to the writer object's associated stream. Every call to WriteListBegin must have a corresponding call to WriteListEnd (see page 3187).

See Also

WriteListEnd (see page 3187)

23.1.54.3.20 TWriter.WriteListEnd Method

Writes an end-of-list marker to the writer object's associated stream.

Pascal

```
procedure WriteListEnd;
```

C++

```
__fastcall WriteListEnd();
```

Description

WriteListEnd is used by other methods that iterate through a group of items that are sequentially written to the stream during a single process.

A call to WriteListEnd is generally preceded by a writing loop. A call to WriteListEnd must correspond to a preceding call to WriteListBegin (see page 3187).

See Also

WriteListBegin (see page 3187)

23.1.54.3.21 TWriter.WritePrefix Method

Pascal

```
procedure WritePrefix(Flags: TFilerFlags; AChildPos: Integer);
```

C++

```
__fastcall WritePrefix(TFilerFlags Flags, int AChildPos);
```

Description

This is WritePrefix, a member of class TWriter.

23.1.54.3.22 TWriter.WriteProperties Method

Pascal

```
procedure WriteProperties(Instance: TPersistent);
```

C++

```
__fastcall WriteProperties(TPersistent * Instance);
```

Description

This is WriteProperties, a member of class TWriter.

23.1.54.3.23 TWriter.WriteProperty Method

Pascal

```
procedure WriteProperty(Instance: TPersistent; PropInfo: PPropInfo);
```

C++

```
__fastcall WriteProperty(TPersistent * Instance, PPropInfo PropInfo);
```

Description

This is WriteProperty, a member of class TWriter.

23.1.54.3.24 TWriter.WritePropName Method

Pascal

```
procedure WritePropName(const PropName: string);
```

C++

```
__fastcall WritePropName(const AnsiString PropName);
```

Description

This is WritePropName, a member of class TWriter.

23.1.54.3.25 TWriter.WriteRootComponent Method

Calls the WriteDescendent (see page 3184) method.

Pascal

```
procedure WriteRootComponent(Root: TComponent);
```

C++

```
__fastcall WriteRootComponent(TComponent * Root);
```

Description

WriteRootComponent is provided for backward compatibility only.

See Also

WriteDescendent (see page 3184)

23.1.54.3.26 TWriter.WriteSignature Method

Writes a filer signature to the writer object's associated stream.

Pascal

```
procedure WriteSignature;
```

C++

```
__fastcall WriteSignature();
```

Description

WriteSignature is used internally by several methods that write the filer signature before processing some action. For example, the WriteRootComponent (see page 3188) method calls WriteSignature before writing its component to the stream. By checking for the signature before loading objects, reader objects can guard against inadvertently reading invalid or corrupted data.

23.1.54.3.27 TWriter.WriteSingle Method

Writes a value of type Single (Delphi) or float (C++) to the writer's stream.

Pascal

```
procedure WriteSingle(const Value: Single);
```

C++

```
__fastcall WriteSingle(const Single Value);
```

Description

WriteSingle is used internally by the component streaming system to write component properties to a stream. Use WriteSingle for writing component properties of Delphi type Single (which corresponds to C++ type float) to streams.

WriteSingle first writes a type identifier before writing the value.

23.1.54.3.28 TWriter.WriteStr Method

Writes raw data to the writer object's stream.

Pascal

```
procedure WriteStr(const Value: string);
```

C++

```
__fastcall WriteStr(const AnsiString Value);
```

Description

Do not call WriteStr directly. WriteStr is for internal use by certain components. WriteStr writes the string passed in Value to the writer object's stream.

Warning: Always use WriteString (see page 3190) for writing component strings to streams. WriteStr can corrupt data if not used correctly.

See Also

WriteString (see page 3190)

23.1.54.3.29 TWriter.WriteString Method

Writes tagged data to the writer object's stream.

Pascal

```
procedure WriteString(const Value: string);
```

C++

```
__fastcall WriteString(const AnsiString Value);
```

Description

WriteString is used internally by the component streaming system to write component properties to a stream. WriteString is otherwise only used by component writers in the DefineProperties and WriteData procedures.

WriteString writes the string passed in Value to the writer object's stream. WriteString checks to ensure that Value is the correct type before calling Write (see page 3181) to write the string and its value type to the stream.

Warning: Always use WriteString for writing component strings to streams. The similarly-named WriteStr (see page 3189) method is for internal use by certain components and can corrupt data if not used correctly.

See Also

WriteChar (see page 3182)

WriteStr (see page 3189)

23.1.54.3.30 TWriter.WriteValue Method

Pascal

```
procedure WriteValue(Value: TValueType);
```

C++

```
__fastcall WriteValue(TValueType Value);
```

Description

This is WriteValue, a member of class TWriter.

23.1.54.3.31 TWriter.WriteVariant Method

Writes a Variant value to the writer's stream.

Pascal

```
procedure WriteVariant(const Value: Variant);
```

C++

```
__fastcall WriteVariant(const Variant Value);
```

Description

WriteVariant is used internally by the streaming system to write component properties to a stream. Use WriteVariant for writing component properties of type Variant to streams.

WriteVariant first writes a type identifier before writing the Variant value.

See Also

ReadVariant (see page 2955)

23.1.54.3.32 TWriter.WriteWideChar Method

Writes the wide character passed in Value to the writer object's stream.

Pascal

```
procedure WriteWideChar(Value: WideChar);
```

C++

```
__fastcall WriteWideChar(WideChar Value);
```

Description

WriteWideChar is used internally by the component streaming system to write component properties to a stream. Use WriteWideChar for writing component wide character properties to streams.

WriteWideChar calls WriteString (see page 3190) which checks to ensure that Value is the correct type before calling Write (see page 3181) to write the wide character and its value type to the stream.

Note: WriteWideChar calls StringChar to enable writing wide character properties that are upwardly compatible with string properties.

See Also

Write (see page 3181)

WriteString (see page 3190)

23.1.54.3.33 TWriter.WriteWideString Method

WriteWideString writes a wide string to the writer's stream.

Pascal

```
procedure WriteWideString(const Value: WideString);
```

C++

```
__fastcall WriteWideString(const BSTR Value);
```

Description

WriteWideString is used internally by the component streaming system to write component properties to a stream. Use WriteWideString for writing WideString component to streams.

WriteWideString first writes a type identifier and the length of the string before writing the value.

See Also

ObjectTextToBinary

WriteDescendent (see page 2994)

WriteComponentResFile

23.1.54.4 TWriter Properties

The properties of the TWriter class are listed here.

Public Properties

	Name	Description
	Position (see page 3192)	Represents the current writing position in the associated stream.
	RootAncestor (see page 3192)	Represents the ancestor of the component in the Root (see page 2840) property.
	UseQualifiedNames (see page 3193)	Indicates whether all class names should be qualified by the name of the unit in which they are defined.

23.1.54.4.1 TWriter.Position Property

Represents the current writing position in the associated stream.

Pascal

```
property Position: Longint;
```

C++

```
__property Longint Position;
```

Description

Position is used internally by writer objects to indicate the current writing position in the stream.

The value of Position will be inside the most recent buffer block read or the next block to be written. Thus, for writing, Position will generally be less than the stream's Position. When Position is set to a location outside the current buffer, the writer's buffer is flushed to the stream.

See Also

FlushBuffer (see page 3181)

23.1.54.4.2 TWriter.RootAncestor Property

Represents the ancestor of the component in the Root (see page 2840) property.

Pascal

```
property RootAncestor: TComponent;
```

C++

```
__property TComponent * RootAncestor;
```

Description

RootAncestor is used internally by writer objects for dealing with form inheritance. It is used to iterate through ancestor components when writing properties in inherited forms.

RootAncestor is always the ancestor form. For each component in Root (see page 2840), which is the inherited form, the Ancestor (see page 2839) property tracks a corresponding component in RootAncestor.

The writer object iterates through each of the Root (see page 2840) form's owned components, comparing each to the corresponding component in the RootAncestor form. It then writes only those properties in the current component that differ in some way from those in the Ancestor (see page 2839) component.

See Also

Ancestor (see page 2839)

Owner (see page 2959)

Root (see page 2840)

WriteDescendent (see page 3184)

23.1.54.4.3 TWriter.UseQualifiedNames Property

Indicates whether all class names should be qualified by the name of the unit in which they are defined.

Pascal

```
property UseQualifiedNames: Boolean;
```

C++

```
__property Boolean UseQualifiedNames;
```

Description

Set UseQualifiedNames to indicate whether class names need to be qualified by the name of the unit in which they are defined. In Windows versions of the IDE, for example, some components (such as TButton) appear both in the Windows-only and in the cross-platform component libraries. When writing such a component, setting UseQualifiedNames to true allows a reader to distinguish the Windows-only TButton control in StdCtrls (see page 20907) from the cross-platform TButton control in QStdCtrls.

See Also

WriteComponent (see page 3183)

23.1.54.5 TWriter Events

The events of the TWriter class are listed here.

Public Events

	Name	Description
	OnFindAncestor (see page 3193)	Occurs when a component is about to be written to a stream.
	OnFindMethodName (see page 3194)	This is OnFindMethodName, a member of class TWriter.

23.1.54.5.1 TWriter.OnFindAncestor Event

Occurs when a component is about to be written to a stream.

Pascal

```
property OnFindAncestor: TFindAncestorEvent;
```

C++

```
__property TFindAncestorEvent OnFindAncestor;
```

Description

OnFindAncestor allows the writer to explicitly specify the ancestor of a component when it is written to a stream. The WriteComponent (see page 3183) method generates this event to override the value of the Ancestor (see page 2839) property before writing a component to a stream.

See Also

Ancestor (see page 2839)

WriteComponent (see page 3183)

RootAncestor (see page 3192)

23.1.54.5.2 TWriter.OnFindMethodName Event

Pascal

```
property OnFindMethodName: TFindMethodNameEvent;
```

C++

```
__property TFindMethodNameEvent OnFindMethodName;
```

Description

This is OnFindMethodName, a member of class TWriter.

23.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IDesignerNotify (see page 3195)	IDesignerNotify is the interface for responding to notifications about changes to components in the designer.
	IInterfaceComponentReference (see page 3197)	IInterfaceComponentReference is the interface for responding to requests for retrieving components.
	IInterfaceList (see page 3198)	IInterfaceList provides access to a list of interfaces.
	IStreamPersist (see page 3207)	IStreamPersist is the interface for responding to requests to load/save an object from/to a persistent stream.
	IStringsAdapter (see page 3208)	This is class Classes.IStringsAdapter.
	IVarStreamable (see page 3210)	IVarStreamable is the interface for loading and saving the values of Variants (see page 23567).
	IVCLComObject (see page 3211)	This is class Classes.IVCLComObject.

23.2.1 IDesignerNotify Interface

IDesignerNotify is the interface for responding to notifications about changes to components in the designer.

Class Hierarchy

Classes.IDesignerNotify

Pascal

```
IDesignerNotify = interface;
```

C++

```
__interface IDesignerNotify;
```

File

Classes

Description

IDesignerNotify introduces two methods, Modified (see page 3196) and Notification (see page 3196), that allow the designer to respond when components are added, deleted, or changed. These methods are called automatically by the form designer or by component and property editors.

See Also

IDesigner (see page 11149)

23.2.1.1 IDesignerNotify Members

The following tables list the members exposed by IDesignerNotify.

Public Methods

	Name	Description
	Modified (see page 3196)	Notifies property and component editors when a change is made to a component.
	Notification (see page 3196)	Allows the designer to respond when a notification is sent to the form.

23.2.1.2 IDesignerNotify Methods

The methods of the IDesignerNotify class are listed here.

Public Methods

	Name	Description
	Modified (see page 3196)	Notifies property and component editors when a change is made to a component.
	Notification (see page 3196)	Allows the designer to respond when a notification is sent to the form.

23.2.1.2.1 IDesignerNotify.Modified Method

Notifies property and component editors when a change is made to a component.

Pascal

```
procedure Modified;
```

C++

```
__fastcall Modified();
```

Description

When any change is made to a component the property and component editors call this method, allowing the designer to respond to the change.

See Also

Modified (see page 11038)

23.2.1.2.2 IDesignerNotify.Notification Method

Allows the designer to respond when a notification is sent to the form.

Pascal

```
procedure Notification(AnObject: TPersistent; Operation: TOperation);
```

C++

```
__fastcall Notification(TPersistent * AnObject, TOperation Operation);
```

Description

When a form receives a notification, it calls the Notification method of the designer, allowing the designer to respond to all the notifications the form receives.

See Also

Notification (see page 2783)

23.2.2 IInterfaceComponentReference Interface

IInterfaceComponentReference is the interface for responding to requests for retrieving components.

Class Hierarchy

Classes.IInterfaceComponentReference

Pascal

```
IInterfaceComponentReference = interface;
```

C++

```
__interface IInterfaceComponentReference;
```

File

Classes

Description

IInterfaceComponentReference introduces one method, GetComponent (see page 3197), that allows the designer to respond a request to retrieve a reference to the component associated with a class.

For example, the TComponent (see page 2762) class itself implements the IInterfaceComponentReference.

See Also

TComponent (see page 2762)

23.2.2.1 IInterfaceComponentReference Members

The following tables list the members exposed by IInterfaceComponentReference.

Public Methods

	Name	Description
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.

23.2.2.2 IInterfaceComponentReference Methods

The methods of the IInterfaceComponentReference class are listed here.

Public Methods

	Name	Description
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.

23.2.2.2.1 IInterfaceComponentReference.GetComponent Method

Returns a reference to the component associated by the implementing class.

Pascal

```
function GetComponent: TComponent;
```

C++

```
__fastcall TComponent * GetComponent();
```

Description

GetComponent returns a TComponent (see page 2762) value that is associated with the implementing class. For example, the TComponent (see page 2762) class itself implements the GetComponent to return a pointer to itself.

See Also

TComponent (see page 2762)

23.2.3 IInterfaceList Interface

IInterfaceList provides access to a list of interfaces.

Class Hierarchy

Classes.IInterfaceList

Pascal

```
IInterfaceList = interface;
```

C++

```
__interface IInterfaceList;
```

File

Classes

Description

Use IInterfaceList to manage a list of interfaces. Typically, IInterfaceList is implemented by the TInterfaceList (see page 2855) class.

Note: C++ method declarations that use IInterfaceList use the _di_IInterfaceList type instead. This type is a DelphiInterface wrapper around the IInterfaceList interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IInterfaceList >  _di_IInterfaceList;
```

See Also

TInterfaceList (see page 2855)

23.2.3.1 IInterfaceList Members

The following tables list the members exposed by IInterfaceList.

Public Methods

	Name	Description
	Add (see page 3199)	Adds an interface to the list.
	Clear (see page 3200)	Clears the interface list.
	Delete (see page 3200)	Deletes an interface from the list.
	Exchange (see page 3201)	Swaps the position of two interfaces in the list.
	First (see page 3201)	Returns the first interface in the list.

	Get (see page 3201)	Returns a specified interface from the list.
	GetCapacity (see page 3202)	Returns the value of the Capacity (see page 3205) property.
	GetCount (see page 3202)	Returns the value of the Count (see page 3206) property.
	IndexOf (see page 3202)	Returns the index of an interface.
	Insert (see page 3202)	Inserts an interface into the list at a specified position.
	Last (see page 3203)	Returns the last interface in the list.
	Lock (see page 3203)	Prevents other threads from accessing the list.
	Put (see page 3204)	Replaces an interface in the list with a specified interface.
	Remove (see page 3204)	Removes an interface from the list.
	SetCapacity (see page 3204)	Sets the value of the Capacity (see page 3205) property.
	SetCount (see page 3205)	Sets the value of the Count (see page 3206) property.
	Unlock (see page 3205)	Unlocks the list, allowing other threads to access it.

Public Properties

	Name	Description
	Capacity (see page 3205)	Specifies the list capacity.
	Count (see page 3206)	Specifies the number of interfaces in the list.
	Items (see page 3206)	Provides direct access to an interface in the list.

23.2.3.2 IInterfaceList Methods

The methods of the IInterfaceList class are listed here.

Public Methods

	Name	Description
	Add (see page 3199)	Adds an interface to the list.
	Clear (see page 3200)	Clears the interface list.
	Delete (see page 3200)	Deletes an interface from the list.
	Exchange (see page 3201)	Swaps the position of two interfaces in the list.
	First (see page 3201)	Returns the first interface in the list.
	Get (see page 3201)	Returns a specified interface from the list.
	GetCapacity (see page 3202)	Returns the value of the Capacity (see page 3205) property.
	GetCount (see page 3202)	Returns the value of the Count (see page 3206) property.
	IndexOf (see page 3202)	Returns the index of an interface.
	Insert (see page 3202)	Inserts an interface into the list at a specified position.
	Last (see page 3203)	Returns the last interface in the list.
	Lock (see page 3203)	Prevents other threads from accessing the list.
	Put (see page 3204)	Replaces an interface in the list with a specified interface.
	Remove (see page 3204)	Removes an interface from the list.
	SetCapacity (see page 3204)	Sets the value of the Capacity (see page 3205) property.
	SetCount (see page 3205)	Sets the value of the Count (see page 3206) property.
	Unlock (see page 3205)	Unlocks the list, allowing other threads to access it.

23.2.3.2.1 IInterfaceList.Add Method

Adds an interface to the list.

Pascal

```
function Add(const Item: IInterface): Integer;
```

C++

```
__fastcall int Add(const IInterface Item);
```

Description

Call Add to add an interface to the list.

See Also

Count (see page 3206)

Add

Items (see page 3206)

Delete (see page 3200)

Capacity (see page 3205)

23.2.3.2.2 IInterfaceList.Clear Method

Clears the interface list.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to remove all interfaces from the list.

See Also

Items (see page 3206)

Delete (see page 3200)

23.2.3.2.3 IInterfaceList.Delete Method

Deletes an interface from the list.

Pascal

```
procedure Delete(Index: Integer);
```

C++

```
__fastcall Delete(int Index);
```

Description

Call Delete to delete an interface from the list.

See Also

Count (see page 3206)

Add (see page 3199)

Items (see page 3206)

Clear (see page 3200)

Remove (see page 3204)

23.2.3.2.4 IInterfaceList.Exchange Method

Swaps the position of two interfaces in the list.

Pascal

```
procedure Exchange(Index1: Integer; Index2: Integer);
```

C++

```
__fastcall Exchange(int Index1, int Index2);
```

Description

Call Exchange to move the interface at position Index1 so that it appears at position Index2 and move the interface at position Index2 so that it appears at position Index1. Index1 and Index2 identify two positions (zero offset) in the Items (see page 3206) property array.

See Also

Items (see page 3206)

23.2.3.2.5 IInterfaceList.First Method

Returns the first interface in the list.

Pascal

```
function First: IInterface;
```

C++

```
__fastcall IInterface First();
```

Description

Call First to obtain the first interface in the list.

See Also

Items (see page 3206)

Last (see page 3203)

23.2.3.2.6 IInterfaceList.Get Method

Returns a specified interface from the list.

Pascal

```
function Get(Index: Integer): IInterface;
```

C++

```
__fastcall IInterface Get(int Index);
```

Description

Get is the read implementation of the Items (see page 3206) property.

Index identifies which interface to return, where 0 is the first interface, 1 is the second interface, and so on.

See Also

Items (see page 3206)

23.2.3.2.7 IInterfaceList.GetCapacity Method

Returns the value of the Capacity (see page 3205) property.

Pascal

```
function GetCapacity: Integer;
```

C++

```
__fastcall int GetCapacity();
```

Description

GetCapacity is the read implementation of the Capacity (see page 3205) property.

See Also

Capacity (see page 3205)

23.2.3.2.8 IInterfaceList.GetCount Method

Returns the value of the Count (see page 3206) property.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the read implementation of the Count (see page 3206) property.

See Also

Count (see page 3206)

23.2.3.2.9 IInterfaceList.IndexOf Method

Returns the index of an interface.

Pascal

```
function IndexOf(const Item: IInterface): Integer;
```

C++

```
__fastcall int IndexOf(const IInterface Item);
```

Description

Call IndexOf to obtain the index of an interface.

See Also

Items (see page 3206)

23.2.3.2.10 IInterfaceList.Insert Method

Inserts an interface into the list at a specified position.

Pascal

```
procedure Insert(Index: Integer; const Item: IInterface);
```

C++

```
__fastcall Insert(int Index, const IInterface Item);
```

Description

Call Insert to insert an interface into the list. Item is the interface to insert, and Index indicates the position (zero-offset) where the interface should be added.

See Also

Count (see page 3206)

Add (see page 3199)

Items (see page 3206)

Delete (see page 3200)

Capacity (see page 3205)

23.2.3.2.11 IInterfaceList.Last Method

Returns the last interface in the list.

Pascal

```
function Last: IInterface;
```

C++

```
__fastcall IInterface Last();
```

Description

Call Last to obtain the last interface in the list.

See Also

First (see page 3201)

Items (see page 3206)

23.2.3.2.12 IInterfaceList.Lock Method

Prevents other threads from accessing the list.

Pascal

```
procedure Lock;
```

C++

```
__fastcall Lock();
```

Description

Call Lock to lock the list, preventing other threads from accessing the list.

See Also

Unlock (see page 3205)

23.2.3.2.13 IInterfaceList.Put Method

Replaces an interface in the list with a specified interface.

Pascal

```
procedure Put(Index: Integer; const Item: IInterface);
```

C++

```
__fastcall Put(int Index, const IInterface Item);
```

Description

Put is the write implementation of the Items (see page 3206) property.

Index is the index of the interface to replace.

Item is the interface that replaces the interface identified by Index.

See Also

Items (see page 3206)

23.2.3.2.14 IInterfaceList.Remove Method

Removes an interface from the list.

Pascal

```
function Remove(const Item: IInterface): Integer;
```

C++

```
__fastcall int Remove(const IInterface Item);
```

Description

Call Remove to remove an interface from the list.

Remove returns the index of the removed interface, or –1 if the interface was not found.

See Also

Add (see page 3199)

Insert (see page 3202)

Delete (see page 3200)

Clear (see page 3200)

23.2.3.2.15 IInterfaceList.SetCapacity Method

Sets the value of the Capacity (see page 3205) property.

Pascal

```
procedure SetCapacity(NewCapacity: Integer);
```

C++

```
__fastcall SetCapacity(int NewCapacity);
```

Description

SetCapacity is the write implementation of the Capacity (see page 3205) property.

NewCapacity is the new capacity for the interface list.

See Also

Capacity (see page 3205)

23.2.3.2.16 IInterfaceList.SetCount Method

Sets the value of the Count (see page 3206) property.

Pascal

```
procedure SetCount(NewCount: Integer);
```

C++

```
__fastcall SetCount(int NewCount);
```

Description

SetCount is the write implementation of the Count (see page 3206) property.

NewCount is the new number of interfaces in the list.

See Also

Count (see page 3206)

23.2.3.2.17 IInterfaceList.Unlock Method

Unlocks the list, allowing other threads to access it.

Pascal

```
procedure Unlock;
```

C++

```
__fastcall Unlock();
```

Description

Call Unlock to unlock the list so that other threads can have access to the interfaces it contains.

See Also

Lock (see page 3203)

23.2.3.3 IInterfaceList Properties

The properties of the IInterfaceList class are listed here.

Public Properties

	Name	Description
	Capacity (see page 3205)	Specifies the list capacity.
	Count (see page 3206)	Specifies the number of interfaces in the list.
	Items (see page 3206)	Provides direct access to an interface in the list.

23.2.3.3.1 IInterfaceList.Capacity Property

Specifies the list capacity.

Pascal

```
property Capacity: Integer;
```

C++

```
__property int Capacity;
```

Description

Use Capacity to determine the list capacity. Before adding many interfaces to the list, you can improve performance by setting Capacity to the final number of interfaces. This prevents the interface list from having to reallocate the list every time an interface is added.

See Also

Count (see page 3206)

Add (see page 3199)

Items (see page 3206)

23.2.3.3.2 IInterfaceList.Count Property

Specifies the number of interfaces in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use Count to determine the number of interfaces in the list.

See Also

Capacity (see page 3205)

Add (see page 3199)

Items (see page 3206)

Delete (see page 3200)

23.2.3.3.3 IInterfaceList.Items Property

Provides direct access to an interface in the list.

Pascal

```
property Items [Index: Integer]: IInterface;
```

C++

```
__property IInterface Items[int Index];
```

Description

Use Items to directly access an interface in the list. Index identifies each interface by its position in the list, where 0 is the first item, 1 is the second item, and so on.

Note: In Delphi, Items is the default property of IInterfaceList (see page 3198). This means that the name of the property can be omitted in code, and the subscript applied directly to the interface.

See Also

Count (see page 3206)

Add (see page 3199)

Delete (see page 3200)

23.2.4 IStreamPersist Interface

IStreamPersist is the interface for responding to requests to load/save an object from/to a persistent stream.

Class Hierarchy

Classes.IStreamPersist

Pascal

```
IStreamPersist = interface;
```

C++

```
__interface IStreamPersist;
```

File

Classes

Description

IStreamPersist introduces two methods, SaveToStream (see page 3208), and LoadFromStreamthat allows the designer to respond a request to store or retrieve the implementing object to a persistent stream.

You use SaveToStream (see page 3208) to serialise an object, and save this data in a stream independent of the running application. This serialised version of the object literally persists and is available for loading back using LoadFromStream (see page 3208). It is then de-serialised, and reconstituted as an object.

For example, the TGraphic class itself implements the IStreamPersist.

See Also

TGraphic (see page 14516)

23.2.4.1 IStreamPersist Members

The following tables list the members exposed by IStreamPersist.

Public Methods

	Name	Description
	LoadFromStream (see page 3208)	Loads an object from a persistent stream.
	SaveToStream (see page 3208)	Saves the implementing class instance to a persistent stream.

23.2.4.2 IStreamPersist Methods

The methods of the IStreamPersist class are listed here.

Public Methods

	Name	Description
	LoadFromStream (see page 3208)	Loads an object from a persistent stream.
	SaveToStream (see page 3208)	Saves the implementing class instance to a persistent stream.

23.2.4.2.1 IStreamPersist.LoadFromStream Method

Loads an object from a persistent stream.

Pascal

```
procedure LoadFromStream(Stream: TStream);
```

C++

```
__fastcall LoadFromStream(TStream Stream);
```

Description

Use LoadFromStream to reconstitute a serialised version of an object saved to a persistent stream by SaveToStream (see page 3208).

See Also

TGraphic (see page 14516)

23.2.4.2.2 IStreamPersist.SaveToStream Method

Saves the implementing class instance to a persistent stream.

Pascal

```
procedure SaveToStream(Stream: TStream);
```

C++

```
__fastcall SaveToStream(TStream Stream);
```

Description

Use SaveToStream to serialise an object, and save this data in a stream independent of the running application. This serialised version of the object literally persists and is available for loading back using LoadFromStream (see page 3208) at any time. It is then de-serialised, and reconstituted as an object.

See Also

TGraphic (see page 14516)

23.2.5 IStringsAdapter Interface

Class Hierarchy

Classes.IStringsAdapter

Pascal

```
IStringsAdapter = interface;
```

C++

```
__interface IStringsAdapter;
```

File

Classes

Description

This is class Classes.IStringsAdapter.

23.2.5.1 IStringsAdapter Members

The following tables list the members exposed by IStringsAdapter.

Public Methods

	Name	Description
	ReferenceStrings (see page 3209)	This is ReferenceStrings, a member of class IStringsAdapter.
	ReleaseStrings (see page 3209)	This is ReleaseStrings, a member of class IStringsAdapter.

23.2.5.2 IStringsAdapter Methods

The methods of the IStringsAdapter class are listed here.

Public Methods

	Name	Description
	ReferenceStrings (see page 3209)	This is ReferenceStrings, a member of class IStringsAdapter.
	ReleaseStrings (see page 3209)	This is ReleaseStrings, a member of class IStringsAdapter.

23.2.5.2.1 IStringsAdapter.ReferenceStrings Method

Pascal

```
procedure ReferenceStrings(S: TStrings);
```

C++

```
__fastcall ReferenceStrings(TStrings S);
```

Description

This is ReferenceStrings, a member of class IStringsAdapter.

23.2.5.2.2 IStringsAdapter.ReleaseStrings Method

Pascal

```
procedure ReleaseStrings;
```

C++

```
__fastcall ReleaseStrings();
```

Description

This is ReleaseStrings, a member of class IStringsAdapter.

23.2.6 IVarStreamable Interface

IVarStreamable is the interface for loading and saving the values of Variants (see page 23567).

Class Hierarchy

Classes.IVarStreamable

Pascal

```
IVarStreamable = interface;
```

C++

```
__interface IVarStreamable;
```

File

Classes

Description

Implement IVarStreamable on a custom Variant type class to allow Variants (see page 23567) of that type to load or save their values. When a TCustomVaraintType descendant implements the IVarStreamable interface, then when Variants (see page 23567) of that custom type are used as the values of published properties, their values are loaded from and saved to form files using the IVarStreamable methods. Without the IVarStreamable interface, the value of the Variant is loaded from and saved to form files as a string.

IVarStreamable defines two methods, StreamIn (see page 3211), and StreamOut (see page 3211), for reading and writing the Variant's value.

See Also

TCustomVariantType (see page 23581)

Loading and Saving Custom Variant Values

23.2.6.1 IVarStreamable Members

The following tables list the members exposed by IVarStreamable.

Public Methods

	Name	Description
	StreamIn (see page 3211)	Reads in the value of the Variant from a stream.
	StreamOut (see page 3211)	Writes the value of the Variant to a stream.

23.2.6.2 IVarStreamable Methods

The methods of the IVarStreamable class are listed here.

Public Methods

	Name	Description
	StreamIn (see page 3211)	Reads in the value of the Variant from a stream.
	StreamOut (see page 3211)	Writes the value of the Variant to a stream.

23.2.6.2.1 IVarStreamable.StreamIn Method

Reads in the value of the Variant from a stream.

Pascal

```
procedure StreamIn(var Dest: TVarData; const Stream: TStream);
```

C++

```
__fastcall StreamIn(TVarData Dest, const TStream Stream);
```

Description

StreamIn reads the Variant's value from a stream and fills in a TVarData data structure to reflect the value.

Dest is the TVarDatathat is filled in by StreamIn.

Stream is a stream that is positioned so that the next value that can be read from the stream is the value of the Variant.

See Also

StreamOut (see page 3211)

23.2.6.2.2 IVarStreamable.StreamOut Method

Writes the value of the Variant to a stream.

Pascal

```
procedure StreamOut(const Source: TVarData; const Stream: TStream);
```

C++

```
__fastcall StreamOut(const TVarData Source, const TStream Stream);
```

Description

StreamIn (see page 3211) writes the value from a TVarData value to a specified stream.

Source is the TVarData that stores the data for the Variant that needs to write its value.

Stream is a stream to which StreamOut writes the value stored in Source.

See Also

StreamIn (see page 3211)

23.2.7 IVCLComObject Interface

Class Hierarchy

Classes.IVCLComObject

Pascal

```
IVCLComObject = interface;
```

C++

```
__interface IVCLComObject;
```

File

Classes

Description

This is class Classes.IVCLComObject.

23.2.7.1 IVCLComObject Members

The following tables list the members exposed by IVCLComObject.

Public Methods

	Name	Description
	FreeOnRelease (see page 3212)	This is FreeOnRelease, a member of class IVCLComObject.
	GetIDsOfNames (see page 3212)	This is GetIDsOfNames, a member of class IVCLComObject.
	GetTypeInfo (see page 3213)	This is GetTypeInfo, a member of class IVCLComObject.
	GetTypeInfoCount (see page 3213)	This is GetTypeInfoCount, a member of class IVCLComObject.
	Invoke (see page 3213)	This is Invoke, a member of class IVCLComObject.
	SafeCallException (see page 3213)	This is SafeCallException, a member of class IVCLComObject.

23.2.7.2 IVCLComObject Methods

The methods of the IVCLComObject class are listed here.

Public Methods

	Name	Description
	FreeOnRelease (see page 3212)	This is FreeOnRelease, a member of class IVCLComObject.
	GetIDsOfNames (see page 3212)	This is GetIDsOfNames, a member of class IVCLComObject.
	GetTypeInfo (see page 3213)	This is GetTypeInfo, a member of class IVCLComObject.
	GetTypeInfoCount (see page 3213)	This is GetTypeInfoCount, a member of class IVCLComObject.
	Invoke (see page 3213)	This is Invoke, a member of class IVCLComObject.
	SafeCallException (see page 3213)	This is SafeCallException, a member of class IVCLComObject.

23.2.7.2.1 IVCLComObject.FreeOnRelease Method

Pascal

```
procedure FreeOnRelease;
```

C++

```
__fastcall FreeOnRelease();
```

Description

This is FreeOnRelease, a member of class IVCLComObject.

23.2.7.2.2 IVCLComObject.GetIDsOfNames Method

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int NameCount, int
LocaleID, void * DispIDs);
```

Description

This is GetIDsOfNames, a member of class IVCLComObject.

23.2.7.2.3 IVCLComObject.GetTypeInfo Method

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

This is GetTypeInfo, a member of class IVCLComObject.

23.2.7.2.4 IVCLComObject.GetTypeInfoCount Method

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

This is GetTypeInfoCount, a member of class IVCLComObject.

23.2.7.2.5 IVCLComObject.Invoke Method

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word Flags,
Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is Invoke, a member of class IVCLComObject.

23.2.7.2.6 IVCLComObject.SafeCallException Method

Pascal

```
function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): HResult;
```

C++

```
__fastcall HRESULT SafeCallException(TObject * ExceptObject, void * ExceptAddr);
```

Description

This is SafeCallException, a member of class IVCLComObject.

23.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	ActivateClassGroup (see page 3215)	Activates a group of classes that derive from TPersistent (see page 2928).
	ActiveClassGroup (see page 3216)	Activates a group of classes that derive from TPersistent (see page 2928).
	AllocateHWnd (see page 3217)	DeallocateHWnd (see page 3221) is used to destroy a hidden window created using AllocateHWnd.
	AncestorIsValid (see page 3217)	AncestorIsValid determines if one component is an ancestor of another.
	BeginGlobalLoading (see page 3218)	This is function Classes.BeginGlobalLoading.
	BinToHex (see page 3218)	Converts a binary value into its hexadecimal representation.
	Bounds (see page 3218)	Creates a TRect structure from the top-left coordinate, and width and height values of a rectangle.
	CheckSynchronize (see page 3219)	Allows background threads to synchronize their execution with the main thread.
	ClassGroupOf (see page 3220)	Returns the group of persistent classes a specified class or instance belongs to.
	ClassGroupOf (see page 3220)	Returns the group of persistent classes a specified class or instance belongs to.
	CollectionsEqual (see page 3220)	Compares the contents of two collections.
	CountGenerations (see page 3221)	Returns the number of intermediate classes between a derived class and its ancestor.
	DeallocateHWnd (see page 3221)	Frees a window that was created using AllocateHWnd (see page 3217).
	EndGlobalLoading (see page 3222)	This is function Classes.EndGlobalLoading.
	ExtractStrings (see page 3222)	Fills a string list with substrings parsed from a delimited list.
	FindClass (see page 3223)	Finds and returns a class that is derived from TPersistent (see page 2928).
	FindGlobalComponent (see page 3223)	Returns a top-level container component.
	FindIdentToInt (see page 3224)	Returns the TIdentToInt (see page 3295) conversion routine for a given data type.
	FindIntToIdent (see page 3224)	Returns the TIntToIdent (see page 3296) conversion routine for a given data type.
	FindNestedComponent (see page 3225)	FindNestedComponent returns a component contained within another.
	FindRootDesigner (see page 3225)	This is function Classes.FindRootDesigner.
	FreeObjectInstance (see page 3226)	This is function Classes.FreeObjectInstance.
	GetClass (see page 3226)	Returns a registered persistent class given its name.
	GetFixupInstanceNames (see page 3228)	This is function Classes.GetFixupInstanceNames.
	GetFixupReferenceNames (see page 3228)	This is function Classes.GetFixupReferenceNames.
	GetUltimateOwner (see page 3228)	Returns the top-level owner of a specified persistent object.
	GetUltimateOwner (see page 3228)	Returns the top-level owner of a specified persistent object.
	GetUltimateOwner (see page 3228)	Returns the top-level owner of a specified persistent object.
	GlobalFixupReferences (see page 3229)	This is function Classes.GlobalFixupReferences.
	GroupDescendentsWith (see page 3229)	Adds a specified class to the group in which another class appears.
	HexToBin (see page 3230)	Converts a string of hexadecimal digits to the corresponding binary value.
	IdentToInt (see page 3230)	Uses a mapping array to convert string identifiers into their corresponding integer values.
	InitComponentRes (see page 3231)	This is function Classes.InitComponentRes.
	InitInheritedComponent (see page 3231)	Initializes streaming of a form file for an inherited root class.
	IntToIdent (see page 3232)	Uses a mapping array to convert integers into their corresponding string identifiers.
	InvalidPoint (see page 3233)	Indicates whether a specified point is equal to (-1,-1).
	InvalidPoint (see page 3233)	Indicates whether a specified point is equal to (-1,-1).
	InvalidPoint (see page 3233)	Indicates whether a specified point is equal to (-1,-1).
	IsDefaultPropertyValue (see page 3233)	This is function Classes.IsDefaultPropertyValue.

	IsUniqueGlobalComponentName (see page 3234)	Indicates whether a proposed component name is unambiguous.
	LineStart (see page 3234)	Finds the end of the last whole line in a buffer.
	MakeObjectInstance (see page 3235)	This is function Classes.MakeObjectInstance.
	NotifyGlobalLoading (see page 3235)	This is function Classes.NotifyGlobalLoading.
	ObjectBinaryToText (see page 3235)	Converts the binary representation of an object into more readily understandable text.
	ObjectBinaryToText (see page 3235)	Converts the binary representation of an object into more readily understandable text.
	ObjectResourceToText (see page 3239)	Converts the binary representation of an object resource into more readily understandable text.
	ObjectResourceToText (see page 3239)	Converts the binary representation of an object resource into more readily understandable text.
	ObjectTextToBinary (see page 3240)	Converts a symbolic text representation of an object into the binary version that is used to save the object to files or memory streams.
	ObjectTextToBinary (see page 3240)	Converts a symbolic text representation of an object into the binary version that is used to save the object to files or memory streams.
	ObjectTextToResource (see page 3244)	Converts a symbolic text representation of an object into an internal binary representation.
	ObjectTextToResource (see page 3244)	Converts a symbolic text representation of an object into an internal binary representation.
	Point (see page 3244)	Creates a TPoint structure from a pair of coordinates.
	PointsEqual (see page 3245)	Indicates whether two points have the same coordinates.
	PointsEqual (see page 3245)	Indicates whether two points have the same coordinates.
	ReadComponentRes (see page 3245)	Reads components and their properties from a specified resource.
	ReadComponentResEx (see page 3246)	Reads a component from a resource.
	ReadComponentResFile (see page 3247)	Reads components and their properties from a specified Windows resource file.
	Rect (see page 3249)	Creates a TRect structure from a set of coordinates.
	Rect (see page 3249)	Creates a TRect structure from a set of coordinates.
	RedirectFixupReferences (see page 3250)	This is function Classes.RedirectFixupReferences.
	RegisterClass (see page 3250)	Registers a class of persistent object so that it's class type can be retrieved.
	RegisterClassAlias (see page 3254)	Registers a class that is identical to another class except for the name.
	RegisterClasses (see page 3255)	Registers a set of classes.
	RegisterComponents (see page 3256)	Registers a set of components so that they all appear on the same page of the component palette.
	RegisterFindGlobalComponentProc (see page 3259)	This is function Classes.RegisterFindGlobalComponentProc.
	RegisterIntegerConsts (see page 3259)	Registers conversion functions for string identifiers that represent type values.
	RegisterNoIcon (see page 3260)	Registers a set of components but does not add them to the component palette.
	RegisterNonActiveX (see page 3260)	Prevents a set of components from being made available to the ActiveX wizard.
	RemoveFixupReferences (see page 3261)	This is function Classes.RemoveFixupReferences.
	RemoveFixups (see page 3261)	This is function Classes.RemoveFixups.
	SmallPoint (see page 3262)	Creates a TSmallPoint structure from a pair of coordinates.
	StartClassGroup (see page 3262)	Starts a class group derived from TPersistent (see page 2928).
	TestStreamFormat (see page 3263)	Identifies the format of a form file when reading it from a stream.
	UnRegisterClass (see page 3263)	Unregisters an object class.
	UnRegisterClasses (see page 3264)	Unregisters a set of classes.
	UnregisterFindGlobalComponentProc (see page 3264)	This is function Classes.UnregisterFindGlobalComponentProc.
	UnregisterIntegerConsts (see page 3264)	Unregisters conversion functions for string identifiers that represent type values.
	UnRegisterModuleClasses (see page 3265)	Unregisters all classes defined in a specified module.
	WriteComponentResFile (see page 3266)	Writes components and their properties to a file using a resource file format.

23.3.1 Classes.ActivateClassGroup Function

Activates a group of classes that derive from TPersistent (see page 2928).

Pascal

```
function ActivateClassGroup(AClass: TPersistentClass): TPersistentClass;
```

C++

```
TPersistentClass ActivateClassGroup(TPersistentClass AClass);
```

File

Classes

Description

The streaming system allows the classes that can be loaded and saved to be registered in separate groups. This allows the IDE to distinguish between cross-platform and Windows-only classes. Call ActivateClassGroup to activate the group of classes that contains the class specified by AClass. If AClass is in more than one group, all groups that contain the class are activated.

ActivateClassGroup returns the class that was last used to activate a class group.

If the requested class group cannot be activated ActivateClassGroup raises an exception.

See Also

StartClassGroup (see page 3262)

GroupDescendantsWith

RegisterClasses (see page 3255)

23.3.2 Classes.ActiveClassGroup Function

Activates a group of classes that derive from TPersistent (see page 2928).

Pascal

```
function ActiveClassGroup: TPersistentClass;
```

C++

```
TPersistentClass ActiveClassGroup();
```

File

Classes

Description

The streaming system allows the classes that can be loaded and saved to be registered in separate groups. This allows the IDE to distinguish between cross-platform and Windows-only classes. Call ActivateClassGroup (see page 3215) to activate the group of classes that contains the class specified by AClass. If AClass is in more than one group, all groups that contain the class are activated.

ActiveClassGroup returns the class that was last used to activate a class group.

If the requested class group cannot be activated ActiveClassGroup raises an exception.

See Also

StartClassGroup (see page 3262)

GroupDescendantsWith

RegisterClasses (see page 3255)

23.3.3 Classes.AllocateHWnd Function

DeallocateHWnd (see page 3221) is used to destroy a hidden window created using AllocateHWnd.

Pascal

```
function AllocateHWnd(Method: TWndMethod): HWND;
```

C++

```
HWND AllocateHWnd(TWndMethod Method);
```

File

Classes

Description

HWnd means a 'Handle on a window'. For Windowless applications, you can call AllocateHWnd to allocate such a handle. This handle is used to allow a Windowless application to proces Windows messages.

You call DeallocateHWnd (see page 3221) to clean up resources after calling AllocateHWnd. You pass the **HWnd** that was allocated to AllocateHWnd.

See Also

AllocateHWnd

TWndMethod (see page 3307)

23.3.4 Classes.AncestorIsValid Function

AncestorIsValid determines if one component is an ancestor of another.

Pascal

```
function AncestorIsValid(Ancestor: TPersistent; Root: TComponent; RootAncestor: TComponent):
Boolean;
```

C++

```
Boolean AncestorIsValid(TPersistent * Ancestor, TComponent * Root, TComponent * RootAncestor);
```

File

Classes

Description

Call AncestorIsValid to determine if component **Root** has the specified **RootAncestor** ancestor. The TObject InheritsFrom method is used to determine this. true is returned if the ancestor is valid, otheriwise false.

See Also

TComponent

TObject

TPersistent

23.3.5 Classes.BeginGlobalLoading Function

Pascal

```
procedure BeginGlobalLoading;
```

C++

```
BeginGlobalLoading();
```

File

Classes

Description

This is function Classes.BeginGlobalLoading.

23.3.6 Classes.BinToHex Function

Converts a binary value into its hexadecimal representation.

Pascal

```
procedure BinToHex(Buffer: PChar; Text: PChar; BufSize: Integer);
```

C++

```
BinToHex(const char * Buffer, const char * Text, int BufSize);
```

File

Classes

Description

Call BinToHex to convert the binary value in a buffer into a string that is its hexadecimal representation.

Buffer is a buffer of bytes that contains the binary value.

Text returns a null-terminated string that represents the value of Buffer as a hexadecimal number.

BufSize is the size of Buffer. Text needs to point to a sequence of characters that has at least 2*BufSize bytes because each hexadecimal character represents two bytes.

See Also

HexToBin (see page 3230)

IntToHex

23.3.7 Classes.Bounds Function

Creates a TRect structure from the top-left coordinate, and width and height values of a rectangle.

Pascal

```
function Bounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer): TRect;
```

C++

```
TRect Bounds(int ALeft, int ATop, int AWidth, int AHeight);
```

File

Classes

Description

Call Bounds to create a TRect that represents the rectangle with the specified values. Use Rect (see page 3249) to construct parameters for functions that require TBounds, rather than setting up local variables for each parameter.

Note: Bounds is a useful alternative to Rect (see page 3249) if the rectangle coordinates would have to be calculated.

See Also

Rect (see page 3249)

Point (see page 3244)

23.3.8 Classes.CheckSynchronize Function

Allows background threads to synchronize their execution with the main thread.

Pascal

```
function CheckSynchronize(Timeout: Integer = 0): Boolean;
```

C++

```
Boolean CheckSynchronize(int Timeout = 0);
```

File

Classes

Description

It is not necessary to call CheckSynchronize in a GUI application. The call to CheckSynchronize is made automatically by the application object. In a non-GUI application, you must call CheckSynchronize if you use the Synchronize method of TThread (see page 3108). To do this, set the WakeMainThread (see page 3314) variable to a procedure that calls CheckSynchronize.

CheckSynchronize allows background threads to synchronize their execution with the main thread, so that it is safe to make method calls in the background thread.

CheckSynchronize returns **True** if a method was synchronized, **False** if it does nothing.

Warning: You should not modify WakeMainThread (see page 3314) for GUI applications because a working handler has been assigned. Console applications may assign a handler to WakeMainThread (see page 3314) if the application needs to be notified of a thread synchronization call. If your console application assigns a handler, this is an optimization that only accelerates deliver of notification that a synchronization request is available. You must use CheckSynchronize to perforrm the synchronization itself.

See Also

TThread (see page 3108)

WakeMainThread (see page 3314)

23.3.9 Classes.ClassGroupOf Function

Returns the group of persistent classes a specified class or instance belongs to.

Pascal

```
function ClassGroupOf(AClass: TPersistentClass): TPersistentClass; overload;
function ClassGroupOf(Instance: TPersistent): TPersistentClass; overload;
```

C++

```
TPersistentClass ClassGroupOf(TPersistentClass AClass);
TPersistentClass ClassGroupOf(TPersistent * Instance);
```

File

Classes

Description

The streaming system allows the classes that can be loaded and saved to be registered in separate groups. This allows the IDE to distinguish between cross-platform and Windows-only classes.

StartClassGroup (see page 3262) starts a new group. GroupDescendantsWith adds a class to an existing group.

ClassGroupOf returns the group that either a **AClass** class has been registered to, or the class of the specified **Instance** has been registered to.

See Also

GroupDescendantsWith

RegisterClass (see page 3250)

RegisterClasses (see page 3255)

StartClassGroup (see page 3262)

23.3.10 Classes.CollectionsEqual Function

Compares the contents of two collections.

Pascal

```
function CollectionsEqual(C1: TCollection; C2: TCollection; Owner1: TComponent; Owner2:
TComponent): Boolean;
```

C++

```
Boolean CollectionsEqual(TCollection C1, TCollection C2, TComponent * Owner1, TComponent *
Owner2);
```

File

Classes

Description

Call CollectionsEqual to determine whether two collections contain identical information. CollectionsEqual returns true if all the items in each collection have the same settings, and the items appear in the same order. CollectionsEqual returns false if the collections differ in number or order of items, or if any of the items in the collections differ.

C1 and C2 are the collections to compare.

Owner1 and Owner2 are the Owners of the collections. Owner1 and Owner2 must be "root" components, such as forms or data modules. They are used for resolving name references in the properties of the collections.

See Also

CompareMem

TStrings (see page 3029)

23.3.11 Classes.CountGenerations Function

Returns the number of intermediate classes between a derived class and its ancestor.

Pascal

```
function CountGenerations(Ancestor: TClass; Descendent: TClass): Integer;
```

C++

```
int CountGenerations(TClass Ancestor, TClass Descendent);
```

File

Classes

Description

Use CountGenerations to determine how close a derived class is to one of its ancestors. For example, if Ancestor and Descendant are the same class, CountGenerations returns 0. If Descendant is derived directly from Ancestor, CountGenerations returns 1.

Ancestor is the ancestor class.

Descendant is the descendant class. If it is not the same as Ancestor or does not derive from Ancestor, CountGenerations returns –1.

Tip: To check that Descendant derives from Ancestor using Delphi, use the is operator to check that Descendant is Ancestor before calling CountGenerations. In C++, use the static InheritsFrom method of Descendant.

23.3.12 Classes.DeallocateHWnd Function

Frees a window that was created using AllocateHWnd (see page 3217).

Pascal

```
procedure DeallocateHWnd(Wnd: HWND);
```

C++

```
DeallocateHWnd(HWND Wnd);
```

File

Classes

Description

Call DeallocateHWnd to free a window that was created using the AllocateHWnd (see page 3217) function. Windows must be freed to release the system resources that they consume.

The Wnd parameter is the window handle of the window to be freed. This value is the same as the handle returned by the call to AllocateHWnd (see page 3217).

Warning: Do not use DeallocateHWnd on the handle of a windowed control. Windowed controls free their handles using the protected DestroyHandle method, which ensures that the control is left in a consistent state.

See Also

AllocateHWnd (see page 3217)

TWinControl

23.3.13 Classes.EndGlobalLoading Function

Pascal

```
procedure EndGlobalLoading;
```

C++

```
EndGlobalLoading();
```

File

Classes

Description

This is function Classes.EndGlobalLoading.

23.3.14 Classes.ExtractStrings Function

Fills a string list with substrings parsed from a delimited list.

Pascal

```
function ExtractStrings(Separators: TSysCharSet; WhiteSpace: TSysCharSet; Content: PChar;
Strings: TStrings): Integer;
```

C++

```
int ExtractStrings(TSysCharSet Separators, TSysCharSet WhiteSpace, const char * Content,
TStrings Strings);
```

File

Classes

Description

Use ExtractStrings to fill a string list with the substrings of the null-terminated string specified by Content.

Separators is a set of characters that are used as delimiters, separating the substrings. Carriage returns, newline characters, and quote characters (single or double) are always treated as separators. Separators are ignored when inside a quoted string until the final end quote. (Note that quoted characters can appear in a quoted string if the quote character is doubled.)

WhiteSpace is a set of characters to be ignored when parsing Content if they occur at the beginning of a string.

Content is the null-terminated string to parse into substrings.

Strings is a string list to which all substrings parsed from Content are added. The string list is not cleared by ExtractStrings, so any strings already in the string list are preserved.

ExtractStrings returns the number of strings added to the Strings parameter.

Note: ExtractStrings does not add empty strings to the list.

See Also

ExtractHTTPFields

23.3.15 Classes.FindClass Function

Finds and returns a class that is derived from TPersistent (see page 2928).

Pascal

```
function FindClass(const ClassName: string): TPersistentClass;
```

C++

```
TPersistentClass FindClass(const AnsiString ClassName);
```

File

Classes

Description

Call FindClass to locate a class type by name. FindClass searches the classes that the streaming system knows about. Classes (see page 2685) can be registered with a call to RegisterClasses (see page 3255) so that the streaming system will recognize that class in a stream and know how to construct it.

If the requested class name cannot be found, FindClass raises an exception.

See Also

RegisterClasses (see page 3255)

GetClass (see page 3226)

23.3.16 Classes.FindGlobalComponent Function

Returns a top-level container component.

Pascal

```
function FindGlobalComponent(const Name: string): TComponent;
```

C++

```
TComponent * FindGlobalComponent(const AnsiString Name);
```

File

Classes

Description

FindGlobalComponent is used by the component streaming system to implement form linking. When a component property refers to a component on another form, data module, or other top-level container component, the streaming system calls

FindGlobalComponent to locate the root object in the form linking reference (e.g. Form1 in Form1.Edit1 (Delphi) or Form1->Edit1 (C++)). If the object cannot be found, FindGlobalComponent returns nil (Delphi) or NULL (C++).

See Also

EnumModules

EnumResourceModules

LibModuleList

IsUniqueGlobalComponentName (see page 3234)

23.3.17 Classes.FindIdentToInt Function

Returns the TIdentToInt (see page 3295) conversion routine for a given data type.

Pascal

```
function FindIdentToInt(AIntegerType: Pointer): TIdentToInt;
```

C++

```
TIdentToInt FindIdentToInt(void * AIntegerType);
```

File

Classes

Description

RegisterIntegerConsts (see page 3259) is used internally within Delphi to define conversions between name-string and integer value pairs. For example, the Graphics (see page 14387) unit uses this routine to register named colours, such as 'clMaroon', which is registered as integer value $000080. The conversion function is defined by type, TColor in this example.

FindIdentToInt is called to return the TIdentToInt (see page 3295) routine to be called to perform an Ident string to integer value conversion. You pass a data type in **AIntegerType** that identifies the conversion. For example, you would pass TypeInfo(TColor) if you wanted to be returned the Graphics (see page 14387) unit registered TColor value converion function.

See Also

IdentToInt (see page 3230)

RegisterIntegerConsts (see page 3259)

TIdentToInt (see page 3295)

23.3.18 Classes.FindIntToIdent Function

Returns the TIntToIdent (see page 3296) conversion routine for a given data type.

Pascal

```
function FindIntToIdent(AIntegerType: Pointer): TIntToIdent;
```

C++

```
TIntToIdent FindIntToIdent(void * AIntegerType);
```

File

Classes

Description

RegisterIntegerConsts (see page 3259) is used internally within Delphi to define conversions between name-string and integer value pairs. For example, the Graphics (see page 14387) unit uses this routine to register named colours, such as 'clMaroon', which is registered as integer value $000080. The conversion function is defined by type, TColor in this example.

FindIntToIdent is called to return the TIntToIdent (see page 3296) routine to be called to perform an integer value to Ident string conversion. You pass a data type in **AIntegerType** that identifies the conversion. For example, you would pass TypeInfo(TColor) if you wanted to be returned the Graphics (see page 14387) unit registered TColor value conversion function.

See Also

IntToIdent (see page 3232)

RegisterIntegerConsts (see page 3259)

TIntToIdent (see page 3296)

23.3.19 Classes.FindNestedComponent Function

FindNestedComponent returns a component contained within another.

Pascal

```
function FindNestedComponent(Root: TComponent; const NamePath: string): TComponent;
```

C++

```
TComponent * FindNestedComponent(TComponent * Root, const AnsiString NamePath);
```

File

Classes

Description

Call FindNestedComponent to find a component by name contained within a specified containing component.

Root specifies the containing component object.

NamePath gives the name of the component nested in **Root**.

If found, a reference to the nested component is returned.

See Also

TComponent

23.3.20 Classes.FindRootDesigner Function

Pascal

```
function FindRootDesigner(Obj: TPersistent): IDesignerNotify;
```

C++

```
IDesignerNotify FindRootDesigner(TPersistent * Obj);
```

File

Classes

Description

This is function Classes.FindRootDesigner.

23.3.21 Classes.FreeObjectInstance Function

Pascal

```
procedure FreeObjectInstance(ObjectInstance: Pointer);
```

C++

```
FreeObjectInstance(void * ObjectInstance);
```

File

Classes

Description

This is function Classes.FreeObjectInstance.

23.3.22 Classes.GetClass Function

Returns a registered persistent class given its name.

Pascal

```
function GetClass(const AClassName: string): TPersistentClass;
```

C++

```
TPersistentClass GetClass(const AnsiString AClassName);
```

File

Classes

Description

Call GetClass to obtain a class from a class name. This class can be used as a parameter to routines that require a class.

The Class must be registered before GetClass can find it. Form classes and component classes that are referenced in a form declaration (instance variables) are automatically registered when the form is loaded. Other classes can be registered by calling RegisterClass (see page 3250) or RegisterClasses (see page 3255).

Note: To obtain an unregistered class from a class name in C++, use the __classid routine.

C++ Examples:

```
/*
The following example allows users to type the name of a
graphics class into an edit control and when a button is
pressed, the name of the default extension for that class is
displayed in another edit control.  In the form's OnCreate
```

```
  event handler (or a similar place) you must register the
  graphics classes so that GetClass can find them.
  */
  TMetaClass *MetaClass;

  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    TClassFinder *myClassFinder = new TClassFinder(MetaClass, False);
    TGraphicClass mygraphclass = (TGraphicClass) (myClassFinder->GetClass(Edit2->Text));
  //  TGraphicClass mygraphclass = (TGraphicClass) GetClass(Edit2->Text);
    Edit1->Text = GraphicExtension(mygraphclass);
  }

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    MetaClass = __classid(TIcon); // ico
    RegisterClasses(&MetaClass, 0);
    MetaClass = __classid(Graphics::TBitmap); // bmp
    RegisterClasses(&MetaClass, 0);
    MetaClass = __classid(TMetafile); // emf
    RegisterClasses(&MetaClass, 0);
    MetaClass = __classid(TShape);  // nographic extension
    RegisterClasses(&MetaClass, 0);
    MetaClass = __classid(TImage); // nographic extension
    RegisterClasses(&MetaClass, 0);
  }
```

Delphi Examples:

```
  {
  The following example allows users to type the name of a
  graphics class into an edit control and when a button is
  pressed, the name of the default extension for that class is
  displayed in another edit control.  In the form's OnCreate
  event handler (or a similar place) you must register the
  graphics classes so that GetClass can find them.
  }

  uses ExtCtrls;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    myClassFinder: TClassFinder;
    mygraphclass: TGraphicClass;
  begin
    { make sure the classes are registered so that GetClass will work -- }
    { Usually, this goes in the initialization section where it is only executed once }
    RegisterClasses([TIcon, TBitmap, TButton, TShape, TImage, TMetafile]);
    try
      myClassFinder := TClassFinder.Create(TIcon, False);
      mygraphclass := TGraphicClass(myClassFinder.GetClass(Edit2.Text));
  //    mygraphclass := TGraphicClass(GetClass(Edit2.Text));  // works as well
      Edit1.Text := GraphicExtension(mygraphclass);
    except
      on EAccessViolation do
        MessageDlg('Invalid class name.', mtError, [mbOK], 0)
    end;
  end;
```

See Also

RegisterClass (see page 3250)

FindClass (see page 3223),

23.3.23 Classes.GetFixupInstanceNames Function

Pascal

```
procedure GetFixupInstanceNames(Root: TComponent; const ReferenceRootName: string; Names:
TStrings);
```

C++

```
GetFixupInstanceNames(TComponent * Root, const AnsiString ReferenceRootName, TStrings Names);
```

File

Classes

Description

This is function Classes.GetFixupInstanceNames.

23.3.24 Classes.GetFixupReferenceNames Function

Pascal

```
procedure GetFixupReferenceNames(Root: TComponent; Names: TStrings);
```

C++

```
GetFixupReferenceNames(TComponent * Root, TStrings Names);
```

File

Classes

Description

This is function Classes.GetFixupReferenceNames.

23.3.25 Classes.GetUltimateOwner Function

Returns the top-level owner of a specified persistent object.

Pascal

```
function GetUltimateOwner(ACollectionItem: TCollectionItem): TPersistent; overload;
function GetUltimateOwner(ACollection: TCollection): TPersistent; overload;
function GetUltimateOwner(APersistent: TPersistent): TPersistent; overload;
```

C++

```
TPersistent * GetUltimateOwner(TCollectionItem ACollectionItem);
TPersistent * GetUltimateOwner(TCollection ACollection);
TPersistent * GetUltimateOwner(TPersistent * APersistent);
```

File

Classes

Description

Left undocumented because it doesn't really do the right thing.

Call GetUltimateOwner to obtain the ultimate owner of a persistent object. GetUltimateOwner traverses the ownership hierarchy, finding first the Owner, and then that Owner's Owner, and so on, until there are no more owners.

The ultimate owner is typically the form or data module that contains the object instance. However, if one of the Owners (other than a form or data module) does not have an Owner, GetUltimateOwner returns nil (Delphi) or NULL (C++).

See Also

GetParentForm

23.3.26 Classes.GlobalFixupReferences Function

Pascal

```
procedure GlobalFixupReferences;
```

C++

```
GlobalFixupReferences();
```

File

Classes

Description

This is function Classes.GlobalFixupReferences.

23.3.27 Classes.GroupDescendentsWith Function

Adds a specified class to the group in which another class appears.

Pascal

```
procedure GroupDescendentsWith(AClass: TPersistentClass; AClassGroup: TPersistentClass);
```

C++

```
GroupDescendentsWith(TPersistentClass AClass, TPersistentClass AClassGroup);
```

File

Classes

Description

Call GroupDescendentsWith to add the class specified by AClass to the same group as the one that contains AClassGroup. If AClassGroup is a member of more than one group, GroupDescendentsWith adds AClass to the group that contains the nearest ancestor of AClassGroup. (The nearest ancestor is the ancestor for which CountGenerations (see page 3221) returns the lowest value).

If AClassGroup is not a member of any group, GroupDescendentsWith raises an exception.

See Also

ActivateClassGroup (see page 3215)

StartClassGroup (see page 3262)

CountGenerations (see page 3221)

23.3.28 Classes.HexToBin Function

Converts a string of hexadecimal digits to the corresponding binary value.

Pascal

```
function HexToBin(Text: PChar; Buffer: PChar; BufSize: Integer): Integer;
```

C++

```
int HexToBin(const char * Text, const char * Buffer, int BufSize);
```

File

Classes

Description

Call HexToBin to convert the hexadecimal string Text to the binary value it represents.

Text is a string representation of a hexadecimal value.

Buffer returns the resulting value in binary.

BufSize is the size of Buffer. Text needs to point to at least 2*BufSize hexadecimal characters, because each two hexadecimal characters represent one byte.

HexToBin returns the number of characters in Buffer that have not been used because Text did not contain valid hexadecimal characters ('0'..'f').

Note: The hexadecimal number must use lower-case characters; HexToBind does not recognize upper-case characters.

See Also

BinToHex (see page 3218)

23.3.29 Classes.IdentToInt Function

Uses a mapping array to convert string identifiers into their corresponding integer values.

Pascal

```
function IdentToInt(const Ident: string; var Int: Longint; const Map: array of
TIdentMapEntry): Boolean;
```

C++

```
Boolean IdentToInt(const AnsiString Ident, Longint Int, const array of TIdentMapEntry Map);
```

File

Classes

Description

IdentToInt provides the underlying translation from string identifiers to integers that occurs, for example, when you register a mapping using the RegisterIntegerConsts (see page 3259) procedure in Delphi.

Ident is the string identifier to translate.

Int returns the corresponding integer value.

Mapis an array of records (Delphi) or structs (C++) that describe the mapping. Each member of the array contains a Value field, which is the integer value to return and a Name field, which is the string identifier to translate.

Note: In C++, Map_Size is the index of the last entry in Map (one less than the number of elements).

IdentToInt looks for the string specified by Ident as the Name field on an entry in Map. If it finds a match, it sets Int to the corresponding Value field and returns true. If it does not find a match, it returns False.

See Also

RegisterIntegerConsts (see page 3259)

IntToIdent (see page 3232)

23.3.30 Classes.InitComponentRes Function

Pascal

```
function InitComponentRes(const ResName: string; Instance: TComponent): Boolean;
```

C++

```
Boolean InitComponentRes(const AnsiString ResName, TComponent * Instance);
```

File

Classes

Description

This is function Classes.InitComponentRes.

23.3.31 Classes.InitInheritedComponent Function

Initializes streaming of a form file for an inherited root class.

Pascal

```
function InitInheritedComponent(Instance: TComponent; RootAncestor: TClass): Boolean;
```

C++

```
Boolean InitInheritedComponent(TComponent * Instance, TClass RootAncestor);
```

File

Classes

Description

Call InitInheritedComponent from the constructor of a root class to start streaming in of information in the associated form file.

Instance is the object being instantiated.

RootAncestor is the base root class of which Instance may represent a descendant. This class must be a root class, meaning a class such as TCustomForm, TFrame, or TDataModule (see page 2824), which is the starting point of the streaming process (the component associated with the form file).

InitInheritedComponent returns true if the form file was successfully streamed in, false otherwise.

The call to InitInheritedComponent is only required for descendant classes of the root class, and must follow the construction of the object instance except for streaming in of persistent properties in the form file. The following Delphi and C++ code samples show typical root class contstructors:

constructor TMyRootClass.Create(AOwner: TComponent (see page 2762));

```
begin
  inherited Create(AOwner); { first construct an object instance }
  if (ClassType <> TMyRootClass) and not (csDesigning in ComponentState) then
  begin
    if not InitInheritedComponent(Self, TMyRootClass) then
      raise EResNotFound.CreateFmt(SResNotFound, [ClassName]);
  end;
end;
```

void TMyRootClass.TMyRootClass(TComponent (see page 2762) AOwner);

```
{
  if ( (ClassType != TMyRootClass) &&
    !(ComponentState.Contains(csDesigning))
  {
    if !(InitInheritedComponent(this, TMyRootClass))
      throw EResNotFound(SResNotFound, TVarRec theClassName[] =
        {ClassName}, 1);
  }
}
```

See Also

ReadComponentResFile (see page 3247)

TCustomForm

TCustomForm

TDataModule (see page 2824)

TDataModule (see page 2824)

23.3.32 Classes.IntToIdent Function

Uses a mapping array to convert integers into their corresponding string identifiers.

Pascal

```
function IntToIdent(Int: Longint; var Ident: string; const Map: array of TIdentMapEntry):
Boolean;
```

C++

```
Boolean IntToIdent(Longint Int, AnsiString Ident, const array of TIdentMapEntry Map);
```

File

Classes

Description

IntToIdent provides the underlying translation from string identifiers to integers that occurs, for example, when you register a mapping using the RegisterIntegerConsts (see page 3259) procedure in Delphi.

Int is the integer to translate.

Ident returns the corresponding string identifier.

Map is an array of records (Delphi) or structs (C++) that describe the mapping. Each member of the array contains a Value field, which is the integer value to translate and a Name field, which is the corresponding string identifier.

Note: In C++, Map_Size is the index of the last entry in Map (one less than the number of elements).

IntToIdent looks for the integer specified by Int as the Value field on an entry in Map. If it finds a match, it sets Ident to the corresponding Name field and returns true. If it does not find a match, it returns false.

See Also

RegisterIntegerConsts (see page 3259)

IdentToInt (see page 3230)

23.3.33 Classes.InvalidPoint Function

Indicates whether a specified point is equal to (-1,-1).

Pascal

```
function InvalidPoint(X: Integer; Y: Integer): Boolean; overload;
function InvalidPoint(const At: TPoint): Boolean; overload;
function InvalidPoint(const At: TSmallPoint): Boolean; overload;
```

C++

```
Boolean InvalidPoint(int X, int Y);
Boolean InvalidPoint(const TPoint At);
Boolean InvalidPoint(const TSmallPoint At);
```

File

Classes

Description

The point (-1,-1) is often used, by convention, to represent an invalid set of coordinates. This convention is often employed by Windows API functionsand by several Linux utilities. When working with other code that uses this convention, use InvalidPoint to identify coordinate values that are invalid.

At is the point to examine.

X and Y identify the point by giving the X and Y coordinates respectively.

InvalidPoint returns true if the specified point is (-1,-1), false otherwise.

23.3.34 Classes.IsDefaultPropertyValue Function

Pascal

```
function IsDefaultPropertyValue(Instance: TObject; PropInfo: PPropInfo; OnGetLookupInfo:
TGetLookupInfoEvent; Writer: TWriter = nil; OnFindMethodName: TFindMethodNameEvent = nil):
Boolean;
```

C++

```
Boolean IsDefaultPropertyValue(TObject * Instance, PPropInfo PropInfo, TGetLookupInfoEvent
OnGetLookupInfo, TWriter Writer = nil, TFindMethodNameEvent OnFindMethodName = nil);
```

File

Classes

Description

This is function Classes.IsDefaultPropertyValue.

23.3.35 Classes.IsUniqueGlobalComponentName Function

Indicates whether a proposed component name is unambiguous.

Pascal

```
function IsUniqueGlobalComponentName(const Name: string): Boolean;
```

C++

```
Boolean IsUniqueGlobalComponentName(const AnsiString Name);
```

File

Classes

Description

IsUniqueGlobalComponentName returns true if the name specified by Name is not currently used by any component on any form, data module, or other top-level container component. IsUniqueGlobalComponentName returns false if another component is already using Name (in which case, references to the component should be qualified by the root object's name.

See Also

FindGlobalComponent (see page 3223)

UniqueString

23.3.36 Classes.LineStart Function

Finds the end of the last whole line in a buffer.

Pascal

```
function LineStart(Buffer: PChar; BufPos: PChar): PChar;
```

C++

```
const char * LineStart(const char * Buffer, const char * BufPos);
```

File

Classes

Description

Use LineStart to find the start of the last partial line in the buffer. LineStart starts at BufPos and scans backwards for a line-feed character (\n). It returns a pointer to that character. If no line-feed char is found, it returns the Buffer pointer. Buffer should point to the beginning of a block of memory and BufPos to the end of the block you want to scan.

Note: The undocumented TParser (see page 2915) class uses LineStart to locate line-endings in chunks of text read into a fixed-size buffer. When the last full line of text is parsed, the parser moves the last partial line of text from the end of the buffer to

the front and fills the rest of the buffer with new data from disk.

23.3.37 Classes.MakeObjectInstance Function

Pascal

```
function MakeObjectInstance(Method: TWndMethod): Pointer;
```

C++

```
void * MakeObjectInstance(TWndMethod Method);
```

File

Classes

Description

This is function Classes.MakeObjectInstance.

23.3.38 Classes.NotifyGlobalLoading Function

Pascal

```
procedure NotifyGlobalLoading;
```

C++

```
NotifyGlobalLoading();
```

File

Classes

Description

This is function Classes.NotifyGlobalLoading.

23.3.39 Classes.ObjectBinaryToText Function

Converts the binary representation of an object into more readily understandable text.

Pascal

```
procedure ObjectBinaryToText(Input: TStream; Output: TStream); overload;
procedure ObjectBinaryToText(Input: TStream; Output: TStream; var OriginalFormat:
TStreamOriginalFormat); overload;
```

C++

```
ObjectBinaryToText(TStream Input, TStream Output);
ObjectBinaryToText(TStream Input, TStream Output, TStreamOriginalFormat OriginalFormat);
```

File

Classes

Description

Call ObjectBinaryToText to convert the representation of an object as it is saved in files and memory streams into a text version that human readers can understand. ObjectBinaryToText creates a TReader (see page 2933) object for the Input stream, and uses this to read the saved representation of the object. A symbolic text representation is then written to the Output stream. OriginalFormat indicates the format in which the object was saved.

Use the WriteComponent method of the Input stream to write a binary representation before calling ObjectBinaryToText.

C++ Examples:

```
/*
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
*/
class MyScrollBar : public TScrollBar
{
__published:    // IDE-managed Components
private:    // User declarations
public:     // User declarations
    __fastcall MyScrollBar(TComponent* Owner);
};

__fastcall MyScrollBar::MyScrollBar(TComponent* Owner)
    : TScrollBar(Owner)
{
}

void RegisterClassesWithStreamingSystem(void)
{
  // Make sure that as part of the startup
  // code our streaming classes are registered
  // with the streaming system.

  #pragma startup RegisterClassesWithStreamingSystem

  Classes::RegisterClass(__classid(MyScrollBar));
}

MyScrollBar *ScrollBar1;

AnsiString __fastcall ComponentToString(TComponent *c)
{
 AnsiString as;
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);

 try
 {
    pms->WriteComponent(c);
    pms->Seek(0, soFromBeginning);
    ObjectBinaryToText(pms, pss);
    pss->Seek(0, soFromBeginning);
    as = pss->DataString;
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 delete pms;
 delete pss;
 return as;
}
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Memo1->Text = ComponentToString(ScrollBar1);
}

TComponent *__fastcall StringToComponent(AnsiString as)

{
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);
 TComponent *pc;

 try
 {
    ObjectTextToBinary(pss, pms);
    pms->Seek(0, soFromBeginning);
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 pc = pms->ReadComponent(NULL);
 delete pms;
 delete pss;
 return pc;
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  TComponent *temp = StringToComponent(Memo1->Text);
  ScrollBar1->Free();

  ScrollBar1 = dynamic_cast<MyScrollBar *>(temp);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ScrollBar1 = new MyScrollBar(Form1);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
  ScrollBar1->Top = 48;
  ScrollBar1->Left = 250;
  ScrollBar1->Name = "Ricksbar";
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete ScrollBar1;
}
```

Delphi Examples:

```
{
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
}
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
```

```
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  MyScrollBar = class(TScrollBar)
  end;

var
  Form1: TForm1;
  ScrollBar1: MyScrollBar;

implementation

{$R *.dfm}

function ComponentToStringProc(Component: TComponent): string;
var
  BinStream:TMemoryStream;
  StrStream: TStringStream;
  s: string;
begin
  BinStream := TMemoryStream.Create;
  try
    StrStream := TStringStream.Create(s);
    try
      BinStream.WriteComponent(Component);
      BinStream.Seek(0, soFromBeginning);
      ObjectBinaryToText(BinStream, StrStream);
      StrStream.Seek(0, soFromBeginning);
      Result:= StrStream.DataString;
    finally
      StrStream.Free;
    end;
  finally
    BinStream.Free
  end;
end;

function StringToComponentProc(Value: string): TComponent;
var
  StrStream:TStringStream;
  BinStream: TMemoryStream;
begin
  StrStream := TStringStream.Create(Value);
  try
    BinStream := TMemoryStream.Create;
    try
      ObjectTextToBinary(StrStream, BinStream);
      BinStream.Seek(0, soFromBeginning);
      Result:= BinStream.ReadComponent(nil);
    finally
      BinStream.Free;
    end;
  finally
    StrStream.Free;
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Memo1.Text:= ComponentToStringProc(ScrollBar1);
end;
```

```
// Edit the scrollbar parameters in the memo, click this
// button and see the scrollbar change.
procedure TForm1.Button2Click(Sender: TObject);
begin
  ScrollBar1.Free;
  ScrollBar1:= (StringToComponentProc(Memo1.Text) as MyScrollBar);
  ScrollBar1.Parent:= Form1;
  ScrollBar1.Visible:= TRUE;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ScrollBar1:= MyScrollBar.Create(Form1);
  ScrollBar1.Parent:= Form1;
  ScrollBar1.Visible:= TRUE;
  ScrollBar1.Top:= 48;
  ScrollBar1.Left:= 250;
  ScrollBar1.Name:= 'Ricksbar';
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  ScrollBar1.Free;
end;

initialization

RegisterClass(MyScrollBar);
```

See Also

TReader (see page 2933)

TStream (see page 2973)

23.3.40 Classes.ObjectResourceToText Function

Converts the binary representation of an object resource into more readily understandable text.

Pascal

```
procedure ObjectResourceToText(Input: TStream; Output: TStream); overload;
procedure ObjectResourceToText(Input: TStream; Output: TStream; var OriginalFormat:
TStreamOriginalFormat); overload;
```

C++

```
ObjectResourceToText(TStream Input, TStream Output);
ObjectResourceToText(TStream Input, TStream Output, TStreamOriginalFormat OriginalFormat);
```

File

Classes

Description

Call ObjectResourceToText to convert the representation of an object as it is saved in module resource files into a text version that that human readers can understand. ObjectResourceToText is the same as ObjectBinaryToText (see page 3235), except that it reads from a stream that includes a resource header.

ObjectResourceToText converts the binary version of a form, as stored in a form file, into the text version that can be seen by right clicking the form and choosing View As Text.

Use the WriteComponentRes method of the Input stream to write the binary representation before calling ObjectResourceToText.

See Also

TResourceStream (see page 2968)

TStream (see page 2973)

23.3.41 Classes.ObjectTextToBinary Function

Converts a symbolic text representation of an object into the binary version that is used to save the object to files or memory streams.

Pascal

```
procedure ObjectTextToBinary(Input: TStream; Output: TStream); overload;
procedure ObjectTextToBinary(Input: TStream; Output: TStream; var OriginalFormat:
TStreamOriginalFormat); overload;
```

C++

```
ObjectTextToBinary(TStream Input, TStream Output);
ObjectTextToBinary(TStream Input, TStream Output, TStreamOriginalFormat OriginalFormat);
```

File

Classes

Description

Call ObjectTextToBinary to create the internal representation of a component from a symbolic text representation. ObjectTextToBinary creates a TWriter (see page 3175) object for the Output stream, and uses this to write the form file representation of the object from the symbolic representation that is read from the Input stream. OriginalFormat indicates whether the output stream stores the object in binary or text format.

Use the ReadComponent method of the output stream to obtain the actual object.

C++ Examples:

```
/*
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
*/
class MyScrollBar : public TScrollBar
{
__published:    // IDE-managed Components
private:    // User declarations
public:     // User declarations
    __fastcall MyScrollBar(TComponent* Owner);
};

__fastcall MyScrollBar::MyScrollBar(TComponent* Owner)
    : TScrollBar(Owner)
{
}

void RegisterClassesWithStreamingSystem(void)
{
  // Make sure that as part of the startup
  // code our streaming classes are registered
  // with the streaming system.
```

```
  #pragma startup RegisterClassesWithStreamingSystem

  Classes::RegisterClass(__classid(MyScrollBar));
}

MyScrollBar *ScrollBar1;

AnsiString __fastcall ComponentToString(TComponent *c)
{
 AnsiString as;
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);

 try
 {
    pms->WriteComponent(c);
    pms->Seek(0, soFromBeginning);
    ObjectBinaryToText(pms, pss);
    pss->Seek(0, soFromBeginning);
    as = pss->DataString;
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 delete pms;
 delete pss;
 return as;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Memo1->Text = ComponentToString(ScrollBar1);
}

TComponent *__fastcall StringToComponent(AnsiString as)

{
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);
 TComponent *pc;

 try
 {
    ObjectTextToBinary(pss, pms);
    pms->Seek(0, soFromBeginning);
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 pc = pms->ReadComponent(NULL);
 delete pms;
 delete pss;
 return pc;
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  TComponent *temp = StringToComponent(Memo1->Text);
  ScrollBar1->Free();

  ScrollBar1 = dynamic_cast<MyScrollBar *>(temp);
  ScrollBar1->Parent = Form1;
```

```
  ScrollBar1->Visible = TRUE;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ScrollBar1 = new MyScrollBar(Form1);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
  ScrollBar1->Top = 48;
  ScrollBar1->Left = 250;
  ScrollBar1->Name = "Ricksbar";
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete ScrollBar1;
}
```

Delphi Examples:

```
{
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
}
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  MyScrollBar = class(TScrollBar)
  end;

var
  Form1: TForm1;
  ScrollBar1: MyScrollBar;

implementation

{$R *.dfm}

function ComponentToStringProc(Component: TComponent): string;
var
  BinStream:TMemoryStream;
  StrStream: TStringStream;
  s: string;
begin
  BinStream := TMemoryStream.Create;
  try
    StrStream := TStringStream.Create(s);
    try
      BinStream.WriteComponent(Component);
      BinStream.Seek(0, soFromBeginning);
      ObjectBinaryToText(BinStream, StrStream);
      StrStream.Seek(0, soFromBeginning);
      Result:= StrStream.DataString;
    finally
      StrStream.Free;
```

```
      end;
    finally
      BinStream.Free
    end;
  end;

  function StringToComponentProc(Value: string): TComponent;
  var
    StrStream:TStringStream;
    BinStream: TMemoryStream;
  begin
    StrStream := TStringStream.Create(Value);
    try
      BinStream := TMemoryStream.Create;
      try
        ObjectTextToBinary(StrStream, BinStream);
        BinStream.Seek(0, soFromBeginning);
        Result:= BinStream.ReadComponent(nil);
      finally
        BinStream.Free;
      end;
    finally
      StrStream.Free;
    end;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Memo1.Text:= ComponentToStringProc(ScrollBar1);
  end;

  // Edit the scrollbar parameters in the memo, click this
  // button and see the scrollbar change.
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    ScrollBar1.Free;
    ScrollBar1:= (StringToComponentProc(Memo1.Text) as MyScrollBar);
    ScrollBar1.Parent:= Form1;
    ScrollBar1.Visible:= TRUE;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ScrollBar1:= MyScrollBar.Create(Form1);
    ScrollBar1.Parent:= Form1;
    ScrollBar1.Visible:= TRUE;
    ScrollBar1.Top:= 48;
    ScrollBar1.Left:= 250;
    ScrollBar1.Name:= 'Ricksbar';
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    ScrollBar1.Free;
  end;

  initialization

  RegisterClass(MyScrollBar);
```

See Also

TStream (see page 2973)

TWriter (see page 3175)

23.3.42 Classes.ObjectTextToResource Function

Converts a symbolic text representation of an object into an internal binary representation.

Pascal

```
procedure ObjectTextToResource(Input: TStream; Output: TStream); overload;
procedure ObjectTextToResource(Input: TStream; Output: TStream; var OriginalFormat:
TStreamOriginalFormat); overload;
```

C++

```
ObjectTextToResource(TStream Input, TStream Output);
ObjectTextToResource(TStream Input, TStream Output, TStreamOriginalFormat OriginalFormat);
```

File

Classes

Description

Call ObjectTextToResource to create a component from the representation of an object as it is saved in module resource files. ObjectTextToResource is the same as ObjectTextToBinary (see page 3240), except that it skips over any object headers and includes a resource header instead. Use the ReadComponentRes (see page 3245) method of the output stream to obtain the actual object.

ObjectTextToResource converts the text version of a form that can be seen by right clicking the form and choosing View As Text into the binary version which is stored in a form file.

See Also

ReadComponentRes (see page 3245)

ReadComponentResFile (see page 3247)

TResourceStream (see page 2968)

TStream (see page 2973)

23.3.43 Classes.Point Function

Creates a TPoint structure from a pair of coordinates.

Pascal

```
function Point(AX: Integer; AY: Integer): TPoint;
```

C++

```
TPoint Point(int AX, int AY);
```

File

Classes

Description

Call Point to create a TPoint that represents the specified coordinates.

C++ Examples:

```
/*
This example draws a polygon in the specified shape, and
fills it with the color teal.
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
  TPoint points[4];
  points[0] = Point(10,10);
  points[1] = Point(30,10);
  points[2] = Point(130,30);
  points[3] = Point(240,120);
  ((TPaintBox *)Sender)->Canvas->Brush->Color = clTeal;
  ((TPaintBox *)Sender)->Canvas->Polygon(points, 3);
}
```

Delphi Examples:

```
{
This example draws a polygon in the specified shape, and
fills it with the color teal.
}
procedure TForm1.FormPaint(Sender: TObject);
begin
  Canvas.Brush.Color := clTeal;
  Canvas.Polygon([Point(10, 10), Point(30, 10),
    Point(130, 30), Point(240, 120)]);
end;
```

See Also

SmallPoint (see page 3262)

23.3.44 Classes.PointsEqual Function

Indicates whether two points have the same coordinates.

Pascal

```
function PointsEqual(const P1: TPoint; const P2: TPoint): Boolean; overload;
function PointsEqual(const P1: TSmallPoint; const P2: TSmallPoint): Boolean; overload;
```

C++

```
Boolean PointsEqual(const TPoint P1, const TPoint P2);
Boolean PointsEqual(const TSmallPoint P1, const TSmallPoint P2);
```

File

Classes

Description

Call PointsEqual to compare two points. PointsEqual returns true if P1 has the same coordinates as P2, false if the two points differ.

23.3.45 Classes.ReadComponentRes Function

Reads components and their properties from a specified resource.

Pascal

```
function ReadComponentRes(const ResName: string; Instance: TComponent): TComponent;
```

C++

```
TComponent * ReadComponentRes(const AnsiString ResName, TComponent * Instance);
```

File

Classes

Description

Call ReadComponentRes to create a component from its representation in a resource header. The ResName parameter is the name assigned to the resource for the component. The Instance parameter is an instance of the component that should be filled in with the attributes of the saved resource. Instance must be an object of the same class as the saved resource, or one of its descendants. Create an instance of the appropriate type of object, and then call ReadComponentRes to read in its property settings. If Instance is nil (Delphi) or NULL (C++), the object class is read from the resource.

ReadComponentRes returns an instance of the component that was read from the resource.

See Also

ReadComponentResFile (see page 3247)

TResourceStream (see page 2968)

WriteComponentResFile (see page 3266)

ReadComponentResEx (see page 3246)

23.3.46 Classes.ReadComponentResEx Function

Reads a component from a resource.

Pascal

```
function ReadComponentResEx(HInstance: THandle; const ResName: string): TComponent;
```

C++

```
TComponent * ReadComponentResEx(THandle HInstance, const AnsiString ResName);
```

File

Classes

Description

Use ReadComponentResEx to load a component from a resource stored in a particular module (as opposed to ReadComponentRes (see page 3245), which assumes the current module).

The HInstance parameter is the instance handle for the module that stores the component as a resource. The ResName parameter is the name assigned to the resource for the component.

ReadComponentResEx returns an instance of the component that was read from the resource.

See Also

ObjectTextToResource (see page 3244)

ReadComponentResFile (see page 3247)

TResourceStream (see page 2968)

WriteComponentResFile (see page 3266)

ReadComponentRes (see page 3245)

23.3.47 Classes.ReadComponentResFile Function

Reads components and their properties from a specified Windows resource file.

Pascal

```
function ReadComponentResFile(const FileName: string; Instance: TComponent): TComponent;
```

C++

```
TComponent * ReadComponentResFile(const AnsiString FileName, TComponent * Instance);
```

File

Classes

Description

Call ReadComponentResFile to create a component and its children from their representation in a Windowsresource file.

The FileName parameter is the name of the compiled resource file that contains the component resource.

The Instance parameter is an instance of the type of component that should be read. Create an instance of the appropriate type of object, and then call ReadComponentResFile to read in its property settings. If Instance is nil (Delphi) or NULL (C++), the object class is read from the resource. Before passing an Instance parameter of nil (Delphi) or NULL (C++), all classes being read must have been registered using the RegisterClass (see page 3250) or RegisterClasses (see page 3255) routine.

ReadComponentResFile returns an instance of the component that was read from the resource file.

Delphi Examples:

```
{
ReadComponentResFile, WriteComponentResFile example
The following code defines a component class (TStreamable
class) and a persistent class (TContainedClass) that is the
type of a property on the component class. When a button is
clicked, an instance of the component class is created, saved
to a file, deleted, and then loaded from the file. By setting
a breakpoint on the property setter of the TContainedClass
SomeData property, you can watch how the streaming system
sets property values when it reads them from a file.
The following unit defines the classes to stream:
}

{$R-,T-,H+,X+}

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TContainedClass = class(TPersistent)
  private
    FSomeData: Integer;
    procedure SetSomeData(Value: Integer);
```

```
    public
      constructor Create;
      // Only published properties
      // are automatically streamed.
    published
      property SomeData: Integer read FSomeData write SetSomeData;
    end;

    TStreamableClass = class(TComponent)
    private
      FContainedClass: TContainedClass;
    public
      constructor Create(AOwner: TComponent); override;
      destructor Destroy; override;
      // Only published properties
      // are automatically streamed.
    published
      property ContainedClass: TContainedClass read FContainedClass write FContainedClass;
    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TContainedClass.SetSomeData(Value: Integer);
  begin
  { Place a breakpoint here and notice how the data is streamed back. }
    FSomeData := Value;
  end;

  constructor TContainedClass.Create;
  begin
    FSomeData := 42;
  end;

  constructor TStreamableClass.Create(AOwner: TComponent);
  begin
    inherited Create(AOwner);
    FContainedClass := TContainedClass.Create;
  end;

  destructor TStreamableClass.Destroy;
  begin
    FContainedClass.Free;
  end;

  {
  The following OnClick event handler should be added to a
  button on the application's main form. It causes the above
  classes to be streamed out and in. Note that you must add
  StrmExmpl to the uses clause of unit1.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    AClassInstance: TStreamableClass;
  begin
    AClassInstance := TStreamableClass.Create(nil);
    WriteComponentResFile('Test', AClassInstance);
    FreeAndNil(AClassInstance);

    AClassInstance := Classes.ReadComponentResFile('Test', nil) as TStreamableClass;
    FreeAndNil(AClassInstance);
  end;
```

```
initialization
  RegisterClasses([TContainedClass, TStreamableClass]);
```

See Also

ReadComponentRes (see page 3245)

ReadComponentResEx (see page 3246)

TStream (see page 2973)

WriteComponentResFile (see page 3266)

RegisterClass (see page 3250)

RegisterClasses (see page 3255)

23.3.48 Classes.Rect Function

Creates a TRect structure from a set of coordinates.

Pascal

```
function Rect(ALeft: Integer; ATop: Integer; ARight: Integer; ABottom: Integer): TRect;
overload;
function Rect(const ATopLeft: TPoint; const ABottomRight: TPoint): TRect; overload;
```

C++

```
TRect Rect(int ALeft, int ATop, int ARight, int ABottom);
TRect Rect(const TPoint ATopLeft, const TPoint ABottomRight);
```

File

Classes

Description

Call Rect to create a TRect that represents the rectangle with the specified coordinates. Use Rect to construct parameters for functions that require TRect, rather than setting up local variables for each parameter.

C++ Examples:

```
/*
The following code displays the text "Hello, world!" in a
rectangle defined by the coordinates (10, 10) and (100, 100).
After displaying the text with the TextRect method, the code
draws a black, vertical line frame around the rectangle.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TRect TheRect = Rect(10,10,100,100);
  Canvas->TextRect(TheRect, 10, 10, "Hello World");
  Canvas->Brush->Color = clBlack;
  Canvas->FrameRect(TheRect);
}
```

Delphi Examples:

```
{
The following code displays the text "Hello, world!" in a
rectangle defined by the coordinates (10, 10) and (100, 100).
After displaying the text with the TextRect method, the code
draws a black, vertical line frame around the rectangle.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
var TheRect: TRect;
begin
  TheRect := Rect(10,10,100,100);
  Form1.Canvas.TextRect(TheRect,10,10,'Hello, world!');
  Form1.Canvas.Brush.Color := clBlack;
  Form1.Canvas.FrameRect(TheRect);
end;
```

See Also

Bounds (see page 3218)

Point (see page 3244)

23.3.49 Classes.RedirectFixupReferences Function

Pascal

```
procedure RedirectFixupReferences(Root: TComponent; const OldRootName: string; const
NewRootName: string);
```

C++

```
RedirectFixupReferences(TComponent * Root, const AnsiString OldRootName, const AnsiString
NewRootName);
```

File

Classes

Description

This is function Classes.RedirectFixupReferences.

23.3.50 Classes.RegisterClass Function

Registers a class of persistent object so that it's class type can be retrieved.

Pascal

```
procedure RegisterClass(AClass: TPersistentClass);
```

C++

```
RegisterClass(TPersistentClass AClass);
```

File

Classes

Description

Call RegisterClass to register a class with the streaming system. Form classes and component classes that are referenced in a form declaration (instance variables) are automatically registered. Any other classes used by an application must be explicitly registered by calling RegisterClass if instances are to be saved.

Once classes are registered, they can be loaded or saved by the component streaming system. IdentToInt (see page 3230) returns nil (Delphi) or NULL (C++) when passed the class name of an unregistered class, and FindClass (see page 3223) raises an exception for unregistered classes.

The AClass parameter is the class that is descended from TPersistent (see page 2928). Put the call to RegisterClass in a Register procedure. In Delphi, you can also put the call in the initialization section of the unit in which the class is defined. In C++, the call can also go in the namespace of the compilation unit that defines the class.

If the class is already registered, RegisterClass does nothing. If a different class with the same name is already registered, RegisterClass raises an EFilerError (see page 2696) exception.

Note: Registering a component using the RegisterNoIcon (see page 3260) or RegisterComponents (see page 3256) method does not automatically register the class. It is still necessary to call RegisterClass for components.

C++ Examples:

```
/*
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
*/
class MyScrollBar : public TScrollBar
{
__published:    // IDE-managed Components
private:    // User declarations
public:     // User declarations
    __fastcall MyScrollBar(TComponent* Owner);
};

__fastcall MyScrollBar::MyScrollBar(TComponent* Owner)
    : TScrollBar(Owner)
{
}

void RegisterClassesWithStreamingSystem(void)
{
  // Make sure that as part of the startup
  // code our streaming classes are registered
  // with the streaming system.

  #pragma startup RegisterClassesWithStreamingSystem

  Classes::RegisterClass(__classid(MyScrollBar));
}

MyScrollBar *ScrollBar1;

AnsiString __fastcall ComponentToString(TComponent *c)
{
 AnsiString as;
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);

 try
 {
    pms->WriteComponent(c);
    pms->Seek(0, soFromBeginning);
    ObjectBinaryToText(pms, pss);
    pss->Seek(0, soFromBeginning);
    as = pss->DataString;
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 delete pms;
```

```
 delete pss;
 return as;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Memo1->Text = ComponentToString(ScrollBar1);
}

TComponent *__fastcall StringToComponent(AnsiString as)

{
 TMemoryStream* pms = new TMemoryStream();
 TStringStream* pss = new TStringStream(as);
 TComponent *pc;

 try
 {
    ObjectTextToBinary(pss, pms);
    pms->Seek(0, soFromBeginning);
 }
 catch(...)
 {
    ShowMessage("Streaming error.");
 }

 pc = pms->ReadComponent(NULL);
 delete pms;
 delete pss;
 return pc;
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  TComponent *temp = StringToComponent(Memo1->Text);
  ScrollBar1->Free();

  ScrollBar1 = dynamic_cast<MyScrollBar *>(temp);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ScrollBar1 = new MyScrollBar(Form1);
  ScrollBar1->Parent = Form1;
  ScrollBar1->Visible = TRUE;
  ScrollBar1->Top = 48;
  ScrollBar1->Left = 250;
  ScrollBar1->Name = "Ricksbar";
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete ScrollBar1;
}
```

Delphi Examples:

```
{
This example shows how to use the built-in component
streaming support to convert any component into a string and
convert that string back into a component.
}
type
  TForm1 = class(TForm)
    Memo1: TMemo;
```

```
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  MyScrollBar = class(TScrollBar)
  end;

var
  Form1: TForm1;
  ScrollBar1: MyScrollBar;

implementation

{$R *.dfm}

function ComponentToStringProc(Component: TComponent): string;
var
  BinStream:TMemoryStream;
  StrStream: TStringStream;
  s: string;
begin
  BinStream := TMemoryStream.Create;
  try
    StrStream := TStringStream.Create(s);
    try
      BinStream.WriteComponent(Component);
      BinStream.Seek(0, soFromBeginning);
      ObjectBinaryToText(BinStream, StrStream);
      StrStream.Seek(0, soFromBeginning);
      Result:= StrStream.DataString;
    finally
      StrStream.Free;
    end;
  finally
    BinStream.Free
  end;
end;

function StringToComponentProc(Value: string): TComponent;
var
  StrStream:TStringStream;
  BinStream: TMemoryStream;
begin
  StrStream := TStringStream.Create(Value);
  try
    BinStream := TMemoryStream.Create;
    try
      ObjectTextToBinary(StrStream, BinStream);
      BinStream.Seek(0, soFromBeginning);
      Result:= BinStream.ReadComponent(nil);
    finally
      BinStream.Free;
    end;
  finally
    StrStream.Free;
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  Memo1.Text:= ComponentToStringProc(ScrollBar1);
end;

// Edit the scrollbar parameters in the memo, click this
// button and see the scrollbar change.
procedure TForm1.Button2Click(Sender: TObject);
begin
  ScrollBar1.Free;
  ScrollBar1:= (StringToComponentProc(Memo1.Text) as MyScrollBar);
  ScrollBar1.Parent:= Form1;
  ScrollBar1.Visible:= TRUE;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ScrollBar1:= MyScrollBar.Create(Form1);
  ScrollBar1.Parent:= Form1;
  ScrollBar1.Visible:= TRUE;
  ScrollBar1.Top:= 48;
  ScrollBar1.Left:= 250;
  ScrollBar1.Name:= 'Ricksbar';
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  ScrollBar1.Free;
end;

initialization

RegisterClass(MyScrollBar);
```

See Also

EFilerError (see page 2696)

IdentToInt (see page 3230)

RegisterClasses (see page 3255)

RegisterComponents (see page 3256)

RegisterNoIcon (see page 3260)

UnRegisterClass (see page 3263)

23.3.51 Classes.RegisterClassAlias Function

Registers a class that is identical to another class except for the name.

Pascal

```
procedure RegisterClassAlias(AClass: TPersistentClass; const Alias: string);
```

C++

```
RegisterClassAlias(TPersistentClass AClass, const AnsiString Alias);
```

File

Classes

Description

Component writers call RegisterClassAlias to register custom classes that are identical to another persistent class except for the name. Registering a class allows it to be loaded and saved by the component streaming system.

The AClass parameter identifies the previously registered persistent class to which the new class is identical. The Alias parameter specifies the class name of the new class.

See Also

RegisterClass (see page 3250)

23.3.52 Classes.RegisterClasses Function

Registers a set of classes.

Pascal

```
procedure RegisterClasses(AClasses: array of TPersistentClass);
```

C++

```
RegisterClasses(array of TPersistentClass AClasses);
```

File

Classes

Description

Call RegisterClasses to register a set of custom classes in a single line. Each class is registered by calling RegisterClass (see page 3250). Unregistered classes can't be loaded or saved by the component streaming system.

Note: In C++, the AClasses parameter identifies the classes to be registered These must be descendants of TPersistent (see page 2928). The AClasses_Size parameter indicates the index of the last class in AClasses (one less than the total number of classes).

C++ Examples:

```
/*
The following example allows users to type the name of a
graphics class into an edit control and when a button is
pressed, the name of the default extension for that class is
displayed in another edit control.  In the form's OnCreate
event handler (or a similar place) you must register the
graphics classes so that GetClass can find them.
*/
TMetaClass *MetaClass;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClassFinder *myClassFinder = new TClassFinder(MetaClass, False);
  TGraphicClass mygraphclass = (TGraphicClass) (myClassFinder->GetClass(Edit2->Text));
//  TGraphicClass mygraphclass = (TGraphicClass) GetClass(Edit2->Text);
  Edit1->Text = GraphicExtension(mygraphclass);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MetaClass = __classid(TIcon); // ico
  RegisterClasses(&MetaClass, 0);
```

```
    MetaClass = __classid(Graphics::TBitmap); // bmp
    RegisterClasses(&MetaClass, 0);
    MetaClass = __classid(TMetafile); // emf
    RegisterClasses(&MetaClass, 0);
    MetaClass = __classid(TShape);  // nographic extension
    RegisterClasses(&MetaClass, 0);
    MetaClass = __classid(TImage); // nographic extension
    RegisterClasses(&MetaClass, 0);
  }
```

Delphi Examples:

```
{
The following example allows users to type the name of a
graphics class into an edit control and when a button is
pressed, the name of the default extension for that class is
displayed in another edit control.  In the form's OnCreate
event handler (or a similar place) you must register the
graphics classes so that GetClass can find them.
}

uses ExtCtrls;

procedure TForm1.Button1Click(Sender: TObject);
var
  myClassFinder: TClassFinder;
  mygraphclass: TGraphicClass;
begin
  { make sure the classes are registered so that GetClass will work -- }
  { Usually, this goes in the initialization section where it is only executed once }
  RegisterClasses([TIcon, TBitmap, TButton, TShape, TImage, TMetafile]);
  try
    myClassFinder := TClassFinder.Create(TIcon, False);
    mygraphclass := TGraphicClass(myClassFinder.GetClass(Edit2.Text));
//    mygraphclass := TGraphicClass(GetClass(Edit2.Text));  // works as well
    Edit1.Text := GraphicExtension(mygraphclass);
  except
    on EAccessViolation do
      MessageDlg('Invalid class name.', mtError, [mbOK], 0)
  end;
end;
```

See Also

IdentToInt (see page 3230)

UnRegisterModuleClasses (see page 3265)

23.3.53 Classes.RegisterComponents Function

Registers a set of components so that they all appear on the same page of the component palette.

Pascal

```
procedure RegisterComponents(const Page: string; ComponentClasses: array of TComponentClass);
```

C++

```
RegisterComponents(const AnsiString Page, array of TComponentClass ComponentClasses);
```

File

Classes

Description

Call RegisterComponents to install a set of components in the IDE. Once a component is registered, it appears on the component palette, where it can be selected and placed on forms or data modules. Registered components can communicate with the Object Inspector to allow the user to get and set properties and events.

Set the Page parameter to the name of the page on the component palette where the components should appear. If the named page already exists, the components are added to that page. If the named page does not exist, a new palette page with that name is created.

Pass the components to be registered in the ComponentClasses parameter.

Note: In C++, the ComponentClasses_Size parameter indicates the index of the last class in the ComponentClasses array (one less than the number of classes).

Call RegisterComponents from the implementation of the Register procedure in one of the units that defines the custom components. The units that define the components must then be compiled into a package and the package must be installed before the custom components appear in the component palette.

Note: After components are registered, users can move them to different palette pages. Once this happens, the component always appears on the new page. Calling RegisterComponents a second time does not influence the page on which the component appears.

Delphi Examples:

```
{
The following code registers several components, some on the
MySystem palette page that already exists, and some on a new
palette page that is created when when the components are
installed.
}
type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TDemoLabel = class(TGraphicControl)
  private
    FFocusControl: TWinControl;
    procedure SetFocusControl(Value: TWinControl);
    procedure CMDialogChar(var Message: TCMDialogChar); message CM_DIALOGCHAR;
    procedure CMTextChanged(var Message: TMessage); message CM_TEXTCHANGED;
  protected
    procedure Notification(AComponent: TComponent;
      Operation: TOperation); override;
    procedure Paint; override;
  public
    constructor Create(AOwner: TComponent); override;
  published
    property Caption;
    property Color;
    property FocusControl: TWinControl read FFocusControl write SetFocusControl;
    property Font;
    property ParentColor;
    property ParentFont;
  end;

procedure Register;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

{ TDemoLabel }

constructor TDemoLabel.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FComponentStyle := FComponentStyle - [csInheritable];
end;

procedure TDemoLabel.Notification(AComponent: TComponent;
  Operation: TOperation);
begin
  inherited Notification(AComponent, Operation);
  if (Operation = opRemove) and (AComponent = FFocusControl) then
    FFocusControl := nil;
end;

procedure TDemoLabel.SetFocusControl(Value: TWinControl);
begin
  FFocusControl := Value;

  { Calling FreeNotification ensures that this component will receive an
    opRemove when Value is either removed from its owner or when it is
    destroyed. }

  Value.FreeNotification(Self);
end;

procedure TDemoLabel.Paint;
var
  Rect: TRect;
begiN
  Rect := ClientRect;
  Canvas.Font := Font;
  Canvas.Brush.Color := Color;
  Canvas.FillRect(Rect);
  DrawText(Canvas.Handle, PChar(Caption), Length(Caption), Rect,
    DT_EXPANDTABS or DT_WORDBREAK or DT_LEFT);
end;

procedure TDemoLabel.CMDialogChar(var Message: TCMDialogChar);
begin
  if (FFocusControl <> nil) and Enabled and
    IsAccel(Message.CharCode, Caption) then
    with FFocusControl do
      if CanFocus then
      begin
        SetFocus;
        Message.Result := 1;
      end;
end;

procedure TDemoLabel.CMTextChanged(var Message: TMessage);
begin
  inherited;
  Invalidate;
end;

procedure Register;
begin
```

```
  Classes.RegisterComponents('MySystem', [TDemoLabel]);
  Classes.RegisterComponents('Samples', [TDemoLabel]);
end;
```

See Also

UnRegisterClasses (see page 3264)

23.3.54 Classes.RegisterFindGlobalComponentProc Function

Pascal

```
procedure RegisterFindGlobalComponentProc(AFindGlobalComponent: TFindGlobalComponent);
```

C++

```
RegisterFindGlobalComponentProc(TFindGlobalComponent AFindGlobalComponent);
```

File

Classes

Description

This is function Classes.RegisterFindGlobalComponentProc.

23.3.55 Classes.RegisterIntegerConsts Function

Registers conversion functions for string identifiers that represent type values.

Pascal

```
procedure RegisterIntegerConsts(AIntegerType: Pointer; AIdentToInt: TIdentToInt; AIntToIdent:
TIntToIdent);
```

C++

```
RegisterIntegerConsts(void * AIntegerType, TIdentToInt AIdentToInt, TIntToIdent AIntToIdent);
```

File

Classes

Description

RegisterIntegerConsts (which can only be called in Delphi) registers conversion functions for type identifiers. Call RegisterIntegerConsts from the initialization section of the unit that defines an integer-based type and a set of strings that represent the values of that type.

AIntegerType is a pointer to the type information for the integer-based type whose values are represented as strings. Its value can be obtained from the base type using the TypeInfo function.

AIdentToInt is the conversion function that converts strings that are symbolic representations of values to the corresponding integers.

AIntToIdent is the conversion function that converts values that are instances of the base type to the corresponding string representation.

Once the conversion functions are registered, string identifiers can represent type values and the conversion functions are used

to convert the string identifiers into the underlying integer values. For example, the TColor type uses this system to convert between the type constants in the Graphics (see page 14387) unit and numeric TColor values.

See Also

TColor

TypeInfo

IdentToInt (see page 3230)

IntToIdent (see page 3232)

UnregisterIntegerConsts (see page 3264)

23.3.56 Classes.RegisterNoIcon Function

Registers a set of components but does not add them to the component palette.

Pascal

```
procedure RegisterNoIcon(const ComponentClasses: array of TComponentClass);
```

C++

```
RegisterNoIcon(const array of TComponentClass ComponentClasses);
```

File

Classes

Description

Call RegisterNoIcon to register a set of custom components so that objects from that class can use the Object Inspector. Because the components are not added to the component palette, instances of components registered using RegisterNoIcon must be explicitly created by calling the constructor, usually from another component.

The ComponentClasses parameter is an array of custom classes that are descended from TComponent (see page 2762).

Note: In C++, the ComponentClasses_Size parameter is index of the last class in the array (one less than the number of classes).

Put the call to RegisterNoIcon in the Register procedure.

To add the components to the component palette, use RegisterComponents (see page 3256) instead.

See Also

RegisterNonActiveX (see page 3260)

23.3.57 Classes.RegisterNonActiveX Function

Prevents a set of components from being made available to the ActiveX wizard.

Pascal

```
procedure RegisterNonActiveX(const ComponentClasses: array of TComponentClass; AxRegType:
TActiveXRegType);
```

C++

```
RegisterNonActiveX(const array of TComponentClass ComponentClasses, TActiveXRegType AxRegType);
```

File

Classes

Description

Call RegisterNonActiveX to prevent a registered set of custom components from being converted into ActiveX controls using the ActiveX wizard. The components must still be registered using RegisterComponents (see page 3256) or RegisterNoIcon (see page 3260) if they are to work with the Object Inspector.

The ComponentClasses parameter is an array of custom classes that are descended from TComponent (see page 2762).

Note: In C++, the ComponentClasses_Size is the index of the last class in ComponentClasses (one less than the number of classes).

The AxRegType parameter indicates whether only the components in the ComponentClasses array should be blocked from the ActiveX wizard (when AxRegType is axrComponentOnly), or whether their descendants should be prohibited from becoming ActiveX controls as well (when AxRegType is axrIncludeDescendants).

Put the call to RegisterNonActiveX in the Register procedure.

See Also

RegisterNoIcon (see page 3260)

Elements of an ActiveX Control

Generating an ActiveX Control from a VCL Control

23.3.58 Classes.RemoveFixupReferences Function

Pascal

```
procedure RemoveFixupReferences(Root: TComponent; const RootName: string);
```

C++

```
RemoveFixupReferences(TComponent * Root, const AnsiString RootName);
```

File

Classes

Description

This is function Classes.RemoveFixupReferences.

23.3.59 Classes.RemoveFixups Function

Pascal

```
procedure RemoveFixups(Instance: TPersistent);
```

C++

```
RemoveFixups(TPersistent * Instance);
```

File

Classes

Description

This is function Classes.RemoveFixups.

23.3.60 Classes.SmallPoint Function

Creates a TSmallPoint structure from a pair of coordinates.

Pascal

```
function SmallPoint(AX: SmallInt; AY: SmallInt): TSmallPoint;
```

C++

```
TSmallPoint SmallPoint(SmallInt AX, SmallInt AY);
```

File

Classes

Description

Call SmallPoint to create a TSmallPoint that represents the specified coordinates.

See Also

Point (see page 3244)

23.3.61 Classes.StartClassGroup Function

Starts a class group derived from TPersistent (see page 2928).

Pascal

```
procedure StartClassGroup(AClass: TPersistentClass);
```

C++

```
StartClassGroup(TPersistentClass AClass);
```

File

Classes

Description

The streaming system allows the classes that can be loaded and saved to be registered in separate groups. This allows the IDE to distinguish between cross-platform and Windows-only classes. StartClassGroup creates a new group of classes, and adds the class specified by AClass to that group.

See Also

ActivateClassGroup (see page 3215)

GroupDescendantsWith

23.3.62 Classes.TestStreamFormat Function

Identifies the format of a form file when reading it from a stream.

Pascal

```
function TestStreamFormat(Stream: TStream): TStreamOriginalFormat;
```

C++

```
TStreamOriginalFormat TestStreamFormat(TStream Stream);
```

File

Classes

Description

Call TestStreamFormat to check the signature of a stream that reads a form file to determine whether it was stored in binary or text format.

Stream is the stream to check.

TestStreamFormat checks the signature of the stream and returns sofBinary if the form file was saved in binary format, sofText if it was saved in text format, and sofUnknown if it could not tell. A return value of sofUnknown usually indicates that the stream is not reading a form file or that the stream is not positioned at the start of the file.

See Also

IdentToInt (see page 3230)

UnRegisterModuleClasses (see page 3265)

23.3.63 Classes.UnRegisterClass Function

Unregisters an object class.

Pascal

```
procedure UnRegisterClass(AClass: TPersistentClass);
```

C++

```
UnRegisterClass(TPersistentClass AClass);
```

File

Classes

Description

Call UnRegisterClass to unregister an object class. When a class is unregistered, it can't be loaded or saved by the component streaming system.

After unregistering a class, its name can be reused to register another object class.

See Also

IdentToInt (see page 3230)

UnRegisterModuleClasses (see page 3265)

23.3.64 Classes.UnRegisterClasses Function

Unregisters a set of classes.

Pascal

```
procedure UnRegisterClasses(AClasses: array of TPersistentClass);
```

C++

```
UnRegisterClasses(array of TPersistentClass AClasses);
```

File

Classes

Description

Call UnRegisterClasses to unregister several persistent object classes in a single line. When a class is unregistered, it can't be loaded or saved by the component streaming system.

The AClasses parameter specifies the classes (descendants of TPersistent (see page 2928)) to unregister.

Note: In C++, the AClasses_Size parameter indicates the index of the last class in the AClasses array (one less than the number of classes).

After unregistering a class, its name can be reused to register another object class.

See Also

IdentToInt (see page 3230)

UnRegisterModuleClasses (see page 3265)

23.3.65 Classes.UnregisterFindGlobalComponentProc Function

Pascal

```
procedure UnregisterFindGlobalComponentProc(AFindGlobalComponent: TFindGlobalComponent);
```

C++

```
UnregisterFindGlobalComponentProc(TFindGlobalComponent AFindGlobalComponent);
```

File

Classes

Description

This is function Classes.UnregisterFindGlobalComponentProc.

23.3.66 Classes.UnregisterIntegerConsts Function

Unregisters conversion functions for string identifiers that represent type values.

Pascal

```
procedure UnregisterIntegerConsts(AIntegerType: Pointer; AIdentToInt: TIdentToInt;
AIntToIdent: TIntToIdent);
```

C++

```
UnregisterIntegerConsts(void * AIntegerType, TIdentToInt AIdentToInt, TIntToIdent AIntToIdent);
```

File

Classes

Description

UnregisterIntegerConsts, which can only be called in Delphi, unregisters conversion functions that were previously registered by a call to RegisterIntegerConsts (see page 3259). Call UnregisterIntegerConsts from the finalization section of the unit that defines the constants and registers the conversion functions in its initialization section. Once the conversion functions are unregistered, string identifiers can't be used to represent type values.

AIntegerType is a pointer to the type information for the integer-based type whose values are represented as strings. Its value can be obtained from the base type using the TypeInfo function.

AIdentToInt is the conversion function that converts strings that are symbolic representations of values to the corresponding integers.

AIntToIdent is the conversion function that converts values that are instances of the base type to the corresponding string representation.

See Also

RegisterIntegerConsts (see page 3259)

TypeInfo

23.3.67 Classes.UnRegisterModuleClasses Function

Unregisters all classes defined in a specified module.

Pascal

```
procedure UnRegisterModuleClasses(Module: HMODULE);
```

C++

```
UnRegisterModuleClasses(HMODULE Module);
```

File

Classes

Description

Call UnRegisterModuleClasses to unregister all object classes that were registered by the module with the handle specified by the Module parameter. When a class is unregistered, it can't be loaded or saved by the component streaming system.

After unregistering a class, its name can be reused to register another object class.

See Also

IdentToInt (see page 3230)

UnRegisterClasses (see page 3264)

23.3.68 Classes.WriteComponentResFile Function

Writes components and their properties to a file using a resource file format.

Pascal

```
procedure WriteComponentResFile(const FileName: string; Instance: TComponent);
```

C++

```
WriteComponentResFile(const AnsiString FileName, TComponent * Instance);
```

File

Classes

Description

Use WriteComponentResFile to save the component specified by the Instance parameter to the specified file, storing it in a resource-file format.

To read a component written with WriteComponentResFile, call ReadComponentResFile (see page 3247).

Delphi Examples:

```
{
ReadComponentResFile, WriteComponentResFile example
The following code defines a component class (TStreamable
class) and a persistent class (TContainedClass) that is the
type of a property on the component class. When a button is
clicked, an instance of the component class is created, saved
to a file, deleted, and then loaded from the file. By setting
a breakpoint on the property setter of the TContainedClass
SomeData property, you can watch how the streaming system
sets property values when it reads them from a file.
The following unit defines the classes to stream:
}

{$R-,T-,H+,X+}

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TContainedClass = class(TPersistent)
  private
    FSomeData: Integer;
    procedure SetSomeData(Value: Integer);
  public
    constructor Create;
    // Only published properties
    // are automatically streamed.
  published
    property SomeData: Integer read FSomeData write SetSomeData;
  end;

  TStreamableClass = class(TComponent)
  private
```

```
    FContainedClass: TContainedClass;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    // Only published properties
    // are automatically streamed.
  published
    property ContainedClass: TContainedClass read FContainedClass write FContainedClass;
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TContainedClass.SetSomeData(Value: Integer);
begin
{ Place a breakpoint here and notice how the data is streamed back. }
  FSomeData := Value;
end;

constructor TContainedClass.Create;
begin
  FSomeData := 42;
end;

constructor TStreamableClass.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FContainedClass := TContainedClass.Create;
end;

destructor TStreamableClass.Destroy;
begin
  FContainedClass.Free;
end;

{
The following OnClick event handler should be added to a
button on the application's main form. It causes the above
classes to be streamed out and in. Note that you must add
StrmExmpl to the uses clause of unit1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  AClassInstance: TStreamableClass;
begin
  AClassInstance := TStreamableClass.Create(nil);
  WriteComponentResFile('Test', AClassInstance);
  FreeAndNil(AClassInstance);

  AClassInstance := Classes.ReadComponentResFile('Test', nil) as TStreamableClass;
  FreeAndNil(AClassInstance);
end;

initialization
  RegisterClasses([TContainedClass, TStreamableClass]);
```

See Also

ReadComponentResFile (see page 3247)

23.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TActiveXRegType (see page 3268)	TActiveXRegType is used when excluding a component class from the ActiveX Wizard.
	TAlignment (see page 3269)	TAlignment specifies how text is aligned within a control.
	TBiDiMode (see page 3269)	TBiDiMode specifies the reading order (bidirectional mode) of a control.
	TCollectionNotification (see page 3270)	TCollectionNotification indicates the type of change that is made to the items in a collection.
	TDuplicates (see page 3271)	TDuplicates indicates the response when an application attempts to add a duplicate entry to a list.
	TFilerFlag (see page 3271)	TFilerFlag indicates information about how a filer should read or write a component and TFilerFlags (see page 3289) is a set of TFilerFlag values.
	THelpType (see page 3272)	THelpType indicates whether components invoke the help system using a context ID or a keyword.
	TListAssignOp (see page 3273)	TListAssignOp indicates how two lists should be merged.
	TListNotification (see page 3274)	TListNotification defines list processing action types.
	TOperation (see page 3275)	TOperation represents the types of operations whose occurrence is broadcast by the Notification method.
	TSeekOrigin (see page 3276)	TSeekOrigin indicates where to start a seek operation.
	TStreamOriginalFormat (see page 3276)	TStreamOriginalFormat indicates the format in which a form file is saved.
	TStreamOwnership (see page 3277)	TStreamOwnership indicates whether a stream object should be freed by the object that uses it.
	TThreadPriority (see page 3278)	TThreadPriority indicates the scheduling priority of a thread object on Windows.
	TValueType (see page 3279)	TValueType defines the kinds of values written to and read from filer objects.
	TVerticalAlignment (see page 3280)	TVerticalAlignment specifies how text is vertically aligned within a control.

Records

	Name	Description
	TIdentMapEntry (see page 3273)	TIdentMapEntry is used to perform ident string to integer value mapping.
	TStringItem (see page 3277)	TStringItem defines a TStringList (see page 3006) entry.
	TSynchronizeRecord (see page 3278)	This is record Classes.TSynchronizeRecord.

23.4.1 Classes.TActiveXRegType Enumeration

TActiveXRegType is used when excluding a component class from the ActiveX Wizard.

Pascal

```
TActiveXRegType = (
  axrComponentOnly,
  axrIncludeDescendants
);
```

C++

```
enum TActiveXRegType {
  axrComponentOnly,
  axrIncludeDescendants
};
```

File

Classes

Description

The TActiveXRegType type indicates whether a component's descendants should be excluded from the ActiveX Wizard when the component is excluded. The following table lists the possible values:

Value	Meaning
axrComponentOnly	Only the component class is blocked from appearing in the ActiveX Wizard.
axrIncludeDescendants	The component class and any of its descendants are blocked from the Wizard.

23.4.2 Classes.TAlignment Enumeration

TAlignment specifies how text is aligned within a control.

Pascal

```
TAlignment = (
  taLeftJustify,
  taRightJustify,
  taCenter
);
```

C++

```
enum TAlignment {
  taLeftJustify,
  taRightJustify,
  taCenter
};
```

File

Classes

Description

The following are possible values of TAlignment:

Value	Meaning
taLeftJustify	Text is left-justified: Lines all begin at the left edge of the control.
taCenter	Text is centered in the control.
taRightJustify	Text is right-justified: Lines all end at the right edge of the control.

See Also

Alignment

23.4.3 Classes.TBiDiMode Enumeration

TBiDiMode specifies the reading order (bidirectional mode) of a control.

Pascal

```
TBiDiMode = (
  bdLeftToRight,
  bdRightToLeft,
  bdRightToLeftNoAlign,
  bdRightToLeftReadingOnly
);
```

C++

```
enum TBiDiMode {
  bdLeftToRight,
  bdRightToLeft,
  bdRightToLeftNoAlign,
  bdRightToLeftReadingOnly
};
```

File

Classes

Description

TBiDiMode determines the reading order of the text, placement of the vertical scrollbar, and any alignment modifications. It is used to tailor the user interface for locales where text is bi-directional.

The following table describes the possible values:

Value	Meaning
bdLeftToRight	Reading order is left to right. Alignment is not changed. The vertical scroll bar appears on the right edge of the control.
bdRightToLeft	Reading order is right to left. Alignment is changed. The vertical scroll bar appears on the left edge of the control.
bdRightToLeftNoAlign	Reading order is right to left. Alignment is not changed. The vertical scroll bar appears on the left edge of the control.
bdRightToLeftReadingOnly	Reading order is right to left. Alignment and scroll bar are not changed.

23.4.4 Classes.TCollectionNotification Enumeration

TCollectionNotification indicates the type of change that is made to the items in a collection.

Pascal

```
TCollectionNotification = (
  cnAdded,
  cnExtracting,
  cnDeleting
);
```

C++

```
enum TCollectionNotification {
  cnAdded,
  cnExtracting,
  cnDeleting
};
```

File

Classes

Description

The following table lists the values for TCollectionNotification:

Value	Meaning
cnAdded	An item was just added to the collection.
cnExtracting	An item is about to be removed from the collection (but not freed).
cnDeleting	An item is about to be removed from the collection and then freed.

See Also

TCollection

23.4.5 Classes.TDuplicates Enumeration

TDuplicates indicates the response when an application attempts to add a duplicate entry to a list.

Pascal

```
TDuplicates = (
  dupIgnore,
  dupAccept,
  dupError
);
```

C++

```
enum TDuplicates {
  dupIgnore,
  dupAccept,
  dupError
};
```

File

Classes

Description

TDuplicates can have one of the following values:

Value	Meaning
dupIgnore	Ignore attempts to add duplicate entries (do not add the duplicate).
dupAccept	Allow the list to contain duplicate entries (add the duplicate).
dupError	Raise an exception when the application tries to add a duplicate.

See Also

TStrings

23.4.6 Classes.TFilerFlag Enumeration

TFilerFlag indicates information about how a filer should read or write a component and TFilerFlags (see page 3289) is a set of TFilerFlag values.

Pascal

```
TFilerFlag = (
  ffInherited,
  ffChildPos,
  ffInline
);
```

C++

```
enum TFilerFlag {
  ffInherited,
  ffChildPos,
  ffInline
};
```

File

Classes

Description

TFilerFlag values are written to form files as a prefix to saved components. They indicate any special status of the component that the streaming system must take into account. TFilerFlags (see page 3289) is a set containing all the flags (if any) that pertain to a single component.

The following table lists the possible TFilerFlag values:

Value	Meaning
ffInherited	The component is an inherited form.
ffChildPos	The order in which this component is created is important.
ffInline	The component is a top-level component such as a form or data module.

23.4.7 Classes.THelpType Enumeration

THelpType indicates whether components invoke the help system using a context ID or a keyword.

Pascal

```
THelpType = (
  htKeyword,
  htContext
);
```

C++

```
enum THelpType {
  htKeyword,
  htContext
};
```

File

Classes

Description

These are the possible values of the THelpType type:

Value	Meaning
htKeyword	Help topics are identified by a string. The HelpKeyword property is used to identify help topics. This mechanism is available for all help systems.
htContext	Help topics are identified by a context ID. The HelpContext property is used to identify help topics. This mechanism is not available for all help systems, but is typically faster for those systems that support it.

23.4.8 Classes.TIdentMapEntry Record

TIdentMapEntry is used to perform ident string to integer value mapping.

Pascal

```
TIdentMapEntry = record
  Value: Integer;
  Name: String;
end;
```

C++

```
struct TIdentMapEntry {
  int Value;
  AnsiString Name;
};
```

File

Classes

Description

The TIdentMapEntry type declares a record of two fields. **Name** gives the ident name as a string. **Value** gives the integer value of this ident.

RegisterIntegerConsts (see page 3259) function registers pairs of ident strings and their corresponding values. It uses IdentToInt (see page 3230) and IntToIdent (see page 3232) to perform the mappings between the ident names and their values.

These routines in turn use TIdentMapEntry arrays to hold these mappings.

For example, the Graphics (see page 14387) unit uses TIdentMapEntry internally to hold the mapping of color names to integer values, such as 'clMaroon' to $000080.

See Also

IdentToInt (see page 3230)

IntToIdent (see page 3232)

RegisterIntegerConsts (see page 3259)

23.4.9 Classes.TListAssignOp Enumeration

TListAssignOp indicates how two lists should be merged.

Pascal

```
TListAssignOp = (
  laCopy,
  laAnd,
  laOr,
  laXor,
  laSrcUnique,
  laDestUnique
);
```

C++

```
enum TListAssignOp {
  laCopy,
  laAnd,
  laOr,
  laXor,
  laSrcUnique,
  laDestUnique
};
```

File

Classes

Description

TListAssignOps describes how the Assign method combines a source and destination list:

Value	Meaning
laAnd	Removes all elements from the destination list that do not appear in the source list. The destination list ends up containing the intersection of the two lists.
laCopy	Overwrites the destination list with the source list.
laDestUnique	Removes all elements from the destination list that appear in the source list. The destination list ends up containing the elements unique to the original destination list. (same as laOr followed by laXor)
laOr	Adds any elements from the source list that do not already appear in the destination list. The destination list ends up containing the union of the two lists.
laSrcUnique	Replaces the destination list with those elements of the source list that do not appear in the destination list. (same as laAnd followed by laXor)
laXor	Removes all elements from the destination list that appear in the source list and adds any elements from the source list that do not appear in the original destination list. The destination list ends up with the elements unique to both the source and destination lists.

23.4.10 Classes.TListNotification Enumeration

TListNotification defines list processing action types.

Pascal

```
TListNotification = (
  lnAdded,
  lnExtracted,
  lnDeleted
);
```

C++

```
enum TListNotification {
```

```
    lnAdded,
    lnExtracted,
    lnDeleted
  };
```

File

Classes

Description

The TListNotification type enumerates the different event types for a list object.

lnAdded means an entry was added to the list.

lnExtracted means an entry was deleted from the list. The entry was specified by reference.

lnDeleted means an entry was deleted from the list. The entry was specified by list position.

TListNotification is principally used internally by TList (see page 2868).

See Also

TList (see page 2868)

23.4.11 Classes.TOperation Enumeration

TOperation represents the types of operations whose occurrence is broadcast by the Notification method.

Pascal

```
  TOperation = (
    opInsert,
    opRemove
  );
```

C++

```
  enum TOperation {
    opInsert,
    opRemove
  };
```

File

Classes

Description

The TOperation type is used in the Notification method to indicate whether the notification indicates the creation or deletion of an object. The following table gives the possible values:

Value	Meaning
opInsert	The specified object has just been created.
opRemove	The specified object has been destroyed, and its memory is about to be freed.

See Also

TComponent

23.4.12 Classes.TSeekOrigin Enumeration

TSeekOrigin indicates where to start a seek operation.

Pascal

```
TSeekOrigin = (
  soBeginning,
  soCurrent,
  soEnd
);
```

C++

```
enum TSeekOrigin {
  soBeginning,
  soCurrent,
  soEnd
};
```

File

Classes

Description

The TSeekOrigin type indicates where to start a seek origin and the direction in which to seek. The following table lists the possible values:

Value	Meaning
soBeginning	Seek from the beginning of the resource. The seek operation moves to a specified position (offset), which must be greater than or equal to zero.
soCurrent	Seek from the current position in the resource. The seek operation moves to an offset from the current position (position + offset). The offset is positive to move forward, negative to move backward.
soEnd	Seek from the end of the resource. The seek operation moves to an offset from the end of the resource, where the offset is expressed as a negative value because it is moving toward the beginning of the resource.

Warning: Using these constants may generate unwanted/wrong code, because an overloaded seek method will be used, which uses `offest : longint`, thus limiting seekrange to 2GB.

23.4.13 Classes.TStreamOriginalFormat Enumeration

TStreamOriginalFormat indicates the format in which a form file is saved.

Pascal

```
TStreamOriginalFormat = (
  sofUnknown,
  sofBinary,
  sofText
);
```

C++

```
enum TStreamOriginalFormat {
  sofUnknown,
  sofBinary,
```

```
  sofText
};
```

File

Classes

Description

Form files can be saved in binary format or as a text file, depending on the options set in the Environment Options dialog. TStreamOriginalFormat specifies the format used for saving a form file. It is one of the following values:

Value	Meaning
sofUnknown	The format of the file from which a form is being read or to which it is being written is unknown. If the format can't be deduced from the stream, an exception is raised.
sofBinary	The form is being read from or written to a form file that uses binary format.
sofText	The form is being read from or written to a text file.

23.4.14 Classes.TStreamOwnership Enumeration

TStreamOwnership indicates whether a stream object should be freed by the object that uses it.

Pascal

```
TStreamOwnership = (
  soReference,
  soOwned
);
```

C++

```
enum TStreamOwnership {
  soReference,
  soOwned
};
```

File

Classes

Description

The following are possible values of TStreamOwnership:

Value	Meaning
soReference	The stream is an independent object with a different Owner. The object that uses the stream does not free it when it is destroyed.
soOwned	The stream is owned by the object that uses it (and exists solely for that object). The object that uses the stream frees the associated stream when it no longer needs to use the stream.

23.4.15 Classes.TStringItem Record

TStringItem defines a TStringList (see page 3006) entry.

Pascal

```
TStringItem = record
  FString: string;
  FObject: TObject;
end;
```

C++

```
struct TStringItem {
  AnsiString FString;
  TObject * FObject;
};
```

File

Classes

Description

The TStringItem type defines a data structure used internally in Delphi.

It is used by TStringList (see page 3006) to hold a list entry.

FString defines string for a list entry.

FObject points to an optional TObject associated with the string.

See Also

TStringList (see page 3006)

23.4.16 Classes.TSynchronizeRecord Record

Pascal

```
TSynchronizeRecord = record
  FThread: TObject;
  FMethod: TThreadMethod;
  FSynchronizeException: TObject;
end;
```

C++

```
struct TSynchronizeRecord {
  TObject * FThread;
  TThreadMethod FMethod;
  TObject * FSynchronizeException;
};
```

File

Classes

Description

This is record Classes.TSynchronizeRecord.

23.4.17 Classes.TThreadPriority Enumeration

TThreadPriority indicates the scheduling priority of a thread object on Windows.

Pascal

```
TThreadPriority = (
  tpIdle,
  tpLowest,
  tpLower,
  tpNormal,
  tpHigher,
  tpHighest,
  tpTimeCritical
);
```

C++

```
enum TThreadPriority {
  tpIdle,
  tpLowest,
  tpLower,
  tpNormal,
  tpHigher,
  tpHighest,
  tpTimeCritical
};
```

File

Classes

Description

The following are possible values of TThreadPriority:

Values	Meaning
tpIdle	The thread executes only when the system is idle. The system will not interrupt other threads to execute a thread with tpIdle priority.
tpLowest	The thread's priority is two points below normal.
tpLower	The thread's priority is one point below normal.
tpNormal	The thread has normal priority.
tpHigher	The thread's priority is one point above normal.
tpHighest	The thread's priority is two points above normal.
tpTimeCritical	The thread gets highest priority.

See Also

TThread

23.4.18 Classes.TValueType Enumeration

TValueType defines the kinds of values written to and read from filer objects.

Pascal

```
TValueType = (
  vaNull,
  vaList,
  vaInt8,
  vaInt16,
  vaInt32,
```

```
    vaExtended,
    vaString,
    vaIdent,
    vaFalse,
    vaTrue,
    vaBinary,
    vaSet,
    vaLString,
    vaNil,
    vaCollection,
    vaSingle,
    vaCurrency,
    vaDate,
    vaWString,
    vaInt64,
    vaUTF8String,
    vaDouble
  );
```

C++

```
  enum TValueType {
    vaNull,
    vaList,
    vaInt8,
    vaInt16,
    vaInt32,
    vaExtended,
    vaString,
    vaIdent,
    vaFalse,
    vaTrue,
    vaBinary,
    vaSet,
    vaLString,
    vaNil,
    vaCollection,
    vaSingle,
    vaCurrency,
    vaDate,
    vaWString,
    vaInt64,
    vaUTF8String,
    vaDouble
  };
```

File

Classes

Description

TValueType is the type used by the TReader (see page 2933) and TWriter (see page 3175) filer objects to type check property values that are read from and written to streams when streaming components and their properties.

See Also

TReader

TWriter

23.4.19 Classes.TVerticalAlignment Enumeration

TVerticalAlignment specifies how text is vertically aligned within a control.

Pascal

```
TVerticalAlignment = (
  taAlignTop,
  taAlignBottom,
  taVerticalCenter
);
```

C++

```
enum TVerticalAlignment {
  taAlignTop,
  taAlignBottom,
  taVerticalCenter
};
```

File

Classes

Description

The following are possible values of TVerticalAlignment:

Value	Meaning
taAlignTop	Text is aligned to the top edge of the control.
taAlignCenter	Text is vertically centered in the control.
taVerticalJustify	Text is vertically-justified: Lines are spaced out to fill the height of the control.

23.5 Types

The following table lists types in this documentation.

Types

Name	Description
PPointerList (see page 3283)	PPointerList is a pointer to a TPointerList (see page 3299) type.
PStringItem (see page 3284)	PStringItem is a TStringItem (see page 3277) pointer.
PStringItemList (see page 3284)	PStringItemList is a TStringItemList (see page 3304) pointer.
PSynchronizeRecord (see page 3285)	This is type Classes.PSynchronizeRecord.
TAncestorNotFoundEvent (see page 3285)	TAncestorNotFoundEvent is a procedure called by the TReader (see page 2933) class OnAncestorNotFound event.
TBasicActionClass (see page 3285)	TBasicActionClass defines the metaclass for TBasicAction (see page 2707).
TBasicActionLinkClass (see page 3286)	TBasicActionLinkClass defines the metaclass for TBasicActionLink (see page 2720).
TCollectionItemClass (see page 3286)	TCollectionItemClass defines the metaclass for TCollectionItem (see page 2755).
TComponentClass (see page 3287)	TComponentClass defines the metaclass for TComponent (see page 2762).
TComponentName (see page 3287)	TComponentName is the type for the Name property of all components.
TComponentState (see page 3287)	TComponentState is a set of flags that describe the current state of a component.
TComponentStyle (see page 3288)	TComponentStyle is a set of flags that describe the current Style of a component.
TCreateComponentEvent (see page 3289)	TCreateComponentEvent is a type of procedure called by the TReader (see page 2933) class OnCreateComponentEvent event.
TFilerFlags (see page 3289)	TFilerFlags is the set of TFilerFlag (see page 3271) values.
TFindAncestorEvent (see page 3290)	TFindAncestorEvent is a procedure called by the TWriter (see page 3175) class OnFindAncestor event.
TFindComponentClassEvent (see page 3290)	This is type Classes.TFindComponentClassEvent.
TFindComponentInstanceEvent (see page 3291)	This is type Classes.TFindComponentInstanceEvent.
TFindGlobalComponent (see page 3291)	This is type Classes.TFindGlobalComponent.
TFindMethodEvent (see page 3291)	TFindMethodEvent is a procedure called by the TReader (see page 2933) class OnFindMethod event.
TFindMethodInstanceEvent (see page 3292)	TFindMethodInstanceEvent is a procedure called by the TReader (see page 2933) class OnFindMethodInstance event.
TFindMethodNameEvent (see page 3292)	TFindMethodNameEvent is a procedure called by the TWriter (see page 3175) class OnFindMethodName event.
TGetChildProc (see page 3292)	TGetChildProc is a type of procedure called used internally by the TComponent (see page 2762) class.
TGetClass (see page 3293)	TGetClass is a callback procedure used by the TClassFinder (see page 2728) class.
TGetLookupInfoEvent (see page 3293)	This is type Classes.TGetLookupInfoEvent.
TGetModuleProc (see page 3294)	This is type Classes.TGetModuleProc.
TGetStrProc (see page 3294)	TGetStrProc is a general purpose type of callback procedure invoked for each of a set of strings.
THelpContext (see page 3295)	THelpContext is the range of values that a HelpContext value may have.
TIdentToInt (see page 3295)	TIdentToInt is a type of callback function used by the IdentToInt (see page 3230) routine.
TIntToIdent (see page 3296)	TIntToIdent is a type of callback function used by the IntToIdent (see page 3232) routine.
TIsUniqueGlobalComponentName (see page 3296)	TIsUniqueGlobalComponentName is a type of function used by IsUniqueGlobalComponentName (see page 3234).
TLeftRight (see page 3297)	TLeftRight is a subset of TAlignment (see page 3269) that allows left or right justification.
TListSortCompare (see page 3297)	TListSortCompare is a type of callback function called used internally by the TList (see page 2868) class.
TNotifyEvent (see page 3298)	TNotifyEvent is used for events that do not require parameters.

TParserErrorEvent (see page 3298)	TParserErrorEvent is a callback procedure used by the TClassFinder (see page 2728) class.
TPersistentClass (see page 3299)	TPersistentClass defines the metaclass for TPersistent (see page 2928).
TPointerList (see page 3299)	TPointerList is an array of pointers.
TReadComponentsProc (see page 3300)	TReadComponentsProc is a type of callback procedure called used internally by the TReader (see page 2933) class ReadComponents method.
TReaderError (see page 3300)	TReaderError is a type of callback procedure called used internally by the TReader (see page 2933) class ReadComponents method.
TReaderProc (see page 3301)	TReaderProc is a type of callback procedure called used by TFiler (see page 2834) and descendant DefineProperty methods.
TReferenceNameEvent (see page 3301)	TReferenceNameEvent is a procedure called by the TReader (see page 2933) class OnReferenceNameEvent event.
TSetNameEvent (see page 3302)	TSetNameEvent is a procedure called by the TReader (see page 2933) class OnSetNameEvent event.
TShiftState (see page 3302)	TShiftState indicates the state of the Alt, Ctrl, and Shift keys and the mouse buttons.
TShortCut (see page 3303)	TStringsDefined (see page 3306) defines an enumeration of string delimiter names.
TStreamProc (see page 3304)	TStreamProc is a type of callback procedure called used by TFiler (see page 2834) and descendant DefineBinaryProperty methods.
TStringItemList (see page 3304)	TStringItemList defines an array of TStringItems.
TStringListSortCompare (see page 3305)	TStringListSortCompare is a type of callback function called used internally by the TStringList (see page 3006) class.
TStringsDefined (see page 3306)	This is type Classes.TStringsDefined.
TThreadMethod (see page 3306)	This is type Classes.TThreadMethod.
TTopBottom (see page 3306)	TTopBottom is a subset of TVerticalAlignment (see page 3280) that allows top or bottom alignment.
TWndMethod (see page 3307)	TWndMethod is the type for Windows message procedures.
TWriterProc (see page 3307)	TWriterProc is a type of callback procedure called used by TFiler (see page 2834) and descendant DefineProperty methods.

23.5.1 Classes.PPointerList Type

PPointerList is a pointer to a TPointerList (see page 3299) type.

Pascal

```
PPointerList = ^TPointerList;
```

C++

```
^TPointerList PPointerList;
```

File

Classes

Description

PPointerList is principally used by the TList (see page 2868) property of the TList (see page 2868) class to provide a pointer to the list of pointers that TList (see page 2868) holds.

See Also

TList (see page 2868)

TPointerList (see page 3299)

23.5.2 Classes.PStringItem Type

PStringItem is a TStringItem (see page 3277) pointer.

Pascal

```
PStringItem = ^TStringItem;
```

C++

```
^TStringItem PStringItem;
```

File

Classes

Description

PStringItem is used internally by TStringList (see page 3006) to point to a TStringItem (see page 3277) value.

TStringItem (see page 3277) comprises :

FString defines string for a list entry.

FObject points to an optional TObject associated with the string.

See Also

TStringItem (see page 3277)

TStringList (see page 3006)

23.5.3 Classes.PStringItemList Type

PStringItemList is a TStringItemList (see page 3304) pointer.

Pascal

```
PStringItemList = ^TStringItemList;
```

C++

```
^TStringItemList PStringItemList;
```

File

Classes

Description

PStringItemList is used internally by TStringList (see page 3006) to refer to its internal array of TStringItem (see page 3277) values. This array holds the list of strings.

TStringItem (see page 3277) comprises :

FString defines string for a list entry.

FObject points to an optional TObject associated with the string.

See Also

TStringItem (see page 3277)

TStringList (see page 3006)

23.5.4 Classes.PSynchronizeRecord Type

Pascal

```
PSynchronizeRecord = ^TSynchronizeRecord;
```

C++

```
^TSynchronizeRecord PSynchronizeRecord;
```

File

Classes

Description

This is type Classes.PSynchronizeRecord.

23.5.5 Classes.TAncestorNotFoundEvent Type

TAncestorNotFoundEvent is a procedure called by the TReader (see page 2933) class OnAncestorNotFound event.

Pascal

```
TAncestorNotFoundEvent = procedure (Reader: TReader; const ComponentName: string;
ComponentClass: TPersistentClass; var Component: TComponent) of object;
```

C++

```
(Reader: TReader; const ComponentName: string; ComponentClass: TPersistentClass; var
Component: TComponent) ( TAncestorNotFoundEvent)();
```

File

Classes

Description

TAncestorNotFoundEvent is for internal use only.

See Also

OnAncestorNotFound (see page 2961)

TReader (see page 2933)

23.5.6 Classes.TBasicActionClass Type

TBasicActionClass defines the metaclass for TBasicAction (see page 2707).

Pascal

```
TBasicActionClass = class of TBasicAction;
```

C++

```
TBasicAction TBasicActionClass;
```

File

Classes

Description

TBasicActionClass is the metaclass for TBasicAction (see page 2707). Its value is the class reference for TBasicAction (see page 2707) or for one of its descendants.

See Also

TBasicAction (see page 2707)

23.5.7 Classes.TBasicActionLinkClass Type

TBasicActionLinkClass defines the metaclass for TBasicActionLink (see page 2720).

Pascal

```
TBasicActionLinkClass = class of TBasicActionLink;
```

C++

```
TBasicActionLink TBasicActionLinkClass;
```

File

Classes

Description

TBasicActionLinkClass is the metaclass for TBasicActionLink (see page 2720). Its value is the class reference for TBasicActionLink (see page 2720) or for one of its descendants.

See Also

TBasicActionLink (see page 2720)

23.5.8 Classes.TCollectionItemClass Type

TCollectionItemClass defines the metaclass for TCollectionItem (see page 2755).

Pascal

```
TCollectionItemClass = class of TCollectionItem;
```

C++

```
TCollectionItem TCollectionItemClass;
```

File

Classes

Description

TCollectionItemClass is the metaclass for TCollectionItem (see page 2755). Its value is the class reference for TCollectionItem (see page 2755) or for one of its descendants.

23.5.9 Classes.TComponentClass Type

TComponentClass defines the metaclass for TComponent (see page 2762).

Pascal

```
TComponentClass = class of TComponent;
```

C++

```
TComponent * TComponentClass;
```

File

Classes

Description

TComponentClass is the metaclass for TComponent (see page 2762). Its value is the class reference for TComponent (see page 2762) or for one of its descendants.

23.5.10 Classes.TComponentName Type

TComponentName is the type for the Name property of all components.

Pascal

```
TComponentName = type string;
```

C++

```
type string TComponentName;
```

File

Classes

Description

TComponentName is a string that represents the name of a component.

23.5.11 Classes.TComponentState Type

TComponentState is a set of flags that describe the current state of a component.

Pascal

```
TComponentState = set of (csLoading, csReading, csWriting, csDestroying, csDesigning,
csAncestor, csUpdating, csFixups, csFreeNotification, csInline, csDesignInstance);
```

C++

```
(csLoading, csReading, csWriting, csDestroying, csDesigning, csAncestor, csUpdating, csFixups,
csFreeNotification, csInline, csDesignInstance) TComponentState;
```

File

Classes

Description

TComponentState defines the set of possible state flags for the ComponentState property of a TComponent (see page 2762) instance. The following table lists the possible values for the TComponentState type and the meaning corresponding to each flag:

Flag	Component state
csAncestor	The component was introduced in an ancestor form. Only set if csDesigning is also set.
csDesigning	The component is in a form being manipulated by the form designer.
csDestroying	The component is about to be destroyed.
csFixups	The component is linked to a component in another form that has not yet been loaded. This flag is cleared when all pending fix-ups are resolved.
csFreeNotification	One or more other components have requested that this component notify them when it is destroyed. This flag is set when another component calls this component's FreeNotification method.
csInline	The component is a top-level component that can be modified at design time and also embedded in a form. This flag is used to identify nested frames while loading and saving.
csLoading	A filer object is currently loading the component. This flag is set when the component is first created and not cleared until the component and all its children are fully loaded (when the Loaded method is called).
csReading	The component is reading its property values from a stream. Note that the csLoading flag is always set as well when csReading is set. That is, csReading is set for the subinterval of the time when a component is loading that covers reading in property values.
csUpdating	The component is being updated to reflect changes in an ancestor form. Only set if csAncestor is also set.
csWriting	The component is writing its property values to a stream.
csDesignInstance	The component is the root object in a designer. For example, it is set for a frame when you are designing it, but not on a frame that acts like a component. This flag always appears with csDesigning.

23.5.12 Classes.TComponentStyle Type

TComponentStyle is a set of flags that describe the current Style of a component.

Pascal

```
TComponentStyle = set of (csInheritable, csCheckPropAvail, csSubComponent, csTransient);
```

C++

```
(csInheritable, csCheckPropAvail, csSubComponent, csTransient) TComponentStyle;
```

File

Classes

Description

The following table indicates the meaning of the various TComponentStyle flags:

Value	Meaning
csInheritable	Descendant form types can inherit from the component. If any of the components in a form do not have the csInheritable style, the form cannot be used as the ancestor of an inherited form.
csCheckPropAvail	The component needs to check its properties for readability. This is only used for COM controls (on Windows), where the Object Inspector cannot tell directly that a property is readable, and therefore displayable.

csSubComponent	The component is a subcomponent of the component that is the value of its Owner property. Unlike top-level components, subcomponents are not saved with the form or data module in which they reside. Instead, a subcomponent appears as the value of a published property of its Owner, and its published properties and events are saved in the form file with the owning component.
csTransient	The component is a temporary object that should not be saved in a form file.

23.5.13 Classes.TCreateComponentEvent Type

TCreateComponentEvent is a type of procedure called by the TReader (see page 2933) class OnCreateComponentEvent event.

Pascal

```
TCreateComponentEvent = procedure (Reader: TReader; ComponentClass: TComponentClass; var
Component: TComponent) of object;
```

C++

```
(Reader: TReader; ComponentClass: TComponentClass; var Component: TComponent) (
TCreateComponentEvent)();
```

File

Classes

Description

TCreateComponentEvent is for internal use only.

See Also

TReader (see page 2933)

23.5.14 Classes.TFilerFlags Type

TFilerFlags is the set of TFilerFlag (see page 3271) values.

Pascal

```
TFilerFlags = set of TFilerFlag;
```

C++

```
TFilerFlag TFilerFlags;
```

File

Classes

Description

TFilerFlags values are written to form files as a prefix to saved components. They indicate any special status of the component that the streaming system must take into account. TFilerFlags is a set containing all the flags that might pertain to a single component.

The following table lists the set of TFilerFlags values:

Value	Meaning
ffInherited	The component is an inherited form.
ffChildPos	The order in which this component is created is important.
ffInline	The component is a top-level component such as a form or data module.

See Also

TFilerFlag (see page 3271)

23.5.15 Classes.TFindAncestorEvent Type

TFindAncestorEvent is a procedure called by the TWriter (see page 3175) class OnFindAncestor event.

Pascal

```
TFindAncestorEvent = procedure (Writer: TWriter; Component: TComponent; const Name: string;
var Ancestor, RootAncestor: TComponent) of object;
```

C++

```
(Writer: TWriter; Component: TComponent; const Name: string; var Ancestor, RootAncestor:
TComponent) ( TFindAncestorEvent)();
```

File

Classes

Description

TFindAncestorEvent is for internal use only.

See Also

TWriter (see page 3175)

23.5.16 Classes.TFindComponentClassEvent Type

Pascal

```
TFindComponentClassEvent = procedure (Reader: TReader; const ClassName: string; var
ComponentClass: TComponentClass) of object;
```

C++

```
(Reader: TReader; const ClassName: string; var ComponentClass: TComponentClass) (
TFindComponentClassEvent)();
```

File

Classes

Description

This is type Classes.TFindComponentClassEvent.

23.5.17 Classes.TFindComponentInstanceEvent Type

Pascal

```
TFindComponentInstanceEvent = procedure (Reader: TReader; const Name: string; var Instance:
Pointer) of object;
```

C++

```
(Reader: TReader; const Name: string; var Instance: Pointer) ( TFindComponentInstanceEvent)();
```

File

Classes

Description

This is type Classes.TFindComponentInstanceEvent.

23.5.18 Classes.TFindGlobalComponent Type

Pascal

```
TFindGlobalComponent = function (const Name: string): TComponent;
```

C++

```
(const Name: string): TComponent ( TFindGlobalComponent)();
```

File

Classes

Description

This is type Classes.TFindGlobalComponent.

23.5.19 Classes.TFindMethodEvent Type

TFindMethodEvent is a procedure called by the TReader (see page 2933) class OnFindMethod event.

Pascal

```
TFindMethodEvent = procedure (Reader: TReader; const MethodName: string; var Address: Pointer;
var Error: Boolean) of object;
```

C++

```
(Reader: TReader; const MethodName: string; var Address: Pointer; var Error: Boolean) (
TFindMethodEvent)();
```

File

Classes

Description

TFindMethodEvent is for internal use only.

See Also

TReader (see page 2933)

23.5.20 Classes.TFindMethodInstanceEvent Type

TFindMethodInstanceEvent is a procedure called by the TReader (see page 2933) class OnFindMethodInstance event.

Pascal

```
TFindMethodInstanceEvent = procedure (Reader: TReader; const MethodName: string; var Method:
TMethod; var Error: Boolean) of object;
```

C++

```
(Reader: TReader; const MethodName: string; var Method: TMethod; var Error: Boolean) (
TFindMethodInstanceEvent)();
```

File

Classes

Description

TFindMethodInstanceEvent is for internal use only.

See Also

TReader (see page 2933)

23.5.21 Classes.TFindMethodNameEvent Type

TFindMethodNameEvent is a procedure called by the TWriter (see page 3175) class OnFindMethodName event.

Pascal

```
TFindMethodNameEvent = procedure (Writer: TWriter; Method: TMethod; var MethodName: string) of
object;
```

C++

```
(Writer: TWriter; Method: TMethod; var MethodName: string) ( TFindMethodNameEvent)();
```

File

Classes

Description

TFindMethodNameEvent is for internal use only.

See Also

TWriter (see page 3175)

23.5.22 Classes.TGetChildProc Type

TGetChildProc is a type of procedure called used internally by the TComponent (see page 2762) class.

Pascal

```
TGetChildProc = procedure (Child: TComponent) of object;
```

C++

```
(Child: TComponent) ( TGetChildProc)();
```

File

Classes

Description

TGetChildProc is for internal use only.

See Also

TComponent (see page 2762)

23.5.23 Classes.TGetClass Type

TGetClass is a callback procedure used by the TClassFinder (see page 2728) class.

Pascal

```
TGetClass = procedure (AClass: TPersistentClass) of object;
```

C++

```
(AClass: TPersistentClass) ( TGetClass)();
```

File

Classes

Description

TGetClass is a type of call back procedure that is called for each class reference found by the TClassFinder (see page 2728) class GetClasses method.

The class is passed to this procedure in **AClass**.

See Also

TClassFinder (see page 2728)

GetClasses (see page 2731)

23.5.24 Classes.TGetLookupInfoEvent Type

Pascal

```
TGetLookupInfoEvent = procedure (var Ancestor: TPersistent; var Root, LookupRoot,
RootAncestor: TComponent) of object;
```

C++

```
(var Ancestor: TPersistent; var Root, LookupRoot, RootAncestor: TComponent) (
TGetLookupInfoEvent)();
```

File

Classes

Description

This is type Classes.TGetLookupInfoEvent.

23.5.25 Classes.TGetModuleProc Type

Pascal

```
TGetModuleProc = procedure (const FileName, UnitName, FormName, DesignClass: string;
CoClasses: TStrings) of object;
```

C++

```
(const FileName, UnitName, FormName, DesignClass: string; CoClasses: TStrings) (
TGetModuleProc)();
```

File

Classes

Description

This is type Classes.TGetModuleProc.

23.5.26 Classes.TGetStrProc Type

TGetStrProc is a general purpose type of callback procedure invoked for each of a set of strings.

Pascal

```
TGetStrProc = procedure (const S: string) of object;
```

C++

```
(const S: string) ( TGetStrProc)();
```

File

Classes

Description

TGetChildProc (see page 3292) is used by a number of routines, such as -- to define the callback procedure to be called for each of a set of strings.

The string value is passed to the procedure in the **S** parameter.

See Also

GetCursorValues (see page 6251)

GetColorValues (see page 14619)

GetCharsetValues (see page 14615)

23.5.27 Classes.THelpContext Type

THelpContext is the range of values that a HelpContext value may have.

Pascal

```
THelpContext = -MaxLongint..MaxLongint;
```

C++

```
-MaxLongint..MaxLongint THelpContext;
```

File

Classes

Description

Help context specifies the help context for the action when it is set. THelpContext is the return type from the HelpContext property of a visual controls.

See Also

HelpContext (see page 311)

23.5.28 Classes.TIdentToInt Type

TIdentToInt is a type of callback function used by the IdentToInt (see page 3230) routine.

Pascal

```
TIdentToInt = function (const Ident: string; var Int: Longint): Boolean;
```

C++

```
(const Ident: string; var Int: Longint): Boolean ( TIdentToInt)();
```

File

Classes

Description

IdentToInt (see page 3230) provides the underlying translation from string identifiers to integers that occurs, for example, when you register a mapping using the RegisterIntegerConsts (see page 3259) procedure in Delphi.

Ident is the string identifier to translate.

Int returns the corresponding integer value.

See Also

RegisterIntegerConsts (see page 3259)

IntToIdent (see page 3232)

TIntToIdent (see page 3296)

23.5.29 Classes.TIntToIdent Type

TIntToIdent is a type of callback function used by the IntToIdent (see page 3232) routine.

Pascal

```
TIntToIdent = function (Int: Longint; var Ident: string): Boolean;
```

C++

```
(Int: Longint; var Ident: string): Boolean ( TIntToIdent)();
```

File

Classes

Description

TIntToIdent provides the underlying translation from integers to string identifiers that occurs, for example, when you register a mapping using the RegisterIntegerConsts (see page 3259) procedure in Delphi.

Int is the integer value to translate.

Ident is the corresponding string value.

See Also

RegisterIntegerConsts (see page 3259)

IntToIdent (see page 3232)

TIdentToInt (see page 3295)

23.5.30 Classes.TIsUniqueGlobalComponentName Type

TIsUniqueGlobalComponentName is a type of function used by IsUniqueGlobalComponentName (see page 3234).

Pascal

```
TIsUniqueGlobalComponentName = function (const Name: string): Boolean;
```

C++

```
(const Name: string): Boolean ( TIsUniqueGlobalComponentName)();
```

File

Classes

Description

TIsUniqueGlobalComponentName is a type cast of the IsUniqueGlobalComponentName (see page 3234) function.

IsUniqueGlobalComponentName (see page 3234) is used to determine whether a proposed component name is already used on any component on any form, data module, or other top-level container component.

See Also

IsUniqueGlobalComponentName (see page 3234)

23.5.31 Classes.TLeftRight Type

TLeftRight is a subset of TAlignment (see page 3269) that allows left or right justification.

Pascal

```
TLeftRight = taLeftJustify..taRightJustify;
```

C++

```
taLeftJustify..taRightJustify TLeftRight;
```

File

Classes

Description

TLeftRight is a subset of the TAlignment (see page 3269) type that allows left or right justification, but no center justification. The following table lists the possible values.

Value	Meaning
taLeftJustify	Left-justified: The left edges line up.
taRightJustify	Right-justified: The right edges line up.

See Also

TAlignment

23.5.32 Classes.TListSortCompare Type

TListSortCompare is a type of callback function called used internally by the TList (see page 2868) class.

Pascal

```
TListSortCompare = function (Item1, Item2: Pointer): Integer;
```

C++

```
(Item1, Item2: Pointer): Integer ( TListSortCompare)();
```

File

Classes

Description

You define a TListSortCompare function when you need to use the TList (see page 2868) Sort method. It is invoked by the internal implementation of the Sort method to sort the TList (see page 2868) elements into sequence. This user supplied aspect of the sort process is required since TList (see page 2868) is a list of pointers with no inherent sort sequence.

Item1 and **Item2** are 2 elements from the list. When these are passed to the TListSortCompare function, the Sort method is asking which order they should be in. The comparison returns a value determined by the relative values of **Item1** and **Item2**, as shown in this table:

Value	Description
>0 (positive)	**Item1** is less than **Item2**

0	**Item1** is equal to **Item2**
<0 (negative)	**Item1** is greater than **Item2**

See Also

Sort (see page 2892)

TList (see page 2868)

23.5.33 Classes.TNotifyEvent Type

TNotifyEvent is used for events that do not require parameters.

Pascal

```
TNotifyEvent = procedure (Sender: TObject) of object;
```

C++

```
(Sender: TObject) ( TNotifyEvent)();
```

File

Classes

Description

The TNotifyEvent type is the type for events that have no event-specific parameters. These events simply notify the component that a specific event occurred. For example, OnClick, which is of type TNotifyEvent, notifies the control that a click event occurred on the control.

The **Sender** parameter is the object whose event handler is called. For example, with the OnClick event of a button, the **Sender** parameter is the button component that is clicked.

23.5.34 Classes.TParserErrorEvent Type

TParserErrorEvent is a callback procedure used by the TClassFinder (see page 2728) class.

Pascal

```
TParserErrorEvent = procedure (Sender: TObject; const Message: string; var Handled: Boolean)
of object;
```

C++

```
(Sender: TObject; const Message: string; var Handled: Boolean) ( TParserErrorEvent)();
```

File

Classes

Description

TParserErrorEvent is a type of call back procedure that is called by the TParser (see page 2915) class OnError event.

Sender is a reference to the TParser (see page 2915) instance that has encountered an error and invoked the callback procedure.

Message is the error message.

Handled is a set to false if you want the error to raise an exception. If you want processing to continue, set this value to true.

See Also

TParser (see page 2915)

OnError (see page 2927)

23.5.35 Classes.TPersistentClass Type

TPersistentClass defines the metaclass for TPersistent (see page 2928).

Pascal

```
TPersistentClass = class of TPersistent;
```

C++

```
TPersistent * TPersistentClass;
```

File

Classes

Description

TPersistentClass is the metaclass for TPersistent (see page 2928). Its value is the class reference for TPersistent (see page 2928) or for one of its descendants.

See Also

TPersistent (see page 2928)

23.5.36 Classes.TPointerList Type

TPointerList is an array of pointers.

Pascal

```
TPointerList = array[0..MaxListSize - 1] of Pointer;
```

C++

```
array[0..MaxListSize - 1] of Pointer TPointerList;
```

File

Classes

Description

The TList (see page 2868) property of the TList (see page 2868) class returns a PPointerList (see page 3283) pointer to the list of pointers that TList (see page 2868) holds. TPointerList describes this array of pointers.

See Also

TList (see page 2868)

PPointerList (see page 3283)

23.5.37 Classes.TReadComponentsProc Type

TReadComponentsProc is a type of callback procedure called used internally by the TReader (see page 2933) class ReadComponents method.

Pascal

```
TReadComponentsProc = procedure (Component: TComponent) of object;
```

C++

```
(Component: TComponent) ( TReadComponentsProc)();
```

File

Classes

Description

TReadComponentsProc is for internal use only.

See Also

ReadComponents (see page 2946)

TReader (see page 2933)

23.5.38 Classes.TReaderError Type

TReaderError is a type of callback procedure called used internally by the TReader (see page 2933) class ReadComponents method.

Pascal

```
TReaderError = procedure (Reader: TReader; const Message: string; var Handled: Boolean) of
object;
```

C++

```
(Reader: TReader; const Message: string; var Handled: Boolean) ( TReaderError)();
```

File

Classes

Description

TReaderError is a type of call back procedure that is called by the TReader (see page 2933) class OnError event.

Reader is a reference to the TReader (see page 2933) instance that has encountered an error and invoked the callback procedure.

Message is the error message.

Handled is a set to false if you want the error to raise an exception. If you want processing to continue, set this value to true.

See Also

OnError (see page 2961)

TReader (see page 2933)

23.5.39 Classes.TReaderProc Type

TReaderProc is a type of callback procedure called used by TFiler (see page 2834) and descendant DefineProperty methods.

Pascal

```
TReaderProc = procedure (Reader: TReader) of object;
```

C++

```
(Reader: TReader) ( TReaderProc)();
```

File

Classes

Description

DefineProperty introduces an abstract method that allows descendants to read or write data like a property.

Each descendant filer object declares a DefineProperty method that defines the data that the descendant object will read or write as if the data were a property.

TReaderProc is the callback procedure parameter in that DefineProperty. When called, it supplies the TFiler (see page 2834) or descendant object, allowing read access.

See Also

DefineProperty (see page 2838)

DefineProperty (see page 2940)

DefineProperty (see page 3180)

TFiler (see page 2834)

TReader (see page 2933)

TWriter (see page 3175)

23.5.40 Classes.TReferenceNameEvent Type

TReferenceNameEvent is a procedure called by the TReader (see page 2933) class OnReferenceNameEvent event.

Pascal

```
TReferenceNameEvent = procedure (Reader: TReader; var Name: string) of object;
```

C++

```
(Reader: TReader; var Name: string) ( TReferenceNameEvent)();
```

File

Classes

Description

TReferenceNameEvent is for internal use only.

See Also

OnReferenceNameEvent

TReader (see page 2933)

23.5.41 Classes.TSetNameEvent Type

TSetNameEvent is a procedure called by the TReader (see page 2933) class OnSetNameEvent event.

Pascal

```
TSetNameEvent = procedure (Reader: TReader; Component: TComponent; var Name: string) of object;
```

C++

```
(Reader: TReader; Component: TComponent; var Name: string) ( TSetNameEvent)();
```

File

Classes

Description

TSetNameEvent is for internal use only.

See Also

OnSetNameEvent

TReader (see page 2933)

23.5.42 Classes.TShiftState Type

TShiftState indicates the state of the Alt, Ctrl, and Shift keys and the mouse buttons.

Pascal

```
TShiftState = set of (ssShift, ssAlt, ssCtrl, ssLeft, ssRight, ssMiddle, ssDouble);
```

C++

```
(ssShift, ssAlt, ssCtrl, ssLeft, ssRight, ssMiddle, ssDouble) TShiftState;
```

File

Classes

Description

The TShiftState type is used by key-event and mouse-event handlers to determine the state of the Alt, Ctrl, and Shift keys and the state of the mouse buttons when the event occurs. It is a set of flags that indicate the following:

Value	Meaning
ssShift	The Shift key is held down.
ssAlt	The Alt key is held down.
ssCtrl	The Ctrl key is held down.
ssLeft	The left mouse button is held down.

ssRight	The right mouse button is held down.
ssMiddle	The middle mouse button is held down.
ssDouble	The mouse was double-clicked.

See Also

TControl

TControl

TControl

OnKeyDown

OnKeyUp

23.5.43 Classes.TShortCut Type

TStringsDefined (see page 3306) defines an enumeration of string delimiter names.

Pascal

```
TShortCut = Low(Word)..High(Word);
```

C++

```
Low(Word)..High(Word) TShortCut;
```

File

Classes

Description

The TStringsDefined (see page 3306) type is used internally by Delphi as the set of possible string delimiters. The values are :

Value	Meaning
sdDelimiter	Separator between strings in a list. Such as read/set by Delimiter and used in DelimitedText.
sdQuoteChar	Quote character around strings in a list. Such as read/set by QuoteChar and used in DelimitedText.
sdNameValueSeparator	Defines the character that separates name and value pair strings (normally '='). Such as read/set by the NameValueSeparator property. Used by Names, Values and ValueFromIndex
sdLineBreak	Defines the line break character(s). Such as read/set by the LineBreak property.
sdStrictDelimiter	Defines whether only the specified sdDelimiter can be used to separate strings in a DelimitedText, or whether blanks or a non printable character can be used as well.

Note: References to TStrings (see page 3029) equally apply to TStringList (see page 3006).

See Also

DelimitedText (see page 3074)

Delimiter (see page 3074)

LineBreak (see page 3075)

NameValueSeparator (see page 3077)

Names (see page 3075)

QuoteChar (see page 3080)

StrictDelimiter (see page 3080)

TStrings (see page 3029)

TStringList (see page 3006)

ValueFromIndex (see page 3085)

Values (see page 3086)

23.5.44 Classes.TStreamProc Type

TStreamProc is a type of callback procedure called used by TFiler (see page 2834) and descendant DefineBinaryProperty methods.

Pascal

```
TStreamProc = procedure (Stream: TStream) of object;
```

C++

```
(Stream: TStream) ( TStreamProc)();
```

File

Classes

Description

DefineBinaryProperty introduces an abstract method that allows descendants to read or write binary data like a property.

Each descendant Filer object declares a DefineProperty method that defines the data that the descendant object will read or write as if the data were a property.

TStreamProc is the callback procedure parameter in that DefineBinaryProperty. When called, it supplies the TFiler (see page 2834) or descendant object, allowing binary data read and write access.

See Also

DefineBinaryProperty (see page 2837)

DefineBinaryProperty (see page 2939)

DefineBinaryProperty (see page 3179)

TFiler (see page 2834)

TReader (see page 2933)

TWriter (see page 3175)

23.5.45 Classes.TStringItemList Type

TStringItemList defines an array of TStringItems.

Pascal

```
TStringItemList = array[0..MaxListSize] of TStringItem;
```

C++

```
array[0..MaxListSize] of TStringItem TStringItemList;
```

File

Classes

Description

TStringItemList is used internally by TStringList (see page 3006) to refer to its internal array of TStringItem (see page 3277) values. This array holds the list of strings.

TStringItem (see page 3277) comprises :

FString defines string for a list entry.

FObject points to an optional TObject associated with the string.

See Also

TStringItem (see page 3277)

TStringList (see page 3006)

23.5.46 Classes.TStringListSortCompare Type

TStringListSortCompare is a type of callback function called used internally by the TStringList (see page 3006) class.

Pascal

```
TStringListSortCompare = function (List: TStringList; Index1, Index2: Integer): Integer;
```

C++

```
(List: TStringList; Index1, Index2: Integer): Integer ( TStringListSortCompare)();
```

File

Classes

Description

You define a TStringListSortCompare function when you need to use the TStringList (see page 3006) CustomSort method. The procedure defined by TStringListSortCompare is invoked by CustomSort to sort the TStringList (see page 3006) elements into sequence.

CustomSort is used instead of Sort when the sort sequence required differs from the default alphanumeric sequence.

List is the list of strings.

Index1 and **Index2** are indexes into the list. When these are passed to the TListSortCompare (see page 3297) function, the CustomSort method is asking which order they should be in.

You return 0 if the entry referred to by **Index1** equals the entry referred to by **Index2**

You return less than 0 if the entry referred to by **Index1** is less than the entry referred to by **Index2**

You return a number above 0 if the entry referred to by **Index1** is greater than the entry referred to by **Index2**

Note: One of the principle advantages of the CustomSort method is that a TStringList (see page 3006) may be sorted according to the objects optionally attachable to each string.

See Also

CustomSort (see page 3014)

TStringList (see page 3006)

23.5.47 Classes.TStringsDefined Type

Pascal

```
TStringsDefined = set of (sdDelimiter, sdQuoteChar, sdNameValueSeparator, sdLineBreak,
sdStrictDelimiter);
```

C++

```
(sdDelimiter, sdQuoteChar, sdNameValueSeparator, sdLineBreak, sdStrictDelimiter)
TStringsDefined;
```

File

Classes

Description

This is type Classes.TStringsDefined.

23.5.48 Classes.TThreadMethod Type

Pascal

```
TThreadMethod = procedure of object;
```

C++

```
( TThreadMethod)();
```

File

Classes

Description

This is type Classes.TThreadMethod.

23.5.49 Classes.TTopBottom Type

TTopBottom is a subset of TVerticalAlignment (see page 3280) that allows top or bottom alignment.

Pascal

```
TTopBottom = taAlignTop..taAlignBottom;
```

C++

```
taAlignTop..taAlignBottom TTopBottom;
```

File

Classes

Description

TTopBottom is a subset of the TVerticalAlignment (see page 3280) type that allows top or bottom alignment, but no vertical center alignment. The following table lists the possible values.

Value	Meaning
taAlignTop	Text is aligned to the top edge of the control.
taAlignBottom	Text is aligned to the bottom edge of the control.

See Also

TVerticalAlignment (see page 3280)

23.5.50 Classes.TWndMethod Type

TWndMethod is the type for Windows message procedures.

Pascal

```
TWndMethod = procedure (var Message: TMessage) of object;
```

C++

```
(var Message: TMessage) ( TWndMethod)();
```

File

Classes

Description

Each Window has a message procedure that responds to calls from Windows. These calls can originate from Windows (such as paint messages), from user actions (such as mouse messages), or from your messages that you send in your own code.

TWndMethod is the type for VCL-based message procedures. Message procedures of this type can be assigned to controls using the WindowProc method. In addition, a window that implements this procedure but which is not associated with a control can be created using the AllocateHWnd (see page 3217) function.

The Message parameter encapsulates the message to which the procedure responds. It contains fields for the incoming message parameters and a result field that holds the return value.

See Also

TControl (see page 5724)

AllocateHWnd (see page 3217)

23.5.51 Classes.TWriterProc Type

TWriterProc is a type of callback procedure called used by TFiler (see page 2834) and descendant DefineProperty methods.

Pascal

```
TWriterProc = procedure (Writer: TWriter) of object;
```

C++

```
(Writer: TWriter) ( TWriterProc)();
```

File

Classes

Description

DefineProperty introduces an abstract method that allows descendants to read or write data like a property.

Each descendant filer object declares a DefineProperty method that defines the data that the descendant object will read or write as if the data were a property.

TWriterProc is the callback procedure parameter in that DefineProperty. When called, it supplies the TFiler (see page 2834) or descendant object, allowing write access.

See Also

DefineProperty (see page 2838)

DefineProperty (see page 2940)

DefineProperty (see page 3180)

TFiler (see page 2834)

TReader (see page 2933)

TWriter (see page 3175)

23.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
AddDataModule (see page 3309)	Defines a procedure to be called when a TDataModule (see page 2824) class is created.
ApplicationHandleException (see page 3310)	This is variable Classes.ApplicationHandleException.
ApplicationShowException (see page 3310)	This is variable Classes.ApplicationShowException.
CreateVCLComObjectProc (see page 3310)	This is variable Classes.CreateVCLComObjectProc.
CurrentGroup (see page 3311)	Keeps track of which objects are associated with which packages.
GlobalNameSpace (see page 3311)	This is variable Classes.GlobalNameSpace.
IsUniqueGlobalComponentNameProc (see page 3311)	IsUniqueGlobalComponentNameProc is called by IsUniqueGlobalComponentName (see page 3234).
RegisterComponentsProc (see page 3312)	RegisterComponentsProc is called by RegisterComponents (see page 3256).
RegisterNoIconProc (see page 3313)	RegisterNoIconProc is called by RegisterNoIcon (see page 3260).
RegisterNonActiveXProc (see page 3313)	RegisterNonActiveXProc is called by RegisterNonActiveX (see page 3260).
RemoveDataModule (see page 3314)	Defines a procedure to be called when a TDataModule (see page 2824) class is destroyed.
WakeMainThread (see page 3314)	Represents a method (event handler) that is forced into the main thread's queue.

23.6.1 Classes.AddDataModule Variable

Defines a procedure to be called when a TDataModule (see page 2824) class is created.

Pascal

```
AddDataModule: procedure (DataModule: TDataModule) of object = nil;
```

C++

```
procedure (DataModule: TDataModule) of object AddDataModule = nil;
```

File

Classes

Description

If the AddDataModule variable is set, both Create and CreateNew will invoke the procedure it defines, passing itself in the **Sender** parameter.

See Also

Create (see page 2827)

CreateNew (see page 2828)

23.6.2 Classes.ApplicationHandleException Variable

Pascal

```
ApplicationHandleException: procedure (Sender: TObject) of object = nil;
```

C++

```
procedure (Sender: TObject) of object ApplicationHandleException = nil;
```

File

Classes

Description

This is variable Classes.ApplicationHandleException.

23.6.3 Classes.ApplicationShowException Variable

Pascal

```
ApplicationShowException: procedure (E: Exception) of object = nil;
```

C++

```
procedure (E: Exception) of object ApplicationShowException = nil;
```

File

Classes

Description

This is variable Classes.ApplicationShowException.

23.6.4 Classes.CreateVCLComObjectProc Variable

Pascal

```
CreateVCLComObjectProc: procedure(Component: TComponent) = nil;
```

C++

```
procedure(Component: TComponent) CreateVCLComObjectProc = nil;
```

File

Classes

Description

This is variable Classes.CreateVCLComObjectProc.

23.6.5 Classes.CurrentGroup Variable

Keeps track of which objects are associated with which packages.

Pascal

```
CurrentGroup: Integer = -1;
```

C++

```
int CurrentGroup = -1;
```

File

Classes

Description

When a design-time package registers a property editor class, the registration list has to keep track of what package that property editor class is associated with. Tracking this association ensures that if the package needs to be unloaded at design time (to recompile the package), the registration list knows which property classes to delete from its list. CurrentGroup is used to obtain an indicator that represents this association. It gets incremented every time a package is loaded at design time.

CurrentGroup is used to write design-time tools that keep track of things stored in other packages.

See Also

RegisterPropertyEditor

23.6.6 Classes.GlobalNameSpace Variable

Pascal

```
GlobalNameSpace: IReadWriteSync;
```

C++

```
IReadWriteSync GlobalNameSpace;
```

File

Classes

Description

This is variable Classes.GlobalNameSpace.

23.6.7 Classes.IsUniqueGlobalComponentNameProc Variable

IsUniqueGlobalComponentNameProc is called by IsUniqueGlobalComponentName (see page 3234).

Pascal

```
IsUniqueGlobalComponentNameProc: TIsUniqueGlobalComponentName;
```

C++

```
TIsUniqueGlobalComponentName IsUniqueGlobalComponentNameProc;
```

File

Classes

Description

IsUniqueGlobalComponentName (see page 3234) is used to determine whether a proposed component name is already used on any component on any form, data module, or other top-level container component.

If IsUniqueGlobalComponentNameProc is defined, it is invoked by IsUniqueGlobalComponentName (see page 3234) in place of this checking process.

See Also

IsUniqueGlobalComponentName (see page 3234)

TIsUniqueGlobalComponentName (see page 3296)

23.6.8 Classes.RegisterComponentsProc Variable

RegisterComponentsProc is called by RegisterComponents (see page 3256).

Pascal

```
RegisterComponentsProc: procedure(const Page: string; const ComponentClasses: array of
TComponentClass) = nil;
```

C++

```
procedure(const Page: string; const ComponentClasses: array of TComponentClass)
RegisterComponentsProc = nil;
```

File

Classes

Description

RegisterComponents (see page 3256) is used to install a set of components in the IDE. When it is invoked, it expects RegisterComponentsProc to be defined, and will throw an exception if not. If defined, it is invoked to carry out the component registration.

The **Page** parameter defines the name of the page on the component palette where the components will appear.

ComponentClasses parameter defines an array of Components to be added to the component palette.

See Also

RegisterComponents (see page 3256)

RegisterNoIconProc (see page 3313)

RegisterNonActiveXProc (see page 3313)

23.6.9 Classes.RegisterNoIconProc Variable

RegisterNoIconProc is called by RegisterNoIcon (see page 3260).

Pascal

```
RegisterNoIconProc: procedure(const ComponentClasses: array of TComponentClass) = nil;
```

C++

```
procedure(const ComponentClasses: array of TComponentClass) RegisterNoIconProc = nil;
```

File

Classes

Description

RegisterNoIcon (see page 3260) is used to register a set of custom components so that objects from that class can use the Object Inspector. When it is invoked, it expects RegisterNoIconProc to be defined, and will throw an exception if not. If defined, it is invoked to carry out the registration.

ComponentClasses parameter defines an array of Components to be registered.

See Also

RegisterComponents (see page 3256)

RegisterComponentsProc (see page 3312)

RegisterNonActiveXProc (see page 3313)

23.6.10 Classes.RegisterNonActiveXProc Variable

RegisterNonActiveXProc is called by RegisterNonActiveX (see page 3260).

Pascal

```
RegisterNonActiveXProc: procedure(const ComponentClasses: array of TComponentClass; AxRegType:
TActiveXRegType) = nil;
```

C++

```
procedure(const ComponentClasses: array of TComponentClass; AxRegType: TActiveXRegType)
RegisterNonActiveXProc = nil;
```

File

Classes

Description

RegisterNonActiveX (see page 3260) is used to prevent a registered set of custom components from being converted into ActiveX controls using the ActiveX wizard. It expects RegisterNonActiveXProc to be defined, and will throw an exception if not. If defined, it is invoked to carry out the protective process.

ComponentClasses parameter an array of custom classes that are descended from TComponent (see page 2762).

The **AxRegType** parameter indicates whether only the components in the **ComponentClasses** array should be blocked from the ActiveX wizard (when AxRegType is axrComponentOnly), or whether their descendants should be prohibited from becoming

ActiveX controls as well (when AxRegType is axrIncludeDescendants).

See Also

RegisterComponents (see page 3256)

RegisterComponentsProc (see page 3312)

RegisterNoIconProc (see page 3313)

23.6.11 Classes.RemoveDataModule Variable

Defines a procedure to be called when a TDataModule (see page 2824) class is destroyed.

Pascal

```
RemoveDataModule: procedure (DataModule: TDataModule) of object = nil;
```

C++

```
procedure (DataModule: TDataModule) of object RemoveDataModule = nil;
```

File

Classes

Description

If the RemoveDataModule variable is set, Destroy will invoke the procedure it defines, passing itself in the **Sender** parameter.

See Also

Destroy (see page 2829)

23.6.12 Classes.WakeMainThread Variable

Represents a method (event handler) that is forced into the main thread's queue.

Pascal

```
WakeMainThread: TNotifyEvent = nil;
```

C++

```
TNotifyEvent WakeMainThread = nil;
```

File

Classes

Description

Assign a method to WakeMainThread before calling a thread's Synchronize method. When you call a thread's Synchronize method, it calls the method assigned to WakeMainThread once it has obtained a lock on the main GUI thread. This allows other threads to quickly synchronize with the GUI thread even if no events are being processed due to an idle state.

Warning: Warning:Only set WakeMainThread from the application's main thread. This variable should never be set by a secondary thread while another thread's Synchronize method is executing.

See Also

CheckSynchronize (see page 3219)

TThread (see page 3108)

TApplication

23.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
fmCreate (see page 3316)	Specifies a file to be created with a specific name.
MaxListSize (see page 3317)	Defines the maximum capacity of certain list classes and types.
scAlt (see page 3317)	Specifies that the Alt key was pressed in a short cut menu.
scCtrl (see page 3318)	Specifies that the Ctrl key was pressed in a short cut menu.
scNone (see page 3318)	Specifies that none of the Shift, Ctrl or Alt keys were pressed in a short cut menu.
scShift (see page 3319)	Specifies that the Shift was pressed in a short cut menu.
soFromBeginning (see page 3319)	Specifies that a stream seek operation should move relative to the stream beginning.
soFromCurrent (see page 3320)	Specifies that a stream seek operation should move relative to the current position.
soFromEnd (see page 3320)	Specifies that a stream seek operation should move relative to the stream end.
toEOF (see page 3321)	Specifies a TParser (see page 2915) End of file token.
toFloat (see page 3322)	Specifies a TParser (see page 2915) floating point number token.
toInteger (see page 3322)	Specifies a TParser (see page 2915) integer token.
toString (see page 3323)	Specifies a TParser (see page 2915) string token.
toSymbol (see page 3324)	Specifies a TParser (see page 2915) symbol token.
toWString (see page 3324)	Specifies a TParser (see page 2915) wide string token.

23.7.1 Classes.fmCreate Constant

Specifies a file to be created with a specific name.

Pascal

```
fmCreate = $FFFF;
```

C++

```
fmCreate = 0xFFFF;
```

File

Classes

Description

fmCreate is one of the values you pass to the Create method when specifying a file to be ready for read/write. The TFileStream (see page 2841) is used to read and write files. It specifies that a new file is to be created.

See Also

fmOpenRead (see page 23046)

fmOpenReadWrite (see page 23047)

fmOpenWrite (see page 23047)

faShareCompat

faShareDenyNone

faShareDenyRead

faShareDenyWrite

faShareExclusive

FileMode (see page 22644)

Create (see page 2843)

23.7.2 Classes.MaxListSize Constant

Defines the maximum capacity of certain list classes and types.

Pascal

```
MaxListSize = Maxint div 16;
```

C++

```
MaxListSize = Maxint div 16;
```

File

Classes

Description

MaxListSize degines the maximum capacity - the maximum number of elements - of list types such as TList (see page 2868). This integer value is a 1/16th of MaxInt - the maximum size of an Integer data type.

See Also

TList (see page 2868)

TPointerList (see page 3299)

TStringItemList (see page 3304)

23.7.3 Classes.scAlt Constant

Specifies that the Alt key was pressed in a short cut menu.

Pascal

```
scAlt = $8000;
```

C++

```
scAlt = 0x8000;
```

File

Classes

Description

scAlt is principally used in the TShortCut (see page 3303) value used by routines such as ShortCut. ShortCut is used to generate short cut menus at run time.

See Also

scCtrl (see page 3318)

scNone (see page 3318)

scShift (see page 3319)

ShortCut

TShortCut (see page 3303)

23.7.4 Classes.scCtrl Constant

Specifies that the Ctrl key was pressed in a short cut menu.

Pascal

```
scCtrl = $4000;
```

C++

```
scCtrl = 0x4000;
```

File

Classes

Description

scCtrl is principally used in the TShortCut (see page 3303) value used by routines such as ShortCut. ShortCut is used to generate short cut menus at run time.

See Also

scAlt (see page 3317)

scNone (see page 3318)

scShift (see page 3319)

ShortCut

TShortCut (see page 3303)

23.7.5 Classes.scNone Constant

Specifies that none of the Shift, Ctrl or Alt keys were pressed in a short cut menu.

Pascal

```
scNone = 0;
```

C++

```
scNone = 0;
```

File

Classes

Description

scNone is principally used in the TShortCut (see page 3303) value used by routines such as ShortCut. ShortCut is used to generate short cut menus at run time.

See Also

scAlt (see page 3317)

scCtrl (see page 3318)

scShift (see page 3319)

ShortCut

TShortCut (see page 3303)

23.7.6 Classes.scShift Constant

Specifies that the Shift was pressed in a short cut menu.

Pascal

```
scShift = $2000;
```

C++

```
scShift = 0x2000;
```

File

Classes

Description

scShift is principally used in the TShortCut (see page 3303) value used by routines such as ShortCut. ShortCut is used to generate short cut menus at run time.

See Also

scAlt (see page 3317)

scCtrl (see page 3318)

scNone (see page 3318)

ShortCut

TShortCut (see page 3303)

23.7.7 Classes.soFromBeginning Constant

Specifies that a stream seek operation should move relative to the stream beginning.

Pascal

```
soFromBeginning = 0;
```

C++

```
soFromBeginning = 0;
```

File

Classes

Description

soFromBeginning is one of the TSeekOrigin (see page 3276) values you use to pass to the TSream Seek method. It specifies that the seek proceed to an offset from the beginning of the stream.

See Also

Seek (see page 2988)

TStream (see page 2973)

soFromCurrent (see page 3320)

soFromEnd (see page 3320)

TSeekOrigin (see page 3276)

23.7.8 Classes.soFromCurrent Constant

Specifies that a stream seek operation should move relative to the current position.

Pascal

```
soFromCurrent = 1;
```

C++

```
soFromCurrent = 1;
```

File

Classes

Description

soFromCurrent is one of the TSeekOrigin (see page 3276) values you use to pass to the TSream Seek method. It specifies that the seek proceed to an offset from the current position in the stream.

See Also

Seek (see page 2988)

TStream (see page 2973)

soFromBeginning (see page 3319)

soFromEnd (see page 3320)

TSeekOrigin (see page 3276)

23.7.9 Classes.soFromEnd Constant

Specifies that a stream seek operation should move relative to the stream end.

Pascal

```
soFromEnd = 2;
```

C++

```
soFromEnd = 2;
```

File

Classes

Description

soFromEnd is one of the TSeekOrigin (see page 3276) values you use to pass to the TSream Seek method. It specifies that the seek proceed to an offset from the end of the stream.

See Also

Seek (see page 2988)

TStream (see page 2973)

soFromCurrent (see page 3320)

soFromBeginning (see page 3319)

TSeekOrigin (see page 3276)

23.7.10 Classes.toEOF Constant

Specifies a TParser (see page 2915) End of file token.

Pascal

```
toEOF = Char(0);
```

C++

```
toEOF = Char(0);
```

File

Classes

Description

toEOF is one of a number of Char values that are used by the TParser (see page 2915). These are tokens, representing the type of information that TParser (see page 2915) has found whilst parsing.

toEOF specifies that the end of the file (EOF) has occured.

Note: TParser (see page 2915) is principally used internally by Delphi to parse DFM (form) source files. It can be used as a more general parser, but only for files similar in format to DFM files.

See Also

toFloat (see page 3322)

toInteger (see page 3322)

toString (see page 3323)

toSymbol (see page 3324)

toWString (see page 3324)

TParser (see page 2915)

23.7.11 Classes.toFloat Constant

Specifies a TParser (see page 2915) floating point number token.

Pascal

```
toFloat = Char(4);
```

C++

```
toFloat = Char(4);
```

File

Classes

Description

toFloat is one of a number of Char values that are used by the TParser (see page 2915). These are tokens, representing the type of information that TParser (see page 2915) has found whilst parsing.

toFloat specifies that a floating point number value has been encountered. The value is available in the TokenFloat property. It is also available in its original string form in the TokenString property.

Note: TParser (see page 2915) is principally used internally by Delphi to parse DFM (form) source files. It can be used as a more general parser, but only for files similar in format to DFM files.

See Also

toEOF (see page 3321)

toInteger (see page 3322)

TokenFloat (see page 2923)

TokenString (see page 2924)

toString (see page 3323)

toSymbol (see page 3324)

toWString (see page 3324)

TParser (see page 2915)

23.7.12 Classes.toInteger Constant

Specifies a TParser (see page 2915) integer token.

Pascal

```
toInteger = Char(3);
```

C++

```
toInteger = Char(3);
```

File

Classes

Description

toInteger is one of a number of Char values that are used by the TParser (see page 2915). These are tokens, representing the type of information that TParser (see page 2915) has found whilst parsing.

toInteger specifies that an integer value has been encountered. The value is available in the TokenInteger property. It is also available in its original string form in the TokenString property.

Note: TParser (see page 2915) is principally used internally by Delphi to parse DFM (form) source files. It can be used as a more general parser, but only for files similar in format to DFM files.

See Also

toEOF (see page 3321)

toFloat (see page 3322)

TokenInteger

TokenString (see page 2924)

toString (see page 3323)

toSymbol (see page 3324)

toWString (see page 3324)

TParser (see page 2915)

23.7.13 Classes.toString Constant

Specifies a TParser (see page 2915) string token.

Pascal

```
toString = Char(2);
```

C++

```
toString = Char(2);
```

File

Classes

Description

toString is one of a number of Char values that are used by the TParser (see page 2915). These are tokens, representing the type of information that TParser (see page 2915) has found whilst parsing.

toString specifies that a string value has been encountered. The value is available in the TokenString property.

Note: TParser (see page 2915) is principally used internally by Delphi to parse DFM (form) source files. It can be used as a more general parser, but only for files similar in format to DFM files.

See Also

toEOF (see page 3321)

toFloat (see page 3322)

TokenString (see page 2924)

toInteger (see page 3322)

toSymbol (see page 3324)

toWString (see page 3324)

TParser (see page 2915)

23.7.14 Classes.toSymbol Constant

Specifies a TParser (see page 2915) symbol token.

Pascal

```
toSymbol = Char(1);
```

C++

```
toSymbol = Char(1);
```

File

Classes

Description

toSymbol is one of a number of Char values that are used by the TParser (see page 2915). These are tokens, representing the type of information that TParser (see page 2915) has found whilst parsing.

toSymbol specifies that a symbol value has been encountered. 'object', 'Left' and 'end' are typical DFM symbols. They are like Delphi keywords, as opposed to integer, string, widestring and float values that occur following a symbol and '=' sign combination.

The value is available in the TokenString property.

Note: TParser (see page 2915) is principally used internally by Delphi to parse DFM (form) source files. It can be used as a more general parser, but only for files similar in format to DFM files.

See Also

toEOF (see page 3321)

toFloat (see page 3322)

TokenString (see page 2924)

toInteger (see page 3322)

toString (see page 3323)

toWString (see page 3324)

TParser (see page 2915)

23.7.15 Classes.toWString Constant

Specifies a TParser (see page 2915) wide string token.

Pascal

```
toWString = Char(5);
```

C++

```
toWString = Char(5);
```

File

Classes

Description

toWString is one of a number of Char values that are used by the TParser (see page 2915). These are tokens, representing the type of information that TParser (see page 2915) has found whilst parsing.

toWString specifies that a wide string value has been encountered. The value is available in the TokenWideString property.

Note: TParser (see page 2915) is principally used internally by Delphi to parse DFM (form) source files. It can be used as a more general parser, but only for files similar in format to DFM files.

See Also

toEOF (see page 3321)

toFloat (see page 3322)

TokenString (see page 2924)

toInteger (see page 3322)

toString (see page 3323)

toSymbol (see page 3324)

TParser (see page 2915)

24 Clipbrd Namespace

This is namespace Clipbrd.

Topics

Name	Description
Classes (see page 3328)	The following table lists classes in this documentation.
Functions (see page 3351)	The following table lists functions in this documentation.
Variables (see page 3354)	The following table lists variables in this documentation.

Classes

	Name	Description
	TClipboard (see page 3328)	TClipboard represents the system clipboard.

Functions

	Name	Description
	Clipboard (see page 3351)	Returns an instance of TClipboard (see page 3328).
	SetClipboard (see page 3353)	Replaces the single global clipboard instance with another clipboard object.

Variables

Name	Description
CF_COMPONENT (see page 3354)	This is variable Clipbrd.CF_COMPONENT.
CF_PICTURE (see page 3354)	This is variable Clipbrd.CF_PICTURE.

24.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TClipboard (see page 3328)	TClipboard represents the system clipboard.

24.1.1 TClipboard Class

TClipboard represents the system clipboard.

Class Hierarchy

Pascal

```
TClipboard = class(TPersistent);
```

C++

```
class TClipboard : public TPersistent;
```

File

Clipbrd

Description

TClipboard is a wrapper for the system clipboard, which contains information that has been cut or copied from an application.

Do not instantiate separate TClipboard objects by calling the constructor. Instead, use the global instance of TClipboard returned by the Clipboard (see page 3351) function. This allows applications to share a single instance of TClipboard and safeguards against accidental deletion of the clipboard.

Before calling Clipboard (see page 3351), your application must include the Clipbrd (see page 3327) unit. That is, in Delphi, the uses clause must include Clipbrd (see page 3327) and in C++ you must have an include statement for Clipbrd.hpp.

TClipboard's properties provide information about the formats that the clipboard is registered to handle.

TClipboard's methods

store and retrieve data in appropriate formats.

open and close the clipboard, maintaining a reference count.

manage formats for objects in the clipboard.

See Also

Clipboard (see page 3351)

Adding the Clipboard Object

Cutting

Selecting Text

24.1.1.1 TClipboard Members

The following tables list the members exposed by TClipboard.

Protected Methods

	Name	Description
V	AssignTo (see page 3333)	Copies the properties of an object to a destination object.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.
	MainWndProc (see page 3342)	This is MainWndProc, a member of class TClipboard.
	SetBuffer (see page 3345)	This is SetBuffer, a member of class TClipboard.
V	WndProc (see page 3346)	This is WndProc, a member of class TClipboard.

Protected Properties

	Name	Description
R	Handle (see page 3349)	This is Handle, a member of class TClipboard.
R	OpenRefCount (see page 3349)	This is OpenRefCount, a member of class TClipboard.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 3331)	Copies an object to the clipboard.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 3333)	Deletes the contents of the clipboard.
V	Close (see page 3336)	Closes the clipboard if it is open.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 3330)	Destroys the TPersistent instance and frees its memory.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAsHandle (see page 3336)	Returns a Windows handle to data from the clipboard in the specified format.
	GetComponent (see page 3339)	Retrieves a component from the clipboard.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetTextBuf (see page 3340)	Retrieves text from the clipboard and copies it to a buffer.
	HasFormat (see page 3340)	Indicates whether the clipboard contains data in a specified format.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Open (see page 3342)	Opens the clipboard, preventing other applications from changing its contents until the clipboard is closed.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAsHandle (see page 3343)	Places data represented by a specified handle on the clipboard.
	SetComponent (see page 3345)	Copies a component to the clipboard.
	SetTextBuf (see page 3346)	Writes a text buffer to the Clipboard (see page 3351).

Public Properties

	Name	Description
	AsText (see page 3347)	Represents the contents of the clipboard as a string.
R	FormatCount (see page 3348)	Specifies the number of formats in the Formats (see page 3348) array property.
R	Formats (see page 3348)	Provides indexed access to all the formats the clipboard contains.

24.1.1.2 TClipboard.Destroy Destructor

Destroys the TPersistent instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TClipboard();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

Free (see page 22300)

Destroy (see page 22287)

24.1.1.3 TClipboard Methods

The methods of the TClipboard class are listed here.

Protected Methods

	Name	Description
V	AssignTo (see page 3333)	Copies the properties of an object to a destination object.
	MainWndProc (see page 3342)	This is MainWndProc, a member of class TClipboard.
	SetBuffer (see page 3345)	This is SetBuffer, a member of class TClipboard.
V	WndProc (see page 3346)	This is WndProc, a member of class TClipboard.

Public Methods

	Name	Description
V	Assign (see page 3331)	Copies an object to the clipboard.
V	Clear (see page 3333)	Deletes the contents of the clipboard.
V	Close (see page 3336)	Closes the clipboard if it is open.
	GetAsHandle (see page 3336)	Returns a Windows handle to data from the clipboard in the specified format.

	GetComponent (see page 3339)	Retrieves a component from the clipboard.
	GetTextBuf (see page 3340)	Retrieves text from the clipboard and copies it to a buffer.
	HasFormat (see page 3340)	Indicates whether the clipboard contains data in a specified format.
V	Open (see page 3342)	Opens the clipboard, preventing other applications from changing its contents until the clipboard is closed.
	SetAsHandle (see page 3343)	Places data represented by a specified handle on the clipboard.
	SetComponent (see page 3345)	Copies a component to the clipboard.
	SetTextBuf (see page 3346)	Writes a text buffer to the Clipboard (see page 3351).

24.1.1.3.1 TClipboard.Assign Method

Copies an object to the clipboard.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use Assign to copy pictures to and from the clipboard. For example, the following code copies the bitmap from a TBitmap instance (Bitmap1) to the clipboard.

```
Clipboard.Assign(Bitmap1); {Delphi}
Clipboard()->Assign(Bitmap1); // C++
```

If a bitmap is on the clipboard, the following code copies it to Bitmap1.

```
Bitmap1.Assign(Clipboard); {Delphi}
Bitmap1->Assign(Clipboard()); // C++
```

The Formats (see page 3348) property lists the formats used by the clipboard. Different kinds of graphic object (such as bitmaps and metafiles) have their own formats. Use the HasFormat (see page 3340) method to determine whether the information on the clipboard is stored in a format compatible with the object you want to assign it to.

C++ Examples:

```
/*
This example uses an image, a button, and a shape component
on a form. When the user clicks the button, an image of the
form is stored in the FormImage variable and copied to the
Clipboard. The image of the form is then copied back to
the image component, producing an interesting result,
especially if the button is clicked multiple times.  Add
ExtCtrls, StdCtrls, and Clipbrd to the uses clause.
}
*/

#include <Clipbrd.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *FormImage = GetFormImage();
  try
  {
    Clipboard()->Assign(FormImage);
    Image1->Picture->Assign(Clipboard());
  }
  __finally
  {
    delete FormImage;
  }
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Shape1->Shape = stEllipse;
  Shape1->Brush->Color = clLime;
  Image1->Stretch = true;
}
```

Delphi Examples:

```
{
The following line copies the bitmap of a speed button named
SpeedButton1 to the clipboard and then copies the clipboard
to the image picture.  Note that you must add Clipbrd to the
uses clause.
}
uses Clipbrd;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Clipboard.Assign(SpeedButton1.Glyph);
  Image1.Stretch := True;
  Image1.Picture.Assign(Clipboard);
end;
{
This example uses an image, a button, and a shape component
on a form. When the user clicks the button, an image of the
form is stored in the FormImage variable and copied to the
Clipboard. The image of the form is then copied back to
the image component, producing an interesting result,
especially if the button is clicked multiple times.  Add
ExtCtrls, StdCtrls, and Clipbrd to the uses clause.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FormImage: TBitmap;
begin
  FormImage := GetFormImage;
  try
    Clipboard.Assign(FormImage);
    Image1.Picture.Assign(Clipboard);
  finally
    FormImage.Free;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Shape1.Shape := stEllipse;
  Shape1.Brush.Color := clLime;
  Image1.Stretch := True;
end;
```

See Also

Formats (see page 3348)

HasFormat (see page 3340)

Assign (see page 14402)

Assign (see page 2930)

24.1.1.3.2 TClipboard.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign (see page 2930)

EConvertError (see page 22724)

24.1.1.3.3 TClipboard.Clear Method

Deletes the contents of the clipboard.

Pascal

```
procedure Clear; virtual;
```

C++

```
virtual __fastcall Clear();
```

Description

Call Clear to empty the clipboard.

Clear is called automatically by TClipboard (see page 3328) operations that add data to the clipboard.

C++ Examples:

```
/*
This example uses a button and an TEdit on a form. When the
user clicks the button, any text on the clipboard is pasted
into the TEdit.  If there is no text on the clipboard, a
message box appears.
```

```
Note:  To use this example, you must add Clipbrd to the uses
clause.  Load the Clipboard by performing a copy in any
application.
*/
#include <Clipbrd.hpp>
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (Clipboard()->HasFormat(CF_TEXT))
  {
    Edit1->Text = Clipboard()->AsText;
    Clipboard()->Clear();
  }
  else
    MessageDlg(
      "There is no text on the Clipboard",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}
/*
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
*/
#include <Clipbrd.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClipboard *cb = Clipboard();
  if (cb->HasFormat(CF_BITMAP))
  {
    Graphics::TBitmap *Bitmap = new Graphics::TBitmap();
    try
    {
      Bitmap->LoadFromClipboardFormat(CF_BITMAP, cb->GetAsHandle(CF_BITMAP), 0);
      Canvas->Draw(5,5,Bitmap);
    }
    catch (...)
    {
      MessageBeep(0);
      ShowMessage("Error reading image from clipboard!");
    }
    delete Bitmap;
    cb->Clear();
  }
  else
  {
    MessageBeep(0);
    ShowMessage("Cannot find image in clipboard!");
  }
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  unsigned int DataHandle;
  HPALETTE APalette;
  unsigned short MyFormat;
  Graphics::TBitmap *Bitmap = new Graphics::TBitmap();
  try
  {
    Bitmap->LoadFromFile("C:\\Program Files\\Common Files\\CodeGear
Shared\\Images\\Splash\\256Color\\FACTORY.BMP");
    // generate a clipboard format, with data and palette
    Bitmap->SaveToClipboardFormat(
      MyFormat,
      DataHandle,
      APalette);
```

```
    // save the data to the clipboard using that format and
    // the generated data
    Clipboard()->SetAsHandle(MyFormat,DataHandle);
  }
  catch (...)
  {
     ShowMessage("Failed to copy image to clipboard!");
  }
  delete Bitmap;
}
```

Delphi Examples:

```
{
This example uses a button and an TEdit on a form. When the
user clicks the button, any text on the clipboard is pasted
into the TEdit.  If there is no text on the clipboard, a
message box appears.
Note:  To use this example, you must add Clipbrd to the uses
clause.  Load the Clipboard by performing a copy in any
application.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
if Clipboard.HasFormat(CF_TEXT) then
  begin
  Edit1.Text := Clipboard.AsText;
  Clipboard.Clear;
  end
else
  MessageDlg(
    'There is no text on the Clipboard',
    mtInformation, [mbOK],0);
end;
{
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitmap;
  s : string;
  Icon: TIcon;
begin
 Bitmap := TBitMap.Create;
 try
   Image1.Picture.RegisterClipboardFormat(cf_BitMap,TIcon);
   Bitmap.LoadFromClipBoardFormat(
     cf_BitMap,ClipBoard.GetAsHandle(cf_Bitmap),0);
   Canvas.draw(0,0,Bitmap);
 finally
   Bitmap.free;
   Clipboard.Clear;
 end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  MyFormat : Word;
  Bitmap : TBitMap;
  AData : THandle;
  APalette : HPALETTE;
begin
  Bitmap := TBitmap.Create;
```

```
    try
      Bitmap.LoadFromFile('c:\Program Files\Common Files\CodeGear
  Shared\Images\Splash\256color\factory.bmp');
      Bitmap.SaveToClipBoardFormat(
        MyFormat,
        AData,
        APalette);
      ClipBoard.SetAsHandle(MyFormat,AData);
    finally
      Bitmap.Free;
    end;
  end;
```

24.1.1.3.4 TClipboard.Close Method

Closes the clipboard if it is open.

Pascal

```
procedure Close; virtual;
```

C++

```
virtual __fastcall Close();
```

Description

Call Close when you are finished adding items to the clipboard.

The clipboard can be opened with multiple calls to the Open (see page 3342) method before being closed. Open (see page 3342) and Close maintain a count of the number of times the clipboard has been opened and will not close it until Close has been called the same number of times.

Delphi Examples:

```
{
Open, Close, GetAsHandle example
The following code locks the memory for text on the Clipboard, then reads the text.
}

uses clipbrd;

procedure TForm1.Button1Click(Sender: TObject);
var
  MyHandle: THandle;
  TextPtr: PChar;
begin
  ClipBoard.Open;
try
  MyHandle := Clipboard.GetAsHandle(CF_TEXT);
  TextPtr := GlobalLock(MyHandle);
  ListBox1.Items.Add(StrPas(TextPtr));
  GlobalUnlock(MyHandle);
finally
  Clipboard.Close;
end;
end;
```

See Also

Open (see page 3342)

24.1.1.3.5 TClipboard.GetAsHandle Method

Returns a Windows handle to data from the clipboard in the specified format.

Pascal

```
function GetAsHandle(Format: Word): THandle;
```

C++

```
__fastcall THandle GetAsHandle(Word Format);
```

Description

GetAsHandle retrieves data from the clipboard using Windows handles.

The handle returned by GetAsHandle is not owned by the application and is valid only as long as the clipboard is open. If you need it for longer, copy the data to another handle.

The Format parameter can specify any of the formats listed under the HasFormat (see page 3340) method. Applications can also define and register their own formats. See the Windows API Help for more information.

C++ Examples:

```
/*
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
*/
#include <Clipbrd.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClipboard *cb = Clipboard();
  if (cb->HasFormat(CF_BITMAP))
  {
    Graphics::TBitmap *Bitmap = new Graphics::TBitmap();
    try
    {
      Bitmap->LoadFromClipboardFormat(CF_BITMAP, cb->GetAsHandle(CF_BITMAP), 0);
      Canvas->Draw(5,5,Bitmap);
    }
    catch (...)
    {
      MessageBeep(0);
      ShowMessage("Error reading image from clipboard!");
    }
    delete Bitmap;
    cb->Clear();
  }
  else
  {
    MessageBeep(0);
    ShowMessage("Cannot find image in clipboard!");
  }
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  unsigned int DataHandle;
  HPALETTE APalette;
  unsigned short MyFormat;
  Graphics::TBitmap *Bitmap = new Graphics::TBitmap();
  try
  {
    Bitmap->LoadFromFile("C:\\Program Files\\Common Files\\CodeGear
Shared\\Images\\Splash\\256Color\\FACTORY.BMP");
    // generate a clipboard format, with data and palette
    Bitmap->SaveToClipboardFormat(
      MyFormat,
```

```
        DataHandle,
        APalette);
      // save the data to the clipboard using that format and
      // the generated data
      Clipboard()->SetAsHandle(MyFormat,DataHandle);
    }
    catch (...)
    {
       ShowMessage("Failed to copy image to clipboard!");
    }
    delete Bitmap;
  }
```

Delphi Examples:

```
{
Open, Close, GetAsHandle example
The following code locks the memory for text on the Clipboard, then reads the text.
}

uses clipbrd;

procedure TForm1.Button1Click(Sender: TObject);
var
  MyHandle: THandle;
  TextPtr: PChar;
begin
  ClipBoard.Open;
try
  MyHandle := Clipboard.GetAsHandle(CF_TEXT);
  TextPtr := GlobalLock(MyHandle);
  ListBox1.Items.Add(StrPas(TextPtr));
  GlobalUnlock(MyHandle);
finally
  Clipboard.Close;
end;
end;
{
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitmap;
  s : string;
  Icon: TIcon;
begin
 Bitmap := TBitMap.Create;
 try
   Image1.Picture.RegisterClipboardFormat(cf_BitMap,TIcon);
   Bitmap.LoadFromClipBoardFormat(
     cf_BitMap,ClipBoard.GetAsHandle(cf_Bitmap),0);
   Canvas.draw(0,0,Bitmap);
 finally
   Bitmap.free;
   Clipboard.Clear;
 end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  MyFormat : Word;
  Bitmap : TBitMap;
  AData : THandle;
```

```
  APalette : HPALETTE;
begin
  Bitmap := TBitmap.Create;
  try
    Bitmap.LoadFromFile('c:\Program Files\Common Files\CodeGear
Shared\Images\Splash\256color\factory.bmp');
    Bitmap.SaveToClipBoardFormat(
      MyFormat,
      AData,
      APalette);
    ClipBoard.SetAsHandle(MyFormat,AData);
  finally
    Bitmap.Free;
  end;
end;
```

See Also

SetAsHandle (see page 3343)

HasFormat (see page 3340)

24.1.1.3.6 TClipboard.GetComponent Method

Retrieves a component from the clipboard.

Pascal

```
function GetComponent(Owner: TComponent; Parent: TComponent): TComponent;
```

C++

```
__fastcall TComponent * GetComponent(TComponent * Owner, TComponent * Parent);
```

Description

Use GetComponent to retrieve a component that is stored on the clipboard.

Owner becomes the owner of the retrieved component and Parent becomes its parent. Owner is usually a form, but both parameters can be nil (Delphi) or NULL (C++).

Before a class is read from the clipboard, it must be registered by a call to RegisterClasses. If you try to read an unregistered class, you'll get an EClassNotFound exception.

Delphi Examples:

```
{
This example uses a button and a group box on a form. When
the user clicks the button, the button is copied to the
Clipboard and then retrieved from the Clipboard and placed
in the new parent of the button, the group box. The name of
the original button is changed to avoid having two components
with the same name at the same time.
}
implementation

uses Clipbrd;

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
  groupButton : TButton;
begin
  { copy the button to the clipboard }
  Clipboard.SetComponent(Button1);
  { rename the button which is still on the form }
```

```
    Button1.Name := 'OriginalButton';
    { Now retrieve the button from the clipboard }
    { and add it to GroupBox1 }
    { Note that the clipboard button is named Button1, while }
    { the source button has been renamed to 'OriginalButton' }
    groupButton := TButton(Clipboard.GetComponent(Self, GroupBox1));
    groupButton.top := Button1.top + 30;
  end;

  initialization
    RegisterClasses([TButton]);{ registers the TButton class }
```

See Also

EClassNotFound (see page 2694)

RegisterClasses

SetComponent (see page 3345)

24.1.1.3.7 TClipboard.GetTextBuf Method

Retrieves text from the clipboard and copies it to a buffer.

Pascal

```
function GetTextBuf(Buffer: PChar; BufSize: Integer): Integer;
```

C++

```
__fastcall int GetTextBuf(const char * Buffer, int BufSize);
```

Description

Use GetTextBuf to retrieve text from the clipboard and copy it into the buffer pointed to by Buffer. GetTextBuf copies up to the number of characters given by BufSize and returns the number of characters copied. The resulting text in Buffer is a null-terminated string.

See Also

SetTextBuf (see page 3346)

GetAsHandle (see page 3336)

24.1.1.3.8 TClipboard.HasFormat Method

Indicates whether the clipboard contains data in a specified format.

Pascal

```
function HasFormat(Format: Word): Boolean;
```

C++

```
__fastcall Boolean HasFormat(Word Format);
```

Description

Use HasFormat to find out whether the clipboard contains data encoded in a specific format. HasFormat returns true if the format is present and false otherwise. TClipboard (see page 3328) keeps a list of available formats in the Formats (see page 3348) array property.

Some of the possible values of the Format parameter follow. HasFormat supports additional formats provided by Windows or by other applications as well as custom formats registered with TPicture::.RegisterClipboardFormat.

Value	Meaning
CF_TEXT	Text with a CR-LF combination at the end of each line. A null character identifies the end of the text.
CF_BITMAP	A Windows bitmap graphic.
CF_METAFILEPICT	A Windows metafile graphic.
CF_PICTURE (see page 3354)	An object of type TPicture.
CF_COMPONENT (see page 3354)	Any persistent object.

C++ Examples:

```
/*
This example uses a button and an TEdit on a form. When the
user clicks the button, any text on the clipboard is pasted
into the TEdit.  If there is no text on the clipboard, a
message box appears.
Note:  To use this example, you must add Clipbrd to the uses
clause.  Load the Clipboard by performing a copy in any
application.
*/
#include <Clipbrd.hpp>
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (Clipboard()->HasFormat(CF_TEXT))
  {
    Edit1->Text = Clipboard()->AsText;
    Clipboard()->Clear();
  }
  else
    MessageDlg(
      "There is no text on the Clipboard",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This example uses a button and an TEdit on a form. When the
user clicks the button, any text on the clipboard is pasted
into the TEdit.  If there is no text on the clipboard, a
message box appears.
Note:  To use this example, you must add Clipbrd to the uses
clause.  Load the Clipboard by performing a copy in any
application.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
if Clipboard.HasFormat(CF_TEXT) then
  begin
  Edit1.Text := Clipboard.AsText;
  Clipboard.Clear;
  end
else
  MessageDlg(
    'There is no text on the Clipboard',
    mtInformation, [mbOK],0);
end;
```

See Also

Formats (see page 3348)

RegisterClipboardFormat (see page 14592)

Pasting Graphics from the Clipboard

24.1.1.3.9 TClipboard.MainWndProc Method

Pascal

```
procedure MainWndProc(var Message: TMessage);
```

C++

```
__fastcall MainWndProc(TMessage Message);
```

Description

This is MainWndProc, a member of class TClipboard.

24.1.1.3.10 TClipboard.Open Method

Opens the clipboard, preventing other applications from changing its contents until the clipboard is closed.

Pascal

```
procedure Open; virtual;
```

C++

```
virtual __fastcall Open();
```

Description

Call Open before adding a series of items to the clipboard. This prevents other applications from overwriting the clipboard until it is closed. (When adding a single item to the clipboard, there is no need to call Open.)

When an application has finished adding items to the clipboard, it should call the Close (see page 3336) method. The clipboard can be opened with multiple calls to the Open method before being closed. Open and Close (see page 3336) maintain a count of the number of times the clipboard has been opened and will not close it until Close (see page 3336) has been called the same number of times.

Delphi Examples:

```
{
Open, Close, GetAsHandle example
The following code locks the memory for text on the Clipboard, then reads the text.
}

uses clipbrd;

procedure TForm1.Button1Click(Sender: TObject);
var
  MyHandle: THandle;
  TextPtr: PChar;
begin
  ClipBoard.Open;
try
  MyHandle := Clipboard.GetAsHandle(CF_TEXT);
  TextPtr := GlobalLock(MyHandle);
  ListBox1.Items.Add(StrPas(TextPtr));
  GlobalUnlock(MyHandle);
finally
  Clipboard.Close;
end;
end;
```

See Also

Close (see page 3336)

24.1.1.3.11 TClipboard.SetAsHandle Method

Places data represented by a specified handle on the clipboard.

Pascal

```
procedure SetAsHandle(Format: Word; Value: THandle);
```

C++

```
__fastcall SetAsHandle(Word Format, THandle Value);
```

Description

SetAsHandle writes data represented by the specified handle (Value) to the clipboard using the specified format.

Do not delete a handle after passing it to SetAsHandle. The handle belongs to the clipboard, which will free it.

The Format parameter can specify any of the formats listed under the HasFormat (see page 3340) method. Applications can also define and register their own formats. See the Windows API Help for more information.

C++ Examples:

```
/*
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
*/
#include <Clipbrd.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClipboard *cb = Clipboard();
  if (cb->HasFormat(CF_BITMAP))
  {
    Graphics::TBitmap *Bitmap = new Graphics::TBitmap();
    try
    {
      Bitmap->LoadFromClipboardFormat(CF_BITMAP, cb->GetAsHandle(CF_BITMAP), 0);
      Canvas->Draw(5,5,Bitmap);
    }
    catch (...)
    {
      MessageBeep(0);
      ShowMessage("Error reading image from clipboard!");
    }
    delete Bitmap;
    cb->Clear();
  }
  else
  {
    MessageBeep(0);
    ShowMessage("Cannot find image in clipboard!");
  }
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  unsigned int DataHandle;
  HPALETTE APalette;
  unsigned short MyFormat;
```

```
  Graphics::TBitmap *Bitmap = new Graphics::TBitmap();
  try
  {
    Bitmap->LoadFromFile("C:\\Program Files\\Common Files\\CodeGear
Shared\\Images\\Splash\\256Color\\FACTORY.BMP");
    // generate a clipboard format, with data and palette
    Bitmap->SaveToClipboardFormat(
      MyFormat,
      DataHandle,
      APalette);
    // save the data to the clipboard using that format and
    // the generated data
    Clipboard()->SetAsHandle(MyFormat,DataHandle);
  }
  catch (...)
  {
     ShowMessage("Failed to copy image to clipboard!");
  }
  delete Bitmap;
}
```

Delphi Examples:

```
{
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitmap;
  s : string;
  Icon: TIcon;
begin
 Bitmap := TBitMap.Create;
 try
   Image1.Picture.RegisterClipboardFormat(cf_BitMap,TIcon);
   Bitmap.LoadFromClipBoardFormat(
     cf_BitMap,ClipBoard.GetAsHandle(cf_Bitmap),0);
   Canvas.draw(0,0,Bitmap);
 finally
   Bitmap.free;
   Clipboard.Clear;
 end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  MyFormat : Word;
  Bitmap : TBitMap;
  AData : THandle;
  APalette : HPALETTE;
begin
  Bitmap := TBitmap.Create;
  try
    Bitmap.LoadFromFile('c:\Program Files\Common Files\CodeGear
Shared\Images\Splash\256color\factory.bmp');
    Bitmap.SaveToClipBoardFormat(
      MyFormat,
      AData,
      APalette);
    ClipBoard.SetAsHandle(MyFormat,AData);
  finally
    Bitmap.Free;
  end;
```

```
end;
```

See Also

GetAsHandle (see page 3336)

HasFormat (see page 3340)

24.1.1.3.12 TClipboard.SetBuffer Method

Pascal

```
procedure SetBuffer(Format: Word; var Buffer; Size: Integer);
```

C++

```
__fastcall SetBuffer(Word Format,  Buffer, int Size);
```

Description

This is SetBuffer, a member of class TClipboard.

24.1.1.3.13 TClipboard.SetComponent Method

Copies a component to the clipboard.

Pascal

```
procedure SetComponent(Component: TComponent);
```

C++

```
__fastcall SetComponent(TComponent * Component);
```

Description

Use SetComponent to copy a component to the clipboard.

Delphi Examples:

```
{
This example uses a button and a group box on a form. When
the user clicks the button, the button is copied to the
Clipboard and then retrieved from the Clipboard and placed
in the new parent of the button, the group box. The name of
the original button is changed to avoid having two components
with the same name at the same time.
}
implementation

uses Clipbrd;

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
  groupButton : TButton;
begin
  { copy the button to the clipboard }
  Clipboard.SetComponent(Button1);
  { rename the button which is still on the form }
  Button1.Name := 'OriginalButton';
  { Now retrieve the button from the clipboard }
  { and add it to GroupBox1 }
  { Note that the clipboard button is named Button1, while }
  { the source button has been renamed to 'OriginalButton' }
  groupButton := TButton(Clipboard.GetComponent(Self, GroupBox1));
```

```
  groupButton.top := Button1.top + 30;
end;

initialization
  RegisterClasses([TButton]);{ registers the TButton class }
```

See Also

GetComponent (see page 3339)

24.1.1.3.14 TClipboard.SetTextBuf Method

Writes a text buffer to the Clipboard (see page 3351).

Pascal

```
procedure SetTextBuf(Buffer: PChar);
```

C++

```
__fastcall SetTextBuf(const char * Buffer);
```

Description

Use SetTextBuf to copy text to the clipboard from a buffer. Buffer must point to a null-terminated string.

See Also

GetTextBuf (see page 3340)

24.1.1.3.15 TClipboard.WndProc Method

Pascal

```
procedure WndProc(var Message: TMessage); virtual;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

This is WndProc, a member of class TClipboard.

24.1.1.4 TClipboard Properties

The properties of the TClipboard class are listed here.

Protected Properties

	Name	Description
R	Handle (see page 3349)	This is Handle, a member of class TClipboard.
R	OpenRefCount (see page 3349)	This is OpenRefCount, a member of class TClipboard.

Public Properties

	Name	Description
	AsText (see page 3347)	Represents the contents of the clipboard as a string.
R	FormatCount (see page 3348)	Specifies the number of formats in the Formats (see page 3348) array property.
R	Formats (see page 3348)	Provides indexed access to all the formats the clipboard contains.

24.1.1.4.1 TClipboard.AsText Property

Represents the contents of the clipboard as a string.

Pascal

```
property AsText: string;
```

C++

```
__property AnsiString AsText;
```

Description

Use the AsText property to place text in and retrieve text from the clipboard. The clipboard must contain a string or an exception occurs. To determine if the clipboard contains a string, pass CF_TEXT to the HasFormat (see page 3340) method.

C++ Examples:

```
/*
This example uses a button and an TEdit on a form. When the
user clicks the button, any text on the clipboard is pasted
into the TEdit.  If there is no text on the clipboard, a
message box appears.
Note:  To use this example, you must add Clipbrd to the uses
clause.  Load the Clipboard by performing a copy in any
application.
*/
#include <Clipbrd.hpp>
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (Clipboard()->HasFormat(CF_TEXT))
  {
    Edit1->Text = Clipboard()->AsText;
    Clipboard()->Clear();
  }
  else
    MessageDlg(
      "There is no text on the Clipboard",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This example uses a button and an TEdit on a form. When the
user clicks the button, any text on the clipboard is pasted
into the TEdit.  If there is no text on the clipboard, a
message box appears.
Note:  To use this example, you must add Clipbrd to the uses
clause.  Load the Clipboard by performing a copy in any
application.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
if Clipboard.HasFormat(CF_TEXT) then
  begin
  Edit1.Text := Clipboard.AsText;
  Clipboard.Clear;
  end
else
  MessageDlg(
    'There is no text on the Clipboard',
    mtInformation, [mbOK],0);
end;
```

See Also

GetTextBuf (see page 3340)

HasFormat (see page 3340)

SetTextBuf (see page 3346)

24.1.1.4.2 TClipboard.FormatCount Property

Specifies the number of formats in the Formats (see page 3348) array property.

Pascal

```
property FormatCount: Integer;
```

C++

```
__property int FormatCount;
```

Description

Use FormatCount to find the number of different format types used to encode the information currently stored in the clipboard. FormatCount is read-only.

To find out if a particular format is available on the clipboard, use the HasFormat (see page 3340) method.

Delphi Examples:

```
{
FormatCount, Formats example
The following code adds each format on the Clipboard to ListBox1 when Button1 is clicked.
Note: You will need to add Clipbrd to the uses clause (Delphi) or include <Clipbrd.hpp> (C++).
}

uses clipbrd;

procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to Clipboard.FormatCount - 1 do
    ListBox1.Items.Add(IntToStr(Clipboard.Formats[I]));
end;
```

See Also

Formats (see page 3348)

HasFormat (see page 3340)

RegisterClipboardFormat (see page 14592)

24.1.1.4.3 TClipboard.Formats Property

Provides indexed access to all the formats the clipboard contains.

Pascal

```
property Formats [Index: Integer]: Word;
```

C++

```
__property Word Formats[int Index];
```

Description

Read Formats to determine what formats encode the information currently on the clipboard. Each format can be accessed by its position in the array.

Usually, when an application copies or cuts something to the clipboard, it places it there in multiple formats. An application can place items of a particular format on the clipboard and retrieve items with a particular format from the clipboard if the format is in the Formats array. To find out if a particular format is available on the clipboard, use the HasFormat (see page 3340) method.

When reading information from the clipboard, use the Formats array to choose the best possible encoding for information that can be encoded in several ways.

Before you can write information to the clipboard in a particular format, the format must be registered. To register a new format, use the RegisterClipboardFormat method of TPicture.

Delphi Examples:

```
{
FormatCount, Formats example
The following code adds each format on the Clipboard to ListBox1 when Button1 is clicked.
Note: You will need to add Clipbrd to the uses clause (Delphi) or include <Clipbrd.hpp> (C++).
}

uses clipbrd;

procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to Clipboard.FormatCount - 1 do
    ListBox1.Items.Add(IntToStr(Clipboard.Formats[I]));
end;
```

See Also

HasFormat (see page 3340)

FormatCount (see page 3348)

RegisterClipboardFormat (see page 14592)

24.1.1.4.4 TClipboard.Handle Property

Pascal

```
property Handle: HWND;
```

C++

```
__property HWND Handle;
```

Description

This is Handle, a member of class TClipboard.

24.1.1.4.5 TClipboard.OpenRefCount Property

Pascal

```
property OpenRefCount: Integer;
```

C++

```
__property int OpenRefCount;
```

Description

This is OpenRefCount, a member of class TClipboard.

24.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	Clipboard (see page 3351)	Returns an instance of TClipboard (see page 3328).
	SetClipboard (see page 3353)	Replaces the single global clipboard instance with another clipboard object.

24.2.1 Clipbrd.Clipboard Function

Returns an instance of TClipboard (see page 3328).

Pascal

```
function Clipboard: TClipboard;
```

C++

```
TClipboard Clipboard();
```

File

Clipbrd

Description

Applications use a single instance of TClipboard (see page 3328) for interacting with the system clipboard. If the application has never used the clipboard, calling Clipboard creates a new instance of TClipboard (see page 3328). If the clipboard has already been used, Clipboard returns the previously created TClipboard (see page 3328) object.

To use Clipboard and the TClipboard (see page 3328) object, add the appropriate unit to the uses clause (Delphi) or include it in the source module (C++).

C++ Examples:

```
/*
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
*/
#include <Clipbrd.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClipboard *cb = Clipboard();
  if (cb->HasFormat(CF_BITMAP))
  {
    Graphics::TBitmap *Bitmap = new Graphics::TBitmap();
    try
    {
      Bitmap->LoadFromClipboardFormat(CF_BITMAP, cb->GetAsHandle(CF_BITMAP), 0);
      Canvas->Draw(5,5,Bitmap);
    }
    catch (...)
    {
```

```
      MessageBeep(0);
      ShowMessage("Error reading image from clipboard!");
    }
    delete Bitmap;
    cb->Clear();
  }
  else
  {
    MessageBeep(0);
    ShowMessage("Cannot find image in clipboard!");
  }
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  unsigned int DataHandle;
  HPALETTE APalette;
  unsigned short MyFormat;
  Graphics::TBitmap *Bitmap = new Graphics::TBitmap();
  try
  {
    Bitmap->LoadFromFile("C:\\Program Files\\Common Files\\CodeGear
Shared\\Images\\Splash\\256Color\\FACTORY.BMP");
    // generate a clipboard format, with data and palette
    Bitmap->SaveToClipboardFormat(
      MyFormat,
      DataHandle,
      APalette);
    // save the data to the clipboard using that format and
    // the generated data
    Clipboard()->SetAsHandle(MyFormat,DataHandle);
  }
  catch (...)
  {
     ShowMessage("Failed to copy image to clipboard!");
  }
  delete Bitmap;
}
```

Delphi Examples:

```
{
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitmap;
  s : string;
  Icon: TIcon;
begin
 Bitmap := TBitMap.Create;
 try
   Image1.Picture.RegisterClipboardFormat(cf_BitMap,TIcon);
   Bitmap.LoadFromClipBoardFormat(
     cf_BitMap,ClipBoard.GetAsHandle(cf_Bitmap),0);
   Canvas.draw(0,0,Bitmap);
 finally
   Bitmap.free;
   Clipboard.Clear;
 end;
end;

procedure TForm1.Button2Click(Sender: TObject);
```

```
var
  MyFormat : Word;
  Bitmap : TBitMap;
  AData : THandle;
  APalette : HPALETTE;
begin
  Bitmap := TBitmap.Create;
  try
    Bitmap.LoadFromFile('c:\Program Files\Common Files\CodeGear
Shared\Images\Splash\256color\factory.bmp');
    Bitmap.SaveToClipBoardFormat(
      MyFormat,
      AData,
      APalette);
    ClipBoard.SetAsHandle(MyFormat,AData);
  finally
    Bitmap.Free;
  end;
end;
```

See Also

TClipboard (see page 3328)

24.2.2 Clipbrd.SetClipboard Function

Replaces the single global clipboard instance with another clipboard object.

Pascal

```
function SetClipboard(NewClipboard: TClipboard): TClipboard;
```

C++

```
TClipboard SetClipboard(TClipboard NewClipboard);
```

File

Clipbrd

Description

Use SetClipboard to replace the global clipboard instance with a customized descendant of TClipboard (see page 3328). SetClipboard returns the currently instantiated clipboard, if any. After replacing the global clipboard instance using SetClipboard, an application must free the old clipboard that is returned.

See Also

Clipboard (see page 3351)

24.3 Variables

The following table lists variables in this documentation.

Variables

Name	Description
CF_COMPONENT (see page 3354)	This is variable Clipbrd.CF_COMPONENT.
CF_PICTURE (see page 3354)	This is variable Clipbrd.CF_PICTURE.

24.3.1 Clipbrd.CF_COMPONENT Variable

Pascal

```
CF_COMPONENT: Word;
```

C++

```
Word CF_COMPONENT;
```

File

Clipbrd

Description

This is variable Clipbrd.CF_COMPONENT.

24.3.2 Clipbrd.CF_PICTURE Variable

Pascal

```
CF_PICTURE: Word;
```

C++

```
Word CF_PICTURE;
```

File

Clipbrd

Description

This is variable Clipbrd.CF_PICTURE.

25 CmAdmCtl Namespace

This is namespace CmAdmCtl.

Topics

Name	Description
Classes (see page 3356)	The following table lists classes in this documentation.

Classes

	Name	Description
	CoCOMAdminCatalog (see page 3356)	This is class CmAdmCtl.CoCOMAdminCatalog.
	CoCOMAdminCatalogCollection (see page 3357)	This is class CmAdmCtl.CoCOMAdminCatalogCollection.
	CoCOMAdminCatalogObject (see page 3358)	This is class CmAdmCtl.CoCOMAdminCatalogObject.
	TCOMAdminCatalog (see page 3359)	TCOMAdminCatalog allows an application to act as an Automation controller for the COM+ administration tool.
	TCOMAdminCatalogCollection (see page 3380)	TCOMAdminCatalogCollection represents a collection from the COM+ catalog.
	TCOMAdminCatalogObject (see page 3396)	TCOMAdminCatalogObject represents an item in a collection from the COM+ catalog.

25.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	CoCOMAdminCatalog (see page 3356)	This is class CmAdmCtl.CoCOMAdminCatalog.
	CoCOMAdminCatalogCollection (see page 3357)	This is class CmAdmCtl.CoCOMAdminCatalogCollection.
	CoCOMAdminCatalogObject (see page 3358)	This is class CmAdmCtl.CoCOMAdminCatalogObject.
	TCOMAdminCatalog (see page 3359)	TCOMAdminCatalog allows an application to act as an Automation controller for the COM+ administration tool.
	TCOMAdminCatalogCollection (see page 3380)	TCOMAdminCatalogCollection represents a collection from the COM+ catalog.
	TCOMAdminCatalogObject (see page 3396)	TCOMAdminCatalogObject represents an item in a collection from the COM+ catalog.

25.1.1 CoCOMAdminCatalog Class

Class Hierarchy

CmAdmCtl.CoCOMAdminCatalog

Pascal

```
CoCOMAdminCatalog = class;
```

C++

```
class CoCOMAdminCatalog;
```

File

CmAdmCtl

Description

This is class CmAdmCtl.CoCOMAdminCatalog.

25.1.1.1 CoCOMAdminCatalog Members

The following tables list the members exposed by CoCOMAdminCatalog.

Public Methods

	Name	Description
	Create (see page 3357)	This is Create, a member of class CoCOMAdminCatalog.

25.1.1.2 CoCOMAdminCatalog Methods

The methods of the CoCOMAdminCatalog class are listed here.

Public Methods

	Name	Description
	Create (see page 3357)	This is Create, a member of class CoCOMAdminCatalog.

25.1.1.2.1 CoCOMAdminCatalog.Create Method

Pascal

```
class function Create: ICOMAdminCatalog;
```

C++

```
__fastcall ICOMAdminCatalog Create();
```

Description

This is Create, a member of class CoCOMAdminCatalog.

25.1.2 CoCOMAdminCatalogCollection Class

Class Hierarchy

CmAdmCtl.CoCOMAdminCatalogCollection

Pascal

```
CoCOMAdminCatalogCollection = class;
```

C++

```
class CoCOMAdminCatalogCollection;
```

File

CmAdmCtl

Description

This is class CmAdmCtl.CoCOMAdminCatalogCollection.

25.1.2.1 CoCOMAdminCatalogCollection Members

The following tables list the members exposed by CoCOMAdminCatalogCollection.

Public Methods

	Name	Description
	Create (see page 3358)	This is Create, a member of class CoCOMAdminCatalogCollection.

25.1.2.2 CoCOMAdminCatalogCollection Methods

The methods of the CoCOMAdminCatalogCollection class are listed here.

Public Methods

	Name	Description
	Create (see page 3358)	This is Create, a member of class CoCOMAdminCatalogCollection.

25.1.2.2.1 CoCOMAdminCatalogCollection.Create Method

Pascal

```
class function Create: ICatalogCollection;
```

C++

```
__fastcall ICatalogCollection Create();
```

Description

This is Create, a member of class CoCOMAdminCatalogCollection.

25.1.3 CoCOMAdminCatalogObject Class

Class Hierarchy

CmAdmCtl.CoCOMAdminCatalogObject

Pascal

```
CoCOMAdminCatalogObject = class;
```

C++

```
class CoCOMAdminCatalogObject;
```

File

CmAdmCtl

Description

This is class CmAdmCtl.CoCOMAdminCatalogObject.

25.1.3.1 CoCOMAdminCatalogObject Members

The following tables list the members exposed by CoCOMAdminCatalogObject.

Public Methods

	Name	Description
	Create (see page 3359)	This is Create, a member of class CoCOMAdminCatalogObject.

25.1.3.2 CoCOMAdminCatalogObject Methods

The methods of the CoCOMAdminCatalogObject class are listed here.

Public Methods

	Name	Description
	Create (see page 3359)	This is Create, a member of class CoCOMAdminCatalogObject.

25.1.3.2.1 CoCOMAdminCatalogObject.Create Method

Pascal

```
class function Create: ICatalogObject;
```

C++

```
__fastcall ICatalogObject Create();
```

Description

This is Create, a member of class CoCOMAdminCatalogObject.

25.1.4 TCOMAdminCatalog Class

TCOMAdminCatalog allows an application to act as an Automation controller for the COM+ administration tool.

Class Hierarchy

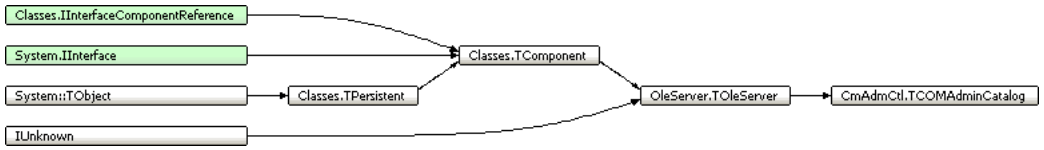

Pascal

```
TCOMAdminCatalog = class(TOleServer);
```

C++

```
class TCOMAdminCatalog : public TOleServer;
```

File

CmAdmCtl

Description

Use TCOMAdminCatalog to automate the administration of COM+ applications and services. TCOMAdminCatalog lets you read and write information that is stored on the COM+ catalog, which holds all COM+ configuration data. TCOMAdminCatalog implements the ICOMAdminCatalog interface, to let you

Access collections on the catalog.

Install COM+ applications and components.

Start and stop services.

Connect (see page 3365) to and administer remote servers.

Use the Connect (see page 3365) method to access the COMAdminCatalog server. You can then use the methods of TCOMAdminCatalog to send commands that administer COM+ applications on this or another machine.

See Also

TComAdminCatalogObject

TCOMAdminCatalogCollection (see page 3380)

25.1.4.1 TCOMAdminCatalog Members

The following tables list the members exposed by TCOMAdminCatalog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_AddRef (see page 18104)	Increments the reference count for the number of interface references dependent on the OLE server.
	_Release (see page 2769)	Called when an application releases a component interface.
	_Release (see page 18104)	Decrements the reference count for the calling interface on a object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ConnectEvents (see page 18105)	Establishes a connection using the EventDispatch (see page 18111) property as an event sink.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DisconnectEvents (see page 18106)	Terminates a connection that was established using the ConnectEvents (see page 18105) method.
	GetAutoConnect (see page 18106)	Returns the value of the AutoConnect (see page 18110) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetConnectKind (see page 18106)	Returns the value of the ConnectKind (see page 18110) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetServer (see page 18107)	Returns an interface for the COM server.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InitServerData (see page 3372)	Initializes the ServerData property to reflect the type library information of the COM server.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	InvokeEvent (see page 18107)	Dispatches an event from the COM server to the appropriate event handler.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 18108)	Connects to the COM server when the Loaded object is first loaded into memory if AutoConnect (see page 18110) is True true.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	QueryInterface (see page 18108)	Returns a reference to a specified interface if the OLE server supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoConnect (see page 18109)	Sets the value of the AutoConnect (see page 18110) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

	SetConnectKind (see page 18109)	Sets the value of the ConnectKind (see page 18110) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	EventDispatch (see page 18111)	This is EventDispatch, a member of class TOleServer.
	ServerData (see page 18111)	Encodes information about the COM server to which ServerData connects.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
	BackupREGDB (see page 3365)	Backs up the COM+ class registration database to a specified file.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Connect (see page 3365)	Establishes a connection to the COM server that implements the COM+ administration catalog.
	ConnectTo (see page 3366)	Connects the TCOMAdminCatalog (see page 3359) component to an existing ICOMAdminCatalog interface.
V	Create (see page 3363)	Creates a new instance of TCOMAdminCatalog (see page 3359).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 3364)	Frees an instance of TCOMAdminCatalog (see page 3359)
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Disconnect (see page 3366)	Terminates the connection to the COM server.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExportApplication (see page 3367)	Exports a COM+ application or application proxy to a file for installation on other machines.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	Get_MajorVersion (see page 3367)	Returns the value of the Get_MajorVersion property.
	Get_MinorVersion (see page 3368)	Returns the value of the Get_MinorVersion property.
	GetCollection (see page 3368)	Returns a TCOMAdminCatalogCollection (see page 3380) object for a specified collection.
	GetCollectionByQuery (see page 3368)	Retrieves a collection from the COM+ catalog given all of its parent items.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetEventClassesForIID (see page 3369)	Retrieves a list of event classes that implement a specified interface.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMultipleComponentsInfo (see page 3370)	Returns information about COM+ components found in specified .dll files.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	ICOMAdminCatalog_Connect (see page 3371)	ICOMAdminCatalog_Connects to a remote computer so that you can administer the collections on that machine.
	ImportComponent (see page 3371)	Imports a registered in-process component to a COM+ application.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstallApplication (see page 3372)	Installs a COM+ application or application proxy from a file.
	InstallComponent (see page 3373)	Installs a component into a COM+ application, and provides default configuration in the COM+ class registration database (RegDB).
	InstallEventClass (see page 3373)	Installs a COM+ event class into an application.
	InstallMultipleComponents (see page 3374)	Installs components from multiple files into a COM+ application.
	InstallMultipleEventClasses (see page 3375)	Installs event classes from multiple files into a COM+ application.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryApplicationFile (see page 3375)	Retrieves information about a COM+ application from a COM+ application file.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshComponents (see page 3376)	Refreshes component registration information from the Registry (see page 18633).
	RefreshRouter (see page 3376)	Refreshes the data being used by the component load balancing service on a CLB server.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreREGDB (see page 3376)	Restores the COM+ class registration database (RegDB) from a specified file.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.

V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ServiceCheck (see page 3377)	Checks the status of a COM+ service.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	ShutdownApplication (see page 3377)	Initiates shutdown of a COM+ application server process.
	StartApplication (see page 3378)	Launches a COM+ server application in a dedicated process.
	StartRouter (see page 3378)	Starts the component load balancing service on a CLB server.
	StopRouter (see page 3378)	Stops the component load balancing service on a CLB server.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	DefaultInterface (see page 3379)	Provides access to the default interface for the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	MajorVersion (see page 3379)	Indicates the major version number of the COMAdmin Library
R	MinorVersion (see page 3380)	Indicates the minor version number of the COMAdmin Library
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	AutoConnect (see page 18110)	Specifies whether the component automatically connects to the server.
	ConnectKind (see page 18110)	Indicates the type of process to which the ConnectKind component establishes a connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RemoteMachineName (see page 18111)	Specifies the machine running the COM server when ConnectKind (see page 18110) is ckRemote.
	Tag (see page 2808)	Stores an integer value as part of a component.

25.1.4.2 TCOMAdminCatalog.Create Constructor

Creates a new instance of TCOMAdminCatalog (see page 3359).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCOMAdminCatalog(TComponent * AOwner);
```

Description

Call Create to instantiate TCOMAdminCatalog (see page 3359) at runtime. TCOMAdminCatalog (see page 3359) objects added at design time are instantiated automatically.

Owner specifies the component (typically a form or data module) that is responsible for freeing the TCOMAdminCatalog (see page 3359) instance.

See Also

Owner (see page 2807)

25.1.4.3 TCOMAdminCatalog.Destroy Destructor

Frees an instance of TCOMAdminCatalog (see page 3359)

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCOMAdminCatalog();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TCOMAdminCatalog (see page 3359) component reference is not nil before calling Destroy. Destroy disconnects from the server, if necessary, before freeing the component.

See Also

Free (see page 22300)

25.1.4.4 TCOMAdminCatalog Methods

The methods of the TCOMAdminCatalog class are listed here.

Protected Methods

	Name	Description
	InitServerData (see page 3372)	Initializes the ServerData property to reflect the type library information of the COM server.

Public Methods

	Name	Description
	BackupREGDB (see page 3365)	Backs up the COM+ class registration database to a specified file.
	Connect (see page 3365)	Establishes a connection to the COM server that implements the COM+ administration catalog.
	ConnectTo (see page 3366)	Connects the TCOMAdminCatalog (see page 3359) component to an existing ICOMAdminCatalog interface.
	Disconnect (see page 3366)	Terminates the connection to the COM server.
	ExportApplication (see page 3367)	Exports a COM+ application or application proxy to a file for installation on other machines.
	Get_MajorVersion (see page 3367)	Returns the value of the Get_MajorVersion property.
	Get_MinorVersion (see page 3368)	Returns the value of the Get_MinorVersion property.
	GetCollection (see page 3368)	Returns a TCOMAdminCatalogCollection (see page 3380) object for a specified collection.
	GetCollectionByQuery (see page 3368)	Retrieves a collection from the COM+ catalog given all of its parent items.
	GetEventClassesForIID (see page 3369)	Retrieves a list of event classes that implement a specified interface.
	GetMultipleComponentsInfo (see page 3370)	Returns information about COM+ components found in specified .dll files.
	ICOMAdminCatalog_Connect (see page 3371)	ICOMAdminCatalog_Connects to a remote computer so that you can administer the collections on that machine.
	ImportComponent (see page 3371)	Imports a registered in-process component to a COM+ application.
	InstallApplication (see page 3372)	Installs a COM+ application or application proxy from a file.
	InstallComponent (see page 3373)	Installs a component into a COM+ application, and provides default configuration in the COM+ class registration database (RegDB).
	InstallEventClass (see page 3373)	Installs a COM+ event class into an application.
	InstallMultipleComponents (see page 3374)	Installs components from multiple files into a COM+ application.
	InstallMultipleEventClasses (see page 3375)	Installs event classes from multiple files into a COM+ application.
	QueryApplicationFile (see page 3375)	Retrieves information about a COM+ application from a COM+ application file.

	RefreshComponents (see page 3376)	Refreshes component registration information from the Registry (see page 18633).
	RefreshRouter (see page 3376)	Refreshes the data being used by the component load balancing service on a CLB server.
	RestoreREGDB (see page 3376)	Restores the COM+ class registration database (RegDB) from a specified file.
	ServiceCheck (see page 3377)	Checks the status of a COM+ service.
	ShutdownApplication (see page 3377)	Initiates shutdown of a COM+ application server process.
	StartApplication (see page 3378)	Launches a COM+ server application in a dedicated process.
	StartRouter (see page 3378)	Starts the component load balancing service on a CLB server.
	StopRouter (see page 3378)	Stops the component load balancing service on a CLB server.

25.1.4.4.1 TCOMAdminCatalog.BackupREGDB Method

Backs up the COM+ class registration database to a specified file.

Pascal

```
procedure BackupREGDB(const bstrBackupFilePath: WideString);
```

C++

```
__fastcall BackupREGDB(const BSTR bstrBackupFilePath);
```

Description

Call BackupREGDB to back up the COM+ class registration database (RegDB). The database can be restored at a later date using the RestoreREGDB (see page 3376) method.

bstrBackupFilePath specifies the fully-qualified name of the file to which RegDB will be backed up.

See Also

RestoreREGDB (see page 3376)

InstallComponent (see page 3373)

25.1.4.4.2 TCOMAdminCatalog.Connect Method

Establishes a connection to the COM server that implements the COM+ administration catalog.

Pascal

```
procedure Connect; override;
```

C++

```
virtual __fastcall Connect();
```

Description

Call Connect to connect the TCOMAdminCatalog (see page 3359) component to the COM server it uses to administer the COM+ catalog data store. If the AutoConnect property is true, the TCOMAdminCatalog (see page 3359) component automatically calls Connect when the application is first run.

Do not use Connect if the value of ConnectKind is ckAttachToInterface. When attaching to an interface, use the ConnectTo (see page 3366) method instead.

If the value of ConnectKind is ckRemote, the RemoteMachineName property must be set before calling Connect.

Note: Do not confuse Connect with the ICOMAdminCatalog_Connect (see page 3371) method. Connect establishes a connection to the server that TCOMAdminCatalog (see page 3359) represents, while ICOMAdminCatalog_Connect (see page 3371) connects to a remote machine on which you want to administer COM+ applications and services.

See Also

AutoConnect (see page 18110)

Disconnect (see page 3366)

ConnectKind (see page 18110)

RemoteMachineName (see page 18111)

ICOMAdminCatalog_Connect (see page 3371)

ConnectTo (see page 3366)

25.1.4.4.3 TCOMAdminCatalog.ConnectTo Method

Connects the TCOMAdminCatalog (see page 3359) component to an existing ICOMAdminCatalog interface.

Pascal

```
procedure ConnectTo(svrIntf: ICOMAdminCatalog);
```

C++

```
__fastcall ConnectTo(ICOMAdminCatalog svrIntf);
```

Description

Use ConnectTo when the ConnectKind property is ckAttachToInterface. ConnectTo allows the TCOMAdminCatalog (see page 3359) to use an existing ICOMAdminCatalog interface rather than establishing a new connection to the server and obtaining a new interface.

If the ConnectKind property is not ckAttachToInterface, use the Connect (see page 3365) method instead.

See Also

Connect (see page 3365)

Disconnect (see page 3366)

ConnectKind (see page 18110)

25.1.4.4.4 TCOMAdminCatalog.Disconnect Method

Terminates the connection to the COM server.

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

Call Disconnect to release a connection formed using the Connect (see page 3365) or ConnectTo (see page 3366) method.

See Also

AutoConnect (see page 18110)

Connect (see page 3365)

ConnectTo (see page 3366)

25.1.4.4.5 TCOMAdminCatalog.ExportApplication Method

Exports a COM+ application or application proxy to a file for installation on other machines.

Pascal

```
procedure ExportApplication(const bstrApplIdOrName: WideString; const bstrApplicationFile:
WideString; lOptions: Integer);
```

C++

```
__fastcall ExportApplication(const BSTR bstrApplIdOrName, const BSTR bstrApplicationFile, int
lOptions);
```

Description

Use ExportApplication to create a file that the InstallApplication (see page 3372) method can use to install a COM+ application or application proxy on other machines.

bstrApplIdOrName is the name or GUID that identifies the application to be exported.

bstrApplicationFile is the name of the file to which ExportApplication exports bstrApplIDOrName. This file name should include the file path. If you do not provide a file name, ExportApplication creates a file with the same name as the application in the default directory.

lOptions indicate how the application should be exported. The value of lOptions is a sum that represents a set of the following flags.

Flag	Meaning
0	Do not export users with roles.
1	Export users with roles.
2	Export application as an application proxy.
4	Overwrite files.

If no option flags are provided, ExportApplication exports the specified COM+ application to an MSI file, without the users assigned to application roles.

See Also

InstallApplication (see page 3372)

QueryApplicationFile (see page 3375)

25.1.4.4.6 TCOMAdminCatalog.Get_MajorVersion Method

Returns the value of the Get_MajorVersion property.

Pascal

```
function Get_MajorVersion: Integer;
```

C++

```
__fastcall int Get_MajorVersion();
```

Description

Get_Get_MajorVersion is the read implementation of the Get_MajorVersion property.

See Also

Get_MajorVersion

25.1.4.4.7 TCOMAdminCatalog.Get_MinorVersion Method

Returns the value of the Get_MinorVersion property.

Pascal

```
function Get_MinorVersion: Integer;
```

C++

```
__fastcall int Get_MinorVersion();
```

Description

Get_Get_MinorVersion is the read implementation of the Get_MinorVersion property.

See Also

Get_MinorVersion

25.1.4.4.8 TCOMAdminCatalog.GetCollection Method

Returns a TCOMAdminCatalogCollection (see page 3380) object for a specified collection.

Pascal

```
function GetCollection(const bstrCollName: WideString): TCOMAdminCatalogCollection;
```

C++

```
__fastcall TCOMAdminCatalogCollection GetCollection(const BSTR bstrCollName);
```

Description

GetCollection retrieves a top-level collection from the COM+ catalog, such as Applications. Subcollections (collections that are not top-level collections, such as Components) must be fetched using the GetCollection method of their parent collection or using the GetCollectionByQuery (see page 3368) method.

bstrCollName is the name of the desired collection. The available collections depend on your version of the COMAdmin Library. See the Microsoft documentation for a list of the available collections in your version of the COMAdmin Library.

Note: The TCOMAdminCatalogCollection (see page 3380) object that GetCollection returns does not contain data from the catalog for items contained in the collection. Use the Populate method to read in data for items in the collection.

See Also

GetCollectionByQuery (see page 3368)

Populate (see page 3391)

GetCollection (see page 3389)

ICOMAdminCatalog_Connect (see page 3371)

25.1.4.4.9 TCOMAdminCatalog.GetCollectionByQuery Method

Retrieves a collection from the COM+ catalog given all of its parent items.

Pascal

```
function GetCollectionByQuery(const bstrCollName: WideString; var aQuery: PSafeArray):
TCOMAdminCatalogCollection;
```

C++

```
__fastcall TCOMAdminCatalogCollection GetCollectionByQuery(const BSTR bstrCollName, PSafeArray
aQuery);
```

Description

Call GetCollectionByQuery to obtain a collection on the catalog given the Key property of each of its parent items. The Key property for each parent item is a GUID that uniquely identifies it on the COM+ catalog.

bstrCollName is the name of the desired collection. The available collections depend on your version of the COMAdmin Library. See the Microsoft documentation for a list of the available collections in your version of the COMAdmin Library.

aQuery is a safe array of containing the Key property for each parent item of the desired collection.

Note: If bstrCollName is 'ErrorInfo', 'RelatedCollectionInfo', or 'PropertyInfo', GetCollectionByQuery ignores the aQuery parameter and returns the named collection immediately relative to the Root collection. To obtain these collections relative to an arbitrary collection in the catalog, use the GetCollection (see page 3368) method of the parent collection.

Note: The TCOMAdminCatalogCollection (see page 3380) object that GetCollectionByQuery returns does not contain data from the catalog for items contained in the collection. Use the Populate method to read in data for items in the collection.

See Also

Populate (see page 3391)

GetCollection (see page 3368)

Key (see page 3405)

GetCollection (see page 3389)

25.1.4.4.10 TCOMAdminCatalog.GetEventClassesForIID Method

Retrieves a list of event classes that implement a specified interface.

Pascal

```
procedure GetEventClassesForIID(const bstrIID: WideString; out varCLSIDS: PSafeArray; out
varProgIDs: PSafeArray; out varDescriptions: PSafeArray);
```

C++

```
__fastcall GetEventClassesForIID(const BSTR bstrIID, PSafeArray varCLSIDS, PSafeArray
varProgIDs, PSafeArray varDescriptions);
```

Description

Call GetEventClassesForIID to determine the available COM+ event objects that implement a specified interface.

bstrIID is the GUID for the interface that you want the event classes to implement. If bstrIID is nil (Delphi) or NULL (C++), GetEventClassesForIID returns all event classes registered on the machine.

varCLSIDS returns an array of CLSIDs for the event classes that implement bstrIID.

varProgIDS returns an array of ProgIDs for the event classes that implement bstrIID.

varDescriptions returns an array of description strings for the event classes that implement bstrIID.

See Also

InstallEventClass (see page 3373)

InstallMultipleEventClasses (see page 3375)

25.1.4.4.11 TCOMAdminCatalog.GetMultipleComponentsInfo Method

Returns information about COM+ components found in specified .dll files.

Pascal

```
procedure GetMultipleComponentsInfo(const bstrApplIdOrName: WideString; var varFileNames:
PSafeArray; out varCLSIDS: PSafeArray; out varClassNames: PSafeArray; out varFileFlags:
PSafeArray; out varComponentFlags: PSafeArray);
```

C++

```
__fastcall GetMultipleComponentsInfo(const BSTR bstrApplIdOrName, PSafeArray varFileNames,
PSafeArray varCLSIDS, PSafeArray varClassNames, PSafeArray varFileFlags, PSafeArray
varComponentFlags);
```

Description

Call GetMultipleComponentsInfo to obtain information on the objects that a specified set of .dll files make available.

bstrApplIdOrName specifes the name or GUID of the application to which the .dll files belong.

varFileNames is an array of WideString values, each of which is the name of a .dll.

varCLSIDS returns an array of CLSIDs for the COM+ components in the specified .dll files.

varClassNames returns an array of class names for the component classes in the specified .dll files.

varFileFlags returns an array of file flags that return information about the specified files. Each entry in the array is built from the following flags:

FlagValue

ComAdminFileFlagLoadable1

ComAdminFileFlagCOM2

COMAdminFileFlagContainsRS4

COMAdminFileFlagContainsComp8

COMAdminFileFlagContainsTLB16

COMAdminFileFlagSelfReg32

COMAdminFileFlagSelfUnReg64

COMAdminFileFlagUnloadableDLL128

COMAdminFileFlagDoesNotExist256

COMAdminFileFlagAlreadyInstalled512

COMAdminFileFlagBadTLB1024

COMAdminFileFlagGetClassObjFailed2048

COMAdminFileFlagClassNotAvailable4096

COMAdminFileFlagRegistrar8192

COMAdminFileFlagNoRegistrar16384

COMAdminFileFlagDLLRegsvrFailed32768

COMAdminFileFlagRegTLBFailed65536

COMAdminFileFlagRegistrarFailed131072

COMAdminFileFlagError262144

varComponentFlags returns an array of file flags that return information about the components in the specified files. Each entry in the array is built from the following flags:

FlagValue

COMAdminComponentFlagTypeInfoFound1

COMAdminComponentCOMPlusPropertiesFound2

COMAdminComponentProxyFound4

COMAdminComponentInterfacesFound8

COMAdminComponentAlreadyInstalled16

COMAdminComponentNotInApplication32

See Also

InstallMultipleComponents (see page 3374)

RefreshComponents (see page 3376)

25.1.4.4.12 TCOMAdminCatalog.ICOMAdminCatalog_Connect Method

ICOMAdminCatalog_Connects to a remote computer so that you can administer the collections on that machine.

Pascal

```
function ICOMAdminCatalog_Connect(const bstrConnectString: WideString):
TCOMAdminCatalogCollection;
```

C++

```
__fastcall TCOMAdminCatalogCollection ICOMAdminCatalog_Connect(const BSTR bstrConnectString);
```

Description

Call ICOMAdminCatalog_ICOMAdminCatalog_Connect to administer the COM+ components and services on a remote machine.

bstrICOMAdminCatalog_ConnectString is the name of the remote computer whose COM+ components and services you want to administer.

ICOMAdminCatalog_ICOMAdminCatalog_Connect returns the root collection on the remote computer. The root collection contains the top-level collections on the catalog of the remote machine. It does not contain any TCOMAdminCatalogObject (see page 3396) objects or support any properties. You can't use the Add or Remove methods of this TCOMAdminCatalogCollection (see page 3380) object.

See Also

GetCollection (see page 3368)

25.1.4.4.13 TCOMAdminCatalog.ImportComponent Method

Imports a registered in-process component to a COM+ application.

Pascal

```
procedure ImportComponent(const bstrApplIdOrName: WideString; const bstrCLSIDOrProgId:
WideString);
```

C++

```
__fastcall ImportComponent(const BSTR bstrApplIdOrName, const BSTR bstrCLSIDOrProgId);
```

Description

Do not use ImportComponent. Use the InstallComponent (see page 3373) method instead. InstallComponent (see page 3373) fully registers the component in the COM+ class registration database (RegDB), whereas ImportComponent does not bring any interface, method, or type library information for the component into the COM+ class registration database. This restricts how the component can be configured. When you export a COM+ application that has an imported component to an application proxy, the proxy will contain no interface or type library information for the component, and marshalling for that component will fail.

bstrAppIdOrName is the name or GUID that represents the application into which to import the component.

bstrCLSIDOrProgID is the CLSID or ProgID that identifies the component to import.

See Also

InstallComponent (see page 3373)

25.1.4.4.14 TCOMAdminCatalog.InitServerData Method

Initializes the ServerData property to reflect the type library information of the COM server.

Pascal

```
procedure InitServerData; override;
```

C++

```
virtual __fastcall InitServerData();
```

Description

The InitServerData constructor calls InitServerData to initialize ServerData with information from the COM server's type library.

InitServerData is an abstract a pure virtual method in InitServerData. Each InitServerData descendant implements this method so that the ServerData property reflects its particular COM server.

See Also

ServerData (see page 18111)

Create (see page 18101)

25.1.4.4.15 TCOMAdminCatalog.InstallApplication Method

Installs a COM+ application or application proxy from a file.

Pascal

```
procedure InstallApplication(const bstrApplicationFile: WideString; const
bstrDestinationDirectory: WideString; lOptions: Integer; const bstrUserId: WideString; const
bstrPassword: WideString; const bstrRSN: WideString);
```

C++

```
__fastcall InstallApplication(const BSTR bstrApplicationFile, const BSTR
bstrDestinationDirectory, int lOptions, const BSTR bstrUserId, const BSTR bstrPassword, const
BSTR bstrRSN);
```

Description

Use InstallApplication to install an application or application proxy that was previously saved by a call to the ExportApplication (see page 3367) method.

bstrApplicationFile is the name of the file created by ExportApplication (see page 3367).

bstrDestinationDirectory is the directory where the components in the application should be installed. If this parameter is nil (Delphi) or NULL (C++), InstallApplication uses the default directory.

lOptions is a set of flags that modify how the application is installed. It is a sum of 0 or more of the following flags:

Flag	Meaning
0	Don't install any users saved in the application file.
1	Install all users that are saved in the application file.
2	Overwrite any files that already exist in the target directory.

bstrUserId specifies the user ID under which to run the application. This value can be nil (Delphi) or NULL (C++).

bstrPassword specifies the password under which to run the application. This value can be nil (Delphi) or NULL (C++).

bstrRSN specifies the remote server name when bstrApplicationFile is an application proxy.

See Also

ExportApplication (see page 3367)

QueryApplicationFile (see page 3375)

25.1.4.4.16 TCOMAdminCatalog.InstallComponent Method

Installs a component into a COM+ application, and provides default configuration in the COM+ class registration database (RegDB).

Pascal

```
procedure InstallComponent(const bstrApplIdOrName: WideString; const bstrDLL: WideString;
const bstrTLB: WideString; const bstrPSDLL: WideString);
```

C++

```
__fastcall InstallComponent(const BSTR bstrApplIdOrName, const BSTR bstrDLL, const BSTR
bstrTLB, const BSTR bstrPSDLL);
```

Description

InstallComponent installs all components (COM classes) in a .dll file into a COM+ application, and registers the components in the COM+ class registration database (RegDB) as configured components, providing a default configuration.

bstrApplIdOrName is the name or GUID that represents the application into which to install the components.

bstrDLL is the name of the .dll file that implements the component or components.

bstrTLB is the name of the external type library file for the .dll. If the type library is embedded in the .dll, the parameter should be an empty string.

bstrPSDLL is the name of the proxy-stub .dll file. If there is no proxy-stub .dll associated with the component, set this parameter to an empty string.

See Also

InstallMultipleComponents (see page 3374)

RefreshComponents (see page 3376)

25.1.4.4.17 TCOMAdminCatalog.InstallEventClass Method

Installs a COM+ event class into an application.

Pascal

```
procedure InstallEventClass(const bstrApplIdOrName: WideString; const bstrDLL: WideString;
const bstrTLB: WideString; const bstrPSDLL: WideString);
```

C++

```
__fastcall InstallEventClass(const BSTR bstrApplIdOrName, const BSTR bstrDLL, const BSTR
bstrTLB, const BSTR bstrPSDLL);
```

Description

Use InstallEventClass to install event classes from a .dll that holds dummy implementations of the event classes. The .dll must be self-registering, there must be a type library to describe the interfaces that the event classes implement, and each event class must have a CLSID and a ProgID.

The dummy implementation of the event class is never actually run. It exists only to register the event class. When the publisher creates the event class, the Events system provides an implementation to send the event to subscribers.

bstrApplIdOrName is the name or GUID that represents the application into which to install the event classes.

bstrDLL is the name of the self-registering .dll file that contains the event classes.

bstrTLB is the name of the external type library file for the .dll. If the type library is embedded in the .dll, set this parameter to an empty string.

bstrPSDLL is the name of the proxy-stub .dll file. If there is no proxy-stub .dll associated with the event class, set this parameter to an empty string.

See Also

InstallMultipleEventClasses (see page 3375)

25.1.4.4.18 TCOMAdminCatalog.InstallMultipleComponents Method

Installs components from multiple files into a COM+ application.

Pascal

```
procedure InstallMultipleComponents(const bstrApplIdOrName: WideString; var varFileNames:
PSafeArray; var varCLSIDS: PSafeArray);
```

C++

```
__fastcall InstallMultipleComponents(const BSTR bstrApplIdOrName, PSafeArray varFileNames,
PSafeArray varCLSIDS);
```

Description

InstallMultipleComponents installs the components in the specified files into the specified COM+ application.

bstrApplIdOrName is the name or GUID that represents the application into which to install the components.

varFileNames is an array of strings, where each string is the name of a .dll file that implements the components to install.

varCLSIDS is an array of Variants (see page 23567) holding CLSIDs for the components to install.

See Also

InstallComponent (see page 3373)

ImportComponent (see page 3371)

25.1.4.4.19 TCOMAdminCatalog.InstallMultipleEventClasses Method

Installs event classes from multiple files into a COM+ application.

Pascal

```
procedure InstallMultipleEventClasses(const bstrApplIdOrName: WideString; var varFileNames:
PSafeArray; var varCLSIDS: PSafeArray);
```

C++

```
__fastcall InstallMultipleEventClasses(const BSTR bstrApplIdOrName, PSafeArray varFileNames,
PSafeArray varCLSIDS);
```

Description

Use InstallMultipleEventClasses to install event classes from multiple self-registering .dll files, each of which holds the dummy implementation of one or more COM+ event classes. There must be a type library to describe the interfaces that the event classes implement, and each event class must have a CLSID and a ProgID.

The dummy implementation of the event class is never actually run. It exists only to register the event class. When the publisher creates the event class, the Events system provides an implementation to send the event to subscribers.

bstrApplIdOrName is the name or GUID that represents the application into which to install the event classes.

varFileNames is an array of strings, where each string is the name of a self-registering .dll file that contains the event classes to install.

varCLSIDS is an array of Variants (see page 23567) holding CLSIDs for the event classes to install.

See Also

InstallEventClass (see page 3373)

25.1.4.4.20 TCOMAdminCatalog.QueryApplicationFile Method

Retrieves information about a COM+ application from a COM+ application file.

Pascal

```
procedure QueryApplicationFile(const bstrApplicationFile: WideString; out bstrApplicationName:
WideString; out bstrApplicationDescription: WideString; out bHasUsers: WordBool; out bIsProxy:
WordBool; out varFileNames: PSafeArray);
```

C++

```
__fastcall QueryApplicationFile(const BSTR bstrApplicationFile, BSTR bstrApplicationName, BSTR
bstrApplicationDescription, WordBool bHasUsers, WordBool bIsProxy, PSafeArray varFileNames);
```

Description

Call QueryApplicationFile to obtain information about a COM+ application from its application file. An application file is created by the ExportApplication (see page 3367) method.

bstrApplicationFile is the name of the ApplicationFile from which to retrieve information.

bstrApplicationName returns the name of the application.

bstrApplicationDescription returns a brief description of the application.

bHasUsers returns true if the application file includes user information associated with its roles, false otherwise.

bIsProxy returns true if the application file contains an application proxy, false otherwise.

varFileNames returns an array of all .dll files containing components that are installed in the application.

See Also

ExportApplication (see page 3367)

25.1.4.4.21 TCOMAdminCatalog.RefreshComponents Method

Refreshes component registration information from the Registry (see page 18633).

Pascal

```
procedure RefreshComponents;
```

C++

```
__fastcall RefreshComponents();
```

Description

RefreshComponents compares CLSIDs and ProgIDs between the COM+ class registration database (RegDB) and the Registry (see page 18633) for mismatches. When it detects a mismatch in CLSIDs and ProgIDs, RefreshComponents updates the COM+ class registration database (RegDB).

RefreshComponents only updates CLSID information. It does not update interface or method information. Components that are refreshed using RefreshComponents can not be configured or secured at the interface or method levels in COM+ applications.

Note: You generally should not use RefreshComponents. The recommended way to update components in COM+ applications is to remove and reinstall the components using InstallComponent (see page 3373), so that complete registration information is updated in RegDB.

See Also

InstallComponent (see page 3373)

25.1.4.4.22 TCOMAdminCatalog.RefreshRouter Method

Refreshes the data being used by the component load balancing service on a CLB server.

Pascal

```
procedure RefreshRouter;
```

C++

```
__fastcall RefreshRouter();
```

Description

Do not use the RefreshRouter method. It is deprecated.

See Also

RestoreREGDB (see page 3376)

ServiceCheck (see page 3377)

25.1.4.4.23 TCOMAdminCatalog.RestoreREGDB Method

Restores the COM+ class registration database (RegDB) from a specified file.

Pascal

```
procedure RestoreREGDB(const bstrBackupFilePath: WideString);
```

C++

```
__fastcall RestoreREGDB(const BSTR bstrBackupFilePath);
```

Description

Call RestoreREGDB to restore the COM+ class registration database (RegDB) from a file that was previously created by the BackupREGDB (see page 3365) method This requires a system reboot in order to take effect.

bstrBackupFilePath is the file previously created by the BackupREGDB (see page 3365) method.

See Also

BackupREGDB (see page 3365)

RefreshComponents (see page 3376)

25.1.4.4.24 TCOMAdminCatalog.ServiceCheck Method

Checks the status of a COM+ service.

Pascal

```
function ServiceCheck(lService: Integer): Integer;
```

C++

```
__fastcall int ServiceCheck(int lService);
```

Description

Call ServiceCheck to check the status of the service specified by lService. A value of 1 specifies the Component Load Balancing service. At present, TCOMAdminCatalog (see page 3359) does not support any other services.

SericeCheck returns one of the following values:

Value	Meaning
0	Service is currently stopped.
1	Service is due to start up.
2	Service is due to stop.
3	Service is running normally.
4	Service is due to continue.
5	Service is due to pause.
6	Service is currently paused.
7	Service status is unknown.

25.1.4.4.25 TCOMAdminCatalog.ShutdownApplication Method

Initiates shutdown of a COM+ application server process.

Pascal

```
procedure ShutdownApplication(const bstrApplIdOrName: WideString);
```

C++

```
__fastcall ShutdownApplication(const BSTR bstrApplIdOrName);
```

Description

Call ShutdownApplication to shut down a COM+ application process.

bstrApplIdOrName is the GUID or name that identifies the application to shut down.

See Also

StartApplication (see page 3378)

25.1.4.4.26 TCOMAdminCatalog.StartApplication Method

Launches a COM+ server application in a dedicated process.

Pascal

```
procedure StartApplication(const bstrApplIdOrName: WideString);
```

C++

```
__fastcall StartApplication(const BSTR bstrApplIdOrName);
```

Description

Call StartApplication to start up a COM+ server application. The components in the application are launched in a new dedicated process.

bstrApplIdOrName is the GUID or name that identifies the application to start.

See Also

ShutdownApplication (see page 3377)

25.1.4.4.27 TCOMAdminCatalog.StartRouter Method

Starts the component load balancing service on a CLB server.

Pascal

```
procedure StartRouter;
```

C++

```
__fastcall StartRouter();
```

Description

StartRouter starts the component load balancing service if it is installed. Subsequent component activation requests are then routed across the application cluster that the router defines.

StartRouter does nothing if called on a machine that does not act as the component load balancing (CLB) server.

See Also

StopRouter (see page 3378)

ServiceCheck (see page 3377)

25.1.4.4.28 TCOMAdminCatalog.StopRouter Method

Stops the component load balancing service on a CLB server.

Pascal

```
procedure StopRouter;
```

C++

```
__fastcall StopRouter();
```

Description

When called on a machine acting as the component load balancing (CLB) server, StopRouter stops the server from routing component activation requests to other servers in the application cluster.

See Also

StartRouter (see page 3378)

ServiceCheck (see page 3377)

25.1.4.5 TCOMAdminCatalog Properties

The properties of the TCOMAdminCatalog class are listed here.

Public Properties

	Name	Description
R	DefaultInterface (see page 3379)	Provides access to the default interface for the component.
R	MajorVersion (see page 3379)	Indicates the major version number of the COMAdmin Library
R	MinorVersion (see page 3380)	Indicates the minor version number of the COMAdmin Library

25.1.4.5.1 TCOMAdminCatalog.DefaultInterface Property

Provides access to the default interface for the component.

Pascal

```
property DefaultInterface: ICOMAdminCatalog;
```

C++

```
__property ICOMAdminCatalog DefaultInterface;
```

Description

Use DefaultInterface to access the default interface for TCOMAdminCatalog (see page 3359). This is the ICOMAdminCatalog interface, whose properties and methods are surfaced as properties and methods of TCOMAdminCatalog (see page 3359).

25.1.4.5.2 TCOMAdminCatalog.MajorVersion Property

Indicates the major version number of the COMAdmin Library

Pascal

```
property MajorVersion: Integer;
```

C++

```
__property int MajorVersion;
```

Description

Use MajorVersion to determine the major version number of the COMAdmin Library, which TCOMAdminCatalog (see page 3359) uses to administer COM+ applications.

See Also

MinorVersion (see page 3380)

25.1.4.5.3 TCOMAdminCatalog.MinorVersion Property

Indicates the minor version number of the COMAdmin Library

Pascal

```
property MinorVersion: Integer;
```

C++

```
__property int MinorVersion;
```

Description

Use MinorVersion to determine the minor version number of the COMAdmin Library, which TCOMAdminCatalog (see page 3359) uses to administer COM+ applications.

See Also

MajorVersion (see page 3379)

25.1.5 TCOMAdminCatalogCollection Class

TCOMAdminCatalogCollection represents a collection from the COM+ catalog.

Class Hierarchy

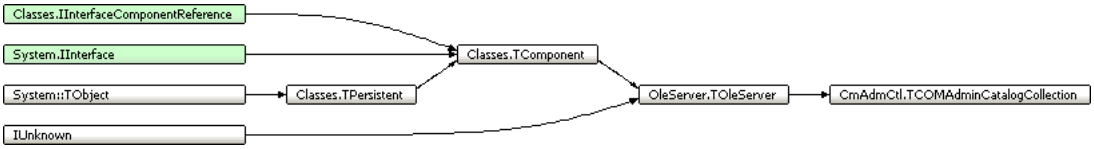

Pascal

```
TCOMAdminCatalogCollection = class(TOleServer);
```

C++

```
class TCOMAdminCatalogCollection : public TOleServer;
```

File

CmAdmCtl

Description

Use TCOMAdminCatalogCollection to work with one of the collections in the COM+ catalog. Use the GetCollection (see page 3389) method of TCOMAdminCatlog to obtain a TCOMAdminCatalogCollection object for a specific collection. Each collection corresponds to a folder that appears in the Component Services administration tool.

TCOMAdminCatalogCollection implements the ICatalogCollection interface, to let you enumerate, add, remove, and retrieve items in a collection, and to access related collections. Each item is represented by a TCOMAdminCatalogObject (see page 3396) instance.

Note: Typically, when you first obtain a TCOMAdminCatalogCollection object, it does not contain data from the catalog for items in the collection. Use the Populate (see page 3391) method to read in data for items in the collection before using the other properties and methods.

See Also

TCOMAdminCatalog (see page 3359)

TCOMAdminCatalogObject (see page 3396)

TCOMAdminCatalog (see page 3359)

25.1.5.1 TCOMAdminCatalogCollection Members

The following tables list the members exposed by TCOMAdminCatalogCollection.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_AddRef (see page 18104)	Increments the reference count for the number of interface references dependent on the OLE server.
	_Release (see page 2769)	Called when an application releases a component interface.
	_Release (see page 18104)	Decrements the reference count for the calling interface on a object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ConnectEvents (see page 18105)	Establishes a connection using the EventDispatch (see page 18111) property as an event sink.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DisconnectEvents (see page 18106)	Terminates a connection that was established using the ConnectEvents (see page 18105) method.
	GetAutoConnect (see page 18106)	Returns the value of the AutoConnect (see page 18110) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetConnectKind (see page 18106)	Returns the value of the ConnectKind (see page 18110) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetServer (see page 18107)	Returns an interface for the COM server.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InitServerData (see page 3390)	Initializes the ServerData property to reflect the type library information of the COM server.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	InvokeEvent (see page 18107)	Dispatches an event from the COM server to the appropriate event handler.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 18108)	Connects to the COM server when the Loaded object is first loaded into memory if AutoConnect (see page 18110) is True true.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	QueryInterface (see page 18108)	Returns a reference to a specified interface if the OLE server supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.

	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoConnect (see page 18109)	Sets the value of the AutoConnect (see page 18110) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnectKind (see page 18109)	Sets the value of the ConnectKind (see page 18110) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	EventDispatch (see page 18111)	This is EventDispatch, a member of class TOleServer.
	ServerData (see page 18111)	Encodes information about the COM server to which ServerData connects.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Add (see page 3386)	Adds a new item to the end of the collection.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Connect (see page 3386)	Not used in TCOMAdminCatalogCollection (see page 3380).
	ConnectTo (see page 3386)	Connects the TCOMAdminCatalogCollection (see page 3380) component to an existing ICatalogCollection interface.
	Create (see page 3384)	Creates a new instance of TCOMAdminCatalogCollection (see page 3380).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 3385)	Frees an instance of TCOMAdminCatalogCollection (see page 3380)
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Disconnect (see page 3387)	Terminates the connection to the COM server.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	Get_AddEnabled (see page 3387)	Returns the value of the Get_AddEnabled property.
	Get_Count (see page 3388)	Returns the number of items in the collection.
	Get_DataStoreMajorVersion (see page 3388)	Returns the value of the Get_DataStoreMajorVersion property.
	Get_DataStoreMinorVersion (see page 3388)	Returns the value of the Get_DataStoreMinorVersion property.
	Get_Item (see page 3389)	Returns a specified item from the collection.
	Get_Name (see page 3389)	Returns the value of the Get_Name property
	Get_RemoveEnabled (see page 3389)	Returns the value of the Get_RemoveEnabled property.
	GetCollection (see page 3389)	Returns a TCOMAdminCatalogCollectionCollection object for a specified collection.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetUtilInterface (see page 3390)	Returns the utility interface associated with this collection.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Populate (see page 3391)	Fills the collection object with items from the catalog.
	PopulateByKey (see page 3391)	Populates the collection with information for the specified objects.
	PopulateByQuery (see page 3392)	Reserved for future use.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Remove (see page 3392)	Removes an item from the collection
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveChanges (see page 3392)	Commits any changes to the collection by writing them to the COM+ catalog.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AddEnabled (see page 3393)	Indicates whether the Add (see page 3386) method can add items to the collection.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Count (see page 3393)	Indicates the number of items in the collection.
R	DataStoreMajorVersion (see page 3394)	Indicates the major version number of the COMAdmin Library
R	DataStoreMinorVersion (see page 3394)	Indicates the minor version number of the COMAdmin Library
R	DefaultInterface (see page 3395)	Provides access to the default interface for the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Name (see page 3395)	Indicates the Name of the collection.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	RemoveEnabled (see page 3395)	Indicates whether the Remove (see page 3392) method can remove items from the collection.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	AutoConnect (see page 18110)	Specifies whether the component automatically connects to the server.
	ConnectKind (see page 18110)	Indicates the type of process to which the ConnectKind component establishes a connection.
	RemoteMachineName (see page 18111)	Specifies the machine running the COM server when ConnectKind (see page 18110) is ckRemote.
	Tag (see page 2808)	Stores an integer value as part of a component.

25.1.5.2 TCOMAdminCatalogCollection.Create Constructor

Creates a new instance of TCOMAdminCatalogCollection (see page 3380).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCOMAdminCatalogCollection(TComponent * AOwner);
```

Description

Call Create to instantiate TCOMAdminCatalogCollection (see page 3380) at runtime. Once you have created a TCOMAdminCatalogCollection (see page 3380) object, you must bind it to an ICatalogCollection interface before you can use it. Use the ConnectTo (see page 3386) method to bind a TCOMAdminCatalogCollection (see page 3380) object to an ICatalogCollection interface.

Note: When you obtain a TCOMAdminCatalogCollection (see page 3380) instance from a method on the TCOMAdminCatalog (see page 3359) object or from another collection, it is already bound to an ICatalogCollection interface.

Owner specifies the component (typically a form or data module) that is responsible for freeing the TCOMAdminCatalogCollection (see page 3380) instance.

See Also

Owner (see page 2807)

ConnectTo (see page 3386)

25.1.5.3 TCOMAdminCatalogCollection.Destroy Destructor

Frees an instance of TCOMAdminCatalogCollection (see page 3380)

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCOMAdminCatalogCollection();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TCOMAdminCatalogCollection (see page 3380) component reference is not nil before calling Destroy.

Note that freeing the TCOMAdminCatalogCollection (see page 3380) object does not remove the collection from the COM+ catalog, it merely deletes the object that lets you work with that collection.

See Also

Free (see page 22300)

25.1.5.4 TCOMAdminCatalogCollection Methods

The methods of the TCOMAdminCatalogCollection class are listed here.

Protected Methods

	Name	Description
	InitServerData (see page 3390)	Initializes the ServerData property to reflect the type library information of the COM server.

Public Methods

	Name	Description
	Add (see page 3386)	Adds a new item to the end of the collection.
	Connect (see page 3386)	Not used in TCOMAdminCatalogCollection (see page 3380).
	ConnectTo (see page 3386)	Connects the TCOMAdminCatalogCollection (see page 3380) component to an existing ICatalogCollection interface.
	Disconnect (see page 3387)	Terminates the connection to the COM server.
	Get_AddEnabled (see page 3387)	Returns the value of the Get_AddEnabled property.
	Get_Count (see page 3388)	Returns the number of items in the collection.
	Get_DataStoreMajorVersion (see page 3388)	Returns the value of the Get_DataStoreMajorVersion property.
	Get_DataStoreMinorVersion (see page 3388)	Returns the value of the Get_DataStoreMinorVersion property.
	Get_Item (see page 3389)	Returns a specified item from the collection.
	Get_Name (see page 3389)	Returns the value of the Get_Name property
	Get_RemoveEnabled (see page 3389)	Returns the value of the Get_RemoveEnabled property.
	GetCollection (see page 3389)	Returns a TCOMAdminCatalogCollectionCollection object for a specified collection.
	GetUtilInterface (see page 3390)	Returns the utility interface associated with this collection.
	Populate (see page 3391)	Fills the collection object with items from the catalog.
	PopulateByKey (see page 3391)	Populates the collection with information for the specified objects.

	PopulateByQuery (see page 3392)	Reserved for future use.
	Remove (see page 3392)	Removes an item from the collection
	SaveChanges (see page 3392)	Commits any changes to the collection by writing them to the COM+ catalog.

25.1.5.4.1 TCOMAdminCatalogCollection.Add Method

Adds a new item to the end of the collection.

Pascal

```
function Add: TCOMAdminCatalogObject;
```

C++

```
__fastcall TCOMAdminCatalogObject Add();
```

Description

Call the Add method to create a new object in a collection. Add returns the TCOMAdminCatalogObject (see page 3396) that represents the new item.

Not all collections allow you to add items. Check the AddEnabled (see page 3393) property before trying to call Add. To install or create objects in collections where AddEnabled (see page 3393) is false, use the catalog utility interfaces.

Note: Add does not commit the change to the catalog data store. To cause the new item to be written to the data store, call the SaveChanges (see page 3392) method.

See Also

SaveChanges (see page 3392)

AddEnabled (see page 3393)

25.1.5.4.2 TCOMAdminCatalogCollection.Connect Method

Not used in TCOMAdminCatalogCollection (see page 3380).

Pascal

```
procedure Connect; override;
```

C++

```
virtual __fastcall Connect();
```

Description

Do not use the Connect method. TCOMAdminCatalogCollection (see page 3380) objects never directly launch the COM server. Rather, obtain a TCOMAdminCatalogCollection (see page 3380) object that is already bound to an ICatalogCollection interface from a method such as the parent collection's GetCollection (see page 3389) method.

See Also

ConnectTo (see page 3386)

GetCollection (see page 3368)

GetCollection (see page 3389)

25.1.5.4.3 TCOMAdminCatalogCollection.ConnectTo Method

Connects the TCOMAdminCatalogCollection (see page 3380) component to an existing ICatalogCollection interface.

Pascal

```
procedure ConnectTo(svrIntf: ICatalogCollection);
```

C++

```
__fastcall ConnectTo(ICatalogCollection svrIntf);
```

Description

Use ConnectTo when the ConnectKind property is ckAttachToInterface. ConnectTo allows the TCOMAdminCatalogCollection (see page 3380) to use an existing ICatalogCollection interface.

Typically, applications do not need to use the ConnectTo method. Rather, they obtain a TCOMAdminCatalogCollection (see page 3380) object that is already bound to an ICatalogCollection object by calling a method such as GetCollection (see page 3389).

See Also

Connect (see page 3386)

GetCollection (see page 3368)

GetCollection (see page 3389)

ConnectKind (see page 18110)

25.1.5.4.4 TCOMAdminCatalogCollection.Disconnect Method

Terminates the connection to the COM server.

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

Disconnect releases the ICatalogCollection interface that TCOMAdminCatalogCollection (see page 3380) uses to implement its properties and methods. Once you have called Disconnect, you can no longer use the TCOMAdminCatalogCollection (see page 3380) instance, unless you rebind it to another interface using the ConnectTo (see page 3386) method.

See Also

ConnectTo (see page 3386)

25.1.5.4.5 TCOMAdminCatalogCollection.Get_AddEnabled Method

Returns the value of the Get_AddEnabled property.

Pascal

```
function Get_AddEnabled: WordBool;
```

C++

```
__fastcall WordBool Get_AddEnabled();
```

Description

Get_Get_AddEnabled is the read implementation of the Get_AddEnabled property.

See Also

Get_AddEnabled

25.1.5.4.6 TCOMAdminCatalogCollection.Get_Count Method

Returns the number of items in the collection.

Pascal

```
function Get_Count: Integer;
```

C++

```
__fastcall int Get_Count();
```

Description

Get_Get_Count is the read implementation of the Get_Count property.

See Also

Get_Count

25.1.5.4.7 TCOMAdminCatalogCollection.Get_DataStoreMajorVersion Method

Returns the value of the Get_DataStoreMajorVersion property.

Pascal

```
function Get_DataStoreMajorVersion: Integer;
```

C++

```
__fastcall int Get_DataStoreMajorVersion();
```

Description

Get_Get_DataStoreMajorVersion is the read implementation of the Get_DataStoreMajorVersion property.

See Also

Get_DataStoreMajorVersion

25.1.5.4.8 TCOMAdminCatalogCollection.Get_DataStoreMinorVersion Method

Returns the value of the Get_DataStoreMinorVersion property.

Pascal

```
function Get_DataStoreMinorVersion: Integer;
```

C++

```
__fastcall int Get_DataStoreMinorVersion();
```

Description

Get_Get_DataStoreMinorVersion is the read implementation of the Get_DataStoreMinorVersion property.

See Also

Get_DataStoreMinorVersion

25.1.5.4.9 TCOMAdminCatalogCollection.Get_Item Method

Returns a specified item from the collection.

Pascal

```
function Get_Item(lIndex: Integer): TCOMAdminCatalogObject;
```

C++

```
__fastcall TCOMAdminCatalogObject Get_Item(int lIndex);
```

Description

Call Get_Get_Item to access an item in the collection

lIndex is the index of the item. It can have any value from 0 to one less than the value of the Count (see page 3393) property.

See Also

Count (see page 3393)

25.1.5.4.10 TCOMAdminCatalogCollection.Get_Name Method

Returns the value of the Get_Name property

Pascal

```
function Get_Name: OleVariant;
```

C++

```
__fastcall OleVariant Get_Name();
```

Description

GetGet_Name is the read implementation of the Get_Name property.

See Also

Get_Name

25.1.5.4.11 TCOMAdminCatalogCollection.Get_RemoveEnabled Method

Returns the value of the Get_RemoveEnabled property.

Pascal

```
function Get_RemoveEnabled: WordBool;
```

C++

```
__fastcall WordBool Get_RemoveEnabled();
```

Description

Get_Get_RemoveEnabled is the read implementation of the Get_RemoveEnabled property.

See Also

Get_RemoveEnabled

25.1.5.4.12 TCOMAdminCatalogCollection.GetCollection Method

Returns a TCOMAdminCatalogCollectionCollection object for a specified collection.

Pascal

```
function GetCollection(const bstrCollName: WideString; varObjectKey: OleVariant):
TCOMAdminCatalogCollection;
```

C++

```
__fastcall TCOMAdminCatalogCollection GetCollection(const BSTR bstrCollName, OleVariant
varObjectKey);
```

Description

GetCollection retrieves a sub-collection that is contained in this collection.

bstrCollName is the name of the desired collection. The available collections depend on your version of the COMAdmin Library. See the Microsoft documentation for a list of the available collections in your version of the COMAdmin Library.

Key is the Key property of the item to which the collection belongs.

Note: The TCOMAdminCatalogCollectionCollection object that GetCollection returns does not contain data from the catalog for items contained in the collection. Use its Populate (see page 3391) method to read in data for items in the collection.

See Also

Key (see page 3405)

Populate (see page 3391)

25.1.5.4.13 TCOMAdminCatalogCollection.GetUtilInterface Method

Returns the utility interface associated with this collection.

Pascal

```
function GetUtilInterface: IDispatch;
```

C++

```
__fastcall IDispatch GetUtilInterface();
```

Description

Call the GetUtilInterface method to obtain an interface to the utility object associated with the collection. Not every collection has an associated utility object. The following table lists the collections for which GetUtilInterface returns an interface, along with the name of the associated utility object:

Collection name	Utility object
Packages	PackageUtil
ComponentsInPackage	ComponentUtil
RemoteComponents	RemoteComponentUtil
RolesForPackageComponent	RoleAssociationUtil
RolesForPackageComponentInterface	RoleAssociationUtil

25.1.5.4.14 TCOMAdminCatalogCollection.InitServerData Method

Initializes the ServerData property to reflect the type library information of the COM server.

Pascal

```
procedure InitServerData; override;
```

C++

```
virtual __fastcall InitServerData();
```

Description

The InitServerData constructor calls InitServerData to initialize ServerData with information from the COM server's type library.

InitServerData is an abstract a pure virtual method in InitServerData. Each InitServerData descendant implements this method so that the ServerData property reflects its particular COM server.

See Also

ServerData (see page 18111)

Create (see page 18101)

25.1.5.4.15 TCOMAdminCatalogCollection.Populate Method

Fills the collection object with items from the catalog.

Pascal

```
procedure Populate;
```

C++

```
__fastcall Populate();
```

Description

Call Populate to set the items of the collection to those currently saved for this collection in the COM+ catalog. When you first obtain an instance of TCOMAdminCatalogCollection (see page 3380), it does not yet contain any items, and you must call Populate to fill it with the associated items.

After making changes to the collection (such as adding or removing objects or changing object properties) are not saved to the COM+ catalog until you call the SaveChanges (see page 3392) method. Thus, if you call Populate after making changes but before calling SaveChanges (see page 3392), all your changes are rolled back and the collection is restored to the state that is saved in the COM+ catalog.

See Also

PopulateByQuery (see page 3392)

PopulateByKey (see page 3391)

TCOMAdminCatalog (see page 3359)

SaveChanges (see page 3392)

25.1.5.4.16 TCOMAdminCatalogCollection.PopulateByKey Method

Populates the collection with information for the specified objects.

Pascal

```
procedure PopulateByKey(aKeys: PSafeArray);
```

C++

```
__fastcall PopulateByKey(PSafeArray aKeys);
```

Description

Use PopulateByKey to refresh the items in the collection that have a specified key. PopulateByKey reloads all items from the

COM+ catalog that have the specified keys.

aKeys is an array of Key values that represent the Key property for each item to be loaded from the COM+ catalog.

See Also

Populate (see page 3391)

PopulateByQuery (see page 3392)

Key (see page 3405)

25.1.5.4.17 TCOMAdminCatalogCollection.PopulateByQuery Method

Reserved for future use.

Pascal

```
procedure PopulateByQuery(const bstrQueryString: WideString; lQueryType: Integer);
```

C++

```
__fastcall PopulateByQuery(const BSTR bstrQueryString, int lQueryType);
```

Description

PopulateByQuery is not currently implemented.

See Also

Populate (see page 3391)

PopulateByKey (see page 3391)

25.1.5.4.18 TCOMAdminCatalogCollection.Remove Method

Removes an item from the collection

Pascal

```
procedure Remove(lIndex: Integer);
```

C++

```
__fastcall Remove(int lIndex);
```

Description

Remove deletes an item from the collection given its index, and re-indexes the items with higher indices. These changes are not reflected on the COM+ catalog data store until you call SaveChanges (see page 3392).

See Also

RemoveEnabled (see page 3395)

SaveChanges (see page 3392)

25.1.5.4.19 TCOMAdminCatalogCollection.SaveChanges Method

Commits any changes to the collection by writing them to the COM+ catalog.

Pascal

```
function SaveChanges: Integer;
```

C++

```
__fastcall int SaveChanges();
```

Description

Call SaveChanges to save all pending changes made to the collection and the items it contains to the COM+ catalog data store. SaveChanges returns the number of changes written to the COM+ catalog.

See Also

Populate (see page 3391)

25.1.5.5 TCOMAdminCatalogCollection Properties

The properties of the TCOMAdminCatalogCollection class are listed here.

Public Properties

	Name	Description
R	AddEnabled (see page 3393)	Indicates whether the Add (see page 3386) method can add items to the collection.
R	Count (see page 3393)	Indicates the number of items in the collection.
R	DataStoreMajorVersion (see page 3394)	Indicates the major version number of the COMAdmin Library
R	DataStoreMinorVersion (see page 3394)	Indicates the minor version number of the COMAdmin Library
R	DefaultInterface (see page 3395)	Provides access to the default interface for the component.
R	Name (see page 3395)	Indicates the Name of the collection.
R	RemoveEnabled (see page 3395)	Indicates whether the Remove (see page 3392) method can remove items from the collection.

25.1.5.5.1 TCOMAdminCatalogCollection.AddEnabled Property

Indicates whether the Add (see page 3386) method can add items to the collection.

Pascal

```
property AddEnabled: WordBool;
```

C++

```
__property WordBool AddEnabled;
```

Description

Read AddEnabled to determine whether your application can add items to this collection. When AddEnabled is false, the Add (see page 3386) method fails.

See Also

Add (see page 3386)

25.1.5.5.2 TCOMAdminCatalogCollection.Count Property

Indicates the number of items in the collection.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use Count to determine the number of items in the collection. Count acts as an upper bound for the index to the Get_Item (see page 3389) method.

When you first obtain a TCOMAdminCatalogCollection (see page 3380) object, the value of Count is 0. Call the Populate (see page 3391) method to fill the collection with its items to determine the actual number of items in this collection of the COM+ catalog.

To change the value of Count, add items using the Add (see page 3386) method or delete them using the Remove (see page 3392) method.

See Also

Add (see page 3386)

Remove (see page 3392)

Get_Item (see page 3389)

Populate (see page 3391)

25.1.5.5.3 TCOMAdminCatalogCollection.DataStoreMajorVersion Property

Indicates the major version number of the COMAdmin Library

Pascal

```
property DataStoreMajorVersion: Integer;
```

C++

```
__property int DataStoreMajorVersion;
```

Description

Use DataStoreMajorVersion to determine the major version number of the COMAdmin Library. This is the same as the MajorVersion property of the TCOMAdminCatalog (see page 3359) object that contains this collection.

See Also

DataStoreMinorVersion (see page 3394)

MajorVersion (see page 3379)

25.1.5.5.4 TCOMAdminCatalogCollection.DataStoreMinorVersion Property

Indicates the minor version number of the COMAdmin Library

Pascal

```
property DataStoreMinorVersion: Integer;
```

C++

```
__property int DataStoreMinorVersion;
```

Description

Use DataStoreMinorVersion to determine the minor version number of the COMAdmin Library. This is the same as the MinorVersion property of the TCOMAdminCatalog (see page 3359) object that contains this collection.

See Also

DataStoreMajorVersion (see page 3394)

MinorVersion (see page 3380)

25.1.5.5.5 TCOMAdminCatalogCollection.DefaultInterface Property

Provides access to the default interface for the component.

Pascal

```
property DefaultInterface: ICatalogCollection;
```

C++

```
__property ICatalogCollection DefaultInterface;
```

Description

Use DefaultInterface to access the default interface for TCOMAdminCatalogCollection (see page 3380). This is the ICatalogCollection interface, whose properties and methods are surfaced as properties and methods of TCOMAdminCatalogCollection (see page 3380).

25.1.5.5.6 TCOMAdminCatalogCollection.Name Property

Indicates the Name of the collection.

Pascal

```
property Name: OleVariant;
```

C++

```
__property OleVariant Name;
```

Description

Use Name to determine the name of the collection. This is the name by which you can identify the collection, for example, when calling the GetCollection (see page 3389) method to obtain a TCOMAdminCatalogCollection (see page 3380) instance.

See Also

GetCollection (see page 3368)

GetCollection (see page 3389)

25.1.5.5.7 TCOMAdminCatalogCollection.RemoveEnabled Property

Indicates whether the Remove (see page 3392) method can remove items from the collection.

Pascal

```
property RemoveEnabled: WordBool;
```

C++

```
__property WordBool RemoveEnabled;
```

Description

Read RemoveEnabled to determine whether your application can remove items from this collection. When RemoveEnabled is false, the Remove (see page 3392) method fails.

See Also

Remove (see page 3392)

25.1.6 TCOMAdminCatalogObject Class

TCOMAdminCatalogObject represents an item in a collection from the COM+ catalog.

Class Hierarchy

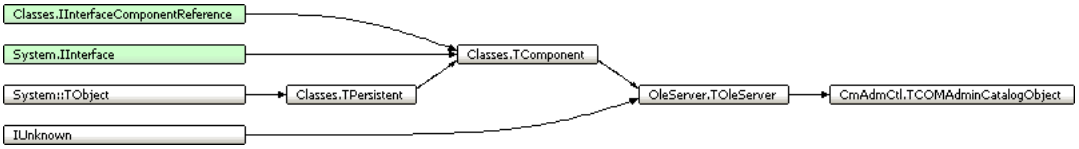

Pascal

```
TCOMAdminCatalogObject = class(TOleServer);
```

C++

```
class TCOMAdminCatalogObject : public TOleServer;
```

File

CmAdmCtl

Description

Use TCOMAdminCatalogObject to work with one of the items in a COM+ catalog collection. Use the Get_Item method of TCOMAdminCatlogCollection to obtain a TCOMAdminCatalogObject instance for a specific item. Each item corresponds to one of the items in a folder that appears in the Component Services administration tool.

TCOMAdminCatalogObject implements the ICatalogObject interface, to let you get and set properties on the items in a catalog collection. Any changes you make to the item are not reflected on the catalog data store until you call SaveChanges for the collection that contains the item.

See Also

TCOMAdminCatalog (see page 3359)

TCOMAdminCatalogCollection (see page 3380)

TCOMAdminCatalogCollection (see page 3380)

TCOMAdminCatalogCollection (see page 3380)

25.1.6.1 TCOMAdminCatalogObject Members

The following tables list the members exposed by TCOMAdminCatalogObject.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_AddRef (see page 18104)	Increments the reference count for the number of interface references dependent on the OLE server.
	_Release (see page 2769)	Called when an application releases a component interface.

	_Release (see page 18104)	Decrements the reference count for the calling interface on a object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ConnectEvents (see page 18105)	Establishes a connection using the EventDispatch (see page 18111) property as an event sink.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DisconnectEvents (see page 18106)	Terminates a connection that was established using the ConnectEvents (see page 18105) method.
	GetAutoConnect (see page 18106)	Returns the value of the AutoConnect (see page 18110) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetConnectKind (see page 18106)	Returns the value of the ConnectKind (see page 18110) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetServer (see page 18107)	Returns an interface for the COM server.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InitServerData (see page 3403)	Initializes the ServerData property to reflect the type library information of the COM server.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	InvokeEvent (see page 18107)	Dispatches an event from the COM server to the appropriate event handler.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 18108)	Connects to the COM server when the Loaded object is first loaded into memory if AutoConnect (see page 18110) is True true.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	QueryInterface (see page 18108)	Returns a reference to a specified interface if the OLE server supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoConnect (see page 18109)	Sets the value of the AutoConnect (see page 18110) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnectKind (see page 18109)	Sets the value of the ConnectKind (see page 18110) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.

	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	EventDispatch (see page 18111)	This is EventDispatch, a member of class TOleServer.
	ServerData (see page 18111)	Encodes information about the COM server to which ServerData connects.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Connect (see page 3401)	Not used in TCOMAdminCatalogObject (see page 3396).
	ConnectTo (see page 3401)	Connects the TCOMAdminCatalogObject (see page 3396) component to an existing ICatalogObject interface.
	Create (see page 3400)	Creates a new instance of TCOMAdminCatalogObject (see page 3396).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 3400)	Frees an instance of TCOMAdminCatalogObject (see page 3396)
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Disconnect (see page 3402)	Terminates the connection to the COM server.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	Get_Key (see page 3402)	Returns the value of the Get_Key property.
	Get_Name (see page 3402)	Returns the value of the Get_Name property.
	Get_Valid (see page 3403)	Returns the value of the Get_Valid property.
	Get_Value (see page 3403)	Returns the value of the Get_Value property.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.

	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsPropertyReadOnly (see page 3404)	Indicates whether a specified property is read-only.
	IsPropertyWriteOnly (see page 3404)	Indicates whether a specified property is write-only.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Set_Value (see page 3404)	Sets the value of the Set_Value property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	DefaultInterface (see page 3405)	Provides access to the default interface for the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Key (see page 3405)	Uniquely identifies the catalog item.
R	Name (see page 3406)	Indicates the Name of the object.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Valid (see page 3406)	Indicates whether all properties of the object have been successfully read from the catalog data store.
	Value (see page 3407)	Represents a the value of a named property of the catalog object.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	AutoConnect (see page 18110)	Specifies whether the component automatically connects to the server.
	ConnectKind (see page 18110)	Indicates the type of process to which the ConnectKind component establishes a connection.
	RemoteMachineName (see page 18111)	Specifies the machine running the COM server when ConnectKind (see page 18110) is ckRemote.
	Tag (see page 2808)	Stores an integer value as part of a component.

25.1.6.2 TCOMAdminCatalogObject.Create Constructor

Creates a new instance of TCOMAdminCatalogObject (see page 3396).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCOMAdminCatalogObject(TComponent * AOwner);
```

Description

Call Create to instantiate TCOMAdminCatalogObject (see page 3396) at runtime. Once you create a TCOMAdminCatalogObject (see page 3396), you must bind it to an ICatalogObject interface before you can use it. Use the ConnectTo (see page 3401) method to bind a TCOMAdminCatalogObject (see page 3396) instance to an ICatalogObject interface.

Note: The TCOMAdminCatalogObject (see page 3396) instance that the catalog object returns from its Get_Item method is already bound to an ICatalogObject interface.

Owner specifies the component (typically a form or data module) that is responsible for freeing the TCOMAdminCatalogObject (see page 3396) instance.

See Also

Owner (see page 2807)

ConnectTo (see page 3401)

Get_Item (see page 3389)

25.1.6.3 TCOMAdminCatalogObject.Destroy Destructor

Frees an instance of TCOMAdminCatalogObject (see page 3396)

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCOMAdminCatalogObject();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TCOMAdminCatalogObject (see page 3396) component reference is not nil before calling Destroy.

Note that freeing a TCOMAdminCatalogObject (see page 3396) instance does not delete the item from its collection. It merely frees the object that lets you work with the item.

See Also

Free (see page 22300)

25.1.6.4 TCOMAdminCatalogObject Methods

The methods of the TCOMAdminCatalogObject class are listed here.

Protected Methods

	Name	Description
	InitServerData (see page 3403)	Initializes the ServerData property to reflect the type library information of the COM server.

Public Methods

	Name	Description
	Connect (see page 3401)	Not used in TCOMAdminCatalogObject (see page 3396).
	ConnectTo (see page 3401)	Connects the TCOMAdminCatalogObject (see page 3396) component to an existing ICatalogObject interface.
	Disconnect (see page 3402)	Terminates the connection to the COM server.
	Get_Key (see page 3402)	Returns the value of the Get_Key property.
	Get_Name (see page 3402)	Returns the value of the Get_Name property.
	Get_Valid (see page 3403)	Returns the value of the Get_Valid property.
	Get_Value (see page 3403)	Returns the value of the Get_Value property.
	IsPropertyReadOnly (see page 3404)	Indicates whether a specified property is read-only.
	IsPropertyWriteOnly (see page 3404)	Indicates whether a specified property is write-only.
	Set_Value (see page 3404)	Sets the value of the Set_Value property.

25.1.6.4.1 TCOMAdminCatalogObject.Connect Method

Not used in TCOMAdminCatalogObject (see page 3396).

Pascal

```
procedure Connect; override;
```

C++

```
virtual __fastcall Connect();
```

Description

Do not use the Connect method. TCOMAdminCatalogObject (see page 3396) objects never directly launch the COM server. Rather, obtain a TCOMAdminCatalogObject (see page 3396) object that is already bound to an ICatalogCollection interface from the parent collection's Get_Item method.

See Also

ConnectTo (see page 3401)

Get_Item (see page 3389)

25.1.6.4.2 TCOMAdminCatalogObject.ConnectTo Method

Connects the TCOMAdminCatalogObject (see page 3396) component to an existing ICatalogObject interface.

Pascal

```
procedure ConnectTo(svrIntf: ICatalogObject);
```

C++

```
__fastcall ConnectTo(ICatalogObject svrIntf);
```

Description

Use ConnectTo when the ConnectKind property is ckAttachToInterface. ConnectTo allows the TCOMAdminCatalogObject (see page 3396) to use an existing ICatalogCollection interface.

Typically, applications do not need to use the ConnectTo method. Rather, they obtain a TCOMAdminCatalogObject (see page 3396) object that is already bound to an ICatalogCollection object by calling the collection object's Get_Item method.

See Also

Create (see page 3400)

Get_Item (see page 3389)

ConnectKind (see page 18110)

25.1.6.4.3 TCOMAdminCatalogObject.Disconnect Method

Terminates the connection to the COM server.

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

Disconnect releases the ICatalogObject interface that TCOMAdminCatalogObject (see page 3396) uses to implement its properties and methods. Once you have called Disconnect, you can no longer use the TCOMAdminCatalogObject (see page 3396) instance, unless you rebind it to another interface using the ConnectTo (see page 3401) method.

See Also

Destroy (see page 3400)

ConnectTo (see page 3401)

25.1.6.4.4 TCOMAdminCatalogObject.Get_Key Method

Returns the value of the Get_Key property.

Pascal

```
function Get_Key: OleVariant;
```

C++

```
__fastcall OleVariant Get_Key();
```

Description

Get_Get_Key is the read implementation of the Get_Key property.

See Also

Get_Key

25.1.6.4.5 TCOMAdminCatalogObject.Get_Name Method

Returns the value of the Get_Name property.

Pascal

```
function Get_Name: OleVariant;
```

C++

```
__fastcall OleVariant Get_Name();
```

Description

Get_Get_Name is the read implementation of the Get_Name property.

See Also

Get_Name

25.1.6.4.6 TCOMAdminCatalogObject.Get_Valid Method

Returns the value of the Get_Valid property.

Pascal

```
function Get_Valid: WordBool;
```

C++

```
__fastcall WordBool Get_Valid();
```

Description

Get_Get_Valid is the read implementation of the Get_Valid property.

See Also

Get_Valid

25.1.6.4.7 TCOMAdminCatalogObject.Get_Value Method

Returns the value of the Get_Value property.

Pascal

```
function Get_Value(const bstrPropName: WideString): OleVariant;
```

C++

```
__fastcall OleVariant Get_Value(const BSTR bstrPropName);
```

Description

Get_Get_Value is the read implementation of the Get_Value property.

bstrPropName is the name of the property whose value you want to read.

See Also

Get_Value

25.1.6.4.8 TCOMAdminCatalogObject.InitServerData Method

Initializes the ServerData property to reflect the type library information of the COM server.

Pascal

```
procedure InitServerData; override;
```

C++

```
virtual __fastcall InitServerData();
```

Description

The InitServerData constructor calls InitServerData to initialize ServerData with information from the COM server's type library.

InitServerData is an abstract a pure virtual method in InitServerData. Each InitServerData descendant implements this method

so that the ServerData property reflects its particular COM server.

See Also

ServerData (see page 18111)

Create (see page 18101)

25.1.6.4.9 TCOMAdminCatalogObject.IsPropertyReadOnly Method

Indicates whether a specified property is read-only.

Pascal

```
function IsPropertyReadOnly(const bstrPropName: WideString): WordBool;
```

C++

```
__fastcall WordBool IsPropertyReadOnly(const BSTR bstrPropName);
```

Description

Call IsPropertyReadOnly to determine whether you can use the Value (see page 3407) property to set a specific property. If IsPropertyReadOnly returns true, you can use Value (see page 3407) to read the property's value but not set it. If IsPropertyReadOnly returns false, you can use Value (see page 3407) to set the property's value as well.

bstrPropName is the name of the property.

See Also

Value (see page 3407)

IsPropertyWriteOnly (see page 3404)

25.1.6.4.10 TCOMAdminCatalogObject.IsPropertyWriteOnly Method

Indicates whether a specified property is write-only.

Pascal

```
function IsPropertyWriteOnly(const bstrPropName: WideString): WordBool;
```

C++

```
__fastcall WordBool IsPropertyWriteOnly(const BSTR bstrPropName);
```

Description

Call IsPropertyWriteOnly to determine whether you can read the value of a specific property using the Value (see page 3407) property. If IsPropertyWriteOnly returns true, you can use Value (see page 3407) to set the property's value but not read it. If IsPropertyWriteOnly returns false, you can use Value (see page 3407) to read the property's value as well.

bstrPropName is the name of the property.

See Also

IsPropertyReadOnly (see page 3404)

Value (see page 3407)

25.1.6.4.11 TCOMAdminCatalogObject.Set_Value Method

Sets the value of the Set_Value property.

Pascal

```
procedure Set_Value(const bstrPropName: WideString; retval: OleVariant);
```

C++

```
__fastcall Set_Value(const BSTR bstrPropName, OleVariant retval);
```

Description

Set_Set_Value is the write implementation of the Set_Value property.

bstrPropName is the name of the property to set.

retVal is the new value for that property.

See Also

Set_Value

IsPropertyReadOnly (see page 3404)

25.1.6.5 TCOMAdminCatalogObject Properties

The properties of the TCOMAdminCatalogObject class are listed here.

Public Properties

	Name	Description
R	DefaultInterface (see page 3405)	Provides access to the default interface for the component.
R	Key (see page 3405)	Uniquely identifies the catalog item.
R	Name (see page 3406)	Indicates the Name of the object.
R	Valid (see page 3406)	Indicates whether all properties of the object have been successfully read from the catalog data store.
	Value (see page 3407)	Represents a the value of a named property of the catalog object.

25.1.6.5.1 TCOMAdminCatalogObject.DefaultInterface Property

Provides access to the default interface for the component.

Pascal

```
property DefaultInterface: ICatalogObject;
```

C++

```
__property ICatalogObject DefaultInterface;
```

Description

Use DefaultInterface to access the default interface for TCOMAdminCatalogObject (see page 3396). This is the ICatalogObject interface, whose properties and methods are surfaced as properties and methods of TCOMAdminCatalogObject (see page 3396).

25.1.6.5.2 TCOMAdminCatalogObject.Key Property

Uniquely identifies the catalog item.

Pascal

```
property Key: OleVariant;
```

C++

```
__property OleVariant Key;
```

Description

The Key property serves as the primary identifier for an object. In some cases, Key is a GUID, such as CLSID for a component; in some cases it is the item name, as with roles. To determine the meaning of the Key property, check the Microsoft documentation for the specific collection to which this item belongs.

Note: If you add a new object to a collection and save it with the Key property of an existing object, you overwrite the existing object.

See Also

Name (see page 3406)

25.1.6.5.3 TCOMAdminCatalogObject.Name Property

Indicates the Name of the object.

Pascal

```
property Name: OleVariant;
```

C++

```
__property OleVariant Name;
```

Description

Use Name to determine the name of the item.

Note: Although every item has a name, the primary identifier for collection items is the Key (see page 3405) property.

See Also

Key (see page 3405)

25.1.6.5.4 TCOMAdminCatalogObject.Valid Property

Indicates whether all properties of the object have been successfully read from the catalog data store.

Pascal

```
property Valid: WordBool;
```

C++

```
__property WordBool Valid;
```

Description

Read Valid to determine whether the object has been initialized to reflect the properties stored in the COM+ catalog for this item. Valid is true if the properties were successfully read from the catalog, false if they could not be read from the catalog. For example, Valid is false for a new item that has never been written to the COM+ catalog.

See Also

Value (see page 3407)

TCOMAdminCatalog (see page 3359)

Populate (see page 3391)

SaveChanges (see page 3392)

25.1.6.5.5 TCOMAdminCatalogObject.Value Property

Represents a the value of a named property of the catalog object.

Pascal

```
property Value [const bstrPropName: WideString]: OleVariant;
```

C++

```
__property OleVariant Value[BSTR const bstrPropName];
```

Description

Use Value to get or set a named property of this item.

bstrPropName is the name of the property to get or set. Valid (see page 3406) property names depend on the type of the item (the collection to which the item belongs). To determine the available properties for this object, check the Microsoft documentation for the specific collection to which this item belongs.

Tip: Use the IsPropertyReadOnly (see page 3404) and IsPropertyWriteOnly (see page 3404) methods to determine whether you can get or set a specific property.

See Also

IsPropertyReadOnly (see page 3404)

IsPropertyWriteOnly (see page 3404)

26 ComApp Namespace

This is namespace ComApp.

Topics

Name	Description
Classes (see page 3410)	The following table lists classes in this documentation.

Classes

	Name	Description
	TWebAppAutoObjectFactory (see page 3410)	

26.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TWebAppAutoObjectFactory (see page 3410)	

26.1.1 TWebAppAutoObjectFactory Class

Class Hierarchy

Pascal

```
TWebAppAutoObjectFactory = class(TWebAppSockObjectFactory);
```

C++

```
class TWebAppAutoObjectFactory : public TWebAppSockObjectFactory;
```

File

ComApp

See Also

ComClassManager

CreateComObject

CreateOleObject

TComObject (see page 5553)

26.1.1.1 TWebAppAutoObjectFactory Members

The following tables list the members exposed by TWebAppAutoObjectFactory.

Public Methods

	Name	Description
	Create (see page 3410)	Instantiates a class factory object.

26.1.1.2 TWebAppAutoObjectFactory.Create Constructor

Instantiates a class factory object.

Pascal

```
constructor Create(AClassID: TGuid; const AClassName: string; const ADescription: string);
```

C++

```
__fastcall TWebAppAutoObjectFactory(TGuid AClassID, const AnsiString AClassName, const
AnsiString ADescription);
```

Description

Create TComObjectFactory instantiates a class factory, usually when the server is loaded. Class factories should be instantiated in the initialization section of the unit that includes their associated server, so that they are automatically available when the server is loaded.

Create TComObjectFactory allocates memory for a class factory object, and adds the object to the class factory list managed by the ComClassManager. Then Create TComObjectFactory initializes the following properties with values passed to it:

Sets ClassID to the ClassID parameter.

Sets ClassName to the ClassName parameter.

Sets Description to the Description parameter.

27 ComCtrls Namespace

This is namespace ComCtrls.

Topics

Name	Description
Classes (see page 3419)	The following table lists classes in this documentation.
Functions (see page 5435)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 5438)	The following table lists structs, records, enums in this documentation.
Types (see page 5477)	The following table lists types in this documentation.
Constants (see page 5521)	The following table lists constants in this documentation.

Classes

	Name	Description
	ECommonCalendarError (see page 3420)	ECommonCalendarError is the exception class for invalid entries in a TCommonCalendar (see page 3558) component.
	EDateTimeError (see page 3421)	EDateTimeError is the exception class for invalid date and time entries.
	EMonthCalError (see page 3421)	EMonthCalError is the exception class for invalid entries in a TMonthCalendar (see page 4397) component.
	ETreeViewError (see page 3422)	ETreeViewError is the exception class for tree view errors.
	TAnimate (see page 3422)	TAnimate is an animation control.
	TComboBoxEx (see page 3463)	TComboBoxEx represents a combo box that supports extended combo box features such as images on the list entries.
	TComboBoxExActionLink (see page 3531)	This is class ComCtrls.TComboBoxExActionLink.
	TComboBoxExStrings (see page 3535)	This is class ComCtrls.TComboBoxExStrings.
	TComboExItem (see page 3546)	TComboExItem represents an item in an extended combo box.
	TComboExItems (see page 3552)	TComboExItems manages a collection of TComboExItem (see page 3546) objects.
	TCommonCalendar (see page 3558)	TCommonCalendar is the base class for the controls TDateTimePicker (see page 4049) and TMonthCalendar (see page 4397).
	TConversion (see page 3582)	TConversion converts rich text data from one encoding format to another.
	TCoolBand (see page 3584)	TCoolBand represents a band within a TCoolBar (see page 3603).
	TCoolBands (see page 3596)	TCoolBands is a container for TCoolBand (see page 3584) objects.
	TCoolBar (see page 3603)	TCoolBar displays a collection of windowed controls within movable, resizable bands.
	TCustomComboBoxEx (see page 3678)	TCustomComboBoxEx is the base class for combo box components that support extended combo box features such as images on the list entries.
	TCustomHeaderControl (see page 3702)	TCustomHeaderControl represents a spreadsheet-style header.

	TCustomHotKey (see page 3739)	TCustomHotKey is the base class for objects that wrap the Windows accelerator key Common Control.
	TCustomListView (see page 3757)	TCustomListView is the base type for all list view components, including TListView (see page 4285).
	TCustomRichEdit (see page 3854)	TCustomRichEdit is the base type for rich edit controls such as TRichEdit (see page 4685).
	TCustomStatusBar (see page 3892)	TCustomStatusBar is the base class for objects that represent status bars.
	TCustomTabControl (see page 3921)	TCustomTabControl is the base type for tab control components such as TTabControl (see page 4841) and TPageControl (see page 4459).
	TCustomTreeView (see page 3957)	TCustomTreeView is the base type for tree-view components such as TTreeView (see page 5293).
	TCustomUpDown (see page 4027)	TCustomUpDown is the base type for up-down size controls such as TUpDown (see page 5380).
	TDateTimePicker (see page 4049)	TDateTimePicker displays a list box for entering dates or times.
	THeaderControl (see page 4118)	THeaderControl is a container for THeaderSection (see page 4173) objects. It provides a set of resizable column headers.
	THeaderSection (see page 4173)	THeaderSection represents a section of a header control.
	THeaderSections (see page 4185)	THeaderSections is a container for THeaderSection (see page 4173) objects.
	THotKey (see page 4190)	THotKey is a wrapper for a Windows Common Control hot key.
	TIconOptions (see page 4233)	TIconOptions describes how the icons in a list view should be arranged.
	TListColumn (see page 4236)	TListColumn represents a column in a report-style list view.
	TListColumns (see page 4245)	TListColumns is a collection of TListColumn (see page 4236) objects.
	TListItem (see page 4250)	TListItem is an individual item of a TListView (see page 4285) control.
	TListItems (see page 4270)	TListItems maintains the collection of items that appear in a list view control.
	TListItemsEnumerator (see page 4283)	This is class ComCtrls.TListItemsEnumerator.
	TListView (see page 4285)	TListView displays a list of items in various ways.
	TListViewActionLink (see page 4388)	This is class ComCtrls.TListViewActionLink.
	TMonthCalColors (see page 4392)	TMonthCalColors defines a color scheme for TDateTimePicker (see page 4049) or TMonthCalendar (see page 4397)'s drop-down calendar.
	TMonthCalendar (see page 4397)	TMonthCalendar is stand-alone calendar in which a user can select a date or range of dates.
	TPageControl (see page 4459)	TPageControl is a set of pages used to make a multiple page dialog box.
	TPageScroller (see page 4548)	TPageScroller defines a display area for a narrow window, such as a toolbar.
	TParaAttributes (see page 4606)	TParaAttributes represents paragraph formatting properties for a rich edit control.
	TProgressBar (see page 4615)	TProgressBar displays a simple progress bar.
	TRichEdit (see page 4685)	TRichEdit is a wrapper for a Windows rich text edit control.
	TStatusBar (see page 4769)	TStatusBar represents a status bar.
	TStatusPanel (see page 4825)	TStatusPanel represents a single panel of a status bar.
	TStatusPanels (see page 4834)	TStatusPanels is a container for TStatusPanel (see page 4825) objects.
	TTabControl (see page 4841)	TTabControl is a tab set that has the appearance of notebook dividers.
	TTabSheet (see page 4919)	TTabSheet is an individual page in a TPageControl (see page 4459) object.
	TTextAttributes (see page 4982)	TTextAttributes represents the font characteristics of a section of text in a rich edit control.
	TToolBar (see page 4994)	TToolBar manages tool buttons and other controls, arranging them in rows and automatically adjusting their sizes and positions.
	TToolBarDockObject (see page 5102)	This is class ComCtrls.TToolBarDockObject.
	TToolBarEnumerator (see page 5105)	This is class ComCtrls.TToolBarEnumerator.
	TToolButton (see page 5107)	TToolButton is a button control in a TToolBar (see page 4994) object.
	TToolButtonActionLink (see page 5166)	TToolButtonActionLink is an action link designed for tool button control clients.
	TTrackBar (see page 5172)	TTrackBar is a wrapper for a track bar control.
	TTreeNode (see page 5230)	TTreeNode describes an individual node in a tree view control.
	TTreeNodes (see page 5266)	TTreeNodes maintains a list of tree nodes in a tree view control.
	TTreeNodesEnumerator (see page 5291)	This is class ComCtrls.TTreeNodesEnumerator.
	TTreeView (see page 5293)	TTreeView represents a window that displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files and directories on a disk.
	TUpDown (see page 5380)	TUpDown is a wrapper for a Windows up-down control.
	TWorkArea (see page 5425)	TWorkArea represents a rectangular subregion of a list view's client area.

	TWorkAreas (see page 5429)	TWorkAreas is a collection of TWorkArea (see page 5425) objects.

Constants

Name	Description
CN_BANDCHANGE (see page 5521)	This is constant ComCtrls.CN_BANDCHANGE.
CN_DROPDOWNCLOSED (see page 5521)	This is constant ComCtrls.CN_DROPDOWNCLOSED.
ColumnHeaderWidth (see page 5522)	This is constant ComCtrls.ColumnHeaderWidth.
ColumnTextWidth (see page 5522)	This is constant ComCtrls.ColumnTextWidth.
ComCtlVersionIE3 (see page 5522)	This is constant ComCtrls.ComCtlVersionIE3.
ComCtlVersionIE4 (see page 5523)	This is constant ComCtrls.ComCtlVersionIE4.
ComCtlVersionIE401 (see page 5523)	This is constant ComCtrls.ComCtlVersionIE401.
ComCtlVersionIE5 (see page 5523)	This is constant ComCtrls.ComCtlVersionIE5.
ComCtlVersionIE501 (see page 5524)	This is constant ComCtrls.ComCtlVersionIE501.
ComCtlVersionIE6 (see page 5524)	This is constant ComCtrls.ComCtlVersionIE6.

Functions

	Name	Description
	CheckCommonControl (see page 5435)	This is function ComCtrls.CheckCommonControl.
	CheckToolMenuDropdown (see page 5435)	Displays a tool button's dropdown menu.
	GetComCtlVersion (see page 5436)	Indicates which version of ComCtl32.dll is installed.
	InitCommonControl (see page 5437)	This is function ComCtrls.InitCommonControl.

Structs, Records, Enums

	Name	Description
	TAddMode (see page 5439)	This is record ComCtrls.TAddMode.
	TAttributeType (see page 5440)	This is record ComCtrls.TAttributeType.
	TAutoCompleteOption (see page 5440)	TAutoCompleteOption and TAutoCompleteOptions (see page 5480) specify a set of autocomplete options.
	TCalDayOfWeek (see page 5441)	TCalDayOfWeek represents a day of the week.
	TComboBoxExStyle (see page 5442)	TComboBoxExStyle is the type for the Style property of extended combo box controls.
	TComboBoxExStyleEx (see page 5442)	TComboBoxExStyleEx and TComboBoxExStyles (see page 5481) configure the behavior of extended combo box controls.
	TCommonAVI (see page 5443)	This is record ComCtrls.TCommonAVI.
	TConsistentAttribute (see page 5444)	This is record ComCtrls.TConsistentAttribute.
	TConversionFormat (see page 5444)	This is record ComCtrls.TConversionFormat.
	TCoolBandMaximize (see page 5445)	TCoolBandMaximize specifies the conditions under which a TCoolBand (see page 3584) is maximized.
	TCustomDrawStage (see page 5445)	TCustomDrawStage specifies a stage in the custom drawing process.
	TCustomDrawTarget (see page 5446)	TCustomDrawTarget specifies the target of a custom drawing event.
	TDateTimeKind (see page 5447)	This is record ComCtrls.TDateTimeKind.
	TDisplayCode (see page 5447)	This is record ComCtrls.TDisplayCode.
	TDTCalAlignment (see page 5447)	This is record ComCtrls.TDTCalAlignment.
	TDTDateFormat (see page 5448)	This is record ComCtrls.TDTDateFormat.
	TDTDateMode (see page 5448)	This is record ComCtrls.TDTDateMode.
	THeaderSectionStyle (see page 5449)	This is record ComCtrls.THeaderSectionStyle.
	THeaderStyle (see page 5449)	THeaderStyle specifies the style of a header control.
	THitTest (see page 5450)	THitTest and THitTests (see page 5488) are used in the GetHitTestInfoAt method.
	THKInvalidKey (see page 5451)	This is record ComCtrls.THKInvalidKey.
	THKModifier (see page 5451)	This is record ComCtrls.THKModifier.
	TIconArrangement (see page 5452)	TIconArrangement indicates how the icons for items are arranged.
	TItemChange (see page 5452)	TItemChange describes the type of change a list item undergoes.
	TItemFind (see page 5453)	TItemFind identifies the type of search in an OnDataFind event handler.
	TItemRequests (see page 5454)	This is record ComCtrls.TItemRequests.

	TItemState (see page 5454)	TItemState and TItemStates (see page 5489) specify the state of an item in a list view control.
	TListArrangement (see page 5455)	This is record ComCtrls.TListArrangement.
	TListHotTrackStyle (see page 5456)	TListHotTrackStyle and TListHotTrackStyles (see page 5489) specify hot tracking styles for a list view control.
	TMultiSelectStyles (see page 5456)	This is record ComCtrls.TMultiSelectStyles.
	TNodeAttachMode (see page 5457)	TNodeAttachMode identifies a new or changed tree node relationship.
	TNodeCache (see page 5457)	TNodeCache is the type for a cached tree-view node.
	TNodeDataInfo (see page 5458)	This is record ComCtrls.TNodeDataInfo.
	TNodeInfo (see page 5458)	This is record ComCtrls.TNodeInfo.
	TNodeState (see page 5459)	This is record ComCtrls.TNodeState.
	TNumberingStyle (see page 5459)	This is record ComCtrls.TNumberingStyle.
	TPageScrollerButton (see page 5460)	TPageScrollerButton identifies a button on a page scroller.
	TPageScrollerButtonState (see page 5460)	TPageScrollerButtonState indicates the state of a page scroller button.
	TPageScrollerOrientation (see page 5461)	TPageScrollerOrientation indicates the orientation of a page scroller control.
	TPositionToolTip (see page 5462)	Specifies the position of the tool tip.
	TProgressBarOrientation (see page 5462)	TProgressBarOrientation specifies the orientation of a progress bar.
	TRichEditStreamInfo (see page 5463)	This is record ComCtrls.TRichEditStreamInfo.
	TSearchDirection (see page 5463)	Indicates the direction in which to search for the nearest item to a specified item or point.
	TSearchType (see page 5464)	Specifies search options.
	TSectionTrackState (see page 5464)	This is record ComCtrls.TSectionTrackState.
	TSortType (see page 5465)	TSortType indicates the criteria to use when sorting items.
	TStatusPanelBevel (see page 5466)	TStatusPanelBevel indicates the type of bevel around a panel in a status bar.
	TStatusPanelStyle (see page 5466)	TStatusPanelStyle indicates the kind of contents in a status bar panel.
	TTabPosition (see page 5467)	This is record ComCtrls.TTabPosition.
	TTabStyle (see page 5467)	TTabStyle indicates whether a TTabSet control is owner-drawn.
	TTBDrawingStyle (see page 5468)	Specifies the drawing style for the tool bar.
	TTickMark (see page 5468)	TTickMark indicates the location of the tick marks on a track bar.
	TTickStyle (see page 5469)	TTickStyle indicates how a track bar determines the increments at which tick marks appear.
	TToolButtonState (see page 5470)	This is record ComCtrls.TToolButtonState.
	TToolButtonStyle (see page 5470)	TToolButtonStyle indicates the type of control a TToolButton (see page 5107) object represents.
	TTrackBarOrientation (see page 5471)	TTrackBarOrientation indicates whether a trackbar is horizontal or vertical.
	TUDAlignButton (see page 5471)	This is record ComCtrls.TUDAlignButton.
	TUDBtnType (see page 5472)	This is record ComCtrls.TUDBtnType.
	TUDOrientation (see page 5472)	This is record ComCtrls.TUDOrientation.
	TUpDownDirection (see page 5473)	TUpDownDirection indicates the direction in which the position of an updown control changes.
	TViewStyle (see page 5473)	TViewStyle indicates how a list view control displays its items.

Types

Name	Description
PConversionFormat (see page 5479)	This is type ComCtrls.PConversionFormat.
PNodeCache (see page 5479)	PNodeCache is a pointer to a TNodeCache (see page 5457) object.
PNodeInfo (see page 5480)	This is type ComCtrls.PNodeInfo.
PRichEditStreamInfo (see page 5480)	This is type ComCtrls.PRichEditStreamInfo.
TAutoCompleteOptions (see page 5480)	TAutoCompleteOptions specify options that control autocompletion.
TComboBoxExStyles (see page 5481)	TComboBoxExStyles is the type for the Style property of extended combo box controls.
TComboExItemClass (see page 5481)	This is type ComCtrls.TComboExItemClass.
TComboExItemsClass (see page 5482)	This is type ComCtrls.TComboExItemsClass.
TConsistentAttributes (see page 5482)	This is type ComCtrls.TConsistentAttributes.
TConversionClass (see page 5482)	This is type ComCtrls.TConversionClass.
TCustomDrawPanelEvent (see page 5483)	TCustomDrawPanelEvent is the type of the OnDrawPanel event handler.
TCustomDrawSectionEvent (see page 5483)	This is type ComCtrls.TCustomDrawSectionEvent.

TCustomDrawState (see page 5483)	TCustomDrawState specifies a custom drawing state.
TCustomHCCreateSectionClassEvent (see page 5484)	This is type ComCtrls.TCustomHCCreateSectionClassEvent.
TCustomSectionNotifyEvent (see page 5484)	This is type ComCtrls.TCustomSectionNotifyEvent.
TCustomSectionTrackEvent (see page 5485)	This is type ComCtrls.TCustomSectionTrackEvent.
TDateTimeColors (see page 5485)	TDateTimeColors specifies colors used in a calendar control.
TDrawPanelEvent (see page 5485)	This is type ComCtrls.TDrawPanelEvent.
TDrawSectionEvent (see page 5486)	This is type ComCtrls.TDrawSectionEvent.
TDrawTabEvent (see page 5486)	TDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet control.
TDTParseInputEvent (see page 5487)	This is type ComCtrls.TDTParseInputEvent.
THCCreateSectionClassEvent (see page 5487)	This is type ComCtrls.THCCreateSectionClassEvent.
THeaderSectionClass (see page 5487)	This is type ComCtrls.THeaderSectionClass.
THitTests (see page 5488)	This is type ComCtrls.THitTests.
THKInvalidKeys (see page 5488)	This is type ComCtrls.THKInvalidKeys.
THKModifiers (see page 5488)	This is type ComCtrls.THKModifiers.
TItemRequest (see page 5489)	This is type ComCtrls.TItemRequest.
TItemStates (see page 5489)	TItemState (see page 5454) and TItemStates specify the state of an item in a list view control.
TListHotTrackStyles (see page 5489)	TListHotTrackStyle (see page 5456) and TListHotTrackStyles specify hot tracking styles for a list view control.
TListItemClass (see page 5490)	This is type ComCtrls.TListItemClass.
TLVAdvancedCustomDrawEvent (see page 5490)	This is type ComCtrls.TLVAdvancedCustomDrawEvent.
TLVAdvancedCustomDrawItemEvent (see page 5491)	This is type ComCtrls.TLVAdvancedCustomDrawItemEvent.
TLVAdvancedCustomDrawSubItemEvent (see page 5491)	This is type ComCtrls.TLVAdvancedCustomDrawSubItemEvent.
TLVChangeEvent (see page 5491)	This is type ComCtrls.TLVChangeEvent.
TLVChangingEvent (see page 5492)	This is type ComCtrls.TLVChangingEvent.
TLVColumnClickEvent (see page 5492)	This is type ComCtrls.TLVColumnClickEvent.
TLVColumnRClickEvent (see page 5492)	This is type ComCtrls.TLVColumnRClickEvent.
TLVCompareEvent (see page 5493)	This is type ComCtrls.TLVCompareEvent.
TLVCreateItemClassEvent (see page 5493)	This is type ComCtrls.TLVCreateItemClassEvent.
TLVCustomDrawEvent (see page 5493)	This is type ComCtrls.TLVCustomDrawEvent.
TLVCustomDrawItemEvent (see page 5494)	This is type ComCtrls.TLVCustomDrawItemEvent.
TLVCustomDrawSubItemEvent (see page 5494)	This is type ComCtrls.TLVCustomDrawSubItemEvent.
TLVDeletedEvent (see page 5494)	TLVDeletedEvent is the type of a list view's OnDeletion event handler.
TLVDrawItemEvent (see page 5495)	TLVDrawItemEvent is the type of the list view OnCustomDrawItem and OnDrawItem event handlers.
TLVEditedEvent (see page 5495)	This is type ComCtrls.TLVEditedEvent.
TLVEditingEvent (see page 5496)	This is type ComCtrls.TLVEditingEvent.
TLVInfoTipEvent (see page 5496)	This is type ComCtrls.TLVInfoTipEvent.
TLVNotifyEvent (see page 5496)	This is type ComCtrls.TLVNotifyEvent.
TLVOwnerDataEvent (see page 5497)	TLVOwnerDataEvent is the type of a list view's OnData event handler.
TLVOwnerDataFindEvent (see page 5497)	TLVOwnerDataFindEvent is the type of a list view's OnDataFind event handler.
TLVOwnerDataHintEvent (see page 5498)	TLVOwnerDataHintEvent is the type of a list view's OnDataHint event handler.
TLVOwnerDataStateChangeEvent (see page 5498)	TLVOwnerDataStateChangeEvent is the type of a list view's OnDataStateChange event handler.
TLVSelectItemEvent (see page 5499)	This is type ComCtrls.TLVSelectItemEvent.
TLVSubItemImageEvent (see page 5499)	This is type ComCtrls.TLVSubItemImageEvent.
TMultiSelectStyle (see page 5499)	This is type ComCtrls.TMultiSelectStyle.
TOnGetMonthInfoEvent (see page 5500)	TOnGetMonthInfoEvent is the type of the TCommonCalendar.OnGetMonthInfo event handler.
TOwnerDrawState (see page 5500)	Indicates state information that can influence how an item is drawn.
TPageScrollEvent (see page 5501)	TPageScrollEvent is the type of a page scroller's OnScroll event handler.
TProgressRange (see page 5502)	This is type ComCtrls.TProgressRange.
TRichEditProtectChange (see page 5502)	This is type ComCtrls.TRichEditProtectChange.
TRichEditResizeEvent (see page 5502)	This is type ComCtrls.TRichEditResizeEvent.
TRichEditSaveClipboard (see page 5503)	This is type ComCtrls.TRichEditSaveClipboard.

TSBCreatePanelClassEvent (see page 5503)	TSBCreatePanelClassEvent is the type of the OnCreatePanelClass event handler.
TSearchTypes (see page 5503)	Set of TSearchType (see page 5464).
TSectionDragEvent (see page 5505)	TSectionDragEvent is the type for event handlers that respond when the user drags a section in a header control.
TSectionNotifyEvent (see page 5506)	This is type ComCtrls.TSectionNotifyEvent.
TSectionTrackEvent (see page 5506)	This is type ComCtrls.TSectionTrackEvent.
TStatusPanelClass (see page 5507)	TStatusPanelClass defines the metaclass for TStatusPanel (see page 4825).
TTabChangingEvent (see page 5507)	This is type ComCtrls.TTabChangingEvent.
TTabGetImageEvent (see page 5507)	This is type ComCtrls.TTabGetImageEvent.
TTBAdvancedCustomDrawBtnEvent (see page 5508)	This is type ComCtrls.TTBAdvancedCustomDrawBtnEvent.
TTBAdvancedCustomDrawEvent (see page 5508)	This is type ComCtrls.TTBAdvancedCustomDrawEvent.
TTBButtonEvent (see page 5508)	TTBButtonEvent is the type of event handlers that respond when the user adds or deletes a tool button in the toolbar customize dialog.
TTBCustomDrawBtnEvent (see page 5509)	This is type ComCtrls.TTBCustomDrawBtnEvent.
TTBCustomDrawEvent (see page 5509)	This is type ComCtrls.TTBCustomDrawEvent.
TTBCustomDrawFlags (see page 5509)	TTBCustomDrawFlags is the type of the OnAdvancedCustomDrawButton event handler.
TTBCustomizeQueryEvent (see page 5510)	TTBCustomizeQueryEvent is the type of event handlers that allow a toolbar to decide whether to allow a user customization.
TTBGradientDrawingOptions (see page 5511)	Specifies the style of the gradient.
TTBNewButtonEvent (see page 5511)	This is type ComCtrls.TTBNewButtonEvent.
TToolButtonActionLinkClass (see page 5511)	This is type ComCtrls.TToolButtonActionLinkClass.
TTreeNodeClass (see page 5512)	This is type ComCtrls.TTreeNodeClass.
TTVAdvancedCustomDrawEvent (see page 5512)	Used in the TCustomTreeView.OnAdvancedCustomDraw event.
TTVAdvancedCustomDrawItemEvent (see page 5513)	Used in the TCustomTreeView.OnAdvancedCustomDrawItem event.
TTVChangedEvent (see page 5513)	Used in the TCustomTreeView.OnChange event.
TTVChangingEvent (see page 5514)	Used in the TCustomTreeView.OnChanging event.
TTVCollapsingEvent (see page 5514)	Used in the TCustomTreeView.OnCollapsing event.
TTVCompareEvent (see page 5515)	Used in the TCustomTreeView.OnCompare event.
TTVCreateNodeClassEvent (see page 5516)	This is type ComCtrls.TTVCreateNodeClassEvent.
TTVCustomDrawEvent (see page 5516)	Used in the TCustomTreeView.OnCustomDraw event.
TTVCustomDrawItemEvent (see page 5516)	Used in the TCustomTreeView.OnCustomDrawItem event.
TTVEditedEvent (see page 5517)	Used in the TCustomTreeView.OnEdited event.
TTVEditingEvent (see page 5518)	Used in the TCustomTreeView.OnEditing event.
TTVExpandedEvent (see page 5518)	Represents the event handler for events that notify a tree view of changes to one of its nodes.
TTVExpandingEvent (see page 5519)	Used in the TCustomTreeView.OnExpanding event.
TUDChangingEvent (see page 5519)	This is type ComCtrls.TUDChangingEvent.
TUDChangingEventEx (see page 5520)	This is type ComCtrls.TUDChangingEventEx.
TUDClickEvent (see page 5520)	This is type ComCtrls.TUDClickEvent.
TWidth (see page 5520)	This is type ComCtrls.TWidth.

27.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ECommonCalendarError (see page 3420)	ECommonCalendarError is the exception class for invalid entries in a TCommonCalendar (see page 3558) component.
	EDateTimeError (see page 3421)	EDateTimeError is the exception class for invalid date and time entries.
	EMonthCalError (see page 3421)	EMonthCalError is the exception class for invalid entries in a TMonthCalendar (see page 4397) component.
	ETreeViewError (see page 3422)	ETreeViewError is the exception class for tree view errors.
	TAnimate (see page 3422)	TAnimate is an animation control.
	TComboBoxEx (see page 3463)	TComboBoxEx represents a combo box that supports extended combo box features such as images on the list entries.
	TComboBoxExActionLink (see page 3531)	This is class ComCtrls.TComboBoxExActionLink.
	TComboBoxExStrings (see page 3535)	This is class ComCtrls.TComboBoxExStrings.
	TComboExItem (see page 3546)	TComboExItem represents an item in an extended combo box.
	TComboExItems (see page 3552)	TComboExItems manages a collection of TComboExItem (see page 3546) objects.
	TCommonCalendar (see page 3558)	TCommonCalendar is the base class for the controls TDateTimePicker (see page 4049) and TMonthCalendar (see page 4397).
	TConversion (see page 3582)	TConversion converts rich text data from one encoding format to another.
	TCoolBand (see page 3584)	TCoolBand represents a band within a TCoolBar (see page 3603).
	TCoolBands (see page 3596)	TCoolBands is a container for TCoolBand (see page 3584) objects.
	TCoolBar (see page 3603)	TCoolBar displays a collection of windowed controls within movable, resizable bands.
	TCustomComboBoxEx (see page 3678)	TCustomComboBoxEx is the base class for combo box components that support extended combo box features such as images on the list entries.
	TCustomHeaderControl (see page 3702)	TCustomHeaderControl represents a spreadsheet-style header.
	TCustomHotKey (see page 3739)	TCustomHotKey is the base class for objects that wrap the Windows accelerator key Common Control.
	TCustomListView (see page 3757)	TCustomListView is the base type for all list view components, including TListView (see page 4285).
	TCustomRichEdit (see page 3854)	TCustomRichEdit is the base type for rich edit controls such as TRichEdit (see page 4685).
	TCustomStatusBar (see page 3892)	TCustomStatusBar is the base class for objects that represent status bars.
	TCustomTabControl (see page 3921)	TCustomTabControl is the base type for tab control components such as TTabControl (see page 4841) and TPageControl (see page 4459).
	TCustomTreeView (see page 3957)	TCustomTreeView is the base type for tree-view components such as TTreeView (see page 5293).
	TCustomUpDown (see page 4027)	TCustomUpDown is the base type for up-down size controls such as TUpDown (see page 5380).
	TDateTimePicker (see page 4049)	TDateTimePicker displays a list box for entering dates or times.
	THeaderControl (see page 4118)	THeaderControl is a container for THeaderSection (see page 4173) objects. It provides a set of resizable column headers.
	THeaderSection (see page 4173)	THeaderSection represents a section of a header control.
	THeaderSections (see page 4185)	THeaderSections is a container for THeaderSection (see page 4173) objects.
	THotKey (see page 4190)	THotKey is a wrapper for a Windows Common Control hot key.
	TIconOptions (see page 4233)	TIconOptions describes how the icons in a list view should be arranged.
	TListColumn (see page 4236)	TListColumn represents a column in a report-style list view.
	TListColumns (see page 4245)	TListColumns is a collection of TListColumn (see page 4236) objects.
	TListItem (see page 4250)	TListItem is an individual item of a TListView (see page 4285) control.
	TListItems (see page 4270)	TListItems maintains the collection of items that appear in a list view control.
	TListItemsEnumerator (see page 4283)	This is class ComCtrls.TListItemsEnumerator.

	TListView (see page 4285)	TListView displays a list of items in various ways.
	TListViewActionLink (see page 4388)	This is class ComCtrls.TListViewActionLink.
	TMonthCalColors (see page 4392)	TMonthCalColors defines a color scheme for TDateTimePicker (see page 4049) or TMonthCalendar (see page 4397)'s drop-down calendar.
	TMonthCalendar (see page 4397)	TMonthCalendar is stand-alone calendar in which a user can select a date or range of dates.
	TPageControl (see page 4459)	TPageControl is a set of pages used to make a multiple page dialog box.
	TPageScroller (see page 4548)	TPageScroller defines a display area for a narrow window, such as a toolbar.
	TParaAttributes (see page 4606)	TParaAttributes represents paragraph formatting properties for a rich edit control.
	TProgressBar (see page 4615)	TProgressBar displays a simple progress bar.
	TRichEdit (see page 4685)	TRichEdit is a wrapper for a Windows rich text edit control.
	TStatusBar (see page 4769)	TStatusBar represents a status bar.
	TStatusPanel (see page 4825)	TStatusPanel represents a single panel of a status bar.
	TStatusPanels (see page 4834)	TStatusPanels is a container for TStatusPanel (see page 4825) objects.
	TTabControl (see page 4841)	TTabControl is a tab set that has the appearance of notebook dividers.
	TTabSheet (see page 4919)	TTabSheet is an individual page in a TPageControl (see page 4459) object.
	TTextAttributes (see page 4982)	TTextAttributes represents the font characteristics of a section of text in a rich edit control.
	TToolBar (see page 4994)	TToolBar manages tool buttons and other controls, arranging them in rows and automatically adjusting their sizes and positions.
	TToolBarDockObject (see page 5102)	This is class ComCtrls.TToolBarDockObject.
	TToolBarEnumerator (see page 5105)	This is class ComCtrls.TToolBarEnumerator.
	TToolButton (see page 5107)	TToolButton is a button control in a TToolBar (see page 4994) object.
	TToolButtonActionLink (see page 5166)	TToolButtonActionLink is an action link designed for tool button control clients.
	TTrackBar (see page 5172)	TTrackBar is a wrapper for a track bar control.
	TTreeNode (see page 5230)	TTreeNode describes an individual node in a tree view control.
	TTreeNodes (see page 5266)	TTreeNodes maintains a list of tree nodes in a tree view control.
	TTreeNodesEnumerator (see page 5291)	This is class ComCtrls.TTreeNodesEnumerator.
	TTreeView (see page 5293)	TTreeView represents a window that displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files and directories on a disk.
	TUpDown (see page 5380)	TUpDown is a wrapper for a Windows up-down control.
	TWorkArea (see page 5425)	TWorkArea represents a rectangular subregion of a list view's client area.
	TWorkAreas (see page 5429)	TWorkAreas is a collection of TWorkArea (see page 5425) objects.

27.1.1 ECommonCalendarError Class

ECommonCalendarError is the exception class for invalid entries in a TCommonCalendar (see page 3558) component.

Class Hierarchy

Pascal

```
ECommonCalendarError = class(Exception);
```

C++

```
class ECommonCalendarError : public Exception;
```

File

ComCtrls

Description

Exceptions derived from ECommonCalendarError are raised by TCommonCalendar (see page 3558) descendants when an invalid date or time is entered.

See Also

TCommonCalendar (see page 3558)

EMonthCalError (see page 3421)

EDateTimeError (see page 3421)

27.1.2 EDateTimeError Class

EDateTimeError is the exception class for invalid date and time entries.

Class Hierarchy

Pascal

```
EDateTimeError = class(ECommonCalendarError);
```

C++

```
class EDateTimeError : public ECommonCalendarError;
```

File

ComCtrls

Description

EDateTimeError is raised by TDateTimePicker (see page 4049) when an invalid date or time is entered.

See Also

TDateTimePicker (see page 4049)

27.1.3 EMonthCalError Class

EMonthCalError is the exception class for invalid entries in a TMonthCalendar (see page 4397) component.

Class Hierarchy

Pascal

```
EMonthCalError = class(ECommonCalendarError);
```

C++

```
class EMonthCalError : public ECommonCalendarError;
```

File

ComCtrls

Description

EMonthCalError is raised by TMonthCalendar (see page 4397) when an invalid date is entered.

See Also

TMonthCalendar (see page 4397)

27.1.4 ETreeViewError Class

ETreeViewError is the exception class for tree view errors.

Class Hierarchy

Pascal

```
ETreeViewError = class(Exception);
```

C++

```
class ETreeViewError : public Exception;
```

File

ComCtrls

Description

ETreeViewError is raised when an error is made in a TCustomTreeView (see page 3957) or TTreeNode (see page 5230) object.

See Also

TCollection (see page 2731)

TComponent (see page 2762)

TCustomTreeView (see page 3957)

TTreeNode (see page 5230)

27.1.5 TAnimate Class

TAnimate is an animation control.

Class Hierarchy

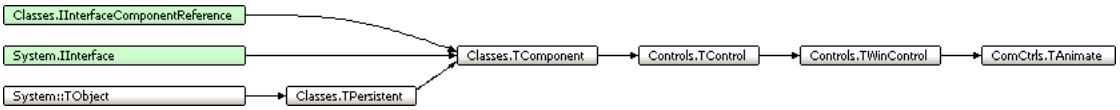

Pascal

```
TAnimate = class(TWinControl);
```

C++

```
class TAnimate : public TWinControl;
```

File

ComCtrls

Description

A TAnimate control displays an animation clip, consisting of a series of image frames. The control can load an animation from a file, a resource, or a set of standard animations. To specify an animation, set the FileName (see page 3450), CommonAVI (see page 3448), ResName (see page 3455), or ResID properties. At design time, you can browse through the frames of the animation using the context menu.

See Also

TImage (see page 12777)

TMediaPlayer (see page 17167)

Working with Multimedia

Animation Control

Graphic Controls

27.1.5.1 TAnimate Members

The following tables list the members exposed by TAnimate.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 3435)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.

	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 3436)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 3436)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 3437)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoClose (see page 3437)	This is DoClose, a member of class TAnimate.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoOpen (see page 3437)	This is DoOpen, a member of class TAnimate.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStart (see page 3437)	This is DoStart, a member of class TAnimate.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoStop (see page 3438)	This is DoStop, a member of class TAnimate.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.

	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 3438)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.

	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.

	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.

	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 3433)	Creates and initializes an animation control.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.

	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.

	Play (see page 3438)	Displays a sequence of frames from the AVI clip.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	Reset (see page 3439)	Clears the values of StartFrame (see page 3457) and StopFrame (see page 3458), and displays the first frame in the animation.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	Seek (see page 3439)	Displays a specified frame in the sequence.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Stop (see page 3440)	Interrupts the animation control when it is playing.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.

R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	FrameCount (see page 3451)	Indicates the total number of frames in the AVI clip.
R	FrameHeight (see page 3451)	Indicates the height, in pixels, of each image in the animation.
R	FrameWidth (see page 3451)	Indicates the width, in pixels, of each image in the animation.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	Open (see page 3452)	Indicates whether the animation is loaded into memory.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ResHandle (see page 3454)	Specifies the Windows handle for the module that contains the AVI clip as a resource.
	ResId (see page 3455)	Specifies the resource ID of the animation.
	ResName (see page 3455)	Specifies the resource name of the animation.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClose (see page 3461)	Occurs immediately after the animation control is closed.
	OnOpen (see page 3461)	Occurs immediately after the animation control is opened.
	OnStart (see page 3462)	Occurs immediately after the animation control starts playing.
	OnStop (see page 3462)	Occurs when the animation control finishes playing.

Published Properties

	Name	Description
	Active (see page 3441)	Indicates whether the animation control is playing the Animation.
	Align (see page 3442)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 3446)	Specifies how the control is anchored to its parent.
	AutoSize (see page 3446)	Specifies whether the control sizes itself automatically to fit animation.
	BorderWidth (see page 3447)	Specifies the width of the control's border.
	Center (see page 3447)	Determines whether the frames of the AVI clip are centered in the animation control.
	Color (see page 3447)	Specifies the background color of the control.
	CommonAVI (see page 3448)	Indicates whether the animation control displays a one of the common Windows AVI clips provided in Shell32.dll.
	Constraints (see page 3450)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	FileName (see page 3450)	Specifies file containing animation.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentColor (see page 3453)	Determines where a control looks for its color information.
	ParentShowHint (see page 3453)	Determines where a control looks to find out if its Help Hint should be shown.
	Repetitions (see page 3454)	Specifies the number of times to repeat the sequence of frames.
	ShowHint (see page 3456)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	StartFrame (see page 3457)	Specifies the first frame displayed when the animation control is activated.
	StopFrame (see page 3458)	Specifies the last frame displayed when the animation control is activated.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Timers (see page 3458)	Specifies whether the frames are displayed using a timer.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Transparent (see page 3459)	Specifies whether the background color of the frame matches the Color (see page 3447) property.
	Visible (see page 3459)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.5.2 TAnimate.Create Constructor

Creates and initializes an animation control.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TAnimate(TComponent * AOwner);
```

Description

Call Create to instantiate an animation control at runtime. Animation controls placed on forms at design time are created automatically.

AOwner is the component that is responsible for freeing the TAnimate (see page 3422) instance. It becomes the value of the Owner property.

C++ Examples:

```
/*
To run this example, create a new project, place a button on
the form, and insert the example code to the button click
event. Place the button on the bottom of the form to leave
room for the TAnimate component that will be created on the
fly.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TForm *TempForm = new TForm(this);
  TAnimate *pAnimate = new TAnimate(TempForm);
  pAnimate->Parent = TempForm;
  pAnimate->CommonAVI = aviFindFile;
  pAnimate->Active = true;
  TempForm->Show();
  // Simulate a lengthy process.  Alter this value
  // to accommodate your machine speed.
  for (int i = 0; i < 90000000; i++)
    Application->ProcessMessages();
  delete TempForm;
}
```

Delphi Examples:

```
{
To run this example, create a new project, place a button on
the form, and insert the example code to the button click
event. Place the button on the bottom of the form to leave
room for the TAnimate component that will be created on the
fly.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  TempForm: TForm;
  I: Integer;
begin
  TempForm := TForm.Create(Self);
  with TAnimate.Create(TempForm) do
  try
    Parent := TempForm;
    CommonAVI := aviFindFile;
    Active := True;
    TempForm.Show;
    // Simulate a lengthy process.  Alter this value
    // to accommodate your machine speed.
    for I := 0 to 90000000 do Application.ProcessMessages;
  finally
    TempForm.Release;
  end;
```

```
end;
```

See Also

CreateParented (see page 6123)

Owner (see page 2807)

27.1.5.3 TAnimate Methods

The methods of the TAnimate class are listed here.

Protected Methods

	Name	Description
	CanAutoSize (see page 3435)	Indicates how the control can be resized.
	CreateParams (see page 3436)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 3436)	Creates the underlying window.
	DestroyWnd (see page 3437)	Destroys the control's window.
	DoClose (see page 3437)	This is DoClose, a member of class TAnimate.
	DoOpen (see page 3437)	This is DoOpen, a member of class TAnimate.
	DoStart (see page 3437)	This is DoStart, a member of class TAnimate.
	DoStop (see page 3438)	This is DoStop, a member of class TAnimate.
	Loaded (see page 3438)	Initializes the control after it is loaded from a stream.

Public Methods

	Name	Description
	Play (see page 3438)	Displays a sequence of frames from the AVI clip.
	Reset (see page 3439)	Clears the values of StartFrame (see page 3457) and StopFrame (see page 3458), and displays the first frame in the animation.
	Seek (see page 3439)	Displays a specified frame in the sequence.
	Stop (see page 3440)	Interrupts the animation control when it is playing.

27.1.5.3.1 TAnimate.CanAutoSize Method

Indicates how the control can be resized.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize (see page 3446) property is true and an attempt is made to resize the control. It allows the animation control to adjust to the size of animation frames.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the control. If the Open (see page 3452) property is true, CanAutoSize adjusts these values so that they reflect the dimensions of the AVI video images and returns true. If the animation control is not Open (see page 3452), CanAutoSize returns false.

See Also

AutoSize (see page 5803)

FrameWidth (see page 3451)

FrameHeight (see page 3451)

27.1.5.3.2 TAnimate.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

27.1.5.3.3 TAnimate.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams (see page 6135)

CreateWindowHandle (see page 6137)

Perform (see page 5776)

DestroyWnd (see page 6141)

27.1.5.3.4 TAnimate.DestroyWnd Method

Destroys the control's window.

Pascal

```
procedure DestroyWnd; override;
```

C++

```
virtual __fastcall DestroyWnd();
```

Description

DestroyWnd is called when the control needs to replace its window. For example, changing properties that are implemented by window parameter settings require the control to be destroyed and then recreated using the CreateWnd method.

Before the window is destroyed, DestroyWnd saves a copy of the control's text in memory, frees any device contexts, and finally calls DestroyWindowHandle. When the window is recreated, the stored copy of the control's text is assigned to the new window.

See Also

DestroyWindowHandle (see page 6140)

RecreateWnd (see page 6171)

27.1.5.3.5 TAnimate.DoClose Method

Pascal

```
procedure DoClose; virtual;
```

C++

```
virtual __fastcall DoClose();
```

Description

This is DoClose, a member of class TAnimate.

27.1.5.3.6 TAnimate.DoOpen Method

Pascal

```
procedure DoOpen; virtual;
```

C++

```
virtual __fastcall DoOpen();
```

Description

This is DoOpen, a member of class TAnimate.

27.1.5.3.7 TAnimate.DoStart Method

Pascal

```
procedure DoStart; virtual;
```

C++

```
virtual __fastcall DoStart();
```

Description

This is DoStart, a member of class TAnimate.

27.1.5.3.8 TAnimate.DoStop Method

Pascal

```
procedure DoStop; virtual;
```

C++

```
virtual __fastcall DoStop();
```

Description

This is DoStop, a member of class TAnimate.

27.1.5.3.9 TAnimate.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded (see page 2782)

Action (see page 5797)

ActionChange (see page 5737)

27.1.5.3.10 TAnimate.Play Method

Displays a sequence of frames from the AVI clip.

Pascal

```
procedure Play(FromFrame: Word; ToFrame: Word; Count: Integer);
```

C++

```
__fastcall Play(Word FromFrame, Word ToFrame, int Count);
```

Description

Call Play to display the specified sequence of frames, repeating the sequence Count times. FromFrame is the first frame in the sequence, where 1 specifies the first frame in the clip, 2 specifies the second frame, and so on. ToFrame is the last frame in the sequence. The value of ToFrame should be greater than or equal to FromFrame and less than or equal to the value of the FrameCount (see page 3451) property. Count specifies the number of times the sequence should be repeated. If Count is 0,

the sequence is repeated until the Stop (see page 3440) method is called.

If the animation control is not Open (see page 3452) when Play is called, Play sets Open (see page 3452) to true.

Note: Calling Play is the same as setting StartFrame (see page 3457) to FromFrame, StopFrame (see page 3458) to ToFrame, Repetitions (see page 3454) to Count, and then setting Active (see page 3441) to true.

See Also

Active (see page 3441)

Open (see page 3452)

Repetitions (see page 3454)

StartFrame (see page 3457)

StopFrame (see page 3458)

27.1.5.3.11 TAnimate.Reset Method

Clears the values of StartFrame (see page 3457) and StopFrame (see page 3458), and displays the first frame in the animation.

Pascal

```
procedure Reset;
```

C++

```
__fastcall Reset();
```

Description

Call Reset to restore the animation control to a default state. Reset closes and reopens the animation control, generating OnClose (see page 3461) and OnOpen (see page 3461) events. The animation control is left open but not active.

See Also

Active (see page 3441)

OnClose (see page 3461)

OnOpen (see page 3461)

Open (see page 3452)

StartFrame (see page 3457)

StopFrame (see page 3458)

27.1.5.3.12 TAnimate.Seek Method

Displays a specified frame in the sequence.

Pascal

```
procedure Seek(Frame: Smallint);
```

C++

```
__fastcall Seek(Smallint Frame);
```

Description

Call Seek to move to a particular frame in the animation. The Frame parameter specifies which frame, where 1 is the first frame

in the sequence, 2 is the second frame, and so on. Frame must be a value less than or equal to FrameCount (see page 3451).

If the animation control is not open when Seek is called, Seek sets Open (see page 3452) to true.

See Also

FrameCount (see page 3451)

Open (see page 3452)

27.1.5.3.13 TAnimate.Stop Method

Interrupts the animation control when it is playing.

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

Call Stop to halt the sequence of images in an active animation control. Stop does the same thing as setting the Active (see page 3441) property to false.

See Also

Active (see page 3441)

Play (see page 3438)

27.1.5.4 TAnimate Properties

The properties of the TAnimate class are listed here.

Public Properties

	Name	Description
R	FrameCount (see page 3451)	Indicates the total number of frames in the AVI clip.
R	FrameHeight (see page 3451)	Indicates the height, in pixels, of each image in the animation.
R	FrameWidth (see page 3451)	Indicates the width, in pixels, of each image in the animation.
	Open (see page 3452)	Indicates whether the animation is loaded into memory.
	ResHandle (see page 3454)	Specifies the Windows handle for the module that contains the AVI clip as a resource.
	ResId (see page 3455)	Specifies the resource ID of the animation.
	ResName (see page 3455)	Specifies the resource name of the animation.

Published Properties

	Name	Description
	Active (see page 3441)	Indicates whether the animation control is playing the Animation.
	Align (see page 3442)	Determines how the control aligns within its container (parent control).
	Anchors (see page 3446)	Specifies how the control is anchored to its parent.
	AutoSize (see page 3446)	Specifies whether the control sizes itself automatically to fit animation.
	BorderWidth (see page 3447)	Specifies the width of the control's border.
	Center (see page 3447)	Determines whether the frames of the AVI clip are centered in the animation control.
	Color (see page 3447)	Specifies the background color of the control.
	CommonAVI (see page 3448)	Indicates whether the animation control displays a one of the common Windows AVI clips provided in Shell32.dll.

	Constraints (see page 3450)	Specifies the size constraints for the control.
	FileName (see page 3450)	Specifies file containing animation.
	ParentColor (see page 3453)	Determines where a control looks for its color information.
	ParentShowHint (see page 3453)	Determines where a control looks to find out if its Help Hint should be shown.
	Repetitions (see page 3454)	Specifies the number of times to repeat the sequence of frames.
	ShowHint (see page 3456)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	StartFrame (see page 3457)	Specifies the first frame displayed when the animation control is activated.
	StopFrame (see page 3458)	Specifies the last frame displayed when the animation control is activated.
	Timers (see page 3458)	Specifies whether the frames are displayed using a timer.
	Transparent (see page 3459)	Specifies whether the background color of the frame matches the Color (see page 3447) property.
	Visible (see page 3459)	Determines whether the component appears onscreen.

27.1.5.4.1 TAnimate.Active Property

Indicates whether the animation control is playing the Animation.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Set Active to true to cycle through the frames specified by the StartFrame (see page 3457) and StopFrame (see page 3458) properties by the number of iterations specified in the Repetitions (see page 3454) property. Set Active to false to interrupt the animation control when it is playing the animation file.

The animation control must be Open (see page 3452) before it can be Active.

C++ Examples:

```
/*
To run this example, create a new project, place a button on
the form, and insert the example code to the button click
event. Place the button on the bottom of the form to leave
room for the TAnimate component that will be created on the
fly.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TForm *TempForm = new TForm(this);
  TAnimate *pAnimate = new TAnimate(TempForm);
  pAnimate->Parent = TempForm;
  pAnimate->CommonAVI = aviFindFile;
  pAnimate->Active = true;
  TempForm->Show();
  // Simulate a lengthy process.  Alter this value
  // to accommodate your machine speed.
  for (int i = 0; i < 90000000; i++)
    Application->ProcessMessages();
  delete TempForm;
}
```

Delphi Examples:

```
{
To run this example, create a new project, place a button on
the form, and insert the example code to the button click
event. Place the button on the bottom of the form to leave
room for the TAnimate component that will be created on the
```

```
fly.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  TempForm: TForm;
  I: Integer;
begin
  TempForm := TForm.Create(Self);
  with TAnimate.Create(TempForm) do
  try
    Parent := TempForm;
    CommonAVI := aviFindFile;
    Active := True;
    TempForm.Show;
    // Simulate a lengthy process.  Alter this value
    // to accommodate your machine speed.
    for I := 0 to 90000000 do Application.ProcessMessages;
  finally
    TempForm.Release;
  end;
end;
```

See Also

OnStart (see page 3462)

OnStop (see page 3462)

Open (see page 3452)

Play (see page 3438)

Repetitions (see page 3454)

StartFrame (see page 3457)

Stop (see page 3440)

StopFrame (see page 3458)

27.1.5.4.2 TAnimate.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
```

```
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
```

```
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
```

```
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

27.1.5.4.3 TAnimate.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.5.4.4 TAnimate.AutoSize Property

Specifies whether the control sizes itself automatically to fit animation.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

If AutoSize is true (default), loading a new animation file causes the control to resize itself to match the new frame height and width.

See Also

FrameHeight (see page 3451)

FrameWidth (see page 3451)

Height (see page 5825)

Width (see page 5849)

27.1.5.4.5 TAnimate.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics or text drawn by the control is clipped to the area within the border.

See Also

ClientRect

BevelOuter (see page 6194)

27.1.5.4.6 TAnimate.Center Property

Determines whether the frames of the AVI clip are centered in the animation control.

Pascal

```
property Center: Boolean;
```

C++

```
__property Boolean Center;
```

Description

Use Center to position the frames of the clip in the animation control when FrameHeight (see page 3451) is not the same as ClientHeight or FrameWidth (see page 3451) is not the same as ClientWidth. When Center is true, the frames are centered both horizontally and vertically in the client area of the control. When Center is false, the top left corner of the frames is positioned at the top left corner of the client area.

See Also

AutoSize (see page 5803)

ClientHeight (see page 5808)

ClientWidth (see page 5811)

FrameHeight (see page 3451)

FrameWidth (see page 3451)

27.1.5.4.7 TAnimate.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

27.1.5.4.8 TAnimate.CommonAVI Property

Indicates whether the animation control displays a one of the common Windows AVI clips provided in Shell32.dll.

Pascal

```
property CommonAVI: TCommonAVI;
```

C++

```
__property TCommonAVI CommonAVI;
```

Description

Set CommonAVI to display one of the common Windows AVI clips in the animation control. The common Windows AVI clips appear in the Windows user interface when the user attempts common file actions. CommonAVI is aviNone when the animation control displays an AVI clip that was specified in some other manner.

Setting CommonAVI automatically sets the FileName (see page 3450) property to an empty string, ResHandle (see page

3454) and ResID to 0, and ResName (see page 3455) to an empty string.

Note: CommonAVI cannot be used when running under NT 3.51.

C++ Examples:

```
/*
To run this example, create a new project, place a button on
the form, and insert the example code to the button click
event. Place the button on the bottom of the form to leave
room for the TAnimate component that will be created on the
fly.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TForm *TempForm = new TForm(this);
  TAnimate *pAnimate = new TAnimate(TempForm);
  pAnimate->Parent = TempForm;
  pAnimate->CommonAVI = aviFindFile;
  pAnimate->Active = true;
  TempForm->Show();
  // Simulate a lengthy process.  Alter this value
  // to accommodate your machine speed.
  for (int i = 0; i < 90000000; i++)
    Application->ProcessMessages();
  delete TempForm;
}
```

Delphi Examples:

```
{
To run this example, create a new project, place a button on
the form, and insert the example code to the button click
event. Place the button on the bottom of the form to leave
room for the TAnimate component that will be created on the
fly.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  TempForm: TForm;
  I: Integer;
begin
  TempForm := TForm.Create(Self);
  with TAnimate.Create(TempForm) do
  try
    Parent := TempForm;
    CommonAVI := aviFindFile;
    Active := True;
    TempForm.Show;
    // Simulate a lengthy process.  Alter this value
    // to accommodate your machine speed.
    for I := 0 to 90000000 do Application.ProcessMessages;
  finally
    TempForm.Release;
  end;
end;
```

See Also

FileName (see page 3450)

ResHandle (see page 3454)

ResId (see page 3455)

ResName (see page 3455)

27.1.5.4.9 TAnimate.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.5.4.10 TAnimate.FileName Property

Specifies file containing animation.

Pascal

```
property FileName: string;
```

C++

```
__property AnsiString FileName;
```

Description

Setting FileName causes the control to load the animation from the specified file. If the control succeeds in loading the animation from the file, it sets the Open (see page 3452) property to true.

FileName must refer to a Windows AVI file. The sound track, if present, is ignored.

Note: Setting FileName automatically sets the CommonAVI (see page 3448) property to aviNone, ResHandle (see page 3454) and ResID to 0, and ResName (see page 3455) to an empty string.

See Also

Open (see page 3452)

CommonAVI (see page 3448)

ResHandle (see page 3454)

ResId (see page 3455)

ResName (see page 3455)

Adding Silent Video Clips to an Application

27.1.5.4.11 TAnimate.FrameCount Property

Indicates the total number of frames in the AVI clip.

Pascal

```
property FrameCount: Integer;
```

C++

```
__property int FrameCount;
```

Description

Read FrameCount to determine the number of frames that can be displayed by the animation control. The animation control can play all the frames in the AVI clip, or run through a subset of those frames.

See Also

Play (see page 3438)

StartFrame (see page 3457)

StopFrame (see page 3458)

27.1.5.4.12 TAnimate.FrameHeight Property

Indicates the height, in pixels, of each image in the animation.

Pascal

```
property FrameHeight: Integer;
```

C++

```
__property int FrameHeight;
```

Description

Read FrameHeight to determine the height required to fully display the frames of the animation. If AutoSize (see page 3446) is true, the animation control automatically sizes itself to the value of FrameHeight.

See Also

AutoSize (see page 5803)

ClientHeight (see page 5808)

FrameWidth (see page 3451)

27.1.5.4.13 TAnimate.FrameWidth Property

Indicates the width, in pixels, of each image in the animation.

Pascal

```
property FrameWidth: Integer;
```

C++

```
__property int FrameWidth;
```

Description

Read FrameWidth to determine the width required to fully display the frames of the animation. If AutoSize (see page 3446) is true, the animation control automatically resizes itself to the value of FrameWidth.

See Also

AutoSize (see page 5803)

ClientWidth (see page 5811)

FrameHeight (see page 3451)

27.1.5.4.14 TAnimate.Open Property

Indicates whether the animation is loaded into memory.

Pascal

```
property Open: Boolean;
```

C++

```
__property Boolean Open;
```

Description

Read Open to determine whether the animation is loaded and ready for playing. When the animation is specified using the CommonAVI (see page 3448), FileName (see page 3450), ResName (see page 3455), or ResID property, the animation control is opened automatically. When Open is true, the animation control displays the frame specified by the StartFrame (see page 3457) property until the clip is played using the Active (see page 3441) property or Play (see page 3438) method.

Set Open to false to free the resources used by the animation. When the animation control is not open, none of the frames can be shown.

Note: When Open is changed from false to true, the StartFrame (see page 3457) property is reset to 1 and the StopFrame (see page 3458) property is reset to FrameCount (see page 3451).

See Also

Active (see page 3441)

CommonAVI (see page 3448)

FileName (see page 3450)

FrameCount (see page 3451)

OnClose (see page 3461)

OnOpen (see page 3461)

Play (see page 3438)

ResHandle (see page 3454)

ResId (see page 3455)

ResName (see page 3455)

StartFrame (see page 3457)

StopFrame (see page 3458)

Adding Silent Video Clips to an Application

27.1.5.4.15 TAnimate.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

27.1.5.4.16 TAnimate.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.5.4.17 TAnimate.Repetitions Property

Specifies the number of times to repeat the sequence of frames.

Pascal

```
property Repetitions: Integer;
```

C++

```
__property int Repetitions;
```

Description

Set Repetitions to specify the number of times the sequence of frames in the animation is repeated when the Active (see page 3441) property is set to true. After the sequence is repeated by the specified number of times, the value of Active (see page 3441) becomes false again.

To show the sequence a single time, set Repetitions to 1. Set Repetitions to 0 to indicate that the sequence of frames should repeat indefinitely. When Repetitions is 0, use the Active (see page 3441) property or Stop (see page 3440) method to interrupt the animation.

The Play (see page 3438) method overrides Repetitions with the value passed as the Count parameter.

See Also

Active (see page 3441)

Play (see page 3438)

Stop (see page 3440)

Adding Silent Video Clips to an Application

27.1.5.4.18 TAnimate.ResHandle Property

Specifies the Windows handle for the module that contains the AVI clip as a resource.

Pascal

```
property ResHandle: THandle;
```

C++

```
__property THandle ResHandle;
```

Description

Set ResHandle at runtime to the instance handle or module handle of the module that includes the animation resource. After setting ResHandle, set the ResID or ResName (see page 3455) property to specify which resource in the indicated module is the AVI animation that should be displayed by the animation control.

If a value is not specified for ResHandle, the animate control uses the instance of the application's main executable.

Note: Setting ResHandle automatically sets the CommonAVI (see page 3448) property to aviNone and the FileName (see page 3450) property to an empty string.

See Also

CommonAVI (see page 3448)

FileName (see page 3450)

Hinstance

ResId (see page 3455)

ResName (see page 3455)

Adding Silent Video Clips to an Application

27.1.5.4.19 TAnimate.ResId Property

Specifies the resource ID of the animation.

Pascal

```
property ResId: Integer;
```

C++

```
__property int ResId;
```

Description

Set ResId to identify an AVI resource that contains the animation. Use ResHandle (see page 3454) to indicate the module that contains the resource identified by ResId.

If the resource file identifies the animation by name rather than number, use the ResName (see page 3455) property instead.

Note: Setting ResId automatically sets the CommonAVI (see page 3448) property to aviNone, the FileName (see page 3450) property to an empty string, and the ResName (see page 3455) property to an empty string.

See Also

CommonAVI (see page 3448)

FileName (see page 3450)

ResHandle (see page 3454)

ResName (see page 3455)

Adding Silent Video Clips to an Application

27.1.5.4.20 TAnimate.ResName Property

Specifies the resource name of the animation.

Pascal

```
property ResName: string;
```

C++

```
__property AnsiString ResName;
```

Description

Set ResName to identify an AVI resource that contains the animation. Use ResHandle (see page 3454) to indicate the module

that contains the resource identified by ResName.

If the resource file identifies the animation by number rather than number, use the ResID property instead.

Note: Setting ResName automatically sets the CommonAVI (see page 3448) property to aviNone, the FileName (see page 3450) property to an empty string, and the ResID property to 0.

See Also

CommonAVI (see page 3448)

FileName (see page 3450)

ResHandle (see page 3454)

ResId (see page 3455)

Adding Silent Video Clips to an Application

27.1.5.4.21 TAnimate.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
```

```
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.5.4.22 TAnimate.StartFrame Property

Specifies the first frame displayed when the animation control is activated.

Pascal

```
property StartFrame: Smallint;
```

C++

```
__property Smallint StartFrame;
```

Description

Set StartFrame to specify the first frame shown when the Active (see page 3441) property is set to true. When StartFrame is 1, the clip is played from the very beginning, when StartFrame is 2, the clip begins with the second frame, and so on. If Repetitions

(see page 3454) is not 1, after the sequence of frames is played, it starts over at the frame specified by StartFrame (which may not be the first frame in the clip).

When the animation control is Open (see page 3452) but not Active (see page 3441), it displays the frame specified by StartFrame.

Note: The Play (see page 3438) method overrides StartFrame with the value passed as the FromFrame parameter.

See Also

Active (see page 3441)

FrameCount (see page 3451)

Play (see page 3438)

Repetitions (see page 3454)

Seek (see page 3439)

StopFrame (see page 3458)

27.1.5.4.23 TAnimate.StopFrame Property

Specifies the last frame displayed when the animation control is activated.

Pascal

```
property StopFrame: Smallint;
```

C++

```
__property Smallint StopFrame;
```

Description

Set StopFrame to specify the last frame shown every time the animation control displays its sequence of frames. When StopFrame is 0, the clip is played to the very end. Otherwise, StopFrame is the number of the frame in the sequence. When the Active (see page 3441) property is set to true, the animation control displays the sequence of frames from StartFrame (see page 3457) to StopFrame, repeating this sequence by the number of iterations specified by the Repetitions (see page 3454) property.

Note: The Play (see page 3438) method overrides StopFrame with the value passed as the ToFrame parameter.

See Also

Active (see page 3441)

FrameCount (see page 3451)

Play (see page 3438)

Repetitions (see page 3454)

StartFrame (see page 3457)

27.1.5.4.24 TAnimate.Timers Property

Specifies whether the frames are displayed using a timer.

Pascal

```
property Timers: Boolean;
```

C++

```
__property Boolean Timers;
```

Description

Set Timers to specify whether the frames in the animation should be changed in response to timer messages, or whether the sequence progresses asynchronously in a separate execution thread. When Timers is true, the frames progress in response to timer messages. This is useful for synchronizing the animation sequence with other actions such as playing a sound track. When Timers is false, the animation is played in a separate thread.

See Also

OnStart (see page 3462)

TMediaPlayer (see page 17167)

Ttimer

27.1.5.4.25 TAnimate.Transparent Property

Specifies whether the background color of the frame matches the Color (see page 3447) property.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Set Transparent to true to replace the background color stored in the animation with the value of the Color (see page 3447) property, or the parent control's color if the value of ParentColor (see page 3453) is true. Set Transparent to false to display the background color stored in the animation.

See Also

Color (see page 5812)

ParentColor (see page 5834)

27.1.5.4.26 TAnimate.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
```

```
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.5.5 TAnimate Events

The events of the TAnimate class are listed here.

Published Events

	Name	Description
	OnClose (see page 3461)	Occurs immediately after the animation control is closed.
	OnOpen (see page 3461)	Occurs immediately after the animation control is opened.
	OnStart (see page 3462)	Occurs immediately after the animation control starts playing.
	OnStop (see page 3462)	Occurs when the animation control finishes playing.

27.1.5.5.1 TAnimate.OnClose Event

Occurs immediately after the animation control is closed.

Pascal

```
property OnClose: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClose;
```

Description

Write an OnClose event handler to take specific action when the Open (see page 3452) property is set to false. For example, if the animation control is used to cycle through a series of AVI clips, the clip could be changed in an OnClose event handler.

To respond when the clip stops playing, but the animation control remains open, use the OnStop (see page 3462) event.

See Also

OnOpen (see page 3461)

OnStop (see page 3462)

Open (see page 3452)

27.1.5.5.2 TAnimate.OnOpen Event

Occurs immediately after the animation control is opened.

Pascal

```
property OnOpen: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnOpen;
```

Description

Write an OnOpen event handler to take specific action when the Open (see page 3452) property changes from false to true.

This is a good opportunity to set the Repetitions (see page 3454), StartFrame (see page 3457), and StopFrame (see page 3458) properties before the AVI clip is activated, or to seek to a particular frame.

See Also

Active (see page 3441)

OnClose (see page 3461)

OnStart (see page 3462)

Open (see page 3452)

Repetitions (see page 3454)

Seek (see page 3439)

StartFrame (see page 3457)

StopFrame (see page 3458)

27.1.5.5.3 TAnimate.OnStart Event

Occurs immediately after the animation control starts playing.

Pascal

```
property OnStart: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnStart;
```

Description

Write an OnStart event handler to take specific action when the animation is played. Setting the Active (see page 3441) property to true or calling the Play (see page 3438) method triggers an OnStart event. OnStart is useful for launching other actions that should be synchronized with the playing of the animation sequence.

See Also

Active (see page 3441)

OnOpen (see page 3461)

OnStop (see page 3462)

Play (see page 3438)

Timers (see page 3458)

27.1.5.5.4 TAnimate.OnStop Event

Occurs when the animation control finishes playing.

Pascal

```
property OnStop: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnStop;
```

Description

Write an OnStop event handler to take specific action when the animation finishes playing all its repetitions. For example, use an OnStop event handler to shut down any simultaneous actions that were started in an OnStart (see page 3462) event handler.

See Also

Active (see page 3441)

OnClose (see page 3461)

OnStart (see page 3462)

Repetitions (see page 3454)

Stop (see page 3440)

27.1.6 TComboBoxEx Class

TComboBoxEx represents a combo box that supports extended combo box features such as images on the list entries.

Class Hierarchy

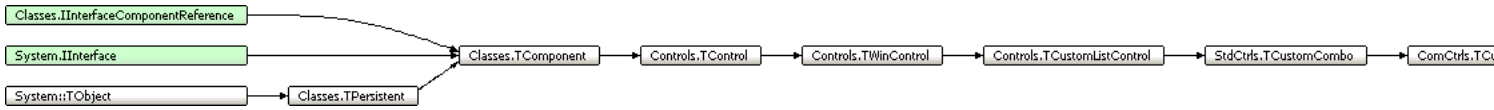

Pascal

```
TComboBoxEx = class(TCustomComboBoxEx);
```

C++

```
class TComboBoxEx : public TCustomComboBoxEx;
```

File

ComCtrls

Description

Add TComboBoxEx to a form when you want to include a combo box that uses the images in an image list or when you want to control how far the items in the drop-down list are indented. TComboBoxEx provides greater support of items than the TComboBox control, but omits certain features as well (owner-drawn items, sorting of list items, and so on).

See Also

TComboBox (see page 21064)

TColorBox (see page 12182)

TDBComboBox (see page 8657)

27.1.6.1 TComboBoxEx Members

The following tables list the members exposed by TComboBoxEx.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnCloseUp (see page 21199)	Occurs when the drop-down list closes up due to some user action.

	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefEditProc (see page 21167)	This is FDefEditProc, a member of class TCustomCombo.
	FDefListProc (see page 21167)	This is FDefListProc, a member of class TCustomCombo.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FDropHandle (see page 21168)	This is FDropHandle, a member of class TCustomCombo.
	FDroppingDown (see page 21168)	This is FDroppingDown, a member of class TCustomCombo.
	FEditHandle (see page 21168)	This is FEditHandle, a member of class TCustomCombo.
	FEditInstance (see page 21168)	This is FEditInstance, a member of class TCustomCombo.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFocusChanged (see page 21168)	This is FFocusChanged, a member of class TCustomCombo.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FIsFocused (see page 21169)	This is FIsFocused, a member of class TCustomCombo.
	FListHandle (see page 21169)	This is FListHandle, a member of class TCustomCombo.
	FListInstance (see page 21169)	This is FListInstance, a member of class TCustomCombo.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveIndex (see page 21169)	This is FSaveIndex, a member of class TCustomCombo.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActionChange (see page 3692)	Responds when the action for which this combo box is a client changes.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustDropDown (see page 21171)	Adjusts the size and position of the drop down list.

	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21171)	Generates an OnChange (see page 21197) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	CloseUp (see page 21172)	Generates an OnCloseUp (see page 21199) event.
	CMColorChanged (see page 3693)	Responds when the Color property changes.
	CMParentColorChanged (see page 3693)	Responds when the ParentColor property changes.
	CNNotify (see page 3693)	Responds to notifications of changes in the combo box.
	ComboExWndProc (see page 3694)	Responds to Windows messages sent to the combo box.
	ComboWndProc (see page 21173)	Responds to Windows messages sent to the edit control or drop-down list.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 3694)	Initializes the window-creation parameter record.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 3694)	Creates the underlying Windows control that implements the extended combo box.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 21174)	Creates the Windows control that represents the combo box.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.

V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWindowHandle (see page 21174)	Destroys all of the windows controls that implement the combo box.
V	DestroyWnd (see page 3695)	Destroys the Windows control that represents the combo box.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DropDown (see page 21175)	Generates an OnDropDown (see page 21199) event.
V	EditWndProc (see page 21175)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
V	GetActionLinkClass (see page 3695)	Returns the action link class that is associated with extended combo boxes.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.

V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetDroppedDown (see page 21176)	Returns the value of the DroppedDown (see page 21185) property.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetItemCount (see page 3696)	Returns the number of items in the Items property list.
A	GetItemCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
V	GetItemHt (see page 3696)	Returns the height, in pixels, allocated for each item in the drop-down list.
A	GetItemHt (see page 21177)	Returns the value of the ItemHeight (see page 21186) property.
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
V	GetItemIndex (see page 21177)	Returns the value of the ItemIndex property.
V	GetItemsClass (see page 3696)	Returns the class used to implement the Items property.
A	GetItemsClass (see page 21177)	Indicates the class that implements the Items (see page 21187) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelLength (see page 21178)	Returns the value of the SelLength (see page 21190) property.
	GetSelStart (see page 21178)	Returns the value of the SelStart (see page 21193) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	ListWndProc (see page 21178)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 21179)	Initializes the combo box after its form is loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 3697)	Responds to notifications that objects are about to be inserted or removed.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.

	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Select (see page 21179)	Generates an OnSelect (see page 21201) event.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetDropDownCount (see page 3697)	Sets the value of the DropDownCount property.
V	SetDropDownCount (see page 21180)	Sets the value of the DropDownCount (see page 21185) property.
	SetDroppedDown (see page 21180)	Sets the value of the DroppedDown (see page 21185) property.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetItemHeight (see page 21181)	Sets the value of the ItemHeight (see page 21186) property.
A	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
V	SetItemIndex (see page 21181)	Sets the value of the ItemIndex property.
V	SetItems (see page 21181)	Sets the value of the Items (see page 21187) property.
	SetMaxLength (see page 21182)	Sets the value of the MaxLength (see page 21189) property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.

V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelLength (see page 21182)	Sets the value of the SelLength (see page 21190) property.
	SetSelStart (see page 21182)	Sets the value of the SelStart (see page 21193) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMHelp (see page 3697)	This is WMHelp, a member of class TCustomComboBoxEx.
	WMLButtonDown (see page 3698)	Responds when the user clicks the left mouse button on the combo box.
V	WndProc (see page 3698)	Processes messages received by the combo box.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WndProc (see page 21183)	Processes messages received by the combo box.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
R	EditHandle (see page 21185)	Returns the window handle for the edit portion of the combo box.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	ItemCount (see page 21186)	Specifies the number of items in the drop-down list.

R	ListHandle (see page 21189)	Provides access to the window handle of the list of items in the combo box.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
A	AddItem (see page 5940)	Adds an item to the list control.
V	AddItem (see page 21170)	Adds an item to the drop-down list of items.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 5940)	Deletes all of the items from the list control.
V	Clear (see page 21172)	Deletes all text from the edit box and all items from the Items (see page 21187) list.
A	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
V	ClearSelection (see page 21172)	Deselects any selected items in the drop-down list.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
A	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
V	CopySelection (see page 21173)	Copies the selected item in the drop-down list to another list control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 3691)	Creates and initializes a TCustomComboBoxEx (see page 3678) instance.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 21166)	Creates and initializes a TCustomCombo (see page 21154) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.

	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DeleteSelected (see page 5941)	Deletes all the selected items.
	DeleteSelected (see page 21174)	Deletes the selected item from the drop-down list.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 3691)	Destroys the combo box and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 21166)	Destroys the combo box and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 3695)	Indicates whether the combo box has input focus.
	Focused (see page 6150)	Determines whether the control has input focus.
	Focused (see page 21175)	Indicates whether the combo box has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.

	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
A	SelectAll (see page 5943)	Selects all items or all text in the selected item.
V	SelectAll (see page 21180)	Selects the entire block of text in the edit box.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.

V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 21184)	Provides access to the drawing surface of the combo box.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DroppedDown (see page 21185)	Indicates whether the drop-down list is currently displayed.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.

	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	Items (see page 21187)	Provides access to the list of items (strings) in the list portion of the combo box.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelLength (see page 21190)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 21193)	Specifies the position of the first selected character in the edit portion of the combo box.
	SelText (see page 3701)	Represents the selected text in the edit region.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnBeginEdit (see page 3504)	Occurs when the user starts editing the text of the combo box.
	OnChange (see page 3504)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 3507)	Occurs when the user clicks the control.
	OnContextPopup (see page 3509)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 3509)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 3510)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 3512)	Occurs when the user drags an object over a control.
	OnDropDown (see page 3514)	Occurs when the user opens the drop-down list.
	OnEndDock (see page 3516)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 3516)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEndEdit (see page 3517)	Occurs when the user finishes editing the text of the combo box.
	OnEnter (see page 3517)	Occurs when a control receives the input focus.
	OnExit (see page 3520)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 3521)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 3524)	Occurs when a key is pressed.
	OnKeyUp (see page 3525)	Occurs when the user releases a key that has been pressed.
	OnMouseMove (see page 3527)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnSelect (see page 3529)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 3530)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 3530)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Action (see page 3476)	Designates the action associated with the control.
	Align (see page 3477)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.

	Anchors (see page 3480)	Specifies how the control is anchored to its parent.
	AutoCompleteOptions (see page 3481)	Specifies options that control autocompletion.
	BiDiMode (see page 3481)	Specifies the bi-directional mode for the control.
	Color (see page 3482)	Specifies the background color of the control.
	Constraints (see page 3482)	Specifies the size constraints for the control.
	Ctl3D (see page 3483)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 3483)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 3484)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 3484)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownCount (see page 3486)	Specifies the maximum number of items displayed in the drop-down list.
	Enabled (see page 3486)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 3487)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Images (see page 3488)	Specifies the images that appear next to items in the combo box list.
	ImeMode (see page 3488)	Determines the behavior of the input method editor (IME).
	ImeName (see page 3489)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	ItemHeight (see page 3489)	Specifies the height, in pixels, of the items in the drop-down list.
	ItemsEx (see page 3490)	Lists the items that appear in the list portion of the combo box.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	MaxLength (see page 3490)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 3491)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 3491)	Determines where a control looks for its color information.
	ParentCtl3D (see page 3492)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 3492)	Determines where a control looks for its font information.
	ParentShowHint (see page 3492)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 3493)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 3498)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 3499)	Determines the display style of the combo box.
	StyleEx (see page 3499)	Determines the appearance and behavior of the items in the list.
	TabOrder (see page 3500)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 3500)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Text (see page 3501)	Contains a text string associated with the control.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 3502)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.6.2 TComboBoxEx Properties

The properties of the TComboBoxEx class are listed here.

Published Properties

	Name	Description
	Action (see page 3476)	Designates the action associated with the control.
	Align (see page 3477)	Determines how the control aligns within its container (parent control).
	Anchors (see page 3480)	Specifies how the control is anchored to its parent.
	AutoCompleteOptions (see page 3481)	Specifies options that control autocompletion.
	BiDiMode (see page 3481)	Specifies the bi-directional mode for the control.
	Color (see page 3482)	Specifies the background color of the control.
	Constraints (see page 3482)	Specifies the size constraints for the control.
	Ctl3D (see page 3483)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 3483)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 3484)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 3484)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownCount (see page 3486)	Specifies the maximum number of items displayed in the drop-down list.
	Enabled (see page 3486)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 3487)	Controls the attributes of text written on or in the control.
	Images (see page 3488)	Specifies the images that appear next to items in the combo box list.
	ImeMode (see page 3488)	Determines the behavior of the input method editor (IME).
	ImeName (see page 3489)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	ItemHeight (see page 3489)	Specifies the height, in pixels, of the items in the drop-down list.
	ItemsEx (see page 3490)	Lists the items that appear in the list portion of the combo box.
	MaxLength (see page 3490)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	ParentBiDiMode (see page 3491)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 3491)	Determines where a control looks for its color information.
	ParentCtl3D (see page 3492)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 3492)	Determines where a control looks for its font information.
	ParentShowHint (see page 3492)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 3493)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 3498)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 3499)	Determines the display style of the combo box.
	StyleEx (see page 3499)	Determines the appearance and behavior of the items in the list.
	TabOrder (see page 3500)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 3500)	Determines if the user can tab to a control.
	Text (see page 3501)	Contains a text string associated with the control.
	Visible (see page 3502)	Determines whether the component appears onscreen.

27.1.6.2.1 TComboBoxEx.Action Property

Designates the action associated with the control.

Pascal

```
property Action: TBasicAction;
```

C++

```
__property TBasicAction Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

27.1.6.2.2 TComboBoxEx.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
```

```
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

27.1.6.2.3 TComboBoxEx.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control

has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.6.2.4 TComboBoxEx.AutoCompleteOptions Property

Specifies options that control autocompletion.

Pascal

```
property AutoCompleteOptions: TAutoCompleteOptions;
```

C++

```
__property TAutoCompleteOptions AutoCompleteOptions;
```

Description

AutoCompleteOptions specifies options that control autocompletion.

27.1.6.2.5 TComboBoxEx.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

27.1.6.2.6 TComboBoxEx.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

27.1.6.2.7 TComboBoxEx.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.6.2.8 TComboBoxEx.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

27.1.6.2.9 TComboBoxEx.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

27.1.6.2.10 TComboBoxEx.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

27.1.6.2.11 TComboBoxEx.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
```

```
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

27.1.6.2.12 TComboBoxEx.DropDownCount Property

Specifies the maximum number of items displayed in the drop-down list.

Pascal

```
property DropDownCount: Integer;
```

C++

```
__property int DropDownCount;__property int DropDownCount;
```

Description

By default, the drop-down list is long enough to contain eight items without requiring the user to scroll to see them all. To make the drop-down list smaller or larger, specify a number larger or smaller than eight as the DropDownCount value.

If the DropDownCount value is larger than the number of items in the Items property, the drop-down list will be just large enough to hold all the possible choices and no more. If the DropDownCount value is smaller then the number of item in the Items property, the drop down list will display a scroll bar.

DropDownCount is meaningful only if Style is not csExSimple.

See Also

ItemsEx (see page 3700)

Style (see page 3701)

27.1.6.2.13 TComboBoxEx.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.6.2.14 TComboBoxEx.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

27.1.6.2.15 TComboBoxEx.Images Property

Specifies the images that appear next to items in the combo box list.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Use Images to supply the list of images that appear beside items in the combo box's drop-down list. Each combo box item has an ImageIndex property that identifies an image in this list. In addition, Images can contain images that are used as overlay masks for the item images (to create non-rectangular images) and images that are used only when items are selected. The combo box items reference these images using their OverlayImageIndex and SelectedImageIndex properties.

See Also

ImageIndex

ItemsEx (see page 3700)

SelectedImageIndex (see page 3552)

OverlayImageIndex (see page 3551)

27.1.6.2.16 TComboBoxEx.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

27.1.6.2.17 TComboBoxEx.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

27.1.6.2.18 TComboBoxEx.ItemHeight Property

Specifies the height, in pixels, of the items in the drop-down list.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Read ItemHeight to determine the height of the items in the drop-down list.

The effect of setting ItemHeight varies, depending on the type of combo box and the style. For example, TComboBox lets you

set the item height only when Style is set to csOwnerDrawFixed or csOwnerDrawVariable. If Style is csOwnerDrawVariable, ItemHeight is the default height for drawing list items, but this value can be overridden by an OnMeasureItem event handler.

TComboBoxEx does not let you change the value of ItemHeight.

Note: Combo boxes that do not support changes to ItemHeight do not raise an exception when you attempt to change ItemHeight; they simply ignore the attempt.

See Also

Items (see page 21187)

OnMeasureItem (see page 21225)

Style (see page 21224)

27.1.6.2.19 TComboBoxEx.ItemsEx Property

Lists the items that appear in the list portion of the combo box.

Pascal

```
property ItemsEx: TComboExItems;
```

C++

```
__property TComboExItems ItemsEx;
```

Description

Read Items to access the list of items that appears in the combo box. ItemsEx is a collection of TComboExItem objects.

Set the value of Items to copy the items from another collection of TComboExItem objects.

Use the methods of Items to add, insert, delete and move items. Use the properties of individual items in the collection to specify the appearance of each item.

Note: To work with only the strings in the combo box, and not the other properties of the items, you can use the Items property instead.

See Also

ItemIndex

TComboExItem (see page 3546)

Items (see page 21187)

27.1.6.2.20 TComboBoxEx.MaxLength Property

Specifies the maximum number of characters the user can type into the edit portion of the combo box.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that the user can enter into the edit portion of the combo box. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the

limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

See Also

Text (see page 5842)

27.1.6.2.21 TComboBoxEx.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

27.1.6.2.22 TComboBoxEx.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

27.1.6.2.23 TComboBoxEx.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

27.1.6.2.24 TComboBoxEx.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

27.1.6.2.25 TComboBoxEx.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.6.2.26 TComboBoxEx.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
```

```
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
```

```
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;
```

```
{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);
```

```
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.6.2.27 TComboBoxEx.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.6.2.28 TComboBoxEx.Style Property

Determines the display style of the combo box.

Pascal

```
property Style: TComboBoxExStyle;
```

C++

```
__property TComboBoxExStyle Style;
```

Description

Set Style to specify how the list and edit region of the combo box should appear.

See Also

Height (see page 5825)

DropDownCount (see page 3699)

27.1.6.2.29 TComboBoxEx.StyleEx Property

Determines the appearance and behavior of the items in the list.

Pascal

```
property StyleEx: TComboBoxExStyles;
```

C++

```
__property TComboBoxExStyles StyleEx;
```

Description

Set StyleEx to configure how the items in the list look and behave.

See Also

Indent

ImageIndex

ItemsEx (see page 3700)

27.1.6.2.30 TComboBoxEx.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

27.1.6.2.31 TComboBoxEx.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

27.1.6.2.32 TComboBoxEx.Text Property

Contains a text string associated with the control.

Pascal

```
property Text: TCaption;
```

C++

```
__property TCaption Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
  Size++;                        //Add room for null character
  char *Buffer = new char[Size];  //Creates Buffer dynamic variable
  Edit1->GetTextBuf(Buffer,Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer;
  delete [] Buffer;
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
```

```
  puts this string in another edit box when the user clicks the
  button on the form.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    Buffer: PChar;
    Size: Byte;
  begin
    Size := Edit1.GetTextLen;       {Get length of string in Edit1}
    Inc(Size);                      {Add room for null character}
    GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
    Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
    Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
    FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
  end;

  {
  Note: the same thing could be accomplished more simply as follows:
  }
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    Edit2.Text := Edit1.Text;
  end;
```

See Also

Caption (see page 5806)

Lines (see page 21380)

27.1.6.2.33 TComboBoxEx.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.6.3 TComboBoxEx Events

The events of the TComboBoxEx class are listed here.

Published Events

	Name	Description
	OnBeginEdit (see page 3504)	Occurs when the user starts editing the text of the combo box.
	OnChange (see page 3504)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 3507)	Occurs when the user clicks the control.
	OnContextPopup (see page 3509)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 3509)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 3510)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 3512)	Occurs when the user drags an object over a control.
	OnDropDown (see page 3514)	Occurs when the user opens the drop-down list.
	OnEndDock (see page 3516)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 3516)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEndEdit (see page 3517)	Occurs when the user finishes editing the text of the combo box.
	OnEnter (see page 3517)	Occurs when a control receives the input focus.
	OnExit (see page 3520)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 3521)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 3524)	Occurs when a key is pressed.
	OnKeyUp (see page 3525)	Occurs when the user releases a key that has been pressed.
	OnMouseMove (see page 3527)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnSelect (see page 3529)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 3530)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 3530)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

27.1.6.3.1 TComboBoxEx.OnBeginEdit Event

Occurs when the user starts editing the text of the combo box.

Pascal

```
property OnBeginEdit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnBeginEdit;
```

Description

Write an OnBeginEdit event handler to take specific action immediately before the user edits the text in the combo box.

See Also

OnEndEdit (see page 3702)

Text (see page 5842)

27.1.6.3.2 TComboBoxEx.OnChange Event

Occurs when the user changes the text displayed in the edit region.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action immediately after the user edits the text in the edit region or selects an item from the list. The Text property gives the new value in the edit region.

Note: OnChange only occurs in response to user actions. Changing the Text property programmatically does not trigger an OnChange event.

C++ Examples:

```
/*
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
*/
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  if (pCB)
    PageControl1->ActivePage = (TTabSheet *)(pCB->Items->Objects[pCB->ItemIndex]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
    ComboBox1->Items->AddObject(PageControl1->Pages[i]->Name, PageControl1->Pages[i]);
    ComboBox1->ItemIndex = 0;
  }
}
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
```

```
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to PageControl1.PageCount - 1 do
    ComboBox1.Items.AddObject(PageControl1.Pages[i].Name,
      PageControl1.Pages[i]);
  ComboBox1.ItemIndex := 0;
end;

procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  if (Sender is TComboBox) then
    with (Sender as TComboBox) do
      PageControl1.ActivePage := TTabSheet(Items.Objects[ItemIndex]);
end;
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

See Also

Text (see page 5842)

OnCloseUp (see page 21199)

27.1.6.3.3 TComboBoxEx.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
    for (int i = 0; i < 10; i++)
    {
      TTabSheet *pPage = new TTabSheet(this);
      pPage->PageControl = PageControl1;
      pPage->Caption = AnsiString("Page") + IntToStr(i);
      TEdit *pEdit = new TEdit(this);
      pEdit->Parent = pPage;
      pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
      pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
    }
    PageControl1Change(Sender);
  }
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

27.1.6.3.4 TComboBoxEx.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.6.3.5 TComboBoxEx.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

27.1.6.3.6 TComboBoxEx.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

27.1.6.3.7 TComboBoxEx.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}
```

```
/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
```

```
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.6.3.8 TComboBoxEx.OnDropDown Event

Occurs when the user opens the drop-down list.

Pascal

```
property OnDropDown: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDropDown;
```

Description

Write an OnDropDown event handler to implement special processing that needs to occur only when the drop-down list is activated.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if (((TForm1 *)Screen->CustomForms[i])->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
    end;
    Items.EndUpdate; {reenable painting }
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;

procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.Show;
end;
```

See Also

DroppedDown (see page 21185)

OnCloseUp (see page 21199)

27.1.6.3.9 TComboBoxEx.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.6.3.10 TComboBoxEx.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
```

```
    if (Sender->ClassNameIs("TLabel"))
    {
      AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
      if (Target)
        S = S + "accepted!";
      else
        S = S + "rejected.";
      Status->Caption = S;
    }
  }
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

27.1.6.3.11 TComboBoxEx.OnEndEdit Event

Occurs when the user finishes editing the text of the combo box.

Pascal

```
property OnEndEdit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEndEdit;
```

Description

Write an OnEndEdit event handler to take specific action immediately after the user edits the text in the combo box.

See Also

OnBeginEdit (see page 3702)

Text (see page 5842)

27.1.6.3.12 TComboBoxEx.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
```

```
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
```

```
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

27.1.6.3.13 TComboBoxEx.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
```

```
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

27.1.6.3.14 TComboBoxEx.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
```

```
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
```

```
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

27.1.6.3.15 TComboBoxEx.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an

OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

27.1.6.3.16 TComboBoxEx.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
```

```
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

27.1.6.3.17 TComboBoxEx.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
```

```
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
```

```
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.6.3.18 TComboBoxEx.OnSelect Event

Occurs when the user selects a string in the drop-down list.

Pascal

```
property OnSelect: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSelect;
```

Description

Write an OnSelect event handler to respond when the user changes the selected item in the drop-down list. OnSelect occurs after the selection has changed and the Text property reflects the new item.

See Also

autocomplete (see page 21220)

ItemIndex

Text (see page 5842)

27.1.6.3.19 TComboBoxEx.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

27.1.6.3.20 TComboBoxEx.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

27.1.7 TComboBoxExActionLink Class

Class Hierarchy

Pascal

```
TComboBoxExActionLink = class(TListActionLink);
```

C++

```
class TComboBoxExActionLink : public TListActionLink;
```

File

ComCtrls

Description

This is class ComCtrls.TComboBoxExActionLink.

27.1.7.1 TComboBoxExActionLink Members

The following tables list the members exposed by TComboBoxExActionLink.

Protected Fields

	Name	Description
	FAction (see page 2722)	This is FAction, a member of class TBasicActionLink.
	FClient (see page 5894)	This is FClient, a member of class TControlActionLink.
	FClient (see page 6236)	This is FClient, a member of class TWinControlActionLink.

Protected Methods

	Name	Description
	AddItem (see page 3534)	This is the overview for the AddItem method overload.
	AddItem (see page 15747)	This is the overview for the AddItem method overload.
	AssignClient (see page 2723)	This is AssignClient, a member of class TBasicActionLink.
	AssignClient (see page 5895)	Assigns a client control that is linked to the associated action.
	AssignClient (see page 6236)	Specifies the client control.
	Change (see page 2723)	This is Change, a member of class TBasicActionLink.
	DoShowHint (see page 5896)	Indicates whether the hint is shown, and modified.
	IsActiveLinked (see page 15748)	This is IsActiveLinked, a member of class TListActionLink.
	IsCaptionLinked (see page 268)	Indicates whether the Caption properties for the action and client are linked.
	IsCaptionLinked (see page 5896)	Indicates whether the caption for the action and the client are linked.
	IsCheckedLinked (see page 269)	Indicates whether the Checked properties for the action and client are linked.
	IsDropdownMenuLinked (see page 5896)	This is IsDropdownMenuLinked, a member of class TControlActionLink.
	IsEnabledLinked (see page 269)	Indicates whether the Enabled properties for the action and client are linked.
	IsEnabledLinked (see page 5897)	Indicates whether the enabled state for the action and the client are linked.
	IsEnableDropdownLinked (see page 5897)	This is IsEnableDropdownLinked, a member of class TControlActionLink.
	IsGroupIndexLinked (see page 270)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsHelpContextLinked (see page 270)	Indicates whether the HelpContext properties for the action and client controls are linked.
	IsHelpContextLinked (see page 6236)	Indicates whether the Help context for the action and the client are linked.
	IsHelpLinked (see page 271)	Indicates whether the help properties for the action and client are linked.
	IsHelpLinked (see page 5897)	Indicates whether the help properties for the action and the client are linked.
	IsHintLinked (see page 271)	Indicates whether the Hint properties for the action and client are linked.
	IsHintLinked (see page 5898)	Indicates whether the Hint property for the action and the client are linked.
	IsImageIndexLinked (see page 272)	Indicates whether the image index for the action and the client are linked.
	IsImagesLinked (see page 15748)	This is IsImagesLinked, a member of class TListActionLink.
	IsOnExecuteLinked (see page 2723)	This is IsOnExecuteLinked, a member of class TBasicActionLink.
	IsOnExecuteLinked (see page 5898)	Indicates whether the OnExecute event handler for the action and the client are linked.
	IsPopupMenuLinked (see page 5898)	This is IsPopupMenuLinked, a member of class TControlActionLink.
	IsShortCutLinked (see page 272)	Indicates whether the ShortCut properties for the action and client are linked.
	IsVisibleLinked (see page 273)	Indicates whether the Visible properties for the action and client are linked.
	IsVisibleLinked (see page 5899)	Indicates whether the visible state for the action and the client are linked.
	RefreshControl (see page 15748)	This is RefreshControl, a member of class TListActionLink.
	SetAction (see page 2724)	This is SetAction, a member of class TBasicActionLink.
	SetAction (see page 15749)	This is SetAction, a member of class TListActionLink.
	SetActive (see page 15749)	This is SetActive, a member of class TListActionLink.
	SetAutoCheck (see page 274)	Introduces an interface for setting the AutoCheck property of a client.
	SetCaption (see page 274)	Introduces an interface for setting the caption on a client.
	SetCaption (see page 5899)	Sets the caption of a control if the client is linked to the action for this event.
	SetChecked (see page 274)	Introduces an interface for setting the checked state of a client.
	SetDropdownMenu (see page 5899)	This is SetDropdownMenu, a member of class TControlActionLink.
	SetEnabled (see page 275)	Introduces an interface for setting the enabled state of a client.
	SetEnabled (see page 5900)	Sets the enabled state of a control if the client is linked to the action for this event.
	SetEnableDropdown (see page 5900)	This is SetEnableDropdown, a member of class TControlActionLink.
	SetGroupIndex (see page 275)	Introduces an interface for setting the GroupIndex property of a client.
	SetHelpContext (see page 276)	Introduces an interface for setting the help context of a client.
	SetHelpContext (see page 5900)	Sets the HelpContext of a control if the client is linked to the action help properties.
	SetHelpContext (see page 6237)	Sets the Help context of a windowed control if the client is linked to the action for this event.
	SetHelpKeyword (see page 276)	Introduces an interface for setting the help keyword of a client.
	SetHelpKeyword (see page 5901)	Sets the HelpKeyword of a control if the client is linked to the action help properties.

	SetHelpType (see page 277)	Introduces an interface for setting the HelpType property of a client.
	SetHelpType (see page 5901)	Sets the HelpType of a control if the client is linked to the action help properties.
	SetHint (see page 278)	Introduces an interface for setting the hint for a client.
	SetHint (see page 5901)	Sets the hint of a control if the client is linked to the action for this event.
	SetImageIndex (see page 278)	Introduces an interface for setting the image index for a client.
	SetImages (see page 15749)	This is SetImages, a member of class TListActionLink.
	SetItemIndex (see page 15749)	This is SetItemIndex, a member of class TListActionLink.
	SetOnExecute (see page 2724)	This is SetOnExecute, a member of class TBasicActionLink.
	SetOnExecute (see page 5902)	Sets the execution event handler for a control if the client is linked to the action for this event.
	SetPopupMenu (see page 5902)	This is SetPopupMenu, a member of class TControlActionLink.
	SetShortCut (see page 279)	Introduces an interface for setting the shortcut for a client.
	SetVisible (see page 279)	Introduces an interface for setting the visible state of a client.
	SetVisible (see page 5902)	Sets the visible state of a control if the client is linked to the action for this event.

Public Events

	Name	Description
	OnChange (see page 2725)	Occurs when an OnChange event is triggered on the action.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2721)	Instantiates and initializes a TBasicActionLink (see page 2720) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2722)	Destroys an instance of an action link object.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 2723)	Invokes the action's Execute method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Update (see page 2724)	Invokes the action's Update method.

Public Properties

	Name	Description
	Action (see page 2725)	Indicates which action the link connects to its client.

27.1.7.2 TComboBoxExActionLink Methods

The methods of the TComboBoxExActionLink class are listed here.

Protected Methods

	Name	Description
	AddItem (see page 3534)	This is the overview for the AddItem method overload.

27.1.7.2.1 AddItem Method

This is the overview for the AddItem method overload.

Overload List

	Name	Description
	TComboBoxExActionLink.AddItem (String, Integer, Pointer) (see page 3534)	This is AddItem, a member of class TComboBoxExActionLink.
	TComboBoxExActionLink.AddItem (TListControlItem) (see page 3534)	This is AddItem, a member of class TComboBoxExActionLink.

27.1.7.2.1.1 TComboBoxExActionLink.AddItem Method (String, Integer, Pointer)

Pascal

```
procedure AddItem(ACaption: String; AImageIndex: Integer; DataPtr: Pointer); override;
```

C++

```
virtual __fastcall AddItem(AnsiString ACaption, int AImageIndex, void * DataPtr);
```

Description

This is AddItem, a member of class TComboBoxExActionLink.

27.1.7.2.1.2 TComboBoxExActionLink.AddItem Method (TListControlItem)

Pascal

```
procedure AddItem(AnItem: TListControlItem); override;
```

C++

```
virtual __fastcall AddItem(TListControlItem AnItem);
```

Description

This is AddItem, a member of class TComboBoxExActionLink.

27.1.8 TComboBoxExStrings Class

Class Hierarchy

System::TObject → Classes.TPersistent → Classes.TStrings → StdCtrls.TCustomComboBoxStrings → ComCtrls.TComboBoxExStrings

Pascal

```
TComboBoxExStrings = class(TCustomComboBoxStrings);
```

C++

```
class TComboBoxExStrings : public TCustomComboBoxStrings;
```

File

ComCtrls

Description

This is class ComCtrls.TComboBoxExStrings.

27.1.8.1 TComboBoxExStrings Members

The following tables list the members exposed by TComboBoxExStrings.

Protected Methods

	Name	Description
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	CompareStrings (see page 3048)	Compares two strings.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 3049)	Reads and writes the Strings (see page 3080) property as if it were published.
	Error (see page 3052)	This is the overview for the Error method overload.
	ExtractName (see page 3053)	Returns the name portion of a string that is a name value pair.
V	GetItemClass (see page 3542)	This is GetItemClass, a member of class TComboBoxExStrings.
V	GetItemsClass (see page 3542)	This is GetItemsClass, a member of class TComboBoxExStrings.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetTextStr (see page 3056)	Returns the value of the Text (see page 3084) property.
V	Put (see page 3064)	Changes the value of the string with a specified index.
V	PutObject (see page 3544)	Changes the object associated with the string at a specified index.
V	SetCapacity (see page 3068)	Changes the amount of memory allocated to hold strings in the list.
V	SetTextStr (see page 3069)	Sets the Text (see page 3084) property.
V	SetUpdateState (see page 3545)	Performs internal adjustments before or after a series of updates.

Protected Properties

	Name	Description
	ComboBox (see page 21232)	Identifies the combo box whose items this object represents.
R	UpdateCount (see page 3085)	Indicates the number of calls to BeginUpdate (see page 3045) that have not been matched by a call to EndUpdate (see page 3050).

Public Methods

	Name	Description
V	Add (see page 3538)	This is Add, a member of class TComboBoxExStrings.
	AddItem (see page 3539)	This is AddItem, a member of class TComboBoxExStrings.

	AddObject (see page 3539)	This is AddObject, a member of class TComboBoxExStrings.
	AddStrings (see page 3039)	Adds a group of strings to the list.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 3041)	Adds a string to the list.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 3043)	Sets the strings in the list, and possibly associated objects, from a source object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 3045)	Enables the TStrings (see page 3029) object to track when the list of strings is changing.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 3539)	Deletes all the strings from the list.
	Create (see page 3537)	This is Create, a member of class TComboBoxExStrings.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 3539)	Removes the item specified by the Index parameter.
	Destroy (see page 3537)	Destroys the TPersistent instance and frees its memory.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 3050)	Enables the TStrings (see page 3029) object to keep track of when the list of strings has finished changing.
	Equals (see page 3051)	Compares the list of strings to the list from another TStrings (see page 3029) object and returns true if the two lists match.
	Exchange (see page 3540)	Swaps the position of two strings in the list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	Get (see page 3540)	Returns a string given its index.
	GetCapacity (see page 3541)	Returns the currently allocated size of the list of strings.
	GetCount (see page 3541)	Returns the number of strings in the list
	GetEnumerator (see page 3054)	Returns a TStrings (see page 3029) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetObject (see page 3542)	Returns the object associated with the item at a specified index.
	GetText (see page 3055)	Allocates a text buffer and fills it with the value of the Text (see page 3084) property.
	IndexOf (see page 3542)	This is IndexOf, a member of class TComboBoxExStrings.
	IndexOfName (see page 3543)	This is IndexOfName, a member of class TComboBoxExStrings.
	IndexOfObject (see page 3059)	Returns the index of the first string in the list associated with a given object.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 3543)	This is Insert, a member of class TComboBoxExStrings.
	InsertObject (see page 3060)	Inserts a string into the list at the specified position, and associates it with an object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LoadFromFile (see page 3061)	Fills the list with the lines of text in a specified file.
	LoadFromStream (see page 3062)	Fills the list with lines of text read from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	Move (see page 3543)	Changes the position of a string in the list.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SaveToFile (see page 3065)	Saves the strings in the list to the specified file.
V	SaveToStream (see page 3067)	Writes the value of the Text (see page 3084) property to a stream object.
V	SetText (see page 3068)	Sets the Text (see page 3084) property.

Public Properties

	Name	Description
	Capacity (see page 3070)	Indicates the number of strings the TStrings (see page 3029) object can hold.
	CommaText (see page 3070)	Lists the strings in the TStrings (see page 3029) object in system data format (SDF).
R	Count (see page 3071)	Introduces an abstract property to represent the number of strings in the list.
	DelimitedText (see page 3074)	Represents all the strings in the TStrings (see page 3029) object as a single delimited string.
	Delimiter (see page 3074)	Specifies the delimiter used by the DelimitedText (see page 3074) property.
	ItemsEx (see page 3545)	This is ItemsEx, a member of class TComboBoxExStrings.
	LineBreak (see page 3075)	Defines the line break character(s).
R	Names (see page 3075)	Indicates the name part of strings that are name-value pairs.
	NameValueSeparator (see page 3077)	Indicates the character used to separate names from values.
	Objects (see page 3077)	Represents a set of objects that are associated one with each of the strings in the Strings (see page 3080) property.
	QuoteChar (see page 3080)	Specifies the quote character used by the DelimitedText (see page 3074) property.
	SortType (see page 3545)	This is SortType, a member of class TComboBoxExStrings.
	StrictDelimiter (see page 3080)	Determines how the Delimiter (see page 3074) property is used.
	Strings (see page 3080)	References the strings in the list by their positions.
	StringsAdapter (see page 3084)	Implements an IStringsAdapter (see page 3208) interface for the TStrings (see page 3029) object.
	Text (see page 3084)	Lists the strings in the TStrings (see page 3029) object as a single string with the individual strings delimited by carriage returns and line feeds.
	ValueFromIndex (see page 3085)	Represents the value part of a string with a given index, on strings that are name-value pairs.
	Values (see page 3086)	Represents the value part of a string associated with a given name, on strings that are name-value pairs.

27.1.8.2 TComboBoxExStrings.Create Constructor

Pascal

```
constructor Create(Owner: TCustomComboBoxEx);
```

C++

```
__fastcall TComboBoxExStrings(TCustomComboBoxEx Owner);
```

Description

This is Create, a member of class TComboBoxExStrings.

27.1.8.3 TComboBoxExStrings.Destroy Destructor

Destroys the TPersistent instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TComboBoxExStrings();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

Free (see page 22300)

Destroy (see page 22287)

27.1.8.4 TComboBoxExStrings Methods

The methods of the TComboBoxExStrings class are listed here.

Protected Methods

	Name	Description
	GetItemClass (see page 3542)	This is GetItemClass, a member of class TComboBoxExStrings.
	GetItemsClass (see page 3542)	This is GetItemsClass, a member of class TComboBoxExStrings.
	PutObject (see page 3544)	Changes the object associated with the string at a specified index.
	SetUpdateState (see page 3545)	Performs internal adjustments before or after a series of updates.

Public Methods

	Name	Description
	Add (see page 3538)	This is Add, a member of class TComboBoxExStrings.
	AddItem (see page 3539)	This is AddItem, a member of class TComboBoxExStrings.
	AddObject (see page 3539)	This is AddObject, a member of class TComboBoxExStrings.
	Clear (see page 3539)	Deletes all the strings from the list.
	Delete (see page 3539)	Removes the item specified by the Index parameter.
	Exchange (see page 3540)	Swaps the position of two strings in the list.
	Get (see page 3540)	Returns a string given its index.
	GetCapacity (see page 3541)	Returns the currently allocated size of the list of strings.
	GetCount (see page 3541)	Returns the number of strings in the list
	GetObject (see page 3542)	Returns the object associated with the item at a specified index.
	IndexOf (see page 3542)	This is IndexOf, a member of class TComboBoxExStrings.
	IndexOfName (see page 3543)	This is IndexOfName, a member of class TComboBoxExStrings.
	Insert (see page 3543)	This is Insert, a member of class TComboBoxExStrings.
	Move (see page 3543)	Changes the position of a string in the list.

27.1.8.4.1 TComboBoxExStrings.Add Method

Pascal

```
function Add(const S: String): Integer; override;
```

C++

```
virtual __fastcall int Add(const AnsiString S);
```

Description

This is Add, a member of class TComboBoxExStrings.

27.1.8.4.2 TComboBoxExStrings.AddItem Method

Pascal

```
function AddItem(const Caption: String; const ImageIndex: Integer; const SelectedImageIndex:
Integer; const OverlayImageIndex: Integer; const Indent: Integer; Data: Pointer): TComboExItem;
```

C++

```
__fastcall TComboExItem AddItem(const AnsiString Caption, const int ImageIndex, const int
SelectedImageIndex, const int OverlayImageIndex, const int Indent, void * Data);
```

Description

This is AddItem, a member of class TComboBoxExStrings.

27.1.8.4.3 TComboBoxExStrings.AddObject Method

Pascal

```
function AddObject(const S: String; AObject: TObject): Integer; override;
```

C++

```
virtual __fastcall int AddObject(const AnsiString S, TObject * AObject);
```

Description

This is AddObject, a member of class TComboBoxExStrings.

27.1.8.4.4 TComboBoxExStrings.Clear Method

Deletes all the strings from the list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call clear to empty the drop-down list of the combo box. All references to associated objects are also removed. However, the objects themselves are not freed.

See Also

Delete (see page 21229)

Objects

Strings

27.1.8.4.5 TComboBoxExStrings.Delete Method

Removes the item specified by the Index parameter.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Call Delete to remove a single item from the list. If an object is associated with the string, the reference to the object is removed as well. Index gives the position of the string in the drop-down list, where 0 is the first string, 1 is the second string, and so on.

See Also

Add (see page 3033)

AddObject (see page 3036)

Clear (see page 21228)

IndexOf (see page 21230)

InsertObject (see page 3060)

Objects

Strings

27.1.8.4.6 TComboBoxExStrings.Exchange Method

Swaps the position of two strings in the list.

Pascal

```
procedure Exchange(Index1: Integer; Index2: Integer); override;
```

C++

```
virtual __fastcall Exchange(int Index1, int Index2);
```

Description

Call Exchange to rearrange the strings in the list. The strings are specified by their index values in the Index1 and Index2 parameters. Indexes are zero-based, so the first string in the list has an index value of 0, the second has an index value of 1, and so on.

If either string has an associated object, Exchange changes the position of the object as well.

See Also

Move (see page 3063)

Objects (see page 3077)

Strings (see page 3080)

27.1.8.4.7 TComboBoxExStrings.Get Method

Returns a string given its index.

Pascal

```
function Get(Index: Integer): String; override;
```

C++

```
virtual __fastcall AnsiString Get(int Index);
```

Description

Get is the protected read implementation of the Strings property. It fetches the specified string from the combo box identified by the ComboBox property.

Index identifies the string to return, where 0 is the index of the first string, 1 is the index of the second string, and so on.

See Also

Strings

IndexOf (see page 21230)

ComboBox (see page 21232)

27.1.8.4.8 TComboBoxExStrings.GetCapacity Method

Returns the currently allocated size of the list of strings.

Pascal

```
function GetCapacity: Integer; override;
```

C++

```
virtual __fastcall int GetCapacity();
```

Description

GetCapacity is the protected read implementation of the Capacity property. In TStrings, GetCapacity returns the value of the Count property. Descendants of TStrings can override this property to let a string list allocate memory for entries that have not been added to the list.

See Also

Capacity (see page 3070)

Count (see page 3071)

27.1.8.4.9 TComboBoxExStrings.GetCount Method

Returns the number of strings in the list

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCount is the protected read implementation of the Count property. It fetches the number of items from the combo box identified by the ComboBox property.

See Also

Count

ComboBox (see page 21232)

27.1.8.4.10 TComboBoxExStrings.GetItemClass Method

Pascal

```
function GetItemClass: TComboExItemClass; virtual;
```

C++

```
virtual __fastcall TComboExItemClass GetItemClass();
```

Description

This is GetItemClass, a member of class TComboBoxExStrings.

27.1.8.4.11 TComboBoxExStrings.GetItemsClass Method

Pascal

```
function GetItemsClass: TComboExItemsClass; virtual;
```

C++

```
virtual __fastcall TComboExItemsClass GetItemsClass();
```

Description

This is GetItemsClass, a member of class TComboBoxExStrings.

27.1.8.4.12 TComboBoxExStrings.GetObject Method

Returns the object associated with the item at a specified index.

Pascal

```
function GetObject(Index: Integer): TObject; override;
```

C++

```
virtual __fastcall TObject * GetObject(int Index);
```

Description

GetObject is the protected read implementation of the Objects property. It fetches the associated object for the identified item from the combo box identified by the ComboBox property.

Index is the index of the item with which the object is associated.

See Also

Objects

ComboBox (see page 21232)

27.1.8.4.13 TComboBoxExStrings.IndexOf Method

Pascal

```
function IndexOf(const S: String): Integer; override;
```

C++

```
virtual __fastcall int IndexOf(const AnsiString S);
```

Description

This is IndexOf, a member of class TComboBoxExStrings.

27.1.8.4.14 TComboBoxExStrings.IndexOfName Method

Pascal

```
function IndexOfName(const Name: String): Integer; override;
```

C++

```
virtual __fastcall int IndexOfName(const AnsiString Name);
```

Description

This is IndexOfName, a member of class TComboBoxExStrings.

27.1.8.4.15 TComboBoxExStrings.Insert Method

Pascal

```
procedure Insert(Index: Integer; const S: String); override;
```

C++

```
virtual __fastcall Insert(int Index, const AnsiString S);
```

Description

This is Insert, a member of class TComboBoxExStrings.

27.1.8.4.16 TComboBoxExStrings.Move Method

Changes the position of a string in the list.

Pascal

```
procedure Move(CurIndex: Integer; NewIndex: Integer); override;
```

C++

```
virtual __fastcall Move(int CurIndex, int NewIndex);
```

Description

Use Move to move the string at position CurIndex so that it occupies the position NewIndex. The positions are specified as 0-based indexes. For example, the following lines of code move the string in the first position to the last position.

```
MyStringsObject.Move(0, MyStringsObject.Count
MyStringsObject->Move(0, MyStringsObject->Count - 1);
```

If the string has an associated object, the object remains associated with the string in its new position.

C++ Examples:

```
/*
This example uses a list box and a button on a form. The
list box contains items when the form appears. When the user
clicks the button, the selected item in the list box is
moved to the top of the list box:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->MultiSelect = false;
  Button1->Caption = "Move to Top";
  for (int i = 1; i <= 10; i++)
    ListBox1->Items->Add("Item " + IntToStr(i));
}
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ListBox1->ItemIndex >= 0 && ListBox1->ItemIndex <= 9)
    ListBox1->Items->Move(ListBox1->ItemIndex, 0);
}
```

Delphi Examples:

```
{
This example uses a list box and a button on a form. The
list box contains items when the form appears. When the user
clicks the button, the selected item in the list box is
moved to the top of the list box:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (ListBox1.ItemIndex in [0..9]) then
    ListBox1.Items.Move(ListBox1.ItemIndex, 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  ListBox1.MultiSelect := False;
  Button1.Caption := 'Move to Top';
  for I := 1 to 10 do
    ListBox1.Items.Add('Item ' + IntToStr(I));
end;
```

See Also

Count (see page 3071)

Objects (see page 3077)

Strings (see page 3080)

27.1.8.4.17 TComboBoxExStrings.PutObject Method

Changes the object associated with the string at a specified index.

Pascal

```
procedure PutObject(Index: Integer; AObject: TObject); override;
```

C++

```
virtual __fastcall PutObject(int Index, TObject * AObject);
```

Description

PutObject is the protected write implementation of the Objects property. It associates the object specified by AObject with the item identified by Index.

Index identifies an item in the drop-down list of the combo box, where 0 is the index of the first item, 1 is the index of the second item, and so on.

AObject is the object to associated with the specified item.

See Also

Objects

27.1.8.4.18 TComboBoxExStrings.SetUpdateState Method

Performs internal adjustments before or after a series of updates.

Pascal

```
procedure SetUpdateState(Updating: Boolean); override;
```

C++

```
virtual __fastcall SetUpdateState(Boolean Updating);
```

Description

SetUpdateState is called at the beginning or end of a series of updates. When the BeginUpdate method is first called and the TCustomComboBoxStrings object is not already in the middle of an update, SetUpdateState is called with Updating set to true. When the EndUpdate method is called and it cancels out the last unmatched call to BeginUpdate, SetUpdateState is called with Updating set to false.

SetUpdateState refreshes the combo box when Updating is false, so that the combo box reflects the new set of strings.

See Also

BeginUpdate (see page 3045)

EndUpdate (see page 3050)

27.1.8.5 TComboBoxExStrings Properties

The properties of the TComboBoxExStrings class are listed here.

Public Properties

	Name	Description
	ItemsEx (see page 3545)	This is ItemsEx, a member of class TComboBoxExStrings.
	SortType (see page 3545)	This is SortType, a member of class TComboBoxExStrings.

27.1.8.5.1 TComboBoxExStrings.ItemsEx Property

Pascal

```
property ItemsEx: TComboExItems;
```

C++

```
__property TComboExItems ItemsEx;
```

Description

This is ItemsEx, a member of class TComboBoxExStrings.

27.1.8.5.2 TComboBoxExStrings.SortType Property

Pascal

```
property SortType: TListItemsSortType;
```

C++

```
__property TListItemsSortType SortType;
```

Description

This is SortType, a member of class TComboBoxExStrings.

27.1.9 TComboExItem Class

TComboExItem represents an item in an extended combo box.

Class Hierarchy

Pascal

```
TComboExItem = class(TListControlItem);
```

C++

```
class TComboExItem : public TListControlItem;
```

File

ComCtrls

Description

TComboExItem represents a single item in the drop-down list of a TComboBoxEx (see page 3463) object. Each TComboBoxEx (see page 3463) item publishes a set of properties that determine its appearance in the list of the combo box. These include

Caption, which is the text that appears in the combo box list.

ImageIndex, which identifies an image to appear beside the caption.

SelectedImageIndex (see page 3552), which identifies an image to appear beside the caption when the item is selected.

OverlayImageIndex (see page 3551), which identifies an overlay bitmap that can mask out portions of the image (or selected image), allowing the background of the item to show through.

Indent (see page 3551), which indicates the number of pixels by which the item is indented in the list.

See Also

TComboExItems (see page 3552)

TComboBoxEx (see page 3463)

27.1.9.1 TComboExItem Members

The following tables list the members exposed by TComboExItem.

Protected Fields

	Name	Description
	FCaption (see page 15752)	This is FCaption, a member of class TListControlItem.
	FData (see page 15752)	This is FData, a member of class TListControlItem.
	FImageIndex (see page 15753)	This is FImageIndex, a member of class TListControlItem.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	Changed (see page 15754)	Responds when any of the item's properties changes.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
	GetDisplayName (see page 15754)	Specifies the name of the item as it appears in the Object Inspector.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCaption (see page 3549)	Sets the value of the Caption property.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetData (see page 3549)	Sets the value of the Data property.
	SetDisplayName (see page 3549)	This is SetDisplayName, a member of class TComboExItem.
	SetImageIndex (see page 3550)	Sets the value of the ImageIndex property.
	SetIndex (see page 3550)	Moves the collection item to a specified position in the parent collection.
	SetOverlayImageIndex (see page 3550)	This is SetOverlayImageIndex, a member of class TComboExItem.
	SetSelectedImageIndex (see page 3551)	This is SetSelectedImageIndex, a member of class TComboExItem.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 3548)	Copies the properties of another TComboExItem (see page 3546) object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2757)	Creates and initializes a TCollectionItem (see page 2755) instance.
	Create (see page 15752)	Creates and initializes a TListControlItem (see page 15750) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	Data (see page 15756)	Points to user-defined data associated with the control.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	Caption (see page 15756)	Specifies the text value of the item.
	ImageIndex (see page 15757)	Identifies the image associated with this item.
	Indent (see page 3551)	Indicate the number of pixels by which the item appears indented in the combo box list.
	OverlayImageIndex (see page 3551)	Identifies an overlay mask that is used to mask out portions of the image and selected image.
	SelectedImageIndex (see page 3552)	Identifies the image associated with this item when it is selected.

27.1.9.2 TComboExItem Methods

The methods of the TComboExItem class are listed here.

Protected Methods

	Name	Description
V	SetCaption (see page 3549)	Sets the value of the Caption property.
V	SetData (see page 3549)	Sets the value of the Data property.
V	SetDisplayName (see page 3549)	This is SetDisplayName, a member of class TComboExItem.
V	SetImageIndex (see page 3550)	Sets the value of the ImageIndex property.
V	SetIndex (see page 3550)	Moves the collection item to a specified position in the parent collection.
V	SetOverlayImageIndex (see page 3550)	This is SetOverlayImageIndex, a member of class TComboExItem.
V	SetSelectedImageIndex (see page 3551)	This is SetSelectedImageIndex, a member of class TComboExItem.

Public Methods

	Name	Description
V	Assign (see page 3548)	Copies the properties of another TComboExItem (see page 3546) object.

27.1.9.2.1 TComboExItem.Assign Method

Copies the properties of another TComboExItem (see page 3546) object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to assign the properties of another combo box item to this one. If the Source parameter is another TComboExItem (

see page 3546) object, Assign copies the values of its Caption, ImageIndex, SelectedImageIndex (see page 3552), OverlayImageIndex (see page 3551), and Indent (see page 3551) properties. Otherwise, Assign calls the inherited method, which copies the properties from any object that has an AssignTo method that can copy to a TComboExItem (see page 3546) object.

See Also

AssignTo (see page 2931)

27.1.9.2.2 TComboExItem.SetCaption Method

Sets the value of the Caption property.

Pascal

```
procedure SetCaption(const Value: String); override;
```

C++

```
virtual __fastcall SetCaption(const AnsiString Value);
```

Description

SetCaption is the protected write implementation of the Caption property. It sets the Caption property to Value, and then calls the Changed method so that the collection can update itself.

See Also

Caption (see page 15756)

Changed (see page 15754)

27.1.9.2.3 TComboExItem.SetData Method

Sets the value of the Data property.

Pascal

```
procedure SetData(const Value: Pointer); override;
```

C++

```
virtual __fastcall SetData(const void * Value);
```

Description

SetData is the protected write implementation of the Data property. It sets the Data property to Value, and then calls the Changed method so that the collection can update itself.

See Also

Data (see page 15756)

Changed (see page 15754)

27.1.9.2.4 TComboExItem.SetDisplayName Method

Pascal

```
procedure SetDisplayName(const Value: String); override;
```

C++

```
virtual __fastcall SetDisplayName(const AnsiString Value);
```

Description

This is SetDisplayName, a member of class TComboExItem.

27.1.9.2.5 TComboExItem.SetImageIndex Method

Sets the value of the ImageIndex property.

Pascal

```
procedure SetImageIndex(const Value: TImageIndex); override;
```

C++

```
virtual __fastcall SetImageIndex(const TImageIndex Value);
```

Description

SetImageIndex is the protected write implementation of the ImageIndex property. It sets the ImageIndex property to Value, and then calls the Changed method so that the collection can update itself.

See Also

ImageIndex (see page 15757)

Changed (see page 15754)

27.1.9.2.6 TComboExItem.SetIndex Method

Moves the collection item to a specified position in the parent collection.

Pascal

```
procedure SetIndex(Value: Integer); override;
```

C++

```
virtual __fastcall SetIndex(int Value);
```

Description

SetIndex is the protected write implementation of the Index property. The Value parameter specifies the new (0-based) position for the item in its parent collection.

See Also

Index (see page 2762)

27.1.9.2.7 TComboExItem.SetOverlayImageIndex Method

Pascal

```
procedure SetOverlayImageIndex(const Value: TImageIndex); virtual;
```

C++

```
virtual __fastcall SetOverlayImageIndex(const TImageIndex Value);
```

Description

This is SetOverlayImageIndex, a member of class TComboExItem.

27.1.9.2.8 TComboExItem.SetSelectedImageIndex Method

Pascal

```
procedure SetSelectedImageIndex(const Value: TImageIndex); virtual;
```

C++

```
virtual __fastcall SetSelectedImageIndex(const TImageIndex Value);
```

Description

This is SetSelectedImageIndex, a member of class TComboExItem.

27.1.9.3 TComboExItem Properties

The properties of the TComboExItem class are listed here.

Published Properties

	Name	Description
	Indent (see page 3551)	Indicate the number of pixels by which the item appears indented in the combo box list.
	OverlayImageIndex (see page 3551)	Identifies an overlay mask that is used to mask out portions of the image and selected image.
	SelectedImageIndex (see page 3552)	Identifies the image associated with this item when it is selected.

27.1.9.3.1 TComboExItem.Indent Property

Indicate the number of pixels by which the item appears indented in the combo box list.

Pascal

```
property Indent: Integer;
```

C++

```
__property int Indent;
```

Description

Use Indent to cause the item to appear indented in the list portion of the combo box. If Indent is –1, the item does not appear indented. If Indent is non-negative, it indicates the number of pixels by which to indent the item.

27.1.9.3.2 TComboExItem.OverlayImageIndex Property

Identifies an overlay mask that is used to mask out portions of the image and selected image.

Pascal

```
property OverlayImageIndex: TImageIndex;
```

C++

```
__property TImageIndex OverlayImageIndex;
```

Description

Set OverlayImageIndex to use an overlay mask with the images that the ImageIndex and SelectedImageIndex (see page 3552) properties identify. An overlay mask is a black-and-white bitmap that is combined with another image so that every place the overlay mask is white appears transparent in the other image.

OverlayImageIndex is an index into the Images property of the combo box. If OverlayImageIndex is -1, this item does not use an

overlay mask.

See Also

Images (see page 3700)

OverlayImageIndex

ImageIndex

27.1.9.3.3 TComboExItem.SelectedImageIndex Property

Identifies the image associated with this item when it is selected.

Pascal

```
property SelectedImageIndex: TImageIndex;
```

C++

```
__property TImageIndex SelectedImageIndex;
```

Description

Set SelectedImageIndex to specify the image that appears beside this item's text when it is selected. SelectedImageIndex is an index into the Images property of the combo box.

When the item is not selected, the combo box displays the image specified by the ImageIndex property instead. If SelectedImageIndex is –1, the combo box also displays the image identified by the ImageIndex property when the item is selected.

Note: To mask out portions of the image that SelectedImageIndex specifies, you can identify an overlay mask using the OverlayImageIndex (see page 3551) property.

See Also

Images (see page 3700)

OverlayImageIndex (see page 3551)

ImageIndex

27.1.10 TComboExItems Class

TComboExItems manages a collection of TComboExItem (see page 3546) objects.

Class Hierarchy

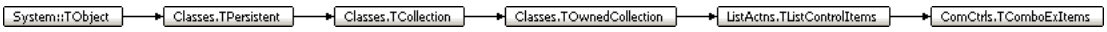

Pascal

```
TComboExItems = class(TListControlItems);
```

C++

```
class TComboExItems : public TListControlItems;
```

File

ComCtrls

Description

TComboExItems manages the collection of items that appear in an extended combo box (TComboBoxEx (see page 3463)).

Use the properties and methods of TComboExItems to:

Access a specific item.

Add (see page 3555) or delete items from the collection.

Find out how many items are in the collection.

Sort the items in the collection.

See Also

TComboExItem (see page 3546)

TComboBoxEx (see page 3463)

27.1.10.1 TComboExItems Members

The following tables list the members exposed by TComboExItems.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	CompareItems (see page 15761)	Returns the relative order of two items in the collection.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2915)	Returns the owner of an object.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Notify (see page 3557)	Responds when items are added to or removed from the collection.
V	SetItem (see page 3557)	This is SetItem, a member of class TComboExItems.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 3555)	Adds a new item to the collection.
	AddItem (see page 3556)	Adds a new item to the collection and initializes its properties.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 2734)	Creates and initializes a collection.
	Create (see page 2914)	Creates and initializes a TOwnedCollection (see page 2912) instance.
	Create (see page 15760)	Creates and initializes a TOwnedCollection instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CustomSort (see page 15761)	Sorts the items in the collection using a specified comparison function.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 3556)	Inserts a new item at a specified position.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Sort (see page 15762)	Sorts the items in the collection.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	ComboItems (see page 3557)	Lists the items in the collection.
R	Count (see page 2749)	Returns the number of items in the collection.

R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 2751)	Lists the items in the collection.
R	Items (see page 15763)	Lists the items in the collection.

Published Events

	Name	Description
	OnCompare (see page 15764)	Occurs to when two items need to be compared for sorting the collection.

Published Properties

	Name	Description
	CaseSensitive (see page 15762)	Controls (see page 5711) whether items are sorted in a case-sensitive or case-insensitive manner.
	SortType (see page 15764)	Indicates how the items should be sorted.

27.1.10.2 TComboExItems Methods

The methods of the TComboExItems class are listed here.

Protected Methods

	Name	Description
V	Notify (see page 3557)	Responds when items are added to or removed from the collection.
V	SetItem (see page 3557)	This is SetItem, a member of class TComboExItems.

Public Methods

	Name	Description
	Add (see page 3555)	Adds a new item to the collection.
	AddItem (see page 3556)	Adds a new item to the collection and initializes its properties.
	Insert (see page 3556)	Inserts a new item at a specified position.

27.1.10.2.1 TComboExItems.Add Method

Adds a new item to the collection.

Pascal

```
function Add: TComboExItem;
```

C++

```
__fastcall TComboExItem Add();
```

Description

Call Add to add a new item to the collection. Add instantiates an object of the type that was specified when the TListControlItems instance was created. It returns the newly created object so that you can specify its properties.

See Also

ComboItems (see page 3557)

Delete (see page 2741)

AddItem (see page 3556)

Insert (see page 3556)

27.1.10.2.2 TComboExItems.AddItem Method

Adds a new item to the collection and initializes its properties.

Pascal

```
function AddItem(const Caption: String; const ImageIndex: Integer; const SelectedImageIndex:
Integer; const OverlayImageIndex: Integer; const Indent: Integer; Data: Pointer): TComboExItem;
```

C++

```
__fastcall TComboExItem AddItem(const AnsiString Caption, const int ImageIndex, const int
SelectedImageIndex, const int OverlayImageIndex, const int Indent, void * Data);
```

Description

Call AddItem to add a new item to the collection. Add (see page 3555) instantiates an object of the type that was specified when the TListControlItems instance was created. It then assigns the values of the Caption, ImageIndex, SelectedImageIndex, OverlayImageIndex, Indent, and Data parameters to the TComboExItem (see page 3546) properties of the same names.

See Also

ComboItems (see page 3557)

Delete (see page 2741)

Create

Add (see page 3555)

Caption

Data

ImageIndex

OverlayImageIndex (see page 3551)

SelectedImageIndex (see page 3552)

Indent (see page 3551)

27.1.10.2.3 TComboExItems.Insert Method

Inserts a new item at a specified position.

Pascal

```
function Insert(Index: Integer): TComboExItem;
```

C++

```
__fastcall TComboExItem Insert(int Index);
```

Description

Call Insert to insert a new item into the collection at a specified position.

Insert instantiates an object of the type that was specified when the TListControlItems instance was created and inserts it into the collection at the position specified by Index. An index of 0 specifies the first position, 1 specifies the second position, and so on.

Insert returns the newly created object so that you can specify its properties.

See Also

ComboItems (see page 3557)

Delete (see page 2741)

Create

Add (see page 3555)

27.1.10.2.4 TComboExItems.Notify Method

Responds when items are added to or removed from the collection.

Pascal

```
procedure Notify(Item: TCollectionItem; Action: TCollectionNotification); override;
```

C++

```
virtual __fastcall Notify(TCollectionItem Item, TCollectionNotification Action);
```

Description

Notify is called automatically when the items in the collection change.

Item is the item that was just added to or that is about to be removed from the collection.

Action indicates whether item was added, is about to be removed, or is about to be deleted:

As implemented in TCollection, Notify calls Added when Action is cnAdded and calls Deleting when Action is cnDeleting. TCollection ignores the cnExtracting action. Descendant classes can override Notify to modify this behavior.

See Also

Added (see page 2736)

Deleting (see page 2741)

27.1.10.2.5 TComboExItems.SetItem Method

Pascal

```
procedure SetItem(const Index: Integer); virtual;
```

C++

```
virtual __fastcall SetItem(const int Index);
```

Description

This is SetItem, a member of class TComboExItems.

27.1.10.3 TComboExItems Properties

The properties of the TComboExItems class are listed here.

Public Properties

	Name	Description
R	ComboItems (see page 3557)	Lists the items in the collection.

27.1.10.3.1 TComboExItems.ComboItems Property

Lists the items in the collection.

Pascal

```
property ComboItems [const Index: Integer]: TComboExItem;
```

C++

```
__property TComboExItem ComboItems[int const Index];
```

Description

Use ComboItems to access the items in this collection.

Index indicates the item to access, where 0 is the index of the first item, 1 is the index of the second item, and so on. The Count property indicates the maximum number of items.

Note: ComboItems is the same as the Items property, except that the type of the property is TComboExItem (see page 3546) instead of TListControlItem, which is the base class for TComboExItem (see page 3546).

See Also

Items (see page 15763)

Count (see page 2749)

27.1.11 TCommonCalendar Class

TCommonCalendar is the base class for the controls TDateTimePicker (see page 4049) and TMonthCalendar (see page 4397).

Class Hierarchy

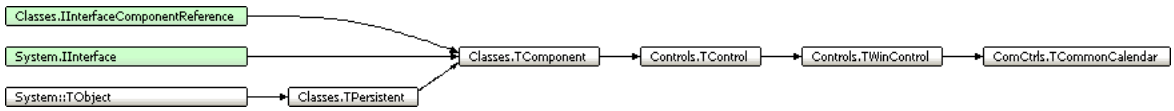

Pascal

```
TCommonCalendar = class(TWinControl);
```

C++

```
class TCommonCalendar : public TWinControl;
```

File

ComCtrls

Description

Use TCommonCalendar as a base class when creating custom controls that represent dates in a calendar-like format. Do not create instances of TCommonCalendar. Instead, to put a calendar control on a form, use a TCommonCalendar descendant such as TDateTimePicker (see page 4049) or TMonthCalendar (see page 4397).

Note: The VCL includes another calendar control that does not descend from TCommonCalendar. This is TCalendar, which is offered as a sample.

27.1.11.1 TCommonCalendar Members

The following tables list the members exposed by TCommonCalendar.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetMonthInfo (see page 3581)	Called whenever a new month is displayed in the calendar.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.

	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
V	CheckEmptyDate (see page 3571)	Clears the check box associated with the calendar.
V	CheckValidDate (see page 3571)	Confirms that a date falls between MinDate (see page 3579) and MaxDate (see page 3578).
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 3572)	Creates the windows control for the calendar.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
A	GetCalendarHandle (see page 3572)	Returns the calendar control's window handle.
V	GetCalStyles (see page 3573)	Returns the style flags that are used when creating the calendar's window.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.

	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	MsgSetCalColors (see page 3573)	Sets the calendar's colors.
	MsgSetDateTime (see page 3574)	Sets the calendar's date/time.
	MsgSetRange (see page 3574)	Implements the MinDate (see page 3579) and MaxDate (see page 3578) properties.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.

	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	CalColors (see page 3575)	Contains the colors used for each part of the calendar.
R	CalendarHandle (see page 3576)	Indicates the calendar control's window handle.
	CalExceptionClass (see page 3576)	Specifies the calendar's exception class.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Date (see page 3576)	Indicates the date that is marked on the calendar.
	DateTime (see page 3577)	Indicates the date that is marked on the calendar.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EndDate (see page 3577)	Indicates the last date that in the range of selected dates.
	FirstDayOfWeek (see page 3578)	Specifies the left-most day on the calendar.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MaxDate (see page 3578)	Indicates the maximum date to which users can scroll the calendar.
	MaxSelectRange (see page 3579)	Specifies the maximum number of days that can be selected.
	MinDate (see page 3579)	Indicates the minimum date that can be selected.
	MonthDelta (see page 3579)	Specifies how many months that the calendar scrolls.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	MultiSelect (see page 3580)	Specifies whether multiple dates can be selected on the calendar.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.

	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ShowToday (see page 3580)	Specifies whether today's date is shown below the calendar.
	ShowTodayCircle (see page 3580)	Specifies whether today's date is circled on the calendar.
	Text (see page 5842)	Contains a text string associated with the control.
	WeekNumbers (see page 3581)	Specifies whether week numbers are shown to the left of the calendar.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BoldDays (see page 3570)	Encodes a set of dates for use in the OnGetMonthInfo (see page 3581) event handler.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 3569)	Creates an instance of TCommonCalendar (see page 3558).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 3570)	Destroys the TCommonCalendar (see page 3558) instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.

	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.

	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.11.2 TCommonCalendar.Create Constructor

Creates an instance of TCommonCalendar (see page 3558).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCommonCalendar(TComponent * AOwner);
```

Description

Call Create to construct and initialize an instance of TCommonCalendar (see page 3558). The AOwner parameter assigns a value to the Owner property, which is responsible for freeing the calendar control.

Create performs the following tasks:

Calls the inherited constructor.

Sets ShowToday (see page 3580) and ShowTodayCircle (see page 3580) to true.

Sets ControlStyle to [csOpaque, csClickEvents, csDoubleClicks].

Sets DateTime (see page 3577) to Now.

Sets FirstDayOfWeek (see page 3578) to dowLocaleDefault.

Sets MaxSelectRange (see page 3579) to 31.

Sets MonthDelta (see page 3579) to 1.

See Also

ShowToday (see page 3580)

ShowTodayCircle (see page 3580)

DateTime (see page 3577)

Now (see page 22930)

FirstDayOfWeek (see page 3578)

MaxSelectRange (see page 3579)

MonthDelta (see page 3579)

ControlStyle (see page 5814)

27.1.11.3 TCommonCalendar.Destroy Destructor

Destroys the TCommonCalendar (see page 3558) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCommonCalendar();
```

Description

Do not call the TCommonCalendar (see page 3558) destructor directly in an application. Instead, call Free, which checks that the calendar reference is not nil before calling Destroy.

Applications do not need to explicitly destroy calendar controls unless they were created without assigning an Owner.

See Also

Free (see page 22300)

27.1.11.4 TCommonCalendar Methods

The methods of the TCommonCalendar class are listed here.

Protected Methods

	Name	Description
	CheckEmptyDate (see page 3571)	Clears the check box associated with the calendar.
	CheckValidDate (see page 3571)	Confirms that a date falls between MinDate (see page 3579) and MaxDate (see page 3578).
	CreateWnd (see page 3572)	Creates the windows control for the calendar.
	GetCalendarHandle (see page 3572)	Returns the calendar control's window handle.
	GetCalStyles (see page 3573)	Returns the style flags that are used when creating the calendar's window.
	MsgSetCalColors (see page 3573)	Sets the calendar's colors.
	MsgSetDateTime (see page 3574)	Sets the calendar's date/time.
	MsgSetRange (see page 3574)	Implements the MinDate (see page 3579) and MaxDate (see page 3578) properties.

Public Methods

	Name	Description
	BoldDays (see page 3570)	Encodes a set of dates for use in the OnGetMonthInfo (see page 3581) event handler.

27.1.11.4.1 TCommonCalendar.BoldDays Method

Encodes a set of dates for use in the OnGetMonthInfo (see page 3581) event handler.

Pascal

```
procedure BoldDays(Days: array of LongWord; var MonthBoldInfo: LongWord);
```

C++

```
__fastcall BoldDays(array of LongWord Days, LongWord MonthBoldInfo);
```

Description

Use BoldDays to encode the days of the current month that should appear in bold. The value returned by BoldDays can be assigned to the MonthBoldInfo parameter of an OnGetMonthInfo (see page 3581) event handler.

Days is an array of unsigned integers corresponding to the days that should be bold.

MonthBoldInfo returns the encoding of the values that were passed in the Days array. This value can be assigned to the MonthBoldInfo parameter of the OnGetMonthInfo (see page 3581) event handler. (or the MonthBoldInfo parameter can be passed in for this argument). You shouldn't work with the MonthBoldInfo variable directly.

For example, to encode the first and eighth days of a month, call

```
BoldDays([1,8], MonthBoldInfo);
unsigned bolddays[2] = {1,2};
BoldDays(bolddays, MonthBoldInfo);
```

See Also

OnGetMonthInfo (see page 3581)

27.1.11.4.2 TCommonCalendar.CheckEmptyDate Method

Clears the check box associated with the calendar.

Pascal

```
procedure CheckEmptyDate; virtual;
```

C++

```
virtual __fastcall CheckEmptyDate();
```

Description

CheckEmptyDate is called automatically when DateTime (see page 3577) is assigned a value that represents a time and not a date value (or Date (see page 3576) is assigned a value of 0). As implemented in TCommonCalendar (see page 3558), CheckEmptyDate does nothing.

TDateTimePicker (see page 4049) overrides CheckEmptyDate to clear the check box associated with the calendar, or raise an EDateTimeError (see page 3421) if the ShowCheckbox property is false. This allows TDateTimePicker (see page 4049) to represent time values that do not include a date portion. (When a TDateTime value represents a time and not a date, the date portion is 0).

See Also

Checked (see page 4073)

ShowCheckbox (see page 4091)

Date (see page 3576)

DateTime (see page 3577)

27.1.11.4.3 TCommonCalendar.CheckValidDate Method

Confirms that a date falls between MinDate (see page 3579) and MaxDate (see page 3578).

Pascal

```
procedure CheckValidDate(Value: TDate); virtual;
```

C++

```
virtual __fastcall CheckValidDate(TDate Value);
```

Description

Calendar controls call CheckValidDate to confirm that a date falls between MinDate (see page 3579) and MaxDate (see page 3578).

Value is the date to check.

If Value is outside MinDate (see page 3579) and MaxDate (see page 3578), an exception of the type specified in the CalExceptionClass (see page 3576) property is raised.

See Also

MinDate (see page 3579)

MaxDate (see page 3578)

CalExceptionClass (see page 3576)

27.1.11.4.4 TCommonCalendar.CreateWnd Method

Creates the windows control for the calendar.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the windowed control is first created or when the window must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the window when it is first created.

CreateWnd calls the TWinControl CreateWnd method to generate the window, and then sends messages to initialize the control so that it reflects the current property settings.

See Also

CreateWnd (see page 6137)

27.1.11.4.5 TCommonCalendar.GetCalendarHandle Method

Returns the calendar control's window handle.

Pascal

```
function GetCalendarHandle: HWND; virtual; abstract;
```

C++

```
virtual __fastcall HWND GetCalendarHandle() = 0;
```

Description

GetCalendarHandle is the protected read implementation of the CalendarHandle (see page 3576) property. For

TCommonCalendar (see page 3558), this is an abstract method. It must be implemented in descendant classes.

See Also

CalendarHandle (see page 3576)

27.1.11.4.6 TCommonCalendar.GetCalStyles Method

Returns the style flags that are used when creating the calendar's window.

Pascal

```
function GetCalStyles: DWORD; virtual;
```

C++

```
virtual __fastcall DWORD GetCalStyles();
```

Description

GetCalStyles is used internally to create the style flags that reflect the current property settings of the calendar control.

See Also

CreateWnd (see page 3572)

TCreateParams (see page 6280)

CreateParams (see page 6135)

27.1.11.4.7 TCommonCalendar.MsgSetCalColors Method

Sets the calendar's colors.

Pascal

```
function MsgSetCalColors(ColorIndex: Integer; ColorValue: TColor): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean MsgSetCalColors(int ColorIndex, TColor ColorValue) = 0;
```

Description

MsgSetCalColors implements the CalColors (see page 3575) property by sending a message to the calendar window handle. MsgSetCalColors is an abstract method, and must be overridden in descendant objects to send the appropriate message.

The ColorIndex parameter indicates which of the TMonthCalColors (see page 4392) properties is being set. It is one of the following values:

Value	Property
0	BackColor
1	TextColor
2	TitleBackColor
3	TitleTextColor
4	MonthBackColor
5	TrailingTextColor

The ColorValue parameter is the value that should be set.

MsgSetCalColors returns true if the color is set by the method.

See Also

CalColors (see page 3575)

CalendarHandle (see page 3576)

TextColor (see page 4395)

BackColor (see page 4395)

TitleBackColor (see page 4396)

TitleTextColor (see page 4396)

MonthBackColor (see page 4395)

TrailingTextColor (see page 4396)

27.1.11.4.8 TCommonCalendar.MsgSetDateTime Method

Sets the calendar's date/time.

Pascal

```
function MsgSetDateTime(Value: TSystemTime): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean MsgSetDateTime(TSystemTime Value) = 0;
```

Description

MsgSetDateTime implements the DateTime (see page 3577) property by sending a message to the calendar window handle. MsgSetDateTime is an abstract method, and must be overridden in descendant objects to send the appropriate message.

The Value parameter indicates the value of DateTime (see page 3577) as a TSystemTime value. This can be obtained from the DateTime (see page 3577) property value by calling DateTimeToSystemTime.

MsgSetDateTime returns true if the date/time value is set by the method.

See Also

DateTime (see page 3577)

CalendarHandle (see page 3576)

DateTimeToSystemTime (see page 22825)

SystemTimeToDateTime (see page 22974)

27.1.11.4.9 TCommonCalendar.MsgSetRange Method

Implements the MinDate (see page 3579) and MaxDate (see page 3578) properties.

Pascal

```
function MsgSetRange(Flags: Integer; SysTime: PSystemTime): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean MsgSetRange(int Flags, PSystemTime SysTime) = 0;
```

Description

MsgSetRange implements the MinDate (see page 3579) and MaxDate (see page 3578) properties by sending a message to the calendar window handle. MsgSetRange is an abstract method, and must be overridden in descendant objects to send the

appropriate message.

The Flags parameter indicates what type of information is included in the SysTime array. It includes the GDTR_MIN flag if a minimum date is supplied and GDTR_MAX if a maximum date is supplied.

The SysTime parameter is a two-element array containing the minimum date as the first element and the maximum date as the second element. These values can be obtained from the MinDate (see page 3579) and MaxDate (see page 3578) properties by calling DateTimeToSystemTime.

MsgSetRange returns true if the range is set by the method.

See Also

MinDate (see page 3579)

MaxDate (see page 3578)

CalendarHandle (see page 3576)

DateTimeToSystemTime (see page 22825)

SystemTimeToDateTime (see page 22974)

27.1.11.5 TCommonCalendar Properties

The properties of the TCommonCalendar class are listed here.

Protected Properties

	Name	Description
	CalColors (see page 3575)	Contains the colors used for each part of the calendar.
R	CalendarHandle (see page 3576)	Indicates the calendar control's window handle.
	CalExceptionClass (see page 3576)	Specifies the calendar's exception class.
	Date (see page 3576)	Indicates the date that is marked on the calendar.
	DateTime (see page 3577)	Indicates the date that is marked on the calendar.
	EndDate (see page 3577)	Indicates the last date that in the range of selected dates.
	FirstDayOfWeek (see page 3578)	Specifies the left-most day on the calendar.
	MaxDate (see page 3578)	Indicates the maximum date to which users can scroll the calendar.
	MaxSelectRange (see page 3579)	Specifies the maximum number of days that can be selected.
	MinDate (see page 3579)	Indicates the minimum date that can be selected.
	MonthDelta (see page 3579)	Specifies how many months that the calendar scrolls.
	MultiSelect (see page 3580)	Specifies whether multiple dates can be selected on the calendar.
	ShowToday (see page 3580)	Specifies whether today's date is shown below the calendar.
	ShowTodayCircle (see page 3580)	Specifies whether today's date is circled on the calendar.
	WeekNumbers (see page 3581)	Specifies whether week numbers are shown to the left of the calendar.

27.1.11.5.1 TCommonCalendar.CalColors Property

Contains the colors used for each part of the calendar.

Pascal

```
property CalColors: TMonthCalColors;
```

C++

```
__property TMonthCalColors CalColors;
```

Description

Use the CalColors property to get or set the calendar's background color, month background color, text color, title background

color, title text color, and trailing text color.

See Also

BoldDays (see page 3570)

MsgSetCalColors (see page 3573)

27.1.11.5.2 TCommonCalendar.CalendarHandle Property

Indicates the calendar control's window handle.

Pascal

```
property CalendarHandle: HWND;
```

C++

```
__property HWND CalendarHandle;
```

Description

Use the CalendarHandle property to get the calendar control's window handle.

The CalendarHandle property is declared but not implemented in TCommonCalendar (see page 3558).

For descendants that use only one window, such as TMonthCalendar (see page 4397), CalendarHandle is the same as the Handle property. However, some TCommonCalendar (see page 3558) descendants such as TDateTimePicker (see page 4049) are implemented using more than one window. In these classes, CalendarHandle is the window handle for the portion of the control that displays date values.

See Also

Handle (see page 6207)

27.1.11.5.3 TCommonCalendar.CalExceptionClass Property

Specifies the calendar's exception class.

Pascal

```
property CalExceptionClass: ExceptClass;
```

C++

```
__property ExceptClass CalExceptionClass;
```

Description

Use CalExceptionClass in a derived class determine the class used for exceptions raised by the calendar control. Descendants of TCommonCalendar (see page 3558) initialize this value in the constructor so that when TCommonCalendar (see page 3558) encounters an unexpected condition, it can raise the appropriate type of exception.

See Also

EMonthCalError (see page 3421)

EDateTimeError (see page 3421)

27.1.11.5.4 TCommonCalendar.Date Property

Indicates the date that is marked on the calendar.

Pascal

```
property Date: TDate;
```

C++

```
__property TDate Date;
```

Description

Use Date to get or set the date that is marked on the calendar. The value of Date must lie within the range specified by the MaxDate (see page 3578) and MinDate (see page 3579) properties.

If MultiSelect (see page 3580) is true, the selected range of dates goes from Date to EndDate (see page 3577).

Note: You can also set the marked date with the DateTime (see page 3577) property.

See Also

DateTime (see page 3577)

EndDate (see page 3577)

MaxDate (see page 3578)

MinDate (see page 3579)

MultiSelect (see page 3580)

27.1.11.5.5 TCommonCalendar.DateTime Property

Indicates the date that is marked on the calendar.

Pascal

```
property DateTime: TDateTime;
```

C++

```
__property TDateTime DateTime;
```

Description

Use DateTime to get or set the date (and, if relevant, time) that is marked on the calendar control. The value of DateTime must lie within the range specified by the MaxDate (see page 3578) and MinDate (see page 3579) properties.

If MultiSelect (see page 3580) is true, the selected range of dates goes from DateTime to EndDate (see page 3577).

Note: You can also set the marked date with the Date (see page 3576) property.

See Also

Date (see page 3576)

EndDate (see page 3577)

MaxDate (see page 3578)

MinDate (see page 3579)

Time (see page 4094)

MultiSelect (see page 3580)

27.1.11.5.6 TCommonCalendar.EndDate Property

Indicates the last date that in the range of selected dates.

Pascal

```
property EndDate: TDate;
```

C++

```
__property TDate EndDate;
```

Description

Use EndDate to get or set the last selected date that is marked on the calendar. The first selected date is given by the Date (see page 3576) property.

Before you can set the EndDate property, you must set the MultiSelect (see page 3580) property to true.

See Also

Date (see page 3576)

MultiSelect (see page 3580)

27.1.11.5.7 TCommonCalendar.FirstDayOfWeek Property

Specifies the left-most day on the calendar.

Pascal

```
property FirstDayOfWeek: TCalDayOfWeek;
```

C++

```
__property TCalDayOfWeek FirstDayOfWeek;
```

Description

Use FirstDayOfWeek to get or set the left-most day on the calendar.

To make the left-most day adjust for the locale, set FirstDayOfWeek to dowLocaleDefault.

See Also

DayOfWeek (see page 22827)

WeekNumbers (see page 3581)

27.1.11.5.8 TCommonCalendar.MaxDate Property

Indicates the maximum date to which users can scroll the calendar.

Pascal

```
property MaxDate: TDate;
```

C++

```
__property TDate MaxDate;
```

Description

Use MaxDate to get or set the maximum date to which users can scroll the calendar. The values of the Date (see page 3576) and EndDate (see page 3577) properties cannot exceed MaxDate.

See Also

Date (see page 3576)

MinDate (see page 3579)

EndDate (see page 3577)

MaxSelectRange (see page 3579)

27.1.11.5.9 TCommonCalendar.MaxSelectRange Property

Specifies the maximum number of days that can be selected.

Pascal

```
property MaxSelectRange: Integer;
```

C++

```
__property int MaxSelectRange;
```

Description

Use MaxSelectRange to limit the number of days that can be selected when MultiSelect (see page 3580) is true. To limit ranges so that they fall between specific date values, use the MaxDate (see page 3578) and MinDate (see page 3579) properties instead.

See Also

Date (see page 3576)

MinDate (see page 3579)

MultiSelect (see page 3580)

EndDate (see page 3577)

MaxDate (see page 3578)

27.1.11.5.10 TCommonCalendar.MinDate Property

Indicates the minimum date that can be selected.

Pascal

```
property MinDate: TDate;
```

C++

```
__property TDate MinDate;
```

Description

Use MinDate to get or set the minimum date that can be selected.

See Also

Date (see page 3576)

MaxDate (see page 3578)

EndDate (see page 3577)

MaxSelectRange (see page 3579)

27.1.11.5.11 TCommonCalendar.MonthDelta Property

Specifies how many months that the calendar scrolls.

Pascal

```
property MonthDelta: Integer;
```

C++

```
__property int MonthDelta;
```

Description

Use MonthDelta to get or set the number of months that the calendar scrolls when the back and forward buttons are clicked.

27.1.11.5.12 TCommonCalendar.MultiSelect Property

Specifies whether multiple dates can be selected on the calendar.

Pascal

```
property MultiSelect: Boolean;
```

C++

```
__property Boolean MultiSelect;
```

Description

Use MultiSelect to specify whether the calendar represents a single date or a range of dates. When MultiSelect is true, the calendar represents the range of dates from Date (see page 3576) to EndDate (see page 3577). When MultiSelect is false, the calendar represents only the value of Date (see page 3576).

See Also

Date (see page 3576)

EndDate (see page 3577)

27.1.11.5.13 TCommonCalendar.ShowToday Property

Specifies whether today's date is shown below the calendar.

Pascal

```
property ShowToday: Boolean;
```

C++

```
__property Boolean ShowToday;
```

Description

Use ShowToday to specify whether today's date (according to the system clock) is shown below the calendar.

See Also

Date (see page 3576)

ShowTodayCircle (see page 3580)

27.1.11.5.14 TCommonCalendar.ShowTodayCircle Property

Specifies whether today's date is circled on the calendar.

Pascal

```
property ShowTodayCircle: Boolean;
```

C++

```
__property Boolean ShowTodayCircle;
```

Description

Use ShowTodayCircle to specify whether today's date (according to the system clock) is circled on the calendar.

See Also

Date (see page 3576)

ShowToday (see page 3580)

27.1.11.5.15 TCommonCalendar.WeekNumbers Property

Specifies whether week numbers are shown to the left of the calendar.

Pascal

```
property WeekNumbers: Boolean;
```

C++

```
__property Boolean WeekNumbers;
```

Description

Use WeekNumbers to indicate whether week numbers are shown to the left of the calendar. The week numbers correspond to the number of the week in the current year.

See Also

FirstDayOfWeek (see page 3578)

27.1.11.6 TCommonCalendar Events

The events of the TCommonCalendar class are listed here.

Protected Events

	Name	Description
	OnGetMonthInfo (see page 3581)	Called whenever a new month is displayed in the calendar.

27.1.11.6.1 TCommonCalendar.OnGetMonthInfo Event

Called whenever a new month is displayed in the calendar.

Pascal

```
property OnGetMonthInfo: TOnGetMonthInfoEvent;
```

C++

```
__property TOnGetMonthInfoEvent OnGetMonthInfo;
```

Description

Use the OnGetMonthInfo event to initialize the display properties of a month. In particular, OnGetMonthInfo allows applications to bold individual days in the calendar (such as holidays).

To specify that specific dates are to be bolded, use the BoldDays (see page 3570) method to obtain a value that can be returned as the MonthBoldInfo parameter. For example, to bold the first and eighth days of every month that is displayed, call

```
BoldDays([1,8], MonthBoldInfo);
```

```
unsigned bolddays[2] = {1,2};
BoldDays(bolddays, MonthBoldInfo);
```

from the OnGetMonthInfo event handler.

See Also

BoldDays (see page 3570)

27.1.12 TConversion Class

TConversion converts rich text data from one encoding format to another.

Class Hierarchy

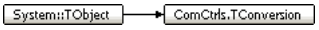

Pascal

```
TConversion = class(TObject);
```

C++

```
class TConversion : public TObject;
```

File

ComCtrls

Description

Use TConversion as the base class for an object that converts rich data from one encoding system to another. TConversion, by itself, does not perform any conversions. It simply reads from or writes to a stream. TConversion provides an interface that can be overridden by derived classes that convert streamed text from one format to another.

Descendants of TCustomRichEdit (see page 3854) use TConversion when streaming text to and from files, when the files contain plain ANSI text or text in rich text format (RTF). To allow rich text edit controls to read from and write to files that use another format, derive a class from TConversion which converts between that format and RTF. Rich text edit controls can use the new class by registering it through their RegisterConversionFormat method, or by making it the default converter.

See Also

TCustomRichEdit (see page 3854)

TCustomRichEdit (see page 3854)

TFileStream (see page 2841)

TStream (see page 2973)

27.1.12.1 TConversion Members

The following tables list the members exposed by TConversion.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	ConvertReadStream (see page 3583)	Reads BufSize bytes from Stream into the memory buffer pointed to by Buffer.
V	ConvertWriteStream (see page 3584)	Writes BufSize bytes to Stream from the memory buffer pointed to by Buffer.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

27.1.12.2 TConversion Methods

The methods of the TConversion class are listed here.

Public Methods

	Name	Description
V	ConvertReadStream (see page 3583)	Reads BufSize bytes from Stream into the memory buffer pointed to by Buffer.
V	ConvertWriteStream (see page 3584)	Writes BufSize bytes to Stream from the memory buffer pointed to by Buffer.

27.1.12.2.1 TConversion.ConvertReadStream Method

Reads BufSize bytes from Stream into the memory buffer pointed to by Buffer.

Pascal

```
function ConvertReadStream(Stream: TStream; Buffer: PChar; BufSize: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int ConvertReadStream(TStream Stream, const char * Buffer, int BufSize);
```

Description

To convert text between two formats, override this method in a descendant class and convert the text in the buffer. To create a descendant class that TCustomRichEdit (see page 3854) can use, convert the text to RTF format.

Note: ConvertReadStream should reverse the conversion that ConvertWriteStream (see page 3584) makes.

See Also

ConvertWriteStream (see page 3584)

27.1.12.2.2 TConversion.ConvertWriteStream Method

Writes BufSize bytes to Stream from the memory buffer pointed to by Buffer.

Pascal

```
function ConvertWriteStream(Stream: TStream; Buffer: PChar; BufSize: Integer): Integer;
virtual;
```

C++

```
virtual __fastcall int ConvertWriteStream(TStream Stream, const char * Buffer, int BufSize);
```

Description

To convert text between two formats, override this method in a descendant class and convert the text in the buffer. To create a descendant class that TCustomRichEdit (see page 3854) can use, convert the text from RTF format.

Note: ConvertWriteStream should reverse the conversion that ConvertReadStream (see page 3583) makes.

See Also

ConvertReadStream (see page 3583)

27.1.13 TCoolBand Class

TCoolBand represents a band within a TCoolBar (see page 3603).

Class Hierarchy

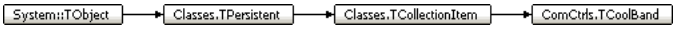

Pascal

```
TCoolBand = class(TCollectionItem);
```

C++

```
class TCoolBand : public TCollectionItem;
```

File

ComCtrls

Description

Use TCoolBand to access one of the resizable bands in a cool bar object. Each TCoolBar (see page 3603) uses a TCoolBands (see page 3596) to maintain a collection of TCoolBand objects.

See Also

TCollection (see page 2731)

TCollectionItem (see page 2755)

TCoolBands (see page 3596)

TCoolBar (see page 3603)

TCoolBar (see page 3603)

27.1.13.1 TCoolBand Members

The following tables list the members exposed by TCoolBand.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 3587)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	ParentBitmapChanged (see page 3588)	This is ParentBitmapChanged, a member of class TCoolBand.
	ParentColorChanged (see page 3588)	This is ParentColorChanged, a member of class TCoolBand.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 3587)	Copies the properties of one band to another.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3586)	Creates and initializes a TCoolBand (see page 3584) instance.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 3586)	Destroys the TCoolBand (see page 3584) and frees its memory.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	Height (see page 3591)	Returns the height of the cool band.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	Bitmap (see page 3589)	Specifies the background image displayed on the band.
	BorderStyle (see page 3589)	Specifies the border style for the band.
	Break (see page 3590)	Causes the band to start on a new line.
	Color (see page 3590)	Specifies the color of the band.
	Control (see page 3590)	Specifies the control that appears on the band.
	FixedBackground (see page 3591)	Determines whether the background is realigned within the band.
	FixedSize (see page 3591)	Determines whether the band can be resized at runtime.
	HorizontalOnly (see page 3592)	Determines whether the band is displayed when the TCoolBar (see page 3603) is arranged vertically.
	ImageIndex (see page 3592)	Determines the image that appears on the band.
	MinHeight (see page 3592)	Determines the minimum height for the band.
	MinWidth (see page 3593)	Determines the minimum width for the band.
	ParentBitmap (see page 3593)	Determines whether the band uses the Bitmap (see page 3589) image of the TCoolBar (see page 3603) that owns it.
	ParentColor (see page 3593)	Determines whether the band uses the color of the TCoolBar (see page 3603) that owns it.
	Text (see page 3594)	Specifies the text string that appears on the band and identifies it in the Bands editor.
	Visible (see page 3595)	Determines whether the band appears at runtime.
	Width (see page 3596)	Specifies the width of the band.

27.1.13.2 TCoolBand.Create Constructor

Creates and initializes a TCoolBand (see page 3584) instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TCoolBand(TCollection Collection);
```

Description

Create generates a TCoolBand (see page 3584) and initializes its properties.

27.1.13.3 TCoolBand.Destroy Destructor

Destroys the TCoolBand (see page 3584) and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCoolBand();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the band is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

27.1.13.4 TCoolBand Methods

The methods of the TCoolBand class are listed here.

Protected Methods

	Name	Description
	GetDisplayName (see page 3587)	Returns the name of the collection item as it appears in the collection editor.
	ParentBitmapChanged (see page 3588)	This is ParentBitmapChanged, a member of class TCoolBand.
	ParentColorChanged (see page 3588)	This is ParentColorChanged, a member of class TCoolBand.

Public Methods

	Name	Description
	Assign (see page 3587)	Copies the properties of one band to another.

27.1.13.4.1 TCoolBand.Assign Method

Copies the properties of one band to another.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Assign copies the properties of Source.

See Also

Text (see page 3594)

Assign (see page 2930)

Width (see page 3596)

27.1.13.4.2 TCoolBand.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName (see page 2761)

GetNamePath (see page 2759)

27.1.13.4.3 TCoolBand.ParentBitmapChanged Method

Pascal

```
procedure ParentBitmapChanged; dynamic;
```

C++

```
__fastcall ParentBitmapChanged();
```

Description

This is ParentBitmapChanged, a member of class TCoolBand.

27.1.13.4.4 TCoolBand.ParentColorChanged Method

Pascal

```
procedure ParentColorChanged; dynamic;
```

C++

```
__fastcall ParentColorChanged();
```

Description

This is ParentColorChanged, a member of class TCoolBand.

27.1.13.5 TCoolBand Properties

The properties of the TCoolBand class are listed here.

Public Properties

	Name	Description
R	Height (see page 3591)	Returns the height of the cool band.

Published Properties

	Name	Description
	Bitmap (see page 3589)	Specifies the background image displayed on the band.
	BorderStyle (see page 3589)	Specifies the border style for the band.
	Break (see page 3590)	Causes the band to start on a new line.
	Color (see page 3590)	Specifies the color of the band.
	Control (see page 3590)	Specifies the control that appears on the band.
	FixedBackground (see page 3591)	Determines whether the background is realigned within the band.
	FixedSize (see page 3591)	Determines whether the band can be resized at runtime.
	HorizontalOnly (see page 3592)	Determines whether the band is displayed when the TCoolBar (see page 3603) is arranged vertically.
	ImageIndex (see page 3592)	Determines the image that appears on the band.
	MinHeight (see page 3592)	Determines the minimum height for the band.
	MinWidth (see page 3593)	Determines the minimum width for the band.

	ParentBitmap (see page 3593)	Determines whether the band uses the Bitmap (see page 3589) image of the TCoolBar (see page 3603) that owns it.
	ParentColor (see page 3593)	Determines whether the band uses the color of the TCoolBar (see page 3603) that owns it.
	Text (see page 3594)	Specifies the text string that appears on the band and identifies it in the Bands editor.
	Visible (see page 3595)	Determines whether the band appears at runtime.
	Width (see page 3596)	Specifies the width of the band.

27.1.13.5.1 TCoolBand.Bitmap Property

Specifies the background image displayed on the band.

Pascal

```
property Bitmap: TBitmap;
```

C++

```
__property TBitmap Bitmap;
```

Description

Use the Bitmap property to select the image that appears on the band. The image is tiled, if necessary, to cover the entire band.

Setting the Bitmap property automatically sets ParentBitmap (see page 3593) to false.

See Also

FixedBackground (see page 3591)

ImageIndex (see page 3592)

ParentBitmap (see page 3593)

TBitmap (see page 14396)

Bitmap (see page 3630)

27.1.13.5.2 TCoolBand.BorderStyle Property

Specifies the border style for the band.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

If BorderStyle is set to bsSingle, the band has a single-line border. If BorderStyle is set to bsNone, no border appears around the band.

Note: If the CoolBand resides in a CoolBar control, the CoolBar's BandBorderStyle property takes precedence over BorderStyle.

See Also

BandBorderStyle (see page 3627)

27.1.13.5.3 TCoolBand.Break Property

Causes the band to start on a new line.

Pascal

```
property Break: Boolean;
```

C++

```
__property Boolean Break;
```

Description

If Break is true (the default), the band starts a new line at the left side of the TCoolBar (see page 3603) control. If Break is false, the band continues on the same line as its predecessor.

27.1.13.5.4 TCoolBand.Color Property

Specifies the color of the band.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use the Color property to select the color of the band. Setting Color automatically sets ParentColor (see page 3593) to false.

See Also

Bitmap (see page 3589)

FixedBackground (see page 3591)

ParentBitmap (see page 3593)

ParentColor (see page 3593)

Bitmap (see page 3630)

27.1.13.5.5 TCoolBand.Control Property

Specifies the control that appears on the band.

Pascal

```
property Control: TWinControl;
```

C++

```
__property TWinControl * Control;
```

Description

The Control property indicates the windowed control that appears on the band. If Control is unassigned, there is no control associated with the band.

27.1.13.5.6 TCoolBand.FixedBackground Property

Determines whether the background is realigned within the band.

Pascal

```
property FixedBackground: Boolean;
```

C++

```
__property Boolean FixedBackground;
```

Description

If ParentBitmap (see page 3593) is true, the band displays the same background as the TCoolBar (see page 3603) that owns it. Ordinarily, when FixedBackground is true, that background is continuous over the entire TCoolBar (see page 3603). Setting FixedBackground to false realigns the background within the individual band, so that the TCoolBar (see page 3603)'s Bitmap (see page 3589) image is positioned relative to the upper left corner of the band.

See Also

Bitmap (see page 3589)

ParentBitmap (see page 3593)

Bitmap (see page 3630)

27.1.13.5.7 TCoolBand.FixedSize Property

Determines whether the band can be resized at runtime.

Pascal

```
property FixedSize: Boolean;
```

C++

```
__property Boolean FixedSize;
```

Description

Setting FixedSize to true prevents the user from resizing the band.

See Also

MinHeight (see page 3592)

MinWidth (see page 3593)

27.1.13.5.8 TCoolBand.Height Property

Returns the height of the cool band.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use Height to get the height of the CoolBand

27.1.13.5.9 TCoolBand.HorizontalOnly Property

Determines whether the band is displayed when the TCoolBar (see page 3603) is arranged vertically.

Pascal

```
property HorizontalOnly: Boolean;
```

C++

```
__property Boolean HorizontalOnly;
```

Description

If the Vertical property of the container CoolBar is true, setting HorizontalOnly to true hides the band. If the Vertical property of the container CoolBar is false, HorizontalOnly has no effect.

See Also

Vertical (see page 3647)

Visible (see page 3595)

27.1.13.5.10 TCoolBand.ImageIndex Property

Determines the image that appears on the band.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

The Images property of TCoolBar (see page 3603) contains a list of bitmaps that can appear to the left of any band in the TCoolBar (see page 3603). To select a bitmap from the list, assign a value to the band's ImageIndex property.

See Also

Bitmap (see page 3589)

Images (see page 3639)

27.1.13.5.11 TCoolBand.MinHeight Property

Determines the minimum height for the band.

Pascal

```
property MinHeight: Integer;
```

C++

```
__property int MinHeight;
```

Description

The MinHeight property determines the smallest height to which the band can be resized at runtime.

See Also

FixedSize (see page 3591)

MinWidth (see page 3593)

27.1.13.5.12 TCoolBand.MinWidth Property

Determines the minimum width for the band.

Pascal

```
property MinWidth: Integer;
```

C++

```
__property int MinWidth;
```

Description

The MinWidth property determines the smallest width to which the band can be resized at runtime.

See Also

FixedSize (see page 3591)

MinHeight (see page 3592)

Width (see page 3596)

27.1.13.5.13 TCoolBand.ParentBitmap Property

Determines whether the band uses the Bitmap (see page 3589) image of the TCoolBar (see page 3603) that owns it.

Pascal

```
property ParentBitmap: Boolean;
```

C++

```
__property Boolean ParentBitmap;
```

Description

If ParentBitmap is set to true, the band displays the same background image as the CoolBar it belongs to. Setting the band's Bitmap (see page 3589) property automatically sets ParentBitmap to false.

See Also

Bitmap (see page 3589)

FixedBackground (see page 3591)

ParentColor (see page 3593)

Bitmap (see page 3630)

27.1.13.5.14 TCoolBand.ParentColor Property

Determines whether the band uses the color of the TCoolBar (see page 3603) that owns it.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

If ParentColor is true, the band has the same color as the CoolBar it belongs to. Setting the Color (see page 3590) property automatically sets ParentColor to false.

See Also

Bitmap (see page 3589)

Color (see page 3590)

ParentBitmap (see page 3593)

Color (see page 5812)

ParentColor (see page 5834)

27.1.13.5.15 TCoolBand.Text Property

Specifies the text string that appears on the band and identifies it in the Bands editor.

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

Assign (see page 3587) a value to Text to create text that appears on the band on the left of the band.

C++ Examples:

```
/*
To run this example, add the example code to a new project.
The example code dynamically creates a TCoolbar and three
TCoolBand objects populated with a windowed control in each
TCoolBand.
*/
void AddBands(TCoolBar *CoolBar, TList *Objects)
{
  TControl *pCurrent;
  for (int i = 0; i < Objects->Count; i++)
  {
    pCurrent = (TControl *)Objects->Items[i];
    // only add windowed controls to the coolbar
    if (dynamic_cast<TWinControl *>(pCurrent) != NULL)
    {
      // Placing the control onto the CoolBar
      // causes the TCoolBar object to create a TCoolBand
      // and place the control within the band.
      pCurrent->Parent = CoolBar;   // This statement increments CoolBar->Bands->Count.
      // Get the reference of the TCoolBand just created and assign text.
      String S = pCurrent->ClassName(pCurrent->ClassType());
      CoolBar->Bands->Items[CoolBar->Bands->Count - 1]->Text = S;
    }
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TCoolBar *CoolBar = new TCoolBar(this);
  TList *List;
  CoolBar->Parent = this;
  CoolBar->Align = alTop;
  List = new TList();
  try
  {
    TControl *Control;
```

```
    Control = new TButton(CoolBar);
    List->Add(Control);
    Control = new TCheckBox(CoolBar);
    List->Add(Control);
    Control = new TEdit(CoolBar);
    List->Add(Control);
    AddBands(CoolBar, List);
  }
  catch (...)
  {
    ShowMessage("Some objects could not be added to Coolband");
  }
  delete List;
}
```

Delphi Examples:

```
{
To run this example, add the example code to a new project.
The example code dynamically creates a TCoolBar and three
TCoolBand objects populated with a windowed control in each
TCoolBand.
}
procedure AddBand(CoolBar: TCoolBar; Const ControlClasses: array of TControlClass);
var
  CurrentControl: TControl;
  I: Integer;
begin
  for I := 0 to High(ControlClasses) do
    begin
      CurrentControl := ControlClasses[i].Create(CoolBar);
      if (CurrentControl is TWinControl) then
      begin
        { Placing the control onto the CoolBar causes the TCoolBar object to create a
TCoolBand and place the control within the band. }
        CurrentControl.Parent := Coolbar;
        { Get the reference of the last TCoolBand created and assign text. }
        with CoolBar.Bands do
          Items[count - 1].Text := CurrentControl.ClassName;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  CoolBar: TCoolBar;
begin
  CoolBar := TCoolBar.Create(Self);
  CoolBar.Parent := Self;
  CoolBar.Align := alClient;
  AddBand(CoolBar, [TCheckBox, TEdit, TButton]);
end;
```

See Also

Control (see page 3590)

27.1.13.5.16 TCoolBand.Visible Property

Determines whether the band appears at runtime.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the band at runtime. If Visible is true, the band appears on the form; if Visible is false, it does not.

27.1.13.5.17 TCoolBand.Width Property

Specifies the width of the band.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

The Width property indicates the width of the band, in pixels.

See Also

MinWidth (see page 3593)

27.1.14 TCoolBands Class

TCoolBands is a container for TCoolBand (see page 3584) objects.

Class Hierarchy

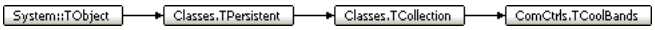

Pascal

```
TCoolBands = class(TCollection);
```

C++

```
class TCoolBands : public TCollection;
```

File

ComCtrls

Description

Each TCoolBands holds a collection of TCoolBand (see page 3584) objects in a TCoolBar (see page 3603). TCoolBands maintains an index of the TCoolBand (see page 3584) objects in its Items (see page 3601) array. The Count property contains the number of bands in the collection.

See Also

TCollection (see page 2731)

TCollectionItem (see page 2755)

TCoolBand (see page 3584)

TCoolBar (see page 3603)

TCoolBar (see page 3603)

27.1.14.1 TCoolBands Members

The following tables list the members exposed by TCoolBands.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 3599)	Returns the owner of an object.
	HaveGraphic (see page 3600)	This is HaveGraphic, a member of class TCoolBands.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 3600)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 3599)	Adds a cool band to the collection.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 3598)	Creates and initializes a TCoolBands (see page 3596) instance.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.

V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindBand (see page 3599)	Returns the band that contains a specified control.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	CoolBar (see page 3601)	Provides access to the TCoolBar (see page 3603) component that owns the TCoolBands (see page 3596).
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 3601)	Lists the bands in the TCoolBands (see page 3596).

27.1.14.2 TCoolBands.Create Constructor

Creates and initializes a TCoolBands (see page 3596) instance.

Pascal

```
constructor Create(CoolBar: TCoolBar);
```

C++

```
__fastcall TCoolBands(TCoolBar CoolBar);
```

Description

Create generates a TCoolBands (see page 3596) instance and initializes its properties.

Typically, applications have no need to instantiate TCoolBands (see page 3596). Instead, this object is created by the TCoolBar (see page 3603) object that uses it to manage its bands.

CoolBar (see page 3601) is the object that owns the TCoolBands (see page 3596) object and uses it to manage its bands.

27.1.14.3 TCoolBands Methods

The methods of the TCoolBands class are listed here.

Protected Methods

	Name	Description
	GetOwner (see page 3599)	Returns the owner of an object.
	HaveGraphic (see page 3600)	This is HaveGraphic, a member of class TCoolBands.
	Update (see page 3600)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 3599)	Adds a cool band to the collection.
	FindBand (see page 3599)	Returns the band that contains a specified control.

27.1.14.3.1 TCoolBands.Add Method

Adds a cool band to the collection.

Pascal

```
function Add: TCoolBand;
```

C++

```
__fastcall TCoolBand Add();
```

Description

Call Add to create an CoolBand in the collection. The new item is placed at the end of the Items (see page 3601) array.

Add returns the new TCoolBand (see page 3584) object.

See Also

Items (see page 3601)

27.1.14.3.2 TCoolBands.FindBand Method

Returns the band that contains a specified control.

Pascal

```
function FindBand(AControl: TControl): TCoolBand;
```

C++

```
__fastcall TCoolBand FindBand(TControl * AControl);
```

Description

Call FindBand to obtain the band containing the specified control.

27.1.14.3.3 TCoolBands.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

27.1.14.3.4 TCoolBands.HaveGraphic Method

Pascal

```
function HaveGraphic: Boolean;
```

C++

```
__fastcall Boolean HaveGraphic();
```

Description

This is HaveGraphic, a member of class TCoolBands.

27.1.14.3.5 TCoolBands.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed (see page 2740)

Changed (see page 2758)

27.1.14.4 TCoolBands Properties

The properties of the TCoolBands class are listed here.

Public Properties

	Name	Description
R	CoolBar (see page 3601)	Provides access to the TCoolBar (see page 3603) component that owns the TCoolBands (see page 3596).
	Items (see page 3601)	Lists the bands in the TCoolBands (see page 3596).

27.1.14.4.1 TCoolBands.CoolBar Property

Provides access to the TCoolBar (see page 3603) component that owns the TCoolBands (see page 3596).

Pascal

```
property CoolBar: TCoolBar;
```

C++

```
__property TCoolBar CoolBar;
```

Description

CoolBar is the TCoolBar (see page 3603) instance to which the TCoolBands (see page 3596) belongs.

See Also

Bands (see page 3628)

27.1.14.4.2 TCoolBands.Items Property

Lists the bands in the TCoolBands (see page 3596).

Pascal

```
property Items [Index: Integer]: TCoolBand;
```

C++

```
__property TCoolBand Items[int Index];
```

Description

Use Items to access individual cool bands. The value of the Index parameter corresponds to the Index property of TCoolBand (see page 3584). It represents the position of the band in the TCoolBands (see page 3596) collection.

C++ Examples:

```
/*
To run this example, add the example code to a new project.
The example code dynamically creates a TCoolbar and three
TCoolBand objects populated with a windowed control in each
TCoolBand.
*/
void AddBands(TCoolBar *CoolBar, TList *Objects)
{
  TControl *pCurrent;
  for (int i = 0; i < Objects->Count; i++)
  {
    pCurrent = (TControl *)Objects->Items[i];
    // only add windowed controls to the coolbar
    if (dynamic_cast<TWinControl *>(pCurrent) != NULL)
```

```
      {
        // Placing the control onto the CoolBar
        // causes the TCoolBar object to create a TCoolBand
        // and place the control within the band.
        pCurrent->Parent = CoolBar;   // This statement increments CoolBar->Bands->Count.
        // Get the reference of the TCoolBand just created and assign text.
        String S = pCurrent->ClassName(pCurrent->ClassType());
        CoolBar->Bands->Items[CoolBar->Bands->Count - 1]->Text = S;
      }
    }
  }

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    TCoolBar *CoolBar = new TCoolBar(this);
    TList *List;
    CoolBar->Parent = this;
    CoolBar->Align = alTop;
    List = new TList();
    try
    {
      TControl *Control;
      Control = new TButton(CoolBar);
      List->Add(Control);
      Control = new TCheckBox(CoolBar);
      List->Add(Control);
      Control = new TEdit(CoolBar);
      List->Add(Control);
      AddBands(CoolBar, List);
    }
    catch (...)
    {
      ShowMessage("Some objects could not be added to Coolband");
    }
    delete List;
  }
```

Delphi Examples:

```
{
To run this example, add the example code to a new project.
The example code dynamically creates a TCoolBar and three
TCoolBand objects populated with a windowed control in each
TCoolBand.
}
procedure AddBand(CoolBar: TCoolBar; Const ControlClasses: array of TControlClass);
var
  CurrentControl: TControl;
  I: Integer;
begin
  for I := 0 to High(ControlClasses) do
    begin
      CurrentControl := ControlClasses[i].Create(CoolBar);
      if (CurrentControl is TWinControl) then
      begin
        { Placing the control onto the CoolBar causes the TCoolBar object to create a
TCoolBand and place the control within the band. }
        CurrentControl.Parent := Coolbar;
        { Get the reference of the last TCoolBand created and assign text. }
        with CoolBar.Bands do
          Items[count - 1].Text := CurrentControl.ClassName;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
```

```
  CoolBar: TCoolBar;
begin
  CoolBar := TCoolBar.Create(Self);
  CoolBar.Parent := Self;
  CoolBar.Align := alClient;
  AddBand(CoolBar, [TCheckBox, TEdit, TButton]);
end;
```

See Also

Items (see page 2751)

Index (see page 2762)

operator_[]

27.1.15 TCoolBar Class

TCoolBar displays a collection of windowed controls within movable, resizable bands.

Class Hierarchy

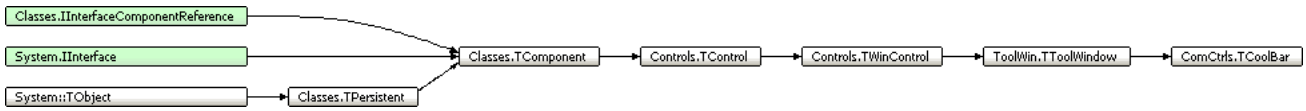

Pascal

```
TCoolBar = class(TToolWindow);
```

C++

```
class TCoolBar : public TToolWindow;
```

File

ComCtrls

Description

TCoolBar is a wrapper class for a Windows rebar control, more commonly known as a CoolBar. A CoolBar contains child controls that can be moved and resized independently. Each control resides on an individual band, represented by a TCoolBand (see page 3584) object listed in the Bands (see page 3628) property. The user positions the controls by dragging the sizing grip to the left of each band.

If its FixedSize (see page 3638) property is true, a CoolBar can contain any kind control. If FixedSize (see page 3638) is false, only wrapper controls (instances of TWinControl descendants) can appear on the CoolBar.

To create a bar that acts as a docking site for tool bars and other controls, use TControlBar instead.

Note: TCoolBar requires version 4.70 or later of COMCTL32.DLL. This file, which usually resides in the WINDOWS\SYSTEM or WINDOWS\SYSTEM32 directory, is required at both design time and runtime. Contact Microsoft for information about deploying COMCTL32.DLL.

See Also

TCoolBand (see page 3584)

TCoolBands (see page 3596)

TControlBar (see page 12316)

Buttons and Similar Controls

Cool Bars (VCL Only)

Adding a Cool Bar Component

Adding a Speed Button to a Panel

Adding a Toolbar Using a Panel Component

Adding a Toolbar Using the Toolbar Component

Adding a Tool Button

Adding Hidden Toolbars

Allowing Toggle Buttons

Allowing Toggled Tool Buttons

Assigning a Menu to a Tool Button

Assigning a Speed Button's Glyph

Assigning Images to Tool Buttons

Creating a Group of Speed Buttons

Creating Groups of Tool Buttons

Designing Toolbars and Cool Bars

Hiding and Showing Toolbars

Responding to Clicks

Setting the Initial Condition of a Speed Button

Setting Tool Button Appearance and Initial Conditions

27.1.15.1 TCoolBar Members

The following tables list the members exposed by TCoolBar.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.

	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 3616)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 3617)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 3617)	This is Change, a member of class TCoolBar.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	CMBorderChanged (see page 23285)	This is CMBorderChanged, a member of class TToolWindow.
	CMCtl3DChanged (see page 23286)	This is CMCtl3DChanged, a member of class TToolWindow.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 3617)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 3618)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.

	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.

	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 3619)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HitTest (see page 3619)	This is HitTest, a member of class TCoolBar.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 3619)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	NCPaint (see page 23286)	This is NCPaint, a member of class TToolWindow.
	Notification (see page 3620)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 3620)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.

	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMNCCalcSize (see page 23286)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCPaint (see page 23286)	This is WMNCPaint, a member of class TToolWindow.
V	WndProc (see page 3621)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 3614)	Creates and initializes a TCoolBar (see page 3603) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 3615)	Destroys the TCoolBar (see page 3603) instance and frees its memory.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
V	FlipChildren (see page 3618)	Flips the control's children.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.

	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.

V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.

	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 3650)	Occurs when the position or size of a band changes.
	OnClick (see page 3650)	Occurs when the user clicks the control.
	OnContextPopup (see page 3652)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 3653)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 3653)	Occurs when another control is docked to the control.
	OnDockOver (see page 3655)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 3657)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 3660)	Occurs when the user drags an object over a control.
	OnEndDock (see page 3662)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 3662)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnGetSiteInfo (see page 3663)	Returns the control's docking information.
	OnMouseActivate (see page 3665)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 3666)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 3669)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 3669)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 3669)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 3672)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 3674)	Occurs immediately after the control is resized.
	OnStartDock (see page 3674)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 3675)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 3676)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 3626)	Determines the alignment of the TCoolBar (see page 3603) component.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 3626)	Specifies how the control is anchored to its parent.
	AutoSize (see page 3627)	Specifies whether the control sizes itself automatically to accommodate its contents.

	BandBorderStyle (see page 3627)	Determines the border style of the bands in the TCoolBar (see page 3603).
	BandMaximize (see page 3628)	Specifies the conditions under which a TCoolBand (see page 3584) is maximized.
	Bands (see page 3628)	Lists the bands (TCoolBands (see page 3596)) of the coolbar.
	Bitmap (see page 3630)	Specifies the image that appears behind the bands in the TCoolBar (see page 3603).
	BorderWidth (see page 3630)	Specifies the width of the control's border.
	Color (see page 3631)	Specifies the background color of the control.
	Constraints (see page 3631)	Specifies the size constraints for the control.
	Ctl3D (see page 3632)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DockSite (see page 3632)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 3633)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 3634)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 3634)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EdgeBorders (see page 3636)	Determines which sides of the control have visible borders.
	EdgeInner (see page 3636)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 3637)	Determines the border style for the control's outer edge.
	Enabled (see page 3637)	Controls whether the control responds to mouse, keyboard, and timer events.
	FixedOrder (see page 3637)	Determines whether the user can reorder the bands in the TCoolBar (see page 3603).
	FixedSize (see page 3638)	Determines whether the bands automatically maintain a uniform height (or width).
	Font (see page 3638)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Images (see page 3639)	Lists images available for bands in the TCoolBar (see page 3603).
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentColor (see page 3640)	Determines where a control looks for its color information.
	ParentFont (see page 3640)	Determines where a control looks for its font information.
	ParentShowHint (see page 3641)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 3641)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 3646)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	ShowText (see page 3647)	Determines whether text is displayed on the bands.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Vertical (see page 3647)	Determines whether the bands are arranged vertically or horizontally.
	Visible (see page 3648)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.15.2 TCoolBar.Create Constructor

Creates and initializes a TCoolBar (see page 3603) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCoolBar(TComponent * AOwner);
```

Description

Call Create to create a cool bar at runtime. Cool bars added at design time are created automatically.

AOwner is the component that is responsible for freeing the cool bar instance. Typically, this is the form. It becomes the value of the Owner property.

Create generates a TCoolBar (see page 3603) object and initializes its properties. If COMCTL32.DLL cannot be found, Create raises the EComponentError exception.

Delphi Examples:

```
{
To run this example, add the example code to a new project.
The example code dynamically creates a TCoolBar and three
TCoolBand objects populated with a windowed control in each
TCoolBand.
}
procedure AddBand(CoolBar: TCoolBar; Const ControlClasses: array of TControlClass);
var
  CurrentControl: TControl;
  I: Integer;
begin
  for I := 0 to High(ControlClasses) do
    begin
      CurrentControl := ControlClasses[i].Create(CoolBar);
      if (CurrentControl is TWinControl) then
      begin
        { Placing the control onto the CoolBar causes the TCoolBar object to create a
TCoolBand and place the control within the band. }
        CurrentControl.Parent := Coolbar;
        { Get the reference of the last TCoolBand created and assign text. }
        with CoolBar.Bands do
          Items[count - 1].Text := CurrentControl.ClassName;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  CoolBar: TCoolBar;
begin
  CoolBar := TCoolBar.Create(Self);
  CoolBar.Parent := Self;
  CoolBar.Align := alClient;
  AddBand(CoolBar, [TCheckBox, TEdit, TButton]);
end;
```

See Also

Owner (see page 2807)

EComponentError (see page 2694)

27.1.15.3 TCoolBar.Destroy Destructor

Destroys the TCoolBar (see page 3603) instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCoolBar();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the coolbar reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

27.1.15.4 TCoolBar Methods

The methods of the TCoolBar class are listed here.

Protected Methods

	Name	Description
	AlignControls (see page 3616)	Aligns any controls for which the control is the parent within a specified area of the control.
	CanAutoSize (see page 3617)	Indicates how the control can be resized.
	Change (see page 3617)	This is Change, a member of class TCoolBar.
	CreateParams (see page 3617)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 3618)	Creates the underlying window.
	GetPalette (see page 3619)	Returns the handle to a palette for use by the control.
	HitTest (see page 3619)	This is HitTest, a member of class TCoolBar.
	Loaded (see page 3619)	Initializes the control after it is loaded from a stream.
	Notification (see page 3620)	Responds to notifications that components are being created or destroyed.
	PaintWindow (see page 3620)	Renders the image of a windowed control.
	WndProc (see page 3621)	Provides specific message responses for the control.

Public Methods

	Name	Description
	FlipChildren (see page 3618)	Flips the control's children.

27.1.15.4.1 TCoolBar.AlignControls Method

Aligns any controls for which the control is the parent within a specified area of the control.

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

Use AlignControls to align all controls within an area of a form, panel, group box, scroll box, or any container control. Specify the area in which to align the controls as the value of the Rect parameter.

AlignControls uses the Align property value for each child control to determine how to align it. The AControl parameter can be nil (Delphi) or NULL (C++). If you specify a control in AControl, that control takes precedence in alignment over other, similarly-aligned controls.

See Also

Align (see page 5798)

27.1.15.4.2 TCoolBar.CanAutoSize Method

Indicates how the control can be resized.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize property is true and an attempt is made to resize the control. It allows controls to implement the AutoSize property before the resize sequence that begins with the OnCanResize and OnConstrainedResize events and ends with the OnResize event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the control. CanAutoSize adjusts these values so that the control's size adjusts to its contents. If the adjustment results in values of NewWidth and NewHeight that differ from the control's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width and Height, CanAutoSize returns false, and the resize sequence is aborted.

As implemented in TWinControl, CanAutoSize returns true, adjusting NewWidth and NewHeight so that all the control's children fit.

See Also

Constraints (see page 5812)

ConstrainedResize (see page 6133)

AdjustSize (see page 6129)

27.1.15.4.3 TCoolBar.Change Method

Pascal

```
procedure Change; dynamic;
```

C++

```
__fastcall Change();
```

Description

This is Change, a member of class TCoolBar.

27.1.15.4.4 TCoolBar.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

27.1.15.4.5 TCoolBar.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams (see page 6135)

CreateWindowHandle (see page 6137)

Perform (see page 5776)

DestroyWnd (see page 6141)

27.1.15.4.6 TCoolBar.FlipChildren Method

Flips the control's children.

Pascal

```
procedure FlipChildren(AllLevels: Boolean); override;
```

C++

```
virtual __fastcall FlipChildren(Boolean AllLevels);
```

Description

FlipChildren overrides the inherited method in order to prevent the flipping of the control bar's children.

Applications call FlipChildren to flip the control's children; that is, to move children on the left side of the control to the right side and vice versa. If an application calls the FlipChildren method of the cool bar's parent, passing in true for the AllLevels parameter, then the cool bar's FlipChildren method is called along with the FlipChildren method of any sibling controls. Because of their special nature, cool bar controls do not respond to the FlipChildren command.

Note: FlipChildren does the same thing as the Flip Children command on the context menu in the IDE.

See Also

FlipChildren (see page 6150)

27.1.15.4.7 TCoolBar.GetPalette Method

Returns the handle to a palette for use by the control.

Pascal

```
function GetPalette: HPALETTE; override;
```

C++

```
virtual __fastcall HPALETTE GetPalette();
```

Description

As implemented in TControl, GetPalette returns a null handle, indicating that the control does not have a palette. Override GetPalette to implement derived objects that use a palette. Create a palette handle for the control by a call to the Windows API function CreatePalette. GetPalette should return the palette handle for the control.

Specifying the palette for a control tells the application that the control's palette needs to be realized and designates the palette to use for realization.

GetPalette is called only if the run-time video mode requires color palette support, for example, in 256-color mode.

See Also

PaletteChanged (see page 5775)

27.1.15.4.8 TCoolBar.HitTest Method

Pascal

```
function HitTest(const Pos: TPoint): TCoolBand;
```

C++

```
__fastcall TCoolBand HitTest(const TPoint Pos);
```

Description

This is HitTest, a member of class TCoolBar.

27.1.15.4.9 TCoolBar.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded (see page 2782)

Action (see page 5797)

ActionChange (see page 5737)

27.1.15.4.10 TCoolBar.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

27.1.15.4.11 TCoolBar.PaintWindow Method

Renders the image of a windowed control.

Pascal

```
procedure PaintWindow(DC: HDC); override;
```

C++

```
virtual __fastcall PaintWindow(HDC DC);
```

Description

Call PaintWindow to repaint the control. PaintWindow sends a WM_PAINT message to the windowed control's DefaultHandler method, setting the message record's WParam field to the value passed in DC and the other parameter and result fields to zeros.

See Also

DefaultHandler (see page 6139)

PaintTo (see page 6167)

DoubleBuffered (see page 6205)

27.1.15.4.12 TCoolBar.WndProc Method

Provides specific message responses for the control.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

Override WndProc to change the initial Windows message handler for the control. The WindowProc property is initialized to point to the WndProc method.

WndProc for TWinControl overrides the inherited method to define responses for focus, mouse, and keyboard messages. It sends all others to its inherited WndProc.

When overriding WndProc to provide specialized responses to messages, call the inherited WndProc at the end to dispatch any unhandled messages.

C++ Examples:

```
/*
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
*/
class TMyListBoxDescendant : public TListBox
{
__published:    // IDE-managed Components
    void __fastcall SubClassWndProc(Messages::TMessage &Message);
    void __fastcall ToggleSubClass(bool On);
    void __fastcall OnDrawItemProc(
      TWinControl *Control, int Index, const TRect &Rect,
      TOwnerDrawState State);
private:    // User declarations
public:     // User declarations
    __fastcall TMyListBoxDescendant(TComponent* Owner);
};

TForm1 *Form1;
TMyListBoxDescendant *MyListBoxDescendant1;
Graphics::TBitmap *bitmap0;

const WM_STYLEMESSAGE = WM_USER + 2000;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
```

```
__fastcall TMyListBoxDescendant::TMyListBoxDescendant(TComponent* Owner)
    : TListBox(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
}

void __fastcall TForm1::SubClassRadioGroup1Click(TObject *Sender)
{
  MyListBoxDescendant1->ToggleSubClass(SubClassRadioGroup1->ItemIndex == 0);
}

void __fastcall TMyListBoxDescendant::SubClassWndProc(Messages::TMessage &Message)
{
  if (Message.Msg == WM_STYLEMESSAGE)
    Style = (TListBoxStyle)Message.WParam;
  else
    WndProc(Message);
}

void __fastcall TMyListBoxDescendant::ToggleSubClass(bool On)
{
  if (On)
    WindowProc = SubClassWndProc;
  else
    WindowProc = WndProc;
}

void __fastcall TMyListBoxDescendant::OnDrawItemProc(TWinControl *Control, int Index,
      const TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *bitmap; // temporary variable for item's bitmap
  int Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *canvas = ((TListBox *)Control)->Canvas;
  canvas->FillRect(Rect); // clear the rectangle
  bitmap = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
  if (bitmap)
  {
    canvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, bitmap->Width, bitmap->Height),
      bitmap, Bounds(0, 0, bitmap->Width, bitmap->Height), clRed); // render bitmap
    Offset += bitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  canvas->TextOut(Rect.Left + Offset, Rect.Top, ((TListBox *)Control)->Items->Strings[Index]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  MyListBoxDescendant1 = new TMyListBoxDescendant(Form1);
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Parent = Form1;
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Left =
    SubClassRadioGroup1->Left + SubClassRadioGroup1->Width + 30;;
  MyListBoxDescendant1->Top = SubClassRadioGroup1->Top;
  MyListBoxDescendant1->Height = SubClassRadioGroup1->Height;
  MyListBoxDescendant1->OnDrawItem =
    MyListBoxDescendant1->OnDrawItemProc;

  bitmap0 = new Graphics::TBitmap;
  ImageList1->GetBitmap(0, bitmap0);
  MyListBoxDescendant1->Items->AddObject("Butterfly", bitmap0);

  SubClassRadioGroup1->Items->Add("SubClassWndProc");
  SubClassRadioGroup1->Items->Add("WndProc");
  SubClassRadioGroup1->ItemIndex = 2;
}
```

Delphi Examples:

```
{
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
}

type
  TMyListBoxDescendant = class(TlistBox)
    procedure SubClassWndProc(var Message: TMessage);
    procedure ToggleSubClass(On: Boolean);
    procedure OnDrawItemProc(
      Control: TWinControl;
      Index: Integer;
      Rect:TRect;
      State: TOwnerDrawState);
  end;
  TForm1 = class(TForm)
    SubClassRadioGroup1: TRadioGroup;
    Button1: TButton;
    ImageList1: TImageList;
    Button2: TButton;
    procedure SubClassRadioGroup1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  MyListBoxDescendant1: TMyListBoxDescendant;
  bitmap0: TBitmap;

implementation

{$R *.dfm}

const WM_STYLEMESSAGE = WM_USER + 2000;
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
end;

procedure TMyListBoxDescendant.SubClassWndProc(var Message: TMessage);
begin
  if (Message.Msg = WM_STYLEMESSAGE) then
    Style:= TListBoxStyle(Message.WParam)
  else
    WndProc(Message);
end;

procedure TMyListBoxDescendant.ToggleSubClass(On: Boolean);
begin
  if On then
    WindowProc := SubClassWndProc
  else
    WindowProc := WndProc;
end;

procedure TForm1.SubClassRadioGroup1Click(Sender: TObject);
begin
  MyListBoxDescendant1.ToggleSubClass(
    SubClassRadioGroup1.ItemIndex = 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyListBoxDescendant1:= TMyListBoxDescendant.Create(self);
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Parent:= Form1;
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Left:=
    SubClassRadioGroup1.Left + SubClassRadioGroup1.Width + 30;;
  MyListBoxDescendant1.Top:= SubClassRadioGroup1.Top;
  MyListBoxDescendant1.Height:= SubClassRadioGroup1.Height;
  MyListBoxDescendant1.OnDrawItem:=
    MyListBoxDescendant1.OnDrawItemProc;

  bitmap0 := TBitmap.Create;
  ImageList1.GetBitmap(0, bitmap0);
  MyListBoxDescendant1.Items.AddObject('Butterfly', bitmap0);

  SubClassRadioGroup1.Items.Add('SubClassWndProc');
  SubClassRadioGroup1.Items.Add('WndProc');
  SubClassRadioGroup1.ItemIndex := 2;
end;

procedure TMyListBoxDescendant.OnDrawItemProc(
  Control: TWinControl;
  Index: Integer;
```

```
    Rect:TRect;
    State: TOwnerDrawState);
  var
    Bitmap: TBitmap;       { temporary variable for the item's bitmap }
    Offset: Integer;       { text offset width }
  begin
    { draw on control canvas, not on the form }
    with (Control as TListBox).Canvas do
    begin
      FillRect(Rect);        { clear the rectangle }
      Offset := 2;           { provide default offset }
      Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
      if Bitmap <> nil then
      begin
        BrushCopy(
          Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
          Bitmap,
          Bounds(0, 0, Bitmap.Width, Bitmap.Height),
          clRed);  {render bitmap}
        Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
      end;
      TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
  text }
    end;
  end;
```

See Also

DefWndProc (see page 6199)

MainWndProc (see page 6165)

WndProc (see page 5791)

WindowProc (see page 5872)

27.1.15.5 TCoolBar Properties

The properties of the TCoolBar class are listed here.

Published Properties

	Name	Description
	Align (see page 3626)	Determines the alignment of the TCoolBar (see page 3603) component.
	Anchors (see page 3626)	Specifies how the control is anchored to its parent.
	AutoSize (see page 3627)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BandBorderStyle (see page 3627)	Determines the border style of the bands in the TCoolBar (see page 3603).
	BandMaximize (see page 3628)	Specifies the conditions under which a TCoolBand (see page 3584) is maximized.
	Bands (see page 3628)	Lists the bands (TCoolBands (see page 3596)) of the coolbar.
	Bitmap (see page 3630)	Specifies the image that appears behind the bands in the TCoolBar (see page 3603).
	BorderWidth (see page 3630)	Specifies the width of the control's border.
	Color (see page 3631)	Specifies the background color of the control.
	Constraints (see page 3631)	Specifies the size constraints for the control.
	Ctl3D (see page 3632)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 3632)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 3633)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 3634)	Specifies whether the control is being dragged normally or for docking.

	DragMode (see page 3634)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EdgeBorders (see page 3636)	Determines which sides of the control have visible borders.
	EdgeInner (see page 3636)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 3637)	Determines the border style for the control's outer edge.
	Enabled (see page 3637)	Controls whether the control responds to mouse, keyboard, and timer events.
	FixedOrder (see page 3637)	Determines whether the user can reorder the bands in the TCoolBar (see page 3603).
	FixedSize (see page 3638)	Determines whether the bands automatically maintain a uniform height (or width).
	Font (see page 3638)	Controls the attributes of text written on or in the control.
	Images (see page 3639)	Lists images available for bands in the TCoolBar (see page 3603).
	ParentColor (see page 3640)	Determines where a control looks for its color information.
	ParentFont (see page 3640)	Determines where a control looks for its font information.
	ParentShowHint (see page 3641)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 3641)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 3646)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	ShowText (see page 3647)	Determines whether text is displayed on the bands.
	Vertical (see page 3647)	Determines whether the bands are arranged vertically or horizontally.
	Visible (see page 3648)	Determines whether the component appears onscreen.

27.1.15.5.1 TCoolBar.Align Property

Determines the alignment of the TCoolBar (see page 3603) component.

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align the CoolBar to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the cool bar changes.

Setting Align to alLeft or alRight automatically sets Vertical (see page 3647) to true (causing bands to be arranged vertically). Setting Align to alTop, alBottom, alClient, or alNone automatically sets Vertical (see page 3647) to false.

See Also

Align (see page 5798)

Vertical (see page 3647)

Anchors (see page 5802)

27.1.15.5.2 TCoolBar.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized.

When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.15.5.3 TCoolBar.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints (see page 5812)

CanAutoSize (see page 5741)

27.1.15.5.4 TCoolBar.BandBorderStyle Property

Determines the border style of the bands in the TCoolBar (see page 3603).

Pascal

```
property BandBorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BandBorderStyle;
```

Description

If BandBorderStyle is bsSingle, each band in the TCoolBar (see page 3603) has a single line around it. If BandBorderStyle is bsNone, the bands have no borders.

Note: BandBorderStyle takes precedence over the BorderStyle property on individual cool bands. Thus, when a CoolBand resides in a cool bar control, its BorderStyle property has no effect.

See Also

BorderStyle (see page 3589)

27.1.15.5.5 TCoolBar.BandMaximize Property

Specifies the conditions under which a TCoolBand (see page 3584) is maximized.

Pascal

```
property BandMaximize: TCoolBandMaximize;
```

C++

```
__property TCoolBandMaximize BandMaximize;
```

Description

Use BandMaximize to determine the user actions that maximize individual bands. When a band is maximized, it takes up as much space as possible in its row or column.

Note: Users can't maximize fixed-size bands.

See Also

OnChange (see page 3650)

FixedSize (see page 3591)

27.1.15.5.6 TCoolBar.Bands Property

Lists the bands (TCoolBands (see page 3596)) of the coolbar.

Pascal

```
property Bands: TCoolBands;
```

C++

```
__property TCoolBands Bands;
```

Description

The Bands property holds a TCoolBands (see page 3596) collection. At design time, you can add, remove, or modify bands with the Bands editor. To open the Bands editor, select the Bands property in the Object Inspector, then double-click in the Value column to the right or click the ellipsis (...) button.

You can also create bands simply by adding new wrapper controls from the palette. When any TWinControl descendant is placed on the form as a child of the TCoolBar (see page 3603), a new band is created for it.

C++ Examples:

```
/*
To run this example, add the example code to a new project.
The example code dynamically creates a TCoolbar and three
TCoolBand objects populated with a windowed control in each
TCoolBand.
*/
void AddBands(TCoolBar *CoolBar, TList *Objects)
{
  TControl *pCurrent;
  for (int i = 0; i < Objects->Count; i++)
  {
    pCurrent = (TControl *)Objects->Items[i];
    // only add windowed controls to the coolbar
```

```
    if (dynamic_cast<TWinControl *>(pCurrent) != NULL)
    {
      // Placing the control onto the CoolBar
      // causes the TCoolBar object to create a TCoolBand
      // and place the control within the band.
      pCurrent->Parent = CoolBar;   // This statement increments CoolBar->Bands->Count.
      // Get the reference of the TCoolBand just created and assign text.
      String S = pCurrent->ClassName(pCurrent->ClassType());
      CoolBar->Bands->Items[CoolBar->Bands->Count - 1]->Text = S;
    }
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TCoolBar *CoolBar = new TCoolBar(this);
  TList *List;
  CoolBar->Parent = this;
  CoolBar->Align = alTop;
  List = new TList();
  try
  {
    TControl *Control;
    Control = new TButton(CoolBar);
    List->Add(Control);
    Control = new TCheckBox(CoolBar);
    List->Add(Control);
    Control = new TEdit(CoolBar);
    List->Add(Control);
    AddBands(CoolBar, List);
  }
  catch (...)
  {
    ShowMessage("Some objects could not be added to Coolband");
  }
  delete List;
}
```

Delphi Examples:

```
{
To run this example, add the example code to a new project.
The example code dynamically creates a TCoolBar and three
TCoolBand objects populated with a windowed control in each
TCoolBand.
}
procedure AddBand(CoolBar: TCoolBar; Const ControlClasses: array of TControlClass);
var
  CurrentControl: TControl;
  I: Integer;
begin
  for I := 0 to High(ControlClasses) do
    begin
      CurrentControl := ControlClasses[i].Create(CoolBar);
      if (CurrentControl is TWinControl) then
      begin
        { Placing the control onto the CoolBar causes the TCoolBar object to create a
TCoolBand and place the control within the band. }
        CurrentControl.Parent := Coolbar;
        { Get the reference of the last TCoolBand created and assign text. }
        with CoolBar.Bands do
          Items[count - 1].Text := CurrentControl.ClassName;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
```

```
var
  CoolBar: TCoolBar;
begin
  CoolBar := TCoolBar.Create(Self);
  CoolBar.Parent := Self;
  CoolBar.Align := alClient;
  AddBand(CoolBar, [TCheckBox, TEdit, TButton]);
end;
```

See Also

TCoolBand (see page 3584)

27.1.15.5.7 TCoolBar.Bitmap Property

Specifies the image that appears behind the bands in the TCoolBar (see page 3603).

Pascal

```
property Bitmap: TBitmap;
```

C++

```
__property TBitmap Bitmap;
```

Description

Bitmap specifies an image that appears behind the bands in the TCoolBar (see page 3603).

By default, the Bitmap image is positioned relative to the upper left corner of the TCoolBar (see page 3603) and covers the client area, and is visible through transparent CoolBands. Setting the FixedBackground, Bitmap, or ParentBitmap properties of the CoolBands can modify this appearance.

See Also

Bitmap (see page 3589)

FixedBackground (see page 3591)

ParentBitmap (see page 3593)

27.1.15.5.8 TCoolBar.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics or text drawn by the control is clipped to the area within the border.

See Also

ClientRect

BevelOuter (see page 6194)

27.1.15.5.9 TCoolBar.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

27.1.15.5.10 TCoolBar.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.15.5.11 TCoolBar.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

27.1.15.5.12 TCoolBar.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite: Boolean;
```

C++

```
__property Boolean DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

27.1.15.5.13 TCoolBar.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

27.1.15.5.14 TCoolBar.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

27.1.15.5.15 TCoolBar.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
```

```
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

27.1.15.5.16 TCoolBar.EdgeBorders Property

Determines which sides of the control have visible borders.

Pascal

```
property EdgeBorders: TEdgeBorders;
```

C++

```
__property TEdgeBorders EdgeBorders;
```

Description

Use the EdgeBorders property to add or remove borders. The appearance of the borders can be set with the EdgeInner and EdgeOuter properties.

See Also

EdgeInner (see page 23287)

EdgeOuter (see page 23288)

27.1.15.5.17 TCoolBar.EdgeInner Property

Determines the border style for the control's inner edge.

Pascal

```
property EdgeInner: TEdgeStyle;
```

C++

```
__property TEdgeStyle EdgeInner;
```

Description

Use EdgeInner to provide the tool window with a three-dimensional beveled look. A tool window has two borders on each side, an outer border drawn next to the outer edge, and an inner border drawn inside the outer border. EdgeInner specifies the appearance of the inner border. It can be raised (esRaised), lowered (esLowered), or left off altogether (esNone).

Note: The EdgeInner property has an effect only on the sides specified by the EdgeBorders property.

Use EdgeInner with EdgeOuter to give the component a three-dimensional look.

See Also

EdgeBorders (see page 23287)

EdgeOuter (see page 23288)

27.1.15.5.18 TCoolBar.EdgeOuter Property

Determines the border style for the control's outer edge.

Pascal

```
property EdgeOuter: TEdgeStyle;
```

C++

```
__property TEdgeStyle EdgeOuter;
```

Description

Use EdgeOuter to provide the tool window with a three-dimensional beveled look. A tool window has two borders on each side, an outer border drawn next to the outer edge, and an inner border drawn inside the outer border. EdgeOuter specifies the appearance of the outer border. It can be raised (esRaised), lowered (esLowered), or left off altogether (esNone).

Note: The EdgeInner property has an effect only on the sides specified by the EdgeBorders property.

Use EdgeOuter with EdgeInner to give the component a three-dimensional look.

See Also

EdgeBorders (see page 23287)

EdgeInner (see page 23287)

27.1.15.5.19 TCoolBar.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.15.5.20 TCoolBar.FixedOrder Property

Determines whether the user can reorder the bands in the TCoolBar (see page 3603).

Pascal

```
property FixedOrder: Boolean;
```

C++

```
__property Boolean FixedOrder;
```

Description

If FixedOrder is set to true, the user can reposition the bands by dragging them with the mouse, but cannot change their order. Bands (see page 3628) can still be reordered from the Collection editor at design time, or by changing their Index properties in program code.

See Also

Index (see page 2762)

27.1.15.5.21 TCoolBar.FixedSize Property

Determines whether the bands automatically maintain a uniform height (or width).

Pascal

```
property FixedSize: Boolean;
```

C++

```
__property Boolean FixedSize;
```

Description

Ordinarily, when Vertical (see page 3647) and FixedSize are both false, the height of each TCoolBand (see page 3584) is determined by the height of its controls or text. When FixedSize is true, the bands are resized so that they all have the same height; increasing the height of the tallest visible band automatically increases the height of all the other bands.

When Vertical (see page 3647) is true, these dependencies are switched. As long as FixedSize is false, the width of each band is determined the width of its controls and text. If FixedSize and Vertical (see page 3647) are both set to true, the bands maintain the same width; increasing the width of the widest visible band automatically increases the width of all the other bands.

See Also

TCoolBand (see page 3584)

Text (see page 3594)

Vertical (see page 3647)

27.1.15.5.22 TCoolBar.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
```

```
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

27.1.15.5.23 TCoolBar.Images Property

Lists images available for bands in the TCoolBar (see page 3603).

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Images is a list of bitmaps that can be displayed to the left of any band in the TCoolBar (see page 3603). Each band can specify an image from this list by setting its ImageIndex property.

See Also

ImageIndex (see page 3592)

27.1.15.5.24 TCoolBar.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

27.1.15.5.25 TCoolBar.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

27.1.15.5.26 TCoolBar.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.15.5.27 TCoolBar.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
```

```
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
```

```
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
```

```
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
```

```
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
```

```
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.15.5.28 TCoolBar.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
```

```
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.15.5.29 TCoolBar.ShowText Property

Determines whether text is displayed on the bands.

Pascal

```
property ShowText: Boolean;
```

C++

```
__property Boolean ShowText;
```

Description

If ShowText is true, the Text string of each TCoolBand (see page 3584) is displayed on the band at runtime. If ShowText is false, only the TWinControl linked to the band is displayed.

See Also

Text (see page 3594)

27.1.15.5.30 TCoolBar.Vertical Property

Determines whether the bands are arranged vertically or horizontally.

Pascal

```
property Vertical: Boolean;
```

C++

```
__property Boolean Vertical;
```

Description

If Vertical is false (the default), the bands in the TCoolBar (see page 3603) are oriented horizontally, from left to right and top to bottom. If Vertical is true, the bands are oriented vertically, from top to bottom and left to right.

See Also

Align (see page 3626)

AutoSize (see page 5803)

FixedSize (see page 3638)

27.1.15.5.31 TCoolBar.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
```

```
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.15.6 TCoolBar Events

The events of the TCoolBar class are listed here.

Published Events

	Name	Description
	OnChange (see page 3650)	Occurs when the position or size of a band changes.
	OnClick (see page 3650)	Occurs when the user clicks the control.

	OnContextPopup (see page 3652)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 3653)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 3653)	Occurs when another control is docked to the control.
	OnDockOver (see page 3655)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 3657)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 3660)	Occurs when the user drags an object over a control.
	OnEndDock (see page 3662)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 3662)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnGetSiteInfo (see page 3663)	Returns the control's docking information.
	OnMouseActivate (see page 3665)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 3666)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 3669)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 3669)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 3669)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 3672)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 3674)	Occurs immediately after the control is resized.
	OnStartDock (see page 3674)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 3675)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 3676)	Occurs when the application tries to undock a control that is docked to the windowed control.

27.1.15.6.1 TCoolBar.OnChange Event

Occurs when the position or size of a band changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

The OnChange event occurs whenever a TCoolBand (see page 3584)'s Break, Index, or Width property is modified at runtime. An OnChange event handler can be used, for example, to record the positions and sizes of the bands in the Windows registry.

See Also

Break (see page 3590)

Index (see page 2762)

Width (see page 3596)

27.1.15.6.2 TCoolBar.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

27.1.15.6.3 TCoolBar.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The

OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.15.6.4 TCoolBar.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

27.1.15.6.5 TCoolBar.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop: TDockDropEvent;
```

C++

```
__property TDockDropEvent OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
```

```
    //OnDockDrop gets called AFTER the client has actually docked,
    //so we check for DockClientCount = 1 before making the dock panel visible.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, True, nil);
    (Sender as TPanel).DockManager.ResetBounds(True);
    //Make DockManager repaints it's clients.
  end;

  procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer; State: TDragState;
    var Accept: Boolean);
  var
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockOver (see page 6216)

27.1.15.6.6 TCoolBar.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver: TDockOverEvent;
```

C++

```
__property TDockOverEvent OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
```

```
    //OnDockDrop gets called AFTER the client has actually docked,
    //so we check for DockClientCount = 1 before making the dock panel visible.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, True, nil);
    (Sender as TPanel).DockManager.ResetBounds(True);
    //Make DockManager repaints it's clients.
  end;

  procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer; State: TDragState;
    var Accept: Boolean);
  var
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnGetSiteInfo (see page 6223)

TDragDockObject (see page 6009)

27.1.15.6.7 TCoolBar.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the

OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;
```

```
procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

27.1.15.6.8 TCoolBar.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
```

```
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.15.6.9 TCoolBar.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.15.6.10 TCoolBar.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

27.1.15.6.11 TCoolBar.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo: TGetSiteInfoEvent;
```

C++

```
__property TGetSiteInfoEvent OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
```

```
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

TDragDockObject (see page 6009)

27.1.15.6.12 TCoolBar.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.15.6.13 TCoolBar.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
```

```
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
```

```
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.15.6.14 TCoolBar.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

27.1.15.6.15 TCoolBar.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

27.1.15.6.16 TCoolBar.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
```

```
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.15.6.17 TCoolBar.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

27.1.15.6.18 TCoolBar.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

27.1.15.6.19 TCoolBar.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

27.1.15.6.20 TCoolBar.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

27.1.15.6.21 TCoolBar.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock: TUnDockEvent;
```

C++

```
__property TUnDockEvent OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
```

```
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
```

```
  shows how set permissions to dock dockable objects onto a
  docking site.
  }

  procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer);
  begin
    //OnDockDrop gets called AFTER the client has actually docked,
    //so we check for DockClientCount = 1 before making the dock panel visible.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, True, nil);
    (Sender as TPanel).DockManager.ResetBounds(True);
    //Make DockManager repaints it's clients.
  end;

  procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer; State: TDragState;
    var Accept: Boolean);
  var
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

27.1.16 TCustomComboBoxEx Class

TCustomComboBoxEx is the base class for combo box components that support extended combo box features such as images on the list entries.

Class Hierarchy

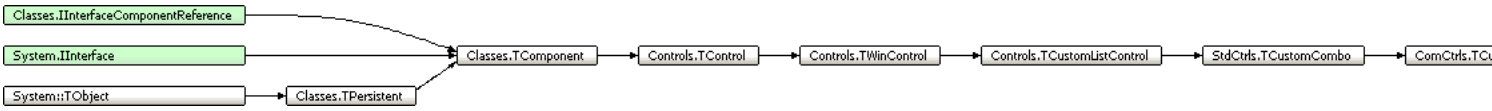

Pascal

```
TCustomComboBoxEx = class(TCustomCombo);
```

C++

```
class TCustomComboBoxEx : public TCustomCombo;
```

File

ComCtrls

Description

Use TCustomComboBoxEx as a base class when defining a control that represents a Windows extended combo box (ComboBoxEx). An extended combo box is an edit box with a scrollable list attached to it that provides native support for item images. Users can select an item from the list or type directly into the edit box. Extended combo boxes can present the list as a drop-down list or as a list box that is always visible.

Note: To create a combo box class that does not use the extended combo box features such as images associated with entries, use the TCustomComboBox class instead.

Do not create instances of TCustomComboBoxEx. To put an extended combo box in a form, use a descendant of TCustomComboBoxEx, such as TComboBoxEx (see page 3463).

See Also

TComboBoxEx (see page 3463)

TCustomActionCombo

TCustomComboBox (see page 21201)

27.1.16.1 TCustomComboBoxEx Members

The following tables list the members exposed by TCustomComboBoxEx.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnChange (see page 21197)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnCloseUp (see page 21199)	Occurs when the drop-down list closes up due to some user action.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.

	OnDropDown (see page 21199)	Occurs when the user opens the drop-down list.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnSelect (see page 21201)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefEditProc (see page 21167)	This is FDefEditProc, a member of class TCustomCombo.
	FDefListProc (see page 21167)	This is FDefListProc, a member of class TCustomCombo.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FDropHandle (see page 21168)	This is FDropHandle, a member of class TCustomCombo.
	FDroppingDown (see page 21168)	This is FDroppingDown, a member of class TCustomCombo.
	FEditHandle (see page 21168)	This is FEditHandle, a member of class TCustomCombo.
	FEditInstance (see page 21168)	This is FEditInstance, a member of class TCustomCombo.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFocusChanged (see page 21168)	This is FFocusChanged, a member of class TCustomCombo.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FIsFocused (see page 21169)	This is FIsFocused, a member of class TCustomCombo.
	FListHandle (see page 21169)	This is FListHandle, a member of class TCustomCombo.
	FListInstance (see page 21169)	This is FListInstance, a member of class TCustomCombo.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

	FSaveIndex (see page 21169)	This is FSaveIndex, a member of class TCustomCombo.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActionChange (see page 3692)	Responds when the action for which this combo box is a client changes.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustDropDown (see page 21171)	Adjusts the size and position of the drop down list.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21171)	Generates an OnChange (see page 21197) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	CloseUp (see page 21172)	Generates an OnCloseUp (see page 21199) event.
	CMColorChanged (see page 3693)	Responds when the Color property changes.
	CMParentColorChanged (see page 3693)	Responds when the ParentColor property changes.
	CNNotify (see page 3693)	Responds to notifications of changes in the combo box.
	ComboExWndProc (see page 3694)	Responds to Windows messages sent to the combo box.
V	ComboWndProc (see page 21173)	Responds to Windows messages sent to the edit control or drop-down list.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 3694)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 3694)	Creates the underlying Windows control that implements the extended combo box.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWindowHandle (see page 21174)	Destroys all of the windows controls that implement the combo box.
	DestroyWnd (see page 3695)	Destroys the Windows control that represents the combo box.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DropDown (see page 21175)	Generates an OnDropDown (see page 21199) event.
	EditWndProc (see page 21175)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 3695)	Returns the action link class that is associated with extended combo boxes.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.

	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetCount (see page 5942)	Returns the number of items in the list.
	GetCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetDroppedDown (see page 21176)	Returns the value of the DroppedDown (see page 21185) property.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetItemCount (see page 3696)	Returns the number of items in the Items property list.
	GetItemHt (see page 3696)	Returns the height, in pixels, allocated for each item in the drop-down list.
	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
	GetItemIndex (see page 21177)	Returns the value of the ItemIndex property.
	GetItemsClass (see page 3696)	Returns the class used to implement the Items property.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelLength (see page 21178)	Returns the value of the SelLength (see page 21190) property.
	GetSelStart (see page 21178)	Returns the value of the SelStart (see page 21193) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	ListWndProc (see page 21178)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	Loaded (see page 21179)	Initializes the combo box after its form is loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 3697)	Responds to notifications that objects are about to be inserted or removed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.

	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Select (see page 21179)	Generates an OnSelect (see page 21201) event.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropDownCount (see page 3697)	Sets the value of the DropDownCount property.
	SetDroppedDown (see page 21180)	Sets the value of the DroppedDown (see page 21185) property.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItemHeight (see page 21181)	Sets the value of the ItemHeight (see page 21186) property.
	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
	SetItemIndex (see page 21181)	Sets the value of the ItemIndex property.
	SetItems (see page 21181)	Sets the value of the Items (see page 21187) property.
	SetMaxLength (see page 21182)	Sets the value of the MaxLength (see page 21189) property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).

	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelLength (see page 21182)	Sets the value of the SelLength (see page 21190) property.
	SetSelStart (see page 21182)	Sets the value of the SelStart (see page 21193) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMHelp (see page 3697)	This is WMHelp, a member of class TCustomComboBoxEx.
	WMLButtonDown (see page 3698)	Responds when the user clicks the left mouse button on the combo box.
V	WndProc (see page 3698)	Processes messages received by the combo box.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
R	EditHandle (see page 21185)	Returns the window handle for the edit portion of the combo box.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.

	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	ItemCount (see page 21186)	Specifies the number of items in the drop-down list.
	ItemHeight (see page 21186)	Specifies the height, in pixels, of the items in the drop-down list.
R	ListHandle (see page 21189)	Provides access to the window handle of the list of items in the combo box.
	MaxLength (see page 21189)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 21190)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnBeginEdit (see page 3702)	Occurs when the user starts editing the text of the combo box.
	OnEndEdit (see page 3702)	Occurs when the user finishes editing the text of the combo box.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
A	AddItem (see page 5940)	Adds an item to the list control.
V	AddItem (see page 21170)	Adds an item to the drop-down list of items.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 5940)	Deletes all of the items from the list control.
V	Clear (see page 21172)	Deletes all text from the edit box and all items from the Items (see page 21187) list.

	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
	ClearSelection (see page 21172)	Deselects any selected items in the drop-down list.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
	CopySelection (see page 21173)	Copies the selected item in the drop-down list to another list control.
	Create (see page 3691)	Creates and initializes a TCustomComboBoxEx (see page 3678) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DeleteSelected (see page 5941)	Deletes all the selected items.
	DeleteSelected (see page 21174)	Deletes the selected item from the drop-down list.
	Destroy (see page 3691)	Destroys the combo box and frees its memory.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 3695)	Indicates whether the combo box has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.

	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
A	SelectAll (see page 5943)	Selects all items or all text in the selected item.
V	SelectAll (see page 21180)	Selects the entire block of text in the edit box.

	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AutoCompleteOptions (see page 3699)	Specifies options that control autocompletion.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 21184)	Provides access to the drawing surface of the combo box.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.

	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropDownCount (see page 3699)	Specifies the maximum number of items displayed in the drop-down list.
	DroppedDown (see page 21185)	Indicates whether the drop-down list is currently displayed.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	Images (see page 3700)	Specifies the images that appear next to items in the combo box list.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	Items (see page 21187)	Provides access to the list of items (strings) in the list portion of the combo box.
	ItemsEx (see page 3700)	Lists the items that appear in the list portion of the combo box.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelLength (see page 21190)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 21193)	Specifies the position of the first selected character in the edit portion of the combo box.
	SelText (see page 3701)	Represents the selected text in the edit region.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	Style (see page 3701)	Determines the display style of the combo box.
	StyleEx (see page 3701)	Determines the appearance and behavior of the items in the list.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TabStop (see page 21196)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic

	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.16.2 TCustomComboBoxEx.Create Constructor

Creates and initializes a TCustomComboBoxEx (see page 3678) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomComboBoxEx(TComponent * AOwner);
```

Description

Call Create to generate an extended combo box at runtime. Combo boxes placed on forms at design time are created automatically.

AOwner is a component, typically the form, that is responsible for freeing the combo box instance.

See Also

Owner (see page 2807)

27.1.16.3 TCustomComboBoxEx.Destroy Destructor

Destroys the combo box and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomComboBoxEx();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free verifies that the combo box is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

27.1.16.4 TCustomComboBoxEx Methods

The methods of the TCustomComboBoxEx class are listed here.

Protected Methods

	Name	Description
	ActionChange (see page 3692)	Responds when the action for which this combo box is a client changes.
	CMColorChanged (see page 3693)	Responds when the Color property changes.
	CMParentColorChanged (see page 3693)	Responds when the ParentColor property changes.
	CNNotify (see page 3693)	Responds to notifications of changes in the combo box.
	ComboExWndProc (see page 3694)	Responds to Windows messages sent to the combo box.
	CreateParams (see page 3694)	Initializes the window-creation parameter record.
	CreateWnd (see page 3694)	Creates the underlying Windows control that implements the extended combo box.
	DestroyWnd (see page 3695)	Destroys the Windows control that represents the combo box.
	GetActionLinkClass (see page 3695)	Returns the action link class that is associated with extended combo boxes.
	GetItemCount (see page 3696)	Returns the number of items in the Items property list.
	GetItemHt (see page 3696)	Returns the height, in pixels, allocated for each item in the drop-down list.
	GetItemsClass (see page 3696)	Returns the class used to implement the Items property.
	Notification (see page 3697)	Responds to notifications that objects are about to be inserted or removed.
	SetDropDownCount (see page 3697)	Sets the value of the DropDownCount property.
	WMHelp (see page 3697)	This is WMHelp, a member of class TCustomComboBoxEx.
	WMLButtonDown (see page 3698)	Responds when the user clicks the left mouse button on the combo box.
	WndProc (see page 3698)	Processes messages received by the combo box.

Public Methods

	Name	Description
	Focused (see page 3695)	Indicates whether the combo box has input focus.

27.1.16.4.1 TCustomComboBoxEx.ActionChange Method

Responds when the action for which this combo box is a client changes.

Pascal

```
procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); override;
```

C++

```
virtual __fastcall ActionChange(TObject * Sender, Boolean CheckDefaults);
```

Description

ActionChange is called automatically when the action object specified by the Action property, or any of its properties, changes.

In TCustomComboBoxEx (see page 3678), ActionChange checks whether the action is a TActionList object, and if so, updates its Images (see page 3700) and ItemIndex properties, as appropriate. It then calls the inherited method, which updates the Caption, Enabled, Hint, and Visible properties and the OnClick event handler.

The Sender parameter is the action object that changed. The CheckDefaults parameter indicates whether the control's properties should all be updated or only those that have not been explicitly changed from their default values. When CheckDefaults is true, only those properties and event handlers that have not been changed from the default values are updated.

See Also

OnChange (see page 2725)

Action (see page 5797)

TActionList (see page 280)

27.1.16.4.2 TCustomComboBoxEx.CMColorChanged Method

Responds when the Color property changes.

Pascal

```
procedure CMColorChanged(var Message: TMessage); message CM_COLORCHANGED;
```

C++

```
__fastcall CMColorChanged(TMessage Message);
```

Description

CMColorChanged is called automatically when the Color property changes. It forces a repaint to reflect the new color.

Message represents the parameters of the Windows message that informs the combo box of the color change.

See Also

Color (see page 5812)

27.1.16.4.3 TCustomComboBoxEx.CMParentColorChanged Method

Responds when the ParentColor property changes.

Pascal

```
procedure CMParentColorChanged(var Message: TMessage); message CM_PARENTCOLORCHANGED;
```

C++

```
__fastcall CMParentColorChanged(TMessage Message);
```

Description

CMParentColorChanged is called automatically when the ParentColor property changes. It forces a repaint to reflect the new color.

Message represents the parameters of the Windows message that informs the combo box of the property change.

See Also

ParentColor (see page 5834)

27.1.16.4.4 TCustomComboBoxEx.CNNotify Method

Responds to notifications of changes in the combo box.

Pascal

```
procedure CNNotify(var Message: TWMNotify); message CN_NOTIFY;
```

C++

```
__fastcall CNNotify(TWMNotify Message);
```

Description

CNNotify is called automatically in response to notification messages from Windows that there has been a change in the combo box. It checks the notification code, and if it is a notification that the user started or stopped editing, generates an OnBeginEdit (see page 3702) or OnEndEdit (see page 3702) event, as appropriate.

Message represents the parameters of the Windows message that informs the combo box of the change.

See Also

OnBeginEdit (see page 3702)

OnEndEdit (see page 3702)

27.1.16.4.5 TCustomComboBoxEx.ComboExWndProc Method

Responds to Windows messages sent to the combo box.

Pascal

```
procedure ComboExWndProc(var Message: TMessage);
```

C++

```
__fastcall ComboExWndProc(TMessage Message);
```

Description

ComboExWndProc is the window procedure of the internal combo box Ex control that is part of the extended combo box. It forwards all messages directed at that control to the ComboWndProc method.

Message represents the parameters of the Windows message.

27.1.16.4.6 TCustomComboBoxEx.CreateParams Method

Initializes the window-creation parameter record.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 3694) method calls CreateParams to initialize the parameters it uses to create the window for the combo box. CreateParams overrides the inherited method to implement the Style (see page 3701) property.

See Also

CreateWnd (see page 3694)

Style (see page 3701)

27.1.16.4.7 TCustomComboBoxEx.CreateWnd Method

Creates the underlying Windows control that implements the extended combo box.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

The CreateHandle method calls CreateWnd to perform the task of creating the underlying Windows control.

The Windows control for the combo box can be temporarily destroyed and recreated in order to set properties that require

changing the window creation parameters. Thus, CreateWnd may be called several times during the lifetime of the TCustomComboBoxEx (see page 3678) instance.

See Also

CreateHandle (see page 6135)

27.1.16.4.8 TCustomComboBoxEx.DestroyWnd Method

Destroys the Windows control that represents the combo box.

Pascal

```
procedure DestroyWnd; override;
```

C++

```
virtual __fastcall DestroyWnd();
```

Description

DestroyWnd is called when the underlying Windows control needs to be destroyed. In addition to freeing this control when the TCustomComboBoxEx (see page 3678) object is destroyed, it can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters.

DestroyWnd saves the contents of Items list before calling the inherited DestroyWnd method. The saved list is used if the window is recreated.

See Also

DestroyWnd (see page 6141)

CreateWnd (see page 3694)

27.1.16.4.9 TCustomComboBoxEx.Focused Method

Indicates whether the combo box has input focus.

Pascal

```
function Focused: Boolean; override;
```

C++

```
virtual __fastcall Boolean Focused();
```

Description

Call Focused to determine whether the combo box is currently the recipient of keyboard input. When Focused returns true, the combo box is the active control and keystroke messages are sent to it. When Focused is false, the combo box is not active (although it may be selected).

See Also

ActiveControl (see page 13896)

27.1.16.4.10 TCustomComboBoxEx.GetActionLinkClass Method

Returns the action link class that is associated with extended combo boxes.

Pascal

```
function GetActionLinkClass: TControlActionLinkClass; override;
```

C++

```
virtual __fastcall TControlActionLinkClass GetActionLinkClass();
```

Description

GetActionLinkClass returns the class type for the action link that manages communication between the combo box and the action specified by the Action property. Each Action link class is designed to link specific properties and event handlers of the action to its client, based on the types of properties the client supports.

See Also

Action (see page 5797)

27.1.16.4.11 TCustomComboBoxEx.GetItemCount Method

Returns the number of items in the Items property list.

Pascal

```
function GetItemCount: Integer; override;
```

C++

```
virtual __fastcall int GetItemCount();
```

Description

GetItemCount is used internally to quickly determine the number of items in the drop-down list of the combo box.

See Also

Items (see page 21187)

ItemsEx (see page 3700)

27.1.16.4.12 TCustomComboBoxEx.GetItemHt Method

Returns the height, in pixels, allocated for each item in the drop-down list.

Pascal

```
function GetItemHt: Integer; override;
```

C++

```
virtual __fastcall int GetItemHt();
```

Description

Extended combo boxes do not support variable-height items. For this reason, TCustomComboBoxEx (see page 3678) overrides the GetItemHt method so that it always returns 16.

See Also

ItemHeight

27.1.16.4.13 TCustomComboBoxEx.GetItemsClass Method

Returns the class used to implement the Items property.

Pascal

```
function GetItemsClass: TCustomComboBoxStringsClass; override;
```

C++

```
virtual __fastcall TCustomComboBoxStringsClass GetItemsClass();
```

Description

Combo box controls use GetItemsClass internally to determine what class to use when implementing the Items property. TCustomComboBoxEx (see page 3678) overrides the inherited method to specify a class that can maintain the relationship between the Items property and the ItemsEx (see page 3700) property.

See Also

Items (see page 21187)

ItemsEx (see page 3700)

27.1.16.4.14 TCustomComboBoxEx.Notification Method

Responds to notifications that objects are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called when the component specified by AComponent is about to be inserted or removed, as specified by Operation. TCustomComboBoxEx (see page 3678) overrides this method to respond to notifications that the image list assigned to the Images (see page 3700) property is going away.

See Also

Images (see page 3700)

27.1.16.4.15 TCustomComboBoxEx.SetDropDownCount Method

Sets the value of the DropDownCount property.

Pascal

```
procedure SetDropDownCount(const Value: Integer); override;
```

C++

```
virtual __fastcall SetDropDownCount(const int Value);
```

Description

SetDropDownCount is the protected write implementation of the DropDownCount property.

See Also

DropDownCount (see page 21185)

27.1.16.4.16 TCustomComboBoxEx.WMHelp Method

Pascal

```
procedure WMHelp(var Message: TWMHelp); message WM_HELP;
```

C++

```
__fastcall WMHelp(TWMHelp Message);
```

Description

This is WMHelp, a member of class TCustomComboBoxEx.

27.1.16.4.17 TCustomComboBoxEx.WMLButtonDown Method

Responds when the user clicks the left mouse button on the combo box.

Pascal

```
procedure WMLButtonDown(var Message: TWMLButtonDown); message WM_LBUTTONDOWN;
```

C++

```
__fastcall WMLButtonDown(TWMLButtonDown Message);
```

Description

WMLButtonDown is called automatically in response to Windows messages that the user clicked the left mouse button over the combo box. This method handles the details of starting drag operations when the DragMode property is dmAutomatic.

Message represents the parameters of the Windows message that informs the combo box of the left mouse click.

See Also

DragMode (see page 5819)

27.1.16.4.18 TCustomComboBoxEx.WndProc Method

Processes messages received by the combo box.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc is the first method to received Windows messages that are sent to the combo box. Messages (see page 16211) sent to the child windows of the combo box (for the edit region and the list portion) go to the EditWndProc or ListWndProc method instead. WndProc handles messages about changes to the system colors defined in the Windows control panel, and closes up the drop-down list when the Enter key or the Escape key is pressed. All other messages are passed to Dispatch for default processing.

See Also

ComboExWndProc (see page 3694)

Dispatch

WndProc (see page 6187)

EditWndProc (see page 21175)

ListWndProc (see page 21178)

27.1.16.5 TCustomComboBoxEx Properties

The properties of the TCustomComboBoxEx class are listed here.

Public Properties

	Name	Description
	AutoCompleteOptions (see page 3699)	Specifies options that control autocompletion.
	DropDownCount (see page 3699)	Specifies the maximum number of items displayed in the drop-down list.
	Images (see page 3700)	Specifies the images that appear next to items in the combo box list.
	ItemsEx (see page 3700)	Lists the items that appear in the list portion of the combo box.
	SelText (see page 3701)	Represents the selected text in the edit region.
	Style (see page 3701)	Determines the display style of the combo box.
	StyleEx (see page 3701)	Determines the appearance and behavior of the items in the list.

27.1.16.5.1 TCustomComboBoxEx.AutoCompleteOptions Property

Specifies options that control autocompletion.

Pascal

```
property AutoCompleteOptions: TAutoCompleteOptions;
```

C++

```
__property TAutoCompleteOptions AutoCompleteOptions;
```

Description

AutoCompleteOptions specifies options that control autocompletion.

27.1.16.5.2 TCustomComboBoxEx.DropDownCount Property

Specifies the maximum number of items displayed in the drop-down list.

Pascal

```
property DropDownCount: Integer;
```

C++

```
__property int DropDownCount;
```

Description

By default, the drop-down list is long enough to contain eight items without requiring the user to scroll to see them all. To make the drop-down list smaller or larger, specify a number larger or smaller than eight as the DropDownCount value.

If the DropDownCount value is larger than the number of items in the Items property, the drop-down list will be just large enough to hold all the possible choices and no more. If the DropDownCount value is smaller then the number of item in the Items property, the drop down list will display a scroll bar.

DropDownCount is meaningful only if Style (see page 3701) is not csExSimple.

See Also

ItemsEx (see page 3700)

Style (see page 3701)

27.1.16.5.3 TCustomComboBoxEx.Images Property

Specifies the images that appear next to items in the combo box list.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Use Images to supply the list of images that appear beside items in the combo box's drop-down list. Each combo box item has an ImageIndex property that identifies an image in this list. In addition, Images can contain images that are used as overlay masks for the item images (to create non-rectangular images) and images that are used only when items are selected. The combo box items reference these images using their OverlayImageIndex and SelectedImageIndex properties.

See Also

ImageIndex

ItemsEx (see page 3700)

SelectedImageIndex (see page 3552)

OverlayImageIndex (see page 3551)

27.1.16.5.4 TCustomComboBoxEx.ItemsEx Property

Lists the items that appear in the list portion of the combo box.

Pascal

```
property ItemsEx: TComboExItems;
```

C++

```
__property TComboExItems ItemsEx;
```

Description

Read Items to access the list of items that appears in the combo box. ItemsEx is a collection of TComboExItem (see page 3546) objects.

Set the value of Items to copy the items from another collection of TComboExItem (see page 3546) objects.

Use the methods of Items to add, insert, delete and move items. Use the properties of individual items in the collection to specify the appearance of each item.

Note: To work with only the strings in the combo box, and not the other properties of the items, you can use the Items property instead.

See Also

ItemIndex

TComboExItem (see page 3546)

Items (see page 21187)

27.1.16.5.5 TCustomComboBoxEx.SelText Property

Represents the selected text in the edit region.

Pascal

```
property SelText: string;
```

C++

```
__property AnsiString SelText;
```

Description

Read SelText to obtain the text that is selected in the edit region of the combo box. When the combo box is the active control, SelText appears highlighted. Set SelText to replace the current selection with a different substring. If no text is selected when a value is assigned to SelText, the SelText string is inserted in the text at the cursor.

See Also

SelLength (see page 21190)

SelStart (see page 21193)

27.1.16.5.6 TCustomComboBoxEx.Style Property

Determines the display style of the combo box.

Pascal

```
property Style: TComboBoxExStyle;
```

C++

```
__property TComboBoxExStyle Style;
```

Description

Set Style to specify how the list and edit region of the combo box should appear.

See Also

Height (see page 5825)

DropDownCount (see page 3699)

27.1.16.5.7 TCustomComboBoxEx.StyleEx Property

Determines the appearance and behavior of the items in the list.

Pascal

```
property StyleEx: TComboBoxExStyles;
```

C++

```
__property TComboBoxExStyles StyleEx;
```

Description

Set StyleEx to configure how the items in the list look and behave.

See Also

Indent

ImageIndex

ItemsEx (see page 3700)

27.1.16.6 TCustomComboBoxEx Events

The events of the TCustomComboBoxEx class are listed here.

Public Events

	Name	Description
	OnBeginEdit (see page 3702)	Occurs when the user starts editing the text of the combo box.
	OnEndEdit (see page 3702)	Occurs when the user finishes editing the text of the combo box.

27.1.16.6.1 TCustomComboBoxEx.OnBeginEdit Event

Occurs when the user starts editing the text of the combo box.

Pascal

```
property OnBeginEdit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnBeginEdit;
```

Description

Write an OnBeginEdit event handler to take specific action immediately before the user edits the text in the combo box.

See Also

OnEndEdit (see page 3702)

Text (see page 5842)

27.1.16.6.2 TCustomComboBoxEx.OnEndEdit Event

Occurs when the user finishes editing the text of the combo box.

Pascal

```
property OnEndEdit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEndEdit;
```

Description

Write an OnEndEdit event handler to take specific action immediately after the user edits the text in the combo box.

See Also

OnBeginEdit (see page 3702)

Text (see page 5842)

27.1.17 TCustomHeaderControl Class

TCustomHeaderControl represents a spreadsheet-style header.

Class Hierarchy

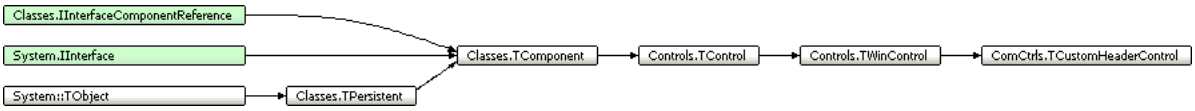

Pascal

```
TCustomHeaderControl = class(TWinControl);
```

C++

```
class TCustomHeaderControl : public TWinControl;
```

File

ComCtrls

Description

TCustomHeaderControl is an abstract base class for control classes that provide column and row headers for spreadsheet-style controls. Use TCustomHeaderControl to create new kinds of header controls.

Do not create instances of TCustomHeaderControl. THeaderControl (see page 4118) provides a minimal implementation of TCustomHeaderControl. Use THeaderControl (see page 4118) or create your own descendant classes that represent spreadsheet-style headers.

See Also

TListColumn (see page 4236)

TListItem (see page 4250)

TListView (see page 4285)

THeaderControl (see page 4118)

27.1.17.1 TCustomHeaderControl Members

The following tables list the members exposed by TCustomHeaderControl.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.

	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 3715)	Initializes a window-creation parameter data structure.
	CreateSection (see page 3715)	This is CreateSection, a member of class TCustomHeaderControl.
	CreateSections (see page 3715)	This is CreateSections, a member of class TCustomHeaderControl.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 3716)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 3716)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.

	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawSection (see page 3716)	This is DrawSection, a member of class TCustomHeaderControl.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 3717)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SectionClick (see page 3717)	This is SectionClick, a member of class TCustomHeaderControl.
	SectionDrag (see page 3717)	This is SectionDrag, a member of class TCustomHeaderControl.
	SectionEndDrag (see page 3718)	This is SectionEndDrag, a member of class TCustomHeaderControl.
	SectionResize (see page 3718)	This is SectionResize, a member of class TCustomHeaderControl.
	SectionTrack (see page 3718)	This is SectionTrack, a member of class TCustomHeaderControl.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetImages (see page 3718)	This is SetImages, a member of class TCustomHeaderControl.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 3719)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.

	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 3714)	Instantiate a TCustomHeaderControl (see page 3702) object.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.

	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 3714)	Destroy.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
V	FlipChildren (see page 3717)	Reverses the order in which sections are displayed.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 3728)	Provides runtime access to the header control's drawing surface.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.

	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnCreateSectionClass (see page 3736)	Runtime specification of section object class.
	OnDrawSection (see page 3737)	Draw header section.
	OnSectionClick (see page 3737)	Respond to mouse clicks on header section.
	OnSectionDrag (see page 3737)	Respond to section drag.
	OnSectionEndDrag (see page 3738)	Respond to moved sections
	OnSectionResize (see page 3738)	Respond to section resizing.
	OnSectionTrack (see page 3738)	Occurs as one of the header control's sections is being resized.

Published Properties

	Name	Description
	Align (see page 3723)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 3727)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 3728)	Specifies the bi-directional mode for the control.
	BorderWidth (see page 3728)	Specifies the width of the control's border.
	Constraints (see page 3730)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 3730)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 3731)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 3731)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DragReorder (see page 3733)	Make header reorderable.
	Enabled (see page 3733)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 3733)	Controls the attributes of text written on or in the control.
	FullDrag (see page 3735)	Specifies whether headers are repainted when they are dragged.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HotTrack (see page 3735)	Highlights the header sections as the mouse passes over them.
	Images (see page 3735)	Specifies header images
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Sections (see page 3735)	Specifies header sections.
	Style (see page 3736)	Specifies the header control's style.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.17.2 TCustomHeaderControl.Create Constructor

Instantiate a TCustomHeaderControl (see page 3702) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomHeaderControl(TComponent * AOwner);
```

Description

TCustomHeaderControl (see page 3702) is an abstract class and should not be instantiated in application code. Concrete subclasses, such as THeaderControl (see page 4118), call the TCustomHeaderControl (see page 3702) constructor in their own constructors.

27.1.17.3 TCustomHeaderControl.Destroy Destructor

Destroy.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomHeaderControl();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free verifies that the object variable is not nil, and only then calls Destroy.

27.1.17.4 TCustomHeaderControl Methods

The methods of the TCustomHeaderControl class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 3715)	Initializes a window-creation parameter data structure.
	CreateSection (see page 3715)	This is CreateSection, a member of class TCustomHeaderControl.
	CreateSections (see page 3715)	This is CreateSections, a member of class TCustomHeaderControl.
	CreateWnd (see page 3716)	Creates the underlying window.
	DestroyWnd (see page 3716)	Destroys the control's window.
	DrawSection (see page 3716)	This is DrawSection, a member of class TCustomHeaderControl.
	Notification (see page 3717)	Responds to notifications that components are being created or destroyed.
	SectionClick (see page 3717)	This is SectionClick, a member of class TCustomHeaderControl.
	SectionDrag (see page 3717)	This is SectionDrag, a member of class TCustomHeaderControl.
	SectionEndDrag (see page 3718)	This is SectionEndDrag, a member of class TCustomHeaderControl.
	SectionResize (see page 3718)	This is SectionResize, a member of class TCustomHeaderControl.
	SectionTrack (see page 3718)	This is SectionTrack, a member of class TCustomHeaderControl.
	SetImages (see page 3718)	This is SetImages, a member of class TCustomHeaderControl.
	WndProc (see page 3719)	Provides specific message responses for the control.

Public Methods

	Name	Description
	FlipChildren (see page 3717)	Reverses the order in which sections are displayed.

27.1.17.4.1 TCustomHeaderControl.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

27.1.17.4.2 TCustomHeaderControl.CreateSection Method

Pascal

```
function CreateSection: THeaderSection; virtual;
```

C++

```
virtual __fastcall THeaderSection CreateSection();
```

Description

This is CreateSection, a member of class TCustomHeaderControl.

27.1.17.4.3 TCustomHeaderControl.CreateSections Method

Pascal

```
function CreateSections: THeaderSections; virtual;
```

C++

```
virtual __fastcall THeaderSections CreateSections();
```

Description

This is CreateSections, a member of class TCustomHeaderControl.

27.1.17.4.4 TCustomHeaderControl.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams (see page 6135)

CreateWindowHandle (see page 6137)

Perform (see page 5776)

DestroyWnd (see page 6141)

27.1.17.4.5 TCustomHeaderControl.DestroyWnd Method

Destroys the control's window.

Pascal

```
procedure DestroyWnd; override;
```

C++

```
virtual __fastcall DestroyWnd();
```

Description

DestroyWnd is called when the control needs to replace its window. For example, changing properties that are implemented by window parameter settings require the control to be destroyed and then recreated using the CreateWnd method.

Before the window is destroyed, DestroyWnd saves a copy of the control's text in memory, frees any device contexts, and finally calls DestroyWindowHandle. When the window is recreated, the stored copy of the control's text is assigned to the new window.

See Also

DestroyWindowHandle (see page 6140)

RecreateWnd (see page 6171)

27.1.17.4.6 TCustomHeaderControl.DrawSection Method

Pascal

```
procedure DrawSection(Section: THeaderSection; const Rect: TRect; Pressed: Boolean); dynamic;
```

C++

```
__fastcall DrawSection(THeaderSection Section, const TRect Rect, Boolean Pressed);
```

Description

This is DrawSection, a member of class TCustomHeaderControl.

27.1.17.4.7 TCustomHeaderControl.FlipChildren Method

Reverses the order in which sections are displayed.

Pascal

```
procedure FlipChildren(AllLevels: Boolean); override;
```

C++

```
virtual __fastcall FlipChildren(Boolean AllLevels);
```

Description

FlipChildren flips the order of the header control's sections; then calls the inherited FlipChildren method to reverse the order of any other child controls.

27.1.17.4.8 TCustomHeaderControl.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

TCustomHeaderControl (see page 3702) overrides Notification in order to call ImageListChanged when the image list is removed.

See Also

Notification (see page 2783)

27.1.17.4.9 TCustomHeaderControl.SectionClick Method

Pascal

```
procedure SectionClick(Section: THeaderSection); dynamic;
```

C++

```
__fastcall SectionClick(THeaderSection Section);
```

Description

This is SectionClick, a member of class TCustomHeaderControl.

27.1.17.4.10 TCustomHeaderControl.SectionDrag Method

Pascal

```
procedure SectionDrag(FromSection: THeaderSection; ToSection: THeaderSection; var AllowDrag:
Boolean); dynamic;
```

C++

```
__fastcall SectionDrag(THeaderSection FromSection, THeaderSection ToSection, Boolean
AllowDrag);
```

Description

This is SectionDrag, a member of class TCustomHeaderControl.

27.1.17.4.11 TCustomHeaderControl.SectionEndDrag Method

Pascal

```
procedure SectionEndDrag; dynamic;
```

C++

```
__fastcall SectionEndDrag();
```

Description

This is SectionEndDrag, a member of class TCustomHeaderControl.

27.1.17.4.12 TCustomHeaderControl.SectionResize Method

Pascal

```
procedure SectionResize(Section: THeaderSection); dynamic;
```

C++

```
__fastcall SectionResize(THeaderSection Section);
```

Description

This is SectionResize, a member of class TCustomHeaderControl.

27.1.17.4.13 TCustomHeaderControl.SectionTrack Method

Pascal

```
procedure SectionTrack(Section: THeaderSection; Width: Integer; State: TSectionTrackState);
dynamic;
```

C++

```
__fastcall SectionTrack(THeaderSection Section, int Width, TSectionTrackState State);
```

Description

This is SectionTrack, a member of class TCustomHeaderControl.

27.1.17.4.14 TCustomHeaderControl.SetImages Method

Pascal

```
procedure SetImages(Value: TCustomImageList); virtual;
```

C++

```
virtual __fastcall SetImages(TCustomImageList Value);
```

Description

This is SetImages, a member of class TCustomHeaderControl.

27.1.17.4.15 TCustomHeaderControl.WndProc Method

Provides specific message responses for the control.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

Override WndProc to change the initial Windows message handler for the control. The WindowProc property is initialized to point to the WndProc method.

WndProc for TWinControl overrides the inherited method to define responses for focus, mouse, and keyboard messages. It sends all others to its inherited WndProc.

When overriding WndProc to provide specialized responses to messages, call the inherited WndProc at the end to dispatch any unhandled messages.

C++ Examples:

```
/*
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
*/
class TMyListBoxDescendant : public TListBox
{
__published:    // IDE-managed Components
    void __fastcall SubClassWndProc(Messages::TMessage &Message);
    void __fastcall ToggleSubClass(bool On);
    void __fastcall OnDrawItemProc(
      TWinControl *Control, int Index, const TRect &Rect,
      TOwnerDrawState State);
private:    // User declarations
public:     // User declarations
    __fastcall TMyListBoxDescendant(TComponent* Owner);
};

TForm1 *Form1;
TMyListBoxDescendant *MyListBoxDescendant1;
Graphics::TBitmap *bitmap0;

const WM_STYLEMESSAGE = WM_USER + 2000;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

__fastcall TMyListBoxDescendant::TMyListBoxDescendant(TComponent* Owner)
    : TListBox(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
```

```
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
}

void __fastcall TForm1::SubClassRadioGroup1Click(TObject *Sender)
{
  MyListBoxDescendant1->ToggleSubClass(SubClassRadioGroup1->ItemIndex == 0);
}

void __fastcall TMyListBoxDescendant::SubClassWndProc(Messages::TMessage &Message)
{
  if (Message.Msg == WM_STYLEMESSAGE)
    Style = (TListBoxStyle)Message.WParam;
  else
    WndProc(Message);
}

void __fastcall TMyListBoxDescendant::ToggleSubClass(bool On)
{
  if (On)
    WindowProc = SubClassWndProc;
  else
    WindowProc = WndProc;
}

void __fastcall TMyListBoxDescendant::OnDrawItemProc(TWinControl *Control, int Index,
      const TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *bitmap; // temporary variable for item's bitmap
  int Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *canvas = ((TListBox *)Control)->Canvas;
  canvas->FillRect(Rect); // clear the rectangle
  bitmap = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
  if (bitmap)
  {
    canvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, bitmap->Width, bitmap->Height),
      bitmap, Bounds(0, 0, bitmap->Width, bitmap->Height), clRed); // render bitmap
    Offset += bitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  canvas->TextOut(Rect.Left + Offset, Rect.Top, ((TListBox *)Control)->Items->Strings[Index]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyListBoxDescendant1 = new TMyListBoxDescendant(Form1);
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Parent = Form1;
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Left =
    SubClassRadioGroup1->Left + SubClassRadioGroup1->Width + 30;;
  MyListBoxDescendant1->Top = SubClassRadioGroup1->Top;
  MyListBoxDescendant1->Height = SubClassRadioGroup1->Height;
  MyListBoxDescendant1->OnDrawItem =
```

```
    MyListBoxDescendant1->OnDrawItemProc;

  bitmap0 = new Graphics::TBitmap;
  ImageList1->GetBitmap(0, bitmap0);
  MyListBoxDescendant1->Items->AddObject("Butterfly", bitmap0);

  SubClassRadioGroup1->Items->Add("SubClassWndProc");
  SubClassRadioGroup1->Items->Add("WndProc");
  SubClassRadioGroup1->ItemIndex = 2;
}
```

Delphi Examples:

```
{
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
}

type
  TMyListBoxDescendant = class(TlistBox)
    procedure SubClassWndProc(var Message: TMessage);
    procedure ToggleSubClass(On: Boolean);
    procedure OnDrawItemProc(
      Control: TWinControl;
      Index: Integer;
      Rect:TRect;
      State: TOwnerDrawState);
  end;
  TForm1 = class(TForm)
    SubClassRadioGroup1: TRadioGroup;
    Button1: TButton;
    ImageList1: TImageList;
    Button2: TButton;
    procedure SubClassRadioGroup1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  MyListBoxDescendant1: TMyListBoxDescendant;
  bitmap0: TBitmap;

implementation

{$R *.dfm}

const WM_STYLEMESSAGE = WM_USER + 2000;

procedure TForm1.Button1Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
end;

procedure TMyListBoxDescendant.SubClassWndProc(var Message: TMessage);
begin
  if (Message.Msg = WM_STYLEMESSAGE) then
    Style:= TListBoxStyle(Message.WParam)
  else
    WndProc(Message);
end;

procedure TMyListBoxDescendant.ToggleSubClass(On: Boolean);
begin
  if On then
    WindowProc := SubClassWndProc
  else
    WindowProc := WndProc;
end;

procedure TForm1.SubClassRadioGroup1Click(Sender: TObject);
begin
  MyListBoxDescendant1.ToggleSubClass(
    SubClassRadioGroup1.ItemIndex = 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyListBoxDescendant1:= TMyListBoxDescendant.Create(self);
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Parent:= Form1;
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Left:=
    SubClassRadioGroup1.Left + SubClassRadioGroup1.Width + 30;;
  MyListBoxDescendant1.Top:= SubClassRadioGroup1.Top;
  MyListBoxDescendant1.Height:= SubClassRadioGroup1.Height;
  MyListBoxDescendant1.OnDrawItem:=
    MyListBoxDescendant1.OnDrawItemProc;

  bitmap0 := TBitmap.Create;
  ImageList1.GetBitmap(0, bitmap0);
  MyListBoxDescendant1.Items.AddObject('Butterfly', bitmap0);

  SubClassRadioGroup1.Items.Add('SubClassWndProc');
  SubClassRadioGroup1.Items.Add('WndProc');
  SubClassRadioGroup1.ItemIndex := 2;
end;

procedure TMyListBoxDescendant.OnDrawItemProc(
  Control: TWinControl;
  Index: Integer;
  Rect:TRect;
  State: TOwnerDrawState);
var
  Bitmap: TBitmap;      { temporary variable for the item's bitmap }
  Offset: Integer;      { text offset width }
begin
  { draw on control canvas, not on the form }
  with (Control as TListBox).Canvas do
  begin
```

```
    FillRect(Rect);        { clear the rectangle }
    Offset := 2;           { provide default offset }
    Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
    if Bitmap <> nil then
    begin
      BrushCopy(
        Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
        Bitmap,
        Bounds(0, 0, Bitmap.Width, Bitmap.Height),
        clRed);  {render bitmap}
      Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
    end;
    TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
text }
  end;
end;
```

See Also

DefWndProc (see page 6199)

MainWndProc (see page 6165)

WndProc (see page 5791)

WindowProc (see page 5872)

27.1.17.5 TCustomHeaderControl Properties

The properties of the TCustomHeaderControl class are listed here.

Public Properties

	Name	Description
R	Canvas (see page 3728)	Provides runtime access to the header control's drawing surface.

Published Properties

	Name	Description
	Align (see page 3723)	Determines how the control aligns within its container (parent control).
	Anchors (see page 3727)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 3728)	Specifies the bi-directional mode for the control.
	BorderWidth (see page 3728)	Specifies the width of the control's border.
	Constraints (see page 3730)	Specifies the size constraints for the control.
	DragCursor (see page 3730)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 3731)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 3731)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DragReorder (see page 3733)	Make header reorderable.
	Enabled (see page 3733)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 3733)	Controls the attributes of text written on or in the control.
	FullDrag (see page 3735)	Specifies whether headers are repainted when they are dragged.
	HotTrack (see page 3735)	Highlights the header sections as the mouse passes over them.
	Images (see page 3735)	Specifies header images
	Sections (see page 3735)	Specifies header sections.
	Style (see page 3736)	Specifies the header control's style.

27.1.17.5.1 TCustomHeaderControl.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;
```

```
  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
```

```
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
```

```
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

27.1.17.5.2 TCustomHeaderControl.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.17.5.3 TCustomHeaderControl.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

27.1.17.5.4 TCustomHeaderControl.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics or text drawn by the control is clipped to the area within the border.

See Also

ClientRect

BevelOuter (see page 6194)

27.1.17.5.5 TCustomHeaderControl.Canvas Property

Provides runtime access to the header control's drawing surface.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

The read-only Canvas property provides access to the header control's drawing surface that can be used when implementing a handler for the OnDrawSection (see page 3737) event.

C++ Examples:

```
/*
Add a THeaderControl to the form and call
HeaderControl1DrawSection for the OnDrawSection event.
*/
void __fastcall TForm1::HeaderControl1DrawSection(THeaderControl *HeaderControl,
      THeaderSection *Section, const TRect &Rect, bool Pressed)
{
  if (Pressed)
    HeaderControl->Canvas->Font->Color = clRed;
  else
    HeaderControl->Canvas->Font->Color = clBlue;
  HeaderControl->Canvas->TextOut(Rect.Left + HeaderControl->Canvas->Font->Size, Rect.Top + 2,
"Custom " + IntToStr(Section->Index));
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  THeaderSection *Section;
  for (int i = 0; i <= 4; i++)
  {
    Section = HeaderControl1->Sections->Add();
    Section->Text = "Section " + IntToStr(i);
    Section->MinWidth =
      (Section->Text).Length() *
      HeaderControl1->Font->Size;
    // Owner draw every other section
    if (i % 2 == 0)
      Section->Style = hsOwnerDraw;
    else
      Section->Style = hsText;
  }
}
```

Delphi Examples:

```
{
Add a THeaderControl to the form and call
HeaderControl1DrawSection for the OnDrawSection event.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  HeaderSection: THeaderSection;
  I: Integer;
begin
  for I := 0 to 4 do
  begin
    HeaderSection := HeaderControl1.Sections.Add;
    HeaderSection.Text := 'Text Section ' + IntToStr(I);
    HeaderSection.MinWidth := length(HeaderSection.Text) * Font.Size;
    // Owner draw every other section
    if (I mod 2 = 0) then
      HeaderSection.Style := hsOwnerDraw
    else
      HeaderSection.Style := hsText;
  end;
end;
```

```
procedure TForm1.HeaderControl1DrawSection(HeaderControl: THeaderControl;
  Section: THeaderSection; const Rect: TRect; Pressed: Boolean);
begin
  with HeaderControl.Canvas do
  begin
    // highlight pressed sections
    if Pressed then
      Font.Color := clRed
    else
      Font.Color := clBlue;
    TextOut(Rect.Left + Font.Size, Rect.Top + 2, 'Owner Drawn text');
  end;
end;
```

See Also

OnDrawSection (see page 3737)

27.1.17.5.6 TCustomHeaderControl.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.17.5.7 TCustomHeaderControl.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

27.1.17.5.8 TCustomHeaderControl.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

27.1.17.5.9 TCustomHeaderControl.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
```

```
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

27.1.17.5.10 TCustomHeaderControl.DragReorder Property

Make header reorderable.

Pascal

```
property DragReorder: Boolean;
```

C++

```
__property Boolean DragReorder;
```

Description

If DragReorder is true (default is false), the user can rearrange headers by dragging them.

See Also

OnSectionDrag (see page 3737)

OnSectionEndDrag (see page 3738)

27.1.17.5.11 TCustomHeaderControl.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.17.5.12 TCustomHeaderControl.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

27.1.17.5.13 TCustomHeaderControl.FullDrag Property

Specifies whether headers are repainted when they are dragged.

Pascal

```
property FullDrag: Boolean;
```

C++

```
__property Boolean FullDrag;
```

Description

Use FullDrag to specify whether headers are repainted while they are dragged.

See Also

OnSectionDrag (see page 3737)

27.1.17.5.14 TCustomHeaderControl.HotTrack Property

Highlights the header sections as the mouse passes over them.

Pascal

```
property HotTrack: Boolean;
```

C++

```
__property Boolean HotTrack;
```

Description

If HotTrack is set to true, the text on each header section appears highlighted when the mouse passes over it at runtime.

See Also

Text (see page 4183)

27.1.17.5.15 TCustomHeaderControl.Images Property

Specifies header images

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Images contains a list of images that can be used as icons. Each section's ImageIndex property determines the specific image displayed on it. If Images is unassigned, items are displayed as captions with no icons.

See Also

ImageIndex

27.1.17.5.16 TCustomHeaderControl.Sections Property

Specifies header sections.

Pascal

```
property Sections: THeaderSections;
```

C++

```
__property THeaderSections Sections;
```

Description

The Sections property refers to the THeaderSections (see page 4185) object that contains the header sections.. This protected property is only useful for creating new control classes derived from TCustomHeaderControl (see page 3702).

27.1.17.5.17 TCustomHeaderControl.Style Property

Specifies the header control's style.

Pascal

```
property Style: THeaderStyle;
```

C++

```
__property THeaderStyle Style;
```

Description

Use Style to determine the appearance of the header sections.

27.1.17.6 TCustomHeaderControl Events

The events of the TCustomHeaderControl class are listed here.

Published Events

	Name	Description
	OnCreateSectionClass (see page 3736)	Runtime specification of section object class.
	OnDrawSection (see page 3737)	Draw header section.
	OnSectionClick (see page 3737)	Respond to mouse clicks on header section.
	OnSectionDrag (see page 3737)	Respond to section drag.
	OnSectionEndDrag (see page 3738)	Respond to moved sections
	OnSectionResize (see page 3738)	Respond to section resizing.
	OnSectionTrack (see page 3738)	Occurs as one of the header control's sections is being resized.

27.1.17.6.1 TCustomHeaderControl.OnCreateSectionClass Event

Runtime specification of section object class.

Pascal

```
property OnCreateSectionClass: TCustomHCCreateSectionClassEvent;
```

C++

```
__property TCustomHCCreateSectionClassEvent OnCreateSectionClass;
```

Description

The OnCreateSectionClass event occurs when a new header section object is about to be created. If the event handler exists, it is called with SectionClass set to THeaderSection (see page 4173). The event handler can modify this value to any descendant of THeaderSection (see page 4173), which will then be used to create the header section object.

27.1.17.6.2 TCustomHeaderControl.OnDrawSection Event

Draw header section.

Pascal

```
property OnDrawSection: TCustomDrawSectionEvent;
```

C++

```
__property TCustomDrawSectionEvent OnDrawSection;
```

Description

Use the OnDrawSection to define custom section drawing code. The event handler takes the following parameters.

Section is the header section that is being redrawn.

Rect is the display rectangle for the header section.

Pressed is true if the section is being displayed as a button, and that button is currently in its depressed state.

27.1.17.6.3 TCustomHeaderControl.OnSectionClick Event

Respond to mouse clicks on header section.

Pascal

```
property OnSectionClick: TCustomSectionNotifyEvent;
```

C++

```
__property TCustomSectionNotifyEvent OnSectionClick;
```

Description

The OnSectionClick event occurs when the user clicks on a header section.

27.1.17.6.4 TCustomHeaderControl.OnSectionDrag Event

Respond to section drag.

Pascal

```
property OnSectionDrag: TSectionDragEvent;
```

C++

```
__property TSectionDragEvent OnSectionDrag;
```

Description

The OnSectionDrag event occurs when the user attempts to rearrange header sections by dragging a section to a new position. This event can only occur when the DragReorder (see page 3733) property is true. The event handler is passed the following parameters.

FromSection is the section being dragged.

ToSection is the section currently occupying the position FromSection was dragged to. If the operation is successful, FromSection will move to this position, and other sections, including ToSection shifted right or left to make room. Note that FromSection and ToSection can be the same section if FromSection was not dragged far enough to trigger a re-arrangement.

AllowDrag is passed a value of true. Change AllowDrag to false to cancel the operation.

See Also

OnSectionEndDrag (see page 3738)

DragReorder (see page 3733)

27.1.17.6.5 TCustomHeaderControl.OnSectionEndDrag Event

Respond to moved sections

Pascal

```
property OnSectionEndDrag: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSectionEndDrag;
```

Description

The OnSectionEndDrag event occurs after the user moves a header section to a new position.

See Also

OnSectionDrag (see page 3737)

27.1.17.6.6 TCustomHeaderControl.OnSectionResize Event

Respond to section resizing.

Pascal

```
property OnSectionResize: TCustomSectionNotifyEvent;
```

C++

```
__property TCustomSectionNotifyEvent OnSectionResize;
```

Description

The OnSectionResize event occurs when the user resizes a header section. If the Tracking property is true (default), a series of OnSectionResize events occur while the user is manipulating the section. Otherwise, OnSectionResize occurs only when the user releases the mouse button.

27.1.17.6.7 TCustomHeaderControl.OnSectionTrack Event

Occurs as one of the header control's sections is being resized.

Pascal

```
property OnSectionTrack: TCustomSectionTrackEvent;
```

C++

```
__property TCustomSectionTrackEvent OnSectionTrack;
```

Description

The OnSectionTrack event tracks the dragging of a header section's border as it happens. OnSectionTrack occurs when the mouse pointer is positioned between two header sections and the left mouse button is depressed.

OnSectionTrack is of type TCustomSectionTrackEvent (see page 5485), whose Width and State parameters indicate the size of the header section (in pixels) and the status of the event. The State property can take three values:

Value	Meaning
tsTrackBegin	The border has not yet been dragged.
tsTrackMove	The border is being dragged.
tsTrackEnd	The border is no longer being dragged.

See Also

OnSectionResize (see page 3738)

27.1.18 TCustomHotKey Class

TCustomHotKey is the base class for objects that wrap the Windows accelerator key Common Control.

Class Hierarchy

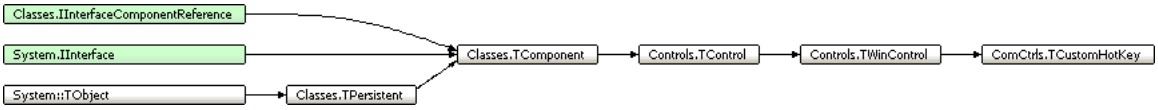

Pascal

```
TCustomHotKey = class(TWinControl);
```

C++

```
class TCustomHotKey : public TWinControl;
```

File

ComCtrls

Description

TCustomHotKey is an abstract class that encapsulates behavior of an object that is used to set a shortcut property at runtime. Accelerator key objects are used to perform an action quickly by enabling users to link the accelerator key to a combination of keystrokes that will execute the action. TCustomHotKey introduces properties to:

Associate the accelerator key with keystrokes to perform an action

Select a key combination at design time

Invalidate specific modifier keys

To change the key combination at runtime hold down the Alt, Ctrl or Shift keys or combinations of them and type a letter or number. The accelerator key property can then be assigned to the shortcut property of a menu item, for example.

Hot key classes that descend from TCustomHotKey inherit the basic functionality of TCustomHotKey. Use TCustomHotKey as a base class for deriving specialized accelerator key objects.

See Also

THotKey (see page 4190)

27.1.18.1 TCustomHotKey Members

The following tables list the members exposed by TCustomHotKey.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnChange (see page 3757)	Occurs when the properties of the accelerator key change.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.

	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	CNCommand (see page 3750)	This is CNCommand, a member of class TCustomHotKey.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 3751)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 3751)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.

	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.

	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.

	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 3752)	This is SetAutoSize, a member of class TCustomHotKey.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.

	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 3752)	Determines whether the accelerator key automatically changes its height to accommodate the font size.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	HotKey (see page 3752)	Contains the current key combination of the accelerator key control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	InvalidKeys (see page 3753)	Allows you to specify that one or more modifier keys should be considered invalid.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	Modifiers (see page 3755)	Specifies the key or keys that are used in combination with a non-modifier key.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	TabStop (see page 3756)	Determines if the user can tab to a control.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 3750)	Instantiates an accelerator key control.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.

R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.

	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.18.2 TCustomHotKey.Create Constructor

Instantiates an accelerator key control.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomHotKey(TComponent * AOwner);
```

Description

Call Create to instantiate an accelerator key control at runtime. For accelerator key controls added at design time, Create is called automatically.

AOwner is the component, typically the form, that is responsible for freeing the hotkey instance. It becomes the value of the Owner property.

See Also

Create (see page 6123)

Owner (see page 2807)

27.1.18.3 TCustomHotKey Methods

The methods of the TCustomHotKey class are listed here.

Protected Methods

	Name	Description
	CNCommand (see page 3750)	This is CNCommand, a member of class TCustomHotKey.
V	CreateParams (see page 3751)	Initializes a window-creation parameter data structure.
V	CreateWnd (see page 3751)	Creates the underlying window.
V	SetAutoSize (see page 3752)	This is SetAutoSize, a member of class TCustomHotKey.

27.1.18.3.1 TCustomHotKey.CNCommand Method

Pascal

```
procedure CNCommand(var Message: TWMCommand); message CN_COMMAND;
```

C++

```
__fastcall CNCommand(TWMCommand Message);
```

Description

This is CNCommand, a member of class TCustomHotKey.

27.1.18.3.2 TCustomHotKey.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

27.1.18.3.3 TCustomHotKey.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams (see page 6135)

CreateWindowHandle (see page 6137)

Perform (see page 5776)

DestroyWnd (see page 6141)

27.1.18.3.4 TCustomHotKey.SetAutoSize Method

Pascal

```
procedure SetAutoSize(Value: Boolean); override;
```

C++

```
virtual __fastcall SetAutoSize(Boolean Value);
```

Description

This is SetAutoSize, a member of class TCustomHotKey.

27.1.18.4 TCustomHotKey Properties

The properties of the TCustomHotKey class are listed here.

Protected Properties

	Name	Description
	AutoSize (see page 3752)	Determines whether the accelerator key automatically changes its height to accommodate the font size.
	HotKey (see page 3752)	Contains the current key combination of the accelerator key control.
	InvalidKeys (see page 3753)	Allows you to specify that one or more modifier keys should be considered invalid.
	Modifiers (see page 3755)	Specifies the key or keys that are used in combination with a non-modifier key.
	TabStop (see page 3756)	Determines if the user can tab to a control.

27.1.18.4.1 TCustomHotKey.AutoSize Property

Determines whether the accelerator key automatically changes its height to accommodate the font size.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to fully display the font on the accelerator key control. When AutoSize is true, the height of the accelerator key changes to display the key combinations typed in it. When AutoSize is false, the accelerator key remains the same height, regardless of any font changes. The default value is true.

27.1.18.4.2 TCustomHotKey.HotKey Property

Contains the current key combination of the accelerator key control.

Pascal

```
property HotKey: TShortCut;
```

C++

```
__property TShortCut HotKey;
```

Description

Use HotKey to set or change the key combination associated with the accelerator key. Alt+A is the default. To modify the value of HotKey either edit it directly or set the value of the Modifiers (see page 3755) property. At runtime, HotKey is reset when the

accelerator key has focus and the user presses valid key combinations.

C++ Examples:

```
/*
To use this example, create a form with a main menu
component and a HotKey component.  At design time, use the
Menu designer to add menu items, including a "File" menu
with subitems that includes a "New" menu item.  Add the
following code to the OnCreate method of the form and the
OnClick event of the File menu item.
*/
void __fastcall TForm1::File1Click(TObject *Sender)
{
  // Assign HotKey to the menu's File "New" item.
  New1->ShortCut = HotKey1->HotKey;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Default to "Ctrl + A"
  HotKey1->Modifiers.Clear();
  HotKey1->Modifiers << hkCtrl;
  // Don't allow shift or alt
  HotKey1->InvalidKeys.Clear();
  (HotKey1->InvalidKeys << hcShift) << hcAlt;
//  HotKey1->Visible = false;
}
```

Delphi Examples:

```
{
To use this example, create a form with a main menu
component and a HotKey component.  At design time, use the
Menu designer to add menu items, including a "File" menu
with subitems that includes a "New" menu item.  Add the
following code to the OnCreate method of the form and the
OnClick event of the File menu item.
}
procedure TForm1.File1Click(Sender: TObject);
begin
  // Assign hotkey to the menu's File "New" item.
  New1.ShortCut := HotKey1.HotKey;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  // Default to "Ctrl + A"
  HotKey1.Modifiers := [hkCtrl];
  // Don't allow shift or alt
  HotKey1.InvalidKeys := [hcShift, hcAlt];
end;
```

See Also

Modifiers (see page 3755)

27.1.18.4.3 TCustomHotKey.InvalidKeys Property

Allows you to specify that one or more modifier keys should be considered invalid.

Pascal

```
property InvalidKeys: THKInvalidKeys;
```

C++

```
__property THKInvalidKeys InvalidKeys;
```

Description

Use InvalidKeys to select which modifier keys are not allowed. If this property is set, a default modifier should also be specified in the Modifiers (see page 3755) property. When the invalid key is selected, the default modifier is displayed in its place.

THKInvalidKey (see page 5451) is the type of the InvalidKeys property. These are the possible values:

Value	Meaning
hcNone	Unmodified keys are invalid.
hcShift	The Shift key is invalid as a modifier.
hcCtrl	The Ctrl key is invalid as a modifier.
hcAlt	The Alt key is invalid as a modifier.
hcShiftCtrl	The Shift+Ctrl key combination is invalid as a modifier.
hcShiftAlt	The Shift+Alt key combinations invalid as a modifier.
hcCtrlAlt	The Ctrl+Alt key combination is invalid as a modifier.
hcShiftCtrlAlt	The Shift+Ctrl+Alt key combination is invalid as a modifier.

The default values are hcNone and hcShift.

C++ Examples:

```
/*
To use this example, create a form with a main menu
component and a HotKey component.  At design time, use the
Menu designer to add menu items, including a "File" menu
with subitems that includes a "New" menu item.  Add the
following code to the OnCreate method of the form and the
OnClick event of the File menu item.
*/
void __fastcall TForm1::File1Click(TObject *Sender)
{
  // Assign HotKey to the menu's File "New" item.
  New1->ShortCut = HotKey1->HotKey;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Default to "Ctrl + A"
  HotKey1->Modifiers.Clear();
  HotKey1->Modifiers << hkCtrl;
  // Don't allow shift or alt
  HotKey1->InvalidKeys.Clear();
  (HotKey1->InvalidKeys << hcShift) << hcAlt;
//  HotKey1->Visible = false;
}
```

Delphi Examples:

```
{
To use this example, create a form with a main menu
component and a HotKey component.  At design time, use the
Menu designer to add menu items, including a "File" menu
with subitems that includes a "New" menu item.  Add the
following code to the OnCreate method of the form and the
OnClick event of the File menu item.
}
procedure TForm1.File1Click(Sender: TObject);
```

```
begin
  // Assign hotkey to the menu's File "New" item.
  New1.ShortCut := HotKey1.HotKey;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  // Default to "Ctrl + A"
  HotKey1.Modifiers := [hkCtrl];
  // Don't allow shift or alt
  HotKey1.InvalidKeys := [hcShift, hcAlt];
end;
```

See Also

Modifiers (see page 3755)

27.1.18.4.4 TCustomHotKey.Modifiers Property

Specifies the key or keys that are used in combination with a non-modifier key.

Pascal

```
property Modifiers: THKModifiers;
```

C++

```
__property THKModifiers Modifiers;
```

Description

Use Modifiers to select a modifier key such as Ctrl, Alt, or Shift for the hot key. These are the possible values:

Value	Meaning
hkShift	The Shift key is used as a modifier.
hkCtrl	The Ctrl key is used as a modifier.
hkAlt	The Alt key is used as a modifier.
hkExt	The Extra key is used as a modifier.

The default value is hkAlt.

Modifier keys are used with other non-modifier keys such as character and function keys, arrow keys, and so on. More than one modifier key can be used.

C++ Examples:

```
/*
To use this example, create a form with a main menu
component and a HotKey component.  At design time, use the
Menu designer to add menu items, including a "File" menu
with subitems that includes a "New" menu item.  Add the
following code to the OnCreate method of the form and the
OnClick event of the File menu item.
*/
void __fastcall TForm1::File1Click(TObject *Sender)
{
  // Assign HotKey to the menu's File "New" item.
  New1->ShortCut = HotKey1->HotKey;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Default to "Ctrl + A"
```

```
  HotKey1->Modifiers.Clear();
  HotKey1->Modifiers << hkCtrl;
  // Don't allow shift or alt
  HotKey1->InvalidKeys.Clear();
  (HotKey1->InvalidKeys << hcShift) << hcAlt;
//  HotKey1->Visible = false;
}
```

Delphi Examples:

```
{
To use this example, create a form with a main menu
component and a HotKey component.  At design time, use the
Menu designer to add menu items, including a "File" menu
with subitems that includes a "New" menu item.  Add the
following code to the OnCreate method of the form and the
OnClick event of the File menu item.
}
procedure TForm1.File1Click(Sender: TObject);
begin
  // Assign hotkey to the menu's File "New" item.
  New1.ShortCut := HotKey1.HotKey;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  // Default to "Ctrl + A"
  HotKey1.Modifiers := [hkCtrl];
  // Don't allow shift or alt
  HotKey1.InvalidKeys := [hcShift, hcAlt];
end;
```

27.1.18.4.5 TCustomHotKey.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

27.1.18.5 TCustomHotKey Events

The events of the TCustomHotKey class are listed here.

Protected Events

	Name	Description
	OnChange (see page 3757)	Occurs when the properties of the accelerator key change.

27.1.18.5.1 TCustomHotKey.OnChange Event

Occurs when the properties of the accelerator key change.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to respond when the accelerator key changes to represent a different combination of keys. OnChange occurs after the TCustomHotKey (see page 3739) properties have been updated to reflect the change.

27.1.19 TCustomListView Class

TCustomListView is the base type for all list view components, including TListView (see page 4285).

Class Hierarchy

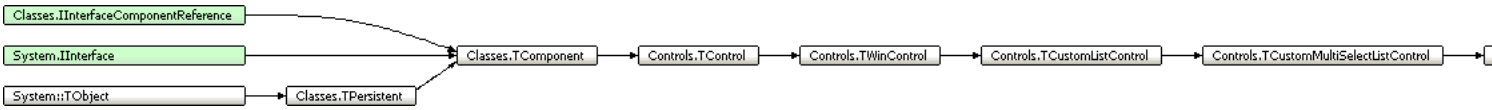

Pascal

```
TCustomListView = class(TCustomMultiSelectListControl);
```

C++

```
class TCustomListView : public TCustomMultiSelectListControl;
```

File

ComCtrls

Description

Use TCustomListView as a base class when defining a control that displays a list of items. TCustomListView allows a list of items to be displayed in columns with column headers and sub-items, or vertically or horizontally, with small or large icons.

Do not create instances of TCustomListView. To put a list view on a form, use a descendant of TCustomListView, such as TListView (see page 4285).

See Also

TListColumn (see page 4236)

TListItem (see page 4250)

TListView (see page 4285)

TCustomHeaderControl (see page 3702)

27.1.19.1 TCustomListView Members

The following tables list the members exposed by TCustomListView.

Protected Events

	Name	Description
	OnAdvancedCustomDraw (see page 3834)	Occurs at discrete points during the rendering of the list view.
	OnAdvancedCustomDrawItem (see page 3835)	Occurs at discrete stages during the painting of an item in an owner-draw list view.
	OnAdvancedCustomDrawSubItem (see page 3835)	Occurs at discrete stages during the painting of a subitem in an owner-drawn list view.
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnChange (see page 3836)	Occurs immediately after an item in the list changes.
	OnChanging (see page 3837)	Occurs when an item in the list is about to change.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnColumnClick (see page 3837)	Occurs when the user clicks on a column header in a list view.
	OnColumnDragged (see page 3840)	Occurs when a column is dragged to a new position.
	OnColumnRightClick (see page 3841)	Occurs when the user right clicks on a column header in a list view.
	OnCompare (see page 3841)	Occurs to when two items need to be compared during a sort of the list.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreateItemClass (see page 3844)	Occurs when a list item object is about to be created.
	OnCustomDraw (see page 3844)	Occurs when an owner-draw list view must be rendered.
	OnCustomDrawItem (see page 3845)	Occurs when an item in an owner-draw list view must be rendered.
	OnCustomDrawSubItem (see page 3846)	Occurs when a subitem must be rendered in an owner-draw list view.
	OnData (see page 3846)	Occurs immediately before an item is displayed in the list view control.
	OnDataFind (see page 3847)	Occurs when the list view control receives a search request, such as when the FindData (see page 3789) and FindCaption (see page 3788) methods are called.
	OnDataHint (see page 3847)	Occurs when the contents of the list view change, such as when the user scrolls the list view.
	OnDataStateChange (see page 3848)	Occurs when a range of items change states.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeletion (see page 3848)	Occurs when an item in the list view is about to be deleted.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 3848)	Occurs when an item should be drawn in the list view.
	OnEdited (see page 3849)	Occurs immediately after the user edits an item's Caption property.
	OnEditing (see page 3850)	Occurs when the user starts to edit a list item's Caption property.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetImageIndex (see page 3850)	Occurs when an item is about to be displayed in the list view control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnGetSubItemImage (see page 3850)	Occurs when a subitem is about to be displayed in the column of a list view control.
	OnInfoTip (see page 3851)	Occurs when the user pauses the mouse over an item in the list view.
	OnInsert (see page 3851)	Occurs immediately after a new item is inserted into the list view.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.

	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnSelectItem (see page 3852)	Occurs when an item is selected in the list view control.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FMultiSelect (see page 5956)	This is FMultiSelect, a member of class TCustomMultiSelectListControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 3773)	Responds when the action for which this list view is a client changes.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.

	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanChange (see page 3777)	Specifies whether the properties of a list item can be changed.
	CanEdit (see page 3778)	Specifies whether the text of a list item can be changed by the user.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 3778)	Generates an OnChange (see page 3836) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 3778)	Rescaling event dispatcher
	Click (see page 5743)	Respond to user click.
	ColClick (see page 3780)	Column (see page 3806) click event dispatcher.
	ColRightClick (see page 3780)	Column (see page 3806) right-click event dispatcher.
	ColumnsShowing (see page 3780)	Indicates whether the ViewStyle (see page 3829) property is vsReport.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateListItem (see page 3781)	Creates a list item for the collection specified by the Items (see page 3814) property.
	CreateListItems (see page 3782)	Creates the TListItems (see page 4270) object that implements the Items (see page 3814) property.
	CreateParams (see page 3782)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 3782)	Creates the window used by the list view.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	CustomDraw (see page 3783)	Custom draw event dispatcher.
	CustomDrawItem (see page 3783)	Custom draw item event dispatcher.
	CustomDrawSubItem (see page 3783)	Custom draw subitem event dispatcher.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	Delete (see page 3785)	Removes an item from the list view and frees the associated memory.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 3786)	Destroys the window referenced by the list view's window handle.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.

	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
V	DoEndDrag (see page 3787)	Drag end event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
V	DoInfoTip (see page 3787)	InfoTip event dispatcher.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
V	DoStartDrag (see page 3787)	Start drag event dispatcher..
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawItem (see page 3788)	Draw item event dispatcher.
	Edit (see page 3788)	Edit event dispatcher.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
V	GetActionLinkClass (see page 3789)	Returns the action link class that is associated with list view controls.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetCount (see page 3789)	Returns the number of items in the list.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 3790)	Returns a drag cursor if a list item is being dragged.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetItemIndex (see page 3791)	This is the overview for the GetItemIndex method overload.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.

	GetSelCount (see page 3794)	Returns the value of the SelCount (see page 3822) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InsertItem (see page 3794)	Generates an OnInsert (see page 3851) event.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsCustomDrawn (see page 3794)	Determines when custom draw event handlers are called.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 3795)	Mouse button up event dispatcher.
	Notification (see page 3796)	Responds to notifications that components are about to be created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	OwnerDataFetch (see page 3796)	Customizes a list view item before it is displayed.
	OwnerDataFind (see page 3797)	Searches for an item in the list view control.
	OwnerDataHint (see page 3797)	Specifies a range of items that might be fetched.
	OwnerDataStateChange (see page 3798)	Specifies items that have changed states.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.

	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetItemIndex (see page 3799)	Sets the value of the ItemIndex property.
V	SetMultiSelect (see page 3800)	Sets the value of the MultiSelect (see page 3820) property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetViewStyle (see page 3800)	Sets the value of the ViewStyle (see page 3829) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateColumn (see page 3801)	Individual column update event dispatcher.
	UpdateColumns (see page 3801)	General column update event dispatcher.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.

	WndProc (see page 3802)	Provides specific message responses for the list view.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AllocBy (see page 3803)	Indicates the number of items the list view can store in memory.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 3803)	Determines whether the list view displays a single line border.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	ColumnClick (see page 3808)	Determines whether the column header behaves like a button.
	Columns (see page 3808)	Describes the properties of the columns in the list view.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	HideSelection (see page 3812)	Determines whether the list view gives a visual indication of which item is selected when focus shifts to another control.
	HoverTime (see page 3813)	Specifies how long the user must pause with the mouse before an item is selected.
	IconOptions (see page 3813)	Determines how icons displayed in the list view are arranged.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	Items (see page 3814)	Contains the list of items displayed by the list view.
	LargeImages (see page 3817)	Provides a list of icon images to display for each item in the list when ViewStyle (see page 3829) is vsIcon.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	MultiSelect (see page 3820)	Determines whether the user can select more than one list item at a time.
	OwnerData (see page 3820)	Specifies whether the list view control is virtual.
	OwnerDraw (see page 3821)	Specifies whether the list view receives an OnDrawItem (see page 3848) event.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 3821)	Determines whether the user can change the contents of the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.

	ShowColumnHeaders (see page 3824)	Determines whether the column headers are displayed.
	ShowWorkAreas (see page 3824)	Determines whether the visual properties of work areas are displayed.
	SmallImages (see page 3825)	Provides a list of icon images to display for each item in the list when ViewStyle (see page 3829) is not vsIcon.
	SortType (see page 3827)	Determines if and how the items in the list are automatically sorted.
	StateImages (see page 3828)	Specifies additional icon images to display next to each item in the list view.
	Text (see page 5842)	Contains a text string associated with the control.
	ViewStyle (see page 3829)	Determines how the list items are displayed.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AddItem (see page 3773)	Adds an item to the list control.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AlphaSort (see page 3774)	Sorts the items in the list view.
	Arrange (see page 3777)	Arranges icons in a list view that has a ViewStyle (see page 3829) of vsIcon or vsSmallIcon.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 3779)	Deletes all of the items from the list control.
V	ClearSelection (see page 3779)	Removes the selection, leaving all items unselected.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	CopySelection (see page 3781)	Copies the selected item or items to another TCustomListControl descendant.
V	Create (see page 3770)	Creates and initializes an instance of TCustomListView (see page 3757).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CustomSort (see page 3784)	Sorts the items in the list using the specified ordering function.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	DeleteSelected (see page 3786)	Deletes all the selected items.

	Destroy (see page 3771)	Destroys an instance of TCustomListView (see page 3757).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindCaption (see page 3788)	Returns a list view item, if any, with the specified caption.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindData (see page 3789)	Returns a list view item, if any, associated with the specified data.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHitTestInfoAt (see page 3790)	Returns information about the specified point in the list view's client area.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItemAt (see page 3790)	Returns the list item at the specified position.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNearestItem (see page 3791)	Returns the list item nearest a specified point, in a particular direction.
	GetNextItem (see page 3792)	Returns the next list item after the StartItem in the specified direction.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetSearchString (see page 3793)	Returns the incremental search string of a list view control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsEditing (see page 3795)	Indicates whether any item in the list view is being edited.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	Scroll (see page 3798)	Scrolls the content of a list view by the indicated amount.
	ScrollBy (see page 6176)	Scroll control contents.
V	SelectAll (see page 3799)	Selects all items or all text in the selected item.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.

	StringWidth (see page 3800)	Returns the width of a string, using the list view's current font.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UpdateItems (see page 3801)	Redraws a range of items, starting at FirstIndex and ending at LastIndex.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
R	BoundingRect (see page 3804)	Denotes the bounding rectangle for the items in the list view.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 3804)	Gives access to the list view's canvas.
	Checkboxes (see page 3805)	Specifies whether check boxes appear next to the items in the list view.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	Column (see page 3806)	Describes the column specified by the Index parameter.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropTarget (see page 3810)	Specifies which item in the list is the target of a drag and drop operation.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.

R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
	FlatScrollBars (see page 3810)	Specifies whether the list view's scroll bars are flat.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	FullDrag (see page 3811)	Specifies whether headers are repainted when they are dragged.
	GridLines (see page 3811)	Determines whether lines are drawn separating items in the list.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	HotTrack (see page 3812)	Specifies whether list items are highlighted when the mouse passes over them.
	HotTrackStyles (see page 3812)	Specifies the hot tracking styles.
	ItemFocused (see page 3813)	Indicates which item, if any, has focus.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	RowSelect (see page 3821)	Determines whether an entire row in the list view can be selected.
R	SelCount (see page 3822)	Specifies the number of selected items in the list view.
	Selected (see page 3822)	Indicates the first selected item in the list view.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
R	TopItem (see page 3828)	Indicates the topmost visible item in the list view.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	ViewOrigin (see page 3829)	Designates the coordinates of the upper left corner of the list, relative to the visible area of the list view control.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.
R	VisibleRowCount (see page 3832)	Indicates number of items that can fit vertically in the visible area of a list view.
R	WorkAreas (see page 3833)	Defines separate working areas that divide the list view's client area.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.

	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.19.2 TCustomListView.Create Constructor

Creates and initializes an instance of TCustomListView (see page 3757).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomListView(TComponent * AOwner);
```

Description

Call Create to instantiate a list view at runtime. List view controls placed on forms at design time are created automatically.

C++ Examples:

```
/*
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses defs file.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const char Names[6][2][10] =
   {{"Rubble","Barny"},
    {"Michael", "Johnson"},
    {"Bunny", "Bugs"},
    {"Silver", "HiHo"},
    {"Simpson", "Bart"},
    {"Squirrel", "Rocky"}};

  TListColumn  *NewColumn;
  TListItem  *ListItem;
  TListView   *ListView = new TListView(this);

  ListView->Parent = this;
  ListView->Align = alClient;
  ListView->ViewStyle = vsReport;
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "Last";
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "First";
  for (int i = 0; i < 6; i++)
  {
    ListItem = ListView->Items->Add();
    ListItem->Caption = Names[i][0];
    ListItem->SubItems->Add(Names[i][1]);
  }
}
```

Delphi Examples:

```
{
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses clause of the unit file.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
const
  Names: array[0..5, 0..1] of string = (
    ('Rubble', 'Barney'),
    ('Michael', 'Johnson'),
    ('Bunny', 'Bugs'),
    ('Silver', 'HiHo'),
    ('Simpson', 'Bart'),
    ('Squirrel', 'Rocky')
    );

var
  I: Integer;
  NewColumn: TListColumn;
  ListItem: TListItem;
  ListView: TListView;
begin
  ListView := TListView.Create(Self);
  with ListView do
  begin
    Parent := Self;
    Align := alClient;
    ViewStyle := vsReport;

    NewColumn := Columns.Add;
    NewColumn.Caption := 'Last';
    NewColumn := Columns.Add;
    NewColumn.Caption := 'First';

    for I := Low(Names) to High(Names) do
    begin
      ListItem := Items.Add;
      ListItem.Caption := Names[I][0];
      ListItem.SubItems.Add(Names[I][1]);
    end;
  end;
end;
```

See Also

Owner (see page 2807)

Create (see page 6123)

ParentWindow (see page 6210)

27.1.19.3 TCustomListView.Destroy Destructor

Destroys an instance of TCustomListView (see page 3757).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomListView();
```

Description

Do not call Destroy directly. Instead, call Free, which checks that the list item is not nil before calling Destroy

See Also

Free (see page 22300)

27.1.19.4 TCustomListView Methods

The methods of the TCustomListView class are listed here.

Protected Methods

	Name	Description
	ActionChange (see page 3773)	Responds when the action for which this list view is a client changes.
	CanChange (see page 3777)	Specifies whether the properties of a list item can be changed.
	CanEdit (see page 3778)	Specifies whether the text of a list item can be changed by the user.
	Change (see page 3778)	Generates an OnChange (see page 3836) event.
	ChangeScale (see page 3778)	Rescaling event dispatcher
	ColClick (see page 3780)	Column (see page 3806) click event dispatcher.
	ColRightClick (see page 3780)	Column (see page 3806) right-click event dispatcher.
	ColumnsShowing (see page 3780)	Indicates whether the ViewStyle (see page 3829) property is vsReport.
	CreateListItem (see page 3781)	Creates a list item for the collection specified by the Items (see page 3814) property.
	CreateListItems (see page 3782)	Creates the TListItems (see page 4270) object that implements the Items (see page 3814) property.
	CreateParams (see page 3782)	Initializes the window-creation parameter record.
	CreateWnd (see page 3782)	Creates the window used by the list view.
	CustomDraw (see page 3783)	Custom draw event dispatcher.
	CustomDrawItem (see page 3783)	Custom draw item event dispatcher.
	CustomDrawSubItem (see page 3783)	Custom draw subitem event dispatcher.
	Delete (see page 3785)	Removes an item from the list view and frees the associated memory.
	DestroyWnd (see page 3786)	Destroys the window referenced by the list view's window handle.
	DoEndDrag (see page 3787)	Drag end event dispatcher.
	DoInfoTip (see page 3787)	InfoTip event dispatcher.
	DoStartDrag (see page 3787)	Start drag event dispatcher..
	DrawItem (see page 3788)	Draw item event dispatcher.
	Edit (see page 3788)	Edit event dispatcher.
	GetActionLinkClass (see page 3789)	Returns the action link class that is associated with list view controls.
	GetCount (see page 3789)	Returns the number of items in the list.
	GetDragImages (see page 3790)	Returns a drag cursor if a list item is being dragged.
	GetItemIndex (see page 3791)	This is the overview for the GetItemIndex method overload.
	GetSelCount (see page 3794)	Returns the value of the SelCount (see page 3822) property.
	InsertItem (see page 3794)	Generates an OnInsert (see page 3851) event.
	IsCustomDrawn (see page 3794)	Determines when custom draw event handlers are called.
	MouseUp (see page 3795)	Mouse button up event dispatcher.
	Notification (see page 3796)	Responds to notifications that components are about to be created or destroyed.
	OwnerDataFetch (see page 3796)	Customizes a list view item before it is displayed.
	OwnerDataFind (see page 3797)	Searches for an item in the list view control.
	OwnerDataHint (see page 3797)	Specifies a range of items that might be fetched.
	OwnerDataStateChange (see page 3798)	Specifies items that have changed states.
	SetItemIndex (see page 3799)	Sets the value of the ItemIndex property.
	SetMultiSelect (see page 3800)	Sets the value of the MultiSelect (see page 3820) property.
	SetViewStyle (see page 3800)	Sets the value of the ViewStyle (see page 3829) property.
	UpdateColumn (see page 3801)	Individual column update event dispatcher.
	UpdateColumns (see page 3801)	General column update event dispatcher.
	WndProc (see page 3802)	Provides specific message responses for the list view.

Public Methods

	Name	Description
	AddItem (see page 3773)	Adds an item to the list control.

	AlphaSort (see page 3774)	Sorts the items in the list view.
	Arrange (see page 3777)	Arranges icons in a list view that has a ViewStyle (see page 3829) of vsIcon or vsSmallIcon.
V	Clear (see page 3779)	Deletes all of the items from the list control.
V	ClearSelection (see page 3779)	Removes the selection, leaving all items unselected.
V	CopySelection (see page 3781)	Copies the selected item or items to another TCustomListControl descendant.
	CustomSort (see page 3784)	Sorts the items in the list using the specified ordering function.
V	DeleteSelected (see page 3786)	Deletes all the selected items.
	FindCaption (see page 3788)	Returns a list view item, if any, with the specified caption.
	FindData (see page 3789)	Returns a list view item, if any, associated with the specified data.
	GetHitTestInfoAt (see page 3790)	Returns information about the specified point in the list view's client area.
	GetItemAt (see page 3790)	Returns the list item at the specified position.
	GetNearestItem (see page 3791)	Returns the list item nearest a specified point, in a particular direction.
	GetNextItem (see page 3792)	Returns the next list item after the StartItem in the specified direction.
	GetSearchString (see page 3793)	Returns the incremental search string of a list view control.
	IsEditing (see page 3795)	Indicates whether any item in the list view is being edited.
	Scroll (see page 3798)	Scrolls the content of a list view by the indicated amount.
V	SelectAll (see page 3799)	Selects all items or all text in the selected item.
	StringWidth (see page 3800)	Returns the width of a string, using the list view's current font.
	UpdateItems (see page 3801)	Redraws a range of items, starting at FirstIndex and ending at LastIndex.

27.1.19.4.1 TCustomListView.ActionChange Method

Responds when the action for which this list view is a client changes.

Pascal

```
procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); override;
```

C++

```
virtual __fastcall ActionChange(TObject * Sender, Boolean CheckDefaults);
```

Description

ActionChange is called automatically when the action object specified by the Action property, or any of its properties, changes.

In TCustomListView (see page 3757), ActionChangechecks whether the action is a TStaticActionList object, and if so, updates the action's SmallImages (see page 3825) and ItemIndex properties, as appropriate. It then calls the inherited method, which updates the Caption, Enabled, Hint, and Visible properties and the OnClick event handler.

The Sender parameter is the action object that changed. The CheckDefaults parameter indicates whether the control's properties should all be updated or only those that have not been explicitly changed from their default values. When CheckDefaults is true, only those properties and event handlers that have not been changed from the default values are updated.

See Also

OnChange (see page 2725)

Action (see page 5797)

27.1.19.4.2 TCustomListView.AddItem Method

Adds an item to the list control.

Pascal

```
procedure AddItem(Item: String; AObject: TObject); override;
```

C++

```
virtual __fastcall AddItem(AnsiString Item, TObject * AObject);
```

Description

AddItem adds an item to the end of the list view, possibly associating it with a specified object.

Item is the caption of the item in the list view. that is the item's value. This becomes the value of the new TListItem (see page 4250) object's Caption property.

AObject is an object that you can associated with the item. It becomes the value of the new TListItem (see page 4250) object's Data property.

See Also

Caption (see page 4258)

Data (see page 4259)

Items (see page 3814)

27.1.19.4.3 TCustomListView.AlphaSort Method

Sorts the items in the list view.

Pascal

```
function AlphaSort: Boolean;
```

C++

```
__fastcall Boolean AlphaSort();
```

Description

Call AlphaSort to sort the items in the manner maintained automatically when the SortType (see page 3827) property is not stNone. If an OnCompare (see page 3841) event handler is assigned, AlphaSort uses that event handler to define the sort order. Otherwise, AlphaSort sorts all list items alphabetically in ascending order. If successful, the method returns true.

C++ Examples:

```
/*
This example shows how to use the OnColumnClick and
OnCompare events of a list view to let users sort the
columns in a report-style list view by clicking on the
column headers. This requires a global variable to keep
track of the column that was clicked.  The OnColumnClick
event handler sets the global variable to indicate the
column to sort and calls AlphaSort:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {
    "C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {
    "Documentation for the C++ icon.",
    "Borland icon.", "Delphi icon."
    };
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
```

```
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  ListView1->ViewStyle = vsReport;
}

int ColumnToSort = 0;

/*
The OnColumnClick event handler sets the global variable to
indicate the column to sort and calls AlphaSort:
*/

void __fastcall TForm1::ListView1ColumnClick(TObject *Sender,
      TListColumn *Column)
{
  ColumnToSort = Column->Index;
  ((TCustomListView *)Sender)->AlphaSort();
}

/*
The OnCompare event handler causes the list view to sort on
the selected column:
*/

void __fastcall TForm1::ListView1Compare(TObject *Sender, TListItem *Item1,
      TListItem *Item2, int Data, int &Compare)
{
  if (ColumnToSort == 0)
    Compare = CompareText(Item1->Caption,Item2->Caption);
  else
  {
    int ix = ColumnToSort - 1;
    Compare =
      CompareText(Item1->SubItems->Strings[ix], Item2->SubItems->Strings[ix]);
  }
}
```

Delphi Examples:

```
{
This example shows how to use the OnColumnClick and
OnCompare events of a list view to let users sort the
columns in a report-style list view by clicking on the
column headers. This requires a global variable to keep
track of the column that was clicked.  The OnColumnClick
event handler sets the global variable to indicate the
column to sort and calls AlphaSort:
}
var ColumnToSort: Integer;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
const
  imagenames: Array[0..2] of string = ('C++ Image', 'Borland Image', 'Delphi Image');
```

```
begin
  Col2Array[0]:= 'Documentation for the C++ icon.';
  Col2Array[1]:= 'Borland icon.';
  Col2Array[2]:= 'Delphi icon.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := imagenames[I];
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
    NewColumn.Width := 200;
    ListView1.ViewStyle := vsReport;
  end;
end;

procedure TForm1.ListView1ColumnClick(Sender: TObject; Column: TListColumn);
begin
  ColumnToSort := Column.Index;
  (Sender as TCustomListView).AlphaSort;
end;

{
The OnCompare event handler causes the list view to sort on
the selected column:
}
procedure TForm1.ListView1Compare(Sender: TObject; Item1, Item2: TListItem; Data: Integer; var
Compare: Integer);
var
  ix: Integer;
begin
  if ColumnToSort = 0 then
    Compare := CompareText(Item1.Caption,Item2.Caption)
  else begin
   ix := ColumnToSort - 1;
   Compare := CompareText(Item1.SubItems[ix],Item2.SubItems[ix]);
  end;
end;
{
Note:   This OnCompare event handler uses the global
CompareText function. An application may want to use
AnsiCompareText, CompareStr, or AnsiCompareStr instead,
depending on whether the comparison should be case-sensitive
and whether the locale should be considered.
}
```

See Also

CustomSort (see page 3784)

OnCompare (see page 3841)

SortType (see page 3827)

27.1.19.4.4 TCustomListView.Arrange Method

Arranges icons in a list view that has a ViewStyle (see page 3829) of vslcon or vsSmallIcon.

Pascal

```
procedure Arrange(Code: TListArrangement);
```

C++

```
__fastcall Arrange(TListArrangement Code);
```

Description

Call Arrange to clean up the arrangement of the icons when the ViewStyle (see page 3829) property is vslcon or vsSmallIcon. The Code parameter specifies how to arrange the icons. These are the possible values for the Code parameter:

Value	Meaning
arAlignBottom	Aligns items along the bottom edge of the list view.
arAlignLeft	Aligns items along the left edge of the window.
arAlignRight	Aligns items along the right edge of the window.
arAlignTop	Aligns items along the top edge of the window.
arDefault	Aligns items according to the style associated with the list view when the window was created. For TCustomListView (see page 3757), this is aligned along the top.
arSnapToGrid	Snaps all icons to the nearest grid position. This value is not supported if the application is running in virtual mode.

See Also

ViewStyle (see page 3829)

27.1.19.4.5 TCustomListView.CanChange Method

Specifies whether the properties of a list item can be changed.

Pascal

```
function CanChange(Item: TListItem; Change: Integer): Boolean; dynamic;
```

C++

```
__fastcall Boolean CanChange(TListItem Item, int Change);
```

Description

CanChange is called automatically when an attempt is made to change the item specified by the Item parameter. The Change (see page 3778) parameter is a Windows-defined constant that specifies the type of change that was attempted. CanChange generates an OnChanging (see page 3837) event to determine whether the change should be permitted. CanChange returns true if the change can proceed. CanChange returns false if the change should be blocked. CanChange always returns true if there is no OnChanging (see page 3837) event handler.

Override CanChange to add additional logic to determine whether a change should proceed or to block the OnChanging (see page 3837) event.

See Also

OnChanging (see page 3837)

27.1.19.4.6 TCustomListView.CanEdit Method

Specifies whether the text of a list item can be changed by the user.

Pascal

```
function CanEdit(Item: TListItem): Boolean; dynamic;
```

C++

```
__fastcall Boolean CanEdit(TListItem Item);
```

Description

CanEdit is called automatically when the user attempts to edit the text of the list item specified by the Item parameter. CanEdit generates an OnEditing (see page 3850) event to determine whether to allow the user to edit the item. CanEdit returns true if the user can proceed. CanEdit returns false if the user should be blocked from editing the item. CanEdit always returns true if there is no OnEditing (see page 3850) event handler.

Override CanEdit to add additional logic to determine whether an item can be edited or to block the OnEditing (see page 3850) event.

See Also

OnEditing (see page 3850)

27.1.19.4.7 TCustomListView.Change Method

Generates an OnChange (see page 3836) event.

Pascal

```
procedure Change(Item: TListItem; Change: Integer); dynamic;
```

C++

```
__fastcall Change(TListItem Item, int Change);
```

Description

Change is called automatically after the properties of the item specified by the Item parameter change. The Change parameter is a Windows-defined constant that specifies the type of change that occurred.

Override Change to perform other actions when the list items change or to block the OnChange (see page 3836) event.

See Also

OnChange (see page 3836)

27.1.19.4.8 TCustomListView.ChangeScale Method

Rescaling event dispatcher

Pascal

```
procedure ChangeScale(M: Integer; D: Integer); override;
```

C++

```
virtual __fastcall ChangeScale(int M, int D);
```

Description

TCustomListView (see page 3757) redefines ChangeScale in order to implement item rescaling.

See Also

ChangeScale (see page 5743)

Height (see page 5825)

Left (see page 5830)

Top (see page 5844)

Width (see page 5849)

27.1.19.4.9 TCustomListView.Clear Method

Deletes all of the items from the list control.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Clear empties the list control so that it contains no items.

See Also

DeleteSelected (see page 3786)

Clear (see page 5940)

27.1.19.4.10 TCustomListView.ClearSelection Method

Removes the selection, leaving all items unselected.

Pascal

```
procedure ClearSelection; override;
```

C++

```
virtual __fastcall ClearSelection();
```

Description

Call ClearSelection to deselect any selected items in the list control. ClearSelection removes the selection, but leaves the items in the list.

See Also

Clear (see page 3779)

CopySelection (see page 3781)

DeleteSelected (see page 3786)

MoveSelection (see page 5942)

SelectAll (see page 3799)

ItemIndex

ClearSelection (see page 5940)

27.1.19.4.11 TCustomListView.ColClick Method

Column (see page 3806) click event dispatcher.

Pascal

```
procedure ColClick(Column: TListColumn); dynamic;
```

C++

```
__fastcall ColClick(TListColumn Column);
```

Description

ColClick is called automatically when the user clicks on a column header if the ColumnClick (see page 3808) and ShowColumnHeaders (see page 3824) properties are true and the ViewStyle (see page 3829) property is vsReport. Override ColClick to take additional action when the user clicks on a column header or to conditionally block the OnColumnClick (see page 3837) event.

See Also

ColumnClick (see page 3808)

OnColumnClick (see page 3837)

ShowColumnHeaders (see page 3824)

ViewStyle (see page 3829)

27.1.19.4.12 TCustomListView.ColRightClick Method

Column (see page 3806) right-click event dispatcher.

Pascal

```
procedure ColRightClick(Column: TListColumn; Point: TPoint); dynamic;
```

C++

```
__fastcall ColRightClick(TListColumn Column, TPoint Point);
```

Description

ColRightClick is called automatically when the user clicks on a column header with the right mouse button if the ShowColumnHeaders (see page 3824) properties is true and the ViewStyle (see page 3829) property is vsReport. Override ColRightClick to take additional action when the user right clicks on a column header or to conditionally block the OnColumnRightClick (see page 3841) event.

See Also

OnColumnRightClick (see page 3841)

ShowColumnHeaders (see page 3824)

ViewStyle (see page 3829)

27.1.19.4.13 TCustomListView.ColumnsShowing Method

Indicates whether the ViewStyle (see page 3829) property is vsReport.

Pascal

```
function ColumnsShowing: Boolean;
```

C++

```
__fastcall Boolean ColumnsShowing();
```

Description

Call ColumnsShowing to determine whether the list view displays its items in columns. ColumnsShowing returns true if the value of ViewStyle (see page 3829) is vsReport, false otherwise.

See Also

ViewStyle (see page 3829)

27.1.19.4.14 TCustomListView.CopySelection Method

Copies the selected item or items to another TCustomListControl descendant.

Pascal

```
procedure CopySelection(Destination: TCustomListControl); override;
```

C++

```
virtual __fastcall CopySelection(TCustomListControl Destination);
```

Description

CopySelection adds all selected items to another TCustomListControl descendant.

Destination is the list control that receives the items.

See Also

ClearSelection (see page 3779)

DeleteSelected (see page 3786)

MoveSelection (see page 5942)

SelectAll (see page 3799)

ItemIndex

27.1.19.4.15 TCustomListView.CreateListItem Method

Creates a list item for the collection specified by the Items (see page 3814) property.

Pascal

```
function CreateListItem: TListItem; virtual;
```

C++

```
virtual __fastcall TListItem CreateListItem();
```

Description

Call CreateListItem to create a TListItem (see page 4250) object that will be added to the Items (see page 3814) collection. CreateListItem creates the list item but does not insert it into the collection.

See Also

Items (see page 3814)

27.1.19.4.16 TCustomListView.CreateListItems Method

Creates the TListItems (see page 4270) object that implements the Items (see page 3814) property.

Pascal

```
function CreateListItems: TListItems; virtual;
```

C++

```
virtual __fastcall TListItems CreateListItems();
```

Description

TCustomListView (see page 3757) calls CreateListItems internally to create the object that implements the Items (see page 3814) property. As implemented in TCustomListView (see page 3757), CreateListItems creates and returns a TListItems (see page 4270) instance. Override this method to use a specialized TListItems (see page 4270) descendant instead.

See Also

Items (see page 3814)

27.1.19.4.17 TCustomListView.CreateParams Method

Initializes the window-creation parameter record.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

CreateParams is called automatically to initialize the window-creation parameters when the list view needs to create its window. CreateParams overrides the inherited method to specify a list view window that implements the property settings of the list view control.

When overriding the CreateParams method, be sure to call the inherited method before changing any of the values in the Params parameter.

See Also

CreateParams (see page 6135)

27.1.19.4.18 TCustomListView.CreateWnd Method

Creates the window used by the list view.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd creates the list view's window, based on the parameter values provided by the CreateParams (see page 3782) method. CreateWnd then sends messages to the newly created window so that it reflects the current property settings of the list view object.

See Also

CreateParams (see page 3782)

27.1.19.4.19 TCustomListView.CustomDraw Method

Custom draw event dispatcher.

Pascal

```
function CustomDraw(const ARect: TRect; Stage: TCustomDrawStage): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CustomDraw(const TRect ARect, TCustomDrawStage Stage);
```

Description

CustomDraw generates an OnCustomDraw (see page 3844) or OnAdvancedCustomDraw (see page 3834) event (depending on the drawing stage and the existence of the event handler). It returns a value indicating whether the list view should continue with the default drawing behavior (true) or not (false).

See Also

OnCustomDraw (see page 3844)

IsCustomDrawn (see page 3794)

OnAdvancedCustomDraw (see page 3834)

27.1.19.4.20 TCustomListView.CustomDrawItem Method

Custom draw item event dispatcher.

Pascal

```
function CustomDrawItem(Item: TListItem; State: TCustomDrawState; Stage: TCustomDrawStage):
Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CustomDrawItem(TListItem Item, TCustomDrawState State,
TCustomDrawStage Stage);
```

Description

CustomDraw (see page 3783) generates an OnCustomDrawItem (see page 3845) or OnAdvancedCustomDrawItem (see page 3835) event (depending on the drawing stage and the existence of the event handler) It returns a value indicating whether the list view should continue by drawing the text of the item (true) or not (false).

See Also

OnCustomDrawItem (see page 3845)

IsCustomDrawn (see page 3794)

OnAdvancedCustomDrawItem (see page 3835)

27.1.19.4.21 TCustomListView.CustomDrawSubItem Method

Custom draw subitem event dispatcher.

Pascal

```
function CustomDrawSubItem(Item: TListItem; SubItem: Integer; State: TCustomDrawState; Stage:
TCustomDrawStage): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CustomDrawSubItem(TListItem Item, int SubItem, TCustomDrawState
State, TCustomDrawStage Stage);
```

Description

CustomDrawSubItem generates an OnCustomDrawSubItem (see page 3846) or OnAdvancedCustomDrawSubItem (see page 3835) event, depending on the current painting stage and the existence of an event handler. It returns a value indicating whether the list view should continue by drawing the text of the subitem (true) or not (false).

See Also

OnCustomDrawSubItem (see page 3846)

OnAdvancedCustomDrawSubItem (see page 3835)

27.1.19.4.22 TCustomListView.CustomSort Method

Sorts the items in the list using the specified ordering function.

Pascal

```
function CustomSort(SortProc: TLVCompare; lParam: Longint): Boolean;
```

C++

```
__fastcall Boolean CustomSort(TLVCompare SortProc, Longint lParam);
```

Description

Call CustomSort to sort the items in the list using the ordering function defined by the SortProc parameter. The SortProc parameter specifies an ordering function that compares the list items passed as lParam1 and lParam2. The ordering function returns an integer that indicates whether lParam1 is the same as lParam2 (return value is 0), lParam1 is greater than lParam2 (return value is greater than 0), or lParam1 is less than lParam2 (return value is less than 0). The lParam parameter of CustomSort is an optional value that is passed as the third parameter to the ordering function.

If the SortProc parameter is nil (Delphi) or NULL (C++), CustomSort compares the list items by generating an OnCompare (see page 3841) event. This allows the OnCompare (see page 3841) event handler to provide different ordering criteria (such as ascending vs. descending order), based on the value of the lParam parameter.

If the ordering function is not provided and there is no OnCompare (see page 3841) event handler, CustomSort sorts items alphabetically by their Caption values.

CustomSort returns true if the list is successfully sorted.

Warning: CustomSort does not work if the application is running in virtual mode.

C++ Examples:

```
/*
The following code orders a list view in reverse
alphabetical order on the click of a button. The callback
function CustomSortProc calls the global CompareText
function and negates its return value.
*/
int __stdcall CustomSortProc(long Item1, long Item2, long ParamSort)
{
```

```
  return -CompareText(((TListItem *)Item1)->Caption,
                      ((TListItem *)Item2)->Caption);
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ListView1->CustomSort(CustomSortProc, 0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *ListItem;
  ListView1->ViewStyle = vsList;
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Apples";
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Oranges";
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Pears";
}
```

Delphi Examples:

```
{
The following code orders a list view in reverse
alphabetical order on the click of a button. The callback
function CustomSortProc calls the global CompareText
function and negates its return value.
}
function CustomSortProc(Item1, Item2: TListItem; ParamSort: integer): integer; stdcall;
begin
  Result := -CompareText(Item1.Caption,Item2.Caption);
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListView1.CustomSort(@CustomSortProc, 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
var ListItem : TListItem;
begin
  ListView1.ViewStyle := vsList;
  ListItem := ListView1.Items.Add;
  ListItem.Caption := 'Apples';
  ListItem := ListView1.Items.Add;
  ListItem.Caption := 'Oranges';
  ListItem := ListView1.Items.Add;
  ListItem.Caption := 'Pears';
end;
```

See Also

AlphaSort (see page 3774)

OnCompare (see page 3841)

27.1.19.4.23 TCustomListView.Delete Method

Removes an item from the list view and frees the associated memory.

Pascal

```
procedure Delete(Item: TListItem); dynamic;
```

C++

```
__fastcall Delete(TListItem Item);
```

Description

Call Delete to remove an item from the list view. Delete generates an OnDeletion (see page 3848) event before freeing the list item.

See Also

OnDeletion (see page 3848)

27.1.19.4.24 TCustomListView.DeleteSelected Method

Deletes all the selected items.

Pascal

```
procedure DeleteSelected; override;
```

C++

```
virtual __fastcall DeleteSelected();
```

Description

Call DeleteSelected to remove all selected items from the list control. DeleteSelected deletes only the item or items that are selected.

See Also

Clear (see page 3779)

CopySelection (see page 3781)

ClearSelection (see page 3779)

MoveSelection (see page 5942)

SelectAll (see page 3799)

ItemIndex

DeleteSelected (see page 5941)

27.1.19.4.25 TCustomListView.DestroyWnd Method

Destroys the window referenced by the list view's window handle.

Pascal

```
procedure DestroyWnd; override;
```

C++

```
virtual __fastcall DestroyWnd();
```

Description

Before the window is destroyed, DestroyWnd saves a copy of the list view in internal storage. The inherited method then frees any device contexts, and destroys the window.

See Also

CreateWnd (see page 3782)

DestroyWnd (see page 6141)

27.1.19.4.26 TCustomListView.DoEndDrag Method

Drag end event dispatcher.

Pascal

```
procedure DoEndDrag(Target: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DoEndDrag(TObject * Target, int X, int Y);
```

Description

After generating an OnEndDrag event, DoEndDrag clears the internal variables used to track which item is the drop target.

See Also

DoStartDrag (see page 3787)

DropTarget (see page 3810)

OnEndDrag (see page 5860)

27.1.19.4.27 TCustomListView.DoInfoTip Method

InfoTip event dispatcher.

Pascal

```
procedure DoInfoTip(Item: TListItem; var InfoTip: string); virtual;
```

C++

```
virtual __fastcall DoInfoTip(TListItem Item, AnsiString InfoTip);
```

Description

DoInfoTip is called automatically when the mouse pauses over the list item specified by the Item parameter. It generates an OnInfoTip (see page 3851) event, allowing the event handler to change the value of the InfoTip parameter.

See Also

OnInfoTip (see page 3851)

27.1.19.4.28 TCustomListView.DoStartDrag Method

Start drag event dispatcher..

Pascal

```
procedure DoStartDrag(var DragObject: TDragObject); override;
```

C++

```
virtual __fastcall DoStartDrag(TDragObject DragObject);
```

Description

DoStartDrag is called automatically when the user starts dragging a list item with the mouse. After generating an OnStartDrag event, DoStartDrag generates a drag image and begins calculating drop target positions.

See Also

DoEndDrag (see page 3787)

DropTarget (see page 3810)

OnStartDrag (see page 5871)

27.1.19.4.29 TCustomListView.DrawItem Method

Draw item event dispatcher.

Pascal

```
procedure DrawItem(Item: TListItem; Rect: TRect; State: TOwnerDrawState); virtual;
```

C++

```
virtual __fastcall DrawItem(TListItem Item, TRect Rect, TOwnerDrawState State);
```

Description

DrawItem generates an OnDrawItem (see page 3848) event if there is an OnDrawItem (see page 3848) event handler. Otherwise, it paints the text of the list view item. This method is only called if the OwnerDraw (see page 3821) property is true.

See Also

OnDrawItem (see page 3848)

OwnerDraw (see page 3821)

27.1.19.4.30 TCustomListView.Edit Method

Edit event dispatcher.

Pascal

```
procedure Edit(const Item: TLVItem); dynamic;
```

C++

```
__fastcall Edit(const TLVItem Item);
```

Description

Edit is called automatically when the user finishes editing the label of a list item. Edit obtains the newly edited value from the Windows list item and generates an OnEdited (see page 3849) event. After allowing the OnEdited (see page 3849) event handler to change the value supplied by the user, Edit sets the Caption property of the list item.

See Also

OnEdited (see page 3849)

Caption (see page 4258)

27.1.19.4.31 TCustomListView.FindCaption Method

Returns a list view item, if any, with the specified caption.

Pascal

```
function FindCaption(StartIndex: Integer; Value: string; Partial: Boolean; Inclusive: Boolean;
Wrap: Boolean): TListItem;
```

C++

```
__fastcall TListItem FindCaption(int StartIndex, AnsiString Value, Boolean Partial, Boolean
Inclusive, Boolean Wrap);
```

Description

Call FindCaption to search for a list view item labeled by the string specified as the Value parameter. If the Inclusive parameter is

true, the list item specified by StartIndex is the first item checked. Otherwise, the search starts on the next item. If the Wrap parameter is true, the search continues at the top of the list if a match has not been found before reaching the bottom. FindCaption returns the first item with the Caption property equal to Value. If the Partial parameter is true, a Caption matches if it starts with the substring Value. If Partial is false, the Caption must match Value exactly. If no item is found, FindCaption returns nil (Delphi) or NULL (C++).

See Also

FindData (see page 3789)

Caption (see page 4258)

27.1.19.4.32 TCustomListView.FindData Method

Returns a list view item, if any, associated with the specified data.

Pascal

```
function FindData(StartIndex: Integer; Value: Pointer; Inclusive: Boolean; Wrap: Boolean):
TListItem;
```

C++

```
__fastcall TListItem FindData(int StartIndex, void * Value, Boolean Inclusive, Boolean Wrap);
```

Description

Call FindData to locate a list view item with its Data property equal to the Value parameter. If the Inclusive parameter is true, the list item specified by StartIndex is the first item checked. Otherwise, the search starts on the next item. If the Wrap parameter is true, the search continues at the top of the list if a match has not been found before reaching the bottom. FindData returns the first item with a Data property equal to the Value parameter. If no item is found, FindData returns nil (Delphi) or NULL (C++).

See Also

FindCaption (see page 3788)

Data (see page 4259)

27.1.19.4.33 TCustomListView.GetActionLinkClass Method

Returns the action link class that is associated with list view controls.

Pascal

```
function GetActionLinkClass: TControlActionLinkClass; override;
```

C++

```
virtual __fastcall TControlActionLinkClass GetActionLinkClass();
```

Description

GetActionLinkClass returns the class type for the action link that manages communication between the list view and the action specified by the Action property. Each Action link class is designed to link specific properties and event handlers of the action to its client, based on the types of properties the client supports.

See Also

Action (see page 5797)

27.1.19.4.34 TCustomListView.GetCount Method

Returns the number of items in the list.

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

Call GetCount to determine the number of items in the list control.

27.1.19.4.35 TCustomListView.GetDragImages Method

Returns a drag cursor if a list item is being dragged.

Pascal

```
function GetDragImages: TDragImageList; override;
```

C++

```
virtual __fastcall TDragImageList GetDragImages();
```

Description

GetDragImages is called automatically to obtain a list of drag images when the user drags the mouse with the button held down. If the user is dragging a list item, GetDragImages returns the drag image list that is created when the user starts dragging a list item. If the user did not start dragging the mouse over a list item, GetDragImages returns nil (Delphi) or NULL (C++).

See Also

DoStartDrag (see page 3787)

27.1.19.4.36 TCustomListView.GetHitTestInfoAt Method

Returns information about the specified point in the list view's client area.

Pascal

```
function GetHitTestInfoAt(X: Integer; Y: Integer): THitTests;
```

C++

```
__fastcall THitTests GetHitTestInfoAt(int X, int Y);
```

Description

Use GetHitTestInfoAt to determine what elements of the list view lie under the point specified by the X and Y parameters.

X is the x-coordinate of the point.

Y is the y-coordinate of the point.

See Also

GetItemAt (see page 3790)

27.1.19.4.37 TCustomListView.GetItemAt Method

Returns the list item at the specified position.

Pascal

```
function GetItemAt(X: Integer; Y: Integer): TListItem;
```

C++

```
__fastcall TListItem GetItemAt(int X, int Y);
```

Description

Call GetItemAt to obtain the TListItem (see page 4250) object for the list item found at the position specified by the X and Y parameters. X and Y specify the coordinates of the position, in pixels, relative to the top left corner of the list view. If there is no item at the location, GetItemAt returns nil (Delphi) or NULL (C++).

See Also

GetNearestItem (see page 3791)

Items (see page 3814)

27.1.19.4.38 GetItemIndex Method

This is the overview for the GetItemIndex method overload.

Overload List

	Name	Description
	TCustomListView.GetItemIndex () (see page 3791)	Returns the index of the specified item in the Items (see page 3814) property array.
	TCustomListView.GetItemIndex (TListItem) (see page 3791)	Returns the index of the specified item in the Items (see page 3814) property array.

27.1.19.4.38.1 TCustomListView.GetItemIndex Method ()

Returns the index of the specified item in the Items (see page 3814) property array.

Pascal

```
function GetItemIndex(Value: TListItem): Integer; overload; reintroduce;
function GetItemIndex: Integer; override; overload; reintroduce;
```

C++

```
__fastcall int GetItemIndex(TListItem Value);
virtual __fastcall int GetItemIndex();
```

Description

Call GetItemIndex to determine the position of a list item, where 0 is the index of the first item, 1 is the index of the second item, and so on. If the item specified by the Value parameter is not in the list view, GetItemIndex returns -1. If the Value parameter is omitted, the index of the Selected (see page 3822) item is returned.

See Also

Items (see page 3814)

GetItemIndex (see page 5942)

Selected (see page 3822)

27.1.19.4.39 TCustomListView.GetNearestItem Method

Returns the list item nearest a specified point, in a particular direction.

Pascal

```
function GetNearestItem(Point: TPoint; Direction: TSearchDirection): TListItem;
```

C++

```
__fastcall TListItem GetNearestItem(TPoint Point, TSearchDirection Direction);
```

Description

Call GetNearestItem to locate a list item from a position specified in pixel coordinates relative to the top left corner of the list view. GetNearestItem starts looking at the position specified by the Point parameter, and moves in the direction indicated by the Direction parameter until it locates a list item.

If no item is found, GetNearestItem returns nil (Delphi) or NULL (C++).

See Also

GetItemAt (see page 3790)

GetNextItem (see page 3792)

Items (see page 3814)

27.1.19.4.40 TCustomListView.GetNextItem Method

Returns the next list item after the StartItem in the specified direction.

Pascal

```
function GetNextItem(StartItem: TListItem; Direction: TSearchDirection; States: TItemStates):
TListItem;
```

C++

```
__fastcall TListItem GetNextItem(TListItem StartItem, TSearchDirection Direction, TItemStates
States);
```

Description

Call GetNextItem to find the next list item after StartItem in the direction given by the Direction parameter. Only items in the state indicated by the States parameter are considered.

C++ Examples:

```
/*
This example demonstrates how to retrieve all selected items
in a ListView component and add the Caption of the selected
ListItems to a ListBox component.
Add a button, a listview, and a listbox to the form.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *ListItem;
  ListView1->ViewStyle = vsList;
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Apples";
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Oranges";
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Pears";
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
   TItemStates selected = TItemStates() << isSelected;
   TListItem *Item = ListView1->Selected;
```

```
    while (Item){
       ListBox1->Items->Add(Item->Caption);
       Item = ListView1->GetNextItem(Item, sdAll, selected);
    }
}
```

Delphi Examples:

```
{
This example demonstrates how to retrieve all selected items
in a ListView component and add the Caption of the selected
ListItems to a ListBox component.
Add a button, a listview, and a listbox to the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Item: TListItem;
begin
  Item := ListView1.Selected;
  while Item <> nil do
  begin
    ListBox1.Items.Add(Item.Caption);
    Item := ListView1.GetNextItem(Item, sdAll, [isSelected]);
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var ListItem : TListItem;
begin
  ListView1.ViewStyle := vsList;
  ListItem := ListView1.Items.Add;
  ListItem.Caption := 'Apples';
  ListItem := ListView1.Items.Add;
  ListItem.Caption := 'Oranges';
  ListItem := ListView1.Items.Add;
  ListItem.Caption := 'Pears';
end;
```

See Also

DropTarget (see page 3810)

GetNearestItem (see page 3791)

ItemFocused (see page 3813)

Selected (see page 3822)

27.1.19.4.41 TCustomListView.GetSearchString Method

Returns the incremental search string of a list view control.

Pascal

```
function GetSearchString: string;
```

C++

```
__fastcall AnsiString GetSearchString();
```

Description

Call GetSearchString to obtain the character sequence typed by the user while the list view has input focus. Each time the user types a character, the list view appends the character to the search string and searches for a matching item. A timeout period of less than a second is associated with each character that the user types. If the timeout period elapses before the user types another character, the incremental search string is reset. If the system finds a match, it selects the item and, if necessary, scrolls

it into view.

See Also

Selected (see page 3822)

Caption (see page 4258)

27.1.19.4.42 TCustomListView.GetSelCount Method

Returns the value of the SelCount (see page 3822) property.

Pascal

```
function GetSelCount: Integer; override;
```

C++

```
virtual __fastcall int GetSelCount();
```

Description

Override GetSelCount to provide the read implementation for the SelCount (see page 3822) property. GetSelCount returns the number of items that are currently selected.

See Also

SelCount (see page 3822)

27.1.19.4.43 TCustomListView.InsertItem Method

Generates an OnInsert (see page 3851) event.

Pascal

```
procedure InsertItem(Item: TListItem); dynamic;
```

C++

```
__fastcall InsertItem(TListItem Item);
```

Description

List view objects call InsertItem internally when new items are added to the Items (see page 3814) property list. Override the InsertItem method to cause descendant classes to take specific action in addition to this event when new items are inserted, or to block the OnInsert (see page 3851) event from occurring.

See Also

Items (see page 3814)

OnInsert (see page 3851)

27.1.19.4.44 TCustomListView.IsCustomDrawn Method

Determines when custom draw event handlers are called.

Pascal

```
function IsCustomDrawn(Target: TCustomDrawTarget; Stage: TCustomDrawStage): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsCustomDrawn(TCustomDrawTarget Target, TCustomDrawStage Stage);
```

Description

TCustomListView (see page 3757) checks IsCustomDrawn to determine when to generate the OnCustomDraw (see page 3844), OnAdvancedCustomDraw (see page 3834), OnCustomDrawItem (see page 3845), OnAdvancedCustomDrawItem (see page 3835), OnCustomDrawSubItem (see page 3846), and OnAdvancedCustomDrawSubItem (see page 3835) events. IsCustomDrawn returns true when the list view should generate an event.

The Target parameter indicates whether the list view is about to draw the entire control, an item, or a subitem. IsCustomDrawn does not return true unless the list view has an appropriate event handler for the target.

The Stage parameter indicates the current stage in the process of painting the list view. IsCustomDrawn returns true if the stage is cdPrePaint or if there is an Advanced custom draw event handler for the Target.

See Also

OnCustomDraw (see page 3844)

OnCustomDrawItem (see page 3845)

OnCustomDrawSubItem (see page 3846)

OnAdvancedCustomDraw (see page 3834)

OnAdvancedCustomDrawItem (see page 3835)

OnAdvancedCustomDrawSubItem (see page 3835)

27.1.19.4.45 TCustomListView.IsEditing Method

Indicates whether any item in the list view is being edited.

Pascal

```
function IsEditing: Boolean;
```

C++

```
__fastcall Boolean IsEditing();
```

Description

Call IsEditing to determine whether the user is editing the caption of any of the items in the list. IsEditing returns true if the user is editing an item, false otherwise.

See Also

OnEdited (see page 3849)

OnEditing (see page 3850)

Caption (see page 4258)

27.1.19.4.46 TCustomListView.MouseUp Method

Mouse button up event dispatcher.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

TCustomListView (see page 3757) overrides the protected MouseUp method to suppress the OnMouseUp event when a click is not on a list item.

MouseUp is called internally in response to any of the Windows mouse-up messages (WM_LBUTTONUP, WM_MBUTTONUP, WM_RBUTTONUP), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter depends on which of the Windows messages triggered the event.

See Also

MouseUp (see page 5774)

OnMouseUp (see page 5867)

27.1.19.4.47 TCustomListView.Notification Method

Responds to notifications that components are about to be created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

After calling the inherited method, Notification checks whether the image list specified by the LargeImages (see page 3817), SmallImages (see page 3825), or StateImages (see page 3828) property is about to be destroyed. If so, Notification sets the appropriate property to nil (Delphi) or NULL (C++).

See Also

LargeImages (see page 3817)

SmallImages (see page 3825)

StateImages (see page 3828)

27.1.19.4.48 TCustomListView.OwnerDataFetch Method

Customizes a list view item before it is displayed.

Pascal

```
function OwnerDataFetch(Item: TListItem; Request: TItemRequest): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean OwnerDataFetch(TListItem Item, TItemRequest Request);
```

Description

OwnerDataFetch initializes an item before it is drawn in a list view control. The Item parameter specifies the item that is about to be drawn. The Request parameter indicates the type of information that should be initialized. OwnerDataFetch returns true if the specified item should be displayed, and false if the item should not be drawn.

As implemented in TCustomListView (see page 3757), OwnerDataFetch generates an OnData (see page 3846) event, passing the Item parameter to the event handler and ignoring the Request parameter.

OwnerDataFetch is called only if OwnerData (see page 3820) is true.

See Also

OnData (see page 3846)

OwnerData (see page 3820)

OwnerDataFetch

27.1.19.4.49 TCustomListView.OwnerDataFind Method

Searches for an item in the list view control.

Pascal

```
function OwnerDataFind(Find: TItemFind; const FindString: string; const FindPosition: TPoint;
FindData: Pointer; StartIndex: Integer; Direction: TSearchDirection; Wrap: Boolean): Integer;
virtual;
```

C++

```
virtual __fastcall int OwnerDataFind(TItemFind Find, const AnsiString FindString, const TPoint
FindPosition, void * FindData, int StartIndex, TSearchDirection Direction, Boolean Wrap);
```

Description

Override OwnerDataFind in a descendant class to implement the searching capabilities of a virtual list view control (that is, when OwnerData (see page 3820) is true).

OwnerDataFind is called when the virtual list view must find an item; for example, OwnerDataFind is called when a user attempts an incremental search.

Find specifies the type of search to perform (any of ifData, ifPartialString, ifExactString, ifNearest).

FindString specifies the item text to search for. FindString is ignored unless Find contains ifPartialString or ifExactString.

FindPosition specifies the point from which to start the search. FindPosition is ignored unless Find contains ifNearest

FindData (see page 3789) specifies the item data to search for. FindData (see page 3789) is ignored unless Find contains ifData.

StartIndex specifies the index of the item from which the search should start.

Direction specifies the direction for the search (sdLeft, sdRight, sdAbove, sdBelow, or sdAll).

Wrap specifies whether the search should continue from the start of the list view control if no match is found.

OwnerDataFind should return the index of the found item. If no item is found, OwnerDataFind should return -1.

See Also

OnDataFind (see page 3847)

OwnerData (see page 3820)

27.1.19.4.50 TCustomListView.OwnerDataHint Method

Specifies a range of items that might be fetched.

Pascal

```
function OwnerDataHint(StartIndex: Integer; EndIndex: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean OwnerDataHint(int StartIndex, int EndIndex);
```

Description

Override OwnerDataHint in a derived class to customize a range of items before OwnerDataFetch (see page 3796) is called on them.

This method of customizing items is more efficient than individually updating items in the OwnerDataFetch (see page 3796) method. However, the range of items does not always match what is handled by the OwnerDataFetch (see page 3796) method; therefore, you must be prepared to update individual items in the OwnerDataFetch (see page 3796) method as well.

StartIndex is the index of the first item that might be fetched.

EndIndex is the index of the last item that might be fetched.

OwnerDataHint is called only if OwnerData (see page 3820) is true.

See Also

OnDataHint (see page 4357)

OwnerDataFetch

OwnerData (see page 4325)

27.1.19.4.51 TCustomListView.OwnerDataStateChange Method

Specifies items that have changed states.

Pascal

```
function OwnerDataStateChange(StartIndex: Integer; EndIndex: Integer; OldState: TItemStates;
NewState: TItemStates): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean OwnerDataStateChange(int StartIndex, int EndIndex, TItemStates
OldState, TItemStates NewState);
```

Description

Override OwnerDataStateChange in a derived class to update items in a virtual list view control with their new states.

StartIndex is the index of the first item that has changed.

EndIndex is the index of the last item that has changed.

OldState is the old state of the items.

NewState is the new state of the items.

OwnerDataStateChange is called only if OwnerDataFetch (see page 3796) is true.

See Also

OnDataStateChange (see page 3848)

27.1.19.4.52 TCustomListView.Scroll Method

Scrolls the content of a list view by the indicated amount.

Pascal

```
procedure Scroll(DX: Integer; DY: Integer);
```

C++

```
__fastcall Scroll(int DX, int DY);
```

Description

Call Scroll to scroll the content of a list view by DX pixels horizontally and DY pixels vertically.

See Also

TopItem (see page 3828)

ViewOrigin (see page 3829)

27.1.19.4.53 TCustomListView.SelectAll Method

Selects all items or all text in the selected item.

Pascal

```
procedure SelectAll; override;
```

C++

```
virtual __fastcall SelectAll();
```

Description

SelectAll selects all items in the view.

See Also

Clear (see page 3779)

CopySelection (see page 3781)

ClearSelection (see page 3779)

DeleteSelected (see page 3786)

MoveSelection (see page 5942)

ItemIndex

SelectAll (see page 5943)

27.1.19.4.54 TCustomListView.SetItemIndex Method

Sets the value of the ItemIndex property.

Pascal

```
procedure SetItemIndex(const Value: Integer); override;
```

C++

```
virtual __fastcall SetItemIndex(const int Value);
```

Description

Override SetItemIndex to provide the write implementation for the ItemIndex property.

Value represents the index of the item to select, where the first item in the list has index 0, the second item has index 1, and so on. If Value is -1, SetItemIndex deselects all items (the same as the ClearSelection (see page 3779) method).

See Also

ItemIndex

GetItemIndex (see page 3791)

ClearSelection (see page 3779)

27.1.19.4.55 TCustomListView.SetMultiSelect Method

Sets the value of the MultiSelect (see page 3820) property.

Pascal

```
procedure SetMultiSelect(Value: Boolean); override;
```

C++

```
virtual __fastcall SetMultiSelect(Boolean Value);
```

Description

Override SetMultiSelect to provide the write implementation for the MultiSelect (see page 3820) property.

Value is true if the control should allow the user to select more than one item at a time, false if the control should support at most one selected item.

See Also

MultiSelect (see page 3820)

27.1.19.4.56 TCustomListView.SetViewStyle Method

Sets the value of the ViewStyle (see page 3829) property.

Pascal

```
procedure SetViewStyle(Value: TViewStyle); virtual;
```

C++

```
virtual __fastcall SetViewStyle(TViewStyle Value);
```

Description

Override SetViewStyle to provide a different write implementation for the ViewStyle (see page 3829) property.

Value indicates the new value of ViewStyle (see page 3829), which controls how the list view displays its items.

See Also

ViewStyle (see page 3829)

27.1.19.4.57 TCustomListView.StringWidth Method

Returns the width of a string, using the list view's current font.

Pascal

```
function StringWidth(S: string): Integer;
```

C++

```
__fastcall int StringWidth(AnsiString S);
```

Description

Call StringWidth to determine the space, in pixels, required to display a string using the list view's current font.

See Also

Font (see page 5823)

27.1.19.4.58 TCustomListView.UpdateColumn Method

Individual column update event dispatcher.

Pascal

```
procedure UpdateColumn(AnIndex: Integer);
```

C++

```
__fastcall UpdateColumn(int AnIndex);
```

Description

UpdateColumn is called automatically when the display properties of a column in the list view change. UpdateColumn adjusts the image of the column to match the properties of the TListColumn (see page 4236) object indexed by the AnIndex parameter in the Columns (see page 3808) property.

UpdateColumn does nothing if ColumnsShowing (see page 3780) returns false.

See Also

Columns (see page 3808)

ColumnsShowing (see page 3780)

TListColumn (see page 4236)

UpdateColumns (see page 3801)

27.1.19.4.59 TCustomListView.UpdateColumns Method

General column update event dispatcher.

Pascal

```
procedure UpdateColumns;
```

C++

```
__fastcall UpdateColumns();
```

Description

UpdateColumns is called automatically when the properties of several columns in the list view change. UpdateColumn (see page 3801) adjusts the image of every column to match the properties of the corresponding TListColumn (see page 4236) object listed in the Columns (see page 3808) property.

UpdateColumns does nothing if ColumnsShowing (see page 3780) returns false.

See Also

Columns (see page 3808)

ColumnsShowing (see page 3780)

TListColumn (see page 4236)

UpdateColumn (see page 3801)

27.1.19.4.60 TCustomListView.UpdateItems Method

Redraws a range of items, starting at FirstIndex and ending at LastIndex.

Pascal

```
procedure UpdateItems(FirstIndex: Integer; LastIndex: Integer);
```

C++

```
__fastcall UpdateItems(int FirstIndex, int LastIndex);
```

Description

Call UpdateItems after making changes that affect the display image of the list items in the specified range.

See Also

Items (see page 3814)

27.1.19.4.61 TCustomListView.WndProc Method

Provides specific message responses for the list view.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc overrides the inherited method to prevent the list view from redrawing while the items are being updated.

See Also

WndProc (see page 6187)

27.1.19.5 TCustomListView Properties

The properties of the TCustomListView class are listed here.

Protected Properties

	Name	Description
	AllocBy (see page 3803)	Indicates the number of items the list view can store in memory.
	BorderStyle (see page 3803)	Determines whether the list view displays a single line border.
	ColumnClick (see page 3808)	Determines whether the column header behaves like a button.
	Columns (see page 3808)	Describes the properties of the columns in the list view.
	HideSelection (see page 3812)	Determines whether the list view gives a visual indication of which item is selected when focus shifts to another control.
	HoverTime (see page 3813)	Specifies how long the user must pause with the mouse before an item is selected.
	IconOptions (see page 3813)	Determines how icons displayed in the list view are arranged.
	Items (see page 3814)	Contains the list of items displayed by the list view.
	LargeImages (see page 3817)	Provides a list of icon images to display for each item in the list when ViewStyle (see page 3829) is vsIcon.
	MultiSelect (see page 3820)	Determines whether the user can select more than one list item at a time.
	OwnerData (see page 3820)	Specifies whether the list view control is virtual.
	OwnerDraw (see page 3821)	Specifies whether the list view receives an OnDrawItem (see page 3848) event.
	ReadOnly (see page 3821)	Determines whether the user can change the contents of the control.
	ShowColumnHeaders (see page 3824)	Determines whether the column headers are displayed.
	ShowWorkAreas (see page 3824)	Determines whether the visual properties of work areas are displayed.

	SmallImages (see page 3825)	Provides a list of icon images to display for each item in the list when ViewStyle (see page 3829) is not vsIcon.
	SortType (see page 3827)	Determines if and how the items in the list are automatically sorted.
	StateImages (see page 3828)	Specifies additional icon images to display next to each item in the list view.
	ViewStyle (see page 3829)	Determines how the list items are displayed.

Public Properties

	Name	Description
R	BoundingRect (see page 3804)	Denotes the bounding rectangle for the items in the list view.
R	Canvas (see page 3804)	Gives access to the list view's canvas.
	Checkboxes (see page 3805)	Specifies whether check boxes appear next to the items in the list view.
R	Column (see page 3806)	Describes the column specified by the Index parameter.
	DropTarget (see page 3810)	Specifies which item in the list is the target of a drag and drop operation.
	FlatScrollBars (see page 3810)	Specifies whether the list view's scroll bars are flat.
	FullDrag (see page 3811)	Specifies whether headers are repainted when they are dragged.
	GridLines (see page 3811)	Determines whether lines are drawn separating items in the list.
	HotTrack (see page 3812)	Specifies whether list items are highlighted when the mouse passes over them.
	HotTrackStyles (see page 3812)	Specifies the hot tracking styles.
	ItemFocused (see page 3813)	Indicates which item, if any, has focus.
	RowSelect (see page 3821)	Determines whether an entire row in the list view can be selected.
R	SelCount (see page 3822)	Specifies the number of selected items in the list view.
	Selected (see page 3822)	Indicates the first selected item in the list view.
R	TopItem (see page 3828)	Indicates the topmost visible item in the list view.
R	ViewOrigin (see page 3829)	Designates the coordinates of the upper left corner of the list, relative to the visible area of the list view control.
R	VisibleRowCount (see page 3832)	Indicates number of items that can fit vertically in the visible area of a list view.
R	WorkAreas (see page 3833)	Defines separate working areas that divide the list view's client area.

27.1.19.5.1 TCustomListView.AllocBy Property

Indicates the number of items the list view can store in memory.

Pascal

```
property AllocBy: Integer;
```

C++

```
__property int AllocBy;
```

Description

Set AllocBy to the total number of items in the list before adding a large number of items to the list view. AllocBy allows the list view to allocate the memory for items in a single block at once rather than bit by bit as items are added.

See Also

Items (see page 3814)

27.1.19.5.2 TCustomListView.BorderStyle Property

Determines whether the list view displays a single line border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the list view control stands out on a form. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the list view is drawn with a single-line border. If BorderStyle is bsNone, there will be no border.

27.1.19.5.3 TCustomListView.BoundingRect Property

Denotes the bounding rectangle for the items in the list view.

Pascal

```
property BoundingRect: TRect;
```

C++

```
__property TRect BoundingRect;
```

Description

Read BoundingRect to obtain the screen coordinates of a rectangle that encloses all the items in the list view. BoundingRect allows an application to adjust the size of the control to fit the items or to determine whether the mouse is over any of the list items.

Note: To use BoundingRect, the list view's ViewStyle (see page 3829) property must be either vsIcon or vsSmallIcon.

See Also

ViewStyle (see page 3829)

27.1.19.5.4 TCustomListView.Canvas Property

Gives access to the list view's canvas.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use the Canvas property to paint to the list view's canvas during custom draw events such as OnDrawItem (see page 3848), OnCustomDraw (see page 3844), OnCustomDrawItem (see page 3845), OnCustomDrawSubItem (see page 3846), OnAdvancedCustomDraw (see page 3834), OnAdvancedCustomDrawItem (see page 3835), and OnAdvancedCustomDrawSubItem (see page 3835).

C++ Examples:

```
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TListView *myListView;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TListView")
```

```
    {
      myListView = (TListView *)ListBox1->Items->Objects[Index];
      // make sure we are using the TControls canvas and not the from's!
      myListView->Canvas->MoveTo(myListView->ClientRect.Left, myListView->ClientRect.Top);
      myListView->Canvas->LineTo(myListView->ClientRect.Right, myListView->ClientRect.Bottom);
      myListView = NULL;
    }
  }
```

Delphi Examples:

```
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TListView) then
  begin
    myListView:= (ListBox1.Items.Objects[Index] as TListView);
    with myListView.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myListView.Canvas.MoveTo(Left, Top);
      myListView.Canvas.LineTo(Right, Bottom);
    end;
  myListView:= nil;
  end;
```

See Also

OnCustomDraw (see page 3844)

OnCustomDrawItem (see page 3845)

OnCustomDrawSubItem (see page 3846)

OwnerDraw (see page 3821)

OnAdvancedCustomDraw (see page 3834)

OnAdvancedCustomDrawItem (see page 3835)

OnAdvancedCustomDrawSubItem (see page 3835)

OnDrawItem (see page 3848)

27.1.19.5.5 TCustomListView.Checkboxes Property

Specifies whether check boxes appear next to the items in the list view.

Pascal

```
property Checkboxes: Boolean;
```

C++

```
__property Boolean Checkboxes;
```

Description

Set Checkboxes to true to make check boxes appear next to the list items when ViewStyle (see page 3829) is vsList or vsReport. Each line in the list displays a single check box.

See Also

Checked (see page 4258)

ViewStyle (see page 3829)

27.1.19.5.6 TCustomListView.Column Property

Describes the column specified by the Index parameter.

Pascal

```
property Column [Index: Integer]: TListColumn;
```

C++

```
__property TListColumn Column[int Index];
```

Description

Read Column to access the TListColumn (see page 4236) object that describes the column attributes of a column in a report style list view. The Index parameter is the index of the column in the Columns (see page 3808) property array.

Note: Column is only relevant when the value of the ViewStyle (see page 3829) property is vsReport.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", (TObject *)vsIcon);
```

```
  ComboBox1->Items->AddObject("vsList", (TObject *)vsList);
  ComboBox1->Items->AddObject("vsReport", (TObject *)vsReport);
  ComboBox1->Items->AddObject("vsSmallIcon", (TObject *)vsSmallIcon);
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := 'Image' + IntToStr(I);
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
    NewColumn.Width := 200;
    // Add View styles and constants to the Combo Box
    ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
    ComboBox1.Items.AddObject('vsList', TObject(vsList));
    ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
    ComboBox1.Items.AddObject(
      'vsSmallIcon', TObject(vsSmallIcon));
    // Display first item in the Combo Box
    ComboBox1.ItemIndex := 0;
```

```
    end;
  end;

  procedure TComboForm.ComboBox1Click(Sender: TObject);
  begin
    with ComboBox1 do
      ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

  end;
```

See Also

Columns (see page 3808)

ViewStyle (see page 3829)

27.1.19.5.7 TCustomListView.ColumnClick Property

Determines whether the column header behaves like a button.

Pascal

```
property ColumnClick: Boolean;
```

C++

```
__property Boolean ColumnClick;
```

Description

Use ColumnClick to specify whether the OnColumnClick (see page 3837) event should occur when the user clicks on the column header. When ColumnClick is true, the OnColumnClick (see page 3837) occurs when the column header is clicked with the mouse. When ColumnClick is false, the OnColumnClick (see page 3837) event never occurs.

Note: For column clicks to work, the column headers must be displayed. To display the headers, set the ShowColumnHeaders (see page 3824) property to true and the ViewStyle (see page 3829) property to vsReport.

See Also

OnColumnClick (see page 3837)

ShowColumnHeaders (see page 3824)

ViewStyle (see page 3829)

27.1.19.5.8 TCustomListView.Columns Property

Describes the properties of the columns in the list view.

Pascal

```
property Columns: TListColumns;
```

C++

```
__property TListColumns Columns;
```

Description

Use Columns to add or delete columns in the list view, or to edit their display properties. When setting Columns at design time a ListView Columns Editor dialog appears. This dialog allows columns to be added or deleted, and the display properties of individual columns to be changed. At runtime, use the Caption, Alignment, and Width properties of the TListColumn (see page 4236) objects accessed by the Columns property to change the appearance of the column headers.

Note: To display the columns, set the ViewStyle (see page 3829) property to vsReport and the ShowColumnHeaders (see

page 3824) property to true.

C++ Examples:

```
/*
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses defs file.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const char Names[6][2][10] =
   {{"Rubble","Barny"},
    {"Michael", "Johnson"},
    {"Bunny", "Bugs"},
    {"Silver", "HiHo"},
    {"Simpson", "Bart"},
    {"Squirrel", "Rocky"}};

  TListColumn  *NewColumn;
  TListItem  *ListItem;
  TListView   *ListView = new TListView(this);

  ListView->Parent = this;
  ListView->Align = alClient;
  ListView->ViewStyle = vsReport;
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "Last";
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "First";
  for (int i = 0; i < 6; i++)
  {
    ListItem = ListView->Items->Add();
    ListItem->Caption = Names[i][0];
    ListItem->SubItems->Add(Names[i][1]);
  }
}
```

Delphi Examples:

```
{
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses clause of the unit file.
}
procedure TForm1.FormCreate(Sender: TObject);
const
  Names: array[0..5, 0..1] of string = (
    ('Rubble', 'Barney'),
    ('Michael', 'Johnson'),
    ('Bunny', 'Bugs'),
    ('Silver', 'HiHo'),
    ('Simpson', 'Bart'),
    ('Squirrel', 'Rocky')
    );

var
  I: Integer;
  NewColumn: TListColumn;
  ListItem: TListItem;
  ListView: TListView;
begin
  ListView := TListView.Create(Self);
  with ListView do
  begin
```

```
      Parent := Self;
      Align := alClient;
      ViewStyle := vsReport;

      NewColumn := Columns.Add;
      NewColumn.Caption := 'Last';
      NewColumn := Columns.Add;
      NewColumn.Caption := 'First';

      for I := Low(Names) to High(Names) do
      begin
        ListItem := Items.Add;
        ListItem.Caption := Names[I][0];
        ListItem.SubItems.Add(Names[I][1]);
      end;
    end;
  end;
```

See Also

ShowColumnHeaders (see page 3824)

TListColumns (see page 4245)

TListColumn (see page 4236)

ViewStyle (see page 3829)

27.1.19.5.9 TCustomListView.DropTarget Property

Specifies which item in the list is the target of a drag and drop operation.

Pascal

```
property DropTarget: TListItem;
```

C++

```
__property TListItem DropTarget;
```

Description

Read DropTarget to determine whether an item in the list is the target of a drag and drop operation. Set DropTarget to change whether a particular item in the list can be the drop target of a dragged item.

Note: DropTarget only indicates the list item that appears to the user as a drop target. The application must still implement the drag-and-drop behavior for the item.

See Also

OnDragDrop (see page 5855)

TListItem (see page 4250)

DropTarget (see page 4260)

DragKind (see page 5819)

OnDragOver (see page 5857)

27.1.19.5.10 TCustomListView.FlatScrollBars Property

Specifies whether the list view's scroll bars are flat.

Pascal

```
property FlatScrollBars: Boolean;
```

C++

```
__property Boolean FlatScrollBars;
```

Description

Use FlatScrollBars to get or set whether the list view's scroll bars are flat. Flat scroll bars do not have the beveled borders used for the three-dimensional effect.

Note: List views only support flat scroll bars if the system has version 472 or later of comctl32.dll.

27.1.19.5.11 TCustomListView.FullDrag Property

Specifies whether headers are repainted when they are dragged.

Pascal

```
property FullDrag: Boolean;
```

C++

```
__property Boolean FullDrag;
```

Description

Use FullDrag to get or set whether headers are repainted when they are dragged.

Note: To display the column headers, set the ViewStyle (see page 3829) property to vsReport and the ShowColumnHeaders (see page 3824) property to true.

See Also

ViewStyle (see page 3829)

Columns (see page 3808)

ShowColumnHeaders (see page 3824)

OnColumnDragged (see page 3840)

27.1.19.5.12 TCustomListView.GridLines Property

Determines whether lines are drawn separating items in the list.

Pascal

```
property GridLines: Boolean;
```

C++

```
__property Boolean GridLines;
```

Description

Set GridLines to true to add lines that separate the items in the list view. The value of GridLines is ignored unless ViewStyle (see page 3829) is vsReport.

See Also

ViewStyle (see page 3829)

27.1.19.5.13 TCustomListView.HideSelection Property

Determines whether the list view gives a visual indication of which item is selected when focus shifts to another control.

Pascal

```
property HideSelection: Boolean;
```

C++

```
__property Boolean HideSelection;
```

Description

Set HideSelection to false to provide visual feedback of the selected item in the list even when the list view does not have focus. Set HideSelection to true to show the selection only when the list view has focus. HideSelection does not affect the actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms with many list views look too busy.

See Also

ItemFocused (see page 3813)

MultiSelect (see page 3820)

SelCount (see page 3822)

Selected (see page 3822)

27.1.19.5.14 TCustomListView.HotTrack Property

Specifies whether list items are highlighted when the mouse passes over them.

Pascal

```
property HotTrack: Boolean;
```

C++

```
__property Boolean HotTrack;
```

Description

Set HotTrack to true to provide visual feedback about which item is under the mouse. When HotTrack is true, the HoverTime (see page 3813) property controls whether pausing the mouse over an item causes it to be selected. When HotTrack is false, there is no visual feedback when the mouse passes over items and users can't select items by pausing the mouse.

To specify hot tracking styles, use the HotTrackStyles (see page 3812) property.

See Also

HideSelection (see page 3812)

HotTrackStyles (see page 3812)

HoverTime (see page 3813)

27.1.19.5.15 TCustomListView.HotTrackStyles Property

Specifies the hot tracking styles.

Pascal

```
property HotTrackStyles: TListHotTrackStyles;
```

C++

```
__property TListHotTrackStyles HotTrackStyles;
```

Description

Use HotTrackStyles to specify the hot tracking styles. The hot tracking style indicates the type of feedback that the list view gives when the mouse passes over list items. This feedback can be a change of the mouse cursor or underline of the list items.

To turn hot tracking on, set HotTrack (see page 3812) to true.

See Also

HotTrack (see page 3812)

27.1.19.5.16 TCustomListView.HoverTime Property

Specifies how long the user must pause with the mouse before an item is selected.

Pascal

```
property HoverTime: Integer;
```

C++

```
__property int HoverTime;
```

Description

Use HoverTime to indicate how long (in milliseconds) the user must pause with the mouse before an item is selected. If HoverTime is -1 (the default value), a default time period must elapse.

HoverTime has no effect unless the HotTrack (see page 3812) property is true.

See Also

HotTrack (see page 3812)

27.1.19.5.17 TCustomListView.IconOptions Property

Determines how icons displayed in the list view are arranged.

Pascal

```
property IconOptions: TIconOptions;
```

C++

```
__property TIconOptions IconOptions;
```

Description

Set the properties of the TIconOptions (see page 4233) object that is the value of IconOptions to specify how the icons in the list view should be arranged. The ViewStyle (see page 3829) of the list view must be set to either vsIcon or vsSmallIcons for these options to affect the control.

See Also

TIconOptions (see page 4233)

ViewStyle (see page 3829)

27.1.19.5.18 TCustomListView.ItemFocused Property

Indicates which item, if any, has focus.

Pascal

```
property ItemFocused: TListItem;
```

C++

```
__property TListItem ItemFocused;
```

Description

Read ItemFocused to determine which item, if any, can be edited by the user. Set ItemFocused to set focus to an item in the list. When an item has focus, it appears surrounded by a standard focus rectangle. If no item has focus, ItemFocused returns nil (Delphi) or NULL (C++). Only one item can have focus at a time.

See Also

Selected (see page 3822)

27.1.19.5.19 TCustomListView.Items Property

Contains the list of items displayed by the list view.

Pascal

```
property Items: TListItems;
```

C++

```
__property TListItems Items;
```

Description

Use Items to directly access the TListItem (see page 4250) objects that represent the items in the list. Setting this property at design time brings up the ListView Items Editor. Use this dialog to add or delete items or subitems, and to edit their display properties. At runtime, use each item's Caption, ImageIndex, and StateIndex properties to change the appearance of the list items.

Warning: Items is read-only if the application is running in virtual mode.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
```

```
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", (TObject *)vsIcon);
  ComboBox1->Items->AddObject("vsList", (TObject *)vsList);
  ComboBox1->Items->AddObject("vsReport", (TObject *)vsReport);
  ComboBox1->Items->AddObject("vsSmallIcon", (TObject *)vsSmallIcon);
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
/*
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses defs file.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const char Names[6][2][10] =
   {{"Rubble","Barny"},
    {"Michael", "Johnson"},
    {"Bunny", "Bugs"},
    {"Silver", "HiHo"},
    {"Simpson", "Bart"},
    {"Squirrel", "Rocky"}};

  TListColumn  *NewColumn;
  TListItem  *ListItem;
  TListView   *ListView = new TListView(this);

  ListView->Parent = this;
  ListView->Align = alClient;
  ListView->ViewStyle = vsReport;
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "Last";
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "First";
  for (int i = 0; i < 6; i++)
  {
    ListItem = ListView->Items->Add();
    ListItem->Caption = Names[i][0];
    ListItem->SubItems->Add(Names[i][1]);
  }
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
```

```
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := 'Image' + IntToStr(I);
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
    NewColumn.Width := 200;
    // Add View styles and constants to the Combo Box
    ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
    ComboBox1.Items.AddObject('vsList', TObject(vsList));
    ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
    ComboBox1.Items.AddObject(
      'vsSmallIcon', TObject(vsSmallIcon));
    // Display first item in the Combo Box
    ComboBox1.ItemIndex := 0;
  end;
end;

procedure TComboForm.ComboBox1Click(Sender: TObject);
begin
  with ComboBox1 do
    ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

end;
{
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses clause of the unit file.
}
procedure TForm1.FormCreate(Sender: TObject);
```

```
const
  Names: array[0..5, 0..1] of string = (
    ('Rubble', 'Barney'),
    ('Michael', 'Johnson'),
    ('Bunny', 'Bugs'),
    ('Silver', 'HiHo'),
    ('Simpson', 'Bart'),
    ('Squirrel', 'Rocky')
    );

var
  I: Integer;
  NewColumn: TListColumn;
  ListItem: TListItem;
  ListView: TListView;
begin
  ListView := TListView.Create(Self);
  with ListView do
  begin
    Parent := Self;
    Align := alClient;
    ViewStyle := vsReport;

    NewColumn := Columns.Add;
    NewColumn.Caption := 'Last';
    NewColumn := Columns.Add;
    NewColumn.Caption := 'First';

    for I := Low(Names) to High(Names) do
    begin
      ListItem := Items.Add;
      ListItem.Caption := Names[I][0];
      ListItem.SubItems.Add(Names[I][1]);
    end;
  end;
end;
```

See Also

TListItem (see page 4250)

Caption (see page 4258)

ImageIndex (see page 4261)

StateIndex (see page 4267)

27.1.19.5.20 TCustomListView.LargeImages Property

Provides a list of icon images to display for each item in the list when ViewStyle (see page 3829) is vsIcon.

Pascal

```
property LargeImages: TCustomImageList;
```

C++

```
__property TCustomImageList LargeImages;
```

Description

Set LargeImages to specify the bitmaps that should be displayed next to items in the list when ViewStyle (see page 3829) is vsIcon. Each item in the Items (see page 3814) list can be associated with a bitmap in this image list by setting its ImageIndex property.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", (TObject *)vsIcon);
  ComboBox1->Items->AddObject("vsList", (TObject *)vsList);
  ComboBox1->Items->AddObject("vsReport", (TObject *)vsReport);
  ComboBox1->Items->AddObject("vsSmallIcon", (TObject *)vsSmallIcon);
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
```

```
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := 'Image' + IntToStr(I);
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
    NewColumn.Width := 200;
    // Add View styles and constants to the Combo Box
    ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
    ComboBox1.Items.AddObject('vsList', TObject(vsList));
    ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
    ComboBox1.Items.AddObject(
      'vsSmallIcon', TObject(vsSmallIcon));
    // Display first item in the Combo Box
    ComboBox1.ItemIndex := 0;
  end;
end;

procedure TComboForm.ComboBox1Click(Sender: TObject);
begin
  with ComboBox1 do
    ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

end;
```

See Also

Items (see page 3814)

SmallImages (see page 3825)

StateImages (see page 3828)

TImageList (see page 6078)

ViewStyle (see page 3829)

27.1.19.5.21 TCustomListView.MultiSelect Property

Determines whether the user can select more than one list item at a time.

Pascal

```
property MultiSelect: Boolean;
```

C++

```
__property Boolean MultiSelect;
```

Description

Set MultiSelect to specify whether users can select multiple items in the list. If MultiSelect is true, the user can select multiple items. If MultiSelect if false, only one item can be selected at a time.

See Also

SelCount (see page 3822)

Selected (see page 3822)

27.1.19.5.22 TCustomListView.OwnerData Property

Specifies whether the list view control is virtual.

Pascal

```
property OwnerData: Boolean;
```

C++

```
__property Boolean OwnerData;
```

Description

Set OwnerData to true to specify that the list view is virtual. A virtual list view can contain a huge number of items (specifically, up to a DWORD). However, you must manage the items of a virtual list view using the OnData (see page 3846), OnDataFind (see page 3847), OnDataHint (see page 3847), and OnDataStateChange (see page 3848) event handlers. For example, you must explicitly provide a value to a list item's StateIndex property if it is to display a check box.

When creating a virtual list view, you must set the Count property for the Items (see page 3814) to the number of items in the virtual list.

The only information contained in a virtual list view control is the selection and focus information.

To improve the performance of a virtual list view control, cache items for quick retrieval. The OnDataHint (see page 3847) event can help optimize retrieval from the cache.

Note: Virtual list views need not be owner-drawn. The OnData (see page 3846), OnDataFind (see page 3847), OnDataHint (see page 3847), and OnDataStateChange (see page 3848) events enable the list view's default drawing, if desired.

See Also

OnData (see page 3846)

OnDataFind (see page 3847)

OnDataHint (see page 3847)

OnDataStateChange (see page 3848)

OwnerDraw (see page 3821)

Count (see page 4281)

27.1.19.5.23 TCustomListView.OwnerDraw Property

Specifies whether the list view receives an OnDrawItem (see page 3848) event.

Pascal

```
property OwnerDraw: Boolean;
```

C++

```
__property Boolean OwnerDraw;
```

Description

Set OwnerDraw to true to allow the list view to receive the OnDrawItem (see page 3848) event instead of the default rendering of list items.

Note: This property is independent of the custom draw events such as OnCustomDraw (see page 3844), OnCustomDrawItem (see page 3845), OnAdvancedCustomDraw (see page 3834), and so on. The custom draw events always occur.

See Also

OnDrawItem (see page 3848)

Canvas (see page 3804)

IsCustomDrawn (see page 3794)

27.1.19.5.24 TCustomListView.ReadOnly Property

Determines whether the user can change the contents of the control.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to determine whether the list view can be used to enter data. If ReadOnly is true, the list view displays values that the user can't edit. If ReadOnly is false, the user can modify the contents. If ReadOnly is false, the OnEditing (see page 3850) and OnEdited (see page 3849) events occur when the user begins and finishes editing the list item, respectively.

See Also

OnEdited (see page 3849)

OnEditing (see page 3850)

27.1.19.5.25 TCustomListView.RowSelect Property

Determines whether an entire row in the list view can be selected.

Pascal

```
property RowSelect: Boolean;
```

C++

```
__property Boolean RowSelect;
```

Description

Set RowSelect to true to allow entire rows to be selected when the ViewStyle (see page 3829) property is vsReport. To allow multiple rows to be selected, use the MultiSelect (see page 3820) property.

See Also

MultiSelect (see page 3820)

Selected (see page 4267)

ViewStyle (see page 3829)

27.1.19.5.26 TCustomListView.SelCount Property

Specifies the number of selected items in the list view.

Pascal

```
property SelCount: Integer;
```

C++

```
__property int SelCount;
```

Description

Read SelCount to determine the number of items that are currently selected. When SelCount is 1, use the Selected (see page 3822) property to access the selected item. To locate the selected items when SelCount is greater than one, check the Selected (see page 3822) property of the list items, beginning with the item specified by the Selected (see page 3822) property.

Note: SelCount can only exceed 1 if the MultiSelect (see page 3820) property is true.

See Also

Items (see page 3814)

MultiSelect (see page 3820)

Selected (see page 3822)

Selected (see page 4267)

27.1.19.5.27 TCustomListView.Selected Property

Indicates the first selected item in the list view.

Pascal

```
property Selected: TListItem;
```

C++

```
__property TListItem Selected;
```

Description

Read Selected to access the properties of the first selected item in the list. If SelCount (see page 3822) is 0, Selected is nil (Delphi) or NULL (C++). If SelCount (see page 3822) is greater than 1, subsequent selected items can be located by checking the Selected property of the items found using the GetNextItem (see page 3792) method.

Set the Selected property to select an item in the list. If MultiSelect (see page 3820) is true, setting Selected adds an item to the selection. If MultiSelect (see page 3820) is false, setting Selected changes the selection. Setting Selected to nil (Delphi) or NULL (C++) deselects all items in the list.

When an item is selected, the OnChanging (see page 3837) and OnChange (see page 3836) events are generated.

C++ Examples:

```
/*
The following example inserts a list item after the selected
item in the list view.  Place a TButton, a TEdit and a
TListView in the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TListItem *InsertItem;
  if (ListView1->Selected == NULL) return;
  InsertItem =
    ListView1->Items->Insert(ListView1->Selected->Index);
  InsertItem->Caption = Edit1->Text;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *ListItem;
  ListView1->ViewStyle = vsList;
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Apples";
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Oranges";
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Pears";
}
```

Delphi Examples:

```
{
The following example inserts a list item after the selected
item in the list view.  Place a TButton, a TEdit and a
TListView in the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var InsertItem : TListItem;
begin
  with ListView1 do
  begin
    if (Selected = nil) then exit;
    InsertItem := Items.Insert(Selected.Index);
  end;
  InsertItem.Caption := Edit1.Text;
end;

procedure TForm1.FormCreate(Sender: TObject);
var ListItem : TListItem;
begin
  ListView1.ViewStyle := vsList;
  ListItem := ListView1.Items.Add;
  ListItem.Caption := 'Apples';
  ListItem := ListView1.Items.Add;
  ListItem.Caption := 'Oranges';
  ListItem := ListView1.Items.Add;
  ListItem.Caption := 'Pears';
end;
```

See Also

GetNextItem (see page 3792)

MultiSelect (see page 3820)

OnChange (see page 3836)

OnChanging (see page 3837)

SelCount (see page 3822)

27.1.19.5.28 TCustomListView.ShowColumnHeaders Property

Determines whether the column headers are displayed.

Pascal

```
property ShowColumnHeaders: Boolean;
```

C++

```
__property Boolean ShowColumnHeaders;
```

Description

Set ShowColumnHeaders to specify whether the columns in the list view should display column headers. A column header is a panel with a caption that identifies the category of items in the column of the list view. Headers are useful for labeling columns of subitems that provide additional information in a list view. Use the Columns (see page 3808) property to create or add headers.

Note: To use columns in a list view, the ViewStyle (see page 3829) property must be set to vsReport.

See Also

ColumnClick (see page 3808)

Columns (see page 3808)

ViewStyle (see page 3829)

FullDrag (see page 3811)

27.1.19.5.29 TCustomListView.ShowWorkAreas Property

Determines whether the visual properties of work areas are displayed.

Pascal

```
property ShowWorkAreas: Boolean;
```

C++

```
__property Boolean ShowWorkAreas;
```

Description

Set ShowWorkAreas to specify whether the list view should paint each work area in its own color and labeled by its display name. If the list view's WorkAreas (see page 3833) property includes any work areas, the icons in the list view are always grouped visually by these work areas. ShowWorkAreas determines whether the properties of individual work area components are used to make each area stand out visually.

When ShowWorkAreas is true, each work area is painted in the color specified by its Color property and labeled by the string in its DisplayName property. When ShowWorkAreas is false, icons appear grouped by work areas, but there is no other visual indication.

Note: To use work areas in a list view, the ViewStyle (see page 3829) property must be set to vsIcon or vsSmallIcon.

Warning: Work areas are not supported if the OwnerData (see page 3820) property is true.

See Also

Color (see page 5428)

DisplayName (see page 2761)

ViewStyle (see page 3829)

WorkAreas (see page 3833)

OwnerData (see page 3820)

27.1.19.5.30 TCustomListView.SmallImages Property

Provides a list of icon images to display for each item in the list when ViewStyle (see page 3829) is not vsIcon.

Pascal

```
property SmallImages: TCustomImageList;
```

C++

```
__property TCustomImageList SmallImages;
```

Description

Set SmallImages to specify the icons that should be displayed next to items in the list when ViewStyle (see page 3829) is vsSmallIcon, vsList, or vsReport. Each item in the Items (see page 3814) list can be associated with an icon in this image list by setting its ImageIndex property.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
```

```
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", (TObject *)vsIcon);
  ComboBox1->Items->AddObject("vsList", (TObject *)vsList);
  ComboBox1->Items->AddObject("vsReport", (TObject *)vsReport);
  ComboBox1->Items->AddObject("vsSmallIcon", (TObject *)vsSmallIcon);
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := 'Image' + IntToStr(I);
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
    NewColumn.Width := 200;
    // Add View styles and constants to the Combo Box
```

```
      ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
      ComboBox1.Items.AddObject('vsList', TObject(vsList));
      ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
      ComboBox1.Items.AddObject(
        'vsSmallIcon', TObject(vsSmallIcon));
      // Display first item in the Combo Box
      ComboBox1.ItemIndex := 0;
    end;
  end;

  procedure TComboForm.ComboBox1Click(Sender: TObject);
  begin
    with ComboBox1 do
      ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

  end;
```

See Also

Items (see page 3814)

LargeImages (see page 3817)

StateImages (see page 3828)

TImageList (see page 6078)

ViewStyle (see page 3829)

27.1.19.5.31 TCustomListView.SortType Property

Determines if and how the items in the list are automatically sorted.

Pascal

```
property SortType: TSortType;
```

C++

```
__property TSortType SortType;
```

Description

Set SortType to specify the order of items in the list. If SortType is not stNone, the list items in the Items (see page 3814) property are automatically sorted. Once a list view is sorted, the original order is lost. That is, setting the SortType back to stNone will not restore the original order of items.

Note: If SortType is stData or stBoth, provide an OnCompare (see page 3841) event handler to compare data values.

Note: If the list view uses work areas, sorting items only compares each item to the other items in its work area.

See Also

Items (see page 3814)

OnCompare (see page 3841)

Caption (see page 4258)

Data (see page 4259)

WorkAreas (see page 3833)

27.1.19.5.32 TCustomListView.StateImages Property

Specifies additional icon images to display next to each item in the list view.

Pascal

```
property StateImages: TCustomImageList;
```

C++

```
__property TCustomImageList StateImages;
```

Description

Set StateImages to specify the icons that should be displayed next to items in the list. Each item in the Items (see page 3814) list can be associated with an icon in this image list by setting its StateIndex property.

See Also

Items (see page 3814)

LargeImages (see page 3817)

SmallImages (see page 3825)

TImageList (see page 6078)

ViewStyle (see page 3829)

StateIndex (see page 4267)

27.1.19.5.33 TCustomListView.TopItem Property

Indicates the topmost visible item in the list view.

Pascal

```
property TopItem: TListItem;
```

C++

```
__property TListItem TopItem;
```

Description

Read TopItem to determine the first item in the list view that the user can see. TopItem reflects how far the list view has been scrolled. The TopItem property is read-only.

You can scroll the list view with the Scroll (see page 3798) or ScrollBy methods. You can't scroll by setting TopItem, since it is read-only.

TopItem is only implemented when ViewStyle (see page 3829) is vsReport or vsList. For vsIcon and vsSmallIcon view styles, use the ViewOrigin (see page 3829) property to determine the scroll position.

See Also

Scroll (see page 3798)

ScrollBy (see page 6176)

ViewOrigin (see page 3829)

ViewStyle (see page 3829)

27.1.19.5.34 TCustomListView.ViewOrigin Property

Designates the coordinates of the upper left corner of the list, relative to the visible area of the list view control.

Pascal

```
property ViewOrigin: TPoint;
```

C++

```
__property TPoint ViewOrigin;
```

Description

Read ViewOrigin to determine the logical origin of the list image. When the list view is not scrolled, ViewOrigin is (0,0). Otherwise, use the x coordinate to determine the number of pixels the view is scrolled horizontally, and the y coordinate to determine the number of pixels the view is scrolled vertically.

ViewOrigin is only implemented when ViewStyle (see page 3829) is vsIcon or vsSmallIcon. For vsReport or vsList, use the TopItem (see page 3828) property instead.

See Also

TopItem (see page 3828)

ViewStyle (see page 3829)

27.1.19.5.35 TCustomListView.ViewStyle Property

Determines how the list items are displayed.

Pascal

```
property ViewStyle: TViewStyle;
```

C++

```
__property TViewStyle ViewStyle;
```

Description

Set ViewStyle to control the visual display of items in a list view. The items can be displayed as a set of movable icons, or as columns of text.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
```

```
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", (TObject *)vsIcon);
  ComboBox1->Items->AddObject("vsList", (TObject *)vsList);
  ComboBox1->Items->AddObject("vsReport", (TObject *)vsReport);
  ComboBox1->Items->AddObject("vsSmallIcon", (TObject *)vsSmallIcon);
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
/*
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses defs file.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const char Names[6][2][10] =
   {{"Rubble","Barny"},
    {"Michael", "Johnson"},
    {"Bunny", "Bugs"},
    {"Silver", "HiHo"},
    {"Simpson", "Bart"},
    {"Squirrel", "Rocky"}};

  TListColumn  *NewColumn;
  TListItem  *ListItem;
  TListView   *ListView = new TListView(this);

  ListView->Parent = this;
  ListView->Align = alClient;
  ListView->ViewStyle = vsReport;
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "Last";
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "First";
  for (int i = 0; i < 6; i++)
  {
    ListItem = ListView->Items->Add();
    ListItem->Caption = Names[i][0];
    ListItem->SubItems->Add(Names[i][1]);
  }
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := 'Image' + IntToStr(I);
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
    NewColumn.Width := 200;
    // Add View styles and constants to the Combo Box
    ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
    ComboBox1.Items.AddObject('vsList', TObject(vsList));
    ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
    ComboBox1.Items.AddObject(
      'vsSmallIcon', TObject(vsSmallIcon));
    // Display first item in the Combo Box
    ComboBox1.ItemIndex := 0;
  end;
end;

procedure TComboForm.ComboBox1Click(Sender: TObject);
begin
  with ComboBox1 do
    ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

end;
{
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses clause of the unit file.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
const
  Names: array[0..5, 0..1] of string = (
    ('Rubble', 'Barney'),
    ('Michael', 'Johnson'),
    ('Bunny', 'Bugs'),
    ('Silver', 'HiHo'),
    ('Simpson', 'Bart'),
    ('Squirrel', 'Rocky')
    );

var
  I: Integer;
  NewColumn: TListColumn;
  ListItem: TListItem;
  ListView: TListView;
begin
  ListView := TListView.Create(Self);
  with ListView do
  begin
    Parent := Self;
    Align := alClient;
    ViewStyle := vsReport;

    NewColumn := Columns.Add;
    NewColumn.Caption := 'Last';
    NewColumn := Columns.Add;
    NewColumn.Caption := 'First';

    for I := Low(Names) to High(Names) do
    begin
      ListItem := Items.Add;
      ListItem.Caption := Names[I][0];
      ListItem.SubItems.Add(Names[I][1]);
    end;
  end;
end;
```

See Also

IconOptions (see page 3813)

LargeImages (see page 3817)

ShowColumnHeaders (see page 3824)

SmallImages (see page 3825)

StateImages (see page 3828)

27.1.19.5.36 TCustomListView.VisibleRowCount Property

Indicates number of items that can fit vertically in the visible area of a list view.

Pascal

```
property VisibleRowCount: Integer;
```

C++

```
__property int VisibleRowCount;
```

Description

Read VisibleRowCount to determine the number of items from a single column that can appear in the visible area of a list view when the ViewStyle (see page 3829) is either vsList or vsReport. Only fully visible items are counted. If VisibleRowCount is

less than the total number of items in a column, use the TopItem (see page 3828) property to determine if the list view is scrolled.

See Also

TopItem (see page 3828)

ViewStyle (see page 3829)

27.1.19.5.37 TCustomListView.WorkAreas Property

Defines separate working areas that divide the list view's client area.

Pascal

```
property WorkAreas: TWorkAreas;
```

C++

```
__property TWorkAreas WorkAreas;
```

Description

Use WorkAreas to define separate work areas that divide the client area of the list view when the ViewStyle (see page 3829) property is vsIcon or vsSmallIcon. Work areas are rectangular regions that the list view uses to arrange its icons. Each icon falls into a single work area, which is specified by its WorkArea property. Within each work area, icons are arranged in rows (or columns), wrapping at the right (or bottom) edge of the work area. If a work area is larger than the client area of the list view, the list view adds a scroll bar.

If WorkAreas contains no TWorkArea (see page 5425) objects, the list view arranges icons based on its client area.

Note: It is possible to create overlapping work areas, but this is not recommended. Icons that fall in the intersection of two or more work areas become members of the work area with the lowest index.

Warning:Work areas are not supported if the application is running in virtual mode.

See Also

ViewStyle (see page 3829)

ShowWorkAreas (see page 3824)

ClientRect

TWorkArea (see page 5425)

WorkArea (see page 4257)

27.1.19.6 TCustomListView Events

The events of the TCustomListView class are listed here.

Protected Events

	Name	Description
	OnAdvancedCustomDraw (see page 3834)	Occurs at discrete points during the rendering of the list view.
	OnAdvancedCustomDrawItem (see page 3835)	Occurs at discrete stages during the painting of an item in an owner-draw list view.
	OnAdvancedCustomDrawSubItem (see page 3835)	Occurs at discrete stages during the painting of a subitem in an owner-drawn list view.
	OnChange (see page 3836)	Occurs immediately after an item in the list changes.
	OnChanging (see page 3837)	Occurs when an item in the list is about to change.

	OnColumnClick (see page 3837)	Occurs when the user clicks on a column header in a list view.
	OnColumnDragged (see page 3840)	Occurs when a column is dragged to a new position.
	OnColumnRightClick (see page 3841)	Occurs when the user right clicks on a column header in a list view.
	OnCompare (see page 3841)	Occurs to when two items need to be compared during a sort of the list.
	OnCreateItemClass (see page 3844)	Occurs when a list item object is about to be created.
	OnCustomDraw (see page 3844)	Occurs when an owner-draw list view must be rendered.
	OnCustomDrawItem (see page 3845)	Occurs when an item in an owner-draw list view must be rendered.
	OnCustomDrawSubItem (see page 3846)	Occurs when a subitem must be rendered in an owner-draw list view.
	OnData (see page 3846)	Occurs immediately before an item is displayed in the list view control.
	OnDataFind (see page 3847)	Occurs when the list view control receives a search request, such as when the FindData (see page 3789) and FindCaption (see page 3788) methods are called.
	OnDataHint (see page 3847)	Occurs when the contents of the list view change, such as when the user scrolls the list view.
	OnDataStateChange (see page 3848)	Occurs when a range of items change states.
	OnDeletion (see page 3848)	Occurs when an item in the list view is about to be deleted.
	OnDrawItem (see page 3848)	Occurs when an item should be drawn in the list view.
	OnEdited (see page 3849)	Occurs immediately after the user edits an item's Caption property.
	OnEditing (see page 3850)	Occurs when the user starts to edit a list item's Caption property.
	OnGetImageIndex (see page 3850)	Occurs when an item is about to be displayed in the list view control.
	OnGetSubItemImage (see page 3850)	Occurs when a subitem is about to be displayed in the column of a list view control.
	OnInfoTip (see page 3851)	Occurs when the user pauses the mouse over an item in the list view.
	OnInsert (see page 3851)	Occurs immediately after a new item is inserted into the list view.
	OnSelectItem (see page 3852)	Occurs when an item is selected in the list view control.

27.1.19.6.1 TCustomListView.OnAdvancedCustomDraw Event

Occurs at discrete points during the rendering of the list view.

Pascal

```
property OnAdvancedCustomDraw: TLVAdvancedCustomDrawEvent;
```

C++

```
__property TLVAdvancedCustomDrawEvent OnAdvancedCustomDraw;
```

Description

Write an OnAdvancedCustomDraw event handler to customize the painting of the list view at various stages before its items are drawn. Unlike the OnDrawItem (see page 3848) event, the list view receives this event even if the OwnerDraw (see page 3821) property is false.

Sender is the list view that is about to be painted.

ARect indicates the dimensions of the list view on its Canvas (see page 3804).

Stage indicates the current stage in the drawing process. Note that the cdPreErase and cdPostErase stages do not receive event notification. The background must be drawn when the item is rendered.

DefaultDraw is used only when Stage is cdPrePaint. It indicates whether the list view should continue with the default painting after the event handler exits. Set DefaultDraw to false to prevent the drawing of the list view after event handler exits. If DefaultDraw remains set to true, the list view continues with the default painting process.

Note: OnAdvancedCustomDraw occurs at various stages during the paint process. If you are only writing code to execute immediately prior to rendering the list view, it is more efficient to use the OnCustomDraw (see page 3844) event.

See Also

OnAdvancedCustomDrawItem (see page 3835)

OnAdvancedCustomDrawSubItem (see page 3835)

Canvas (see page 3804)

OwnerDraw (see page 3821)

OnCustomDraw (see page 3844)

27.1.19.6.2 TCustomListView.OnAdvancedCustomDrawItem Event

Occurs at discrete stages during the painting of an item in an owner-draw list view.

Pascal

```
property OnAdvancedCustomDrawItem: TLVAdvancedCustomDrawItemEvent;
```

C++

```
__property TLVAdvancedCustomDrawItemEvent OnAdvancedCustomDrawItem;
```

Description

Write an OnAdvancedCustomDrawItem event handler to customize the drawing of individual items on the list view's canvas (using the Canvas (see page 3804) property). OnAdvancedCustomDrawItem occurs at various stages in the rendering of a list item.

Unlike the OnDrawItem (see page 3848) event, the list view receives this event even if the OwnerDraw (see page 3821) property is false. The OnAdvancedCustomDrawItem event handler provides slightly different state information than the OnDrawItem (see page 3848) event and occurs after the default rendering as well as before.

Sender is the list view whose item is drawn.

Item is the item that is currently being painted.

State indicates the state of the item, so that the event handler can adjust the image to reflect whether the item is selected, disabled, hot, and so on.

Stage indicates the current stage in drawing the list item. Note that the cdPreErase and cdPostErase stages do not receive event notification. The background must be drawn when the item is rendered.

DefaultDraw is used only when Stage is cdPrePaint. Set DefaultDraw to false if you don't want the control to paint the item's text after the event handler exits. If DefaultDraw remains true, the list view adds the item's text to the image on the canvas.

Note: OnAdvancedCustomDrawItem occurs at various stages during the paint process. If you are only writing code to execute immediately prior to rendering the list item, it is more efficient to use the OnCustomDrawItem (see page 3845) event.

See Also

OnAdvancedCustomDraw (see page 3834)

OnAdvancedCustomDrawSubItem (see page 3835)

Canvas (see page 3804)

OwnerDraw (see page 3821)

OnCustomDrawItem (see page 3845)

27.1.19.6.3 TCustomListView.OnAdvancedCustomDrawSubItem Event

Occurs at discrete stages during the painting of a subitem in an owner-drawn list view.

Pascal

```
property OnAdvancedCustomDrawSubItem: TLVAdvancedCustomDrawSubItemEvent;
```

C++

```
__property TLVAdvancedCustomDrawSubItemEvent OnAdvancedCustomDrawSubItem;
```

Description

Write an OnCustomDrawSubItem (see page 3846) event handler to customize the drawing of subitems that appear in additional columns to the right of each item when ViewStyle (see page 3829) is vsReport. Use the list view's Canvas (see page 3804) property as a drawing surface. OnAdvancedCustomDrawItem (see page 3835) occurs at various stages in the rendering of a subitems. Unlike the OnDrawItem (see page 3848) event, the list view receives this event even if the OwnerDraw (see page 3821) property is false.

The Sender Parameter specifies the list view that owns the subitems.

The Item parameter is the current Item being drawn.

The SubItem parameter is the index of the subitem of that list item in its SubItems property.

The State property indicates various attributes that can affect the way the subitem is drawn.

The Stage property indicates the current stage in the painting process. Note that the cdPreErase and cdPostErase stages do not receive event notification. The background must be drawn when the item is rendered.

DefaultDraw is only used when Stage is cdPrepaint. Set DefaultDraw to false to prevent the list view from adding the subitem's text after the event handler exits.

Note: OnAdvancedCustomDrawSubItem occurs at various stages during the paint process. If you are only writing code to execute immediately prior to rendering the subitem, it is more efficient to use the OnCustomDrawSubItem (see page 3846) event.

See Also

OnCustomDrawItem (see page 3845)

SubItems (see page 4268)

ViewStyle (see page 3829)

Canvas (see page 3804)

OwnerDraw (see page 3821)

OnAdvancedCustomDrawSubItem

27.1.19.6.4 TCustomListView.OnChange Event

Occurs immediately after an item in the list changes.

Pascal

```
property OnChange: TLVChangeEvent;
```

C++

```
__property TLVChangeEvent OnChange;
```

Description

Write an OnChange event handler to respond to changes in the list items. OnChange allows a response once the list has been successfully changed.

The Item parameter is the list item that just changed. The Change (see page 3778) parameter indicates the type of change

that just occurred. Change (see page 3778) is ctText if the Caption property of the item changed. Change (see page 3778) is ctImage if the ImageIndex property of the item changed or the appropriate image list changed in the list view. Change (see page 3778) is ctState if the Cut, Focused, or Selected (see page 3822) property of the item changed.

See Also

LargeImages (see page 3817)

OnChanging (see page 3837)

SmallImages (see page 3825)

Cut (see page 4259)

Focused (see page 4260)

Caption (see page 4258)

Selected (see page 4267)

ImageIndex (see page 4261)

27.1.19.6.5 TCustomListView.OnChanging Event

Occurs when an item in the list is about to change.

Pascal

```
property OnChanging: TLVChangingEvent;
```

C++

```
__property TLVChangingEvent OnChanging;
```

Description

Write an OnChanging event handler to conditionally block changes to an item in the list. Set the AllowChange parameter to false to prevent the change from taking place. The Item parameter is the list item that is about to change. The Change (see page 3778) parameter indicates the type of change that is about to happen.

See Also

OnChange (see page 3836)

TListItem (see page 4250)

27.1.19.6.6 TCustomListView.OnColumnClick Event

Occurs when the user clicks on a column header in a list view.

Pascal

```
property OnColumnClick: TLVColumnClickEvent;
```

C++

```
__property TLVColumnClickEvent OnColumnClick;
```

Description

Write an OnColumnClick event handler to respond to clicks made in the column header when ViewStyle (see page 3829) is vsReport. The Column (see page 3806) parameter is the TListColumn (see page 4236) object from the Columns (see page 3808) property that describes the column that was clicked.

OnColumnClick only occurs when ViewStyle (see page 3829) is vsReport and ShowColumnHeaders (see page 3824) is

true.

C++ Examples:

```
/*
This example shows how to use the OnColumnClick and
OnCompare events of a list view to let users sort the
columns in a report-style list view by clicking on the
column headers. This requires a global variable to keep
track of the column that was clicked.  The OnColumnClick
event handler sets the global variable to indicate the
column to sort and calls AlphaSort:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {
    "C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {
    "Documentation for the C++ icon.",
    "Borland icon.", "Delphi icon."
    };
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  ListView1->ViewStyle = vsReport;
}

int ColumnToSort = 0;

/*
The OnColumnClick event handler sets the global variable to
indicate the column to sort and calls AlphaSort:
*/

void __fastcall TForm1::ListView1ColumnClick(TObject *Sender,
      TListColumn *Column)
{
  ColumnToSort = Column->Index;
  ((TCustomListView *)Sender)->AlphaSort();
}

/*
The OnCompare event handler causes the list view to sort on
the selected column:
*/

void __fastcall TForm1::ListView1Compare(TObject *Sender, TListItem *Item1,
      TListItem *Item2, int Data, int &Compare)
{
  if (ColumnToSort == 0)
```

```
    Compare = CompareText(Item1->Caption,Item2->Caption);
  else
  {
    int ix = ColumnToSort - 1;
    Compare =
      CompareText(Item1->SubItems->Strings[ix], Item2->SubItems->Strings[ix]);
  }
}
```

Delphi Examples:

```
{
This example shows how to use the OnColumnClick and
OnCompare events of a list view to let users sort the
columns in a report-style list view by clicking on the
column headers. This requires a global variable to keep
track of the column that was clicked.  The OnColumnClick
event handler sets the global variable to indicate the
column to sort and calls AlphaSort:
}
var ColumnToSort: Integer;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
const
  imagenames: Array[0..2] of string = ('C++ Image', 'Borland Image', 'Delphi Image');
begin
  Col2Array[0]:= 'Documentation for the C++ icon.';
  Col2Array[1]:= 'Borland icon.';
  Col2Array[2]:= 'Delphi icon.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := imagenames[I];
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
    NewColumn.Width := 200;
    ListView1.ViewStyle := vsReport;
  end;
end;

procedure TForm1.ListView1ColumnClick(Sender: TObject; Column: TListColumn);
begin
  ColumnToSort := Column.Index;
  (Sender as TCustomListView).AlphaSort;
end;

{
The OnCompare event handler causes the list view to sort on
the selected column:
}
procedure TForm1.ListView1Compare(Sender: TObject; Item1, Item2: TListItem; Data: Integer; var
```

```
  Compare: Integer);
  var
    ix: Integer;
  begin
    if ColumnToSort = 0 then
      Compare := CompareText(Item1.Caption,Item2.Caption)
    else begin
     ix := ColumnToSort - 1;
     Compare := CompareText(Item1.SubItems[ix],Item2.SubItems[ix]);
    end;
  end;
  {
  Note:   This OnCompare event handler uses the global
  CompareText function. An application may want to use
  AnsiCompareText, CompareStr, or AnsiCompareStr instead,
  depending on whether the comparison should be case-sensitive
  and whether the locale should be considered.
  }
```

See Also

ColumnClick (see page 3808)

Columns (see page 3808)

ShowColumnHeaders (see page 3824)

TListColumn (see page 4236)

ViewStyle (see page 3829)

OnColumnRightClick (see page 3841)

27.1.19.6.7 TCustomListView.OnColumnDragged Event

Occurs when a column is dragged to a new position.

Pascal

```
property OnColumnDragged: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnColumnDragged;
```

Description

Write an OnColumnDragged event handler to respond when the user drags a column to a new position. OnColumnDragged occurs after all columns are in their new positions but before the list view repaints to reflect the change.

OnColumnClick (see page 3837) only occurs when ViewStyle (see page 3829) is vsReport and ShowColumnHeaders (see page 3824) is true.

Tip: To uniquely identify columns so that you can recognize them after they are repositioned, use the list column's Tag property.

See Also

ViewStyle (see page 3829)

ShowColumnHeaders (see page 3824)

Tag (see page 4244)

Columns (see page 3808)

OnColumnClick (see page 3837)

27.1.19.6.8 TCustomListView.OnColumnRightClick Event

Occurs when the user right clicks on a column header in a list view.

Pascal

```
property OnColumnRightClick: TLVColumnRClickEvent;
```

C++

```
__property TLVColumnRClickEvent OnColumnRightClick;
```

Description

Write an OnColumnRightClick event handler to respond to right clicks made in the column header when ViewStyle (see page 3829) is vsReport.

The Sender parameter is the list view that the user clicked.

The Column (see page 3806) parameter is the TListColumn (see page 4236) object from the Columns (see page 3808) property for the column that was clicked.

The point parameter indicates where the user clicked the mouse, in client coordinates.

OnColumnRightClick only occurs when ViewStyle (see page 3829) is vsReport, and ShowColumnHeaders (see page 3824) is true, and ColumnRightClick is true.

See Also

Columns (see page 3808)

ShowColumnHeaders (see page 3824)

TListColumn (see page 4236)

ViewStyle (see page 3829)

OnContextPopup (see page 5854)

27.1.19.6.9 TCustomListView.OnCompare Event

Occurs to when two items need to be compared during a sort of the list.

Pascal

```
property OnCompare: TLVCompareEvent;
```

C++

```
__property TLVCompareEvent OnCompare;
```

Description

Write an OnCompare event handler to implement a sort order for the list. An OnCompare event handler is called when the SortType (see page 3827) property is stData or stBoth, when the AlphaSort (see page 3774) method is called, or when the CustomSort (see page 3784) method is called without a SortProc parameter.

The OnCompare event handler compares the list items passed as the Item1 and Item2 parameters. If Item1 is the same as Item2 in the sort order, set the Compare parameter to 0. If Item1 is less than Item2, set the Compare parameter to a value less than 0. If Item1 is greater than Item2, set the Compare parameter to a value greater than 0.

The Data parameter is 0 when the event handler is called to maintain the sort order of a list view with a SortType (see page 3827) of stData or stBoth. Similarly, when OnCompare occurs in response to the AlphaSort (see page 3774) method, the Data parameter is 0. When OnCompare occurs in response to the CustomSort (see page 3784) method, the Data parameter is the

value of the LParam parameter of CustomSort (see page 3784).

C++ Examples:

```
/*
This example shows how to use the OnColumnClick and
OnCompare events of a list view to let users sort the
columns in a report-style list view by clicking on the
column headers. This requires a global variable to keep
track of the column that was clicked.  The OnColumnClick
event handler sets the global variable to indicate the
column to sort and calls AlphaSort:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {
    "C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {
    "Documentation for the C++ icon.",
    "Borland icon.", "Delphi icon."
    };
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  ListView1->ViewStyle = vsReport;
}

int ColumnToSort = 0;

/*
The OnColumnClick event handler sets the global variable to
indicate the column to sort and calls AlphaSort:
*/

void __fastcall TForm1::ListView1ColumnClick(TObject *Sender,
      TListColumn *Column)
{
  ColumnToSort = Column->Index;
  ((TCustomListView *)Sender)->AlphaSort();
}

/*
The OnCompare event handler causes the list view to sort on
the selected column:
*/

void __fastcall TForm1::ListView1Compare(TObject *Sender, TListItem *Item1,
      TListItem *Item2, int Data, int &Compare)
{
  if (ColumnToSort == 0)
```

```
    Compare = CompareText(Item1->Caption,Item2->Caption);
  else
  {
    int ix = ColumnToSort - 1;
    Compare =
      CompareText(Item1->SubItems->Strings[ix], Item2->SubItems->Strings[ix]);
  }
}
```

Delphi Examples:

```
{
This example shows how to use the OnColumnClick and
OnCompare events of a list view to let users sort the
columns in a report-style list view by clicking on the
column headers. This requires a global variable to keep
track of the column that was clicked.  The OnColumnClick
event handler sets the global variable to indicate the
column to sort and calls AlphaSort:
}
var ColumnToSort: Integer;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
const
  imagenames: Array[0..2] of string = ('C++ Image', 'Borland Image', 'Delphi Image');
begin
  Col2Array[0]:= 'Documentation for the C++ icon.';
  Col2Array[1]:= 'Borland icon.';
  Col2Array[2]:= 'Delphi icon.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := imagenames[I];
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
    NewColumn.Width := 200;
    ListView1.ViewStyle := vsReport;
  end;
end;

procedure TForm1.ListView1ColumnClick(Sender: TObject; Column: TListColumn);
begin
  ColumnToSort := Column.Index;
  (Sender as TCustomListView).AlphaSort;
end;

{
The OnCompare event handler causes the list view to sort on
the selected column:
}
procedure TForm1.ListView1Compare(Sender: TObject; Item1, Item2: TListItem; Data: Integer; var
```

```
  Compare: Integer);
  var
    ix: Integer;
  begin
    if ColumnToSort = 0 then
      Compare := CompareText(Item1.Caption,Item2.Caption)
    else begin
     ix := ColumnToSort - 1;
     Compare := CompareText(Item1.SubItems[ix],Item2.SubItems[ix]);
    end;
  end;
  {
  Note:   This OnCompare event handler uses the global
  CompareText function. An application may want to use
  AnsiCompareText, CompareStr, or AnsiCompareStr instead,
  depending on whether the comparison should be case-sensitive
  and whether the locale should be considered.
  }
```

See Also

AlphaSort (see page 3774)

CustomSort (see page 3784)

SortType (see page 3827)

27.1.19.6.10 TCustomListView.OnCreateItemClass Event

Occurs when a list item object is about to be created.

Pascal

```
property OnCreateItemClass: TLVCreateItemClassEvent;
```

C++

```
__property TLVCreateItemClassEvent OnCreateItemClass;
```

Description

OnCreateItemClass occurs when a new item object is about to be created.

Sender is the list view object that needs to add a new panel.

Item class returns a class reference that is instantiated to create a new item object.

27.1.19.6.11 TCustomListView.OnCustomDraw Event

Occurs when an owner-draw list view must be rendered.

Pascal

```
property OnCustomDraw: TLVCustomDrawEvent;
```

C++

```
__property TLVCustomDrawEvent OnCustomDraw;
```

Description

Write an OnCustomDraw event handler to draw to the list view's Canvas (see page 3804) before its items are painted. Unlike the OnDrawItem (see page 3848) event, the list view receives this event even if the OwnerDraw (see page 3821) property is false.

Sender is the list view that is about to be painted.

ARect indicates the dimensions of the list view on its Canvas (see page 3804).

DefaultDraw indicates whether the list view should continue with the default painting after the event handler exits. Set DefaultDraw to false to prevent the drawing of the list view after event handler exits. If DefaultDraw remains set to true, the list view continues with the default painting process.

Note: OnCustomDraw occurs immediately prior to the rendering of the list view control. To augment the default drawing process at other stages (such as when the background is erased or after the list view is drawn), use the OnAdvancedCustomDraw (see page 3834) event instead.

See Also

OnCustomDrawItem (see page 3845)

OnCustomDrawSubItem (see page 3846)

Canvas (see page 3804)

OwnerDraw (see page 3821)

OnAdvancedCustomDraw (see page 3834)

27.1.19.6.12 TCustomListView.OnCustomDrawItem Event

Occurs when an item in an owner-draw list view must be rendered.

Pascal

```
property OnCustomDrawItem: TLVCustomDrawItemEvent;
```

C++

```
__property TLVCustomDrawItemEvent OnCustomDrawItem;
```

Description

Write code in an OnCustomDrawItem handler to draw individual items on the list view's canvas (using the Canvas (see page 3804) property) before an item is painted.

Unlike the OnDrawItem (see page 3848) event, the list view receives this event even if the OwnerDraw (see page 3821) property is false. The OnCustomDrawItem event handler provides slightly different state information than the OnDrawItem (see page 3848) event.

Sender is the list view whose item is to be drawn.

Item is the item that should be rendered.

State indicates the state of the item, so that the event handler can adjust the image to reflect whether the item is selected, disabled, hot, and so on.

Set DefaultDraw to false if you don't want the control to paint the item's text after the event handler exits. If DefaultDraw remains true, the list view adds the item's text to the image on the canvas.

Note: OnCustomDrawItem occurs immediately prior to the rendering of each list item. To augment the default drawing process at other stages (such as after the list item is drawn), use the OnAdvancedCustomDrawItem (see page 3835) event instead.

See Also

OnCustomDraw (see page 3844)

OnCustomDrawSubItem (see page 3846)

Canvas (see page 3804)

OwnerDraw (see page 3821)

OnDrawItem (see page 3848)

OnAdvancedCustomDrawItem (see page 3835)

27.1.19.6.13 TCustomListView.OnCustomDrawSubItem Event

Occurs when a subitem must be rendered in an owner-draw list view.

Pascal

```
property OnCustomDrawSubItem: TLVCustomDrawSubItemEvent;
```

C++

```
__property TLVCustomDrawSubItemEvent OnCustomDrawSubItem;
```

Description

Write code in an OnCustomDrawSubItem handler to draw to the subitems that appear in additional columns to the right of each item when ViewStyle (see page 3829) is vsReport. Use the list view's Canvas (see page 3804) property as a drawing surface. Unlike the OnDrawItem (see page 3848) event, the list view receives this event even if the OwnerDraw (see page 3821) property is false.

The Sender Parameter specifies the list view that owns the subitems. The Item parameter is the current Item being drawn. The SubItem parameter is the index of the subitem of that list item in its SubItems property. The State property indicates various attributes that can affect the way the subitem is drawn. Set DefaultDraw to False to prevent the list view from adding the subitem's text after the event handler exits.

Note: OnCustomDrawSubItem occurs immediately prior to the rendering of each subitem. To augment the default drawing process at other stages (such as after the subitem is drawn), use the OnAdvancedCustomDrawSubItem (see page 3835) event instead.

See Also

OnCustomDrawItem (see page 3845)

SubItems (see page 4268)

ViewStyle (see page 3829)

Canvas (see page 3804)

OwnerDraw (see page 3821)

OnAdvancedCustomDrawSubItem (see page 3835)

27.1.19.6.14 TCustomListView.OnData Event

Occurs immediately before an item is displayed in the list view control.

Pascal

```
property OnData: TLVOwnerDataEvent;
```

C++

```
__property TLVOwnerDataEvent OnData;
```

Description

Use OnData to customize an item before it is displayed in the list view control. Set the properties of the list item in the event handler so that they appear correctly when the item is drawn.

OnData occurs only if OwnerData (see page 3820) is true.

See Also

OwnerData (see page 3820)

OnDataFind (see page 3847)

OnDataHint (see page 3847)

OnDataStateChange (see page 3848)

27.1.19.6.15 TCustomListView.OnDataFind Event

Occurs when the list view control receives a search request, such as when the FindData (see page 3789) and FindCaption (see page 3788) methods are called.

Pascal

```
property OnDataFind: TLVOwnerDataFindEvent;
```

C++

```
__property TLVOwnerDataFindEvent OnDataFind;
```

Description

Use OnDataFind to locate an item and return custom data associated with it.

OnDataFind occurs only if OwnerData (see page 3820) is true.

See Also

OwnerData (see page 3820)

OnData (see page 3846)

OnDataHint (see page 3847)

OnDataStateChange (see page 3848)

27.1.19.6.16 TCustomListView.OnDataHint Event

Occurs when the contents of the list view change, such as when the user scrolls the list view.

Pascal

```
property OnDataHint: TLVOwnerDataHintEvent;
```

C++

```
__property TLVOwnerDataHintEvent OnDataHint;
```

Description

Use OnDataHint to update a range of items before the OnData (see page 3846) event occurs. This method of updating items is more efficient than individually updating items in the OnData (see page 3846) event handler. The range of items specified in the OnDataHint does not always match the range of items for which the OnData (see page 3846) event handler is called; therefore, you must be prepared to update individual items in the OnData (see page 3846) event handler as well.

OnDataHint occurs only if OwnerData (see page 3820) is true.

See Also

OwnerData (see page 3820)

OnDataFind (see page 3847)

OnData (see page 3846)

OnDataStateChange (see page 3848)

27.1.19.6.17 TCustomListView.OnDataStateChange Event

Occurs when a range of items change states.

Pascal

```
property OnDataStateChange: TLVOwnerDataStateChangeEvent;
```

C++

```
__property TLVOwnerDataStateChangeEvent OnDataStateChange;
```

Description

Use OnDataStateChange to respond when an item's state changes.

OnDataStateChange occurs only if OwnerData (see page 3820) is true.

See Also

OwnerData (see page 3820)

OnDataFind (see page 3847)

OnDataHint (see page 3847)

OnData (see page 3846)

27.1.19.6.18 TCustomListView.OnDeletion Event

Occurs when an item in the list view is about to be deleted.

Pascal

```
property OnDeletion: TLVDeletedEvent;
```

C++

```
__property TLVDeletedEvent OnDeletion;
```

Description

Write an OnDeletion event handler to respond when an item in the list is about to be deleted. The Item parameter is the TListItem (see page 4250) object from the Items (see page 3814) property that is about to be deleted.

See Also

Items (see page 3814)

OnInsert (see page 3851)

TListItem (see page 4250)

27.1.19.6.19 TCustomListView.OnDrawItem Event

Occurs when an item should be drawn in the list view.

Pascal

```
property OnDrawItem: TLVDrawItemEvent;
```

C++

```
__property TLVDrawItemEvent OnDrawItem;
```

Description

Write code in an OnDrawItem handler to manually draw items in the list view. This event occurs only if OwnerDraw (see page 3821) is true.

Note: List views receive several other custom draw events, including OnCustomDraw (see page 3844), OnCustomDrawItem (see page 3845), OnCustomDrawSubItem (see page 3846), OnAdvancedCustomDraw (see page 3834), OnAdvancedCustomDrawItem (see page 3835), and OnAdvancedCustomDrawSubItem (see page 3835). These other events, unlike OnDrawItem, occur regardless of the value of the OwnerDraw (see page 3821) property. They provide slightly different information about the state of the item to be drawn, and the OnAdvancedXXX events occur at more points during the paint process.

See Also

OnCustomDraw (see page 3844)

OnCustomDrawSubItem (see page 3846)

Canvas (see page 3804)

OwnerDraw (see page 3821)

OnCustomDrawItem (see page 3845)

OnAdvancedCustomDraw (see page 3834)

OnAdvancedCustomDrawItem (see page 3835)

OnAdvancedCustomDrawSubItem (see page 3835)

27.1.19.6.20 TCustomListView.OnEdited Event

Occurs immediately after the user edits an item's Caption property.

Pascal

```
property OnEdited: TLVEditedEvent;
```

C++

```
__property TLVEditedEvent OnEdited;
```

Description

Write an OnEdited event to take specific action when a user has finished editing the text of a list item. An OnEdited event handler can respond to the value the user typed, or change the value that is assigned to the list item's Caption property. For example, if an OnEdited event handler can check the validity of the string typed by the user, and only allow the Caption property to be changed if the new value is valid.

The Item parameter is the list item that was just edited. The S parameter is the new value the user typed for the item's Caption property. Change (see page 3778) the value of S to override the value the user typed. When OnEdited occurs, the item's Caption property has not yet been changed, so S can still be compared to the original value of Caption.

This event can occur only if ReadOnly (see page 3821) is set to false.

See Also

OnEditing (see page 3850)

ReadOnly (see page 3821)

Caption (see page 4258)

27.1.19.6.21 TCustomListView.OnEditing Event

Occurs when the user starts to edit a list item's Caption property.

Pascal

```
property OnEditing: TLVEditingEvent;
```

C++

```
__property TLVEditingEvent OnEditing;
```

Description

Write an OnEditing event handler to take specific action just before a user edits a list item's Caption property. To prevent the user from editing the item specified by the ListItem parameter, set the AllowEdit parameter to false.

This event can occur only if the ReadOnly (see page 3821) property is set to false.

See Also

OnEdited (see page 3849)

ReadOnly (see page 3821)

Caption (see page 4258)

27.1.19.6.22 TCustomListView.OnGetImageIndex Event

Occurs when an item is about to be displayed in the list view control.

Pascal

```
property OnGetImageIndex: TLVNotifyEvent;
```

C++

```
__property TLVNotifyEvent OnGetImageIndex;
```

Description

Use OnGetImageIndex to dynamically assign an ImageIndex to the item before it is displayed in the list view control.

See Also

OnGetSubItemImage (see page 3850)

ImageIndex (see page 4261)

27.1.19.6.23 TCustomListView.OnGetSubItemImage Event

Occurs when a subitem is about to be displayed in the column of a list view control.

Pascal

```
property OnGetSubItemImage: TLVSubItemImageEvent;
```

C++

```
__property TLVSubItemImageEvent OnGetSubItemImage;
```

Description

Use OnGetSubItemImage to dynamically assign an image to a subitem before it is displayed in the column of a list view control.

This event overrides the value specified by the list item's SubItemImages property.

The Sender parameter is the list view that is being painted.

The Item parameter is the item whose subitem is being painted.

The SubItem parameter identifies the subitem from the list item's SubItems property.

The ImageIndex parameter is an index into the list view's SmallImages (see page 3825) property. On entry to the event handler, ImageIndex is initialized to the value specified by the list item's SubItemImages property for the subitem. Use this event handler to change that value or to suppress the image by setting ImageIndex to –1.

Note: OnGetSubItemImage only occurs if the list view's ViewStyle (see page 3829) property is vsReport and the current list item's SubItems property contains subitems.

See Also

OnGetImageIndex (see page 3850)

SubItemImages (see page 4268)

SmallImages (see page 3825)

27.1.19.6.24 TCustomListView.OnInfoTip Event

Occurs when the user pauses the mouse over an item in the list view.

Pascal

```
property OnInfoTip: TLVInfoTipEvent;
```

C++

```
__property TLVInfoTipEvent OnInfoTip;
```

Description

Use OnInfoTip to provide help hints that are specific to individual items in the list view.

By default, when Help hints are enabled (that is, when ShowHint is true), the list view displays the hint specified by its Hint property. OnInfoTip allows the list view to override this value to specify a hint that is specific to the item under the mouse.

The Sender parameter is the list view under the mouse.

The Item parameter is the list item under the mouse.

The InfoTip parameter returns a string that appears in a help hint window. On entry to the event handler, InfoTip is the item's Caption. If the event handler assigns an empty string, no hint is displayed. Otherwise, a hint window appears that displays the value of InfoTip.

See Also

Hint (see page 5827)

Caption (see page 4258)

ShowHint (see page 5840)

27.1.19.6.25 TCustomListView.OnInsert Event

Occurs immediately after a new item is inserted into the list view.

Pascal

```
property OnInsert: TLVDeletedEvent;
```

C++

```
__property TLVDeletedEvent OnInsert;
```

Description

Write an OnInsert event handler to respond when an item has just been added to the list. The Item parameter is the TListItem (see page 4250) object that was added to the Items (see page 3814) property.

See Also

Items (see page 3814)

OnDeletion (see page 3848)

TListItem (see page 4250)

27.1.19.6.26 TCustomListView.OnSelectItem Event

Occurs when an item is selected in the list view control.

Pascal

```
property OnSelectItem: TLVSelectItemEvent;
```

C++

```
__property TLVSelectItemEvent OnSelectItem;
```

Description

Use OnSelectItem to perform actions when an item is selected.

C++ Examples:

```
/*
This code resizes the active control to twice as wide and
half as high:
*/
void __fastcall TForm1::ButtonTSClick(TObject *Sender)
{
  TRect MyRect;
  if (myActiveControl == NULL) exit;
  MyRect = myActiveControl->BoundsRect;
  MyRect.Right = MyRect.Left + (MyRect.Right - MyRect.Left) / 2;
  MyRect.Bottom = MyRect.Top + 2 * (MyRect.Bottom - MyRect.Top);
  myActiveControl->BoundsRect = MyRect;
}
void __fastcall TForm1::ButtonSFClick(TObject *Sender)
{
  TRect MyRect;
  if (myActiveControl == NULL) exit;
  MyRect = myActiveControl->BoundsRect;
  MyRect.Right = MyRect.Left + 2 * (MyRect.Right - MyRect.Left);
  MyRect.Bottom = MyRect.Top + (MyRect.Bottom - MyRect.Top) / 2;
  myActiveControl->BoundsRect = MyRect;
}
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TComponent *Temp;
  TListItem *ListItem;
  ListView1->ViewStyle = vsList;
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Components: ";
  for (int I = ComponentCount - 1; I >= 0; I--)
  {
    Temp = Components[I];
```

```
    ListItem = ListView1->Items->Add();
    ListItem->Caption = Temp->Name;
  }
}
void __fastcall TForm1::ListView1SelectItem(TObject *Sender, TListItem *Item,
      bool Selected)
{
  TComponent *comp = FindComponent(Item->Caption);
  if (dynamic_cast<TWinControl *>(comp) != NULL)
    myActiveControl = (TWinControl *)comp;
}
```

Delphi Examples:

```
{
This code resizes the active control to twice as wide and
half as high:
}
procedure TForm1.ButtonSFClick(Sender: TObject);
var
  MyRect: TRect;
begin
  if (myActiveControl = nil) then exit;
  MyRect := myActiveControl.BoundsRect;
  MyRect.Right := MyRect.Left + (MyRect.Right - MyRect.Left) * 2;
  MyRect.Bottom := MyRect.Top + (MyRect.Bottom - MyRect.Top) div 2;
  myActiveControl.BoundsRect := MyRect;
end;

procedure TForm1.ButtonTSClick(Sender: TObject);
var
  MyRect: TRect;
begin
  if (myActiveControl = nil) then exit;
  MyRect := myActiveControl.BoundsRect;
  MyRect.Right := MyRect.Left + (MyRect.Right - MyRect.Left) div 2;
  MyRect.Bottom := MyRect.Top + (MyRect.Bottom - MyRect.Top) * 2;
  myActiveControl.BoundsRect := MyRect;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
  Temp: TComponent;
  ListItem: TListItem;
begin
  ListView1.ViewStyle := vsList;
  ListItem := ListView1.Items.Add;
  ListItem.Caption := 'Components: ';
  for I := ComponentCount - 1 downto 0 do
  begin
    Temp := Components[I];
    begin
      ListItem := ListView1.Items.Add;
      ListItem.Caption := Temp.Name;
    end;
  end;
end;

procedure TForm1.ListView1SelectItem(Sender: TObject; Item: TListItem;
  Selected: Boolean);
var comp: TComponent;
begin
  comp := FindComponent(Item.Caption);
  if comp is TWinControl then
    myActiveControl := TWinControl(comp);
end;
```

See Also

Selected (see page 4267)

27.1.20 TCustomRichEdit Class

TCustomRichEdit is the base type for rich edit controls such as TRichEdit (see page 4685).

Class Hierarchy

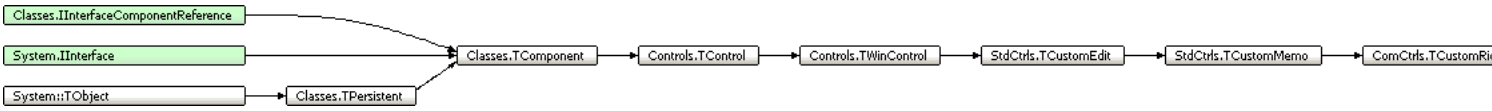

Pascal

```
TCustomRichEdit = class(TCustomMemo);
```

C++

```
class TCustomRichEdit : public TCustomMemo;
```

File

ComCtrls

Description

TCustomRichEdit introduces new properties and methods to expand the capabilities of TCustomMemo for rich text.

Do not create instances of TCustomRichEdit. Use TCustomRichEdit as a base class when declaring control objects that implement rich text capabilities in an edit control. Properties and methods of TCustomRichEdit provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TConversion (see page 3582)

TParaAttributes (see page 4606)

TRichEdit (see page 4685)

TTextAttributes (see page 4982)

27.1.20.1 TCustomRichEdit Members

The following tables list the members exposed by TCustomRichEdit.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnChange (see page 21275)	Occurs when the text for the edit control may have changed.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnProtectChange (see page 3888)	Occurs when a user attempts to modify text that is marked as protected.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnResizeRequest (see page 3889)	Occurs when the text has grown either smaller or larger than the control's window size.
	OnSaveClipboard (see page 3890)	Occurs when the window of the edit control is about to be destroyed and the edit control has delayed rendering of text copied to the clipboard..
	OnSelectionChange (see page 3890)	Occurs when the current selection changes.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21245)	Generates an OnChange (see page 21275) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 3868)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWindowHandle (see page 21248)	Creates a Windows control to represent the control.
V	CreateWindowHandle (see page 21376)	Creates a window based on the window-creation parameter record.
V	CreateWnd (see page 3868)	Creates a Windows control corresponding to the edit control.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.

	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 3868)	Destroys the rich edit control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoSetMaxLength (see page 3869)	Sets the MaxLength property.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetCaretPos (see page 3871)	Returns the cursor position.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.

	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelLength (see page 3871)	Returns the SelLength property.
	GetSelStart (see page 3872)	Returns the SelStart property.
	GetSelText (see page 3872)	Returns the SelText property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyPress (see page 21377)	Ensures that return characters are not entered into the text when WantReturns (see page 21384) is false.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	Loaded (see page 21377)	Initializes the memo control when it is first loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.

	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	RequestSize (see page 3874)	Resize request event dispatcher.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectionChange (see page 3875)	Selection change event dispatcher.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAlignment (see page 21377)	Sets the Alignment (see page 21379) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetAutoSize (see page 21257)	Sets the AutoSize (see page 21260) property.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetCaretPos (see page 3875)	Sets the cursor position in the text.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetLines (see page 21378)	Sets the Lines (see page 21380) property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetScrollBars (see page 21378)	Sets the ScrollBars (see page 21381) property.
	SetSelLength (see page 3876)	Sets the SelLength property.
	SetSelStart (see page 3876)	Returns the SelStart property.
	SetWordWrap (see page 21379)	Sets the WordWrap (see page 21384) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.

	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	Alignment (see page 21379)	Determines how the text is aligned within the memo control.
	AutoSelect (see page 21260)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AutoSize (see page 21260)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 21261)	Determines whether the edit control has a single line border around the client area.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	CharCase (see page 21262)	Determines the case of the text within the edit control.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	HideScrollBars (see page 3879)	Determines whether the scroll bars disappear when not needed.
	HideSelection (see page 3881)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	Lines (see page 3882)	Contains the individual lines of text in the rich text edit control.

	MaxLength (see page 21263)	Specifies the maximum number of characters the user can enter into the edit control.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	OEMConvert (see page 21264)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 21264)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PasswordChar (see page 21265)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PlainText (see page 3886)	Controls (see page 5711) whether the rich edit control treats the text as plain text or rich text when streaming to or from a file.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ScrollBars (see page 21381)	Determines whether the memo control has scroll bars.
	Text (see page 5842)	Contains a text string associated with the control.
	WantReturns (see page 21384)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 21384)	Determines whether the user can insert tab characters into the text.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.
	WordWrap (see page 21384)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 3867)	Deletes all text from the rich edit control.
	ClearSelection (see page 21245)	Removes the selected text from the edit control.
	ClearUndo (see page 21246)	Clears the undo buffer so that no changes to the text can be backed out.

	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopyToClipboard (see page 21246)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
V	Create (see page 3866)	Creates an instance of TCustomRichEdit (see page 3854).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CutToClipboard (see page 21249)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 21250)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 3866)	Destroys an instance of TCustomRichEdit (see page 3854).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindText (see page 3869)	Searches a given range in the text for a target string.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
V	GetControlsAlignment (see page 21376)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.

	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetSelTextBuf (see page 3872)	Copies the selected text into a buffer and returns the number of characters copied.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	PasteFromClipboard (see page 21254)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	Print (see page 3873)	Formats and prints the contents of the rich edit control.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RegisterConversionFormat (see page 3874)	Registers an association between a file extension and a TConversion (see page 3582) object that can convert between the native file format and rich text format (RTF).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.

	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectAll (see page 21255)	Selects all text in the edit control.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSelTextBuf (see page 21258)	Replaces the selected text with a null-terminated string.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Undo (see page 21259)	Backs out all changes in the undo buffer.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	CanUndo (see page 21262)	Indicates whether the edit control contains changes that can be backed out.
	CaretPos (see page 21380)	Indicates the position of the caret in the client area of the memo.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DefAttributes (see page 3877)	Describes the rich text characteristics of the default font for the rich edit control

	DefaultConverter (see page 3878)	Specifies the class of the TConversion (see page 3582) object that is used to convert between native file format and the internal format of a rich text control when working with files that do not have a registered extension.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Modified (see page 21263)	Indicates whether the user edited the text of the edit control.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	PageRect (see page 3884)	Specifies the dimensions, in pixels, of the logical page size used when printing the contents of a rich text edit control.
R	Paragraph (see page 3884)	Specifies the formatting information for the current paragraphs.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ReadOnly (see page 21265)	Determines whether the user can change the text of the edit control.
	SelAttributes (see page 3887)	Describes the rich text characteristics of the selected text in the rich edit control.
	SelLength (see page 21265)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 21269)	Specifies the position of the first selected character in the text.
	SelText (see page 21272)	Specifies the selected portion of the edit control's text.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	Text (see page 21273)	Contains a text string associated with the control.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.

	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	TabStop (see page 21273)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.20.2 TCustomRichEdit.Create Constructor

Creates an instance of TCustomRichEdit (see page 3854).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomRichEdit(TComponent * AOwner);
```

Description

Call Create to create an instance of TCustomRichEdit (see page 3854) at runtime. Rich edit controls added to forms at design time are created automatically.

AOwner is a component, typically the form, that becomes the value of the Owner property.

See Also

Owner (see page 2807)

ParentWindow (see page 6210)

27.1.20.3 TCustomRichEdit.Destroy Destructor

Destroys an instance of TCustomRichEdit (see page 3854).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomRichEdit();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TCustomRichEdit (see page 3854) reference is not nil and only then calls Destroy.

Destroy overrides the inherited method to destroy all the helper objects that were allocated to manage the rich text, before calling

the inherited destructor.

See Also

Free (see page 22300)

27.1.20.4 TCustomRichEdit Methods

The methods of the TCustomRichEdit class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 3868)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateWnd (see page 3868)	Creates a Windows control corresponding to the edit control.
	DestroyWnd (see page 3868)	Destroys the rich edit control's window.
	DoSetMaxLength (see page 3869)	Sets the MaxLength property.
	GetCaretPos (see page 3871)	Returns the cursor position.
	GetSelLength (see page 3871)	Returns the SelLength property.
	GetSelStart (see page 3872)	Returns the SelStart property.
	GetSelText (see page 3872)	Returns the SelText property.
	RequestSize (see page 3874)	Resize request event dispatcher.
	SelectionChange (see page 3875)	Selection change event dispatcher.
	SetCaretPos (see page 3875)	Sets the cursor position in the text.
	SetSelLength (see page 3876)	Sets the SelLength property.
	SetSelStart (see page 3876)	Returns the SelStart property.

Public Methods

	Name	Description
	Clear (see page 3867)	Deletes all text from the rich edit control.
	FindText (see page 3869)	Searches a given range in the text for a target string.
	GetSelTextBuf (see page 3872)	Copies the selected text into a buffer and returns the number of characters copied.
	Print (see page 3873)	Formats and prints the contents of the rich edit control.
	RegisterConversionFormat (see page 3874)	Registers an association between a file extension and a TConversion (see page 3582) object that can convert between the native file format and rich text format (RTF).

27.1.20.4.1 TCustomRichEdit.Clear Method

Deletes all text from the rich edit control.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Use Clear to replace the Text property with an empty string. Clear removes all text, not just the selected text. To remove just the selected text, use ClearSelection.

See Also

ClearSelection (see page 21245)

Text (see page 5842)

27.1.20.4.2 TCustomRichEdit.CreateParams Method

Initializes a window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 3868) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. TCustomRichEdit (see page 3854) overrides the inherited method to specify a Windows rich edit control.

See Also

CreateWindowHandle (see page 21376)

CreateWnd (see page 3868)

27.1.20.4.3 TCustomRichEdit.CreateWnd Method

Creates a Windows control corresponding to the edit control.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

The CreateWnd method is called when the rich edit control is created, and whenever a change to the rich edit control requires the window to be recreated. CreateWnd first calls the inherited CreateWnd, and then sends the newly created window messages to request notification of Windows events, and, if the window is being recreated, reloads the rich text from a memory stream.

See Also

CreateParams (see page 3868)

CreateWindowHandle (see page 21376)

DestroyWnd (see page 3868)

CreateWnd (see page 6137)

27.1.20.4.4 TCustomRichEdit.DestroyWnd Method

Destroys the rich edit control's window.

Pascal

```
procedure DestroyWnd; override;
```

C++

```
virtual __fastcall DestroyWnd();
```

Description

DestroyWnd is called automatically when the rich edit control is destroyed or when its window must be temporarily destroyed and

recreated to implement a property change. Before the window is destroyed, DestroyWnd saves a copy of the control's text in a memory stream, frees any device contexts, and finally calls DestroyWindowHandle.

See Also

CreateWnd (see page 3868)

DestroyWindowHandle (see page 6140)

27.1.20.4.5 TCustomRichEdit.DoSetMaxLength Method

Sets the MaxLength property.

Pascal

```
procedure DoSetMaxLength(Value: Integer); override;
```

C++

```
virtual __fastcall DoSetMaxLength(int Value);
```

Description

DoSetMaxLength overrides the inherited method to implement the MaxLength property for a rich edit window. DoSetMaxLength sends an EM_EXLIMITTEXT message to the window of the rich edit control.

See Also

MaxLength (see page 21263)

27.1.20.4.6 TCustomRichEdit.FindText Method

Searches a given range in the text for a target string.

Pascal

```
function FindText(const SearchStr: string; StartPos: Integer; Length: Integer; Options:
TSearchTypes): Integer;
```

C++

```
__fastcall int FindText(const AnsiString SearchStr, int StartPos, int Length, TSearchTypes
Options);
```

Description

Use FindText to search a range of text in a rich edit control for the string specified by the SearchStr parameter. Only the text in the range starting at the position StartPos and continuing through the next Length positions will be searched. Use Options to specify whether the search should match whole words only and whether the search should be case sensitive. FindText returns the position of the search string, where 0 is the position of the first character in the rich edit control (as opposed to the first position in the search range). If the search string does not appear in the indicated range, FindText returns -1.

Use FindText rather than programmatically searching through the text of the control, to keep from being thrown off by the encoding of rich text characteristics.

C++ Examples:

```
/*
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
*/
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FindDialog1->Position = Point(RichEdit1->Left + RichEdit1->Width, RichEdit1->Top);
  FindDialog1->Execute();
}

void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  int FoundAt, StartPos, ToEnd;
  TSearchTypes mySearchTypes;
  // begin the search after the current selection
  // if there is one
  // otherwise, begin at the start of the text
  if (FindDialog1->Options.Contains(frMatchCase))
    mySearchTypes << stMatchCase;
  if (FindDialog1->Options.Contains(frWholeWord))
    mySearchTypes << stWholeWord;
  if (RichEdit1->SelLength)
    StartPos = RichEdit1->SelStart + RichEdit1->SelLength;
  else
    StartPos = 0;
  // ToEnd is the length from StartPos
  // to the end of the text in the rich edit control
  ToEnd = RichEdit1->Text.Length() - StartPos;
  FoundAt = RichEdit1->FindText(FindDialog1->FindText, StartPos, ToEnd, mySearchTypes);
  if (FoundAt != -1)
  {
    RichEdit1->SetFocus();
    RichEdit1->SelStart = FoundAt;
    RichEdit1->SelLength = FindDialog1->FindText.Length();
  }
  else Beep();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("../Overview.RTF");  // starts in Debug
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FindDialog1.Position :=
    Point(RichEdit1.Left + RichEdit1.Width, RichEdit1.Top);
  FindDialog1.Execute;
end;

procedure TForm1.FindDialog1Find(Sender: TObject);
var
  FoundAt: LongInt;
  StartPos, ToEnd: Integer;
  mySearchTypes : TSearchTypes;
  myFindOptions : TFindOptions;
begin
  with RichEdit1 do
  begin
```

```
      if frMatchCase in FindDialog1.Options then
         mySearchTypes := mySearchTypes + [stMatchCase];
      if frWholeWord in FindDialog1.Options then
         mySearchTypes := mySearchTypes + [stWholeWord];
      { begin the search after the current selection if there is one }
      { otherwise, begin at the start of the text }
      if SelLength <> 0 then
        StartPos := SelStart + SelLength
      else
        StartPos := 0;
      { ToEnd is the length from StartPos to the end of the
        text in the rich edit control }
      ToEnd := Length(Text) - StartPos;
      FoundAt :=
        FindText(FindDialog1.FindText, StartPos, ToEnd, mySearchTypes);
      if FoundAt <> -1 then
      begin
        SetFocus;
        SelStart := FoundAt;
        SelLength := Length(FindDialog1.FindText);
      end
      else Beep;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const Path = 'OverView.RTF';
  begin
    RichEdit1.PlainText := False;
    RichEdit1.Lines.LoadFromFile(Path);
    RichEdit1.ScrollBars := ssVertical;
  end;
```

See Also

Text (see page 5842)

27.1.20.4.7 TCustomRichEdit.GetCaretPos Method

Returns the cursor position.

Pascal

```
function GetCaretPos: TPoint; override;
```

C++

```
virtual __fastcall TPoint GetCaretPos();
```

Description

GetCaretPos is the read implementation of the CaretPos property.

See Also

CaretPos (see page 21380)

27.1.20.4.8 TCustomRichEdit.GetSelLength Method

Returns the SelLength property.

Pascal

```
function GetSelLength: Integer; override;
```

C++

```
virtual __fastcall int GetSelLength();
```

Description

GetSelLength overrides the inherited method to implement the SelLength property for a rich edit window. GetSelLength sends an EM_EXGETSEL message to the window of the edit control and returns the difference between the starting position and the ending position.

See Also

SelLength

SetSelLength (see page 3876)

27.1.20.4.9 TCustomRichEdit.GetSelStart Method

Returns the SelStart property.

Pascal

```
function GetSelStart: Integer; override;
```

C++

```
virtual __fastcall int GetSelStart();
```

Description

GetSelStart overrides the inherited method to implement the SelStart property for a rich edit window. GetSelLength (see page 3871) sends an EM_EXGETSEL message to the window of the edit control and returns the starting position.

See Also

SelStart

SetSelStart (see page 3876)

27.1.20.4.10 TCustomRichEdit.GetSelText Method

Returns the SelText property.

Pascal

```
function GetSelText: string; override;
```

C++

```
virtual __fastcall AnsiString GetSelText();
```

Description

GetSelText overrides the inherited method to implement the SelText property for a rich edit window. GetSelText sends an EM_GETSELTEXT message to the window of the rich edit control.

See Also

SelText

27.1.20.4.11 TCustomRichEdit.GetSelTextBuf Method

Copies the selected text into a buffer and returns the number of characters copied.

Pascal

```
function GetSelTextBuf(Buffer: PChar; BufSize: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetSelTextBuf(const char * Buffer, int BufSize);
```

Description

Use GetSelTextBuf to copy the selected text into a character buffer. If there is no selection, the buffer receives an empty string. If the selection contains more than (BufSize - 1) characters, only the first (BufSize - 1) characters are copied. GetSelTextBuf returns the number of characters that were actually copied into the buffer.

See Also

CopyToClipboard (see page 21246)

GetTextBuf (see page 5766)

SelLength

SelText

SetSelTextBuf (see page 21258)

27.1.20.4.12 TCustomRichEdit.Print Method

Formats and prints the contents of the rich edit control.

Pascal

```
procedure Print(const Caption: string); virtual;
```

C++

```
virtual __fastcall Print(const AnsiString Caption);
```

Description

Use Print to print the contents of a rich edit control. The Caption parameter specifies the title that appears in the print manager and on network title pages.

C++ Examples:

```
/*
Place a button and a richedit on the form.  No print dialog
appears.  The contents of the richedit is printed to the
currently configured printer.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
// you may need to change this path to suit your environment
  char const *Path = "..\\OVERVIEW.RTF";
  RichEdit1->Lines->LoadFromFile(Path);
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
// The parameter string shows in the print queue under "Document   // name".
  RichEdit1->Print("My Document Name");
}
```

Delphi Examples:

```
{
Place a button and a richedit on the form.  No print dialog
appears.  The contents of the richedit is printed to the
```

```
currently configured printer.
}
procedure TForm1.FormCreate(Sender: TObject);
const
  // The path may need to be changed to accommodate
  // locations of files on your system.
  Path = 'OVERVIEW.RTF';
begin
   RichEdit1.Lines.LoadFromFile(Path);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  // The parameter string shows in the print queue
  // under "Document/name".
  RichEdit1.Print('My Document Name');
end;
```

See Also

PageRect (see page 3884)

27.1.20.4.13 TCustomRichEdit.RegisterConversionFormat Method

Registers an association between a file extension and a TConversion (see page 3582) object that can convert between the native file format and rich text format (RTF).

Pascal

```
class procedure RegisterConversionFormat(const AExtension: string; AConversionClass:
TConversionClass);
```

C++

```
__fastcall RegisterConversionFormat(const AnsiString AExtension, TConversionClass
AConversionClass);
```

Description

Rich edit controls encode text attributes in RTF format. To read from or write to files that encode text attributes in some other format, a converter is required. The converter is a descendant of TConversion (see page 3582) that translates between the native file format and RTF when the text is streamed in or out. RegisterConversionFormat sets up an association between a file extension and a converter, so that all attempts to stream between a file with that extension and a rich edit control automatically invoke the appropriate converter.

Note: Use the Lines (see page 3882) property to stream the text to or from a file.

See Also

DefaultConverter (see page 3878)

Lines (see page 3882)

PlainText (see page 3886)

TConversion (see page 3582)

_classid

27.1.20.4.14 TCustomRichEdit.RequestSize Method

Resize request event dispatcher.

Pascal

```
procedure RequestSize(const Rect: TRect); virtual;
```

C++

```
virtual __fastcall RequestSize(const TRect Rect);
```

Description

RequestSize merely calls the OnResizeRequest (see page 3889) event handler. RequestSize is exposed as a protected method so that derived classes can change the response to Windows messages about an impending window resize. When overriding RequestSize, include a call to the OnResizeRequest (see page 3889) event handler.

See Also

BoundsRect (see page 5803)

ClientRect

OnResizeRequest (see page 3889)

Text (see page 5842)

27.1.20.4.15 TCustomRichEdit.SelectionChange Method

Selection change event dispatcher.

Pascal

```
procedure SelectionChange; dynamic;
```

C++

```
__fastcall SelectionChange();
```

Description

SelectionChange merely calls the OnSelectionChange (see page 3890) event handler. SelectionChange is exposed as a protected method so that derived classes can change the response to Windows messages about selection changes. When overriding SelectionChange, include a call to the OnSelectionChange (see page 3890) event handler.

See Also

AutoSelect (see page 21260)

ClearSelection (see page 21245)

OnSelectionChange (see page 3890)

SelectAll (see page 21255)

SelAttributes (see page 3887)

SelLength

SelStart

SelText

SetSelTextBuf (see page 21258)

27.1.20.4.16 TCustomRichEdit.SetCaretPos Method

Sets the cursor position in the text.

Pascal

```
procedure SetCaretPos(const Value: TPoint); override;
```

C++

```
virtual __fastcall SetCaretPos(const TPoint Value);
```

Description

SetCaretPos is the write implementation of the CaretPos property.

Value is the new position of the text cursor.

See Also

CaretPos (see page 21380)

27.1.20.4.17 TCustomRichEdit.SetSelLength Method

Sets the SelLength property.

Pascal

```
procedure SetSelLength(Value: Integer); override;
```

C++

```
virtual __fastcall SetSelLength(int Value);
```

Description

SetSelLength overrides the inherited method to implement the SelLength property for a rich edit window. SetSelLength sends an EM_EXSETSEL message to the window of the edit control, specifying a selection that begins at SelStart and ends at SelStart + Value.

See Also

GetSelLength (see page 3871)

SelLength

SelStart

27.1.20.4.18 TCustomRichEdit.SetSelStart Method

Returns the SelStart property.

Pascal

```
procedure SetSelStart(Value: Integer); override;
```

C++

```
virtual __fastcall SetSelStart(int Value);
```

Description

SetSelStart overrides the inherited method to implement the SelStart property for a rich edit window. SetSelLength (see page 3876) sends an EM_EXSETSEL message to the window of the edit control to position the cursor on the position indicated by Value.

See Also

GetSelStart (see page 3872)

SelStart

27.1.20.5 TCustomRichEdit Properties

The properties of the TCustomRichEdit class are listed here.

Protected Properties

	Name	Description
	HideScrollBars (see page 3879)	Determines whether the scroll bars disappear when not needed.
	HideSelection (see page 3881)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	Lines (see page 3882)	Contains the individual lines of text in the rich text edit control.
	PlainText (see page 3886)	Controls (see page 5711) whether the rich edit control treats the text as plain text or rich text when streaming to or from a file.

Public Properties

	Name	Description
	DefAttributes (see page 3877)	Describes the rich text characteristics of the default font for the rich edit control
	DefaultConverter (see page 3878)	Specifies the class of the TConversion (see page 3582) object that is used to convert between native file format and the internal format of a rich text control when working with files that do not have a registered extension.
	PageRect (see page 3884)	Specifies the dimensions, in pixels, of the logical page size used when printing the contents of a rich text edit control.
R	Paragraph (see page 3884)	Specifies the formatting information for the current paragraphs.
	SelAttributes (see page 3887)	Describes the rich text characteristics of the selected text in the rich edit control.

27.1.20.5.1 TCustomRichEdit.DefAttributes Property

Describes the rich text characteristics of the default font for the rich edit control

Pascal

```
property DefAttributes: TTextAttributes;
```

C++

```
__property TTextAttributes DefAttributes;
```

Description

Use DefAttributes to discover or set the default font characteristics that the rich edit control uses for newly inserted text. These are the characteristics of the text before any special attributes have been applied. Once any special attributes are applied to a portion of text, no text from that point on is considered to have the default attributes, even if the attributes match the DefAttributes.

For example, if a rich edit control has had no attributes applied, the entire text has the default attributes. Selecting a portion of text in the middle and applying a specific attribute (such as a font change) results in three sections: a first section with the default attributes, a middle section with the applied attribute, and a final section with a non-default set of attributes that match the default attributes. Changing the DefAttributes property affects only the first section.

When inserting new text, the font characteristics of the new text will match the font characteristics at the cursor position, or, if the typing replaces a selection, the font characteristics of the selected text. New text will only have the default attributes when typed into the section of text that has the default attributes.

Note: DefAttributes is available only at runtime.

C++ Examples:

```
/*
```

```
This example requires a TRichEdit and two TButtons. Leave
the default text, usually "RichEdit1", in the lines
property. The DefAttributes will apply to this text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->SelAttributes->Color = clRed;
  RichEdit1->SelAttributes->Height += 5;
  RichEdit1->Lines->Add("This line of text will be red. ");
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  RichEdit1->DefAttributes->Color = clBlue;
  RichEdit1->DefAttributes->Style =
    RichEdit1->DefAttributes->Style << fsBold << fsItalic;
}
```

Delphi Examples:

```
{
This example requires a TRichEdit and two TButtons. Leave
the default text, usually "RichEdit1", in the lines
property. The DefAttributes will apply to this text.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  with RichEdit1.SelAttributes do
  begin
    Color := clRed;
    Height := Height + 5;
  end;
  RichEdit1.Lines.Add('This line of text will be red.');
end;


procedure TForm1.Button2Click(Sender: TObject);
begin
  RichEdit1.DefAttributes.Color := clBlue;
  RichEdit1.DefAttributes.Style := [fsBold, fsItalic];
end;
```

See Also

PlainText (see page 3886)

SelAttributes (see page 3887)

27.1.20.5.2 TCustomRichEdit.DefaultConverter Property

Specifies the class of the TConversion (see page 3582) object that is used to convert between native file format and the internal format of a rich text control when working with files that do not have a registered extension.

Pascal

```
property DefaultConverter: TConversionClass;
```

C++

```
__property TConversionClass DefaultConverter;
```

Description

Use DefaultConverter to specify the type of conversion to use when streaming text to or from files that do not have a registered extension. If DefaultConverter is not set, TCustomRichEdit (see page 3854) uses TConversion (see page 3582), which performs no actual conversion. TCustomRichEdit (see page 3854) encodes text attributes in rich-text format (RTF). Thus,

unrecognized file extensions lead to no conversion, which works for files that contain plain text or any subset of RTF format.

To support streaming to or from files that encode rich text characteristics in some other format, register a conversion format with the file extension used by those files. Use DefaultConverter only to change the conversion that is used for unrecognized file extensions.

Note: Use the Lines (see page 3882) property to stream the text to or from a file.

See Also

Lines (see page 3882)

PlainText (see page 3886)

RegisterConversionFormat (see page 3874)

TConversion (see page 3582)

27.1.20.5.3 TCustomRichEdit.HideScrollBars Property

Determines whether the scroll bars disappear when not needed.

Pascal

```
property HideScrollBars: Boolean;
```

C++

```
__property Boolean HideScrollBars;
```

Description

Use HideScrollBars to determine whether the scroll bars disappear from the edit control when the text appears completely within the edit window. Set HideScrollBars to false to prevent the scroll bars from flashing on and off when the contents of the rich edit control change.

HideScrollBars does nothing when the ScrollBars property is set to ssNone (The default for TRichEdit (see page 4685)).

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
```

```
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
    ssHorizontal,
    ssNone,
    ssVertical);
begin
  RichEdit1.ScrollBars :=
    ScrollBarA[RadioGroup1.ItemIndex];
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with RadioGroup1, Items do
  begin
    Caption := 'Scroll Bars';
```

```
      Clear;
      Add('ssBoth');
      Add('ssHorizontal');
      Add('ssNone');
      Add('ssVertical');
    end;
    with CheckBox1 do
    begin
      Caption := 'HideScrollBars';
      Checked := True;
    end;
    Button1.Caption := 'Loaded';
    RichEdit1.Text := 'Just a little text.';
    RichEdit1.WordWrap := False;
  end;
```

See Also

ScrollBars (see page 21381)

27.1.20.5.4 TCustomRichEdit.HideSelection Property

Determines whether the visual indication of the selected text remains when focus shifts to another control.

Pascal

```
property HideSelection: Boolean;
```

C++

```
__property Boolean HideSelection;
```

Description

Set HideSelection to false when it is important to provide visual feedback of the selected portion of the text even when the edit control does not have focus. Set HideSelection to true to make the visual indication of selection appear only when the edit control has focus. HideSelection does not affect the actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms containing many edit controls look busy.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
HideSelection, Height example
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
        HideSelection := False;
    end;
end;
```

See Also

SelLength

SelStart

SelText

SetSelTextBuf (see page 21258)

27.1.20.5.5 TCustomRichEdit.Lines Property

Contains the individual lines of text in the rich text edit control.

Pascal

```
property Lines: TStrings;
```

C++

```
__property TStrings Lines;
```

Description

Use Lines to manipulate the text in the rich text edit control on a line by line basis. Lines is a TStrings object, so TStrings methods may be used for Lines to perform manipulations such as counting the lines of text, adding lines, deleting lines, or replacing the text in lines.

To work with the text as one chunk, use the Text property. To manipulate individual lines of text, the Lines property works better.

Warning: If you paste formatted text into the Lines property at design time, the formatting is lost when the form is saved (it does not appear at runtime). To set the value of a rich edit control at design time, you must save the formatted text in some other manner and use the LoadFromStream or LoadFromFile method of the Lines property to load the value in the OnShow method of the parent form.

C++ Examples:

```
/*
This example requires only a form.  A Rich Edit control is
created on the form and aligned to take the entire client
area of the form.  A series of lines are written to the Rich
Edit control's Lines property, and the Paragraph property
manipulated such that the first items displayed are bulleted,
a second group of lines are unbulleted and centered, and the
last group is left justified and indented.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TRichEdit *pRich = new TRichEdit(this);
  pRich->Parent = this;
  pRich->Align = alClient;
  pRich->Lines->Clear();
  // set numbering style
  pRich->Paragraph->Numbering = nsBullet;
  pRich->Lines->Add("Introduction");
  pRich->Lines->Add("New members to our team");
  pRich->Lines->Add("New Budget discussion");
  pRich->Lines->Add("Facilities");
  pRich->Lines->Add("Q & A");
  pRich->Paragraph->Numbering = nsNone;
  pRich->Paragraph->Alignment = taCenter;
  pRich->Lines->Add("");
  pRich->Lines->Add("Suggested Topics: ");
  pRich->Lines->Add("");
  pRich->Paragraph->Alignment = taLeftJustify;
  pRich->Paragraph->FirstIndent = 10;
  pRich->Lines->Add("");
  pRich->Lines->Add("Parking lot repair");
  pRich->Lines->Add("Cost overruns");
}
```

Delphi Examples:

```
{
This example requires only a form.  A Rich Edit control is
created on the form and aligned to take the entire client
area of the form.  A series of lines are written to the Rich
Edit control's Lines property, and the Paragraph property
manipulated such that the first items displayed are bulleted,
a second group of lines are unbulleted and centered, and the
last group is left justified and indented.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with TRichEdit.Create(Self) do
  begin
    Parent := Self;
    Align := alClient;
    Lines.Clear;
    // set numbering style
    Paragraph.Numbering := nsBullet;
    Lines.Add('Introduction');
    Lines.Add('New members to our team');
    Lines.Add('New Budget discussion');
    Lines.Add('Facilities');
    Lines.Add('Q & A');
    Paragraph.Numbering := nsNone;
    Paragraph.Alignment := taCenter;
    Lines.Add('');
    Lines.Add('Suggested Topics:');
    Lines.Add('');
    Paragraph.Alignment := taLeftJustify;
```

```
      Paragraph.FirstIndent := 10;
      Lines.Add('');
      Lines.Add('Parking lot repair');
      Lines.Add('Cost overruns');
    end;
  end;
```

See Also

SelText

Text (see page 5842)

WordWrap (see page 21384)

LoadFromFile (see page 3061)

LoadFromStream (see page 3062)

27.1.20.5.6 TCustomRichEdit.PageRect Property

Specifies the dimensions, in pixels, of the logical page size used when printing the contents of a rich text edit control.

Pascal

```
property PageRect: TRect;
```

C++

```
__property TRect PageRect;
```

Description

Use PageRect to specify the logical page size used when formatting the contents of the rich edit control for printing. Specify the dimensions in pixels.

See Also

Print (see page 3873)

27.1.20.5.7 TCustomRichEdit.Paragraph Property

Specifies the formatting information for the current paragraphs.

Pascal

```
property Paragraph: TParaAttributes;
```

C++

```
__property TParaAttributes Paragraph;
```

Description

Read Paragraph to get the TParaAttributes (see page 4606) object used by the rich edit control to specify paragraph formatting information. Use the TParaAttributes (see page 4606) object to read or write the paragraph formatting information for the current paragraphs. Paragraph formatting information includes alignment, indentation, numbering, and tabs.

Paragraph is a read-only property, because a TCustomRichEdit (see page 3854) object has only one TParaAttributes (see page 4606) object, which cannot be changed. The attributes of the current paragraphs, however, can be changed, by setting the properties of the TParaAttributes (see page 4606) object.

The current paragraphs are the paragraphs that contain the selected text. If no text is selected, the current paragraph is the one containing the cursor.

Note: Paragraph is available only at runtime.

C++ Examples:

```
/*
This example requires only a form.  A Rich Edit control is
created on the form and aligned to take the entire client
area of the form.  A series of lines are written to the Rich
Edit control's Lines property, and the Paragraph property
manipulated such that the first items displayed are bulleted,
a second group of lines are unbulleted and centered, and the
last group is left justified and indented.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TRichEdit *pRich = new TRichEdit(this);
  pRich->Parent = this;
  pRich->Align = alClient;
  pRich->Lines->Clear();
  // set numbering style
  pRich->Paragraph->Numbering = nsBullet;
  pRich->Lines->Add("Introduction");
  pRich->Lines->Add("New members to our team");
  pRich->Lines->Add("New Budget discussion");
  pRich->Lines->Add("Facilities");
  pRich->Lines->Add("Q & A");
  pRich->Paragraph->Numbering = nsNone;
  pRich->Paragraph->Alignment = taCenter;
  pRich->Lines->Add("");
  pRich->Lines->Add("Suggested Topics: ");
  pRich->Lines->Add("");
  pRich->Paragraph->Alignment = taLeftJustify;
  pRich->Paragraph->FirstIndent = 10;
  pRich->Lines->Add("");
  pRich->Lines->Add("Parking lot repair");
  pRich->Lines->Add("Cost overruns");
}
```

Delphi Examples:

```
{
This example requires only a form.  A Rich Edit control is
created on the form and aligned to take the entire client
area of the form.  A series of lines are written to the Rich
Edit control's Lines property, and the Paragraph property
manipulated such that the first items displayed are bulleted,
a second group of lines are unbulleted and centered, and the
last group is left justified and indented.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with TRichEdit.Create(Self) do
  begin
    Parent := Self;
    Align := alClient;
    Lines.Clear;
    // set numbering style
    Paragraph.Numbering := nsBullet;
    Lines.Add('Introduction');
    Lines.Add('New members to our team');
    Lines.Add('New Budget discussion');
    Lines.Add('Facilities');
    Lines.Add('Q & A');
    Paragraph.Numbering := nsNone;
    Paragraph.Alignment := taCenter;
```

```
      Lines.Add('');
      Lines.Add('Suggested Topics:');
      Lines.Add('');
      Paragraph.Alignment := taLeftJustify;
      Paragraph.FirstIndent := 10;
      Lines.Add('');
      Lines.Add('Parking lot repair');
      Lines.Add('Cost overruns');
    end;
  end;
```

See Also

OnSelectionChange (see page 3890)

SelAttributes (see page 3887)

SelStart

SelText

TParaAttributes (see page 4606)

27.1.20.5.8 TCustomRichEdit.PlainText Property

Controls (see page 5711) whether the rich edit control treats the text as plain text or rich text when streaming to or from a file.

Pascal

```
property PlainText: Boolean;
```

C++

```
__property Boolean PlainText;
```

Description

To write the rich text in the control to a plain text file, set PlainText to true before streaming the text to a file. To ignore the rich text information encoded in a file, set PlainText to true before streaming the text to the control. To stream in the rich text attributes encoded in a file, or save the encoding of the rich text attributes to a file, set PlainText to false.

If the rich text attributes of a file are encoded in some format other than rich text format (RTF), it is necessary to use a converter on the text, even when PlainText is true.

Note: Rich edit controls do not directly support streaming. Use the Lines (see page 3882) property to stream to or from a file.

C++ Examples:

```
/*
This example requires two TRichEdit controls placed on the
form.  When the form becomes visible, the first Rich Edit
control will display the rich text in its raw form, including
formatting characters. The second will show the rich text in
its intended format.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
// you may need to change this path to suit your environment
  char const *Path = "..\\OVERVIEW.RTF";
  RichEdit1->PlainText = true;
  RichEdit1->Lines->LoadFromFile(Path);
  RichEdit2->PlainText = false;
  RichEdit2->Lines->LoadFromFile(Path);
}
```

Delphi Examples:

```
{
This example requires two TRichEdit controls placed on the
form.  When the form becomes visible, the first Rich Edit
control will display the rich text in its raw form, including
formatting characters. The second will show the rich text in
its intended format.
}

procedure TForm1.FormCreate(Sender: TObject);
const
  // you may need to change this path to suit your environment
  Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := True;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
  RichEdit2.PlainText := False;
  RichEdit2.Lines.LoadFromFile(Path);
  RichEdit2.ScrollBars := ssVertical;
end;
```

See Also

DefaultConverter (see page 3878)

Lines (see page 3882)

RegisterConversionFormat (see page 3874)

27.1.20.5.9 TCustomRichEdit.SelAttributes Property

Describes the rich text characteristics of the selected text in the rich edit control.

Pascal

```
property SelAttributes: TTextAttributes;
```

C++

```
__property TTextAttributes SelAttributes;
```

Description

Use SelAttributes to discover or set the font characteristics of the currently selected text. SelAttributes specifies characteristics such as font face, color, size, style, and pitch. To change a single attribute of the currently selected text, read SelAttributes, and set one of its properties. To change all of the attributes of the currently selected text, set SelAttributes to a TTextAttributes (see page 4982) object that represents the desired configuration of attributes. If no text is selected, SelAttributes represents the attributes of the cursor position.

When inserting new text, the font characteristics of the new text will match SelAttributes.

Note: SelAttributes is available only at runtime.

C++ Examples:

```
/*
This example requires a TRichEdit and two TButtons. Leave
the default text, usually "RichEdit1", in the lines
property. The DefAttributes will apply to this text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  RichEdit1->SelAttributes->Color = clRed;
  RichEdit1->SelAttributes->Height += 5;
  RichEdit1->Lines->Add("This line of text will be red. ");
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  RichEdit1->DefAttributes->Color = clBlue;
  RichEdit1->DefAttributes->Style =
    RichEdit1->DefAttributes->Style << fsBold << fsItalic;
}
```

Delphi Examples:

```
{
This example requires a TRichEdit and two TButtons. Leave
the default text, usually "RichEdit1", in the lines
property. The DefAttributes will apply to this text.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  with RichEdit1.SelAttributes do
  begin
    Color := clRed;
    Height := Height + 5;
  end;
  RichEdit1.Lines.Add('This line of text will be red.');
end;


procedure TForm1.Button2Click(Sender: TObject);
begin
  RichEdit1.DefAttributes.Color := clBlue;
  RichEdit1.DefAttributes.Style := [fsBold, fsItalic];
end;
```

See Also

DefAttributes (see page 3877)

OnProtectChange (see page 3888)

PlainText (see page 3886)

27.1.20.6 TCustomRichEdit Events

The events of the TCustomRichEdit class are listed here.

Protected Events

	Name	Description
	OnProtectChange (see page 3888)	Occurs when a user attempts to modify text that is marked as protected.
	OnResizeRequest (see page 3889)	Occurs when the text has grown either smaller or larger than the control's window size.
	OnSaveClipboard (see page 3890)	Occurs when the window of the edit control is about to be destroyed and the edit control has delayed rendering of text copied to the clipboard..
	OnSelectionChange (see page 3890)	Occurs when the current selection changes.

27.1.20.6.1 TCustomRichEdit.OnProtectChange Event

Occurs when a user attempts to modify text that is marked as protected.

Pascal

```
property OnProtectChange: TRichEditProtectChange;
```

C++

```
__property TRichEditProtectChange OnProtectChange;
```

Description

Write an OnProtectChange event handler to allow modification of text with the protected attribute. The StartPos and EndPos parameters indicate the beginning and end of the range of text that would be affected by the modification, where 0 is the first character in the text. To allow modification of protected text, set AllowChange to true.

To set or unset the protected attribute for a section of text, select the text and use the Protected property of the rich edit control's SelAttributes (see page 3887) property. To cause text to have the protected attribute by default, use the Protected property of the rich edit object's DefAttributes (see page 3877) property.

See Also

DefAttributes (see page 3877)

SelAttributes (see page 3887)

SelLength

SelStart

27.1.20.6.2 TCustomRichEdit.OnResizeRequest Event

Occurs when the text has grown either smaller or larger than the control's window size.

Pascal

```
property OnResizeRequest: TRichEditResizeEvent;
```

C++

```
__property TRichEditResizeEvent OnResizeRequest;
```

Description

Write an OnResizeRequest event handler to respond to notifications that the edit control is trying to resize due to changes in the text. The Rect parameter gives the desired bounding rectangle that will display all the text, in screen coordinates.

C++ Examples:

```
/*
The following code resizes the rich edit control to
accommodate growing or shrinking lines of text.  This
example requires a button, a TEdit and a TRichEdit.
Double click on the TRichEdit OnResizeRequest event
to create the event handler.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Add(Edit1->Text);
}

void __fastcall TForm1::RichEdit1ResizeRequest(TObject *Sender, TRect &Rect)
{
  ((TRichEdit *)Sender)->BoundsRect = Rect;
}
```

Delphi Examples:

```
{
```

```
The following code resizes the rich edit control to
accommodate growing or shrinking lines of text.  This
example requires a button, a TEdit and a TRichEdit.
Double click on the TRichEdit OnResizeRequest event
to create the event handler.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  RichEdit1.Lines.Add(Edit1.Text);
end;

procedure TForm1.RichEdit1ResizeRequest(Sender: TObject; Rect: TRect);
begin
  (Sender as TRichEdit).BoundsRect := Rect;
end;
```

See Also

BoundsRect (see page 5803)

ClientRect

Text (see page 5842)

27.1.20.6.3 TCustomRichEdit.OnSaveClipboard Event

Occurs when the window of the edit control is about to be destroyed and the edit control has delayed rendering of text copied to the clipboard..

Pascal

```
property OnSaveClipboard: TRichEditSaveClipboard;
```

C++

```
__property TRichEditSaveClipboard OnSaveClipboard;
```

Description

Write an OnSaveClipboard event handler to respond to notifications that the window of the edit control is about to be destroyed and the Clipboard needs information from the edit control. The parameter NumObjects is the number of objects on the Clipboard, and NumChars is the number of characters on the Clipboard. To clear all information about the edit control from the Clipboard, set SaveClipboard to false.

See Also

CopyToClipboard (see page 21246)

CutToClipboard (see page 21249)

27.1.20.6.4 TCustomRichEdit.OnSelectionChange Event

Occurs when the current selection changes.

Pascal

```
property OnSelectionChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSelectionChange;
```

Description

Write an OnSelectionChange event handler to respond to changes in the selected text. Changes in selection can occur when the user changes the selection using the mouse or keyboard, or when the selection is changed through the SelText property.

Portions of the selection can be changed through the Lines (see page 3882) property.

C++ Examples:

```
/*
This example requires a TStatusBar and a TRichEdit.  The
code within the form's OnCreate will take care of sizing the
Rich Edit and setting the OnSelectionChange event to be
handled by RichEdit1SelectionChange (or you can paste the
code directly into the OnSelectionChange event handler.)  As
you type in the Rich Edit control, the current character
location the selection begin and end will display in the
status bar control.
*/
void __fastcall TForm1::RichEdit1SelectionChange(TObject *Sender)
{
  StatusBar1->SimpleText = "  Selection Start: " + IntToStr(RichEdit1->SelStart) +
      "  Selection End: " + IntToStr(RichEdit1->SelStart + RichEdit1->SelLength);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  char const *Path = "..\\OVERVIEW.RTF";
  RichEdit1->Lines->LoadFromFile(Path);
  RichEdit1->Align = alClient;
  RichEdit1->OnSelectionChange = RichEdit1SelectionChange;
  StatusBar1->SimplePanel = True;
}
```

Delphi Examples:

```
{
This example requires a TStatusBar and a TRichEdit.  The
code within the form's OnCreate will take care of sizing the
Rich Edit and setting the OnSelectionChange event to be
handled by RichEdit1SelectionChange (or you can paste the
code directly into the OnSelectionChange event handler.)  As
you type in the Rich Edit control, the current character
location the selection begin and end will display in the
status bar control.
}
procedure TForm1.RichEdit1SelectionChange(Sender: TObject);
begin
  with RichEdit1 do
    begin
    StatusBar1.SimpleText := '  Selection Start: ' + IntToStr(SelStart) +
      '  Selection End: ' + IntToStr(SelStart + SelLength);
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
const
  Path = 'OVERVIEW.RTF';
begin
  with RichEdit1 do
  begin
    Lines.LoadFromFile(Path);
    Align := alClient;
    OnSelectionChange := RichEdit1SelectionChange;
  end;
  StatusBar1.SimplePanel := True;
end;
```

See Also

AutoSelect (see page 21260)

ClearSelection (see page 21245)

Lines (see page 3882)

SelAttributes (see page 3887)

SelectAll (see page 21255)

SelLength

SelStart

SelText

SetSelTextBuf (see page 21258)

27.1.21 TCustomStatusBar Class

TCustomStatusBar is the base class for objects that represent status bars.

Class Hierarchy

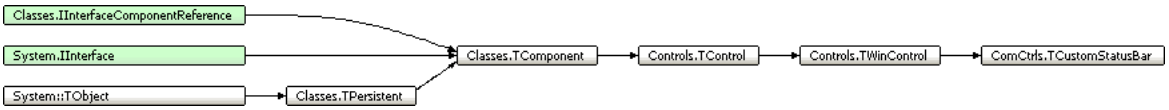

Pascal

```
TCustomStatusBar = class(TWinControl);
```

C++

```
class TCustomStatusBar : public TWinControl;
```

File

ComCtrls

Description

TCustomStatusBar component is the base class for status bar controls. Status bars consist of a row of panels, usually aligned at the bottom of a form, that display information about an application as it runs. Each panel is represented by a TStatusPanel (see page 4825) object listed in the Panels (see page 3912) property.

TCustomStatusBar is intended only as a base class. To add a status bar to a form, use a TCustomStatusBar descendant such as TStatusBar (see page 4769).

See Also

TStatusBar (see page 4769)

TStatusPanel (see page 4825)

TStatusPanels (see page 4834)

27.1.21.1 TCustomStatusBar Members

The following tables list the members exposed by TCustomStatusBar.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.

	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	ChangeScale (see page 3904)	Repositions and resizes the status bar by a specified ratio.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreatePanel (see page 3905)	Creates a new panel that can be added to the Panels (see page 3912) property.
V	CreatePanels (see page 3905)	Creates the TStatusPanels (see page 4834) object that implements the Panels (see page 3912) property.
V	CreateParams (see page 3905)	Initializes the window-creation parameter record when the status bar window is created.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 3906)	Creates the Windows control that represents the status bar.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoHint (see page 3906)	Generates an OnHint (see page 3920) event.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawPanel (see page 3907)	Draws an owner-drawn panel.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.

	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPanelClass (see page 3908)	Returns the class to use for implementing the panels of the status bar.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsFontStored (see page 3908)	Indicates whether the value of the Font property should be saved in the form file.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.

	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 3909)	Sets the value of the Parent property.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnCreatePanelClass (see page 3919)	Occurs when a status panel needs to be created.
	OnDrawPanel (see page 3919)	Occurs when a status panel needs to be redrawn.

	OnHint (see page 3920)	Occurs when the AutoHint (see page 3910) property is true and the status bar receives a hint action.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 3903)	Creates and initializes a TCustomStatusBar (see page 3892) component.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 3903)	Destroys TCustomStatusBar (see page 3892) and frees its memory.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
V	ExecuteAction (see page 3907)	Processes actions for the status bar.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.

	FlipChildren (see page 3908)	Reverses the positions of the status bar's panels.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 3909)	Sets the Left, Top, Width, and Height properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AutoHint (see page 3910)	Specifies whether the status bar's text is set automatically to the current hint.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 3910)	Provides runtime access to the status bar's drawing surface.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.

R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Panels (see page 3912)	Lists the panels (TStatusPanel (see page 4825) objects) in the status bar.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	SimplePanel (see page 3915)	Determines whether the status bar displays a single panel or multiple panels.
	SimpleText (see page 3916)	Contains the string that is displayed in the status bar when SimplePanel (see page 3915) is set to true.
	SizeGrip (see page 3918)	Determines whether the status bar is resizable at runtime.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	UseSystemFont (see page 3918)	Specifies whether the status bar uses the system font.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.

	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.21.2 TCustomStatusBar.Create Constructor

Creates and initializes a TCustomStatusBar (see page 3892) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomStatusBar(TComponent * AOwner);
```

Description

The Create method creates a status bar along with its canvas and status panels.

AOwner is a component, typically the form, that becomes the value of the Owner property.

See Also

TCanvas (see page 14441)

TStatusPanels (see page 4834)

Owner (see page 2807)

27.1.21.3 TCustomStatusBar.Destroy Destructor

Destroys TCustomStatusBar (see page 3892) and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomStatusBar();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks for a nil object before calling Destroy.

The Destroy method destroys the status bar along with its canvas and status panels.

See Also

TCanvas (see page 14441)

TStatusPanels (see page 4834)

Free (see page 22300)

27.1.21.4 TCustomStatusBar Methods

The methods of the TCustomStatusBar class are listed here.

Protected Methods

	Name	Description
	ChangeScale (see page 3904)	Repositions and resizes the status bar by a specified ratio.
	CreatePanel (see page 3905)	Creates a new panel that can be added to the Panels (see page 3912) property.
	CreatePanels (see page 3905)	Creates the TStatusPanels (see page 4834) object that implements the Panels (see page 3912) property.
	CreateParams (see page 3905)	Initializes the window-creation parameter record when the status bar window is created.
	CreateWnd (see page 3906)	Creates the Windows control that represents the status bar.
	DoHint (see page 3906)	Generates an OnHint (see page 3920) event.
	DrawPanel (see page 3907)	Draws an owner-drawn panel.
	GetPanelClass (see page 3908)	Returns the class to use for implementing the panels of the status bar.
	IsFontStored (see page 3908)	Indicates whether the value of the Font property should be saved in the form file.
	SetParent (see page 3909)	Sets the value of the Parent property.

Public Methods

	Name	Description
	ExecuteAction (see page 3907)	Processes actions for the status bar.
	FlipChildren (see page 3908)	Reverses the positions of the status bar's panels.
	SetBounds (see page 3909)	Sets the Left, Top, Width, and Height properties all at once.

27.1.21.4.1 TCustomStatusBar.ChangeScale Method

Repositions and resizes the status bar by a specified ratio.

Pascal

```
procedure ChangeScale(M: Integer; D: Integer); override;
```

C++

```
virtual __fastcall ChangeScale(int M, int D);
```

Description

ChangeScale is called automatically when the form that contains the status bar is rescaled. ChangeScale modifies the position as well as the size of a control. Thus, ChangeScale modifies the control's Top, Left, Width, and Height properties.

The M and D parameters define a fraction by which to scale the control. The M parameter is the multiplier and the D parameter is the divisor. For example, to make a control 75% of its original size, specify the value of M as 75, and the value of D as 100 (75/100). Alternately, the same results are achieved by specifying the value of M as 3, and the value of D as 4 (3/4). Both fractions are equal and result in the control being scaled by the same amount, 75%.

TCustomStatusBar (see page 3892) overrides the inherited method to ensure that the new size of the status bar matches the system font if the UseSystemFont (see page 3918) property is true.

See Also

UseSystemFont (see page 3918)

27.1.21.4.2 TCustomStatusBar.CreatePanel Method

Creates a new panel that can be added to the Panels (see page 3912) property.

Pascal

```
function CreatePanel: TStatusPanel; virtual;
```

C++

```
virtual __fastcall TStatusPanel CreatePanel();
```

Description

The TStatusPanels (see page 4834) object that implements the Panels (see page 3912) property calls CreatePanel when it needs to add a new panel. CreatePanel generates an OnCreatePanelClass (see page 3919) event to determine the class of the new TStatusPanel (see page 4825) object, and then creates an instance of that class. CreatePanel returns the new status panel instance.

See Also

OnCreatePanelClass (see page 3919)

Panels (see page 3912)

AddItem (see page 4838)

27.1.21.4.3 TCustomStatusBar.CreatePanels Method

Creates the TStatusPanels (see page 4834) object that implements the Panels (see page 3912) property.

Pascal

```
function CreatePanels: TStatusPanels; virtual;
```

C++

```
virtual __fastcall TStatusPanels CreatePanels();
```

Description

TCustomStatusBar (see page 3892) calls CreatePanels internally generate the implementation object for the Panels (see page 3912) property.

Override this method to substitute a TStatusPanels (see page 4834) descendant as the implementation of the Panels (see page 3912) property or to provide special initializations to the TStatusPanels (see page 4834) instance.

See Also

Panels (see page 3912)

27.1.21.4.4 TCustomStatusBar.CreateParams Method

Initializes the window-creation parameter record when the status bar window is created.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 3906) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. TCustomStatusBar (see page 3892) overrides CreateParams to customize the parameters to specify a status bar control that reflects the properties of TCustomStatusBar (see page 3892).

The fields of the Params parameter become the parameters to a call to the CreateWindowEx API function.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 3906)

27.1.21.4.5 TCustomStatusBar.CreateWnd Method

Creates the Windows control that represents the status bar.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

The CreateHandle method calls CreateWnd to handle the particulars of creating the status bar window. TCustomStatusBar (see page 3892) overrides the inherited method to ensure that the newly-created window and the TCustomStatusBar (see page 3892) properties correctly reflect each other.

The Windows control for the status bar can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. Thus, CreateWnd may be called more than once during the lifetime of the TCustomStatusBar (see page 3892) instance.

See Also

CreateHandle (see page 6135)

27.1.21.4.6 TCustomStatusBar.DoHint Method

Generates an OnHint (see page 3920) event.

Pascal

```
function DoHint: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoHint();
```

Description

DoHint is called automatically when the value of AutoHint (see page 3910) is true and the status bar receives notification that it should display a hint.

If there is an OnHint (see page 3920) event handler, DoHint calls the event handler and returns true. If no OnHint (see page 3920) event handler is assigned, DoHint returns false.

See Also

OnHint (see page 3920)

AutoHint (see page 3910)

27.1.21.4.7 TCustomStatusBar.DrawPanel Method

Draws an owner-drawn panel.

Pascal

```
procedure DrawPanel(Panel: TStatusPanel; const Rect: TRect); dynamic;
```

C++

```
__fastcall DrawPanel(TStatusPanel Panel, const TRect Rect);
```

Description

DrawPanel is called automatically for each status panel that has a Style of psOwnerDraw. If there is an event handler for the OnDrawPanel (see page 3919) event, DrawPanel calls the OnDrawPanel (see page 3919) event handler. Otherwise, it fills the panel background using the color specified by the Color property.

Panel is the owner drawn status panel that needs to be drawn.

Rect indicates the dimensions of that panel on the status bar's canvas.

See Also

Style (see page 4830)

OnDrawPanel (see page 3919)

Color (see page 5812)

Canvas (see page 3910)

27.1.21.4.8 TCustomStatusBar.ExecuteAction Method

Processes actions for the status bar.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

When an action executes, the application makes a series of calls to respond to that action. If, in the course of that processing, the application identifies the status bar as a potential target of the action, it calls the status bar's ExecuteAction method.

If the action in a hint action (THintAction), ExecuteAction checks the AutoHint (see page 3910) property, and if it is true, displays the hint and returns true. Otherwise, it allows the inherited method to respond to any other actions.

The Action parameter specifies the action that was invoked.

ExecuteAction returns true if the action was successfully dispatched, and false if the component could not handle the action.

See Also

AutoHint (see page 3910)

Execute (see page 2712)

THintAction (see page 20707)

27.1.21.4.9 TCustomStatusBar.FlipChildren Method

Reverses the positions of the status bar's panels.

Pascal

```
procedure FlipChildren(AllLevels: Boolean); override;
```

C++

```
virtual __fastcall FlipChildren(Boolean AllLevels);
```

Description

FlipChildren iterates through the panels of the status bar, reversing their order but otherwise leaving their properties intact.

AllLevels specifies whether FlipChildren should be called recursively on the control's children. TCustomStatusBar (see page 3892) ignores this parameter when reversing the panels.

See Also

BiDiMode (see page 5803)

Panels (see page 3912)

27.1.21.4.10 TCustomStatusBar.GetPanelClass Method

Returns the class to use for implementing the panels of the status bar.

Pascal

```
function GetPanelClass: TStatusPanelClass; virtual;
```

C++

```
virtual __fastcall TStatusPanelClass GetPanelClass();
```

Description

The CreatePanel (see page 3905) method calls GetPanelClass to determine the default class to use when creating new panels for the status bar. This class can then be changed by an OnCreatePanelClass (see page 3919) event handler.

TCustomStatusBar (see page 3892) returns TStatusPanel (see page 4825) as the default class for status bar panels. Descendants that use a TStatusPanel (see page 4825) descendant by default can override this method to indicate a different default class.

See Also

OnCreatePanelClass (see page 3919)

CreatePanel (see page 3905)

27.1.21.4.11 TCustomStatusBar.IsFontStored Method

Indicates whether the value of the Font property should be saved in the form file.

Pascal

```
function IsFontStored: Boolean;
```

C++

```
__fastcall Boolean IsFontStored();
```

Description

TCustomStatusBar (see page 3892) uses IsFontStored internally to determine whether the value of the Font property should be stored in a form file with the other properties of the status bar object. IsFontStored checks the UseSystemFont (see page 3918), ParentFont, and DesktopFont properties. If any of these is true, the status bar gets its font from an external source, and the value of the Font property does not need to be saved.

See Also

UseSystemFont (see page 3918)

ParentFont (see page 5834)

DesktopFont (see page 5818)

27.1.21.4.12 TCustomStatusBar.SetBounds Method

Sets the Left, Top, Width, and Height properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

TCustomStatusBar (see page 3892) overrides SetBounds in order to update the size grip.

ALeft, ATop, AWidth, and AHeight specify the new values for the Left, Top, Width, and Height properties, respectively.

See Also

SetBounds (see page 5783)

SizeGrip (see page 3918)

27.1.21.4.13 TCustomStatusBar.SetParent Method

Sets the value of the Parent property.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected write implementation of the Parent property.

AParent is the new parent of the status bar.

See Also

Parent

27.1.21.5 TCustomStatusBar Properties

The properties of the TCustomStatusBar class are listed here.

Public Properties

	Name	Description
	AutoHint (see page 3910)	Specifies whether the status bar's text is set automatically to the current hint.
R	Canvas (see page 3910)	Provides runtime access to the status bar's drawing surface.
	Panels (see page 3912)	Lists the panels (TStatusPanel (see page 4825) objects) in the status bar.
	SimplePanel (see page 3915)	Determines whether the status bar displays a single panel or multiple panels.
	SimpleText (see page 3916)	Contains the string that is displayed in the status bar when SimplePanel (see page 3915) is set to true.
	SizeGrip (see page 3918)	Determines whether the status bar is resizable at runtime.
	UseSystemFont (see page 3918)	Specifies whether the status bar uses the system font.

27.1.21.5.1 TCustomStatusBar.AutoHint Property

Specifies whether the status bar's text is set automatically to the current hint.

Pascal

```
property AutoHint: Boolean;
```

C++

```
__property Boolean AutoHint;
```

Description

Use AutoHint to specify whether the status bar's text is set automatically to the current hint. When AutoHint is true, the status bar automatically responds to hint actions by displaying the long version of the hint's text in the first panel.

See Also

Hint (see page 13712)

Hint (see page 5827)

ShowHint (see page 13721)

THintAction (see page 20707)

OnHint (see page 3920)

27.1.21.5.2 TCustomStatusBar.Canvas Property

Provides runtime access to the status bar's drawing surface.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

The read-only Canvas property provides access to the status bar's drawing surface that you can use when implementing a handler for the OnDrawPanel (see page 3919) event.

C++ Examples:

```
/*
The following code is the OnDrawPanel event handler of a
status bar.  It draws each panel of the status bar, adding
text and icons stored in ImageList1. The image list contains
as many icons as there are header sections.  This example
requires a populated image list and a statusbar with several
panels added to the Panels property.  Select each panel and
set the Style property to psOwnerDraw.
*/
void __fastcall TForm1::StatusBar1DrawPanel(TStatusBar *StatusBar,
        TStatusPanel *Panel, const TRect &Rect)
{
  TCanvas *canvas = StatusBar->Canvas;
  canvas->Brush->Color = clRed;
  canvas->FillRect(Rect);
  canvas->Font->Color = clYellow;
  ImageList1->Draw(canvas,Rect.Left,Rect.Top, Panel->Index, true);
  canvas->TextOut(Rect.left + 30, Rect.top + 2, "Panel" + IntToStr(Panel->Index));
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TStatusBar *myStatusBar;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TStatusBar")
  {
    myStatusBar = (TStatusBar *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myStatusBar->Canvas->MoveTo(myStatusBar->ClientRect.Left, myStatusBar->ClientRect.Top);
    myStatusBar->Canvas->LineTo(myStatusBar->ClientRect.Right, myStatusBar->ClientRect.Bottom);
    myStatusBar = NULL;
  }
}
```

Delphi Examples:

```
{
The following code is the OnDrawPanel event handler of a
status bar.  It draws each panel of the status bar, adding
text and icons stored in ImageList1. The image list contains
as many icons as there are header sections.  This example
requires a populated image list and a statusbar with several
panels added to the Panels property.  Select each panel and
set the Style property to psOwnerDraw.
}
procedure TForm1.StatusBar1DrawPanel(StatusBar: TStatusBar;
  Panel: TStatusPanel; const Rect: TRect);
begin
with StatusBar1.Canvas do
  begin
    Brush.Color := clRed;
    FillRect(Rect);
    Font.Color := clYellow;
    ImageList1.Draw(StatusBar1.Canvas,Rect.Left,Rect.Top,Panel.Index);
    TextOut(Rect.left + 30, Rect.top + 2, 'Panel' + IntToStr(Panel.Index));
  end;
end;
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
```

```
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TStatusBar) then
  begin
    myStatusBar:= (ListBox1.Items.Objects[Index] as TStatusBar);
    with myStatusBar.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myStatusBar.Canvas.MoveTo(Left, Top);
      myStatusBar.Canvas.LineTo(Right, Bottom);
    end;
  myStatusBar:= nil;
  end;
```

See Also

TCanvas (see page 14441)

OnDrawPanel (see page 3919)

27.1.21.5.3 TCustomStatusBar.Panels Property

Lists the panels (TStatusPanel (see page 4825) objects) in the status bar.

Pascal

```
property Panels: TStatusPanels;
```

C++

```
__property TStatusPanels Panels;
```

Description

The Panels property holds a TStatusPanels (see page 4834)—that is, a collection of TStatusPanel (see page 4825) objects. At design time, you can add, remove, or modify panels with the Panels editor. To open the Panels editor, select the Panels property in the Object Inspector, then double-click in the Value column to the right or click the ellipsis (...) button.

C++ Examples:

```
/*
The following code is the OnDrawPanel event handler of a
status bar.  It draws each panel of the status bar, adding
text and icons stored in ImageList1. The image list contains
as many icons as there are header sections.  This example
requires a populated image list and a statusbar with several
panels added to the Panels property.  Select each panel and
set the Style property to psOwnerDraw.
*/
void __fastcall TForm1::StatusBar1DrawPanel(TStatusBar *StatusBar,
        TStatusPanel *Panel, const TRect &Rect)
{
  TCanvas *canvas = StatusBar->Canvas;
  canvas->Brush->Color = clRed;
  canvas->FillRect(Rect);
  canvas->Font->Color = clYellow;
  ImageList1->Draw(canvas,Rect.Left,Rect.Top, Panel->Index, true);
  canvas->TextOut(Rect.left + 30, Rect.top + 2, "Panel" + IntToStr(Panel->Index));
}
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
```

```
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following code is the OnDrawPanel event handler of a
status bar.  It draws each panel of the status bar, adding
text and icons stored in ImageList1. The image list contains
as many icons as there are header sections.  This example
requires a populated image list and a statusbar with several
panels added to the Panels property.  Select each panel and
set the Style property to psOwnerDraw.
}
procedure TForm1.StatusBar1DrawPanel(StatusBar: TStatusBar;
  Panel: TStatusPanel; const Rect: TRect);
```

```
begin
with StatusBar1.Canvas do
  begin
    Brush.Color := clRed;
    FillRect(Rect);
    Font.Color := clYellow;
    ImageList1.Draw(StatusBar1.Canvas,Rect.Left,Rect.Top,Panel.Index);
    TextOut(Rect.left + 30, Rect.top + 2, 'Panel' + IntToStr(Panel.Index));
  end;
end;
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

TStatusPanel (see page 4825)

27.1.21.5.4 TCustomStatusBar.SimplePanel Property

Determines whether the status bar displays a single panel or multiple panels.

Pascal

```
property SimplePanel: Boolean;
```

C++

```
__property Boolean SimplePanel;
```

Description

If SimplePanel is set to true, the status bar consists of a single panel displaying the text in SimpleText (see page 3916). If SimplePanel is set to false, the status bar displays a separate panel for each item in its Panels (see page 3912) property.

Note: When SimplePanel is false, you can't use the SimpleText (see page 3916) property to display text in the status bar. Instead, you must set the Text property on one of the panel objects.

C++ Examples:

```
/*
This example adds text to a TRichEdit and to a StatusBar.
This example requires no additional controls. All controls are
created dynamically within the form's OnCreate event handler.
Note: Be sure to add ComCtrls to your uses clause to support
TRichEdit.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TStatusBar *status = new TStatusBar(this);
  // Display the StatusBar control on the surface
  // of the form.
  status->Parent = this;
  // Allow a single line of text -- no panels.
  status->SimplePanel = true;
  // Provide sample text.
  // Create a new instance of TRichEdit, set its
  // properties, and display on the form.
  TRichEdit *rich = new TRichEdit(this);
  // Display the RichEdit control on the surface
  // of the form.
  rich->Parent = this;
  // The RichText control takes all space left
  // over from the StatusBar.
  rich->Align = alClient;
  // Add 50 lines to the RichText
  for (int i = 1; i <= 50; i++)
    rich->Lines->Add(AnsiString("This is line ") + IntToStr(i));
  rich->SelStart = 0; // set text cursor to first character
  status->SimpleText = IntToStr(rich->Lines->Count) + " Lines loaded";
}
```

Delphi Examples:

```
{
This example adds text to a TRichEdit and to a StatusBar.
```

```
This example requires no additional controls. All controls are
created dynamically within the form's OnCreate event handler.
Note: Be sure to add ComCtrls to your uses clause to support
TRichEdit.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  with TStatusBar.Create(Self) do
  begin
    // Display the StatusBar control on the surface
    // of the form.
    Parent := Self;
    // Allow a single line of text -- no panels.
    SimplePanel := True;
    // Provide sample text.
    // Create a new instance of TRichEdit, set its
    // properties, and display on the form.
    with TRichEdit.Create(Self) do
    begin
      // Display the RichEdit control on the surface
      // of the form.
      Parent := Self;
      // The RichText control takes all space left
      // over from the StatusBar.
      Align := alClient;
      // Add 50 lines to the RichText
      for i := 1 to 50 do
        Lines.Add('Adding line ' + IntToStr(i));
      SelStart := 0; // set text cursor to first character
      SimpleText := 'Number of Lines loaded to RichText: ' +
        IntToStr(Lines.Count);
    end;
  end;
end;
```

See Also

SimpleText (see page 3916)

Panels (see page 3912)

27.1.21.5.5 TCustomStatusBar.SimpleText Property

Contains the string that is displayed in the status bar when SimplePanel (see page 3915) is set to true.

Pascal

```
property SimpleText: string;
```

C++

```
__property AnsiString SimpleText;
```

Description

SimpleText contains a text string to be displayed in the status panel when SimplePanel (see page 3915) is true.

C++ Examples:

```
/*
This example adds text to a TRichEdit and to a StatusBar.
This example requires no additional controls. All controls are
created dynamically within the form's OnCreate event handler.
Note: Be sure to add ComCtrls to your uses clause to support
TRichEdit.
```

```
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TStatusBar *status = new TStatusBar(this);
  // Display the StatusBar control on the surface
  // of the form.
  status->Parent = this;
  // Allow a single line of text -- no panels.
  status->SimplePanel = true;
  // Provide sample text.
  // Create a new instance of TRichEdit, set its
  // properties, and display on the form.
  TRichEdit *rich = new TRichEdit(this);
  // Display the RichEdit control on the surface
  // of the form.
  rich->Parent = this;
  // The RichText control takes all space left
  // over from the StatusBar.
  rich->Align = alClient;
  // Add 50 lines to the RichText
  for (int i = 1; i <= 50; i++)
    rich->Lines->Add(AnsiString("This is line ") + IntToStr(i));
  rich->SelStart = 0; // set text cursor to first character
  status->SimpleText = IntToStr(rich->Lines->Count) + " Lines loaded";
}
```

Delphi Examples:

```
{
This example adds text to a TRichEdit and to a StatusBar.
This example requires no additional controls. All controls are
created dynamically within the form's OnCreate event handler.
Note: Be sure to add ComCtrls to your uses clause to support
TRichEdit.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  with TStatusBar.Create(Self) do
  begin
    // Display the StatusBar control on the surface
    // of the form.
    Parent := Self;
    // Allow a single line of text -- no panels.
    SimplePanel := True;
    // Provide sample text.
    // Create a new instance of TRichEdit, set its
    // properties, and display on the form.
    with TRichEdit.Create(Self) do
    begin
      // Display the RichEdit control on the surface
      // of the form.
      Parent := Self;
      // The RichText control takes all space left
      // over from the StatusBar.
      Align := alClient;
      // Add 50 lines to the RichText
      for i := 1 to 50 do
        Lines.Add('Adding line ' + IntToStr(i));
      SelStart := 0; // set text cursor to first character
      SimpleText := 'Number of Lines loaded to RichText: ' +
        IntToStr(Lines.Count);
    end;
  end;
```

```
end;
```

See Also

SimplePanel (see page 3915)

27.1.21.5.6 TCustomStatusBar.SizeGrip Property

Determines whether the status bar is resizable at runtime.

Pascal

```
property SizeGrip: Boolean;
```

C++

```
__property Boolean SizeGrip;
```

Description

If SizeGrip is set to true, the status bar has a triangular grip on the lower right corner. The user can resize the status bar by dragging the grip with the mouse.

A size grip will not be drawn unless the Parent property refers to the child of a TCustomForm descendant with a BorderStyle of bsSizeable or bsSizeToolWin.

Note: If Align is set to alBottom, alRight, or alClient, the sizing grip will not work. If the Alignment property of the last panel in the status bar is set to taRightJustify, the sizing grip will truncate the panel's text.

See Also

Align (see page 5798)

Alignment (see page 4829)

Parent

BorderStyle (see page 13900)

27.1.21.5.7 TCustomStatusBar.UseSystemFont Property

Specifies whether the status bar uses the system font.

Pascal

```
property UseSystemFont: Boolean;
```

C++

```
__property Boolean UseSystemFont;
```

Description

UseSystemFont specifies whether the status bar uses the system font. To change fonts, use the Font property.

See Also

Font (see page 5823)

ParentFont (see page 5834)

DesktopFont (see page 5818)

27.1.21.6 TCustomStatusBar Events

The events of the TCustomStatusBar class are listed here.

Public Events

	Name	Description
	OnCreatePanelClass (see page 3919)	Occurs when a status panel needs to be created.
	OnDrawPanel (see page 3919)	Occurs when a status panel needs to be redrawn.
	OnHint (see page 3920)	Occurs when the AutoHint (see page 3910) property is true and the status bar receives a hint action.

27.1.21.6.1 TCustomStatusBar.OnCreatePanelClass Event

Occurs when a status panel needs to be created.

Pascal

```
property OnCreatePanelClass: TSBCreatePanelClassEvent;
```

C++

```
__property TSBCreatePanelClassEvent OnCreatePanelClass;
```

Description

Write an OnCreatePanelClass event handler to specify a custom TStatusPanel (see page 4825) descendant that the status bar should use to implement its panels.

See Also

Panels (see page 3912)

TStatusPanel (see page 4825)

27.1.21.6.2 TCustomStatusBar.OnDrawPanel Event

Occurs when a status panel needs to be redrawn.

Pascal

```
property OnDrawPanel: TCustomDrawPanelEvent;
```

C++

```
__property TCustomDrawPanelEvent OnDrawPanel;
```

Description

The OnDrawPanel event occurs when a status panel needs to be redisplayed—for example, when the user resizes the status bar. OnDrawPanel occurs only if the status panel's Style property is set to psOwnerDraw.

C++ Examples:

```
/*
The following code is the OnDrawPanel event handler of a
status bar.  It draws each panel of the status bar, adding
text and icons stored in ImageList1. The image list contains
as many icons as there are header sections.  This example
requires a populated image list and a statusbar with several
panels added to the Panels property.  Select each panel and
set the Style property to psOwnerDraw.
*/
void __fastcall TForm1::StatusBar1DrawPanel(TStatusBar *StatusBar,
```

```
        TStatusPanel *Panel, const TRect &Rect)
{
  TCanvas *canvas = StatusBar->Canvas;
  canvas->Brush->Color = clRed;
  canvas->FillRect(Rect);
  canvas->Font->Color = clYellow;
  ImageList1->Draw(canvas,Rect.Left,Rect.Top, Panel->Index, true);
  canvas->TextOut(Rect.left + 30, Rect.top + 2, "Panel" + IntToStr(Panel->Index));
}
```

Delphi Examples:

```
{
The following code is the OnDrawPanel event handler of a
status bar.  It draws each panel of the status bar, adding
text and icons stored in ImageList1. The image list contains
as many icons as there are header sections.  This example
requires a populated image list and a statusbar with several
panels added to the Panels property.  Select each panel and
set the Style property to psOwnerDraw.
}
procedure TForm1.StatusBar1DrawPanel(StatusBar: TStatusBar;
  Panel: TStatusPanel; const Rect: TRect);
begin
with StatusBar1.Canvas do
  begin
    Brush.Color := clRed;
    FillRect(Rect);
    Font.Color := clYellow;
    ImageList1.Draw(StatusBar1.Canvas,Rect.Left,Rect.Top,Panel.Index);
    TextOut(Rect.left + 30, Rect.top + 2, 'Panel' + IntToStr(Panel.Index));
  end;
end;
```

See Also

Style (see page 4830)

27.1.21.6.3 TCustomStatusBar.OnHint Event

Occurs when the AutoHint (see page 3910) property is true and the status bar receives a hint action.

Pascal

```
property OnHint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnHint;
```

Description

Write an OnHint event handler to customize the way the status bar displays hints when the AutoHint (see page 3910) property is true. When AutoHint (see page 3910) is true and the status bar needs to display a hint, it generates an OnHint event. If the status bar has an OnHint event handler, then the status bar assumes that the event handler displays the hint and does not set the panel text to the value of the application's current hint. If there is no OnHint event handler (and AutoHint (see page 3910) is true), the status bar displays the text of the hint in the first panel.

OnHint does not occur if the AutoHint (see page 3910) property is false.

27.1.22 TCustomTabControl Class

TCustomTabControl is the base type for tab control components such as TTabControl (see page 4841) and TPageControl (see page 4459).

Class Hierarchy

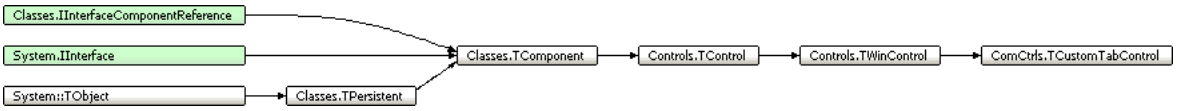

Pascal

```
TCustomTabControl = class(TWinControl);
```

C++

```
class TCustomTabControl : public TWinControl;
```

File

ComCtrls

Description

Use TCustomTabControl as a base class when defining a control that includes a set of notebook style tabs across the top. TCustomTabControl introduces many properties, methods, and events to manage appearance and behavior of the tabs, and to respond when the user selects one of the tabs.

Note: When using one of the page controls, if you want to restrict a user from switching to a tab, you cannot set TTabSheet.Enabled to **false** to accomplish that restriction. Instead, use the OnChanging (see page 3955) event to prevent the user from selecting a tab.

See Also

TPageControl (see page 4459)

TTabControl (see page 4841)

27.1.22.1 TCustomTabControl Members

The following tables list the members exposed by TCustomTabControl.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnChange (see page 3952)	Occurs after a new tab is selected.
	OnChanging (see page 3955)	Occurs immediately before a new tab is selected.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDrawTab (see page 3956)	Occurs when a tab is about to be drawn.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetImageIndex (see page 3956)	Occurs when a tab is about to display its associated image.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.

	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AdjustClientRect (see page 3933)	Calculates the control's display area given a rectangle.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanChange (see page 3933)	Change (see page 3934) permission event dispatcher.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	CanShowTab (see page 3934)	Selection permission event dispatcher.
	Change (see page 3934)	Pending tab change event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 3935)	Sets up the window-creation parameters for the tab control.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 3935)	Creates the window for a tab control.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.

	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawTab (see page 3935)	Tab drawing event dispatcher.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetImageIndex (see page 3936)	Generates an OnGetImageIndex (see page 3956) event
	GetOwner (see page 2778)	Returns the owner of a component.

	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 3937)	Form load event dispatcher.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 3937)	Event dispatcher for creation and destruction of child objects.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.

	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetTabIndex (see page 3938)	This is SetTabIndex, a member of class TCustomTabControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateTabImages (see page 3939)	Updates the tab images.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
R	DisplayRect (see page 3941)	Specifies the bounding rectangle of a tab control client area.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	HotTrack (see page 3941)	Determines whether labels on the tab under the mouse are automatically highlighted.
	Images (see page 3941)	Specifies the images drawn in tabs.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	MultiLine (see page 3942)	Determines whether the tabs can appear on more than one row.
	MultiSelect (see page 3943)	Specifies whether multiple tabs can be selected.
	OwnerDraw (see page 3944)	Specifies whether the tab control handles its own painting.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	RaggedRight (see page 3944)	Specifies whether rows of tabs stretch to fill the width of the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ScrollOpposite (see page 3944)	Determines how the rows of tabs are scrolled in a multi-line tab control.
	Style (see page 3945)	Specifies the style of the tab control.
	TabHeight (see page 3945)	Specifies the height, in pixels, of the tabs in the tab control.
	TabIndex (see page 3946)	Identifies the selected tab on a tab control.
	TabPosition (see page 3948)	Determines whether tabs appear at the top or bottom.
	Tabs (see page 3948)	Contains the list of text strings that label the tabs of the tab control.
	TabWidth (see page 3950)	Specifies the horizontal size, in pixels, of the tabs in the tab control.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 3932)	Creates and initializes an instance of TCustomTabControl (see page 3921).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 3932)	Destroys an instance of TCustomTabControl (see page 3921).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHitTestInfoAt (see page 3936)	Returns information about the location of a point relative to the client area of the tab control.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	IndexOfTabAt (see page 3936)	Indicates the index of the tab at a specified point.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	RowCount (see page 3938)	Sets TabIndex (see page 3946) property.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollTabs (see page 3938)	Scrolls the tabs that are visible when the tab control is not multi-line.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	TabRect (see page 3939)	Returns the bounding rectangle for a specified tab.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 3940)	Gives access to the tab control's canvas.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 3950)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.

	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.22.2 TCustomTabControl.Create Constructor

Creates and initializes an instance of TCustomTabControl (see page 3921).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomTabControl(TComponent * AOwner);
```

Description

Call Create to instantiate a tab control at runtime. Tab controls placed on forms at design time are created automatically. Create calls the inherited Create method and sets the initial values for the tab control, including the creation of a tab list that contains the text strings for the tabs.

AOwner is a component, typically the form, that is responsible for freeing the tab control. It becomes the value of the Owner property.

See Also

Create (see page 6123)

Owner (see page 2807)

27.1.22.3 TCustomTabControl.Destroy Destructor

Destroys an instance of TCustomTabControl (see page 3921).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomTabControl();
```

Description

Do not call Destroy directly in an application. Instead, use Free. Free checks that the tab control is not nil, and only then calls Destroy. Destroy frees the tab list that holds the list of text strings for the tabs and calls the inherited destructor.

See Also

Free (see page 22300)

27.1.22.4 TCustomTabControl Methods

The methods of the TCustomTabControl class are listed here.

Protected Methods

	Name	Description
	AdjustClientRect (see page 3933)	Calculates the control's display area given a rectangle.
	CanChange (see page 3933)	Change (see page 3934) permission event dispatcher.
	CanShowTab (see page 3934)	Selection permission event dispatcher.
	Change (see page 3934)	Pending tab change event dispatcher.
	CreateParams (see page 3935)	Sets up the window-creation parameters for the tab control.
	CreateWnd (see page 3935)	Creates the window for a tab control.
	DrawTab (see page 3935)	Tab drawing event dispatcher.
	GetImageIndex (see page 3936)	Generates an OnGetImageIndex (see page 3956) event
	Loaded (see page 3937)	Form load event dispatcher.
	Notification (see page 3937)	Event dispatcher for creation and destruction of child objects.
	SetTabIndex (see page 3938)	This is SetTabIndex, a member of class TCustomTabControl.
	UpdateTabImages (see page 3939)	Updates the tab images.

Public Methods

	Name	Description
	GetHitTestInfoAt (see page 3936)	Returns information about the location of a point relative to the client area of the tab control.
	IndexOfTabAt (see page 3936)	Indicates the index of the tab at a specified point.
	RowCount (see page 3938)	Sets TabIndex (see page 3946) property.
	ScrollTabs (see page 3938)	Scrolls the tabs that are visible when the tab control is not multi-line.
	TabRect (see page 3939)	Returns the bounding rectangle for a specified tab.

27.1.22.4.1 TCustomTabControl.AdjustClientRect Method

Calculates the control's display area given a rectangle.

Pascal

```
procedure AdjustClientRect(var Rect: TRect); override;
```

C++

```
virtual __fastcall AdjustClientRect(TRect Rect);
```

Description

AdjustClientRect is called internally when the tab control needs accurate information on where to place child controls within the client area. As implemented in TCustomTabControl (see page 3921), returns the DisplayRect (see page 3941) property as the Rect parameter.

See Also

DisplayRect (see page 3941)

27.1.22.4.2 TCustomTabControl.CanChange Method

Change (see page 3934) permission event dispatcher.

Pascal

```
function CanChange: Boolean; dynamic;
```

C++

```
__fastcall Boolean CanChange();
```

Description

CanChange is called automatically when an attempt is made to change the selected tab. CanChange generates an OnChanging (see page 3955) event so that an OnChanging (see page 3955) event handler can prevent the selection from changing. If there is no OnChanging (see page 3955) event handler, CanChange returns true, allowing the change.

Override CanChange to specify additional processing before the OnChanging (see page 3955) event handler executes or to block the OnChanging (see page 3955) event.

See Also

OnChanging (see page 3955)

27.1.22.4.3 TCustomTabControl.CanShowTab Method

Selection permission event dispatcher.

Pascal

```
function CanShowTab(TabIndex: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanShowTab(int TabIndex);
```

Description

CanShowTab is called automatically when an attempt is made to show a tab. CanShowTab is exposed as a protected method so that descendants can override this method to provide application-specific criteria that determines whether a tab is accessible.

TabIndex (see page 3946) is the index of the tab in the list of labels maintained by the Tabs (see page 3948) property.

CanShowTab returns a Boolean value that indicates whether the tab specified in TabIndex (see page 3946) can be accessed.

Tip: CanShowTab can be used with such methods as CanFocus (TWinControl) to determine whether the tab can both be accessed and can receive focus. Criteria for an overridden CanShowTab might include checking whether the tab is enabled.

See Also

CanFocus (see page 6131)

Tabs (see page 3948)

27.1.22.4.4 TCustomTabControl.Change Method

Pending tab change event dispatcher.

Pascal

```
procedure Change; dynamic;
```

C++

```
__fastcall Change();
```

Description

Change is called automatically when the selected tab changes. Override Change to specify additional processing before the OnChange (see page 3952) event handler executes, or to block the OnChange (see page 3952) event.

See Also

OnChange (see page 3952)

27.1.22.4.5 TCustomTabControl.CreateParams Method

Sets up the window-creation parameters for the tab control.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

Tab controls call CreateParams internally when the window for the tab control must be generated. CreateParams overrides the inherited method to specify a tab control window with the current property settings.

See Also

CreateWnd (see page 3935)

27.1.22.4.6 TCustomTabControl.CreateWnd Method

Creates the window for a tab control.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the tab control's window is recreated. CreateWnd calls the inherited method, before assigning any saved tab text strings in internal storage.

See Also

DestroyWnd (see page 6141)

27.1.22.4.7 TCustomTabControl.DrawTab Method

Tab drawing event dispatcher.

Pascal

```
procedure DrawTab(TabIndex: Integer; const Rect: TRect; Active: Boolean); virtual;
```

C++

```
virtual __fastcall DrawTab(int TabIndex, const TRect Rect, Boolean Active);
```

Description

Override DrawTab in a derived class to control the painting of tabs.

TabIndex (see page 3946) is the index of the tab that requires painting.

Rect is the area in which the tab should be painted.

Active is whether the tab is active.

Use the Canvas (see page 3940) property to paint the tab.

See Also

OnDrawTab (see page 3956)

27.1.22.4.8 TCustomTabControl.GetHitTestInfoAt Method

Returns information about the location of a point relative to the client area of the tab control.

Pascal

```
function GetHitTestInfoAt(X: Integer; Y: Integer): THitTests;
```

C++

```
__fastcall THitTests GetHitTestInfoAt(int X, int Y);
```

Description

Call GetHitTestInfoAt to determine what portion of the tab control, if any, sits under the point specified by the X and Y parameters.

GetHitTestInfo returns a THitTests (see page 5488) type. This set describes the possible elements under the mouse. When interpreting these values, each tab can be considered an "item" of the tab control.

See Also

IndexOfTabAt (see page 3936)

27.1.22.4.9 TCustomTabControl.GetImageIndex Method

Generates an OnGetImageIndex (see page 3956) event

Pascal

```
function GetImageIndex(TabIndex: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int GetImageIndex(int TabIndex);
```

Description

Override GetImageIndex in a derived class to specify the index of the image to display for a particular tab.

TabIndex (see page 3946) is the index of the tab that is about to be displayed.

Integer is the index of the image in the Images (see page 3941) property that will be displayed in the tab.

See Also

OnGetImageIndex (see page 3956)

27.1.22.4.10 TCustomTabControl.IndexOfTabAt Method

Indicates the index of the tab at a specified point.

Pascal

```
function IndexOfTabAt(X: Integer; Y: Integer): Integer;
```

C++

```
__fastcall int IndexOfTabAt(int X, int Y);
```

Description

Use IndexOfTabAt to locate which tab is at the point specified by X and Y. If the point with X-coordinate X and Y-coordinate Y is over a tab, IndexOfTabAt returns the 0-offset index of that tab in the Tabs (see page 3948) property (TTabControl (see page 4841)) or in the Pages property (TPageControl (see page 4459)). If the point defined by X and Y is not over a tab, IndexOfTabAt returns –1.

See Also

GetHitTestInfoAt (see page 3936)

Tabs (see page 3948)

Pages (see page 4498)

DisplayRect (see page 3941)

27.1.22.4.11 TCustomTabControl.Loaded Method

Form load event dispatcher.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

The protected Loaded method is called automatically when the form that contains the tab control is loaded into memory. It provides an opportunity for the tab control to perform any initializations after the control and all other components in the form are loaded into memory, but before the first paint operation.

In TCustomTabControl (see page 3921), Loaded ensures that the layout of the tabs accommodate the images specified by the Images (see page 3941) property.

See Also

Images (see page 3941)

27.1.22.4.12 TCustomTabControl.Notification Method

Event dispatcher for creation and destruction of child objects.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification checks whether the notification indicates that the component that implements the Images (see page 3941) property is about to be destroyed, and if so, sets that property to nil (Delphi) or NULL (C++).

See Also

Images (see page 3941)

27.1.22.4.13 TCustomTabControl.RowCount Method

Sets TabIndex (see page 3946) property.

Pascal

```
function RowCount: Integer;
```

C++

```
__fastcall int RowCount();
```

Description

RowCount is called to change the value of the TabIndex (see page 3946) property. Descendant classes should override RowCount to provide special implementations of this property.

See Also

TabIndex (see page 3946)

27.1.22.4.14 TCustomTabControl.ScrollTabs Method

Scrolls the tabs that are visible when the tab control is not multi-line.

Pascal

```
procedure ScrollTabs(Delta: Integer);
```

C++

```
__fastcall ScrollTabs(int Delta);
```

Description

Call ScrollTabs to change the tabs that are visible in the tab control. When Delta is positive, Delta additional tabs to the right are scrolled into view. When Delta is negative, the tabs to the left are scrolled into view, with the absolute value of Delta indicating how many tabs are scrolled. If there are fewer than the absolute value of Delta tabs in the indicated direction, ScrollTabs scrolls as far as possible.

ScrollTabs has no effect if there are no scrolled tabs, either because the control is wide enough to display all tabs or because the MultiLine (see page 3942) property is true.

Note: ScrollTabs scrolls the visible tabs. It does not change the current selection. Calling ScrollTabs may scroll the selected tab out of view.

See Also

Tabs (see page 3948)

MultiLine (see page 3942)

27.1.22.4.15 TCustomTabControl.SetTabIndex Method

Pascal

```
procedure SetTabIndex(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetTabIndex(int Value);
```

Description

This is SetTabIndex, a member of class TCustomTabControl.

27.1.22.4.16 TCustomTabControl.TabRect Method

Returns the bounding rectangle for a specified tab.

Pascal

```
function TabRect(Index: Integer): TRect;
```

C++

```
__fastcall TRect TabRect(int Index);
```

Description

Use TabRect to determine the current position of the tab with the specified index. Index is the 0-offset index of a tab in the Tabs (see page 3948) property (TTabControl (see page 4841)) or in the Pages property (TPageControl (see page 4459)). TabRect returns the bounding rectangle (in client coordinates) of the specified tab.

See Also

Tabs (see page 3948)

Pages (see page 4498)

27.1.22.4.17 TCustomTabControl.UpdateTabImages Method

Updates the tab images.

Pascal

```
procedure UpdateTabImages;
```

C++

```
__fastcall UpdateTabImages();
```

Description

UpdateTabImages is called automatically to update the tab images after streaming in the control.

See Also

Images (see page 3941)

27.1.22.5 TCustomTabControl Properties

The properties of the TCustomTabControl class are listed here.

Protected Properties

	Name	Description
R	DisplayRect (see page 3941)	Specifies the bounding rectangle of a tab control client area.
	HotTrack (see page 3941)	Determines whether labels on the tab under the mouse are automatically highlighted.
	Images (see page 3941)	Specifies the images drawn in tabs.
	MultiLine (see page 3942)	Determines whether the tabs can appear on more than one row.
	MultiSelect (see page 3943)	Specifies whether multiple tabs can be selected.
	OwnerDraw (see page 3944)	Specifies whether the tab control handles its own painting.
	RaggedRight (see page 3944)	Specifies whether rows of tabs stretch to fill the width of the control.
	ScrollOpposite (see page 3944)	Determines how the rows of tabs are scrolled in a multi-line tab control.
	Style (see page 3945)	Specifies the style of the tab control.
	TabHeight (see page 3945)	Specifies the height, in pixels, of the tabs in the tab control.

	TabIndex (see page 3946)	Identifies the selected tab on a tab control.
	TabPosition (see page 3948)	Determines whether tabs appear at the top or bottom.
	Tabs (see page 3948)	Contains the list of text strings that label the tabs of the tab control.
	TabWidth (see page 3950)	Specifies the horizontal size, in pixels, of the tabs in the tab control.

Public Properties

	Name	Description
R	Canvas (see page 3940)	Gives access to the tab control's canvas.
	TabStop (see page 3950)	Determines if the user can tab to a control.

27.1.22.5.1 TCustomTabControl.Canvas Property

Gives access to the tab control's canvas.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use Canvas to paint to the tab control's canvas during the OnDrawTab (see page 3956) event.

C++ Examples:

```
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TPageControl *myPageControl;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TPageControl")
  {
    myPageControl = (TPageControl *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myPageControl->Canvas->MoveTo(myPageControl->ClientRect.Left,
myPageControl->ClientRect.Top);
    myPageControl->Canvas->LineTo(myPageControl->ClientRect.Right,
myPageControl->ClientRect.Bottom);
    myPageControl = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TPageControl) then
  begin
    myPageControl:= (ListBox1.Items.Objects[Index] as TPageControl);
    with myPageControl.ClientRect do
```

```
    begin // make sure we are using the TControl's canvas and not the form's!
      myPageControl.Canvas.MoveTo(Left, Top);
      myPageControl.Canvas.LineTo(Right, Bottom);
    end;
  myPageControl:= nil;
  end;
```

See Also

OwnerDraw (see page 3944)

OnDrawTab (see page 3956)

27.1.22.5.2 TCustomTabControl.DisplayRect Property

Specifies the bounding rectangle of a tab control client area.

Pascal

```
property DisplayRect: TRect;
```

C++

```
__property TRect DisplayRect;
```

Description

Use DisplayRect to obtain the coordinates, in pixels, of the portion of the tab control that represents the contents of the tab pages. The appearance of this portion of the tab control does not change when different tabs are selected.

Use DisplayRect rather than the ClientRect property to determine the portion of the tab control that can contain other objects.

See Also

ClientRect

AdjustClientRect (see page 3933)

27.1.22.5.3 TCustomTabControl.HotTrack Property

Determines whether labels on the tab under the mouse are automatically highlighted.

Pascal

```
property HotTrack: Boolean;
```

C++

```
__property Boolean HotTrack;
```

Description

Set HotTrack to true to provide visual feedback to the user about which tab would be selected if the mouse button is clicked. HotTrack is especially useful when the tab control contains multiple rows of tabs.

See Also

MultiLine (see page 3942)

HotImages

27.1.22.5.4 TCustomTabControl.Images Property

Specifies the images drawn in tabs.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Images specifies the images that are drawn in tabs. If DisabledImages is defined, those images are used instead for disabled tabs; otherwise, grayed versions of the images are used.

In the implementation of TCustomTabControl (see page 3921), the order of images in image list properties directly corresponds to the associated tabs in the control. In the descendant class, TPageControl (see page 4459), this behavior is replaced by support for individual TTabSheet (see page 4919) objects, each with an ImageIndex property. In any case, the application programmer can define customized image displays by defining an OnGetImageIndex (see page 3956) event handler.

See Also

OnGetImageIndex (see page 3956)

Pages (see page 4498)

ImageIndex (see page 4943)

DisabledImages

27.1.22.5.5 TCustomTabControl.MultiLine Property

Determines whether the tabs can appear on more than one row.

Pascal

```
property MultiLine: Boolean;
```

C++

```
__property Boolean MultiLine;
```

Description

Use MultiLine to determine how the tabs are displayed. If MultiLine is true, the tabs are displayed on more than one row when the number of tabs exceeds the number that fits across the top of the tab control. How many rows is determined by how many tabs are in the tab control. If MultiLine is false, the tabs are displayed on one row only, and the user must scroll the displayed scroll arrows to view all the tabs.

C++ Examples:

```
/*
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
*/

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->MultiLine = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  CheckBox1->Caption = "Stack Tabs?";
  for (int i = 0; i < 20; i++)
  {
    TTabSheet *pPage = new TTabSheet(PageControl1);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
  }
}
```

Delphi Examples:

```
{
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.MultiLine := CheckBox1.Checked;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Checkbox1.Caption := 'Stack Tabs?';
  for i := 0 to 20 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'Tab #' + IntToStr(i);
    end;
end;
```

See Also

ScrollOpposite (see page 3944)

27.1.22.5.6 TCustomTabControl.MultiSelect Property

Specifies whether multiple tabs can be selected.

Pascal

```
property MultiSelect: Boolean;
```

C++

```
__property Boolean MultiSelect;
```

Description

Use MultiSelect to get or set whether multiple tabs can be selected.

MultiSelect can only be true if Style (see page 3945) is tsFlatButtons or tsButtons.

See Also

ScrollOpposite (see page 3944)

27.1.22.5.7 TCustomTabControl.OwnerDraw Property

Specifies whether the tab control handles its own painting.

Pascal

```
property OwnerDraw: Boolean;
```

C++

```
__property Boolean OwnerDraw;
```

Description

Set OwnerDraw to true to override any default painting behavior.

See Also

Canvas (see page 3940)

OnDrawTab (see page 3956)

27.1.22.5.8 TCustomTabControl.RaggedRight Property

Specifies whether rows of tabs stretch to fill the width of the control.

Pascal

```
property RaggedRight: Boolean;
```

C++

```
__property Boolean RaggedRight;
```

Description

Use RaggedRight to get or set whether rows of tabs stretch to fill the width of the control.

27.1.22.5.9 TCustomTabControl.ScrollOpposite Property

Determines how the rows of tabs are scrolled in a multi-line tab control.

Pascal

```
property ScrollOpposite: Boolean;
```

C++

```
__property Boolean ScrollOpposite;
```

Description

Set ScrollOpposite to determine where previous rows of tabs in a multi-line tab control move when the user selects a tab in another row. When ScrollOpposite is true, previous rows of tabs are moved to the bottom of the tab control if TabPosition (see page 3948) is tpTop, or to the top if TabPosition (see page 3948) is tpBottom. When ScrollOpposite is false, previous rows of tabs are moved to the back of all other rows of tabs.

For example, in a tab control with three rows of tabs at the top, if the user selects a tab in the second row, ScrollOpposite determines where the first row moves. If ScrollOpposite is true, the second row now appears at the front (bottom) row on top of the tab control, followed by the third row behind (above) it. The first row moves to the bottom of the tab control. If ScrollOpposite is false, the first row moves to the back (top), so that the tab control displays the second row in front (at the bottom of the tab region), followed by the third row (now in the middle), followed by the first row.

Note: Setting ScrollOpposite to true automatically sets the MultiLine (see page 3942) property to true.

See Also

MultiLine (see page 3942)

TabPosition (see page 3948)

27.1.22.5.10 TCustomTabControl.Style Property

Specifies the style of the tab control.

Pascal

```
property Style: TTabStyle;
```

C++

```
__property TTabStyle Style;
```

Description

Use Style to determine the appearance of the tabs. These can appear as notebook tabs or as buttons.

27.1.22.5.11 TCustomTabControl.TabHeight Property

Specifies the height, in pixels, of the tabs in the tab control.

Pascal

```
property TabHeight: Smallint;
```

C++

```
__property Smallint TabHeight;
```

Description

Set TabHeight to control the height of the tabs that appear above the client region of the tab control. The TabHeight property is the vertical size in pixels of the individual tabs. If TabHeight is set to 0, the tabs automatically size themselves to fit their text.

C++ Examples:

```
/*
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
     AnsiString s = PageControl1->Pages[i]->Caption;
     if (Canvas->TextWidth(s) * 2 > PageControl1->TabWidth)
        PageControl1->TabWidth = Canvas->TextWidth(s) * 2;
      if (Canvas->TextHeight(s) * 2 > PageControl1->TabHeight)
        PageControl1->TabHeight = Canvas->TextHeight(s) * 2;
  }
}
```

Delphi Examples:

```
{
This project requires a TPageControl and several TTabPages.
```

```
  To add new pages, right-click the TPageControl and select
  New Page.  During the form creation the width and height of
  all tabs on the Page Control becomes double the largest
  previous value. So if there are three tabs with captions that
  are 12, 23, and 19 characters long, the TabWidth would be the
  number of pixels required to fit 46 characters.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    with PageControl1 do
    begin
      for i := 0 to PageCount - 1 do
      begin
        if (Canvas.TextWidth(Pages[i].Caption) * 2) > TabWidth then
          TabWidth := Canvas.TextWidth(Pages[i].Caption) * 2;
        if (Canvas.TextHeight(Pages[i].Caption) * 2) > TabHeight then
          TabHeight := Canvas.TextHeight(Pages[i].Caption) * 2;
      end;
    end;
  end;
```

See Also

DisplayRect (see page 3941)

Tabs (see page 3948)

TabWidth (see page 3950)

27.1.22.5.12 TCustomTabControl.TabIndex Property

Identifies the selected tab on a tab control.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Read TabIndex to determine which tab was selected by the user. TabIndex is the index of the tab in the list of labels maintained by the Tabs (see page 3948) property. The first (leftmost) tab has a TabIndex of 0, the next has 1, and so on. If no tabs are selected, TabIndex has a value of -1.

Set TabIndex to programmatically change the selected tab in the tab control.

C++ Examples:

```
/*
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
*/
void __fastcall TForm1::Add_a_fileClick(TObject *Sender)
{
```

```
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    int index = TabControl1->Tabs->Add(OpenDialog1->FileName);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[index]);
  }
}

void __fastcall TForm1::Assign_filesClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TabControl1->Tabs->Assign(OpenDialog1->Files);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[TabControl1->TabIndex]);
  }
}
/*
Place the following code in the tab control's OnChange event
handler:
*/
void __fastcall TForm1::TabControl1Change(TObject *Sender)
{
    Memo1->Lines->LoadFromFile(
      TabControl1->Tabs->Strings[TabControl1->TabIndex]);
}
```

Delphi Examples:

```
{
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
}
procedure TForm1.Add_a_fileClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    index:= TabControl1.Tabs.Add(OpenDialog1.FileName);
    Memo1.Lines.LoadFromFile(TabControl1.Tabs[index]);
  end;
end;

procedure TForm1.Assign_filesClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    TabControl1.Tabs.Assign(OpenDialog1.Files);
    Memo1.Lines.LoadFromFile(
      TabControl1.Tabs[TabControl1.TabIndex]);
  end;
end;

{
```

```
Place the following code in the tab control's OnChange event
handler:
}
procedure TForm1.TabControl1Change(Sender: TObject);
begin
  with TabControl1 do
    Memo1.Lines.LoadFromFile(Tabs[TabIndex]);
end;
```

See Also

Tabs (see page 3948)

27.1.22.5.13 TCustomTabControl.TabPosition Property

Determines whether tabs appear at the top or bottom.

Pascal

```
property TabPosition: TTabPosition;
```

C++

```
__property TTabPosition TabPosition;
```

Description

Set TabPosition to tpTop to display the tabs at the top of the tab control. Set TabPosition to tpBottom to display the tabs at the bottom. If the ScrollOpposite (see page 3944) property is true, tabs can appear at both top and bottom, with the row containing the current tab and all subsequent rows on the side indicated by TabPosition.

Due to a limitation in the windows common control, if TabPosition is tpLeft or tpRight, Font must be a True Type font. Otherwise, tab text may paint incorrectly.

See Also

ScrollOpposite (see page 3944)

27.1.22.5.14 TCustomTabControl.Tabs Property

Contains the list of text strings that label the tabs of the tab control.

Pascal

```
property Tabs: TStrings;
```

C++

```
__property TStrings Tabs;
```

Description

Tabs contains a list of strings that label the tabs in the tab control. Tabs can also contain objects associated with the labels, using the TStrings Objects property. Add new strings to the Tabs property to add new tabs to the tab control. Rearrange the list to rearrange the tabs.

C++ Examples:

```
/*
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
```

```
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
*/
void __fastcall TForm1::Add_a_fileClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    int index = TabControl1->Tabs->Add(OpenDialog1->FileName);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[index]);
  }
}

void __fastcall TForm1::Assign_filesClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TabControl1->Tabs->Assign(OpenDialog1->Files);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[TabControl1->TabIndex]);
  }
}
/*
Place the following code in the tab control's OnChange event
handler:
*/
void __fastcall TForm1::TabControl1Change(TObject *Sender)
{
    Memo1->Lines->LoadFromFile(
      TabControl1->Tabs->Strings[TabControl1->TabIndex]);
}
```

Delphi Examples:

```
{
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
}
procedure TForm1.Add_a_fileClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    index:= TabControl1.Tabs.Add(OpenDialog1.FileName);
    Memo1.Lines.LoadFromFile(TabControl1.Tabs[index]);
  end;
end;

procedure TForm1.Assign_filesClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    TabControl1.Tabs.Assign(OpenDialog1.Files);
```

```
      Memo1.Lines.LoadFromFile(
        TabControl1.Tabs[TabControl1.TabIndex]);
    end;
  end;

  {
  Place the following code in the tab control's OnChange event
  handler:
  }
  procedure TForm1.TabControl1Change(Sender: TObject);
  begin
    with TabControl1 do
      Memo1.Lines.LoadFromFile(Tabs[TabIndex]);
  end;
```

See Also

TabIndex (see page 3946)

Objects (see page 3077)

TTabControl (see page 4841)

27.1.22.5.15 TCustomTabControl.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

27.1.22.5.16 TCustomTabControl.TabWidth Property

Specifies the horizontal size, in pixels, of the tabs in the tab control.

Pascal

```
property TabWidth: Smallint;
```

C++

```
__property Smallint TabWidth;
```

Description

TabWidth specifies a fixed width for all tabs. If TabWidth is 0, then each tab is automatically made wide enough to show its caption.

C++ Examples:

```
/*
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
     AnsiString s = PageControl1->Pages[i]->Caption;
     if (Canvas->TextWidth(s) * 2 > PageControl1->TabWidth)
        PageControl1->TabWidth = Canvas->TextWidth(s) * 2;
      if (Canvas->TextHeight(s) * 2 > PageControl1->TabHeight)
        PageControl1->TabHeight = Canvas->TextHeight(s) * 2;
  }
}
```

Delphi Examples:

```
{
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  with PageControl1 do
  begin
    for i := 0 to PageCount - 1 do
    begin
      if (Canvas.TextWidth(Pages[i].Caption) * 2) > TabWidth then
        TabWidth := Canvas.TextWidth(Pages[i].Caption) * 2;
      if (Canvas.TextHeight(Pages[i].Caption) * 2) > TabHeight then
        TabHeight := Canvas.TextHeight(Pages[i].Caption) * 2;
    end;
  end;
end;
```

See Also

TabHeight (see page 3945)

Tabs (see page 3948)

27.1.22.6 TCustomTabControl Events

The events of the TCustomTabControl class are listed here.

Protected Events

	Name	Description
	OnChange (see page 3952)	Occurs after a new tab is selected.
	OnChanging (see page 3955)	Occurs immediately before a new tab is selected.
	OnDrawTab (see page 3956)	Occurs when a tab is about to be drawn.

	OnGetImageIndex (see page 3956)	Occurs when a tab is about to display its associated image.

27.1.22.6.1 TCustomTabControl.OnChange Event

Occurs after a new tab is selected.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action immediately after the selected tab changes. Use the TabIndex (see page 3946) property to determine which tab is now selected. This is the opportunity to make any changes to the control that reflect the new state implied by the selected tag.

Before the value of TabIndex (see page 3946) changes, an OnChanging (see page 3955) event occurs.

C++ Examples:

```
/*
This project requires a Page Control.  You will need to
populate the Page Control by right clicking and selecting
New Page from the context menu. As the tabs are selected by
the user, the Page Control OnChange event fires, which
causes the form caption to display the captions of each of
the TTabSheets attached to the Page Control.
Note the use of the Form's OnShow event handler to set the
caption when the form first appears.  This occurs before
any changes trigger the OnChange event of the Page Control.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  Caption = AnsiString("Now working on tab: " + PageControl1->ActivePage->Caption);
}

void __fastcall TForm1::FormShow(TObject *Sender)
{
  PageControl1Change(Sender);
}
/*
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
*/
void __fastcall TForm1::Add_a_fileClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    int index = TabControl1->Tabs->Add(OpenDialog1->FileName);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[index]);
  }
}
```

```
void __fastcall TForm1::Assign_filesClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TabControl1->Tabs->Assign(OpenDialog1->Files);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[TabControl1->TabIndex]);
  }
}
/*
Place the following code in the tab control's OnChange event
handler:
*/
void __fastcall TForm1::TabControl1Change(TObject *Sender)
{
    Memo1->Lines->LoadFromFile(
      TabControl1->Tabs->Strings[TabControl1->TabIndex]);
}
/*
This example requires a new TPageControl, with no new pages
created at design time.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
Page Control's OnChange event handler displays a message
dialog when the user changes tabs.  The message dialog
contains the captions for the tabs immediately before and
after the active tab.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(PageControl1);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    pPage->Name = AnsiString("ts") + pPage->Caption;
  }
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  AnsiString PrevCaption, NextCaption;
  TPageControl *pPC = (TPageControl *)Sender;
  PrevCaption = pPC->FindNextPage(pPC->ActivePage, false, false)->Caption;
  NextCaption = pPC->FindNextPage(pPC->ActivePage, true, false)->Caption;
  ShowMessage(AnsiString("Previous tab caption: '") + PrevCaption + AnsiString("'  Next tab
Caption: '") + NextCaption + AnsiString("'"));
}
```

Delphi Examples:

```
{
This project requires a Page Control.  You will need to
populate the Page Control by right clicking and selecting
New Page from the context menu. As the tabs are selected by
the user, the Page Control OnChange event fires, which
causes the form caption to display the captions of each of
the TTabSheets attached to the Page Control.
Note the use of the Form's OnShow event handler to set the
caption when the form first appears.  This occurs before
any changes trigger the OnChange event of the Page Control.
}
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  Caption := ' Now working on tab:  ' + PageControl1.ActivePage.Caption;
end;

procedure TForm1.FormShow(Sender: TObject);
```

```
begin
  PageControl1Change(Sender);
end;
{
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
}
procedure TForm1.Add_a_fileClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    index:= TabControl1.Tabs.Add(OpenDialog1.FileName);
    Memo1.Lines.LoadFromFile(TabControl1.Tabs[index]);
  end;
end;

procedure TForm1.Assign_filesClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    TabControl1.Tabs.Assign(OpenDialog1.Files);
    Memo1.Lines.LoadFromFile(
      TabControl1.Tabs[TabControl1.TabIndex]);
  end;
end;

{
Place the following code in the tab control’s OnChange event
handler:
}
procedure TForm1.TabControl1Change(Sender: TObject);
begin
  with TabControl1 do
    Memo1.Lines.LoadFromFile(Tabs[TabIndex]);
end;
{
This example requires a new TPageControl, with no new pages
created at design time.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
Page Control’s OnChange event handler displays a message
dialog when the user changes tabs.  The message dialog
contains the captions for the tabs immediately before and
after the active tab.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
```

```
      Caption := 'TabSheet #' + IntToStr(i);
    end;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
var
  PrevCaption, NextCaption: ShortString;
begin
  with (Sender as TPageControl) do
  begin
    PrevCaption :=
      FindNextPage(ActivePage, False, False).Caption;
    NextCaption :=
      FindNextPage(ActivePage, True, False).Caption;
  end;
  ShowMessage('Previous tab caption: "' + PrevCaption +
    '"    Next tab Caption: "' + NextCaption + '"');
end;
```

See Also

OnChanging (see page 3955)

TabIndex (see page 3946)

27.1.22.6.2 TCustomTabControl.OnChanging Event

Occurs immediately before a new tab is selected.

Pascal

```
property OnChanging: TTabChangingEvent;
```

C++

```
__property TTabChangingEvent OnChanging;
```

Description

Write an OnChanging event handler to take specific action immediately before the selected tab changes. Set the AllowChange parameter to false to prevent the change.

Use an OnChanging event handler to prevent the user from leaving a tab setting until certain conditions have been met. An OnChanging event handler can also be used to save information about the current state of the tab control before it is changed by a new tab selection.

Note: The event is not called if you change the active page in code, for example, by setting the value of TPageControl (see page 4459). ActivePage.

C++ Examples:

```
/*
This project requires a TPageControl, several TTabSheets (to
add sheets, right click the Page Control and select "New
Page" from the context menu), and a TEdit control on the
first tab sheet.  The user is not allowed to change to a new
page if the active page is "TabSheet1" and there is no text
in the Edit Control.
*/
void __fastcall TForm1::PageControl1Changing(TObject *Sender, bool &AllowChange)
{
  if (((TPageControl *)Sender)->ActivePage == TabSheet1)
    AllowChange = (!Edit1->Text.IsEmpty());
  else
```

```
    AllowChange = true;
  if (AllowChange == false)
  {
    MessageBeep(0);
    Edit1->SetFocus();
  }
}
```

Delphi Examples:

```
{
This project requires a TPageControl, several TTabSheets (to
add sheets, right click the Page Control and select "New
Page" from the context menu), and a TEdit control on the
first tab sheet.  The user is not allowed to change to a new
page if the active page is "TabSheet1" and there is no text
in the Edit Control.
}
procedure TForm1.PageControl1Changing(Sender: TObject; var AllowChange: Boolean);
begin
  if ((Sender as TPageControl).ActivePage = TabSheet1) then
    AllowChange := (Edit1.Text <> '')
  else
    AllowChange := True;
end;
```

See Also

OnChange (see page 3952)

TabIndex (see page 3946)

27.1.22.6.3 TCustomTabControl.OnDrawTab Event

Occurs when a tab is about to be drawn.

Pascal

```
property OnDrawTab: TDrawTabEvent;
```

C++

```
__property TDrawTabEvent OnDrawTab;
```

Description

Use OnDrawTab to customize the painting of tabs.

You can paint the tab using the Canvas (see page 3940) property.

See Also

Canvas (see page 3940)

OwnerDraw (see page 3944)

27.1.22.6.4 TCustomTabControl.OnGetImageIndex Event

Occurs when a tab is about to display its associated image.

Pascal

```
property OnGetImageIndex: TTabGetImageEvent;
```

C++

```
__property TTabGetImageEvent OnGetImageIndex;
```

Description

Write an OnGetImageIndex event handler to specify the ImageIndex for a given TabIndex (see page 3946).

On entry to the event handler, the ImageIndex parameter is the same as the TabIndex (see page 3946) parameter, where TabIndex (see page 3946) is the index of the tab which is about to display its image. Change (see page 3934) the ImageIndex parameter to specify the index into the Images (see page 3941) property that specifies the image that should be painted.

See Also

Images (see page 3941)

TabIndex (see page 3946)

27.1.23 TCustomTreeView Class

TCustomTreeView is the base type for tree-view components such as TTreeView (see page 5293).

Class Hierarchy

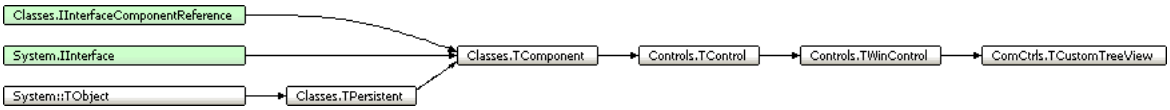

Pascal

```
TCustomTreeView = class(TWinControl);
```

C++

```
class TCustomTreeView : public TWinControl;
```

File

ComCtrls

Description

Use TCustomTreeView as a base class when defining controls that present information in a hierarchical structure with nodes that can be expanded or collapsed. Most of the properties defined in TCustomTreeView are protected so that you can choose whether to publish them in your own tree view components.

Do not create instances of TCustomTreeView. Use the generic descendant TTreeView (see page 5293), or create your own descendant class.

See Also

TTreeNode (see page 5230)

TTreeNodes (see page 5266)

TTreeView (see page 5293)

27.1.23.1 TCustomTreeView Members

The following tables list the members exposed by TCustomTreeView.

Protected Events

	Name	Description
	OnAddition (see page 4016)	Occurs when new node is added.
	OnAdvancedCustomDraw (see page 4016)	Occurs at discrete stages during the painting of the tree view control.
	OnAdvancedCustomDrawItem (see page 4016)	Occurs at discrete stages during the painting of tree view nodes.
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCancelEdit (see page 4017)	Occurs when user cancels editing of a node's Caption.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnChange (see page 4017)	Occurs whenever the selection has changed from one node to another.
	OnChanging (see page 4018)	Occurs when the selection is about to change from one node to another.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnCollapsed (see page 4018)	Occurs after a node has been collapsed.
	OnCollapsing (see page 4018)	Occurs when a node is about to be collapsed.
	OnCompare (see page 4019)	Occurs when two nodes must be compared during a sort of the nodes in the tree view.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreateNodeClass (see page 4019)	Occurs when a node in the treeview is about to be created.
	OnCustomDraw (see page 4019)	Occurs immediately prior to painting the tree view control.
	OnCustomDrawItem (see page 4021)	Occurs immediately prior to painting a node in a tree view control.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeletion (see page 4024)	Occurs when a node in the tree view is deleted.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEdited (see page 4024)	Occurs after the user edits the Text property of a node.
	OnEditing (see page 4024)	Occurs when the user starts to edit the Text property of a node.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnExpanded (see page 4025)	Occurs after a node is expanded.
	OnExpanding (see page 4025)	Occurs when a node is about to be expanded.
	OnGetImageIndex (see page 4026)	Occurs when the tree view looks up the ImageIndex of a node.
	OnGetSelectedIndex (see page 4026)	Occurs when the tree view looks up the SelectedIndex of a node.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.

	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FChangeTimer (see page 3970)	This is FChangeTimer, a member of class TCustomTreeView.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	Added (see page 3972)	This is Added, a member of class TCustomTreeView.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanChange (see page 3972)	Pending node change event dispatcher.
	CanCollapse (see page 3973)	Pending node collapse event dispatcher.
	CanEdit (see page 3973)	Node editing event dispatcher.
	CanExpand (see page 3974)	Node expand event dispatcher.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.

	Change (see page 3974)	Completed node change event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	Collapse (see page 3975)	Completed node collapse event dispatcher.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateNode (see page 3975)	Creates a new TTreeNode (see page 5230) instance with the TTreeNodes (see page 5266) specified by Items (see page 4004) as its Owner.
V	CreateNodes (see page 3976)	This is CreateNodes, a member of class TCustomTreeView.
V	CreateParams (see page 3976)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 3976)	Creates the window used by the tree view component.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
V	CustomDraw (see page 3977)	Generates an OnCustomDraw (see page 4019) or OnAdvancedCustomDraw (see page 4016) event.
V	CustomDrawItem (see page 3977)	Generates an OnCustomDrawItem (see page 4021) or OnAdvancedCustomDrawItem (see page 4016) event.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	Delete (see page 3979)	Generates an OnDeletion (see page 4024) event.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 3980)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
V	DoEndDrag (see page 3981)	End drag event dispatcher.
V	DoEnter (see page 3981)	Respond to receiving input focus.
V	DoExit (see page 3981)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.

	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 3982)	Drag start event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	Edit (see page 3982)	Edited event dispatcher.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	Expand (see page 3983)	Completed node expansion event dispatcher.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 3984)	Returns the image list to be displayed during drag operations.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetImageIndex (see page 3986)	Obtain image index.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelectedIndex (see page 3988)	Obtain selection index.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	InvalidateSelectionsRects (see page 3989)	This is InvalidateSelectionsRects, a member of class TCustomTreeView.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
V	IsCustomDrawn (see page 3989)	Specifies when the control should generate custom draw events.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 3990)	Initializes the tree view after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 3991)	OnMouseDown event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 3992)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 3994)	Sets the drag mode for the tree view.

	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateSelection (see page 3995)	This is ValidateSelection, a member of class TCustomTreeView.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 3995)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoExpand (see page 3996)	Specifies whether the nodes in the tree view automatically expand and collapse depending on the selection.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 3997)	Determines whether the tree view control has a border.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.

	ChangeDelay (see page 3999)	Specifies the delay between when a node is selected and when the OnChange (see page 4017) event occurs.
	Color (see page 5812)	Specifies the background color of the control.
	CreateWndRestores (see page 4000)	This is CreateWndRestores, a member of class TCustomTreeView.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	HideSelection (see page 4000)	Determines whether a selected node appears selected when the focus shifts to another control.
	HotTrack (see page 4001)	Specifies whether list items are highlighted when the mouse passes over them.
	Images (see page 4001)	Determines which image list is associated with the tree view.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Indent (see page 4001)	Specifies the amount of indentation in pixels when a list of child nodes is expanded.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	Items (see page 4004)	Lists the individual nodes that appear in the tree view control.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	MultiSelect (see page 4005)	Determines whether the user can select more than one tree node at a time.
	MultiSelectStyle (see page 4005)	Determines how multiple node selections work.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 4006)	Determines whether the user can edit the node labels.
	RightClickSelect (see page 4006)	Determines whether the Selected (see page 4007) property returns nodes that are selected using the right mouse button.
	RowSelect (see page 4007)	Specifies whether the entire row of the selected item is highlighted.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ShowButtons (see page 4013)	Specifies whether to display plus (+) and minus (-) buttons to the left side of each parent item.
	ShowLines (see page 4013)	Specifies whether to display the lines that link child nodes to their corresponding parent nodes.
	ShowRoot (see page 4013)	Specifies whether lines connecting top-level nodes are displayed.
	SortType (see page 4014)	Determines if and how the nodes in a tree view are automatically sorted.
	StateImages (see page 4014)	Determines which image list to use for state images.
	Text (see page 5842)	Contains a text string associated with the control.
	ToolTips (see page 4014)	Specifies whether the items in the tree view have tool tips.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AlphaSort (see page 3972)	Sort tree nodes.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	ClearSelection (see page 3974)	Deselect (see page 3980) all selected nodes, optionally excluding primary selection.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 3969)	Creates an instance of a TTreeView (see page 5293) control.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CustomSort (see page 3978)	Sorts the nodes in the tree view into a customized sort order.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Deselect (see page 3980)	Deselects a specified node.
V	Destroy (see page 3970)	Destroys the tree view control and is called by the Delete (see page 3979) method.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
V	FindNextToSelect (see page 3983)	Returns next selectable node.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	FullCollapse (see page 3983)	Collapses all the nodes within a tree view control.
	FullExpand (see page 3984)	Expands all nodes within the tree view control.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHitTestInfoAt (see page 3985)	Returns information about the location of a point relative to the client area of the tree view control.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNodeAt (see page 3987)	Returns the node that is found at the specified position.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetSelections (see page 3989)	Obtain selections list.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsEditing (see page 3990)	Indicates whether a node is currently being edited by the user.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	LoadFromFile (see page 3990)	Reads the file specified in FileName and loads the data into the tree view.
	LoadFromStream (see page 3991)	Reads tree view data from a stream and stores the contents in the tree view.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 3992)	Saves a the tree view to the file specified in FileName.
	SaveToStream (see page 3993)	Writes the data in the tree view to the stream passed as the Stream parameter.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
V	Select (see page 3993)	This is the overview for the Select method overload.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Subselect (see page 3994)	Toggle node selection.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.

	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 3997)	Provides access to the canvas.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropTarget (see page 4000)	Specifies which item in the tree view appears as the target of a drag and drop operation.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	Selected (see page 4007)	Specifies the selected node in the tree view.
R	SelectionCount (see page 4012)	Returns number of nodes selected.
R	Selections (see page 4012)	Returns nodes selected.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.

R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TopItem (see page 4015)	Specifies the topmost node that appears in the tree view.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.23.2 TCustomTreeView.Create Constructor

Creates an instance of a TTreeView (see page 5293) control.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomTreeView(TComponent * AOwner);
```

Description

Call Create to instantiate a tree view at runtime. Tree view controls placed on forms at design time are created automatically. After calling the inherited constructor, Create initializes many properties the control, including setting BorderStyle (see page 3997) to bsSingle and setting ShowButtons (see page 4013), ShowRoot (see page 4013), and ShowLines (see page 4013), and HideSelection (see page 4000) to true.

See Also

BorderStyle (see page 3997)

HideSelection (see page 4000)

ShowButtons (see page 4013)

ShowLines (see page 4013)

ShowRoot (see page 4013)

27.1.23.3 TCustomTreeView.Destroy Destructor

Destroys the tree view control and is called by the Delete (see page 3979) method.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomTreeView();
```

Description

Do not call the destructor directly in an application. Instead, use the Free method. Free checks that the tree view is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

27.1.23.4 TCustomTreeView Fields

The fields of the TCustomTreeView class are listed here.

Protected Fields

	Name	Description
	FChangeTimer (see page 3970)	This is FChangeTimer, a member of class TCustomTreeView.

27.1.23.4.1 TCustomTreeView.FChangeTimer Field

Pascal

```
FChangeTimer: TTimer;
```

C++

```
TTimer FChangeTimer;
```

Description

This is FChangeTimer, a member of class TCustomTreeView.

27.1.23.5 TCustomTreeView Methods

The methods of the TCustomTreeView class are listed here.

Protected Methods

	Name	Description
	Added (see page 3972)	This is Added, a member of class TCustomTreeView.
	CanChange (see page 3972)	Pending node change event dispatcher.
	CanCollapse (see page 3973)	Pending node collapse event dispatcher.
	CanEdit (see page 3973)	Node editing event dispatcher.
	CanExpand (see page 3974)	Node expand event dispatcher.

	Change (see page 3974)	Completed node change event dispatcher.
	Collapse (see page 3975)	Completed node collapse event dispatcher.
V	CreateNode (see page 3975)	Creates a new TTreeNode (see page 5230) instance with the TTreeNodes (see page 5266) specified by Items (see page 4004) as its Owner.
V	CreateNodes (see page 3976)	This is CreateNodes, a member of class TCustomTreeView.
V	CreateParams (see page 3976)	Initializes a window-creation parameter data structure.
V	CreateWnd (see page 3976)	Creates the window used by the tree view component.
V	CustomDraw (see page 3977)	Generates an OnCustomDraw (see page 4019) or OnAdvancedCustomDraw (see page 4016) event.
V	CustomDrawItem (see page 3977)	Generates an OnCustomDrawItem (see page 4021) or OnAdvancedCustomDrawItem (see page 4016) event.
	Delete (see page 3979)	Generates an OnDeletion (see page 4024) event.
V	DestroyWnd (see page 3980)	Destroys the control's window.
V	DoEndDrag (see page 3981)	End drag event dispatcher.
V	DoEnter (see page 3981)	Respond to receiving input focus.
V	DoExit (see page 3981)	Respond to losing input focus
V	DoStartDrag (see page 3982)	Drag start event dispatcher.
	Edit (see page 3982)	Edited event dispatcher.
	Expand (see page 3983)	Completed node expansion event dispatcher.
V	GetDragImages (see page 3984)	Returns the image list to be displayed during drag operations.
V	GetImageIndex (see page 3986)	Obtain image index.
V	GetSelectedIndex (see page 3988)	Obtain selection index.
	InvalidateSelectionsRects (see page 3989)	This is InvalidateSelectionsRects, a member of class TCustomTreeView.
V	IsCustomDrawn (see page 3989)	Specifies when the control should generate custom draw events.
V	Loaded (see page 3990)	Initializes the tree view after it is loaded from a stream.
V	MouseDown (see page 3991)	OnMouseDown event dispatcher.
V	Notification (see page 3992)	Responds to notifications that components are being created or destroyed.
V	SetDragMode (see page 3994)	Sets the drag mode for the tree view.
	ValidateSelection (see page 3995)	This is ValidateSelection, a member of class TCustomTreeView.
V	WndProc (see page 3995)	Provides specific message responses for the control.

Public Methods

	Name	Description
	AlphaSort (see page 3972)	Sort tree nodes.
V	ClearSelection (see page 3974)	Deselect (see page 3980) all selected nodes, optionally excluding primary selection.
	CustomSort (see page 3978)	Sorts the nodes in the tree view into a customized sort order.
V	Deselect (see page 3980)	Deselects a specified node.
V	FindNextToSelect (see page 3983)	Returns next selectable node.
	FullCollapse (see page 3983)	Collapses all the nodes within a tree view control.
	FullExpand (see page 3984)	Expands all nodes within the tree view control.
	GetHitTestInfoAt (see page 3985)	Returns information about the location of a point relative to the client area of the tree view control.
	GetNodeAt (see page 3987)	Returns the node that is found at the specified position.
	GetSelections (see page 3989)	Obtain selections list.
	IsEditing (see page 3990)	Indicates whether a node is currently being edited by the user.
	LoadFromFile (see page 3990)	Reads the file specified in FileName and loads the data into the tree view.
	LoadFromStream (see page 3991)	Reads tree view data from a stream and stores the contents in the tree view.
	SaveToFile (see page 3992)	Saves a the tree view to the file specified in FileName.
	SaveToStream (see page 3993)	Writes the data in the tree view to the stream passed as the Stream parameter.
V	Select (see page 3993)	This is the overview for the Select method overload.
V	Subselect (see page 3994)	Toggle node selection.

27.1.23.5.1 TCustomTreeView.Added Method

Pascal

```
procedure Added(Node: TTreeNode); dynamic;
```

C++

```
__fastcall Added(TTreeNode Node);
```

Description

This is Added, a member of class TCustomTreeView.

27.1.23.5.2 TCustomTreeView.AlphaSort Method

Sort tree nodes.

Pascal

```
function AlphaSort(ARecurse: Boolean = True): Boolean;
```

C++

```
__fastcall Boolean AlphaSort(Boolean ARecurse = True);
```

Description

AlphaSort triggers node sorting or resorting. If an OnCompare (see page 4019) event handler is defined, that routine determines sort order. If no OnCompare (see page 4019) handler is defined, nodes are sorted by a simple case-sensitive compare of their captions.

The optional ARecurse parameter (default true) specifies that sorting should recursively descend the node tree and sort each subtree in turn.

Calling AlphaSort is equivalent to calling CustomSort (see page 3978) with a nil (Delphi) or NULL (C++) procedure parameter and a data parameter of 0.

Calling AlphaSort has the same effect as calling the same method for the Items (see page 4004) property, except that in TTreeNodes (see page 5266), AlphaSort is not recursive by default.

To sort a subtree, use the AlphaSort method in TTreeNodes (see page 5266).

See Also

CustomSort (see page 3978)

SortType (see page 4014)

AlphaSort (see page 5234)

AlphaSort (see page 5279)

27.1.23.5.3 TCustomTreeView.CanChange Method

Pending node change event dispatcher.

Pascal

```
function CanChange(Node: TTreeNode): Boolean; dynamic;
```

C++

```
__fastcall Boolean CanChange(TTreeNode Node);
```

Description

CanChange is called automatically whenever an attempt is made to change the selected node. The Node parameter indicates the currently selected node. CanChange returns true if selection can move away from the specified node, false otherwise.

If there is no OnChanging (see page 4018) event handler, CanChange returns true.

Override this method to make additional, class-specific responses in addition to the OnChanging (see page 4018) event or to suppress the OnChanging (see page 4018) event.

See Also

OnChanging (see page 4018)

27.1.23.5.4 TCustomTreeView.CanCollapse Method

Pending node collapse event dispatcher.

Pascal

```
function CanCollapse(Node: TTreeNode): Boolean; dynamic;
```

C++

```
__fastcall Boolean CanCollapse(TTreeNode Node);
```

Description

CanCollapse is called automatically whenever an attempt is made to collapse the node specified by the Node parameter. CanCollapse returns true if node can be collapsed, false otherwise.

If there is no OnCollapsing (see page 4018) event handler, CanCollapse returns true.

Override this method to make additional, class-specific responses in addition to the OnCollapsing (see page 4018) event or to suppress the OnCollapsing (see page 4018) event.

See Also

OnCollapsing (see page 4018)

27.1.23.5.5 TCustomTreeView.CanEdit Method

Node editing event dispatcher.

Pascal

```
function CanEdit(Node: TTreeNode): Boolean; dynamic;
```

C++

```
__fastcall Boolean CanEdit(TTreeNode Node);
```

Description

CanEdit is called automatically whenever an attempt is made to edit the node specified by the Node parameter. CanEdit returns true if the user can edit the specified node, false otherwise.

If there is no OnEditing (see page 4024) event handler, CanEdit returns true.

Override this method to make additional, class-specific responses in addition to the OnEditing (see page 4024) event or to suppress the OnEditing (see page 4024) event.

See Also

OnEditing (see page 4024)

27.1.23.5.6 TCustomTreeView.CanExpand Method

Node expand event dispatcher.

Pascal

```
function CanExpand(Node: TTreeNode): Boolean; dynamic;
```

C++

```
__fastcall Boolean CanExpand(TTreeNode Node);
```

Description

CanExpand is called automatically whenever an attempt is made to expand the node specified by the Node parameter. CanExpand returns true if node can be expanded, false otherwise.

If there is no OnExpanding (see page 4025) event handler, CanExpand returns true.

Override this method to make additional, class-specific responses in addition to the OnExpanding (see page 4025) event or to suppress the OnExpanding (see page 4025) event.

See Also

OnExpanding (see page 4025)

27.1.23.5.7 TCustomTreeView.Change Method

Completed node change event dispatcher.

Pascal

```
procedure Change(Node: TTreeNode); dynamic;
```

C++

```
__fastcall Change(TTreeNode Node);
```

Description

Change is called automatically whenever the selected node changes. The Node parameter indicates the newly selected node.

Override this method to make additional, class-specific responses in addition to the OnChange (see page 4017) event or to suppress the OnChange (see page 4017) event.

See Also

OnChanging (see page 4018)

ChangeDelay (see page 3999)

27.1.23.5.8 TCustomTreeView.ClearSelection Method

Deselect (see page 3980) all selected nodes, optionally excluding primary selection.

Pascal

```
procedure ClearSelection(KeepPrimary: Boolean = False); virtual;
```

C++

```
virtual __fastcall ClearSelection(Boolean KeepPrimary = False);
```

Description

ClearSelection deselects all selected nodes. If the KeepPrimary parameter is true, the primary selection (in the first element of

the Selections (see page 4012) property) is not toggled.

See Also

Selections (see page 4012)

27.1.23.5.9 TCustomTreeView.Collapse Method

Completed node collapse event dispatcher.

Pascal

```
procedure Collapse(Node: TTreeNode); dynamic;
```

C++

```
__fastcall Collapse(TTreeNode Node);
```

Description

Collapse is called automatically after the node specified by the Node parameter collapses.

Override this method to make additional, class-specific responses in addition to the OnCollapsed (see page 4018) event or to suppress the OnCollapsed (see page 4018) event.

See Also

OnChanging (see page 4018)

ChangeDelay (see page 3999)

27.1.23.5.10 TCustomTreeView.CreateNode Method

Creates a new TTreeNode (see page 5230) instance with the TTreeNodes (see page 5266) specified by Items (see page 4004) as its Owner.

Pascal

```
function CreateNode: TTreeNode; virtual;
```

C++

```
virtual __fastcall TTreeNode CreateNode();
```

Description

Do not call CreateNode in application code. The TTreeNodes (see page 5266) object specified by the Items (see page 4004) property uses this method in the implementation of its Add, AddFirst, AddChild, AddChildFirst, AddObject, AddObjectFirst, AddChildObject, and AddChildObjectFirst methods. To create a new node for the tree view, use one of these methods instead. These methods ensure that the node is inserted properly and its properties reflect its position in the tree view.

See Also

Create (see page 5232)

Add (see page 5270)

AddFirst (see page 5275)

AddChild (see page 5271)

AddChildFirst (see page 5273)

AddObject (see page 5276)

AddObjectFirst (see page 5279)

AddChildObject (see page 5273)

AddChildObjectFirst (see page 5274)

Items (see page 4004)

27.1.23.5.11 TCustomTreeView.CreateNodes Method

Pascal

```
function CreateNodes: TTreeNodes; virtual;
```

C++

```
virtual __fastcall TTreeNodes CreateNodes();
```

Description

This is CreateNodes, a member of class TCustomTreeView.

27.1.23.5.12 TCustomTreeView.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 3976) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle.

CreateParams fills in the data structure that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

CreateParams sets the value of the Params parameter to reflect the current property settings of the tree view.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 3976)

AddBiDiModeExStyle (see page 6128)

27.1.23.5.13 TCustomTreeView.CreateWnd Method

Creates the window used by the tree view component.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the tree view is first created or when the window must be destroyed and recreated to reflect property changes.

CreateWnd first calls the CreateParams (see page 3976) method to initialize the window-creation parameters, then calls CreateWindowHandle to create the window for the control. If the tree view's window was temporarily destroyed and recreated, CreateWnd reads various property settings from an in-memory cache where they were stored while there was no window. CreateWnd then sends a series of Windows messages to the newly-created tree view control to initialize its property settings.

See Also

CreateParams (see page 3976)

CreateWindowHandle (see page 6137)

Perform (see page 5776)

DestroyWnd (see page 3980)

27.1.23.5.14 TCustomTreeView.CustomDraw Method

Generates an OnCustomDraw (see page 4019) or OnAdvancedCustomDraw (see page 4016) event.

Pascal

```
function CustomDraw(const ARect: TRect; Stage: TCustomDrawStage): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CustomDraw(const TRect ARect, TCustomDrawStage Stage);
```

Description

CustomDraw is called automatically at discrete stages in the painting process. It determines what event handlers, if any, are appropriate for the current drawing stage, and generates the appropriate events.

ARect is the boundaries of the tree view in the coordinates of the Canvas (see page 3997) property.

Stage is the current stage of the painting process.

CustomDraw returns true if the tree view should continue with the default painting process, false if the method has completely rendered the tree view and there is no more painting to be done.

See Also

OnCustomDraw (see page 4019)

Canvas (see page 3997)

OnAdvancedCustomDraw (see page 4016)

27.1.23.5.15 TCustomTreeView.CustomDrawItem Method

Generates an OnCustomDrawItem (see page 4021) or OnAdvancedCustomDrawItem (see page 4016) event.

Pascal

```
function CustomDrawItem(Node: TTreeNode; State: TCustomDrawState; Stage: TCustomDrawStage; var
PaintImages: Boolean): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CustomDrawItem(TTreeNode Node, TCustomDrawState State,
TCustomDrawStage Stage, Boolean PaintImages);
```

Description

CustomDrawItem is called automatically at discrete stages during the painting of tree nodes. It determines what event handlers, if any, are appropriate for the current drawing stage, and generates the appropriate events.

Node is the item that is about to be painted.

State is the item's current state: one or more of cdsSelected, cdsGrayed, cdsDisabled, cdsChecked, cdsFocused, cdsDefault, cdsHot, cdsMarked, cdsIndeterminate.

Stage is the current stage of the painting process.

PaintImages returns a value that indicates whether the tree view should paint the images associated with the current node. This value is only meaningful when Stage is csPrePaint.

CustomDrawItem returns true if the tree view should continue with the default rendering of the item, false if the method completely handled painting the item and there is nothing left to paint.

See Also

OnCustomDrawItem (see page 4021)

Canvas (see page 3997)

27.1.23.5.16 TCustomTreeView.CustomSort Method

Sorts the nodes in the tree view into a customized sort order.

Pascal

```
function CustomSort(SortProc: TTVCompare; Data: Longint; ARecurse: Boolean = True): Boolean;
```

C++

```
__fastcall Boolean CustomSort(TTVCompare SortProc, Longint Data, Boolean ARecurse = True);
```

Description

CustomSort triggers node sorting or resorting, using a comparison routine indicated by the SortProc parameter. The Data parameter is passed to the comparison routine. The optional ARecurse parameter (default true) specifies that sorting should recursively descend the node tree and sort each subtree in turn.

If SortProc is nil (Delphi) or NULL (C++), a default comparison routine is used. The default routine uses the OnCompare (see page 4019) event handler, if defined. If the OnCompare (see page 4019) event handler is not defined, the default routine uses a simple case-sensitive compare of node captions.

In the comparison routine, the lParam1 and lParam2 parameters refer to two nodes when cast to TTreeNode (see page 5230). The lParamSort parameter is the value previously passed to the Data parameter of CustomSort. The return value of the comparison routine indicates the relative sort order of lParam1 and lParam2:

Return ValueMeaning

< 0lParam1 comes before lParam2

0lParam1 and lParam2 are equivalent

> 0lParam2 comes before lParam1

Calling CustomSort has the same effect as calling the same method for the Items (see page 4004) property, except that in TTreeNodes (see page 5266), CustomSort is not recursive by default.

To sort a subtree, call the CustomSort method of the Items (see page 4004) property.

C++ Examples:

```
/*
This example shows how to use the CustomSort method to order
a tree view in case-insensitive alphabetical order (either
forward or backward). The application must provide a
callback function such as CompareFunc below, which calls the
```

```
global AnsiStrIComp function to perform the actual
comparison.  This example requires a button and a populated
TreeView.
*/
int CALLBACK CompareFunc(long lParam1, long lParam2, long Reverse)
{
  TTreeNode *Node1 = ((TTreeNode *)lParam1);
  TTreeNode *Node2 = ((TTreeNode *)lParam2);
  if ((Node1 == NULL) || (Node2 == NULL)) return 0;
  int GT = AnsiStrIComp(Node1->Text.c_str(), Node2->Text.c_str());
  if (Reverse)
    return -GT;
  return GT;
}

/*
This procedure can then be used as a parameter to CustomSort
to sort the nodes of the tree view.  To sort in ascending
order, call:
*/

#include <CommCtrl.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TreeView1->CustomSort(CompareFunc, 0);
}
```

Delphi Examples:

```
{
This example shows how to use the CustomSort method to order
a tree view in case-insensitive alphabetical order (either
forward or backward). The application must provide a
callback function such as CompareFunc below, which calls the
global AnsiStrIComp function to perform the actual
comparison.  This example requires a button and a populated
TreeView.
}
function CustomSortProc(Node1, Node2: TTreeNode; Data: Integer): Integer; stdcall;
begin
  Result := -AnsiStrIComp(PChar(Node1.Text), PChar(Node2.Text));
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  TreeView1.CustomSort(@CustomSortProc, 0);
end;
```

See Also

AlphaSort (see page 3972)

OnCompare (see page 4019)

SortType (see page 4014)

27.1.23.5.17 TCustomTreeView.Delete Method

Generates an OnDeletion (see page 4024) event.

Pascal

```
procedure Delete(Node: TTreeNode); dynamic;
```

C++

```
__fastcall Delete(TTreeNode Node);
```

Description

Delete is called automatically immediately before the node specified by the Node parameter is deleted.

Override this method to make additional, class-specific responses in addition to the OnDeletion (see page 4024) event or to suppress the OnDeletion (see page 4024) event.

See Also

Items (see page 4004)

OnDeletion (see page 4024)

27.1.23.5.18 TCustomTreeView.Deselect Method

Deselects a specified node.

Pascal

```
procedure Deselect(Node: TTreeNode); virtual;
```

C++

```
virtual __fastcall Deselect(TTreeNode Node);
```

Description

Deselect removes the selection from Node. This method has no effect unless MultiSelect (see page 4005) is true and MultiSelectStyle (see page 4005) includes msControlSelect.

See Also

Items (see page 4004)

OnDeletion (see page 4024)

MultiSelect (see page 4005)

MultiSelectStyle (see page 4005)

27.1.23.5.19 TCustomTreeView.DestroyWnd Method

Destroys the control's window.

Pascal

```
procedure DestroyWnd; override;
```

C++

```
virtual __fastcall DestroyWnd();
```

Description

DestroyWnd is called when the tree view needs to replace its window. For example, changing properties that are implemented by window parameter settings require the tree view to be destroyed and then recreated using the CreateWnd (see page 3976) method.

Before the window is destroyed, DestroyWnd saves information about the tree view's items in memory, frees any device contexts, and finally calls DestroyWindowHandle. When the window is recreated, the stored items are added to the new window.

See Also

DestroyWindowHandle (see page 6140)

RecreateWnd (see page 6171)

CreateWnd (see page 3976)

Items (see page 4004)

27.1.23.5.20 TCustomTreeView.DoEndDrag Method

End drag event dispatcher.

Pascal

```
procedure DoEndDrag(Target: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall DoEndDrag(TObject * Target, int X, int Y);
```

Description

DoEndDrag is called automatically when the user stops dragging the tree view or one of its items. The drag operation may end either because the tree view was dropped on the object specified by Target, or because the user canceled the drag operation (if Target is nil (Delphi) or NULL (C++)).

Override DoEndDrag to perform any additional adjustments when the user stops dragging the tree view.

See Also

OnEndDrag (see page 5860)

27.1.23.5.21 TCustomTreeView.DoEnter Method

Respond to receiving input focus.

Pascal

```
procedure DoEnter; override;
```

C++

```
virtual __fastcall DoEnter();
```

Description

DoEnter is called automatically when the control receives the input focus. As implemented in TWinControl, DoEnter calls the OnEnter event handler, if defined.

Descendant classes that override DoEnter should always call the inherited method.

See Also

DoExit (see page 6144)

OnEnter (see page 6219)

27.1.23.5.22 TCustomTreeView.DoExit Method

Respond to losing input focus

Pascal

```
procedure DoExit; override;
```

C++

```
virtual __fastcall DoExit();
```

Description

DoExit is called automatically when the control receives the input focus. As implemented in TWinControl, DoExit calls the OnExit event handler, if defined.

Descendant classes that override DoExit should always call the inherited method.

See Also

DoEnter (see page 6144)

OnExit (see page 6221)

27.1.23.5.23 TCustomTreeView.DoStartDrag Method

Drag start event dispatcher.

Pascal

```
procedure DoStartDrag(var DragObject: TDragObject); override;
```

C++

```
virtual __fastcall DoStartDrag(TDragObject DragObject);
```

Description

DoStartDrag is called automatically when the user starts dragging the tree view or one of its items. It generates an OnStartDrag event, and then identifies the node (if any) that is being dragged. If the user is dragging a node (as opposed to the entire tree view), DoStartDrag generates an image for the node drag.

Override DoStartDrag to perform any additional adjustments when the user starts a drag-and-drop operation.

See Also

OnStartDrag (see page 5871)

GetDragImages (see page 3984)

27.1.23.5.24 TCustomTreeView.Edit Method

Edited event dispatcher.

Pascal

```
procedure Edit(const Item: TTVItem); dynamic;
```

C++

```
__fastcall Edit(const TTVItem Item);
```

Description

Edit is called automatically when the user edits the text of a tree view node. The Item parameter is the data structure supplied by the Windows control that identifies the node that was edited and the new value of its text. Edit unpacks this information into the parameters of an OnEdited (see page 4024) event handler, and allows the event handler to change the value of the node's text.

See Also

OnEdited (see page 4024)

Text (see page 5265)

27.1.23.5.25 TCustomTreeView.Expand Method

Completed node expansion event dispatcher.

Pascal

```
procedure Expand(Node: TTreeNode); dynamic;
```

C++

```
__fastcall Expand(TTreeNode Node);
```

Description

Expand is called automatically after the node specified by the Node parameter expands.

Override this method to make additional, class-specific responses in addition to the OnExpanded (see page 4025) event or to suppress the OnExpanded (see page 4025) event.

See Also

OnEdited (see page 4024)

Text (see page 5265)

27.1.23.5.26 TCustomTreeView.FindNextToSelect Method

Returns next selectable node.

Pascal

```
function FindNextToSelect: TTreeNode; virtual;
```

C++

```
virtual __fastcall TTreeNode FindNextToSelect();
```

Description

Starting with the node indicated by the Selected (see page 4007) property, FindNextToSelect searches for a deselected node, with preference given to subsequent siblings. If the Selected (see page 4007) node and all its siblings are selected, the search proceeds to the parent node and its siblings, and so on up the tree. If the top level is reached without finding a deselected node, nil is returned.

Note that the Selected (see page 4007) property can refer to a deselected node when MultiSelect (see page 4005) is true. In this case, FindNextToSelect returns that node.

See Also

Selected (see page 4007)

MultiSelect (see page 4005)

27.1.23.5.27 TCustomTreeView.FullCollapse Method

Collapses all the nodes within a tree view control.

Pascal

```
procedure FullCollapse;
```

C++

```
__fastcall FullCollapse();
```

Description

Call FullCollapse to hide all the nodes in the tree view except those at the first level.

See Also

OnCollapsing (see page 4018)

OnCollapsed (see page 4018)

Collapse (see page 5235)

27.1.23.5.28 TCustomTreeView.FullExpand Method

Expands all nodes within the tree view control.

Pascal

```
procedure FullExpand;
```

C++

```
__fastcall FullExpand();
```

Description

Call FullExpand to display all the nodes in the tree view. If ShowButtons (see page 4013) is true, all buttons will change from '+' to '-'.

See Also

OnExpanded (see page 4025)

OnExpanding (see page 4025)

ShowButtons (see page 4013)

Expand (see page 5242)

27.1.23.5.29 TCustomTreeView.GetDragImages Method

Returns the image list to be displayed during drag operations.

Pascal

```
function GetDragImages: TDragImageList; override;
```

C++

```
virtual __fastcall TDragImageList GetDragImages();
```

Description

GetDragImages is called automatically during drag-and-drop operations to allow the tree view to supply an image that is displayed while the tree view or one of its items is dragged. If an item (and not the entire tree view) is being dragged, GetDragImages returns the image for the node that was generated by the DoStartDrag (see page 3982) method.

See Also

DoStartDrag (see page 3982)

27.1.23.5.30 TCustomTreeView.GetHitTestInfoAt Method

Returns information about the location of a point relative to the client area of the tree view control.

Pascal

```
function GetHitTestInfoAt(X: Integer; Y: Integer): THitTests;
```

C++

```
__fastcall THitTests GetHitTestInfoAt(int X, int Y);
```

Description

Call GetHitTestInfoAt to determine what portion of the tree view, if any, sits under the point specified by the X and Y parameters. For example, use GetHitTestInfoAt to provide feedback about how to expand or collapse nodes when the mouse is over the relevant portions of the tree view.

GetHitTestInfo returns a THitTests (see page 5488) type. This set describes the possible tree view elements under the mouse.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code checks a list of htHitTest types and adds
the ones that occur when the mouse was pressed while on the
tree view.
}
procedure TForm1.TVMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  MyHitTest : THitTests;
```

```
begin
  MyHitTest := TV.GetHitTestInfoAt(X,Y);
  if htNowhere in MyHitTest then
    ListBox1.Items.Add('NoWhere');
  if htOnItem in MyHitTest then
    ListBox1.Items.Add('OnItem');
  if htOnButton in MyHitTest then
    ListBox1.Items.Add('OnButton');
  if htOnIndent in MyHitTest then
    ListBox1.Items.Add('OnIndent');
  if htOnLabel in MyHitTest then
    ListBox1.Items.Add('OnLabel');
  if htOnRight in MyHitTest then
    ListBox1.Items.Add('OnRight');
end;
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

GetNodeAt (see page 3987)

27.1.23.5.31 TCustomTreeView.GetImageIndex Method

Obtain image index.

Pascal

```
procedure GetImageIndex(Node: TTreeNode); virtual;
```

C++

```
virtual __fastcall GetImageIndex(TTreeNode Node);
```

Description

Call GetImageIndex to set the image index of a tree node. GetImageIndex as implemented in TCustomTreeView (see page

3957) does nothing except invoke the handler for the OnGetImageIndex (see page 4026) event, if one is assigned.

Descendants can override GetImageIndex to provide a class-specific implementation for setting the tree node's image index.

Node is the tree node in which the image index is set.

See Also

OnGetImageIndex (see page 4026)

Items (see page 4004)

TTreeNode (see page 5230)

ImageIndex (see page 5259)

TImageList (see page 6078)

27.1.23.5.32 TCustomTreeView.GetNodeAt Method

Returns the node that is found at the specified position.

Pascal

```
function GetNodeAt(X: Integer; Y: Integer): TTreeNode;
```

C++

```
__fastcall TTreeNode GetNodeAt(int X, int Y);
```

Description

Call GetNodeAt to access the node at the position specified by the X and Y parameters. X and Y specify the position in pixels relative to the top left corner of the tree view. If there is no node at the location, GetNodeAt returns nil (Delphi) or NULL (C++).

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
```

```
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

Items (see page 4004)

27.1.23.5.33 TCustomTreeView.GetSelectedIndex Method

Obtain selection index.

Pascal

```
procedure GetSelectedIndex(Node: TTreeNode); virtual;
```

C++

```
virtual __fastcall GetSelectedIndex(TTreeNode Node);
```

Description

Call GetSelectedIndex to set the image index of a tree node when the tree node is selected. GetSelectedIndex invokes the handler for the OnGetSelectedIndex (see page 4026) event, if one is assigned.

Descendants can override GetSelectedIndex to implement a class-specific method of setting the selected tree node's image index.

Node is the tree node in which the image index is set.

See Also

OnGetSelectedIndex (see page 4026)

Items (see page 4004)

TTreeNode (see page 5230)

SelectedIndex (see page 5265)

TImageList (see page 6078)

27.1.23.5.34 TCustomTreeView.GetSelections Method

Obtain selections list.

Pascal

```
function GetSelections(AList: TList): TTreeNode;
```

C++

```
__fastcall TTreeNode GetSelections(TList AList);
```

Description

GetSelections clears AList and copies all selected node objects to it. The return value is the Selected (see page 4007) property.

See Also

Selected (see page 4007)

27.1.23.5.35 TCustomTreeView.InvalidateSelectionsRects Method

Pascal

```
procedure InvalidateSelectionsRects;
```

C++

```
__fastcall InvalidateSelectionsRects();
```

Description

This is InvalidateSelectionsRects, a member of class TCustomTreeView.

27.1.23.5.36 TCustomTreeView.IsCustomDrawn Method

Specifies when the control should generate custom draw events.

Pascal

```
function IsCustomDrawn(Target: TCustomDrawTarget; Stage: TCustomDrawStage): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsCustomDrawn(TCustomDrawTarget Target, TCustomDrawStage Stage);
```

Description

TCustomTreeView (see page 3957) checks IsCustomDrawn to determine when to generate an OnCustomDraw (see page 4019), OnAdvancedCustomDraw (see page 4016), OnCustomDrawItem (see page 4021), or OnAdvancedCustomDrawItem (see page 4016) event. IsCustomDrawn returns true when the tree view should generate an event.

The Target parameter indicates whether the tree view is about to draw the entire tree or about to draw a node. IsCustomDrawn does not return true unless the tree view has an appropriate event handler for the target.

The Stage parameter indicates the current stage in the process of painting the tree view. IsCustomDrawn returns true if the

stage is cdPrePaint or if there is an advanced custom draw event handler for the target.

See Also

OnCustomDraw (see page 4019)

OnCustomDrawItem (see page 4021)

27.1.23.5.37 TCustomTreeView.IsEditing Method

Indicates whether a node is currently being edited by the user.

Pascal

```
function IsEditing: Boolean;
```

C++

```
__fastcall Boolean IsEditing();
```

Description

IsEditing returns true if any node label in the tree view is being edited.

See Also

Items (see page 4004)

27.1.23.5.38 TCustomTreeView.Loaded Method

Initializes the tree view after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the tree view's form is loaded into memory so that the tree view can complete any initializations that depend on other objects in the form.

Loaded ensures that the tree view appears fully expanded when it is first loaded into the designer.

See Also

FullExpand (see page 3984)

Loaded (see page 2782)

27.1.23.5.39 TCustomTreeView.LoadFromFile Method

Reads the file specified in FileName and loads the data into the tree view.

Pascal

```
procedure LoadFromFile(const FileName: string);
```

C++

```
__fastcall LoadFromFile(const AnsiString FileName);
```

Description

Use the LoadFromFile method to retrieve tree view nodes from a file and add them to the tree view. FileName should reference a file that was saved using the SaveToFile (see page 3992) method. Tree view files are text files that list the Text property of each node in a tree view, one node per line. Each node text value is preceded by spaces to indicate the depth of nesting.

Note: Tree view files contain only the text of the tree view nodes. They do not contain any data associated with the nodes.

See Also

LoadFromStream (see page 3991)

SaveToFile (see page 3992)

Text (see page 5265)

27.1.23.5.40 TCustomTreeView.LoadFromStream Method

Reads tree view data from a stream and stores the contents in the tree view.

Pascal

```
procedure LoadFromStream(Stream: TStream);
```

C++

```
__fastcall LoadFromStream(TStream Stream);
```

Description

Use LoadFromStream to read the nodes of the tree view from the specified stream. LoadFromStream adds the nodes in the stream data to the tree view.

The format for tree view data used by the SaveToStream (see page 3993) and LoadFromStream methods lists the Text property for each node. Data associated with the nodes is not saved or loaded.

For example, an application can save the information displayed in a tree view as the data in a Binary Large Object (BLOB) field of a dataset using the SaveToStream (see page 3993) method. LoadFromStream can later retrieve the data using a BLOB stream.

See Also

LoadFromFile (see page 3990)

SaveToStream (see page 3993)

CreateBlobStream (see page 6922)

Text (see page 5265)

27.1.23.5.41 TCustomTreeView.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove (see page 5773)

MouseUp (see page 5774)

OnMouseDown (see page 5862)

27.1.23.5.42 TCustomTreeView.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

27.1.23.5.43 TCustomTreeView.SaveToFile Method

Saves a the tree view to the file specified in FileName.

Pascal

```
procedure SaveToFile(const FileName: string);
```

C++

```
__fastcall SaveToFile(const AnsiString FileName);
```

Description

Use the SaveToFile method to store tree view data to a text file. The nodes can later be reloaded from the file into a new tree view object using the LoadFromFile (see page 3990) method.

Tree view files are text files that list the Text property of each node in a tree view, one node per line. Each node text value is preceded by spaces to indicate the depth of nesting.

Note: Tree view files contain only the text of the tree view nodes. They do not contain any data associated with the nodes.

See Also

LoadFromFile (see page 3990)

SaveToStream (see page 3993)

27.1.23.5.44 TCustomTreeView.SaveToStream Method

Writes the data in the tree view to the stream passed as the Stream parameter.

Pascal

```
procedure SaveToStream(Stream: TStream);
```

C++

```
__fastcall SaveToStream(TStream Stream);
```

Description

Use the SaveToStream method to stream out the nodes in a tree view. It can be streamed back in to another tree view object using the LoadFromStream (see page 3991) method.

The format for tree view data used by the SaveToStream and LoadFromStream (see page 3991) methods lists the Text property for each node. Data associated with the nodes is not saved or loaded.

See Also

LoadFromStream (see page 3991)

SaveToFile (see page 3992)

27.1.23.5.45 Select Method

This is the overview for the Select method overload.

Overload List

	Name	Description
	TCustomTreeView.Select (array of TTreeNode) (see page 3994)	Select specified nodes.
	TCustomTreeView.Select (TList) (see page 3994)	Select specified nodes.
	TCustomTreeView.Select (TTreeNode, TShiftState) (see page 3994)	Select specified nodes.

27.1.23.5.45.1 TCustomTreeView.Select Method (TList)

Select specified nodes.

Pascal

```
procedure Select(Node: TTreeNode; ShiftState: TShiftState = []); virtual; overload;
procedure Select(const Nodes: array of TTreeNode); virtual; overload;
procedure Select(Nodes: TList); virtual; overload;
```

C++

```
virtual __fastcall Select(TTreeNode Node, TShiftState ShiftState = []);
virtual __fastcall Select(const array of TTreeNode Nodes);
virtual __fastcall Select(TList Nodes);
```

Description

The select method selects one or more tree nodes.

In the first two forms, all nodes in the Nodes parameter are selected. Any other selected nodes are deselected.

In the third form, Node is selected as if clicked with the mouse. To achieve the effect of using the Control key, the Shift key, or the right mouse button, include ssCtrl, ssShift, or ssRight in the ShiftState parameter.

27.1.23.5.46 TCustomTreeView.SetDragMode Method

Sets the drag mode for the tree view.

Pascal

```
procedure SetDragMode(Value: TDragMode); override;
```

C++

```
virtual __fastcall SetDragMode(TDragMode Value);
```

Description

SetDragMode is the protected write implementation for the DragMode property. It sends a message to the tree view window to suppress automatic drag behavior when DragMode is set to dmManual.

See Also

DragMode

27.1.23.5.47 TCustomTreeView.Subselect Method

Toggle node selection.

Pascal

```
procedure Subselect(Node: TTreeNode; Validate: Boolean = False); virtual;
```

C++

```
virtual __fastcall Subselect(TTreeNode Node, Boolean Validate = False);
```

Description

Subselect toggles the selection state of Node. If Validate is true, Subselect ensures that the selection state information in view, list, and node objects are consistent, and that only those nodes allowed by MultiSelectStyle (see page 4005) are actually selected.

If Subselect is called when MultiSelect (see page 4005) is false, ETreeViewError (see page 3422) is raised.

See Also

ETreeViewError (see page 3422)

27.1.23.5.48 TCustomTreeView.ValidateSelection Method

Pascal

```
procedure ValidateSelection;
```

C++

```
__fastcall ValidateSelection();
```

Description

This is ValidateSelection, a member of class TCustomTreeView.

27.1.23.5.49 TCustomTreeView.WndProc Method

Provides specific message responses for the control.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc is the initial Windows message handler for the control. The WindowProc property is initialized to point to the WndProc method.

WndProc adjusts for some Windows message idiosyncrasies before calling the WndProc method inherited from TWinControl.

See Also

WindowProc (see page 5872)

DefWndProc (see page 6199)

MainWndProc (see page 6165)

WndProc (see page 6187)

27.1.23.6 TCustomTreeView Properties

The properties of the TCustomTreeView class are listed here.

Protected Properties

	Name	Description
	AutoExpand (see page 3996)	Specifies whether the nodes in the tree view automatically expand and collapse depending on the selection.
	BorderStyle (see page 3997)	Determines whether the tree view control has a border.
	ChangeDelay (see page 3999)	Specifies the delay between when a node is selected and when the OnChange (see page 4017) event occurs.
	CreateWndRestores (see page 4000)	This is CreateWndRestores, a member of class TCustomTreeView.
	HideSelection (see page 4000)	Determines whether a selected node appears selected when the focus shifts to another control.
	HotTrack (see page 4001)	Specifies whether list items are highlighted when the mouse passes over them.

	Images (see page 4001)	Determines which image list is associated with the tree view.
	Indent (see page 4001)	Specifies the amount of indentation in pixels when a list of child nodes is expanded.
	Items (see page 4004)	Lists the individual nodes that appear in the tree view control.
	MultiSelect (see page 4005)	Determines whether the user can select more than one tree node at a time.
	MultiSelectStyle (see page 4005)	Determines how multiple node selections work.
	ReadOnly (see page 4006)	Determines whether the user can edit the node labels.
	RightClickSelect (see page 4006)	Determines whether the Selected (see page 4007) property returns nodes that are selected using the right mouse button.
	RowSelect (see page 4007)	Specifies whether the entire row of the selected item is highlighted.
	ShowButtons (see page 4013)	Specifies whether to display plus (+) and minus (-) buttons to the left side of each parent item.
	ShowLines (see page 4013)	Specifies whether to display the lines that link child nodes to their corresponding parent nodes.
	ShowRoot (see page 4013)	Specifies whether lines connecting top-level nodes are displayed.
	SortType (see page 4014)	Determines if and how the nodes in a tree view are automatically sorted.
	StateImages (see page 4014)	Determines which image list to use for state images.
	ToolTips (see page 4014)	Specifies whether the items in the tree view have tool tips.

Public Properties

	Name	Description
R	Canvas (see page 3997)	Provides access to the canvas.
	DropTarget (see page 4000)	Specifies which item in the tree view appears as the target of a drag and drop operation.
	Selected (see page 4007)	Specifies the selected node in the tree view.
R	SelectionCount (see page 4012)	Returns number of nodes selected.
R	Selections (see page 4012)	Returns nodes selected.
	TopItem (see page 4015)	Specifies the topmost node that appears in the tree view.

27.1.23.6.1 TCustomTreeView.AutoExpand Property

Specifies whether the nodes in the tree view automatically expand and collapse depending on the selection.

Pascal

```
property AutoExpand: Boolean;
```

C++

```
__property Boolean AutoExpand;
```

Description

Set AutoExpand to true to cause the selected item to expand and the unselected items to collapse.

See Also

Selected (see page 4007)

Expanded (see page 5257)

Selected (see page 5264)

Expand (see page 5242)

Collapse (see page 5235)

OnExpanded (see page 4025)

OnExpanding (see page 4025)

OnCollapsed (see page 4018)

OnCollapsing (see page 4018)

27.1.23.6.2 TCustomTreeView.BorderStyle Property

Determines whether the tree view control has a border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Set BorderStyle to specify whether the tree view control should be outlined with a single-line border. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

See Also

BoundsRect (see page 5803)

27.1.23.6.3 TCustomTreeView.Canvas Property

Provides access to the canvas.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use the Canvas property to paint to the canvas from the OnCustomDraw (see page 4019) and OnCustomDrawItem (see page 4021) event handlers.

C++ Examples:

```
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TTreeView *myTreeView;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TTreeView")
  {
    myTreeView = (TTreeView *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myTreeView->Canvas->MoveTo(myTreeView->ClientRect.Left, myTreeView->ClientRect.Top);
    myTreeView->Canvas->LineTo(myTreeView->ClientRect.Right, myTreeView->ClientRect.Bottom);
```

```
    myTreeView = NULL;
  }
}
/*
The following example is taken from the custom draw demo. It
shows how the OnCustomDraw event handler draws the
background for the tree view before the items and lines are
drawn.
*/
void __fastcall TCustomDrawForm::TVCustomDraw(TCustomTreeView *Sender,
      const TRect &ARect, bool &DefaultDraw)
{
/*
This event should be used to draw any background colors or
images.  ARect represents the entire client area of the TreeView.
Use the TreeView's canvas to do the drawing.
Note that drawing a background bitmap is not really
supported by CustomDraw, so scrolling can get messy. Best to
subclass the TreeView and handle scrolling messages.
*/
  if (None1 != NULL)
  {
    if (None1->Checked) //no picture
    {
      TV->Canvas->Brush->Color = BkgColorDialog->Color;
      TV->Canvas->Brush->Style = FBrushStyle;
      TV->Canvas->FillRect(ARect);
    }
    else if (Tile1->Checked) //tile bitmap
    {
      TV->Canvas->Brush->Bitmap = Image1->Picture->Bitmap;
      TV->Canvas->FillRect(ARect);
    }
    else //Stretch across the canvas.
      TV->Canvas->StretchDraw(ARect, Image1->Picture->Bitmap);
  };
  DefaultDraw = FDefaultDraw;
  // Setting DefaultDraw to false here prevents all calls to
  // OnCustomDrawItem.
}
```

Delphi Examples:

```
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TTreeView) then
  begin
    myTreeView:= (ListBox1.Items.Objects[Index] as TTreeView);
    with myTreeView.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myTreeView.Canvas.MoveTo(Left, Top);
      myTreeView.Canvas.LineTo(Right, Bottom);
    end;
  myTreeView:= nil;
  end;
{
The following example is taken from the custom draw demo. It
shows how the OnCustomDraw event handler draws the
background for the tree view before the items and lines are
drawn.
}
procedure TCustomDrawForm.TVCustomDraw(Sender: TCustomTreeView; const ARect: TRect;
```

```
    var DefaultDraw: Boolean);
  begin
  {
  This event should be used to draw any background colors or
  images.  ARect represents the entire client area of the TreeView.
  Use the TreeView's canvas to do the drawing.
  Note that drawing a background bitmap is not really
  supported by CustomDraw, so scrolling can get messy. Best to
  subclass the TreeView and handle scrolling messages.
  }
    with TV.Canvas do
    begin
      if None1 <> nil then
      begin
        if None1.Checked then //no picture
        begin
          Brush.Color := BkgColorDialog.Color;
          Brush.Style := FBrushStyle;
          FillRect(ARect);
        end else
          if Tile1.Checked then //tile bitmap
          begin
            Brush.Bitmap := Image1.Picture.Bitmap;
              FillRect(ARect);
            end else //Stretch across the canvas.
                StretchDraw(ARect, Image1.Picture.Bitmap);
      end;
    end;
    DefaultDraw := FDefaultDraw;
    // Setting DefaultDraw to false here prevents all calls to
    // OnCustomDrawItem.
  end;
```

See Also

OnCustomDraw (see page 4019)

OnCustomDrawItem (see page 4021)

27.1.23.6.4 TCustomTreeView.ChangeDelay Property

Specifies the delay between when a node is selected and when the OnChange (see page 4017) event occurs.

Pascal

```
property ChangeDelay: Integer;
```

C++

```
__property int ChangeDelay;
```

Description

Use ChangeDelay to get or set the delay, in milliseconds, between when a node is selected and when the OnChange (see page 4017) event occurs.

Set the ChangeDelay to 50 milliseconds to emulate the behavior of the tree-view control used in Windows Explorer.

See Also

Selected (see page 4007)

OnChange (see page 4017)

27.1.23.6.5 TCustomTreeView.CreateWndRestores Property

Pascal

```
property CreateWndRestores: Boolean;
```

C++

```
__property Boolean CreateWndRestores;
```

Description

This is CreateWndRestores, a member of class TCustomTreeView.

27.1.23.6.6 TCustomTreeView.DropTarget Property

Specifies which item in the tree view appears as the target of a drag and drop operation.

Pascal

```
property DropTarget: TTreeNode;
```

C++

```
__property TTreeNode DropTarget;
```

Description

Read DropTarget to determine whether a node in the tree view is drawn as the target of a drag and drop operation. Set DropTarget when specifying a particular node in the tree view as the drop target of a dragged item.

Note: When DropTarget is set, the application must still handle the actual logic of accepting the dragged object by the indicated node.

See Also

EndDrag (see page 5757)

DropTarget (see page 5257)

27.1.23.6.7 TCustomTreeView.HideSelection Property

Determines whether a selected node appears selected when the focus shifts to another control.

Pascal

```
property HideSelection: Boolean;
```

C++

```
__property Boolean HideSelection;
```

Description

Use HideSelection to specify whether the user is given visual feedback about the current selection in the tree view when it does not have focus. If true, the selected node is not visually distinct from other nodes until focus returns to the control. If false, the node always appears selected.

See Also

Selected (see page 4007)

27.1.23.6.8 TCustomTreeView.HotTrack Property

Specifies whether list items are highlighted when the mouse passes over them.

Pascal

```
property HotTrack: Boolean;
```

C++

```
__property Boolean HotTrack;
```

Description

Set HotTrack to true to provide visual feedback about which item is under the mouse. Set HotTrack to false to suppress the visual feedback about which item is under the mouse.

See Also

HideSelection (see page 4000)

RowSelect (see page 4007)

27.1.23.6.9 TCustomTreeView.Images Property

Determines which image list is associated with the tree view.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Use Images to provide a customized list of bitmaps that can be displayed to the left of a node's label. Individual nodes specify the image from this list that should appear by setting their ImageIndex property.

See Also

OnGetImageIndex (see page 4026)

StateImages (see page 4014)

ImageIndex (see page 5259)

SelectedIndex (see page 5265)

27.1.23.6.10 TCustomTreeView.Indent Property

Specifies the amount of indentation in pixels when a list of child nodes is expanded.

Pascal

```
property Indent: Integer;
```

C++

```
__property int Indent;
```

Description

Use Indent to determine how far child nodes are indented from their parent nodes when the parent is expanded.

C++ Examples:

```
/*
The following example shows how the OnCustomDrawItem event
handler draws items and lines of the tree view after the
OnCustomDraw event handler has filled in the background.
*/
void __fastcall TCustomDrawForm::TVCustomDrawItem(TCustomTreeView *Sender,
      TTreeNode *Node, TCustomDrawState State, bool &DefaultDraw)
{
  TRect NodeRect;
/*
    If DefaultDraw it is true, any of the node's font
    properties can be changed. Note also that when
    DefaultDraw = True, Windows draws the buttons and
    ignores our font background colors, using instead the
    TreeView's Color property.
*/
    if (State.Contains(cdsSelected))
    {
      TV->Canvas->Font->Assign(SelectedFontDialog->Font);
      TV->Canvas->Brush->Color = SelBkgColorDialog->Color;
    };

    DefaultDraw = FDefaultDrawItem;
/*
    DefaultDraw = False means you have to handle all the
    item drawing yourself, including the buttons, lines,
    images, and text.
*/
    if (!DefaultDraw)
    {
      //draw the selection rect.
      if (State.Contains(cdsSelected))
      {
        NodeRect = Node->DisplayRect(True);
        TV->Canvas->FillRect(NodeRect);
      };
      NodeRect = Node->DisplayRect(False);

      if (None1->Checked)
      //no bitmap, so paint in the background color.
      {
        TV->Canvas->Brush->Color = BkgColorDialog->Color;
        TV->Canvas->Brush->Style = FBrushStyle;
        TV->Canvas->FillRect(NodeRect);
      }
      else
        //don't paint over the background bitmap.
        TV->Canvas->Brush->Style = bsClear;

      NodeRect.Left = NodeRect.Left + (Node->Level * TV->Indent);
      // NodeRect.Left now represents the left-most portion
      // of the expand button
      DrawButton(&NodeRect, Node); // See the CustomDraw demo

      NodeRect.Left = NodeRect.Left + TV->Indent + FButtonSize;
      //NodeRect->Left is now the leftmost portion of the image.
      DrawImage(&NodeRect, Node->ImageIndex); // See the CustomDraw demo

      NodeRect.Left = NodeRect.Left + ImageList->Width;
      //Now we are finally in a position to draw the text.

      TV->Canvas->TextOut(NodeRect.Left, NodeRect.Top, Node->Text);
    };
}
```

Delphi Examples:

```
{
The following example shows how the OnCustomDrawItem event
handler draws items and lines of the tree view after the
OnCustomDraw event handler has filled in the background.
}
procedure TCustomDrawForm.TVCustomDrawItem(Sender: TCustomTreeView; Node: TTreeNode;
  State: TCustomDrawState; var DefaultDraw: Boolean);
var
  NodeRect: TRect;
begin
  with TV.Canvas do
  begin
{
    If DefaultDraw it is true, any of the node's font
    properties can be changed. Note also that when
    DefaultDraw = True, Windows draws the buttons and
    ignores our font background colors, using instead the
    TreeView's Color property.
}
    if cdsSelected in State then
    begin
      Font.Assign(SelectedFontDialog.Font);
      Brush.Color := SelBkgColorDialog.Color;
    end;

    DefaultDraw := False; // FDefaultDrawItem;
{
    DefaultDraw = False means you have to handle all the
    item drawing yourself, including the buttons, lines,
    images, and text.
}
    if not DefaultDraw then
    begin
      //draw the selection rect.
      if cdsSelected in State then
      begin
        NodeRect := Node.DisplayRect(True);
        FillRect(NodeRect);
      end;
      NodeRect := Node.DisplayRect(False);

      if None1.Checked then
      //no bitmap, so paint in the background color.
      begin
        Brush.Color := BkgColorDialog.Color;
        Brush.Style := FBrushStyle;
        FillRect(NodeRect)
      end
      else
        //don't paint over the background bitmap.
        Brush.Style := bsClear;

      NodeRect.Left := NodeRect.Left + (Node.Level * TV.Indent);
      // NodeRect.Left now represents the left-most portion
      // of the expand button
      DrawButton(NodeRect, Node); // See the CustomDraw demo

      NodeRect.Left := NodeRect.Left + TV.Indent + FButtonSize;
      //NodeRect.Left is now the leftmost portion of the image.
      DrawImage(NodeRect, Node.ImageIndex); // See the CustomDraw demo

      NodeRect.Left := NodeRect.Left + ImageList.Width;
      //Now we are finally in a position to draw the text.
```

```
      TextOut(NodeRect.Left, NodeRect.Top, Node.Text);
    end;
  end;
end;
```

See Also

FullExpand (see page 3984)

27.1.23.6.11 TCustomTreeView.Items Property

Lists the individual nodes that appear in the tree view control.

Pascal

```
property Items: TTreeNodes;
```

C++

```
__property TTreeNodes Items;
```

Description

Individual nodes in a tree view are TTreeNode (see page 5230) objects. These individual nodes can be accessed by using the Items property along with the item's index into the tree view. For example, to access the second item in the tree view, you could use the following code.

```
MyTreeNode := TreeView1.Items[1];
MyTreeNode = TreeView1->Items[1];
```

When setting this property at design-time in the Object Inspector the Tree View Items Editor appears. Use the New Item and New SubItem buttons to add items to the tree view. Use the Text property to modify what text is displayed in the label of the item.

At run-time nodes can be added and inserted by using the TTreeNodes (see page 5266) methods AddChildFirst, AddChild, AddChildObjectFirst, AddChildObject, AddFirst, Add, AddObjectFirst, AddObject, and Insert.

Note: Accessing tree view items by index can be time-intensive, particularly when the tree view contains many items. For optimal performance, try to design your application so that it has as few dependencies on the tree view's item index as possible.

C++ Examples:

```
/*
The following example deletes an item in the tree view when
the user clicks on it, and if the checkbox is checked. This example
requires a populated TreeView.
*/
void __fastcall TForm1::TVMouseDown(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  THitTests HT;
  if (CheckBox1->Checked && Sender->ClassNameIs("TTreeView"))
  {
    TTreeView *pTV = (TTreeView *)Sender;
    HT = pTV->GetHitTestInfoAt(X,Y);
    if (HT.Contains(htOnItem))
      pTV->Items->Delete(pTV->GetNodeAt(X,Y));
  }
}
```

Delphi Examples:

```
{
The following example deletes an item in the tree view when
the user clicks on it, and if the checkbox is checked. This example
requires a populated TreeView.
```

```
}
procedure TForm1.TVMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  HT : THitTests;
begin
if (CheckBox1.Checked) and (Sender is TTreeView) then
  begin
  with Sender as TTreeView do
    begin
    HT := GetHitTestInfoAt(X,Y);
    if (htOnItem in HT) then
      Items.Delete(GetNodeAt(X,Y));
    end;
  end;
end;
```

See Also

TreeView (see page 5266)

AddChildFirst (see page 5273)

AddChild (see page 5271)

AddChildObject (see page 5273)

AddChildObjectFirst (see page 5274)

Add (see page 5270)

AddObject (see page 5276)

AddObjectFirst (see page 5279)

AddFirst (see page 5275)

27.1.23.6.12 TCustomTreeView.MultiSelect Property

Determines whether the user can select more than one tree node at a time.

Pascal

```
property MultiSelect: Boolean;
```

C++

```
__property Boolean MultiSelect;
```

Description

Set MultiSelect to specify whether users can select multiple nodes using the Control and Shift keys. A selection style must also be chosen in MultiSelectStyle (see page 4005).

See Also

MultiSelectStyle (see page 4005)

27.1.23.6.13 TCustomTreeView.MultiSelectStyle Property

Determines how multiple node selections work.

Pascal

```
property MultiSelectStyle: TMultiSelectStyle;
```

C++

```
__property TMultiSelectStyle MultiSelectStyle;
```

Description

MultiSelectStyle determines how multiple selections are made when MultiSelect (see page 4005) is true. MultiSelectStyle must include at least one of the following values.

Value	Meaning
msControlSelect	Clicking on any node with the Control key pressed toggles the selection of that node.
msShiftSelect	Clicking on any node with the Shift key pressed selects that node, the last single node selected, and the nodes in between. All other nodes are deselected.
msVisibleOnly	Multiple selections with the Shift key do not include child nodes of collapsed nodes.
msSiblingOnly	Selected (see page 4007) nodes are restricted to a single set of siblings.

If msControlSelect or msShiftSelect are in effect, the last singly-selected node becomes the primary selection, referenced by Selections (see page 4012)[0]. The primary selection is the anchor for extended selections using the Shift key.

See Also

Selections (see page 4012)

27.1.23.6.14 TCustomTreeView.ReadOnly Property

Determines whether the user can edit the node labels.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to specify whether the user can edit the nodes of the tree view. If ReadOnly is true, the user can expand and collapse nodes, but can't edit their labels. If ReadOnly is false, the user can edit the labels as well. The default value is false.

See Also

Text (see page 5265)

27.1.23.6.15 TCustomTreeView.RightClickSelect Property

Determines whether the Selected (see page 4007) property returns nodes that are selected using the right mouse button.

Pascal

```
property RightClickSelect: Boolean;
```

C++

```
__property Boolean RightClickSelect;
```

Description

Use RightClickSelect to allow the Selected (see page 4007) property to indicate nodes the user clicks with the right mouse button. If RightClickSelect is true, the value of Selected (see page 4007) is the value of the node last clicked with either the right or left mouse button. If RightClickSelect is false, the value of Selected (see page 4007) is the node last clicked using the left mouse button.

RightClickSelect affects only the value of the Selected (see page 4007) property. It does not cause the tree view to highlight a new node if the node is selected using the right mouse button.

Note: RightClickSelect must be set to true before the user right-clicks the tree view for it to affect the value of the Selected (see page 4007) property.

See Also

Selected (see page 4007)

OnMouseDown (see page 5862)

27.1.23.6.16 TCustomTreeView.RowSelect Property

Specifies whether the entire row of the selected item is highlighted.

Pascal

```
property RowSelect: Boolean;
```

C++

```
__property Boolean RowSelect;
```

Description

Set RowSelect to true to cause the entire row of the selected item to be highlighted.

RowSelect is ignored if ShowLines (see page 4013) is true.

See Also

Selected (see page 4007)

ShowLines (see page 4013)

27.1.23.6.17 TCustomTreeView.Selected Property

Specifies the selected node in the tree view.

Pascal

```
property Selected: TTreeNode;
```

C++

```
__property TTreeNode Selected;
```

Description

Read Selected to access the selected node of the tree view. If there is no selected node, the value of Selected is nil (Delphi) or NULL (C++).

Set Selected to set a node in the tree view. When a node becomes selected, the tree view's OnChanging (see page 4018) and OnChange (see page 4017) events occur. Also, if the specified node is the child of a collapsed parent item, the parent's list of child items is expanded to reveal the specified node. In this case, the tree view's OnExpanded (see page 4025) and OnExpanding (see page 4025) events occur as well.

If the RightClickSelect (see page 4006) property is true, the value of Selected is the last node that was clicked in the tree view, even if it was clicked with the right mouse button. This may differ from the node that is highlighted to indicate selection.

If the MultiSelect (see page 4005) property is true and the MultiSelectStyle (see page 4005) property includes msControlSelect, then Selected returns the last node clicked on, even if that click deselected the node. For a current selection status when MultiSelect (see page 4005) is true, refer to the Selections (see page 4012) property.

C++ Examples:

```
/*
This example requires two buttons, three TEdits, a label and
a populated TreeView.  The following code defines a record
type of TMyRec and a record pointer type of PMyRec.
*/
typedef struct MyRec
{
  AnsiString FName, LName;
} TMyRec;

typedef TMyRec* PMyRec;
/*
Assuming these types are used, the following code adds a
node to TreeView1 as the last sibling of the node specified
by the Index.  A TMyRec record is associated with the added
item. The FName and LName fields are obtained from edit
boxes Edit1 and Edit2. The Index parameter is obtained from
edit box Edit3.  The item is added only if the Index is a
valid value.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PMyRec  MyRecPtr;
  int TreeViewIndex;
  TTreeNodes* items;

  MyRecPtr = new TMyRec;
  MyRecPtr->FName = Edit1->Text;
  MyRecPtr->LName = Edit2->Text;
  TreeViewIndex = StrToInt(Edit3->Text);
  items = TreeView1->Items;
  if (items->Count == 0)
    items->AddObject(NULL, "Item" + IntToStr(TreeViewIndex), MyRecPtr);
  else if ((TreeViewIndex < items->Count) && (TreeViewIndex >= 0))
    items->AddObject(items->Item[TreeViewIndex], "Item" + IntToStr(TreeViewIndex) + "after" +
items->Item[TreeViewIndex]->Text, MyRecPtr);
}
/*
After an item containing a TMyRec record has been added, the
following code retrieves the FName and LName values
associated with the item and displays the values in a label.
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  if (TreeView1->Selected->Data != NULL)
    Label1->Caption = PMyRec(TreeView1->Selected->Data)->FName + " " +
                  PMyRec(TreeView1->Selected->Data)->LName;
}
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
```

```
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
/*
The following code uses Item to add the labels of all child
nodes of the selected node to a list box when a button is
clicked.  This example requires a button, a listbox, and a
populated treeview.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < TreeView1->Selected->Count; i++)
    ListBox1->Items->Add(TreeView1->Selected->Item[i]->Text);
}
```

Delphi Examples:

```
{
This example is an event handler for a popup menu item. The
handler enables editing of the current item in a TListView
control. The example would also work with a TTreeView
control, or with any instance of a control class derived
from TCustomViewControl.
}
procedure TForm1.EditItem1Click(Sender: TObject);
begin
  TreeView1.Selected.EditText;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  MyTreeNode1, MyTreeNode2: TTreeNode;
begin
  with TreeView1.Items do
  begin
    Clear; { remove any existing nodes }
    MyTreeNode1 := Add(nil, 'RootTreeNode1'); { Add a root node }
    { Add a child node to the node just added }
    AddChild(MyTreeNode1,'ChildNode1');

    {Add another root node}
    MyTreeNode2 := Add(MyTreeNode1, 'RootTreeNode2');
    {Give MyTreeNode2 to a child }
    AddChild(MyTreeNode2,'ChildNode2');

    {Change MyTreeNode2 to ChildNode2 }
    { and add a child node to it}
    MyTreeNode2 := TreeView1.Items[3];
    AddChild(MyTreeNode2,'ChildNode2a');

    {Add another child to ChildNode2, after ChildNode2a }
    AddChild(MyTreeNode2,'ChildNode2b');

    {add another root node}
```

```
    Add(MyTreeNode1, 'RootTreeNode3');
  end;
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  PopupMenu1.Popup(Left + X, Top + Y);
end;
{
This example requires three buttons, four TEdits, two labels
and a populated TreeView.  The following code defines a
record type of TMyRec and a record pointer type of PMyRec.
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Edit1: TEdit;
    TreeView1: TTreeView;
    Label2: TLabel;
    Label3: TLabel;
    Button3: TButton;
    Edit4: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  type
  PMyRec = ^TMyRec;
  TMyRec = record
    FName: string;
    LName: string;
  end;


var
  Form1: TForm1;

implementation

{$R *.dfm}

{
Assuming these types are used, the following code adds a
node to TreeView1 as the last sibling of the selected node.
A TMyRec record is associated with the added item. The FName
and LName fields are obtained from edit boxes Edit1 and
Edit2.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  MyRecPtr: PMyRec;
  TreeViewIndex: LongInt;
begin
  New(MyRecPtr);
  MyRecPtr^.FName := Edit1.Text;
  MyRecPtr^.LName := Edit2.Text;
  with TreeView1 do
  begin
```

```
    TreeViewIndex := Selected.AbsoluteIndex;
    if Items.Count = 0 then
      Items.AddObject(nil, 'Item' + IntToStr(TreeViewIndex), MyRecPtr)
    else if (TreeViewIndex < Items.Count) and (TreeViewIndex >= 0) then
      Items.AddObject(Items[TreeViewIndex], 'Item' + IntToStr(TreeViewIndex), MyRecPtr);
  end;
end;
{
After an item containing a TMyRec record has been added, the
following code retrieves the FName and LName values
associated with the item and displays the values in a label.
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  if (TreeView1.Selected.Data <> nil) then { query only works on new nodes }
    Edit3.Text := PMyRec(TreeView1.Selected.Data)^.FName + ' ' +
                  PMyRec(TreeView1.Selected.Data)^.LName;
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
  Edit4.Text := IntToStr(TreeView1.Selected.AbsoluteIndex);
end;
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
{
The following code uses Item to add the labels of all child
nodes of the selected node to a list box when a button is
clicked.  This example requires a button, a listbox, and a
populated treeview.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I : Integer;
begin
  for I := 0 to (TreeView1.Selected.Count - 1) do
    ListBox1.Items.Add(TreeView1.Selected.Item[I].Text);
```

```
end;
```

See Also

OnChange (see page 4017)

OnChanging (see page 4018)

SelectedIndex (see page 5265)

RightClickSelect (see page 4006)

RowSelect (see page 4007)

OnExpanded (see page 4025)

OnExpanding (see page 4025)

MultiSelect (see page 4005)

MultiSelectStyle (see page 4005)

Selections (see page 4012)

27.1.23.6.18 TCustomTreeView.SelectionCount Property

Returns number of nodes selected.

Pascal

```
property SelectionCount: Cardinal;
```

C++

```
__property unsigned SelectionCount;
```

Description

SelectionCount returns the number of nodes currently selected.

27.1.23.6.19 TCustomTreeView.Selections Property

Returns nodes selected.

Pascal

```
property Selections [Index: Integer]: TTreeNode;
```

C++

```
__property TTreeNode Selections[int Index];
```

Description

Selections returns one of selected nodes. The maximum value of Index is SelectionCount (see page 4012)-1. If more than one node is selected, Selections[0] is the primary selected node. This node is the starting point for extended selections if MultiSelectStyle (see page 4005) includes msShiftSelect.

See Also

SelectionCount (see page 4012)

MultiSelectStyle (see page 4005)

27.1.23.6.20 TCustomTreeView.ShowButtons Property

Specifies whether to display plus (+) and minus (-) buttons to the left side of each parent item.

Pascal

```
property ShowButtons: Boolean;
```

C++

```
__property Boolean ShowButtons;
```

Description

If ShowButtons is true, a button will appear to the left of each parent item. The user can click the button to expand or collapse the child items as an alternative to double-clicking the parent item.

See Also

ShowLines (see page 4013)

ShowRoot (see page 4013)

27.1.23.6.21 TCustomTreeView.ShowLines Property

Specifies whether to display the lines that link child nodes to their corresponding parent nodes.

Pascal

```
property ShowLines: Boolean;
```

C++

```
__property Boolean ShowLines;
```

Description

If ShowLines is true, lines linking child nodes to their parent nodes are displayed. Nodes at the root of the hierarchy are not automatically linked. To link nodes at the root, the ShowRoot (see page 4013) property must also be set to true.

See Also

ShowButtons (see page 4013)

ShowRoot (see page 4013)

27.1.23.6.22 TCustomTreeView.ShowRoot Property

Specifies whether lines connecting top-level nodes are displayed.

Pascal

```
property ShowRoot: Boolean;
```

C++

```
__property Boolean ShowRoot;
```

Description

To show lines connecting top-level nodes to a single root, set the tree view's ShowRoot and ShowLines (see page 4013) properties to true.

See Also

ShowButtons (see page 4013)

ShowLines (see page 4013)

27.1.23.6.23 TCustomTreeView.SortType Property

Determines if and how the nodes in a tree view are automatically sorted.

Pascal

```
property SortType: TSortType;
```

C++

```
__property TSortType SortType;
```

Description

Once a tree view is sorted, the original hierarchy is lost. That is, setting the SortType back to stNone will not restore the original order of items. These are the possible values:

Value	Meaning
stNone	No sorting is done.
stData	The items are sorted when the Data object or SortType is changed.
stText	The items are sorted when the Caption or SortType is changed.
stBoth	The items are sorted when either the Data object, the Caption or SortType is changed

Optionally, the OnCompare (see page 4019) event can be hooked to handle comparisons.

See Also

AlphaSort (see page 3972)

OnCompare (see page 4019)

27.1.23.6.24 TCustomTreeView.StateImages Property

Determines which image list to use for state images.

Pascal

```
property StateImages: TCustomImageList;
```

C++

```
__property TCustomImageList StateImages;
```

Description

Use StateImages to provide a set of bitmaps that reflect the state of tree view nodes. The state image appears as an additional image to the left of the item's icon.

See Also

Images (see page 4001)

StateIndex (see page 5265)

27.1.23.6.25 TCustomTreeView.ToolTips Property

Specifies whether the items in the tree view have tool tips.

Pascal

```
property ToolTips: Boolean;
```

C++

```
__property Boolean ToolTips;
```

Description

Set ToolTips to true to specify that items in the tree view control have tool tips (Help Hints).

Specify the ToolTip text in an OnHint event handler using the Hint property.

See Also

Hint (see page 5827)

OnHint (see page 13731)

27.1.23.6.26 TCustomTreeView.TopItem Property

Specifies the topmost node that appears in the tree view.

Pascal

```
property TopItem: TTreeNode;
```

C++

```
__property TTreeNode TopItem;
```

Description

When TopItem is changed, the tree view scrolls vertically so that the specified node is topmost in the list view.

See Also

GetFirstNode (see page 5284)

27.1.23.7 TCustomTreeView Events

The events of the TCustomTreeView class are listed here.

Protected Events

	Name	Description
	OnAddition (see page 4016)	Occurs when new node is added.
	OnAdvancedCustomDraw (see page 4016)	Occurs at discrete stages during the painting of the tree view control.
	OnAdvancedCustomDrawItem (see page 4016)	Occurs at discrete stages during the painting of tree view nodes.
	OnCancelEdit (see page 4017)	Occurs when user cancels editing of a node's Caption.
	OnChange (see page 4017)	Occurs whenever the selection has changed from one node to another.
	OnChanging (see page 4018)	Occurs when the selection is about to change from one node to another.
	OnCollapsed (see page 4018)	Occurs after a node has been collapsed.
	OnCollapsing (see page 4018)	Occurs when a node is about to be collapsed.
	OnCompare (see page 4019)	Occurs when two nodes must be compared during a sort of the nodes in the tree view.
	OnCreateNodeClass (see page 4019)	Occurs when a node in the treeview is about to be created.
	OnCustomDraw (see page 4019)	Occurs immediately prior to painting the tree view control.
	OnCustomDrawItem (see page 4021)	Occurs immediately prior to painting a node in a tree view control.
	OnDeletion (see page 4024)	Occurs when a node in the tree view is deleted.
	OnEdited (see page 4024)	Occurs after the user edits the Text property of a node.
	OnEditing (see page 4024)	Occurs when the user starts to edit the Text property of a node.

	OnExpanded (see page 4025)	Occurs after a node is expanded.
	OnExpanding (see page 4025)	Occurs when a node is about to be expanded.
	OnGetImageIndex (see page 4026)	Occurs when the tree view looks up the ImageIndex of a node.
	OnGetSelectedIndex (see page 4026)	Occurs when the tree view looks up the SelectedIndex of a node.

27.1.23.7.1 TCustomTreeView.OnAddition Event

Occurs when new node is added.

Pascal

```
property OnAddition: TTVExpandedEvent;
```

C++

```
__property TTVExpandedEvent OnAddition;
```

Description

OnAddition occurs when a new node is added to the control.

27.1.23.7.2 TCustomTreeView.OnAdvancedCustomDraw Event

Occurs at discrete stages during the painting of the tree view control.

Pascal

```
property OnAdvancedCustomDraw: TTVAdvancedCustomDrawEvent;
```

C++

```
__property TTVAdvancedCustomDrawEvent OnAdvancedCustomDraw;
```

Description

Write an OnAdvancedCustomDraw event handler to paint an owner-drawn tree view. Use the Canvas (see page 3997) property as a drawing surface when painting the image of the tree view.

To paint individual items, use the OnAdvancedCustomDrawItem (see page 4016) or OnCustomDrawItem (see page 4021) event instead.

Note: OnAdvancedCustomDraw occurs at several stages during the paint process, not just immediately prior to the default rendering. If you only need to use the cdPrePaint stage, it is more efficient to use the OnCustomDraw (see page 4019) event.

See Also

Canvas (see page 3997)

OnAdvancedCustomDrawItem (see page 4016)

OnCustomDraw (see page 4019)

27.1.23.7.3 TCustomTreeView.OnAdvancedCustomDrawItem Event

Occurs at discrete stages during the painting of tree view nodes.

Pascal

```
property OnAdvancedCustomDrawItem: TTVAdvancedCustomDrawItemEvent;
```

C++

```
__property TTVAdvancedCustomDrawItemEvent OnAdvancedCustomDrawItem;
```

Description

Write an OnAdvancedCustomDrawItem event handler to customize the painting of individual items in the tree view.

Note: OnAdvancedCustomDrawItem occurs at several stages during the paint process and allows you to suppress the default painting of tree node images while still allowing the default painting of the node. If you only need to use the cdPrePaint stage and do not want to suppress only the painting of tree node images, it is more efficient to use the OnCustomDrawItem (see page 4021) event.

See Also

Canvas (see page 3997)

OnAdvancedCustomDraw (see page 4016)

OnCustomDrawItem (see page 4021)

27.1.23.7.4 TCustomTreeView.OnCancelEdit Event

Occurs when user cancels editing of a node's Caption.

Pascal

```
property OnCancelEdit: TTVChangedEvent;
```

C++

```
__property TTVChangedEvent OnCancelEdit;
```

Description

The OnCancelEdit event occurs when a user cancels a node-editing operation.

Sender is the tree view that contains the node.

Node is the node whose editing was canceled.

27.1.23.7.5 TCustomTreeView.OnChange Event

Occurs whenever the selection has changed from one node to another.

Pascal

```
property OnChange: TTVChangedEvent;
```

C++

```
__property TTVChangedEvent OnChange;
```

Description

Write an OnChange event handler to take specific action when the selected node changes. The Sender parameter is the tree view whose selected node changes, and the Node parameter is the newly selected node.

Note: The OnChange event does not occur for nodes selected using the right mouse button when RightClickSelect (see page 4006) is true. To respond to changes in the value of the Selected (see page 4007) property when RightClickSelect (see page 4006) is true, use the OnMouseUp event.

See Also

OnChanging (see page 4018)

Selected (see page 4007)

RightClickSelect (see page 4006)

ChangeDelay (see page 3999)

OnMouseUp (see page 5867)

27.1.23.7.6 TCustomTreeView.OnChanging Event

Occurs when the selection is about to change from one node to another.

Pascal

```
property OnChanging: TTVChangingEvent;
```

C++

```
__property TTVChangingEvent OnChanging;
```

Description

Write an OnChanging event handler to selectively prevent selection from moving to specific nodes.

The TTVChangingEvent (see page 5514) type points to a method that is called when the selection is about to be changed from one node to another. Set AllowChange to false, to prevent selection from moving to a new node. The Node parameter specifies the node that is about to be selected.

See Also

OnChange (see page 4017)

Selected (see page 4007)

27.1.23.7.7 TCustomTreeView.OnCollapsed Event

Occurs after a node has been collapsed.

Pascal

```
property OnCollapsed: TTVExpandedEvent;
```

C++

```
__property TTVExpandedEvent OnCollapsed;
```

Description

Write an OnCollapsed event handler to respond after a node in the tree view collapses. The Node parameter is the node whose children are no longer visible.

See Also

Collapse (see page 5235)

FullCollapse (see page 3983)

27.1.23.7.8 TCustomTreeView.OnCollapsing Event

Occurs when a node is about to be collapsed.

Pascal

```
property OnCollapsing: TTVCollapsingEvent;
```

C++

```
__property TTVCollapsingEvent OnCollapsing;
```

Description

The TTVCollapsingEvent (see page 5514) type points to a method that is called when a node is about to be collapsed. Set the AllowCollapse parameter to false to prevent the node specified by the Node parameter from being collapsed.

See Also

Collapse (see page 5235)

FullCollapse (see page 3983)

27.1.23.7.9 TCustomTreeView.OnCompare Event

Occurs when two nodes must be compared during a sort of the nodes in the tree view.

Pascal

```
property OnCompare: TTVCompareEvent;
```

C++

```
__property TTVCompareEvent OnCompare;
```

Description

Write an OnCompare event handler to customize the sort order of the nodes in the tree view. Set the Compare parameter to a value less than 0 if Node1 is less than Node2. Set Compare to 0 if Node1 is equivalent to Node2, and set Compare to a value greater than 0 if Node1 is greater than Node2. If an OnCompare event handler is not provided, tree view nodes are sorted alphabetically, based on their labels.

See Also

AlphaSort (see page 3972)

CustomSort (see page 3978)

SortType (see page 4014)

27.1.23.7.10 TCustomTreeView.OnCreateNodeClass Event

Occurs when a node in the treeview is about to be created.

Pascal

```
property OnCreateNodeClass: TTVCreateNodeClassEvent;
```

C++

```
__property TTVCreateNodeClassEvent OnCreateNodeClass;
```

Description

OnCreateNodeClass occurs when a new node object is about to be created.

Sender is the tree view object that is about to add a new node.

NodeClass returns a class reference that is instantiated to create a new node object.

27.1.23.7.11 TCustomTreeView.OnCustomDraw Event

Occurs immediately prior to painting the tree view control.

Pascal

```
property OnCustomDraw: TTVCustomDrawEvent;
```

C++

```
__property TTVCustomDrawEvent OnCustomDraw;
```

Description

Write an OnCustomDraw event handler to paint an owner-drawn tree view. Use the Canvas (see page 3997) property as a drawing surface when painting the image of the tree view.

To paint individual items, use the OnCustomDrawItem (see page 4021) event instead.

Note: OnCustomDraw only occurs prior to painting the tree view control. To customize the painting at other stages of the paint process (such as after the default drawing), use OnAdvancedCustomDraw (see page 4016) instead.

C++ Examples:

```
/*
The following example is taken from the custom draw demo. It
shows how the OnCustomDraw event handler draws the
background for the tree view before the items and lines are
drawn.
*/
void __fastcall TCustomDrawForm::TVCustomDraw(TCustomTreeView *Sender,
      const TRect &ARect, bool &DefaultDraw)
{
/*
This event should be used to draw any background colors or
images.  ARect represents the entire client area of the TreeView.
Use the TreeView's canvas to do the drawing.
Note that drawing a background bitmap is not really
supported by CustomDraw, so scrolling can get messy. Best to
subclass the TreeView and handle scrolling messages.
*/
  if (None1 != NULL)
  {
    if (None1->Checked) //no picture
    {
      TV->Canvas->Brush->Color = BkgColorDialog->Color;
      TV->Canvas->Brush->Style = FBrushStyle;
      TV->Canvas->FillRect(ARect);
    }
    else if (Tile1->Checked) //tile bitmap
    {
      TV->Canvas->Brush->Bitmap = Image1->Picture->Bitmap;
      TV->Canvas->FillRect(ARect);
    }
    else //Stretch across the canvas.
      TV->Canvas->StretchDraw(ARect, Image1->Picture->Bitmap);
  };
  DefaultDraw = FDefaultDraw;
  // Setting DefaultDraw to false here prevents all calls to
  // OnCustomDrawItem.
}
```

Delphi Examples:

```
{
The following example is taken from the custom draw demo. It
shows how the OnCustomDraw event handler draws the
background for the tree view before the items and lines are
drawn.
```

```
}
procedure TCustomDrawForm.TVCustomDraw(Sender: TCustomTreeView; const ARect: TRect;
  var DefaultDraw: Boolean);
begin
{
This event should be used to draw any background colors or
images.  ARect represents the entire client area of the TreeView.
Use the TreeView's canvas to do the drawing.
Note that drawing a background bitmap is not really
supported by CustomDraw, so scrolling can get messy. Best to
subclass the TreeView and handle scrolling messages.
}
  with TV.Canvas do
  begin
    if None1 <> nil then
    begin
      if None1.Checked then //no picture
      begin
        Brush.Color := BkgColorDialog.Color;
        Brush.Style := FBrushStyle;
        FillRect(ARect);
      end else
        if Tile1.Checked then //tile bitmap
        begin
          Brush.Bitmap := Image1.Picture.Bitmap;
            FillRect(ARect);
          end else //Stretch across the canvas.
              StretchDraw(ARect, Image1.Picture.Bitmap);
    end;
  end;
  DefaultDraw := FDefaultDraw;
  // Setting DefaultDraw to false here prevents all calls to
  // OnCustomDrawItem.
end;
```

See Also

Canvas (see page 3997)

OnCustomDrawItem (see page 4021)

OnAdvancedCustomDraw (see page 4016)

27.1.23.7.12 TCustomTreeView.OnCustomDrawItem Event

Occurs immediately prior to painting a node in a tree view control.

Pascal

```
property OnCustomDrawItem: TTVCustomDrawItemEvent;
```

C++

```
__property TTVCustomDrawItemEvent OnCustomDrawItem;
```

Description

Write an OnCustomDrawItem event handler to paint individual items in the tree view, or to provide a background to the item before the default rendering of the item.

Note: OnCustomDrawItem only occurs prior to painting individual tree items. To customize the painting of items at other stages of the paint process (such as after the item is painted), use OnAdvancedCustomDrawItem (see page 4016) instead.

C++ Examples:

```
/*
The following example shows how the OnCustomDrawItem event
handler draws items and lines of the tree view after the
OnCustomDraw event handler has filled in the background.
*/
void __fastcall TCustomDrawForm::TVCustomDrawItem(TCustomTreeView *Sender,
      TTreeNode *Node, TCustomDrawState State, bool &DefaultDraw)
{
  TRect NodeRect;
/*
    If DefaultDraw it is true, any of the node's font
    properties can be changed. Note also that when
    DefaultDraw = True, Windows draws the buttons and
    ignores our font background colors, using instead the
    TreeView's Color property.
*/
    if (State.Contains(cdsSelected))
    {
      TV->Canvas->Font->Assign(SelectedFontDialog->Font);
      TV->Canvas->Brush->Color = SelBkgColorDialog->Color;
    };

    DefaultDraw = FDefaultDrawItem;
/*
    DefaultDraw = False means you have to handle all the
    item drawing yourself, including the buttons, lines,
    images, and text.
*/
    if (!DefaultDraw)
    {
      //draw the selection rect.
      if (State.Contains(cdsSelected))
      {
        NodeRect = Node->DisplayRect(True);
        TV->Canvas->FillRect(NodeRect);
      };
      NodeRect = Node->DisplayRect(False);

      if (None1->Checked)
      //no bitmap, so paint in the background color.
      {
        TV->Canvas->Brush->Color = BkgColorDialog->Color;
        TV->Canvas->Brush->Style = FBrushStyle;
        TV->Canvas->FillRect(NodeRect);
      }
      else
        //don't paint over the background bitmap.
        TV->Canvas->Brush->Style = bsClear;

      NodeRect.Left = NodeRect.Left + (Node->Level * TV->Indent);
      // NodeRect.Left now represents the left-most portion
      // of the expand button
      DrawButton(&NodeRect, Node); // See the CustomDraw demo

      NodeRect.Left = NodeRect.Left + TV->Indent + FButtonSize;
      //NodeRect->Left is now the leftmost portion of the image.
      DrawImage(&NodeRect, Node->ImageIndex); // See the CustomDraw demo

      NodeRect.Left = NodeRect.Left + ImageList->Width;
      //Now we are finally in a position to draw the text.

      TV->Canvas->TextOut(NodeRect.Left, NodeRect.Top, Node->Text);
    };
}
```

Delphi Examples:

```
{
The following example shows how the OnCustomDrawItem event
handler draws items and lines of the tree view after the
OnCustomDraw event handler has filled in the background.
}
procedure TCustomDrawForm.TVCustomDrawItem(Sender: TCustomTreeView; Node: TTreeNode;
  State: TCustomDrawState; var DefaultDraw: Boolean);
var
  NodeRect: TRect;
begin
  with TV.Canvas do
  begin
{
    If DefaultDraw it is true, any of the node's font
    properties can be changed. Note also that when
    DefaultDraw = True, Windows draws the buttons and
    ignores our font background colors, using instead the
    TreeView's Color property.
}
    if cdsSelected in State then
    begin
      Font.Assign(SelectedFontDialog.Font);
      Brush.Color := SelBkgColorDialog.Color;
    end;

    DefaultDraw := False; // FDefaultDrawItem;
{
    DefaultDraw = False means you have to handle all the
    item drawing yourself, including the buttons, lines,
    images, and text.
}
    if not DefaultDraw then
    begin
      //draw the selection rect.
      if cdsSelected in State then
      begin
        NodeRect := Node.DisplayRect(True);
        FillRect(NodeRect);
      end;
      NodeRect := Node.DisplayRect(False);

      if None1.Checked then
      //no bitmap, so paint in the background color.
      begin
        Brush.Color := BkgColorDialog.Color;
        Brush.Style := FBrushStyle;
        FillRect(NodeRect)
      end
      else
        //don't paint over the background bitmap.
        Brush.Style := bsClear;

      NodeRect.Left := NodeRect.Left + (Node.Level * TV.Indent);
      // NodeRect.Left now represents the left-most portion
      // of the expand button
      DrawButton(NodeRect, Node); // See the CustomDraw demo

      NodeRect.Left := NodeRect.Left + TV.Indent + FButtonSize;
      //NodeRect.Left is now the leftmost portion of the image.
      DrawImage(NodeRect, Node.ImageIndex); // See the CustomDraw demo

      NodeRect.Left := NodeRect.Left + ImageList.Width;
      //Now we are finally in a position to draw the text.

      TextOut(NodeRect.Left, NodeRect.Top, Node.Text);
    end;
```

```
    end;
  end;
```

See Also

Canvas (see page 3997)

OnCustomDraw (see page 4019)

OnAdvancedCustomDrawItem (see page 4016)

27.1.23.7.13 TCustomTreeView.OnDeletion Event

Occurs when a node in the tree view is deleted.

Pascal

```
property OnDeletion: TTVExpandedEvent;
```

C++

```
__property TTVExpandedEvent OnDeletion;
```

Description

Write an OnDeletion event handler to respond when a node is deleted from the tree view control.

See Also

Delete (see page 5237)

27.1.23.7.14 TCustomTreeView.OnEdited Event

Occurs after the user edits the Text property of a node.

Pascal

```
property OnEdited: TTVEditedEvent;
```

C++

```
__property TTVEditedEvent OnEdited;
```

Description

Write an OnEdited event handler to respond to changes the user makes to the node labels. The Node parameter is the node whose label was edited. The S parameter is the new value of the node's Text property. The node's label can be changed in an OnEdited event handler before the user's edits are committed. This event can occur only if ReadOnly (see page 4006) is set to false.

See Also

OnEditing (see page 4024)

ReadOnly (see page 4006)

Text (see page 5265)

27.1.23.7.15 TCustomTreeView.OnEditing Event

Occurs when the user starts to edit the Text property of a node.

Pascal

```
property OnEditing: TTVEditingEvent;
```

C++

```
__property TTVEditingEvent OnEditing;
```

Description

Write an OnEditing event handler to determine whether the user is allowed to edit the label of a specific node in the tree view. Set the AllowEdit parameter to false to prevent the user from editing the node specified by the Node parameter. To disallow editing of all nodes in the tree view, use the ReadOnly (see page 4006) property instead.

See Also

OnEdited (see page 4024)

ReadOnly (see page 4006)

27.1.23.7.16 TCustomTreeView.OnExpanded Event

Occurs after a node is expanded.

Pascal

```
property OnExpanded: TTVExpandedEvent;
```

C++

```
__property TTVExpandedEvent OnExpanded;
```

Description

Write an OnExpanded event handler to respond when a node in the tree view is expanded. The Node parameter specifies the node whose children are now displayed to the user.

See Also

FullExpand (see page 3984)

OnExpanding (see page 4025)

Expand (see page 5242)

27.1.23.7.17 TCustomTreeView.OnExpanding Event

Occurs when a node is about to be expanded.

Pascal

```
property OnExpanding: TTVExpandingEvent;
```

C++

```
__property TTVExpandingEvent OnExpanding;
```

Description

Write an OnExpanding event handler to determine whether a node can be expanded. Set the AllowExpansion parameter to false to prevent the node from expanding.

See Also

FullExpand (see page 3984)

OnExpanded (see page 4025)

Expand (see page 5242)

27.1.23.7.18 TCustomTreeView.OnGetImageIndex Event

Occurs when the tree view looks up the ImageIndex of a node.

Pascal

```
property OnGetImageIndex: TTVExpandedEvent;
```

C++

```
__property TTVExpandedEvent OnGetImageIndex;
```

Description

Write an OnGetImageIndex event handler to change the image index for the particular node before it is drawn. For example, the bitmap of a node can be changed to indicate a different state for the node.

Delphi Examples:

```
{
The following example shows how the OnGetImageIndex event
handler can be used to indicate a different state for the
node.
}
procedure TCustomDrawForm.TVGetImageIndex(Sender: TObject; Node: TTreeNode);
begin
  if Node.HasChildren then
    if Node.Expanded then
      Node.ImageIndex := 3
    else
      Node.ImageIndex := 0
  else
    Node.ImageIndex := 1;
end;

procedure TCustomDrawForm.TVGetSelectedIndex(Sender: TObject; Node: TTreeNode);
begin
  Node.SelectedIndex := Node.ImageIndex;
end;
```

See Also

Images (see page 4001)

OnGetSelectedIndex (see page 4026)

ImageIndex (see page 5259)

27.1.23.7.19 TCustomTreeView.OnGetSelectedIndex Event

Occurs when the tree view looks up the SelectedIndex of a node.

Pascal

```
property OnGetSelectedIndex: TTVExpandedEvent;
```

C++

```
__property TTVExpandedEvent OnGetSelectedIndex;
```

Description

Write an OnGetSelectedIndex event handler to change the selected image index of a node before it is drawn.

Delphi Examples:

```
{
```

```
The following example shows how the OnGetImageIndex event
handler can be used to indicate a different state for the
node.
}
procedure TCustomDrawForm.TVGetImageIndex(Sender: TObject; Node: TTreeNode);
begin
  if Node.HasChildren then
    if Node.Expanded then
      Node.ImageIndex := 3
    else
      Node.ImageIndex := 0
  else
    Node.ImageIndex := 1;
end;

procedure TCustomDrawForm.TVGetSelectedIndex(Sender: TObject; Node: TTreeNode);
begin
  Node.SelectedIndex := Node.ImageIndex;
end;
```

See Also

Images (see page 4001)

OnGetImageIndex (see page 4026)

SelectedIndex (see page 5265)

27.1.24 TCustomUpDown Class

TCustomUpDown is the base type for up-down size controls such as TUpDown (see page 5380).

Class Hierarchy

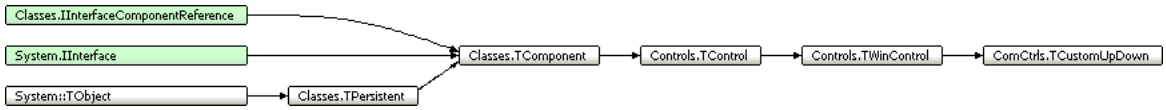

Pascal

```
TCustomUpDown = class(TWinControl);
```

C++

```
class TCustomUpDown : public TWinControl;
```

File

ComCtrls

Description

Use TCustomUpDown as a base class when defining up-down controls such as the arrows that appear in a spin box. Up-down controls allow users to change the size of a numerical value by clicking on arrow buttons. TCustomUpDown introduces several properties, events, and methods to associate an up-down control with another control in a form and to communicate the size changes specified by the user.

Do not create instances of TCustomUpDown. To put an up-down control in a form, use TUpDown (see page 5380).

See Also

TUpDown (see page 5380)

27.1.24.1 TCustomUpDown Members

The following tables list the members exposed by TCustomUpDown.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnChanging (see page 4046)	Occurs when the position of an up-down control is about to change.
	OnChangingEx (see page 4047)	Occurs when the position of an up-down control is about to change.
	OnClick (see page 4047)	Occurs when the user clicks an arrow button.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.

	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanChange (see page 4039)	Indicates whether Position (see page 4045) can change.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 4039)	Generates an OnClick (see page 4047) event.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 4039)	Initializes the window creation parameters for the up-down control.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 4040)	Creates the window used by the up-down control.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.

	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DoCanChange (see page 4040)	Pending change event dispatcher.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.

	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 4041)	Responds to notifications when components are about to be created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.

	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AlignButton (see page 4042)	Specifies the position of the up-down control relative to its companion control.
	ArrowKeys (see page 4042)	Determines whether the up-down control receives input from the Up arrow and Down arrow keys.
	Associate (see page 4043)	Specifies a companion control for the up-down control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Increment (see page 4043)	Specifies the amount the Position (see page 4045) value changes each time the up or down button is pressed.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	Max (see page 4043)	Specifies the maximum value of the Position (see page 4045) property.
	Min (see page 4044)	Specifies the minimum value of the Position (see page 4045) property.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	Orientation (see page 4044)	Specifies the direction that the arrow keys point.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	Position (see page 4045)	Specifies the current value represented by the up-down control.

	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	Thousands (see page 4045)	Determines whether a thousands separator appears between every three digits of a decimal string.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.
	Wrap (see page 4045)	Specifies how the up-down control handles attempts to exceed the value of Max (see page 4043) or Min (see page 4044).

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 4038)	Creates and initializes an instance of TCustomUpDown (see page 4027).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.

	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.

R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.24.2 TCustomUpDown.Create Constructor

Creates and initializes an instance of TCustomUpDown (see page 4027).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomUpDown(TComponent * AOwner);
```

Description

Call Create to instantiate an up-down control at runtime. Up-down controls placed on forms at design time are created automatically.AOwner is a component, typically the form, that is responsible for freeing the up-down control. It becomes the value of the Owner property.

See Also

Owner (see page 2807)

27.1.24.3 TCustomUpDown Methods

The methods of the TCustomUpDown class are listed here.

Protected Methods

	Name	Description
	CanChange (see page 4039)	Indicates whether Position (see page 4045) can change.
	Click (see page 4039)	Generates an OnClick (see page 4047) event.
V	CreateParams (see page 4039)	Initializes the window creation parameters for the up-down control.
V	CreateWnd (see page 4040)	Creates the window used by the up-down control.
	DoCanChange (see page 4040)	Pending change event dispatcher.
V	Notification (see page 4041)	Responds to notifications when components are about to be created or destroyed.

27.1.24.3.1 TCustomUpDown.CanChange Method

Indicates whether Position (see page 4045) can change.

Pascal

```
function CanChange: Boolean; dynamic;
```

C++

```
__fastcall Boolean CanChange();
```

Description

CanChange is called automatically when the user clicks the arrow buttons or an attempt is made to change the value of Position (see page 4045) programmatically. CanChange generates an OnChanging (see page 4046) event followed by an OnChangingEx (see page 4047) event. It allows the change unless an OnChanging (see page 4046) or OnChangingEx (see page 4047) event handler prevents it.

See Also

OnChanging (see page 4046)

Position (see page 4045)

OnChangingEx (see page 4047)

DoCanChange (see page 4040)

27.1.24.3.2 TCustomUpDown.Click Method

Generates an OnClick (see page 4047) event.

Pascal

```
procedure Click(Button: TUDBtnType); dynamic; reintroduce;
```

C++

```
__fastcall Click(TUDBtnType Button);
```

Description

Click is called automatically when the user clicks an arrow button or presses an Up or Down arrow key. Override this method to specify special processing that should occur before the OnClick (see page 4047) event handler executes, or to block the OnClick (see page 4047) event. The Button parameter specifies which button was clicked: btNext refers to the Up or Right arrow, and btPrev refers to the Down or Left arrow.

See Also

OnClick (see page 4047)

27.1.24.3.3 TCustomUpDown.CreateParams Method

Initializes the window creation parameters for the up-down control.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

CreateParams is called automatically before the window for an up-down control is created. CreateParams calls the inherited method, then sets the Params parameter to reflect the values of the AlignButton (see page 4042), ArrowKeys (see page 4042), Orientation (see page 4044), Thousands (see page 4045), and Wrap (see page 4045) properties.

See Also

AlignButton (see page 4042)

ArrowKeys (see page 4042)

Orientation (see page 4044)

Thousands (see page 4045)

CreateParams (see page 6135)

Wrap (see page 4045)

27.1.24.3.4 TCustomUpDown.CreateWnd Method

Creates the window used by the up-down control.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

After calling the inherited method to create the up-down control's window, CreateWnd sends the newly created windows messages so that it reflects the current values of the Associate (see page 4043), Increment (see page 4043), Max (see page 4043), Min (see page 4044), and Position (see page 4045) properties.

See Also

Associate (see page 4043)

CreateParams (see page 4039)

Increment (see page 4043)

Max (see page 4043)

Min (see page 4044)

Position (see page 4045)

CreateWnd (see page 6137)

27.1.24.3.5 TCustomUpDown.DoCanChange Method

Pending change event dispatcher.

Pascal

```
function DoCanChange(NewVal: SmallInt; Delta: SmallInt): Boolean;
```

C++

```
__fastcall Boolean DoCanChange(SmallInt NewVal, SmallInt Delta);
```

Description

DoCanChange is called automatically when the user clicks the arrow buttons or an attempt is made to change the value of Position (see page 4045) programmatically. DoCanChange stores the value of the NewVal and Delta parameters and then calls CanChange (see page 4039) to generate an OnChanging (see page 4046) event followed by an OnChangingEx (see page 4047) event. It allows the change to position unless an OnChanging (see page 4046) or OnChangingEx (see page 4047) event handler prevents it.

NewVal specifies the proposed new value of Position (see page 4045) if the change is allowed.

Delta indicates the size of the change, where positive values increase Position (see page 4045) and negative values decrease Position (see page 4045).

DoCanChange returns true if the change is allowed, false otherwise.

See Also

OnChanging (see page 4046)

Position (see page 4045)

OnChangingEx (see page 4047)

CanChange (see page 4039)

27.1.24.3.6 TCustomUpDown.Notification Method

Responds to notifications when components are about to be created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

After calling the inherited method, Notification checks whether the control specified by the Associate (see page 4043) property is about to be destroyed, and if so, sets the Associate (see page 4043) property to nil (Delphi) or NULL (C++).

See Also

Associate (see page 4043)

27.1.24.4 TCustomUpDown Properties

The properties of the TCustomUpDown class are listed here.

Protected Properties

	Name	Description
	AlignButton (see page 4042)	Specifies the position of the up-down control relative to its companion control.
	ArrowKeys (see page 4042)	Determines whether the up-down control receives input from the Up arrow and Down arrow keys.
	Associate (see page 4043)	Specifies a companion control for the up-down control.
	Increment (see page 4043)	Specifies the amount the Position (see page 4045) value changes each time the up or down button is pressed.
	Max (see page 4043)	Specifies the maximum value of the Position (see page 4045) property.
	Min (see page 4044)	Specifies the minimum value of the Position (see page 4045) property.
	Orientation (see page 4044)	Specifies the direction that the arrow keys point.

	Position (see page 4045)	Specifies the current value represented by the up-down control.
	Thousands (see page 4045)	Determines whether a thousands separator appears between every three digits of a decimal string.
	Wrap (see page 4045)	Specifies how the up-down control handles attempts to exceed the value of Max (see page 4043) or Min (see page 4044).

27.1.24.4.1 TCustomUpDown.AlignButton Property

Specifies the position of the up-down control relative to its companion control.

Pascal

```
property AlignButton: TUDAlignButton;
```

C++

```
__property TUDAlignButton AlignButton;
```

Description

Set AlignButton to indicate where to position the up-down control. The up-down control appears attached to the outer edge of the control specified by the Associate (see page 4043) property. These are the possible values:

Value	Meaning
udLeft	Position (see page 4045) the up-down control on the left edge of the companion control.
udRight	Position (see page 4045) the up-down control on the right edge of the companion control.

The up-down control resizes itself to the height of the companion control.

See Also

Associate (see page 4043)

27.1.24.4.2 TCustomUpDown.ArrowKeys Property

Determines whether the up-down control receives input from the Up arrow and Down arrow keys.

Pascal

```
property ArrowKeys: Boolean;
```

C++

```
__property Boolean ArrowKeys;
```

Description

Use ArrowKeys to specify whether the up-down control responds to keyboard input. ArrowKeys is used only when the up-down control is associated with a companion control. When ArrowKeys is true, the up-down control responds to the Up arrow and Down arrow keys, even when its companion control has input focus. When ArrowKeys is false, the user must click the buttons of the up-down control with the mouse. To set a companion control use the Associate (see page 4043) property.

Note: Up-down controls can respond to only the Up and Down arrow keys, even when the value of Orientation (see page 4044) is udHorizontal.

See Also

Associate (see page 4043)

Orientation (see page 4044)

27.1.24.4.3 TCustomUpDown.Associate Property

Specifies a companion control for the up-down control.

Pascal

```
property Associate: TWinControl;
```

C++

```
__property TWinControl * Associate;
```

Description

Use Associate to link the up-down control to a companion control (sometimes called a buddy window). The up-down control automatically positions itself on the outer edge of the companion control and adjusts its height to match the companion control.

At design-time the companion control does not move with the up-down control. To keep the up-down control and its companion control together at design time, either multi-select both controls before moving them, or first move the companion control and then reset the Associate property for the up-down control.

See Also

AlignButton (see page 4042)

ArrowKeys (see page 4042)

Position (see page 4045)

27.1.24.4.4 TCustomUpDown.Increment Property

Specifies the amount the Position (see page 4045) value changes each time the up or down button is pressed.

Pascal

```
property Increment: Integer;
```

C++

```
__property int Increment;
```

Description

Use Increment to control the granularity of changes the up-down control can make. Set Increment to indicate the number of units between Min (see page 4044) and Max (see page 4043) the Position (see page 4045) moves whenever the user clicks on one of the arrow buttons. Increment represents the mapping between the units used to specify Min (see page 4044), Max (see page 4043), and Position (see page 4045), and the logical units represented by the up-down control.

See Also

Max (see page 4043)

Min (see page 4044)

Position (see page 4045)

27.1.24.4.5 TCustomUpDown.Max Property

Specifies the maximum value of the Position (see page 4045) property.

Pascal

```
property Max: SmallInt;
```

C++

```
__property SmallInt Max;
```

Description

Use Max to provide an upper limit to the range of values the Position (see page 4045) property can assume.

See Also

Min (see page 4044)

Position (see page 4045)

27.1.24.4.6 TCustomUpDown.Min Property

Specifies the minimum value of the Position (see page 4045) property.

Pascal

```
property Min: SmallInt;
```

C++

```
__property SmallInt Min;
```

Description

Use Min to provide a lower limit to the range of values the Position (see page 4045) property can assume.

See Also

Max (see page 4043)

Position (see page 4045)

27.1.24.4.7 TCustomUpDown.Orientation Property

Specifies the direction that the arrow keys point.

Pascal

```
property Orientation: TUDOrientation;
```

C++

```
__property TUDOrientation Orientation;
```

Description

Use Orientation to determine the orientation of the arrow buttons. These are the possible values:

Value	Meaning
udHorizontal	The arrow buttons point to the left and right, and appear side by side.
udVertical	The arrow buttons point up and down and are stacked vertically.

Note: Even when the Orientation is udHorizontal, the up-down control can respond only to the Up and Down arrow keys.

See Also

ArrowKeys (see page 4042)

27.1.24.4.8 TCustomUpDown.Position Property

Specifies the current value represented by the up-down control.

Pascal

```
property Position: SmallInt;
```

C++

```
__property SmallInt Position;
```

Description

Use Position to get or change the value the up-down control represents. Position must be a value between Min (see page 4044) and Max (see page 4043).

When the Associate (see page 4043) property specifies an edit control, its text is automatically set to the value of Position. When Position changes, the text of the edit control is updated. If the Associate (see page 4043) property is nil (Delphi) or NULL (C++), use the OnClick (see page 4047) event to respond to changes in the value of Position.

See Also

Associate (see page 4043)

Max (see page 4043)

Min (see page 4044)

OnChanging (see page 4046)

Thousands (see page 4045)

27.1.24.4.9 TCustomUpDown.Thousands Property

Determines whether a thousands separator appears between every three digits of a decimal string.

Pascal

```
property Thousands: Boolean;
```

C++

```
__property Boolean Thousands;
```

Description

Use Thousands to control the formatting of the text of the companion control of the up-down control. If Thousands is true, a separator appears between every three digits. The numeric value that is displayed by the companion control is given by the Position (see page 4045) property.

See Also

Associate (see page 4043)

Position (see page 4045)

27.1.24.4.10 TCustomUpDown.Wrap Property

Specifies how the up-down control handles attempts to exceed the value of Max (see page 4043) or Min (see page 4044).

Pascal

```
property Wrap: Boolean;
```

C++

```
__property Boolean Wrap;
```

Description

Use Wrap to specify whether the up-down control treats the range determined by the Max (see page 4043) and Min (see page 4044) properties as a continuous loop. If Wrap is true, the value of Position (see page 4045) becomes the value of the Min (see page 4044) property when the user increments Position (see page 4045) beyond the value of Max (see page 4043), and becomes the value of Max (see page 4043) when the user decrements Position (see page 4045) beyond the value of Min (see page 4044).

See Also

Max (see page 4043)

Min (see page 4044)

Position (see page 4045)

27.1.24.5 TCustomUpDown Events

The events of the TCustomUpDown class are listed here.

Protected Events

	Name	Description
	OnChanging (see page 4046)	Occurs when the position of an up-down control is about to change.
	OnChangingEx (see page 4047)	Occurs when the position of an up-down control is about to change.
	OnClick (see page 4047)	Occurs when the user clicks an arrow button.

27.1.24.5.1 TCustomUpDown.OnChanging Event

Occurs when the position of an up-down control is about to change.

Pascal

```
property OnChanging: TUDChangingEvent;
```

C++

```
__property TUDChangingEvent OnChanging;
```

Description

Write an OnChanging event handler to conditionally disallow changes to the value of Position (see page 4045). Set the AllowChange parameter to false to prevent a change to the current value of Position (see page 4045).

OnChanging occurs both when the user clicks the arrow buttons of the up-down control and when an attempt is made to change the value of Position (see page 4045) programmatically. To determine which button the user clicked, use the OnChangingEx (see page 4047) event or the OnClick (see page 4047) event, both of which follow the OnChanging event.

See Also

CanChange (see page 4039)

OnClick (see page 4047)

OnChangingEx (see page 4047)

Position (see page 4045)

27.1.24.5.2 TCustomUpDown.OnChangingEx Event

Occurs when the position of an up-down control is about to change.

Pascal

```
property OnChangingEx: TUDChangingEventEx;
```

C++

```
__property TUDChangingEventEx OnChangingEx;
```

Description

Write an OnChangingEx event handler to conditionally disallow changes to the value of Position (see page 4045). Set the AllowChange parameter to false to prevent a change to the current value of Position (see page 4045).

Sender is the up-down control that received the event. This may be in response to a user click on the arrow buttons or an attempt to change the value of Position (see page 4045) programmatically.

AllowChange returns a value that indicates whether the change is permitted.

NewValue indicates the new value of Position (see page 4045) if the change is allowed.

Direction indicates whether the value of Position (see page 4045) is to be increased or decreased.

See Also

CanChange (see page 4039)

Position (see page 4045)

OnChanging (see page 4046)

27.1.24.5.3 TCustomUpDown.OnClick Event

Occurs when the user clicks an arrow button.

Pascal

```
property OnClick: TUDClickEvent;
```

C++

```
__property TUDClickEvent OnClick;
```

Description

Write an OnClick event handler to take specific action when the user clicks one of the arrow buttons. If the Associate (see page 4043) property is not used to automatically update the text of an edit control when the value of Position (see page 4045) changes, use the OnClick event handler to respond to changes just before the value of Position (see page 4045) is changed.

The Button parameter specifies how the value of Position (see page 4045) is about to change. When Button is btNext, the Up or Right arrow was clicked, and the value of Position (see page 4045) is about to increase by Increment (see page 4043). When Button is btPrev, the Down or Left arrow was clicked, and the value of Position (see page 4045) is about to decrease by Increment (see page 4043).

C++ Examples:

```
/*
This example requires a new TPageControl, as-is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
```

```
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
evaluates as true.  If Button is not btNext, the previous Tab Page
is selected.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
const
  TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *page = new TTabSheet(PageControl1);
    page->PageControl = PageControl1;
    page->Caption = AnsiString("Page") + IntToStr(i);
    page->Brush->Color = colorPalette[i];
  }
}

void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  PageControl1->SelectNextPage(Button == btNext);
}
```

Delphi Examples:

```
{
This example requires a new TPageControl, as-is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
evaluates as true.  If Button is not btNext, the previous Tab Page
is selected.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
const
  colorPalette: Array[0..11] of TColor =
    (clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet #' + IntToStr(i);
      Brush.Color := colorPalette[i];
    end;
end;

procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
begin
  PageControl1.SelectNextPage(Button = btNext);
end;
```

See Also

Associate (see page 4043)

Increment (see page 4043)

OnChanging (see page 4046)

Position (see page 4045)

27.1.25 TDateTimePicker Class

TDateTimePicker displays a list box for entering dates or times.

Class Hierarchy

Pascal

```
TDateTimePicker = class(TCommonCalendar);
```

C++

```
class TDateTimePicker : public TCommonCalendar;
```

File

ComCtrls

Description

TDateTimePicker is a visual component designed specifically for entering dates or times. In dmComboBox date mode, it resembles a list box or combo box, except that the drop-down list is replaced with a calendar illustration; users can select a date from the calendar. Dates or times can also be selected by scrolling with Up and Down arrows and by typing.

Date (see page 4075)-time picker ignores the BiDiMode (see page 4072) setting for right-to-left reading, displaying dates according to the system locale.

TDateTimePicker formats date and time values according to the date and time settings in the Regional Settings of the Control panel on the user's system. Because TDateTimePicker is a wrapper for a Windows control, these formats can't be changed by changing the formatting variables in the SysUtils (see page 22709) unit. However, you can use the Windows API call DateTime_SetFormat to programmatically specify these settings.

Note: The underlying Windows control is known to behave unpredictably when used to set dates in 1752, the year English-speaking countries changed to the Gregorian calendar. Similar problems may appear for different dates when a non-English locale is used. If an application might be used to specify dates on or before the changeover in the current locale (for example, 1918 or before in Russian locales), be sure to test the application using the dates affected.

See Also

TComboBox (see page 21064)

TListBox (see page 21598)

TDateTimePicker

Date-time Pickers and Month Calendars

List Controls

27.1.25.1 TDateTimePicker Members

The following tables list the members exposed by TDateTimePicker.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetMonthInfo (see page 3581)	Called whenever a new month is displayed in the calendar.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.

	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 4061)	This is Change, a member of class TDateTimePicker.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CheckEmptyDate (see page 4061)	Clears the check box associated with the calendar.
	CheckValidDate (see page 3571)	Confirms that a date falls between MinDate (see page 3579) and MaxDate (see page 3578).
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 4062)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 4062)	Creates the windows control for the calendar.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus

	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetCalendarHandle (see page 4063)	Returns the calendar control's window handle.
	GetCalStyles (see page 3573)	Returns the style flags that are used when creating the calendar's window.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).

	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	MsgSetCalColors (see page 4063)	Sets the calendar's colors.
	MsgSetDateTime (see page 4064)	Sets the calendar's date/time.
	MsgSetRange (see page 4064)	Implements the MinDate and MaxDate properties.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.

	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	CalendarHandle (see page 3576)	Indicates the calendar control's window handle.
	CalExceptionClass (see page 3576)	Specifies the calendar's exception class.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.

	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	EndDate (see page 3577)	Indicates the last date that in the range of selected dates.
	FirstDayOfWeek (see page 3578)	Specifies the left-most day on the calendar.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MaxSelectRange (see page 3579)	Specifies the maximum number of days that can be selected.
	MonthDelta (see page 3579)	Specifies how many months that the calendar scrolls.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	MultiSelect (see page 3580)	Specifies whether multiple dates can be selected on the calendar.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ShowToday (see page 3580)	Specifies whether today's date is shown below the calendar.
	ShowTodayCircle (see page 3580)	Specifies whether today's date is circled on the calendar.
	Text (see page 5842)	Contains a text string associated with the control.
	WeekNumbers (see page 3581)	Specifies whether week numbers are shown to the left of the calendar.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BoldDays (see page 3570)	Encodes a set of dates for use in the OnGetMonthInfo (see page 3581) event handler.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 4060)	Creates and initializes a TDateTimePicker (see page 4049) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.

	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 3570)	Destroys the TCommonCalendar (see page 3558) instance.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.

	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetDesignVisible (see page 6178)	Makes a control visible at design time.
	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Update (see page 5789)	Processes any pending paint messages immediately.
	Update (see page 6183)	Forces the control to update.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DateTime (see page 4076)	Indicates the date and time.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	DroppedDown (see page 4079)	Specifies whether the date time picker control is dropped down.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.

	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 4096)	Occurs when a date or time is entered.
	OnClick (see page 4097)	Occurs when the user clicks the control.
	OnCloseUp (see page 4099)	Occurs when the drop-down calendar closes.
	OnContextPopup (see page 4099)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 4100)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 4100)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4102)	Occurs when the user drags an object over a control.
	OnDropDown (see page 4105)	Occurs when the user opens the drop-down calendar by clicking the arrow at the right of the control.
	OnEndDock (see page 4105)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4105)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4106)	Occurs when a control receives the input focus.
	OnExit (see page 4109)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 4110)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 4113)	Occurs when a key is pressed.
	OnKeyUp (see page 4114)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 4116)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4116)	Occurs when the user moves the mouse outside of a control.
	OnStartDock (see page 4116)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4117)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUserInput (see page 4118)	Occurs when the user types directly into the component's edit box.

Published Properties

	Name	Description
	Align (see page 4066)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 4069)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 4070)	Specifies which edges of the control are beveled.
	BevelInner (see page 4070)	Specifies the cut of the inner bevel.
	BevelKind (see page 4071)	Specifies the control's bevel style.
	BevelOuter (see page 4071)	Specifies the cut of the outer bevel.
	BevelWidth (see page 4072)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 4072)	Specifies the bi-directional mode for the control.
	CalAlignment (see page 4073)	Determines the alignment of the drop-down calendar.
	CalColors (see page 4073)	Contains the colors used for each part of the calendar.
	Checked (see page 4073)	Indicates whether the check box next to the date or time is selected.
	Color (see page 4074)	Specifies the background color of the control.
	Constraints (see page 4074)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.

	Date (see page 4075)	Indicates the date that is marked on the calendar.
	DateFormat (see page 4076)	Specifies format in which the date is presented.
	DateMode (see page 4076)	Determines the method of date selection used by the component.
	DragCursor (see page 4077)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4077)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4077)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4079)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4080)	Controls the attributes of text written on or in the control.
	Format (see page 4081)	Specify format for date-time string.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 4082)	Determines the behavior of the input method editor (IME).
	ImeName (see page 4082)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Kind (see page 4083)	Determines whether the component is a date selector or a time selector.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	MaxDate (see page 4084)	Determines the latest allowable date that can be entered.
	MinDate (see page 4084)	Determines the earliest allowable date that can be entered.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 4084)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 4085)	Determines where a control looks for its color information.
	ParentFont (see page 4085)	Determines where a control looks for its font information.
	ParentShowHint (see page 4086)	Determines where a control looks to find out if its Help Hint should be shown.
	ParseInput (see page 4086)	This is ParseInput, a member of class TDateTimePicker.
	PopupMenu (see page 4086)	Identifies the pop-up menu associated with the control.
	ShowCheckbox (see page 4091)	Displays a check box next to the date or time.
	ShowHint (see page 4091)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 4093)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 4093)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Time (see page 4094)	Indicates the time entered by the user.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 4094)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.25.2 TDateTimePicker.Create Constructor

Creates and initializes a TDateTimePicker (see page 4049) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDateTimePicker(TComponent * AOwner);
```

Description

Create calls the inherited constructor method and initializes the Date (see page 4075) and Time (see page 4094) properties to the current date and time. It also sets the following ControlStyle flags: csCaptureMouse, csOpaque, csClickEvents, csDoubleClicks, and csFixedHeight.

See Also

ControlStyle (see page 5814)

Date (see page 3576)

Time (see page 4094)

Create (see page 6123)

27.1.25.3 TDateTimePicker Methods

The methods of the TDateTimePicker class are listed here.

Protected Methods

	Name	Description
	Change (see page 4061)	This is Change, a member of class TDateTimePicker.
V	CheckEmptyDate (see page 4061)	Clears the check box associated with the calendar.
V	CreateParams (see page 4062)	Initializes a window-creation parameter data structure.
V	CreateWnd (see page 4062)	Creates the windows control for the calendar.
V	GetCalendarHandle (see page 4063)	Returns the calendar control's window handle.
V	MsgSetCalColors (see page 4063)	Sets the calendar's colors.
V	MsgSetDateTime (see page 4064)	Sets the calendar's date/time.
V	MsgSetRange (see page 4064)	Implements the MinDate and MaxDate properties.

27.1.25.3.1 TDateTimePicker.Change Method

Pascal

```
procedure Change; dynamic;
```

C++

```
__fastcall Change();
```

Description

This is Change, a member of class TDateTimePicker.

27.1.25.3.2 TDateTimePicker.CheckEmptyDate Method

Clears the check box associated with the calendar.

Pascal

```
procedure CheckEmptyDate; override;
```

C++

```
virtual __fastcall CheckEmptyDate();
```

Description

CheckEmptyDate is called automatically when DateTime is assigned a value that represents a time and not a date value (or Date is assigned a value of 0). As implemented in TCommonCalendar, CheckEmptyDate does nothing.

TDateTimePicker overrides CheckEmptyDate to clear the check box associated with the calendar, or raise an EDateTimeError if the ShowCheckbox property is false. This allows TDateTimePicker to represent time values that do not include a date portion. (When a TDateTime value represents a time and not a date, the date portion is 0).

See Also

Checked (see page 4073)

ShowCheckbox (see page 4091)

Date (see page 3576)

DateTime (see page 3577)

27.1.25.3.3 TDateTimePicker.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

27.1.25.3.4 TDateTimePicker.CreateWnd Method

Creates the windows control for the calendar.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the windowed control is first created or when the window must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the window when it is first created.

CreateWnd calls the TWinControl CreateWnd method to generate the window, and then sends messages to initialize the control so that it reflects the current property settings.

See Also

CreateWnd (see page 6137)

27.1.25.3.5 TDateTimePicker.GetCalendarHandle Method

Returns the calendar control's window handle.

Pascal

```
function GetCalendarHandle: HWND; override;
```

C++

```
virtual __fastcall HWND GetCalendarHandle();
```

Description

GetCalendarHandle is the protected read implementation of the CalendarHandle property. For TCommonCalendar, this is an abstract method. It must be implemented in descendant classes.

See Also

CalendarHandle (see page 3576)

27.1.25.3.6 TDateTimePicker.MsgSetCalColors Method

Sets the calendar's colors.

Pascal

```
function MsgSetCalColors(ColorIndex: Integer; ColorValue: TColor): Boolean; override;
```

C++

```
virtual __fastcall Boolean MsgSetCalColors(int ColorIndex, TColor ColorValue);
```

Description

MsgSetCalColors implements the CalColors property by sending a message to the calendar window handle. MsgSetCalColors is an abstract method, and must be overridden in descendant objects to send the appropriate message.

The ColorIndex parameter indicates which of the TMonthCalColors properties is being set. It is one of the following values:

Value	Property
0	BackColor
1	TextColor
2	TitleBackColor
3	TitleTextColor
4	MonthBackColor
5	TrailingTextColor

The ColorValue parameter is the value that should be set.

MsgSetCalColors returns true if the color is set by the method.

See Also

CalColors (see page 3575)

CalendarHandle (see page 3576)

TextColor (see page 4395)

BackColor (see page 4395)

TitleBackColor (see page 4396)

TitleTextColor (see page 4396)

MonthBackColor (see page 4395)

TrailingTextColor (see page 4396)

27.1.25.3.7 TDateTimePicker.MsgSetDateTime Method

Sets the calendar's date/time.

Pascal

```
function MsgSetDateTime(Value: TSystemTime): Boolean; override;
```

C++

```
virtual __fastcall Boolean MsgSetDateTime(TSystemTime Value);
```

Description

MsgSetDateTime implements the DateTime property by sending a message to the calendar window handle. MsgSetDateTime is an abstract method, and must be overridden in descendant objects to send the appropriate message.

The Value parameter indicates the value of DateTime as a TSystemTime value. This can be obtained from the DateTime property value by calling DateTimeToSystemTime.

MsgSetDateTime returns true if the date/time value is set by the method.

See Also

DateTime (see page 3577)

CalendarHandle (see page 3576)

DateTimeToSystemTime (see page 22825)

SystemTimeToDateTime (see page 22974)

27.1.25.3.8 TDateTimePicker.MsgSetRange Method

Implements the MinDate and MaxDate properties.

Pascal

```
function MsgSetRange(Flags: Integer; SysTime: PSystemTime): Boolean; override;
```

C++

```
virtual __fastcall Boolean MsgSetRange(int Flags, PSystemTime SysTime);
```

Description

MsgSetRange implements the MinDate and MaxDate properties by sending a message to the calendar window handle. MsgSetRange is an abstract method, and must be overridden in descendant objects to send the appropriate message.

The Flags parameter indicates what type of information is included in the SysTime array. It includes the GDTR_MIN flag if a minimum date is supplied and GDTR_MAX if a maximum date is supplied.

The SysTime parameter is a two-element array containing the minimum date as the first element and the maximum date as the second element. These values can be obtained from the MinDate and MaxDate properties by calling DateTimeToSystemTime.

MsgSetRange returns true if the range is set by the method.

See Also

MinDate (see page 3579)

MaxDate (see page 3578)

CalendarHandle (see page 3576)

DateTimeToSystemTime (see page 22825)

SystemTimeToDateTime (see page 22974)

27.1.25.4 TDateTimePicker Properties

The properties of the TDateTimePicker class are listed here.

Public Properties

	Name	Description
	DateTime (see page 4076)	Indicates the date and time.
R	DroppedDown (see page 4079)	Specifies whether the date time picker control is dropped down.

Published Properties

	Name	Description
	Align (see page 4066)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4069)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 4070)	Specifies which edges of the control are beveled.
	BevelInner (see page 4070)	Specifies the cut of the inner bevel.
	BevelKind (see page 4071)	Specifies the control's bevel style.
	BevelOuter (see page 4071)	Specifies the cut of the outer bevel.
	BevelWidth (see page 4072)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 4072)	Specifies the bi-directional mode for the control.
	CalAlignment (see page 4073)	Determines the alignment of the drop-down calendar.
	CalColors (see page 4073)	Contains the colors used for each part of the calendar.
	Checked (see page 4073)	Indicates whether the check box next to the date or time is selected.
	Color (see page 4074)	Specifies the background color of the control.
	Constraints (see page 4074)	Specifies the size constraints for the control.
	Date (see page 4075)	Indicates the date that is marked on the calendar.
	DateFormat (see page 4076)	Specifies format in which the date is presented.
	DateMode (see page 4076)	Determines the method of date selection used by the component.
	DragCursor (see page 4077)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4077)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4077)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4079)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4080)	Controls the attributes of text written on or in the control.
	Format (see page 4081)	Specify format for date-time string.
	ImeMode (see page 4082)	Determines the behavior of the input method editor (IME).
	ImeName (see page 4082)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

	Kind (see page 4083)	Determines whether the component is a date selector or a time selector.
	MaxDate (see page 4084)	Determines the latest allowable date that can be entered.
	MinDate (see page 4084)	Determines the earliest allowable date that can be entered.
	ParentBiDiMode (see page 4084)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 4085)	Determines where a control looks for its color information.
	ParentFont (see page 4085)	Determines where a control looks for its font information.
	ParentShowHint (see page 4086)	Determines where a control looks to find out if its Help Hint should be shown.
	ParseInput (see page 4086)	This is ParseInput, a member of class TDateTimePicker.
	PopupMenu (see page 4086)	Identifies the pop-up menu associated with the control.
	ShowCheckbox (see page 4091)	Displays a check box next to the date or time.
	ShowHint (see page 4091)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 4093)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 4093)	Determines if the user can tab to a control.
	Time (see page 4094)	Indicates the time entered by the user.
	Visible (see page 4094)	Determines whether the component appears onscreen.

27.1.25.4.1 TDateTimePicker.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
```

```
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
```

```
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
```

```
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

27.1.25.4.2 TDateTimePicker.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.25.4.3 TDateTimePicker.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

27.1.25.4.4 TDateTimePicker.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

27.1.25.4.5 TDateTimePicker.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

27.1.25.4.6 TDateTimePicker.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

27.1.25.4.7 TDateTimePicker.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

27.1.25.4.8 TDateTimePicker.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

27.1.25.4.9 TDateTimePicker.CalAlignment Property

Determines the alignment of the drop-down calendar.

Pascal

```
property CalAlignment: TDTCalAlignment;
```

C++

```
__property TDTCalAlignment CalAlignment;
```

Description

Set CalAlignment to dtaLeft or dtaRight to align the drop-down calendar on the left or right side of the component. This property is applicable only if the Kind (see page 4083) property is dtkDate and the DateMode (see page 4076) property is dmComboBox.

See Also

DateMode (see page 4076)

Kind (see page 4083)

27.1.25.4.10 TDateTimePicker.CalColors Property

Contains the colors used for each part of the calendar.

Pascal

```
property CalColors: TMonthCalColors;
```

C++

```
__property TMonthCalColors CalColors;
```

Description

Use the CalColors property to get or set the calendar's background color, month background color, text color, title background color, title text color, and trailing text color.

See Also

BoldDays (see page 3570)

MsgSetCalColors (see page 3573)

27.1.25.4.11 TDateTimePicker.Checked Property

Indicates whether the check box next to the date or time is selected.

Pascal

```
property Checked: Boolean;
```

C++

```
__property Boolean Checked;
```

Description

Checked is true if the check box in the TDateTimePicker (see page 4049) component is selected. Setting Checked to true at design time makes the check box appear selected by default.

See Also

ShowCheckbox (see page 4091)

27.1.25.4.12 TDateTimePicker.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

27.1.25.4.13 TDateTimePicker.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.25.4.14 TDateTimePicker.Date Property

Indicates the date that is marked on the calendar.

Pascal

```
property Date: TDate;
```

C++

```
__property TDate Date;
```

Description

Use Date to get or set the date that is marked on the calendar. The value of Date must lie within the range specified by the MaxDate and MinDate properties.

If MultiSelect is true, the selected range of dates goes from Date to EndDate.

Note: You can also set the marked date with the DateTime property.

See Also

DateTime (see page 3577)

EndDate (see page 3577)

MaxDate (see page 3578)

MinDate (see page 3579)

MultiSelect (see page 3580)

27.1.25.4.15 TDateTimePicker.DateFormat Property

Specifies format in which the date is presented.

Pascal

```
property DateFormat: TDTDateFormat;
```

C++

```
__property TDTDateFormat DateFormat;
```

Description

If DateFormat is set to dfShort, the date is formatted using the short date format specified in the Windows control panel. By default, it appears like this: 3/21/97.

If DateFormat is set to dfLong, the date is formatted using the long date format specified in the Windows control panel. By default, it appears like this: Friday, March 21, 1997. (Applies only when Kind (see page 4083) is dtkDate.)

For custom date formats, set the Format (see page 4081) property.

See Also

Kind (see page 4083)

Format (see page 4081)

27.1.25.4.16 TDateTimePicker.DateMode Property

Determines the method of date selection used by the component.

Pascal

```
property DateMode: TDTDateMode;
```

C++

```
__property TDTDateMode DateMode;
```

Description

If DateMode is dmComboBox, the component has a drop-down calendar illustration from which the user can select a date. If DateMode is dmUpDown, the component has Up/Down arrows with which the user can adjust the date. (Applies only when Kind (see page 4083) is dtkDate.)

See Also

Kind (see page 4083)

27.1.25.4.17 TDateTimePicker.DateTime Property

Indicates the date and time.

Pascal

```
property DateTime: TDateTime;
```

C++

```
__property TDateTime DateTime;
```

Description

DateTime represents both a date and a time. If the Kind (see page 4083) property is switched at runtime, TDateTimePicker (

see page 4049) continues to track both the last date entered and the last time entered. If the Kind (see page 4083) property never changes, DateTime can be set programmatically (for example, by reading system date and time information) to indicate both a date and a time.

See Also

Kind (see page 4083)

Date (see page 3576)

Time (see page 4094)

27.1.25.4.18 TDateTimePicker.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

27.1.25.4.19 TDateTimePicker.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

27.1.25.4.20 TDateTimePicker.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
```

```
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

27.1.25.4.21 TDateTimePicker.DroppedDown Property

Specifies whether the date time picker control is dropped down.

Pascal

```
property DroppedDown: Boolean;
```

C++

```
__property Boolean DroppedDown;
```

Description

DroppedDown specifies whether the date time picker control is dropped down.

27.1.25.4.22 TDateTimePicker.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.25.4.23 TDateTimePicker.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
```

```
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

27.1.25.4.24 TDateTimePicker.Format Property

Specify format for date-time string.

Pascal

```
property Format: String;
```

C++

```
__property AnsiString Format;
```

Description

Format specifies custom format string for the date-time display, overriding the control panel strings. A custom format can include both date and time fields, but time fields are not editable unless the Kind (see page 4083) property is dtkTime. The following format characters are understood.

Element	Description
d	The one- or two-digit day.
dd	The two-digit day. Single-digit day values are preceded by a zero.
ddd	The three-character weekday abbreviation.
dddd	The full weekday name.
h	The one- or two-digit hour in 12-hour format.
hh	The two-digit hour in 12-hour format. Single-digit values are preceded by a zero.
H	The one- or two-digit hour in 24-hour format.
HH	The two-digit hour in 24-hour format. Single-digit values are preceded by a zero.
m	The one- or two-digit minute.
mm	The two-digit minute. Single-digit values are preceded by a zero.
M	The one- or two-digit month number.
MM	The two-digit month number. Single-digit values are preceded by a zero.
MMM	The three-character month abbreviation.
MMMM	The full month name.
t	The one-letter AM/PM abbreviation (that is, AM is displayed as "A").
tt	The two-letter AM/PM abbreviation (that is, AM is displayed as "AM").
yy	The last two digits of the year (that is, 2001 would be displayed as "01").
yyyy	The full year (that is, 2001 would be displayed as "2001").

To include literal strings in the format, enclose them in single quotes. Failing to quote literal strings can have unpredictable results, even if they do not include formatting codes. Use two single quotes to represent one single quote character in a literal string. For example,

'Today''s date:' MMM dd, yyyy

entered in the property editor, produces the following results for April 1, 2001:

"Today's date: Apr 01, 2001"

Note: When setting Format programmatically in Delphi code, double each single quote again:

```
DateTimePicker1.Format := '''Today''''s date:'' MMM dd, yyyy';
```

See Also

DateFormat (see page 4076)

DateMode (see page 4076)

27.1.25.4.25 TDateTimePicker.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

27.1.25.4.26 TDateTimePicker.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

27.1.25.4.27 TDateTimePicker.Kind Property

Determines whether the component is a date selector or a time selector.

Pascal

```
property Kind: TDateTimeKind;
```

C++

```
__property TDateTimeKind Kind;
```

Description

If the Kind property is dtkDate, the control is configured to support date editing. The default format gives only the date portion of the control value, and a drop-down calendar is available if the DateMode (see page 4076) property is dmComboxBox.

If the Kind property is dtkTime, the control is configured to support time editing. The default format give only the time portion of the control value, but the date portion can also be edited if the Format (see page 4081) property includes date field codes..

See Also

DateFormat (see page 4076)

DateMode (see page 4076)

Format (see page 4081)

27.1.25.4.28 TDateTimePicker.MaxDate Property

Determines the latest allowable date that can be entered.

Pascal

```
property MaxDate: TDate;
```

C++

```
__property TDate MaxDate;
```

Description

Assign a value to MaxDate to limit the range of dates that the user can enter. Out-of-range dates raise the EDateTimeError (see page 3421) exception. (Applies only when Kind (see page 4083) = dtkDate.)

See Also

EDateTimeError (see page 3421)

Kind (see page 4083)

MinDate (see page 4084)

27.1.25.4.29 TDateTimePicker.MinDate Property

Determines the earliest allowable date that can be entered.

Pascal

```
property MinDate: TDate;
```

C++

```
__property TDate MinDate;
```

Description

Assign a value to MinDate to limit the range of dates that the user can enter. Out-of-range dates raise the EDateTimeError (see page 3421) exception. (Applies only when Kind (see page 4083) = dtkDate.)

See Also

EDateTimeError (see page 3421)

Kind (see page 4083)

MaxDate (see page 4084)

27.1.25.4.30 TDateTimePicker.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

27.1.25.4.31 TDateTimePicker.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

27.1.25.4.32 TDateTimePicker.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

27.1.25.4.33 TDateTimePicker.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.25.4.34 TDateTimePicker.ParseInput Property

Pascal

```
property ParseInput: Boolean;
```

C++

```
__property Boolean ParseInput;
```

Description

This is ParseInput, a member of class TDateTimePicker.

27.1.25.4.35 TDateTimePicker.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
```

```
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
```

```
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
```

```
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
```

```
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.25.4.36 TDateTimePicker.ShowCheckbox Property

Displays a check box next to the date or time.

Pascal

```
property ShowCheckbox: Boolean;
```

C++

```
__property Boolean ShowCheckbox;
```

Description

Set ShowCheckbox to true to display a check box in the TDateTimePicker (see page 4049) component.

See Also

Checked (see page 4073)

27.1.25.4.37 TDateTimePicker.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
```

```
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.25.4.38 TDateTimePicker.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

27.1.25.4.39 TDateTimePicker.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

27.1.25.4.40 TDateTimePicker.Time Property

Indicates the time entered by the user.

Pascal

```
property Time: TTime;
```

C++

```
__property TTime Time;
```

Description

Time represents the time entered by the user. Set Time in program code or in the Object Inspector to determine which time is selected by default. (Applies only when Kind (see page 4083) = dtkTime.)

See Also

Date (see page 3576)

Kind (see page 4083)

DateTime (see page 4076)

27.1.25.4.41 TDateTimePicker.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
```

```
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
```

```
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.25.5 TDateTimePicker Events

The events of the TDateTimePicker class are listed here.

Published Events

	Name	Description
	OnChange (see page 4096)	Occurs when a date or time is entered.
	OnClick (see page 4097)	Occurs when the user clicks the control.
	OnCloseUp (see page 4099)	Occurs when the drop-down calendar closes.
	OnContextPopup (see page 4099)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 4100)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 4100)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4102)	Occurs when the user drags an object over a control.
	OnDropDown (see page 4105)	Occurs when the user opens the drop-down calendar by clicking the arrow at the right of the control.
	OnEndDock (see page 4105)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4105)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4106)	Occurs when a control receives the input focus.
	OnExit (see page 4109)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 4110)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 4113)	Occurs when a key is pressed.
	OnKeyUp (see page 4114)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 4116)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4116)	Occurs when the user moves the mouse outside of a control.
	OnStartDock (see page 4116)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4117)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUserInput (see page 4118)	Occurs when the user types directly into the component's edit box.

27.1.25.5.1 TDateTimePicker.OnChange Event

Occurs when a date or time is entered.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the date or time entered in the component changes.

See Also

Time (see page 4094)

DateTime (see page 3577)

Date (see page 3576)

27.1.25.5.2 TDateTimePicker.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

27.1.25.5.3 TDateTimePicker.OnCloseUp Event

Occurs when the drop-down calendar closes.

Pascal

```
property OnCloseUp: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCloseUp;
```

Description

Write an OnCloseUp event handler to implement special processing that needs to occur only when the drop-down calendar closes.

See Also

Kind (see page 4083)

DateMode (see page 4076)

OnDropDown (see page 4105)

27.1.25.5.4 TDateTimePicker.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.25.5.5 TDateTimePicker.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

27.1.25.5.6 TDateTimePicker.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

27.1.25.5.7 TDateTimePicker.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
```

```
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code comes from an application that contains a list box
  and three labels, each with a different font and color.  The
  DragMode property for each of the labels is dmAutomatic.
  The user can select a label and drag it to a list box and
  drop it. When the label is dropped, the items in the list
  box assume the color and font of the dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.25.5.8 TDateTimePicker.OnDropDown Event

Occurs when the user opens the drop-down calendar by clicking the arrow at the right of the control.

Pascal

```
property OnDropDown: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDropDown;
```

Description

Write an OnDropDown event handler to implement special processing that needs to occur only when the drop-down calendar is opened.

See Also

Kind (see page 4083)

DateMode (see page 4076)

OnCloseUp (see page 4099)

27.1.25.5.9 TDateTimePicker.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.25.5.10 TDateTimePicker.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

27.1.25.5.11 TDateTimePicker.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
```

```
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
```

```
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

27.1.25.5.12 TDateTimePicker.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

27.1.25.5.13 TDateTimePicker.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can

respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
```

```
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
```

```
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

27.1.25.5.14 TDateTimePicker.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the

OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

27.1.25.5.15 TDateTimePicker.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
```

```
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
```

```
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

27.1.25.5.16 TDateTimePicker.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

27.1.25.5.17 TDateTimePicker.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

27.1.25.5.18 TDateTimePicker.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

27.1.25.5.19 TDateTimePicker.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

27.1.25.5.20 TDateTimePicker.OnUserInput Event

Occurs when the user types directly into the component's edit box.

Pascal

```
property OnUserInput: TDTParseInputEvent;
```

C++

```
__property TDTParseInputEvent OnUserInput;
```

Description

This event occurs only if ParseInput (see page 4086) is set to true. Write an OnUserInput event handler implement special processing that needs to occur when the user types directly into the TDateTimePicker (see page 4049) (rather than selecting with the drop-down calendar or scroll arrows). UserString is the string that the user types, and DateAndTime represents the value of the Date (see page 4075) or Time (see page 4094) property.

See Also

Date (see page 3576)

ParseInput (see page 4086)

Time (see page 4094)

27.1.26 THeaderControl Class

THeaderControl is a container for THeaderSection (see page 4173) objects. It provides a set of resizable column headers.

Class Hierarchy

Pascal

```
THeaderControl = class(TCustomHeaderControl);
```

C++

```
class THeaderControl : public TCustomHeaderControl;
```

File

ComCtrls

Description

A THeaderControl is a set of column headers that the user can resize at runtime. Each header is represented by a THeaderSection (see page 4173) object listed in the Sections (see page 4147) property.

The header sections can be positioned above columns or fields of information. For example, header sections could be placed over a list box (TListBox). Because THeaderControl provides a generic header element that can be paired with virtually any other visual component, runtime manipulation of the header sections is not automatically extended to the components beneath it. If, for example, you want the column beneath a header section to be resized whenever the header section is resized, you will have to write a custom OnSectionResize (see page 4171) event handler to accomplish this.

See Also

THeaderSection (see page 4173)

THeaderSections (see page 4185)

TListBox (see page 21598)

Grouping Controls

Header Controls

27.1.26.1 THeaderControl Members

The following tables list the members exposed by THeaderControl.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.

	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 3715)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateSection (see page 3715)	This is CreateSection, a member of class TCustomHeaderControl.
V	CreateSections (see page 3715)	This is CreateSections, a member of class TCustomHeaderControl.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 3716)	Creates the underlying window.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.

	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 3716)	Destroys the control's window.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawSection (see page 3716)	This is DrawSection, a member of class TCustomHeaderControl.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.

	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 3717)	Responds to notifications that components are being created or destroyed.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.

	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SectionClick (see page 3717)	This is SectionClick, a member of class TCustomHeaderControl.
	SectionDrag (see page 3717)	This is SectionDrag, a member of class TCustomHeaderControl.
	SectionEndDrag (see page 3718)	This is SectionEndDrag, a member of class TCustomHeaderControl.
	SectionResize (see page 3718)	This is SectionResize, a member of class TCustomHeaderControl.
	SectionTrack (see page 3718)	This is SectionTrack, a member of class TCustomHeaderControl.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetImages (see page 3718)	This is SetImages, a member of class TCustomHeaderControl.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.

	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 3719)	Provides specific message responses for the control.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 3714)	Instantiate a TCustomHeaderControl (see page 3702) object.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 3714)	Destroy.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.

	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
V	FlipChildren (see page 3717)	Reverses the order in which sections are displayed.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.

	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 3728)	Provides runtime access to the header control's drawing surface.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.

R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnContextPopup (see page 4152)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreateSectionClass (see page 4153)	Runtime specification of section object class.
	OnDragDrop (see page 4153)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4155)	Occurs when the user drags an object over a control.
	OnDrawSection (see page 4158)	Occurs when a header section needs to be redrawn.
	OnEndDock (see page 4159)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4160)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

	OnMouseActivate (see page 4161)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4162)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4164)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4164)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4164)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4167)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 4169)	Occurs immediately after the control is resized.
	OnSectionClick (see page 4170)	Occurs when one of the header control's sections is clicked.
	OnSectionDrag (see page 4170)	Occurs when the user drags a header section to a new position.
	OnSectionEndDrag (see page 4170)	Occurs after the header control rearranges its sections in response to a drag operation.
	OnSectionResize (see page 4171)	Occurs when one of the header control's sections is resized.
	OnSectionTrack (see page 4171)	Occurs as one of the header control's sections is being resized.
	OnStartDock (see page 4172)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4172)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 4130)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 4134)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 4134)	Specifies the bi-directional mode for the control.
	BorderWidth (see page 4135)	Specifies the width of the control's border.
	Constraints (see page 4135)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 4136)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4136)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4137)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DragReorder (see page 4138)	Make header reorderable.
	Enabled (see page 4138)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4139)	Controls the attributes of text written on or in the control.
	FullDrag (see page 4140)	Specifies whether headers are repainted when they are dragged.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HotTrack (see page 4140)	Highlights the header sections as the mouse passes over them.
	Images (see page 4141)	Specifies header images
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 4141)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 4141)	Determines where a control looks for its font information.
	ParentShowHint (see page 4142)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4142)	Identifies the pop-up menu associated with the control.
	Sections (see page 4147)	Lists the header sections (column headings).

	ShowHint (see page 4148)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 4150)	Specifies the header control's style.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 4150)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.26.2 THeaderControl Properties

The properties of the THeaderControl class are listed here.

Published Properties

	Name	Description
	Align (see page 4130)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4134)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 4134)	Specifies the bi-directional mode for the control.
	BorderWidth (see page 4135)	Specifies the width of the control's border.
	Constraints (see page 4135)	Specifies the size constraints for the control.
	DragCursor (see page 4136)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4136)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4137)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DragReorder (see page 4138)	Make header reorderable.
	Enabled (see page 4138)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4139)	Controls the attributes of text written on or in the control.
	FullDrag (see page 4140)	Specifies whether headers are repainted when they are dragged.
	HotTrack (see page 4140)	Highlights the header sections as the mouse passes over them.
	Images (see page 4141)	Specifies header images
	ParentBiDiMode (see page 4141)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 4141)	Determines where a control looks for its font information.
	ParentShowHint (see page 4142)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4142)	Identifies the pop-up menu associated with the control.
	Sections (see page 4147)	Lists the header sections (column headings).
	ShowHint (see page 4148)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 4150)	Specifies the header control's style.
	Visible (see page 4150)	Determines whether the component appears onscreen.

27.1.26.2.1 THeaderControl.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
```

```
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;
```

```
procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

27.1.26.2.2 THeaderControl.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.26.2.3 THeaderControl.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

27.1.26.2.4 THeaderControl.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics or text drawn by the control is clipped to the area within the border.

See Also

ClientRect

BevelOuter (see page 6194)

27.1.26.2.5 THeaderControl.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.26.2.6 THeaderControl.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

27.1.26.2.7 THeaderControl.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

27.1.26.2.8 THeaderControl.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
```

```
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

27.1.26.2.9 THeaderControl.DragReorder Property

Make header reorderable.

Pascal

```
property DragReorder: Boolean;
```

C++

```
__property Boolean DragReorder;
```

Description

If DragReorder is true (default is false), the user can rearrange headers by dragging them.

See Also

OnSectionDrag (see page 3737)

OnSectionEndDrag (see page 3738)

27.1.26.2.10 THeaderControl.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.26.2.11 THeaderControl.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
```

```
  is loaded into the combo box. When the combo box is clicked,
  the memo or Rich Edit control font is set to the
  corresponding font name in the combo box.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    i : Integer;
  begin
    for i := 0 to Screen.Fonts.Count - 1 do
     ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
  //  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
    RichEdit1.Lines.Text := 'How is it to be you?';
  end;

  procedure TForm1.ComboBox1Click(Sender: TObject);
  begin
    RichEdit1.Font.Name := ComboBox1.Text;
  end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

27.1.26.2.12 THeaderControl.FullDrag Property

Specifies whether headers are repainted when they are dragged.

Pascal

```
property FullDrag: Boolean;
```

C++

```
__property Boolean FullDrag;
```

Description

Use FullDrag to specify whether headers are repainted while they are dragged.

See Also

OnSectionDrag (see page 3737)

27.1.26.2.13 THeaderControl.HotTrack Property

Highlights the header sections as the mouse passes over them.

Pascal

```
property HotTrack: Boolean;
```

C++

```
__property Boolean HotTrack;
```

Description

If HotTrack is set to true, the text on each header section appears highlighted when the mouse passes over it at runtime.

See Also

Text (see page 4183)

27.1.26.2.14 THeaderControl.Images Property

Specifies header images

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Images contains a list of images that can be used as icons. Each section's ImageIndex property determines the specific image displayed on it. If Images is unassigned, items are displayed as captions with no icons.

See Also

ImageIndex

27.1.26.2.15 THeaderControl.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

27.1.26.2.16 THeaderControl.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform

appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

27.1.26.2.17 THeaderControl.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.26.2.18 THeaderControl.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse

button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
```

```
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
```

```
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
```

```
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
```

```
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.26.2.19 THeaderControl.Sections Property

Lists the header sections (column headings).

Pascal

```
property Sections: THeaderSections;
```

C++

```
__property THeaderSections Sections;
```

Description

The Sections property holds a THeaderSections (see page 4185)—that is, a collection of THeaderSection (see page 4173) objects. At design time, header sections can be added, removed, or modified with the Sections editor. To open the Sections editor, select the Sections property in the Object Inspector, then double-click in the Value column to the right or click the ellipsis (...) button.

C++ Examples:

```
/*
Add a THeaderControl to the form and call
HeaderControl1DrawSection for the OnDrawSection event.
*/
void __fastcall TForm1::HeaderControl1DrawSection(THeaderControl *HeaderControl,
      THeaderSection *Section, const TRect &Rect, bool Pressed)
{
  if (Pressed)
    HeaderControl->Canvas->Font->Color = clRed;
  else
    HeaderControl->Canvas->Font->Color = clBlue;
  HeaderControl->Canvas->TextOut(Rect.Left + HeaderControl->Canvas->Font->Size, Rect.Top + 2,
"Custom " + IntToStr(Section->Index));
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  THeaderSection *Section;
  for (int i = 0; i <= 4; i++)
  {
    Section = HeaderControl1->Sections->Add();
```

```
      Section->Text = "Section " + IntToStr(i);
      Section->MinWidth =
        (Section->Text).Length() *
        HeaderControl1->Font->Size;
      // Owner draw every other section
      if (i % 2 == 0)
        Section->Style = hsOwnerDraw;
      else
        Section->Style = hsText;
    }
  }
```

Delphi Examples:

```
{
Add a THeaderControl to the form and call
HeaderControl1DrawSection for the OnDrawSection event.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  HeaderSection: THeaderSection;
  I: Integer;
begin
  for I := 0 to 4 do
  begin
    HeaderSection := HeaderControl1.Sections.Add;
    HeaderSection.Text := 'Text Section ' + IntToStr(I);
    HeaderSection.MinWidth := length(HeaderSection.Text) * Font.Size;
    // Owner draw every other section
    if (I mod 2 = 0) then
      HeaderSection.Style := hsOwnerDraw
    else
      HeaderSection.Style := hsText;
  end;
end;

procedure TForm1.HeaderControl1DrawSection(HeaderControl: THeaderControl;
  Section: THeaderSection; const Rect: TRect; Pressed: Boolean);
begin
  with HeaderControl.Canvas do
  begin
    // highlight pressed sections
    if Pressed then
      Font.Color := clRed
    else
      Font.Color := clBlue;
    TextOut(Rect.Left + Font.Size, Rect.Top + 2, 'Owner Drawn text');
  end;
end;
```

See Also

THeaderSection (see page 4173)

27.1.26.2.20 THeaderControl.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
```

```
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.26.2.21 THeaderControl.Style Property

Specifies the header control's style.

Pascal

```
property Style: THeaderStyle;
```

C++

```
__property THeaderStyle Style;
```

Description

Use Style to determine the appearance of the header sections.

27.1.26.2.22 THeaderControl.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
```

```
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.26.3 THeaderControl Events

The events of the THeaderControl class are listed here.

Published Events

	Name	Description
	OnContextPopup (see page 4152)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreateSectionClass (see page 4153)	Runtime specification of section object class.
	OnDragDrop (see page 4153)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4155)	Occurs when the user drags an object over a control.
	OnDrawSection (see page 4158)	Occurs when a header section needs to be redrawn.
	OnEndDock (see page 4159)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4160)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 4161)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4162)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4164)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4164)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4164)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4167)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 4169)	Occurs immediately after the control is resized.
	OnSectionClick (see page 4170)	Occurs when one of the header control's sections is clicked.
	OnSectionDrag (see page 4170)	Occurs when the user drags a header section to a new position.
	OnSectionEndDrag (see page 4170)	Occurs after the header control rearranges its sections in response to a drag operation.
	OnSectionResize (see page 4171)	Occurs when one of the header control's sections is resized.
	OnSectionTrack (see page 4171)	Occurs as one of the header control's sections is being resized.
	OnStartDock (see page 4172)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4172)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

27.1.26.3.1 THeaderControl.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to

override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.26.3.2 THeaderControl.OnCreateSectionClass Event

Runtime specification of section object class.

Pascal

```
property OnCreateSectionClass: TCustomHCCreateSectionClassEvent;
```

C++

```
__property TCustomHCCreateSectionClassEvent OnCreateSectionClass;
```

Description

The OnCreateSectionClass event occurs when a new header section object is about to be created. If the event handler exists, it is called with SectionClass set to THeaderSection. The event handler can modify this value to any descendant of THeaderSection, which will then be used to create the header section object.

27.1.26.3.3 THeaderControl.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
```

```
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

27.1.26.3.4 THeaderControl.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.26.3.5 THeaderControl.OnDrawSection Event

Occurs when a header section needs to be redrawn.

Pascal

```
property OnDrawSection: TDrawSectionEvent;
```

C++

```
__property TDrawSectionEvent OnDrawSection;
```

Description

The OnDrawSection event occurs when a header section needs to be redisplayed—for example, when the user selects or resizes a section. OnDrawSection occurs only if the header section's Style (see page 4150) property is set to hsOwnerDraw.

C++ Examples:

```
/*
Add a THeaderControl to the form and call
HeaderControl1DrawSection for the OnDrawSection event.
*/
void __fastcall TForm1::HeaderControl1DrawSection(THeaderControl *HeaderControl,
      THeaderSection *Section, const TRect &Rect, bool Pressed)
{
  if (Pressed)
    HeaderControl->Canvas->Font->Color = clRed;
  else
    HeaderControl->Canvas->Font->Color = clBlue;
  HeaderControl->Canvas->TextOut(Rect.Left + HeaderControl->Canvas->Font->Size, Rect.Top + 2,
"Custom " + IntToStr(Section->Index));
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  THeaderSection *Section;
  for (int i = 0; i <= 4; i++)
  {
    Section = HeaderControl1->Sections->Add();
    Section->Text = "Section " + IntToStr(i);
    Section->MinWidth =
      (Section->Text).Length() *
      HeaderControl1->Font->Size;
    // Owner draw every other section
    if (i % 2 == 0)
```

```
      Section->Style = hsOwnerDraw;
    else
      Section->Style = hsText;
  }
}
```

Delphi Examples:

```
{
Add a THeaderControl to the form and call
HeaderControl1DrawSection for the OnDrawSection event.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  HeaderSection: THeaderSection;
  I: Integer;
begin
  for I := 0 to 4 do
  begin
    HeaderSection := HeaderControl1.Sections.Add;
    HeaderSection.Text := 'Text Section ' + IntToStr(I);
    HeaderSection.MinWidth := length(HeaderSection.Text) * Font.Size;
    // Owner draw every other section
    if (I mod 2 = 0) then
      HeaderSection.Style := hsOwnerDraw
    else
      HeaderSection.Style := hsText;
  end;
end;

procedure TForm1.HeaderControl1DrawSection(HeaderControl: THeaderControl;
  Section: THeaderSection; const Rect: TRect; Pressed: Boolean);
begin
  with HeaderControl.Canvas do
  begin
    // highlight pressed sections
    if Pressed then
      Font.Color := clRed
    else
      Font.Color := clBlue;
    TextOut(Rect.Left + Font.Size, Rect.Top + 2, 'Owner Drawn text');
  end;
end;
```

See Also

Style (see page 4181)

27.1.26.3.6 THeaderControl.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.26.3.7 THeaderControl.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

27.1.26.3.8 THeaderControl.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.26.3.9 THeaderControl.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
```

```
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
```

```
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.26.3.10 THeaderControl.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

27.1.26.3.11 THeaderControl.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

27.1.26.3.12 THeaderControl.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
```

```
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.26.3.13 THeaderControl.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
```

```
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
```

```
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

27.1.26.3.14 THeaderControl.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

27.1.26.3.15 THeaderControl.OnSectionClick Event

Occurs when one of the header control's sections is clicked.

Pascal

```
property OnSectionClick: TSectionNotifyEvent;
```

C++

```
__property TSectionNotifyEvent OnSectionClick;
```

Description

The OnSectionClick event occurs when a header section is clicked at runtime. To disable a section from being clicked set its AllowClick property to false.

See Also

AllowClick (see page 4178)

27.1.26.3.16 THeaderControl.OnSectionDrag Event

Occurs when the user drags a header section to a new position.

Pascal

```
property OnSectionDrag: TSectionDragEvent;
```

C++

```
__property TSectionDragEvent OnSectionDrag;
```

Description

Write an OnSectionDrag event handler to respond when a user tries to reorder the sections in the header control. The OnSectionDrag event handler can conditionally prevent the user from dragging a header to a new position.

If the AllowDrag parameter is true on exit from the event handler, the header control reorders the sections and generates an OnSectionEndDrag (see page 4170) event when the reorder is complete.

OnSectionDrag only occurs if the DragReorder (see page 4138) property is true.

See Also

DragReorder (see page 3733)

Sections (see page 4147)

OnSectionEndDrag (see page 4170)

27.1.26.3.17 THeaderControl.OnSectionEndDrag Event

Occurs after the header control rearranges its sections in response to a drag operation.

Pascal

```
property OnSectionEndDrag: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSectionEndDrag;
```

Description

Write an OnSectionEndDrag event handler to respond after a user reorders the sections in the header control.

OnSectionEndDrag only occurs if the DragReorder (see page 4138) property is true.

See Also

DragReorder (see page 3733)

Sections (see page 4147)

OnSectionDrag (see page 4170)

27.1.26.3.18 THeaderControl.OnSectionResize Event

Occurs when one of the header control's sections is resized.

Pascal

```
property OnSectionResize: TSectionNotifyEvent;
```

C++

```
__property TSectionNotifyEvent OnSectionResize;
```

Description

The OnSectionResize event occurs when a header section is resized at runtime. This happens when the user positions the mouse pointer between two sections and drags their border to the right or left.

See Also

OnSectionTrack (see page 4171)

27.1.26.3.19 THeaderControl.OnSectionTrack Event

Occurs as one of the header control's sections is being resized.

Pascal

```
property OnSectionTrack: TSectionTrackEvent;
```

C++

```
__property TSectionTrackEvent OnSectionTrack;
```

Description

The OnSectionTrack event tracks the dragging of a header section's border as it happens. OnSectionTrack occurs when the mouse pointer is positioned between two header sections and the left mouse button is depressed.

OnSectionTrack is of type TSectionTrackEvent (see page 5506), whose Width and State parameters indicate the size of the header section (in pixels) and the status of the event. The State property can take three values:

Value	Meaning
tsTrackBegin	The border has not yet been dragged.
tsTrackMove	The border is being dragged.
tsTrackEnd	The border is no longer being dragged.

Use OnSectionTrack when you want something to happen at runtime as a header section is being resized.

See Also

OnSectionResize (see page 4171)

27.1.26.3.20 THeaderControl.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

27.1.26.3.21 THeaderControl.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

27.1.27 THeaderSection Class

THeaderSection represents a section of a header control.

Class Hierarchy

Pascal

```
THeaderSection = class(TCollectionItem);
```

C++

```
class THeaderSection : public TCollectionItem;
```

File

ComCtrls

Description

THeaderControl (see page 4118) objects use THeaderSection objects to maintain collections of THeaderSection objects. These THeaderSection objects represent resizable column headers. Each column header includes a text label and possibly a graphic image.

See Also

TCollection (see page 2731)

TCollectionItem (see page 2755)

THeaderSections (see page 4185)

THeaderControl (see page 4118)

27.1.27.1 THeaderSection Members

The following tables list the members exposed by THeaderSection.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 4176)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 4176)	Copies the contents of the source header section to a new header section.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 4175)	Creates and initializes a header section.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentBiDiModeChanged (see page 4176)	Synchronizes the header section's BiDiMode (see page 4179) with the associated header control's BiDiMode (see page 4179).
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	UseRightToLeftAlignment (see page 4177)	Specifies whether the control's alignment is in a right-to-left mode.
	UseRightToLeftReading (see page 4177)	Specifies whether the control is using a right-to-left reading order.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
R	Left (see page 4180)	Specifies the horizontal position of the header section's left edge.
R	Right (see page 4181)	Specifies the horizontal position of the header section's right edge.

Published Properties

	Name	Description
	Alignment (see page 4178)	Specifies how text is aligned within the header section.
	AllowClick (see page 4178)	Allows the section to respond to mouse clicks at runtime.
	AutoSize (see page 4179)	Indicates whether the header section should size itself to exactly hold its contents.
	BiDiMode (see page 4179)	Specifies the bi-directional mode for the header section.
	ImageIndex (see page 4179)	Identifies the image that appears next to the text on the header section.
	MaxWidth (see page 4180)	Specifies the maximum width of the header section.
	MinWidth (see page 4180)	Specifies the minimum width of the header section.
	ParentBiDiMode (see page 4181)	Specifies whether the header section uses its parent's bi-directional mode.
	Style (see page 4181)	Determines how the header section's text is displayed.
	Text (see page 4183)	Specifies the header section's caption.
	Width (see page 4184)	Specifies the width of the header section.

27.1.27.2 THeaderSection.Create Constructor

Creates and initializes a header section.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall THeaderSection(TCollection Collection);
```

Description

Do not instantiate a THeaderSection (see page 4173) object. Header sections are created automatically by the THeaderSections (see page 4185) objects that use them. To add a section to a THeaderSections (see page 4185) object, use its Add method instead.

Create takes a THeaderSections (see page 4185) instance as its argument.

See Also

THeaderSections (see page 4185)

Add (see page 4188)

27.1.27.3 THeaderSection Methods

The methods of the THeaderSection class are listed here.

Protected Methods

	Name	Description
V	GetDisplayName (see page 4176)	Returns the name of the collection item as it appears in the collection editor.

Public Methods

	Name	Description
	Assign (see page 4176)	Copies the contents of the source header section to a new header section.
	ParentBiDiModeChanged (see page 4176)	Synchronizes the header section's BiDiMode (see page 4179) with the associated header control's BiDiMode (see page 4179).
	UseRightToLeftAlignment (see page 4177)	Specifies whether the control's alignment is in a right-to-left mode.
	UseRightToLeftReading (see page 4177)	Specifies whether the control is using a right-to-left reading order.

27.1.27.3.1 THeaderSection.Assign Method

Copies the contents of the source header section to a new header section.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Assign copies the values of the source object's Alignment (see page 4178), AllowClick (see page 4178), MaxWidth (see page 4180), MinWidth (see page 4180), Style (see page 4181), Text (see page 4183), and Width (see page 4184) properties to another header section.

See Also

Assign (see page 2930)

27.1.27.3.2 THeaderSection.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName (see page 2761)

GetNamePath (see page 2759)

27.1.27.3.3 THeaderSection.ParentBiDiModeChanged Method

Synchronizes the header section's BiDiMode (see page 4179) with the associated header control's BiDiMode (see page 4179).

Pascal

```
procedure ParentBiDiModeChanged;
```

C++

```
__fastcall ParentBiDiModeChanged();
```

Description

Use ParentBiDiModeChanged to synchronize the header section's BiDiMode (see page 4179) with the associated header control's BiDiMode (see page 4179).

27.1.27.3.4 THeaderSection.UseRightToLeftAlignment Method

Specifies whether the control's alignment is in a right-to-left mode.

Pascal

```
function UseRightToLeftAlignment: Boolean;
```

C++

```
__fastcall Boolean UseRightToLeftAlignment();
```

Description

Call UseRightToLeftAlignment to determine whether the control's alignment is in a right-to-left mode. This information can be used when painting the control, or when responding to mouse events.

UseRightToLeft alignment returns true if the MiddleEast field of the global SysLocale is true and the BiDiMode (see page 4179) property is bdRightToLeft.

See Also

BiDiMode (see page 5803)

UseRightToLeftAlignment (see page 5790)

variables

TSysLocale (see page 23008)

27.1.27.3.5 THeaderSection.UseRightToLeftReading Method

Specifies whether the control is using a right-to-left reading order.

Pascal

```
function UseRightToLeftReading: Boolean;
```

C++

```
__fastcall Boolean UseRightToLeftReading();
```

Description

Call UseRightToLeftReading to determine which style flags to set for the control when it is created and which flags to set for drawing text (for example, when calling ExTextOut or DrawText).

UseRightToLeftReading returns true if the MiddleEast field of the global SysLocale is true and the BiDiMode (see page 4179) property is not bdRightToLeft.

See Also

BiDiMode (see page 5803)

UseRightToLeftReading (see page 5790)

27.1.27.4 THeaderSection Properties

The properties of the THeaderSection class are listed here.

Public Properties

	Name	Description
R	Left (see page 4180)	Specifies the horizontal position of the header section's left edge.
R	Right (see page 4181)	Specifies the horizontal position of the header section's right edge.

Published Properties

	Name	Description
	Alignment (see page 4178)	Specifies how text is aligned within the header section.
	AllowClick (see page 4178)	Allows the section to respond to mouse clicks at runtime.
	AutoSize (see page 4179)	Indicates whether the header section should size itself to exactly hold its contents.
	BiDiMode (see page 4179)	Specifies the bi-directional mode for the header section.
	ImageIndex (see page 4179)	Identifies the image that appears next to the text on the header section.
	MaxWidth (see page 4180)	Specifies the maximum width of the header section.
	MinWidth (see page 4180)	Specifies the minimum width of the header section.
	ParentBiDiMode (see page 4181)	Specifies whether the header section uses its parent's bi-directional mode.
	Style (see page 4181)	Determines how the header section's text is displayed.
	Text (see page 4183)	Specifies the header section's caption.
	Width (see page 4184)	Specifies the width of the header section.

27.1.27.4.1 THeaderSection.Alignment Property

Specifies how text is aligned within the header section.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to specify whether the header section text should be left-justified, right-justified, or centered.

These are the possible values of Alignment:

Value	Meaning
taLeftJustify	Align text on the left side of the header section.
taCenter	Center text in the header section.
taRightJustify	Align text on the right side of the header section.

See Also

Text (see page 4183)

Style (see page 4181)

27.1.27.4.2 THeaderSection.AllowClick Property

Allows the section to respond to mouse clicks at runtime.

Pascal

```
property AllowClick: Boolean;
```

C++

```
__property Boolean AllowClick;
```

Description

If AllowClick is set to true (the default), the header section can be clicked with the mouse at runtime. Header sections behave like buttons when clicked: They loose their raised borders and appear depressed on the form. To attach functionality to the clicking of header sections, write an event handler for the header control's OnSectionClick event.

See Also

OnSectionClick (see page 4170)

27.1.27.4.3 THeaderSection.AutoSize Property

Indicates whether the header section should size itself to exactly hold its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

When AutoSize is true, the header section (and the header control that contains it) automatically resizes to adjust to changes. When AutoSize is false, the header section size is fixed unless changed programmatically.

27.1.27.4.4 THeaderSection.BiDiMode Property

Specifies the bi-directional mode for the header section.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to specify the bi-directional mode for the header section.

27.1.27.4.5 THeaderSection.ImageIndex Property

Identifies the image that appears next to the text on the header section.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

Use ImageIndex to specify an image that appears next to the text on the header section. ImageIndex is the (0-offset) index of the

image in the TCustomImageList component that is available through the header control's Images property.

See Also

Images (see page 3735)

Text (see page 4183)

27.1.27.4.6 THeaderSection.Left Property

Specifies the horizontal position of the header section's left edge.

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

The Left property is the horizontal coordinate of the header section's left edge, relative to the left border of the header control and measured in pixels.

See Also

Right (see page 4181)

Width (see page 4184)

27.1.27.4.7 THeaderSection.MaxWidth Property

Specifies the maximum width of the header section.

Pascal

```
property MaxWidth: Integer;
```

C++

```
__property int MaxWidth;
```

Description

Sets the maximum width, in pixels, for the header section. If MaxWidth = MinWidth (see page 4180), the header section cannot be resized at runtime.

See Also

Width (see page 4184)

MinWidth (see page 4180)

27.1.27.4.8 THeaderSection.MinWidth Property

Specifies the minimum width of the header section.

Pascal

```
property MinWidth: Integer;
```

C++

```
__property int MinWidth;
```

Description

Sets the minimum width, in pixels, for the header section. If MinWidth = MaxWidth (see page 4180), the header section cannot be resized at runtime.

See Also

Width (see page 4184)

MaxWidth (see page 4180)

27.1.27.4.9 THeaderSection.ParentBiDiMode Property

Specifies whether the header section uses its parent's bi-directional mode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use BiDiMode (see page 4179) to specify whether the header section uses its parent's bi-directional mode.

27.1.27.4.10 THeaderSection.Right Property

Specifies the horizontal position of the header section's right edge.

Pascal

```
property Right: Integer;
```

C++

```
__property int Right;
```

Description

The Right property is the horizontal coordinate of the header section's right edge, relative to the left border of the header control and measured in pixels. The Right value is the value of Left (see page 4180) plus the value of Width (see page 4184).

See Also

Left (see page 4180)

Width (see page 4184)

27.1.27.4.11 THeaderSection.Style Property

Determines how the header section's text is displayed.

Pascal

```
property Style: THeaderSectionStyle;
```

C++

```
__property THeaderSectionStyle Style;
```

Description

If Style is set to hsText (the default), the string contained in the Text (see page 4183) property is displayed in the header section, using the alignment specified by Alignment (see page 4178). The font is determined by the header control's Font

property. A graphic image can appear beside the text if the ImageIndex (see page 4179) property is set.

If Style is set to hsOwnerDraw, the content displayed in the header section is drawn at run time on the header control's canvas by code in an OnDrawSection event handler.

C++ Examples:

```
/*
Add a THeaderControl to the form and call
HeaderControl1DrawSection for the OnDrawSection event.
*/
void __fastcall TForm1::HeaderControl1DrawSection(THeaderControl *HeaderControl,
      THeaderSection *Section, const TRect &Rect, bool Pressed)
{
  if (Pressed)
    HeaderControl->Canvas->Font->Color = clRed;
  else
    HeaderControl->Canvas->Font->Color = clBlue;
  HeaderControl->Canvas->TextOut(Rect.Left + HeaderControl->Canvas->Font->Size, Rect.Top + 2,
"Custom " + IntToStr(Section->Index));
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  THeaderSection *Section;
  for (int i = 0; i <= 4; i++)
  {
    Section = HeaderControl1->Sections->Add();
    Section->Text = "Section " + IntToStr(i);
    Section->MinWidth =
      (Section->Text).Length() *
      HeaderControl1->Font->Size;
    // Owner draw every other section
    if (i % 2 == 0)
      Section->Style = hsOwnerDraw;
    else
      Section->Style = hsText;
  }
}
```

Delphi Examples:

```
{
Add a THeaderControl to the form and call
HeaderControl1DrawSection for the OnDrawSection event.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  HeaderSection: THeaderSection;
  I: Integer;
begin
  for I := 0 to 4 do
  begin
    HeaderSection := HeaderControl1.Sections.Add;
    HeaderSection.Text := 'Text Section ' + IntToStr(I);
    HeaderSection.MinWidth := length(HeaderSection.Text) * Font.Size;
    // Owner draw every other section
    if (I mod 2 = 0) then
      HeaderSection.Style := hsOwnerDraw
    else
      HeaderSection.Style := hsText;
  end;
end;

procedure TForm1.HeaderControl1DrawSection(HeaderControl: THeaderControl;
  Section: THeaderSection; const Rect: TRect; Pressed: Boolean);
begin
```

```
    with HeaderControl.Canvas do
    begin
      // highlight pressed sections
      if Pressed then
        Font.Color := clRed
      else
        Font.Color := clBlue;
      TextOut(Rect.Left + Font.Size, Rect.Top + 2, 'Owner Drawn text');
    end;
  end;
```

See Also

Text (see page 4183)

Alignment (see page 4178)

Font (see page 5823)

Canvas (see page 3728)

OnDrawSection (see page 4158)

ImageIndex (see page 4179)

27.1.27.4.12 THeaderSection.Text Property

Specifies the header section's caption.

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

The Text property contains a string that identifies the header section or the column beneath it. If Style (see page 4181) is set to hsText, the value of Text appears in the header section.

C++ Examples:

```
/*
Add a THeaderControl to the form and call
HeaderControl1DrawSection for the OnDrawSection event.
*/
void __fastcall TForm1::HeaderControl1DrawSection(THeaderControl *HeaderControl,
      THeaderSection *Section, const TRect &Rect, bool Pressed)
{
  if (Pressed)
    HeaderControl->Canvas->Font->Color = clRed;
  else
    HeaderControl->Canvas->Font->Color = clBlue;
  HeaderControl->Canvas->TextOut(Rect.Left + HeaderControl->Canvas->Font->Size, Rect.Top + 2,
"Custom " + IntToStr(Section->Index));
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  THeaderSection *Section;
  for (int i = 0; i <= 4; i++)
  {
    Section = HeaderControl1->Sections->Add();
    Section->Text = "Section " + IntToStr(i);
    Section->MinWidth =
```

```
        (Section->Text).Length() *
        HeaderControl1->Font->Size;
      // Owner draw every other section
      if (i % 2 == 0)
        Section->Style = hsOwnerDraw;
      else
        Section->Style = hsText;
    }
  }
```

Delphi Examples:

```
{
Add a THeaderControl to the form and call
HeaderControl1DrawSection for the OnDrawSection event.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  HeaderSection: THeaderSection;
  I: Integer;
begin
  for I := 0 to 4 do
  begin
    HeaderSection := HeaderControl1.Sections.Add;
    HeaderSection.Text := 'Text Section ' + IntToStr(I);
    HeaderSection.MinWidth := length(HeaderSection.Text) * Font.Size;
    // Owner draw every other section
    if (I mod 2 = 0) then
      HeaderSection.Style := hsOwnerDraw
    else
      HeaderSection.Style := hsText;
  end;
end;

procedure TForm1.HeaderControl1DrawSection(HeaderControl: THeaderControl;
  Section: THeaderSection; const Rect: TRect; Pressed: Boolean);
begin
  with HeaderControl.Canvas do
  begin
    // highlight pressed sections
    if Pressed then
      Font.Color := clRed
    else
      Font.Color := clBlue;
    TextOut(Rect.Left + Font.Size, Rect.Top + 2, 'Owner Drawn text');
  end;
end;
```

See Also

Style (see page 4181)

Alignment (see page 4178)

27.1.27.4.13 THeaderSection.Width Property

Specifies the width of the header section.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

The Width property determines the default width of the header section, in pixels. Header sections can be resized at runtime by dragging their borders.

See Also

Left (see page 4180)

Right (see page 4181)

MaxWidth (see page 4180)

MinWidth (see page 4180)

27.1.28 THeaderSections Class

THeaderSections is a container for THeaderSection (see page 4173) objects.

Class Hierarchy

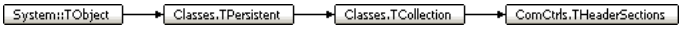

Pascal

```
THeaderSections = class(TCollection);
```

C++

```
class THeaderSections : public TCollection;
```

File

ComCtrls

Description

Each THeaderSections objects holds a collection of THeaderSection (see page 4173) objects displayed in a specific THeaderControl (see page 4118) object. THeaderSections object maintains an index of the header sections in its Items (see page 4190) array. The Count property contains the number of header sections in the collection. At design time, use the header control's Sections editor to add, remove, or modify header sections.

See Also

TCollection (see page 2731)

TCollectionItem (see page 2755)

THeaderSection (see page 4173)

THeaderControl (see page 4118)

THeaderControl (see page 4118)

27.1.28.1 THeaderSections Members

The following tables list the members exposed by THeaderSections.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 4188)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 4189)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 4188)	Creates a new THeaderSection (see page 4173) instance and adds it to the Items (see page 4190) array.
	AddItem (see page 4188)	Add (see page 4188) section object to collection.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 4187)	Creates and initializes a THeaderSections (see page 4185) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 4189)	Add (see page 4188) new section object to collection.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 4190)	Lists the header sections in the collection.

27.1.28.2 THeaderSections.Create Constructor

Creates and initializes a THeaderSections (see page 4185) object.

Pascal

```
constructor Create(HeaderControl: TCustomHeaderControl);
```

C++

```
__fastcall THeaderSections(TCustomHeaderControl HeaderControl);
```

Description

The Create method takes a THeaderControl (see page 4118) instance as a parameter.

See Also

THeaderControl (see page 4118)

27.1.28.3 THeaderSections Methods

The methods of the THeaderSections class are listed here.

Protected Methods

	Name	Description
V	GetOwner (see page 4188)	Returns the owner of an object.
V	Update (see page 4189)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 4188)	Creates a new THeaderSection (see page 4173) instance and adds it to the Items (see page 4190) array.
	AddItem (see page 4188)	Add (see page 4188) section object to collection.
	Insert (see page 4189)	Add (see page 4188) new section object to collection.

27.1.28.3.1 THeaderSections.Add Method

Creates a new THeaderSection (see page 4173) instance and adds it to the Items (see page 4190) array.

Pascal

```
function Add: THeaderSection;
```

C++

```
__fastcall THeaderSection Add();
```

Description

Add returns the new header section. At design time, use the header control's Sections editor to add sections to the header.

See Also

Items (see page 4190)

Sections (see page 4147)

27.1.28.3.2 THeaderSections.AddItem Method

Add (see page 4188) section object to collection.

Pascal

```
function AddItem(Item: THeaderSection; Index: Integer): THeaderSection;
```

C++

```
__fastcall THeaderSection AddItem(THeaderSection Item, int Index);
```

Description

The AddItem method is similar to the Add (see page 4188) method, but is more flexible.

The Item parameter specifies the section object to add. If Item is nil, then a new panel object is created automatically. To determine the object type at runtime, define the status bar's OnCreateSectionClass event handler. If this event handler is not defined, the new section object is an instance of THeaderSection (see page 4173).

The Index parameter determines the Index property of the panel object. If Index is negative, the panel object is added at the end of the collection.

AddItem returns the panel object added.

See Also

Add (see page 4188)

Insert (see page 4189)

27.1.28.3.3 THeaderSections.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

27.1.28.3.4 THeaderSections.Insert Method

Add (see page 4188) new section object to collection.

Pascal

```
function Insert(Index: Integer): THeaderSection;
```

C++

```
__fastcall THeaderSection Insert(int Index);
```

Description

The Insert method creates a new section object and adds it the the collection. To determine the object type at runtime, define the header control's OnCreateSectionClass event handler. If this event handler is not defined, the new section object is an instance of THeaderSection (see page 4173).

The Index parameter determines the Index property of the panel object. If Index is negative, the panel object is added at the end of the collection.

Insert returns the new panel object.

See Also

Add (see page 4188)

AddItem (see page 4188)

27.1.28.3.5 THeaderSections.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed (see page 2740)

Changed (see page 2758)

27.1.28.4 THeaderSections Properties

The properties of the THeaderSections class are listed here.

Public Properties

	Name	Description
	Items (see page 4190)	Lists the header sections in the collection.

27.1.28.4.1 THeaderSections.Items Property

Lists the header sections in the collection.

Pascal

```
property Items [Index: Integer]: THeaderSection;
```

C++

```
__property THeaderSection Items[int Index];
```

Description

The value of the Index parameter corresponds to the Index property of THeaderSection (see page 4173). It represents the position of the header section in the header.

See Also

THeaderSection (see page 4173)

Index (see page 2762)

operator_[]

27.1.29 THotKey Class

THotKey is a wrapper for a Windows Common Control hot key.

Class Hierarchy

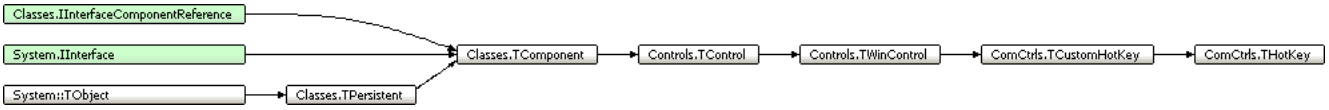

Pascal

```
THotKey = class(TCustomHotKey);
```

C++

```
class THotKey : public TCustomHotKey;
```

File

ComCtrls

Description

THotKey implements the generic behavior introduced in TCustomHotKey (see page 3739). THotKey publishes many of the properties inherited from TCustomHotKey (see page 3739), but does not introduce any new behavior.

Use THotKey to create key combinations that can be used as a shortcut.

See Also

Hot Key Controls (VCL Only)

Specialized Input Controls

27.1.29.1 THotKey Members

The following tables list the members exposed by THotKey.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.

	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	CNCommand (see page 3750)	This is CNCommand, a member of class TCustomHotKey.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.

	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 3751)	Initializes a window-creation parameter data structure.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 3751)	Creates the underlying window.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.

	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.

	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 3752)	This is SetAutoSize, a member of class TCustomHotKey.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.

	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.

	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 3750)	Instantiates an accelerator key control.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.

	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.

	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 4220)	Occurs when the properties of the accelerator key change.
	OnContextPopup (see page 4220)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnEnter (see page 4221)	Occurs when a control receives the input focus.
	OnExit (see page 4223)	Occurs when the input focus shifts away from one control to another.
	OnMouseActivate (see page 4225)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

	OnMouseDown (see page 4226)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4228)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4228)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4228)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4231)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 4202)	Specifies how the control is anchored to its parent.
	AutoSize (see page 4202)	Determines whether the accelerator key automatically changes its height to accommodate the font size.
	BiDiMode (see page 4203)	Specifies the bi-directional mode for the control.
	Constraints (see page 4203)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Enabled (see page 4204)	Controls whether the control responds to mouse, keyboard, and timer events.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 4204)	Contains the text string that can appear when the user moves the mouse over the control.
	HotKey (see page 4206)	Contains the current key combination of the accelerator key control.
	InvalidKeys (see page 4207)	Allows you to specify that one or more modifier keys should be considered invalid.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Modifiers (see page 4209)	Specifies the key or keys that are used in combination with a non-modifier key.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 4210)	Specifies whether the control uses its parent's BiDiMode.
	ParentShowHint (see page 4210)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4211)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 4215)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 4217)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 4217)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 4218)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.29.2 THotKey Properties

The properties of the THotKey class are listed here.

Published Properties

	Name	Description
	Anchors (see page 4202)	Specifies how the control is anchored to its parent.
	AutoSize (see page 4202)	Determines whether the accelerator key automatically changes its height to accommodate the font size.
	BiDiMode (see page 4203)	Specifies the bi-directional mode for the control.

	Constraints (see page 4203)	Specifies the size constraints for the control.
	Enabled (see page 4204)	Controls whether the control responds to mouse, keyboard, and timer events.
	Hint (see page 4204)	Contains the text string that can appear when the user moves the mouse over the control.
	HotKey (see page 4206)	Contains the current key combination of the accelerator key control.
	InvalidKeys (see page 4207)	Allows you to specify that one or more modifier keys should be considered invalid.
	Modifiers (see page 4209)	Specifies the key or keys that are used in combination with a non-modifier key.
	ParentBiDiMode (see page 4210)	Specifies whether the control uses its parent's BiDiMode.
	ParentShowHint (see page 4210)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4211)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 4215)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 4217)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 4217)	Determines if the user can tab to a control.
	Visible (see page 4218)	Determines whether the component appears onscreen.

27.1.29.2.1 THotKey.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.29.2.2 THotKey.AutoSize Property

Determines whether the accelerator key automatically changes its height to accommodate the font size.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;__property Boolean AutoSize;
```

Description

Use AutoSize to fully display the font on the accelerator key control. When AutoSize is true, the height of the accelerator key changes to display the key combinations typed in it. When AutoSize is false, the accelerator key remains the same height, regardless of any font changes. The default value is true.

27.1.29.2.3 THotKey.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

27.1.29.2.4 THotKey.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict,

the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.29.2.5 THotKey.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.29.2.6 THotKey.Hint Property

Contains the text string that can appear when the user moves the mouse over the control.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Use the Hint property to provide a string of help text either as a Help Hint, or as help text on a particular location such as a status bar.

A Help Hint is a box containing help text that appears for a control when the user moves the mouse pointer over the control and pauses momentarily. To set up Help Hints:

Specify the Hint property of each control for which a Help Hint should appear.

Set the ShowHint property of each appropriate control to true, or set the ParentShowHint property of all controls to true and set the ShowHint property of the form to true.

At runtime, set the value of the application's ShowHint property to true.

To show the Hint on a status bar or other location, use the OnHint event handler of the application. The application's OnHint event occurs when the mouse pointer moves over the control.

Specify a hint to be used for both a Help Hint box and by an OnHint handler by specifying two values separated by a | character (the vertical bar "or" symbol). For example,

```
Edit1.Hint := 'Name|Enter Name in the edit box';
Edit1->Hint = "Name|Enter Name in the edit box";
```

The "Name" part appears in the Help Hint box and the "Enter full name in the edit box" part can be extracted using the GetLongHint function for use in an OnHint event handler.

If Hint contains only one value, the entire string is used as a Help Hint and returned by the GetLongHint function. If a control has no Hint value specified, but its parent control does, the control uses the Hint value of the parent control (as long as the control's ShowHint property is true).

Note: If the application's ShowHint property is false, the Help Hint does not appear, but the OnHint event handler is still called.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
```

```
    Edit1.Hint := 'Enter your name';
    Edit1.ShowHint := True;
    with ListBox1 do
    begin
      for I := 1 to 10 do
        Items.Add('Item ' + IntToStr(I));
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

GetLongHint (see page 6252)

GetShortHint (see page 6253)

ParentShowHint (see page 5835)

ShowHint (see page 5840)

Hint (see page 13712)

OnHint (see page 13731)

ShowHint (see page 13721)

Hint (see page 312)

27.1.29.2.7 THotKey.HotKey Property

Contains the current key combination of the accelerator key control.

Pascal

```
property HotKey: TShortCut;
```

C++

```
__property TShortCut HotKey;
```

Description

Use HotKey to set or change the key combination associated with the accelerator key. Alt+A is the default. To modify the value of HotKey either edit it directly or set the value of the Modifiers property. At runtime, HotKey is reset when the accelerator key has focus and the user presses valid key combinations.

C++ Examples:

```
/*
To use this example, create a form with a main menu
component and a HotKey component.  At design time, use the
Menu designer to add menu items, including a "File" menu
with subitems that includes a "New" menu item.  Add the
following code to the OnCreate method of the form and the
OnClick event of the File menu item.
*/
void __fastcall TForm1::File1Click(TObject *Sender)
{
  // Assign HotKey to the menu's File "New" item.
  New1->ShortCut = HotKey1->HotKey;
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Default to "Ctrl + A"
  HotKey1->Modifiers.Clear();
  HotKey1->Modifiers << hkCtrl;
  // Don't allow shift or alt
  HotKey1->InvalidKeys.Clear();
  (HotKey1->InvalidKeys << hcShift) << hcAlt;
//  HotKey1->Visible = false;
}
```

Delphi Examples:

```
{
To use this example, create a form with a main menu
component and a HotKey component.  At design time, use the
Menu designer to add menu items, including a "File" menu
with subitems that includes a "New" menu item.  Add the
following code to the OnCreate method of the form and the
OnClick event of the File menu item.
}
procedure TForm1.File1Click(Sender: TObject);
begin
  // Assign hotkey to the menu's File "New" item.
  New1.ShortCut := HotKey1.HotKey;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  // Default to "Ctrl + A"
  HotKey1.Modifiers := [hkCtrl];
  // Don't allow shift or alt
  HotKey1.InvalidKeys := [hcShift, hcAlt];
end;
```

See Also

Modifiers (see page 3755)

27.1.29.2.8 THotKey.InvalidKeys Property

Allows you to specify that one or more modifier keys should be considered invalid.

Pascal

```
property InvalidKeys: THKInvalidKeys;
```

C++

```
__property THKInvalidKeys InvalidKeys;
```

Description

Use InvalidKeys to select which modifier keys are not allowed. If this property is set, a default modifier should also be specified in the Modifiers property. When the invalid key is selected, the default modifier is displayed in its place.

THKInvalidKey is the type of the InvalidKeys property. These are the possible values:

Value	Meaning
hcNone	Unmodified keys are invalid.
hcShift	The Shift key is invalid as a modifier.
hcCtrl	The Ctrl key is invalid as a modifier.

hcAlt	The Alt key is invalid as a modifier.
hcShiftCtrl	The Shift+Ctrl key combination is invalid as a modifier.
hcShiftAlt	The Shift+Alt key combinations invalid as a modifier.
hcCtrlAlt	The Ctrl+Alt key combination is invalid as a modifier.
hcShiftCtrlAlt	The Shift+Ctrl+Alt key combination is invalid as a modifier.

The default values are hcNone and hcShift.

C++ Examples:

```
/*
To use this example, create a form with a main menu
component and a HotKey component.  At design time, use the
Menu designer to add menu items, including a "File" menu
with subitems that includes a "New" menu item.  Add the
following code to the OnCreate method of the form and the
OnClick event of the File menu item.
*/
void __fastcall TForm1::File1Click(TObject *Sender)
{
  // Assign HotKey to the menu's File "New" item.
  New1->ShortCut = HotKey1->HotKey;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Default to "Ctrl + A"
  HotKey1->Modifiers.Clear();
  HotKey1->Modifiers << hkCtrl;
  // Don't allow shift or alt
  HotKey1->InvalidKeys.Clear();
  (HotKey1->InvalidKeys << hcShift) << hcAlt;
//  HotKey1->Visible = false;
}
```

Delphi Examples:

```
{
To use this example, create a form with a main menu
component and a HotKey component.  At design time, use the
Menu designer to add menu items, including a "File" menu
with subitems that includes a "New" menu item.  Add the
following code to the OnCreate method of the form and the
OnClick event of the File menu item.
}
procedure TForm1.File1Click(Sender: TObject);
begin
  // Assign hotkey to the menu's File "New" item.
  New1.ShortCut := HotKey1.HotKey;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  // Default to "Ctrl + A"
  HotKey1.Modifiers := [hkCtrl];
  // Don't allow shift or alt
  HotKey1.InvalidKeys := [hcShift, hcAlt];
end;
```

See Also

Modifiers (see page 3755)

27.1.29.2.9 THotKey.Modifiers Property

Specifies the key or keys that are used in combination with a non-modifier key.

Pascal

```
property Modifiers: THKModifiers;
```

C++

```
__property THKModifiers Modifiers;
```

Description

Use Modifiers to select a modifier key such as Ctrl, Alt, or Shift for the hot key. These are the possible values:

Value	Meaning
hkShift	The Shift key is used as a modifier.
hkCtrl	The Ctrl key is used as a modifier.
hkAlt	The Alt key is used as a modifier.
hkExt	The Extra key is used as a modifier.

The default value is hkAlt.

Modifier keys are used with other non-modifier keys such as character and function keys, arrow keys, and so on. More than one modifier key can be used.

C++ Examples:

```
/*
To use this example, create a form with a main menu
component and a HotKey component.  At design time, use the
Menu designer to add menu items, including a "File" menu
with subitems that includes a "New" menu item.  Add the
following code to the OnCreate method of the form and the
OnClick event of the File menu item.
*/
void __fastcall TForm1::File1Click(TObject *Sender)
{
  // Assign HotKey to the menu's File "New" item.
  New1->ShortCut = HotKey1->HotKey;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Default to "Ctrl + A"
  HotKey1->Modifiers.Clear();
  HotKey1->Modifiers << hkCtrl;
  // Don't allow shift or alt
  HotKey1->InvalidKeys.Clear();
  (HotKey1->InvalidKeys << hcShift) << hcAlt;
//  HotKey1->Visible = false;
}
```

Delphi Examples:

```
{
To use this example, create a form with a main menu
component and a HotKey component.  At design time, use the
Menu designer to add menu items, including a "File" menu
with subitems that includes a "New" menu item.  Add the
following code to the OnCreate method of the form and the
OnClick event of the File menu item.
```

```
}
procedure TForm1.File1Click(Sender: TObject);
begin
  // Assign hotkey to the menu's File "New" item.
  New1.ShortCut := HotKey1.HotKey;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  // Default to "Ctrl + A"
  HotKey1.Modifiers := [hkCtrl];
  // Don't allow shift or alt
  HotKey1.InvalidKeys := [hcShift, hcAlt];
end;
```

27.1.29.2.10 THotKey.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

27.1.29.2.11 THotKey.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.29.2.12 THotKey.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
```

```
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Cut1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      const int num_items = 4;
      for (int index = 0; index < num_items; ++index)
      {
          TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
          PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
          NewItem->Caption = "Menu Item " + IntToStr(index);
          NewItem->Tag = index;
          NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
          TNotifyEvent();
      }
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
          {
              case 0:
              {
                  ShowMessage("first item clicked");
```

```
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
```

```
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
```

```
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.29.2.13 THotKey.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
```

```
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.29.2.14 THotKey.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

27.1.29.2.15 THotKey.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

27.1.29.2.16 THotKey.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
```

```
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.29.3 THotKey Events

The events of the THotKey class are listed here.

Published Events

	Name	Description
	OnChange (see page 4220)	Occurs when the properties of the accelerator key change.
	OnContextPopup (see page 4220)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

	OnEnter (see page 4221)	Occurs when a control receives the input focus.
	OnExit (see page 4223)	Occurs when the input focus shifts away from one control to another.
	OnMouseActivate (see page 4225)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4226)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4228)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4228)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4228)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4231)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

27.1.29.3.1 THotKey.OnChange Event

Occurs when the properties of the accelerator key change.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to respond when the accelerator key changes to represent a different combination of keys. OnChange occurs after the TCustomHotKey properties have been updated to reflect the change.

27.1.29.3.2 THotKey.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.29.3.3 THotKey.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
```

```
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
```

```
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

27.1.29.3.4 THotKey.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on

the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
```

```
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

27.1.29.3.5 THotKey.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.29.3.6 THotKey.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
```

```
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
```

```
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.29.3.7 THotKey.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

27.1.29.3.8 THotKey.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

27.1.29.3.9 THotKey.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
```

```
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.29.3.10 THotKey.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
```

```
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
```

```
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

27.1.30 TIconOptions Class

TIconOptions describes how the icons in a list view should be arranged.

Class Hierarchy

Pascal

```
TIconOptions = class(TPersistent);
```

C++

```
class TIconOptions : public TPersistent;
```

File

ComCtrls

Description

Use TIconOptions with a list view object that has a ViewStyle property set to vsIcon or vsSmallIcon. The IconOptions property of the view contains a TIconOptions object that determines the arrangement of the icons.

See Also

TCustomForm (see page 13849)

TPicture (see page 14584)

TCustomListView (see page 3757)

TCustomListView (see page 3757)

TCustomListView (see page 3757)

27.1.30.1 TIconOptions Members

The following tables list the members exposed by TIconOptions.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 4235)	Creates an instance of TIconOptions (see page 4233).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Published Properties

	Name	Description
	Arrangement (see page 4235)	Determines where the items are to be aligned in the view.
	AutoArrange (see page 4236)	Determines if icons in a view are automatically rearranged when an item is added, removed, or moved.
	WrapText (see page 4236)	Determines whether the icon caption is displayed all on one line, or if it can wrap.

27.1.30.2 TIconOptions.Create Constructor

Creates an instance of TIconOptions (see page 4233).

Pascal

```
constructor Create(AOwner: TCustomListView);
```

C++

```
__fastcall TIconOptions(TCustomListView AOwner);
```

Description

Do not call Create directly in an application. The properties of TIconOptions (see page 4233) are only valid when it is owned by a view object. The view instantiates a TIconOptions (see page 4233) object to represent the arrangement properties of its icon views.

After calling the inherited constructor, Create initializes

Arrangement (see page 4235) to iaTop.

AutoArrange (see page 4236) to false.

WrapText (see page 4236) to true.

See Also

Arrangement (see page 4235)

AutoArrange (see page 4236)

WrapText (see page 4236)

27.1.30.3 TIconOptions Properties

The properties of the TIconOptions class are listed here.

Published Properties

	Name	Description
	Arrangement (see page 4235)	Determines where the items are to be aligned in the view.
	AutoArrange (see page 4236)	Determines if icons in a view are automatically rearranged when an item is added, removed, or moved.
	WrapText (see page 4236)	Determines whether the icon caption is displayed all on one line, or if it can wrap.

27.1.30.3.1 TIconOptions.Arrangement Property

Determines where the items are to be aligned in the view.

Pascal

```
property Arrangement: TIconArrangement;
```

C++

```
__property TIconArrangement Arrangement;
```

Description

Set Arrangement to indicate whether the view should display icon entries in rows or columns. Arrangement has no effect on the arrangement of entries in the view when the ViewStyle is not vsIcon or vsSmallIcon. These are the possible values of

Arrangement:

Value	Meaning
iaTop	Items are aligned left to right in rows at the top of the view control
iaLeft	Items are aligned in columns top to bottom at the left of the view control

See Also

AutoArrange (see page 4236)

ViewStyle (see page 3829)

27.1.30.3.2 TIconOptions.AutoArrange Property

Determines if icons in a view are automatically rearranged when an item is added, removed, or moved.

Pascal

```
property AutoArrange: Boolean;
```

C++

```
__property Boolean AutoArrange;
```

Description

Set AutoArrange to specify whether the view should rearrange its items whenever changes are made. If AutoArrange is true, the icons are automatically arranged. If AutoArrange is false, the icons are not arranged until the Arrange method of the view is called.

See Also

Arrangement (see page 4235)

Arrange (see page 3777)

27.1.30.3.3 TIconOptions.WrapText Property

Determines whether the icon caption is displayed all on one line, or if it can wrap.

Pascal

```
property WrapText: Boolean;
```

C++

```
__property Boolean WrapText;
```

Description

Set WrapText to true to have the caption of icons wrap rather than display on one line when it exceeds the width of the icon.

See Also

TCustomListView (see page 3757)

27.1.31 TListColumn Class

TListColumn represents a column in a report-style list view.

Class Hierarchy

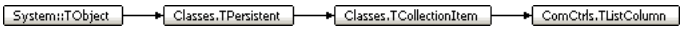

Pascal

```
TListColumn = class(TCollectionItem);
```

C++

```
class TListColumn : public TCollectionItem;
```

File

ComCtrls

Description

TListView (see page 4285) objects use TListColumns (see page 4245) to maintain a collection of TListColumn objects.

See Also

TListColumns (see page 4245)

TListView (see page 4285)

27.1.31.1 TListColumn Members

The following tables list the members exposed by TListColumn.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 4240)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 4241)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 4241)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 4239)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 4238)	Creates and initializes a TCollectionItem instance.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 4239)	Destroys the list column and frees its memory.

V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
R	WidthType (see page 4245)	This is WidthType, a member of class TListColumn.

Published Properties

	Name	Description
	Alignment (see page 4242)	Specifies how all text is aligned within the list column.
	AutoSize (see page 4242)	Specifies whether the list column automatically sizes itself to the width of its text.
	Caption (see page 4242)	Specifies the text that appears at the top of the column.
	ImageIndex (see page 4244)	This is ImageIndex, a member of class TListColumn.
	MaxWidth (see page 4244)	This is MaxWidth, a member of class TListColumn.
	MinWidth (see page 4244)	This is MinWidth, a member of class TListColumn.
	Tag (see page 4244)	This is Tag, a member of class TListColumn.
	Width (see page 4245)	Specifies the width of the column.

27.1.31.2 TListColumn.Create Constructor

Creates and initializes a TCollectionItem instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TListColumn(TCollection Collection);
```

Description

Do not call Create to instantiate a collection item. Instead, call the Add method of the collection to which the item should belong.

Collection identifies the TCollection instance to which the new item belongs.

See Also

Add (see page 2736)

Collection (see page 2761)

27.1.31.3 TListColumn.Destroy Destructor

Destroys the list column and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TListColumn();
```

Description

Do not call Destroy. TListColumn (see page 4236) objects are destroyed by the TListColumns (see page 4245) object that contains them.

Destroy eliminates the TListColumn (see page 4236) instance and, if necessary, deletes the column from the list view control.

See Also

Free (see page 22300)

TListView (see page 4285)

TListColumns (see page 4245)

27.1.31.4 TListColumn Methods

The methods of the TListColumn class are listed here.

Protected Methods

	Name	Description
	DefineProperties (see page 4240)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 4241)	Returns the name of the collection item as it appears in the collection editor.
	SetIndex (see page 4241)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	Assign (see page 4239)	Copies the contents of another, similar object.

27.1.31.4.1 TListColumn.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo (see page 2931)

EConvertError (see page 22724)

27.1.31.4.2 TListColumn.DefineProperties Method

Provides an interface for a method that reads and writes otherwise unpublished data.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

Descendants of TPersistent override DefineProperties to designate a method for storing the object's unpublished data to a stream such as a form file. By default, writing an object to a stream writes the values of all its published properties, and reading the object in reads those values and assigns them to the properties. Objects can also specify methods that read and write data other than published properties by overriding the DefineProperties method.

When overriding DefineProperties, consider including some or all of the following:

A call to the inherited method

Calls to the filer object's DefineProperty method

Calls to the filer object's DefineBinaryProperty method

DefineProperties is virtual, so descendant classes can override it as necessary but are not required to do so.

See Also

DefineProperties (see page 2771)

DefineBinaryProperty (see page 2837)

DefineProperty (see page 2838)

27.1.31.4.3 TListColumn.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName (see page 2761)

GetNamePath (see page 2759)

27.1.31.4.4 TListColumn.SetIndex Method

Moves the collection item to a specified position in the parent collection.

Pascal

```
procedure SetIndex(Value: Integer); override;
```

C++

```
virtual __fastcall SetIndex(int Value);
```

Description

SetIndex is the protected write implementation of the Index property. The Value parameter specifies the new (0-based) position for the item in its parent collection.

See Also

Index (see page 2762)

27.1.31.5 TListColumn Properties

The properties of the TListColumn class are listed here.

Public Properties

	Name	Description
R	WidthType (see page 4245)	This is WidthType, a member of class TListColumn.

Published Properties

	Name	Description
	Alignment (see page 4242)	Specifies how all text is aligned within the list column.

	AutoSize (see page 4242)	Specifies whether the list column automatically sizes itself to the width of its text.
	Caption (see page 4242)	Specifies the text that appears at the top of the column.
	ImageIndex (see page 4244)	This is ImageIndex, a member of class TListColumn.
	MaxWidth (see page 4244)	This is MaxWidth, a member of class TListColumn.
	MinWidth (see page 4244)	This is MinWidth, a member of class TListColumn.
	Tag (see page 4244)	This is Tag, a member of class TListColumn.
	Width (see page 4245)	Specifies the width of the column.

27.1.31.5.1 TListColumn.Alignment Property

Specifies how all text is aligned within the list column.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to specify the horizontal placement of text in the column. The text includes the Caption (see page 4242) that appears at the top of the column and the text labels of individual list items that appear in the column.

See Also

Caption (see page 4242)

Caption (see page 4258)

27.1.31.5.2 TListColumn.AutoSize Property

Specifies whether the list column automatically sizes itself to the width of its text.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to get or set whether the list column automatically sizes itself to the width of its text.

27.1.31.5.3 TListColumn.Caption Property

Specifies the text that appears at the top of the column.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to label the type of item that appears in the column. If the list view control's ViewStyle and ShowColumnHeaders properties are set to vsReport and true, respectively, the value of Caption appears at the top of the column.

C++ Examples:

```
/*
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses defs file.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const char Names[6][2][10] =
   {{"Rubble","Barny"},
    {"Michael", "Johnson"},
    {"Bunny", "Bugs"},
    {"Silver", "HiHo"},
    {"Simpson", "Bart"},
    {"Squirrel", "Rocky"}};

  TListColumn  *NewColumn;
  TListItem  *ListItem;
  TListView   *ListView = new TListView(this);

  ListView->Parent = this;
  ListView->Align = alClient;
  ListView->ViewStyle = vsReport;
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "Last";
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "First";
  for (int i = 0; i < 6; i++)
  {
    ListItem = ListView->Items->Add();
    ListItem->Caption = Names[i][0];
    ListItem->SubItems->Add(Names[i][1]);
  }
}
```

Delphi Examples:

```
{
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses clause of the unit file.
}
procedure TForm1.FormCreate(Sender: TObject);
const
  Names: array[0..5, 0..1] of string = (
    ('Rubble', 'Barney'),
    ('Michael', 'Johnson'),
    ('Bunny', 'Bugs'),
    ('Silver', 'HiHo'),
    ('Simpson', 'Bart'),
    ('Squirrel', 'Rocky')
    );

var
  I: Integer;
  NewColumn: TListColumn;
  ListItem: TListItem;
  ListView: TListView;
begin
  ListView := TListView.Create(Self);
  with ListView do
  begin
    Parent := Self;
    Align := alClient;
    ViewStyle := vsReport;

    NewColumn := Columns.Add;
    NewColumn.Caption := 'Last';
```

```
      NewColumn := Columns.Add;
      NewColumn.Caption := 'First';

      for I := Low(Names) to High(Names) do
      begin
        ListItem := Items.Add;
        ListItem.Caption := Names[I][0];
        ListItem.SubItems.Add(Names[I][1]);
      end;
    end;
  end;
```

See Also

ShowColumnHeaders (see page 3824)

ViewStyle (see page 3829)

27.1.31.5.4 TListColumn.ImageIndex Property

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

This is ImageIndex, a member of class TListColumn.

27.1.31.5.5 TListColumn.MaxWidth Property

Pascal

```
property MaxWidth: TWidth;
```

C++

```
__property TWidth MaxWidth;
```

Description

This is MaxWidth, a member of class TListColumn.

27.1.31.5.6 TListColumn.MinWidth Property

Pascal

```
property MinWidth: TWidth;
```

C++

```
__property TWidth MinWidth;
```

Description

This is MinWidth, a member of class TListColumn.

27.1.31.5.7 TListColumn.Tag Property

Pascal

```
property Tag: Integer;
```

C++

```
__property int Tag;
```

Description

This is Tag, a member of class TListColumn.

27.1.31.5.8 TListColumn.Width Property

Specifies the width of the column.

Pascal

```
property Width: TWidth;
```

C++

```
__property TWidth Width;
```

Description

The Width property determines the default width of the column, in pixels. List columns can be resized at runtime by dragging their borders.

See Also

Caption (see page 4242)

Columns (see page 3808)

AutoSize

27.1.31.5.9 TListColumn.WidthType Property

Pascal

```
property WidthType: TWidth;
```

C++

```
__property TWidth WidthType;
```

Description

This is WidthType, a member of class TListColumn.

27.1.32 TListColumns Class

TListColumns is a collection of TListColumn (see page 4236) objects.

Class Hierarchy

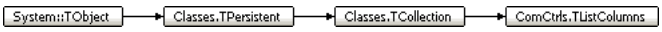

Pascal

```
TListColumns = class(TCollection);
```

C++

```
class TListColumns : public TCollection;
```

File

ComCtrls

Description

Each TListColumns represents a collection of TListColumn (see page 4236) objects in a TListView (see page 4285). At design time, use the list view control's Columns editor to add, remove, or modify columns.

See Also

TListColumn (see page 4236)

TListView (see page 4285)

TCustomListView (see page 3757)

27.1.32.1 TListColumns Members

The following tables list the members exposed by TListColumns.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 4248)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 4249)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 4248)	Creates a new TListColumn (see page 4236) instance and adds it to the Items (see page 4250) array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 4247)	Creates and initializes a TListColumns (see page 4245) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 4249)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 4250)	Lists the columns in the collection.

27.1.32.2 TListColumns.Create Constructor

Creates and initializes a TListColumns (see page 4245) object.

Pascal

```
constructor Create(AOwner: TCustomListView);
```

C++

```
__fastcall TListColumns(TCustomListView AOwner);
```

Description

Applications should not need to call the Create method directly. Instances of TListView (see page 4285) create the TListColumns (see page 4245) object from their own constructors. The AOwner parameter is the TListView (see page 4285) instance that uses the TListColumns (see page 4245) object to implement its Columns property.

See Also

Columns (see page 3808)

27.1.32.3 TListColumns Methods

The methods of the TListColumns class are listed here.

Protected Methods

	Name	Description
	GetOwner (see page 4248)	Returns the owner of an object.
	Update (see page 4249)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 4248)	Creates a new TListColumn (see page 4236) instance and adds it to the Items (see page 4250) array.
	Owner (see page 4249)	Returns the Owner of the collection.

27.1.32.3.1 TListColumns.Add Method

Creates a new TListColumn (see page 4236) instance and adds it to the Items (see page 4250) array.

Pascal

```
function Add: TListColumn;
```

C++

```
__fastcall TListColumn Add();
```

Description

Call Add to add a new list column to the end of the items array. Add returns the new list column. At design time, use the list view control's Columns editor to add columns to the list view.

See Also

Items (see page 4250)

Columns (see page 3808)

27.1.32.3.2 TListColumns.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

27.1.32.3.3 TListColumns.Owner Method

Returns the Owner of the collection.

Pascal

```
function Owner: TCustomListView;
```

C++

```
__fastcall TCustomListView Owner();
```

Description

Call Owner to obtain a reference to the list view that owns this collection.

27.1.32.3.4 TListColumns.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed (see page 2740)

Changed (see page 2758)

27.1.32.4 TListColumns Properties

The properties of the TListColumns class are listed here.

Public Properties

	Name	Description
	Items (see page 4250)	Lists the columns in the collection.

27.1.32.4.1 TListColumns.Items Property

Lists the columns in the collection.

Pascal

```
property Items [Index: Integer]: TListColumn;
```

C++

```
__property TListColumn Items[int Index];
```

Description

Use Items to directly access the list column objects in the collection. The value of the Index parameter corresponds to the Index property of TListColumn (see page 4236). It represents the position of the column in the list view.

See Also

Count (see page 2749)

operator_[]

27.1.33 TListItem Class

TListItem is an individual item of a TListView (see page 4285) control.

Class Hierarchy

Pascal

```
TListItem = class(TPersistent);
```

C++

```
class TListItem : public TPersistent;
```

File

ComCtrls

Description

Use TListItem to specify the appearance and data associations of an item in a list view. All the list items of the list view are collected by a TListItems (see page 4270) object to make up the Items property of the TListView (see page 4285) control.

See Also

TListItems (see page 4270)

TListView (see page 4285)

27.1.33.1 TListItem Members

The following tables list the members exposed by TListItem.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 4253)	Copies the properties of another list item.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CancelEdit (see page 4254)	Cancels the editing of the list item's caption.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 4252)	Creates an instance of TListItem (see page 4250).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 4254)	Deletes the list item from its list view.
	Destroy (see page 4252)	Destroys an instance of TListItem (see page 4250).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DisplayRect (see page 4254)	Returns the bounding rectangle of the list item.
	EditCaption (see page 4255)	Begins in-place editing of the list item's caption.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetPosition (see page 4255)	Returns the position of the list item as a point.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MakeVisible (see page 4256)	Scrolls the list view, if necessary, to ensure a list item is in view.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetPosition (see page 4256)	Positions the list item in its list view control.

	Update (see page 4256)	Updates the image of the list item in the list view display.
	WorkArea (see page 4257)	Returns the work area (if any) that contains the list item.

Public Properties

	Name	Description
	Caption (see page 4258)	Specifies the text that labels the item in the list.
	Checked (see page 4258)	Determines whether a check mark appears next to the list item.
	Cut (see page 4259)	Determines whether the list item is drawn as if it is selected for a cut operation.
	Data (see page 4259)	Specifies any application-specific data associated with the list item.
R	Deleting (see page 4260)	Indicates whether an item's Destroy (see page 4252) method has been called and the item is in the process of being deleted.
	DropTarget (see page 4260)	Determines whether the list item appears selected as a drag and drop target.
	Focused (see page 4260)	Indicates whether the list item has input focus.
R	Handle (see page 4261)	Indicates the window handle of the list view that displays the list item.
	ImageIndex (see page 4261)	Determines which image is displayed as the icon for the list item.
	Indent (see page 4263)	Specifies how far the list item is indented.
R	Index (see page 4264)	Indicates the position of the list item in the TListItems (see page 4270) collection.
	Left (see page 4265)	Specifies the distance, in pixels, from the left edge of the list view to the left edge of the list item.
R	ListView (see page 4265)	Indicates the list view control that displays the list item.
	OverlayIndex (see page 4266)	Determines which image from the image list is used as an overlay mask.
R	Owner (see page 4266)	Specifies the TListItems (see page 4270) object that owns the list item.
	Position (see page 4267)	Specifies the position of the list item.
	Selected (see page 4267)	Indicates whether the list item is selected.
	StateIndex (see page 4267)	Specifies which image from the StateImages image list to display for the item.
	SubItemImages (see page 4268)	Indicates which images (if any) should appear next to subitems of the item.
	SubItems (see page 4268)	Contains any strings that appear as subitems to the list item.
	Top (see page 4270)	Specifies the distance, in pixels, from the top of the list view to the top of the list item.

27.1.33.2 TListItem.Create Constructor

Creates an instance of TListItem (see page 4250).

Pascal

```
constructor Create(AOwner: TListItems);
```

C++

```
__fastcall TListItem(TListItems AOwner);
```

Description

Call Create to instantiate a TListItem (see page 4250) object at runtime. List items added to the list view at design time are created automatically. At runtime, use the Add method of the TListItems (see page 4270) object to create a list item and add it to the list view.

See Also

Add (see page 4273)

27.1.33.3 TListItem.Destroy Destructor

Destroys an instance of TListItem (see page 4250).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TListItem();
```

Description

Do not call Destroy directly. Instead, use Delete (see page 4254), which removes the item from the list view windows before freeing the associated memory.

See Also

Delete (see page 4254)

Free (see page 22300)

27.1.33.4 TListItem Methods

The methods of the TListItem class are listed here.

Public Methods

	Name	Description
	Assign (see page 4253)	Copies the properties of another list item.
	CancelEdit (see page 4254)	Cancels the editing of the list item's caption.
	Delete (see page 4254)	Deletes the list item from its list view.
	DisplayRect (see page 4254)	Returns the bounding rectangle of the list item.
	EditCaption (see page 4255)	Begins in-place editing of the list item's caption.
	GetPosition (see page 4255)	Returns the position of the list item as a point.
	MakeVisible (see page 4256)	Scrolls the list view, if necessary, to ensure a list item is in view.
	SetPosition (see page 4256)	Positions the list item in its list view control.
	Update (see page 4256)	Updates the image of the list item in the list view display.
	WorkArea (see page 4257)	Returns the work area (if any) that contains the list item.

27.1.33.4.1 TListItem.Assign Method

Copies the properties of another list item.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

It the Source parameter is a TListItem (see page 4250) object, Assign copies its Caption (see page 4258), Data (see page 4259), ImageIndex (see page 4261), OverlayIndex (see page 4266), StateIndex (see page 4267), and SubItems (see page 4268) properties. Otherwise, Assign calls the inherited method so that any object that copies properties to a list item in its AssignTo method can do so.

See Also

Caption (see page 4258)

Data (see page 4259)

ImageIndex (see page 4261)

OverlayIndex (see page 4266)

StateIndex (see page 4267)

SubItems (see page 4268)

AssignTo (see page 2931)

27.1.33.4.2 TListItem.CancelEdit Method

Cancels the editing of the list item's caption.

Pascal

```
procedure CancelEdit;
```

C++

```
__fastcall CancelEdit();
```

Description

Call CancelEdit to end the editing of the item's caption, discarding any changes made by the user.

See Also

Caption (see page 4258)

EditCaption (see page 4255)

27.1.33.4.3 TListItem.Delete Method

Deletes the list item from its list view.

Pascal

```
procedure Delete;
```

C++

```
__fastcall Delete();
```

Description

Use the Delete method to delete a list item.

See Also

Destroy (see page 4252)

Free (see page 22300)

27.1.33.4.4 TListItem.DisplayRect Method

Returns the bounding rectangle of the list item.

Pascal

```
function DisplayRect(Code: TDisplayCode): TRect;
```

C++

```
__fastcall TRect DisplayRect(TDisplayCode Code);
```

Description

Use DisplayRect to get the coordinates of the list item in the client coordinates of the containing listview. The Code parameter

specifies what part of the item the bounding rectangle surrounds. These are the possible values for the Code parameter.

Value	Meaning
drBounds	Returns the bounding rectangle of the entire list item, including the icon and label.
drIcon	Returns the bounding rectangle of the icon or small icon.
drLabel	Returns the bounding rectangle of the item text.
drSelectBounds	Returns the union of the icon and label rectangles, but excludes columns in report view.

See Also

GetPosition (see page 4255)

Left (see page 4265)

Top (see page 4270)

27.1.33.4.5 TListItem.EditCaption Method

Begins in-place editing of the list item's caption.

Pascal

```
function EditCaption: Boolean;
```

C++

```
__fastcall Boolean EditCaption();
```

Description

Call EditCaption to put the list item into edit mode. Before bringing up the in-place editor for the item's caption, EditCaption makes the item the only selected item in the list view and gives it input focus.

See Also

CancelEdit (see page 4254)

Caption (see page 4258)

Focused (see page 4260)

Selected (see page 4267)

OnEdited (see page 3849)

OnEditing (see page 3850)

27.1.33.4.6 TListItem.GetPosition Method

Returns the position of the list item as a point.

Pascal

```
function GetPosition: TPoint;
```

C++

```
__fastcall TPoint GetPosition();
```

Description

Call GetPosition to obtain the upper left coordinate of the list item. GetPosition returns a point with the x coordinate set to the value of Left (see page 4265) and the y coordinate set to the value of Top (see page 4270).

See Also

DisplayRect (see page 4254)

Left (see page 4265)

SetPosition (see page 4256)

Top (see page 4270)

27.1.33.4.7 TListItem.MakeVisible Method

Scrolls the list view, if necessary, to ensure a list item is in view.

Pascal

```
procedure MakeVisible(PartialOK: Boolean);
```

C++

```
__fastcall MakeVisible(Boolean PartialOK);
```

Description

Call MakeVisible to ensure that the user can see the list item in the list view. The PartialOK parameter specifies whether the item must be entirely visible. If PartialOK is true, no scrolling occurs if the item is at least partially visible. If PartialOK is false, MakeVisible ensures that the entire list item is visible in the list view.

See Also

DisplayRect (see page 4254)

27.1.33.4.8 TListItem.SetPosition Method

Positions the list item in its list view control.

Pascal

```
procedure SetPosition(const Value: TPoint);
```

C++

```
__fastcall SetPosition(const TPoint Value);
```

Description

Call SetPosition to move the list item to a specified position. The Value parameter specifies the desired coordinates of the upper left corner of the list item.

Note: The list view control must have its ViewStyle property set to either vsIcon or vsSmallIcon.

See Also

DisplayRect (see page 4254)

Left (see page 4265)

Top (see page 4270)

ViewStyle (see page 3829)

27.1.33.4.9 TListItem.Update Method

Updates the image of the list item in the list view display.

Pascal

```
procedure Update;
```

C++

```
__fastcall Update();
```

Description

Call Update to update the image of the list view to reflect any changes made to the list item. Changing the Caption (see page 4258), position, or state of the list item automatically updates the list item's image.

See Also

Caption (see page 4258)

Cut (see page 4259)

Focused (see page 4260)

Left (see page 4265)

Selected (see page 4267)

Top (see page 4270)

27.1.33.4.10 TListItem.WorkArea Method

Returns the work area (if any) that contains the list item.

Pascal

```
function WorkArea: Integer;
```

C++

```
__fastcall int WorkArea();
```

Description

When the list view that contains the list item organizes its client area into separate rectangular work areas, each list item is associated with a single work area. WorkArea returns the index of this work area in the list maintained by the list view's WorkAreas property. If the list view does not use work areas, WorkArea returns –1.

To change the work area of a list item, move it into a position of the list view's client area that is covered by a different work area.

Note: Work areas are not supported if the application is running in virtual mode.

See Also

WorkAreas (see page 3833)

TWorkArea (see page 5425)

Position (see page 4267)

27.1.33.5 TListItem Properties

The properties of the TListItem class are listed here.

Public Properties

	Name	Description
	Caption (see page 4258)	Specifies the text that labels the item in the list.

	Checked (see page 4258)	Determines whether a check mark appears next to the list item.
	Cut (see page 4259)	Determines whether the list item is drawn as if it is selected for a cut operation.
	Data (see page 4259)	Specifies any application-specific data associated with the list item.
R	Deleting (see page 4260)	Indicates whether an item's Destroy (see page 4252) method has been called and the item is in the process of being deleted.
	DropTarget (see page 4260)	Determines whether the list item appears selected as a drag and drop target.
	Focused (see page 4260)	Indicates whether the list item has input focus.
R	Handle (see page 4261)	Indicates the window handle of the list view that displays the list item.
	ImageIndex (see page 4261)	Determines which image is displayed as the icon for the list item.
	Indent (see page 4263)	Specifies how far the list item is indented.
R	Index (see page 4264)	Indicates the position of the list item in the TListItems (see page 4270) collection.
	Left (see page 4265)	Specifies the distance, in pixels, from the left edge of the list view to the left edge of the list item.
R	ListView (see page 4265)	Indicates the list view control that displays the list item.
	OverlayIndex (see page 4266)	Determines which image from the image list is used as an overlay mask.
R	Owner (see page 4266)	Specifies the TListItems (see page 4270) object that owns the list item.
	Position (see page 4267)	Specifies the position of the list item.
	Selected (see page 4267)	Indicates whether the list item is selected.
	StateIndex (see page 4267)	Specifies which image from the StateImages image list to display for the item.
	SubItemImages (see page 4268)	Indicates which images (if any) should appear next to subitems of the item.
	SubItems (see page 4268)	Contains any strings that appear as subitems to the list item.
	Top (see page 4270)	Specifies the distance, in pixels, from the top of the list view to the top of the list item.

27.1.33.5.1 TListItem.Caption Property

Specifies the text that labels the item in the list.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to name the list item. The Caption appears with an associated image that depends on the ViewStyle property of the list view.

If the ReadOnly property of the list view is false, the user can edit the caption.

See Also

ImageIndex (see page 4261)

ReadOnly (see page 3821)

ViewStyle (see page 3829)

27.1.33.5.2 TListItem.Checked Property

Determines whether a check mark appears next to the list item.

Pascal

```
property Checked: Boolean;
```

C++

```
__property Boolean Checked;
```

Description

Use Checked to indicate whether the list item represents a selected option by displaying a check mark next to the item in the list view. When Checked is true, a check mark appears next to the item. When Checked is false, no check mark appears.

Note: Set the CheckBoxes property of the List View to true before using the Checked property.

See Also

Selected (see page 4267)

StateIndex (see page 4267)

Checkboxes (see page 3805)

27.1.33.5.3 TListItem.Cut Property

Determines whether the list item is drawn as if it is selected for a cut operation.

Pascal

```
property Cut: Boolean;
```

C++

```
__property Boolean Cut;
```

Description

Set the Cut property to give the list item the grayed-out appearance of an item selected for cutting. Cut does not copy the item to the Clipboard or remove it from the list: the application must still perform these operations.

See Also

DropTarget (see page 4260)

Selected (see page 4267)

StateIndex (see page 4267)

27.1.33.5.4 TListItem.Data Property

Specifies any application-specific data associated with the list item.

Pascal

```
property Data: Pointer;
```

C++

```
__property void * Data;
```

Description

Use Data to associate arbitrary data structure with the list item. When the user selects or deletes the list item, Data allows the application to quickly access information about the meaning of the list item to implement the appropriate response.

See Also

Delete (see page 4254)

Selected (see page 4267)

27.1.33.5.5 TListItem.Deleting Property

Indicates whether an item's Destroy (see page 4252) method has been called and the item is in the process of being deleted.

Pascal

```
property Deleting: Boolean;
```

C++

```
__property Boolean Deleting;
```

Description

Deleting returns true if the item object is in the process of being deleted. Use Deleting to prevent infinite recursion in event handlers.

See Also

Destroy (see page 4252)

OnDeletion (see page 3848)

27.1.33.5.6 TListItem.DropTarget Property

Determines whether the list item appears selected as a drag and drop target.

Pascal

```
property DropTarget: Boolean;
```

C++

```
__property Boolean DropTarget;
```

Description

Use DropTarget to display the list item as the current drop target of the object the user is dragging. Setting DropTarget to true sets the DropTarget property of all other items in the list view to false.

See Also

Cut (see page 4259)

StateIndex (see page 4267)

27.1.33.5.7 TListItem.Focused Property

Indicates whether the list item has input focus.

Pascal

```
property Focused: Boolean;
```

C++

```
__property Boolean Focused;
```

Description

Set Focused to true to allow the user to edit the Caption (see page 4258).

Note: Only one item in the view should have focus at a time. When setting Focused to true, be sure to set the Focused property of all other items in the list view to false. When the list item is Focused, it appears surrounded by a standard focus rectangle.

See Also

Caption (see page 4258)

Selected (see page 4267)

27.1.33.5.8 TListItem.Handle Property

Indicates the window handle of the list view that displays the list item.

Pascal

```
property Handle: HWND;
```

C++

```
__property HWND Handle;
```

Description

A window handle identifies a the window to the underlying API. Handle is a read-only property containing the handle of the list view that displays the list item.

See Also

Handle (see page 6207)

27.1.33.5.9 TListItem.ImageIndex Property

Determines which image is displayed as the icon for the list item.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

Set ImageIndex to associate the list item with one of the images in the LargeImages or SmallImages property of the list view. If the ViewStyle of the list view is vsIcon, ImageIndex is the index of the image in the LargeImages property of the list view, where 0 specifies the first image. If the ViewStyle of the list view is vsSmallIcon, vsList, or vsReport, ImageIndex is the index of the image in the SmallImages property of the list view.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", (TObject *)vsIcon);
  ComboBox1->Items->AddObject("vsList", (TObject *)vsList);
  ComboBox1->Items->AddObject("vsReport", (TObject *)vsReport);
  ComboBox1->Items->AddObject("vsSmallIcon", (TObject *)vsSmallIcon);
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
```

```
      SmallImages := ImageList1;
      LargeImages := ImageList1;
      for I := 0 to ImageList1.Count - 1 do
      begin
        ListItem := Items.Add;
        Listitem.Caption := 'Image' + IntToStr(I);
        ListItem.ImageIndex := I;
        Listitem.SubItems.Add(Col2Array[I]);
      end;
      // Create two columns to show during viewing as vsReport
      NewColumn := Columns.Add;
      NewColumn.Caption := 'Column 1';
      NewColumn.Width := 200;
      NewColumn := Columns.Add;
      NewColumn.Caption := 'Column 2';
      NewColumn.Width := 200;
      // Add View styles and constants to the Combo Box
      ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
      ComboBox1.Items.AddObject('vsList', TObject(vsList));
      ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
      ComboBox1.Items.AddObject(
        'vsSmallIcon', TObject(vsSmallIcon));
      // Display first item in the Combo Box
      ComboBox1.ItemIndex := 0;
    end;
  end;

  procedure TComboForm.ComboBox1Click(Sender: TObject);
  begin
    with ComboBox1 do
      ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

  end;
```

See Also

OverlayIndex (see page 4266)

StateIndex (see page 4267)

LargeImages (see page 3817)

SmallImages (see page 3825)

ViewStyle (see page 3829)

27.1.33.5.10 TListItem.Indent Property

Specifies how far the list item is indented.

Pascal

```
property Indent: Integer;
```

C++

```
__property int Indent;
```

Description

Use Indent to get or set how far the list item is indented when the listview's ViewStyle property is vsReport. Indent specifies the factor times which the Width property of the SmallImages image list is multiplied to obtain the indent in a report style list view. That is, items are indented by

```
Indent*ListView.SmallImages.Width.
Indent*ListView->SmallImages->Width
```

See Also

ViewStyle (see page 3829)

SmallImages (see page 3825)

Width (see page 15392)

27.1.33.5.11 TListItem.Index Property

Indicates the position of the list item in the TListItems (see page 4270) collection.

Pascal

```
property Index: Integer;
```

C++

```
__property int Index;
```

Description

Use Index to locate the item in the list view. The first item has an index of 0, the second item has an index of 1, and so on.

C++ Examples:

```
/*
The following example inserts a list item after the selected
item in the list view.  Place a TButton, a TEdit and a
TListView in the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TListItem *InsertItem;
  if (ListView1->Selected == NULL) return;
  InsertItem =
    ListView1->Items->Insert(ListView1->Selected->Index);
  InsertItem->Caption = Edit1->Text;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *ListItem;
  ListView1->ViewStyle = vsList;
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Apples";
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Oranges";
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Pears";
}
```

Delphi Examples:

```
{
The following example inserts a list item after the selected
item in the list view.  Place a TButton, a TEdit and a
TListView in the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var InsertItem : TListItem;
begin
  with ListView1 do
  begin
    if (Selected = nil) then exit;
    InsertItem := Items.Insert(Selected.Index);
  end;
```

```
    InsertItem.Caption := Edit1.Text;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var ListItem : TListItem;
  begin
    ListView1.ViewStyle := vsList;
    ListItem := ListView1.Items.Add;
    ListItem.Caption := 'Apples';
    ListItem := ListView1.Items.Add;
    ListItem.Caption := 'Oranges';
    ListItem := ListView1.Items.Add;
    ListItem.Caption := 'Pears';
  end;
```

See Also

Items (see page 3814)

TListItems (see page 4270)

27.1.33.5.12 TListItem.Left Property

Specifies the distance, in pixels, from the left edge of the list view to the left edge of the list item.

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

Use Left to horizontally position the list item within the list view. Only use the Left property when the ViewStyle property of the list view is vsIcon or vsSmallIcon.

See Also

DisplayRect (see page 4254)

GetPosition (see page 4255)

SetPosition (see page 4256)

Top (see page 4270)

ViewStyle (see page 3829)

27.1.33.5.13 TListItem.ListView Property

Indicates the list view control that displays the list item.

Pascal

```
property ListView: TCustomListView;
```

C++

```
__property TCustomListView ListView;
```

Description

Read ListView to access the list view object that displays the item. Use the properties and methods of the list view object to make changes that affect the list item or to manipulate other items in the list.

Do not confuse the ListView that displays the item with the Owner (see page 4266) of the item. The Owner (see page 4266) is the TListItems (see page 4270) object that manages the collection of all list items. The list view is the Owner (see page 4266) of that TListItems (see page 4270) object.

See Also

Owner (see page 4266)

TListView (see page 4285)

27.1.33.5.14 TListItem.OverlayIndex Property

Determines which image from the image list is used as an overlay mask.

Pascal

```
property OverlayIndex: TImageIndex;
```

C++

```
__property TImageIndex OverlayIndex;
```

Description

Use OverlayIndex to display an overlay mask on an item in the list when the ViewStyle of the list view is vsIcon or vsSmallIcon. OverlayIndex is the index of the overlay image in the StateImages property of the list view.

An overlay mask is an image drawn transparently over the icon for the item in the list view. For example, to indicate that an item is no longer available, an application could use an overlay image of an X over the item's image.

See Also

ImageIndex (see page 4261)

StateIndex (see page 4267)

StateImages (see page 3828)

ViewStyle (see page 3829)

27.1.33.5.15 TListItem.Owner Property

Specifies the TListItems (see page 4270) object that owns the list item.

Pascal

```
property Owner: TListItems;
```

C++

```
__property TListItems Owner;
```

Description

Use Owner to access the TListItems (see page 4270) object that manages all the list items for the list view. Use the methods of the Owner object to temporarily prevent updates to the image of the list item while it is changed or to add or delete list items.

Do not confuse the Owner property with the list view control that displays the item. The ListView (see page 4265) property specifies the list view control that owns the TListItems (see page 4270) object.

See Also

ListView (see page 4265)

27.1.33.5.16 TListItem.Position Property

Specifies the position of the list item.

Pascal

```
property Position: TPoint;
```

C++

```
__property TPoint Position;
```

Description

Use Position to get or set the position of the list item. Position indicates the position, in client coordinates, of the list item's icon in the client area of the parent list view.

Note: Position has no effect unless the list view has a ViewStyle of vsIcon or vsSmallIcon.

See Also

ViewStyle (see page 3829)

27.1.33.5.17 TListItem.Selected Property

Indicates whether the list item is selected.

Pascal

```
property Selected: Boolean;
```

C++

```
__property Boolean Selected;
```

Description

Use Selected to select or unselect the list item. If the MultiSelect property of the list view is false, setting Selected to true sets the Selected property of all other list items in the list view to false. The appearance of a selected item depends on whether it has focus and whether the system colors are used for selection.

See Also

Focused (see page 4260)

MultiSelect (see page 3820)

27.1.33.5.18 TListItem.StateIndex Property

Specifies which image from the StateImages image list to display for the item.

Pascal

```
property StateIndex: TImageIndex;
```

C++

```
__property TImageIndex StateIndex;
```

Description

Set StateIndex to associate the list item with one of the images in the StateImages property of the list view, where 0 specifies the first image. Change StateIndex to display a different image when the state of the item changes.

The state image appears to the left of the normal list item.

See Also

ImageIndex (see page 4261)

StateImages (see page 3828)

27.1.33.5.19 TListItem.SubItemImages Property

Indicates which images (if any) should appear next to subitems of the item.

Pascal

```
property SubItemImages [Index: Integer]: Integer;
```

C++

```
__property int SubItemImages[int Index];
```

Description

Use SubItemImages to associate the subitems of this list item with images from the list view's SmallImages property, where 0 identifies the first image in SmallImages, 1 identifies the second image, and so on.

The Index (see page 4264) parameter identifies the subitem. It corresponds to the index of each subitem string in the SubItems (see page 4268) property.

The value of any subitem image index can be overridden in an OnGetSubItemImage event handler.

Note: Subitems are only visible when the list view's ViewStyle property is set to vsReport.

See Also

SmallImages (see page 3825)

SubItems (see page 4268)

OnGetSubItemImage (see page 3850)

27.1.33.5.20 TListItem.SubItems Property

Contains any strings that appear as subitems to the list item.

Pascal

```
property SubItems: TStrings;
```

C++

```
__property TStrings SubItems;
```

Description

Use SubItems to add strings that appear in the additional columns to the right of the list view item when the list view's ViewStyle property is vsReport. The TStrings object that implements the SubItems property is created when the list item is created. To the methods of the SubItems object to add or delete subitems.

Note: Subitems are only visible when the list view's ViewStyle property is set to vsReport.

C++ Examples:

```
/*
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses defs file.
```

```
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const char Names[6][2][10] =
   {{"Rubble","Barny"},
    {"Michael", "Johnson"},
    {"Bunny", "Bugs"},
    {"Silver", "HiHo"},
    {"Simpson", "Bart"},
    {"Squirrel", "Rocky"}};

  TListColumn  *NewColumn;
  TListItem  *ListItem;
  TListView   *ListView = new TListView(this);

  ListView->Parent = this;
  ListView->Align = alClient;
  ListView->ViewStyle = vsReport;
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "Last";
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "First";
  for (int i = 0; i < 6; i++)
  {
    ListItem = ListView->Items->Add();
    ListItem->Caption = Names[i][0];
    ListItem->SubItems->Add(Names[i][1]);
  }
}
```

Delphi Examples:

```
{
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses clause of the unit file.
}
procedure TForm1.FormCreate(Sender: TObject);
const
  Names: array[0..5, 0..1] of string = (
    ('Rubble', 'Barney'),
    ('Michael', 'Johnson'),
    ('Bunny', 'Bugs'),
    ('Silver', 'HiHo'),
    ('Simpson', 'Bart'),
    ('Squirrel', 'Rocky')
    );

var
  I: Integer;
  NewColumn: TListColumn;
  ListItem: TListItem;
  ListView: TListView;
begin
  ListView := TListView.Create(Self);
  with ListView do
  begin
    Parent := Self;
    Align := alClient;
    ViewStyle := vsReport;

    NewColumn := Columns.Add;
    NewColumn.Caption := 'Last';
    NewColumn := Columns.Add;
    NewColumn.Caption := 'First';

    for I := Low(Names) to High(Names) do
```

```
      begin
        ListItem := Items.Add;
        ListItem.Caption := Names[I][0];
        ListItem.SubItems.Add(Names[I][1]);
      end;
    end;
  end;
```

See Also

ViewStyle (see page 3829)

27.1.33.5.21 TListItem.Top Property

Specifies the distance, in pixels, from the top of the list view to the top of the list item.

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

Use Top to vertically position the list item within the list view. Only use the Top property when the ViewStyle property of the list view is vsIcon or vsSmallIcon.

See Also

DisplayRect (see page 4254)

GetPosition (see page 4255)

Left (see page 4265)

SetPosition (see page 4256)

ViewStyle (see page 3829)

27.1.34 TListItems Class

TListItems maintains the collection of items that appear in a list view control.

Class Hierarchy

Pascal

```
TListItems = class(TPersistent);
```

C++

```
class TListItems : public TPersistent;
```

File

ComCtrls

Description

Use the properties and methods of TListItems to manipulate the list of items displayed by a list view control. TListItems only lists the main items in the list view. Subitems are specified using a property of the individual TListItem (see page 4250) objects.

See Also

TListItem (see page 4250)

TListView (see page 4285)

27.1.34.1 TListItems Members

The following tables list the members exposed by TListItems.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CreateItem (see page 4276)	This is CreateItem, a member of class TListItems.
	DefineProperties (see page 4277)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetCount (see page 4278)	This is GetCount, a member of class TListItems.
	GetHandle (see page 4278)	This is GetHandle, a member of class TListItems.
	GetItem (see page 4278)	This is GetItem, a member of class TListItems.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCount (see page 4280)	Sets the number of list items.
	SetItem (see page 4280)	This is SetItem, a member of class TListItems.
	SetUpdateState (see page 4281)	This is SetUpdateState, a member of class TListItems.

Public Methods

	Name	Description
	Add (see page 4273)	Creates a new list item and adds it to the list view control.
	AddItem (see page 4275)	Adds an item in a specified location.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 4275)	Copies the list of list items from another TListItems (see page 4270) object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 4276)	Prevents updating of the list view until the EndUpdate (see page 4277) method is called.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 4276)	Removes all items from the list.
	Create (see page 4272)	Creates an instance of TListItems (see page 4270).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 4277)	Deletes a specified item and updates the list view accordingly.
	Destroy (see page 4272)	Destroys an instance of TListItems (see page 4270).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 4277)	Reenables screen repainting that was turned off with the BeginUpdate (see page 4276) method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 4278)	This is GetEnumerator, a member of class TListItems.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	IndexOf (see page 4278)	Returns the position of a list item in the list view window.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 4279)	Creates a new list item and inserts it into the list view.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Count (see page 4281)	Indicates the number of items in the Item (see page 4282) property array.
R	Handle (see page 4281)	Specifies the window handle for the list view that displays the items managed by TListItems (see page 4270).
	Item (see page 4282)	Lists all list items managed by the TListItems (see page 4270) object.
R	Owner (see page 4283)	Indicates the list view object that uses the list items object to manage its items.

27.1.34.2 TListItems.Create Constructor

Creates an instance of TListItems (see page 4270).

Pascal

```
constructor Create(AOwner: TCustomListView);
```

C++

```
__fastcall TListItems(TCustomListView AOwner);
```

Description

Create instantiates a TListItems (see page 4270) object. This constructor is rarely used in application code, because the list view object creates the list items object as needed.

See Also

Items (see page 3814)

27.1.34.3 TListItems.Destroy Destructor

Destroys an instance of TListItems (see page 4270).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TListItems();
```

Description

Do not call Destroy directly. Instead, call Free, which verifies that the TListItems (see page 4270) reference is not nil before calling Destroy

Destroy~TListItems (see page 4270) frees the list before calling the inherited destructor.

See Also

Free (see page 22300)

27.1.34.4 TListItems Methods

The methods of the TListItems class are listed here.

Protected Methods

	Name	Description
	CreateItem (see page 4276)	This is CreateItem, a member of class TListItems.
	DefineProperties (see page 4277)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetCount (see page 4278)	This is GetCount, a member of class TListItems.
	GetHandle (see page 4278)	This is GetHandle, a member of class TListItems.
	GetItem (see page 4278)	This is GetItem, a member of class TListItems.
	SetCount (see page 4280)	Sets the number of list items.
	SetItem (see page 4280)	This is SetItem, a member of class TListItems.
	SetUpdateState (see page 4281)	This is SetUpdateState, a member of class TListItems.

Public Methods

	Name	Description
	Add (see page 4273)	Creates a new list item and adds it to the list view control.
	AddItem (see page 4275)	Adds an item in a specified location.
	Assign (see page 4275)	Copies the list of list items from another TListItems (see page 4270) object.
	BeginUpdate (see page 4276)	Prevents updating of the list view until the EndUpdate (see page 4277) method is called.
	Clear (see page 4276)	Removes all items from the list.
	Delete (see page 4277)	Deletes a specified item and updates the list view accordingly.
	EndUpdate (see page 4277)	Reenables screen repainting that was turned off with the BeginUpdate (see page 4276) method.
	GetEnumerator (see page 4278)	This is GetEnumerator, a member of class TListItems.
	IndexOf (see page 4278)	Returns the position of a list item in the list view window.
	Insert (see page 4279)	Creates a new list item and inserts it into the list view.

27.1.34.4.1 TListItems.Add Method

Creates a new list item and adds it to the list view control.

Pascal

```
function Add: TListItem;
```

C++

```
__fastcall TListItem Add();
```

Description

Call Add to add a new list item to the end of the list. Add returns the newly created TListItem (see page 4250) object.

To create an item and insert it into the beginning or middle of the list, use the Insert (see page 4279) method instead.

C++ Examples:

```
/*
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
```

```
You must add comctrls to the uses defs file.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const char Names[6][2][10] =
   {{"Rubble","Barny"},
    {"Michael", "Johnson"},
    {"Bunny", "Bugs"},
    {"Silver", "HiHo"},
    {"Simpson", "Bart"},
    {"Squirrel", "Rocky"}};

  TListColumn  *NewColumn;
  TListItem  *ListItem;
  TListView   *ListView = new TListView(this);

  ListView->Parent = this;
  ListView->Align = alClient;
  ListView->ViewStyle = vsReport;
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "Last";
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "First";
  for (int i = 0; i < 6; i++)
  {
    ListItem = ListView->Items->Add();
    ListItem->Caption = Names[i][0];
    ListItem->SubItems->Add(Names[i][1]);
  }
}
```

Delphi Examples:

```
{
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses clause of the unit file.
}
procedure TForm1.FormCreate(Sender: TObject);
const
  Names: array[0..5, 0..1] of string = (
    ('Rubble', 'Barney'),
    ('Michael', 'Johnson'),
    ('Bunny', 'Bugs'),
    ('Silver', 'HiHo'),
    ('Simpson', 'Bart'),
    ('Squirrel', 'Rocky')
    );

var
  I: Integer;
  NewColumn: TListColumn;
  ListItem: TListItem;
  ListView: TListView;
begin
  ListView := TListView.Create(Self);
  with ListView do
  begin
    Parent := Self;
    Align := alClient;
    ViewStyle := vsReport;

    NewColumn := Columns.Add;
    NewColumn.Caption := 'Last';
    NewColumn := Columns.Add;
    NewColumn.Caption := 'First';
```

```
      for I := Low(Names) to High(Names) do
      begin
        ListItem := Items.Add;
        ListItem.Caption := Names[I][0];
        ListItem.SubItems.Add(Names[I][1]);
      end;
    end;
  end;
```

See Also

AddItem (see page 4275)

Delete (see page 4277)

Insert (see page 4279)

27.1.34.4.2 TListItems.AddItem Method

Adds an item in a specified location.

Pascal

```
function AddItem(Item: TListItem; Index: Integer = -1): TListItem;
```

C++

```
__fastcall TListItem AddItem(TListItem Item, int Index = -1);
```

Description

Call AddItem to add an Item (see page 4282) at any place in the list. The properties of Item (see page 4282) are duplicated if Item (see page 4282) is not nil. Index is location of the added item; if Index is negative, the TListItem (see page 4250) is appended to the end of the list.

AddItem returns the TListItem (see page 4250) that was added.

See Also

Add (see page 4273)

Delete

Insert (see page 4279)

27.1.34.4.3 TListItems.Assign Method

Copies the list of list items from another TListItems (see page 4270) object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the list of items from another object. If the Source parameter specifies another TListItems (see page 4270) object, Assign discards the current value of Item (see page 4282) and rebuilds the list by copying the items maintained by Source. If Source is any other type of object, Assign calls the inherited method, which will copy the list from any object that specifies how to copy to a TListItems (see page 4270) object in its AssignTo method.

See Also

Item (see page 4282)

AssignTo (see page 2931)

27.1.34.4.4 TListItems.BeginUpdate Method

Prevents updating of the list view until the EndUpdate (see page 4277) method is called.

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

Call BeginUpdate before making multiple changes to the list of items. When all changes are complete, call EndUpdate (see page 4277) so that the changes can be reflected on screen. BeginUpdate and EndUpdate (see page 4277) prevent excessive redraws and speed processing time when new items are added, deleted, or inserted.

See Also

EndUpdate (see page 4277)

27.1.34.4.5 TListItems.Clear Method

Removes all items from the list.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the list view of all items. Clear frees all memory associated with the list items and updates the image of the list view.

See Also

Delete (see page 4277)

27.1.34.4.6 TListItems.CreateItem Method

Pascal

```
function CreateItem(Index: Integer; ListItem: TListItem): TLVItem;
```

C++

```
__fastcall TLVItem CreateItem(int Index, TListItem ListItem);
```

Description

This is CreateItem, a member of class TListItems.

27.1.34.4.7 TListItems.DefineProperties Method

Provides an interface for a method that reads and writes otherwise unpublished data.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TListItems (see page 4270) overrides DefineProperties in order to persist individual items.

See Also

DefineProperties (see page 2931)

27.1.34.4.8 TListItems.Delete Method

Deletes a specified item and updates the list view accordingly.

Pascal

```
procedure Delete(Index: Integer);
```

C++

```
__fastcall Delete(int Index);
```

Description

The Delete method deletes the item in the list view specified with the Index parameter. The first item has an index of 0, the second item has an index of 1, and so on.

See Also

Add (see page 4273)

Clear (see page 4276)

Insert (see page 4279)

Item (see page 4282)

27.1.34.4.9 TListItems.EndUpdate Method

Reenables screen repainting that was turned off with the BeginUpdate (see page 4276) method.

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

Call EndUpdate after completing changes to the list that were begun with a call to the BeginUpdate (see page 4276) method. When every call to BeginUpdate (see page 4276) is matched by a subsequent call to EndUpdate, the list view image updates to reflect all changes that occurred while repaints were disabled.

See Also

BeginUpdate (see page 4276)

27.1.34.4.10 TListItems.GetCount Method

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

This is GetCount, a member of class TListItems.

27.1.34.4.11 TListItems.GetEnumerator Method

Pascal

```
function GetEnumerator: TListItemsEnumerator;
```

C++

```
__fastcall TListItemsEnumerator GetEnumerator();
```

Description

This is GetEnumerator, a member of class TListItems.

27.1.34.4.12 TListItems.GetHandle Method

Pascal

```
function GetHandle: HWND;
```

C++

```
__fastcall HWND GetHandle();
```

Description

This is GetHandle, a member of class TListItems.

27.1.34.4.13 TListItems.GetItem Method

Pascal

```
function GetItem(Index: Integer): TListItem;
```

C++

```
__fastcall TListItem GetItem(int Index);
```

Description

This is GetItem, a member of class TListItems.

27.1.34.4.14 TListItems.IndexOf Method

Returns the position of a list item in the list view window.

Pascal

```
function IndexOf(Value: TListItem): Integer;
```

C++

```
__fastcall int IndexOf(TListItem Value);
```

Description

Call IndexOf to determine the position of the list item specified by the Value parameter. IndexOf returns -1 if the list item is not in the list view.

See Also

Item (see page 4282)

TListItem (see page 4250)

27.1.34.4.15 TListItems.Insert Method

Creates a new list item and inserts it into the list view.

Pascal

```
function Insert(Index: Integer): TListItem;
```

C++

```
__fastcall TListItem Insert(int Index);
```

Description

Call Insert to insert a new TListItem (see page 4250) object into the list view at the specified Index. Insert returns the list item that is inserted.

C++ Examples:

```
/*
The following example inserts a list item after the selected
item in the list view.  Place a TButton, a TEdit and a
TListView in the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TListItem *InsertItem;
  if (ListView1->Selected == NULL) return;
  InsertItem =
    ListView1->Items->Insert(ListView1->Selected->Index);
  InsertItem->Caption = Edit1->Text;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *ListItem;
  ListView1->ViewStyle = vsList;
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Apples";
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Oranges";
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Pears";
}
```

Delphi Examples:

```
{
The following example inserts a list item after the selected
```

```
  item in the list view.  Place a TButton, a TEdit and a
  TListView in the form.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var InsertItem : TListItem;
  begin
    with ListView1 do
    begin
      if (Selected = nil) then exit;
      InsertItem := Items.Insert(Selected.Index);
    end;
    InsertItem.Caption := Edit1.Text;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var ListItem : TListItem;
  begin
    ListView1.ViewStyle := vsList;
    ListItem := ListView1.Items.Add;
    ListItem.Caption := 'Apples';
    ListItem := ListView1.Items.Add;
    ListItem.Caption := 'Oranges';
    ListItem := ListView1.Items.Add;
    ListItem.Caption := 'Pears';
  end;
```

See Also

Add (see page 4273)

Delete (see page 4277)

27.1.34.4.16 TListItems.SetCount Method

Sets the number of list items.

Pascal

```
procedure SetCount(Value: Integer);
```

C++

```
__fastcall SetCount(int Value);
```

Description

Call SetCount to set the number of list items.

27.1.34.4.17 TListItems.SetItem Method

Pascal

```
procedure SetItem(Index: Integer; Value: TListItem);
```

C++

```
__fastcall SetItem(int Index, TListItem Value);
```

Description

This is SetItem, a member of class TListItems.

27.1.34.4.18 TListItems.SetUpdateState Method

Pascal

```
procedure SetUpdateState(Updating: Boolean);
```

C++

```
__fastcall SetUpdateState(Boolean Updating);
```

Description

This is SetUpdateState, a member of class TListItems.

27.1.34.5 TListItems Properties

The properties of the TListItems class are listed here.

Public Properties

	Name	Description
	Count (see page 4281)	Indicates the number of items in the Item (see page 4282) property array.
R	Handle (see page 4281)	Specifies the window handle for the list view that displays the items managed by TListItems (see page 4270).
	Item (see page 4282)	Lists all list items managed by the TListItems (see page 4270) object.
R	Owner (see page 4283)	Indicates the list view object that uses the list items object to manage its items.

27.1.34.5.1 TListItems.Count Property

Indicates the number of items in the Item (see page 4282) property array.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use Count to determine the number of items displayed by the list view. Count provides an upper bound when iterating through the individual TListItem (see page 4250) objects in the Item (see page 4282) property array.

Set the Count when the list view's OwnerData property is false to make the control allocate its internal data structures for the specified number of items. This prevents the control from having to allocate the data structures every time an item is added.

Set the Count when the list view's OwnerData property is true to set the virtual number of items that the control contains.

See Also

Item (see page 4282)

27.1.34.5.2 TListItems.Handle Property

Specifies the window handle for the list view that displays the items managed by TListItems (see page 4270).

Pascal

```
property Handle: HWND;
```

C++

```
__property HWND Handle;
```

Description

Use Handle as a parameter for calls to the underlying API that require a window handle.

See Also

Owner (see page 4283)

27.1.34.5.3 TListItems.Item Property

Lists all list items managed by the TListItems (see page 4270) object.

Pascal

```
property Item [Index: Integer]: TListItem;
```

C++

```
__property TListItem Item[int Index];
```

Description

Use Item to directly access a list item, given its position in the list view. The first item has an index of 0, the second an index of 1, and so on.

C++ Examples:

```
/*
The following code uses Item to add the labels of all child
nodes of the selected node to a list box when a button is
clicked.  This example requires a button, a listbox, and a
populated treeview.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < TreeView1->Selected->Count; i++)
    ListBox1->Items->Add(TreeView1->Selected->Item[i]->Text);
}
```

Delphi Examples:

```
{
The following code uses Item to add the labels of all child
nodes of the selected node to a list box when a button is
clicked.  This example requires a button, a listbox, and a
populated treeview.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I : Integer;
begin
  for I := 0 to (TreeView1.Selected.Count - 1) do
    ListBox1.Items.Add(TreeView1.Selected.Item[I].Text);
end;
```

See Also

Count (see page 4281)

IndexOf (see page 4278)

operator_[]

27.1.34.5.4 TListItems.Owner Property

Indicates the list view object that uses the list items object to manage its items.

Pascal

```
property Owner: TCustomListView;
```

C++

```
__property TCustomListView Owner;
```

Description

Use Owner to access the list view object that uses this TListItems (see page 4270) object to implement its Items property.

See Also

Items (see page 3814)

27.1.35 TListItemsEnumerator Class

Class Hierarchy

ComCtrls.TListItemsEnumerator

Pascal

```
TListItemsEnumerator = class;
```

C++

```
class TListItemsEnumerator;
```

File

ComCtrls

Description

This is class ComCtrls.TListItemsEnumerator.

27.1.35.1 TListItemsEnumerator Members

The following tables list the members exposed by TListItemsEnumerator.

Public Methods

	Name	Description
	Create (see page 4284)	This is Create, a member of class TListItemsEnumerator.
	GetCurrent (see page 4284)	This is GetCurrent, a member of class TListItemsEnumerator.
	MoveNext (see page 4284)	This is MoveNext, a member of class TListItemsEnumerator.

Public Properties

	Name	Description
R	Current (see page 4285)	This is Current, a member of class TListItemsEnumerator.

27.1.35.2 TListItemsEnumerator.Create Constructor

Pascal

```
constructor Create(AListItems: TListItems);
```

C++

```
__fastcall TListItemsEnumerator(TListItems AListItems);
```

Description

This is Create, a member of class TListItemsEnumerator.

27.1.35.3 TListItemsEnumerator Methods

The methods of the TListItemsEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 4284)	This is GetCurrent, a member of class TListItemsEnumerator.
	MoveNext (see page 4284)	This is MoveNext, a member of class TListItemsEnumerator.

27.1.35.3.1 TListItemsEnumerator.GetCurrent Method

Pascal

```
function GetCurrent: TListItem;
```

C++

```
__fastcall TListItem GetCurrent();
```

Description

This is GetCurrent, a member of class TListItemsEnumerator.

27.1.35.3.2 TListItemsEnumerator.MoveNext Method

Pascal

```
function MoveNext: Boolean;
```

C++

```
__fastcall Boolean MoveNext();
```

Description

This is MoveNext, a member of class TListItemsEnumerator.

27.1.35.4 TListItemsEnumerator Properties

The properties of the TListItemsEnumerator class are listed here.

Public Properties

	Name	Description
R	Current (see page 4285)	This is Current, a member of class TListItemsEnumerator.

27.1.35.4.1 TListItemsEnumerator.Current Property

Pascal

```
property Current: TListItem;
```

C++

```
__property TListItem Current;
```

Description

This is Current, a member of class TListItemsEnumerator.

27.1.36 TListView Class

TListView displays a list of items in various ways.

Class Hierarchy

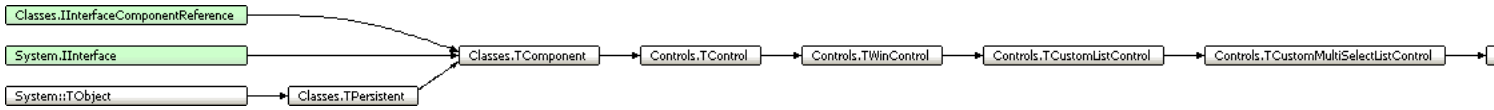

Pascal

```
TListView = class(TCustomListView);
```

C++

```
class TListView : public TCustomListView;
```

File

ComCtrls

Description

Use TListView to manage and display a list of items in a form. The items can be displayed in columns with column headers and sub-items, or vertically or horizontally, with small or large icons.

TListView publishes many of the properties, events, and methods of TCustomListView (see page 3757), but does not introduce any new behavior.

See Also

TListColumn (see page 4236)

TListItem (see page 4250)

List Controls

List Views

27.1.36.1 TListView Members

The following tables list the members exposed by TListView.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnCreateItemClass (see page 3844)	Occurs when a list item object is about to be created.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FMultiSelect (see page 5956)	This is FMultiSelect, a member of class TCustomMultiSelectListControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActionChange (see page 3773)	Responds when the action for which this list view is a client changes.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.

	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanChange (see page 3777)	Specifies whether the properties of a list item can be changed.
	CanEdit (see page 3778)	Specifies whether the text of a list item can be changed by the user.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 3778)	Generates an OnChange (see page 3836) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 3778)	Rescaling event dispatcher
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ColClick (see page 3780)	Column (see page 3806) click event dispatcher.
	ColRightClick (see page 3780)	Column (see page 3806) right-click event dispatcher.
	ColumnsShowing (see page 3780)	Indicates whether the ViewStyle (see page 3829) property is vsReport.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateListItem (see page 3781)	Creates a list item for the collection specified by the Items (see page 3814) property.
	CreateListItems (see page 3782)	Creates the TListItems (see page 4270) object that implements the Items (see page 3814) property.
	CreateParams (see page 3782)	Initializes the window-creation parameter record.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 3782)	Creates the window used by the list view.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	CustomDraw (see page 3783)	Custom draw event dispatcher.
	CustomDrawItem (see page 3783)	Custom draw item event dispatcher.
	CustomDrawSubItem (see page 3783)	Custom draw subitem event dispatcher.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	Delete (see page 3785)	Removes an item from the list view and frees the associated memory.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 3786)	Destroys the window referenced by the list view's window handle.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.

	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
V	DoEndDrag (see page 3787)	Drag end event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
V	DoInfoTip (see page 3787)	InfoTip event dispatcher.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
V	DoStartDrag (see page 3787)	Start drag event dispatcher..
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawItem (see page 3788)	Draw item event dispatcher.
	Edit (see page 3788)	Edit event dispatcher.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
V	GetActionLinkClass (see page 3789)	Returns the action link class that is associated with list view controls.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetCount (see page 3789)	Returns the number of items in the list.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 3790)	Returns a drag cursor if a list item is being dragged.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.

V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
V	GetItemIndex (see page 3791)	This is the overview for the GetItemIndex method overload.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
V	GetSelCount (see page 3794)	Returns the value of the SelCount (see page 3822) property.
A	GetSelCount (see page 5956)	Returns the value of the SelCount (see page 5957) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InsertItem (see page 3794)	Generates an OnInsert (see page 3851) event.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
V	IsCustomDrawn (see page 3794)	Determines when custom draw event handlers are called.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
V	MouseUp (see page 3795)	Mouse button up event dispatcher.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 3796)	Responds to notifications that components are about to be created or destroyed.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	OwnerDataFetch (see page 3796)	Customizes a list view item before it is displayed.
V	OwnerDataFind (see page 3797)	Searches for an item in the list view control.
V	OwnerDataHint (see page 3797)	Specifies a range of items that might be fetched.
V	OwnerDataStateChange (see page 3798)	Specifies items that have changed states.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.

	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetItemIndex (see page 3799)	Sets the value of the ItemIndex property.
A	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
V	SetMultiSelect (see page 3800)	Sets the value of the MultiSelect (see page 3820) property.
A	SetMultiSelect (see page 5956)	Sets the value of the MultiSelect (see page 5957) property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetViewStyle (see page 3800)	Sets the value of the ViewStyle (see page 3829) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.

	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateColumn (see page 3801)	Individual column update event dispatcher.
	UpdateColumns (see page 3801)	General column update event dispatcher.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 3802)	Provides specific message responses for the list view.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AddItem (see page 3773)	Adds an item to the list control.

	AddItem (see page 5940)	Adds an item to the list control.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AlphaSort (see page 3774)	Sorts the items in the list view.
	Arrange (see page 3777)	Arranges icons in a list view that has a ViewStyle (see page 3829) of vsIcon or vsSmallIcon.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 3779)	Deletes all of the items from the list control.
	Clear (see page 5940)	Deletes all of the items from the list control.
	ClearSelection (see page 3779)	Removes the selection, leaving all items unselected.
	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopySelection (see page 3781)	Copies the selected item or items to another TCustomListControl descendant.
	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 3770)	Creates and initializes an instance of TCustomListView (see page 3757).
	Create (see page 5731)	Creates an instance of TControl (see page 5724).
	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CustomSort (see page 3784)	Sorts the items in the list using the specified ordering function.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DeleteSelected (see page 3786)	Deletes all the selected items.
	DeleteSelected (see page 5941)	Deletes all the selected items.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 3771)	Destroys an instance of TCustomListView (see page 3757).
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindCaption (see page 3788)	Returns a list view item, if any, with the specified caption.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindData (see page 3789)	Returns a list view item, if any, associated with the specified data.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHitTestInfoAt (see page 3790)	Returns information about the specified point in the list view's client area.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItemAt (see page 3790)	Returns the list item at the specified position.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNearestItem (see page 3791)	Returns the list item nearest a specified point, in a particular direction.
	GetNextItem (see page 3792)	Returns the next list item after the StartItem in the specified direction.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetSearchString (see page 3793)	Returns the incremental search string of a list view control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsEditing (see page 3795)	Indicates whether any item in the list view is being edited.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	Scroll (see page 3798)	Scrolls the content of a list view by the indicated amount.
	ScrollBy (see page 6176)	Scroll control contents.
V	SelectAll (see page 3799)	Selects all items or all text in the selected item.
A	SelectAll (see page 5943)	Selects all items or all text in the selected item.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	StringWidth (see page 3800)	Returns the width of a string, using the list view's current font.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UpdateItems (see page 3801)	Redraws a range of items, starting at FirstIndex and ending at LastIndex.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
R	BoundingRect (see page 3804)	Denotes the bounding rectangle for the items in the list view.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 3804)	Gives access to the list view's canvas.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	Column (see page 3806)	Describes the column specified by the Index parameter.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropTarget (see page 3810)	Specifies which item in the list is the target of a drag and drop operation.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemFocused (see page 3813)	Indicates which item, if any, has focus.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.

R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	SelCount (see page 3822)	Specifies the number of selected items in the list view.
R	SelCount (see page 5957)	Indicates the number of selected items.
	Selected (see page 3822)	Indicates the first selected item in the list view.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
R	TopItem (see page 3828)	Indicates the topmost visible item in the list view.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	ViewOrigin (see page 3829)	Designates the coordinates of the upper left corner of the list, relative to the visible area of the list view control.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.
R	VisibleRowCount (see page 3832)	Indicates number of items that can fit vertically in the visible area of a list view.
R	WorkAreas (see page 3833)	Defines separate working areas that divide the list view's client area.

Published Events

	Name	Description
	OnAdvancedCustomDraw (see page 4346)	Occurs at discrete points during the rendering of the list view.
	OnAdvancedCustomDrawItem (see page 4347)	Occurs at discrete stages during the painting of an item in an owner-draw list view.
	OnAdvancedCustomDrawSubItem (see page 4347)	Occurs at discrete stages during the painting of a subitem in an owner-drawn list view.
	OnChange (see page 4348)	Occurs immediately after an item in the list changes.
	OnChanging (see page 4349)	Occurs when an item in the list is about to change.
	OnClick (see page 4349)	Occurs when the user clicks the control.
	OnColumnClick (see page 4351)	Occurs when the user clicks on a column header in a list view.
	OnColumnDragged (see page 4352)	Occurs when a column is dragged to a new position.
	OnColumnRightClick (see page 4352)	Occurs when the user right clicks on a column header in a list view.
	OnCompare (see page 4353)	Occurs to when two items need to be compared during a sort of the list.
	OnContextPopup (see page 4353)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCustomDraw (see page 4354)	Occurs when an owner-draw list view must be rendered.
	OnCustomDrawItem (see page 4355)	Occurs when an item in an owner-draw list view must be rendered.
	OnCustomDrawSubItem (see page 4355)	Occurs when a subitem must be rendered in an owner-draw list view.
	OnData (see page 4356)	Occurs immediately before an item is displayed in the list view control.
	OnDataFind (see page 4356)	Occurs when the list view control receives a search request, such as when the FindData (see page 3789) and FindCaption (see page 3788) methods are called.
	OnDataHint (see page 4357)	Occurs when the contents of the list view change, such as when the user scrolls the list view.
	OnDataStateChange (see page 4357)	Occurs when a range of items change states.
	OnDblClick (see page 4358)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeletion (see page 4358)	Occurs when an item in the list view is about to be deleted.
	OnDragDrop (see page 4359)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4361)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 4363)	Occurs when an item should be drawn in the list view.
	OnEdited (see page 4364)	Occurs immediately after the user edits an item's Caption property.
	OnEditing (see page 4364)	Occurs when the user starts to edit a list item's Caption property.

	OnEndDock (see page 4365)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4365)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4366)	Occurs when a control receives the input focus.
	OnExit (see page 4369)	Occurs when the input focus shifts away from one control to another.
	OnGetImageIndex (see page 4370)	Occurs when an item is about to be displayed in the list view control.
	OnGetSubItemImage (see page 4371)	Occurs when a subitem is about to be displayed in the column of a list view control.
	OnInfoTip (see page 4371)	Occurs when the user pauses the mouse over an item in the list view.
	OnInsert (see page 4372)	Occurs immediately after a new item is inserted into the list view.
	OnKeyDown (see page 4372)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 4375)	Occurs when a key is pressed.
	OnKeyUp (see page 4376)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 4378)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4379)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4381)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4381)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4381)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4384)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 4386)	Occurs immediately after the control is resized.
	OnSelectItem (see page 4387)	Occurs when an item is selected in the list view control.
	OnStartDock (see page 4387)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4387)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Action (see page 4300)	Designates the action associated with the control.
	Align (see page 4300)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AllocBy (see page 4304)	Indicates the number of items the list view can store in memory.
	Anchors (see page 4304)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 4304)	Specifies which edges of the control are beveled.
	BevelInner (see page 4305)	Specifies the cut of the inner bevel.
	BevelKind (see page 4305)	Specifies the control's bevel style.
	BevelOuter (see page 4306)	Specifies the cut of the outer bevel.
	BevelWidth (see page 4306)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 4307)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 4307)	Determines whether the list view displays a single line border.
	BorderWidth (see page 4307)	Specifies the width of the control's border.
	Checkboxes (see page 4308)	Specifies whether check boxes appear next to the items in the list view.
	Color (see page 4308)	Specifies the background color of the control.
	ColumnClick (see page 4309)	Determines whether the column header behaves like a button.
	Columns (see page 4309)	Describes the properties of the columns in the list view.
	Constraints (see page 4311)	Specifies the size constraints for the control.
	Ctl3D (see page 4312)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 4312)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4312)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4313)	Determines how the control initiates drag-and-drop or drag-and-dock operations.

	Enabled (see page 4314)	Controls whether the control responds to mouse, keyboard, and timer events.
	FlatScrollBars (see page 4315)	Specifies whether the list view's scroll bars are flat.
	Font (see page 4315)	Controls the attributes of text written on or in the control.
	FullDrag (see page 4316)	Specifies whether headers are repainted when they are dragged.
	GridLines (see page 4317)	Determines whether lines are drawn separating items in the list.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	HideSelection (see page 4317)	Determines whether the list view gives a visual indication of which item is selected when focus shifts to another control.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HotTrack (see page 4317)	Specifies whether list items are highlighted when the mouse passes over them.
	HotTrackStyles (see page 4318)	Specifies the hot tracking styles.
	HoverTime (see page 4318)	Specifies how long the user must pause with the mouse before an item is selected.
	IconOptions (see page 4318)	Determines how icons displayed in the list view are arranged.
	Items (see page 4319)	Contains the list of items displayed by the list view.
	LargeImages (see page 4322)	Provides a list of icon images to display for each item in the list when ViewStyle is vsIcon.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	MultiSelect (see page 4324)	Determines whether the user can select more than one list item at a time.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OwnerData (see page 4325)	Specifies whether the list view control is virtual.
	OwnerDraw (see page 4325)	Specifies whether the list view receives an OnDrawItem event.
	ParentBiDiMode (see page 4326)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 4326)	Determines where a control looks for its color information.
	ParentFont (see page 4327)	Determines where a control looks for its font information.
	ParentShowHint (see page 4327)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4328)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 4332)	Determines whether the user can change the contents of the control.
	RowSelect (see page 4333)	Determines whether an entire row in the list view can be selected.
	ShowColumnHeaders (see page 4333)	Determines whether the column headers are displayed.
	ShowHint (see page 4334)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	ShowWorkAreas (see page 4335)	Determines whether the visual properties of work areas are displayed.
	SmallImages (see page 4336)	Provides a list of icon images to display for each item in the list when ViewStyle is not vsIcon.
	SortType (see page 4338)	Determines if and how the items in the list are automatically sorted.
	StateImages (see page 4339)	Specifies additional icon images to display next to each item in the list view.
	TabOrder (see page 4339)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 4340)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	ViewStyle (see page 4340)	Determines how the list items are displayed.
	Visible (see page 4343)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.36.2 TListView Properties

The properties of the TListView class are listed here.

Published Properties

	Name	Description
	Action (see page 4300)	Designates the action associated with the control.
	Align (see page 4300)	Determines how the control aligns within its container (parent control).
	AllocBy (see page 4304)	Indicates the number of items the list view can store in memory.
	Anchors (see page 4304)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 4304)	Specifies which edges of the control are beveled.
	BevelInner (see page 4305)	Specifies the cut of the inner bevel.
	BevelKind (see page 4305)	Specifies the control's bevel style.
	BevelOuter (see page 4306)	Specifies the cut of the outer bevel.
	BevelWidth (see page 4306)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 4307)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 4307)	Determines whether the list view displays a single line border.
	BorderWidth (see page 4307)	Specifies the width of the control's border.
	Checkboxes (see page 4308)	Specifies whether check boxes appear next to the items in the list view.
	Color (see page 4308)	Specifies the background color of the control.
	ColumnClick (see page 4309)	Determines whether the column header behaves like a button.
	Columns (see page 4309)	Describes the properties of the columns in the list view.
	Constraints (see page 4311)	Specifies the size constraints for the control.
	Ctl3D (see page 4312)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 4312)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4312)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4313)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4314)	Controls whether the control responds to mouse, keyboard, and timer events.
	FlatScrollBars (see page 4315)	Specifies whether the list view's scroll bars are flat.
	Font (see page 4315)	Controls the attributes of text written on or in the control.
	FullDrag (see page 4316)	Specifies whether headers are repainted when they are dragged.
	GridLines (see page 4317)	Determines whether lines are drawn separating items in the list.
	HideSelection (see page 4317)	Determines whether the list view gives a visual indication of which item is selected when focus shifts to another control.
	HotTrack (see page 4317)	Specifies whether list items are highlighted when the mouse passes over them.
	HotTrackStyles (see page 4318)	Specifies the hot tracking styles.
	HoverTime (see page 4318)	Specifies how long the user must pause with the mouse before an item is selected.
	IconOptions (see page 4318)	Determines how icons displayed in the list view are arranged.
	Items (see page 4319)	Contains the list of items displayed by the list view.
	LargeImages (see page 4322)	Provides a list of icon images to display for each item in the list when ViewStyle is vsIcon.
	MultiSelect (see page 4324)	Determines whether the user can select more than one list item at a time.
	OwnerData (see page 4325)	Specifies whether the list view control is virtual.
	OwnerDraw (see page 4325)	Specifies whether the list view receives an OnDrawItem event.
	ParentBiDiMode (see page 4326)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 4326)	Determines where a control looks for its color information.
	ParentFont (see page 4327)	Determines where a control looks for its font information.
	ParentShowHint (see page 4327)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4328)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 4332)	Determines whether the user can change the contents of the control.
	RowSelect (see page 4333)	Determines whether an entire row in the list view can be selected.
	ShowColumnHeaders (see page 4333)	Determines whether the column headers are displayed.
	ShowHint (see page 4334)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	ShowWorkAreas (see page 4335)	Determines whether the visual properties of work areas are displayed.
	SmallImages (see page 4336)	Provides a list of icon images to display for each item in the list when ViewStyle is not vsIcon.

	SortType (see page 4338)	Determines if and how the items in the list are automatically sorted.
	StateImages (see page 4339)	Specifies additional icon images to display next to each item in the list view.
	TabOrder (see page 4339)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 4340)	Determines if the user can tab to a control.
	ViewStyle (see page 4340)	Determines how the list items are displayed.
	Visible (see page 4343)	Determines whether the component appears onscreen.

27.1.36.2.1 TListView.Action Property

Designates the action associated with the control.

Pascal

```
property Action: TBasicAction;
```

C++

```
__property TBasicAction Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

27.1.36.2.2 TListView.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
```

```
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
```

```
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

27.1.36.2.3 TListView.AllocBy Property

Indicates the number of items the list view can store in memory.

Pascal

```
property AllocBy: Integer;
```

C++

```
__property int AllocBy;
```

Description

Set AllocBy to the total number of items in the list before adding a large number of items to the list view. AllocBy allows the list view to allocate the memory for items in a single block at once rather than bit by bit as items are added.

See Also

Items (see page 3814)

27.1.36.2.4 TListView.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.36.2.5 TListView.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

27.1.36.2.6 TListView.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

27.1.36.2.7 TListView.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

27.1.36.2.8 TListView.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

27.1.36.2.9 TListView.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

27.1.36.2.10 TListView.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

27.1.36.2.11 TListView.BorderStyle Property

Determines whether the list view displays a single line border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the list view control stands out on a form. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the list view is drawn with a single-line border. If BorderStyle is bsNone, there will be no border.

27.1.36.2.12 TListView.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics or text drawn by the control is clipped to the area within

the border.

See Also

ClientRect

BevelOuter (see page 6194)

27.1.36.2.13 TListView.Checkboxes Property

Specifies whether check boxes appear next to the items in the list view.

Pascal

```
property Checkboxes: Boolean;
```

C++

```
__property Boolean Checkboxes;
```

Description

Set Checkboxes to true to make check boxes appear next to the list items when ViewStyle is vsList or vsReport. Each line in the list displays a single check box.

See Also

Checked (see page 4258)

ViewStyle (see page 3829)

27.1.36.2.14 TListView.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

27.1.36.2.15 TListView.ColumnClick Property

Determines whether the column header behaves like a button.

Pascal

```
property ColumnClick: Boolean;
```

C++

```
__property Boolean ColumnClick;
```

Description

Use ColumnClick to specify whether the OnColumnClick event should occur when the user clicks on the column header. When ColumnClick is true, the OnColumnClick occurs when the column header is clicked with the mouse. When ColumnClick is false, the OnColumnClick event never occurs.

Note: For column clicks to work, the column headers must be displayed. To display the headers, set the ShowColumnHeaders property to true and the ViewStyle property to vsReport.

See Also

OnColumnClick (see page 3837)

ShowColumnHeaders (see page 3824)

ViewStyle (see page 3829)

27.1.36.2.16 TListView.Columns Property

Describes the properties of the columns in the list view.

Pascal

```
property Columns: TListColumns;
```

C++

```
__property TListColumns Columns;
```

Description

Use Columns to add or delete columns in the list view, or to edit their display properties. When setting Columns at design time a ListView Columns Editor dialog appears. This dialog allows columns to be added or deleted, and the display properties of individual columns to be changed. At runtime, use the Caption, Alignment, and Width properties of the TListColumn objects

accessed by the Columns property to change the appearance of the column headers.

Note: To display the columns, set the ViewStyle property to vsReport and the ShowColumnHeaders property to true.

C++ Examples:

```
/*
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses defs file.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const char Names[6][2][10] =
   {{"Rubble","Barny"},
    {"Michael", "Johnson"},
    {"Bunny", "Bugs"},
    {"Silver", "HiHo"},
    {"Simpson", "Bart"},
    {"Squirrel", "Rocky"}};

  TListColumn  *NewColumn;
  TListItem  *ListItem;
  TListView   *ListView = new TListView(this);

  ListView->Parent = this;
  ListView->Align = alClient;
  ListView->ViewStyle = vsReport;
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "Last";
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "First";
  for (int i = 0; i < 6; i++)
  {
    ListItem = ListView->Items->Add();
    ListItem->Caption = Names[i][0];
    ListItem->SubItems->Add(Names[i][1]);
  }
}
```

Delphi Examples:

```
{
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses clause of the unit file.
}
procedure TForm1.FormCreate(Sender: TObject);
const
  Names: array[0..5, 0..1] of string = (
    ('Rubble', 'Barney'),
    ('Michael', 'Johnson'),
    ('Bunny', 'Bugs'),
    ('Silver', 'HiHo'),
    ('Simpson', 'Bart'),
    ('Squirrel', 'Rocky')
    );

var
  I: Integer;
  NewColumn: TListColumn;
  ListItem: TListItem;
  ListView: TListView;
begin
  ListView := TListView.Create(Self);
```

```
    with ListView do
    begin
      Parent := Self;
      Align := alClient;
      ViewStyle := vsReport;

      NewColumn := Columns.Add;
      NewColumn.Caption := 'Last';
      NewColumn := Columns.Add;
      NewColumn.Caption := 'First';

      for I := Low(Names) to High(Names) do
      begin
        ListItem := Items.Add;
        ListItem.Caption := Names[I][0];
        ListItem.SubItems.Add(Names[I][1]);
      end;
    end;
  end;
```

See Also

ShowColumnHeaders (see page 3824)

TListColumns (see page 4245)

TListColumn (see page 4236)

ViewStyle (see page 3829)

27.1.36.2.17 TListView.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.36.2.18 TListView.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

27.1.36.2.19 TListView.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

27.1.36.2.20 TListView.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

27.1.36.2.21 TListView.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

27.1.36.2.22 TListView.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls

appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.36.2.23 TListView.FlatScrollBars Property

Specifies whether the list view's scroll bars are flat.

Pascal

```
property FlatScrollBars: Boolean;
```

C++

```
__property Boolean FlatScrollBars;
```

Description

Use FlatScrollBars to get or set whether the list view's scroll bars are flat. Flat scroll bars do not have the beveled borders used for the three-dimensional effect.

Note: List views only support flat scroll bars if the system has version 472 or later of comctl32.dll.

27.1.36.2.24 TListView.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}
```

```
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

27.1.36.2.25 TListView.FullDrag Property

Specifies whether headers are repainted when they are dragged.

Pascal

```
property FullDrag: Boolean;
```

C++

```
__property Boolean FullDrag;
```

Description

Use FullDrag to get or set whether headers are repainted when they are dragged.

Note: To display the column headers, set the ViewStyle property to vsReport and the ShowColumnHeaders property to true.

See Also

ViewStyle (see page 3829)

Columns (see page 3808)

ShowColumnHeaders (see page 3824)

OnColumnDragged (see page 3840)

27.1.36.2.26 TListView.GridLines Property

Determines whether lines are drawn separating items in the list.

Pascal

```
property GridLines: Boolean;
```

C++

```
__property Boolean GridLines;
```

Description

Set GridLines to true to add lines that separate the items in the list view. The value of GridLines is ignored unless ViewStyle is vsReport.

See Also

ViewStyle (see page 3829)

27.1.36.2.27 TListView.HideSelection Property

Determines whether the list view gives a visual indication of which item is selected when focus shifts to another control.

Pascal

```
property HideSelection: Boolean;
```

C++

```
__property Boolean HideSelection;
```

Description

Set HideSelection to false to provide visual feedback of the selected item in the list even when the list view does not have focus. Set HideSelection to true to show the selection only when the list view has focus. HideSelection does not affect the actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms with many list views look too busy.

See Also

ItemFocused (see page 3813)

MultiSelect (see page 3820)

SelCount (see page 3822)

Selected (see page 3822)

27.1.36.2.28 TListView.HotTrack Property

Specifies whether list items are highlighted when the mouse passes over them.

Pascal

```
property HotTrack: Boolean;
```

C++

```
__property Boolean HotTrack;
```

Description

Set HotTrack to true to provide visual feedback about which item is under the mouse. When HotTrack is true, the HoverTime

property controls whether pausing the mouse over an item causes it to be selected. When HotTrack is false, there is no visual feedback when the mouse passes over items and users can't select items by pausing the mouse.

To specify hot tracking styles, use the HotTrackStyles property.

See Also

HideSelection (see page 3812)

HotTrackStyles (see page 3812)

HoverTime (see page 3813)

27.1.36.2.29 TListView.HotTrackStyles Property

Specifies the hot tracking styles.

Pascal

```
property HotTrackStyles: TListHotTrackStyles;
```

C++

```
__property TListHotTrackStyles HotTrackStyles;
```

Description

Use HotTrackStyles to specify the hot tracking styles. The hot tracking style indicates the type of feedback that the list view gives when the mouse passes over list items. This feedback can be a change of the mouse cursor or underline of the list items.

To turn hot tracking on, set HotTrack to true.

See Also

HotTrack (see page 3812)

27.1.36.2.30 TListView.HoverTime Property

Specifies how long the user must pause with the mouse before an item is selected.

Pascal

```
property HoverTime: Integer;
```

C++

```
__property int HoverTime;
```

Description

Use HoverTime to indicate how long (in milliseconds) the user must pause with the mouse before an item is selected. If HoverTime is -1 (the default value), a default time period must elapse.

HoverTime has no effect unless the HotTrack property is true.

See Also

HotTrack (see page 3812)

27.1.36.2.31 TListView.IconOptions Property

Determines how icons displayed in the list view are arranged.

Pascal

```
property IconOptions: TIconOptions;
```

C++

```
__property TIconOptions IconOptions;
```

Description

Set the properties of the TIconOptions object that is the value of IconOptions to specify how the icons in the list view should be arranged. The ViewStyle of the list view must be set to either vsIcon or vsSmallIcons for these options to affect the control.

See Also

TIconOptions (see page 4233)

ViewStyle (see page 3829)

27.1.36.2.32 TListView.Items Property

Contains the list of items displayed by the list view.

Pascal

```
property Items: TListItems;
```

C++

```
__property TListItems Items;
```

Description

Use Items to directly access the TListItem objects that represent the items in the list. Setting this property at design time brings up the ListView Items Editor. Use this dialog to add or delete items or subitems, and to edit their display properties. At runtime, use each item's Caption, ImageIndex, and StateIndex properties to change the appearance of the list items.

Warning: Items is read-only if the application is running in virtual mode.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
```

```
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", (TObject *)vsIcon);
  ComboBox1->Items->AddObject("vsList", (TObject *)vsList);
  ComboBox1->Items->AddObject("vsReport", (TObject *)vsReport);
  ComboBox1->Items->AddObject("vsSmallIcon", (TObject *)vsSmallIcon);
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
/*
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses defs file.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const char Names[6][2][10] =
   {{"Rubble","Barny"},
    {"Michael", "Johnson"},
    {"Bunny", "Bugs"},
    {"Silver", "HiHo"},
    {"Simpson", "Bart"},
    {"Squirrel", "Rocky"}};

  TListColumn  *NewColumn;
  TListItem  *ListItem;
  TListView   *ListView = new TListView(this);

  ListView->Parent = this;
  ListView->Align = alClient;
  ListView->ViewStyle = vsReport;
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "Last";
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "First";
  for (int i = 0; i < 6; i++)
  {
    ListItem = ListView->Items->Add();
    ListItem->Caption = Names[i][0];
    ListItem->SubItems->Add(Names[i][1]);
  }
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
```

```
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := 'Image' + IntToStr(I);
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
    NewColumn.Width := 200;
    // Add View styles and constants to the Combo Box
    ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
    ComboBox1.Items.AddObject('vsList', TObject(vsList));
    ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
    ComboBox1.Items.AddObject(
      'vsSmallIcon', TObject(vsSmallIcon));
    // Display first item in the Combo Box
    ComboBox1.ItemIndex := 0;
  end;
end;

procedure TComboForm.ComboBox1Click(Sender: TObject);
begin
  with ComboBox1 do
    ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

end;
{
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses clause of the unit file.
}
procedure TForm1.FormCreate(Sender: TObject);
const
```

```
    Names: array[0..5, 0..1] of string = (
      ('Rubble', 'Barney'),
      ('Michael', 'Johnson'),
      ('Bunny', 'Bugs'),
      ('Silver', 'HiHo'),
      ('Simpson', 'Bart'),
      ('Squirrel', 'Rocky')
      );

  var
    I: Integer;
    NewColumn: TListColumn;
    ListItem: TListItem;
    ListView: TListView;
  begin
    ListView := TListView.Create(Self);
    with ListView do
    begin
      Parent := Self;
      Align := alClient;
      ViewStyle := vsReport;

      NewColumn := Columns.Add;
      NewColumn.Caption := 'Last';
      NewColumn := Columns.Add;
      NewColumn.Caption := 'First';

      for I := Low(Names) to High(Names) do
      begin
        ListItem := Items.Add;
        ListItem.Caption := Names[I][0];
        ListItem.SubItems.Add(Names[I][1]);
      end;
    end;
  end;
```

See Also

TListItem (see page 4250)

Caption (see page 4258)

ImageIndex (see page 4261)

StateIndex (see page 4267)

27.1.36.2.33 TListView.LargeImages Property

Provides a list of icon images to display for each item in the list when ViewStyle is vsIcon.

Pascal

```
property LargeImages: TCustomImageList;
```

C++

```
__property TCustomImageList LargeImages;
```

Description

Set LargeImages to specify the bitmaps that should be displayed next to items in the list when ViewStyle is vsIcon. Each item in the Items list can be associated with a bitmap in this image list by setting its ImageIndex property.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
```

```
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", (TObject *)vsIcon);
  ComboBox1->Items->AddObject("vsList", (TObject *)vsList);
  ComboBox1->Items->AddObject("vsReport", (TObject *)vsReport);
  ComboBox1->Items->AddObject("vsSmallIcon", (TObject *)vsSmallIcon);
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
```

```
  as, for instance, ListView1.->ViewStyle := vsIcon.
  }

  procedure TComboForm.FormCreate(Sender: TObject);
  var
    I: Integer;
    ListItem: TListItem;
    NewColumn: TListColumn;
  begin
    Col2Array[0]:= 'Documentation for Image0.';
    Col2Array[1]:= 'Documentation for Image1.';
    Col2Array[2]:= 'Documentation for Image2.';
    // Create a ListView item for each image in the ImageList
    with ListView1 do
    begin
      SmallImages := ImageList1;
      LargeImages := ImageList1;
      for I := 0 to ImageList1.Count - 1 do
      begin
        ListItem := Items.Add;
        Listitem.Caption := 'Image' + IntToStr(I);
        ListItem.ImageIndex := I;
        Listitem.SubItems.Add(Col2Array[I]);
      end;
      // Create two columns to show during viewing as vsReport
      NewColumn := Columns.Add;
      NewColumn.Caption := 'Column 1';
      NewColumn.Width := 200;
      NewColumn := Columns.Add;
      NewColumn.Caption := 'Column 2';
      NewColumn.Width := 200;
      // Add View styles and constants to the Combo Box
      ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
      ComboBox1.Items.AddObject('vsList', TObject(vsList));
      ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
      ComboBox1.Items.AddObject(
        'vsSmallIcon', TObject(vsSmallIcon));
      // Display first item in the Combo Box
      ComboBox1.ItemIndex := 0;
    end;
  end;

  procedure TComboForm.ComboBox1Click(Sender: TObject);
  begin
    with ComboBox1 do
      ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

  end;
```

See Also

Items (see page 3814)

SmallImages (see page 3825)

StateImages (see page 3828)

TImageList (see page 6078)

ViewStyle (see page 3829)

27.1.36.2.34 TListView.MultiSelect Property

Determines whether the user can select more than one list item at a time.

Pascal

```
property MultiSelect: Boolean;
```

C++

```
__property Boolean MultiSelect;__property Boolean MultiSelect;
```

Description

Set MultiSelect to specify whether users can select multiple items in the list. If MultiSelect is true, the user can select multiple items. If MultiSelect if false, only one item can be selected at a time.

See Also

SelCount (see page 3822)

Selected (see page 3822)

27.1.36.2.35 TListView.OwnerData Property

Specifies whether the list view control is virtual.

Pascal

```
property OwnerData: Boolean;
```

C++

```
__property Boolean OwnerData;
```

Description

Set OwnerData to true to specify that the list view is virtual. A virtual list view can contain a huge number of items (specifically, up to a DWORD). However, you must manage the items of a virtual list view using the OnData, OnDataFind, OnDataHint, and OnDataStateChange event handlers. For example, you must explicitly provide a value to a list item's StateIndex property if it is to display a check box.

When creating a virtual list view, you must set the Count property for the Items to the number of items in the virtual list.

The only information contained in a virtual list view control is the selection and focus information.

To improve the performance of a virtual list view control, cache items for quick retrieval. The OnDataHint event can help optimize retrieval from the cache.

Note: Virtual list views need not be owner-drawn. The OnData, OnDataFind, OnDataHint, and OnDataStateChange events enable the list view's default drawing, if desired.

See Also

OnData (see page 3846)

OnDataFind (see page 3847)

OnDataHint (see page 3847)

OnDataStateChange (see page 3848)

OwnerDraw (see page 3821)

Count (see page 4281)

27.1.36.2.36 TListView.OwnerDraw Property

Specifies whether the list view receives an OnDrawItem event.

Pascal

```
property OwnerDraw: Boolean;
```

C++

```
__property Boolean OwnerDraw;
```

Description

Set OwnerDraw to true to allow the list view to receive the OnDrawItem event instead of the default rendering of list items.

Note: This property is independent of the custom draw events such as OnCustomDraw, OnCustomDrawItem, OnAdvancedCustomDraw, and so on. The custom draw events always occur.

See Also

OnDrawItem (see page 3848)

Canvas (see page 3804)

IsCustomDrawn (see page 3794)

27.1.36.2.37 TListView.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

27.1.36.2.38 TListView.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

27.1.36.2.39 TListView.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

27.1.36.2.40 TListView.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.36.2.41 TListView.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
```

```
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
```

```
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
```

```
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
```

```
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.36.2.42 TListView.ReadOnly Property

Determines whether the user can change the contents of the control.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to determine whether the list view can be used to enter data. If ReadOnly is true, the list view displays values that the user can't edit. If ReadOnly is false, the user can modify the contents. If ReadOnly is false, the OnEditing and OnEdited events occur when the user begins and finishes editing the list item, respectively.

See Also

OnEdited (see page 3849)

OnEditing (see page 3850)

27.1.36.2.43 TListView.RowSelect Property

Determines whether an entire row in the list view can be selected.

Pascal

```
property RowSelect: Boolean;
```

C++

```
__property Boolean RowSelect;
```

Description

Set RowSelect to true to allow entire rows to be selected when the ViewStyle property is vsReport. To allow multiple rows to be selected, use the MultiSelect property.

See Also

MultiSelect (see page 3820)

Selected (see page 4267)

ViewStyle (see page 3829)

27.1.36.2.44 TListView.ShowColumnHeaders Property

Determines whether the column headers are displayed.

Pascal

```
property ShowColumnHeaders: Boolean;
```

C++

```
__property Boolean ShowColumnHeaders;
```

Description

Set ShowColumnHeaders to specify whether the columns in the list view should display column headers. A column header is a panel with a caption that identifies the category of items in the column of the list view. Headers are useful for labeling columns of subitems that provide additional information in a list view. Use the Columns property to create or add headers.

Note: To use columns in a list view, the ViewStyle property must be set to vsReport.

See Also

ColumnClick (see page 3808)

Columns (see page 3808)

ViewStyle (see page 3829)

FullDrag (see page 3811)

27.1.36.2.45 TListView.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
```

```
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.36.2.46 TListView.ShowWorkAreas Property

Determines whether the visual properties of work areas are displayed.

Pascal

```
property ShowWorkAreas: Boolean;
```

C++

```
__property Boolean ShowWorkAreas;
```

Description

Set ShowWorkAreas to specify whether the list view should paint each work area in its own color and labeled by its display name. If the list view's WorkAreas property includes any work areas, the icons in the list view are always grouped visually by these work areas. ShowWorkAreas determines whether the properties of individual work area components are used to make each area stand out visually.

When ShowWorkAreas is true, each work area is painted in the color specified by its Color property and labeled by the string in its DisplayName property. When ShowWorkAreas is false, icons appear grouped by work areas, but there is no other visual indication.

Note: To use work areas in a list view, the ViewStyle property must be set to vsIcon or vsSmallIcon.

Warning: Work areas are not supported if the OwnerData property is true.

See Also

Color (see page 5428)

DisplayName (see page 2761)

ViewStyle (see page 3829)

WorkAreas (see page 3833)

OwnerData (see page 3820)

27.1.36.2.47 TListView.SmallImages Property

Provides a list of icon images to display for each item in the list when ViewStyle is not vsIcon.

Pascal

```
property SmallImages: TCustomImageList;
```

C++

```
__property TCustomImageList SmallImages;
```

Description

Set SmallImages to specify the icons that should be displayed next to items in the list when ViewStyle is vsSmallIcon, vsList, or vsReport. Each item in the Items list can be associated with an icon in this image list by setting its ImageIndex property.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
```

```
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", (TObject *)vsIcon);
  ComboBox1->Items->AddObject("vsList", (TObject *)vsList);
  ComboBox1->Items->AddObject("vsReport", (TObject *)vsReport);
  ComboBox1->Items->AddObject("vsSmallIcon", (TObject *)vsSmallIcon);
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := 'Image' + IntToStr(I);
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
    NewColumn.Width := 200;
    // Add View styles and constants to the Combo Box
    ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
```

```
      ComboBox1.Items.AddObject('vsList', TObject(vsList));
      ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
      ComboBox1.Items.AddObject(
        'vsSmallIcon', TObject(vsSmallIcon));
      // Display first item in the Combo Box
      ComboBox1.ItemIndex := 0;
    end;
  end;

  procedure TComboForm.ComboBox1Click(Sender: TObject);
  begin
    with ComboBox1 do
      ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

  end;
```

See Also

Items (see page 3814)

LargeImages (see page 3817)

StateImages (see page 3828)

TImageList (see page 6078)

ViewStyle (see page 3829)

27.1.36.2.48 TListView.SortType Property

Determines if and how the items in the list are automatically sorted.

Pascal

```
property SortType: TSortType;
```

C++

```
__property TSortType SortType;
```

Description

Set SortType to specify the order of items in the list. If SortType is not stNone, the list items in the Items property are automatically sorted. Once a list view is sorted, the original order is lost. That is, setting the SortType back to stNone will not restore the original order of items.

Note: If SortType is stData or stBoth, provide an OnCompare event handler to compare data values.

Note: If the list view uses work areas, sorting items only compares each item to the other items in its work area.

See Also

Items (see page 3814)

OnCompare (see page 3841)

Caption (see page 4258)

Data (see page 4259)

WorkAreas (see page 3833)

27.1.36.2.49 TListView.StateImages Property

Specifies additional icon images to display next to each item in the list view.

Pascal

```
property StateImages: TCustomImageList;
```

C++

```
__property TCustomImageList StateImages;
```

Description

Set StateImages to specify the icons that should be displayed next to items in the list. Each item in the Items list can be associated with an icon in this image list by setting its StateIndex property.

See Also

Items (see page 3814)

LargeImages (see page 3817)

SmallImages (see page 3825)

TImageList (see page 6078)

ViewStyle (see page 3829)

StateIndex (see page 4267)

27.1.36.2.50 TListView.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

27.1.36.2.51 TListView.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

27.1.36.2.52 TListView.ViewStyle Property

Determines how the list items are displayed.

Pascal

```
property ViewStyle: TViewStyle;
```

C++

```
__property TViewStyle ViewStyle;
```

Description

Set ViewStyle to control the visual display of items in a list view. The items can be displayed as a set of movable icons, or as columns of text.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", (TObject *)vsIcon);
  ComboBox1->Items->AddObject("vsList", (TObject *)vsList);
  ComboBox1->Items->AddObject("vsReport", (TObject *)vsReport);
  ComboBox1->Items->AddObject("vsSmallIcon", (TObject *)vsSmallIcon);
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
/*
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses defs file.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const char Names[6][2][10] =
   {{"Rubble","Barny"},
    {"Michael", "Johnson"},
    {"Bunny", "Bugs"},
    {"Silver", "HiHo"},
    {"Simpson", "Bart"},
    {"Squirrel", "Rocky"}};

  TListColumn  *NewColumn;
  TListItem  *ListItem;
  TListView   *ListView = new TListView(this);

  ListView->Parent = this;
  ListView->Align = alClient;
  ListView->ViewStyle = vsReport;
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "Last";
  NewColumn = ListView->Columns->Add();
  NewColumn->Caption = "First";
  for (int i = 0; i < 6; i++)
  {
    ListItem = ListView->Items->Add();
    ListItem->Caption = Names[i][0];
```

```
    ListItem->SubItems->Add(Names[i][1]);
  }
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := 'Image' + IntToStr(I);
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
    NewColumn.Width := 200;
    // Add View styles and constants to the Combo Box
    ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
    ComboBox1.Items.AddObject('vsList', TObject(vsList));
    ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
    ComboBox1.Items.AddObject(
      'vsSmallIcon', TObject(vsSmallIcon));
    // Display first item in the Combo Box
    ComboBox1.ItemIndex := 0;
  end;
end;

procedure TComboForm.ComboBox1Click(Sender: TObject);
begin
  with ComboBox1 do
    ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);
```

```
end;
{
This example requires only a blank form.  All other objects:
TListView, TListColumns, TListItems, are created dynamically.
You must add comctrls to the uses clause of the unit file.
}
procedure TForm1.FormCreate(Sender: TObject);
const
  Names: array[0..5, 0..1] of string = (
    ('Rubble', 'Barney'),
    ('Michael', 'Johnson'),
    ('Bunny', 'Bugs'),
    ('Silver', 'HiHo'),
    ('Simpson', 'Bart'),
    ('Squirrel', 'Rocky')
    );

var
  I: Integer;
  NewColumn: TListColumn;
  ListItem: TListItem;
  ListView: TListView;
begin
  ListView := TListView.Create(Self);
  with ListView do
  begin
    Parent := Self;
    Align := alClient;
    ViewStyle := vsReport;

    NewColumn := Columns.Add;
    NewColumn.Caption := 'Last';
    NewColumn := Columns.Add;
    NewColumn.Caption := 'First';

    for I := Low(Names) to High(Names) do
    begin
      ListItem := Items.Add;
      ListItem.Caption := Names[I][0];
      ListItem.SubItems.Add(Names[I][1]);
    end;
  end;
end;
```

See Also

IconOptions (see page 3813)

LargeImages (see page 3817)

ShowColumnHeaders (see page 3824)

SmallImages (see page 3825)

StateImages (see page 3828)

27.1.36.2.53 TListView.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
```

```
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.36.3 TListView Events

The events of the TListView class are listed here.

Published Events

	Name	Description
	OnAdvancedCustomDraw (see page 4346)	Occurs at discrete points during the rendering of the list view.
	OnAdvancedCustomDrawItem (see page 4347)	Occurs at discrete stages during the painting of an item in an owner-draw list view.
	OnAdvancedCustomDrawSubItem (see page 4347)	Occurs at discrete stages during the painting of a subitem in an owner-drawn list view.
	OnChange (see page 4348)	Occurs immediately after an item in the list changes.
	OnChanging (see page 4349)	Occurs when an item in the list is about to change.
	OnClick (see page 4349)	Occurs when the user clicks the control.
	OnColumnClick (see page 4351)	Occurs when the user clicks on a column header in a list view.
	OnColumnDragged (see page 4352)	Occurs when a column is dragged to a new position.
	OnColumnRightClick (see page 4352)	Occurs when the user right clicks on a column header in a list view.
	OnCompare (see page 4353)	Occurs to when two items need to be compared during a sort of the list.
	OnContextPopup (see page 4353)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCustomDraw (see page 4354)	Occurs when an owner-draw list view must be rendered.
	OnCustomDrawItem (see page 4355)	Occurs when an item in an owner-draw list view must be rendered.
	OnCustomDrawSubItem (see page 4355)	Occurs when a subitem must be rendered in an owner-draw list view.
	OnData (see page 4356)	Occurs immediately before an item is displayed in the list view control.
	OnDataFind (see page 4356)	Occurs when the list view control receives a search request, such as when the FindData (see page 3789) and FindCaption (see page 3788) methods are called.
	OnDataHint (see page 4357)	Occurs when the contents of the list view change, such as when the user scrolls the list view.
	OnDataStateChange (see page 4357)	Occurs when a range of items change states.
	OnDblClick (see page 4358)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeletion (see page 4358)	Occurs when an item in the list view is about to be deleted.
	OnDragDrop (see page 4359)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4361)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 4363)	Occurs when an item should be drawn in the list view.
	OnEdited (see page 4364)	Occurs immediately after the user edits an item's Caption property.

	OnEditing (see page 4364)	Occurs when the user starts to edit a list item's Caption property.
	OnEndDock (see page 4365)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4365)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4366)	Occurs when a control receives the input focus.
	OnExit (see page 4369)	Occurs when the input focus shifts away from one control to another.
	OnGetImageIndex (see page 4370)	Occurs when an item is about to be displayed in the list view control.
	OnGetSubItemImage (see page 4371)	Occurs when a subitem is about to be displayed in the column of a list view control.
	OnInfoTip (see page 4371)	Occurs when the user pauses the mouse over an item in the list view.
	OnInsert (see page 4372)	Occurs immediately after a new item is inserted into the list view.
	OnKeyDown (see page 4372)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 4375)	Occurs when a key is pressed.
	OnKeyUp (see page 4376)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 4378)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4379)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4381)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4381)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4381)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4384)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 4386)	Occurs immediately after the control is resized.
	OnSelectItem (see page 4387)	Occurs when an item is selected in the list view control.
	OnStartDock (see page 4387)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4387)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

27.1.36.3.1 TListView.OnAdvancedCustomDraw Event

Occurs at discrete points during the rendering of the list view.

Pascal

```
property OnAdvancedCustomDraw: TLVAdvancedCustomDrawEvent;
```

C++

```
__property TLVAdvancedCustomDrawEvent OnAdvancedCustomDraw;
```

Description

Write an OnAdvancedCustomDraw event handler to customize the painting of the list view at various stages before its items are drawn. Unlike the OnDrawItem (see page 4363) event, the list view receives this event even if the OwnerDraw (see page 4325) property is false.

Sender is the list view that is about to be painted.

ARect indicates the dimensions of the list view on its Canvas (see page 3804).

Stage indicates the current stage in the drawing process. Note that the cdPreErase and cdPostErase stages do not receive event notification. The background must be drawn when the item is rendered.

DefaultDraw is used only when Stage is cdPrePaint. It indicates whether the list view should continue with the default painting after the event handler exits. Set DefaultDraw to false to prevent the drawing of the list view after event handler exits. If DefaultDraw remains set to true, the list view continues with the default painting process.

Note: OnAdvancedCustomDraw occurs at various stages during the paint process. If you are only writing code to execute immediately prior to rendering the list view, it is more efficient to use the OnCustomDraw (see page 4354) event.

See Also

OnAdvancedCustomDrawItem (see page 4347)

OnAdvancedCustomDrawSubItem (see page 4347)

OwnerDraw (see page 4325)

OnCustomDraw (see page 4354)

27.1.36.3.2 TListView.OnAdvancedCustomDrawItem Event

Occurs at discrete stages during the painting of an item in an owner-draw list view.

Pascal

```
property OnAdvancedCustomDrawItem: TLVAdvancedCustomDrawItemEvent;
```

C++

```
__property TLVAdvancedCustomDrawItemEvent OnAdvancedCustomDrawItem;
```

Description

Write an OnAdvancedCustomDrawItem event handler to customize the drawing of individual items on the list view's canvas (using the Canvas (see page 3804) property). OnAdvancedCustomDrawItem occurs at various stages in the rendering of a list item.

Unlike the OnDrawItem (see page 4363) event, the list view receives this event even if the OwnerDraw (see page 4325) property is false. The OnAdvancedCustomDrawItem event handler provides slightly different state information than the OnDrawItem (see page 4363) event and occurs after the default rendering as well as before.

Sender is the list view whose item is drawn.

Item is the item that is currently being painted.

State indicates the state of the item, so that the event handler can adjust the image to reflect whether the item is selected, disabled, hot, and so on.

Stage indicates the current stage in drawing the list item. Note that the cdPreErase and cdPostErase stages do not receive event notification. The background must be drawn when the item is rendered.

DefaultDraw is used only when Stage is cdPrePaint. Set DefaultDraw to false if you don't want the control to paint the item's text after the event handler exits. If DefaultDraw remains true, the list view adds the item's text to the image on the canvas.

Note: OnAdvancedCustomDrawItem occurs at various stages during the paint process. If you are only writing code to execute immediately prior to rendering the list item, it is more efficient to use the OnCustomDrawItem (see page 4355) event.

See Also

OnAdvancedCustomDraw (see page 4346)

OnAdvancedCustomDrawSubItem (see page 4347)

Canvas (see page 3804)

OwnerDraw (see page 4325)

OnCustomDrawItem (see page 4355)

27.1.36.3.3 TListView.OnAdvancedCustomDrawSubItem Event

Occurs at discrete stages during the painting of a subitem in an owner-drawn list view.

Pascal

```
property OnAdvancedCustomDrawSubItem: TLVAdvancedCustomDrawSubItemEvent;
```

C++

```
__property TLVAdvancedCustomDrawSubItemEvent OnAdvancedCustomDrawSubItem;
```

Description

Write an OnCustomDrawSubItem (see page 4355) event handler to customize the drawing of subitems that appear in additional columns to the right of each item when ViewStyle (see page 4340) is vsReport. Use the list view's Canvas (see page 3804) property as a drawing surface. OnAdvancedCustomDrawItem (see page 4347) occurs at various stages in the rendering of a subitems. Unlike the OnDrawItem (see page 4363) event, the list view receives this event even if the OwnerDraw (see page 4325) property is false.

The Sender Parameter specifies the list view that owns the subitems.

The Item parameter is the current Item being drawn.

The SubItem parameter is the index of the subitem of that list item in its SubItems property.

The State property indicates various attributes that can affect the way the subitem is drawn.

The Stage property indicates the current stage in the painting process. Note that the cdPreErase and cdPostErase stages do not receive event notification. The background must be drawn when the item is rendered.

DefaultDraw is only used when Stage is cdPrepaint. Set DefaultDraw to false to prevent the list view from adding the subitem's text after the event handler exits.

Note: OnAdvancedCustomDrawSubItem occurs at various stages during the paint process. If you are only writing code to execute immediately prior to rendering the subitem, it is more efficient to use the OnCustomDrawSubItem (see page 4355) event.

See Also

OnCustomDrawItem (see page 4355)

SubItems (see page 4268)

ViewStyle (see page 4340)

Canvas (see page 3804)

OwnerDraw (see page 4325)

OnAdvancedCustomDrawSubItem

27.1.36.3.4 TListView.OnChange Event

Occurs immediately after an item in the list changes.

Pascal

```
property OnChange: TLVChangeEvent;
```

C++

```
__property TLVChangeEvent OnChange;
```

Description

Write an OnChange event handler to respond to changes in the list items. OnChange allows a response once the list has been successfully changed.

The Item parameter is the list item that just changed. The Change (see page 3778) parameter indicates the type of change

that just occurred. Change (see page 3778) is ctText if the Caption property of the item changed. Change (see page 3778) is ctImage if the ImageIndex property of the item changed or the appropriate image list changed in the list view. Change (see page 3778) is ctState if the Cut, Focused, or Selected (see page 3822) property of the item changed.

See Also

LargeImages (see page 4322)

OnChanging (see page 4349)

SmallImages (see page 4336)

Cut (see page 4259)

Focused (see page 4260)

Caption (see page 4258)

Selected (see page 4267)

ImageIndex (see page 4261)

27.1.36.3.5 TListView.OnChanging Event

Occurs when an item in the list is about to change.

Pascal

```
property OnChanging: TLVChangingEvent;
```

C++

```
__property TLVChangingEvent OnChanging;
```

Description

Write an OnChanging event handler to conditionally block changes to an item in the list. Set the AllowChange parameter to false to prevent the change from taking place. The Item parameter is the list item that is about to change. The Change (see page 3778) parameter indicates the type of change that is about to happen.

See Also

OnChange (see page 4348)

TListItem (see page 4250)

27.1.36.3.6 TListView.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This

event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;
```

```
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

27.1.36.3.7 TListView.OnColumnClick Event

Occurs when the user clicks on a column header in a list view.

Pascal

```
property OnColumnClick: TLVColumnClickEvent;
```

C++

```
__property TLVColumnClickEvent OnColumnClick;
```

Description

Write an OnColumnClick event handler to respond to clicks made in the column header when ViewStyle (see page 4340) is vsReport. The Column (see page 3806) parameter is the TListColumn (see page 4236) object from the Columns (see page 4309) property that describes the column that was clicked.

OnColumnClick only occurs when ViewStyle (see page 4340) is vsReport and ShowColumnHeaders (see page 4333) is true.

See Also

ColumnClick (see page 4309)

Columns (see page 4309)

ShowColumnHeaders (see page 4333)

TListColumn (see page 4236)

ViewStyle (see page 4340)

OnColumnRightClick (see page 4352)

27.1.36.3.8 TListView.OnColumnDragged Event

Occurs when a column is dragged to a new position.

Pascal

```
property OnColumnDragged: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnColumnDragged;
```

Description

Write an OnColumnDragged event handler to respond when the user drags a column to a new position. OnColumnDragged occurs after all columns are in their new positions but before the list view repaints to reflect the change.

OnColumnClick (see page 4351) only occurs when ViewStyle (see page 4340) is vsReport and ShowColumnHeaders (see page 4333) is true.

Tip: To uniquely identify columns so that you can recognize them after they are repositioned, use the list column's Tag property.

See Also

ViewStyle (see page 4340)

ShowColumnHeaders (see page 4333)

Tag (see page 4244)

Columns (see page 4309)

OnColumnClick (see page 4351)

27.1.36.3.9 TListView.OnColumnRightClick Event

Occurs when the user right clicks on a column header in a list view.

Pascal

```
property OnColumnRightClick: TLVColumnRClickEvent;
```

C++

```
__property TLVColumnRClickEvent OnColumnRightClick;
```

Description

Write an OnColumnRightClick event handler to respond to right clicks made in the column header when ViewStyle (see page 4340) is vsReport.

The Sender parameter is the list view that the user clicked.

The Column (see page 3806) parameter is the TListColumn (see page 4236) object from the Columns (see page 4309) property for the column that was clicked.

The point parameter indicates where the user clicked the mouse, in client coordinates.

OnColumnRightClick only occurs when ViewStyle (see page 4340) is vsReport, and ShowColumnHeaders (see page 4333) is true, and ColumnRightClick is true.

See Also

Columns (see page 4309)

ShowColumnHeaders (see page 4333)

TListColumn (see page 4236)

ViewStyle (see page 4340)

OnContextPopup (see page 5854)

27.1.36.3.10 TListView.OnCompare Event

Occurs to when two items need to be compared during a sort of the list.

Pascal

```
property OnCompare: TLVCompareEvent;
```

C++

```
__property TLVCompareEvent OnCompare;
```

Description

Write an OnCompare event handler to implement a sort order for the list. An OnCompare event handler is called when the SortType (see page 4338) property is stData or stBoth, when the AlphaSort (see page 3774) method is called, or when the CustomSort (see page 3784) method is called without a SortProc parameter.

The OnCompare event handler compares the list items passed as the Item1 and Item2 parameters. If Item1 is the same as Item2 in the sort order, set the Compare parameter to 0. If Item1 is less than Item2, set the Compare parameter to a value less than 0. If Item1 is greater than Item2, set the Compare parameter to a value greater than 0.

The Data parameter is 0 when the event handler is called to maintain the sort order of a list view with a SortType (see page 4338) of stData or stBoth. Similarly, when OnCompare occurs in response to the AlphaSort (see page 3774) method, the Data parameter is 0. When OnCompare occurs in response to the CustomSort (see page 3784) method, the Data parameter is the value of the LParam parameter of CustomSort (see page 3784).

See Also

AlphaSort (see page 3774)

CustomSort (see page 3784)

SortType (see page 4338)

27.1.36.3.11 TListView.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The

OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.36.3.12 TListView.OnCustomDraw Event

Occurs when an owner-draw list view must be rendered.

Pascal

```
property OnCustomDraw: TLVCustomDrawEvent;
```

C++

```
__property TLVCustomDrawEvent OnCustomDraw;
```

Description

Write an OnCustomDraw event handler to draw to the list view's Canvas (see page 3804) before its items are painted. Unlike the OnDrawItem (see page 4363) event, the list view receives this event even if the OwnerDraw (see page 4325) property is false.

Sender is the list view that is about to be painted.

ARect indicates the dimensions of the list view on its Canvas (see page 3804).

DefaultDraw indicates whether the list view should continue with the default painting after the event handler exits. Set DefaultDraw to false to prevent the drawing of the list view after event handler exits. If DefaultDraw remains set to true, the list view continues with the default painting process.

Note: OnCustomDraw occurs immediately prior to the rendering of the list view control. To augment the default drawing process at other stages (such as when the background is erased or after the list view is drawn), use the OnAdvancedCustomDraw (see page 4346) event instead.

See Also

OnCustomDrawItem (see page 4355)

OnCustomDrawSubItem (see page 4355)

Canvas (see page 3804)

OwnerDraw (see page 4325)

OnAdvancedCustomDraw (see page 4346)

27.1.36.3.13 TListView.OnCustomDrawItem Event

Occurs when an item in an owner-draw list view must be rendered.

Pascal

```
property OnCustomDrawItem: TLVCustomDrawItemEvent;
```

C++

```
__property TLVCustomDrawItemEvent OnCustomDrawItem;
```

Description

Write code in an OnCustomDrawItem handler to draw individual items on the list view's canvas (using the Canvas (see page 3804) property) before an item is painted.

Unlike the OnDrawItem (see page 4363) event, the list view receives this event even if the OwnerDraw (see page 4325) property is false. The OnCustomDrawItem event handler provides slightly different state information than the OnDrawItem (see page 4363) event.

Sender is the list view whose item is to be drawn.

Item is the item that should be rendered.

State indicates the state of the item, so that the event handler can adjust the image to reflect whether the item is selected, disabled, hot, and so on.

Set DefaultDraw to false if you don't want the control to paint the item's text after the event handler exits. If DefaultDraw remains true, the list view adds the item's text to the image on the canvas.

Note: OnCustomDrawItem occurs immediately prior to the rendering of each list item. To augment the default drawing process at other stages (such as after the list item is drawn), use the OnAdvancedCustomDrawItem (see page 4347) event instead.

See Also

OnCustomDraw (see page 4354)

OnCustomDrawSubItem (see page 4355)

Canvas (see page 3804)

OwnerDraw (see page 4325)

OnDrawItem (see page 4363)

OnAdvancedCustomDrawItem (see page 4347)

27.1.36.3.14 TListView.OnCustomDrawSubItem Event

Occurs when a subitem must be rendered in an owner-draw list view.

Pascal

```
property OnCustomDrawSubItem: TLVCustomDrawSubItemEvent;
```

C++

```
__property TLVCustomDrawSubItemEvent OnCustomDrawSubItem;
```

Description

Write code in an OnCustomDrawSubItem handler to draw to the subitems that appear in additional columns to the right of each item when ViewStyle (see page 4340) is vsReport. Use the list view's Canvas (see page 3804) property as a drawing surface. Unlike the OnDrawItem (see page 4363) event, the list view receives this event even if the OwnerDraw (see page 4325) property is false.

The Sender Parameter specifies the list view that owns the subitems. The Item parameter is the current Item being drawn. The SubItem parameter is the index of the subitem of that list item in its SubItems property. The State property indicates various attributes that can affect the way the subitem is drawn. Set DefaultDraw to False to prevent the list view from adding the subitem's text after the event handler exits.

Note: OnCustomDrawSubItem occurs immediately prior to the rendering of each subitem. To augment the default drawing process at other stages (such as after the subitem is drawn), use the OnAdvancedCustomDrawSubItem (see page 4347) event instead.

See Also

OnCustomDrawItem (see page 4355)

SubItems (see page 4268)

ViewStyle (see page 4340)

Canvas (see page 3804)

OwnerDraw (see page 4325)

OnAdvancedCustomDrawSubItem (see page 4347)

27.1.36.3.15 TListView.OnData Event

Occurs immediately before an item is displayed in the list view control.

Pascal

```
property OnData: TLVOwnerDataEvent;
```

C++

```
__property TLVOwnerDataEvent OnData;
```

Description

Use OnData to customize an item before it is displayed in the list view control. Set the properties of the list item in the event handler so that they appear correctly when the item is drawn.

OnData occurs only if OwnerData (see page 4325) is true.

See Also

OwnerData (see page 4325)

OnDataFind (see page 4356)

OnDataHint (see page 4357)

OnDataStateChange (see page 4357)

27.1.36.3.16 TListView.OnDataFind Event

Occurs when the list view control receives a search request, such as when the FindData (see page 3789) and FindCaption (see page 3788) methods are called.

Pascal

```
property OnDataFind: TLVOwnerDataFindEvent;
```

C++

```
__property TLVOwnerDataFindEvent OnDataFind;
```

Description

Use OnDataFind to locate an item and return custom data associated with it.

OnDataFind occurs only if OwnerData (see page 4325) is true.

See Also

OwnerData (see page 4325)

OnData (see page 4356)

OnDataHint (see page 4357)

OnDataStateChange (see page 4357)

27.1.36.3.17 TListView.OnDataHint Event

Occurs when the contents of the list view change, such as when the user scrolls the list view.

Pascal

```
property OnDataHint: TLVOwnerDataHintEvent;
```

C++

```
__property TLVOwnerDataHintEvent OnDataHint;
```

Description

Use OnDataHint to update a range of items before the OnData (see page 4356) event occurs. This method of updating items is more efficient than individually updating items in the OnData (see page 4356) event handler. The range of items specified in the OnDataHint does not always match the range of items for which the OnData (see page 4356) event handler is called; therefore, you must be prepared to update individual items in the OnData (see page 4356) event handler as well.

OnDataHint occurs only if OwnerData (see page 4325) is true.

See Also

OwnerData (see page 4325)

OnDataFind (see page 4356)

OnData (see page 4356)

OnDataStateChange (see page 4357)

27.1.36.3.18 TListView.OnDataStateChange Event

Occurs when a range of items change states.

Pascal

```
property OnDataStateChange: TLVOwnerDataStateChangeEvent;
```

C++

```
__property TLVOwnerDataStateChangeEvent OnDataStateChange;
```

Description

Use OnDataStateChange to respond when an item's state changes.

OnDataStateChange occurs only if OwnerData (see page 4325) is true.

See Also

OwnerData (see page 4325)

OnDataFind (see page 4356)

OnDataHint (see page 4357)

OnData (see page 4356)

27.1.36.3.19 TListView.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

27.1.36.3.20 TListView.OnDeletion Event

Occurs when an item in the list view is about to be deleted.

Pascal

```
property OnDeletion: TLVDeletedEvent;
```

C++

```
__property TLVDeletedEvent OnDeletion;
```

Description

Write an OnDeletion event handler to respond when an item in the list is about to be deleted. The Item parameter is the TListItem (see page 4250) object from the Items (see page 4319) property that is about to be deleted.

See Also

Items (see page 4319)

OnInsert (see page 4372)

TListItem (see page 4250)

27.1.36.3.21 TListView.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

27.1.36.3.22 TListView.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}
```

```
/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
```

```
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.36.3.23 TListView.OnDrawItem Event

Occurs when an item should be drawn in the list view.

Pascal

```
property OnDrawItem: TLVDrawItemEvent;
```

C++

```
__property TLVDrawItemEvent OnDrawItem;
```

Description

Write code in an OnDrawItem handler to manually draw items in the list view. This event occurs only if OwnerDraw (see page 4325) is true.

Note: List views receive several other custom draw events, including OnCustomDraw (see page 4354), OnCustomDrawItem (see page 4355), OnCustomDrawSubItem (see page 4355), OnAdvancedCustomDraw (see page 4346),

OnAdvancedCustomDrawItem (see page 4347), and OnAdvancedCustomDrawSubItem (see page 4347). These other events, unlike OnDrawItem, occur regardless of the value of the OwnerDraw (see page 4325) property. They provide slightly different information about the state of the item to be drawn, and the OnAdvancedXXX events occur at more points during the paint process.

See Also

OnCustomDraw (see page 4354)

OnCustomDrawSubItem (see page 4355)

Canvas (see page 3804)

OwnerDraw (see page 4325)

OnCustomDrawItem (see page 4355)

OnAdvancedCustomDraw (see page 4346)

OnAdvancedCustomDrawItem (see page 4347)

OnAdvancedCustomDrawSubItem (see page 4347)

27.1.36.3.24 TListView.OnEdited Event

Occurs immediately after the user edits an item's Caption property.

Pascal

```
property OnEdited: TLVEditedEvent;
```

C++

```
__property TLVEditedEvent OnEdited;
```

Description

Write an OnEdited event to take specific action when a user has finished editing the text of a list item. An OnEdited event handler can respond to the value the user typed, or change the value that is assigned to the list item's Caption property. For example, if an OnEdited event handler can check the validity of the string typed by the user, and only allow the Caption property to be changed if the new value is valid.

The Item parameter is the list item that was just edited. The S parameter is the new value the user typed for the item's Caption property. Change (see page 3778) the value of S to override the value the user typed. When OnEdited occurs, the item's Caption property has not yet been changed, so S can still be compared to the original value of Caption.

This event can occur only if ReadOnly (see page 4332) is set to false.

See Also

OnEditing (see page 4364)

ReadOnly (see page 4332)

Caption (see page 4258)

27.1.36.3.25 TListView.OnEditing Event

Occurs when the user starts to edit a list item's Caption property.

Pascal

```
property OnEditing: TLVEditingEvent;
```

C++

```
__property TLVEditingEvent OnEditing;
```

Description

Write an OnEditing event handler to take specific action just before a user edits a list item's Caption property. To prevent the user from editing the item specified by the ListItem parameter, set the AllowEdit parameter to false.

This event can occur only if the ReadOnly (see page 4332) property is set to false.

See Also

OnEdited (see page 4364)

ReadOnly (see page 4332)

Caption (see page 4258)

27.1.36.3.26 TListView.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.36.3.27 TListView.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
```

```
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

27.1.36.3.28 TListView.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

27.1.36.3.29 TListView.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
```

```
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

27.1.36.3.30 TListView.OnGetImageIndex Event

Occurs when an item is about to be displayed in the list view control.

Pascal

```
property OnGetImageIndex: TLVNotifyEvent;
```

C++

```
__property TLVNotifyEvent OnGetImageIndex;
```

Description

Use OnGetImageIndex to dynamically assign an ImageIndex to the item before it is displayed in the list view control.

See Also

OnGetSubItemImage (see page 4371)

ImageIndex (see page 4261)

27.1.36.3.31 TListView.OnGetSubItemImage Event

Occurs when a subitem is about to be displayed in the column of a list view control.

Pascal

```
property OnGetSubItemImage: TLVSubItemImageEvent;
```

C++

```
__property TLVSubItemImageEvent OnGetSubItemImage;
```

Description

Use OnGetSubItemImage to dynamically assign an image to a subitem before it is displayed in the column of a list view control. This event overrides the value specified by the list item's SubItemImages property.

The Sender parameter is the list view that is being painted.

The Item parameter is the item whose subitem is being painted.

The SubItem parameter identifies the subitem from the list item's SubItems property.

The ImageIndex parameter is an index into the list view's SmallImages (see page 4336) property. On entry to the event handler, ImageIndex is initialized to the value specified by the list item's SubItemImages property for the subitem. Use this event handler to change that value or to suppress the image by setting ImageIndex to –1.

Note: OnGetSubItemImage only occurs if the list view's ViewStyle (see page 4340) property is vsReport and the current list item's SubItems property contains subitems.

See Also

OnGetImageIndex (see page 4370)

SubItemImages (see page 4268)

SmallImages (see page 4336)

27.1.36.3.32 TListView.OnInfoTip Event

Occurs when the user pauses the mouse over an item in the list view.

Pascal

```
property OnInfoTip: TLVInfoTipEvent;
```

C++

```
__property TLVInfoTipEvent OnInfoTip;
```

Description

Use OnInfoTip to provide help hints that are specific to individual items in the list view.

By default, when Help hints are enabled (that is, when ShowHint (see page 4334) is true), the list view displays the hint specified by its Hint property. OnInfoTip allows the list view to override this value to specify a hint that is specific to the item under the mouse.

The Sender parameter is the list view under the mouse.

The Item parameter is the list item under the mouse.

The InfoTip parameter returns a string that appears in a help hint window. On entry to the event handler, InfoTip is the item's Caption. If the event handler assigns an empty string, no hint is displayed. Otherwise, a hint window appears that displays the value of InfoTip.

See Also

Hint (see page 5827)

Caption (see page 4258)

ShowHint (see page 5840)

27.1.36.3.33 TListView.OnInsert Event

Occurs immediately after a new item is inserted into the list view.

Pascal

```
property OnInsert: TLVDeletedEvent;
```

C++

```
__property TLVDeletedEvent OnInsert;
```

Description

Write an OnInsert event handler to respond when an item has just been added to the list. The Item parameter is the TListItem (see page 4250) object that was added to the Items (see page 4319) property.

See Also

Items (see page 4319)

OnDeletion (see page 4358)

TListItem (see page 4250)

27.1.36.3.34 TListView.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the

OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
```

```
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

27.1.36.3.35 TListView.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
```

```
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

27.1.36.3.36 TListView.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
```

```
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

27.1.36.3.37 TListView.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.36.3.38 TListView.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
```

```
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
```

```
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.36.3.39 TListView.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

27.1.36.3.40 TListView.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

27.1.36.3.41 TListView.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
```

```
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.36.3.42 TListView.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
```

```
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
```

```
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

27.1.36.3.43 TListView.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

27.1.36.3.44 TListView.OnSelectItem Event

Occurs when an item is selected in the list view control.

Pascal

```
property OnSelectItem: TLVSelectItemEvent;
```

C++

```
__property TLVSelectItemEvent OnSelectItem;
```

Description

Use OnSelectItem to perform actions when an item is selected.

See Also

Selected (see page 4267)

27.1.36.3.45 TListView.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

27.1.36.3.46 TListView.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

27.1.37 TListViewActionLink Class

Class Hierarchy

Pascal

```
TListViewActionLink = class(TListActionLink);
```

C++

```
class TListViewActionLink : public TListActionLink;
```

File

ComCtrls

Description

This is class ComCtrls.TListViewActionLink.

27.1.37.1 TListViewActionLink Members

The following tables list the members exposed by TListViewActionLink.

Protected Fields

	Name	Description
	FAction (see page 2722)	This is FAction, a member of class TBasicActionLink.
	FClient (see page 5894)	This is FClient, a member of class TControlActionLink.
	FClient (see page 6236)	This is FClient, a member of class TWinControlActionLink.

Protected Methods

	Name	Description
	AddItem (see page 4391)	This is the overview for the AddItem method overload.
	AddItem (see page 15747)	This is the overview for the AddItem method overload.
	AssignClient (see page 2723)	This is AssignClient, a member of class TBasicActionLink.
	AssignClient (see page 5895)	Assigns a client control that is linked to the associated action.
	AssignClient (see page 6236)	Specifies the client control.
	Change (see page 2723)	This is Change, a member of class TBasicActionLink.
	DoShowHint (see page 5896)	Indicates whether the hint is shown, and modified.
	IsActiveLinked (see page 15748)	This is IsActiveLinked, a member of class TListActionLink.
	IsCaptionLinked (see page 268)	Indicates whether the Caption properties for the action and client are linked.
	IsCaptionLinked (see page 5896)	Indicates whether the caption for the action and the client are linked.
	IsCheckedLinked (see page 269)	Indicates whether the Checked properties for the action and client are linked.
	IsDropdownMenuLinked (see page 5896)	This is IsDropdownMenuLinked, a member of class TControlActionLink.
	IsEnabledLinked (see page 269)	Indicates whether the Enabled properties for the action and client are linked.
	IsEnabledLinked (see page 5897)	Indicates whether the enabled state for the action and the client are linked.
	IsEnableDropdownLinked (see page 5897)	This is IsEnableDropdownLinked, a member of class TControlActionLink.
	IsGroupIndexLinked (see page 270)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsHelpContextLinked (see page 270)	Indicates whether the HelpContext properties for the action and client controls are linked.
	IsHelpContextLinked (see page 6236)	Indicates whether the Help context for the action and the client are linked.
	IsHelpLinked (see page 271)	Indicates whether the help properties for the action and client are linked.
	IsHelpLinked (see page 5897)	Indicates whether the help properties for the action and the client are linked.
	IsHintLinked (see page 271)	Indicates whether the Hint properties for the action and client are linked.
	IsHintLinked (see page 5898)	Indicates whether the Hint property for the action and the client are linked.
	IsImageIndexLinked (see page 272)	Indicates whether the image index for the action and the client are linked.
	IsImagesLinked (see page 15748)	This is IsImagesLinked, a member of class TListActionLink.
	IsOnExecuteLinked (see page 2723)	This is IsOnExecuteLinked, a member of class TBasicActionLink.
	IsOnExecuteLinked (see page 5898)	Indicates whether the OnExecute event handler for the action and the client are linked.
	IsPopupMenuLinked (see page 5898)	This is IsPopupMenuLinked, a member of class TControlActionLink.
	IsShortCutLinked (see page 272)	Indicates whether the ShortCut properties for the action and client are linked.
	IsVisibleLinked (see page 273)	Indicates whether the Visible properties for the action and client are linked.
	IsVisibleLinked (see page 5899)	Indicates whether the visible state for the action and the client are linked.
	RefreshControl (see page 15748)	This is RefreshControl, a member of class TListActionLink.
	SetAction (see page 2724)	This is SetAction, a member of class TBasicActionLink.
	SetAction (see page 15749)	This is SetAction, a member of class TListActionLink.
	SetActive (see page 15749)	This is SetActive, a member of class TListActionLink.
	SetAutoCheck (see page 274)	Introduces an interface for setting the AutoCheck property of a client.
	SetCaption (see page 274)	Introduces an interface for setting the caption on a client.
	SetCaption (see page 5899)	Sets the caption of a control if the client is linked to the action for this event.
	SetChecked (see page 274)	Introduces an interface for setting the checked state of a client.
	SetDropdownMenu (see page 5899)	This is SetDropdownMenu, a member of class TControlActionLink.

	SetEnabled (see page 275)	Introduces an interface for setting the enabled state of a client.
	SetEnabled (see page 5900)	Sets the enabled state of a control if the client is linked to the action for this event.
	SetEnableDropdown (see page 5900)	This is SetEnableDropdown, a member of class TControlActionLink.
	SetGroupIndex (see page 275)	Introduces an interface for setting the GroupIndex property of a client.
	SetHelpContext (see page 276)	Introduces an interface for setting the help context of a client.
	SetHelpContext (see page 5900)	Sets the HelpContext of a control if the client is linked to the action help properties.
	SetHelpContext (see page 6237)	Sets the Help context of a windowed control if the client is linked to the action for this event.
	SetHelpKeyword (see page 276)	Introduces an interface for setting the help keyword of a client.
	SetHelpKeyword (see page 5901)	Sets the HelpKeyword of a control if the client is linked to the action help properties.
	SetHelpType (see page 277)	Introduces an interface for setting the HelpType property of a client.
	SetHelpType (see page 5901)	Sets the HelpType of a control if the client is linked to the action help properties.
	SetHint (see page 278)	Introduces an interface for setting the hint for a client.
	SetHint (see page 5901)	Sets the hint of a control if the client is linked to the action for this event.
	SetImageIndex (see page 278)	Introduces an interface for setting the image index for a client.
	SetImages (see page 4392)	This is SetImages, a member of class TListViewActionLink.
	SetItemIndex (see page 15749)	This is SetItemIndex, a member of class TListActionLink.
	SetOnExecute (see page 2724)	This is SetOnExecute, a member of class TBasicActionLink.
	SetOnExecute (see page 5902)	Sets the execution event handler for a control if the client is linked to the action for this event.
	SetPopupMenu (see page 5902)	This is SetPopupMenu, a member of class TControlActionLink.
	SetShortCut (see page 279)	Introduces an interface for setting the shortcut for a client.
	SetVisible (see page 279)	Introduces an interface for setting the visible state of a client.
	SetVisible (see page 5902)	Sets the visible state of a control if the client is linked to the action for this event.

Public Events

	Name	Description
	OnChange (see page 2725)	Occurs when an OnChange event is triggered on the action.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2721)	Instantiates and initializes a TBasicActionLink (see page 2720) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2722)	Destroys an instance of an action link object.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 2723)	Invokes the action's Execute method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Update (see page 2724)	Invokes the action's Update method.

Public Properties

	Name	Description
	Action (see page 2725)	Indicates which action the link connects to its client.

27.1.37.2 TListViewActionLink Methods

The methods of the TListViewActionLink class are listed here.

Protected Methods

	Name	Description
V	AddItem (see page 4391)	This is the overview for the AddItem method overload.
V	SetImages (see page 4392)	This is SetImages, a member of class TListViewActionLink.

27.1.37.2.1 AddItem Method

This is the overview for the AddItem method overload.

Overload List

	Name	Description
V	TListViewActionLink.AddItem (String, Integer, Pointer) (see page 4391)	This is AddItem, a member of class TListViewActionLink.
V	TListViewActionLink.AddItem (TListControlItem) (see page 4391)	This is AddItem, a member of class TListViewActionLink.

27.1.37.2.1.1 TListViewActionLink.AddItem Method (String, Integer, Pointer)

Pascal

```
procedure AddItem(ACaption: String; AImageIndex: Integer; DataPtr: Pointer); override;
```

C++

```
virtual __fastcall AddItem(AnsiString ACaption, int AImageIndex, void * DataPtr);
```

Description

This is AddItem, a member of class TListViewActionLink.

27.1.37.2.1.2 TListViewActionLink.AddItem Method (TListControlItem)

Pascal

```
procedure AddItem(AnItem: TListControlItem); override;
```

C++

```
virtual __fastcall AddItem(TListControlItem AnItem);
```

Description

This is AddItem, a member of class TListViewActionLink.

27.1.37.2.2 TListViewActionLink.SetImages Method

Pascal

```
procedure SetImages(Value: TCustomImageList); override;
```

C++

```
virtual __fastcall SetImages(TCustomImageList Value);
```

Description

This is SetImages, a member of class TListViewActionLink.

27.1.38 TMonthCalColors Class

TMonthCalColors defines a color scheme for TDateTimePicker (see page 4049) or TMonthCalendar (see page 4397)'s drop-down calendar.

Class Hierarchy

Pascal

```
TMonthCalColors = class(TPersistent);
```

C++

```
class TMonthCalColors : public TPersistent;
```

File

ComCtrls

Description

The CalColors property of TDateTimePicker (see page 4049) or TMonthCalendar (see page 4397) holds a TMonthCalColors object. TMonthCalColors determines the colors of a calendar display.

See Also

TDateTimePicker (see page 4049)

TCommoncalendar

TDateTimePicker (see page 4049)

TDateTimePicker (see page 4049)

27.1.38.1 TMonthCalColors Members

The following tables list the members exposed by TMonthCalColors.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 4394)	Copies the properties from another TMonthCalColors (see page 4392) instance.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 4394)	Creates and initializes a TMonthCalColors (see page 4392) instance.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Published Properties

	Name	Description
	BackColor (see page 4395)	The background color displayed between months in the calendar.
	MonthBackColor (see page 4395)	The background color of the calendar.
	TextColor (see page 4395)	The color of the font used for dates that fall within the displayed month.
	TitleBackColor (see page 4396)	The background color of the calendar's title bar.
	TitleTextColor (see page 4396)	The color of the font used for the calendar's title bar.
	TrailingTextColor (see page 4396)	The color of the font used for dates not in the currently displayed month.

27.1.38.2 TMonthCalColors.Create Constructor

Creates and initializes a TMonthCalColors (see page 4392) instance.

Pascal

```
constructor Create(AOwner: TCommonCalendar);
```

C++

```
__fastcall TMonthCalColors(TCommonCalendar AOwner);
```

Description

Applications do not need to instantiate TMonthCalColors (see page 4392). This object is created by the calendar display control that uses it. That calendar display control is passed as the AOwner parameter.

27.1.38.3 TMonthCalColors Methods

The methods of the TMonthCalColors class are listed here.

Public Methods

	Name	Description
	Assign (see page 4394)	Copies the properties from another TMonthCalColors (see page 4392) instance.

27.1.38.3.1 TMonthCalColors.Assign Method

Copies the properties from another TMonthCalColors (see page 4392) instance.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call the Assign method to duplicate the contents of another TMonthCalColors (see page 4392) in a new object. If the source parameter is not a TMonthCalColors (see page 4392) instance (or nil (Delphi) or NULL (C++)), Assign raises an EConvertError exception.

See Also

EConvertError (see page 22724)

27.1.38.4 TMonthCalColors Properties

The properties of the TMonthCalColors class are listed here.

Published Properties

	Name	Description
	BackColor (see page 4395)	The background color displayed between months in the calendar.
	MonthBackColor (see page 4395)	The background color of the calendar.
	TextColor (see page 4395)	The color of the font used for dates that fall within the displayed month.

	TitleBackColor (see page 4396)	The background color of the calendar's title bar.
	TitleTextColor (see page 4396)	The color of the font used for the calendar's title bar.
	TrailingTextColor (see page 4396)	The color of the font used for dates not in the currently displayed month.

27.1.38.4.1 TMonthCalColors.BackColor Property

The background color displayed between months in the calendar.

Pascal

```
property BackColor: TColor;
```

C++

```
__property TColor BackColor;
```

Description

BackColor may not be implemented in some Windows versions.

See Also

MonthBackColor (see page 4395)

27.1.38.4.2 TMonthCalColors.MonthBackColor Property

The background color of the calendar.

Pascal

```
property MonthBackColor: TColor;
```

C++

```
__property TColor MonthBackColor;
```

Description

MonthBackColor determines the background color of the main section of the calendar, where the dates are displayed.

See Also

BackColor (see page 4395)

27.1.38.4.3 TMonthCalColors.TextColor Property

The color of the font used for dates that fall within the displayed month.

Pascal

```
property TextColor: TColor;
```

C++

```
__property TColor TextColor;
```

Description

TextColor determines the color of the numerals for dates that fall within the displayed month. To set the color of dates in the last week of the preceding month and the first week of the the next month, use TrailingTextColor (see page 4396).

See Also

TrailingTextColor (see page 4396)

27.1.38.4.4 TMonthCalColors.TitleBackColor Property

The background color of the calendar's title bar.

Pascal

```
property TitleBackColor: TColor;
```

C++

```
__property TColor TitleBackColor;
```

Description

TitleBackColor determines the color of the title bar (above the main section of the calendar), where the month and year are displayed.

See Also

TitleTextColor (see page 4396)

27.1.38.4.5 TMonthCalColors.TitleTextColor Property

The color of the font used for the calendar's title bar.

Pascal

```
property TitleTextColor: TColor;
```

C++

```
__property TColor TitleTextColor;
```

Description

TitleTextColor determines the color in which the month and year are displayed on the calendar's title bar.

See Also

TitleBackColor (see page 4396)

27.1.38.4.6 TMonthCalColors.TrailingTextColor Property

The color of the font used for dates not in the currently displayed month.

Pascal

```
property TrailingTextColor: TColor;
```

C++

```
__property TColor TrailingTextColor;
```

Description

TrailingTextColor determines the color of the numerals for dates that fall in the last week of the preceding month or the first week of the next month.

See Also

TextColor (see page 4395)

27.1.39 TMonthCalendar Class

TMonthCalendar is stand-alone calendar in which a user can select a date or range of dates.

Class Hierarchy

Pascal

```
TMonthCalendar = class(TCommonCalendar);
```

C++

```
class TMonthCalendar : public TCommonCalendar;
```

File

ComCtrls

Description

Use TMonthCalendar to allow a user to specify a date or range of dates using a standard calendar user interface.

TMonthCalendar displays dates according to the system locale (ignoring the BiDiMode (see page 4417) setting).

See Also

List Controls

27.1.39.1 TMonthCalendar Members

The following tables list the members exposed by TMonthCalendar.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.

	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 4408)	Specifies whether the control can size itself automatically to display a full month.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
V	CheckEmptyDate (see page 3571)	Clears the check box associated with the calendar.
V	CheckValidDate (see page 3571)	Confirms that a date falls between MinDate (see page 3579) and MaxDate (see page 3578).
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 4409)	Overrides the inherited ConstrainedResize method.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 4409)	Overrides the inherited CreateParams method.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.

	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 3572)	Creates the windows control for the calendar.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetCalendarHandle (see page 4410)	Returns the calendar control's window handle.
	GetCalStyles (see page 3573)	Returns the style flags that are used when creating the calendar's window.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.

	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	MsgSetCalColors (see page 4410)	Sets the calendar's colors.
V	MsgSetDateTime (see page 4410)	Sets the calendar's current date.
V	MsgSetRange (see page 4411)	Selects a range of dates.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.

	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.

	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
R	CalendarHandle (see page 3576)	Indicates the calendar control's window handle.
	CalExceptionClass (see page 3576)	Specifies the calendar's exception class.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DateTime (see page 3577)	Indicates the date that is marked on the calendar.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MonthDelta (see page 3579)	Specifies how many months that the calendar scrolls.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BoldDays (see page 3570)	Encodes a set of dates for use in the OnGetMonthInfo (see page 3581) event handler.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 4408)	Creates and initializes an instance of TMonthCalendar (see page 4397).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 3570)	Destroys the TCommonCalendar (see page 3558) instance.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.

	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.

R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 4438)	Occurs when the user clicks the control.
	OnContextPopup (see page 4440)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 4440)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 4441)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4443)	Occurs when the user drags an object over a control.
	OnEndDock (see page 4445)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4446)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4447)	Occurs when a control receives the input focus.
	OnExit (see page 4449)	Occurs when the input focus shifts away from one control to another.
	OnGetMonthInfo (see page 4451)	Called whenever a new month is displayed in the calendar.
	OnKeyDown (see page 4451)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 4454)	Occurs when a key is pressed.

	OnKeyUp (see page 4455)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 4457)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4457)	Occurs when the user moves the mouse outside of a control.
	OnStartDock (see page 4457)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4458)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 4412)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 4416)	Specifies how the control is anchored to its parent.
	AutoSize (see page 4416)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BiDiMode (see page 4417)	Specifies the bi-directional mode for the control.
	BorderWidth (see page 4417)	Specifies the width of the control's border.
	CalColors (see page 4417)	Contains the colors used for each part of the calendar.
	Constraints (see page 4418)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Date (see page 4418)	Indicates the date that is marked on the calendar.
	DragCursor (see page 4419)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4419)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4420)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4421)	Controls whether the control responds to mouse, keyboard, and timer events.
	EndDate (see page 4422)	Indicates the last date that in the range of selected dates.
	FirstDayOfWeek (see page 4422)	Specifies the left-most day on the calendar.
	Font (see page 4422)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 4423)	Determines the behavior of the input method editor (IME).
	ImeName (see page 4424)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	MaxDate (see page 4425)	Indicates the maximum date to which users can scroll the calendar.
	MaxSelectRange (see page 4425)	Specifies the maximum number of days that can be selected.
	MinDate (see page 4425)	Indicates the minimum date that can be selected.
	MultiSelect (see page 4426)	Specifies whether multiple dates can be selected on the calendar.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 4426)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 4427)	Determines where a control looks for its font information.
	ParentShowHint (see page 4427)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4428)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 4432)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	ShowToday (see page 4434)	Specifies whether today's date is shown below the calendar.
	ShowTodayCircle (see page 4434)	Specifies whether today's date is circled on the calendar.
	TabOrder (see page 4434)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 4435)	Determines if the user can tab to a control.

	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 4435)	Determines whether the component appears onscreen.
	WeekNumbers (see page 4437)	Specifies whether week numbers are shown to the left of the calendar.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.39.2 TMonthCalendar.Create Constructor

Creates and initializes an instance of TMonthCalendar (see page 4397).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TMonthCalendar(TComponent * AOwner);
```

Description

Create performs the following tasks:

Sets CalExceptionClass (see page 3576) to EMonthCalError (see page 3421).

Sets Width to 176.

Sets Height to 153.

The AOwner parameter assigns an Owner, which is responsible for freeing the calendar's memory.

See Also

CalExceptionClass (see page 3576)

EMonthCalError (see page 3421)

Owner (see page 2807)

Width (see page 5849)

Height (see page 5825)

27.1.39.3 TMonthCalendar Methods

The methods of the TMonthCalendar class are listed here.

Protected Methods

	Name	Description
	CanAutoSize (see page 4408)	Specifies whether the control can size itself automatically to display a full month.
	ConstrainedResize (see page 4409)	Overrides the inherited ConstrainedResize method.
	CreateParams (see page 4409)	Overrides the inherited CreateParams method.
	GetCalendarHandle (see page 4410)	Returns the calendar control's window handle.
	MsgSetCalColors (see page 4410)	Sets the calendar's colors.
	MsgSetDateTime (see page 4410)	Sets the calendar's current date.
	MsgSetRange (see page 4411)	Selects a range of dates.

27.1.39.3.1 TMonthCalendar.CanAutoSize Method

Specifies whether the control can size itself automatically to display a full month.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize (see page 4416) property is true and an attempt is made to resize the calendar. It implements the AutoSize (see page 4416) property by ensuring that NewWidth and NewHeight represent the width and height required to display a full month.

If the values of NewWidth and NewHeight are adjusted to reflect the size needed to display a month, CanAutoSize returns true. Otherwise, it returns false, aborting the resize attempt.

See Also

AutoSize (see page 5803)

27.1.39.3.2 TMonthCalendar.ConstrainedResize Method

Overrides the inherited ConstrainedResize method.

Pascal

```
procedure ConstrainedResize(var MinWidth: Integer; var MinHeight: Integer; var MaxWidth:
Integer; var MaxHeight: Integer); override;
```

C++

```
virtual __fastcall ConstrainedResize(int MinWidth, int MinHeight, int MaxWidth, int MaxHeight);
```

Description

Assures that the size constraints allow a full month to be displayed in the calendar. Then calls the inherited ConstrainedResize method.

The MinWidth, MinHeight, MaxWidth, and MaxHeight parameters are passed to an OnConstrainedResize event handler, where they can be altered before they are applied to the attempted resize.

See Also

OnConstrainedResize (see page 5853)

27.1.39.3.3 TMonthCalendar.CreateParams Method

Overrides the inherited CreateParams method.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

CreateParams incorporates the TMonthCalendar (see page 4397) window styles and then calls the inherited CreateParams method.

See Also

CreateParams (see page 6135)

27.1.39.3.4 TMonthCalendar.GetCalendarHandle Method

Returns the calendar control's window handle.

Pascal

```
function GetCalendarHandle: HWND; override;
```

C++

```
virtual __fastcall HWND GetCalendarHandle();
```

Description

Use GetCalendarHandle to get the calendar control's window handle, which can be used for direct Windows API calls.

27.1.39.3.5 TMonthCalendar.MsgSetCalColors Method

Sets the calendar's colors.

Pascal

```
function MsgSetCalColors(ColorIndex: Integer; ColorValue: TColor): Boolean; override;
```

C++

```
virtual __fastcall Boolean MsgSetCalColors(int ColorIndex, TColor ColorValue);
```

Description

MsgSetCalColors is used to implement the CalColors (see page 4417) property. This protected method sets the calendar's colors using the Windows API call MonthCal_SetColor function.

ColorIndex indicates the area in the calendar for which a new color is specified. It is one of the following values:

ValueCalColors subpropertyArea

Value	CalColors (see page 4417) subproperty	Area
0	Background (displayed between months).	
1	Dates in the current month and text labels.	
2	Background of the title bar.	
3	Text label in the title bar.	
4	Background of the calendar (each month).	
5	Dates on non-current month.	

ColorValue is the new color.

MsgSetCalColors returns true if the colors are successfully applied, false otherwise.

See Also

CalColors (see page 3575)

27.1.39.3.6 TMonthCalendar.MsgSetDateTime Method

Sets the calendar's current date.

Pascal

```
function MsgSetDateTime(Value: TSystemTime): Boolean; override;
```

C++

```
virtual __fastcall Boolean MsgSetDateTime(TSystemTime Value);
```

Description

MsgSetCalColors (see page 4410) is used to implement the Date (see page 4418) property. This protected method sets the calendar's current date using the Windows API call MonthCal_SetCurSel function.

Value is the date and time to set, expressed as a TSystemTime value.

MsgSetDateTime returns true if the date is set successfully, false otherwise.

See Also

Date (see page 3576)

27.1.39.3.7 TMonthCalendar.MsgSetRange Method

Selects a range of dates.

Pascal

```
function MsgSetRange(Flags: Integer; SysTime: PSystemTime): Boolean; override;
```

C++

```
virtual __fastcall Boolean MsgSetRange(int Flags, PSystemTime SysTime);
```

Description

MsgSetCalColors (see page 4410) is used to implement the range specified by the Date (see page 4418) and MaxDate (see page 4425) properties. This protected method sets the selected dates using the Windows API call MonthCal_SetRange function.

The Flags parameter indicates what type of information is included in the SysTime array. It includes the GDTR_MIN flag if a minimum date is supplied and GDTR_MAX if a maximum date is supplied.

The SysTime parameter is a two-element array containing the minimum date as the first element and the maximum date as the second element. These values can be obtained from the MinDate (see page 4425) and MaxDate (see page 4425) properties by calling DateTimeToSystemTime.

MsgSetRange returns true if the range is set successfully, false otherwise.

See Also

Date (see page 3576)

MaxDate (see page 3578)

27.1.39.4 TMonthCalendar Properties

The properties of the TMonthCalendar class are listed here.

Published Properties

	Name	Description
	Align (see page 4412)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4416)	Specifies how the control is anchored to its parent.
	AutoSize (see page 4416)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BiDiMode (see page 4417)	Specifies the bi-directional mode for the control.
	BorderWidth (see page 4417)	Specifies the width of the control's border.

	CalColors (see page 4417)	Contains the colors used for each part of the calendar.
	Constraints (see page 4418)	Specifies the size constraints for the control.
	Date (see page 4418)	Indicates the date that is marked on the calendar.
	DragCursor (see page 4419)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4419)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4420)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4421)	Controls whether the control responds to mouse, keyboard, and timer events.
	EndDate (see page 4422)	Indicates the last date that in the range of selected dates.
	FirstDayOfWeek (see page 4422)	Specifies the left-most day on the calendar.
	Font (see page 4422)	Controls the attributes of text written on or in the control.
	ImeMode (see page 4423)	Determines the behavior of the input method editor (IME).
	ImeName (see page 4424)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	MaxDate (see page 4425)	Indicates the maximum date to which users can scroll the calendar.
	MaxSelectRange (see page 4425)	Specifies the maximum number of days that can be selected.
	MinDate (see page 4425)	Indicates the minimum date that can be selected.
	MultiSelect (see page 4426)	Specifies whether multiple dates can be selected on the calendar.
	ParentBiDiMode (see page 4426)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 4427)	Determines where a control looks for its font information.
	ParentShowHint (see page 4427)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4428)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 4432)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	ShowToday (see page 4434)	Specifies whether today's date is shown below the calendar.
	ShowTodayCircle (see page 4434)	Specifies whether today's date is circled on the calendar.
	TabOrder (see page 4434)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 4435)	Determines if the user can tab to a control.
	Visible (see page 4435)	Determines whether the component appears onscreen.
	WeekNumbers (see page 4437)	Specifies whether week numbers are shown to the left of the calendar.

27.1.39.4.1 TMonthCalendar.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls

into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
```

```
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form’s width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
```

```
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

27.1.39.4.2 TMonthCalendar.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.39.4.3 TMonthCalendar.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints (see page 5812)

CanAutoSize (see page 5741)

27.1.39.4.4 TMonthCalendar.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

27.1.39.4.5 TMonthCalendar.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics or text drawn by the control is clipped to the area within the border.

See Also

ClientRect

BevelOuter (see page 6194)

27.1.39.4.6 TMonthCalendar.CalColors Property

Contains the colors used for each part of the calendar.

Pascal

```
property CalColors: TMonthCalColors;
```

C++

```
__property TMonthCalColors CalColors;
```

Description

Use the CalColors property to get or set the calendar's background color, month background color, text color, title background color, title text color, and trailing text color.

See Also

BoldDays (see page 3570)

MsgSetCalColors (see page 3573)

27.1.39.4.7 TMonthCalendar.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.39.4.8 TMonthCalendar.Date Property

Indicates the date that is marked on the calendar.

Pascal

```
property Date: TDate;
```

C++

```
__property TDate Date;
```

Description

Use Date to get or set the date that is marked on the calendar. The value of Date must lie within the range specified by the MaxDate and MinDate properties.

If MultiSelect is true, the selected range of dates goes from Date to EndDate.

Note: You can also set the marked date with the DateTime property.

See Also

DateTime (see page 3577)

EndDate (see page 3577)

MaxDate (see page 3578)

MinDate (see page 3579)

MultiSelect (see page 3580)

27.1.39.4.9 TMonthCalendar.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

27.1.39.4.10 TMonthCalendar.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

27.1.39.4.11 TMonthCalendar.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
```

```
    AttachMode: TNodeAttachMode;
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

27.1.39.4.12 TMonthCalendar.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.39.4.13 TMonthCalendar.EndDate Property

Indicates the last date that in the range of selected dates.

Pascal

```
property EndDate: TDate;
```

C++

```
__property TDate EndDate;
```

Description

Use EndDate to get or set the last selected date that is marked on the calendar. The first selected date is given by the Date property.

Before you can set the EndDate property, you must set the MultiSelect property to true.

See Also

Date (see page 3576)

MultiSelect (see page 3580)

27.1.39.4.14 TMonthCalendar.FirstDayOfWeek Property

Specifies the left-most day on the calendar.

Pascal

```
property FirstDayOfWeek: TCalDayOfWeek;
```

C++

```
__property TCalDayOfWeek FirstDayOfWeek;
```

Description

Use FirstDayOfWeek to get or set the left-most day on the calendar.

To make the left-most day adjust for the locale, set FirstDayOfWeek to dowLocaleDefault.

See Also

DayOfWeek (see page 22827)

WeekNumbers (see page 3581)

27.1.39.4.15 TMonthCalendar.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name,

Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

27.1.39.4.16 TMonthCalendar.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

27.1.39.4.17 TMonthCalendar.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

27.1.39.4.18 TMonthCalendar.MaxDate Property

Indicates the maximum date to which users can scroll the calendar.

Pascal

```
property MaxDate: TDate;
```

C++

```
__property TDate MaxDate;
```

Description

Use MaxDate to get or set the maximum date to which users can scroll the calendar. The values of the Date and EndDate properties cannot exceed MaxDate.

See Also

Date (see page 3576)

MinDate (see page 3579)

EndDate (see page 3577)

MaxSelectRange (see page 3579)

27.1.39.4.19 TMonthCalendar.MaxSelectRange Property

Specifies the maximum number of days that can be selected.

Pascal

```
property MaxSelectRange: Integer;
```

C++

```
__property int MaxSelectRange;
```

Description

Use MaxSelectRange to limit the number of days that can be selected when MultiSelect is true. To limit ranges so that they fall between specific date values, use the MaxDate and MinDate properties instead.

See Also

Date (see page 3576)

MinDate (see page 3579)

MultiSelect (see page 3580)

EndDate (see page 3577)

MaxDate (see page 3578)

27.1.39.4.20 TMonthCalendar.MinDate Property

Indicates the minimum date that can be selected.

Pascal

```
property MinDate: TDate;
```

C++

```
__property TDate MinDate;
```

Description

Use MinDate to get or set the minimum date that can be selected.

See Also

Date (see page 3576)

MaxDate (see page 3578)

EndDate (see page 3577)

MaxSelectRange (see page 3579)

27.1.39.4.21 TMonthCalendar.MultiSelect Property

Specifies whether multiple dates can be selected on the calendar.

Pascal

```
property MultiSelect: Boolean;
```

C++

```
__property Boolean MultiSelect;
```

Description

Use MultiSelect to specify whether the calendar represents a single date or a range of dates. When MultiSelect is true, the calendar represents the range of dates from Date to EndDate. When MultiSelect is false, the calendar represents only the value of Date.

See Also

Date (see page 3576)

EndDate (see page 3577)

27.1.39.4.22 TMonthCalendar.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

27.1.39.4.23 TMonthCalendar.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

27.1.39.4.24 TMonthCalendar.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.39.4.25 TMonthCalendar.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
```

```
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
```

```
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
```

```
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

```
procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.39.4.26 TMonthCalendar.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.39.4.27 TMonthCalendar.ShowToday Property

Specifies whether today's date is shown below the calendar.

Pascal

```
property ShowToday: Boolean;
```

C++

```
__property Boolean ShowToday;
```

Description

Use ShowToday to specify whether today's date (according to the system clock) is shown below the calendar.

See Also

Date (see page 3576)

ShowTodayCircle (see page 3580)

27.1.39.4.28 TMonthCalendar.ShowTodayCircle Property

Specifies whether today's date is circled on the calendar.

Pascal

```
property ShowTodayCircle: Boolean;
```

C++

```
__property Boolean ShowTodayCircle;
```

Description

Use ShowTodayCircle to specify whether today's date (according to the system clock) is circled on the calendar.

See Also

Date (see page 3576)

ShowToday (see page 3580)

27.1.39.4.29 TMonthCalendar.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a

TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

27.1.39.4.30 TMonthCalendar.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

27.1.39.4.31 TMonthCalendar.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
```

```
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.39.4.32 TMonthCalendar.WeekNumbers Property

Specifies whether week numbers are shown to the left of the calendar.

Pascal

```
property WeekNumbers: Boolean;
```

C++

```
__property Boolean WeekNumbers;
```

Description

Use WeekNumbers to indicate whether week numbers are shown to the left of the calendar. The week numbers correspond to the number of the week in the current year.

See Also

FirstDayOfWeek (see page 3578)

27.1.39.5 TMonthCalendar Events

The events of the TMonthCalendar class are listed here.

Published Events

	Name	Description
	OnClick (see page 4438)	Occurs when the user clicks the control.
	OnContextPopup (see page 4440)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 4440)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 4441)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4443)	Occurs when the user drags an object over a control.
	OnEndDock (see page 4445)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4446)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4447)	Occurs when a control receives the input focus.
	OnExit (see page 4449)	Occurs when the input focus shifts away from one control to another.
	OnGetMonthInfo (see page 4451)	Called whenever a new month is displayed in the calendar.
	OnKeyDown (see page 4451)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 4454)	Occurs when a key is pressed.
	OnKeyUp (see page 4455)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 4457)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4457)	Occurs when the user moves the mouse outside of a control.

	OnStartDock (see page 4457)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4458)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

27.1.39.5.1 TMonthCalendar.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
```

```
      TTabSheet *pPage = new TTabSheet(this);
      pPage->PageControl = PageControl1;
      pPage->Caption = AnsiString("Page") + IntToStr(i);
      TEdit *pEdit = new TEdit(this);
      pEdit->Parent = pPage;
      pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
      pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
    }
    PageControl1Change(Sender);
  }
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

27.1.39.5.2 TMonthCalendar.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.39.5.3 TMonthCalendar.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

27.1.39.5.4 TMonthCalendar.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

27.1.39.5.5 TMonthCalendar.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}
```

```
/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
```

```
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.39.5.6 TMonthCalendar.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.39.5.7 TMonthCalendar.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

27.1.39.5.8 TMonthCalendar.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
```

```
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
```

```
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

27.1.39.5.9 TMonthCalendar.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK

button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

27.1.39.5.10 TMonthCalendar.OnGetMonthInfo Event

Called whenever a new month is displayed in the calendar.

Pascal

```
property OnGetMonthInfo: TOnGetMonthInfoEvent;
```

C++

```
__property TOnGetMonthInfoEvent OnGetMonthInfo;
```

Description

Use the OnGetMonthInfo event to initialize the display properties of a month. In particular, OnGetMonthInfo allows applications to bold individual days in the calendar (such as holidays).

To specify that specific dates are to be bolded, use the BoldDays method to obtain a value that can be returned as the MonthBoldInfo parameter. For example, to bold the first and eighth days of every month that is displayed, call

```
BoldDays([1,8], MonthBoldInfo);
unsigned bolddays[2] = {1,2};
BoldDays(bolddays, MonthBoldInfo);
```

from the OnGetMonthInfo event handler.

See Also

BoldDays (see page 3570)

27.1.39.5.11 TMonthCalendar.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys

that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;
```

```
procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
```

```
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

27.1.39.5.12 TMonthCalendar.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
```

```
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

27.1.39.5.13 TMonthCalendar.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
```

```
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
```

```
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

27.1.39.5.14 TMonthCalendar.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

27.1.39.5.15 TMonthCalendar.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

27.1.39.5.16 TMonthCalendar.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is

automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

27.1.39.5.17 TMonthCalendar.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

27.1.40 TPageControl Class

TPageControl is a set of pages used to make a multiple page dialog box.

Class Hierarchy

Pascal

```
TPageControl = class(TCustomTabControl);
```

C++

```
class TPageControl : public TCustomTabControl;
```

File

ComCtrls

Description

Use TPageControl to create a multiple page dialog or tabbed notebook. TPageControl displays multiple overlapping pages that are TTabSheet (see page 4919) objects. The user selects a page by clicking the page's tab that appears at the top of the control. To add a new page to a TPageControl object at design time, right-click the TPageControl object and choose New Page.

To create a tabbed control that uses only a single body portion (page), use TTabControl (see page 4841) instead.

Note: When using one of the page controls, if you want to restrict a user from switching to a tab, you cannot set TTabSheet.Enabled (see page 4493) to **false** to accomplish that restriction. Instead, use the OnChanging (see page 4516) event to prevent the user from selecting a tab.

See Also

TTabControl (see page 4841)

TTabSheet (see page 4919)

Grouping Controls

Page Controls

Tab Controls

27.1.40.1 TPageControl Members

The following tables list the members exposed by TPageControl.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.

	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 3933)	Calculates the control's display area given a rectangle.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanChange (see page 3933)	Change (see page 3934) permission event dispatcher.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
V	CanShowTab (see page 4473)	Indicates whether a tab sheet in the page control can be selected.
V	Change (see page 4473)	Responds to changes in the page control.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.

	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 3935)	Sets up the window-creation parameters for the tab control.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 3935)	Creates the window for a tab control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
V	DoAddDockClient (see page 4474)	Provides necessary adjustments when a new client is docked in the page control.
V	DockOver (see page 4474)	Indicates whether an object that is dragged over the page control can be docked.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
V	DoRemoveDockClient (see page 4474)	Provides necessary adjustments when a control is undocked from the page control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.

	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawTab (see page 3935)	Tab drawing event dispatcher.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
V	GetChildren (see page 4476)	Calls a specified procedure for every page in the page control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetImageIndex (see page 4477)	Specifies the image to display for a particular page.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPageFromDockClient (see page 4477)	Returns the tab sheet on which a client is docked.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
V	GetSiteInfo (see page 4477)	Returns the page control's docking information.
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 4478)	Initializes the page control after all its parts have been loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.

	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 3937)	Event dispatcher for creation and destruction of child objects.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActivePage (see page 4480)	This is SetActivePage, a member of class TPageControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
V	SetChildOrder (see page 4480)	Assigns indexes to the pages that reflect the order in which pages are loaded from a stream.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.

V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetTabIndex (see page 4480)	This is SetTabIndex, a member of class TPageControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 4481)	Displays a specified page.
V	UpdateActivePage (see page 4481)	Updates the ActivePage (see page 4482) property when changes are made to the Pages (see page 4498) property.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateTabImages (see page 3939)	Updates the tab images.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
R	DisplayRect (see page 3941)	Specifies the bounding rectangle of a tab control client area.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.

	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	MultiSelect (see page 3943)	Specifies whether multiple tabs can be selected.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Tabs (see page 3948)	Contains the list of text strings that label the tabs of the tab control.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 4470)	Creates an instance of PageControl.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 4472)	Destroys an instance of TPageControl (see page 4459).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.

	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindNextPage (see page 4475)	Returns the next page in the page control before or after a specified page.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHitTestInfoAt (see page 3936)	Returns information about the location of a point relative to the client area of the tab control.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	IndexOfTabAt (see page 3936)	Indicates the index of the tab at a specified point.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.

	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	RowCount (see page 3938)	Sets TabIndex (see page 3946) property.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollTabs (see page 3938)	Scrolls the tabs that are visible when the tab control is not multi-line.
	SelectNextPage (see page 4478)	Changes the ActivePage (see page 4482) to the first visible page that is before or after the currently active page.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	TabRect (see page 3939)	Returns the bounding rectangle for a specified tab.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	ActivePageIndex (see page 4484)	Specifies the page currently displayed by the page control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 3940)	Gives access to the tab control's canvas.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
R	PageCount (see page 4497)	Indicates the number of pages in the TPageControl (see page 4459) object.
R	Pages (see page 4498)	Lists all the pages in the TPageControl (see page 4459).
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.

	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 4516)	Occurs after a new tab is selected.
	OnChanging (see page 4516)	Occurs immediately before a new tab is selected.
	OnContextPopup (see page 4517)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDockDrop (see page 4518)	Occurs when another control is docked to the control.
	OnDockOver (see page 4520)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 4522)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4524)	Occurs when the user drags an object over a control.
	OnDrawTab (see page 4527)	Occurs when a tab is about to be drawn.
	OnEndDock (see page 4527)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4527)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4528)	Occurs when a control receives the input focus.
	OnExit (see page 4531)	Occurs when the input focus shifts away from one control to another.
	OnGetImageIndex (see page 4532)	Occurs when a tab is about to display its associated image.
	OnGetSiteInfo (see page 4533)	Returns the control's docking information.
	OnMouseActivate (see page 4535)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4536)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4538)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4538)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4539)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4541)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 4543)	Occurs immediately after the control is resized.
	OnStartDock (see page 4544)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4544)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 4545)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	ActivePage (see page 4482)	Specifies the page currently displayed by the page control.
	Align (see page 4485)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 4488)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 4489)	Specifies the bi-directional mode for the control.
	Constraints (see page 4489)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DockSite (see page 4490)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 4491)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4491)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4491)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4493)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4493)	Controls the attributes of text written on or in the control.

	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HotTrack (see page 4495)	Determines whether labels on the tab under the mouse are automatically highlighted.
	Images (see page 4495)	Specifies the images drawn in tabs.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	MultiLine (see page 4495)	Determines whether the tabs can appear on more than one row.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OwnerDraw (see page 4497)	Specifies whether the tab control handles its own painting.
	ParentBiDiMode (see page 4500)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 4500)	Determines where a control looks for its font information.
	ParentShowHint (see page 4501)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4501)	Identifies the pop-up menu associated with the control.
	RaggedRight (see page 4506)	Specifies whether rows of tabs stretch to fill the width of the control.
	ScrollOpposite (see page 4506)	Determines how the rows of tabs are scrolled in a multi-line tab control.
	ShowHint (see page 4506)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 4508)	Specifies the style of the tab control.
	TabHeight (see page 4508)	Specifies the height, in pixels, of the tabs in the tab control.
	TabIndex (see page 4509)	Identifies the selected tab on a tab control.
	TabOrder (see page 4511)	Indicates the position of the control in its parent's tab order.
	TabPosition (see page 4512)	Determines whether tabs appear at the top or bottom.
	TabStop (see page 4512)	Determines if the user can tab to a control.
	TabWidth (see page 4512)	Specifies the horizontal size, in pixels, of the tabs in the tab control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 4514)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.40.2 TPageControl.Create Constructor

Creates an instance of PageControl.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TPageControl(TComponent * AOwner);
```

Description

Call Create to instantiate a page control at runtime. Page controls added to forms at design time are created automatically.

AOwner is the form in which the page control should appear. It becomes the value of the Owner property.

When creating a page control at runtime, the pages must be added programmatically. To add a page to a page control at runtime, create the TTabSheet (see page 4919) object for the page and set its PageControl property.

C++ Examples:

```
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
```

```
    end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

ControlStyle (see page 5814)

TTabSheet (see page 4919)

Owner (see page 2807)

ParentWindow (see page 6210)

27.1.40.3 TPageControl.Destroy Destructor

Destroys an instance of TPageControl (see page 4459).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPageControl();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TPageControl (see page 4459) reference is not nil, and only then calls Destroy. Destroy frees the TList object that holds the pages of the control, and then calls the inherited destructor.

See Also

Free (see page 22300)

27.1.40.4 TPageControl Methods

The methods of the TPageControl class are listed here.

Protected Methods

	Name	Description
	CanShowTab (see page 4473)	Indicates whether a tab sheet in the page control can be selected.
	Change (see page 4473)	Responds to changes in the page control.
	DoAddDockClient (see page 4474)	Provides necessary adjustments when a new client is docked in the page control.
	DockOver (see page 4474)	Indicates whether an object that is dragged over the page control can be docked.
	DoRemoveDockClient (see page 4474)	Provides necessary adjustments when a control is undocked from the page control.
	GetChildren (see page 4476)	Calls a specified procedure for every page in the page control.
	GetImageIndex (see page 4477)	Specifies the image to display for a particular page.

	GetPageFromDockClient (see page 4477)	Returns the tab sheet on which a client is docked.
V	GetSiteInfo (see page 4477)	Returns the page control's docking information.
V	Loaded (see page 4478)	Initializes the page control after all its parts have been loaded from a stream.
	SetActivePage (see page 4480)	This is SetActivePage, a member of class TPageControl.
V	SetChildOrder (see page 4480)	Assigns indexes to the pages that reflect the order in which pages are loaded from a stream.
V	SetTabIndex (see page 4480)	This is SetTabIndex, a member of class TPageControl.
V	ShowControl (see page 4481)	Displays a specified page.
V	UpdateActivePage (see page 4481)	Updates the ActivePage (see page 4482) property when changes are made to the Pages (see page 4498) property.

Public Methods

	Name	Description
	FindNextPage (see page 4475)	Returns the next page in the page control before or after a specified page.
	SelectNextPage (see page 4478)	Changes the ActivePage (see page 4482) to the first visible page that is before or after the currently active page.

27.1.40.4.1 TPageControl.CanShowTab Method

Indicates whether a tab sheet in the page control can be selected.

Pascal

```
function CanShowTab(TabIndex: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanShowTab(int TabIndex);
```

Description

Call CanShowTab to determine whether a tab can be selected or accessed.

TabIndex (see page 4509) is the index of the tab in the list of tab sheets maintained by the Pages (see page 4498) property.

CanShowTab returns true if the specified Tab sheet is enabled (its Enabled (see page 4493) property is true).

See Also

Pages (see page 4498)

Enabled (see page 5821)

TabVisible (see page 4956)

27.1.40.4.2 TPageControl.Change Method

Responds to changes in the page control.

Pascal

```
procedure Change; override;
```

C++

```
virtual __fastcall Change();
```

Description

Change is called automatically when Windows notifies the page control of a change. TPageControl (see page 4459) responds to this notification by updating its display to show the new current page.

See Also

ActivePage (see page 4482)

27.1.40.4.3 TPageControl.DoAddDockClient Method

Provides necessary adjustments when a new client is docked in the page control.

Pascal

```
procedure DoAddDockClient(Client: TControl; const ARect: TRect); override;
```

C++

```
virtual __fastcall DoAddDockClient(TControl * Client, const TRect ARect);
```

Description

DoAddDockClient is called when a new client is docked to the page control. It sets the page control's tab sheet as the new client's parent.

Client is the client being docked.

See Also

DoRemoveDockClient (see page 4474)

27.1.40.4.4 TPageControl.DockOver Method

Indicates whether an object that is dragged over the page control can be docked.

Pascal

```
procedure DockOver(Source: TDragDockObject; X: Integer; Y: Integer; State: TDragState; var
Accept: Boolean); override;
```

C++

```
virtual __fastcall DockOver(TDragDockObject Source, int X, int Y, TDragState State, Boolean
Accept);
```

Description

DockOver is called automatically when the user drags a dockable window over the page control.

The Source parameter describes the object that is being dragged.

The X and Y parameters give the current position of the dragged control.

The State parameter indicates how the mouse is moving.

DockOver initializes the Source parameter to reflect the dockable region of the page control and generates an OnDockOver (see page 4520) event. An OnDockOver (see page 4520) event handler can change the Accept parameter to false if the Source control should not be docked.

Note: DockOver is only called if the DockSite (see page 4490) property is true.

See Also

DockSite (see page 6204)

OnDockOver (see page 6216)

27.1.40.4.5 TPageControl.DoRemoveDockClient Method

Provides necessary adjustments when a control is undocked from the page control.

Pascal

```
procedure DoRemoveDockClient(Client: TControl); override;
```

C++

```
virtual __fastcall DoRemoveDockClient(TControl * Client);
```

Description

DoRemoveDockClient is called when a client is undocked from the page control. It selects the next page in the tab sheet.

Client is the client being undocked.

See Also

DoAddDockClient (see page 4474)

27.1.40.4.6 TPageControl.FindNextPage Method

Returns the next page in the page control before or after a specified page.

Pascal

```
function FindNextPage(CurPage: TTabSheet; GoForward: Boolean; CheckTabVisible: Boolean):
TTabSheet;
```

C++

```
__fastcall TTabSheet FindNextPage(TTabSheet CurPage, Boolean GoForward, Boolean
CheckTabVisible);
```

Description

Call FindNextPage to locate a page in the page control by its position relative to the page specified by the CurPage parameter. If the GoForward parameter is true, FindNextPage returns the next page in the page control after CurPage. If GoForward is false, FindNextPage returns the previous page. If CurPage is not a page in the page control, FindNextPage returns the first page when GoForward is true or the last page when GoForward is false. If GoForward is true and CurPage is the last page in the page control, FindNextPage returns the first page. If GoForward is false and CurPage is the first page in the page control, FindNextPage returns the last page.

When CheckTabVisible is true, FindNextPage returns the next page with a TabVisible property set to true, if CheckTabVisible is false, FindNextPage returns the next page, including pages with TabVisible set to false.

C++ Examples:

```
/*
This example requires a new TPageControl, with no new pages
created at design time.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
Page Control's OnChange event handler displays a message
dialog when the user changes tabs.  The message dialog
contains the captions for the tabs immediately before and
after the active tab.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(PageControl1);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    pPage->Name = AnsiString("ts") + pPage->Caption;
  }
}
```

```
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  AnsiString PrevCaption, NextCaption;
  TPageControl *pPC = (TPageControl *)Sender;
  PrevCaption = pPC->FindNextPage(pPC->ActivePage, false, false)->Caption;
  NextCaption = pPC->FindNextPage(pPC->ActivePage, true, false)->Caption;
  ShowMessage(AnsiString("Previous tab caption: '") + PrevCaption + AnsiString("'  Next tab
Caption: '") + NextCaption + AnsiString("'"));
}
```

Delphi Examples:

```
{
This example requires a new TPageControl, with no new pages
created at design time.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
Page Control's OnChange event handler displays a message
dialog when the user changes tabs.  The message dialog
contains the captions for the tabs immediately before and
after the active tab.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet #' + IntToStr(i);
    end;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
var
  PrevCaption, NextCaption: ShortString;
begin
  with (Sender as TPageControl) do
  begin
    PrevCaption :=
      FindNextPage(ActivePage, False, False).Caption;
    NextCaption :=
      FindNextPage(ActivePage, True, False).Caption;
  end;
  ShowMessage('Previous tab caption: "' + PrevCaption +
    '"    Next tab Caption: "' + NextCaption + '"');
end;
```

See Also

Pages (see page 4498)

SelectNextPage (see page 4478)

TabVisible (see page 4956)

27.1.40.4.7 TPageControl.GetChildren Method

Calls a specified procedure for every page in the page control.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

Call GetChildren to execute a callback for every page in the page control. The Proc parameter is the callback. The Root parameter is ignored by TPageControl (see page 4459).

GetChildren is intended primarily for the streaming system that loads and saves components.

See Also

Pages (see page 4498)

27.1.40.4.8 TPageControl.GetImageIndex Method

Specifies the image to display for a particular page.

Pascal

```
function GetImageIndex(TabIndex: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetImageIndex(int TabIndex);
```

Description

Override GetImageIndex in a derived class to specify the index of the image to display for a particular page.

TabIndex (see page 4509) is the page's index that is about to be displayed.

Integer is the image's index (in the Images (see page 4495) property) that will be displayed in the tab.

By default, if there is no OnGetImageIndex (see page 4532) event handler, GetImageIndex returns the ImageIndex property for the page with the specified index; otherwise, GetImageIndex calls the inherited GetImageIndex method, returning the result.

27.1.40.4.9 TPageControl.GetPageFromDockClient Method

Returns the tab sheet on which a client is docked.

Pascal

```
function GetPageFromDockClient(Client: TControl): TTabSheet;
```

C++

```
__fastcall TTabSheet GetPageFromDockClient(TControl * Client);
```

Description

Call GetPageFromDockClient to get the tab sheet on which Client is docked.

Returns nil (Delphi) or NULL (C++) if Client is not docked.

27.1.40.4.10 TPageControl.GetSiteInfo Method

Returns the page control's docking information.

Pascal

```
procedure GetSiteInfo(Client: TControl; var InfluenceRect: TRect; MousePos: TPoint; var
CanDock: Boolean); override;
```

C++

```
virtual __fastcall GetSiteInfo(TControl * Client, TRect InfluenceRect, TPoint MousePos,
Boolean CanDock);
```

Description

Call GetSiteInfo to determine the docking region of the page control and whether a dragged window can be docked.

Client is a potential docking client.

InfluenceRect returns the area in which the client can be docked.

MousePos is the current mouse position.

CanDock indicates whether the client can be docked in the page control.

GetSiteInfo Initializes CanDock to true if Client is not already docked in the page control. It then inflates the influence rectangle by 10 pixels and generates an OnGetSiteInfo (see page 4533) event.

See Also

OnGetSiteInfo (see page 6223)

DockClients (see page 6201)

27.1.40.4.11 TPageControl.Loaded Method

Initializes the page control after all its parts have been loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Loaded is called automatically when the form containing the page control is first loaded into memory. TPageControl (see page 4459) overrides the Loaded method to ensure that all tab sheets with a Highlighted property of true appear highlighted.

See Also

Highlighted (see page 4942)

Loaded (see page 2782)

27.1.40.4.12 TPageControl.SelectNextPage Method

Changes the ActivePage (see page 4482) to the first visible page that is before or after the currently active page.

Pascal

```
procedure SelectNextPage(GoForward: Boolean; CheckTabVisible: Boolean = True);
```

C++

```
__fastcall SelectNextPage(Boolean GoForward, Boolean CheckTabVisible = True);
```

Description

Use SelectNextPage to select the page next to the currently active page.

If the GoForward parameter is true, SelectNextPage selects the first page after ActivePage (see page 4482); if GoForward is false, SelectNextPage selects the first page previous to ActivePage (see page 4482).

If CheckTabVisible is true, SelectNextPage only considers pages with their TabVisible property set to true.

Note: To select a page at design-time, right-click the TPageControl (see page 4459) and choose Previous Page or Next

Page.

C++ Examples:

```
/*
This example requires a new TPageControl, as-is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
evaluates as true.  If Button is not btNext, the previous Tab Page
is selected.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
const
  TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *page = new TTabSheet(PageControl1);
    page->PageControl = PageControl1;
    page->Caption = AnsiString("Page") + IntToStr(i);
    page->Brush->Color = colorPalette[i];
  }
}

void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  PageControl1->SelectNextPage(Button == btNext);
}
```

Delphi Examples:

```
{
This example requires a new TPageControl, as-is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
evaluates as true.  If Button is not btNext, the previous Tab Page
is selected.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
const
  colorPalette: Array[0..11] of TColor =
    (clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet #' + IntToStr(i);
      Brush.Color := colorPalette[i];
    end;
end;
```

```
procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
begin
  PageControl1.SelectNextPage(Button = btNext);
end;
```

See Also

ActivePage (see page 4482)

FindNextPage (see page 4475)

TabVisible (see page 4956)

27.1.40.4.13 TPageControl.SetActivePage Method

Pascal

```
procedure SetActivePage(Page: TTabSheet);
```

C++

```
__fastcall SetActivePage(TTabSheet Page);
```

Description

This is SetActivePage, a member of class TPageControl.

27.1.40.4.14 TPageControl.SetChildOrder Method

Assigns indexes to the pages that reflect the order in which pages are loaded from a stream.

Pascal

```
procedure SetChildOrder(Child: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Child, int Order);
```

Description

SetChildOrder is called automatically when the children of the page control are loaded from a stream. It assigns each page an index that reflects the order in which pages are loaded and saved.

The Child parameter is a page of the page control. The Order parameter indicates when the page is loaded or saved. The first page loaded or saved is called with Order set to 0, the second with Order set to 1, and so on.

SetChildOrder assigns the PageIndex property of the individual pages to the value of Order.

See Also

PageIndex (see page 4946)

27.1.40.4.15 TPageControl.SetTabIndex Method

Pascal

```
procedure SetTabIndex(Value: Integer); override;
```

C++

```
virtual __fastcall SetTabIndex(int Value);
```

Description

This is SetTabIndex, a member of class TPageControl.

27.1.40.4.16 TPageControl.ShowControl Method

Displays a specified page.

Pascal

```
procedure ShowControl(AControl: TControl); override;
```

C++

```
virtual __fastcall ShowControl(TControl * AControl);
```

Description

ShowControl is called automatically when a particular page of the page control should be visible. The AControl parameter specifies the page that should be displayed on screen.

ShowControl makes the specified page active it is not already active, and sets its own Visible (see page 4514) property to true. It then calls the ShowControl method of its parent object to ensure that the page control is showing (and therefore able to display the active page).

See Also

ActivePage (see page 4482)

Visible (see page 5847)

27.1.40.4.17 TPageControl.UpdateActivePage Method

Updates the ActivePage (see page 4482) property when changes are made to the Pages (see page 4498) property.

Pascal

```
procedure UpdateActivePage; virtual;
```

C++

```
virtual __fastcall UpdateActivePage();
```

Description

This protected method is used internally when the pages listed in the Pages (see page 4498) property change. It ensures that the page control is internally consistent and that the active page is displayed.

See Also

Pages (see page 4498)

27.1.40.5 TPageControl Properties

The properties of the TPageControl class are listed here.

Public Properties

	Name	Description
	ActivePageIndex (see page 4484)	Specifies the page currently displayed by the page control.
R	PageCount (see page 4497)	Indicates the number of pages in the TPageControl (see page 4459) object.
R	Pages (see page 4498)	Lists all the pages in the TPageControl (see page 4459).

Published Properties

	Name	Description
	ActivePage (see page 4482)	Specifies the page currently displayed by the page control.

	Align (see page 4485)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4488)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 4489)	Specifies the bi-directional mode for the control.
	Constraints (see page 4489)	Specifies the size constraints for the control.
	DockSite (see page 4490)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 4491)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4491)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4491)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4493)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4493)	Controls the attributes of text written on or in the control.
	HotTrack (see page 4495)	Determines whether labels on the tab under the mouse are automatically highlighted.
	Images (see page 4495)	Specifies the images drawn in tabs.
	MultiLine (see page 4495)	Determines whether the tabs can appear on more than one row.
	OwnerDraw (see page 4497)	Specifies whether the tab control handles its own painting.
	ParentBiDiMode (see page 4500)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 4500)	Determines where a control looks for its font information.
	ParentShowHint (see page 4501)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4501)	Identifies the pop-up menu associated with the control.
	RaggedRight (see page 4506)	Specifies whether rows of tabs stretch to fill the width of the control.
	ScrollOpposite (see page 4506)	Determines how the rows of tabs are scrolled in a multi-line tab control.
	ShowHint (see page 4506)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 4508)	Specifies the style of the tab control.
	TabHeight (see page 4508)	Specifies the height, in pixels, of the tabs in the tab control.
	TabIndex (see page 4509)	Identifies the selected tab on a tab control.
	TabOrder (see page 4511)	Indicates the position of the control in its parent's tab order.
	TabPosition (see page 4512)	Determines whether tabs appear at the top or bottom.
	TabStop (see page 4512)	Determines if the user can tab to a control.
	TabWidth (see page 4512)	Specifies the horizontal size, in pixels, of the tabs in the tab control.
	Visible (see page 4514)	Determines whether the component appears onscreen.

27.1.40.5.1 TPageControl.ActivePage Property

Specifies the page currently displayed by the page control.

Pascal

```
property ActivePage: TTabSheet;
```

C++

```
__property TTabSheet ActivePage;
```

Description

Read ActivePage to determine which page is selected by the user. The user selects a page by clicking the mouse on its tab. Set ActivePage to bring a page to the foreground. Only one page can be active at a time.

To change the ActivePage to the page that precedes or follows the active page, use the SelectNextPage (see page 4478) method. To iterate through all the pages to locate a specific page, use the FindNextPage (see page 4475) method.

C++ Examples:

```
/*
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
```

```
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
*/
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  if (pCB)
    PageControl1->ActivePage = (TTabSheet *)(pCB->Items->Objects[pCB->ItemIndex]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
    ComboBox1->Items->AddObject(PageControl1->Pages[i]->Name, PageControl1->Pages[i]);
    ComboBox1->ItemIndex = 0;
  }
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to PageControl1.PageCount - 1 do
    ComboBox1.Items.AddObject(PageControl1.Pages[i].Name,
```

```
      PageControl1.Pages[i]);
    ComboBox1.ItemIndex := 0;
  end;

  procedure TForm1.ComboBox1Change(Sender: TObject);
  begin
    if (Sender is TComboBox) then
      with (Sender as TComboBox) do
        PageControl1.ActivePage := TTabSheet(Items.Objects[ItemIndex]);
  end;
  {
  This example requires a TPageControl populated by several
  TabSheets, a TEdit and a TButton. When the button is
  clicked, the caption of the active tab changes to that of
  the TEdit.  These lines of code could also be written:
  PageControl1.ActivePage.Caption := Edit1.Text;
  This would obviate the need for a TTabSheet variable since
  PageControl1.->ActivePage is already of type TTabSheet.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    sheet: TTabSheet;
  begin
    sheet := PageControl1.ActivePage;
    sheet.Caption := Edit1.Text;
  end;
```

See Also

FindNextPage (see page 4475)

SelectNextPage (see page 4478)

ActivePageIndex (see page 4484)

27.1.40.5.2 TPageControl.ActivePageIndex Property

Specifies the page currently displayed by the page control.

Pascal

```
property ActivePageIndex: Integer;
```

C++

```
__property int ActivePageIndex;
```

Description

Use ActivePageIndex to get or set the active page by index rather than by object. The value of ActivePageIndex identifies a page in the Pages (see page 4498) property array. Changing the value of ActivePageIndex changes the ActivePage (see page 4482) property, and vice versa.

If there is no active page, reading ActivePageIndex gives a value of -1.

Setting ActivePageIndex to a value that is out of bounds (less than 0 or greater than PageCount (see page 4497) - 1) results in the page control having no active page.

See Also

ActivePage (see page 4482)

Pages (see page 4498)

PageCount (see page 4497)

27.1.40.5.3 TPageControl.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
```

```
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
```

```
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
```

```
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

27.1.40.5.4 TPageControl.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.40.5.5 TPageControl.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

27.1.40.5.6 TPageControl.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.40.5.7 TPageControl.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite: Boolean;
```

C++

```
__property Boolean DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
```

```
    // Only 2 dock clients means the host must be destroyed and
    // the remaining window undocked to its old position and size.
    // Recall that OnUnDock gets called before the undocking
    // actually occurs.
    if Client is TDockableForm then
      TDockableForm(Client).DockSite := True;
    if (DockClientCount = 2) and (NewTarget <> Self) then
      PostMessage(Self.Handle, WM_CLOSE, 0, 0);
    UpdateCaption(Client);
  end;
```

27.1.40.5.8 TPageControl.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

27.1.40.5.9 TPageControl.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

27.1.40.5.10 TPageControl.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
```

```
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

27.1.40.5.11 TPageControl.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.40.5.12 TPageControl.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

27.1.40.5.13 TPageControl.HotTrack Property

Determines whether labels on the tab under the mouse are automatically highlighted.

Pascal

```
property HotTrack: Boolean;
```

C++

```
__property Boolean HotTrack;
```

Description

Set HotTrack to true to provide visual feedback to the user about which tab would be selected if the mouse button is clicked. HotTrack is especially useful when the tab control contains multiple rows of tabs.

See Also

MultiLine (see page 3942)

HotImages

27.1.40.5.14 TPageControl.Images Property

Specifies the images drawn in tabs.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Images specifies the images that are drawn in tabs. If DisabledImages is defined, those images are used instead for disabled tabs; otherwise, grayed versions of the images are used.

In the implementation of TCustomTabControl, the order of images in image list properties directly corresponds to the associated tabs in the control. In the descendant class, TPageControl, this behavior is replaced by support for individual TTabSheet objects, each with an ImageIndex property. In any case, the application programmer can define customized image displays by defining an OnGetImageIndex event handler.

See Also

OnGetImageIndex (see page 3956)

Pages (see page 4498)

ImageIndex (see page 4943)

DisabledImages

27.1.40.5.15 TPageControl.MultiLine Property

Determines whether the tabs can appear on more than one row.

Pascal

```
property MultiLine: Boolean;
```

C++

```
__property Boolean MultiLine;
```

Description

Use MultiLine to determine how the tabs are displayed. If MultiLine is true, the tabs are displayed on more than one row when the number of tabs exceeds the number that fits across the top of the tab control. How many rows is determined by how many tabs are in the tab control. If MultiLine is false, the tabs are displayed on one row only, and the user must scroll the displayed scroll arrows to view all the tabs.

C++ Examples:

```
/*
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
*/

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->MultiLine = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->Caption = "Stack Tabs?";
  for (int i = 0; i < 20; i++)
  {
    TTabSheet *pPage = new TTabSheet(PageControl1);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
  }
}
```

Delphi Examples:

```
{
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.MultiLine := CheckBox1.Checked;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Checkbox1.Caption := 'Stack Tabs?';
  for i := 0 to 20 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'Tab #' + IntToStr(i);
    end;
```

```
end;
```

See Also

ScrollOpposite (see page 3944)

27.1.40.5.16 TPageControl.OwnerDraw Property

Specifies whether the tab control handles its own painting.

Pascal

```
property OwnerDraw: Boolean;
```

C++

```
__property Boolean OwnerDraw;
```

Description

Set OwnerDraw to true to override any default painting behavior.

See Also

Canvas (see page 3940)

OnDrawTab (see page 3956)

27.1.40.5.17 TPageControl.PageCount Property

Indicates the number of pages in the TPageControl (see page 4459) object.

Pascal

```
property PageCount: Integer;
```

C++

```
__property int PageCount;
```

Description

Use PageCount to determine the number of pages listed by the Pages (see page 4498) property. If the MultiLine (see page 4495) property is false, not all pages may have tabs visible to the user.

C++ Examples:

```
/*
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
*/
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  if (pCB)
    PageControl1->ActivePage = (TTabSheet *)(pCB->Items->Objects[pCB->ItemIndex]);
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
    ComboBox1->Items->AddObject(PageControl1->Pages[i]->Name, PageControl1->Pages[i]);
    ComboBox1->ItemIndex = 0;
  }
}
```

Delphi Examples:

```
{
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to PageControl1.PageCount - 1 do
    ComboBox1.Items.AddObject(PageControl1.Pages[i].Name,
      PageControl1.Pages[i]);
  ComboBox1.ItemIndex := 0;
end;

procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  if (Sender is TComboBox) then
    with (Sender as TComboBox) do
      PageControl1.ActivePage := TTabSheet(Items.Objects[ItemIndex]);
end;
```

See Also

MultiLine (see page 3942)

Pages (see page 4498)

27.1.40.5.18 TPageControl.Pages Property

Lists all the pages in the TPageControl (see page 4459).

Pascal

```
property Pages [Index: Integer]: TTabSheet;
```

C++

```
__property TTabSheet Pages[int Index];
```

Description

Use Pages to gain direct access to a particular page in the page control. For example, use Pages to disable individual pages in the page control. Specify which page to access using Index, where an Index of 0 specifies the first page, an Index of 1 specifies the second page, and so on. Each page is a TTabSheet (see page 4919) object.

To locate a page in the page control by its position relative to another page, use the FindNextPage (see page 4475) method.

Note: Pages is a read-only property. To add a tab sheet to a page control at design time, right click and select New Page. To add a tab sheet to a page control at runtime, create the tab sheet and set its PageControl property to the page control.

C++ Examples:

```
/*
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
     AnsiString s = PageControl1->Pages[i]->Caption;
     if (Canvas->TextWidth(s) * 2 > PageControl1->TabWidth)
        PageControl1->TabWidth = Canvas->TextWidth(s) * 2;
      if (Canvas->TextHeight(s) * 2 > PageControl1->TabHeight)
        PageControl1->TabHeight = Canvas->TextHeight(s) * 2;
  }
}
```

Delphi Examples:

```
{
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  with PageControl1 do
  begin
    for i := 0 to PageCount - 1 do
    begin
      if (Canvas.TextWidth(Pages[i].Caption) * 2) > TabWidth then
        TabWidth := Canvas.TextWidth(Pages[i].Caption) * 2;
      if (Canvas.TextHeight(Pages[i].Caption) * 2) > TabHeight then
        TabHeight := Canvas.TextHeight(Pages[i].Caption) * 2;
    end;
  end;
end;
```

See Also

ActivePage (see page 4482)

FindNextPage (see page 4475)

PageCount (see page 4497)

PageControl (see page 4944)

27.1.40.5.19 TPageControl.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

27.1.40.5.20 TPageControl.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

27.1.40.5.21 TPageControl.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.40.5.22 TPageControl.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
```

```
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
```

```
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
```

```
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
```

```
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
```

```
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.40.5.23 TPageControl.RaggedRight Property

Specifies whether rows of tabs stretch to fill the width of the control.

Pascal

```
property RaggedRight: Boolean;
```

C++

```
__property Boolean RaggedRight;
```

Description

Use RaggedRight to get or set whether rows of tabs stretch to fill the width of the control.

27.1.40.5.24 TPageControl.ScrollOpposite Property

Determines how the rows of tabs are scrolled in a multi-line tab control.

Pascal

```
property ScrollOpposite: Boolean;
```

C++

```
__property Boolean ScrollOpposite;
```

Description

Set ScrollOpposite to determine where previous rows of tabs in a multi-line tab control move when the user selects a tab in another row. When ScrollOpposite is true, previous rows of tabs are moved to the bottom of the tab control if TabPosition is tpTop, or to the top if TabPosition is tpBottom. When ScrollOpposite is false, previous rows of tabs are moved to the back of all other rows of tabs.

For example, in a tab control with three rows of tabs at the top, if the user selects a tab in the second row, ScrollOpposite determines where the first row moves. If ScrollOpposite is true, the second row now appears at the front (bottom) row on top of the tab control, followed by the third row behind (above) it. The first row moves to the bottom of the tab control. If ScrollOpposite is false, the first row moves to the back (top), so that the tab control displays the second row in front (at the bottom of the tab region), followed by the third row (now in the middle), followed by the first row.

Note: Setting ScrollOpposite to true automatically sets the MultiLine property to true.

See Also

MultiLine (see page 3942)

TabPosition (see page 3948)

27.1.40.5.25 TPageControl.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
```

```
    Edit1.Hint := 'Enter your name';
    Edit1.ShowHint := True;
    with ListBox1 do
    begin
      for I := 1 to 10 do
        Items.Add('Item ' + IntToStr(I));
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.40.5.26 TPageControl.Style Property

Specifies the style of the tab control.

Pascal

```
property Style: TTabStyle;
```

C++

```
__property TTabStyle Style;
```

Description

Use Style to determine the appearance of the tabs. These can appear as notebook tabs or as buttons.

27.1.40.5.27 TPageControl.TabHeight Property

Specifies the height, in pixels, of the tabs in the tab control.

Pascal

```
property TabHeight: Smallint;
```

C++

```
__property Smallint TabHeight;
```

Description

Set TabHeight to control the height of the tabs that appear above the client region of the tab control. The TabHeight property is the vertical size in pixels of the individual tabs. If TabHeight is set to 0, the tabs automatically size themselves to fit their text.

C++ Examples:

```
/*
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
```

```
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
     AnsiString s = PageControl1->Pages[i]->Caption;
     if (Canvas->TextWidth(s) * 2 > PageControl1->TabWidth)
        PageControl1->TabWidth = Canvas->TextWidth(s) * 2;
      if (Canvas->TextHeight(s) * 2 > PageControl1->TabHeight)
        PageControl1->TabHeight = Canvas->TextHeight(s) * 2;
  }
}
```

Delphi Examples:

```
{
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  with PageControl1 do
  begin
    for i := 0 to PageCount - 1 do
    begin
      if (Canvas.TextWidth(Pages[i].Caption) * 2) > TabWidth then
        TabWidth := Canvas.TextWidth(Pages[i].Caption) * 2;
      if (Canvas.TextHeight(Pages[i].Caption) * 2) > TabHeight then
        TabHeight := Canvas.TextHeight(Pages[i].Caption) * 2;
    end;
  end;
end;
```

See Also

DisplayRect (see page 3941)

Tabs (see page 3948)

TabWidth (see page 3950)

27.1.40.5.28 TPageControl.TabIndex Property

Identifies the selected tab on a tab control.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Read TabIndex to determine which tab was selected by the user. TabIndex is the index of the tab in the list of labels maintained by the Tabs property. The first (leftmost) tab has a TabIndex of 0, the next has 1, and so on. If no tabs are selected, TabIndex

has a value of -1.

Set TabIndex to programmatically change the selected tab in the tab control.

C++ Examples:

```
/*
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
*/
void __fastcall TForm1::Add_a_fileClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    int index = TabControl1->Tabs->Add(OpenDialog1->FileName);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[index]);
  }
}

void __fastcall TForm1::Assign_filesClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TabControl1->Tabs->Assign(OpenDialog1->Files);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[TabControl1->TabIndex]);
  }
}
/*
Place the following code in the tab control’s OnChange event
handler:
*/
void __fastcall TForm1::TabControl1Change(TObject *Sender)
{
    Memo1->Lines->LoadFromFile(
      TabControl1->Tabs->Strings[TabControl1->TabIndex]);
}
```

Delphi Examples:

```
{
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
}
procedure TForm1.Add_a_fileClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
```

```
    begin
      index:= TabControl1.Tabs.Add(OpenDialog1.FileName);
      Memo1.Lines.LoadFromFile(TabControl1.Tabs[index]);
    end;
  end;

  procedure TForm1.Assign_filesClick(Sender: TObject);
  var index : integer;
  begin
    OpenDialog1.Options :=
      [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
    if OpenDialog1.Execute then
    begin
      TabControl1.Tabs.Assign(OpenDialog1.Files);
      Memo1.Lines.LoadFromFile(
        TabControl1.Tabs[TabControl1.TabIndex]);
    end;
  end;

  {
  Place the following code in the tab control's OnChange event
  handler:
  }
  procedure TForm1.TabControl1Change(Sender: TObject);
  begin
    with TabControl1 do
      Memo1.Lines.LoadFromFile(Tabs[TabIndex]);
  end;
```

See Also

Tabs (see page 3948)

27.1.40.5.29 TPageControl.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore

cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

27.1.40.5.30 TPageControl.TabPosition Property

Determines whether tabs appear at the top or bottom.

Pascal

```
property TabPosition: TTabPosition;
```

C++

```
__property TTabPosition TabPosition;
```

Description

Set TabPosition to tpTop to display the tabs at the top of the tab control. Set TabPosition to tpBottom to display the tabs at the bottom. If the ScrollOpposite property is true, tabs can appear at both top and bottom, with the row containing the current tab and all subsequent rows on the side indicated by TabPosition.

Due to a limitation in the windows common control, if TabPosition is tpLeft or tpRight, Font must be a True Type font. Otherwise, tab text may paint incorrectly.

See Also

ScrollOpposite (see page 3944)

27.1.40.5.31 TPageControl.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

27.1.40.5.32 TPageControl.TabWidth Property

Specifies the horizontal size, in pixels, of the tabs in the tab control.

Pascal

```
property TabWidth: Smallint;
```

C++

```
__property Smallint TabWidth;
```

Description

TabWidth specifies a fixed width for all tabs. If TabWidth is 0, then each tab is automatically made wide enough to show its caption.

C++ Examples:

```
/*
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
     AnsiString s = PageControl1->Pages[i]->Caption;
     if (Canvas->TextWidth(s) * 2 > PageControl1->TabWidth)
        PageControl1->TabWidth = Canvas->TextWidth(s) * 2;
      if (Canvas->TextHeight(s) * 2 > PageControl1->TabHeight)
        PageControl1->TabHeight = Canvas->TextHeight(s) * 2;
  }
}
```

Delphi Examples:

```
{
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  with PageControl1 do
  begin
    for i := 0 to PageCount - 1 do
    begin
      if (Canvas.TextWidth(Pages[i].Caption) * 2) > TabWidth then
        TabWidth := Canvas.TextWidth(Pages[i].Caption) * 2;
      if (Canvas.TextHeight(Pages[i].Caption) * 2) > TabHeight then
        TabHeight := Canvas.TextHeight(Pages[i].Caption) * 2;
    end;
  end;
end;
```

See Also

TabHeight (see page 3945)

Tabs (see page 3948)

27.1.40.5.33 TPageControl.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;
```

```
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.40.6 TPageControl Events

The events of the TPageControl class are listed here.

Published Events

	Name	Description
	OnChange (see page 4516)	Occurs after a new tab is selected.
	OnChanging (see page 4516)	Occurs immediately before a new tab is selected.
	OnContextPopup (see page 4517)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDockDrop (see page 4518)	Occurs when another control is docked to the control.
	OnDockOver (see page 4520)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 4522)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4524)	Occurs when the user drags an object over a control.
	OnDrawTab (see page 4527)	Occurs when a tab is about to be drawn.
	OnEndDock (see page 4527)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4527)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4528)	Occurs when a control receives the input focus.
	OnExit (see page 4531)	Occurs when the input focus shifts away from one control to another.
	OnGetImageIndex (see page 4532)	Occurs when a tab is about to display its associated image.
	OnGetSiteInfo (see page 4533)	Returns the control's docking information.
	OnMouseActivate (see page 4535)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4536)	Occurs when the user presses a mouse button with the mouse pointer over a control.

	OnMouseEnter (see page 4538)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4538)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4539)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4541)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 4543)	Occurs immediately after the control is resized.
	OnStartDock (see page 4544)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4544)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 4545)	Occurs when the application tries to undock a control that is docked to the windowed control.

27.1.40.6.1 TPageControl.OnChange Event

Occurs after a new tab is selected.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action immediately after the selected tab changes. Use the TabIndex (see page 4509) property to determine which tab is now selected. This is the opportunity to make any changes to the control that reflect the new state implied by the selected tag.

Before the value of TabIndex (see page 4509) changes, an OnChanging (see page 4516) event occurs.

Note: The event is not called if you change the active page in code, for example, by setting the value of TPageControl (see page 4459). ActivePage (see page 4482).

27.1.40.6.2 TPageControl.OnChanging Event

Occurs immediately before a new tab is selected.

Pascal

```
property OnChanging: TTabChangingEvent;
```

C++

```
__property TTabChangingEvent OnChanging;
```

Description

Write an OnChanging event handler to take specific action immediately before the selected tab changes. Set the AllowChange parameter to false to prevent the change.

Use an OnChanging event handler to prevent the user from leaving a tab setting until certain conditions have been met. An OnChanging event handler can also be used to save information about the current state of the tab control before it is changed by a new tab selection.

Note: The event is not called if you change the active page in code, for example, by setting the value of TPageControl. ActivePage.

C++ Examples:

```
/*
This project requires a TPageControl, several TTabSheets (to
add sheets, right click the Page Control and select "New
Page" from the context menu), and a TEdit control on the
first tab sheet.  The user is not allowed to change to a new
page if the active page is "TabSheet1" and there is no text
in the Edit Control.
*/
void __fastcall TForm1::PageControl1Changing(TObject *Sender, bool &AllowChange)
{
  if (((TPageControl *)Sender)->ActivePage == TabSheet1)
    AllowChange = (!Edit1->Text.IsEmpty());
  else
    AllowChange = true;
  if (AllowChange == false)
  {
    MessageBeep(0);
    Edit1->SetFocus();
  }
}
```

Delphi Examples:

```
{
This project requires a TPageControl, several TTabSheets (to
add sheets, right click the Page Control and select "New
Page" from the context menu), and a TEdit control on the
first tab sheet.  The user is not allowed to change to a new
page if the active page is "TabSheet1" and there is no text
in the Edit Control.
}
procedure TForm1.PageControl1Changing(Sender: TObject; var AllowChange: Boolean);
begin
  if ((Sender as TPageControl).ActivePage = TabSheet1) then
    AllowChange := (Edit1.Text <> '')
  else
    AllowChange := True;
end;
```

See Also

OnChange (see page 3952)

TabIndex (see page 3946)

27.1.40.6.3 TPageControl.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property

of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.40.6.4 TPageControl.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop: TDockDropEvent;
```

C++

```
__property TDockDropEvent OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
```

```
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
```

```
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockOver (see page 6216)

27.1.40.6.5 TPageControl.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver: TDockOverEvent;
```

C++

```
__property TDockOverEvent OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
```

```
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
```

```
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnGetSiteInfo (see page 6223)

TDragDockObject (see page 6009)

27.1.40.6.6 TPageControl.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

27.1.40.6.7 TPageControl.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
```

```
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code comes from an application that contains a list box
  and three labels, each with a different font and color.  The
  DragMode property for each of the labels is dmAutomatic.
  The user can select a label and drag it to a list box and
  drop it. When the label is dropped, the items in the list
  box assume the color and font of the dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.40.6.8 TPageControl.OnDrawTab Event

Occurs when a tab is about to be drawn.

Pascal

```
property OnDrawTab: TDrawTabEvent;
```

C++

```
__property TDrawTabEvent OnDrawTab;
```

Description

Use OnDrawTab to customize the painting of tabs.

You can paint the tab using the Canvas property.

See Also

Canvas (see page 3940)

OwnerDraw (see page 3944)

27.1.40.6.9 TPageControl.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.40.6.10 TPageControl.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

27.1.40.6.11 TPageControl.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that

includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;
```

```
procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

27.1.40.6.12 TPageControl.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
```

```
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

27.1.40.6.13 TPageControl.OnGetImageIndex Event

Occurs when a tab is about to display its associated image.

Pascal

```
property OnGetImageIndex: TTabGetImageEvent;
```

C++

```
__property TTabGetImageEvent OnGetImageIndex;
```

Description

Write an OnGetImageIndex event handler to specify the ImageIndex for a given TabIndex.

On entry to the event handler, the ImageIndex parameter is the same as the TabIndex parameter, where TabIndex is the index

of the tab which is about to display its image. Change the ImageIndex parameter to specify the index into the Images property that specifies the image that should be painted.

See Also

Images (see page 3941)

TabIndex (see page 3946)

27.1.40.6.14 TPageControl.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo: TGetSiteInfoEvent;
```

C++

```
__property TGetSiteInfoEvent OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
```

```
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
```

```
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

TDragDockObject (see page 6009)

27.1.40.6.15 TPageControl.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.40.6.16 TPageControl.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}
```

```
// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
```

```
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.40.6.17 TPageControl.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

27.1.40.6.18 TPageControl.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

27.1.40.6.19 TPageControl.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.40.6.20 TPageControl.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
```

```
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
```

```
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

27.1.40.6.21 TPageControl.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

27.1.40.6.22 TPageControl.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

27.1.40.6.23 TPageControl.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it

contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

27.1.40.6.24 TPageControl.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock: TUnDockEvent;
```

C++

```
__property TUnDockEvent OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
```

```
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
```

```
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

27.1.41 TPageScroller Class

TPageScroller defines a display area for a narrow window, such as a toolbar.

Class Hierarchy

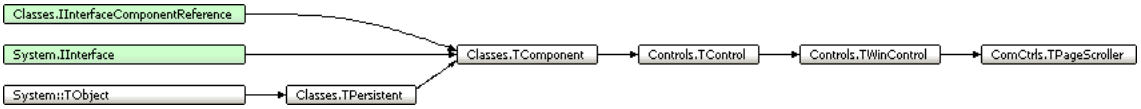

Pascal

```
TPageScroller = class(TWinControl);
```

C++

```
class TPageScroller : public TWinControl;
```

File

ComCtrls

Description

Use TPageScroller to define a display area for a narrow window, such as a toolbar.

If the window is larger than the display area, TPageScroller attaches arrows to the edges of the window, allowing it to be scrolled within the display area.

The page scroller control is similar to the scroll box control. However, the page scroller provides arrows that extend over the edges of the display area, whereas the scroll box provides scroll bars. Also, the page scroller supports scrolling for only one orientation (horizontal or vertical), so the constrained window must fit either the width or height of the display area.

27.1.41.1 TPageScroller Members

The following tables list the members exposed by TPageScroller.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.

	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 4559)	Aligns controls in the contained window.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.

	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 4559)	Sets the page scroller's window parameters.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 4560)	Creates the page scroller's window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.

	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 4560)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.

	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Scroll (see page 4561)	Called immediately before the page scroller scrolls.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.

	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.

	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 4558)	Creates and initializes an instance of TPageScroller (see page 4548).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetButtonState (see page 4560)	Returns the state of one of the page scroller's buttons.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.

R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 4585)	Occurs when the user clicks the control.
	OnContextPopup (see page 4587)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 4588)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 4588)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4590)	Occurs when the user drags an object over a control.
	OnEndDock (see page 4593)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4593)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4594)	Occurs when a control receives the input focus.
	OnExit (see page 4597)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 4598)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 4601)	Occurs when a key is pressed.
	OnKeyUp (see page 4602)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 4604)	Occurs when the mouse wheel is rotated.
	OnResize (see page 4604)	Occurs immediately after the control is resized.
	OnScroll (see page 4605)	Occurs immediately before the page scroller scrolls.
	OnStartDock (see page 4605)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4605)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 4562)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 4565)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 4566)	Specifies whether the page scroller scrolls when the mouse hovers over one of its arrows.
	BorderWidth (see page 4566)	Specifies the width of the control's border.
	ButtonSize (see page 4567)	Specifies the size of the page scroller's arrows.
	Color (see page 4567)	Specifies the background color of the control.
	Constraints (see page 4568)	Specifies the size constraints for the control.
	Control (see page 4568)	Specifies the control contained in the page scroller.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DockSite (see page 4568)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 4569)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4570)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4570)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DragScroll (see page 4572)	Specifies whether the contained window can be a drag-and-drop target.
	Enabled (see page 4572)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4572)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margin (see page 4574)	Specifies the margin between the contained window and the inside edges of the page scroller.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Orientation (see page 4574)	Specifies whether the page scroller is horizontal or vertical.
	ParentBackground (see page 4574)	Determines whether control uses parent's theme background.
	ParentColor (see page 4574)	Determines where a control looks for its color information.
	ParentFont (see page 4575)	Determines where a control looks for its font information.
	ParentShowHint (see page 4575)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4576)	Identifies the pop-up menu associated with the control.
	Position (see page 4580)	Specifies the scroll position of the page scroller.
	ShowHint (see page 4581)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 4582)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 4583)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 4583)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.41.2 TPageScroller.Create Constructor

Creates and initializes an instance of TPageScroller (see page 4548).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TPageScroller(TComponent * AOwner);
```

Description

Create constructs an instance of TPageScroller (see page 4548).

27.1.41.3 TPageScroller Methods

The methods of the TPageScroller class are listed here.

Protected Methods

	Name	Description
	AlignControls (see page 4559)	Aligns controls in the contained window.
	CreateParams (see page 4559)	Sets the page scroller's window parameters.
	CreateWnd (see page 4560)	Creates the page scroller's window.
	Notification (see page 4560)	Responds to notifications that components are being created or destroyed.
	Scroll (see page 4561)	Called immediately before the page scroller scrolls.

Public Methods

	Name	Description
	GetButtonState (see page 4560)	Returns the state of one of the page scroller's buttons.

27.1.41.3.1 TPageScroller.AlignControls Method

Aligns controls in the contained window.

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

Call AlignControls to align the controls in the page scroller's contained window according to values of their Align (see page 4562) properties.

Overrides the inherited AlignControls property in order to take the orientation of the page scroller into account.

27.1.41.3.2 TPageScroller.CreateParams Method

Sets the page scroller's window parameters.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

CreateParams is called automatically when the page scroller window is created.

CreateParams overrides the inherited CreateParams method in order to set window styles specific to the page scroller control.

27.1.41.3.3 TPageScroller.CreateWnd Method

Creates the page scroller's window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

Call CreateWnd to create the page scroller's window.

CreateWnd overrides the inherited CreateWnd method in order to initialize the page scroller's color, handle, button size, margin, and position.

27.1.41.3.4 TPageScroller.GetButtonState Method

Returns the state of one of the page scroller's buttons.

Pascal

```
function GetButtonState(Button: TPageScrollerButton): TPageScrollerButtonState;
```

C++

```
__fastcall TPageScrollerButtonState GetButtonState(TPageScrollerButton Button);
```

Description

Call GetButtonState to get the state of one of the page scroller's buttons.

Button is the button for which to get the state information.

GetButtonState returns the button's state.

27.1.41.3.5 TPageScroller.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

27.1.41.3.6 TPageScroller.Scroll Method

Called immediately before the page scroller scrolls.

Pascal

```
procedure Scroll(Shift: TShiftState; X: Integer; Y: Integer; Orientation:
TPageScrollerOrientation; var Delta: Integer); dynamic;
```

C++

```
__fastcall Scroll(TShiftState Shift, int X, int Y, TPageScrollerOrientation Orientation, int
Delta);
```

Description

Override the Scroll method in a derived class to handle the scrolling event.

By default, Scroll calls the OnScroll (see page 4605) event handler.

Shift is the state of the Shift, Alt, Ctrl keys and mouse buttons.

X is the horizontal position of the page scroller.

Y is the vertical position of the page scroller.

Orientation (see page 4574) is the direction for the scroll.

Delta is the amount of the scroll.

27.1.41.4 TPageScroller Properties

The properties of the TPageScroller class are listed here.

Published Properties

	Name	Description
	Align (see page 4562)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4565)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 4566)	Specifies whether the page scroller scrolls when the mouse hovers over one of its arrows.
	BorderWidth (see page 4566)	Specifies the width of the control's border.
	ButtonSize (see page 4567)	Specifies the size of the page scroller's arrows.
	Color (see page 4567)	Specifies the background color of the control.
	Constraints (see page 4568)	Specifies the size constraints for the control.
	Control (see page 4568)	Specifies the control contained in the page scroller.
	DockSite (see page 4568)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 4569)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4570)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4570)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DragScroll (see page 4572)	Specifies whether the contained window can be a drag-and-drop target.
	Enabled (see page 4572)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4572)	Controls the attributes of text written on or in the control.

	Margin (see page 4574)	Specifies the margin between the contained window and the inside edges of the page scroller.
	Orientation (see page 4574)	Specifies whether the page scroller is horizontal or vertical.
	ParentBackground (see page 4574)	Determines whether control uses parent's theme background.
	ParentColor (see page 4574)	Determines where a control looks for its color information.
	ParentFont (see page 4575)	Determines where a control looks for its font information.
	ParentShowHint (see page 4575)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4576)	Identifies the pop-up menu associated with the control.
	Position (see page 4580)	Specifies the scroll position of the page scroller.
	ShowHint (see page 4581)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 4582)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 4583)	Determines if the user can tab to a control.
	Visible (see page 4583)	Determines whether the component appears onscreen.

27.1.41.4.1 TPageScroller.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
```

```
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
```

```
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
```

```
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

27.1.41.4.2 TPageScroller.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.41.4.3 TPageScroller.AutoScroll Property

Specifies whether the page scroller scrolls when the mouse hovers over one of its arrows.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

Use AutoScroll to specify whether the page scroller scrolls when the mouse hovers over one of its arrows.

27.1.41.4.4 TPageScroller.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics or text drawn by the control is clipped to the area within the border.

See Also

ClientRect

BevelOuter (see page 6194)

27.1.41.4.5 TPageScroller.ButtonSize Property

Specifies the size of the page scroller's arrows.

Pascal

```
property ButtonSize: Integer;
```

C++

```
__property int ButtonSize;
```

Description

Use ButtonSize to specify the size of the page scroller's arrows.

27.1.41.4.6 TPageScroller.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
```

```
end;
```

See Also

ParentColor (see page 5834)

27.1.41.4.7 TPageScroller.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.41.4.8 TPageScroller.Control Property

Specifies the control contained in the page scroller.

Pascal

```
property Control: TWinControl;
```

C++

```
__property TWinControl * Control;
```

Description

Use Control to specify the control contained in the page scroller.

27.1.41.4.9 TPageScroller.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite: Boolean;
```

C++

```
__property Boolean DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

27.1.41.4.10 TPageScroller.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

27.1.41.4.11 TPageScroller.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

27.1.41.4.12 TPageScroller.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
```

```
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

27.1.41.4.13 TPageScroller.DragScroll Property

Specifies whether the contained window can be a drag-and-drop target.

Pascal

```
property DragScroll: Boolean;
```

C++

```
__property Boolean DragScroll;
```

Description

Use DragScroll to specify whether the contained window can be a drag-and-drop target. If DragScroll is true, the page scroller will scroll when an item is dragged over the page scroller's arrows.

27.1.41.4.14 TPageScroller.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.41.4.15 TPageScroller.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

27.1.41.4.16 TPageScroller.Margin Property

Specifies the margin between the contained window and the inside edges of the page scroller.

Pascal

```
property Margin: Integer;
```

C++

```
__property int Margin;
```

Description

Use Margin to specify the margin between the contained window and the inside edges of the page scroller.

27.1.41.4.17 TPageScroller.Orientation Property

Specifies whether the page scroller is horizontal or vertical.

Pascal

```
property Orientation: TPageScrollerOrientation;
```

C++

```
__property TPageScrollerOrientation Orientation;
```

Description

Use Orientation to specify whether the page scroller is horizontal or vertical.

27.1.41.4.18 TPageScroller.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground: Boolean;
```

C++

```
__property Boolean ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

27.1.41.4.19 TPageScroller.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

27.1.41.4.20 TPageScroller.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

27.1.41.4.21 TPageScroller.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.41.4.22 TPageScroller.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
```

```
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
```

```
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
```

```
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
```

```
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.41.4.23 TPageScroller.Position Property

Specifies the scroll position of the page scroller.

Pascal

```
property Position: Integer;
```

C++

```
__property int Position;
```

Description

Use Position to specify the scroll position of the page scroller.

27.1.41.4.24 TPageScroller.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.41.4.25 TPageScroller.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the

control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

27.1.41.4.26 TPageScroller.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

27.1.41.4.27 TPageScroller.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
```

```
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.41.5 TPageScroller Events

The events of the TPageScroller class are listed here.

Published Events

	Name	Description
	OnClick (see page 4585)	Occurs when the user clicks the control.
	OnContextPopup (see page 4587)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 4588)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 4588)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4590)	Occurs when the user drags an object over a control.
	OnEndDock (see page 4593)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4593)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4594)	Occurs when a control receives the input focus.
	OnExit (see page 4597)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 4598)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 4601)	Occurs when a key is pressed.
	OnKeyUp (see page 4602)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 4604)	Occurs when the mouse wheel is rotated.
	OnResize (see page 4604)	Occurs immediately after the control is resized.
	OnScroll (see page 4605)	Occurs immediately before the page scroller scrolls.
	OnStartDock (see page 4605)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4605)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

27.1.41.5.1 TPageScroller.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).

- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
```

```
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

27.1.41.5.2 TPageScroller.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.41.5.3 TPageScroller.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

27.1.41.5.4 TPageScroller.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}
```

```
/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
```

```
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

27.1.41.5.5 TPageScroller.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;
```

```
procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.41.5.6 TPageScroller.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.41.5.7 TPageScroller.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
```

```
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

27.1.41.5.8 TPageScroller.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
```

```
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

27.1.41.5.9 TPageScroller.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

27.1.41.5.10 TPageScroller.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the

OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
```

```
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

27.1.41.5.11 TPageScroller.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
```

```
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

27.1.41.5.12 TPageScroller.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
```

```
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

27.1.41.5.13 TPageScroller.OnMouseWheel Event

Occurs when the mouse wheel is rotated.

Pascal

```
property OnMouseWheel: TMouseWheelEvent;
```

C++

```
__property TMouseWheelEvent OnMouseWheel;
```

Description

Write code in the OnMouseWheel event handler to respond to mouse wheel messages. If there is no OnMouseWheel event handler, or if the mouse wheel message is not handled in the OnMouseWheel event handler, then an OnMouseWheelUp or OnMouseWheelDown event occurs, depending on the direction the mouse wheel was rotated.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheelUp (see page 5870)

27.1.41.5.14 TPageScroller.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

27.1.41.5.15 TPageScroller.OnScroll Event

Occurs immediately before the page scroller scrolls.

Pascal

```
property OnScroll: TPageScrollEvent;
```

C++

```
__property TPageScrollEvent OnScroll;
```

Description

Write code in the OnScroll event handler to perform actions when the page scroller scrolls.

27.1.41.5.16 TPageScroller.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

27.1.41.5.17 TPageScroller.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse

button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

27.1.42 TParaAttributes Class

TParaAttributes represents paragraph formatting properties for a rich edit control.

Class Hierarchy

Pascal

```
TParaAttributes = class(TPersistent);
```

C++

```
class TParaAttributes : public TPersistent;
```

File

ComCtrls

Description

TParaAttributes includes a number of properties that describe the formatting characteristics of a paragraph in a rich text document. TParaAttributes is tailored to allow a descendant of TCustomRichEdit (see page 3854) to read and write formatting

characteristics of selected paragraphs. The properties of a TParaAttributes object are invalid unless it is owned by a descendant of TCustomRichEdit (see page 3854).

See Also

TCustomRichEdit (see page 3854)

27.1.42.1 TParaAttributes Members

The following tables list the members exposed by TParaAttributes.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 4608)	Copies the properties specified in another TParaAttributes (see page 4606) object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 4608)	Creates an instance of TParaAttributes (see page 4606).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Alignment (see page 4609)	Specifies how text is aligned within a paragraph.
	FirstIndent (see page 4610)	Specifies the indent, in pixels, of the first line of the paragraph relative to the left margin.
	LeftIndent (see page 4612)	Specifies the indent, in pixels, of the paragraph relative to the left margin.
	Numbering (see page 4612)	Specifies whether the paragraph is one of a set of bulleted paragraphs.
	RightIndent (see page 4614)	Specifies the indent, in pixels, of the paragraph relative to the right margin.
	Tab (see page 4614)	Specifies the position, in points, of a particular tab stop.
	TabCount (see page 4615)	Indicates the number of tab stops that are defined for the paragraph.

27.1.42.2 TParaAttributes.Create Constructor

Creates an instance of TParaAttributes (see page 4606).

Pascal

```
constructor Create(AOwner: TCustomRichEdit);
```

C++

```
__fastcall TParaAttributes(TCustomRichEdit AOwner);
```

Description

Do not call the constructor for a TParaAttributes (see page 4606) object, as the properties of a TParaAttributes (see page 4606) object are not valid outside of a TCustomRichEdit (see page 3854) object. Create is called from the constructor of a rich edit control that uses TParaAttributes (see page 4606).

See Also

Create (see page 22287)

27.1.42.3 TParaAttributes Methods

The methods of the TParaAttributes class are listed here.

Public Methods

	Name	Description
	Assign (see page 4608)	Copies the properties specified in another TParaAttributes (see page 4606) object.

27.1.42.3.1 TParaAttributes.Assign Method

Copies the properties specified in another TParaAttributes (see page 4606) object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use Assign to set the paragraph attributes of the selected paragraph from one rich edit control to match the attributes of the selected paragraph of another rich edit control. Since the properties of a TParaAttributes (see page 4606) object are only valid when the TParaAttributes (see page 4606) exists within a rich edit control, Assign is only useful for copying attributes between

rich edit controls.

Note: Assign only succeeds at runtime when the Source is a TParaAttributes (see page 4606) object or an object that has an AssignTo method that is enabled for a TParaAttributes (see page 4606) object. Other source objects will raise an EConvertError exception.

See Also

EConvertError (see page 22724)

Alignment (see page 4609)

FirstIndent (see page 4610)

LeftIndent (see page 4612)

Numbering (see page 4612)

RightIndent (see page 4614)

Tab (see page 4614)

27.1.42.4 TParaAttributes Properties

The properties of the TParaAttributes class are listed here.

Public Properties

	Name	Description
	Alignment (see page 4609)	Specifies how text is aligned within a paragraph.
	FirstIndent (see page 4610)	Specifies the indent, in pixels, of the first line of the paragraph relative to the left margin.
	LeftIndent (see page 4612)	Specifies the indent, in pixels, of the paragraph relative to the left margin.
	Numbering (see page 4612)	Specifies whether the paragraph is one of a set of bulleted paragraphs.
	RightIndent (see page 4614)	Specifies the indent, in pixels, of the paragraph relative to the right margin.
	Tab (see page 4614)	Specifies the position, in points, of a particular tab stop.
	TabCount (see page 4615)	Indicates the number of tab stops that are defined for the paragraph.

27.1.42.4.1 TParaAttributes.Alignment Property

Specifies how text is aligned within a paragraph.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to specify the alignment of text within individual paragraphs. Unlike the Alignment property of a TCustomMemo, which applies to the entire body of text, the Alignment property of TParaAttributes (see page 4606) applies only to the selected paragraphs.

These are the possible values:

Value	Meaning
taLeftJustify	Align text to the left side of the paragraph.
taCenter	Center text horizontally in the paragraph.

taRightJustify	Align text to the right side of the paragraph.

C++ Examples:

```
/*
To use this example, place a Toolbar and a RichEdit on the
form. Right click the toolbar and select New Button three
times.  Place the ToolButton1Click handling code in the on
click event of the first TToolButton, then attach the same
OnClick handler for each button.  Double click on the Image-
List to load the images.  Load in the order right, center
and left.  Name the Imagelist ToolbarImages.  Select the
Toolbar and set the Images property to ToolbarImages.  Then
select each ToolButton and set the ImageIndex property.
*/
void __fastcall TForm1::ToolButton1Click(TObject *Sender)
{
  if (Sender == ToolButton1)
    RichEdit1->Paragraph->Alignment = taLeftJustify;
  else if (Sender == ToolButton2)
    RichEdit1->Paragraph->Alignment = taCenter;
  else if (Sender == ToolButton3)
    RichEdit1->Paragraph->Alignment = taRightJustify;
}
```

Delphi Examples:

```
{
To use this example, place a Toolbar and a RichEdit on the
form. Right click the toolbar and select New Button three
times.  Place the ToolButton1Click handling code in the on
click event of the first TToolButton, then attach the same
OnClick handler for each button.  Double click on the Image-
List to load the images.  Load in the order right, center
and left.  Name the Imagelist ToolbarImages.  Select the
Toolbar and set the Images property to ToolbarImages.  Then
select each ToolButton and set the ImageIndex property.
}
procedure TForm1.ToolButton1Click(Sender: TObject);
begin
   if (Sender = ToolButton1) then
    RichEdit1.Paragraph.Alignment := taLeftJustify
  else if (Sender = Toolbutton2) then
    RichEdit1.Paragraph.Alignment := taCenter
  else if (Sender = Toolbutton3) then
    RichEdit1.Paragraph.Alignment := taRightJustify;

end;
```

See Also

LeftIndent (see page 4612)

RightIndent (see page 4614)

Alignment (see page 21379)

27.1.42.4.2 TParaAttributes.FirstIndent Property

Specifies the indent, in pixels, of the first line of the paragraph relative to the left margin.

Pascal

```
property FirstIndent: Longint;
```

C++

```
__property Longint FirstIndent;
```

Description

Use FirstIndent to specify the number of pixels to indent the first line of the paragraph. This value is relative to the value of LeftIndent (see page 4612).

C++ Examples:

```
/*
This example requires only a form.  A Rich Edit control is
created on the form and aligned to take the entire client
area of the form.  A series of lines are written to the Rich
Edit control's Lines property, and the Paragraph property
manipulated such that the first items displayed are bulleted,
a second group of lines are unbulleted and centered, and the
last group is left justified and indented.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TRichEdit *pRich = new TRichEdit(this);
  pRich->Parent = this;
  pRich->Align = alClient;
  pRich->Lines->Clear();
  // set numbering style
  pRich->Paragraph->Numbering = nsBullet;
  pRich->Lines->Add("Introduction");
  pRich->Lines->Add("New members to our team");
  pRich->Lines->Add("New Budget discussion");
  pRich->Lines->Add("Facilities");
  pRich->Lines->Add("Q & A");
  pRich->Paragraph->Numbering = nsNone;
  pRich->Paragraph->Alignment = taCenter;
  pRich->Lines->Add("");
  pRich->Lines->Add("Suggested Topics: ");
  pRich->Lines->Add("");
  pRich->Paragraph->Alignment = taLeftJustify;
  pRich->Paragraph->FirstIndent = 10;
  pRich->Lines->Add("");
  pRich->Lines->Add("Parking lot repair");
  pRich->Lines->Add("Cost overruns");
}
```

Delphi Examples:

```
{
This example requires only a form.  A Rich Edit control is
created on the form and aligned to take the entire client
area of the form.  A series of lines are written to the Rich
Edit control's Lines property, and the Paragraph property
manipulated such that the first items displayed are bulleted,
a second group of lines are unbulleted and centered, and the
last group is left justified and indented.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with TRichEdit.Create(Self) do
  begin
    Parent := Self;
    Align := alClient;
    Lines.Clear;
    // set numbering style
    Paragraph.Numbering := nsBullet;
```

```
      Lines.Add('Introduction');
      Lines.Add('New members to our team');
      Lines.Add('New Budget discussion');
      Lines.Add('Facilities');
      Lines.Add('Q & A');
      Paragraph.Numbering := nsNone;
      Paragraph.Alignment := taCenter;
      Lines.Add('');
      Lines.Add('Suggested Topics:');
      Lines.Add('');
      Paragraph.Alignment := taLeftJustify;
      Paragraph.FirstIndent := 10;
      Lines.Add('');
      Lines.Add('Parking lot repair');
      Lines.Add('Cost overruns');
    end;
  end;
```

See Also

LeftIndent (see page 4612)

Tab (see page 4614)

27.1.42.4.3 TParaAttributes.LeftIndent Property

Specifies the indent, in pixels, of the paragraph relative to the left margin.

Pascal

```
property LeftIndent: Longint;
```

C++

```
__property Longint LeftIndent;
```

Description

Use LeftIndent to indent a paragraph from the left margin of the entire body of text. LeftIndent is useful for making individual paragraphs stand out from the rest of the text.

Note: Adding bullets to a set of paragraphs by using the Numbering (see page 4612) property also affects the left indentation.

See Also

FirstIndent (see page 4610)

Numbering (see page 4612)

RightIndent (see page 4614)

27.1.42.4.4 TParaAttributes.Numbering Property

Specifies whether the paragraph is one of a set of bulleted paragraphs.

Pascal

```
property Numbering: TNumberingStyle;
```

C++

```
__property TNumberingStyle Numbering;
```

Description

To insert a bullet to the left of the paragraph, and indent the paragraph sufficiently to accommodate the bullet, set Numbering to nsBullet. If the LeftIndent (see page 4612) property is set, bullets will appear at the indentation indicated by LeftIndent (see

page 4612), and the body of the paragraph will be further indented to accommodate the bullets.

These are the possible values:

Value	Meaning
nsNone	No numbering on the left.
nsBullet	Bullets on the left with sufficient indentation for the bullets.

C++ Examples:

```
/*
This example requires only a form.  A Rich Edit control is
created on the form and aligned to take the entire client
area of the form.  A series of lines are written to the Rich
Edit control's Lines property, and the Paragraph property
manipulated such that the first items displayed are bulleted,
a second group of lines are unbulleted and centered, and the
last group is left justified and indented.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TRichEdit *pRich = new TRichEdit(this);
  pRich->Parent = this;
  pRich->Align = alClient;
  pRich->Lines->Clear();
  // set numbering style
  pRich->Paragraph->Numbering = nsBullet;
  pRich->Lines->Add("Introduction");
  pRich->Lines->Add("New members to our team");
  pRich->Lines->Add("New Budget discussion");
  pRich->Lines->Add("Facilities");
  pRich->Lines->Add("Q & A");
  pRich->Paragraph->Numbering = nsNone;
  pRich->Paragraph->Alignment = taCenter;
  pRich->Lines->Add("");
  pRich->Lines->Add("Suggested Topics: ");
  pRich->Lines->Add("");
  pRich->Paragraph->Alignment = taLeftJustify;
  pRich->Paragraph->FirstIndent = 10;
  pRich->Lines->Add("");
  pRich->Lines->Add("Parking lot repair");
  pRich->Lines->Add("Cost overruns");
}
```

Delphi Examples:

```
{
This example requires only a form.  A Rich Edit control is
created on the form and aligned to take the entire client
area of the form.  A series of lines are written to the Rich
Edit control's Lines property, and the Paragraph property
manipulated such that the first items displayed are bulleted,
a second group of lines are unbulleted and centered, and the
last group is left justified and indented.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with TRichEdit.Create(Self) do
  begin
    Parent := Self;
    Align := alClient;
    Lines.Clear;
    // set numbering style
```

```
      Paragraph.Numbering := nsBullet;
      Lines.Add('Introduction');
      Lines.Add('New members to our team');
      Lines.Add('New Budget discussion');
      Lines.Add('Facilities');
      Lines.Add('Q & A');
      Paragraph.Numbering := nsNone;
      Paragraph.Alignment := taCenter;
      Lines.Add('');
      Lines.Add('Suggested Topics:');
      Lines.Add('');
      Paragraph.Alignment := taLeftJustify;
      Paragraph.FirstIndent := 10;
      Lines.Add('');
      Lines.Add('Parking lot repair');
      Lines.Add('Cost overruns');
    end;
  end;
```

See Also

LeftIndent (see page 4612)

27.1.42.4.5 TParaAttributes.RightIndent Property

Specifies the indent, in pixels, of the paragraph relative to the right margin.

Pascal

```
property RightIndent: Longint;
```

C++

```
__property Longint RightIndent;
```

Description

Use RightIndent to indent a paragraph from the right margin of the entire body of text. RightIndent is useful for making individual paragraphs stand out from the rest of the text, especially when Alignment (see page 4609) is set to taRightJustify.

See Also

Alignment (see page 4609)

LeftIndent (see page 4612)

27.1.42.4.6 TParaAttributes.Tab Property

Specifies the position, in points, of a particular tab stop.

Pascal

```
property Tab [Index: Byte]: Longint;
```

C++

```
__property Longint Tab[Byte Index];
```

Description

Tab is an indexed property into an array of absolute tab stop positions. The position of the first tab stop is Tab[0], the next tab stop is Tab[1], and so on to Tab[TabCount (see page 4615) - 1].

When setting Tab, values of Index below TabCount (see page 4615) change the positions of existing tab stops. Values of Index greater than or equal to TabCount (see page 4615) increase the number of tab stops in the paragraph and change the value of TabCount (see page 4615).

Note: Setting Tab does not insert a tab into the text of the paragraph. It sets the positions within the paragraph where tabs will occur if they are inserted. The actual tab characters are inserted into the paragraph by using the Tab key.

See Also

TabCount (see page 4615)

27.1.42.4.7 TParaAttributes.TabCount Property

Indicates the number of tab stops that are defined for the paragraph.

Pascal

```
property TabCount: Integer;
```

C++

```
__property int TabCount;
```

Description

Read TabCount to determine the number of tab stops in the Tab (see page 4614) array. Use TabCount with the Tab (see page 4614) property to iterate over all the tab stops defined for the paragraph.

See Also

Tab (see page 4614)

27.1.43 TProgressBar Class

TProgressBar displays a simple progress bar.

Class Hierarchy

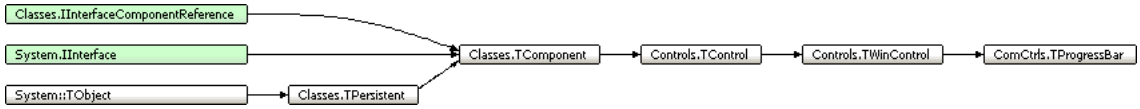

Pascal

```
TProgressBar = class(TWinControl);
```

C++

```
class TProgressBar : public TWinControl;
```

File

ComCtrls

Description

Use TProgressBar to add a progress bar to a form. Progress bars provide users with visual feedback about the progress of a procedure within an application. As the procedure progresses, the rectangular progress bar gradually fills from left to right with the system highlight color.

See Also

Progress Bars

Display Controls

27.1.43.1 TProgressBar Members

The following tables list the members exposed by TProgressBar.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.

	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 4627)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 4628)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 4628)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus

	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.

	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 4626)	Creates an instance of TProgressBar (see page 4615).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.

	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetDesignVisible (see page 6178)	Makes a control visible at design time.
	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	StepBy (see page 4629)	Advances the Position (see page 4655) of the progress bar by a specified amount.
	StepIt (see page 4631)	Advances Position (see page 4655) by the amount specified in the Step (see page 4660) property.
	Update (see page 5789)	Processes any pending paint messages immediately.
	Update (see page 6183)	Forces the control to update.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.

	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnContextPopup (see page 4665)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 4665)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4668)	Occurs when the user drags an object over a control.
	OnEndDock (see page 4670)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4670)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4671)	Occurs when a control receives the input focus.
	OnExit (see page 4674)	Occurs when the input focus shifts away from one control to another.
	OnMouseActivate (see page 4675)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4676)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4679)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4679)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4679)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4681)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 4684)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4684)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 4633)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 4637)	Specifies how the control is anchored to its parent.
	BorderWidth (see page 4637)	Specifies the width of the control's border.
	Constraints (see page 4637)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 4638)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4638)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4639)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4640)	Controls whether the control responds to mouse, keyboard, and timer events.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 4641)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.

	Margins (see page 5831)	Specifies the margins for the control.
	Max (see page 4643)	Specifies the upper limit of the range of possible positions.
	Min (see page 4646)	Specifies the lower limit of the range of possible positions.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Orientation (see page 4649)	Specifies whether the progress bar is oriented vertically or horizontally.
	ParentShowHint (see page 4649)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4650)	Identifies the pop-up menu associated with the control.
	Position (see page 4655)	Specifies the current position of the progress bar.
	ShowHint (see page 4658)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Smooth (see page 4660)	Specifies whether the progress bar is smooth or segmented.
	Step (see page 4660)	Specifies the amount that Position (see page 4655) increases when the StepIt (see page 4631) method is called.
	TabOrder (see page 4662)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 4662)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 4663)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.43.2 TProgressBar.Create Constructor

Creates an instance of TProgressBar (see page 4615).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TProgressBar(TComponent * AOwner);
```

Description

Call Create to create a progress bar at runtime. Progress bars placed on forms at design time are created automatically.

AOwner is another component, typically the form, that is responsible for freeing the progress bar. It becomes the value of the Owner property.

C++ Examples:

```
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
```

Delphi Examples:

```
{
Dynamically create a TProgressBar control and align it to
```

```
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
```

See Also

Height (see page 5825)

Max (see page 4643)

Min (see page 4646)

Step (see page 4660)

Width (see page 5849)

27.1.43.3 TProgressBar Methods

The methods of the TProgressBar class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 4627)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 4628)	Creates the underlying window.
	DestroyWnd (see page 4628)	Destroys the control's window.

Public Methods

	Name	Description
	StepBy (see page 4629)	Advances the Position (see page 4655) of the progress bar by a specified amount.
	StepIt (see page 4631)	Advances Position (see page 4655) by the amount specified in the Step (see page 4660) property.

27.1.43.3.1 TProgressBar.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call

the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

27.1.43.3.2 TProgressBar.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams (see page 6135)

CreateWindowHandle (see page 6137)

Perform (see page 5776)

DestroyWnd (see page 6141)

27.1.43.3.3 TProgressBar.DestroyWnd Method

Destroys the control's window.

Pascal

```
procedure DestroyWnd; override;
```

C++

```
virtual __fastcall DestroyWnd();
```

Description

DestroyWnd is called when the control needs to replace its window. For example, changing properties that are implemented by window parameter settings require the control to be destroyed and then recreated using the CreateWnd method.

Before the window is destroyed, DestroyWnd saves a copy of the control's text in memory, frees any device contexts, and finally calls DestroyWindowHandle. When the window is recreated, the stored copy of the control's text is assigned to the new window.

See Also

DestroyWindowHandle (see page 6140)

RecreateWnd (see page 6171)

27.1.43.3.4 TProgressBar.StepBy Method

Advances the Position (see page 4655) of the progress bar by a specified amount.

Pascal

```
procedure StepBy(Delta: Integer);
```

C++

```
__fastcall StepBy(int Delta);
```

Description

Call StepBy to increase the value of Position (see page 4655) by the value of the Delta parameter. To advance Position (see page 4655) by a default amount that represents a single step in the process, use the StepIt (see page 4631) method.

C++ Examples:

```
/*
StepBy, CreateForm, Run, Show, Update, Initialize example
This example shows progress of loading forms as an application starts up. The code example is
placed in the project source (*.dpr) file. To see project source, select the View | Project
Source menu item.
You will need to set up your project with the following steps before using the code example:
o          Add four additional forms to a default project.
o          Place a TProgressBar on Form5
o          Take the Project|Options|Forms menu option and place Form5 on the available forms
list.
o          Change the code of your project (*.dpr) file to look like the example.
*/
#include <vcl\vcl.h>
#include "Unit5.h"
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("TAppCreateForm.cpp", Form1);
USERES("Project1.res");
USEFORM("Unit2.cpp", Form2);
USEFORM("Unit3.cpp", Form3);
USEFORM("Unit4.cpp", Form4);
USEFORM("Unit5.cpp", Form5);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
  try
  {
    Application->Initialize();
    SetApplicationMainFormOnTaskBar(Application, true);
    Application->CreateForm(__classid(TForm5), &Form5);
    Form5->ProgressBar1->Max = 100;
    Form5->Show();   // show a splash screen contain ProgressBar control
    Form5->Update(); // force display of Form5
    Application->CreateForm(__classid(TForm1), &Form1);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form1 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm2), &Form2);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form2 loaded successfully.";
```

```
      Form5->Update(); // force display of Form5
      Sleep(3000);
      Application->CreateForm(__classid(TForm3), &Form3);
      Form5->ProgressBar1->StepBy(25);
      Form5->Label1->Caption =  "Form3 loaded successfully.";
      Form5->Update(); // force display of Form5
      Sleep(3000);
      Application->CreateForm(__classid(TForm4), &Form4);
      Form5->ProgressBar1->StepBy(25);
      Form5->Label1->Caption =  "Form4 loaded successfully.";
      Form5->Update(); // force display of Form5
      Sleep(3000);
//    delete Form5;
      Application->Run();
    }
    catch (Exception &exception)
    {
      Application->ShowException(&exception);
    }
    return 0;
  }
```

Delphi Examples:

```
{
This example shows progress of loading forms as an
application starts up. The code example is placed in the
project source (*.dpr) file.  To see project source, select
the View | Project Source menu item.
You will need to set up your project with the following steps
before using the code example:
    Add four additional forms to a default project.
    Place a TProgressBar on Form5
    Take the Project|Options|Forms menu option and place
          Form5 on the available forms list.
    Change the code of your project (*.dpr) file to look
          like the example.
}
begin
  Application.Initialize;
  with TForm5.Create(nil) do
  try
    Application.MainFormOnTaskbar := True;
    ProgressBar1.Max := 100;
    Show;  // show a splash screen contain ProgressBar control
    Update; // force display of Form5
    Application.CreateForm(TForm1, Form1);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form1 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm2, Form2);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form2 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm3, Form3);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form3 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm4, Form4);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form4 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
  finally
```

```
    Free;
  end;
  Application.Run;
```

See Also

Position (see page 4655)

StepIt (see page 4631)

27.1.43.3.5 TProgressBar.StepIt Method

Advances Position (see page 4655) by the amount specified in the Step (see page 4660) property.

Pascal

```
procedure StepIt;
```

C++

```
__fastcall StepIt();
```

Description

Call the StepIt method to increase the value of Position (see page 4655) by the value of the Step (see page 4660) property. If Step (see page 4660) represents the size of one logical step in the process tracked by the progress bar, call Step (see page 4660) after each logical step is completed.

C++ Examples:

```
/*
This example reads bytes from a file in tenths of the size
of the file. The ProgressBar displays the status of each
1/10th step as it's read into the buffer.  This example
requires a TButton and a TProgressBar control on the form.
*/
#include <stdio.h>     // for FILE, fopen, fstat, fileno, fread and fclose
#include <sys\stat.h>  // for fstat and the stat struct
#include <Math.hpp>    // for Min

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FILE *F;
  char MyData[2047];
  long BytesRead;

  F = fopen("../about.txt", "r");   // path relative to Debug
                                    // use a file larger than 2048 bytes to make it
interesting.
  if (F)
  {
    struct stat statbuf;
    fstat(fileno(F), &statbuf);
    ProgressBar1->Max = statbuf.st_size;
    if (ProgressBar1->Max > 10)
    {
      ProgressBar1->Step = (ProgressBar1->Max)/10;
      ProgressBar1->Step = Min(ProgressBar1->Step, 2047);
    }
    else
      ProgressBar1->Step = ProgressBar1->Max;
    char *DataBuffer = new char[ProgressBar1->Step];
    for (ProgressBar1->Position = 0;
         ProgressBar1->Position < ProgressBar1->Max;
         ProgressBar1->StepIt())  // move the ProgressBar Position using StepIt
    {
      fread(DataBuffer, ProgressBar1->Step, 1, F);
```

```
        // Do this or the read will wrap and start over!
        ProgressBar1->Step =
          Min(ProgressBar1->Step, ProgressBar1->Max - ProgressBar1->Position);
      }
      delete DataBuffer;
      fclose(F);
    }
  }
```

Delphi Examples:

```
{
This example reads bytes from a file in tenths of the size
of the file. The ProgressBar displays the status of each
1/10th step as it's read into the buffer.  This example
requires a TButton and a TProgressBar control on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
const
  FName = 'about.txt';  // use a file larger than 2048 bytes to make it interesting.
var
  F: File;
  MyData: array[1..2048] of byte;
  BytesRead: LongInt;
begin
  AssignFile(F, FName);
  try
    Reset(F, 1);
    ProgressBar1.Max := FileSize(F);
    if (ProgressBar1.Max > 10) then
    begin
      // amount to move when StepIt method called
      ProgressBar1.Step := ProgressBar1.Max div 10;
      ProgressBar1.Step := Min(ProgressBar1.Step, 2048);
    end
    else
      ProgressBar1.Step := ProgressBar1.Max;
    while (ProgressBar1.Position < ProgressBar1.Max) do
    begin
      // read one Step size chunk of data to buffer
      BlockRead(F, MyData, ProgressBar1.Step, BytesRead);
      // move the ProgressBar Position using StepIt
      ProgressBar1.StepIt; // move by Step amount
      // Do this or the read will wrap and start over!
      ProgressBar1.Step :=
          Min(ProgressBar1.Step, ProgressBar1.Max - ProgressBar1.Position);

    end;
  finally;
    CloseFile(F);
  end;
end;
```

See Also

Position (see page 4655)

Step (see page 4660)

StepBy (see page 4629)

27.1.43.4 TProgressBar Properties

The properties of the TProgressBar class are listed here.

Published Properties

	Name	Description
	Align (see page 4633)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4637)	Specifies how the control is anchored to its parent.
	BorderWidth (see page 4637)	Specifies the width of the control's border.
	Constraints (see page 4637)	Specifies the size constraints for the control.
	DragCursor (see page 4638)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4638)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4639)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4640)	Controls whether the control responds to mouse, keyboard, and timer events.
	Hint (see page 4641)	Contains the text string that can appear when the user moves the mouse over the control.
	Max (see page 4643)	Specifies the upper limit of the range of possible positions.
	Min (see page 4646)	Specifies the lower limit of the range of possible positions.
	Orientation (see page 4649)	Specifies whether the progress bar is oriented vertically or horizontally.
	ParentShowHint (see page 4649)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4650)	Identifies the pop-up menu associated with the control.
	Position (see page 4655)	Specifies the current position of the progress bar.
	ShowHint (see page 4658)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Smooth (see page 4660)	Specifies whether the progress bar is smooth or segmented.
	Step (see page 4660)	Specifies the amount that Position (see page 4655) increases when the StepIt (see page 4631) method is called.
	TabOrder (see page 4662)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 4662)	Determines if the user can tab to a control.
	Visible (see page 4663)	Determines whether the component appears onscreen.

27.1.43.4.1 TProgressBar.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of

the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
```

```
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form’s width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
```

```
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

27.1.43.4.2 TProgressBar.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.43.4.3 TProgressBar.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the progress bar's border. This is the area within the edge of the control that always shows the background color, as specified by the Color property. The progress fill and the caption are clipped to the area within the border.

See Also

Color (see page 5812)

27.1.43.4.4 TProgressBar.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.43.4.5 TProgressBar.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

27.1.43.4.6 TProgressBar.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

27.1.43.4.7 TProgressBar.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

27.1.43.4.8 TProgressBar.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls

appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.43.4.9 TProgressBar.Hint Property

Contains the text string that can appear when the user moves the mouse over the control.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Use the Hint property to provide a string of help text either as a Help Hint, or as help text on a particular location such as a status bar.

A Help Hint is a box containing help text that appears for a control when the user moves the mouse pointer over the control and pauses momentarily. To set up Help Hints:

Specify the Hint property of each control for which a Help Hint should appear.

Set the ShowHint property of each appropriate control to true, or set the ParentShowHint property of all controls to true and set the ShowHint property of the form to true.

At runtime, set the value of the application's ShowHint property to true.

To show the Hint on a status bar or other location, use the OnHint event handler of the application. The application's OnHint event occurs when the mouse pointer moves over the control.

Specify a hint to be used for both a Help Hint box and by an OnHint handler by specifying two values separated by a | character (the vertical bar "or" symbol). For example,

```
Edit1.Hint := 'Name|Enter Name in the edit box';
Edit1->Hint = "Name|Enter Name in the edit box";
```

The "Name" part appears in the Help Hint box and the "Enter full name in the edit box" part can be extracted using the GetLongHint function for use in an OnHint event handler.

If Hint contains only one value, the entire string is used as a Help Hint and returned by the GetLongHint function. If a control has no Hint value specified, but its parent control does, the control uses the Hint value of the parent control (as long as the control's ShowHint property is true).

Note: If the application's ShowHint property is false, the Help Hint does not appear, but the OnHint event handler is still called.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
```

```
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

GetLongHint (see page 6252)

GetShortHint (see page 6253)

ParentShowHint (see page 5835)

ShowHint (see page 5840)

Hint (see page 13712)

OnHint (see page 13731)

ShowHint (see page 13721)

Hint (see page 312)

27.1.43.4.10 TProgressBar.Max Property

Specifies the upper limit of the range of possible positions.

Pascal

```
property Max: Integer;
```

C++

```
__property int Max;
```

Description

Use Max along with the Min (see page 4646) property to establish the range of possible positions a progress bar. When the process tracked by the progress bar is complete, the value of Position (see page 4655) should equal Max.

C++ Examples:

```
/*
This example requires two ProgressBars, two labels and a
Memo.  The ProgressBar and the label captions are updated as
the mouse moves inside the Memo.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ProgressBar1->Min = 0;
  ProgressBar1->Max = Memo1->Width;
  ProgressBar2->Min = 0;
  ProgressBar1->Max = Memo1->Height;

}

void __fastcall TForm1::Memo1MouseMove(TObject *Sender, TShiftState Shift,
      int X, int Y)
{
  Label1->Caption = IntToStr(X);
  Label2->Caption = IntToStr(Y);
  ProgressBar1->Position = X;
  ProgressBar2->Position = Y;
}
/*
Reads through all records in the Customers table.
Updates the ProgressBar accordingly.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ProgressBar1->Min = 0;
  ProgressBar1->Max = Customers->RecordCount;
  Customers->First();
  for (int i = ProgressBar1->Min; i <= ProgressBar1->Max; i++)
  {
    ProgressBar1->Position = i;
    Customers->Next();
    // do something with the current record
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  Customers = new TTable(Form1);
  Customers->Active = false; // The Table component must not be active
  Customers->DatabaseName = "DBDEMOS";
  Customers->TableType = ttParadox;
  Customers->TableName = "CustInfo";
  Customers->Active = False;
  if (Customers->Exists) // Don't overwrite an existing table
  {
    Customers->Close();
    Customers->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Customers->FieldDefs->Clear();
  TFieldDef *pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Customers->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Customers->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Customers->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Customers->CreateTable();
  Customers->Active = True;
  for (int i = 1; i <= 20; i++)
    Customers->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  DS2->DataSet = Customers;
  DBGrid2->DataSource->DataSet = Customers;
  Customers->Active = True;
}
```

Delphi Examples:

```
{
This example requires two ProgressBars, two labels and a
Memo.  The ProgressBar and the label captions are updated as
the mouse moves inside the Memo.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ProgressBar1.Min := 0;
  ProgressBar1.Max := Memo1.Width;
  ProgressBar2.Min := 0;
  ProgressBar1.Max := Memo1.Height;
end;

procedure TForm1.Memo1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  Label1.Caption := IntToStr(X);
  Label2.Caption := IntToStr(Y);
  ProgressBar1.Position := X;
  ProgressBar2.Position := Y;
end;
{
Reads through all records in the Customers table.
Updates the ProgressBar accordingly.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  i: Integer;
begin
  with ProgressBar1 do
  begin
    Min := 0;
    Max := Customers.RecordCount;
    Customers.First;
    for i := Min to Max do
    begin
      Position := i;
      Customers.Next;
      // do something with the current record
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Customers:= TTable.Create(Form1);
  with Customers do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Active := False;
//  Overwrite any existing table
    if Customers.Exists then
    begin
      Customers.Close;
      Customers.DeleteTable;
    end;
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
```

```
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
      Customers.Active:= True;
      for i := 1 to 100 do
        Customers.AppendRecord([i*111, i*222]);
    end;
  end;
  DS2.DataSet:= Customers;
  DBGrid2.DataSource.DataSet:= Customers;
  Customers.Active:= True;
end;
```

See Also

Min (see page 4646)

Position (see page 4655)

27.1.43.4.11 TProgressBar.Min Property

Specifies the lower limit of the range of possible positions.

Pascal

```
property Min: Integer;
```

C++

```
__property int Min;
```

Description

Use Max (see page 4643) along with the Min property to establish the range of possible positions a progress bar. When the process tracked by the progress bar begins, the value of Position (see page 4655) should equal Min.

C++ Examples:

```
/*
This example requires two ProgressBars, two labels and a
Memo.  The ProgressBar and the label captions are updated as
the mouse moves inside the Memo.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ProgressBar1->Min = 0;
  ProgressBar1->Max = Memo1->Width;
  ProgressBar2->Min = 0;
  ProgressBar1->Max = Memo1->Height;

}

void __fastcall TForm1::Memo1MouseMove(TObject *Sender, TShiftState Shift,
      int X, int Y)
{
  Label1->Caption = IntToStr(X);
  Label2->Caption = IntToStr(Y);
  ProgressBar1->Position = X;
  ProgressBar2->Position = Y;
}
```

```
/*
Reads through all records in the Customers table.
Updates the ProgressBar accordingly.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ProgressBar1->Min = 0;
  ProgressBar1->Max = Customers->RecordCount;
  Customers->First();
  for (int i = ProgressBar1->Min; i <= ProgressBar1->Max; i++)
  {
    ProgressBar1->Position = i;
    Customers->Next();
    // do something with the current record
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Customers = new TTable(Form1);
  Customers->Active = false; // The Table component must not be active
  Customers->DatabaseName = "DBDEMOS";
  Customers->TableType = ttParadox;
  Customers->TableName = "CustInfo";
  Customers->Active = False;
  if (Customers->Exists) // Don't overwrite an existing table
  {
    Customers->Close();
    Customers->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Customers->FieldDefs->Clear();
  TFieldDef *pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Customers->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Customers->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Customers->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Customers->CreateTable();
  Customers->Active = True;
  for (int i = 1; i <= 20; i++)
    Customers->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  DS2->DataSet = Customers;
  DBGrid2->DataSource->DataSet = Customers;
  Customers->Active = True;
}
```

Delphi Examples:

```
{
This example requires two ProgressBars, two labels and a
Memo.  The ProgressBar and the label captions are updated as
the mouse moves inside the Memo.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    ProgressBar1.Min := 0;
    ProgressBar1.Max := Memo1.Width;
    ProgressBar2.Min := 0;
    ProgressBar1.Max := Memo1.Height;
  end;

  procedure TForm1.Memo1MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
  begin
    Label1.Caption := IntToStr(X);
    Label2.Caption := IntToStr(Y);
    ProgressBar1.Position := X;
    ProgressBar2.Position := Y;
  end;
  {
  Reads through all records in the Customers table.
  Updates the ProgressBar accordingly.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    i: Integer;
  begin
    with ProgressBar1 do
    begin
      Min := 0;
      Max := Customers.RecordCount;
      Customers.First;
      for i := Min to Max do
      begin
        Position := i;
        Customers.Next;
        // do something with the current record
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    Customers:= TTable.Create(Form1);
    with Customers do
    begin
      DatabaseName := 'DBDEMOS';
      TableType := ttParadox;
      TableName := 'CustInfo';
      Active := False;
  //  Overwrite any existing table
      if Customers.Exists then
      begin
        Customers.Close;
        Customers.DeleteTable;
      end;
      begin
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
        { Next, describe the fields in the table }
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
```

```
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Customers.Active:= True;
        for i := 1 to 100 do
          Customers.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Customers;
    DBGrid2.DataSource.DataSet:= Customers;
    Customers.Active:= True;
  end;
```

See Also

Max (see page 4643)

Position (see page 4655)

27.1.43.4.12 TProgressBar.Orientation Property

Specifies whether the progress bar is oriented vertically or horizontally.

Pascal

```
property Orientation: TProgressBarOrientation;
```

C++

```
__property TProgressBarOrientation Orientation;
```

Description

Use Orientation to get or set whether the progress bar is oriented vertically or horizontally.

27.1.43.4.13 TProgressBar.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.43.4.14 TProgressBar.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
```

```
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
```

```
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
```

```
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
```

```
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.43.4.15 TProgressBar.Position Property

Specifies the current position of the progress bar.

Pascal

```
property Position: Integer;
```

C++

```
__property int Position;
```

Description

You can read Position to determine how far the process tracked by the progress bar has advanced from Min (see page 4646) toward Max (see page 4643). Set Position to cause the progress bar to display a position between Min (see page 4646) and Max (see page 4643). For example, when the process tracked by the progress bar completes, set Position to Max (see page 4643) so that it appears completely filled.

When a progress bar is created, Min (see page 4646) and Max (see page 4643) represent percentages, where Min (see page 4646) is 0 (0% complete) and Max (see page 4643) is 100 (100% complete). If these values are not changed, Position is the percentage of the process that has already been completed.

C++ Examples:

```
/*
This example requires two ProgressBars, two labels and a
Memo.  The ProgressBar and the label captions are updated as
the mouse moves inside the Memo.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ProgressBar1->Min = 0;
  ProgressBar1->Max = Memo1->Width;
  ProgressBar2->Min = 0;
  ProgressBar1->Max = Memo1->Height;

}

void __fastcall TForm1::Memo1MouseMove(TObject *Sender, TShiftState Shift,
      int X, int Y)
{
  Label1->Caption = IntToStr(X);
  Label2->Caption = IntToStr(Y);
  ProgressBar1->Position = X;
  ProgressBar2->Position = Y;
}
/*
Reads through all records in the Customers table.
Updates the ProgressBar accordingly.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ProgressBar1->Min = 0;
  ProgressBar1->Max = Customers->RecordCount;
  Customers->First();
  for (int i = ProgressBar1->Min; i <= ProgressBar1->Max; i++)
  {
    ProgressBar1->Position = i;
    Customers->Next();
    // do something with the current record
```

```
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Customers = new TTable(Form1);
  Customers->Active = false; // The Table component must not be active
  Customers->DatabaseName = "DBDEMOS";
  Customers->TableType = ttParadox;
  Customers->TableName = "CustInfo";
  Customers->Active = False;
  if (Customers->Exists) // Don't overwrite an existing table
  {
    Customers->Close();
    Customers->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Customers->FieldDefs->Clear();
  TFieldDef *pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Customers->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Customers->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Customers->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Customers->CreateTable();
  Customers->Active = True;
  for (int i = 1; i <= 20; i++)
    Customers->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  DS2->DataSet = Customers;
  DBGrid2->DataSource->DataSet = Customers;
  Customers->Active = True;
}
```

Delphi Examples:

```
{
This example requires two ProgressBars, two labels and a
Memo.  The ProgressBar and the label captions are updated as
the mouse moves inside the Memo.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ProgressBar1.Min := 0;
  ProgressBar1.Max := Memo1.Width;
  ProgressBar2.Min := 0;
  ProgressBar1.Max := Memo1.Height;
end;

procedure TForm1.Memo1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  Label1.Caption := IntToStr(X);
  Label2.Caption := IntToStr(Y);
  ProgressBar1.Position := X;
  ProgressBar2.Position := Y;
end;
```

```
{
Reads through all records in the Customers table.
Updates the ProgressBar accordingly.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
begin
  with ProgressBar1 do
  begin
    Min := 0;
    Max := Customers.RecordCount;
    Customers.First;
    for i := Min to Max do
    begin
      Position := i;
      Customers.Next;
      // do something with the current record
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Customers:= TTable.Create(Form1);
  with Customers do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Active := False;
//  Overwrite any existing table
    if Customers.Exists then
    begin
      Customers.Close;
      Customers.DeleteTable;
    end;
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
```

```
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Customers.Active:= True;
        for i := 1 to 100 do
          Customers.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Customers;
    DBGrid2.DataSource.DataSet:= Customers;
    Customers.Active:= True;
  end;
```

See Also

Max (see page 4643)

Min (see page 4646)

StepBy (see page 4629)

StepIt (see page 4631)

27.1.43.4.16 TProgressBar.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
```

```
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.43.4.17 TProgressBar.Smooth Property

Specifies whether the progress bar is smooth or segmented.

Pascal

```
property Smooth: Boolean;
```

C++

```
__property Boolean Smooth;
```

Description

Use Smooth to specify whether the progress bar is smooth or segmented.

27.1.43.4.18 TProgressBar.Step Property

Specifies the amount that Position (see page 4655) increases when the StepIt (see page 4631) method is called.

Pascal

```
property Step: Integer;
```

C++

```
__property int Step;
```

Description

Set Step to specify the granularity of the progress bar. Step should reflect the size of each step in the process tracked by the progress bar, in the logical units used by the Max (see page 4643) and Min (see page 4646) properties.

When a progress bar is created, Min (see page 4646) and Max (see page 4643) represent percentages, where Min (see page 4646) is 0 (0% complete) and Max (see page 4643) is 100 (100% complete). If these values are not changed, Step is the percentage of the process completed before the user is provided with additional visual feedback.

When the StepIt (see page 4631) method is called, the value of Position (see page 4655) increases by Step.

C++ Examples:

```
/*
This example reads bytes from a file in tenths of the size
of the file. The ProgressBar displays the status of each
1/10th step as it's read into the buffer.  This example
requires a TButton and a TProgressBar control on the form.
*/
#include <stdio.h>     // for FILE, fopen, fstat, fileno, fread and fclose
#include <sys\stat.h>  // for fstat and the stat struct
#include <Math.hpp>    // for Min

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FILE *F;
  char MyData[2047];
  long BytesRead;

  F = fopen("../about.txt", "r");   // path relative to Debug
                                    // use a file larger than 2048 bytes to make it
interesting.
  if (F)
  {
    struct stat statbuf;
    fstat(fileno(F), &statbuf);
    ProgressBar1->Max = statbuf.st_size;
```

```
    if (ProgressBar1->Max > 10)
    {
      ProgressBar1->Step = (ProgressBar1->Max)/10;
      ProgressBar1->Step = Min(ProgressBar1->Step, 2047);
    }
    else
      ProgressBar1->Step = ProgressBar1->Max;
    char *DataBuffer = new char[ProgressBar1->Step];
    for (ProgressBar1->Position = 0;
         ProgressBar1->Position < ProgressBar1->Max;
         ProgressBar1->StepIt())  // move the ProgressBar Position using StepIt
    {
      fread(DataBuffer, ProgressBar1->Step, 1, F);
      // Do this or the read will wrap and start over!
      ProgressBar1->Step =
        Min(ProgressBar1->Step, ProgressBar1->Max - ProgressBar1->Position);
    }
    delete DataBuffer;
    fclose(F);
  }
}
```

Delphi Examples:

```
{
This example reads bytes from a file in tenths of the size
of the file. The ProgressBar displays the status of each
1/10th step as it's read into the buffer.  This example
requires a TButton and a TProgressBar control on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
const
  FName = 'about.txt';  // use a file larger than 2048 bytes to make it interesting.
var
  F: File;
  MyData: array[1..2048] of byte;
  BytesRead: LongInt;
begin
  AssignFile(F, FName);
  try
    Reset(F, 1);
    ProgressBar1.Max := FileSize(F);
    if (ProgressBar1.Max > 10) then
    begin
      // amount to move when StepIt method called
      ProgressBar1.Step := ProgressBar1.Max div 10;
      ProgressBar1.Step := Min(ProgressBar1.Step, 2048);
    end
    else
      ProgressBar1.Step := ProgressBar1.Max;
    while (ProgressBar1.Position < ProgressBar1.Max) do
    begin
      // read one Step size chunk of data to buffer
      BlockRead(F, MyData, ProgressBar1.Step, BytesRead);
      // move the ProgressBar Position using StepIt
      ProgressBar1.StepIt; // move by Step amount
      // Do this or the read will wrap and start over!
      ProgressBar1.Step :=
          Min(ProgressBar1.Step, ProgressBar1.Max - ProgressBar1.Position);

    end;
  finally;
    CloseFile(F);
  end;
end;
```

See Also

Max (see page 4643)

Min (see page 4646)

Position (see page 4655)

StepIt (see page 4631)

27.1.43.4.19 TProgressBar.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

27.1.43.4.20 TProgressBar.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

27.1.43.4.21 TProgressBar.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
```

```
  edit control placed somewhere on the client area of the
  sheet.  Unchecking the checkbox will cause the client area
  (not the tab area) of the Tab Control to become invisible.
  }
  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    PageControl1.ActivePage.Visible := CheckBox1.Checked;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    CheckBox1.Checked := PageControl1.ActivePage.Visible;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.43.5 TProgressBar Events

The events of the TProgressBar class are listed here.

Published Events

	Name	Description
	OnContextPopup (see page 4665)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 4665)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4668)	Occurs when the user drags an object over a control.
	OnEndDock (see page 4670)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4670)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4671)	Occurs when a control receives the input focus.
	OnExit (see page 4674)	Occurs when the input focus shifts away from one control to another.
	OnMouseActivate (see page 4675)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4676)	Occurs when the user presses a mouse button with the mouse pointer over a control.

	OnMouseEnter (see page 4679)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4679)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4679)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4681)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 4684)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4684)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

27.1.43.5.1 TProgressBar.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.43.5.2 TProgressBar.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

27.1.43.5.3 TProgressBar.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.
```

```
procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.43.5.4 TProgressBar.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.43.5.5 TProgressBar.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

27.1.43.5.6 TProgressBar.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
```

```
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;
```

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

27.1.43.5.7 TProgressBar.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
```

```
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

27.1.43.5.8 TProgressBar.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.43.5.9 TProgressBar.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
```

```
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.43.5.10 TProgressBar.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

27.1.43.5.11 TProgressBar.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

27.1.43.5.12 TProgressBar.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
```

```
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.43.5.13 TProgressBar.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
```

```
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
```

```
      end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

27.1.43.5.14 TProgressBar.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

27.1.43.5.15 TProgressBar.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

27.1.44 TRichEdit Class

TRichEdit is a wrapper for a Windows rich text edit control.

Class Hierarchy

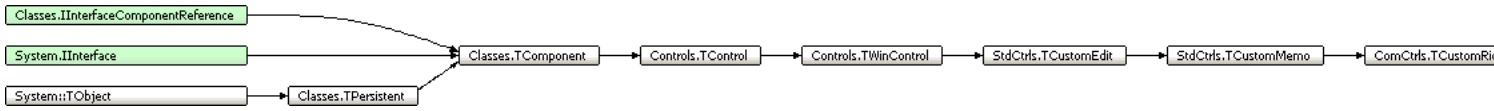

Pascal

```
TRichEdit = class(TCustomRichEdit);
```

C++

```
class TRichEdit : public TCustomRichEdit;
```

File

ComCtrls

Description

Use a TRichEdit object to put a standard Windows rich text edit control on a form. Rich text edit controls let the user enter text that includes variation in font attributes and paragraph formatting information.

TRichEdit provides the properties and methods to enter and work with rich text. TRichEdit does not provide any user interface components to make these formatting options available to the user. Applications must implement the user interface components to surface the rich text capabilities of a TRichEdit object. To make this easier, a number of pre-defined standard actions operate

on a rich edit control to perform common formatting tasks.

TRichEdit publishes many of the properties inherited from TCustomRichEdit (see page 3854), but does not introduce any new behavior.

See Also

TDBMemo (see page 9149)

TMemo (see page 21678)

TParaAttributes (see page 4606)

TTextAttributes (see page 4982)

TRichEditAction (see page 11992)

Adding Scroll Bars at Runtime

Setting Text Alignment

Memo and Rich Edit Controls

Edit Controls

27.1.44.1 TRichEdit Members

The following tables list the members exposed by TRichEdit.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21245)	Generates an OnChange (see page 21275) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 3868)	Initializes a window-creation parameter record passed in the Params parameter.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 21248)	Initializes a window-creation parameter record passed in the Params parameter.
V	CreateParams (see page 21375)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWindowHandle (see page 21248)	Creates a Windows control to represent the control.
V	CreateWindowHandle (see page 21376)	Creates a window based on the window-creation parameter record.
V	CreateWnd (see page 3868)	Creates a Windows control corresponding to the edit control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CreateWnd (see page 21248)	Creates a Windows control corresponding to the edit control.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 3868)	Destroys the rich edit control's window.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21251)	Destroys the Windows control that corresponds to the edit control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoSetMaxLength (see page 3869)	Sets the MaxLength property.
	DoSetMaxLength (see page 21251)	Sets the MaxLength (see page 21263) property.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetCaretPos (see page 3871)	Returns the cursor position.
	GetCaretPos (see page 21376)	Returns the value of the CaretPos (see page 21380) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.

	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelLength (see page 3871)	Returns the SelLength property.
	GetSelLength (see page 21252)	Returns the SelLength (see page 21265) property.
	GetSelStart (see page 3872)	Returns the SelStart property.
	GetSelStart (see page 21252)	Returns the SelStart (see page 21269) property.
	GetSelText (see page 3872)	Returns the SelText property.
	GetSelText (see page 21252)	Returns the SelText (see page 21272) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyPress (see page 21377)	Ensures that return characters are not entered into the text when WantReturns (see page 21384) is false.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	Loaded (see page 21377)	Initializes the memo control when it is first loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.

	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
V	RequestSize (see page 3874)	Resize request event dispatcher.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectionChange (see page 3875)	Selection change event dispatcher.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAlignment (see page 21377)	Sets the Alignment (see page 21379) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetAutoSize (see page 21257)	Sets the AutoSize (see page 21260) property.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
V	SetCaretPos (see page 3875)	Sets the cursor position in the text.
V	SetCaretPos (see page 21378)	Sets the CaretPos (see page 21380) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetLines (see page 21378)	Sets the Lines (see page 21380) property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.

	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetScrollBars (see page 21378)	Sets the ScrollBars (see page 21381) property.
	SetSelLength (see page 3876)	Sets the SelLength property.
	SetSelLength (see page 21257)	Sets the SelLength (see page 21265) property.
	SetSelStart (see page 3876)	Returns the SelStart property.
	SetSelStart (see page 21257)	Sets the SelStart (see page 21269) property.
	SetWordWrap (see page 21379)	Sets the WordWrap (see page 21384) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSelect (see page 21260)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AutoSize (see page 21260)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	CharCase (see page 21262)	Determines the case of the text within the edit control.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	OEMConvert (see page 21264)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	PasswordChar (see page 21265)	Indicates the character, if any, to display in place of the actual characters typed in the control.

	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 3867)	Deletes all text from the rich edit control.
V	Clear (see page 21245)	Deletes all text from the edit control.
	ClearSelection (see page 21245)	Removes the selected text from the edit control.
	ClearUndo (see page 21246)	Clears the undo buffer so that no changes to the text can be backed out.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopyToClipboard (see page 21246)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 3866)	Creates an instance of TCustomRichEdit (see page 3854).
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 21244)	Creates an instance of TCustomEdit (see page 21232).
V	Create (see page 21374)	Creates and initializes an instance of TCustomMemo (see page 21362).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CutToClipboard (see page 21249)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 21250)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 3866)	Destroys an instance of TCustomRichEdit (see page 3854).
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 21374)	Destroys an instance of TCustomMemo (see page 21362).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindText (see page 3869)	Searches a given range in the text for a target string.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetControlsAlignment (see page 21376)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetSelTextBuf (see page 3872)	Copies the selected text into a buffer and returns the number of characters copied.
	GetSelTextBuf (see page 21253)	Copies the selected text into a buffer and returns the number of characters copied.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.

	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	PasteFromClipboard (see page 21254)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
V	Print (see page 3873)	Formats and prints the contents of the rich edit control.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RegisterConversionFormat (see page 3874)	Registers an association between a file extension and a TConversion (see page 3582) object that can convert between the native file format and rich text format (RTF).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectAll (see page 21255)	Selects all text in the edit control.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.

	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSelTextBuf (see page 21258)	Replaces the selected text with a null-terminated string.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Undo (see page 21259)	Backs out all changes in the undo buffer.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	CanUndo (see page 21262)	Indicates whether the edit control contains changes that can be backed out.
	CaretPos (see page 21380)	Indicates the position of the caret in the client area of the memo.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DefAttributes (see page 3877)	Describes the rich text characteristics of the default font for the rich edit control
	DefaultConverter (see page 3878)	Specifies the class of the TConversion (see page 3582) object that is used to convert between native file format and the internal format of a rich text control when working with files that do not have a registered extension.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.

R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Modified (see page 21263)	Indicates whether the user edited the text of the edit control.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	PageRect (see page 3884)	Specifies the dimensions, in pixels, of the logical page size used when printing the contents of a rich text edit control.
R	Paragraph (see page 3884)	Specifies the formatting information for the current paragraphs.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelAttributes (see page 3887)	Describes the rich text characteristics of the selected text in the rich edit control.
	SelLength (see page 21265)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 21269)	Specifies the position of the first selected character in the text.
	SelText (see page 21272)	Specifies the selected portion of the edit control's text.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	Text (see page 21273)	Contains a text string associated with the control.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 4738)	Occurs when the text for the edit control may have changed.
	OnClick (see page 4738)	Occurs when the user clicks the control.
	OnContextPopup (see page 4740)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 4741)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 4741)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4743)	Occurs when the user drags an object over a control.
	OnEndDock (see page 4746)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4746)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4747)	Occurs when a control receives the input focus.
	OnExit (see page 4749)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 4751)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 4754)	Occurs when a key is pressed.
	OnKeyUp (see page 4755)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 4757)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4758)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4760)	Occurs when the user moves the mouse into a control.

	OnMouseLeave (see page 4760)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4760)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4763)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 4765)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 4765)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 4766)	Occurs when the mouse wheel is rotated upward.
	OnProtectChange (see page 4766)	Occurs when a user attempts to modify text that is marked as protected.
	OnResizeRequest (see page 4766)	Occurs when the text has grown either smaller or larger than the control's window size.
	OnSaveClipboard (see page 4767)	Occurs when the window of the edit control is about to be destroyed and the edit control has delayed rendering of text copied to the clipboard..
	OnSelectionChange (see page 4767)	Occurs when the current selection changes.
	OnStartDock (see page 4768)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4768)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 4699)	Determines how the control aligns within its container (parent control).
	Alignment (see page 4703)	Determines how the text is aligned within the memo control.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 4703)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 4704)	Specifies which edges of the control are beveled.
	BevelInner (see page 4704)	Specifies the cut of the inner bevel.
	BevelKind (see page 4705)	Specifies the control's bevel style.
	BevelOuter (see page 4705)	Specifies the cut of the outer bevel.
	BevelWidth (see page 4705)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 4706)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 4706)	Determines whether the edit control has a single line border around the client area.
	BorderWidth (see page 4707)	Specifies the width of the control's border.
	Color (see page 4707)	Specifies the background color of the control.
	Constraints (see page 4708)	Specifies the size constraints for the control.
	Ctl3D (see page 4708)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 4709)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4709)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4710)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4711)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4712)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	HideScrollBars (see page 4713)	Determines whether the scroll bars disappear when not needed.
	HideSelection (see page 4715)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 4716)	Determines the behavior of the input method editor (IME).
	ImeName (see page 4716)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Lines (see page 4717)	Contains the individual lines of text in the rich text edit control.
	Margins (see page 5831)	Specifies the margins for the control.
	MaxLength (see page 4719)	Specifies the maximum number of characters the user can enter into the edit control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 4719)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 4720)	Determines where a control looks for its color information.
	ParentCtl3D (see page 4720)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 4721)	Determines where a control looks for its font information.
	ParentShowHint (see page 4721)	Determines where a control looks to find out if its Help Hint should be shown.
	PlainText (see page 4722)	Controls whether the rich edit control treats the text as plain text or rich text when streaming to or from a file.
	PopupMenu (see page 4723)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 4727)	Determines whether the user can change the text of the edit control.
	ScrollBars (see page 4728)	Determines whether the memo control has scroll bars.
	ShowHint (see page 4730)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 4731)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 4732)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 4732)	Determines whether the component appears onscreen.
	WantReturns (see page 4734)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 4734)	Determines whether the user can insert tab characters into the text.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	WordWrap (see page 4735)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

27.1.44.2 TRichEdit Properties

The properties of the TRichEdit class are listed here.

Published Properties

	Name	Description
	Align (see page 4699)	Determines how the control aligns within its container (parent control).
	Alignment (see page 4703)	Determines how the text is aligned within the memo control.
	Anchors (see page 4703)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 4704)	Specifies which edges of the control are beveled.
	BevelInner (see page 4704)	Specifies the cut of the inner bevel.
	BevelKind (see page 4705)	Specifies the control's bevel style.
	BevelOuter (see page 4705)	Specifies the cut of the outer bevel.
	BevelWidth (see page 4705)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 4706)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 4706)	Determines whether the edit control has a single line border around the client area.
	BorderWidth (see page 4707)	Specifies the width of the control's border.
	Color (see page 4707)	Specifies the background color of the control.
	Constraints (see page 4708)	Specifies the size constraints for the control.
	Ctl3D (see page 4708)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 4709)	Indicates the image used to represent the mouse pointer when the control is being dragged.

	DragKind (see page 4709)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4710)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4711)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4712)	Controls the attributes of text written on or in the control.
	HideScrollBars (see page 4713)	Determines whether the scroll bars disappear when not needed.
	HideSelection (see page 4715)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 4716)	Determines the behavior of the input method editor (IME).
	ImeName (see page 4716)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Lines (see page 4717)	Contains the individual lines of text in the rich text edit control.
	MaxLength (see page 4719)	Specifies the maximum number of characters the user can enter into the edit control.
	ParentBiDiMode (see page 4719)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 4720)	Determines where a control looks for its color information.
	ParentCtl3D (see page 4720)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 4721)	Determines where a control looks for its font information.
	ParentShowHint (see page 4721)	Determines where a control looks to find out if its Help Hint should be shown.
	PlainText (see page 4722)	Controls whether the rich edit control treats the text as plain text or rich text when streaming to or from a file.
	PopupMenu (see page 4723)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 4727)	Determines whether the user can change the text of the edit control.
	ScrollBars (see page 4728)	Determines whether the memo control has scroll bars.
	ShowHint (see page 4730)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 4731)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 4732)	Determines if the user can tab to a control.
	Visible (see page 4732)	Determines whether the component appears onscreen.
	WantReturns (see page 4734)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 4734)	Determines whether the user can insert tab characters into the text.
	WordWrap (see page 4735)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

27.1.44.2.1 TRichEdit.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
```

```
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
```

```
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
```

```
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

27.1.44.2.2 TRichEdit.Alignment Property

Determines how the text is aligned within the memo control.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to change the way the text is formatted by the memo control. Alignment can take one of the following values:

Value	Meaning
taLeftJustify	Align text to the left side of the control
taCenter	Center text horizontally in the control.
taRightJustify	Align text to the right side of the control.

See Also

SetAlignment (see page 21377)

Text (see page 5842)

Displaying and Editing Text in a Memo Control

27.1.44.2.3 TRichEdit.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other

aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.44.2.4 TRichEdit.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

27.1.44.2.5 TRichEdit.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

27.1.44.2.6 TRichEdit.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

27.1.44.2.7 TRichEdit.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

27.1.44.2.8 TRichEdit.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

27.1.44.2.9 TRichEdit.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

27.1.44.2.10 TRichEdit.BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the edit control stands out. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the edit control has a single-line border around the client area. If

BorderStyle is bsNone, there will be no border.

27.1.44.2.11 TRichEdit.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics or text drawn by the control is clipped to the area within the border.

See Also

ClientRect

BevelOuter (see page 6194)

27.1.44.2.12 TRichEdit.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
```

```
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

27.1.44.2.13 TRichEdit.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.44.2.14 TRichEdit.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

27.1.44.2.15 TRichEdit.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

27.1.44.2.16 TRichEdit.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

27.1.44.2.17 TRichEdit.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
```

```
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

27.1.44.2.18 TRichEdit.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.44.2.19 TRichEdit.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

27.1.44.2.20 TRichEdit.HideScrollBars Property

Determines whether the scroll bars disappear when not needed.

Pascal

```
property HideScrollBars: Boolean;
```

C++

```
__property Boolean HideScrollBars;
```

Description

Use HideScrollBars to determine whether the scroll bars disappear from the edit control when the text appears completely within the edit window. Set HideScrollBars to false to prevent the scroll bars from flashing on and off when the contents of the rich edit control change.

HideScrollBars does nothing when the ScrollBars property is set to ssNone (The default for TRichEdit).

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
```

```
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
    ssHorizontal,
    ssNone,
    ssVertical);
begin
  RichEdit1.ScrollBars :=
    ScrollBarA[RadioGroup1.ItemIndex];
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with RadioGroup1, Items do
  begin
    Caption := 'Scroll Bars';
    Clear;
    Add('ssBoth');
    Add('ssHorizontal');
    Add('ssNone');
    Add('ssVertical');
  end;
  with CheckBox1 do
  begin
    Caption := 'HideScrollBars';
    Checked := True;
  end;
  Button1.Caption := 'Loaded';
  RichEdit1.Text := 'Just a little text.';
  RichEdit1.WordWrap := False;
```

```
end;
```

See Also

ScrollBars (see page 21381)

27.1.44.2.21 TRichEdit.HideSelection Property

Determines whether the visual indication of the selected text remains when focus shifts to another control.

Pascal

```
property HideSelection: Boolean;
```

C++

```
__property Boolean HideSelection;__property Boolean HideSelection;
```

Description

Set HideSelection to false when it is important to provide visual feedback of the selected portion of the text even when the edit control does not have focus. Set HideSelection to true to make the visual indication of selection appear only when the edit control has focus. HideSelection does not affect the actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms containing many edit controls look busy.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
HideSelection, Height example
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
```

```
    for I := 0 to 2 do
      with TRichEdit.Create(Self) do
      begin
        Parent := Self;
        Top := Height * I;
        Text := 'This text will take the focus.';
        SelStart := 0;
        SelLength := Length(Text);
        if (I = 1) then
          HideSelection := False;
      end;
  end;
```

See Also

SelLength

SelStart

SelText

SetSelTextBuf (see page 21258)

27.1.44.2.22 TRichEdit.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

27.1.44.2.23 TRichEdit.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

27.1.44.2.24 TRichEdit.Lines Property

Contains the individual lines of text in the rich text edit control.

Pascal

```
property Lines: TStrings;
```

C++

```
__property TStrings Lines;__property TStrings Lines;
```

Description

Use Lines to manipulate the text in the rich text edit control on a line by line basis. Lines is a TStrings object, so TStrings methods may be used for Lines to perform manipulations such as counting the lines of text, adding lines, deleting lines, or replacing the text in lines.

To work with the text as one chunk, use the Text property. To manipulate individual lines of text, the Lines property works better.

Warning: If you paste formatted text into the Lines property at design time, the formatting is lost when the form is saved (it does not appear at runtime). To set the value of a rich edit control at design time, you must save the formatted text in some other manner and use the LoadFromStream or LoadFromFile method of the Lines property to load the value in the OnShow method of the parent form.

C++ Examples:

```
/*
This example requires only a form.  A Rich Edit control is
created on the form and aligned to take the entire client
area of the form.  A series of lines are written to the Rich
Edit control's Lines property, and the Paragraph property
manipulated such that the first items displayed are bulleted,
a second group of lines are unbulleted and centered, and the
last group is left justified and indented.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TRichEdit *pRich = new TRichEdit(this);
  pRich->Parent = this;
  pRich->Align = alClient;
  pRich->Lines->Clear();
  // set numbering style
  pRich->Paragraph->Numbering = nsBullet;
  pRich->Lines->Add("Introduction");
  pRich->Lines->Add("New members to our team");
  pRich->Lines->Add("New Budget discussion");
  pRich->Lines->Add("Facilities");
  pRich->Lines->Add("Q & A");
  pRich->Paragraph->Numbering = nsNone;
  pRich->Paragraph->Alignment = taCenter;
  pRich->Lines->Add("");
  pRich->Lines->Add("Suggested Topics: ");
  pRich->Lines->Add("");
  pRich->Paragraph->Alignment = taLeftJustify;
  pRich->Paragraph->FirstIndent = 10;
  pRich->Lines->Add("");
  pRich->Lines->Add("Parking lot repair");
  pRich->Lines->Add("Cost overruns");
}
```

Delphi Examples:

```
{
This example requires only a form.  A Rich Edit control is
created on the form and aligned to take the entire client
area of the form.  A series of lines are written to the Rich
Edit control's Lines property, and the Paragraph property
manipulated such that the first items displayed are bulleted,
a second group of lines are unbulleted and centered, and the
last group is left justified and indented.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with TRichEdit.Create(Self) do
  begin
    Parent := Self;
    Align := alClient;
    Lines.Clear;
    // set numbering style
    Paragraph.Numbering := nsBullet;
    Lines.Add('Introduction');
    Lines.Add('New members to our team');
    Lines.Add('New Budget discussion');
    Lines.Add('Facilities');
    Lines.Add('Q & A');
    Paragraph.Numbering := nsNone;
    Paragraph.Alignment := taCenter;
    Lines.Add('');
    Lines.Add('Suggested Topics:');
    Lines.Add('');
    Paragraph.Alignment := taLeftJustify;
```

```
      Paragraph.FirstIndent := 10;
      Lines.Add('');
      Lines.Add('Parking lot repair');
      Lines.Add('Cost overruns');
    end;
  end;
```

See Also

SelText

Text (see page 5842)

WordWrap (see page 21384)

LoadFromFile (see page 3061)

LoadFromStream (see page 3062)

27.1.44.2.25 TRichEdit.MaxLength Property

Specifies the maximum number of characters the user can enter into the edit control.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the edit control. A value of 0 indicates that there is no application-defined limit on the length.

Use MaxLength to limit the length of the text in an edit control if that text will be copied into a fixed-length buffer.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

Note: The unit of MaxLength is characters (not bytes) if the application is using themes.

See Also

Text (see page 5842)

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

27.1.44.2.26 TRichEdit.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

27.1.44.2.27 TRichEdit.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

27.1.44.2.28 TRichEdit.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

27.1.44.2.29 TRichEdit.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

27.1.44.2.30 TRichEdit.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.44.2.31 TRichEdit.PlainText Property

Controls whether the rich edit control treats the text as plain text or rich text when streaming to or from a file.

Pascal

```
property PlainText: Boolean;
```

C++

```
__property Boolean PlainText;
```

Description

To write the rich text in the control to a plain text file, set PlainText to true before streaming the text to a file. To ignore the rich text information encoded in a file, set PlainText to true before streaming the text to the control. To stream in the rich text attributes encoded in a file, or save the encoding of the rich text attributes to a file, set PlainText to false.

If the rich text attributes of a file are encoded in some format other than rich text format (RTF), it is necessary to use a converter on the text, even when PlainText is true.

Note: Rich edit controls do not directly support streaming. Use the Lines property to stream to or from a file.

C++ Examples:

```
/*
This example requires two TRichEdit controls placed on the
form.  When the form becomes visible, the first Rich Edit
control will display the rich text in its raw form, including
formatting characters. The second will show the rich text in
its intended format.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
// you may need to change this path to suit your environment
  char const *Path = "..\\OVERVIEW.RTF";
  RichEdit1->PlainText = true;
  RichEdit1->Lines->LoadFromFile(Path);
  RichEdit2->PlainText = false;
  RichEdit2->Lines->LoadFromFile(Path);
}
```

Delphi Examples:

```
{
This example requires two TRichEdit controls placed on the
form.  When the form becomes visible, the first Rich Edit
control will display the rich text in its raw form, including
formatting characters. The second will show the rich text in
its intended format.
}

procedure TForm1.FormCreate(Sender: TObject);
const
  // you may need to change this path to suit your environment
  Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := True;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
```

```
  RichEdit2.PlainText := False;
  RichEdit2.Lines.LoadFromFile(Path);
  RichEdit2.ScrollBars := ssVertical;
end;
```

See Also

DefaultConverter (see page 3878)

Lines (see page 3882)

RegisterConversionFormat (see page 3874)

27.1.44.2.32 TRichEdit.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
```

```
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
```

```
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
```

```
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
```

```
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.44.2.33 TRichEdit.ReadOnly Property

Determines whether the user can change the text of the edit control.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

Setting ReadOnly to true ensures that the text is not altered, while still allowing the user to select text. The selected text can then be manipulated by the application, or copied to the Clipboard.

See Also

CopyToClipboard (see page 21246)

GetSelTextBuf (see page 21253)

27.1.44.2.34 TRichEdit.ScrollBars Property

Determines whether the memo control has scroll bars.

Pascal

```
property ScrollBars: TScrollStyle;
```

C++

```
__property TScrollStyle ScrollBars;
```

Description

Use ScrollBars to give a multiline edit control horizontal or vertical scroll bars. ScrollBars can take one of the following values:

Value	Meaning
ssNone	The control has no scroll bars.
ssHorizontal	The control has a single scroll bar on the bottom edge.
ssVertical	The control has a single scroll bar on the right edge.
ssBoth	The control has a scroll bar on both the bottom and right edges.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
    ssHorizontal,
    ssNone,
    ssVertical);
begin
  RichEdit1.ScrollBars :=
    ScrollBarA[RadioGroup1.ItemIndex];
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with RadioGroup1, Items do
  begin
    Caption := 'Scroll Bars';
    Clear;
    Add('ssBoth');
    Add('ssHorizontal');
    Add('ssNone');
```

```
    Add('ssVertical');
  end;
  with CheckBox1 do
  begin
    Caption := 'HideScrollBars';
    Checked := True;
  end;
  Button1.Caption := 'Loaded';
  RichEdit1.Text := 'Just a little text.';
  RichEdit1.WordWrap := False;
end;
```

See Also

SetScrollBars (see page 21378)

WordWrap (see page 21384)

Displaying and Editing Text in a Memo Control

27.1.44.2.35 TRichEdit.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
```

```
      strcat(string, index);
      ListBox1->Items->Add(string);
    }
    ListBox1->Hint = "Select an item";
    ListBox1->ShowHint = true;
    Application->ShowHint = true;
  }
  /*
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  */
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.44.2.36 TRichEdit.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder

value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

27.1.44.2.37 TRichEdit.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

27.1.44.2.38 TRichEdit.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false,

the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
```

```
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.44.2.39 TRichEdit.WantReturns Property

Determines whether the user can insert return characters into the text.

Pascal

```
property WantReturns: Boolean;
```

C++

```
__property Boolean WantReturns;
```

Description

Set WantReturns to true to allow users to enter return characters into the text. Set WantReturns to false to allow the form to handle return characters instead.

For example, in a form with a default button (such as an OK button) and a memo control, if WantReturns is false, pressing Enter chooses the default button. If WantReturns is true, pressing Enter inserts a return character in the text.

Note: If WantReturns is false, users can still enter return characters into the text by pressing Ctrl+Enter.

See Also

Text (see page 5842)

WantTabs (see page 21384)

WordWrap (see page 21384)

27.1.44.2.40 TRichEdit.WantTabs Property

Determines whether the user can insert tab characters into the text.

Pascal

```
property WantTabs: Boolean;
```

C++

```
__property Boolean WantTabs;
```

Description

Set WantTabs to true to allow users to enter tab characters into the text. Set WantTabs to false if you want the tab character to select the next control on the form instead.

Note: If WantTabs is true, users can tab into the edit control, but they can't tab out.

See Also

Text (see page 5842)

WantReturns (see page 21384)

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

27.1.44.2.41 TRichEdit.WordWrap Property

Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to make the edit control wrap text at the right margin so it fits in the client area. The wrapping is cosmetic only. The text does not include any return characters that were not explicitly entered. Set WordWrap to false to have the edit control show a separate line only where return characters were explicitly entered into the text.

Note: There should be no use for a horizontal scroll bar if WordWrap is true.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
    ssHorizontal,
    ssNone,
    ssVertical);
begin
  RichEdit1.ScrollBars :=
    ScrollBarA[RadioGroup1.ItemIndex];
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with RadioGroup1, Items do
  begin
    Caption := 'Scroll Bars';
    Clear;
    Add('ssBoth');
    Add('ssHorizontal');
    Add('ssNone');
    Add('ssVertical');
  end;
```

```
  with CheckBox1 do
  begin
    Caption := 'HideScrollBars';
    Checked := True;
  end;
  Button1.Caption := 'Loaded';
  RichEdit1.Text := 'Just a little text.';
  RichEdit1.WordWrap := False;
end;
```

See Also

Lines (see page 21380)

ScrollBars (see page 21381)

Text (see page 5842)

WantReturns (see page 21384)

Displaying and Editing Text in a Memo Control

27.1.44.3 TRichEdit Events

The events of the TRichEdit class are listed here.

Published Events

	Name	Description
	OnChange (see page 4738)	Occurs when the text for the edit control may have changed.
	OnClick (see page 4738)	Occurs when the user clicks the control.
	OnContextPopup (see page 4740)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 4741)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 4741)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4743)	Occurs when the user drags an object over a control.
	OnEndDock (see page 4746)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4746)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4747)	Occurs when a control receives the input focus.
	OnExit (see page 4749)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 4751)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 4754)	Occurs when a key is pressed.
	OnKeyUp (see page 4755)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 4757)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4758)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4760)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4760)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4760)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4763)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 4765)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 4765)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 4766)	Occurs when the mouse wheel is rotated upward.
	OnProtectChange (see page 4766)	Occurs when a user attempts to modify text that is marked as protected.

	OnResizeRequest (see page 4766)	Occurs when the text has grown either smaller or larger than the control's window size.
	OnSaveClipboard (see page 4767)	Occurs when the window of the edit control is about to be destroyed and the edit control has delayed rendering of text copied to the clipboard..
	OnSelectionChange (see page 4767)	Occurs when the current selection changes.
	OnStartDock (see page 4768)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4768)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

27.1.44.3.1 TRichEdit.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change (see page 21245)

Text (see page 5842)

27.1.44.3.2 TRichEdit.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.

- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
```

```
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

27.1.44.3.3 TRichEdit.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.44.3.4 TRichEdit.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

27.1.44.3.5 TRichEdit.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
```

```
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:
```

```
procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

27.1.44.3.6 TRichEdit.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler

is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;
```

```
procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.44.3.7 TRichEdit.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.44.3.8 TRichEdit.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

27.1.44.3.9 TRichEdit.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
```

```
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

27.1.44.3.10 TRichEdit.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

27.1.44.3.11 TRichEdit.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not

generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
```

```
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;
```

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

27.1.44.3.12 TRichEdit.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

27.1.44.3.13 TRichEdit.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

```
void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

27.1.44.3.14 TRichEdit.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.44.3.15 TRichEdit.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
```

```
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
```

```
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.44.3.16 TRichEdit.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

27.1.44.3.17 TRichEdit.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

27.1.44.3.18 TRichEdit.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
```

```
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.44.3.19 TRichEdit.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
```

```
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
```

```
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

27.1.44.3.20 TRichEdit.OnMouseWheel Event

Occurs when the mouse wheel is rotated.

Pascal

```
property OnMouseWheel: TMouseWheelEvent;
```

C++

```
__property TMouseWheelEvent OnMouseWheel;
```

Description

Write code in the OnMouseWheel event handler to respond to mouse wheel messages. If there is no OnMouseWheel event handler, or if the mouse wheel message is not handled in the OnMouseWheel event handler, then an OnMouseWheelUp or OnMouseWheelDown event occurs, depending on the direction the mouse wheel was rotated.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheelUp (see page 5870)

27.1.44.3.21 TRichEdit.OnMouseWheelDown Event

Occurs when the mouse wheel is rotated downward.

Pascal

```
property OnMouseWheelDown: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelDown;
```

Description

Write code in the OnMouseWheelDown event handler to perform actions when the mouse wheel is rotated downward. OnMouseWheelDown occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheel (see page 5869)

OnMouseWheelUp (see page 5870)

27.1.44.3.22 TRichEdit.OnMouseWheelUp Event

Occurs when the mouse wheel is rotated upward.

Pascal

```
property OnMouseWheelUp: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelUp;
```

Description

Write code in the OnMouseWheelUp event handler to perform actions when the mouse wheel is rotated upward. OnMouseWheelUp occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheel (see page 5869)

27.1.44.3.23 TRichEdit.OnProtectChange Event

Occurs when a user attempts to modify text that is marked as protected.

Pascal

```
property OnProtectChange: TRichEditProtectChange;
```

C++

```
__property TRichEditProtectChange OnProtectChange;
```

Description

Write an OnProtectChange event handler to allow modification of text with the protected attribute. The StartPos and EndPos parameters indicate the beginning and end of the range of text that would be affected by the modification, where 0 is the first character in the text. To allow modification of protected text, set AllowChange to true.

To set or unset the protected attribute for a section of text, select the text and use the Protected property of the rich edit control's SelAttributes (see page 3887) property. To cause text to have the protected attribute by default, use the Protected property of the rich edit object's DefAttributes (see page 3877) property.

See Also

DefAttributes (see page 3877)

SelAttributes (see page 3887)

SelLength

SelStart

27.1.44.3.24 TRichEdit.OnResizeRequest Event

Occurs when the text has grown either smaller or larger than the control's window size.

Pascal

```
property OnResizeRequest: TRichEditResizeEvent;
```

C++

```
__property TRichEditResizeEvent OnResizeRequest;
```

Description

Write an OnResizeRequest event handler to respond to notifications that the edit control is trying to resize due to changes in the text. The Rect parameter gives the desired bounding rectangle that will display all the text, in screen coordinates.

See Also

BoundsRect (see page 5803)

ClientRect

Text (see page 5842)

27.1.44.3.25 TRichEdit.OnSaveClipboard Event

Occurs when the window of the edit control is about to be destroyed and the edit control has delayed rendering of text copied to the clipboard..

Pascal

```
property OnSaveClipboard: TRichEditSaveClipboard;
```

C++

```
__property TRichEditSaveClipboard OnSaveClipboard;
```

Description

Write an OnSaveClipboard event handler to respond to notifications that the window of the edit control is about to be destroyed and the Clipboard needs information from the edit control. The parameter NumObjects is the number of objects on the Clipboard, and NumChars is the number of characters on the Clipboard. To clear all information about the edit control from the Clipboard, set SaveClipboard to false.

See Also

CopyToClipboard (see page 21246)

CutToClipboard (see page 21249)

27.1.44.3.26 TRichEdit.OnSelectionChange Event

Occurs when the current selection changes.

Pascal

```
property OnSelectionChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSelectionChange;
```

Description

Write an OnSelectionChange event handler to respond to changes in the selected text. Changes in selection can occur when the user changes the selection using the mouse or keyboard, or when the selection is changed through the SelText property. Portions of the selection can be changed through the Lines (see page 4717) property.

See Also

AutoSelect (see page 21260)

ClearSelection (see page 21245)

Lines (see page 4717)

SelAttributes (see page 3887)

SelectAll (see page 21255)

SelLength

SelStart

SelText

SetSelTextBuf (see page 21258)

27.1.44.3.27 TRichEdit.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

27.1.44.3.28 TRichEdit.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

27.1.45 TStatusBar Class

TStatusBar represents a status bar.

Class Hierarchy

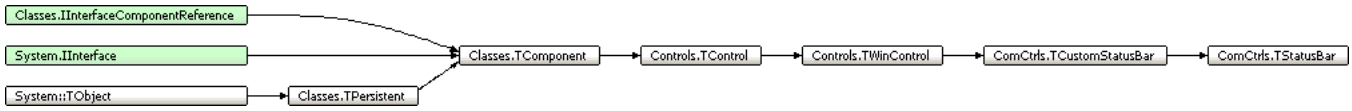

Pascal

```
TStatusBar = class(TCustomStatusBar);
```

C++

```
class TStatusBar : public TCustomStatusBar;
```

File

ComCtrls

Description

The TStatusBar component is a row of panels, usually aligned at the bottom of a form, that display information about an application as it runs. Each panel is represented by a TStatusPanel (see page 4825) object listed in the Panels (see page 4792) property. The SimplePanel (see page 4800) property can be used to toggle the status bar at runtime between a single- and multiple-panel display.

See Also

TStatusPanel (see page 4825)

TStatusPanels (see page 4834)

Status Bars

Display Controls

27.1.45.1 TStatusBar Members

The following tables list the members exposed by TStatusBar.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.

	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 3904)	Repositions and resizes the status bar by a specified ratio.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreatePanel (see page 3905)	Creates a new panel that can be added to the Panels (see page 3912) property.
	CreatePanels (see page 3905)	Creates the TStatusPanels (see page 4834) object that implements the Panels (see page 3912) property.
	CreateParams (see page 3905)	Initializes the window-creation parameter record when the status bar window is created.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 3906)	Creates the Windows control that represents the status bar.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.

	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoHint (see page 3906)	Generates an OnHint (see page 3920) event.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawPanel (see page 3907)	Draws an owner-drawn panel.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
V	GetPanelClass (see page 3908)	Returns the class to use for implementing the panels of the status bar.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsFontStored (see page 3908)	Indicates whether the value of the Font property should be saved in the form file.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.

	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 3909)	Sets the value of the Parent property.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.

	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 3903)	Creates and initializes a TCustomStatusBar (see page 3892) component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 3903)	Destroys TCustomStatusBar (see page 3892) and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteAction (see page 3907)	Processes actions for the status bar.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
V	FlipChildren (see page 3908)	Reverses the positions of the status bar's panels.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.

	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.

	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 3909)	Sets the Left, Top, Width, and Height properties all at once.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 3910)	Provides runtime access to the status bar's drawing surface.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.

R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 4804)	Occurs when the user clicks the control.
	OnContextPopup (see page 4806)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreatePanelClass (see page 4806)	Occurs when a status panel needs to be created.
	OnDblClick (see page 4807)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 4807)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4809)	Occurs when the user drags an object over a control.
	OnDrawPanel (see page 4812)	Occurs when a status panel needs to be redrawn.
	OnEndDock (see page 4813)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4813)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnHint (see page 4814)	Occurs before the status bar's hint is displayed.
	OnMouseActivate (see page 4814)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4815)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4818)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4818)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4818)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4820)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 4823)	Occurs immediately after the control is resized.
	OnStartDock (see page 4823)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4824)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Action (see page 4781)	Designates the action associated with the control.
	Align (see page 4781)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 4785)	Specifies how the control is anchored to its parent.
	AutoHint (see page 4786)	Specifies whether the status bar's text is set automatically to the current hint.
	BiDiMode (see page 4786)	Specifies the bi-directional mode for the control.
	BorderWidth (see page 4786)	Specifies the width of the control's border.
	Color (see page 4787)	Specifies the background color of the control.
	Constraints (see page 4787)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 4788)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4788)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4789)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4790)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4791)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Panels (see page 4792)	Lists the panels (TStatusPanel (see page 4825)) in the status bar.
	ParentBiDiMode (see page 4792)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 4793)	Determines where a control looks for its color information.
	ParentFont (see page 4793)	Determines where a control looks for its font information.
	ParentShowHint (see page 4794)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4794)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 4799)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SimplePanel (see page 4800)	Determines whether the status bar displays a single panel or multiple panels.
	SimpleText (see page 4801)	Contains the string that is displayed in the status bar when SimplePanel (see page 4800) is set to true.
	SizeGrip (see page 4801)	Determines whether the status bar is resizable at runtime.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	UseSystemFont (see page 4801)	Specifies whether the status bar uses the system font.
	Visible (see page 4802)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.45.2 TStatusBar Properties

The properties of the TStatusBar class are listed here.

Published Properties

	Name	Description
	Action (see page 4781)	Designates the action associated with the control.
	Align (see page 4781)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4785)	Specifies how the control is anchored to its parent.
	AutoHint (see page 4786)	Specifies whether the status bar's text is set automatically to the current hint.
	BiDiMode (see page 4786)	Specifies the bi-directional mode for the control.
	BorderWidth (see page 4786)	Specifies the width of the control's border.
	Color (see page 4787)	Specifies the background color of the control.
	Constraints (see page 4787)	Specifies the size constraints for the control.
	DragCursor (see page 4788)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4788)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4789)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4790)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4791)	Controls the attributes of text written on or in the control.
	Panels (see page 4792)	Lists the panels (TStatusPanel (see page 4825)) in the status bar.
	ParentBiDiMode (see page 4792)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 4793)	Determines where a control looks for its color information.
	ParentFont (see page 4793)	Determines where a control looks for its font information.
	ParentShowHint (see page 4794)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4794)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 4799)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SimplePanel (see page 4800)	Determines whether the status bar displays a single panel or multiple panels.
	SimpleText (see page 4801)	Contains the string that is displayed in the status bar when SimplePanel (see page 4800) is set to true.
	SizeGrip (see page 4801)	Determines whether the status bar is resizable at runtime.
	UseSystemFont (see page 4801)	Specifies whether the status bar uses the system font.
	Visible (see page 4802)	Determines whether the component appears onscreen.

27.1.45.2.1 TStatusBar.Action Property

Designates the action associated with the control.

Pascal

```
property Action: TBasicAction;
```

C++

```
__property TBasicAction Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

27.1.45.2.2 TStatusBar.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;
```

```
  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
```

```
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
```

```
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

27.1.45.2.3 TStatusBar.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.45.2.4 TStatusBar.AutoHint Property

Specifies whether the status bar's text is set automatically to the current hint.

Pascal

```
property AutoHint: Boolean;
```

C++

```
__property Boolean AutoHint;
```

Description

Use AutoHint to specify whether the status bar's text is set automatically to the current hint.

27.1.45.2.5 TStatusBar.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

27.1.45.2.6 TStatusBar.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the status bar's border. This is the area within the edge of the control and outside panel bevels. Text and graphics are clipped to the area within the border.

27.1.45.2.7 TStatusBar.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

27.1.45.2.8 TStatusBar.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.45.2.9 TStatusBar.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

27.1.45.2.10 TStatusBar.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

27.1.45.2.11 TStatusBar.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

27.1.45.2.12 TStatusBar.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls

appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.45.2.13 TStatusBar.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
```

```
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

27.1.45.2.14 TStatusBar.Panels Property

Lists the panels (TStatusPanel (see page 4825)) in the status bar.

Pascal

```
property Panels: TStatusPanels;
```

C++

```
__property TStatusPanels Panels;
```

Description

The Panels property holds a TStatusPanels (see page 4834)—that is, a collection of TStatusPanel (see page 4825) objects. At design time, you can add, remove, or modify panels with the Panels editor. To open the Panels editor, select the Panels property in the Object Inspector, then double-click in the Value column to the right or click the ellipsis (...) button.

See Also

TStatusPanel (see page 4825)

27.1.45.2.15 TStatusBar.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

27.1.45.2.16 TStatusBar.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

27.1.45.2.17 TStatusBar.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

27.1.45.2.18 TStatusBar.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.45.2.19 TStatusBar.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
```

```
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
```

```
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
```

```
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
```

```
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
```

```
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.45.2.20 TStatusBar.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
```

```
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.45.2.21 TStatusBar.SimplePanel Property

Determines whether the status bar displays a single panel or multiple panels.

Pascal

```
property SimplePanel: Boolean;
```

C++

```
__property Boolean SimplePanel;
```

Description

If SimplePanel is set to true, the status bar consists of a single panel displaying the text in SimpleText (see page 4801). If SimplePanel is set to false, the status bar displays a separate panel for each item in its Panels (see page 4792) property..

See Also

SimpleText (see page 4801)

Panels (see page 4792)

27.1.45.2.22 TStatusBar.SimpleText Property

Contains the string that is displayed in the status bar when SimplePanel (see page 4800) is set to true.

Pascal

```
property SimpleText: string;
```

C++

```
__property AnsiString SimpleText;
```

Description

SimpleText contains a text string to be displayed in the status panel when SimplePanel (see page 4800) is true.

See Also

SimplePanel (see page 4800)

27.1.45.2.23 TStatusBar.SizeGrip Property

Determines whether the status bar is resizable at runtime.

Pascal

```
property SizeGrip: Boolean;
```

C++

```
__property Boolean SizeGrip;
```

Description

If SizeGrip is set to true, the status bar has a triangular grip on the lower right corner. The user can resize the status bar by dragging the grip with the mouse.

A size grip will not be drawn unless the TStatusBar (see page 4769) is the child of a TCustomForm with a BorderStyle of bsSizeable or bsSizeToolWin.

Note: If Align (see page 4781) is set to alBottom, alRight, or alClient, the sizing grip will not work. If the Alignment property of the last panel in the status bar is set to taRightJustify, the sizing grip will truncate the panel's text.

See Also

Align (see page 5798)

Alignment (see page 4829)

27.1.45.2.24 TStatusBar.UseSystemFont Property

Specifies whether the status bar uses the system font.

Pascal

```
property UseSystemFont: Boolean;
```

C++

```
__property Boolean UseSystemFont;
```

Description

UseSystemFont specifies whether the status bar uses the system font. To change fonts, use the Font property.

See Also

Font (see page 5823)

ParentFont (see page 5834)

DesktopFont (see page 5818)

27.1.45.2.25 TStatusBar.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
```

```
  edit control placed somewhere on the client area of the
  sheet.  Unchecking the checkbox will cause the client area
  (not the tab area) of the Tab Control to become invisible.
  }
  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    PageControl1.ActivePage.Visible := CheckBox1.Checked;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    CheckBox1.Checked := PageControl1.ActivePage.Visible;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.45.3 TStatusBar Events

The events of the TStatusBar class are listed here.

Published Events

	Name	Description
	OnClick (see page 4804)	Occurs when the user clicks the control.
	OnContextPopup (see page 4806)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreatePanelClass (see page 4806)	Occurs when a status panel needs to be created.
	OnDblClick (see page 4807)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 4807)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4809)	Occurs when the user drags an object over a control.
	OnDrawPanel (see page 4812)	Occurs when a status panel needs to be redrawn.
	OnEndDock (see page 4813)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4813)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

	OnHint (see page 4814)	Occurs before the status bar's hint is displayed.
	OnMouseActivate (see page 4814)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4815)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4818)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4818)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4818)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4820)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 4823)	Occurs immediately after the control is resized.
	OnStartDock (see page 4823)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4824)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

27.1.45.3.1 TStatusBar.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
```

```
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

27.1.45.3.2 TStatusBar.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.45.3.3 TStatusBar.OnCreatePanelClass Event

Occurs when a status panel needs to be created.

Pascal

```
property OnCreatePanelClass: TSBCreatePanelClassEvent;
```

C++

```
__property TSBCreatePanelClassEvent OnCreatePanelClass;
```

Description

Write an OnCreatePanelClass event handler to specify a custom TStatusPanel descendant that the status bar should use to implement its panels.

See Also

Panels (see page 3912)

TStatusPanel (see page 4825)

27.1.45.3.4 TStatusBar.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

27.1.45.3.5 TStatusBar.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
```

```
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

27.1.45.3.6 TStatusBar.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
```

```
    TForm1(Source).Top := Y;
  end;

  procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code comes from an application that contains a list box
  and three labels, each with a different font and color.  The
  DragMode property for each of the labels is dmAutomatic.
  The user can select a label and drag it to a list box and
  drop it. When the label is dropped, the items in the list
  box assume the color and font of the dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.45.3.7 TStatusBar.OnDrawPanel Event

Occurs when a status panel needs to be redrawn.

Pascal

```
property OnDrawPanel: TDrawPanelEvent;
```

C++

```
__property TDrawPanelEvent OnDrawPanel;
```

Description

The OnDrawPanel event occurs when a status panel needs to be redisplayed—for example, when the user resizes the status bar. The Rect parameter gives the (new) dimensions of the status panel. OnDrawPanel occurs only if the status panel's Style property is set to psOwnerDraw.

C++ Examples:

```
/*
The following code is the OnDrawPanel event handler of a
status bar.  It draws each panel of the status bar, adding
text and icons stored in ImageList1. The image list contains
as many icons as there are header sections.  This example
requires a populated image list and a statusbar with several
panels added to the Panels property.  Select each panel and
set the Style property to psOwnerDraw.
*/
void __fastcall TForm1::StatusBar1DrawPanel(TStatusBar *StatusBar,
        TStatusPanel *Panel, const TRect &Rect)
{
  TCanvas *canvas = StatusBar->Canvas;
  canvas->Brush->Color = clRed;
  canvas->FillRect(Rect);
  canvas->Font->Color = clYellow;
  ImageList1->Draw(canvas,Rect.Left,Rect.Top, Panel->Index, true);
  canvas->TextOut(Rect.left + 30, Rect.top + 2, "Panel" + IntToStr(Panel->Index));
}
```

Delphi Examples:

```
{
The following code is the OnDrawPanel event handler of a
status bar.  It draws each panel of the status bar, adding
text and icons stored in ImageList1. The image list contains
as many icons as there are header sections.  This example
requires a populated image list and a statusbar with several
panels added to the Panels property.  Select each panel and
set the Style property to psOwnerDraw.
}
procedure TForm1.StatusBar1DrawPanel(StatusBar: TStatusBar;
  Panel: TStatusPanel; const Rect: TRect);
begin
with StatusBar1.Canvas do
  begin
    Brush.Color := clRed;
```

```
      FillRect(Rect);
      Font.Color := clYellow;
      ImageList1.Draw(StatusBar1.Canvas,Rect.Left,Rect.Top,Panel.Index);
      TextOut(Rect.left + 30, Rect.top + 2, 'Panel' + IntToStr(Panel.Index));
    end;
  end;
```

See Also

Style (see page 4830)

27.1.45.3.8 TStatusBar.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.45.3.9 TStatusBar.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
```

```
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

27.1.45.3.10 TStatusBar.OnHint Event

Occurs before the status bar's hint is displayed.

Pascal

```
property OnHint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnHint;
```

Description

Write code in the OnHint event handler to customize the hint text.

27.1.45.3.11 TStatusBar.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.45.3.12 TStatusBar.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse

button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
```

```
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.45.3.13 TStatusBar.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

27.1.45.3.14 TStatusBar.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

27.1.45.3.15 TStatusBar.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
```

```
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.45.3.16 TStatusBar.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
```

```
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

27.1.45.3.17 TStatusBar.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

27.1.45.3.18 TStatusBar.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

27.1.45.3.19 TStatusBar.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

27.1.46 TStatusPanel Class

TStatusPanel represents a single panel of a status bar.

Class Hierarchy

Pascal

```
TStatusPanel = class(TCollectionItem);
```

C++

```
class TStatusPanel : public TCollectionItem;
```

File

ComCtrls

Description

Each TStatusPanel object is a member of a TStatusPanels (see page 4834) collection and represents a panel in a TStatusBar (see page 4769) control.

See Also

TCollection (see page 2731)

TStatusPanels (see page 4834)

TStatusBar (see page 4769)

27.1.46.1 TStatusPanel Members

The following tables list the members exposed by TStatusPanel.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 4828)	Returns the display name of the panel.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 4827)	Copies the contents of the source status panel to a new status panel.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 4827)	Creates and initializes a status panel.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentBiDiModeChanged (see page 4828)	Synchronizes the status panel's BiDiMode (see page 4830) with its parent's BiDiMode (see page 4830)..
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	UseRightToLeftAlignment (see page 4828)	Specifies whether the control's alignment is in a right-to-left mode.
	UseRightToLeftReading (see page 4828)	Specifies whether the control is using a right-to-left reading order.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	Alignment (see page 4829)	Specifies how text is aligned within the status panel.
	Bevel (see page 4829)	Determines the style of the panel's border.
	BiDiMode (see page 4830)	Specifies the bi-directional mode for the status panel.
	ParentBiDiMode (see page 4830)	Specifies whether the status panel uses its parent's bi-directional mode.
	Style (see page 4830)	Determines how the status panel's text is displayed.
	Text (see page 4831)	Specifies the text appearing in the status panel.
	Width (see page 4834)	Specifies the width of the status panel.

27.1.46.2 TStatusPanel.Create Constructor

Creates and initializes a status panel.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TStatusPanel(TCollection Collection);
```

Description

Create takes a TStatusPanels (see page 4834) instance as its argument.

See Also

TStatusPanels (see page 4834)

Add (see page 4837)

27.1.46.3 TStatusPanel Methods

The methods of the TStatusPanel class are listed here.

Protected Methods

	Name	Description
	GetDisplayName (see page 4828)	Returns the display name of the panel.

Public Methods

	Name	Description
	Assign (see page 4827)	Copies the contents of the source status panel to a new status panel.
	ParentBiDiModeChanged (see page 4828)	Synchronizes the status panel's BiDiMode (see page 4830) with its parent's BiDiMode (see page 4830)..
	UseRightToLeftAlignment (see page 4828)	Specifies whether the control's alignment is in a right-to-left mode.
	UseRightToLeftReading (see page 4828)	Specifies whether the control is using a right-to-left reading order.

27.1.46.3.1 TStatusPanel.Assign Method

Copies the contents of the source status panel to a new status panel.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Assign copies the values of Alignment (see page 4829), Bevel (see page 4829), Style (see page 4830), Text (see page 4831), and Width (see page 4834) from the source status panel.

See Also

Assign (see page 2930)

27.1.46.3.2 TStatusPanel.GetDisplayName Method

Returns the display name of the panel.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

Use GetDisplayName to return the display name of the panel.

If Text (see page 4831) is set, GetDisplayName overrides the inherited method to return Text (see page 4831); Otherwise, GetDisplayName returns the result of a call to the inherited GetDisplayName method.

27.1.46.3.3 TStatusPanel.ParentBiDiModeChanged Method

Synchronizes the status panel's BiDiMode (see page 4830) with its parent's BiDiMode (see page 4830)..

Pascal

```
procedure ParentBiDiModeChanged;
```

C++

```
__fastcall ParentBiDiModeChanged();
```

Description

Use ParentBiDiModeChanged to synchronize the status panel's BiDiMode (see page 4830) with its parent's BiDiMode (see page 4830).

27.1.46.3.4 TStatusPanel.UseRightToLeftAlignment Method

Specifies whether the control's alignment is in a right-to-left mode.

Pascal

```
function UseRightToLeftAlignment: Boolean;
```

C++

```
__fastcall Boolean UseRightToLeftAlignment();
```

Description

Call UseRightToLeftAlignment to determine whether the control's alignment is in a right-to-left mode. This information can be used when painting the control, or when responding to mouse events.

27.1.46.3.5 TStatusPanel.UseRightToLeftReading Method

Specifies whether the control is using a right-to-left reading order.

Pascal

```
function UseRightToLeftReading: Boolean;
```

C++

```
__fastcall Boolean UseRightToLeftReading();
```

Description

Call UseRightToLeftReading to determine which style flags to set for the control when it is created and which flags to set for drawing text (for example, when calling ExTextOut or DrawText).

27.1.46.4 TStatusPanel Properties

The properties of the TStatusPanel class are listed here.

Published Properties

	Name	Description
	Alignment (see page 4829)	Specifies how text is aligned within the status panel.
	Bevel (see page 4829)	Determines the style of the panel's border.
	BiDiMode (see page 4830)	Specifies the bi-directional mode for the status panel.
	ParentBiDiMode (see page 4830)	Specifies whether the status panel uses its parent's bi-directional mode.
	Style (see page 4830)	Determines how the status panel's text is displayed.
	Text (see page 4831)	Specifies the text appearing in the status panel.
	Width (see page 4834)	Specifies the width of the status panel.

27.1.46.4.1 TStatusPanel.Alignment Property

Specifies how text is aligned within the status panel.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to indicate panel text alignment. These are the possible values:

Value	Meaning
taLeftJustify	Align text on the left side of the panel.
taCenter	Center the text in the panel.
taRightJustify	Align text on the right side of the panel.

Note: If the status bar has a sizing grip, setting the Alignment of the last status panel to taRightJustify truncates the panel's text.

See Also

Text (see page 4831)

Style (see page 4830)

SizeGrip (see page 4801)

27.1.46.4.2 TStatusPanel.Bevel Property

Determines the style of the panel's border.

Pascal

```
property Bevel: TStatusPanelBevel;
```

C++

```
__property TStatusPanelBevel Bevel;
```

Description

These are the possible values of Bevel:

Value	Meaning
pbNone	The panel has no bevel and appears flat.
pbLowered	The panel appears recessed.
pbRaised	The panel appears raised.

27.1.46.4.3 TStatusPanel.BiDiMode Property

Specifies the bi-directional mode for the status panel.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to specify the bi-directional mode for the status panel.

27.1.46.4.4 TStatusPanel.ParentBiDiMode Property

Specifies whether the status panel uses its parent's bi-directional mode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use BiDiMode (see page 4830) to specify whether the status panel uses its parent's bi-directional mode.

27.1.46.4.5 TStatusPanel.Style Property

Determines how the status panel's text is displayed.

Pascal

```
property Style: TStatusPanelStyle;
```

C++

```
__property TStatusPanelStyle Style;
```

Description

If Style is set to psText (the default), the string contained in the Text (see page 4831) property is displayed in the status panel, using the alignment specified by Alignment (see page 4829). The font is determined by the status bar's Font property.

If Style is set to psOwnerDraw, the content displayed in the status panel is drawn at runtime on the status bar's canvas by code

in an OnDrawPanel event handler.

C++ Examples:

```
/*
The following code is the OnDrawPanel event handler of a
status bar.  It draws each panel of the status bar, adding
text and icons stored in ImageList1. The image list contains
as many icons as there are header sections.  This example
requires a populated image list and a statusbar with several
panels added to the Panels property.  Select each panel and
set the Style property to psOwnerDraw.
*/
void __fastcall TForm1::StatusBar1DrawPanel(TStatusBar *StatusBar,
        TStatusPanel *Panel, const TRect &Rect)
{
  TCanvas *canvas = StatusBar->Canvas;
  canvas->Brush->Color = clRed;
  canvas->FillRect(Rect);
  canvas->Font->Color = clYellow;
  ImageList1->Draw(canvas,Rect.Left,Rect.Top, Panel->Index, true);
  canvas->TextOut(Rect.left + 30, Rect.top + 2, "Panel" + IntToStr(Panel->Index));
}
```

Delphi Examples:

```
{
The following code is the OnDrawPanel event handler of a
status bar.  It draws each panel of the status bar, adding
text and icons stored in ImageList1. The image list contains
as many icons as there are header sections.  This example
requires a populated image list and a statusbar with several
panels added to the Panels property.  Select each panel and
set the Style property to psOwnerDraw.
}
procedure TForm1.StatusBar1DrawPanel(StatusBar: TStatusBar;
  Panel: TStatusPanel; const Rect: TRect);
begin
with StatusBar1.Canvas do
  begin
    Brush.Color := clRed;
    FillRect(Rect);
    Font.Color := clYellow;
    ImageList1.Draw(StatusBar1.Canvas,Rect.Left,Rect.Top,Panel.Index);
    TextOut(Rect.left + 30, Rect.top + 2, 'Panel' + IntToStr(Panel.Index));
  end;
end;
```

See Also

Text (see page 4831)

Alignment (see page 4829)

Font (see page 5823)

Canvas (see page 3910)

OnDrawPanel (see page 4812)

27.1.46.4.6 TStatusPanel.Text Property

Specifies the text appearing in the status panel.

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

The Text property contains a string that appears in the status panel if Style (see page 4830) is set to psText.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
```

```
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

27.1.46.4.7 TStatusPanel.Width Property

Specifies the width of the status panel.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

The Width property determines the horizontal size of the status panel, in pixels.

27.1.47 TStatusPanels Class

TStatusPanels is a container for TStatusPanel (see page 4825) objects.

Class Hierarchy

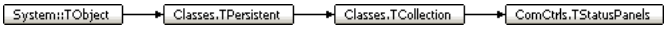

Pascal

```
TStatusPanels = class(TCollection);
```

C++

```
class TStatusPanels : public TCollection;
```

File

ComCtrls

Description

Each TStatusPanels holds a collection of TStatusPanel (see page 4825) objects in a TStatusBar (see page 4769). TStatusPanels maintains an index of the status panels in its Items (see page 4840) array. The Count property contains the number of status panels in the collection. At design time, use the status bar's Panels editor to add, remove, or modify status panels.

See Also

TCollection (see page 2731)

TCollectionItem (see page 2755)

TStatusPanel (see page 4825)

TStatusBar (see page 4769)

TStatusBar (see page 4769)

27.1.47.1 TStatusPanels Members

The following tables list the members exposed by TStatusPanels.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 4839)	Returns the owner.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 4839)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 4837)	Creates a new TStatusPanel (see page 4825) instance and adds it to the Items (see page 4840) array.
	AddItem (see page 4838)	Add (see page 4837) node object to collection.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 4836)	Creates and initializes a TStatusPanels (see page 4834) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 4839)	Add (see page 4837) new panel object to collection.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 4840)	Lists the status panels in the collection.

27.1.47.2 TStatusPanels.Create Constructor

Creates and initializes a TStatusPanels (see page 4834) object.

Pascal

```
constructor Create(StatusBar: TCustomStatusBar);
```

C++

```
__fastcall TStatusPanels(TCustomStatusBar StatusBar);
```

Description

Do not call Create directly. Instances of TStatusBar (see page 4769) create the TStatusPanels (see page 4834) object from their own constructors.

The Create method takes, as a parameter, the TStatusBar (see page 4769) instance that uses TStatusPanels (see page 4834) to manage its panels.

See Also

TStatusBar (see page 4769)

27.1.47.3 TStatusPanels Methods

The methods of the TStatusPanels class are listed here.

Protected Methods

	Name	Description
	GetOwner (see page 4839)	Returns the owner.
	Update (see page 4839)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 4837)	Creates a new TStatusPanel (see page 4825) instance and adds it to the Items (see page 4840) array.
	AddItem (see page 4838)	Add (see page 4837) node object to collection.
	Insert (see page 4839)	Add (see page 4837) new panel object to collection.

27.1.47.3.1 TStatusPanels.Add Method

Creates a new TStatusPanel (see page 4825) instance and adds it to the Items (see page 4840) array.

Pascal

```
function Add: TStatusPanel;
```

C++

```
__fastcall TStatusPanel Add();
```

Description

Add returns the new status panel. At design time, use the Panels editor to add panels to the status bar.

C++ Examples:

```
/*
The following example adds a panel to the status bar control
when the user clicks the button and adds a caption to the
panel. The code uses BeginUpdate and EndUpdate to prevent
repaints until the operation is complete.  A try...finally
block ensures EndUpdate is called even when an exception
occurs.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int PanelIndex;

  StatusBar1->Panels->BeginUpdate();
  PanelIndex = StatusBar1->Panels->Count - 1;
  try
  {
    StatusBar1->Panels->Add();
    PanelIndex++;
    StatusBar1->Panels->Items[PanelIndex]->Text = "Panel" + IntToStr(PanelIndex);
  }
  __finally
  {
    StatusBar1->Panels->EndUpdate();
  }
}
```

Delphi Examples:

```
{
The following example adds a panel to the status bar control
when the user clicks the button and adds a caption to the
panel. The code uses BeginUpdate and EndUpdate to prevent
repaints until the operation is complete.  A try...finally
block ensures EndUpdate is called even when an exception
```

```
occurs.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  PanelIndex : Integer;
begin
  with StatusBar1 do
  begin
    Panels.BeginUpdate;
    PanelIndex := StatusBar1.Panels.Count - 1;
    try
      Panels.Add;
      Inc(PanelIndex);
      Panels.Items[PanelIndex].Text :=
        'Panel' + IntToStr(PanelIndex);
    finally
      Panels.EndUpdate;
    end;
  end;
end;
```

See Also

Delete (see page 2741)

Items (see page 4840)

Panels (see page 4792)

AddItem (see page 4838)

27.1.47.3.2 TStatusPanels.AddItem Method

Add (see page 4837) node object to collection.

Pascal

```
function AddItem(Item: TStatusPanel; Index: Integer): TStatusPanel;
```

C++

```
__fastcall TStatusPanel AddItem(TStatusPanel Item, int Index);
```

Description

The AddItem method is similar to the Add (see page 4837) method, but is more flexible.

The Item parameter specifies the panel object to add. If Item is nil, then a new panel object is created automatically. To determine the object type at runtime, define the status bar's OnCreatePanelClass event handler. If this event handler is not defined, the new panel object is an instance of TStatusPanel (see page 4825).

The Index parameter determines the Index property of the panel object. If Index is negative, the panel object is added at the end of the collection.

AddItem returns the panel object added.

See Also

Add (see page 4837)

Insert (see page 4839)

OnCreatePanelClass (see page 3919)

27.1.47.3.3 TStatusPanels.GetOwner Method

Returns the owner.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

In TStatusPanels (see page 4834), GetOwner returns the TStatusBar (see page 4769) object that contains the panels.

See Also

GetOwner (see page 2932)

27.1.47.3.4 TStatusPanels.Insert Method

Add (see page 4837) new panel object to collection.

Pascal

```
function Insert(Index: Integer): TStatusPanel;
```

C++

```
__fastcall TStatusPanel Insert(int Index);
```

Description

The Insert method creates a new panel object and adds it the the collection. To determine the object type at runtime, define the status bar's OnCreatePanelClass event handler. If this event handler is not defined, the new panel object is an instance of TStatusPanel (see page 4825).

The Index parameter determines the Index property of the panel object. If Index is negative, the panel object is added at the end of the collection.

Insert returns the new panel object.

See Also

Add (see page 4837)

AddItem (see page 4838)

27.1.47.3.5 TStatusPanels.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

In TStatusPanels (see page 4834), a call to Update for any panel causes the entire status bar layout to be recalculated.

See Also

Update (see page 2747)

27.1.47.4 TStatusPanels Properties

The properties of the TStatusPanels class are listed here.

Public Properties

	Name	Description
	Items (see page 4840)	Lists the status panels in the collection.

27.1.47.4.1 TStatusPanels.Items Property

Lists the status panels in the collection.

Pascal

```
property Items [Index: Integer]: TStatusPanel;
```

C++

```
__property TStatusPanel Items[int Index];
```

Description

The value of the Index parameter corresponds to the Index property of TStatusPanel (see page 4825). It represents the position of the status panel in the status bar.

C++ Examples:

```
/*
The following example adds a panel to the status bar control
when the user clicks the button and adds a caption to the
panel. The code uses BeginUpdate and EndUpdate to prevent
repaints until the operation is complete.  A try...finally
block ensures EndUpdate is called even when an exception
occurs.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int PanelIndex;

  StatusBar1->Panels->BeginUpdate();
  PanelIndex = StatusBar1->Panels->Count - 1;
  try
  {
    StatusBar1->Panels->Add();
    PanelIndex++;
    StatusBar1->Panels->Items[PanelIndex]->Text = "Panel" + IntToStr(PanelIndex);
  }
  __finally
  {
    StatusBar1->Panels->EndUpdate();
  }
}
```

Delphi Examples:

```
{
The following example adds a panel to the status bar control
when the user clicks the button and adds a caption to the
panel. The code uses BeginUpdate and EndUpdate to prevent
```

```
  repaints until the operation is complete.  A try...finally
  block ensures EndUpdate is called even when an exception
  occurs.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    PanelIndex : Integer;
  begin
    with StatusBar1 do
    begin
      Panels.BeginUpdate;
      PanelIndex := StatusBar1.Panels.Count - 1;
      try
        Panels.Add;
        Inc(PanelIndex);
        Panels.Items[PanelIndex].Text :=
          'Panel' + IntToStr(PanelIndex);
      finally
        Panels.EndUpdate;
      end;
    end;
  end;
```

See Also

Index (see page 2762)

operator_[]

27.1.48 TTabControl Class

TTabControl is a tab set that has the appearance of notebook dividers.

Class Hierarchy

Pascal

```
TTabControl = class(TCustomTabControl);
```

C++

```
class TTabControl : public TCustomTabControl;
```

File

ComCtrls

Description

Use TTabControl to add a control with multiple tab settings to a form. Unlike a page control, TTabControl is not made up of several pages that contain different controls. Instead, TTabControl is a single object. When the current tab changes, the tab control must directly update its contents to reflect the change in an OnChange (see page 4884) event handler.

Disabling a TTabControl object disables all tabs because it is one control. To disable individual tabs, use TPageControl (see page 4459) and disable the individual TTabSheet (see page 4919) controls that represent the pages.

Note: When using one of the page controls, if you want to restrict a user from switching to a tab, you cannot set TTabSheet.Enabled (see page 4862) to **false** to accomplish that restriction. Instead, use the OnChanging (see page 4888)

event to prevent the user from selecting a tab.

See Also

TPageControl (see page 4459)

Grouping Controls

Tab Controls

27.1.48.1 TTabControl Members

The following tables list the members exposed by TTabControl.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 3933)	Calculates the control's display area given a rectangle.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.

	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanChange (see page 3933)	Change (see page 3934) permission event dispatcher.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
V	CanShowTab (see page 3934)	Selection permission event dispatcher.
	Change (see page 3934)	Pending tab change event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 3935)	Sets up the window-creation parameters for the tab control.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 3935)	Creates the window for a tab control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.

	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawTab (see page 3935)	Tab drawing event dispatcher.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetImageIndex (see page 3936)	Generates an OnGetImageIndex (see page 3956) event
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.

	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 3937)	Form load event dispatcher.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 3937)	Event dispatcher for creation and destruction of child objects.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.

	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetTabIndex (see page 3938)	This is SetTabIndex, a member of class TCustomTabControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateTabImages (see page 3939)	Updates the tab images.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 3932)	Creates and initializes an instance of TCustomTabControl (see page 3921).
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 3932)	Destroys an instance of TCustomTabControl (see page 3921).
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHitTestInfoAt (see page 3936)	Returns information about the location of a point relative to the client area of the tab control.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	IndexOfTabAt (see page 3936)	Indicates the index of the tab at a specified point.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	RowCount (see page 3938)	Sets TabIndex (see page 3946) property.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.

	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollTabs (see page 3938)	Scrolls the tabs that are visible when the tab control is not multi-line.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	TabRect (see page 3939)	Returns the bounding rectangle for a specified tab.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 3940)	Gives access to the tab control's canvas.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayRect (see page 4859)	Specifies the bounding rectangle of a tab control client area.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.

	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 4884)	Occurs after a new tab is selected.
	OnChanging (see page 4888)	Occurs immediately before a new tab is selected.
	OnContextPopup (see page 4889)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDockDrop (see page 4890)	Occurs when another control is docked to the control.
	OnDockOver (see page 4892)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 4894)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4896)	Occurs when the user drags an object over a control.
	OnDrawTab (see page 4898)	Occurs when a tab is about to be drawn.
	OnEndDock (see page 4899)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4899)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4900)	Occurs when a control receives the input focus.
	OnExit (see page 4903)	Occurs when the input focus shifts away from one control to another.
	OnGetImageIndex (see page 4904)	Occurs when a tab is about to display its associated image.
	OnGetSiteInfo (see page 4904)	Returns the control's docking information.
	OnMouseActivate (see page 4907)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4908)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4910)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4910)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4910)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4913)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 4915)	Occurs immediately after the control is resized.

	OnStartDock (see page 4915)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4916)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 4917)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 4853)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 4857)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 4858)	Specifies the bi-directional mode for the control.
	Constraints (see page 4858)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DockSite (see page 4859)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 4860)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4860)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4861)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4862)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4863)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HotTrack (see page 4864)	Determines whether labels on the tab under the mouse are automatically highlighted.
	Images (see page 4864)	Specifies the images drawn in tabs.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	MultiLine (see page 4865)	Determines whether the tabs can appear on more than one row.
	MultiSelect (see page 4866)	Specifies whether multiple tabs can be selected.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OwnerDraw (see page 4866)	Specifies whether the tab control handles its own painting.
	ParentBiDiMode (see page 4866)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 4867)	Determines where a control looks for its font information.
	ParentShowHint (see page 4867)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4868)	Identifies the pop-up menu associated with the control.
	RaggedRight (see page 4873)	Specifies whether rows of tabs stretch to fill the width of the control.
	ScrollOpposite (see page 4873)	Determines how the rows of tabs are scrolled in a multi-line tab control.
	ShowHint (see page 4873)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 4875)	Specifies the style of the tab control.
	TabHeight (see page 4875)	Specifies the height, in pixels, of the tabs in the tab control.
	TabIndex (see page 4876)	Identifies the selected tab on a tab control.
	TabOrder (see page 4878)	Indicates the position of the control in its parent's tab order.
	TabPosition (see page 4879)	Determines whether tabs appear at the top or bottom.
	Tabs (see page 4879)	Contains the list of text strings that label the tabs of the tab control.
	TabStop (see page 4881)	Determines if the user can tab to a control.
	TabWidth (see page 4881)	Specifies the horizontal size, in pixels, of the tabs in the tab control.
	Tag (see page 2808)	Stores an integer value as part of a component.

	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 4882)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.48.2 TTabControl Properties

The properties of the TTabControl class are listed here.

Public Properties

	Name	Description
	DisplayRect (see page 4859)	Specifies the bounding rectangle of a tab control client area.

Published Properties

	Name	Description
	Align (see page 4853)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4857)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 4858)	Specifies the bi-directional mode for the control.
	Constraints (see page 4858)	Specifies the size constraints for the control.
	DockSite (see page 4859)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 4860)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4860)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4861)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4862)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4863)	Controls the attributes of text written on or in the control.
	HotTrack (see page 4864)	Determines whether labels on the tab under the mouse are automatically highlighted.
	Images (see page 4864)	Specifies the images drawn in tabs.
	MultiLine (see page 4865)	Determines whether the tabs can appear on more than one row.
	MultiSelect (see page 4866)	Specifies whether multiple tabs can be selected.
	OwnerDraw (see page 4866)	Specifies whether the tab control handles its own painting.
	ParentBiDiMode (see page 4866)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 4867)	Determines where a control looks for its font information.
	ParentShowHint (see page 4867)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4868)	Identifies the pop-up menu associated with the control.
	RaggedRight (see page 4873)	Specifies whether rows of tabs stretch to fill the width of the control.
	ScrollOpposite (see page 4873)	Determines how the rows of tabs are scrolled in a multi-line tab control.
	ShowHint (see page 4873)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 4875)	Specifies the style of the tab control.
	TabHeight (see page 4875)	Specifies the height, in pixels, of the tabs in the tab control.
	TabIndex (see page 4876)	Identifies the selected tab on a tab control.
	TabOrder (see page 4878)	Indicates the position of the control in its parent's tab order.
	TabPosition (see page 4879)	Determines whether tabs appear at the top or bottom.
	Tabs (see page 4879)	Contains the list of text strings that label the tabs of the tab control.
	TabStop (see page 4881)	Determines if the user can tab to a control.
	TabWidth (see page 4881)	Specifies the horizontal size, in pixels, of the tabs in the tab control.
	Visible (see page 4882)	Determines whether the component appears onscreen.

27.1.48.2.1 TTabControl.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;
```

```
  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
```

```
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
```

```
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

27.1.48.2.2 TTabControl.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.48.2.3 TTabControl.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

27.1.48.2.4 TTabControl.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.48.2.5 TTabControl.DisplayRect Property

Specifies the bounding rectangle of a tab control client area.

Pascal

```
property DisplayRect: TRect;
```

C++

```
__property TRect DisplayRect;
```

Description

Use DisplayRect to obtain the coordinates, in pixels, of the portion of the tab control that represents the contents of the tab pages. The appearance of this portion of the tab control does not change when different tabs are selected.

Use DisplayRect rather than the ClientRect property to determine the portion of the tab control that can contain other objects.

See Also

ClientRect

AdjustClientRect (see page 3933)

27.1.48.2.6 TTabControl.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite: Boolean;
```

C++

```
__property Boolean DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
```

```
    if ((DockClientCount == 2) && (NewTarget != this))        // Self?
      PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
    UpdateCaption(Client);
  }
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

27.1.48.2.7 TTabControl.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

27.1.48.2.8 TTabControl.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

27.1.48.2.9 TTabControl.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
```

```
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

27.1.48.2.10 TTabControl.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.48.2.11 TTabControl.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
```

```
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

27.1.48.2.12 TTabControl.HotTrack Property

Determines whether labels on the tab under the mouse are automatically highlighted.

Pascal

```
property HotTrack: Boolean;
```

C++

```
__property Boolean HotTrack;
```

Description

Set HotTrack to true to provide visual feedback to the user about which tab would be selected if the mouse button is clicked. HotTrack is especially useful when the tab control contains multiple rows of tabs.

See Also

MultiLine (see page 3942)

HotImages

27.1.48.2.13 TTabControl.Images Property

Specifies the images drawn in tabs.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Images specifies the images that are drawn in tabs. If DisabledImages is defined, those images are used instead for disabled tabs; otherwise, grayed versions of the images are used.

In the implementation of TCustomTabControl, the order of images in image list properties directly corresponds to the associated tabs in the control. In the descendant class, TPageControl, this behavior is replaced by support for individual TTabSheet objects, each with an ImageIndex property. In any case, the application programmer can define customized image displays by defining an OnGetImageIndex event handler.

See Also

OnGetImageIndex (see page 3956)

Pages (see page 4498)

ImageIndex (see page 4943)

DisabledImages

27.1.48.2.14 TTabControl.MultiLine Property

Determines whether the tabs can appear on more than one row.

Pascal

```
property MultiLine: Boolean;
```

C++

```
__property Boolean MultiLine;
```

Description

Use MultiLine to determine how the tabs are displayed. If MultiLine is true, the tabs are displayed on more than one row when the number of tabs exceeds the number that fits across the top of the tab control. How many rows is determined by how many tabs are in the tab control. If MultiLine is false, the tabs are displayed on one row only, and the user must scroll the displayed scroll arrows to view all the tabs.

C++ Examples:

```
/*
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
*/

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->MultiLine = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->Caption = "Stack Tabs?";
  for (int i = 0; i < 20; i++)
  {
    TTabSheet *pPage = new TTabSheet(PageControl1);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
  }
}
```

Delphi Examples:

```
{
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.MultiLine := CheckBox1.Checked;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Checkbox1.Caption := 'Stack Tabs?';
  for i := 0 to 20 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'Tab #' + IntToStr(i);
    end;
end;
```

See Also

ScrollOpposite (see page 3944)

27.1.48.2.15 TTabControl.MultiSelect Property

Specifies whether multiple tabs can be selected.

Pascal

```
property MultiSelect: Boolean;
```

C++

```
__property Boolean MultiSelect;
```

Description

Use MultiSelect to get or set whether multiple tabs can be selected.

MultiSelect can only be true if Style is tsFlatButtons or tsButtons.

See Also

ScrollOpposite (see page 3944)

27.1.48.2.16 TTabControl.OwnerDraw Property

Specifies whether the tab control handles its own painting.

Pascal

```
property OwnerDraw: Boolean;
```

C++

```
__property Boolean OwnerDraw;
```

Description

Set OwnerDraw to true to override any default painting behavior.

See Also

Canvas (see page 3940)

OnDrawTab (see page 3956)

27.1.48.2.17 TTabControl.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

27.1.48.2.18 TTabControl.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

27.1.48.2.19 TTabControl.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.48.2.20 TTabControl.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
```

```
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
```

```
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
```

```
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.48.2.21 TTabControl.RaggedRight Property

Specifies whether rows of tabs stretch to fill the width of the control.

Pascal

```
property RaggedRight: Boolean;
```

C++

```
__property Boolean RaggedRight;
```

Description

Use RaggedRight to get or set whether rows of tabs stretch to fill the width of the control.

27.1.48.2.22 TTabControl.ScrollOpposite Property

Determines how the rows of tabs are scrolled in a multi-line tab control.

Pascal

```
property ScrollOpposite: Boolean;
```

C++

```
__property Boolean ScrollOpposite;
```

Description

Set ScrollOpposite to determine where previous rows of tabs in a multi-line tab control move when the user selects a tab in another row. When ScrollOpposite is true, previous rows of tabs are moved to the bottom of the tab control if TabPosition is tpTop, or to the top if TabPosition is tpBottom. When ScrollOpposite is false, previous rows of tabs are moved to the back of all other rows of tabs.

For example, in a tab control with three rows of tabs at the top, if the user selects a tab in the second row, ScrollOpposite determines where the first row moves. If ScrollOpposite is true, the second row now appears at the front (bottom) row on top of the tab control, followed by the third row behind (above) it. The first row moves to the bottom of the tab control. If ScrollOpposite is false, the first row moves to the back (top), so that the tab control displays the second row in front (at the bottom of the tab region), followed by the third row (now in the middle), followed by the first row.

Note: Setting ScrollOpposite to true automatically sets the MultiLine property to true.

See Also

MultiLine (see page 3942)

TabPosition (see page 3948)

27.1.48.2.23 TTabControl.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine

whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
```

```
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.48.2.24 TTabControl.Style Property

Specifies the style of the tab control.

Pascal

```
property Style: TTabStyle;
```

C++

```
__property TTabStyle Style;
```

Description

Use Style to determine the appearance of the tabs. These can appear as notebook tabs or as buttons.

27.1.48.2.25 TTabControl.TabHeight Property

Specifies the height, in pixels, of the tabs in the tab control.

Pascal

```
property TabHeight: Smallint;
```

C++

```
__property Smallint TabHeight;
```

Description

Set TabHeight to control the height of the tabs that appear above the client region of the tab control. The TabHeight property is the vertical size in pixels of the individual tabs. If TabHeight is set to 0, the tabs automatically size themselves to fit their text.

C++ Examples:

```
/*
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
     AnsiString s = PageControl1->Pages[i]->Caption;
     if (Canvas->TextWidth(s) * 2 > PageControl1->TabWidth)
```

```
        PageControl1->TabWidth = Canvas->TextWidth(s) * 2;
      if (Canvas->TextHeight(s) * 2 > PageControl1->TabHeight)
        PageControl1->TabHeight = Canvas->TextHeight(s) * 2;
  }
}
```

Delphi Examples:

```
{
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  with PageControl1 do
  begin
    for i := 0 to PageCount - 1 do
    begin
      if (Canvas.TextWidth(Pages[i].Caption) * 2) > TabWidth then
        TabWidth := Canvas.TextWidth(Pages[i].Caption) * 2;
      if (Canvas.TextHeight(Pages[i].Caption) * 2) > TabHeight then
        TabHeight := Canvas.TextHeight(Pages[i].Caption) * 2;
    end;
  end;
end;
```

See Also

DisplayRect (see page 3941)

Tabs (see page 3948)

TabWidth (see page 3950)

27.1.48.2.26 TTabControl.TabIndex Property

Identifies the selected tab on a tab control.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Read TabIndex to determine which tab was selected by the user. TabIndex is the index of the tab in the list of labels maintained by the Tabs property. The first (leftmost) tab has a TabIndex of 0, the next has 1, and so on. If no tabs are selected, TabIndex has a value of -1.

Set TabIndex to programmatically change the selected tab in the tab control.

C++ Examples:

```
/*
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
```

```
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
*/
void __fastcall TForm1::Add_a_fileClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    int index = TabControl1->Tabs->Add(OpenDialog1->FileName);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[index]);
  }
}

void __fastcall TForm1::Assign_filesClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TabControl1->Tabs->Assign(OpenDialog1->Files);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[TabControl1->TabIndex]);
  }
}
/*
Place the following code in the tab control's OnChange event
handler:
*/
void __fastcall TForm1::TabControl1Change(TObject *Sender)
{
    Memo1->Lines->LoadFromFile(
      TabControl1->Tabs->Strings[TabControl1->TabIndex]);
}
```

Delphi Examples:

```
{
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
}
procedure TForm1.Add_a_fileClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    index:= TabControl1.Tabs.Add(OpenDialog1.FileName);
    Memo1.Lines.LoadFromFile(TabControl1.Tabs[index]);
  end;
end;

procedure TForm1.Assign_filesClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
```

```
      [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
    if OpenDialog1.Execute then
    begin
      TabControl1.Tabs.Assign(OpenDialog1.Files);
      Memo1.Lines.LoadFromFile(
        TabControl1.Tabs[TabControl1.TabIndex]);
    end;
  end;

  {
  Place the following code in the tab control's OnChange event
  handler:
  }
  procedure TForm1.TabControl1Change(Sender: TObject);
  begin
    with TabControl1 do
      Memo1.Lines.LoadFromFile(Tabs[TabIndex]);
  end;
```

See Also

Tabs (see page 3948)

27.1.48.2.27 TTabControl.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

27.1.48.2.28 TTabControl.TabPosition Property

Determines whether tabs appear at the top or bottom.

Pascal

```
property TabPosition: TTabPosition;
```

C++

```
__property TTabPosition TabPosition;
```

Description

Set TabPosition to tpTop to display the tabs at the top of the tab control. Set TabPosition to tpBottom to display the tabs at the bottom. If the ScrollOpposite property is true, tabs can appear at both top and bottom, with the row containing the current tab and all subsequent rows on the side indicated by TabPosition.

Due to a limitation in the windows common control, if TabPosition is tpLeft or tpRight, Font must be a True Type font. Otherwise, tab text may paint incorrectly.

See Also

ScrollOpposite (see page 3944)

27.1.48.2.29 TTabControl.Tabs Property

Contains the list of text strings that label the tabs of the tab control.

Pascal

```
property Tabs: TStrings;
```

C++

```
__property TStrings Tabs;
```

Description

Tabs contains a list of strings that label the tabs in the tab control. Tabs can also contain objects associated with the labels, using the TStrings Objects property. Add new strings to the Tabs property to add new tabs to the tab control. Rearrange the list to rearrange the tabs.

C++ Examples:

```
/*
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
*/
void __fastcall TForm1::Add_a_fileClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    int index = TabControl1->Tabs->Add(OpenDialog1->FileName);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[index]);
  }
```

```
}

void __fastcall TForm1::Assign_filesClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TabControl1->Tabs->Assign(OpenDialog1->Files);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[TabControl1->TabIndex]);
  }
}
/*
Place the following code in the tab control's OnChange event
handler:
*/
void __fastcall TForm1::TabControl1Change(TObject *Sender)
{
    Memo1->Lines->LoadFromFile(
      TabControl1->Tabs->Strings[TabControl1->TabIndex]);
}
```

Delphi Examples:

```
{
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
}
procedure TForm1.Add_a_fileClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    index:= TabControl1.Tabs.Add(OpenDialog1.FileName);
    Memo1.Lines.LoadFromFile(TabControl1.Tabs[index]);
  end;
end;

procedure TForm1.Assign_filesClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    TabControl1.Tabs.Assign(OpenDialog1.Files);
    Memo1.Lines.LoadFromFile(
      TabControl1.Tabs[TabControl1.TabIndex]);
  end;
end;

{
Place the following code in the tab control's OnChange event
handler:
}
procedure TForm1.TabControl1Change(Sender: TObject);
begin
  with TabControl1 do
```

```
    Memo1.Lines.LoadFromFile(Tabs[TabIndex]);
end;
```

See Also

TabIndex (see page 3946)

Objects (see page 3077)

TTabControl (see page 4841)

27.1.48.2.30 TTabControl.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

27.1.48.2.31 TTabControl.TabWidth Property

Specifies the horizontal size, in pixels, of the tabs in the tab control.

Pascal

```
property TabWidth: Smallint;
```

C++

```
__property Smallint TabWidth;
```

Description

TabWidth specifies a fixed width for all tabs. If TabWidth is 0, then each tab is automatically made wide enough to show its caption.

C++ Examples:

```
/*
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < PageControl1->PageCount; i++)
```

```
    {
       AnsiString s = PageControl1->Pages[i]->Caption;
       if (Canvas->TextWidth(s) * 2 > PageControl1->TabWidth)
          PageControl1->TabWidth = Canvas->TextWidth(s) * 2;
        if (Canvas->TextHeight(s) * 2 > PageControl1->TabHeight)
          PageControl1->TabHeight = Canvas->TextHeight(s) * 2;
    }
  }
```

Delphi Examples:

```
{
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  with PageControl1 do
  begin
    for i := 0 to PageCount - 1 do
    begin
      if (Canvas.TextWidth(Pages[i].Caption) * 2) > TabWidth then
        TabWidth := Canvas.TextWidth(Pages[i].Caption) * 2;
      if (Canvas.TextHeight(Pages[i].Caption) * 2) > TabHeight then
        TabHeight := Canvas.TextHeight(Pages[i].Caption) * 2;
    end;
  end;
end;
```

See Also

TabHeight (see page 3945)

Tabs (see page 3948)

27.1.48.2.32 TTabControl.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
```

```
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.48.3 TTabControl Events

The events of the TTabControl class are listed here.

Published Events

	Name	Description
	OnChange (see page 4884)	Occurs after a new tab is selected.
	OnChanging (see page 4888)	Occurs immediately before a new tab is selected.
	OnContextPopup (see page 4889)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDockDrop (see page 4890)	Occurs when another control is docked to the control.
	OnDockOver (see page 4892)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 4894)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4896)	Occurs when the user drags an object over a control.
	OnDrawTab (see page 4898)	Occurs when a tab is about to be drawn.
	OnEndDock (see page 4899)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4899)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4900)	Occurs when a control receives the input focus.
	OnExit (see page 4903)	Occurs when the input focus shifts away from one control to another.
	OnGetImageIndex (see page 4904)	Occurs when a tab is about to display its associated image.
	OnGetSiteInfo (see page 4904)	Returns the control's docking information.
	OnMouseActivate (see page 4907)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4908)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4910)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4910)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4910)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4913)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 4915)	Occurs immediately after the control is resized.
	OnStartDock (see page 4915)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4916)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 4917)	Occurs when the application tries to undock a control that is docked to the windowed control.

27.1.48.3.1 TTabControl.OnChange Event

Occurs after a new tab is selected.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action immediately after the selected tab changes. Use the TabIndex property to determine which tab is now selected. This is the opportunity to make any changes to the control that reflect the new state implied by the selected tag.

Before the value of TabIndex changes, an OnChanging event occurs.

C++ Examples:

```
/*
This project requires a Page Control.  You will need to
populate the Page Control by right clicking and selecting
New Page from the context menu. As the tabs are selected by
the user, the Page Control OnChange event fires, which
causes the form caption to display the captions of each of
the TTabSheets attached to the Page Control.
Note the use of the Form's OnShow event handler to set the
caption when the form first appears.  This occurs before
any changes trigger the OnChange event of the Page Control.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  Caption = AnsiString("Now working on tab: " + PageControl1->ActivePage->Caption);
}

void __fastcall TForm1::FormShow(TObject *Sender)
{
  PageControl1Change(Sender);
}
/*
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
*/
void __fastcall TForm1::Add_a_fileClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    int index = TabControl1->Tabs->Add(OpenDialog1->FileName);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[index]);
  }
}

void __fastcall TForm1::Assign_filesClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TabControl1->Tabs->Assign(OpenDialog1->Files);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[TabControl1->TabIndex]);
  }
}
/*
```

```
Place the following code in the tab control's OnChange event
handler:
*/
void __fastcall TForm1::TabControl1Change(TObject *Sender)
{
    Memo1->Lines->LoadFromFile(
      TabControl1->Tabs->Strings[TabControl1->TabIndex]);
}
/*
This example requires a new TPageControl, with no new pages
created at design time.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
Page Control's OnChange event handler displays a message
dialog when the user changes tabs.  The message dialog
contains the captions for the tabs immediately before and
after the active tab.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(PageControl1);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    pPage->Name = AnsiString("ts") + pPage->Caption;
  }
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  AnsiString PrevCaption, NextCaption;
  TPageControl *pPC = (TPageControl *)Sender;
  PrevCaption = pPC->FindNextPage(pPC->ActivePage, false, false)->Caption;
  NextCaption = pPC->FindNextPage(pPC->ActivePage, true, false)->Caption;
  ShowMessage(AnsiString("Previous tab caption: '") + PrevCaption + AnsiString("'  Next tab
Caption: '") + NextCaption + AnsiString("'"));
}
```

Delphi Examples:

```
{
This project requires a Page Control.  You will need to
populate the Page Control by right clicking and selecting
New Page from the context menu. As the tabs are selected by
the user, the Page Control OnChange event fires, which
causes the form caption to display the captions of each of
the TTabSheets attached to the Page Control.
Note the use of the Form's OnShow event handler to set the
caption when the form first appears.  This occurs before
any changes trigger the OnChange event of the Page Control.
}
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  Caption := ' Now working on tab:  ' + PageControl1.ActivePage.Caption;
end;

procedure TForm1.FormShow(Sender: TObject);
begin
  PageControl1Change(Sender);
end;
{
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
```

```
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
}
procedure TForm1.Add_a_fileClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    index:= TabControl1.Tabs.Add(OpenDialog1.FileName);
    Memo1.Lines.LoadFromFile(TabControl1.Tabs[index]);
  end;
end;

procedure TForm1.Assign_filesClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    TabControl1.Tabs.Assign(OpenDialog1.Files);
    Memo1.Lines.LoadFromFile(
      TabControl1.Tabs[TabControl1.TabIndex]);
  end;
end;

{
Place the following code in the tab control's OnChange event
handler:
}
procedure TForm1.TabControl1Change(Sender: TObject);
begin
  with TabControl1 do
    Memo1.Lines.LoadFromFile(Tabs[TabIndex]);
end;
{
This example requires a new TPageControl, with no new pages
created at design time.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
Page Control's OnChange event handler displays a message
dialog when the user changes tabs.  The message dialog
contains the captions for the tabs immediately before and
after the active tab.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet #' + IntToStr(i);
    end;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
var
  PrevCaption, NextCaption: ShortString;
begin
  with (Sender as TPageControl) do
  begin
```

```
      PrevCaption :=
        FindNextPage(ActivePage, False, False).Caption;
      NextCaption :=
        FindNextPage(ActivePage, True, False).Caption;
    end;
    ShowMessage('Previous tab caption: "' + PrevCaption +
      '"    Next tab Caption: "' + NextCaption + '"');
  end;
```

See Also

OnChanging (see page 3955)

TabIndex (see page 3946)

27.1.48.3.2 TTabControl.OnChanging Event

Occurs immediately before a new tab is selected.

Pascal

```
property OnChanging: TTabChangingEvent;
```

C++

```
__property TTabChangingEvent OnChanging;
```

Description

Write an OnChanging event handler to take specific action immediately before the selected tab changes. Set the AllowChange parameter to false to prevent the change.

Use an OnChanging event handler to prevent the user from leaving a tab setting until certain conditions have been met. An OnChanging event handler can also be used to save information about the current state of the tab control before it is changed by a new tab selection.

Note: The event is not called if you change the active page in code, for example, by setting the value of TPageControl. ActivePage.

C++ Examples:

```
/*
This project requires a TPageControl, several TTabSheets (to
add sheets, right click the Page Control and select "New
Page" from the context menu), and a TEdit control on the
first tab sheet.  The user is not allowed to change to a new
page if the active page is "TabSheet1" and there is no text
in the Edit Control.
*/
void __fastcall TForm1::PageControl1Changing(TObject *Sender, bool &AllowChange)
{
  if (((TPageControl *)Sender)->ActivePage == TabSheet1)
    AllowChange = (!Edit1->Text.IsEmpty());
  else
    AllowChange = true;
  if (AllowChange == false)
  {
    MessageBeep(0);
    Edit1->SetFocus();
  }
}
```

Delphi Examples:

```
{
```

```
This project requires a TPageControl, several TTabSheets (to
add sheets, right click the Page Control and select "New
Page" from the context menu), and a TEdit control on the
first tab sheet.  The user is not allowed to change to a new
page if the active page is "TabSheet1" and there is no text
in the Edit Control.
}
procedure TForm1.PageControl1Changing(Sender: TObject; var AllowChange: Boolean);
begin
  if ((Sender as TPageControl).ActivePage = TabSheet1) then
    AllowChange := (Edit1.Text <> '')
  else
    AllowChange := True;
end;
```

See Also

OnChange (see page 3952)

TabIndex (see page 3946)

27.1.48.3.3 TTabControl.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.48.3.4 TTabControl.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop: TDockDropEvent;
```

C++

```
__property TDockDropEvent OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
```

```
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock = DockClient->ClassNameIs("TDockableForm");
    if (!CanDock)
    {
       ShowMessage("GetSiteInfo CanDock went bad!");
    }
  }
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockOver (see page 6216)

27.1.48.3.5 TTabControl.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver: TDockOverEvent;
```

C++

```
__property TDockOverEvent OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
```

```
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnGetSiteInfo (see page 6223)

TDragDockObject (see page 6009)

27.1.48.3.6 TTabControl.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
```

```
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

27.1.48.3.7 TTabControl.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
```

```
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
```

```
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.48.3.8 TTabControl.OnDrawTab Event

Occurs when a tab is about to be drawn.

Pascal

```
property OnDrawTab: TDrawTabEvent;
```

C++

```
__property TDrawTabEvent OnDrawTab;
```

Description

Use OnDrawTab to customize the painting of tabs.

You can paint the tab using the Canvas property.

See Also

Canvas (see page 3940)

OwnerDraw (see page 3944)

27.1.48.3.9 TTabControl.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.48.3.10 TTabControl.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
```

```
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

27.1.48.3.11 TTabControl.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
```

```
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

27.1.48.3.12 TTabControl.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

27.1.48.3.13 TTabControl.OnGetImageIndex Event

Occurs when a tab is about to display its associated image.

Pascal

```
property OnGetImageIndex: TTabGetImageEvent;
```

C++

```
__property TTabGetImageEvent OnGetImageIndex;
```

Description

Write an OnGetImageIndex event handler to specify the ImageIndex for a given TabIndex.

On entry to the event handler, the ImageIndex parameter is the same as the TabIndex parameter, where TabIndex is the index of the tab which is about to display its image. Change the ImageIndex parameter to specify the index into the Images property that specifies the image that should be painted.

See Also

Images (see page 3941)

TabIndex (see page 3946)

27.1.48.3.14 TTabControl.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo: TGetSiteInfoEvent;
```

C++

```
__property TGetSiteInfoEvent OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
```

```
      ShowMessage("GetSiteInfo CanDock went bad!");
    }
  }
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

TDragDockObject (see page 6009)

27.1.48.3.15 TTabControl.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.48.3.16 TTabControl.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
```

```
      }
      else
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
```

```
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.48.3.17 TTabControl.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

27.1.48.3.18 TTabControl.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

27.1.48.3.19 TTabControl.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
```

```
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.48.3.20 TTabControl.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
```

```
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
```

```
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

27.1.48.3.21 TTabControl.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

27.1.48.3.22 TTabControl.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

27.1.48.3.23 TTabControl.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

27.1.48.3.24 TTabControl.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock: TUnDockEvent;
```

C++

```
__property TUnDockEvent OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
```

```
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
```

```
  shows how set permissions to dock dockable objects onto a
  docking site.
  }

  procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer);
  begin
    //OnDockDrop gets called AFTER the client has actually docked,
    //so we check for DockClientCount = 1 before making the dock panel visible.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, True, nil);
    (Sender as TPanel).DockManager.ResetBounds(True);
    //Make DockManager repaints it's clients.
  end;

  procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer; State: TDragState;
    var Accept: Boolean);
  var
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

27.1.49 TTabSheet Class

TTabSheet is an individual page in a TPageControl (see page 4459) object.

Class Hierarchy

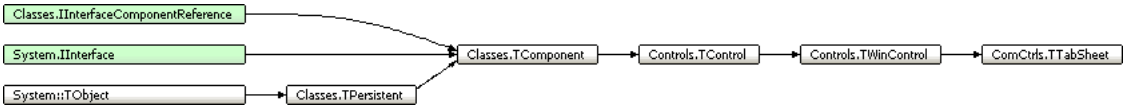

Pascal

```
TTabSheet = class(TWinControl);
```

C++

```
class TTabSheet : public TWinControl;
```

File

ComCtrls

Description

Use TTabSheet to represent a single page in a tabbed page control. Tab sheets are typically referred to as pages. TPageControl (see page 4459) maintains an indexed array of its tab sheets in its Pages property. Users can click on a tab to activate the tab sheet.

See Also

TPageControl (see page 4459)

27.1.49.1 TTabSheet Members

The following tables list the members exposed by TTabSheet.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 4932)	Initializes window-creation parameters for the tab sheet.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.

	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoHide (see page 4933)	Called when the user selects another tab.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoShow (see page 4933)	Called when the user selects another tab.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.

	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.

	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 4933)	Prepare control to be initialized from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 4934)	This is UpdateControlOriginalParentSize, a member of class TTabSheet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.

V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 4930)	Creates and initializes an instance of TTabSheet (see page 4919).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 4932)	Destroys an instance of TTabSheet (see page 4919).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.

V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.

	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	PageControl (see page 4944)	Indicates the page control object that contains the tab sheet.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
R	TabIndex (see page 4955)	Indicates the position of the tab sheet in the set of visible tabs in a TPageControl (see page 4459) object.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnContextPopup (see page 4961)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 4962)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4964)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 4966)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4967)	Occurs when a control receives the input focus.
	OnExit (see page 4970)	Occurs when the input focus shifts away from one control to another.
	OnHide (see page 4971)	Occurs when the TabVisible (see page 4956) property changes to false.
	OnMouseActivate (see page 4972)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4973)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4975)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4975)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4975)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4978)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 4980)	Occurs immediately after the control is resized.
	OnShow (see page 4980)	Occurs when the TabVisible (see page 4956) property changes to true.

	OnStartDrag (see page 4981)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	BorderWidth (see page 4934)	Specifies the width of the control's border.
	Caption (see page 4935)	Specifies a text string that identifies the control to the user.
	Constraints (see page 4937)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragMode (see page 4938)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4940)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4940)	Controls the attributes of text written on or in the control.
	Height (see page 4941)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Highlighted (see page 4942)	Indicates whether the tab sheet appears highlighted.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImageIndex (see page 4943)	Specifies an image for the tab.
	Left (see page 4943)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	PageIndex (see page 4946)	Indicates the index of the tab sheet in the list of tab sheets maintained by the page control.
	ParentFont (see page 4948)	Determines where a control looks for its font information.
	ParentShowHint (see page 4948)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4949)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 4953)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabVisible (see page 4956)	Specifies whether the tab of the TTabSheet (see page 4919) object appears in its TPageControl (see page 4459).
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 4957)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 4958)	Determines whether the component appears onscreen.
	Width (see page 4960)	Specifies the horizontal size of the control or form in pixels.

27.1.49.2 TTabSheet.Create Constructor

Creates and initializes an instance of TTabSheet (see page 4919).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TTabSheet(TComponent * AOwner);
```

Description

Call Create to instantiate a tab sheet at runtime. Tab sheets added to a page control at design time are created automatically.

Create calls the inherited Create method, then sets the initial values for the tab sheet component.

When creating tab sheets at runtime, use the PageControl (see page 4944) property to insert them into a page control object.

C++ Examples:

```
/*
This example creates ten Tab Sheets and sets the captions to
display PageIndex and TabIndex for each tab.  Odd numbered
tabs have their TabVisible property set off to demonstrate
the effect on PageIndex vs TabIndex.  On the series of
visible tabs, PageIndex will read 0, 2, 4, 6, 8, and
TabIndex will read 0, 1, 2, 3, 4.  PageIndex counts only
visible tabs.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  PageControl1->MultiLine = true;
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *page = new TTabSheet(this);
    page->PageControl = PageControl1;
    page->TabVisible = (i % 2 == 0);
    page->Caption =
      AnsiString("PageIndex: ") + IntToStr(page->PageIndex) +
      AnsiString("  TabIndex: ") + IntToStr(page->TabIndex);
    page->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example creates ten Tab Sheets and sets the captions to
display PageIndex and TabIndex for each tab.  Odd numbered
tabs have their TabVisible property set off to demonstrate
the effect on PageIndex vs TabIndex.  On the series of
visible tabs, PageIndex will read 0, 2, 4, 6, 8, and
TabIndex will read 0, 1, 2, 3, 4.  PageIndex counts only
visible tabs.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  with PageControl1 do
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        TabVisible := (PageIndex mod 2 = 0);
        Brush.Color := colorpalette[i];
        Caption := 'PageIndex: ' +
        IntToStr(PageIndex) + '  TabIndex: ' +
          IntToStr(TabIndex);
      end;
end;
```

See Also

PageControl (see page 4944)

Create (see page 6123)

27.1.49.3 TTabSheet.Destroy Destructor

Destroys an instance of TTabSheet (see page 4919).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTabSheet();
```

Description

Do not call Destroy directly at runtime. Instead, call Free. Free verifies that the tab sheet is not nil and only then calls Destroy. Destroy removes the tab sheet from the page control and then calls the inherited Destroy method.

See Also

Free (see page 22300)

PageControl (see page 4944)

27.1.49.4 TTabSheet Methods

The methods of the TTabSheet class are listed here.

Protected Methods

	Name	Description
V	CreateParams (see page 4932)	Initializes window-creation parameters for the tab sheet.
	DoHide (see page 4933)	Called when the user selects another tab.
	DoShow (see page 4933)	Called when the user selects another tab.
V	ReadState (see page 4933)	Prepare control to be initialized from a stream.
V	UpdateControlOriginalParentSize (see page 4934)	This is UpdateControlOriginalParentSize, a member of class TTabSheet.

27.1.49.4.1 TTabSheet.CreateParams Method

Initializes window-creation parameters for the tab sheet.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The tab sheet calls CreateParams internally when the tab sheet must be generated. CreateParams overrides the inherited method to specify the type of window for the tab sheet.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

CreateParams (see page 6135)

27.1.49.4.2 TTabSheet.DoHide Method

Called when the user selects another tab.

Pascal

```
procedure DoHide; dynamic;
```

C++

```
__fastcall DoHide();
```

Description

Generates an OnHide (see page 4971) event. DoHide is called automatically when the TabVisible (see page 4956) property changes to **false**. Override DoHide in a derived class to perform class-specific actions when the tab sheet is temporarily removed from its page control, or to block the OnHide (see page 4971) event.

See Also

OnHide (see page 4971)

27.1.49.4.3 TTabSheet.DoShow Method

Called when the user selects another tab.

Pascal

```
procedure DoShow; dynamic;
```

C++

```
__fastcall DoShow();
```

Description

Generates an OnShow (see page 4980) event. DoShow is called automatically when the TabVisible (see page 4956) property changes to false. Override DoShow in a derived class to perform class-specific actions when the tab sheet appears in its page control after being temporarily hidden, or to block the OnHide (see page 4971) event.

See Also

OnShow (see page 4980)

27.1.49.4.4 TTabSheet.ReadState Method

Prepare control to be initialized from a stream.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

TTabSheet (see page 4919) reimplements in order to assign Reader's parent as the Tab Sheet's parent.

27.1.49.4.5 TTabSheet.UpdateControlOriginalParentSize Method

Pascal

```
procedure UpdateControlOriginalParentSize(AControl: TControl; var AOriginalParentSize:
TPoint); override;
```

C++

```
virtual __fastcall UpdateControlOriginalParentSize(TControl * AControl, TPoint
AOriginalParentSize);
```

Description

This is UpdateControlOriginalParentSize, a member of class TTabSheet.

27.1.49.5 TTabSheet Properties

The properties of the TTabSheet class are listed here.

Public Properties

	Name	Description
	PageControl (see page 4944)	Indicates the page control object that contains the tab sheet.
R	TabIndex (see page 4955)	Indicates the position of the tab sheet in the set of visible tabs in a TPageControl (see page 4459) object.

Published Properties

	Name	Description
	BorderWidth (see page 4934)	Specifies the width of the control's border.
	Caption (see page 4935)	Specifies a text string that identifies the control to the user.
	Constraints (see page 4937)	Specifies the size constraints for the control.
	DragMode (see page 4938)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4940)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4940)	Controls the attributes of text written on or in the control.
	Height (see page 4941)	Specifies the vertical size of the control in pixels.
	Highlighted (see page 4942)	Indicates whether the tab sheet appears highlighted.
	ImageIndex (see page 4943)	Specifies an image for the tab.
	Left (see page 4943)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	PageIndex (see page 4946)	Indicates the index of the tab sheet in the list of tab sheets maintained by the page control.
	ParentFont (see page 4948)	Determines where a control looks for its font information.
	ParentShowHint (see page 4948)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4949)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 4953)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabVisible (see page 4956)	Specifies whether the tab of the TTabSheet (see page 4919) object appears in its TPageControl (see page 4459).
	Top (see page 4957)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 4958)	Determines whether the component appears onscreen.
	Width (see page 4960)	Specifies the horizontal size of the control or form in pixels.

27.1.49.5.1 TTabSheet.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics (see page 14387) or text drawn by the control is clipped to the area within the border.

27.1.49.5.2 TTabSheet.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
```

```
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
```

```
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

27.1.49.5.3 TTabSheet.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or

minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.49.5.4 TTabSheet.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
```

```
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

27.1.49.5.5 TTabSheet.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.49.5.6 TTabSheet.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

27.1.49.5.7 TTabSheet.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
```

```
      pRich->SelStart = 0;
      pRich->SelLength = pRich->Text.Length();
      if (I == 1)
        pRich->HideSelection = false;
    }
  }
```

Delphi Examples:

```
{
HideSelection, Height example
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
        HideSelection := False;
    end;
end;
```

See Also

ClientHeight (see page 5808)

Width (see page 5849)

Using TDBListBox and TDBComboBox

27.1.49.5.8 TTabSheet.Highlighted Property

Indicates whether the tab sheet appears highlighted.

Pascal

```
property Highlighted: Boolean;
```

C++

```
__property Boolean Highlighted;
```

Description

Use Highlighted to make a tab sheet stand out visually. When Highlighted is true, the tab sheet is drawn using a highlight color so that it stands out visually from other tab sheets.

Note: Highlighted is not linked to the active page of the host page control. A page control can have only one active page, but can have multiple highlighted pages.

See Also

TabVisible (see page 4956)

27.1.49.5.9 TTabSheet.ImageIndex Property

Specifies an image for the tab.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

Use ImageIndex to get or set the index of an image to be drawn in the tab. This index identifies an image from the Images property of the page control that contains the tab sheet. Specify -1 to prevent an image from being drawn.

See Also

Images (see page 3941)

27.1.49.5.10 TTabSheet.Left Property

Specifies the horizontal coordinate of the left edge of a component relative to its parent.

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

Use the Left property to determine where the left side of the control begins or to reposition the left side of the control.

If the control is contained in another control, the Left and Top properties are relative to the parent control. If the control is contained directly by the form, the property values are relative to the form. For forms, the value of the Left property is relative to the screen in pixels.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Timer1->Interval = 100;
  if (ActiveControl)
    ActiveControl->Left = ActiveControl->Left + 1;
}
```

Delphi Examples:

```
{
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
```

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  if ActiveControl <> nil then
    ActiveControl.Left := ActiveControl.Left + 1;
end;
```

See Also

Top (see page 5844)

27.1.49.5.11 TTabSheet.PageControl Property

Indicates the page control object that contains the tab sheet.

Pascal

```
property PageControl: TPageControl;
```

C++

```
__property TPageControl PageControl;
```

Description

Read PageControl to gain access to the page control object that uses the tab sheet. The properties and methods of the PageControl object can be used to locate the selected page, iterate through the other pages in the page control, or change the display properties of the tabs.

Set PageControl to remove the tab sheet from its current page control (if any) and insert it into a new page control.

C++ Examples:

```
/*
This example requires a new TPageControl, with no new pages
created at design time.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
Page Control's OnChange event handler displays a message
dialog when the user changes tabs.  The message dialog
contains the captions for the tabs immediately before and
after the active tab.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(PageControl1);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    pPage->Name = AnsiString("ts") + pPage->Caption;
  }
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  AnsiString PrevCaption, NextCaption;
  TPageControl *pPC = (TPageControl *)Sender;
  PrevCaption = pPC->FindNextPage(pPC->ActivePage, false, false)->Caption;
  NextCaption = pPC->FindNextPage(pPC->ActivePage, true, false)->Caption;
  ShowMessage(AnsiString("Previous tab caption: '") + PrevCaption + AnsiString("'  Next tab
Caption: '") + NextCaption + AnsiString("'"));
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
```

```
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example requires a new TPageControl, with no new pages
created at design time.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
Page Control's OnChange event handler displays a message
dialog when the user changes tabs.  The message dialog
contains the captions for the tabs immediately before and
after the active tab.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet #' + IntToStr(i);
    end;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
var
  PrevCaption, NextCaption: ShortString;
begin
  with (Sender as TPageControl) do
  begin
    PrevCaption :=
      FindNextPage(ActivePage, False, False).Caption;
    NextCaption :=
      FindNextPage(ActivePage, True, False).Caption;
  end;
  ShowMessage('Previous tab caption: "' + PrevCaption +
    '"    Next tab Caption: "' + NextCaption + '"');
end;
```

```
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

27.1.49.5.12 TTabSheet.PageIndex Property

Indicates the index of the tab sheet in the list of tab sheets maintained by the page control.

Pascal

```
property PageIndex: Integer;
```

C++

```
__property int PageIndex;
```

Description

Use PageIndex to determine where the tab sheet sits in its page control. Each tab sheet in a page control is automatically assigned a PageIndex when it is inserted in the page control. The first tab sheet receives a value of 0, the second has a value of 1, and so on. PageIndex values are reassigned if tab sheets are deleted or moved.

To access a particular tab sheet given its index value, use the indexed Pages property of the TPageControl (see page 4459) object.

C++ Examples:

```
/*
This example creates ten Tab Sheets and sets the captions to
display PageIndex and TabIndex for each tab.  Odd numbered
tabs have their TabVisible property set off to demonstrate
the effect on PageIndex vs TabIndex.  On the series of
visible tabs, PageIndex will read 0, 2, 4, 6, 8, and
TabIndex will read 0, 1, 2, 3, 4.  PageIndex counts only
visible tabs.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  PageControl1->MultiLine = true;
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *page = new TTabSheet(this);
    page->PageControl = PageControl1;
    page->TabVisible = (i % 2 == 0);
    page->Caption =
      AnsiString("PageIndex: ") + IntToStr(page->PageIndex) +
      AnsiString("  TabIndex: ") + IntToStr(page->TabIndex);
    page->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example creates ten Tab Sheets and sets the captions to
display PageIndex and TabIndex for each tab.  Odd numbered
tabs have their TabVisible property set off to demonstrate
the effect on PageIndex vs TabIndex.  On the series of
visible tabs, PageIndex will read 0, 2, 4, 6, 8, and
TabIndex will read 0, 1, 2, 3, 4.  PageIndex counts only
visible tabs.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  with PageControl1 do
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        TabVisible := (PageIndex mod 2 = 0);
        Brush.Color := colorpalette[i];
        Caption := 'PageIndex: ' +
        IntToStr(PageIndex) + '  TabIndex: ' +
          IntToStr(TabIndex);
      end;
end;
```

See Also

PageControl (see page 4944)

Pages (see page 4498)

27.1.49.5.13 TTabSheet.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

27.1.49.5.14 TTabSheet.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.49.5.15 TTabSheet.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
```

```
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
```

```
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
```

```
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.49.5.16 TTabSheet.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.49.5.17 TTabSheet.TabIndex Property

Indicates the position of the tab sheet in the set of visible tabs in a TPageControl (see page 4459) object.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Read TabIndex to determine where the user sees the tab sheet. A value of 0 indicates the first visible tab sheet, a value of 1 indicates the second visible tab sheet, and so on. TabIndex is always less than or equal to PageIndex (see page 4946). For example, a tab sheet that has a PageIndex (see page 4946) of 3 will have a TabIndex of 2 if one of the previous tab sheets in the page control is not visible.

If a tab sheet's TabVisible (see page 4956) property is false, the TabIndex property is -1.

C++ Examples:

```
/*
This example creates ten Tab Sheets and sets the captions to
display PageIndex and TabIndex for each tab.  Odd numbered
tabs have their TabVisible property set off to demonstrate
the effect on PageIndex vs TabIndex.  On the series of
visible tabs, PageIndex will read 0, 2, 4, 6, 8, and
TabIndex will read 0, 1, 2, 3, 4.  PageIndex counts only
visible tabs.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  PageControl1->MultiLine = true;
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *page = new TTabSheet(this);
    page->PageControl = PageControl1;
    page->TabVisible = (i % 2 == 0);
    page->Caption =
      AnsiString("PageIndex: ") + IntToStr(page->PageIndex) +
      AnsiString("  TabIndex: ") + IntToStr(page->TabIndex);
    page->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example creates ten Tab Sheets and sets the captions to
display PageIndex and TabIndex for each tab.  Odd numbered
tabs have their TabVisible property set off to demonstrate
the effect on PageIndex vs TabIndex.  On the series of
visible tabs, PageIndex will read 0, 2, 4, 6, 8, and
TabIndex will read 0, 1, 2, 3, 4.  PageIndex counts only
visible tabs.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
```

```
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  with PageControl1 do
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        TabVisible := (PageIndex mod 2 = 0);
        Brush.Color := colorpalette[i];
        Caption := 'PageIndex: ' +
        IntToStr(PageIndex) + '  TabIndex: ' +
          IntToStr(TabIndex);
      end;
end;
```

See Also

TabVisible (see page 4956)

27.1.49.5.18 TTabSheet.TabVisible Property

Specifies whether the tab of the TTabSheet (see page 4919) object appears in its TPageControl (see page 4459).

Pascal

```
property TabVisible: Boolean;
```

C++

```
__property Boolean TabVisible;
```

Description

Use TabVisible to temporarily remove a tab sheet from a page control. When TabVisible is false, the tab does not appear in the page control and its TabIndex (see page 4955) property is -1. Setting TabVisible to true allows the user to see the tab of the tab sheet again.

C++ Examples:

```
/*
This example creates ten Tab Sheets and sets the captions to
display PageIndex and TabIndex for each tab.  Odd numbered
tabs have their TabVisible property set off to demonstrate
the effect on PageIndex vs TabIndex.  On the series of
visible tabs, PageIndex will read 0, 2, 4, 6, 8, and
TabIndex will read 0, 1, 2, 3, 4.  PageIndex counts only
visible tabs.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  PageControl1->MultiLine = true;
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *page = new TTabSheet(this);
    page->PageControl = PageControl1;
    page->TabVisible = (i % 2 == 0);
    page->Caption =
      AnsiString("PageIndex: ") + IntToStr(page->PageIndex) +
      AnsiString("  TabIndex: ") + IntToStr(page->TabIndex);
    page->Brush->Color = colorPalette[i];
```

```
  }
}
```

Delphi Examples:

```
{
This example creates ten Tab Sheets and sets the captions to
display PageIndex and TabIndex for each tab.  Odd numbered
tabs have their TabVisible property set off to demonstrate
the effect on PageIndex vs TabIndex.  On the series of
visible tabs, PageIndex will read 0, 2, 4, 6, 8, and
TabIndex will read 0, 1, 2, 3, 4.  PageIndex counts only
visible tabs.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  with PageControl1 do
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        TabVisible := (PageIndex mod 2 = 0);
        Brush.Color := colorpalette[i];
        Caption := 'PageIndex: ' +
        IntToStr(PageIndex) + '  TabIndex: ' +
          IntToStr(TabIndex);
      end;
end;
```

See Also

TabIndex (see page 4955)

27.1.49.5.19 TTabSheet.Top Property

Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

Use Top to locate the top of the control or reposition the control to a different Y coordinate. The Top property, like the Left property, is the position of the control relative to its container. Thus, if a control is contained in a TPanel, the Left and Top properties are relative to the panel. If the control is contained directly by the form, it is relative to the form. For forms, the value of the Top property is relative to the screen in pixels.

C++ Examples:

```
/*
This example uses an updown control and a group box on a
form, with several controls contained within the group box.
When the updown control or is clicked, the controls within
the group box are moved in the appropriate direction.
*/
void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
```

```
{
  int I;
  TControl *ChildControl;

  for (I = 0; I < GroupBox1->ControlCount; I++)
  {
    ChildControl = GroupBox1->Controls[I];
    if (Button == btNext)
      ChildControl->Top = ChildControl->Top + 15;
    else
      ChildControl->Top = ChildControl->Top - 15;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorarray[5] = {clYellow, clGreen, clBlue, clLime, clFuchsia};
  GroupBox1->Brush->Color = clRed;
  for (int i = 0; i < GroupBox1->ControlCount; i++)
    ((TWinControl *)GroupBox1->Controls[i])->Brush->Color = colorarray[i];
}
```

Delphi Examples:

```
{
This example uses an updown control and a group box on a
form, with several controls contained within the group box.
When the updown control or is clicked, the controls within
the group box are moved in the appropriate direction.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
const
  colorarray : Array[0..4] of TColor = (clYellow, clGreen, clBlue, clLime, clFuchsia);
begin
   GroupBox1.Brush.Color := clRed;
  for I:= 0 to GroupBox1.ControlCount -1 do
  begin
    TWinControl(GroupBox1.Controls[I]).Brush.Color := colorarray[I];
  end;
end;

procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
var
  I: Integer;
  ChildControl: TControl;
begin
  for I:= 0 to GroupBox1.ControlCount -1 do
  begin
    ChildControl := GroupBox1.Controls[I];
    if Button = btNext then
      ChildControl.Top := ChildControl.Top + 15
    else
      ChildControl.Top := ChildControl.Top - 15;
  end;
end;
```

See Also

Left (see page 5830)

27.1.49.5.20 TTabSheet.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.49.5.21 TTabSheet.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

See Also

ClientWidth (see page 5811)

Height (see page 5825)

Left (see page 5830)

27.1.49.6 TTabSheet Events

The events of the TTabSheet class are listed here.

Published Events

	Name	Description
	OnContextPopup (see page 4961)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 4962)	Occurs when the user drops an object being dragged.

	OnDragOver (see page 4964)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 4966)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4967)	Occurs when a control receives the input focus.
	OnExit (see page 4970)	Occurs when the input focus shifts away from one control to another.
	OnHide (see page 4971)	Occurs when the TabVisible (see page 4956) property changes to false.
	OnMouseActivate (see page 4972)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4973)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4975)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4975)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4975)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4978)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 4980)	Occurs immediately after the control is resized.
	OnShow (see page 4980)	Occurs when the TabVisible (see page 4956) property changes to true.
	OnStartDrag (see page 4981)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

27.1.49.6.1 TTabSheet.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.49.6.2 TTabSheet.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

27.1.49.6.3 TTabSheet.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}
```

```
/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
```

```
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.49.6.4 TTabSheet.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
```

```
  message displayed depends on whether the dragged label
  control was successfully dropped into and accepted by a
  another control.
  */
  void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
        int Y)
  {
    if (Sender->ClassNameIs("TLabel"))
    {
      AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
      if (Target)
        S = S + "accepted!";
      else
        S = S + "rejected.";
      Status->Caption = S;
    }
  }
```

Delphi Examples:

```
  {
  This code displays a message in a label named Status. The
  message displayed depends on whether the dragged label
  control was successfully dropped into and accepted by a
  another control.
  }
  procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
  var
    S: string;
  begin
    S := (Sender as TLabel).Name + ' was dropped... and ';
    if Target <> nil then
      S := S + 'accepted!'
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

27.1.49.6.5 TTabSheet.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
  property OnEnter: TNotifyEvent;
```

C++

```
  __property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
```

```
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
```

```
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

27.1.49.6.6 TTabSheet.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
```

```
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

27.1.49.6.7 TTabSheet.OnHide Event

Occurs when the TabVisible (see page 4956) property changes to false.

Pascal

```
property OnHide: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnHide;
```

Description

Write an OnHide event handler to take specific action when the tab sheet is temporarily removed from its page control.

See Also

TabVisible (see page 4956)

OnShow (see page 4980)

27.1.49.6.8 TTabSheet.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.49.6.9 TTabSheet.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
```

```
      }
      else
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
```

```
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.49.6.10 TTabSheet.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

27.1.49.6.11 TTabSheet.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

27.1.49.6.12 TTabSheet.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
```

```
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.49.6.13 TTabSheet.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
```

```
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
```

```
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

27.1.49.6.14 TTabSheet.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

27.1.49.6.15 TTabSheet.OnShow Event

Occurs when the TabVisible (see page 4956) property changes to true.

Pascal

```
property OnShow: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnShow;
```

Description

Write an OnShow event handler to take specific action when the tab sheet appears in its page control after being temporarily hidden.

See Also

OnHide (see page 4971)

TabVisible (see page 4956)

27.1.49.6.16 TTabSheet.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

27.1.50 TTextAttributes Class

TTextAttributes represents the font characteristics of a section of text in a rich edit control.

Class Hierarchy

Pascal

```
TTextAttributes = class(TPersistent);
```

C++

```
class TTextAttributes : public TPersistent;
```

File

ComCtrls

Description

TTextAttributes includes a number of properties that describe the font characteristics of some text in a rich text document. TTextAttributes is tailored to allow a descendant of TCustomRichEdit (see page 3854) to read and write the default text characteristics (via its DefAttributes property) or the characteristics of the selected text (via its SelAttributes property). The properties of a TTextAttributes object are invalid unless it is owned by a descendant of TCustomRichEdit (see page 3854).

Most of the font characteristics represented by a TTextAttributes object are the same as the properties of a TFont object with the same names. Applications can assign fonts to text attributes and vice versa.

See Also

TCustomRichEdit (see page 3854)

TFont (see page 14497)

27.1.50.1 TTextAttributes Members

The following tables list the members exposed by TTextAttributes.

Protected Methods

	Name	Description
	AssignTo (see page 4986)	Enables the properties of a TTextAttributes (see page 4982) object to be assigned to a TFont object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.
	InitFormat (see page 4986)	Initializes a Windows TCharFormat structure.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 4984)	Sets the properties of a TTextAttributes (see page 4982) object to match the properties specified in another TTextAttributes (see page 4982) object or in a TFont object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 4983)	Creates an instance of TTextAttributes (see page 4982).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Charset (see page 4987)	Specifies the character set of the font.
	Color (see page 4987)	Indicates the foreground color of the text.
R	ConsistentAttributes (see page 4989)	Indicates which of the properties of the TTextAttributes (see page 4982) object are consistent throughout the text of the current selection in the rich edit object.
	Height (see page 4989)	Specifies the height of the font in pixels.
	Name (see page 4991)	Specifies the typeface of the font described by the TTextAttributes (see page 4982) object.
	Pitch (see page 4991)	Specifies whether the characters in the font all have the same width.
	Protected (see page 4992)	Indicates whether the text represented by the TTextAttributes (see page 4982) object is protected.
	Size (see page 4992)	Specifies the height of the font in points.
	Style (see page 4992)	Determines whether the font is normal, italic, underlined, bold, or strikeout.

27.1.50.2 TTextAttributes.Create Constructor

Creates an instance of TTextAttributes (see page 4982).

Pascal

```
constructor Create(AOwner: TCustomRichEdit; AttributeType: TAttributeType);
```

C++

```
__fastcall TTextAttributes(TCustomRichEdit AOwner, TAttributeType AttributeType);
```

Description

Do not call the constructor for TTextAttributes (see page 4982). TTextAttributes (see page 4982) objects are created from the constructor of the rich edit control. The rich edit control uses an attribute type of atDefaultText to implement its DefAttributes property, and an attribute type of atSelected to implement its SelAttributes property.

See Also

DefAttributes (see page 3877)

SelAttributes (see page 3887)

27.1.50.3 TTextAttributes Methods

The methods of the TTextAttributes class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 4986)	Enables the properties of a TTextAttributes (see page 4982) object to be assigned to a TFont object.
	InitFormat (see page 4986)	Initializes a Windows TCharFormat structure.

Public Methods

	Name	Description
	Assign (see page 4984)	Sets the properties of a TTextAttributes (see page 4982) object to match the properties specified in another TTextAttributes (see page 4982) object or in a TFont object.

27.1.50.3.1 TTextAttributes.Assign Method

Sets the properties of a TTextAttributes (see page 4982) object to match the properties specified in another TTextAttributes (see page 4982) object or in a TFont object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use Assign to change all the text attributes at once. Assign can set the font characteristics of the selected text to match the default font characteristics or vice versa. When the Source is a TTextAttributes (see page 4982) object, Assign matches only the Color (see page 4987), Name (see page 4991), Style (see page 4992), and Pitch (see page 4991) properties. When the Source is a TFont object, Assign matches the Size (see page 4992) as well.

Note: Assign only succeeds at runtime when the Source is a TTextAttributes (see page 4982) object, a TFont object, or an object that has implemented an AssignTo (see page 4986) method that deals with the TTextAttributes (see page 4982) object. Other source objects will raise an EConvertError exception.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
```

```
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
```

See Also

TFont (see page 14497)

AssignTo (see page 4986)

Color (see page 4987)

ConsistentAttributes (see page 4989)

Name (see page 4991)

Pitch (see page 4991)

Size (see page 4992)

Style (see page 4992)

27.1.50.3.2 TTextAttributes.AssignTo Method

Enables the properties of a TTextAttributes (see page 4982) object to be assigned to a TFont object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

TTextAttributes (see page 4982) overrides AssignTo to enable its font properties to be assigned to a TFont object. Applications should not call AssignTo. It is called when the Assign (see page 4984) method of the TFont object is called. AssignTo copies the Charset (see page 4987), Color (see page 4987), Name (see page 4991), Style (see page 4992), Size (see page 4992), and Pitch (see page 4991) properties to the TFont object.

See Also

Assign (see page 4984)

Color (see page 4987)

ConsistentAttributes (see page 4989)

Name (see page 4991)

Pitch (see page 4991)

Size (see page 4992)

Style (see page 4992)

27.1.50.3.3 TTextAttributes.InitFormat Method

Initializes a Windows TCharFormat structure.

Pascal

```
procedure InitFormat(var Format: TCharFormat);
```

C++

```
__fastcall InitFormat(TCharFormat Format);
```

Description

TTextAttributes (see page 4982) uses the TCharFormat structure to communicate with the rich edit control window when it gets and sets properties. InitFormat initializes this structure before any particular properties are read or set. InitFormat is exposed as a protected method so that derived classes can use the TCharFormat structure in the implementation of any other

text properties.

See Also

Color (see page 4987)

ConsistentAttributes (see page 4989)

Name (see page 4991)

Pitch (see page 4991)

Protected (see page 4992)

Size (see page 4992)

Style (see page 4992)

27.1.50.4 TTextAttributes Properties

The properties of the TTextAttributes class are listed here.

Public Properties

	Name	Description
	Charset (see page 4987)	Specifies the character set of the font.
	Color (see page 4987)	Indicates the foreground color of the text.
R	ConsistentAttributes (see page 4989)	Indicates which of the properties of the TTextAttributes (see page 4982) object are consistent throughout the text of the current selection in the rich edit object.
	Height (see page 4989)	Specifies the height of the font in pixels.
	Name (see page 4991)	Specifies the typeface of the font described by the TTextAttributes (see page 4982) object.
	Pitch (see page 4991)	Specifies whether the characters in the font all have the same width.
	Protected (see page 4992)	Indicates whether the text represented by the TTextAttributes (see page 4982) object is protected.
	Size (see page 4992)	Specifies the height of the font in points.
	Style (see page 4992)	Determines whether the font is normal, italic, underlined, bold, or strikeout.

27.1.50.4.1 TTextAttributes.Charset Property

Specifies the character set of the font.

Pascal

```
property Charset: TFontCharset;
```

C++

```
__property TFontCharset Charset;
```

Description

Set Charset to identify the character set of the font. Each typeface (specified by the Name (see page 4991) property) supports one or more character sets. Check the information supplied by the font vendor to determine what values of Charset are valid.

See Also

Name (see page 4991)

27.1.50.4.2 TTextAttributes.Color Property

Indicates the foreground color of the text.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use color to indicate the color of the actual characters.

Note: The background color is a property of the rich edit control that uses the TTextAttributes (see page 4982) object, and should be set by the Color property of the rich edit control.

C++ Examples:

```
/*
This example requires a TRichEdit and two TButtons. Leave
the default text, usually "RichEdit1", in the lines
property. The DefAttributes will apply to this text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->SelAttributes->Color = clRed;
  RichEdit1->SelAttributes->Height += 5;
  RichEdit1->Lines->Add("This line of text will be red. ");
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  RichEdit1->DefAttributes->Color = clBlue;
  RichEdit1->DefAttributes->Style =
    RichEdit1->DefAttributes->Style << fsBold << fsItalic;
}
```

Delphi Examples:

```
{
This example requires a TRichEdit and two TButtons. Leave
the default text, usually "RichEdit1", in the lines
property. The DefAttributes will apply to this text.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  with RichEdit1.SelAttributes do
  begin
    Color := clRed;
    Height := Height + 5;
  end;
  RichEdit1.Lines.Add('This line of text will be red.');
end;


procedure TForm1.Button2Click(Sender: TObject);
begin
  RichEdit1.DefAttributes.Color := clBlue;
  RichEdit1.DefAttributes.Style := [fsBold, fsItalic];
end;
```

See Also

ConsistentAttributes (see page 4989)

Color (see page 5812)

27.1.50.4.3 TTextAttributes.ConsistentAttributes Property

Indicates which of the properties of the TTextAttributes (see page 4982) object are consistent throughout the text of the current selection in the rich edit object.

Pascal

```
property ConsistentAttributes: TConsistentAttributes;
```

C++

```
__property TConsistentAttributes ConsistentAttributes;
```

Description

Read the ConsistentAttributes of the SelAttributes property of a rich text object to discover which of the properties of the TTextAttributes (see page 4982) are meaningful for the current selection. This information is relevant if you want to then read the text characteristics of the current selection. Setting a property of TTextAttributes (see page 4982) will cause that property to become a consistent attribute until the selection changes.

ConsistentAttributes is a set drawn from the following values:

Value	Meaning
caBold	Characters are consistently either bold or not bold throughout the selection.
caColor	The color of character font is consistent throughout the selection.
caFace	The typeface of the character font is consistent throughout the selection.
caItalic	Characters are consistently either italic or not italic throughout the selection.
caSize	The size of the character font is consistent throughout the selection.
caStrikeOut	Characters consistently either have the strikeout style or they do not have the strikeout style throughout the selection.
caUnderline	Characters are consistently either underlined or not underlined throughout the selection.
caProtected	Text is consistently either protected or not protected.

Note: Use ConsistentAttributes with the SelAttributes property of the rich edit control. The ConsistentAttributes of the DefAttributes property of the rich edit control is the empty set, even though all attributes are consistent in the section of text that has the default attributes.

See Also

Color (see page 4987)

Name (see page 4991)

Protected (see page 4992)

Size (see page 4992)

Style (see page 4992)

27.1.50.4.4 TTextAttributes.Height Property

Specifies the height of the font in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

To specify a font height in pixels, use the Height property. To specify the size of the font in points instead, use the Size (see page 4992) property. Users usually specify font size in points within an application, while programmers are often concerned with the actual size of the font in pixels when displaying a font on the screen.

The relationship between the Height and Size (see page 4992) properties is expressed by the formula:

```
Height = Size * ScreenPixelsPerInch / 72
```

Note: Unlike the Height property of a TFont object, the Height property of a TTextAttributes (see page 4982) object always includes the internal leading that appears at the top of the font.

C++ Examples:

```
/*
This example requires a TRichEdit and two TButtons. Leave
the default text, usually "RichEdit1", in the lines
property. The DefAttributes will apply to this text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->SelAttributes->Color = clRed;
  RichEdit1->SelAttributes->Height += 5;
  RichEdit1->Lines->Add("This line of text will be red. ");
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  RichEdit1->DefAttributes->Color = clBlue;
  RichEdit1->DefAttributes->Style =
    RichEdit1->DefAttributes->Style << fsBold << fsItalic;
}
```

Delphi Examples:

```
{
This example requires a TRichEdit and two TButtons. Leave
the default text, usually "RichEdit1", in the lines
property. The DefAttributes will apply to this text.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  with RichEdit1.SelAttributes do
  begin
    Color := clRed;
    Height := Height + 5;
  end;
  RichEdit1.Lines.Add('This line of text will be red.');
end;


procedure TForm1.Button2Click(Sender: TObject);
begin
  RichEdit1.DefAttributes.Color := clBlue;
  RichEdit1.DefAttributes.Style := [fsBold, fsItalic];
end;
```

See Also

ConsistentAttributes (see page 4989)

Size (see page 4992)

27.1.50.4.5 TTextAttributes.Name Property

Specifies the typeface of the font described by the TTextAttributes (see page 4982) object.

Pascal

```
property Name: TFontName;
```

C++

```
__property TFontName Name;
```

Description

Use Name to specify the typeface of the font. Typefaces are strings of up to 32 characters used by Windows to identify a font. The Name is the identifier for the font that most font selection controls show.

See Also

ConsistentAttributes (see page 4989)

Pitch (see page 4991)

Charset (see page 4987)

27.1.50.4.6 TTextAttributes.Pitch Property

Specifies whether the characters in the font all have the same width.

Pascal

```
property Pitch: TFontPitch;
```

C++

```
__property TFontPitch Pitch;
```

Description

Each typeface has an associated pitch. The characters in fixed-pitch fonts are all the same width. The characters in variable-pitch fonts differ in width. Fixed pitch fonts are generally used for representing source code, because they allow the indentation to line up properly. Variable pitch fonts are generally used to give text a more natural appearance.

Here are the possible values for Pitch:

Value	Meaning
fpDefault	The font pitch is set to the default value, which depends on the font specified in the Name (see page 4991) property.
fpFixed	The font pitch is set to fixed. All characters in the font have the same width.
fpVariable	The font pitch is set to variable. The characters in the font have different widths.

Note: Setting the Pitch of a fixed-pitch font to fpVariable or a variable-pitch font to fpFixed will cause Windows to find what it considers the best match to all of the font characteristics that have been specified. This might have no effect on the appearance of a font, or might cause Windows to substitute a different font with the appropriate pitch. For example, setting the pitch of MS Serif (a variable-pitch font) to fpFixed causes Courier to be displayed.

See Also

Name (see page 4991)

27.1.50.4.7 TTextAttributes.Protected Property

Indicates whether the text represented by the TTextAttributes (see page 4982) object is protected.

Pascal

```
property Protected: Boolean;
```

C++

```
__property Boolean Protected;
```

Description

Protected text can't be changed by the user. When the user attempts to edit a selection that includes protected text, the rich edit control calls the OnProtectChange event handler. The OnProtectChange event handler can then permit or disallow the editing of the protected text. If there is no OnProtectChange event handler, protected text is read-only.

See Also

ConsistentAttributes (see page 4989)

OnProtectChange (see page 3888)

27.1.50.4.8 TTextAttributes.Size Property

Specifies the height of the font in points.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

To specify a font height in points, use the Size property. To specify the size of the font in pixels instead, use the Height (see page 4989) property. Users usually specify font size in points within an application, while programmers are usually concerned with the actual size of the font in pixels when displaying a font on the screen.

The relationship between the Height (see page 4989) and Size properties is expressed by the formula:

```
Size = Height * 72 / ScreenPixelsPerInch
```

Note: Unlike the Size property of a TFont object, the Size property of a TTextAttributes (see page 4982) object always omits the internal leading that appears at the top of the font.

See Also

ConsistentAttributes (see page 4989)

Height (see page 4989)

27.1.50.4.9 TTextAttributes.Style Property

Determines whether the font is normal, italic, underlined, bold, or strikeout.

Pascal

```
property Style: TFontStyles;
```

C++

```
__property TFontStyles Style;
```

Description

The Style property is a set drawn from the following values:

Value	Meaning
fsBold	The font is boldfaced.
fsItalic	The font is italicized.
fsUnderline	The font is underlined.
fsStrikeOut	The font is displayed with a horizontal line through it.

Because the Style property is a set, it can contain multiple values. For example, a font could be both boldfaced and italicized.

C++ Examples:

```
/*
This example requires a TRichEdit and two TButtons. Leave
the default text, usually "RichEdit1", in the lines
property. The DefAttributes will apply to this text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->SelAttributes->Color = clRed;
  RichEdit1->SelAttributes->Height += 5;
  RichEdit1->Lines->Add("This line of text will be red. ");
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  RichEdit1->DefAttributes->Color = clBlue;
  RichEdit1->DefAttributes->Style =
    RichEdit1->DefAttributes->Style << fsBold << fsItalic;
}
```

Delphi Examples:

```
{
This example requires a TRichEdit and two TButtons. Leave
the default text, usually "RichEdit1", in the lines
property. The DefAttributes will apply to this text.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  with RichEdit1.SelAttributes do
  begin
    Color := clRed;
    Height := Height + 5;
  end;
  RichEdit1.Lines.Add('This line of text will be red.');
end;


procedure TForm1.Button2Click(Sender: TObject);
begin
  RichEdit1.DefAttributes.Color := clBlue;
  RichEdit1.DefAttributes.Style := [fsBold, fsItalic];
end;
```

27.1.51 TToolBar Class

TToolBar manages tool buttons and other controls, arranging them in rows and automatically adjusting their sizes and positions.

Class Hierarchy

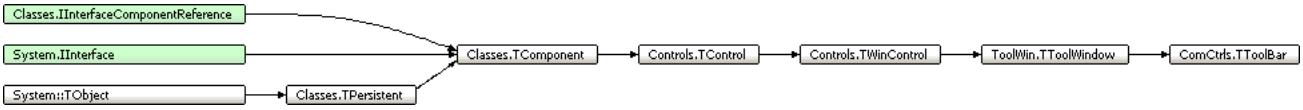

Pascal

```
TToolBar = class(TToolWindow);
```

C++

```
class TToolBar : public TToolWindow;
```

File

ComCtrls

Description

TToolBar is a container for tool buttons (TToolButton (see page 5107)). It provides an easy way to arrange and manage visual controls.

All tool buttons on a toolbar maintain a uniform width and height.

Other controls can sit on a toolbar. These controls (which are held in place by invisible tool buttons) maintain a uniform height.

Controls (see page 5711) can automatically wrap around and start a new row when they do not fit horizontally on the toolbar.

The Flat (see page 5042) property allows the background to show through the toolbar and gives pop-up borders to the tool buttons.

Spaces and dividers (which are in fact specially configured tool buttons) can group controls on the toolbar both visually and functionally.

Typically, the tool buttons correspond to items in an application's menu and give the user more direct access to the application's commands.

See Also

TToolButton (see page 5107)

TCoolBar (see page 3603)

TToolDockObject (see page 23271)

Buttons and Similar Controls

Toolbars

Adding a Cool Bar Component

Adding a Speed Button to a Panel

Adding a Toolbar Using a Panel Component

Adding a Toolbar Using the Toolbar Component

Adding a Tool Button

Adding Hidden Toolbars

Allowing Toggle Buttons

Allowing Toggled Tool Buttons

Assigning a Menu to a Tool Button

Assigning a Speed Button's Glyph

Assigning Images to Tool Buttons

Creating a Group of Speed Buttons

Creating Groups of Tool Buttons

Designing Toolbars and Cool Bars

Hiding and Showing Toolbars

Responding to Clicks

Setting the Initial Condition of a Speed Button

Setting Tool Button Appearance and Initial Conditions

27.1.51.1 TToolBar Members

The following tables list the members exposed by TToolBar.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 5009)	Aligns any child controls of the toolbar within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5010)	Indicates whether the toolbar can resize itself automatically to accommodate its contents.
	CancelMenu (see page 5010)	Called when the menu associated with a toolbar button is canceled.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	ChangeScale (see page 5010)	Repositions and resizes the toolbar by a specified ratio.
	CheckMenuDropdown (see page 5011)	Displays a toolbar button's drop-down menu.
	Click (see page 5743)	Respond to user click.
	ClickButton (see page 5011)	Simulates the clicking of a toolbar button.
	CMBorderChanged (see page 23285)	This is CMBorderChanged, a member of class TToolWindow.
	CMCtl3DChanged (see page 23286)	This is CMCtl3DChanged, a member of class TToolWindow.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 5012)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 5012)	Creates the window that implements the toolbar.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
V	CustomDraw (see page 5012)	Generates OnCustomDraw (see page 5066) and OnAdvancedCustomDraw (see page 5062) events.
V	CustomDrawButton (see page 5013)	Generates OnCustomDrawButton (see page 5067) and OnAdvancedCustomDrawButton (see page 5063) events.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.

	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoQueryDelete (see page 5013)	Generates an OnCustomizeCanDelete (see page 5068) event.
	DoQueryInsert (see page 5014)	Generates an OnCustomizeCanInsert (see page 5069) event.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindButtonFromAccel (see page 5014)	Returns the toolbar button with the specified accelerator.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 5015)	Allows the streaming system to load and save child controls in the order they appear in the toolbar.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.

	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GradientDrawButton (see page 5016)	Use this method for drawing toolbuttons when gdoGradient and dsGradient are both set.
	GradientDrawToolBar (see page 5016)	Use this method for drawing the toolbar when gdoGradient and dsGradient are both set.
	InitMenu (see page 5016)	Called when the menu associated with a drop-down toolbar button is about to be displayed.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsCustomDrawn (see page 5016)	Indicates whether an event handler exists to respond at a particular stage in the paint process.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 5017)	Initializes the component after the form file has been read into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	NCPaint (see page 23286)	This is NCPaint, a member of class TToolWindow.
	Notification (see page 5017)	Responds to a notification that a component has been added or is about to be deleted.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.

	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RepositionButton (see page 5018)	Updates the position of a specified button in the toolbar.
	RepositionButtons (see page 5018)	Updates the positions of all buttons in the toolbar from a specified button on.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5019)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5019)	This is SetAutoSize, a member of class TToolBar.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMNCCalcSize (see page 23286)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCPaint (see page 23286)	This is WMNCPaint, a member of class TToolWindow.
	WndProc (see page 5020)	Provides specific message responses for the toolbar.
	WrapButtons (see page 5020)	Rearranges the toolbar's buttons so they fit in the toolbar.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.

	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 5006)	Creates and initializes a TToolBar (see page 4994) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5008)	Destroys the TToolBar (see page 4994) instance and frees its memory.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
V	FlipChildren (see page 5014)	Prevents the toolbar buttons from reversing position when the parent control is flipped.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 5015)	Returns a TComponent enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.

	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	TrackMenu (see page 5019)	Called when the accelerator for a toolbar button is pressed.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	ButtonCount (see page 5027)	Gives the number of buttons in the toolbar.
R	Buttons (see page 5029)	Lists the tool buttons (TToolButton (see page 5107)) in the toolbar.
R	Canvas (see page 5029)	Specifies the drawing surface for the toolbar.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	CustomizeKeyName (see page 5036)	Specifies the registry key where the tool bar saves its current button configuration before the user changes it using the toolbar customize dialog.
	CustomizeValueName (see page 5036)	Specifies the registry value under which the tool bar saves its current button configuration before the user changes it using the toolbar customize dialog.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.

	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	RowCount (see page 5054)	Indicates the number of rows in the toolbar.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAdvancedCustomDraw (see page 5062)	Occurs at discrete stages during the painting of the toolbar's background.
	OnAdvancedCustomDrawButton (see page 5063)	Occurs at discrete stages during the painting of the toolbar's buttons.
	OnClick (see page 5064)	Occurs when the user clicks the control.
	OnContextPopup (see page 5066)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCustomDraw (see page 5066)	Occurs when the toolbar must be painted.
	OnCustomDrawButton (see page 5067)	Occurs when one of the buttons in the toolbar must be painted.
	OnCustomizeAdded (see page 5068)	Occurs when the user adds a button to the toolbar.
	OnCustomizeCanDelete (see page 5068)	Occurs when user tries to delete one of the buttons on the toolbar.
	OnCustomizeCanInsert (see page 5069)	Occurs when user tries to add a button to the toolbar.
	OnCustomized (see page 5069)	Occurs when the user finishes making changes to the toolbar.
	OnCustomizeDelete (see page 5070)	Occurs when the user deletes a button from the toolbar.
	OnCustomizeNewButton (see page 5070)	Occurs when the user tries to add a new button to the tool bar.
	OnCustomizeReset (see page 5071)	Occurs when the user cancels a customization to the toolbar.
	OnCustomizing (see page 5072)	Occurs when the user displays the toolbar customize dialog in order to make changes to the toolbar.
	OnDblClick (see page 5072)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 5073)	Occurs when another control is docked to the control.
	OnDockOver (see page 5075)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5077)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5079)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5081)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

	OnEndDrag (see page 5082)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 5083)	Occurs when a control receives the input focus.
	OnExit (see page 5085)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 5087)	Returns the control's docking information.
	OnMouseActivate (see page 5089)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5090)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5092)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5092)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5093)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5095)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 5097)	Occurs immediately after the control is resized.
	OnStartDock (see page 5098)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 5098)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 5099)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 5022)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AllowTextButtons (see page 5025)	Determines whether tool buttons that consist only of text can be placed on the Toolbar.
	Anchors (see page 5026)	Specifies how the control is anchored to its parent.
	AutoSize (see page 5026)	Indicates whether the toolbar's height changes automatically to accommodate the controls it contains.
	BorderWidth (see page 5026)	Specifies the width of the tool bar's border.
	ButtonHeight (see page 5028)	Specifies the height of the controls in the toolbar.
	ButtonWidth (see page 5029)	Specifies the width of the tool buttons (TToolButton (see page 5107)) in the toolbar.
	Caption (see page 5030)	Specifies a text string that identifies the control to the user.
	Color (see page 5033)	Specifies the background color of the control.
	Constraints (see page 5034)	Specifies the size constraints for the control.
	Ctl3D (see page 5035)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Customizable (see page 5035)	Specifies whether the user can customize the toolbar at runtime.
	DisabledImages (see page 5037)	Lists the images that can appear on disabled tool buttons.
	DockSite (see page 5037)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 5038)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5039)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5039)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DrawingStyle (see page 5040)	Stores the style type.
	EdgeBorders (see page 5041)	Determines which sides of the toolbar have borders.
	EdgeInner (see page 5041)	Determines the border style for the toolbar's inner edge.
	EdgeOuter (see page 5041)	Determines the border style for the toolbar's outer edge.
	Enabled (see page 5042)	Controls whether the control responds to mouse, keyboard, and timer events.
	Flat (see page 5042)	Makes the toolbar transparent and eliminates tool-button borders.
	Font (see page 5043)	Controls the attributes of text written on or in the control.
	GradientDirection (see page 5044)	Specifies whether the direction of the gradient is horizontal or vertical.
	GradientDrawingOptions (see page 5044)	Stores the drawing option chosen.

	GradientEndColor (see page 5044)	Color (see page 5033) used as end color for the gradient.
	GradientStartColor (see page 5044)	Color (see page 5033) used as start color for the gradient.
	Height (see page 5045)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	HideClippedButtons (see page 5046)	Specifies handling of buttons obscured by adjacent toolbar.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HotImages (see page 5046)	Lists the images that can appear on tool buttons when the mouse points to them.
	HotTrackColor (see page 5047)	Color (see page 5033) used as hot tracking color when in gradient style.
	Images (see page 5047)	Lists the images that can appear on tool buttons.
	Indent (see page 5047)	Specifies the margin on the left side of the toolbar.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	List (see page 5048)	Aligns captions on the right and images on the left in tool buttons.
	Margins (see page 5831)	Specifies the margins for the control.
	Menu (see page 5048)	Causes the toolbar to populate itself with buttons that correspond to items in a menu.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentColor (see page 5048)	Determines where a control looks for its color information.
	ParentFont (see page 5049)	Determines where a control looks for its font information.
	ParentShowHint (see page 5049)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 5050)	Identifies the pop-up menu associated with the control.
	ShowCaptions (see page 5055)	Determines whether text captions are displayed on tool buttons.
	ShowHint (see page 5057)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 5058)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 5059)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Transparent (see page 5059)	Specifies whether the toolbar is transparent.
	Visible (see page 5060)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	Wrapable (see page 5061)	Enables automatic line-wrap for components in the toolbar.

27.1.51.2 TToolBar.Create Constructor

Creates and initializes a TToolBar (see page 4994) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TToolBar(TComponent * AOwner);
```

Description

Call Create to create a toolbar at runtime. Toolbars added to forms at design time are created automatically.

AOwner is the component, typically the form, that is responsible for freeing the toolbar instance. It becomes the value of the Owner property.

C++ Examples:

```
/*
```

```
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
```

```
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ToolBar1 := TToolBar.Create(Self);
    ToolBar1.Parent := Self;
    AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
    ToolBar1.ShowCaptions := True;
    ToolBar1.Height := 40;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I: Integer;
  begin
    for I := ToolBar1.ButtonCount - 1 downto 0 do
      ToolBar1.Buttons[I].Free;
    ToolBar1.Free;
  end;
```

See Also

Owner (see page 2807)

27.1.51.3 TToolBar.Destroy Destructor

Destroys the TToolBar (see page 4994) instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TToolBar();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TToolBar (see page 4994) reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

27.1.51.4 TToolBar Methods

The methods of the TToolBar class are listed here.

Protected Methods

	Name	Description
	AlignControls (see page 5009)	Aligns any child controls of the toolbar within a specified area of the control.
	CanAutoSize (see page 5010)	Indicates whether the toolbar can resize itself automatically to accommodate its contents.
	CancelMenu (see page 5010)	Called when the menu associated with a toolbar button is canceled.
	ChangeScale (see page 5010)	Repositions and resizes the toolbar by a specified ratio.
	CheckMenuDropdown (see page 5011)	Displays a toolbar button's drop-down menu.
	ClickButton (see page 5011)	Simulates the clicking of a toolbar button.
	CreateParams (see page 5012)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateWnd (see page 5012)	Creates the window that implements the toolbar.

	CustomDraw (see page 5012)	Generates OnCustomDraw (see page 5066) and OnAdvancedCustomDraw (see page 5062) events.
	CustomDrawButton (see page 5013)	Generates OnCustomDrawButton (see page 5067) and OnAdvancedCustomDrawButton (see page 5063) events.
	DoQueryDelete (see page 5013)	Generates an OnCustomizeCanDelete (see page 5068) event.
	DoQueryInsert (see page 5014)	Generates an OnCustomizeCanInsert (see page 5069) event.
	FindButtonFromAccel (see page 5014)	Returns the toolbar button with the specified accelerator.
	GetChildren (see page 5015)	Allows the streaming system to load and save child controls in the order they appear in the toolbar.
	GradientDrawButton (see page 5016)	Use this method for drawing toolbuttons when gdoGradient and dsGradient are both set.
	GradientDrawToolBar (see page 5016)	Use this method for drawing the toolbar when gdoGradient and dsGradient are both set.
	InitMenu (see page 5016)	Called when the menu associated with a drop-down toolbar button is about to be displayed.
	IsCustomDrawn (see page 5016)	Indicates whether an event handler exists to respond at a particular stage in the paint process.
	Loaded (see page 5017)	Initializes the component after the form file has been read into memory.
	Notification (see page 5017)	Responds to a notification that a component has been added or is about to be deleted.
	RepositionButton (see page 5018)	Updates the position of a specified button in the toolbar.
	RepositionButtons (see page 5018)	Updates the positions of all buttons in the toolbar from a specified button on.
	Resize (see page 5019)	Respond to control resize.
	SetAutoSize (see page 5019)	This is SetAutoSize, a member of class TToolBar.
	WndProc (see page 5020)	Provides specific message responses for the toolbar.
	WrapButtons (see page 5020)	Rearranges the toolbar's buttons so they fit in the toolbar.

Public Methods

	Name	Description
	FlipChildren (see page 5014)	Prevents the toolbar buttons from reversing position when the parent control is flipped.
	GetEnumerator (see page 5015)	Returns a TComponent enumerator.
	TrackMenu (see page 5019)	Called when the accelerator for a toolbar button is pressed.

27.1.51.4.1 TToolBar.AlignControls Method

Aligns any child controls of the toolbar within a specified area of the control.

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

The protected AlignControls method is called automatically to align all controls within an area of container control (in this case the tool bar). In TToolBar (see page 4994), AlignControls takes into account the special needs of a tool bar in arranging its buttons.

Control is a child control that should have precedence when laying out the child controls. If Control is nil (Delphi) or NULL (C++), none of the child controls has precedence.

Rect is the area in which to align the child controls.

See Also

Align (see page 5798)

Buttons (see page 5029)

27.1.51.4.2 TToolBar.CanAutoSize Method

Indicates whether the toolbar can resize itself automatically to accommodate its contents.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize (see page 5026) property is true and an attempt is made to resize the control. It allows the toolbar to implement the AutoSize (see page 5026) property before the resize sequence that ends with the OnResize (see page 5097) event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the toolbar. CanAutoSize adjusts these values so that the toolbar's size adjusts to its buttons. If the adjustment results in values of NewWidth and NewHeight that differ from the control's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width and Height (see page 5045), CanAutoSize returns false, and the resize sequence is aborted.

See Also

AutoSize (see page 5803)

OnResize (see page 5870)

Width (see page 5849)

Height (see page 5825)

27.1.51.4.3 TToolBar.CancelMenu Method

Called when the menu associated with a toolbar button is canceled.

Pascal

```
procedure CancelMenu; dynamic;
```

C++

```
__fastcall CancelMenu();
```

Description

Override CancelMenu in a derived class to handle the canceling of a menu on one of the tool buttons.

By default, CancelMenu clears the highlighting and hot item in the menu.

See Also

HotImages (see page 5046)

DropdownMenu (see page 5131)

CheckMenuDropdown (see page 5011)

27.1.51.4.4 TToolBar.ChangeScale Method

Repositions and resizes the toolbar by a specified ratio.

Pascal

```
procedure ChangeScale(M: Integer; D: Integer); override;
```

C++

```
virtual __fastcall ChangeScale(int M, int D);
```

Description

Use ChangeScale to change the scale of the toolbar.

ChangeScale modifies the position and size of a toolbar as well as rescaling the buttons. Thus, ChangeScale modifies the Top, Left, Width, and Height (see page 5045) properties.

The M and D parameters define a fraction by which to scale the control. The M parameter is the multiplier and the D parameter is the divisor. For example, to make a control 75% of its original size, specify the value of M as 75, and the value of D as 100 (75/100). Alternately, the same results are achieved by specifying the value of M as 3, and the value of D as 4 (3/4). Both fractions are equal and result in the control being scaled by the same amount, 75%.

See Also

Top (see page 5844)

Left (see page 5830)

Width (see page 5849)

Height (see page 5825)

27.1.51.4.5 TToolBar.CheckMenuDropdown Method

Displays a toolbar button's drop-down menu.

Pascal

```
function CheckMenuDropdown(Button: TToolButton): Boolean; dynamic;
```

C++

```
__fastcall Boolean CheckMenuDropdown(TToolButton Button);
```

Description

CheckMenuDropdown is called automatically when the drop-down arrow portion of a toolbutton is clicked or if a toolbutton has an assigned MenuItem property.

Button is the drop-down toolbar button that was clicked.

CheckMenuDropdown has no effect unless the button's Style property contains tbsDropDown.

See Also

CancelMenu (see page 5010)

DropdownMenu (see page 5131)

27.1.51.4.6 TToolBar.ClickButton Method

Simulates the clicking of a toolbar button.

Pascal

```
procedure ClickButton(Button: TToolButton); dynamic;
```

C++

```
__fastcall ClickButton(TToolButton Button);
```

Description

Call ClickButton to simulate the clicking of a toolbar button.

Button is the toolbar button to click.

27.1.51.4.7 TToolBar.CreateParams Method

Initializes a window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

CreateParams is called automatically to initialize the window-creation parameters whenever the window for the toolbar needs to be created. CreateParams overrides the inherited method to implement those properties that require a specialize window style.

See Also

CreateParams (see page 6135)

27.1.51.4.8 TToolBar.CreateWnd Method

Creates the window that implements the toolbar.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

The Windows control for the toolbar can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. TToolBar (see page 4994) calls CreateWnd to create this Windows control when necessary.

See Also

CreateWnd (see page 6137)

27.1.51.4.9 TToolBar.CustomDraw Method

Generates OnCustomDraw (see page 5066) and OnAdvancedCustomDraw (see page 5062) events.

Pascal

```
function CustomDraw(const ARect: TRect; Stage: TCustomDrawStage): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CustomDraw(const TRect ARect, TCustomDrawStage Stage);
```

Description

CustomDraw is called automatically at discrete stages during the drawing process to generate the OnCustomDraw (see page

5066) and OnAdvancedCustomDraw (see page 5062) events.

ARect specifies the boundaries of the toolbar on the toolbar's Canvas (see page 5029).

Stage indicates the current stage of the paint process. When Stage is cdPrePaint, CustomDraw generates an OnCustomDraw (see page 5066) event followed by an OnAdvancedCustomDraw (see page 5062) event. At all other stages, CustomDraw generates only the OnAdvancedCustomDraw (see page 5062) event.

CustomDraw returns true if the toolbar should continue with its default painting sequence, false if the painting sequence should end after the call.

See Also

OnCustomDraw (see page 5066)

OnAdvancedCustomDraw (see page 5062)

27.1.51.4.10 TToolBar.CustomDrawButton Method

Generates OnCustomDrawButton (see page 5067) and OnAdvancedCustomDrawButton (see page 5063) events.

Pascal

```
function CustomDrawButton(Button: TToolButton; State: TCustomDrawState; Stage:
TCustomDrawStage; var Flags: TTBCustomDrawFlags): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CustomDrawButton(TToolButton Button, TCustomDrawState State,
TCustomDrawStage Stage, TTBCustomDrawFlags Flags);
```

Description

CustomDrawButton is called automatically at discrete stages during the drawing process to generate the OnCustomDrawButton (see page 5067) and OnAdvancedCustomDrawButton (see page 5063) events.

Button specifies the tool button that needs to be drawn.

State describes the state of the button, so that the event handler can reflect that state in the button's image.

Stage indicates the current stage of the paint process. When Stage is cdPrePaint, CustomDrawButton generates an OnCustomDrawButton (see page 5067) event followed by an OnAdvancedCustomDrawButton (see page 5063) event. At all other stages, CustomDrawButton generates only the OnAdvancedCustomDrawButton (see page 5063) event.

Flags is used only when Stage is cdPrePaint. It influences how the button is subsequently rendered by the default drawing process. CustomDrawButton returns the set of flags that indicates how the toolbar should proceed with rendering the button.

See Also

OnCustomDrawButton (see page 5067)

OnAdvancedCustomDrawButton (see page 5063)

27.1.51.4.11 TToolBar.DoQueryDelete Method

Generates an OnCustomizeCanDelete (see page 5068) event.

Pascal

```
function DoQueryDelete(Index: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoQueryDelete(int Index);
```

Description

DoQueryDelete is called automatically when the user tries to delete a button from a customizable tool bar. It generates an OnCustomizeCanDelete (see page 5068) event, allowing the application to block the deletion.

Index is the index (in the Buttons (see page 5029) property) of the button that the user is trying to delete.

See Also

OnCustomizeCanDelete (see page 5068)

Buttons (see page 5029)

27.1.51.4.12 TToolBar.DoQueryInsert Method

Generates an OnCustomizeCanInsert (see page 5069) event.

Pascal

```
function DoQueryInsert(Index: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoQueryInsert(int Index);
```

Description

DoQueryInsert is called automatically when the user tries to add a button to a customizable tool bar. It generates an OnCustomizeCanInsert (see page 5069) event, allowing the application to block the insertion.

Index is the index (in the Buttons (see page 5029) property) where the user is trying to insert a new button.

See Also

OnCustomizeCanInsert (see page 5069)

Buttons (see page 5029)

27.1.51.4.13 TToolBar.FindButtonFromAccel Method

Returns the toolbar button with the specified accelerator.

Pascal

```
function FindButtonFromAccel(Accel: Word): TToolButton;
```

C++

```
__fastcall TToolButton FindButtonFromAccel(Word Accel);
```

Description

Use FindButtonFromAccel to determine which toolbar button has the specified accelerator.

If the accelerator is not found, FindButtonFromAccel returns nil (Delphi) or NULL (C++).

See Also

ClickButton (see page 5011)

27.1.51.4.14 TToolBar.FlipChildren Method

Prevents the toolbar buttons from reversing position when the parent control is flipped.

Pascal

```
procedure FlipChildren(AllLevels: Boolean); override;
```

C++

```
virtual __fastcall FlipChildren(Boolean AllLevels);
```

Description

FlipChildren overrides the inherited method in order to prevent the flipping of the toolbar's children.

See Also

FlipChildren (see page 6150)

27.1.51.4.15 TToolBar.GetChildren Method

Allows the streaming system to load and save child controls in the order they appear in the toolbar.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is introduced in TComponent for the streaming system that loads and saves VCL components. It executes the callback specified by Proc for all the button in the toolbar, as well as any other controls that have the toolbar as a parent and the component specified by Root as an Owner.

Proc is a callback that the streaming system uses to load or save persistent objects on a form.

Root is the the top-level component (such as a form or data module) that is currently being loaded or saved.

See Also

GetChildren (see page 6153)

GetChildren

GetParentComponent

27.1.51.4.16 TToolBar.GetEnumerator Method

Returns a TComponent enumerator.

Pascal

```
function GetEnumerator: TToolBarEnumerator;
```

C++

```
__fastcall TToolBarEnumerator GetEnumerator();
```

Description

GetEnumerator returns a TComponentEnumerator reference, which enumerates the components contained within a specified containing component.

To process all these sub components, call the TComponentEnumerator GetCurrent method within a While MoveNext do loop.

See Also

TComponentEnumerator (see page 2812)

27.1.51.4.17 TToolBar.GradientDrawButton Method

Use this method for drawing toolbuttons when gdoGradient and dsGradient are both set.

Pascal

```
function GradientDrawButton(Button: TToolButton; State: TCustomDrawState; var Flags:
TTBCustomDrawFlags): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GradientDrawButton(TToolButton Button, TCustomDrawState State,
TTBCustomDrawFlags Flags);
```

Description

This method is used for drawing toolbuttons when gdoGradient and dsGradient are both set.

27.1.51.4.18 TToolBar.GradientDrawToolBar Method

Use this method for drawing the toolbar when gdoGradient and dsGradient are both set.

Pascal

```
function GradientDrawToolBar(const ARect: TRect): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GradientDrawToolBar(const TRect ARect);
```

Description

This method is used for drawing the toolbar when gdoGradient and dsGradient are both set

27.1.51.4.19 TToolBar.InitMenu Method

Called when the menu associated with a drop-down toolbar button is about to be displayed.

Pascal

```
procedure InitMenu(Button: TToolButton); dynamic;
```

C++

```
__fastcall InitMenu(TToolButton Button);
```

Description

InitMenu initializes the highlighting and hot-tracking for the drop-down menu associated with the tool button that Button specifies.

Override InitMenu in a derived class to perform other initializations to the toolbar button's menu.

See Also

DropdownMenu (see page 5131)

27.1.51.4.20 TToolBar.IsCustomDrawn Method

Indicates whether an event handler exists to respond at a particular stage in the paint process.

Pascal

```
function IsCustomDrawn(Target: TCustomDrawTarget; Stage: TCustomDrawStage): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsCustomDrawn(TCustomDrawTarget Target, TCustomDrawStage Stage);
```

Description

TToolBar (see page 4994) uses IsCustomDrawn internally to determine whether it needs generate the various custom draw events.

Target indicates what object is currently being painted. If Target is dtControl, it is the background of the toolbar, and IsCustomDrawn checks whether there is an OnCustomDraw (see page 5066) or OnAdvancedCustomDraw (see page 5062) event handler. If Target is dtItem, it is one of the tool buttons, and IsCustomDrawn checks for an OnCustomDrawButton (see page 5067) or OnAdvancedCustomDrawButton (see page 5063) event handler. TToolBar (see page 4994) does not use dtSubItem as a target.

Stage indicates the current stage of the paint process. If stage is not cdPrePaint, then the tool bar only generates an event if it has one of the advanced custom draw event handlers.

IsCustomDrawn returns true if the toolbar should generate an event (there is an event handler to respond.) IsCustomDrawn returns false if there is no need to generate a custom draw event.

See Also

OnCustomDraw (see page 5066)

OnAdvancedCustomDraw (see page 5062)

OnCustomDrawButton (see page 5067)

OnAdvancedCustomDrawButton (see page 5063)

27.1.51.4.21 TToolBar.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

TToolBar (see page 4994) overrides Loaded in order to managed the ButtonWidth (see page 5029) property.

See Also

Loaded (see page 2782)

27.1.51.4.22 TToolBar.Notification Method

Responds to a notification that a component has been added or is about to be deleted.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when a component has just been added to the form or is about to be deleted.

AComponent is the component that was just added or that is about to be deleted.

Operation indicates whether AComponent was added or is about to be deleted.

As implemented in TToolBar (see page 4994), Notification checks whether the image list that is the value of the Images (see page 5047), HotImages (see page 5046), or DisabledImages (see page 5037) property is about to be deleted. If so, it resets the property to nil (Delphi) or NULL (C++).

See Also

Notification (see page 2783)

Images (see page 5047)

HotImages (see page 5046)

DisabledImages (see page 5037)

27.1.51.4.23 TToolBar.RepositionButton Method

Updates the position of a specified button in the toolbar.

Pascal

```
procedure RepositionButton(Index: Integer);
```

C++

```
__fastcall RepositionButton(int Index);
```

Description

TToolBar (see page 4994) uses RepositionButton internally when laying out the buttons in the toolbar. RepositionButton calculates where to place the button with the specified index, given the positions of all previous buttons in the toolbar. It then changes the position and size of the specified tool button.

See Also

RepositionButtons (see page 5018)

Buttons (see page 5029)

27.1.51.4.24 TToolBar.RepositionButtons Method

Updates the positions of all buttons in the toolbar from a specified button on.

Pascal

```
procedure RepositionButtons(Index: Integer);
```

C++

```
__fastcall RepositionButtons(int Index);
```

Description

TToolBar (see page 4994) uses RepositionButtons internally when laying out the buttons in the toolbar. RepositionButtons calculates where to place all the buttons with the specified index or greater, and updates their positions and sizes.

See Also

RepositionButton (see page 5018)

Buttons (see page 5029)

27.1.51.4.25 TToolBar.Resize Method

Respond to control resize.

Pascal

```
procedure Resize; override;
```

C++

```
virtual __fastcall Resize();
```

Description

Resize is called automatically immediately after the control's dimensions change. As implemented in TControl, Resize calls the control's OnResize event handler if it is assigned.

See Also

OnResize (see page 5870)

27.1.51.4.26 TToolBar.SetAutoSize Method

Pascal

```
procedure SetAutoSize(Value: Boolean); override;
```

C++

```
virtual __fastcall SetAutoSize(Boolean Value);
```

Description

This is SetAutoSize, a member of class TToolBar.

27.1.51.4.27 TToolBar.TrackMenu Method

Called when the accelerator for a toolbar button is pressed.

Pascal

```
function TrackMenu(Button: TToolButton): Boolean; dynamic;
```

C++

```
__fastcall Boolean TrackMenu(TToolButton Button);
```

Description

TrackMenu is called automatically in response to the accelerator on a toolbar button. It initializes the menu of a drop-down menu button, and forwards a click to the underlying Windows control.

Button is the toolbar button for which an accelerator was pressed.

Override TrackMenu in a derived class to perform additional actions when the accelerator for a toolbar button is pressed.

See Also

DropdownMenu (see page 5131)

ClickButton (see page 5011)

27.1.51.4.28 TToolBar.WndProc Method

Provides specific message responses for the toolbar.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc overrides the WndProc for TWinControl to handle mouse messages, and sends all others to the inherited WndProc.

Derived classes that override WndProc to provide specialized responses to messages should call the inherited WndProc at the end to dispatch any other messages.

See Also

WndProc (see page 6187)

27.1.51.4.29 TToolBar.WrapButtons Method

Rearranges the toolbar's buttons so they fit in the toolbar.

Pascal

```
function WrapButtons(var NewWidth: Integer; var NewHeight: Integer): Boolean;
```

C++

```
__fastcall Boolean WrapButtons(int NewWidth, int NewHeight);
```

Description

Call WrapButtons to rearrange the toolbar's buttons.

NewWidth is the width in which the buttons must fit.

NewHeight is the height in which the buttons must fit.

WrapButtons returns true if the buttons fit; otherwise, WrapButtons returns false.

27.1.51.5 TToolBar Properties

The properties of the TToolBar class are listed here.

Public Properties

	Name	Description
R	ButtonCount (see page 5027)	Gives the number of buttons in the toolbar.
R	Buttons (see page 5029)	Lists the tool buttons (TToolButton (see page 5107)) in the toolbar.
R	Canvas (see page 5029)	Specifies the drawing surface for the toolbar.
	CustomizeKeyName (see page 5036)	Specifies the registry key where the tool bar saves its current button configuration before the user changes it using the toolbar customize dialog.
	CustomizeValueName (see page 5036)	Specifies the registry value under which the tool bar saves its current button configuration before the user changes it using the toolbar customize dialog.
R	RowCount (see page 5054)	Indicates the number of rows in the toolbar.

Published Properties

	Name	Description
	Align (see page 5022)	Determines how the control aligns within its container (parent control).
	AllowTextButtons (see page 5025)	Determines whether tool buttons that consist only of text can be placed on the Toolbar.
	Anchors (see page 5026)	Specifies how the control is anchored to its parent.
	AutoSize (see page 5026)	Indicates whether the toolbar's height changes automatically to accommodate the controls it contains.
	BorderWidth (see page 5026)	Specifies the width of the tool bar's border.
	ButtonHeight (see page 5028)	Specifies the height of the controls in the toolbar.
	ButtonWidth (see page 5029)	Specifies the width of the tool buttons (TToolButton (see page 5107)) in the toolbar.
	Caption (see page 5030)	Specifies a text string that identifies the control to the user.
	Color (see page 5033)	Specifies the background color of the control.
	Constraints (see page 5034)	Specifies the size constraints for the control.
	Ctl3D (see page 5035)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Customizable (see page 5035)	Specifies whether the user can customize the toolbar at runtime.
	DisabledImages (see page 5037)	Lists the images that can appear on disabled tool buttons.
	DockSite (see page 5037)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 5038)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5039)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5039)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DrawingStyle (see page 5040)	Stores the style type.
	EdgeBorders (see page 5041)	Determines which sides of the toolbar have borders.
	EdgeInner (see page 5041)	Determines the border style for the toolbar's inner edge.
	EdgeOuter (see page 5041)	Determines the border style for the toolbar's outer edge.
	Enabled (see page 5042)	Controls whether the control responds to mouse, keyboard, and timer events.
	Flat (see page 5042)	Makes the toolbar transparent and eliminates tool-button borders.
	Font (see page 5043)	Controls the attributes of text written on or in the control.
	GradientDirection (see page 5044)	Specifies whether the direction of the gradient is horizontal or vertical.
	GradientDrawingOptions (see page 5044)	Stores the drawing option chosen.
	GradientEndColor (see page 5044)	Color (see page 5033) used as end color for the gradient.
	GradientStartColor (see page 5044)	Color (see page 5033) used as start color for the gradient.
	Height (see page 5045)	Specifies the vertical size of the control in pixels.
	HideClippedButtons (see page 5046)	Specifies handling of buttons obscured by adjacent toolbar.
	HotImages (see page 5046)	Lists the images that can appear on tool buttons when the mouse points to them.
	HotTrackColor (see page 5047)	Color (see page 5033) used as hot tracking color when in gradient style.
	Images (see page 5047)	Lists the images that can appear on tool buttons.
	Indent (see page 5047)	Specifies the margin on the left side of the toolbar.
	List (see page 5048)	Aligns captions on the right and images on the left in tool buttons.
	Menu (see page 5048)	Causes the toolbar to populate itself with buttons that correspond to items in a menu.
	ParentColor (see page 5048)	Determines where a control looks for its color information.
	ParentFont (see page 5049)	Determines where a control looks for its font information.
	ParentShowHint (see page 5049)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 5050)	Identifies the pop-up menu associated with the control.
	ShowCaptions (see page 5055)	Determines whether text captions are displayed on tool buttons.
	ShowHint (see page 5057)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 5058)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 5059)	Determines if the user can tab to a control.
	Transparent (see page 5059)	Specifies whether the toolbar is transparent.
	Visible (see page 5060)	Determines whether the component appears onscreen.
	Wrapable (see page 5061)	Enables automatic line-wrap for components in the toolbar.

27.1.51.5.1 TToolBar.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
```

```
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
```

```
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
```

```
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

27.1.51.5.2 TToolBar.AllowTextButtons Property

Determines whether tool buttons that consist only of text can be placed on the Toolbar.

Pascal

```
property AllowTextButtons: Boolean;
```

C++

```
__property Boolean AllowTextButtons;
```

Description

When set to true, the tbsTextButton style is available for the TToolButton (see page 5107) control, allowing you to create tool buttons on the Toolbar that consist only of text. When set to false, the tbsTextButton style is not available for the TToolButton (see page 5107) control.

See Also

Adding a Tool Button

TToolButton (see page 5107)

Style

27.1.51.5.3 TToolBar.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.51.5.4 TToolBar.AutoSize Property

Indicates whether the toolbar's height changes automatically to accommodate the controls it contains.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

If AutoSize is set to true, the height of the toolbar adjusts automatically so that the controls fit in it with no vertical margin.

See Also

Wrapable (see page 5061)

Controls

27.1.51.5.5 TToolBar.BorderWidth Property

Specifies the width of the tool bar's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the tool bar's border in pixels. The controls that the tool bar contains appear within this distance from the edge of the tool bar.

27.1.51.5.6 TToolBar.ButtonCount Property

Gives the number of buttons in the toolbar.

Pascal

```
property ButtonCount: Integer;
```

C++

```
__property int ButtonCount;
```

Description

ButtonCount gives the number of buttons in the Buttons (see page 5029) array.

C++ Examples:

```
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

```
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

See Also

Buttons (see page 5029)

27.1.51.5.7 TToolBar.ButtonHeight Property

Specifies the height of the controls in the toolbar.

Pascal

```
property ButtonHeight: Integer;
```

C++

```
__property int ButtonHeight;
```

Description

ButtonHeight represents the height, in pixels, of the controls in the toolbar.

See Also

ButtonWidth (see page 5029)

27.1.51.5.8 TToolBar.Buttons Property

Lists the tool buttons (TToolButton (see page 5107)) in the toolbar.

Pascal

```
property Buttons [Index: Integer]: TToolButton;
```

C++

```
__property TToolButton Buttons[int Index];
```

Description

Buttons maintains a list of TToolButton (see page 5107) instances. All tool buttons that share a TToolBar (see page 4994) parent have the same height and (except for separators and dividers) the same width. Other controls on a toolbar are held in place by invisible separators, which are automatically created and destroyed.

To add tool buttons to the toolbar at design time, select the toolbar, right-click, and choose New Button. To create a space (separator) between one button and the next, select New Separator. To create a divider between buttons, add a button and set its Style property to tbsDivider. Other controls may be added to the toolbar directly from the Component palette.

See Also

TToolBar.Buttons Property

27.1.51.5.9 TToolBar.ButtonWidth Property

Specifies the width of the tool buttons (TToolButton (see page 5107)) in the toolbar.

Pascal

```
property ButtonWidth: Integer;
```

C++

```
__property int ButtonWidth;
```

Description

ButtonWidth represents the width, in pixels, of the TToolButton (see page 5107) components on the toolbar. It has no effect on tool buttons whose Style property is set to tbsSeparator or tbsDivider.

See Also

ButtonHeight (see page 5028)

Buttons (see page 5029)

TToolButton (see page 5107)

Style (see page 5144)

27.1.51.5.10 TToolBar.Canvas Property

Specifies the drawing surface for the toolbar.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use Canvas to draw or paint on the surface of the toolbar in an OnCustomDraw (see page 5066) or OnCustomDrawItem event handler. The TCanvas object provides the properties and methods needed for drawing and painting.

C++ Examples:

```
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TToolBar *myToolBar;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TToolBar")
  {
    myToolBar = (TToolBar *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myToolBar->Canvas->MoveTo(myToolBar->ClientRect.Left, myToolBar->ClientRect.Top);
    myToolBar->Canvas->LineTo(myToolBar->ClientRect.Right, myToolBar->ClientRect.Bottom);
    myToolBar = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TToolBar) then
  begin
    myToolBar:= (ListBox1.Items.Objects[Index] as TToolBar);
    with myToolBar.ClientRect do
    begin // make sure we are using the TControls canvas and not the from's!
      myToolBar.Canvas.MoveTo(Left, Top);
      myToolBar.Canvas.LineTo(Right, Bottom);
    end;
  myToolBar:= nil;
  end;
```

See Also

OnCustomDraw (see page 5066)

OnCustomDrawButton (see page 5067)

27.1.51.5.11 TToolBar.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
```

```
      button->Style = tbsButton;
    }
  }

  TToolBar *myToolBar;

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    myToolBar = new TToolBar(this);
    myToolBar->Parent = this;
    TStringList *captions = new TStringList();
    captions->Add("New");
    captions->Add("Save");
    captions->Add("|");
    captions->Add("Cut");
    captions->Add("Copy");
    captions->Add("Paste");
    AddButtons(myToolBar, captions);
    delete captions;
    myToolBar->ShowCaptions = True;
    myToolBar->Height = 40;
  }

  void __fastcall TForm1::FormDestroy(TObject *Sender)
  {
    for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
      delete myToolBar->Buttons[I];
    delete myToolBar;
  }
```

Delphi Examples:

```
  {
  The following code keeps the size of the label control
  constant, even though the length of the label's caption
  changes. As a result, the caption of the label is probably
  too long to display in the label when the user clicks the
  button:
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Label1.AutoSize := False;
    Label1.Caption :=
      'This string is too long as the caption of this label';
  end;
  {
  This example requires a TPageControl populated by several
  TabSheets, a TEdit and a TButton. When the button is
  clicked, the caption of the active tab changes to that of
  the TEdit.  These lines of code could also be written:
  PageControl1.ActivePage.Caption := Edit1.Text;
  This would obviate the need for a TTabSheet variable since
  PageControl1.->ActivePage is already of type TTabSheet.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    sheet: TTabSheet;
  begin
    sheet := PageControl1.ActivePage;
    sheet.Caption := Edit1.Text;
  end;
  {
  This example creates a ToolBar and adds ToolButtons to it.
  Include ComCtlrs in the uses clause.
  }
  procedure AddButtons(ToolBar: TToolBar;
    Const ButtonCaptions: array of String);
```

```
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

27.1.51.5.12 TToolBar.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

27.1.51.5.13 TToolBar.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.51.5.14 TToolBar.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

27.1.51.5.15 TToolBar.Customizable Property

Specifies whether the user can customize the toolbar at runtime.

Pascal

```
property Customizable: Boolean;
```

C++

```
__property Boolean Customizable;
```

Description

Set Customizable to true to allow the user to add, delete, and move toolbar buttons at runtime. There are two ways the user can change a customizable toolbar:

By holding the shift key, the user can drag a button to a new position on the tool bar, or drag it off the toolbar entirely to delete the button.

By double-clicking on the toolbar, the user can display the toolbar customize dialog. The toolbar customize dialog shows all the possible buttons in two list boxes, one that contains buttons not on the tool bar, and one that contains buttons currently on the tool bar. The user can use this dialog to add or remove buttons and to change their order.

Note: When Customizable is true, a number of events inform the tool bar component when the user makes changes.

See Also

CustomizeKeyName (see page 5036)

CustomizeValueName (see page 5036)

OnCustomizing (see page 5072)

OnCustomized (see page 5069)

OnCustomizeAdded (see page 5068)

OnCustomizeDelete (see page 5070)

OnCustomizeCanDelete (see page 5068)

OnCustomizeReset (see page 5071)

OnCustomizeNewButton (see page 5070)

27.1.51.5.16 TToolBar.CustomizeKeyName Property

Specifies the registry key where the tool bar saves its current button configuration before the user changes it using the toolbar customize dialog.

Pascal

```
property CustomizeKeyName: string;
```

C++

```
__property AnsiString CustomizeKeyName;
```

Description

Use CustomizeKeyName to specify where the toolbar saves its button configuration when the user first displays the toolbar customize dialog. This is a subkey of HKEY_CURRENT_USER.

When the user double-clicks the tool bar to display the toolbar customize dialog, the tool bar receives an OnCustomizing (see page 5072) event and then automatically saves its current configuration to this key using the name specified by the CustomizeValueName (see page 5036) property. If the user instructs the dialog to restore this configuration, the toolbar fetches the saved configuration from this registry key.

See Also

Customizable (see page 5035)

CustomizeValueName (see page 5036)

OnCustomizing (see page 5072)

OnCustomizeReset (see page 5071)

27.1.51.5.17 TToolBar.CustomizeValueName Property

Specifies the registry value under which the tool bar saves its current button configuration before the user changes it using the toolbar customize dialog.

Pascal

```
property CustomizeValueName: string;
```

C++

```
__property AnsiString CustomizeValueName;
```

Description

Use CustomizeValueName to specify the name that the toolbar uses to save its button configuration when the user first displays the toolbar customize dialog. This is a value under the registry key specified by the CustomizeKeyName (see page 5036) property.

When the user double-clicks the tool bar to display the toolbar customize dialog, the tool bar receives an OnCustomizing (see page 5072) event and then automatically saves its current configuration to the Registry (see page 18633) using this value. If

the user instructs the dialog to restore this configuration, the toolbar fetches this saved configuration.

See Also

Customizable (see page 5035)

CustomizeKeyName (see page 5036)

OnCustomizing (see page 5072)

OnCustomizeReset (see page 5071)

27.1.51.5.18 TToolBar.DisabledImages Property

Lists the images that can appear on disabled tool buttons.

Pascal

```
property DisabledImages: TCustomImageList;
```

C++

```
__property TCustomImageList DisabledImages;
```

Description

DisabledImages contains a list of images that can appear on tool buttons when they are disabled. Each tool button's ImageIndex property determines the specific image displayed on it. If DisabledImages is unassigned, disabled tool buttons display grayed versions of their regular images (specified by the Images (see page 5047) property).

See Also

HotImages (see page 5046)

Images (see page 5047)

Enabled (see page 5821)

ImageIndex (see page 5135)

27.1.51.5.19 TToolBar.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite: Boolean;
```

C++

```
__property Boolean DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
```

```
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

27.1.51.5.20 TToolBar.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

27.1.51.5.21 TToolBar.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

27.1.51.5.22 TToolBar.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
```

```
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

27.1.51.5.23 TToolBar.DrawingStyle Property

Stores the style type.

Pascal

```
property DrawingStyle: TTBDrawingStyle;
```

C++

```
__property TTBDrawingStyle DrawingStyle;
```

Description

This property stores either the dsNormal or dsGradient style choice.

27.1.51.5.24 TToolBar.EdgeBorders Property

Determines which sides of the toolbar have borders.

Pascal

```
property EdgeBorders: TEdgeBorders;
```

C++

```
__property TEdgeBorders EdgeBorders;
```

Description

Use the EdgeBorders property to add or remove borders. The appearance of the borders can be set with the EdgeInner (see page 5041) and EdgeOuter (see page 5041) properties.

27.1.51.5.25 TToolBar.EdgeInner Property

Determines the border style for the toolbar's inner edge.

Pascal

```
property EdgeInner: TEdgeStyle;
```

C++

```
__property TEdgeStyle EdgeInner;
```

Description

Use EdgeInner to provide the toolbar with a three-dimensional beveled look. A toolbar has two borders on each side, an outer border drawn next to the outer edge, and an inner border drawn inside the outer border. EdgeInner specifies the appearance of the inner border. It can be raised (esRaised), lowered (esLowered), or left off altogether (esNone).

Note: The EdgeInner property has an effect only on the sides specified by the EdgeBorders (see page 5041) property.

Use EdgeInner with EdgeOuter (see page 5041) to give the component a three-dimensional look.

See Also

EdgeBorders (see page 5041)

EdgeOuter (see page 5041)

27.1.51.5.26 TToolBar.EdgeOuter Property

Determines the border style for the toolbar's outer edge.

Pascal

```
property EdgeOuter: TEdgeStyle;
```

C++

```
__property TEdgeStyle EdgeOuter;
```

Description

Use EdgeOuter to provide the toolbar with a three-dimensional beveled look. A toolbar has two borders on each side: an outer border drawn next to the outer edge, and an inner border drawn inside the outer border. EdgeOuter specifies the appearance of the outer border. The outer border can be raised (esRaised), lowered (esLowered), or left off altogether (esNone).

Note: The EdgeInner (see page 5041) property has an effect only on the sides specified by the EdgeBorders (see page 5041) property.

Use EdgeOuter with EdgeInner (see page 5041) to give the component a three-dimensional look.

See Also

EdgeBorders (see page 5041)

EdgeInner (see page 5041)

27.1.51.5.27 TToolBar.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.51.5.28 TToolBar.Flat Property

Makes the toolbar transparent and eliminates tool-button borders.

Pascal

```
property Flat: Boolean;
```

C++

```
__property Boolean Flat;
```

Description

If Flat is set to true, the toolbar and its tool buttons are transparent; background colors and graphics show through them. In addition, when Flat is true, tool buttons (TToolButton (see page 5107)) on the toolbar have pop-up borders that appear only

when the mouse is passed over the button.

See Also

Buttons (see page 5029)

HotImages (see page 5046)

TToolButton (see page 5107)

27.1.51.5.29 TToolBar.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;
```

```
procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

27.1.51.5.30 TToolBar.GradientDirection Property

Specifies whether the direction of the gradient is horizontal or vertical.

Pascal

```
property GradientDirection: TGradientDirection;
```

C++

```
__property TGradientDirection GradientDirection;
```

Description

This property stores the value for the gradient direction.

27.1.51.5.31 TToolBar.GradientDrawingOptions Property

Stores the drawing option chosen.

Pascal

```
property GradientDrawingOptions: TTBGradientDrawingOptions;
```

C++

```
__property TTBGradientDrawingOptions GradientDrawingOptions;
```

Description

This property contains either the gdoHotTrack or gdoGradient option, depending on which one was chosen.

27.1.51.5.32 TToolBar.GradientEndColor Property

Color (see page 5033) used as end color for the gradient.

Pascal

```
property GradientEndColor: TColor;
```

C++

```
__property TColor GradientEndColor;
```

Description

This property stores the color that is used to paint the gradient when the drawing ends. (The end color blends with the start color).

27.1.51.5.33 TToolBar.GradientStartColor Property

Color (see page 5033) used as start color for the gradient.

Pascal

```
property GradientStartColor: TColor;
```

C++

```
__property TColor GradientStartColor;
```

Description

The property stores the value that represents the drawing color when the gradient starts painting.

27.1.51.5.34 TToolBar.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
HideSelection, Height example
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
```

```
    I: Integer;
  begin
    for I := 0 to 2 do
      with TRichEdit.Create(Self) do
      begin
        Parent := Self;
        Top := Height * I;
        Text := 'This text will take the focus.';
        SelStart := 0;
        SelLength := Length(Text);
        if (I = 1) then
          HideSelection := False;
      end;
  end;
```

See Also

ClientHeight (see page 5808)

Width (see page 5849)

Using TDBListBox and TDBComboBox

27.1.51.5.35 TToolBar.HideClippedButtons Property

Specifies handling of buttons obscured by adjacent toolbar.

Pascal

```
property HideClippedButtons: Boolean;
```

C++

```
__property Boolean HideClippedButtons;
```

Description

If HideClippedButtons is set to true, any button partially obscured by a neighboring band is hidden.

27.1.51.5.36 TToolBar.HotImages Property

Lists the images that can appear on tool buttons when the mouse points to them.

Pascal

```
property HotImages: TCustomImageList;
```

C++

```
__property TCustomImageList HotImages;
```

Description

HotImages contains a list of images that can appear on tool buttons when the mouse passes over them at runtime. Each tool button's ImageIndex property determines the specific image displayed when the mouse points to it. If HotImages is unassigned, tool buttons retain their regular images (specified by the Images (see page 5047) property) when pointed to. Use a TImageList control to hold the images that will be shown for hot images. Typically, then, the application will have two TImageList controls, one for the Images (see page 5047) property and one for the HotImages property.

To use HotImages set the Flat (see page 5042) property to **True**.

See Also

DisabledImages (see page 5037)

Images (see page 5047)

ImageIndex (see page 5135)

27.1.51.5.37 TToolBar.HotTrackColor Property

Color (see page 5033) used as hot tracking color when in gradient style.

Pascal

```
property HotTrackColor: TColor;
```

C++

```
__property TColor HotTrackColor;
```

Description

This is the color used as the hot tracking color when the toolbar's DrawingStyle (see page 5040) property is set to dsGradient. The hot tracking color is the color the button turns when the mouse moves over it.

27.1.51.5.38 TToolBar.Images Property

Lists the images that can appear on tool buttons.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Images contains a list of images that can appear on the toolbar's tool buttons. Each tool button's ImageIndex property determines the specific image displayed on it. If Images is unassigned, no images are displayed on the tool buttons.

Images in the list should be 16x16. To accommodate every system on which your application runs, it is recommended that you do not use 256 colors.

See Also

DisabledImages (see page 5037)

HotImages (see page 5046)

ImageIndex (see page 5135)

27.1.51.5.39 TToolBar.Indent Property

Specifies the margin on the left side of the toolbar.

Pascal

```
property Indent: Integer;
```

C++

```
__property int Indent;
```

Description

Indent specifies, in pixels, the margin that separates the left edge of the toolbar from the controls within it.

27.1.51.5.40 TToolBar.List Property

Aligns captions on the right and images on the left in tool buttons.

Pascal

```
property List: Boolean;
```

C++

```
__property Boolean List;
```

Description

When List is false, tool buttons (TToolButton (see page 5107)) appear in the toolbar with their images (if any) centered and their captions (if visible) underneath. Setting List to true aligns captions to the right and images to the left.

See Also

Images (see page 5047)

ShowCaptions (see page 5055)

TToolButton (see page 5107)

27.1.51.5.41 TToolBar.Menu Property

Causes the toolbar to populate itself with buttons that correspond to items in a menu.

Pascal

```
property Menu: TMainMenu;
```

C++

```
__property TMainMenu Menu;
```

Description

Use the menu property to populate the tool bar to match a menu component. Setting Menu causes the toolbar to delete all of its current buttons and replace them with tool buttons that match the items in the specified menu component.

In applications that use Menu to populate the toolbar, do not change the Buttons (see page 5029) property by adding, deleting, or editing toolbar buttons.

Note: The toolbar does not dynamically track any changes to the menu specified by Menu. That is, it sets the tool buttons to reflect the set of items in the menu at the point when the application sets this property, and ignores any subsequent changes.

See Also

Buttons (see page 5029)

Items (see page 16084)

27.1.51.5.42 TToolBar.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

27.1.51.5.43 TToolBar.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

27.1.51.5.44 TToolBar.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show

them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.51.5.45 TToolBar.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
```

```
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
```

```
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
```

```
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.51.5.46 TToolBar.RowCount Property

Indicates the number of rows in the toolbar.

Pascal

```
property RowCount: Integer;
```

C++

```
__property int RowCount;
```

Description

Reading RowCount returns the number of rows in which the tool buttons are arranged.

See Also

Wrapable (see page 5061)

27.1.51.5.47 TToolBar.ShowCaptions Property

Determines whether text captions are displayed on tool buttons.

Pascal

```
property ShowCaptions: Boolean;
```

C++

```
__property Boolean ShowCaptions;
```

Description

If ShowCaptions is set to true, tool buttons appear in the toolbar with their captions displayed. Setting ShowCaptions to true may increase the size of the tool buttons.

C++ Examples:

```
/*
To create a toolbar that corresponds to an existing menu:

1   Drop a ToolBar on the form and add a ToolButton for each
  top-level menu item you wish to create.
2   Set the MenuItem property of each ToolButton to correspond
  to the top level menu items.
3   Set the Grouped property of each ToolButton to true.
4   Clear the Menu property of the Form (if it is assigned)
5   Set the Visible property on the ToolBar and ToolButtons to
  true.
6   Set the ShowCaptions property on the ToolBar to true.
7   Set the MenuItem | Visible property of each ToolButton to
  true.
*/
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
```

```
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
To create a toolbar that corresponds to an existing menu:

1   Drop a ToolBar on the form and add a ToolButton for each top-level menu item you wish to
create.
2   Set the MenuItem property of each ToolButton to correspond to the top level menu items.
3   Set the Grouped property of each ToolButton to true.
4   Clear the Menu property of the Form (if it is assigned)
5   Set the Visible property on the ToolBar and ToolButtons to true.
6   Set the ShowCaptions property on the ToolBar to true.
7   Set the MenuItem | Visible property of each ToolButton to true.
}
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

See Also

Images (see page 5047)

Caption (see page 5806)

27.1.51.5.48 TToolBar.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
```

```
    {
      strcpy(string, "Item");
      itoa(i, index, 10);
      strcat(string, index);
      ListBox1->Items->Add(string);
    }
    ListBox1->Hint = "Select an item";
    ListBox1->ShowHint = true;
    Application->ShowHint = true;
  }
  /*
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  */
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.51.5.49 TToolBar.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

27.1.51.5.50 TToolBar.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

27.1.51.5.51 TToolBar.Transparent Property

Specifies whether the toolbar is transparent.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Use Transparent to get or set whether the toolbar is transparent. Note the Transparent property applies only to the toolbar itself—not its buttons.

Tip: To avoid painting problems, set the Transparent property before the toolbar becomes visible.

27.1.51.5.52 TToolBar.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
```

```
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.51.5.53 TToolBar.Wrapable Property

Enables automatic line-wrap for components in the toolbar.

Pascal

```
property Wrapable: Boolean;
```

C++

```
__property Boolean Wrapable;
```

Description

If Wrapable is true, components automatically wrap around and form new rows when they do not fit horizontally in the toolbar.

See Also

RowCount (see page 5054)

Wrap (see page 5147)

27.1.51.6 TToolBar Events

The events of the TToolBar class are listed here.

Published Events

	Name	Description
	OnAdvancedCustomDraw (see page 5062)	Occurs at discrete stages during the painting of the toolbar's background.
	OnAdvancedCustomDrawButton (see page 5063)	Occurs at discrete stages during the painting of the toolbar's buttons.
	OnClick (see page 5064)	Occurs when the user clicks the control.
	OnContextPopup (see page 5066)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCustomDraw (see page 5066)	Occurs when the toolbar must be painted.
	OnCustomDrawButton (see page 5067)	Occurs when one of the buttons in the toolbar must be painted.
	OnCustomizeAdded (see page 5068)	Occurs when the user adds a button to the toolbar.
	OnCustomizeCanDelete (see page 5068)	Occurs when user tries to delete one of the buttons on the toolbar.
	OnCustomizeCanInsert (see page 5069)	Occurs when user tries to add a button to the toolbar.
	OnCustomized (see page 5069)	Occurs when the user finishes making changes to the toolbar.
	OnCustomizeDelete (see page 5070)	Occurs when the user deletes a button from the toolbar.
	OnCustomizeNewButton (see page 5070)	Occurs when the user tries to add a new button to the tool bar.
	OnCustomizeReset (see page 5071)	Occurs when the user cancels a customization to the toolbar.
	OnCustomizing (see page 5072)	Occurs when the user displays the toolbar customize dialog in order to make changes to the toolbar.
	OnDblClick (see page 5072)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 5073)	Occurs when another control is docked to the control.
	OnDockOver (see page 5075)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5077)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5079)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5081)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5082)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 5083)	Occurs when a control receives the input focus.
	OnExit (see page 5085)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 5087)	Returns the control's docking information.
	OnMouseActivate (see page 5089)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5090)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5092)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5092)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5093)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5095)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 5097)	Occurs immediately after the control is resized.
	OnStartDock (see page 5098)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 5098)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 5099)	Occurs when the application tries to undock a control that is docked to the windowed control.

27.1.51.6.1 TToolBar.OnAdvancedCustomDraw Event

Occurs at discrete stages during the painting of the toolbar's background.

Pascal

```
property OnAdvancedCustomDraw: TTBAdvancedCustomDrawEvent;
```

C++

```
__property TTBAdvancedCustomDrawEvent OnAdvancedCustomDraw;
```

Description

Write an OnAdvancedCustomDraw event handler to enhance or replace the automatic painting of the toolbar. Use the toolbar's Canvas (see page 5029) property to render the toolbar's image. OnAdvancedCustomDraw is for painting the background of the toolbar. The painting of individual buttons on the toolbar should be handled in an OnCustomDrawButton (see page 5067) or OnAdvancedCustomDrawButton (see page 5063) event handler.

Sender is the toolbar that needs to be painted.

ARect indicates the dimensions of the toolbar on the canvas.

Stage indicates the current stage of the drawing process.

Leave DefaultDraw as true if the toolbar should continue with the normal painting process. Set DefaultDraw to false if the OnAdvancedCustomDraw event handler has completely painted the toolbar. DefaultDraw is only meaningful when Stage is cdPrePaint.

Note: OnAdvancedCustomDraw occurs at several points during the painting of the toolbar's background. If you only need to respond immediately prior to the default painting of the toolbar, use OnCustomDraw (see page 5066) instead.

See Also

Canvas (see page 5029)

OnCustomDrawButton (see page 5067)

OnCustomDraw (see page 5066)

OnAdvancedCustomDrawButton (see page 5063)

27.1.51.6.2 TToolBar.OnAdvancedCustomDrawButton Event

Occurs at discrete stages during the painting of the toolbar's buttons.

Pascal

```
property OnAdvancedCustomDrawButton: TTBAdvancedCustomDrawBtnEvent;
```

C++

```
__property TTBAdvancedCustomDrawBtnEvent OnAdvancedCustomDrawButton;
```

Description

Write an OnAdvancedCustomDrawButton event handler to enhance or replace the automatic painting of the toolbar's buttons. Use the toolbar's Canvas (see page 5029) property to render the button's image. OnAdvancedCustomDrawButton occurs during the erasure and painting of individual buttons on the toolbar. The painting of the toolbar itself should be handled in an OnCustomDraw (see page 5066) or OnAdvancedCustomDraw (see page 5062) event handler.

Sender is the toolbar that contains the button.

Button is the button that needs to be painted. Use its properties to determine what to paint and where the button sits on the toolbar's canvas.

State provides information that you may want to visually represent when painting the button.

Stage indicates the current stage of the drawing process.

Flags is used only when Stage is cdPrePaint. It influences how the button is subsequently rendered by the default drawing process. Return the set of flags that indicate how you want the tool button to be drawn.

DefaultDraw is used only when Stage is cdPrePaint. Leave DefaultDraw as true if the button should be painted using the normal process after the event handler exists. Set DefaultDraw to false if the OnAdvancedCustomDrawButton event handler has completely painted the tool button.

Note: OnAdvancedCustomDrawButton occurs at several points during the painting of the toolbar's buttons. If you only need to respond immediately prior to the default painting of each button and are not using the Flags parameter, use OnCustomDrawButton (see page 5067) instead.

See Also

Canvas (see page 5029)

OnAdvancedCustomDraw (see page 5062)

OnCustomDrawButton (see page 5067)

27.1.51.6.3 TToolBar.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
```

```
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

27.1.51.6.4 TToolBar.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.51.6.5 TToolBar.OnCustomDraw Event

Occurs when the toolbar must be painted.

Pascal

```
property OnCustomDraw: TTBCustomDrawEvent;
```

C++

```
__property TTBCustomDrawEvent OnCustomDraw;
```

Description

Write an OnCustomDraw event handler to enhance or replace the automatic painting of the toolbar. Use the toolbar's Canvas (see page 5029) property to render the toolbar's image. OnCustomDraw is for painting the background of the toolbar. The painting of individual buttons on the toolbar should be handled in an OnCustomDrawButton (see page 5067) or OnAdvancedCustomDrawButton (see page 5063) event handler.

Sender is the toolbar that needs to be painted.

ARect indicates the dimensions of the toolbar on the canvas.

Leave DefaultDraw as true if the toolbar should continue with the normal painting process. Set DefaultDraw to false if the OnCustomDraw event handler has completely painted the toolbar.

Note: OnCustomDraw only occurs prior to painting the toolbar's background. To respond at other points in the paint process (when the background is erased, or after the default painting of the toolbar's background), use OnAdvancedCustomDraw (see page 5062) instead.

See Also

Canvas (see page 5029)

OnCustomDrawButton (see page 5067)

OnAdvancedCustomDraw (see page 5062)

OnAdvancedCustomDrawButton (see page 5063)

27.1.51.6.6 TToolBar.OnCustomDrawButton Event

Occurs when one of the buttons in the toolbar must be painted.

Pascal

```
property OnCustomDrawButton: TTBCustomDrawBtnEvent;
```

C++

```
__property TTBCustomDrawBtnEvent OnCustomDrawButton;
```

Description

Write an OnCustomDrawButton event handler to enhance or replace the automatic painting of the toolbar's buttons. Use the toolbar's Canvas (see page 5029) property to render the button's image. OnCustomDrawButton is for painting individual buttons on the toolbar. The painting of the toolbar itself should be handled in an OnCustomDraw (see page 5066) or OnAdvancedCustomDraw (see page 5062) event handler.

Sender is the toolbar that contains the button.

Button is the button that needs to be painted. Use its properties to determine what to paint and where the button sits on the toolbar's canvas.

State provides information that you may want to visually represent when painting the button.

Leave DefaultDraw as true if the button should be painted using the normal process after the event handler exists. Set DefaultDraw to false if the OnCustomDrawButton event handler has completely painted the tool button.

Note: OnCustomDrawButton only occurs prior to painting the button. To respond at other points in the process of painting a button (when the background is erased, or after the default painting of the button), use OnAdvancedCustomDrawButton (see page 5063) instead.

See Also

Canvas (see page 5029)

OnCustomDraw (see page 5066)

OnAdvancedCustomDrawButton (see page 5063)

27.1.51.6.7 TToolBar.OnCustomizeAdded Event

Occurs when the user adds a button to the toolbar.

Pascal

```
property OnCustomizeAdded: TTBButtonEvent;
```

C++

```
__property TTBButtonEvent OnCustomizeAdded;
```

Description

Write an OnCustomizeAdded handler to respond when the user adds a new button to the toolbar using the toolbar customize dialog.

This event occurs after the new button is already added to the toolbar. To respond before the button is added, possibly even preventing the user from adding the button, use the OnCustomizeCanInsert (see page 5069) event instead.

See Also

Customizable (see page 5035)

OnCustomizeCanDelete (see page 5068)

OnCustomizeCanInsert (see page 5069)

OnCustomized (see page 5069)

OnCustomizeDelete (see page 5070)

OnCustomizeNewButton (see page 5070)

OnCustomizeReset (see page 5071)

OnCustomizing (see page 5072)

27.1.51.6.8 TToolBar.OnCustomizeCanDelete Event

Occurs when user tries to delete one of the buttons on the toolbar.

Pascal

```
property OnCustomizeCanDelete: TTBCustomizeQueryEvent;
```

C++

```
__property TTBCustomizeQueryEvent OnCustomizeCanDelete;
```

Description

When the Customizable (see page 5035) property is true, the user can remove buttons from the toolbar by dragging them off the toolbar when the Shift key is pressed or using the toolbar customize dialog. OnCustomizeCanDelete occurs when the user tries to delete a button in one of these ways, and allows the application to reject the deletion. If the OnCustomizeCanDelete event handler sets its Allow parameter to false, the tool button returns to the position it occupied before the user tried to delete it.

See Also

Customizable (see page 5035)

OnCustomizeAdded (see page 5068)

OnCustomizeCanInsert (see page 5069)

OnCustomized (see page 5069)

OnCustomizeDelete (see page 5070)

OnCustomizeNewButton (see page 5070)

OnCustomizeReset (see page 5071)

OnCustomizing (see page 5072)

27.1.51.6.9 TToolBar.OnCustomizeCanInsert Event

Occurs when user tries to add a button to the toolbar.

Pascal

```
property OnCustomizeCanInsert: TTBCustomizeQueryEvent;
```

C++

```
__property TTBCustomizeQueryEvent OnCustomizeCanInsert;
```

Description

When the Customizable (see page 5035) property is true, the user can add buttons to the toolbar using the toolbar customize dialog. OnCustomizeCanInsert occurs when the user tries to add a button, and allows the application to reject the insertion. If the OnCustomizeCanInsert event handler sets its Allow parameter to false, the toolbar customize dialog rejects the insertion.

See Also

Customizable (see page 5035)

OnCustomizeAdded (see page 5068)

OnCustomizeCanDelete (see page 5068)

OnCustomized (see page 5069)

OnCustomizeDelete (see page 5070)

OnCustomizeNewButton (see page 5070)

OnCustomizeReset (see page 5071)

OnCustomizing (see page 5072)

27.1.51.6.10 TToolBar.OnCustomized Event

Occurs when the user finishes making changes to the toolbar.

Pascal

```
property OnCustomized: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCustomized;
```

Description

Write an OnCustomized event handler to take action after the user has finished making changes to the toolbar. This event occurs when the user finishes a drag operation (with the Shift key down) that moves or deletes a tool button, and when the user exits the toolbar customize dialog.

See Also

Customizable (see page 5035)

OnCustomizeAdded (see page 5068)

OnCustomizeCanDelete (see page 5068)

OnCustomizeCanInsert (see page 5069)

OnCustomizeDelete (see page 5070)

OnCustomizeNewButton (see page 5070)

OnCustomizeReset (see page 5071)

OnCustomizing (see page 5072)

27.1.51.6.11 TToolBar.OnCustomizeDelete Event

Occurs when the user deletes a button from the toolbar.

Pascal

```
property OnCustomizeDelete: TTBButtonEvent;
```

C++

```
__property TTBButtonEvent OnCustomizeDelete;
```

Description

Write an OnCustomizeDelete event handler to respond when the user deletes a tool button by dragging it off the toolbar when the Shift key is depressed or by using the toolbar customize dialog.

This event occurs after the button has been removed from the toolbar. To respond before the button is deleting, possibly preventing the user from deleting the button, use the OnCustomizeCanDelete (see page 5068) event instead.

See Also

Customizable (see page 5035)

OnCustomizeCanDelete (see page 5068)

OnCustomizeCanInsert (see page 5069)

OnCustomized (see page 5069)

OnCustomizeAdded (see page 5068)

OnCustomizeNewButton (see page 5070)

OnCustomizeReset (see page 5071)

OnCustomizing (see page 5072)

27.1.51.6.12 TToolBar.OnCustomizeNewButton Event

Occurs when the user tries to add a new button to the tool bar.

Pascal

```
property OnCustomizeNewButton: TTBNewButtonEvent;
```

C++

```
__property TTBNewButtonEvent OnCustomizeNewButton;
```

Description

Write an OnCustomizeNewButton event handler to supply your own TToolButton (see page 5107) objects to represent the tool buttons that a user adds in the toolbar customize dialog.

Sender is the toolbar to which the user is adding a button.

Index is the index in the Buttons (see page 5029) property where the new button will appear. This should be assigned as the Index property of the new button.

Button returns a TToolButton (see page 5107) object that the event handler creates and initializes. It is used to represent the new tool button in the Buttons (see page 5029) property. If the event handler does not return a TToolButton (see page 5107) object, the add operation fails.

See Also

Customizable (see page 5035)

OnCustomizeCanDelete (see page 5068)

OnCustomizeCanInsert (see page 5069)

OnCustomized (see page 5069)

OnCustomizeAdded (see page 5068)

OnCustomizeDelete (see page 5070)

OnCustomizeReset (see page 5071)

OnCustomizing (see page 5072)

Buttons (see page 5029)

Index (see page 5135)

27.1.51.6.13 TToolBar.OnCustomizeReset Event

Occurs when the user cancels a customization to the toolbar.

Pascal

```
property OnCustomizeReset: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCustomizeReset;
```

Description

Write an OnCustomizeReset event handler to respond when the user cancels changes made in the toolbar customize dialog.

See Also

Customizable (see page 5035)

OnCustomizeCanDelete (see page 5068)

OnCustomizeCanInsert (see page 5069)

OnCustomized (see page 5069)

OnCustomizeAdded (see page 5068)

OnCustomizeDelete (see page 5070)

OnCustomizeNewButton (see page 5070)

OnCustomizing (see page 5072)

CustomizeKeyName (see page 5036)

CustomizeValueName (see page 5036)

27.1.51.6.14 TToolBar.OnCustomizing Event

Occurs when the user displays the toolbar customize dialog in order to make changes to the toolbar.

Pascal

```
property OnCustomizing: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCustomizing;
```

Description

Write an OnCustomizing event handler to take action when the user first displays the toolbar customize dialog.

See Also

Customizable (see page 5035)

OnCustomizeCanDelete (see page 5068)

OnCustomizeCanInsert (see page 5069)

OnCustomized (see page 5069)

OnCustomizeAdded (see page 5068)

OnCustomizeDelete (see page 5070)

OnCustomizeNewButton (see page 5070)

OnCustomizeReset (see page 5071)

27.1.51.6.15 TToolBar.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

27.1.51.6.16 TToolBar.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop: TDockDropEvent;
```

C++

```
__property TDockDropEvent OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
```

```
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockOver (see page 6216)

27.1.51.6.17 TToolBar.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver: TDockOverEvent;
```

C++

```
__property TDockOverEvent OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
```

```
        TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
        bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnGetSiteInfo (see page 6223)

TDragDockObject (see page 6009)

27.1.51.6.18 TToolBar.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
```

```
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

27.1.51.6.19 TToolBar.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
```

```
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.51.6.20 TToolBar.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.51.6.21 TToolBar.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

27.1.51.6.22 TToolBar.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
```

```
    ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
```

```
    else if myActiveControl is TRichEdit then
      with myActiveControl as TRichEdit do
        SelAttributes.Assign(TFontDialog(Sender).Font)
    else
      Beep;
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

27.1.51.6.23 TToolBar.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
```

```
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

27.1.51.6.24 TToolBar.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo: TGetSiteInfoEvent;
```

C++

```
__property TGetSiteInfoEvent OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
```

```
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
```

```
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

TDragDockObject (see page 6009)

27.1.51.6.25 TToolBar.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.51.6.26 TToolBar.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}
```

```
// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
```

```
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.51.6.27 TToolBar.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

27.1.51.6.28 TToolBar.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

27.1.51.6.29 TToolBar.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.51.6.30 TToolBar.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
```

```
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
```

```
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

27.1.51.6.31 TToolBar.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

27.1.51.6.32 TToolBar.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

27.1.51.6.33 TToolBar.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it

contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

27.1.51.6.34 TToolBar.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock: TUnDockEvent;
```

C++

```
__property TUnDockEvent OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
```

```
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
```

```
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

27.1.52 TToolBarDockObject Class

Class Hierarchy

System::TObject → Controls.TDragObject → Controls.TBaseDragControlObject → Controls.TDragDockObject → ComCtrls.TToolBarDockObject

Pascal

```
TToolBarDockObject = class(TDragDockObject);
```

C++

```
class TToolBarDockObject : public TDragDockObject;
```

File

ComCtrls

Description

This is class ComCtrls.TToolBarDockObject.

27.1.52.1 TToolBarDockObject Members

The following tables list the members exposed by TToolBarDockObject.

Protected Methods

	Name	Description
	AdjustDockRect (see page 5104)	This is AdjustDockRect, a member of class TToolBarDockObject.
	DrawDragDockImage (see page 5104)	Called when the dock image for a control is drawn.
	EndDrag (see page 5723)	Generates an OnEndDrag event for the control that is being dragged.
	EndDrag (see page 6013)	Called when a control is released from dragging.
	EraseDragDockImage (see page 5104)	Called when the dock image for a control is erased.
	Finished (see page 5723)	Finished ends a drag and drop operation.
	Finished (see page 6035)	Responds when the mouse is released indicating the drag operation is complete.
	GetDragCursor (see page 5105)	Returns the cursor used during dragging.
	GetDragImages (see page 6035)	Returns an image list containing the image to be displayed while dragging.
	GetEraseWhenMoving (see page 6014)	This is GetEraseWhenMoving, a member of class TDragDockObject.
	GetFrameWidth (see page 6014)	Returns the value of the FrameWidth (see page 6018) property.
	MainWndProc (see page 6037)	This is MainWndProc, a member of class TDragObject.
	ToolDockImage (see page 5105)	This is ToolDockImage, a member of class TToolBarDockObject.
	WndProc (see page 6038)	This is WndProc, a member of class TDragObject.

Public Methods

	Name	Description
	AfterConstruction (see page 6033)	Responds after the last constructor has executed.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 5722)	Copies the properties of another drag control object.
	Assign (see page 6012)	Enables assignments of TDragDockObject (see page 6009) objects.
	Assign (see page 6034)	Copies the properties of another drag control object.
	BeforeDestruction (see page 6034)	Responds before the first destructor executes.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 5721)	Creates an instance of the TBaseDragControlObject (see page 5719) and stores a reference to the control that is being dragged.
V	Create (see page 6011)	Creates and initializes an instance of TDragDockObject (see page 6009).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 6012)	Destroys an instance of TDragDockObject (see page 6009).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetName (see page 6036)	Returns a string that identifies the drag operation.
V	HideDragImage (see page 6036)	Provides the interface for a method that hides the current drag image.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Instance (see page 6037)	Returns the instance (module) handle of the EXE or DLL that created the drag object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	ShowDragImage (see page 6037)	Provides the interface for a method that hides the current drag image.

Public Properties

	Name	Description
	AlwaysShowDragImages (see page 6038)	This is AlwaysShowDragImages, a member of class TDragObject.
	Brush (see page 6015)	Specifies the brush used to draw the dock image.
	Cancelling (see page 6038)	Specifies whether the drag operation is being canceled.
	Control (see page 5724)	Indicates the control that is being dragged.
	DockRect (see page 6015)	Specifies the rectangle on which the control is docked if released.
	DragHandle (see page 6039)	Specifies the window handle of the dragged object.
	DragPos (see page 6039)	Specifies the position of the dragged object.
	DragTarget (see page 6039)	Specifies the dragged object's target.
	DragTargetPos (see page 6040)	Specifies the position of the drag object relative to the drag target.
R	DropAlign (see page 6017)	Specifies how the control will be aligned when docked.
R	DropOnControl (see page 6017)	Specifies the control on which the dragged control will be docked if released.
R	Dropped (see page 6040)	Drag ended with drop.
	EraseDockRect (see page 6017)	This is EraseDockRect, a member of class TDragDockObject.
R	EraseWhenMoving (see page 6018)	This is EraseWhenMoving, a member of class TDragDockObject.
	Floating (see page 6018)	Specifies whether the control is floating.

R	FrameWidth (see page 6018)	Specifies the border width, in pixels, of the frame drawn around the drag image.
R	MouseDeltaX (see page 6040)	Specifies the x-coordinate of the mouse position relative to the position of the dragged object's target.
R	MouseDeltaY (see page 6041)	Specifies the y-coordinate of the mouse position relative to the position of the dragged object's target.
	RightClickCancels (see page 6041)	This is RightClickCancels, a member of class TDragObject.

27.1.52.2 TToolBarDockObject Methods

The methods of the TToolBarDockObject class are listed here.

Protected Methods

	Name	Description
V	AdjustDockRect (see page 5104)	This is AdjustDockRect, a member of class TToolBarDockObject.
V	DrawDragDockImage (see page 5104)	Called when the dock image for a control is drawn.
V	EraseDragDockImage (see page 5104)	Called when the dock image for a control is erased.
V	GetDragCursor (see page 5105)	Returns the cursor used during dragging.
V	ToolDockImage (see page 5105)	This is ToolDockImage, a member of class TToolBarDockObject.

27.1.52.2.1 TToolBarDockObject.AdjustDockRect Method

Pascal

```
procedure AdjustDockRect(ARect: TRect); override;
```

C++

```
virtual __fastcall AdjustDockRect(TRect ARect);
```

Description

This is AdjustDockRect, a member of class TToolBarDockObject.

27.1.52.2.2 TToolBarDockObject.DrawDragDockImage Method

Called when the dock image for a control is drawn.

Pascal

```
procedure DrawDragDockImage; override;
```

C++

```
virtual __fastcall DrawDragDockImage();
```

Description

Override DrawDragDockImage in a derived class to change the way the dock image is drawn.

27.1.52.2.3 TToolBarDockObject.EraseDragDockImage Method

Called when the dock image for a control is erased.

Pascal

```
procedure EraseDragDockImage; override;
```

C++

```
virtual __fastcall EraseDragDockImage();
```

Description

Override EraseDragDockImage in a derived class to perform actions when the dock image for a control is erased.

As implemented in TDragDockObject, EndDrag calls the dragged control's EraseDragDockImage method.

27.1.52.2.4 TToolBarDockObject.GetDragCursor Method

Returns the cursor used during dragging.

Pascal

```
function GetDragCursor(Accepted: Boolean; X: Integer; Y: Integer): TCursor; override;
```

C++

```
virtual __fastcall TCursor GetDragCursor(Boolean Accepted, int X, int Y);
```

Description

Override GetDragCursor to specify the cursor used during dragging.

By default, GetDragCursor returns crDefault.

See Also

Cursor (see page 5815)

27.1.52.2.5 TToolBarDockObject.ToolDockImage Method

Pascal

```
function ToolDockImage(Erase: Boolean): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ToolDockImage(Boolean Erase);
```

Description

This is ToolDockImage, a member of class TToolBarDockObject.

27.1.53 TToolBarEnumerator Class

Class Hierarchy

ComCtrls.TToolBarEnumerator

Pascal

```
TToolBarEnumerator = class;
```

C++

```
class TToolBarEnumerator;
```

File

ComCtrls

Description

This is class ComCtrls.TToolBarEnumerator.

27.1.53.1 TToolBarEnumerator Members

The following tables list the members exposed by TToolBarEnumerator.

Public Methods

	Name	Description
	Create (see page 5106)	This is Create, a member of class TToolBarEnumerator.
	GetCurrent (see page 5106)	This is GetCurrent, a member of class TToolBarEnumerator.
	MoveNext (see page 5106)	This is MoveNext, a member of class TToolBarEnumerator.

Public Properties

	Name	Description
R	Current (see page 5107)	This is Current, a member of class TToolBarEnumerator.

27.1.53.2 TToolBarEnumerator.Create Constructor

Pascal

```
constructor Create(AToolBar: TToolBar);
```

C++

```
__fastcall TToolBarEnumerator(TToolBar AToolBar);
```

Description

This is Create, a member of class TToolBarEnumerator.

27.1.53.3 TToolBarEnumerator Methods

The methods of the TToolBarEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 5106)	This is GetCurrent, a member of class TToolBarEnumerator.
	MoveNext (see page 5106)	This is MoveNext, a member of class TToolBarEnumerator.

27.1.53.3.1 TToolBarEnumerator.GetCurrent Method

Pascal

```
function GetCurrent: TToolButton;
```

C++

```
__fastcall TToolButton GetCurrent();
```

Description

This is GetCurrent, a member of class TToolBarEnumerator.

27.1.53.3.2 TToolBarEnumerator.MoveNext Method

Pascal

```
function MoveNext: Boolean;
```

C++

```
__fastcall Boolean MoveNext();
```

Description

This is MoveNext, a member of class TToolBarEnumerator.

27.1.53.4 TToolBarEnumerator Properties

The properties of the TToolBarEnumerator class are listed here.

Public Properties

	Name	Description
R	Current (see page 5107)	This is Current, a member of class TToolBarEnumerator.

27.1.53.4.1 TToolBarEnumerator.Current Property

Pascal

```
property Current: TToolButton;
```

C++

```
__property TToolButton Current;
```

Description

This is Current, a member of class TToolBarEnumerator.

27.1.54 TToolButton Class

TToolButton is a button control in a TToolBar (see page 4994) object.

Class Hierarchy

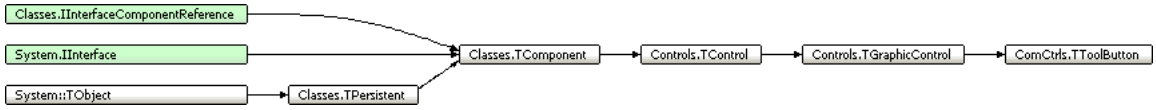

Pascal

```
TToolButton = class(TGraphicControl);
```

C++

```
class TToolButton : public TGraphicControl;
```

File

ComCtrls

Description

Use TToolButton to implement buttons on a toolbar. While other controls (including TButton and TSpeedButton) can be placed on toolbars, TToolButton utilizes special toolbar features to simplify the configuration of buttons and offers added display options such as pop-up borders and transparency.

To place tool buttons on a toolbar at design time, select the toolbar, right-click, and choose New Button.

See Also

TButton (see page 20909)

TSpeedButton (see page 2335)

TToolBar (see page 4994)

TToolBar (see page 4994)

Adding a Cool Bar Component

Adding a Speed Button to a Panel

Adding a Toolbar Using a Panel Component

Adding a Toolbar Using the Toolbar Component

Adding a Tool Button

Adding Hidden Toolbars

Allowing Toggle Buttons

Allowing Toggled Tool Buttons

Assigning a Menu to a Tool Button

Assigning a Speed Button's Glyph

Assigning Images to Tool Buttons

Creating a Group of Speed Buttons

Creating Groups of Tool Buttons

Designing Toolbars and Cool Bars

Hiding and Showing Toolbars

Responding to Clicks

Setting the Initial Condition of a Speed Button

Setting Tool Button Appearance and Initial Conditions

27.1.54.1 TToolButton Members

The following tables list the members exposed by TToolButton.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FToolBar (see page 5117)	This is FToolBar, a member of class TToolButton.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActionChange (see page 5118)	Called when the action associated with the button changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 5119)	Called when a button is used in an assignment.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
V	BeginUpdate (see page 5119)	Suspends screen repainting.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	EndUpdate (see page 5120)	Re-enables screen repainting.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.

	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5121)	Returns TToolButtonActionLink (see page 5166).
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5121)	Responds when the user clicks the button with the mouse.
	MouseMove (see page 5122)	Responds when the user moves the mouse over the tool button.
	MouseUp (see page 5122)	Responds when the user releases the mouse.
	Notification (see page 5123)	Responds to notifications that components are being created or destroyed.
	Paint (see page 5123)	Renders the image of the tool button.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RefreshControl (see page 5124)	Repaints the button if necessary.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5124)	Sets the value of the AutoSize (see page 5128) property.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetToolBar (see page 5125)	Associates the tool button with a tool bar.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControl (see page 5125)	Ensures the associated tool bar responds to changes in the tool button.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
V	ValidateContainer (see page 5126)	Updates the tool button's width and height when it is inserted into a tool bar.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	CheckMenuDropdown (see page 5120)	Displays a button's drop-down menu.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Click (see page 5120)	Generates an OnClick (see page 5148) event.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Create (see page 5116)	Creates and initializes a TToolButton (see page 5107) instance.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.

	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 5124)	Changes the position and size of the button in a single operation.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.

	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
R	Index (see page 5135)	Specifies the index of the tool button.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnClick (see page 5148)	Occurs when the user clicks the control.
	OnContextPopup (see page 5150)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 5151)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5153)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5155)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5156)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5157)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5158)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5160)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5160)	Occurs when the user moves the mouse outside of a control.

	OnMouseMove (see page 5161)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5163)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 5165)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 5166)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Action (see page 5127)	Designates the action associated with the control.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AllowAllUp (see page 5127)	Indicates whether all the tool buttons in a group can be unselected at the same time.
	AutoSize (see page 5128)	Specifies whether the tool button automatically sizes itself to accommodate its text and image.
	Caption (see page 5128)	Specifies button caption.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Down (see page 5128)	Specifies whether the button is selected (down) or unselected (up).
	DragCursor (see page 5129)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5129)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5130)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropdownMenu (see page 5131)	Identifies a pop-up menu associated with the tool button.
	Enabled (see page 5132)	Controls whether the control responds to mouse, keyboard, and timer events.
	EnableDropdown (see page 5132)	This is EnableDropdown, a member of class TToolButton.
	Grouped (see page 5132)	Aggregates a sequence of adjacent tbsCheck buttons.
	Height (see page 5133)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImageIndex (see page 5135)	Determines which image appears on the tool button.
	Indeterminate (see page 5135)	Indicates that the tool button is neither selected nor unselected.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Marked (see page 5136)	Specifies whether the button is marked.
	MenuItem (see page 5136)	Specifies the menu item corresponding to the button.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentShowHint (see page 5137)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 5137)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 5142)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 5144)	Determines the style of the tool button.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 5145)	Determines whether the component appears onscreen.
	Width (see page 5147)	Specifies the horizontal size of the control or form in pixels.
	Wrap (see page 5147)	Forces a new row after the tool button.

27.1.54.2 TToolButton.Create Constructor

Creates and initializes a TToolButton (see page 5107) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TToolButton(TComponent * AOwner);
```

Description

Call Create to create a TToolButton (see page 5107) object at runtime. Tool buttons added at design time are created automatically.

AOwner is the component, typically the form, that is responsible for freeing the button. It becomes the value of the Owner property.

C++ Examples:

```
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
```

```
  delete myToolBar;
}
```

Delphi Examples:

```
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

See Also

Owner (see page 2807)

27.1.54.3 TToolButton Fields

The fields of the TToolButton class are listed here.

Protected Fields

	Name	Description
	FToolBar (see page 5117)	This is FToolBar, a member of class TToolButton.

27.1.54.3.1 TToolButton.FToolBar Field

Pascal

```
FToolBar: TToolBar;
```

C++

```
TToolBar FToolBar;
```

Description

This is FToolBar, a member of class TToolButton.

27.1.54.4 TToolButton Methods

The methods of the TToolButton class are listed here.

Protected Methods

	Name	Description
	ActionChange (see page 5118)	Called when the action associated with the button changes.
	AssignTo (see page 5119)	Called when a button is used in an assignment.
	BeginUpdate (see page 5119)	Suspends screen repainting.
	EndUpdate (see page 5120)	Re-enables screen repainting.
	GetActionLinkClass (see page 5121)	Returns TToolButtonActionLink (see page 5166).
	MouseDown (see page 5121)	Responds when the user clicks the button with the mouse.
	MouseMove (see page 5122)	Responds when the user moves the mouse over the tool button.
	MouseUp (see page 5122)	Responds when the user releases the mouse.
	Notification (see page 5123)	Responds to notifications that components are being created or destroyed.
	Paint (see page 5123)	Renders the image of the tool button.
	RefreshControl (see page 5124)	Repaints the button if necessary.
	SetAutoSize (see page 5124)	Sets the value of the AutoSize (see page 5128) property.
	SetToolBar (see page 5125)	Associates the tool button with a tool bar.
	UpdateControl (see page 5125)	Ensures the associated tool bar responds to changes in the tool button.
	ValidateContainer (see page 5126)	Updates the tool button's width and height when it is inserted into a tool bar.

Public Methods

	Name	Description
	CheckMenuDropdown (see page 5120)	Displays a button's drop-down menu.
	Click (see page 5120)	Generates an OnClick (see page 5148) event.
	SetBounds (see page 5124)	Changes the position and size of the button in a single operation.

27.1.54.4.1 TToolButton.ActionChange Method

Called when the action associated with the button changes.

Pascal

```
procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); override;
```

C++

```
virtual __fastcall ActionChange(TObject * Sender, Boolean CheckDefaults);
```

Description

ActionChange is called automatically when the action object specified by the Action (see page 5127) property, or any of its properties, changes. ActionChange updates the control properties that correspond to properties of the associated action. These properties include the Down (see page 5128), ImageIndex (see page 5135), Caption (see page 5128), Enabled (see page 5132), Hint, and Visible (see page 5145) properties and the OnClick (see page 5148) event handler.

The Sender parameter is the action object that changed. The CheckDefaults parameter indicates whether the control's properties should all be updated or only those that have not been explicitly changed from their default values. When CheckDefaults is true, only those properties and event handlers that have not been changed from the default values are updated.

See Also

Action (see page 5797)

Down (see page 5128)

ImageIndex (see page 5135)

Caption (see page 5806)

Enabled (see page 5821)

Hint (see page 5827)

Visible (see page 5847)

OnClick (see page 5851)

27.1.54.4.2 TToolButton.AssignTo Method

Called when a button is used in an assignment.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

The Assign method of the destination object specified by Dest calls AssignTo if it doesn't know how to copy the methods of a tool button. AssignTo copies properties if Dest is a TCustomAction descendant.

See Also

ActionChange (see page 5118)

TCustomAction (see page 297)

Assign (see page 2930)

27.1.54.4.3 TToolButton.BeginUpdate Method

Suspends screen repainting.

Pascal

```
procedure BeginUpdate; virtual;
```

C++

```
virtual __fastcall BeginUpdate();
```

Description

The BeginUpdate method prevents the tool bar that contains the button from adjusting to changes in the buttons size and position until the EndUpdate (see page 5120) method is called. Use BeginUpdate to speed processing and avoid flicker while making changes to the tool button's properties.

See Also

EndUpdate (see page 5120)

27.1.54.4.4 TToolButton.CheckMenuDropdown Method

Displays a button's drop-down menu.

Pascal

```
function CheckMenuDropdown: Boolean; dynamic;
```

C++

```
__fastcall Boolean CheckMenuDropdown();
```

Description

Call CheckMenuDropdown to manually display a button's pop-up menu. It returns true and displays the menu if the tool button has an associated menu. It returns false if there is no menu to display.

See Also

Style (see page 5144)

27.1.54.4.5 TToolButton.Click Method

Generates an OnClick (see page 5148) event.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Click calls any event handler attached to the OnClick (see page 5148) event. Override Click to provide other responses in addition to the inherited event-handler call when the user clicks the control or the application simulates a user click.

A control calls Click whenever it receives a left mouse-button mouse-up message (WM_LBUTTONUP) if that message indicates that the mouse up occurred near the corresponding mouse-down. Many controls call Click in other circumstances, such as when a user types the shortcut key for a button or menu item.

See Also

DblClick (see page 5747)

OnClick (see page 5851)

27.1.54.4.6 TToolButton.EndUpdate Method

Re-enables screen repainting.

Pascal

```
procedure EndUpdate; virtual;
```

C++

```
virtual __fastcall EndUpdate();
```

Description

Use EndUpdate to enable the associated tool bar to adjust to changes in the button's size and position, after this was turned off by the BeginUpdate (see page 5119) method. Calls to BeginUpdate (see page 5119) and EndUpdate can be nested. The tool bar will not adjust to changes in the tool button until every call to BeginUpdate (see page 5119) has been matched to a

corresponding call to EndUpdate.

See Also

BeginUpdate (see page 5119)

27.1.54.4.7 TToolButton.GetActionLinkClass Method

Returns TToolButtonActionLink (see page 5166).

Pascal

```
function GetActionLinkClass: TControlActionLinkClass; override;
```

C++

```
virtual __fastcall TControlActionLinkClass GetActionLinkClass();
```

Description

GetActionLinkClass returns the appropriate class type for the action link used with a tool button. This class is used internally to create an action link object for the tool button when it has an associated action. Each Action (see page 5127) link class is designed to link specific properties and event handlers of the action to its client, based on the types of properties the client supports.

As implemented in TToolButton (see page 5107), GetActionLinkClass returns the TToolButtonActionLink (see page 5166) class type, which associates the Down (see page 5128), ImageIndex (see page 5135), ShowHint (see page 5142), Caption (see page 5128), Enabled (see page 5132) and Visible (see page 5145) properties and the OnClick (see page 5148) event handler.

See Also

Action (see page 5797)

ActionLink (see page 5798)

TToolButtonActionLink (see page 5166)

27.1.54.4.8 TToolButton.MouseDown Method

Responds when the user clicks the button with the mouse.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Do not call MouseDown. It is called automatically when the user clicks the tool button with the mouse. As implemented in TToolButton (see page 5107), MouseDown first toggles the Down (see page 5128) property if the button's style is tbsDropDown and the user clicked the left mouse button. Then it generates an OnMouseDown (see page 5158) event.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

MouseDown helps ensure that the tool button's behavior remains unchanged, regardless of the version of ComCtl32.dll that is installed on the users system.

See Also

MouseMove (see page 5122)

MouseUp (see page 5122)

OnMouseDown (see page 5862)

Down (see page 5128)

Style (see page 5144)

27.1.54.4.9 TToolButton.MouseMove Method

Responds when the user moves the mouse over the tool button.

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Do not call MouseMove. It is called automatically when the user moves the mouse over the tool button or when the tool button has captured the mouse. As implemented in TToolButton (see page 5107), MouseMove ensures that the Down (see page 5128) property is false if the button's style is tbsDropDown and the user moved the mouse off the tool button. Then it generates an OnMouseMove (see page 5161) event.

The Shift parameter indicates which keys (Shift, Ctrl, and Alt) were down when the mouse moved. X and Y indicate the new position of the mouse pointer.

MouseDown (see page 5121) helps ensure that the tool button's behavior remains unchanged, regardless of the version of ComCtl32.dll that is installed on the users system.

See Also

MouseDown (see page 5121)

MouseUp (see page 5122)

OnMouseMove (see page 5865)

Down (see page 5128)

Style (see page 5144)

27.1.54.4.10 TToolButton.MouseUp Method

Responds when the user releases the mouse.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Do not call MouseUp. It is called automatically when the user releases the mouse. As implemented in TToolButton (see page 5107), MouseUp first generates an OnMouseUp (see page 5163) event. Then, if the left mouse button while over the tool

button and the button's style is tbsDropDown, MouseUp sets the Down (see page 5128) property to false. Finally, MouseUp generates an OnClick (see page 5148) event.

The Button parameter determines which mouse button the user released. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user released the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

MouseUp helps ensure that the tool button's behavior remains unchanged, regardless of the version of ComCtl32.dll that is installed on the users system.

See Also

MouseDown (see page 5121)

MouseMove (see page 5122)

OnMouseUp (see page 5867)

Down (see page 5128)

Style (see page 5144)

OnClick (see page 5851)

27.1.54.4.11 TToolButton.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TToolButton (see page 5107) to update its DropDownMenu or MenuItem (see page 5136) property when the menu or menu item to which it refers is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created.

See Also

DropdownMenu (see page 5131)

MenuItem (see page 5136)

Notification (see page 2783)

27.1.54.4.12 TToolButton.Paint Method

Renders the image of the tool button.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Paint draws the edges of the tool button to reflect the value of the Style (see page 5144) property. If the tool button is not contained by a tool bar, Paint does nothing.

See Also

Style (see page 5144)

TToolBar (see page 4994)

SetToolBar (see page 5125)

27.1.54.4.13 TToolButton.RefreshControl Method

Repaints the button if necessary.

Pascal

```
procedure RefreshControl; virtual;
```

C++

```
virtual __fastcall RefreshControl();
```

Description

Use RefreshControl to repaint the button if necessary. Use RefreshControl rather than the Refresh method to ensure that the tool button honors the BeginUpdate (see page 5119) method.

RefreshControl is called when the button is aligned, wrapped, or changed in style.

See Also

BeginUpdate (see page 5119)

Refresh (see page 5779)

27.1.54.4.14 TToolButton.SetAutoSize Method

Sets the value of the AutoSize (see page 5128) property.

Pascal

```
procedure SetAutoSize(Value: Boolean); override;
```

C++

```
virtual __fastcall SetAutoSize(Boolean Value);
```

Description

SetAutoSize is the protected write implementation of the AutoSize (see page 5128) property. TToolButton (see page 5107) overrides the inherited method so that it can force the parent toolbar to update its layout if the tool button resizes in response to changing AutoSize (see page 5128) to true.

See Also

AutoSize (see page 5128)

27.1.54.4.15 TToolButton.SetBounds Method

Changes the position and size of the button in a single operation.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Call SetBounds to set the Left, Top, Width (see page 5147), and Height (see page 5133) properties at once. By calling SetBounds, you guard against a paint operation occurring after one property is set and before the next, which can cause screen flicker.

SetBounds also provides the underlying implementation for setting these four properties.

As implemented in TToolButton (see page 5107), SetBounds ensures that the new position and dimensions of the tool button fit within the constraints imposed by its parent toolbar, and cause the parent toolbar to resize, if necessary, to accommodate the changes in the tool button.

See Also

Left (see page 5830)

Top (see page 5844)

Width (see page 5849)

Height (see page 5825)

27.1.54.4.16 TToolButton.SetToolBar Method

Associates the tool button with a tool bar.

Pascal

```
procedure SetToolBar(AToolBar: TToolBar);
```

C++

```
__fastcall SetToolBar(TToolBar AToolBar);
```

Description

Call SetToolBar to add the tool button to a tool bar. Specify the tool bar as the AToolBar parameter.

If the tool button is already contained by a tool bar, SetToolBar removes the button from its current tool bar before moving it to the new tool bar.

See Also

Buttons (see page 5029)

27.1.54.4.17 TToolButton.UpdateControl Method

Ensures the associated tool bar responds to changes in the tool button.

Pascal

```
procedure UpdateControl; virtual;
```

C++

```
virtual __fastcall UpdateControl();
```

Description

Tool buttons call UpdateControl after any property changes that could affect the layout of the tool bar that contains the button. UpdateControl informs the tool bar that the button has changed so that it can make any necessary adjustments.

See Also

SetToolBar (see page 5125)

27.1.54.4.18 TToolButton.ValidateContainer Method

Updates the tool button's width and height when it is inserted into a tool bar.

Pascal

```
procedure ValidateContainer(AComponent: TComponent); override;
```

C++

```
virtual __fastcall ValidateContainer(TComponent * AComponent);
```

Description

ValidateContainer is called when the tool button is about to be inserted into the container specified by the AComponent parameter. After calling the ValidateContainer method of TComponent to ensure that the insertion is legal, ValidateContainer checks whether AComponent is a tool bar. If so, it updates the Height (see page 5133) and Width (see page 5147) properties to reflect the tool bar's ButtonHeight and ButtonWidth properties.

See Also

ValidateContainer (see page 2792)

SetToolBar (see page 5125)

ButtonWidth (see page 5029)

ButtonHeight (see page 5028)

Height (see page 5825)

Width (see page 5849)

27.1.54.5 TToolButton Properties

The properties of the TToolButton class are listed here.

Public Properties

	Name	Description
R	Index (see page 5135)	Specifies the index of the tool button.

Published Properties

	Name	Description
	Action (see page 5127)	Designates the action associated with the control.
	AllowAllUp (see page 5127)	Indicates whether all the tool buttons in a group can be unselected at the same time.
	AutoSize (see page 5128)	Specifies whether the tool button automatically sizes itself to accommodate its text and image.
	Caption (see page 5128)	Specifies button caption.
	Down (see page 5128)	Specifies whether the button is selected (down) or unselected (up).
	DragCursor (see page 5129)	Indicates the image used to represent the mouse pointer when the control is being dragged.

	DragKind (see page 5129)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5130)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropdownMenu (see page 5131)	Identifies a pop-up menu associated with the tool button.
	Enabled (see page 5132)	Controls whether the control responds to mouse, keyboard, and timer events.
	EnableDropdown (see page 5132)	This is EnableDropdown, a member of class TToolButton.
	Grouped (see page 5132)	Aggregates a sequence of adjacent tbsCheck buttons.
	Height (see page 5133)	Specifies the vertical size of the control in pixels.
	ImageIndex (see page 5135)	Determines which image appears on the tool button.
	Indeterminate (see page 5135)	Indicates that the tool button is neither selected nor unselected.
	Marked (see page 5136)	Specifies whether the button is marked.
	MenuItem (see page 5136)	Specifies the menu item corresponding to the button.
	ParentShowHint (see page 5137)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 5137)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 5142)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 5144)	Determines the style of the tool button.
	Visible (see page 5145)	Determines whether the component appears onscreen.
	Width (see page 5147)	Specifies the horizontal size of the control or form in pixels.
	Wrap (see page 5147)	Forces a new row after the tool button.

27.1.54.5.1 TToolButton.Action Property

Designates the action associated with the control.

Pascal

```
property Action: TBasicAction;
```

C++

```
__property TBasicAction Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

27.1.54.5.2 TToolButton.AllowAllUp Property

Indicates whether all the tool buttons in a group can be unselected at the same time.

Pascal

```
property AllowAllUp: Boolean;
```

C++

```
__property Boolean AllowAllUp;
```

Description

If AllowAllUp is set to true, all of the tool buttons in the group can be unselected. If AllowAllUp is set to false, at least one tool button in the group must be selected at any time. (Determine which tool button is initially selected by setting its Down (see page 5128) property to true.)

AllowAllUp is effective only when Grouped (see page 5132) is true and Style (see page 5144) is set to tbsCheck. Changing

the value of AllowAllUp for one button in a group automatically changes the value for all buttons in the group.

See Also

Down (see page 5128)

Grouped (see page 5132)

Style (see page 5144)

AllowAllUp (see page 2354)

27.1.54.5.3 TToolButton.AutoSize Property

Specifies whether the tool button automatically sizes itself to accommodate its text and image.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

If AutoSize is true, the tool button sizes itself automatically; otherwise, the tool button keeps a fixed size.

27.1.54.5.4 TToolButton.Caption Property

Specifies button caption.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the caption displayed on the button.

To underline a character in a Caption that labels a tool button, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the tool button by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

See Also

ImageIndex (see page 5135)

Name

27.1.54.5.5 TToolButton.Down Property

Specifies whether the button is selected (down) or unselected (up).

Pascal

```
property Down: Boolean;
```

C++

```
__property Boolean Down;
```

Description

Read Down to determine whether a tool button is selected. Setting Down to true at design time causes the button to be initially selected.

Tool buttons can remain Down only if Style (see page 5144) is tbsCheck.

See Also

AllowAllUp (see page 5127)

Grouped (see page 5132)

Indeterminate (see page 5135)

Style (see page 5144)

27.1.54.5.6 TToolButton.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

27.1.54.5.7 TToolButton.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

27.1.54.5.8 TToolButton.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
```

```
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

27.1.54.5.9 TToolButton.DropdownMenu Property

Identifies a pop-up menu associated with the tool button.

Pascal

```
property DropdownMenu: TPopupMenu;
```

C++

```
__property TPopupMenu DropdownMenu;
```

Description

Assign a value to DropdownMenu to make a menu appear when the button is pressed. (To display the menu programmatically, call the tool button's Click (see page 5120) method.) If the menu's AutoPopup property is set to true, the menu appears automatically. If AutoPopup is false, display the menu by calling the menu's Popup method in the tool button's OnClick (see page 5148) event handler.

When associating a drop-down menu with the tool button, set the Style (see page 5144) to tbsDropDown.

See Also

Click (see page 5743)

Down (see page 5128)

OnClick (see page 5851)

AutoPopup (see page 16178)

Popup (see page 16174)

CheckToolMenuDropDown

Style (see page 5144)

27.1.54.5.10 TToolButton.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.54.5.11 TToolButton.EnableDropdown Property

Pascal

```
property EnableDropdown: Boolean;
```

C++

```
__property Boolean EnableDropdown;
```

Description

This is EnableDropdown, a member of class TToolButton.

27.1.54.5.12 TToolButton.Grouped Property

Aggregates a sequence of adjacent tbsCheck buttons.

Pascal

```
property Grouped: Boolean;
```

C++

```
__property Boolean Grouped;
```

Description

When an unbroken sequence of adjacent tool buttons each has its Grouped property set to true and Style (see page 5144) set to tbsCheck, then no more than one of the buttons can be selected at the same time. Selecting a grouped button at runtime unselects the other buttons in its group.

To create distinct groups of buttons within a toolbar, separate the groups with any of the following: tool buttons whose Grouped property is false, tool buttons whose Style (see page 5144) property is not tbsCheck (tbsSeparator and tbsDivider buttons are useful for this), or controls other than tool buttons.

C++ Examples:

```
/*
To create a toolbar that corresponds to an existing menu:

1   Drop a ToolBar on the form and add a ToolButton for each
  top-level menu item you wish to create.
2   Set the MenuItem property of each ToolButton to correspond
  to the top level menu items.
3   Set the Grouped property of each ToolButton to true.
4   Clear the Menu property of the Form (if it is assigned)
5   Set the Visible property on the ToolBar and ToolButtons to
  true.
6   Set the ShowCaptions property on the ToolBar to true.
7   Set the MenuItem | Visible property of each ToolButton to
  true.
*/
```

Delphi Examples:

```
{
To create a toolbar that corresponds to an existing menu:

1   Drop a ToolBar on the form and add a ToolButton for each top-level menu item you wish to
create.
2   Set the MenuItem property of each ToolButton to correspond to the top level menu items.
3   Set the Grouped property of each ToolButton to true.
4   Clear the Menu property of the Form (if it is assigned)
5   Set the Visible property on the ToolBar and ToolButtons to true.
6   Set the ShowCaptions property on the ToolBar to true.
7   Set the MenuItem | Visible property of each ToolButton to true.
}
```

See Also

AllowAllUp (see page 5127)

Down (see page 5128)

Indeterminate (see page 5135)

Style (see page 5144)

27.1.54.5.13 TToolButton.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
HideSelection, Height example
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
        HideSelection := False;
    end;
end;
```

See Also

ClientHeight (see page 5808)

Width (see page 5849)

Using TDBListBox and TDBComboBox

27.1.54.5.14 TToolButton.ImageIndex Property

Determines which image appears on the tool button.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

The ImageIndex property determines which image—from TToolBar (see page 4994)'s Images list—appears on the tool button. If the toolbar's DisabledImages and HotImages properties are set, different images appear when the button is disabled and when the mouse pointer passes over the button.

These lists are zero-based; the first image is indexed by 0, the second by 1, and so forth. Since ImageIndex holds only a single integer value, the positions of the selected images must coincide on the different lists. For example, if ImageIndex = 2, the button displays the third image in each list.

See Also

DisabledImages (see page 5037)

HotImages (see page 5046)

Images (see page 5047)

27.1.54.5.15 TToolButton.Indeterminate Property

Indicates that the tool button is neither selected nor unselected.

Pascal

```
property Indeterminate: Boolean;
```

C++

```
__property Boolean Indeterminate;
```

Description

Indeterminate provides a third state for tool buttons that are neither selected nor unselected. Set this property in program code to indicate that a button is inapplicable in a particular context. Buttons (see page 2289) with the Indeterminate property set to true appear grayed.

When Indeterminate is set to true, Down (see page 5128) is automatically set to false.

See Also

Down (see page 5128)

27.1.54.5.16 TToolButton.Index Property

Specifies the index of the tool button.

Pascal

```
property Index: Integer;
```

C++

```
__property int Index;
```

Description

Use Index to get the index of the tool button. This index identifies the position of the tool button in its parent toolbar's Buttons property array.

See Also

Buttons (see page 5029)

27.1.54.5.17 TToolButton.Marked Property

Specifies whether the button is marked.

Pascal

```
property Marked: Boolean;
```

C++

```
__property Boolean Marked;
```

Description

Use Marked to get or set whether the button is marked. A marked button has a shaded surface.

See Also

Down (see page 5128)

Indeterminate (see page 5135)

27.1.54.5.18 TToolButton.MenuItem Property

Specifies the menu item corresponding to the button.

Pascal

```
property MenuItem: TMenuItem;
```

C++

```
__property TMenuItem MenuItem;
```

Description

Use MenuItem to get or set the menu item corresponding to the button.

Note: To use the tool button as a menu item, the Style (see page 5144) property must be tbsDropDown. If you are using multiple drop-down tool buttons, set the Grouped (see page 5132) property to true.

C++ Examples:

```
/*
To create a toolbar that corresponds to an existing menu:

1   Drop a ToolBar on the form and add a ToolButton for each
  top-level menu item you wish to create.
2   Set the MenuItem property of each ToolButton to correspond
  to the top level menu items.
3   Set the Grouped property of each ToolButton to true.
4   Clear the Menu property of the Form (if it is assigned)
5   Set the Visible property on the ToolBar and ToolButtons to
  true.
6   Set the ShowCaptions property on the ToolBar to true.
```

```
7   Set the MenuItem | Visible property of each ToolButton to
  true.
*/
```

Delphi Examples:

```
{
To create a toolbar that corresponds to an existing menu:

1   Drop a ToolBar on the form and add a ToolButton for each top-level menu item you wish to
create.
2   Set the MenuItem property of each ToolButton to correspond to the top level menu items.
3   Set the Grouped property of each ToolButton to true.
4   Clear the Menu property of the Form (if it is assigned)
5   Set the Visible property on the ToolBar and ToolButtons to true.
6   Set the ShowCaptions property on the ToolBar to true.
7   Set the MenuItem | Visible property of each ToolButton to true.
}
```

See Also

Grouped (see page 5132)

Style (see page 5144)

27.1.54.5.19 TToolButton.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.54.5.20 TToolButton.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
```

```
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
```

```
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
```

```
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
```

```
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.54.5.21 TToolButton.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
```

```
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.54.5.22 TToolButton.Style Property

Determines the style of the tool button.

Pascal

```
property Style: TToolButtonStyle;
```

C++

```
__property TToolButtonStyle Style;
```

Description

These are the possible values of Style:

Value	Meaning
tbsButton	The button appears and functions like a normal speed button.
tbsCheck	Clicking the button toggles the Down (see page 5128) property. Once selected, the button remains selected until clicked again.
tbsDropDown	The button displays a downward-pointing arrow (suitable for accessing a drop-down menu).
tbsSeparator	The button appears as an empty space on the toolbar (used to separate other controls).
tbsDivider	The button appears as a vertical line on the toolbar (used to separate other controls).
tbsTextButton	The button appears as text only.

C++ Examples:

```
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
```

```
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

See Also

AllowAllUp (see page 5127)

DropdownMenu (see page 5131)

TSpeedButton (see page 2335)

27.1.54.5.23 TToolButton.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.54.5.24 TToolButton.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

See Also

ClientWidth (see page 5811)

Height (see page 5825)

Left (see page 5830)

27.1.54.5.25 TToolButton.Wrap Property

Forces a new row after the tool button.

Pascal

```
property Wrap: Boolean;
```

C++

```
__property Boolean Wrap;
```

Description

Set Wrap to true to make the tool button the last in its row. Wrap is disabled when the toolbar's Wrapable property is true.

See Also

Wrapable (see page 5061)

27.1.54.6 TToolButton Events

The events of the TToolButton class are listed here.

Published Events

	Name	Description
	OnClick (see page 5148)	Occurs when the user clicks the control.
	OnContextPopup (see page 5150)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 5151)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5153)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5155)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5156)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5157)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5158)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5160)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5160)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5161)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5163)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 5165)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 5166)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

27.1.54.6.1 TToolButton.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.

- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

27.1.54.6.2 TToolButton.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple

times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.54.6.3 TToolButton.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
```

```
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
```

```
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

27.1.54.6.4 TToolButton.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/
```

```
// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
```

```
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.54.6.5 TToolButton.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.54.6.6 TToolButton.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
```

```
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

27.1.54.6.7 TToolButton.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.54.6.8 TToolButton.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}
```

```
// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
```

```
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.54.6.9 TToolButton.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

27.1.54.6.10 TToolButton.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

27.1.54.6.11 TToolButton.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.54.6.12 TToolButton.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
```

```
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
```

```
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

27.1.54.6.13 TToolButton.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

27.1.54.6.14 TToolButton.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

27.1.55 TToolButtonActionLink Class

TToolButtonActionLink is an action link designed for tool button control clients.

Class Hierarchy

Pascal

```
TToolButtonActionLink = class(TControlActionLink);
```

C++

```
class TToolButtonActionLink : public TControlActionLink;
```

File

ComCtrls

Description

TToolButtonActionLink is designed to link actions to tool button control clients. It overrides methods to determine whether the action and client are linked, and to ensure that action events are appropriate for properties and events of tool button control clients.

TToolButtonActionLink inherits, from its ancestors, support for linking actions with the properties of components, generic controls, and windowed controls. TToolButtonActionLink adds support for linking the following properties to the appropriate behavior for a tool button control:

Checked

ImageIndex

Not all actions correspond to properties and events of the same name. For example, the tool button 'down' property is linked to the 'checked' action.

27.1.55.1 TToolButtonActionLink Members

The following tables list the members exposed by TToolButtonActionLink.

Protected Fields

	Name	Description
	FAction (see page 2722)	This is FAction, a member of class TBasicActionLink.
	FClient (see page 5169)	This is FClient, a member of class TToolButtonActionLink.

Protected Methods

	Name	Description
	AssignClient (see page 5170)	Assigns a client tool button that is linked to the associated action.
	Change (see page 2723)	This is Change, a member of class TBasicActionLink.
	DoShowHint (see page 5896)	Indicates whether the hint is shown, and modified.
	IsCaptionLinked (see page 268)	Indicates whether the Caption properties for the action and client are linked.
	IsCaptionLinked (see page 5896)	Indicates whether the caption for the action and the client are linked.
	IsCheckedLinked (see page 5170)	Indicates whether the checked state for the action and the client are linked.
	IsDropdownMenuLinked (see page 5170)	This is IsDropdownMenuLinked, a member of class TToolButtonActionLink.
	IsEnabledLinked (see page 269)	Indicates whether the Enabled properties for the action and client are linked.
	IsEnabledLinked (see page 5897)	Indicates whether the enabled state for the action and the client are linked.
	IsEnableDropdownLinked (see page 5171)	This is IsEnableDropdownLinked, a member of class TToolButtonActionLink.
	IsGroupIndexLinked (see page 270)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsHelpContextLinked (see page 270)	Indicates whether the HelpContext properties for the action and client controls are linked.
	IsHelpLinked (see page 271)	Indicates whether the help properties for the action and client are linked.
	IsHelpLinked (see page 5897)	Indicates whether the help properties for the action and the client are linked.
	IsHintLinked (see page 271)	Indicates whether the Hint properties for the action and client are linked.
	IsHintLinked (see page 5898)	Indicates whether the Hint property for the action and the client are linked.
	IsImageIndexLinked (see page 5171)	Indicates whether the image index for the action and the client are linked.
	IsOnExecuteLinked (see page 2723)	This is IsOnExecuteLinked, a member of class TBasicActionLink.
	IsOnExecuteLinked (see page 5898)	Indicates whether the OnExecute event handler for the action and the client are linked.

	IsPopupMenuLinked (see page 5898)	This is IsPopupMenuLinked, a member of class TControlActionLink.
	IsShortCutLinked (see page 272)	Indicates whether the ShortCut properties for the action and client are linked.
	IsVisibleLinked (see page 273)	Indicates whether the Visible properties for the action and client are linked.
	IsVisibleLinked (see page 5899)	Indicates whether the visible state for the action and the client are linked.
	SetAction (see page 2724)	This is SetAction, a member of class TBasicActionLink.
	SetAutoCheck (see page 274)	Introduces an interface for setting the AutoCheck property of a client.
	SetCaption (see page 274)	Introduces an interface for setting the caption on a client.
	SetCaption (see page 5899)	Sets the caption of a control if the client is linked to the action for this event.
	SetChecked (see page 5171)	Sets the checked state of a tool button control if the client is linked to the action for this event.
	SetDropdownMenu (see page 5172)	This is SetDropdownMenu, a member of class TToolButtonActionLink.
	SetEnabled (see page 275)	Introduces an interface for setting the enabled state of a client.
	SetEnabled (see page 5900)	Sets the enabled state of a control if the client is linked to the action for this event.
	SetEnableDropdown (see page 5172)	This is SetEnableDropdown, a member of class TToolButtonActionLink.
	SetGroupIndex (see page 275)	Introduces an interface for setting the GroupIndex property of a client.
	SetHelpContext (see page 276)	Introduces an interface for setting the help context of a client.
	SetHelpContext (see page 5900)	Sets the HelpContext of a control if the client is linked to the action help properties.
	SetHelpKeyword (see page 276)	Introduces an interface for setting the help keyword of a client.
	SetHelpKeyword (see page 5901)	Sets the HelpKeyword of a control if the client is linked to the action help properties.
	SetHelpType (see page 277)	Introduces an interface for setting the HelpType property of a client.
	SetHelpType (see page 5901)	Sets the HelpType of a control if the client is linked to the action help properties.
	SetHint (see page 278)	Introduces an interface for setting the hint for a client.
	SetHint (see page 5901)	Sets the hint of a control if the client is linked to the action for this event.
	SetImageIndex (see page 5172)	Sets the image index of a tool button control if the client is linked to the action for this event.
	SetOnExecute (see page 2724)	This is SetOnExecute, a member of class TBasicActionLink.
	SetOnExecute (see page 5902)	Sets the execution event handler for a control if the client is linked to the action for this event.
	SetPopupMenu (see page 5902)	This is SetPopupMenu, a member of class TControlActionLink.
	SetShortCut (see page 279)	Introduces an interface for setting the shortcut for a client.
	SetVisible (see page 279)	Introduces an interface for setting the visible state of a client.
	SetVisible (see page 5902)	Sets the visible state of a control if the client is linked to the action for this event.

Public Events

	Name	Description
	OnChange (see page 2725)	Occurs when an OnChange event is triggered on the action.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2721)	Instantiates and initializes a TBasicActionLink (see page 2720) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2722)	Destroys an instance of an action link object.
	Destroy (see page 22287)	Disposes of an object instance.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 2723)	Invokes the action's Execute method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Update (see page 2724)	Invokes the action's Update method.

Public Properties

	Name	Description
	Action (see page 2725)	Indicates which action the link connects to its client.

27.1.55.2 TToolButtonActionLink Fields

The fields of the TToolButtonActionLink class are listed here.

Protected Fields

	Name	Description
	FClient (see page 5169)	This is FClient, a member of class TToolButtonActionLink.

27.1.55.2.1 TToolButtonActionLink.FClient Field

Pascal

```
FClient: TToolButton;
```

C++

```
TToolButton FClient;
```

Description

This is FClient, a member of class TToolButtonActionLink.

27.1.55.3 TToolButtonActionLink Methods

The methods of the TToolButtonActionLink class are listed here.

Protected Methods

	Name	Description
	AssignClient (see page 5170)	Assigns a client tool button that is linked to the associated action.
	IsCheckedLinked (see page 5170)	Indicates whether the checked state for the action and the client are linked.

	IsDropdownMenuLinked (see page 5170)	This is IsDropdownMenuLinked, a member of class TToolButtonActionLink.
	IsEnableDropdownLinked (see page 5171)	This is IsEnableDropdownLinked, a member of class TToolButtonActionLink.
	IsImageIndexLinked (see page 5171)	Indicates whether the image index for the action and the client are linked.
	SetChecked (see page 5171)	Sets the checked state of a tool button control if the client is linked to the action for this event.
	SetDropdownMenu (see page 5172)	This is SetDropdownMenu, a member of class TToolButtonActionLink.
	SetEnableDropdown (see page 5172)	This is SetEnableDropdown, a member of class TToolButtonActionLink.
	SetImageIndex (see page 5172)	Sets the image index of a tool button control if the client is linked to the action for this event.

27.1.55.3.1 TToolButtonActionLink.AssignClient Method

Assigns a client tool button that is linked to the associated action.

Pascal

```
procedure AssignClient(AClient: TObject); override;
```

C++

```
virtual __fastcall AssignClient(TObject * AClient);
```

Description

TToolButtonControlActionLink overrides AssignClient to ensure that the client for the link is a tool button control.

See Also

Action (see page 5797)

TToolButton (see page 5107)

27.1.55.3.2 TToolButtonActionLink.IsCheckedLinked Method

Indicates whether the checked state for the action and the client are linked.

Pascal

```
function IsCheckedLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCheckedLinked();
```

Description

IsCheckedLinked indicates whether the Checked property is linked to the appropriate client property. IsCheckedLinked returns true if the client requests that the action control this property, false otherwise.

See Also

Checked (see page 308)

Down (see page 5128)

SetChecked (see page 5171)

27.1.55.3.3 TToolButtonActionLink.IsDropdownMenuLinked Method

Pascal

```
function IsDropdownMenuLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsDropdownMenuLinked();
```

Description

This is IsDropdownMenuLinked, a member of class TToolButtonActionLink.

27.1.55.3.4 TToolButtonActionLink.IsEnableDropdownLinked Method

Pascal

```
function IsEnableDropdownLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsEnableDropdownLinked();
```

Description

This is IsEnableDropdownLinked, a member of class TToolButtonActionLink.

27.1.55.3.5 TToolButtonActionLink.IsImageIndexLinked Method

Indicates whether the image index for the action and the client are linked.

Pascal

```
function IsImageIndexLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsImageIndexLinked();
```

Description

IsImageIndexLinked indicates whether the ImageIndex property is linked to the appropriate client property. IsImageIndexLinked returns true if the client requests that the action control this property, false otherwise.

See Also

ImageIndex (see page 313)

ImageIndex (see page 5135)

SetImageIndex (see page 5172)

27.1.55.3.6 TToolButtonActionLink.SetChecked Method

Sets the checked state of a tool button control if the client is linked to the action for this event.

Pascal

```
procedure SetChecked(Value: Boolean); override;
```

C++

```
virtual __fastcall SetChecked(Boolean Value);
```

Description

SetChecked sets the checked state of a tool button control, and is called only if the client is linked for this event. Value specifies whether the checked state should be set.

See Also

Checked (see page 308)

Down (see page 5128)

IsCheckedLinked (see page 5170)

27.1.55.3.7 TToolButtonActionLink.SetDropdownMenu Method

Pascal

```
procedure SetDropdownMenu(Value: TPopupMenu); override;
```

C++

```
virtual __fastcall SetDropdownMenu(TPopupMenu Value);
```

Description

This is SetDropdownMenu, a member of class TToolButtonActionLink.

27.1.55.3.8 TToolButtonActionLink.SetEnableDropdown Method

Pascal

```
procedure SetEnableDropdown(Value: Boolean); override;
```

C++

```
virtual __fastcall SetEnableDropdown(Boolean Value);
```

Description

This is SetEnableDropdown, a member of class TToolButtonActionLink.

27.1.55.3.9 TToolButtonActionLink.SetImageIndex Method

Sets the image index of a tool button control if the client is linked to the action for this event.

Pascal

```
procedure SetImageIndex(Value: Integer); override;
```

C++

```
virtual __fastcall SetImageIndex(int Value);
```

Description

SetImageIndex sets the image index of a tool button control, and is called only if the client is linked for this event. Value specifies what the image index should be.

See Also

ImageIndex (see page 313)

ImageIndex (see page 5135)

IsImageIndexLinked (see page 5171)

27.1.56 TTrackBar Class

TTrackBar is a wrapper for a track bar control.

Class Hierarchy

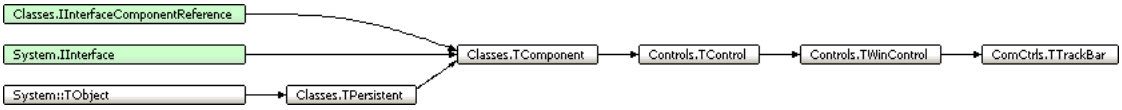

Pascal

```
TTrackBar = class(TWinControl);
```

C++

```
class TTrackBar : public TWinControl;
```

File

ComCtrls

Description

Use TTrackBar to put a track bar on a form. A track bar represents a position along a continuum using a slider and, optionally, tick marks. A track bar can also display a selected range marked by triangular ticks at the starting and ending positions of the selection.

During program execution, the slider can be moved to the desired position by dragging it with the mouse or by clicking the mouse on the bar. To use the keyboard to move the slider, press the arrow keys or the Page Up and Page Down keys.

See Also

TScrollBar (see page 21821)

Specialized Input Controls

Track Bars

27.1.56.1 TTrackBar Members

The following tables list the members exposed by TTrackBar.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5185)	Sends a CM_CHANGED message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.

	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 5185)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 5186)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 5186)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.

	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.

	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.

	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 5184)	Creates and initializes an instance of TTrackBar (see page 5172).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	SetTick (see page 5187)	Adds a tick mark to the track bar at a specified location.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.

	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 5212)	Occurs immediately after the value of Position (see page 5204) changes.
	OnContextPopup (see page 5212)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 5213)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5215)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5217)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5218)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 5219)	Occurs when a control receives the input focus.
	OnExit (see page 5221)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 5223)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 5226)	Occurs when a key is pressed.
	OnKeyUp (see page 5227)	Occurs when the user releases a key that has been pressed.
	OnStartDock (see page 5228)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 5229)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 5188)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 5191)	Specifies how the control is anchored to its parent.
	BorderWidth (see page 5192)	Specifies the width of the control's border.
	Constraints (see page 5192)	Specifies the size constraints for the control.
	Ctl3D (see page 5193)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 5193)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5194)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5194)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 5196)	Controls whether the control responds to mouse, keyboard, and timer events.
	Frequency (see page 5196)	Specifies the increment between tick marks on the track bar.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	LineSize (see page 5196)	Indicates the amount the value of Position (see page 5204) changes when the user presses the arrow keys.
	Margins (see page 5831)	Specifies the margins for the control.
	Max (see page 5197)	Specifies the maximum Position (see page 5204) of a TTrackBar (see page 5172).
	Min (see page 5197)	Specifies the minimum Position (see page 5204) of a TTrackBar (see page 5172).
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Orientation (see page 5198)	Specifies whether the track bar is horizontal or vertical.
	PageSize (see page 5198)	Specifies the amount Position (see page 5204) is changed when the PageUp or PageDown key is pressed, or when the bar is clicked.
	ParentCtl3D (see page 5198)	Determines where a component looks to determine if it should appear three dimensional.
	ParentShowHint (see page 5199)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 5199)	Identifies the pop-up menu associated with the control.
	Position (see page 5204)	Contains the current position of the slider of a TTrackBar (see page 5172).
	PositionToolTip (see page 5204)	Determines the position of the (optional) tooltip displaying the position of the track bar.
	SelEnd (see page 5205)	Specifies the position of the end point of the selection range.
	SelStart (see page 5205)	Specifies the position of the starting point of the selection range.
	ShowHint (see page 5206)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	ShowSelRange (see page 5207)	Specifies whether the track bar shows the selection range.
	SliderVisible (see page 5207)	Specifies whether the track bar's slider is visible.
	TabOrder (see page 5208)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 5208)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	ThumbLength (see page 5209)	Specifies the length of the track bar's slider.
	TickMarks (see page 5209)	Specifies the location of the tick marks.
	TickStyle (see page 5209)	Specifies how tick marks are placed on the track bar.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.

	Visible (see page 5210)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.56.2 TTrackBar.Create Constructor

Creates and initializes an instance of TTrackBar (see page 5172).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TTrackBar(TComponent * AOwner);
```

Description

Call Create to instantiate a TTrackBar (see page 5172) object at runtime. Track bars placed on forms at design time are created automatically.

Create allocates memory and initializes the following properties:

ControlStyle is set to [csCaptureMouse, csClickEvents, csSetCaption].

Frequency (see page 5196) is set to 1.

Height is set to 45 and Width to 150.

LineSize (see page 5196) is set to 1 and PageSize (see page 5198) to 2.

Max (see page 5197) is set to 10 and Min (see page 5197) to 0.

Orientation (see page 5198) is set to trHorizontal.

TickMarks (see page 5209) is set to tbBottomRight and TickStyle (see page 5209) to tsAuto.

TabStop (see page 5208) is set to true.

See Also

ControlStyle (see page 5814)

Frequency (see page 5196)

Height (see page 5825)

LineSize (see page 5196)

Max (see page 5197)

Min (see page 5197)

Orientation (see page 5198)

PageSize (see page 5198)

TabStop (see page 6211)

TickMarks (see page 5209)

TickStyle (see page 5209)

Create (see page 6123)

27.1.56.3 TTrackBar Methods

The methods of the TTrackBar class are listed here.

Protected Methods

	Name	Description
	Changed (see page 5185)	Sends a CM_CHANGED message to the control.
V	CreateParams (see page 5185)	Initializes a window-creation parameter data structure.
V	CreateWnd (see page 5186)	Creates the underlying window.
V	DestroyWnd (see page 5186)	Destroys the control's window.

Public Methods

	Name	Description
	SetTick (see page 5187)	Adds a tick mark to the track bar at a specified location.

27.1.56.3.1 TTrackBar.Changed Method

Sends a CM_CHANGED message to the control.

Pascal

```
procedure Changed; dynamic;
```

C++

```
__fastcall Changed();
```

Description

Call Changed after changing the properties of the control in a way that affects the properties of a parent object. Changed sends a CM_CHANGED message back to the control. If the control is a windowed control, this message is bubbled up to the parent so that it can make any necessary adjustments.

See Also

Parent (see page 5832)

27.1.56.3.2 TTrackBar.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard

controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

27.1.56.3.3 TTrackBar.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams (see page 6135)

CreateWindowHandle (see page 6137)

Perform (see page 5776)

DestroyWnd (see page 6141)

27.1.56.3.4 TTrackBar.DestroyWnd Method

Destroys the control's window.

Pascal

```
procedure DestroyWnd; override;
```

C++

```
virtual __fastcall DestroyWnd();
```

Description

DestroyWnd is called when the control needs to replace its window. For example, changing properties that are implemented by window parameter settings require the control to be destroyed and then recreated using the CreateWnd method.

Before the window is destroyed, DestroyWnd saves a copy of the control's text in memory, frees any device contexts, and finally calls DestroyWindowHandle. When the window is recreated, the stored copy of the control's text is assigned to the new window.

See Also

DestroyWindowHandle (see page 6140)

RecreateWnd (see page 6171)

27.1.56.3.5 TTrackBar.SetTick Method

Adds a tick mark to the track bar at a specified location.

Pascal

```
procedure SetTick(Value: Integer);
```

C++

```
__fastcall SetTick(int Value);
```

Description

Use SetTick to place a tick mark on the bar at the position that corresponds to the Value parameter. Tick marks specified in this manner are only displayed when the TickStyle (see page 5209) is set to tsManual.

See Also

Position (see page 5204)

TickStyle (see page 5209)

27.1.56.4 TTrackBar Properties

The properties of the TTrackBar class are listed here.

Published Properties

	Name	Description
	Align (see page 5188)	Determines how the control aligns within its container (parent control).
	Anchors (see page 5191)	Specifies how the control is anchored to its parent.
	BorderWidth (see page 5192)	Specifies the width of the control's border.
	Constraints (see page 5192)	Specifies the size constraints for the control.
	Ctl3D (see page 5193)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 5193)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5194)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5194)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 5196)	Controls whether the control responds to mouse, keyboard, and timer events.
	Frequency (see page 5196)	Specifies the increment between tick marks on the track bar.
	LineSize (see page 5196)	Indicates the amount the value of Position (see page 5204) changes when the user presses the arrow keys.
	Max (see page 5197)	Specifies the maximum Position (see page 5204) of a TTrackBar (see page 5172).
	Min (see page 5197)	Specifies the minimum Position (see page 5204) of a TTrackBar (see page 5172).
	Orientation (see page 5198)	Specifies whether the track bar is horizontal or vertical.
	PageSize (see page 5198)	Specifies the amount Position (see page 5204) is changed when the PageUp or PageDown key is pressed, or when the bar is clicked.
	ParentCtl3D (see page 5198)	Determines where a component looks to determine if it should appear three dimensional.
	ParentShowHint (see page 5199)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 5199)	Identifies the pop-up menu associated with the control.
	Position (see page 5204)	Contains the current position of the slider of a TTrackBar (see page 5172).
	PositionToolTip (see page 5204)	Determines the position of the (optional) tooltip displaying the position of the track bar.
	SelEnd (see page 5205)	Specifies the position of the end point of the selection range.
	SelStart (see page 5205)	Specifies the position of the starting point of the selection range.

	ShowHint (see page 5206)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	ShowSelRange (see page 5207)	Specifies whether the track bar shows the selection range.
	SliderVisible (see page 5207)	Specifies whether the track bar's slider is visible.
	TabOrder (see page 5208)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 5208)	Determines if the user can tab to a control.
	ThumbLength (see page 5209)	Specifies the length of the track bar's slider.
	TickMarks (see page 5209)	Specifies the location of the tick marks.
	TickStyle (see page 5209)	Specifies how tick marks are placed on the track bar.
	Visible (see page 5210)	Determines whether the component appears onscreen.

27.1.56.4.1 TTrackBar.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
```

```
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
```

```
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

27.1.56.4.2 TTrackBar.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control

has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.56.4.3 TTrackBar.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics or text drawn by the control is clipped to the area within the border.

See Also

ClientRect

BevelOuter (see page 6194)

27.1.56.4.4 TTrackBar.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict,

the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.56.4.5 TTrackBar.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

27.1.56.4.6 TTrackBar.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

27.1.56.4.7 TTrackBar.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

27.1.56.4.8 TTrackBar.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
```

```
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

27.1.56.4.9 TTrackBar.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.56.4.10 TTrackBar.Frequency Property

Specifies the increment between tick marks on the track bar.

Pascal

```
property Frequency: Integer;
```

C++

```
__property int Frequency;
```

Description

Use Frequency to specify the spacing of the tick marks, using the logical units used by the Position (see page 5204) property. For example, a Frequency of 5 sets a tick mark at every fifth possible increment. To provide visual feedback about the units used by the track bar, set Frequency to LineSize (see page 5196) or PageSize (see page 5198).

Note: If the value of the TickStyle (see page 5209) property is not set to tsAuto, the value of Frequency is ignored.

See Also

LineSize (see page 5196)

PageSize (see page 5198)

SetTick (see page 5187)

TickStyle (see page 5209)

27.1.56.4.11 TTrackBar.LineSize Property

Indicates the amount the value of Position (see page 5204) changes when the user presses the arrow keys.

Pascal

```
property LineSize: Integer;
```

C++

```
__property int LineSize;
```

Description

Use LineSize to specify the distance along the slider that represents a single unit. The current Position (see page 5204) is marked by a slider. The right and down arrow keys increment the value of Position (see page 5204) by LineSize, and the left and up arrow keys decrement the value of Position (see page 5204) by LineSize.

Use the PageSize (see page 5198) property to set the number of ticks moved using the Page Up and Page Down keys.

See Also

PageSize (see page 5198)

Position (see page 5204)

27.1.56.4.12 TTrackBar.Max Property

Specifies the maximum Position (see page 5204) of a TTrackBar (see page 5172).

Pascal

```
property Max: Integer;
```

C++

```
__property int Max;
```

Description

Use Max to set an upper limit to the value that can be represented using the track bar. A slider indicates the current Position (see page 5204) in a range between Min (see page 5197) and Max.

See Also

Min (see page 5197)

Position (see page 5204)

27.1.56.4.13 TTrackBar.Min Property

Specifies the minimum Position (see page 5204) of a TTrackBar (see page 5172).

Pascal

```
property Min: Integer;
```

C++

```
__property int Min;
```

Description

Use Min to set a lower limit to the value that can be represented using the track bar. A slider indicates the current Position (see page 5204) in a range between Min and Max (see page 5197).

See Also

Max (see page 5197)

Position (see page 5204)

27.1.56.4.14 TTrackBar.Orientation Property

Specifies whether the track bar is horizontal or vertical.

Pascal

```
property Orientation: TTrackBarOrientation;
```

C++

```
__property TTrackBarOrientation Orientation;
```

Description

Set Orientation to the desired orientation of the track bar. Orientation has these possible values:

Value	Meaning
tbHorizontal	The long edge of the track is parallel to the top of the form. The Min (see page 5197) position is on the left and the Max (see page 5197) position is on the right.
tbVertical	The long edge of the track is parallel to the side of the form. The Min (see page 5197) position is on the top and the Max (see page 5197) position is on the bottom.

See Also

BoundsRect (see page 5803)

27.1.56.4.15 TTrackBar.PageSize Property

Specifies the amount Position (see page 5204) is changed when the PageUp or PageDown key is pressed, or when the bar is clicked.

Pascal

```
property PageSize: Integer;
```

C++

```
__property int PageSize;
```

Description

Set PageSize to the distance along the track bar that corresponds to a single view. A slider marks the current Position (see page 5204). The Page Up key increments the value of Position (see page 5204) by PageSize, and Page Down key decrements the value of Position (see page 5204) by PageSize.

Use the LineSize (see page 5196) property to set the number of ticks moved using the arrow keys.

See Also

LineSize (see page 5196)

Position (see page 5204)

27.1.56.4.16 TTrackBar.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

27.1.56.4.17 TTrackBar.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.56.4.18 TTrackBar.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup

property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
```

```
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
```

```
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
```

```
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
```

```
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.56.4.19 TTrackBar.Position Property

Contains the current position of the slider of a TTrackBar (see page 5172).

Pascal

```
property Position: Integer;
```

C++

```
__property int Position;
```

Description

Read Position to determine the current value represented by the track bar. Position is a value in the range between Min (see page 5197) and Max (see page 5197) (inclusive). Set Position to programmatically move the slider of the track bar to a new value.

See Also

Max (see page 5197)

Min (see page 5197)

27.1.56.4.20 TTrackBar.PositionToolTip Property

Determines the position of the (optional) tooltip displaying the position of the track bar.

Pascal

```
property PositionToolTip: TPositionToolTip;
```

C++

```
__property TPositionToolTip PositionToolTip;
```

Description

You can set this property to the following values:

- ptBottom
- ptLeft

- ptNone
- ptRight
- ptLeft

27.1.56.4.21 TTrackBar.SelEnd Property

Specifies the position of the end point of the selection range.

Pascal

```
property SelEnd: Integer;
```

C++

```
__property int SelEnd;
```

Description

Set SelEnd to determine the length of a selection that begins at SelStart (see page 5205). The track bar indicates a selection range on the bar by placing special tick marks at the positions specified by the SelStart (see page 5205) and SelEnd properties and highlighting the selected portion of the bar.

SelEnd must be greater than or equal to Position (see page 5204). If SelEnd is set to a value less than Position (see page 5204), Position (see page 5204) changes to SelEnd. SelEnd must be greater than or equal to SelStart (see page 5205). If SelEnd is the same as SelStart (see page 5205), there is no selection, and the value of Position (see page 5204) is the same as SelStart (see page 5205) and SelEnd.

See Also

Position (see page 5204)

SelStart (see page 5205)

27.1.56.4.22 TTrackBar.SelStart Property

Specifies the position of the starting point of the selection range.

Pascal

```
property SelStart: Integer;
```

C++

```
__property int SelStart;
```

Description

Set SelStart to determine the beginning of a selection that runs to SelEnd (see page 5205). The track bar indicates a selection range on the bar by placing special tick marks at the positions specified by the SelStart and SelEnd (see page 5205) properties and highlighting the selected portion of the bar.

SelStart must be less than or equal to Position (see page 5204). If SelStart is set to a value greater than Position (see page 5204), Position (see page 5204) changes to SelStart. SelStart must be less than or equal to SelEnd (see page 5205). If SelStart is the same as SelEnd (see page 5205), there is no selection, and the value of Position (see page 5204) is the same as SelStart and SelEnd (see page 5205).

See Also

Position (see page 5204)

SelEnd (see page 5205)

27.1.56.4.23 TTrackBar.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.56.4.24 TTrackBar.ShowSelRange Property

Specifies whether the track bar shows the selection range.

Pascal

```
property ShowSelRange: Boolean;
```

C++

```
__property Boolean ShowSelRange;
```

Description

If set to True, the selection range on the track bar is displayed. If set to False, the selection range is not displayed.

27.1.56.4.25 TTrackBar.SliderVisible Property

Specifies whether the track bar's slider is visible.

Pascal

```
property SliderVisible: Boolean;
```

C++

```
__property Boolean SliderVisible;
```

Description

Use SliderVisible to get or set whether the track bar's slider is visible.

27.1.56.4.26 TTrackBar.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

27.1.56.4.27 TTrackBar.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

27.1.56.4.28 TTrackBar.ThumbLength Property

Specifies the length of the track bar's slider.

Pascal

```
property ThumbLength: Integer;
```

C++

```
__property int ThumbLength;
```

Description

Use ThumbLength to get or set the length of the track bar's slider.

27.1.56.4.29 TTrackBar.TickMarks Property

Specifies the location of the tick marks.

Pascal

```
property TickMarks: TTickMark;
```

C++

```
__property TTickMark TickMarks;
```

Description

Set TickMarks to specify where the tick marks should be drawn. TickMarks has these possible values:

Value	Meaning
tmBottomRight	Tick marks are on the bottom or right of the track bar, depending of the value of Orientation (see page 5198).
tmTopLeft	Tick marks are on the top or left of the track bar, depending of the value of Orientation (see page 5198).
tmBoth	Tick marks are on both sides of the track bar.

If the TickStyle (see page 5209) property is tsNone, the value of TickMarks is ignored.

See Also

TickStyle (see page 5209)

27.1.56.4.30 TTrackBar.TickStyle Property

Specifies how tick marks are placed on the track bar.

Pascal

```
property TickStyle: TTickStyle;
```

C++

```
__property TTickStyle TickStyle;
```

Description

Set TickStyle to specify whether the track bar should display tick marks, and if so, how those tick marks are set. TickStyle has these possible values:

Value	Meaning
tsAuto	Tick marks are automatically displayed at increments equal to the value of the Frequency (see page 5196) property.
tsManual	Tick marks are displayed at the Min (see page 5197) and Max (see page 5197) values. Additional tick marks can be set using the SetTick (see page 5187) method.
tsNone	No tick marks are displayed.

See Also

Frequency (see page 5196)

Max (see page 5197)

Min (see page 5197)

SetTick (see page 5187)

27.1.56.4.31 TTrackBar.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
```

```
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.56.5 TTrackBar Events

The events of the TTrackBar class are listed here.

Published Events

	Name	Description
	OnChange (see page 5212)	Occurs immediately after the value of Position (see page 5204) changes.
	OnContextPopup (see page 5212)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

	OnDragDrop (see page 5213)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5215)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5217)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5218)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 5219)	Occurs when a control receives the input focus.
	OnExit (see page 5221)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 5223)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 5226)	Occurs when a key is pressed.
	OnKeyUp (see page 5227)	Occurs when the user releases a key that has been pressed.
	OnStartDock (see page 5228)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 5229)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

27.1.56.5.1 TTrackBar.OnChange Event

Occurs immediately after the value of Position (see page 5204) changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the position of the slider may have changed. For example, if the track bar is being used to control another object, update the other object from an OnChange event handler.

See Also

Position (see page 5204)

27.1.56.5.2 TTrackBar.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.56.5.3 TTrackBar.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
```

```
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

27.1.56.5.4 TTrackBar.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
```

```
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
```

```
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.56.5.5 TTrackBar.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.56.5.6 TTrackBar.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
```

```
    S := (Sender as TLabel).Name + ' was dropped... and ';
    if Target <> nil then
      S := S + 'accepted!'
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

27.1.56.5.7 TTrackBar.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
```

```
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

27.1.56.5.8 TTrackBar.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
```

```
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

27.1.56.5.9 TTrackBar.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
```

```
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
```

```
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
  {
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

27.1.56.5.10 TTrackBar.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

27.1.56.5.11 TTrackBar.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
```

```
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

27.1.56.5.12 TTrackBar.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

27.1.56.5.13 TTrackBar.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

27.1.57 TTreeNode Class

TTreeNode describes an individual node in a tree view control.

Class Hierarchy

Pascal

```
TTreeNode = class(TPersistent);
```

C++

```
class TTreeNode : public TPersistent;
```

File

ComCtrls

Description

Each node in a tree view control consists of a label and an optional bitmapped image. Each item can be the parent of a list of subitems. By clicking an item, the user can expand or collapse the associated list of subitems.

See Also

TImageList (see page 6078)

TTreeNodes (see page 5266)

TTreeView (see page 5293)

27.1.57.1 TTreeNode Members

The following tables list the members exposed by TTreeNode.

Protected Methods

	Name	Description
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetState (see page 5248)	This is GetState, a member of class TTreeNode.
	SetSelectedBit (see page 5251)	This is SetSelectedBit, a member of class TTreeNode.
	SetState (see page 5251)	This is SetState, a member of class TTreeNode.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AlphaSort (see page 5234)	Sorts tree nodes.
V	Assign (see page 5234)	Copies the properties of another tree view node.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Collapse (see page 5235)	Collapses a node.
	Create (see page 5232)	Creates an instance of TTreeNode (see page 5230).
	CustomSort (see page 5236)	Triggers node sorting.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 5237)	Destroys the node and all its children.
	DeleteChildren (see page 5238)	Deletes all children of the node.
V	Destroy (see page 5233)	Destroys the tree node object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DisplayRect (see page 5238)	Returns the bounding rectangle for a tree node.
	EditText (see page 5240)	Allows editing a tree node.
	EndEdit (see page 5241)	Ends the editing of a node's label.
	Expand (see page 5242)	Expands the node to display all child nodes.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	getFirstChild (see page 5242)	Returns the first child node of a tree node.
	GetHandle (see page 5243)	Returns the Handle (see page 5258) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetLastChild (see page 5243)	Returns the last immediate child node of the calling node.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNext (see page 5243)	Returns the next node after the calling node in the tree view.
	GetNextChild (see page 5244)	Returns the next child node after Value.
	getNextSibling (see page 5245)	Returns the next node in the tree view at the same level as the calling node.
	GetNextVisible (see page 5245)	Returns the next visible node in the tree view after the calling node.
	GetPrev (see page 5246)	Returns the previous node in the tree view before the calling node.
	GetPrevChild (see page 5246)	Returns the previous child node before Value.
	getPrevSibling (see page 5247)	Returns the previous node before the calling node and at the same level.
	GetPrevVisible (see page 5247)	Returns the previous visible node before the calling node.
	HasAsParent (see page 5248)	Returns true if Value is a parent node of the calling node.
	IndexOf (see page 5248)	Returns the position of an immediate child node of the calling node.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsFirstNode (see page 5249)	Indicates whether the tree node is very first.
	MakeVisible (see page 5249)	Makes a node visible.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	MoveTo (see page 5249)	Moves the node to another location in the tree view.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	AbsoluteIndex (see page 5252)	Indicates the index of the tree node relative to the first tree node in a tree node.
R	Count (see page 5253)	Indicates the number of direct descendants of a tree node.
	Cut (see page 5254)	Indicates if the tree node object is drawn as if selected as part of a cut and paste operation.
	Data (see page 5254)	Points to application-defined data associated with the tree node.
R	Deleting (see page 5256)	Indicates whether a node's Destroy (see page 5233) method has been called and the node is in the process of being deleted.
	DropTarget (see page 5257)	Indicates whether the tree node is drawn as a drag and drop target.
	Expanded (see page 5257)	Specifies whether the tree node is expanded.
	Focused (see page 5257)	Indicates whether the node appears to have focus.
R	Handle (see page 5258)	Contains the window Handle of the tree view that contains the node.
	HasChildren (see page 5258)	Indicates whether a node has any children.
	ImageIndex (see page 5259)	Specifies which image is displayed when a node is in its normal state and is not currently selected.
R	Index (see page 5259)	Specifies the position of the node in the list of child nodes maintained by its parent node.
R	IsVisible (see page 5259)	Indicates whether the tree node is currently visible in the tree view image.
	Item (see page 5260)	Provides access to a child node by its position in the list of child nodes.
R	ItemId (see page 5260)	Contains a handle that uniquely identifies each node in a tree view.
R	Level (see page 5260)	Indicates the level of indentation of a node within the tree view control.
	OverlayIndex (see page 5263)	Determines which image from the image list is used as an overlay mask.
R	Owner (see page 5263)	Indicates which tree nodes object contains the tree node.
R	Parent (see page 5263)	Identifies the parent node of the tree node.
	Selected (see page 5264)	Determines whether the node is selected.
	SelectedIndex (see page 5265)	Specifies the index in the tree view's image list of the image displayed for the node when it is selected.
	StateIndex (see page 5265)	Indicates which image from the StateImages list to display for the node.
	Text (see page 5265)	Specifies the label that identifies a tree node.
R	TreeView (see page 5266)	Specifies the tree view that displays the node.

27.1.57.2 TTreeNode.Create Constructor

Creates an instance of TTreeNode (see page 5230).

Pascal

```
constructor Create(AOwner: TTreeNodes);
```

C++

```
__fastcall TTreeNode(TTreeNodes AOwner);
```

Description

Call Create to instantiate a tree node at runtime. Nodes added to a tree view control at design time are created automatically. Use the methods of the TTreeNodes (see page 5266) object instead to create nodes as part of an existing tree view.

After calling the inherited constructor, Create sets the OverlayIndex (see page 5263) and StateIndex (see page 5265) properties to -1 and then sets the Owner (see page 5263) property of the tree node to the AOwner TTreeNodes (see page 5266) object.

See Also

OverlayIndex (see page 5263)

Owner (see page 5263)

StateIndex (see page 5265)

Add (see page 5270)

Insert (see page 5285)

27.1.57.3 TTreeNode.Destroy Destructor

Destroys the tree node object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTreeNode();
```

Description

Do not call Destroy directly in an application. Instead, use the Delete (see page 5237) method. Delete (see page 5237) removes the image of the node and all its children from the tree view window, and guards against recursive deletes.

See Also

Delete (see page 5237)

27.1.57.4 TTreeNode Methods

The methods of the TTreeNode class are listed here.

Protected Methods

	Name	Description
	GetState (see page 5248)	This is GetState, a member of class TTreeNode.
	SetSelectedBit (see page 5251)	This is SetSelectedBit, a member of class TTreeNode.
	SetState (see page 5251)	This is SetState, a member of class TTreeNode.

Public Methods

	Name	Description
	AlphaSort (see page 5234)	Sorts tree nodes.
V	Assign (see page 5234)	Copies the properties of another tree view node.
	Collapse (see page 5235)	Collapses a node.
	CustomSort (see page 5236)	Triggers node sorting.
	Delete (see page 5237)	Destroys the node and all its children.
	DeleteChildren (see page 5238)	Deletes all children of the node.
	DisplayRect (see page 5238)	Returns the bounding rectangle for a tree node.
	EditText (see page 5240)	Allows editing a tree node.
	EndEdit (see page 5241)	Ends the editing of a node's label.
	Expand (see page 5242)	Expands the node to display all child nodes.
	getFirstChild (see page 5242)	Returns the first child node of a tree node.
	GetHandle (see page 5243)	Returns the Handle (see page 5258) property.
	GetLastChild (see page 5243)	Returns the last immediate child node of the calling node.

	GetNext (see page 5243)	Returns the next node after the calling node in the tree view.
	GetNextChild (see page 5244)	Returns the next child node after Value.
	getNextSibling (see page 5245)	Returns the next node in the tree view at the same level as the calling node.
	GetNextVisible (see page 5245)	Returns the next visible node in the tree view after the calling node.
	GetPrev (see page 5246)	Returns the previous node in the tree view before the calling node.
	GetPrevChild (see page 5246)	Returns the previous child node before Value.
	getPrevSibling (see page 5247)	Returns the previous node before the calling node and at the same level.
	GetPrevVisible (see page 5247)	Returns the previous visible node before the calling node.
	HasAsParent (see page 5248)	Returns true if Value is a parent node of the calling node.
	IndexOf (see page 5248)	Returns the position of an immediate child node of the calling node.
	IsFirstNode (see page 5249)	Indicates whether the tree node is very first.
	MakeVisible (see page 5249)	Makes a node visible.
V	MoveTo (see page 5249)	Moves the node to another location in the tree view.

27.1.57.4.1 TTreeNode.AlphaSort Method

Sorts tree nodes.

Pascal

```
function AlphaSort(ARecurse: Boolean = False): Boolean;
```

C++

```
__fastcall Boolean AlphaSort(Boolean ARecurse = False);
```

Description

AlphaSort triggers node sorting or resorting of the node's child nodes. If an OnCompare event handler is defined for the associated TCustomTreeView (see page 3957), that routine determines sort order. If no OnCompare handler is defined, nodes are sorted by a simple case-sensitive compare of their captions.

The optional ARecurse parameter (default false) specifies that sorting should recursively descend the node tree and sort each subtree in turn. By default, only the node's immediate children are sorted.

Calling AlphaSort is equivalent to calling CustomSort (see page 5236) with a nil (Delphi) or NULL (C++) procedure parameter and a data parameter of 0.

To sort the top-level nodes, use the AlphaSort method of TTreeNodes (see page 5266).

See Also

CustomSort (see page 5236)

SortType (see page 4014)

Text (see page 5265)

AlphaSort (see page 3972)

AlphaSort (see page 5279)

27.1.57.4.2 TTreeNode.Assign Method

Copies the properties of another tree view node.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

If the Source parameter is a TTreeNode (see page 5230) object, Assign copies the Text (see page 5265), Data (see page 5254), ImageIndex (see page 5259), SelectedIndex (see page 5265), OverlayIndex (see page 5263), Focused (see page 5257), DropTarget (see page 5257), Cut (see page 5254), and HasChildren (see page 5258) properties from the source node. If Source is any other type of object, Assign calls its inherited method, which copies information from any object that can copy to a tree node in its AssignTo method.

Note: While Assign copies the HasChildren (see page 5258) property, it does not copy the child nodes. Be sure to copy any descendants after assigning the properties of another node that has children.

See Also

Cut (see page 5254)

Data (see page 5254)

DropTarget (see page 5257)

Focused (see page 5257)

HasChildren (see page 5258)

ImageIndex (see page 5259)

OverlayIndex (see page 5263)

SelectedIndex (see page 5265)

Text (see page 5265)

AssignTo (see page 2931)

27.1.57.4.3 TTreeNode.Collapse Method

Collapses a node.

Pascal

```
procedure Collapse(Recurse: Boolean);
```

C++

```
__fastcall Collapse(Boolean Recurse);
```

Description

When a node is collapsed, all of its subnodes are hidden. The plus button may be displayed, depending on whether the tree view's ShowButtons property is set. If Recurse is true, then all subnodes will be collapsed as well. When the node is next expanded the children will still be collapsed. If Recurse is false, the child nodes won't be collapsed and the next time the node is expanded, the children will be in the same state as when Collapse was called.

See Also

Expanded (see page 5257)

OnCollapsed (see page 4018)

OnCollapsing (see page 4018)

FullCollapse (see page 3983)

27.1.57.4.4 TTreeNode.CustomSort Method

Triggers node sorting.

Pascal

```
function CustomSort(SortProc: TTVCompare; Data: Longint; ARecurse: Boolean = False): Boolean;
```

C++

```
__fastcall Boolean CustomSort(TTVCompare SortProc, Longint Data, Boolean ARecurse = False);
```

Description

CustomSort triggers node sorting or resorting of the node's child nodes, using a comparison routine indicated by the SortProc parameter. The Data (see page 5254) parameter is passed to the comparison routine. The optional ARecurse parameter (default false) specifies that sorting should recursively descend the node tree and sort each subtree in turn.

If SortProc is nil (Delphi) or NULL (C++), a default comparison routine is used. The default routine uses the OnCompare event handler for the associated TCustomTreeview object, if defined. If the OnCompare event handler is not defined, the default routine uses a simple case-sensitive compare of node captions.

In the comparison routine, the lParam1 and lParam2 parameters refer to two nodes when cast to TTreeNode (see page 5230). The lParamSort parameter is the value previously passed to the Data (see page 5254) parameter of CustomSort. The return value of the comparison routine indicates the relative sort order of lParam1 and lParam2:

Return Value	Meaning
< 0	lParam1 comes before lParam2
0	lParam1 and lParam2 are equivalent
> 0	lParam2 comes before lParam1

To sort the top-level nodes, use the CustomSort method of TTreeNodes (see page 5266).

C++ Examples:

```
/*
This example shows how to use the CustomSort method to order
a subtree in case-insensitive alphabetical order (either
forward or backward). The callback function CompareFunc
below calls the global AnsiStrIComp function to perform the
actual comparison.  This example requires two buttons and a
populated TreeView.
*/
int CALLBACK CompareFunc(long lParam1, long lParam2, long Reverse)
{
  TTreeNode *Node1 = ((TTreeNode *)lParam1);
  TTreeNode *Node2 = ((TTreeNode *)lParam2);
  if ((Node1 == NULL) || (Node2 == NULL)) return 0;
  int GT = AnsiStrIComp(Node1->Text.c_str(), Node2->Text.c_str());
  if (Reverse)
    return -GT;
  return GT;
}

/*
This procedure can then be used as a parameter to CustomSort
to sort the nodes of the tree view.  To sort in ascending
order, call:
*/

#include <CommCtrl.hpp>
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTreeNode *node = TreeView1->Selected;
  node->CustomSort(CompareFunc, 0); // Sorts in ascending order
}
```

Delphi Examples:

```
{
This example shows how to use the CustomSort method to order
a subtree in case-insensitive alphabetical order (either
forward or backward). The callback function CompareFunc
below calls the global AnsiStrIComp function to perform the
actual comparison.  This example requires two buttons and a
populated TreeView.
}
function CustomSortProc(Node1, Node2: TTreeNode; Data: Longint): Integer; stdcall;
begin
  if Data = 0 then
    Result :=
      AnsiStrIComp(Pchar(Node1.Text), PChar(Node2.Text))
  else
    Result :=
      -AnsiStrIComp(PChar(Node1.Text), PChar(Node2.Text));
end;

{
This procedure can then be used as a parameter to CustomSort
to sort the children of a node:
}

procedure TForm1.Button1Click(Sender: TObject);
var node: TTreeNode;
begin
  node := TreeView1.Selected;
  node.CustomSort(@CustomSortProc, 0);  { Sorts in ascending order }
end;

procedure TForm1.Button2Click(Sender: TObject);
var node: TTreeNode;
begin
  node := TreeView1.Selected;
  node.CustomSort(@CustomSortProc, 1);  { Sorts in descending order }
end;
```

See Also

AlphaSort (see page 5234)

CustomSort (see page 3978)

27.1.57.4.5 TTreeNode.Delete Method

Destroys the node and all its children.

Pascal

```
procedure Delete;
```

C++

```
__fastcall Delete();
```

Description

Use the Delete method to delete a tree node and free all associated memory.

See Also

DeleteChildren (see page 5238)

Deleting (see page 5256)

Destroy (see page 5233)

OnDeletion (see page 4024)

27.1.57.4.6 TTreeNode.DeleteChildren Method

Deletes all children of the node.

Pascal

```
procedure DeleteChildren;
```

C++

```
__fastcall DeleteChildren();
```

Description

Use the DeleteChildren method to delete all children of a tree node, freeing all associated memory.

See Also

Delete (see page 5237)

Deleting (see page 5256)

OnDeletion (see page 4024)

27.1.57.4.7 TTreeNode.DisplayRect Method

Returns the bounding rectangle for a tree node.

Pascal

```
function DisplayRect(TextOnly: Boolean): TRect;
```

C++

```
__fastcall TRect DisplayRect(Boolean TextOnly);
```

Description

If the TextOnly parameter is true, the bounding rectangle includes only the text of the node. Otherwise, it includes the entire line that the node occupies in the tree-view control.

C++ Examples:

```
/*
The following example shows how the OnCustomDrawItem event
handler draws items and lines of the tree view after the
OnCustomDraw event handler has filled in the background.
*/
void __fastcall TCustomDrawForm::TVCustomDrawItem(TCustomTreeView *Sender,
      TTreeNode *Node, TCustomDrawState State, bool &DefaultDraw)
{
  TRect NodeRect;
/*
    If DefaultDraw it is true, any of the node's font
    properties can be changed. Note also that when
    DefaultDraw = True, Windows draws the buttons and
```

```
    ignores our font background colors, using instead the
    TreeView's Color property.
*/
    if (State.Contains(cdsSelected))
    {
      TV->Canvas->Font->Assign(SelectedFontDialog->Font);
      TV->Canvas->Brush->Color = SelBkgColorDialog->Color;
    };

    DefaultDraw = FDefaultDrawItem;
/*
    DefaultDraw = False means you have to handle all the
    item drawing yourself, including the buttons, lines,
    images, and text.
*/
    if (!DefaultDraw)
    {
      //draw the selection rect.
      if (State.Contains(cdsSelected))
      {
        NodeRect = Node->DisplayRect(True);
        TV->Canvas->FillRect(NodeRect);
      };
      NodeRect = Node->DisplayRect(False);

      if (None1->Checked)
      //no bitmap, so paint in the background color.
      {
        TV->Canvas->Brush->Color = BkgColorDialog->Color;
        TV->Canvas->Brush->Style = FBrushStyle;
        TV->Canvas->FillRect(NodeRect);
      }
      else
        //don't paint over the background bitmap.
        TV->Canvas->Brush->Style = bsClear;

      NodeRect.Left = NodeRect.Left + (Node->Level * TV->Indent);
      // NodeRect.Left now represents the left-most portion
      // of the expand button
      DrawButton(&NodeRect, Node); // See the CustomDraw demo

      NodeRect.Left = NodeRect.Left + TV->Indent + FButtonSize;
      //NodeRect->Left is now the leftmost portion of the image.
      DrawImage(&NodeRect, Node->ImageIndex); // See the CustomDraw demo

      NodeRect.Left = NodeRect.Left + ImageList->Width;
      //Now we are finally in a position to draw the text.

      TV->Canvas->TextOut(NodeRect.Left, NodeRect.Top, Node->Text);
    };
}
```

Delphi Examples:

```
{
The following example shows how the OnCustomDrawItem event
handler draws items and lines of the tree view after the
OnCustomDraw event handler has filled in the background.
}
procedure TCustomDrawForm.TVCustomDrawItem(Sender: TCustomTreeView; Node: TTreeNode;
  State: TCustomDrawState; var DefaultDraw: Boolean);
var
  NodeRect: TRect;
begin
  with TV.Canvas do
  begin
{
```

```
      If DefaultDraw it is true, any of the node's font
      properties can be changed. Note also that when
      DefaultDraw = True, Windows draws the buttons and
      ignores our font background colors, using instead the
      TreeView's Color property.
  }
      if cdsSelected in State then
      begin
        Font.Assign(SelectedFontDialog.Font);
        Brush.Color := SelBkgColorDialog.Color;
      end;

      DefaultDraw := False; // FDefaultDrawItem;
  {
      DefaultDraw = False means you have to handle all the
      item drawing yourself, including the buttons, lines,
      images, and text.
  }
      if not DefaultDraw then
      begin
        //draw the selection rect.
        if cdsSelected in State then
        begin
          NodeRect := Node.DisplayRect(True);
          FillRect(NodeRect);
        end;
        NodeRect := Node.DisplayRect(False);

        if None1.Checked then
        //no bitmap, so paint in the background color.
        begin
          Brush.Color := BkgColorDialog.Color;
          Brush.Style := FBrushStyle;
          FillRect(NodeRect)
        end
        else
          //don't paint over the background bitmap.
          Brush.Style := bsClear;

        NodeRect.Left := NodeRect.Left + (Node.Level * TV.Indent);
        // NodeRect.Left now represents the left-most portion
        // of the expand button
        DrawButton(NodeRect, Node); // See the CustomDraw demo

        NodeRect.Left := NodeRect.Left + TV.Indent + FButtonSize;
        //NodeRect.Left is now the leftmost portion of the image.
        DrawImage(NodeRect, Node.ImageIndex); // See the CustomDraw demo

        NodeRect.Left := NodeRect.Left + ImageList.Width;
        //Now we are finally in a position to draw the text.

        TextOut(NodeRect.Left, NodeRect.Top, Node.Text);
      end;
    end;
  end;
```

See Also

Text (see page 5265)

27.1.57.4.8 TTreeNode.EditText Method

Allows editing a tree node.

Pascal

```
function EditText: Boolean;
```

C++

```
__fastcall Boolean EditText();
```

Description

Call EditText to allow a user to edit tree node label text.

Delphi Examples:

```
{
This example is an event handler for a popup menu item. The
handler enables editing of the current item in a TListView
control. The example would also work with a TTreeView
control, or with any instance of a control class derived
from TCustomViewControl.
}
procedure TForm1.EditItem1Click(Sender: TObject);
begin
  TreeView1.Selected.EditText;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  MyTreeNode1, MyTreeNode2: TTreeNode;
begin
  with TreeView1.Items do
  begin
    Clear; { remove any existing nodes }
    MyTreeNode1 := Add(nil, 'RootTreeNode1'); { Add a root node }
    { Add a child node to the node just added }
    AddChild(MyTreeNode1,'ChildNode1');

    {Add another root node}
    MyTreeNode2 := Add(MyTreeNode1, 'RootTreeNode2');
    {Give MyTreeNode2 to a child }
    AddChild(MyTreeNode2,'ChildNode2');

    {Change MyTreeNode2 to ChildNode2 }
    { and add a child node to it}
    MyTreeNode2 := TreeView1.Items[3];
    AddChild(MyTreeNode2,'ChildNode2a');

    {Add another child to ChildNode2, after ChildNode2a }
    AddChild(MyTreeNode2,'ChildNode2b');

    {add another root node}
    Add(MyTreeNode1, 'RootTreeNode3');
  end;
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  PopupMenu1.Popup(Left + X, Top + Y);
end;
```

27.1.57.4.9 TTreeNode.EndEdit Method

Ends the editing of a node's label.

Pascal

```
procedure EndEdit(Cancel: Boolean);
```

C++

```
__fastcall EndEdit(Boolean Cancel);
```

Description

Call EndEdit to take the node out of edit mode. If the Cancel parameter is true, all changes made by the user are discarded. If Cancel is false, EndEdit updates the node's Text (see page 5265) property and the tree view's OnEdited event occurs.

See Also

Text (see page 5265)

OnEdited (see page 4024)

27.1.57.4.10 TTreeNode.Expand Method

Expands the node to display all child nodes.

Pascal

```
procedure Expand(Recurse: Boolean);
```

C++

```
__fastcall Expand(Boolean Recurse);
```

Description

When a node is expanded, its immediate subnodes are displayed. The minus '-' button may be displayed, depending on whether the tree view's ShowButtons property is set. If Recurse is true, all descendants of the immediate subnodes are expanded as well.

See Also

Collapse (see page 5235)

ShowButtons (see page 4013)

27.1.57.4.11 TTreeNode.getFirstChild Method

Returns the first child node of a tree node.

Pascal

```
function getFirstChild: TTreeNode;
```

C++

```
__fastcall TTreeNode getFirstChild();
```

Description

Call GetFirstChild to access the first child node of the tree view node. If the node has no children, GetFirstChild returns nil (Delphi) or NULL (C++).

Note: GetFirstChild returns the same value as Item (see page 5260)[0].

See Also

GetNextChild (see page 5244)

GetNodeAt (see page 3987)

GetPrevChild (see page 5246)

Item (see page 5260)

27.1.57.4.12 TTreeNode.GetHandle Method

Returns the Handle (see page 5258) property.

Pascal

```
function GetHandle: HWND;
```

C++

```
__fastcall HWND GetHandle();
```

Description

Calling GetHandle is the same as reading the Handle (see page 5258) property.

See Also

Handle (see page 5258)

27.1.57.4.13 TTreeNode.GetLastChild Method

Returns the last immediate child node of the calling node.

Pascal

```
function GetLastChild: TTreeNode;
```

C++

```
__fastcall TTreeNode GetLastChild();
```

Description

Call GetLastChild to find the last immediate child of a node. If the calling node has no children, GetLastChild returns nil (Delphi) or NULL (C++). GetLastChild returns the same value as Item (see page 5260)[Count (see page 5253)-1].

See Also

Count (see page 5253)

GetNext (see page 5243)

GetNodeAt (see page 3987)

Item (see page 5260)

27.1.57.4.14 TTreeNode.GetNext Method

Returns the next node after the calling node in the tree view.

Pascal

```
function GetNext: TTreeNode;
```

C++

```
__fastcall TTreeNode GetNext();
```

Description

If the calling node is the last node, GetNext returns nil (Delphi) or NULL (C++). Otherwise, it returns the next node including nodes that aren't visible and child nodes. To get the next node at the same level as the calling node, use GetNextSibling. To get the next visible node, use GetNextVisible (see page 5245).

C++ Examples:

```
/*
The following example adds the Text for all items in the
tree view to the list box.  This example requires a ListBox
and a populated TreeView.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TTreeNode *CurItem = TreeView1->Items->GetFirstNode();
  while (CurItem)
  {
    ListBox1->Items->Add(CurItem->Text);
    CurItem = CurItem->GetNext();
  }
```

Delphi Examples:

```
{
The following example adds the Text for all items in the
tree view to the list box.  This example requires a ListBox
and a populated TreeView.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  CurItem: TTreeNode;
begin
  CurItem := TreeView1.Items.GetFirstNode;
  while CurItem <> nil do
  begin
    ListBox1.Items.Add(CurItem.Text);
    CurItem := CurItem.GetNext;
  end;
end;
```

See Also

getNextSibling (see page 5245)

GetNextChild (see page 5244)

GetNextVisible (see page 5245)

27.1.57.4.15 TTreeNode.GetNextChild Method

Returns the next child node after Value.

Pascal

```
function GetNextChild(Value: TTreeNode): TTreeNode;
```

C++

```
__fastcall TTreeNode GetNextChild(TTreeNode Value);
```

Description

Call GetNextChild to locate the next node in the list of immediate children of the tree view node. If the calling node has no children or there is no node after Value, GetNextChild returns nil (Delphi) or NULL (C++).

C++ Examples:

```
/*
The following code tests to see if the currently selected
item has a sibling; if so, HasSibling will be set to true.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
TTreeNode *SelNode = TreeView1->Selected;
TTreeNode *ParentNode = SelNode->Parent;
bool HasSibling = (ParentNode->GetPrevChild(SelNode) ||
                   ParentNode->GetNextChild(SelNode));
  if (HasSibling) Edit1->Text = "True";
  else Edit1->Text = "False";
}
```

Delphi Examples:

```
{
The following code tests to see if the currently selected
item has a sibling; if so, HasSibling will be set to true.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  HasSibling: Boolean;
  SelNode: TTreeNode;
  ParentNode: TTreeNode;
begin
  SelNode := TreeView1.Selected;
  ParentNode := SelNode.Parent;
  HasSibling := (ParentNode.GetPrevChild(SelNode) <> nil) or
                (ParentNode.GetNextChild(SelNode) <> nil);
  if (HasSibling) then Edit1.Text := 'True'
  else Edit1.Text := 'False';
end;
```

See Also

GetNext (see page 5243)

GetPrevChild (see page 5246)

Item (see page 5260)

27.1.57.4.16 TTreeNode.getNextSibling Method

Returns the next node in the tree view at the same level as the calling node.

Pascal

```
function getNextSibling: TTreeNode;
```

C++

```
__fastcall TTreeNode getNextSibling();
```

Description

GetNextSibling will return the next node, regardless of whether it's visible. To find the next node in the tree view including child nodes, use GetNext (see page 5243).

See Also

GetNext (see page 5243)

27.1.57.4.17 TTreeNode.GetNextVisible Method

Returns the next visible node in the tree view after the calling node.

Pascal

```
function GetNextVisible: TTreeNode;
```

C++

```
__fastcall TTreeNode GetNextVisible();
```

Description

Use GetNextVisible when iterating through all the visible nodes in the tree view. A node is visible if all its parent nodes are expanded.

See Also

GetNext (see page 5243)

IsVisible (see page 5259)

27.1.57.4.18 TTreeNode.GetPrev Method

Returns the previous node in the tree view before the calling node.

Pascal

```
function GetPrev: TTreeNode;
```

C++

```
__fastcall TTreeNode GetPrev();
```

Description

GetPrev will return the previous node whether or not it is visible. To get the previous visible node, use GetPrevVisible (see page 5247).

See Also

GetPrevChild (see page 5246)

getPrevSibling (see page 5247)

GetPrevVisible (see page 5247)

27.1.57.4.19 TTreeNode.GetPrevChild Method

Returns the previous child node before Value.

Pascal

```
function GetPrevChild(Value: TTreeNode): TTreeNode;
```

C++

```
__fastcall TTreeNode GetPrevChild(TTreeNode Value);
```

Description

Call GetPrevChild to locate the previous node in the list of immediate children of the tree view node. If the calling node has no children or there is no node before Value, GetPrevChild returns nil (Delphi) or NULL (C++).

C++ Examples:

```
/*
The following code tests to see if the currently selected
item has a sibling; if so, HasSibling will be set to true.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
TTreeNode *SelNode = TreeView1->Selected;
TTreeNode *ParentNode = SelNode->Parent;
```

```
bool HasSibling = (ParentNode->GetPrevChild(SelNode) ||
                   ParentNode->GetNextChild(SelNode));
  if (HasSibling) Edit1->Text = "True";
  else Edit1->Text = "False";
}
```

Delphi Examples:

```
{
The following code tests to see if the currently selected
item has a sibling; if so, HasSibling will be set to true.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  HasSibling: Boolean;
  SelNode: TTreeNode;
  ParentNode: TTreeNode;
begin
  SelNode := TreeView1.Selected;
  ParentNode := SelNode.Parent;
  HasSibling := (ParentNode.GetPrevChild(SelNode) <> nil) or
                (ParentNode.GetNextChild(SelNode) <> nil);
  if (HasSibling) then Edit1.Text := 'True'
  else Edit1.Text := 'False';
end;
```

See Also

GetNextChild (see page 5244)

GetPrev (see page 5246)

27.1.57.4.20 TTreeNode.getPrevSibling Method

Returns the previous node before the calling node and at the same level.

Pascal

```
function getPrevSibling: TTreeNode;
```

C++

```
__fastcall TTreeNode getPrevSibling();
```

Description

GetPrevSibling returns the previous sibling node, regardless of whether it's visible. To find the previous node in the tree view including all levels, use GetPrev (see page 5246).

See Also

GetPrev (see page 5246)

27.1.57.4.21 TTreeNode.GetPrevVisible Method

Returns the previous visible node before the calling node.

Pascal

```
function GetPrevVisible: TTreeNode;
```

C++

```
__fastcall TTreeNode GetPrevVisible();
```

Description

To get the previous node, including non-visible, use GetPrev (see page 5246).

See Also

GetPrev (see page 5246)

27.1.57.4.22 TTreeNode.GetState Method

Pascal

```
function GetState(NodeState: TNodeState): Boolean;
```

C++

```
__fastcall Boolean GetState(TNodeState NodeState);
```

Description

This is GetState, a member of class TTreeNode.

27.1.57.4.23 TTreeNode.HasAsParent Method

Returns true if Value is a parent node of the calling node.

Pascal

```
function HasAsParent(Value: TTreeNode): Boolean;
```

C++

```
__fastcall Boolean HasAsParent(TTreeNode Value);
```

Description

Use the HasAsParent method to determine if a node is a parent to a particular node.

See Also

Parent (see page 5263)

27.1.57.4.24 TTreeNode.IndexOf Method

Returns the position of an immediate child node of the calling node.

Pascal

```
function IndexOf(Value: TTreeNode): Integer;
```

C++

```
__fastcall int IndexOf(TTreeNode Value);
```

Description

Call IndexOf to obtain the position of a child node among the children of the calling node. If Value isn't an immediate child of the calling node, IndexOf returns -1. The first child node has an index on 0, the second an index of 1, and so on.

See Also

HasChildren (see page 5258)

Index (see page 5259)

27.1.57.4.25 TTreeNode.IsFirstNode Method

Indicates whether the tree node is very first.

Pascal

```
function IsFirstNode: Boolean;
```

C++

```
__fastcall Boolean IsFirstNode();
```

Description

IsFirstNode returns true for the first node, with no parent node or previous sibling.

See Also

Expanded (see page 5257)

27.1.57.4.26 TTreeNode.MakeVisible Method

Makes a node visible.

Pascal

```
procedure MakeVisible;
```

C++

```
__fastcall MakeVisible();
```

Description

MakeVisible makes the node visible to the user, expanding parent nodes and scrolling as necessary.

See Also

Expanded (see page 5257)

IsVisible (see page 5259)

27.1.57.4.27 TTreeNode.MoveTo Method

Moves the node to another location in the tree view.

Pascal

```
procedure MoveTo(Destination: TTreeNode; Mode: TNodeAttachMode); virtual;
```

C++

```
virtual __fastcall MoveTo(TTreeNode Destination, TNodeAttachMode Mode);
```

Description

The Destination parameter specifies a node that will be this node's parent or sibling after the move. The Mode parameter specifies the new relationship between this node and the destination node.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
```

```
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

InsertObject (see page 5286)

Index (see page 5259)

27.1.57.4.28 TTreeNode.SetSelectedBit Method

Pascal

```
procedure SetSelectedBit(Value: Boolean);
```

C++

```
__fastcall SetSelectedBit(Boolean Value);
```

Description

This is SetSelectedBit, a member of class TTreeNode.

27.1.57.4.29 TTreeNode.SetState Method

Pascal

```
procedure SetState(NodeState: TNodeState; Value: Boolean);
```

C++

```
__fastcall SetState(TNodeState NodeState, Boolean Value);
```

Description

This is SetState, a member of class TTreeNode.

27.1.57.5 TTreeNode Properties

The properties of the TTreeNode class are listed here.

Public Properties

	Name	Description
R	AbsoluteIndex (see page 5252)	Indicates the index of the tree node relative to the first tree node in a tree node.
R	Count (see page 5253)	Indicates the number of direct descendants of a tree node.
	Cut (see page 5254)	Indicates if the tree node object is drawn as if selected as part of a cut and paste operation.
	Data (see page 5254)	Points to application-defined data associated with the tree node.
R	Deleting (see page 5256)	Indicates whether a node's Destroy (see page 5233) method has been called and the node is in the process of being deleted.
	DropTarget (see page 5257)	Indicates whether the tree node is drawn as a drag and drop target.
	Expanded (see page 5257)	Specifies whether the tree node is expanded.
	Focused (see page 5257)	Indicates whether the node appears to have focus.
R	Handle (see page 5258)	Contains the window Handle of the tree view that contains the node.
	HasChildren (see page 5258)	Indicates whether a node has any children.
	ImageIndex (see page 5259)	Specifies which image is displayed when a node is in its normal state and is not currently selected.
R	Index (see page 5259)	Specifies the position of the node in the list of child nodes maintained by its parent node.
R	IsVisible (see page 5259)	Indicates whether the tree node is currently visible in the tree view image.
	Item (see page 5260)	Provides access to a child node by its position in the list of child nodes.
R	ItemId (see page 5260)	Contains a handle that uniquely identifies each node in a tree view.
R	Level (see page 5260)	Indicates the level of indentation of a node within the tree view control.
	OverlayIndex (see page 5263)	Determines which image from the image list is used as an overlay mask.
R	Owner (see page 5263)	Indicates which tree nodes object contains the tree node.
R	Parent (see page 5263)	Identifies the parent node of the tree node.
	Selected (see page 5264)	Determines whether the node is selected.

	SelectedIndex (see page 5265)	Specifies the index in the tree view's image list of the image displayed for the node when it is selected.
	StateIndex (see page 5265)	Indicates which image from the StateImages list to display for the node.
	Text (see page 5265)	Specifies the label that identifies a tree node.
R	TreeView (see page 5266)	Specifies the tree view that displays the node.

27.1.57.5.1 TTreeNode.AbsoluteIndex Property

Indicates the index of the tree node relative to the first tree node in a tree node.

Pascal

```
property AbsoluteIndex: Integer;
```

C++

```
__property int AbsoluteIndex;
```

Description

Use AbsoluteIndex to determine absolute position of a node in a tree nodes object. The first tree node in a tree nodes object has an index of 0 and subsequent nodes are numbered sequentially. If a node has any children, its AbsoluteIndex is one less than the index of its first child.

Delphi Examples:

```
{
This example requires three buttons, four TEdits, two labels
and a populated TreeView.  The following code defines a
record type of TMyRec and a record pointer type of PMyRec.
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Edit1: TEdit;
    TreeView1: TTreeView;
    Label2: TLabel;
    Label3: TLabel;
    Button3: TButton;
    Edit4: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  type
  PMyRec = ^TMyRec;
  TMyRec = record
    FName: string;
    LName: string;
  end;


var
  Form1: TForm1;

implementation

{$R *.dfm}
```

```
{
Assuming these types are used, the following code adds a
node to TreeView1 as the last sibling of the selected node.
A TMyRec record is associated with the added item. The FName
and LName fields are obtained from edit boxes Edit1 and
Edit2.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  MyRecPtr: PMyRec;
  TreeViewIndex: LongInt;
begin
  New(MyRecPtr);
  MyRecPtr^.FName := Edit1.Text;
  MyRecPtr^.LName := Edit2.Text;
  with TreeView1 do
  begin
    TreeViewIndex := Selected.AbsoluteIndex;
    if Items.Count = 0 then
      Items.AddObject(nil, 'Item' + IntToStr(TreeViewIndex), MyRecPtr)
    else if (TreeViewIndex < Items.Count) and (TreeViewIndex >= 0) then
      Items.AddObject(Items[TreeViewIndex], 'Item' + IntToStr(TreeViewIndex), MyRecPtr);
  end;
end;
{
After an item containing a TMyRec record has been added, the
following code retrieves the FName and LName values
associated with the item and displays the values in a label.
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  if (TreeView1.Selected.Data <> nil) then { query only works on new nodes }
    Edit3.Text := PMyRec(TreeView1.Selected.Data)^.FName + ' ' +
                  PMyRec(TreeView1.Selected.Data)^.LName;
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
  Edit4.Text := IntToStr(TreeView1.Selected.AbsoluteIndex);
end;
```

See Also

Index (see page 5259)

TTreeNodes (see page 5266)

Items (see page 4004)

27.1.57.5.2 TTreeNode.Count Property

Indicates the number of direct descendants of a tree node.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use Count to determine how many child nodes belong to a tree node. Count includes only immediate children, and not their descendants. Count can be useful when iterating through the children of a tree node.

See Also

HasChildren (see page 5258)

Item (see page 5260)

TTreeNodes (see page 5266)

27.1.57.5.3 TTreeNode.Cut Property

Indicates if the tree node object is drawn as if selected as part of a cut and paste operation.

Pascal

```
property Cut: Boolean;
```

C++

```
__property Boolean Cut;
```

Description

Use Cut to alter the appearance of the tree node when it is selected as part of a cut and paste operation.

See Also

DropTarget (see page 5257)

Focused (see page 5257)

Selected (see page 5264)

27.1.57.5.4 TTreeNode.Data Property

Points to application-defined data associated with the tree node.

Pascal

```
property Data: Pointer;
```

C++

```
__property void * Data;
```

Description

Use the Data property to associate data with a tree node. Data allows applications to quickly access information about the entity represented by the node.

C++ Examples:

```
/*
This example requires two buttons, three TEdits, a label and
a populated TreeView.  The following code defines a record
type of TMyRec and a record pointer type of PMyRec.
*/
typedef struct MyRec
{
  AnsiString FName, LName;
} TMyRec;

typedef TMyRec* PMyRec;
/*
Assuming these types are used, the following code adds a
node to TreeView1 as the last sibling of the node specified
by the Index.  A TMyRec record is associated with the added
```

```
item. The FName and LName fields are obtained from edit
boxes Edit1 and Edit2. The Index parameter is obtained from
edit box Edit3.  The item is added only if the Index is a
valid value.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PMyRec  MyRecPtr;
  int TreeViewIndex;
  TTreeNodes* items;

  MyRecPtr = new TMyRec;
  MyRecPtr->FName = Edit1->Text;
  MyRecPtr->LName = Edit2->Text;
  TreeViewIndex = StrToInt(Edit3->Text);
  items = TreeView1->Items;
  if (items->Count == 0)
    items->AddObject(NULL, "Item" + IntToStr(TreeViewIndex), MyRecPtr);
  else if ((TreeViewIndex < items->Count) && (TreeViewIndex >= 0))
    items->AddObject(items->Item[TreeViewIndex], "Item" + IntToStr(TreeViewIndex) + "after" +
items->Item[TreeViewIndex]->Text, MyRecPtr);
}
/*
After an item containing a TMyRec record has been added, the
following code retrieves the FName and LName values
associated with the item and displays the values in a label.
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  if (TreeView1->Selected->Data != NULL)
    Label1->Caption = PMyRec(TreeView1->Selected->Data)->FName + " " +
                  PMyRec(TreeView1->Selected->Data)->LName;
}
```

Delphi Examples:

```
{
This example requires three buttons, four TEdits, two labels
and a populated TreeView.  The following code defines a
record type of TMyRec and a record pointer type of PMyRec.
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Edit1: TEdit;
    TreeView1: TTreeView;
    Label2: TLabel;
    Label3: TLabel;
    Button3: TButton;
    Edit4: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  type
  PMyRec = ^TMyRec;
  TMyRec = record
    FName: string;
    LName: string;
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

{
Assuming these types are used, the following code adds a
node to TreeView1 as the last sibling of the selected node.
A TMyRec record is associated with the added item. The FName
and LName fields are obtained from edit boxes Edit1 and
Edit2.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  MyRecPtr: PMyRec;
  TreeViewIndex: LongInt;
begin
  New(MyRecPtr);
  MyRecPtr^.FName := Edit1.Text;
  MyRecPtr^.LName := Edit2.Text;
  with TreeView1 do
  begin
    TreeViewIndex := Selected.AbsoluteIndex;
    if Items.Count = 0 then
      Items.AddObject(nil, 'Item' + IntToStr(TreeViewIndex), MyRecPtr)
    else if (TreeViewIndex < Items.Count) and (TreeViewIndex >= 0) then
      Items.AddObject(Items[TreeViewIndex], 'Item' + IntToStr(TreeViewIndex), MyRecPtr);
  end;
end;
{
After an item containing a TMyRec record has been added, the
following code retrieves the FName and LName values
associated with the item and displays the values in a label.
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  if (TreeView1.Selected.Data <> nil) then { query only works on new nodes }
    Edit3.Text := PMyRec(TreeView1.Selected.Data)^.FName + ' ' +
                  PMyRec(TreeView1.Selected.Data)^.LName;
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
  Edit4.Text := IntToStr(TreeView1.Selected.AbsoluteIndex);
end;
```

27.1.57.5.5 TTreeNode.Deleting Property

Indicates whether a node's Destroy (see page 5233) method has been called and the node is in the process of being deleted.

Pascal

```
property Deleting: Boolean;
```

C++

```
__property Boolean Deleting;
```

Description

Deleting returns true if the node object is in the process of being deleted. Use Deleting to prevent infinite recursion in event handlers.

See Also

Destroy (see page 5233)

OnDeletion (see page 4024)

27.1.57.5.6 TTreeNode.DropTarget Property

Indicates whether the tree node is drawn as a drag and drop target.

Pascal

```
property DropTarget: Boolean;
```

C++

```
__property Boolean DropTarget;
```

Description

Use DropTarget to indicate that the node is a drag and drop target. When DropTarget is true, the node is drawn in a style used to indicate a drag and drop target.

Note: Setting DropTarget to true does cause the node to automatically accept dragged objects when they are released. The application must still implement the drop behavior.

See Also

Cut (see page 5254)

Selected (see page 5264)

Focused (see page 5257)

MoveTo (see page 5249)

27.1.57.5.7 TTreeNode.Expanded Property

Specifies whether the tree node is expanded.

Pascal

```
property Expanded: Boolean;
```

C++

```
__property Boolean Expanded;
```

Description

When a tree node is expanded, the minus button is shown if the ShowButtons property of the tree view is true and child nodes are displayed. Set Expanded to true to display the children of a node. Set Expanded to false to collapse the node, hiding all of its descendants.

See Also

Expand (see page 5242)

Collapse (see page 5235)

27.1.57.5.8 TTreeNode.Focused Property

Indicates whether the node appears to have focus.

Pascal

```
property Focused: Boolean;
```

C++

```
__property Boolean Focused;
```

Description

Tree view nodes are not windowed controls and so can't receive input focus. However, they can be edited by the user when the tree view control has focus. When Focused is true, the node is surrounded by a standard focus rectangle and the user can edit the label. Use Focused to determine if a particular node in a tree view can currently be edited by the user.

See Also

Selected (see page 5264)

OnChange (see page 4017)

OnChanging (see page 4018)

27.1.57.5.9 TTreeNode.Handle Property

Contains the window Handle of the tree view that contains the node.

Pascal

```
property Handle: HWND;
```

C++

```
__property HWND Handle;
```

Description

Use Handle to obtain the handle of the tree view that owns the node. This handle can be passed as a parameter to Windows API function calls that require a handle to the tree view.

See Also

Handle (see page 6207)

27.1.57.5.10 TTreeNode.HasChildren Property

Indicates whether a node has any children.

Pascal

```
property HasChildren: Boolean;
```

C++

```
__property Boolean HasChildren;
```

Description

HasChildren is true if the node has subnodes, or false if the node has no subnodes. If ShowButtons of the tree view is true, and HasChildren is true, a plus (+) button will appear to the left of the node when it is collapsed, and a minus (-) button will appear when the node is expanded.

Note: If a node has no children, setting HasChildren to true will show a (+) plus button, but will not add any child nodes and the node cannot be expanded.

See Also

Count (see page 5253)

HasAsParent (see page 5248)

getFirstChild (see page 5242)

27.1.57.5.11 TTreeNode.ImageIndex Property

Specifies which image is displayed when a node is in its normal state and is not currently selected.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

Use the ImageIndex property with the Images property of the tree view to specify the image for the node in its normal state.

See Also

Images (see page 4001)

27.1.57.5.12 TTreeNode.Index Property

Specifies the position of the node in the list of child nodes maintained by its parent node.

Pascal

```
property Index: Integer;
```

C++

```
__property int Index;
```

Description

Use Index to determine the position of the node relative to its sibling nodes. The first child of the parent node has an Index value of 0, and subsequent children are indexed sequentially.

See Also

AbsoluteIndex (see page 5252)

Item (see page 5260)

27.1.57.5.13 TTreeNode.IsVisible Property

Indicates whether the tree node is currently visible in the tree view image.

Pascal

```
property IsVisible: Boolean;
```

C++

```
__property Boolean IsVisible;
```

Description

A node is visible if it is on level 0 or if all its parents are expanded. IsVisible indicates whether the node is part of the current tree view image. It does not indicate whether or not the node is scrolled into view when the tree view image is larger than the size of the tree view control.

See Also

Expanded (see page 5257)

27.1.57.5.14 TTreeNode.Item Property

Provides access to a child node by its position in the list of child nodes.

Pascal

```
property Item [Index: Integer]: TTreeNode;
```

C++

```
__property TTreeNode Item[int Index];
```

Description

Use Item to access a child node based on its Index (see page 5259) property. The first child node has an index of 0, the second an index of 1, and so on.

See Also

Index (see page 5259)

operator_[]

27.1.57.5.15 TTreeNode.ItemId Property

Contains a handle that uniquely identifies each node in a tree view.

Pascal

```
property ItemId: HTreeItem;
```

C++

```
__property HTreeItem ItemId;
```

Description

Use this property to reference the nodes when making Windows API calls or calling the GetNode method of the TTreeNodes (see page 5266) that owns the item.

See Also

GetNode (see page 5285)

27.1.57.5.16 TTreeNode.Level Property

Indicates the level of indentation of a node within the tree view control.

Pascal

```
property Level: Integer;
```

C++

```
__property int Level;
```

Description

The value of Level is 0 for nodes on the top level. The value of Level is 1 for their children, and so on.

C++ Examples:

```
/*
The following example shows how the OnCustomDrawItem event
handler draws items and lines of the tree view after the
OnCustomDraw event handler has filled in the background.
*/
void __fastcall TCustomDrawForm::TVCustomDrawItem(TCustomTreeView *Sender,
      TTreeNode *Node, TCustomDrawState State, bool &DefaultDraw)
{
  TRect NodeRect;
/*
    If DefaultDraw it is true, any of the node's font
    properties can be changed. Note also that when
    DefaultDraw = True, Windows draws the buttons and
    ignores our font background colors, using instead the
    TreeView's Color property.
*/
    if (State.Contains(cdsSelected))
    {
      TV->Canvas->Font->Assign(SelectedFontDialog->Font);
      TV->Canvas->Brush->Color = SelBkgColorDialog->Color;
    };

    DefaultDraw = FDefaultDrawItem;
/*
    DefaultDraw = False means you have to handle all the
    item drawing yourself, including the buttons, lines,
    images, and text.
*/
    if (!DefaultDraw)
    {
      //draw the selection rect.
      if (State.Contains(cdsSelected))
      {
        NodeRect = Node->DisplayRect(True);
        TV->Canvas->FillRect(NodeRect);
      };
      NodeRect = Node->DisplayRect(False);

      if (None1->Checked)
      //no bitmap, so paint in the background color.
      {
        TV->Canvas->Brush->Color = BkgColorDialog->Color;
        TV->Canvas->Brush->Style = FBrushStyle;
        TV->Canvas->FillRect(NodeRect);
      }
      else
        //don't paint over the background bitmap.
        TV->Canvas->Brush->Style = bsClear;

      NodeRect.Left = NodeRect.Left + (Node->Level * TV->Indent);
      // NodeRect.Left now represents the left-most portion
      // of the expand button
      DrawButton(&NodeRect, Node); // See the CustomDraw demo

      NodeRect.Left = NodeRect.Left + TV->Indent + FButtonSize;
      //NodeRect->Left is now the leftmost portion of the image.
      DrawImage(&NodeRect, Node->ImageIndex); // See the CustomDraw demo

      NodeRect.Left = NodeRect.Left + ImageList->Width;
      //Now we are finally in a position to draw the text.

      TV->Canvas->TextOut(NodeRect.Left, NodeRect.Top, Node->Text);
    };
}
```

Delphi Examples:

```
{
The following example shows how the OnCustomDrawItem event
handler draws items and lines of the tree view after the
OnCustomDraw event handler has filled in the background.
}
procedure TCustomDrawForm.TVCustomDrawItem(Sender: TCustomTreeView; Node: TTreeNode;
  State: TCustomDrawState; var DefaultDraw: Boolean);
var
  NodeRect: TRect;
begin
  with TV.Canvas do
  begin
{
    If DefaultDraw it is true, any of the node's font
    properties can be changed. Note also that when
    DefaultDraw = True, Windows draws the buttons and
    ignores our font background colors, using instead the
    TreeView's Color property.
}
    if cdsSelected in State then
    begin
      Font.Assign(SelectedFontDialog.Font);
      Brush.Color := SelBkgColorDialog.Color;
    end;

    DefaultDraw := False; // FDefaultDrawItem;
{
    DefaultDraw = False means you have to handle all the
    item drawing yourself, including the buttons, lines,
    images, and text.
}
    if not DefaultDraw then
    begin
      //draw the selection rect.
      if cdsSelected in State then
      begin
        NodeRect := Node.DisplayRect(True);
        FillRect(NodeRect);
      end;
      NodeRect := Node.DisplayRect(False);

      if None1.Checked then
      //no bitmap, so paint in the background color.
      begin
        Brush.Color := BkgColorDialog.Color;
        Brush.Style := FBrushStyle;
        FillRect(NodeRect)
      end
      else
        //don't paint over the background bitmap.
        Brush.Style := bsClear;

      NodeRect.Left := NodeRect.Left + (Node.Level * TV.Indent);
      // NodeRect.Left now represents the left-most portion
      // of the expand button
      DrawButton(NodeRect, Node); // See the CustomDraw demo

      NodeRect.Left := NodeRect.Left + TV.Indent + FButtonSize;
      //NodeRect.Left is now the leftmost portion of the image.
      DrawImage(NodeRect, Node.ImageIndex); // See the CustomDraw demo

      NodeRect.Left := NodeRect.Left + ImageList.Width;
      //Now we are finally in a position to draw the text.

      TextOut(NodeRect.Left, NodeRect.Top, Node.Text);
    end;
```

```
  end;
end;
```

See Also

AbsoluteIndex (see page 5252)

27.1.57.5.17 TTreeNode.OverlayIndex Property

Determines which image from the image list is used as an overlay mask.

Pascal

```
property OverlayIndex: Integer;
```

C++

```
__property int OverlayIndex;
```

Description

An overlay mask is an image drawn transparently over another image in the tree view. For example, to indicate that a node is no longer available, use an overlay image that puts an X over the current node's image.

See Also

ImageIndex (see page 5259)

Images (see page 4001)

27.1.57.5.18 TTreeNode.Owner Property

Indicates which tree nodes object contains the tree node.

Pascal

```
property Owner: TTreeNodes;
```

C++

```
__property TTreeNodes Owner;
```

Description

Use the Owner property to determine the owner of the tree node. Do not confuse Owner with the TreeView (see page 5266) property. Owner is the list of nodes used by the tree view to manage its nodes. The TreeView (see page 5266) property is the tree view that uses that list.

See Also

TreeView (see page 5266)

27.1.57.5.19 TTreeNode.Parent Property

Identifies the parent node of the tree node.

Pascal

```
property Parent: TTreeNode;
```

C++

```
__property TTreeNode Parent;
```

Description

A Parent node is one level higher than the node and contains the node as a subnode.

C++ Examples:

```
/*
The following code tests to see if the currently selected
item has a sibling; if so, HasSibling will be set to true.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
TTreeNode *SelNode = TreeView1->Selected;
TTreeNode *ParentNode = SelNode->Parent;
bool HasSibling = (ParentNode->GetPrevChild(SelNode) ||
                   ParentNode->GetNextChild(SelNode));
  if (HasSibling) Edit1->Text = "True";
  else Edit1->Text = "False";
}
```

Delphi Examples:

```
{
The following code tests to see if the currently selected
item has a sibling; if so, HasSibling will be set to true.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  HasSibling: Boolean;
  SelNode: TTreeNode;
  ParentNode: TTreeNode;
begin
  SelNode := TreeView1.Selected;
  ParentNode := SelNode.Parent;
  HasSibling := (ParentNode.GetPrevChild(SelNode) <> nil) or
                (ParentNode.GetNextChild(SelNode) <> nil);
  if (HasSibling) then Edit1.Text := 'True'
  else Edit1.Text := 'False';
end;
```

See Also

Level (see page 5260)

TopItem (see page 4015)

27.1.57.5.20 TTreeNode.Selected Property

Determines whether the node is selected.

Pascal

```
property Selected: Boolean;
```

C++

```
__property Boolean Selected;
```

Description

Set Selected to true to select the node. The appearance of a selected node depends on whether it has the focus and on whether the system colors are used for selection. When a node becomes selected, the tree view's OnChanging and OnChanged events are triggered.

See Also

Focused (see page 5257)

OnChange (see page 4017)

OnChanging (see page 4018)

27.1.57.5.21 TTreeNode.SelectedIndex Property

Specifies the index in the tree view's image list of the image displayed for the node when it is selected.

Pascal

```
property SelectedIndex: Integer;
```

C++

```
__property int SelectedIndex;
```

Description

Use the SelectedIndex property to specify an image to display when the tree node is selected.

See Also

ImageIndex (see page 5259)

OnGetSelectedIndex (see page 4026)

Selected (see page 5264)

27.1.57.5.22 TTreeNode.StateIndex Property

Indicates which image from the StateImages list to display for the node.

Pascal

```
property StateIndex: Integer;
```

C++

```
__property int StateIndex;
```

Description

Use StateIndex to display an additional image for the node that reflects state information. If StateIndex is -1 then no state image is drawn.

Valid index values are 1..15 only. Values of 0 or -1 mean no state icon.

See Also

ImageIndex (see page 5259)

StateImages (see page 4014)

27.1.57.5.23 TTreeNode.Text Property

Specifies the label that identifies a tree node.

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

Use Text to specify the string that is displayed in the tree view. The value of Text can be assigned directly at run-time or can be set within the TreeView (see page 5266) Items Editor while modifying the Items property of the TTreeView (see page 5293) component. If the tree view allows users to edit its nodes, read Text to determine the value given the node by the user.

See Also

Data (see page 5254)

ReadOnly (see page 4006)

27.1.57.5.24 TTreeNode.TreeView Property

Specifies the tree view that displays the node.

Pascal

```
property TreeView: TCustomTreeView;
```

C++

```
__property TCustomTreeView TreeView;
```

Description

Use TreeView to determine the tree view associated with the tree node.

See Also

Owner (see page 5263)

27.1.58 TTreeNodes Class

TTreeNodes maintains a list of tree nodes in a tree view control.

Class Hierarchy

Pascal

```
TTreeNodes = class(TPersistent);
```

C++

```
class TTreeNodes : public TPersistent;
```

File

ComCtrls

Description

The Items property of the tree view control is a TTreeNodes object and maintains the collection of nodes in the tree view. Nodes can be added, deleted, inserted, and moved within the tree view. Access the nodes in the tree view through the Items property of the tree view.

See Also

TTreeView (see page 5293)

TTreeNode (see page 5230)

27.1.58.1 TTreeNodes Members

The following tables list the members exposed by TTreeNodes.

Protected Methods

	Name	Description
	AddItem (see page 5275)	This is AddItem, a member of class TTreeNodes.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CreateItem (see page 5281)	This is CreateItem, a member of class TTreeNodes.
V	DefineProperties (see page 5281)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetCount (see page 5283)	This is GetCount, a member of class TTreeNodes.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetItem (see page 5286)	This is SetItem, a member of class TTreeNodes.
	SetUpdateState (see page 5286)	This is SetUpdateState, a member of class TTreeNodes.

Protected Properties

	Name	Description
R	Reading (see page 5290)	This is Reading, a member of class TTreeNodes.

Public Methods

	Name	Description
	Add (see page 5270)	Adds a new tree node to a tree view control.
	AddChild (see page 5271)	Adds a new tree node to a tree view.
	AddChildFirst (see page 5273)	Adds a new tree node to a tree view.
	AddChildObject (see page 5273)	Adds a new tree node containing data to a tree view.
	AddChildObjectFirst (see page 5274)	Adds a new tree node containing data to a tree view.
	AddFirst (see page 5275)	Adds a new tree node to a tree view.
	AddNode (see page 5276)	Add (see page 5270) a node to a tree view.
	AddObject (see page 5276)	Adds a new node containing data to a tree view.
	AddObjectFirst (see page 5279)	Adds a new node containing data to a tree view.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AlphaSort (see page 5279)	Sort tree nodes.
V	Assign (see page 5279)	Discards any current property information and replaces it with the information from the Source.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 5280)	Prevents the updating of the tree view until the EndUpdate (see page 5283) method is called.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 5280)	Deletes all tree nodes contained from the list managed by TTreeNodes (see page 5266).
	Create (see page 5268)	Creates an instance of a TTreeNodes (see page 5266) object
	CustomSort (see page 5281)	Sort tree nodes.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 5282)	Removes a node from the tree view.
V	Destroy (see page 5269)	Destroys an instance of TTreeNodes (see page 5266).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 5283)	Re-enables screen repainting and node reindexing that was turned off with the BeginUpdate (see page 5280) method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetEnumerator (see page 5283)	This is GetEnumerator, a member of class TTreeNodes.
	GetFirstNode (see page 5284)	Returns the first tree node in the tree view.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNode (see page 5285)	Returns the tree node given the ItemId of the tree node.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 5285)	Inserts a tree node into the tree view before the node specified by the Node parameter.
	InsertNode (see page 5285)	Inserts a tree node with data.
	InsertObject (see page 5286)	Inserts a tree node containing data into the tree view before the node specified by the Node parameter.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Count (see page 5287)	Indicates the number of nodes maintained by the TTreeNodes (see page 5266) object.
R	Handle (see page 5289)	Contains the window handle of the tree view control that owns the tree nodes object.
R	Item (see page 5289)	Lists all tree nodes managed by the TTreeNodes (see page 5266) object.
R	Owner (see page 5290)	Specifies the tree view control that uses the tree nodes object to implement its Items property.

27.1.58.2 TTreeNodes.Create Constructor

Creates an instance of a TTreeNodes (see page 5266) object

Pascal

```
constructor Create(AOwner: TCustomTreeView);
```

C++

```
__fastcall TTreeNodes(TCustomTreeView AOwner);
```

Description

Call Create to instantiate a TTreeNodes (see page 5266) object at runtime. The TTreeNodes (see page 5266) object used by TTreeView (see page 5293) is created automatically from the constructor of the tree view object.

After calling the inherited constructor, Create sets the Owner (see page 5290) property of the tree nodes object to the value of the AOwner parameter.

See Also

Owner (see page 5290)

TTreeView (see page 5293)

27.1.58.3 TTreeNodes.Destroy Destructor

Destroys an instance of TTreeNodes (see page 5266).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTreeNodes();
```

Description

Do not call Destroy directly in an application. The tree view that owns the TTreeNodes (see page 5266) object destroys the tree nodes object by calling its Free method.

See Also

Free (see page 22300)

27.1.58.4 TTreeNodes Methods

The methods of the TTreeNodes class are listed here.

Protected Methods

	Name	Description
	AddItem (see page 5275)	This is AddItem, a member of class TTreeNodes.
	CreateItem (see page 5281)	This is CreateItem, a member of class TTreeNodes.
V	DefineProperties (see page 5281)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetCount (see page 5283)	This is GetCount, a member of class TTreeNodes.
	SetItem (see page 5286)	This is SetItem, a member of class TTreeNodes.
	SetUpdateState (see page 5286)	This is SetUpdateState, a member of class TTreeNodes.

Public Methods

	Name	Description
	Add (see page 5270)	Adds a new tree node to a tree view control.
	AddChild (see page 5271)	Adds a new tree node to a tree view.
	AddChildFirst (see page 5273)	Adds a new tree node to a tree view.
	AddChildObject (see page 5273)	Adds a new tree node containing data to a tree view.
	AddChildObjectFirst (see page 5274)	Adds a new tree node containing data to a tree view.
	AddFirst (see page 5275)	Adds a new tree node to a tree view.
	AddNode (see page 5276)	Add (see page 5270) a node to a tree view.
	AddObject (see page 5276)	Adds a new node containing data to a tree view.
	AddObjectFirst (see page 5279)	Adds a new node containing data to a tree view.
	AlphaSort (see page 5279)	Sort tree nodes.
V	Assign (see page 5279)	Discards any current property information and replaces it with the information from the Source.
	BeginUpdate (see page 5280)	Prevents the updating of the tree view until the EndUpdate (see page 5283) method is called.
	Clear (see page 5280)	Deletes all tree nodes contained from the list managed by TTreeNodes (see page 5266).
	CustomSort (see page 5281)	Sort tree nodes.
	Delete (see page 5282)	Removes a node from the tree view.
	EndUpdate (see page 5283)	Re-enables screen repainting and node reindexing that was turned off with the BeginUpdate (see page 5280) method.

	GetEnumerator (see page 5283)	This is GetEnumerator, a member of class TTreeNodes.
	GetFirstNode (see page 5284)	Returns the first tree node in the tree view.
	GetNode (see page 5285)	Returns the tree node given the ItemId of the tree node.
	Insert (see page 5285)	Inserts a tree node into the tree view before the node specified by the Node parameter.
	InsertNode (see page 5285)	Inserts a tree node with data.
	InsertObject (see page 5286)	Inserts a tree node containing data into the tree view before the node specified by the Node parameter.

27.1.58.4.1 TTreeNodes.Add Method

Adds a new tree node to a tree view control.

Pascal

```
function Add(Sibling: TTreeNode; const S: string): TTreeNode;
```

C++

```
__fastcall TTreeNode Add(TTreeNode Sibling, const AnsiString S);
```

Description

The node is added as the last sibling of the Node parameter. The S parameter specifies the Text property of the new node. Add returns the node that has been added. If the tree view is sorted, Add inserts the node in the correct sort order position rather than as the last child of the Node parameter's parent.

C++ Examples:

```
/*
The following example demonstrates how to add nodes and
child nodes to a TTreeView control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTreeNode *Node1;
  TreeView1->Items->Clear(); // remove any existing nodes
  // Add a root node
  TreeView1->Items->Add(NULL, "RootNode1");

  /* Set MyTreeNode to first node in tree view and add a child node to it */
  Node1 = TreeView1->Items->Item[0];
  TreeView1->Items->AddChild(Node1,"ChildNode1");

  // Add another root node
  TreeView1->Items->Add(Node1, "RootNode2");

  /* Reset Node1 to RootNode2 and add a child node to it */
  Node1 = TreeView1->Items->Item[2];
  TreeView1->Items->AddChild(Node1,"ChildNode2");

  /* Reset Node1 to ChildNode2 and add a child node to it */
  Node1 = TreeView1->Items->Item[3];
  TreeView1->Items->AddChild(Node1,"ChildNode2a");

   /* Add another child to ChildNode2 following ChildNode2a */
  TreeView1->Items->AddChild(Node1,"ChildNode2b");

  // add another root node
  TreeView1->Items->Add(TreeView1->Items->Item[0], "RootTreeNode3");
}
```

Delphi Examples:

```
{
The following example demonstrates how to add nodes and
```

```
  child nodes to a TTreeView control.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    MyTreeNode1, MyTreeNode2: TTreeNode;
  begin
    with TreeView1.Items do
    begin
      Clear; { remove any existing nodes }
      MyTreeNode1 := Add(nil, 'RootTreeNode1'); { Add a root node }
      { Add a child node to the node just added }
      AddChild(MyTreeNode1,'ChildNode1');

      {Add another root node}
      MyTreeNode2 := Add(MyTreeNode1, 'RootTreeNode2');
      {Give MyTreeNode2 to a child }
      AddChild(MyTreeNode2,'ChildNode2');

      {Change MyTreeNode2 to ChildNode2 }
      { and add a child node to it}
      MyTreeNode2 := TreeView1.Items[3];
      AddChild(MyTreeNode2,'ChildNode2a');

      {Add another child to ChildNode2, after ChildNode2a }
      AddChild(MyTreeNode2,'ChildNode2b');

      {add another root node}
      Add(MyTreeNode1, 'RootTreeNode3');
    end;
  end;
```

See Also

AddChild (see page 5271)

Insert (see page 5285)

27.1.58.4.2 TTreeNodes.AddChild Method

Adds a new tree node to a tree view.

Pascal

```
function AddChild(Parent: TTreeNode; const S: string): TTreeNode;
```

C++

```
__fastcall TTreeNode AddChild(TTreeNode Parent, const AnsiString S);
```

Description

The node is added as a child of the node specified by the Node parameter. It is added to the end of Node's list of child nodes. The S parameter specifies the Text property of the new node. AddChild returns the node that has been added. If the tree view is sorted, AddChild inserts the node in the correct sort order position, rather than as the last child of the Node parameter.

C++ Examples:

```
/*
The following example demonstrates how to add nodes and
child nodes to a TTreeView control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTreeNode *Node1;
  TreeView1->Items->Clear(); // remove any existing nodes
  // Add a root node
  TreeView1->Items->Add(NULL, "RootNode1");
```

```
  /* Set MyTreeNode to first node in tree view and add a child node to it */
  Node1 = TreeView1->Items->Item[0];
  TreeView1->Items->AddChild(Node1,"ChildNode1");

  // Add another root node
  TreeView1->Items->Add(Node1, "RootNode2");

  /* Reset Node1 to RootNode2 and add a child node to it */
  Node1 = TreeView1->Items->Item[2];
  TreeView1->Items->AddChild(Node1,"ChildNode2");

  /* Reset Node1 to ChildNode2 and add a child node to it */
  Node1 = TreeView1->Items->Item[3];
  TreeView1->Items->AddChild(Node1,"ChildNode2a");

   /* Add another child to ChildNode2 following ChildNode2a */
  TreeView1->Items->AddChild(Node1,"ChildNode2b");

  // add another root node
  TreeView1->Items->Add(TreeView1->Items->Item[0], "RootTreeNode3");
}
```

Delphi Examples:

```
{
The following example demonstrates how to add nodes and
child nodes to a TTreeView control.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  MyTreeNode1, MyTreeNode2: TTreeNode;
begin
  with TreeView1.Items do
  begin
    Clear; { remove any existing nodes }
    MyTreeNode1 := Add(nil, 'RootTreeNode1'); { Add a root node }
    { Add a child node to the node just added }
    AddChild(MyTreeNode1,'ChildNode1');

    {Add another root node}
    MyTreeNode2 := Add(MyTreeNode1, 'RootTreeNode2');
    {Give MyTreeNode2 to a child }
    AddChild(MyTreeNode2,'ChildNode2');

    {Change MyTreeNode2 to ChildNode2 }
    { and add a child node to it}
    MyTreeNode2 := TreeView1.Items[3];
    AddChild(MyTreeNode2,'ChildNode2a');

    {Add another child to ChildNode2, after ChildNode2a }
    AddChild(MyTreeNode2,'ChildNode2b');

    {add another root node}
    Add(MyTreeNode1, 'RootTreeNode3');
  end;
end;
```

See Also

Add (see page 5270)

AddChildFirst (see page 5273)

27.1.58.4.3 TTreeNodes.AddChildFirst Method

Adds a new tree node to a tree view.

Pascal

```
function AddChildFirst(Parent: TTreeNode; const S: string): TTreeNode;
```

C++

```
__fastcall TTreeNode AddChildFirst(TTreeNode Parent, const AnsiString S);
```

Description

Use AddChildFirst to insert a node as the first child of the node specified by the Node parameter. The S parameter specifies the Text property of the new node. Nodes that appear after the added node are moved down one row and reindexed with valid Index values. AddChildFirst returns the node that has been added.

See Also

AddChild (see page 5271)

AddFirst (see page 5275)

27.1.58.4.4 TTreeNodes.AddChildObject Method

Adds a new tree node containing data to a tree view.

Pascal

```
function AddChildObject(Parent: TTreeNode; const S: string; Ptr: Pointer): TTreeNode;
```

C++

```
__fastcall TTreeNode AddChildObject(TTreeNode Parent, const AnsiString S, void * Ptr);
```

Description

The node is added as the last child of the node specified by the Node parameter. The S parameter specifies the Text property of the new node. The Ptr parameter specifies the Data property value of the new node. AddChildObject returns the node that has been added.

Note: The memory referenced by Ptr is not freed when the tree nodes object is freed.

C++ Examples:

```
/*
The following example adds a new child to the selected
item's list of child items in the tree view control. The new
item is identified by the text 'New Item'. The TBitmap
object is attached to the new item.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *MyBitMap =
    new Graphics::TBitmap();
  MyBitMap->LoadFromFile("..\\littleB16.bmp");
  TreeView1->Items->BeginUpdate(); // keep the node from painting until it's built
  TreeView1->Items->AddChildObject(
  TreeView1->Selected, "New Item", MyBitMap);
  TreeView1->Items->EndUpdate();
}
```

Delphi Examples:

```
{
The following example adds a new child to the selected
item's list of child items in the tree view control. The new
item is identified by the text 'New Item'. The TBitmap
object is attached to the new item.
}
procedure TForm1.Button1Click(Sender: TObject);
var MyBitMap : TBitmap;
begin
  MyBitMap := TBitmap.Create;
  MyBitMap.LoadFromFile('littleB16.bmp');
  TreeView1.Items.BeginUpdate;  // keep the node from painting until it's built
  TreeView1.Items.AddChildObject(
    TreeView1.Selected, 'New Item', MyBitMap);
  TreeView1.Items.EndUpdate;
end;
```

See Also

AddObject (see page 5276)

Data (see page 5254)

27.1.58.4.5 TTreeNodes.AddChildObjectFirst Method

Adds a new tree node containing data to a tree view.

Pascal

```
function AddChildObjectFirst(Parent: TTreeNode; const S: string; Ptr: Pointer): TTreeNode;
```

C++

```
__fastcall TTreeNode AddChildObjectFirst(TTreeNode Parent, const AnsiString S, void * Ptr);
```

Description

The node is added as the first child of the node specified by the Node parameter. Nodes that appear after the added node are moved down one row and reindexed with valid Index values. The S parameter specifies the Text property of the new node. The Ptr parameter specifies the Data property value of the new node. AddChildObjectFirst returns the node that has been added.

Note: The memory referenced by Ptr is not freed when the tree nodes object is freed.

C++ Examples:

```
/*
The following example adds a new item to the selected item's
list of child items in the tree view control. The new item
is added first in the list of child items. The new item is
identified by the text 'New Item'. The TBitmap object is
attached to the new item.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *MyBitMap =
    new Graphics::TBitmap();
  MyBitMap->LoadFromFile("..\\littleB.bmp");
  TreeView1->Items->BeginUpdate(); // keep the node from painting until it's built
  TreeView1->Items->AddChildObjectFirst(
    TreeView1->Selected, "New Item", MyBitMap);
  TreeView1->Items->EndUpdate();
}
```

Delphi Examples:

```
{
The following example adds a new item to the selected item's
list of child items in the tree view control. The new item
is added first in the list of child items. The new item is
identified by the text 'New Item'. The TBitmap object is
attached to the new item.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  MyBitMap : TBitmap;
  node : TTreeNode;
begin
  MyBitMap := TBitmap.Create;
  MyBitMap.LoadFromFile('littleB.bmp');
  TreeView1.Items.BeginUpdate;
  node:= TreeView1.Items.AddChildObjectFirst(
    TreeView1.Selected, 'New Item', MyBitMap);
//  node.ImageIndex:= -1;
  TreeView1.Items.EndUpdate;
end;
```

See Also

AddChildObject (see page 5273)

AddObject (see page 5276)

Data (see page 5254)

27.1.58.4.6 TTreeNodes.AddFirst Method

Adds a new tree node to a tree view.

Pascal

```
function AddFirst(Sibling: TTreeNode; const S: string): TTreeNode;
```

C++

```
__fastcall TTreeNode AddFirst(TTreeNode Sibling, const AnsiString S);
```

Description

The node is added as the first sibling of the node specified by the Node parameter. Nodes that appear after the added node are moved down one row and re-indexed with valid Index values. The S parameter specifies the Text property of the new node. AddFirst returns the node that has been added.

See Also

AddChildFirst (see page 5273)

Add (see page 5270)

27.1.58.4.7 TTreeNodes.AddItem Method

Pascal

```
function AddItem(Parent: HTreeItem; Target: HTreeItem; const Item: TTVItem; AddMode:
TAddMode): HTreeItem;
```

C++

```
__fastcall HTreeItem AddItem(HTreeItem Parent, HTreeItem Target, const TTVItem Item, TAddMode
AddMode);
```

Description

This is AddItem, a member of class TTreeNodes.

27.1.58.4.8 TTreeNodes.AddNode Method

Add (see page 5270) a node to a tree view.

Pascal

```
function AddNode(Node: TTreeNode; Relative: TTreeNode; const S: string; Ptr: Pointer; Method:
TNodeAttachMode): TTreeNode;
```

C++

```
__fastcall TTreeNode AddNode(TTreeNode Node, TTreeNode Relative, const AnsiString S, void *
Ptr, TNodeAttachMode Method);
```

Description

AddNode adds a node object to the collection. AddNode returns the newly-added node, and takes the following parameters.

Node is the node object to be added. If Node is nil, then the tree view's CreateNode method is called to provide one.

The Relative and Method parameters together define the location of the new node. Relative is an existing node that will be either parent or sibling to the new node. Method specifies the precise relationship between the new node and relative, as given by a TNodeAttachMode (see page 5457) constant.

The S parameter specifies the Text property of the new node.

The Ptr parameter specifies the Data property value of the new node.

Note: The memory referenced by Ptr is not freed when the tree nodes object is freed.

See Also

AddChildObject (see page 5273)

Data (see page 5254)

27.1.58.4.9 TTreeNodes.AddObject Method

Adds a new node containing data to a tree view.

Pascal

```
function AddObject(Sibling: TTreeNode; const S: string; Ptr: Pointer): TTreeNode;
```

C++

```
__fastcall TTreeNode AddObject(TTreeNode Sibling, const AnsiString S, void * Ptr);
```

Description

The node is added as the last sibling of the node specified by the Node parameter. The S parameter specifies the Text property of the new node. The Ptr parameter specifies the Data property value of the new node. AddObject returns the node that has been added.

Note: The memory referenced by Ptr is not freed when the tree nodes object is freed.

C++ Examples:

```
/*
This example requires two buttons, three TEdits, a label and
```

```
a populated TreeView.  The following code defines a record
type of TMyRec and a record pointer type of PMyRec.
*/
typedef struct MyRec
{
  AnsiString FName, LName;
} TMyRec;

typedef TMyRec* PMyRec;
/*
Assuming these types are used, the following code adds a
node to TreeView1 as the last sibling of the node specified
by the Index.  A TMyRec record is associated with the added
item. The FName and LName fields are obtained from edit
boxes Edit1 and Edit2. The Index parameter is obtained from
edit box Edit3.  The item is added only if the Index is a
valid value.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PMyRec  MyRecPtr;
  int TreeViewIndex;
  TTreeNodes* items;

  MyRecPtr = new TMyRec;
  MyRecPtr->FName = Edit1->Text;
  MyRecPtr->LName = Edit2->Text;
  TreeViewIndex = StrToInt(Edit3->Text);
  items = TreeView1->Items;
  if (items->Count == 0)
    items->AddObject(NULL, "Item" + IntToStr(TreeViewIndex), MyRecPtr);
  else if ((TreeViewIndex < items->Count) && (TreeViewIndex >= 0))
    items->AddObject(items->Item[TreeViewIndex], "Item" + IntToStr(TreeViewIndex) + "after" +
items->Item[TreeViewIndex]->Text, MyRecPtr);
}
/*
After an item containing a TMyRec record has been added, the
following code retrieves the FName and LName values
associated with the item and displays the values in a label.
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  if (TreeView1->Selected->Data != NULL)
    Label1->Caption = PMyRec(TreeView1->Selected->Data)->FName + " " +
                  PMyRec(TreeView1->Selected->Data)->LName;
}
```

Delphi Examples:

```
{
This example requires three buttons, four TEdits, two labels
and a populated TreeView.  The following code defines a
record type of TMyRec and a record pointer type of PMyRec.
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Edit1: TEdit;
    TreeView1: TTreeView;
    Label2: TLabel;
    Label3: TLabel;
    Button3: TButton;
    Edit4: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    procedure Button1Click(Sender: TObject);
```

```
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  type
  PMyRec = ^TMyRec;
  TMyRec = record
    FName: string;
    LName: string;
  end;


var
  Form1: TForm1;

implementation

{$R *.dfm}

{
Assuming these types are used, the following code adds a
node to TreeView1 as the last sibling of the selected node.
A TMyRec record is associated with the added item. The FName
and LName fields are obtained from edit boxes Edit1 and
Edit2.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  MyRecPtr: PMyRec;
  TreeViewIndex: LongInt;
begin
  New(MyRecPtr);
  MyRecPtr^.FName := Edit1.Text;
  MyRecPtr^.LName := Edit2.Text;
  with TreeView1 do
  begin
    TreeViewIndex := Selected.AbsoluteIndex;
    if Items.Count = 0 then
      Items.AddObject(nil, 'Item' + IntToStr(TreeViewIndex), MyRecPtr)
    else if (TreeViewIndex < Items.Count) and (TreeViewIndex >= 0) then
      Items.AddObject(Items[TreeViewIndex], 'Item' + IntToStr(TreeViewIndex), MyRecPtr);
  end;
end;
{
After an item containing a TMyRec record has been added, the
following code retrieves the FName and LName values
associated with the item and displays the values in a label.
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  if (TreeView1.Selected.Data <> nil) then { query only works on new nodes }
    Edit3.Text := PMyRec(TreeView1.Selected.Data)^.FName + ' ' +
                  PMyRec(TreeView1.Selected.Data)^.LName;
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
  Edit4.Text := IntToStr(TreeView1.Selected.AbsoluteIndex);
end;
```

See Also

AddChildObject (see page 5273)

Data (see page 5254)

27.1.58.4.10 TTreeNodes.AddObjectFirst Method

Adds a new node containing data to a tree view.

Pascal

```
function AddObjectFirst(Sibling: TTreeNode; const S: string; Ptr: Pointer): TTreeNode;
```

C++

```
__fastcall TTreeNode AddObjectFirst(TTreeNode Sibling, const AnsiString S, void * Ptr);
```

Description

The node is added first to the list to which the node specified by the Node parameter belongs. nodes that appear after the added node are moved down one row and reindexed with valid Index values. The S parameter specifies the Text property of the new node. The Ptr parameter specifies the Data property value of the new node. AddObjectFirst returns the node that has been added.

See Also

AddObject (see page 5276)

AddChildObject (see page 5273)

Data (see page 5254)

27.1.58.4.11 TTreeNodes.AlphaSort Method

Sort tree nodes.

Pascal

```
function AlphaSort(ARecurse: Boolean = False): Boolean;
```

C++

```
__fastcall Boolean AlphaSort(Boolean ARecurse = False);
```

Description

AlphaSort triggers node sorting or resorting. If an OnCompare event handler is defined for the associated TCustomTreeView (see page 3957), that routine determines sort order. If no OnCompare handler is defined, nodes are sorted by a simple case-sensitive compare of their captions.

The optional ARecurse parameter (default false) specifies that sorting should recursively descend the node tree and sort each subtree in turn.

Calling AlphaSort is equivalent to calling CustomSort (see page 5281) with a nil (Delphi) or NULL (C++) procedure parameter and a data parameter of 0.

See Also

AlphaSort (see page 3972)

CustomSort (see page 5281)

OnCompare (see page 4019)

27.1.58.4.12 TTreeNodes.Assign Method

Discards any current property information and replaces it with the information from the Source.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use Assign to copy information from one tree nodes object to another. If Source is any other type of object, Assign calls its inherited method, which copies properties from any object that can copy to a TTreeNodes (see page 5266) object in its AssignTo method.

See Also

Assign (see page 2930)

27.1.58.4.13 TTreeNodes.BeginUpdate Method

Prevents the updating of the tree view until the EndUpdate (see page 5283) method is called.

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

BeginUpdate prevents the screen from being repainted when new nodes are added, deleted, or inserted. Tree nodes affected by the changes will have invalid Index values until EndUpdate (see page 5283) is called.

Use BeginUpdate to prevent screen repaints and to speed processing time while adding nodes to the tree view.

Note: Calls to BeginUpdate are cumulative; for every call to BeginUpdate there must be a corresponding call to EndUpdate (see page 5283).

See Also

EndUpdate (see page 5283)

27.1.58.4.14 TTreeNodes.Clear Method

Deletes all tree nodes contained from the list managed by TTreeNodes (see page 5266).

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Use the Clear method to remove all the nodes of a tree view.

See Also

Delete (see page 5282)

27.1.58.4.15 TTreeNodes.CreateItem Method

Pascal

```
function CreateItem(Node: TTreeNode): TTVItem;
```

C++

```
__fastcall TTVItem CreateItem(TTreeNode Node);
```

Description

This is CreateItem, a member of class TTreeNodes.

27.1.58.4.16 TTreeNodes.CustomSort Method

Sort tree nodes.

Pascal

```
function CustomSort(SortProc: TTVCompare; Data: Longint; ARecurse: Boolean = False): Boolean;
```

C++

```
__fastcall Boolean CustomSort(TTVCompare SortProc, Longint Data, Boolean ARecurse = False);
```

Description

CustomSort triggers node sorting or resorting, using a comparison routine indicated by the SortProc parameter. The Data parameter is passed to the comparison routine. The optional ARecurse parameter (default false) specifies that sorting should recursively descend the node tree and sort each subtree in turn.

If SortProc is nil (Delphi) or NULL (C++), a default comparison routine is used. The default routine uses the OnCompare event handler for the associated TCustomTreeView (see page 3957) object, if defined. If the OnCompare event handler is not defined, the default routine uses a simple case-sensitive compare of node captions.

In the comparison routine, the lParam1 and lParam2 parameters refer to two nodes when cast to TTreeNode (see page 5230). The lParamSort parameter is the value previously passed to the Data parameter of CustomSort. The return value of the comparison routine indicates the relative sort order of lParam1 and lParam2:

Return Value	Meaning
< 0	lParam1 comes before lParam2
0	lParam1 and lParam2 are equivalent
> 0	lParam2 comes before lParam1

See Also

CustomSort (see page 3978)

27.1.58.4.17 TTreeNodes.DefineProperties Method

Provides an interface for a method that reads and writes otherwise unpublished data.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

Descendants of TPersistent override DefineProperties to designate a method for storing the object's unpublished data to a stream such as a form file. By default, writing an object to a stream writes the values of all its published properties, and reading the object in reads those values and assigns them to the properties. Objects can also specify methods that read and write data other than published properties by overriding the DefineProperties method.

When overriding DefineProperties, consider including some or all of the following:

A call to the inherited method

Calls to the filer object's DefineProperty method

Calls to the filer object's DefineBinaryProperty method

DefineProperties is virtual, so descendant classes can override it as necessary but are not required to do so.

See Also

DefineProperties (see page 2771)

DefineBinaryProperty (see page 2837)

DefineProperty (see page 2838)

27.1.58.4.18 TTreeNodes.Delete Method

Removes a node from the tree view.

Pascal

```
procedure Delete(Node: TTreeNode);
```

C++

```
__fastcall Delete(TTreeNode Node);
```

Description

Delete removes the tree node specified by the Node parameter and all its children from the tree view.

C++ Examples:

```
/*
The following example deletes an item in the tree view when
the user clicks on it, and if the checkbox is checked. This example
requires a populated TreeView.
*/
void __fastcall TForm1::TVMouseDown(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  THitTests HT;
  if (CheckBox1->Checked && Sender->ClassNameIs("TTreeView"))
  {
    TTreeView *pTV = (TTreeView *)Sender;
    HT = pTV->GetHitTestInfoAt(X,Y);
    if (HT.Contains(htOnItem))
      pTV->Items->Delete(pTV->GetNodeAt(X,Y));
  }
}
```

Delphi Examples:

```
{
The following example deletes an item in the tree view when
the user clicks on it, and if the checkbox is checked. This example
requires a populated TreeView.
```

```
}
procedure TForm1.TVMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  HT : THitTests;
begin
if (CheckBox1.Checked) and (Sender is TTreeView) then
  begin
  with Sender as TTreeView do
    begin
    HT := GetHitTestInfoAt(X,Y);
    if (htOnItem in HT) then
      Items.Delete(GetNodeAt(X,Y));
    end;
  end;
end;
```

See Also

OnDeletion (see page 4024)

DeleteChildren (see page 5238)

27.1.58.4.19 TTreeNodes.EndUpdate Method

Re-enables screen repainting and node reindexing that was turned off with the BeginUpdate (see page 5280) method.

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

Use the EndUpdate method to enable screen updating after BeginUpdate (see page 5280) has been called. Calls to BeginUpdate (see page 5280) are cumulative, so calling EndUpdate will only update the tree view if every other call to BeginUpdate (see page 5280) has already been matched by a call to EndUpdate.

See Also

BeginUpdate (see page 5280)

27.1.58.4.20 TTreeNodes.GetCount Method

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

This is GetCount, a member of class TTreeNodes.

27.1.58.4.21 TTreeNodes.GetEnumerator Method

Pascal

```
function GetEnumerator: TTreeNodesEnumerator;
```

C++

```
__fastcall TTreeNodesEnumerator GetEnumerator();
```

Description

This is GetEnumerator, a member of class TTreeNodes.

27.1.58.4.22 TTreeNodes.GetFirstNode Method

Returns the first tree node in the tree view.

Pascal

```
function GetFirstNode: TTreeNode;
```

C++

```
__fastcall TTreeNode GetFirstNode();
```

Description

Use the GetFirstNode method to retrieve the first node in the tree view. GetFirstNode returns the value of Item (see page 5289)[0].

C++ Examples:

```
/*
The following example adds the Text for all items in the
tree view to the list box.  This example requires a ListBox
and a populated TreeView.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TTreeNode *CurItem = TreeView1->Items->GetFirstNode();
  while (CurItem)
  {
    ListBox1->Items->Add(CurItem->Text);
    CurItem = CurItem->GetNext();
  }
```

Delphi Examples:

```
{
The following example adds the Text for all items in the
tree view to the list box.  This example requires a ListBox
and a populated TreeView.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  CurItem: TTreeNode;
begin
  CurItem := TreeView1.Items.GetFirstNode;
  while CurItem <> nil do
  begin
    ListBox1.Items.Add(CurItem.Text);
    CurItem := CurItem.GetNext;
  end;
end;
```

See Also

getFirstChild (see page 5242)

GetNode (see page 5285)

Item (see page 5289)

27.1.58.4.23 TTreeNodes.GetNode Method

Returns the tree node given the ItemId of the tree node.

Pascal

```
function GetNode(ItemId: HTreeItem): TTreeNode;
```

C++

```
__fastcall TTreeNode GetNode(HTreeItem ItemId);
```

Description

ItemId is a handle to the node in the tree view.

See Also

GetFirstNode (see page 5284)

ItemId (see page 5260)

27.1.58.4.24 TTreeNodes.Insert Method

Inserts a tree node into the tree view before the node specified by the Node parameter.

Pascal

```
function Insert(Sibling: TTreeNode; const S: string): TTreeNode;
```

C++

```
__fastcall TTreeNode Insert(TTreeNode Sibling, const AnsiString S);
```

Description

Call Insert to add a new sibling to the Node parameter, immediately preceding Node. The S parameter specifies the Text property of the new node. Insert returns the new node.

See Also

Add (see page 5270)

InsertNode (see page 5285)

27.1.58.4.25 TTreeNodes.InsertNode Method

Inserts a tree node with data.

Pascal

```
function InsertNode(Node: TTreeNode; Sibling: TTreeNode; const S: string; Ptr: Pointer):
TTreeNode;
```

C++

```
__fastcall TTreeNode InsertNode(TTreeNode Node, TTreeNode Sibling, const AnsiString S, void *
Ptr);
```

Description

Call InsertNode to add a new sibling to the Node parameter, immediately preceding Node. The S parameter specifies the Text property of the new node. The Ptr parameter specifies data associated with the node. InsertNode returns the new node.

Note: The memory referenced by Ptr is not freed when the tree nodes object is freed.

See Also

Insert (see page 5285)

27.1.58.4.26 TTreeNodes.InsertObject Method

Inserts a tree node containing data into the tree view before the node specified by the Node parameter.

Pascal

```
function InsertObject(Sibling: TTreeNode; const S: string; Ptr: Pointer): TTreeNode;
```

C++

```
__fastcall TTreeNode InsertObject(TTreeNode Sibling, const AnsiString S, void * Ptr);
```

Description

Call InsertObject to add a new sibling to the Node parameter, immediately preceding Node. The S parameter specifies the Text property of the new node and the Ptr parameter specifies the Data property of the new node. InsertObject returns the new node.

Note: The memory referenced by Ptr is not freed when the tree nodes object is freed.

See Also

AddObject (see page 5276)

27.1.58.4.27 TTreeNodes.SetItem Method

Pascal

```
procedure SetItem(Index: Integer; Value: TTreeNode);
```

C++

```
__fastcall SetItem(int Index, TTreeNode Value);
```

Description

This is SetItem, a member of class TTreeNodes.

27.1.58.4.28 TTreeNodes.SetUpdateState Method

Pascal

```
procedure SetUpdateState(Updating: Boolean);
```

C++

```
__fastcall SetUpdateState(Boolean Updating);
```

Description

This is SetUpdateState, a member of class TTreeNodes.

27.1.58.5 TTreeNodes Properties

The properties of the TTreeNodes class are listed here.

Protected Properties

	Name	Description
R	Reading (see page 5290)	This is Reading, a member of class TTreeNodes.

Public Properties

	Name	Description
R	Count (see page 5287)	Indicates the number of nodes maintained by the TTreeNodes (see page 5266) object.
R	Handle (see page 5289)	Contains the window handle of the tree view control that owns the tree nodes object.
R	Item (see page 5289)	Lists all tree nodes managed by the TTreeNodes (see page 5266) object.
R	Owner (see page 5290)	Specifies the tree view control that uses the tree nodes object to implement its Items property.

27.1.58.5.1 TTreeNodes.Count Property

Indicates the number of nodes maintained by the TTreeNodes (see page 5266) object.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use Count to determine the number of tree nodes in the tree view that owns the tree nodes object. Count provides an upper bound when iterating through the entries in the Item (see page 5289) property array.

C++ Examples:

```
/*
This example requires two buttons, three TEdits, a label and
a populated TreeView.  The following code defines a record
type of TMyRec and a record pointer type of PMyRec.
*/
typedef struct MyRec
{
  AnsiString FName, LName;
} TMyRec;

typedef TMyRec* PMyRec;
/*
Assuming these types are used, the following code adds a
node to TreeView1 as the last sibling of the node specified
by the Index.  A TMyRec record is associated with the added
item. The FName and LName fields are obtained from edit
boxes Edit1 and Edit2. The Index parameter is obtained from
edit box Edit3.  The item is added only if the Index is a
valid value.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PMyRec  MyRecPtr;
  int TreeViewIndex;
  TTreeNodes* items;

  MyRecPtr = new TMyRec;
  MyRecPtr->FName = Edit1->Text;
  MyRecPtr->LName = Edit2->Text;
  TreeViewIndex = StrToInt(Edit3->Text);
  items = TreeView1->Items;
  if (items->Count == 0)
    items->AddObject(NULL, "Item" + IntToStr(TreeViewIndex), MyRecPtr);
  else if ((TreeViewIndex < items->Count) && (TreeViewIndex >= 0))
    items->AddObject(items->Item[TreeViewIndex], "Item" + IntToStr(TreeViewIndex) + "after" +
items->Item[TreeViewIndex]->Text, MyRecPtr);
```

```
}
/*
After an item containing a TMyRec record has been added, the
following code retrieves the FName and LName values
associated with the item and displays the values in a label.
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  if (TreeView1->Selected->Data != NULL)
    Label1->Caption = PMyRec(TreeView1->Selected->Data)->FName + " " +
                  PMyRec(TreeView1->Selected->Data)->LName;
}
```

Delphi Examples:

```
{
This example requires three buttons, four TEdits, two labels
and a populated TreeView.  The following code defines a
record type of TMyRec and a record pointer type of PMyRec.
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Edit1: TEdit;
    TreeView1: TTreeView;
    Label2: TLabel;
    Label3: TLabel;
    Button3: TButton;
    Edit4: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  type
  PMyRec = ^TMyRec;
  TMyRec = record
    FName: string;
    LName: string;
  end;


var
  Form1: TForm1;

implementation

{$R *.dfm}

{
Assuming these types are used, the following code adds a
node to TreeView1 as the last sibling of the selected node.
A TMyRec record is associated with the added item. The FName
and LName fields are obtained from edit boxes Edit1 and
Edit2.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  MyRecPtr: PMyRec;
  TreeViewIndex: LongInt;
begin
```

```
    New(MyRecPtr);
    MyRecPtr^.FName := Edit1.Text;
    MyRecPtr^.LName := Edit2.Text;
    with TreeView1 do
    begin
      TreeViewIndex := Selected.AbsoluteIndex;
      if Items.Count = 0 then
        Items.AddObject(nil, 'Item' + IntToStr(TreeViewIndex), MyRecPtr)
      else if (TreeViewIndex < Items.Count) and (TreeViewIndex >= 0) then
        Items.AddObject(Items[TreeViewIndex], 'Item' + IntToStr(TreeViewIndex), MyRecPtr);
    end;
  end;
  {
  After an item containing a TMyRec record has been added, the
  following code retrieves the FName and LName values
  associated with the item and displays the values in a label.
  }
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    if (TreeView1.Selected.Data <> nil) then { query only works on new nodes }
      Edit3.Text := PMyRec(TreeView1.Selected.Data)^.FName + ' ' +
                    PMyRec(TreeView1.Selected.Data)^.LName;
  end;
  procedure TForm1.Button3Click(Sender: TObject);
  begin
    Edit4.Text := IntToStr(TreeView1.Selected.AbsoluteIndex);
  end;
```

See Also

Owner (see page 5290)

27.1.58.5.2 TTreeNodes.Handle Property

Contains the window handle of the tree view control that owns the tree nodes object.

Pascal

```
property Handle: HWND;
```

C++

```
__property HWND Handle;
```

Description

Use Handle to obtain the handle of the tree view that owns the tree nodes object.

See Also

Owner (see page 5290)

Handle (see page 6207)

27.1.58.5.3 TTreeNodes.Item Property

Lists all tree nodes managed by the TTreeNodes (see page 5266) object.

Pascal

```
property Item [Index: Integer]: TTreeNode;
```

C++

```
__property TTreeNode Item[int Index];
```

Description

Use Item to access to a node by its position in the tree view. The first node has an index of 0, the second an index of 1, and so on.

Note: Item is the default property for TTreeNodes (see page 5266). This means that the name of the Item property can be omitted when indexing into the set of tree nodes. Thus, the line

```
FirstNode := TreeView1.Items.Item[0];
```

Note: can be written

```
FirstNode := TreeView1.Items[0];
```

See Also

AbsoluteIndex (see page 5252)

operator_[]

27.1.58.5.4 TTreeNodes.Owner Property

Specifies the tree view control that uses the tree nodes object to implement its Items property.

Pascal

```
property Owner: TCustomTreeView;
```

C++

```
__property TCustomTreeView Owner;
```

Description

Use the Owner property to access the tree view control that displays the nodes maintained by the TTreeNodes (see page 5266) object.

See Also

Handle (see page 5289)

27.1.58.5.5 TTreeNodes.Reading Property

Pascal

```
property Reading: Boolean;
```

C++

```
__property Boolean Reading;
```

Description

This is Reading, a member of class TTreeNodes.

27.1.59 TTreeNodesEnumerator Class

Class Hierarchy

ComCtrls.TTreeNodesEnumerator

Pascal

```
TTreeNodesEnumerator = class;
```

C++

```
class TTreeNodesEnumerator;
```

File

ComCtrls

Description

This is class ComCtrls.TTreeNodesEnumerator.

27.1.59.1 TTreeNodesEnumerator Members

The following tables list the members exposed by TTreeNodesEnumerator.

Public Methods

	Name	Description
	Create (see page 5291)	This is Create, a member of class TTreeNodesEnumerator.
	GetCurrent (see page 5292)	This is GetCurrent, a member of class TTreeNodesEnumerator.
	MoveNext (see page 5292)	This is MoveNext, a member of class TTreeNodesEnumerator.

Public Properties

	Name	Description
R	Current (see page 5292)	This is Current, a member of class TTreeNodesEnumerator.

27.1.59.2 TTreeNodesEnumerator.Create Constructor

Pascal

```
constructor Create(ATreeNodes: TTreeNodes);
```

C++

```
__fastcall TTreeNodesEnumerator(TTreeNodes ATreeNodes);
```

Description

This is Create, a member of class TTreeNodesEnumerator.

27.1.59.3 TTreeNodesEnumerator Methods

The methods of the TTreeNodesEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 5292)	This is GetCurrent, a member of class TTreeNodesEnumerator.
	MoveNext (see page 5292)	This is MoveNext, a member of class TTreeNodesEnumerator.

27.1.59.3.1 TTreeNodesEnumerator.GetCurrent Method

Pascal

```
function GetCurrent: TTreeNode;
```

C++

```
__fastcall TTreeNode GetCurrent();
```

Description

This is GetCurrent, a member of class TTreeNodesEnumerator.

27.1.59.3.2 TTreeNodesEnumerator.MoveNext Method

Pascal

```
function MoveNext: Boolean;
```

C++

```
__fastcall Boolean MoveNext();
```

Description

This is MoveNext, a member of class TTreeNodesEnumerator.

27.1.59.4 TTreeNodesEnumerator Properties

The properties of the TTreeNodesEnumerator class are listed here.

Public Properties

	Name	Description
R	Current (see page 5292)	This is Current, a member of class TTreeNodesEnumerator.

27.1.59.4.1 TTreeNodesEnumerator.Current Property

Pascal

```
property Current: TTreeNode;
```

C++

```
__property TTreeNode Current;
```

Description

This is Current, a member of class TTreeNodesEnumerator.

27.1.60 TTreeView Class

TTreeView represents a window that displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files and directories on a disk.

Class Hierarchy

Pascal

```
TTreeView = class(TCustomTreeView);
```

C++

```
class TTreeView : public TCustomTreeView;
```

File

ComCtrls

Description

Use TTreeView to add an expanding and contracting outline to a form. Each node in a tree view control consists of a label and a number of optional bitmapped images. Each node can have a list of subnodes associated with it. By clicking on a node, the user can expand or collapse the associated list of subnodes.

See Also

TTreeNode (see page 5230)

TTreeNodes (see page 5266)

List Controls

Tree Views

27.1.60.1 TTreeView Members

The following tables list the members exposed by TTreeView.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCancelEdit (see page 4017)	Occurs when user cancels editing of a node's Caption.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.

	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FChangeTimer (see page 3970)	This is FChangeTimer, a member of class TCustomTreeView.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	Added (see page 3972)	This is Added, a member of class TCustomTreeView.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanChange (see page 3972)	Pending node change event dispatcher.
	CanCollapse (see page 3973)	Pending node collapse event dispatcher.
	CanEdit (see page 3973)	Node editing event dispatcher.
	CanExpand (see page 3974)	Node expand event dispatcher.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 3974)	Completed node change event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.

	Click (see page 5743)	Respond to user click.
	Collapse (see page 3975)	Completed node collapse event dispatcher.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateNode (see page 3975)	Creates a new TTreeNode (see page 5230) instance with the TTreeNodes (see page 5266) specified by Items (see page 4004) as its Owner.
V	CreateNodes (see page 3976)	This is CreateNodes, a member of class TCustomTreeView.
V	CreateParams (see page 3976)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 3976)	Creates the window used by the tree view component.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
V	CustomDraw (see page 3977)	Generates an OnCustomDraw (see page 4019) or OnAdvancedCustomDraw (see page 4016) event.
V	CustomDrawItem (see page 3977)	Generates an OnCustomDrawItem (see page 4021) or OnAdvancedCustomDrawItem (see page 4016) event.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	Delete (see page 3979)	Generates an OnDeletion (see page 4024) event.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 3980)	Destroys the control's window.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
V	DoEndDrag (see page 3981)	End drag event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
V	DoEnter (see page 3981)	Respond to receiving input focus.
	DoEnter (see page 6144)	Respond to receiving input focus.
V	DoExit (see page 3981)	Respond to losing input focus
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.

	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
V	DoStartDrag (see page 3982)	Drag start event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	Edit (see page 3982)	Edited event dispatcher.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	Expand (see page 3983)	Completed node expansion event dispatcher.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 3984)	Returns the image list to be displayed during drag operations.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetImageIndex (see page 3986)	Obtain image index.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
V	GetSelectedIndex (see page 3988)	Obtain selection index.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	InvalidateSelectionsRects (see page 3989)	This is InvalidateSelectionsRects, a member of class TCustomTreeView.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
V	IsCustomDrawn (see page 3989)	Specifies when the control should generate custom draw events.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 3990)	Initializes the tree view after it is loaded from a stream.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 3991)	OnMouseDown event dispatcher.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 3992)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.

	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 3994)	Sets the drag mode for the tree view.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateSelection (see page 3995)	This is ValidateSelection, a member of class TCustomTreeView.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 3995)	Provides specific message responses for the control.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	CreateWndRestores (see page 4000)	This is CreateWndRestores, a member of class TCustomTreeView.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AlphaSort (see page 3972)	Sort tree nodes.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearSelection (see page 3974)	Deselect (see page 3980) all selected nodes, optionally excluding primary selection.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 3969)	Creates an instance of a TTreeView (see page 5293) control.
	Create (see page 5731)	Creates an instance of TControl (see page 5724).

	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CustomSort (see page 3978)	Sorts the nodes in the tree view into a customized sort order.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Deselect (see page 3980)	Deselects a specified node.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 3970)	Destroys the tree view control and is called by the Delete (see page 3979) method.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindNextToSelect (see page 3983)	Returns next selectable node.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	FullCollapse (see page 3983)	Collapses all the nodes within a tree view control.
	FullExpand (see page 3984)	Expands all nodes within the tree view control.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHitTestInfoAt (see page 3985)	Returns information about the location of a point relative to the client area of the tree view control.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNodeAt (see page 3987)	Returns the node that is found at the specified position.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetSelections (see page 3989)	Obtain selections list.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsEditing (see page 3990)	Indicates whether a node is currently being edited by the user.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	LoadFromFile (see page 3990)	Reads the file specified in FileName and loads the data into the tree view.
	LoadFromStream (see page 3991)	Reads tree view data from a stream and stores the contents in the tree view.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

	SaveToFile (see page 3992)	Saves a the tree view to the file specified in FileName.
	SaveToStream (see page 3993)	Writes the data in the tree view to the stream passed as the Stream parameter.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
V	Select (see page 3993)	This is the overview for the Select method overload.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Subselect (see page 3994)	Toggle node selection.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 3997)	Provides access to the canvas.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.

	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropTarget (see page 4000)	Specifies which item in the tree view appears as the target of a drag and drop operation.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	Selected (see page 4007)	Specifies the selected node in the tree view.
R	SelectionCount (see page 4012)	Returns number of nodes selected.
R	Selections (see page 4012)	Returns nodes selected.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TopItem (see page 4015)	Specifies the topmost node that appears in the tree view.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAddition (see page 5342)	Occurs when new node is added.
	OnAdvancedCustomDraw (see page 5342)	Occurs at discrete stages during the painting of the tree view control.
	OnAdvancedCustomDrawItem (see page 5342)	Occurs at discrete stages during the painting of tree view nodes.
	OnChange (see page 5343)	Occurs whenever the selection has changed from one node to another.
	OnChanging (see page 5343)	Occurs when the selection is about to change from one node to another.
	OnClick (see page 5344)	Occurs when the user clicks the control.
	OnCollapsed (see page 5345)	Occurs after a node has been collapsed.
	OnCollapsing (see page 5346)	Occurs when a node is about to be collapsed.
	OnCompare (see page 5346)	Occurs when two nodes must be compared during a sort of the nodes in the tree view.
	OnContextPopup (see page 5347)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreateNodeClass (see page 5347)	Occurs when a node in the treeview is about to be created.
	OnCustomDraw (see page 5348)	Occurs immediately prior to painting the tree view control.
	OnCustomDrawItem (see page 5349)	Occurs immediately prior to painting a node in a tree view control.
	OnDblClick (see page 5352)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeletion (see page 5352)	Occurs when a node in the tree view is deleted.

	OnDragDrop (see page 5352)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5355)	Occurs when the user drags an object over a control.
	OnEdited (see page 5357)	Occurs after the user edits the Text property of a node.
	OnEditing (see page 5357)	Occurs when the user starts to edit the Text property of a node.
	OnEndDock (see page 5358)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5358)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 5359)	Occurs when a control receives the input focus.
	OnExit (see page 5362)	Occurs when the input focus shifts away from one control to another.
	OnExpanded (see page 5363)	Occurs after a node is expanded.
	OnExpanding (see page 5364)	Occurs when a node is about to be expanded.
	OnGetImageIndex (see page 5364)	Occurs when the tree view looks up the ImageIndex of a node.
	OnGetSelectedIndex (see page 5365)	Occurs when the tree view looks up the SelectedIndex of a node.
	OnKeyDown (see page 5365)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 5368)	Occurs when a key is pressed.
	OnKeyUp (see page 5369)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5371)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5372)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5374)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5374)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5375)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5377)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 5379)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 5380)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 5307)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 5310)	Specifies how the control is anchored to its parent.
	AutoExpand (see page 5311)	Specifies whether the nodes in the tree view automatically expand and collapse depending on the selection.
	BevelEdges (see page 5311)	Specifies which edges of the control are beveled.
	BevelInner (see page 5312)	Specifies the cut of the inner bevel.
	BevelKind (see page 5312)	Specifies the control's bevel style.
	BevelOuter (see page 5312)	Specifies the cut of the outer bevel.
	BevelWidth (see page 5313)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 5313)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 5314)	Determines whether the tree view control has a border.
	BorderWidth (see page 5314)	Specifies the width of the control's border.
	ChangeDelay (see page 5315)	Specifies the delay between when a node is selected and when the OnChange event occurs.
	Color (see page 5315)	Specifies the background color of the control.
	Constraints (see page 5316)	Specifies the size constraints for the control.
	Ctl3D (see page 5316)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 5317)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5317)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5318)	Determines how the control initiates drag-and-drop or drag-and-dock operations.

	Enabled (see page 5319)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 5320)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	HideSelection (see page 5321)	Determines whether a selected node appears selected when the focus shifts to another control.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HotTrack (see page 5321)	Specifies whether list items are highlighted when the mouse passes over them.
	Images (see page 5321)	Determines which image list is associated with the tree view.
	Indent (see page 5322)	Specifies the amount of indentation in pixels when a list of child nodes is expanded.
	Items (see page 5324)	Lists the individual nodes that appear in the tree view control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	MultiSelect (see page 5325)	Determines whether the user can select more than one tree node at a time.
	MultiSelectStyle (see page 5326)	Determines how multiple node selections work.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 5326)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 5327)	Determines where a control looks for its color information.
	ParentCtl3D (see page 5327)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5328)	Determines where a control looks for its font information.
	ParentShowHint (see page 5328)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 5329)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 5333)	Determines whether the user can edit the node labels.
	RightClickSelect (see page 5333)	Determines whether the Selected property returns nodes that are selected using the right mouse button.
	RowSelect (see page 5334)	Specifies whether the entire row of the selected item is highlighted.
	ShowButtons (see page 5334)	Specifies whether to display plus (+) and minus (-) buttons to the left side of each parent item.
	ShowHint (see page 5335)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	ShowLines (see page 5336)	Specifies whether to display the lines that link child nodes to their corresponding parent nodes.
	ShowRoot (see page 5336)	Specifies whether lines connecting top-level nodes are displayed.
	SortType (see page 5337)	Determines if and how the nodes in a tree view are automatically sorted.
	StateImages (see page 5337)	Determines which image list to use for state images.
	TabOrder (see page 5338)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 5338)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	ToolTips (see page 5339)	Specifies whether the items in the tree view have tool tips.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 5339)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

27.1.60.2 TTreeView Properties

The properties of the TTreeView class are listed here.

Published Properties

	Name	Description
	Align (see page 5307)	Determines how the control aligns within its container (parent control).
	Anchors (see page 5310)	Specifies how the control is anchored to its parent.
	AutoExpand (see page 5311)	Specifies whether the nodes in the tree view automatically expand and collapse depending on the selection.
	BevelEdges (see page 5311)	Specifies which edges of the control are beveled.
	BevelInner (see page 5312)	Specifies the cut of the inner bevel.
	BevelKind (see page 5312)	Specifies the control's bevel style.
	BevelOuter (see page 5312)	Specifies the cut of the outer bevel.
	BevelWidth (see page 5313)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 5313)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 5314)	Determines whether the tree view control has a border.
	BorderWidth (see page 5314)	Specifies the width of the control's border.
	ChangeDelay (see page 5315)	Specifies the delay between when a node is selected and when the OnChange event occurs.
	Color (see page 5315)	Specifies the background color of the control.
	Constraints (see page 5316)	Specifies the size constraints for the control.
	Ctl3D (see page 5316)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 5317)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5317)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5318)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 5319)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 5320)	Controls the attributes of text written on or in the control.
	HideSelection (see page 5321)	Determines whether a selected node appears selected when the focus shifts to another control.
	HotTrack (see page 5321)	Specifies whether list items are highlighted when the mouse passes over them.
	Images (see page 5321)	Determines which image list is associated with the tree view.
	Indent (see page 5322)	Specifies the amount of indentation in pixels when a list of child nodes is expanded.
	Items (see page 5324)	Lists the individual nodes that appear in the tree view control.
	MultiSelect (see page 5325)	Determines whether the user can select more than one tree node at a time.
	MultiSelectStyle (see page 5326)	Determines how multiple node selections work.
	ParentBiDiMode (see page 5326)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 5327)	Determines where a control looks for its color information.
	ParentCtl3D (see page 5327)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5328)	Determines where a control looks for its font information.
	ParentShowHint (see page 5328)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 5329)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 5333)	Determines whether the user can edit the node labels.
	RightClickSelect (see page 5333)	Determines whether the Selected property returns nodes that are selected using the right mouse button.
	RowSelect (see page 5334)	Specifies whether the entire row of the selected item is highlighted.
	ShowButtons (see page 5334)	Specifies whether to display plus (+) and minus (-) buttons to the left side of each parent item.
	ShowHint (see page 5335)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	ShowLines (see page 5336)	Specifies whether to display the lines that link child nodes to their corresponding parent nodes.
	ShowRoot (see page 5336)	Specifies whether lines connecting top-level nodes are displayed.
	SortType (see page 5337)	Determines if and how the nodes in a tree view are automatically sorted.
	StateImages (see page 5337)	Determines which image list to use for state images.
	TabOrder (see page 5338)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 5338)	Determines if the user can tab to a control.

	ToolTips (see page 5339)	Specifies whether the items in the tree view have tool tips.
	Visible (see page 5339)	Determines whether the component appears onscreen.

27.1.60.2.1 TTreeView.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;
```

```
  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
```

```
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

27.1.60.2.2 TTreeView.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other

aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.60.2.3 TTreeView.AutoExpand Property

Specifies whether the nodes in the tree view automatically expand and collapse depending on the selection.

Pascal

```
property AutoExpand: Boolean;
```

C++

```
__property Boolean AutoExpand;
```

Description

Set AutoExpand to true to cause the selected item to expand and the unselected items to collapse.

See Also

Selected (see page 4007)

Expanded (see page 5257)

Selected (see page 5264)

Expand (see page 5242)

Collapse (see page 5235)

OnExpanded (see page 4025)

OnExpanding (see page 4025)

OnCollapsed (see page 4018)

OnCollapsing (see page 4018)

27.1.60.2.4 TTreeView.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

27.1.60.2.5 TTreeView.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

27.1.60.2.6 TTreeView.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

27.1.60.2.7 TTreeView.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

27.1.60.2.8 TTreeView.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

27.1.60.2.9 TTreeView.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

27.1.60.2.10 TTreeView.BorderStyle Property

Determines whether the tree view control has a border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Set BorderStyle to specify whether the tree view control should be outlined with a single-line border. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

See Also

BoundsRect (see page 5803)

27.1.60.2.11 TTreeView.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics or text drawn by the control is clipped to the area within the border.

See Also

ClientRect

BevelOuter (see page 6194)

27.1.60.2.12 TTreeView.ChangeDelay Property

Specifies the delay between when a node is selected and when the OnChange event occurs.

Pascal

```
property ChangeDelay: Integer;
```

C++

```
__property int ChangeDelay;
```

Description

Use ChangeDelay to get or set the delay, in milliseconds, between when a node is selected and when the OnChange event occurs.

Set the ChangeDelay to 50 milliseconds to emulate the behavior of the tree-view control used in Windows Explorer.

See Also

Selected (see page 4007)

OnChange (see page 4017)

27.1.60.2.13 TTreeView.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
```

```
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

27.1.60.2.14 TTreeView.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.60.2.15 TTreeView.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

27.1.60.2.16 TTreeView.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

27.1.60.2.17 TTreeView.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

27.1.60.2.18 TTreeView.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
```

```
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

27.1.60.2.19 TTreeView.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.60.2.20 TTreeView.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

27.1.60.2.21 TTreeView.HideSelection Property

Determines whether a selected node appears selected when the focus shifts to another control.

Pascal

```
property HideSelection: Boolean;
```

C++

```
__property Boolean HideSelection;
```

Description

Use HideSelection to specify whether the user is given visual feedback about the current selection in the tree view when it does not have focus. If true, the selected node is not visually distinct from other nodes until focus returns to the control. If false, the node always appears selected.

See Also

Selected (see page 4007)

27.1.60.2.22 TTreeView.HotTrack Property

Specifies whether list items are highlighted when the mouse passes over them.

Pascal

```
property HotTrack: Boolean;
```

C++

```
__property Boolean HotTrack;
```

Description

Set HotTrack to true to provide visual feedback about which item is under the mouse. Set HotTrack to false to suppress the visual feedback about which item is under the mouse.

See Also

HideSelection (see page 4000)

RowSelect (see page 4007)

27.1.60.2.23 TTreeView.Images Property

Determines which image list is associated with the tree view.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Use Images to provide a customized list of bitmaps that can be displayed to the left of a node's label. Individual nodes specify the image from this list that should appear by setting their ImageIndex property.

See Also

OnGetImageIndex (see page 4026)

StateImages (see page 4014)

ImageIndex (see page 5259)

SelectedIndex (see page 5265)

27.1.60.2.24 TTreeView.Indent Property

Specifies the amount of indentation in pixels when a list of child nodes is expanded.

Pascal

```
property Indent: Integer;
```

C++

```
__property int Indent;
```

Description

Use Indent to determine how far child nodes are indented from their parent nodes when the parent is expanded.

C++ Examples:

```
/*
The following example shows how the OnCustomDrawItem event
handler draws items and lines of the tree view after the
OnCustomDraw event handler has filled in the background.
*/
void __fastcall TCustomDrawForm::TVCustomDrawItem(TCustomTreeView *Sender,
      TTreeNode *Node, TCustomDrawState State, bool &DefaultDraw)
{
  TRect NodeRect;
/*
    If DefaultDraw it is true, any of the node's font
    properties can be changed. Note also that when
    DefaultDraw = True, Windows draws the buttons and
    ignores our font background colors, using instead the
    TreeView's Color property.
*/
    if (State.Contains(cdsSelected))
    {
      TV->Canvas->Font->Assign(SelectedFontDialog->Font);
      TV->Canvas->Brush->Color = SelBkgColorDialog->Color;
    };

    DefaultDraw = FDefaultDrawItem;
/*
    DefaultDraw = False means you have to handle all the
    item drawing yourself, including the buttons, lines,
    images, and text.
*/
    if (!DefaultDraw)
    {
      //draw the selection rect.
      if (State.Contains(cdsSelected))
      {
        NodeRect = Node->DisplayRect(True);
        TV->Canvas->FillRect(NodeRect);
      };
      NodeRect = Node->DisplayRect(False);
```

```
        if (None1->Checked)
        //no bitmap, so paint in the background color.
        {
          TV->Canvas->Brush->Color = BkgColorDialog->Color;
          TV->Canvas->Brush->Style = FBrushStyle;
          TV->Canvas->FillRect(NodeRect);
        }
        else
          //don't paint over the background bitmap.
          TV->Canvas->Brush->Style = bsClear;

        NodeRect.Left = NodeRect.Left + (Node->Level * TV->Indent);
        // NodeRect.Left now represents the left-most portion
        // of the expand button
        DrawButton(&NodeRect, Node); // See the CustomDraw demo

        NodeRect.Left = NodeRect.Left + TV->Indent + FButtonSize;
        //NodeRect->Left is now the leftmost portion of the image.
        DrawImage(&NodeRect, Node->ImageIndex); // See the CustomDraw demo

        NodeRect.Left = NodeRect.Left + ImageList->Width;
        //Now we are finally in a position to draw the text.

        TV->Canvas->TextOut(NodeRect.Left, NodeRect.Top, Node->Text);
      };
}
```

Delphi Examples:

```
{
The following example shows how the OnCustomDrawItem event
handler draws items and lines of the tree view after the
OnCustomDraw event handler has filled in the background.
}
procedure TCustomDrawForm.TVCustomDrawItem(Sender: TCustomTreeView; Node: TTreeNode;
  State: TCustomDrawState; var DefaultDraw: Boolean);
var
  NodeRect: TRect;
begin
  with TV.Canvas do
  begin
{
    If DefaultDraw it is true, any of the node's font
    properties can be changed. Note also that when
    DefaultDraw = True, Windows draws the buttons and
    ignores our font background colors, using instead the
    TreeView's Color property.
}
    if cdsSelected in State then
    begin
      Font.Assign(SelectedFontDialog.Font);
      Brush.Color := SelBkgColorDialog.Color;
    end;

    DefaultDraw := False; // FDefaultDrawItem;
{
    DefaultDraw = False means you have to handle all the
    item drawing yourself, including the buttons, lines,
    images, and text.
}
    if not DefaultDraw then
    begin
      //draw the selection rect.
      if cdsSelected in State then
      begin
        NodeRect := Node.DisplayRect(True);
        FillRect(NodeRect);
```

```
        end;
        NodeRect := Node.DisplayRect(False);

        if None1.Checked then
        //no bitmap, so paint in the background color.
        begin
          Brush.Color := BkgColorDialog.Color;
          Brush.Style := FBrushStyle;
          FillRect(NodeRect)
        end
        else
          //don't paint over the background bitmap.
          Brush.Style := bsClear;

        NodeRect.Left := NodeRect.Left + (Node.Level * TV.Indent);
        // NodeRect.Left now represents the left-most portion
        // of the expand button
        DrawButton(NodeRect, Node); // See the CustomDraw demo

        NodeRect.Left := NodeRect.Left + TV.Indent + FButtonSize;
        //NodeRect.Left is now the leftmost portion of the image.
        DrawImage(NodeRect, Node.ImageIndex); // See the CustomDraw demo

        NodeRect.Left := NodeRect.Left + ImageList.Width;
        //Now we are finally in a position to draw the text.

        TextOut(NodeRect.Left, NodeRect.Top, Node.Text);
      end;
    end;
  end;
```

See Also

FullExpand (see page 3984)

27.1.60.2.25 TTreeView.Items Property

Lists the individual nodes that appear in the tree view control.

Pascal

```
property Items: TTreeNodes;
```

C++

```
__property TTreeNodes Items;
```

Description

Individual nodes in a tree view are TTreeNode objects. These individual nodes can be accessed by using the Items property along with the item's index into the tree view. For example, to access the second item in the tree view, you could use the following code.

```
MyTreeNode := TreeView1.Items[1];
MyTreeNode = TreeView1->Items[1];
```

When setting this property at design-time in the Object Inspector the Tree View Items Editor appears. Use the New Item and New SubItem buttons to add items to the tree view. Use the Text property to modify what text is displayed in the label of the item.

At run-time nodes can be added and inserted by using the TTreeNodes methods AddChildFirst, AddChild, AddChildObjectFirst, AddChildObject, AddFirst, Add, AddObjectFirst, AddObject, and Insert.

Note: Accessing tree view items by index can be time-intensive, particularly when the tree view contains many items. For optimal performance, try to design your application so that it has as few dependencies on the tree view's item index as possible.

C++ Examples:

```
/*
The following example deletes an item in the tree view when
the user clicks on it, and if the checkbox is checked. This example
requires a populated TreeView.
*/
void __fastcall TForm1::TVMouseDown(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  THitTests HT;
  if (CheckBox1->Checked && Sender->ClassNameIs("TTreeView"))
  {
    TTreeView *pTV = (TTreeView *)Sender;
    HT = pTV->GetHitTestInfoAt(X,Y);
    if (HT.Contains(htOnItem))
      pTV->Items->Delete(pTV->GetNodeAt(X,Y));
  }
}
```

Delphi Examples:

```
{
The following example deletes an item in the tree view when
the user clicks on it, and if the checkbox is checked. This example
requires a populated TreeView.
}
procedure TForm1.TVMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  HT : THitTests;
begin
if (CheckBox1.Checked) and (Sender is TTreeView) then
  begin
  with Sender as TTreeView do
    begin
    HT := GetHitTestInfoAt(X,Y);
    if (htOnItem in HT) then
      Items.Delete(GetNodeAt(X,Y));
    end;
  end;
end;
```

See Also

TreeView (see page 5266)

AddChildFirst (see page 5273)

AddChild (see page 5271)

AddChildObject (see page 5273)

AddChildObjectFirst (see page 5274)

Add (see page 5270)

AddObject (see page 5276)

AddObjectFirst (see page 5279)

AddFirst (see page 5275)

27.1.60.2.26 TTreeView.MultiSelect Property

Determines whether the user can select more than one tree node at a time.

Pascal

```
property MultiSelect: Boolean;
```

C++

```
__property Boolean MultiSelect;
```

Description

Set MultiSelect to specify whether users can select multiple nodes using the Control and Shift keys. A selection style must also be chosen in MultiSelectStyle.

See Also

MultiSelectStyle (see page 4005)

27.1.60.2.27 TTreeView.MultiSelectStyle Property

Determines how multiple node selections work.

Pascal

```
property MultiSelectStyle: TMultiSelectStyle;
```

C++

```
__property TMultiSelectStyle MultiSelectStyle;
```

Description

MultiSelectStyle determines how multiple selections are made when MultiSelect is true. MultiSelectStyle must include at least one of the following values.

Value	Meaning
msControlSelect	Clicking on any node with the Control key pressed toggles the selection of that node.
msShiftSelect	Clicking on any node with the Shift key pressed selects that node, the last single node selected, and the nodes in between. All other nodes are deselected.
msVisibleOnly	Multiple selections with the Shift key do not include child nodes of collapsed nodes.
msSiblingOnly	Selected nodes are restricted to a single set of siblings.

If msControlSelect or msShiftSelect are in effect, the last singly-selected node becomes the primary selection, referenced by Selections[0]. The primary selection is the anchor for extended selections using the Shift key.

See Also

Selections (see page 4012)

27.1.60.2.28 TTreeView.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's

BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

27.1.60.2.29 TTreeView.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

27.1.60.2.30 TTreeView.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

27.1.60.2.31 TTreeView.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

27.1.60.2.32 TTreeView.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.60.2.33 TTreeView.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
```

```
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
```

```
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
```

```
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.60.2.34 TTreeView.ReadOnly Property

Determines whether the user can edit the node labels.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to specify whether the user can edit the nodes of the tree view. If ReadOnly is true, the user can expand and collapse nodes, but can't edit their labels. If ReadOnly is false, the user can edit the labels as well. The default value is false.

See Also

Text (see page 5265)

27.1.60.2.35 TTreeView.RightClickSelect Property

Determines whether the Selected property returns nodes that are selected using the right mouse button.

Pascal

```
property RightClickSelect: Boolean;
```

C++

```
__property Boolean RightClickSelect;
```

Description

Use RightClickSelect to allow the Selected property to indicate nodes the user clicks with the right mouse button. If RightClickSelect is true, the value of Selected is the value of the node last clicked with either the right or left mouse button. If RightClickSelect is false, the value of Selected is the node last clicked using the left mouse button.

RightClickSelect affects only the value of the Selected property. It does not cause the tree view to highlight a new node if the node is selected using the right mouse button.

Note: RightClickSelect must be set to true before the user right-clicks the tree view for it to affect the value of the Selected property.

See Also

Selected (see page 4007)

OnMouseDown (see page 5862)

27.1.60.2.36 TTreeView.RowSelect Property

Specifies whether the entire row of the selected item is highlighted.

Pascal

```
property RowSelect: Boolean;
```

C++

```
__property Boolean RowSelect;
```

Description

Set RowSelect to true to cause the entire row of the selected item to be highlighted.

RowSelect is ignored if ShowLines is true.

See Also

Selected (see page 4007)

ShowLines (see page 4013)

27.1.60.2.37 TTreeView.ShowButtons Property

Specifies whether to display plus (+) and minus (-) buttons to the left side of each parent item.

Pascal

```
property ShowButtons: Boolean;
```

C++

```
__property Boolean ShowButtons;
```

Description

If ShowButtons is true, a button will appear to the left of each parent item. The user can click the button to expand or collapse the child items as an alternative to double-clicking the parent item.

See Also

ShowLines (see page 4013)

ShowRoot (see page 4013)

27.1.60.2.38 TTreeView.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.60.2.39 TTreeView.ShowLines Property

Specifies whether to display the lines that link child nodes to their corresponding parent nodes.

Pascal

```
property ShowLines: Boolean;
```

C++

```
__property Boolean ShowLines;
```

Description

If ShowLines is true, lines linking child nodes to their parent nodes are displayed. Nodes at the root of the hierarchy are not automatically linked. To link nodes at the root, the ShowRoot property must also be set to true.

See Also

ShowButtons (see page 4013)

ShowRoot (see page 4013)

27.1.60.2.40 TTreeView.ShowRoot Property

Specifies whether lines connecting top-level nodes are displayed.

Pascal

```
property ShowRoot: Boolean;
```

C++

```
__property Boolean ShowRoot;
```

Description

To show lines connecting top-level nodes to a single root, set the tree view's ShowRoot and ShowLines properties to true.

See Also

ShowButtons (see page 4013)

ShowLines (see page 4013)

27.1.60.2.41 TTreeView.SortType Property

Determines if and how the nodes in a tree view are automatically sorted.

Pascal

```
property SortType: TSortType;
```

C++

```
__property TSortType SortType;
```

Description

Once a tree view is sorted, the original hierarchy is lost. That is, setting the SortType back to stNone will not restore the original order of items. These are the possible values:

Value	Meaning
stNone	No sorting is done.
stData	The items are sorted when the Data object or SortType is changed.
stText	The items are sorted when the Caption or SortType is changed.
stBoth	The items are sorted when either the Data object, the Caption or SortType is changed

Optionally, the OnCompare event can be hooked to handle comparisons.

See Also

AlphaSort (see page 3972)

OnCompare (see page 4019)

27.1.60.2.42 TTreeView.StateImages Property

Determines which image list to use for state images.

Pascal

```
property StateImages: TCustomImageList;
```

C++

```
__property TCustomImageList StateImages;
```

Description

Use StateImages to provide a set of bitmaps that reflect the state of tree view nodes. The state image appears as an additional

image to the left of the item's icon.

See Also

Images (see page 4001)

StateIndex (see page 5265)

27.1.60.2.43 TTreeView.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

27.1.60.2.44 TTreeView.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the

Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

27.1.60.2.45 TTreeView.ToolTips Property

Specifies whether the items in the tree view have tool tips.

Pascal

```
property ToolTips: Boolean;
```

C++

```
__property Boolean ToolTips;
```

Description

Set ToolTips to true to specify that items in the tree view control have tool tips (Help Hints).

Specify the ToolTip text in an OnHint event handler using the Hint property.

See Also

Hint (see page 5827)

OnHint (see page 13731)

27.1.60.2.46 TTreeView.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
```

```
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.60.3 TTreeView Events

The events of the TTreeView class are listed here.

Published Events

	Name	Description
	OnAddition (see page 5342)	Occurs when new node is added.
	OnAdvancedCustomDraw (see page 5342)	Occurs at discrete stages during the painting of the tree view control.
	OnAdvancedCustomDrawItem (see page 5342)	Occurs at discrete stages during the painting of tree view nodes.
	OnChange (see page 5343)	Occurs whenever the selection has changed from one node to another.
	OnChanging (see page 5343)	Occurs when the selection is about to change from one node to another.
	OnClick (see page 5344)	Occurs when the user clicks the control.
	OnCollapsed (see page 5345)	Occurs after a node has been collapsed.
	OnCollapsing (see page 5346)	Occurs when a node is about to be collapsed.
	OnCompare (see page 5346)	Occurs when two nodes must be compared during a sort of the nodes in the tree view.
	OnContextPopup (see page 5347)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreateNodeClass (see page 5347)	Occurs when a node in the treeview is about to be created.
	OnCustomDraw (see page 5348)	Occurs immediately prior to painting the tree view control.
	OnCustomDrawItem (see page 5349)	Occurs immediately prior to painting a node in a tree view control.
	OnDblClick (see page 5352)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeletion (see page 5352)	Occurs when a node in the tree view is deleted.
	OnDragDrop (see page 5352)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5355)	Occurs when the user drags an object over a control.
	OnEdited (see page 5357)	Occurs after the user edits the Text property of a node.
	OnEditing (see page 5357)	Occurs when the user starts to edit the Text property of a node.
	OnEndDock (see page 5358)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5358)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 5359)	Occurs when a control receives the input focus.
	OnExit (see page 5362)	Occurs when the input focus shifts away from one control to another.
	OnExpanded (see page 5363)	Occurs after a node is expanded.
	OnExpanding (see page 5364)	Occurs when a node is about to be expanded.
	OnGetImageIndex (see page 5364)	Occurs when the tree view looks up the ImageIndex of a node.
	OnGetSelectedIndex (see page 5365)	Occurs when the tree view looks up the SelectedIndex of a node.
	OnKeyDown (see page 5365)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 5368)	Occurs when a key is pressed.
	OnKeyUp (see page 5369)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5371)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5372)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5374)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5374)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5375)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5377)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 5379)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 5380)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

27.1.60.3.1 TTreeView.OnAddition Event

Occurs when new node is added.

Pascal

```
property OnAddition: TTVExpandedEvent;
```

C++

```
__property TTVExpandedEvent OnAddition;
```

Description

OnAddition occurs when a new node is added to the control.

27.1.60.3.2 TTreeView.OnAdvancedCustomDraw Event

Occurs at discrete stages during the painting of the tree view control.

Pascal

```
property OnAdvancedCustomDraw: TTVAdvancedCustomDrawEvent;
```

C++

```
__property TTVAdvancedCustomDrawEvent OnAdvancedCustomDraw;
```

Description

Write an OnAdvancedCustomDraw event handler to paint an owner-drawn tree view. Use the Canvas property as a drawing surface when painting the image of the tree view.

To paint individual items, use the OnAdvancedCustomDrawItem or OnCustomDrawItem event instead.

Note: OnAdvancedCustomDraw occurs at several stages during the paint process, not just immediately prior to the default rendering. If you only need to use the cdPrePaint stage, it is more efficient to use the OnCustomDraw event.

See Also

Canvas (see page 3997)

OnAdvancedCustomDrawItem (see page 4016)

OnCustomDraw (see page 4019)

27.1.60.3.3 TTreeView.OnAdvancedCustomDrawItem Event

Occurs at discrete stages during the painting of tree view nodes.

Pascal

```
property OnAdvancedCustomDrawItem: TTVAdvancedCustomDrawItemEvent;
```

C++

```
__property TTVAdvancedCustomDrawItemEvent OnAdvancedCustomDrawItem;
```

Description

Write an OnAdvancedCustomDrawItem event handler to customize the painting of individual items in the tree view.

Note: OnAdvancedCustomDrawItem occurs at several stages during the paint process and allows you to suppress the default painting of tree node images while still allowing the default painting of the node. If you only need to use the cdPrePaint stage and do not want to suppress only the painting of tree node images, it is more efficient to use the OnCustomDrawItem event.

See Also

Canvas (see page 3997)

OnAdvancedCustomDraw (see page 4016)

OnCustomDrawItem (see page 4021)

27.1.60.3.4 TTreeView.OnChange Event

Occurs whenever the selection has changed from one node to another.

Pascal

```
property OnChange: TTVChangedEvent;
```

C++

```
__property TTVChangedEvent OnChange;
```

Description

Write an OnChange event handler to take specific action when the selected node changes. The Sender parameter is the tree view whose selected node changes, and the Node parameter is the newly selected node.

Note: The OnChange event does not occur for nodes selected using the right mouse button when RightClickSelect is true. To respond to changes in the value of the Selected property when RightClickSelect is true, use the OnMouseUp event.

See Also

OnChanging (see page 4018)

Selected (see page 4007)

RightClickSelect (see page 4006)

ChangeDelay (see page 3999)

OnMouseUp (see page 5867)

27.1.60.3.5 TTreeView.OnChanging Event

Occurs when the selection is about to change from one node to another.

Pascal

```
property OnChanging: TTVChangingEvent;
```

C++

```
__property TTVChangingEvent OnChanging;
```

Description

Write an OnChanging event handler to selectively prevent selection from moving to specific nodes.

The TTVChangingEvent type points to a method that is called when the selection is about to be changed from one node to another. Set AllowChange to false, to prevent selection from moving to a new node. The Node parameter specifies the node that is about to be selected.

See Also

OnChange (see page 4017)

Selected (see page 4007)

27.1.60.3.6 TTreeView.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
```

```
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

27.1.60.3.7 TTreeView.OnCollapsed Event

Occurs after a node has been collapsed.

Pascal

```
property OnCollapsed: TTVExpandedEvent;
```

C++

```
__property TTVExpandedEvent OnCollapsed;
```

Description

Write an OnCollapsed event handler to respond after a node in the tree view collapses. The Node parameter is the node whose children are no longer visible.

See Also

Collapse (see page 5235)

FullCollapse (see page 3983)

27.1.60.3.8 TTreeView.OnCollapsing Event

Occurs when a node is about to be collapsed.

Pascal

```
property OnCollapsing: TTVCollapsingEvent;
```

C++

```
__property TTVCollapsingEvent OnCollapsing;
```

Description

The TTVCollapsingEvent type points to a method that is called when a node is about to be collapsed. Set the AllowCollapse parameter to false to prevent the node specified by the Node parameter from being collapsed.

See Also

Collapse (see page 5235)

FullCollapse (see page 3983)

27.1.60.3.9 TTreeView.OnCompare Event

Occurs when two nodes must be compared during a sort of the nodes in the tree view.

Pascal

```
property OnCompare: TTVCompareEvent;
```

C++

```
__property TTVCompareEvent OnCompare;
```

Description

Write an OnCompare event handler to customize the sort order of the nodes in the tree view. Set the Compare parameter to a value less than 0 if Node1 is less than Node2. Set Compare to 0 if Node1 is equivalent to Node2, and set Compare to a value greater than 0 if Node1 is greater than Node2. If an OnCompare event handler is not provided, tree view nodes are sorted alphabetically, based on their labels.

See Also

AlphaSort (see page 3972)

CustomSort (see page 3978)

SortType (see page 4014)

27.1.60.3.10 TTreeView.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.60.3.11 TTreeView.OnCreateNodeClass Event

Occurs when a node in the treeview is about to be created.

Pascal

```
property OnCreateNodeClass: TTVCreateNodeClassEvent;
```

C++

```
__property TTVCreateNodeClassEvent OnCreateNodeClass;
```

Description

OnCreateNodeClass occurs when a new node object is about to be created.

Sender is the tree view object that is about to add a new node.

NodeClass returns a class reference that is instantiated to create a new node object.

27.1.60.3.12 TTreeView.OnCustomDraw Event

Occurs immediately prior to painting the tree view control.

Pascal

```
property OnCustomDraw: TTVCustomDrawEvent;
```

C++

```
__property TTVCustomDrawEvent OnCustomDraw;
```

Description

Write an OnCustomDraw event handler to paint an owner-drawn tree view. Use the Canvas property as a drawing surface when painting the image of the tree view.

To paint individual items, use the OnCustomDrawItem event instead.

Note: OnCustomDraw only occurs prior to painting the tree view control. To customize the painting at other stages of the paint process (such as after the default drawing), use OnAdvancedCustomDraw instead.

C++ Examples:

```
/*
The following example is taken from the custom draw demo. It
shows how the OnCustomDraw event handler draws the
background for the tree view before the items and lines are
drawn.
*/
void __fastcall TCustomDrawForm::TVCustomDraw(TCustomTreeView *Sender,
      const TRect &ARect, bool &DefaultDraw)
{
/*
This event should be used to draw any background colors or
images.  ARect represents the entire client area of the TreeView.
Use the TreeView's canvas to do the drawing.
Note that drawing a background bitmap is not really
supported by CustomDraw, so scrolling can get messy. Best to
subclass the TreeView and handle scrolling messages.
*/
  if (None1 != NULL)
  {
    if (None1->Checked) //no picture
    {
      TV->Canvas->Brush->Color = BkgColorDialog->Color;
      TV->Canvas->Brush->Style = FBrushStyle;
      TV->Canvas->FillRect(ARect);
    }
    else if (Tile1->Checked) //tile bitmap
    {
      TV->Canvas->Brush->Bitmap = Image1->Picture->Bitmap;
      TV->Canvas->FillRect(ARect);
    }
    else //Stretch across the canvas.
      TV->Canvas->StretchDraw(ARect, Image1->Picture->Bitmap);
  };
  DefaultDraw = FDefaultDraw;
  // Setting DefaultDraw to false here prevents all calls to
  // OnCustomDrawItem.
}
```

Delphi Examples:

```
{
The following example is taken from the custom draw demo. It
shows how the OnCustomDraw event handler draws the
background for the tree view before the items and lines are
drawn.
}
procedure TCustomDrawForm.TVCustomDraw(Sender: TCustomTreeView; const ARect: TRect;
  var DefaultDraw: Boolean);
begin
{
This event should be used to draw any background colors or
images.  ARect represents the entire client area of the TreeView.
Use the TreeView's canvas to do the drawing.
Note that drawing a background bitmap is not really
supported by CustomDraw, so scrolling can get messy. Best to
subclass the TreeView and handle scrolling messages.
}
  with TV.Canvas do
  begin
    if None1 <> nil then
    begin
      if None1.Checked then //no picture
      begin
        Brush.Color := BkgColorDialog.Color;
        Brush.Style := FBrushStyle;
        FillRect(ARect);
      end else
        if Tile1.Checked then //tile bitmap
        begin
          Brush.Bitmap := Image1.Picture.Bitmap;
            FillRect(ARect);
          end else //Stretch across the canvas.
              StretchDraw(ARect, Image1.Picture.Bitmap);
    end;
  end;
  DefaultDraw := FDefaultDraw;
  // Setting DefaultDraw to false here prevents all calls to
  // OnCustomDrawItem.
end;
```

See Also

Canvas (see page 3997)

OnCustomDrawItem (see page 4021)

OnAdvancedCustomDraw (see page 4016)

27.1.60.3.13 TTreeView.OnCustomDrawItem Event

Occurs immediately prior to painting a node in a tree view control.

Pascal

```
property OnCustomDrawItem: TTVCustomDrawItemEvent;
```

C++

```
__property TTVCustomDrawItemEvent OnCustomDrawItem;
```

Description

Write an OnCustomDrawItem event handler to paint individual items in the tree view, or to provide a background to the item before the default rendering of the item.

Note: OnCustomDrawItem only occurs prior to painting individual tree items. To customize the painting of items at other stages

of the paint process (such as after the item is painted), use OnAdvancedCustomDrawItem instead.

C++ Examples:

```
/*
The following example shows how the OnCustomDrawItem event
handler draws items and lines of the tree view after the
OnCustomDraw event handler has filled in the background.
*/
void __fastcall TCustomDrawForm::TVCustomDrawItem(TCustomTreeView *Sender,
      TTreeNode *Node, TCustomDrawState State, bool &DefaultDraw)
{
  TRect NodeRect;
/*
    If DefaultDraw it is true, any of the node's font
    properties can be changed. Note also that when
    DefaultDraw = True, Windows draws the buttons and
    ignores our font background colors, using instead the
    TreeView's Color property.
*/
    if (State.Contains(cdsSelected))
    {
      TV->Canvas->Font->Assign(SelectedFontDialog->Font);
      TV->Canvas->Brush->Color = SelBkgColorDialog->Color;
    };

    DefaultDraw = FDefaultDrawItem;
/*
    DefaultDraw = False means you have to handle all the
    item drawing yourself, including the buttons, lines,
    images, and text.
*/
    if (!DefaultDraw)
    {
      //draw the selection rect.
      if (State.Contains(cdsSelected))
      {
        NodeRect = Node->DisplayRect(True);
        TV->Canvas->FillRect(NodeRect);
      };
      NodeRect = Node->DisplayRect(False);

      if (None1->Checked)
      //no bitmap, so paint in the background color.
      {
        TV->Canvas->Brush->Color = BkgColorDialog->Color;
        TV->Canvas->Brush->Style = FBrushStyle;
        TV->Canvas->FillRect(NodeRect);
      }
      else
        //don't paint over the background bitmap.
        TV->Canvas->Brush->Style = bsClear;

      NodeRect.Left = NodeRect.Left + (Node->Level * TV->Indent);
      // NodeRect.Left now represents the left-most portion
      // of the expand button
      DrawButton(&NodeRect, Node); // See the CustomDraw demo

      NodeRect.Left = NodeRect.Left + TV->Indent + FButtonSize;
      //NodeRect->Left is now the leftmost portion of the image.
      DrawImage(&NodeRect, Node->ImageIndex); // See the CustomDraw demo

      NodeRect.Left = NodeRect.Left + ImageList->Width;
      //Now we are finally in a position to draw the text.
```

```
      TV->Canvas->TextOut(NodeRect.Left, NodeRect.Top, Node->Text);
    };
}
```

Delphi Examples:

```
{
The following example shows how the OnCustomDrawItem event
handler draws items and lines of the tree view after the
OnCustomDraw event handler has filled in the background.
}
procedure TCustomDrawForm.TVCustomDrawItem(Sender: TCustomTreeView; Node: TTreeNode;
  State: TCustomDrawState; var DefaultDraw: Boolean);
var
  NodeRect: TRect;
begin
  with TV.Canvas do
  begin
{
    If DefaultDraw it is true, any of the node's font
    properties can be changed. Note also that when
    DefaultDraw = True, Windows draws the buttons and
    ignores our font background colors, using instead the
    TreeView's Color property.
}
    if cdsSelected in State then
    begin
      Font.Assign(SelectedFontDialog.Font);
      Brush.Color := SelBkgColorDialog.Color;
    end;

    DefaultDraw := False; // FDefaultDrawItem;
{
    DefaultDraw = False means you have to handle all the
    item drawing yourself, including the buttons, lines,
    images, and text.
}
    if not DefaultDraw then
    begin
      //draw the selection rect.
      if cdsSelected in State then
      begin
        NodeRect := Node.DisplayRect(True);
        FillRect(NodeRect);
      end;
      NodeRect := Node.DisplayRect(False);

      if None1.Checked then
      //no bitmap, so paint in the background color.
      begin
        Brush.Color := BkgColorDialog.Color;
        Brush.Style := FBrushStyle;
        FillRect(NodeRect)
      end
      else
        //don't paint over the background bitmap.
        Brush.Style := bsClear;

      NodeRect.Left := NodeRect.Left + (Node.Level * TV.Indent);
      // NodeRect.Left now represents the left-most portion
      // of the expand button
      DrawButton(NodeRect, Node); // See the CustomDraw demo

      NodeRect.Left := NodeRect.Left + TV.Indent + FButtonSize;
      //NodeRect.Left is now the leftmost portion of the image.
      DrawImage(NodeRect, Node.ImageIndex); // See the CustomDraw demo
```

```
        NodeRect.Left := NodeRect.Left + ImageList.Width;
        //Now we are finally in a position to draw the text.

        TextOut(NodeRect.Left, NodeRect.Top, Node.Text);
      end;
    end;
  end;
```

See Also

Canvas (see page 3997)

OnCustomDraw (see page 4019)

OnAdvancedCustomDrawItem (see page 4016)

27.1.60.3.14 TTreeView.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

27.1.60.3.15 TTreeView.OnDeletion Event

Occurs when a node in the tree view is deleted.

Pascal

```
property OnDeletion: TTVExpandedEvent;
```

C++

```
__property TTVExpandedEvent OnDeletion;
```

Description

Write an OnDeletion event handler to respond when a node is deleted from the tree view control.

See Also

Delete (see page 5237)

27.1.60.3.16 TTreeView.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;
```

```
procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

27.1.60.3.17 TTreeView.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
```

```
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;
```

```
// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.60.3.18 TTreeView.OnEdited Event

Occurs after the user edits the Text property of a node.

Pascal

```
property OnEdited: TTVEditedEvent;
```

C++

```
__property TTVEditedEvent OnEdited;
```

Description

Write an OnEdited event handler to respond to changes the user makes to the node labels. The Node parameter is the node whose label was edited. The S parameter is the new value of the node's Text property. The node's label can be changed in an OnEdited event handler before the user's edits are committed. This event can occur only if ReadOnly is set to false.

See Also

OnEditing (see page 4024)

ReadOnly (see page 4006)

Text (see page 5265)

27.1.60.3.19 TTreeView.OnEditing Event

Occurs when the user starts to edit the Text property of a node.

Pascal

```
property OnEditing: TTVEditingEvent;
```

C++

```
__property TTVEditingEvent OnEditing;
```

Description

Write an OnEditing event handler to determine whether the user is allowed to edit the label of a specific node in the tree view. Set the AllowEdit parameter to false to prevent the user from editing the node specified by the Node parameter. To disallow editing of all nodes in the tree view, use the ReadOnly property instead.

See Also

OnEdited (see page 4024)

ReadOnly (see page 4006)

27.1.60.3.20 TTreeView.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

27.1.60.3.21 TTreeView.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
```

```
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

27.1.60.3.22 TTreeView.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

27.1.60.3.23 TTreeView.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
```

```
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

27.1.60.3.24 TTreeView.OnExpanded Event

Occurs after a node is expanded.

Pascal

```
property OnExpanded: TTVExpandedEvent;
```

C++

```
__property TTVExpandedEvent OnExpanded;
```

Description

Write an OnExpanded event handler to respond when a node in the tree view is expanded. The Node parameter specifies the node whose children are now displayed to the user.

See Also

FullExpand (see page 3984)

OnExpanding (see page 4025)

Expand (see page 5242)

27.1.60.3.25 TTreeView.OnExpanding Event

Occurs when a node is about to be expanded.

Pascal

```
property OnExpanding: TTVExpandingEvent;
```

C++

```
__property TTVExpandingEvent OnExpanding;
```

Description

Write an OnExpanding event handler to determine whether a node can be expanded. Set the AllowExpansion parameter to false to prevent the node from expanding.

See Also

FullExpand (see page 3984)

OnExpanded (see page 4025)

Expand (see page 5242)

27.1.60.3.26 TTreeView.OnGetImageIndex Event

Occurs when the tree view looks up the ImageIndex of a node.

Pascal

```
property OnGetImageIndex: TTVExpandedEvent;
```

C++

```
__property TTVExpandedEvent OnGetImageIndex;
```

Description

Write an OnGetImageIndex event handler to change the image index for the particular node before it is drawn. For example, the bitmap of a node can be changed to indicate a different state for the node.

Delphi Examples:

```
{
The following example shows how the OnGetImageIndex event
handler can be used to indicate a different state for the
node.
}
procedure TCustomDrawForm.TVGetImageIndex(Sender: TObject; Node: TTreeNode);
begin
  if Node.HasChildren then
    if Node.Expanded then
      Node.ImageIndex := 3
    else
      Node.ImageIndex := 0
  else
    Node.ImageIndex := 1;
end;

procedure TCustomDrawForm.TVGetSelectedIndex(Sender: TObject; Node: TTreeNode);
begin
  Node.SelectedIndex := Node.ImageIndex;
end;
```

See Also

Images (see page 4001)

OnGetSelectedIndex (see page 4026)

ImageIndex (see page 5259)

27.1.60.3.27 TTreeView.OnGetSelectedIndex Event

Occurs when the tree view looks up the SelectedIndex of a node.

Pascal

```
property OnGetSelectedIndex: TTVExpandedEvent;
```

C++

```
__property TTVExpandedEvent OnGetSelectedIndex;
```

Description

Write an OnGetSelectedIndex event handler to change the selected image index of a node before it is drawn.

Delphi Examples:

```
{
The following example shows how the OnGetImageIndex event
handler can be used to indicate a different state for the
node.
}
procedure TCustomDrawForm.TVGetImageIndex(Sender: TObject; Node: TTreeNode);
begin
  if Node.HasChildren then
    if Node.Expanded then
      Node.ImageIndex := 3
    else
      Node.ImageIndex := 0
  else
    Node.ImageIndex := 1;
end;

procedure TCustomDrawForm.TVGetSelectedIndex(Sender: TObject; Node: TTreeNode);
begin
  Node.SelectedIndex := Node.ImageIndex;
end;
```

See Also

Images (see page 4001)

OnGetImageIndex (see page 4026)

SelectedIndex (see page 5265)

27.1.60.3.28 TTreeView.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
```

```
      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

27.1.60.3.29 TTreeView.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys

that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

27.1.60.3.30 TTreeView.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
```

```
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

27.1.60.3.31 TTreeView.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the

OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.60.3.32 TTreeView.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
```

```
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
```

```
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.60.3.33 TTreeView.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

27.1.60.3.34 TTreeView.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

27.1.60.3.35 TTreeView.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
```

```
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
```

```
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.60.3.36 TTreeView.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
```

```
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
```

```
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

27.1.60.3.37 TTreeView.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

27.1.60.3.38 TTreeView.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

27.1.61 TUpDown Class

TUpDown is a wrapper for a Windows up-down control.

Class Hierarchy

Pascal

```
TUpDown = class(TCustomUpDown);
```

C++

```
class TUpDown : public TCustomUpDown;
```

File

ComCtrls

Description

Use TUpDown to add an up-down control to a form. Up-down controls consist of a pair of arrow buttons, such as the arrows that appear in a spin box. Up-down controls allow users to change the size of a numerical value by clicking on arrow buttons.

See Also

TScrollBar (see page 21821)

TTrackBar (see page 5172)

Specialized Input Controls

Up-down Controls (VCL Only)

27.1.61.1 TUpDown Members

The following tables list the members exposed by TUpDown.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.

	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanChange (see page 4039)	Indicates whether Position (see page 4045) can change.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 4039)	Generates an OnClick (see page 4047) event.
	Click (see page 5743)	Respond to user click.

	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 4039)	Initializes the window creation parameters for the up-down control.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 4040)	Creates the window used by the up-down control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DoCanChange (see page 4040)	Pending change event dispatcher.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.

	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.

	Notification (see page 4041)	Responds to notifications when components are about to be created or destroyed.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.

	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.

	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 4038)	Creates and initializes an instance of TCustomUpDown (see page 4027).
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.

	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.

	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChanging (see page 5409)	Occurs when the position of an up-down control is about to change.
	OnChangingEx (see page 5410)	Occurs when the position of an up-down control is about to change.
	OnClick (see page 5410)	Occurs when the user clicks an arrow button.
	OnContextPopup (see page 5412)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnEnter (see page 5412)	Occurs when a control receives the input focus.
	OnExit (see page 5415)	Occurs when the input focus shifts away from one control to another.
	OnMouseActivate (see page 5416)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5417)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5420)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5420)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5420)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5422)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Published Properties

	Name	Description
	AlignButton (see page 5392)	Specifies the position of the up-down control relative to its companion control.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 5393)	Specifies how the control is anchored to its parent.
	ArrowKeys (see page 5393)	Determines whether the up-down control receives input from the Up arrow and Down arrow keys.
	Associate (see page 5394)	Specifies a companion control for the up-down control.
	Constraints (see page 5394)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Enabled (see page 5395)	Controls whether the control responds to mouse, keyboard, and timer events.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5395)	Contains the text string that can appear when the user moves the mouse over the control.
	Increment (see page 5397)	Specifies the amount the Position value changes each time the up or down button is pressed.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Max (see page 5397)	Specifies the maximum value of the Position property.
	Min (see page 5398)	Specifies the minimum value of the Position property.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Orientation (see page 5398)	Specifies the direction that the arrow keys point.
	ParentShowHint (see page 5399)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 5399)	Identifies the pop-up menu associated with the control.
	Position (see page 5404)	Specifies the current value represented by the up-down control.
	ShowHint (see page 5404)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 5406)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 5406)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Thousands (see page 5407)	Determines whether a thousands separator appears between every three digits of a decimal string.

	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 5407)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	Wrap (see page 5408)	Specifies how the up-down control handles attempts to exceed the value of Max or Min.

27.1.61.2 TUpDown Properties

The properties of the TUpDown class are listed here.

Published Properties

	Name	Description
	AlignButton (see page 5392)	Specifies the position of the up-down control relative to its companion control.
	Anchors (see page 5393)	Specifies how the control is anchored to its parent.
	ArrowKeys (see page 5393)	Determines whether the up-down control receives input from the Up arrow and Down arrow keys.
	Associate (see page 5394)	Specifies a companion control for the up-down control.
	Constraints (see page 5394)	Specifies the size constraints for the control.
	Enabled (see page 5395)	Controls whether the control responds to mouse, keyboard, and timer events.
	Hint (see page 5395)	Contains the text string that can appear when the user moves the mouse over the control.
	Increment (see page 5397)	Specifies the amount the Position value changes each time the up or down button is pressed.
	Max (see page 5397)	Specifies the maximum value of the Position property.
	Min (see page 5398)	Specifies the minimum value of the Position property.
	Orientation (see page 5398)	Specifies the direction that the arrow keys point.
	ParentShowHint (see page 5399)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 5399)	Identifies the pop-up menu associated with the control.
	Position (see page 5404)	Specifies the current value represented by the up-down control.
	ShowHint (see page 5404)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 5406)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 5406)	Determines if the user can tab to a control.
	Thousands (see page 5407)	Determines whether a thousands separator appears between every three digits of a decimal string.
	Visible (see page 5407)	Determines whether the component appears onscreen.
	Wrap (see page 5408)	Specifies how the up-down control handles attempts to exceed the value of Max or Min.

27.1.61.2.1 TUpDown.AlignButton Property

Specifies the position of the up-down control relative to its companion control.

Pascal

```
property AlignButton: TUDAlignButton;
```

C++

```
__property TUDAlignButton AlignButton;
```

Description

Set AlignButton to indicate where to position the up-down control. The up-down control appears attached to the outer edge of the control specified by the Associate property. These are the possible values:

Value	Meaning
udLeft	Position the up-down control on the left edge of the companion control.
udRight	Position the up-down control on the right edge of the companion control.

The up-down control resizes itself to the height of the companion control.

See Also

Associate (see page 4043)

27.1.61.2.2 TUpDown.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

27.1.61.2.3 TUpDown.ArrowKeys Property

Determines whether the up-down control receives input from the Up arrow and Down arrow keys.

Pascal

```
property ArrowKeys: Boolean;
```

C++

```
__property Boolean ArrowKeys;
```

Description

Use ArrowKeys to specify whether the up-down control responds to keyboard input. ArrowKeys is used only when the up-down control is associated with a companion control. When ArrowKeys is true, the up-down control responds to the Up arrow and Down arrow keys, even when its companion control has input focus. When ArrowKeys is false, the user must click the buttons of

the up-down control with the mouse. To set a companion control use the Associate property.

Note: Up-down controls can respond to only the Up and Down arrow keys, even when the value of Orientation is udHorizontal.

See Also

Associate (see page 4043)

Orientation (see page 4044)

27.1.61.2.4 TUpDown.Associate Property

Specifies a companion control for the up-down control.

Pascal

```
property Associate: TWinControl;
```

C++

```
__property TWinControl * Associate;
```

Description

Use Associate to link the up-down control to a companion control (sometimes called a buddy window). The up-down control automatically positions itself on the outer edge of the companion control and adjusts its height to match the companion control.

At design-time the companion control does not move with the up-down control. To keep the up-down control and its companion control together at design time, either multi-select both controls before moving them, or first move the companion control and then reset the Associate property for the up-down control.

See Also

AlignButton (see page 4042)

ArrowKeys (see page 4042)

Position (see page 4045)

27.1.61.2.5 TUpDown.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

27.1.61.2.6 TUpDown.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

27.1.61.2.7 TUpDown.Hint Property

Contains the text string that can appear when the user moves the mouse over the control.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Use the Hint property to provide a string of help text either as a Help Hint, or as help text on a particular location such as a status bar.

A Help Hint is a box containing help text that appears for a control when the user moves the mouse pointer over the control and pauses momentarily. To set up Help Hints:

Specify the Hint property of each control for which a Help Hint should appear.

Set the ShowHint property of each appropriate control to true, or set the ParentShowHint property of all controls to true and set the ShowHint property of the form to true.

At runtime, set the value of the application's ShowHint property to true.

To show the Hint on a status bar or other location, use the OnHint event handler of the application. The application's OnHint event occurs when the mouse pointer moves over the control.

Specify a hint to be used for both a Help Hint box and by an OnHint handler by specifying two values separated by a | character (the vertical bar "or" symbol). For example,

```
Edit1.Hint := 'Name|Enter Name in the edit box';
Edit1->Hint = "Name|Enter Name in the edit box";
```

The "Name" part appears in the Help Hint box and the "Enter full name in the edit box" part can be extracted using the GetLongHint function for use in an OnHint event handler.

If Hint contains only one value, the entire string is used as a Help Hint and returned by the GetLongHint function. If a control has no Hint value specified, but its parent control does, the control uses the Hint value of the parent control (as long as the control's ShowHint property is true).

Note: If the application's ShowHint property is false, the Help Hint does not appear, but the OnHint event handler is still called.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
```

```
    Edit1.Hint := 'Enter your name';
    Edit1.ShowHint := True;
    with ListBox1 do
    begin
      for I := 1 to 10 do
        Items.Add('Item ' + IntToStr(I));
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

GetLongHint (see page 6252)

GetShortHint (see page 6253)

ParentShowHint (see page 5835)

ShowHint (see page 5840)

Hint (see page 13712)

OnHint (see page 13731)

ShowHint (see page 13721)

Hint (see page 312)

27.1.61.2.8 TUpDown.Increment Property

Specifies the amount the Position value changes each time the up or down button is pressed.

Pascal

```
property Increment: Integer;
```

C++

```
__property int Increment;
```

Description

Use Increment to control the granularity of changes the up-down control can make. Set Increment to indicate the number of units between Min and Max the Position moves whenever the user clicks on one of the arrow buttons. Increment represents the mapping between the units used to specify Min, Max, and Position, and the logical units represented by the up-down control.

See Also

Max (see page 4043)

Min (see page 4044)

Position (see page 4045)

27.1.61.2.9 TUpDown.Max Property

Specifies the maximum value of the Position property.

Pascal

```
property Max: SmallInt;
```

C++

```
__property SmallInt Max;
```

Description

Use Max to provide an upper limit to the range of values the Position property can assume.

See Also

Min (see page 4044)

Position (see page 4045)

27.1.61.2.10 TUpDown.Min Property

Specifies the minimum value of the Position property.

Pascal

```
property Min: SmallInt;
```

C++

```
__property SmallInt Min;
```

Description

Use Min to provide a lower limit to the range of values the Position property can assume.

See Also

Max (see page 4043)

Position (see page 4045)

27.1.61.2.11 TUpDown.Orientation Property

Specifies the direction that the arrow keys point.

Pascal

```
property Orientation: TUDOrientation;
```

C++

```
__property TUDOrientation Orientation;
```

Description

Use Orientation to determine the orientation of the arrow buttons. These are the possible values:

Value	Meaning
udHorizontal	The arrow buttons point to the left and right, and appear side by side.
udVertical	The arrow buttons point up and down and are stacked vertically.

Note: Even when the Orientation is udHorizontal, the up-down control can respond only to the Up and Down arrow keys.

See Also

ArrowKeys (see page 4042)

27.1.61.2.12 TUpDown.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

27.1.61.2.13 TUpDown.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
```

```
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
```

```
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
```

```
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
```

```
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
```

```
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

27.1.61.2.14 TUpDown.Position Property

Specifies the current value represented by the up-down control.

Pascal

```
property Position: SmallInt;
```

C++

```
__property SmallInt Position;
```

Description

Use Position to get or change the value the up-down control represents. Position must be a value between Min and Max.

When the Associate property specifies an edit control, its text is automatically set to the value of Position. When Position changes, the text of the edit control is updated. If the Associate property is nil (Delphi) or NULL (C++), use the OnClick event to respond to changes in the value of Position.

See Also

Associate (see page 4043)

Max (see page 4043)

Min (see page 4044)

OnChanging (see page 4046)

Thousands (see page 4045)

27.1.61.2.15 TUpDown.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

27.1.61.2.16 TUpDown.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

27.1.61.2.17 TUpDown.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

27.1.61.2.18 TUpDown.Thousands Property

Determines whether a thousands separator appears between every three digits of a decimal string.

Pascal

```
property Thousands: Boolean;
```

C++

```
__property Boolean Thousands;
```

Description

Use Thousands to control the formatting of the text of the companion control of the up-down control. If Thousands is true, a separator appears between every three digits. The numeric value that is displayed by the companion control is given by the Position property.

See Also

Associate (see page 4043)

Position (see page 4045)

27.1.61.2.19 TUpDown.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

27.1.61.2.20 TUpDown.Wrap Property

Specifies how the up-down control handles attempts to exceed the value of Max or Min.

Pascal

```
property Wrap: Boolean;
```

C++

```
__property Boolean Wrap;
```

Description

Use Wrap to specify whether the up-down control treats the range determined by the Max and Min properties as a continuous loop. If Wrap is true, the value of Position becomes the value of the Min property when the user increments Position beyond the value of Max, and becomes the value of Max when the user decrements Position beyond the value of Min.

See Also

Max (see page 4043)

Min (see page 4044)

Position (see page 4045)

27.1.61.3 TUpDown Events

The events of the TUpDown class are listed here.

Published Events

	Name	Description
	OnChanging (see page 5409)	Occurs when the position of an up-down control is about to change.
	OnChangingEx (see page 5410)	Occurs when the position of an up-down control is about to change.
	OnClick (see page 5410)	Occurs when the user clicks an arrow button.
	OnContextPopup (see page 5412)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnEnter (see page 5412)	Occurs when a control receives the input focus.
	OnExit (see page 5415)	Occurs when the input focus shifts away from one control to another.
	OnMouseActivate (see page 5416)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5417)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5420)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5420)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5420)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5422)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

27.1.61.3.1 TUpDown.OnChanging Event

Occurs when the position of an up-down control is about to change.

Pascal

```
property OnChanging: TUDChangingEvent;
```

C++

```
__property TUDChangingEvent OnChanging;
```

Description

Write an OnChanging event handler to conditionally disallow changes to the value of Position. Set the AllowChange parameter to false to prevent a change to the current value of Position.

OnChanging occurs both when the user clicks the arrow buttons of the up-down control and when an attempt is made to change the value of Position programmatically. To determine which button the user clicked, use the OnChangingEx event or the OnClick event, both of which follow the OnChanging event.

See Also

CanChange (see page 4039)

OnClick (see page 4047)

OnChangingEx (see page 4047)

Position (see page 4045)

27.1.61.3.2 TUpDown.OnChangingEx Event

Occurs when the position of an up-down control is about to change.

Pascal

```
property OnChangingEx: TUDChangingEventEx;
```

C++

```
__property TUDChangingEventEx OnChangingEx;
```

Description

Write an OnChangingEx event handler to conditionally disallow changes to the value of Position. Set the AllowChange parameter to false to prevent a change to the current value of Position.

Sender is the up-down control that received the event. This may be in response to a user click on the arrow buttons or an attempt to change the value of Position programmatically.

AllowChange returns a value that indicates whether the change is permitted.

NewValue indicates the new value of Position if the change is allowed.

Direction indicates whether the value of Position is to be increased or decreased.

See Also

CanChange (see page 4039)

Position (see page 4045)

OnChanging (see page 4046)

27.1.61.3.3 TUpDown.OnClick Event

Occurs when the user clicks an arrow button.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TUDClickEvent OnClick;__property TNotifyEvent OnClick;
```

Description

Write an OnClick event handler to take specific action when the user clicks one of the arrow buttons. If the Associate property is not used to automatically update the text of an edit control when the value of Position changes, use the OnClick event handler to respond to changes just before the value of Position is changed.

The Button parameter specifies how the value of Position is about to change. When Button is btNext, the Up or Right arrow was clicked, and the value of Position is about to increase by Increment. When Button is btPrev, the Down or Left arrow was clicked, and the value of Position is about to decrease by Increment.

C++ Examples:

```
/*
This example requires a new TPageControl, as-is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
evaluates as true.  If Button is not btNext, the previous Tab Page
is selected.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
const
  TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *page = new TTabSheet(PageControl1);
    page->PageControl = PageControl1;
    page->Caption = AnsiString("Page") + IntToStr(i);
    page->Brush->Color = colorPalette[i];
  }
}

void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  PageControl1->SelectNextPage(Button == btNext);
}
```

Delphi Examples:

```
{
This example requires a new TPageControl, as-is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
evaluates as true.  If Button is not btNext, the previous Tab Page
is selected.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
const
  colorPalette: Array[0..11] of TColor =
    (clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet #' + IntToStr(i);
      Brush.Color := colorPalette[i];
    end;
end;
```

```
procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
begin
  PageControl1.SelectNextPage(Button = btNext);
end;
```

See Also

Associate (see page 4043)

Increment (see page 4043)

OnChanging (see page 4046)

Position (see page 4045)

27.1.61.3.4 TUpDown.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

27.1.61.3.5 TUpDown.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
```

```
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;
```

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

27.1.61.3.6 TUpDown.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
```

```
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

27.1.61.3.7 TUpDown.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.61.3.8 TUpDown.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
```

```
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.61.3.9 TUpDown.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

27.1.61.3.10 TUpDown.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

27.1.61.3.11 TUpDown.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
```

```
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

27.1.61.3.12 TUpDown.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
```

```
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
```

```
      end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

27.1.62 TWorkArea Class

TWorkArea represents a rectangular subregion of a list view's client area.

Class Hierarchy

System::TObject → Classes.TPersistent → Classes.TCollectionItem → ComCtrls.TWorkArea

Pascal

```
TWorkArea = class(TCollectionItem);
```

C++

```
class TWorkArea : public TCollectionItem;
```

File

ComCtrls

Description

Use TWorkArea to describe the dimensions and display characteristics of one work area of a list view component. Work areas divide the client area of a list view into logical subregions. The icons of the list view are then arranged within each work area, to provide an additional layer of organization.

Each list item in the list view belongs to a single work area. This association is based on whether the position of the list item falls within the bounding rectangle specified by the work area's Rect (see page 5428) property.

In addition, If the list view's ShowWorkAreas property is true, the list view draws the work area, using its Color (see page 5428) and DisplayName properties.

Note: Work areas are only used when the list view has a ViewStyle of vsIcon or vsSmallIcon.

See Also

TWorkAreas (see page 5429)

TCustomListView (see page 3757)

TCustomListView (see page 3757)

27.1.62.1 TWorkArea Members

The following tables list the members exposed by TWorkArea.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.

	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
V	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
V	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 5427)	Creates an instance of TWorkArea (see page 5425).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
V	GetDisplayName (see page 5427)	Returns the value of the DisplayName property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetDisplayName (see page 5427)	Sets the value of the DisplayName property.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	Color (see page 5428)	Specifies the background color of the work area.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.

	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
	Rect (see page 5428)	Indicates the portion of the list view's client area covered by the work area.

27.1.62.2 TWorkArea.Create Constructor

Creates an instance of TWorkArea (see page 5425).

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TWorkArea(TCollection Collection);
```

Description

It is seldom necessary to call the constructor for TWorkArea (see page 5425) directly. Instead, create TWorkArea (see page 5425) objects indirectly by calling the Add method of TWorkAreas (see page 5429).

See Also

Add (see page 5431)

27.1.62.3 TWorkArea Methods

The methods of the TWorkArea class are listed here.

Public Methods

	Name	Description
	GetDisplayName (see page 5427)	Returns the value of the DisplayName property.
	SetDisplayName (see page 5427)	Sets the value of the DisplayName property.

27.1.62.3.1 TWorkArea.GetDisplayName Method

Returns the value of the DisplayName property.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the read implementer of the DisplayName property. The DisplayName indicates the name that appears as a label for the work area in the list view when the list view's ShowWorkAreas property is true.

See Also

DisplayName (see page 2761)

ShowWorkAreas (see page 3824)

27.1.62.3.2 TWorkArea.SetDisplayName Method

Sets the value of the DisplayName property.

Pascal

```
procedure SetDisplayName(const Value: string); override;
```

C++

```
virtual __fastcall SetDisplayName(const AnsiString Value);
```

Description

SetDisplayName is the write implementer of the DisplayName property. The DisplayName indicates the name that appears as a label for the work area in the list view when the list view's ShowWorkAreas property is true.

See Also

DisplayName (see page 2761)

ShowWorkAreas (see page 3824)

27.1.62.4 TWorkArea Properties

The properties of the TWorkArea class are listed here.

Public Properties

	Name	Description
	Color (see page 5428)	Specifies the background color of the work area.
	Rect (see page 5428)	Indicates the portion of the list view's client area covered by the work area.

27.1.62.4.1 TWorkArea.Color Property

Specifies the background color of the work area.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to specify the color in which the list view paints the portion of its client area circumscribed by the Rect (see page 5428) property. Color is not used unless the list view's ShowWorkAreas property is true.

See Also

Rect (see page 5428)

ShowWorkAreas (see page 3824)

27.1.62.4.2 TWorkArea.Rect Property

Indicates the portion of the list view's client area covered by the work area.

Pascal

```
property Rect: TRect;
```

C++

```
__property TRect Rect;
```

Description

Set Rect to specify the sphere of influence of a work area. Rect is the bounding rectangle of a portion of the list view's client area. If Rect specifies a region larger than the list view's client area, the list view introduces scroll bars to support the logical size specified by Rect.

List items are assigned to the work area when they fall into the region of the list view bounded by Rect. If a list item falls into a region that is contained by more than one work area, it is assigned to the list item with the lowest Index property.

If the list view paints itself to display work areas, it paints the portion of its client area that is outlined by Rect using the color specified by the Color (see page 5428) property.

See Also

Color (see page 5428)

ClientRect

WorkArea (see page 4257)

Index (see page 2762)

ShowWorkAreas (see page 3824)

27.1.63 TWorkAreas Class

TWorkAreas is a collection of TWorkArea (see page 5425) objects.

Class Hierarchy

Pascal

```
TWorkAreas = class(TOwnedCollection);
```

C++

```
class TWorkAreas : public TOwnedCollection;
```

File

ComCtrls

Description

Use TWorkAreas to represent the work areas of a list view control. Each work area is represented by a single TWorkArea (see page 5425) object. Most importantly, these work areas define rectangles that cover portions of the list view's client area. List items are associated with a work area when they are placed in the portion of the client area covered by a work area's rectangle. These associations are used to add an extra level of organization to the list view, with list items arranged within each work area.

See Also

TWorkArea (see page 5425)

TCustomListView (see page 3757)

27.1.63.1 TWorkAreas Members

The following tables list the members exposed by TWorkAreas.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 5432)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2915)	Returns the owner of an object.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 5433)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 5431)	Creates a new TWorkArea (see page 5425) object and adds it to the end of the Items (see page 5434) property array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 2734)	Creates and initializes a collection.
	Create (see page 2914)	Creates and initializes a TOwnedCollection (see page 2912) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 5432)	Removes a TWorkArea (see page 5425) object from the Items (see page 5434) property array.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.

V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 5433)	Creates a new TWorkArea (see page 5425) object and adds it to the Items (see page 5434) property array in a specified position.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 5434)	Provides indexed access to the TWorkArea (see page 5425) objects.

27.1.63.2 TWorkAreas Methods

The methods of the TWorkAreas class are listed here.

Protected Methods

	Name	Description
	Changed (see page 5432)	Responds when the collection or any of its items changes.
V	Update (see page 5433)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 5431)	Creates a new TWorkArea (see page 5425) object and adds it to the end of the Items (see page 5434) property array.
	Delete (see page 5432)	Removes a TWorkArea (see page 5425) object from the Items (see page 5434) property array.
	Insert (see page 5433)	Creates a new TWorkArea (see page 5425) object and adds it to the Items (see page 5434) property array in a specified position.

27.1.63.2.1 TWorkAreas.Add Method

Creates a new TWorkArea (see page 5425) object and adds it to the end of the Items (see page 5434) property array.

Pascal

```
function Add: TWorkArea;
```

C++

```
__fastcall TWorkArea Add();
```

Description

Call Add to add a new work area to the list view that owns this TWorkAreas (see page 5429) object. Add returns the newly created TWorkArea (see page 5425) object. After adding a new work area, set the work area's Rect property to indicate what portion of the list view's client area it represents.

See Also

Insert (see page 5433)

Items (see page 5434)

Rect (see page 5428)

27.1.63.2.2 TWorkAreas.Changed Method

Responds when the collection or any of its items changes.

Pascal

```
procedure Changed;
```

C++

```
__fastcall Changed();
```

Description

Changed is called automatically when items in the collection change or when the EndUpdate method signals that an update is complete. It checks the value of UpdateCount, and if it is 0, calls the Update method, which performs any necessary updates.

When writing a TCollection descendant, there is no need to call Changed. Instead, bracket any changes by calls to BeginUpdate and EndUpdate.

See Also

Changed (see page 2758)

Update (see page 2747)

EndUpdate (see page 2742)

UpdateCount (see page 2753)

27.1.63.2.3 TWorkAreas.Delete Method

Removes a TWorkArea (see page 5425) object from the Items (see page 5434) property array.

Pascal

```
procedure Delete(Index: Integer);
```

C++

```
__fastcall Delete(int Index);
```

Description

Call Delete to remove the work area specified by Items (see page 5434)[Index]. Delete removes the work area from the Items

(see page 5434) property array and frees the TWorkArea (see page 5425) object.

See Also

Insert (see page 5433)

Items (see page 5434)

Add (see page 5431)

27.1.63.2.4 TWorkAreas.Insert Method

Creates a new TWorkArea (see page 5425) object and adds it to the Items (see page 5434) property array in a specified position.

Pascal

```
function Insert(Index: Integer): TWorkArea;
```

C++

```
__fastcall TWorkArea Insert(int Index);
```

Description

Call Insert to add a new work area to the list view that owns this TWorkAreas (see page 5429) object. Insert adds a new work area at the position specified by the Index parameter. Existing work areas (starting from the specified position) are moved up in the Items (see page 5434) array. Insert returns the newly created TWorkArea (see page 5425) object. After inserting a new work area, set the work area's Rect property to indicate what portion of the list view's client area it represents.

Note: Usually, there is no meaningful difference between using the Add (see page 5431) method or the Insert method to add work areas. However, if two or more work areas specify overlapping regions, then the order of the work areas in the Items (see page 5434) array determines which work area is associated with list items in the overlapping region. Specifically, list items in the overlapping region are associated with the work area that has the lower index.

See Also

Add (see page 5431)

Items (see page 5434)

Rect (see page 5428)

27.1.63.2.5 TWorkAreas.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary

adjustments.

See Also

Changed (see page 2740)

Changed (see page 2758)

27.1.63.3 TWorkAreas Properties

The properties of the TWorkAreas class are listed here.

Public Properties

	Name	Description
	Items (see page 5434)	Provides indexed access to the TWorkArea (see page 5425) objects.

27.1.63.3.1 TWorkAreas.Items Property

Provides indexed access to the TWorkArea (see page 5425) objects.

Pascal

```
property Items [Index: Integer]: TWorkArea;
```

C++

```
__property TWorkArea Items[int Index];
```

Description

Use Items to access the properties of the TWorkArea (see page 5425) objects maintained by TWorkAreas (see page 5429).

Note: In C++ code, Items is the default property of TWorkAreas (see page 5429), so the name of the Items property can be left off when using this property. Thus the line

```
ListView1.WorkAreas.Items[0].Color := 'clLtGray';
```

Note: can be written

```
ListView1.WorkAreas[0].Color := 'clLtGray';
```

See Also

Cookies (see page 15210)

operator_[]

27.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CheckCommonControl (see page 5435)	This is function ComCtrls.CheckCommonControl.
	CheckToolMenuDropdown (see page 5435)	Displays a tool button's dropdown menu.
	GetComCtlVersion (see page 5436)	Indicates which version of ComCtl32.dll is installed.
	InitCommonControl (see page 5437)	This is function ComCtrls.InitCommonControl.

27.2.1 ComCtrls.CheckCommonControl Function

Pascal

```
procedure CheckCommonControl(CC: Integer);
```

C++

```
CheckCommonControl(int CC);
```

File

ComCtrls

Description

This is function ComCtrls.CheckCommonControl.

27.2.2 ComCtrls.CheckToolMenuDropdown Function

Displays a tool button's dropdown menu.

Pascal

```
procedure CheckToolMenuDropdown(ToolButton: TToolButton);
```

C++

```
CheckToolMenuDropdown(TToolButton ToolButton);
```

File

ComCtrls

Description

CallCheckToolMenuDropdown to programmatically display the dropdown menu for a tool button of style tbsDropDown. If the tool button has no associated dropdown menu,CheckToolMenuDropdown does nothing.

Note: CheckToolMenuDropdown is included for backward compatibility. Applications should use the tool button's CheckMenuDropdown method instead.

See Also

TToolButton

TToolButton

27.2.3 ComCtrls.GetComCtlVersion Function

Indicates which version of ComCtl32.dll is installed.

Pascal

```
function GetComCtlVersion: Integer;
```

C++

```
int GetComCtlVersion();
```

File

ComCtrls

Description

The appearance and behavior of some common controls, such as tool bars and cool bars, varies depending on which version of ComCtl32.dll is installed. UseGetComCtlVersion to determine which version is installed. GetComCtlVersion returns an integer that includes the major version number as the high-order word and the minor version number as the low-order word.

Installing Microsoft's Internet Explorer on a system affects which version of ComCtl32.dll is installed. To help determine which version of Internet Explorer is installed, the ComCtrls (see page 3413) unit defines several constants:

ConstantValueMeaning

Constant	Value	Meaning
ComCtlVersionIE3 (see page 5522)	$00040046	The version of ComCtl32.dll installed with Internet Explorer version 3.
ComCtlVersionIE4 (see page 5523)	$00040047	The version of ComCtl32.dll installed with Internet Explorer version 4.
ComCtlVersionIE401 (see page 5523)	$00040048	The version of ComCtl32.dll installed with Internet Explorer version 4.01.
ComCtlVersionIE5 (see page 5523)	$00050050	The version of ComCtl32.dll installed with Internet Explorer version 5.

To quickly determine which version of Internet Explorer is installed, you can compare the return value ofGetComCtlVersion with these constants. For example:

ifGetComCtlVersion >= ComCtlVersionIE4 (see page 5523) then { IE4 is installed }

if (GetComCtlVersion() >= ComCtlVersionIE4 (see page 5523)) // IE4 is installed

See Also

TCoolBar

TToolBar

27.2.4 ComCtrls.InitCommonControl Function

Pascal

```
function InitCommonControl(CC: Integer): Boolean;
```

C++

```
Boolean InitCommonControl(int CC);
```

File

ComCtrls

Description

This is function ComCtrls.InitCommonControl.

27.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAddMode (see page 5439)	This is record ComCtrls.TAddMode.
	TAttributeType (see page 5440)	This is record ComCtrls.TAttributeType.
	TAutoCompleteOption (see page 5440)	TAutoCompleteOption and TAutoCompleteOptions (see page 5480) specify a set of autocomplete options.
	TCalDayOfWeek (see page 5441)	TCalDayOfWeek represents a day of the week.
	TComboBoxExStyle (see page 5442)	TComboBoxExStyle is the type for the Style property of extended combo box controls.
	TComboBoxExStyleEx (see page 5442)	TComboBoxExStyleEx and TComboBoxExStyles (see page 5481) configure the behavior of extended combo box controls.
	TCommonAVI (see page 5443)	This is record ComCtrls.TCommonAVI.
	TConsistentAttribute (see page 5444)	This is record ComCtrls.TConsistentAttribute.
	TCoolBandMaximize (see page 5445)	TCoolBandMaximize specifies the conditions under which a TCoolBand (see page 3584) is maximized.
	TCustomDrawStage (see page 5445)	TCustomDrawStage specifies a stage in the custom drawing process.
	TCustomDrawTarget (see page 5446)	TCustomDrawTarget specifies the target of a custom drawing event.
	TDateTimeKind (see page 5447)	This is record ComCtrls.TDateTimeKind.
	TDisplayCode (see page 5447)	This is record ComCtrls.TDisplayCode.
	TDTCalAlignment (see page 5447)	This is record ComCtrls.TDTCalAlignment.
	TDTDateFormat (see page 5448)	This is record ComCtrls.TDTDateFormat.
	TDTDateMode (see page 5448)	This is record ComCtrls.TDTDateMode.
	THeaderSectionStyle (see page 5449)	This is record ComCtrls.THeaderSectionStyle.
	THeaderStyle (see page 5449)	THeaderStyle specifies the style of a header control.
	THitTest (see page 5450)	THitTest and THitTests (see page 5488) are used in the GetHitTestInfoAt method.
	THKInvalidKey (see page 5451)	This is record ComCtrls.THKInvalidKey.
	THKModifier (see page 5451)	This is record ComCtrls.THKModifier.
	TIconArrangement (see page 5452)	TIconArrangement indicates how the icons for items are arranged.
	TItemChange (see page 5452)	TItemChange describes the type of change a list item undergoes.
	TItemFind (see page 5453)	TItemFind identifies the type of search in an OnDataFind event handler.
	TItemRequests (see page 5454)	This is record ComCtrls.TItemRequests.
	TItemState (see page 5454)	TItemState and TItemStates (see page 5489) specify the state of an item in a list view control.
	TListArrangement (see page 5455)	This is record ComCtrls.TListArrangement.
	TListHotTrackStyle (see page 5456)	TListHotTrackStyle and TListHotTrackStyles (see page 5489) specify hot tracking styles for a list view control.
	TMultiSelectStyles (see page 5456)	This is record ComCtrls.TMultiSelectStyles.
	TNodeAttachMode (see page 5457)	TNodeAttachMode identifies a new or changed tree node relationship.
	TNodeState (see page 5459)	This is record ComCtrls.TNodeState.
	TNumberingStyle (see page 5459)	This is record ComCtrls.TNumberingStyle.
	TPageScrollerButton (see page 5460)	TPageScrollerButton identifies a button on a page scroller.
	TPageScrollerButtonState (see page 5460)	TPageScrollerButtonState indicates the state of a page scroller button.
	TPageScrollerOrientation (see page 5461)	TPageScrollerOrientation indicates the orientation of a page scroller control.
	TPositionToolTip (see page 5462)	Specifies the position of the tool tip.
	TProgressBarOrientation (see page 5462)	TProgressBarOrientation specifies the orientation of a progress bar.
	TSearchDirection (see page 5463)	Indicates the direction in which to search for the nearest item to a specified item or point.
	TSearchType (see page 5464)	Specifies search options.

	TSectionTrackState (see page 5464)	This is record ComCtrls.TSectionTrackState.
	TSortType (see page 5465)	TSortType indicates the criteria to use when sorting items.
	TStatusPanelBevel (see page 5466)	TStatusPanelBevel indicates the type of bevel around a panel in a status bar.
	TStatusPanelStyle (see page 5466)	TStatusPanelStyle indicates the kind of contents in a status bar panel.
	TTabPosition (see page 5467)	This is record ComCtrls.TTabPosition.
	TTabStyle (see page 5467)	TTabStyle indicates whether a TTabSet control is owner-drawn.
	TTBDrawingStyle (see page 5468)	Specifies the drawing style for the tool bar.
	TTickMark (see page 5468)	TTickMark indicates the location of the tick marks on a track bar.
	TTickStyle (see page 5469)	TTickStyle indicates how a track bar determines the increments at which tick marks appear.
	TToolButtonState (see page 5470)	This is record ComCtrls.TToolButtonState.
	TToolButtonStyle (see page 5470)	TToolButtonStyle indicates the type of control a TToolButton (see page 5107) object represents.
	TTrackBarOrientation (see page 5471)	TTrackBarOrientation indicates whether a trackbar is horizontal or vertical.
	TUDAlignButton (see page 5471)	This is record ComCtrls.TUDAlignButton.
	TUDBtnType (see page 5472)	This is record ComCtrls.TUDBtnType.
	TUDOrientation (see page 5472)	This is record ComCtrls.TUDOrientation.
	TUpDownDirection (see page 5473)	TUpDownDirection indicates the direction in which the position of an updown control changes.
	TViewStyle (see page 5473)	TViewStyle indicates how a list view control displays its items.

Records

	Name	Description
	TConversionFormat (see page 5444)	This is record ComCtrls.TConversionFormat.
	TNodeCache (see page 5457)	TNodeCache is the type for a cached tree-view node.
	TNodeDataInfo (see page 5458)	This is record ComCtrls.TNodeDataInfo.
	TNodeInfo (see page 5458)	This is record ComCtrls.TNodeInfo.
	TRichEditStreamInfo (see page 5463)	This is record ComCtrls.TRichEditStreamInfo.

27.3.1 ComCtrls.TAddMode Enumeration

Pascal

```
TAddMode = (
  taAddFirst,
  taAdd,
  taInsert
);
```

C++

```
enum TAddMode {
  taAddFirst,
  taAdd,
  taInsert
};
```

File

ComCtrls

Description

This is record ComCtrls.TAddMode.

27.3.2 ComCtrls.TAttributeType Enumeration

Pascal

```
TAttributeType = (
  atSelected,
  atDefaultText
);
```

C++

```
enum TAttributeType {
  atSelected,
  atDefaultText
};
```

File

ComCtrls

Description

This is record ComCtrls.TAttributeType.

27.3.3 ComCtrls.TAutoCompleteOption Enumeration

TAutoCompleteOption and TAutoCompleteOptions (see page 5480) specify a set of autocomplete options.

Pascal

```
TAutoCompleteOption = (
  acoAutoSuggest,
  acoAutoAppend,
  acoSearch,
  acoFilterPrefixes,
  acoUseTab,
  acoUpDownKeyDropsList,
  acoRtlReading
);
```

C++

```
enum TAutoCompleteOption {
  acoAutoSuggest,
  acoAutoAppend,
  acoSearch,
  acoFilterPrefixes,
  acoUseTab,
  acoUpDownKeyDropsList,
  acoRtlReading
};
```

File

ComCtrls

Description

TAutoCompleteOptions (see page 5480) defines options for controls that support autocompletion. These are:

Value	Meaning
acoAutoSuggest	Enable the autosuggest drop-down list.
acoAutoAppend	Enable autoappend.
acoSearch	Add a search item to the list of completed strings. Selecting this item launches a search engine.
acoFilterPrefixes	Don't match common prefixes, such as "www." and "http://".
acoUseTab	Use the tab key to select items.
acoUpDownKeyDropsList	Use up and down arrow keys to display suggestion list.
acoRtlReading	Text direction is opposite of parent.

27.3.4 ComCtrls.TCalDayOfWeek Enumeration

TCalDayOfWeek represents a day of the week.

Pascal

```
TCalDayOfWeek = (
  dowMonday,
  dowTuesday,
  dowWednesday,
  dowThursday,
  dowFriday,
  dowSaturday,
  dowSunday,
  dowLocaleDefault
);
```

C++

```
enum TCalDayOfWeek {
  dowMonday,
  dowTuesday,
  dowWednesday,
  dowThursday,
  dowFriday,
  dowSaturday,
  dowSunday,
  dowLocaleDefault
};
```

File

ComCtrls

Description

TCalDayOfWeek is used to represent a day of the week for a column in a calendar. It can be any of the following values:

Value	Meaning
dowMonday	Monday
dowTuesday	Tuesday
dowWednesday	Wednesday
dowThursday	Thursday

dowFriday	Friday
dowSaturday	Saturday
dowSunday	Sunday
dowLocaleDefault	The day of the week for the column is determined by the locale in which the application runs.

See Also

TCommonCalendar

27.3.5 ComCtrls.TComboBoxExStyle Enumeration

TComboBoxExStyle is the type for the Style property of extended combo box controls.

Pascal

```
TComboBoxExStyle = (
  csExDropDown,
  csExSimple,
  csExDropDownList
);
```

C++

```
enum TComboBoxExStyle {
  csExDropDown,
  csExSimple,
  csExDropDownList
};
```

File

ComCtrls

Description

These are the values in the TComboBoxExStyle type:

Value	Meaning
csExDropDown	Creates a drop-down list with an edit box for manually entered text.
csExSimple	Creates an edit box with a fixed list (list box) underneath. The length of the list is determined by the Height of the combo box.
csExDropDownList	Creates a drop-down list with no edit box; the user cannot enter text manually.

27.3.6 ComCtrls.TComboBoxExStyleEx Enumeration

TComboBoxExStyleEx and TComboBoxExStyles (see page 5481) configure the behavior of extended combo box controls.

Pascal

```
TComboBoxExStyleEx = (
  csExCaseSensitive,
  csExNoEditImage,
  csExNoEditImageIndent,
```

```
    csExNoSizeLimit,
    csExPathWordBreak
  );
```

C++

```
  enum TComboBoxExStyleEx {
    csExCaseSensitive,
    csExNoEditImage,
    csExNoEditImageIndent,
    csExNoSizeLimit,
    csExPathWordBreak
  };
```

File

ComCtrls

Description

TComboBoxExStyles (see page 5481) is a set of TComboBoxExStyleEx values. These are the values in the TComboBoxExStyleEx type:

Value	Meaning
csExCaseSensitive	Searches against item captions are case sensitive. This includes the automatic location of items in the list as the user types.
csExNoEditImage	The items in the list appear without any corresponding images (The combo box ignores their ImageIndex property).
csExNoEditImageIndent	The items in the list can't appear indented. (The combo box ignores their Indent property).
csExNoSizeLimit	The extended combo box can be resized vertically smaller than the edit region and drop-down button. When resized in this way, the combo box appears clipped.
csExPathWordBreak	The backslash (\), forward slash (/), and period (.) characters are treated as word delimiters for keyboard navigation purposes. This enables keyboard navigation of path names and URLs. (This style is not supported on Windows 9x).

27.3.7 ComCtrls.TCommonAVI Enumeration

Pascal

```
  TCommonAVI = (
    aviNone,
    aviFindFolder,
    aviFindFile,
    aviFindComputer,
    aviCopyFiles,
    aviCopyFile,
    aviRecycleFile,
    aviEmptyRecycle,
    aviDeleteFile
  );
```

C++

```
  enum TCommonAVI {
    aviNone,
    aviFindFolder,
    aviFindFile,
    aviFindComputer,
    aviCopyFiles,
```

```
    aviCopyFile,
    aviRecycleFile,
    aviEmptyRecycle,
    aviDeleteFile
  };
```

File

ComCtrls

Description

This is record ComCtrls.TCommonAVI.

27.3.8 ComCtrls.TConsistentAttribute Enumeration

Pascal

```
  TConsistentAttribute = (
    caBold,
    caColor,
    caFace,
    caItalic,
    caSize,
    caStrikeOut,
    caUnderline,
    caProtected
  );
```

C++

```
  enum TConsistentAttribute {
    caBold,
    caColor,
    caFace,
    caItalic,
    caSize,
    caStrikeOut,
    caUnderline,
    caProtected
  };
```

File

ComCtrls

Description

This is record ComCtrls.TConsistentAttribute.

27.3.9 ComCtrls.TConversionFormat Record

Pascal

```
  TConversionFormat = record
    ConversionClass: TConversionClass;
    Extension: string;
    Next: PConversionFormat;
  end;
```

C++

```
struct TConversionFormat {
  TConversionClass ConversionClass;
  AnsiString Extension;
  PConversionFormat Next;
};
```

File

ComCtrls

Description

This is record ComCtrls.TConversionFormat.

27.3.10 ComCtrls.TCoolBandMaximize Enumeration

TCoolBandMaximize specifies the conditions under which a TCoolBand (see page 3584) is maximized.

Pascal

```
TCoolBandMaximize = (
  bmNone,
  bmClick,
  bmDblClick
);
```

C++

```
enum TCoolBandMaximize {
  bmNone,
  bmClick,
  bmDblClick
};
```

File

ComCtrls

Description

TCoolBandMaximize specifies the conditions under which a TCoolBand (see page 3584) becomes maximized:

Value	Meaning
bmNone	The cool band is never maximized.
bmClick	The cool band is maximized when the user clicks with the left mouse button.
bmDblClick	The cool band is maximized when the user double-clicks with the left mouse button.

27.3.11 ComCtrls.TCustomDrawStage Enumeration

TCustomDrawStage specifies a stage in the custom drawing process.

Pascal

```
TCustomDrawStage = (
  cdPrePaint,
  cdPostPaint,
```

```
    cdPreErase,
    cdPostErase
  );
```

C++

```
  enum TCustomDrawStage {
    cdPrePaint,
    cdPostPaint,
    cdPreErase,
    cdPostErase
  };
```

File

ComCtrls

Description

Use TCustomDrawStage to determine when an advanced custom drawing event occurs:

Value	Meaning
cdPrePaint	Before painting.
cdPostPaint	After painting.
cdPreErase	Before erasing.
cdPostErase	After erasing.

27.3.12 ComCtrls.TCustomDrawTarget Enumeration

TCustomDrawTarget specifies the target of a custom drawing event.

Pascal

```
  TCustomDrawTarget = (
    dtControl,
    dtItem,
    dtSubItem
  );
```

C++

```
  enum TCustomDrawTarget {
    dtControl,
    dtItem,
    dtSubItem
  };
```

File

ComCtrls

Description

TCustomDrawTarget is used to indicate a specific type of custom drawing event. The following table lists the possible values:

Value	Meaning
dtControl	The event occurs when the entire control needs to be painted.
dtItem	The event occurs when an individual item within the control needs to be painted.
dtSubItem	The event occurs when a subitem needs to be painted.

27.3.13 ComCtrls.TDateTimeKind Enumeration

Pascal

```
TDateTimeKind = (
  dtkDate,
  dtkTime
);
```

C++

```
enum TDateTimeKind {
  dtkDate,
  dtkTime
};
```

File

ComCtrls

Description

This is record ComCtrls.TDateTimeKind.

27.3.14 ComCtrls.TDisplayCode Enumeration

Pascal

```
TDisplayCode = (
  drBounds,
  drIcon,
  drLabel,
  drSelectBounds
);
```

C++

```
enum TDisplayCode {
  drBounds,
  drIcon,
  drLabel,
  drSelectBounds
};
```

File

ComCtrls

Description

This is record ComCtrls.TDisplayCode.

27.3.15 ComCtrls.TDTCalAlignment Enumeration

Pascal

```
TDTCalAlignment = (
```

```
    dtaLeft,
    dtaRight
  );
```

C++

```
  enum TDTCalAlignment {
    dtaLeft,
    dtaRight
  };
```

File

ComCtrls

Description

This is record ComCtrls.TDTCalAlignment.

27.3.16 ComCtrls.TDTDateFormat Enumeration

Pascal

```
  TDTDateFormat = (
    dfShort,
    dfLong
  );
```

C++

```
  enum TDTDateFormat {
    dfShort,
    dfLong
  };
```

File

ComCtrls

Description

This is record ComCtrls.TDTDateFormat.

27.3.17 ComCtrls.TDTDateMode Enumeration

Pascal

```
  TDTDateMode = (
    dmComboBox,
    dmUpDown
  );
```

C++

```
  enum TDTDateMode {
    dmComboBox,
    dmUpDown
  };
```

File

ComCtrls

Description

This is record ComCtrls.TDTDateMode.

27.3.18 ComCtrls.THeaderSectionStyle Enumeration

Pascal

```
THeaderSectionStyle = (
  hsText,
  hsOwnerDraw
);
```

C++

```
enum THeaderSectionStyle {
  hsText,
  hsOwnerDraw
};
```

File

ComCtrls

Description

This is record ComCtrls.THeaderSectionStyle.

27.3.19 ComCtrls.THeaderStyle Enumeration

THeaderStyle specifies the style of a header control.

Pascal

```
THeaderStyle = (
  hsButtons,
  hsFlat
);
```

C++

```
enum THeaderStyle {
  hsButtons,
  hsFlat
};
```

File

ComCtrls

Description

THeaderStyle can have one of the following values:

Value	Meaning
hsButtons	The column sections are buttons.
hsFlat	The column sections are flat buttons.

27.3.20 ComCtrls.THitTest Enumeration

THitTest and THitTests (see page 5488) are used in the GetHitTestInfoAt method.

Pascal

```
THitTest = (
  htAbove,
  htBelow,
  htNowhere,
  htOnItem,
  htOnButton,
  htOnIcon,
  htOnIndent,
  htOnLabel,
  htOnRight,
  htOnStateIcon,
  htToLeft,
  htToRight
);
```

C++

```
enum THitTest {
  htAbove,
  htBelow,
  htNowhere,
  htOnItem,
  htOnButton,
  htOnIcon,
  htOnIndent,
  htOnLabel,
  htOnRight,
  htOnStateIcon,
  htToLeft,
  htToRight
};
```

File

ComCtrls

Description

THitTests (see page 5488) is a set of THitTest values. THitTest can have any of the following values:

Value	Means the point is
htAbove	Above the client area.
htBelow	Below the client area.
htNowhere	Inside the control, but not on an item.
htOnItem	On an item, its text, or its bitmap.
htOnButton	On a button.
htOnIcon	On an icon.
htOnIndent	On the indented area of an item.
htOnLabel	On a label.
htOnRight	On the right side of an item.

htOnStateIcon	On a state icon or bitmap associated with an item.
htToLeft	To the left of the client area.
htToRight	To the right of the client area.

27.3.21 ComCtrls.THKInvalidKey Enumeration

Pascal

```
THKInvalidKey = (
  hcNone,
  hcShift,
  hcCtrl,
  hcAlt,
  hcShiftCtrl,
  hcShiftAlt,
  hcCtrlAlt,
  hcShiftCtrlAlt
);
```

C++

```
enum THKInvalidKey {
  hcNone,
  hcShift,
  hcCtrl,
  hcAlt,
  hcShiftCtrl,
  hcShiftAlt,
  hcCtrlAlt,
  hcShiftCtrlAlt
};
```

File

ComCtrls

Description

This is record ComCtrls.THKInvalidKey.

27.3.22 ComCtrls.THKModifier Enumeration

Pascal

```
THKModifier = (
  hkShift,
  hkCtrl,
  hkAlt,
  hkExt
);
```

C++

```
enum THKModifier {
  hkShift,
  hkCtrl,
  hkAlt,
  hkExt
```

```
};
```

File

ComCtrls

Description

This is record ComCtrls.THKModifier.

27.3.23 ComCtrls.TIconArrangement Enumeration

TIconArrangement indicates how the icons for items are arranged.

Pascal

```
TIconArrangement = (
  iaTop,
  iaLeft
);
```

C++

```
enum TIconArrangement {
  iaTop,
  iaLeft
};
```

File

ComCtrls

Description

These are the possible TIconArrangement values:

Value	Meaning
iaTop	Items are aligned left to right in rows at the top of the view control
iaLeft	Items are aligned in columns top to bottom at the left of the view control

27.3.24 ComCtrls.TItemChange Enumeration

TItemChange describes the type of change a list item undergoes.

Pascal

```
TItemChange = (
  ctText,
  ctImage,
  ctState
);
```

C++

```
enum TItemChange {
  ctText,
  ctImage,
  ctState
};
```

File

ComCtrls

Description

TItemChange can have one of the following values:

Value	Meaning
ctText	A change to the Caption property of the list item.
ctImage	A change to the list item's ImageIndex property or the list view's image list.
ctState	A change to the list item's

See Also

TListItem

TListItem

TCustomViewItem

TListItem

TListItem

TCustomViewItem

TListItem

TCustomViewItem

TCustomViewItem

TCustomViewControl

TCustomListView

TCustomListview

27.3.25 ComCtrls.TItemFind Enumeration

TItemFind identifies the type of search in an OnDataFind event handler.

Pascal

```
TItemFind = (
  ifData,
  ifPartialString,
  ifExactString,
  ifNearest
);
```

C++

```
enum TItemFind {
  ifData,
  ifPartialString,
  ifExactString,
  ifNearest
};
```

File

ComCtrls

Description

TItemFind can have one of the following values:

Value	Meaning
ifData	Search for matching Data.
ifPartialString	Search for a partial string match.
ifExactString	Search for an exact string match.
ifNearest	Search for the nearest match.

27.3.26 ComCtrls.TItemRequests Enumeration

Pascal

```
TItemRequests = (
  irText,
  irImage,
  irParam,
  irState,
  irIndent
);
```

C++

```
enum TItemRequests {
  irText,
  irImage,
  irParam,
  irState,
  irIndent
};
```

File

ComCtrls

Description

This is record ComCtrls.TItemRequests.

27.3.27 ComCtrls.TItemState Enumeration

TItemState and TItemStates (see page 5489) specify the state of an item in a list view control.

Pascal

```
TItemState = (
  isNone,
  isCut,
  isDropHilited,
  isFocused,
  isSelected,
```

```
  isActivating
);
```

C++

```
enum TItemState {
  isNone,
  isCut,
  isDropHilited,
  isFocused,
  isSelected,
  isActivating
};
```

File

ComCtrls

Description

TItemStates (see page 5489) is a set of TItemState values. TItemState indicates the state of an item in a list view. It can have any of the following values:

Value	Meaning
isNone	Item is in its default state
isCut	Item is cut (for cut and paste operation)
isDropHilited	Item is highlighted as a drag-and-drop target
isFocused	Item is focused
isSelected	Item is selected
isActivating	Item is active

27.3.28 ComCtrls.TListArrangement Enumeration

Pascal

```
TListArrangement = (
  arAlignBottom,
  arAlignLeft,
  arAlignRight,
  arAlignTop,
  arDefault,
  arSnapToGrid
);
```

C++

```
enum TListArrangement {
  arAlignBottom,
  arAlignLeft,
  arAlignRight,
  arAlignTop,
  arDefault,
  arSnapToGrid
};
```

File

ComCtrls

Description

This is record ComCtrls.TListArrangement.

27.3.29 ComCtrls.TListHotTrackStyle Enumeration

TListHotTrackStyle and TListHotTrackStyles (see page 5489) specify hot tracking styles for a list view control.

Pascal

```
TListHotTrackStyle = (
  htHandPoint,
  htUnderlineCold,
  htUnderlineHot
);
```

C++

```
enum TListHotTrackStyle {
  htHandPoint,
  htUnderlineCold,
  htUnderlineHot
};
```

File

ComCtrls

Description

TListHotTrackStyles (see page 5489) describes how hot tracking occurs in a list view control. It can have any of the following values:

Value	Meaning
htHandPoint	The mouse cursor turns into a hand.
htUnderlineCold	An underline is displayed under all untracked items.
htUnderlineHot	An underline is displayed under the tracked item.

27.3.30 ComCtrls.TMultiSelectStyles Enumeration

Pascal

```
TMultiSelectStyles = (
  msControlSelect,
  msShiftSelect,
  msVisibleOnly,
  msSiblingOnly
);
```

C++

```
enum TMultiSelectStyles {
  msControlSelect,
  msShiftSelect,
  msVisibleOnly,
  msSiblingOnly
};
```

File

ComCtrls

Description

This is record ComCtrls.TMultiSelectStyles.

27.3.31 ComCtrls.TNodeAttachMode Enumeration

TNodeAttachMode identifies a new or changed tree node relationship.

Pascal

```
TNodeAttachMode = (
  naAdd,
  naAddFirst,
  naAddChild,
  naAddChildFirst,
  naInsert
);
```

C++

```
enum TNodeAttachMode {
  naAdd,
  naAddFirst,
  naAddChild,
  naAddChildFirst,
  naInsert
};
```

File

ComCtrls

Description

The TNodeAttachMode type specifies the way a new or relocated tree node will be related to some other node.

Value	Meaning
naAdd	The new or relocated node becomes the last sibling of the other node.
naAddFirst	The new or relocated node becomes the first sibling of the other node.
naInsert	The new or relocated node becomes the sibling immediately before the other node.
naAddChild	The new or relocated node becomes the last child of the other node.
naAddChildFirst	The new or relocated node becomes the first child of the other node.

27.3.32 ComCtrls.TNodeCache Record

TNodeCache is the type for a cached tree-view node.

Pascal

```
TNodeCache = record
  CacheNode: TTreeNode;
  CacheIndex: Integer;
```

```
end;
```

C++

```
struct TNodeCache {
  TTreeNode CacheNode;
  int CacheIndex;
};
```

File

ComCtrls

Description

Use TNodeCache to cache a tree-view node.

27.3.33 ComCtrls.TNodeDataInfo Record

Pascal

```
TNodeDataInfo = packed record
  ImageIndex: Integer;
  SelectedIndex: Integer;
  StateIndex: Integer;
  OverlayIndex: Integer;
  Data: Pointer;
  Count: Integer;
  TextLen: Byte;
end;
```

C++

```
struct TNodeDataInfo {
  int ImageIndex;
  int SelectedIndex;
  int StateIndex;
  int OverlayIndex;
  void * Data;
  int Count;
  Byte TextLen;
};
```

File

ComCtrls

Description

This is record ComCtrls.TNodeDataInfo.

27.3.34 ComCtrls.TNodeInfo Record

Pascal

```
TNodeInfo = packed record
  ImageIndex: Integer;
  SelectedIndex: Integer;
  StateIndex: Integer;
  OverlayIndex: Integer;
  Data: Pointer;
  Count: Integer;
```

```
    Text: string[255];
  end;
```

C++

```
  struct TNodeInfo {
    int ImageIndex;
    int SelectedIndex;
    int StateIndex;
    int OverlayIndex;
    void * Data;
    int Count;
    string[255] Text;
  };
```

File

ComCtrls

Description

This is record ComCtrls.TNodeInfo.

27.3.35 ComCtrls.TNodeState Enumeration

Pascal

```
  TNodeState = (
    nsCut,
    nsDropHilited,
    nsFocused,
    nsSelected,
    nsExpanded
  );
```

C++

```
  enum TNodeState {
    nsCut,
    nsDropHilited,
    nsFocused,
    nsSelected,
    nsExpanded
  };
```

File

ComCtrls

Description

This is record ComCtrls.TNodeState.

27.3.36 ComCtrls.TNumberingStyle Enumeration

Pascal

```
  TNumberingStyle = (
    nsNone,
    nsBullet
  );
```

C++

```
enum TNumberingStyle {
  nsNone,
  nsBullet
};
```

File

ComCtrls

Description

This is record ComCtrls.TNumberingStyle.

27.3.37 ComCtrls.TPageScrollerButton Enumeration

TPageScrollerButton identifies a button on a page scroller.

Pascal

```
TPageScrollerButton = (
  sbFirst,
  sbLast
);
```

C++

```
enum TPageScrollerButton {
  sbFirst,
  sbLast
};
```

File

ComCtrls

Description

TPageScrollerButton has one of the following values:

Value	Meaning
sbFirst	The top button (vertical orientation) or the left button (horizontal orientation).
sbLast	The bottom button (vertical orientation) or the right button (horizontal orientation).

27.3.38 ComCtrls.TPageScrollerButtonState Enumeration

TPageScrollerButtonState indicates the state of a page scroller button.

Pascal

```
TPageScrollerButtonState = (
  bsNormal,
  bsInvisible,
  bsGrayed,
  bsDepressed,
  bsHot
);
```

C++

```
enum TPageScrollerButtonState {
  bsNormal,
  bsInvisible,
  bsGrayed,
  bsDepressed,
  bsHot
};
```

File

ComCtrls

Description

TPageScrollerButtonState has one of the following values:

Value	Meaning
bsNormal	The button is in normal state.
bsInvisible	The button is not visible.
bsGrayed	The button is grayed.
bsDepressed	The button is pressed.
bsHot	The button is hot (the mouse is hovering over the button).

27.3.39 ComCtrls.TPageScrollerOrientation Enumeration

TPageScrollerOrientation indicates the orientation of a page scroller control.

Pascal

```
TPageScrollerOrientation = (
  soHorizontal,
  soVertical
);
```

C++

```
enum TPageScrollerOrientation {
  soHorizontal,
  soVertical
};
```

File

ComCtrls

Description

TPageScrollerOrientation has one of the following values:

Value	Meaning
soHorizontal	The page scroller is oriented horizontally.
soVertical	The page scroller is oriented vertically.

27.3.40 ComCtrls.TPositionToolTip Enumeration

Specifies the position of the tool tip.

Pascal

```
TPositionToolTip = (
  ptNone,
  ptTop,
  ptLeft,
  ptBottom,
  ptRight
);
```

C++

```
enum TPositionToolTip {
  ptNone,
  ptTop,
  ptLeft,
  ptBottom,
  ptRight
};
```

File

ComCtrls

Description

Set the position of the tool tip to one of the following values:

- ptNone
- ptTop
- ptLeft
- ptBottom
- ptRight

27.3.41 ComCtrls.TProgressBarOrientation Enumeration

TProgressBarOrientation specifies the orientation of a progress bar.

Pascal

```
TProgressBarOrientation = (
  pbHorizontal,
  pbVertical
);
```

C++

```
enum TProgressBarOrientation {
  pbHorizontal,
  pbVertical
};
```

File

ComCtrls

Description

TProgressBarOrientation has one of the following values:

Value	Meaning
pbHorizontal	The progress bar is oriented horizontally.
pbVertical	The progress bar is oriented vertically.

See Also

TProgressBar

27.3.42 ComCtrls.TRichEditStreamInfo Record

Pascal

```
TRichEditStreamInfo = record
  Converter: TConversion;
  Stream: TStream;
end;
```

C++

```
struct TRichEditStreamInfo {
  TConversion Converter;
  TStream Stream;
};
```

File

ComCtrls

Description

This is record ComCtrls.TRichEditStreamInfo.

27.3.43 ComCtrls.TSearchDirection Enumeration

Indicates the direction in which to search for the nearest item to a specified item or point.

Pascal

```
TSearchDirection = (
  sdLeft,
  sdRight,
  sdAbove,
  sdBelow,
  sdAll
);
```

C++

```
enum TSearchDirection {
  sdLeft,
  sdRight,
  sdAbove,
  sdBelow,
  sdAll
```

```
};
```

File

ComCtrls

Description

TSearchDirection defines the following search directions:

Value	Meaning
sdLeft	Look to the left of a specified item.
sdRight	Look to the right of a specified item.
sdAbove	Look above a specified item.
sdBelow	Look below a specified item.
sdAll	Look in the order that items appear in the Items property.

27.3.44 ComCtrls.TSearchType Enumeration

Specifies search options.

Pascal

```
TSearchType = (
  stWholeWord,
  stMatchCase
);
```

C++

```
enum TSearchType {
  stWholeWord,
  stMatchCase
};
```

File

ComCtrls

Description

The available options are shown in the following table.

Option	Description
stWholeWord	Specifies that the search should find the whole word only, rather than picking out partial search results. For example, if you search for *foo* with this options set, the search result would not include *foobar*.
stMatchCase	Specifies that the search results must match the letter casing of the search term.

27.3.45 ComCtrls.TSectionTrackState Enumeration

Pascal

```
TSectionTrackState = (
  tsTrackBegin,
```

```
    tsTrackMove,
    tsTrackEnd
  );
```

C++

```
  enum TSectionTrackState {
    tsTrackBegin,
    tsTrackMove,
    tsTrackEnd
  };
```

File

ComCtrls

Description

This is record ComCtrls.TSectionTrackState.

27.3.46 ComCtrls.TSortType Enumeration

TSortType indicates the criteria to use when sorting items.

Pascal

```
  TSortType = (
    stNone,
    stData,
    stText,
    stBoth
  );
```

C++

```
  enum TSortType {
    stNone,
    stData,
    stText,
    stBoth
  };
```

File

ComCtrls

Description

Value	Meaning
stNone	No sorting is done.
stData	Items are sorted based on their Data property. An event handler must be provided to compare Data values.
stText	Items are sorted based on their Caption property.
stBoth	Items are sorted based on both the Data and Caption. The Data values are ignored unless an event handler is provided to compare them.

27.3.47 ComCtrls.TStatusPanelBevel Enumeration

TStatusPanelBevel indicates the type of bevel around a panel in a status bar.

Pascal

```
TStatusPanelBevel = (
  pbNone,
  pbLowered,
  pbRaised
);
```

C++

```
enum TStatusPanelBevel {
  pbNone,
  pbLowered,
  pbRaised
};
```

File

ComCtrls

Description

These are the possible TStatusPanelBevel values:

Value	Meaning
pbNone	The panel has no bevel and appears flat.
pbLowered	The panel appears recessed.
pbRaised	The panel appears raised.

27.3.48 ComCtrls.TStatusPanelStyle Enumeration

TStatusPanelStyle indicates the kind of contents in a status bar panel.

Pascal

```
TStatusPanelStyle = (
  psText,
  psOwnerDraw
);
```

C++

```
enum TStatusPanelStyle {
  psText,
  psOwnerDraw
};
```

File

ComCtrls

Description

TStatusPanelStyle enumerates the possible values of the Style property in TStatusPanel (see page 4825):

Value	Meaning
psText	The panel shows the value of the Text property
psOwnerDraw	The panel is drawn by the OnDrawPanel (TStatusBar (see page 4769)) event handler.

See Also

TStatusPanel

TStatusPanel

TStatusBar

27.3.49 ComCtrls.TTabPosition Enumeration

Pascal

```
TTabPosition = (
  tpTop,
  tpBottom,
  tpLeft,
  tpRight
);
```

C++

```
enum TTabPosition {
  tpTop,
  tpBottom,
  tpLeft,
  tpRight
};
```

File

ComCtrls

Description

This is record ComCtrls.TTabPosition.

27.3.50 ComCtrls.TTabStyle Enumeration

TTabStyle indicates whether a TTabSet control is owner-drawn.

Pascal

```
TTabStyle = (
  tsTabs,
  tsButtons,
  tsFlatButtons
);
```

C++

```
enum TTabStyle {
  tsTabs,
  tsButtons,
  tsFlatButtons
```

```
};
```

File

ComCtrls

Description

These are the possible values:

Value	Meaning
tsStandard	Each tab has the standard size and look.
tsOwnerDraw	Each tab has the height specified with the TabHeight property and width needed to hold the text or glyph.

27.3.51 ComCtrls.TTBDrawingStyle Enumeration

Specifies the drawing style for the tool bar.

Pascal

```
TTBDrawingStyle = (
  dsNormal,
  dsGradient
);
```

C++

```
enum TTBDrawingStyle {
  dsNormal,
  dsGradient
};
```

File

ComCtrls

Description

This type allows you to specify one of the following values:

- dsNormal
- dsGradient

27.3.52 ComCtrls.TTickMark Enumeration

TTickMark indicates the location of the tick marks on a track bar.

Pascal

```
TTickMark = (
  tmBottomRight,
  tmTopLeft,
  tmBoth
);
```

C++

```
enum TTickMark {
```

```
  tmBottomRight,
  tmTopLeft,
  tmBoth
};
```

File

ComCtrls

Description

These are the possible values of TTickMark:

Value	Meaning
tmBottomRight	Tick marks are on the bottom or right of the track bar, depending of the value of Orientation.
tmTopLeft	Tick marks are on the top or left of the track bar, depending of the value of Orientation.
tmBoth	Tick marks are on both sides of the track bar.

27.3.53 ComCtrls.TTickStyle Enumeration

TTickStyle indicates how a track bar determines the increments at which tick marks appear.

Pascal

```
TTickStyle = (
  tsNone,
  tsAuto,
  tsManual
);
```

C++

```
enum TTickStyle {
  tsNone,
  tsAuto,
  tsManual
};
```

File

ComCtrls

Description

TTickStyle has these possible values:

Value	Meaning
tsAuto	Tick marks are automatically displayed at increments equal to the value of the Frequency property.
tsManual	Tick marks are displayed at the Min and Max values. Additional tick marks can be set using the SetTick method.
tsNone	No tick marks are displayed.

27.3.54 ComCtrls.TToolButtonState Enumeration

Pascal

```
TToolButtonState = (
  tbsChecked,
  tbsPressed,
  tbsEnabled,
  tbsHidden,
  tbsIndeterminate,
  tbsWrap,
  tbsEllipses,
  tbsMarked
);
```

C++

```
enum TToolButtonState {
  tbsChecked,
  tbsPressed,
  tbsEnabled,
  tbsHidden,
  tbsIndeterminate,
  tbsWrap,
  tbsEllipses,
  tbsMarked
};
```

File

ComCtrls

Description

This is record ComCtrls.TToolButtonState.

27.3.55 ComCtrls.TToolButtonStyle Enumeration

TToolButtonStyle indicates the type of control a TToolButton (see page 5107) object represents.

Pascal

```
TToolButtonStyle = (
  tbsButton,
  tbsCheck,
  tbsDropDown,
  tbsSeparator,
  tbsDivider,
  tbsTextButton
);
```

C++

```
enum TToolButtonStyle {
  tbsButton,
  tbsCheck,
  tbsDropDown,
  tbsSeparator,
  tbsDivider,
  tbsTextButton
};
```

File

ComCtrls

Description

These are the possible values of TToolButtonStyle:

Value	Meaning
tbsButton	The button appears and functions like a normal speed button.
tbsCheck	Clicking the button toggles the Down property. Once selected, the button remains selected until clicked again.
tbsDropDown	The button displays a downward-pointing arrow (suitable for accessing a drop-down menu).
tbsSeparator	The button appears as an empty space on the toolbar (used to separate other controls).
tbsDivider	The button appears as a vertical line on the toolbar (used to separate other controls).

27.3.56 ComCtrls.TTrackBarOrientation Enumeration

TTrackBarOrientation indicates whether a trackbar is horizontal or vertical.

Pascal

```
TTrackBarOrientation = (
  trHorizontal,
  trVertical
);
```

C++

```
enum TTrackBarOrientation {
  trHorizontal,
  trVertical
};
```

File

ComCtrls

Description

These are the possible values of TTrackBarOrientation:

Value	Meaning
trHorizontal	The long edge of the track is parallel to the top of the form. The Min position is on the left and the Max position is on the right.
trVertical	The long edge of the track is parallel to the side of the form. The Min position is on the top and the Max position is on the bottom.

27.3.57 ComCtrls.TUDAlignButton Enumeration

Pascal

```
TUDAlignButton = (
```

```
    udLeft,
    udRight
  );
```

C++

```
  enum TUDAlignButton {
    udLeft,
    udRight
  };
```

File

ComCtrls

Description

This is record ComCtrls.TUDAlignButton.

27.3.58 ComCtrls.TUDBtnType Enumeration

Pascal

```
  TUDBtnType = (
    btNext,
    btPrev
  );
```

C++

```
  enum TUDBtnType {
    btNext,
    btPrev
  };
```

File

ComCtrls

Description

This is record ComCtrls.TUDBtnType.

27.3.59 ComCtrls.TUDOrientation Enumeration

Pascal

```
  TUDOrientation = (
    udHorizontal,
    udVertical
  );
```

C++

```
  enum TUDOrientation {
    udHorizontal,
    udVertical
  };
```

File

ComCtrls

Description

This is record ComCtrls.TUDOrientation.

27.3.60 ComCtrls.TUpDownDirection Enumeration

TUpDownDirection indicates the direction in which the position of an updown control changes.

Pascal

```
TUpDownDirection = (
  updNone,
  updUp,
  updDown
);
```

C++

```
enum TUpDownDirection {
  updNone,
  updUp,
  updDown
};
```

File

ComCtrls

Description

The following table lists the possible values:

Value	Description
updNone	The position is changing to an illegal value (less than the minimum value or greater than the maximum value).
updUp	The position is increasing.
updDown	The position is decreasing.

27.3.61 ComCtrls.TViewStyle Enumeration

TViewStyle indicates how a list view control displays its items.

Pascal

```
TViewStyle = (
  vsIcon,
  vsSmallIcon,
  vsList,
  vsReport
);
```

C++

```
enum TViewStyle {
  vsIcon,
  vsSmallIcon,
  vsList,
```

```
  vsReport
};
```

File

ComCtrls

Description

TViewStyle can have one of the following values:

Value	Meaning
vsIcon	Each item appears as a full-sized icon with a label below it. The user can drag the items to any location in the list view window.
vsSmallIcon	Each item appears as a small icon with the label to the right of it. The user can drag the items to any location.
vsList	Each item appears as a small icon with a label to the right of it. Items are arranged in columns and cannot be dragged by the user.
vsReport	Each item appears on its own line with information arranged in columns. The leftmost column contains the small icon and label, and subsequent columns contain subitems as specified by the application. Unless ShowColumnHeaders is false, each column has a header.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
```

```
  ComboBox1->Items->AddObject("vsIcon", (TObject *)vsIcon);
  ComboBox1->Items->AddObject("vsList", (TObject *)vsList);
  ComboBox1->Items->AddObject("vsReport", (TObject *)vsReport);
  ComboBox1->Items->AddObject("vsSmallIcon", (TObject *)vsSmallIcon);
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := 'Image' + IntToStr(I);
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
    NewColumn.Width := 200;
    // Add View styles and constants to the Combo Box
    ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
    ComboBox1.Items.AddObject('vsList', TObject(vsList));
    ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
    ComboBox1.Items.AddObject(
      'vsSmallIcon', TObject(vsSmallIcon));
    // Display first item in the Combo Box
```

```
      ComboBox1.ItemIndex := 0;
    end;
  end;

  procedure TComboForm.ComboBox1Click(Sender: TObject);
  begin
    with ComboBox1 do
      ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

  end;
```

27.4 Types

The following table lists types in this documentation.

Types

Name	Description
PConversionFormat (see page 5479)	This is type ComCtrls.PConversionFormat.
PNodeCache (see page 5479)	PNodeCache is a pointer to a TNodeCache (see page 5457) object.
PNodeInfo (see page 5480)	This is type ComCtrls.PNodeInfo.
PRichEditStreamInfo (see page 5480)	This is type ComCtrls.PRichEditStreamInfo.
TAutoCompleteOptions (see page 5480)	TAutoCompleteOptions specify options that control autocompletion.
TComboBoxExStyles (see page 5481)	TComboBoxExStyles is the type for the Style property of extended combo box controls.
TComboExItemClass (see page 5481)	This is type ComCtrls.TComboExItemClass.
TComboExItemsClass (see page 5482)	This is type ComCtrls.TComboExItemsClass.
TConsistentAttributes (see page 5482)	This is type ComCtrls.TConsistentAttributes.
TConversionClass (see page 5482)	This is type ComCtrls.TConversionClass.
TCustomDrawPanelEvent (see page 5483)	TCustomDrawPanelEvent is the type of the OnDrawPanel event handler.
TCustomDrawSectionEvent (see page 5483)	This is type ComCtrls.TCustomDrawSectionEvent.
TCustomDrawState (see page 5483)	TCustomDrawState specifies a custom drawing state.
TCustomHCCreateSectionClassEvent (see page 5484)	This is type ComCtrls.TCustomHCCreateSectionClassEvent.
TCustomSectionNotifyEvent (see page 5484)	This is type ComCtrls.TCustomSectionNotifyEvent.
TCustomSectionTrackEvent (see page 5485)	This is type ComCtrls.TCustomSectionTrackEvent.
TDateTimeColors (see page 5485)	TDateTimeColors specifies colors used in a calendar control.
TDrawPanelEvent (see page 5485)	This is type ComCtrls.TDrawPanelEvent.
TDrawSectionEvent (see page 5486)	This is type ComCtrls.TDrawSectionEvent.
TDrawTabEvent (see page 5486)	TDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet control.
TDTParseInputEvent (see page 5487)	This is type ComCtrls.TDTParseInputEvent.
THCCreateSectionClassEvent (see page 5487)	This is type ComCtrls.THCCreateSectionClassEvent.
THeaderSectionClass (see page 5487)	This is type ComCtrls.THeaderSectionClass.
THitTests (see page 5488)	This is type ComCtrls.THitTests.
THKInvalidKeys (see page 5488)	This is type ComCtrls.THKInvalidKeys.
THKModifiers (see page 5488)	This is type ComCtrls.THKModifiers.
TItemRequest (see page 5489)	This is type ComCtrls.TItemRequest.
TItemStates (see page 5489)	TItemState (see page 5454) and TItemStates specify the state of an item in a list view control.
TListHotTrackStyles (see page 5489)	TListHotTrackStyle (see page 5456) and TListHotTrackStyles specify hot tracking styles for a list view control.
TListItemClass (see page 5490)	This is type ComCtrls.TListItemClass.
TLVAdvancedCustomDrawEvent (see page 5490)	This is type ComCtrls.TLVAdvancedCustomDrawEvent.
TLVAdvancedCustomDrawItemEvent (see page 5491)	This is type ComCtrls.TLVAdvancedCustomDrawItemEvent.
TLVAdvancedCustomDrawSubItemEvent (see page 5491)	This is type ComCtrls.TLVAdvancedCustomDrawSubItemEvent.
TLVChangeEvent (see page 5491)	This is type ComCtrls.TLVChangeEvent.
TLVChangingEvent (see page 5492)	This is type ComCtrls.TLVChangingEvent.
TLVColumnClickEvent (see page 5492)	This is type ComCtrls.TLVColumnClickEvent.
TLVColumnRClickEvent (see page 5492)	This is type ComCtrls.TLVColumnRClickEvent.
TLVCompareEvent (see page 5493)	This is type ComCtrls.TLVCompareEvent.
TLVCreateItemClassEvent (see page 5493)	This is type ComCtrls.TLVCreateItemClassEvent.
TLVCustomDrawEvent (see page 5493)	This is type ComCtrls.TLVCustomDrawEvent.
TLVCustomDrawItemEvent (see page 5494)	This is type ComCtrls.TLVCustomDrawItemEvent.
TLVCustomDrawSubItemEvent (see page 5494)	This is type ComCtrls.TLVCustomDrawSubItemEvent.

TLVDeletedEvent (see page 5494)	TLVDeletedEvent is the type of a list view's OnDeletion event handler.
TLVDrawItemEvent (see page 5495)	TLVDrawItemEvent is the type of the list view OnCustomDrawItem and OnDrawItem event handlers.
TLVEditedEvent (see page 5495)	This is type ComCtrls.TLVEditedEvent.
TLVEditingEvent (see page 5496)	This is type ComCtrls.TLVEditingEvent.
TLVInfoTipEvent (see page 5496)	This is type ComCtrls.TLVInfoTipEvent.
TLVNotifyEvent (see page 5496)	This is type ComCtrls.TLVNotifyEvent.
TLVOwnerDataEvent (see page 5497)	TLVOwnerDataEvent is the type of a list view's OnData event handler.
TLVOwnerDataFindEvent (see page 5497)	TLVOwnerDataFindEvent is the type of a list view's OnDataFind event handler.
TLVOwnerDataHintEvent (see page 5498)	TLVOwnerDataHintEvent is the type of a list view's OnDataHint event handler.
TLVOwnerDataStateChangeEvent (see page 5498)	TLVOwnerDataStateChangeEvent is the type of a list view's OnDataStateChange event handler.
TLVSelectItemEvent (see page 5499)	This is type ComCtrls.TLVSelectItemEvent.
TLVSubItemImageEvent (see page 5499)	This is type ComCtrls.TLVSubItemImageEvent.
TMultiSelectStyle (see page 5499)	This is type ComCtrls.TMultiSelectStyle.
TOnGetMonthInfoEvent (see page 5500)	TOnGetMonthInfoEvent is the type of the TCommonCalendar.OnGetMonthInfo event handler.
TOwnerDrawState (see page 5500)	Indicates state information that can influence how an item is drawn.
TPageScrollEvent (see page 5501)	TPageScrollEvent is the type of a page scroller's OnScroll event handler.
TProgressRange (see page 5502)	This is type ComCtrls.TProgressRange.
TRichEditProtectChange (see page 5502)	This is type ComCtrls.TRichEditProtectChange.
TRichEditResizeEvent (see page 5502)	This is type ComCtrls.TRichEditResizeEvent.
TRichEditSaveClipboard (see page 5503)	This is type ComCtrls.TRichEditSaveClipboard.
TSBCreatePanelClassEvent (see page 5503)	TSBCreatePanelClassEvent is the type of the OnCreatePanelClass event handler.
TSearchTypes (see page 5503)	Set of TSearchType (see page 5464).
TSectionDragEvent (see page 5505)	TSectionDragEvent is the type for event handlers that respond when the user drags a section in a header control.
TSectionNotifyEvent (see page 5506)	This is type ComCtrls.TSectionNotifyEvent.
TSectionTrackEvent (see page 5506)	This is type ComCtrls.TSectionTrackEvent.
TStatusPanelClass (see page 5507)	TStatusPanelClass defines the metaclass for TStatusPanel (see page 4825).
TTabChangingEvent (see page 5507)	This is type ComCtrls.TTabChangingEvent.
TTabGetImageEvent (see page 5507)	This is type ComCtrls.TTabGetImageEvent.
TTBAdvancedCustomDrawBtnEvent (see page 5508)	This is type ComCtrls.TTBAdvancedCustomDrawBtnEvent.
TTBAdvancedCustomDrawEvent (see page 5508)	This is type ComCtrls.TTBAdvancedCustomDrawEvent.
TTBButtonEvent (see page 5508)	TTBButtonEvent is the type of event handlers that respond when the user adds or deletes a tool button in the toolbar customize dialog.
TTBCustomDrawBtnEvent (see page 5509)	This is type ComCtrls.TTBCustomDrawBtnEvent.
TTBCustomDrawEvent (see page 5509)	This is type ComCtrls.TTBCustomDrawEvent.
TTBCustomDrawFlags (see page 5509)	TTBCustomDrawFlags is the type of the OnAdvancedCustomDrawButton event handler.
TTBCustomizeQueryEvent (see page 5510)	TTBCustomizeQueryEvent is the type of event handlers that allow a toolbar to decide whether to allow a user customization.
TTBGradientDrawingOptions (see page 5511)	Specifies the style of the gradient.
TTBNewButtonEvent (see page 5511)	This is type ComCtrls.TTBNewButtonEvent.
TToolButtonActionLinkClass (see page 5511)	This is type ComCtrls.TToolButtonActionLinkClass.
TTreeNodeClass (see page 5512)	This is type ComCtrls.TTreeNodeClass.
TTVAdvancedCustomDrawEvent (see page 5512)	Used in the TCustomTreeView.OnAdvancedCustomDraw event.
TTVAdvancedCustomDrawItemEvent (see page 5513)	Used in the TCustomTreeView.OnAdvancedCustomDrawItem event.
TTVChangedEvent (see page 5513)	Used in the TCustomTreeView.OnChange event.
TTVChangingEvent (see page 5514)	Used in the TCustomTreeView.OnChanging event.
TTVCollapsingEvent (see page 5514)	Used in the TCustomTreeView.OnCollapsing event.
TTVCompareEvent (see page 5515)	Used in the TCustomTreeView.OnCompare event.
TTVCreateNodeClassEvent (see page 5516)	This is type ComCtrls.TTVCreateNodeClassEvent.
TTVCustomDrawEvent (see page 5516)	Used in the TCustomTreeView.OnCustomDraw event.
TTVCustomDrawItemEvent (see page 5516)	Used in the TCustomTreeView.OnCustomDrawItem event.
TTVEditedEvent (see page 5517)	Used in the TCustomTreeView.OnEdited event.

TTVEditingEvent (see page 5518)	Used in the TCustomTreeView.OnEditing event.
TTVExpandedEvent (see page 5518)	Represents the event handler for events that notify a tree view of changes to one of its nodes.
TTVExpandingEvent (see page 5519)	Used in the TCustomTreeView.OnExpanding event.
TUDChangingEvent (see page 5519)	This is type ComCtrls.TUDChangingEvent.
TUDChangingEventEx (see page 5520)	This is type ComCtrls.TUDChangingEventEx.
TUDClickEvent (see page 5520)	This is type ComCtrls.TUDClickEvent.
TWidth (see page 5520)	This is type ComCtrls.TWidth.

27.4.1 ComCtrls.PConversionFormat Type

Pascal

```
PConversionFormat = ^TConversionFormat;
```

C++

```
^TConversionFormat PConversionFormat;
```

File

ComCtrls

Description

This is type ComCtrls.PConversionFormat.

27.4.2 ComCtrls.PNodeCache Type

PNodeCache is a pointer to a TNodeCache (see page 5457) object.

Pascal

```
PNodeCache = ^TNodeCache;
```

C++

```
^TNodeCache PNodeCache;
```

File

ComCtrls

Description

PNodeCache is a pointer to a cached node of an instance of the TTreeView (see page 5293) class defined in the ComCtrls (see page 3413) unit.

See Also

TTreeView

TTreeNodes

27.4.3 ComCtrls.PNodeInfo Type

Pascal

```
PNodeInfo = ^TNodeInfo;
```

C++

```
^TNodeInfo PNodeInfo;
```

File

ComCtrls

Description

This is type ComCtrls.PNodeInfo.

27.4.4 ComCtrls.PRichEditStreamInfo Type

Pascal

```
PRichEditStreamInfo = ^TRichEditStreamInfo;
```

C++

```
^TRichEditStreamInfo PRichEditStreamInfo;
```

File

ComCtrls

Description

This is type ComCtrls.PRichEditStreamInfo.

27.4.5 ComCtrls.TAutoCompleteOptions Type

TAutoCompleteOptions specify options that control autocompletion.

Pascal

```
TAutoCompleteOptions = set of TAutoCompleteOption;
```

C++

```
TAutoCompleteOption TAutoCompleteOptions;
```

File

ComCtrls

Description

TAutoCompleteOptions defines options for controls that support autocompletion. These are:

Value	Meaning
acoAutoSuggest	Enable the autosuggest drop-down list.

acoAutoAppend	Enable autoappend.
acoSearch	Add a search item to the list of completed strings. Selecting this item launches a search engine.
acoFilterPrefixes	Don't match common prefixes, such as "www." and "http://".
acoUseTab	Use the tab key to select items.
acoUpDownKeyDropsList	Use up and down arrow keys to display suggestion list.
acoRtlReading	Text direction is opposite of parent.

27.4.6 ComCtrls.TComboBoxExStyles Type

TComboBoxExStyles is the type for the Style property of extended combo box controls.

Pascal

```
TComboBoxExStyles = set of TComboBoxExStyleEx;
```

C++

```
TComboBoxExStyleEx TComboBoxExStyles;
```

File

ComCtrls

Description

These are the values in the TComboBoxExStyles type:

Value	Meaning
csExDropDown	Creates a drop-down list with an edit box for manually entered text.
csExSimple	Creates an edit box with a fixed list (list box) underneath. The length of the list is determined by the Height of the combo box.
csExDropDownList	Creates a drop-down list with no edit box; the user cannot enter text manually.

27.4.7 ComCtrls.TComboExItemClass Type

Pascal

```
TComboExItemClass = class of TComboExItem;
```

C++

```
TComboExItem TComboExItemClass;
```

File

ComCtrls

Description

This is type ComCtrls.TComboExItemClass.

27.4.8 ComCtrls.TComboExItemsClass Type

Pascal

```
TComboExItemsClass = class of TComboExItems;
```

C++

```
TComboExItems TComboExItemsClass;
```

File

ComCtrls

Description

This is type ComCtrls.TComboExItemsClass.

27.4.9 ComCtrls.TConsistentAttributes Type

Pascal

```
TConsistentAttributes = set of TConsistentAttribute;
```

C++

```
TConsistentAttribute TConsistentAttributes;
```

File

ComCtrls

Description

This is type ComCtrls.TConsistentAttributes.

27.4.10 ComCtrls.TConversionClass Type

Pascal

```
TConversionClass = class of TConversion;
```

C++

```
TConversion TConversionClass;
```

File

ComCtrls

Description

This is type ComCtrls.TConversionClass.

27.4.11 ComCtrls.TCustomDrawPanelEvent Type

TCustomDrawPanelEvent is the type of the OnDrawPanel event handler.

Pascal

```
TCustomDrawPanelEvent = procedure (StatusBar: TCustomStatusBar; Panel: TStatusPanel; const
Rect: TRect) of object;
```

C++

```
(StatusBar: TCustomStatusBar; Panel: TStatusPanel; const Rect: TRect) (
TCustomDrawPanelEvent)();
```

File

ComCtrls

Description

StatusBar is the status bar object that contains the panel.

Panel is the panel that needs to be redrawn.

Rect gives the (new) dimensions of the status panel.

27.4.12 ComCtrls.TCustomDrawSectionEvent Type

Pascal

```
TCustomDrawSectionEvent = procedure (HeaderControl: TCustomHeaderControl; Section:
THeaderSection; const Rect: TRect; Pressed: Boolean) of object;
```

C++

```
(HeaderControl: TCustomHeaderControl; Section: THeaderSection; const Rect: TRect; Pressed:
Boolean) ( TCustomDrawSectionEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TCustomDrawSectionEvent.

27.4.13 ComCtrls.TCustomDrawState Type

TCustomDrawState specifies a custom drawing state.

Pascal

```
TCustomDrawState = set of (cdsSelected, cdsGrayed, cdsDisabled, cdsChecked, cdsFocused,
cdsDefault, cdsHot, cdsMarked, cdsIndeterminate);
```

C++

```
(cdsSelected, cdsGrayed, cdsDisabled, cdsChecked, cdsFocused, cdsDefault, cdsHot, cdsMarked,
```

```
cdsIndeterminate) TCustomDrawState;
```

File

ComCtrls

Description

TCustomDrawState indicates state information that can affect how a control is drawn. It includes a set drawn from the following values:

Value	Meaning
cdsSelected	The control or item is selected.
cdsGrayed	The control or item should appear grayed.
cdsDisabled	The control or item is disabled.
cdsChecked	The item should appear checked.
cdsFocused	The control or item has input focus.
cdsDefault	The item is the default item.
cdsHot	The item is under the mouse and the HotTrack property is True.
cdsMarked	The item is marked (marked items appear shaded by default).
cdsIndeterminate	The item is neither selected nor unselected.

27.4.14 ComCtrls.TCustomHCCreateSectionClassEvent Type

Pascal

```
TCustomHCCreateSectionClassEvent = procedure (Sender: TCustomHeaderControl; var SectionClass:
THeaderSectionClass) of object;
```

C++

```
(Sender: TCustomHeaderControl; var SectionClass: THeaderSectionClass) (
TCustomHCCreateSectionClassEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TCustomHCCreateSectionClassEvent.

27.4.15 ComCtrls.TCustomSectionNotifyEvent Type

Pascal

```
TCustomSectionNotifyEvent = procedure (HeaderControl: TCustomHeaderControl; Section:
THeaderSection) of object;
```

C++

```
(HeaderControl: TCustomHeaderControl; Section: THeaderSection) ( TCustomSectionNotifyEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TCustomSectionNotifyEvent.

27.4.16 ComCtrls.TCustomSectionTrackEvent Type

Pascal

```
TCustomSectionTrackEvent = procedure (HeaderControl: TCustomHeaderControl; Section:
THeaderSection; Width: Integer; State: TSectionTrackState) of object;
```

C++

```
(HeaderControl: TCustomHeaderControl; Section: THeaderSection; Width: Integer; State:
TSectionTrackState) ( TCustomSectionTrackEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TCustomSectionTrackEvent.

27.4.17 ComCtrls.TDateTimeColors Type

TDateTimeColors specifies colors used in a calendar control.

Pascal

```
TDateTimeColors = TMonthCalColors;
```

C++

```
TMonthCalColors TDateTimeColors;
```

File

ComCtrls

Description

TDateTimeColors is provided for backward compatibility on the TDateTimePicker (see page 4049) object.

See Also

TdateTimepicker

27.4.18 ComCtrls.TDrawPanelEvent Type

Pascal

```
TDrawPanelEvent = procedure (StatusBar: TStatusBar; Panel: TStatusPanel; const Rect: TRect) of
object;
```

C++

```
(StatusBar: TStatusBar; Panel: TStatusPanel; const Rect: TRect) ( TDrawPanelEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TDrawPanelEvent.

27.4.19 ComCtrls.TDrawSectionEvent Type

Pascal

```
TDrawSectionEvent = procedure (HeaderControl: THeaderControl; Section: THeaderSection; const
Rect: TRect; Pressed: Boolean) of object;
```

C++

```
(HeaderControl: THeaderControl; Section: THeaderSection; const Rect: TRect; Pressed: Boolean)
( TDrawSectionEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TDrawSectionEvent.

27.4.20 ComCtrls.TDrawTabEvent Type

TDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet control.

Pascal

```
TDrawTabEvent = procedure (Control: TCustomTabControl; TabIndex: Integer; const Rect: TRect;
Active: Boolean) of object;
```

C++

```
(Control: TCustomTabControl; TabIndex: Integer; const Rect: TRect; Active: Boolean) (
TDrawTabEvent)();
```

File

ComCtrls

Description

Control is the tab control on which a tab is to be drawn.

TabIndex is the position of the tab that is to be drawn.

Rect is the area in which the tab is to be drawn.

Active indicates whether the tab is active.

Note: The TDrawTabEvent type defined in the tabs unit for the OnDrawTab event handler differs from the TDrawTabEvent type defined in the comctrls unit. Do not use the comctrls unit in source files where you are working with the tab control.

27.4.21 ComCtrls.TDTParseInputEvent Type

Pascal

```
TDTParseInputEvent = procedure (Sender: TObject; const UserString: string; var DateAndTime:
TDateTime; var AllowChange: Boolean) of object;
```

C++

```
(Sender: TObject; const UserString: string; var DateAndTime: TDateTime; var AllowChange:
Boolean) ( TDTParseInputEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TDTParseInputEvent.

27.4.22 ComCtrls.THCCreateSectionClassEvent Type

Pascal

```
THCCreateSectionClassEvent = procedure (Sender: THeaderControl; var SectionClass:
THeaderSectionClass) of object;
```

C++

```
(Sender: THeaderControl; var SectionClass: THeaderSectionClass) (
THCCreateSectionClassEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.THCCreateSectionClassEvent.

27.4.23 ComCtrls.THeaderSectionClass Type

Pascal

```
THeaderSectionClass = class of THeaderSection;
```

C++

```
THeaderSection THeaderSectionClass;
```

File

ComCtrls

Description

This is type ComCtrls.THeaderSectionClass.

27.4.24 ComCtrls.THitTests Type

Pascal

```
THitTests = set of THitTest;
```

C++

```
THitTest THitTests;
```

File

ComCtrls

Description

This is type ComCtrls.THitTests.

27.4.25 ComCtrls.THKInvalidKeys Type

Pascal

```
THKInvalidKeys = set of THKInvalidKey;
```

C++

```
THKInvalidKey THKInvalidKeys;
```

File

ComCtrls

Description

This is type ComCtrls.THKInvalidKeys.

27.4.26 ComCtrls.THKModifiers Type

Pascal

```
THKModifiers = set of THKModifier;
```

C++

```
THKModifier THKModifiers;
```

File

ComCtrls

Description

This is type ComCtrls.THKModifiers.

27.4.27 ComCtrls.TItemRequest Type

Pascal

```
TItemRequest = set of TItemRequests;
```

C++

```
TItemRequests TItemRequest;
```

File

ComCtrls

Description

This is type ComCtrls.TItemRequest.

27.4.28 ComCtrls.TItemStates Type

TItemState (see page 5454) and TItemStates specify the state of an item in a list view control.

Pascal

```
TItemStates = set of TItemState;
```

C++

```
TItemState TItemStates;
```

File

ComCtrls

Description

TItemStates is a set of TItemState (see page 5454) values. TItemState (see page 5454) indicates the state of an item in a list view. It can have any of the following values:

Value	Meaning
isNone	Item is in its default state
isCut	Item is cut (for cut and paste operation)
isDropHilited	Item is highlighted as a drag-and-drop target
isFocused	Item is focused
isSelected	Item is selected
isActivating	Item is active

27.4.29 ComCtrls.TListHotTrackStyles Type

TListHotTrackStyle (see page 5456) and TListHotTrackStyles specify hot tracking styles for a list view control.

Pascal

```
TListHotTrackStyles = set of TListHotTrackStyle;
```

C++

```
TListHotTrackStyle TListHotTrackStyles;
```

File

ComCtrls

Description

TListHotTrackStyles describes how hot tracking occurs in a list view control. It can have any of the following values:

Value	Meaning
htHandPoint	The mouse cursor turns into a hand.
htUnderlineCold	An underline is displayed under all untracked items.
htUnderlineHot	An underline is displayed under the tracked item.

27.4.30 ComCtrls.TListItemClass Type

Pascal

```
TListItemClass = class of TListItem;
```

C++

```
TListItem TListItemClass;
```

File

ComCtrls

Description

This is type ComCtrls.TListItemClass.

27.4.31 ComCtrls.TLVAdvancedCustomDrawEvent Type

Pascal

```
TLVAdvancedCustomDrawEvent = procedure (Sender: TCustomListView; const ARect: TRect; Stage:
TCustomDrawStage; var DefaultDraw: Boolean) of object;
```

C++

```
(Sender: TCustomListView; const ARect: TRect; Stage: TCustomDrawStage; var DefaultDraw:
Boolean) ( TLVAdvancedCustomDrawEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TLVAdvancedCustomDrawEvent.

27.4.32 ComCtrls.TLVAdvancedCustomDrawItemEvent Type

Pascal

```
TLVAdvancedCustomDrawItemEvent = procedure (Sender: TCustomListView; Item: TListItem; State:
TCustomDrawState; Stage: TCustomDrawStage; var DefaultDraw: Boolean) of object;
```

C++

```
(Sender: TCustomListView; Item: TListItem; State: TCustomDrawState; Stage: TCustomDrawStage;
var DefaultDraw: Boolean) ( TLVAdvancedCustomDrawItemEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TLVAdvancedCustomDrawItemEvent.

27.4.33 ComCtrls.TLVAdvancedCustomDrawSubItemEvent Type

Pascal

```
TLVAdvancedCustomDrawSubItemEvent = procedure (Sender: TCustomListView; Item: TListItem;
SubItem: Integer; State: TCustomDrawState; Stage: TCustomDrawStage; var DefaultDraw: Boolean)
of object;
```

C++

```
(Sender: TCustomListView; Item: TListItem; SubItem: Integer; State: TCustomDrawState; Stage:
TCustomDrawStage; var DefaultDraw: Boolean) ( TLVAdvancedCustomDrawSubItemEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TLVAdvancedCustomDrawSubItemEvent.

27.4.34 ComCtrls.TLVChangeEvent Type

Pascal

```
TLVChangeEvent = procedure (Sender: TObject; Item: TListItem; Change: TItemChange) of object;
```

C++

```
(Sender: TObject; Item: TListItem; Change: TItemChange) ( TLVChangeEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TLVChangeEvent.

27.4.35 ComCtrls.TLVChangingEvent Type

Pascal

```
TLVChangingEvent = procedure (Sender: TObject; Item: TListItem; Change: TItemChange; var
AllowChange: Boolean) of object;
```

C++

```
(Sender: TObject; Item: TListItem; Change: TItemChange; var AllowChange: Boolean) (
TLVChangingEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TLVChangingEvent.

27.4.36 ComCtrls.TLVColumnClickEvent Type

Pascal

```
TLVColumnClickEvent = procedure (Sender: TObject; Column: TListColumn) of object;
```

C++

```
(Sender: TObject; Column: TListColumn) ( TLVColumnClickEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TLVColumnClickEvent.

27.4.37 ComCtrls.TLVColumnRClickEvent Type

Pascal

```
TLVColumnRClickEvent = procedure (Sender: TObject; Column: TListColumn; Point: TPoint) of
object;
```

C++

```
(Sender: TObject; Column: TListColumn; Point: TPoint) ( TLVColumnRClickEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TLVColumnRClickEvent.

27.4.38 ComCtrls.TLVCompareEvent Type

Pascal

```
TLVCompareEvent = procedure (Sender: TObject; Item1, Item2: TListItem; Data: Integer; var
Compare: Integer) of object;
```

C++

```
(Sender: TObject; Item1, Item2: TListItem; Data: Integer; var Compare: Integer) (
TLVCompareEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TLVCompareEvent.

27.4.39 ComCtrls.TLVCreateItemClassEvent Type

Pascal

```
TLVCreateItemClassEvent = procedure (Sender: TCustomListView; var ItemClass: TListItemClass)
of object;
```

C++

```
(Sender: TCustomListView; var ItemClass: TListItemClass) ( TLVCreateItemClassEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TLVCreateItemClassEvent.

27.4.40 ComCtrls.TLVCustomDrawEvent Type

Pascal

```
TLVCustomDrawEvent = procedure (Sender: TCustomListView; const ARect: TRect; var DefaultDraw:
Boolean) of object;
```

C++

```
(Sender: TCustomListView; const ARect: TRect; var DefaultDraw: Boolean) (
TLVCustomDrawEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TLVCustomDrawEvent.

27.4.41 ComCtrls.TLVCustomDrawItemEvent Type

Pascal

```
TLVCustomDrawItemEvent = procedure (Sender: TCustomListView; Item: TListItem; State:
TCustomDrawState; var DefaultDraw: Boolean) of object;
```

C++

```
(Sender: TCustomListView; Item: TListItem; State: TCustomDrawState; var DefaultDraw: Boolean)
( TLVCustomDrawItemEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TLVCustomDrawItemEvent.

27.4.42 ComCtrls.TLVCustomDrawSubItemEvent Type

Pascal

```
TLVCustomDrawSubItemEvent = procedure (Sender: TCustomListView; Item: TListItem; SubItem:
Integer; State: TCustomDrawState; var DefaultDraw: Boolean) of object;
```

C++

```
(Sender: TCustomListView; Item: TListItem; SubItem: Integer; State: TCustomDrawState; var
DefaultDraw: Boolean) ( TLVCustomDrawSubItemEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TLVCustomDrawSubItemEvent.

27.4.43 ComCtrls.TLVDeletedEvent Type

TLVDeletedEvent is the type of a list view's OnDeletion event handler.

Pascal

```
TLVDeletedEvent = procedure (Sender: TObject; Item: TListItem) of object;
```

C++

```
(Sender: TObject; Item: TListItem) ( TLVDeletedEvent)();
```

File

ComCtrls

Description

Sender is the list view controlthat contains the item that is about to be deleted.

Item is the item that is about to be deleted.

27.4.44 ComCtrls.TLVDrawItemEvent Type

TLVDrawItemEvent is the type of the list view OnCustomDrawItem and OnDrawItem event handlers.

Pascal

```
TLVDrawItemEvent = procedure (Sender: TCustomListView; Item: TListItem; Rect: TRect; State:
TOwnerDrawState) of object;
```

C++

```
(Sender: TCustomListView; Item: TListItem; Rect: TRect; State: TOwnerDrawState) (
TLVDrawItemEvent)();
```

File

ComCtrls

Description

Sender is the list view control in which an item is about to be drawn.

Item is the item that is about to be drawn.

Rect defines the coordinates of the item on the list view's canvas.

State indicates various conditions that can influence how the item is drawn, such as whether it is selected, grayed, or focused.

27.4.45 ComCtrls.TLVEditedEvent Type

Pascal

```
TLVEditedEvent = procedure (Sender: TObject; Item: TListItem; var S: string) of object;
```

C++

```
(Sender: TObject; Item: TListItem; var S: string) ( TLVEditedEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TLVEditedEvent.

27.4.46 ComCtrls.TLVEditingEvent Type

Pascal

```
TLVEditingEvent = procedure (Sender: TObject; Item: TListItem; var AllowEdit: Boolean) of
object;
```

C++

```
(Sender: TObject; Item: TListItem; var AllowEdit: Boolean) ( TLVEditingEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TLVEditingEvent.

27.4.47 ComCtrls.TLVInfoTipEvent Type

Pascal

```
TLVInfoTipEvent = procedure (Sender: TObject; Item: TListItem; var InfoTip: string) of object;
```

C++

```
(Sender: TObject; Item: TListItem; var InfoTip: string) ( TLVInfoTipEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TLVInfoTipEvent.

27.4.48 ComCtrls.TLVNotifyEvent Type

Pascal

```
TLVNotifyEvent = procedure (Sender: TObject; Item: TListItem) of object;
```

C++

```
(Sender: TObject; Item: TListItem) ( TLVNotifyEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TLVNotifyEvent.

27.4.49 ComCtrls.TLVOwnerDataEvent Type

TLVOwnerDataEvent is the type of a list view's OnData event handler.

Pascal

```
TLVOwnerDataEvent = procedure (Sender: TObject; Item: TListItem) of object;
```

C++

```
(Sender: TObject; Item: TListItem) ( TLVOwnerDataEvent)();
```

File

ComCtrls

Description

Sender is the list view control in which an item is about to be displayed.

Item is the item that is about to be displayed. Any changes that you make to the item will be reflected in the list view control.

27.4.50 ComCtrls.TLVOwnerDataFindEvent Type

TLVOwnerDataFindEvent is the type of a list view's OnDataFind event handler.

Pascal

```
TLVOwnerDataFindEvent = procedure (Sender: TObject; Find: TItemFind; const FindString: string;
const FindPosition: TPoint; FindData: Pointer; StartIndex: Integer; Direction:
TSearchDirection; Wrap: Boolean; var Index: Integer) of object;
```

C++

```
(Sender: TObject; Find: TItemFind; const FindString: string; const FindPosition: TPoint;
FindData: Pointer; StartIndex: Integer; Direction: TSearchDirection; Wrap: Boolean; var Index:
Integer) ( TLVOwnerDataFindEvent)();
```

File

ComCtrls

Description

Sender is the list view control in which a search has been requested.

Find specifies the type of search to perform (any of ifData, ifPartialString, ifExactString, ifNearest).

FindString specifies the item text to search for. FindString is ignored unless Find contains ifPartialString or ifExactString.

FindPosition specifies the point from which to start the search. FindPosition is ignored unless Find contains ifNearest.

FindData specifies the item data to search for. FindData is ignored unless Find contains ifData.

StartIndex specifies the index of the item from which the search should start.

Direction specifies the direction for the search (sdLeft, sdRight, sdAbove, sdBelow, or sdAll).

Wrap specifies whether the search should continue from the start of the list view control if no match is found.

Index is where to store the index of the found item. Set Index to –1 if no item is found.

27.4.51 ComCtrls.TLVOwnerDataHintEvent Type

TLVOwnerDataHintEvent is the type of a list view's OnDataHint event handler.

Pascal

```
TLVOwnerDataHintEvent = procedure (Sender: TObject; StartIndex, EndIndex: Integer) of object;
```

C++

```
(Sender: TObject; StartIndex, EndIndex: Integer) ( TLVOwnerDataHintEvent)();
```

File

ComCtrls

Description

StartIndex is the first item in the range for which the OnData event may occur.

EndIndex is the last item in the range for which the OnData event may occur.

Use the index information to update the items in the range before the OnData event occurs.

See Also

TCustomListView

27.4.52 ComCtrls.TLVOwnerDataStateChangeEvent Type

TLVOwnerDataStateChangeEvent is the type of a list view's OnDataStateChange event handler.

Pascal

```
TLVOwnerDataStateChangeEvent = procedure (Sender: TObject; StartIndex, EndIndex: Integer;
OldState, NewState: TItemStates) of object;
```

C++

```
(Sender: TObject; StartIndex, EndIndex: Integer; OldState, NewState: TItemStates) (
TLVOwnerDataStateChangeEvent)();
```

File

ComCtrls

Description

Sender is the list view control containing the items that have changed.

StartIndex is the index of the first item that has changed.

EndIndex is the index of the last item that has changed.

OldState is the old state of the items.

NewState is the new state of the items.

See Also

TCustomListView

27.4.53 ComCtrls.TLVSelectItemEvent Type

Pascal

```
TLVSelectItemEvent = procedure (Sender: TObject; Item: TListItem; Selected: Boolean) of object;
```

C++

```
(Sender: TObject; Item: TListItem; Selected: Boolean) ( TLVSelectItemEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TLVSelectItemEvent.

27.4.54 ComCtrls.TLVSubItemImageEvent Type

Pascal

```
TLVSubItemImageEvent = procedure (Sender: TObject; Item: TListItem; SubItem: Integer; var
ImageIndex: Integer) of object;
```

C++

```
(Sender: TObject; Item: TListItem; SubItem: Integer; var ImageIndex: Integer) (
TLVSubItemImageEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TLVSubItemImageEvent.

27.4.55 ComCtrls.TMultiSelectStyle Type

Pascal

```
TMultiSelectStyle = set of TMultiSelectStyles;
```

C++

```
TMultiSelectStyles TMultiSelectStyle;
```

File

ComCtrls

Description

This is type ComCtrls.TMultiSelectStyle.

27.4.56 ComCtrls.TOnGetMonthInfoEvent Type

TOnGetMonthInfoEvent is the type of the TCommonCalendar.OnGetMonthInfo event handler.

Pascal

```
TOnGetMonthInfoEvent = procedure (Sender: TObject; Month: LongWord; var MonthBoldInfo:
LongWord) of object;
```

C++

```
(Sender: TObject; Month: LongWord; var MonthBoldInfo: LongWord) ( TOnGetMonthInfoEvent)();
```

File

ComCtrls

Description

Use TOnGetMonthInfoEvent along with the TCommonCalendar.BoldDays method to set which days are bold in the month that is about to be displayed in the calendar.

Sender is the TObject for the calendar.

Month is a value that corresponds to the month that is about to be displayed in the calendar. If Month is 1, January is about to be displayed, if Month is 2, February is about to be displayed, and so on.

MonthBoldInfo specifies which days of the month should be bold. You should pass MonthBoldInfo to the BoldDays method. You shouldn't work with BoldDays directly.

The first 31 bits of the MonthBoldInfo variable are used to determine whether a particular day of the month should be bold.

See Also

TCommonCalendar (see page 3558)

27.4.57 ComCtrls.TOwnerDrawState Type

Indicates state information that can influence how an item is drawn.

Pascal

```
TOwnerDrawState = Windows.TOwnerDrawState;
```

C++

```
Windows.TOwnerDrawState TOwnerDrawState;
```

File

ComCtrls

Description

TOwnerDrawState is used by event handlers in owner-draw controls to indicate the state of an item about to be drawn. It is a set that includes zero or more of the following:

Value	Meaning
odSelected	The item is selected.

odGrayed	The item should appear grayed.
odDisabled	The item is disabled.
odChecked	The item should appear checked. (This does not apply to all objects)
odFocused	Keyboard input is directed to the item.
odDefault	The item is the default item (This does not apply to all objects)
odHotLight	The item is hot-lighted because under the mouse (only if the operating system supports hot-lighting)
odInactive	The item and its associated menu are inactive
odNoAccel	Any underline of an accelerator character should be skipped when drawing the control
odNoFocusRect	Do not display visual cues that indicate focus
odReserved1	Not currently used.
odReserved2	Not currently used.
odComboBoxEdit	The drawing takes place in the edit box of a combo box control.

27.4.58 ComCtrls.TPageScrollEvent Type

TPageScrollEvent is the type of a page scroller's OnScroll event handler.

Pascal

```
TPageScrollEvent = procedure (Sender: TObject; Shift: TShiftState; X, Y: Integer; Orientation:
TPageScrollerOrientation; var Delta: Integer) of object;
```

C++

```
(Sender: TObject; Shift: TShiftState; X, Y: Integer; Orientation: TPageScrollerOrientation;
var Delta: Integer) ( TPageScrollEvent)();
```

File

ComCtrls

Description

Sender is the page scroller that is being scrolled.

Shift is the state of the Shift, Alt, Ctrl keys and mouse buttons.

X is the horizontal position of the page scroller.

Y is the vertical position of the page scroller.

Orientation is the direction for the scroll.

Delta is the amount of the scroll.

See Also

TPageScroller (see page 4548)

27.4.59 ComCtrls.TProgressRange Type

Pascal

```
TProgressRange = Integer;
```

C++

```
int TProgressRange;
```

File

ComCtrls

Description

This is type ComCtrls.TProgressRange.

27.4.60 ComCtrls.TRichEditProtectChange Type

Pascal

```
TRichEditProtectChange = procedure (Sender: TObject; StartPos, EndPos: Integer; var
AllowChange: Boolean) of object;
```

C++

```
(Sender: TObject; StartPos, EndPos: Integer; var AllowChange: Boolean) (
TRichEditProtectChange)();
```

File

ComCtrls

Description

This is type ComCtrls.TRichEditProtectChange.

27.4.61 ComCtrls.TRichEditResizeEvent Type

Pascal

```
TRichEditResizeEvent = procedure (Sender: TObject; Rect: TRect) of object;
```

C++

```
(Sender: TObject; Rect: TRect) ( TRichEditResizeEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TRichEditResizeEvent.

27.4.62 ComCtrls.TRichEditSaveClipboard Type

Pascal

```
TRichEditSaveClipboard = procedure (Sender: TObject; NumObjects, NumChars: Integer; var
SaveClipboard: Boolean) of object;
```

C++

```
(Sender: TObject; NumObjects, NumChars: Integer; var SaveClipboard: Boolean) (
TRichEditSaveClipboard)();
```

File

ComCtrls

Description

This is type ComCtrls.TRichEditSaveClipboard.

27.4.63 ComCtrls.TSBCreatePanelClassEvent Type

TSBCreatePanelClassEvent is the type of the OnCreatePanelClass event handler.

Pascal

```
TSBCreatePanelClassEvent = procedure (Sender: TCustomStatusBar; var PanelClass:
TStatusPanelClass) of object;
```

C++

```
(Sender: TCustomStatusBar; var PanelClass: TStatusPanelClass) ( TSBCreatePanelClassEvent)();
```

File

ComCtrls

Description

Sender is the status bar object that needs to add a new panel.

PanelClass returns a class reference to the TStatusPanel (see page 4825) descendant that should be used to implement the new panel.

27.4.64 ComCtrls.TSearchTypes Type

Set of TSearchType (see page 5464).

Pascal

```
TSearchTypes = set of TSearchType;
```

C++

```
TSearchType TSearchTypes;
```

File

ComCtrls

Description

C++ Examples:

```
/*
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FindDialog1->Position = Point(RichEdit1->Left + RichEdit1->Width, RichEdit1->Top);
  FindDialog1->Execute();
}

void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  int FoundAt, StartPos, ToEnd;
  TSearchTypes mySearchTypes;
  // begin the search after the current selection
  // if there is one
  // otherwise, begin at the start of the text
  if (FindDialog1->Options.Contains(frMatchCase))
    mySearchTypes << stMatchCase;
  if (FindDialog1->Options.Contains(frWholeWord))
    mySearchTypes << stWholeWord;
  if (RichEdit1->SelLength)
    StartPos = RichEdit1->SelStart + RichEdit1->SelLength;
  else
    StartPos = 0;
  // ToEnd is the length from StartPos
  // to the end of the text in the rich edit control
  ToEnd = RichEdit1->Text.Length() - StartPos;
  FoundAt = RichEdit1->FindText(FindDialog1->FindText, StartPos, ToEnd, mySearchTypes);
  if (FoundAt != -1)
  {
    RichEdit1->SetFocus();
    RichEdit1->SelStart = FoundAt;
    RichEdit1->SelLength = FindDialog1->FindText.Length();
  }
  else Beep();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("../Overview.RTF");  // starts in Debug
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FindDialog1.Position :=
```

```
      Point(RichEdit1.Left + RichEdit1.Width, RichEdit1.Top);
    FindDialog1.Execute;
  end;

  procedure TForm1.FindDialog1Find(Sender: TObject);
  var
    FoundAt: LongInt;
    StartPos, ToEnd: Integer;
    mySearchTypes : TSearchTypes;
    myFindOptions : TFindOptions;
  begin
    with RichEdit1 do
    begin
      if frMatchCase in FindDialog1.Options then
         mySearchTypes := mySearchTypes + [stMatchCase];
      if frWholeWord in FindDialog1.Options then
         mySearchTypes := mySearchTypes + [stWholeWord];
      { begin the search after the current selection if there is one }
      { otherwise, begin at the start of the text }
      if SelLength <> 0 then
        StartPos := SelStart + SelLength
      else
        StartPos := 0;
      { ToEnd is the length from StartPos to the end of the
        text in the rich edit control }
      ToEnd := Length(Text) - StartPos;
      FoundAt :=
        FindText(FindDialog1.FindText, StartPos, ToEnd, mySearchTypes);
      if FoundAt <> -1 then
      begin
        SetFocus;
        SelStart := FoundAt;
        SelLength := Length(FindDialog1.FindText);
      end
      else Beep;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const Path = 'OverView.RTF';
  begin
    RichEdit1.PlainText := False;
    RichEdit1.Lines.LoadFromFile(Path);
    RichEdit1.ScrollBars := ssVertical;
  end;
```

27.4.65 ComCtrls.TSectionDragEvent Type

TSectionDragEvent is the type for event handlers that respond when the user drags a section in a header control.

Pascal

```
TSectionDragEvent = procedure (Sender: TObject; FromSection, ToSection: THeaderSection; var
AllowDrag: Boolean) of object;
```

C++

```
(Sender: TObject; FromSection, ToSection: THeaderSection; var AllowDrag: Boolean) (
TSectionDragEvent)();
```

File

ComCtrls

Description

Sender is the header control that includes the dragged section.

FromSection is the header section that the user is dragging.

ToSection is the header section currently in the position where the user dragged FromSection.

AllowDrag returns a value that indicates whether the header control repositions the headers to accommodate the user's drag. If AllowDrag is false on exit from the event handler, the headers remain in their original positions, as if the user had not tried a drag operation.

See Also

THeaderControl (see page 4118)

27.4.66 ComCtrls.TSectionNotifyEvent Type

Pascal

```
TSectionNotifyEvent = procedure (HeaderControl: THeaderControl; Section: THeaderSection) of
object;
```

C++

```
(HeaderControl: THeaderControl; Section: THeaderSection) ( TSectionNotifyEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TSectionNotifyEvent.

27.4.67 ComCtrls.TSectionTrackEvent Type

Pascal

```
TSectionTrackEvent = procedure (HeaderControl: THeaderControl; Section: THeaderSection; Width:
Integer; State: TSectionTrackState) of object;
```

C++

```
(HeaderControl: THeaderControl; Section: THeaderSection; Width: Integer; State:
TSectionTrackState) ( TSectionTrackEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TSectionTrackEvent.

27.4.68 ComCtrls.TStatusPanelClass Type

TStatusPanelClass defines the metaclass for TStatusPanel (see page 4825).

Pascal

```
TStatusPanelClass = class of TStatusPanel;
```

C++

```
TStatusPanel TStatusPanelClass;
```

File

ComCtrls

Description

TStatusPanelClass is the metaclass for TStatusPanel (see page 4825). Its value is the class reference for TStatusPanel (see page 4825) or for one of its descendants.

27.4.69 ComCtrls.TTabChangingEvent Type

Pascal

```
TTabChangingEvent = procedure (Sender: TObject; var AllowChange: Boolean) of object;
```

C++

```
(Sender: TObject; var AllowChange: Boolean) ( TTabChangingEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TTabChangingEvent.

27.4.70 ComCtrls.TTabGetImageEvent Type

Pascal

```
TTabGetImageEvent = procedure (Sender: TObject; TabIndex: Integer; var ImageIndex: Integer) of
object;
```

C++

```
(Sender: TObject; TabIndex: Integer; var ImageIndex: Integer) ( TTabGetImageEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TTabGetImageEvent.

27.4.71 ComCtrls.TTBAdvancedCustomDrawBtnEvent Type

Pascal

```
TTBAdvancedCustomDrawBtnEvent = procedure (Sender: TToolBar; Button: TToolButton; State:
TCustomDrawState; Stage: TCustomDrawStage; var Flags: TTBCustomDrawFlags; var DefaultDraw:
Boolean) of object;
```

C++

```
(Sender: TToolBar; Button: TToolButton; State: TCustomDrawState; Stage: TCustomDrawStage; var
Flags: TTBCustomDrawFlags; var DefaultDraw: Boolean) ( TTBAdvancedCustomDrawBtnEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TTBAdvancedCustomDrawBtnEvent.

27.4.72 ComCtrls.TTBAdvancedCustomDrawEvent Type

Pascal

```
TTBAdvancedCustomDrawEvent = procedure (Sender: TToolBar; const ARect: TRect; Stage:
TCustomDrawStage; var DefaultDraw: Boolean) of object;
```

C++

```
(Sender: TToolBar; const ARect: TRect; Stage: TCustomDrawStage; var DefaultDraw: Boolean) (
TTBAdvancedCustomDrawEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TTBAdvancedCustomDrawEvent.

27.4.73 ComCtrls.TTBButtonEvent Type

TTBButtonEvent is the type of event handlers that respond when the user adds or deletes a tool button in the toolbar customize dialog.

Pascal

```
TTBButtonEvent = procedure (Sender: TToolbar; Button: TToolButton) of object;
```

C++

```
(Sender: TToolbar; Button: TToolButton) ( TTBButtonEvent)();
```

File

ComCtrls

Description

TTBButtonEvent is the type for the OnCustomizeAdded and OnCustomizeDelete events. It allows a toolbar to respond to notifications of changes that the user makes to the items in the toolbar.

Sender is the toolbar whose button was just added or deleted.

Button is the button that was just added or deleted.

See Also

TToolBar (see page 4994)

27.4.74 ComCtrls.TTBCustomDrawBtnEvent Type

Pascal

```
TTBCustomDrawBtnEvent = procedure (Sender: TToolBar; Button: TToolButton; State:
TCustomDrawState; var DefaultDraw: Boolean) of object;
```

C++

```
(Sender: TToolBar; Button: TToolButton; State: TCustomDrawState; var DefaultDraw: Boolean) (
TTBCustomDrawBtnEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TTBCustomDrawBtnEvent.

27.4.75 ComCtrls.TTBCustomDrawEvent Type

Pascal

```
TTBCustomDrawEvent = procedure (Sender: TToolBar; const ARect: TRect; var DefaultDraw:
Boolean) of object;
```

C++

```
(Sender: TToolBar; const ARect: TRect; var DefaultDraw: Boolean) ( TTBCustomDrawEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TTBCustomDrawEvent.

27.4.76 ComCtrls.TTBCustomDrawFlags Type

TTBCustomDrawFlags is the type of the OnAdvancedCustomDrawButton event handler.

Pascal

```
TTBCustomDrawFlags = set of (tbNoEdges, tbHiliteHotTrack, tbNoOffset, tbNoMark,
tbNoEtchedEffect);
```

C++

```
(tbNoEdges, tbHiliteHotTrack, tbNoOffset, tbNoMark, tbNoEtchedEffect) TTBCustomDrawFlags;
```

File

ComCtrls

Description

TTBCustomDrawFlags influences the subsequent painting of a tool button following an OnAdvancedCustomDrawButton event in the cdPrePaint stage:

Value	Meaning
tbNoEdges	The button is drawn without edges to distinguish it from the tool bar.
tbHiliteHotTrack	Enables the change in background color when the button is hot tracked. If this is not included in the set, the background color does not change for hot tracking.
tbNoOffset	The button is not offset when in the "down" position.
tbNoMark	Disables the default change in background color on buttons with the Marked property set to Truetrue.
tbNoEtchedEffect	Disables the etched effect that appears by default on disabled tool buttons.

See Also

TToolBar

27.4.77 ComCtrls.TTBCustomizeQueryEvent Type

TTBCustomizeQueryEvent is the type of event handlers that allow a toolbar to decide whether to allow a user customization.

Pascal

```
TTBCustomizeQueryEvent = procedure (Sender: TToolbar; Index: Integer; var Allow: Boolean) of
object;
```

C++

```
(Sender: TToolbar; Index: Integer; var Allow: Boolean) ( TTBCustomizeQueryEvent)();
```

File

ComCtrls

Description

TTBCustomizeQueryEvent is the type for the OnCustomizeCanInsert and OnCustomizeCanDelete events. It allows a toolbar to approve or block the insertion or deletion of a button by the user.

Sender is the customizable toolbar that the user is trying to modify.

Index is the index of the button to be deleted or the index where a new button is to be inserted.

Allow returns an indication of whether the user can proceed with the operation. On entry, Allow is T t rue, indicating that the user can add or delete the button. The event handler can change Allow to F f alse to block the change to the toolbar.

See Also

TToolBar (see page 4994)

27.4.78 ComCtrls.TTBGradientDrawingOptions Type

Specifies the style of the gradient.

Pascal

```
TTBGradientDrawingOptions = set of (gdoHotTrack, gdoGradient);
```

C++

```
(gdoHotTrack, gdoGradient) TTBGradientDrawingOptions;
```

File

ComCtrls

Description

This type allows you to use one of the following options:

- gdoHotTrack
- gdoGradient

27.4.79 ComCtrls.TTBNewButtonEvent Type

Pascal

```
TTBNewButtonEvent = procedure (Sender: TToolbar; Index: Integer; var Button: TToolButton) of
object;
```

C++

```
(Sender: TToolbar; Index: Integer; var Button: TToolButton) ( TTBNewButtonEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TTBNewButtonEvent.

27.4.80 ComCtrls.TToolButtonActionLinkClass Type

Pascal

```
TToolButtonActionLinkClass = class of TToolButtonActionLink;
```

C++

```
TToolButtonActionLink TToolButtonActionLinkClass;
```

File

ComCtrls

Description

This is type ComCtrls.TToolButtonActionLinkClass.

27.4.81 ComCtrls.TTreeNodeClass Type

Pascal

```
TTreeNodeClass = class of TTreeNode;
```

C++

```
TTreeNode TTreeNodeClass;
```

File

ComCtrls

Description

This is type ComCtrls.TTreeNodeClass.

27.4.82 ComCtrls.TTVAdvancedCustomDrawEvent Type

Used in the TCustomTreeView.OnAdvancedCustomDraw event.

Pascal

```
TTVAdvancedCustomDrawEvent = procedure (Sender: TCustomTreeView; const ARect: TRect; Stage:
TCustomDrawStage; var DefaultDraw: Boolean) of object;
```

C++

```
(Sender: TCustomTreeView; const ARect: TRect; Stage: TCustomDrawStage; var DefaultDraw:
Boolean) ( TTVAdvancedCustomDrawEvent)();
```

File

ComCtrls

Description

This event is used in the TCustomTreeView.OnAdvancedCustomDraw event when a tree-view control is being painted.

Sender is the tree-view control that is being painted.

ARect is the rectangle that defines the boundaries of the tree-view control on its canvas.

Stage indicates the current stage of the painting process.

DefaultDraw specifies whether the control should paint itself after the event handler exits. This parameter is only used when **Stage** is **cdPrePaint**.

See Also

OnAdvancedCustomDraw (see page 4016)

27.4.83 ComCtrls.TTVAdvancedCustomDrawItemEvent Type

Used in the TCustomTreeView.OnAdvancedCustomDrawItem event.

Pascal

```
TTVAdvancedCustomDrawItemEvent = procedure (Sender: TCustomTreeView; Node: TTreeNode; State:
TCustomDrawState; Stage: TCustomDrawStage; var PaintImages, DefaultDraw: Boolean) of object;
```

C++

```
(Sender: TCustomTreeView; Node: TTreeNode; State: TCustomDrawState; Stage: TCustomDrawStage;
var PaintImages, DefaultDraw: Boolean) ( TTVAdvancedCustomDrawItemEvent)();
```

File

ComCtrls

Description

This event is used in the TCustomTreeView.OnAdvancedCustomDrawItem event when an item is about to be painted.

Sender is the tree-view control in which an item is about to be painted.

Node is the item that is about to be painted.

State is the item's current state, which is a TCustomDrawState (see page 5483) type: one or more of **cdsSelected**, **cdsGrayed**, **cdsDisabled**, **cdsChecked**, **cdsFocused**, **csDefault**, **cdsHot**, **cdsMarked**, **cdsIndeterminate**.

Stage is a TCustomDrawStage (see page 5445) type that indicates the current stage of the painting process.

PaintImages specifies whether the control should paint the images associated with the control. This parameter is only used when **Stage** is **cdPrePaint**.

DefaultDraw specifies whether the control should paint the text of the item after the event handler exits. This parameter is only used when **Stage** is **cdPrePaint**.

See Also

OnAdvancedCustomDrawItem (see page 4016)

TCustomDrawState (see page 5483)

TCustomDrawStage (see page 5445)

27.4.84 ComCtrls.TTVChangedEvent Type

Used in the TCustomTreeView.OnChange event.

Pascal

```
TTVChangedEvent = procedure (Sender: TObject; Node: TTreeNode) of object;
```

C++

```
(Sender: TObject; Node: TTreeNode) ( TTVChangedEvent)();
```

File

ComCtrls

Description

This event is used in the TCustomTreeView.OnChange event whenever the selection has changed from one node to another.

The **Sender** parameter is the tree view whose selected node changes, and the **Node** parameter is the newly selected node.

See Also

ChangeDelay (see page 3999)

RightClickSelect (see page 4006)

Selected (see page 4007)

OnChange (see page 4017)

OnChanging (see page 4018)

OnMouseUp (see page 5867)

27.4.85 ComCtrls.TTVChangingEvent Type

Used in the TCustomTreeView.OnChanging event.

Pascal

```
TTVChangingEvent = procedure (Sender: TObject; Node: TTreeNode; var AllowChange: Boolean) of
object;
```

C++

```
(Sender: TObject; Node: TTreeNode; var AllowChange: Boolean) ( TTVChangingEvent)();
```

File

ComCtrls

Description

This event is used in the TCustomTreeView.OnChanging event when the selection is about to change from one node to another.

The **Sender** parameter is the tree view whose selected node changes, and the **Node** parameter specifies the node that is about to be selected. **AllowChange** indicates whether the node can be selected or not.

See Also

Selected (see page 4007)

OnChanging (see page 4018)

OnChange (see page 4017)

27.4.86 ComCtrls.TTVCollapsingEvent Type

Used in the TCustomTreeView.OnCollapsing event.

Pascal

```
TTVCollapsingEvent = procedure (Sender: TObject; Node: TTreeNode; var AllowCollapse: Boolean)
of object;
```

C++

```
(Sender: TObject; Node: TTreeNode; var AllowCollapse: Boolean) ( TTVCollapsingEvent)();
```

File

ComCtrls

Description

This event is used in the TCustomTreeView.OnCollapsing event when a node is about to be collapsed.

The **Sender** parameter is the tree view whose selected node collapses, and the **Node** parameter specifies the node that is about to be collapsed. **AllowCollapse** indicates whether the node can collapse or not.

See Also

FullCollapse (see page 3983)

Collapse (see page 5235)

OnCollapsing (see page 4018)

27.4.87 ComCtrls.TTVCompareEvent Type

Used in the TCustomTreeView.OnCompare event.

Pascal

```
TTVCompareEvent = procedure (Sender: TObject; Node1, Node2: TTreeNode; Data: Integer; var
Compare: Integer) of object;
```

C++

```
(Sender: TObject; Node1, Node2: TTreeNode; Data: Integer; var Compare: Integer) (
TTVCompareEvent)();
```

File

ComCtrls

Description

This event is used in the TCustomTreeView.OnCompare event when two nodes must be compared during a sort of the nodes in the tree view.

The **Sender** parameter is the tree view whose nodes are compared.

Node1 and **Node2** are the nodes being compared.

Compare parameter is less than 0 if **Node1** is less than **Node2**. **Compare** is 0 if **Node1** is equivalent to **Node2**, and **Compare** is greater than 0 if **Node1** is greater than **Node2**.

See Also

SortType (see page 4014)

AlphaSort (see page 3972)

CustomSort (see page 3978)

OnCompare (see page 4019)

27.4.88 ComCtrls.TTVCreateNodeClassEvent Type

Pascal

```
TTVCreateNodeClassEvent = procedure (Sender: TCustomTreeView; var NodeClass: TTreeNodeClass)
of object;
```

C++

```
(Sender: TCustomTreeView; var NodeClass: TTreeNodeClass) ( TTVCreateNodeClassEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TTVCreateNodeClassEvent.

27.4.89 ComCtrls.TTVCustomDrawEvent Type

Used in the TCustomTreeView.OnCustomDraw event.

Pascal

```
TTVCustomDrawEvent = procedure (Sender: TCustomTreeView; const ARect: TRect; var DefaultDraw:
Boolean) of object;
```

C++

```
(Sender: TCustomTreeView; const ARect: TRect; var DefaultDraw: Boolean) (
TTVCustomDrawEvent)();
```

File

ComCtrls

Description

This event is used in the TCustomTreeView.OnCustomDraw event.

Sender is the tree-view control that is about to be painted.

ARect is the rectangle that defines the boundaries of the tree-view control.

DefaultDraw specifies whether the control should paint itself after the event handler exits.

See Also

OnCustomDraw (see page 4019)

27.4.90 ComCtrls.TTVCustomDrawItemEvent Type

Used in the TCustomTreeView.OnCustomDrawItem event.

Pascal

```
TTVCustomDrawItemEvent = procedure (Sender: TCustomTreeView; Node: TTreeNode; State:
```

```
TCustomDrawState; var DefaultDraw: Boolean) of object;
```

C++

```
(Sender: TCustomTreeView; Node: TTreeNode; State: TCustomDrawState; var DefaultDraw: Boolean)
( TTVCustomDrawItemEvent)();
```

File

ComCtrls

Description

This event is used in the TCustomTreeView.OnCustomDrawItem event.

Sender is the tree-view control in which an item is about to be drawn.

Node is the item that is about to be drawn.

State is the item's current state, which is a TCustomDrawState (see page 5483) type: one or more of **cdsSelected**, **cdsGrayed**, **cdsDisabled**, **cdsChecked**, **cdsFocused**, **cdsDefault**, **cdsHot**, **cdsMarked**, **cdsIndeterminate**.

DefaultDraw specifies whether the control should paint the item after the event handler exits.

See Also

OnCustomDrawItem (see page 4021)

TCustomDrawState (see page 5483)

27.4.91 ComCtrls.TTVEditedEvent Type

Used in the TCustomTreeView.OnEdited event.

Pascal

```
TTVEditedEvent = procedure (Sender: TObject; Node: TTreeNode; var S: string) of object;
```

C++

```
(Sender: TObject; Node: TTreeNode; var S: string) ( TTVEditedEvent)();
```

File

ComCtrls

Description

This event is used in the TCustomTreeView.OnEdited event after the user edits the Text property of a node.

Sender is the tree-view control in which a node's label was edited.

Node is the item whose label was edited.

The **S** parameter is the new value of the node’s Text property.

See Also

ReadOnly (see page 4006)

Text (see page 5265)

OnEdited (see page 4024)

OnEditing (see page 4024)

27.4.92 ComCtrls.TTVEditingEvent Type

Used in the TCustomTreeView.OnEditing event.

Pascal

```
TTVEditingEvent = procedure (Sender: TObject; Node: TTreeNode; var AllowEdit: Boolean) of
object;
```

C++

```
(Sender: TObject; Node: TTreeNode; var AllowEdit: Boolean) ( TTVEditingEvent)();
```

File

ComCtrls

Description

This event is used in the TCustomTreeView.OnEditing event when the user starts to edit the Text property of a node.

Sender is the tree-view control in which an item is about to be edited.

Node is the item whose label is about to be edited.

AllowEdit indicates whether the user is allowed to edit the label of the node specified by the **Node** parameter.

See Also

ReadOnly (see page 4006)

OnEdited (see page 4024)

OnEditing (see page 4024)

27.4.93 ComCtrls.TTVExpandedEvent Type

Represents the event handler for events that notify a tree view of changes to one of its nodes.

Pascal

```
TTVExpandedEvent = procedure (Sender: TObject; Node: TTreeNode) of object;
```

C++

```
(Sender: TObject; Node: TTreeNode) ( TTVExpandedEvent)();
```

File

ComCtrls

Description

TTVExpandedEvent is used to respond to notifications of events that occur to the nodes of a tree view control. It is a generic event handler for various events that notify the tree view of something that happened to a node.

Sender is the tree-view control that contains the node that is the focus of the notification.

Node is the item that has been expanded, collapsed, deleted, or which is about to be drawn.

See Also

OnCollapsed (see page 4018)

OnDeletion (see page 4024)

OnExpanded (see page 4025)

OnGetImageIndex (see page 4026)

OnGetSelectedIndex (see page 4026)

27.4.94 ComCtrls.TTVExpandingEvent Type

Used in the TCustomTreeView.OnExpanding event.

Pascal

```
TTVExpandingEvent = procedure (Sender: TObject; Node: TTreeNode; var AllowExpansion: Boolean)
of object;
```

C++

```
(Sender: TObject; Node: TTreeNode; var AllowExpansion: Boolean) ( TTVExpandingEvent)();
```

File

ComCtrls

Description

This event is used in the TCustomTreeView.OnExpanding event when a node is about to be expanded.

Sender is the tree-view control in which a node is about to be expanded.

Node is the node about to be expanded.

AllowExpansion indicates whether the node can be expanded.

See Also

FullExpand (see page 3984)

Expand (see page 5242)

OnExpanding (see page 4025)

OnExpanded (see page 4025)

27.4.95 ComCtrls.TUDChangingEvent Type

Pascal

```
TUDChangingEvent = procedure (Sender: TObject; var AllowChange: Boolean) of object;
```

C++

```
(Sender: TObject; var AllowChange: Boolean) ( TUDChangingEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TUDChangingEvent.

27.4.96 ComCtrls.TUDChangingEventEx Type

Pascal

```
TUDChangingEventEx = procedure (Sender: TObject; var AllowChange: Boolean; NewValue: SmallInt;
Direction: TUpDownDirection) of object;
```

C++

```
(Sender: TObject; var AllowChange: Boolean; NewValue: SmallInt; Direction: TUpDownDirection) (
TUDChangingEventEx)();
```

File

ComCtrls

Description

This is type ComCtrls.TUDChangingEventEx.

27.4.97 ComCtrls.TUDClickEvent Type

Pascal

```
TUDClickEvent = procedure (Sender: TObject; Button: TUDBtnType) of object;
```

C++

```
(Sender: TObject; Button: TUDBtnType) ( TUDClickEvent)();
```

File

ComCtrls

Description

This is type ComCtrls.TUDClickEvent.

27.4.98 ComCtrls.TWidth Type

Pascal

```
TWidth = ColumnHeaderWidth..MaxInt;
```

C++

```
ColumnHeaderWidth..MaxInt TWidth;
```

File

ComCtrls

Description

This is type ComCtrls.TWidth.

27.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CN_BANDCHANGE (see page 5521)	This is constant ComCtrls.CN_BANDCHANGE.
CN_DROPDOWNCLOSED (see page 5521)	This is constant ComCtrls.CN_DROPDOWNCLOSED.
ColumnHeaderWidth (see page 5522)	This is constant ComCtrls.ColumnHeaderWidth.
ColumnTextWidth (see page 5522)	This is constant ComCtrls.ColumnTextWidth.
ComCtlVersionIE3 (see page 5522)	This is constant ComCtrls.ComCtlVersionIE3.
ComCtlVersionIE4 (see page 5523)	This is constant ComCtrls.ComCtlVersionIE4.
ComCtlVersionIE401 (see page 5523)	This is constant ComCtrls.ComCtlVersionIE401.
ComCtlVersionIE5 (see page 5523)	This is constant ComCtrls.ComCtlVersionIE5.
ComCtlVersionIE501 (see page 5524)	This is constant ComCtrls.ComCtlVersionIE501.
ComCtlVersionIE6 (see page 5524)	This is constant ComCtrls.ComCtlVersionIE6.

27.5.1 ComCtrls.CN_BANDCHANGE Constant

Pascal

```
CN_BANDCHANGE = WM_USER + $1000;
```

C++

```
CN_BANDCHANGE = WM_USER + $1000;
```

File

ComCtrls

Description

This is constant ComCtrls.CN_BANDCHANGE.

27.5.2 ComCtrls.CN_DROPDOWNCLOSED Constant

Pascal

```
CN_DROPDOWNCLOSED = WM_USER + $1000;
```

C++

```
CN_DROPDOWNCLOSED = WM_USER + $1000;
```

File

ComCtrls

Description

This is constant ComCtrls.CN_DROPDOWNCLOSED.

27.5.3 ComCtrls.ColumnHeaderWidth Constant

Pascal

```
ColumnHeaderWidth = LVSCW_AUTOSIZE_USEHEADER;
```

C++

```
ColumnHeaderWidth = LVSCW_AUTOSIZE_USEHEADER;
```

File

ComCtrls

Description

This is constant ComCtrls.ColumnHeaderWidth.

27.5.4 ComCtrls.ColumnTextWidth Constant

Pascal

```
ColumnTextWidth = LVSCW_AUTOSIZE;
```

C++

```
ColumnTextWidth = LVSCW_AUTOSIZE;
```

File

ComCtrls

Description

This is constant ComCtrls.ColumnTextWidth.

27.5.5 ComCtrls.ComCtlVersionIE3 Constant

Pascal

```
ComCtlVersionIE3 = $00040046;
```

C++

```
ComCtlVersionIE3 = 0x00040046;
```

File

ComCtrls

Description

This is constant ComCtrls.ComCtlVersionIE3.

27.5.6 ComCtrls.ComCtlVersionIE4 Constant

Pascal

```
ComCtlVersionIE4 = $00040047;
```

C++

```
ComCtlVersionIE4 = 0x00040047;
```

File

ComCtrls

Description

This is constant ComCtrls.ComCtlVersionIE4.

27.5.7 ComCtrls.ComCtlVersionIE401 Constant

Pascal

```
ComCtlVersionIE401 = $00040048;
```

C++

```
ComCtlVersionIE401 = 0x00040048;
```

File

ComCtrls

Description

This is constant ComCtrls.ComCtlVersionIE401.

27.5.8 ComCtrls.ComCtlVersionIE5 Constant

Pascal

```
ComCtlVersionIE5 = $00050050;
```

C++

```
ComCtlVersionIE5 = 0x00050050;
```

File

ComCtrls

Description

This is constant ComCtrls.ComCtlVersionIE5.

27.5.9 ComCtrls.ComCtlVersionIE501 Constant

Pascal

```
ComCtlVersionIE501 = $00050051;
```

C++

```
ComCtlVersionIE501 = 0x00050051;
```

File

ComCtrls

Description

This is constant ComCtrls.ComCtlVersionIE501.

27.5.10 ComCtrls.ComCtlVersionIE6 Constant

Pascal

```
ComCtlVersionIE6 = $00060000;
```

C++

```
ComCtlVersionIE6 = 0x00060000;
```

File

ComCtrls

Description

This is constant ComCtrls.ComCtlVersionIE6.

28 ComObj Namespace

This is namespace ComObj.

Topics

Name	Description
Classes (see page 5528)	The following table lists classes in this documentation.
Interfaces (see page 5588)	The following table lists interfaces in this documentation.
Functions (see page 5590)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 5604)	The following table lists structs, records, enums in this documentation.
Types (see page 5606)	The following table lists types in this documentation.
Variables (see page 5610)	The following table lists variables in this documentation.

Classes

	Name	Description
	EOleError (see page 5528)	EOleError is the exception class for low-level OLE errors.
	EOleException (see page 5529)	EOleException is the exception class for OLE automation errors that occur in methods or properties of an object.
	EOleRegistrationError (see page 5531)	EOleRegistrationError is the exception class for errors that occur when registering an OLE object.
	EOleSysError (see page 5532)	EOleSysError is the exception class for errors specific to the OLE IDispatch interface.
	TAutoIntfObject (see page 5534)	TAutoIntfObject implements the IDispatch and ISupportErrorInfo interfaces.
	TAutoObject (see page 5540)	TAutoObject is a CoClass that supports the IDispatch interface, and that can be used as a base class for ActiveX Automation servers.
	TAutoObjectFactory (see page 5545)	TAutoObjectFactory is the class factory for TAutoObject (see page 5540).
	TComClassManager (see page 5550)	TComClassManager is the class type of the ComClassManager (see page 5591) variable in the ComObj unit.
	TComObject (see page 5553)	TComObject is the base class for creating simple COM classes, such as those used for creating Shell extensions.
	TComObjectFactory (see page 5563)	TComObjectFactory is the type of the class factory object used to instantiate a TComObject (see page 5553) class.
	TComServerObject (see page 5575)	TComServerObject is the abstract base class for TComServer.
	TTypedComObject (see page 5581)	TTypedComObject supports the IProvideClassInfo interface and is the class type for COM objects that can provide the type information without requiring a type library to be loaded.
	TTypedComObjectFactory (see page 5583)	TTypedComObjectFactory is the type of the class factory object used to instantiate a TypedComObject class.

Functions

	Name	Description
	ClassIDToProgID (see page 5590)	Returns the programmatic ID for a specified class given its class ID (CLSID).
	ComClassManager (see page 5591)	Returns a TComClassManager (see page 5550) object.
	CreateClassID (see page 5591)	CreateClassID generates a new GUID and returns it as a string.
	CreateComObject (see page 5592)	Instantiates a single instance of a COM object.
	CreateOleObject (see page 5592)	Instantiates an Automation object.
	CreateRegKey (see page 5593)	Creates or opens a registry key that is a subkey of HKEY_CLASSES_ROOT.
	CreateRemoteComObject (see page 5594)	Creates a Com object on another machine and returns an IUnknown interface for it.
	DeleteRegKey (see page 5595)	Deletes a single key that is a subkey of HKEY_CLASSES_ROOT from the system registry.
	DispatchInvoke (see page 5595)	This is function ComObj.DispatchInvoke.
	DispatchInvokeError (see page 5596)	This is function ComObj.DispatchInvokeError.
	GetActiveOleObject (see page 5596)	Retrieves a reference to an IDispatch interface to a currently running, registered COM object.
	GetDispatchPropValue (see page 5596)	Returns the value of a property on an IDispatch interface.
	GetDispatchPropValue (see page 5596)	Returns the value of a property on an IDispatch interface.
	GetRegStringValue (see page 5597)	Retrieves a value stored under a subkey of HKEY_CLASSES_ROOT from the system registry.
	GUIDToString (see page 5597)	This is function ComObj.GUIDToString.
	HandleSafeCallException (see page 5598)	This is function ComObj.HandleSafeCallException.
	InterfaceConnect (see page 5598)	Connects an IConnectionPoint interface.
	InterfaceDisconnect (see page 5599)	Disconnects an IConnectionPoint interface connection that was previously made by the InterfaceConnect (see page 5598) procedure.
	OleCheck (see page 5600)	Throws an EOleSysError (see page 5532) exception if the result code indicates an error.
	OleError (see page 5600)	Raises an EOleSysError (see page 5532) exception.
	ProgIDToClassID (see page 5601)	Returns the class ID (CLSID) GUID that corresponds to the string specified as the ProgID parameter.
	RegisterAsService (see page 5601)	Registers a COM object as an NT service.
	RegisterComServer (see page 5602)	Registers an in-process COM server with the operating system.
	SetDispatchPropValue (see page 5602)	Sets the value of a property on an IDispatch interface.
	SetDispatchPropValue (see page 5602)	Sets the value of a property on an IDispatch interface.
	StringToGUID (see page 5603)	This is function ComObj.StringToGUID.
	StringToLPOLESTR (see page 5603)	This is function ComObj.StringToLPOLESTR.

Interfaces

	Name	Description
	IServerExceptionHandler (see page 5588)	This is class ComObj.IServerExceptionHandler.

Structs, Records, Enums

	Name	Description
	TClassInstancing (see page 5604)	This is record ComObj.TClassInstancing.
	TThreadingModel (see page 5604)	This is record ComObj.TThreadingModel.

Types

Name	Description
TAutoClass (see page 5606)	This is type ComObj.TAutoClass.
TCoAddRefServerProcessProc (see page 5606)	This is type ComObj.TCoAddRefServerProcessProc.
TCoCreateInstanceExProc (see page 5607)	This is type ComObj.TCoCreateInstanceExProc.
TCoInitializeExProc (see page 5607)	This is type ComObj.TCoInitializeExProc.
TComClass (see page 5607)	This is type ComObj.TComClass.
TConnectEvent (see page 5608)	This is type ComObj.TConnectEvent.
TCoReleaseServerProcessProc (see page 5608)	This is type ComObj.TCoReleaseServerProcessProc.
TCoResumeClassObjectsProc (see page 5608)	This is type ComObj.TCoResumeClassObjectsProc.

TCoSuspendClassObjectsProc (see page 5609)	This is type ComObj.TCoSuspendClassObjectsProc.
TFactoryProc (see page 5609)	This is type ComObj.TFactoryProc.
TTypedComClass (see page 5609)	This is type ComObj.TTypedComClass.

Variables

Name	Description
CoAddRefServerProcess (see page 5610)	This is variable ComObj.CoAddRefServerProcess.
CoCreateInstanceEx (see page 5610)	This is variable ComObj.CoCreateInstanceEx.
CoInitFlags (see page 5611)	Specifies the level of threading support requested for a COM server .EXE.
CoInitializeEx (see page 5611)	This is variable ComObj.CoInitializeEx.
CoReleaseServerProcess (see page 5612)	This is variable ComObj.CoReleaseServerProcess.
CoResumeClassObjects (see page 5612)	This is variable ComObj.CoResumeClassObjects.
CoSuspendClassObjects (see page 5612)	This is variable ComObj.CoSuspendClassObjects.

28.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EOleError (see page 5528)	EOleError is the exception class for low-level OLE errors.
	EOleException (see page 5529)	EOleException is the exception class for OLE automation errors that occur in methods or properties of an object.
	EOleRegistrationError (see page 5531)	EOleRegistrationError is the exception class for errors that occur when registering an OLE object.
	EOleSysError (see page 5532)	EOleSysError is the exception class for errors specific to the OLE IDispatch interface.
	TAutoIntfObject (see page 5534)	TAutoIntfObject implements the IDispatch and ISupportErrorInfo interfaces.
	TAutoObject (see page 5540)	TAutoObject is a CoClass that supports the IDispatch interface, and that can be used as a base class for ActiveX Automation servers.
	TAutoObjectFactory (see page 5545)	TAutoObjectFactory is the class factory for TAutoObject (see page 5540).
	TComClassManager (see page 5550)	TComClassManager is the class type of the ComClassManager (see page 5591) variable in the ComObj (see page 5525) unit.
	TComObject (see page 5553)	TComObject is the base class for creating simple COM classes, such as those used for creating Shell extensions.
	TComObjectFactory (see page 5563)	TComObjectFactory is the type of the class factory object used to instantiate a TComObject (see page 5553) class.
	TComServerObject (see page 5575)	TComServerObject is the abstract base class for TComServer.
	TTypedComObject (see page 5581)	TTypedComObject supports the IProvideClassInfo interface and is the class type for COM objects that can provide the type information without requiring a type library to be loaded.
	TTypedComObjectFactory (see page 5583)	TTypedComObjectFactory is the type of the class factory object used to instantiate a TypedComObject class.

28.1.1 EOleError Class

EOleError is the exception class for low-level OLE errors.

Class Hierarchy

Pascal

```
EOleError = class(Exception);
```

C++

```
class EOleError : public Exception;
```

File

ComObj

Description

EOleError is raised when the system returns a non-zero HRESULT involving low-level OLE errors or and old OLE 2 errors. The library checks for, but does not generate, these exceptions.

See Also

EOleException (see page 5529)

EOleSysError (see page 5532)

TObject (see page 22285)

TComObject (see page 5553)

28.1.2 EOleException Class

EOleException is the exception class for OLE automation errors that occur in methods or properties of an object.

Class Hierarchy

Pascal

```
EOleException = class(EOleSysError);
```

C++

```
class EOleException : public EOleSysError;
```

File

ComObj

Description

EOleException is the exception class for errors specific to the OLE IDispatch interface. EOleException is raised when an error occurs in the middle of a method call from IDispatch. EOleException is generated when a server receives a failure return value from a property or method that was called with the IDispatch interface Invoke method.

When an error occurs, objects invoked through IDispatch can return essential information about the exception, including the source of the error, error code, description, help file, and ID number for context-sensitive help. EOleException captures this information, and the error description and ID number (if available) are included in a message generated by the constructor; the other information is stored in the Source (see page 5531) and HelpFile (see page 5531) properties defined for EOleException and the ErrorCode (see page 5534) property inherited from EOleSysError (see page 5532).

See Also

Invoke

28.1.2.1 EOleException Members

The following tables list the members exposed by EOleException.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.

	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5530)	Creates an ole exception object.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	ErrorCode (see page 5534)	The status code returned by the failed method call, indicating the reason for the failure.
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	HelpFile (see page 5531)	Specifies the name and location of the help file that contains information about the error that occurred.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.
	Source (see page 5531)	Designates the application program in which the exception occurred.

28.1.2.2 EOleException.Create Constructor

Creates an ole exception object.

Pascal

```
constructor Create(const Message: string; ErrorCode: HRESULT; const Source: string; const
HelpFile: string; HelpContext: Integer);
```

C++

```
__fastcall EOleException(const AnsiString Message, HRESULT ErrorCode, const AnsiString Source,
```

```
const AnsiString HelpFile, int HelpContext);
```

Description

Create instantiates an EOleException (see page 5529) object.

Create allocates memory for the exception and initializes the ErrorCode (see page 5534), HelpFile (see page 5531), and Source (see page 5531) properties using information passed in the OLE EXCEPINFO structure. It then calls the inherited CreateHelp method to display the error description and ID number from EXCEPINFO.

See Also

Create (see page 5533)

28.1.2.3 EOleException Properties

The properties of the EOleException class are listed here.

Public Properties

	Name	Description
	HelpFile (see page 5531)	Specifies the name and location of the help file that contains information about the error that occurred.
	Source (see page 5531)	Designates the application program in which the exception occurred.

28.1.2.3.1 EOleException.HelpFile Property

Specifies the name and location of the help file that contains information about the error that occurred.

Pascal

```
property HelpFile: string;
```

C++

```
__property AnsiString HelpFile;
```

Description

HelpFile lists the name and complete directory path of the help file that contains information about the error.

28.1.2.3.2 EOleException.Source Property

Designates the application program in which the exception occurred.

Pascal

```
property Source: string;
```

C++

```
__property AnsiString Source;
```

Description

Source gives the source of the exception. Usually, this is the name of an application program.

28.1.3 EOleRegistrationError Class

EOleRegistrationError is the exception class for errors that occur when registering an OLE object.

Class Hierarchy

System::TObject → SysUtils.Exception → ComObj.EOleError → ComObj.EOleRegistrationError

Pascal

```
EOleRegistrationError = class(EOleError);
```

C++

```
class EOleRegistrationError : public EOleError;
```

File

ComObj

Description

EOleRegistrationError is raised by the CreateRegKey (see page 5593) method when it encounters an error while trying to register a COM class with the system registry.

28.1.4 EOleSysError Class

EOleSysError is the exception class for errors specific to the OLE IDispatch interface.

Class Hierarchy

System::TObject → SysUtils.Exception → ComObj.EOleError → ComObj.EOleSysError

Pascal

```
EOleSysError = class(EOleError);
```

C++

```
class EOleSysError : public EOleError;
```

File

ComObj

Description

EOleSysError is the exception class for errors that occur during an attempt to invoke a method or property of an OLE automation object. When the IDispatch Invoke method fails, EOleSysError is raised. Objects invoked through IDispatch can return the error code—an OLE status code or SCODE—indicating the reason for the error.

28.1.4.1 EOleSysError Members

The following tables list the members exposed by EOleSysError.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5533)	Creates an EOleSysError (see page 5532) object.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	ErrorCode (see page 5534)	The status code returned by the failed method call, indicating the reason for the failure.
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.

28.1.4.2 EOleSysError.Create Constructor

Creates an EOleSysError (see page 5532) object.

Pascal

```
constructor Create(const Message: string; ErrorCode: HRESULT; HelpContext: Integer);
```

C++

```
__fastcall EOleSysError(const AnsiString Message, HRESULT ErrorCode, int HelpContext);
```

Description

Create calls SysErrorMessage to turn the OLE error code, ErrorCode (see page 5534), into a message string that it passes to the inherited constructor. EOleSysError (see page 5532) initializes the ErrorCode (see page 5534) property to the

ErrorCode (see page 5534) string.

See Also

CreateHelp (see page 22755)

ErrorCode (see page 5534)

28.1.4.3 EOleSysError Properties

The properties of the EOleSysError class are listed here.

Public Properties

	Name	Description
	ErrorCode (see page 5534)	The status code returned by the failed method call, indicating the reason for the failure.

28.1.4.3.1 EOleSysError.ErrorCode Property

The status code returned by the failed method call, indicating the reason for the failure.

Pascal

```
property ErrorCode: HRESULT;
```

C++

```
__property HRESULT ErrorCode;
```

Description

ErrorCode is the numerical error code returned by a failed call to an OLE automation object's method or property.

ErrorCode is initialized in the constructor.

See Also

Create (see page 5533)

28.1.5 TAutoIntfObject Class

TAutoIntfObject implements the IDispatch and ISupportErrorInfo interfaces.

Class Hierarchy

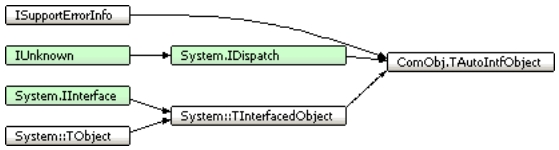

Pascal

```
TAutoIntfObject = class(TInterfacedObject, IDispatch, ISupportErrorInfo);
```

C++

```
class TAutoIntfObject : public TInterfacedObject, public IDispatch, public ISupportErrorInfo;
```

File

ComObj

Description

TAutoIntfObject is used internally by objects that support dispatch interfaces. By implementing the IDispatch interface, TAutoIntfObject encapsulates the functionality of an Automation object.

TAutoIntfObject does not have a class factory, so it can only be instantiated by calling the constructor. Objects that are externally instantiated typically have a class factory so that they can be instantiated in a standard way.

TAutoIntfObject supports OLE exception handling through its implementation of the ISupportErrorInfo interface.

TAutoIntfObject is similar to TAutoObject (see page 5540) in that both classes implement the IDispatch and (directly or indirectly) ISupportErrorInfo interfaces and both require a type library to support dual interfaces. TAutoIntfObject differs from TAutoObject (see page 5540) in that it does not have a class factory.

GetIDsOfNames (see page 5537), GetTypeInfo (see page 5537), GetTypeInfoCount (see page 5537), Invoke (see page 5538).

See Also

TAutoObject (see page 5540)

TAutoObjectFactory (see page 5545)

28.1.5.1 TAutoIntfObject Members

The following tables list the members exposed by TAutoIntfObject.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetIDsOfNames (see page 5537)	This is GetIDsOfNames, a member of class TAutoIntfObject.
	GetTypeInfo (see page 5537)	This is GetTypeInfo, a member of class TAutoIntfObject.
	GetTypeInfoCount (see page 5537)	This is GetTypeInfoCount, a member of class TAutoIntfObject.
	InterfaceSupportsErrorInfo (see page 5538)	This is InterfaceSupportsErrorInfo, a member of class TAutoIntfObject.
	Invoke (see page 5538)	This is Invoke, a member of class TAutoIntfObject.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5536)	Create instantiates an automation interface object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 5538)	SafeCallException is used for OLE exception handling.

Public Properties

	Name	Description
R	DispIID (see page 5539)	DispIID is the dispatch identifier.
R	DispIntfEntry (see page 5539)	DispIntfEntry contains the entry for the dispatch interface implemented by the automation object.
R	DispTypeInfo (see page 5539)	DispTypeInfo returns an ITypeInfo interface reference to the type information for the dispinterface implemented by the automation object.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

28.1.5.2 TAutoIntfObject.Create Constructor

Create instantiates an automation interface object.

Pascal

```
constructor Create(const TypeLib: ITypeLib; const DispIntf: TGUID);
```

C++

```
__fastcall TAutoIntfObject(const ITypeLib TypeLib, const TGUID DispIntf);
```

Description

You must use the constructor to instantiate TAutoIntfObject (see page 5534) descendants because there is no built-in class factory support.

TypeLib is an ITypeLib interface that can return the type information for DispIntf.

DispIntf is the GUID for the object you are creating.

Create allocates memory for an automation interface object, uses TypeLib to access the type information for DispIntf, and initializes the DispIntfEntry (see page 5539) property.

28.1.5.3 TAutoIntfObject Methods

The methods of the TAutoIntfObject class are listed here.

Protected Methods

	Name	Description
	GetIDsOfNames (see page 5537)	This is GetIDsOfNames, a member of class TAutoIntfObject.
	GetTypeInfo (see page 5537)	This is GetTypeInfo, a member of class TAutoIntfObject.
	GetTypeInfoCount (see page 5537)	This is GetTypeInfoCount, a member of class TAutoIntfObject.
	InterfaceSupportsErrorInfo (see page 5538)	This is InterfaceSupportsErrorInfo, a member of class TAutoIntfObject.
	Invoke (see page 5538)	This is Invoke, a member of class TAutoIntfObject.

Public Methods

	Name	Description
	SafeCallException (see page 5538)	SafeCallException is used for OLE exception handling.

28.1.5.3.1 TAutoIntfObject.GetIDsOfNames Method

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int NameCount, int
LocaleID, void * DispIDs);
```

Description

This is GetIDsOfNames, a member of class TAutoIntfObject.

28.1.5.3.2 TAutoIntfObject.GetTypeInfo Method

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

This is GetTypeInfo, a member of class TAutoIntfObject.

28.1.5.3.3 TAutoIntfObject.GetTypeInfoCount Method

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

This is GetTypeInfoCount, a member of class TAutoIntfObject.

28.1.5.3.4 TAutoIntfObject.InterfaceSupportsErrorInfo Method

Pascal

```
function InterfaceSupportsErrorInfo(const iid: TIID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT InterfaceSupportsErrorInfo(const TIID iid);
```

Description

This is InterfaceSupportsErrorInfo, a member of class TAutoIntfObject.

28.1.5.3.5 TAutoIntfObject.Invoke Method

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word Flags,
Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is Invoke, a member of class TAutoIntfObject.

28.1.5.3.6 TAutoIntfObject.SafeCallException Method

SafeCallException is used for OLE exception handling.

Pascal

```
function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): HResult; override;
```

C++

```
virtual __fastcall HRESULT SafeCallException(TObject * ExceptObject, void * ExceptAddr);
```

Description

SafeCallException prevents an exception from escaping its scope. Any exceptions raised must also be handled within the method.

The implementation of a safecall method guarantees that no exceptions escape by enclosing the body of the method in an implied exception block.

Note: SafeCallException calls the global routine HandleSafeCallException (see page 5598), which provides a standard way of handling this behavior. Methods of a CoClass that supports OLE exception handling (that implements the ISupportErrorInfo interface) are affected by this function. If those CoClass methods use the safecall calling convention, they are modified to return an HRESULT. Procedures will return an HRESULT, and functions will pass their usual result value as the last (out) parameter, and will return an HRESULT instead. The method is enclosed in an exception block, after which the HRESULT is returned, and the environment is set up so that the ISupportErrorInfo implementation will work correctly.

28.1.5.4 TAutoIntfObject Properties

The properties of the TAutoIntfObject class are listed here.

Public Properties

	Name	Description
R	DispIID (see page 5539)	DispIID is the dispatch identifier.
R	DispIntfEntry (see page 5539)	DispIntfEntry contains the entry for the dispatch interface implemented by the automation object.
R	DispTypeInfo (see page 5539)	DispTypeInfo returns an ITypeInfo interface reference to the type information for the dispinterface implemented by the automation object.

28.1.5.4.1 TAutoIntfObject.DispIID Property

DispIID is the dispatch identifier.

Pascal

```
property DispIID: TGUID;
```

C++

```
__property TGUID DispIID;
```

Description

DispIID is used for exception handling.

28.1.5.4.2 TAutoIntfObject.DispIntfEntry Property

DispIntfEntry contains the entry for the dispatch interface implemented by the automation object.

Pascal

```
property DispIntfEntry: PInterfaceEntry;
```

C++

```
__property PInterfaceEntry DispIntfEntry;
```

Description

DispIntfEntry is initialized in the constructor. DispIntfEntry returns the entry for a dispatch interface.

See Also

Create (see page 5536)

28.1.5.4.3 TAutoIntfObject.DispTypeInfo Property

DispTypeInfo returns an ITypeInfo interface reference to the type information for the dispinterface implemented by the automation object.

Pascal

```
property DispTypeInfo: ITypeInfo;
```

C++

```
__property ITypeInfo DispTypeInfo;
```

Description

DispTypeInfo is used to provide type information on the automation object.

28.1.6 TAutoObject Class

TAutoObject is a CoClass that supports the IDispatch interface, and that can be used as a base class for ActiveX Automation servers.

Class Hierarchy

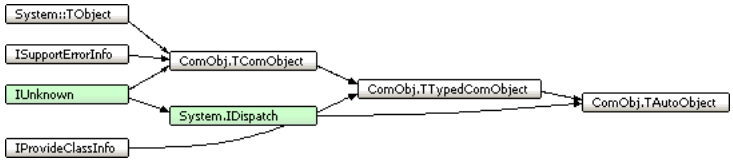

Pascal

```
TAutoObject = class(TTypedComObject, IDispatch);
```

C++

```
class TAutoObject : public TTypedComObject, public IDispatch;
```

File

ComObj

Description

TAutoObject is used as a base class for ActiveX control classes and provides support for ActiveX Automation objects.

To create a new Automation object, use the Automation wizard. The steps for creating an Automation object are covered in depth in the Developer's Guide or in the online document, Developing COM-based applications.

By implementing the IDispatch interface, TAutoObject encapsulates the following functionality:

Provides access to properties and methods exposed by an object by calling a method or by accessing a property of a specified dispatch interface (dispinterface), given any necessary parameters.

Converts text names of properties and methods (including their arguments) to their corresponding DISPIDs which identify them.

Determines whether there is type information available for this dispatch interface.

Retrieves the type information for a specified dispatch interface.

TAutoObject requires a type library because it supports dual interfaces, and because it inherits the IProvideClassInfo interface support which requires a type library. TAutoObject has a class factory. For creating automation objects that are used internally and do not need a class factory, use TAutoIntfObject (see page 5534) as a base class.

GetIDsOfNames (see page 5543), GetTypeInfo (see page 5543), GetTypeInfoCount (see page 5544), and Invoke (see page 5544).

See Also

TAutoIntfObject (see page 5534)

TAutoObjectFactory (see page 5545)

Creating an Active Server Object

Code Generated by Wizards

28.1.6.1 TAutoObject Members

The following tables list the members exposed by TAutoObject.

Protected Methods

	Name	Description
	_AddRef (see page 5558)	This is _AddRef, a member of class TComObject.
	_Release (see page 5558)	This is _Release, a member of class TComObject.
	EventConnect (see page 5542)	This is EventConnect, a member of class TAutoObject.
V	EventSinkChanged (see page 5543)	This is EventSinkChanged, a member of class TAutoObject.
	GetClassInfo (see page 5583)	Returns a pointer to a TTypedComObject (see page 5581)'s type information.
V	GetIDsOfNames (see page 5543)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetTypeInfo (see page 5543)	Retrieves the type information for an object.
V	GetTypeInfoCount (see page 5544)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InterfaceSupportsErrorInfo (see page 5559)	This is InterfaceSupportsErrorInfo, a member of class TComObject.
V	Invoke (see page 5544)	Provides access to Automation properties and methods exposed by an object that supports the IDispatch interface.
	QueryInterface (see page 5560)	This is QueryInterface, a member of class TComObject.

Protected Properties

	Name	Description
R	AutoFactory (see page 5545)	The class factory that created this TAutoObject (see page 5540).
	EventSink (see page 5545)	This is EventSink, a member of class TAutoObject.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5555)	Instantiates a COM object that is not part of an aggregate.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateAggregated (see page 5556)	Instantiates a COM object that is part of an aggregate.
	CreateFromFactory (see page 5556)	Instantiates a COM object and initializes its values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5557)	Disposes of an instance of a COM object.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 5544)	Provides an interface to a method that initializes the Automation object.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	ObjAddRef (see page 5559)	Implements the AddRef method of the IUnknown interface.
V	ObjQueryInterface (see page 5559)	Implements the QueryInterface (see page 5560) method of the IUnknown interface.
V	ObjRelease (see page 5559)	Implements the Release method of the IUnknown interface.
V	SafeCallException (see page 5560)	Handles OLE exceptions.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Controller (see page 5561)	Specifies the controlling IUnknown interface that controls interfaces within an aggregate object.
R	Factory (see page 5561)	Specifies the class factory object for the TComObject (see page 5553) class.
R	RefCount (see page 5562)	Indicates the number of references to the COM object that are currently in use.
	ServerExceptionHandler (see page 5562)	Provides access to the SafeCall exception notification process.

28.1.6.2 TAutoObject Methods

The methods of the TAutoObject class are listed here.

Protected Methods

	Name	Description
	EventConnect (see page 5542)	This is EventConnect, a member of class TAutoObject.
V	EventSinkChanged (see page 5543)	This is EventSinkChanged, a member of class TAutoObject.
V	GetIDsOfNames (see page 5543)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetTypeInfo (see page 5543)	Retrieves the type information for an object.
V	GetTypeInfoCount (see page 5544)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	Invoke (see page 5544)	Provides access to Automation properties and methods exposed by an object that supports the IDispatch interface.

Public Methods

	Name	Description
V	Initialize (see page 5544)	Provides an interface to a method that initializes the Automation object.

28.1.6.2.1 TAutoObject.EventConnect Method

Pascal

```
procedure EventConnect(const Sink: IUnknown; Connecting: Boolean);
```

C++

```
__fastcall EventConnect(const IUnknown Sink, Boolean Connecting);
```

Description

This is EventConnect, a member of class TAutoObject.

28.1.6.2.2 TAutoObject.EventSinkChanged Method

Pascal

```
procedure EventSinkChanged(const EventSink: IUnknown); virtual;
```

C++

```
virtual __fastcall EventSinkChanged(const IUnknown EventSink);
```

Description

This is EventSinkChanged, a member of class TAutoObject.

28.1.6.2.3 TAutoObject.GetIDsOfNames Method

Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int
NameCount, int LocaleID, void * DispIDs);
```

Description

GetIDsOfNames implements the IDispatch interface GetIDsOfNames method. GetIDsOfNames calls this method for the interface represented by the TAutoObject (see page 5540), passing the specified parameters. The returned value can be used on subsequent calls to the Invoke (see page 5544) method.

See Also

GetTypeInfo (see page 5543)

GetTypeInfoCount (see page 5544)

Invoke (see page 5544)

28.1.6.2.4 TAutoObject.GetTypeInfo Method

Retrieves the type information for an object.

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; virtual;
stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

GetTypeInfo implements the IDispatch interface GetTypeInfo method. GetTypeInfo calls the GetTypeInfo method for the interface for the interface represented by the TAutoObject (see page 5540), passing it the specified parameters. Use the returned value to get the type information for an interface implemented by the component.

See Also

GetIDsOfNames (see page 5543)

GetTypeInfoCount (see page 5544)

Invoke (see page 5544)

28.1.6.2.5 TAutoObject.GetTypeInfoCount Method

Returns the number of type information interfaces that an object provides (either 0 or 1).

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

GetTypeInfoCount implements the IDispatch interface GetTypeInfoCount method. GetTypeInfoCount calls this method for the interface represented by the TAutoObject (see page 5540). The out parameter points to a location that receives the number of type information interfaces provided by the object. If the object provides type information, this number is 1; otherwise the number is 0.

See Also

GetIDsOfNames (see page 5543)

GetTypeInfo (see page 5543)

Invoke (see page 5544)

28.1.6.2.6 TAutoObject.Initialize Method

Provides an interface to a method that initializes the Automation object.

Pascal

```
procedure Initialize; override;
```

C++

```
virtual __fastcall Initialize();
```

Description

Initialize currently does nothing. Descendant Automation objects can override Initialize to provide some post-creation initialization of the object.

28.1.6.2.7 TAutoObject.Invoke Method

Provides access to Automation properties and methods exposed by an object that supports the IDispatch interface.

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word
Flags,  Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

Invoke implements the IDispatch interface Invoke method. Invoke is the standard mechanism for accessing the exposed properties and methods of an Automation object. Invoke calls the Invoke method for the interface represented by the TAutoObject (see page 5540), passing it the parameters specified by the function.

See Also

GetIDsOfNames (see page 5543)

GetTypeInfo (see page 5543)

GetTypeInfoCount (see page 5544)

28.1.6.3 TAutoObject Properties

The properties of the TAutoObject class are listed here.

Protected Properties

	Name	Description
R	AutoFactory (see page 5545)	The class factory that created this TAutoObject (see page 5540).
	EventSink (see page 5545)	This is EventSink, a member of class TAutoObject.

28.1.6.3.1 TAutoObject.AutoFactory Property

The class factory that created this TAutoObject (see page 5540).

Pascal

```
property AutoFactory: TAutoObjectFactory;
```

C++

```
__property TAutoObjectFactory AutoFactory;
```

Description

Factory (see page 5561) is the class factory that is used to instantiate a TAutoObject (see page 5540) instance.

28.1.6.3.2 TAutoObject.EventSink Property

Pascal

```
property EventSink: IUnknown;
```

C++

```
__property IUnknown EventSink;
```

Description

This is EventSink, a member of class TAutoObject.

28.1.7 TAutoObjectFactory Class

TAutoObjectFactory is the class factory for TAutoObject (see page 5540).

Class Hierarchy

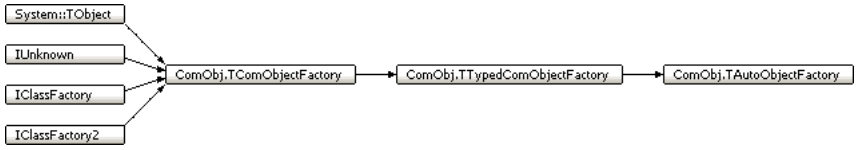

Pascal

```
TAutoObjectFactory = class(TTypedComObjectFactory);
```

C++

```
class TAutoObjectFactory : public TTypedComObjectFactory;
```

File

ComObj

Description

TAutoObjectFactory creates instances of the TAutoObject (see page 5540) class, and has methods that manage the functionality of all of the instances of the automation objects it creates.

TAutoObjectFactory inherits support for the IClassFactory and IClassFactory2 interfaces.

Note: In Delphi applications, an Automation object is a COM object, which means that the module in which it is implemented must provide a factory object so that the system can create the Automation object. To add an Automation object factory to your program, create the factory object in the initialization section of the unit that defines the Automation object. For more information about COM factories, see TComObjectFactory (see page 5563).

See Also

TAutoObject (see page 5540)

TComObjectFactory (see page 5563)

28.1.7.1 TAutoObjectFactory Members

The following tables list the members exposed by TAutoObjectFactory.

Protected Methods

	Name	Description
	_AddRef (see page 5567)	This is _AddRef, a member of class TComObjectFactory.
	_Release (see page 5567)	This is _Release, a member of class TComObjectFactory.
	CreateInstance (see page 5568)	This is CreateInstance, a member of class TComObjectFactory.
	CreateInstanceLic (see page 5568)	This is CreateInstanceLic, a member of class TComObjectFactory.
V	GetLicenseString (see page 5568)	This is GetLicenseString, a member of class TComObjectFactory.
	GetLicInfo (see page 5568)	This is GetLicInfo, a member of class TComObjectFactory.
V	GetProgID (see page 5569)	This is GetProgID, a member of class TComObjectFactory.
V	HasMachineLicense (see page 5569)	This is HasMachineLicense, a member of class TComObjectFactory.
	LockServer (see page 5569)	This is LockServer, a member of class TComObjectFactory.
	QueryInterface (see page 5569)	This is QueryInterface, a member of class TComObjectFactory.
	RequestLicKey (see page 5570)	This is RequestLicKey, a member of class TComObjectFactory.
V	ValidateUserLicense (see page 5571)	This is ValidateUserLicense, a member of class TComObjectFactory.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5548)	This is Create, a member of class TAutoObjectFactory.
V	CreateComObject (see page 5567)	Instantiates a TComObject (see page 5553) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5566)	Disposes of an instance of a class factory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetInterfaceTypeInfo (see page 5586)	Retrieves the type information for the typed COM object created by the factory.
V	GetIntfEntry (see page 5548)	Returns the entry for a specific interface implemented in a class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RegisterClassObject (see page 5569)	Registers the COM class associated with the factory object.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	UpdateRegistry (see page 5570)	Registers COM objects created by the factory.
V	UpdateRegistry (see page 5586)	Registers the typed COM object class.

Public Properties

	Name	Description
R	ClassID (see page 5571)	This is ClassID, a member of class TComObjectFactory.
R	ClassInfo (see page 5587)	Specifies an interface reference that provides type information without having to load a type library.
R	ClassName (see page 5572)	Specifies the name of the class that the class factory instantiates.
R	ComClass (see page 5572)	This is ComClass, a member of class TComObjectFactory.
R	ComServer (see page 5572)	This is ComServer, a member of class TComObjectFactory.
R	Description (see page 5572)	Describes the class object.
R	DispIntfEntry (see page 5549)	Contains the entry for the dispatch interface implemented by the OLE Automation object created by the factory.
R	DispTypeInfo (see page 5549)	Returns an ITypeInfo interface reference to the type information for the dispinterface implemented by the OLE Automation object.
	ErrorIID (see page 5573)	Indicates the interface of the CoClass in which an error occurred.
R	EventIID (see page 5549)	This is EventIID, a member of class TAutoObjectFactory.
R	EventTypeInfo (see page 5550)	This is EventTypeInfo, a member of class TAutoObjectFactory.
R	Instancing (see page 5573)	Indicates the process space of COM object instances.
	LicString (see page 5573)	Contains licensing information about the object.
R	ProgID (see page 5574)	Specifies the name of the class factory.
	ShowErrors (see page 5574)	Indicates whether the COM object factory displays a message box when exceptions are raised.
	SupportsLicensing (see page 5574)	Indicates whether the object being instantiated supports licensing.
R	ThreadingModel (see page 5575)	Indicates the type of thread support provided by objects that the factory creates.

28.1.7.2 TAutoObjectFactory.Create Constructor

Pascal

```
constructor Create(ComServer: TComServerObject; AutoClass: TAutoClass; const ClassID: TGUID;
Instancing: TClassInstancing; ThreadingModel: TThreadingModel = tmSingle);
```

C++

```
__fastcall TAutoObjectFactory(TComServerObject ComServer, TAutoClass AutoClass, const TGUID
ClassID, TClassInstancing Instancing, TThreadingModel ThreadingModel = tmSingle);
```

Description

This is Create, a member of class TAutoObjectFactory.

28.1.7.3 TAutoObjectFactory Methods

The methods of the TAutoObjectFactory class are listed here.

Public Methods

	Name	Description
	GetIntfEntry (see page 5548)	Returns the entry for a specific interface implemented in a class.

28.1.7.3.1 TAutoObjectFactory.GetIntfEntry Method

Returns the entry for a specific interface implemented in a class.

Pascal

```
function GetIntfEntry(Guid: TGUID): PInterfaceEntry; virtual;
```

C++

```
virtual __fastcall PInterfaceEntry GetIntfEntry(TGUID Guid);
```

Description

GetIntfEntry calls the GetInterfaceEntry method for the CoClass associated with the class factory.

GetIntfEntry returns the entry for the interface, specified by the IID parameter, in the class. GetIntfEntry is used internally. It is rarely called directly.

One use for this method is to describe to OLE how to automatically dispatch calls to a dual-IDispatch interface. An interface type name, such as IUnknown, can be supplied as the IID parameter since the compiler will replace the interface with the associated GUID.

See Also

GetInterfaceEntry

28.1.7.4 TAutoObjectFactory Properties

The properties of the TAutoObjectFactory class are listed here.

Public Properties

	Name	Description
R	DispIntfEntry (see page 5549)	Contains the entry for the dispatch interface implemented by the OLE Automation object created by the factory.
R	DispTypeInfo (see page 5549)	Returns an ITypeInfo interface reference to the type information for the dispinterface implemented by the OLE Automation object.
R	EventIID (see page 5549)	This is EventIID, a member of class TAutoObjectFactory.
R	EventTypeInfo (see page 5550)	This is EventTypeInfo, a member of class TAutoObjectFactory.

28.1.7.4.1 TAutoObjectFactory.DispIntfEntry Property

Contains the entry for the dispatch interface implemented by the OLE Automation object created by the factory.

Pascal

```
property DispIntfEntry: PInterfaceEntry;
```

C++

```
__property PInterfaceEntry DispIntfEntry;
```

Description

DispIntfEntry is initialized in the constructor. It indicates the entry for a dispatch interface.

See Also

Create

28.1.7.4.2 TAutoObjectFactory.DispTypeInfo Property

Returns an ITypeInfo interface reference to the type information for the dispinterface implemented by the OLE Automation object.

Pascal

```
property DispTypeInfo: ITypeInfo;
```

C++

```
__property ITypeInfo DispTypeInfo;
```

Description

DispTypeInfo is used to provide type information on the OLE Automation object created by the factory.

28.1.7.4.3 TAutoObjectFactory.EventIID Property

Pascal

```
property EventIID: TGUID;
```

C++

```
__property TGUID EventIID;
```

Description

This is EventIID, a member of class TAutoObjectFactory.

28.1.7.4.4 TAutoObjectFactory.EventTypeInfo Property

Pascal

```
property EventTypeInfo: ITypeInfo;
```

C++

```
__property ITypeInfo EventTypeInfo;
```

Description

This is EventTypeInfo, a member of class TAutoObjectFactory.

28.1.8 TComClassManager Class

TComClassManager is the class type of the ComClassManager (see page 5591) variable in the ComObj (see page 5525) unit.

Class Hierarchy

Pascal

```
TComClassManager = class(TObject);
```

C++

```
class TComClassManager : public TObject;
```

File

ComObj

Description

An instance of TComClassManager is returned by the ComClassManager (see page 5591) function in the ComObj (see page 5525) unit. This instance of TComClassManager manages the class factory objects owned by COM servers.

TComClassManager is a class designed for internal use to keep track of class factories.

For all servers linked to the module containing this unit, the manager returned by ComClassManager (see page 5591) maintains a list of pointers to the class factory objects. This list is updated by the individual COM servers, which add class factories objects when they are created and delete them when they are destroyed. The ComClass manager has methods that are used to manipulate class factories based upon their associated servers. These methods process iterative actions on factories or return factories based upon the class object type they create.

See Also

ComClassManager

TComServer (see page 5623)

28.1.8.1 TComClassManager Members

The following tables list the members exposed by TComClassManager.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5551)	Creates an instance of the TComClassManager (see page 5550) class.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 5552)	Destroys an instance of TComClassManager (see page 5550).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	ForEachFactory (see page 5552)	Processes iterative actions on factories associated with a given server.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetFactoryFromClass (see page 5553)	Returns the class factory for a given COM class.
	GetFactoryFromClassID (see page 5553)	Returns the class factory for a given COM class identifier.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

28.1.8.2 TComClassManager.Create Constructor

Creates an instance of the TComClassManager (see page 5550) class.

Pascal

```
constructor Create;
```

C++

```
__fastcall TComClassManager();
```

Description

Do not create instances of TComClassManager (see page 5550). Instead, use the ComClassManager (see page 5591) function to obtain a reference to the global COM class manager.

See Also

comclassmanager

28.1.8.3 TComClassManager.Destroy Destructor

Destroys an instance of TComClassManager (see page 5550).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TComClassManager();
```

Description

Do not destroy the COM class manager. This is a global instance that is managed by Delphi.

See Also

Free (see page 22300)

28.1.8.4 TComClassManager Methods

The methods of the TComClassManager class are listed here.

Public Methods

	Name	Description
	ForEachFactory (see page 5552)	Processes iterative actions on factories associated with a given server.
	GetFactoryFromClass (see page 5553)	Returns the class factory for a given COM class.
	GetFactoryFromClassID (see page 5553)	Returns the class factory for a given COM class identifier.

28.1.8.4.1 TComClassManager.ForEachFactory Method

Processes iterative actions on factories associated with a given server.

Pascal

```
procedure ForEachFactory(ComServer: TComServerObject; FactoryProc: TFactoryProc);
```

C++

```
__fastcall ForEachFactory(TComServerObject ComServer, TFactoryProc FactoryProc);
```

Description

ForEachFactory is used to process an action, specified by the FactoryProc parameter, on all class factories in the factory list that are owned by the ComServer specified by the ComServer parameter.

ForEachFactory is used internally by the COM server to:

Destroy (see page 5552) factories when the associated COM server unloads.

Have factories register their class objects in the system registry, when the associated COM server initializes.

Have the factories update the system registry when necessary.

TFactoryProc (see page 5609) is the method-pointer type of the FactoryProc parameter of ForEachProperty. TFactoryProc (see page 5609) defines a method that takes a class factory as a parameter.

See Also

TComObjectFactory (see page 5563)

TComServerObject (see page 5575)

28.1.8.4.2 TComClassManager.GetFactoryFromClass Method

Returns the class factory for a given COM class.

Pascal

```
function GetFactoryFromClass(ComClass: TClass): TComObjectFactory;
```

C++

```
__fastcall TComObjectFactory GetFactoryFromClass(TClass ComClass);
```

Description

GetFactoryFromClass returns the class factory associated with the COM class specified by the ComClass parameter. If no such class factory is found in the list maintained by the ComClass manager, an EOleError (see page 5528) exception is raised indicating that the class factory for the specified COM class is not in the current list.

See Also

EOleError (see page 5528)

28.1.8.4.3 TComClassManager.GetFactoryFromClassID Method

Returns the class factory for a given COM class identifier.

Pascal

```
function GetFactoryFromClassID(const ClassID: TGUID): TComObjectFactory;
```

C++

```
__fastcall TComObjectFactory GetFactoryFromClassID(const TGUID ClassID);
```

Description

GetFactoryFromClassID is used internally by DllGetClassObject to obtain a reference to the IClassFactory interface of the corresponding class factory object for a given class object specified by the ClassID parameter.

GetFactoryFromClassID returns the class factory associated with the COM class specified by the ClassID parameter. If the class cannot be found, GetFactoryFromClassID returns nil (Delphi) or NULL (C++).

See Also

DllGetClassObject

28.1.9 TComObject Class

TComObject is the base class for creating simple COM classes, such as those used for creating Shell extensions.

Class Hierarchy

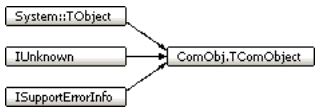

Pascal

```
TComObject = class(TObject, IUnknown, ISupportErrorInfo);
```

C++

```
class TComObject : public TObject, public IUnknown, public ISupportErrorInfo;
```

File

ComObj

Description

TComObject is a COM object that supports the IUnknown and ISupportErrorInfo interfaces. TComObject provides the implementation for simple COM objects that have:

A class identifier (CLSID) for external instantiation using a class factory.

Optional support for aggregation implemented through the IUnknown methods.

Support for the SafeCall calling convention and OLE exception-handling by implementing IProvideErrorInfo.

Support for the the IErrorInfo mechanism.

TComObject can be used as a base class for creating classes for COM objects that must have a class identifier (CLSID). CLSIDs are used to register the class in the database registry and to externally instantiate the class using a class factory. The class factory for TComObject is TComObjectFactory (see page 5563). The class factory properties provide information about TComObject class, such as its name, description, CLSID, and so on. The TComObjectFactory (see page 5563) methods are used to register the TComObject class in the registry and to instantiate it.

TComObject can be instantiated either as a single COM object, or as part of an aggregate. The IUnknown methods are implemented in TComObject to support aggregation by appropriately delegating to the controlling IUnknown interface when the instantiated COM object is the inner object of an aggregate. Consequently, for all interfaces that are implemented by the inner COM object, its reference count will not be directly affected when an interface reference is created.

TComObject supports the ISupportErrorInfo interface which allows OLE Automation controllers to query whether an error object will be available, and ensures that error information can be propagated correctly up the call chain. By implementing the ISupportErrorInfo interface method, InterfaceSupportsErrorInfo (see page 5559), TComObject supports the IErrorInfo mechanism, thereby providing support for OLE exception-handling and the safecall calling convention. The information available through the IErrorInfo interface is used for SafeCall error handling.

Using TComObject will not require the instantiated object to have a type library.

InterfaceSupportsErrorInfo (see page 5559) method

See Also

TComObjectFactory (see page 5563)

28.1.9.1 TComObject Members

The following tables list the members exposed by TComObject.

Protected Methods

	Name	Description
	_AddRef (see page 5558)	This is _AddRef, a member of class TComObject.
	_Release (see page 5558)	This is _Release, a member of class TComObject.
	InterfaceSupportsErrorInfo (see page 5559)	This is InterfaceSupportsErrorInfo, a member of class TComObject.
	QueryInterface (see page 5560)	This is QueryInterface, a member of class TComObject.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5555)	Instantiates a COM object that is not part of an aggregate.
	CreateAggregated (see page 5556)	Instantiates a COM object that is part of an aggregate.
	CreateFromFactory (see page 5556)	Instantiates a COM object and initializes its values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5557)	Disposes of an instance of a COM object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 5558)	Provides an interface to a method that initializes the COM object.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	ObjAddRef (see page 5559)	Implements the AddRef method of the IUnknown interface.
V	ObjQueryInterface (see page 5559)	Implements the QueryInterface (see page 5560) method of the IUnknown interface.
V	ObjRelease (see page 5559)	Implements the Release method of the IUnknown interface.
V	SafeCallException (see page 5560)	Handles OLE exceptions.

Public Properties

	Name	Description
R	Controller (see page 5561)	Specifies the controlling IUnknown interface that controls interfaces within an aggregate object.
R	Factory (see page 5561)	Specifies the class factory object for the TComObject (see page 5553) class.
R	RefCount (see page 5562)	Indicates the number of references to the COM object that are currently in use.
	ServerExceptionHandler (see page 5562)	Provides access to the SafeCall exception notification process.

28.1.9.2 TComObject.Create Constructor

Instantiates a COM object that is not part of an aggregate.

Pascal

```
constructor Create;
```

C++

```
__fastcall TComObject();
```

Description

Call Create to instantiate a COM object that is not part of an aggregate, at runtime.

Create allocates memory for a COM object, and then directly calls the CreateFromFactory (see page 5556) constructor, passing it nil as the Controller (see page 5561) parameter to indicate that the COM object is not being created as part of an aggregate.

See Also

Controller (see page 5561)

CreateFromFactory (see page 5556)

28.1.9.3 TComObject.CreateAggregated Constructor

Instantiates a COM object that is part of an aggregate.

Pascal

```
constructor CreateAggregated(const Controller: IUnknown);
```

C++

```
__fastcall TComObject(const IUnknown Controller);
```

Description

Call CreateAggregated to instantiate a COM object that is part of an aggregate by passing it the IUnknown control object as the Controller (see page 5561) parameter.

CreateAggregated allocates memory for a COM object and then calls the CreateFromFactory (see page 5556) constructor. Because the newly instantiated COM object is part of an aggregate, CreateAggregated passes Controller (see page 5561) to CreateFromFactory (see page 5556) as its Controller (see page 5561) parameter. Controller (see page 5561) also becomes the value of the Controller (see page 5561) property of the COM object.

See Also

Controller (see page 5561)

CreateFromFactory (see page 5556)

28.1.9.4 TComObject.CreateFromFactory Constructor

Instantiates a COM object and initializes its values.

Pascal

```
constructor CreateFromFactory(Factory: TComObjectFactory; const Controller: IUnknown);
```

C++

```
__fastcall TComObject(TComObjectFactory Factory, const IUnknown Controller);
```

Description

Call Create (see page 5555) to instantiate a COM object that is not part of an aggregate, at runtime.

Create (see page 5555) allocates memory for a COM object, and then initializes the following properties:

Sets RefCount (see page 5562) to 1, as a means of artificial reference counting in the case that the newly created object has a zero reference count and is passed as a parameter.

Sets Factory (see page 5561) to the Factory (see page 5561) parameter, which is the instance of the class factory responsible for creating the object.

Sets Controller (see page 5561) to the Controller (see page 5561) parameter, which is either a control object, if the object is part of an aggregate, or nil if it is not.

Calls on the COM server to increment the reference count of objects dependent on it.

Calls Initialize (see page 5558).

Decrements the RefCount (see page 5562) that was artificially incremented.

See Also

Controller (see page 5561)

Factory (see page 5561)

Initialize (see page 5558)

RefCount (see page 5562)

comserver (see page 5572)

TComServerObject (see page 5575)

ObjectCount (see page 5631)

28.1.9.5 TComObject.Destroy Destructor

Disposes of an instance of a COM object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TComObject();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the COM object is not nil before it calls Destroy.

Destroy checks to see if the Factory (see page 5561) property is nil. If it is nil, then Destroy passes False to the CountObject method of the Factory (see page 5561) object's ComServer, allowing the DLL (server) to be unloaded if no more objects are dependent upon the server.

See Also

Factory (see page 5561)

comserver (see page 5572)

ObjectCount (see page 5631)

Free (see page 22300)

28.1.9.6 TComObject Methods

The methods of the TComObject class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 5558)	This is _AddRef, a member of class TComObject.
	_Release (see page 5558)	This is _Release, a member of class TComObject.
	InterfaceSupportsErrorInfo (see page 5559)	This is InterfaceSupportsErrorInfo, a member of class TComObject.
	QueryInterface (see page 5560)	This is QueryInterface, a member of class TComObject.

Public Methods

	Name	Description
V	Initialize (see page 5558)	Provides an interface to a method that initializes the COM object.
V	ObjAddRef (see page 5559)	Implements the AddRef method of the IUnknown interface.
V	ObjQueryInterface (see page 5559)	Implements the QueryInterface (see page 5560) method of the IUnknown interface.
V	ObjRelease (see page 5559)	Implements the Release method of the IUnknown interface.
V	SafeCallException (see page 5560)	Handles OLE exceptions.

28.1.9.6.1 TComObject._AddRef Method

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

This is _AddRef, a member of class TComObject.

28.1.9.6.2 TComObject._Release Method

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

This is _Release, a member of class TComObject.

28.1.9.6.3 TComObject.Initialize Method

Provides an interface to a method that initializes the COM object.

Pascal

```
procedure Initialize; virtual;
```

C++

```
virtual __fastcall Initialize();
```

Description

Initialize currently does nothing. Descendant COM objects can override Initialize to provide some post-creation initialization of the object.

See Also

Create_c

CreateFromFactory (see page 5556)

28.1.9.6.4 TComObject.InterfaceSupportsErrorInfo Method

Pascal

```
function InterfaceSupportsErrorInfo(const iid: TIID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT InterfaceSupportsErrorInfo(const TIID iid);
```

Description

This is InterfaceSupportsErrorInfo, a member of class TComObject.

28.1.9.6.5 TComObject.ObjAddRef Method

Implements the AddRef method of the IUnknown interface.

Pascal

```
function ObjAddRef: Integer; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall int ObjAddRef();
```

Description

ObjAddRef is used as the implementation of the IUnknown AddRef method, incrementing the reference count for the object.

28.1.9.6.6 TComObject.ObjQueryInterface Method

Implements the QueryInterface (see page 5560) method of the IUnknown interface.

Pascal

```
function ObjQueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT ObjQueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface (see page 5560) queries whether the COM object supports a given interface specified by the IID parameter. QueryInterface (see page 5560) is used as the implementation of the IUnknown QueryInterface (see page 5560) method when the COM object (TComObject (see page 5553)) is not part of an aggregate.

28.1.9.6.7 TComObject.ObjRelease Method

Implements the Release method of the IUnknown interface.

Pascal

```
function ObjRelease: Integer; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall int ObjRelease();
```

Description

ObjRelease is used as the implementation of the IUnknown Release method. It decrements the reference count for the object. If the reference count reaches 0, it automatically frees the object.

28.1.9.6.8 TComObject.QueryInterface Method

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

This is QueryInterface, a member of class TComObject.

28.1.9.6.9 TComObject.SafeCallException Method

Handles OLE exceptions.

Pascal

```
function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): HResult; override;
```

C++

```
virtual __fastcall HRESULT SafeCallException(TObject * ExceptObject, void * ExceptAddr);
```

Description

SafeCallException prevents an exception from being raised to escape its scope. Any exceptions raised must also be handled within the method.

The implementation of a safecall method guarantees that no exceptions are raised by enclosing the statement part of the method in an implied exception block.

Note: SafeCallException calls the global routine HandleSafeCallException (see page 5598), which provides a standard way of handling this behavior. Methods of a CoClass that supports OLE exception handling (that implements the ISupportErrorInfo interface) are affected by this function. If those CoClass methods use the safecall calling convention, they are modified to return an HRESULT. Procedures will return an HRESULT, and functions will pass their usual result value as the last (out) parameter, and will return an HRESULT instead. The method is enclosed in a exception block, after which the HRESULT is returned, and the environment is set up so that the ISupportErrorInfo implementation will work correctly.

28.1.9.7 TComObject Properties

The properties of the TComObject class are listed here.

Public Properties

	Name	Description
R	Controller (see page 5561)	Specifies the controlling IUnknown interface that controls interfaces within an aggregate object.
R	Factory (see page 5561)	Specifies the class factory object for the TComObject (see page 5553) class.
R	RefCount (see page 5562)	Indicates the number of references to the COM object that are currently in use.
	ServerExceptionHandler (see page 5562)	Provides access to the SafeCall exception notification process.

28.1.9.7.1 TComObject.Controller Property

Specifies the controlling IUnknown interface that controls interfaces within an aggregate object.

Pascal

```
property Controller: IUnknown;
```

C++

```
__property IUnknown Controller;
```

Description

Controller is the controlling IUnknown interface for the inner object in an aggregate. The QueryInterface (see page 5560), AddRef, and Release methods of all other interfaces of the inner object (except IUnknown) must delegate to the controlling IUnknown. Because of this delegation, the inner object's reference count is not affected when these methods are called.

Controller is nil (Delphi) or NULL (C++) if the object is not part of an aggregate. If Controller is not nil (Delphi) or NULL (C++), the object is part of an aggregate, and Controller is the controlling interface.

See Also

Create_c

CreateAggregated (see page 5556)

TAggregatedObject (see page 22248)

TComObjectFactory (see page 5563)

TContainedObject (see page 22253)

28.1.9.7.2 TComObject.Factory Property

Specifies the class factory object for the TComObject (see page 5553) class.

Pascal

```
property Factory: TComObjectFactory;
```

C++

```
__property TComObjectFactory Factory;
```

Description

Factory is the class factory which is used to instantiate a TComObject (see page 5553).

See Also

TComClassManager (see page 5550)

TComObjectFactory (see page 5563)

28.1.9.7.3 TComObject.RefCount Property

Indicates the number of references to the COM object that are currently in use.

Pascal

```
property RefCount: Integer;
```

C++

```
__property int RefCount;
```

Description

RefCount is used to determine when the object can be destroyed. RefCount is the number of references to interfaces implemented by the COM object. When RefCount is zero, the object can be is destroyed.

RefCount is automatically incremented when references to interfaces implemented by the COM object are created, and is automatically decremented when those references are destroyed. RefCount is usually incremented by the IUnknown AddRef method and decremented by the IUnknown method, Release.

RefCount can be artificially incremented when a new object is created which may be passed as a function parameter. In this case, RefCount should not be zero, and should be incremented prior to the function call and decremented when the function returns.

Note: In Delphi code, the CreateFromFactory (see page 5556) method artificially increment and decrements RefCount.

Note: In C++ code, if you pass a class factory to the constructor as the first parameter, the constructor artificially increments and decrements RefCount.

See Also

CreateFromFactory (see page 5556)

Create_c

AddRef

Release

28.1.9.7.4 TComObject.ServerExceptionHandler Property

Provides access to the SafeCall exception notification process.

Pascal

```
property ServerExceptionHandler: IServerExceptionHandler;
```

C++

```
__property IServerExceptionHandler ServerExceptionHandler;
```

Description

The ServerExceptionHandler property provides a way for you to access the SafeCall exception notification process so that you can log or reroute those notifications to an external monitor. The monitor can be another object within the same COM server module, or it could be on a different machine entirely.

For example, server exception handlers could be implemented to:

Log COM server exceptions to the NT system event log

Write exceptions to a text file

Send email to an administrator

Alert support personnel through a pager

The ServerExceptionHandler property is useful for administering remote servers, particularly if a server is physically unreachable. For example, when an exception is raised in a COM object, if the server application is a VCL forms-based application (not just a DLL), then the VCL default exception handler may show a message box on the server. No client requests on that COM object instance can be serviced until the message box is closed. By reporting the exception to an event log, others can see the information from any remote location, as long as the remote users have access rights on the server.

Tip: The ServerExceptionHandler property would be particularly useful for COM objects (either stand-alone or those running in the MTS environment), that are used in active server pages. You can use the ServerExceptionHandler property to write exception information in HTML or in any other format you desire.

See Also

CreateAggregated (see page 5556)

TAggregatedObject (see page 22248)

TComObjectFactory (see page 5563)

TContainedObject (see page 22253)

28.1.10 TComObjectFactory Class

TComObjectFactory is the type of the class factory object used to instantiate a TComObject (see page 5553) class.

Class Hierarchy

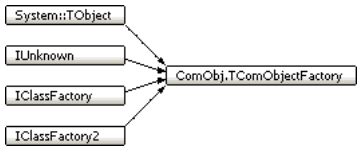

Pascal

```
TComObjectFactory = class(TObject, IUnknown, IClassFactory, IClassFactory2);
```

C++

```
class TComObjectFactory : public TObject, public IUnknown, public IClassFactory, public
IClassFactory2;
```

File

ComObj

Description

TComObjectFactory supports the IUnknown, IClassFactory, and IClassFactory2 interfaces. The IClassFactory interface is implemented by class objects associated with class identifiers (CLSIDs) for the classes that they instantiate. IClassFactory must be implemented for every object definition, so that objects of that type can be created. The IClassFactory2 interface is implemented by class factories for classes that need support for licensing.

Class factories instantiate the COM classes specified by a registered class identifier (CLSID). Class factory objects are created automatically by the COM servers that contain the code for the class objects and their associated classes. When a class factory is created, the CLSID is specified in a parameter to the constructor of the class factory object. This CLSID indicates the type of object for which the class factory creates instances.

Class factory objects can also be externally instantiated by directly calling the Windows API function CoCreateClassObject. The functions CreateComObject (see page 5567) or CreateOleObject (see page 5592) can also be called to instantiate a COM class. These functions indirectly use class factories to instantiate the classes. When creating multiple instances of a class that has a CLSID, however, it is more efficient to instantiate a class factory, obtain a pointer to its IClassFactory interface, and use its methods as needed.

Class factories are managed on the server by a global class manager accessed using the ComClassManager (see page 5591) function. This class manager maintains a list of class factories through which the factories are manipulated, usually according to the COM server that owns them.

IUnknown, IClassFactory, IClassFactory2

See Also

ComClassManager (see page 5591)

CreateComObject (see page 5592)

CreateOleObject (see page 5592)

TComObject (see page 5553)

28.1.10.1 TComObjectFactory Members

The following tables list the members exposed by TComObjectFactory.

Protected Methods

	Name	Description
	_AddRef (see page 5567)	This is _AddRef, a member of class TComObjectFactory.
	_Release (see page 5567)	This is _Release, a member of class TComObjectFactory.
	CreateInstance (see page 5568)	This is CreateInstance, a member of class TComObjectFactory.
	CreateInstanceLic (see page 5568)	This is CreateInstanceLic, a member of class TComObjectFactory.
V	GetLicenseString (see page 5568)	This is GetLicenseString, a member of class TComObjectFactory.
	GetLicInfo (see page 5568)	This is GetLicInfo, a member of class TComObjectFactory.
V	GetProgID (see page 5569)	This is GetProgID, a member of class TComObjectFactory.
V	HasMachineLicense (see page 5569)	This is HasMachineLicense, a member of class TComObjectFactory.
	LockServer (see page 5569)	This is LockServer, a member of class TComObjectFactory.
	QueryInterface (see page 5569)	This is QueryInterface, a member of class TComObjectFactory.
	RequestLicKey (see page 5570)	This is RequestLicKey, a member of class TComObjectFactory.
V	ValidateUserLicense (see page 5571)	This is ValidateUserLicense, a member of class TComObjectFactory.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5565)	Instantiates a class factory object.
V	CreateComObject (see page 5567)	Instantiates a TComObject (see page 5553) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5566)	Disposes of an instance of a class factory.

V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RegisterClassObject (see page 5569)	Registers the COM class associated with the factory object.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	UpdateRegistry (see page 5570)	Registers COM objects created by the factory.

Public Properties

	Name	Description
R	ClassID (see page 5571)	This is ClassID, a member of class TComObjectFactory.
R	ClassName (see page 5572)	Specifies the name of the class that the class factory instantiates.
R	ComClass (see page 5572)	This is ComClass, a member of class TComObjectFactory.
R	ComServer (see page 5572)	This is ComServer, a member of class TComObjectFactory.
R	Description (see page 5572)	Describes the class object.
	ErrorIID (see page 5573)	Indicates the interface of the CoClass in which an error occurred.
R	Instancing (see page 5573)	Indicates the process space of COM object instances.
	LicString (see page 5573)	Contains licensing information about the object.
R	ProgID (see page 5574)	Specifies the name of the class factory.
	ShowErrors (see page 5574)	Indicates whether the COM object factory displays a message box when exceptions are raised.
	SupportsLicensing (see page 5574)	Indicates whether the object being instantiated supports licensing.
R	ThreadingModel (see page 5575)	Indicates the type of thread support provided by objects that the factory creates.

28.1.10.2 TComObjectFactory.Create Constructor

Instantiates a class factory object.

Pascal

```
constructor Create(ComServer: TComServerObject; ComClass: TComClass; const ClassID: TGUID;
const ClassName: string; const Description: string; Instancing: TClassInstancing;
ThreadingModel: TThreadingModel = tmSingle);
```

C++

```
__fastcall TComObjectFactory(TComServerObject ComServer, TComClass ComClass, const TGUID
ClassID, const AnsiString ClassName, const AnsiString Description, TClassInstancing
Instancing, TThreadingModel ThreadingModel = tmSingle);
```

Description

Create instantiates a class factory, usually when the server is loaded. Class factories should be instantiated in the initialization section of the unit that includes their associated server, so that they are automatically available when the server is loaded.

Create allocates memory for a class factory object, and adds the object to the class factory list managed by the

ComClassManager (see page 5591). Then Create initializes the following properties with values passed to it:

Sets ComServer (see page 5572) to the ComServer (see page 5572) parameter.

Sets ComClass (see page 5572) to the ComClass (see page 5572) parameter.

Sets ClassID (see page 5571) to the ClassID (see page 5571) parameter.

Sets ClassName (see page 5572) to the ClassName (see page 5572) parameter.

Sets Description (see page 5572) to the Description (see page 5572) parameter.

Sets Instancing (see page 5573) to the Instancing (see page 5573) parameter.

Sets ThreadingModel (see page 5575) to the ThreadingModel (see page 5575) parameter.

Sets ErrorIID (see page 5573) to IUnknown.

See Also

ComClassManager

Instancing (see page 5573)

ThreadingModel (see page 5575)

28.1.10.3 TComObjectFactory.Destroy Destructor

Disposes of an instance of a class factory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TComObjectFactory();
```

Description

Do not call Destroy directly in an application. Class factories are automatically destroyed when their corresponding servers are unloaded.

See Also

Free (see page 22300)

28.1.10.4 TComObjectFactory Methods

The methods of the TComObjectFactory class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 5567)	This is _AddRef, a member of class TComObjectFactory.
	_Release (see page 5567)	This is _Release, a member of class TComObjectFactory.
	CreateInstance (see page 5568)	This is CreateInstance, a member of class TComObjectFactory.
	CreateInstanceLic (see page 5568)	This is CreateInstanceLic, a member of class TComObjectFactory.
V	GetLicenseString (see page 5568)	This is GetLicenseString, a member of class TComObjectFactory.
	GetLicInfo (see page 5568)	This is GetLicInfo, a member of class TComObjectFactory.
V	GetProgID (see page 5569)	This is GetProgID, a member of class TComObjectFactory.
V	HasMachineLicense (see page 5569)	This is HasMachineLicense, a member of class TComObjectFactory.

	LockServer (see page 5569)	This is LockServer, a member of class TComObjectFactory.
	QueryInterface (see page 5569)	This is QueryInterface, a member of class TComObjectFactory.
	RequestLicKey (see page 5570)	This is RequestLicKey, a member of class TComObjectFactory.
V	ValidateUserLicense (see page 5571)	This is ValidateUserLicense, a member of class TComObjectFactory.

Public Methods

	Name	Description
V	CreateComObject (see page 5567)	Instantiates a TComObject (see page 5553) object.
	RegisterClassObject (see page 5569)	Registers the COM class associated with the factory object.
V	UpdateRegistry (see page 5570)	Registers COM objects created by the factory.

28.1.10.4.1 TComObjectFactory._AddRef Method

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

This is _AddRef, a member of class TComObjectFactory.

28.1.10.4.2 TComObjectFactory._Release Method

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

This is _Release, a member of class TComObjectFactory.

28.1.10.4.3 TComObjectFactory.CreateComObject Method

Instantiates a TComObject (see page 5553) object.

Pascal

```
function CreateComObject(const Controller: IUnknown): TComObject; virtual;
```

C++

```
virtual __fastcall TComObject CreateComObject(const IUnknown Controller);
```

Description

CreateComObject is the method that instantiates the class type for which TComObjectFactory (see page 5563) is the class factory. CreateComObject calls the CreateFromFactory constructor of the TComObject (see page 5553) class to create a COM object. CreateComObject passes itself as the Factory parameter to CreateFromFactory.

CreateComClass creates a COM object based upon the value of the ComClass (see page 5572) property. ComClass (see page 5572) is the class identifier (CLSID) used by other applications to identify the COM class for which they request an object instance.

See Also

comclass (see page 5572)

CreateFromFactory (see page 5556)

TComObject (see page 5553)

28.1.10.4.4 TComObjectFactory.CreateInstance Method

Pascal

```
function CreateInstance(const UnkOuter: IUnknown; const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT CreateInstance(const IUnknown UnkOuter, const TGUID IID,  Obj);
```

Description

This is CreateInstance, a member of class TComObjectFactory.

28.1.10.4.5 TComObjectFactory.CreateInstanceLic Method

Pascal

```
function CreateInstanceLic(const unkOuter: IUnknown; const unkReserved: IUnknown; const iid:
TIID; const bstrKey: WideString; out vObject): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT CreateInstanceLic(const IUnknown unkOuter, const IUnknown
unkReserved, const TIID iid, const BSTR bstrKey,  vObject);
```

Description

This is CreateInstanceLic, a member of class TComObjectFactory.

28.1.10.4.6 TComObjectFactory.GetLicenseString Method

Pascal

```
function GetLicenseString: WideString; virtual;
```

C++

```
virtual __fastcall BSTR GetLicenseString();
```

Description

This is GetLicenseString, a member of class TComObjectFactory.

28.1.10.4.7 TComObjectFactory.GetLicInfo Method

Pascal

```
function GetLicInfo(var licInfo: TLicInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetLicInfo(TLicInfo licInfo);
```

Description

This is GetLicInfo, a member of class TComObjectFactory.

28.1.10.4.8 TComObjectFactory.GetProgID Method

Pascal

```
function GetProgID: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetProgID();
```

Description

This is GetProgID, a member of class TComObjectFactory.

28.1.10.4.9 TComObjectFactory.HasMachineLicense Method

Pascal

```
function HasMachineLicense: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean HasMachineLicense();
```

Description

This is HasMachineLicense, a member of class TComObjectFactory.

28.1.10.4.10 TComObjectFactory.LockServer Method

Pascal

```
function LockServer(fLock: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT LockServer(BOOL fLock);
```

Description

This is LockServer, a member of class TComObjectFactory.

28.1.10.4.11 TComObjectFactory.QueryInterface Method

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

This is QueryInterface, a member of class TComObjectFactory.

28.1.10.4.12 TComObjectFactory.RegisterClassObject Method

Registers the COM class associated with the factory object.

Pascal

```
procedure RegisterClassObject;
```

C++

```
__fastcall RegisterClassObject();
```

Description

RegisterClassObject registers an EXE class object with OLE so other applications can connect to it. EXE object applications should call RegisterClassObject on startup. It can also be used to register internal objects for use by the same EXE or other code (such as DLLs) that the EXE uses.

If RegisterClassObject is not successful, an EOleSysError (see page 5532) is raised with the following error code possibilities:

Value	Meaning
E_INVALIDARG	One or more the arguments are invalid.
E_OUTOFMEMORY	Not enough memory.
E_UNEXPECTED	An unexpected error occurred.
CO_E_OBJISREG	The class was already registered in the class object table.

Only EXE object applications call RegisterClassObject. Object handlers or DLL object applications do not call this function. Instead, they must implement and export the DllGetClassObject function.

At startup, a multiple-use EXE object application must create a class object (with the IClassFactory interface on it), and call RegisterClassObject to register the class object. Object applications that support several different classes (such as multiple types of embeddable objects) must allocate and register a different class object for each.

Multiple registrations of the same class object are independent and do not produce an error. Each subsequent registration yields a unique key.

See Also

DllGetClassObject

28.1.10.4.13 TComObjectFactory.RequestLicKey Method

Pascal

```
function RequestLicKey(dwResrved: Longint; out bstrKey: WideString): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT RequestLicKey(Longint dwResrved, BSTR bstrKey);
```

Description

This is RequestLicKey, a member of class TComObjectFactory.

28.1.10.4.14 TComObjectFactory.UpdateRegistry Method

Registers COM objects created by the factory.

Pascal

```
procedure UpdateRegistry(Register: Boolean); virtual;
```

C++

```
virtual __fastcall UpdateRegistry(Boolean Register);
```

Description

UpdateRegistry is called for EXEs when the EXE is run, and for DLLs or EXEs when the routines are called to specifically

register or unregister the server.

See Also

DLLRegisterServer

DLLUnregisterServer

RegisterClassObject (see page 5569)

28.1.10.4.15 TComObjectFactory.ValidateUserLicense Method

Pascal

```
function ValidateUserLicense(const LicStr: WideString): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ValidateUserLicense(const BSTR LicStr);
```

Description

This is ValidateUserLicense, a member of class TComObjectFactory.

28.1.10.5 TComObjectFactory Properties

The properties of the TComObjectFactory class are listed here.

Public Properties

	Name	Description
R	ClassID (see page 5571)	This is ClassID, a member of class TComObjectFactory.
R	ClassName (see page 5572)	Specifies the name of the class that the class factory instantiates.
R	ComClass (see page 5572)	This is ComClass, a member of class TComObjectFactory.
R	ComServer (see page 5572)	This is ComServer, a member of class TComObjectFactory.
R	Description (see page 5572)	Describes the class object.
	ErrorIID (see page 5573)	Indicates the interface of the CoClass in which an error occurred.
R	Instancing (see page 5573)	Indicates the process space of COM object instances.
	LicString (see page 5573)	Contains licensing information about the object.
R	ProgID (see page 5574)	Specifies the name of the class factory.
	ShowErrors (see page 5574)	Indicates whether the COM object factory displays a message box when exceptions are raised.
	SupportsLicensing (see page 5574)	Indicates whether the object being instantiated supports licensing.
R	ThreadingModel (see page 5575)	Indicates the type of thread support provided by objects that the factory creates.

28.1.10.5.1 TComObjectFactory.ClassID Property

Pascal

```
property ClassID: TGUID;
```

C++

```
__property TGUID ClassID;
```

Description

This is ClassID, a member of class TComObjectFactory.

28.1.10.5.2 TComObjectFactory.ClassName Property

Specifies the name of the class that the class factory instantiates.

Pascal

```
property ClassName: string;
```

C++

```
__property AnsiString ClassName;
```

Description

ClassName is the name identifying the COM class that the class factory instantiates.

ClassName is initialized, in the class factory constructor, to the class name that is passed to the constructor.

See Also

ClassIDToProgID

Create (see page 5565)

ProgIDToClassID

Create (see page 5565)

28.1.10.5.3 TComObjectFactory.ComClass Property

Pascal

```
property ComClass: TClass;
```

C++

```
__property TClass ComClass;
```

Description

This is ComClass, a member of class TComObjectFactory.

28.1.10.5.4 TComObjectFactory.ComServer Property

Pascal

```
property ComServer: TComServerObject;
```

C++

```
__property TComServerObject ComServer;
```

Description

This is ComServer, a member of class TComObjectFactory.

28.1.10.5.5 TComObjectFactory.Description Property

Describes the class object.

Pascal

```
property Description: string;
```

C++

```
__property AnsiString Description;
```

Description

Description is used in the registration database, when the class factory is registered, to describe the class object.

Description is initialized, in the class factory constructor, to the Description parameter that is passed to the constructor. This description is retrieved from the type library.

See Also

Create (see page 5565)

28.1.10.5.6 TComObjectFactory.ErrorIID Property

Indicates the interface of the CoClass in which an error occurred.

Pascal

```
property ErrorIID: TGUID;
```

C++

```
__property TGUID ErrorIID;
```

Description

Use ErrorIID to determine the interface in which an error occurred. For most CoClasses, this is the primary interface for that CoClass, as indicated in the type library.

See Also

Create (see page 5565)

28.1.10.5.7 TComObjectFactory.Instancing Property

Indicates the process space of COM object instances.

Pascal

```
property Instancing: TClassInstancing;
```

C++

```
__property TClassInstancing Instancing;
```

Description

Instancing determines the process space in which the factory can create COM objects and whether the factory can create multiple instances within a single process space. Instancing is initialized in the class factory constructor to the Instancing parameter that is passed to the constructor.

See Also

Create (see page 5565)

28.1.10.5.8 TComObjectFactory.LicString Property

Contains licensing information about the object.

Pascal

```
property LicString: WideString;
```

C++

```
__property BSTR LicString;
```

Description

LicString is used to verify the user has valid access to the object for development. LicString can be any string defined by the developer who is optionally licensing the object.

28.1.10.5.9 TComObjectFactory.ProgID Property

Specifies the name of the class factory.

Pascal

```
property ProgID: string;
```

C++

```
__property AnsiString ProgID;
```

Description

ProgID is the programmatic identifier that specifies the vendor, component, and version of the class factory object.

Every object class that can be inserted into an OLE container has ProgID.

See Also

ClassIDToProgID

ProgIDToClassID

28.1.10.5.10 TComObjectFactory.ShowErrors Property

Indicates whether the COM object factory displays a message box when exceptions are raised.

Pascal

```
property ShowErrors: Boolean;
```

C++

```
__property Boolean ShowErrors;
```

Description

When ShowErrors is true, exceptions that are raised when trying to create a COM object are caught and a message box appears to describe the error. When ShowErrors is false, the exception is not caught by the COM object factory.

28.1.10.5.11 TComObjectFactory.SupportsLicensing Property

Indicates whether the object being instantiated supports licensing.

Pascal

```
property SupportsLicensing: Boolean;
```

C++

```
__property Boolean SupportsLicensing;
```

Description

SupportsLicensing is used to determine whether a user has valid access to the object.

28.1.10.5.12 TComObjectFactory.ThreadingModel Property

Indicates the type of thread support provided by objects that the factory creates.

Pascal

```
property ThreadingModel: TThreadingModel;
```

C++

```
__property TThreadingModel ThreadingModel;
```

Description

The threading model of a factory is initialized in the constructor. The factory uses this value to indicate how COM should serialize calls to the objects that it creates. Use this value to determine the type of thread support you must provide in the implementation of the associated COM object.

See Also

Create (see page 5565)

28.1.11 TComServerObject Class

TComServerObject is the abstract base class for TComServer.

Class Hierarchy

Pascal

```
TComServerObject = class(TObject);
```

C++

```
class TComServerObject : public TObject;
```

File

ComObj

Description

TComServerObject is used as a base class for the TComServer class. All of its methods are abstract and protected. The TComServerObject introduces protected abstract methods that are overridden in descendant classes as accessor functions for its properties. These methods are implemented in the TComServer class. All of the properties of TComServerObject are read-only.

To use a COM server object, include the ComServ (see page 5621) unit in the project. This unit automatically instantiates the descendant TComServer class and assigns that object to the ComServ (see page 5621) variable in the ComServ (see page 5621) unit.

See Also

ComServer

TComServer (see page 5623)

28.1.11.1 TComServerObject Members

The following tables list the members exposed by TComServerObject.

Protected Methods

	Name	Description
	CountFactory (see page 5577)	This is CountFactory, a member of class TComServerObject.
	CountObject (see page 5577)	This is CountObject, a member of class TComServerObject.
	GetHelpFileName (see page 5577)	This is GetHelpFileName, a member of class TComServerObject.
	GetServerFileName (see page 5578)	This is GetServerFileName, a member of class TComServerObject.
	GetServerKey (see page 5578)	This is GetServerKey, a member of class TComServerObject.
	GetServerName (see page 5578)	This is GetServerName, a member of class TComServerObject.
	GetStartSuspended (see page 5578)	This is GetStartSuspended, a member of class TComServerObject.
	GetTypeLib (see page 5579)	This is GetTypeLib, a member of class TComServerObject.
	SetHelpFileName (see page 5579)	This is SetHelpFileName, a member of class TComServerObject.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	HelpFileName (see page 5579)	Specifies the name of the Help file associated with the module.
	ServerFileName (see page 5580)	Specifies the complete path and filename of the COM server.

R	ServerKey (see page 5580)	Indicates whether the COM server is an in-process server or whether it is a local server.
R	ServerName (see page 5580)	Specifies the name of the executable file for the server.
R	StartSuspended (see page 5580)	Specifies whether the server started in a suspended state.
R	TypeLib (see page 5581)	Provides access to the type library associated with the COM server.

28.1.11.2 TComServerObject Methods

The methods of the TComServerObject class are listed here.

Protected Methods

	Name	Description
A	CountFactory (see page 5577)	This is CountFactory, a member of class TComServerObject.
A	CountObject (see page 5577)	This is CountObject, a member of class TComServerObject.
A	GetHelpFileName (see page 5577)	This is GetHelpFileName, a member of class TComServerObject.
A	GetServerFileName (see page 5578)	This is GetServerFileName, a member of class TComServerObject.
A	GetServerKey (see page 5578)	This is GetServerKey, a member of class TComServerObject.
A	GetServerName (see page 5578)	This is GetServerName, a member of class TComServerObject.
A	GetStartSuspended (see page 5578)	This is GetStartSuspended, a member of class TComServerObject.
A	GetTypeLib (see page 5579)	This is GetTypeLib, a member of class TComServerObject.
A	SetHelpFileName (see page 5579)	This is SetHelpFileName, a member of class TComServerObject.

28.1.11.2.1 TComServerObject.CountFactory Method

Pascal

```
function CountFactory(Created: Boolean): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int CountFactory(Boolean Created) = 0;
```

Description

This is CountFactory, a member of class TComServerObject.

28.1.11.2.2 TComServerObject.CountObject Method

Pascal

```
function CountObject(Created: Boolean): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int CountObject(Boolean Created) = 0;
```

Description

This is CountObject, a member of class TComServerObject.

28.1.11.2.3 TComServerObject.GetHelpFileName Method

Pascal

```
function GetHelpFileName: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetHelpFileName() = 0;
```

Description

This is GetHelpFileName, a member of class TComServerObject.

28.1.11.2.4 TComServerObject.GetServerFileName Method

Pascal

```
function GetServerFileName: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetServerFileName() = 0;
```

Description

This is GetServerFileName, a member of class TComServerObject.

28.1.11.2.5 TComServerObject.GetServerKey Method

Pascal

```
function GetServerKey: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetServerKey() = 0;
```

Description

This is GetServerKey, a member of class TComServerObject.

28.1.11.2.6 TComServerObject.GetServerName Method

Pascal

```
function GetServerName: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetServerName() = 0;
```

Description

This is GetServerName, a member of class TComServerObject.

28.1.11.2.7 TComServerObject.GetStartSuspended Method

Pascal

```
function GetStartSuspended: Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean GetStartSuspended() = 0;
```

Description

This is GetStartSuspended, a member of class TComServerObject.

28.1.11.2.8 TComServerObject.GetTypeLib Method

Pascal

```
function GetTypeLib: ITypeLib; virtual; abstract;
```

C++

```
virtual __fastcall ITypeLib GetTypeLib() = 0;
```

Description

This is GetTypeLib, a member of class TComServerObject.

28.1.11.2.9 TComServerObject.SetHelpFileName Method

Pascal

```
procedure SetHelpFileName(const Value: string); virtual; abstract;
```

C++

```
virtual __fastcall SetHelpFileName(const AnsiString Value) = 0;
```

Description

This is SetHelpFileName, a member of class TComServerObject.

28.1.11.3 TComServerObject Properties

The properties of the TComServerObject class are listed here.

Public Properties

	Name	Description
	HelpFileName (see page 5579)	Specifies the name of the Help file associated with the module.
R	ServerFileName (see page 5580)	Specifies the complete path and filename of the COM server.
R	ServerKey (see page 5580)	Indicates whether the COM server is an in-process server or whether it is a local server.
R	ServerName (see page 5580)	Specifies the name of the executable file for the server.
R	StartSuspended (see page 5580)	Specifies whether the server started in a suspended state.
R	TypeLib (see page 5581)	Provides access to the type library associated with the COM server.

28.1.11.3.1 TComServerObject.HelpFileName Property

Specifies the name of the Help file associated with the module.

Pascal

```
property HelpFileName: string;
```

C++

```
__property AnsiString HelpFileName;
```

Description

The abstract accessor methods for HelpFileName are overridden in descendant classes.

28.1.11.3.2 TComServerObject.ServerFileName Property

Specifies the complete path and filename of the COM server.

Pascal

```
property ServerFileName: string;
```

C++

```
__property AnsiString ServerFileName;
```

Description

The abstract accessor methods for ServerFileName are overridden in descendant classes.

See Also

ServerName (see page 5580)

28.1.11.3.3 TComServerObject.ServerKey Property

Indicates whether the COM server is an in-process server or whether it is a local server.

Pascal

```
property ServerKey: string;
```

C++

```
__property AnsiString ServerKey;
```

Description

The abstract accessor methods for ServerKey are overridden in descendant classes.

28.1.11.3.4 TComServerObject.ServerName Property

Specifies the name of the executable file for the server.

Pascal

```
property ServerName: string;
```

C++

```
__property AnsiString ServerName;
```

Description

The abstract accessor methods for ServerName are overridden in descendant classes.

See Also

ServerFileName (see page 5580)

28.1.11.3.5 TComServerObject.StartSuspended Property

Specifies whether the server started in a suspended state.

Pascal

```
property StartSuspended: Boolean;
```

C++

```
__property Boolean StartSuspended;
```

Description

StartSuspended specifies whether the server started in a suspended state.

See Also

ServerName (see page 5580)

28.1.11.3.6 TComServerObject.TypeLib Property

Provides access to the type library associated with the COM server.

Pascal

```
property TypeLib: ITypeLib;
```

C++

```
__property ITypeLib TypeLib;
```

Description

The abstract accessor methods for TypeLib are overridden in descendant classes.

28.1.12 TTypedComObject Class

TTypedComObject supports the IProvideClassInfo interface and is the class type for COM objects that can provide the type information without requiring a type library to be loaded.

Class Hierarchy

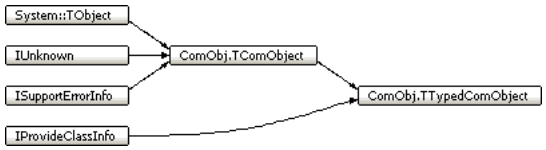

Pascal

```
TTypedComObject = class(TComObject, IProvideClassInfo);
```

C++

```
class TTypedComObject : public TComObject, public IProvideClassInfo;
```

File

ComObj

Description

TTypedComObject supports the IProvideClassInfo interface which gives an object the ability to describe its interfaces and types when everything else about it, such as its class identifier (CLSID), is unknown. TTypedComObject requires a type library for the IProvideClassInfo interface. This class is designed to require a type library so that it can support dual interfaces, which are created by default.

IProvideClassInfo has a single method, GetClassInfo (see page 5583), which is implemented in TTypedComObject to provide access to the type information for the object's CoClass entry in its type library. This means that if the TTypedComObject has a

running instance, a client can query for the IProvideClassInfo interface and retrieve this information without having to load and navigate a type library to find the ITypeInfo pointer for the object's CoClass.

Use TTypedComObject as a base class for objects that can conveniently provide type information without having to load a type library, such as an unknown object received through a drag-and-drop operation. The only objects that must implement IProvideClassInfo are those with custom events such as ActiveX controls.

Note: There is no support for type information in a particular language. Type information appropriate to the user's international settings on the current machine is assumed to have been loaded.

GetClassClassInfo.

See Also

Code Generated by Wizards

28.1.12.1 TTypedComObject Members

The following tables list the members exposed by TTypedComObject.

Protected Methods

	Name	Description
	_AddRef (see page 5558)	This is _AddRef, a member of class TComObject.
	_Release (see page 5558)	This is _Release, a member of class TComObject.
	GetClassInfo (see page 5583)	Returns a pointer to a TTypedComObject (see page 5581)'s type information.
	InterfaceSupportsErrorInfo (see page 5559)	This is InterfaceSupportsErrorInfo, a member of class TComObject.
	QueryInterface (see page 5560)	This is QueryInterface, a member of class TComObject.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5555)	Instantiates a COM object that is not part of an aggregate.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateAggregated (see page 5556)	Instantiates a COM object that is part of an aggregate.
	CreateFromFactory (see page 5556)	Instantiates a COM object and initializes its values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5557)	Disposes of an instance of a COM object.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 5558)	Provides an interface to a method that initializes the COM object.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	ObjAddRef (see page 5559)	Implements the AddRef method of the IUnknown interface.
V	ObjQueryInterface (see page 5559)	Implements the QueryInterface (see page 5560) method of the IUnknown interface.
V	ObjRelease (see page 5559)	Implements the Release method of the IUnknown interface.
V	SafeCallException (see page 5560)	Handles OLE exceptions.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Controller (see page 5561)	Specifies the controlling IUnknown interface that controls interfaces within an aggregate object.
R	Factory (see page 5561)	Specifies the class factory object for the TComObject (see page 5553) class.
R	RefCount (see page 5562)	Indicates the number of references to the COM object that are currently in use.
	ServerExceptionHandler (see page 5562)	Provides access to the SafeCall exception notification process.

28.1.12.2 TTypedComObject Methods

The methods of the TTypedComObject class are listed here.

Protected Methods

	Name	Description
	GetClassInfo (see page 5583)	Returns a pointer to a TTypedComObject (see page 5581)'s type information.

28.1.12.2.1 TTypedComObject.GetClassInfo Method

Returns a pointer to a TTypedComObject (see page 5581)'s type information.

Pascal

```
function GetClassInfo(out TypeInfo: ITypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetClassInfo(ITypeInfo TypeInfo);
```

Description

The GetClassInfo method returns a pointer to a type information object that describes the incoming and outgoing interfaces to the ActiveX control. The method gets a pointer to the control's type library, and through it, uses the ITypeLib interface to retrieve the type information.

28.1.13 TTypedComObjectFactory Class

TTypedComObjectFactory is the type of the class factory object used to instantiate a TypedComObject class.

Class Hierarchy

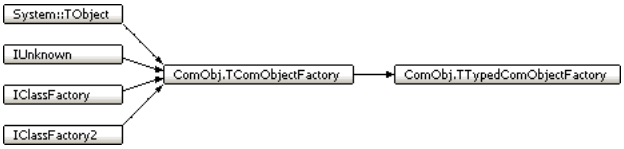

Pascal

```
TTypedComObjectFactory = class(TComObjectFactory);
```

C++

```
class TTypedComObjectFactory : public TComObjectFactory;
```

File

ComObj

Description

TTypedComObjectFactory inherits support for the IUnknown, IClassFactory, and IClassFactory2 interfaces. TTypedComObjectFactory is used to instantiate the TTypedComObject (see page 5581) class.

28.1.13.1 TTypedComObjectFactory Members

The following tables list the members exposed by TTypedComObjectFactory.

Protected Methods

	Name	Description
	_AddRef (see page 5567)	This is _AddRef, a member of class TComObjectFactory.
	_Release (see page 5567)	This is _Release, a member of class TComObjectFactory.
	CreateInstance (see page 5568)	This is CreateInstance, a member of class TComObjectFactory.
	CreateInstanceLic (see page 5568)	This is CreateInstanceLic, a member of class TComObjectFactory.
V	GetLicenseString (see page 5568)	This is GetLicenseString, a member of class TComObjectFactory.
	GetLicInfo (see page 5568)	This is GetLicInfo, a member of class TComObjectFactory.
V	GetProgID (see page 5569)	This is GetProgID, a member of class TComObjectFactory.
V	HasMachineLicense (see page 5569)	This is HasMachineLicense, a member of class TComObjectFactory.
	LockServer (see page 5569)	This is LockServer, a member of class TComObjectFactory.
	QueryInterface (see page 5569)	This is QueryInterface, a member of class TComObjectFactory.
	RequestLicKey (see page 5570)	This is RequestLicKey, a member of class TComObjectFactory.
V	ValidateUserLicense (see page 5571)	This is ValidateUserLicense, a member of class TComObjectFactory.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5585)	Instantiates a TTypedComObjectFactory (see page 5583) object.
V	CreateComObject (see page 5567)	Instantiates a TComObject (see page 5553) object.

V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5566)	Disposes of an instance of a class factory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetInterfaceTypeInfo (see page 5586)	Retrieves the type information for the typed COM object created by the factory.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RegisterClassObject (see page 5569)	Registers the COM class associated with the factory object.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	UpdateRegistry (see page 5586)	Registers the typed COM object class.

Public Properties

	Name	Description
R	ClassID (see page 5571)	This is ClassID, a member of class TComObjectFactory.
R	ClassInfo (see page 5587)	Specifies an interface reference that provides type information without having to load a type library.
R	ClassName (see page 5572)	Specifies the name of the class that the class factory instantiates.
R	ComClass (see page 5572)	This is ComClass, a member of class TComObjectFactory.
R	ComServer (see page 5572)	This is ComServer, a member of class TComObjectFactory.
R	Description (see page 5572)	Describes the class object.
	ErrorIID (see page 5573)	Indicates the interface of the CoClass in which an error occurred.
R	Instancing (see page 5573)	Indicates the process space of COM object instances.
	LicString (see page 5573)	Contains licensing information about the object.
R	ProgID (see page 5574)	Specifies the name of the class factory.
	ShowErrors (see page 5574)	Indicates whether the COM object factory displays a message box when exceptions are raised.
	SupportsLicensing (see page 5574)	Indicates whether the object being instantiated supports licensing.
R	ThreadingModel (see page 5575)	Indicates the type of thread support provided by objects that the factory creates.

28.1.13.2 TTypedComObjectFactory.Create Constructor

Instantiates a TTypedComObjectFactory (see page 5583) object.

Pascal

```
constructor Create(ComServer: TComServerObject; TypedComClass: TTypedComClass; const ClassID:
TGUID; Instancing: TClassInstancing; ThreadingModel: TThreadingModel = tmSingle);
```

C++

```
__fastcall TTypedComObjectFactory(TComServerObject ComServer, TTypedComClass TypedComClass,
const TGUID ClassID, TClassInstancing Instancing, TThreadingModel ThreadingModel = tmSingle);
```

Description

Create is used to instantiate a TTypedComObjectFactory (see page 5583). Create is automatically called when the COM server that contains the class definition for a TTypedComObject (see page 5581) is loaded. Class factories should be instantiated in the initialization section of the unit that includes their associated server.

Create allocates memory for a TTypedComObjectFactory (see page 5583), and verifies that a type library exists and is available, before calling the inherited Create. If the type library is not available, an EOleSysError (see page 5532) exception is raised.

See Also

EOleSysError (see page 5532)

Create (see page 5565)

28.1.13.3 TTypedComObjectFactory Methods

The methods of the TTypedComObjectFactory class are listed here.

Public Methods

	Name	Description
	GetInterfaceTypeInfo (see page 5586)	Retrieves the type information for the typed COM object created by the factory.
V	UpdateRegistry (see page 5586)	Registers the typed COM object class.

28.1.13.3.1 TTypedComObjectFactory.GetInterfaceTypeInfo Method

Retrieves the type information for the typed COM object created by the factory.

Pascal

```
function GetInterfaceTypeInfo(TypeFlags: Integer): ITypeInfo;
```

C++

```
__fastcall ITypeInfo GetInterfaceTypeInfo(int TypeFlags);
```

Description

GetInterfaceTypeInfo is called to retrieve type information on the TTypedComObject (see page 5581) instances that are created by the factory.

28.1.13.3.2 TTypedComObjectFactory.UpdateRegistry Method

Registers the typed COM object class.

Pascal

```
procedure UpdateRegistry(Register: Boolean); override;
```

C++

```
virtual __fastcall UpdateRegistry(Boolean Register);
```

Description

UpdateRegistry is called for EXEs when the EXE is run, and for DLLs or EXEs when the routines are called to specifically register or unregister the server.

See Also

DLLRegisterServer

DLLUnregisterServer

RegisterClassObject (see page 5569)

28.1.13.4 TTypedComObjectFactory Properties

The properties of the TTypedComObjectFactory class are listed here.

Public Properties

	Name	Description
	ClassInfo (see page 5587)	Specifies an interface reference that provides type information without having to load a type library.

28.1.13.4.1 TTypedComObjectFactory.ClassInfo Property

Specifies an interface reference that provides type information without having to load a type library.

Pascal

```
property ClassInfo: ITypeInfo;
```

C++

```
__property ITypeInfo ClassInfo;
```

Description

ClassInfo is of type ITypeInfo, which is an interface usually used for reading information about objects. Use ITypeInfo to extract information about the characteristics and capabilities of objects from type libraries.

Type information interfaces are intended to describe the parts of the application that can be called by outside clients, rather than those that might be used internally to build an application.

The ITypeInfo interface provides access to the following:

The set of function descriptions associated with the type; for interfaces, this contains the set of member functions in the interface.

The set of data member descriptions associated with the type; for structures, this contains the set of fields of the type.

The general attributes of the type, such as whether it describes a structure, an interface, and so on.

An ITypeInfo instance provides various information about a type, and is used in different ways. A compiler can use an ITypeInfo to compile references to members of the type. A type interface browser can use it to find information about each member of the type.

Note: An IDispatch implementer can use it to provide automatic delegation of IDispatch calls to an interface.

Note: A type library is a container for type descriptions. A type library contains descriptions of one or more objects, and is accessed through the ITypeLib interface. The ITypeLib interface provides access to information about the type description in a type library. The descriptions of individual objects are accessed through the ITypeInfo interface.

28.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IServerExceptionHandler (see page 5588)	This is class ComObj.IServerExceptionHandler.

28.2.1 IServerExceptionHandler Interface

Class Hierarchy

ComObj.IServerExceptionHandler

Pascal

```
IServerExceptionHandler = interface;
```

C++

```
__interface IServerExceptionHandler;
```

File

ComObj

Description

This is class ComObj.IServerExceptionHandler.

28.2.1.1 IServerExceptionHandler Members

The following tables list the members exposed by IServerExceptionHandler.

Public Methods

	Name	Description
	OnException (see page 5589)	This is OnException, a member of class IServerExceptionHandler.

28.2.1.2 IServerExceptionHandler Methods

The methods of the IServerExceptionHandler class are listed here.

Public Methods

	Name	Description
	OnException (see page 5589)	This is OnException, a member of class IServerExceptionHandler.

28.2.1.2.1 IServerExceptionHandler.OnException Method

Pascal

```
procedure OnException(const ServerClass: WideString; const ExceptionClass: WideString; const
ErrorMessage: WideString; ExceptAddr: Integer; const ErrorIID: WideString; const ProgID:
WideString; var Handled: Integer; var Result: HResult);
```

C++

```
__fastcall OnException(const BSTR ServerClass, const BSTR ExceptionClass, const BSTR
ErrorMessage, int ExceptAddr, const BSTR ErrorIID, const BSTR ProgID, int Handled, HRESULT
Result);
```

Description

This is OnException, a member of class IServerExceptionHandler.

28.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	ClassIDToProgID (see page 5590)	Returns the programmatic ID for a specified class given its class ID (CLSID).
	ComClassManager (see page 5591)	Returns a TComClassManager (see page 5550) object.
	CreateClassID (see page 5591)	CreateClassID generates a new GUID and returns it as a string.
	CreateComObject (see page 5592)	Instantiates a single instance of a COM object.
	CreateOleObject (see page 5592)	Instantiates an Automation object.
	CreateRegKey (see page 5593)	Creates or opens a registry key that is a subkey of HKEY_CLASSES_ROOT.
	CreateRemoteComObject (see page 5594)	Creates a Com object on another machine and returns an IUnknown interface for it.
	DeleteRegKey (see page 5595)	Deletes a single key that is a subkey of HKEY_CLASSES_ROOT from the system registry.
	DispatchInvoke (see page 5595)	This is function ComObj.DispatchInvoke.
	DispatchInvokeError (see page 5596)	This is function ComObj.DispatchInvokeError.
	GetActiveOleObject (see page 5596)	Retrieves a reference to an IDispatch interface to a currently running, registered COM object.
	GetDispatchPropValue (see page 5596)	Returns the value of a property on an IDispatch interface.
	GetDispatchPropValue (see page 5596)	Returns the value of a property on an IDispatch interface.
	GetRegStringValue (see page 5597)	Retrieves a value stored under a subkey of HKEY_CLASSES_ROOT from the system registry.
	GUIDToString (see page 5597)	This is function ComObj.GUIDToString.
	HandleSafeCallException (see page 5598)	This is function ComObj.HandleSafeCallException.
	InterfaceConnect (see page 5598)	Connects an IConnectionPoint interface.
	InterfaceDisconnect (see page 5599)	Disconnects an IConnectionPoint interface connection that was previously made by the InterfaceConnect (see page 5598) procedure.
	OleCheck (see page 5600)	Throws an EOleSysError (see page 5532) exception if the result code indicates an error.
	OleError (see page 5600)	Raises an EOleSysError (see page 5532) exception.
	ProgIDToClassID (see page 5601)	Returns the class ID (CLSID) GUID that corresponds to the string specified as the ProgID parameter.
	RegisterAsService (see page 5601)	Registers a COM object as an NT service.
	RegisterComServer (see page 5602)	Registers an in-process COM server with the operating system.
	SetDispatchPropValue (see page 5602)	Sets the value of a property on an IDispatch interface.
	SetDispatchPropValue (see page 5602)	Sets the value of a property on an IDispatch interface.
	StringToGUID (see page 5603)	This is function ComObj.StringToGUID.
	StringToLPOLESTR (see page 5603)	This is function ComObj.StringToLPOLESTR.

28.3.1 ComObj.ClassIDToProgID Function

Returns the programmatic ID for a specified class given its class ID (CLSID).

Pascal

```
function ClassIDToProgID(const ClassID: TGUID): string;
```

C++

```
AnsiString ClassIDToProgID(const TGUID ClassID);
```

File

ComObj

Description

ClassIDToProgID retrieves the programmatic ID for the CLSID specified by the ClassID parameter.

If ClassIDToProgID succeeds, it returns the requested ProgID. If ClassIDToProgID fails an EOleSysError (see page 5532) exception is raised with an error code indicating the cause of the error. The possible error codes are as follows:

Error code	Meaning
REGDB_E_CLASSNOTREG	Class not registered in the registry.
REGDB_E_READREGDB	Error reading registry.

See Also

EOleSysError (see page 5532)

OleError (see page 5600)

28.3.2 ComObj.ComClassManager Function

Returns a TComClassManager (see page 5550) object.

Pascal

```
function ComClassManager: TComClassManager;
```

C++

```
TComClassManager ComClassManager();
```

File

ComObj

Description

Use ComClassManager to obtain a reference to the global COM class manager. This COM class manager manages the class factory objects owned by COM servers.

See Also

TComServer

ComServer

28.3.3 ComObj.CreateClassID Function

CreateClassID generates a new GUID and returns it as a string.

Pascal

```
function CreateClassID: string;
```

C++

```
AnsiString CreateClassID();
```

File

ComObj

Description

Use CreateClassID to create a new GUID and access it as a string. A Delphi demo of the CreateClassID function is available in the Demos area.

See Also

CreateOleObject (see page 5592)

CreateRemoteComObject (see page 5594)

28.3.4 ComObj.CreateComObject Function

Instantiates a single instance of a COM object.

Pascal

```
function CreateComObject(const ClassID: TGUID): IUnknown;
```

C++

```
IUnknown CreateComObject(const TGUID ClassID);
```

File

ComObj

Description

CreateComObject creates a single uninitialized object of the class associated with the CLSID specified by the ClassID parameter. CreateComObject is used to create an object of a specified type when the CLSID is known, and when the object is on a local or in-proc server.

Note: In Delphi code, CreateComObject is called once to create each new single instance of a class. To create multiple instance of the same class it is recommended to use a class factory.

CreateComObject returns a reference to the interface to be used to communicate with the object. For CreateComObject this interface is of type IUnknown. To create OLE objects that return references to the IDispatch interface, use CreateOleObject (see page 5592).

Tip: If the interface returned by CreateCOMObject is assigned to a Variant, you can release the interface by assigning the Unassigned constant to that Variant.

See Also

CreateOleObject (see page 5592)

CreateRemoteComObject (see page 5594)

Unassigned

28.3.5 ComObj.CreateOleObject Function

Instantiates an Automation object.

Pascal

```
function CreateOleObject(const ClassName: string): IDispatch;
```

C++

```
IDispatch CreateOleObject(const AnsiString ClassName);
```

File

ComObj

Description

CreateOleObject creates a single uninitialized object of the class specified by the ClassName parameter. ClassName specifies the string representation of the Class ID (CLSID). CreateOleObject is used to create an object of a specified type when the CLSID is known, and when the object is on a local or in-proc server. Only objects that are not part of an aggregate are created using CreateOleObject.

Note: In Delphi code, CreateOleObject is called once to create each new single instance of a class. To create multiple instance of the same class, using a class factory is recommended.

CreateOleObject returns a reference to the interface that can be used to communicate with the object. For CreateOleObject this interface is of type IDispatch. To create a COM object that does not support an IDispatch interface, use CreateComObject (see page 5592).

Tip: As shown in the example, if the interface returned by CreateOleObject is assigned to a Variant, you can release the interface by assigning the Unassigned constant to that Variant.

See Also

CreateComObject (see page 5592)

Unassigned

Connecting to a Server

Controlling an Automation Server Using a Dispatch Interface

28.3.6 ComObj.CreateRegKey Function

Creates or opens a registry key that is a subkey of HKEY_CLASSES_ROOT.

Pascal

```
procedure CreateRegKey(const Key: string; const ValueName: string; const Value: string;
RootKey: DWord = HKEY_CLASSES_ROOT);
```

C++

```
CreateRegKey(const AnsiString Key, const AnsiString ValueName, const AnsiString Value, DWord
RootKey = HKEY_CLASSES_ROOT);
```

File

ComObj

Description

CreateRegKey creates or opens the key specified by the Key parameter. If the key does not already exist in the registry, CreateRegKey creates it, otherwise CreateRegKey opens it.

Note: CreateRegKey is useful for implementing the UpdateRegistry method of a class factory.

Key specifies a string that cannot begin with the backslash character ('\'). It must be a subkey of RootKey.

When the key is created, additional value and type information for the specified key can be set from the parameters passed to CreateRegKey.

ValueName points to a string that specifies the name of the subkey associated with value.

Value specifies the data to be stored with the specified value name.

RootKey is the top-level registry key under which the new key is created or opened. If omitted, it is HKEY_CLASSES_ROOT.

If CreateRegKey is not successful, an EOleRegistrationError (see page 5531) exception is raised.

See Also

EOleRegistrationError (see page 5531)

TRegistry

TRegistryIniFile

28.3.7 ComObj.CreateRemoteComObject Function

Creates a Com object on another machine and returns an IUnknown interface for it.

Pascal

```
function CreateRemoteComObject(const MachineName: WideString; const ClassID: TGUID): IUnknown;
```

C++

```
IUnknown CreateRemoteComObject(const BSTR MachineName, const TGUID ClassID);
```

File

ComObj

Description

Use CreateRemoteComObject to create an object on a remote server. CreateRemoteComObject creates a single uninitialized object of the class associated with the CLSID specified by the ClassID parameter. The MachineName parameter specifies the machine that lists the ClassID in its type library.

Usually, MachineName is the UNC name of the machine, which can be obtained by looking up its Machine Name under the Network settings in the Windows control panel. However, under some configurations of COM, MachineName is the IP address or Host name of the remote system.

CreateRemoteComObject returns a reference to the IUnknown interface for the object. Use this interface to obtain information about other interfaces supported by the object.

Tip: If the interface returned by CreateRemoteComObject is assigned to a Variant, you can release the interface by assigning the Unassigned constant to that Variant.

See Also

CreateComObject (see page 5592)

Unassigned

28.3.8 ComObj.DeleteRegKey Function

Deletes a single key that is a subkey of HKEY_CLASSES_ROOT from the system registry.

Pascal

```
procedure DeleteRegKey(const Key: string; RootKey: DWord = HKEY_CLASSES_ROOT);
```

C++

```
DeleteRegKey(const AnsiString Key, DWord RootKey = HKEY_CLASSES_ROOT);
```

File

ComObj

Description

Use DeleteRegKey to update the system registry.

Note: DeleteRegKey is useful for implementing the UpdateRegistry method of a class factory.

DeleteRegKey deletes the subkey of RootKey that is passed in as the Key parameter.

Key must specify a null-terminated string, which is the name of the key to delete. It cannot be NULL. The Key to delete must not have subkeys.

RootKey is the top-level key under which the key to delete appears. It defaults to HKEY_CLASSES_ROOT.

If DeleteRegKey succeeds it removes the entire specified key from the registry, including all of its values. An exception is raised if it fails.

See Also

CreateRegKey (see page 5593)

TRegistry

TRegistryIniFile

GetRegStringValue (see page 5597)

28.3.9 ComObj.DispatchInvoke Function

Pascal

```
procedure DispatchInvoke(const Dispatch: IDispatch; CallDesc: PCallDesc; DispIDs: PDispIDList;
Params: Pointer; Result: PVariant);
```

C++

```
DispatchInvoke(const IDispatch Dispatch, PCallDesc CallDesc, PDispIDList DispIDs, void *
Params, PVariant Result);
```

File

ComObj

Description

This is function ComObj.DispatchInvoke.

28.3.10 ComObj.DispatchInvokeError Function

Pascal

```
procedure DispatchInvokeError(Status: Integer; const ExcepInfo: TExcepInfo);
```

C++

```
DispatchInvokeError(int Status, const TExcepInfo ExcepInfo);
```

File

ComObj

Description

This is function ComObj.DispatchInvokeError.

28.3.11 ComObj.GetActiveOleObject Function

Retrieves a reference to an IDispatch interface to a currently running, registered COM object.

Pascal

```
function GetActiveOleObject(const ClassName: string): IDispatch;
```

C++

```
IDispatch GetActiveOleObject(const AnsiString ClassName);
```

File

ComObj

Description

GetActiveOleObject returns a reference to the IDispatch interface of the active object for the given class indicated by ClassName. ClassName is the programmatic ID for the class of the active object from the OLE running object table. GetActiveOleObject is used to access the OLE running object table. To add an object to this table, use the RegisterActiveObject API.

If either the the class object or the IDispatch interface are not successfully returned, an EOleSysError (see page 5532) exception is raised.

See Also

EOleSysError (see page 5532)

OleCheck (see page 5600)

28.3.12 ComObj.GetDispatchPropValue Function

Returns the value of a property on an IDispatch interface.

Pascal

```
function GetDispatchPropValue(Disp: IDispatch; DispID: Integer): OleVariant; overload;
```

```
function GetDispatchPropValue(Disp: IDispatch; Name: WideString): OleVariant; overload;
```

C++

```
OleVariant GetDispatchPropValue(IDispatch Disp, int DispID);
OleVariant GetDispatchPropValue(IDispatch Disp, BSTR Name);
```

File

ComObj

Description

Call GetDispatchPropValue to read the value of the property with the dispatch ID specified by DispID from the IDispatch interface given by Disp.

See Also

EnumDispatchProperties

SetDispatchPropValue (see page 5602)

28.3.13 ComObj.GetRegStringValue Function

Retrieves a value stored under a subkey of HKEY_CLASSES_ROOT from the system registry.

Pascal

```
function GetRegStringValue(const Key: string; const ValueName: string; RootKey: DWord =
HKEY_CLASSES_ROOT): string;
```

C++

```
AnsiString GetRegStringValue(const AnsiString Key, const AnsiString ValueName, DWord RootKey =
HKEY_CLASSES_ROOT);
```

File

ComObj

Description

GetRegStringValue returns the value specified by the ValueName property that is stored under the key specified by Key. Key must be a subkey of RootKey, which defaults to HKEY_CLASSES_ROOT. If there is no such value stored in the system registry, GetRegStringValue returns an empty string.

See Also

CreateRegKey (see page 5593)

TRegistry

TRegistryIniFile

28.3.14 ComObj.GUIDToString Function

Pascal

```
function GUIDToString(const ClassID: TGUID): string;
```

C++

```
AnsiString GUIDToString(const TGUID ClassID);
```

File

ComObj

Description

This is function ComObj.GUIDToString.

28.3.15 ComObj.HandleSafeCallException Function

Pascal

```
function HandleSafeCallException(ExceptObject: TObject; ExceptAddr: Pointer; const ErrorIID:
TGUID; const ProgID: WideString; const HelpFileName: WideString): HResult;
```

C++

```
HRESULT HandleSafeCallException(TObject * ExceptObject, void * ExceptAddr, const TGUID
ErrorIID, const BSTR ProgID, const BSTR HelpFileName);
```

File

ComObj

Description

This is function ComObj.HandleSafeCallException.

28.3.16 ComObj.InterfaceConnect Function

Connects an IConnectionPoint interface.

Pascal

```
procedure InterfaceConnect(const Source: IUnknown; const IID: TIID; const Sink: IUnknown; var
Connection: Longint);
```

C++

```
InterfaceConnect(const IUnknown Source, const TIID IID, const IUnknown Sink, Longint
Connection);
```

File

ComObj

Description

Call InterfaceConnect from a client application to register an object that implements an outgoing server connection. Typically, this object is an event sink that a client uses to respond to server events. When an event occurs, the server (event source) calls its outgoing interface for the event, and the client, which implements the interface as an event sink, receives the call

InterfaceConnect obtains an IConnectionPoint interface from a COM server advising on a particular connection point. IConnectionPoint lets the server expose an outgoing interface for an object, typically an event sink. IConnectionPointContainer enumerates the connection points supported by the server, so that a caller can find the right connection point. The IConnectionPointContainer and IConnectionPoint interfaces comprise the standard COM event-handling mechanism.

Source is an IUnknown interface for the server object that defines and calls the outgoing interface.

IID is the GUID of the outgoing interface.

Sink is the IUnknown interface of the client object that implements the outgoing interface.

Connection returns a token that represents the connection. It must be saved for use by the InterfaceDisconnect (see page 5599) procedure that is called to terminate the connection established by calling InterfaceConnect.

Note: A demo containing this procedure resides in Demos/ActiveX directory.

See Also

InterfaceDisconnect (see page 5599)

Handling Events in an Automation Controller

28.3.17 ComObj.InterfaceDisconnect Function

Disconnects an IConnectionPoint interface connection that was previously made by the InterfaceConnect (see page 5598) procedure.

Pascal

```
procedure InterfaceDisconnect(const Source: IUnknown; const IID: TIID; var Connection:
Longint);
```

C++

```
InterfaceDisconnect(const IUnknown Source, const TIID IID, Longint Connection);
```

File

ComObj

Description

Call InterfaceDisconnect from a client application to tell the server to stop calling its sink interface. InterfaceDisconnect disconnects an IConnectionPoint interface connection that was previously made by the InterfaceConnect (see page 5598) procedure. These procedures are wrappers for the COM event-handling mechanism, which is based on the IConnectionPointContainer and IConnectionPoint interfaces.

Source is an IUnknown interface for the server object that defines and calls the outgoing interface.

IID is the GUID of the outgoing interface.

Connection is the token that represents the connection, which was returned by the InterfaceConnect (see page 5598) procedure.

Note: A demo containing this procedure resides in Demos/ActiveX directory.

See Also

InterfaceConnect (see page 5598)

Handling Events in an Automation Controller

28.3.18 ComObj.OleCheck Function

Throws an EOleSysError (see page 5532) exception if the result code indicates an error.

Pascal

```
procedure OleCheck(Result: HResult);
```

C++

```
OleCheck(HRESULT Result);
```

File

ComObj

Description

OleCheck is used to wrap many COM routines, so that if that routine fails, users will have an opportunity to handle it in the resulting exception that is raised.

If ECode is a value less than zero, OleCheck raises an EOleSysError (see page 5532) exception that specifies the error code.

Delphi Examples:

```
{
OleCheck, GuidToString example
The following example manufactures a GUID and displays it in a dialog box after translating it
to a string.
}
uses ActiveX, ComObj;
procedure TForm1.Button1Click(Sender: TObject);
var
  G: TGUID;
begin
  OleCheck(CoCreateGuid(G));
  ShowMessage(SysUtils.GuidToString(G));
end;
```

See Also

EOleSysError (see page 5532)

OleError (see page 5600)

28.3.19 ComObj.OleError Function

Raises an EOleSysError (see page 5532) exception.

Pascal

```
procedure OleError(ErrorCode: HResult);
```

C++

```
OleError(HRESULT ErrorCode);
```

File

ComObj

Description

OleError raises an EOleSysError (see page 5532) exception from an error code passed in as the ErrorCode parameter.

OleError is used internally to catch failed function calls by throwing an EOleSysError (see page 5532) exception that indicates the cause of the failure as an error code. Unlike the OleCheck (see page 5600) procedure, which checks its parameter to determine whether it represents an error, OleError always raises an exception.

See Also

EOleSysError (see page 5532)

OleCheck (see page 5600)

28.3.20 ComObj.ProgIDToClassID Function

Returns the class ID (CLSID) GUID that corresponds to the string specified as the ProgID parameter.

Pascal

```
function ProgIDToClassID(const ProgID: string): TGUID;
```

C++

```
TGUID ProgIDToClassID(const AnsiString ProgID);
```

File

ComObj

Description

ProgIDToClassID retrieves, from the registry, the CLSID for a given programmatic ID. The ProgID parameter specifies the programmatic ID for which the CLSID is requested. A programmatic ID is a string that is associated with the CLSID in the registry.

If ProgIDToClassID succeeds it returns the requested CLSID. If ProgIDToClassID fails an EOleSysError (see page 5532) exception is raised with an error code indicating the cause of the error. The error code possibilities are as follows:

Error code	Meaning
CO_E_CLASSSTRING	The registered CLSID for the ProgID is invalid.
REGDB_E_WRITEREGDB	An error occurred writing the CLSID to the registry.

Note: If the ProgID cannot be found in the registry, ProgIDToClassID creates an OLE 1 CLSID for the ProgID and a CLSID entry in the registry.

See Also

EOleSysError (see page 5532)

OleError (see page 5600)

28.3.21 ComObj.RegisterAsService Function

Registers a COM object as an NT service.

Pascal

```
procedure RegisterAsService(const ClassID: string; const ServiceName: string);
```

C++

```
RegisterAsService(const AnsiString ClassID, const AnsiString ServiceName);
```

File

ComObj

Description

Use RegisterAsService to register a service given a classID. ClassID specifies the classID of the COM object that implements the service. ServiceName specifies the name under which the object is registered.

See Also

TService

28.3.22 ComObj.RegisterComServer Function

Registers an in-process COM server with the operating system.

Pascal

```
procedure RegisterComServer(const DLLName: string);
```

C++

```
RegisterComServer(const AnsiString DLLName);
```

File

ComObj

Description

COM servers are always registered as dynamic link libraries, DLLs. The DLLName parameter specifies the name of the DLL that implements the COM controls.

28.3.23 ComObj.SetDispatchPropValue Function

Sets the value of a property on an IDispatch interface.

Pascal

```
procedure SetDispatchPropValue(Disp: IDispatch; DispID: Integer; const Value: OleVariant);
overload;
procedure SetDispatchPropValue(Disp: IDispatch; Name: WideString; const Value: OleVariant);
overload;
```

C++

```
SetDispatchPropValue(IDispatch Disp, int DispID, const OleVariant Value);
SetDispatchPropValue(IDispatch Disp, BSTR Name, const OleVariant Value);
```

File

ComObj

Description

Call SetDispatchPropValue to assign the value specified by the Value parameter to the specified property of the IDispatch interface given by the Disp parameter. The DispID parameter gives the dispatch ID of the property to set.

See Also

EnumDispatchProperties

GetDispatchPropValue (see page 5596)

28.3.24 ComObj.StringToGUID Function

Pascal

```
function StringToGUID(const S: string): TGUID;
```

C++

```
TGUID StringToGUID(const AnsiString S);
```

File

ComObj

Description

This is function ComObj.StringToGUID.

28.3.25 ComObj.StringToLPOLESTR Function

Pascal

```
function StringToLPOLESTR(const Source: string): POleStr;
```

C++

```
POleStr StringToLPOLESTR(const AnsiString Source);
```

File

ComObj

Description

This is function ComObj.StringToLPOLESTR.

28.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TClassInstancing (see page 5604)	This is record ComObj.TClassInstancing.
	TThreadingModel (see page 5604)	This is record ComObj.TThreadingModel.

28.4.1 ComObj.TClassInstancing Enumeration

Pascal

```
TClassInstancing = (
  ciInternal,
  ciSingleInstance,
  ciMultiInstance
);
```

C++

```
enum TClassInstancing {
  ciInternal,
  ciSingleInstance,
  ciMultiInstance
};
```

File

ComObj

Description

This is record ComObj.TClassInstancing.

28.4.2 ComObj.TThreadingModel Enumeration

Pascal

```
TThreadingModel = (
  tmSingle,
  tmApartment,
  tmFree,
  tmBoth,
  tmNeutral
);
```

C++

```
enum TThreadingModel {
  tmSingle,
  tmApartment,
  tmFree,
  tmBoth,
```

```
    tmNeutral
  };
```

File

ComObj

Description

This is record ComObj.TThreadingModel.

28.5 Types

The following table lists types in this documentation.

Types

Name	Description
TAutoClass (see page 5606)	This is type ComObj.TAutoClass.
TCoAddRefServerProcessProc (see page 5606)	This is type ComObj.TCoAddRefServerProcessProc.
TCoCreateInstanceExProc (see page 5607)	This is type ComObj.TCoCreateInstanceExProc.
TCoInitializeExProc (see page 5607)	This is type ComObj.TCoInitializeExProc.
TComClass (see page 5607)	This is type ComObj.TComClass.
TConnectEvent (see page 5608)	This is type ComObj.TConnectEvent.
TCoReleaseServerProcessProc (see page 5608)	This is type ComObj.TCoReleaseServerProcessProc.
TCoResumeClassObjectsProc (see page 5608)	This is type ComObj.TCoResumeClassObjectsProc.
TCoSuspendClassObjectsProc (see page 5609)	This is type ComObj.TCoSuspendClassObjectsProc.
TFactoryProc (see page 5609)	This is type ComObj.TFactoryProc.
TTypedComClass (see page 5609)	This is type ComObj.TTypedComClass.

28.5.1 ComObj.TAutoClass Type

Pascal

```
TAutoClass = class of TAutoObject;
```

C++

```
TAutoObject TAutoClass;
```

File

ComObj

Description

This is type ComObj.TAutoClass.

28.5.2 ComObj.TCoAddRefServerProcessProc Type

Pascal

```
TCoAddRefServerProcessProc = function :Longint;
```

C++

```
:Longint ( TCoAddRefServerProcessProc)();
```

File

ComObj

Description

This is type ComObj.TCoAddRefServerProcessProc.

28.5.3 ComObj.TCoCreateInstanceExProc Type

Pascal

```
TCoCreateInstanceExProc = function (const clsid: TCLSID; unkOuter: IUnknown; dwClsCtx:
Longint; ServerInfo: PCoServerInfo; dwCount: Longint; rgmqResults: PMultiQIArray): HResult
stdcall;
```

C++

```
(const clsid: TCLSID; unkOuter: IUnknown; dwClsCtx: Longint; ServerInfo: PCoServerInfo;
dwCount: Longint; rgmqResults: PMultiQIArray): HResult stdcall ( TCoCreateInstanceExProc)();
```

File

ComObj

Description

This is type ComObj.TCoCreateInstanceExProc.

28.5.4 ComObj.TCoInitializeExProc Type

Pascal

```
TCoInitializeExProc = function (pvReserved: Pointer; coInit: Longint): HResult;
```

C++

```
(pvReserved: Pointer; coInit: Longint): HResult ( TCoInitializeExProc)();
```

File

ComObj

Description

This is type ComObj.TCoInitializeExProc.

28.5.5 ComObj.TComClass Type

Pascal

```
TComClass = class of TComObject;
```

C++

```
TComObject TComClass;
```

File

ComObj

Description

This is type ComObj.TComClass.

28.5.6 ComObj.TConnectEvent Type

Pascal

```
TConnectEvent = procedure (const Sink: IUnknown; Connecting: Boolean) of object;
```

C++

```
(const Sink: IUnknown; Connecting: Boolean) ( TConnectEvent)();
```

File

ComObj

Description

This is type ComObj.TConnectEvent.

28.5.7 ComObj.TCoReleaseServerProcessProc Type

Pascal

```
TCoReleaseServerProcessProc = function :Longint;
```

C++

```
:Longint ( TCoReleaseServerProcessProc)();
```

File

ComObj

Description

This is type ComObj.TCoReleaseServerProcessProc.

28.5.8 ComObj.TCoResumeClassObjectsProc Type

Pascal

```
TCoResumeClassObjectsProc = function :HResult;
```

C++

```
:HResult ( TCoResumeClassObjectsProc)();
```

File

ComObj

Description

This is type ComObj.TCoResumeClassObjectsProc.

28.5.9 ComObj.TCoSuspendClassObjectsProc Type

Pascal

```
TCoSuspendClassObjectsProc = function :HResult;
```

C++

```
:HResult ( TCoSuspendClassObjectsProc)();
```

File

ComObj

Description

This is type ComObj.TCoSuspendClassObjectsProc.

28.5.10 ComObj.TFactoryProc Type

Pascal

```
TFactoryProc = procedure (Factory: TComObjectFactory) of object;
```

C++

```
(Factory: TComObjectFactory) ( TFactoryProc)();
```

File

ComObj

Description

This is type ComObj.TFactoryProc.

28.5.11 ComObj.TTypedComClass Type

Pascal

```
TTypedComClass = class of TTypedComObject;
```

C++

```
TTypedComObject TTypedComClass;
```

File

ComObj

Description

This is type ComObj.TTypedComClass.

28.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
CoAddRefServerProcess (see page 5610)	This is variable ComObj.CoAddRefServerProcess.
CoCreateInstanceEx (see page 5610)	This is variable ComObj.CoCreateInstanceEx.
CoInitFlags (see page 5611)	Specifies the level of threading support requested for a COM server .EXE.
CoInitializeEx (see page 5611)	This is variable ComObj.CoInitializeEx.
CoReleaseServerProcess (see page 5612)	This is variable ComObj.CoReleaseServerProcess.
CoResumeClassObjects (see page 5612)	This is variable ComObj.CoResumeClassObjects.
CoSuspendClassObjects (see page 5612)	This is variable ComObj.CoSuspendClassObjects.

28.6.1 ComObj.CoAddRefServerProcess Variable

Pascal

```
CoAddRefServerProcess: TCoAddRefServerProcessProc = nil;
```

C++

```
TCoAddRefServerProcessProc CoAddRefServerProcess = nil;
```

File

ComObj

Description

This is variable ComObj.CoAddRefServerProcess.

28.6.2 ComObj.CoCreateInstanceEx Variable

Pascal

```
CoCreateInstanceEx: TCoCreateInstanceExProc = nil;
```

C++

```
TCoCreateInstanceExProc CoCreateInstanceEx = nil;
```

File

ComObj

Description

This is variable ComObj.CoCreateInstanceEx.

28.6.3 ComObj.CoInitFlags Variable

Specifies the level of threading support requested for a COM server .EXE.

Pascal

```
CoInitFlags: Integer = -1;
```

C++

```
int CoInitFlags = -1;
```

File

ComObj

Description

CoInitFlags controls the way COM is initialized for threading support. This variable only affects executables (.EXEs), not in-process servers (.DLLs and .OCXs).

Typically, code to initialize the level of threading support is added automatically by the wizard that creates the COM object in your application. In applications containing multiple COM objects, COM is initialized for the highest level of support, where single threading is lowest and free threading is highest. To manually set the level of COM threading support, assign a value to CoInitFlags in the program's main source file before the call to Application.Initialize.

The following table lists the constants (defined in the ActiveX unit) that can be or'ed together to create a value for CoInitFlags.

ConstantValueMeaning

Constant	Value	Meaning
COINIT_MULTITHREADED	0	COM calls objects on any thread (free threading)
COINIT_APARTMENTTHREADED	2	Each object has its own thread (apartment model)
COINIT_DISABLE_OLE1DDE	4	Don't use DDE for Ole1 support.
COINIT_SPEED_OVER_MEMORY	8	Optimize for speed rather than memory.

The value of CoInitFlags is –1 for single threading.

See Also

TApplication

IsMultiThread

28.6.4 ComObj.CoInitializeEx Variable

Pascal

```
CoInitializeEx: TCoInitializeExProc = nil;
```

C++

```
TCoInitializeExProc CoInitializeEx = nil;
```

File

ComObj

Description

This is variable ComObj.CoInitializeEx.

28.6.5 ComObj.CoReleaseServerProcess Variable

Pascal

```
CoReleaseServerProcess: TCoReleaseServerProcessProc = nil;
```

C++

```
TCoReleaseServerProcessProc CoReleaseServerProcess = nil;
```

File

ComObj

Description

This is variable ComObj.CoReleaseServerProcess.

28.6.6 ComObj.CoResumeClassObjects Variable

Pascal

```
CoResumeClassObjects: TCoResumeClassObjectsProc = nil;
```

C++

```
TCoResumeClassObjectsProc CoResumeClassObjects = nil;
```

File

ComObj

Description

This is variable ComObj.CoResumeClassObjects.

28.6.7 ComObj.CoSuspendClassObjects Variable

Pascal

```
CoSuspendClassObjects: TCoSuspendClassObjectsProc = nil;
```

C++

```
TCoSuspendClassObjectsProc CoSuspendClassObjects = nil;
```

File

ComObj

Description

This is variable ComObj.CoSuspendClassObjects.

29 CompProd Namespace

This is namespace CompProd.

Topics

Name	Description
Classes (see page 5614)	The following table lists classes in this documentation.

Classes

	Name	Description
	TComponentsPageProducer (see page 5614)	TComponentsPageProducer produces a string of HTML commands based on an input template.

29.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TComponentsPageProducer (see page 5614)	TComponentsPageProducer produces a string of HTML commands based on an input template.

29.1.1 TComponentsPageProducer Class

TComponentsPageProducer produces a string of HTML commands based on an input template.

Class Hierarchy

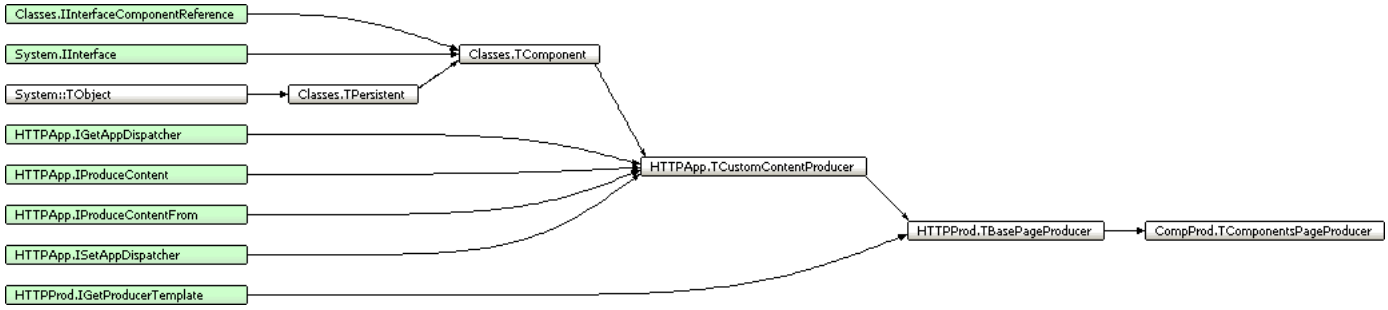

Pascal

```
TComponentsPageProducer = class(TBasePageProducer);
```

C++

```
class TComponentsPageProducer : public TBasePageProducer;
```

File

CompProd

Description

Use TComponentsPageProducer to convert an HTML template that includes content generated by other components into a string of HTML commands that can be interpreted by a client application such as a Web browser. HTML templates include HTML commands and HTML-transparent tags that are replaced with customized content by the components page producer in an OnHTMLTag (see page 5620) event handler.

Components page producers differ from other page producer components in that they replace special HTML-transparent tags of the format

```
<#COMPONENT Name=WebContentComponent>
```

with the HTML produced by the component named WebContentComponent. WebContentComponent must have the same Owner as the components page producer and support the IWebContent interface, which the components page producer uses to obtain the replacement HTML for the COMPONENT tag.

See Also

IWebContent (see page 24792)

TPageItemsProducer (see page 18383)

TInetXPageProducer (see page 17145)

29.1.1.1 TComponentsPageProducer Members

The following tables list the members exposed by TComponentsPageProducer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoTagEvent (see page 5618)	This is DoTagEvent, a member of class TComponentsPageProducer.
	FindComponent (see page 5619)	This is FindComponent, a member of class TComponentsPageProducer.
	FindComponentName (see page 5619)	This is FindComponentName, a member of class TComponentsPageProducer.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetComponentContent (see page 5619)	This is GetComponentContent, a member of class TComponentsPageProducer.
	GetContentOptions (see page 5619)	This is GetContentOptions, a member of class TComponentsPageProducer.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetProducerTemplateFileName (see page 15276)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.
	GetProducerTemplateStream (see page 15276)	This is GetProducerTemplateStream, a member of class TBasePageProducer.
	GetScriptEngine (see page 15277)	This is GetScriptEngine, a member of class TBasePageProducer.
	GetTagID (see page 15277)	This is GetTagID, a member of class TBasePageProducer.
	GetTemplateFileName (see page 15277)	This is GetTemplateFileName, a member of class TBasePageProducer.
	GetTemplateStream (see page 15277)	This is GetTemplateStream, a member of class TBasePageProducer.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleScriptTag (see page 15277)	This is HandleScriptTag, a member of class TBasePageProducer.
	HandleTag (see page 5620)	This is HandleTag, a member of class TComponentsPageProducer.
	ImplHandleTag (see page 15278)	This is ImplHandleTag, a member of class TBasePageProducer.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.

	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 15108)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ServerScriptFromStream (see page 15278)	This is ServerScriptFromStream, a member of class TBasePageProducer.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UseScriptEngine (see page 15278)	This is UseScriptEngine, a member of class TBasePageProducer.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 15106)	Returns the content produced by the content producer.
	Content (see page 15274)	Returns the content produced by the content producer.
	ContentFromStream (see page 15107)	Returns the content produced in response to input read from a stream.
	ContentFromStream (see page 15275)	Returns the content produced in response to input read from a stream.
	ContentFromString (see page 15107)	Returns the content produced in response to an input string.
	ContentFromString (see page 15275)	Returns the content produced in response to an input string.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.

	Create (see page 15274)	Creates an instance of TBasePageProducer (see page 15269).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProducerTemplateFileName (see page 15320)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.
	GetProducerTemplateStream (see page 15320)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ScriptEngine (see page 15279)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 15279)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebModuleContext (see page 15279)	Provides access to the Web module context object.

Published Events

	Name	Description
	OnHTMLTag (see page 5620)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

29.1.1.2 TComponentsPageProducer Methods

The methods of the TComponentsPageProducer class are listed here.

Protected Methods

	Name	Description
V	DoTagEvent (see page 5618)	This is DoTagEvent, a member of class TComponentsPageProducer.
V	FindComponent (see page 5619)	This is FindComponent, a member of class TComponentsPageProducer.
	FindComponentName (see page 5619)	This is FindComponentName, a member of class TComponentsPageProducer.
V	GetComponentContent (see page 5619)	This is GetComponentContent, a member of class TComponentsPageProducer.
V	GetContentOptions (see page 5619)	This is GetContentOptions, a member of class TComponentsPageProducer.
V	HandleTag (see page 5620)	This is HandleTag, a member of class TComponentsPageProducer.

29.1.1.2.1 TComponentsPageProducer.DoTagEvent Method

Pascal

```
procedure DoTagEvent(Tag: TTag; const TagString: string; TagParams: TStrings; var ReplaceText:
string); override;
```

C++

```
virtual __fastcall DoTagEvent(TTag Tag, const AnsiString TagString, TStrings TagParams,
AnsiString ReplaceText);
```

Description

This is DoTagEvent, a member of class TComponentsPageProducer.

29.1.1.2.2 TComponentsPageProducer.FindComponent Method

Pascal

```
function FindComponent(TagParams: TStrings): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * FindComponent(TStrings TagParams);
```

Description

This is FindComponent, a member of class TComponentsPageProducer.

29.1.1.2.3 TComponentsPageProducer.FindComponentName Method

Pascal

```
function FindComponentName(TagParams: TStrings): string;
```

C++

```
__fastcall AnsiString FindComponentName(TStrings TagParams);
```

Description

This is FindComponentName, a member of class TComponentsPageProducer.

29.1.1.2.4 TComponentsPageProducer.GetComponentContent Method

Pascal

```
function GetComponentContent(TagParams: TStrings): string; virtual;
```

C++

```
virtual __fastcall AnsiString GetComponentContent(TStrings TagParams);
```

Description

This is GetComponentContent, a member of class TComponentsPageProducer.

29.1.1.2.5 TComponentsPageProducer.GetContentOptions Method

Pascal

```
function GetContentOptions(var Owned: Boolean): TWebContentOptions; virtual;
```

C++

```
virtual __fastcall TWebContentOptions GetContentOptions(Boolean Owned);
```

Description

This is GetContentOptions, a member of class TComponentsPageProducer.

29.1.1.2.6 TComponentsPageProducer.HandleTag Method

Pascal

```
function HandleTag(const TagString: string; TagParams: TStrings): string; override;
```

C++

```
virtual __fastcall AnsiString HandleTag(const AnsiString TagString, TStrings TagParams);
```

Description

This is HandleTag, a member of class TComponentsPageProducer.

29.1.1.3 TComponentsPageProducer Events

The events of the TComponentsPageProducer class are listed here.

Published Events

	Name	Description
	OnHTMLTag (see page 5620)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

29.1.1.3.1 TComponentsPageProducer.OnHTMLTag Event

Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Pascal

```
property OnHTMLTag: THTMLTagEvent;
```

C++

```
__property THTMLTagEvent OnHTMLTag;
```

Description

Write an OnHTMLTag event handler to convert an HTML-transparent tag into the corresponding HTML commands that can be interpreted by a Web browser. If an OnHTMLTag event is not supplied, the HTML-transparent tags in the HTML template will be replaced by empty strings.

Note: The sequence of HTML that the event handler produces for an HTML-transparent tag can contain HTML-transparent tags for another page producer to convert. For example, one page producer might have a template that represents the format of the final HTTP response. It could read the HTTP request and for each section in the request assemble a set of HTML-transparent tags with parameters based on the request. Another page producer might take the content of the first page producer, and interpret those parameters.

See Also

Names (see page 3075)

Values (see page 3086)

Customizing the InternetExpress Page Producer Template

Chaining Page Producers Together

Converting HTML-transparent Tags

30 ComServ Namespace

This is namespace ComServ.

Topics

Name	Description
Classes (see page 5623)	The following table lists classes in this documentation.
Functions (see page 5634)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 5638)	The following table lists structs, records, enums in this documentation.
Types (see page 5639)	The following table lists types in this documentation.
Variables (see page 5640)	The following table lists variables in this documentation.

Classes

	Name	Description
	TComServer (see page 5623)	TComServer is the class type of the ComServ variable that is declared and instantiated in the ComServ unit.

Functions

	Name	Description
	DllCanUnloadNow (see page 5634)	Tells Windows whether it can unload the in-process server exporting this function.
	DllGetClassObject (see page 5635)	Obtains a class factory for an ActiveX object when the ActiveX object resides in an in-process ActiveX server (DLL).
	DllRegisterServer (see page 5636)	Registers an in-process ActiveX server in the current module within the system registry.
	DllUnregisterServer (see page 5636)	Unregisters an OLE automation server in the current module with the system registry.
	LoadTypeLibrary (see page 5637)	This is function ComServ.LoadTypeLibrary.

Structs, Records, Enums

	Name	Description
	TStartMode (see page 5638)	This is record ComServ.TStartMode.

Types

Name	Description
TLastReleaseEvent (see page 5639)	This is type ComServ.TLastReleaseEvent.

Variables

Name	Description
ComServer (see page 5640)	Provides class and registry information about server objects.

30.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TComServer (see page 5623)	TComServer is the class type of the ComServ (see page 5621) variable that is declared and instantiated in the ComServ (see page 5621) unit.

30.1.1 TComServer Class

TComServer is the class type of the ComServ (see page 5621) variable that is declared and instantiated in the ComServ (see page 5621) unit.

Class Hierarchy

System::TObject → ComObj.TComServerObject → ComServ.TComServer

Pascal

```
TComServer = class(TComServerObject);
```

C++

```
class TComServer : public TComServerObject;
```

File

ComServ

Description

TComServer is the class type used for a COM server, which is a module that contains COM or OLE class definitions. That module can be either an in-process server (DLL) or a local server (EXE).

TComServer properties contain all necessary information about the server, such as its type library, name, Help file, and so on. They also indicate when and how the server should be loaded and unloaded.

TComServer methods are used

To access TComServer properties.

To create associated class factories to instantiate the COM and OLE classes based upon their class identifier (CLSID).

To update the registry database with information about the COM and OLE classes.

TComServer is the type of the ComServ (see page 5621) variable that is automatically instantiated in the ComServ (see page 5621) unit. Each module that is created as a server for COM objects has its own instance of ComServ (see page 5621).

To manage its class factories, the ComServer (see page 5640) uses a TComClassManager object, which is accessed using the ComClassManager function in the ComObj (see page 5525) unit. The class factories are used primarily by calls to TComClassManager methods.

Note: The ComServ (see page 5621) unit declares and implements global routines that must be exported by each in-process server. These routines automatically handle the registering, unregistering, and unloading of the COM server.

See Also

ComClassManager

ComServer

DllCanUnLoadNow

DllGetClassObject

DllRegisterServer

DllUnregisterServer

TComClassManager (see page 5550)

30.1.1.1 TComServer Members

The following tables list the members exposed by TComServer.

Protected Methods

	Name	Description
	CountFactory (see page 5626)	This is CountFactory, a member of class TComServer.
	CountObject (see page 5627)	This is CountObject, a member of class TComServer.
	GetHelpFileName (see page 5627)	This is GetHelpFileName, a member of class TComServer.
	GetServerFileName (see page 5627)	This is GetServerFileName, a member of class TComServer.
	GetServerKey (see page 5627)	This is GetServerKey, a member of class TComServer.
	GetServerName (see page 5628)	This is GetServerName, a member of class TComServer.
	GetStartSuspended (see page 5628)	This is GetStartSuspended, a member of class TComServer.
	GetTypeLib (see page 5628)	This is GetTypeLib, a member of class TComServer.
	SetHelpFileName (see page 5629)	This is SetHelpFileName, a member of class TComServer.

Public Events

	Name	Description
	OnLastRelease (see page 5632)	Occurs when the last object using the server is released.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5625)	Creates a COM server object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 5626)	Destroys an instance of a COM server object.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	Initialize (see page 5628)	Updates the registry and registers the class objects.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LoadTypeLib (see page 5629)	Loads the type library for the server.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetServerName (see page 5629)	Sets the ServerName property.
	UpdateRegistry (see page 5630)	Registers information on classes implemented in the COM server.

Public Properties

	Name	Description
	HelpFileName (see page 5579)	Specifies the name of the Help file associated with the module.
	IsInprocServer (see page 5630)	Indicates whether the COM server is an in-process server.
R	ObjectCount (see page 5631)	Indicates how many objects are currently dependent upon the server being loaded.
R	ServerFileName (see page 5580)	Specifies the complete path and filename of the COM server.
R	ServerKey (see page 5580)	Indicates whether the COM server is an in-process server or whether it is a local server.
R	ServerName (see page 5580)	Specifies the name of the executable file for the server.
R	StartMode (see page 5631)	Indicates why the COM server was started.
R	StartSuspended (see page 5580)	Specifies whether the server started in a suspended state.
R	TypeLib (see page 5581)	Provides access to the type library associated with the COM server.
	UIInteractive (see page 5632)	Controls (see page 5711) whether the server displays error messages to the user.

30.1.1.2 TComServer.Create Constructor

Creates a COM server object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TComServer();
```

Description

Applications do not need to call the constructor for TComServer (see page 5623). An instance of TComServer (see page 5623) is automatically created as the ComServ (see page 5621) variable in modules that are servers for COM objects.

Create allocates memory for a COM server object, and initializes its IsInprocServer (see page 5630) and StartMode (see page 5631) properties. It is not necessary to directly call the constructor to create a COM server object. A global ComServer (see page 5640) variable is declared in the ComServ (see page 5621) unit and is instantiated in the initialization section.

See Also

ComServer

IsInprocServer (see page 5630)

StartMode (see page 5631)

30.1.1.3 TComServer.Destroy Destructor

Destroys an instance of a COM server object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TComServer();
```

Description

It is not necessary to directly call Free (or Destroy) to destroy a COM server object. The global ComServ (see page 5621) variable, declared in the ComServ (see page 5621) unit, is automatically created and destroyed in the initialization and finalization sections.

Destroy destroys the COM server object and calls the ComClassManager to free, from the list it maintains, each of the class factories that the server owns.

See Also

Free (see page 22300)

30.1.1.4 TComServer Methods

The methods of the TComServer class are listed here.

Protected Methods

	Name	Description
	CountFactory (see page 5626)	This is CountFactory, a member of class TComServer.
	CountObject (see page 5627)	This is CountObject, a member of class TComServer.
	GetHelpFileName (see page 5627)	This is GetHelpFileName, a member of class TComServer.
	GetServerFileName (see page 5627)	This is GetServerFileName, a member of class TComServer.
	GetServerKey (see page 5627)	This is GetServerKey, a member of class TComServer.
	GetServerName (see page 5628)	This is GetServerName, a member of class TComServer.
	GetStartSuspended (see page 5628)	This is GetStartSuspended, a member of class TComServer.
	GetTypeLib (see page 5628)	This is GetTypeLib, a member of class TComServer.
	SetHelpFileName (see page 5629)	This is SetHelpFileName, a member of class TComServer.

Public Methods

	Name	Description
	Initialize (see page 5628)	Updates the registry and registers the class objects.
	LoadTypeLib (see page 5629)	Loads the type library for the server.
	SetServerName (see page 5629)	Sets the ServerName property.
	UpdateRegistry (see page 5630)	Registers information on classes implemented in the COM server.

30.1.1.4.1 TComServer.CountFactory Method

Pascal

```
function CountFactory(Created: Boolean): Integer; override;
```

C++

```
virtual __fastcall int CountFactory(Boolean Created);
```

Description

This is CountFactory, a member of class TComServer.

30.1.1.4.2 TComServer.CountObject Method

Pascal

```
function CountObject(Created: Boolean): Integer; override;
```

C++

```
virtual __fastcall int CountObject(Boolean Created);
```

Description

This is CountObject, a member of class TComServer.

30.1.1.4.3 TComServer.GetHelpFileName Method

Pascal

```
function GetHelpFileName: string; override;
```

C++

```
virtual __fastcall AnsiString GetHelpFileName();
```

Description

This is GetHelpFileName, a member of class TComServer.

30.1.1.4.4 TComServer.GetServerFileName Method

Pascal

```
function GetServerFileName: string; override;
```

C++

```
virtual __fastcall AnsiString GetServerFileName();
```

Description

This is GetServerFileName, a member of class TComServer.

30.1.1.4.5 TComServer.GetServerKey Method

Pascal

```
function GetServerKey: string; override;
```

C++

```
virtual __fastcall AnsiString GetServerKey();
```

Description

This is GetServerKey, a member of class TComServer.

30.1.1.4.6 TComServer.GetServerName Method

Pascal

```
function GetServerName: string; override;
```

C++

```
virtual __fastcall AnsiString GetServerName();
```

Description

This is GetServerName, a member of class TComServer.

30.1.1.4.7 TComServer.GetStartSuspended Method

Pascal

```
function GetStartSuspended: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetStartSuspended();
```

Description

This is GetStartSuspended, a member of class TComServer.

30.1.1.4.8 TComServer.GetTypeLib Method

Pascal

```
function GetTypeLib: ITypeLib; override;
```

C++

```
virtual __fastcall ITypeLib GetTypeLib();
```

Description

This is GetTypeLib, a member of class TComServer.

30.1.1.4.9 TComServer.Initialize Method

Updates the registry and registers the class objects.

Pascal

```
procedure Initialize;
```

C++

```
__fastcall Initialize();
```

Description

Initialize updates the registry if the server was not started only to remove the server from the system registry. Then, if the server was started as an interactive stand-alone application, or in response to a request from an automation controller, Initialize calls on each class factory, owned by the server, to register its class object in the system registry.

See Also

RegisterClassObject (see page 5569)

UpdateRegistry (see page 5630)

30.1.1.4.10 TComServer.LoadTypeLib Method

Loads the type library for the server.

Pascal

```
procedure LoadTypeLib;
```

C++

```
__fastcall LoadTypeLib();
```

Description

LoadTypeLib loads the type library for the server. If the type library is not already loaded when LoadTypeLib is called, it also registers the type library.

If LoadTypeLib is not successful, it raises an EOleSysError indicating the error code. The possible error codes are as follows:

Error code	Meaning
E_OUTOFMEMORY	Out of memory.
E_INVALIDARG	One or more of the arguments is invalid.
TYPE_E_IOERROR	The function could not write to the file.
TYPE_E_INVALIDSTATE	The type library could not be opened.
TYPE_E_INVDATAREAD	The function could not read from the file.
TYPE_E_UNSUPFORMAT	The type library has an older format.
TYPE_E_UNKNOWNLCID	The LCID could not be found in the OLE support DLLs.
TYPE_E_CANTLOADLIBRARY	The type library or DLL could not be loaded.
Other return codes	All FACILITY_STORAGE errors may be returned.

See Also

EOleSysError (see page 5532)

TypeLib

30.1.1.4.11 TComServer.SetHelpFileName Method

Pascal

```
procedure SetHelpFileName(const Value: string); override;
```

C++

```
virtual __fastcall SetHelpFileName(const AnsiString Value);
```

Description

This is SetHelpFileName, a member of class TComServer.

30.1.1.4.12 TComServer.SetServerName Method

Sets the ServerName property.

Pascal

```
procedure SetServerName(const Name: string);
```

C++

```
__fastcall SetServerName(const AnsiString Name);
```

Description

Call SetServerName to provide a name for the server. If the server has an associated type library (the TypeLib property is nil for Delphi or NULL for C++), SetServerName does nothing, because the name for the server must match the name specified in the type library.

See Also

ServerName

TypeLib

30.1.1.4.13 TComServer.UpdateRegistry Method

Registers information on classes implemented in the COM server.

Pascal

```
procedure UpdateRegistry(Register: Boolean);
```

C++

```
__fastcall UpdateRegistry(Boolean Register);
```

Description

UpdateRegistry updates the registry to reflect the current status of the COM server. Accordingly, UpdateRegistry either registers or unregisters the type library information and the CLSIDs associated with the class factories instantiated by the COM server.

The global functions that register and unregister the server call UpdateRegistry. UpdateRegistry is also called by the Initialize (see page 5628) method.

See Also

DLLRegisterServer

DLLUnregisterServer

Initialize (see page 5628)

30.1.1.5 TComServer Properties

The properties of the TComServer class are listed here.

Public Properties

	Name	Description
	IsInprocServer (see page 5630)	Indicates whether the COM server is an in-process server.
R	ObjectCount (see page 5631)	Indicates how many objects are currently dependent upon the server being loaded.
R	StartMode (see page 5631)	Indicates why the COM server was started.
	UIInteractive (see page 5632)	Controls (see page 5711) whether the server displays error messages to the user.

30.1.1.5.1 TComServer.IsInprocServer Property

Indicates whether the COM server is an in-process server.

Pascal

```
property IsInprocServer: Boolean;
```

C++

```
__property Boolean IsInprocServer;
```

Description

InProcServer indicates whether the COM server is an in-process server, meaning that it is running the same process space as the object's container. The value of this property is set in the COM server's constructor and is determined by the IsLibrary variable.

The library assumes that all OLE automation servers in DLLs are in-process servers, and that those in applications are not.

See Also

IsLibrary (see page 22648)

30.1.1.5.2 TComServer.ObjectCount Property

Indicates how many objects are currently dependent upon the server being loaded.

Pascal

```
property ObjectCount: Integer;
```

C++

```
__property int ObjectCount;
```

Description

ObjectCount indicates when the server can be unloaded. ObjectCount contains the number of objects that are currently dependent upon the server.

ObjectCount is incremented when new objects are created from class factories, and is decremented when objects are destroyed.

When ObjectCount is zero the server can be unloaded. At this time, code written for the OnLastRelease (see page 5632) event handler can process special actions after the last object is released and before the server shuts down.

See Also

OnLastRelease (see page 5632)

30.1.1.5.3 TComServer.StartMode Property

Indicates why the COM server was started.

Pascal

```
property StartMode: TStartMode;
```

C++

```
__property TStartMode StartMode;
```

Description

StartMode indicates the reason a COM server was started. It can be started as a stand-alone application, as an interactive application, as an embedded object, or for registration or unregistration with the system Registry (see page 18633).

TStartMode (see page 5638) is the type of the StartMode property, indicating the different reasons why an OLE automation server can be started.

The following table explains each of the possible start-mode values and the command-line switch that corresponds to each.

Start mode	Switch	Meaning
smAutomation	embedding	The application was started by Windows in response to a request from an automation controller.
smRegServer	regserver	The application was started only to add the server to the system registry.
smStandalone	-	The user started the application as a stand-alone, interactive application.
smUnregServer	unregserver	The application was started only to remove the server from the system registry.

See Also

OnLastRelease (see page 5632)

30.1.1.5.4 TComServer.UIInteractive Property

Controls (see page 5711) whether the server displays error messages to the user.

Pascal

```
property UIInteractive: Boolean;
```

C++

```
__property Boolean UIInteractive;
```

Description

Use UIInteractive to determine whether the server has a user interface. Set UIInteractive to False to suppress error messages that appear when there are server errors. For example, UIInteractive should be False for COM servers running as a service

30.1.1.6 TComServer Events

The events of the TComServer class are listed here.

Public Events

	Name	Description
	OnLastRelease (see page 5632)	Occurs when the last object using the server is released.

30.1.1.6.1 TComServer.OnLastRelease Event

Occurs when the last object using the server is released.

Pascal

```
property OnLastRelease: TLastReleaseEvent;
```

C++

```
__property TLastReleaseEvent OnLastRelease;
```

Description

Use OnLastRelease to write a handler to process some action after the last object using the server is released, and just before the server shuts down. ShutDown indicates whether the server should be unloaded.

TLastReleaseEvent (see page 5639) is the type of the OnLastRelease event. Shutdown indicates whether the server will shut down after processing the OnLastRelease event.

See Also

StartMode (see page 5631)

30.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DllCanUnloadNow (see page 5634)	Tells Windows whether it can unload the in-process server exporting this function.
	DllGetClassObject (see page 5635)	Obtains a class factory for an ActiveX object when the ActiveX object resides in an in-process ActiveX server (DLL).
	DllRegisterServer (see page 5636)	Registers an in-process ActiveX server in the current module within the system registry.
	DllUnregisterServer (see page 5636)	Unregisters an OLE automation server in the current module with the system registry.
	LoadTypeLibrary (see page 5637)	This is function ComServ.LoadTypeLibrary.

30.2.1 ComServ.DllCanUnloadNow Function

Tells Windows whether it can unload the in-process server exporting this function.

Pascal

```
function DllCanUnloadNow: HResult; stdcall;
```

C++

```
__stdcall HRESULT DllCanUnloadNow();
```

File

ComServ

Description

Do not call DllCanUnloadNow directly. DllCanUnloadNow is exported by ActiveX servers (DLLs) and is called by the operating system to indicate whether the server can be unloaded from memory because it is no longer in use. A DLL is no longer in use when it is not managing any existing objects (the reference count on all of its objects is 0).

The return values for DllCanUnloadNow are as follows:

Value	Meaning
S_OK	The DLL can be unloaded.
S_FALSE	The DLL cannot be unloaded now.

Note: DllCanUnloadNow is one of four functions in the ComServ (see page 5621) unit that every in-process ActiveX server must export so that Windows can manage the server resources properly. The others are DllGetClassObject (see page 5635), DllRegisterServer (see page 5636), and DllUnregisterServer (see page 5636).

See Also

DllGetClassObject

DllRegisterServer

DllUnregisterServer

30.2.2 ComServ.DllGetClassObject Function

Obtains a class factory for an ActiveX object when the ActiveX object resides in an in-process ActiveX server (DLL).

Pascal

```
function DllGetClassObject(const CLSID: TGUID; const IID: TGUID; var Obj): HResult; stdcall;
```

C++

```
__stdcall HRESULT DllGetClassObject(const TGUID CLSID, const TGUID IID,  Obj);
```

File

ComServ

Description

Do not call DllGetClassObject directly. DllGetClassObject is exported by all ActiveX server DLLs, and it is called internally by the operating system. When a user calls CoGetClassObject (or CoCreateInstance, which internally calls CoGetClassObject) to obtain a class factory for an ActiveX object, the OLE engine loads the ActiveX server DLL into memory and calls its DllGetClassObject function.

The parameters to DllGetClassObject are:

CLSID is the class identifier for the ActiveX object to be created by the class factory.

IID is the interface indentifier for the interface through which the caller will communicate with the class factory object to create instances of the object referenced by CLSID. This parameter is nearly always IID_IClassFactory or IID_IClassFactory2.

Obj will hold a pointer to the resulting class factory interface upon successful completion of the function. If an error occurs, Obj will be set to nil.

The return values for DLLGetClassObject are as follows:

Value	Meaning
S_OK	The object was retrieved successfully.
E_NOINTERFACE	The object referenced by Obj doesn't support the interface passed in the IID parameter.
CLASS_E_CLASSNOTAVAILABLE	The DLL does not support the class type referenced by the CLSID parameter.

Note: DllGetClassObject is one of four functions in the ComServ (see page 5621) unit that every in-process server must export so that Windows can manage the server resources properly. The others are DllCanUnloadNow (see page 5634), DllRegisterServer (see page 5636), and DllUnregisterServer (see page 5636).

See Also

DllCanUnloadNow

DllRegisterServer

DllUnregisterServer

TComClassManager

Requirements for Transactional Objects

30.2.3 ComServ.DllRegisterServer Function

Registers an in-process ActiveX server in the current module within the system registry.

Pascal

```
function DllRegisterServer: HResult; stdcall;
```

C++

```
__stdcall HRESULT DllRegisterServer();
```

File

ComServ

Description

Do not call DllRegisterServer directly. DllRegisterServer is exported by in-process ActiveX servers (DLLs) and instructs the DLL to create its registry entries for the type library and all classes supported in that server module. DLLRegisterServer is usually called when the ActiveX server is installed on a user's system.

If this function fails, the state of the registry for its type library and its classes is indeterminate.

The return values for DllRegisterServer are as follows:

Value	Meaning
S_OK	The registry entries were created successfully.
E_FAIL	The registry entries were not completed.

Note: DllRegisterServer is one of four functions in the ComServ (see page 5621) unit that every in-process server must export so that Windows can manage the server resources properly. The others are DllCanUnloadNow (see page 5634), DllGetClassObject (see page 5635), and DllUnregisterServer (see page 5636).

See Also

DllCanUnloadNow

DllGetClassObject

DllUnregisterServer

Requirements for Transactional Objects

30.2.4 ComServ.DllUnregisterServer Function

Unregisters an OLE automation server in the current module with the system registry.

Pascal

```
function DllUnregisterServer: HResult; stdcall;
```

C++

```
__stdcall HRESULT DllUnregisterServer();
```

File

ComServ

Description

Do not call DllUnregisterServer directly. DllUnregisterServer is exported by in-process automation servers (DLLs) and instructs the DLL to remove entries created through DllRegisterServer (see page 5636). DLLRegisterServer is called when a server is uninstalled.

The return values for DllUnregisterServer are as follows:

Value	Meaning
S_OK	The registry entries were successfully removed.
S_FALSE	Some or all registry entries were not removed.

Note: DllUnregisterServer is one of four functions in the ComServ (see page 5621) unit that every in-process server must export so that Windows can manage the server resources properly. The others are DllCanUnloadNow (see page 5634), DllGetClassObject (see page 5635), and DllRegisterServer (see page 5636). It should never be necessary for users to directly call or alter any of these functions.

See Also

DllCanUnloadNow

DllGetClassObject

DllRegisterServer

30.2.5 ComServ.LoadTypeLibrary Function

Pascal

```
function LoadTypeLibrary(const ModuleName: string): ITypeLib;
```

C++

```
ITypeLib LoadTypeLibrary(const AnsiString ModuleName);
```

File

ComServ

Description

This is function ComServ.LoadTypeLibrary.

30.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TStartMode (see page 5638)	This is record ComServ.TStartMode.

30.3.1 ComServ.TStartMode Enumeration

Pascal

```
TStartMode = (
  smStandalone,
  smAutomation,
  smRegServer,
  smUnregServer
);
```

C++

```
enum TStartMode {
  smStandalone,
  smAutomation,
  smRegServer,
  smUnregServer
};
```

File

ComServ

Description

This is record ComServ.TStartMode.

30.4 Types

The following table lists types in this documentation.

Types

Name	Description
TLastReleaseEvent (see page 5639)	This is type ComServ.TLastReleaseEvent.

30.4.1 ComServ.TLastReleaseEvent Type

Pascal

```
TLastReleaseEvent = procedure (var Shutdown: Boolean) of object;
```

C++

```
(var Shutdown: Boolean) ( TLastReleaseEvent)();
```

File

ComServ

Description

This is type ComServ.TLastReleaseEvent.

30.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
ComServer (see page 5640)	Provides class and registry information about server objects.

30.5.1 ComServ.ComServer Variable

Provides class and registry information about server objects.

Pascal

```
ComServer: TComServer;
```

C++

```
TComServer ComServer;
```

File

ComServ

Description

ComServ (see page 5621) declares an instance of COM server. By default, when the ComServ (see page 5621) unit is included in a project, Delphi constructs a COM server object and assigns it to ComServ (see page 5621). ComServ (see page 5621) has several properties and methods that provide class and registry information about the objects loaded on the server. See TComServer (see page 5623) object for a list of properties and methods.

31 Contnrs Namespace

This is namespace Contnrs.

Topics

Name	Description
Classes (see page 5643)	The following table lists classes in this documentation.
Functions (see page 5705)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 5706)	The following table lists structs, records, enums in this documentation.
Types (see page 5709)	The following table lists types in this documentation.

Classes

	Name	Description
	TBucketList (see page 5643)	TBucketList is a simple hash table that stores pointers indexed by pointers.
	TClassList (see page 5646)	TClassList maintains a list of classes.
	TComponentList (see page 5653)	TComponentList maintains a list of components, updating itself when a component is freed.
	TCustomBucketList (see page 5661)	TCustomBucketList is the base class for lists that store items under a limited number of categories.
	TIntegerBucketList (see page 5670)	This is class Contnrs.TIntegerBucketList.
	TObjectBucketList (see page 5673)	TObjectBucketList is a simple hash table that stores objects that are identified by other objects.
	TObjectList (see page 5677)	TObjectList maintains a list of (owned) objects.
	TObjectQueue (see page 5687)	TObjectQueue maintains a first-in first-out list of objects.
	TObjectStack (see page 5690)	TObjectStack maintains a last-in first-out list of objects.
	TOrderedList (see page 5693)	TOrderedList maintains an array of pointers that must be accessed in serial order.
	TQueue (see page 5699)	TQueue maintains a first-in first-out array of pointers.
	TStack (see page 5701)	TStack maintains a last-in first-out array of pointers.

Functions

	Name	Description
	RaiseListError (see page 5705)	This is function Contnrs.RaiseListError.

Structs, Records, Enums

	Name	Description
	TBucket (see page 5706)	TBucketArray (see page 5709) and TBucket represent a list of buckets that each store an array of items.

	TBucketItem (see page 5707)	TBucketItemArray (see page 5710) and TBucketItem represent the items in a TBucket (see page 5706) record (Delphi) or structure (C++).
	TBucketListSizes (see page 5707)	TBucketListSizes indicates the number of distinct buckets in a TBucketList (see page 5643) object.

Types

Name	Description
TBucketArray (see page 5709)	This is type Contnrs.TBucketArray.
TBucketEvent (see page 5709)	This is type Contnrs.TBucketEvent.
TBucketItemArray (see page 5710)	This is type Contnrs.TBucketItemArray.
TBucketProc (see page 5710)	This is type Contnrs.TBucketProc.

31.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TBucketList (see page 5643)	TBucketList is a simple hash table that stores pointers indexed by pointers.
	TClassList (see page 5646)	TClassList maintains a list of classes.
	TComponentList (see page 5653)	TComponentList maintains a list of components, updating itself when a component is freed.
	TCustomBucketList (see page 5661)	TCustomBucketList is the base class for lists that store items under a limited number of categories.
	TIntegerBucketList (see page 5670)	This is class Contnrs.TIntegerBucketList.
	TObjectBucketList (see page 5673)	TObjectBucketList is a simple hash table that stores objects that are identified by other objects.
	TObjectList (see page 5677)	TObjectList maintains a list of (owned) objects.
	TObjectQueue (see page 5687)	TObjectQueue maintains a first-in first-out list of objects.
	TObjectStack (see page 5690)	TObjectStack maintains a last-in first-out list of objects.
	TOrderedList (see page 5693)	TOrderedList maintains an array of pointers that must be accessed in serial order.
	TQueue (see page 5699)	TQueue maintains a first-in first-out array of pointers.
	TStack (see page 5701)	TStack maintains a last-in first-out array of pointers.

31.1.1 TBucketList Class

TBucketList is a simple hash table that stores pointers indexed by pointers.

Class Hierarchy

Pascal

```
TBucketList = class(TCustomBucketList);
```

C++

```
class TBucketList : public TCustomBucketList;
```

File

Contnrs

Description

Use TBucketList as a simple hash table to store data pointers using identifiers represented as pointers. TBucketList includes methods to add, remove, and locate items in the list, as well as a method that executes a callback for every item in the list.

See Also

Working with Lists

31.1.1.1 TBucketList Members

The following tables list the members exposed by TBucketList.

Protected Methods

	Name	Description
V	AddItem (see page 5664)	Inserts an item and its associated data into a specified bucket.
V	BucketFor (see page 5645)	Returns the index of the bucket in which a specified item belongs.
V	DeleteItem (see page 5666)	Deletes an item from a bucket in the bucket list.
V	FindItem (see page 5667)	Returns the location of an item in the bucket list.

Protected Properties

	Name	Description
	BucketCount (see page 5668)	Indicates the number of buckets in the bucket list.
R	Buckets (see page 5669)	Lists the buckets in the bucket list

Public Methods

	Name	Description
	Add (see page 5664)	Inserts an item and its associated data into the bucket list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 5665)	Replaces all the items in this list with the items from another bucket list.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 5665)	Empties the bucket list of all its items.
	Create (see page 5645)	Creates and initializes a TBucketList (see page 5643) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5663)	Destroys an instance of TCustomBucketList (see page 5661).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Exists (see page 5666)	Indicates whether the bucket list contains a specified item.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Find (see page 5666)	Returns the data associated with a specified item.
	ForEach (see page 5667)	This is the overview for the ForEach method overload.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Remove (see page 5668)	Removes a specified item from the list and returns its data.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Data (see page 5669)	Represents the data associated with a specified item in the bucket list.

31.1.1.2 TBucketList.Create Constructor

Creates and initializes a TBucketList (see page 5643) object.

Pascal

```
constructor Create(ABuckets: TBucketListSizes = bl16);
```

C++

```
__fastcall TBucketList(TBucketListSizes ABuckets = bl16);
```

Description

Call Create to create a bucket list at runtime.

31.1.1.3 TBucketList Methods

The methods of the TBucketList class are listed here.

Protected Methods

	Name	Description
	BucketFor (see page 5645)	Returns the index of the bucket in which a specified item belongs.

31.1.1.3.1 TBucketList.BucketFor Method

Returns the index of the bucket in which a specified item belongs.

Pascal

```
function BucketFor(AItem: Pointer): Integer; override;
```

C++

```
virtual __fastcall int BucketFor(void * AItem);
```

Description

TBucketList (see page 5643) uses BucketFor internally to determine the bucket into which an arbitrary item belongs.

AItem is the identifier of the item for which you want to find a bucket.

BucketFor returns the index of the bucket in the Buckets (see page 5669) property array.

See Also

Buckets (see page 5669)

31.1.2 TClassList Class

TClassList maintains a list of classes.

Class Hierarchy

Pascal

```
TClassList = class(TList);
```

C++

```
class TClassList : public TList;
```

File

Contnrs

Description

Use a TClassList object to store and maintain a list of classes.

See Also

TList (see page 2868)

TCollection (see page 2731)

TStrings (see page 3029)

TComponent (see page 2762)

TObjectList (see page 5677)

TComponentList (see page 5653)

TStack (see page 5701)

Working with Lists

31.1.2.1 TClassList Members

The following tables list the members exposed by TClassList.

Protected Methods

	Name	Description
	Get (see page 2878)	This is Get, a member of class TList.
	GetItems (see page 5649)	This is GetItems, a member of class TClassList.
V	Grow (see page 2879)	This is Grow, a member of class TList.
V	Notify (see page 2884)	Responds when items are added, removed, or extracted from the list.
	Put (see page 2887)	This is Put, a member of class TList.
	SetCapacity (see page 2891)	This is SetCapacity, a member of class TList.
	SetCount (see page 2892)	This is SetCount, a member of class TList.
	SetItems (see page 5652)	This is SetItems, a member of class TClassList.

Public Methods

	Name	Description
	Add (see page 5648)	Inserts a new class at the end of the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2874)	Copies elements of one list to another.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 2874)	Deletes all items from the list.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2874)	Removes the item at the position given by the Index parameter.
V	Destroy (see page 2870)	Destroys an instance of TList (see page 2868).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Error (see page 2875)	This is the overview for the Error method overload.
	Exchange (see page 2876)	Swaps the position of two items in the Items (see page 2898) array.
	Expand (see page 2877)	Increases the Capacity (see page 2895) of the list.
	Extract (see page 5649)	Removes a class from the list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	First (see page 5649)	Returns Items (see page 5652)[0].
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2878)	Returns a TList (see page 2868) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	IndexOf (see page 5650)	Returns the index of the first occurrence in the list of a specified class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 5650)	Firsts a class to the list at a specified position.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Last (see page 5651)	Returns the last class listed in the Items (see page 5652) array.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	Move (see page 2884)	Changes the position of an item in the Items (see page 2898) array.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Pack (see page 2885)	Deletes all nil (Delphi) or NULL (C++) items from the Items (see page 2898) array.
	Remove (see page 5651)	Removes the first occurrence of a specified class from the list.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Sort (see page 2892)	Performs a QuickSort on the list based on the comparison function Compare.

Public Properties

	Name	Description
	Capacity (see page 2895)	Specifies the allocated size of the array of pointers maintained by the TList (see page 2868) object.
	Count (see page 2896)	Indicates the number of entries in the list that are in use.
	Items (see page 5652)	Lists the classes in the TClassList (see page 5646).
R	List (see page 2900)	Specifies the array of pointers that make up the Items (see page 2898) array.

31.1.2.2 TClassList Methods

The methods of the TClassList class are listed here.

Protected Methods

	Name	Description
	GetItems (see page 5649)	This is GetItems, a member of class TClassList.
	SetItems (see page 5652)	This is SetItems, a member of class TClassList.

Public Methods

	Name	Description
	Add (see page 5648)	Inserts a new class at the end of the list.
	Extract (see page 5649)	Removes a class from the list.
	First (see page 5649)	Returns Items (see page 5652)[0].
	IndexOf (see page 5650)	Returns the index of the first occurrence in the list of a specified class.
	Insert (see page 5650)	Firsts a class to the list at a specified position.
	Last (see page 5651)	Returns the last class listed in the Items (see page 5652) array.
	Remove (see page 5651)	Removes the first occurrence of a specified class from the list.

31.1.2.2.1 TClassList.Add Method

Inserts a new class at the end of the list.

Pascal

```
function Add(AClass: TClass): Integer;
```

C++

```
__fastcall int Add(TClass AClass);
```

Description

Call Add to insert a new class at the end of the Items (see page 5652) array. Add returns the index of the new class, where the first class in the list has an index of 0.

Add increments Count and, if necessary, allocates memory by increasing the value of Capacity.

See Also

Capacity (see page 2895)

Count (see page 2896)

Delete (see page 2874)

Insert (see page 2882)

Items (see page 5652)

Remove (see page 2887)

Clear (see page 2874)

Pack (see page 2885)

31.1.2.2.2 TClassList.Extract Method

Removes a class from the list.

Pascal

```
function Extract(Item: TClass): TClass;
```

C++

```
__fastcall TClass Extract(TClass Item);
```

Description

Call Extract to remove a class from the list. After the class is removed, all the classes that follow it are moved up in index position and Count is decremented.

In TClassList (see page 5646), Extract does the same thing as the Delete method, except that it returns the class that was removed from the list.

To remove the reference to a class without deleting the entry from the Items (see page 5652) array and changing the Count, set the Items (see page 5652) property for Index to nil (Delphi) or NULL (C++).

See Also

Items (see page 5652)

Remove (see page 5651)

Count (see page 2896)

Delete (see page 2874)

31.1.2.2.3 TClassList.First Method

Returns Items (see page 5652)[0].

Pascal

```
function First: TClass;
```

C++

```
__fastcall TClass First();
```

Description

Call First to retrieve the first class in the Items (see page 5652) array.

See Also

Items (see page 5652)

Last (see page 5651)

31.1.2.2.4 TClassList.GetItems Method

Pascal

```
function GetItems(Index: Integer): TClass;
```

C++

```
__fastcall TClass GetItems(int Index);
```

Description

This is GetItems, a member of class TClassList.

31.1.2.2.5 TClassList.IndexOf Method

Returns the index of the first occurrence in the list of a specified class.

Pascal

```
function IndexOf(AClass: TClass): Integer;
```

C++

```
__fastcall int IndexOf(TClass AClass);
```

Description

Call IndexOf to get the index for a class in the Items (see page 5652) array. The first class in the array has index 0, the second class has index 1, and so on.

If a class is not in the list, IndexOf returns -1. If a class appears more than once in the list, IndexOf returns the index of the first appearance.

See Also

Delete (see page 2874)

Items (see page 5652)

31.1.2.2.6 TClassList.Insert Method

Firsts a class to the list at a specified position.

Pascal

```
procedure Insert(Index: Integer; AClass: TClass);
```

C++

```
__fastcall Insert(int Index, TClass AClass);
```

Description

Call Insert to First (see page 5649) a class at a specified position in the list, shifting the item that previously occupied that position (and all subsequent items) up. Insert increments Count and, if necessary, allocates memory by increasing the value of Capacity.

The Index parameter is zero-based, so the first position in the list has an index of 0.

To replace an empty reference with a new class without growing the array, set the Items (see page 5652) property directly.

See Also

First (see page 5649)

Capacity (see page 2895)

Count (see page 2896)

Delete (see page 2874)

Items (see page 5652)

Remove (see page 5651)

Pack (see page 2885)

Clear (see page 2874)

31.1.2.2.7 TClassList.Last Method

Returns the last class listed in the Items (see page 5652) array.

Pascal

```
function Last: TClass;
```

C++

```
__fastcall TClass Last();
```

Description

Call Last to retrieve the last class in the Items (see page 5652) array.

See Also

Items (see page 5652)

First (see page 5649)

31.1.2.2.8 TClassList.Remove Method

Removes the first occurrence of a specified class from the list.

Pascal

```
function Remove(AClass: TClass): Integer;
```

C++

```
__fastcall int Remove(TClass AClass);
```

Description

Call Remove to delete a specific class from the list when its index is unknown. The value returned is the index of the class in the Items (see page 5652) array before it was removed. If the specified object is not found on the list, Remove returns –1.

After a class is deleted, all the classes that follow it are moved up in index position and Count is decremented. If a class appears more than once on the list, Remove deletes only the first appearance.

To use an index position (rather than an object reference) to specify the class to be removed, call Delete.

See Also

First (see page 5649)

Count (see page 2896)

Delete (see page 2874)

Insert (see page 5650)

Items (see page 5652)

Pack (see page 2885)

Extract (see page 5649)

31.1.2.2.9 TClassList.SetItems Method

Pascal

```
procedure SetItems(Index: Integer; AClass: TClass);
```

C++

```
__fastcall SetItems(int Index, TClass AClass);
```

Description

This is SetItems, a member of class TClassList.

31.1.2.3 TClassList Properties

The properties of the TClassList class are listed here.

Public Properties

	Name	Description
	Items (see page 5652)	Lists the classes in the TClassList (see page 5646).

31.1.2.3.1 TClassList.Items Property

Lists the classes in the TClassList (see page 5646).

Pascal

```
property Items [Index: Integer]: TClass;
```

C++

```
__property TClass Items[int Index];
```

Description

Use Items to access classes in the list. Items is a zero-based array: The first class is indexed as 0, the second class is indexed as 1, and so forth. You can read or change the value at a specific index, or use Items with the Count property to iterate through the list.

Items can contain empty class references. To remove empty references and reduce the size of the array, call the Pack method.

Note: In Delphi, Items is the default property for TClassList (see page 5646). This means that the property name can be omitted in code. Thus, the following two lines are both acceptable and do the same thing:

```
RegisterClass(ClassList1.Items[i]);
RegisterClass(ClassList1[i]);
```

Note: In C++, Items can be accessed using the [] operator, to achieve an affect similar to the default property in Delphi.

See Also

Count (see page 2896)

List (see page 2900)

Pack (see page 2885)

Delete (see page 2874)

Remove (see page 2887)

Clear (see page 2874)

Add (see page 5648)

Insert (see page 5650)

operator_sb

31.1.3 TComponentList Class

TComponentList maintains a list of components, updating itself when a component is freed.

Class Hierarchy

Pascal

```
TComponentList = class(TObjectList);
```

C++

```
class TComponentList : public TObjectList;
```

File

Contnrs

Description

Use TComponentList to store and maintain a list of components. TComponentList provides properties and methods to add, delete, rearrange, locate, access, and sort components. Like TObjectList (see page 5677), TComponentList controls the memory of its components; unless the OwnsObjects (see page 5686) property is set to false (or the components are removed with the Extract (see page 5656) method). TComponentList frees its components when they are removed from the list or when the TComponentList instance is itself destroyed.

In addition, TComponentList tracks its components. When a component is freed, TComponentList automatically updates itself by deleting the reference to the freed component and packing the list.

See Also

Working with Lists

31.1.3.1 TComponentList Members

The following tables list the members exposed by TComponentList.

Protected Methods

	Name	Description
	Get (see page 2878)	This is Get, a member of class TList.
	GetItem (see page 5682)	This is GetItem, a member of class TObjectList.
	GetItems (see page 5657)	This is GetItems, a member of class TComponentList.
V	Grow (see page 2879)	This is Grow, a member of class TList.
	HandleFreeNotify (see page 5657)	This is HandleFreeNotify, a member of class TComponentList.
V	Notify (see page 5659)	Responds when items are added, removed, or extracted from the list.
	Put (see page 2887)	This is Put, a member of class TList.
	SetCapacity (see page 2891)	This is SetCapacity, a member of class TList.
	SetCount (see page 2892)	This is SetCount, a member of class TList.

	SetItem (see page 5685)	This is SetItem, a member of class TObjectList.
	SetItems (see page 5660)	This is SetItems, a member of class TComponentList.

Public Methods

	Name	Description
	Add (see page 5656)	Inserts a component at the end of the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2874)	Copies elements of one list to another.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 2874)	Deletes all items from the list.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 5679)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2874)	Removes the item at the position given by the Index parameter.
V	Destroy (see page 5655)	Destroys an instance of TComponentList (see page 5653).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Error (see page 2875)	This is the overview for the Error method overload.
	Exchange (see page 2876)	Swaps the position of two items in the Items (see page 2898) array.
	Expand (see page 2877)	Increases the Capacity (see page 2895) of the list.
	Extract (see page 5656)	Removes a component from the list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindInstanceOf (see page 5682)	Finds the first instance of a specified class in the list.
	First (see page 5657)	Returns Items (see page 5660)[0].
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2878)	Returns a TList (see page 2868) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	IndexOf (see page 5658)	Returns the index of the first component in the list with a specified value.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 5658)	Adds a component to the list at a specified position.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Last (see page 5659)	Returns the component at the end of the Items (see page 5660) array.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	Move (see page 2884)	Changes the position of an item in the Items (see page 2898) array.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Pack (see page 2885)	Deletes all nil (Delphi) or NULL (C++) items from the Items (see page 2898) array.
	Remove (see page 5659)	Removes a specified component from the list and (if OwnsObjects (see page 5686) is true) frees the component.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

	Sort (see page 2892)	Performs a QuickSort on the list based on the comparison function Compare.

Public Properties

	Name	Description
	Capacity (see page 2895)	Specifies the allocated size of the array of pointers maintained by the TList (see page 2868) object.
	Count (see page 2896)	Indicates the number of entries in the list that are in use.
	Items (see page 5660)	Lists the component references.
R	List (see page 2900)	Specifies the array of pointers that make up the Items (see page 2898) array.
	OwnsObjects (see page 5686)	Allows TObjectList (see page 5677) to free objects when they are deleted from the list or the list is destroyed.

31.1.3.2 TComponentList.Destroy Destructor

Destroys an instance of TComponentList (see page 5653).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TComponentList();
```

Description

Do not call Destroy directly. Instead, call Free, which checks that the component list is not nil before calling Destroy.

Before destroying the TComponentList (see page 5653), Destroy calls Clear, which frees the components in the list if OwnsObjects (see page 5686) is true.

See Also

Items (see page 5660)

OwnsObjects (see page 5686)

Clear (see page 2874)

31.1.3.3 TComponentList Methods

The methods of the TComponentList class are listed here.

Protected Methods

	Name	Description
	GetItems (see page 5657)	This is GetItems, a member of class TComponentList.
	HandleFreeNotify (see page 5657)	This is HandleFreeNotify, a member of class TComponentList.
V	Notify (see page 5659)	Responds when items are added, removed, or extracted from the list.
	SetItems (see page 5660)	This is SetItems, a member of class TComponentList.

Public Methods

	Name	Description
	Add (see page 5656)	Inserts a component at the end of the list.
	Extract (see page 5656)	Removes a component from the list.
	First (see page 5657)	Returns Items (see page 5660)[0].
	IndexOf (see page 5658)	Returns the index of the first component in the list with a specified value.
	Insert (see page 5658)	Adds a component to the list at a specified position.
	Last (see page 5659)	Returns the component at the end of the Items (see page 5660) array.

	Remove (see page 5659)	Removes a specified component from the list and (if OwnsObjects (see page 5686) is true) frees the component.

31.1.3.3.1 TComponentList.Add Method

Inserts a component at the end of the list.

Pascal

```
function Add(AComponent: TComponent): Integer;
```

C++

```
__fastcall int Add(TComponent * AComponent);
```

Description

Call Add to insert a component at the end of the list. Add places the component after the last item, even if the array contains nil (Delphi) or NULL (C++) references, and returns the index of the inserted component. (The first component in the list has an index of 0.)

Add increments Count and, if necessary, allocates memory by increasing the value of Capacity.

See Also

Capacity (see page 2895)

Count (see page 2896)

Delete (see page 2874)

Insert (see page 5658)

Items (see page 5660)

Remove (see page 5659)

Extract (see page 2877)

Extract (see page 5656)

Clear (see page 2874)

Pack (see page 2885)

31.1.3.3.2 TComponentList.Extract Method

Removes a component from the list.

Pascal

```
function Extract(Item: TComponent): TComponent;
```

C++

```
__fastcall TComponent * Extract(TComponent * Item);
```

Description

Call Extract to remove a component from the list. After the component is removed, all the components that follow it are moved up in index position and Count is decremented.

Extract does not free the component when it is removed from the list. This differs, for example, from the Delete method, which frees the component unless OwnsObjects (see page 5686) is set to false.

To remove the reference to a component without deleting the entry from the Items (see page 5660) array and changing the

Count, set the Items (see page 5660) property for Index to nil (Delphi) or NULL (C++).

See Also

Items (see page 5660)

Remove (see page 5659)

Count (see page 2896)

Delete (see page 2874)

OwnsObjects (see page 5686)

31.1.3.3.3 TComponentList.First Method

Returns Items (see page 5660)[0].

Pascal

```
function First: TComponent;
```

C++

```
__fastcall TComponent * First();
```

Description

Call First to retrieve the first component in the Items (see page 5660) array.

See Also

Items (see page 5660)

Last (see page 5659)

31.1.3.3.4 TComponentList.GetItems Method

Pascal

```
function GetItems(Index: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetItems(int Index);
```

Description

This is GetItems, a member of class TComponentList.

31.1.3.3.5 TComponentList.HandleFreeNotify Method

Pascal

```
procedure HandleFreeNotify(Sender: TObject; AComponent: TComponent);
```

C++

```
__fastcall HandleFreeNotify(TObject * Sender, TComponent * AComponent);
```

Description

This is HandleFreeNotify, a member of class TComponentList.

31.1.3.3.6 TComponentList.IndexOf Method

Returns the index of the first component in the list with a specified value.

Pascal

```
function IndexOf(AComponent: TComponent): Integer;
```

C++

```
__fastcall int IndexOf(TComponent * AComponent);
```

Description

Call IndexOf to get the index for a specified component in the list.

AComponent is the component to locate in the list.

Index of returns the index of AComponent, where the first component has index 0, the second component has index 1, and so on. If a component is not in the list, IndexOf returns -1. If a component appears more than once, IndexOf returns the index of the first appearance.

See Also

Delete (see page 2874)

Items (see page 5660)

FindInstanceOf (see page 5682)

31.1.3.3.7 TComponentList.Insert Method

Adds a component to the list at a specified position.

Pascal

```
procedure Insert(Index: Integer; AComponent: TComponent);
```

C++

```
__fastcall Insert(int Index, TComponent * AComponent);
```

Description

Call Insert to add a component at a specified position in the list, shifting the item that previously occupied that position (and all subsequent items) up. Insert increments Count and, if necessary, allocates memory by increasing the value of Capacity.

The Index parameter is zero-based, so the first position in the list has an index of 0.

To replace a nil (Delphi) or NULL (C++) reference with a new component without growing the array, set the Items (see page 5660) property directly.

See Also

Add (see page 5656)

Capacity (see page 2895)

Count (see page 2896)

Delete (see page 2874)

Items (see page 5660)

Remove (see page 5659)

Pack (see page 2885)

Clear (see page 2874)

Extract (see page 5656)

31.1.3.3.8 TComponentList.Last Method

Returns the component at the end of the Items (see page 5660) array.

Pascal

```
function Last: TComponent;
```

C++

```
__fastcall TComponent * Last();
```

Description

Call Last to retrieve the last component in the Items (see page 5660) array.

See Also

Items (see page 5660)

First (see page 5657)

31.1.3.3.9 TComponentList.Notify Method

Responds when items are added, removed, or extracted from the list.

Pascal

```
procedure Notify(Ptr: Pointer; Action: TListNotification); override;
```

C++

```
virtual __fastcall Notify(void * Ptr, TListNotification Action);
```

Description

Notify is called automatically when the items in the collection change.

Ptr points to item affected by the change.

Action indicates the change.

As implemented in TList, Notify does nothing. Descendant classes can override Notify in order to implement an appropriate response to the change notification.

31.1.3.3.10 TComponentList.Remove Method

Removes a specified component from the list and (if OwnsObjects (see page 5686) is true) frees the component.

Pascal

```
function Remove(AComponent: TComponent): Integer;
```

C++

```
__fastcall int Remove(TComponent * AComponent);
```

Description

Call Remove to delete a specific component from the list when its index is unknown. The value returned is the index of the component in the Items (see page 5660) array before it was removed. If the specified component is not found on the list, Remove returns –1. If OwnsObjects (see page 5686) is true, Remove frees the component in addition to removing it from the

list.

After a component is deleted, all the components that follow it are moved up in index position and Count is decremented. If a component appears more than once on the list, Remove deletes only the first appearance. Hence, if OwnsObjects (see page 5686) is true, removing a component that appears more than once results in empty references later in the list.

To use an index position (rather than an object reference) to specify the component to be removed, call Delete.

To remove a component from the list without freeing it, call Extract (see page 5656).

See Also

Add (see page 5656)

Count (see page 2896)

Delete (see page 2874)

Insert (see page 5658)

Items (see page 5660)

OwnsObjects (see page 5686)

Extract (see page 5656)

Pack (see page 2885)

31.1.3.3.11 TComponentList.SetItems Method

Pascal

```
procedure SetItems(Index: Integer; AComponent: TComponent);
```

C++

```
__fastcall SetItems(int Index, TComponent * AComponent);
```

Description

This is SetItems, a member of class TComponentList.

31.1.3.4 TComponentList Properties

The properties of the TComponentList class are listed here.

Public Properties

	Name	Description
	Items (see page 5660)	Lists the component references.

31.1.3.4.1 TComponentList.Items Property

Lists the component references.

Pascal

```
property Items [Index: Integer]: TComponent;
```

C++

```
__property TComponent * Items[int Index];
```

Description

Use Items to access components in the list. Items is a zero-based array: The first component is indexed as 0, the second component is indexed as 1, and so forth. You can read or change the value at a specific index, or use Items with the Count property to iterate through the list.

If OwnsObjects (see page 5686) is true, reassigning an Items value frees the component that previously occupied that position in the list.

Items can contain nil (Delphi) or NULL (C++) references. To remove nil (Delphi) or NULL (C++) references and reduce the size of the array, call the Pack method.

Note: In Delphi, Items is the default property of TComponentList (see page 5653). This means that the property name can be eliminated in code that uses this property. Thus, the following two lines are both acceptable and do the same thing:

```
ComponentList1->Items[i] := Screen.CustomForms[i];
ComponentList1[i] := Screen.CustomForms[i];
```

Note: In C++, Items can be accessed using the [] operator, to achieve an affect similar to the default property in Delphi.

See Also

OwnsObjects (see page 5686)

Add (see page 5656)

Insert (see page 5658)

Delete (see page 2874)

Remove (see page 5659)

List (see page 2900)

Extract (see page 2877)

Extract (see page 5656)

Count (see page 2896)

Clear (see page 2874)

Pack (see page 2885)

operator_sb

31.1.4 TCustomBucketList Class

TCustomBucketList is the base class for lists that store items under a limited number of categories.

Class Hierarchy

Pascal

```
TCustomBucketList = class(TObject);
```

C++

```
class TCustomBucketList : public TObject;
```

File

Contnrs

Description

Use TCustomBucketList as a base class for lists of items that fit into a limited number of categories (or buckets). Each bucket can contain multiple items, and there is a well-defined function (the protected BucketFor (see page 5665) method) that determines the bucket into which an item belongs. TCustomBucketList descendants can include lists of items that fall into natural categories, or implementations of a hash table.

Do not create instances of TCustomBucketList. In TCustomBucketList, the BucketFor (see page 5665) method is abstract (pure virtual), meaning that descendant classes must override it and provide an implementation.

31.1.4.1 TCustomBucketList Members

The following tables list the members exposed by TCustomBucketList.

Protected Methods

	Name	Description
V	AddItem (see page 5664)	Inserts an item and its associated data into a specified bucket.
A	BucketFor (see page 5665)	Returns the index of the bucket in which a specified item belongs.
V	DeleteItem (see page 5666)	Deletes an item from a bucket in the bucket list.
V	FindItem (see page 5667)	Returns the location of an item in the bucket list.

Protected Properties

	Name	Description
	BucketCount (see page 5668)	Indicates the number of buckets in the bucket list.
R	Buckets (see page 5669)	Lists the buckets in the bucket list

Public Methods

	Name	Description
	Add (see page 5664)	Inserts an item and its associated data into the bucket list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 5665)	Replaces all the items in this list with the items from another bucket list.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 5665)	Empties the bucket list of all its items.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5663)	Destroys an instance of TCustomBucketList (see page 5661).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Exists (see page 5666)	Indicates whether the bucket list contains a specified item.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Find (see page 5666)	Returns the data associated with a specified item.
	ForEach (see page 5667)	This is the overview for the ForEach method overload.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Remove (see page 5668)	Removes a specified item from the list and returns its data.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Data (see page 5669)	Represents the data associated with a specified item in the bucket list.

31.1.4.2 TCustomBucketList.Destroy Destructor

Destroys an instance of TCustomBucketList (see page 5661).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomBucketList();
```

Description

Do not call Destroy directly. Instead, call Free, which checks that the component list is not nil before calling Destroy.

Before destroying the TCustomBucketList (see page 5661), Destroy calls Clear (see page 5665), which deletes all of the items from all of the buckets in the list.

See Also

Clear (see page 5665)

31.1.4.3 TCustomBucketList Methods

The methods of the TCustomBucketList class are listed here.

Protected Methods

	Name	Description
	AddItem (see page 5664)	Inserts an item and its associated data into a specified bucket.
	BucketFor (see page 5665)	Returns the index of the bucket in which a specified item belongs.
	DeleteItem (see page 5666)	Deletes an item from a bucket in the bucket list.
	FindItem (see page 5667)	Returns the location of an item in the bucket list.

Public Methods

	Name	Description
	Add (see page 5664)	Inserts an item and its associated data into the bucket list.

	Assign (see page 5665)	Replaces all the items in this list with the items from another bucket list.
	Clear (see page 5665)	Empties the bucket list of all its items.
	Exists (see page 5666)	Indicates whether the bucket list contains a specified item.
	Find (see page 5666)	Returns the data associated with a specified item.
	ForEach (see page 5667)	This is the overview for the ForEach method overload.
	Remove (see page 5668)	Removes a specified item from the list and returns its data.

31.1.4.3.1 TCustomBucketList.Add Method

Inserts an item and its associated data into the bucket list.

Pascal

```
function Add(AItem: Pointer; AData: Pointer): Pointer;
```

C++

```
__fastcall void * Add(void * AItem, void * AData);
```

Description

Call Add to add an item and its associated data into the bucket list. Add locates the bucket into which the specified item belongs. If the item is already in the bucket, Add raises an EListError exception. Otherwise, it adds the item to the correct bucket and returns AData.

See Also

BucketFor (see page 5665)

EListError (see page 2701)

31.1.4.3.2 TCustomBucketList.AddItem Method

Inserts an item and its associated data into a specified bucket.

Pascal

```
function AddItem(ABucket: Integer; AItem: Pointer; AData: Pointer): Pointer; virtual;
```

C++

```
virtual __fastcall void * AddItem(int ABucket, void * AItem, void * AData);
```

Description

TCustomBucketList (see page 5661) uses AddItem internally to add items into buckets. AddItem does not check whether the specified item correctly belongs in the specified bucket; it simply handles the mechanics of adding the item into the bucket. If you need to determine the bucket in which an item belongs, use the Add (see page 5664) method instead.

ABucket is the index of the bucket in which the item belongs.

AItem is the item to add to ABucket.

AData is the data to associate with AItem.

AddItem returns the value of AData.

See Also

Add (see page 5664)

31.1.4.3.3 TCustomBucketList.Assign Method

Replaces all the items in this list with the items from another bucket list.

Pascal

```
procedure Assign(AList: TCustomBucketList);
```

C++

```
__fastcall Assign(TCustomBucketList AList);
```

Description

Assign empties the bucket list of any items it currently holds and then adds in all the items in the bucket list specified by the AList parameter.

Note: Assign merely copies the items from another bucket list, not the organization of the bucket list. Thus, the items may end up in entirely different buckets.

31.1.4.3.4 TCustomBucketList.BucketFor Method

Returns the index of the bucket in which a specified item belongs.

Pascal

```
function BucketFor(AItem: Pointer): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int BucketFor(void * AItem) = 0;
```

Description

TCustomBucketList (see page 5661) uses BucketFor internally to determine the bucket into which an arbitrary item belongs. TCustomBucketList (see page 5661) introduces BucketFor as an abstract (pure virtual) method, which means it has no implementation. Descendant classes must override BucketFor to provide an implementation that determines the bucket for an arbitrary item.

AItem is the item for which you want to find a bucket.

BucketFor returns the index of the bucket in the Buckets (see page 5669) property array.

See Also

Buckets (see page 5669)

31.1.4.3.5 TCustomBucketList.Clear Method

Empties the bucket list of all its items.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the bucket list. Clear deletes all of the items from all of the buckets in the bucket list.

Note: Clearing the bucket list does not free any of the items in the list, it merely removes them from the bucket list.

See Also

Remove (see page 5668)

31.1.4.3.6 TCustomBucketList.DeleteItem Method

Deletes an item from a bucket in the bucket list.

Pascal

```
function DeleteItem(ABucket: Integer; AIndex: Integer): Pointer; virtual;
```

C++

```
virtual __fastcall void * DeleteItem(int ABucket, int AIndex);
```

Description

TCustomBucketList (see page 5661) uses DeleteItem to remove items from the bucket list. DeleteItem removes the specified item from the specified bucket and moves all items that came after the deleted item up by one position within the bucket.

ABucket is the bucket that contains the item to delete.

AIndex is the index of the item to delete within the bucket specified by ABucket.

DeleteItem returns the data associated with the deleted item.

See Also

Remove (see page 5668)

Clear (see page 5665)

31.1.4.3.7 TCustomBucketList.Exists Method

Indicates whether the bucket list contains a specified item.

Pascal

```
function Exists(AItem: Pointer): Boolean;
```

C++

```
__fastcall Boolean Exists(void * AItem);
```

Description

Call Exists to determine whether the item specified by AItem is already in the bucket list. Exists returns true if the bucket list contains AItem, false if AItem is not in the list.

See Also

Find (see page 5666)

31.1.4.3.8 TCustomBucketList.Find Method

Returns the data associated with a specified item.

Pascal

```
function Find(AItem: Pointer; out AData: Pointer): Boolean;
```

C++

```
__fastcall Boolean Find(void * AItem, void * AData);
```

Description

Call Find to retrieve the data associated with an item in the list. Calling the Find method differs from reading the Data (see page 5669) property in that it does not raise an exception if the item is not in the bucket list.

AItem is the item to find in the bucket list.

AData returns the data associated with AItem.

Find returns true if AItem is in the bucket list, false otherwise. The value of AData is not set if Find returns false.

See Also

Exists (see page 5666)

Data (see page 5669)

31.1.4.3.9 TCustomBucketList.FindItem Method

Returns the location of an item in the bucket list.

Pascal

```
function FindItem(AItem: Pointer; out ABucket: Integer; out AIndex: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean FindItem(void * AItem, int ABucket, int AIndex);
```

Description

TCustomBucketList (see page 5661) uses FindItem internally to locate items in the bucket list.

AItem is the item to find in the bucket list.

ABucket returns the index of the bucket that contains the item.

AIndex returns the index of the item in the bucket specified by ABucket.

FindItem returns true if AItem is in the bucket list, false otherwise.

See Also

Find (see page 5666)

Buckets (see page 5669)

31.1.4.3.10 ForEach Method

This is the overview for the ForEach method overload.

Overload List

	Name	Description
	TCustomBucketList.ForEach (TBucketEvent) (see page 5667)	Executes a callback for every item in the bucket list.
	TCustomBucketList.ForEach (TBucketProc, Pointer) (see page 5667)	Executes a callback for every item in the bucket list.

31.1.4.3.10.1 TCustomBucketList.ForEach Method (TBucketEvent)

Executes a callback for every item in the bucket list.

Pascal

```
function ForEach(AProc: TBucketProc; AInfo: Pointer = nil): Boolean; overload;
function ForEach(AEvent: TBucketEvent): Boolean; overload;
```

C++

```
__fastcall Boolean ForEach(TBucketProc AProc, void * AInfo = nil);
__fastcall Boolean ForEach(TBucketEvent AEvent);
```

Description

Call ForEach to execute the procedure specified by AProc for each item in the bucket list. ForEach iterates through all the items in the bucket list, passing in each item and its associated data to the specified callback.

AProc is the callback to execute.

AInfo has no predefined meaning. It is passed to the AInfo parameter of the callback.

ForEach returns true if it executes the callback for every item in the bucket list. It returns false if the callback set AContinue to false for some item, causing any subsequent items to be skipped.

31.1.4.3.11 TCustomBucketList.Remove Method

Removes a specified item from the list and returns its data.

Pascal

```
function Remove(AItem: Pointer): Pointer;
```

C++

```
__fastcall void * Remove(void * AItem);
```

Description

Call Remove to delete a specific item from the list.

AItem is the item to remove.

Remove returns the data associated with AItem. If AItem is not in the bucket list, Remove returns nil (Delphi) or NULL (C++).

See Also

Clear (see page 5665)

31.1.4.4 TCustomBucketList Properties

The properties of the TCustomBucketList class are listed here.

Protected Properties

	Name	Description
	BucketCount (see page 5668)	Indicates the number of buckets in the bucket list.
R	Buckets (see page 5669)	Lists the buckets in the bucket list

Public Properties

	Name	Description
	Data (see page 5669)	Represents the data associated with a specified item in the bucket list.

31.1.4.4.1 TCustomBucketList.BucketCount Property

Indicates the number of buckets in the bucket list.

Pascal

```
property BucketCount: Integer;
```

C++

```
__property int BucketCount;
```

Description

Use BucketCount to determine the number of buckets, or categories, that can hold items in the list. BucketCount determines the number of buckets that are listed by the Buckets (see page 5669) property.

See Also

Buckets (see page 5669)

31.1.4.4.2 TCustomBucketList.Buckets Property

Lists the buckets in the bucket list

Pascal

```
property Buckets: TBucketArray;
```

C++

```
__property TBucketArray Buckets;
```

Description

Use Buckets to access the buckets in the bucket list. Buckets is a dynamic array of TBucket (see page 5706) values, each of which can hold a set of items.

See Also

BucketCount (see page 5668)

BucketFor (see page 5665)

Find (see page 5666)

31.1.4.4.3 TCustomBucketList.Data Property

Represents the data associated with a specified item in the bucket list.

Pascal

```
property Data [AItem: Pointer]: Pointer;
```

C++

```
__property void * Data[void * AItem];
```

Description

Use Data to get or set the data associated with an item in the bucket list.

AItem points to an item currently in the bucket list. If AItem does not specify an item in one of the buckets of the bucket list, trying to read or set Data causes TCustomBucketList (see page 5661) to raise an EListError exception.

Note: In Delphi, Data is the default property of TCustomBucketList (see page 5661). This means that the property name can be omitted when using this property. For example, the following two lines of code are both acceptable and do the same thing:

```
BucketList1.Data[Name as Pointer] := Pointer(Form1.FindChildControl(Name));
```

```
BucketList1[Name as Pointer] := Pointer(Form1.FindChildControl(Name));
```

Note: In C++, Data can be accessed using the [] operator, to achieve an affect similar to the default property in Delphi.

See Also

Find (see page 5666)

EListError (see page 2701)

operator_sb

31.1.5 TIntegerBucketList Class

Class Hierarchy

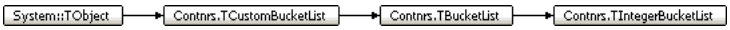

Pascal

```
TIntegerBucketList = class(TBucketList);
```

C++

```
class TIntegerBucketList : public TBucketList;
```

File

Contnrs

Description

This is class Contnrs.TIntegerBucketList.

31.1.5.1 TIntegerBucketList Members

The following tables list the members exposed by TIntegerBucketList.

Protected Methods

	Name	Description
V	AddItem (see page 5664)	Inserts an item and its associated data into a specified bucket.
V	BucketFor (see page 5645)	Returns the index of the bucket in which a specified item belongs.
A	BucketFor (see page 5665)	Returns the index of the bucket in which a specified item belongs.
V	DeleteItem (see page 5666)	Deletes an item from a bucket in the bucket list.
V	FindItem (see page 5667)	Returns the location of an item in the bucket list.
	GetData (see page 5672)	This is GetData, a member of class TIntegerBucketList.
	SetData (see page 5672)	This is SetData, a member of class TIntegerBucketList.

Protected Properties

	Name	Description
	BucketCount (see page 5668)	Indicates the number of buckets in the bucket list.
R	Buckets (see page 5669)	Lists the buckets in the bucket list

Public Methods

	Name	Description
	Add (see page 5672)	This is Add, a member of class TIntegerBucketList.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 5665)	Replaces all the items in this list with the items from another bucket list.

	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 5665)	Empties the bucket list of all its items.
	Create (see page 5645)	Creates and initializes a TBucketList (see page 5643) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 5663)	Destroys an instance of TCustomBucketList (see page 5661).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Exists (see page 5666)	Indicates whether the bucket list contains a specified item.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Find (see page 5666)	Returns the data associated with a specified item.
	ForEach (see page 5667)	This is the overview for the ForEach method overload.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Remove (see page 5672)	This is Remove, a member of class TIntegerBucketList.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Data (see page 5673)	Represents the data associated with a specified item in the bucket list.

31.1.5.2 TIntegerBucketList Methods

The methods of the TIntegerBucketList class are listed here.

Protected Methods

	Name	Description
	GetData (see page 5672)	This is GetData, a member of class TIntegerBucketList.
	SetData (see page 5672)	This is SetData, a member of class TIntegerBucketList.

Public Methods

	Name	Description
	Add (see page 5672)	This is Add, a member of class TIntegerBucketList.

	Remove (see page 5672)	This is Remove, a member of class TIntegerBucketList.

31.1.5.2.1 TIntegerBucketList.Add Method

Pascal

```
function Add(AItem: Integer; AData: Integer): Integer;
```

C++

```
__fastcall int Add(int AItem, int AData);
```

Description

This is Add, a member of class TIntegerBucketList.

31.1.5.2.2 TIntegerBucketList.GetData Method

Pascal

```
function GetData(AItem: Integer): Integer;
```

C++

```
__fastcall int GetData(int AItem);
```

Description

This is GetData, a member of class TIntegerBucketList.

31.1.5.2.3 TIntegerBucketList.Remove Method

Pascal

```
function Remove(AItem: Integer): Integer;
```

C++

```
__fastcall int Remove(int AItem);
```

Description

This is Remove, a member of class TIntegerBucketList.

31.1.5.2.4 TIntegerBucketList.SetData Method

Pascal

```
procedure SetData(AItem: Integer; const AData: Integer);
```

C++

```
__fastcall SetData(int AItem, const int AData);
```

Description

This is SetData, a member of class TIntegerBucketList.

31.1.5.3 TIntegerBucketList Properties

The properties of the TIntegerBucketList class are listed here.

Public Properties

	Name	Description
	Data (see page 5673)	Represents the data associated with a specified item in the bucket list.

31.1.5.3.1 TIntegerBucketList.Data Property

Represents the data associated with a specified item in the bucket list.

Pascal

```
property Data [AItem: Integer]: Integer;
```

C++

```
__property int Data[int AItem];
```

Description

Use Data to get or set the data associated with an item in the bucket list.

AItem points to an item currently in the bucket list. If AItem does not specify an item in one of the buckets of the bucket list, trying to read or set Data causes TCustomBucketList to raise an EListError exception.

Note: In Delphi, Data is the default property of TCustomBucketList. This means that the property name can be omitted when using this property. For example, the following two lines of code are both acceptable and do the same thing:

```
BucketList1.Data[Name as Pointer] := Pointer(Form1.FindChildControl(Name));
BucketList1[Name as Pointer] := Pointer(Form1.FindChildControl(Name));
```

Note: In C++, Data can be accessed using the [] operator, to achieve an affect similar to the default property in Delphi.

See Also

Find (see page 5666)

EListError (see page 2701)

operator_sb

31.1.6 TObjectBucketList Class

TObjectBucketList is a simple hash table that stores objects that are identified by other objects.

Class Hierarchy

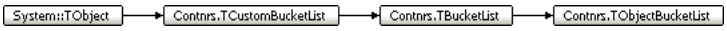

Pascal

```
TObjectBucketList = class(TBucketList);
```

C++

```
class TObjectBucketList : public TBucketList;
```

File

Contnrs

Description

Use TObjectBucketList as a simple hash table to store objects that are indexed by other objects. TObjectBucketList includes methods to add, remove, and locate items in the list, as well as a method that executes a callback for every item in the list.

See Also

Working with Lists

31.1.6.1 TObjectBucketList Members

The following tables list the members exposed by TObjectBucketList.

Protected Methods

	Name	Description
V	AddItem (see page 5664)	Inserts an item and its associated data into a specified bucket.
V	BucketFor (see page 5645)	Returns the index of the bucket in which a specified item belongs.
A	BucketFor (see page 5665)	Returns the index of the bucket in which a specified item belongs.
V	DeleteItem (see page 5666)	Deletes an item from a bucket in the bucket list.
V	FindItem (see page 5667)	Returns the location of an item in the bucket list.
	GetData (see page 5676)	This is GetData, a member of class TObjectBucketList.
	SetData (see page 5676)	This is SetData, a member of class TObjectBucketList.

Protected Properties

	Name	Description
	BucketCount (see page 5668)	Indicates the number of buckets in the bucket list.
R	Buckets (see page 5669)	Lists the buckets in the bucket list

Public Methods

	Name	Description
	Add (see page 5675)	Inserts an item and its associated data into the object bucket list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 5665)	Replaces all the items in this list with the items from another bucket list.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 5665)	Empties the bucket list of all its items.
	Create (see page 5645)	Creates and initializes a TBucketList (see page 5643) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5663)	Destroys an instance of TCustomBucketList (see page 5661).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Exists (see page 5666)	Indicates whether the bucket list contains a specified item.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Find (see page 5666)	Returns the data associated with a specified item.
	ForEach (see page 5667)	This is the overview for the ForEach method overload.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Remove (see page 5676)	Removes a specified item from the list and returns its data.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Data (see page 5677)	Represents the data associated with a specified item in the object bucket list.

31.1.6.2 TObjectBucketList Methods

The methods of the TObjectBucketList class are listed here.

Protected Methods

	Name	Description
	GetData (see page 5676)	This is GetData, a member of class TObjectBucketList.
	SetData (see page 5676)	This is SetData, a member of class TObjectBucketList.

Public Methods

	Name	Description
	Add (see page 5675)	Inserts an item and its associated data into the object bucket list.
	Remove (see page 5676)	Removes a specified item from the list and returns its data.

31.1.6.2.1 TObjectBucketList.Add Method

Inserts an item and its associated data into the object bucket list.

Pascal

```
function Add(AItem: TObject; AData: TObject): TObject;
```

C++

```
__fastcall TObject * Add(TObject * AItem, TObject * AData);
```

Description

Call Add to add an item and its associated data into the object bucket list. Add locates the bucket into which the specified item belongs. If the item is already in the bucket, Add raises an EListError exception. Otherwise, it adds the item to the correct bucket and returns AData.

See Also

EListError (see page 2701)

Data (see page 5677)

31.1.6.2.2 TObjectBucketList.GetData Method

Pascal

```
function GetData(AItem: TObject): TObject;
```

C++

```
__fastcall TObject * GetData(TObject * AItem);
```

Description

This is GetData, a member of class TObjectBucketList.

31.1.6.2.3 TObjectBucketList.Remove Method

Removes a specified item from the list and returns its data.

Pascal

```
function Remove(AItem: TObject): TObject;
```

C++

```
__fastcall TObject * Remove(TObject * AItem);
```

Description

Call Remove to delete a specific item from the list.

AItem is the item to remove.

Remove returns the data associated with AItem. If AItem is not in the object bucket list, Remove returns nil (Delphi) or NULL (C++).

See Also

Clear (see page 5665)

Add (see page 5675)

31.1.6.2.4 TObjectBucketList.SetData Method

Pascal

```
procedure SetData(AItem: TObject; const AData: TObject);
```

C++

```
__fastcall SetData(TObject * AItem, const TObject * AData);
```

Description

This is SetData, a member of class TObjectBucketList.

31.1.6.3 TObjectBucketList Properties

The properties of the TObjectBucketList class are listed here.

Public Properties

	Name	Description
	Data (see page 5677)	Represents the data associated with a specified item in the object bucket list.

31.1.6.3.1 TObjectBucketList.Data Property

Represents the data associated with a specified item in the object bucket list.

Pascal

```
property Data [AItem: TObject]: TObject;
```

C++

```
__property TObject * Data[TObject * AItem];
```

Description

Use Data to get or set the data object associated with an item in the bucket list.

AItem is the identifier of an item currently in the object bucket list. If AItem does not specify an item in one of the buckets of the bucket list, trying to read or set Data causes TObjectBucketList (see page 5673) to raise an EListError exception.

Note: In Delphi, Data is the default property of TObjectBucketList (see page 5673). This means that the property name can be omitted when using this property. Thus, for example, instead of

Note: ObjectBucketList1.Data[MyObject];

Note: you can write

Note: ObjectBucketList1[MyObject];

See Also

Find (see page 5666)

EListError (see page 2701)

operator_sb

31.1.7 TObjectList Class

TObjectList maintains a list of (owned) objects.

Class Hierarchy

Pascal

```
TObjectList = class(TList);
```

C++

```
class TObjectList : public TList;
```

File

Contnrs

Description

Use TObjectList to store and maintain a list of objects. TObjectList provides properties and methods to add, delete, rearrange, locate, access, and sort objects. If the OwnsObjects (see page 5686) property is set to true (the default), TObjectList controls the memory of its objects, freeing an object when its index is reassigned; when it is removed from the list with the Delete, Remove (see page 5684), or Clear method; or when the TObjectList instance is itself destroyed.

See Also

TList (see page 2868)

TComponentList (see page 5653)

TClassList (see page 5646)

TStack (see page 5701)

TObjectStack (see page 5690)

TObjectQueue (see page 5687)

TInterfaceList (see page 2855)

Working with Lists

31.1.7.1 TObjectList Members

The following tables list the members exposed by TObjectList.

Protected Methods

	Name	Description
	Get (see page 2878)	This is Get, a member of class TList.
	GetItem (see page 5682)	This is GetItem, a member of class TObjectList.
V	Grow (see page 2879)	This is Grow, a member of class TList.
V	Notify (see page 5684)	Responds when items are added, removed, or extracted from the list.
	Put (see page 2887)	This is Put, a member of class TList.
	SetCapacity (see page 2891)	This is SetCapacity, a member of class TList.
	SetCount (see page 2892)	This is SetCount, a member of class TList.
	SetItem (see page 5685)	This is SetItem, a member of class TObjectList.

Public Methods

	Name	Description
	Add (see page 5680)	Inserts an object at the end of the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2874)	Copies elements of one list to another.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 2874)	Deletes all items from the list.
	Create (see page 5679)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2874)	Removes the item at the position given by the Index parameter.

	Destroy (see page 2870)	Destroys an instance of TList (see page 2868).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Error (see page 2875)	This is the overview for the Error method overload.
	Exchange (see page 2876)	Swaps the position of two items in the Items (see page 2898) array.
	Expand (see page 2877)	Increases the Capacity (see page 2895) of the list.
	Extract (see page 5681)	Removes a specified object from the list without freeing it.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindInstanceOf (see page 5682)	Finds the first instance of a specified class in the list.
	First (see page 5682)	Returns Items (see page 5685)[0].
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2878)	Returns a TList (see page 2868) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	IndexOf (see page 5683)	Returns the index of the first object in the list with a specified value.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 5683)	Adds an object to the list at a specified position.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Last (see page 5684)	Returns the object at the end of the Items (see page 5685) array.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	Move (see page 2884)	Changes the position of an item in the Items (see page 2898) array.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Pack (see page 2885)	Deletes all nil (Delphi) or NULL (C++) items from the Items (see page 2898) array.
	Remove (see page 5684)	Removes a specified object from the list and (if OwnsObjects (see page 5686) is true) frees the object.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Sort (see page 2892)	Performs a QuickSort on the list based on the comparison function Compare.

Public Properties

	Name	Description
	Capacity (see page 2895)	Specifies the allocated size of the array of pointers maintained by the TList (see page 2868) object.
	Count (see page 2896)	Indicates the number of entries in the list that are in use.
	Items (see page 5685)	Lists the object references.
R	List (see page 2900)	Specifies the array of pointers that make up the Items (see page 2898) array.
	OwnsObjects (see page 5686)	Allows TObjectList (see page 5677) to free objects when they are deleted from the list or the list is destroyed.

31.1.7.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TObjectList.Create () (see page 5680)	Creates and initializes a TObjectList (see page 5677).

	TObjectList.Create (Boolean) (see page 5680)	Creates and initializes a TObjectList (see page 5677).

31.1.7.2.1 TObjectList.Create Constructor ()

Creates and initializes a TObjectList (see page 5677).

Pascal

```
constructor Create; overload;
constructor Create(AOwnsObjects: Boolean); overload;
```

C++

```
__fastcall TObjectList();
__fastcall TObjectList(Boolean AOwnsObjects);
```

Description

Create creates a TObjectList (see page 5677) instance. If no parameter is specified, or if the constructor is called with AOwnsObjects set to true, the new TObjectList (see page 5677) is initialized with OwnsObjects (see page 5686) set to true.

See Also

OwnsObjects (see page 5686)

31.1.7.3 TObjectList Methods

The methods of the TObjectList class are listed here.

Protected Methods

	Name	Description
	GetItem (see page 5682)	This is GetItem, a member of class TObjectList.
V	Notify (see page 5684)	Responds when items are added, removed, or extracted from the list.
	SetItem (see page 5685)	This is SetItem, a member of class TObjectList.

Public Methods

	Name	Description
	Add (see page 5680)	Inserts an object at the end of the list.
	Extract (see page 5681)	Removes a specified object from the list without freeing it.
	FindInstanceOf (see page 5682)	Finds the first instance of a specified class in the list.
	First (see page 5682)	Returns Items (see page 5685)[0].
	IndexOf (see page 5683)	Returns the index of the first object in the list with a specified value.
	Insert (see page 5683)	Adds an object to the list at a specified position.
	Last (see page 5684)	Returns the object at the end of the Items (see page 5685) array.
	Remove (see page 5684)	Removes a specified object from the list and (if OwnsObjects (see page 5686) is true) frees the object.

31.1.7.3.1 TObjectList.Add Method

Inserts an object at the end of the list.

Pascal

```
function Add(AObject: TObject): Integer;
```

C++

```
__fastcall int Add(TObject * AObject);
```

Description

Call Add to insert an object at the end of the list. Add places the object after the last item, even if the array contains nil (Delphi) or NULL (C++) references, and returns the index of the inserted object. (The first object in the list has an index of 0.)

Add increments Count and, if necessary, allocates memory by increasing the value of Capacity.

See Also

Capacity (see page 2895)

Count (see page 2896)

Delete (see page 2874)

Insert (see page 5683)

Items (see page 5685)

Remove (see page 5684)

Extract (see page 5681)

Clear (see page 2874)

Pack (see page 2885)

31.1.7.3.2 TObjectList.Extract Method

Removes a specified object from the list without freeing it.

Pascal

```
function Extract(Item: TObject): TObject;
```

C++

```
__fastcall TObject * Extract(TObject * Item);
```

Description

Call Extract to remove an object from the list without freeing the object itself. After an object is removed, all the objects that follow it are moved up in index position and Count is decremented.

See Also

Add (see page 5680)

Capacity (see page 2895)

Clear (see page 2874)

Count (see page 2896)

Insert (see page 5683)

Items (see page 5685)

Remove (see page 5684)

Delete (see page 2874)

OwnsObjects (see page 5686)

31.1.7.3.3 TObjectList.FindInstanceOf Method

Finds the first instance of a specified class in the list.

Pascal

```
function FindInstanceOf(AClass: TClass; AExact: Boolean = True; AStartAt: Integer = 0):
Integer;
```

C++

```
__fastcall int FindInstanceOf(TClass AClass, Boolean AExact = True, int AStartAt = 0);
```

Description

FindInstanceOf returns the index of the first instance of AClass appearing after AStartAt in the Items (see page 5685) array. If AExact is true, FindInstanceOf returns instances only of AClass itself, ignoring instances of descendant classes. If no instance of the specified class is found, FindInstanceOf returns –1.

See Also

IndexOf (see page 5683)

Items (see page 5685)

Count (see page 2896)

Sort (see page 2892)

31.1.7.3.4 TObjectList.First Method

Returns Items (see page 5685)[0].

Pascal

```
function First: TObject;
```

C++

```
__fastcall TObject * First();
```

Description

Call First to retrieve the first object reference in the Items (see page 5685) array.

See Also

Items (see page 5685)

Last (see page 5684)

31.1.7.3.5 TObjectList.GetItem Method

Pascal

```
function GetItem(Index: Integer): TObject;
```

C++

```
__fastcall TObject * GetItem(int Index);
```

Description

This is GetItem, a member of class TObjectList.

31.1.7.3.6 TObjectList.IndexOf Method

Returns the index of the first object in the list with a specified value.

Pascal

```
function IndexOf(AObject: TObject): Integer;
```

C++

```
__fastcall int IndexOf(TObject * AObject);
```

Description

Call IndexOf to get the index for a specified object in the list, where the first object has index 0, the second object has index 1, and so on. If an object is not in the list, IndexOf returns -1. If an object appears more than once, IndexOf returns the index of the first appearance.

See Also

Delete (see page 2874)

Items (see page 5685)

FindInstanceOf (see page 5682)

31.1.7.3.7 TObjectList.Insert Method

Adds an object to the list at a specified position.

Pascal

```
procedure Insert(Index: Integer; AObject: TObject);
```

C++

```
__fastcall Insert(int Index, TObject * AObject);
```

Description

Call Insert to add an object at a specified position in the list, shifting the item that previously occupied that position (and all subsequent items) up. Insert increments Count and, if necessary, allocates memory by increasing the value of Capacity.

The Index parameter is zero-based, so the first position in the list has an index of 0.

To replace a nil (Delphi) or NULL (C++) reference with a new object without growing the array, set the Items (see page 5685) property directly.

See Also

Add (see page 5680)

Capacity (see page 2895)

Count (see page 2896)

Delete (see page 2874)

Items (see page 5685)

Remove (see page 5684)

Pack (see page 2885)

Clear (see page 2874)

Extract (see page 5681)

31.1.7.3.8 TObjectList.Last Method

Returns the object at the end of the Items (see page 5685) array.

Pascal

```
function Last: TObject;
```

C++

```
__fastcall TObject * Last();
```

Description

Call Last to retrieve the last object in the Items (see page 5685) array.

See Also

Items (see page 5685)

First (see page 5682)

31.1.7.3.9 TObjectList.Notify Method

Responds when items are added, removed, or extracted from the list.

Pascal

```
procedure Notify(Ptr: Pointer; Action: TListNotification); override;
```

C++

```
virtual __fastcall Notify(void * Ptr, TListNotification Action);
```

Description

Notify is called automatically when the items in the collection change.

Ptr points to item affected by the change.

Action indicates the change.

As implemented in TList, Notify does nothing. Descendant classes can override Notify in order to implement an appropriate response to the change notification.

31.1.7.3.10 TObjectList.Remove Method

Removes a specified object from the list and (if OwnsObjects (see page 5686) is true) frees the object.

Pascal

```
function Remove(AObject: TObject): Integer;
```

C++

```
__fastcall int Remove(TObject * AObject);
```

Description

Call Remove to delete a specific object from the list when its index is unknown. The value returned is the index of the object in the Items (see page 5685) array before it was removed. If the specified object is not found on the list, Remove returns –1. If OwnsObjects (see page 5686) is true, Remove frees the object in addition to removing it from the list.

After an object is deleted, all the objects that follow it are moved up in index position and Count is decremented. If an object appears more than once on the list, Remove deletes only the first appearance. Hence, if OwnsObjects (see page 5686) is true,

removing an object that appears more than once results in empty object references later in the list.

To use an index position (rather than an object reference) to specify the object to be removed, call Delete.

To remove an object from the list without freeing it, call Extract (see page 5681).

See Also

Add (see page 5680)

Count (see page 2896)

Delete (see page 2874)

Insert (see page 5683)

Items (see page 5685)

OwnsObjects (see page 5686)

Extract (see page 5681)

Pack (see page 2885)

31.1.7.3.11 TObjectList.SetItem Method

Pascal

```
procedure SetItem(Index: Integer; AObject: TObject);
```

C++

```
__fastcall SetItem(int Index, TObject * AObject);
```

Description

This is SetItem, a member of class TObjectList.

31.1.7.4 TObjectList Properties

The properties of the TObjectList class are listed here.

Public Properties

	Name	Description
	Items (see page 5685)	Lists the object references.
	OwnsObjects (see page 5686)	Allows TObjectList (see page 5677) to free objects when they are deleted from the list or the list is destroyed.

31.1.7.4.1 TObjectList.Items Property

Lists the object references.

Pascal

```
property Items [Index: Integer]: TObject;
```

C++

```
__property TObject * Items[int Index];
```

Description

Use Items to access objects in the list. Items is a zero-based array: The first object is indexed as 0, the second object is indexed as 1, and so forth. You can read or change the value at a specific index, or use Items with the Count property to iterate through

the list.

If OwnsObjects (see page 5686) is true, reassigning an Items index frees the object that previously occupied that position in the list.

Items can contain nil (Delphi) or NULL (C++) references. To remove nil (Delphi) or NULL (C++) references and reduce the size of the array, call the Pack method.

Note: In Delphi, Items is the default property of TObjectList (see page 5677). This means you can omit the property name. Thus, the following two lines of code are both acceptable and do the same thing:

Note: ObjectList1.Items[i] := Form1.FindChildControl(ListBox1.Items[i]);

Note: ObjectList1[i] := Form1.FindChildControl(ListBox1.Items[i]);

See Also

OwnsObjects (see page 5686)

Add (see page 5680)

Insert (see page 5683)

Delete (see page 2874)

Remove (see page 5684)

List (see page 2900)

Extract (see page 5681)

Count (see page 2896)

Clear (see page 2874)

Pack (see page 2885)

operator_sb

31.1.7.4.2 TObjectList.OwnsObjects Property

Allows TObjectList (see page 5677) to free objects when they are deleted from the list or the list is destroyed.

Pascal

```
property OwnsObjects: Boolean;
```

C++

```
__property Boolean OwnsObjects;
```

Description

OwnsObjects allows TObjectList (see page 5677) to control the memory of its objects. If OwnsObjects is true (the default),

calling Delete or Remove (see page 5684) frees the deleted object in addition to removing it from the list.

calling Clear frees all the objects in the list in addition to emptying the list.

calling the destructor frees all the objects in the list in addition to destroying the TObjectList (see page 5677) itself.

assigning a new value to an index in Items (see page 5685) frees the object that previously occupied that position in the list.

Even if OwnsObjects is true, the Extract (see page 5681) method can be used to remove objects from the list without freeing them.

See Also

Items (see page 5685)

Delete (see page 2874)

Clear (see page 2874)

Destroy (see page 2870)

Remove (see page 5684)

Extract (see page 5681)

31.1.8 TObjectQueue Class

TObjectQueue maintains a first-in first-out list of objects.

Class Hierarchy

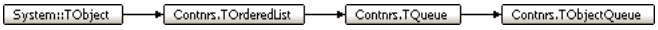

Pascal

```
TObjectQueue = class(TQueue);
```

C++

```
class TObjectQueue : public TQueue;
```

File

Contnrs

Description

Use a TObjectQueue to store and maintain a first-in first-out list of objects.

See Also

TStack (see page 5701)

TList (see page 2868)

TQueue (see page 5699)

TObjectStack (see page 5690)

TObjectList (see page 5677)

Working with Lists

31.1.8.1 TObjectQueue Members

The following tables list the members exposed by TObjectQueue.

Protected Methods

	Name	Description
	PeekItem (see page 5697)	Returns a pointer to the next item in the list.

V	PopItem (see page 5697)	Removes and returns a pointer to the next item in the list.
A	PushItem (see page 5698)	Adds an item to the list.
V	PushItem (see page 5701)	Adds an item to the list.

Protected Properties

	Name	Description
R	List (see page 5699)	Identifies the internal list used to store and manage the items in the ordered list.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AtLeast (see page 5696)	Checks the size of the list.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Count (see page 5696)	Returns the size of the list.
	Create (see page 5695)	Creates a TOrderedList (see page 5693) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5695)	Destroys the TOrderedList (see page 5693) instance.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Peek (see page 5689)	Returns a reference to the front of the queue.
	Pop (see page 5689)	Removes and returns the object at the front of the queue.
	Push (see page 5689)	Adds an object to the end of the queue.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

31.1.8.2 TObjectQueue Methods

The methods of the TObjectQueue class are listed here.

Public Methods

	Name	Description
	Peek (see page 5689)	Returns a reference to the front of the queue.
	Pop (see page 5689)	Removes and returns the object at the front of the queue.
	Push (see page 5689)	Adds an object to the end of the queue.

31.1.8.2.1 TObjectQueue.Peek Method

Returns a reference to the front of the queue.

Pascal

```
function Peek: TObject;
```

C++

```
__fastcall TObject * Peek();
```

Description

Call Peek to access the queue without removing an object. Peek returns a reference to the next object in the queue (that is, the earliest object added).

To remove an object from the queue, call Pop (see page 5689).

See Also

Pop (see page 5689)

Push (see page 5689)

31.1.8.2.2 TObjectQueue.Pop Method

Removes and returns the object at the front of the queue.

Pascal

```
function Pop: TObject;
```

C++

```
__fastcall TObject * Pop();
```

Description

Call Pop to retrieve an object from the queue. Pop removes the next object from the queue (that is, the earliest object added), then returns a reference to the deleted object.

To access the queue without removing an object, call Peek (see page 5689).

See Also

Push (see page 5689)

Peek (see page 5689)

31.1.8.2.3 TObjectQueue.Push Method

Adds an object to the end of the queue.

Pascal

```
function Push(AObject: TObject): TObject;
```

C++

```
__fastcall TObject * Push(TObject * AObject);
```

Description

Call Push to add an object to the queue.

AObject is the object to add to the queue.

Push returns a reference to the object that was just added (AObject).

See Also

Pop (see page 5689)

Peek (see page 5689)

31.1.9 TObjectStack Class

TObjectStack maintains a last-in first-out list of objects.

Class Hierarchy

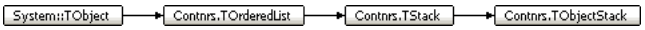

Pascal

```
TObjectStack = class(TStack);
```

C++

```
class TObjectStack : public TStack;
```

File

Contnrs

Description

Use TObjectStack to store and maintain a last-in first-out list of objects.

See Also

TStack (see page 5701)

TList (see page 2868)

TQueue (see page 5699)

TObjectQueue (see page 5687)

TObjectList (see page 5677)

Working with Lists

31.1.9.1 TObjectStack Members

The following tables list the members exposed by TObjectStack.

Protected Methods

	Name	Description
V	PeekItem (see page 5697)	Returns a pointer to the next item in the list.
V	PopItem (see page 5697)	Removes and returns a pointer to the next item in the list.
A	PushItem (see page 5698)	Adds an item to the list.
V	PushItem (see page 5703)	Adds an item to the list.

Protected Properties

	Name	Description
R	List (see page 5699)	Identifies the internal list used to store and manage the items in the ordered list.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AtLleast (see page 5696)	Checks the size of the list.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Count (see page 5696)	Returns the size of the list.
	Create (see page 5695)	Creates a TOrderedList (see page 5693) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5695)	Destroys the TOrderedList (see page 5693) instance.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Peek (see page 5692)	Returns a reference to the top of the stack.
	Pop (see page 5692)	Removes and returns the object at the top of the stack.
	Push (see page 5693)	Adds an object to the top of the stack.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

31.1.9.2 TObjectStack Methods

The methods of the TObjectStack class are listed here.

Public Methods

	Name	Description
	Peek (see page 5692)	Returns a reference to the top of the stack.
	Pop (see page 5692)	Removes and returns the object at the top of the stack.
	Push (see page 5693)	Adds an object to the top of the stack.

31.1.9.2.1 TObjectStack.Peek Method

Returns a reference to the top of the stack.

Pascal

```
function Peek: TObject;
```

C++

```
__fastcall TObject * Peek();
```

Description

Call Peek to access the stack without removing an object. Peek returns a reference to the next object in the stack (that is, the last object added).

To remove an object from the stack, call Pop (see page 5692).

See Also

Push (see page 5693)

Pop (see page 5692)

31.1.9.2.2 TObjectStack.Pop Method

Removes and returns the object at the top of the stack.

Pascal

```
function Pop: TObject;
```

C++

```
__fastcall TObject * Pop();
```

Description

Call Pop to retrieve an object from the stack. Pop removes the next object from the stack (that is, the last object added), then returns a reference to the deleted object.

To access the stack without removing an object, call Peek (see page 5692).

See Also

Push (see page 5693)

Peek (see page 5692)

31.1.9.2.3 TObjectStack.Push Method

Adds an object to the top of the stack.

Pascal

```
function Push(AObject: TObject): TObject;
```

C++

```
__fastcall TObject * Push(TObject * AObject);
```

Description

Call Push to add an object to the stack.

AObject is the object to add to the beginning of the stack.

Push returns a reference to the object that was just added (AObject).

See Also

Pop (see page 5692)

Peek (see page 5692)

31.1.10 TOrderedList Class

TOrderedList maintains an array of pointers that must be accessed in serial order.

Class Hierarchy

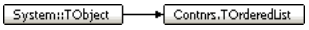

Pascal

```
TOrderedList = class(TObject);
```

C++

```
class TOrderedList : public TObject;
```

File

Contnrs

Description

TOrderedList is an abstract class that represents a list. Descendant classes like TStack (see page 5701) and TQueue (see page 5699) override the protected PushItem (see page 5698) method to determine the order in which to access items.

See Also

TList (see page 2868)

TQueue (see page 5699)

TStack (see page 5701)

TObjectStack (see page 5690)

TObjectQueue (see page 5687)

31.1.10.1 TOrderedList Members

The following tables list the members exposed by TOrderedList.

Protected Methods

	Name	Description
V	PeekItem (see page 5697)	Returns a pointer to the next item in the list.
V	PopItem (see page 5697)	Removes and returns a pointer to the next item in the list.
A	PushItem (see page 5698)	Adds an item to the list.

Protected Properties

	Name	Description
R	List (see page 5699)	Identifies the internal list used to store and manage the items in the ordered list.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AtLeast (see page 5696)	Checks the size of the list.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Count (see page 5696)	Returns the size of the list.
	Create (see page 5695)	Creates a TOrderedList (see page 5693) instance.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5695)	Destroys the TOrderedList (see page 5693) instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Peek (see page 5696)	Returns a pointer to the next item in the list.
	Pop (see page 5697)	Removes and returns the next item in the list.
	Push (see page 5698)	Adds an item to the list.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

31.1.10.2 TOrderedList.Create Constructor

Creates a TOrderedList (see page 5693) instance.

Pascal

```
constructor Create;
```

C++

```
__fastcall TOrderedList();
```

Description

Create generates a new list.

31.1.10.3 TOrderedList.Destroy Destructor

Destroys the TOrderedList (see page 5693) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TOrderedList();
```

Description

Do not call Destroy directly. Instead, call Free. Free verifies that the TOrderedList (see page 5693) reference is not nil, and only then calls Destroy.

Destroy frees the memory used to store the TOrderedList (see page 5693) instance. It does not free the memory pointed to by the items in the list.

See Also

Free (see page 22300)

31.1.10.4 TOrderedList Methods

The methods of the TOrderedList class are listed here.

Protected Methods

	Name	Description
V	PeekItem (see page 5697)	Returns a pointer to the next item in the list.
V	PopItem (see page 5697)	Removes and returns a pointer to the next item in the list.
A	PushItem (see page 5698)	Adds an item to the list.

Public Methods

	Name	Description
	AtLeast (see page 5696)	Checks the size of the list.
	Count (see page 5696)	Returns the size of the list.
	Peek (see page 5696)	Returns a pointer to the next item in the list.
	Pop (see page 5697)	Removes and returns the next item in the list.
	Push (see page 5698)	Adds an item to the list.

31.1.10.4.1 TOrderedList.AtLeast Method

Checks the size of the list.

Pascal

```
function AtLeast(ACount: Integer): Boolean;
```

C++

```
__fastcall Boolean AtLeast(int ACount);
```

Description

Call AtLeast to verify the size of the list. AtLeast returns true if the number of items in the list is greater than or equal to ACount.

See Also

Count (see page 5696)

31.1.10.4.2 TOrderedList.Count Method

Returns the size of the list.

Pascal

```
function Count: Integer;
```

C++

```
__fastcall int Count();
```

Description

Count returns the number of items in the list.

See Also

AtLeast (see page 5696)

31.1.10.4.3 TOrderedList.Peek Method

Returns a pointer to the next item in the list.

Pascal

```
function Peek: Pointer;
```

C++

```
__fastcall void * Peek();
```

Description

Call Peek to access the list without removing an item. Peek returns a pointer to the next item in the list.

Each TOrderedList (see page 5693) descendant implements Peek to return a specific item from the list. For example, TQueue (see page 5699) returns the item that was added first, while TStack (see page 5701) returns the item that was added last.

To remove an item from the list, call Pop (see page 5697).

See Also

Push (see page 5698)

Pop (see page 5697)

31.1.10.4.4 TOrderedList.PeekItem Method

Returns a pointer to the next item in the list.

Pascal

```
function PeekItem: Pointer; virtual;
```

C++

```
virtual __fastcall void * PeekItem();
```

Description

The protected PeekItem method provides the underlying implementation for the Peek (see page 5696) method. It returns the last item in the list specified by the List (see page 5699) property.

Override this method to return a different item in descendant classes that do not maintain the list in reverse output order.

See Also

Peek (see page 5696)

List (see page 5699)

31.1.10.4.5 TOrderedList.Pop Method

Removes and returns the next item in the list.

Pascal

```
function Pop: Pointer;
```

C++

```
__fastcall void * Pop();
```

Description

Call Pop to retrieve an item from the list. Pop removes the next item from the list, then returns a pointer to the deleted item.

Each TOrderedList (see page 5693) descendant implements Pop to remove and return a specific item from the list. For example, TQueue (see page 5699) returns the item that was added first, while TStack (see page 5701) returns the item that was added last.

To access the list without removing an item, call Peek (see page 5696).

See Also

Push (see page 5698)

Peek (see page 5696)

31.1.10.4.6 TOrderedList.PopItem Method

Removes and returns a pointer to the next item in the list.

Pascal

```
function PopItem: Pointer; virtual;
```

C++

```
virtual __fastcall void * PopItem();
```

Description

The protected PopItem method provides the underlying implementation for the Pop (see page 5697) method. It removes the last item from the list specified by the List (see page 5699) property and returns a pointer to the deleted item.

Override this method to remove and return a different item in descendant classes that do not maintain the list in reverse output order.

See Also

Pop (see page 5697)

List (see page 5699)

31.1.10.4.7 TOrderedList.Push Method

Adds an item to the list.

Pascal

```
function Push(AItem: Pointer): Pointer;
```

C++

```
__fastcall void * Push(void * AItem);
```

Description

Call Push to add an item to the list.

AItem is the item to add.

Push returns a reference to the item that was just added (AItem).

See Also

Pop (see page 5697)

Peek (see page 5696)

31.1.10.4.8 TOrderedList.PushItem Method

Adds an item to the list.

Pascal

```
procedure PushItem(AItem: Pointer); virtual; abstract;
```

C++

```
virtual __fastcall PushItem(void * AItem) = 0;
```

Description

The protected PushItem method provides the underlying implementation for the Push (see page 5698) method. In TOrderedList (see page 5693), PushItem has no implementation (it is abstract or, in C++ terminology, pure virtual).

Descendant classes override this method to add items to the list specified by the List (see page 5699) property. Each descendant class adds items so as to maintain the list in reverse output order. In TStack (see page 5701), items are added to the end of the list (so that the list is last-in first-out). In TQueue (see page 5699), items are added to the beginning (so that the list is first-in first-out).

See Also

Push (see page 5698)

Pop (see page 5697)

TStack (see page 5701)

TQueue (see page 5699)

List (see page 5699)

31.1.10.5 TOrderedList Properties

The properties of the TOrderedList class are listed here.

Protected Properties

	Name	Description
R	List (see page 5699)	Identifies the internal list used to store and manage the items in the ordered list.

31.1.10.5.1 TOrderedList.List Property

Identifies the internal list used to store and manage the items in the ordered list.

Pascal

```
property List: TList;
```

C++

```
__property TList List;
```

Description

Applications can't use the protected List property. It is an internal TList object owned by TOrderedList (see page 5693), which stores and manages the items in the ordered list.

31.1.11 TQueue Class

TQueue maintains a first-in first-out array of pointers.

Class Hierarchy

Pascal

```
TQueue = class(TOrderedList);
```

C++

```
class TQueue : public TOrderedList;
```

File

Contnrs

Description

Use a TQueue object to store and maintain a first-in first-out list.

See Also

TList (see page 2868)

TStack (see page 5701)

TObjectQueue (see page 5687)

Working with Lists

31.1.11.1 TQueue Members

The following tables list the members exposed by TQueue.

Protected Methods

	Name	Description
	PeekItem (see page 5697)	Returns a pointer to the next item in the list.
	PopItem (see page 5697)	Removes and returns a pointer to the next item in the list.
	PushItem (see page 5701)	Adds an item to the list.

Protected Properties

	Name	Description
	List (see page 5699)	Identifies the internal list used to store and manage the items in the ordered list.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AtLeast (see page 5696)	Checks the size of the list.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Count (see page 5696)	Returns the size of the list.
	Create (see page 5695)	Creates a TOrderedList (see page 5693) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 5695)	Destroys the TOrderedList (see page 5693) instance.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Peek (see page 5696)	Returns a pointer to the next item in the list.
	Pop (see page 5697)	Removes and returns the next item in the list.
	Push (see page 5698)	Adds an item to the list.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

31.1.11.2 TQueue Methods

The methods of the TQueue class are listed here.

Protected Methods

	Name	Description
	PushItem (see page 5701)	Adds an item to the list.

31.1.11.2.1 TQueue.PushItem Method

Adds an item to the list.

Pascal

```
procedure PushItem(AItem: Pointer); override;
```

C++

```
virtual __fastcall PushItem(void * AItem);
```

Description

The protected PushItem method provides the underlying implementation for the Push method. In TOrderedList, PushItem has no implementation (it is abstract or, in C++ terminology, pure virtual).

Descendant classes override this method to add items to the list specified by the List property. Each descendant class adds items so as to maintain the list in reverse output order. In TStack, items are added to the end of the list (so that the list is last-in first-out). In TQueue, items are added to the beginning (so that the list is first-in first-out).

See Also

Push (see page 5698)

Pop (see page 5697)

TStack (see page 5701)

TQueue (see page 5699)

List (see page 5699)

31.1.12 TStack Class

TStack maintains a last-in first-out array of pointers.

Class Hierarchy

Pascal

```
TStack = class(TOrderedList);
```

C++

```
class TStack : public TOrderedList;
```

File

Contnrs

Description

Use a TStack object to store and maintain a last-in first-out list.

See Also

TList (see page 2868)

TQueue (see page 5699)

TObjectStack (see page 5690)

Working with Lists

31.1.12.1 TStack Members

The following tables list the members exposed by TStack.

Protected Methods

	Name	Description
	PeekItem (see page 5697)	Returns a pointer to the next item in the list.
	PopItem (see page 5697)	Removes and returns a pointer to the next item in the list.
	PushItem (see page 5703)	Adds an item to the list.

Protected Properties

	Name	Description
	List (see page 5699)	Identifies the internal list used to store and manage the items in the ordered list.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AtLeast (see page 5696)	Checks the size of the list.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Count (see page 5696)	Returns the size of the list.
	Create (see page 5695)	Creates a TOrderedList (see page 5693) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 5695)	Destroys the TOrderedList (see page 5693) instance.
	Destroy (see page 22287)	Disposes of an object instance.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Peek (see page 5696)	Returns a pointer to the next item in the list.
	Pop (see page 5697)	Removes and returns the next item in the list.
	Push (see page 5698)	Adds an item to the list.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

31.1.12.2 TStack Methods

The methods of the TStack class are listed here.

Protected Methods

	Name	Description
	PushItem (see page 5703)	Adds an item to the list.

31.1.12.2.1 TStack.PushItem Method

Adds an item to the list.

Pascal

```
procedure PushItem(AItem: Pointer); override;
```

C++

```
virtual __fastcall PushItem(void * AItem);
```

Description

The protected PushItem method provides the underlying implementation for the Push method. In TOrderedList, PushItem has no implementation (it is abstract or, in C++ terminology, pure virtual).

Descendant classes override this method to add items to the list specified by the List property. Each descendant class adds items so as to maintain the list in reverse output order. In TStack, items are added to the end of the list (so that the list is last-in first-out). In TQueue, items are added to the beginning (so that the list is first-in first-out).

See Also

Push (see page 5698)

Pop (see page 5697)

TStack (see page 5701)

TQueue (see page 5699)

List (see page 5699)

31.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	RaiseListError (see page 5705)	This is function Contnrs.RaiseListError.

31.2.1 Contnrs.RaiseListError Function

Pascal

```
procedure RaiseListError(const ATemplate: string; const AData: array of const);
```

C++

```
RaiseListError(const AnsiString ATemplate, const array of const AData);
```

File

Contnrs

Description

This is function Contnrs.RaiseListError.

31.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TBucketListSizes (see page 5707)	TBucketListSizes indicates the number of distinct buckets in a TBucketList (see page 5643) object.

Records

	Name	Description
	TBucket (see page 5706)	TBucketArray (see page 5709) and TBucket represent a list of buckets that each store an array of items.
	TBucketItem (see page 5707)	TBucketItemArray (see page 5710) and TBucketItem represent the items in a TBucket (see page 5706) record (Delphi) or structure (C++).

31.3.1 Contnrs.TBucket Record

TBucketArray (see page 5709) and TBucket represent a list of buckets that each store an array of items.

Pascal

```
TBucket = record
  Count: Integer;
  Items: TBucketItemArray;
end;
```

C++

```
struct TBucket {
  int Count;
  TBucketItemArray Items;
};
```

File

Contnrs

Description

TBucketArray (see page 5709) is a dynamic array of TBucket vakyes. Each TBucket represents a dynamic array of items.

Count is the number of items in a bucket.

Items is the array of items that the bucket stores.

TBucketList (see page 5643) uses TBucketArray (see page 5709) to represent its collection of buckets, or categories, under which it stores items.

See Also

TBucketList

31.3.2 Contnrs.TBucketItem Record

TBucketItemArray (see page 5710) and TBucketItem represent the items in a TBucket (see page 5706) record (Delphi) or structure (C++).

Pascal

```
TBucketItem = record
  Data: Pointer;
  Item: Pointer;
end;
```

C++

```
struct TBucketItem {
  void * Data;
  void * Item;
};
```

File

Contnrs

Description

TBucketItemArray (see page 5710) is a dynamic array of TBucketItem values. Each TBucketItem stores an item and its associated data pointer.

Item points to the item in a bucket.

Data points to the data that is associated with Item.

See Also

TBucketArray

31.3.3 Contnrs.TBucketListSizes Enumeration

TBucketListSizes indicates the number of distinct buckets in a TBucketList (see page 5643) object.

Pascal

```
TBucketListSizes = (
  bl2,
  bl4,
  bl8,
  bl16,
  bl32,
  bl64,
  bl128,
  bl256
);
```

C++

```
enum TBucketListSizes {
  bl2,
  bl4,
  bl8,
  bl16,
  bl32,
```

```
    bl64,
    bl128,
    bl256
  };
```

File

Contnrs

Description

The following table lists the possible values:

Value	Number of buckets
bl2	2
bl4	4
bl8	8
bl16	16
bl32	32
bl64	64
bl128	128
bl256	256

31.4 Types

The following table lists types in this documentation.

Types

Name	Description
TBucketArray (see page 5709)	This is type Contnrs.TBucketArray.
TBucketEvent (see page 5709)	This is type Contnrs.TBucketEvent.
TBucketItemArray (see page 5710)	This is type Contnrs.TBucketItemArray.
TBucketProc (see page 5710)	This is type Contnrs.TBucketProc.

31.4.1 Contnrs.TBucketArray Type

Pascal

```
TBucketArray = array of TBucket;
```

C++

```
array of TBucket TBucketArray;
```

File

Contnrs

Description

This is type Contnrs.TBucketArray.

31.4.2 Contnrs.TBucketEvent Type

Pascal

```
TBucketEvent = procedure (AItem, AData: Pointer; out AContinue: Boolean) of object;
```

C++

```
(AItem, AData: Pointer; out AContinue: Boolean) ( TBucketEvent)();
```

File

Contnrs

Description

This is type Contnrs.TBucketEvent.

31.4.3 Contnrs.TBucketItemArray Type

Pascal

```
TBucketItemArray = array of TBucketItem;
```

C++

```
array of TBucketItem TBucketItemArray;
```

File

Contnrs

Description

This is type Contnrs.TBucketItemArray.

31.4.4 Contnrs.TBucketProc Type

Pascal

```
TBucketProc = procedure (AInfo, AItem, AData: Pointer; out AContinue: Boolean);
```

C++

```
(AInfo, AItem, AData: Pointer; out AContinue: Boolean) ( TBucketProc)();
```

File

Contnrs

Description

This is type Contnrs.TBucketProc.

32 Controls Namespace

This is namespace Controls.

Topics

Name	Description
Classes (see page 5719)	The following table lists classes in this documentation.
Interfaces (see page 6238)	The following table lists interfaces in this documentation.
Functions (see page 6246)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6267)	The following table lists structs, records, enums in this documentation.
Types (see page 6290)	The following table lists types in this documentation.
Variables (see page 6320)	The following table lists variables in this documentation.
Constants (see page 6323)	The following table lists constants in this documentation.

Classes

	Name	Description
	TBaseDragControlObject (see page 5719)	TBaseDragControlObject object provides support for drag-and-drop or drag-and-dock operations.
	TControl (see page 5724)	TControl is the base class for all components that are visible at runtime.
	TControlAction (see page 5877)	This is class Controls.TControlAction.
	TControlActionLink (see page 5892)	TControlActionLink links action objects to clients that descend from TControl (see page 5724).
	TControlCanvas (see page 5903)	TControlCanvas is a specialized canvas object associated with a control.
	TCustomControl (see page 5908)	Base class for controls that wrap Windows screen objects but perform their own rendering.
	TCustomControlAction (see page 5923)	This is class Controls.TCustomControlAction.
	TCustomListControl (see page 5928)	TCustomListControl is the base class for controls that display a list of items.
	TCustomMultiSelectListControl (see page 5944)	TCustomMultiSelectListControl is the base class for list controls that allow multiple items to be selected.
	TCustomPanningWindow (see page 5958)	This is an abstract base class that is used as an ancestor for users to implement a custom panning window.
	TCustomTransparentControl (see page 5970)	Base class for transparent controls.
	TDockTree (see page 5983)	TDockTree is the default docking manager for TWinControl (see page 6112).
	TDockZone (see page 5997)	This is class Controls.TDockZone.
	TDragControlObject (see page 6003)	TDragControlObject object provides for normal drag and drop operations.
	TDragControlObjectEx (see page 6007)	TDragControlObjectEx object provides for normal drag and drop operations.
	TDragDockObject (see page 6009)	Contains drag-and-dock information for a control.
	TDragDockObjectEx (see page 6018)	Contains drag-and-dock information for a control.

	TDragImageList (see page 6021)	TDragImageList represents a set of images that appear during drag operations.
	TDragObject (see page 6031)	TDragObject is a base class used for managing drag operations.
	TDragObjectEx (see page 6041)	TDragObjectEx is a base class used for managing drag operations.
	TGraphicControl (see page 6044)	TGraphicControl is the base class for all lightweight controls.
	THintWindow (see page 6055)	THintWindow implements the small pop-up window that appears over a control at runtime when the control has its ShowHint (see page 5840) property set to true.
	TImageList (see page 6078)	TImageList represents a collection of same-sized images, each of which can be referred to by its index.
	TMargins (see page 6087)	Contains margins for the control.
	TMouse (see page 6096)	TMouse surfaces information about the mouse and specifies how the application responds to mouse messages.
	TPadding (see page 6103)	Contains the padding for the control.
	TSizeConstraints (see page 6107)	Contains size constraints for a control.
	TWinControl (see page 6112)	TWinControl is the base class for all controls that are wrappers for Microsoft Windows screen objects.
	TWinControlActionLink (see page 6233)	TWinControlActionLink is an action link used by descendants of TWinControl (see page 6112).

Constants

Name	Description
AnchorAlign (see page 6325)	Defines the default sets of anchor types for all component alignment values.
CM_ACTIONEXECUTE (see page 6326)	This is constant Controls.CM_ACTIONEXECUTE.
CM_ACTIONUPDATE (see page 6327)	This is constant Controls.CM_ACTIONUPDATE.
CM_ACTIVATE (see page 6327)	This is constant Controls.CM_ACTIVATE.
CM_ALLCHILDRENFLIPPED (see page 6327)	This is constant Controls.CM_ALLCHILDRENFLIPPED.
CM_APPKEYDOWN (see page 6328)	This is constant Controls.CM_APPKEYDOWN.
CM_APPSYSCOMMAND (see page 6328)	This is constant Controls.CM_APPSYSCOMMAND.
CM_BASE (see page 6328)	This is constant Controls.CM_BASE.
CM_BIDIMODECHANGED (see page 6329)	This is constant Controls.CM_BIDIMODECHANGED.
CM_BORDERCHANGED (see page 6329)	This is constant Controls.CM_BORDERCHANGED.
CM_BUTTONPRESSED (see page 6329)	This is constant Controls.CM_BUTTONPRESSED.
CM_CANCELMODE (see page 6330)	This is constant Controls.CM_CANCELMODE.
CM_CHANGED (see page 6330)	This is constant Controls.CM_CHANGED.
CM_CHILDKEY (see page 6330)	This is constant Controls.CM_CHILDKEY.
CM_COLORCHANGED (see page 6331)	This is constant Controls.CM_COLORCHANGED.
CM_CONTROLCHANGE (see page 6331)	This is constant Controls.CM_CONTROLCHANGE.
CM_CONTROLLISTCHANGE (see page 6331)	This is constant Controls.CM_CONTROLLISTCHANGE.
CM_CONTROLLISTCHANGING (see page 6332)	This is constant Controls.CM_CONTROLLISTCHANGING.
CM_CREATEPOPUP (see page 6332)	This is constant Controls.CM_CREATEPOPUP.
CM_CTL3DCHANGED (see page 6332)	This is constant Controls.CM_CTL3DCHANGED.
CM_CTLCOLOREDITGLASS (see page 6333)	This is constant Controls.CM_CTLCOLOREDITGLASS.
CM_CURSORCHANGED (see page 6333)	This is constant Controls.CM_CURSORCHANGED.
CM_DEACTIVATE (see page 6333)	This is constant Controls.CM_DEACTIVATE.
CM_DESIGNHITTEST (see page 6334)	This is constant Controls.CM_DESIGNHITTEST.
CM_DESTROYHANDLE (see page 6334)	This is constant Controls.CM_DESTROYHANDLE.
CM_DIALOGCHAR (see page 6334)	This is constant Controls.CM_DIALOGCHAR.
CM_DIALOGHANDLE (see page 6335)	This is constant Controls.CM_DIALOGHANDLE.
CM_DIALOGKEY (see page 6335)	This is constant Controls.CM_DIALOGKEY.
CM_DOCKCLIENT (see page 6335)	This is constant Controls.CM_DOCKCLIENT.
CM_DOCKNOTIFICATION (see page 6336)	This is constant Controls.CM_DOCKNOTIFICATION.
CM_DOCWINDOWACTIVATE (see page 6336)	This is constant Controls.CM_DOCWINDOWACTIVATE.
CM_DRAG (see page 6336)	This is constant Controls.CM_DRAG.
CM_ENABLEDCHANGED (see page 6337)	This is constant Controls.CM_ENABLEDCHANGED.
CM_ENTER (see page 6337)	This is constant Controls.CM_ENTER.
CM_EXIT (see page 6337)	This is constant Controls.CM_EXIT.

CM_FLOAT (see page 6338)	This is constant Controls.CM_FLOAT.
CM_FOCUSCHANGED (see page 6338)	This is constant Controls.CM_FOCUSCHANGED.
CM_FONTCHANGE (see page 6338)	This is constant Controls.CM_FONTCHANGE.
CM_FONTCHANGED (see page 6339)	This is constant Controls.CM_FONTCHANGED.
CM_GETDATALINK (see page 6339)	This is constant Controls.CM_GETDATALINK.
CM_GOTFOCUS (see page 6339)	This is constant Controls.CM_GOTFOCUS.
CM_HINTSHOW (see page 6340)	This is constant Controls.CM_HINTSHOW.
CM_HINTSHOWPAUSE (see page 6340)	This is constant Controls.CM_HINTSHOWPAUSE.
CM_HITTEST (see page 6340)	This is constant Controls.CM_HITTEST.
CM_ICONCHANGED (see page 6341)	This is constant Controls.CM_ICONCHANGED.
CM_INVALIDATE (see page 6341)	This is constant Controls.CM_INVALIDATE.
CM_INVALIDATEDOCKHOST (see page 6341)	This is constant Controls.CM_INVALIDATEDOCKHOST.
CM_INVOKEHELP (see page 6342)	This is constant Controls.CM_INVOKEHELP.
CM_ISSHORTCUT (see page 6342)	This is constant Controls.CM_ISSHORTCUT.
CM_ISTOOLCONTROL (see page 6342)	This is constant Controls.CM_ISTOOLCONTROL.
CM_LOSTFOCUS (see page 6343)	This is constant Controls.CM_LOSTFOCUS.
CM_MENUCHANGED (see page 6343)	This is constant Controls.CM_MENUCHANGED.
CM_MOUSEACTIVATE (see page 6343)	This is constant Controls.CM_MOUSEACTIVATE.
CM_MOUSEENTER (see page 6344)	This is constant Controls.CM_MOUSEENTER.
CM_MOUSELEAVE (see page 6344)	This is constant Controls.CM_MOUSELEAVE.
CM_MOUSEWHEEL (see page 6344)	This is constant Controls.CM_MOUSEWHEEL.
CM_PARENTBIDIMODECHANGED (see page 6345)	This is constant Controls.CM_PARENTBIDIMODECHANGED.
CM_PARENTCOLORCHANGED (see page 6345)	This is constant Controls.CM_PARENTCOLORCHANGED.
CM_PARENTCTL3DCHANGED (see page 6345)	This is constant Controls.CM_PARENTCTL3DCHANGED.
CM_PARENTFONTCHANGED (see page 6346)	This is constant Controls.CM_PARENTFONTCHANGED.
CM_PARENTSHOWHINTCHANGED (see page 6346)	This is constant Controls.CM_PARENTSHOWHINTCHANGED.
CM_POPUPHWNDDESTROY (see page 6346)	This is constant Controls.CM_POPUPHWNDDESTROY.
CM_RECREATEWND (see page 6347)	This is constant Controls.CM_RECREATEWND.
CM_RELEASE (see page 6347)	This is constant Controls.CM_RELEASE.
CM_SETACTIVECONTROL (see page 6347)	This is constant Controls.CM_SETACTIVECONTROL.
CM_SHOWHINTCHANGED (see page 6348)	This is constant Controls.CM_SHOWHINTCHANGED.
CM_SHOWINGCHANGED (see page 6348)	This is constant Controls.CM_SHOWINGCHANGED.
CM_SYSCOLORCHANGE (see page 6348)	This is constant Controls.CM_SYSCOLORCHANGE.
CM_SYSFONTCHANGED (see page 6349)	This is constant Controls.CM_SYSFONTCHANGED.
CM_TABSTOPCHANGED (see page 6349)	This is constant Controls.CM_TABSTOPCHANGED.
CM_TEXTCHANGED (see page 6349)	This is constant Controls.CM_TEXTCHANGED.
CM_TIMECHANGE (see page 6350)	This is constant Controls.CM_TIMECHANGE.
CM_UIACTIVATE (see page 6350)	This is constant Controls.CM_UIACTIVATE.
CM_UIDEACTIVATE (see page 6350)	This is constant Controls.CM_UIDEACTIVATE.
CM_UNDOCKCLIENT (see page 6351)	This is constant Controls.CM_UNDOCKCLIENT.
CM_VISIBLECHANGED (see page 6351)	This is constant Controls.CM_VISIBLECHANGED.
CM_WANTSPECIALKEY (see page 6351)	This is constant Controls.CM_WANTSPECIALKEY.
CM_WINDOWHOOK (see page 6352)	This is constant Controls.CM_WINDOWHOOK.
CM_WININICHANGE (see page 6352)	This is constant Controls.CM_WININICHANGE.
CN_BASE (see page 6352)	This is constant Controls.CN_BASE.
CN_CHAR (see page 6353)	This is constant Controls.CN_CHAR.
CN_CHARTOITEM (see page 6353)	This is constant Controls.CN_CHARTOITEM.
CN_COMMAND (see page 6353)	This is constant Controls.CN_COMMAND.
CN_COMPAREITEM (see page 6354)	This is constant Controls.CN_COMPAREITEM.
CN_CTLCOLORBTN (see page 6354)	This is constant Controls.CN_CTLCOLORBTN.
CN_CTLCOLORDLG (see page 6354)	This is constant Controls.CN_CTLCOLORDLG.
CN_CTLCOLOREDIT (see page 6355)	This is constant Controls.CN_CTLCOLOREDIT.
CN_CTLCOLORLISTBOX (see page 6355)	This is constant Controls.CN_CTLCOLORLISTBOX.
CN_CTLCOLORMSGBOX (see page 6355)	This is constant Controls.CN_CTLCOLORMSGBOX.

CN_CTLCOLORSCROLLBAR (see page 6356)	This is constant Controls.CN_CTLCOLORSCROLLBAR.
CN_CTLCOLORSTATIC (see page 6356)	This is constant Controls.CN_CTLCOLORSTATIC.
CN_DELETEITEM (see page 6356)	This is constant Controls.CN_DELETEITEM.
CN_DRAWITEM (see page 6357)	This is constant Controls.CN_DRAWITEM.
CN_HSCROLL (see page 6357)	This is constant Controls.CN_HSCROLL.
CN_KEYDOWN (see page 6357)	This is constant Controls.CN_KEYDOWN.
CN_KEYUP (see page 6358)	This is constant Controls.CN_KEYUP.
CN_MEASUREITEM (see page 6358)	This is constant Controls.CN_MEASUREITEM.
CN_NOTIFY (see page 6358)	This is constant Controls.CN_NOTIFY.
CN_PARENTNOTIFY (see page 6359)	This is constant Controls.CN_PARENTNOTIFY.
CN_SYSCHAR (see page 6359)	This is constant Controls.CN_SYSCHAR.
CN_SYSKEYDOWN (see page 6359)	This is constant Controls.CN_SYSKEYDOWN.
CN_VKEYTOITEM (see page 6360)	This is constant Controls.CN_VKEYTOITEM.
CN_VSCROLL (see page 6360)	This is constant Controls.CN_VSCROLL.
crAppStart (see page 6360)	Identifies the default cursor.
crArrow (see page 6361)	Identifies an arrow cursor.
crCross (see page 6361)	Identifies an cross cursor.
crDefault (see page 6362)	Identifies the default cursor.
crDrag (see page 6362)	Identifies a drag cursor.
crHandPoint (see page 6363)	Identifies a pointing hand cursor.
crHelp (see page 6363)	Identifies a help cursor.
crHourGlass (see page 6364)	Identifies an hour glass cursor.
crHSplit (see page 6364)	Identifies a horizontal split cursor.
crIBeam (see page 6365)	Identifies a text insert cursor.
crMultiDrag (see page 6365)	Identifies a multiple item drag cursor.
crNo (see page 6366)	Identifies a not allowed cursor.
crNoDrop (see page 6366)	Identifies a not allowed to drop cursor.
crNone (see page 6367)	Identifies an invisible cursor.
crSize (see page 6367)	Identifies an obsolete cursor.
crSizeAll (see page 6368)	Identifies a size in all directions cursor.
crSizeNESW (see page 6368)	Identifies a North-East to South-West sizing cursor.
crSizeNS (see page 6369)	Identifies a vertical sizing cursor.
crSizeNWSE (see page 6369)	Identifies a North-West to South-East sizing cursor.
crSizeWE (see page 6370)	Identifies a horizontal sizing cursor.
crSQLWait (see page 6370)	Identifies an SQL wait hour glass cursor.
crUpArrow (see page 6371)	Identifies an upwards pointing arrow cursor.
crVSplit (see page 6371)	Identifies a vertical split cursor.
CTL3D_ALL (see page 6372)	This is constant Controls.CTL3D_ALL.
mrAbort (see page 6372)	Indicates that Abort was selected in a modal dialog.
mrAll (see page 6372)	This is constant Controls.mrAll.
mrCancel (see page 6373)	Indicates that Cancel was selected in a modal dialog.
mrIgnore (see page 6373)	Indicates that Ignore was selected in a modal dialog.
mrNo (see page 6374)	Indicates that No was selected in a modal dialog.
mrNone (see page 6374)	The default modal dialog return value.
mrNoToAll (see page 6375)	Indicates that 'No To All' was selected in a modal dialog.
mrOk (see page 6375)	This is constant Controls.mrOk.
mrRetry (see page 6375)	Indicates that Retry was selected in a modal dialog.
mrYes (see page 6376)	Indicates that Yes was selected in a modal dialog.
mrYesToAll (see page 6376)	Indicates that 'Yes To All' was selected in a modal dialog.
NullDockSite (see page 6377)	This is constant Controls.NullDockSite.

Functions

	Name	Description
	CancelDrag (see page 6247)	Cancels a drag-and-drop or drag-and-dock operation that is in progress.

	ChangeBiDiModeAlignment (see page 6247)	This is function Controls.ChangeBiDiModeAlignment.
	CursorToIdent (see page 6247)	Returns the name of the predefined constant that corresponds to a cursor.
	CursorToString (see page 6248)	Returns the AnsiString representation of a cursor.
	DragDone (see page 6249)	This is function Controls.DragDone.
	FindControl (see page 6249)	Returns the windowed control (Windows) or widget (cross-platform) whose window is identified by the specifiedhandle or which appears at a specified position.
	FindDragTarget (see page 6249)	Returns the control at a specified screen coordinate.
	FindVCLWindow (see page 6250)	Returns the windowed control, if any, at the specified location.
	GetCaptureControl (see page 6251)	Returns the control, if any, that currently receives all mouse messages.
	GetCursorValues (see page 6251)	Passes the name of every predefined cursor constant to a callback.
	GetLongHint (see page 6252)	Returns the second part of a two-part hint string.
	GetShortHint (see page 6253)	Returns the first part of a two-part hint string.
	IdentToCursor (see page 6254)	Translates a predefined cursor constant into a cursor.
	Imm32GetCompositionString (see page 6254)	This is function Controls.Imm32GetCompositionString.
	Imm32GetContext (see page 6255)	This is function Controls.Imm32GetContext.
	Imm32GetConversionStatus (see page 6255)	This is function Controls.Imm32GetConversionStatus.
	Imm32IsIME (see page 6255)	This is function Controls.Imm32IsIME.
	Imm32NotifyIME (see page 6256)	This is function Controls.Imm32NotifyIME.
	Imm32ReleaseContext (see page 6256)	This is function Controls.Imm32ReleaseContext.
	Imm32SetCompositionFont (see page 6256)	This is function Controls.Imm32SetCompositionFont.
	Imm32SetCompositionWindow (see page 6257)	This is function Controls.Imm32SetCompositionWindow.
	Imm32SetConversionStatus (see page 6257)	This is function Controls.Imm32SetConversionStatus.
	Imm32SetOpenStatus (see page 6257)	This is function Controls.Imm32SetOpenStatus.
	InitWndProc (see page 6258)	This is function Controls.InitWndProc.
	IsAbortResult (see page 6258)	Checks the return value from a modal form dialog and indicates whether the user selected Abort or Cancel.
	IsAnAllResult (see page 6259)	Checks the return value from a modal form dialog and indicates whether the user selected All, Yes to All, or No to All.
	IsDragObject (see page 6259)	Indicates whether a specified object is a TDragObject (see page 6031) descendant.
	IsNegativeResult (see page 6260)	Checks the return value from a modal form dialog and indicates whether the user selected No or No to All.
	IsPositiveResult (see page 6260)	Checks the return value from a modal form dialog and indicates whether the user selected Ok, Yes, Yes to All, or All.
	IsVCLControl (see page 6261)	IsVCLControl returns true if a control is a VCL control.
	MoveWindowOrg (see page 6262)	Changes the location of the point (0,0) within a device context.
	PerformEraseBackground (see page 6262)	This is function Controls.PerformEraseBackground.
	SendAppMessage (see page 6262)	Sends a Windows message to the application.
	SetCaptureControl (see page 6263)	Causes the specified control to receive all mouse messages.
	SetImeMode (see page 6264)	This is function Controls.SetImeMode.
	SetImeName (see page 6264)	This is function Controls.SetImeName.
	StringToCursor (see page 6264)	Returns a TCursor (see page 6305) value given its string representation.
	StripAllFromResult (see page 6265)	Converts a TModalResult (see page 6312) value from a constant that refers to "all" to the corresponding simple constant.
	Win32NLSEnableIME (see page 6266)	This is function Controls.Win32NLSEnableIME.

Interfaces

	Name	Description
	IDockManager (see page 6238)	IDockManager is the interface for docking managers.

Structs, Records, Enums

	Name	Description
	TAlign (see page 6268)	TAlign specifies the alignment of a control within its parent.
	TAlignInfo (see page 6269)	This is record Controls.TAlignInfo.
	TAnchorKind (see page 6269)	TAnchorKind and TAnchors (see page 6294) specify how a control is anchored to its parent.

	TBevelCut (see page 6270)	TBevelCut specifies a bevel's cut.
	TBevelEdge (see page 6270)	TBevelEdge and TBevelEdges (see page 6295) specify which edges of a window are beveled.
	TBevelKind (see page 6271)	TBevelKind specifies the kind of bevel that appears on a control.
	TCMCancelMode (see page 6272)	This is record Controls.TCMCancelMode.
	TCMChanged (see page 6272)	This is record Controls.TCMChanged.
	TCMChildKey (see page 6273)	This is record Controls.TCMChildKey.
	TCMControlChange (see page 6273)	This is record Controls.TCMControlChange.
	TCMControlListChange (see page 6274)	This is record Controls.TCMControlListChange.
	TCMControlListChanging (see page 6274)	This is record Controls.TCMControlListChanging.
	TCMCreatePopup (see page 6275)	This is record Controls.TCMCreatePopup.
	TCMDockClient (see page 6275)	This is record Controls.TCMDockClient.
	TCMDockNotification (see page 6276)	This is record Controls.TCMDockNotification.
	TCMDrag (see page 6276)	This is record Controls.TCMDrag.
	TCMFloat (see page 6277)	This is record Controls.TCMFloat.
	TCMFocusChanged (see page 6277)	This is record Controls.TCMFocusChanged.
	TCMMouseActivate (see page 6278)	This is record Controls.TCMMouseActivate.
	TCMMouseWheel (see page 6278)	This is record Controls.TCMMouseWheel.
	TCMPopupHWndDestroy (see page 6279)	This is record Controls.TCMPopupHWndDestroy.
	TCMUnDockClient (see page 6279)	This is record Controls.TCMUnDockClient.
	TControlListItem (see page 6280)	This is record Controls.TControlListItem.
	TCreateParams (see page 6280)	TCreateParams is the window-creation parameter record used to specify the type of window to create for a windowed control.
	TDockNotifyRec (see page 6281)	TDockNotifyRec represents the information from a Windows docking notification message, and PDockNotifyRec (see page 6291) is a pointer to a TDockNotifyRec value.
	TDockOrientation (see page 6282)	TDockOrientation specifies the orientation of controls in a docking zone.
	TDragKind (see page 6282)	TDragKind indicates the type of operation that occurs when the user drags a control.
	TDragMessage (see page 6283)	This is record Controls.TDragMessage.
	TDragMode (see page 6283)	TDragMode indicates how a control initiates drag operations.
	TDragRec (see page 6284)	This is record Controls.TDragRec.
	TDragState (see page 6284)	TDragState is used to specify how the mouse is moving in relation to a control.
	TImeMode (see page 6285)	TImeMode represents the current mode of an Input Method Editor (IME).
	TMouseActivate (see page 6287)	TMouseActivate defines values for the OnMouseActivate event.
	TMouseActivateRec (see page 6288)	This is record Controls.TMouseActivateRec.
	TMouseButton (see page 6288)	TMouseButton identifies a mouse button in a mouse-event handler.
	TPopupFormInfo (see page 6289)	This is record Controls.TPopupFormInfo.

Types

Name	Description
PControlListItem (see page 6291)	This is type Controls.PControlListItem.
PDockNotifyRec (see page 6291)	This is type Controls.PDockNotifyRec.
PDragRec (see page 6292)	This is type Controls.PDragRec.
PMouseActivateRec (see page 6292)	This is type Controls.PMouseActivateRec.
PPopupFormInfo (see page 6292)	This is type Controls.PPopupFormInfo.
TAlignInsertBeforeEvent (see page 6293)	TAlignInsertBeforeEvent is the function type for an OnAlignInsertBefore event.
TAlignPositionEvent (see page 6293)	TAlignPositionEvent is the function type for an OnAlignPosition event.
TAlignSet (see page 6294)	TAlignSet specifies the alignment of a child control within its parent.
TAnchors (see page 6294)	TBorderWidth (see page 6296) defines the distance, in pixels, between the outer and inner bevels of a control.
TAnimateWindowProc (see page 6295)	This is type Controls.TAnimateWindowProc.
TBevelEdges (see page 6295)	TBevelEdge (see page 6270) and TBevelEdges specify which edges of a window are beveled.
TBevelWidth (see page 6296)	This is type Controls.TBevelWidth.
TBorderWidth (see page 6296)	This is type Controls.TBorderWidth.
TCanResizeEvent (see page 6296)	TCanResizeEvent is the type of the OnCanResize event handler.

TCaption (see page 6297)	TCaption is the type of the Caption property.
TCMActivate (see page 6297)	This is type Controls.TCMActivate.
TCMDeactivate (see page 6297)	This is type Controls.TCMDeactivate.
TCMDesignHitTest (see page 6298)	This is type Controls.TCMDesignHitTest.
TCMDialogChar (see page 6298)	This is type Controls.TCMDialogChar.
TCMDialogKey (see page 6298)	This is type Controls.TCMDialogKey.
TCMEnter (see page 6299)	This is type Controls.TCMEnter.
TCMExit (see page 6299)	This is type Controls.TCMExit.
TCMGotFocus (see page 6299)	This is type Controls.TCMGotFocus.
TCMHitTest (see page 6300)	This is type Controls.TCMHitTest.
TCMLostFocus (see page 6300)	This is type Controls.TCMLostFocus.
TCMWantSpecialKey (see page 6300)	This is type Controls.TCMWantSpecialKey.
TConstrainedResizeEvent (see page 6301)	This is type Controls.TConstrainedResizeEvent.
TConstraintSize (see page 6301)	This is type Controls.TConstraintSize.
TContextPopupEvent (see page 6301)	This is type Controls.TContextPopupEvent.
TControlActionLinkClass (see page 6302)	TControlActionLinkClass defines the metaclass for TControlActionLink (see page 5892).
TControlClass (see page 6302)	TControlClass defines the metaclass for TControl (see page 5724).
TControlState (see page 6302)	TControlState describes the current state of a control.
TControlStyle (see page 6303)	The TControlStyle type describes the current style of the control.
TCursor (see page 6305)	TCursor identifies the cursor type.
TDate (see page 6306)	TDate is used to represent a date.
TDockDropEvent (see page 6307)	TDockDropEvent is the type of the OnDockDrop event handler.
TDockOverEvent (see page 6307)	TDockOverEvent is the type of the OnDockOver event handler.
TDockTreeClass (see page 6308)	TDockTreeClass defines the metaclass for TDockTree (see page 5983).
TDragDropEvent (see page 6308)	This is type Controls.TDragDropEvent.
TDragObjectClass (see page 6308)	TDragObjectClass defines the metaclass for TDragObject (see page 6031).
TDragOverEvent (see page 6309)	This is type Controls.TDragOverEvent.
TEndDragEvent (see page 6309)	This is type Controls.TEndDragEvent.
TForEachZoneProc (see page 6309)	This is type Controls.TForEachZoneProc.
TGetSiteInfoEvent (see page 6310)	TGetSiteInfoEvent is the type of the OnGetSiteInfo event handler.
THintWindowClass (see page 6310)	THintWindowClass defines the metaclass for THintWindow (see page 6055).
TImeName (see page 6311)	TImeName is the type that represents the name of an Input Method Editor.
TKeyEvent (see page 6311)	This is type Controls.TKeyEvent.
TKeyPressEvent (see page 6311)	This is type Controls.TKeyPressEvent.
TMarginSize (see page 6312)	This is type Controls.TMarginSize.
TModalResult (see page 6312)	TModalResult represents the return value from a modal dialog.
TMouseActivateEvent (see page 6313)	This is type Controls.TMouseActivateEvent.
TMouseEvent (see page 6313)	This is type Controls.TMouseEvent.
TMouseMoveEvent (see page 6314)	This is type Controls.TMouseMoveEvent.
TMouseWheelEvent (see page 6314)	This is type Controls.TMouseWheelEvent.
TMouseWheelUpDownEvent (see page 6314)	This is type Controls.TMouseWheelUpDownEvent.
TPanningWindowClass (see page 6315)	TPanningWindowClass defines the metaclass for TPanningWindow.
TScalingFlags (see page 6315)	TScalingFlags describes the way controls are scaled.
TStartDockEvent (see page 6316)	TStartDockEvent is the type of the OnStartDock event handler.
TStartDragEvent (see page 6316)	This is type Controls.TStartDragEvent.
TTabOrder (see page 6317)	TTabOrder defines the tab order of a control.
TTime (see page 6317)	TTime is used to represent a time.
TUnDockEvent (see page 6317)	TUnDockEvent is the type of the OnUnDock event handler.
TWinControlActionLinkClass (see page 6318)	TWinControlActionLinkClass defines the metaclass for TWinControlActionLink (see page 6233).
TWinControlClass (see page 6318)	TWinControlClass defines the metaclass for TWinControl (see page 6112).
TWndMethod (see page 6319)	TWndMethod is the type for Windows message procedures.

Variables

Name	Description
AnimateWindowProc (see page 6320)	This is variable Controls.AnimateWindowProc.
CreationControl (see page 6320)	This is variable Controls.CreationControl.
DefaultDockTreeClass (see page 6321)	Provides the default implementation class for the DockManager property of container controls.
IsVCLControlHook (see page 6321)	This is variable Controls.IsVCLControlHook.
Mouse (see page 6321)	Contains information about the system's mouse.
NewStyleControls (see page 6322)	Determines whether controls are drawn using the Windows 3.x "look".

32.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TBaseDragControlObject (see page 5719)	TBaseDragControlObject object provides support for drag-and-drop or drag-and-dock operations.
	TControl (see page 5724)	TControl is the base class for all components that are visible at runtime.
	TControlAction (see page 5877)	This is class Controls.TControlAction.
	TControlActionLink (see page 5892)	TControlActionLink links action objects to clients that descend from TControl (see page 5724).
	TControlCanvas (see page 5903)	TControlCanvas is a specialized canvas object associated with a control.
	TCustomControl (see page 5908)	Base class for controls that wrap Windows screen objects but perform their own rendering.
	TCustomControlAction (see page 5923)	This is class Controls.TCustomControlAction.
	TCustomListControl (see page 5928)	TCustomListControl is the base class for controls that display a list of items.
	TCustomMultiSelectListControl (see page 5944)	TCustomMultiSelectListControl is the base class for list controls that allow multiple items to be selected.
	TCustomPanningWindow (see page 5958)	This is an abstract base class that is used as an ancestor for users to implement a custom panning window.
	TCustomTransparentControl (see page 5970)	Base class for transparent controls.
	TDockTree (see page 5983)	TDockTree is the default docking manager for TWinControl (see page 6112).
	TDockZone (see page 5997)	This is class Controls.TDockZone.
	TDragControlObject (see page 6003)	TDragControlObject object provides for normal drag and drop operations.
	TDragControlObjectEx (see page 6007)	TDragControlObjectEx object provides for normal drag and drop operations.
	TDragDockObject (see page 6009)	Contains drag-and-dock information for a control.
	TDragDockObjectEx (see page 6018)	Contains drag-and-dock information for a control.
	TDragImageList (see page 6021)	TDragImageList represents a set of images that appear during drag operations.
	TDragObject (see page 6031)	TDragObject is a base class used for managing drag operations.
	TDragObjectEx (see page 6041)	TDragObjectEx is a base class used for managing drag operations.
	TGraphicControl (see page 6044)	TGraphicControl is the base class for all lightweight controls.
	THintWindow (see page 6055)	THintWindow implements the small pop-up window that appears over a control at runtime when the control has its ShowHint (see page 5840) property set to true.
	TImageList (see page 6078)	TImageList represents a collection of same-sized images, each of which can be referred to by its index.
	TMargins (see page 6087)	Contains margins for the control.
	TMouse (see page 6096)	TMouse surfaces information about the mouse and specifies how the application responds to mouse messages.
	TPadding (see page 6103)	Contains the padding for the control.
	TSizeConstraints (see page 6107)	Contains size constraints for a control.
	TWinControl (see page 6112)	TWinControl is the base class for all controls that are wrappers for Microsoft Windows screen objects.
	TWinControlActionLink (see page 6233)	TWinControlActionLink is an action link used by descendants of TWinControl (see page 6112).

32.1.1 TBaseDragControlObject Class

TBaseDragControlObject object provides support for drag-and-drop or drag-and-dock operations.

Class Hierarchy

System::TObject → Controls.TDragObject → Controls.TBaseDragControlObject

Pascal

```
TBaseDragControlObject = class(TDragObject);
```

C++

```
class TBaseDragControlObject : public TDragObject;
```

File

Controls

Description

Do not create instances of TBaseDragControlObject. TBaseDragControlObject, like its immediate ancestor TDragObject (see page 6031), handles operations common to both drag-and-drop and drag-and-dock behavior. TBaseDragControlObject adds support for associating a control with the drag object.

TBaseDragControlObject has two descendants: TDragControlObject (see page 6003), which is specialized for drag-and-drop operations, and TDragDockObject (see page 6009), which is specialized for drag-and-dock operations.

See Also

TDragDockObject (see page 6009)

32.1.1.1 TBaseDragControlObject Members

The following tables list the members exposed by TBaseDragControlObject.

Protected Methods

	Name	Description
	EndDrag (see page 5723)	Generates an OnEndDrag event for the control that is being dragged.
	Finished (see page 5723)	Finished ends a drag and drop operation.
	GetDragCursor (see page 6035)	Indicates which cursor should be displayed in response to mouse moves.
	GetDragImages (see page 6035)	Returns an image list containing the image to be displayed while dragging.
	MainWndProc (see page 6037)	This is MainWndProc, a member of class TDragObject.
	WndProc (see page 6038)	This is WndProc, a member of class TDragObject.

Public Methods

	Name	Description
	AfterConstruction (see page 6033)	Responds after the last constructor has executed.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 5722)	Copies the properties of another drag control object.
	BeforeDestruction (see page 6034)	Responds before the first destructor executes.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5721)	Creates an instance of the TBaseDragControlObject (see page 5719) and stores a reference to the control that is being dragged.

V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetName (see page 6036)	Returns a string that identifies the drag operation.
V	HideDragImage (see page 6036)	Provides the interface for a method that hides the current drag image.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Instance (see page 6037)	Returns the instance (module) handle of the EXE or DLL that created the drag object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	ShowDragImage (see page 6037)	Provides the interface for a method that hides the current drag image.

Public Properties

	Name	Description
	AlwaysShowDragImages (see page 6038)	This is AlwaysShowDragImages, a member of class TDragObject.
	Cancelling (see page 6038)	Specifies whether the drag operation is being canceled.
	Control (see page 5724)	Indicates the control that is being dragged.
	DragHandle (see page 6039)	Specifies the window handle of the dragged object.
	DragPos (see page 6039)	Specifies the position of the dragged object.
	DragTarget (see page 6039)	Specifies the dragged object's target.
	DragTargetPos (see page 6040)	Specifies the position of the drag object relative to the drag target.
R	Dropped (see page 6040)	Drag ended with drop.
R	MouseDeltaX (see page 6040)	Specifies the x-coordinate of the mouse position relative to the position of the dragged object's target.
R	MouseDeltaY (see page 6041)	Specifies the y-coordinate of the mouse position relative to the position of the dragged object's target.
	RightClickCancels (see page 6041)	This is RightClickCancels, a member of class TDragObject.

32.1.1.2 TBaseDragControlObject.Create Constructor

Creates an instance of the TBaseDragControlObject (see page 5719) and stores a reference to the control that is being dragged.

Pascal

```
constructor Create(AControl: TControl); virtual;
```

C++

```
virtual __fastcall TBaseDragControlObject(TControl * AControl);
```

Description

Call Create to instantiate a specialized descendant of TBaseDragControlObject (see page 5719) from the OnStartDrag event

handler of a control that can be dragged. Pass the control that is about to be dragged as the AControl parameter. This control becomes the value of the Control (see page 5724) property.

See Also

Control (see page 5724)

OnStartDrag (see page 5871)

OnStartDock (see page 5871)

32.1.1.3 TBaseDragControlObject Methods

The methods of the TBaseDragControlObject class are listed here.

Protected Methods

	Name	Description
	EndDrag (see page 5723)	Generates an OnEndDrag event for the control that is being dragged.
	Finished (see page 5723)	Finished ends a drag and drop operation.

Public Methods

	Name	Description
	Assign (see page 5722)	Copies the properties of another drag control object.

32.1.1.3.1 TBaseDragControlObject.Assign Method

Copies the properties of another drag control object.

Pascal

```
procedure Assign(Source: TDragObject); override;
```

C++

```
virtual __fastcall Assign(TDragObject Source);
```

Description

Call Assign to copy the properties of another drag object. Assign changes the DragTarget (see page 6039), DragHandle (see page 6039), DragPos (see page 6039), DragTargetPos (see page 6040), MouseDeltaX (see page 6040) and MouseDeltaY (see page 6041) properties to match a drag object supplied as the Source parameter. If Source is a descendant of TBaseDragControlObject (see page 5719) (normally the case), assign copies the Control (see page 5724) property as well.

See Also

Control (see page 5724)

DragTarget (see page 6039)

DragHandle (see page 6039)

DragPos (see page 6039)

DragTargetPos (see page 6040)

MouseDeltaX (see page 6040)

MouseDeltaY (see page 6041)

Assign (see page 2930)

32.1.1.3.2 TBaseDragControlObject.EndDrag Method

Generates an OnEndDrag event for the control that is being dragged.

Pascal

```
procedure EndDrag(Target: TObject; X: Integer; Y: Integer); virtual;
```

C++

```
virtual __fastcall EndDrag(TObject * Target, int X, int Y);
```

Description

EndDrag is called automatically when the user drops the control or cancels the drag operation. It calls the DoEndDrag method of the control specified by the Control (see page 5724) property. By default, this method generates an OnEndDrag event for that control.

The Target parameter indicates the object that accepted the control. If the drag operation was canceled or the control was dragged to a floating position (in a drag-and-dock operation), Target is nil (Delphi) or NULL (C++). The X and Y parameters indicate where the control is dropped.

See Also

Control (see page 5724)

OnEndDrag (see page 5860)

DoEndDrag (see page 5751)

32.1.1.3.3 TBaseDragControlObject.Finished Method

Finished ends a drag and drop operation.

Pascal

```
procedure Finished(Target: TObject; X: Integer; Y: Integer; Accepted: Boolean); override;
```

C++

```
virtual __fastcall Finished(TObject * Target, int X, int Y, Boolean Accepted);
```

Description

Finished is called automatically when the user drops the control or cancels the drag operation. Accepted indicates whether the Target control at X, Y position accepts the drag and drop or drag and dock operation. If a target does not accept the dragged object, its DragCanceled method is called before the EndDrag (see page 5723) method generates an OnEndDrag event.

See Also

DragCanceled (see page 5754)

EndDrag (see page 5723)

32.1.1.4 TBaseDragControlObject Properties

The properties of the TBaseDragControlObject class are listed here.

Public Properties

	Name	Description
	Control (see page 5724)	Indicates the control that is being dragged.

32.1.1.4.1 TBaseDragControlObject.Control Property

Indicates the control that is being dragged.

Pascal

```
property Control: TControl;
```

C++

```
__property TControl * Control;
```

Description

Use Control to gain access to the object that is being dragged. Depending on the operation, its properties can affect the appearance of the drag cursor and whether a drag target accepts the control.

See Also

Create (see page 5721)

32.1.2 TControl Class

TControl is the base class for all components that are visible at runtime.

Class Hierarchy

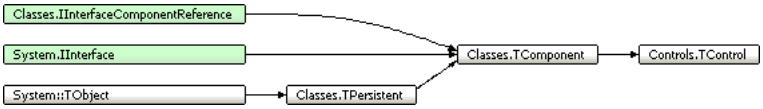

Pascal

```
TControl = class(TComponent);
```

C++

```
class TControl : public TComponent;
```

File

Controls

Description

Controls (see page 5711) are visual components, meaning the user can see them and possibly interact with them at runtime. All controls have properties, methods, and events that describe aspects of their appearance, such as the position of the control, the cursor or hint associated with the control, methods to paint or move the control, and events that respond to user actions.

TControl has many protected properties and methods that are used or published by its descendants.

See Also

TWinControl (see page 6112)

Controlling How Child Controls Are Docked

Controlling How Child Controls Are Undocked

Controlling How Child Controls Respond to Drag-and-dock Operations

Implementing Drag and Dock in Controls

Making a Control a Dockable Child

Making a Windowed Control a Docking Site

TControl Branch

VCL Overview

32.1.2.1 TControl Members

The following tables list the members exposed by TControl.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.

	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.

	Create (see page 5731)	Creates an instance of TControl (see page 5724).
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.

	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.

R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

32.1.2.2 TControl.Create Constructor

Creates an instance of TControl (see page 5724).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TControl(TComponent * AOwner);
```

Description

Calling Create constructs and initializes an instance of TControl (see page 5724). However, you should never attempt to instantiate a TControl (see page 5724). This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

Create calls the parent constructor and initializes the control.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a control's constructor allocates resources or memory, also override the destructor to free those resources.

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Destroy (see page 5732)

Create (see page 2766)

32.1.2.3 TControl.Destroy Destructor

Destroys an instance of TControl (see page 5724).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TControl();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the control is not nil, and only then calls Destroy.

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Frees the TFont object in its Font (see page 5823) property.

Calls its inherited destructor.

Override Destroy to free any memory or resources allocated in the Create (see page 5731) method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

Create (see page 5731)

Free (see page 22300)

Owner (see page 2807)

32.1.2.4 TControl Fields

The fields of the TControl class are listed here.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

32.1.2.4.1 TControl.FAnchorMove Field

Pascal

```
FAnchorMove: Boolean;
```

C++

```
Boolean FAnchorMove;
```

Description

This is FAnchorMove, a member of class TControl.

32.1.2.4.2 TControl.FAnchorOrigin Field

Pascal

```
FAnchorOrigin: TPoint;
```

C++

```
TPoint FAnchorOrigin;
```

Description

This is FAnchorOrigin, a member of class TControl.

32.1.2.4.3 TControl.FAnchorRules Field

Pascal

```
FAnchorRules: TPoint;
```

C++

```
TPoint FAnchorRules;
```

Description

This is FAnchorRules, a member of class TControl.

32.1.2.4.4 TControl.FExplicitHeight Field

Pascal

```
FExplicitHeight: Integer;
```

C++

```
int FExplicitHeight;
```

Description

This is FExplicitHeight, a member of class TControl.

32.1.2.4.5 TControl.FExplicitLeft Field

Pascal

```
FExplicitLeft: Integer;
```

C++

```
int FExplicitLeft;
```

Description

This is FExplicitLeft, a member of class TControl.

32.1.2.4.6 TControl.FExplicitTop Field

Pascal

```
FExplicitTop: Integer;
```

C++

```
int FExplicitTop;
```

Description

This is FExplicitTop, a member of class TControl.

32.1.2.4.7 TControl.FExplicitWidth Field

Pascal

```
FExplicitWidth: Integer;
```

C++

```
int FExplicitWidth;
```

Description

This is FExplicitWidth, a member of class TControl.

32.1.2.4.8 TControl.FOriginalParentSize Field

Pascal

```
FOriginalParentSize: TPoint;
```

C++

```
TPoint FOriginalParentSize;
```

Description

This is FOriginalParentSize, a member of class TControl.

32.1.2.5 TControl Methods

The methods of the TControl class are listed here.

Protected Methods

	Name	Description
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.

	DoDock (see page 5750)	Docking event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.

	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.

Public Methods

	Name	Description
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
V	Invalidate (see page 5770)	Completely repaint control.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	Refresh (see page 5779)	Repaints the control on the screen.
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

32.1.2.5.1 TControl.ActionChange Method

Responds when the action associated with the control changes.

Pascal

```
procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); dynamic;
```

C++

```
__fastcall ActionChange(TObject * Sender, Boolean CheckDefaults);
```

Description

ActionChange is called automatically when the action object specified by the Action (see page 5797) property, or any of its properties, changes. By default, the control's action link's OnChange event handler calls ActionChange.

ActionChange updates the control properties that correspond to properties of the associated action. These properties include the Caption (see page 5806), Enabled (see page 5821), Hint (see page 5827), and Visible (see page 5847) properties and the OnClick (see page 5851) event handler.

The Sender parameter is the action object that changed. The CheckDefaults parameter indicates whether the control's properties should all be updated or only those that have not been explicitly changed from their default values. When CheckDefaults is true, only those properties and event handlers that have not been changed from the default values are updated.

See Also

Action (see page 5797)

OnChange (see page 2725)

32.1.2.5.2 TControl.AdjustSize Method

Sets the size of the control as determined by the AutoSize (see page 5803) property.

Pascal

```
procedure AdjustSize; dynamic;
```

C++

```
__fastcall AdjustSize();
```

Description

If the AutoSize (see page 5803) property is true, AdjustSize is called automatically when the control needs to automatically resize itself to its contents. As implemented in TControl (see page 5724), AdjustSize calls SetBounds (see page 5783) with the values of its Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties. This call can result in a change of size if these values are altered in the control's OnCanResize (see page 5851) or OnConstrainedResize (see page 5853) event handlers, or if the control's OnResize (see page 5870) event handler makes any final adjustments.

Descendant classes override AdjustSize to change the size of the control to adjust to its contents.

Other controls can call AdjustSize after changing the control's contents. Do not call AdjustSize in application code.

See Also

AutoSize (see page 5803)

SetBounds (see page 5783)

OnCanResize (see page 5851)

OnConstrainedResize (see page 5853)

OnResize (see page 5870)

32.1.2.5.3 TControl.AssignTo Method

Overrides the inherited AssignTo function to enable assignments to TAction objects.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Do not call the protected AssignTo method in application code. The Assign method of a persistent object calls AssignTo if it is passed the control as a Source and the persistent object does not know how to copy the properties of the control. The Dest parameter is the persistent object that should have its properties copied from the control.

As implemented in TControl (see page 5724), AssignTo checks whether the Dest parameter is a TCustomAction object, and if so, assigns values to its Enabled (see page 5821), Hint (see page 5827), Caption (see page 5806), and Visible (see page 5847) properties and its OnExecute event. This allows arbitrary controls to assign these to an associated action object.

See Also

Assign (see page 2930)

TcustomAction

ActionChange (see page 5737)

32.1.2.5.4 TControl.BeginAutoDrag Method

Begin Automatic Drag.

Pascal

```
procedure BeginAutoDrag; dynamic;
```

C++

```
__fastcall BeginAutoDrag();
```

Description

BeginAutoDrag is called automatically when a user drags a control whose DragMode (see page 5819) is dmAutomatic. Do not call BeginAutoDrag in application code.

As implemented in TControl (see page 5724), BeginAutoDrag calls BeginDrag (see page 5740), passing in the global Mouse (see page 6321) variable's DragImmediate and DragThreshold properties.

See Also

DragKind (see page 5819)

DragMode (see page 5819)

EndDrag (see page 5757)

Mouse (see page 6321)

DragImmediate (see page 6100)

DragThreshold (see page 6101)

Mouse (see page 6321)

32.1.2.5.5 TControl.BeginDrag Method

Starts the dragging of a control.

Pascal

```
procedure BeginDrag(Immediate: Boolean; Threshold: Integer = -1);
```

C++

```
__fastcall BeginDrag(Boolean Immediate, int Threshold = -1);
```

Description

Call BeginDrag to start a drag operation. BeginDrag is called in application code only when the value of the control's DragMode (see page 5819) is dmManual. If DragMode (see page 5819) is dmAutomatic, BeginDrag is called automatically.

If the Immediate parameter is true, the mouse pointer changes to the value of the DragCursor (see page 5818) property and dragging begins immediately. If Immediate is false, the mouse pointer doesn't change to the value of the DragCursor (see page 5818) property and dragging doesn't begin until the user moves the mouse pointer the number of pixels specified by the Threshold parameter. If the caller passes a Threshold value less than 0 (such as the default value for this parameter), BeginDrag uses the DragThreshold property of the global Mouse (see page 6321) variable.

Setting Immediate to false allows the control to accept mouse clicks without beginning a drag-and-drop or drag-and-dock operation.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

EndDrag (see page 5757)

DragThreshold (see page 6101)

Mouse (see page 6321)

DragKind (see page 5819)

32.1.2.5.6 TControl.BringToFront Method

Puts the control in front of all other controls in its parent control.

Pascal

```
procedure BringToFront;
```

C++

```
__fastcall BringToFront();
```

Description

Call BringToFront to ensure that a form is visible or to reorder overlapping controls within a form.

Note that controls that wrap Windows screen objects (control classes descended from TWinControl (see page 6112)) always "stack" above lightweight controls (control classes descended from TGraphicControl (see page 6044)). BringToFront can force a lightweight control, such as a Shape, to stack above other lightweight controls, but wrapped controls, such as an Edit, will still stack above the Shape.

Without BringToFront, controls are stacked in their order they were placed on the form.

C++ Examples:

```
/*
The following code uses two forms. Form1 has a button on it. The
second form is used as a tool palette. This code makes the palette
form visible, and ensures it is the top form by bringing it to the
front.  To run this example, you must include Unit2.h in this
module.
*/
#include "Unit2.h"

void __fastcall TForm1::ShowPaletteButtonClick(TObject *Sender)
{
  if (!Form2->Visible)
  {
    Form2->Visible = true;
    Form2->BringToFront();
  }
}
```

Delphi Examples:

```
{
The following code uses two forms. Form1 has a button on it. The
second form is used as a tool palette. This code makes the palette
form visible, and ensures it is the top form by bringing it to the
front.  To run this example, you must add Unit2 in the uses clause
of your unit.
}
procedure TForm1.ShowPaletteButtonClick(Sender: TObject);
begin
if Form2.Visible = False then Form2.Visible := True;
Form2.BringToFront;
end;
```

See Also

SendToBack (see page 5782)

32.1.2.5.7 TControl.CalcCursorPos Method

Pascal

```
function CalcCursorPos: TPoint;
```

C++

```
__fastcall TPoint CalcCursorPos();
```

Description

This is CalcCursorPos, a member of class TControl.

32.1.2.5.8 TControl.CanAutoSize Method

Indicates whether the control can resize itself automatically to accommodate its contents.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize (see page 5803) property is true and an attempt is made to resize the control. It allows controls to implement the AutoSize (see page 5803) property before the resize sequence that begins with the

OnCanResize (see page 5851) and OnConstrainedResize (see page 5853) events and ends with the OnResize (see page 5870) event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the control. CanAutoSize adjusts these values so that the control's size adjusts to its contents. If the adjustment results in values of NewWidth and NewHeight that differ from the control's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width (see page 5849) and Height (see page 5825), CanAutoSize returns true, and the resize sequence is aborted.

As implemented in TControl (see page 5724), CanAutoSize always returns true without changing NewWidth or NewHeight. Override CanAutoSize to implement the AutoSize (see page 5803) property for a particular control class's contents.

See Also

AutoSize (see page 5803)

OnResize (see page 5870)

OnCanResize (see page 5851)

OnConstrainedResize (see page 5853)

Width (see page 5849)

Height (see page 5825)

32.1.2.5.9 TControl.CanResize Method

OnCanResize (see page 5851) event dispatcher.

Pascal

```
function CanResize(var NewWidth: Integer; var NewHeight: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanResize(int NewWidth, int NewHeight);
```

Description

CanResize is called automatically when an attempt is made to resize the control, after any autosizing has occurred. The NewWidth and NewHeight arguments specify the attempted new values for the control's height and width. CanResize optionally adjusts the values of NewWidth and NewHeight, and returns true if the control can be resized to the returned values. CanResize returns false if the control cannot be resized.

CanResize generates an OnCanResize (see page 5851) event. An OnCanResize (see page 5851) event handler may change the values of NewHeight and NewWidth.

See Also

OnCanResize (see page 5851)

AutoSize (see page 5803)

32.1.2.5.10 TControl.Changed Method

Sends a CM_CHANGED (see page 6330) message to the control.

Pascal

```
procedure Changed;
```

C++

```
__fastcall Changed();
```

Description

Call Changed after changing the properties of the control in a way that affects the properties of a parent object. Changed sends a CM_CHANGED (see page 6330) message back to the control. If the control is a windowed control, this message is bubbled up to the parent so that it can make any necessary adjustments.

See Also

Parent (see page 5832)

32.1.2.5.11 TControl.ChangeScale Method

Rescaling event dispatcher.

Pascal

```
procedure ChangeScale(M: Integer; D: Integer); dynamic;
```

C++

```
__fastcall ChangeScale(int M, int D);
```

Description

ChangeScale is called automatically when some event requires that the control resize itself. Most often, this event is call to the ScaleBy method of the control itself or the control's parent. This allows the control to adjust its own special properties, such as font sizes and display regions.

ChangeScale parameters have the same meaning as for ScaleBy.

Note: Because control sizes and coordinates are integers, scaling can result in round-off error. Repeated calls to ChangeScale can make the control gradually shrink away or creep off the form. Consider destroying and reloading the form after several rescalings.

See Also

Height (see page 5825)

Left (see page 5830)

Top (see page 5844)

Width (see page 5849)

ScaleBy (see page 6175)

32.1.2.5.12 TControl.Click Method

Respond to user click.

Pascal

```
procedure Click; dynamic;
```

C++

```
__fastcall Click();
```

Description

Click is called automatically when the user left-clicks the control. Component or application code can call Click to simulate a user

mouse click. This is often done in menu actions and hotkey handlers.

As implemented in TControl (see page 5724), Click queries whether the OnClick (see page 5851) event handler exists and is different from the OnExecute handler for the control's Action (see page 5797). If this is true, the OnClick (see page 5851) event handler is called. If this is false, and the Action (see page 5797) OnExecute event handler exists, the Action (see page 5797)'s Execute method is called. Override Click to provide additional behavior.

See Also

Action (see page 5797)

OnExecute (see page 2716)

Execute (see page 2712)

DblClick (see page 5747)

OnClick (see page 5851)

32.1.2.5.13 TControl.ClientToParent Method

Translates client coordinates to parent coordinates.

Pascal

```
function ClientToParent(const Point: TPoint; AParent: TWinControl = nil): TPoint;
```

C++

```
__fastcall TPoint ClientToParent(const TPoint Point, TWinControl * AParent = nil);
```

Description

ClientToParent translates Point from the current control's coordinate system to AParent's coordinate system. AParent must be a control that can be found by iterating up the chain of Parent (see page 5832) properties. If AParent is nil (Delphi) or NULL (C++), the control's immediate parent is used.

An EInvalidOperation exception is thrown if AParent is not an ancestor control or the Parent (see page 5832) property is nil (Delphi) or NULL (C++).

See Also

ParentToClient (see page 5776)

ClientToScreen (see page 5744)

32.1.2.5.14 TControl.ClientToScreen Method

Translates a given point from client area coordinates to global screen coordinates.

Pascal

```
function ClientToScreen(const Point: TPoint): TPoint;
```

C++

```
__fastcall TPoint ClientToScreen(const TPoint Point);
```

Description

Use ClientToScreen to convert a point whose coordinates are expressed locally to the control to the corresponding point in screen coordinates. In client area coordinates (0, 0) corresponds to the upper left corner of the control's client area. In screen coordinates (0, 0) corresponds to the upper left corner of the screen.

Use ScreenToClient (see page 5781) along with ClientToScreen to convert from one control's coordinate system to another

control's coordinate system. For example,

```
P := TargetControl.ScreenToClient(SourceControl.ClientToScreen(P));
P = TargetControl->ScreenToClient(SourceControl->ClientToScreen(P));
```

converts P from coordinates in SourceControl to coordinates in TargetControl.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
```

```
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

ScreenToClient (see page 5781)

32.1.2.5.15 TControl.ConstrainedResize Method

Notify control of resize with constraints.

Pascal

```
procedure ConstrainedResize(var MinWidth: Integer; var MinHeight: Integer; var MaxWidth:
Integer; var MaxHeight: Integer); virtual;
```

C++

```
virtual __fastcall ConstrainedResize(int MinWidth, int MinHeight, int MaxWidth, int MaxHeight);
```

Description

ConstrainedResize is called automatically when a control is resized and has nonzero Constraints (see page 5812). This allows the control to modify the constraints before they are applied.

As implemented in TControl (see page 5724), ConstrainedResize simply calls the OnConstrainedResize (see page 5853) event handler.

See Also

OnConstrainedResize (see page 5853)

CanResize (see page 5742)

Constraints (see page 5812)

32.1.2.5.16 TControl.DblClick Method

OnDblClick (see page 5854) event dispatcher.

Pascal

```
procedure DblClick; dynamic;
```

C++

```
__fastcall DblClick();
```

Description

DblClick calls any event handler attached to the OnDblClick (see page 5854) event. Override DblClick to provide other responses in addition to the inherited event-handler call when the user double-clicks the control.

The DblClick method is triggered by left mouse-button double-click message (WM_LBUTTONDBLCLK) from Windows.

See Also

Click (see page 5743)

OnDblClick (see page 5854)

32.1.2.5.17 TControl.DefaultDockImage Method

Handles the painting and erasing of a dock image.

Pascal

```
procedure DefaultDockImage(DragDockObject: TDragDockObject; Erase: Boolean); dynamic;
```

C++

```
__fastcall DefaultDockImage(TDragDockObject DragDockObject, Boolean Erase);
```

Description

The DrawDragDockImage (see page 5755) and EraseDragDockImage (see page 5757) methods call DefaultDockImage to paint or erase the drag image of a control while the user drags it in a drag-and-dock operation.

DragDockObject contains the drag-and-dock information for the control.

Erase specifies whether to draw or erase the image.

See Also

DrawDragDockImage (see page 5755)

EraseDragDockImage (see page 5757)

OnStartDock (see page 5871)

32.1.2.5.18 TControl.DefaultHandler Method

Provides message handling for all messages that do not have specific handlers.

Pascal

```
procedure DefaultHandler(var Message); override;
```

C++

```
virtual __fastcall DefaultHandler( Message);
```

Description

Override DefaultHandler to extend the default handling of messages. DefaultHandler is the final opportunity for component writers to handle messages to controls. The method referenced by WindowProc (see page 5872) provides the first opportunity. Individual message methods such as Click (see page 5743) provide additional opportunities.

DefaultHandler handles the messages Windows sends to manage the control's text: WM_GETTEXT, WM_SETTEXT, and WM_GETTEXTLENGTH.

See Also

DefaultHandler

WndProc (see page 5791)

WindowProc (see page 5872)

TMessage

32.1.2.5.19 TControl.DefineProperties Method

Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

DefineProperties is called automatically by the streaming system that loads and saves VCL components. It allows persistent objects to read and write properties that are not published. Published properties are loaded and stored automatically.

DefineProperties uses the methods of the filer passed as the Filer parameter to assign methods that can load and save the IsControl (see page 5829) property.

Override DefineProperties to persist unpublished properties. Call the DefineBinaryProperty or DefineProperty method of the Filer parameter to assign methods that load or save property values.

Note: When overriding this method, call the parent class's DefineProperties method first.

See Also

DefineBinaryProperty (see page 2837)

DefineProperty (see page 2838)

DefineProperties (see page 2771)

IsControl (see page 5829)

32.1.2.5.20 TControl.DesignWndProc Method

Pascal

```
function DesignWndProc(var Message: TMessage): Boolean; dynamic;
```

C++

```
__fastcall Boolean DesignWndProc(TMessage Message);
```

Description

This is DesignWndProc, a member of class TControl.

32.1.2.5.21 TControl.Dock Method

Used internally to dock the control.

Pascal

```
procedure Dock(NewDockSite: TWinControl; ARect: TRect); dynamic;
```

C++

```
__fastcall Dock(TWinControl * NewDockSite, TRect ARect);
```

Description

Do not call Dock in applications. It is called automatically to implement many of the details of docking the control to a dock site.

To dock the control programmatically, use ManualDock (see page 5771) instead.

The NewDockSite parameter specifies the docking site to which the control is being docked. The ARect parameter indicates the new dimensions of the control when it is docked to NewDockSite.

See Also

ManualDock (see page 5771)

32.1.2.5.22 TControl.DockTrackNoTarget Method

Handles dragging of the control over an invalid docking target.

Pascal

```
procedure DockTrackNoTarget(Source: TDragDockObject; X: Integer; Y: Integer); dynamic;
```

C++

```
__fastcall DockTrackNoTarget(TDragDockObject Source, int X, int Y);
```

Description

DockTrackNoTarget is called automatically when the control is dragged over an invalid docking target. It updates the docking rectangle of the TDragDockObject (see page 6009) specified by Source to reflect the current mouse coordinates, as given by X and Y.

See Also

DockRect (see page 6015)

PositionDockRect (see page 5777)

32.1.2.5.23 TControl.DoContextPopup Method

OnContextPopup (see page 5854) event dispatcher.

Pascal

```
procedure DoContextPopup(MousePos: TPoint; var Handled: Boolean); dynamic;
```

C++

```
__fastcall DoContextPopup(TPoint MousePos, Boolean Handled);
```

Description

OnContextPopup (see page 5854) calls any event handler attached to the OnContextPopup (see page 5854) event. When creating a component that descends from TControl (see page 5724), override OnContextPopup (see page 5854) to provide other responses in addition to the inherited event-handler call when the user invokes the control's context menu.

MousePos indicates the position where the user right-clicked the control. If the user invoked the context menu using the keyboard, MousePos is (-1,-1).

Handled returns a Boolean value indicating whether the control should continue with its default response to the user action.

See Also

OnContextPopup (see page 5854)

32.1.2.5.24 TControl.DoDock Method

Docking event dispatcher.

Pascal

```
procedure DoDock(NewDockSite: TWinControl; var ARect: TRect); dynamic;
```

C++

```
__fastcall DoDock(TWinControl * NewDockSite, TRect ARect);
```

Description

DoDock is called automatically as part of the docking process. It updates the control's dimensions and causes it to repaint.

NewDockSite is the control's new dock site. This value can be nil (Delphi) or NULL (C++) or NullDockSite (see page 6377) (a constant that indicates no dock site).

ARect specifies the control's new dimensions once it is docked to NewDockSite.

See Also

ManualDock (see page 5771)

ManualFloat (see page 5772)

32.1.2.5.25 TControl.DoEndDock Method

OnEndDock (see page 5859) event dispatcher.

Pascal

```
procedure DoEndDock(Target: TObject; X: Integer; Y: Integer); dynamic;
```

C++

```
__fastcall DoEndDock(TObject * Target, int X, int Y);
```

Description

OnEndDock (see page 5859) calls any event handler attached to the OnEndDock (see page 5859) event. When creating a component that descends from TControl (see page 5724), override OnEndDock (see page 5859) to provide other responses in addition to the inherited event-handler call when the user finishes a drag-and-dock operation.

See Also

DoStartDock (see page 5753)

OnEndDock (see page 5859)

32.1.2.5.26 TControl.DoEndDrag Method

OnEndDrag (see page 5860) event dispatcher.

Pascal

```
procedure DoEndDrag(Target: TObject; X: Integer; Y: Integer); dynamic;
```

C++

```
__fastcall DoEndDrag(TObject * Target, int X, int Y);
```

Description

DoEndDrag calls any event handler attached to the OnEndDrag (see page 5860) event. When creating a component that descends from TControl (see page 5724), override DoEndDrag to provide other responses in addition to the inherited event-handler call when the user drops an object or cancels a drag-and-drop operation.

See Also

DoStartDrag (see page 5753)

OnEndDrag (see page 5860)

32.1.2.5.27 TControl.DoMouseWheel Method

Processes mouse wheel motion.

Pascal

```
function DoMouseWheel(Shift: TShiftState; WheelDelta: Integer; MousePos: TPoint): Boolean;
dynamic;
```

C++

```
__fastcall Boolean DoMouseWheel(TShiftState Shift, int WheelDelta, TPoint MousePos);
```

Description

DoMouseWheel is called automatically when the user rotates the mouse wheel.

The Shift parameter indicates the current state of the Shift, Alt, and Ctrl keys.

The WheelDelta parameter indicates the distance the wheel was rotated. WheelDelta is positive if the mouse was rotated upward, negative if the mouse was rotated downward.

The MousePos parameter indicates the current position of the mouse pointer.

DoMouseWheel returns true if an event handler was found and indicated that it handled the event.

As implemented in TControl (see page 5724), DoMouseWheel first attempts to call the OnMouseWheel (see page 5869) event handler. If this routine exists and succesfully handles the event, DoMouseWheel returns true. Otherwise, DoMouseWheel repeatedly calls DoMouseWheelDown (see page 5752) or DoMouseWheelUp (see page 5752), depending on the direction

the user turned the mouse wheel. These routines, in turn, call the OnMouseWheelUp (see page 5870) or OnMouseWheelDown (see page 5869) event handlers.

Before the first call to DoMouseWheelDown (see page 5752) or DoMouseWheelUp (see page 5752), DoMouseWheel adds WheelDelta to the WheelAccumulator (see page 5849) property. After each call, DoMouseWheel adjusts WheelAccumulator (see page 5849) towards zero; each adjustment is by an internal constant that defines the minimum meaningful wheel motion. When the absolute value of of WheelAccumulator (see page 5849) drops below the value of this constant, DoMouseWheel exits, returning the value returned by the final call to DoMouseWheelDown (see page 5752) or DoMouseWheelUp (see page 5752).

See Also

OnMouseWheel (see page 5869)

OnMouseWheelDown (see page 5869)

OnMouseWheelUp (see page 5870)

MouseWheelHandler (see page 5774)

WheelAccumulator (see page 5849)

32.1.2.5.28 TControl.DoMouseWheelDown Method

OnMouseWheelDown (see page 5869) event dispatcher.

Pascal

```
function DoMouseWheelDown(Shift: TShiftState; MousePos: TPoint): Boolean; dynamic;
```

C++

```
__fastcall Boolean DoMouseWheelDown(TShiftState Shift, TPoint MousePos);
```

Description

The DoMouseWheel (see page 5751) method calls DoMouseWheelDown to generate an OnMouseWheelDown (see page 5869) event.

The Shift parameter indicates the state of the Shift, Alt, and Ctrl keys.

The MousePos parameter indicates the position of the mouse pointer.

DoMouseWheelDown returns true if an event handler indicates that it handled the message, and false otherwise.

Override DoMouseWheelDown to perform class-specific tasks in addition to calling the event handler when the mouse wheel is rotated downward.

See Also

OnMouseWheelDown (see page 5869)

DoMouseWheel (see page 5751)

32.1.2.5.29 TControl.DoMouseWheelUp Method

OnMouseWheelUp (see page 5870) event dispatcher.

Pascal

```
function DoMouseWheelUp(Shift: TShiftState; MousePos: TPoint): Boolean; dynamic;
```

C++

```
__fastcall Boolean DoMouseWheelUp(TShiftState Shift, TPoint MousePos);
```

Description

The DoMouseWheel (see page 5751) method calls DoMouseWheelUp to generate an OnMouseWheelUp (see page 5870) event.

The Shift parameter indicates the state of the Shift, Alt, and Ctrl keys.

The MousePos parameter indicates the position of the mouse pointer.

DoMouseWheelUp returns true if an event handler indicates that it handled the message, and false otherwise.

Override DoMouseWheelUp to perform class-specific tasks in addition to calling the event handler when the mouse wheel is rotated upward.

See Also

OnMouseWheelUp (see page 5870)

DoMouseWheel (see page 5751)

32.1.2.5.30 TControl.DoStartDock Method

OnStartDock (see page 5871) event dispatcher.

Pascal

```
procedure DoStartDock(var DragObject: TDragObject); dynamic;
```

C++

```
__fastcall DoStartDock(TDragObject DragObject);
```

Description

DoStartDock calls any event handler attached to the OnStartDock (see page 5871) event. When creating a component that descends from TControl (see page 5724), override DoStartDock to provide other responses in addition to the inherited event-handler call when the user starts a drag-and-dock operation.

See Also

DoEndDock (see page 5750)

OnStartDock (see page 5871)

32.1.2.5.31 TControl.DoStartDrag Method

OnStartDrag (see page 5871) event dispatcher.

Pascal

```
procedure DoStartDrag(var DragObject: TDragObject); dynamic;
```

C++

```
__fastcall DoStartDrag(TDragObject DragObject);
```

Description

DoStartDrag calls any event handler attached to the OnStartDrag (see page 5871) event. When creating a component that descends from TControl (see page 5724), override DoStartDrag to provide other responses in addition to the inherited event-handler call when the user starts a drag-and-drop operation.

See Also

DoEndDrag (see page 5751)

OnStartDrag (see page 5871)

32.1.2.5.32 TControl.DragCanceled Method

Provides the interface for a method that responds to the cancellation of a drag.

Pascal

```
procedure DragCanceled; dynamic;
```

C++

```
__fastcall DragCanceled();
```

Description

DragCanceled is called automatically when a drag operation is canceled. As implemented in TControl (see page 5724), DragCanceled does nothing. When creating a component that descends from TControl (see page 5724), override DragCanceled to provide code that executes when a drag-and-drop or drag-and-dock operation is canceled.

See Also

OnEndDrag (see page 5860)

OnEndDock (see page 5859)

32.1.2.5.33 TControl.DragDrop Method

OnDragDrop (see page 5855) event dispatcher.

Pascal

```
procedure DragDrop(Source: TObject; X: Integer; Y: Integer); dynamic;
```

C++

```
__fastcall DragDrop(TObject * Source, int X, int Y);
```

Description

When creating a component that descends from TControl (see page 5724), override DragDrop to add additional code that executes before the OnDragDrop (see page 5855) event handler is called.

The Source parameter is the object that was dropped onto the control. The X and Y parameters are the mouse coordinates where the object was dropped.

See Also

DragCanceled (see page 5754)

OnDragDrop (see page 5855)

32.1.2.5.34 TControl.Dragging Method

Indicates whether a control is being dragged.

Pascal

```
function Dragging: Boolean;
```

C++

```
__fastcall Boolean Dragging();
```

Description

Use Dragging to determine the drag state of the control. If Dragging returns true, the control is being dragged. If Dragging is false, the control is not being dragged.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

OnDragDrop (see page 5855)

OnDockOver (see page 6216)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

32.1.2.5.35 TControl.DragOver Method

OnDragOver (see page 5857) event dispatcher.

Pascal

```
procedure DragOver(Source: TObject; X: Integer; Y: Integer; State: TDragState; var Accept:
Boolean); dynamic;
```

C++

```
__fastcall DragOver(TObject * Source, int X, int Y, TDragState State, Boolean Accept);
```

Description

Override DragOver to add additional code that executes before the OnDragOver (see page 5857) event handler is called.

DragOver sets the Accept parameter to true to indicate that the user can drop the dragged object on the control. It sets Accept to false to indicate that the user cannot drop the dragged object on the control.

The Source parameter is the object being dragged. The State parameter indicates how the dragged object is moving in relation to the control. X and Y indicate the current position of the mouse.

See Also

DragDrop (see page 5754)

OnDragOver (see page 5857)

32.1.2.5.36 TControl.DrawDragDockImage Method

Draws the dock image for the control.

Pascal

```
procedure DrawDragDockImage(DragDockObject: TDragDockObject); dynamic;
```

C++

```
__fastcall DrawDragDockImage(TDragDockObject DragDockObject);
```

Description

DrawDragDockImage is called automatically to draw the control (or, more typically, the drag rectangle) when the control is manipulated in drag-and-dock operations.

DrawDragDockImage calls DefaultDockImage (see page 5747) to render the image.

See Also

DefaultDockImage (see page 5747)

EraseDragDockImage (see page 5757)

32.1.2.5.37 TControl.DrawTextBiDiModeFlags Method

Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.

Pascal

```
function DrawTextBiDiModeFlags(Flags: Longint): Longint;
```

C++

```
__fastcall Longint DrawTextBiDiModeFlags(Longint Flags);
```

Description

Call DrawTextBiDiModeFlags to obtain a value for the TextFlags property of a canvas when drawing the control's text or caption. The return value can also be used with the Windows API DrawText function.

Flags are the current text flags, unaltered by any bi-directional mode considerations.

DrawTextBiDiModeFlags adds or removes the DT_RIGHT or DT_RTLREADING flags based on the values returned from the UseRightToLeftAlignment (see page 5790) and UseRightToLeftReading (see page 5790) methods.

See Also

TextFlags (see page 14495)

DrawTextBiDiModeFlagsReadingOnly (see page 5756)

BiDiMode (see page 5803)

Text (see page 5842)

Caption (see page 5806)

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

32.1.2.5.38 TControl.DrawTextBiDiModeFlagsReadingOnly Method

Returns the text flag to add that indicates whether the control's text should read from right to left.

Pascal

```
function DrawTextBiDiModeFlagsReadingOnly: Longint;
```

C++

```
__fastcall Longint DrawTextBiDiModeFlagsReadingOnly();
```

Description

Call DrawTextBiDiModeFlagsReadingOnly to determine whether the DT_RTLREADING flag should be added when calling the Windows API DrawText or when assigning the TextFlags property of a canvas.

DrawTextBiDiModeFlagsReadingOnly returns the DT_RTLREADING flag or 0, depending on the value returned from the UseRightToLeftReading (see page 5790) method.

To obtain the text flags that indicate the bi-directional alignment as well as text direction for the control, use DrawTextBiDiModeFlags (see page 5756) instead.

See Also

TextFlags (see page 14495)

DrawTextBiDiModeFlags (see page 5756)

BiDiMode (see page 5803)

Text (see page 5842)

Caption (see page 5806)

UseRightToLeftReading (see page 5790)

32.1.2.5.39 TControl.EndDrag Method

Stops a control from being dragged any further.

Pascal

```
procedure EndDrag(Drop: Boolean);
```

C++

```
__fastcall EndDrag(Boolean Drop);
```

Description

Use EndDrag to stop a dragging operation that began with a call to the BeginDrag (see page 5740) method.

If the Drop parameter is true, the control being dragged is dropped or docked. If the Drop parameter is false, the control is not dropped or docked and dragging is canceled.

See Also

BeginDrag (see page 5740)

OnEndDrag (see page 5860)

OnEndDock (see page 5859)

DragKind (see page 5819)

32.1.2.5.40 TControl.EraseDragDockImage Method

Erases the dock image for the control.

Pascal

```
procedure EraseDragDockImage(DragDockObject: TDragDockObject); dynamic;
```

C++

```
__fastcall EraseDragDockImage(TDragDockObject DragDockObject);
```

Description

DrawDragDockImage (see page 5755) is called automatically to erase the control (or, more typically, the drag rectangle) when the control is manipulated in drag-and-dock operations.

See Also

DefaultDockImage (see page 5747)

EraseDragDockImage

32.1.2.5.41 TControl.GetAction Method

Pascal

```
function GetAction: TBasicAction; virtual;
```

C++

```
virtual __fastcall TBasicAction GetAction();
```

Description

This is GetAction, a member of class TControl.

32.1.2.5.42 TControl.GetActionLinkClass Method

Returns the appropriate action link class that is associated with the control class.

Pascal

```
function GetActionLinkClass: TControlActionLinkClass; dynamic;
```

C++

```
__fastcall TControlActionLinkClass GetActionLinkClass();
```

Description

GetActionLinkClass returns the appropriate class type for the action link used with a control. This class is used internally to create an action link object for the control when it has an associated action. The action link links the action to the control client. Each Action (see page 5797) link class is designed to link specific properties and event handlers of the action to its client, based on the types of properties the client supports.

As implemented in TControl (see page 5724), GetActionLinkClass returns the TControlActionLink (see page 5892) class type, which associates the ShowHint (see page 5840), Caption (see page 5806), Enabled (see page 5821) and Visible (see page 5847) properties and the OnClick (see page 5851) event handler. Descendants of TControl (see page 5724) override this method to specify a descendant of TControlActionLink (see page 5892) that handles their configuration of properties and events.

See Also

TControlActionLink (see page 5892)

ActionLink (see page 5798)

Action (see page 5797)

32.1.2.5.43 TControl.GetClientOrigin Method

Returns the screen coordinates of the top-left corner of the control's client area.

Pascal

```
function GetClientOrigin: TPoint; virtual;
```

C++

```
virtual __fastcall TPoint GetClientOrigin();
```

Description

Override GetClientOrigin to change how the ClientOrigin (see page 5809) is calculated.

The GetClientOrigin method is the protected method for reading the ClientOrigin (see page 5809) property. GetClientOrigin

calculates the ClientOrigin (see page 5809) by adding the control's Left (see page 5830) and Top (see page 5844) values to the ClientOrigin (see page 5809) coordinates of its parent.

See Also

ClientOrigin (see page 5809)

GetClientRect (see page 5759)

Left (see page 5830)

Top (see page 5844)

32.1.2.5.44 TControl.GetClientRect Method

Returns a rectangle defining the client area of the control.

Pascal

```
function GetClientRect: TRect; virtual;
```

C++

```
virtual __fastcall TRect GetClientRect();
```

Description

Override GetClientRect to change how the client area is calculated.

The GetClientRect method is the protected method for reading the ClientRect (see page 5810) property. GetClientRect returns a rectangle with the rectangle's Top (see page 5844) and Left (see page 5830) fields set to zero, and its Bottom and Right fields set to the control's ClientHeight (see page 5808) and ClientWidth (see page 5811), respectively.

See Also

BoundsRect (see page 5803)

ClientHeight (see page 5808)

ClientWidth (see page 5811)

GetClientOrigin (see page 5758)

32.1.2.5.45 TControl.GetControlsAlignment Method

Indicates how text is aligned within the control.

Pascal

```
function GetControlsAlignment: TAlignment; dynamic;
```

C++

```
__fastcall TAlignment GetControlsAlignment();
```

Description

Use GetControlsAlignment to determine the alignment of the control. For controls that do not have an Alignment property, GetControlsAlignment returns taLeftJustify.

Note: GetControlsAlignment returns the alignment of text before it is altered by the BiDiMode (see page 5803) property. To check if the alignment should be reversed to reflect the BiDiMode (see page 5803) property, call UseRightToLeftAlignment (see page 5790).

See Also

BiDiMode (see page 5803)

Text (see page 5842)

Caption (see page 5806)

UseRightToLeftAlignment (see page 5790)

32.1.2.5.46 TControl.GetDeviceContext Method

Returns a device context for the control.

Pascal

```
function GetDeviceContext(var WindowHandle: HWnd): HDC; virtual;
```

C++

```
virtual __fastcall HDC GetDeviceContext(HWnd WindowHandle);
```

Description

Call GetDeviceContext to obtain a device context and window handle. Override GetDeviceContext to change how the device context is obtained.

The window handle used by the control is returned in the WindowHandle parameter. GetDeviceContext returns the device context of this, the parent control's window.

C++ Examples:

```
/*
The GetFormImage allows you to easily obtain a bitmap of a
form. The following method can be added to a custom
TWinControl descendant to save itself as a bitmap.  Check the
produced BMP file to confirm.  Click the button and them
double click on the newly created foo.bmp file to view.
*/
class TMyControl : public TColorListBox
{
__published:    // IDE-managed Components
    void __fastcall SaveAsBmp(TFileName fileName);
private:    // User declarations
public:     // User declarations
    __fastcall TMyControl(TComponent* Owner);
};

TForm1 *Form1;
TMyControl *MyControl1;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
__fastcall TMyControl::TMyControl(TComponent* Owner)
    : TColorListBox(Owner)
{
}

typedef struct {
  TLogPalette lpal;
  TPaletteEntry dummy[256];
} LogPal;

void __fastcall TMyControl::SaveAsBmp(TFileName fileName)
```

```
{
  TCanvas *tempCanvas = new TCanvas;
  try
  {
    HWND notUsed;
    tempCanvas->Handle = GetDeviceContext(notUsed);
    TImage *image2save = new TImage(Form1);
    try
    {
      image2save->Height = Height;
      image2save->Width = Width;
      TRect destRect = Rect(0,0,Width,Height);
      TRect sourceRect = destRect;
      image2save->Canvas->CopyRect(destRect, tempCanvas, sourceRect);
      LogPal SysPal;
      SysPal.lpal.palVersion = 0x300;
      SysPal.lpal.palNumEntries = 256;
      GetSystemPaletteEntries(tempCanvas->Handle,0,256,SysPal.lpal.palPalEntry);
      image2save->Picture->Bitmap->Palette = CreatePalette((const tagLOGPALETTE
*)&SysPal.lpal);
      image2save->Picture->SaveToFile(fileName);
    }
    __finally
    {
     delete image2save;
    }
  }
  __finally
  {
    delete tempCanvas;
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  MyControl1->SaveAsBmp("../foo.bmp");
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyControl1 = new TMyControl(Form1);
  MyControl1->Parent = Form1;
  MyControl1->Visible  = true;
}
```

Delphi Examples:

```
{
The GetFormImage allows you to easily obtain a bitmap of a
form. The following method can be added to a custom
TWinControl descendant to save itself as a bitmap.  Check the
produced BMP file to confirm.  Click the button and them
double click on the newly created foo.bmp file to view.
}
type
  TMyControl = class(TColorListBox)
    procedure SaveAsBmp(fileName: TFileName);
  end;
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;
  MyControl1: TMyControl;
implementation

{$R *.dfm}

type
  LogPal = record
  lpal : TLogPalette;
  dummy:Array[0..255] of TPaletteEntry;
  end;

procedure TMyControl.SaveAsBmp(fileName: TFileName);
var
  Source: TComponent;
  SysPal : LogPal;
  tempCanvas: TCanvas;
  sourceRect, destRect: TRect;
  image2save: TImage;
  notUsed: HWND;
begin
  tempCanvas := TCanvas.Create;
  try
    tempCanvas.Handle := GetDeviceContext(notUsed);
    image2save:=TImage.create(self);
    try
      with image2save do
      begin
        Height := Self.Height;
        Width :=  Self.Width;
        destRect := Rect(0,0,Width,Height);
        sourceRect := destRect;
        Canvas.CopyRect(destRect,tempCanvas,sourceRect);
        SysPal.lPal.palVersion:=$300;
        SysPal.lPal.palNumEntries:=256;
        GetSystemPaletteEntries(
          tempCanvas.Handle,0,256,SysPal.lpal.palPalEntry);
        Picture.Bitmap.Palette:= CreatePalette(Syspal.lpal);
      end;
      image2save.Picture.SaveToFile(fileName);
    finally
     image2save.Free;
    end;
  finally
    tempCanvas.Free;
  end;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  MyControl1.SaveAsBmp('foo.bmp');
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyControl1:= TMyControl.Create(Form1);
  MyControl1.Parent:= Form1;
  MyControl1.visible := true;
end;
```

See Also

Parent (see page 5832)

WindowHandle (see page 6212)

Handle (see page 14489)

32.1.2.5.47 TControl.GetDockEdge Method

Returns the edge on which another control should be docked when it is dragged over the control.

Pascal

```
function GetDockEdge(MousePos: TPoint): TAlign; dynamic;
```

C++

```
__fastcall TAlign GetDockEdge(TPoint MousePos);
```

Description

Use GetDockEdge to get the edge on which another control should be docked when it is dragged over the control.

MousePos is the position of the mouse.

GetDockEdge is called automatically during drag-and-dock operations involving the control.

See Also

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

ManualDock (see page 5771)

32.1.2.5.48 TControl.GetDragImages Method

Returns the control's image list containing the image to be displayed while dragging.

Pascal

```
function GetDragImages: TDragImageList; virtual;
```

C++

```
virtual __fastcall TDragImageList GetDragImages();
```

Description

Override GetDragImages to provide a custom image list for the control. As implemented in TControl (see page 5724), GetDragImages returns nil (Delphi) or NULL (C++).

See Also

DragCursor (see page 5818)

OnStartDrag (see page 5871)

32.1.2.5.49 TControl.GetEnabled Method

Returns the value of the Enabled (see page 5821) property.

Pascal

```
function GetEnabled: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetEnabled();
```

Description

GetEnabled is the read implementation of the Enabled (see page 5821) property.

Override GetEnabled and SetEnabled (see page 5785) to change the implementation of the Enabled (see page 5821) property. For example, A descendant of TControl (see page 5724) might change the way the Enabled (see page 5821) property interacts with an associated action.

See Also

Enabled (see page 5821)

SetEnabled (see page 5785)

32.1.2.5.50 TControl.GetFloating Method

Returns the value of the Floating (see page 5823) property.

Pascal

```
function GetFloating: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetFloating();
```

Description

GetFloating is the implementation of the read-only Floating (see page 5823) property. GetFloating returns true if the HostDockSite (see page 5829) property is either nil (Delphi) or NULL (C++) or of type FloatingDockSiteClass (see page 5823).

Override this method to change the way a control determines if it is floating.

See Also

Floating (see page 5823)

FloatingDockSiteClass (see page 5823)

HostDockSite (see page 5829)

32.1.2.5.51 TControl.GetFloatingDockSiteClass Method

Returns the class used for the control when it is floating.

Pascal

```
function GetFloatingDockSiteClass: TWinControlClass; virtual;
```

C++

```
virtual __fastcall TWinControlClass GetFloatingDockSiteClass();
```

Description

GetFloatingDockSiteClass is the read implementation of the FloatingDockSiteClass (see page 5823) property. Override this method to change the implementation of FloatingDockSiteClass (see page 5823).

See Also

FloatingDockSiteClass (see page 5823)

32.1.2.5.52 TControl.GetPalette Method

Returns the handle to a palette for use by the control.

Pascal

```
function GetPalette: HPALETTE; dynamic;
```

C++

```
__fastcall HPALETTE GetPalette();
```

Description

As implemented in TControl (see page 5724), GetPalette returns a null handle, indicating that the control does not have a palette. Override GetPalette to implement derived objects that use a palette. Create (see page 5731) a palette handle for the control by a call to the Windows API function CreatePalette. GetPalette should return the palette handle for the control.

Specifying the palette for a control tells the application that the control's palette needs to be realized and designates the palette to use for realization.

GetPalette is called only if the run-time video mode requires color palette support, for example, in 256-color mode.

See Also

PaletteChanged (see page 5775)

32.1.2.5.53 TControl.GetParentComponent Method

Returns the parent of the control.

Pascal

```
function GetParentComponent: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetParentComponent();
```

Description

GetParentComponent is called by the streaming system that loads and saves VCL components. The component returned by GetParentComponent is responsible for loading and saving the control when it is streamed in or out. As implemented in TControl (see page 5724), GetParentComponent returns the value of the Parent (see page 5832) property.

See Also

Parent (see page 5832)

GetParentComponent

SetParentComponent (see page 5787)

HasParent (see page 5768)

32.1.2.5.54 TControl.GetPopupMenu Method

Returns the pop-up menu associated with the control.

Pascal

```
function GetPopupMenu: TPopupMenu; dynamic;
```

C++

```
__fastcall TPopupMenu GetPopupMenu();
```

Description

Call GetPopupMenu to find the popup menu associated with the control. Override GetPopupMenu to change which popup menu is associated with the control or to specify different popup menus depending on the state of the control.

GetPopupMenu returns the value of the Popup property.

See Also

PopupMenu (see page 5835)

32.1.2.5.55 TControl.GetTextBuf Method

Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.

Pascal

```
function GetTextBuf(Buffer: PChar; BufSize: Integer): Integer;
```

C++

```
__fastcall int GetTextBuf(const char * Buffer, int BufSize);
```

Description

Call GetTextBuf to retrieve the text of a control into a fixed size buffer. The text copied is the value of the Text (see page 5842) property. GetTextBuf returns the number of characters that were actually copied, which is either the length of the Text (see page 5842) property, or BufSize - 1, whichever is smaller.

To find out how many characters the buffer needs to hold the entire text, call the GetTextLen (see page 5767) method before allocating a buffer for GetTextBuf.

Note: GetTextBuf is available for compatibility with 16-bit code. Where backward compatibility is not an issue, use the Text (see page 5842) property.

Note: To obtain the control's text as an AnsiString, use the Text (see page 5842) property instead.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
  Size++;                        //Add room for null character
  char *Buffer = new char[Size];  //Creates Buffer dynamic variable
  Edit1->GetTextBuf(Buffer,Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer;
  delete [] Buffer;
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
```

```
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
  Inc(Size);                      {Add room for null character}
  GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
  Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
end;
```

See Also

GetTextLen (see page 5767)

Text (see page 5842)

32.1.2.5.56 TControl.GetTextLen Method

Returns the length of the control's text.

Pascal

```
function GetTextLen: Integer;
```

C++

```
__fastcall int GetTextLen();
```

Description

Call GetTextLen to find the length of the control's text. This is the size needed for a text buffer to be used by the GetTextBuf (see page 5766) method.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
```

```
  Size++;                        //Add room for null character
  char *Buffer = new char[Size];  //Creates Buffer dynamic variable
  Edit1->GetTextBuf(Buffer,Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer;
  delete [] Buffer;
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
  Inc(Size);                      {Add room for null character}
  GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
  Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
end;
```

See Also

GetTextBuf (see page 5766)

Text (see page 5842)

32.1.2.5.57 TControl.HasParent Method

Indicates whether the control has a parent.

Pascal

```
function HasParent: Boolean; override;
```

C++

```
virtual __fastcall Boolean HasParent();
```

Description

HasParent indicates whether the Parent (see page 5832) property has been assigned a value. This method is used by the streaming system to determine whether the control has a parent that is responsible for writing it to a stream. As implemented in TControl (see page 5724), HasParent returns true if the Parent (see page 5832) property is not nil (Delphi) or NULL (C++),

and false otherwise.

See Also

GetParentComponent (see page 5765)

SetParentComponent (see page 5787)

Parent (see page 5832)

32.1.2.5.58 TControl.Hide Method

Makes the control invisible.

Pascal

```
procedure Hide;
```

C++

```
__fastcall Hide();
```

Description

Call Hide to hide a control. Hide sets the Visible (see page 5847) property of the control to false.

Although a control that is hidden is not visible, its properties and methods are still available.

See Also

Show (see page 5788)

Visible (see page 5847)

32.1.2.5.59 TControl.InitiateAction Method

Calls the action link's Update (see page 5789) method if the control is associated with an action link.

Pascal

```
procedure InitiateAction; virtual;
```

C++

```
virtual __fastcall InitiateAction();
```

Description

When the application is idle, the library makes a series of calls to allow controls to update their associated actions so that they reflect the current properties of the controls. The first of these is to the InitiateAction method of the each form, followed by the InitiateAction methods of its visible menus and controls.

InitiateAction calls the action's update method indirectly (through its action link), which leads to a number of events, any of which may update the action and stop the update process:

1.First, the action list that contains the action receives an OnUpdate event.

2.If the action list does not handle the OnUpdate event, then the Application object receives an OnActionUpdate event.

3.If the OnActionUpdate event handler does not update the action, the action itself receives an OnUpdate event.

4.If that does not update the action, the active control's UpdateAction method is called.

5. Finally, if the action is still not updated, the active form's UpdateAction method is called.

See Also

OnActionUpdate (see page 13726)

Action (see page 5797)

ActionLink (see page 5798)

UpdateAction

OnUpdate (see page 327)

OnUpdate (see page 2718)

Update (see page 2714)

32.1.2.5.60 TControl.Invalidate Method

Completely repaint control.

Pascal

```
procedure Invalidate; virtual;
```

C++

```
virtual __fastcall Invalidate();
```

Description

Use Invalidate when the entire control needs to be repainted. When more than one region within the control needs repainting, Invalidate will cause the entire window to be repainted in a single pass, avoiding flicker caused by redundant repaints. There is no performance penalty for calling Invalidate multiple times before the control is actually repainted.

See Also

Refresh (see page 5779)

Repaint (see page 5779)

Update (see page 5789)

32.1.2.5.61 TControl.IsRightToLeft Method

Indicates whether the control should be reversed right to left.

Pascal

```
function IsRightToLeft: Boolean;
```

C++

```
__fastcall Boolean IsRightToLeft();
```

Description

Use IsRightToLeft to determine whether the control should be reversed in any way because it is running on a system with a Middle Eastern locale. IsRightToLeft is intended for component writers to ease the process of implementing the BiDiMode (see page 5803) property.

IsRightToLeft returns true if the application is running on a Middle East-enabled version of Windows and the BiDiMode (see page 5803) property indicates that the control should make any adjustments.

See Also

BiDiMode (see page 5803)

UseRightToLeftReading (see page 5790)

UseRightToLeftAlignment (see page 5790)

UseRightToLeftScrollBar (see page 5791)

32.1.2.5.62 TControl.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action (see page 5797). To change the properties the control copies from its action, override the ActionChange (see page 5737) method.

See Also

Loaded (see page 2782)

Action (see page 5797)

ActionChange (see page 5737)

32.1.2.5.63 TControl.ManualDock Method

Docks the control.

Pascal

```
function ManualDock(NewDockSite: TWinControl; DropControl: TControl = nil; ControlSide: TAlign
= alNone): Boolean;
```

C++

```
__fastcall Boolean ManualDock(TWinControl * NewDockSite, TControl * DropControl = nil, TAlign
ControlSide = alNone);
```

Description

Use ManualDock to dock the control programmatically. ManualDock undocks the control from its current dock site, then docks the control to its new dock site.

NewDockSite is the control's new dock site.

DropControl is the control in the new dock site, if any, on which to drop the control. For example, when docking to a page control, DropControl would be a tab sheet.

ControlSide specifies on which side of DropControl or NewDockSite (if DropControl is nil (Delphi) or NULL (C++)) the control should be docked. This value can be obtained by calling the dock site's GetDockEdge (see page 5763) method.

See Also

ManualFloat (see page 5772)

GetDockEdge (see page 5763)

32.1.2.5.64 TControl.ManualFloat Method

Undocks the control.

Pascal

```
function ManualFloat(ScreenPos: TRect): Boolean;
```

C++

```
__fastcall Boolean ManualFloat(TRect ScreenPos);
```

Description

Use ManualFloat to undock the control programmatically.

ScreenPos is the rectangle in which the control is displayed when it starts floating.

C++ Examples:

```
{
This example shows how to undock a dockable object.  This example
is part of the Docking Demo.
}

void __fastcall TTabDockHost::FormClose(TObject *Sender, TCloseAction &Action)
{
  TRect ARect;
  TPoint point;
  if (PageControl1->DockClientCount == 1)
  {
    point =  PageControl1->DockClients[0]->ClientToScreen(Point(0, 0));
    ARect.left = point.x;
    ARect.top = point.y;
    point = PageControl1->DockClients[0]->ClientToScreen(
      Point(PageControl1->DockClients[0]->UndockWidth,
PageControl1->DockClients[0]->UndockHeight));
    ARect.right = point.x;
    ARect.bottom = point.y;
    PageControl1->DockClients[0]->ManualFloat(ARect);
    Action = caFree;
  }
  else
    Action = caHide;
}
```

Delphi Examples:

```
procedure TTabDockHost.FormClose(Sender: TObject;
  var Action: TCloseAction);
var
  ARect: TRect;
begin
  if PageControl1.DockClientCount = 1 then
  begin
    with PageControl1.DockClients[0] do
    begin
      ARect.TopLeft := ClientToScreen(Point(0, 0));
      ARect.BottomRight := ClientToScreen(Point(UndockWidth, UndockHeight));
      ManualFloat(ARect);
    end;
    Action := caFree;
  end else
    Action := caHide;
end;
```

See Also

ManualDock (see page 5771)

Floating (see page 5823)

32.1.2.5.65 TControl.MouseActivate Method

Pascal

```
function MouseActivate(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer;
HitTest: Integer): TMouseActivate; dynamic;
```

C++

```
__fastcall TMouseActivate MouseActivate(TMouseButton Button, TShiftState Shift, int X, int Y,
int HitTest);
```

Description

This is MouseActivate, a member of class TControl.

32.1.2.5.66 TControl.MouseDown Method

OnMouseDown (see page 5862) event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); dynamic;
```

C++

```
__fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown (see page 5862) event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove (see page 5773)

MouseUp (see page 5774)

OnMouseDown (see page 5862)

32.1.2.5.67 TControl.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); dynamic;
```

C++

```
__fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove (see page 5865) event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown (see page 5773)

MouseUp (see page 5774)

OnMouseMove (see page 5865)

32.1.2.5.68 TControl.MouseUp Method

OnMouseUp (see page 5867) event dispatcher.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); dynamic;
```

C++

```
__fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown (see page 5773) method to provide other responses in addition to calling the OnMouseDown (see page 5862) event handler when the user releases a previously pressed mouse button while the cursor's hotspot is over the control.

A control calls MouseUp in response to any of the Windows mouse-up messages (WM_LBUTTONUP, WM_MBUTTONUP, WM_RBUTTONUP), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

See Also

MouseDown (see page 5773)

MouseMove (see page 5773)

OnMouseUp (see page 5867)

32.1.2.5.69 TControl.MouseWheelHandler Method

Dispatches messages received from a mouse wheel.

Pascal

```
procedure MouseWheelHandler(var Message: TMessage); dynamic;
```

C++

```
__fastcall MouseWheelHandler(TMessage Message);
```

Description

The WndProc (see page 5791) method calls MouseWheelHandler when it receives a mouse wheel message. MouseWheelHandler checks whether the windowed control is embedded in a form, and if so, calls the form's MouseWheelHandler method. This allows the form to respond to mouse wheel messages before they are handled by individual controls. If the windowed control is not in a form, MouseWheelHandler passes the mouse wheel message to the Perform (see page 5776) method, which generates an OnMouseWheel (see page 5869) event, followed (if necessary) by an OnMouseWheelUp (see page 5870) or OnMouseWheelDown (see page 5869) event.

See Also

MouseWheelHandler (see page 13881)

OnMouseWheel (see page 5869)

Perform (see page 5776)

WndProc (see page 6187)

OnMouseWheelUp (see page 5870)

OnMouseWheelDown (see page 5869)

32.1.2.5.70 TControl.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl (see page 5724) to update its PopupMenu (see page 5835) property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

32.1.2.5.71 TControl.PaletteChanged Method

Responds to changes in the system's palette by realizing the control's palette.

Pascal

```
function PaletteChanged(Foreground: Boolean): Boolean; dynamic;
```

C++

```
__fastcall Boolean PaletteChanged(Boolean Foreground);
```

Description

Override PaletteChanged to change how the control responds to Windows notifications that the system palette is changing. TControl (see page 5724) responds to these notifications by trying to realize the control's palette, if any, into the current device context. If the change in the system palette causes the actual value of the control's palette to change, PaletteChanged invalidates the control so that it can repaint with the new palette.

Windows paints the active window with a foreground palette, while other windows are forced to use background palettes. Background palettes are approximate matches to the colors specified by the control's logical palette, given the limitations imposed by implementing the foreground palette. Windows only allows a single foreground palette.

PaletteChanged allows the control to obtain a new realization of its palette when the system palette changes. When Foreground is true, the form has been activated and the control is specifying the new foreground palette. When Foreground is false, another application or form has changed the foreground palette, and controls that are sensitive to the available palette should realize new background palettes to best match their logical palettes.

If the control does not have a logical palette to be realized into the current device context, GetPalette (see page 5765) returns a handle of 0, and PaletteChanged returns false, indicating that no palette was realized. If GetPalette (see page 5765) returns the handle to a logical palette, PaletteChanged realizes this palette into the current device context, and returns true.

Note: PaletteChanged is called only when the runtime video mode requires palette support, such as for 256 color mode, but not 16 million color mode.

See Also

GetDeviceContext (see page 5760)

GetPalette (see page 5765)

32.1.2.5.72 TControl.ParentToClient Method

Translate parent coordinates to client coordinates.

Pascal

```
function ParentToClient(const Point: TPoint; AParent: TWinControl = nil): TPoint;
```

C++

```
__fastcall TPoint ParentToClient(const TPoint Point, TWinControl * AParent = nil);
```

Description

ParentToClient translates Point from the AParent's coordinate system to the current control's coordinate system. AParent must be a control that can be found by iterating up the chain of Parent (see page 5832) properties. If AParent is nil (Delphi) or NULL (C++), the control's immediate parent is used.

An EInvalidOperation exception is thrown if AParent is not an ancestor control or the Parent (see page 5832) property is nil (Delphi) or NULL (C++).

See Also

ClientToParent (see page 5744)

32.1.2.5.73 TControl.Perform Method

Responds as if the control received a specified Windows message.

Pascal

```
function Perform(Msg: Cardinal; WParam: Longint; LParam: Longint): Longint;
```

C++

```
__fastcall Longint Perform(unsigned Msg, Longint WParam, Longint LParam);
```

Description

Call Perform to bypass the Windows message queue and send a message directly to the control's window procedure.

Perform fills a message record (of type TMessage) with the message ID passed in the Msg parameter, the message parameters passed in WParam and LParam, and a result field of zero. Perform then passes the message record to the WindowProc (see page 5872) method for processing.

C++ Examples:

```
/*
The following example demonstrates the use of the Perform method
to send a windows message. The EM_SCROLLCARET message scrolls the
caret in the rich edit into view. The caret position is set to the
contents of the mask edit prior to calling Perform.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    RichEdit1->SelStart = StrToInt(MaskEdit1->Text);
    Edit1->Text = IntToStr(RichEdit1->SelStart);
    RichEdit1->Perform(EM_SCROLLCARET, 0, 0);
    RichEdit1->SetFocus();
}
```

Delphi Examples:

```
{
The following example demonstrates the use of the Perform method
to send a windows message. The EM_SCROLLCARET message scrolls the
caret in the rich edit into view. The caret position is set to the
contents of the mask edit prior to calling Perform.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
with RichEdit1 do
  Begin
    SelStart := StrToInt(MaskEdit1.Text);
    Edit1.Text := InttoStr(SelStart);
    RichEdit1.Perform(EM_SCROLLCARET, 0, 0);
    RichEdit1.SetFocus;
  end;
end;
```

See Also

WindowProc (see page 5872)

TMessage

32.1.2.5.74 TControl.PositionDockRect Method

Positions the docking rectangle according to the current drag-and-dock state.

Pascal

```
procedure PositionDockRect(DragDockObject: TDragDockObject); dynamic;
```

C++

```
__fastcall PositionDockRect(TDragDockObject DragDockObject);
```

Description

Use PositionDockRect to adjust the docking rectangle of the drag-and-dock object specified by DragDockObject. DragDockObject's DockRect property is updated to reflect the current drag-and-dock state. This updating process makes use of the UndockWidth (see page 5846), UndockHeight (see page 5845), LRDockWidth (see page 5831), or TBDockHeight (see page 5842) property, as appropriate.

See Also

AdjustDockRect (see page 6012)

UndockWidth (see page 5846)

UndockHeight (see page 5845)

LRDockWidth (see page 5831)

TBDockHeight (see page 5842)

DockRect (see page 6015)

PositionDockRect (see page 6242)

32.1.2.5.75 TControl.ReadState Method

Prepares the control for having its properties assigned values read from a stream.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

ReadState is part of a sequence of calls used by the streaming system that loads and saves component state. It reads the values of all the control's published properties, stored data, and owned components from the reader object passed in Reader.

Override ReadState to change the preparations the control makes as it readies itself to have its property values assigned from a stream. For example, a control might destroy temporary internal data structures or objects before new instances are loaded from the stream. Remember to include a call to the inherited class's method when overriding ReadState.

As implemented in TControl (see page 5724), ReadState assigns the Reader parameter's Parent (see page 5832) property to the control's own Parent (see page 5832) property before calling the inherited ReadState method of TComponent. Finally, ReadState synchronizes the Font (see page 5823), Color (see page 5812), and ShowHint (see page 5840) properties with the parent control, as indicated by the ParentFont (see page 5834), ParentColor (see page 5834), and ParentShowHint (see page 5835) properties, respectively.

See Also

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

ReadState

TReader (see page 2933)

32.1.2.5.76 TControl.Refresh Method

Repaints the control on the screen.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh method to repaint the control immediately. Refresh calls the Repaint (see page 5779) method. Use the Refresh and Repaint (see page 5779) methods interchangeably.

See Also

Invalidate (see page 5770)

Parent (see page 5832)

Repaint (see page 5779)

Update (see page 5789)

32.1.2.5.77 TControl.Repaint Method

Forces the control to repaint its image on the screen.

Pascal

```
procedure Repaint; virtual;
```

C++

```
virtual __fastcall Repaint();
```

Description

Call Repaint to force the control to repaint its image immediately. If the ControlStyle (see page 5814) property includes csOpaque, the control paints itself directly. Otherwise, the Repaint method calls the Invalidate (see page 5770) method and then the Update (see page 5789) method so that any visible portions of controls beneath the control will be repainted as well.

See Also

ControlStyle (see page 5814)

GetDeviceContext (see page 5760)

Invalidate (see page 5770)

Parent (see page 5832)

Update (see page 5789)

32.1.2.5.78 TControl.ReplaceDockedControl Method

Docks the control in the place where another control is already docked.

Pascal

```
function ReplaceDockedControl(Control: TControl; NewDockSite: TWinControl; DropControl:
TControl; ControlSide: TAlign): Boolean;
```

C++

```
__fastcall Boolean ReplaceDockedControl(TControl * Control, TWinControl * NewDockSite,
TControl * DropControl, TAlign ControlSide);
```

Description

Call ReplaceDockedControl to replace a control that is already docked while moving that control to another dock site.

Control is the control whose place this control is taking.

NewDockSite is the new dock site for the control that is replaced.

DropControl is a control in the new dock site on which to drop the control that is replaced. For example, if NewDockSite were a page control, DropControl would be a tab sheet.

ControlSide specifies on which side of DropControl or NewDockSite (if DropControl is nil (Delphi) or NULL (C++)) the replaced control should be docked. This value can be obtained by calling the dock site's GetDockEdge (see page 5763) method.

ReplaceDockedControl does the same thing as calling the ManualDock (see page 5771) method of the control specified by the Control parameter (using the values of NewDockSite, DropControl, and ControlSide) in addition to calling ManualDock (see page 5771) for the control whose method this is, docking it in the current position of the Control parameter. However, calling ReplaceDockedControl is more efficient, and prevents unnecessary flicker.

See Also

ManualDock (see page 5771)

32.1.2.5.79 TControl.RequestAlign Method

Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.

Pascal

```
procedure RequestAlign; virtual;
```

C++

```
virtual __fastcall RequestAlign();
```

Description

Controls (see page 5711) call RequestAlign internally when changes are made to the size or position of the control, so that they remain properly aligned within the parent control. Do not call RequestAlign in application code.

See Also

Align (see page 5798)

Parent (see page 5832)

32.1.2.5.80 TControl.Resize Method

Respond to control resize.

Pascal

```
procedure Resize; dynamic;
```

C++

```
__fastcall Resize();
```

Description

Resize is called automatically immediately after the control's dimensions change. As implemented in TControl (see page 5724), Resize calls the control's OnResize (see page 5870) event handler if it is assigned.

See Also

OnResize (see page 5870)

32.1.2.5.81 TControl.ScaleConstraints Method

Pascal

```
procedure ScaleConstraints(M: Integer; D: Integer);
```

C++

```
__fastcall ScaleConstraints(int M, int D);
```

Description

This is ScaleConstraints, a member of class TControl.

32.1.2.5.82 TControl.ScreenToClient Method

Converts the screen coordinates of a specified point on the screen to client coordinates.

Pascal

```
function ScreenToClient(const Point: TPoint): TPoint;
```

C++

```
__fastcall TPoint ScreenToClient(const TPoint Point);
```

Description

Use ScreenToClient to convert a point in screen coordinates to local, or client area, coordinates. In client area coordinates (0, 0) corresponds to the upper left corner of the control's client area. In screen coordinates (0, 0) corresponds to the upper left corner of the screen.

Use ScreenToClient along with ClientToScreen (see page 5744) to convert from one control's coordinate system to another control's coordinate system. For example,

```
P := TargetControl.ScreenToClient(SourceControl.ClientToScreen(P));
P = TargetControl->ScreenToClient(SourceControl->ClientToScreen(P));
```

converts P from coordinates in SourceControl to coordinates in TargetControl.

See Also

ClientToScreen (see page 5744)

32.1.2.5.83 TControl.SendCancelMode Method

Cancels the modal state of the control.

Pascal

```
procedure SendCancelMode(Sender: TControl);
```

C++

```
__fastcall SendCancelMode(TControl * Sender);
```

Description

Call SendCancelMode to cancel the modal state of the control. Various controls implemented in the VCL support a model state where the user must respond to the control before interacting with any other object on the form. An example is the dropped-down state of a combo box when the drop-down list is showing. For these controls, specific user actions send a Windows message to the control to end the model state, such as a mouse click that closes up the dropped-down list.

SendCancelMode allows an application to end the modal state of the control in the absence of a user action, or when implementing a new user interface response.

See Also

Perform (see page 5776)

DroppedDown

32.1.2.5.84 TControl.SendDockNotification Method

Sends a Windows message to the control's host dock site.

Pascal

```
procedure SendDockNotification(Msg: Cardinal; WParam: Integer; LParam: Integer);
```

C++

```
__fastcall SendDockNotification(unsigned Msg, int WParam, int LParam);
```

Description

Use SendDockNotification to send a Windows message to the control's host dock site.

SendDockNotification is called automatically when the control's text or visibility changes.

See Also

WindowProc (see page 5872)

TMessage

WndProc (see page 6187)

32.1.2.5.85 TControl.SendToBack Method

Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.

Pascal

```
procedure SendToBack;
```

C++

```
__fastcall SendToBack();
```

Description

Use SendToBack to change the order of overlapping controls or forms.

The order in which controls stack on top of each other (also called the Z order) depends on the order the controls are placed on the form. For example, if you put a label and an image on a form so that one is on top of the other, the one that was placed first

on the form becomes the one on the bottom. Because both the label and the image are non-windowed controls, they "stack" as you would expect them to. Call the SendToBack method for the top object to move it below the other object.

The stacking order of two windowed controls is the same as the stacking of two non-windowed controls. For example, if you put a memo on a form, then put a check box on top of it, the check box remains on top. Calling SendToBack for the check box makes the memo appear on top.

The stacking order of windowed and non-windowed controls cannot be mingled. For example, if you put a memo, a windowed control, on a form, and then put a label, a non-windowed control, on top of it, the label disappears behind the memo. Windowed controls always stack on top of non-windowed controls. In this example, calling the SendToBack method of the memo does nothing, the label remains behind the memo.

If the control has the input focus when the SendToBack method executes, it loses the input focus.

See Also

BringToFront (see page 5740)

32.1.2.5.86 TControl.SetAutoSize Method

Pascal

```
procedure SetAutoSize(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetAutoSize(Boolean Value);
```

Description

This is SetAutoSize, a member of class TControl.

32.1.2.5.87 TControl.SetBiDiMode Method

Sets the control's bi-directional mode.

Pascal

```
procedure SetBiDiMode(Value: TBiDiMode); virtual;
```

C++

```
virtual __fastcall SetBiDiMode(TBiDiMode Value);
```

Description

SetBiDiMode is the write implementation of the control's BiDiMode (see page 5803) property. Override this method to change the implementation of BiDiMode (see page 5803).

See Also

BiDiMode (see page 5803)

32.1.2.5.88 TControl.SetBounds Method

Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer); virtual;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to change all of the component's boundary properties at one time. The same effect can be achieved by setting the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties separately, but SetBounds changes all four properties at once ensuring that the control will not repaint between changes.

Specify the values for the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties as the value of the ALeft, ATop, AWidth, and AHeight parameters, respectively.

Calling SetBounds does not necessarily result in the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties changing to the specified values. Before the properties are changed, the AutoSize (see page 5803) or Constraints (see page 5812) property may limit the changes, and an OnCanResize (see page 5851) (or OnConstrainedResize (see page 5853)) event handler may change the new values. After the control's Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties are changed, SetBounds generates an OnResize (see page 5870) event.

Note: Component writers can change the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties while bypassing all resize events and constraint or autosize logic by using the UpdateBoundsRect (see page 5789) method instead.

See Also

Height (see page 5825)

Left (see page 5830)

Top (see page 5844)

Width (see page 5849)

BoundsRect (see page 5803)

AutoSize (see page 5803)

Constraints (see page 5812)

OnCanResize (see page 5851)

OnConstrainedResize (see page 5853)

OnResize (see page 5870)

UpdateBoundsRect (see page 5789)

32.1.2.5.89 TControl.SetDesignVisible Method

Makes a control visible at design time.

Pascal

```
procedure SetDesignVisible(Value: Boolean); dynamic;
```

C++

```
__fastcall SetDesignVisible(Boolean Value);
```

Description

SetDesignVisible is used internally by Delphi during form design.

See Also

TComponentState

32.1.2.5.90 TControl.SetDragMode Method

Sets the drag mode for the control.

Pascal

```
procedure SetDragMode(Value: TDragMode); virtual;
```

C++

```
virtual __fastcall SetDragMode(TDragMode Value);
```

Description

SetDragMode is the protected write implementation for the DragMode (see page 5819) property. Override SetDragMode to execute additional code when the value of the DragMode (see page 5819) property changes.

See Also

DragMode (see page 5819)

32.1.2.5.91 TControl.SetEnabled Method

Sets the value of the Enabled (see page 5821) property.

Pascal

```
procedure SetEnabled(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetEnabled(Boolean Value);
```

Description

SetEnabled is the protected write implementation of the Enabled (see page 5821) property. After changing the internal representation of the Enabled (see page 5821) property, SetEnabled sends a Windows message back to the control so that it can respond to the change. When TControl (see page 5724) receives this message, it invalidates the control so that its appearance reflects the change (for example, so that it appears grayed). Some descendants respond to this message in other ways (for example, windowed controls ensure that a disabled control does not have focus).

Override GetEnabled (see page 5763) and SetEnabled to change the implementation of the Enabled (see page 5821) property. For example, A descendant of TControl (see page 5724) might change the way the Enabled (see page 5821) property interacts with an associated action. When overriding SetEnabled, be sure to call the inherited method.

See Also

GetEnabled (see page 5763)

Enabled (see page 5821)

Invalidate (see page 5770)

32.1.2.5.92 TControl.SetName Method

Sets the name of the control.

Pascal

```
procedure SetName(const Value: TComponentName); override;
```

C++

```
virtual __fastcall SetName(const TComponentName Value);
```

Description

SetName is the protected write implementation of the Name property. TControl (see page 5724) updates the Text (see page 5842) or Caption (see page 5806) property, if appropriate, after changing the value of Name.

Override SetName to execute additional code when the value of the Name property changes.

See Also

Name

Caption (see page 5806)

Text (see page 5842)

32.1.2.5.93 TControl.SetParent Method

Sets the parent of the control.

Pascal

```
procedure SetParent(AParent: TWinControl); virtual;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent (see page 5832) property. Override SetParent to execute additional code when the value of the Parent (see page 5832) property changes.

The AParent parameter specifies the new parent of the control.

If the control already has a parent, SetParent removes the control from that parent's list of controls by calling the parent's RemoveControl method. If AParent is not nil (Delphi) or NULL (C++), SetParent calls its InsertControl method to add the control to its list of controls.

The TControl (see page 5724) destructor calls SetParent(nil (Delphi) or NULL (C++)) to remove the control from its parent's control list before destroying the component.

See Also

Destroy (see page 5732)

InsertControl (see page 6160)

Parent (see page 5832)

RemoveControl (see page 6172)

32.1.2.5.94 TControl.SetParentBiDiMode Method

Specifies whether the control uses its parent's BiDiMode (see page 5803).

Pascal

```
procedure SetParentBiDiMode(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetParentBiDiMode(Boolean Value);
```

Description

SetParentBiDiMode is the protected write implementation of the ParentBiDiMode (see page 5833) property.

See Also

ParentBiDiMode (see page 5833)

32.1.2.5.95 TControl.SetParentComponent Method

Sets the parent for the control.

Pascal

```
procedure SetParentComponent(Value: TComponent); override;
```

C++

```
virtual __fastcall SetParentComponent(TComponent * Value);
```

Description

Do not call the SetParentComponent method. It is used by the streaming system that loads and saves VCL components. SetParentComponent makes the component specified by the Value parameter the parent of the control.

SetParentComponent calls the SetParent (see page 5786) method only if the component specified as the parent is a windowed control.

See Also

HasParent (see page 5768)

SetParent (see page 5786)

GetParentComponent (see page 5765)

SetParentComponent

32.1.2.5.96 TControl.SetTextBuf Method

Sets the text of the control.

Pascal

```
procedure SetTextBuf(Buffer: PChar);
```

C++

```
__fastcall SetTextBuf(const char * Buffer);
```

Description

Call the SetTextBuf method to set the text of the control to the text in the specified buffer.

The text is set to the text in the buffer pointed to by the Buffer parameter. Buffer must point to a null-terminated string. The SetTextBuf method sends the WM_SETTEXT and CM_TEXTCHANGED (see page 6349) messages. If done in the tight execution of a function, this could result in the change not being seen until messages from the Windows message queue being processed after the function finishes executing. To interrupt a function's execution and process messages, use the ProcessMessages method of the Application object.

Note: The SetTextBuf method is provided for backwards compatibility only. To set the text of a control now, assign a string

value to the Text (see page 5842) property.

See Also

Text (see page 5842)

application

ProcessMessages (see page 13691)

32.1.2.5.97 TControl.SetZOrder Method

Repositions the control in its parent's control list, thereby changing position onscreen.

Pascal

```
procedure SetZOrder(TopMost: Boolean); dynamic;
```

C++

```
__fastcall SetZOrder(Boolean TopMost);
```

Description

Call SetZOrder to change the z-order of the control. To make the control the topmost control, specify the TopMost value as true. To make the control the bottommost, set TopMost to false.

After changing the order, SetZOrder invalidates the control to ensure repainting to reflect the new order.

The stacking order of windowed and non-windowed controls cannot be mingled. For example, if you put a memo, a windowed control, on a form, and then put a label, a non-windowed control, on top of it, the label disappears behind the memo. Windowed controls always stack on top of non-windowed controls. In this example, calling the SetZOrder method of the label or the memo has no effect, the label always remains behind the memo.

See Also

BringToFront (see page 5740)

Invalidate (see page 5770)

SendToBack (see page 5782)

32.1.2.5.98 TControl.Show Method

Makes a control visible.

Pascal

```
procedure Show;
```

C++

```
__fastcall Show();
```

Description

Use the Show method to make visible a control that was previously hidden.

The Show method sets the control's Visible (see page 5847) property to true and ensures that the parent control is also visible.

See Also

Hide (see page 5769)

Visible (see page 5847)

32.1.2.5.99 TControl.Update Method

Processes any pending paint messages immediately.

Pascal

```
procedure Update; virtual;
```

C++

```
virtual __fastcall Update();
```

Description

Call Update to force the control to be repainted before any more, possibly time-consuming, processing takes place. Use Update to provide immediate feedback to the user that cannot wait for the Windows paint message to arrive.

Update does not invalidate the control, but simply forces a repaint of any regions that have already been invalidated. Call Repaint (see page 5779) instead to invalidate the control as well.

See Also

Invalidate (see page 5770)

Refresh (see page 5779)

Repaint (see page 5779)

32.1.2.5.100 TControl.UpdateBoundsRect Method

Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.

Pascal

```
procedure UpdateBoundsRect(const R: TRect);
```

C++

```
__fastcall UpdateBoundsRect(const TRect R);
```

Description

UpdateBoundsRect is used internally by controls to stay synchronized with changes made by the underlying Windows screen object. UpdateBoundsRect updates the properties to match those passed in the rectangle as the R parameter.

See Also

BoundsRect (see page 5803)

SetBounds (see page 5783)

32.1.2.5.101 TControl.UpdateExplicitBounds Method

Pascal

```
procedure UpdateExplicitBounds;
```

C++

```
__fastcall UpdateExplicitBounds();
```

Description

This is UpdateExplicitBounds, a member of class TControl.

32.1.2.5.102 TControl.UseRightToLeftAlignment Method

Specifies whether the control's alignment should be reversed right-to-left.

Pascal

```
function UseRightToLeftAlignment: Boolean; dynamic;
```

C++

```
__fastcall Boolean UseRightToLeftAlignment();
```

Description

Controls (see page 5711) call UseRightToLeftAlignment when implementing the BiDiMode (see page 5803) property. UseRightToLeftAlignment indicates whether the control's alignment should be reversed right-to-left. This information can be used when painting the control or when responding to mouse events.

UseRightToLeftAlignment returns true if the application is running on a system with a Middle Eastern locale and the BiDiMode (see page 5803) property is bdRightToLeft. Otherwise, it returns false. Descendants override UseRightToLeftAlignment if alignment in Middle Eastern locales depends on other factors.

See Also

BiDiMode (see page 5803)

GetControlsAlignment (see page 5759)

Text (see page 5842)

Caption (see page 5806)

32.1.2.5.103 TControl.UseRightToLeftReading Method

Specifies whether the control is using a right-to-left reading order.

Pascal

```
function UseRightToLeftReading: Boolean;
```

C++

```
__fastcall Boolean UseRightToLeftReading();
```

Description

Controls (see page 5711) call UseRightToLeftReading when implementing the BiDiMode (see page 5803) property. UseRightToLeftReading indicates whether the control's text should read from right to left. This information can be used to determine which style flags to set for the control when it is created and which flags to set for drawing text.

UseRightToLeftReading returns true if the application is running on a system with a Middle Eastern locale and the BiDiMode (see page 5803) property is not bdLeftToRight. UseRightToLeftReading returns falseotherwise.

See Also

BiDiMode (see page 5803)

Text (see page 5842)

Caption (see page 5806)

DrawTextBiDiModeFlags (see page 5756)

DrawTextBiDiModeFlagsReadingOnly (see page 5756)

32.1.2.5.104 TControl.UseRightToLeftScrollBar Method

Specifies whether the vertical scroll bar appears on the left side of the control.

Pascal

```
function UseRightToLeftScrollBar: Boolean;
```

C++

```
__fastcall Boolean UseRightToLeftScrollBar();
```

Description

Controls (see page 5711) call UseRightToLeftScrollBar when implementing the BiDiMode (see page 5803) property. UseRightToLeftScrollBar returns true when the vertical scroll bar appears on the left side of the control, and false when it appears on the right.

UseRightToLeftScrollBar returns true if the application is running on a system with a Middle Eastern locale and the BiDiMode (see page 5803) property is bdRightToLeft or bdRightToLeftNoAlign.

See Also

BiDiMode (see page 5803)

32.1.2.5.105 TControl.VisibleChanging Method

Provides the interface for a method that responds to changes in the control's visibility.

Pascal

```
procedure VisibleChanging; dynamic;
```

C++

```
__fastcall VisibleChanging();
```

Description

VisibleChanging is called before the value of a control's Visible (see page 5847) property changes. As implemented in TControl (see page 5724), it does nothing. Override VisibleChanging to respond to changes in the control's visibility. For example, if a control should not change its visibility, override VisibleChanging to raise an exception when an attempt is made to alter the control's visibility.

See Also

Visible (see page 5847)

32.1.2.5.106 TControl.WndProc Method

Provides specific message responses for the control.

Pascal

```
procedure WndProc(var Message: TMessage); virtual;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

Override WndProc to change the initial Windows message handler for the control. The WindowProc (see page 5872) property is initialized to WndProc.

WndProc implements the mouse behavior specified by the ControlStyle (see page 5814) and DragMode (see page 5819) properties, and updates the ControlState (see page 5813) property. It passes any unhandled messages on to the Dispatch method.

When overriding WndProc to provide specialized responses to messages, be sure to call the inherited WndProc at the end to dispatch any unhandled messages.

C++ Examples:

```
/*
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
*/
class TMyListBoxDescendant : public TListBox
{
__published:    // IDE-managed Components
    void __fastcall SubClassWndProc(Messages::TMessage &Message);
    void __fastcall ToggleSubClass(bool On);
    void __fastcall OnDrawItemProc(
      TWinControl *Control, int Index, const TRect &Rect,
      TOwnerDrawState State);
private:    // User declarations
public:     // User declarations
    __fastcall TMyListBoxDescendant(TComponent* Owner);
};

TForm1 *Form1;
TMyListBoxDescendant *MyListBoxDescendant1;
Graphics::TBitmap *bitmap0;

const WM_STYLEMESSAGE = WM_USER + 2000;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

__fastcall TMyListBoxDescendant::TMyListBoxDescendant(TComponent* Owner)
    : TListBox(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
}

void __fastcall TForm1::SubClassRadioGroup1Click(TObject *Sender)
{
```

```
  MyListBoxDescendant1->ToggleSubClass(SubClassRadioGroup1->ItemIndex == 0);
}

void __fastcall TMyListBoxDescendant::SubClassWndProc(Messages::TMessage &Message)
{
  if (Message.Msg == WM_STYLEMESSAGE)
    Style = (TListBoxStyle)Message.WParam;
  else
    WndProc(Message);
}

void __fastcall TMyListBoxDescendant::ToggleSubClass(bool On)
{
  if (On)
    WindowProc = SubClassWndProc;
  else
    WindowProc = WndProc;
}

void __fastcall TMyListBoxDescendant::OnDrawItemProc(TWinControl *Control, int Index,
      const TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *bitmap; // temporary variable for item's bitmap
  int Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *canvas = ((TListBox *)Control)->Canvas;
  canvas->FillRect(Rect); // clear the rectangle
  bitmap = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
  if (bitmap)
  {
    canvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, bitmap->Width, bitmap->Height),
      bitmap, Bounds(0, 0, bitmap->Width, bitmap->Height), clRed); // render bitmap
    Offset += bitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  canvas->TextOut(Rect.Left + Offset, Rect.Top, ((TListBox *)Control)->Items->Strings[Index]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyListBoxDescendant1 = new TMyListBoxDescendant(Form1);
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Parent = Form1;
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Left =
    SubClassRadioGroup1->Left + SubClassRadioGroup1->Width + 30;;
  MyListBoxDescendant1->Top = SubClassRadioGroup1->Top;
  MyListBoxDescendant1->Height = SubClassRadioGroup1->Height;
  MyListBoxDescendant1->OnDrawItem =
    MyListBoxDescendant1->OnDrawItemProc;

  bitmap0 = new Graphics::TBitmap;
  ImageList1->GetBitmap(0, bitmap0);
  MyListBoxDescendant1->Items->AddObject("Butterfly", bitmap0);

  SubClassRadioGroup1->Items->Add("SubClassWndProc");
  SubClassRadioGroup1->Items->Add("WndProc");
  SubClassRadioGroup1->ItemIndex = 2;
}
```

Delphi Examples:

```
{
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
```

```
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
}

type
  TMyListBoxDescendant = class(TlistBox)
    procedure SubClassWndProc(var Message: TMessage);
    procedure ToggleSubClass(On: Boolean);
    procedure OnDrawItemProc(
      Control: TWinControl;
      Index: Integer;
      Rect:TRect;
      State: TOwnerDrawState);
  end;
  TForm1 = class(TForm)
    SubClassRadioGroup1: TRadioGroup;
    Button1: TButton;
    ImageList1: TImageList;
    Button2: TButton;
    procedure SubClassRadioGroup1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  MyListBoxDescendant1: TMyListBoxDescendant;
  bitmap0: TBitmap;

implementation

{$R *.dfm}

const WM_STYLEMESSAGE = WM_USER + 2000;

procedure TForm1.Button1Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
end;

procedure TMyListBoxDescendant.SubClassWndProc(var Message: TMessage);
begin
  if (Message.Msg = WM_STYLEMESSAGE) then
    Style:= TListBoxStyle(Message.WParam)
  else
    WndProc(Message);
```

```
end;

procedure TMyListBoxDescendant.ToggleSubClass(On: Boolean);
begin
  if On then
    WindowProc := SubClassWndProc
  else
    WindowProc := WndProc;
end;

procedure TForm1.SubClassRadioGroup1Click(Sender: TObject);
begin
  MyListBoxDescendant1.ToggleSubClass(
    SubClassRadioGroup1.ItemIndex = 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyListBoxDescendant1:= TMyListBoxDescendant.Create(self);
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Parent:= Form1;
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Left:=
    SubClassRadioGroup1.Left + SubClassRadioGroup1.Width + 30;;
  MyListBoxDescendant1.Top:= SubClassRadioGroup1.Top;
  MyListBoxDescendant1.Height:= SubClassRadioGroup1.Height;
  MyListBoxDescendant1.OnDrawItem:=
    MyListBoxDescendant1.OnDrawItemProc;

  bitmap0 := TBitmap.Create;
  ImageList1.GetBitmap(0, bitmap0);
  MyListBoxDescendant1.Items.AddObject('Butterfly', bitmap0);

  SubClassRadioGroup1.Items.Add('SubClassWndProc');
  SubClassRadioGroup1.Items.Add('WndProc');
  SubClassRadioGroup1.ItemIndex := 2;
end;

procedure TMyListBoxDescendant.OnDrawItemProc(
  Control: TWinControl;
  Index: Integer;
  Rect:TRect;
  State: TOwnerDrawState);
var
  Bitmap: TBitmap;      { temporary variable for the item’s bitmap }
  Offset: Integer;      { text offset width }
begin
  { draw on control canvas, not on the form }
  with (Control as TListBox).Canvas do
  begin
    FillRect(Rect);       { clear the rectangle }
    Offset := 2;          { provide default offset }
    Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
    if Bitmap <> nil then
    begin
      BrushCopy(
        Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
        Bitmap,
        Bounds(0, 0, Bitmap.Width, Bitmap.Height),
        clRed);  {render bitmap}
      Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
    end;
    TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
text }
  end;
end;
```

See Also

ControlState (see page 5813)

ControlStyle (see page 5814)

DragMode (see page 5819)

Dispatch

WindowProc (see page 5872)

32.1.2.6 TControl Properties

The properties of the TControl class are listed here.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Constraints (see page 5812)	Specifies the size constraints for the control.

	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Parent (see page 5832)	Indicates the parent of the control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

32.1.2.6.1 TControl.Action Property

Designates the action associated with the control.

Pascal

```
property Action: TBasicAction;
```

C++

```
__property TBasicAction Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick (see page 5851) event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

32.1.2.6.2 TControl.ActionLink Property

Contains the action link object associated with the control.

Pascal

```
property ActionLink: TControlActionLink;
```

C++

```
__property TControlActionLink ActionLink;
```

Description

ActionLink handles the association between the control and the action specified by the Action (see page 5797) property. The action link determines which properties and events of the action object determine the values of corresponding properties and events of the control.

Most applications do not use the ActionLink property. Component writers can change this property in TControl (see page 5724) descendants to be a custom descendant of TControl (see page 5724) ActionLink that handles a desired configuration of properties and events.

See Also

TBasicAction (see page 2707)

Action (see page 5797)

32.1.2.6.3 TControl.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors (see page 5802) property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
```

```
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

```
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

32.1.2.6.4 TControl.AlignWithMargins Property

Indicates whether a control should be constrained by margins.

Pascal

```
property AlignWithMargins: Boolean;
```

C++

```
__property Boolean AlignWithMargins;
```

Description

If AlignWithMargins is true, use the Margins (see page 5831) property of the control to govern the spacing relative to other controls that are aligned with this one. The controls are not allowed to be any closer than the spacing specified in Margins (see page 5831). This spacing is maintained as controls are moved when the parent control resizes.

AlignWithMargins is **true** if the style **csAlignWithMargins**, a TControlStyle (see page 6303) type, is in the control's ControlStyle (see page 5814) property and false otherwise.

See Also

TMargins (see page 6087)

ControlStyle (see page 5814)

Margins (see page 5831)

TControlStyle

32.1.2.6.5 TControl.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align (see page 5798) property instead. Unlike Anchors, Align (see page 5798) allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

32.1.2.6.6 TControl.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints (see page 5812)

CanAutoSize (see page 5741)

32.1.2.6.7 TControl.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

32.1.2.6.8 TControl.BoundsRect Property

Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.

Pascal

```
property BoundsRect: TRect;
```

C++

```
__property TRect BoundsRect;
```

Description

Use BoundsRect as a quick way to obtain the pixel locations of all corners of the control all at once.

For example, the statement

```
R := Control.BoundsRect;
R = Control->BoundsRect;
```

corresponds to

```
R.Left := Control.Left;
R.Top := Control.Top;
R.Right := Control.Left + Control.Width;
R.Bottom := Control.Top + Control.Height;
R.Left = Control->Left;
R.Top = Control->Top;
R.Right = Control->Left + Control->Width;
R.Bottom = Control->Top + Control->Height;
```

The origin of the pixel coordinate system is in the top left corner of the parent window.

Note: A point is considered within the control's bounds rectangle if it lies on the left or top side but not if it lies on the right or bottom side. That is, to be inside the bounds rectangle, the x-coordinate must be greater than or equal to BoundsRect.Left (see page 5830) and less than BoundsRect.Right, and the y-coordinate must be greater than or equal to BoundsRect.Top (see page 5844) and less than BoundsRect.Bottom.

C++ Examples:

```
/*
This code resizes the active control to twice as wide and
half as high:
*/
void __fastcall TForm1::ButtonTSClick(TObject *Sender)
{
  TRect MyRect;
  if (myActiveControl == NULL) exit;
  MyRect = myActiveControl->BoundsRect;
  MyRect.Right = MyRect.Left + (MyRect.Right - MyRect.Left) / 2;
  MyRect.Bottom = MyRect.Top + 2 * (MyRect.Bottom - MyRect.Top);
  myActiveControl->BoundsRect = MyRect;
}
void __fastcall TForm1::ButtonSFClick(TObject *Sender)
{
  TRect MyRect;
  if (myActiveControl == NULL) exit;
  MyRect = myActiveControl->BoundsRect;
  MyRect.Right = MyRect.Left + 2 * (MyRect.Right - MyRect.Left);
  MyRect.Bottom = MyRect.Top + (MyRect.Bottom - MyRect.Top) / 2;
  myActiveControl->BoundsRect = MyRect;
}
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TComponent *Temp;
  TListItem *ListItem;
  ListView1->ViewStyle = vsList;
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Components: ";
```

```
  for (int I = ComponentCount - 1; I >= 0; I--)
  {
    Temp = Components[I];
    ListItem = ListView1->Items->Add();
    ListItem->Caption = Temp->Name;
  }
}
void __fastcall TForm1::ListView1SelectItem(TObject *Sender, TListItem *Item,
      bool Selected)
{
  TComponent *comp = FindComponent(Item->Caption);
  if (dynamic_cast<TWinControl *>(comp) != NULL)
    myActiveControl = (TWinControl *)comp;
}
```

Delphi Examples:

```
{
This code resizes the active control to twice as wide and
half as high:
}
procedure TForm1.ButtonSFClick(Sender: TObject);
var
  MyRect: TRect;
begin
  if (myActiveControl = nil) then exit;
  MyRect := myActiveControl.BoundsRect;
  MyRect.Right := MyRect.Left + (MyRect.Right - MyRect.Left) * 2;
  MyRect.Bottom := MyRect.Top + (MyRect.Bottom - MyRect.Top) div 2;
  myActiveControl.BoundsRect := MyRect;
end;

procedure TForm1.ButtonTSClick(Sender: TObject);
var
  MyRect: TRect;
begin
  if (myActiveControl = nil) then exit;
  MyRect := myActiveControl.BoundsRect;
  MyRect.Right := MyRect.Left + (MyRect.Right - MyRect.Left) div 2;
  MyRect.Bottom := MyRect.Top + (MyRect.Bottom - MyRect.Top) * 2;
  myActiveControl.BoundsRect := MyRect;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
  Temp: TComponent;
  ListItem: TListItem;
begin
  ListView1.ViewStyle := vsList;
  ListItem := ListView1.Items.Add;
  ListItem.Caption := 'Components: ';
  for I := ComponentCount - 1 downto 0 do
  begin
    Temp := Components[I];
    begin
      ListItem := ListView1.Items.Add;
      ListItem.Caption := Temp.Name;
    end;
  end;
end;

procedure TForm1.ListView1SelectItem(Sender: TObject; Item: TListItem;
  Selected: Boolean);
var comp: TComponent;
begin
  comp := FindComponent(Item.Caption);
```

```
  if comp is TWinControl then
    myActiveControl := TWinControl(comp);
end;
```

See Also

Height (see page 5825)

Left (see page 5830)

Top (see page 5844)

Width (see page 5849)

32.1.2.6.9 TControl.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls (see page 5711) that display text use either the Caption property or the Text (see page 5842) property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text (see page 5842) is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
```

```
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
```

```
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

32.1.2.6.10 TControl.ClientHeight Property

Specifies the height of the control's client area in pixels.

Pascal

```
property ClientHeight: Integer;
```

C++

```
__property int ClientHeight;
```

Description

Use ClientHeight to read or change the height of the control's client area.

For TControl (see page 5724), ClientHeight is the same as Height (see page 5825). Derived classes may implement a ClientHeight property that differs from Height (see page 5825). For example, the ClientHeight of a form is the value of the Height (see page 5825) property minus the height of the title bar, resize border, and scroll bars.

C++ Examples:

```
/*
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  const int iPageDelta = 10;

  if (Key == VK_NEXT)
    VertScrollBar->Position += iPageDelta;
  else if (Key == VK_PRIOR)
    VertScrollBar->Position -= iPageDelta;
}
```

Delphi Examples:

```
{
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
}
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
const
  PageDelta = 10;
begin
  With VertScrollbar do
    if Key = VK_NEXT then
      Position := Position + PageDelta
    else if Key = VK_PRIOR then
      Position := Position - PageDelta;
end;
```

See Also

ClientOrigin (see page 5809)

ClientRect (see page 5810)

ClientWidth (see page 5811)

Height (see page 5825)

32.1.2.6.11 TControl.ClientOrigin Property

Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.

Pascal

```
property ClientOrigin: TPoint;
```

C++

```
__property TPoint ClientOrigin;
```

Description

Read ClientOrigin to locate the top-left corner of the control's client area. ClientOrigin returns X and Y coordinates as a TPoint type, where X specifies the horizontal coordinate of the point and Y specifies the vertical coordinate.

The screen coordinates of a control that is descended from TControl (see page 5724) and not TWinControl (see page 6112) are the screen coordinates of the control's parent added to its Left (see page 5830) and Top (see page 5844) properties. If the control doesn't have a parent, an EInvalidOperation exception is raised when ClientOrigin is read.

See Also

ClientHeight (see page 5808)

ClientRect (see page 5810)

ClientWidth (see page 5811)

EInvalidOperation (see page 2700)

32.1.2.6.12 TControl.ClientRect Property

Specifies the size (in pixels) of a control's client area.

Pascal

```
property ClientRect: TRect;
```

C++

```
__property TRect ClientRect;
```

Description

Read ClientRect to find the size of the client area of a control. ClientRect returns a rectangle with its Top (see page 5844) and Left (see page 5830) fields set to zero, and its Bottom and Right fields set to the control's Height (see page 5825) and Width (see page 5849), respectively. ClientRect is equivalent to Rect(0, 0, ClientWidth (see page 5811), ClientHeight (see page 5808)).

A point is considered within the control's client rectangle if it lies on the left or top side but not if it lies on the right or bottom side. That is, to be inside the client rectangle, the x-coordinate must be greater than or equal to ClientRect.Left (see page 5830) and less than ClientRect.Right, and the y-coordinate must be greater than or equal to ClientRect.Top (see page 5844) and less than ClientRect.Bottom.

Note: ClientRect is the size of the physical client area of the control, not its logical client area. If the control supports scrolling, the ClientRect is not the entire scrolling range, but only the region that is available at any given time.

C++ Examples:

```
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TToolBar *myToolBar;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TToolBar")
  {
    myToolBar = (TToolBar *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
```

```
    myToolBar->Canvas->MoveTo(myToolBar->ClientRect.Left, myToolBar->ClientRect.Top);
    myToolBar->Canvas->LineTo(myToolBar->ClientRect.Right, myToolBar->ClientRect.Bottom);
    myToolBar = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TToolBar) then
  begin
    myToolBar:= (ListBox1.Items.Objects[Index] as TToolBar);
    with myToolBar.ClientRect do
    begin // make sure we are using the TControls canvas and not the from's!
      myToolBar.Canvas.MoveTo(Left, Top);
      myToolBar.Canvas.LineTo(Right, Bottom);
    end;
  myToolBar:= nil;
  end;
```

See Also

BoundsRect (see page 5803)

ClientHeight (see page 5808)

ClientWidth (see page 5811)

Height (see page 5825)

Left (see page 5830)

Top (see page 5844)

Width (see page 5849)

Rect (see page 23299)

32.1.2.6.13 TControl.ClientWidth Property

Specifies the horizontal size of the control's client area in pixels.

Pascal

```
property ClientWidth: Integer;
```

C++

```
__property int ClientWidth;
```

Description

Use ClientWidth to read or change the width of the control's client area. ClientWidth is equivalent to ClientRect.Right.

For TControl (see page 5724), ClientWidth is the same as Width (see page 5849). Derived classes may implement a ClientWidth property that differs from Width (see page 5849). For example, the ClientWidth of a form is the value of the Width (see page 5849) property minus the width of the resize border and scroll bars.

See Also

ClientHeight (see page 5808)

ClientOrigin (see page 5809)

ClientRect (see page 5810)

Width (see page 5849)

32.1.2.6.14 TControl.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor (see page 5834) property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor (see page 5834) property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

32.1.2.6.15 TControl.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight (see page 5808) when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight (see page 5808) is represented by x: `Constraints.MinHeight := x + Height (see page 5825) - ClientHeight (see page 5808)`.

Warning: Do not set up constraints that conflict with the value of the Align (see page 5798) or Anchors (see page 5802) property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

32.1.2.6.16 TControl.ControlState Property

Indicates the current state of a control at runtime.

Pascal

```
property ControlState: TControlState;
```

C++

```
__property TControlState ControlState;
```

Description

Read ControlState to find out various conditions that affect the control such as whether it has been clicked or needs alignment. ControlState reflects transient conditions of an instance of the control, as opposed to attributes of the control class as a whole. ControlState consists of a set of flags drawn from the following values:

Flag	Meaning
csLButtonDown	The left mouse button was clicked and not yet released. This is set for all mouse-down events.
csClicked	The same as csLButtonDown, but only set if ControlStyle (see page 5814) contains csClickEvents, meaning that mouse-down events are interpreted as clicks.
csPalette	The system palette has changed and the control or one of its descendants has not finished adjusting by realizing its palette.
csReadingState	The control is reading its state from a stream.

csAlignmentNeeded	The control needs to realign itself when alignment is re-enabled.
csFocusing	The application is processing messages intended to give the control focus. This does not guarantee the control will receive focus, but prevents recursive calls.
csCreating	The control and/or its owner and subcontrols are being created. This flag clears when all have finished creating.
csPaintCopy	The control is being replicated, meaning a copy of the control is being painted. The ControlStyle (see page 5814) flag csReplicatable must be set for this state to occur.
csCustomPaint	The control is processing custom paint messages.
csDestroyingHandle	The control's window is being destroyed.
csDocking	The control is being docked.

The flags in ControlState are specific to controls, and augment the state flags in the ComponentState property.

ControlState is primarily used by component writers in the implementation of components derived from TControl (see page 5724).

See Also

ControlStyle (see page 5814)

ComponentState (see page 2804)

32.1.2.6.17 TControl.ControlStyle Property

Determines style characteristics of the control.

Pascal

```
property ControlStyle: TControlStyle;
```

C++

```
__property TControlStyle ControlStyle;
```

Description

Use ControlStyle to determine various attributes of the control, such as whether the control can capture the mouse or has a fixed size. The ControlStyle property contains a set of style flags indicating these attributes. The following table lists the flags and their meanings:

Flag	Meaning
csAcceptsControls	The control becomes the parent of any controls dropped on it at design time.
csCaptureMouse	The control captures mouse events when it is clicked.
csDesignInteractive	The control maps right mouse-button clicks at design time into left mouse-button clicks to manipulate the control.
csClickEvents	The control can receive and respond to mouse clicks. Some controls may not inherit this attribute, for example, TButton.
csFramed	The control has a 3D frame.
csSetCaption	The control's Caption (see page 5806) should match the Name property if it has not been explicitly set to something else.
csOpaque	The control completely fills its client rectangle.
csDoubleClicks	The control can receive and respond to double-click messages. Otherwise, map double-clicks into clicks.
csFixedWidth	The width of the control does not vary or scale.

csFixedHeight	The height of the control does not vary or scale.
csNoDesignVisible	The control is not visible at design time.
csReplicatable	The control can be copied using the PaintTo method to draw its image to an arbitrary canvas.
csNoStdEvents	Standard events such as mouse, key, and click events are ignored. This flag allows an application to run faster when there is no need respond to these events.
csDisplayDragImage	If some control that has an associated drag image list is dragged across a control with this setting, then the image list is used to enhance the drag cursor while the cursor is over it. Otherwise, the drag cursor is used on its own.
csReflector	The control responds to Windows dialog messages, focus messages, or size change messages. Use this setting if the control can be used as an ActiveX control, so that it receives notification of these events.
csActionClient	The control is linked to an action object. This flag is set when the Action (see page 5797) property is set, and cleared if the Action (see page 5797) property is cleared.
csMenuEvents	The control responds to system menu commands.

ControlStyle describes all controls instantiated from a given class. ControlStyle should not be modified at run time, except in the constructor. The ControlState (see page 5813) property records transient conditions.

The TControl (see page 5724) constructor initializes ControlStyle to the set of csCaptureMouse, csClickEvents, csSetCaption, and csDoubleClicks.

ControlStyle is primarily used by component writers when implementing components derived from TControl (see page 5724).

See Also

ControlState (see page 5813)

Create (see page 5731)

32.1.2.6.18 TControl.Cursor Property

Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.

Pascal

```
property Cursor: TCursor;
```

C++

```
__property TCursor Cursor;
```

Description

Change the value of Cursor to provide feedback to the user when the mouse pointer enters the control. The value of Cursor is the index of the cursor in the list of cursors maintained by the global variable, Screen. In addition to the built-in cursors provided by TScreen, applications can add custom cursors to the list.

C++ Examples:

```
/*
This example shows how to add custom cursors to an
application.  The following code makes this cursor available
to the application via the constant crMyCursor, and sets it
as the global cursor to the application.
*/
Graphics::TBitmap *bmpMask, *bmpColor;
TIconInfo *iconInfo;

const crMyCursor = 5;
```

```
  void __fastcall TForm1::Button2Click(TObject *Sender)
{
  bmpMask = new Graphics::TBitmap();
  bmpColor = new Graphics::TBitmap();
  iconInfo = new TIconInfo();

  bmpMask->LoadFromFile("../SquareMask.bmp");
  bmpColor->LoadFromFile("../Square.bmp");

  iconInfo->fIcon = false;
  iconInfo->xHotspot = 15;
  iconInfo->yHotspot = 15;
  iconInfo->hbmMask = bmpMask->Handle;
  iconInfo->hbmColor = bmpColor->Handle;

  Screen->Cursors[crMyCursor] = CreateIconIndirect(iconInfo);

  Screen->Cursor = crMyCursor;

  delete bmpMask;
  delete bmpColor;
  delete iconInfo;
}
```

Delphi Examples:

```
{
This example uses a splitter and two image controls to
create a "slide show" effect.  The example displays an image
(the current "slide"), which can be hidden behind a screen
that looks like a venetian blind. When the blind is pulled
down to completely cover the current slide, the slide changes
to a different image.  When the blind is raised, the new
image appears.
To prepare this example, follow these steps:
1. Place an image control on the form.
Set the picture property to a bitmap of a venetian blind,
Set the Stretch property to True.
Set the Align property to alTop.
2. Add a splitter control to the form.
Set the Align property to alTop.
Set the Cursor property to crVSplit.
Set the OnMoved event handler to the code that appears below.
Set the MinSize property to 1.
3. Add a second image control to the form.
Set the Align property to alClient.
Set the Stretch property to True.
4. Select the form and add the OnCreate and OnDestroy event
   handlers that appear below.
5. Add the global variables below to the unit.
}
var
  CurImage: Integer;
  BMPs: Array[1..5] of TBitmap;

procedure TForm1.Splitter1Moved(Sender: TObject);
begin
  { check if the blind is closed }
  if Image2.Height <= 1 then
  begin
    { the blind is closed, change the current slide }
    CurImage := CurImage + 1;
    if CurImage = 6 then
       CurImage := 1;
    Image2.Picture.Bitmap := BMPs[CurImage];
  end;
```

```
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { Initialize the array of slides.
    These must be freed in the OnDestroy event handler. }
  BMPs[1] := TBitmap.Create;
  BMPs[1].LoadFromFile('Factory.bmp');
  BMPs[2] := TBitmap.Create;
  BMPs[2].LoadFromFile('Rhododendron.bmp');
  BMPs[3] := TBitmap.Create;
  BMPs[3].LoadFromFile('littleB_64.bmp');
  BMPs[4] := TBitmap.Create;
  BMPs[4].LoadFromFile('Soap bubbles.bmp');
  BMPs[5] := TBitmap.Create;
  BMPs[5].LoadFromFile('Gone Fishing.bmp');
  CurImage := 1;
  Image2.Picture.Bitmap := BMPs[1];
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  { clean up }
  for I := 1 to 5 do
    BMPs[I].Free;
end;
{
This example shows how to add custom cursors to an
application.  The following code makes this cursor available
to the application via the constant crMyCursor, and sets it
as the global cursor to the application.
}
var
  bmpMask : TBitmap;
  bmpColor : TBitmap;
  iconInfo : TIconInfo;
const
  crMyCursor = 5;

procedure TForm1.Button2Click(Sender: TObject);
begin
  bmpMask := TBitmap.Create;
  bmpColor := TBitmap.Create;

  bmpMask.LoadFromFile('SquareMask.bmp');
  bmpColor.LoadFromFile('Square.bmp');

  with iconInfo do
  begin
    fIcon := false;
    xHotspot := 15;
    yHotspot := 15;
    hbmMask := bmpMask.Handle;
    hbmColor := bmpColor.Handle;
  end;

  Screen.Cursors[crMyCursor] := CreateIconIndirect(iconInfo);

  Screen.Cursor := crMyCursor;

  bmpMask.Free;
  bmpColor.Free;
end;
```

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
  DestroyIcon(Screen.Cursors[crMyCursor]);
end;
```

See Also

DragCursor (see page 5818)

Cursors (see page 14212)

Screen (see page 14385)

ProcessMessages (see page 13691)

32.1.2.6.19 TControl.DesktopFont Property

Specifies whether the control uses the Windows icon font when writing text.

Pascal

```
property DesktopFont: Boolean;
```

C++

```
__property Boolean DesktopFont;
```

Description

Set DesktopFont to true to specify that the control should use the Windows icon font for its Font (see page 5823) property. The Windows icon font is the value of the IconFont property of the global Screen variable.

When DesktopFont is true, the control's Font (see page 5823) property updates every time the Windows icon font changes. This font can be changed by setting the IconFont property of the global Screen variable, or by the actions of other programs. Setting the Font (see page 5823) property from the IconFont property of the Screen object does not allow controls to dynamically respond in this manner when the Windows icon font changes.

See Also

Font (see page 5823)

IconFont (see page 14226)

32.1.2.6.20 TControl.DockOrientation Property

Specifies how the control is docked relative to other controls docked in the same parent.

Pascal

```
property DockOrientation: TDockOrientation;
```

C++

```
__property TDockOrientation DockOrientation;
```

Description

Use DockOrientation to determine the position of the control relative to other controls docked to the same parent. Docking zones can be arranged from left to right or from top to bottom.

32.1.2.6.21 TControl.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor (see page 5815) property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

32.1.2.6.22 TControl.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

32.1.2.6.23 TControl.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
```

```
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

32.1.2.6.24 TControl.Enabled Property

Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

32.1.2.6.25 TControl.ExplicitHeight Property

Specifies the explicit vertical size of the control in pixels.

Pascal

```
property ExplicitHeight: Integer;
```

C++

```
__property int ExplicitHeight;
```

Description

ExplicitHeight is a read only property used internally by Delphi. Use Height (see page 5825) in applications, thereby allowing read and write access.

See Also

Height (see page 5825)

32.1.2.6.26 TControl.ExplicitLeft Property

Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.

Pascal

```
property ExplicitLeft: Integer;
```

C++

```
__property int ExplicitLeft;
```

Description

ExplicitLeft is a read only property used internally by Delphi. Use Left (see page 5830) in applications, thereby allowing read and write access.

See Also

Left (see page 5830)

32.1.2.6.27 TControl.ExplicitTop Property

Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.

Pascal

```
property ExplicitTop: Integer;
```

C++

```
__property int ExplicitTop;
```

Description

ExplicitTop is a read only property used internally by Delphi. Use Top (see page 5844) in applications, thereby allowing read and write access.

See Also

Top (see page 5844)

32.1.2.6.28 TControl.ExplicitWidth Property

Specifies the explicit horizontal size of the control in pixels.

Pascal

```
property ExplicitWidth: Integer;
```

C++

```
__property int ExplicitWidth;
```

Description

ExplicitWidth is a read only property used internally by Delphi. Use Width (see page 5849) in applications, thereby allowing read and write access.

See Also

Width (see page 5849)

32.1.2.6.29 TControl.Floating Property

Indicates whether the control is floating.

Pascal

```
property Floating: Boolean;
```

C++

```
__property Boolean Floating;
```

Description

Read Floating to determine whether a dockable control appears in a floating window, rather than docked to a parent control.

Note: Use Floating rather than the Parent (see page 5832) property to determine whether a control is free-floating. Floating controls may have a non-nil (Delphi) or NULL (C++) parent if the FloatingDockSiteClass (see page 5823) property specifies a floating window type.

See Also

DragKind (see page 5819)

FloatingDockSiteClass (see page 5823)

Parent (see page 5832)

HostDockSite (see page 5829)

32.1.2.6.30 TControl.FloatingDockSiteClass Property

Specifies the class of the temporary control that hosts the control when it is floating.

Pascal

```
property FloatingDockSiteClass: TWinControlClass;
```

C++

```
__property TWinControlClass FloatingDockSiteClass;
```

Description

Use FloatingDockSiteClass to specify the class that houses the control when it is floating. This class must be a descendant of TWinControl (see page 6112). By default, it is TCustomDockForm. When the control is undocked, a temporary control of this type is created and the control is docked to that control. When the control is docked to another control, the temporary floating dock site is freed.

If the control can float as a stand-alone window, FloatingDockSiteClass is nil (Delphi) or NULL (C++).

See Also

Floating (see page 5823)

Parent (see page 5832)

HostDockSite (see page 5829)

TCustomDockForm (see page 13829)

32.1.2.6.31 TControl.Font Property

Controls (see page 5711) the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color (see page 5812), Height (see page 5825), Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

32.1.2.6.32 TControl.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
HideSelection, Height example
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
```

```
        HideSelection := False;
    end;
end;
```

See Also

ClientHeight (see page 5808)

Width (see page 5849)

Using TDBListBox and TDBComboBox

32.1.2.6.33 TControl.HelpContext Property

Numeric ID for control's context-sensitive help topic

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext supports help systems that use numeric topic IDs. For help systems that use topic keywords, see HelpKeyword (see page 5826).

To enable context-sensitive help for a control, set HelpType (see page 5827) to htContext and set HelpContext to a numeric topic ID. A topic ID of 0 (default) means that no help topic is provided for the control.

See Also

HelpType (see page 5827)

HelpKeyword (see page 5826)

InvokeHelp

32.1.2.6.34 TControl.HelpKeyword Property

Keyword for control's context-sensitive help topic

Pascal

```
property HelpKeyword: String;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword supports help systems that use topic keywords. For help systems that use numeric topic IDs, see HelpContext (see page 5826).

To enable context-sensitive help for a control, set HelpType (see page 5827) to htKeyword and set HelpKeyword to a keyword string. No keyword (zero-length string; default) means that no help topic is provided for the control.

See Also

HelpType (see page 5827)

HelpContext (see page 5826)

InvokeHelp

32.1.2.6.35 TControl.HelpType Property

Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

Set HelpType to specify whether the control identifies its context-sensitive help topic by ID or keyword. When HelpType is htContext, the HelpContext (see page 5826) property identifies the control's help topic. When HelpType is htKeyword, the HelpKeyword (see page 5826) property identifies the control's help topic.

32.1.2.6.36 TControl.Hint Property

Contains the text string that can appear when the user moves the mouse over the control.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Use the Hint property to provide a string of help text either as a Help Hint, or as help text on a particular location such as a status bar.

A Help Hint is a box containing help text that appears for a control when the user moves the mouse pointer over the control and pauses momentarily. To set up Help Hints:

Specify the Hint property of each control for which a Help Hint should appear.

Set the ShowHint (see page 5840) property of each appropriate control to true, or set the ParentShowHint (see page 5835) property of all controls to true and set the ShowHint (see page 5840) property of the form to true.

At runtime, set the value of the application's ShowHint (see page 5840) property to true.

To show the Hint on a status bar or other location, use the OnHint event handler of the application. The application's OnHint event occurs when the mouse pointer moves over the control.

Specify a hint to be used for both a Help Hint box and by an OnHint handler by specifying two values separated by a | character (the vertical bar "or" symbol). For example,

```
Edit1.Hint := 'Name|Enter Name in the edit box';
Edit1->Hint = "Name|Enter Name in the edit box";
```

The "Name" part appears in the Help Hint box and the "Enter full name in the edit box" part can be extracted using the GetLongHint (see page 6252) function for use in an OnHint event handler.

If Hint contains only one value, the entire string is used as a Help Hint and returned by the GetLongHint (see page 6252) function. If a control has no Hint value specified, but its parent control does, the control uses the Hint value of the parent control (as long as the control's ShowHint (see page 5840) property is true).

Note: If the application's ShowHint (see page 5840) property is false, the Help Hint does not appear, but the OnHint event handler is still called.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

GetLongHint (see page 6252)

GetShortHint (see page 6253)

ParentShowHint (see page 5835)

ShowHint (see page 5840)

Hint (see page 13712)

OnHint (see page 13731)

ShowHint (see page 13721)

Hint (see page 312)

32.1.2.6.37 TControl.HostDockSite Property

Specifies the control in which the control is docked.

Pascal

```
property HostDockSite: TWinControl;
```

C++

```
__property TWinControl * HostDockSite;
```

Description

Read HostDockSite to access the windowed control to which the control is docked. If the control is floating, HostDockSite is a temporary control of type FloatingDockSiteClass (see page 5823) or (if there is no floating dock site class) nil (Delphi) or NULL (C++).

Most applications should use the ManualDock (see page 5771) method rather than setting HostDockSite. Setting HostDockSite automatically undocks the control from its previous host (if any) and docks it to the specified control, but does not adjust the control's position and alignment and bypasses the standard docking events.

For dock clients, the value of HostDockSite is the same as the value of Parent (see page 5832). For controls that are not docking clients, HostDockSite is nil (Delphi) or NULL (C++) while Parent (see page 5832) specifies the containing control.

See Also

DockClients (see page 6201)

ManualDock (see page 5771)

FloatingDockSiteClass (see page 5823)

Parent (see page 5832)

32.1.2.6.38 TControl.IsControl Property

Determines whether a form stores its form-specific properties to a stream.

Pascal

```
property IsControl: Boolean;
```

C++

```
__property Boolean IsControl;
```

Description

Use IsControl to use a form as a complex control, rather than as a desktop form. IsControl affects the way the form gets saved. When IsControl is true, form-specific properties do not get saved with the form. When IsControl is false, the form-specific properties get saved along with the other properties.

IsControl permits you to use the forms designer to create complex controls by creating those controls as forms, placing and naming their contained controls, and attaching code to events. After saving the form, edit the form as text, and in the text version set IsControl to true. The next time the form is loaded and saved, only those form properties appropriate to use as a control are stored, not the properties specific to TForm. Next, edit the form file as text, changing the type of the component from TForm to the desired control type, such as TPanel.

See Also

TForm (see page 13992)

32.1.2.6.39 TControl.Left Property

Specifies the horizontal coordinate of the left edge of a component relative to its parent.

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

Use the Left property to determine where the left side of the control begins or to reposition the left side of the control.

If the control is contained in another control, the Left and Top (see page 5844) properties are relative to the parent control. If the control is contained directly by the form, the property values are relative to the form. For forms, the value of the Left property is relative to the screen in pixels.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Timer1->Interval = 100;
  if (ActiveControl)
    ActiveControl->Left = ActiveControl->Left + 1;
}
```

Delphi Examples:

```
{
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  if ActiveControl <> nil then
    ActiveControl.Left := ActiveControl.Left + 1;
end;
```

See Also

Top (see page 5844)

32.1.2.6.40 TControl.LRDockWidth Property

Specifies the width of the control when it is docked horizontally.

Pascal

```
property LRDockWidth: Integer;
```

C++

```
__property int LRDockWidth;
```

Description

Read LRDockWidth to get the width of the control from the last time it was docked horizontally. Set LRDockWidth to indicate the width the control should have the next time it is docked horizontally.

LRDockWidth allows a control to "remember" its docked size even when its width changes because it is undocked or docked with a vertical orientation.

See Also

Width (see page 5849)

DockOrientation (see page 5818)

TBDockHeight (see page 5842)

UndockWidth (see page 5846)

32.1.2.6.41 TControl.Margins Property

Specifies the margins for the control.

Pascal

```
property Margins: TMargins;
```

C++

```
__property TMargins Margins;
```

Description

Use the Margins property to set the margins for the control.

32.1.2.6.42 TControl.MouseCapture Property

Indicates whether the control has "captured" mouse events.

Pascal

```
property MouseCapture: Boolean;
```

C++

```
__property Boolean MouseCapture;
```

Description

Use MouseCapture to determine whether a control has captured the mouse. When a control captures the mouse, all subsequent mouse events go to that control until the user releases the mouse button.

A control captures the mouse when the user drags an item from it. In addition, if the control has the csCaptureMouse flag set in its ControlStyle (see page 5814) property, it captures the mouse when the user presses the left mouse button over it, until the

user releases the mouse button.

See Also

ControlStyle (see page 5814)

GetCaptureControl (see page 6251)

SetCaptureControl (see page 6263)

OnMouseDown (see page 5862)

32.1.2.6.43 TControl.Parent Property

Indicates the parent of the control.

Pascal

```
property Parent: TWinControl;
```

C++

```
__property TWinControl * Parent;
```

Description

Use the Parent property to get or set the parent of this control. The parent of a control is the control that contains the control. For example, if an application includes three radio buttons in a group box, the group box is the parent of the three radio buttons, and the radio buttons are the child controls of the group box.

To serve as a parent, a control must be an instance of a descendant of TWinControl (see page 6112).

When creating a new control at runtime, assign a Parent property value for the new control. Usually, this is a form, panel, group box, or some control that is designed to contain another. Changing the parent of a control moves the control onscreen so that it is displayed within the new parent. When the parent control moves, the child moves with the parent.

Some controls (such as ActiveX controls) are contained in native windows rather than in a parent VCL control. For these controls, the value of Parent is nil (Delphi) or NULL (C++) and the ParentWindow property specifies the window.

Note: The Parent property declared in TControl (see page 5724) is similar to the Owner property declared in TComponent, in that the Parent of a control frees the control just as the Owner of a component frees that Component. However, the Parent of a control is always a windowed control that visually contains the control, and is responsible for writing the control to a stream when the form is saved. The Owner of a component is the component that was passed as a parameter in the constructor and, if assigned, initiates the process of saving all objects (including the control and its parent) when the form is saved.

C++ Examples:

```
/*
This example uses a button placed next to a group box. When the
user clicks the button, the group box becomes the parent of the
button, so the button moves inside the group box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (Button1->Parent != GroupBox1) // second click?
  {
    RemoveControl(Button1);
    GroupBox1->InsertControl(Button1);
  }
}

/*
Note that it was necessary to remove the button from the Controls
```

```
property of the form before the button actually moves into the
group box. This code accomplishes the same thing:
*/

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Button2->Parent = GroupBox1;
}
```

Delphi Examples:

```
{
This example uses a button placed next to a group box. When the
user clicks the button, the group box becomes the parent of the
button, so the button moves inside the group box.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (Button1.Parent <> GroupBox1) then // second click?
  begin
    RemoveControl(Button1);
    GroupBox1.InsertControl(Button1);
  end;
end;

{
Note that it was necessary to remove the button from the Controls
property of the form before the button actually moves into the
group box. This code accomplishes the same thing:
}

procedure TForm1.Button2Click(Sender: TObject);
begin
  Button2.Parent := GroupBox1;
end;
```

See Also

Owner (see page 2807)

Controls (see page 6197)

HostDockSite (see page 5829)

ParentWindow (see page 6210)

32.1.2.6.44 TControl.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode (see page 5803).

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode (see page 5803). When ParentBiDiMode is true, the control's BiDiMode (see page 5803) property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode (see page 5803) property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode (see page 5803) property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

32.1.2.6.45 TControl.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color (see page 5812) property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color (see page 5812) property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

32.1.2.6.46 TControl.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font (see page 5823) property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font (see page 5823) property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

32.1.2.6.47 TControl.ParentShowHint Property

Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint (see page 5840) property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint (see page 5840) property.

To provide Help Hints for only selected controls on a form, set the ShowHint (see page 5840) property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint (see page 5840) property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

32.1.2.6.48 TControl.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup (see page 5854) event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
```

```
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
```

```
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
```

```
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
```

```
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

32.1.2.6.49 TControl.ScalingFlags Property

Reflects which attributes of the control have been scaled.

Pascal

```
property ScalingFlags: TScalingFlags;
```

C++

```
__property TScalingFlags ScalingFlags;
```

Description

ScalingFlags is an internal set of flags used primarily in scaling inherited forms and their controls. Applications should not use this property without a serious need.

These are the flags and their meanings:

Flag	Meaning
sfLeft	Left (see page 5830) side of control not yet scaled.
sfTop	Top (see page 5844) of control not yet scaled.
sfWidth	Width (see page 5849) of control not yet scaled.
sfHeight	Height (see page 5825) of control not yet scaled.
sfFont	Font (see page 5823) of control not yet scaled.
sfDesignSize	Design-time dimensions of control not yet scaled.

See Also

ChangeScale (see page 5743)

32.1.2.6.50 TControl.ShowHint Property

Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint (see page 5827) is the value of the Hint (see page 5827) property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint (see page 5827) appears for the control.

To enable Help Hint (see page 5827) for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint (see page 5835) property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint (see page 5835) property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint (see page 5827).

Changing the ShowHint value automatically sets the ParentShowHint (see page 5835) property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
```

```
    with ListBox1 do
    begin
      for I := 1 to 10 do
        Items.Add('Item ' + IntToStr(I));
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

32.1.2.6.51 TControl.TBDockHeight Property

Specifies the height of the control when it is docked vertically.

Pascal

```
property TBDockHeight: Integer;
```

C++

```
__property int TBDockHeight;
```

Description

Read TBDockHeight to get the height of the control from the last time it was docked vertically. Set TBDockHeight to indicate the height the control should have the next time it is docked vertically.

TBDockHeight allows a control to "remember" its docked size even when its height changes because it is undocked or docked with a horizontal orientation.

See Also

Height (see page 5825)

DockOrientation (see page 5818)

LRDockWidth (see page 5831)

UndockHeight (see page 5845)

32.1.2.6.52 TControl.Text Property

Contains a text string associated with the control.

Pascal

```
property Text: TCaption;
```

C++

```
__property TCaption Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls (see page 5711) that display text use either the Caption (see page 5806) property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption (see page 5806) is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
  Size++;                        //Add room for null character
  char *Buffer = new char[Size];  //Creates Buffer dynamic variable
  Edit1->GetTextBuf(Buffer,Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer;
  delete [] Buffer;
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
  Inc(Size);                      {Add room for null character}
  GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
  Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
end;
```

See Also

Caption (see page 5806)

Lines (see page 21380)

32.1.2.6.53 TControl.Top Property

Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

Use Top to locate the top of the control or reposition the control to a different Y coordinate. The Top property, like the Left (see page 5830) property, is the position of the control relative to its container. Thus, if a control is contained in a TPanel, the Left (see page 5830) and Top properties are relative to the panel. If the control is contained directly by the form, it is relative to the form. For forms, the value of the Top property is relative to the screen in pixels.

C++ Examples:

```
/*
This example uses an updown control and a group box on a
form, with several controls contained within the group box.
When the updown control or is clicked, the controls within
the group box are moved in the appropriate direction.
*/
void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  int I;
  TControl *ChildControl;

  for (I = 0; I < GroupBox1->ControlCount; I++)
  {
    ChildControl = GroupBox1->Controls[I];
    if (Button == btNext)
      ChildControl->Top = ChildControl->Top + 15;
    else
      ChildControl->Top = ChildControl->Top - 15;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorarray[5] = {clYellow, clGreen, clBlue, clLime, clFuchsia};
  GroupBox1->Brush->Color = clRed;
  for (int i = 0; i < GroupBox1->ControlCount; i++)
    ((TWinControl *)GroupBox1->Controls[i])->Brush->Color = colorarray[i];
}
```

Delphi Examples:

```
{
This example uses an updown control and a group box on a
form, with several controls contained within the group box.
When the updown control or is clicked, the controls within
the group box are moved in the appropriate direction.
}
procedure TForm1.FormCreate(Sender: TObject);
var
```

```
    I: Integer;
  const
    colorarray : Array[0..4] of TColor = (clYellow, clGreen, clBlue, clLime, clFuchsia);
  begin
     GroupBox1.Brush.Color := clRed;
    for I:= 0 to GroupBox1.ControlCount -1 do
    begin
      TWinControl(GroupBox1.Controls[I]).Brush.Color := colorarray[I];
    end;
  end;

  procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
  var
    I: Integer;
    ChildControl: TControl;
  begin
    for I:= 0 to GroupBox1.ControlCount -1 do
    begin
      ChildControl := GroupBox1.Controls[I];
      if Button = btNext then
        ChildControl.Top := ChildControl.Top + 15
      else
        ChildControl.Top := ChildControl.Top - 15;
    end;
  end;
```

See Also

Left (see page 5830)

32.1.2.6.54 TControl.UndockHeight Property

Specifies the height of the control when it is floating.

Pascal

```
property UndockHeight: Integer;
```

C++

```
__property int UndockHeight;
```

Description

Read UndockHeight to get the height of the control from the last time it was floating. Set UndockHeight to indicate the height the control should have the next time it is undocked.

UndockHeight allows a control to "remember" its floating size even when its height changes because it is docked.

C++ Examples:

```
{
This example shows how to undock a dockable object.  This example
is part of the Docking Demo.
}

void __fastcall TTabDockHost::FormClose(TObject *Sender, TCloseAction &Action)
{
  TRect ARect;
  TPoint point;
  if (PageControl1->DockClientCount == 1)
  {
    point =  PageControl1->DockClients[0]->ClientToScreen(Point(0, 0));
    ARect.left = point.x;
    ARect.top = point.y;
    point = PageControl1->DockClients[0]->ClientToScreen(
      Point(PageControl1->DockClients[0]->UndockWidth,
```

```
PageControl1->DockClients[0]->UndockHeight));
    ARect.right = point.x;
    ARect.bottom = point.y;
    PageControl1->DockClients[0]->ManualFloat(ARect);
    Action = caFree;
  }
  else
    Action = caHide;
}
```

Delphi Examples:

```
procedure TTabDockHost.FormClose(Sender: TObject;
  var Action: TCloseAction);
var
  ARect: TRect;
begin
  if PageControl1.DockClientCount = 1 then
  begin
    with PageControl1.DockClients[0] do
    begin
      ARect.TopLeft := ClientToScreen(Point(0, 0));
      ARect.BottomRight := ClientToScreen(Point(UndockWidth, UndockHeight));
      ManualFloat(ARect);
    end;
    Action := caFree;
  end else
    Action := caHide;
end;
```

See Also

Height (see page 5825)

Floating (see page 5823)

TBDockHeight (see page 5842)

UndockWidth (see page 5846)

32.1.2.6.55 TControl.UndockWidth Property

Specifies the width of the control when it is floating.

Pascal

```
property UndockWidth: Integer;
```

C++

```
__property int UndockWidth;
```

Description

Read UndockWidth to get the width of the control from the last time it was floating. Set UndockWidth to indicate the width the control should have the next time it is undocked.

UndockWidth allows a control to "remember" its floating size even when its width changes because it is docked.

C++ Examples:

```
{
This example shows how to undock a dockable object.  This example
is part of the Docking Demo.
}

void __fastcall TTabDockHost::FormClose(TObject *Sender, TCloseAction &Action)
{
```

```
  TRect ARect;
  TPoint point;
  if (PageControl1->DockClientCount == 1)
  {
    point =  PageControl1->DockClients[0]->ClientToScreen(Point(0, 0));
    ARect.left = point.x;
    ARect.top = point.y;
    point = PageControl1->DockClients[0]->ClientToScreen(
      Point(PageControl1->DockClients[0]->UndockWidth,
PageControl1->DockClients[0]->UndockHeight));
    ARect.right = point.x;
    ARect.bottom = point.y;
    PageControl1->DockClients[0]->ManualFloat(ARect);
    Action = caFree;
  }
  else
    Action = caHide;
}
```

Delphi Examples:

```
procedure TTabDockHost.FormClose(Sender: TObject;
  var Action: TCloseAction);
var
  ARect: TRect;
begin
  if PageControl1.DockClientCount = 1 then
  begin
    with PageControl1.DockClients[0] do
    begin
      ARect.TopLeft := ClientToScreen(Point(0, 0));
      ARect.BottomRight := ClientToScreen(Point(UndockWidth, UndockHeight));
      ManualFloat(ARect);
    end;
    Action := caFree;
  end else
    Action := caHide;
end;
```

See Also

Width (see page 5849)

Floating (see page 5823)

LRDockWidth (see page 5831)

UndockHeight (see page 5845)

32.1.2.6.56 TControl.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show (see page 5788) method sets the control's Visible property to true. Calling the Hide (see page 5769)

method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
```

```
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

32.1.2.6.57 TControl.WheelAccumulator Property

Accumulates mouse wheel rotations.

Pascal

```
property WheelAccumulator: Integer;
```

C++

```
__property int WheelAccumulator;
```

Description

WheelAccumulator contains a value indicating the progress of event handlers in response to mouse wheel motion. See DoMouseWheel (see page 5751) for more information.

See Also

DoMouseWheel (see page 5751)

32.1.2.6.58 TControl.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

See Also

ClientWidth (see page 5811)

Height (see page 5825)

Left (see page 5830)

32.1.2.6.59 TControl.WindowText Property

Contains the text string of the control.

Pascal

```
property WindowText: PChar;
```

C++

```
__property const char * WindowText;
```

Description

Use WindowText to determine the text associated with the control. When setting WindowText, the control frees the memory associated with the previous value of WindowText before assigning a new value. Any memory associated with WindowText is freed when the control is destroyed. By default, WindowText is the same string as the Text (see page 5842) property. Descendants of TControl (see page 5724) can override the Text (see page 5842) property to implement a different value.

For an edit box, the WindowText is the content of the control. For a combo box, the text string is the content of the edit box part of the combo box. For a button, the string is the button name. For all other controls, the string is the window title.

See Also

Caption (see page 5806)

Text (see page 5842)

32.1.2.7 TControl Events

The events of the TControl class are listed here.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.

	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

32.1.2.7.1 TControl.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize: TCanResizeEvent;
```

C++

```
__property TCanResizeEvent OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize (see page 5853) event.

See Also

OnConstrainedResize (see page 5853)

SetBounds (see page 5783)

OnResize (see page 5870)

32.1.2.7.2 TControl.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.

- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption (see page 5806) property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click (see page 5743) method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

32.1.2.7.3 TControl.OnConstrainedResize Event

Adjust resize constraints.

Pascal

```
property OnConstrainedResize: TConstrainedResizeEvent;
```

C++

```
__property TConstrainedResizeEvent OnConstrainedResize;
```

Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints (see page 5812) object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize (see page 5741) method or an OnCanResize (see page 5851) event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize (see page 5870) event occurs to allow any final adjustments or responses.

Note: The OnConstrainedResize handler is called immediately after the OnCanResize (see page 5851) handler.

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

Constraints (see page 5812)

OnResize (see page 5870)

CanAutoSize (see page 5741)

32.1.2.7.4 TControl.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu (see page 5835) property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent (see page 5832) controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

32.1.2.7.5 TControl.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

32.1.2.7.6 TControl.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
```

```
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

32.1.2.7.7 TControl.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor (see page 5818) property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
```

```
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

32.1.2.7.8 TControl.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

32.1.2.7.9 TControl.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

32.1.2.7.10 TControl.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate (see page 5773) is a protected virtual function in TControl (see page 5724). MouseActivate (see page 5773) can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate (see page 5773) as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate (see page 5773), the behavior is the same as if a control set TMouseActivate (see page 6287) to maActivate. At any point, a control can set the value of TMouseActivate (see page 6287) and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate (see page 6287) is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate (see page 5773), based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl (see page 5724) level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl (see page 5724) descendants also have OnMouseDown (see page 5862), OnMouseMove (see page 5865), and OnMouseUp (see page 5867) events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

32.1.2.7.11 TControl.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}
```

```
// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
```

```
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

32.1.2.7.12 TControl.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

32.1.2.7.13 TControl.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

32.1.2.7.14 TControl.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown (see page 5862) event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

32.1.2.7.15 TControl.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
```

```
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
```

```
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

32.1.2.7.16 TControl.OnMouseWheel Event

Occurs when the mouse wheel is rotated.

Pascal

```
property OnMouseWheel: TMouseWheelEvent;
```

C++

```
__property TMouseWheelEvent OnMouseWheel;
```

Description

Write code in the OnMouseWheel event handler to respond to mouse wheel messages. If there is no OnMouseWheel event handler, or if the mouse wheel message is not handled in the OnMouseWheel event handler, then an OnMouseWheelUp (see page 5870) or OnMouseWheelDown (see page 5869) event occurs, depending on the direction the mouse wheel was rotated.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheelUp (see page 5870)

32.1.2.7.17 TControl.OnMouseWheelDown Event

Occurs when the mouse wheel is rotated downward.

Pascal

```
property OnMouseWheelDown: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelDown;
```

Description

Write code in the OnMouseWheelDown event handler to perform actions when the mouse wheel is rotated downward. OnMouseWheelDown occurs only if the message is not already handled in an OnMouseWheel (see page 5869) event handler.

See Also

OnMouseWheel (see page 5869)

OnMouseWheelUp (see page 5870)

32.1.2.7.18 TControl.OnMouseWheelUp Event

Occurs when the mouse wheel is rotated upward.

Pascal

```
property OnMouseWheelUp: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelUp;
```

Description

Write code in the OnMouseWheelUp event handler to perform actions when the mouse wheel is rotated upward. OnMouseWheelUp occurs only if the message is not already handled in an OnMouseWheel (see page 5869) event handler.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheel (see page 5869)

32.1.2.7.19 TControl.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize (see page 5851) or OnConstrainedResize (see page 5853).

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
```

```
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

32.1.2.7.20 TControl.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx (see page 6018) object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx (see page 6018) as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject (see page 6009), your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject (see page 6009) object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

32.1.2.7.21 TControl.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind (see page 5819) is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx (see page 6007) instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx (see page 6007) instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject (see page 6003) instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject (see page 6003) object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag (see page 5860) events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

32.1.2.7.22 TControl.WindowProc Event

Points to the window procedure that responds to messages sent to the control.

Pascal

```
property WindowProc: TWndMethod;
```

C++

```
__property TWndMethod WindowProc;
```

Description

Use the WindowProc property to temporarily replace or subclass the window procedure of the control. Component writers that are customizing the window procedure for a descendant class should override the WndProc (see page 5791) method instead.

Before assigning a new value to WindowProc, store the original value. WindowProc is initially set to the WndProc (see page 5791) method, so if the value has not been changed since then, the original value need not be stored. Within the procedure that is used as the new value for WindowProc, pass any unhandled messages to the original procedure that was the value of WindowProc. After completing any specialized message handling, restore the value of WindowProc to the original procedure.

C++ Examples:

```
/*
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
*/
class TMyListBoxDescendant : public TListBox
{
```

```
__published:    // IDE-managed Components
    void __fastcall SubClassWndProc(Messages::TMessage &Message);
    void __fastcall ToggleSubClass(bool On);
    void __fastcall OnDrawItemProc(
      TWinControl *Control, int Index, const TRect &Rect,
      TOwnerDrawState State);
private:    // User declarations
public:     // User declarations
    __fastcall TMyListBoxDescendant(TComponent* Owner);
};

TForm1 *Form1;
TMyListBoxDescendant *MyListBoxDescendant1;
Graphics::TBitmap *bitmap0;

const WM_STYLEMESSAGE = WM_USER + 2000;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

__fastcall TMyListBoxDescendant::TMyListBoxDescendant(TComponent* Owner)
    : TListBox(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
}

void __fastcall TForm1::SubClassRadioGroup1Click(TObject *Sender)
{
  MyListBoxDescendant1->ToggleSubClass(SubClassRadioGroup1->ItemIndex == 0);
}

void __fastcall TMyListBoxDescendant::SubClassWndProc(Messages::TMessage &Message)
{
  if (Message.Msg == WM_STYLEMESSAGE)
    Style = (TListBoxStyle)Message.WParam;
  else
    WndProc(Message);
}

void __fastcall TMyListBoxDescendant::ToggleSubClass(bool On)
{
  if (On)
    WindowProc = SubClassWndProc;
  else
    WindowProc = WndProc;
}
```

```
void __fastcall TMyListBoxDescendant::OnDrawItemProc(TWinControl *Control, int Index,
      const TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *bitmap; // temporary variable for item's bitmap
  int Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *canvas = ((TListBox *)Control)->Canvas;
  canvas->FillRect(Rect); // clear the rectangle
  bitmap = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
  if (bitmap)
  {
    canvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, bitmap->Width, bitmap->Height),
      bitmap, Bounds(0, 0, bitmap->Width, bitmap->Height), clRed); // render bitmap
    Offset += bitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  canvas->TextOut(Rect.Left + Offset, Rect.Top, ((TListBox *)Control)->Items->Strings[Index]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyListBoxDescendant1 = new TMyListBoxDescendant(Form1);
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Parent = Form1;
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Left =
    SubClassRadioGroup1->Left + SubClassRadioGroup1->Width + 30;;
  MyListBoxDescendant1->Top = SubClassRadioGroup1->Top;
  MyListBoxDescendant1->Height = SubClassRadioGroup1->Height;
  MyListBoxDescendant1->OnDrawItem =
    MyListBoxDescendant1->OnDrawItemProc;

  bitmap0 = new Graphics::TBitmap;
  ImageList1->GetBitmap(0, bitmap0);
  MyListBoxDescendant1->Items->AddObject("Butterfly", bitmap0);

  SubClassRadioGroup1->Items->Add("SubClassWndProc");
  SubClassRadioGroup1->Items->Add("WndProc");
  SubClassRadioGroup1->ItemIndex = 2;
}
```

Delphi Examples:

```
{
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
}

type
  TMyListBoxDescendant = class(TlistBox)
    procedure SubClassWndProc(var Message: TMessage);
    procedure ToggleSubClass(On: Boolean);
    procedure OnDrawItemProc(
      Control: TWinControl;
      Index: Integer;
      Rect:TRect;
      State: TOwnerDrawState);
  end;
  TForm1 = class(TForm)
    SubClassRadioGroup1: TRadioGroup;
    Button1: TButton;
    ImageList1: TImageList;
```

```
    Button2: TButton;
    procedure SubClassRadioGroup1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  MyListBoxDescendant1: TMyListBoxDescendant;
  bitmap0: TBitmap;

implementation

{$R *.dfm}

const WM_STYLEMESSAGE = WM_USER + 2000;

procedure TForm1.Button1Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
end;

procedure TMyListBoxDescendant.SubClassWndProc(var Message: TMessage);
begin
  if (Message.Msg = WM_STYLEMESSAGE) then
    Style:= TListBoxStyle(Message.WParam)
  else
    WndProc(Message);
end;

procedure TMyListBoxDescendant.ToggleSubClass(On: Boolean);
begin
  if On then
    WindowProc := SubClassWndProc
  else
    WindowProc := WndProc;
end;

procedure TForm1.SubClassRadioGroup1Click(Sender: TObject);
begin
  MyListBoxDescendant1.ToggleSubClass(
    SubClassRadioGroup1.ItemIndex = 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyListBoxDescendant1:= TMyListBoxDescendant.Create(self);
```

```
    MyListBoxDescendant1.Visible:= True;
    MyListBoxDescendant1.Parent:= Form1;
    MyListBoxDescendant1.Visible:= True;
    MyListBoxDescendant1.Left:=
      SubClassRadioGroup1.Left + SubClassRadioGroup1.Width + 30;;
    MyListBoxDescendant1.Top:= SubClassRadioGroup1.Top;
    MyListBoxDescendant1.Height:= SubClassRadioGroup1.Height;
    MyListBoxDescendant1.OnDrawItem:=
      MyListBoxDescendant1.OnDrawItemProc;

    bitmap0 := TBitmap.Create;
    ImageList1.GetBitmap(0, bitmap0);
    MyListBoxDescendant1.Items.AddObject('Butterfly', bitmap0);

    SubClassRadioGroup1.Items.Add('SubClassWndProc');
    SubClassRadioGroup1.Items.Add('WndProc');
    SubClassRadioGroup1.ItemIndex := 2;
  end;

  procedure TMyListBoxDescendant.OnDrawItemProc(
    Control: TWinControl;
    Index: Integer;
    Rect:TRect;
    State: TOwnerDrawState);
  var
    Bitmap: TBitmap;      { temporary variable for the item's bitmap }
    Offset: Integer;      { text offset width }
  begin
    { draw on control canvas, not on the form }
    with (Control as TListBox).Canvas do
    begin
      FillRect(Rect);       { clear the rectangle }
      Offset := 2;          { provide default offset }
      Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
      if Bitmap <> nil then
      begin
        BrushCopy(
          Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
          Bitmap,
          Bounds(0, 0, Bitmap.Width, Bitmap.Height),
          clRed);  {render bitmap}
        Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
      end;
      TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
  text }
    end;
  end;
```

See Also

WndProc (see page 5791)

32.1.3 TControlAction Class

Class Hierarchy

Pascal

```
TControlAction = class(TCustomControlAction);
```

C++

```
class TControlAction : public TCustomControlAction;
```

File

Controls

Description

This is class Controls.TControlAction.

32.1.3.1 TControlAction Members

The following tables list the members exposed by TControlAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.

	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnExecute (see page 5888)	Occurs when the client event that is linked to it fires.
	OnHint (see page 5890)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 5890)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 5881)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 5882)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 5882)	Indicates whether client controls and menu items appear checked.
	DropdownMenu (see page 5883)	This is DropdownMenu, a member of class TControlAction.

	Enabled (see page 5884)	Indicates whether client controls and menu items are enabled.
	EnableDropdown (see page 5884)	This is EnableDropdown, a member of class TControlAction.
	GroupIndex (see page 5884)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 5885)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 5885)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 5886)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 5886)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 5886)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	PopupMenu (see page 5887)	This is PopupMenu, a member of class TControlAction.
	SecondaryShortCuts (see page 5887)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 5887)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 5888)	Specifies the Visible property for client controls and menu items.

32.1.3.2 TControlAction Properties

The properties of the TControlAction class are listed here.

Published Properties

	Name	Description
	AutoCheck (see page 5881)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 5882)	Represents the caption of client controls and menu items.
	Checked (see page 5882)	Indicates whether client controls and menu items appear checked.
	DropdownMenu (see page 5883)	This is DropdownMenu, a member of class TControlAction.
	Enabled (see page 5884)	Indicates whether client controls and menu items are enabled.
	EnableDropdown (see page 5884)	This is EnableDropdown, a member of class TControlAction.
	GroupIndex (see page 5884)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 5885)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 5885)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 5886)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 5886)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 5886)	Indicates the ImageIndex property for client controls and menu items.
	PopupMenu (see page 5887)	This is PopupMenu, a member of class TControlAction.
	SecondaryShortCuts (see page 5887)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 5887)	Specifies the ShortCut property for client menu items.
	Visible (see page 5888)	Specifies the Visible property for client controls and menu items.

32.1.3.2.1 TControlAction.AutoCheck Property

Controls whether the Checked property toggles when the action executes.

Pascal

```
property AutoCheck: Boolean;
```

C++

```
__property Boolean AutoCheck;
```

Description

AutoCheck causes the checked property to toggle when the action executes. This allows the action's Checked property to remain in sync with the client's Checked property (or equivalent). If the client is a menu item (where the Checked property does

not necessarily toggle), AutoCheck is propagated to the AutoCheck property of the menu item.

See Also

Checked (see page 308)

Execute (see page 304)

AutoCheck (see page 16134)

32.1.3.2.2 TControlAction.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

32.1.3.2.3 TControlAction.Checked Property

Indicates whether client controls and menu items appear checked.

Pascal

```
property Checked: Boolean;
```

C++

```
__property Boolean Checked;
```

Description

Checked specifies the checked state for the action. The value of Checked is propagated to client controls and menu items.

Note: If the action has a GroupIndex value greater than 0, then setting Checked to true causes all other actions in the group to have their Checked property set to false.

Note: Use the AutoCheck property to ensure that the action's Checked property stays synchronized with client's Checked properties.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
```

```
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

GroupIndex (see page 311)

AutoCheck (see page 308)

32.1.3.2.4 TControlAction.DropdownMenu Property

Pascal

```
property DropdownMenu: TPopupMenu;
```

C++

```
__property TPopupMenu DropdownMenu;
```

Description

This is DropdownMenu, a member of class TControlAction.

32.1.3.2.5 TControlAction.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

32.1.3.2.6 TControlAction.EnableDropdown Property

Pascal

```
property EnableDropdown: Boolean;
```

C++

```
__property Boolean EnableDropdown;
```

Description

This is EnableDropdown, a member of class TControlAction.

32.1.3.2.7 TControlAction.GroupIndex Property

Indicates whether the action acts like a radio button with other actions in the same action list.

Pascal

```
property GroupIndex: Integer;
```

C++

```
__property int GroupIndex;
```

Description

GroupIndex is used to define groups of actions that act like radio buttons. When GroupIndex is greater than 0, it identifies the group to which the action belongs. When the Checked property of any action in that group is set to true, the Checked property of all other actions in the group is set to false. That is, only one action in the group can be checked at a time.

Note: All actions in a group must be listed by the same action list or action manager.

See Also

Checked (see page 308)

GroupIndex (see page 2360)

32.1.3.2.8 TControlAction.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

32.1.3.2.9 TControlAction.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

32.1.3.2.10 TControlAction.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

32.1.3.2.11 TControlAction.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

32.1.3.2.12 TControlAction.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an

image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

32.1.3.2.13 TControlAction.PopupMenu Property

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

This is PopupMenu, a member of class TControlAction.

32.1.3.2.14 TControlAction.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

32.1.3.2.15 TControlAction.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

32.1.3.2.16 TControlAction.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

32.1.3.3 TControlAction Events

The events of the TControlAction class are listed here.

Published Events

	Name	Description
	OnExecute (see page 5888)	Occurs when the client event that is linked to it fires.
	OnHint (see page 5890)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 5890)	Occurs when the application is idle or when the action list updates.

32.1.3.3.1 TControlAction.OnExecute Event

Occurs when the client event that is linked to it fires.

Pascal

```
property OnExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExecute;
```

Description

Write an OnExecute event handler when you want to respond when the user triggers the client object's default event (typically an OnClick event). For most target clients, OnExecute is triggered on a Click method, and is associated with the OnClick event.

OnExecute also occurs when the user types the shortcut (or one of the secondary shortcuts) associated with the action or its client.

Warning: If you assign an OnExecute event handler to a predefined action, the default behavior of that action will not occur.

C++ Examples:

```
/*
This example requires a TApplicationEvents, a TActionList
and a TListBox in the form.  Select the TApplicationEvents,
double click on the OnActionExecute event and add the
following code to the handler.  Create an TAction named
MyAction in a TActionList.  The TActionList must have no
OnExecute event handler, and the TAction has an OnExecute
event handler called MyActionExecute.  Then set an object's
(such as a button) Action to MyAction.  The Handled
parameter of the event handler returns false by default. If
the handler handles the event, it should change Handled to
true, thereby preventing further attempts to handle the
action.
*/
void __fastcall TAppEventForm::ApplicationEventsActionExecute(
      TBasicAction *Action, bool &Handled)
{
    lbOther->Items->Add("Event OnActionExecute");
}

void __fastcall TAppEventForm::ActionExecute(TObject *Sender)
{
    ShowMessage("Action executed");
}
/*
When a client component or control is clicked, the OnExecute
event occurs for it’s associated action. For example, the
following code illustrates the OnExecute event handler for
an action that toggles the visibility of a toolbar when the
action is executed. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TActionList must have no OnExecute event handler, and the
TAction has an OnExecute event handler called Action1Execute.
Then set an object's (such as a button or menu item) Action
to Action1.  Set the TToolbar color to something that will
make it obvious when it is visible.  Also remember to uncheck
Project|Options|Applications|Enable runtime themes.
*/
void __fastcall TMainForm::Action1Execute(TObject *Sender)
{
  // Toggle Toolbar1's visibility
  ToolBar1->Visible = !ToolBar1->Visible;
  if (ToolBar1->Visible)
    lbOther->Items->Add("Event Action1Execute: toolbar visible");
  else
    lbOther->Items->Add("Event Action1Execute: toolbar not visible");
}
```

Delphi Examples:

```
{
This example requires a TApplicationEvents, a TActionList
and a TListBox in the form.  Select the TApplicationEvents,
double click on the OnActionExecute event and add the
following code to the handler.  Create an TAction named
MyAction in a TActionList.  The TActionList must have no
OnExecute event handler, and the TAction has an OnExecute
event handler called MyActionExecute.  Then set an object's
(such as a button) Action to MyAction.  The Handled
parameter of the event handler returns false by default. If
the handler handles the event, it should change Handled to
true, thereby preventing further attempts to handle the
action.
}
```

```
procedure TMainForm.ApplicationEventsActionExecute(Action: TBasicAction;
  var Handled: Boolean);
begin
  lbOther.Items.Add('Event OnActionExecute');
end;

procedure TMainForm.MyActionExecute(Sender: TObject);
begin
  ShowMessage('Action executed');
end;
{
When a client component or control is clicked, the OnExecute
event occurs for it's associated action. For example, the
following code illustrates the OnExecute event handler for
an action that toggles the visibility of a toolbar when the
action is executed. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TActionList must have no OnExecute event handler, and the
TAction has an OnExecute event handler called Action1Execute.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Execute(Sender: TObject);
begin
    { Toggle Toolbar1's visibility }
    ToolBar1.Visible := not ToolBar1.Visible;
  if ToolBar1.Visible then
    lbOther.Items.Add('Event Action1Execute: toolbar visible')
  else
    lbOther.Items.Add('Event Action1Execute: toolbar not visible')
end;
```

See Also

OnClick (see page 5851)

ShortCut (see page 314)

32.1.3.3.2 TControlAction.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

32.1.3.3.3 TControlAction.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

32.1.4 TControlActionLink Class

TControlActionLink links action objects to clients that descend from TControl (see page 5724).

Class Hierarchy

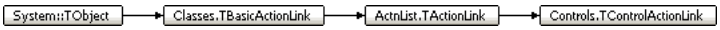

Pascal

```
TControlActionLink = class(TActionLink);
```

C++

```
class TControlActionLink : public TActionLink;
```

File

Controls

Description

TControlActionLink is designed to link actions to generic control clients. It overrides methods to determine whether the action and client are linked, and to ensure that action events are appropriate for properties and events of control clients.

TControlActionLink overrides methods to support linking the following properties and events with control client properties or events, most of which have the same name:

Caption

Enabled

Hint

Visible

OnExecute

Use TControlActionLink as a base class when creating an action link class that targets a more specific client type. Descendant classes of TControlActionLink already support certain branches of the VCL, such as TWinControlActionLink (see page 6233), which supports TWinControl (see page 6112) client types.

See Also

TWinControlActionLink (see page 6233)

TControl (see page 5724)

TBasicAction (see page 2707)

TControl (see page 5724)

32.1.4.1 TControlActionLink Members

The following tables list the members exposed by TControlActionLink.

Protected Fields

	Name	Description
	FAction (see page 2722)	This is FAction, a member of class TBasicActionLink.
	FClient (see page 5894)	This is FClient, a member of class TControlActionLink.

Protected Methods

	Name	Description
	AssignClient (see page 5895)	Assigns a client control that is linked to the associated action.
	Change (see page 2723)	This is Change, a member of class TBasicActionLink.
	DoShowHint (see page 5896)	Indicates whether the hint is shown, and modified.
	IsCaptionLinked (see page 5896)	Indicates whether the caption for the action and the client are linked.
	IsCheckedLinked (see page 269)	Indicates whether the Checked properties for the action and client are linked.
	IsDropdownMenuLinked (see page 5896)	This is IsDropdownMenuLinked, a member of class TControlActionLink.
	IsEnabledLinked (see page 5897)	Indicates whether the enabled state for the action and the client are linked.
	IsEnableDropdownLinked (see page 5897)	This is IsEnableDropdownLinked, a member of class TControlActionLink.
	IsGroupIndexLinked (see page 270)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsHelpContextLinked (see page 270)	Indicates whether the HelpContext properties for the action and client controls are linked.
	IsHelpLinked (see page 5897)	Indicates whether the help properties for the action and the client are linked.
	IsHintLinked (see page 5898)	Indicates whether the Hint property for the action and the client are linked.
	IsImageIndexLinked (see page 272)	Indicates whether the image index for the action and the client are linked.
	IsOnExecuteLinked (see page 5898)	Indicates whether the OnExecute event handler for the action and the client are linked.
	IsPopupMenuLinked (see page 5898)	This is IsPopupMenuLinked, a member of class TControlActionLink.
	IsShortCutLinked (see page 272)	Indicates whether the ShortCut properties for the action and client are linked.
	IsVisibleLinked (see page 5899)	Indicates whether the visible state for the action and the client are linked.
	SetAction (see page 2724)	This is SetAction, a member of class TBasicActionLink.
	SetAutoCheck (see page 274)	Introduces an interface for setting the AutoCheck property of a client.
	SetCaption (see page 5899)	Sets the caption of a control if the client is linked to the action for this event.
	SetChecked (see page 274)	Introduces an interface for setting the checked state of a client.
	SetDropdownMenu (see page 5899)	This is SetDropdownMenu, a member of class TControlActionLink.
	SetEnabled (see page 5900)	Sets the enabled state of a control if the client is linked to the action for this event.
	SetEnableDropdown (see page 5900)	This is SetEnableDropdown, a member of class TControlActionLink.
	SetGroupIndex (see page 275)	Introduces an interface for setting the GroupIndex property of a client.
	SetHelpContext (see page 5900)	Sets the HelpContext of a control if the client is linked to the action help properties.
	SetHelpKeyword (see page 5901)	Sets the HelpKeyword of a control if the client is linked to the action help properties.
	SetHelpType (see page 5901)	Sets the HelpType of a control if the client is linked to the action help properties.
	SetHint (see page 5901)	Sets the hint of a control if the client is linked to the action for this event.
	SetImageIndex (see page 278)	Introduces an interface for setting the image index for a client.
	SetOnExecute (see page 5902)	Sets the execution event handler for a control if the client is linked to the action for this event.
	SetPopupMenu (see page 5902)	This is SetPopupMenu, a member of class TControlActionLink.
	SetShortCut (see page 279)	Introduces an interface for setting the shortcut for a client.
	SetVisible (see page 5902)	Sets the visible state of a control if the client is linked to the action for this event.

Public Events

	Name	Description
	OnChange (see page 2725)	Occurs when an OnChange event is triggered on the action.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2721)	Instantiates and initializes a TBasicActionLink (see page 2720) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2722)	Destroys an instance of an action link object.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Execute (see page 2723)	Invokes the action's Execute method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Update (see page 2724)	Invokes the action's Update method.

Public Properties

	Name	Description
	Action (see page 2725)	Indicates which action the link connects to its client.

32.1.4.2 TControlActionLink Fields

The fields of the TControlActionLink class are listed here.

Protected Fields

	Name	Description
	FClient (see page 5894)	This is FClient, a member of class TControlActionLink.

32.1.4.2.1 TControlActionLink.FClient Field

Pascal

```
FClient: TControl;
```

C++

```
TControl * FClient;
```

Description

This is FClient, a member of class TControlActionLink.

32.1.4.3 TControlActionLink Methods

The methods of the TControlActionLink class are listed here.

Protected Methods

	Name	Description
	AssignClient (see page 5895)	Assigns a client control that is linked to the associated action.
	DoShowHint (see page 5896)	Indicates whether the hint is shown, and modified.
	IsCaptionLinked (see page 5896)	Indicates whether the caption for the action and the client are linked.
	IsDropdownMenuLinked (see page 5896)	This is IsDropdownMenuLinked, a member of class TControlActionLink.
	IsEnabledLinked (see page 5897)	Indicates whether the enabled state for the action and the client are linked.
	IsEnableDropdownLinked (see page 5897)	This is IsEnableDropdownLinked, a member of class TControlActionLink.
	IsHelpLinked (see page 5897)	Indicates whether the help properties for the action and the client are linked.
	IsHintLinked (see page 5898)	Indicates whether the Hint property for the action and the client are linked.
	IsOnExecuteLinked (see page 5898)	Indicates whether the OnExecute event handler for the action and the client are linked.
	IsPopupMenuLinked (see page 5898)	This is IsPopupMenuLinked, a member of class TControlActionLink.
	IsVisibleLinked (see page 5899)	Indicates whether the visible state for the action and the client are linked.
	SetCaption (see page 5899)	Sets the caption of a control if the client is linked to the action for this event.
	SetDropdownMenu (see page 5899)	This is SetDropdownMenu, a member of class TControlActionLink.
	SetEnabled (see page 5900)	Sets the enabled state of a control if the client is linked to the action for this event.
	SetEnableDropdown (see page 5900)	This is SetEnableDropdown, a member of class TControlActionLink.
	SetHelpContext (see page 5900)	Sets the HelpContext of a control if the client is linked to the action help properties.
	SetHelpKeyword (see page 5901)	Sets the HelpKeyword of a control if the client is linked to the action help properties.
	SetHelpType (see page 5901)	Sets the HelpType of a control if the client is linked to the action help properties.
	SetHint (see page 5901)	Sets the hint of a control if the client is linked to the action for this event.
	SetOnExecute (see page 5902)	Sets the execution event handler for a control if the client is linked to the action for this event.
	SetPopupMenu (see page 5902)	This is SetPopupMenu, a member of class TControlActionLink.
	SetVisible (see page 5902)	Sets the visible state of a control if the client is linked to the action for this event.

32.1.4.3.1 TControlActionLink.AssignClient Method

Assigns a client control that is linked to the associated action.

Pascal

```
procedure AssignClient(AClient: TObject); override;
```

C++

```
virtual __fastcall AssignClient(TObject * AClient);
```

Description

TControlActionLink (see page 5892) overrides AssignClient to ensure that the client for the action descends from TControl (see page 5724). Descendants of TControlActionLink (see page 5892) can further limit the type of client associated with the link.

See Also

Action (see page 5797)

32.1.4.3.2 TControlActionLink.DoShowHint Method

Indicates whether the hint is shown, and modified.

Pascal

```
function DoShowHint(var HintStr: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoShowHint(AnsiString HintStr);
```

Description

DoShowHint determines whether a hint is shown when the mouse moves across a control. DoShowHint also provides an opportunity to modify the hint, and/or append a shortcut to the end of the hint string. If either the application, the control, or the action veto showing the hint, then the hint is not shown. DoShowHint returns true by default. You can override this function to return false if the hint is not shown or modified, or if the shortcut is not appended.

See Also

ShowHint (see page 5840)

DoHint (see page 303)

32.1.4.3.3 TControlActionLink.IsCaptionLinked Method

Indicates whether the caption for the action and the client are linked.

Pascal

```
function IsCaptionLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCaptionLinked();
```

Description

IsCaptionLinked indicates whether the Caption property is linked to the appropriate client property. IsCaptionLinked returns true if the client requests that the action control this property, false otherwise.

See Also

Caption (see page 308)

Caption (see page 5806)

SetCaption (see page 5899)

32.1.4.3.4 TControlActionLink.IsDropdownMenuLinked Method

Pascal

```
function IsDropdownMenuLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsDropdownMenuLinked();
```

Description

This is IsDropdownMenuLinked, a member of class TControlActionLink.

32.1.4.3.5 TControlActionLink.IsEnabledLinked Method

Indicates whether the enabled state for the action and the client are linked.

Pascal

```
function IsEnabledLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsEnabledLinked();
```

Description

IsEnabledLinked indicates whether the Enabled property is linked to the appropriate client property. IsEnabledLinked returns true if the client requests that the action control this property, false otherwise.

See Also

Enabled (see page 310)

Enabled (see page 5821)

SetEnabled (see page 5900)

32.1.4.3.6 TControlActionLink.IsEnableDropdownLinked Method

Pascal

```
function IsEnableDropdownLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsEnableDropdownLinked();
```

Description

This is IsEnableDropdownLinked, a member of class TControlActionLink.

32.1.4.3.7 TControlActionLink.IsHelpLinked Method

Indicates whether the help properties for the action and the client are linked.

Pascal

```
function IsHelpLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsHelpLinked();
```

Description

IsHelpLinked indicates whether the HelpContext, HelpKeyword, and HelpType properties are linked to the appropriate client properties. IsHelpLinked returns true if the client requests that the action control all of these properties, false otherwise.

See Also

HelpKeyword (see page 311)

HelpContext (see page 311)

HelpType (see page 312)

HelpKeyword (see page 5826)

HelpContext (see page 5826)

HelpType (see page 5827)

SetHelpContext (see page 5900)

SetHelpKeyword (see page 5901)

SetHelpType (see page 5901)

32.1.4.3.8 TControlActionLink.IsHintLinked Method

Indicates whether the Hint property for the action and the client are linked.

Pascal

```
function IsHintLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsHintLinked();
```

Description

IsHintLinked indicates whether the Hint property is linked to the appropriate client property. IsHintLinked returns true if the client requests that the action control this property, false otherwise.

See Also

Hint (see page 312)

Hint (see page 5827)

SetHint (see page 5901)

32.1.4.3.9 TControlActionLink.IsOnExecuteLinked Method

Indicates whether the OnExecute event handler for the action and the client are linked.

Pascal

```
function IsOnExecuteLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsOnExecuteLinked();
```

Description

IsOnExecuteLinked indicates whether the OnExecute event handler is linked to the appropriate client event. IsOnExecuteLinked returns true if the client requests that the action control this event, false otherwise.

See Also

OnExecute (see page 2716)

OnClick (see page 5851)

SetOnExecute (see page 5902)

32.1.4.3.10 TControlActionLink.IsPopupMenuLinked Method

Pascal

```
function IsPopupMenuLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsPopupMenuLinked();
```

Description

This is IsPopupMenuLinked, a member of class TControlActionLink.

32.1.4.3.11 TControlActionLink.IsVisibleLinked Method

Indicates whether the visible state for the action and the client are linked.

Pascal

```
function IsVisibleLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsVisibleLinked();
```

Description

IsVisibleLinked indicates whether the Visible property is linked to the appropriate client property. IsVisibleLinked returns true if the client requests that the action control this property, false otherwise.

See Also

Visible (see page 314)

Visible (see page 5847)

SetVisible (see page 5902)

32.1.4.3.12 TControlActionLink.SetCaption Method

Sets the caption of a control if the client is linked to the action for this event.

Pascal

```
procedure SetCaption(const Value: string); override;
```

C++

```
virtual __fastcall SetCaption(const AnsiString Value);
```

Description

SetCaption sets the caption of a control, and is called only if the client is linked for this event. Value specifies the string to which the caption should be set.

See Also

Caption (see page 308)

Caption (see page 5806)

IsCaptionLinked (see page 5896)

32.1.4.3.13 TControlActionLink.SetDropdownMenu Method

Pascal

```
procedure SetDropdownMenu(Value: TPopupMenu); virtual;
```

C++

```
virtual __fastcall SetDropdownMenu(TPopupMenu Value);
```

Description

This is SetDropdownMenu, a member of class TControlActionLink.

32.1.4.3.14 TControlActionLink.SetEnabled Method

Sets the enabled state of a control if the client is linked to the action for this event.

Pascal

```
procedure SetEnabled(Value: Boolean); override;
```

C++

```
virtual __fastcall SetEnabled(Boolean Value);
```

Description

SetEnabled sets the enabled state of a control, and is called only if the client is linked for this event. Value specifies whether the enabled state should be set.

See Also

Enabled (see page 310)

Enabled (see page 5821)

IsEnabledLinked (see page 5897)

32.1.4.3.15 TControlActionLink.SetEnableDropdown Method

Pascal

```
procedure SetEnableDropdown(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetEnableDropdown(Boolean Value);
```

Description

This is SetEnableDropdown, a member of class TControlActionLink.

32.1.4.3.16 TControlActionLink.SetHelpContext Method

Sets the HelpContext of a control if the client is linked to the action help properties.

Pascal

```
procedure SetHelpContext(Value: THelpContext); override;
```

C++

```
virtual __fastcall SetHelpContext(THelpContext Value);
```

Description

SetHelpContext sets the HelpContext property of a control, and is called only if the client is linked for help properties. Value specifies the help context ID to which the HelpContext property should be set.

See Also

HelpContext (see page 311)

HelpContext (see page 5826)

IsHelpLinked (see page 5897)

32.1.4.3.17 TControlActionLink.SetHelpKeyword Method

Sets the HelpKeyword of a control if the client is linked to the action help properties.

Pascal

```
procedure SetHelpKeyword(const Value: string); override;
```

C++

```
virtual __fastcall SetHelpKeyword(const AnsiString Value);
```

Description

SetHelpKeyword sets the HelpKeyword property of a control, and is called only if the client is linked for help properties. Value specifies the keyword to which the HelpKeyword property should be set.

See Also

HelpKeyword (see page 311)

HelpKeyword (see page 5826)

IsHelpLinked (see page 5897)

32.1.4.3.18 TControlActionLink.SetHelpType Method

Sets the HelpType of a control if the client is linked to the action help properties.

Pascal

```
procedure SetHelpType(Value: THelpType); override;
```

C++

```
virtual __fastcall SetHelpType(THelpType Value);
```

Description

SetHelpType sets the HelpType property of a control, and is called only if the client is linked for help properties. Value indicates the value to which the HelpType property should be set.

See Also

HelpType (see page 312)

HelpType (see page 5827)

IsHelpLinked (see page 5897)

32.1.4.3.19 TControlActionLink.SetHint Method

Sets the hint of a control if the client is linked to the action for this event.

Pascal

```
procedure SetHint(const Value: string); override;
```

C++

```
virtual __fastcall SetHint(const AnsiString Value);
```

Description

SetHint sets the hint of a control, and is called only if the client is linked for this event. Value specifies the string to which the hint should be set.

See Also

Hint (see page 312)

Hint (see page 5827)

IsHintLinked (see page 5898)

DoShowHint (see page 5896)

32.1.4.3.20 TControlActionLink.SetOnExecute Method

Sets the execution event handler for a control if the client is linked to the action for this event.

Pascal

```
procedure SetOnExecute(Value: TNotifyEvent); override;
```

C++

```
virtual __fastcall SetOnExecute(TNotifyEvent Value);
```

Description

SetOnExecute sets the appropriate execution event handler of a control, and is called only if the client is linked for this event. Value specifies the procedure that should be used for this event.

See Also

OnExecute (see page 2716)

OnClick (see page 5851)

IsOnExecuteLinked (see page 5898)

32.1.4.3.21 TControlActionLink.SetPopupMenu Method

Pascal

```
procedure SetPopupMenu(Value: TPopupMenu); virtual;
```

C++

```
virtual __fastcall SetPopupMenu(TPopupMenu Value);
```

Description

This is SetPopupMenu, a member of class TControlActionLink.

32.1.4.3.22 TControlActionLink.SetVisible Method

Sets the visible state of a control if the client is linked to the action for this event.

Pascal

```
procedure SetVisible(Value: Boolean); override;
```

C++

```
virtual __fastcall SetVisible(Boolean Value);
```

Description

SetVisible sets the visible state of a control, and is called only if the client is linked for this event. Value specifies whether the visible state should be set.

See Also

Visible (see page 314)

Visible (see page 5847)

IsVisibleLinked (see page 5899)

32.1.5 TControlCanvas Class

TControlCanvas is a specialized canvas object associated with a control.

Class Hierarchy

Pascal

```
TControlCanvas = class(TCanvas);
```

C++

```
class TControlCanvas : public TCanvas;
```

File

Controls

Description

TControlCanvas provides a graphics context for controls that render their own images.

Not all controls need a canvas. Many controls rely on rely on underlying Windows screen objects to provide rendering logic.

TControlCanvas is descended from TCanvas, which provides many properties, events, and methods to aid in the creation of rich images. TControlCanvas adds properties and methods to interact with a control object, and to manage the device context associated with the control.

See Also

TCustomControl (see page 5908)

TGraphicControl (see page 6044)

32.1.5.1 TControlCanvas Members

The following tables list the members exposed by TControlCanvas.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 14450)	This is Changed, a member of class TCanvas.
	Changing (see page 14450)	This is Changing, a member of class TCanvas.
	CreateHandle (see page 5906)	This is CreateHandle, a member of class TControlCanvas.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.
	RequiredState (see page 14470)	This is RequiredState, a member of class TCanvas.

Public Events

	Name	Description
	OnChange (see page 14496)	Occurs when the image has just changed.
	OnChanging (see page 14496)	Occurs just before a change is made to the image.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Arc (see page 14448)	Draws an arc on the image along the perimeter of the ellipse bounded by the specified rectangle.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BrushCopy (see page 14449)	Copies a portion of a bitmap onto a rectangle on the canvas, replacing one of the colors of the bitmap with the brush of the canvas.
	Chord (see page 14451)	Draws a closed figure represented by the intersection of a line and an ellipse.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CopyRect (see page 14452)	Copies part of an image from another canvas into the canvas.
	Create (see page 14443)	Creates an instance of TCanvas (see page 14441).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 5906)	Destroys an instance of TControlCanvas (see page 5903).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Draw (see page 14454)	Renders the graphic specified by the Graphic parameter on the canvas at the location given by the coordinates (X, Y).
	DrawFocusRect (see page 14455)	Draws a rectangle in the style used to indicate that the object inside the rectangle has focus.
	Ellipse (see page 14457)	This is the overview for the Ellipse method overload.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FillRect (see page 14458)	Fills the specified rectangle on the canvas using the current brush.
	FloodFill (see page 14459)	Fills an area of the canvas using the current brush.
	FrameRect (see page 14460)	Draws a rectangle using the Brush (see page 14484) of the canvas to draw the border.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeHandle (see page 5906)	Releases the handle to a device context (HDC) used by the control canvas object.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	HandleAllocated (see page 14461)	Indicates whether the TCanvas (see page 14441) object has acquired a handle to a device context.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LineTo (see page 14462)	Draws a line on the canvas from PenPos (see page 14494) to the point specified by X and Y, and sets the pen position to (X, Y).
	Lock (see page 14463)	Prevents other threads from drawing on the canvas.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveTo (see page 14463)	Changes the current drawing position to the point (X,Y).
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Pie (see page 14464)	Draws a pie-shaped the section of the ellipse bounded by the rectangle (X1, Y1) and (X2, Y2) on the canvas.
	PolyBezier (see page 14464)	Draws a set of Bezier curves.
	PolyBezierTo (see page 14465)	Draws a set of Bezier curves and updates the value of PenPos (see page 14494).
	Polygon (see page 14466)	Draws a series of lines on the canvas connecting the points passed in and closing the shape by drawing a line from the last point to the first point.
	Polyline (see page 14467)	Draws a series of lines on the canvas with the current pen, connecting each of the points passed to it in Points.
	Rectangle (see page 14468)	This is the overview for the Rectangle method overload.
	Refresh (see page 14470)	Deselects the Pen (see page 14492), Brush (see page 14484), and Font (see page 14488) from the device context.
	RoundRect (see page 14471)	Draws a rectangle with rounded corners on the canvas.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	StretchDraw (see page 14472)	Draws the graphic specified by the Graphic parameter in the rectangle specified by the Rect parameter.
	TextExtent (see page 14474)	Returns the width and height, in pixels, of a string rendered in the current font.
	TextHeight (see page 14474)	Returns the height, in pixels, of a string rendered in the current font.
	TextOut (see page 14475)	Writes a string on the canvas, starting at the point (X,Y), and then updates the PenPos (see page 14494) to the end of the string.
	TextRect (see page 14480)	This is the overview for the TextRect method overload.
	TextWidth (see page 14481)	Returns the width, in pixels, of a string rendered in the current font.
	TryLock (see page 14483)	Locks the canvas if it is currently unlocked.
	Unlock (see page 14483)	Decreases the LockCount (see page 14492) property, allowing other threads to execute when LockCount (see page 14492) reaches 0.
	UpdateTextFlags (see page 5907)	This is UpdateTextFlags, a member of class TControlCanvas.

Public Properties

	Name	Description
R	CanvasOrientation (see page 14485)	Determines the orientation of the canvas as left-to-right or right-to-left.
R	ClipRect (see page 14486)	Read-only property that specifies the boundaries of the clipping rectangle.
	Control (see page 5907)	Specifies the control associated with the control canvas object.
	Handle (see page 14489)	Specifies the handle for this canvas.
R	LockCount (see page 14492)	Indicates the number of times the canvas has been locked to prevent interference from other threads.
	PenPos (see page 14494)	Specifies the current drawing position of the Pen (see page 14492).
	Pixels (see page 14494)	Specifies the color of the pixels within the current ClipRect (see page 14486).
	TextFlags (see page 14495)	Specifies how text is written to the canvas.

Published Properties

	Name	Description
	Brush (see page 14484)	Determines the color and pattern for filling graphical shapes and backgrounds.
	CopyMode (see page 14487)	Specifies how a graphical image is copied onto the canvas.
	Font (see page 14488)	Specifies the font to use when writing text on the image.
	Pen (see page 14492)	Specifies the kind of pen the canvas uses for drawing lines and outlining shapes.

32.1.5.2 TControlCanvas.Destroy Destructor

Destroys an instance of TControlCanvas (see page 5903).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TControlCanvas();
```

Description

Do not call Destroy directly in an application. The controls that use a TControlCanvas (see page 5903) object destroy it by calling Free from their own destructors. Free verifies that the TControlCanvas (see page 5903) object is not nil and only then calls Destroy.

Destroy releases the handle to a device context (HDC) the control canvas uses before freeing the memory associated with the object.

See Also

Free (see page 22300)

FreeHandle (see page 5906)

Handle (see page 14489)

32.1.5.3 TControlCanvas Methods

The methods of the TControlCanvas class are listed here.

Protected Methods

	Name	Description
	CreateHandle (see page 5906)	This is CreateHandle, a member of class TControlCanvas.

Public Methods

	Name	Description
	FreeHandle (see page 5906)	Releases the handle to a device context (HDC) used by the control canvas object.
	UpdateTextFlags (see page 5907)	This is UpdateTextFlags, a member of class TControlCanvas.

32.1.5.3.1 TControlCanvas.CreateHandle Method

Pascal

```
procedure CreateHandle; override;
```

C++

```
virtual __fastcall CreateHandle();
```

Description

This is CreateHandle, a member of class TControlCanvas.

32.1.5.3.2 TControlCanvas.FreeHandle Method

Releases the handle to a device context (HDC) used by the control canvas object.

Pascal

```
procedure FreeHandle;
```

C++

```
__fastcall FreeHandle();
```

Description

Call FreeHandle to free the HDC used by the control canvas. FreeHandle sets the Handle property to NULL. After calling FreeHandle, an HDC will be reacquired if any of the drawing methods are called.

Note: Setting the Handle property to NULL does not free the HDC. Set the Handle property to NULL rather than call FreeHandle if the HDC is release by the application.

See Also

Handle (see page 14489)

32.1.5.3.3 TControlCanvas.UpdateTextFlags Method

Pascal

```
procedure UpdateTextFlags;
```

C++

```
__fastcall UpdateTextFlags();
```

Description

This is UpdateTextFlags, a member of class TControlCanvas.

32.1.5.4 TControlCanvas Properties

The properties of the TControlCanvas class are listed here.

Public Properties

	Name	Description
	Control (see page 5907)	Specifies the control associated with the control canvas object.

32.1.5.4.1 TControlCanvas.Control Property

Specifies the control associated with the control canvas object.

Pascal

```
property Control: TControl;
```

C++

```
__property TControl * Control;
```

Description

Read Control to gain access to the control associated with the control canvas object. Descendants of TWinControl (see page 6112) that use a control canvas set the Control property to form the association. The control canvas object uses the Control property to get information from its associated control.

32.1.6 TCustomControl Class

Base class for controls that wrap Windows screen objects but perform their own rendering.

Class Hierarchy

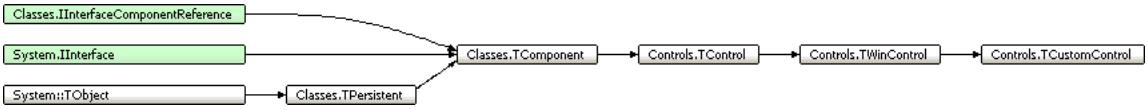

Pascal

```
TCustomControl = class(TWinControl);
```

C++

```
class TCustomControl : public TWinControl;
```

File

Controls

Description

In most descendants of TWinControl (see page 6112), the job of drawing the control's surface belongs to the underlying Windows screen object. If a control has visible features that cannot be rendered by a Windows screen object, it requires access to a canvas object so it can do its own drawing.

TCustomControl is one of two base classes for controls that draw their own surfaces. Because TCustomControl is descended from TWinControl (see page 6112), instances of its descendants can receive the focus and serve as containers.

Classes (see page 2685) of controls that perform their own screen rendering and do not need the features of TWinControl (see page 6112) should be descendants of TGraphicControl (see page 6044) instead of TCustomControl. TGraphicControl (see page 6044) objects can render themselves with less overhead, because they do not have underlying Windows screen objects.

TCustomControl objects have a Paint (see page 5920) method, which is called whenever the control needs to render its image. Custom controls paint themselves using the Canvas (see page 5921) property.

See Also

TCanvas (see page 14441)

TGraphicControl (see page 6044)

TWinControl (see page 6112)

32.1.6.1 TCustomControl Members

The following tables list the members exposed by TCustomControl.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.

	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.

	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.

	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.

	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 5920)	Renders the image of a custom control.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.

	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.

	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 5918)	Creates an instance of TCustomControl (see page 5908).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.

	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.

V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.

R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

32.1.6.2 TCustomControl.Create Constructor

Creates an instance of TCustomControl (see page 5908).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomControl(TComponent * AOwner);
```

Description

Calling Create constructs and initializes an instance of TCustomControl (see page 5908). However, you should never attempt to instantiate a TCustomControl (see page 5908). This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants. Controls (see page 5711) placed on forms at design time are created automatically.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes a TCanvas object for its Canvas (see page 5921) property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create (see page 6123)

Destroy (see page 5919)

Canvas (see page 5921)

Owner (see page 2807)

32.1.6.3 TCustomControl.Destroy Destructor

Destroys an instance of TCustomControl (see page 5908).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomControl();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the control is not nil, and only then calls Destroy.

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Destroys the TCanvas object in its Canvas (see page 5921) property.

Override Destroy to free any memory or resources allocated in the Create (see page 5918) method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

Free (see page 22300)

delete

Create (see page 5918)

32.1.6.4 TCustomControl Methods

The methods of the TCustomControl class are listed here.

Protected Methods

	Name	Description
	Paint (see page 5920)	Renders the image of a custom control.
	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.

32.1.6.4.1 TCustomControl.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; virtual;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl (see page 5908) does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow (see page 5920) method, after that method has supplied the Canvas (see page 5921) with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas (see page 5921)

PaintWindow (see page 5920)

GetDeviceContext (see page 6155)

ClipRect (see page 14486)

32.1.6.4.2 TCustomControl.PaintWindow Method

Renders the image of the control on a specified device context.

Pascal

```
procedure PaintWindow(DC: HDC); override;
```

C++

```
virtual __fastcall PaintWindow(HDC DC);
```

Description

PaintWindow is called automatically when a custom control receives a WM_PAINT message. PaintWindow assigns the device context specified by the DC parameter to the custom control's canvas, calls the Paint (see page 5920) method, and finally removes the device context. Override PaintWindow to provide additional responses to the WM_PAINT message. To draw the

image of the control, override the Paint (see page 5920) method instead.

See Also

Canvas (see page 5921)

Paint (see page 5920)

GetDeviceContext (see page 6155)

32.1.6.5 TCustomControl Properties

The properties of the TCustomControl class are listed here.

Protected Properties

	Name	Description
	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.

32.1.6.5.1 TCustomControl.Canvas Property

Specifies the TCanvas object that presents a drawing surface for the control.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control. Canvas encapsulates a Windows device context, providing all the tools and methods needed for drawing and painting.

Canvas is a protected property that is usually redeclared as public in descendants of TCustomControl (see page 5908).

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas,Rect.Left,Rect.Top,index);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(index) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TStringGrid *myStringGrid;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TStringGrid")
  {
    myStringGrid = (TStringGrid *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myStringGrid->Canvas->MoveTo(myStringGrid->ClientRect.Left, myStringGrid->ClientRect.Top);
    myStringGrid->Canvas->LineTo(myStringGrid->ClientRect.Right,
myStringGrid->ClientRect.Bottom);
    myStringGrid = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
```

```
if (ListBox1.Items.Objects[Index] is TStringGrid) then
begin
  myStringGrid:= (ListBox1.Items.Objects[Index] as TStringGrid);
  with myStringGrid.ClientRect do
  begin // make sure we are using the TControl's canvas and not the form's!
    myStringGrid.Canvas.MoveTo(Left, Top);
    myStringGrid.Canvas.LineTo(Right, Bottom);
  end;
myStringGrid:= nil;
end;
```

See Also

Paint (see page 5920)

32.1.7 TCustomControlAction Class

Class Hierarchy

Pascal

```
TCustomControlAction = class(TCustomAction);
```

C++

```
class TCustomControlAction : public TCustomAction;
```

File

Controls

Description

This is class Controls.TCustomControlAction.

32.1.7.1 TCustomControlAction Members

The following tables list the members exposed by TCustomControlAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.

	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	DropdownMenu (see page 5927)	This is DropdownMenu, a member of class TCustomControlAction.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	EnableDropdown (see page 5928)	This is EnableDropdown, a member of class TCustomControlAction.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	PopupMenu (see page 5928)	This is PopupMenu, a member of class TCustomControlAction.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Properties

	Name	Description
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

32.1.7.2 TCustomControlAction Properties

The properties of the TCustomControlAction class are listed here.

Public Properties

	Name	Description
	DropdownMenu (see page 5927)	This is DropdownMenu, a member of class TCustomControlAction.
	EnableDropdown (see page 5928)	This is EnableDropdown, a member of class TCustomControlAction.
	PopupMenu (see page 5928)	This is PopupMenu, a member of class TCustomControlAction.

32.1.7.2.1 TCustomControlAction.DropdownMenu Property

Pascal

```
property DropdownMenu: TPopupMenu;
```

C++

```
__property TPopupMenu DropdownMenu;
```

Description

This is DropdownMenu, a member of class TCustomControlAction.

32.1.7.2.2 TCustomControlAction.EnableDropdown Property

Pascal

```
property EnableDropdown: Boolean;
```

C++

```
__property Boolean EnableDropdown;
```

Description

This is EnableDropdown, a member of class TCustomControlAction.

32.1.7.2.3 TCustomControlAction.PopupMenu Property

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

This is PopupMenu, a member of class TCustomControlAction.

32.1.8 TCustomListControl Class

TCustomListControl is the base class for controls that display a list of items.

Class Hierarchy

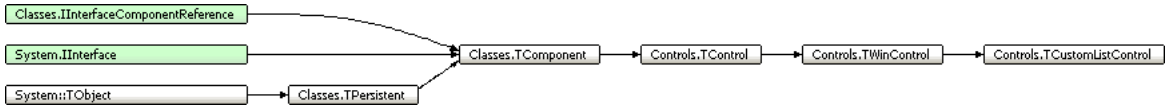

Pascal

```
TCustomListControl = class(TWinControl);
```

C++

```
class TCustomListControl : public TWinControl;
```

File

Controls

Description

TCustomListControl is the common base class for controls that represent a list of items. Its descendants include combo boxes, list boxes, and list views. TCustomListControl introduces a common set of properties and methods for working with the items in the list. This includes the ability to

Add items to the list.

Delete items from the list.

Select items in the list.

Copy items to another TCustomListControl descendant.

Rearrange the order of items.

TCustomListControl cannot be instantiated directly.

See Also

TComboBox (see page 21064)

TComboBoxEx (see page 3463)

TColorBox (see page 12182)

TListBox (see page 21598)

TListView (see page 4285)

32.1.8.1 TCustomListControl Members

The following tables list the members exposed by TCustomListControl.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.

	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.

	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.

	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.

	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
A	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.

	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.

	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
A	AddItem (see page 5940)	Adds an item to the list control.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 5940)	Deletes all of the items from the list control.
A	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
A	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	DeleteSelected (see page 5941)	Deletes all the selected items.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.

	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.

	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
A	SelectAll (see page 5943)	Selects all items or all text in the selected item.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.

	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

32.1.8.2 TCustomListControl Methods

The methods of the TCustomListControl class are listed here.

Protected Methods

	Name	Description
A	GetCount (see page 5942)	Returns the number of items in the list.
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
A	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.

Public Methods

	Name	Description
A	AddItem (see page 5940)	Adds an item to the list control.
A	Clear (see page 5940)	Deletes all of the items from the list control.
A	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
A	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
A	DeleteSelected (see page 5941)	Deletes all the selected items.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
A	SelectAll (see page 5943)	Selects all items or all text in the selected item.

32.1.8.2.1 TCustomListControl.AddItem Method

Adds an item to the list control.

Pascal

```
procedure AddItem(Item: String; AObject: TObject); virtual; abstract;
```

C++

```
virtual __fastcall AddItem(AnsiString Item, TObject * AObject) = 0;
```

Description

AddItem adds an item to the end of the list, possibly associating it with a specified object.

Item is the text that is the item's value. This is the string that appears for the item in the list control.

AObject is an object that you can associated with the item. Note that the list control does not take ownership of this object. That is, if you delete the item, the corresponding object is not freed.

32.1.8.2.2 TCustomListControl.Clear Method

Deletes all of the items from the list control.

Pascal

```
procedure Clear; virtual; abstract;
```

C++

```
virtual __fastcall Clear() = 0;
```

Description

Clear empties the list control so that it contains no items.

See Also

DeleteSelected (see page 5941)

32.1.8.2.3 TCustomListControl.ClearSelection Method

Removes the selection, leaving all items unselected.

Pascal

```
procedure ClearSelection; virtual; abstract;
```

C++

```
virtual __fastcall ClearSelection() = 0;
```

Description

Call ClearSelection to deselect any selected items in the list control. ClearSelection removes the selection, but leaves the items in the list.

See Also

Clear (see page 5940)

CopySelection (see page 5941)

DeleteSelected (see page 5941)

MoveSelection (see page 5942)

SelectAll (see page 5943)

ItemIndex (see page 5944)

32.1.8.2.4 TCustomListControl.CopySelection Method

Copies the selected item or items to another TCustomListControl (see page 5928) descendant.

Pascal

```
procedure CopySelection(Destination: TCustomListControl); virtual; abstract;
```

C++

```
virtual __fastcall CopySelection(TCustomListControl Destination) = 0;
```

Description

CopySelection adds all selected items to another TCustomListControl (see page 5928) descendant.

Destination is the list control that receives the items.

See Also

ClearSelection (see page 5940)

DeleteSelected (see page 5941)

MoveSelection (see page 5942)

SelectAll (see page 5943)

ItemIndex (see page 5944)

AddItem (see page 5940)

32.1.8.2.5 TCustomListControl.DeleteSelected Method

Deletes all the selected items.

Pascal

```
procedure DeleteSelected; virtual; abstract;
```

C++

```
virtual __fastcall DeleteSelected() = 0;
```

Description

Call DeleteSelected to remove all selected items from the list control. DeleteSelected deletes only the item or items that are selected.

See Also

Clear (see page 5940)

CopySelection (see page 5941)

ClearSelection (see page 5940)

MoveSelection (see page 5942)

SelectAll (see page 5943)

ItemIndex (see page 5944)

32.1.8.2.6 TCustomListControl.GetCount Method

Returns the number of items in the list.

Pascal

```
function GetCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetCount() = 0;
```

Description

Call GetCount to determine the number of items in the list control.

32.1.8.2.7 TCustomListControl.GetItemIndex Method

Returns the value of the ItemIndex (see page 5944) property.

Pascal

```
function GetItemIndex: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetItemIndex() = 0;
```

Description

Override GetItemIndex to provide the read implementation for the ItemIndex (see page 5944) property. GetItemIndex returns the index of the selected item, where the first item in the list has index 0, the second item has index 1, and so on. If no item is selected, GetItemIndex returns -1. If the list control supports multiple selected items, GetItemIndex returns the index of the selected item that has focus.

See Also

ItemIndex (see page 5944)

SetItemIndex (see page 5943)

32.1.8.2.8 TCustomListControl.MoveSelection Method

Moves all selected items to another list control.

Pascal

```
procedure MoveSelection(Destination: TCustomListControl); virtual;
```

C++

```
virtual __fastcall MoveSelection(TCustomListControl Destination);
```

Description

Call MoveSelection to copy all selected items to another list control and then delete them from this list control.

See Also

Clear (see page 5940)

CopySelection (see page 5941)

ClearSelection (see page 5940)

DeleteSelected (see page 5941)

SelectAll (see page 5943)

ItemIndex (see page 5944)

32.1.8.2.9 TCustomListControl.SelectAll Method

Selects all items or all text in the selected item.

Pascal

```
procedure SelectAll; virtual; abstract;
```

C++

```
virtual __fastcall SelectAll() = 0;
```

Description

SelectAll performs different tasks, depending on the custom list control descendant. In list controls that allow multiple items to be selected, SelectAll selects all items in the list. In list controls that allow only one item to be selected, SelectAll selects the entire text of the current item.

See Also

Clear (see page 5940)

CopySelection (see page 5941)

ClearSelection (see page 5940)

DeleteSelected (see page 5941)

MoveSelection (see page 5942)

ItemIndex (see page 5944)

32.1.8.2.10 TCustomListControl.SetItemIndex Method

Sets the value of the ItemIndex (see page 5944) property.

Pascal

```
procedure SetItemIndex(const Value: Integer); virtual; abstract; overload;
```

C++

```
virtual __fastcall SetItemIndex(const int Value) = 0;
```

Description

Override SetItemIndex to provide the write implementation for the ItemIndex (see page 5944) property.

Value represents the index of the item to select, where the first item in the list has index 0, the second item has index 1, and so on. If Value is –1, SetItemIndex deselects all items (the same as the ClearSelection (see page 5940) method).

See Also

ItemIndex (see page 5944)

GetItemIndex (see page 5942)

ClearSelection (see page 5940)

32.1.8.3 TCustomListControl Properties

The properties of the TCustomListControl class are listed here.

Public Properties

	Name	Description
	ItemIndex (see page 5944)	Specifies the index of the selected item.

32.1.8.3.1 TCustomListControl.ItemIndex Property

Specifies the index of the selected item.

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

Description

Read ItemIndex to determine which item is selected. The first item in the list has index 0, the second item has index 1, and so on. If no item is selected, the value of ItemIndex is -1. If the list control supports multiple selected items, ItemIndex is the index of the selected item that has focus.

Set ItemIndex programmatically to select an item by passing in the index value.

Note: To implement the ItemIndex property in a TCustomListControl (see page 5928) descendant, override the protected GetItemIndex (see page 5942) and SetItemIndex (see page 5943) methods.

See Also

GetItemIndex (see page 5942)

SetItemIndex (see page 5943)

32.1.9 TCustomMultiSelectListControl Class

TCustomMultiSelectListControl is the base class for list controls that allow multiple items to be selected.

Class Hierarchy

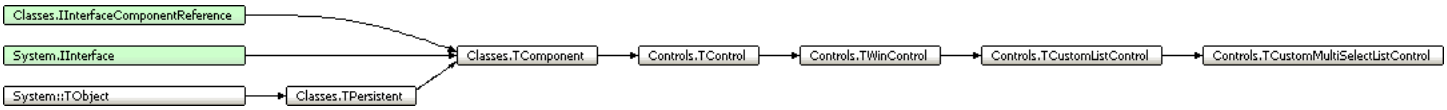

Pascal

```
TCustomMultiSelectListControl = class(TCustomListControl);
```

C++

```
class TCustomMultiSelectListControl : public TCustomListControl;
```

File

Controls

Description

TCustomMultiSelectListControl is the common base class for controls that represent a list of items where the user can select one or more of those items. Its descendants include list boxes and list views. TCustomMultiSelectListControl implements a common set of properties and methods for working with the items in the list. This includes the ability to

Specify whether the control allows one or multiple items to be selected.

Determine the number of selected items.

Add items to the list.

Delete items from the list.

Select items in the list.

Copy items to another list control.

Rearrange the order of items.

TCustomMultiSelectListControl is an abstract class and cannot be directly instantiated.

See Also

TListBox (see page 21598)

TListView (see page 4285)

32.1.9.1 TCustomMultiSelectListControl Members

The following tables list the members exposed by TCustomMultiSelectListControl.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FMultiSelect (see page 5956)	This is FMultiSelect, a member of class TCustomMultiSelectListControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.

	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.

	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
A	GetSelCount (see page 5956)	Returns the value of the SelCount (see page 5957) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.

V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
A	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.

	SetMultiSelect (see page 5956)	Sets the value of the MultiSelect (see page 5957) property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.

	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
A	AddItem (see page 5940)	Adds an item to the list control.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 5940)	Deletes all of the items from the list control.
A	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
A	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.

V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	DeleteSelected (see page 5941)	Deletes all the selected items.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.

	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
A	SelectAll (see page 5943)	Selects all items or all text in the selected item.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.

	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.

R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	MultiSelect (see page 5957)	Specifies whether the user can select more than one item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	SelCount (see page 5957)	Indicates the number of selected items.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

32.1.9.2 TCustomMultiSelectListControl Fields

The fields of the TCustomMultiSelectListControl class are listed here.

Protected Fields

	Name	Description
	FMultiSelect (see page 5956)	This is FMultiSelect, a member of class TCustomMultiSelectListControl.

32.1.9.2.1 TCustomMultiSelectListControl.FMultiSelect Field

Pascal

```
FMultiSelect: Boolean;
```

C++

```
Boolean FMultiSelect;
```

Description

This is FMultiSelect, a member of class TCustomMultiSelectListControl.

32.1.9.3 TCustomMultiSelectListControl Methods

The methods of the TCustomMultiSelectListControl class are listed here.

Protected Methods

	Name	Description
	GetSelCount (see page 5956)	Returns the value of the SelCount (see page 5957) property.
	SetMultiSelect (see page 5956)	Sets the value of the MultiSelect (see page 5957) property.

32.1.9.3.1 TCustomMultiSelectListControl.GetSelCount Method

Returns the value of the SelCount (see page 5957) property.

Pascal

```
function GetSelCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetSelCount() = 0;
```

Description

Override GetSelCount to provide the read implementation for the SelCount (see page 5957) property. GetSelCount returns the number of items that are currently selected.

See Also

SelCount (see page 5957)

32.1.9.3.2 TCustomMultiSelectListControl.SetMultiSelect Method

Sets the value of the MultiSelect (see page 5957) property.

Pascal

```
procedure SetMultiSelect(Value: Boolean); virtual; abstract;
```

C++

```
virtual __fastcall SetMultiSelect(Boolean Value) = 0;
```

Description

Override SetMultiSelect to provide the write implementation for the MultiSelect (see page 5957) property.

Value is true if the control should allow the user to select more than one item at a time, false if the control should support at most one selected item.

See Also

MultiSelect (see page 5957)

32.1.9.4 TCustomMultiSelectListControl Properties

The properties of the TCustomMultiSelectListControl class are listed here.

Public Properties

	Name	Description
	MultiSelect (see page 5957)	Specifies whether the user can select more than one item.
R	SelCount (see page 5957)	Indicates the number of selected items.

32.1.9.4.1 TCustomMultiSelectListControl.MultiSelect Property

Specifies whether the user can select more than one item.

Pascal

```
property MultiSelect: Boolean;
```

C++

```
__property Boolean MultiSelect;
```

Description

Use MultiSelect to specify whether the user can select more than one item. When MultiSelect is true, the user can select multiple items in the control, and the SelCount (see page 5957) property indicates the number of selected items. When MultiSelect is false, the user can only select one item, which is indicated by the ItemIndex (see page 5944) property.

Note: To implement the MultiSelect property in a TCustomMultiSelectListControl (see page 5944) descendant, override the protected SetMultiSelect (see page 5956) method.

See Also

SelCount (see page 5957)

ItemIndex (see page 5944)

32.1.9.4.2 TCustomMultiSelectListControl.SelCount Property

Indicates the number of selected items.

Pascal

```
property SelCount: Integer;
```

C++

```
__property int SelCount;
```

Description

Read SelCount to determine the number of items that are currently selected. If the value of the MultiSelect (see page 5957)

property is false, SelCount is always -1. If MultiSelect (see page 5957) is true, the value of SelCount can be greater than 1 as well.

Note: To implement the SelCount property in a TCustomMultiSelectListControl (see page 5944) descendant, override the protected GetSelCount (see page 5956) method.

See Also

MultiSelect (see page 5957)

ItemIndex (see page 5944)

32.1.10 TCustomPanningWindow Class

This is an abstract base class that is used as an ancestor for users to implement a custom panning window.

Class Hierarchy

Pascal

```
TCustomPanningWindow = class(TCustomControl);
```

C++

```
class TCustomPanningWindow : public TCustomControl;
```

File

Controls

Description

To create a custom panning window, create a sub-class of this class and over-ride the method for the part of the implementation that you want to customize.

See Also

TCustomPanningWindow

GetIsPanning

StartPanning

StopPanning

32.1.10.1 TCustomPanningWindow Members

The following tables list the members exposed by TCustomPanningWindow.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.

	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.

	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.

	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.

	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 5920)	Renders the image of a custom control.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.

	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.

	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 5918)	Creates an instance of TCustomControl (see page 5908).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
A	GetIsPanning (see page 5969)	Returns whether or not the mouse is actively panning.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
A	StartPanning (see page 5969)	Tells the panning window to start the panning operation.
A	StopPanning (see page 5969)	Tells the panning window to stop the panning operation.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.

R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.

	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

32.1.10.2 TCustomPanningWindow Methods

The methods of the TCustomPanningWindow class are listed here.

Public Methods

	Name	Description
	GetIsPanning (see page 5969)	Returns whether or not the mouse is actively panning.
	StartPanning (see page 5969)	Tells the panning window to start the panning operation.
	StopPanning (see page 5969)	Tells the panning window to stop the panning operation.

32.1.10.2.1 TCustomPanningWindow.GetIsPanning Method

Returns whether or not the mouse is actively panning.

Pascal

```
function GetIsPanning: Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean GetIsPanning() = 0;
```

Description

This method brings back a boolean value. True means that the mouse is actively panning; False indicates that no panning is occurring.

32.1.10.2.2 TCustomPanningWindow.StartPanning Method

Tells the panning window to start the panning operation.

Pascal

```
function StartPanning(AHandle: THandle; AControl: TControl): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean StartPanning(THandle AHandle, TControl * AControl) = 0;
```

Description

Include this method in your code where you want the panning window to start the panning operation.

32.1.10.2.3 TCustomPanningWindow.StopPanning Method

Tells the panning window to stop the panning operation.

Pascal

```
procedure StopPanning; virtual; abstract;
```

C++

```
virtual __fastcall StopPanning() = 0;
```

Description

Add this method to your code where you want the panning window to stop the panning operation.

32.1.11 TCustomTransparentControl Class

Base class for transparent controls.

Class Hierarchy

Pascal

```
TCustomTransparentControl = class(TCustomControl);
```

C++

```
class TCustomTransparentControl : public TCustomControl;
```

File

Controls

Description

Use TCustomTransparentControl as the base class for drawing transparent controls.

32.1.11.1 TCustomTransparentControl Members

The following tables list the members exposed by TCustomTransparentControl.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.

	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 5981)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.

	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateControlsUnderneath (see page 5982)	This is InvalidateControlsUnderneath, a member of class TCustomTransparentControl.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.

	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 5920)	Renders the image of a custom control.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMNCHitTest (see page 5982)	This is WMNCHitTest, a member of class TCustomTransparentControl.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.

R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.

	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 5980)	Creates an instance of TCustomControl.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.

	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5982)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.

	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.

	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	InterceptMouse (see page 5983)	Specifies whether the control is transparent to the mouse.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

32.1.11.2 TCustomTransparentControl.Create Constructor

Creates an instance of TCustomControl.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomTransparentControl(TComponent * AOwner);
```

Description

Calling Create constructs and initializes an instance of TCustomControl. However, you should never attempt to instantiate a TCustomControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants. Controls placed on forms at design time are created automatically.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes a TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create (see page 6123)

Destroy (see page 5919)

Canvas (see page 5921)

Owner (see page 2807)

32.1.11.3 TCustomTransparentControl Methods

The methods of the TCustomTransparentControl class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 5981)	Initializes a window-creation parameter data structure.
	InvalidateControlsUnderneath (see page 5982)	This is InvalidateControlsUnderneath, a member of class TCustomTransparentControl.
	WMNCHitTest (see page 5982)	This is WMNCHitTest, a member of class TCustomTransparentControl.

Public Methods

	Name	Description
	Invalidate (see page 5982)	Schedules a control repaint.

32.1.11.3.1 TCustomTransparentControl.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

32.1.11.3.2 TCustomTransparentControl.Invalidate Method

Schedules a control repaint.

Pascal

```
procedure Invalidate; override;
```

C++

```
virtual __fastcall Invalidate();
```

Description

Invalidate informs a control that its entire surface needs to be repainted. Calling Invalidate can prevent flicker caused by a series of partial repaints. There is no performance penalty for calling Invalidate multiple times before the control is actually repainted.

The actual repaint does not occur until the control is updated. To force an immediate repaint, call Repaint instead.

See Also

Repaint (see page 6174)

Update (see page 6183)

32.1.11.3.3 TCustomTransparentControl.InvalidateControlsUnderneath Method

Pascal

```
procedure InvalidateControlsUnderneath;
```

C++

```
__fastcall InvalidateControlsUnderneath();
```

Description

This is InvalidateControlsUnderneath, a member of class TCustomTransparentControl.

32.1.11.3.4 TCustomTransparentControl.WMNCHitTest Method

Pascal

```
procedure WMNCHitTest(var Message: TWMNCHitTest); message WM_NCHITTEST;
```

C++

```
__fastcall WMNCHitTest(TWMNCHitTest Message);
```

Description

This is WMNCHitTest, a member of class TCustomTransparentControl.

32.1.11.4 TCustomTransparentControl Properties

The properties of the TCustomTransparentControl class are listed here.

Public Properties

	Name	Description
	InterceptMouse (see page 5983)	Specifies whether the control is transparent to the mouse.

32.1.11.4.1 TCustomTransparentControl.InterceptMouse Property

Specifies whether the control is transparent to the mouse.

Pascal

```
property InterceptMouse: Boolean;
```

C++

```
__property Boolean InterceptMouse;
```

Description

If set to True, the transparent control captures mouse clicks. If set to False, the transparent control ignores mouse clicks. By default, this property is set to False.

32.1.12 TDockTree Class

TDockTree is the default docking manager for TWinControl (see page 6112).

Class Hierarchy

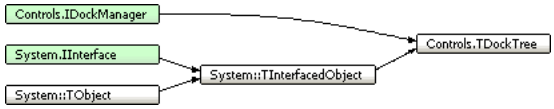

Pascal

```
TDockTree = class(TInterfacedObject, IDockManager);
```

C++

```
class TDockTree : public TInterfacedObject, public IDockManager;
```

File

Controls

Description

TDockTree handles the tasks of inserting and removing controls when they are docked to or undocked from a windowed control, including details such as positioning controls and any painting that is associated with docking operations. TDockTree implements the type of docking zones that appear in the IDE, with splitters that let users resize the docking zones, and grab regions where users must click to start an undock operation.

If an application does not explicitly assign a value to a windowed control's DockManager property, then the windowed control uses TDockTree to handle docking operations (if the UseDockManager property is true).

TDockTree implements the IDockManager (see page 6238) interface.

See Also

DefaultDockTreeClass (see page 6321)

TWinControl (see page 6112)

TWinControl (see page 6112)

IDockManager (see page 6238)

32.1.12.1 TDockTree Members

The following tables list the members exposed by TDockTree.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
V	AdjustDockRect (see page 5987)	Adjusts the boundaries of the docking zone where a specified control is docked.
V	AdjustFrameRect (see page 5988)	This is AdjustFrameRect, a member of class TDockTree.
	BeginUpdate (see page 5988)	Responds when the docked controls are about to change.
	EndUpdate (see page 5988)	Responds when the docked controls have finished changing.
	FindControlAtPos (see page 5989)	This is FindControlAtPos, a member of class TDockTree.
	GetControlBounds (see page 5989)	Returns the dimensions allotted to a control that is already docked.
V	HitTest (see page 5990)	Indicates which docked control is at a specified mouse position.
V	InsertControl (see page 5990)	Handles the docking of a control into the associated DockSite (see page 5996).
V	LoadFromStream (see page 5991)	Creates docking zones for a set of docked controls read from a stream.
V	MouseDown (see page 5991)	This is MouseDown, a member of class TDockTree.
V	MouseMove (see page 5991)	This is MouseMove, a member of class TDockTree.
V	MouseUp (see page 5992)	This is MouseUp, a member of class TDockTree.
V	PaintDockFrame (see page 5992)	Paints the elements of a docking zone that surround a docked control.
V	PositionDockRect (see page 5993)	Computes the dimensions of a new docking zone.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
V	ReferenceFromOrient (see page 5993)	This is ReferenceFromOrient, a member of class TDockTree.
V	RemoveControl (see page 5993)	Handles the undocking of a control from the associated DockSite (see page 5996).
V	ResetBounds (see page 5994)	Recomputes the layout of docked controls.
V	SaveToStream (see page 5994)	Saves a list of the set of docked controls and their positions to a stream.
	SetReplacingControl (see page 5995)	Prepares the dock tree for replacing one docked control with another.
V	ShowHint (see page 5995)	This is ShowHint, a member of class TDockTree.
	UpdateAll (see page 5995)	Makes any internal adjustments to reflect the current state of the docked controls.
V	WndProc (see page 5996)	This is WndProc, a member of class TDockTree.
V	ZoneCaptionHitTest (see page 5996)	This is ZoneCaptionHitTest, a member of class TDockTree.

Protected Properties

	Name	Description
	DockSite (see page 5996)	Specifies the windowed control that uses this dock tree to manage docking operations.
	RelativeSizes (see page 5997)	This is RelativeSizes, a member of class TDockTree.

R	TopZone (see page 5997)	This is TopZone, a member of class TDockTree.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 5985)	Creates and initializes a TDockTree (see page 5983) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5986)	Frees the memory associated with the TDockTree (see page 5983) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	PaintSite (see page 5992)	Paints the elements that surround all docked controls.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

32.1.12.2 TDockTree.Create Constructor

Creates and initializes a TDockTree (see page 5983) object.

Pascal

```
constructor Create(DockSite: TWinControl); virtual;
```

C++

```
virtual __fastcall TDockTree(TWinControl * DockSite);
```

Description

Createinstantiates a new TDockTree (see page 5983) object to manage docking operations for the control specified by the DockSite (see page 5996) parameter. Typically, applications do not create TDockTree (see page 5983) objects. Instead, the windowed control that is the value of the DockSite (see page 5996) parameter creates the dock tree when an application sets its UseDockManager property to true.

DockSite (see page 5996) is the windowed control that uses the dock tree to manage the docking of child controls. It becomes the value of the DockSite (see page 5996) property.

Createinitializes the new dock tree to reflect any controls currently docked in DockSite (see page 5996). It then replaces the WindowProc property of DockSite (see page 5996) with a method that handles all docking-related messages, passing any other messages on to DockSite (see page 5996)'s previous window procedure.

See Also

UseDockManager (see page 6212)

DockManager (see page 6202)

DefaultDockTreeClass (see page 6321)

DockSite (see page 5996)

WindowProc (see page 5872)

32.1.12.3 TDockTree.Destroy Destructor

Frees the memory associated with the TDockTree (see page 5983) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDockTree();
```

Description

Do not explicitly destroy TDockTree (see page 5983) objects. As interfaced objects, dock trees are destroyed automatically when the reference count of their IDockManager (see page 6238) interface drops to zero.

Destroy detaches the dock tree from the windowed control specified by the DockSite (see page 5996) property, restoring its WindowProc property to the value it had when it called the TDockTree (see page 5983) constructor.

See Also

RefCount (see page 22281)

Create (see page 5985)

DockSite (see page 5996)

WindowProc (see page 5872)

32.1.12.4 TDockTree Methods

The methods of the TDockTree class are listed here.

Protected Methods

	Name	Description
	AdjustDockRect (see page 5987)	Adjusts the boundaries of the docking zone where a specified control is docked.
	AdjustFrameRect (see page 5988)	This is AdjustFrameRect, a member of class TDockTree.
	BeginUpdate (see page 5988)	Responds when the docked controls are about to change.
	EndUpdate (see page 5988)	Responds when the docked controls have finished changing.
	FindControlAtPos (see page 5989)	This is FindControlAtPos, a member of class TDockTree.
	GetControlBounds (see page 5989)	Returns the dimensions allotted to a control that is already docked.
	HitTest (see page 5990)	Indicates which docked control is at a specified mouse position.
	InsertControl (see page 5990)	Handles the docking of a control into the associated DockSite (see page 5996).
	LoadFromStream (see page 5991)	Creates docking zones for a set of docked controls read from a stream.
	MouseDown (see page 5991)	This is MouseDown, a member of class TDockTree.
	MouseMove (see page 5991)	This is MouseMove, a member of class TDockTree.
	MouseUp (see page 5992)	This is MouseUp, a member of class TDockTree.
	PaintDockFrame (see page 5992)	Paints the elements of a docking zone that surround a docked control.
	PositionDockRect (see page 5993)	Computes the dimensions of a new docking zone.
	ReferenceFromOrient (see page 5993)	This is ReferenceFromOrient, a member of class TDockTree.
	RemoveControl (see page 5993)	Handles the undocking of a control from the associated DockSite (see page 5996).
	ResetBounds (see page 5994)	Recomputes the layout of docked controls.
	SaveToStream (see page 5994)	Saves a list of the set of docked controls and their positions to a stream.
	SetReplacingControl (see page 5995)	Prepares the dock tree for replacing one docked control with another.
	ShowHint (see page 5995)	This is ShowHint, a member of class TDockTree.
	UpdateAll (see page 5995)	Makes any internal adjustments to reflect the current state of the docked controls.
	WndProc (see page 5996)	This is WndProc, a member of class TDockTree.
	ZoneCaptionHitTest (see page 5996)	This is ZoneCaptionHitTest, a member of class TDockTree.

Public Methods

	Name	Description
	PaintSite (see page 5992)	Paints the elements that surround all docked controls.

32.1.12.4.1 TDockTree.AdjustDockRect Method

Adjusts the boundaries of the docking zone where a specified control is docked.

Pascal

```
procedure AdjustDockRect(Control: TControl; var ARect: TRect); virtual;
```

C++

```
virtual __fastcall AdjustDockRect(TControl * Control, TRect ARect);
```

Description

Applications can't call the protected AdjustDockRect method. It is called automatically to make any final adjustments to the rectangle where a client control is docked.

Control is the control that is docked.

On entry, ARect indicates the boundaries of Control when it is docked to the windowed control specified by the DockSite (see page 5996) property. AdjustDockRect changes this to make any final adjustments to the space in which Control is docked.

In TDockTree (see page 5983), AdjustDockRect expands the ARect parameter to make room for the "grab region" around a docked control. (The grab region is indicated by a double bevel, and is where the user can click with the mouse to start an undocking operation.) AdjustDockRect checks the Align property of the dock site to determine whether the grab region should appear on the side or the top of the docked control.

See Also

Align (see page 5798)

DockSite (see page 5996)

32.1.12.4.2 TDockTree.AdjustFrameRect Method

Pascal

```
procedure AdjustFrameRect(Control: TControl; var ARect: TRect); virtual;
```

C++

```
virtual __fastcall AdjustFrameRect(TControl * Control, TRect ARect);
```

Description

This is AdjustFrameRect, a member of class TDockTree.

32.1.12.4.3 TDockTree.BeginUpdate Method

Responds when the docked controls are about to change.

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

BeginUpdate implements the BeginUpdate method of the IDockManager (see page 6238) interface. This method is called before controls are docked or undocked. For example, TDockTree (see page 5983) calls BeginUpdate internally from the LoadFromStream (see page 5991) method, before docking the controls it reads from a stream.

Once all docking or undocking is complete, the EndUpdate (see page 5988) method is called so that the dock tree can make any final adjustments. Every call to BeginUpdate should be paired with a call to EndUpdate (see page 5988). BeginUpdate/EndUpdate (see page 5988) pairs can be nested.

See Also

EndUpdate (see page 5988)

LoadFromStream (see page 5991)

32.1.12.4.4 TDockTree.EndUpdate Method

Responds when the docked controls have finished changing.

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

EndUpdate implements the EndUpdate method of the IDockManager (see page 6238) interface. This method is called after controls are docked or undocked. For example, TDockTree (see page 5983) calls EndUpdate internally from the LoadFromStream (see page 5991) method, after docking the controls it reads from a stream.

EndUpdate checks that every previous call to the BeginUpdate (see page 5988) method has been matched with an EndUpdate call. If so, it calls the UpdateAll (see page 5995) method to perform any final adjustments to the docked controls, now that they are all in place.

Each call to EndUpdate should be paired by a call to BeginUpdate (see page 5988), which is made before docking or undocking controls. BeginUpdate (see page 5988)/EndUpdate pairs can be nested.

See Also

BeginUpdate (see page 5988)

LoadFromStream (see page 5991)

UpdateAll (see page 5995)

32.1.12.4.5 TDockTree.FindControlAtPos Method

Pascal

```
function FindControlAtPos(const Pos: TPoint): TControl;
```

C++

```
__fastcall TControl * FindControlAtPos(const TPoint Pos);
```

Description

This is FindControlAtPos, a member of class TDockTree.

32.1.12.4.6 TDockTree.GetControlBounds Method

Returns the dimensions allotted to a control that is already docked.

Pascal

```
procedure GetControlBounds(Control: TControl; out CtlBounds: TRect);
```

C++

```
__fastcall GetControlBounds(TControl * Control, TRect CtlBounds);
```

Description

GetControlBounds returns the dimensions of the zone in which a control is currently docked.

Control is the currently docked control.

CtlBounds returns the dimensions of the zone in which control is docked. If Control does not specify an object that is currently docked in the associated dock site, CtlBounds returns (0,0,0,0).

See Also

PositionDockRect (see page 5993)

ResetBounds (see page 5994)

32.1.12.4.7 TDockTree.HitTest Method

Indicates which docked control is at a specified mouse position.

Pascal

```
function HitTest(const MousePos: TPoint; out HTFlag: Integer): TControl; virtual;
```

C++

```
virtual __fastcall TControl * HitTest(const TPoint MousePos, int HTFlag);
```

Description

Applications can't call this protected method. TDockTree (see page 5983) uses HitTest internally to determine how to respond to mouse clicks.

MousePos indicates the position of the mouse pointer.

HTFlag indicates where the mouse pointer is relative to the docked control that is the return value. It is one of the following values:

Value	Meaning
HTNOWHERE	The mouse pointer is not anywhere in the docking region.
HTCLIENT	The mouse pointer is over the client area of the docked control.
HTBORDER	The mouse pointer is over the splitter that allows docking zones to be resized.
HTCAPTION	The mouse pointer is over the grab region where users can start to drag a docked control out of its docked position.
HTCLOSE	The mouse pointer is over the portion of the grab region that contains the "close" button for closing the docked control window.

HitTest returns the docked client that is under the mouse.

See Also

GetControlBounds (see page 5989)

32.1.12.4.8 TDockTree.InsertControl Method

Handles the docking of a control into the associated DockSite (see page 5996).

Pascal

```
procedure InsertControl(Control: TControl; InsertAt: TAlign; DropCtl: TControl); virtual;
```

C++

```
virtual __fastcall InsertControl(TControl * Control, TAlign InsertAt, TControl * DropCtl);
```

Description

InsertControl is called immediately after the windowed control that uses the dock tree adds a new control to its list of docked controls.

Control is the newly docked control.

InsertAt indicates where Control was docked within the windowed control.

DropCtl is the windowed control in which Control was docked. It should be the same as the value of the DockSite (see page 5996) property.

InsertControl sets up the docking zone for the control (including a splitter for resizing the zone and a grab region so that the control can be undocked. It then invalidates the windowed control in which Control was docked so that it can repaint itself to display the new docking region and docked object.

See Also

DockSite (see page 5996)

Invalidate (see page 6161)

RemoveControl (see page 5993)

32.1.12.4.9 TDockTree.LoadFromStream Method

Creates docking zones for a set of docked controls read from a stream.

Pascal

```
procedure LoadFromStream(Stream: TStream); virtual;
```

C++

```
virtual __fastcall LoadFromStream(TStream Stream);
```

Description

LoadFromStream updates the dock tree to reflect the set of docked controls it reads from a stream. The stream must reflect the same format as is written by the SaveToStream (see page 5994) method.

Stream is the stream object to use for reading the docked controls.

See Also

SaveToStream (see page 5994)

32.1.12.4.10 TDockTree.MouseDown Method

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer; var
Handled: Boolean); virtual;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y, Boolean
Handled);
```

Description

This is MouseDown, a member of class TDockTree.

32.1.12.4.11 TDockTree.MouseMove Method

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer; var Handled: Boolean); virtual;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y, Boolean Handled);
```

Description

This is MouseMove, a member of class TDockTree.

32.1.12.4.12 TDockTree.MouseUp Method

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer; var
Handled: Boolean); virtual;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y, Boolean
Handled);
```

Description

This is MouseUp, a member of class TDockTree.

32.1.12.4.13 TDockTree.PaintDockFrame Method

Paints the elements of a docking zone that surround a docked control.

Pascal

```
procedure PaintDockFrame(Canvas: TCanvas; Control: TControl; const ARect: TRect); virtual;
```

C++

```
virtual __fastcall PaintDockFrame(TCanvas Canvas, TControl * Control, const TRect ARect);
```

Description

Application's can't call the protected PaintDockFrame method. It is used by the PaintSite (see page 5992) method to render the grab lines and close buttons that make up the grab region of a docking zone.

Canvas is the drawing surface on which to paint.

Control is the control that is docked in the zone being rendered.

ARect indicates the boundaries of the entire docking zone.

Note: Descendants classes can override this method to change the way a dock tree paints docking zones. The descendant class should only paint in the portions of ARect that are not occupied by Control.

See Also

PaintSite (see page 5992)

32.1.12.4.14 TDockTree.PaintSite Method

Paints the elements that surround all docked controls.

Pascal

```
procedure PaintSite(DC: HDC); virtual;
```

C++

```
virtual __fastcall PaintSite(HDC DC);
```

Description

Windowed controls that use a docking manager call PaintSite when they need to paint their docked child controls.

DC is the handle to the device context to use when rendering those elements of the docking zones other than the docked controls themselves.

PaintSite iterates through the docked controls, and for each one, calls the PaintDockFrame (see page 5992) method to

perform the actual painting.

See Also

PaintDockFrame (see page 5992)

PaintControls (see page 6166)

UseDockManager (see page 6212)

32.1.12.4.15 TDockTree.PositionDockRect Method

Computes the dimensions of a new docking zone.

Pascal

```
procedure PositionDockRect(Client: TControl; DropCtl: TControl; DropAlign: TAlign; var
DockRect: TRect); virtual;
```

C++

```
virtual __fastcall PositionDockRect(TControl * Client, TControl * DropCtl, TAlign DropAlign,
TRect DockRect);
```

Description

The control that is being docked calls PositionDockRect from its own PositionDockRect method when it is being docked to a windowed control that uses a dock manager. It allows the dock tree to indicate where a specified control would appear if the user released the mouse.

Client is the control that is being dragged to a docking position.

DropCtl is the windowed control into which Client will be docked. It should be the same as the value of DockSite (see page 5996).

DropAlign indicates where Control is to be docked within the windowed control.

DockRect returns the boundaries of the docking zone where Control can be docked.

See Also

PositionDockRect (see page 5777)

UseDockManager (see page 6212)

DockSite (see page 5996)

32.1.12.4.16 TDockTree.ReferenceFromOrient Method

Pascal

```
function ReferenceFromOrient(const Orient: TDockOrientation): Integer; virtual;
```

C++

```
virtual __fastcall int ReferenceFromOrient(const TDockOrientation Orient);
```

Description

This is ReferenceFromOrient, a member of class TDockTree.

32.1.12.4.17 TDockTree.RemoveControl Method

Handles the undocking of a control from the associated DockSite (see page 5996).

Pascal

```
procedure RemoveControl(Control: TControl); virtual;
```

C++

```
virtual __fastcall RemoveControl(TControl * Control);
```

Description

The windowed control that uses the dock tree calls RemoveControl when one of its docked controls is undocked.

Control is the control that was just undocked.

RemoveControl adjusts the dock tree's internal structures to indicate that the control is no longer docked, changes the control's DockOrientation property to doNoOrient, and calls DockSite (see page 5996)'s Invalidate method so that the region will be repainted.

See Also

InsertControl (see page 5990)

DockSite (see page 5996)

DockOrientation (see page 5818)

Invalidate (see page 6161)

32.1.12.4.18 TDockTree.ResetBounds Method

Recomputes the layout of docked controls.

Pascal

```
procedure ResetBounds(Force: Boolean); virtual;
```

C++

```
virtual __fastcall ResetBounds(Boolean Force);
```

Description

ResetBounds updates the layout of the docked controls and their surrounding docking zones, based on the current dimensions of DockSite (see page 5996). It is used to update the dock tree to reflect current conditions (for example, when DockSite (see page 5996) is resized).

Force indicates whether ResetBounds should recompute the layout if the dimensions of DockSite (see page 5996) have not changed. When Force is false, ResetBounds only updates the layout of docked controls if the dimensions of DockSite (see page 5996) have changed. When Force is true, ResetBounds updates the layout of docked controls even if the dimensions of DockSite (see page 5996) have not changed.

See Also

DockSite (see page 5996)

UpdateAll (see page 5995)

32.1.12.4.19 TDockTree.SaveToStream Method

Saves a list of the set of docked controls and their positions to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream); virtual;
```

C++

```
virtual __fastcall SaveToStream(TStream Stream);
```

Description

SaveToStream writes enough information to a stream that the dock tree can reconstruct the docking zone structure when it reads that information back at a later time.

Stream is the stream object to use for writing the information.

Note: To reconstruct the docking structure at a later time, use the LoadFromStream (see page 5991) method.

See Also

LoadFromStream (see page 5991)

32.1.12.4.20 TDockTree.SetReplacingControl Method

Prepares the dock tree for replacing one docked control with another.

Pascal

```
procedure SetReplacingControl(Control: TControl);
```

C++

```
__fastcall SetReplacingControl(TControl * Control);
```

Description

SetReplacingControl is called from a control's ReplaceDockedControl method. It informs the dock tree that the next control that is removed will be immediately replaced by a different control. It allows the dock tree to optimize the work of the RemoveControl (see page 5993) and InsertControl (see page 5990) methods.

See Also

ReplaceDockedControl (see page 5779)

InsertControl (see page 5990)

RemoveControl (see page 5993)

32.1.12.4.21 TDockTree.ShowHint Method

Pascal

```
procedure ShowHint(CursorPos: TPoint; var CursorRect: TRect; var HintStr: string); virtual;
```

C++

```
virtual __fastcall ShowHint(TPoint CursorPos, TRect CursorRect, AnsiString HintStr);
```

Description

This is ShowHint, a member of class TDockTree.

32.1.12.4.22 TDockTree.UpdateAll Method

Makes any internal adjustments to reflect the current state of the docked controls.

Pascal

```
procedure UpdateAll;
```

C++

```
__fastcall UpdateAll();
```

Description

Applications can't call the protected UpdateAll method. It is called by the EndUpdate (see page 5988) method when a series of updates to the dock tree have been completed.

UpdateAll adjusts the internal structures of the dock tree to reflect the current layout of all docked controls.

See Also

ResetBounds (see page 5994)

EndUpdate (see page 5988)

32.1.12.4.23 TDockTree.WndProc Method

Pascal

```
procedure WndProc(var Message: TMessage); virtual;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

This is WndProc, a member of class TDockTree.

32.1.12.4.24 TDockTree.ZoneCaptionHitTest Method

Pascal

```
function ZoneCaptionHitTest(const Zone: TDockZone; const MousePos: TPoint; var HTFlag:
Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ZoneCaptionHitTest(const TDockZone Zone, const TPoint MousePos, int
HTFlag);
```

Description

This is ZoneCaptionHitTest, a member of class TDockTree.

32.1.12.5 TDockTree Properties

The properties of the TDockTree class are listed here.

Protected Properties

	Name	Description
	DockSite (see page 5996)	Specifies the windowed control that uses this dock tree to manage docking operations.
	RelativeSizes (see page 5997)	This is RelativeSizes, a member of class TDockTree.
R	TopZone (see page 5997)	This is TopZone, a member of class TDockTree.

32.1.12.5.1 TDockTree.DockSite Property

Specifies the windowed control that uses this dock tree to manage docking operations.

Pascal

```
property DockSite: TWinControl;
```

C++

```
__property TWinControl * DockSite;
```

Description

Applications can't use the protected DockSite property. TDockTree (see page 5983) uses this property internally when it needs to interact with the windowed control for which it manages docking operations.

DockSite is set in the TDockTree (see page 5983) constructor.

See Also

UseDockManager (see page 6212)

Create (see page 5985)

32.1.12.5.2 TDockTree.RelativeSizes Property

Pascal

```
property RelativeSizes: Boolean;
```

C++

```
__property Boolean RelativeSizes;
```

Description

This is RelativeSizes, a member of class TDockTree.

32.1.12.5.3 TDockTree.TopZone Property

Pascal

```
property TopZone: TDockZone;
```

C++

```
__property TDockZone TopZone;
```

Description

This is TopZone, a member of class TDockTree.

32.1.13 TDockZone Class

Class Hierarchy

Pascal

```
TDockZone = class;
```

C++

```
class TDockZone;
```

File

Controls

Description

This is class Controls.TDockZone.

32.1.13.1 TDockZone Members

The following tables list the members exposed by TDockZone.

Public Methods

	Name	Description
	Create (see page 5998)	This is Create, a member of class TDockZone.
	ExpandZoneLimit (see page 5999)	This is ExpandZoneLimit, a member of class TDockZone.
	FirstVisibleChild (see page 5999)	This is FirstVisibleChild, a member of class TDockZone.
	NextVisible (see page 5999)	This is NextVisible, a member of class TDockZone.
	PrevVisible (see page 5999)	This is PrevVisible, a member of class TDockZone.
	ResetChildren (see page 6000)	This is ResetChildren, a member of class TDockZone.
	ResetZoneLimits (see page 6000)	This is ResetZoneLimits, a member of class TDockZone.
	Update (see page 6000)	This is Update, a member of class TDockZone.

Public Properties

	Name	Description
R	ChildControl (see page 6001)	This is ChildControl, a member of class TDockZone.
R	ChildCount (see page 6001)	This is ChildCount, a member of class TDockZone.
R	Height (see page 6001)	This is Height, a member of class TDockZone.
R	Left (see page 6001)	This is Left, a member of class TDockZone.
R	LimitBegin (see page 6002)	This is LimitBegin, a member of class TDockZone.
R	LimitSize (see page 6002)	This is LimitSize, a member of class TDockZone.
R	Top (see page 6002)	This is Top, a member of class TDockZone.
R	Visible (see page 6002)	This is Visible, a member of class TDockZone.
R	VisibleChildCount (see page 6002)	This is VisibleChildCount, a member of class TDockZone.
R	Width (see page 6003)	This is Width, a member of class TDockZone.
	ZoneLimit (see page 6003)	This is ZoneLimit, a member of class TDockZone.

32.1.13.2 TDockZone.Create Constructor

Pascal

```
constructor Create(Tree: TDockTree);
```

C++

```
__fastcall TDockZone(TDockTree Tree);
```

Description

This is Create, a member of class TDockZone.

32.1.13.3 TDockZone Methods

The methods of the TDockZone class are listed here.

Public Methods

	Name	Description
	ExpandZoneLimit (see page 5999)	This is ExpandZoneLimit, a member of class TDockZone.
	FirstVisibleChild (see page 5999)	This is FirstVisibleChild, a member of class TDockZone.
	NextVisible (see page 5999)	This is NextVisible, a member of class TDockZone.
	PrevVisible (see page 5999)	This is PrevVisible, a member of class TDockZone.
	ResetChildren (see page 6000)	This is ResetChildren, a member of class TDockZone.
	ResetZoneLimits (see page 6000)	This is ResetZoneLimits, a member of class TDockZone.
	Update (see page 6000)	This is Update, a member of class TDockZone.

32.1.13.3.1 TDockZone.ExpandZoneLimit Method

Pascal

```
procedure ExpandZoneLimit(NewLimit: Integer);
```

C++

```
__fastcall ExpandZoneLimit(int NewLimit);
```

Description

This is ExpandZoneLimit, a member of class TDockZone.

32.1.13.3.2 TDockZone.FirstVisibleChild Method

Pascal

```
function FirstVisibleChild: TDockZone;
```

C++

```
__fastcall TDockZone FirstVisibleChild();
```

Description

This is FirstVisibleChild, a member of class TDockZone.

32.1.13.3.3 TDockZone.NextVisible Method

Pascal

```
function NextVisible: TDockZone;
```

C++

```
__fastcall TDockZone NextVisible();
```

Description

This is NextVisible, a member of class TDockZone.

32.1.13.3.4 TDockZone.PrevVisible Method

Pascal

```
function PrevVisible: TDockZone;
```

C++

```
__fastcall TDockZone PrevVisible();
```

Description

This is PrevVisible, a member of class TDockZone.

32.1.13.3.5 TDockZone.ResetChildren Method

Pascal

```
procedure ResetChildren;
```

C++

```
__fastcall ResetChildren();
```

Description

This is ResetChildren, a member of class TDockZone.

32.1.13.3.6 TDockZone.ResetZoneLimits Method

Pascal

```
procedure ResetZoneLimits;
```

C++

```
__fastcall ResetZoneLimits();
```

Description

This is ResetZoneLimits, a member of class TDockZone.

32.1.13.3.7 TDockZone.Update Method

Pascal

```
procedure Update;
```

C++

```
__fastcall Update();
```

Description

This is Update, a member of class TDockZone.

32.1.13.4 TDockZone Properties

The properties of the TDockZone class are listed here.

Public Properties

	Name	Description
R	ChildControl (see page 6001)	This is ChildControl, a member of class TDockZone.
R	ChildCount (see page 6001)	This is ChildCount, a member of class TDockZone.
R	Height (see page 6001)	This is Height, a member of class TDockZone.
R	Left (see page 6001)	This is Left, a member of class TDockZone.
R	LimitBegin (see page 6002)	This is LimitBegin, a member of class TDockZone.
R	LimitSize (see page 6002)	This is LimitSize, a member of class TDockZone.
R	Top (see page 6002)	This is Top, a member of class TDockZone.
R	Visible (see page 6002)	This is Visible, a member of class TDockZone.

R	VisibleChildCount (see page 6002)	This is VisibleChildCount, a member of class TDockZone.
R	Width (see page 6003)	This is Width, a member of class TDockZone.
	ZoneLimit (see page 6003)	This is ZoneLimit, a member of class TDockZone.

32.1.13.4.1 TDockZone.ChildControl Property

Pascal

```
property ChildControl: TControl;
```

C++

```
__property TControl * ChildControl;
```

Description

This is ChildControl, a member of class TDockZone.

32.1.13.4.2 TDockZone.ChildCount Property

Pascal

```
property ChildCount: Integer;
```

C++

```
__property int ChildCount;
```

Description

This is ChildCount, a member of class TDockZone.

32.1.13.4.3 TDockZone.Height Property

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

This is Height, a member of class TDockZone.

32.1.13.4.4 TDockZone.Left Property

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

This is Left, a member of class TDockZone.

32.1.13.4.5 TDockZone.LimitBegin Property

Pascal

```
property LimitBegin: Integer;
```

C++

```
__property int LimitBegin;
```

Description

This is LimitBegin, a member of class TDockZone.

32.1.13.4.6 TDockZone.LimitSize Property

Pascal

```
property LimitSize: Integer;
```

C++

```
__property int LimitSize;
```

Description

This is LimitSize, a member of class TDockZone.

32.1.13.4.7 TDockZone.Top Property

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

This is Top, a member of class TDockZone.

32.1.13.4.8 TDockZone.Visible Property

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

This is Visible, a member of class TDockZone.

32.1.13.4.9 TDockZone.VisibleChildCount Property

Pascal

```
property VisibleChildCount: Integer;
```

C++

```
__property int VisibleChildCount;
```

Description

This is VisibleChildCount, a member of class TDockZone.

32.1.13.4.10 TDockZone.Width Property

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

This is Width, a member of class TDockZone.

32.1.13.4.11 TDockZone.ZoneLimit Property

Pascal

```
property ZoneLimit: Integer;
```

C++

```
__property int ZoneLimit;
```

Description

This is ZoneLimit, a member of class TDockZone.

32.1.14 TDragControlObject Class

TDragControlObject object provides for normal drag and drop operations.

Class Hierarchy

Pascal

```
TDragControlObject = class(TBaseDragControlObject);
```

C++

```
class TDragControlObject : public TBaseDragControlObject;
```

File

Controls

Description

TDragControlObject object performs normal drag and drop operations. When TDragControlObject is used, the OnDragOver and OnDragDrop events receive the control being dragged as the Sender, instead of the drag object itself.

Note: TDragControlObject is not automatically freed at the end of a drag operation. For a drag object that is automatically freed, use TDragControlObjectEx (see page 6007) instead.

See Also

TDragDockObject (see page 6009)

TControl (see page 5724)

TControl (see page 5724)

TDragControlObjectEx (see page 6007)

32.1.14.1 TDragControlObject Members

The following tables list the members exposed by TDragControlObject.

Protected Methods

	Name	Description
	EndDrag (see page 5723)	Generates an OnEndDrag event for the control that is being dragged.
	Finished (see page 5723)	Finished ends a drag and drop operation.
	Finished (see page 6035)	Responds when the mouse is released indicating the drag operation is complete.
	GetDragCursor (see page 6005)	Returns the correct drag cursor to display at the X, Y position for the drag and drop operation.
	GetDragImages (see page 6006)	Returns the control's image list containing the image to be displayed while dragging.
	MainWndProc (see page 6037)	This is MainWndProc, a member of class TDragObject.
	WndProc (see page 6038)	This is WndProc, a member of class TDragObject.

Public Methods

	Name	Description
	AfterConstruction (see page 6033)	Responds after the last constructor has executed.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 5722)	Copies the properties of another drag control object.
	Assign (see page 6034)	Copies the properties of another drag control object.
	BeforeDestruction (see page 6034)	Responds before the first destructor executes.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5721)	Creates an instance of the TBaseDragControlObject (see page 5719) and stores a reference to the control that is being dragged.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetName (see page 6036)	Returns a string that identifies the drag operation.
	HideDragImage (see page 6006)	Hides the image used in the drag cursor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Instance (see page 6037)	Returns the instance (module) handle of the EXE or DLL that created the drag object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	ShowDragImage (see page 6006)	Shows the image used in the drag cursor.

Public Properties

	Name	Description
	AlwaysShowDragImages (see page 6038)	This is AlwaysShowDragImages, a member of class TDragObject.
	Cancelling (see page 6038)	Specifies whether the drag operation is being canceled.
	Control (see page 5724)	Indicates the control that is being dragged.
	DragHandle (see page 6039)	Specifies the window handle of the dragged object.
	DragPos (see page 6039)	Specifies the position of the dragged object.
	DragTarget (see page 6039)	Specifies the dragged object's target.
	DragTargetPos (see page 6040)	Specifies the position of the drag object relative to the drag target.
R	Dropped (see page 6040)	Drag ended with drop.
R	MouseDeltaX (see page 6040)	Specifies the x-coordinate of the mouse position relative to the position of the dragged object's target.
R	MouseDeltaY (see page 6041)	Specifies the y-coordinate of the mouse position relative to the position of the dragged object's target.
	RightClickCancels (see page 6041)	This is RightClickCancels, a member of class TDragObject.

32.1.14.2 TDragControlObject Methods

The methods of the TDragControlObject class are listed here.

Protected Methods

	Name	Description
V	GetDragCursor (see page 6005)	Returns the correct drag cursor to display at the X, Y position for the drag and drop operation.
V	GetDragImages (see page 6006)	Returns the control's image list containing the image to be displayed while dragging.

Public Methods

	Name	Description
V	HideDragImage (see page 6006)	Hides the image used in the drag cursor.
V	ShowDragImage (see page 6006)	Shows the image used in the drag cursor.

32.1.14.2.1 TDragControlObject.GetDragCursor Method

Returns the correct drag cursor to display at the X, Y position for the drag and drop operation.

Pascal

```
function GetDragCursor(Accepted: Boolean; X: Integer; Y: Integer): TCursor; override;
```

C++

```
virtual __fastcall TCursor GetDragCursor(Boolean Accepted, int X, int Y);
```

Description

If the control at the X, Y position accepts the drag (in an OnDragOver event handler), the drag object sets the drag cursor to the

control's DragCursor property; otherwise, it displays the crNoDrop (see page 6366) cursor.

See Also

DragCursor (see page 5818)

OnDragOver (see page 5857)

32.1.14.2.2 TDragControlObject.GetDragImages Method

Returns the control's image list containing the image to be displayed while dragging.

Pascal

```
function GetDragImages: TDragImageList; override;
```

C++

```
virtual __fastcall TDragImageList GetDragImages();
```

Description

GetDragImages calls the associated control's GetDragImages method. The image list normally contains only one image, which the control creates and maintains. For many controls in the VCL, GetDragImages returns nil (Delphi) or NULL (C++).

See Also

Control (see page 5724)

GetDragImages (see page 5763)

32.1.14.2.3 TDragControlObject.HideDragImage Method

Hides the image used in the drag cursor.

Pascal

```
procedure HideDragImage; override;
```

C++

```
virtual __fastcall HideDragImage();
```

Description

If the control being dragged contains an image list containing an image for dragging, it will be hidden. This allows control over when a drag image is displayed.

See Also

ShowDragImage (see page 6006)

GetDragImages (see page 6006)

32.1.14.2.4 TDragControlObject.ShowDragImage Method

Shows the image used in the drag cursor.

Pascal

```
procedure ShowDragImage; override;
```

C++

```
virtual __fastcall ShowDragImage();
```

Description

If the control being dragged contains an image list containing an image for dragging, it will be shown. This allows control over when a drag image is shown.

See Also

HideDragImage (see page 6006)

GetDragImages (see page 6006)

32.1.15 TDragControlObjectEx Class

TDragControlObjectEx object provides for normal drag and drop operations.

Class Hierarchy

Pascal

```
TDragControlObjectEx = class(TDragControlObject);
```

C++

```
class TDragControlObjectEx : public TDragControlObject;
```

File

Controls

Description

TDragControlObjectEx object performs normal drag and drop operations. When TDragControlObjectEx is used, the OnDragOver and OnDragDrop events receive the control being dragged as the Sender, instead of the drag object itself.

Note: Unlike its immediate ancestor (TDragControlObject (see page 6003)), TDragControlObjectEx is automatically freed at the end of a drag operation.

See Also

TDragDockObject (see page 6009)

TControl (see page 5724)

TControl (see page 5724)

32.1.15.1 TDragControlObjectEx Members

The following tables list the members exposed by TDragControlObjectEx.

Protected Methods

	Name	Description
	EndDrag (see page 5723)	Generates an OnEndDrag event for the control that is being dragged.
	Finished (see page 5723)	Finished ends a drag and drop operation.
	Finished (see page 6035)	Responds when the mouse is released indicating the drag operation is complete.
	GetDragCursor (see page 6005)	Returns the correct drag cursor to display at the X, Y position for the drag and drop operation.
	GetDragCursor (see page 6035)	Indicates which cursor should be displayed in response to mouse moves.

	GetDragImages (see page 6006)	Returns the control's image list containing the image to be displayed while dragging.
	GetDragImages (see page 6035)	Returns an image list containing the image to be displayed while dragging.
	MainWndProc (see page 6037)	This is MainWndProc, a member of class TDragObject.
	WndProc (see page 6038)	This is WndProc, a member of class TDragObject.

Public Methods

	Name	Description
	AfterConstruction (see page 6033)	Responds after the last constructor has executed.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 5722)	Copies the properties of another drag control object.
	Assign (see page 6034)	Copies the properties of another drag control object.
	BeforeDestruction (see page 6009)	Allows the application to detect that the drag object needs to be freed.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5721)	Creates an instance of the TBaseDragControlObject (see page 5719) and stores a reference to the control that is being dragged.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetName (see page 6036)	Returns a string that identifies the drag operation.
	HideDragImage (see page 6006)	Hides the image used in the drag cursor.
	HideDragImage (see page 6036)	Provides the interface for a method that hides the current drag image.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Instance (see page 6037)	Returns the instance (module) handle of the EXE or DLL that created the drag object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ShowDragImage (see page 6006)	Shows the image used in the drag cursor.
	ShowDragImage (see page 6037)	Provides the interface for a method that hides the current drag image.

Public Properties

	Name	Description
	AlwaysShowDragImages (see page 6038)	This is AlwaysShowDragImages, a member of class TDragObject.
	Cancelling (see page 6038)	Specifies whether the drag operation is being canceled.

	Control (see page 5724)	Indicates the control that is being dragged.
	DragHandle (see page 6039)	Specifies the window handle of the dragged object.
	DragPos (see page 6039)	Specifies the position of the dragged object.
	DragTarget (see page 6039)	Specifies the dragged object's target.
	DragTargetPos (see page 6040)	Specifies the position of the drag object relative to the drag target.
R	Dropped (see page 6040)	Drag ended with drop.
R	MouseDeltaX (see page 6040)	Specifies the x-coordinate of the mouse position relative to the position of the dragged object's target.
R	MouseDeltaY (see page 6041)	Specifies the y-coordinate of the mouse position relative to the position of the dragged object's target.
	RightClickCancels (see page 6041)	This is RightClickCancels, a member of class TDragObject.

32.1.15.2 TDragControlObjectEx Methods

The methods of the TDragControlObjectEx class are listed here.

Public Methods

	Name	Description
V	BeforeDestruction (see page 6009)	Allows the application to detect that the drag object needs to be freed.

32.1.15.2.1 TDragControlObjectEx.BeforeDestruction Method

Allows the application to detect that the drag object needs to be freed.

Pascal

```
procedure BeforeDestruction; override;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

Do not call BeforeDestruction. This method is called automatically before any TDragControlObjectEx (see page 6007) destructors are called. It suppresses the inherited method, which prevents the drag object from being freed at the end of a drag operation.

32.1.16 TDragDockObject Class

Contains drag-and-dock information for a control.

Class Hierarchy

Pascal

```
TDragDockObject = class(TBaseDragControlObject);
```

C++

```
class TDragDockObject : public TBaseDragControlObject;
```

File

Controls

Description

TDragDockObject is used in several events and methods that use drag-and-dock information, including the OnDockDrop and OnDockOver events of TWinControl (see page 6112).

Note: TDragDockObject is not automatically freed at the end of a drag-and-dock operation. To work with a drag object that is freed at the end of the drag-and-dock operation, use TDragDockObjectEx (see page 6018) instead.

See Also

TWinControl (see page 6112)

TWinControl (see page 6112)

TDragDockObjectEx (see page 6018)

32.1.16.1 TDragDockObject Members

The following tables list the members exposed by TDragDockObject.

Protected Methods

	Name	Description
	AdjustDockRect (see page 6012)	This is AdjustDockRect, a member of class TDragDockObject.
	DrawDragDockImage (see page 6013)	Called when the dock image for a control is drawn.
	EndDrag (see page 6013)	Called when a control is released from dragging.
	EraseDragDockImage (see page 6013)	Called when the dock image for a control is erased.
	Finished (see page 5723)	Finished ends a drag and drop operation.
	Finished (see page 6035)	Responds when the mouse is released indicating the drag operation is complete.
	GetDragCursor (see page 6014)	Returns the cursor used during dragging.
	GetDragImages (see page 6035)	Returns an image list containing the image to be displayed while dragging.
	GetEraseWhenMoving (see page 6014)	This is GetEraseWhenMoving, a member of class TDragDockObject.
	GetFrameWidth (see page 6014)	Returns the value of the FrameWidth (see page 6018) property.
	MainWndProc (see page 6037)	This is MainWndProc, a member of class TDragObject.
	WndProc (see page 6038)	This is WndProc, a member of class TDragObject.

Public Methods

	Name	Description
	AfterConstruction (see page 6033)	Responds after the last constructor has executed.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 6012)	Enables assignments of TDragDockObject (see page 6009) objects.
	BeforeDestruction (see page 6034)	Responds before the first destructor executes.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6011)	Creates and initializes an instance of TDragDockObject (see page 6009).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 6012)	Destroys an instance of TDragDockObject (see page 6009).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetName (see page 6036)	Returns a string that identifies the drag operation.
V	HideDragImage (see page 6036)	Provides the interface for a method that hides the current drag image.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Instance (see page 6037)	Returns the instance (module) handle of the EXE or DLL that created the drag object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	ShowDragImage (see page 6037)	Provides the interface for a method that hides the current drag image.

Public Properties

	Name	Description
	AlwaysShowDragImages (see page 6038)	This is AlwaysShowDragImages, a member of class TDragObject.
	Brush (see page 6015)	Specifies the brush used to draw the dock image.
	Cancelling (see page 6038)	Specifies whether the drag operation is being canceled.
	Control (see page 5724)	Indicates the control that is being dragged.
	DockRect (see page 6015)	Specifies the rectangle on which the control is docked if released.
	DragHandle (see page 6039)	Specifies the window handle of the dragged object.
	DragPos (see page 6039)	Specifies the position of the dragged object.
	DragTarget (see page 6039)	Specifies the dragged object's target.
	DragTargetPos (see page 6040)	Specifies the position of the drag object relative to the drag target.
R	DropAlign (see page 6017)	Specifies how the control will be aligned when docked.
R	DropOnControl (see page 6017)	Specifies the control on which the dragged control will be docked if released.
R	Dropped (see page 6040)	Drag ended with drop.
	EraseDockRect (see page 6017)	This is EraseDockRect, a member of class TDragDockObject.
R	EraseWhenMoving (see page 6018)	This is EraseWhenMoving, a member of class TDragDockObject.
	Floating (see page 6018)	Specifies whether the control is floating.
R	FrameWidth (see page 6018)	Specifies the border width, in pixels, of the frame drawn around the drag image.
R	MouseDeltaX (see page 6040)	Specifies the x-coordinate of the mouse position relative to the position of the dragged object's target.
R	MouseDeltaY (see page 6041)	Specifies the y-coordinate of the mouse position relative to the position of the dragged object's target.
	RightClickCancels (see page 6041)	This is RightClickCancels, a member of class TDragObject.

32.1.16.2 TDragDockObject.Create Constructor

Creates and initializes an instance of TDragDockObject (see page 6009).

Pascal

```
constructor Create(AControl: TControl); override;
```

C++

```
virtual __fastcall TDragDockObject(TControl * AControl);
```

Description

Create calls the inherited constructor and allocates memory for the Brush (see page 6015) property.

32.1.16.3 TDragDockObject.Destroy Destructor

Destroys an instance of TDragDockObject (see page 6009).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDragDockObject();
```

Description

Do not call Destroy directly. Instead, call Free.

Destroy frees the memory associated with the Brush (see page 6015) property and then calls the inherited destructor.

32.1.16.4 TDragDockObject Methods

The methods of the TDragDockObject class are listed here.

Protected Methods

	Name	Description
	AdjustDockRect (see page 6012)	This is AdjustDockRect, a member of class TDragDockObject.
	DrawDragDockImage (see page 6013)	Called when the dock image for a control is drawn.
	EndDrag (see page 6013)	Called when a control is released from dragging.
	EraseDragDockImage (see page 6013)	Called when the dock image for a control is erased.
	GetDragCursor (see page 6014)	Returns the cursor used during dragging.
	GetEraseWhenMoving (see page 6014)	This is GetEraseWhenMoving, a member of class TDragDockObject.
	GetFrameWidth (see page 6014)	Returns the value of the FrameWidth (see page 6018) property.

Public Methods

	Name	Description
	Assign (see page 6012)	Enables assignments of TDragDockObject (see page 6009) objects.

32.1.16.4.1 TDragDockObject.AdjustDockRect Method

Pascal

```
procedure AdjustDockRect(ARect: TRect); virtual;
```

C++

```
virtual __fastcall AdjustDockRect(TRect ARect);
```

Description

This is AdjustDockRect, a member of class TDragDockObject.

32.1.16.4.2 TDragDockObject.Assign Method

Enables assignments of TDragDockObject (see page 6009) objects.

Pascal

```
procedure Assign(Source: TDragObject); override;
```

C++

```
virtual __fastcall Assign(TDragObject Source);
```

Description

Assign calls the inherited Assign method and then handles the assignment of TDragDockObject (see page 6009) objects.

32.1.16.4.3 TDragDockObject.DrawDragDockImage Method

Called when the dock image for a control is drawn.

Pascal

```
procedure DrawDragDockImage; virtual;
```

C++

```
virtual __fastcall DrawDragDockImage();
```

Description

Override DrawDragDockImage in a derived class to change the way the dock image is drawn.

32.1.16.4.4 TDragDockObject.EndDrag Method

Called when a control is released from dragging.

Pascal

```
procedure EndDrag(Target: TObject; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall EndDrag(TObject * Target, int X, int Y);
```

Description

Override EndDrag in a derived class to perform actions when a control is released from dragging.

As implemented in TDragDockObject (see page 6009), EndDrag calls the dragged control's DoEndDock method.

Target is the object in which the control is docked. If the control is not docked, Target is nil (Delphi) or NULL (C++).

X and Y are the coordinates of the control.

See Also

DoEndDock (see page 5750)

32.1.16.4.5 TDragDockObject.EraseDragDockImage Method

Called when the dock image for a control is erased.

Pascal

```
procedure EraseDragDockImage; virtual;
```

C++

```
virtual __fastcall EraseDragDockImage();
```

Description

Override EraseDragDockImage in a derived class to perform actions when the dock image for a control is erased.

As implemented in TDragDockObject (see page 6009), EndDrag (see page 6013) calls the dragged control's EraseDragDockImage method.

32.1.16.4.6 TDragDockObject.GetDragCursor Method

Returns the cursor used during dragging.

Pascal

```
function GetDragCursor(Accepted: Boolean; X: Integer; Y: Integer): TCursor; override;
```

C++

```
virtual __fastcall TCursor GetDragCursor(Boolean Accepted, int X, int Y);
```

Description

Override GetDragCursor to specify the cursor used during dragging.

By default, GetDragCursor returns crDefault (see page 6362).

See Also

Cursor (see page 5815)

32.1.16.4.7 TDragDockObject.GetEraseWhenMoving Method

Pascal

```
function GetEraseWhenMoving: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetEraseWhenMoving();
```

Description

This is GetEraseWhenMoving, a member of class TDragDockObject.

32.1.16.4.8 TDragDockObject.GetFrameWidth Method

Returns the value of the FrameWidth (see page 6018) property.

Pascal

```
function GetFrameWidth: Integer; virtual;
```

C++

```
virtual __fastcall int GetFrameWidth();
```

Description

Override GetFrameWidth to change the width of the frame that is drawn around the drag image. As implemented in TDragDockObject (see page 6009), GetFrameWidth returns 4.

See Also

FrameWidth (see page 6018)

32.1.16.5 TDragDockObject Properties

The properties of the TDragDockObject class are listed here.

Public Properties

	Name	Description
	Brush (see page 6015)	Specifies the brush used to draw the dock image.
	DockRect (see page 6015)	Specifies the rectangle on which the control is docked if released.
R	DropAlign (see page 6017)	Specifies how the control will be aligned when docked.
R	DropOnControl (see page 6017)	Specifies the control on which the dragged control will be docked if released.
	EraseDockRect (see page 6017)	This is EraseDockRect, a member of class TDragDockObject.
R	EraseWhenMoving (see page 6018)	This is EraseWhenMoving, a member of class TDragDockObject.
	Floating (see page 6018)	Specifies whether the control is floating.
R	FrameWidth (see page 6018)	Specifies the border width, in pixels, of the frame drawn around the drag image.

32.1.16.5.1 TDragDockObject.Brush Property

Specifies the brush used to draw the dock image.

Pascal

```
property Brush: TBrush;
```

C++

```
__property TBrush Brush;
```

Description

Use Brush to specify the brush used to draw the dock image.

32.1.16.5.2 TDragDockObject.DockRect Property

Specifies the rectangle on which the control is docked if released.

Pascal

```
property DockRect: TRect;
```

C++

```
__property TRect DockRect;
```

Description

Use DockRect to determine the rectangle on which the control is docked if released.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
```

```
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
```

```
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

32.1.16.5.3 TDragDockObject.DropAlign Property

Specifies how the control will be aligned when docked.

Pascal

```
property DropAlign: TAlign;
```

C++

```
__property TAlign DropAlign;
```

Description

Use DropAlign to determine how the control will be aligned when docked.

32.1.16.5.4 TDragDockObject.DropOnControl Property

Specifies the control on which the dragged control will be docked if released.

Pascal

```
property DropOnControl: TControl;
```

C++

```
__property TControl * DropOnControl;
```

Description

Use DropOnControl to determine the control on which the dragged control would be dropped if released.

32.1.16.5.5 TDragDockObject.EraseDockRect Property

Pascal

```
property EraseDockRect: TRect;
```

C++

```
__property TRect EraseDockRect;
```

Description

This is EraseDockRect, a member of class TDragDockObject.

32.1.16.5.6 TDragDockObject.EraseWhenMoving Property

Pascal

```
property EraseWhenMoving: Boolean;
```

C++

```
__property Boolean EraseWhenMoving;
```

Description

This is EraseWhenMoving, a member of class TDragDockObject.

32.1.16.5.7 TDragDockObject.Floating Property

Specifies whether the control is floating.

Pascal

```
property Floating: Boolean;
```

C++

```
__property Boolean Floating;
```

Description

Use Floating to determine whether the control is floating.

32.1.16.5.8 TDragDockObject.FrameWidth Property

Specifies the border width, in pixels, of the frame drawn around the drag image.

Pascal

```
property FrameWidth: Integer;
```

C++

```
__property int FrameWidth;
```

Description

Read FrameWidth to determine the size of the frame drawn around the drag image. As implemented in TDragDockObject (see page 6009), this value is always 4 pixels.

See Also

DrawDragDockImage (see page 6013)

32.1.17 TDragDockObjectEx Class

Contains drag-and-dock information for a control.

Class Hierarchy

System::TObject → Controls.TDragObject → Controls.TBaseDragControlObject → Controls.TDragDockObject → Controls.TDragDockObjectEx

Pascal

```
TDragDockObjectEx = class(TDragDockObject);
```

C++

```
class TDragDockObjectEx : public TDragDockObject;
```

File

Controls

Description

TDragDockObjectEx is used in several events and methods that use drag-and-dock information, including the OnDockDrop and OnDockOver events of TWinControl (see page 6112). Unlike its immediate ancestor, TDragDockObject (see page 6009), TDragDockObjectEx is automatically freed at the end of a drag-and-dock operation.

See Also

TWinControl (see page 6112)

TWinControl (see page 6112)

TControl (see page 5724)

32.1.17.1 TDragDockObjectEx Members

The following tables list the members exposed by TDragDockObjectEx.

Protected Methods

	Name	Description
	AdjustDockRect (see page 6012)	This is AdjustDockRect, a member of class TDragDockObject.
	DrawDragDockImage (see page 6013)	Called when the dock image for a control is drawn.
	EndDrag (see page 5723)	Generates an OnEndDrag event for the control that is being dragged.
	EndDrag (see page 6013)	Called when a control is released from dragging.
	EraseDragDockImage (see page 6013)	Called when the dock image for a control is erased.
	Finished (see page 5723)	Finished ends a drag and drop operation.
	Finished (see page 6035)	Responds when the mouse is released indicating the drag operation is complete.
	GetDragCursor (see page 6014)	Returns the cursor used during dragging.
	GetDragCursor (see page 6035)	Indicates which cursor should be displayed in response to mouse moves.
	GetDragImages (see page 6035)	Returns an image list containing the image to be displayed while dragging.
	GetEraseWhenMoving (see page 6014)	This is GetEraseWhenMoving, a member of class TDragDockObject.
	GetFrameWidth (see page 6014)	Returns the value of the FrameWidth (see page 6018) property.
	MainWndProc (see page 6037)	This is MainWndProc, a member of class TDragObject.
	WndProc (see page 6038)	This is WndProc, a member of class TDragObject.

Public Methods

	Name	Description
	AfterConstruction (see page 6033)	Responds after the last constructor has executed.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 5722)	Copies the properties of another drag control object.
	Assign (see page 6012)	Enables assignments of TDragDockObject (see page 6009) objects.
	Assign (see page 6034)	Copies the properties of another drag control object.
	BeforeDestruction (see page 6021)	Allows the application to detect that the drag object needs to be freed.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 5721)	Creates an instance of the TBaseDragControlObject (see page 5719) and stores a reference to the control that is being dragged.
V	Create (see page 6011)	Creates and initializes an instance of TDragDockObject (see page 6009).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 6012)	Destroys an instance of TDragDockObject (see page 6009).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetName (see page 6036)	Returns a string that identifies the drag operation.
V	HideDragImage (see page 6036)	Provides the interface for a method that hides the current drag image.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Instance (see page 6037)	Returns the instance (module) handle of the EXE or DLL that created the drag object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	ShowDragImage (see page 6037)	Provides the interface for a method that hides the current drag image.

Public Properties

	Name	Description
	AlwaysShowDragImages (see page 6038)	This is AlwaysShowDragImages, a member of class TDragObject.
	Brush (see page 6015)	Specifies the brush used to draw the dock image.
	Cancelling (see page 6038)	Specifies whether the drag operation is being canceled.
	Control (see page 5724)	Indicates the control that is being dragged.
	DockRect (see page 6015)	Specifies the rectangle on which the control is docked if released.
	DragHandle (see page 6039)	Specifies the window handle of the dragged object.
	DragPos (see page 6039)	Specifies the position of the dragged object.
	DragTarget (see page 6039)	Specifies the dragged object's target.
	DragTargetPos (see page 6040)	Specifies the position of the drag object relative to the drag target.
R	DropAlign (see page 6017)	Specifies how the control will be aligned when docked.
R	DropOnControl (see page 6017)	Specifies the control on which the dragged control will be docked if released.
R	Dropped (see page 6040)	Drag ended with drop.
	EraseDockRect (see page 6017)	This is EraseDockRect, a member of class TDragDockObject.
R	EraseWhenMoving (see page 6018)	This is EraseWhenMoving, a member of class TDragDockObject.
	Floating (see page 6018)	Specifies whether the control is floating.

R	FrameWidth (see page 6018)	Specifies the border width, in pixels, of the frame drawn around the drag image.
R	MouseDeltaX (see page 6040)	Specifies the x-coordinate of the mouse position relative to the position of the dragged object's target.
R	MouseDeltaY (see page 6041)	Specifies the y-coordinate of the mouse position relative to the position of the dragged object's target.
	RightClickCancels (see page 6041)	This is RightClickCancels, a member of class TDragObject.

32.1.17.2 TDragDockObjectEx Methods

The methods of the TDragDockObjectEx class are listed here.

Public Methods

	Name	Description
V	BeforeDestruction (see page 6021)	Allows the application to detect that the drag object needs to be freed.

32.1.17.2.1 TDragDockObjectEx.BeforeDestruction Method

Allows the application to detect that the drag object needs to be freed.

Pascal

```
procedure BeforeDestruction; override;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

Do not call BeforeDestruction. This method is called automatically before any TDragDockObjectEx (see page 6018) destructors are called. It suppresses the inherited method, which prevents the drag object from being freed at the end of a drag operation.

32.1.18 TDragImageList Class

TDragImageList represents a set of images that appear during drag operations.

Class Hierarchy

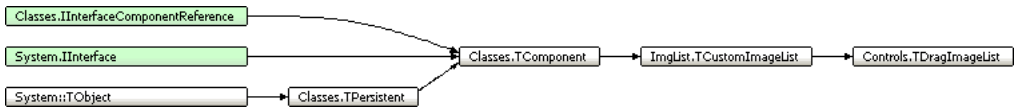

Pascal

```
TDragImageList = class(TCustomImageList);
```

C++

```
class TDragImageList : public TCustomImageList;
```

File

Controls

Description

TDragImageList is used to hold images that are displayed during drag and drop or drag and dock operations. It is created by the

control that is dragged, which returns the drag image list in its GetDragImages method. During a drag operation, a TDragControlObject (see page 6003) instance uses the drag image list to display or hide the images in the list.

See Also

GetDragImages

32.1.18.1 TDragImageList Members

The following tables list the members exposed by TDragImageList.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 15370)	Copies the image list's properties to another image list.
	Change (see page 15371)	Generates an OnChange (see page 15393) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 15371)	Allows bitmap data to be saved and loaded.
V	DoDraw (see page 15372)	Draws the specified image onto the provided Canvas in the indicated style.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetImages (see page 15378)	Returns a specified image and its mask as bitmaps.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleNeeded (see page 15381)	This is HandleNeeded, a member of class TCustomImageList.
V	Initialize (see page 6028)	Sets numerous image list properties to their default values and creates instances of objects used internal to the image list.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadData (see page 15383)	This is ReadData, a member of class TCustomImageList.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteData (see page 15386)	This is WriteData, a member of class TCustomImageList.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnChange (see page 15393)	Occurs when the image list changes.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Add (see page 15368)	Adds an Image and its Mask (see page 15797) to the image list object.
	AddIcon (see page 15368)	Adds an icon to the image list object.
	AddImage (see page 15368)	Adds single image from another image list.
	AddImages (see page 15369)	Adds all images from another image list.
	AddMasked (see page 15369)	Adds an image to the image list along with a color which is used to generate the transparent mask.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	Assign (see page 15370)	Copies the information from one image list to another.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 6026)	Starts the dragging of an image list.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 15371)	Clears all images from the image list.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 15365)	Creates an empty image list.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateSize (see page 15366)	Creates an empty image list of a specified size.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 15372)	Deletes a particular image from the image list.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 15366)	Destroys the image list and releases the memory allocated to it.
V	Destroy (see page 22287)	Disposes of an object instance.

	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DragLock (see page 6026)	Associates the drag image with the given Window handle and draws it at the position given by the parameters XPos and YPos while the window is in drag mode.
	DragMove (see page 6027)	Moves the drag image set by SetDragImage (see page 6029) to a new position specified by X and Y within its currently associated window while the window is in drag mode.
	DragUnlock (see page 6027)	Removes the association between the drag image and a window.
	Draw (see page 15373)	This is the overview for the Draw method overload.
	DrawOverlay (see page 15374)	This is the overview for the DrawOverlay method overload.
	EndDrag (see page 6028)	Ends the drag operation.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FileLoad (see page 15375)	Loads a resourced image into the image list.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetBitmap (see page 15376)	Retrieves a specified image as a bitmap.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetHotSpot (see page 6028)	Returns the hot spot location associated with the current drag image.
	GetIcon (see page 15377)	This is the overview for the GetIcon method overload.
	GetImageBitmap (see page 15378)	Returns a handle to a bitmap containing all of the images in the image list.
	GetInstRes (see page 15378)	This is the overview for the GetInstRes method overload.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaskBitmap (see page 15379)	Returns a handle to a bitmap containing all masks for images in the image list.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetResource (see page 15380)	Loads the specified bitmap, cursor, or icon resource into the image list.
	HandleAllocated (see page 15380)	Indicates whether a window handle exists for the image list.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HideDragImage (see page 6028)	Hides the current drag image if it was previously shown.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 15381)	Inserts a bitmap and its mask into the image list following a specified image.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertIcon (see page 15382)	Inserts an icon at a specified position in the image list.
	InsertMasked (see page 15382)	Inserts a bitmap at a specified position, creating the mask from a specified transparent color.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	Move (see page 15382)	Moves a specified image to another position.

V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Overlay (see page 15383)	Identifies an image in the list as an image that can be used as an overlay mask.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 15383)	Registers an object to be notified when changes occur in the image list.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Replace (see page 15384)	Replaces an image in the image list with a new image and mask.
	ReplaceIcon (see page 15384)	Replaces an image in the image list with an icon.
	ReplaceMasked (see page 15384)	Replaces an image in the image list with a new image and automatically-generated mask.
	ResInstLoad (see page 15385)	Loads a resource into the image list.
	ResourceLoad (see page 15385)	Loads a resource into the image list.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetDragImage (see page 6029)	Sets an image within the image list to be shown during a drag and drop operation.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	ShowDragImage (see page 6029)	Shows the current drag image if it was previously hidden.
	UnRegisterChanges (see page 15386)	Removes an object from the list of objects to be notified when the image list changes.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	AllocBy (see page 15387)	Sets the number of images by which the image list grows when it needs to make room for new images.
	BkColor (see page 15387)	Determines which background color to use when drawing an image.
	BlendColor (see page 15387)	Determines which foreground color to use when drawing an image.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Count (see page 15388)	Indicates the number of images in the image list.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DragCursor (see page 6030)	Specifies the current drag cursor that is associated with the image list.
R	Dragging (see page 6030)	Indicates whether an image in the image list is being dragged.
	DragHotspot (see page 6030)	This is DragHotspot, a member of class TDragImageList.
	DrawingStyle (see page 15390)	Specifies the style to be used when the image list is drawing an image.
	Handle (see page 15390)	Designates the handle for the image list.
	Height (see page 15391)	Specifies the height of images within the image list.
	ImageType (see page 15391)	Determines whether the image list uses the associated mask when drawing.
	Masked (see page 15392)	Specifies whether the image list includes masks that can be combined with the images.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ShareImages (see page 15392)	Specifies whether the image list destroys its handle when the image list is destroyed.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Width (see page 15392)	Specifies the width of all images within the image list.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

32.1.18.2 TDragImageList Methods

The methods of the TDragImageList class are listed here.

Protected Methods

	Name	Description
	Initialize (see page 6028)	Sets numerous image list properties to their default values and creates instances of objects used internal to the image list.

Public Methods

	Name	Description
	BeginDrag (see page 6026)	Starts the dragging of an image list.
	DragLock (see page 6026)	Associates the drag image with the given Window handle and draws it at the position given by the parameters XPos and YPos while the window is in drag mode.
	DragMove (see page 6027)	Moves the drag image set by SetDragImage (see page 6029) to a new position specified by X and Y within its currently associated window while the window is in drag mode.
	DragUnlock (see page 6027)	Removes the association between the drag image and a window.
	EndDrag (see page 6028)	Ends the drag operation.
	GetHotSpot (see page 6028)	Returns the hot spot location associated with the current drag image.
	HideDragImage (see page 6028)	Hides the current drag image if it was previously shown.
	SetDragImage (see page 6029)	Sets an image within the image list to be shown during a drag and drop operation.
	ShowDragImage (see page 6029)	Shows the current drag image if it was previously hidden.

32.1.18.2.1 TDragImageList.BeginDrag Method

Starts the dragging of an image list.

Pascal

```
function BeginDrag(Window: HWND; X: Integer; Y: Integer): Boolean;
```

C++

```
__fastcall Boolean BeginDrag(HWND Window, int X, int Y);
```

Description

Use BeginDrag to start dragging the image set by SetDragImage (see page 6029). The image starts at the position specified by X and Y, which are the coordinates of the drag image relative to the window whose handle is specified by the Window parameter. BeginDrag returns true if it was successful in starting the drag of the image.

See Also

EndDrag (see page 6028)

Dragging (see page 6030)

32.1.18.2.2 TDragImageList.DragLock Method

Associates the drag image with the given Window handle and draws it at the position given by the parameters XPos and YPos while the window is in drag mode.

Pascal

```
function DragLock(Window: HWND; XPos: Integer; YPos: Integer): Boolean;
```

C++

```
__fastcall Boolean DragLock(HWND Window, int XPos, int YPos);
```

Description

Call DragLock to use an image in the image list as a drag image when a particular window is dragged. DragLock returns true if successful.

See Also

DragUnlock (see page 6027)

Dragging (see page 6030)

DragMove (see page 6027)

32.1.18.2.3 TDragImageList.DragMove Method

Moves the drag image set by SetDragImage (see page 6029) to a new position specified by X and Y within its currently associated window while the window is in drag mode.

Pascal

```
function DragMove(X: Integer; Y: Integer): Boolean;
```

C++

```
__fastcall Boolean DragMove(int X, int Y);
```

Description

Use the DragMove method to move the image that is being dragged during a drag-and-drop operation. DragMove returns true if successful.

See Also

SetDragImage (see page 6029)

DragLock (see page 6026)

32.1.18.2.4 TDragImageList.DragUnlock Method

Removes the association between the drag image and a window.

Pascal

```
procedure DragUnlock;
```

C++

```
__fastcall DragUnlock();
```

Description

Call DragUnlock when an image is no longer needed as a drag image for a particular window. DragUnlock removes the association started by the DragLock (see page 6026) method.

See Also

DragLock (see page 6026)

32.1.18.2.5 TDragImageList.EndDrag Method

Ends the drag operation.

Pascal

```
function EndDrag: Boolean;
```

C++

```
__fastcall Boolean EndDrag();
```

Description

Use EndDrag after BeginDrag (see page 6026) has been called to end the drag operation. EndDrag returns true if successful.

See Also

BeginDrag (see page 6026)

32.1.18.2.6 TDragImageList.GetHotSpot Method

Returns the hot spot location associated with the current drag image.

Pascal

```
function GetHotSpot: TPoint; override;
```

C++

```
virtual __fastcall TPoint GetHotSpot();
```

Description

Use the GetHotSpot method to obtain the position of the hot spot of the current drag image. The hot spot represents the current mouse position.

32.1.18.2.7 TDragImageList.HideDragImage Method

Hides the current drag image if it was previously shown.

Pascal

```
procedure HideDragImage;
```

C++

```
__fastcall HideDragImage();
```

Description

Most applications do not need to use HideDragImage to hide the current drag image. The drag control object calls this method when it is needed.

See Also

ShowDragImage (see page 6029)

32.1.18.2.8 TDragImageList.Initialize Method

Sets numerous image list properties to their default values and creates instances of objects used internal to the image list.

Pascal

```
procedure Initialize; override;
```

C++

```
virtual __fastcall Initialize();
```

Description

Initialize is called from the constructors of the image list.

See Also

Create

32.1.18.2.9 TDragImageList.SetDragImage Method

Sets an image within the image list to be shown during a drag and drop operation.

Pascal

```
function SetDragImage(Index: Integer; HotSpotX: Integer; HotSpotY: Integer): Boolean;
```

C++

```
__fastcall Boolean SetDragImage(int Index, int HotSpotX, int HotSpotY);
```

Description

Use the SetDragImage method to set the image specified by Index within the image list to be shown when the image list is being used for drag and drop. The hot spot of the image is specified by HotSpotX and HotSpotY.

See Also

ShowDragImage (see page 6029)

GetHotSpot (see page 6028)

32.1.18.2.10 TDragImageList.ShowDragImage Method

Shows the current drag image if it was previously hidden.

Pascal

```
procedure ShowDragImage;
```

C++

```
__fastcall ShowDragImage();
```

Description

Most applications do not need to use ShowDragImage to display the current drag image. The drag control object calls this method when it is needed.

See Also

HideDragImage (see page 6028)

32.1.18.3 TDragImageList Properties

The properties of the TDragImageList class are listed here.

Public Properties

	Name	Description
	DragCursor (see page 6030)	Specifies the current drag cursor that is associated with the image list.

R	Dragging (see page 6030)	Indicates whether an image in the image list is being dragged.
	DragHotspot (see page 6030)	This is DragHotspot, a member of class TDragImageList.

32.1.18.3.1 TDragImageList.DragCursor Property

Specifies the current drag cursor that is associated with the image list.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use DragCursor to determine which cursor should be displayed when an image in the image list is dragged. When an image is dragged, a new drag image is created by combining the given image (typically a mouse cursor image) with the current drag cursor.

See Also

Cursor (see page 5815)

DragCursor (see page 5818)

32.1.18.3.2 TDragImageList.Dragging Property

Indicates whether an image in the image list is being dragged.

Pascal

```
property Dragging: Boolean;
```

C++

```
__property Boolean Dragging;
```

Description

When Dragging returns true, an image in the image list is being dragged. When Dragging returns false, no image is being dragged.

See Also

BeginDrag (see page 6026)

32.1.18.3.3 TDragImageList.DragHotspot Property

Pascal

```
property DragHotspot: TPoint;
```

C++

```
__property TPoint DragHotspot;
```

Description

This is DragHotspot, a member of class TDragImageList.

32.1.19 TDragObject Class

TDragObject is a base class used for managing drag operations.

Class Hierarchy

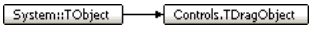

Pascal

```
TDragObject = class(TObject);
```

C++

```
class TDragObject : public TObject;
```

File

Controls

Description

Use TDragObject as a base class when creating a custom drag object to manage drag-and-drop operations or drag-and-dock operations. If the drag operation is associated with a single control that is dragged, derive the custom drag object from TBaseDragControlObject (see page 5719) instead. Create the custom drag object in the OnStartDrag event of the dragged control. Use the public IsDragObject (see page 6259) function within a target's OnDragOver event when accepting a drag drop.

Note: TDragObject is not automatically freed at the end of a drag operation. To work with a drag object that is freed at the end of the drag operation, use TDragObjectEx (see page 6041) instead.

In Applications that do not need to override the virtual methods of TDragObject, a drag object is created automatically to manage drag operations.

TDragObject allows more flexible drag operation handling. Normally, the Source parameter of the OnDragOver and OnDragDrop events is the control that starts the drag operation. If multiple controls of differing kinds need to start a drag of the same kind of data (for example, a file name, text, or a dollar amount), the source would need support for each kind of control. A drag object allows the target to handle drag sources as a single class of object, because each source control creates the same kind of drag object in their OnStartDrag events. The OnDragOver and OnDragDrop event handlers can tell if the source is a drag object, as opposed to the control, by calling IsDragObject (see page 6259).

Drag objects can be dragged between multiple .DLLs as well as inside the main .EXE. This is useful if you are not using packages but still want drag operations to function from forms implemented in .EXEs to forms implemented in .DLLs.

See Also

TDragControlObject (see page 6003)

TControl (see page 5724)

IsDragObject (see page 6259)

TDragObjectEx (see page 6041)

Customizing Drag and Drop with a Drag Object

32.1.19.1 TDragObject Members

The following tables list the members exposed by TDragObject.

Protected Methods

	Name	Description
	Finished (see page 6035)	Responds when the mouse is released indicating the drag operation is complete.
	GetDragCursor (see page 6035)	Indicates which cursor should be displayed in response to mouse moves.
	GetDragImages (see page 6035)	Returns an image list containing the image to be displayed while dragging.
	MainWndProc (see page 6037)	This is MainWndProc, a member of class TDragObject.
	WndProc (see page 6038)	This is WndProc, a member of class TDragObject.

Public Methods

	Name	Description
	AfterConstruction (see page 6033)	Responds after the last constructor has executed.
	Assign (see page 6034)	Copies the properties of another drag control object.
	BeforeDestruction (see page 6034)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetName (see page 6036)	Returns a string that identifies the drag operation.
	HideDragImage (see page 6036)	Provides the interface for a method that hides the current drag image.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Instance (see page 6037)	Returns the instance (module) handle of the EXE or DLL that created the drag object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ShowDragImage (see page 6037)	Provides the interface for a method that hides the current drag image.

Public Properties

	Name	Description
	AlwaysShowDragImages (see page 6038)	This is AlwaysShowDragImages, a member of class TDragObject.
	Cancelling (see page 6038)	Specifies whether the drag operation is being canceled.
	DragHandle (see page 6039)	Specifies the window handle of the dragged object.
	DragPos (see page 6039)	Specifies the position of the dragged object.
	DragTarget (see page 6039)	Specifies the dragged object's target.
	DragTargetPos (see page 6040)	Specifies the position of the drag object relative to the drag target.
R	Dropped (see page 6040)	Drag ended with drop.
R	MouseDeltaX (see page 6040)	Specifies the x-coordinate of the mouse position relative to the position of the dragged object's target.
R	MouseDeltaY (see page 6041)	Specifies the y-coordinate of the mouse position relative to the position of the dragged object's target.
	RightClickCancels (see page 6041)	This is RightClickCancels, a member of class TDragObject.

32.1.19.2 TDragObject Methods

The methods of the TDragObject class are listed here.

Protected Methods

	Name	Description
V	Finished (see page 6035)	Responds when the mouse is released indicating the drag operation is complete.
V	GetDragCursor (see page 6035)	Indicates which cursor should be displayed in response to mouse moves.
V	GetDragImages (see page 6035)	Returns an image list containing the image to be displayed while dragging.
	MainWndProc (see page 6037)	This is MainWndProc, a member of class TDragObject.
V	WndProc (see page 6038)	This is WndProc, a member of class TDragObject.

Public Methods

	Name	Description
V	AfterConstruction (see page 6033)	Responds after the last constructor has executed.
V	Assign (see page 6034)	Copies the properties of another drag control object.
V	BeforeDestruction (see page 6034)	Responds before the first destructor executes.
V	GetName (see page 6036)	Returns a string that identifies the drag operation.
V	HideDragImage (see page 6036)	Provides the interface for a method that hides the current drag image.
V	Instance (see page 6037)	Returns the instance (module) handle of the EXE or DLL that created the drag object.
V	ShowDragImage (see page 6037)	Provides the interface for a method that hides the current drag image.

32.1.19.2.1 TDragObject.AfterConstruction Method

Responds after the last constructor has executed.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

The System::TObject::AfterConstruction method implemented in System::TObject::TObject does nothing. Override this method when creating a class that takes some action after the object is created. For example, TCustomForm overrides

System::TObject::AfterConstruction to generate an OnCreate event.

See Also

System::TObject::BeforeDestruction (see page 22289)

OnCreate (see page 13951)

System::TObject::Create (see page 22287)

32.1.19.2.2 TDragObject.Assign Method

Copies the properties of another drag control object.

Pascal

```
procedure Assign(Source: TDragObject); virtual;
```

C++

```
virtual __fastcall Assign(TDragObject Source);
```

Description

Call Assign to copy the properties of another drag object. Assign changes the DragTarget (see page 6039), DragHandle (see page 6039), DragPos (see page 6039), DragTargetPos (see page 6040), MouseDeltaX (see page 6040) and MouseDeltaY (see page 6041) properties of the drag object to match those supplied by the Source parameter.

See Also

DragTarget (see page 6039)

DragHandle (see page 6039)

DragPos (see page 6039)

DragTargetPos (see page 6040)

MouseDeltaX (see page 6040)

MouseDeltaY (see page 6041)

32.1.19.2.3 TDragObject.BeforeDestruction Method

Responds before the first destructor executes.

Pascal

```
procedure BeforeDestruction; override;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

System::TObject::BeforeDestruction is called automatically before the object's first destructor executes. Do not call it explicitly in your applications.

The System::TObject::BeforeDestruction method implemented in System::TObject::TObject does nothing. Override this method when creating a class that takes some action before the object is destroyed. For example, TCustomForm overrides System::TObject::BeforeDestruction to generate an OnDestroy event.

Note: System::TObject::BeforeDestruction is not called when the object is destroyed before it is fully constructed. That is, if the object's constructor raises an exception, the destructor is called to dispose of the object, but System::TObject::BeforeDestruction

is not called.

See Also

System::TObject::AfterConstruction (see page 22289)

OnDestroy (see page 13953)

System::TObject::Destroy (see page 22287)

32.1.19.2.4 TDragObject.Finished Method

Responds when the mouse is released indicating the drag operation is complete.

Pascal

```
procedure Finished(Target: TObject; X: Integer; Y: Integer; Accepted: Boolean); virtual;
```

C++

```
virtual __fastcall Finished(TObject * Target, int X, int Y, Boolean Accepted);
```

Description

Finished does nothing as implemented in TDragObject (see page 6031). Override this method to provide more meaningful behavior. The Target is the object that accepted the drag. If the drag operation is canceled or a drag-and-dock operation ends with the dragged object in a free-floating position, Target is nil (Delphi) or NULL (C++). X and Y indicate the position where the drag ended. Accepted is true if the Target accepted the dragged object.

Finished is overridden and implemented in TBaseDragControlObject (see page 5719).

See Also

Finished (see page 5723)

32.1.19.2.5 TDragObject.GetDragCursor Method

Indicates which cursor should be displayed in response to mouse moves.

Pascal

```
function GetDragCursor(Accepted: Boolean; X: Integer; Y: Integer): TCursor; virtual;
```

C++

```
virtual __fastcall TCursor GetDragCursor(Boolean Accepted, int X, int Y);
```

Description

When the mouse is moved, GetDragCursor is called automatically. The Accepted parameter indicates whether the object under the mouse accepts the drag. The X and Y parameters represent the current position of the mouse. The TCursor (see page 6305) value returned by GetDragCursor is used to indicate the current drag cursor.

GetDragCursor returns crDrag (see page 6362) to when Accepted is true, crNoDrop (see page 6366) otherwise.

See Also

DragCursor (see page 5818)

32.1.19.2.6 TDragObject.GetDragImages Method

Returns an image list containing the image to be displayed while dragging.

Pascal

```
function GetDragImages: TDragImageList; virtual;
```

C++

```
virtual __fastcall TDragImageList GetDragImages();
```

Description

GetDragImages is called at the beginning of a drag and drop operation. As implemented in TDragObject (see page 6031), GetDragImages returns nil (Delphi) or NULL (C++), meaning that only the cursor is used to indicate the drag operation.

In descendant classes, the image list normally contains only one image, and the drag object automatically creates and maintains the image list.

See Also

TImageList (see page 6078)

32.1.19.2.7 TDragObject.GetName Method

Returns a string that identifies the drag operation.

Pascal

```
function GetName: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetName();
```

Description

By default GetName returns the name of the drag object class, but can be overridden to return a more meaningful string.

See Also

ClassName (see page 22290)

32.1.19.2.8 TDragObject.HideDragImage Method

Provides the interface for a method that hides the current drag image.

Pascal

```
procedure HideDragImage; virtual;
```

C++

```
virtual __fastcall HideDragImage();
```

Description

Because TDragObject (see page 6031) does not implement drag images (relying only on the cursor) HideDragImage does nothing. Descendant classes override HideDragImage to hide the drag image provided by the GetDragImages (see page 6035) method.

See Also

ShowDragImage (see page 6037)

GetDragCursor (see page 6035)

GetDragImages (see page 6035)

32.1.19.2.9 TDragObject.Instance Method

Returns the instance (module) handle of the EXE or DLL that created the drag object.

Pascal

```
function Instance: THandle; virtual;
```

C++

```
virtual __fastcall THandle Instance();
```

Description

Use Instance to obtain the instance handle for loading a resource, such as a cursor, associated with the drag object. See the example for the Cursors property of TScreen for an example of using the instance handle to load a cursor.

See Also

Cursors (see page 14212)

DragHandle (see page 6039)

32.1.19.2.10 TDragObject.MainWndProc Method

Pascal

```
procedure MainWndProc(var Message: TMessage);
```

C++

```
__fastcall MainWndProc(TMessage Message);
```

Description

This is MainWndProc, a member of class TDragObject.

32.1.19.2.11 TDragObject.ShowDragImage Method

Provides the interface for a method that hides the current drag image.

Pascal

```
procedure ShowDragImage; virtual;
```

C++

```
virtual __fastcall ShowDragImage();
```

Description

Because TDragObject (see page 6031) does not implement drag images (relying only on the cursor) ShowDragImage does nothing. Descendant classes override ShowDragImage to show the drag image.

See Also

HideDragImage (see page 6036)

GetDragCursor (see page 6035)

GetDragImages (see page 6035)

32.1.19.2.12 TDragObject.WndProc Method

Pascal

```
procedure WndProc(var Msg: TMessage); virtual;
```

C++

```
virtual __fastcall WndProc(TMessage Msg);
```

Description

This is WndProc, a member of class TDragObject.

32.1.19.3 TDragObject Properties

The properties of the TDragObject class are listed here.

Public Properties

	Name	Description
	AlwaysShowDragImages (see page 6038)	This is AlwaysShowDragImages, a member of class TDragObject.
	Cancelling (see page 6038)	Specifies whether the drag operation is being canceled.
	DragHandle (see page 6039)	Specifies the window handle of the dragged object.
	DragPos (see page 6039)	Specifies the position of the dragged object.
	DragTarget (see page 6039)	Specifies the dragged object's target.
	DragTargetPos (see page 6040)	Specifies the position of the drag object relative to the drag target.
R	Dropped (see page 6040)	Drag ended with drop.
R	MouseDeltaX (see page 6040)	Specifies the x-coordinate of the mouse position relative to the position of the dragged object's target.
R	MouseDeltaY (see page 6041)	Specifies the y-coordinate of the mouse position relative to the position of the dragged object's target.
	RightClickCancels (see page 6041)	This is RightClickCancels, a member of class TDragObject.

32.1.19.3.1 TDragObject.AlwaysShowDragImages Property

Pascal

```
property AlwaysShowDragImages: Boolean;
```

C++

```
__property Boolean AlwaysShowDragImages;
```

Description

This is AlwaysShowDragImages, a member of class TDragObject.

32.1.19.3.2 TDragObject.Cancelling Property

Specifies whether the drag operation is being canceled.

Pascal

```
property Cancelling: Boolean;
```

C++

```
__property Boolean Cancelling;
```

Description

Use Cancelling to specify whether the drag operation is being canceled. Cancelling is set to true in the global CancelDrag (see page 6247) procedure to prevent the canceling code from being executed more than once.

See Also

CancelDrag (see page 6247)

32.1.19.3.3 TDragObject.DragHandle Property

Specifies the window handle of the dragged object.

Pascal

```
property DragHandle: HWND;
```

C++

```
__property HWND DragHandle;
```

Description

Use DragHandle to get or set the window handle of the dragged object.

See Also

Handle (see page 6207)

32.1.19.3.4 TDragObject.DragPos Property

Specifies the position of the dragged object.

Pascal

```
property DragPos: TPoint;
```

C++

```
__property TPoint DragPos;
```

Description

Use DragPos to get the position of the dragged object.

See Also

DragTargetPos (see page 6040)

32.1.19.3.5 TDragObject.DragTarget Property

Specifies the dragged object's target.

Pascal

```
property DragTarget: Pointer;
```

C++

```
__property void * DragTarget;
```

Description

Use DragTarget to get the control (if any) that is the current drop target of the drag operation. If there is no control that will accept the dragged object if it is released, DragTarget is nil (Delphi) or NULL (C++).

See Also

DragTargetPos (see page 6040)

32.1.19.3.6 TDragObject.DragTargetPos Property

Specifies the position of the drag object relative to the drag target.

Pascal

```
property DragTargetPos: TPoint;
```

C++

```
__property TPoint DragTargetPos;
```

Description

Use DragTargetPos to get the position of the dragged object relative to the upper left corner of the drag target. To get the position of the dragged object in screen coordinates, use the DragPos (see page 6039) property instead.

See Also

DragTarget (see page 6039)

DragPos (see page 6039)

32.1.19.3.7 TDragObject.Dropped Property

Drag ended with drop.

Pascal

```
property Dropped: Boolean;
```

C++

```
__property Boolean Dropped;
```

Description

The Dropped property contains true if the last drag operation ended with a successful drop.

32.1.19.3.8 TDragObject.MouseDeltaX Property

Specifies the x-coordinate of the mouse position relative to the position of the dragged object's target.

Pascal

```
property MouseDeltaX: Double;
```

C++

```
__property Double MouseDeltaX;
```

Description

Use MouseDeltaX to determine the x-coordinate of the mouse position relative to the position where the dragged object would appear if released on the current target.

See Also

MouseDeltaY (see page 6041)

32.1.19.3.9 TDragObject.MouseDeltaY Property

Specifies the y-coordinate of the mouse position relative to the position of the dragged object's target.

Pascal

```
property MouseDeltaY: Double;
```

C++

```
__property Double MouseDeltaY;
```

Description

Use MouseDeltaY to determine the y-coordinate of the mouse position relative to the position where the dragged object would appear if released on the current target.

See Also

MouseDeltaX (see page 6040)

32.1.19.3.10 TDragObject.RightClickCancels Property

Pascal

```
property RightClickCancels: Boolean;
```

C++

```
__property Boolean RightClickCancels;
```

Description

This is RightClickCancels, a member of class TDragObject.

32.1.20 TDragObjectEx Class

TDragObjectEx is a base class used for managing drag operations.

Class Hierarchy

Pascal

```
TDragObjectEx = class(TDragObject);
```

C++

```
class TDragObjectEx : public TDragObject;
```

File

Controls

Description

Use TDragObjectEx as a base class when creating a custom drag object to manage drag-and-drop operations or drag-and-dock operations. Unlike its immediate ancestor, TDragObject (see page 6031), TDragObjectEx and its descendants are automatically freed at the end of a drag operation.

Applications can create the custom drag object in the OnStartDrag event of the dragged control. Use the public IsDragObject (see page 6259) function within a target's OnDragOver event when accepting a drag drop.

In Applications that do not need to override the virtual methods of TDragObjectEx, a drag object is created automatically to manage drag operations.

Drag objects allow more flexible drag operation handling. Normally, the Source parameter of the OnDragOver and OnDragDrop events is the control that starts the drag operation. If multiple controls of differing kinds need to start a drag of the same kind of data (for example, a file name, text, or a dollar amount), the source would need support for each kind of control. A drag object allows the target to handle drag sources as a single class of object, because each source control creates the same kind of drag object in their OnStartDrag events. The OnDragOver and OnDragDrop event handlers can tell if the source is a drag object, as opposed to the control, by calling IsDragObject (see page 6259).

Drag objects can be dragged between multiple .DLLs as well as inside the main .EXE. This is useful if you are not using packages but still want drag operations to function from forms implemented in .EXEs to forms implemented in .DLLs.

See Also

TDragControlObjectEx (see page 6007)

TDragDockObjectEx (see page 6018)

TControl (see page 5724)

IsDragObject (see page 6259)

TDragObject (see page 6031)

32.1.20.1 TDragObjectEx Members

The following tables list the members exposed by TDragObjectEx.

Protected Methods

	Name	Description
	Finished (see page 6035)	Responds when the mouse is released indicating the drag operation is complete.
	GetDragCursor (see page 6035)	Indicates which cursor should be displayed in response to mouse moves.
	GetDragImages (see page 6035)	Returns an image list containing the image to be displayed while dragging.
	MainWndProc (see page 6037)	This is MainWndProc, a member of class TDragObject.
	WndProc (see page 6038)	This is WndProc, a member of class TDragObject.

Public Methods

	Name	Description
	AfterConstruction (see page 6033)	Responds after the last constructor has executed.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 6034)	Copies the properties of another drag control object.
	BeforeDestruction (see page 6043)	Allows the application to detect that the drag object needs to be freed.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetName (see page 6036)	Returns a string that identifies the drag operation.
V	HideDragImage (see page 6036)	Provides the interface for a method that hides the current drag image.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Instance (see page 6037)	Returns the instance (module) handle of the EXE or DLL that created the drag object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	ShowDragImage (see page 6037)	Provides the interface for a method that hides the current drag image.

Public Properties

	Name	Description
	AlwaysShowDragImages (see page 6038)	This is AlwaysShowDragImages, a member of class TDragObject.
	Cancelling (see page 6038)	Specifies whether the drag operation is being canceled.
	DragHandle (see page 6039)	Specifies the window handle of the dragged object.
	DragPos (see page 6039)	Specifies the position of the dragged object.
	DragTarget (see page 6039)	Specifies the dragged object's target.
	DragTargetPos (see page 6040)	Specifies the position of the drag object relative to the drag target.
R	Dropped (see page 6040)	Drag ended with drop.
R	MouseDeltaX (see page 6040)	Specifies the x-coordinate of the mouse position relative to the position of the dragged object's target.
R	MouseDeltaY (see page 6041)	Specifies the y-coordinate of the mouse position relative to the position of the dragged object's target.
	RightClickCancels (see page 6041)	This is RightClickCancels, a member of class TDragObject.

32.1.20.2 TDragObjectEx Methods

The methods of the TDragObjectEx class are listed here.

Public Methods

	Name	Description
V	BeforeDestruction (see page 6043)	Allows the application to detect that the drag object needs to be freed.

32.1.20.2.1 TDragObjectEx.BeforeDestruction Method

Allows the application to detect that the drag object needs to be freed.

Pascal

```
procedure BeforeDestruction; override;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

Do not call BeforeDestruction. This method is called automatically before any TDragObjectEx (see page 6041) destructors are called. It suppresses the inherited method, which prevents the drag object from being freed at the end of a drag operation.

32.1.21 TGraphicControl Class

TGraphicControl is the base class for all lightweight controls.

Class Hierarchy

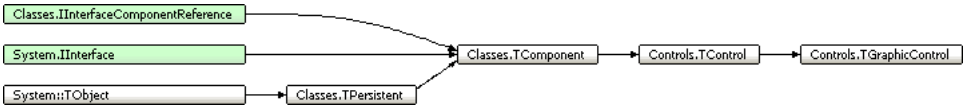

Pascal

```
TGraphicControl = class(TControl);
```

C++

```
class TGraphicControl : public TControl;
```

File

Controls

Description

TGraphicControl supports simple lightweight controls that do not need the ability to accept keyboard input or contain other controls. Since lightweight controls do not wrap Windows screen objects, they are faster and user fewer resources than controls based on TWinControl (see page 6112).

TGraphicControl provides a Canvas (see page 6053) property for access to the control's drawing surface and a virtual Paint (see page 6052) method called in response to paint requests received by the parent control.

32.1.21.1 TGraphicControl Members

The following tables list the members exposed by TGraphicControl.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Paint (see page 6052)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.

V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.

	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 6051)	Creates an instance of TGraphicControl (see page 6044).
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.

	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.

	Parent (see page 5832)	Indicates the parent of the control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

32.1.21.2 TGraphicControl.Create Constructor

Creates an instance of TGraphicControl (see page 6044).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TGraphicControl(TComponent * AOwner);
```

Description

Calling Create constructs and initializes an instance of TGraphicControl (see page 6044). However, you should never attempt to instantiate a TGraphicControl (see page 6044). This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants.

AOwner is the component, typically a form, that is responsible for freeing the graphic control. It becomes the value of the Owner property. If you do not explicitly provide an Owner to the constructor (that is, if AOwner is nil (Delphi) or NULL (C++)), your application is responsible for explicitly freeing the graphic control.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Owner (see page 2807)

Create (see page 5731)

Destroy (see page 6052)

32.1.21.3 TGraphicControl.Destroy Destructor

Destroys an instance of TGraphicControl (see page 6044).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TGraphicControl();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the graphic control reference is not nil, and only then calls Destroy.

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Frees the TCanvas object in its Canvas (see page 6053) property.

Override Destroy to free any memory or resources allocated in the Create (see page 6051) method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

Create (see page 6051)

Free (see page 22300)

32.1.21.4 TGraphicControl Methods

The methods of the TGraphicControl class are listed here.

Protected Methods

	Name	Description
V	Paint (see page 6052)	Render control surface.

32.1.21.4.1 TGraphicControl.Paint Method

Render control surface.

Pascal

```
procedure Paint; virtual;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas (see page 6053) property to render the control surface.

As implemented in TGraphicControl (see page 6044), Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas (see page 6053)

32.1.21.5 TGraphicControl Properties

The properties of the TGraphicControl class are listed here.

Protected Properties

	Name	Description
	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.

32.1.21.5.1 TGraphicControl.Canvas Property

Provides the drawing surface used by the graphic control.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control.

Since a TGraphicControl (see page 6044) instance does not have its own Windows screen object, it obtains its device context from its parent control.

C++ Examples:

```
/*
When you do custom drawing, such as shown in the code for
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   PaintBox1->Canvas->Brush->Color = clRed;
   PaintBox1->Canvas->FillRect(PaintBox1->Canvas->ClipRect);
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}

/*
In order for the drawing to be persistent, you need to also
include an OnPaint event handler that tells the PaintBox how
to redraw itself when it needs to. In the code below, an
ellipse will be drawn everytime the PaintBox renders itself,
but the PaintBox will only paint the client area of the
control red after the Button is clicked. This client area
will only stay red until it is invalidated, but the ellipse
will persist. This behavior is different from the TImage
component, because the TImage component maintains an
internal bitmap that stores this drawing information for
you.
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
```

```
{
   PaintBox1->Canvas->Brush->Color = clBlue;
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TLabel *myLabel;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TLabel")
  {
    myLabel = (TLabel *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myLabel->Canvas->MoveTo(myLabel->ClientRect.Left, myLabel->ClientRect.Top);
    myLabel->Canvas->LineTo(myLabel->ClientRect.Right, myLabel->ClientRect.Bottom);
    myLabel = NULL;
  }
}
/*
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PaintBox1->Canvas->Draw(0,0, Image1->Picture->Graphic);
}
```

Delphi Examples:

```
{
When you do custom drawing, such as shown in the code for
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   PaintBox1.Canvas.Brush.Color := clRed;
   PaintBox1.Canvas.FillRect(PaintBox1.Canvas.ClipRect);
   PaintBox1.Canvas.Brush.Color := clBlue;
   PaintBox1.Canvas.Ellipse(0,0,100,50);
end;
{
In order for the drawing to be persistent, you need to also
include an OnPaint event handler that tells the PaintBox how
to redraw itself when it needs to. In the code below, an
ellipse will be drawn everytime the PaintBox renders itself,
but the PaintBox will only paint the client area of the
control red after the Button is clicked. This client area
will only stay red until it is invalidated, but the ellipse
will persist. This behavior is different from the TImage
component, because the TImage component maintains an
internal bitmap that stores this drawing information for
you.
}
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
   PaintBox1.Canvas.Brush.Color := clBlue;
   PaintBox1.Canvas.Ellipse(0,0,100,50);
```

```
end;
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TLabel) then
  begin
    myLabel:= (ListBox1.Items.Objects[Index] as TLabel);
    with myLabel.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myLabel.Canvas.MoveTo(Left, Top);
      myLabel.Canvas.LineTo(Right, Bottom);
    end;
  myLabel:= nil;
  end;
{
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  PaintBox1.Canvas.Draw(0,0, Image1.Picture.Graphic);
end;
```

32.1.22 THintWindow Class

THintWindow implements the small pop-up window that appears over a control at runtime when the control has its ShowHint (see page 5840) property set to true.

Class Hierarchy

Pascal

```
THintWindow = class(TCustomControl);
```

C++

```
class THintWindow : public TCustomControl;
```

File

Controls

Description

Use THintWindow to display a Help Hint (see page 5827) pop-up window directly from an application. Override THintWindow to customize the window that appears automatically for controls with their ShowHint (see page 5840) property set to true. After overriding THintWindow to create a new derived type, assign the new type to the global HintWindowClass variable at application startup, so that the new hint window type is used for Help Hints.

The hint window's BiDiMode (see page 6071) is always set to the BiDiMode (see page 6071) of the control that activates it. If the mode is bdRightToLeft, then the hint is left-aligned to the cursor.

See Also

TControl (see page 5724)

TApplication (see page 13660)

THintInfo

HintWindowClass

Help and Hint Properties

32.1.22.1 THintWindow Members

The following tables list the members exposed by THintWindow.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.

	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.

	CreateParams (see page 6069)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.

	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	NCPaint (see page 6069)	This is NCPaint, a member of class THintWindow.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 6070)	Renders the image of a custom control.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.

	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.

	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMPrint (see page 6071)	This is WMPrint, a member of class THintWindow.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.

	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	ActivateHint (see page 6067)	Displays the hint window.
V	ActivateHintData (see page 6067)	Displays the hint window using a provided data value.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
V	CalcHintRect (see page 6068)	Returns the size needed to display a given hint.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 6066)	Creates and initializes an instance of THintWindow (see page 6055).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.

	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
V	IsHintMsg (see page 6069)	Determines whether an application message requires hiding the hint window.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	ReleaseHandle (see page 6070)	Destroys the window displayed by THintWindow (see page 6055).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
V	ShouldHideHint (see page 6070)	This is ShouldHideHint, a member of class THintWindow.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 6071)	Specifies the bi-directional mode for the control.

	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	Canvas (see page 6072)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 6073)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 6076)	Specifies the background color of the control.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Font (see page 6077)	Controls the attributes of text written on or in the control.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.

	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

32.1.22.2 THintWindow.Create Constructor

Creates and initializes an instance of THintWindow (see page 6055).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall THintWindow(TComponent * AOwner);
```

Description

Call Create to instantiate a hint window at runtime. Hint (see page 5827) windows activated by the application in response to the ShowHint (see page 5840) property of a control are created automatically.

See Also

ActivateHint (see page 6067)

ShowHint (see page 5840)

32.1.22.3 THintWindow Methods

The methods of the THintWindow class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 6069)	Initializes a window-creation parameter data structure.
	NCPaint (see page 6069)	This is NCPaint, a member of class THintWindow.
	Paint (see page 6070)	Renders the image of a custom control.
	WMPrint (see page 6071)	This is WMPrint, a member of class THintWindow.

Public Methods

	Name	Description
	ActivateHint (see page 6067)	Displays the hint window.
	ActivateHintData (see page 6067)	Displays the hint window using a provided data value.
	CalcHintRect (see page 6068)	Returns the size needed to display a given hint.
	IsHintMsg (see page 6069)	Determines whether an application message requires hiding the hint window.
	ReleaseHandle (see page 6070)	Destroys the window displayed by THintWindow (see page 6055).
	ShouldHideHint (see page 6070)	This is ShouldHideHint, a member of class THintWindow.

32.1.22.3.1 THintWindow.ActivateHint Method

Displays the hint window.

Pascal

```
procedure ActivateHint(Rect: TRect; const AHint: string); virtual;
```

C++

```
virtual __fastcall ActivateHint(TRect Rect, const AnsiString AHint);
```

Description

Call ActivateHint to display the hint window at the coordinates provided by the Rect parameter. If the Rect parameter appears off screen, ActivateHint moves the hint window to the closest approximation to the coordinates supplied by Rect that appears entirely on screen. ActivateHint sets the Caption (see page 6073) property to the AHint parameter before showing the window, so that the AHint string appears to the user.

After an application displays a hint window, it automatically deactivates it after a time period has elapsed, or when it receives a message that requires hiding the hint message. To hide the hint window that is brought up automatically by the application, use the HideHint method of the global Application object. To hide a hint window that is activated directly by calling ActivateHint, use the ReleaseHandle (see page 6070) method.

See Also

application

IsHintMsg (see page 6069)

ReleaseHandle (see page 6070)

HideHint (see page 13681)

Caption (see page 5806)

32.1.22.3.2 THintWindow.ActivateHintData Method

Displays the hint window using a provided data value.

Pascal

```
procedure ActivateHintData(Rect: TRect; const AHint: string; AData: Pointer); virtual;
```

C++

```
virtual __fastcall ActivateHintData(TRect Rect, const AnsiString AHint, void * AData);
```

Description

Call ActivateHintData to display the hint window at the coordinates provided by the Rect parameter. The ActivateHintData method for THintWindow (see page 6055) sets the Caption (see page 6073) property to the AHint parameter and ignores the AData parameter. Descendant classes can override this method to combine the AHint and AData parameters into a customized Hint (see page 5827) value.

After an application displays a hint window, it automatically deactivates it after a time period has elapsed, or when it receives a message that requires hiding the hint message. To hide the hint window that is brought up automatically by the application, use the HideHint method of the global Application object. To hide a hint window that is activated directly by calling ActivateHint (see page 6067), use the ReleaseHandle (see page 6070) method.

See Also

ActivateHint (see page 6067)

application

IsHintMsg (see page 6069)

ReleaseHandle (see page 6070)

HideHint (see page 13681)

Caption (see page 5806)

32.1.22.3.3 THintWindow.CalcHintRect Method

Returns the size needed to display a given hint.

Pascal

```
function CalcHintRect(MaxWidth: Integer; const AHint: string; AData: Pointer): TRect; virtual;
```

C++

```
virtual __fastcall TRect CalcHintRect(int MaxWidth, const AnsiString AHint, void * AData);
```

Description

Call CalcHintRect to determine the size needed to display a hint window for a specified string and data value. The MaxWidth parameter limits how wide the Hint (see page 5827) window can grow before the text wraps to a new line.

MaxWidth is the maximum length, in pixels, for the hint window. If the hint requires more than this width, the text in the hint window wraps to a new line.

AHint is the hint string that appears in the hint window.

AData is data associated with the AHint.

Note: CalcHintRect ignores the AData parameter. Descendant classes can override this method to make use of associated hint data.

See Also

ActivateHintData (see page 6067)

32.1.22.3.4 THintWindow.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

32.1.22.3.5 THintWindow.IsHintMsg Method

Determines whether an application message requires hiding the hint window.

Pascal

```
function IsHintMsg(var Msg: TMsg): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsHintMsg(TMsg Msg);
```

Description

Call IsHintMsg to determine whether the message specified by the Msg parameter requires hiding the hint window. Upon seeing a mouse, keyboard, command, or activation message, IsHintMsg returns true. The global Application object calls IsHintMsg to check messages while the hint window is on the screen, and hides the window if IsHintMsg returns true.

See Also

application

32.1.22.3.6 THintWindow.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); virtual;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class THintWindow.

32.1.22.3.7 THintWindow.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas (see page 5921)

PaintWindow (see page 5920)

GetDeviceContext (see page 6155)

ClipRect (see page 14486)

32.1.22.3.8 THintWindow.ReleaseHandle Method

Destroys the window displayed by THintWindow (see page 6055).

Pascal

```
procedure ReleaseHandle;
```

C++

```
__fastcall ReleaseHandle();
```

Description

Call ReleaseHandle to destroy a hint window that was activated by directly calling the ActivateHint (see page 6067) method. Hint (see page 5827) windows activated automatically by the application object are deactivated by the application as well.

See Also

ActivateHint (see page 6067)

32.1.22.3.9 THintWindow.ShouldHideHint Method

Pascal

```
function ShouldHideHint: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ShouldHideHint();
```

Description

This is ShouldHideHint, a member of class THintWindow.

32.1.22.3.10 THintWindow.WMPrint Method

Pascal

```
procedure WMPrint(var Message: TMessage); message WM_PRINT;
```

C++

```
__fastcall WMPrint(TMessage Message);
```

Description

This is WMPrint, a member of class THintWindow.

32.1.22.4 THintWindow Properties

The properties of the THintWindow class are listed here.

Public Properties

	Name	Description
	BiDiMode (see page 6071)	Specifies the bi-directional mode for the control.
	Canvas (see page 6072)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 6073)	Specifies a text string that identifies the control to the user.
	Color (see page 6076)	Specifies the background color of the control.
	Font (see page 6077)	Controls the attributes of text written on or in the control.

32.1.22.4.1 THintWindow.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

32.1.22.4.2 THintWindow.Canvas Property

Specifies the TCanvas object that presents a drawing surface for the control.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control. Canvas encapsulates a Windows device context, providing all the tools and methods needed for drawing and painting.

Canvas is a protected property that is usually redeclared as public in descendants of TCustomControl.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas,Rect.Left,Rect.Top,index);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(index) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TStringGrid *myStringGrid;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TStringGrid")
  {
    myStringGrid = (TStringGrid *)ListBox1->Items->Objects[Index];
```

```
    // make sure we are using the TControls canvas and not the from's!
    myStringGrid->Canvas->MoveTo(myStringGrid->ClientRect.Left, myStringGrid->ClientRect.Top);
    myStringGrid->Canvas->LineTo(myStringGrid->ClientRect.Right,
myStringGrid->ClientRect.Bottom);
    myStringGrid = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TStringGrid) then
  begin
    myStringGrid:= (ListBox1.Items.Objects[Index] as TStringGrid);
    with myStringGrid.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myStringGrid.Canvas.MoveTo(Left, Top);
      myStringGrid.Canvas.LineTo(Right, Bottom);
    end;
  myStringGrid:= nil;
  end;
```

See Also

Paint (see page 5920)

32.1.22.4.3 THintWindow.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
```

```
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
```

```
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

32.1.22.4.4 THintWindow.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

32.1.22.4.5 THintWindow.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
```

```
  the memo or Rich Edit control font is set to the
  corresponding font name in the combo box.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    i : Integer;
  begin
    for i := 0 to Screen.Fonts.Count - 1 do
     ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
  //  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
    RichEdit1.Lines.Text := 'How is it to be you?';
  end;

  procedure TForm1.ComboBox1Click(Sender: TObject);
  begin
    RichEdit1.Font.Name := ComboBox1.Text;
  end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

32.1.23 TImageList Class

TImageList represents a collection of same-sized images, each of which can be referred to by its index.

Class Hierarchy

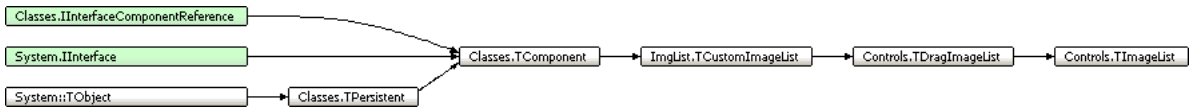

Pascal

```
TImageList = class(TDragImageList);
```

C++

```
class TImageList : public TDragImageList;
```

File

Controls

Description

Image lists are used to efficiently manage large sets of icons or bitmaps. All images in an image list are contained in a single, wide bitmap in screen device format. An image list may also include a monochrome bitmap that contains masks used to draw images transparently (icon style).

The image list is capable of holding a large number of same sized images and retrieving them via their index within the range 0 to n - 1. The image list also has methods to facilitate storing, retrieving, and drawing of the stored images.

To add images to an image list at design time, right click and choose Image List Editor

Note: Image lists depend on Comctl32.dll. If the system does not have the latest version installed, there may be problems getting images to appear.

See Also

TListView (see page 4285)

TImage (see page 12777)

List Controls

Adding Icons to Menus and Toolbars

32.1.23.1 TImageList Members

The following tables list the members exposed by TImageList.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 15370)	Copies the image list's properties to another image list.
	Change (see page 15371)	Generates an OnChange (see page 15393) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 15371)	Allows bitmap data to be saved and loaded.
V	DoDraw (see page 15372)	Draws the specified image onto the provided Canvas in the indicated style.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetImages (see page 15378)	Returns a specified image and its mask as bitmaps.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleNeeded (see page 15381)	This is HandleNeeded, a member of class TCustomImageList.
V	Initialize (see page 6028)	Sets numerous image list properties to their default values and creates instances of objects used internal to the image list.
V	Initialize (see page 15381)	Initializes a newly allocated image list.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadData (see page 15383)	This is ReadData, a member of class TCustomImageList.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.

	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteData (see page 15386)	This is WriteData, a member of class TCustomImageList.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Add (see page 15368)	Adds an Image and its Mask (see page 15797) to the image list object.
	AddIcon (see page 15368)	Adds an icon to the image list object.
	AddImage (see page 15368)	Adds single image from another image list.
	AddImages (see page 15369)	Adds all images from another image list.
	AddMasked (see page 15369)	Adds an image to the image list along with a color which is used to generate the transparent mask.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	Assign (see page 15370)	Copies the information from one image list to another.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 6026)	Starts the dragging of an image list.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 15371)	Clears all images from the image list.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 15365)	Creates an empty image list.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateSize (see page 15366)	Creates an empty image list of a specified size.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 15372)	Deletes a particular image from the image list.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 15366)	Destroys the image list and releases the memory allocated to it.
V	Destroy (see page 22287)	Disposes of an object instance.

	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DragLock (see page 6026)	Associates the drag image with the given Window handle and draws it at the position given by the parameters XPos and YPos while the window is in drag mode.
	DragMove (see page 6027)	Moves the drag image set by SetDragImage (see page 6029) to a new position specified by X and Y within its currently associated window while the window is in drag mode.
	DragUnlock (see page 6027)	Removes the association between the drag image and a window.
	Draw (see page 15373)	This is the overview for the Draw method overload.
	DrawOverlay (see page 15374)	This is the overview for the DrawOverlay method overload.
	EndDrag (see page 6028)	Ends the drag operation.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FileLoad (see page 15375)	Loads a resourced image into the image list.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetBitmap (see page 15376)	Retrieves a specified image as a bitmap.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetHotSpot (see page 6028)	Returns the hot spot location associated with the current drag image.
V	GetHotSpot (see page 15376)	Returns the hot spot location associated with the current drag image.
	GetIcon (see page 15377)	This is the overview for the GetIcon method overload.
	GetImageBitmap (see page 15378)	Returns a handle to a bitmap containing all of the images in the image list.
	GetInstRes (see page 15378)	This is the overview for the GetInstRes method overload.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaskBitmap (see page 15379)	Returns a handle to a bitmap containing all masks for images in the image list.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetResource (see page 15380)	Loads the specified bitmap, cursor, or icon resource into the image list.
	HandleAllocated (see page 15380)	Indicates whether a window handle exists for the image list.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HideDragImage (see page 6028)	Hides the current drag image if it was previously shown.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 15381)	Inserts a bitmap and its mask into the image list following a specified image.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertIcon (see page 15382)	Inserts an icon at a specified position in the image list.
	InsertMasked (see page 15382)	Inserts a bitmap at a specified position, creating the mask from a specified transparent color.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	Move (see page 15382)	Moves a specified image to another position.

V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Overlay (see page 15383)	Identifies an image in the list as an image that can be used as an overlay mask.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 15383)	Registers an object to be notified when changes occur in the image list.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Replace (see page 15384)	Replaces an image in the image list with a new image and mask.
	ReplaceIcon (see page 15384)	Replaces an image in the image list with an icon.
	ReplaceMasked (see page 15384)	Replaces an image in the image list with a new image and automatically-generated mask.
	ResInstLoad (see page 15385)	Loads a resource into the image list.
	ResourceLoad (see page 15385)	Loads a resource into the image list.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetDragImage (see page 6029)	Sets an image within the image list to be shown during a drag and drop operation.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	ShowDragImage (see page 6029)	Shows the current drag image if it was previously hidden.
	UnRegisterChanges (see page 15386)	Removes an object from the list of objects to be notified when the image list changes.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Count (see page 15388)	Indicates the number of images in the image list.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DragCursor (see page 6030)	Specifies the current drag cursor that is associated with the image list.
R	Dragging (see page 6030)	Indicates whether an image in the image list is being dragged.
	DragHotspot (see page 6030)	This is DragHotspot, a member of class TDragImageList.
	Handle (see page 15390)	Designates the handle for the image list.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 6087)	Occurs when the image list changes.

Published Properties

	Name	Description
	AllocBy (see page 6083)	Sets the number of images by which the image list grows when it needs to make room for new images.
	BkColor (see page 6083)	Determines which background color to use when drawing an image.
	BlendColor (see page 6084)	Determines which foreground color to use when drawing an image.
	DrawingStyle (see page 6084)	Specifies the style to be used when the image list is drawing an image.
	Height (see page 6085)	Specifies the height of images within the image list.

	ImageType (see page 6085)	Determines whether the image list uses the associated mask when drawing.
	Masked (see page 6086)	Specifies whether the image list includes masks that can be combined with the images.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ShareImages (see page 6086)	Specifies whether the image list destroys its handle when the image list is destroyed.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Width (see page 6086)	Specifies the width of all images within the image list.

32.1.23.2 TImageList Properties

The properties of the TImageList class are listed here.

Published Properties

	Name	Description
	AllocBy (see page 6083)	Sets the number of images by which the image list grows when it needs to make room for new images.
	BkColor (see page 6083)	Determines which background color to use when drawing an image.
	BlendColor (see page 6084)	Determines which foreground color to use when drawing an image.
	DrawingStyle (see page 6084)	Specifies the style to be used when the image list is drawing an image.
	Height (see page 6085)	Specifies the height of images within the image list.
	ImageType (see page 6085)	Determines whether the image list uses the associated mask when drawing.
	Masked (see page 6086)	Specifies whether the image list includes masks that can be combined with the images.
	ShareImages (see page 6086)	Specifies whether the image list destroys its handle when the image list is destroyed.
	Width (see page 6086)	Specifies the width of all images within the image list.

32.1.23.2.1 TImageList.AllocBy Property

Sets the number of images by which the image list grows when it needs to make room for new images.

Pascal

```
property AllocBy: Integer;
```

C++

```
__property int AllocBy;
```

Description

Use the AllocBy method to specify the number of new images that can be added to the resized image list.

See Also

Count (see page 15388)

Add (see page 15368)

AddIcon (see page 15368)

AddImages (see page 15369)

AddMasked (see page 15369)

32.1.23.2.2 TImageList.BkColor Property

Determines which background color to use when drawing an image.

Pascal

```
property BkColor: TColor;
```

C++

```
__property TColor BkColor;
```

Description

Use BkColor to specify the color used for the masked regions of the images in the list. When BkColor is clNone, the image is drawn transparently using the mask to define the transparent areas. Otherwise, the value of BkColor appears in the masked out portions.

Note: BkColor has no effect when the Masked property is false.

See Also

BlendColor (see page 15387)

Masked (see page 15392)

ImageType (see page 15391)

32.1.23.2.3 TImageList.BlendColor Property

Determines which foreground color to use when drawing an image.

Pascal

```
property BlendColor: TColor;
```

C++

```
__property TColor BlendColor;
```

Description

Set BlendColor to cause the images in the list to be combined with the specified color when the DrawingStyle is dsFocus or dsSelected. More of BlendColor is combined with the images when DrawingStyle is dsSelected. clNone specifies no blend color while clDefault specifies the system highlight .

See Also

BkColor (see page 15387)

DrawingStyle (see page 15390)

32.1.23.2.4 TImageList.DrawingStyle Property

Specifies the style to be used when the image list is drawing an image.

Pascal

```
property DrawingStyle: TDrawingStyle;
```

C++

```
__property TDrawingStyle DrawingStyle;
```

Description

Use the DrawingStyle property to control how the image is to appear. DrawingStyle controls how much of the blend color specified by BlendColor is combined with the foreground, or what background color to use.

See Also

Draw (see page 15373)

DrawOverlay (see page 15375)

ImageType (see page 15391)

BlendColor (see page 15387)

BkColor (see page 15387)

32.1.23.2.5 TImageList.Height Property

Specifies the height of images within the image list.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use the Height property to specify the height of images before any are added to the image list. The image list is cleared when this value changes.

See Also

Width (see page 15392)

CreateSize (see page 15366)

Create (see page 15365)

32.1.23.2.6 TImageList.ImageType Property

Determines whether the image list uses the associated mask when drawing.

Pascal

```
property ImageType: TImageType;
```

C++

```
__property TImageType ImageType;
```

Description

Use the ImageType property to control whether the images are drawn using the image mask (itMask), or without the mask (itImage).

See Also

DrawingStyle (see page 15390)

Draw (see page 15373)

Masked (see page 15392)

DrawOverlay (see page 15375)

32.1.23.2.7 TImageList.Masked Property

Specifies whether the image list includes masks that can be combined with the images.

Pascal

```
property Masked: Boolean;
```

C++

```
__property Boolean Masked;
```

Description

Set the Masked property to true to combine the images with masks. The masked portion of the bitmap is drawn transparently, or replaced by the color specified by the BkColor property.

Masked indicates whether the image list stores masks to combine with the images. To use those masks, the ImageType property indicates whether to draw the image or the mask.

See Also

AddMasked (see page 15369)

InsertMasked (see page 15382)

BkColor (see page 15387)

ImageType (see page 15391)

32.1.23.2.8 TImageList.ShareImages Property

Specifies whether the image list destroys its handle when the image list is destroyed.

Pascal

```
property ShareImages: Boolean;
```

C++

```
__property Boolean ShareImages;
```

Description

Set the ShareImages property to false to ensure that the handle to the image list is freed when the image list is destroyed.

See Also

Destroy (see page 15366)

Handle (see page 15390)

32.1.23.2.9 TImageList.Width Property

Specifies the width of all images within the image list.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use the Width property to specify the width of images before any are added to the image list. The image list is cleared when this

value changes.

See Also

Height (see page 15391)

Create (see page 15365)

CreateSize (see page 15366)

32.1.23.3 TImageList Events

The events of the TImageList class are listed here.

Published Events

	Name	Description
	OnChange (see page 6087)	Occurs when the image list changes.

32.1.23.3.1 TImageList.OnChange Event

Occurs when the image list changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to execute code when the image list changes. Any objects registered with the RegisterChanges method will also be notified.

See Also

RegisterChanges (see page 15383)

32.1.24 TMargins Class

Contains margins for the control.

Class Hierarchy

Pascal

```
TMargins = class(TPersistent);
```

C++

```
class TMargins : public TPersistent;
```

File

Controls

Description

TMargins is used in the Margins property of TControl (see page 5724) and its descendants. TMargins help define the relative position between components on a form, and between the edges of the form and the component. For example, when you set a left margin for a component to 10 pixels, the component will not come closer than 10 pixels to the edge of the container, or to another component on the left edge. The number of pixels by which two components are separated is the sum of the pixels of both components.

For example, if you place a component on the form with a left margin of 10 pixels, then add another component to it's left with a right margin of 20 pixels, the closed the two components can come together on those edges is 30 pixels.

You can define the amount of margin that should surround the component on the top, left, bottom, or right by changing the pixel value for the `Margins` property in the **Object Inspector**.

Note: For the Margins

settings to take effect on a component, you must also set its `AlignWithMargins` property to **True**.

32.1.24.1 TMargins Members

The following tables list the members exposed by TMargins.

Protected Methods

	Name	Description
	AssignTo (see page 6090)	Copies the properties of an object to a destination object.
	Change (see page 6091)	This is Change, a member of class TMargins.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetControlBound (see page 6091)	This is GetControlBound, a member of class TMargins.
	GetOwner (see page 2932)	Returns the owner of an object.
	InitDefaults (see page 6091)	This is InitDefaults, a member of class TMargins.

Protected Properties

	Name	Description
	Control (see page 6093)	This is Control, a member of class TMargins.

Public Events

	Name	Description
	OnChange (see page 6096)	Occurs when margins change.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6089)	This is Create, a member of class TMargins.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetBounds (see page 6091)	Sets the properties for the left, top, right, and bottom margins all at once.
	SetControlBounds (see page 6091)	This is the overview for the SetControlBounds method overload.

Public Properties

	Name	Description
	ControlHeight (see page 6093)	Sets the height of the control.
	ControlLeft (see page 6093)	Sets the starting point for the left side of the control.
	ControlTop (see page 6094)	Sets the starting point for the top of the control.
	ControlWidth (see page 6094)	Sets the width of the control.
	ExplicitHeight (see page 6094)	This is ExplicitHeight, a member of class TMargins.
	ExplicitLeft (see page 6094)	This is ExplicitLeft, a member of class TMargins.
	ExplicitTop (see page 6095)	This is ExplicitTop, a member of class TMargins.
	ExplicitWidth (see page 6095)	This is ExplicitWidth, a member of class TMargins.

Published Properties

	Name	Description
	Bottom (see page 6093)	Specifies the bottom edge of the margin.
	Left (see page 6095)	Specifies the left edge of the margin.
	Right (see page 6095)	Specifies the right edge of the margin.
	Top (see page 6095)	Specifies the top edge of the margin.

32.1.24.2 TMargins.Create Constructor

Pascal

```
constructor Create(Control: TControl); virtual;
```

C++

```
virtual __fastcall TMargins(TControl * Control);
```

Description

This is Create, a member of class TMargins.

32.1.24.3 TMargins Methods

The methods of the TMargins class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 6090)	Copies the properties of an object to a destination object.
	Change (see page 6091)	This is Change, a member of class TMargins.
	GetControlBound (see page 6091)	This is GetControlBound, a member of class TMargins.
	InitDefaults (see page 6091)	This is InitDefaults, a member of class TMargins.

Public Methods

	Name	Description
	SetBounds (see page 6091)	Sets the properties for the left, top, right, and bottom margins all at once.
	SetControlBounds (see page 6091)	This is the overview for the SetControlBounds method overload.

32.1.24.3.1 TMargins.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign (see page 2930)

EConvertError (see page 22724)

32.1.24.3.2 TMargins.Change Method

Pascal

```
procedure Change; virtual;
```

C++

```
virtual __fastcall Change();
```

Description

This is Change, a member of class TMargins.

32.1.24.3.3 TMargins.GetControlBound Method

Pascal

```
function GetControlBound(Index: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int GetControlBound(int Index);
```

Description

This is GetControlBound, a member of class TMargins.

32.1.24.3.4 TMargins.InitDefaults Method

Pascal

```
class procedure InitDefaults(Margins: TMargins); virtual;
```

C++

```
virtual __fastcall InitDefaults(TMargins Margins);
```

Description

This is InitDefaults, a member of class TMargins.

32.1.24.3.5 TMargins.SetBounds Method

Sets the properties for the left, top, right, and bottom margins all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; ARight: Integer; ABottom: Integer);
```

C++

```
__fastcall SetBounds(int ALeft, int ATop, int ARight, int ABottom);
```

Description

Use the SetBounds method to specify the values for the Left (see page 6095), Top (see page 6095), Right (see page 6095), andBottom properties as the value of the ALeft, ATop, ARight, and ABottom parameters, respectively.

32.1.24.3.6 SetControlBounds Method

This is the overview for the SetControlBounds method overload.

Overload List

	Name	Description
	TMargins.SetControlBounds (Integer, Integer, Integer, Integer, Boolean) (see page 6092)	This is SetControlBounds, a member of class TMargins.
	TMargins.SetControlBounds (TRect, Boolean) (see page 6092)	This is SetControlBounds, a member of class TMargins.

32.1.24.3.6.1 TMargins.SetControlBounds Method (Integer, Integer, Integer, Integer, Boolean)

Pascal

```
procedure SetControlBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer;
Aligning: Boolean = False); overload;
```

C++

```
__fastcall SetControlBounds(int ALeft, int ATop, int AWidth, int AHeight, Boolean Aligning =
False);
```

Description

This is SetControlBounds, a member of class TMargins.

32.1.24.3.6.2 TMargins.SetControlBounds Method (TRect, Boolean)

Pascal

```
procedure SetControlBounds(const ARect: TRect; Aligning: Boolean = False); overload;
```

C++

```
__fastcall SetControlBounds(const TRect ARect, Boolean Aligning = False);
```

Description

This is SetControlBounds, a member of class TMargins.

32.1.24.4 TMargins Properties

The properties of the TMargins class are listed here.

Protected Properties

	Name	Description
R	Control (see page 6093)	This is Control, a member of class TMargins.

Public Properties

	Name	Description
R	ControlHeight (see page 6093)	Sets the height of the control.
R	ControlLeft (see page 6093)	Sets the starting point for the left side of the control.
R	ControlTop (see page 6094)	Sets the starting point for the top of the control.
R	ControlWidth (see page 6094)	Sets the width of the control.
R	ExplicitHeight (see page 6094)	This is ExplicitHeight, a member of class TMargins.
R	ExplicitLeft (see page 6094)	This is ExplicitLeft, a member of class TMargins.
R	ExplicitTop (see page 6095)	This is ExplicitTop, a member of class TMargins.
R	ExplicitWidth (see page 6095)	This is ExplicitWidth, a member of class TMargins.

Published Properties

	Name	Description
	Bottom (see page 6093)	Specifies the bottom edge of the margin.
	Left (see page 6095)	Specifies the left edge of the margin.
	Right (see page 6095)	Specifies the right edge of the margin.
	Top (see page 6095)	Specifies the top edge of the margin.

32.1.24.4.1 TMargins.Bottom Property

Specifies the bottom edge of the margin.

Pascal

```
property Bottom: TMarginSize;
```

C++

```
__property TMarginSize Bottom;
```

Description

Use the Bottom property to specify the bottom edge of the margin.

32.1.24.4.2 TMargins.Control Property

Pascal

```
property Control: TControl;
```

C++

```
__property TControl * Control;
```

Description

This is Control, a member of class TMargins.

32.1.24.4.3 TMargins.ControlHeight Property

Sets the height of the control.

Pascal

```
property ControlHeight: Integer;
```

C++

```
__property int ControlHeight;
```

Description

Use the ControlHeight property to set the height of the control. This height includes the height of the margins.

32.1.24.4.4 TMargins.ControlLeft Property

Sets the starting point for the left side of the control.

Pascal

```
property ControlLeft: Integer;
```

C++

```
__property int ControlLeft;
```

Description

Use the ControlLeft property to set the starting point of the left side of the control. This value includes the margins.

32.1.24.4.5 TMargins.ControlTop Property

Sets the starting point for the top of the control.

Pascal

```
property ControlTop: Integer;
```

C++

```
__property int ControlTop;
```

Description

Use the ControlTop property to set the starting point of the top of the control. This value includes the margins.

32.1.24.4.6 TMargins.ControlWidth Property

Sets the width of the control.

Pascal

```
property ControlWidth: Integer;
```

C++

```
__property int ControlWidth;
```

Description

Use the ControlWidth property to set the width of the control. This width includes the width of the margins.

32.1.24.4.7 TMargins.ExplicitHeight Property

Pascal

```
property ExplicitHeight: Integer;
```

C++

```
__property int ExplicitHeight;
```

Description

This is ExplicitHeight, a member of class TMargins.

32.1.24.4.8 TMargins.ExplicitLeft Property

Pascal

```
property ExplicitLeft: Integer;
```

C++

```
__property int ExplicitLeft;
```

Description

This is ExplicitLeft, a member of class TMargins.

32.1.24.4.9 TMargins.ExplicitTop Property

Pascal

```
property ExplicitTop: Integer;
```

C++

```
__property int ExplicitTop;
```

Description

This is ExplicitTop, a member of class TMargins.

32.1.24.4.10 TMargins.ExplicitWidth Property

Pascal

```
property ExplicitWidth: Integer;
```

C++

```
__property int ExplicitWidth;
```

Description

This is ExplicitWidth, a member of class TMargins.

32.1.24.4.11 TMargins.Left Property

Specifies the left edge of the margin.

Pascal

```
property Left: TMarginSize;
```

C++

```
__property TMarginSize Left;
```

Description

Use the Left property to specify the left edge of the margin.

32.1.24.4.12 TMargins.Right Property

Specifies the right edge of the margin.

Pascal

```
property Right: TMarginSize;
```

C++

```
__property TMarginSize Right;
```

Description

Use the Right property to specify the right edge of the margin.

32.1.24.4.13 TMargins.Top Property

Specifies the top edge of the margin.

Pascal

```
property Top: TMarginSize;
```

C++

```
__property TMarginSize Top;
```

Description

Use the Top property to specify the top edge of the margin.

32.1.24.5 TMargins Events

The events of the TMargins class are listed here.

Public Events

	Name	Description
	OnChange (see page 6096)	Occurs when margins change.

32.1.24.5.1 TMargins.OnChange Event

Occurs when margins change.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Use OnChange to write a handler to perform an action then the margin changes.

32.1.25 TMouse Class

TMouse surfaces information about the mouse and specifies how the application responds to mouse messages.

Class Hierarchy

Controls.TMouse

Pascal

```
TMouse = class;
```

C++

```
class TMouse;
```

File

Controls

Description

Use the global Mouse (see page 6321) variable, which is a TMouse object, to get information about the mouse.

TControl (see page 5724) descendants surface several events that let you respond to actions users make with the mouse. Typically, the event handlers for these events provide most of the information about the mouse that you need and you do not

need to use the global Mouse (see page 6321) variable. Instead, use the Mouse (see page 6321) variable to obtain similar information outside of a mouse-related event or to specify more global aspects of how your application responds to mouse messages, such as how much the user must move the mouse to start a drag operation.

See Also

Mouse (see page 6321)

TControl (see page 5724)

TControl (see page 5724)

TControl (see page 5724)

TControl (see page 5724)

TControl (see page 5724)

TControl (see page 5724)

32.1.25.1 TMouse Members

The following tables list the members exposed by TMouse.

Public Methods

	Name	Description
	Create (see page 6097)	Creates and initializes a TMouse (see page 6096) object.
	CreatePanningWindow (see page 6098)	Creates a panning window.
V	Destroy (see page 6098)	Destroys an instance of TMouse (see page 6096).
	SettingChanged (see page 6098)	Responds when a Mouse (see page 6321) setting is changed.

Public Properties

	Name	Description
	Capture (see page 6099)	Specifies the handle of the window that is capturing mouse events.
	CursorPos (see page 6100)	Specifies the position of the mouse cursor.
	DragImmediate (see page 6100)	Specifies whether clicking the left mouse button starts a drag event immediately.
	DragThreshold (see page 6101)	Specifies the number of pixels that the left mouse button must be moved in order to cause a drag operation.
R	IsDragging (see page 6101)	Is there an active dragging operation?
R	IsPanning (see page 6101)	Returns the Boolean value True if the panning window is actively panning. .
R	MousePresent (see page 6101)	Specifies whether a mouse is present on the system.
	PanningWindow (see page 6102)	Returns a reference to the active panning window. .
	PanningWindowClass (see page 6102)	The class type used for creating the panning window class.
R	RegWheelMessage (see page 6102)	Indicates the windows message that is sent when the user scrolls the mouse wheel.
R	WheelPresent (see page 6103)	Specifies whether the mouse has a wheel.
R	WheelScrollLines (see page 6103)	Specifies the number of lines that are scrolled for each notch that the mouse wheel is rotated.

32.1.25.2 TMouse.Create Constructor

Creates and initializes a TMouse (see page 6096) object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TMouse();
```

Description

Do not create instances of TMouse (see page 6096). A single instance of TMouse (see page 6096) is available as the global Mouse (see page 6321) variable.

32.1.25.3 TMouse.Destroy Destructor

Destroys an instance of TMouse (see page 6096).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMouse();
```

Description

Do not destroy the global instance of TMouse (see page 6096). This object is automatically destroyed when the application shuts down.

32.1.25.4 TMouse Methods

The methods of the TMouse class are listed here.

Public Methods

	Name	Description
	CreatePanningWindow (see page 6098)	Creates a panning window.
	SettingChanged (see page 6098)	Responds when a Mouse (see page 6321) setting is changed.

32.1.25.4.1 TMouse.CreatePanningWindow Method

Creates a panning window.

Pascal

```
function CreatePanningWindow: TCustomPanningWindow;
```

C++

```
__fastcall TCustomPanningWindow CreatePanningWindow();
```

Description

Creates a panning window when a panning operation is requested on a control that supports the csPanning ControlStyle. The window created is of the TCustomPanningWindow (see page 5958) class.

32.1.25.4.2 TMouse.SettingChanged Method

Responds when a Mouse (see page 6321) setting is changed.

Pascal

```
procedure SettingChanged(Setting: Integer);
```

C++

```
__fastcall SettingChanged(int Setting);
```

Description

SettingChanged is called automatically when the application receives notification from Windows that the control panel settings have changed. It updates the Mouse (see page 6321) object to reflect the current control panel settings.

Applications should not need to call this method.

32.1.25.5 TMouse Properties

The properties of the TMouse class are listed here.

Public Properties

	Name	Description
	Capture (see page 6099)	Specifies the handle of the window that is capturing mouse events.
	CursorPos (see page 6100)	Specifies the position of the mouse cursor.
	DragImmediate (see page 6100)	Specifies whether clicking the left mouse button starts a drag event immediately.
	DragThreshold (see page 6101)	Specifies the number of pixels that the left mouse button must be moved in order to cause a drag operation.
R	IsDragging (see page 6101)	Is there an active dragging operation?
R	IsPanning (see page 6101)	Returns the Boolean value True if the panning window is actively panning. .
R	MousePresent (see page 6101)	Specifies whether a mouse is present on the system.
	PanningWindow (see page 6102)	Returns a reference to the active panning window. .
	PanningWindowClass (see page 6102)	The class type used for creating the panning window class.
R	RegWheelMessage (see page 6102)	Indicates the windows message that is sent when the user scrolls the mouse wheel.
R	WheelPresent (see page 6103)	Specifies whether the mouse has a wheel.
R	WheelScrollLines (see page 6103)	Specifies the number of lines that are scrolled for each notch that the mouse wheel is rotated.

32.1.25.5.1 TMouse.Capture Property

Specifies the handle of the window that is capturing mouse events.

Pascal

```
property Capture: HWND;
```

C++

```
__property HWND Capture;
```

Description

Read Capture to get the handle of the window that receives all mouse messages. This is the window that is currently responding to the user's mouse actions.

Set Capture to the Handle property of a control to let it "capture" the mouse. When a window captures the mouse, it receives all mouse messages until it releases the mouse.

Typically, applications do not need to explicitly set the Capture property. Individual controls automatically capture the mouse in response to mouse clicks. They then translate incoming mouse message into events such as OnMouseDown or OnMouseMove. You only need to set Capture to override this default behavior. For example, when writing an object that creates a child control in response to a mouse click, you might set Capture to the child control's Handle property so that it receives subsequent mouse messages until the mouse button is released.

See Also

MouseCapture (see page 5831)

Handle (see page 6207)

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

OnMouseWheel (see page 5869)

OnMouseWheelDown (see page 5869)

OnMouseWheelUp (see page 5870)

32.1.25.5.2 TMouse.CursorPos Property

Specifies the position of the mouse cursor.

Pascal

```
property CursorPos: TPoint;
```

C++

```
__property TPoint CursorPos;
```

Description

Use CursorPos to get the position of the mouse cursor in global coordinates. Use this position to position UI elements in relation to the mouse. For example, the position of popup menus or the display of the drag rectangle during drag-and-dock operations is based on the value of CursorPos.

TMouse (see page 6096). CursorPos can raise an exception. The property getter API call is wrapped with Win32Check, which raises the exception, EOSException, if the function fails. The API call is shown in the following code:

```
function TMouse.GetCursorPos: TPoint;
begin
  Win32Check(Windows.GetCursorPos(Result));
end;
```

32.1.25.5.3 TMouse.DragImmediate Property

Specifies whether clicking the left mouse button starts a drag event immediately.

Pascal

```
property DragImmediate: Boolean;
```

C++

```
__property Boolean DragImmediate;
```

Description

Use DragImmediate to determine whether clicking the left mouse button starts a drag event immediately. If DragImmediate is false, then drag events are delayed until the mouse cursor is moved the number of pixels specified in the DragThreshold (see page 6101) property.

See Also

DragThreshold (see page 6101)

32.1.25.5.4 TMouse.DragThreshold Property

Specifies the number of pixels that the left mouse button must be moved in order to cause a drag operation.

Pascal

```
property DragThreshold: Integer;
```

C++

```
__property int DragThreshold;
```

Description

Use DragThreshold to specify the number of pixels that the mouse cursor must be moved with the left mouse button down in order to initiate a drag operation. DragThreshold only has an effect if the DragImmediate (see page 6100) property is false.

Note: DragThreshold must be greater than or equal to zero. Negative values are treated as 0.

See Also

DragImmediate (see page 6100)

32.1.25.5.5 TMouse.IsDragging Property

Is there an active dragging operation?

Pascal

```
property IsDragging: Boolean;
```

C++

```
__property Boolean IsDragging;
```

Description

Use IsDragging to determine whether a dragging operation is in progress.

32.1.25.5.6 TMouse.IsPanning Property

Returns the Boolean value True if the panning window is actively panning. .

Pascal

```
property IsPanning: Boolean;
```

C++

```
__property Boolean IsPanning;
```

Description

If a False value is returned, no panning is currently occurring.

32.1.25.5.7 TMouse.MousePresent Property

Specifies whether a mouse is present on the system.

Pascal

```
property MousePresent: Boolean;
```

C++

```
__property Boolean MousePresent;
```

Description

Use MousePresent to determine whether a mouse is present on the system.

See Also

WheelPresent (see page 6103)

32.1.25.5.8 TMouse.PanningWindow Property

Returns a reference to the active panning window. .

Pascal

```
property PanningWindow: TCustomPanningWindow;
```

C++

```
__property TCustomPanningWindow PanningWindow;
```

Description

Add this property to your code to return a reference to the active panning window.

32.1.25.5.9 TMouse.PanningWindowClass Property

The class type used for creating the panning window class.

Pascal

```
property PanningWindowClass: TPanningWindowClass;
```

C++

```
__property TPanningWindowClass PanningWindowClass;
```

Description

This class type is used for creating the panning window class.

32.1.25.5.10 TMouse.RegWheelMessage Property

Indicates the windows message that is sent when the user scrolls the mouse wheel.

Pascal

```
property RegWheelMessage: UINT;
```

C++

```
__property UINT RegWheelMessage;
```

Description

Use RegWheelMessage to get the windows message that is sent when the user scrolls the mouse wheel. Typically, applications do not need to use RegWheelMessage. TWinControl (see page 6112) checks for this value when it receives window messages, and translates it into appropriate Mouse (see page 6321) Wheel events.

RegWheelMessage is used in Windows 95, but not NT or later versions of Windows. (Mouse (see page 6321) support is built into the later operating systems.)

See Also

WheelPresent (see page 6103)

32.1.25.5.11 TMouse.WheelPresent Property

Specifies whether the mouse has a wheel.

Pascal

```
property WheelPresent: Boolean;
```

C++

```
__property Boolean WheelPresent;
```

Description

Use WheelPresent to determine whether the mouse has a wheel.

See Also

MousePresent (see page 6101)

RegWheelMessage (see page 6102)

32.1.25.5.12 TMouse.WheelScrollLines Property

Specifies the number of lines that are scrolled for each notch that the mouse wheel is rotated.

Pascal

```
property WheelScrollLines: Integer;
```

C++

```
__property int WheelScrollLines;
```

Description

Use WheelScrollLines to get the number of lines that are scrolled for each notch that the mouse wheel is rotated.

WheelScrollLines can be set under Mouse (see page 6321) in the Windows Control Panel.

See Also

WheelPresent (see page 6103)

32.1.26 TPadding Class

Contains the padding for the control.

Class Hierarchy

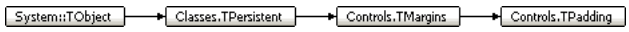

Pascal

```
TPadding = class(TMargins);
```

C++

```
class TPadding : public TMargins;
```

File

Controls

Description

TPadding is used in the Padding property of TControl (see page 5724) and its descendants. It specifies the padding of a control.

32.1.26.1 TPadding Members

The following tables list the members exposed by TPadding.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 6090)	Copies the properties of an object to a destination object.
	Change (see page 6091)	This is Change, a member of class TMargins.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetControlBound (see page 6091)	This is GetControlBound, a member of class TMargins.
	GetOwner (see page 2932)	Returns the owner of an object.
	InitDefaults (see page 6105)	This is InitDefaults, a member of class TPadding.

Protected Properties

	Name	Description
	Control (see page 6093)	This is Control, a member of class TMargins.

Public Events

	Name	Description
	OnChange (see page 6096)	Occurs when margins change.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6089)	This is Create, a member of class TMargins.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetBounds (see page 6091)	Sets the properties for the left, top, right, and bottom margins all at once.
	SetControlBounds (see page 6091)	This is the overview for the SetControlBounds method overload.

Public Properties

	Name	Description
R	ControlHeight (see page 6093)	Sets the height of the control.
R	ControlLeft (see page 6093)	Sets the starting point for the left side of the control.
R	ControlTop (see page 6094)	Sets the starting point for the top of the control.
R	ControlWidth (see page 6094)	Sets the width of the control.
R	ExplicitHeight (see page 6094)	This is ExplicitHeight, a member of class TMargins.
R	ExplicitLeft (see page 6094)	This is ExplicitLeft, a member of class TMargins.
R	ExplicitTop (see page 6095)	This is ExplicitTop, a member of class TMargins.
R	ExplicitWidth (see page 6095)	This is ExplicitWidth, a member of class TMargins.

Published Properties

	Name	Description
	Bottom (see page 6106)	Specifies the bottom edge of the padding.
	Left (see page 6106)	Specifies the left edge of the padding.
	Right (see page 6106)	Specifies the right edge of the padding.
	Top (see page 6107)	Specifies the top edge of the padding.

32.1.26.2 TPadding Methods

The methods of the TPadding class are listed here.

Protected Methods

	Name	Description
V	InitDefaults (see page 6105)	This is InitDefaults, a member of class TPadding.

32.1.26.2.1 TPadding.InitDefaults Method

Pascal

```
class procedure InitDefaults(Margins: TMargins); override;
```

C++

```
virtual __fastcall InitDefaults(TMargins Margins);
```

Description

This is InitDefaults, a member of class TPadding.

32.1.26.3 TPadding Properties

The properties of the TPadding class are listed here.

Published Properties

	Name	Description
	Bottom (see page 6106)	Specifies the bottom edge of the padding.
	Left (see page 6106)	Specifies the left edge of the padding.
	Right (see page 6106)	Specifies the right edge of the padding.
	Top (see page 6107)	Specifies the top edge of the padding.

32.1.26.3.1 TPadding.Bottom Property

Specifies the bottom edge of the padding.

Pascal

```
property Bottom: TMarginSize;
```

C++

```
__property TMarginSize Bottom;
```

Description

Use the Bottom property to specify the bottom edge of the padding.

32.1.26.3.2 TPadding.Left Property

Specifies the left edge of the padding.

Pascal

```
property Left: TMarginSize;
```

C++

```
__property TMarginSize Left;
```

Description

Use the Left property to specify the left edge of the padding.

32.1.26.3.3 TPadding.Right Property

Specifies the right edge of the padding.

Pascal

```
property Right: TMarginSize;
```

C++

```
__property TMarginSize Right;
```

Description

Use the Right property to specify the right edge of the padding.

32.1.26.3.4 TPadding.Top Property

Specifies the top edge of the padding.

Pascal

```
property Top: TMarginSize;
```

C++

```
__property TMarginSize Top;
```

Description

Use the Top property to specify the top edge of the padding.

32.1.27 TSizeConstraints Class

Contains size constraints for a control.

Class Hierarchy

Pascal

```
TSizeConstraints = class(TPersistent);
```

C++

```
class TSizeConstraints : public TPersistent;
```

File

Controls

Description

TSizeConstraints is used in the Constraints property of TControl (see page 5724) and its descendants. It specifies the minimum and maximum sizes of a control, preventing the control from being resized larger than the maximum or smaller than the minimum.

See Also

TControl (see page 5724)

32.1.27.1 TSizeConstraints Members

The following tables list the members exposed by TSizeConstraints.

Protected Methods

	Name	Description
	AssignTo (see page 6109)	Handles assignments for TSizeConstraint objects.
	Change (see page 6109)	Called when the size constraints change.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Protected Properties

	Name	Description
R	Control (see page 6110)	Specifies the control to which the size constraints apply.

Public Events

	Name	Description
	OnChange (see page 6112)	Occurs when the size constraints change.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 6109)	Creates and initializes an instance of TSizeConstraints (see page 6107).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Published Properties

	Name	Description
	MaxHeight (see page 6110)	Specifies the maximum height for the control.
	MaxWidth (see page 6111)	Specifies the maximum width for the control.
	MinHeight (see page 6111)	Specifies the minimum height for the control.
	MinWidth (see page 6111)	Specifies the minimum width for the control.

32.1.27.2 TSizeConstraints.Create Constructor

Creates and initializes an instance of TSizeConstraints (see page 6107).

Pascal

```
constructor Create(Control: TControl); virtual;
```

C++

```
virtual __fastcall TSizeConstraints(TControl * Control);
```

Description

Do not create instances of TSizeContraints. This object is created automatically by controls to implement their Constraints property.

Create sets the Control (see page 6110) property to Control (see page 6110) and then calls the inherited constructor.

See Also

Control (see page 6110)

Constraints (see page 5812)

32.1.27.3 TSizeConstraints Methods

The methods of the TSizeConstraints class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 6109)	Handles assignments for TSizeConstraint objects.
	Change (see page 6109)	Called when the size constraints change.

32.1.27.3.1 TSizeConstraints.AssignTo Method

Handles assignments for TSizeConstraint objects.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Call AssignTo to copy the values of one TSizeConstraints (see page 6107) object to another.

AssignTo overrides the inherited method in order to handle assignments specific to TSizeConstraints (see page 6107) objects.

See Also

Assign (see page 2930)

32.1.27.3.2 TSizeConstraints.Change Method

Called when the size constraints change.

Pascal

```
procedure Change; virtual;
```

C++

```
virtual __fastcall Change();
```

Description

Override Change in a derived class to perform actions when the size constraints change.

By default, Change calls the OnChange (see page 6112) event handler.

32.1.27.4 TSizeConstraints Properties

The properties of the TSizeConstraints class are listed here.

Protected Properties

	Name	Description
R	Control (see page 6110)	Specifies the control to which the size constraints apply.

Published Properties

	Name	Description
	MaxHeight (see page 6110)	Specifies the maximum height for the control.
	MaxWidth (see page 6111)	Specifies the maximum width for the control.
	MinHeight (see page 6111)	Specifies the minimum height for the control.
	MinWidth (see page 6111)	Specifies the minimum width for the control.

32.1.27.4.1 TSizeConstraints.Control Property

Specifies the control to which the size constraints apply.

Pascal

```
property Control: TControl;
```

C++

```
__property TControl * Control;
```

Description

Use Control to determine the control to which the size constraints apply.

Control is set automatically when a TSizeConstraints (see page 6107) object is created for the Constraints property of a control.

See Also

Constraints (see page 5812)

32.1.27.4.2 TSizeConstraints.MaxHeight Property

Specifies the maximum height for the control.

Pascal

```
property MaxHeight: TConstraintSize;
```

C++

```
__property TConstraintSize MaxHeight;
```

Description

Use MaxHeight to get or set the maximum height for the control.

32.1.27.4.3 TSizeConstraints.MaxWidth Property

Specifies the maximum width for the control.

Pascal

```
property MaxWidth: TConstraintSize;
```

C++

```
__property TConstraintSize MaxWidth;
```

Description

Use MaxWidth to get or set the maximum width for the control.

32.1.27.4.4 TSizeConstraints.MinHeight Property

Specifies the minimum height for the control.

Pascal

```
property MinHeight: TConstraintSize;
```

C++

```
__property TConstraintSize MinHeight;
```

Description

Use MinHeight to get or set the minimum height for the control.

32.1.27.4.5 TSizeConstraints.MinWidth Property

Specifies the minimum width for the control.

Pascal

```
property MinWidth: TConstraintSize;
```

C++

```
__property TConstraintSize MinWidth;
```

Description

Use MinWidth to get or set the minimum width for the control.

32.1.27.5 TSizeConstraints Events

The events of the TSizeConstraints class are listed here.

Public Events

	Name	Description
	OnChange (see page 6112)	Occurs when the size constraints change.

32.1.27.5.1 TSizeConstraints.OnChange Event

Occurs when the size constraints change.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Use OnChange to perform actions when the size constraints change.

32.1.28 TWinControl Class

TWinControl is the base class for all controls that are wrappers for Microsoft Windows screen objects.

Class Hierarchy

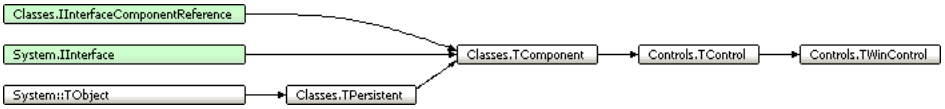

Pascal

```
TWinControl = class(TControl);
```

C++

```
class TWinControl : public TControl;
```

File

Controls

Description

TWinControl provides the common functionality for all controls that act as wrappers for Microsoft Windows screen objects ("windows"). Controls (see page 6197) that are wrap underlying windows have the following features:

The control can incorporate the functionality of an underlying window. For example, if the underlying screen object is a text editor, the control can incorporate the editor ability to manage and display a text buffer.

The control can receive user input focus. The focused control can handle keyboard input events. Some controls change their appearance when they have the focus. For example, button controls typically indicate the focus by drawing a rectangle around the caption.

The control can serve as a container for other controls, referred to as child controls. This relationship is signified by the child's Parent (see page 5832) property. Container controls provide important services to their children, including display services for controls that do not implement their own canvases. Examples of container controls include forms, panels, and toolbars.

Controls (see page 6197) based on TWinControl can display standard screen objects provided by Microsoft Windows, or customized screen objects developed by the VCL programmer.

Descendants of TWinControl include abstract base classes that support most kinds of user interface objects. The most significant descendant is TCustomControl (see page 5908), which provides code to implement a canvas and handle paint messages. Other important abstract descendants include TScrollingWinControl, TButtonControl, TCustomComboBox,

TCustomEdit, and TCustomListBox. When defining new control classes, consider these descendants before deriving directly from TWinControl.

Every TWinControl object has a Handle (see page 6207) property which provides the window handle for the underlying Microsoft Windows screen object. Use the Handle (see page 6207) property to bypass the VCL API and directly access the underlying window.

See Also

TButtonControl (see page 20981)

TCustomComboBox (see page 21201)

TCustomControl (see page 5908)

TCustomEdit (see page 21232)

TCustomListBox (see page 21308)

Accepting Dragged Items

Changing the Drag Mouse Pointer

Controlling How Child Controls Are Docked

Controlling How Child Controls Are Undocked

Controlling How Child Controls Respond to Drag-and-dock Operations

Customizing Drag and Drop with a Drag Object

Implementing Drag and Drop in Controls

Dropping Items

Ending a Drag Operation

Implementing Drag and Dock in Controls

Making a Control a Dockable Child

Making a Windowed Control a Docking Site

Starting a Drag Operation

Help and Hint Properties

32.1.28.1 TWinControl Members

The following tables list the members exposed by TWinControl.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.

	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.

	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.

	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.

	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.

R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

32.1.28.2 TWinControl.Create Constructor

Creates an instance of TWinControl (see page 6112).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWinControl(TComponent * AOwner);
```

Description

Call Create to construct and initialize a new control and insert the newly-constructed control into its owner, as specified by the AOwner parameter. Create inserts the control in the owner by calling the owner's InsertComponent method.

Most controls override Create to initialize their unique properties. Objects that override the Create method must always call the inherited Create method first, and then proceed with the component-specific initialization. Specify the override directive when overriding the Create method.

If a component's Create method allocates resources or memory, override the Destroy (see page 6124) method to free those resources.

See Also

Brush (see page 6195)

TabOrder (see page 6211)

CreateParented (see page 6123)

InsertComponent (see page 2780)

Create (see page 5731)

Owner (see page 2807)

32.1.28.3 TWinControl.CreateParented Constructor

Creates and initializes a control as the child of a specified non-VCL container.

Pascal

```
constructor CreateParented(ParentWindow: HWnd);
```

C++

```
__fastcall TWinControl(HWnd ParentWindow);
```

Description

Call CreateParented to embed a new control in a non-VCL parent window. TActiveXControl objects call CreateParented to create an ActiveX control as a child of the host application's client site window.

32.1.28.4 TWinControl.Destroy Destructor

Destroys an instance of TWinControl (see page 6112).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWinControl();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TWinControl (see page 6112) reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

Destroy (see page 5732)

32.1.28.5 TWinControl Fields

The fields of the TWinControl class are listed here.

Protected Fields

	Name	Description
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.

32.1.28.5.1 TWinControl.FDesignSize Field

Pascal

```
FDesignSize: TPoint;
```

C++

```
TPoint FDesignSize;
```

Description

This is FDesignSize, a member of class TWinControl.

32.1.28.5.2 TWinControl.FDoubleBuffered Field

Pascal

```
FDoubleBuffered: Boolean;
```

C++

```
Boolean FDoubleBuffered;
```

Description

This is FDoubleBuffered, a member of class TWinControl.

32.1.28.5.3 TWinControl.FInImeComposition Field

Pascal

```
FInImeComposition: Boolean;
```

C++

```
Boolean FInImeComposition;
```

Description

This is FInImeComposition, a member of class TWinControl.

32.1.28.6 TWinControl Methods

The methods of the TWinControl class are listed here.

Protected Methods

	Name	Description
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.

	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.

	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	WndProc (see page 6187)	Provides specific message responses for the control.

Public Methods

	Name	Description
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	Invalidate (see page 6161)	Schedules a control repaint.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	Realign (see page 6171)	Forces the control to realign children.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	Repaint (see page 6174)	Repaints the entire control.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScrollBy (see page 6176)	Scroll control contents.
	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 6178)	Makes a control visible at design time.
	SetFocus (see page 6179)	Gives the input focus to the control.
	Update (see page 6183)	Forces the control to update.
	UpdateControlState (see page 6186)	Respond to state change.

32.1.28.6.1 TWinControl.ActionChange Method

Updates the control to reflect changes in its associated action.

Pascal

```
procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); override;
```

C++

```
virtual __fastcall ActionChange(TObject * Sender, Boolean CheckDefaults);
```

Description

ActionChange is called automatically when the associated action changes. It updates the control to reflect the action's current Caption (see page 5806), Enabled (see page 5821), Hint (see page 5827), Visible (see page 5847), and HelpContext (see page 5826) properties and its OnClick (see page 5851) event handler.

The Sender parameter is the associated action. The CheckDefaults parameter specifies whether properties and event handlers

should be changed if they differ from the default values. When CheckDefaults is true, properties and event handlers are only changed when the current values are the default values (that is, when the values have not been previously changed). When CheckDefaults is false, properties and event handlers are changed, regardless of their current values.

See Also

RegisterChanges (see page 2713)

32.1.28.6.2 TWinControl.AddBiDiModeExStyle Method

Sets control's window style according to its bi-directional support.

Pascal

```
procedure AddBiDiModeExStyle(var ExStyle: DWORD);
```

C++

```
__fastcall AddBiDiModeExStyle(DWORD ExStyle);
```

Description

The CreateParams (see page 6135) method calls AddBiDiModeExStyle to update the control's style flags (passed as the ExStyle parameter) according the control's bi-directional support. AddBiDiModeExStyle uses the UseRightToLeftReading (see page 5790), UseRightToLeftScrollbar, UseRightToLeftAlignment (see page 5790), and GetControlsAlignment (see page 5759) methods to determine the current bi-directional support.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftScrollBar (see page 5791)

UseRightToLeftReading (see page 5790)

GetControlsAlignment (see page 5759)

CreateParams (see page 6135)

32.1.28.6.3 TWinControl.AdjustClientRect Method

Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.

Pascal

```
procedure AdjustClientRect(var Rect: TRect); virtual;
```

C++

```
virtual __fastcall AdjustClientRect(TRect Rect);
```

Description

AdjustClientRect is called internally when the control needs accurate information on where to place child controls within the client area. As implemented in TWinControl (see page 6112), AdjustClientRect does nothing. Override this method in descendants such as TTabControl where the ClientRect (see page 5810) property is not the same as the region in which child objects can appear.

See Also

ClientRect

Controls (see page 6197)

32.1.28.6.4 TWinControl.AdjustSize Method

Adjusts the control's size according to its contents and constraints.

Pascal

```
procedure AdjustSize; override;
```

C++

```
virtual __fastcall AdjustSize();
```

Description

If the AutoSize (see page 5803) property is true, AdjustSize is called automatically when the control needs to automatically resize itself to its contents. As implemented in TWinControl (see page 6112), AdjustSize calls SetBounds (see page 6178) with the values of its Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties. This call can result in a change of size if these values are altered in the control's OnCanResize (see page 5851) or OnConstrainedResize (see page 5853) event handlers, or if the control's OnResize (see page 5870) event handler makes any final adjustments.

Descendant classes override AdjustSize to change the size of the control to adjust to its contents.

Component writers may call AdjustSize when making changes to a control's contents.

See Also

OnCanResize (see page 5851)

OnConstrainedResize (see page 5853)

OnResize (see page 5870)

SetBounds (see page 6178)

AutoSize (see page 5803)

CanAutoSize (see page 6131)

32.1.28.6.5 TWinControl.AlignControls Method

Aligns any controls for which the control is the parent within a specified area of the control.

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); virtual;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

Use AlignControls to align all controls within an area of a form, panel, group box, scroll box, or any container control. Specify the area in which to align the controls as the value of the Rect parameter.

AlignControls uses the Align (see page 5798) property value for each child control to determine how to align it. The AControl parameter can be nil (Delphi) or NULL (C++). If you specify a control in AControl, that control takes precedence in alignment over other, similarly-aligned controls.

See Also

Align (see page 5798)

32.1.28.6.6 TWinControl.ArrangeControl Method

Pascal

```
procedure ArrangeControl(AControl: TControl; const ParentSize: TPoint; AAlign: TAlign;
AAlignInfo: TAlignInfo; var Rect: TRect; UpdateAnchorOrigin: Boolean = False);
```

C++

```
__fastcall ArrangeControl(TControl * AControl, const TPoint ParentSize, TAlign AAlign,
TAlignInfo AAlignInfo, TRect Rect, Boolean UpdateAnchorOrigin = False);
```

Description

This is ArrangeControl, a member of class TWinControl.

32.1.28.6.7 TWinControl.AssignTo Method

Copies the properties of the windowed control to another object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Do not call the protected AssignTo method. Instead, applications call the Assign method to copy the properties of another object. If the object's Assign method does not include the ability to copy from the source object, the Assign method calls the source object's protected AssignTo method, to copy the source objects properties to the object specified by the Dest parameter. This allows objects to expand the abilities of another object's Assign method to copy from additional sources.

The AssignTo method implemented in TWinControl (see page 6112) copies the Caption (see page 5806), Enabled (see page 5821), Hint (see page 5827), Visible (see page 5847), and HelpContext (see page 5826) properties and the OnClick (see page 5851) event handler to a TCustomAction object.

See Also

Assign (see page 2930)

TCustomAction (see page 297)

Caption (see page 5806)

Enabled (see page 5821)

Hint (see page 5827)

Visible (see page 5847)

HelpContext (see page 5826)

OnClick (see page 5851)

32.1.28.6.8 TWinControl.Broadcast Method

Sends a message to each of the child controls.

Pascal

```
procedure Broadcast(var Message);
```

C++

```
__fastcall Broadcast( Message);
```

Description

Use Broadcast when you want to send the same message to each of the child controls contained within the windowed control. Specify the message to be passed as the value of the Message parameter.

See Also

NotifyControls (see page 6165)

Broadcasting a Message to All Controls in a Form

32.1.28.6.9 TWinControl.CanAutoSize Method

Indicates how the control can be resized.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize (see page 5803) property is true and an attempt is made to resize the control. It allows controls to implement the AutoSize (see page 5803) property before the resize sequence that begins with the OnCanResize (see page 5851) and OnConstrainedResize (see page 5853) events and ends with the OnResize (see page 5870) event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the control. CanAutoSize adjusts these values so that the control's size adjusts to its contents. If the adjustment results in values of NewWidth and NewHeight that differ from the control's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width (see page 5849) and Height (see page 5825), CanAutoSize returns false, and the resize sequence is aborted.

As implemented in TWinControl (see page 6112), CanAutoSize returns true, adjusting NewWidth and NewHeight so that all the control's children fit.

See Also

Constraints (see page 5812)

ConstrainedResize (see page 6133)

AdjustSize (see page 6129)

32.1.28.6.10 TWinControl.CanFocus Method

Indicates whether a control can receive focus.

Pascal

```
function CanFocus: Boolean; dynamic;
```

C++

```
__fastcall Boolean CanFocus();
```

Description

Call CanFocus to find out if the control can receive input from the user.

CanFocus returns true if both the control and its parent(s) have their Visible (see page 5847) and Enabled (see page 5821) properties set to true. If any of the Visible (see page 5847) and Enabled (see page 5821) properties of the control or the controls in which it is contained are not true, then CanFocus returns false.

See Also

Enabled (see page 5821)

Parent (see page 5832)

Visible (see page 5847)

32.1.28.6.11 TWinControl.CanResize Method

Indicates how the control can respond to a resize attempt.

Pascal

```
function CanResize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanResize(int NewWidth, int NewHeight);
```

Description

CanResize is called automatically when an attempt is made to resize the control. The NewWidth and NewHeight parameters specify the attempted new values for the control's height and width. CanResize returns true if the control can be resized to the returned values of NewWidth and NewHeight. CanResize returns false if the control can't be resized.

CanResize generates an OnCanResize (see page 5851) event. An OnCanResize (see page 5851) event handler may change the values of NewHeight and NewWidth.

See Also

OnCanResize (see page 5851)

32.1.28.6.12 TWinControl.ChangeScale Method

Resize (see page 5780) and reposition children and self.

Pascal

```
procedure ChangeScale(M: Integer; D: Integer); override;
```

C++

```
virtual __fastcall ChangeScale(int M, int D);
```

Description

ChangeScale rescales the control in the same manner as the ChangeScale method in TControl (see page 5724). If the control contains any children, their ChangeScale methods are called before the parent itself is rescaled.

To rescale the control's children without rescaling the control itself, use ScaleControls (see page 6176). To rescale the control without repositioning, use ScaleBy (see page 6175).

See Also

ScaleBy (see page 6175)

ChangeScale (see page 5743)

32.1.28.6.13 TWinControl.ConstrainedResize Method

Generates an OnConstrainedResize (see page 5853) event.

Pascal

```
procedure ConstrainedResize(var MinWidth: Integer; var MinHeight: Integer; var MaxWidth:
Integer; var MaxHeight: Integer); override;
```

C++

```
virtual __fastcall ConstrainedResize(int MinWidth, int MinHeight, int MaxWidth, int MaxHeight);
```

Description

ConstrainedResize is called automatically as part of a control's resize sequence. It generates an OnConstrainedResize (see page 5853) event, passing in the MinWidth, MinHeight, MaxWidth, and MaxHeight parameters, which can be altered by the event handler before they are applied to the attempted resize.

See Also

OnConstrainedResize (see page 5853)

CanResize (see page 5742)

Constraints (see page 5812)

32.1.28.6.14 TWinControl.ContainsControl Method

Indicates whether a specified control exists within the control.

Pascal

```
function ContainsControl(Control: TControl): Boolean;
```

C++

```
__fastcall Boolean ContainsControl(TControl * Control);
```

Description

Use ContainsControl to find out if a particular control exists within this control. ContainsControl returns true if the specified value of the Control parameter is a child control of this control. The Control parameter need not be an immediate child (in the Controls (see page 6197) property), but may be contained in a child of the control, or in a child of a child of the control, for indefinitely many levels deep.

If the method returns false, the specified control is not within the control.

See Also

ControlAtPos (see page 6133)

Controls (see page 6197)

Parent (see page 5832)

32.1.28.6.15 TWinControl.ControlAtPos Method

Returns the child control located at a specified position within the control.

Pascal

```
function ControlAtPos(const Pos: TPoint; AllowDisabled: Boolean; AllowWinControls: Boolean =
False; AllLevels: Boolean = False): TControl;
```

C++

```
__fastcall TControl * ControlAtPos(const TPoint Pos, Boolean AllowDisabled, Boolean
AllowWinControls = False, Boolean AllLevels = False);
```

Description

Use ControlAtPos to determine which child control is at the specified location within the control. ControlAtPos returns an immediate child of the control; that is, one of the entries of the Controls (see page 6197) property, that has this control for its Parent (see page 5832) property.

Specify the position in client coordinates as the value of the Pos parameter. Pos can be anywhere within the boundaries of the child control, not just the upper left corner.

The AllowDisabled parameter determines whether the search for controls includes disabled controls.

The AllowWinControls parameter determines whether descendants of TWinControl (see page 6112) are considered when looking for the child control.

If there is no control that matches the AllowDisabled and AllowWinControls parameters at the specified position, ControlAtPos returns nil (Delphi) or NULL (C++).

See Also

ContainsControl (see page 6133)

Controls (see page 6197)

Parent (see page 5832)

ScreenToClient (see page 5781)

FindVCLWindow (see page 6250)

32.1.28.6.16 TWinControl.ControlsAligned Method

Pascal

```
procedure ControlsAligned; dynamic;
```

C++

```
__fastcall ControlsAligned();
```

Description

This is ControlsAligned, a member of class TWinControl.

32.1.28.6.17 TWinControl.CreateDockManager Method

Creates the dock manager for the control.

Pascal

```
function CreateDockManager: IDockManager; dynamic;
```

C++

```
__fastcall IDockManager CreateDockManager();
```

Description

CreateDockManager is called automatically during drag-and-dock operations when the UseDockManager (see page 6212) property is true. CreateDockManager creates the object that implements the DockManager (see page 6202) property.

Override CreateDockManager to create a custom dock manager for the control.

See Also

DockSite (see page 6204)

UseDockManager (see page 6212)

DockManager (see page 6202)

32.1.28.6.18 TWinControl.CreateHandle Method

Creates underlying screen object.

Pascal

```
procedure CreateHandle; virtual;
```

C++

```
virtual __fastcall CreateHandle();
```

Description

CreateHandle creates the Microsoft Windows screen object that underlies the control. if it does not already exist. The new window handle becomes the value of the Handle (see page 6207) property.

CreateHandle creates or locates the parent control's window before it creates a window for the control itself.

Note: If the Parent (see page 5832) property is nil (Delphi) or NULL (C++), CreateHandle raises an exception.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

Handle (see page 6207)

Parent (see page 5832)

32.1.28.6.19 TWinControl.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); virtual;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 6137) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle (see page 6137). Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams (see page 6280) record become the

parameters to a call to the CreateWindowEx API function.

TWinControl (see page 6112) implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

32.1.28.6.20 TWinControl.CreateParentedControl Method

Creates and initializes a control as the child of a specified non-VCL window.

Pascal

```
class function CreateParentedControl(ParentWindow: HWnd): TWinControl;
```

C++

```
__fastcall TWinControl * CreateParentedControl(HWnd ParentWindow);
```

Description

Call CreateParentedControl to embed a new control in a non-VCL parent.

CreateParentedControl allocates memory for a new instance of the same class as the control (Delphi) or is the class specified by the vmt parameter (C++), sets its ParentWindow (see page 6210) property to ParentWindow (see page 6210), and calls the constructor, passing in nil (Delphi) or NULL (C++) for the Owner parameter. CreateParentedControl returns the newly created control.

CreateParentedControl has the same purpose as CreateParented (see page 6123), but is a simple class function instead of a constructor. Unlike CreateParented (see page 6123), CreateParentedControl can be called from C++ code.

See Also

CreateParented (see page 6123)

ParentWindow (see page 6210)

32.1.28.6.21 TWinControl.CreateSubClass Method

Creates a windowed control derived from an existing Windows window class.

Pascal

```
procedure CreateSubClass(var Params: TCreateParams; ControlClassName: PChar);
```

C++

```
__fastcall CreateSubClass(TCreateParams Params, const char * ControlClassName);
```

Description

Call CreateSubClass in the CreateParams (see page 6135) method of a subclassed control, after calling the inherited CreateParams (see page 6135). Specify the parameter record as the value of the Params parameter. CreateSubClass allows VCL controls to create registered Windows controls.

See Also

CreateParams (see page 6135)

CreateWindowHandle (see page 6137)

32.1.28.6.22 TWinControl.CreateWindowHandle Method

Creates a Windows control to represent the control.

Pascal

```
procedure CreateWindowHandle(const Params: TCreateParams); virtual;
```

C++

```
virtual __fastcall CreateWindowHandle(const TCreateParams Params);
```

Description

The CreateWnd (see page 6137) method calls CreateWindowHandle to create the window for a control. CreateWindowHandle creates the window by calling the CreateWindowEx API function, passing parameters from the record passed in the Params parameter. Once the window is created, its handle is available as the Handle (see page 6207) property.

See Also

CreateParams (see page 6135)

CreateWnd (see page 6137)

Handle (see page 6207)

32.1.28.6.23 TWinControl.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; virtual;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams (see page 6135)

CreateWindowHandle (see page 6137)

Perform (see page 5776)

DestroyWnd (see page 6141)

32.1.28.6.24 TWinControl.CustomAlignInsertBefore Method

Returns placement order of custom-aligned child control.

Pascal

```
function CustomAlignInsertBefore(C1: TControl; C2: TControl): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CustomAlignInsertBefore(TControl * C1, TControl * C2);
```

Description

CustomAlignInsertBefore determines the order in which custom-aligned child controls are placed. The control calls CustomAlignInsertBefore once for each pair of child controls with an Align (see page 5798) property of **alCustom**. CustomAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise.

Once the custom-aligned child controls are sorted, their specific positions are determined by calls to CustomAlignPosition (see page 6138).

CustomAlignInsertBefore triggers the OnAlignInsertBefore (see page 6213) event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore (see page 6213). Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

As implemented in TWinControl (see page 6112), CustomAlignInsertBefore returns false if OnAlignInsertBefore (see page 6213) is not implemented, so the controls are placed in the order they appear in the Controls (see page 6197) property.

See Also

CustomAlignPosition (see page 6138)

Align (see page 5798)

Controls (see page 6197)

OnAlignInsertBefore (see page 6213)

32.1.28.6.25 TWinControl.CustomAlignPosition Method

Returns aligned position of custom-aligned child control.

Pascal

```
procedure CustomAlignPosition(Control: TControl; var NewLeft: Integer; var NewTop: Integer;
var NewWidth: Integer; var NewHeight: Integer; var AlignRect: TRect; AlignInfo: TAlignInfo);
virtual;
```

C++

```
virtual __fastcall CustomAlignPosition(TControl * Control, int NewLeft, int NewTop, int
NewWidth, int NewHeight, TRect AlignRect, TAlignInfo AlignInfo);
```

Description

CustomAlignPosition determines the specific position of custom-aligned child controls. CustomAlignPosition is called once for each child control with an Align (see page 5798) property of **alCustom**, in an order previously determined by calls to CustomAlignInsertBefore (see page 6137).

CustomAlignPosition triggers the OnAlignPosition (see page 6214) event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition (see page 6214). Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition.

These are the parameters:

Control	The child control being placed.
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors (see page 5802) TControl_Anchors constraints for the control. CustomAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. CustomAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo (see page 6269).

As implemented in TWinControl (see page 6112), CustomAlignPosition does nothing.	

See Also

CustomAlignInsertBefore (see page 6137)

Align (see page 5798)

OnAlignPosition (see page 6214)

TAlignInfo (see page 6269)

32.1.28.6.26 TWinControl.DefaultHandler Method

Provides message handling for all messages that the control does not fully process itself.

Pascal

```
procedure DefaultHandler(var Message); override;
```

C++

```
virtual __fastcall DefaultHandler( Message);
```

Description

Override DefaultHandler to change the default message handling for the control. The Message parameter can be cast to a TMessage type, to obtain the WParam, LParam, and Result of the message. If the Result of the message is non-zero, the message has already been handled. Set the Result field to a non-zero value to prevent further processing of the message by the inherited method.

Note: In Delphi code, calling inherited in a message-handling method results in a call to the ancestor's DefaultHandler method unless that ancestor specifies an explicit handler for the message.

TWinControl (see page 6112) overrides the TControl (see page 5724) DefaultHandler method to handle messages for all its descendant types. DefaultHandler passes any otherwise-unhandled messages to the control's window procedure using the CallWindowProc API function.

See Also

WindowProc (see page 5872)

DefWndProc (see page 6199)

32.1.28.6.27 TWinControl.DefineProperties Method

Provides methods to read and write the IsControl property to a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

DefineProperties is called automatically by the streaming system that loads and saves VCL components. It allows persistent objects to read and write properties that are not published. Published properties are loaded and stored automatically.

DefineProperties uses the methods of the filer passed as the Filer parameter to assign methods that can load and save the IsControl property.

Override DefineProperties to persist unpublished properties. Call the DefineBinaryProperty or DefineProperty method of the Filer parameter to assign methods that load or save property values.

Note: When overriding this method, call the parent class's DefineProperties method first.

See Also

DefineBinaryProperty (see page 2837)

DefineProperty (see page 2838)

DefineProperties (see page 2771)

IsControl (see page 5829)

32.1.28.6.28 TWinControl.DestroyHandle Method

Destroys the control's window without destroying the control.

Pascal

```
procedure DestroyHandle; virtual;
```

C++

```
virtual __fastcall DestroyHandle();
```

Description

Call DestroyHandle to dispose of the window, but leave the control intact. The control can later recreate the window if needed. DestroyHandle is the converse operation to CreateHandle (see page 6135). Applications should call the high-level CreateHandle (see page 6135) and DestroyHandle methods, rather than the lower-level methods of CreateWnd (see page 6137) and DestroyWnd (see page 6141), whenever possible.

If the control has TWinControl (see page 6112) objects as child controls, DestroyHandle calls each of their DestroyHandle methods before calling DestroyWnd (see page 6141) to destroy its own handle.

See Also

CreateHandle (see page 6135)

DestroyWnd (see page 6141)

32.1.28.6.29 TWinControl.DestroyWindowHandle Method

Destroys the window created in the CreateWindowHandle (see page 6137) method.

Pascal

```
procedure DestroyWindowHandle; virtual;
```

C++

```
virtual __fastcall DestroyWindowHandle();
```

Description

Call DestroyWindowHandle to dispose of the window for the control.

TWinControl (see page 6112)'s Destroy (see page 6124) method calls DestroyWindowHandle to destroy any window associated with a windowed control before destroying the object. The DestroyWnd (see page 6141) method also calls DestroyWindowHandle.

See Also

CreateWindowHandle (see page 6137)

Destroy (see page 6124)

DestroyWnd (see page 6141)

32.1.28.6.30 TWinControl.DestroyWnd Method

Destroys the control's window.

Pascal

```
procedure DestroyWnd; virtual;
```

C++

```
virtual __fastcall DestroyWnd();
```

Description

DestroyWnd is called when the control needs to replace its window. For example, changing properties that are implemented by window parameter settings require the control to be destroyed and then recreated using the CreateWnd (see page 6137) method.

Before the window is destroyed, DestroyWnd saves a copy of the control's text in memory, frees any device contexts, and finally calls DestroyWindowHandle (see page 6140). When the window is recreated, the stored copy of the control's text is assigned to the new window.

See Also

DestroyWindowHandle (see page 6140)

RecreateWnd (see page 6171)

32.1.28.6.31 TWinControl.DisableAlign Method

Disables the realignment of child controls.

Pascal

```
procedure DisableAlign;
```

C++

```
__fastcall DisableAlign();
```

Description

Call DisableAlign to temporarily prevent child controls from realigning. For example, while performing multiple manipulations of controls, such as reading from a form file or scaling, performance improves if child controls are not realigned until all manipulations are complete. The EnableAlign (see page 6148) method restores normal child realignment.

Every call to DisableAlign must be followed by a matching call to EnableAlign (see page 6148). If an exception could be raised after the call to DisableAlign, use an exception block to ensure that EnableAlign (see page 6148) will always be called.

DisableAlign/EnableAlign (see page 6148) call sequences can be nested. An internal counter records the level of nesting. As long as the counter is positive, realignment is disabled and the AlignDisabled (see page 6192) property returns true. Once the counter returns to zero, realignment is enabled and AlignDisabled (see page 6192) returns false.

See Also

EnableAlign (see page 6148)

AlignDisabled (see page 6192)

32.1.28.6.32 TWinControl.DoAddDockClient Method

Performs actions when a control is docked to the windowed control.

Pascal

```
procedure DoAddDockClient(Client: TControl; const ARect: TRect); dynamic;
```

C++

```
__fastcall DoAddDockClient(TControl * Client, const TRect ARect);
```

Description

DoAddDockClient is called automatically when a client control is docked to the windowed control. As implemented in TWinControl (see page 6112), DoAddDockClient sets the client's Parent (see page 5832) property to the control.

Client is the control that has been docked.

ARect describes the boundaries of the region where Client has been docked.

Override DoAddDockClient to perform actions when a control is docked to the windowed control.

Note: DoAddDockClient is called in response to a message sent by the DockDrop (see page 6142) method, before the OnDockDrop (see page 6214) event.

See Also

Parent (see page 5832)

DockClients (see page 6201)

ManualDock (see page 5771)

DoRemoveDockClient (see page 6147)

DockDrop (see page 6142)

OnDockDrop (see page 6214)

32.1.28.6.33 TWinControl.DockDrop Method

Generates an OnDockDrop (see page 6214) event.

Pascal

```
procedure DockDrop(Source: TDragDockObject; X: Integer; Y: Integer); dynamic;
```

C++

```
__fastcall DockDrop(TDragDockObject Source, int X, int Y);
```

Description

DockDrop is called automatically when a control is docked to the windowed control. It instructs the control that is being dropped to prepare for the dock operation and then generates an OnDockDrop (see page 6214) event.

Although you can override DockDrop to perform actions in addition to the generated OnDockDrop (see page 6214) event, typically descendant classes override the DoAddDockClient (see page 6142) method instead. This is because DoAddDockClient (see page 6142) is called by the client after it has performed its own preparations, but before the OnDockDrop (see page 6214) event.

DockDrop is called only if DockSite (see page 6204) is true.

See Also

OnDockDrop (see page 6214)

DockSite (see page 6204)

DoAddDockClient (see page 6142)

32.1.28.6.34 TWinControl.DockOver Method

Responds when a docking client is dragged over the windowed control.

Pascal

```
procedure DockOver(Source: TDragDockObject; X: Integer; Y: Integer; State: TDragState; var
Accept: Boolean); dynamic;
```

C++

```
__fastcall DockOver(TDragDockObject Source, int X, int Y, TDragState State, Boolean Accept);
```

Description

DockOver is called automatically for dock sites when a dockable object is dragged over them. As implemented in TWinControl (see page 6112), DockOver positions the docking rectangle and then calls DoDockOver (see page 6143) to generate an OnDockOver (see page 6216) event.

Override DockOver to perform additional actions when another control is dragged over the control.

DockOver is called only if DockSite (see page 6204) is true.

See Also

OnDockOver (see page 6216)

DockSite (see page 6204)

DoDockOver (see page 6143)

32.1.28.6.35 TWinControl.DockReplaceDockClient Method

Pascal

```
function DockReplaceDockClient(Client: TControl; NewDockSite: TWinControl; DropControl:
TControl; ControlSide: TAlign; ReplacementClient: TControl): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DockReplaceDockClient(TControl * Client, TWinControl * NewDockSite,
TControl * DropControl, TAlign ControlSide, TControl * ReplacementClient);
```

Description

This is DockReplaceDockClient, a member of class TWinControl.

32.1.28.6.36 TWinControl.DoDockOver Method

Generates an OnDockOver (see page 6216) event.

Pascal

```
procedure DoDockOver(Source: TDragDockObject; X: Integer; Y: Integer; State: TDragState; var
Accept: Boolean); dynamic;
```

C++

```
__fastcall DoDockOver(TDragDockObject Source, int X, int Y, TDragState State, Boolean Accept);
```

Description

DoDockOver is called by the DockOver (see page 6143) method to generate an OnDockOver (see page 6216) event when the user drags a dockable control over the windowed control. Override this method to perform some other action or to suppress the event.

See Also

OnDockOver (see page 6216)

DockOver (see page 6143)

32.1.28.6.37 TWinControl.DoEnter Method

Respond to receiving input focus.

Pascal

```
procedure DoEnter; dynamic;
```

C++

```
__fastcall DoEnter();
```

Description

DoEnter is called automatically when the control receives the input focus. As implemented in TWinControl (see page 6112), DoEnter calls the OnEnter (see page 6219) event handler, if defined.

Descendant classes that override DoEnter should always call the inherited method.

See Also

DoExit (see page 6144)

OnEnter (see page 6219)

32.1.28.6.38 TWinControl.DoExit Method

Respond to losing input focus

Pascal

```
procedure DoExit; dynamic;
```

C++

```
__fastcall DoExit();
```

Description

DoExit is called automatically when the control receives the input focus. As implemented in TWinControl (see page 6112), DoExit calls the OnExit (see page 6221) event handler, if defined.

Descendant classes that override DoExit should always call the inherited method.

See Also

DoEnter (see page 6144)

OnExit (see page 6221)

32.1.28.6.39 TWinControl.DoFlipChildren Method

Flips the control's children.

Pascal

```
procedure DoFlipChildren; dynamic;
```

C++

```
__fastcall DoFlipChildren();
```

Description

The FlipChildren (see page 6150) method calls DoFlipChildren to reverse the positions of child controls. DoFlipChildren moves the control's children to opposite positions on the control; that is, children on the left side of the control are moved to the right side of the control and vice versa.

Override FlipChildren (see page 6150) to control how the control's children are flipped.

Note: DoFlipChildren is a protected helper method for the public FlipChildren (see page 6150) method. Unlike FlipChildren (see page 6150), DoFlipChildren does not adjust the alignment of controls that are aligned to one of the edges of the windowed control.

See Also

BiDiMode (see page 5803)

FlipChildren (see page 6150)

32.1.28.6.40 TWinControl.DoKeyDown Method

Performs some preprocessing before generating an OnKeyDown (see page 6225) event.

Pascal

```
function DoKeyDown(var Message: TWMKey): Boolean;
```

C++

```
__fastcall Boolean DoKeyDown(TWMKey Message);
```

Description

The return value from DoKeyDown determines if the control should continue processing key-down messages passed in the Message parameter. A return value of true indicates that the key-down occurrence has been handled completely, and no further processing is needed. A return value of false indicates that the application should continue passing the key-down message through the inherited processing.

DoKeyDown checks whether the KeyPreview property of the parent form is true; if so, it allows the form to preemptively process key-down messages before they are handled by the windowed control. If the form does not handle the message, DoKeyDown translates the message parameters into the appropriate types and calls KeyDown (see page 6163), which in turn calls the OnKeyDown (see page 6225) event handler, if any.

See Also

DoKeyPress (see page 6146)

DoKeyUp (see page 6146)

KeyDown (see page 6163)

OnKeyDown (see page 6225)

KeyPreview (see page 13918)

32.1.28.6.41 TWinControl.DoKeyPress Method

Performs some preprocessing before generating an OnKeyPress (see page 6227) event.

Pascal

```
function DoKeyPress(var Message: TWMKey): Boolean;
```

C++

```
__fastcall Boolean DoKeyPress(TWMKey Message);
```

Description

The return value from DoKeyPress determines whether the control should continue processing the key-press message passed in the Message parameter. A return value of true indicates that the key-press occurrence has been handled completely, and no further processing is needed. A return value of false indicates that the application should continue passing the key-press message through the inherited processing.

DoKeyPress checks whether the KeyPreview property of the parent form is true, and if so, allows the form to preemptively process key-press messages before they are handled by the windowed control. If the form does not handle the message, DoKeyPress translates the message parameters into the appropriate types and calls KeyPress (see page 6163), which in turn calls the OnKeyPress (see page 6227) event handler, if any.

See Also

DoKeyDown (see page 6145)

DoKeyUp (see page 6146)

KeyPress (see page 6163)

OnKeyPress (see page 6227)

KeyPreview (see page 13918)

32.1.28.6.42 TWinControl.DoKeyUp Method

Performs some preprocessing before generating an OnKeyUp (see page 6228) event.

Pascal

```
function DoKeyUp(var Message: TWMKey): Boolean;
```

C++

```
__fastcall Boolean DoKeyUp(TWMKey Message);
```

Description

The return value from DoKeyUp determines whether the control should continue processing the key-up message passed in the Message parameter. A return value of true indicates that the key-up occurrence has been handled completely, and no further processing is needed. A return value of false indicates that the application should continue passing the key-up message through the inherited processing.

DoKeyUp checks whether the KeyPreview property of the parent form is true, and if so, allows the form to preemptively process key-up messages before they are handled by the windowed control. If the form does not handle the message, DoKeyUp translates the message parameters into the appropriate types and calls KeyUp (see page 6164), which in turn calls the OnKeyUp (see page 6228) event handler, if any.

See Also

DoKeyDown (see page 6145)

KeyUp (see page 6164)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

32.1.28.6.43 TWinControl.DoRemoveDockClient Method

Responds when a control is undocked from the windowed control.

Pascal

```
procedure DoRemoveDockClient(Client: TControl); dynamic;
```

C++

```
__fastcall DoRemoveDockClient(TControl * Client);
```

Description

DoRemoveDockClient is called automatically when a client control is undocked from the windowed control. As implemented in TWinControl (see page 6112), DoRemoveDockClient does nothing.

Client is the control that has been undocked.

Override DoRemoveDockClient to perform any necessary adjustments when a control is undocked from the windowed control.

Note: DoRemoveDockClient occurs at the end of the undocking process. To make changes when undocking starts, override the DoUnDock (see page 6147) method instead.

See Also

DockClients (see page 6201)

ManualDock (see page 5771)

DoAddDockClient (see page 6142)

DoUnDock (see page 6147)

32.1.28.6.44 TWinControl.DoUnDock Method

Undocks a control that is currently docked to the windowed control.

Pascal

```
function DoUnDock(NewTarget: TWinControl; Client: TControl): Boolean; dynamic;
```

C++

```
__fastcall Boolean DoUnDock(TWinControl * NewTarget, TControl * Client);
```

Description

DoUnDock is called automatically when a control is undocked from the windowed control. It handles all necessary changes to the windowed control to accomplish the undocking. These are

1.Generates an OnUnDock (see page 6230) event.

2.If there is no OnUnDock (see page 6230) event handler, or if the event handler indicates that the client can be undocked, it removes the client from the DockClients (see page 6201) property list.

The NewTarget parameter indicates the new host to which the undocked control is moving.

The Client parameter indicates the control to be undocked from the windowed control.

DoUnDock returns true if the client is successfully undocked, false if the undocking attempt was blocked by an OnUnDock (see page 6230) event handler.

Override DoUnDock to change the way controls are undocked from the windowed control.

Note: DoUnDock occurs at the start of the undocking process. To handle changes when undocking is complete, override the DoRemoveDockClient (see page 6147) method instead.

See Also

OnUnDock (see page 6230)

DockClients (see page 6201)

DoRemoveDockClient (see page 6147)

32.1.28.6.45 TWinControl.EnableAlign Method

Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.

Pascal

```
procedure EnableAlign;
```

C++

```
__fastcall EnableAlign();
```

Description

Call EnableAlign to allow child controls within the control to realign again after they were prevented from realigning by a call to DisableAlign (see page 6141).

Each time the DisableAlign (see page 6141) method is called, it increments a reference count. Each time EnableAlign is called, it decrements the same reference count. When the reference count reaches zero, EnableAlign calls the Realign (see page 6171) method to perform any pending realignments.

Be sure to pair each call to DisableAlign (see page 6141) with a call to EnableAlign. If an exception could be raised after the call to DisableAlign (see page 6141), use an exception block to ensure that the corresponding call to EnableAlign is executed.

See Also

DisableAlign (see page 6141)

32.1.28.6.46 TWinControl.FindChildControl Method

Returns a child control given its name.

Pascal

```
function FindChildControl(const ControlName: string): TControl;
```

C++

```
__fastcall TControl * FindChildControl(const AnsiString ControlName);
```

Description

Call FindChildControl to locate a specified child control. FindChildControl searches the control's children and returns the control with the specified name. If no child controls match the specified name, FindChildControl returns nil (Delphi) or NULL (C++).

Note: FindChildControl only locates immediate children of the control. It can't find a control that is a child of one of the control's children.

See Also

FindNextControl (see page 6149)

Controls (see page 6197)

FindVCLWindow (see page 6250)

FindControl (see page 6249)

32.1.28.6.47 TWinControl.FindNextControl Method

Returns the control's next child in the tab order after the specified control.

Pascal

```
function FindNextControl(CurControl: TWinControl; GoForward: Boolean; CheckTabStop: Boolean;
CheckParent: Boolean): TWinControl;
```

C++

```
__fastcall TWinControl * FindNextControl(TWinControl * CurControl, Boolean GoForward, Boolean
CheckTabStop, Boolean CheckParent);
```

Description

Call FindNextControl to find the next child control in the tab order after CurControl. If CurControl is not a child of the control, FindNextControl returns the first child control in the tab order.

The GoForward parameter controls the direction of the search. If GoForward is true, FindNextControl searches forward through the child controls in tab order. If Go Forward is false, FindNextControl searches backward through the controls.

The CheckTabStop and CheckParent parameters control whether FindNextControl performs certain checks on the controls it finds. If CheckTabStop is true, the returned control must have its TabStop (see page 6211) property set to true. If CheckParent is true, the returned control's Parent (see page 5832) property must indicate the parent control.

FindNextControl calls the GetTabOrderList (see page 6158) method to build its list of possible "next" controls.

See Also

GetTabOrderList (see page 6158)

TabOrder (see page 6211)

TabStop (see page 6211)

SelectNext (see page 6177)

32.1.28.6.48 TWinControl.FixupTabList Method

Sorts the child controls by their tab order.

Pascal

```
procedure FixupTabList;
```

C++

```
__fastcall FixupTabList();
```

Description

Applications should not call FixupTabList directly. The ReadState (see page 6171) method calls FixupTabList to initialize the

tab order list based on the TabOrder (see page 6211) properties of the child controls being read from a stream.

See Also

ReadState (see page 6171)

TabOrder (see page 6211)

32.1.28.6.49 TWinControl.FlipChildren Method

Reverses the positions of child controls.

Pascal

```
procedure FlipChildren(AllLevels: Boolean); dynamic;
```

C++

```
__fastcall FlipChildren(Boolean AllLevels);
```

Description

Call FlipChildren to flip the control's children; that is, to move children on the left side of the control to the right side and vice versa. FlipChildren reverses the position of all child controls and adjusts their Align (see page 5798) properties, if necessary, to enforce the new position.

AllLevels specifies whether FlipChildren should be called recursively on the control's children.

FlipChildren can be used to reverse the layout of an application when it is running in Middle Eastern locales where users read from right to left instead of left to right: Call UseRightToLeftAlignment (see page 5790) to determine whether the BiDiMode (see page 5803) property dictates a reversal of alignment based on the system locale.

Note: FlipChildren does the same thing as the Flip Children command on the control's context menu in the IDE.

See Also

BiDiMode (see page 5803)

UseRightToLeftAlignment (see page 5790)

32.1.28.6.50 TWinControl.Focused Method

Determines whether the control has input focus.

Pascal

```
function Focused: Boolean; dynamic;
```

C++

```
__fastcall Boolean Focused();
```

Description

Use the Focused method to see if the control is the active control. When Focused returns true, the control has the input focus. If Focused returns false, the user cannot interact with the control.

C++ Examples:

```
/*
This example uses the OnActiveControlChange event to detect
when focus changes on the form. When focus changes, the hint
for the active control is displayed on the status bar. To use
this example, you must add a status bar to the form and set
its SimplePanel property to true.  Add a new public procedure,
ActiveControlChanged, to the TForm1 class declaration.
```

```
*/
void _fastcall TForm1::ActiveControlChanged(System::TObject *Sender)
{
  TWinControl *Active = NULL;
  for (int I = 0; I < Form1->ControlCount; I++)
  {
    TWinControl *Temp = dynamic_cast<TWinControl *>(Form1->Controls[I]);
    if (Temp && Temp->Focused())
      Active = Temp;
  }
  if ((Active != NULL) && (Active->Hint != ""))
    StatusBar2->SimpleText = GetLongHint(Active->Hint) + " focus";
}

void __fastcall TForm1::MouseOverChanged(TObject *Sender)
{
  TWinControl *Active = dynamic_cast<TWinControl *>(Sender);
  if (Active != NULL)
    StatusBar1->SimpleText = GetLongHint(Active->Hint) + " mouse over";
}

/*
Assign this method as the OnActiveControlChange event
handler by setting it from the form's OnCreate event handler:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Screen->OnActiveControlChange = ActiveControlChanged;
}

/*
Make sure you clean up the screen object when the form is
freed by adding this OnDestroy event handler to the form:
*/
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  Screen->OnActiveControlChange = 0;
}
```

Delphi Examples:

```
{
This example uses the OnActiveControlChange event to detect
when focus changes on the form. When focus changes, the hint
for the active control is displayed on the status bar. To use
this example, you must add a status bar to the form and set
its SimplePanel property to true.  Add a new public procedure,
ActiveControlChanged, to the TForm1 class declaration.
}

type
  TForm1 = class(TForm)
    StatusBar1, StatusBar2: TStatusBar;
    Button1: TButton;
    RadioButton1: TRadioButton;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure MouseOverChanged(Sender: TObject);
  private
    { Private declarations }
  public
    procedure ActiveControlChanged(Sender: TObject);
  end;

var
  Active : TWinControl;
  I: Integer;
```

```
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.ActiveControlChanged(Sender: TObject);
  var
    Active : TWinControl;
    I: Integer;
  begin
    Active := nil;
    for I:= 0 to ControlCount -1 do
    begin
      if (Controls[I] is TWinControl) then
        if (Controls[I] as TWinControl).Focused then
          Active := TWinControl(Controls[I]);
    end;
    if ((Active <> nil) and (Active.Hint <> '')) then
    begin
      StatusBar2.SimpleText := GetLongHint(Active.Hint) + ' focus';
    end
  end;

  procedure TForm1.MouseOverChanged(Sender: TObject);
  var
    Active : TWinControl;
    I: Integer;
  begin
    if (Sender is TWinControl) then
    begin
      Active := TWinControl(Sender);
      StatusBar1.SimpleText := GetLongHint(Active.Hint) + ' mouse over';
    end;
  end;

  {
  Assign this method as the OnActiveControlChange event
  handler by setting it from the form's OnCreate event handler:
  }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Screen.OnActiveControlChange := ActiveControlChanged;
  end;

  {
  Make sure you clean up the screen object when the form is
  freed by adding this OnDestroy event handler to the form:
  }
  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    Screen.OnActiveControlChange := nil;
  end;
```

See Also

ActiveControl (see page 13896)

ActiveControl (see page 14206)

32.1.28.6.51 TWinControl.GetActionLinkClass Method

Returns the associated action link class.

Pascal

```
function GetActionLinkClass: TControlActionLinkClass; override;
```

C++

```
virtual __fastcall TControlActionLinkClass GetActionLinkClass();
```

Description

GetActionLinkClass returns the appropriate class type for the action link used with a control. This class is used internally to create an action link object for the control when it has an associated action. The action link links the action to the control client. Each Action (see page 5797) link class is designed to link specific properties and event handlers of the action to its client, based on the types of properties the client supports.

As implemented in TWinControl (see page 6112), GetActionLinkClass returns the TWinControlActionLink (see page 6233) class type, which associates the HelpContext (see page 5826), ShowHint (see page 5840), Caption (see page 5806), Enabled (see page 5821) and Visible (see page 5847) properties and the OnClick (see page 5851) event handler. Descendants of TWinControl (see page 6112) override this method to specify a descendant of TControlActionLink (see page 5892) that handles their configuration of properties and events.

See Also

TWinControlActionLink (see page 6233)

ActionLink (see page 5798)

Action (see page 5797)

32.1.28.6.52 TWinControl.GetChildren Method

Calls a specified method for each child of the control.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the streaming system that loads and saves components. Applications seldom need to call this routine.

GetChildren executes the callback specified by the Proc parameter for every child control listed by the Controls (see page 6197) property that lists the Root parameter as its Owner.

Override GetChildren to limit or augment which child controls are saved with the control. When overriding GetChildren, call the procedure passed as the Proc parameter for every child control that should be saved. The Root parameter indicates the component (usually a form) that owns the control.

See Also

Controls (see page 6197)

Owner (see page 2807)

32.1.28.6.53 TWinControl.GetClientOrigin Method

Returns the value of the ClientOrigin (see page 5809) property.

Pascal

```
function GetClientOrigin: TPoint; override;
```

C++

```
virtual __fastcall TPoint GetClientOrigin();
```

Description

GetClientOrigin is the protected implementation of the ClientOrigin (see page 5809) property.

GetClientOrigin returns a point indicating the position of the top-left corner of the control in screen coordinates.

See Also

ClientOrigin

ClientToScreen (see page 5744)

GetClientRect (see page 6154)

32.1.28.6.54 TWinControl.GetClientRect Method

Returns the value of the ClientRect (see page 5810) property.

Pascal

```
function GetClientRect: TRect; override;
```

C++

```
virtual __fastcall TRect GetClientRect();
```

Description

GetClientRect is the protected implementation of the ClientRect (see page 5810) property. GetClientRect returns a rectangle with the rectangle's Top (see page 5844) and Left (see page 5830) fields set to zero, and its Bottom and Right fields set to the control's ClientHeight (see page 5808) and ClientWidth (see page 5811), respectively.

See Also

ClientHeight (see page 5808)

ClientWidth (see page 5811)

GetClientOrigin (see page 6153)

32.1.28.6.55 TWinControl.GetControlExtents Method

Returns the smallest rectangle in which all the control's children fit.

Pascal

```
function GetControlExtents: TRect; virtual;
```

C++

```
virtual __fastcall TRect GetControlExtents();
```

Description

Call GetControlExtents to determine the smallest rectangle in which all the control's children fit. Windowed controls call GetControlExtents when implementing the AutoSize (see page 5803) property.

See Also

CanAutoSize (see page 6131)

AutoSize (see page 5803)

32.1.28.6.56 TWinControl.GetDeviceContext Method

Provides access to a device context for the control.

Pascal

```
function GetDeviceContext(var WindowHandle: HWnd): HDC; override;
```

C++

```
virtual __fastcall HDC GetDeviceContext(HWnd WindowHandle);
```

Description

Call GetDeviceContext to obtain a handle to a device context for the control.

GetDeviceContext calls the Windows API function GetDC, passing the windowed control's Handle (see page 6207) property. It returns the window's handle in the WindowHandle (see page 6212) parameter and the HDC as the return value. If the call is unsuccessful, the EOutOfResources exception is raised.

C++ Examples:

```
/*
The GetFormImage allows you to easily obtain a bitmap of a
form. The following method can be added to a custom
TWinControl descendant to save itself as a bitmap.  Check the
produced BMP file to confirm.  Click the button and them
double click on the newly created foo.bmp file to view.
*/
class TMyControl : public TColorListBox
{
__published:    // IDE-managed Components
    void __fastcall SaveAsBmp(TFileName fileName);
private:    // User declarations
public:     // User declarations
    __fastcall TMyControl(TComponent* Owner);
};

TForm1 *Form1;
TMyControl *MyControl1;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
__fastcall TMyControl::TMyControl(TComponent* Owner)
    : TColorListBox(Owner)
{
}

typedef struct {
  TLogPalette lpal;
  TPaletteEntry dummy[256];
} LogPal;

void __fastcall TMyControl::SaveAsBmp(TFileName fileName)
{
  TCanvas *tempCanvas = new TCanvas;
  try
  {
    HWND notUsed;
    tempCanvas->Handle = GetDeviceContext(notUsed);
    TImage *image2save = new TImage(Form1);
    try
```

```
      {
        image2save->Height = Height;
        image2save->Width = Width;
        TRect destRect = Rect(0,0,Width,Height);
        TRect sourceRect = destRect;
        image2save->Canvas->CopyRect(destRect, tempCanvas, sourceRect);
        LogPal SysPal;
        SysPal.lpal.palVersion = 0x300;
        SysPal.lpal.palNumEntries = 256;
        GetSystemPaletteEntries(tempCanvas->Handle,0,256,SysPal.lpal.palPalEntry);
        image2save->Picture->Bitmap->Palette = CreatePalette((const tagLOGPALETTE
*)&SysPal.lpal);
        image2save->Picture->SaveToFile(fileName);
      }
      __finally
      {
       delete image2save;
      }
    }
    __finally
    {
      delete tempCanvas;
    }
  }

  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    MyControl1->SaveAsBmp("../foo.bmp");
  }

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    MyControl1 = new TMyControl(Form1);
    MyControl1->Parent = Form1;
    MyControl1->Visible  = true;
  }
```

Delphi Examples:

```
  {
  The GetFormImage allows you to easily obtain a bitmap of a
  form. The following method can be added to a custom
  TWinControl descendant to save itself as a bitmap.  Check the
  produced BMP file to confirm.  Click the button and them
  double click on the newly created foo.bmp file to view.
  }
  type
    TMyControl = class(TColorListBox)
      procedure SaveAsBmp(fileName: TFileName);
    end;
    TForm1 = class(TForm)
      Button1: TButton;
      procedure Button1Click(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      { Public declarations }
    end;

  var
    Form1: TForm1;
    MyControl1: TMyControl;
  implementation

  {$R *.dfm}
```

```
type
  LogPal = record
  lpal : TLogPalette;
  dummy:Array[0..255] of TPaletteEntry;
  end;

procedure TMyControl.SaveAsBmp(fileName: TFileName);
var
  Source: TComponent;
  SysPal : LogPal;
  tempCanvas: TCanvas;
  sourceRect, destRect: TRect;
  image2save: TImage;
  notUsed: HWND;
begin
  tempCanvas := TCanvas.Create;
  try
    tempCanvas.Handle := GetDeviceContext(notUsed);
    image2save:=TImage.create(self);
    try
      with image2save do
      begin
        Height := Self.Height;
        Width :=  Self.Width;
        destRect := Rect(0,0,Width,Height);
        sourceRect := destRect;
        Canvas.CopyRect(destRect,tempCanvas,sourceRect);
        SysPal.lPal.palVersion:=$300;
        SysPal.lPal.palNumEntries:=256;
        GetSystemPaletteEntries(
          tempCanvas.Handle,0,256,SysPal.lpal.palPalEntry);
        Picture.Bitmap.Palette:= CreatePalette(Syspal.lpal);
      end;
      image2save.Picture.SaveToFile(fileName);
    finally
     image2save.Free;
    end;
  finally
    tempCanvas.Free;
  end;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  MyControl1.SaveAsBmp('foo.bmp');
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyControl1:= TMyControl.Create(Form1);
  MyControl1.Parent:= Form1;
  MyControl1.visible := true;
end;
```

See Also

EOutOfResources (see page 2702)

Handle (see page 6207)

32.1.28.6.57 TWinControl.GetParentHandle Method

Returns the window handle for the window of the control's parent.

Pascal

```
function GetParentHandle: HWnd;
```

C++

```
__fastcall HWnd GetParentHandle();
```

Description

Use the handle returned by GetParentHandle as a parameter to Windows API function calls that need the window handle for the parent window of the control. If the Parent (see page 5832) property is not nil (Delphi) or NULL (C++), GetParentHandle returns the Handle (see page 6207) property of the parent control. If the Parent (see page 5832) property is nil (Delphi) or NULL (C++), GetParentHandle returns the value of ParentWindow (see page 6210).

See Also

GetTopParentHandle (see page 6159)

Handle (see page 6207)

Parent (see page 5832)

ParentWindow (see page 6210)

32.1.28.6.58 TWinControl.GetSiteInfo Method

Generates an OnGetSiteInfo (see page 6223) event

Pascal

```
procedure GetSiteInfo(Client: TControl; var InfluenceRect: TRect; MousePos: TPoint; var
CanDock: Boolean); dynamic;
```

C++

```
__fastcall GetSiteInfo(TControl * Client, TRect InfluenceRect, TPoint MousePos, Boolean
CanDock);
```

Description

GetSiteInfo is called automatically during drag-and-dock operations to initialize a TDragDockObject (see page 6009) with information about where a dragged object can be docked, if at all.

Client is a potential docking client.

InfluenceRect returns the area in which the client would be docked.

MousePos is the current mouse position.

CanDock returns whether the client can dock to the windowed control.

As implemented in TWinControl (see page 6112), GetSiteInfo generates an influence rectangle equal to BoundsRect (see page 5803) expanded by 10 pixels on every side, and then calls the OnGetSiteInfo (see page 6223) event handler.

See Also

OnGetSiteInfo (see page 6223)

BoundsRect (see page 5803)

32.1.28.6.59 TWinControl.GetTabOrderList Method

Builds a list of controls in tab order.

Pascal

```
procedure GetTabOrderList(List: TList); dynamic;
```

C++

```
__fastcall GetTabOrderList(TList List);
```

Description

Call GetTabOrderList to construct a list of child controls in tab order.

GetTabOrderList iterates through the control's internal tab-order list, adding each of the child controls to List, including any controls contained in those controls. The result is a list of all the controls and their owned controls, in tab order.

The FindNextControl (see page 6149) method calls GetTabOrderList to build a complete list of the controls that FindNextControl (see page 6149) uses to locate the next control in the tab order.

See Also

FindNextControl (see page 6149)

TabOrder (see page 6211)

32.1.28.6.60 TWinControl.GetTopParentHandle Method

Returns the window handle of the first window not associated with a VCL control that contains the windowed control.

Pascal

```
function GetTopParentHandle: HWnd;
```

C++

```
__fastcall HWnd GetTopParentHandle();
```

Description

Use GetTopParentHandle to access the handle of the window that is hosting the windowed control. GetTopParentHandle walks up the containers of the windowed control until it finds a control whose Parent (see page 5832) property is nil (Delphi) or NULL (C++). If that topmost parent is contained in a non-VCL window (its ParentWindow (see page 6210) is nonzero), GetTopParentHandle returns the non-VCL window. If the windowed control is not contained in a non-VCL window, GetTopParentHandle returns the windowed control's own window handle.

See Also

GetParentHandle (see page 6157)

Handle (see page 6207)

Parent (see page 5832)

ParentWindow (see page 6210)

32.1.28.6.61 TWinControl.HandleAllocated Method

Reports whether a screen object handle exists for the control.

Pascal

```
function HandleAllocated: Boolean;
```

C++

```
__fastcall Boolean HandleAllocated();
```

Description

Query HandleAllocated to find out if the control's underlying screen object has been generated.

If the screen object exists, HandleAllocated returns true. If the screen object does not exist, HandleAllocated returns false. Testing the Handle (see page 6207) property of a control directly causes the window to be created if it does not already exist. Call the HandleAllocated method to determine whether a window exists without creating one as a side effect.

See Also

Handle (see page 6207)

HandleNeeded (see page 6160)

32.1.28.6.62 TWinControl.HandleNeeded Method

Creates a screen object for the control if it doesn't already exist.

Pascal

```
procedure HandleNeeded;
```

C++

```
__fastcall HandleNeeded();
```

Description

Call HandleNeeded to create a screen object for the control.

If the screen object does not exist, HandleNeeded calls the CreateHandle (see page 6135) method for the parent of the control before it creates a screen object for this control.

See Also

Handle (see page 6207)

HandleAllocated (see page 6159)

32.1.28.6.63 TWinControl.InsertControl Method

Inserts a control into the Controls (see page 6197) array property.

Pascal

```
procedure InsertControl(AControl: TControl);
```

C++

```
__fastcall InsertControl(TControl * AControl);
```

Description

Applications should not need to call InsertControl directly. Child controls are automatically inserted and removed when added or deleted at design time. At runtime, use the Parent (see page 5832) property of the child control to insert it in the Controls (see page 6197) array. If the child control is already the child of another control, setting the Parent (see page 5832) property ensures that the child is removed from the Controls (see page 6197) of the original parent.

InsertControl makes the inserted control a child, and the containing control the parent. The AControl parameter is the child control that is inserted into the Controls (see page 6197) array.

C++ Examples:

```
/*
This example uses a button placed next to a group box. When the
user clicks the button, the group box becomes the parent of the
button, so the button moves inside the group box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
  if (Button1->Parent != GroupBox1) // second click?
  {
    RemoveControl(Button1);
    GroupBox1->InsertControl(Button1);
  }
}

/*
Note that it was necessary to remove the button from the Controls
property of the form before the button actually moves into the
group box. This code accomplishes the same thing:
*/

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Button2->Parent = GroupBox1;
}
```

Delphi Examples:

```
{
This example uses a button placed next to a group box. When the
user clicks the button, the group box becomes the parent of the
button, so the button moves inside the group box.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (Button1.Parent <> GroupBox1) then // second click?
  begin
    RemoveControl(Button1);
    GroupBox1.InsertControl(Button1);
  end;
end;

{
Note that it was necessary to remove the button from the Controls
property of the form before the button actually moves into the
group box. This code accomplishes the same thing:
}

procedure TForm1.Button2Click(Sender: TObject);
begin
  Button2.Parent := GroupBox1;
end;
```

See Also

Controls (see page 6197)

Parent (see page 5832)

RemoveControl (see page 6172)

32.1.28.6.64 TWinControl.Invalidate Method

Schedules a control repaint.

Pascal

```
procedure Invalidate; override;
```

C++

```
virtual __fastcall Invalidate();
```

Description

Invalidate informs a control that its entire surface needs to be repainted. Calling Invalidate can prevent flicker caused by a series of partial repaints. There is no performance penalty for calling Invalidate multiple times before the control is actually repainted.

The actual repaint does not occur until the control is updated. To force an immediate repaint, call Repaint (see page 6174) instead.

See Also

Repaint (see page 6174)

Update (see page 6183)

32.1.28.6.65 TWinControl.InvalidateDockHostSite Method

Pascal

```
procedure InvalidateDockHostSite(FocusLost: Boolean);
```

C++

```
__fastcall InvalidateDockHostSite(Boolean FocusLost);
```

Description

This is InvalidateDockHostSite, a member of class TWinControl.

32.1.28.6.66 TWinControl.IsControlActivateMsg Method

Pascal

```
function IsControlActivateMsg(var Message: TWMMouseActivate; Control: TControl = nil): Boolean;
```

C++

```
__fastcall Boolean IsControlActivateMsg(TWMMouseActivate Message, TControl * Control = nil);
```

Description

This is IsControlActivateMsg, a member of class TWinControl.

32.1.28.6.67 TWinControl.IsControlMouseMsg Method

Indicates whether a specified mouse message is directed to one of the windowed control's child controls.

Pascal

```
function IsControlMouseMsg(var Message: TWMMouse): Boolean;
```

C++

```
__fastcall Boolean IsControlMouseMsg(TWMMouse Message);
```

Description

Call IsControlMouseMsg to find out if a mouse message is directed to one of the control's child controls. Specify the mouse message as the value of the Message parameter.

Windows takes care of sending messages to windowed child controls, but for nonwindowed child controls, Windows sends the messages to the parent control, which must then determine which, if any, of its child controls should receive the message.

The WndProc (see page 6187) method of a windowed control calls IsControlMouseMsg to process all mouse message sent to the windowed control.

See Also

WndProc (see page 6187)

32.1.28.6.68 TWinControl.IsQualifyingSite Method

Pascal

```
function IsQualifyingSite(const Client: TControl): Boolean; dynamic;
```

C++

```
__fastcall Boolean IsQualifyingSite(const TControl * Client);
```

Description

This is IsQualifyingSite, a member of class TWinControl.

32.1.28.6.69 TWinControl.KeyDown Method

Respond to key press events.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); dynamic;
```

C++

```
__fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-down message (WM_KEYDOWN) from Windows, its message handler calls the DoKeyDown (see page 6145) method. If DoKeyDown (see page 6145) determines that the control should, in fact, process the character, it calls KeyDown, passing the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown (see page 6225) event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown (see page 6225) event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

See Also

KeyUp (see page 6164)

OnKeyDown (see page 6225)

DoKeyDown (see page 6145)

DoKeyPress (see page 6146)

DoKeyUp (see page 6146)

32.1.28.6.70 TWinControl.KeyPress Method

Respond to keyboard input.

Pascal

```
procedure KeyPress(var Key: Char); dynamic;
```

C++

```
__fastcall KeyPress(Char Key);
```

Description

When a windowed control receives a key-press message (WM_CHAR) from Windows, its message handler calls the DoKeyPress (see page 6146) method. If DoKeyPress (see page 6146) determines that the control should, in fact, process the character, it calls KeyPress, passing the key code in the Key parameter.

KeyPress calls any event handler attached to the OnKeyPress (see page 6227) event. Override KeyPress to provide other responses in addition to the event-handler call.

Either KeyPress or the OnKeyPress (see page 6227) event handler it calls can suppress further processing of a character by setting the Key parameter to zero.

Note: The Key parameter is the character represented by the key that is pressed, not a Windows virtual key code.

See Also

KeyDown (see page 6163)

KeyUp (see page 6164)

OnKeyPress (see page 6227)

32.1.28.6.71 TWinControl.KeyUp Method

Respond to released key.

Pascal

```
procedure KeyUp(var Key: Word; Shift: TShiftState); dynamic;
```

C++

```
__fastcall KeyUp(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-up message (WM_KEYUP) from Windows, its message handler calls the DoKeyUp (see page 6146) method. If DoKeyUp (see page 6146) determines that the control should, in fact, process the character, it calls KeyUp, passing the key code and shift-key state to KeyUp in the Key and Shift parameters, respectively.

KeyUp calls any event handler attached to the OnKeyUp (see page 6228) event. Override KeyUp to provide other responses in addition to the event-handler call.

Either KeyUp or the OnKeyUp (see page 6228) event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use WinAPI virtual key codes to determine the key pressed. For more information, search virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

See Also

KeyDown (see page 6163)

KeyPress (see page 6163)

OnKeyUp (see page 6228)

32.1.28.6.72 TWinControl.MainWndProc Method

Receives Windows messages for the control.

Pascal

```
procedure MainWndProc(var Message: TMessage);
```

C++

```
__fastcall MainWndProc(TMessage Message);
```

Description

MainWndProc is the window procedure for the control that is associated with the control's window when it is created. When Windows sends a message to the control's window, MainWndProc receives it.

MainWndProc does not process or dispatch the messages itself, but rather calls the method specified by WindowProc (see page 5872) to do that. MainWndProc provides an exception-handling block around WindowProc (see page 5872), ensuring that if any unhandled exceptions occur during the processing of a message, the application's HandleException method will handle them.

See Also

HandleException (see page 13674)

WindowProc (see page 5872)

32.1.28.6.73 TWinControl.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

32.1.28.6.74 TWinControl.NotifyControls Method

Sends a message to all the child controls.

Pascal

```
procedure NotifyControls(Msg: Word);
```

C++

```
__fastcall NotifyControls(Word Msg);
```

Description

NotifyControls sends a message with the message ID passed in the Msg parameter to all the controls in the windowed control's Controls (see page 6197) array property. NotifyControls is used to update all the controls in a form of such occurrences as changes in the parent color or font.

NotifyControls constructs a generic message record, filling its message ID field with the value of Msg and setting its parameter and result fields to zeros, then calls the Broadcast (see page 6130) method to send the message to all the child controls.

See Also

Broadcast (see page 6130)

Controls (see page 6197)

32.1.28.6.75 TWinControl.PaintControls Method

Paints each of the child controls in a windowed control using the specified device context.

Pascal

```
procedure PaintControls(DC: HDC; First: TControl);
```

C++

```
__fastcall PaintControls(HDC DC, TControl * First);
```

Description

Call PaintControls to paint all the child controls of this control. Pass the device context as the value of the DC parameter. The First parameter indicates the starting point in the windowed control's child-control list to paint. If First is nil (Delphi) or NULL (C++) or does not indicate one of the child controls, PaintControls paints all the child controls.

The Repaint (see page 6174) method calls PaintControls for the control's parent, passing the control in First to repaint the control and any controls it might intersect. PaintHandler (see page 6166) also calls PaintControls after calling PaintWindow (see page 6169).

See Also

PaintHandler (see page 6166)

PaintWindow (see page 6169)

Repaint (see page 6174)

32.1.28.6.76 TWinControl.PaintHandler Method

Responds to WM_PAINT messages.

Pascal

```
procedure PaintHandler(var Message: TWMPaint);
```

C++

```
__fastcall PaintHandler(TWMPaint Message);
```

Description

PaintHandler is called automatically when the control receives a WM_PAINT message. It calls the BeginPaint and EndPaint API functions, and between them paints the control's background by calling PaintWindow (see page 6169) and any child controls by calling PaintControls (see page 6166).

See Also

PaintControls (see page 6166)

PaintWindow (see page 6169)

32.1.28.6.77 PaintTo Method

This is the overview for the PaintTo method overload.

Overload List

	Name	Description
	TWinControl.PaintTo (HDC, Integer, Integer) (see page 6167)	Draws the windowed control to a device context.
	TWinControl.PaintTo (TCanvas, Integer, Integer) (see page 6167)	Draws the windowed control to a device context.

32.1.28.6.77.1 TWinControl.PaintTo Method (HDC, Integer, Integer)

Draws the windowed control to a device context.

Pascal

```
procedure PaintTo(DC: HDC; X: Integer; Y: Integer); overload;
procedure PaintTo(Canvas: TCanvas; X: Integer; Y: Integer); overload;
```

C++

```
__fastcall PaintTo(HDC DC, int X, int Y);
__fastcall PaintTo(TCanvas Canvas, int X, int Y);
```

Description

Call PaintTo to draw the control on a device context. Specify the device context as the value of the DC parameter and specify the X and Y coordinates on the device context where the top-left corner of the windowed control is to be drawn. PaintTo first erases the background of the device context and then paints the control.

PaintTo is useful for drawing an image of the control into a bitmap DC.

Warning: When using PaintTo to draw on a canvas, you must lock the canvas first (and unlock it after the call to PaintTo. If you do not lock the canvas, Windows calls that occur while the control is painting can cause the canvas to lose its handle.

C++ Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
```

```
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PrintDialog1->Options.Clear();
  PrintDialog1->Options << poPageNums << poSelection;
  PrintDialog1->FromPage = 1;
  PrintDialog1->MinPage = 1;
  PrintDialog1->ToPage = PageControl1->PageCount;
  PrintDialog1->MaxPage = PageControl1->PageCount;
  if (PrintDialog1->Execute())
  {
    int Start, Stop;
    // determine the range the user wants to print
    switch (PrintDialog1->PrintRange)
    {
      case prSelection:
        Start = PageControl1->ActivePage->PageIndex;
        Stop = Start;
        break;
      case prPageNums:
        Start = PrintDialog1->FromPage - 1;
        Stop =  PrintDialog1->ToPage - 1;
        break;
      default:  // prAllPages
        Start = PrintDialog1->MinPage - 1;
        Stop = PrintDialog1->MaxPage - 1;
        break;
    }
    // now, print the pages
    Printer()->BeginDoc();
    for (int i = Start; i <= Stop; i++)
    {
      PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
      if (i != Stop)
        Printer()->NewPage();
    }
    Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
}
procedure TForm1.Button1Click(Sender:TObject);
var
  I, Start, Stop: Integer;
begin
PrintDialog1.Options := [poPageNums, poSelection];
PrintDialog1.FromPage := 1;
PrintDialog1.MinPage := 1;
PrintDialog1.ToPage := PageControl1.PageCount;
PrintDialog1.MaxPage := PageControl1.PageCount;

if PrintDialog1.Execute() then
  begin
    { determine the range the user wants to print }
    with PrintDialog1 do
    begin
```

```
        if PrintRange = prAllPages then
          begin
          Start := MinPage - 1;
          Stop := MaxPage - 1;
        end
        else if PrintRange = prSelection then
        begin
          Start := PageControl1.ActivePage.PageIndex;
          Stop := Start;
          end
        else  { PrintRange = prPageNums }
        begin
          Start := FromPage - 1;
          Stop := ToPage - 1;
        end;
      end;
      { now, print the pages }
      with Printer do
      begin
        BeginDoc;
        for I := Start to Stop do
        begin
          PageControl1.Pages[I].PaintTo(Handle, 10, 10);
          if I <> Stop then
            NewPage;
        end;
        EndDoc;
      end;
    end;
  end;
```

See Also

PaintWindow (see page 6169)

DoubleBuffered (see page 6205)

Lock (see page 14463)

32.1.28.6.78 TWinControl.PaintWindow Method

Renders the image of a windowed control.

Pascal

```
procedure PaintWindow(DC: HDC); virtual;
```

C++

```
virtual __fastcall PaintWindow(HDC DC);
```

Description

Call PaintWindow to repaint the control. PaintWindow sends a WM_PAINT message to the windowed control's DefaultHandler (see page 6139) method, setting the message record's WParam field to the value passed in DC and the other parameter and result fields to zeros.

See Also

DefaultHandler (see page 6139)

PaintTo (see page 6167)

DoubleBuffered (see page 6205)

32.1.28.6.79 TWinControl.PaletteChanged Method

Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.

Pascal

```
function PaletteChanged(Foreground: Boolean): Boolean; override;
```

C++

```
virtual __fastcall Boolean PaletteChanged(Boolean Foreground);
```

Description

PaletteChanged is called automatically when the Windows system palette changes. TWinControl (see page 6112) responds to these notifications by trying to realize the windowed control's palette, if any, into the current device context, and then passing on the PaletteChanged notification to each of the child controls in turn. If the change in the system palette causes the actual value of the control's palette to change, PaletteChanged will invalidate the control so that it can repaint with the new palette.

Windows paints the active window with a foreground palette, while other windows are forced to use background palettes. Background palettes are approximate matches to the colors specified by the control's logical palette, given the limitations imposed by implementing the foreground palette. Windows only allows a single foreground palette.

PaletteChanged allows the control to obtain a new realization of its palette when the system palette changes. When Foreground is true, the form has been activated and the control is specifying the new foreground palette. When Foreground is false, another application or form has changed the foreground palette, and controls that are sensitive to the available palette should realize new background palettes to best match their logical palettes.

If the control does not have a logical palette to be realized into the current device context, GetPalette (see page 5765) will return a handle of 0, and PaletteChanged will not try to realize a palette for the windowed control. Instead, it simply passes the notification on to the child controls. If the windowed control or any of its child controls realize a palette into the current device context in response to PaletteChanged, this method returns true.

Override PaletteChanged to change the way in which controls are given an opportunity to realize a new palette when the Windows system palette changes. For example, override PaletteChanged to change the order in which child controls realize their palettes, giving a particular control the foreground palette when the form is activated, or to pass palette change notification to custom controls implemented outside of the VCL, such as in DLLs.

Override GetPalette (see page 5765) instead to affect whether the windowed control actually has a palette to realize, or to change the value of the control's logical palette.

Note: PaletteChanged is called only when the run-time video mode requires palette support, such as for 256 color mode, but not 16 million color mode.

See Also

GetDeviceContext (see page 6155)

GetPalette (see page 5765)

32.1.28.6.80 TWinControl.PreProcessMessage Method

Pascal

```
function PreProcessMessage(var Msg: TMsg): Boolean; dynamic;
```

C++

```
__fastcall Boolean PreProcessMessage(TMsg Msg);
```

Description

This is PreProcessMessage, a member of class TWinControl.

32.1.28.6.81 TWinControl.ReadState Method

Prepares the control for having its properties assigned values from a stream.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

Override ReadState to change the preparations the control makes as it readies itself to have its property values assigned from a stream. For example, a control might destroy temporary internal data structures or objects before new instances are loaded from the stream. Be sure to include a call to the inherited method when overriding ReadState.

As implemented in TWinControl (see page 6112), ReadState disables control alignment until all the child controls have been read from the stream, calls the inherited ReadState method, and re-enables the alignment of all the controls. The method then builds the tab order list of controls and synchronizes its visibility with that of its parent control.

See Also

Parent (see page 5832)

ReadState (see page 5778)

32.1.28.6.82 TWinControl.Realign Method

Forces the control to realign children.

Pascal

```
procedure Realign;
```

C++

```
__fastcall Realign();
```

Description

The EnableAlign (see page 6148) method calls Realign when the reference count reaches zero. It adjusts the size and position of any child controls according to their Align (see page 5798) properties.

If all of the child controls have their Align (see page 5798) properties set to alNone, Realign has no effect.

See Also

Align (see page 5798)

DisableAlign (see page 6141)

EnableAlign (see page 6148)

32.1.28.6.83 TWinControl.RecreateWnd Method

Recreate underlying Windows screen object.

Pascal

```
procedure RecreateWnd;
```

C++

```
__fastcall RecreateWnd();
```

Description

Call RecreateWnd to recreate the control from scratch. RecreateWnd destroys the underlying Windows screen object. The next time the Handle (see page 6207) property is referenced (when the object is repainted, for example), a new Windows screen object is created. Some changes in control properties are not implemented until this re-creation occurs.

See Also

DestroyHandle (see page 6140)

Handle (see page 6207)

HandleNeeded (see page 6160)

32.1.28.6.84 TWinControl.ReloadDockedControl Method

Reloads a docked control.

Pascal

```
procedure ReloadDockedControl(const AControlName: string; var AControl: TControl); dynamic;
```

C++

```
__fastcall ReloadDockedControl(const AnsiString AControlName, TControl * AControl);
```

Description

Use ReloadDockedControl to obtain a reference to control given its name. The AControlName parameter is the name of the child control. The corresponding control is returned as the AControl parameter. ReloadDockedControl can only locate a control if it has the same Owner as the windowed control.

The dock manager calls ReloadDockedControl to obtain a control reference that it then docks to the windowed control.

See Also

DockManager (see page 6202)

FindChildControl (see page 6148)

FindComponent (see page 2772)

Owner (see page 2807)

32.1.28.6.85 TWinControl.RemoveControl Method

Removes a specified control from the Controls (see page 6197) array.

Pascal

```
procedure RemoveControl(AControl: TControl);
```

C++

```
__fastcall RemoveControl(TControl * AControl);
```

Description

RemoveControl removes a child control from the Controls (see page 6197) property. After calling RemoveControl, the control

is no longer the parent of the child specified by the AControl parameter.

Applications should not call RemoveControl directly. Child controls are automatically inserted and removed when added or deleted at design time. At runtime, use the Parent (see page 5832) property of the child control to remove it from the Controls (see page 6197) array.

C++ Examples:

```
/*
This example uses a button placed next to a group box. When the
user clicks the button, the group box becomes the parent of the
button, so the button moves inside the group box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (Button1->Parent != GroupBox1) // second click?
  {
    RemoveControl(Button1);
    GroupBox1->InsertControl(Button1);
  }
}

/*
Note that it was necessary to remove the button from the Controls
property of the form before the button actually moves into the
group box. This code accomplishes the same thing:
*/

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Button2->Parent = GroupBox1;
}
```

Delphi Examples:

```
{
This example uses a button placed next to a group box. When the
user clicks the button, the group box becomes the parent of the
button, so the button moves inside the group box.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (Button1.Parent <> GroupBox1) then // second click?
  begin
    RemoveControl(Button1);
    GroupBox1.InsertControl(Button1);
  end;
end;

{
Note that it was necessary to remove the button from the Controls
property of the form before the button actually moves into the
group box. This code accomplishes the same thing:
}

procedure TForm1.Button2Click(Sender: TObject);
begin
  Button2.Parent := GroupBox1;
end;
```

See Also

Controls (see page 6197)

InsertControl (see page 6160)

Parent (see page 5832)

32.1.28.6.86 TWinControl.RemoveWindowProps Method

Pascal

```
procedure RemoveWindowProps;
```

C++

```
__fastcall RemoveWindowProps();
```

Description

This is RemoveWindowProps, a member of class TWinControl.

32.1.28.6.87 TWinControl.Repaint Method

Repaints the entire control.

Pascal

```
procedure Repaint; override;
```

C++

```
virtual __fastcall Repaint();
```

Description

Call Repaint to repaint the control.

As implemented in TWinControl (see page 6112), Repaint calls the Invalidate (see page 6161) method and then the Update (see page 6183) method to repaint the control.

See Also

Invalidate (see page 6161)

Update (see page 6183)

32.1.28.6.88 TWinControl.ResetIme Method

Restores the input method editor (IME) that was active when the application started.

Pascal

```
procedure ResetIme;
```

C++

```
__fastcall ResetIme();
```

Description

For controls that specify the ImeName (see page 6208) property, call ResetIme to revert to the IME that was active when the application started. If the IME was deactivated by setting the ImeMode (see page 6207) property to imDisable, ResetIme restarts the default IME.

Applications cannot call ResetIme. ResetIme is called automatically when the control loses input focus. It is exposed as a protected method so descendant objects can restore the default IME at other times.

See Also

ImeMode (see page 6207)

ImeName (see page 6208)

SetIme (see page 6181)

DefaultIme (see page 14216)

32.1.28.6.89 TWinControl.ResetImeComposition Method

Directs the composition window of the input method editor (IME) to perform a specific action.

Pascal

```
function ResetImeComposition(Action: DWORD): Boolean;
```

C++

```
__fastcall Boolean ResetImeComposition(DWORD Action);
```

Description

Call ResetImeComposition to direct what the composition window does with the input typed by the user. The composition window is the window displayed by the IME that provides feedback to the user about what keystrokes have been typed, and how they are converted into asian characters.

The Action (see page 5797) parameter indicates what the composition window should do with its current contents. Action (see page 5797) can have one of the following values:

Value	Meaning
CPS_CANCEL	Clear the composition string and set the status to no composition string.
CPS_COMPLETE	Set the composition string as the result string.
CPS_CONVERT	Convert the composition string.
CPS_REVERT	Cancel the current composition string and revert to the unconverted string.

ResetImeComposition returns true if the composition window performs the requested action.

See Also

ImeMode (see page 6207)

ImeName (see page 6208)

SetImeCompositionWindow (see page 6181)

32.1.28.6.90 TWinControl.ScaleBy Method

Rescale control and its children.

Pascal

```
procedure ScaleBy(M: Integer; D: Integer);
```

C++

```
__fastcall ScaleBy(int M, int D);
```

Description

ScaleBy resizes a control without moving its upper left corner. This is similar to changing the Height (see page 5825) and Width (see page 5849) properties, but the control also attempts to rescale and rearrange any child controls to maintain their relative size and placement.

The M and D parameters define a multiplier and divisor by which to scale the control. For example, to make a control 75% of its original size, specify the value of M as 75, and the value of D as 100. Any pair of values that has the same ratio has the same

effect. Thus M = 3 and D = 4 also makes the control 75% of its previous size.

To rescale the control's children without rescaling the control itself, use ScaleControls (see page 6176).

See Also

ChangeScale (see page 6132)

ScaleControls (see page 6176)

Height (see page 5825)

Width (see page 5849)

Considerations When Dynamically Resizing Forms and Controls

32.1.28.6.91 TWinControl.ScaleControls Method

Rescale child controls only.

Pascal

```
procedure ScaleControls(M: Integer; D: Integer);
```

C++

```
__fastcall ScaleControls(int M, int D);
```

Description

Call ScaleControls to rescale only the children of the control, while leaving the control itself the same size. The parameters have the same meaning as in the ScaleBy (see page 6175) method.

See Also

ChangeScale (see page 6132)

ScaleBy (see page 6175)

32.1.28.6.92 TWinControl.ScrollBy Method

Scroll control contents.

Pascal

```
procedure ScrollBy(DeltaX: Integer; DeltaY: Integer);
```

C++

```
__fastcall ScrollBy(int DeltaX, int DeltaY);
```

Description

Call ScrollBy to scroll the contents within the control. While ScrollBy can be used for any TWinControl (see page 6112), it makes the most sense to use it for descendants of TScrollingWinControl.

Applications seldom need to call the ScrollBy method unless they implement their own scrolling interface rather than relying on a scroll bar.

The DeltaX parameter is the change in pixels along the X axis. A positive DeltaX value scrolls the contents to the right; a negative value scrolls the contents to the left. The DeltaY parameter is the change in pixels along the Y axis. A positive DeltaY value scrolls the contents down; a negative value scrolls the contents up.

See Also

HorzScrollBar (see page 14329)

ScrollInView (see page 14328)

VertScrollBar (see page 14330)

32.1.28.6.93 TWinControl.SelectFirst Method

Locates first selectable child control.

Pascal

```
procedure SelectFirst;
```

C++

```
__fastcall SelectFirst();
```

Description

Call SelectFirst to find the first child that can be selected within the control's tab order. SelectFirst calls the FindNextControl (see page 6149) method, passing nil (Delphi) or NULL (C++) as the current control. Once the first selectable control is found, SelectFirst makes the control the active control on the form.

See Also

FindNextControl (see page 6149)

SelectNext (see page 6177)

ActiveControl (see page 13896)

32.1.28.6.94 TWinControl.SelectNext Method

Moves the input focus from the current child control to the next one in the tab order.

Pascal

```
procedure SelectNext(CurControl: TWinControl; GoForward: Boolean; CheckTabStop: Boolean);
```

C++

```
__fastcall SelectNext(TWinControl * CurControl, Boolean GoForward, Boolean CheckTabStop);
```

Description

Call SelectNext to move the child control focus. SelectNext selects the first child that follows or precedes CurControl in the tab order and that meets the criteria specified in the other parameters.

The GoForward parameter controls the direction of the search. If GoForward is true, FindNextControl (see page 6149) searches forward through the child controls in tab order. If GoForward is false, SelectNext searches backward through the controls. The search wraps past the end of the collection back to CurControl.

The CheckTabStop parameter controls whether the control SelectNext finds must be a tab stop. If CheckTabStop is true, the returned control must have its TabStop (see page 6211) property set to true, or the search for the next control continues.

If a child control matches the search criteria, that control obtains the focus. If no such child control is found, the focus remains unchanged.

See Also

SelectFirst (see page 6177)

TabOrder (see page 6211)

32.1.28.6.95 TWinControl.SetBounds Method

Sets the windowed control's boundary properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to change all of the control's boundary properties at once. The same effect can be achieved by setting the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties separately. By setting all four properties at once, SetBounds ensures that the control will not repaint between changes

Specify the values for the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties as the value of the ALeft, ATop, AWidth, and AHeight parameters, respectively.

See Also

Height (see page 5825)

Left (see page 5830)

Top (see page 5844)

Width (see page 5849)

32.1.28.6.96 TWinControl.SetChildOrder Method

Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.

Pascal

```
procedure SetChildOrder(Child: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Child, int Order);
```

Description

Use SetChildOrder when you want to change the order in which child objects are streamed in. Specify the child component whose order you want to change as the value of the Child parameter. Indicate the position you want the component to be in as the value of the Order parameter.

When SetChildOrder is called, items previously below the child's old position move up, and those below the new position move down.

See Also

GetChildren (see page 6153)

32.1.28.6.97 TWinControl.SetDesignVisible Method

Makes a control visible at design time.

Pascal

```
procedure SetDesignVisible(Value: Boolean); override;
```

C++

```
virtual __fastcall SetDesignVisible(Boolean Value);
```

Description

SetDesignVisible is used internally by Delphi during form design.

See Also

TComponentState

32.1.28.6.98 TWinControl.SetFocus Method

Gives the input focus to the control.

Pascal

```
procedure SetFocus; virtual;
```

C++

```
virtual __fastcall SetFocus();
```

Description

Use SetFocus to change input focus to the control. When a control has focus, it receives keyboard events.

C++ Examples:

```
/*
The following OnFind event handler searches a memo component
for the text specified in the FindText property of a find
dialog component. If found, the first occurrence of the text
in Memo1 is selected. The code uses the Pos function to
compare strings, and stores the number of characters to skip
when determining the selection position in the SkipChars
variable. Because there is no handling of case, whole word,
or search direction in this algorithm, it is assumed that
the Options property of FindDialog1 is set to
[frHideMatchCase, frHideWholeWord, frHideUpDown].
*/
void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  for (int I = 0; I < Memo1->Lines->Count; I++)
  {
    int PosReturn = Memo1->Lines->Strings[I].Pos(FindDialog1->FindText);
    if (PosReturn) //found!
    {
      int Skipchars = 0;
      for (int J = 0; J < I; J++)
        Skipchars += Memo1->Lines->Strings[J].Length();
      Skipchars += I*2; // add CR/LF for all skipped lines
      Skipchars += PosReturn - 1;
      Memo1->SetFocus();
      Memo1->SelStart = Skipchars;
      Memo1->SelLength = FindDialog1->FindText.Length();
      break;
    }
  }
}
/*
The following example demonstrates the use of the Perform method
to send a windows message. The EM_SCROLLCARET message scrolls the
```

```
caret in the rich edit into view. The caret position is set to the
contents of the mask edit prior to calling Perform.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    RichEdit1->SelStart = StrToInt(MaskEdit1->Text);
    Edit1->Text = IntToStr(RichEdit1->SelStart);
    RichEdit1->Perform(EM_SCROLLCARET, 0, 0);
    RichEdit1->SetFocus();
}
```

Delphi Examples:

```
{
The following OnFind event handler searches a memo component
for the text specified in the FindText property of a find
dialog component. If found, the first occurrence of the text
in Memo1 is selected. The code uses the Pos function to
compare strings, and stores the number of characters to skip
when determining the selection position in the SkipChars
variable. Because there is no handling of case, whole word,
or search direction in this algorithm, it is assumed that
the Options property of FindDialog1 is set to
[frHideMatchCase, frHideWholeWord, frHideUpDown].
}
procedure TForm1.FindDialog1Find(Sender: TObject);
var
  I, J, PosReturn, SkipChars: Integer;
begin
  for I := 0 to Memo1.Lines.Count do
  begin
    PosReturn := Pos(FindDialog1.FindText,Memo1.Lines[I]);
    if PosReturn <> 0 then {found!}
    begin
      SkipChars := 0;
      for J := 0 to I - 1 do
        SkipChars := SkipChars + Length(Memo1.Lines[J]);
      SkipChars := SkipChars + (I*2);
      SkipChars := SkipChars + PosReturn - 1;
      Memo1.SetFocus;
      Memo1.SelStart := SkipChars;
      Memo1.SelLength := Length(FindDialog1.FindText);
      Break;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
const Path = 'country.list';
begin
  Memo1.Lines.LoadFromFile(Path);
end;
{
The following example demonstrates the use of the Perform method
to send a windows message. The EM_SCROLLCARET message scrolls the
caret in the rich edit into view. The caret position is set to the
contents of the mask edit prior to calling Perform.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
with RichEdit1 do
  Begin
    SelStart := StrToInt(MaskEdit1.Text);
    Edit1.Text := InttoStr(SelStart);
    RichEdit1.Perform(EM_SCROLLCARET, 0, 0);
    RichEdit1.SetFocus;
  end;
```

```
end;
```

See Also

CanFocus (see page 6131)

FocusControl (see page 13875)

Focused (see page 6150)

32.1.28.6.99 TWinControl.SetIme Method

Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.

Pascal

```
procedure SetIme;
```

C++

```
__fastcall SetIme();
```

Description

Applications cannot call SetIme. SetIme is called automatically when the control receives input focus. It is exposed as a protected method so that descendant objects can cause the ImeMode (see page 6207) and ImeName (see page 6208) properties to take effect at other times.

See Also

ImeMode (see page 6207)

ImeName (see page 6208)

32.1.28.6.100 TWinControl.SetImeCompositionWindow Method

Sets the position and font of the composition window.

Pascal

```
function SetImeCompositionWindow(Font: TFont; XPos: Integer; YPos: Integer): Boolean;
```

C++

```
__fastcall Boolean SetImeCompositionWindow(TFont Font, int XPos, int YPos);
```

Description

Call SetImeComposition to change the position or font of the composition window. The composition window is the window displayed by the input method editor (IME) that provides feedback to the user about what keystrokes have been typed, and how they are converted into asian characters.

The Font (see page 5823) parameter describes the font that the composition window should use. Setting the Font (see page 5823) parameter to nil (Delphi) or NULL (C++) repositions the composition window without changing the font. XPos and YPos specify the desired global coordinates for the composition window.

SetImeCompositionWindow returns true if the font and position are successfully assigned.

See Also

ImeMode (see page 6207)

ImeName (see page 6208)

TFont (see page 14497)

32.1.28.6.101 TWinControl.SetParent Method

Sets the parent of the control.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. Override SetParent to execute additional code when the value of the Parent property changes.

The AParent parameter specifies the new parent of the control.

If the control already has a parent, SetParent removes the control from that parent's list of controls by calling the parent's RemoveControl method. If AParent is not nil (Delphi) or NULL (C++), SetParent calls its InsertControl method to add the control to its list of controls.

The TControl destructor calls SetParent(nil (Delphi) or NULL (C++)) to remove the control from its parent's control list before destroying the component.

See Also

Destroy (see page 5732)

InsertControl (see page 6160)

Parent (see page 5832)

RemoveControl (see page 6172)

32.1.28.6.102 TWinControl.SetParentBackground Method

Pascal

```
procedure SetParentBackground(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetParentBackground(Boolean Value);
```

Description

This is SetParentBackground, a member of class TWinControl.

32.1.28.6.103 TWinControl.SetZOrder Method

Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.

Pascal

```
procedure SetZOrder(TopMost: Boolean); override;
```

C++

```
virtual __fastcall SetZOrder(Boolean TopMost);
```

Description

Use SetZOrder to rearrange overlapping controls within a parent control or (if the control has no parent) overlapping windows on

the screen. If the TopMost parameter is true, the control becomes the top control; otherwise, it becomes the bottom control.

TWinControl (see page 6112) controls always stack on top of other controls, regardless of the Z order.

After changing the order, SetZOrder invalidates the control to ensure repainting to reflect the new order.

See Also

BringToFront (see page 5740)

Invalidate (see page 6161)

Parent (see page 5832)

SendToBack (see page 5782)

32.1.28.6.104 TWinControl.ShowControl Method

Ensures that a specified child control is visible.

Pascal

```
procedure ShowControl(AControl: TControl); virtual;
```

C++

```
virtual __fastcall ShowControl(TControl * AControl);
```

Description

Call ShowControl to display a child control. Specify the control you want to ensure will be visible as the value of the AControl parameter.

As implemented in TWinControl (see page 6112), ShowControl simply calls the ShowControl method of the control's parent. Derived objects override ShowControl to perform whatever actions are necessary to allow the indicated child object to be visible. Possible actions include scrolling the particular child object into view, or changing to the appropriate notebook page.

See Also

Show (see page 5788)

Parent (see page 5832)

32.1.28.6.105 TWinControl.Update Method

Forces the control to update.

Pascal

```
procedure Update; override;
```

C++

```
virtual __fastcall Update();
```

Description

Update repaints any part of the control surface that is out of date. Normally, updates occur automatically, but an Update call may be necessary before lengthy processing that might interfere with automatic updates. Calling Update unnecessarily can increase overhead and cause screen flicker.

Update only repaints areas of the control the have been determined to be out of date. To force immediate repainting of the entire control, call the Repaint (see page 6174) method. To notify a control that it is out of date (without forcing an immediate repaint), call the Invalidate (see page 6161) method.

C++ Examples:

```
/*
StepBy, CreateForm, Run, Show, Update, Initialize example
This example shows progress of loading forms as an application starts up. The code example is
placed in the project source (*.dpr) file. To see project source, select the View | Project
Source menu item.
You will need to set up your project with the following steps before using the code example:
o          Add four additional forms to a default project.
o          Place a TProgressBar on Form5
o          Take the Project|Options|Forms menu option and place Form5 on the available forms
list.
o          Change the code of your project (*.dpr) file to look like the example.
*/
#include <vcl\vcl.h>
#include "Unit5.h"
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("TAppCreateForm.cpp", Form1);
USERES("Project1.res");
USEFORM("Unit2.cpp", Form2);
USEFORM("Unit3.cpp", Form3);
USEFORM("Unit4.cpp", Form4);
USEFORM("Unit5.cpp", Form5);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
  try
  {
    Application->Initialize();
    SetApplicationMainFormOnTaskBar(Application, true);
    Application->CreateForm(__classid(TForm5), &Form5);
    Form5->ProgressBar1->Max = 100;
    Form5->Show();   // show a splash screen contain ProgressBar control
    Form5->Update(); // force display of Form5
    Application->CreateForm(__classid(TForm1), &Form1);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form1 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm2), &Form2);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form2 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm3), &Form3);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form3 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm4), &Form4);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form4 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
//  delete Form5;
    Application->Run();
  }
  catch (Exception &exception)
  {
    Application->ShowException(&exception);
  }
  return 0;
}
```

Delphi Examples:

```
{
This example shows progress of loading forms as an
application starts up. The code example is placed in the
project source (*.dpr) file.  To see project source, select
the View | Project Source menu item.
You will need to set up your project with the following steps
before using the code example:
    Add four additional forms to a default project.
    Place a TProgressBar on Form5
    Take the Project|Options|Forms menu option and place
          Form5 on the available forms list.
    Change the code of your project (*.dpr) file to look
          like the example.
}
begin
  Application.Initialize;
  with TForm5.Create(nil) do
  try
    Application.MainFormOnTaskbar := True;
    ProgressBar1.Max := 100;
    Show;  // show a splash screen contain ProgressBar control
    Update; // force display of Form5
    Application.CreateForm(TForm1, Form1);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form1 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm2, Form2);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form2 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm3, Form3);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form3 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm4, Form4);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form4 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
  finally
    Free;
  end;
  Application.Run;
```

See Also

Invalidate (see page 6161)

Repaint (see page 6174)

32.1.28.6.106 TWinControl.UpdateBounds Method

Pascal

```
procedure UpdateBounds;
```

C++

```
__fastcall UpdateBounds();
```

Description

This is UpdateBounds, a member of class TWinControl.

32.1.28.6.107 TWinControl.UpdateControlOriginalParentSize Method

Pascal

```
procedure UpdateControlOriginalParentSize(AControl: TControl; var AOriginalParentSize:
TPoint); virtual;
```

C++

```
virtual __fastcall UpdateControlOriginalParentSize(TControl * AControl, TPoint
AOriginalParentSize);
```

Description

This is UpdateControlOriginalParentSize, a member of class TWinControl.

32.1.28.6.108 TWinControl.UpdateControlState Method

Respond to state change.

Pascal

```
procedure UpdateControlState;
```

C++

```
__fastcall UpdateControlState();
```

Description

UpdateControlState is called automatically when any display change occurs that might affect the internal state of the control.

As implemented in TWinControl (see page 6112), UpdateControlState climbs the chain of parent controls to verify that each control in the chain has Showing (see page 6210) equal to true. If this is the case, UpdateControlState makes sure that the Showing (see page 6210) property is accurate for this control and for all its descendants. Descendant classes can extend this behavior.

See Also

Handle (see page 6207)

Show (see page 5788)

ShowControl (see page 6183)

Showing (see page 6210)

32.1.28.6.109 TWinControl.UpdateRecreatingFlag Method

Pascal

```
procedure UpdateRecreatingFlag(Recreating: Boolean);
```

C++

```
__fastcall UpdateRecreatingFlag(Boolean Recreating);
```

Description

This is UpdateRecreatingFlag, a member of class TWinControl.

32.1.28.6.110 TWinControl.UpdateUIState Method

Updates form User Interface State as needed

Pascal

```
procedure UpdateUIState(CharCode: Word);
```

C++

```
__fastcall UpdateUIState(Word CharCode);
```

Description

UpdateUIState is called in component code to ensure that the User Interface State is consistently updated. User Interface State determinates whether keyboard accelerators and focus indicators are hidden or shown. A control should call UpdateUIState whenever it receives a keyboard event that might invalidate the User Interface State. CharCode is the virtual key code of the keyboard event.

32.1.28.6.111 TWinControl.WndProc Method

Provides specific message responses for the control.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

Override WndProc to change the initial Windows message handler for the control. The WindowProc (see page 5872) property is initialized to point to the WndProc method.

WndProc for TWinControl (see page 6112) overrides the inherited method to define responses for focus, mouse, and keyboard messages. It sends all others to its inherited WndProc.

When overriding WndProc to provide specialized responses to messages, call the inherited WndProc at the end to dispatch any unhandled messages.

C++ Examples:

```
/*
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
*/
class TMyListBoxDescendant : public TListBox
{
__published:    // IDE-managed Components
    void __fastcall SubClassWndProc(Messages::TMessage &Message);
    void __fastcall ToggleSubClass(bool On);
    void __fastcall OnDrawItemProc(
      TWinControl *Control, int Index, const TRect &Rect,
      TOwnerDrawState State);
private:    // User declarations
public:     // User declarations
    __fastcall TMyListBoxDescendant(TComponent* Owner);
};

TForm1 *Form1;
TMyListBoxDescendant *MyListBoxDescendant1;
Graphics::TBitmap *bitmap0;

const WM_STYLEMESSAGE = WM_USER + 2000;
```

```
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

__fastcall TMyListBoxDescendant::TMyListBoxDescendant(TComponent* Owner)
    : TListBox(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
}

void __fastcall TForm1::SubClassRadioGroup1Click(TObject *Sender)
{
  MyListBoxDescendant1->ToggleSubClass(SubClassRadioGroup1->ItemIndex == 0);
}

void __fastcall TMyListBoxDescendant::SubClassWndProc(Messages::TMessage &Message)
{
  if (Message.Msg == WM_STYLEMESSAGE)
    Style = (TListBoxStyle)Message.WParam;
  else
    WndProc(Message);
}

void __fastcall TMyListBoxDescendant::ToggleSubClass(bool On)
{
  if (On)
    WindowProc = SubClassWndProc;
  else
    WindowProc = WndProc;
}

void __fastcall TMyListBoxDescendant::OnDrawItemProc(TWinControl *Control, int Index,
      const TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *bitmap; // temporary variable for item's bitmap
  int Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *canvas = ((TListBox *)Control)->Canvas;
  canvas->FillRect(Rect); // clear the rectangle
  bitmap = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
  if (bitmap)
  {
    canvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, bitmap->Width, bitmap->Height),
      bitmap, Bounds(0, 0, bitmap->Width, bitmap->Height), clRed); // render bitmap
    Offset += bitmap->Width + 4;   // add four pixels between bitmap and text
  }
```

```
  // display the text
  canvas->TextOut(Rect.Left + Offset, Rect.Top, ((TListBox *)Control)->Items->Strings[Index]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyListBoxDescendant1 = new TMyListBoxDescendant(Form1);
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Parent = Form1;
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Left =
    SubClassRadioGroup1->Left + SubClassRadioGroup1->Width + 30;;
  MyListBoxDescendant1->Top = SubClassRadioGroup1->Top;
  MyListBoxDescendant1->Height = SubClassRadioGroup1->Height;
  MyListBoxDescendant1->OnDrawItem =
    MyListBoxDescendant1->OnDrawItemProc;

  bitmap0 = new Graphics::TBitmap;
  ImageList1->GetBitmap(0, bitmap0);
  MyListBoxDescendant1->Items->AddObject("Butterfly", bitmap0);

  SubClassRadioGroup1->Items->Add("SubClassWndProc");
  SubClassRadioGroup1->Items->Add("WndProc");
  SubClassRadioGroup1->ItemIndex = 2;
}
```

Delphi Examples:

```
{
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
}

type
  TMyListBoxDescendant = class(TlistBox)
    procedure SubClassWndProc(var Message: TMessage);
    procedure ToggleSubClass(On: Boolean);
    procedure OnDrawItemProc(
      Control: TWinControl;
      Index: Integer;
      Rect:TRect;
      State: TOwnerDrawState);
  end;
  TForm1 = class(TForm)
    SubClassRadioGroup1: TRadioGroup;
    Button1: TButton;
    ImageList1: TImageList;
    Button2: TButton;
    procedure SubClassRadioGroup1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  MyListBoxDescendant1: TMyListBoxDescendant;
  bitmap0: TBitmap;
```

```
implementation

{$R *.dfm}

const WM_STYLEMESSAGE = WM_USER + 2000;

procedure TForm1.Button1Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
end;

procedure TMyListBoxDescendant.SubClassWndProc(var Message: TMessage);
begin
  if (Message.Msg = WM_STYLEMESSAGE) then
    Style:= TListBoxStyle(Message.WParam)
  else
    WndProc(Message);
end;

procedure TMyListBoxDescendant.ToggleSubClass(On: Boolean);
begin
  if On then
    WindowProc := SubClassWndProc
  else
    WindowProc := WndProc;
end;

procedure TForm1.SubClassRadioGroup1Click(Sender: TObject);
begin
  MyListBoxDescendant1.ToggleSubClass(
    SubClassRadioGroup1.ItemIndex = 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyListBoxDescendant1:= TMyListBoxDescendant.Create(self);
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Parent:= Form1;
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Left:=
    SubClassRadioGroup1.Left + SubClassRadioGroup1.Width + 30;;
  MyListBoxDescendant1.Top:= SubClassRadioGroup1.Top;
  MyListBoxDescendant1.Height:= SubClassRadioGroup1.Height;
  MyListBoxDescendant1.OnDrawItem:=
    MyListBoxDescendant1.OnDrawItemProc;

  bitmap0 := TBitmap.Create;
  ImageList1.GetBitmap(0, bitmap0);
  MyListBoxDescendant1.Items.AddObject('Butterfly', bitmap0);

  SubClassRadioGroup1.Items.Add('SubClassWndProc');
  SubClassRadioGroup1.Items.Add('WndProc');
```

```
  SubClassRadioGroup1.ItemIndex := 2;
end;

procedure TMyListBoxDescendant.OnDrawItemProc(
  Control: TWinControl;
  Index: Integer;
  Rect:TRect;
  State: TOwnerDrawState);
var
  Bitmap: TBitmap;      { temporary variable for the item's bitmap }
  Offset: Integer;      { text offset width }
begin
  { draw on control canvas, not on the form }
  with (Control as TListBox).Canvas do
  begin
    FillRect(Rect);       { clear the rectangle }
    Offset := 2;          { provide default offset }
    Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
    if Bitmap <> nil then
    begin
      BrushCopy(
        Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
        Bitmap,
        Bounds(0, 0, Bitmap.Width, Bitmap.Height),
        clRed);  {render bitmap}
      Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
    end;
    TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
text }
  end;
end;
```

See Also

DefWndProc (see page 6199)

MainWndProc (see page 6165)

WndProc (see page 5791)

WindowProc (see page 5872)

32.1.28.7 TWinControl Properties

The properties of the TWinControl class are listed here.

Protected Properties

	Name	Description
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.

	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.

Public Properties

	Name	Description
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	Padding (see page 6209)	Specifies the padding of a control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

32.1.28.7.1 TWinControl.AlignDisabled Property

Indicates child control realignment disabled.

Pascal

```
property AlignDisabled: Boolean;
```

C++

```
__property Boolean AlignDisabled;
```

Description

AlignDisabled returns true if child control realignment has been temporarily disabled by a call to DisableAlign (see page 6141).

See Also

DisableAlign (see page 6141)

EnableAlign (see page 6148)

32.1.28.7.2 TWinControl.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner (see page 6193), BevelOuter (see page 6194), and BevelKind (see page 6193) properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

32.1.28.7.3 TWinControl.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter (see page 6194) is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

32.1.28.7.4 TWinControl.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth (see page 6194) and the cut of the bevel specified by BevelInner (see page 6193) or BevelOuter (see page 6194), can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

32.1.28.7.5 TWinControl.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner (see page 6193) to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

32.1.28.7.6 TWinControl.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

32.1.28.7.7 TWinControl.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics (see page 14387) or text drawn by the control is clipped to the area within the border.

See Also

ClientRect

BevelOuter (see page 6194)

32.1.28.7.8 TWinControl.Brush Property

Determines the color and pattern used for painting the background of the control.

Pascal

```
property Brush: TBrush;
```

C++

```
__property TBrush Brush;
```

Description

The Brush property accesses the TBrush object that determines pattern and color for the control background. Brush is a read-only property, but an application can manipulate the TBrush object by setting its properties or by using its Assign method.

See Also

Assign (see page 14433)

Color (see page 14437)

Style (see page 14438)

Bitmap (see page 14436)

Color (see page 14437)

Assign (see page 14433)

32.1.28.7.9 TWinControl.ControlCount Property

Returns the number of child controls.

Pascal

```
property ControlCount: Integer;
```

C++

```
__property int ControlCount;
```

Description

Read ControlCount when iterating over all the children of this control. The children of the control are listed in the Controls (see page 6197) property array.

ControlCount is a read-only property.

Note: The value of ControlCount is always 1 greater than the highest Controls (see page 6197) index, because the first Controls (see page 6197) index is 0.

C++ Examples:

```
/*
This example uses an updown control and a group box on a
form, with several controls contained within the group box.
When the updown control or is clicked, the controls within
the group box are moved in the appropriate direction.
*/
void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  int I;
  TControl *ChildControl;

  for (I = 0; I < GroupBox1->ControlCount; I++)
  {
    ChildControl = GroupBox1->Controls[I];
    if (Button == btNext)
      ChildControl->Top = ChildControl->Top + 15;
    else
      ChildControl->Top = ChildControl->Top - 15;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorarray[5] = {clYellow, clGreen, clBlue, clLime, clFuchsia};
  GroupBox1->Brush->Color = clRed;
  for (int i = 0; i < GroupBox1->ControlCount; i++)
    ((TWinControl *)GroupBox1->Controls[i])->Brush->Color = colorarray[i];
}
```

Delphi Examples:

```
{
This example uses an updown control and a group box on a
form, with several controls contained within the group box.
When the updown control or is clicked, the controls within
the group box are moved in the appropriate direction.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
const
  colorarray : Array[0..4] of TColor = (clYellow, clGreen, clBlue, clLime, clFuchsia);
begin
   GroupBox1.Brush.Color := clRed;
  for I:= 0 to GroupBox1.ControlCount -1 do
  begin
    TWinControl(GroupBox1.Controls[I]).Brush.Color := colorarray[I];
  end;
end;

procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
var
```

```
  I: Integer;
  ChildControl: TControl;
begin
  for I:= 0 to GroupBox1.ControlCount -1 do
  begin
    ChildControl := GroupBox1.Controls[I];
    if Button = btNext then
      ChildControl.Top := ChildControl.Top + 15
    else
      ChildControl.Top := ChildControl.Top - 15;
  end;
end;
```

See Also

Controls (see page 6197)

32.1.28.7.10 TWinControl.Controls Property

Lists all child controls.

Pascal

```
property Controls [Index: Integer]: TControl;
```

C++

```
__property TControl * Controls[int Index];
```

Description

Controls is an array of all the child controls. These are all controls that list this control as their Parent (see page 5832) property. The Controls property is convenient for referring to the children of a control by number rather than name. For example, Controls may be used to iterate over all the child controls.

Don't confuse the Controls property with the Components property. The Controls property lists all the controls that are child windows of the control, while the Components property lists all components that it owns. The form owns all components put on it, and therefore, they appear in the form's Components property list, even when they are child windows of a control on the form.

Controls is a read-only property. To add or delete a child control, use the InsertControl (see page 6160) or RemoveControl (see page 6172) methods. To move a child control from one parent to another, set the Parent (see page 5832) of the child control, as that will handle both the RemoveControl (see page 6172) from the original parent and the InsertControl (see page 6160) to the new parent.

C++ Examples:

```
/*
This example uses an updown control and a group box on a
form, with several controls contained within the group box.
When the updown control or is clicked, the controls within
the group box are moved in the appropriate direction.
*/
void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  int I;
  TControl *ChildControl;

  for (I = 0; I < GroupBox1->ControlCount; I++)
  {
    ChildControl = GroupBox1->Controls[I];
    if (Button == btNext)
      ChildControl->Top = ChildControl->Top + 15;
    else
      ChildControl->Top = ChildControl->Top - 15;
  }
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorarray[5] = {clYellow, clGreen, clBlue, clLime, clFuchsia};
  GroupBox1->Brush->Color = clRed;
  for (int i = 0; i < GroupBox1->ControlCount; i++)
    ((TWinControl *)GroupBox1->Controls[i])->Brush->Color = colorarray[i];
}
```

Delphi Examples:

```
{
This example uses an updown control and a group box on a
form, with several controls contained within the group box.
When the updown control or is clicked, the controls within
the group box are moved in the appropriate direction.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
const
  colorarray : Array[0..4] of TColor = (clYellow, clGreen, clBlue, clLime, clFuchsia);
begin
   GroupBox1.Brush.Color := clRed;
  for I:= 0 to GroupBox1.ControlCount -1 do
  begin
    TWinControl(GroupBox1.Controls[I]).Brush.Color := colorarray[I];
  end;
end;

procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
var
  I: Integer;
  ChildControl: TControl;
begin
  for I:= 0 to GroupBox1.ControlCount -1 do
  begin
    ChildControl := GroupBox1.Controls[I];
    if Button = btNext then
      ChildControl.Top := ChildControl.Top + 15
    else
      ChildControl.Top := ChildControl.Top - 15;
  end;
end;
```

See Also

Components (see page 2799)

ControlCount (see page 6195)

InsertControl (see page 6160)

Parent (see page 5832)

RemoveControl (see page 6172)

32.1.28.7.11 TWinControl.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

32.1.28.7.12 TWinControl.DefWndProc Property

Specifies the default window procedure for the windowed control.

Pascal

```
property DefWndProc: Pointer;
```

C++

```
__property void * DefWndProc;
```

Description

Windowed controls use the Windows API function CallWindowProc with DefWndProc to invoke the standard Windows message handling for a window message. Using DefWndProc instead of the WndProc (see page 6187) method bypasses any message processing introduced by the WndProc (see page 6187) method.

DefWndProc fills the role for windowed controls that DefaultHandler (see page 6139) fills for all objects. The DefaultHandler (see page 6139) for TWinControl (see page 6112) uses DefWndProc to pass messages to Windows for processing.

For windowed controls, DefWndProc is initially set to the window procedure of the window class specified in the Params parameter in the CreateParams (see page 6135) method. Change DefWndProc to subclass the window class of a windowed control.

See Also

CreateParams (see page 6135)

DefaultHandler (see page 6139)

WndProc (see page 6187)

32.1.28.7.13 TWinControl.DockClientCount Property

Specifies the number of controls that are docked on the windowed control.

Pascal

```
property DockClientCount: Integer;
```

C++

```
__property int DockClientCount;
```

Description

Use DockClientCount to get the number of controls that are docked to the windowed control. This value can be used as an upper bound when iterating through the DockClients (see page 6201) property.

Note: The DockClients (see page 6201) property can contain controls that are not visible. To get the number of docked clients that are visible, use the VisibleDockClientCount (see page 6212) property instead.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

See Also

DockClients (see page 6201)

VisibleDockClientCount (see page 6212)

32.1.28.7.14 TWinControl.DockClients Property

Lists the controls that are docked to the windowed control.

Pascal

```
property DockClients [Index: Integer]: TControl;
```

C++

```
__property TControl * DockClients[int Index];
```

Description

Use DockClients to access any control that is docked to the windowed control.

C++ Examples:

```
{
This example shows how to undock a dockable object.  This example
is part of the Docking Demo.
}

void __fastcall TTabDockHost::FormClose(TObject *Sender, TCloseAction &Action)
{
  TRect ARect;
  TPoint point;
  if (PageControl1->DockClientCount == 1)
  {
    point =  PageControl1->DockClients[0]->ClientToScreen(Point(0, 0));
    ARect.left = point.x;
    ARect.top = point.y;
    point = PageControl1->DockClients[0]->ClientToScreen(
      Point(PageControl1->DockClients[0]->UndockWidth,
PageControl1->DockClients[0]->UndockHeight));
    ARect.right = point.x;
    ARect.bottom = point.y;
    PageControl1->DockClients[0]->ManualFloat(ARect);
    Action = caFree;
  }
  else
    Action = caHide;
}
```

Delphi Examples:

```
procedure TTabDockHost.FormClose(Sender: TObject;
  var Action: TCloseAction);
var
  ARect: TRect;
begin
  if PageControl1.DockClientCount = 1 then
  begin
    with PageControl1.DockClients[0] do
    begin
      ARect.TopLeft := ClientToScreen(Point(0, 0));
      ARect.BottomRight := ClientToScreen(Point(UndockWidth, UndockHeight));
      ManualFloat(ARect);
    end;
    Action := caFree;
  end else
    Action := caHide;
end;
```

See Also

DockClientCount (see page 6199)

DockSite (see page 6204)

VisibleDockClientCount (see page 6212)

32.1.28.7.15 TWinControl.DockManager Property

Specifies the control's docking manager interface.

Pascal

```
property DockManager: IDockManager;
```

C++

```
__property IDockManager DockManager;
```

Description

Use DockManager to specify the control's docking manager. The docking manager handles the layout of docking zones (where controls are docked) and any painting associated with docking zones.

If you set DockSite (see page 6204) and UseDockManager (see page 6212) to true, but do not assign a value to DockManager, the windowed control generates a default docking manager, using the global DefaultDockTreeClass (see page 6321) variable.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
```

```
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
```

```
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

UseDockManager (see page 6212)

DefaultDockTreeClass (see page 6321)

32.1.28.7.16 TWinControl.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite: Boolean;
```

C++

```
__property Boolean DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
```

```
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

32.1.28.7.17 TWinControl.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered: Boolean;
```

C++

```
__property Boolean DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
```

```
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite (see page 6204)

DockManager (see page 6202)

PaintWindow (see page 6169)

32.1.28.7.18 TWinControl.Handle Property

Provides access to the underlying Windows screen object for the control.

Pascal

```
property Handle: HWnd;
```

C++

```
__property HWnd Handle;
```

Description

The Handle property returns the window handle for the underlying Microsoft Windows screen object. Use the Handle property to bypass the VCL API and directly access the underlying window.

Do not refer to the Handle property during component creation or streaming. The underlying window does not exist until the first time the Handle property is referenced. When this occurs, the HandleNeeded (see page 6160) method is called automatically.

Handle is a read-only property.

C++ Examples:

```
/*
The following code uses the Windows API function ShowWindow
to display Form2 as an icon, but does not activate it.
*/

#include "Unit2.h"

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ShowWindow(Form2->Handle, SW_SHOWMINNOACTIVE);
}
```

Delphi Examples:

```
{
The following code uses the Windows API function ShowWindow
to display Form2 as an icon, but does not activate it.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowWindow(Form2.Handle, SW_SHOWMINNOACTIVE);
end;
```

See Also

HandleNeeded (see page 6160)

32.1.28.7.19 TWinControl.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted

characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme (see page 6181) method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

32.1.28.7.20 TWinControl.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme (see page 6181) method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

32.1.28.7.21 TWinControl.MouseInClient Property

Pascal

```
property MouseInClient: Boolean;
```

C++

```
__property Boolean MouseInClient;
```

Description

This is MouseInClient, a member of class TWinControl.

32.1.28.7.22 TWinControl.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding: TPadding;
```

C++

```
__property TPadding Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding (see page 6103).

See Also

TPadding (see page 6103)

32.1.28.7.23 TWinControl.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground: Boolean;
```

C++

```
__property Boolean ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

32.1.28.7.24 TWinControl.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D (see page 6198) property.

See Also

Ctl3D (see page 6198)

32.1.28.7.25 TWinControl.ParentWindow Property

Reference to parent's underlying control.

Pascal

```
property ParentWindow: HWnd;
```

C++

```
__property HWnd ParentWindow;
```

Description

ParentWindow refers to the window handle that underlies the parent control. To designate a non-VCL control as a parent, assign that control's handle to ParentWindow. This assignment causes the control to be moved into the parent's screen area. Setting ParentWindow has no effect if Parent (see page 5832) is not nil (Delphi) or NULL (C++).

TActiveXControl objects use ParentWindow to insert a control in an ActiveX container window. ParentWindow is set automatically when a control is constructed with a call to CreateParented (see page 6123) (Delphi) or the appropriate overloaded constructor (C++).

Use ParentWindow with windowed controls that are packaged in a DLL. This allows references to the control by applications based on other applications.

See Also

CreateParented (see page 6123)

CreateParentedControl (see page 6136)

Create (see page 6123)

Parent (see page 5832)

TActiveXControl (see page 1517)

32.1.28.7.26 TWinControl.Showing Property

Indicates whether the control is showing on the screen.

Pascal

```
property Showing: Boolean;
```

C++

```
__property Boolean Showing;
```

Description

Showing is used internally to optimize the allocation of Windows resources. Use Showing to determine when you must allocate resources that are required when the control is visible. When Showing is false, the control is not visible and you can delay the allocation of resources.

If the Visible (see page 5847) properties of a component and all the parents in its parent hierarchy are true, Showing is guaranteed to be true. If one of the parents containing the control has a Visible (see page 5847) property value of false, Showing may be either true or false.

Showing is a read-only property.

See Also

Visible (see page 5847)

32.1.28.7.27 TWinControl.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop (see page 6211) property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop (see page 6211) property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

32.1.28.7.28 TWinControl.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

32.1.28.7.29 TWinControl.UseDockManager Property

Specifies whether the docking manager is used in drag-and-dock operations.

Pascal

```
property UseDockManager: Boolean;
```

C++

```
__property Boolean UseDockManager;
```

Description

Use UseDockManager to get or set whether a docking manager is used in drag-and-dock operations.

The docking manager handles the positioning of docked controls and any painting of docking zones around those controls. If you do not use a docking manager, docked controls are aligned to the windowed control based on the closest edge to where they are released. When you do not use a docking manager, there is no region for a user to grab to automatically undock a docked control.

See Also

DockManager (see page 6202)

32.1.28.7.30 TWinControl.VisibleDockClientCount Property

Specifies the number of visible controls that are docked on the windowed control.

Pascal

```
property VisibleDockClientCount: Integer;
```

C++

```
__property int VisibleDockClientCount;
```

Description

Use VisibleDockClientCount to determine the number of controls in the DockClients (see page 6201) list that have a Visible (see page 5847) property value of true. This value is less than or equal to the value of DockClientCount (see page 6199).

See Also

DockClientCount (see page 6199)

DockClients (see page 6201)

Visible (see page 5847)

32.1.28.7.31 TWinControl.WindowHandle Property

Provides access to a window handle for the control.

Pascal

```
property WindowHandle: HWnd;
```

C++

```
__property HWnd WindowHandle;
```

Description

The WindowHandle property provides access to the same window handle as the Handle (see page 6207) property, but WindowHandle is protected, and therefore only accessible to code inside the control.

The advantage to using WindowHandle is that it can be written to, while the Handle (see page 6207) property is read-only. Use WindowHandle when implementing methods that need to change the value of the control's window handle. Unlike the Handle (see page 6207) property, reading the value of WindowHandle doesn't automatically create a valid handle. Reading WindowHandle can return a zero value.

See Also

Handle (see page 6207)

HandleNeeded (see page 6160)

32.1.28.8 TWinControl Events

The events of the TWinControl class are listed here.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

32.1.28.8.1 TWinControl.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore: TAlignInsertBeforeEvent;
```

C++

```
__property TAlignInsertBeforeEvent OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align (see page 5798) property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent (see page 6293) type.

CustomAlignInsertBefore (see page 6137) triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore (see page 6137) returns the value returned by OnAlignInsertBefore. Defining this event allows users

to define the order of control alignment without overriding CustomAlignInsertBefore (see page 6137).

See Also

CustomAlignInsertBefore (see page 6137)

Align (see page 5798)

TAlignInsertBeforeEvent (see page 6293)

32.1.28.8.2 TWinControl.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition: TAlignPositionEvent;
```

C++

```
__property TAlignPositionEvent OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align (see page 5798) property of **alCustom** are aligned.

CustomAlignPosition (see page 6138) triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition (see page 6138) uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition (see page 6138). It holds a TAlignPositionEvent (see page 6293) type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors (see page 5802) TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo (see page 6269).
As implemented in TWinControl (see page 6112), OnAlignPosition does nothing.	

See Also

CustomAlignPosition (see page 6138)

Align (see page 5798)

Anchors (see page 5802)

TAlignInfo (see page 6269)

TAlignPositionEvent (see page 6293)

32.1.28.8.3 TWinControl.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop: TDockDropEvent;
```

C++

```
__property TDockDropEvent OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite (see page 6204) is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockOver (see page 6216)

32.1.28.8.4 TWinControl.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver: TDockOverEvent;
```

C++

```
__property TDockOverEvent OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject (see page 6009) has been initialized by an OnGetSiteInfo (see page 6223) event.

OnDockOver occurs only if DockSite (see page 6204) is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
```

```
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnGetSiteInfo (see page 6223)

TDragDockObject (see page 6009)

32.1.28.8.5 TWinControl.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit (see page 6221) event of the container occurs after the OnExit (see page 6221) event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit (see page 6221) event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit (see page 6221) event for the radio button occurs followed by an OnExit (see page 6221) event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
```

```
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

32.1.28.8.6 TWinControl.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter (see page 6219) event of the container occurs before the OnEnter (see page 6219) event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter (see page 6219) event on the group box, and finally an OnEnter (see page 6219) event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter (see page 6219) event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

32.1.28.8.7 TWinControl.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo: TGetSiteInfoEvent;
```

C++

```
__property TGetSiteInfoEvent OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject (see page 6009) with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver (see page 6216) event.

OnGetSiteInfo occurs only if DockSite (see page 6204) is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
```

```
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

TDragDockObject (see page 6009)

32.1.28.8.8 TWinControl.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent (see page 6311) type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress (see page 6227) event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress (see page 6227) event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
```

```
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
```

```
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

32.1.28.8.9 TWinControl.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown (see page 6225) or OnKeyUp (see page 6228) event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown (see page 6225) event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown (see page 6225) event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown (see page 6225) event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

32.1.28.8.10 TWinControl.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent (see page 6311) type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
```

```
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

32.1.28.8.11 TWinControl.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock: TUnDockEvent;
```

C++

```
__property TUnDockEvent OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
```

```
  undocked (if it is a dockable form). In addition, when the
  next to last docked control is undocked, the conjoinment
  docking site sends itself a close message so that the last
  docked control is undocked to its old position and size.
  */
  void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
        TWinControl *NewTarget, bool &Allow)
  {
    // Only 2 dock clients means the host must be destroyed and
    // the remaining window undocked to its old position and size.
    // Recall that OnUnDock gets called before the undocking
    // actually occurs.
    if (Client->ClassNameIs("TDockableForm"))
      dynamic_cast<TDockableForm *>(Client)->DockSite = True;
    if ((DockClientCount == 2) && (NewTarget != this))        // Self?
      PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
    UpdateCaption(Client);
  }
  /*
  The following example is taken from the docking demo. It
  shows how set permissions to dock dockable objects onto a
  docking site.
  */

  void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
        TDragDockObject *Source, int X, int Y)
  {
    //OnDockDrop gets called AFTER the client has actually docked,
    //so we check for DockClientCount = 1 before making the dock panel visible.
    TPanel *panel = dynamic_cast<TPanel *>(Sender);
    if (panel->DockClientCount == 1)
      ShowDockPanel(panel, True, NULL);
    panel->DockManager->ResetBounds(True);
    //Make DockManager repaints it's clients.
  }

  void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
        TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
  {
    Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
    if (Accept)
    {
      // Modify the DockRect to preview dock area.
      Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
      Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
        Point(this->ClientWidth/3, LeftDockPanel->Height));
      Source->DockRect = Types::TRect(TopLeft, BottomRight);
    }
  }

  void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
        TControl *Client, TWinControl *NewTarget, bool &Allow)
  {
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    TPanel *panel = dynamic_cast<TPanel *>(Sender);
    if (panel->DockClientCount == 1)
      ShowDockPanel(panel, False, NULL);
  }

  void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
        TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
        bool &CanDock)
  {
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock = DockClient->ClassNameIs("TDockableForm");
```

```
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
```

```
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

32.1.29 TWinControlActionLink Class

TWinControlActionLink is an action link used by descendants of TWinControl (see page 6112).

Class Hierarchy

System::TObject → Classes.TBasicActionLink → ActnList.TActionLink → Controls.TControlActionLink → Controls.TWinControlActionLink

Pascal

```
TWinControlActionLink = class(TControlActionLink);
```

C++

```
class TWinControlActionLink : public TControlActionLink;
```

File

Controls

Description

TWinControlActionLink is designed to link actions to controls based on TWinControl (see page 6112). TWinControlActionLink inherits support for linking actions with the properties of components and controls and adds support for linking the help context.

32.1.29.1 TWinControlActionLink Members

The following tables list the members exposed by TWinControlActionLink.

Protected Fields

	Name	Description
	FAction (see page 2722)	This is FAction, a member of class TBasicActionLink.
	FClient (see page 6236)	This is FClient, a member of class TWinControlActionLink.

Protected Methods

	Name	Description
V	AssignClient (see page 6236)	Specifies the client control.
V	Change (see page 2723)	This is Change, a member of class TBasicActionLink.

	DoShowHint (see page 5896)	Indicates whether the hint is shown, and modified.
	IsCaptionLinked (see page 268)	Indicates whether the Caption properties for the action and client are linked.
	IsCaptionLinked (see page 5896)	Indicates whether the caption for the action and the client are linked.
	IsCheckedLinked (see page 269)	Indicates whether the Checked properties for the action and client are linked.
	IsDropdownMenuLinked (see page 5896)	This is IsDropdownMenuLinked, a member of class TControlActionLink.
	IsEnabledLinked (see page 269)	Indicates whether the Enabled properties for the action and client are linked.
	IsEnabledLinked (see page 5897)	Indicates whether the enabled state for the action and the client are linked.
	IsEnableDropdownLinked (see page 5897)	This is IsEnableDropdownLinked, a member of class TControlActionLink.
	IsGroupIndexLinked (see page 270)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsHelpContextLinked (see page 6236)	Indicates whether the Help context for the action and the client are linked.
	IsHelpLinked (see page 271)	Indicates whether the help properties for the action and client are linked.
	IsHelpLinked (see page 5897)	Indicates whether the help properties for the action and the client are linked.
	IsHintLinked (see page 271)	Indicates whether the Hint properties for the action and client are linked.
	IsHintLinked (see page 5898)	Indicates whether the Hint property for the action and the client are linked.
	IsImageIndexLinked (see page 272)	Indicates whether the image index for the action and the client are linked.
	IsOnExecuteLinked (see page 2723)	This is IsOnExecuteLinked, a member of class TBasicActionLink.
	IsOnExecuteLinked (see page 5898)	Indicates whether the OnExecute event handler for the action and the client are linked.
	IsPopupMenuLinked (see page 5898)	This is IsPopupMenuLinked, a member of class TControlActionLink.
	IsShortCutLinked (see page 272)	Indicates whether the ShortCut properties for the action and client are linked.
	IsVisibleLinked (see page 273)	Indicates whether the Visible properties for the action and client are linked.
	IsVisibleLinked (see page 5899)	Indicates whether the visible state for the action and the client are linked.
	SetAction (see page 2724)	This is SetAction, a member of class TBasicActionLink.
	SetAutoCheck (see page 274)	Introduces an interface for setting the AutoCheck property of a client.
	SetCaption (see page 274)	Introduces an interface for setting the caption on a client.
	SetCaption (see page 5899)	Sets the caption of a control if the client is linked to the action for this event.
	SetChecked (see page 274)	Introduces an interface for setting the checked state of a client.
	SetDropdownMenu (see page 5899)	This is SetDropdownMenu, a member of class TControlActionLink.
	SetEnabled (see page 275)	Introduces an interface for setting the enabled state of a client.
	SetEnabled (see page 5900)	Sets the enabled state of a control if the client is linked to the action for this event.
	SetEnableDropdown (see page 5900)	This is SetEnableDropdown, a member of class TControlActionLink.
	SetGroupIndex (see page 275)	Introduces an interface for setting the GroupIndex property of a client.
	SetHelpContext (see page 6237)	Sets the Help context of a windowed control if the client is linked to the action for this event.
	SetHelpKeyword (see page 276)	Introduces an interface for setting the help keyword of a client.
	SetHelpKeyword (see page 5901)	Sets the HelpKeyword of a control if the client is linked to the action help properties.
	SetHelpType (see page 277)	Introduces an interface for setting the HelpType property of a client.
	SetHelpType (see page 5901)	Sets the HelpType of a control if the client is linked to the action help properties.
	SetHint (see page 278)	Introduces an interface for setting the hint for a client.
	SetHint (see page 5901)	Sets the hint of a control if the client is linked to the action for this event.
	SetImageIndex (see page 278)	Introduces an interface for setting the image index for a client.
	SetOnExecute (see page 2724)	This is SetOnExecute, a member of class TBasicActionLink.
	SetOnExecute (see page 5902)	Sets the execution event handler for a control if the client is linked to the action for this event.
	SetPopupMenu (see page 5902)	This is SetPopupMenu, a member of class TControlActionLink.
	SetShortCut (see page 279)	Introduces an interface for setting the shortcut for a client.
	SetVisible (see page 279)	Introduces an interface for setting the visible state of a client.
	SetVisible (see page 5902)	Sets the visible state of a control if the client is linked to the action for this event.

Public Events

	Name	Description
	OnChange (see page 2725)	Occurs when an OnChange event is triggered on the action.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2721)	Instantiates and initializes a TBasicActionLink (see page 2720) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2722)	Destroys an instance of an action link object.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 2723)	Invokes the action's Execute method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Update (see page 2724)	Invokes the action's Update method.

Public Properties

	Name	Description
	Action (see page 2725)	Indicates which action the link connects to its client.

32.1.29.2 TWinControlActionLink Fields

The fields of the TWinControlActionLink class are listed here.

Protected Fields

	Name	Description
	FClient (see page 6236)	This is FClient, a member of class TWinControlActionLink.

32.1.29.2.1 TWinControlActionLink.FClient Field

Pascal

```
FClient: TWinControl;
```

C++

```
TWinControl * FClient;
```

Description

This is FClient, a member of class TWinControlActionLink.

32.1.29.3 TWinControlActionLink Methods

The methods of the TWinControlActionLink class are listed here.

Protected Methods

	Name	Description
	AssignClient (see page 6236)	Specifies the client control.
	IsHelpContextLinked (see page 6236)	Indicates whether the Help context for the action and the client are linked.
	SetHelpContext (see page 6237)	Sets the Help context of a windowed control if the client is linked to the action for this event.

32.1.29.3.1 TWinControlActionLink.AssignClient Method

Specifies the client control.

Pascal

```
procedure AssignClient(AClient: TObject); override;
```

C++

```
virtual __fastcall AssignClient(TObject * AClient);
```

Description

TWinControlActionLink (see page 6233) overrides AssignClient to ensure that the client is a TWinControl (see page 6112) object.

32.1.29.3.2 TWinControlActionLink.IsHelpContextLinked Method

Indicates whether the Help context for the action and the client are linked.

Pascal

```
function IsHelpContextLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsHelpContextLinked();
```

Description

IsHelpContextLinked indicates whether HelpContext property is linked to the appropriate client property. IsHelpContextLinked returns true if the client requests that the action control this property, false otherwise.

See Also

HelpContext (see page 16148)

HelpContext (see page 5826)

HelpContext (see page 311)

32.1.29.3.3 TWinControlActionLink.SetHelpContext Method

Sets the Help context of a windowed control if the client is linked to the action for this event.

Pascal

```
procedure SetHelpContext(Value: THelpContext); override;
```

C++

```
virtual __fastcall SetHelpContext(THelpContext Value);
```

Description

SetHelpContext sets the help context of a client windowed control, and is called only if the client is linked for this event. Value specifies whether what the help context should be.

See Also

IsHelpContextLinked (see page 6236)

32.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IDockManager (see page 6238)	IDockManager is the interface for docking managers.

32.2.1 IDockManager Interface

IDockManager is the interface for docking managers.

Class Hierarchy

Controls.IDockManager

Pascal

```
IDockManager = interface;
```

C++

```
__interface IDockManager;
```

File

Controls

Description

IDockManager handles the tasks of inserting and removing controls when they are docked to or undocked from a container control, including details such as positioning controls and any painting that is associated with docking operations. IDockManager is the type of a container control's DockManager property.

Each docking manager is associated with a single TWinControl (see page 6112) object. To use a docking manager, set the container control's UseDockManager property to true. An application can explicitly assign an object that implements the IDockManager interface as the value of a container control's DockManager property, or it can let the container control create its own docking manager.

Note: C++ declarations that use IDockManager use the _di_IDockManager type instead. This type is a DelphiInterface wrapper around the IDockManager interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IDockManager >  _di_IDockManager;
```

See Also

TWinControl (see page 6112)

TWinControl (see page 6112)

TDockTree (see page 5983)

32.2.1.1 IDockManager Members

The following tables list the members exposed by IDockManager.

Public Methods

	Name	Description
	BeginUpdate (see page 6239)	Responds when the docked controls are about to change.
	EndUpdate (see page 6240)	Responds when the docked controls have finished changing.
	GetControlBounds (see page 6240)	Returns the dimensions allotted to a control that is already docked.
	InsertControl (see page 6240)	Handles the docking of a control into the associated container control.
	LoadFromStream (see page 6241)	Lays out a set of docked controls read from a stream.
	PaintSite (see page 6241)	Paints the elements that surround all docked controls.
	PositionDockRect (see page 6242)	Computes the dimensions of a new docking zone.
	RemoveControl (see page 6242)	Handles the undocking of a control from the associated container control.
	ResetBounds (see page 6243)	Recomputes the layout of docked controls.
	SaveToStream (see page 6245)	Saves a list of the set of docked controls and their positions to a stream.
	SetReplacingControl (see page 6245)	This is SetReplacingControl, a member of class IDockManager.

32.2.1.2 IDockManager Methods

The methods of the IDockManager class are listed here.

Public Methods

	Name	Description
	BeginUpdate (see page 6239)	Responds when the docked controls are about to change.
	EndUpdate (see page 6240)	Responds when the docked controls have finished changing.
	GetControlBounds (see page 6240)	Returns the dimensions allotted to a control that is already docked.
	InsertControl (see page 6240)	Handles the docking of a control into the associated container control.
	LoadFromStream (see page 6241)	Lays out a set of docked controls read from a stream.
	PaintSite (see page 6241)	Paints the elements that surround all docked controls.
	PositionDockRect (see page 6242)	Computes the dimensions of a new docking zone.
	RemoveControl (see page 6242)	Handles the undocking of a control from the associated container control.
	ResetBounds (see page 6243)	Recomputes the layout of docked controls.
	SaveToStream (see page 6245)	Saves a list of the set of docked controls and their positions to a stream.
	SetReplacingControl (see page 6245)	This is SetReplacingControl, a member of class IDockManager.

32.2.1.2.1 IDockManager.BeginUpdate Method

Responds when the docked controls are about to change.

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

BeginUpdate is called before controls are docked or undocked.

Once all docking or undocking is complete, the EndUpdate (see page 6240) method is called so that the dock manager can make any final adjustments to reflect the final set of docked controls. Every call to BeginUpdate should be paired with a call to EndUpdate (see page 6240). BeginUpdate/EndUpdate (see page 6240) pairs can be nested.

See Also

EndUpdate (see page 6240)

32.2.1.2.2 IDockManager.EndUpdate Method

Responds when the docked controls have finished changing.

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

EndUpdate is called after a series of changes to the docked controls is completed. It checks that every previous call to the BeginUpdate (see page 6239) method has been matched with an EndUpdate call. If so, it performs any final adjustments to the docked controls, now that they are all in place.

Each call to EndUpdate should be paired by a call to BeginUpdate (see page 6239), which is made before docking or undocking controls. BeginUpdate (see page 6239)/EndUpdate pairs can be nested.

See Also

BeginUpdate (see page 6239)

32.2.1.2.3 IDockManager.GetControlBounds Method

Returns the dimensions allotted to a control that is already docked.

Pascal

```
procedure GetControlBounds(Control: TControl; out CtlBounds: TRect);
```

C++

```
__fastcall GetControlBounds(TControl * Control, TRect CtlBounds);
```

Description

GetControlBounds returns the dimensions of the zone in which a control is currently docked.

Control is the currently docked control.

CtlBounds returns the dimensions of the zone in which Control is docked. If Control does not specify an object that is currently docked in the container control that uses this dock manager, CtlBounds returns (0,0,0,0).

See Also

PositionDockRect (see page 6242)

ResetBounds (see page 6243)

32.2.1.2.4 IDockManager.InsertControl Method

Handles the docking of a control into the associated container control.

Pascal

```
procedure InsertControl(Control: TControl; InsertAt: TAlign; DropCtl: TControl);
```

C++

```
__fastcall InsertControl(TControl * Control, TAlign InsertAt, TControl * DropCtl);
```

Description

InsertControl is called immediately after the container control that uses the dock manager adds a new control to its list of docked controls.

Control is the newly docked control.

InsertAt indicates where Control was docked within the container control.

DropCtl is the container control in which Control was docked.

InsertControl handles any tasks the dock manager needs to manage the layout and appearance of docked controls, forcing a repaint of the container control if necessary.

See Also

RemoveControl (see page 6242)

32.2.1.2.5 IDockManager.LoadFromStream Method

Lays out a set of docked controls read from a stream.

Pascal

```
procedure LoadFromStream(Stream: TStream);
```

C++

```
__fastcall LoadFromStream(TStream Stream);
```

Description

LoadFromStream updates the dock manager to reflect the set of docked controls it reads from a stream. The stream must reflect the same format as is written by the SaveToStream (see page 6245) method.

Stream is the stream object to use for reading the docked controls.

See Also

SaveToStream (see page 6245)

32.2.1.2.6 IDockManager.PaintSite Method

Paints the elements that surround all docked controls.

Pascal

```
procedure PaintSite(DC: HDC);
```

C++

```
__fastcall PaintSite(HDC DC);
```

Description

Container controls that use a docking manager call PaintSite when they need to paint their docked child controls.

DC is the handle to the device context to use when rendering the visual elements that surround the docked controls.

PaintSite paints the visual elements associated with the docked controls other than the docked controls themselves.

See Also

PaintControls (see page 6166)

UseDockManager (see page 6212)

32.2.1.2.7 IDockManager.PositionDockRect Method

Computes the dimensions of a new docking zone.

Pascal

```
procedure PositionDockRect(Client: TControl; DropCtl: TControl; DropAlign: TAlign; var
DockRect: TRect);
```

C++

```
__fastcall PositionDockRect(TControl * Client, TControl * DropCtl, TAlign DropAlign, TRect
DockRect);
```

Description

The control that is being dragged calls PositionDockRect from its own PositionDockRect method when it is being docked to a container control that uses a dock manager. It allows the dock manager to indicate where a specified control would appear if the user released the mouse.

Client is the control that is being dragged to a docking position.

DropCtl is the container control into which Client will be docked.

DropAlign indicates where Control is to be docked within the container control.

DockRect returns the boundaries of the docking zone where Control can be docked.

See Also

PositionDockRect (see page 5777)

UseDockManager (see page 6212)

32.2.1.2.8 IDockManager.RemoveControl Method

Handles the undocking of a control from the associated container control.

Pascal

```
procedure RemoveControl(Control: TControl);
```

C++

```
__fastcall RemoveControl(TControl * Control);
```

Description

The container control that uses the dock manager calls RemoveControl when one of its docked controls is undocked.

Control is the control that was just undocked.

RemoveControl performs any internal adjustments that are needed now that the control is no longer docked, forcing a repaint of the container control if necessary.

See Also

InsertControl (see page 6240)

32.2.1.2.9 IDockManager.ResetBounds Method

Recomputes the layout of docked controls.

Pascal

```
procedure ResetBounds(Force: Boolean);
```

C++

```
__fastcall ResetBounds(Boolean Force);
```

Description

ResetBounds updates the layout of the docked controls and their surrounding docking zones, based on the current dimensions of the associated container control. It is used to update the dock manager to reflect current conditions (for example, when the container control that uses the dock manager is resized).

Force indicates whether ResetBounds should recompute the layout if the dimensions of the associated container control have not changed. When Force is false, ResetBounds only updates the layout of docked controls if the dimensions of DockSite have changed. When Force is true, ResetBounds updates the layout of docked controls even if the dimensions of the associated container control have not changed.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
```

```
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

32.2.1.2.10 IDockManager.SaveToStream Method

Saves a list of the set of docked controls and their positions to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream);
```

C++

```
__fastcall SaveToStream(TStream Stream);
```

Description

SaveToStream writes enough information to a stream that the dock manager can reconstruct the docking zone structure when it reads that information back at a later time.

Stream is the stream object to use for writing the information.

Note: To reconstruct the docking structure at a later time, use the LoadFromStream (see page 6241) method.

See Also

LoadFromStream (see page 6241)

32.2.1.2.11 IDockManager.SetReplacingControl Method

Pascal

```
procedure SetReplacingControl(Control: TControl);
```

C++

```
__fastcall SetReplacingControl(TControl * Control);
```

Description

This is SetReplacingControl, a member of class IDockManager.

32.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CancelDrag (see page 6247)	Cancels a drag-and-drop or drag-and-dock operation that is in progress.
	ChangeBiDiModeAlignment (see page 6247)	This is function Controls.ChangeBiDiModeAlignment.
	CursorToIdent (see page 6247)	Returns the name of the predefined constant that corresponds to a cursor.
	CursorToString (see page 6248)	Returns the AnsiString representation of a cursor.
	DragDone (see page 6249)	This is function Controls.DragDone.
	FindControl (see page 6249)	Returns the windowed control (Windows) or widget (cross-platform) whose window is identified by the specifiedhandle or which appears at a specified position.
	FindDragTarget (see page 6249)	Returns the control at a specified screen coordinate.
	FindVCLWindow (see page 6250)	Returns the windowed control, if any, at the specified location.
	GetCaptureControl (see page 6251)	Returns the control, if any, that currently receives all mouse messages.
	GetCursorValues (see page 6251)	Passes the name of every predefined cursor constant to a callback.
	GetLongHint (see page 6252)	Returns the second part of a two-part hint string.
	GetShortHint (see page 6253)	Returns the first part of a two-part hint string.
	IdentToCursor (see page 6254)	Translates a predefined cursor constant into a cursor.
	Imm32GetCompositionString (see page 6254)	This is function Controls.Imm32GetCompositionString.
	Imm32GetContext (see page 6255)	This is function Controls.Imm32GetContext.
	Imm32GetConversionStatus (see page 6255)	This is function Controls.Imm32GetConversionStatus.
	Imm32IsIME (see page 6255)	This is function Controls.Imm32IsIME.
	Imm32NotifyIME (see page 6256)	This is function Controls.Imm32NotifyIME.
	Imm32ReleaseContext (see page 6256)	This is function Controls.Imm32ReleaseContext.
	Imm32SetCompositionFont (see page 6256)	This is function Controls.Imm32SetCompositionFont.
	Imm32SetCompositionWindow (see page 6257)	This is function Controls.Imm32SetCompositionWindow.
	Imm32SetConversionStatus (see page 6257)	This is function Controls.Imm32SetConversionStatus.
	Imm32SetOpenStatus (see page 6257)	This is function Controls.Imm32SetOpenStatus.
	InitWndProc (see page 6258)	This is function Controls.InitWndProc.
	IsAbortResult (see page 6258)	Checks the return value from a modal form dialog and indicates whether the user selected Abort or Cancel.
	IsAnAllResult (see page 6259)	Checks the return value from a modal form dialog and indicates whether the user selected All, Yes to All, or No to All.
	IsDragObject (see page 6259)	Indicates whether a specified object is a TDragObject (see page 6031) descendant.
	IsNegativeResult (see page 6260)	Checks the return value from a modal form dialog and indicates whether the user selected No or No to All.
	IsPositiveResult (see page 6260)	Checks the return value from a modal form dialog and indicates whether the user selected Ok, Yes, Yes to All, or All.
	IsVCLControl (see page 6261)	IsVCLControl returns true if a control is a VCL control.
	MoveWindowOrg (see page 6262)	Changes the location of the point (0,0) within a device context.
	PerformEraseBackground (see page 6262)	This is function Controls.PerformEraseBackground.
	SendAppMessage (see page 6262)	Sends a Windows message to the application.
	SetCaptureControl (see page 6263)	Causes the specified control to receive all mouse messages.
	SetImeMode (see page 6264)	This is function Controls.SetImeMode.
	SetImeName (see page 6264)	This is function Controls.SetImeName.
	StringToCursor (see page 6264)	Returns a TCursor (see page 6305) value given its string representation.
	StripAllFromResult (see page 6265)	Converts a TModalResult (see page 6312) value from a constant that refers to "all" to the corresponding simple constant.
	Win32NLSEnableIME (see page 6266)	This is function Controls.Win32NLSEnableIME.

32.3.1 Controls.CancelDrag Function

Cancels a drag-and-drop or drag-and-dock operation that is in progress.

Pascal

```
procedure CancelDrag;
```

C++

```
CancelDrag();
```

File

Controls

Description

Call CancelDrag to terminate the current drag-and-drop or drag-and-dock operation without dropping or docking the dragged object.

32.3.2 Controls.ChangeBiDiModeAlignment Function

Pascal

```
procedure ChangeBiDiModeAlignment(var Alignment: TAlignment);
```

C++

```
ChangeBiDiModeAlignment(TAlignment Alignment);
```

File

Controls

Description

This is function Controls.ChangeBiDiModeAlignment.

32.3.3 Controls.CursorToIdent Function

Returns the name of the predefined constant that corresponds to a cursor.

Pascal

```
function CursorToIdent(Cursor: Longint; var Ident: string): Boolean;
```

C++

```
Boolean CursorToIdent(Longint Cursor, AnsiString Ident);
```

File

Controls

Description

Use CursorToIdent to convert the cursor used by a control or one of the cursors made available by the global screen variable into the name of the corresponding constant. Pass the cursor value as the Cursor parameter. CursorToIdent returns the constant

name as the Ident parameter. If the cursor does not have an associated constant (for example, a custom cursor), CursorToIdent returns false.

To obtain a string representation of the cursor, even if there is no built-in cursor constant for it, use the CursorToString (see page 6248) function.

CursorToIdent can be used to identify the current cursor in a list of possible cursor constants that is built using GetCursorValues (see page 6251).

See Also

TControl (see page 5724)

CursorToString (see page 6248)

IdentToCursor (see page 6254)

TScreen

GetCursorValues (see page 6251)

32.3.4 Controls.CursorToString Function

Returns the AnsiString representation of a cursor.

Pascal

```
function CursorToString(Cursor: TCursor): string;
```

C++

```
AnsiString CursorToString(TCursor Cursor);
```

File

Controls

Description

Use CursorToString to convert the cursor used by a control or one of the cursors made available by the global screen variable into a string that can be displayed to the user. Pass the cursor value as the Cursor parameter. If there is a predefined constant for the cursor, CursorToString returns the constant name. If the cursor does not have an associated constant (for example, a custom cursor), CursorToString returns a string representation of the cursor's numeric value.

To obtain a string representation of the cursor only if there is a built-in cursor constant, use CursorToIdent (see page 6247).

See Also

TControl (see page 5724)

CursorToIdent (see page 6247)

StringToCursor (see page 6264)

TScreen

32.3.5 Controls.DragDone Function

Pascal

```
procedure DragDone(Drop: Boolean);
```

C++

```
DragDone(Boolean Drop);
```

File

Controls

Description

This is function Controls.DragDone.

32.3.6 Controls.FindControl Function

Returns the windowed control (Windows) or widget (cross-platform) whose window is identified by the specifiedhandle or which appears at a specified position.

Pascal

```
function FindControl(Handle: HWnd): TWinControl;
```

C++

```
TWinControl * FindControl(HWnd Handle);
```

File

Controls

Description

Call FindControl to access the control whose underlying screen object is identified by the Handle parameter or which appears at the position specified by the Pos parameter. If Handle is not the handle of a screen object or if there is no widget at the specified position, FindControl returns nil (Delphi) or NULL (C++).

See Also

TWinControl (see page 6112)

FindVCLWindow (see page 6250)

TWidgetControl

32.3.7 Controls.FindDragTarget Function

Returns the control at a specified screen coordinate.

Pascal

```
function FindDragTarget(const Pos: TPoint; AllowDisabled: Boolean): TControl;
```

C++

```
TControl * FindDragTarget(const TPoint Pos, Boolean AllowDisabled);
```

File

Controls

Description

Call FindDragTarget to determine the potential drag-and-drop or drag-and-dock target currently under the mouse. FindDragTarget returns the control at the position specified by the Pos parameter. If there is not control at the specified position, FindDragTarget returns nil (Delphi) or NULL (C++). The AllowDisabled parameter determines whether the search includes disabled controls.

If FindDragTarget returns a control, this does not mean that the control will necessarily accept the dragged object. The control will only accept the dragged object if it has an OnDragOver or OnDockOver event handler that accepts the object or if its DockSite property is true.

See Also

TControl (see page 5724)

TWinControl (see page 6112)

TWincontrol

32.3.8 Controls.FindVCLWindow Function

Returns the windowed control, if any, at the specified location.

Pascal

```
function FindVCLWindow(const Pos: TPoint): TWinControl;
```

C++

```
TWinControl * FindVCLWindow(const TPoint Pos);
```

File

Controls

Description

Call FindVCLWindow to locate the windowed control under a certain point. For example, use FindVCLWindow to identify the windowed control that is under the mouse from another control that has captured the mouse. The Pos parameter specifies the location that must be over the returned windowed control. If there is no windowed control under the Pos parameter, FindVCLWindow returns nil (Delphi) or NULL (C++).

See Also

FindControl (see page 6249)

SetCaptureControl (see page 6263)

TControl (see page 5724)

32.3.9 Controls.GetCaptureControl Function

Returns the control, if any, that currently receives all mouse messages.

Pascal

```
function GetCaptureControl: TControl;
```

C++

```
TControl * GetCaptureControl();
```

File

Controls

Description

Call GetCaptureControl to determine which control is currently processing mouse messages. When a control has captured the mouse, it receives all mouse messages, even if the mouse is not over the control.

GetCaptureControl is equivalent to GetMouseGrabControl.

See Also

SetCaptureControl (see page 6263)

TControl (see page 5724)

TControl (see page 5724)

TControl (see page 5724)

GetMouseGrabControl

32.3.10 Controls.GetCursorValues Function

Passes the name of every predefined cursor constant to a callback.

Pascal

```
procedure GetCursorValues(Proc: TGetStrProc);
```

C++

```
GetCursorValues(TGetStrProc Proc);
```

File

Controls

Description

Call GetCursorValues to execute a callback for every built-in cursor constant in the VCL. The Proc parameter is the callback function that is called for every cursor constant name.

Note: To convert the strings that are passed to the callback into actual cursors, use IdentToCursor (see page 6254). To convert a cursor into a constant name that can be compared with the string passed to the callback, use CursorToIdent (see page 6247).

See Also

CursorToIdent (see page 6247)

IdentToCursor (see page 6254)

32.3.11 Controls.GetLongHint Function

Returns the second part of a two-part hint string.

Pascal

```
function GetLongHint(const Hint: string): string;
```

C++

```
AnsiString GetLongHint(const AnsiString Hint);
```

File

Controls

Description

Call GetLongHint to obtain the long version of the hint specified as the value of a Hint property. The second part of the hint is the text following the | character. If the Hint string value is not separated into two parts, GetLongHint returns the entire string.

C++ Examples:

```
/*
Note:   It is not actually necessary to write an OnHint
event handler to display hints on the status bar. This can
be accomplished more simply by setting the status bar's
AutoHint property and SimplePanel property.
This shows how the OnHint event handler is declared as a
public method of the form.
*/

/*
Here is the code to add to the unit file.  It provides the
implementation of the DisplayHint method and assigns it to
the Application in the form's OnCreate event handler.
Note:   It is not actually necessary to write an OnHint event
handler to display hints on the status bar. This can be
accomplished more simply by setting the status bar's AutoHint
property.
*/
void __fastcall TForm1::DisplayHint(TObject *Sender)
{
  StatusBar1->SimpleText = GetLongHint(Application->Hint);
}

// Here is the form's OnCreate event handler.
// It assigns the application's OnHint event handler at
// runtime because the Application is not available in the
// Object Inspector
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnHint = DisplayHint;
}
```

Delphi Examples:

```
{
```

```
Note:   It is not actually necessary to write an OnHint
event handler to display hints on the status bar. This can
be accomplished more simply by setting the status bar's
AutoHint property and SimplePanel property.
This shows how the OnHint event handler is declared as a
public method of the form.
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    StatusBar1: TStatusBar;
    Edit1: TEdit;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure DisplayHint(Sender: TObject);
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}

{ Here is the implementation of the OnHint event handler }
{ It displays the application's current hint in the status bar }
procedure TForm1.DisplayHint(Sender: TObject);
begin
  StatusBar1.SimpleText := GetLongHint(Application.Hint);
end;
{ Here is the form's OnCreate event handler. }
{ It assign's the application's OnHint event handler at runtime }
{ because the Application is not available in the Object Inspector }
{ at design time }
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnHint := DisplayHint;
end;
```

See Also

GetShortHint (see page 6253)

TApplication

TApplication

TControl (see page 5724)

32.3.12 Controls.GetShortHint Function

Returns the first part of a two-part hint string.

Pascal

```
function GetShortHint(const Hint: string): string;
```

C++

```
AnsiString GetShortHint(const AnsiString Hint);
```

File

Controls

Description

Call GetShortHint to obtain the short version of the hint specified as the value of a Hint property. The first part of the hint is the text preceding the | character. If the Hint string value is not separated into two parts, GetShortHint returns the entire string.

See Also

GetLongHint (see page 6252)

TApplication

TApplication

TControl (see page 5724)

32.3.13 Controls.IdentToCursor Function

Translates a predefined cursor constant into a cursor.

Pascal

```
function IdentToCursor(const Ident: string; var Cursor: Longint): Boolean;
```

C++

```
Boolean IdentToCursor(const AnsiString Ident, Longint Cursor);
```

File

Controls

Description

Call IdentToCursor to reverse the translation performed by the CursorToIdent (see page 6247) function. This method is useful for converting the strings generated by GetCursorValues (see page 6251) into useable cursors. IdentToCursor returns true if the Ident parameter is successfully converted into a cursor. IdentToCursor returns false if the Ident parameter is not the name of one of the built-in cursor constants.

Note: If the string identifier for a cursor may represent either a numeric value or a cursor constant, use the StringToCursor (see page 6264) function instead.

See Also

TControl (see page 5724)

CursorToIdent (see page 6247)

StringToCursor (see page 6264)

GetCursorValues (see page 6251)

32.3.14 Controls.Imm32GetCompositionString Function

Pascal

```
function Imm32GetCompositionString(hImc: HIMC; dWord1: DWORD; lpBuf: pointer; dwBufLen:
DWORD): Longint;
```

C++

```
Longint Imm32GetCompositionString(HIMC hImc, DWORD dWord1, void * lpBuf, DWORD dwBufLen);
```

File

Controls

Description

This is function Controls.Imm32GetCompositionString.

32.3.15 Controls.Imm32GetContext Function

Pascal

```
function Imm32GetContext(hWnd: HWND): HIMC;
```

C++

```
HIMC Imm32GetContext(HWND hWnd);
```

File

Controls

Description

This is function Controls.Imm32GetContext.

32.3.16 Controls.Imm32GetConversionStatus Function

Pascal

```
function Imm32GetConversionStatus(hImc: HIMC; var Conversion: DWORD; var Sentence: DWORD):
Boolean;
```

C++

```
Boolean Imm32GetConversionStatus(HIMC hImc, DWORD Conversion, DWORD Sentence);
```

File

Controls

Description

This is function Controls.Imm32GetConversionStatus.

32.3.17 Controls.Imm32IsIME Function

Pascal

```
function Imm32IsIME(hKl: HKL): Boolean;
```

C++

```
Boolean Imm32IsIME(HKL hKl);
```

File

Controls

Description

This is function Controls.Imm32IsIME.

32.3.18 Controls.Imm32NotifyIME Function

Pascal

```
function Imm32NotifyIME(hImc: HIMC; dwAction: DWORD; dwIndex: DWORD; dwValue: DWORD): Boolean;
```

C++

```
Boolean Imm32NotifyIME(HIMC hImc, DWORD dwAction, DWORD dwIndex, DWORD dwValue);
```

File

Controls

Description

This is function Controls.Imm32NotifyIME.

32.3.19 Controls.Imm32ReleaseContext Function

Pascal

```
function Imm32ReleaseContext(hWnd: HWND; hImc: HIMC): Boolean;
```

C++

```
Boolean Imm32ReleaseContext(HWND hWnd, HIMC hImc);
```

File

Controls

Description

This is function Controls.Imm32ReleaseContext.

32.3.20 Controls.Imm32SetCompositionFont Function

Pascal

```
function Imm32SetCompositionFont(hImc: HIMC; lpLogfont: PLOGFONTA): Boolean;
```

C++

```
Boolean Imm32SetCompositionFont(HIMC hImc, PLOGFONTA lpLogfont);
```

File

Controls

Description

This is function Controls.Imm32SetCompositionFont.

32.3.21 Controls.Imm32SetCompositionWindow Function

Pascal

```
function Imm32SetCompositionWindow(hImc: HIMC; lpCompForm: PCOMPOSITIONFORM): Boolean;
```

C++

```
Boolean Imm32SetCompositionWindow(HIMC hImc, PCOMPOSITIONFORM lpCompForm);
```

File

Controls

Description

This is function Controls.Imm32SetCompositionWindow.

32.3.22 Controls.Imm32SetConversionStatus Function

Pascal

```
function Imm32SetConversionStatus(hImc: HIMC; Conversion: DWORD; Sentence: DWORD): Boolean;
```

C++

```
Boolean Imm32SetConversionStatus(HIMC hImc, DWORD Conversion, DWORD Sentence);
```

File

Controls

Description

This is function Controls.Imm32SetConversionStatus.

32.3.23 Controls.Imm32SetOpenStatus Function

Pascal

```
function Imm32SetOpenStatus(hImc: HIMC; fOpen: Boolean): Boolean;
```

C++

```
Boolean Imm32SetOpenStatus(HIMC hImc, Boolean fOpen);
```

File

Controls

Description

This is function Controls.Imm32SetOpenStatus.

32.3.24 Controls.InitWndProc Function

Pascal

```
function InitWndProc(HWindow: HWnd; Message: Longint; WParam: Longint; LParam: Longint):
Longint; stdcall;
```

C++

```
__stdcall Longint InitWndProc(HWnd HWindow, Longint Message, Longint WParam, Longint LParam);
```

File

Controls

Description

This is function Controls.InitWndProc.

32.3.25 Controls.IsAbortResult Function

Checks the return value from a modal form dialog and indicates whether the user selected Abort or Cancel.

Pascal

```
function IsAbortResult(const AModalResult: TModalResult): Boolean;
```

C++

```
Boolean IsAbortResult(const TModalResult AModalResult);
```

File

Controls

Description

Call IsAbortResult to check whether the user clicked "Abort" or "Cancel" to close a modal form. Pass the return value from the form's ShowModal method as the AModalResult argument. IsAbortResult returns true if AModalResult is mrAbort (see page 6372) or mrCancel (see page 6373), false otherwise.

See Also

TCustomForm

TButton

IsAnALlResult

IsNegativeResult (see page 6260)

IsPositiveResult (see page 6260)

StripAllFromResult (see page 6265)

TCustomForm

32.3.26 Controls.IsAnAllResult Function

Checks the return value from a modal form dialog and indicates whether the user selected All, Yes to All, or No to All.

Pascal

```
function IsAnAllResult(const AModalResult: TModalResult): Boolean;
```

C++

```
Boolean IsAnAllResult(const TModalResult AModalResult);
```

File

Controls

Description

Call IsAnAllResult to check whether the user closed a modal form by indicating a response that applies to "all". Pass the return value from the form's ShowModal method as the AModalResult argument. IsAnAllResult returns true if AModalResult is mrAll (see page 6372), mrNoToAll (see page 6375), or mrYesToAll (see page 6376), false otherwise.

Note: If IsAnAllResult returns

true, you can convert the result to a simple Ok, No, or Yes result by calling StripAllFromResult (see page 6265).

See Also

TCustomForm

TButton

IsAbortResult (see page 6258)

IsNegativeResult (see page 6260)

IsPositiveResult (see page 6260)

StripAllFromResult (see page 6265)

TCustomForm

32.3.27 Controls.IsDragObject Function

Indicates whether a specified object is a TDragObject (see page 6031) descendant.

Pascal

```
function IsDragObject(Sender: TObject): Boolean;
```

C++

```
Boolean IsDragObject(TObject * Sender);
```

File

Controls

Description

Use IsDragObject to determine whether the object being dragged in a drag-and-drop or drag-and-dock operation is a specialized TDragObject (see page 6031) that represents the dragged control's data rather then the control itself. Use IsDragObject on the

Source parameter in an OnDragOver or OnDockOver event handler to determine the type of object being dragged and whether it should be accepted. Use IsDragObject on the Source parameter of an OnDragDrop or OnDockDrop event handler to determine how to interpret the dragged object's data.

See Also

TDragObject (see page 6031)

TControl (see page 5724)

TControl (see page 5724)

TWinControl (see page 6112)

TWinControl (see page 6112)

Customizing Drag and Drop with a Drag Object

32.3.28 Controls.IsNegativeResult Function

Checks the return value from a modal form dialog and indicates whether the user selected No or No to All.

Pascal

```
function IsNegativeResult(const AModalResult: TModalResult): Boolean;
```

C++

```
Boolean IsNegativeResult(const TModalResult AModalResult);
```

File

Controls

Description

Call IsNegativeResult to check whether the user closed a modal form by indicating a "No" response. Pass the return value from the form's ShowModal method as the AModalResult argument. IsNegativeResult returns true if AModalResult is mrNo (see page 6374) or mrNoToAll (see page 6375), false otherwise.

See Also

TCustomForm

TButton

IsAbortResult (see page 6258)

IsAnAllResult (see page 6259)

IsPositiveResult (see page 6260)

StripAllFromResult (see page 6265)

TCustomForm

32.3.29 Controls.IsPositiveResult Function

Checks the return value from a modal form dialog and indicates whether the user selected Ok, Yes, Yes to All, or All.

Pascal

```
function IsPositiveResult(const AModalResult: TModalResult): Boolean;
```

C++

```
Boolean IsPositiveResult(const TModalResult AModalResult);
```

File

Controls

Description

Call IsPositiveResult to check whether the user closed a modal form by indicating a "Yes" or "Ok" response. Pass the return value from the form's ShowModal method as the AModalResult argument. IsPositiveResult returns true if AModalResult is mrYes (see page 6376), mrOk (see page 6375), mrYesToAll (see page 6376) or mrAll (see page 6372), false otherwise.

See Also

TCustomForm

TButton

IsAbortResult (see page 6258)

IsAnAllResult (see page 6259)

IsNegativeResult (see page 6260)

StripAllFromResult (see page 6265)

TCustomForm

32.3.30 Controls.IsVCLControl Function

IsVCLControl returns true if a control is a VCL control.

Pascal

```
function IsVCLControl(Handle: HWnd): Boolean;
```

C++

```
Boolean IsVCLControl(HWnd Handle);
```

File

Controls

Description

IsVCLControl returns true if the visual control specified by the Windows **Handle** is a VCL component. Otherwise, false is returned.

Use FindControl (see page 6249) to get a reference to the control via its handle.

See Also

FindControl (see page 6249)

32.3.31 Controls.MoveWindowOrg Function

Changes the location of the point (0,0) within a device context.

Pascal

```
procedure MoveWindowOrg(DC: HDC; DX: Integer; DY: Integer);
```

C++

```
MoveWindowOrg(HDC DC, int DX, int DY);
```

File

Controls

Description

Call MoveWindowOrg to change the location of where items are drawn on the canvas of an object. When drawing to a canvas, objects specify the location of the image they draw as a set of coordinates, which are offsets from the upper left corner. MoveWindowOrg allows the same drawing code to draw its image in a different location. The DC parameter is a handle to the device context of the canvas, printer, or window where the image is to be drawn. The DX and DY parameters specify the number of pixels to move the location of the (0,0) point on that drawing surface.

MoveWindowOrg is especially useful when playing metafiles, which draw to a particular position on a canvas, because it provides the ability to play the metafile to a different location on the canvas.

See Also

TCanvas

TPrinter

TWinControl (see page 6112)

32.3.32 Controls.PerformEraseBackground Function

Pascal

```
procedure PerformEraseBackground(Control: TControl; DC: HDC);
```

C++

```
PerformEraseBackground(TControl * Control, HDC DC);
```

File

Controls

Description

This is function Controls.PerformEraseBackground.

32.3.33 Controls.SendAppMessage Function

Sends a Windows message to the application.

Pascal

```
function SendAppMessage(Msg: Cardinal; WParam: Longint; LParam: Longint): Longint;
```

C++

```
Longint SendAppMessage(unsigned Msg, Longint WParam, Longint LParam);
```

File

Controls

Description

Call SendAppMessage to send a Windows message to the global application object.

Msg represents the Windows Message code.

WParam represents the WParam of the message. Its interpretation depends on the value of Msg.

LParam represents the LParam of the message. Its interpretation depends on the value of Msg.

SendAppMessage returns the result provided by the application's window procedure. If the application did not handle the message, SendAppMessage returns 0.

See Also

TApplication

TControl (see page 5724)

TApplication

TControl (see page 5724)

Application

TMessage

32.3.34 Controls.SetCaptureControl Function

Causes the specified control to receive all mouse messages.

Pascal

```
procedure SetCaptureControl(Control: TControl);
```

C++

```
SetCaptureControl(TControl * Control);
```

File

Controls

Description

Call SetCaptureControl to allow a control to capture the mouse. When the control has the mouse captured, it receives all mouse messages until another control captures the mouse.

SetCaptureControl is equivalent to SetMouseGrabControl.

Note: Many controls capture the mouse in response to mouse events. Application code should avoid explicitly capturing the mouse.

See Also

GetCaptureControl (see page 6251)

TControl (see page 5724)

TControl (see page 5724)

TControl (see page 5724)

SetMouseGrabControl

32.3.35 Controls.SetImeMode Function

Pascal

```
procedure SetImeMode(hWnd: HWND; Mode: TImeMode);
```

C++

```
SetImeMode(HWND hWnd, TImeMode Mode);
```

File

Controls

Description

This is function Controls.SetImeMode.

32.3.36 Controls.SetImeName Function

Pascal

```
procedure SetImeName(Name: TImeName);
```

C++

```
SetImeName(TImeName Name);
```

File

Controls

Description

This is function Controls.SetImeName.

32.3.37 Controls.StringToCursor Function

Returns a TCursor (see page 6305) value given its string representation.

Pascal

```
function StringToCursor(const S: string): TCursor;
```

C++

```
TCursor StringToCursor(const AnsiString S);
```

File

Controls

Description

Call StringToCursor to reverse the translation performed by CursorToString (see page 6248). This method is useful for converting the strings entered by the user into useable TCursor (see page 6305) values.

The S parameter can be either the name of a built-in cursor constant such as "crHelp (see page 6363)", or the string representation of a valid TCursor (see page 6305) value such as "10". StringToCursor returns the TCursor (see page 6305) value that corresponds to the S parameter.

Note: The value returned by StringToCursor is not guaranteed to be a valid cursor. If the Cursors property of the global Screen variable does not include a corresponding entry, the value returned by StringToCursor can not be used to set the Cursor property of a control.

See Also

TControl (see page 5724)

IdentToCursor (see page 6254)

CursorToString (see page 6248)

TScreen

32.3.38 Controls.StripAllFromResult Function

Converts a TModalResult (see page 6312) value from a constant that refers to "all" to the corresponding simple constant.

Pascal

```
function StripAllFromResult(const AModalResult: TModalResult): TModalResult;
```

C++

```
TModalResult StripAllFromResult(const TModalResult AModalResult);
```

File

Controls

Description

Call StripAllFromResult to remove the "all" interpretation from a TModalResult (see page 6312) value. For example, after checking the result from a form's ShowModal method with the IsAnAllResult (see page 6259) function, you can call StripAllFromResult to allow common code to process results that apply to a single selected item or all items.

StripAllFromResult makes the following conversions:

AModalResult	Return value
mrAll (see page 6372)	mrOk (see page 6375)
mrNoToAll (see page 6375)	mrNo (see page 6374)
mrYesToAll (see page 6376)	mrYes (see page 6376)

See Also

TCustomForm

TButton

IsAbortResult (see page 6258)

IsAnAllResult (see page 6259)

IsPositiveResult (see page 6260)

IsNegativeResult (see page 6260)

TCustomForm

32.3.39 Controls.Win32NLSEnableIME Function

Pascal

```
function Win32NLSEnableIME(hWnd: HWND; Enable: Boolean): Boolean;
```

C++

```
Boolean Win32NLSEnableIME(HWND hWnd, Boolean Enable);
```

File

Controls

Description

This is function Controls.Win32NLSEnableIME.

32.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAlign (see page 6268)	TAlign specifies the alignment of a control within its parent.
	TAnchorKind (see page 6269)	TAnchorKind and TAnchors (see page 6294) specify how a control is anchored to its parent.
	TBevelCut (see page 6270)	TBevelCut specifies a bevel's cut.
	TBevelEdge (see page 6270)	TBevelEdge and TBevelEdges (see page 6295) specify which edges of a window are beveled.
	TBevelKind (see page 6271)	TBevelKind specifies the kind of bevel that appears on a control.
	TDockOrientation (see page 6282)	TDockOrientation specifies the orientation of controls in a docking zone.
	TDragKind (see page 6282)	TDragKind indicates the type of operation that occurs when the user drags a control.
	TDragMessage (see page 6283)	This is record Controls.TDragMessage.
	TDragMode (see page 6283)	TDragMode indicates how a control initiates drag operations.
	TDragState (see page 6284)	TDragState is used to specify how the mouse is moving in relation to a control.
	TImeMode (see page 6285)	TImeMode represents the current mode of an Input Method Editor (IME).
	TMouseActivate (see page 6287)	TMouseActivate defines values for the OnMouseActivate event.
	TMouseButton (see page 6288)	TMouseButton identifies a mouse button in a mouse-event handler.

Records

	Name	Description
	TAlignInfo (see page 6269)	This is record Controls.TAlignInfo.
	TCMCancelMode (see page 6272)	This is record Controls.TCMCancelMode.
	TCMChanged (see page 6272)	This is record Controls.TCMChanged.
	TCMChildKey (see page 6273)	This is record Controls.TCMChildKey.
	TCMControlChange (see page 6273)	This is record Controls.TCMControlChange.
	TCMControlListChange (see page 6274)	This is record Controls.TCMControlListChange.
	TCMControlListChanging (see page 6274)	This is record Controls.TCMControlListChanging.
	TCMCreatePopup (see page 6275)	This is record Controls.TCMCreatePopup.
	TCMDockClient (see page 6275)	This is record Controls.TCMDockClient.
	TCMDockNotification (see page 6276)	This is record Controls.TCMDockNotification.
	TCMDrag (see page 6276)	This is record Controls.TCMDrag.
	TCMFloat (see page 6277)	This is record Controls.TCMFloat.
	TCMFocusChanged (see page 6277)	This is record Controls.TCMFocusChanged.
	TCMMouseActivate (see page 6278)	This is record Controls.TCMMouseActivate.
	TCMMouseWheel (see page 6278)	This is record Controls.TCMMouseWheel.
	TCMPopupHWndDestroy (see page 6279)	This is record Controls.TCMPopupHWndDestroy.
	TCMUnDockClient (see page 6279)	This is record Controls.TCMUnDockClient.
	TControlListItem (see page 6280)	This is record Controls.TControlListItem.
	TCreateParams (see page 6280)	TCreateParams is the window-creation parameter record used to specify the type of window to create for a windowed control.
	TDockNotifyRec (see page 6281)	TDockNotifyRec represents the information from a Windows docking notification message, and PDockNotifyRec (see page 6291) is a pointer to a TDockNotifyRec value.
	TDragRec (see page 6284)	This is record Controls.TDragRec.
	TMouseActivateRec (see page 6288)	This is record Controls.TMouseActivateRec.
	TPopupFormInfo (see page 6289)	This is record Controls.TPopupFormInfo.

32.4.1 Controls.TAlign Enumeration

TAlign specifies the alignment of a control within its parent.

Pascal

```
TAlign = (
  alNone,
  alTop,
  alBottom,
  alLeft,
  alRight,
  alClient,
  alCustom
);
```

C++

```
enum TAlign {
  alNone,
  alTop,
  alBottom,
  alLeft,
  alRight,
  alClient,
  alCustom
};
```

File

Controls

Description

TAlign specifies how a control is placed relative to its parent. It can have one of the following values:

Value	Meaning
alNone	The control remains where it was placed. This is the default value.
alTop	The control moves to the top of its parent and resizes to fill the width of its parent. The height of the control is not affected.
alBottom	The control moves to the bottom of its parent and resizes to fill the width of its parent. The height of the control is not affected.
alLeft	The control moves to the left side of its parent and resizes to fill the height of its parent. The width of the control is not affected.
alRight	The control moves to the right side of its parent and resizes to fill the height of its parent. The width of the control is not affected.
alClient	The control resizes to fill the client area of its parent. If another control already occupies part of the client area, the control resizes to fit within the remaining client area.
alCustom	The control's positioning is determined by calls to its parent's CustomAlignInsertBefore and CustomAlignPosition methods.

See Also

CustomAlignInsertBefore

CustomAlignPosition

32.4.2 Controls.TAlignInfo Record

Pascal

```
TAlignInfo = record
  AlignList: TList;
  ControlIndex: Integer;
  Align: TAlign;
  Scratch: Integer;
end;
```

C++

```
struct TAlignInfo {
  TList AlignList;
  int ControlIndex;
  TAlign Align;
  int Scratch;
};
```

File

Controls

Description

This is record Controls.TAlignInfo.

32.4.3 Controls.TAnchorKind Enumeration

TAnchorKind and TAnchors (see page 6294) specify how a control is anchored to its parent.

Pascal

```
TAnchorKind = (
  akLeft,
  akTop,
  akRight,
  akBottom
);
```

C++

```
enum TAnchorKind {
  akLeft,
  akTop,
  akRight,
  akBottom
};
```

File

Controls

Description

TAnchors (see page 6294) is a set of TAnchorKind values. TAnchorKind can have one or more of the following values:

Value	Meaning
akTop	The control's position is fixed with respect to the top edge of its parent.

akLeft	The control's position is fixed with respect to the left edge of its parent.
akRight	The control's position is fixed with respect to the right edge of its parent.
akBottom	The control's position is fixed with respect to the bottom edge of its parent.

32.4.4 Controls.TBevelCut Enumeration

TBevelCut specifies a bevel's cut.

Pascal

```
TBevelCut = (
  bvNone,
  bvLowered,
  bvRaised,
  bvSpace
);
```

C++

```
enum TBevelCut {
  bvNone,
  bvLowered,
  bvRaised,
  bvSpace
};
```

File

Controls

Description

TBevelCut can have one of the following values:

Value	Meaning
bvNone	The bevel does not exist.
bvLowered	The bevel appears lowered.
bvRaised	The bevel appears raised.
bvSpace	The bevel appears as a space if its kind is not bkTile. Otherwise, the bevel appears raised.

See Also

TWinControl

TWinControl

32.4.5 Controls.TBevelEdge Enumeration

TBevelEdge and TBevelEdges (see page 6295) specify which edges of a window are beveled.

Pascal

```
TBevelEdge = (
  beLeft,
```

```
    beTop,
    beRight,
    beBottom
  );
```

C++

```
  enum TBevelEdge {
    beLeft,
    beTop,
    beRight,
    beBottom
  };
```

File

Controls

Description

TBevelEdges (see page 6295) can have one or more of the following values:

Value	Meaning
beLeft	The left edge is beveled.
beTop	The top edge is beveled.
beRight	The right edge is beveled.
beBottom	The bottom edge is beveled.

32.4.6 Controls.TBevelKind Enumeration

TBevelKind specifies the kind of bevel that appears on a control.

Pascal

```
  TBevelKind = (
    bkNone,
    bkTile,
    bkSoft,
    bkFlat
  );
```

C++

```
  enum TBevelKind {
    bkNone,
    bkTile,
    bkSoft,
    bkFlat
  };
```

File

Controls

Description

TBevelKind can have one of the following values:

Value	Meaning
bkNone	No bevel is added.

bkTile	The bevel appears sharply defined.
bkSoft	The bevel uses softer contrasts than bkTile.
bkFlat	The bevel has a broad, flat appearance.

32.4.7 Controls.TCMCancelMode Record

Pascal

```
TCMCancelMode = record
  Msg: Cardinal;
  Unused: Integer;
  Sender: TControl;
  Result: Longint;
end;
```

C++

```
struct TCMCancelMode {
  unsigned Msg;
  int Unused;
  TControl * Sender;
  Longint Result;
};
```

File

Controls

Description

This is record Controls.TCMCancelMode.

32.4.8 Controls.TCMChanged Record

Pascal

```
TCMChanged = record
  Msg: Cardinal;
  Unused: Longint;
  Child: TControl;
  Result: Longint;
end;
```

C++

```
struct TCMChanged {
  unsigned Msg;
  Longint Unused;
  TControl * Child;
  Longint Result;
};
```

File

Controls

Description

This is record Controls.TCMChanged.

32.4.9 Controls.TCMChildKey Record

Pascal

```
TCMChildKey = record
  Msg: Cardinal;
  CharCode: Word;
  Unused: Word;
  Sender: TWinControl;
  Result: Longint;
end;
```

C++

```
struct TCMChildKey {
  unsigned Msg;
  Word CharCode;
  Word Unused;
  TWinControl * Sender;
  Longint Result;
};
```

File

Controls

Description

This is record Controls.TCMChildKey.

32.4.10 Controls.TCMControlChange Record

Pascal

```
TCMControlChange = record
  Msg: Cardinal;
  Control: TControl;
  Inserting: LongBool;
  Result: Longint;
end;
```

C++

```
struct TCMControlChange {
  unsigned Msg;
  TControl * Control;
  LongBool Inserting;
  Longint Result;
};
```

File

Controls

Description

This is record Controls.TCMControlChange.

32.4.11 Controls.TCMControlListChange Record

Pascal

```
TCMControlListChange = record
  Msg: Cardinal;
  Control: TControl;
  Inserting: LongBool;
  Result: Longint;
end;
```

C++

```
struct TCMControlListChange {
  unsigned Msg;
  TControl * Control;
  LongBool Inserting;
  Longint Result;
};
```

File

Controls

Description

This is record Controls.TCMControlListChange.

32.4.12 Controls.TCMControlListChanging Record

Pascal

```
TCMControlListChanging = record
  Msg: Cardinal;
  ControlListItem: PControlListItem;
  Inserting: LongBool;
  Result: Longint;
end;
```

C++

```
struct TCMControlListChanging {
  unsigned Msg;
  PControlListItem ControlListItem;
  LongBool Inserting;
  Longint Result;
};
```

File

Controls

Description

This is record Controls.TCMControlListChanging.

32.4.13 Controls.TCMCreatePopup Record

Pascal

```
TCMCreatePopup = packed record
  Msg: Cardinal;
  PopupID: Integer;
  OwnerWnd: HWND;
  Result: Integer;
end;
```

C++

```
struct TCMCreatePopup {
  unsigned Msg;
  int PopupID;
  HWND OwnerWnd;
  int Result;
};
```

File

Controls

Description

This is record Controls.TCMCreatePopup.

32.4.14 Controls.TCMDockClient Record

Pascal

```
TCMDockClient = packed record
  Msg: Cardinal;
  DockSource: TDragDockObject;
  MousePos: TSmallPoint;
  Result: Integer;
end;
```

C++

```
struct TCMDockClient {
  unsigned Msg;
  TDragDockObject DockSource;
  TSmallPoint MousePos;
  int Result;
};
```

File

Controls

Description

This is record Controls.TCMDockClient.

32.4.15 Controls.TCMDockNotification Record

Pascal

```
TCMDockNotification = packed record
  Msg: Cardinal;
  Client: TControl;
  NotifyRec: PDockNotifyRec;
  Result: Integer;
end;
```

C++

```
struct TCMDockNotification {
  unsigned Msg;
  TControl * Client;
  PDockNotifyRec NotifyRec;
  int Result;
};
```

File

Controls

Description

This is record Controls.TCMDockNotification.

32.4.16 Controls.TCMDrag Record

Pascal

```
TCMDrag = packed record
  Msg: Cardinal;
  DragMessage: TDragMessage;
  Reserved1: Byte;
  Reserved2: Word;
  DragRec: PDragRec;
  Result: Longint;
end;
```

C++

```
struct TCMDrag {
  unsigned Msg;
  TDragMessage DragMessage;
  Byte Reserved1;
  Word Reserved2;
  PDragRec DragRec;
  Longint Result;
};
```

File

Controls

Description

This is record Controls.TCMDrag.

32.4.17 Controls.TCMFloat Record

Pascal

```
TCMFloat = packed record
  Msg: Cardinal;
  Reserved: Integer;
  DockSource: TDragDockObject;
  Result: Integer;
end;
```

C++

```
struct TCMFloat {
  unsigned Msg;
  int Reserved;
  TDragDockObject DockSource;
  int Result;
};
```

File

Controls

Description

This is record Controls.TCMFloat.

32.4.18 Controls.TCMFocusChanged Record

Pascal

```
TCMFocusChanged = record
  Msg: Cardinal;
  Unused: Integer;
  Sender: TWinControl;
  Result: Longint;
end;
```

C++

```
struct TCMFocusChanged {
  unsigned Msg;
  int Unused;
  TWinControl * Sender;
  Longint Result;
};
```

File

Controls

Description

This is record Controls.TCMFocusChanged.

32.4.19 Controls.TCMMouseActivate Record

Pascal

```
TCMMouseActivate = packed record
  Msg: Cardinal;
  Reserved: Integer;
  MouseActivateRec: PMouseActivateRec;
  Result: Integer;
end;
```

C++

```
struct TCMMouseActivate {
  unsigned Msg;
  int Reserved;
  PMouseActivateRec MouseActivateRec;
  int Result;
};
```

File

Controls

Description

This is record Controls.TCMMouseActivate.

32.4.20 Controls.TCMMouseWheel Record

Pascal

```
TCMMouseWheel = record
  Msg: Cardinal;
  ShiftState: TShiftState;
  Unused: Byte;
  WheelDelta: SmallInt;
  case Integer of
    0: (XPos: Smallint;
        YPos: Smallint;);
    1: (Pos: TSmallPoint;
        Result: Longint;);
end;
```

C++

```
struct TCMMouseWheel {
  unsigned Msg;
  TShiftState ShiftState;
  Byte Unused;
  SmallInt WheelDelta;
  Smallint XPos;
  Smallint YPos;
  TSmallPoint Pos;
  Longint Result;
};
```

File

Controls

Description

This is record Controls.TCMMouseWheel.

32.4.21 Controls.TCMPopupHWndDestroy Record

Pascal

```
TCMPopupHWndDestroy = packed record
  Msg: Cardinal;
  PopupFormInfo: PPopupFormInfo;
  PopupControlWnd: HWND;
  Result: Integer;
end;
```

C++

```
struct TCMPopupHWndDestroy {
  unsigned Msg;
  PPopupFormInfo PopupFormInfo;
  HWND PopupControlWnd;
  int Result;
};
```

File

Controls

Description

This is record Controls.TCMPopupHWndDestroy.

32.4.22 Controls.TCMUnDockClient Record

Pascal

```
TCMUnDockClient = packed record
  Msg: Cardinal;
  NewTarget: TControl;
  Client: TControl;
  Result: Integer;
end;
```

C++

```
struct TCMUnDockClient {
  unsigned Msg;
  TControl * NewTarget;
  TControl * Client;
  int Result;
};
```

File

Controls

Description

This is record Controls.TCMUnDockClient.

32.4.23 Controls.TControlListItem Record

Pascal

```
TControlListItem = record
  Control: TControl;
  Parent: TWinControl;
end;
```

C++

```
struct TControlListItem {
  TControl * Control;
  TWinControl * Parent;
};
```

File

Controls

Description

This is record Controls.TControlListItem.

32.4.24 Controls.TCreateParams Record

TCreateParams is the window-creation parameter record used to specify the type of window to create for a windowed control.

Pascal

```
TCreateParams = record
  Caption: PChar;
  Style: DWORD;
  ExStyle: DWORD;
  X: Integer;
  Y: Integer;
  Height: Integer;
  Width: Integer;
  WndParent: HWnd;
  Param: Pointer;
  WindowClass: TWndClass;
  WinClassName: array[0..63] of Char;
end;
```

C++

```
struct TCreateParams {
  const char * Caption;
  DWORD Style;
  DWORD ExStyle;
  int X;
  int Y;
  int Height;
  int Width;
  HWnd WndParent;
  void * Param;
  TWndClass WindowClass;
  array[0..63] of Char WinClassName;
};
```

File

Controls

Description

TCreateParams is a data structure for holding information needed when telling Windows to create a window. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function. TCreateParams contains the following fields:

Field	Meaning
Caption	The window caption. This is usually the value of the Caption or Text property of the control.
Style	The Windows style for the window. This is a bit array of flags such as WS_CHILD or WS_DISABLED. See the Microsoft Windows documentation for possible flags.
ExStyle	The extended Windows style for the window. This is one of the Windows predefined extended style constants such as WS_EX_TOOLWINDOW or WS_EX_CONTROLPARENT. See the Microsoft Windows documentation for possible values.
X	The x coordinate of the left side of the window. This is usually the same as the Left property.
Y	The y coordinate of the top of the window. This is usually the same as the Top property.
Width	The width, in pixels, of the window. This is usually the same as the Width property.
Height	The height, in pixels, of the window. This is usually the same as the Height property.
WndParent	The window handle of the parent window. This is the same as the Handle property of the parent control.
Param	A pointer to the windows creation parameter that is passed as the LParam of the WM_CREATE message.
WindowClass	The WindowClass record that describes the class of window being created. The WindowClass record includes information about the window procedure, style, default cursor, background brush, associated menus, and so on.
WinClassName	The window class name. This must be the name of a window class that has already been registered. Windows provides several built-in window classes with predefined names. Applications may also register their own custom window classes.

See Also

TWinControl

32.4.25 Controls.TDockNotifyRec Record

TDockNotifyRec represents the information from a Windows docking notification message, and PDockNotifyRec (see page 6291) is a pointer to a TDockNotifyRec value.

Pascal

```
TDockNotifyRec = record
  ClientMsg: Cardinal;
  MsgWParam: Integer;
  MsgLParam: Integer;
end;
```

C++

```
struct TDockNotifyRec {
  unsigned ClientMsg;
  int MsgWParam;
  int MsgLParam;
```

```
};
```

File

Controls

Description

ClientMsg is the Windows message ID.

MsgWParam is the message's WParam.

MsgLParam is the message's LParam.

32.4.26 Controls.TDockOrientation Enumeration

TDockOrientation specifies the orientation of controls in a docking zone.

Pascal

```
TDockOrientation = (
  doNoOrient,
  doHorizontal,
  doVertical
);
```

C++

```
enum TDockOrientation {
  doNoOrient,
  doHorizontal,
  doVertical
};
```

File

Controls

Description

TDockOrientation can have one of the following values:

Value	Meaning
doNoOrient	The zone contains a control instead of child zones.
doHorizontal	The zone's children are arranged top to bottom.
doVertical	The zone's children are arranged left to right.

32.4.27 Controls.TDragKind Enumeration

TDragKind indicates the type of operation that occurs when the user drags a control.

Pascal

```
TDragKind = (
  dkDrag,
  dkDock
);
```

C++

```
enum TDragKind {
  dkDrag,
  dkDock
};
```

File

Controls

Description

TDragKind can have one of the following values:

Value	Meaning
dkDrag	The object is being dragged for a drag-and-drop operation.
dkDock	The object is being dragged for a drag-and-dock operation.

32.4.28 Controls.TDragMessage Enumeration

Pascal

```
TDragMessage = (
  dmDragEnter,
  dmDragLeave,
  dmDragMove,
  dmDragDrop,
  dmDragCancel,
  dmFindTarget
);
```

C++

```
enum TDragMessage {
  dmDragEnter,
  dmDragLeave,
  dmDragMove,
  dmDragDrop,
  dmDragCancel,
  dmFindTarget
};
```

File

Controls

Description

This is record Controls.TDragMessage.

32.4.29 Controls.TDragMode Enumeration

TDragMode indicates how a control initiates drag operations.

Pascal

```
TDragMode = (
```

```
    dmManual,
    dmAutomatic
  );
```

C++

```
  enum TDragMode {
    dmManual,
    dmAutomatic
  };
```

File

Controls

Description

TDragMode can have one of the following values:

Value	Meaning
dmAutomatic	The control begins a drag operation automatically when the user clicks and drags it.
dmManual	The control can't be dragged until the application calls the BeginDrag method.

32.4.30 Controls.TDragRec Record

Pascal

```
  TDragRec = record
    Pos: TPoint;
    Source: TDragObject;
    Target: Pointer;
    Docking: Boolean;
  end;
```

C++

```
  struct TDragRec {
    TPoint Pos;
    TDragObject Source;
    void * Target;
    Boolean Docking;
  };
```

File

Controls

Description

This is record Controls.TDragRec.

32.4.31 Controls.TDragState Enumeration

TDragState is used to specify how the mouse is moving in relation to a control.

Pascal

```
  TDragState = (
    dsDragEnter,
```

```
    dsDragLeave,
    dsDragMove
  );
```

C++

```
  enum TDragState {
    dsDragEnter,
    dsDragLeave,
    dsDragMove
  };
```

File

Controls

Description

TDragState describes how the mouse is moving in relation to a control that it is passing over. It has one of the following values:

Value	Meaning
dsDragEnter	The mouse just moved over the control.
dsDragLeave	The mouse is moving off of the control.
dsDragMove	The mouse is moving while over the control.

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
```

32.4.32 Controls.TImeMode Enumeration

TImeMode represents the current mode of an Input Method Editor (IME).

Pascal

```
  TImeMode = (
    imDisable,
```

```
    imClose,
    imOpen,
    imDontCare,
    imSAlpha,
    imAlpha,
    imHira,
    imSKata,
    imKata,
    imChinese,
    imSHanguel,
    imHanguel
  );
```

C++

```
  enum TImeMode {
    imDisable,
    imClose,
    imOpen,
    imDontCare,
    imSAlpha,
    imAlpha,
    imHira,
    imSKata,
    imKata,
    imChinese,
    imSHanguel,
    imHanguel
  };
```

File

Controls

Description

TImeMode represents the way an Input Method Editor operates to convert keyboard input typed on an Asian version of Windows. TImeMode has the following values:

Value	Meaning
imDisable	Shut down the IME. imDisable has no effect on Chinese, Taiwanese, or Korean IMEs.
imClose	Close the IME conversion window, but leave the IME running in the background. The IME can be reactivated by a hot key combination.
imOpen	Open the IME conversion window. The conversion mode is the last conversion mode used by the IME.
imDontCare	Launch the IME if it is disabled. The conversion mode is the last conversion mode used by the IME.
imSAlpha	Open the IME conversion window and set the conversion mode to accept single-width Roman alphabet input.
imAlpha	Open the IME conversion window and set the conversion mode to accept double-width Roman alphabet input.
imHira	Open the IME conversion window and set the conversion mode to double-width Hiragana. imHira is only available for Japanese IMEs.
imSKata	Open the IME conversion window and set the conversion mode to single-width Katakana (Hankaku Katakana). imSKata is only available for Japanese IMEs.
imKata	Open the IME conversion window and set the conversion mode to double-width Katakana (Zenkaku Katakana). imKata is only available for Japanese IMEs.
imChinese	Open the IME conversion window and set the conversion mode to double-width Chinese. imChinese is only available for Chinese IMEs.
imSHanguel	Open the IME conversion window and set the conversion mode to single-width Hanguel. imSHanguel is only available for Korean IMEs.

imHanguel	Open the IME conversion window and set the conversion mode to double-width Hanguel. imHanguel is only available for Korean IMEs.

32.4.33 Controls.TMouseActivate Enumeration

TMouseActivate defines values for the OnMouseActivate event.

Pascal

```
TMouseActivate = (
  maDefault,
  maActivate,
  maActivateAndEat,
  maNoActivate,
  maNoActivateAndEat
);
```

C++

```
enum TMouseActivate {
  maDefault,
  maActivate,
  maActivateAndEat,
  maNoActivate,
  maNoActivateAndEat
};
```

File

Controls

Description

The TMouseActivate type provides the following values to the OnMouseActivate event.

Type	Description
maDefault	Allows the parent control to process the OnMouseActivate event.
maActivate	Activates the top level form and allows the mouse button down message to go through. No Parent window processing occurs.
maActivateAndEat	Same as maActivate, except that the button down message is suppressed.
maNoActivate	Does not activate the top-level window, but allows the mouse button down message to continue to be processed.
maNoActivateAndEat	Same as maNoActivate, except that the button down message is suppressed.

See Also

TControl

TControl

TControl

32.4.34 Controls.TMouseActivateRec Record

Pascal

```
TMouseActivateRec = record
  MousePos: TPoint;
  HitTest: Integer;
  Button: TMouseButton;
  ShiftState: TShiftState;
  TopLevel: HWND;
end;
```

C++

```
struct TMouseActivateRec {
  TPoint MousePos;
  int HitTest;
  TMouseButton Button;
  TShiftState ShiftState;
  HWND TopLevel;
};
```

File

Controls

Description

This is record Controls.TMouseActivateRec.

32.4.35 Controls.TMouseButton Enumeration

TMouseButton identifies a mouse button in a mouse-event handler.

Pascal

```
TMouseButton = (
  mbLeft,
  mbRight,
  mbMiddle
);
```

C++

```
enum TMouseButton {
  mbLeft,
  mbRight,
  mbMiddle
};
```

File

Controls

Description

The TMouseButton type defines the mouse-button constants used by mouse-event handlers to distinguish which button generated the mouse event.

See Also

TControl

TControl

TControl

32.4.36 Controls.TPopupFormInfo Record

Pascal

```
TPopupFormInfo = record
  PopupID: Integer;
  PopupWnd: HWND;
  IsPopup: Boolean;
end;
```

C++

```
struct TPopupFormInfo {
  int PopupID;
  HWND PopupWnd;
  Boolean IsPopup;
};
```

File

Controls

Description

This is record Controls.TPopupFormInfo.

32.5 Types

The following table lists types in this documentation.

Types

Name	Description
PControlListItem (see page 6291)	This is type Controls.PControlListItem.
PDockNotifyRec (see page 6291)	This is type Controls.PDockNotifyRec.
PDragRec (see page 6292)	This is type Controls.PDragRec.
PMouseActivateRec (see page 6292)	This is type Controls.PMouseActivateRec.
PPopupFormInfo (see page 6292)	This is type Controls.PPopupFormInfo.
TAlignInsertBeforeEvent (see page 6293)	TAlignInsertBeforeEvent is the function type for an OnAlignInsertBefore event.
TAlignPositionEvent (see page 6293)	TAlignPositionEvent is the function type for an OnAlignPosition event.
TAlignSet (see page 6294)	TAlignSet specifies the alignment of a child control within its parent.
TAnchors (see page 6294)	TBorderWidth (see page 6296) defines the distance, in pixels, between the outer and inner bevels of a control.
TAnimateWindowProc (see page 6295)	This is type Controls.TAnimateWindowProc.
TBevelEdges (see page 6295)	TBevelEdge (see page 6270) and TBevelEdges specify which edges of a window are beveled.
TBevelWidth (see page 6296)	This is type Controls.TBevelWidth.
TBorderWidth (see page 6296)	This is type Controls.TBorderWidth.
TCanResizeEvent (see page 6296)	TCanResizeEvent is the type of the OnCanResize event handler.
TCaption (see page 6297)	TCaption is the type of the Caption property.
TCMActivate (see page 6297)	This is type Controls.TCMActivate.
TCMDeactivate (see page 6297)	This is type Controls.TCMDeactivate.
TCMDesignHitTest (see page 6298)	This is type Controls.TCMDesignHitTest.
TCMDialogChar (see page 6298)	This is type Controls.TCMDialogChar.
TCMDialogKey (see page 6298)	This is type Controls.TCMDialogKey.
TCMEnter (see page 6299)	This is type Controls.TCMEnter.
TCMExit (see page 6299)	This is type Controls.TCMExit.
TCMGotFocus (see page 6299)	This is type Controls.TCMGotFocus.
TCMHitTest (see page 6300)	This is type Controls.TCMHitTest.
TCMLostFocus (see page 6300)	This is type Controls.TCMLostFocus.
TCMWantSpecialKey (see page 6300)	This is type Controls.TCMWantSpecialKey.
TConstrainedResizeEvent (see page 6301)	This is type Controls.TConstrainedResizeEvent.
TConstraintSize (see page 6301)	This is type Controls.TConstraintSize.
TContextPopupEvent (see page 6301)	This is type Controls.TContextPopupEvent.
TControlActionLinkClass (see page 6302)	TControlActionLinkClass defines the metaclass for TControlActionLink (see page 5892).
TControlClass (see page 6302)	TControlClass defines the metaclass for TControl (see page 5724).
TControlState (see page 6302)	TControlState describes the current state of a control.
TControlStyle (see page 6303)	The TControlStyle type describes the current style of the control.
TCursor (see page 6305)	TCursor identifies the cursor type.
TDate (see page 6306)	TDate is used to represent a date.
TDockDropEvent (see page 6307)	TDockDropEvent is the type of the OnDockDrop event handler.
TDockOverEvent (see page 6307)	TDockOverEvent is the type of the OnDockOver event handler.
TDockTreeClass (see page 6308)	TDockTreeClass defines the metaclass for TDockTree (see page 5983).
TDragDropEvent (see page 6308)	This is type Controls.TDragDropEvent.
TDragObjectClass (see page 6308)	TDragObjectClass defines the metaclass for TDragObject (see page 6031).
TDragOverEvent (see page 6309)	This is type Controls.TDragOverEvent.
TEndDragEvent (see page 6309)	This is type Controls.TEndDragEvent.

TForEachZoneProc (see page 6309)	This is type Controls.TForEachZoneProc.
TGetSiteInfoEvent (see page 6310)	TGetSiteInfoEvent is the type of the OnGetSiteInfo event handler.
THintWindowClass (see page 6310)	THintWindowClass defines the metaclass for THintWindow (see page 6055).
TImeName (see page 6311)	TImeName is the type that represents the name of an Input Method Editor.
TKeyEvent (see page 6311)	This is type Controls.TKeyEvent.
TKeyPressEvent (see page 6311)	This is type Controls.TKeyPressEvent.
TMarginSize (see page 6312)	This is type Controls.TMarginSize.
TModalResult (see page 6312)	TModalResult represents the return value from a modal dialog.
TMouseActivateEvent (see page 6313)	This is type Controls.TMouseActivateEvent.
TMouseEvent (see page 6313)	This is type Controls.TMouseEvent.
TMouseMoveEvent (see page 6314)	This is type Controls.TMouseMoveEvent.
TMouseWheelEvent (see page 6314)	This is type Controls.TMouseWheelEvent.
TMouseWheelUpDownEvent (see page 6314)	This is type Controls.TMouseWheelUpDownEvent.
TPanningWindowClass (see page 6315)	TPanningWindowClass defines the metaclass for TPanningWindow.
TScalingFlags (see page 6315)	TScalingFlags describes the way controls are scaled.
TStartDockEvent (see page 6316)	TStartDockEvent is the type of the OnStartDock event handler.
TStartDragEvent (see page 6316)	This is type Controls.TStartDragEvent.
TTabOrder (see page 6317)	TTabOrder defines the tab order of a control.
TTime (see page 6317)	TTime is used to represent a time.
TUnDockEvent (see page 6317)	TUnDockEvent is the type of the OnUnDock event handler.
TWinControlActionLinkClass (see page 6318)	TWinControlActionLinkClass defines the metaclass for TWinControlActionLink (see page 6233).
TWinControlClass (see page 6318)	TWinControlClass defines the metaclass for TWinControl (see page 6112).
TWndMethod (see page 6319)	TWndMethod is the type for Windows message procedures.

32.5.1 Controls.PControlListItem Type

Pascal

```
PControlListItem = ^TControlListItem;
```

C++

```
^TControlListItem PControlListItem;
```

File

Controls

Description

This is type Controls.PControlListItem.

32.5.2 Controls.PDockNotifyRec Type

Pascal

```
PDockNotifyRec = ^TDockNotifyRec;
```

C++

```
^TDockNotifyRec PDockNotifyRec;
```

File

Controls

Description

This is type Controls.PDockNotifyRec.

32.5.3 Controls.PDragRec Type

Pascal

```
PDragRec = ^TDragRec;
```

C++

```
^TDragRec PDragRec;
```

File

Controls

Description

This is type Controls.PDragRec.

32.5.4 Controls.PMouseActivateRec Type

Pascal

```
PMouseActivateRec = ^TMouseActivateRec;
```

C++

```
^TMouseActivateRec PMouseActivateRec;
```

File

Controls

Description

This is type Controls.PMouseActivateRec.

32.5.5 Controls.PPopupFormInfo Type

Pascal

```
PPopupFormInfo = ^TPopupFormInfo;
```

C++

```
^TPopupFormInfo PPopupFormInfo;
```

File

Controls

Description

This is type Controls.PPopupFormInfo.

32.5.6 Controls.TAlignInsertBeforeEvent Type

TAlignInsertBeforeEvent is the function type for an OnAlignInsertBefore event.

Pascal

```
TAlignInsertBeforeEvent = function (Sender: TWinControl; C1, C2: TControl): Boolean of object;
```

C++

```
(Sender: TWinControl; C1, C2: TControl): Boolean ( TAlignInsertBeforeEvent)();
```

File

Controls

Description

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**. It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise.

TAlignInsertBeforeEvent has the following parameters:

Value	Meaning
Sender	Parent control.
C1	First control to be ordered.
C2	Second control to be ordered.

See Also

CustomAlignInsertBefore (see page 6137)

OnAlignInsertBefore (see page 6213)

32.5.7 Controls.TAlignPositionEvent Type

TAlignPositionEvent is the function type for an OnAlignPosition event.

Pascal

```
TAlignPositionEvent = procedure (Sender: TWinControl; Control: TControl; var NewLeft, NewTop,
NewWidth, NewHeight: Integer; var AlignRect: TRect; AlignInfo: TAlignInfo) of object;
```

C++

```
(Sender: TWinControl; Control: TControl; var NewLeft, NewTop, NewWidth, NewHeight: Integer;
var AlignRect: TRect; AlignInfo: TAlignInfo) ( TAlignPositionEvent)();
```

File

Controls

Description

CustomAlignPosition triggers the OnAlignPosition event. OnAlignPosition occurs for each child control with an Align property of **alCustom** when it is aligned. It can determine where the control is placed by specifying the alignment parameters.

TAlignPositionEvent has the following parameters:

Parameter	Meaning
Sender	Parent control.
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo (see page 6269).

See Also

CustomAlignPosition (see page 6138)

OnAlignPosition (see page 6214)

TAlignInfo (see page 6269)

32.5.8 Controls.TAlignSet Type

TAlignSet specifies the alignment of a child control within its parent.

Pascal

```
TAlignSet = set of TAlign;
```

C++

```
TAlign TAlignSet;
```

File

Controls

32.5.9 Controls.TAnchors Type

TBorderWidth (see page 6296) defines the distance, in pixels, between the outer and inner bevels of a control.

Pascal

```
TAnchors = set of TAnchorKind;
```

C++

```
TAnchorKind TAnchors;
```

File

Controls

Description

TBorderWidth (see page 6296) is used to specify the border width of a control via its BorderWidth property.

Note: Do not confuse the border of the panel with line drawn around the panel itself. The line around the panel is specified by the BorderStyle property.

See Also

BevelWidth (see page 13052)

BorderStyle (see page 13054)

32.5.10 Controls.TAnimateWindowProc Type

Pascal

```
TAnimateWindowProc = function (hWnd: HWND; dwTime: DWORD; dwFlags: DWORD): BOOL;
```

C++

```
(hWnd: HWND; dwTime: DWORD; dwFlags: DWORD): BOOL ( TAnimateWindowProc)();
```

File

Controls

Description

This is type Controls.TAnimateWindowProc.

32.5.11 Controls.TBevelEdges Type

TBevelEdge (see page 6270) and TBevelEdges specify which edges of a window are beveled.

Pascal

```
TBevelEdges = set of TBevelEdge;
```

C++

```
TBevelEdge TBevelEdges;
```

File

Controls

Description

TBevelEdges can have one or more of the following values:

Value	Meaning
beLeft	The left edge is beveled.
beTop	The top edge is beveled.
beRight	The right edge is beveled.
beBottom	The bottom edge is beveled.

32.5.12 Controls.TBevelWidth Type

Pascal

```
TBevelWidth = 1..MaxInt;
```

C++

```
1..MaxInt TBevelWidth;
```

File

Controls

Description

This is type Controls.TBevelWidth.

32.5.13 Controls.TBorderWidth Type

Pascal

```
TBorderWidth = 0..MaxInt;
```

C++

```
0..MaxInt TBorderWidth;
```

File

Controls

Description

This is type Controls.TBorderWidth.

32.5.14 Controls.TCanResizeEvent Type

TCanResizeEvent is the type of the OnCanResize event handler.

Pascal

```
TCanResizeEvent = procedure (Sender: TObject; var NewWidth, NewHeight: Integer; var Resize:
Boolean) of object;
```

C++

```
(Sender: TObject; var NewWidth, NewHeight: Integer; var Resize: Boolean) ( TCanResizeEvent)();
```

File

Controls

Description

Sender is the object that is being resized.

NewWidth returns the new width of the object.

NewHeight returns the new height of the object.

Resize returns true if the resize attempt can proceed and false if the attempt to resize should be aborted.

32.5.15 Controls.TCaption Type

TCaption is the type of the Caption property.

Pascal

```
TCaption = type string;
```

C++

```
type string TCaption;
```

File

Controls

Description

TCaption is the string that represents the caption or text of a control.

32.5.16 Controls.TCMActivate Type

Pascal

```
TCMActivate = TWMNoParams;
```

C++

```
TWMNoParams TCMActivate;
```

File

Controls

Description

This is type Controls.TCMActivate.

32.5.17 Controls.TCMDeactivate Type

Pascal

```
TCMDeactivate = TWMNoParams;
```

C++

```
TWMNoParams TCMDeactivate;
```

File

Controls

Description

This is type Controls.TCMDeactivate.

32.5.18 Controls.TCMDesignHitTest Type

Pascal

```
TCMDesignHitTest = TWMMouse;
```

C++

```
TWMMouse TCMDesignHitTest;
```

File

Controls

Description

This is type Controls.TCMDesignHitTest.

32.5.19 Controls.TCMDialogChar Type

Pascal

```
TCMDialogChar = TWMKey;
```

C++

```
TWMKey TCMDialogChar;
```

File

Controls

Description

This is type Controls.TCMDialogChar.

32.5.20 Controls.TCMDialogKey Type

Pascal

```
TCMDialogKey = TWMKey;
```

C++

```
TWMKey TCMDialogKey;
```

File

Controls

Description

This is type Controls.TCMDialogKey.

32.5.21 Controls.TCMEnter Type

Pascal

```
TCMEnter = TWMNoParams;
```

C++

```
TWMNoParams TCMEnter;
```

File

Controls

Description

This is type Controls.TCMEnter.

32.5.22 Controls.TCMExit Type

Pascal

```
TCMExit = TWMNoParams;
```

C++

```
TWMNoParams TCMExit;
```

File

Controls

Description

This is type Controls.TCMExit.

32.5.23 Controls.TCMGotFocus Type

Pascal

```
TCMGotFocus = TWMNoParams;
```

C++

```
TWMNoParams TCMGotFocus;
```

File

Controls

Description

This is type Controls.TCMGotFocus.

32.5.24 Controls.TCMHitTest Type

Pascal

```
TCMHitTest = TWMNCHitTest;
```

C++

```
TWMNCHitTest TCMHitTest;
```

File

Controls

Description

This is type Controls.TCMHitTest.

32.5.25 Controls.TCMLostFocus Type

Pascal

```
TCMLostFocus = TWMNoParams;
```

C++

```
TWMNoParams TCMLostFocus;
```

File

Controls

Description

This is type Controls.TCMLostFocus.

32.5.26 Controls.TCMWantSpecialKey Type

Pascal

```
TCMWantSpecialKey = TWMKey;
```

C++

```
TWMKey TCMWantSpecialKey;
```

File

Controls

Description

This is type Controls.TCMWantSpecialKey.

32.5.27 Controls.TConstrainedResizeEvent Type

Pascal

```
TConstrainedResizeEvent = procedure (Sender: TObject; var MinWidth, MinHeight, MaxWidth,
MaxHeight: Integer) of object;
```

C++

```
(Sender: TObject; var MinWidth, MinHeight, MaxWidth, MaxHeight: Integer) (
TConstrainedResizeEvent)();
```

File

Controls

Description

This is type Controls.TConstrainedResizeEvent.

32.5.28 Controls.TConstraintSize Type

Pascal

```
TConstraintSize = 0..MaxInt;
```

C++

```
0..MaxInt TConstraintSize;
```

File

Controls

Description

This is type Controls.TConstraintSize.

32.5.29 Controls.TContextPopupEvent Type

Pascal

```
TContextPopupEvent = procedure (Sender: TObject; MousePos: TPoint; var Handled: Boolean) of
object;
```

C++

```
(Sender: TObject; MousePos: TPoint; var Handled: Boolean) ( TContextPopupEvent)();
```

File

Controls

Description

This is type Controls.TContextPopupEvent.

32.5.30 Controls.TControlActionLinkClass Type

TControlActionLinkClass defines the metaclass for TControlActionLink (see page 5892).

Pascal

```
TControlActionLinkClass = class of TControlActionLink;
```

C++

```
TControlActionLink TControlActionLinkClass;
```

File

Controls

Description

TControlActionLinkClass is the metaclass for TControlActionLink (see page 5892). Its value is a class reference for TControlActionLink (see page 5892) or one of its descendants.

32.5.31 Controls.TControlClass Type

TControlClass defines the metaclass for TControl (see page 5724).

Pascal

```
TControlClass = class of TControl;
```

C++

```
TControl * TControlClass;
```

File

Controls

Description

TControlClass is the metaclass for TControl (see page 5724). Its value is the class reference for TControl (see page 5724) or for one of its descendants.

See Also

TControl (see page 5724)

32.5.32 Controls.TControlState Type

TControlState describes the current state of a control.

Pascal

```
TControlState = set of (csLButtonDown, csClicked, csPalette, csReadingState,
csAlignmentNeeded, csFocusing, csCreating, csPaintCopy, csCustomPaint, csDestroyingHandle,
csDocking, csDesignerHide, csPanning, csRecreating, csAligning);
```

C++

```
(csLButtonDown, csClicked, csPalette, csReadingState, csAlignmentNeeded, csFocusing,
csCreating, csPaintCopy, csCustomPaint, csDestroyingHandle, csDocking, csDesignerHide,
csPanning, csRecreating, csAligning) TControlState;
```

File

Controls

Description

TControlState defines a set of flags used to defined the state of a control:

Flag	Meaning
csLButtonDown	The left mouse button was clicked and not yet released. This is set for all mouse-down events.
csClicked	The same as csLButtonDown, but only set if ControlStyle contains csClickEvents, meaning that mouse-down events are interpreted as clicks.
csPalette	The palette has changed and the control or one of its descendants has not finished adjusting by realizing its palette.
csReadingState	The control is reading its state from a stream.
csAlignmentNeeded	The control needs to realign itself when alignment is re-enabled.
csFocusing	The application is processing messages intended to give the control focus. This does not guarantee the control will receive focus, but prevents recursive calls.
csCreating	The control and/or its owner and subcontrols are being created. This flag clears when all have finished creating.
csPaintCopy	The control is being replicated, meaning a copy of the control is being painted. The ControlStyle flag csReplicatable must be set for this state to occur.
csCustomPaint	The control is processing custom paint messages.
csDestroyingHandle	The control's window is being destroyed.
csDocking	The control is being docked.
csPanning	Set if the control is actively being panned.
csRecreating	The control and/or its owner and subcontrols are being recreated. This flag clears when all have finished recreating.

32.5.33 Controls.TControlStyle Type

The TControlStyle type describes the current style of the control.

Pascal

```
TControlStyle = set of (csAcceptsControls, csCaptureMouse, csDesignInteractive, csClickEvents,
csFramed, csSetCaption, csOpaque, csDoubleClicks, csFixedWidth, csFixedHeight,
csNoDesignVisible, csReplicatable, csNoStdEvents, csDisplayDragImage, csReflector,
csActionClient, csMenuEvents, csNeedsBorderPaint, csParentBackground, csPannable,
csAlignWithMargins);
```

C++

```
(csAcceptsControls, csCaptureMouse, csDesignInteractive, csClickEvents, csFramed,
csSetCaption, csOpaque, csDoubleClicks, csFixedWidth, csFixedHeight, csNoDesignVisible,
csReplicatable, csNoStdEvents, csDisplayDragImage, csReflector, csActionClient, csMenuEvents,
csNeedsBorderPaint, csParentBackground, csPannable, csAlignWithMargins) TControlStyle;
```

File

Controls

Description

TControlStyle defines a set of style flags indicating control attributes:

The following table lists the flags and their meanings:

Flag	Meaning
csAcceptsControls	The control becomes the parent of any controls dropped on it at design time.
csCaptureMouse	The control captures mouse events when it is clicked.
csDesignInteractive	The control maps right mouse-button clicks at design time into left mouse-button clicks to manipulate the control.
csClickEvents	The control can receive and respond to mouse clicks.
csFramed	The control has a 3D frame.
csSetCaption	The control's Caption should match the Name property if it has not been explicitly set to something else.
csOpaque	The control completely fills its client rectangle.
csDoubleClicks	The control can receive and respond to double-click messages. Otherwise, map double-clicks into clicks.
csFixedWidth	The width of the control does not vary or scale.
csFixedHeight	The height of the control does not vary or scale.
csNoDesignVisible	The control is not visible at design time.
csReplicatable	The control can be copied using the PaintTo method to draw its image to an arbitrary canvas.
csNoStdEvents	Standard events such as mouse, key, and click events are ignored. This flag allows an application to run faster when there is no need respond to these events.
csDisplayDragImage	If some control that has an associated drag image list is dragged across a control with this setting, then the image list is used to enhance the drag cursor while the cursor is over it. Otherwise, the drag cursor is used on its own.
csReflector	The control responds to Windows dialog messages, focus messages, or size change messages. Use this setting if the control can be used as an ActiveX control, so that it receives notification of these events.
csActionClient	The control is linked to an action object. This flag is set when the Action property is set, and cleared if the Action property is cleared.
csMenuEvents	The control responds to system menu commands.
csPannable	Set this In your control style if it is to support mouse panning.
csNeedsBorderPaint	Only applicable when Themes are Enabled in applications on Windows XP. Causes the ThemeServices to paint the border of a control with the current theme.
csParentBackground	Only applicable when Themes are Enabled in applications on Windows XP. Causes the parent to draw its background into the Control's background. This is useful for controls which need to show their parent's theme elements, such as a TPanel or TFrame that appear on a TPageControl. TWinControl (see page 6112) introduces a protected ParentBackground property that includes/excludes the **csParentBackground** control style.
csAlignWithMargins	Use the Margins property of the control to govern the spacing relative to other controls that are aligned with this one. The controls are not allowed to be any closer than the spacing specified in Margins. This spacing is maintained as controls are moved when the parent control resizes.

See Also

TFrame (see page 14114)

TMargins (see page 6087)

TPageControl (see page 4459)

TPanel (see page 13032)

TWinControl (see page 6112)

ParentBackground (see page 6209)

TControl (see page 5724)

Margins (see page 5831)

32.5.34 Controls.TCursor Type

TCursor identifies the cursor type.

Pascal

```
TCursor = -32768..32767;
```

C++

```
-32768..32767 TCursor;
```

File

Controls

Description

TCursor type defines a cursor ident (integer). It may have one of the following values :

Value	Meaning
crDefault (see page 6362)	Default cursor : thikc arrow pointing up and left.
crArrow (see page 6361)	Same as the default cursor
crCross (see page 6361)	Fine cross shaped cursor used in graphic applications for precise positioning.
crIBeam (see page 6365)	Text insert cursor in the form of a thin capital I.
crSizeNESW (see page 6368)	Resizing cursor from North-East to South-West resizing.
crSizeNS (see page 6369)	Vertical resizing cursor.
crSizeNWSE (see page 6369)	Resizing cursor from North-West to South-East resizing.
crSizeWE (see page 6370)	Horizontal resizing cursor.
crUpArrow (see page 6371)	Thin upward pointing cursor.

crHourGlass (see page 6364)	Hour glass cursor to indicate a busy process.
crDrag (see page 6362)	Drag cursor for single items being dragged.
crNoDrop (see page 6366)	Whit cross in a white circle indicating that a drag operation is hovering over an invalid drop target.
crHSplit (see page 6364)	Cursor shown when the mouse is over a Horizontal splitter.
crVSplit (see page 6371)	Cursor shown when the mouse is over a Veryical splitter.
crMultiDrag (see page 6365)	Drag cursor for multiple items being dragged.
crSQLWait (see page 6370)	Hour glass cursor to indicate a busy SQL database operation.
crNo (see page 6366)	Black cross in a black circle indicating invalid mouse target location.
crAppStart (see page 6360)	Hour glass and standard pointer combination cursor show at application startup.
crHelp (see page 6363)	Question mark and standard pointer operation
crHandPoint (see page 6363)	An upward pointing hand cursor. This is normally used to identify a selectable item, such as a web page link.
crSizeAll (see page 6368)	Cursor for resizing up, down, left and right.

Use the Screen Cursor property to set the cursor type to one of the TCursor values.

See Also

Cursor (see page 14209)

32.5.35 Controls.TDate Type

TDate is used to represent a date.

Pascal

```
TDate = type TDateTime;
```

C++

```
type TDateTime TDate;
```

File

Controls

Description

TDate represents a special type of TDateTime value that has no decimal part. A TDate value represents the number of days that have elapsed since 12/30/1899.

See Also

TTimeStamp (see page 23009)

32.5.36 Controls.TDockDropEvent Type

TDockDropEvent is the type of the OnDockDrop event handler.

Pascal

```
TDockDropEvent = procedure (Sender: TObject; Source: TDragDockObject; X, Y: Integer) of object;
```

C++

```
(Sender: TObject; Source: TDragDockObject; X, Y: Integer) ( TDockDropEvent)();
```

File

Controls

Description

Sender is the object on which another object is being dropped.

Source is information about the object that is being dropped.

X and Y are the coordinates of the mouse pointer.

32.5.37 Controls.TDockOverEvent Type

TDockOverEvent is the type of the OnDockOver event handler.

Pascal

```
TDockOverEvent = procedure (Sender: TObject; Source: TDragDockObject; X, Y: Integer; State:
TDragState; var Accept: Boolean) of object;
```

C++

```
(Sender: TObject; Source: TDragDockObject; X, Y: Integer; State: TDragState; var Accept:
Boolean) ( TDockOverEvent)();
```

File

Controls

Description

Sender is the object over which another object is being dragged.

Source is information about the object that is being dragged.

X and Y are the coordinates of the mouse pointer.

DragState is the current drag state.

Accept returns whether or not the control accepts the dragged object for docking.

32.5.38 Controls.TDockTreeClass Type

TDockTreeClass defines the metaclass for TDockTree (see page 5983).

Pascal

```
TDockTreeClass = class of TDockTree;
```

C++

```
TDockTree TDockTreeClass;
```

File

Controls

Description

TDockTreeClass is the metaclass for TDockTree (see page 5983). Its value is the class reference for TDockTree (see page 5983) or for one of its descendants.

See Also

TDockTree (see page 5983)

32.5.39 Controls.TDragDropEvent Type

Pascal

```
TDragDropEvent = procedure (Sender, Source: TObject; X, Y: Integer) of object;
```

C++

```
(Sender, Source: TObject; X, Y: Integer) ( TDragDropEvent)();
```

File

Controls

Description

This is type Controls.TDragDropEvent.

32.5.40 Controls.TDragObjectClass Type

TDragObjectClass defines the metaclass for TDragObject (see page 6031).

Pascal

```
TDragObjectClass = class of TDragObject;
```

C++

```
TDragObject TDragObjectClass;
```

File

Controls

Description

TDragObjectClass is the metaclass for TDragObject (see page 6031). Its value is the class reference for TDragObject (see page 6031) or for one of its descendants.

See Also

TDragObject (see page 6031)

32.5.41 Controls.TDragOverEvent Type

Pascal

```
TDragOverEvent = procedure (Sender, Source: TObject; X, Y: Integer; State: TDragState; var
Accept: Boolean) of object;
```

C++

```
(Sender, Source: TObject; X, Y: Integer; State: TDragState; var Accept: Boolean) (
TDragOverEvent)();
```

File

Controls

Description

This is type Controls.TDragOverEvent.

32.5.42 Controls.TEndDragEvent Type

Pascal

```
TEndDragEvent = procedure (Sender, Target: TObject; X, Y: Integer) of object;
```

C++

```
(Sender, Target: TObject; X, Y: Integer) ( TEndDragEvent)();
```

File

Controls

Description

This is type Controls.TEndDragEvent.

32.5.43 Controls.TForEachZoneProc Type

Pascal

```
TForEachZoneProc = procedure (Zone: TDockZone) of object;
```

C++

```
(Zone: TDockZone) ( TForEachZoneProc)();
```

File

Controls

Description

This is type Controls.TForEachZoneProc.

32.5.44 Controls.TGetSiteInfoEvent Type

TGetSiteInfoEvent is the type of the OnGetSiteInfo event handler.

Pascal

```
TGetSiteInfoEvent = procedure (Sender: TObject; DockClient: TControl; var InfluenceRect:
TRect; MousePos: TPoint; var CanDock: Boolean) of object;
```

C++

```
(Sender: TObject; DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint; var
CanDock: Boolean) ( TGetSiteInfoEvent)();
```

File

Controls

Description

Sender is the dock site.

DockClient is the control being dragged.

InfluenceRect is the rectangle in which the mouse must be positioned for the dock site to accept a dragged control. By default, InfluenceRect is the area of the target control, expanded by 10 pixels on each side. This can be adjusted by the event handler.

MousePos is the position of the mouse pointer.

CanDock returns whether the dragged control can be docked to the dock site.

32.5.45 Controls.THintWindowClass Type

THintWindowClass defines the metaclass for THintWindow (see page 6055).

Pascal

```
THintWindowClass = class of THintWindow;
```

C++

```
THintWindow THintWindowClass;
```

File

Controls

Description

THintWindowClass is the metaclass for THintWindow (see page 6055). Its value is the class reference for THintWindow (see page 6055) or for one of its descendants.

See Also

THintWindow (see page 6055)

32.5.46 Controls.TImeName Type

TImeName is the type that represents the name of an Input Method Editor.

Pascal

```
TImeName = type string;
```

C++

```
type string TImeName;
```

File

Controls

Description

TImeName is the string that identifies an IME.

32.5.47 Controls.TKeyEvent Type

Pascal

```
TKeyEvent = procedure (Sender: TObject; var Key: Word; Shift: TShiftState) of object;
```

C++

```
(Sender: TObject; var Key: Word; Shift: TShiftState) ( TKeyEvent)();
```

File

Controls

Description

This is type Controls.TKeyEvent.

32.5.48 Controls.TKeyPressEvent Type

Pascal

```
TKeyPressEvent = procedure (Sender: TObject; var Key: Char) of object;
```

C++

```
(Sender: TObject; var Key: Char) ( TKeyPressEvent)();
```

File

Controls

Description

This is type Controls.TKeyPressEvent.

32.5.49 Controls.TMarginSize Type

Pascal

```
TMarginSize = 0..MaxInt;
```

C++

```
0..MaxInt TMarginSize;
```

File

Controls

Description

This is type Controls.TMarginSize.

32.5.50 Controls.TModalResult Type

TModalResult represents the return value from a modal dialog.

Pascal

```
TModalResult = Low(Integer)..High(Integer);
```

C++

```
Low(Integer)..High(Integer) TModalResult;
```

File

Controls

Description

TModalResult represents the value returned by a modal dialog. An application can use any integer value as a modal result value. Although TModalResult can take any integer value, the following constants are defined for commonly used TModalResult values:

Constant	Value	Meaning
mrNone (see page 6374)	0	None. Used as a default value before the user exits.
mrOk (see page 6375)	idOK	The user exited with OK button.
mrCancel (see page 6373)	idCancel	The user exited with the CANCEL button.
mrAbort (see page 6372)	idAbort	The user exited with the ABORT button.
mrRetry (see page 6375)	idRetry	The user exited with the RETRY button.
mrIgnore (see page 6373)	idIgnore	The user exited with the IGNORE button.
mrYes (see page 6376)	idYes	The user exited with the YES button.

mrNo (see page 6374)	idNo	The user exited with the NO button.
mrAll (see page 6372)	mrNo (see page 6374) + 1	The user exited with the ALL button.
mrNoToAll (see page 6375)	mrAll (see page 6372) + 1	The user exited with the NO TO ALL button.
mrYesToAll (see page 6376)	mrNoToAll (see page 6375) + 1	The user exited with the YES TO ALL button.

See Also

TCustomForm (see page 13849)

TDBEngineErrorDlg

TButton (see page 20909)

IsAbortResult (see page 6258)

IsAnALlResult

IsNegativeResult (see page 6260)

IsPositiveResult (see page 6260)

StripAllFromResult (see page 6265)

32.5.51 Controls.TMouseActivateEvent Type

Pascal

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

C++

```
(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer; HitTest: Integer;
var MouseActivate: TMouseActivate) ( TMouseActivateEvent)();
```

File

Controls

Description

This is type Controls.TMouseActivateEvent.

32.5.52 Controls.TMouseEvent Type

Pascal

```
TMouseEvent = procedure (Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer) of object;
```

C++

```
(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer) ( TMouseEvent)();
```

File

Controls

Description

This is type Controls.TMouseEvent.

32.5.53 Controls.TMouseMoveEvent Type

Pascal

```
TMouseMoveEvent = procedure (Sender: TObject; Shift: TShiftState; X, Y: Integer) of object;
```

C++

```
(Sender: TObject; Shift: TShiftState; X, Y: Integer) ( TMouseMoveEvent)();
```

File

Controls

Description

This is type Controls.TMouseMoveEvent.

32.5.54 Controls.TMouseWheelEvent Type

Pascal

```
TMouseWheelEvent = procedure (Sender: TObject; Shift: TShiftState; WheelDelta: Integer;
MousePos: TPoint; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; Shift: TShiftState; WheelDelta: Integer; MousePos: TPoint; var Handled:
Boolean) ( TMouseWheelEvent)();
```

File

Controls

Description

This is type Controls.TMouseWheelEvent.

32.5.55 Controls.TMouseWheelUpDownEvent Type

Pascal

```
TMouseWheelUpDownEvent = procedure (Sender: TObject; Shift: TShiftState; MousePos: TPoint; var
Handled: Boolean) of object;
```

C++

```
(Sender: TObject; Shift: TShiftState; MousePos: TPoint; var Handled: Boolean) (
TMouseWheelUpDownEvent)();
```

File

Controls

Description

This is type Controls.TMouseWheelUpDownEvent.

32.5.56 Controls.TPanningWindowClass Type

TPanningWindowClass defines the metaclass for TPanningWindow.

Pascal

```
TPanningWindowClass = class of TCustomPanningWindow;
```

C++

```
TCustomPanningWindow TPanningWindowClass;
```

File

Controls

Description

TPanningWindowClass is the metaclass for TPanningWindow. Its value is the class reference for TPanningWindow or for one of its descendants.

See Also

TPanningWindow

32.5.57 Controls.TScalingFlags Type

TScalingFlags describes the way controls are scaled.

Pascal

```
TScalingFlags = set of (sfLeft, sfTop, sfWidth, sfHeight, sfFont, sfDesignSize);
```

C++

```
(sfLeft, sfTop, sfWidth, sfHeight, sfFont, sfDesignSize) TScalingFlags;
```

File

Controls

Description

TScalingFlags describes a set of flags used in scaling inherited forms and their controls.

Flag	Meaning
sfLeft	Left side of control not yet scaled.

sfTop	Top of control not yet scaled.
sfWidth	Width of control not yet scaled.
sfHeight	Height of control not yet scaled.
sfFont	Font of control not yet scaled.
sfDesignSize	Design-time dimensions of control not yet scaled.

See Also

TControl

32.5.58 Controls.TStartDockEvent Type

TStartDockEvent is the type of the OnStartDock event handler.

Pascal

```
TStartDockEvent = procedure (Sender: TObject; var DragObject: TDragDockObject) of object;
```

C++

```
(Sender: TObject; var DragObject: TDragDockObject) ( TStartDockEvent)();
```

File

Controls

Description

Sender is the object on which a control is being docked.

DragObject is information about the control that is being docked.

See Also

OnStartDock

32.5.59 Controls.TStartDragEvent Type

Pascal

```
TStartDragEvent = procedure (Sender: TObject; var DragObject: TDragObject) of object;
```

C++

```
(Sender: TObject; var DragObject: TDragObject) ( TStartDragEvent)();
```

File

Controls

Description

This is type Controls.TStartDragEvent.

32.5.60 Controls.TTabOrder Type

TTabOrder defines the tab order of a control.

Pascal

```
TTabOrder = -1..32767;
```

C++

```
-1..32767 TTabOrder;
```

File

Controls

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears. Subsequent controls in the tab sequence have value 1, 2, 3 and so on.

TTabOrder is the data type used by the TabOrder property.

See Also

TabOrder (see page 6211)

32.5.61 Controls.TTime Type

TTime is used to represent a time.

Pascal

```
TTime = type TDateTime;
```

C++

```
type TDateTime TTime;
```

File

Controls

Description

TTime is simply a TDateTime type.

See Also

TDateTime (see page 22612)

32.5.62 Controls.TUnDockEvent Type

TUnDockEvent is the type of the OnUnDock event handler.

Pascal

```
TUnDockEvent = procedure (Sender: TObject; Client: TControl; NewTarget: TWinControl; var
Allow: Boolean) of object;
```

C++

```
(Sender: TObject; Client: TControl; NewTarget: TWinControl; var Allow: Boolean) (
TUnDockEvent)();
```

File

Controls

Description

Sender is the object from which a control is being undocked.

Client is the control that is being undocked.

NewTarget is the control to which Client is being docked.

Allow is whether to allow the undocking.

See Also

OnUnDock

32.5.63 Controls.TWinControlActionLinkClass Type

TWinControlActionLinkClass defines the metaclass for TWinControlActionLink (see page 6233).

Pascal

```
TWinControlActionLinkClass = class of TWinControlActionLink;
```

C++

```
TWinControlActionLink TWinControlActionLinkClass;
```

File

Controls

Description

TWinControlActionLinkClass is the metaclass for TWinControlActionLink (see page 6233). Its value is the class reference for TWinControlActionLink (see page 6233) or for one of its descendants.

See Also

TWinControlActionLink (see page 6233)

32.5.64 Controls.TWinControlClass Type

TWinControlClass defines the metaclass for TWinControl (see page 6112).

Pascal

```
TWinControlClass = class of TWinControl;
```

C++

```
TWinControl * TWinControlClass;
```

File

Controls

Description

TWinControlClass is the metaclass for TWinControl (see page 6112). Its value is the class reference for TWinControl (see page 6112) or for one of its descendants.

32.5.65 Controls.TWndMethod Type

TWndMethod is the type for Windows message procedures.

Pascal

```
TWndMethod = Classes.TWndMethod;
```

C++

```
Classes.TWndMethod TWndMethod;
```

File

Controls

Description

Each Window has a message procedure that responds to calls from Windows. These calls can originate from Windows (such as paint messages), from user actions (such as mouse messages), or from your messages that you send in your own code.

TWndMethod is the type for VCL-based message procedures. Message procedures of this type can be assigned to controls using the WindowProc method. In addition, a window that implements this procedure but which is not associated with a control can be created using the AllocateHWnd function.

The Message parameter encapsulates the message to which the procedure responds. It contains fields for the incoming message parameters and a result field that holds the return value.

See Also

TControl (see page 5724)

AllocateHWnd (see page 3217)

32.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
AnimateWindowProc (see page 6320)	This is variable Controls.AnimateWindowProc.
CreationControl (see page 6320)	This is variable Controls.CreationControl.
DefaultDockTreeClass (see page 6321)	Provides the default implementation class for the DockManager property of container controls.
IsVCLControlHook (see page 6321)	This is variable Controls.IsVCLControlHook.
Mouse (see page 6321)	Contains information about the system's mouse.
NewStyleControls (see page 6322)	Determines whether controls are drawn using the Windows 3.x "look".

32.6.1 Controls.AnimateWindowProc Variable

Pascal

```
AnimateWindowProc: TAnimateWindowProc = nil;
```

C++

```
TAnimateWindowProc AnimateWindowProc = nil;
```

File

Controls

Description

This is variable Controls.AnimateWindowProc.

32.6.2 Controls.CreationControl Variable

Pascal

```
CreationControl: TWinControl = nil;
```

C++

```
TWinControl * CreationControl = nil;
```

File

Controls

Description

This is variable Controls.CreationControl.

32.6.3 Controls.DefaultDockTreeClass Variable

Provides the default implementation class for the DockManager property of container controls.

Pascal

```
DefaultDockTreeClass: TDockTreeClass = TDockTree;
```

C++

```
TDockTreeClass DefaultDockTreeClass = TDockTree;
```

File

Controls

Description

When you set the UseDockManager property of a container control to true, it uses the value of its DockManager property to manage docking operations. If the application does not explicitly assign a value to DockManager, the control generates a dock manager by instantiating the class specified by DefaultDockTreeClass.

The Controls (see page 5711) unit sets DefaultDockTreeClass to TDockTree (see page 5983). Change this value to provide a different default implementation of the dock manager that container controls use when the DockManager property is not explicitly assigned. The class specified by DefaultDockTreeClass must be a descendant of TDockTree (see page 5983).

See Also

IDockManager (see page 6238)

TWinControl (see page 6112)

TWinControl (see page 6112)

TDockTree (see page 5983)

32.6.4 Controls.IsVCLControlHook Variable

Pascal

```
IsVCLControlHook: function (Handle: HWnd): Boolean of object;
```

C++

```
function (Handle: HWnd): Boolean of object IsVCLControlHook;
```

File

Controls

Description

This is variable Controls.IsVCLControlHook.

32.6.5 Controls.Mouse Variable

Contains information about the system's mouse.

Pascal

```
Mouse: TMouse;
```

C++

```
TMouse Mouse;
```

File

Controls

Description

Use Mouse to get information about the system's mouse, including which window (if any) has captured the mouse, the position of the mouse, and mouse settings from the Control panel. Mouse can also be used to specify global aspects of how your application responds to mouse messages, such as how much the user must move the mouse to start a drag operation.

32.6.6 Controls.NewStyleControls Variable

Determines whether controls are drawn using the Windows 3.x "look".

Pascal

```
NewStyleControls: Boolean;
```

C++

```
Boolean NewStyleControls;
```

File

Controls

Description

NewStyleControls determines whether controls in the application are drawn using the effects that were introduced with Windows 95. When NewStyleControls is false, controls have a flatter look.

By default, NewStyleControls is true unless the application is running on a Windows 3.x system with Win32S. To change this value to false, set the value of NewStyleControls in the project file, before the call to the Application object's Run method.

32.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
AnchorAlign (see page 6325)	Defines the default sets of anchor types for all component alignment values.
CM_ACTIONEXECUTE (see page 6326)	This is constant Controls.CM_ACTIONEXECUTE.
CM_ACTIONUPDATE (see page 6327)	This is constant Controls.CM_ACTIONUPDATE.
CM_ACTIVATE (see page 6327)	This is constant Controls.CM_ACTIVATE.
CM_ALLCHILDRENFLIPPED (see page 6327)	This is constant Controls.CM_ALLCHILDRENFLIPPED.
CM_APPKEYDOWN (see page 6328)	This is constant Controls.CM_APPKEYDOWN.
CM_APPSYSCOMMAND (see page 6328)	This is constant Controls.CM_APPSYSCOMMAND.
CM_BASE (see page 6328)	This is constant Controls.CM_BASE.
CM_BIDIMODECHANGED (see page 6329)	This is constant Controls.CM_BIDIMODECHANGED.
CM_BORDERCHANGED (see page 6329)	This is constant Controls.CM_BORDERCHANGED.
CM_BUTTONPRESSED (see page 6329)	This is constant Controls.CM_BUTTONPRESSED.
CM_CANCELMODE (see page 6330)	This is constant Controls.CM_CANCELMODE.
CM_CHANGED (see page 6330)	This is constant Controls.CM_CHANGED.
CM_CHILDKEY (see page 6330)	This is constant Controls.CM_CHILDKEY.
CM_COLORCHANGED (see page 6331)	This is constant Controls.CM_COLORCHANGED.
CM_CONTROLCHANGE (see page 6331)	This is constant Controls.CM_CONTROLCHANGE.
CM_CONTROLLISTCHANGE (see page 6331)	This is constant Controls.CM_CONTROLLISTCHANGE.
CM_CONTROLLISTCHANGING (see page 6332)	This is constant Controls.CM_CONTROLLISTCHANGING.
CM_CREATEPOPUP (see page 6332)	This is constant Controls.CM_CREATEPOPUP.
CM_CTL3DCHANGED (see page 6332)	This is constant Controls.CM_CTL3DCHANGED.
CM_CTLCOLOREDITGLASS (see page 6333)	This is constant Controls.CM_CTLCOLOREDITGLASS.
CM_CURSORCHANGED (see page 6333)	This is constant Controls.CM_CURSORCHANGED.
CM_DEACTIVATE (see page 6333)	This is constant Controls.CM_DEACTIVATE.
CM_DESIGNHITTEST (see page 6334)	This is constant Controls.CM_DESIGNHITTEST.
CM_DESTROYHANDLE (see page 6334)	This is constant Controls.CM_DESTROYHANDLE.
CM_DIALOGCHAR (see page 6334)	This is constant Controls.CM_DIALOGCHAR.
CM_DIALOGHANDLE (see page 6335)	This is constant Controls.CM_DIALOGHANDLE.
CM_DIALOGKEY (see page 6335)	This is constant Controls.CM_DIALOGKEY.
CM_DOCKCLIENT (see page 6335)	This is constant Controls.CM_DOCKCLIENT.
CM_DOCKNOTIFICATION (see page 6336)	This is constant Controls.CM_DOCKNOTIFICATION.
CM_DOCWINDOWACTIVATE (see page 6336)	This is constant Controls.CM_DOCWINDOWACTIVATE.
CM_DRAG (see page 6336)	This is constant Controls.CM_DRAG.
CM_ENABLEDCHANGED (see page 6337)	This is constant Controls.CM_ENABLEDCHANGED.
CM_ENTER (see page 6337)	This is constant Controls.CM_ENTER.
CM_EXIT (see page 6337)	This is constant Controls.CM_EXIT.
CM_FLOAT (see page 6338)	This is constant Controls.CM_FLOAT.
CM_FOCUSCHANGED (see page 6338)	This is constant Controls.CM_FOCUSCHANGED.
CM_FONTCHANGE (see page 6338)	This is constant Controls.CM_FONTCHANGE.
CM_FONTCHANGED (see page 6339)	This is constant Controls.CM_FONTCHANGED.
CM_GETDATALINK (see page 6339)	This is constant Controls.CM_GETDATALINK.
CM_GOTFOCUS (see page 6339)	This is constant Controls.CM_GOTFOCUS.
CM_HINTSHOW (see page 6340)	This is constant Controls.CM_HINTSHOW.
CM_HINTSHOWPAUSE (see page 6340)	This is constant Controls.CM_HINTSHOWPAUSE.
CM_HITTEST (see page 6340)	This is constant Controls.CM_HITTEST.

CM_ICONCHANGED (see page 6341)	This is constant Controls.CM_ICONCHANGED.
CM_INVALIDATE (see page 6341)	This is constant Controls.CM_INVALIDATE.
CM_INVALIDATEDOCKHOST (see page 6341)	This is constant Controls.CM_INVALIDATEDOCKHOST.
CM_INVOKEHELP (see page 6342)	This is constant Controls.CM_INVOKEHELP.
CM_ISSHORTCUT (see page 6342)	This is constant Controls.CM_ISSHORTCUT.
CM_ISTOOLCONTROL (see page 6342)	This is constant Controls.CM_ISTOOLCONTROL.
CM_LOSTFOCUS (see page 6343)	This is constant Controls.CM_LOSTFOCUS.
CM_MENUCHANGED (see page 6343)	This is constant Controls.CM_MENUCHANGED.
CM_MOUSEACTIVATE (see page 6343)	This is constant Controls.CM_MOUSEACTIVATE.
CM_MOUSEENTER (see page 6344)	This is constant Controls.CM_MOUSEENTER.
CM_MOUSELEAVE (see page 6344)	This is constant Controls.CM_MOUSELEAVE.
CM_MOUSEWHEEL (see page 6344)	This is constant Controls.CM_MOUSEWHEEL.
CM_PARENTBIDIMODECHANGED (see page 6345)	This is constant Controls.CM_PARENTBIDIMODECHANGED.
CM_PARENTCOLORCHANGED (see page 6345)	This is constant Controls.CM_PARENTCOLORCHANGED.
CM_PARENTCTL3DCHANGED (see page 6345)	This is constant Controls.CM_PARENTCTL3DCHANGED.
CM_PARENTFONTCHANGED (see page 6346)	This is constant Controls.CM_PARENTFONTCHANGED.
CM_PARENTSHOWHINTCHANGED (see page 6346)	This is constant Controls.CM_PARENTSHOWHINTCHANGED.
CM_POPUPHWNDDESTROY (see page 6346)	This is constant Controls.CM_POPUPHWNDDESTROY.
CM_RECREATEWND (see page 6347)	This is constant Controls.CM_RECREATEWND.
CM_RELEASE (see page 6347)	This is constant Controls.CM_RELEASE.
CM_SETACTIVECONTROL (see page 6347)	This is constant Controls.CM_SETACTIVECONTROL.
CM_SHOWHINTCHANGED (see page 6348)	This is constant Controls.CM_SHOWHINTCHANGED.
CM_SHOWINGCHANGED (see page 6348)	This is constant Controls.CM_SHOWINGCHANGED.
CM_SYSCOLORCHANGE (see page 6348)	This is constant Controls.CM_SYSCOLORCHANGE.
CM_SYSFONTCHANGED (see page 6349)	This is constant Controls.CM_SYSFONTCHANGED.
CM_TABSTOPCHANGED (see page 6349)	This is constant Controls.CM_TABSTOPCHANGED.
CM_TEXTCHANGED (see page 6349)	This is constant Controls.CM_TEXTCHANGED.
CM_TIMECHANGE (see page 6350)	This is constant Controls.CM_TIMECHANGE.
CM_UIACTIVATE (see page 6350)	This is constant Controls.CM_UIACTIVATE.
CM_UIDEACTIVATE (see page 6350)	This is constant Controls.CM_UIDEACTIVATE.
CM_UNDOCKCLIENT (see page 6351)	This is constant Controls.CM_UNDOCKCLIENT.
CM_VISIBLECHANGED (see page 6351)	This is constant Controls.CM_VISIBLECHANGED.
CM_WANTSPECIALKEY (see page 6351)	This is constant Controls.CM_WANTSPECIALKEY.
CM_WINDOWHOOK (see page 6352)	This is constant Controls.CM_WINDOWHOOK.
CM_WININICHANGE (see page 6352)	This is constant Controls.CM_WININICHANGE.
CN_BASE (see page 6352)	This is constant Controls.CN_BASE.
CN_CHAR (see page 6353)	This is constant Controls.CN_CHAR.
CN_CHARTOITEM (see page 6353)	This is constant Controls.CN_CHARTOITEM.
CN_COMMAND (see page 6353)	This is constant Controls.CN_COMMAND.
CN_COMPAREITEM (see page 6354)	This is constant Controls.CN_COMPAREITEM.
CN_CTLCOLORBTN (see page 6354)	This is constant Controls.CN_CTLCOLORBTN.
CN_CTLCOLORDLG (see page 6354)	This is constant Controls.CN_CTLCOLORDLG.
CN_CTLCOLOREDIT (see page 6355)	This is constant Controls.CN_CTLCOLOREDIT.
CN_CTLCOLORLISTBOX (see page 6355)	This is constant Controls.CN_CTLCOLORLISTBOX.
CN_CTLCOLORMSGBOX (see page 6355)	This is constant Controls.CN_CTLCOLORMSGBOX.
CN_CTLCOLORSCROLLBAR (see page 6356)	This is constant Controls.CN_CTLCOLORSCROLLBAR.
CN_CTLCOLORSTATIC (see page 6356)	This is constant Controls.CN_CTLCOLORSTATIC.
CN_DELETEITEM (see page 6356)	This is constant Controls.CN_DELETEITEM.
CN_DRAWITEM (see page 6357)	This is constant Controls.CN_DRAWITEM.
CN_HSCROLL (see page 6357)	This is constant Controls.CN_HSCROLL.
CN_KEYDOWN (see page 6357)	This is constant Controls.CN_KEYDOWN.
CN_KEYUP (see page 6358)	This is constant Controls.CN_KEYUP.
CN_MEASUREITEM (see page 6358)	This is constant Controls.CN_MEASUREITEM.
CN_NOTIFY (see page 6358)	This is constant Controls.CN_NOTIFY.

CN_PARENTNOTIFY (see page 6359)	This is constant Controls.CN_PARENTNOTIFY.
CN_SYSCHAR (see page 6359)	This is constant Controls.CN_SYSCHAR.
CN_SYSKEYDOWN (see page 6359)	This is constant Controls.CN_SYSKEYDOWN.
CN_VKEYTOITEM (see page 6360)	This is constant Controls.CN_VKEYTOITEM.
CN_VSCROLL (see page 6360)	This is constant Controls.CN_VSCROLL.
crAppStart (see page 6360)	Identifies the default cursor.
crArrow (see page 6361)	Identifies an arrow cursor.
crCross (see page 6361)	Identifies an cross cursor.
crDefault (see page 6362)	Identifies the default cursor.
crDrag (see page 6362)	Identifies a drag cursor.
crHandPoint (see page 6363)	Identifies a pointing hand cursor.
crHelp (see page 6363)	Identifies a help cursor.
crHourGlass (see page 6364)	Identifies an hour glass cursor.
crHSplit (see page 6364)	Identifies a horizontal split cursor.
crIBeam (see page 6365)	Identifies a text insert cursor.
crMultiDrag (see page 6365)	Identifies a multiple item drag cursor.
crNo (see page 6366)	Identifies a not allowed cursor.
crNoDrop (see page 6366)	Identifies a not allowed to drop cursor.
crNone (see page 6367)	Identifies an invisible cursor.
crSize (see page 6367)	Identifies an obsolete cursor.
crSizeAll (see page 6368)	Identifies a size in all directions cursor.
crSizeNESW (see page 6368)	Identifies a North-East to South-West sizing cursor.
crSizeNS (see page 6369)	Identifies a vertical sizing cursor.
crSizeNWSE (see page 6369)	Identifies a North-West to South-East sizing cursor.
crSizeWE (see page 6370)	Identifies a horizontal sizing cursor.
crSQLWait (see page 6370)	Identifies an SQL wait hour glass cursor.
crUpArrow (see page 6371)	Identifies an upwards pointing arrow cursor.
crVSplit (see page 6371)	Identifies a vertical split cursor.
CTL3D_ALL (see page 6372)	This is constant Controls.CTL3D_ALL.
mrAbort (see page 6372)	Indicates that Abort was selected in a modal dialog.
mrAll (see page 6372)	This is constant Controls.mrAll.
mrCancel (see page 6373)	Indicates that Cancel was selected in a modal dialog.
mrIgnore (see page 6373)	Indicates that Ignore was selected in a modal dialog.
mrNo (see page 6374)	Indicates that No was selected in a modal dialog.
mrNone (see page 6374)	The default modal dialog return value.
mrNoToAll (see page 6375)	Indicates that 'No To All' was selected in a modal dialog.
mrOk (see page 6375)	This is constant Controls.mrOk.
mrRetry (see page 6375)	Indicates that Retry was selected in a modal dialog.
mrYes (see page 6376)	Indicates that Yes was selected in a modal dialog.
mrYesToAll (see page 6376)	Indicates that 'Yes To All' was selected in a modal dialog.
NullDockSite (see page 6377)	This is constant Controls.NullDockSite.

32.7.1 Controls.AnchorAlign Constant

Defines the default sets of anchor types for all component alignment values.

Pascal

```
AnchorAlign: array[TAlign] of TAnchors = ( [akLeft, akTop], [akLeft, akTop, akRight], [akLeft,
akRight, akBottom], [akLeft, akTop, akBottom], [akRight, akTop, akBottom], [akLeft, akTop,
akRight, akBottom], [akLeft, akTop] );
```

C++

```
array[TAlign] of TAnchors AnchorAlign = ( [akLeft, akTop], [akLeft, akTop, akRight], [akLeft,
```

```
akRight, akBottom], [akLeft, akTop, akBottom], [akRight, akTop, akBottom], [akLeft, akTop,
akRight, akBottom], [akLeft, akTop] );
```

File

Controls

Description

Component alignment and anchoring are closely related. Setting an anchor to true means that a side of a component will remain the designed distance from the edge of the container at run time. Alignment simply forces this distance to be zero at design time (if we disregard margin and padding values).

For example, a component with Align set to alLeft in the Object Inspector will hug the left side of its container regardless of any effort to reposition it. You can, however, adjust its width since the right side position is not forced to align with the right side of the container - the akRight anchor value is false.

AnchorAlign defines the default set of TAnchorKind (see page 6269) anchor values for each possible TAlign (see page 6268) value :

TAlign (see page 6268) value	TAnchorKind (see page 6269) values set	Meaning
alNone	akLeft, akTop	No design time forced alignment is performed.
alTop	akLeft, akTop, akRight	Component is forced to align with top of the container - along with the left and right.
alBottom	akLeft, akRight, akBottom	Component is forced to align with bottom of the container - along with the left and right.
alLeft	akLeft, akTop, akBottom	Component is forced to align with left of the container - along with the top and bottom.
alRight	akRight, akTop, akBottom	Component is forced to align with right of the container - along with the top and bottom.
alClient	akLeft, akTop, akRight, akBottom	Component is forced to align with all sides of the container. If another control already occupies part of the container, the control resizes to fit within the remaining client area.
alCustom	akLeft, akTop	The control's positioning is determined by calls to its parent's CustomAlignInsertBefore and CustomAlignPosition methods.

Note: In the above, the container is the parent component inside which the current component is being positioned. If that parent already has aligned component(s), the current component respects them. For example, a 2nd alLeft aligned component will align to the immediate right of the first component.

See Also

TAlign (see page 6268)

TAnchorKind (see page 6269)

32.7.2 Controls.CM_ACTIONEXECUTE Constant

Pascal

```
CM_ACTIONEXECUTE = CM_BASE + 64;
```

C++

```
CM_ACTIONEXECUTE = CM_BASE + 64;
```

File

Controls

Description

This is constant Controls.CM_ACTIONEXECUTE.

32.7.3 Controls.CM_ACTIONUPDATE Constant

Pascal

```
CM_ACTIONUPDATE = CM_BASE + 63;
```

C++

```
CM_ACTIONUPDATE = CM_BASE + 63;
```

File

Controls

Description

This is constant Controls.CM_ACTIONUPDATE.

32.7.4 Controls.CM_ACTIVATE Constant

Pascal

```
CM_ACTIVATE = CM_BASE + 0;
```

C++

```
CM_ACTIVATE = CM_BASE + 0;
```

File

Controls

Description

This is constant Controls.CM_ACTIVATE.

32.7.5 Controls.CM_ALLCHILDRENFLIPPED Constant

Pascal

```
CM_ALLCHILDRENFLIPPED = CM_BASE + 62;
```

C++

```
CM_ALLCHILDRENFLIPPED = CM_BASE + 62;
```

File

Controls

Description

This is constant Controls.CM_ALLCHILDRENFLIPPED.

32.7.6 Controls.CM_APPKEYDOWN Constant

Pascal

```
CM_APPKEYDOWN = CM_BASE + 22;
```

C++

```
CM_APPKEYDOWN = CM_BASE + 22;
```

File

Controls

Description

This is constant Controls.CM_APPKEYDOWN.

32.7.7 Controls.CM_APPSYSCOMMAND Constant

Pascal

```
CM_APPSYSCOMMAND = CM_BASE + 23;
```

C++

```
CM_APPSYSCOMMAND = CM_BASE + 23;
```

File

Controls

Description

This is constant Controls.CM_APPSYSCOMMAND.

32.7.8 Controls.CM_BASE Constant

Pascal

```
CM_BASE = $B000;
```

C++

```
CM_BASE = 0xB000;
```

File

Controls

Description

This is constant Controls.CM_BASE.

32.7.9 Controls.CM_BIDIMODECHANGED Constant

Pascal

```
CM_BIDIMODECHANGED = CM_BASE + 60;
```

C++

```
CM_BIDIMODECHANGED = CM_BASE + 60;
```

File

Controls

Description

This is constant Controls.CM_BIDIMODECHANGED.

32.7.10 Controls.CM_BORDERCHANGED Constant

Pascal

```
CM_BORDERCHANGED = CM_BASE + 59;
```

C++

```
CM_BORDERCHANGED = CM_BASE + 59;
```

File

Controls

Description

This is constant Controls.CM_BORDERCHANGED.

32.7.11 Controls.CM_BUTTONPRESSED Constant

Pascal

```
CM_BUTTONPRESSED = CM_BASE + 24;
```

C++

```
CM_BUTTONPRESSED = CM_BASE + 24;
```

File

Controls

Description

This is constant Controls.CM_BUTTONPRESSED.

32.7.12 Controls.CM_CANCELMODE Constant

Pascal

```
CM_CANCELMODE = CM_BASE + 4;
```

C++

```
CM_CANCELMODE = CM_BASE + 4;
```

File

Controls

Description

This is constant Controls.CM_CANCELMODE.

32.7.13 Controls.CM_CHANGED Constant

Pascal

```
CM_CHANGED = CM_BASE + 55;
```

C++

```
CM_CHANGED = CM_BASE + 55;
```

File

Controls

Description

This is constant Controls.CM_CHANGED.

32.7.14 Controls.CM_CHILDKEY Constant

Pascal

```
CM_CHILDKEY = CM_BASE + 46;
```

C++

```
CM_CHILDKEY = CM_BASE + 46;
```

File

Controls

Description

This is constant Controls.CM_CHILDKEY.

32.7.15 Controls.CM_COLORCHANGED Constant

Pascal

```
CM_COLORCHANGED = CM_BASE + 13;
```

C++

```
CM_COLORCHANGED = CM_BASE + 13;
```

File

Controls

Description

This is constant Controls.CM_COLORCHANGED.

32.7.16 Controls.CM_CONTROLCHANGE Constant

Pascal

```
CM_CONTROLCHANGE = CM_BASE + 54;
```

C++

```
CM_CONTROLCHANGE = CM_BASE + 54;
```

File

Controls

Description

This is constant Controls.CM_CONTROLCHANGE.

32.7.17 Controls.CM_CONTROLLISTCHANGE Constant

Pascal

```
CM_CONTROLLISTCHANGE = CM_BASE + 44;
```

C++

```
CM_CONTROLLISTCHANGE = CM_BASE + 44;
```

File

Controls

Description

This is constant Controls.CM_CONTROLLISTCHANGE.

32.7.18 Controls.CM_CONTROLLISTCHANGING Constant

Pascal

```
CM_CONTROLLISTCHANGING = CM_BASE + 76;
```

C++

```
CM_CONTROLLISTCHANGING = CM_BASE + 76;
```

File

Controls

Description

This is constant Controls.CM_CONTROLLISTCHANGING.

32.7.19 Controls.CM_CREATEPOPUP Constant

Pascal

```
CM_CREATEPOPUP = CM_BASE + 73;
```

C++

```
CM_CREATEPOPUP = CM_BASE + 73;
```

File

Controls

Description

This is constant Controls.CM_CREATEPOPUP.

32.7.20 Controls.CM_CTL3DCHANGED Constant

Pascal

```
CM_CTL3DCHANGED = CM_BASE + 16;
```

C++

```
CM_CTL3DCHANGED = CM_BASE + 16;
```

File

Controls

Description

This is constant Controls.CM_CTL3DCHANGED.

32.7.21 Controls.CM_CTLCOLOREDITGLASS Constant

Pascal

```
CM_CTLCOLOREDITGLASS = CM_BASE + 77;
```

C++

```
CM_CTLCOLOREDITGLASS = CM_BASE + 77;
```

File

Controls

Description

This is constant Controls.CM_CTLCOLOREDITGLASS.

32.7.22 Controls.CM_CURSORCHANGED Constant

Pascal

```
CM_CURSORCHANGED = CM_BASE + 15;
```

C++

```
CM_CURSORCHANGED = CM_BASE + 15;
```

File

Controls

Description

This is constant Controls.CM_CURSORCHANGED.

32.7.23 Controls.CM_DEACTIVATE Constant

Pascal

```
CM_DEACTIVATE = CM_BASE + 1;
```

C++

```
CM_DEACTIVATE = CM_BASE + 1;
```

File

Controls

Description

This is constant Controls.CM_DEACTIVATE.

32.7.24 Controls.CM_DESIGNHITTEST Constant

Pascal

```
CM_DESIGNHITTEST = CM_BASE + 28;
```

C++

```
CM_DESIGNHITTEST = CM_BASE + 28;
```

File

Controls

Description

This is constant Controls.CM_DESIGNHITTEST.

32.7.25 Controls.CM_DESTROYHANDLE Constant

Pascal

```
CM_DESTROYHANDLE = CM_BASE + 74;
```

C++

```
CM_DESTROYHANDLE = CM_BASE + 74;
```

File

Controls

Description

This is constant Controls.CM_DESTROYHANDLE.

32.7.26 Controls.CM_DIALOGCHAR Constant

Pascal

```
CM_DIALOGCHAR = CM_BASE + 6;
```

C++

```
CM_DIALOGCHAR = CM_BASE + 6;
```

File

Controls

Description

This is constant Controls.CM_DIALOGCHAR.

32.7.27 Controls.CM_DIALOGHANDLE Constant

Pascal

```
CM_DIALOGHANDLE = CM_BASE + 49;
```

C++

```
CM_DIALOGHANDLE = CM_BASE + 49;
```

File

Controls

Description

This is constant Controls.CM_DIALOGHANDLE.

32.7.28 Controls.CM_DIALOGKEY Constant

Pascal

```
CM_DIALOGKEY = CM_BASE + 5;
```

C++

```
CM_DIALOGKEY = CM_BASE + 5;
```

File

Controls

Description

This is constant Controls.CM_DIALOGKEY.

32.7.29 Controls.CM_DOCKCLIENT Constant

Pascal

```
CM_DOCKCLIENT = CM_BASE + 56;
```

C++

```
CM_DOCKCLIENT = CM_BASE + 56;
```

File

Controls

Description

This is constant Controls.CM_DOCKCLIENT.

32.7.30 Controls.CM_DOCKNOTIFICATION Constant

Pascal

```
CM_DOCKNOTIFICATION = CM_BASE + 66;
```

C++

```
CM_DOCKNOTIFICATION = CM_BASE + 66;
```

File

Controls

Description

This is constant Controls.CM_DOCKNOTIFICATION.

32.7.31 Controls.CM_DOCWINDOWACTIVATE Constant

Pascal

```
CM_DOCWINDOWACTIVATE = CM_BASE + 43;
```

C++

```
CM_DOCWINDOWACTIVATE = CM_BASE + 43;
```

File

Controls

Description

This is constant Controls.CM_DOCWINDOWACTIVATE.

32.7.32 Controls.CM_DRAG Constant

Pascal

```
CM_DRAG = CM_BASE + 47;
```

C++

```
CM_DRAG = CM_BASE + 47;
```

File

Controls

Description

This is constant Controls.CM_DRAG.

32.7.33 Controls.CM_ENABLEDCHANGED Constant

Pascal

```
CM_ENABLEDCHANGED = CM_BASE + 12;
```

C++

```
CM_ENABLEDCHANGED = CM_BASE + 12;
```

File

Controls

Description

This is constant Controls.CM_ENABLEDCHANGED.

32.7.34 Controls.CM_ENTER Constant

Pascal

```
CM_ENTER = CM_BASE + 26;
```

C++

```
CM_ENTER = CM_BASE + 26;
```

File

Controls

Description

This is constant Controls.CM_ENTER.

32.7.35 Controls.CM_EXIT Constant

Pascal

```
CM_EXIT = CM_BASE + 27;
```

C++

```
CM_EXIT = CM_BASE + 27;
```

File

Controls

Description

This is constant Controls.CM_EXIT.

32.7.36 Controls.CM_FLOAT Constant

Pascal

```
CM_FLOAT = CM_BASE + 58;
```

C++

```
CM_FLOAT = CM_BASE + 58;
```

File

Controls

Description

This is constant Controls.CM_FLOAT.

32.7.37 Controls.CM_FOCUSCHANGED Constant

Pascal

```
CM_FOCUSCHANGED = CM_BASE + 7;
```

C++

```
CM_FOCUSCHANGED = CM_BASE + 7;
```

File

Controls

Description

This is constant Controls.CM_FOCUSCHANGED.

32.7.38 Controls.CM_FONTCHANGE Constant

Pascal

```
CM_FONTCHANGE = CM_BASE + 38;
```

C++

```
CM_FONTCHANGE = CM_BASE + 38;
```

File

Controls

Description

This is constant Controls.CM_FONTCHANGE.

32.7.39 Controls.CM_FONTCHANGED Constant

Pascal

```
CM_FONTCHANGED = CM_BASE + 14;
```

C++

```
CM_FONTCHANGED = CM_BASE + 14;
```

File

Controls

Description

This is constant Controls.CM_FONTCHANGED.

32.7.40 Controls.CM_GETDATALINK Constant

Pascal

```
CM_GETDATALINK = CM_BASE + 45;
```

C++

```
CM_GETDATALINK = CM_BASE + 45;
```

File

Controls

Description

This is constant Controls.CM_GETDATALINK.

32.7.41 Controls.CM_GOTFOCUS Constant

Pascal

```
CM_GOTFOCUS = CM_BASE + 2;
```

C++

```
CM_GOTFOCUS = CM_BASE + 2;
```

File

Controls

Description

This is constant Controls.CM_GOTFOCUS.

32.7.42 Controls.CM_HINTSHOW Constant

Pascal

```
CM_HINTSHOW = CM_BASE + 48;
```

C++

```
CM_HINTSHOW = CM_BASE + 48;
```

File

Controls

Description

This is constant Controls.CM_HINTSHOW.

32.7.43 Controls.CM_HINTSHOWPAUSE Constant

Pascal

```
CM_HINTSHOWPAUSE = CM_BASE + 65;
```

C++

```
CM_HINTSHOWPAUSE = CM_BASE + 65;
```

File

Controls

Description

This is constant Controls.CM_HINTSHOWPAUSE.

32.7.44 Controls.CM_HITTEST Constant

Pascal

```
CM_HITTEST = CM_BASE + 10;
```

C++

```
CM_HITTEST = CM_BASE + 10;
```

File

Controls

Description

This is constant Controls.CM_HITTEST.

32.7.45 Controls.CM_ICONCHANGED Constant

Pascal

```
CM_ICONCHANGED = CM_BASE + 29;
```

C++

```
CM_ICONCHANGED = CM_BASE + 29;
```

File

Controls

Description

This is constant Controls.CM_ICONCHANGED.

32.7.46 Controls.CM_INVALIDATE Constant

Pascal

```
CM_INVALIDATE = CM_BASE + 52;
```

C++

```
CM_INVALIDATE = CM_BASE + 52;
```

File

Controls

Description

This is constant Controls.CM_INVALIDATE.

32.7.47 Controls.CM_INVALIDATEDOCKHOST Constant

Pascal

```
CM_INVALIDATEDOCKHOST = CM_BASE + 70;
```

C++

```
CM_INVALIDATEDOCKHOST = CM_BASE + 70;
```

File

Controls

Description

This is constant Controls.CM_INVALIDATEDOCKHOST.

32.7.48 Controls.CM_INVOKEHELP Constant

Pascal

```
CM_INVOKEHELP = CM_BASE + 31;
```

C++

```
CM_INVOKEHELP = CM_BASE + 31;
```

File

Controls

Description

This is constant Controls.CM_INVOKEHELP.

32.7.49 Controls.CM_ISSHORTCUT Constant

Pascal

```
CM_ISSHORTCUT = CM_BASE + 68;
```

C++

```
CM_ISSHORTCUT = CM_BASE + 68;
```

File

Controls

Description

This is constant Controls.CM_ISSHORTCUT.

32.7.50 Controls.CM_ISTOOLCONTROL Constant

Pascal

```
CM_ISTOOLCONTROL = CM_BASE + 50;
```

C++

```
CM_ISTOOLCONTROL = CM_BASE + 50;
```

File

Controls

Description

This is constant Controls.CM_ISTOOLCONTROL.

32.7.51 Controls.CM_LOSTFOCUS Constant

Pascal

```
CM_LOSTFOCUS = CM_BASE + 3;
```

C++

```
CM_LOSTFOCUS = CM_BASE + 3;
```

File

Controls

Description

This is constant Controls.CM_LOSTFOCUS.

32.7.52 Controls.CM_MENUCHANGED Constant

Pascal

```
CM_MENUCHANGED = CM_BASE + 21;
```

C++

```
CM_MENUCHANGED = CM_BASE + 21;
```

File

Controls

Description

This is constant Controls.CM_MENUCHANGED.

32.7.53 Controls.CM_MOUSEACTIVATE Constant

Pascal

```
CM_MOUSEACTIVATE = CM_BASE + 75;
```

C++

```
CM_MOUSEACTIVATE = CM_BASE + 75;
```

File

Controls

Description

This is constant Controls.CM_MOUSEACTIVATE.

32.7.54 Controls.CM_MOUSEENTER Constant

Pascal

```
CM_MOUSEENTER = CM_BASE + 19;
```

C++

```
CM_MOUSEENTER = CM_BASE + 19;
```

File

Controls

Description

This is constant Controls.CM_MOUSEENTER.

32.7.55 Controls.CM_MOUSELEAVE Constant

Pascal

```
CM_MOUSELEAVE = CM_BASE + 20;
```

C++

```
CM_MOUSELEAVE = CM_BASE + 20;
```

File

Controls

Description

This is constant Controls.CM_MOUSELEAVE.

32.7.56 Controls.CM_MOUSEWHEEL Constant

Pascal

```
CM_MOUSEWHEEL = CM_BASE + 67;
```

C++

```
CM_MOUSEWHEEL = CM_BASE + 67;
```

File

Controls

Description

This is constant Controls.CM_MOUSEWHEEL.

32.7.57 Controls.CM_PARENTBIDIMODECHANGED Constant

Pascal

```
CM_PARENTBIDIMODECHANGED = CM_BASE + 61;
```

C++

```
CM_PARENTBIDIMODECHANGED = CM_BASE + 61;
```

File

Controls

Description

This is constant Controls.CM_PARENTBIDIMODECHANGED.

32.7.58 Controls.CM_PARENTCOLORCHANGED Constant

Pascal

```
CM_PARENTCOLORCHANGED = CM_BASE + 9;
```

C++

```
CM_PARENTCOLORCHANGED = CM_BASE + 9;
```

File

Controls

Description

This is constant Controls.CM_PARENTCOLORCHANGED.

32.7.59 Controls.CM_PARENTCTL3DCHANGED Constant

Pascal

```
CM_PARENTCTL3DCHANGED = CM_BASE + 17;
```

C++

```
CM_PARENTCTL3DCHANGED = CM_BASE + 17;
```

File

Controls

Description

This is constant Controls.CM_PARENTCTL3DCHANGED.

32.7.60 Controls.CM_PARENTFONTCHANGED Constant

Pascal

```
CM_PARENTFONTCHANGED = CM_BASE + 8;
```

C++

```
CM_PARENTFONTCHANGED = CM_BASE + 8;
```

File

Controls

Description

This is constant Controls.CM_PARENTFONTCHANGED.

32.7.61 Controls.CM_PARENTSHOWHINTCHANGED Constant

Pascal

```
CM_PARENTSHOWHINTCHANGED = CM_BASE + 35;
```

C++

```
CM_PARENTSHOWHINTCHANGED = CM_BASE + 35;
```

File

Controls

Description

This is constant Controls.CM_PARENTSHOWHINTCHANGED.

32.7.62 Controls.CM_POPUPHWNDDESTROY Constant

Pascal

```
CM_POPUPHWNDDESTROY = CM_BASE + 72;
```

C++

```
CM_POPUPHWNDDESTROY = CM_BASE + 72;
```

File

Controls

Description

This is constant Controls.CM_POPUPHWNDDESTROY.

32.7.63 Controls.CM_RECREATEWND Constant

Pascal

```
CM_RECREATEWND = CM_BASE + 51;
```

C++

```
CM_RECREATEWND = CM_BASE + 51;
```

File

Controls

Description

This is constant Controls.CM_RECREATEWND.

32.7.64 Controls.CM_RELEASE Constant

Pascal

```
CM_RELEASE = CM_BASE + 33;
```

C++

```
CM_RELEASE = CM_BASE + 33;
```

File

Controls

Description

This is constant Controls.CM_RELEASE.

32.7.65 Controls.CM_SETACTIVECONTROL Constant

Pascal

```
CM_SETACTIVECONTROL = CM_BASE + 71;
```

C++

```
CM_SETACTIVECONTROL = CM_BASE + 71;
```

File

Controls

Description

This is constant Controls.CM_SETACTIVECONTROL.

32.7.66 Controls.CM_SHOWHINTCHANGED Constant

Pascal

```
CM_SHOWHINTCHANGED = CM_BASE + 34;
```

C++

```
CM_SHOWHINTCHANGED = CM_BASE + 34;
```

File

Controls

Description

This is constant Controls.CM_SHOWHINTCHANGED.

32.7.67 Controls.CM_SHOWINGCHANGED Constant

Pascal

```
CM_SHOWINGCHANGED = CM_BASE + 25;
```

C++

```
CM_SHOWINGCHANGED = CM_BASE + 25;
```

File

Controls

Description

This is constant Controls.CM_SHOWINGCHANGED.

32.7.68 Controls.CM_SYSCOLORCHANGE Constant

Pascal

```
CM_SYSCOLORCHANGE = CM_BASE + 36;
```

C++

```
CM_SYSCOLORCHANGE = CM_BASE + 36;
```

File

Controls

Description

This is constant Controls.CM_SYSCOLORCHANGE.

32.7.69 Controls.CM_SYSFONTCHANGED Constant

Pascal

```
CM_SYSFONTCHANGED = CM_BASE + 53;
```

C++

```
CM_SYSFONTCHANGED = CM_BASE + 53;
```

File

Controls

Description

This is constant Controls.CM_SYSFONTCHANGED.

32.7.70 Controls.CM_TABSTOPCHANGED Constant

Pascal

```
CM_TABSTOPCHANGED = CM_BASE + 40;
```

C++

```
CM_TABSTOPCHANGED = CM_BASE + 40;
```

File

Controls

Description

This is constant Controls.CM_TABSTOPCHANGED.

32.7.71 Controls.CM_TEXTCHANGED Constant

Pascal

```
CM_TEXTCHANGED = CM_BASE + 18;
```

C++

```
CM_TEXTCHANGED = CM_BASE + 18;
```

File

Controls

Description

This is constant Controls.CM_TEXTCHANGED.

32.7.72 Controls.CM_TIMECHANGE Constant

Pascal

```
CM_TIMECHANGE = CM_BASE + 39;
```

C++

```
CM_TIMECHANGE = CM_BASE + 39;
```

File

Controls

Description

This is constant Controls.CM_TIMECHANGE.

32.7.73 Controls.CM_UIACTIVATE Constant

Pascal

```
CM_UIACTIVATE = CM_BASE + 41;
```

C++

```
CM_UIACTIVATE = CM_BASE + 41;
```

File

Controls

Description

This is constant Controls.CM_UIACTIVATE.

32.7.74 Controls.CM_UIDEACTIVATE Constant

Pascal

```
CM_UIDEACTIVATE = CM_BASE + 42;
```

C++

```
CM_UIDEACTIVATE = CM_BASE + 42;
```

File

Controls

Description

This is constant Controls.CM_UIDEACTIVATE.

32.7.75 Controls.CM_UNDOCKCLIENT Constant

Pascal

```
CM_UNDOCKCLIENT = CM_BASE + 57;
```

C++

```
CM_UNDOCKCLIENT = CM_BASE + 57;
```

File

Controls

Description

This is constant Controls.CM_UNDOCKCLIENT.

32.7.76 Controls.CM_VISIBLECHANGED Constant

Pascal

```
CM_VISIBLECHANGED = CM_BASE + 11;
```

C++

```
CM_VISIBLECHANGED = CM_BASE + 11;
```

File

Controls

Description

This is constant Controls.CM_VISIBLECHANGED.

32.7.77 Controls.CM_WANTSPECIALKEY Constant

Pascal

```
CM_WANTSPECIALKEY = CM_BASE + 30;
```

C++

```
CM_WANTSPECIALKEY = CM_BASE + 30;
```

File

Controls

Description

This is constant Controls.CM_WANTSPECIALKEY.

32.7.78 Controls.CM_WINDOWHOOK Constant

Pascal

```
CM_WINDOWHOOK = CM_BASE + 32;
```

C++

```
CM_WINDOWHOOK = CM_BASE + 32;
```

File

Controls

Description

This is constant Controls.CM_WINDOWHOOK.

32.7.79 Controls.CM_WININICHANGE Constant

Pascal

```
CM_WININICHANGE = CM_BASE + 37;
```

C++

```
CM_WININICHANGE = CM_BASE + 37;
```

File

Controls

Description

This is constant Controls.CM_WININICHANGE.

32.7.80 Controls.CN_BASE Constant

Pascal

```
CN_BASE = $BC00;
```

C++

```
CN_BASE = 0xBC00;
```

File

Controls

Description

This is constant Controls.CN_BASE.

32.7.81 Controls.CN_CHAR Constant

Pascal

```
CN_CHAR = CN_BASE + WM_CHAR;
```

C++

```
CN_CHAR = CN_BASE + WM_CHAR;
```

File

Controls

Description

This is constant Controls.CN_CHAR.

32.7.82 Controls.CN_CHARTOITEM Constant

Pascal

```
CN_CHARTOITEM = CN_BASE + WM_CHARTOITEM;
```

C++

```
CN_CHARTOITEM = CN_BASE + WM_CHARTOITEM;
```

File

Controls

Description

This is constant Controls.CN_CHARTOITEM.

32.7.83 Controls.CN_COMMAND Constant

Pascal

```
CN_COMMAND = CN_BASE + WM_COMMAND;
```

C++

```
CN_COMMAND = CN_BASE + WM_COMMAND;
```

File

Controls

Description

This is constant Controls.CN_COMMAND.

32.7.84 Controls.CN_COMPAREITEM Constant

Pascal

```
CN_COMPAREITEM = CN_BASE + WM_COMPAREITEM;
```

C++

```
CN_COMPAREITEM = CN_BASE + WM_COMPAREITEM;
```

File

Controls

Description

This is constant Controls.CN_COMPAREITEM.

32.7.85 Controls.CN_CTLCOLORBTN Constant

Pascal

```
CN_CTLCOLORBTN = CN_BASE + WM_CTLCOLORBTN;
```

C++

```
CN_CTLCOLORBTN = CN_BASE + WM_CTLCOLORBTN;
```

File

Controls

Description

This is constant Controls.CN_CTLCOLORBTN.

32.7.86 Controls.CN_CTLCOLORDLG Constant

Pascal

```
CN_CTLCOLORDLG = CN_BASE + WM_CTLCOLORDLG;
```

C++

```
CN_CTLCOLORDLG = CN_BASE + WM_CTLCOLORDLG;
```

File

Controls

Description

This is constant Controls.CN_CTLCOLORDLG.

32.7.87 Controls.CN_CTLCOLOREDIT Constant

Pascal

```
CN_CTLCOLOREDIT = CN_BASE + WM_CTLCOLOREDIT;
```

C++

```
CN_CTLCOLOREDIT = CN_BASE + WM_CTLCOLOREDIT;
```

File

Controls

Description

This is constant Controls.CN_CTLCOLOREDIT.

32.7.88 Controls.CN_CTLCOLORLISTBOX Constant

Pascal

```
CN_CTLCOLORLISTBOX = CN_BASE + WM_CTLCOLORLISTBOX;
```

C++

```
CN_CTLCOLORLISTBOX = CN_BASE + WM_CTLCOLORLISTBOX;
```

File

Controls

Description

This is constant Controls.CN_CTLCOLORLISTBOX.

32.7.89 Controls.CN_CTLCOLORMSGBOX Constant

Pascal

```
CN_CTLCOLORMSGBOX = CN_BASE + WM_CTLCOLORMSGBOX;
```

C++

```
CN_CTLCOLORMSGBOX = CN_BASE + WM_CTLCOLORMSGBOX;
```

File

Controls

Description

This is constant Controls.CN_CTLCOLORMSGBOX.

32.7.90 Controls.CN_CTLCOLORSCROLLBAR Constant

Pascal

```
CN_CTLCOLORSCROLLBAR = CN_BASE + WM_CTLCOLORSCROLLBAR;
```

C++

```
CN_CTLCOLORSCROLLBAR = CN_BASE + WM_CTLCOLORSCROLLBAR;
```

File

Controls

Description

This is constant Controls.CN_CTLCOLORSCROLLBAR.

32.7.91 Controls.CN_CTLCOLORSTATIC Constant

Pascal

```
CN_CTLCOLORSTATIC = CN_BASE + WM_CTLCOLORSTATIC;
```

C++

```
CN_CTLCOLORSTATIC = CN_BASE + WM_CTLCOLORSTATIC;
```

File

Controls

Description

This is constant Controls.CN_CTLCOLORSTATIC.

32.7.92 Controls.CN_DELETEITEM Constant

Pascal

```
CN_DELETEITEM = CN_BASE + WM_DELETEITEM;
```

C++

```
CN_DELETEITEM = CN_BASE + WM_DELETEITEM;
```

File

Controls

Description

This is constant Controls.CN_DELETEITEM.

32.7.93 Controls.CN_DRAWITEM Constant

Pascal

```
CN_DRAWITEM = CN_BASE + WM_DRAWITEM;
```

C++

```
CN_DRAWITEM = CN_BASE + WM_DRAWITEM;
```

File

Controls

Description

This is constant Controls.CN_DRAWITEM.

32.7.94 Controls.CN_HSCROLL Constant

Pascal

```
CN_HSCROLL = CN_BASE + WM_HSCROLL;
```

C++

```
CN_HSCROLL = CN_BASE + WM_HSCROLL;
```

File

Controls

Description

This is constant Controls.CN_HSCROLL.

32.7.95 Controls.CN_KEYDOWN Constant

Pascal

```
CN_KEYDOWN = CN_BASE + WM_KEYDOWN;
```

C++

```
CN_KEYDOWN = CN_BASE + WM_KEYDOWN;
```

File

Controls

Description

This is constant Controls.CN_KEYDOWN.

32.7.96 Controls.CN_KEYUP Constant

Pascal

```
CN_KEYUP = CN_BASE + WM_KEYUP;
```

C++

```
CN_KEYUP = CN_BASE + WM_KEYUP;
```

File

Controls

Description

This is constant Controls.CN_KEYUP.

32.7.97 Controls.CN_MEASUREITEM Constant

Pascal

```
CN_MEASUREITEM = CN_BASE + WM_MEASUREITEM;
```

C++

```
CN_MEASUREITEM = CN_BASE + WM_MEASUREITEM;
```

File

Controls

Description

This is constant Controls.CN_MEASUREITEM.

32.7.98 Controls.CN_NOTIFY Constant

Pascal

```
CN_NOTIFY = CN_BASE + WM_NOTIFY;
```

C++

```
CN_NOTIFY = CN_BASE + WM_NOTIFY;
```

File

Controls

Description

This is constant Controls.CN_NOTIFY.

32.7.99 Controls.CN_PARENTNOTIFY Constant

Pascal

```
CN_PARENTNOTIFY = CN_BASE + WM_PARENTNOTIFY;
```

C++

```
CN_PARENTNOTIFY = CN_BASE + WM_PARENTNOTIFY;
```

File

Controls

Description

This is constant Controls.CN_PARENTNOTIFY.

32.7.100 Controls.CN_SYSCHAR Constant

Pascal

```
CN_SYSCHAR = CN_BASE + WM_SYSCHAR;
```

C++

```
CN_SYSCHAR = CN_BASE + WM_SYSCHAR;
```

File

Controls

Description

This is constant Controls.CN_SYSCHAR.

32.7.101 Controls.CN_SYSKEYDOWN Constant

Pascal

```
CN_SYSKEYDOWN = CN_BASE + WM_SYSKEYDOWN;
```

C++

```
CN_SYSKEYDOWN = CN_BASE + WM_SYSKEYDOWN;
```

File

Controls

Description

This is constant Controls.CN_SYSKEYDOWN.

32.7.102 Controls.CN_VKEYTOITEM Constant

Pascal

```
CN_VKEYTOITEM = CN_BASE + WM_VKEYTOITEM;
```

C++

```
CN_VKEYTOITEM = CN_BASE + WM_VKEYTOITEM;
```

File

Controls

Description

This is constant Controls.CN_VKEYTOITEM.

32.7.103 Controls.CN_VSCROLL Constant

Pascal

```
CN_VSCROLL = CN_BASE + WM_VSCROLL;
```

C++

```
CN_VSCROLL = CN_BASE + WM_VSCROLL;
```

File

Controls

Description

This is constant Controls.CN_VSCROLL.

32.7.104 Controls.crAppStart Constant

Identifies the default cursor.

Pascal

```
crAppStart = TCursor(-19);
```

C++

```
crAppStart = TCursor(-19);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -19 identifies the Application Start cursor, a combination of arrow and hourglass.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.105 Controls.crArrow Constant

Identifies an arrow cursor.

Pascal

```
crArrow = TCursor(-2);
```

C++

```
crArrow = TCursor(-2);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -2 identifies a cursor that is an arrow pointing up and to the left.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.106 Controls.crCross Constant

Identifies an cross cursor.

Pascal

```
crCross = TCursor(-3);
```

C++

```
crCross = TCursor(-3);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -3 identifies a cross cursor. This is principally used for precise screen work.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.107 Controls.crDefault Constant

Identifies the default cursor.

Pascal

```
crDefault = TCursor(0);
```

C++

```
crDefault = TCursor(0);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The default value of 0 represents the default cursor, an arrow pointing up and to the left.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.108 Controls.crDrag Constant

Identifies a drag cursor.

Pascal

```
crDrag = TCursor(-12);
```

C++

```
crDrag = TCursor(-12);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -12 identifies a drag cursor. It is deployed when a user clicks, holds and then drags a screen item that supports dragging. It is shown as an arrow and document combination.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.109 Controls.crHandPoint Constant

Identifies a pointing hand cursor.

Pascal

```
crHandPoint = TCursor(-21);
```

C++

```
crHandPoint = TCursor(-21);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -21 identifies an upward pointing hand cursor. This is normally used to identify a selectable item, such as a web page link.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.110 Controls.crHelp Constant

Identifies a help cursor.

Pascal

```
crHelp = TCursor(-20);
```

C++

```
crHelp = TCursor(-20);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -20 identifies a help cursor, comprising the default arrow pointing up and to the left combined with a question mark.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.111 Controls.crHourGlass Constant

Identifies an hour glass cursor.

Pascal

```
crHourGlass = TCursor(-11);
```

C++

```
crHourGlass = TCursor(-11);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -11 identifies an hour glass cursor. This is typically used to inform the user to wait until an application completes its current operation.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.112 Controls.crHSplit Constant

Identifies a horizontal split cursor.

Pascal

```
crHSplit = TCursor(-14);
```

C++

```
crHSplit = TCursor(-14);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -14 identifies a horizontal split cursor. This is used to indicate when a TSplitter control under the mouse can be dragged horizontally. It is represented as 2 horizontal lines bracketed by 2 arrows. Like a lift door open symbol.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

TSplitter (see page 13210)

32.7.113 Controls.crIBeam Constant

Identifies a text insert cursor.

Pascal

```
crIBeam = TCursor(-4);
```

C++

```
crIBeam = TCursor(-4);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -4 identifies a text insert cursor, in the shape of a thin capital I. It is used when choosing where in a passage of text where typed characters next.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.114 Controls.crMultiDrag Constant

Identifies a multiple item drag cursor.

Pascal

```
crMultiDrag = TCursor(-16);
```

C++

```
crMultiDrag = TCursor(-16);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -16 identifies a multi item drag cursor. It is used to indicate that a group of items have been selected and are being dragged.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.115 Controls.crNo Constant

Identifies a not allowed cursor.

Pascal

```
crNo = TCursor(-18);
```

C++

```
crNo = TCursor(-18);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -18 identifies an action denied cursor. It is used to indicate to the user that the cursor is hovering over an area which is not a valid target. It is represented by a black cross within a black circle.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.116 Controls.crNoDrop Constant

Identifies a not allowed to drop cursor.

Pascal

```
crNoDrop = TCursor(-13);
```

C++

```
crNoDrop = TCursor(-13);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -13 identifies a drop disallowed cursor. It is used to indicate to the user that the cursor is hovering over an area which is not a valid target for a drag and drop operation. It is represented by a white cross within a white circle.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.117 Controls.crNone Constant

Identifies an invisible cursor.

Pascal

```
crNone = TCursor(-1);
```

C++

```
crNone = TCursor(-1);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -1 identifies an invisible cursor. The cursor literally disappears when hovering over the application - except for the title bar.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.118 Controls.crSize Constant

Identifies an obsolete cursor.

Pascal

```
crSize = TCursor(-22);
```

C++

```
crSize = TCursor(-22);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -22 identifies a sizing cursor that is no longer available. It is now replaced by 5 different orientation sizing cursors.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.119 Controls.crSizeAll Constant

Identifies a size in all directions cursor.

Pascal

```
crSizeAll = TCursor(-22);
```

C++

```
crSizeAll = TCursor(-22);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -22 identifies a resizing cursor that allows for resizing in all 4 directions, up, down, left and right. It is represented as a set of 4 arrows pointing outwards in those 4 directions.

Note: crSizeAll has the same value of -22 as the obsolete crSize (see page 6367).

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.120 Controls.crSizeNESW Constant

Identifies a North-East to South-West sizing cursor.

Pascal

```
crSizeNESW = TCursor(-6);
```

C++

```
crSizeNESW = TCursor(-6);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -6 identifies a resizing cursor that allows for resizing in either up and to the right (to the North East) or down and to the left (to the South West). It is identified by a double headed arrow sloping 45 degrees to the right.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.121 Controls.crSizeNS Constant

Identifies a vertical sizing cursor.

Pascal

```
crSizeNS = TCursor(-7);
```

C++

```
crSizeNS = TCursor(-7);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -7 identifies a resizing cursor that allows for vertical resizing. It is identified by a double headed arrow aligned vertically.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.122 Controls.crSizeNWSE Constant

Identifies a North-West to South-East sizing cursor.

Pascal

```
crSizeNWSE = TCursor(-8);
```

C++

```
crSizeNWSE = TCursor(-8);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -8 identifies a resizing cursor that allows for resizing in either up and to the left (to the North West) or down and to the right (to the South East). It is identified by a double headed arrow sloping 45 degrees to the left.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.123 Controls.crSizeWE Constant

Identifies a horizontal sizing cursor.

Pascal

```
crSizeWE = TCursor(-9);
```

C++

```
crSizeWE = TCursor(-9);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -9 identifies a resizing cursor that allows for horizontal resizing. It is identified by a double headed arrow aligned horizontally.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.124 Controls.crSQLWait Constant

Identifies an SQL wait hour glass cursor.

Pascal

```
crSQLWait = TCursor(-17);
```

C++

```
crSQLWait = TCursor(-17);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -17 identifies an hour glass cursor used for SQL operations to indicate that the current SQL operation is still in progress.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.125 Controls.crUpArrow Constant

Identifies an upwards pointing arrow cursor.

Pascal

```
crUpArrow = TCursor(-10);
```

C++

```
crUpArrow = TCursor(-10);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -10 identifies an upwards pointing arrow cursor. This is a thin arrow of the type used in sizing operations, rather than the thick default cursor arrow.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

32.7.126 Controls.crVSplit Constant

Identifies a vertical split cursor.

Pascal

```
crVSplit = TCursor(-15);
```

C++

```
crVSplit = TCursor(-15);
```

File

Controls

Description

The TCursor (see page 6305) type defines a cursor ident (integer). The value -15 identifies a vertical split cursor. This is used to indicate when a TSplitter control under the mouse can be dragged vertically. It is represented as 2 horizontal lines bracketed by 2 arrows.

Use the Screen Cursor property to set the cursor type to one of the TCursor (see page 6305) values.

See Also

TCursor (see page 6305)

Cursor (see page 14209)

TSplitter (see page 13210)

32.7.127 Controls.CTL3D_ALL Constant

Pascal

```
CTL3D_ALL = $FFFF;
```

C++

```
CTL3D_ALL = 0xFFFF;
```

File

Controls

Description

This is constant Controls.CTL3D_ALL.

32.7.128 Controls.mrAbort Constant

Indicates that Abort was selected in a modal dialog.

Pascal

```
mrAbort = idAbort;
```

C++

```
mrAbort = idAbort;
```

File

Controls

Description

The MessageDlg function provides a convenient mechanism for asking the user for a decision. mrAbort is the returned TModalResult (see page 6312) value when the MsgDlgBtn mbAbort button was pressed.

See Also

MessageDlg (see page 11535)

MsgDlgBtn

TModalResult (see page 6312)

32.7.129 Controls.mrAll Constant

Pascal

```
mrAll = mrNo + 1;
```

C++

```
mrAll = mrNo + 1;
```

File

Controls

Description

This is constant Controls.mrAll.

32.7.130 Controls.mrCancel Constant

Indicates that Cancel was selected in a modal dialog.

Pascal

```
mrCancel = idCancel;
```

C++

```
mrCancel = idCancel;
```

File

Controls

Description

The MessageDlg function provides a convenient mechanism for asking the user for a decision. mrCancel is the returned TModalResult (see page 6312) value when the MsgDlgBtn mbCancel button was pressed.

See Also

MessageDlg (see page 11535)

MsgDlgBtn

TModalResult (see page 6312)

32.7.131 Controls.mrIgnore Constant

Indicates that Ignore was selected in a modal dialog.

Pascal

```
mrIgnore = idIgnore;
```

C++

```
mrIgnore = idIgnore;
```

File

Controls

Description

The MessageDlg function provides a convenient mechanism for asking the user for a decision. mrIgnore is returned TModalResult (see page 6312) value when the MsgDlgBtn mbIgnore button was pressed.

See Also

MessageDlg (see page 11535)

MsgDlgBtn

TModalResult (see page 6312)

32.7.132 Controls.mrNo Constant

Indicates that No was selected in a modal dialog.

Pascal

```
mrNo = idNo;
```

C++

```
mrNo = idNo;
```

File

Controls

Description

The MessageDlg function provides a convenient mechanism for asking the user for a decision. mrNo is returned TModalResult (see page 6312) value when the MsgDlgBtn mbNo button was pressed.

See Also

MessageDlg (see page 11535)

MsgDlgBtn

TModalResult (see page 6312)

32.7.133 Controls.mrNone Constant

The default modal dialog return value.

Pascal

```
mrNone = 0;
```

C++

```
mrNone = 0;
```

File

Controls

Description

The MessageDlg function provides a convenient mechanism for asking the user for a decision. mrNone is the default value before a user exits the dialog.

See Also

MessageDlg (see page 11535)

MsgDlgBtn

TModalResult (see page 6312)

32.7.134 Controls.mrNoToAll Constant

Indicates that 'No To All' was selected in a modal dialog.

Pascal

```
mrNoToAll = mrAll + 1;
```

C++

```
mrNoToAll = mrAll + 1;
```

File

Controls

Description

The MessageDlg function provides a convenient mechanism for asking the user for a decision. mrNoToAll is returned TModalResult (see page 6312) value when the MsgDlgBtn mbNoToAll button was pressed.

See Also

MessageDlg (see page 11535)

MsgDlgBtn

TModalResult (see page 6312)

32.7.135 Controls.mrOk Constant

Pascal

```
mrOk = idOk;
```

C++

```
mrOk = idOk;
```

File

Controls

Description

This is constant Controls.mrOk.

32.7.136 Controls.mrRetry Constant

Indicates that Retry was selected in a modal dialog.

Pascal

```
mrRetry = idRetry;
```

C++

```
mrRetry = idRetry;
```

File

Controls

Description

The MessageDlg function provides a convenient mechanism for asking the user for a decision. mrRetry is returned TModalResult (see page 6312) value when the MsgDlgBtn mbRetry button was pressed.

See Also

MessageDlg (see page 11535)

MsgDlgBtn

TModalResult (see page 6312)

32.7.137 Controls.mrYes Constant

Indicates that Yes was selected in a modal dialog.

Pascal

```
mrYes = idYes;
```

C++

```
mrYes = idYes;
```

File

Controls

Description

The MessageDlg function provides a convenient mechanism for asking the user for a decision. mrYes is returned TModalResult (see page 6312) value when the MsgDlgBtn mbYes button was pressed.

See Also

MessageDlg (see page 11535)

MsgDlgBtn

TModalResult (see page 6312)

32.7.138 Controls.mrYesToAll Constant

Indicates that 'Yes To All' was selected in a modal dialog.

Pascal

```
mrYesToAll = mrNoToAll + 1;
```

C++

```
mrYesToAll = mrNoToAll + 1;
```

File

Controls

Description

The MessageDlg function provides a convenient mechanism for asking the user for a decision. mrYesToAll is returned TModalResult (see page 6312) value when the MsgDlgBtn mbYesToAll button was pressed.

See Also

MessageDlg (see page 11535)

MsgDlgBtn

TModalResult (see page 6312)

32.7.139 Controls.NullDockSite Constant

Pascal

```
NullDockSite = TWinControl($FFFFFFFF);
```

C++

```
NullDockSite = TWinControl($FFFFFFFF);
```

File

Controls

Description

This is constant Controls.NullDockSite.

33 ConvUtils Namespace

This is namespace ConvUtils.

Topics

Name	Description
Classes (see page 6381)	The following table lists classes in this documentation.
Functions (see page 6395)	The following table lists functions in this documentation.
Types (see page 6416)	The following table lists types in this documentation.
Variables (see page 6419)	The following table lists variables in this documentation.
Constants (see page 6420)	The following table lists constants in this documentation.

Classes

	Name	Description
	EConversionError (see page 6381)	EConversionError is the exception class for measurement conversion errors.
	TConvTypeFactor (see page 6382)	TConvTypeFactor describes a single registered conversion type (measurement unit) that differs from its family's base units by a simple conversion factor.
	TConvTypeInfo (see page 6386)	TConvTypeInfo describes a single registered conversion type (measurement unit).
	TConvTypeProcs (see page 6391)	TConvTypeProcs describes a single registered conversion type (measurement unit) that is registered by supplying conversion methods.

Constants

Name	Description
CIllegalConvFamily (see page 6420)	This is constant ConvUtils.CIllegalConvFamily.
CIllegalConvType (see page 6420)	This is constant ConvUtils.CIllegalConvType.

Functions

	Name	Description
	CompatibleConversionType (see page 6396)	Indicates whether a specified conversion type is registered with a specified conversion family.
	CompatibleConversionTypes (see page 6396)	Indicates whether the Convert (see page 6397) function can convert between two specified conversion types.
	Convert (see page 6397)	Converts a measurement from one set of units to another.
	Convert (see page 6397)	Converts a measurement from one set of units to another.
	ConvertFrom (see page 6398)	Converts a measurement from the specified units to the base units of its conversion family.
	ConvertTo (see page 6399)	Converts a measurement from the base units of a conversion family into a specified conversion type.

	ConvFamilyToDescription (see page 6399)	Returns the string description of what a conversion family measures.
	ConvTypeToDescription (see page 6400)	Returns the string description of a conversion type (measurement unit).
	ConvTypeToFamily (see page 6400)	Returns the identifier for the conversion family with which a conversion type or pair of types is registered.
	ConvTypeToFamily (see page 6400)	Returns the identifier for the conversion family with which a conversion type or pair of types is registered.
	ConvUnitAdd (see page 6401)	Adds two measurements and returns the result using a specified unit of measurement.
	ConvUnitCompareValue (see page 6402)	Indicates the relationship between two measurements.
	ConvUnitDec (see page 6402)	Decrements a specified measurement by a specified amount.
	ConvUnitDec (see page 6402)	Decrements a specified measurement by a specified amount.
	ConvUnitDiff (see page 6403)	Subtracts one measurement from another and returns the result using a specified unit of measurement.
	ConvUnitInc (see page 6404)	Increments a specified measurement by a specified amount.
	ConvUnitInc (see page 6404)	Increments a specified measurement by a specified amount.
	ConvUnitSameValue (see page 6404)	Indicates whether two measurements are equivalent.
	ConvUnitToStr (see page 6405)	Formats a measurement and its conversion unit into a human-readable string.
	ConvUnitWithinNext (see page 6406)	Indicates whether a specified measurement exceeds another measurement by at most a specified amount.
	ConvUnitWithinPrevious (see page 6406)	Indicates whether a specified measurement is at most a specified amount less than another measurement.
	DescriptionToConvFamily (see page 6407)	Retrieves the identifier for a conversion family given its name.
	DescriptionToConvType (see page 6408)	Retrieves the identifier for a conversion type given its name and family.
	DescriptionToConvType (see page 6408)	Retrieves the identifier for a conversion type given its name and family.
	GetConvFamilies (see page 6408)	Returns a list of all registered conversion families.
	GetConvTypes (see page 6409)	Returns a list of all registered conversion types in a specified conversion family.
	RaiseConversionError (see page 6409)	Raises an EConversionError (see page 6381) exception.
	RaiseConversionError (see page 6409)	Raises an EConversionError (see page 6381) exception.
	RaiseConversionRegError (see page 6410)	This is function ConvUtils.RaiseConversionRegError.
	RegisterConversionFamily (see page 6410)	Registers a new conversion family and returns its identifier.
	RegisterConversionType (see page 6411)	Registers a new conversion type and returns its identifier.
	RegisterConversionType (see page 6411)	Registers a new conversion type and returns its identifier.
	RegisterConversionType (see page 6411)	Registers a new conversion type and returns its identifier.
	StrToConvUnit (see page 6412)	Parses a string into a value and conversion type.
	TryConvTypeToFamily (see page 6413)	Retrieves the identifier for the conversion family with which a conversion type or pair of types is registered.
	TryConvTypeToFamily (see page 6413)	Retrieves the identifier for the conversion family with which a conversion type or pair of types is registered.
	TryStrToConvUnit (see page 6414)	Parses a string into a value and conversion type.
	UnregisterConversionFamily (see page 6414)	Unregisters a conversion family previously registered using RegisterConversionFamily (see page 6410).
	UnregisterConversionType (see page 6415)	Unregisters a conversion type previously registered using RegisterConversionType (see page 6411).

Types

Name	Description
TConversionProc (see page 6416)	This is type ConvUtils.TConversionProc.
TConvFamily (see page 6416)	This is type ConvUtils.TConvFamily.
TConvFamilyArray (see page 6417)	TConvFamilyArray represents a dynamic array of conversion families.
TConvType (see page 6417)	This is type ConvUtils.TConvType.
TConvTypeArray (see page 6417)	TConvTypeArray represents a dynamic array of conversion types (measurement units)
TConvTypeList (see page 6418)	This is type ConvUtils.TConvTypeList.

Variables

Name	Description
GConvUnitToStrFmt (see page 6419)	This is variable ConvUtils.GConvUnitToStrFmt.

33.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EConversionError (see page 6381)	EConversionError is the exception class for measurement conversion errors.
	TConvTypeFactor (see page 6382)	TConvTypeFactor describes a single registered conversion type (measurement unit) that differs from its family's base units by a simple conversion factor.
	TConvTypeInfo (see page 6386)	TConvTypeInfo describes a single registered conversion type (measurement unit).
	TConvTypeProcs (see page 6391)	TConvTypeProcs describes a single registered conversion type (measurement unit) that is registered by supplying conversion methods.

33.1.1 EConversionError Class

EConversionError is the exception class for measurement conversion errors.

Class Hierarchy

Pascal

```
EConversionError = class(Exception);
```

C++

```
class EConversionError : public Exception;
```

File

ConvUtils

Description

EConversionError is raised by the global Convert (see page 6397) function when

The specified units (TConvType (see page 6417) values) are not compatible with each other.

The specified units are not registered with any conversion family.

It is also thrown by the RegisterConversionType (see page 6411) function when an application tries to register a conversion type with an unregistered conversion family.

See Also

RaiseConversionError

Convert

TConvType

RegisterConversionType

33.1.2 TConvTypeFactor Class

TConvTypeFactor describes a single registered conversion type (measurement unit) that differs from its family's base units by a simple conversion factor.

Class Hierarchy

Pascal

```
TConvTypeFactor = class(TConvTypeInfo);
```

C++

```
class TConvTypeFactor : public TConvTypeInfo;
```

File

ConvUtils

Description

The global RegisterConversionType (see page 6411) function creates a TConvTypeFactor instance automatically when you register a conversion type by supplying a conversion factor. TConvTypeFactor implements the FromCommon (see page 6384) and ToCommon (see page 6385) methods to perform their conversions by using a conversion factor that is the value of the Factor (see page 6385) property.

There is no need to use TConvTypeFactor directly. To register a conversion type that uses a simple conversion factor, call RegisterConversionType (see page 6411) using the syntax that supplies a conversion factor.

You can, however, use TConvTypeFactor as a base class when implementing your own conversion type that performs addition actions beyond a conversion factor. For example, you could write a class that multiplies by a conversion factor and then shifts the zero point by adding a constant.

See Also

TConvType (see page 6417)

TConvFamily (see page 6416)

RegisterConversionType (see page 6411)

Convert (see page 6397)

TConvTypeProcs (see page 6391)

33.1.2.1 TConvTypeFactor Members

The following tables list the members exposed by TConvTypeFactor.

Protected Properties

	Name	Description
R	Factor (see page 6385)	Specifies the conversion factor that converts from the conversion type into the base units of the conversion family.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6383)	Creates an instance of TConvTypeFactor (see page 6382).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
V	FromCommon (see page 6384)	Converts measurements in the base unit for the conversion family into the units represented by this conversion type.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	ToCommon (see page 6385)	Converts measurements in the units represented by this conversion type to the base units of the conversion family.

Public Properties

	Name	Description
R	ConvFamily (see page 6390)	Identifies the conversion family to which the conversion type belongs.
R	ConvType (see page 6390)	Indicates the registered identifier for the conversion type.
R	Description (see page 6391)	Names the measurement unit represented by the conversion type.

33.1.2.2 TConvTypeFactor.Create Constructor

Creates an instance of TConvTypeFactor (see page 6382).

Pascal

```
constructor Create(const AConvFamily: TConvFamily; const ADescription: string; const AFactor:
Double);
```

C++

```
__fastcall TConvTypeFactor(const TConvFamily AConvFamily, const AnsiString ADescription, const
Double AFactor);
```

Description

Call Create to instantiate a TConvTypeFactor (see page 6382) descendant at runtime. This descendant can then be passed to the RegisterConversionType (see page 6411) function in order to register a new conversion type. There is no need to instantiate TConvTypeFactor (see page 6382) instances directly, because the RegisterConversionType (see page 6411) function creates a new TConvTypeFactor (see page 6382) instance when you register a conversion type by specifying a conversion factor.

Create assigns the AFactor parameter to the Factor (see page 6385) property, the AConvFamily parameter to the ConvFamily (see page 6390) property, and the ADescription parameter to the Description (see page 6391) property.

If AFactor is zero, Create raises an EZeroDivide exception. If AConvFamily is not the identifier of a registered conversion family, Create raises an EConversionError (see page 6381) exception.

See Also

RegisterConversionType (see page 6411)

ConvFamily (see page 6390)

Description (see page 6391)

EConversionError (see page 6381)

EZeroDivide (see page 22761)

33.1.2.3 TConvTypeFactor Methods

The methods of the TConvTypeFactor class are listed here.

Public Methods

	Name	Description
	FromCommon (see page 6384)	Converts measurements in the base unit for the conversion family into the units represented by this conversion type.
	ToCommon (see page 6385)	Converts measurements in the units represented by this conversion type to the base units of the conversion family.

33.1.2.3.1 TConvTypeFactor.FromCommon Method

Converts measurements in the base unit for the conversion family into the units represented by this conversion type.

Pascal

```
function FromCommon(const AValue: Double): Double; override;
```

C++

```
virtual __fastcall Double FromCommon(const Double AValue);
```

Description

FromCommon provides part of the underlying implementation of the global Convert (see page 6397) function. In order to convert a measurement from one set of units to another, Convert (see page 6397) calls the ToCommon (see page 6385) method of the source conversion type's TConvTypeFactor (see page 6382) instance to obtain the measurement in the base units of the conversion family, and then calls the FromCommon method of the destination conversion type's TConvTypeFactor (see page 6382) instance to obtain the measurement in the destination units.

AValue is a value expressed in the base units of the conversion family.

FromCommon divides AValue by the value of the Factor (see page 6385) property and returns the result.

See Also

ToCommon (see page 6385)

ConvFamily (see page 6390)

Convert (see page 6397)

Factor (see page 6385)

33.1.2.3.2 TConvTypeFactor.ToCommon Method

Converts measurements in the units represented by this conversion type to the base units of the conversion family.

Pascal

```
function ToCommon(const AValue: Double): Double; override;
```

C++

```
virtual __fastcall Double ToCommon(const Double AValue);
```

Description

ToCommon provides part of the underlying implementation of the global Convert (see page 6397) function. In order to convert a measurement from one set of units to another, Convert (see page 6397) calls the ToCommon method of the source conversion type's TConvTypeFactor (see page 6382) instance to obtain the measurement in the base units of the conversion family, and then calls the ToCommon method of the destination conversion type's TConvTypeFactor (see page 6382) instance to obtain the measurement in the destination units.

AValue is a value expressed in the units represented by this conversion type.

ToCommon multiplies AValue by the value of the Factor (see page 6385) property and returns the result.

See Also

FromCommon (see page 6384)

ConvFamily (see page 6390)

Convert (see page 6397)

Factor (see page 6385)

33.1.2.4 TConvTypeFactor Properties

The properties of the TConvTypeFactor class are listed here.

Protected Properties

	Name	Description
R	Factor (see page 6385)	Specifies the conversion factor that converts from the conversion type into the base units of the conversion family.

33.1.2.4.1 TConvTypeFactor.Factor Property

Specifies the conversion factor that converts from the conversion type into the base units of the conversion family.

Pascal

```
property Factor: Double;
```

C++

```
__property Double Factor;
```

Description

Factor is the conversion factor that defines the relationship between this conversion type and the base units of its conversion family. The ToCommon (see page 6385) method multiplies a measurement made using this conversion type to obtain the corresponding value that uses the base units for the conversion family. The FromCommon (see page 6384) method divides a measurement made using the base units for the conversion family to obtain the corresponding value expressed using this conversion unit.

See Also

FromCommon (see page 6384)

ToCommon (see page 6385)

33.1.3 TConvTypeInfo Class

TConvTypeInfo describes a single registered conversion type (measurement unit).

Class Hierarchy

Pascal

```
TConvTypeInfo = class(TObject);
```

C++

```
class TConvTypeInfo : public TObject;
```

File

ConvUtils

Description

TConvTypeInfo contains information about a conversion type, including

The ConvFamily (see page 6390) property, which indicates the conversion family to which the conversion type belongs.

The ConvType (see page 6390) property, which is the TConvType (see page 6417) identifier for the conversion type.

The Description (see page 6391) property, which is a human-readable description of the conversion type.

In addition, each TConvTypeInfo instance defines two methods for converting between the conversion type and the base units of its conversion family. These methods are abstract (pure virual) in TConvTypeInfo, and implemented by the descendant classes created by the RegisterConversionType (see page 6411) function.

When you call RegisterConversionType (see page 6411) to register a new conversion type (measurement unit), you can describe the new unit in one of three ways:

Specify the conversion family, a description, and supply a factor for converting from the new type to the base units of the conversion family.

Specify the conversion family, a description, and supply two methods for converting between the new type and the base units of the conversion family.

Provide a TConvTypeInfo descendant whose properties describe the conversion family and description and whose methods

convert between the new type and the base units of the conversion family.

If you choose either of the first two methods, RegisterConversionType (see page 6411) creates a special TConvTypeInfo descendant to represent the new conversion type.

By creating your own TConvTypeInfo descendant, you can make use of a common conversion scheme that relies only on a simple parameterized conversion method rather than creating many nearly identical conversion functions, one for each conversion type. Thus, for example, the TConvTypeFactor (see page 6382) class, which is created automatically when you register a conversion type by specifying a conversion factor, implements the FromCommon (see page 6389) and ToCommon (see page 6389) methods only once, using a property to specify the conversion factor.

See Also

TConvType (see page 6417)

TConvFamily (see page 6416)

RegisterConversionType (see page 6411)

Convert (see page 6397)

TConvTypeFactor (see page 6382)

TConvTypeProcs (see page 6391)

33.1.3.1 TConvTypeInfo Members

The following tables list the members exposed by TConvTypeInfo.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6388)	Creates an instance of TConvTypeInfo (see page 6386).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FromCommon (see page 6389)	Converts measurements in the base unit for the conversion family into the units represented by this conversion type.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
A	ToCommon (see page 6389)	Converts measurements in the units represented by this conversion type to the base units of the conversion family.

Public Properties

	Name	Description
R	ConvFamily (see page 6390)	Identifies the conversion family to which the conversion type belongs.
R	ConvType (see page 6390)	Indicates the registered identifier for the conversion type.
R	Description (see page 6391)	Names the measurement unit represented by the conversion type.

33.1.3.2 TConvTypeInfo.Create Constructor

Creates an instance of TConvTypeInfo (see page 6386).

Pascal

```
constructor Create(const AConvFamily: TConvFamily; const ADescription: string);
```

C++

```
__fastcall TConvTypeInfo(const TConvFamily AConvFamily, const AnsiString ADescription);
```

Description

Call Create to instantiate a TConvTypeInfo (see page 6386) descendant at runtime. This descendant can then be passed to the RegisterConversionType (see page 6411) function in order to register a new conversion type.

Create assigns the AConvFamily parameter to the ConvFamily (see page 6390) property and the ADescription parameter to the Description (see page 6391) property. If AConvFamily is not the identifier of a registered conversion family, Create raises an EConversionError (see page 6381) exception.

See Also

RegisterConversionType (see page 6411)

ConvFamily (see page 6390)

Description (see page 6391)

EConversionError (see page 6381)

33.1.3.3 TConvTypeInfo Methods

The methods of the TConvTypeInfo class are listed here.

Public Methods

	Name	Description
A	FromCommon (see page 6389)	Converts measurements in the base unit for the conversion family into the units represented by this conversion type.
A	ToCommon (see page 6389)	Converts measurements in the units represented by this conversion type to the base units of the conversion family.

33.1.3.3.1 TConvTypeInfo.FromCommon Method

Converts measurements in the base unit for the conversion family into the units represented by this conversion type.

Pascal

```
function FromCommon(const AValue: Double): Double; virtual; abstract;
```

C++

```
virtual __fastcall Double FromCommon(const Double AValue) = 0;
```

Description

FromCommon provides part of the underlying implementation of the global Convert (see page 6397) function. In order to convert a measurement from one set of units to another, Convert (see page 6397) calls the ToCommon (see page 6389) method of the source conversion type's TConvTypeInfo (see page 6386) instance to obtain the measurement in the base units of the conversion family, and then calls the FromCommon method of the destination conversion type's TConvTypeInfo (see page 6386) instance to obtain the measurement in the destination units.

TConvTypeInfo (see page 6386) introduces FromCommon as an abstract (pure virtual) method. Descendant classes implement this method to perform the conversion from the base units of a conversion family into the measurement units represented by this conversion type.

See Also

ToCommon (see page 6389)

ConvFamily (see page 6390)

Convert (see page 6397)

33.1.3.3.2 TConvTypeInfo.ToCommon Method

Converts measurements in the units represented by this conversion type to the base units of the conversion family.

Pascal

```
function ToCommon(const AValue: Double): Double; virtual; abstract;
```

C++

```
virtual __fastcall Double ToCommon(const Double AValue) = 0;
```

Description

ToCommon provides part of the underlying implementation of the global Convert (see page 6397) function. In order to convert a measurement from one set of units to another, Convert (see page 6397) calls the ToCommon method of the source conversion type's TConvTypeInfo (see page 6386) instance to obtain the measurement in the base units of the conversion family, and then calls the ToCommon method of the destination conversion type's TConvTypeInfo (see page 6386) instance to obtain the measurement in the destination units.

TConvTypeInfo (see page 6386) introduces ToCommon as an abstract (pure virtual) method. Descendant classes implement this method to perform the conversion from the units represented by this conversion type to the base units of its conversion family.

See Also

FromCommon (see page 6389)

ConvFamily (see page 6390)

Convert (see page 6397)

33.1.3.4 TConvTypeInfo Properties

The properties of the TConvTypeInfo class are listed here.

Public Properties

	Name	Description
R	ConvFamily (see page 6390)	Identifies the conversion family to which the conversion type belongs.
R	ConvType (see page 6390)	Indicates the registered identifier for the conversion type.
R	Description (see page 6391)	Names the measurement unit represented by the conversion type.

33.1.3.4.1 TConvTypeInfo.ConvFamily Property

Identifies the conversion family to which the conversion type belongs.

Pascal

```
property ConvFamily: TConvFamily;
```

C++

```
__property TConvFamily ConvFamily;
```

Description

ConvFamily is the identifier for the conversion family to which the conversion type belong. The RegisterConversionType (see page 6411) function associates each conversion type with its family.

Measurements in the units represented by the conversion type can be converted to any other conversion type in the same family.

See Also

ConvType (see page 6390)

RegisterConversionType (see page 6411)

RegisterConversionFamily (see page 6410)

33.1.3.4.2 TConvTypeInfo.ConvType Property

Indicates the registered identifier for the conversion type.

Pascal

```
property ConvType: TConvType;
```

C++

```
__property TConvType ConvType;
```

Description

ConvType is the identifier for the conversion type described by this TConvTypeInfo (see page 6386) instance. It is generated by the RegisterConversionType (see page 6411) function to uniquely identify the conversion type (measurement unit).

See Also

ConvFamily (see page 6390)

RegisterConversionType (see page 6411)

Description (see page 6391)

33.1.3.4.3 TConvTypeInfo.Description Property

Names the measurement unit represented by the conversion type.

Pascal

```
property Description: string;
```

C++

```
__property AnsiString Description;
```

Description

Description is the descriptive name supplied for the conversion type when it is registered. Typically, this is the name of a type of measurement unit. For example, the Description for the tuFortnights conversion type is 'Fortnights'.

See Also

ConvType (see page 6390)

RegisterConversionType (see page 6411)

33.1.4 TConvTypeProcs Class

TConvTypeProcs describes a single registered conversion type (measurement unit) that is registered by supplying conversion methods.

Class Hierarchy

Pascal

```
TConvTypeProcs = class(TConvTypeInfo);
```

C++

```
class TConvTypeProcs : public TConvTypeInfo;
```

File

ConvUtils

Description

The global RegisterConversionType (see page 6411) function creates a TConvTypeProcs instance automatically when you register a conversion type by supplying functions to convert between the that type and the base units in its conversion family. TConvTypeProcs implements the FromCommon (see page 6393) and ToCommon (see page 6394) methods to perform their conversions by using the functions supplied to its constructor.

There is no need to use TConvTypeProcs directly. To register a conversion type that uses custom conversion functions, call RegisterConversionType (see page 6411) using the syntax that supplies two conversion functions.

You can, however, use TConvTypeProcs as a base class when implementing your own conversion type that performs addition common actions before or after the custom conversion procedure. For example, you could write a class that always rounds the result to a specified number of decimal places after a conversion is computed using the supplied functions.

See Also

TConvType (see page 6417)

TConvFamily (see page 6416)

RegisterConversionType (see page 6411)

Convert (see page 6397)

TConvTypeFactor (see page 6382)

33.1.4.1 TConvTypeProcs Members

The following tables list the members exposed by TConvTypeProcs.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6393)	Creates an instance of TConvTypeProcs (see page 6391).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
V	FromCommon (see page 6393)	Converts measurements in the base unit for the conversion family into the units represented by this conversion type.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	ToCommon (see page 6394)	Converts measurements in the units represented by this conversion type to the base units of the conversion family.

Public Properties

	Name	Description
R	ConvFamily (see page 6390)	Identifies the conversion family to which the conversion type belongs.
R	ConvType (see page 6390)	Indicates the registered identifier for the conversion type.
R	Description (see page 6391)	Names the measurement unit represented by the conversion type.

33.1.4.2 TConvTypeProcs.Create Constructor

Creates an instance of TConvTypeProcs (see page 6391).

Pascal

```
constructor Create(const AConvFamily: TConvFamily; const ADescription: string; const
AToCommonProc: TConversionProc; const AFromCommonProc: TConversionProc);
```

C++

```
__fastcall TConvTypeProcs(const TConvFamily AConvFamily, const AnsiString ADescription, const
TConversionProc AToCommonProc, const TConversionProc AFromCommonProc);
```

Description

Call Create to instantiate a TConvTypeProcs (see page 6391) descendant at runtime. This descendant can then be passed to the RegisterConversionType (see page 6411) function in order to register a new conversion type. There is no need to instantiate TConvTypeProcs (see page 6391) instances directly, because the RegisterConversionType (see page 6411) function creates a new TConvTypeProcs (see page 6391) instance when you register a conversion type by specifying two conversion functions.

Create assigns the AConvFamily parameter to the ConvFamily (see page 6390) property, and the ADescription parameter to the Description (see page 6391) property. It saves the AToCommonProc and AFromCommonProc parameters, which it then uses to implement the FromCommon (see page 6393) and ToCommon (see page 6394) methods.

If AConvFamily is not the identifier of a registered conversion family, Create raises an EConversionError (see page 6381) exception.

See Also

RegisterConversionType (see page 6411)

ConvFamily (see page 6390)

Description (see page 6391)

EConversionError (see page 6381)

33.1.4.3 TConvTypeProcs Methods

The methods of the TConvTypeProcs class are listed here.

Public Methods

	Name	Description
	FromCommon (see page 6393)	Converts measurements in the base unit for the conversion family into the units represented by this conversion type.
	ToCommon (see page 6394)	Converts measurements in the units represented by this conversion type to the base units of the conversion family.

33.1.4.3.1 TConvTypeProcs.FromCommon Method

Converts measurements in the base unit for the conversion family into the units represented by this conversion type.

Pascal

```
function FromCommon(const AValue: Double): Double; override;
```

C++

```
virtual __fastcall Double FromCommon(const Double AValue);
```

Description

FromCommon provides part of the underlying implementation of the global Convert (see page 6397) function. In order to convert a measurement from one set of units to another, Convert (see page 6397) calls the ToCommon (see page 6394) method of the source conversion type's TConvTypeProcs (see page 6391) instance to obtain the measurement in the base units of the conversion family, and then calls the FromCommon method of the destination conversion type's TConvTypeProcs (see page 6391) instance to obtain the measurement in the destination units.

AValue is a value expressed in the base units of the conversion family.

FromCommon executes the AFromCommonProc function that was supplied to the constructor.

See Also

ToCommon (see page 6394)

ConvFamily (see page 6390)

Convert (see page 6397)

Create (see page 6393)

33.1.4.3.2 TConvTypeProcs.ToCommon Method

Converts measurements in the units represented by this conversion type to the base units of the conversion family.

Pascal

```
function ToCommon(const AValue: Double): Double; override;
```

C++

```
virtual __fastcall Double ToCommon(const Double AValue);
```

Description

ToCommon provides part of the underlying implementation of the global Convert (see page 6397) function. In order to convert a measurement from one set of units to another, Convert (see page 6397) calls the ToCommon method of the source conversion type's TConvTypeProcs (see page 6391) instance to obtain the measurement in the base units of the conversion family, and then calls the ToCommon method of the destination conversion type's TConvTypeProcs (see page 6391) instance to obtain the measurement in the destination units.

AValue is a value expressed in the units represented by this conversion type.

ToCommon executes the AToCommonProc function that was supplied to the constructor.

See Also

FromCommon (see page 6393)

ConvFamily (see page 6390)

Convert (see page 6397)

Create (see page 6393)

33.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CompatibleConversionType (see page 6396)	Indicates whether a specified conversion type is registered with a specified conversion family.
	CompatibleConversionTypes (see page 6396)	Indicates whether the Convert (see page 6397) function can convert between two specified conversion types.
	Convert (see page 6397)	Converts a measurement from one set of units to another.
	Convert (see page 6397)	Converts a measurement from one set of units to another.
	ConvertFrom (see page 6398)	Converts a measurement from the specified units to the base units of its conversion family.
	ConvertTo (see page 6399)	Converts a measurement from the base units of a conversion family into a specified conversion type.
	ConvFamilyToDescription (see page 6399)	Returns the string description of what a conversion family measures.
	ConvTypeToDescription (see page 6400)	Returns the string description of a conversion type (measurement unit).
	ConvTypeToFamily (see page 6400)	Returns the identifier for the conversion family with which a conversion type or pair of types is registered.
	ConvTypeToFamily (see page 6400)	Returns the identifier for the conversion family with which a conversion type or pair of types is registered.
	ConvUnitAdd (see page 6401)	Adds two measurements and returns the result using a specified unit of measurement.
	ConvUnitCompareValue (see page 6402)	Indicates the relationship between two measurements.
	ConvUnitDec (see page 6402)	Decrements a specified measurement by a specified amount.
	ConvUnitDec (see page 6402)	Decrements a specified measurement by a specified amount.
	ConvUnitDiff (see page 6403)	Subtracts one measurement from another and returns the result using a specified unit of measurement.
	ConvUnitInc (see page 6404)	Increments a specified measurement by a specified amount.
	ConvUnitInc (see page 6404)	Increments a specified measurement by a specified amount.
	ConvUnitSameValue (see page 6404)	Indicates whether two measurements are equivalent.
	ConvUnitToStr (see page 6405)	Formats a measurement and its conversion unit into a human-readable string.
	ConvUnitWithinNext (see page 6406)	Indicates whether a specified measurement exceeds another measurement by at most a specified amount.
	ConvUnitWithinPrevious (see page 6406)	Indicates whether a specified measurement is at most a specified amount less than another measurement.
	DescriptionToConvFamily (see page 6407)	Retrieves the identifier for a conversion family given its name.
	DescriptionToConvType (see page 6408)	Retrieves the identifier for a conversion type given its name and family.
	DescriptionToConvType (see page 6408)	Retrieves the identifier for a conversion type given its name and family.
	GetConvFamilies (see page 6408)	Returns a list of all registered conversion families.
	GetConvTypes (see page 6409)	Returns a list of all registered conversion types in a specified conversion family.
	RaiseConversionError (see page 6409)	Raises an EConversionError (see page 6381) exception.
	RaiseConversionError (see page 6409)	Raises an EConversionError (see page 6381) exception.
	RaiseConversionRegError (see page 6410)	This is function ConvUtils.RaiseConversionRegError.
	RegisterConversionFamily (see page 6410)	Registers a new conversion family and returns its identifier.
	RegisterConversionType (see page 6411)	Registers a new conversion type and returns its identifier.
	RegisterConversionType (see page 6411)	Registers a new conversion type and returns its identifier.
	RegisterConversionType (see page 6411)	Registers a new conversion type and returns its identifier.
	StrToConvUnit (see page 6412)	Parses a string into a value and conversion type.
	TryConvTypeToFamily (see page 6413)	Retrieves the identifier for the conversion family with which a conversion type or pair of types is registered.
	TryConvTypeToFamily (see page 6413)	Retrieves the identifier for the conversion family with which a conversion type or pair of types is registered.

	TryStrToConvUnit (see page 6414)	Parses a string into a value and conversion type.
	UnregisterConversionFamily (see page 6414)	Unregisters a conversion family previously registered using RegisterConversionFamily (see page 6410).
	UnregisterConversionType (see page 6415)	Unregisters a conversion type previously registered using RegisterConversionType (see page 6411).

33.2.1 ConvUtils.CompatibleConversionType Function

Indicates whether a specified conversion type is registered with a specified conversion family.

Pascal

```
function CompatibleConversionType(const AType: TConvType; const AFamily: TConvFamily): Boolean;
```

C++

```
Boolean CompatibleConversionType(const TConvType AType, const TConvFamily AFamily);
```

File

ConvUtils

Description

CompatibleConversionType returns true if AType represents a member of the family specified by AFamily, false otherwise.

When a conversion type is a member of a conversion family, the Convert (see page 6397) function can convert between that type and any other type in the family.

See Also

Convert (see page 6397)

RegisterConversionType (see page 6411)

ConvTypeToFamily (see page 6400)

TryConvTypeToFamily (see page 6413)

33.2.2 ConvUtils.CompatibleConversionTypes Function

Indicates whether the Convert (see page 6397) function can convert between two specified conversion types.

Pascal

```
function CompatibleConversionTypes(const AFrom: TConvType; const ATo: TConvType): Boolean;
```

C++

```
Boolean CompatibleConversionTypes(const TConvType AFrom, const TConvType ATo);
```

File

ConvUtils

Description

CompatibleConversionTypes returns true if AFrom and ATo are in the same conversion family, false otherwise.

When two conversion types are in the same family, the Convert (see page 6397) function can convert between measurements that use them.

See Also

Convert (see page 6397)

RegisterConversionType (see page 6411)

ConvTypeToFamily (see page 6400)

GetConvTypes (see page 6409)

CompatibleConversionType (see page 6396)

TryConvTypeToFamily (see page 6413)

Performing Conversions

33.2.3 ConvUtils.Convert Function

Converts a measurement from one set of units to another.

Pascal

```
function Convert(const AValue: Double; const AFrom: TConvType; const ATo: TConvType): Double;
overload;
function Convert(const AValue: Double; const AFrom1: TConvType; const AFrom2: TConvType; const
ATo1: TConvType; const ATo2: TConvType): Double; overload;
```

C++

```
Double Convert(const Double AValue, const TConvType AFrom, const TConvType ATo);
Double Convert(const Double AValue, const TConvType AFrom1, const TConvType AFrom2, const
TConvType ATo1, const TConvType ATo2);
```

File

ConvUtils

Description

Call Convert to convert the value AValue from one set of measurement units to another.

Using the first syntax, the AFrom parameter specifies the current units of the measurement AValue. Convert returns the corresponding measurement in the units specified by ATo. For example, the following line converts a temperature measurement from degrees Fahrenheit to degrees Kelvin:

Note: TempInKelvin := Convert(StrToFloat(Edit1.Text), tuFahrenheit, tuKelvin);

Note: TempInKelvin = Convert(StrToFloat(Edit1->Text), tuFahrenheit, tuKelvin);

Using the second syntax, the AFrom1 parameter and AFrom2 parameter specify the current units of AValue, where the units for AValue are AFrom1 per AFrom2. Convert returns the corresponding measurement in the units ATo1 per ATo2. For example, the following call converts miles per gallon to kilometers per liter:

Note: nKPL := Convert(StrToFloat(Edit1.Text), duMiles, vuGallons, duKilometers, vuLiter);

Note: double nKPL = Convert(StrToFloat(Edit1.Text), duMiles, vuGallons, duKilometers, vuLiter);

The units specified by AFrom and ATo, AFrom1 and ATo1, and AFrom2 and ATo2 must be in the same conversion family (they must measure the same thing). You can check whether two TConvType (see page 6417) values are in the same conversion

family by calling CompatibleConversionTypes (see page 6396). If the units are not compatible, Convert raises an EConversionError (see page 6381) exception.

Note: The StdConvs (see page 20827) unit defines several families of TConvType (see page 6417) values. See conversion family variables for a list of the predefined families of measurement units and the measurement units in each family. In addition, you can register your own conversion families and units using the RegisterConversionType (see page 6411) and RegisterConversionFamily (see page 6410) functions.

Note: Convert performs its conversion by converting AValue from the AFrom units to the base units of its conversion family, and then from the base units to the ATo units. If the AFrom units and the ATo units differ greatly from the base units of the conversion family, this can lead to problems such as overflow. You can ameliorate these problems by registering your own conversion families that are narrower subsets of a broad conversion family.

See Also

RegisterConversionType (see page 6411)

CompatibleConversionTypes (see page 6396)

TConvFamily (see page 6416)

RegisterConversionFamily (see page 6410)

EConversionError (see page 6381)

33.2.4 ConvUtils.ConvertFrom Function

Converts a measurement from the specified units to the base units of its conversion family.

Pascal

```
function ConvertFrom(const AFrom: TConvType; const AValue: Double): Double;
```

C++

```
Double ConvertFrom(const TConvType AFrom, const Double AValue);
```

File

ConvUtils

Description

Call ConvertFrom to convert the value AValue from the units specified by AFrom to the base units of the conversion family for AFrom.

If AFrom is not a registered conversion type, ConvertFrom raises an EConversionError (see page 6381) exception.

See Also

RegisterConversionType (see page 6411)

TConvFamily (see page 6416)

Convert (see page 6397)

ConvertTo (see page 6399)

33.2.5 ConvUtils.ConvertTo Function

Converts a measurement from the base units of a conversion family into a specified conversion type.

Pascal

```
function ConvertTo(const AValue: Double; const ATo: TConvType): Double;
```

C++

```
Double ConvertTo(const Double AValue, const TConvType ATo);
```

File

ConvUtils

Description

Call ConvertTo to convert the value AValue from the base units of the conversion family for ATo into the conversion units specified by ATo.

If ATo is not a registered conversion type, ConvertTo raises an EConversionError (see page 6381) exception.

See Also

RegisterConversionType (see page 6411)

TConvFamily (see page 6416)

Convert (see page 6397)

ConvertFrom (see page 6398)

33.2.6 ConvUtils.ConvFamilyToDescription Function

Returns the string description of what a conversion family measures.

Pascal

```
function ConvFamilyToDescription(const AFamily: TConvFamily): string;
```

C++

```
AnsiString ConvFamilyToDescription(const TConvFamily AFamily);
```

File

ConvUtils

Description

ConvFamilyToDescription returns the string supplied when the conversion family was registered. Typically, this string is the name of whatever it is that the conversion family measures. For example, ConvFamilyToDescription returns "Volume" when passed an argument of cbVolume.

AFamily is the TConvFamily (see page 6416) value that represents the conversion family. This is the value that is returned by RegisterConversionFamily (see page 6410) when the family is registered.

See Also

ConvTYpeToDescription

REgisterConversionFamily

DescriptionToConvFamily (see page 6407)

33.2.7 ConvUtils.ConvTypeToDescription Function

Returns the string description of a conversion type (measurement unit).

Pascal

```
function ConvTypeToDescription(const AType: TConvType): string;
```

C++

```
AnsiString ConvTypeToDescription(const TConvType AType);
```

File

ConvUtils

Description

ConvTypeToDescription returns the string supplied when the conversion type specified by AType was registered. Typically, this string is the (plural) name of a unit of measurement. For example, ConvTypeToDescription returns 'Millennia' when passed an argument of tuMillennia.

See Also

ConvFamilyToDescription (see page 6399)

RegisterConversionType (see page 6411)

DescriptionToConvType (see page 6408)

ConvTypeToFamily (see page 6400)

ConvUnitToStr (see page 6405)

33.2.8 ConvUtils.ConvTypeToFamily Function

Returns the identifier for the conversion family with which a conversion type or pair of types is registered.

Pascal

```
function ConvTypeToFamily(const AType: TConvType): TConvFamily; overload;
function ConvTypeToFamily(const AFrom: TConvType; const ATo: TConvType): TConvFamily; overload;
```

C++

```
TConvFamily ConvTypeToFamily(const TConvType AType);
TConvFamily ConvTypeToFamily(const TConvType AFrom, const TConvType ATo);
```

File

ConvUtils

Description

ConvTypeToFamily returns the conversion family identifier for the family of the specified conversion type or types. Conversion types are associated with a conversion family by the RegisterConversionType (see page 6411) function when the conversion type is registered.

AType is the identifier for a registered conversion type (unit of measurement).

AFrom and ATo are two registered conversion types that are assumed to be in the same family.

If AType is not registered, or if AFrom and ATo are not registered with the same conversion family, ConvTypeToFamily returns CIllegalConvFamily (see page 6420) (0).

See Also

DescriptionToConvFamily (see page 6407)

RegisterConversionType (see page 6411)

GetConvTypes (see page 6409)

ConvTypeToDescription (see page 6400)

CompatibleConversionType (see page 6396)

TryConvTypeToFamily (see page 6413)

33.2.9 ConvUtils.ConvUnitAdd Function

Adds two measurements and returns the result using a specified unit of measurement.

Pascal

```
function ConvUnitAdd(const AValue1: Double; const AType1: TConvType; const AValue2: Double;
const AType2: TConvType; const AResultType: TConvType): Double;
```

C++

```
Double ConvUnitAdd(const Double AValue1, const TConvType AType1, const Double AValue2, const
TConvType AType2, const TConvType AResultType);
```

File

ConvUtils

Description

Call ConvUnitAdd two add two measurements and obtain the result in a specified conversion type.

AValue1 and AValue2 are the two measurements to be added.

AType1 is the unit of measurement for AValue1.

AType2 is the unit of measurement for AValue2.

AResultType is the unit of measurement for the resulting value.

See Also

ConvUnitDec (see page 6402)

ConvUnitInc (see page 6404)

ConvUnitDiff (see page 6403)

33.2.10 ConvUtils.ConvUnitCompareValue Function

Indicates the relationship between two measurements.

Pascal

```
function ConvUnitCompareValue(const AValue1: Double; const AType1: TConvType; const AValue2:
Double; const AType2: TConvType): TValueRelationship;
```

C++

```
TValueRelationship ConvUnitCompareValue(const Double AValue1, const TConvType AType1, const
Double AValue2, const TConvType AType2);
```

File

ConvUtils

Description

Call ConvUnitCompareValue to compare the two measurements. The two measurements must measure the same thing (use the same conversion family), but do not need to use the same units of measurement.

AValue 1 and AValue2 are the two measurements to compare.

AType1 is the unit of measurement for AValue1.

AType2 is the unit of measurement for AValue2.

ConvUnitCompareValue returns

LessThanValue if AValue1 represents a smaller value than AValue2.

EqualsValue if AValue1 is equivalent to AValue2.

GreaterThanValue if AValue1 represents a larger value than AValue2.

See Also

ConvUnitSameValue (see page 6404)

CompareValue

33.2.11 ConvUtils.ConvUnitDec Function

Decrements a specified measurement by a specified amount.

Pascal

```
function ConvUnitDec(const AValue: Double; const AType: TConvType; const AAmountType:
TConvType): Double; overload;
function ConvUnitDec(const AValue: Double; const AType: TConvType; const AAmount: Double;
const AAmountType: TConvType): Double; overload;
```

C++

```
Double ConvUnitDec(const Double AValue, const TConvType AType, const TConvType AAmountType);
Double ConvUnitDec(const Double AValue, const TConvType AType, const Double AAmount, const
TConvType AAmountType);
```

File

ConvUtils

Description

Call ConvUnitDec two decrement a measurement by a specified amount. The result is returned in the same measurement units as the original amount, although the amount by which the measurement is decremented can be expressed using a different unit of measurement in the same family.

AValue is the original measurement.

AType is the unit of measurement of the starting amount and the return value.

AAmount is the amount by which to decrease AValue. If this value is omitted, it is assumed to be one.

AAmountType is the unit of measurement for AAmount.

See Also

ConvUnitAdd (see page 6401)

ConvUnitInc (see page 6404)

ConvUnitDiff (see page 6403)

Dec

33.2.12 ConvUtils.ConvUnitDiff Function

Subtracts one measurement from another and returns the result using a specified unit of measurement.

Pascal

```
function ConvUnitDiff(const AValue1: Double; const AType1: TConvType; const AValue2: Double;
const AType2: TConvType; const AResultType: TConvType): Double;
```

C++

```
Double ConvUnitDiff(const Double AValue1, const TConvType AType1, const Double AValue2, const
TConvType AType2, const TConvType AResultType);
```

File

ConvUtils

Description

Call ConvUnitDiff to find the difference between two measurements, with the result expressed in a specified conversion type.

AValue1 is the starting value.

AType1 is the unit of measurement for AValue1.

AValue2 is the value to be subtracted from AValue1.

AType2 is the unit of measurement for AValue2.

AResultType is the unit of measurement for the resulting value.

See Also

ConvUnitAdd (see page 6401)

ConvUnitInc (see page 6404)

ConvUnitDec (see page 6402)

ConvUnitWithinNext (see page 6406)

ConvUnitWithinPrevious (see page 6406)

33.2.13 ConvUtils.ConvUnitInc Function

Increments a specified measurement by a specified amount.

Pascal

```
function ConvUnitInc(const AValue: Double; const AType: TConvType; const AAmountType:
TConvType): Double; overload;
function ConvUnitInc(const AValue: Double; const AType: TConvType; const AAmount: Double;
const AAmountType: TConvType): Double; overload;
```

C++

```
Double ConvUnitInc(const Double AValue, const TConvType AType, const TConvType AAmountType);
Double ConvUnitInc(const Double AValue, const TConvType AType, const Double AAmount, const
TConvType AAmountType);
```

File

ConvUtils

Description

Call ConvUnitInc to increment a measurement by a specified amount. The result is returned in the same measurement units as the original amount, although the amount by which the measurement is incremented can be expressed using a different unit of measurement in the same family.

AValue is the original measurement.

AType is the unit of measurement of the starting amount and the return value.

AAmount is the amount by which to increase AValue. If this parameter is omitted, it is assumed to be one.

AAmountType is the unit of measurement for AAmount.

See Also

ConvUnitAdd (see page 6401)

ConvUnitDec (see page 6402)

ConvUnitDiff (see page 6403)

Inc

33.2.14 ConvUtils.ConvUnitSameValue Function

Indicates whether two measurements are equivalent.

Pascal

```
function ConvUnitSameValue(const AValue1: Double; const AType1: TConvType; const AValue2:
Double; const AType2: TConvType): Boolean;
```

C++

```
Boolean ConvUnitSameValue(const Double AValue1, const TConvType AType1, const Double AValue2,
const TConvType AType2);
```

File

ConvUtils

Description

Call ConvUnitSameValue to compare determine whether two measurements are equivalent. The two values can use different units of measurement, but the two units of measurement must be in the same conversion family.

AValue 1 and AValue2 are the two measurements to compare.

AType1 is the unit of measurement for AValue1.

AType2 is the unit of measurement for AValue2.

ConvUnitSameValue returns true if the two measurements are equivalent, false otherwise.

See Also

ConvUnitCompareValue (see page 6402)

SameValue

33.2.15 ConvUtils.ConvUnitToStr Function

Formats a measurement and its conversion unit into a human-readable string.

Pascal

```
function ConvUnitToStr(const AValue: Double; const AType: TConvType): string;
```

C++

```
AnsiString ConvUnitToStr(const Double AValue, const TConvType AType);
```

File

ConvUtils

Description

Call ConvUnitToStr to obtain a string representation of a measurement.

AValue is the number of units.

AType is the conversion type (unit of measurement).

ConvUnitToString formats the result as the string representation of AValue followed by the description that was provided when AType was registered. Thus, for example,

ConvUnitToStr(3.5, duMiles)

returns the string "3.5 Miles"

Warning: The resulting string may seem awkward, depending on the description provided when registering the conversion type. For example, ConvUnitToStr(53, duFahrenheit) produces the string "53 Fahrenheit", not "53 degrees Fahrenheit".

See Also

StrToConvUnit (see page 6412)

RegisterConversionType (see page 6411)

33.2.16 ConvUtils.ConvUnitWithinNext Function

Indicates whether a specified measurement exceeds another measurement by at most a specified amount.

Pascal

```
function ConvUnitWithinNext(const AValue: Double; const ATest: Double; const AType: TConvType;
const AAmount: Double; const AAmountType: TConvType): Boolean;
```

C++

```
Boolean ConvUnitWithinNext(const Double AValue, const Double ATest, const TConvType AType,
const Double AAmount, const TConvType AAmountType);
```

File

ConvUtils

Description

Call ConvUnitWithinNext to determine whether the value specified by ATest exceeds the value specified by AValue by at most AAmount.

AValue is the base value against which to check ATest.

ATest is a measurement to test against AValue.

AType is the unit of measurement for both AValue and ATest.

AAmount is the amount by which ATest can exceed AValue.

AAmountType is the unit of measurement for AAmount.

ConvUnitWithinNext returns true if ATest equals AValue or exceeds it by at most AAmount.

See Also

ConvUnitDiff (see page 6403)

ConvUnitWithinPrevious (see page 6406)

33.2.17 ConvUtils.ConvUnitWithinPrevious Function

Indicates whether a specified measurement is at most a specified amount less than another measurement.

Pascal

```
function ConvUnitWithinPrevious(const AValue: Double; const ATest: Double; const AType:
TConvType; const AAmount: Double; const AAmountType: TConvType): Boolean;
```

C++

```
Boolean ConvUnitWithinPrevious(const Double AValue, const Double ATest, const TConvType AType,
const Double AAmount, const TConvType AAmountType);
```

File

ConvUtils

Description

Call ConvUnitWithinPrevious to determine whether the value specified by ATest is less than the value specified by AValue by at most AAmount.

AValue is the base value against which to check ATest.

ATest is a measurement to test against AValue.

AType is the unit of measurement for both AValue and ATest.

AAmount is the amount by which ATest can be less than AValue.

AAmountType is the unit of measurement for AAmount.

ConvUnitWithinPrevious returns true if ATest equals AValue or is at most AAmount less than AValue.

See Also

ConvUnitDiff (see page 6403)

ConvUnitWithinPrevious

33.2.18 ConvUtils.DescriptionToConvFamily Function

Retrieves the identifier for a conversion family given its name.

Pascal

```
function DescriptionToConvFamily(const ADescription: string; out AFamily: TConvFamily):
Boolean;
```

C++

```
Boolean DescriptionToConvFamily(const AnsiString ADescription, TConvFamily AFamily);
```

File

ConvUtils

Description

DescriptionToConvFamily retrieves the conversion family identifier for the family with the name specified by ADescription.

ADescription is the name supplied to the RegisterConversionFamily (see page 6410) method when the family was registered. Typically, it identifies the type of thing measured by conversion types in the family. For example, the description for the cbTemperature family is 'Temperature'.

AFamily returns the identifier for the specified conversion family when DescriptionToConvFamilyt returns true.

DescriptionToConvFamily returns true if ADescription identifies a registered conversion family, false otherwise.

See Also

ConvTypeToFamily (see page 6400)

RegisterConversionFamily (see page 6410)

DescriptionToConvType (see page 6408)

ConvFamilyToDescription (see page 6399)

TryConvTypeToFamily (see page 6413)

33.2.19 ConvUtils.DescriptionToConvType Function

Retrieves the identifier for a conversion type given its name and family.

Pascal

```
function DescriptionToConvType(const ADescription: string; out AType: TConvType): Boolean;
overload;
function DescriptionToConvType(const AFamily: TConvFamily; const ADescription: string; out
AType: TConvType): Boolean; overload;
```

C++

```
Boolean DescriptionToConvType(const AnsiString ADescription, TConvType AType);
Boolean DescriptionToConvType(const TConvFamily AFamily, const AnsiString ADescription,
TConvType AType);
```

File

ConvUtils

Description

DescriptionToConvType returns the conversion type identifier for the conversion type (measurement unit) that was registered under the name specified by ADescription.

AFamily is the identifier for the conversion family supplied to the RegisterConversionType (see page 6411) function when the conversion type was registered.

ADescription is the name supplied to the RegisterConversionType (see page 6411) function when the conversion type was registered. Typically, it is the plural form of the measurement unit name. For example, the description for the auAcres conversion type is 'Acres'.

AType returns the identifier for the conversion type.

DescriptionToConvType returns true if it can successfully locate and return the specified conversion type, false otherwise.

See Also

ConvTypeToDescription (see page 6400)

RegisterConversionType (see page 6411)

DescriptionToConvFamily (see page 6407)

StrToConvUnit (see page 6412)

33.2.20 ConvUtils.GetConvFamilies Function

Returns a list of all registered conversion families.

Pascal

```
procedure GetConvFamilies(out AFamilies: TConvFamilyArray);
```

C++

```
GetConvFamilies(TConvFamilyArray AFamilies);
```

File

ConvUtils

Description

GetConvFamilies returns a dynamic array of identifiers for all currently registered conversion families. Every entry in the array represents a conversion family registered by the RegisterConversionFamily (see page 6410) function.

See Also

RegisterConversionFamily (see page 6410)

GetConvTypes (see page 6409)

33.2.21 ConvUtils.GetConvTypes Function

Returns a list of all registered conversion types in a specified conversion family.

Pascal

```
procedure GetConvTypes(const AFamily: TConvFamily; out ATypes: TConvTypeArray);
```

C++

```
GetConvTypes(const TConvFamily AFamily, TConvTypeArray ATypes);
```

File

ConvUtils

Description

GetConvTypes fills a dynamic array with all the conversion types (TConvType (see page 6417) values) that have been registered for a specified conversion family. Any measurement based on one of these conversion types can be converted into a measurement based on any other in the family.

AFamily is the identifier for the conversion family.

ATypes is the array to fill with conversion type identifiers.

See Also

RegisterConversionType (see page 6411)

GetConvFamilies (see page 6408)

33.2.22 ConvUtils.RaiseConversionError Function

Raises an EConversionError (see page 6381) exception.

Pascal

```
procedure RaiseConversionError(const AText: string); overload;
procedure RaiseConversionError(const AText: string; const AArgs: array of const); overload;
```

C++

```
RaiseConversionError(const AnsiString AText);
RaiseConversionError(const AnsiString AText, const array of const AArgs);
```

File

ConvUtils

Description

Call RaiseConversionError to raise an EConversionError (see page 6381) exception. Most applications do not need to call this method because most conversion errors are raised by the Convert (see page 6397) function.

However, you might want to raise an EConversionError (see page 6381) exception from a method you supply to the RegisterConversionType (see page 6411) function for converting between a new conversion type and the base units of a conversion family. For example, if a unit of measurement does not permit negative values, you could check for negative values and call RaiseConversionError if an attempt is made to use one in a conversion.

AText is the error message for the exception that is raised. It can contain standard string formatting specifiers.

AArgs is a list of arguments that are supplied for the format specifiers in AText.

Note: In C++, Args_Size is the index of the last argument in AArgs. (One less than the number of elements).

See Also

EConversionError (see page 6381)

RegisterConversionType (see page 6411)

Convert (see page 6397)

33.2.23 ConvUtils.RaiseConversionRegError Function

Pascal

```
procedure RaiseConversionRegError(AFamily: TConvFamily; const ADescription: string);
```

C++

```
RaiseConversionRegError(TConvFamily AFamily, const AnsiString ADescription);
```

File

ConvUtils

Description

This is function ConvUtils.RaiseConversionRegError.

33.2.24 ConvUtils.RegisterConversionFamily Function

Registers a new conversion family and returns its identifier.

Pascal

```
function RegisterConversionFamily(const ADescription: string): TConvFamily;
```

C++

```
TConvFamily RegisterConversionFamily(const AnsiString ADescription);
```

File

ConvUtils

Description

Call RegisterConversionFamily to create a new conversion family. A conversion family represents a type of thing that can be measured using different measurement units. For example, the StdConvs (see page 20827) unit defines conversion families for distance, volume, area, temperature, and so on.

Once you have registered a new conversion family, you can register units of measurement that belong to the family using the RegisterConversionType (see page 6411) function. RegisterConversionType (see page 6411) takes the conversion family identifier returned by RegisterConversionFamily as an argument. The global Convert (see page 6397) function can then convert between any two conversion types (units of measurement) that have been registered with the same conversion family.

ADescription is a string that names the conversion family. Typically, this names the thing measured, such as 'Distance', 'Volume', 'Area', 'Temperature', and so on.

RegisterConversionFamily returns an identifier for the new conversion family.

See Also

RegisterConversionType (see page 6411)

UnregisterConversionFamily (see page 6414)

GetConvFamilies (see page 6408)

Convert (see page 6397)

Adding New Measurement Types

Converting Measurements

33.2.25 ConvUtils.RegisterConversionType Function

Registers a new conversion type and returns its identifier.

Pascal

```
function RegisterConversionType(const AFamily: TConvFamily; const ADescription: string; const
AFactor: Double): TConvType; overload;
function RegisterConversionType(const AFamily: TConvFamily; const ADescription: string; const
AToCommonProc: TConversionProc; const AFromCommonProc: TConversionProc): TConvType; overload;
function RegisterConversionType(AConvTypeInfo: TConvTypeInfo; out AType: TConvType): Boolean;
overload;
```

C++

```
TConvType RegisterConversionType(const TConvFamily AFamily, const AnsiString ADescription,
const Double AFactor);
TConvType RegisterConversionType(const TConvFamily AFamily, const AnsiString ADescription,
const TConversionProc AToCommonProc, const TConversionProc AFromCommonProc);
Boolean RegisterConversionType(TConvTypeInfo AConvTypeInfo, TConvType AType);
```

File

ConvUtils

Description

Call RegisterConversionType to add a new conversion type (unit of measurement) to a registered conversion family. The global Convert (see page 6397) function can convert between any two conversion types that have been registered with the same conversion family.

AFamily is the identifier for the conversion family to which the new conversion type belongs. This can be one of the pre-defined

conversion families in the StdConvs (see page 20827) unit, or the value returned from a call to RegisterConversionFamily (see page 6410).

ADescription is a string that names the conversion type. Typically, this is the plural form of the measurement unit name. For example, the description for the duMeters conversion type is 'Meters'.

AFactor is the factor by which a measurement using the new conversion type must be multiplied to convert to the base units of the conversion family. Thus, for example, the factor for duKilometers is 1000 because the base unit for the distance family is meters.

AToCommonProc and AFromCommonProc are functions that convert between the new conversion type and the base units for the conversion family. They are used when conversions between the new conversion type and the base units can't be accomplished by a simple multiplication. For example, temperature measurements use different values for the 0 point, and so require conversion functions rather than a factor. AToCommonProc converts measurements in the new conversion type to the base units for the family, while AFromCommonProc converts measurements in the base units for the family to the new conversion type.

AConvTypeInfo is an instance of a TConvTypeInfo (see page 6386) descendant that represents the new conversion type. On entry, its ConvFamily and Description properties must be set to the conversion family that contains the new conversion type and the string that names the new type (analogous to the AFamily and ADescription parameters). In addition, its ToCommon and FromCommon methods convert between the new conversion type and the base units of the conversion family (analogous to the AToCommonProc and AFromCommonProc parameters). RegisterConversionType sets its ConvType property, which it also returns as the AType parameter. When using this syntax, RegisterConversionType returns true if the new type is successfully registered, false if the new type could not be registered (either because the family was not registered or because the description was already registered).

See Also

RegisterConversionFamily (see page 6410)

UnregisterConversionType (see page 6415)

GetConvTypes (see page 6409)

Convert (see page 6397)

Adding New Measurement Types

Converting Measurements

33.2.26 ConvUtils.StrToConvUnit Function

Parses a string into a value and conversion type.

Pascal

```
function StrToConvUnit(AText: string; out AType: TConvType): Double;
```

C++

```
Double StrToConvUnit(AnsiString AText, TConvType AType);
```

File

ConvUtils

Description

Call StrToConvUnit to reverse the conversion performed by the ConvUnitToStr (see page 6405) function.

AText is the string to parse. It should represent a numeric value followed by the name of a conversion type. For example, '2.5 hours'.

AType returns the identifier for the conversion type (unit of measurement).

StrToConvUnit returns the number of units (the numeric portion of the string).

If AText can't be parsed into a value and conversion type, StrToConvUnit raises an EConversionError (see page 6381) exception.

See Also

ConvUnitToStr (see page 6405)

RegisterConversionType (see page 6411)

DescriptionToConvType (see page 6408)

TryStrToConvUnit (see page 6414)

EConversionError (see page 6381)

33.2.27 ConvUtils.TryConvTypeToFamily Function

Retrieves the identifier for the conversion family with which a conversion type or pair of types is registered.

Pascal

```
function TryConvTypeToFamily(const AType: TConvType; out AFamily: TConvFamily): Boolean;
overload;
function TryConvTypeToFamily(const AFrom: TConvType; const ATo: TConvType; out AFamily:
TConvFamily): Boolean; overload;
```

C++

```
Boolean TryConvTypeToFamily(const TConvType AType, TConvFamily AFamily);
Boolean TryConvTypeToFamily(const TConvType AFrom, const TConvType ATo, TConvFamily AFamily);
```

File

ConvUtils

Description

TryConvTypeToFamily looks up the conversion family identifier for the family of the specified conversion type or types. Conversion types are associated with a conversion family by the RegisterConversionType (see page 6411) function when the conversion type is registered.

AType is the identifier for a registered conversion type (unit of measurement).

AFrom and ATo are two registered conversion types that are assumed to be in the same family.

AFamily returns the identifier of the conversion family.

TryConvTypeToFamily returns true if the family can be obtained (if AType is a registered conversion type or if AFrom and ATo are both registered in the same family), false otherwise.

See Also

DescriptionToConvFamily (see page 6407)

RegisterConversionType (see page 6411)

GetConvTypes (see page 6409)

ConvTypeToDescription (see page 6400)

CompatibleConversionType (see page 6396)

ConvTypeToFamily (see page 6400)

33.2.28 ConvUtils.TryStrToConvUnit Function

Parses a string into a value and conversion type.

Pascal

```
function TryStrToConvUnit(AText: string; out AValue: Double; out AType: TConvType): Boolean;
```

C++

```
Boolean TryStrToConvUnit(AnsiString AText, Double AValue, TConvType AType);
```

File

ConvUtils

Description

Call TryStrToConvUnit to reverse the conversion performed by the ConvUnitToStr (see page 6405) function.

AText is the string to parse. It should represent a numeric value followed by the name of a conversion type. For example, '2.5 hours'.

AValue returns the number of units (the numeric value in the string).

AType returns the identifier for the conversion type (unit of measurement).

TryStrToConvUnit returns true if AText can be converted, false otherwise.

See Also

ConvUnitToStr (see page 6405)

RegisterConversionType (see page 6411)

DescriptionToConvType (see page 6408)

StrToConvUnit (see page 6412)

33.2.29 ConvUtils.UnregisterConversionFamily Function

Unregisters a conversion family previously registered using RegisterConversionFamily (see page 6410).

Pascal

```
procedure UnregisterConversionFamily(const AFamily: TConvFamily);
```

C++

```
UnregisterConversionFamily(const TConvFamily AFamily);
```

File

ConvUtils

Description

Call UnregisterConversionFamily when a conversion family becomes unavailable. Typically, this function is called from the finalization section of a unit that implements the conversion family. UnregisterConversionFamily unregisters all of the conversion types registered with the family and then unregisters the family itself.

AFamily is the conversion family identifier for the family to unregister.

After a conversion family is unregistered, it is no longer listed by the GetConvFamilies (see page 6408) procedure. Calling the Convert (see page 6397) function with conversion types from an unregistered family results in an EConversionError (see page 6381) exception.

See Also

UnregisterConversionType (see page 6415)

RegisterConversionFamily (see page 6410)

GetConvFamilies (see page 6408)

EConversionError (see page 6381)

Convert (see page 6397)

33.2.30 ConvUtils.UnregisterConversionType Function

Unregisters a conversion type previously registered using RegisterConversionType (see page 6411).

Pascal

```
procedure UnregisterConversionType(const AType: TConvType);
```

C++

```
UnregisterConversionType(const TConvType AType);
```

File

ConvUtils

Description

Call UnregisterConversionType when a conversion type becomes unavailable. Typically, this function is called from the finalization section of a unit that implements the conversion type.

AType is the conversion type identifier for the type to unregister.

After a conversion type is unregistered, it is no longer listed by the GetConvTypes (see page 6409) procedure. Calling the Convert (see page 6397) function with an unregistered conversion type results in an EConversionError (see page 6381) exception.

See Also

UnregisterConversionFamily (see page 6414)

RegisterConversionType (see page 6411)

GetConvTypes (see page 6409)

Convert (see page 6397)

EConversionError (see page 6381)

33.3 Types

The following table lists types in this documentation.

Types

Name	Description
TConversionProc (see page 6416)	This is type ConvUtils.TConversionProc.
TConvFamily (see page 6416)	This is type ConvUtils.TConvFamily.
TConvFamilyArray (see page 6417)	TConvFamilyArray represents a dynamic array of conversion families.
TConvType (see page 6417)	This is type ConvUtils.TConvType.
TConvTypeArray (see page 6417)	TConvTypeArray represents a dynamic array of conversion types (measurement units)
TConvTypeList (see page 6418)	This is type ConvUtils.TConvTypeList.

33.3.1 ConvUtils.TConversionProc Type

Pascal

```
TConversionProc = function (const AValue: Double): Double;
```

C++

```
(const AValue: Double): Double ( TConversionProc)();
```

File

ConvUtils

Description

This is type ConvUtils.TConversionProc.

33.3.2 ConvUtils.TConvFamily Type

Pascal

```
TConvFamily = type Word;
```

C++

```
type Word TConvFamily;
```

File

ConvUtils

Description

This is type ConvUtils.TConvFamily.

33.3.3 ConvUtils.TConvFamilyArray Type

TConvFamilyArray represents a dynamic array of conversion families.

Pascal

```
TConvFamilyArray = array of TConvFamily;
```

C++

```
array of TConvFamily TConvFamilyArray;
```

File

ConvUtils

Description

TConvFamilyArray is a dynamic array of TConvFamily (see page 6416) values. Each value in the array is the identifier for a family of TConvType (see page 6417) values. That is, each entry represents a set of registered conversion types that can be converted into each other.

See Also

IDH

33.3.4 ConvUtils.TConvType Type

Pascal

```
TConvType = type Word;
```

C++

```
type Word TConvType;
```

File

ConvUtils

Description

This is type ConvUtils.TConvType.

33.3.5 ConvUtils.TConvTypeArray Type

TConvTypeArray represents a dynamic array of conversion types (measurement units)

Pascal

```
TConvTypeArray = array of TConvType;
```

C++

```
array of TConvType TConvTypeArray;
```

File

ConvUtils

Description

TConvTypeArray is a dynamic array of TConvType (see page 6417) values. Each value in the array is the identifier for a single conversion type (measurement unit).

See Also

IDH

33.3.6 ConvUtils.TConvTypeList Type

Pascal

```
TConvTypeList = array of TConvTypeInfo;
```

C++

```
array of TConvTypeInfo TConvTypeList;
```

File

ConvUtils

Description

This is type ConvUtils.TConvTypeList.

33.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
GConvUnitToStrFmt (see page 6419)	This is variable ConvUtils.GConvUnitToStrFmt.

33.4.1 ConvUtils.GConvUnitToStrFmt Variable

Pascal

```
GConvUnitToStrFmt: string = '%f %s';
```

C++

```
AnsiString GConvUnitToStrFmt = '%f %s';
```

File

ConvUtils

Description

This is variable ConvUtils.GConvUnitToStrFmt.

33.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CIllegalConvFamily (see page 6420)	This is constant ConvUtils.CIllegalConvFamily.
CIllegalConvType (see page 6420)	This is constant ConvUtils.CIllegalConvType.

33.5.1 ConvUtils.CIllegalConvFamily Constant

Pascal

```
CIllegalConvFamily: TConvFamily = 0;
```

C++

```
TConvFamily CIllegalConvFamily = 0;
```

File

ConvUtils

Description

This is constant ConvUtils.CIllegalConvFamily.

33.5.2 ConvUtils.CIllegalConvType Constant

Pascal

```
CIllegalConvType: TConvType = 0;
```

C++

```
TConvType CIllegalConvType = 0;
```

File

ConvUtils

Description

This is constant ConvUtils.CIllegalConvType.

34 CorbaObj Namespace

This is namespace CorbaObj.

Topics

Name	Description
Classes (see page 6423)	The following table lists classes in this documentation.
Interfaces (see page 6475)	The following table lists interfaces in this documentation.
Functions (see page 6477)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6484)	The following table lists structs, records, enums in this documentation.
Types (see page 6487)	The following table lists types in this documentation.
Variables (see page 6493)	The following table lists variables in this documentation.

Classes

	Name	Description
	ECorbaDispatch (see page 6423)	This is class CorbaObj.ECorbaDispatch.
	ECorbaException (see page 6423)	This is class CorbaObj.ECorbaException.
	ECorbaUserException (see page 6426)	This is class CorbaObj.ECorbaUserException.
	TBOA (see page 6431)	This is class CorbaObj.TBOA.
	TCorbaDispatchStub (see page 6433)	This is class CorbaObj.TCorbaDispatchStub.
	TCorbaFactory (see page 6436)	This is class CorbaObj.TCorbaFactory.
	TCorbaFactoryManager (see page 6442)	This is class CorbaObj.TCorbaFactoryManager.
	TCorbaImplementation (see page 6444)	This is class CorbaObj.TCorbaImplementation.
	TCorbaInterfaceIDManager (see page 6449)	This is class CorbaObj.TCorbaInterfaceIDManager.
	TCorbaListManager (see page 6451)	This is class CorbaObj.TCorbaListManager.
	TCorbaObjectFactory (see page 6453)	This is class CorbaObj.TCorbaObjectFactory.
	TCorbaSkeleton (see page 6456)	This is class CorbaObj.TCorbaSkeleton.
	TCorbaSkeletonManager (see page 6460)	This is class CorbaObj.TCorbaSkeletonManager.
	TCorbaStub (see page 6462)	This is class CorbaObj.TCorbaStub.
	TCorbaStubManager (see page 6466)	This is class CorbaObj.TCorbaStubManager.
	TORB (see page 6467)	This is class CorbaObj.TORB.

Functions

	Name	Description
	AnyToObject (see page 6477)	This is function CorbaObj.AnyToObject.
	BOA (see page 6477)	This is function CorbaObj.BOA.
	CorbaBind (see page 6478)	This is function CorbaObj.CorbaBind.

	CorbaBind (see page 6478)	This is function CorbaObj.CorbaBind.
	CorbaFactoryCreateStub (see page 6478)	This is function CorbaObj.CorbaFactoryCreateStub.
	CorbaInitialize (see page 6479)	This is function CorbaObj.CorbaInitialize.
	MakePrincipal (see page 6479)	This is function CorbaObj.MakePrincipal.
	MarshalAny (see page 6479)	This is function CorbaObj.MarshalAny.
	MarshalObject (see page 6480)	This is function CorbaObj.MarshalObject.
	MarshalWordBool (see page 6480)	This is function CorbaObj.MarshalWordBool.
	ORB (see page 6480)	This is function CorbaObj.ORB.
	SequenceToVariantArray (see page 6481)	This is function CorbaObj.SequenceToVariantArray.
	UnmarshalAny (see page 6481)	This is function CorbaObj.UnmarshalAny.
	UnmarshalObject (see page 6481)	This is function CorbaObj.UnmarshalObject.
	UnmarshalText (see page 6482)	This is function CorbaObj.UnmarshalText.
	UnmarshalWideText (see page 6482)	This is function CorbaObj.UnmarshalWideText.
	UnmarshalWordBool (see page 6482)	This is function CorbaObj.UnmarshalWordBool.
	VariantArrayToSequence (see page 6483)	This is function CorbaObj.VariantArrayToSequence.

Interfaces

	Name	Description
	ICorbaObject (see page 6475)	This is class CorbaObj.ICorbaObject.

Structs, Records, Enums

	Name	Description
	TCorbaInstancing (see page 6484)	This is record CorbaObj.TCorbaInstancing.
	TCorbaThreadModel (see page 6484)	This is record CorbaObj.TCorbaThreadModel.
	TInterfaceIDEntryDesc (see page 6485)	This is record CorbaObj.TInterfaceIDEntryDesc.
	TSkeletonEntryDesc (see page 6485)	This is record CorbaObj.TSkeletonEntryDesc.
	TStubEntryDesc (see page 6486)	This is record CorbaObj.TStubEntryDesc.

Types

Name	Description
CorbaBoolean (see page 6487)	This is type CorbaObj.CorbaBoolean.
CorbaULong (see page 6487)	This is type CorbaObj.CorbaULong.
IObject (see page 6488)	This is type CorbaObj.IObject.
ITypeCode (see page 6488)	This is type CorbaObj.ITypeCode.
TAny (see page 6488)	This is type CorbaObj.TAny.
TCKind (see page 6489)	This is type CorbaObj.TCKind.
TCommandLine (see page 6489)	This is type CorbaObj.TCommandLine.
TCorbaImplementationClass (see page 6489)	This is type CorbaObj.TCorbaImplementationClass.
TCorbaPrincipal (see page 6490)	This is type CorbaObj.TCorbaPrincipal.
TCorbaSkeletonClass (see page 6490)	This is type CorbaObj.TCorbaSkeletonClass.
TCorbaStubClass (see page 6490)	This is type CorbaObj.TCorbaStubClass.
TFactoryList (see page 6491)	This is type CorbaObj.TFactoryList.
TInterfaceIDList (see page 6491)	This is type CorbaObj.TInterfaceIDList.
TSkeletonList (see page 6491)	This is type CorbaObj.TSkeletonList.
TStubList (see page 6492)	This is type CorbaObj.TStubList.

Variables

Name	Description
CorbaInterfaceIDManager (see page 6493)	This is variable CorbaObj.CorbaInterfaceIDManager.
CorbaSkeletonManager (see page 6493)	This is variable CorbaObj.CorbaSkeletonManager.
CorbaStubManager (see page 6493)	This is variable CorbaObj.CorbaStubManager.

34.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ECorbaDispatch (see page 6423)	This is class CorbaObj.ECorbaDispatch.
	ECorbaException (see page 6423)	This is class CorbaObj.ECorbaException.
	ECorbaUserException (see page 6426)	This is class CorbaObj.ECorbaUserException.
	TBOA (see page 6431)	This is class CorbaObj.TBOA.
	TCorbaDispatchStub (see page 6433)	This is class CorbaObj.TCorbaDispatchStub.
	TCorbaFactory (see page 6436)	This is class CorbaObj.TCorbaFactory.
	TCorbaFactoryManager (see page 6442)	This is class CorbaObj.TCorbaFactoryManager.
	TCorbaImplementation (see page 6444)	This is class CorbaObj.TCorbaImplementation.
	TCorbaInterfaceIDManager (see page 6449)	This is class CorbaObj.TCorbaInterfaceIDManager.
	TCorbaListManager (see page 6451)	This is class CorbaObj.TCorbaListManager.
	TCorbaObjectFactory (see page 6453)	This is class CorbaObj.TCorbaObjectFactory.
	TCorbaSkeleton (see page 6456)	This is class CorbaObj.TCorbaSkeleton.
	TCorbaSkeletonManager (see page 6460)	This is class CorbaObj.TCorbaSkeletonManager.
	TCorbaStub (see page 6462)	This is class CorbaObj.TCorbaStub.
	TCorbaStubManager (see page 6466)	This is class CorbaObj.TCorbaStubManager.
	TORB (see page 6467)	This is class CorbaObj.TORB.

34.1.1 ECorbaDispatch Class

Class Hierarchy

Pascal

```
ECorbaDispatch = class(Exception);
```

C++

```
class ECorbaDispatch : public Exception;
```

File

CorbaObj

Description

This is class CorbaObj.ECorbaDispatch.

34.1.2 ECorbaException Class

Class Hierarchy

Pascal

```
ECorbaException = class(Exception);
```

C++

```
class ECorbaException : public Exception;
```

File

CorbaObj

Description

This is class CorbaObj.ECorbaException.

34.1.2.1 ECorbaException Members

The following tables list the members exposed by ECorbaException.

Protected Methods

	Name	Description
	GetMessage (see page 6425)	This is GetMessage, a member of class ECorbaException.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 22752)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.
R	Name (see page 6425)	This is Name, a member of class ECorbaException.

34.1.2.2 ECorbaException Methods

The methods of the ECorbaException class are listed here.

Protected Methods

	Name	Description
	GetMessage (see page 6425)	This is GetMessage, a member of class ECorbaException.

34.1.2.2.1 ECorbaException.GetMessage Method

Pascal

```
function GetMessage: string;
```

C++

```
__fastcall AnsiString GetMessage();
```

Description

This is GetMessage, a member of class ECorbaException.

34.1.2.3 ECorbaException Properties

The properties of the ECorbaException class are listed here.

Public Properties

	Name	Description
R	Name (see page 6425)	This is Name, a member of class ECorbaException.

34.1.2.3.1 ECorbaException.Name Property

Pascal

```
property Name: string;
```

C++

```
__property AnsiString Name;
```

Description

This is Name, a member of class ECorbaException.

34.1.3 ECorbaUserException Class

Class Hierarchy

System::TObject → SysUtils.Exception → CorbaObj.ECorbaException → CorbaObj.ECorbaUserException

Pascal

```
ECorbaUserException = class(ECorbaException);
```

C++

```
class ECorbaUserException : public ECorbaException;
```

File

CorbaObj

Description

This is class CorbaObj.ECorbaUserException.

34.1.3.1 ECorbaUserException Members

The following tables list the members exposed by ECorbaUserException.

Protected Methods

	Name	Description
	GetMessage (see page 6425)	This is GetMessage, a member of class ECorbaException.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Copy (see page 6430)	This is Copy, a member of class ECorbaUserException.
	Create (see page 6427)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.

	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Throw (see page 6430)	This is Throw, a member of class ECorbaUserException.

Public Properties

	Name	Description
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.
R	Name (see page 6425)	This is Name, a member of class ECorbaException.
R	Proxy (see page 6430)	This is Proxy, a member of class ECorbaUserException.

34.1.3.2 ECorbaUserException.Create Constructor

Instantiates an instance of an exception with a simple message string.

Pascal

```
constructor Create(const Name: string);
```

C++

```
__fastcall ECorbaUserException(const AnsiString Name);
```

Description

Call Create to construct an exception object with a simple message string.

Msg is the string containing the runtime error message to display in the exception dialog box. Msg can be a hard-coded string, or can be a function call that returns a string.

C++ Examples:

```
/*
The following example creates a directory 'C:\temp' if it
does not already exist.
}
*/
#include <Filectrl.hpp>
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
  if (!DirectoryExists("c:\\temp"))
  {
    if (!CreateDir("C:\\temp"))
      throw Exception("Cannot create c:\\temp directory.");
  }
  if (!CreateDir("C:\\temp"))
    throw Exception("The c:\\temp directory already exists.");
}
```

Delphi Examples:

```
{
The following example creates a directory 'C:\temp' if it
does not already exist.
}
uses FileCtrl;
procedure TForm1.Button1Click(Sender: TObject);
begin
  if not SysUtils.DirectoryExists('C:\temp') then
    if not CreateDir('C:\temp') then
      raise Exception.Create('Cannot create C:\temp');
  if not CreateDir('C:\temp') then
    raise Exception.Create('Cannot create C:\temp again!');
end;
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      Pchar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          { Write out the length of each string, followed by the string itself. }
          StringLen := Length(StringGrid1.Cells[X,Y]);
```

```
            FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
            GetMem(Buffer, StringLen); { allocate the buffer }
            cellString := StringGrid1.Cells[X,Y];
            for I := 1 to StringLen do
               Buffer[I - 1] := cellString[I];
            FileWrite(FileHandle, Buffer^, StringLen);
          finally
            FreeMem(Buffer, StringLen);
          end;
        end;
      end;
      FileClose(FileHandle);
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with StringGrid1 do
    begin
      Cells[1,0] := 'Column 1';
      Cells[2,0] := 'Column 2';
      Cells[3,0] := 'Column 3';
      Cells[4,0] := 'Column 4';
      Cells[0,1] := 'Row 1';
      Cells[1,1] := 'Object';
      Cells[2,1] := 'Pascal';
      Cells[3,1] := 'is';
      Cells[4,1] := 'excellent';
      Cells[0,2] := 'Row 2';
      Cells[1,2] := 'Delphi';
      Cells[2,2] := 'is';
      Cells[4,2] := 'RAD';
    end;
  end;
  {
  In addition to displaying the exception message, which
  happens by default, the following code shuts down the
  application when an exception is not caught and handled.
  AppException should be declared a method of TForm1.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Application.OnException := AppException;
  end;
  procedure TForm1.AppException(Sender: TObject; E: Exception);
  begin
    Application.ShowException(E);
    Application.Terminate;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    raise EPasswordInvalid.Create('Incorrect password entered');
  end;
```

See Also

CreateFmt (see page 22754)

CreateRes (see page 22756)

CreateResFmt (see page 22757)

Message (see page 22760)

34.1.3.3 ECorbaUserException Methods

The methods of the ECorbaUserException class are listed here.

Public Methods

	Name	Description
	Copy (see page 6430)	This is Copy, a member of class ECorbaUserException.
	Throw (see page 6430)	This is Throw, a member of class ECorbaUserException.

34.1.3.3.1 ECorbaUserException.Copy Method

Pascal

```
procedure Copy(const InBuf: IMarshalInBuffer); virtual; abstract;
```

C++

```
virtual __fastcall Copy(const IMarshalInBuffer InBuf) = 0;
```

Description

This is Copy, a member of class ECorbaUserException.

34.1.3.3.2 ECorbaUserException.Throw Method

Pascal

```
procedure Throw;
```

C++

```
__fastcall Throw();
```

Description

This is Throw, a member of class ECorbaUserException.

34.1.3.4 ECorbaUserException Properties

The properties of the ECorbaUserException class are listed here.

Public Properties

	Name	Description
	Proxy (see page 6430)	This is Proxy, a member of class ECorbaUserException.

34.1.3.4.1 ECorbaUserException.Proxy Property

Pascal

```
property Proxy: PUserExceptionProxy;
```

C++

```
__property PUserExceptionProxy Proxy;
```

Description

This is Proxy, a member of class ECorbaUserException.

34.1.4 TBOA Class

Class Hierarchy

CorbaObj.TBOA

Pascal

```
TBOA = class;
```

C++

```
class TBOA;
```

File

CorbaObj

Description

This is class CorbaObj.TBOA.

34.1.4.1 TBOA Members

The following tables list the members exposed by TBOA.

Public Methods

	Name	Description
	Deactivate (see page 6431)	This is Deactivate, a member of class TBOA.
	GetPrincipal (see page 6432)	This is GetPrincipal, a member of class TBOA.
	ImplIsReady (see page 6432)	This is ImplIsReady, a member of class TBOA.
	Initialize (see page 6432)	This is Initialize, a member of class TBOA.
	ObjIsReady (see page 6432)	This is ObjIsReady, a member of class TBOA.

34.1.4.2 TBOA Methods

The methods of the TBOA class are listed here.

Public Methods

	Name	Description
	Deactivate (see page 6431)	This is Deactivate, a member of class TBOA.
	GetPrincipal (see page 6432)	This is GetPrincipal, a member of class TBOA.
	ImplIsReady (see page 6432)	This is ImplIsReady, a member of class TBOA.
	Initialize (see page 6432)	This is Initialize, a member of class TBOA.
	ObjIsReady (see page 6432)	This is ObjIsReady, a member of class TBOA.

34.1.4.2.1 TBOA.Deactivate Method

Pascal

```
procedure Deactivate(const Obj: IObject);
```

C++

```
__fastcall Deactivate(const IObject Obj);
```

Description

This is Deactivate, a member of class TBOA.

34.1.4.2.2 TBOA.GetPrincipal Method

Pascal

```
function GetPrincipal(const Obj: IObject): TCorbaPrincipal;
```

C++

```
__fastcall TCorbaPrincipal GetPrincipal(const IObject Obj);
```

Description

This is GetPrincipal, a member of class TBOA.

34.1.4.2.3 TBOA.ImplIsReady Method

Pascal

```
procedure ImplIsReady;
```

C++

```
__fastcall ImplIsReady();
```

Description

This is ImplIsReady, a member of class TBOA.

34.1.4.2.4 TBOA.Initialize Method

Pascal

```
class procedure Initialize(const CommandLine: TCommandLine);
```

C++

```
__fastcall Initialize(const TCommandLine CommandLine);
```

Description

This is Initialize, a member of class TBOA.

34.1.4.2.5 TBOA.ObjIsReady Method

Pascal

```
procedure ObjIsReady(const Obj: IObject);
```

C++

```
__fastcall ObjIsReady(const IObject Obj);
```

Description

This is ObjIsReady, a member of class TBOA.

34.1.5 TCorbaDispatchStub Class

Class Hierarchy

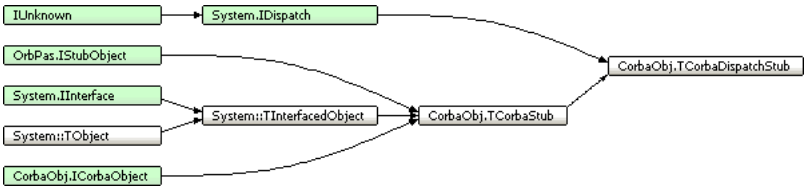

Pascal

```
TCorbaDispatchStub = class(TCorbaStub, IDispatch);
```

C++

```
class TCorbaDispatchStub : public TCorbaStub, public IDispatch;
```

File

CorbaObj

Description

This is class CorbaObj.TCorbaDispatchStub.

34.1.5.1 TCorbaDispatchStub Members

The following tables list the members exposed by TCorbaDispatchStub.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Fields

	Name	Description
	FStub (see page 6464)	This is FStub, a member of class TCorbaStub.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
V	GetIDsOfNames (see page 6435)	This is GetIDsOfNames, a member of class TCorbaDispatchStub.
	GetStub (see page 6465)	This is GetStub, a member of class TCorbaStub.
V	GetTypeInfo (see page 6435)	This is GetTypeInfo, a member of class TCorbaDispatchStub.
V	GetTypeInfoCount (see page 6435)	This is GetTypeInfoCount, a member of class TCorbaDispatchStub.
	Hash (see page 6465)	This is Hash, a member of class TCorbaStub.
V	Invoke (see page 6435)	This is Invoke, a member of class TCorbaDispatchStub.
	IsA (see page 6465)	This is IsA, a member of class TCorbaStub.
	NonExistent (see page 6465)	This is NonExistent, a member of class TCorbaStub.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

	SetPrincipal (see page 6466)	This is SetPrincipal, a member of class TCorbaStub.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 6463)	This is Create, a member of class TCorbaStub.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 6464)	Disposes of an object instance.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetStub (see page 18212)	This is GetStub, a member of class IStubObject.
	Hash (see page 6476)	This is Hash, a member of class ICorbaObject.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsA (see page 6476)	This is IsA, a member of class ICorbaObject.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NonExistent (see page 6476)	This is NonExistent, a member of class ICorbaObject.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetPrincipal (see page 6476)	This is SetPrincipal, a member of class ICorbaObject.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

34.1.5.2 TCorbaDispatchStub Methods

The methods of the TCorbaDispatchStub class are listed here.

Protected Methods

	Name	Description
	GetIDsOfNames (see page 6435)	This is GetIDsOfNames, a member of class TCorbaDispatchStub.
	GetTypeInfo (see page 6435)	This is GetTypeInfo, a member of class TCorbaDispatchStub.
	GetTypeInfoCount (see page 6435)	This is GetTypeInfoCount, a member of class TCorbaDispatchStub.
	Invoke (see page 6435)	This is Invoke, a member of class TCorbaDispatchStub.

34.1.5.2.1 TCorbaDispatchStub.GetIDsOfNames Method

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int
NameCount, int LocaleID, void * DispIDs);
```

Description

This is GetIDsOfNames, a member of class TCorbaDispatchStub.

34.1.5.2.2 TCorbaDispatchStub.GetTypeInfo Method

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; virtual;
stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

This is GetTypeInfo, a member of class TCorbaDispatchStub.

34.1.5.2.3 TCorbaDispatchStub.GetTypeInfoCount Method

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

This is GetTypeInfoCount, a member of class TCorbaDispatchStub.

34.1.5.2.4 TCorbaDispatchStub.Invoke Method

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
```

```
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word
Flags,  Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is Invoke, a member of class TCorbaDispatchStub.

34.1.6 TCorbaFactory Class

Class Hierarchy

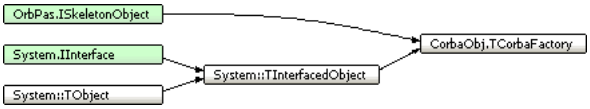

Pascal

```
TCorbaFactory = class(TInterfacedObject, ISkeletonObject);
```

C++

```
class TCorbaFactory : public TInterfacedObject, public ISkeletonObject;
```

File

CorbaObj

Description

This is class CorbaObj.TCorbaFactory.

34.1.6.1 TCorbaFactory Members

The following tables list the members exposed by TCorbaFactory.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
V	CreateInstance (see page 6439)	This is CreateInstance, a member of class TCorbaFactory.
V	CreateInterface (see page 6439)	This is CreateInterface, a member of class TCorbaFactory.
	Execute (see page 6439)	This is Execute, a member of class TCorbaFactory.
	GetImplementation (see page 6439)	This is GetImplementation, a member of class TCorbaFactory.
	GetSkeleton (see page 6440)	This is GetSkeleton, a member of class TCorbaFactory.
	GetTypeInfo (see page 6440)	This is GetTypeInfo, a member of class TCorbaFactory.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterFactory (see page 6440)	This is RegisterFactory, a member of class TCorbaFactory.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6437)	This is Create, a member of class TCorbaFactory.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 6438)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	InstanceName (see page 6440)	This is InstanceName, a member of class TCorbaFactory.
R	Instancing (see page 6441)	This is Instancing, a member of class TCorbaFactory.
R	InterfaceName (see page 6441)	This is InterfaceName, a member of class TCorbaFactory.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	RepositoryID (see page 6441)	This is RepositoryID, a member of class TCorbaFactory.
R	ThreadModel (see page 6441)	This is ThreadModel, a member of class TCorbaFactory.

34.1.6.2 TCorbaFactory.Create Constructor

Pascal

```
constructor Create(const InterfaceName: string; const InstanceName: string; const
```

```
RepositoryId: string; const ImplGUID: TGUID; Instancing: TCorbaInstancing = iMultiInstance;
ThreadModel: TCorbaThreadModel = tmSingleThread);
```

C++

```
__fastcall TCorbaFactory(const AnsiString InterfaceName, const AnsiString InstanceName, const
AnsiString RepositoryId, const TGUID ImplGUID, TCorbaInstancing Instancing = iMultiInstance,
TCorbaThreadModel ThreadModel = tmSingleThread);
```

Description

This is Create, a member of class TCorbaFactory.

34.1.6.3 TCorbaFactory.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCorbaFactory();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

34.1.6.4 TCorbaFactory Methods

The methods of the TCorbaFactory class are listed here.

Protected Methods

	Name	Description
V	CreateInstance (see page 6439)	This is CreateInstance, a member of class TCorbaFactory.
V	CreateInterface (see page 6439)	This is CreateInterface, a member of class TCorbaFactory.
	Execute (see page 6439)	This is Execute, a member of class TCorbaFactory.
	GetImplementation (see page 6439)	This is GetImplementation, a member of class TCorbaFactory.
	GetSkeleton (see page 6440)	This is GetSkeleton, a member of class TCorbaFactory.

	GetTypeInfo (see page 6440)	This is GetTypeInfo, a member of class TCorbaFactory.
	RegisterFactory (see page 6440)	This is RegisterFactory, a member of class TCorbaFactory.

34.1.6.4.1 TCorbaFactory.CreateInstance Method

Pascal

```
function CreateInstance(const InstanceName: string): ISkeletonObject; virtual;
```

C++

```
virtual __fastcall ISkeletonObject CreateInstance(const AnsiString InstanceName);
```

Description

This is CreateInstance, a member of class TCorbaFactory.

34.1.6.4.2 TCorbaFactory.CreateInterface Method

Pascal

```
function CreateInterface(const InstanceName: string): IObject; virtual;
```

C++

```
virtual __fastcall IObject CreateInterface(const AnsiString InstanceName);
```

Description

This is CreateInterface, a member of class TCorbaFactory.

34.1.6.4.3 TCorbaFactory.Execute Method

Pascal

```
function Execute(Operation: PChar; const Strm: IMarshalInBuffer; Cookie: Pointer):
CorbaBoolean; stdcall;
```

C++

```
__fastcall __stdcall CorbaBoolean Execute(const char * Operation, const IMarshalInBuffer Strm,
void * Cookie);
```

Description

This is Execute, a member of class TCorbaFactory.

34.1.6.4.4 TCorbaFactory.GetImplementation Method

Pascal

```
procedure GetImplementation(out Impl: IObject); stdcall;
```

C++

```
__fastcall __stdcall GetImplementation(IObject Impl);
```

Description

This is GetImplementation, a member of class TCorbaFactory.

34.1.6.4.5 TCorbaFactory.GetSkeleton Method

Pascal

```
procedure GetSkeleton(out Skeleton: ISkeleton); stdcall;
```

C++

```
__fastcall __stdcall GetSkeleton(ISkeleton Skeleton);
```

Description

This is GetSkeleton, a member of class TCorbaFactory.

34.1.6.4.6 TCorbaFactory.GetTypeInfo Method

Pascal

```
function GetTypeInfo(out TypeInfo): HRESULT;
```

C++

```
__fastcall HRESULT GetTypeInfo( TypeInfo);
```

Description

This is GetTypeInfo, a member of class TCorbaFactory.

34.1.6.4.7 TCorbaFactory.RegisterFactory Method

Pascal

```
procedure RegisterFactory;
```

C++

```
__fastcall RegisterFactory();
```

Description

This is RegisterFactory, a member of class TCorbaFactory.

34.1.6.5 TCorbaFactory Properties

The properties of the TCorbaFactory class are listed here.

Public Properties

	Name	Description
R	InstanceName (see page 6440)	This is InstanceName, a member of class TCorbaFactory.
R	Instancing (see page 6441)	This is Instancing, a member of class TCorbaFactory.
R	InterfaceName (see page 6441)	This is InterfaceName, a member of class TCorbaFactory.
R	RepositoryID (see page 6441)	This is RepositoryID, a member of class TCorbaFactory.
R	ThreadModel (see page 6441)	This is ThreadModel, a member of class TCorbaFactory.

34.1.6.5.1 TCorbaFactory.InstanceName Property

Pascal

```
property InstanceName: string;
```

C++

```
__property AnsiString InstanceName;
```

Description

This is InstanceName, a member of class TCorbaFactory.

34.1.6.5.2 TCorbaFactory.Instancing Property

Pascal

```
property Instancing: TCorbaInstancing;
```

C++

```
__property TCorbaInstancing Instancing;
```

Description

This is Instancing, a member of class TCorbaFactory.

34.1.6.5.3 TCorbaFactory.InterfaceName Property

Pascal

```
property InterfaceName: string;
```

C++

```
__property AnsiString InterfaceName;
```

Description

This is InterfaceName, a member of class TCorbaFactory.

34.1.6.5.4 TCorbaFactory.RepositoryID Property

Pascal

```
property RepositoryID: string;
```

C++

```
__property AnsiString RepositoryID;
```

Description

This is RepositoryID, a member of class TCorbaFactory.

34.1.6.5.5 TCorbaFactory.ThreadModel Property

Pascal

```
property ThreadModel: TCorbaThreadModel;
```

C++

```
__property TCorbaThreadModel ThreadModel;
```

Description

This is ThreadModel, a member of class TCorbaFactory.

34.1.7 TCorbaFactoryManager Class

Class Hierarchy

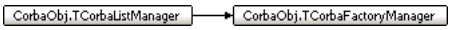

Pascal

```
TCorbaFactoryManager = class(TCorbaListManager);
```

C++

```
class TCorbaFactoryManager : public TCorbaListManager;
```

File

CorbaObj

Description

This is class CorbaObj.TCorbaFactoryManager.

34.1.7.1 TCorbaFactoryManager Members

The following tables list the members exposed by TCorbaFactoryManager.

Protected Methods

	Name	Description
	BeginRead (see page 6452)	This is BeginRead, a member of class TCorbaListManager.
	BeginWrite (see page 6452)	This is BeginWrite, a member of class TCorbaListManager.
	EndRead (see page 6453)	This is EndRead, a member of class TCorbaListManager.
	EndWrite (see page 6453)	This is EndWrite, a member of class TCorbaListManager.

Public Methods

	Name	Description
	AddFactory (see page 6443)	This is AddFactory, a member of class TCorbaFactoryManager.
	Create (see page 6451)	This is Create, a member of class TCorbaListManager.
V	Destroy (see page 6442)	This is Destroy, a member of class TCorbaFactoryManager.
	Find (see page 6443)	This is Find, a member of class TCorbaFactoryManager.
	RegisterFactories (see page 6443)	This is RegisterFactories, a member of class TCorbaFactoryManager.

34.1.7.2 TCorbaFactoryManager.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCorbaFactoryManager();
```

Description

This is Destroy, a member of class TCorbaFactoryManager.

34.1.7.3 TCorbaFactoryManager Methods

The methods of the TCorbaFactoryManager class are listed here.

Public Methods

	Name	Description
	AddFactory (see page 6443)	This is AddFactory, a member of class TCorbaFactoryManager.
	Find (see page 6443)	This is Find, a member of class TCorbaFactoryManager.
	RegisterFactories (see page 6443)	This is RegisterFactories, a member of class TCorbaFactoryManager.

34.1.7.3.1 TCorbaFactoryManager.AddFactory Method

Pascal

```
procedure AddFactory(Factory: TCorbaFactory);
```

C++

```
__fastcall AddFactory(TCorbaFactory Factory);
```

Description

This is AddFactory, a member of class TCorbaFactoryManager.

34.1.7.3.2 TCorbaFactoryManager.Find Method

Pascal

```
function Find(const RepositoryID: string; const InterfaceName: string; const InstanceName:
string): TCorbaFactory;
```

C++

```
__fastcall TCorbaFactory Find(const AnsiString RepositoryID, const AnsiString InterfaceName,
const AnsiString InstanceName);
```

Description

This is Find, a member of class TCorbaFactoryManager.

34.1.7.3.3 TCorbaFactoryManager.RegisterFactories Method

Pascal

```
procedure RegisterFactories;
```

C++

```
__fastcall RegisterFactories();
```

Description

This is RegisterFactories, a member of class TCorbaFactoryManager.

34.1.8 TCorbaImplementation Class

Class Hierarchy

Pascal

```
TCorbaImplementation = class(TObject, IUnknown);
```

C++

```
class TCorbaImplementation : public TObject, public IUnknown;
```

File

CorbaObj

Description

This is class CorbaObj.TCorbaImplementation.

34.1.8.1 TCorbaImplementation Members

The following tables list the members exposed by TCorbaImplementation.

Protected Fields

	Name	Description
	FController (see page 6445)	This is FController, a member of class TCorbaImplementation.
	FFactory (see page 6446)	This is FFactory, a member of class TCorbaImplementation.
	FRefCount (see page 6446)	This is FRefCount, a member of class TCorbaImplementation.

Protected Methods

	Name	Description
	_AddRef (see page 6446)	This is _AddRef, a member of class TCorbaImplementation.
	_Release (see page 6447)	This is _Release, a member of class TCorbaImplementation.
	GetIDsOfNames (see page 6447)	This is GetIDsOfNames, a member of class TCorbaImplementation.
	GetTypeInfo (see page 6447)	This is GetTypeInfo, a member of class TCorbaImplementation.
	GetTypeInfoCount (see page 6447)	This is GetTypeInfoCount, a member of class TCorbaImplementation.
	Invoke (see page 6448)	This is Invoke, a member of class TCorbaImplementation.
V	ObjAddRef (see page 6448)	This is ObjAddRef, a member of class TCorbaImplementation.
V	ObjQueryInterface (see page 6448)	This is ObjQueryInterface, a member of class TCorbaImplementation.
V	ObjRelease (see page 6448)	This is ObjRelease, a member of class TCorbaImplementation.
	QueryInterface (see page 6448)	This is QueryInterface, a member of class TCorbaImplementation.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 6445)	This is Create, a member of class TCorbaImplementation.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

34.1.8.2 TCorbaImplementation.Create Constructor

Pascal

```
constructor Create(Controller: IObject; AFactory: TCorbaFactory); virtual;
```

C++

```
virtual __fastcall TCorbaImplementation(IObject Controller, TCorbaFactory AFactory);
```

Description

This is Create, a member of class TCorbaImplementation.

34.1.8.3 TCorbaImplementation Fields

The fields of the TCorbaImplementation class are listed here.

Protected Fields

	Name	Description
	FController (see page 6445)	This is FController, a member of class TCorbaImplementation.
	FFactory (see page 6446)	This is FFactory, a member of class TCorbaImplementation.
	FRefCount (see page 6446)	This is FRefCount, a member of class TCorbaImplementation.

34.1.8.3.1 TCorbaImplementation.FController Field

Pascal

```
FController: Pointer;
```

C++

```
void * FController;
```

Description

This is FController, a member of class TCorbaImplementation.

34.1.8.3.2 TCorbaImplementation.FFactory Field

Pascal

```
FFactory: TCorbaFactory;
```

C++

```
TCorbaFactory FFactory;
```

Description

This is FFactory, a member of class TCorbaImplementation.

34.1.8.3.3 TCorbaImplementation.FRefCount Field

Pascal

```
FRefCount: Integer;
```

C++

```
int FRefCount;
```

Description

This is FRefCount, a member of class TCorbaImplementation.

34.1.8.4 TCorbaImplementation Methods

The methods of the TCorbaImplementation class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 6446)	This is _AddRef, a member of class TCorbaImplementation.
	_Release (see page 6447)	This is _Release, a member of class TCorbaImplementation.
	GetIDsOfNames (see page 6447)	This is GetIDsOfNames, a member of class TCorbaImplementation.
	GetTypeInfo (see page 6447)	This is GetTypeInfo, a member of class TCorbaImplementation.
	GetTypeInfoCount (see page 6447)	This is GetTypeInfoCount, a member of class TCorbaImplementation.
	Invoke (see page 6448)	This is Invoke, a member of class TCorbaImplementation.
V	ObjAddRef (see page 6448)	This is ObjAddRef, a member of class TCorbaImplementation.
V	ObjQueryInterface (see page 6448)	This is ObjQueryInterface, a member of class TCorbaImplementation.
V	ObjRelease (see page 6448)	This is ObjRelease, a member of class TCorbaImplementation.
	QueryInterface (see page 6448)	This is QueryInterface, a member of class TCorbaImplementation.

34.1.8.4.1 TCorbaImplementation._AddRef Method

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

This is _AddRef, a member of class TCorbaImplementation.

34.1.8.4.2 TCorbaImplementation._Release Method

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

This is _Release, a member of class TCorbaImplementation.

34.1.8.4.3 TCorbaImplementation.GetIDsOfNames Method

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int NameCount, int
LocaleID, void * DispIDs);
```

Description

This is GetIDsOfNames, a member of class TCorbaImplementation.

34.1.8.4.4 TCorbaImplementation.GetTypeInfo Method

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

This is GetTypeInfo, a member of class TCorbaImplementation.

34.1.8.4.5 TCorbaImplementation.GetTypeInfoCount Method

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

This is GetTypeInfoCount, a member of class TCorbaImplementation.

34.1.8.4.6 TCorbaImplementation.Invoke Method

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word Flags,
Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is Invoke, a member of class TCorbaImplementation.

34.1.8.4.7 TCorbaImplementation.ObjAddRef Method

Pascal

```
function ObjAddRef: Integer; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall int ObjAddRef();
```

Description

This is ObjAddRef, a member of class TCorbaImplementation.

34.1.8.4.8 TCorbaImplementation.ObjQueryInterface Method

Pascal

```
function ObjQueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT ObjQueryInterface(const TGUID IID,  Obj);
```

Description

This is ObjQueryInterface, a member of class TCorbaImplementation.

34.1.8.4.9 TCorbaImplementation.ObjRelease Method

Pascal

```
function ObjRelease: Integer; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall int ObjRelease();
```

Description

This is ObjRelease, a member of class TCorbaImplementation.

34.1.8.4.10 TCorbaImplementation.QueryInterface Method

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

This is QueryInterface, a member of class TCorbaImplementation.

34.1.9 TCorbaInterfaceIDManager Class

Class Hierarchy

Pascal

```
TCorbaInterfaceIDManager = class(TCorbaListManager);
```

C++

```
class TCorbaInterfaceIDManager : public TCorbaListManager;
```

File

CorbaObj

Description

This is class CorbaObj.TCorbaInterfaceIDManager.

34.1.9.1 TCorbaInterfaceIDManager Members

The following tables list the members exposed by TCorbaInterfaceIDManager.

Protected Methods

	Name	Description
	BeginRead (see page 6452)	This is BeginRead, a member of class TCorbaListManager.
	BeginWrite (see page 6452)	This is BeginWrite, a member of class TCorbaListManager.
	EndRead (see page 6453)	This is EndRead, a member of class TCorbaListManager.
	EndWrite (see page 6453)	This is EndWrite, a member of class TCorbaListManager.

Public Methods

	Name	Description
	Create (see page 6451)	This is Create, a member of class TCorbaListManager.
V	Destroy (see page 6452)	This is Destroy, a member of class TCorbaListManager.
	FindGUID (see page 6450)	This is FindGUID, a member of class TCorbaInterfaceIDManager.
	FindID (see page 6450)	This is FindID, a member of class TCorbaInterfaceIDManager.
	RegisterInterface (see page 6450)	This is RegisterInterface, a member of class TCorbaInterfaceIDManager.
	SearchGUID (see page 6450)	This is SearchGUID, a member of class TCorbaInterfaceIDManager.
	SearchID (see page 6451)	This is SearchID, a member of class TCorbaInterfaceIDManager.

34.1.9.2 TCorbaInterfaceIDManager Methods

The methods of the TCorbaInterfaceIDManager class are listed here.

Public Methods

	Name	Description
	FindGUID (see page 6450)	This is FindGUID, a member of class TCorbaInterfaceIDManager.
	FindID (see page 6450)	This is FindID, a member of class TCorbaInterfaceIDManager.
	RegisterInterface (see page 6450)	This is RegisterInterface, a member of class TCorbaInterfaceIDManager.
	SearchGUID (see page 6450)	This is SearchGUID, a member of class TCorbaInterfaceIDManager.
	SearchID (see page 6451)	This is SearchID, a member of class TCorbaInterfaceIDManager.

34.1.9.2.1 TCorbaInterfaceIDManager.FindGUID Method

Pascal

```
function FindGUID(const RepositoryID: string): TGUID;
```

C++

```
__fastcall TGUID FindGUID(const AnsiString RepositoryID);
```

Description

This is FindGUID, a member of class TCorbaInterfaceIDManager.

34.1.9.2.2 TCorbaInterfaceIDManager.FindID Method

Pascal

```
function FindID(const IID: TGUID): string;
```

C++

```
__fastcall AnsiString FindID(const TGUID IID);
```

Description

This is FindID, a member of class TCorbaInterfaceIDManager.

34.1.9.2.3 TCorbaInterfaceIDManager.RegisterInterface Method

Pascal

```
procedure RegisterInterface(const IID: TGUID; const RepositoryID: string);
```

C++

```
__fastcall RegisterInterface(const TGUID IID, const AnsiString RepositoryID);
```

Description

This is RegisterInterface, a member of class TCorbaInterfaceIDManager.

34.1.9.2.4 TCorbaInterfaceIDManager.SearchGUID Method

Pascal

```
function SearchGUID(const RepositoryID: string; out IID: TGUID): Boolean;
```

C++

```
__fastcall Boolean SearchGUID(const AnsiString RepositoryID, TGUID IID);
```

Description

This is SearchGUID, a member of class TCorbaInterfaceIDManager.

34.1.9.2.5 TCorbaInterfaceIDManager.SearchID Method

Pascal

```
function SearchID(const IID: TGUID; out RepositoryID: string): Boolean;
```

C++

```
__fastcall Boolean SearchID(const TGUID IID, AnsiString RepositoryID);
```

Description

This is SearchID, a member of class TCorbaInterfaceIDManager.

34.1.10 TCorbaListManager Class

Class Hierarchy

Pascal

```
TCorbaListManager = class;
```

C++

```
class TCorbaListManager;
```

File

CorbaObj

Description

This is class CorbaObj.TCorbaListManager.

34.1.10.1 TCorbaListManager Members

The following tables list the members exposed by TCorbaListManager.

Protected Methods

	Name	Description
	BeginRead (see page 6452)	This is BeginRead, a member of class TCorbaListManager.
	BeginWrite (see page 6452)	This is BeginWrite, a member of class TCorbaListManager.
	EndRead (see page 6453)	This is EndRead, a member of class TCorbaListManager.
	EndWrite (see page 6453)	This is EndWrite, a member of class TCorbaListManager.

Public Methods

	Name	Description
	Create (see page 6451)	This is Create, a member of class TCorbaListManager.
V	Destroy (see page 6452)	This is Destroy, a member of class TCorbaListManager.

34.1.10.2 TCorbaListManager.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TCorbaListManager();
```

Description

This is Create, a member of class TCorbaListManager.

34.1.10.3 TCorbaListManager.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCorbaListManager();
```

Description

This is Destroy, a member of class TCorbaListManager.

34.1.10.4 TCorbaListManager Methods

The methods of the TCorbaListManager class are listed here.

Protected Methods

	Name	Description
	BeginRead (see page 6452)	This is BeginRead, a member of class TCorbaListManager.
	BeginWrite (see page 6452)	This is BeginWrite, a member of class TCorbaListManager.
	EndRead (see page 6453)	This is EndRead, a member of class TCorbaListManager.
	EndWrite (see page 6453)	This is EndWrite, a member of class TCorbaListManager.

34.1.10.4.1 TCorbaListManager.BeginRead Method

Pascal

```
procedure BeginRead;
```

C++

```
__fastcall BeginRead();
```

Description

This is BeginRead, a member of class TCorbaListManager.

34.1.10.4.2 TCorbaListManager.BeginWrite Method

Pascal

```
procedure BeginWrite;
```

C++

```
__fastcall BeginWrite();
```

Description

This is BeginWrite, a member of class TCorbaListManager.

34.1.10.4.3 TCorbaListManager.EndRead Method

Pascal

```
procedure EndRead;
```

C++

```
__fastcall EndRead();
```

Description

This is EndRead, a member of class TCorbaListManager.

34.1.10.4.4 TCorbaListManager.EndWrite Method

Pascal

```
procedure EndWrite;
```

C++

```
__fastcall EndWrite();
```

Description

This is EndWrite, a member of class TCorbaListManager.

34.1.11 TCorbaObjectFactory Class

Class Hierarchy

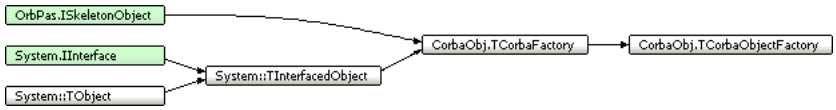

Pascal

```
TCorbaObjectFactory = class(TCorbaFactory);
```

C++

```
class TCorbaObjectFactory : public TCorbaFactory;
```

File

CorbaObj

Description

This is class CorbaObj.TCorbaObjectFactory.

34.1.11.1 TCorbaObjectFactory Members

The following tables list the members exposed by TCorbaObjectFactory.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
V	CreateInstance (see page 6439)	This is CreateInstance, a member of class TCorbaFactory.
V	CreateInterface (see page 6455)	This is CreateInterface, a member of class TCorbaObjectFactory.
	Execute (see page 6439)	This is Execute, a member of class TCorbaFactory.
	GetImplementation (see page 6439)	This is GetImplementation, a member of class TCorbaFactory.
	GetSkeleton (see page 6440)	This is GetSkeleton, a member of class TCorbaFactory.
	GetTypeInfo (see page 6440)	This is GetTypeInfo, a member of class TCorbaFactory.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterFactory (see page 6440)	This is RegisterFactory, a member of class TCorbaFactory.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6455)	This is Create, a member of class TCorbaObjectFactory.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 6438)	Disposes of an object instance.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 18207)	This is Execute, a member of class ISkeletonObject.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetImplementation (see page 18207)	This is GetImplementation, a member of class ISkeletonObject.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetSkeleton (see page 18207)	This is GetSkeleton, a member of class ISkeletonObject.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.

V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ImplementationClass (see page 6456)	This is ImplementationClass, a member of class TCorbaObjectFactory.
R	InstanceName (see page 6440)	This is InstanceName, a member of class TCorbaFactory.
R	Instancing (see page 6441)	This is Instancing, a member of class TCorbaFactory.
R	InterfaceName (see page 6441)	This is InterfaceName, a member of class TCorbaFactory.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	RepositoryID (see page 6441)	This is RepositoryID, a member of class TCorbaFactory.
R	ThreadModel (see page 6441)	This is ThreadModel, a member of class TCorbaFactory.

34.1.11.2 TCorbaObjectFactory.Create Constructor

Pascal

```
constructor Create(const InterfaceName: string; const InstanceName: string; const
RepositoryId: string; const ImplGUID: TGUID; ImplementationClass: TCorbaImplementationClass;
Instancing: TCorbaInstancing = iMultiInstance; ThreadModel: TCorbaThreadModel =
tmSingleThread);
```

C++

```
__fastcall TCorbaObjectFactory(const AnsiString InterfaceName, const AnsiString InstanceName,
const AnsiString RepositoryId, const TGUID ImplGUID, TCorbaImplementationClass
ImplementationClass, TCorbaInstancing Instancing = iMultiInstance, TCorbaThreadModel
ThreadModel = tmSingleThread);
```

Description

This is Create, a member of class TCorbaObjectFactory.

34.1.11.3 TCorbaObjectFactory Methods

The methods of the TCorbaObjectFactory class are listed here.

Protected Methods

	Name	Description
V	CreateInterface (see page 6455)	This is CreateInterface, a member of class TCorbaObjectFactory.

34.1.11.3.1 TCorbaObjectFactory.CreateInterface Method

Pascal

```
function CreateInterface(const InstanceName: string): IObject; override;
```

C++

```
virtual __fastcall IObject CreateInterface(const AnsiString InstanceName);
```

Description

This is CreateInterface, a member of class TCorbaObjectFactory.

34.1.11.4 TCorbaObjectFactory Properties

The properties of the TCorbaObjectFactory class are listed here.

Public Properties

	Name	Description
R	ImplementationClass (see page 6456)	This is ImplementationClass, a member of class TCorbaObjectFactory.

34.1.11.4.1 TCorbaObjectFactory.ImplementationClass Property

Pascal

```
property ImplementationClass: TCorbaImplementationClass;
```

C++

```
__property TCorbaImplementationClass ImplementationClass;
```

Description

This is ImplementationClass, a member of class TCorbaObjectFactory.

34.1.12 TCorbaSkeleton Class

Class Hierarchy

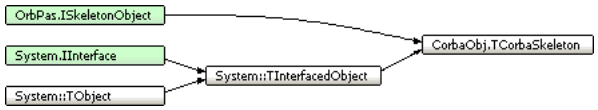

Pascal

```
TCorbaSkeleton = class(TInterfacedObject, ISkeletonObject);
```

C++

```
class TCorbaSkeleton : public TInterfacedObject, public ISkeletonObject;
```

File

CorbaObj

Description

This is class CorbaObj.TCorbaSkeleton.

34.1.12.1 TCorbaSkeleton Members

The following tables list the members exposed by TCorbaSkeleton.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Fields

	Name	Description
	FSkeleton (see page 6459)	This is FSkeleton, a member of class TCorbaSkeleton.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	Execute (see page 6459)	This is Execute, a member of class TCorbaSkeleton.
V	GetImplementation (see page 6459)	This is GetImplementation, a member of class TCorbaSkeleton.
	GetSkeleton (see page 6460)	This is GetSkeleton, a member of class TCorbaSkeleton.
	InitSkeleton (see page 6460)	This is InitSkeleton, a member of class TCorbaSkeleton.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 6458)	This is Create, a member of class TCorbaSkeleton.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 6458)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

34.1.12.2 TCorbaSkeleton.Create Constructor

Pascal

```
constructor Create(const InstanceName: string; const Impl: IObject); virtual;
```

C++

```
virtual __fastcall TCorbaSkeleton(const AnsiString InstanceName, const IObject Impl);
```

Description

This is Create, a member of class TCorbaSkeleton.

34.1.12.3 TCorbaSkeleton.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCorbaSkeleton();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

34.1.12.4 TCorbaSkeleton Fields

The fields of the TCorbaSkeleton class are listed here.

Protected Fields

	Name	Description
	FSkeleton (see page 6459)	This is FSkeleton, a member of class TCorbaSkeleton.

34.1.12.4.1 TCorbaSkeleton.FSkeleton Field

Pascal

```
FSkeleton: ISkeleton;
```

C++

```
ISkeleton FSkeleton;
```

Description

This is FSkeleton, a member of class TCorbaSkeleton.

34.1.12.5 TCorbaSkeleton Methods

The methods of the TCorbaSkeleton class are listed here.

Protected Methods

	Name	Description
	Execute (see page 6459)	This is Execute, a member of class TCorbaSkeleton.
V	GetImplementation (see page 6459)	This is GetImplementation, a member of class TCorbaSkeleton.
	GetSkeleton (see page 6460)	This is GetSkeleton, a member of class TCorbaSkeleton.
	InitSkeleton (see page 6460)	This is InitSkeleton, a member of class TCorbaSkeleton.

34.1.12.5.1 TCorbaSkeleton.Execute Method

Pascal

```
function Execute(Operation: PChar; const Strm: IMarshalInBuffer; Cookie: Pointer):
CorbaBoolean; stdcall;
```

C++

```
__fastcall __stdcall CorbaBoolean Execute(const char * Operation, const IMarshalInBuffer Strm,
void * Cookie);
```

Description

This is Execute, a member of class TCorbaSkeleton.

34.1.12.5.2 TCorbaSkeleton.GetImplementation Method

Pascal

```
procedure GetImplementation(out Impl: IObject); virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall GetImplementation(IObject Impl);
```

Description

This is GetImplementation, a member of class TCorbaSkeleton.

34.1.12.5.3 TCorbaSkeleton.GetSkeleton Method

Pascal

```
procedure GetSkeleton(out Skeleton: ISkeleton); stdcall;
```

C++

```
__fastcall __stdcall GetSkeleton(ISkeleton Skeleton);
```

Description

This is GetSkeleton, a member of class TCorbaSkeleton.

34.1.12.5.4 TCorbaSkeleton.InitSkeleton Method

Pascal

```
procedure InitSkeleton(const InterfaceName: string; const InstanceName: string; const
RepositoryID: string; ThreadModel: TCorbaThreadModel; ClientRefCount: Boolean);
```

C++

```
__fastcall InitSkeleton(const AnsiString InterfaceName, const AnsiString InstanceName, const
AnsiString RepositoryID, TCorbaThreadModel ThreadModel, Boolean ClientRefCount);
```

Description

This is InitSkeleton, a member of class TCorbaSkeleton.

34.1.13 TCorbaSkeletonManager Class

Class Hierarchy

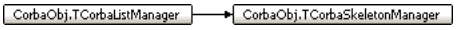

Pascal

```
TCorbaSkeletonManager = class(TCorbaListManager);
```

C++

```
class TCorbaSkeletonManager : public TCorbaListManager;
```

File

CorbaObj

Description

This is class CorbaObj.TCorbaSkeletonManager.

34.1.13.1 TCorbaSkeletonManager Members

The following tables list the members exposed by TCorbaSkeletonManager.

Protected Methods

	Name	Description
	BeginRead (see page 6452)	This is BeginRead, a member of class TCorbaListManager.
	BeginWrite (see page 6452)	This is BeginWrite, a member of class TCorbaListManager.

	EndRead (see page 6453)	This is EndRead, a member of class TCorbaListManager.
	EndWrite (see page 6453)	This is EndWrite, a member of class TCorbaListManager.

Public Methods

	Name	Description
	Create (see page 6451)	This is Create, a member of class TCorbaListManager.
	CreateSkeleton (see page 6461)	This is CreateSkeleton, a member of class TCorbaSkeletonManager.
	Destroy (see page 6452)	This is Destroy, a member of class TCorbaListManager.
	RegisterSkeleton (see page 6461)	This is RegisterSkeleton, a member of class TCorbaSkeletonManager.

34.1.13.2 TCorbaSkeletonManager Methods

The methods of the TCorbaSkeletonManager class are listed here.

Public Methods

	Name	Description
	CreateSkeleton (see page 6461)	This is CreateSkeleton, a member of class TCorbaSkeletonManager.
	RegisterSkeleton (see page 6461)	This is RegisterSkeleton, a member of class TCorbaSkeletonManager.

34.1.13.2.1 TCorbaSkeletonManager.CreateSkeleton Method

Pascal

```
function CreateSkeleton(IID: TGUID; const InstanceName: string; const Impl: IObject):
ISkeletonObject;
```

C++

```
__fastcall ISkeletonObject CreateSkeleton(TGUID IID, const AnsiString InstanceName, const
IObject Impl);
```

Description

This is CreateSkeleton, a member of class TCorbaSkeletonManager.

34.1.13.2.2 TCorbaSkeletonManager.RegisterSkeleton Method

Pascal

```
procedure RegisterSkeleton(IID: TGUID; Skeleton: TCorbaSkeletonClass);
```

C++

```
__fastcall RegisterSkeleton(TGUID IID, TCorbaSkeletonClass Skeleton);
```

Description

This is RegisterSkeleton, a member of class TCorbaSkeletonManager.

34.1.14 TCorbaStub Class

Class Hierarchy

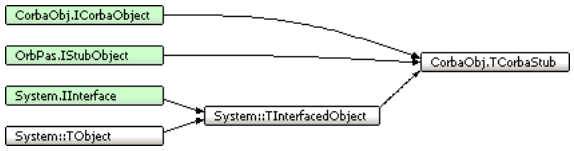

Pascal

```
TCorbaStub = class(TInterfacedObject, IStubObject, ICorbaObject);
```

C++

```
class TCorbaStub : public TInterfacedObject, public IStubObject, public ICorbaObject;
```

File

CorbaObj

Description

This is class CorbaObj.TCorbaStub.

34.1.14.1 TCorbaStub Members

The following tables list the members exposed by TCorbaStub.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Fields

	Name	Description
	FStub (see page 6464)	This is FStub, a member of class TCorbaStub.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetStub (see page 6465)	This is GetStub, a member of class TCorbaStub.
	Hash (see page 6465)	This is Hash, a member of class TCorbaStub.
	IsA (see page 6465)	This is IsA, a member of class TCorbaStub.
	NonExistent (see page 6465)	This is NonExistent, a member of class TCorbaStub.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetPrincipal (see page 6466)	This is SetPrincipal, a member of class TCorbaStub.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.

	AfterConstruction (see page 22279)	Responds after the last constructor executes.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6463)	This is Create, a member of class TCorbaStub.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 6464)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

34.1.14.2 TCorbaStub.Create Constructor

Pascal

```
constructor Create(const Stub: IStub); virtual;
```

C++

```
virtual __fastcall TCorbaStub(const IStub Stub);
```

Description

This is Create, a member of class TCorbaStub.

34.1.14.3 TCorbaStub.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCorbaStub();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

34.1.14.4 TCorbaStub Fields

The fields of the TCorbaStub class are listed here.

Protected Fields

	Name	Description
	FStub (see page 6464)	This is FStub, a member of class TCorbaStub.

34.1.14.4.1 TCorbaStub.FStub Field

Pascal

```
FStub: IStub;
```

C++

```
IStub FStub;
```

Description

This is FStub, a member of class TCorbaStub.

34.1.14.5 TCorbaStub Methods

The methods of the TCorbaStub class are listed here.

Protected Methods

	Name	Description
	GetStub (see page 6465)	This is GetStub, a member of class TCorbaStub.
	Hash (see page 6465)	This is Hash, a member of class TCorbaStub.
	IsA (see page 6465)	This is IsA, a member of class TCorbaStub.
	NonExistent (see page 6465)	This is NonExistent, a member of class TCorbaStub.
	SetPrincipal (see page 6466)	This is SetPrincipal, a member of class TCorbaStub.

34.1.14.5.1 TCorbaStub.GetStub Method

Pascal

```
procedure GetStub(out Stub: IStub); stdcall;
```

C++

```
__fastcall __stdcall GetStub(IStub Stub);
```

Description

This is GetStub, a member of class TCorbaStub.

34.1.14.5.2 TCorbaStub.Hash Method

Pascal

```
function Hash(Maximum: Integer): Integer;
```

C++

```
__fastcall int Hash(int Maximum);
```

Description

This is Hash, a member of class TCorbaStub.

34.1.14.5.3 TCorbaStub.IsA Method

Pascal

```
function IsA(const LogicalTypeId: string): Boolean;
```

C++

```
__fastcall Boolean IsA(const AnsiString LogicalTypeId);
```

Description

This is IsA, a member of class TCorbaStub.

34.1.14.5.4 TCorbaStub.NonExistent Method

Pascal

```
function NonExistent: Boolean;
```

C++

```
__fastcall Boolean NonExistent();
```

Description

This is NonExistent, a member of class TCorbaStub.

34.1.14.5.5 TCorbaStub.SetPrincipal Method

Pascal

```
procedure SetPrincipal(const Prinicpal: TCorbaPrincipal);
```

C++

```
__fastcall SetPrincipal(const TCorbaPrincipal Prinicpal);
```

Description

This is SetPrincipal, a member of class TCorbaStub.

34.1.15 TCorbaStubManager Class

Class Hierarchy

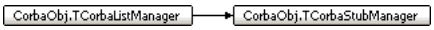

Pascal

```
TCorbaStubManager = class(TCorbaListManager);
```

C++

```
class TCorbaStubManager : public TCorbaListManager;
```

File

CorbaObj

Description

This is class CorbaObj.TCorbaStubManager.

34.1.15.1 TCorbaStubManager Members

The following tables list the members exposed by TCorbaStubManager.

Protected Methods

	Name	Description
	BeginRead (see page 6452)	This is BeginRead, a member of class TCorbaListManager.
	BeginWrite (see page 6452)	This is BeginWrite, a member of class TCorbaListManager.
	EndRead (see page 6453)	This is EndRead, a member of class TCorbaListManager.
	EndWrite (see page 6453)	This is EndWrite, a member of class TCorbaListManager.

Public Methods

	Name	Description
	Create (see page 6451)	This is Create, a member of class TCorbaListManager.
	CreateStub (see page 6467)	This is CreateStub, a member of class TCorbaStubManager.

	Destroy (see page 6452)	This is Destroy, a member of class TCorbaListManager.
	RegisterStub (see page 6467)	This is RegisterStub, a member of class TCorbaStubManager.

34.1.15.2 TCorbaStubManager Methods

The methods of the TCorbaStubManager class are listed here.

Public Methods

	Name	Description
	CreateStub (see page 6467)	This is CreateStub, a member of class TCorbaStubManager.
	RegisterStub (see page 6467)	This is RegisterStub, a member of class TCorbaStubManager.

34.1.15.2.1 TCorbaStubManager.CreateStub Method

Pascal

```
function CreateStub(IID: TGUID; const Stub: IStub): IObject;
```

C++

```
__fastcall IObject CreateStub(TGUID IID, const IStub Stub);
```

Description

This is CreateStub, a member of class TCorbaStubManager.

34.1.15.2.2 TCorbaStubManager.RegisterStub Method

Pascal

```
procedure RegisterStub(IID: TGUID; Stub: TCorbaStubClass);
```

C++

```
__fastcall RegisterStub(TGUID IID, TCorbaStubClass Stub);
```

Description

This is RegisterStub, a member of class TCorbaStubManager.

34.1.16 TORB Class

Class Hierarchy

CorbaObj.TORB

Pascal

```
TORB = class;
```

C++

```
class TORB;
```

File

CorbaObj

Description

This is class CorbaObj.TORB.

34.1.16.1 TORB Members

The following tables list the members exposed by TORB.

Public Methods

	Name	Description
	Bind (see page 6468)	This is the overview for the Bind method overload.
	FindTypeCode (see page 6469)	This is FindTypeCode, a member of class TORB.
	Initialize (see page 6469)	This is the overview for the Initialize method overload.
	MakeAlias (see page 6470)	This is MakeAlias, a member of class TORB.
	MakeAliasTypeCode (see page 6470)	This is MakeAliasTypeCode, a member of class TORB.
	MakeArray (see page 6471)	This is the overview for the MakeArray method overload.
	MakeObjectRefTypeCode (see page 6471)	This is MakeObjectRefTypeCode, a member of class TORB.
	MakeSequence (see page 6471)	This is the overview for the MakeSequence method overload.
	MakeSequenceTypeCode (see page 6472)	This is MakeSequenceTypeCode, a member of class TORB.
	MakeStructure (see page 6472)	This is MakeStructure, a member of class TORB.
	MakeStructureTypeCode (see page 6473)	This is MakeStructureTypeCode, a member of class TORB.
	MakeTypeCode (see page 6473)	This is MakeTypeCode, a member of class TORB.
	ObjectToString (see page 6473)	This is ObjectToString, a member of class TORB.
	Shutdown (see page 6473)	This is Shutdown, a member of class TORB.
	StringToObject (see page 6473)	This is StringToObject, a member of class TORB.

34.1.16.2 TORB Methods

The methods of the TORB class are listed here.

Public Methods

	Name	Description
	Bind (see page 6468)	This is the overview for the Bind method overload.
	FindTypeCode (see page 6469)	This is FindTypeCode, a member of class TORB.
	Initialize (see page 6469)	This is the overview for the Initialize method overload.
	MakeAlias (see page 6470)	This is MakeAlias, a member of class TORB.
	MakeAliasTypeCode (see page 6470)	This is MakeAliasTypeCode, a member of class TORB.
	MakeArray (see page 6471)	This is the overview for the MakeArray method overload.
	MakeObjectRefTypeCode (see page 6471)	This is MakeObjectRefTypeCode, a member of class TORB.
	MakeSequence (see page 6471)	This is the overview for the MakeSequence method overload.
	MakeSequenceTypeCode (see page 6472)	This is MakeSequenceTypeCode, a member of class TORB.
	MakeStructure (see page 6472)	This is MakeStructure, a member of class TORB.
	MakeStructureTypeCode (see page 6473)	This is MakeStructureTypeCode, a member of class TORB.
	MakeTypeCode (see page 6473)	This is MakeTypeCode, a member of class TORB.
	ObjectToString (see page 6473)	This is ObjectToString, a member of class TORB.
	Shutdown (see page 6473)	This is Shutdown, a member of class TORB.
	StringToObject (see page 6473)	This is StringToObject, a member of class TORB.

34.1.16.2.1 Bind Method

This is the overview for the Bind method overload.

Overload List

	Name	Description
	TORB.Bind (string, string, string) (see page 6469)	This is Bind, a member of class TORB.
	TORB.Bind (TGUID, string, string) (see page 6469)	This is Bind, a member of class TORB.

34.1.16.2.1.1 TORB.Bind Method (TGUID, string, string)

Pascal

```
function Bind(const InterfaceID: TGUID; const ObjectName: string = ''; const HostName: string
= ''): IObject; overload;
```

C++

```
__fastcall IObject Bind(const TGUID InterfaceID, const AnsiString ObjectName = '', const
AnsiString HostName = '');
```

Description

This is Bind, a member of class TORB.

34.1.16.2.1.2 TORB.Bind Method (string, string, string)

Pascal

```
function Bind(const RepositoryID: string; const ObjectName: string = ''; const HostName:
string = ''): IObject; overload;
```

C++

```
__fastcall IObject Bind(const AnsiString RepositoryID, const AnsiString ObjectName = '', const
AnsiString HostName = '');
```

Description

This is Bind, a member of class TORB.

34.1.16.2.2 TORB.FindTypeCode Method

Pascal

```
function FindTypeCode(const RepositoryID: string): ITypeCode;
```

C++

```
__fastcall ITypeCode FindTypeCode(const AnsiString RepositoryID);
```

Description

This is FindTypeCode, a member of class TORB.

34.1.16.2.3 Initialize Method

This is the overview for the Initialize method overload.

Overload List

	Name	Description
	TORB.Initialize () (see page 6470)	This is Initialize, a member of class TORB.

	TORB.Initialize (TCommandLine) (see page 6470)	This is Initialize, a member of class TORB.

34.1.16.2.3.1 TORB.Initialize Method ()

Pascal

```
class procedure Initialize; overload;
```

C++

```
__fastcall Initialize();
```

Description

This is Initialize, a member of class TORB.

34.1.16.2.3.2 TORB.Initialize Method (TCommandLine)

Pascal

```
class procedure Initialize(const CommandLine: TCommandLine); overload;
```

C++

```
__fastcall Initialize(const TCommandLine CommandLine);
```

Description

This is Initialize, a member of class TORB.

34.1.16.2.4 TORB.MakeAlias Method

Pascal

```
function MakeAlias(const RepositoryID: string; const TypeName: string; Value: TAny; Test:
TAny): TAny;
```

C++

```
__fastcall TAny MakeAlias(const AnsiString RepositoryID, const AnsiString TypeName, TAny
Value, TAny Test);
```

Description

This is MakeAlias, a member of class TORB.

34.1.16.2.5 TORB.MakeAliasTypeCode Method

Pascal

```
function MakeAliasTypeCode(const RepositoryID: string; const Name: string; const TC:
ITypeCode): ITypeCode;
```

C++

```
__fastcall ITypeCode MakeAliasTypeCode(const AnsiString RepositoryID, const AnsiString Name,
const ITypeCode TC);
```

Description

This is MakeAliasTypeCode, a member of class TORB.

34.1.16.2.6 MakeArray Method

This is the overview for the MakeArray method overload.

Overload List

	Name	Description
	TORB.MakeArray (ITypeCode, array of TAny) (see page 6471)	This is MakeArray, a member of class TORB.
	TORB.MakeArray (TCKind, array of TAny) (see page 6471)	This is MakeArray, a member of class TORB.

34.1.16.2.6.1 TORB.MakeArray Method (ITypeCode, array of TAny)

Pascal

```
function MakeArray(TypeCode: ITypeCode; const Elements: array of TAny): TAny; overload;
```

C++

```
__fastcall TAny MakeArray(ITypeCode TypeCode, const array of TAny Elements);
```

Description

This is MakeArray, a member of class TORB.

34.1.16.2.6.2 TORB.MakeArray Method (TCKind, array of TAny)

Pascal

```
function MakeArray(Kind: TCKind; const Elements: array of TAny): TAny; overload;
```

C++

```
__fastcall TAny MakeArray(TCKind Kind, const array of TAny Elements);
```

Description

This is MakeArray, a member of class TORB.

34.1.16.2.7 TORB.MakeObjectRefTypeCode Method

Pascal

```
function MakeObjectRefTypeCode(const RepositoryID: string; const Name: string): ITypeCode;
```

C++

```
__fastcall ITypeCode MakeObjectRefTypeCode(const AnsiString RepositoryID, const AnsiString
Name);
```

Description

This is MakeObjectRefTypeCode, a member of class TORB.

34.1.16.2.8 MakeSequence Method

This is the overview for the MakeSequence method overload.

Overload List

	Name	Description
	TORB.MakeSequence (ITypeCode, array of TAny) (see page 6472)	This is MakeSequence, a member of class TORB.
	TORB.MakeSequence (TCKind, array of TAny) (see page 6472)	This is MakeSequence, a member of class TORB.

34.1.16.2.8.1 TORB.MakeSequence Method (ITypeCode, array of TAny)

Pascal

```
function MakeSequence(TypeCode: ITypeCode; const Elements: array of TAny): TAny; overload;
```

C++

```
__fastcall TAny MakeSequence(ITypeCode TypeCode, const array of TAny Elements);
```

Description

This is MakeSequence, a member of class TORB.

34.1.16.2.8.2 TORB.MakeSequence Method (TCKind, array of TAny)

Pascal

```
function MakeSequence(Kind: TCKind; const Elements: array of TAny): TAny; overload;
```

C++

```
__fastcall TAny MakeSequence(TCKind Kind, const array of TAny Elements);
```

Description

This is MakeSequence, a member of class TORB.

34.1.16.2.9 TORB.MakeSequenceTypeCode Method

Pascal

```
function MakeSequenceTypeCode(Bound: CorbaULong; const TC: ITypeCode): ITypeCode;
```

C++

```
__fastcall ITypeCode MakeSequenceTypeCode(CorbaULong Bound, const ITypeCode TC);
```

Description

This is MakeSequenceTypeCode, a member of class TORB.

34.1.16.2.10 TORB.MakeStructure Method

Pascal

```
function MakeStructure(TypeCode: ITypeCode; const Elements: array of TAny): TAny;
```

C++

```
__fastcall TAny MakeStructure(ITypeCode TypeCode, const array of TAny Elements);
```

Description

This is MakeStructure, a member of class TORB.

34.1.16.2.11 TORB.MakeStructureTypeCode Method

Pascal

```
function MakeStructureTypeCode(const RepositoryID: string; const Name: string; Members:
TStructMembers): ITypeCode;
```

C++

```
__fastcall ITypeCode MakeStructureTypeCode(const AnsiString RepositoryID, const AnsiString
Name, TStructMembers Members);
```

Description

This is MakeStructureTypeCode, a member of class TORB.

34.1.16.2.12 TORB.MakeTypeCode Method

Pascal

```
function MakeTypeCode(Kind: TCKind): ITypeCode;
```

C++

```
__fastcall ITypeCode MakeTypeCode(TCKind Kind);
```

Description

This is MakeTypeCode, a member of class TORB.

34.1.16.2.13 TORB.ObjectToString Method

Pascal

```
function ObjectToString(const Obj: IObject): string;
```

C++

```
__fastcall AnsiString ObjectToString(const IObject Obj);
```

Description

This is ObjectToString, a member of class TORB.

34.1.16.2.14 TORB.Shutdown Method

Pascal

```
procedure Shutdown;
```

C++

```
__fastcall Shutdown();
```

Description

This is Shutdown, a member of class TORB.

34.1.16.2.15 TORB.StringToObject Method

Pascal

```
function StringToObject(const ObjectString: string): IObject;
```

C++

```
__fastcall IObject StringToObject(const AnsiString ObjectString);
```

Description

This is StringToObject, a member of class TORB.

34.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	ICorbaObject (see page 6475)	This is class CorbaObj.ICorbaObject.

34.2.1 ICorbaObject Interface

Class Hierarchy

CorbaObj.ICorbaObject

Pascal

```
ICorbaObject = interface;
```

C++

```
__interface ICorbaObject;
```

File

CorbaObj

Description

This is class CorbaObj.ICorbaObject.

34.2.1.1 ICorbaObject Members

The following tables list the members exposed by ICorbaObject.

Public Methods

	Name	Description
	Hash (see page 6476)	This is Hash, a member of class ICorbaObject.
	IsA (see page 6476)	This is IsA, a member of class ICorbaObject.
	NonExistent (see page 6476)	This is NonExistent, a member of class ICorbaObject.
	SetPrincipal (see page 6476)	This is SetPrincipal, a member of class ICorbaObject.

34.2.1.2 ICorbaObject Methods

The methods of the ICorbaObject class are listed here.

Public Methods

	Name	Description
	Hash (see page 6476)	This is Hash, a member of class ICorbaObject.
	IsA (see page 6476)	This is IsA, a member of class ICorbaObject.
	NonExistent (see page 6476)	This is NonExistent, a member of class ICorbaObject.

	SetPrincipal (see page 6476)	This is SetPrincipal, a member of class ICorbaObject.

34.2.1.2.1 ICorbaObject.Hash Method

Pascal

```
function Hash(Maximum: Integer): Integer;
```

C++

```
__fastcall int Hash(int Maximum);
```

Description

This is Hash, a member of class ICorbaObject.

34.2.1.2.2 ICorbaObject.IsA Method

Pascal

```
function IsA(const LogicalTypeId: string): Boolean;
```

C++

```
__fastcall Boolean IsA(const AnsiString LogicalTypeId);
```

Description

This is IsA, a member of class ICorbaObject.

34.2.1.2.3 ICorbaObject.NonExistent Method

Pascal

```
function NonExistent: Boolean;
```

C++

```
__fastcall Boolean NonExistent();
```

Description

This is NonExistent, a member of class ICorbaObject.

34.2.1.2.4 ICorbaObject.SetPrincipal Method

Pascal

```
procedure SetPrincipal(const Prinicpal: TCorbaPrincipal);
```

C++

```
__fastcall SetPrincipal(const TCorbaPrincipal Prinicpal);
```

Description

This is SetPrincipal, a member of class ICorbaObject.

34.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AnyToObject (see page 6477)	This is function CorbaObj.AnyToObject.
	BOA (see page 6477)	This is function CorbaObj.BOA.
	CorbaBind (see page 6478)	This is function CorbaObj.CorbaBind.
	CorbaBind (see page 6478)	This is function CorbaObj.CorbaBind.
	CorbaFactoryCreateStub (see page 6478)	This is function CorbaObj.CorbaFactoryCreateStub.
	CorbaInitialize (see page 6479)	This is function CorbaObj.CorbaInitialize.
	MakePrincipal (see page 6479)	This is function CorbaObj.MakePrincipal.
	MarshalAny (see page 6479)	This is function CorbaObj.MarshalAny.
	MarshalObject (see page 6480)	This is function CorbaObj.MarshalObject.
	MarshalWordBool (see page 6480)	This is function CorbaObj.MarshalWordBool.
	ORB (see page 6480)	This is function CorbaObj.ORB.
	SequenceToVariantArray (see page 6481)	This is function CorbaObj.SequenceToVariantArray.
	UnmarshalAny (see page 6481)	This is function CorbaObj.UnmarshalAny.
	UnmarshalObject (see page 6481)	This is function CorbaObj.UnmarshalObject.
	UnmarshalText (see page 6482)	This is function CorbaObj.UnmarshalText.
	UnmarshalWideText (see page 6482)	This is function CorbaObj.UnmarshalWideText.
	UnmarshalWordBool (see page 6482)	This is function CorbaObj.UnmarshalWordBool.
	VariantArrayToSequence (see page 6483)	This is function CorbaObj.VariantArrayToSequence.

34.3.1 CorbaObj.AnyToObject Function

Pascal

```
function AnyToObject(Any: TAny; IID: TGUID): IObject;
```

C++

```
IObject AnyToObject(TAny Any, TGUID IID);
```

File

CorbaObj

Description

This is function CorbaObj.AnyToObject.

34.3.2 CorbaObj.BOA Function

Pascal

```
function BOA: TBOA;
```

C++

```
TBOA BOA();
```

File

CorbaObj

Description

This is function CorbaObj.BOA.

34.3.3 CorbaObj.CorbaBind Function

Pascal

```
function CorbaBind(const InterfaceID: TGUID; const ObjectName: string = ''; const HostName:
string = ''): IObject; overload;
```

C++

```
IObject CorbaBind(const TGUID InterfaceID, const AnsiString ObjectName = '', const AnsiString
HostName = '');
```

File

CorbaObj

Description

This is function CorbaObj.CorbaBind.

34.3.4 CorbaObj.CorbaBind Function

Pascal

```
function CorbaBind(const RepositoryID: string; const ObjectName: string = ''; const HostName:
string = ''): IObject; overload;
```

C++

```
IObject CorbaBind(const AnsiString RepositoryID, const AnsiString ObjectName = '', const
AnsiString HostName = '');
```

File

CorbaObj

Description

This is function CorbaObj.CorbaBind.

34.3.5 CorbaObj.CorbaFactoryCreateStub Function

Pascal

```
function CorbaFactoryCreateStub(const RepId: string; const FactoryId: string; const
InstanceName: string; const HostName: string; IID: TGUID): IObject;
```

C++

```
IObject CorbaFactoryCreateStub(const AnsiString RepId, const AnsiString FactoryId, const
AnsiString InstanceName, const AnsiString HostName, TGUID IID);
```

File

CorbaObj

Description

This is function CorbaObj.CorbaFactoryCreateStub.

34.3.6 CorbaObj.CorbaInitialize Function

Pascal

```
procedure CorbaInitialize;
```

C++

```
CorbaInitialize();
```

File

CorbaObj

Description

This is function CorbaObj.CorbaInitialize.

34.3.7 CorbaObj.MakePrincipal Function

Pascal

```
function MakePrincipal(const Bytes: array of Byte): TCorbaPrincipal;
```

C++

```
TCorbaPrincipal MakePrincipal(const array of Byte Bytes);
```

File

CorbaObj

Description

This is function CorbaObj.MakePrincipal.

34.3.8 CorbaObj.MarshalAny Function

Pascal

```
procedure MarshalAny(const OutBuf: IMarshalOutBuffer; const OV: Variant);
```

C++

```
MarshalAny(const IMarshalOutBuffer OutBuf, const Variant OV);
```

File

CorbaObj

Description

This is function CorbaObj.MarshalAny.

34.3.9 CorbaObj.MarshalObject Function

Pascal

```
procedure MarshalObject(const OutBuf: IMarshalOutBuffer; IID: TGUID; const Intf: IObject);
```

C++

```
MarshalObject(const IMarshalOutBuffer OutBuf, TGUID IID, const IObject Intf);
```

File

CorbaObj

Description

This is function CorbaObj.MarshalObject.

34.3.10 CorbaObj.MarshalWordBool Function

Pascal

```
procedure MarshalWordBool(const OutBuf: IMarshalOutBuffer; Value: WordBool);
```

C++

```
MarshalWordBool(const IMarshalOutBuffer OutBuf, WordBool Value);
```

File

CorbaObj

Description

This is function CorbaObj.MarshalWordBool.

34.3.11 CorbaObj.ORB Function

Pascal

```
function ORB: TORB;
```

C++

```
TORB ORB();
```

File

CorbaObj

Description

This is function CorbaObj.ORB.

34.3.12 CorbaObj.SequenceToVariantArray Function

Pascal

```
function SequenceToVariantArray(Sequence: TAny): Variant;
```

C++

```
Variant SequenceToVariantArray(TAny Sequence);
```

File

CorbaObj

Description

This is function CorbaObj.SequenceToVariantArray.

34.3.13 CorbaObj.UnmarshalAny Function

Pascal

```
function UnmarshalAny(const InBuf: IMarshalInBuffer): Variant;
```

C++

```
Variant UnmarshalAny(const IMarshalInBuffer InBuf);
```

File

CorbaObj

Description

This is function CorbaObj.UnmarshalAny.

34.3.14 CorbaObj.UnmarshalObject Function

Pascal

```
function UnmarshalObject(const InBuf: IMarshalInBuffer; IID: TGUID): IObject;
```

C++

```
IObject UnmarshalObject(const IMarshalInBuffer InBuf, TGUID IID);
```

File

CorbaObj

Description

This is function CorbaObj.UnmarshalObject.

34.3.15 CorbaObj.UnmarshalText Function

Pascal

```
function UnmarshalText(const InBuf: IMarshalInBuffer): string;
```

C++

```
AnsiString UnmarshalText(const IMarshalInBuffer InBuf);
```

File

CorbaObj

Description

This is function CorbaObj.UnmarshalText.

34.3.16 CorbaObj.UnmarshalWideText Function

Pascal

```
function UnmarshalWideText(const InBuf: IMarshalInBuffer): WideString;
```

C++

```
BSTR UnmarshalWideText(const IMarshalInBuffer InBuf);
```

File

CorbaObj

Description

This is function CorbaObj.UnmarshalWideText.

34.3.17 CorbaObj.UnmarshalWordBool Function

Pascal

```
function UnmarshalWordBool(const InBuf: IMarshalInBuffer): WordBool;
```

C++

```
WordBool UnmarshalWordBool(const IMarshalInBuffer InBuf);
```

File

CorbaObj

Description

This is function CorbaObj.UnmarshalWordBool.

34.3.18 CorbaObj.VariantArrayToSequence Function

Pascal

```
function VariantArrayToSequence(TypeCode: ITypeCode; const VariantArray: Variant): TAny;
```

C++

```
TAny VariantArrayToSequence(ITypeCode TypeCode, const Variant VariantArray);
```

File

CorbaObj

Description

This is function CorbaObj.VariantArrayToSequence.

34.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCorbaInstancing (see page 6484)	This is record CorbaObj.TCorbaInstancing.
	TCorbaThreadModel (see page 6484)	This is record CorbaObj.TCorbaThreadModel.

Records

	Name	Description
	TInterfaceIDEntryDesc (see page 6485)	This is record CorbaObj.TInterfaceIDEntryDesc.
	TSkeletonEntryDesc (see page 6485)	This is record CorbaObj.TSkeletonEntryDesc.
	TStubEntryDesc (see page 6486)	This is record CorbaObj.TStubEntryDesc.

34.4.1 CorbaObj.TCorbaInstancing Enumeration

Pascal

```
TCorbaInstancing = (
  iSingleInstance,
  iMultiInstance
);
```

C++

```
enum TCorbaInstancing {
  iSingleInstance,
  iMultiInstance
};
```

File

CorbaObj

Description

This is record CorbaObj.TCorbaInstancing.

34.4.2 CorbaObj.TCorbaThreadModel Enumeration

Pascal

```
TCorbaThreadModel = (
  tmMultiThreaded,
  tmSingleThread
);
```

C++

```
enum TCorbaThreadModel {
  tmMultiThreaded,
  tmSingleThread
```

```
};
```

File

CorbaObj

Description

This is record CorbaObj.TCorbaThreadModel.

34.4.3 CorbaObj.TInterfaceIDEntryDesc Record

Pascal

```
TInterfaceIDEntryDesc = record
  RepositoryID: string;
  IID: TGUID;
end;
```

C++

```
struct TInterfaceIDEntryDesc {
  AnsiString RepositoryID;
  TGUID IID;
};
```

File

CorbaObj

Description

This is record CorbaObj.TInterfaceIDEntryDesc.

34.4.4 CorbaObj.TSkeletonEntryDesc Record

Pascal

```
TSkeletonEntryDesc = record
  IID: TGUID;
  SkeletonClass: TCorbaSkeletonClass;
end;
```

C++

```
struct TSkeletonEntryDesc {
  TGUID IID;
  TCorbaSkeletonClass SkeletonClass;
};
```

File

CorbaObj

Description

This is record CorbaObj.TSkeletonEntryDesc.

34.4.5 CorbaObj.TStubEntryDesc Record

Pascal

```
TStubEntryDesc = record
  IID: TGUID;
  StubClass: TCorbaStubClass;
end;
```

C++

```
struct TStubEntryDesc {
  TGUID IID;
  TCorbaStubClass StubClass;
};
```

File

CorbaObj

Description

This is record CorbaObj.TStubEntryDesc.

34.5 Types

The following table lists types in this documentation.

Types

Name	Description
CorbaBoolean (see page 6487)	This is type CorbaObj.CorbaBoolean.
CorbaULong (see page 6487)	This is type CorbaObj.CorbaULong.
IObject (see page 6488)	This is type CorbaObj.IObject.
ITypeCode (see page 6488)	This is type CorbaObj.ITypeCode.
TAny (see page 6488)	This is type CorbaObj.TAny.
TCKind (see page 6489)	This is type CorbaObj.TCKind.
TCommandLine (see page 6489)	This is type CorbaObj.TCommandLine.
TCorbaImplementationClass (see page 6489)	This is type CorbaObj.TCorbaImplementationClass.
TCorbaPrincipal (see page 6490)	This is type CorbaObj.TCorbaPrincipal.
TCorbaSkeletonClass (see page 6490)	This is type CorbaObj.TCorbaSkeletonClass.
TCorbaStubClass (see page 6490)	This is type CorbaObj.TCorbaStubClass.
TFactoryList (see page 6491)	This is type CorbaObj.TFactoryList.
TInterfaceIDList (see page 6491)	This is type CorbaObj.TInterfaceIDList.
TSkeletonList (see page 6491)	This is type CorbaObj.TSkeletonList.
TStubList (see page 6492)	This is type CorbaObj.TStubList.

34.5.1 CorbaObj.CorbaBoolean Type

Pascal

```
CorbaBoolean = ORBPAS.CorbaBoolean;
```

C++

```
ORBPAS.CorbaBoolean CorbaBoolean;
```

File

CorbaObj

Description

This is type CorbaObj.CorbaBoolean.

34.5.2 CorbaObj.CorbaULong Type

Pascal

```
CorbaULong = ORBPAS.CorbaULong;
```

C++

```
ORBPAS.CorbaULong CorbaULong;
```

File

CorbaObj

Description

This is type CorbaObj.CorbaULong.

34.5.3 CorbaObj.IObject Type

Pascal

```
IObject = System.IUnknown;
```

C++

```
System.IUnknown IObject;
```

File

CorbaObj

Description

This is type CorbaObj.IObject.

34.5.4 CorbaObj.ITypeCode Type

Pascal

```
ITypeCode = ORBPAS.ITypeCode;
```

C++

```
ORBPAS.ITypeCode ITypeCode;
```

File

CorbaObj

Description

This is type CorbaObj.ITypeCode.

34.5.5 CorbaObj.TAny Type

Pascal

```
TAny = Variant;
```

C++

```
Variant TAny;
```

File

CorbaObj

Description

This is type CorbaObj.TAny.

34.5.6 CorbaObj.TCKind Type

Pascal

```
TCKind = ORBPAS.TCKind;
```

C++

```
ORBPAS.TCKind TCKind;
```

File

CorbaObj

Description

This is type CorbaObj.TCKind.

34.5.7 CorbaObj.TCommandLine Type

Pascal

```
TCommandLine = ORBPAS.TArgv;
```

C++

```
ORBPAS.TArgv TCommandLine;
```

File

CorbaObj

Description

This is type CorbaObj.TCommandLine.

34.5.8 CorbaObj.TCorbaImplementationClass Type

Pascal

```
TCorbaImplementationClass = class of TCorbaImplementation;
```

C++

```
TCorbaImplementation TCorbaImplementationClass;
```

File

CorbaObj

Description

This is type CorbaObj.TCorbaImplementationClass.

34.5.9 CorbaObj.TCorbaPrincipal Type

Pascal

```
TCorbaPrincipal = array of Byte;
```

C++

```
array of Byte TCorbaPrincipal;
```

File

CorbaObj

Description

This is type CorbaObj.TCorbaPrincipal.

34.5.10 CorbaObj.TCorbaSkeletonClass Type

Pascal

```
TCorbaSkeletonClass = class of TCorbaSkeleton;
```

C++

```
TCorbaSkeleton TCorbaSkeletonClass;
```

File

CorbaObj

Description

This is type CorbaObj.TCorbaSkeletonClass.

34.5.11 CorbaObj.TCorbaStubClass Type

Pascal

```
TCorbaStubClass = class of TCorbaStub;
```

C++

```
TCorbaStub TCorbaStubClass;
```

File

CorbaObj

Description

This is type CorbaObj.TCorbaStubClass.

34.5.12 CorbaObj.TFactoryList Type

Pascal

```
TFactoryList = array of TCorbaFactory;
```

C++

```
array of TCorbaFactory TFactoryList;
```

File

CorbaObj

Description

This is type CorbaObj.TFactoryList.

34.5.13 CorbaObj.TInterfaceIDList Type

Pascal

```
TInterfaceIDList = array of TInterfaceIDEntryDesc;
```

C++

```
array of TInterfaceIDEntryDesc TInterfaceIDList;
```

File

CorbaObj

Description

This is type CorbaObj.TInterfaceIDList.

34.5.14 CorbaObj.TSkeletonList Type

Pascal

```
TSkeletonList = array of TSkeletonEntryDesc;
```

C++

```
array of TSkeletonEntryDesc TSkeletonList;
```

File

CorbaObj

Description

This is type CorbaObj.TSkeletonList.

34.5.15 CorbaObj.TStubList Type

Pascal

```
TStubList = array of TStubEntryDesc;
```

C++

```
array of TStubEntryDesc TStubList;
```

File

CorbaObj

Description

This is type CorbaObj.TStubList.

34.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
CorbaInterfaceIDManager (see page 6493)	This is variable CorbaObj.CorbaInterfaceIDManager.
CorbaSkeletonManager (see page 6493)	This is variable CorbaObj.CorbaSkeletonManager.
CorbaStubManager (see page 6493)	This is variable CorbaObj.CorbaStubManager.

34.6.1 CorbaObj.CorbaInterfaceIDManager Variable

Pascal

```
CorbaInterfaceIDManager: TCorbaInterfaceIDManager;
```

C++

```
TCorbaInterfaceIDManager CorbaInterfaceIDManager;
```

File

CorbaObj

Description

This is variable CorbaObj.CorbaInterfaceIDManager.

34.6.2 CorbaObj.CorbaSkeletonManager Variable

Pascal

```
CorbaSkeletonManager: TCorbaSkeletonManager;
```

C++

```
TCorbaSkeletonManager CorbaSkeletonManager;
```

File

CorbaObj

Description

This is variable CorbaObj.CorbaSkeletonManager.

34.6.3 CorbaObj.CorbaStubManager Variable

Pascal

```
CorbaStubManager: TCorbaStubManager;
```

C++

```
TCorbaStubManager CorbaStubManager;
```

File

CorbaObj

Description

This is variable CorbaObj.CorbaStubManager.

35 CtlPanel Namespace

This is namespace CtlPanel.

Topics

Name	Description
Classes (see page 6497)	The following table lists classes in this documentation.
Functions (see page 6519)	The following table lists functions in this documentation.
Types (see page 6520)	The following table lists types in this documentation.
Variables (see page 6526)	The following table lists variables in this documentation.

Classes

	Name	Description
	EAppletException (see page 6497)	EAppletException is the exception class for errors that occur in a Control Panel Application (see page 6526).
	TAppletApplication (see page 6497)	TAppletApplication represents a Windows control panel application.
	TAppletModule (see page 6507)	TAppletModule represents a single Control Panel Applet in a control panel applet application.

Functions

	Name	Description
	CPlApplet (see page 6519)	This is function CtlPanel.CPlApplet.

Types

Name	Description
TActivateEvent (see page 6520)	TActivateEvent is the type of the OnActivate event handler.
TAppletModuleClass (see page 6520)	This is type CtlPanel.TAppletModuleClass.
TCountEvent (see page 6521)	TCountEvent is the type of the OnCount event handler.
TCPLAppletClass (see page 6521)	This is type CtlPanel.TCPLAppletClass.
TDataModuleClass (see page 6522)	This is type CtlPanel.TDataModuleClass.
TExitEvent (see page 6522)	TExitEvent is the type for event handlers that respond when a control panel application is about to shut down.
TInitEvent (see page 6522)	TInitEvent is the type of the OnInit event handler.
TInquireEvent (see page 6523)	TInquireEvent is the type of the OnInquire event handler.
TNewInquireEvent (see page 6523)	TNewInquireEvent is the type of the OnNewInquire event handler.
TOnAppletExceptionEvent (see page 6524)	TOnAppletExceptionEvent is the type for event handlers that respond when a control panel applet encounters an exception.

TSetupEvent (see page 6524)	TSetupEvent is the type for event handlers that respond when a control panel applet is invoked from the command line during system installation.
TStartWParmsEvent (see page 6524)	TStartWParmsEvent is the type of the OnStartWParms event handler.
TStopEvent (see page 6525)	TStopEvent is the type of the OnStop event handler.

Variables

Name	Description
Application (see page 6526)	Represents the application for a control panel applet project.

35.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EAppletException (see page 6497)	EAppletException is the exception class for errors that occur in a Control Panel Application (see page 6526).
	TAppletApplication (see page 6497)	TAppletApplication represents a Windows control panel application.
	TAppletModule (see page 6507)	TAppletModule represents a single Control Panel Applet in a control panel applet application.

35.1.1 EAppletException Class

EAppletException is the exception class for errors that occur in a Control Panel Application (see page 6526).

Class Hierarchy

Pascal

```
EAppletException = class(Exception);
```

C++

```
class EAppletException : public Exception;
```

File

CtlPanel

Description

EAppletException is the exception for errors specific to applications created by the control panel wizard.

See Also

TAppletApplication (see page 6497)

TAppletModule (see page 6507)

35.1.2 TAppletApplication Class

TAppletApplication represents a Windows control panel application.

Class Hierarchy

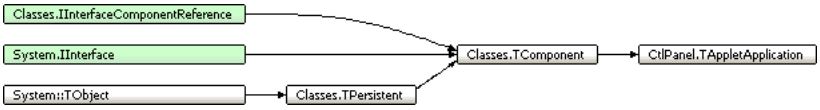

Pascal

```
TAppletApplication = class(TComponent);
```

C++

```
class TAppletApplication : public TComponent;
```

File

CtlPanel

Description

TAppletApplication represents an application that implements one or more applets, each of which appear in the Windows control panel. Each applet that the application implements is built using an AppletModule, which is a special data module that holds the nonvisual components in the applet and which responds when users execute the applet from the control panel. TAppletApplication contains a set of AppletModules. It lets you respond to a number of events that are called by the Windows control panel for starting up or shutting down the application.

A global Application (see page 6526) variable of type TAppletApplication is declared in the CtlPanel (see page 6495) unit.

See Also

TAppletModule (see page 6507)

EAppletException (see page 6497)

35.1.2.1 TAppletApplication Members

The following tables list the members exposed by TAppletApplication.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoCount (see page 6502)	This is DoCount, a member of class TAppletApplication.
	DoExit (see page 6503)	This is DoExit, a member of class TAppletApplication.
	DoHandleException (see page 6503)	This is DoHandleException, a member of class TAppletApplication.
	DoInit (see page 6503)	This is DoInit, a member of class TAppletApplication.
	DoSetup (see page 6503)	This is DoSetup, a member of class TAppletApplication.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.

	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnCount (see page 6506)	Occurs when the control panel requests a count of the AppletModules.
	OnException (see page 6507)	Occurs when an exception raised by the application is not handled.
	OnInit (see page 6507)	Occurs when the Application (see page 6526) is first initialized.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 6501)	Instantiates a TAppletApplication (see page 6497) object.
V	CreateForm (see page 6502)	Creates the Applet object or objects for the applet application.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 6501)	Disposes of an instance of a TAppletApplication (see page 6497) object.

	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandleException (see page 6503)	Responds to exceptions that are not caught by a try block in the application.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 6504)	Sets the initial state of the application and hides any associated MainForm.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Run (see page 6504)	Sets the Applications Exception handler.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.

R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlPanelHandle (see page 6505)	Indicates the window handle for the controlling application.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	ModuleCount (see page 6505)	Indicates the number of AppletModules available.
	Modules (see page 6505)	Lists the available modules in the Applet application.
	OnExit (see page 6506)	Occurs when the Application (see page 6526) should unload.
	OnSetup (see page 6506)	Occurs when the applet is invoked from the command line during system installation.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

35.1.2.2 TAppletApplication.Create Constructor

Instantiates a TAppletApplication (see page 6497) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TAppletApplication(TComponent * AOwner);
```

Description

Applications do not need to call the constructor directly. TAppletApplication (see page 6497) is instantiated by selecting File | New | Other from the menu of the IDE main window and selecting Applet Application (see page 6526) in the New Items dialog. An Application (see page 6526) variable of type TAppletApplication (see page 6497) is declared and initialized as the instance of the application.

35.1.2.3 TAppletApplication.Destroy Destructor

Disposes of an instance of a TAppletApplication (see page 6497) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAppletApplication();
```

Description

Do not call the destructor directly. TAppletApplication (see page 6497) is destroyed when your application closes.

The destructor frees the memory associated with sub-objects.

See Also

Free (see page 22300)

35.1.2.4 TAppletApplication Methods

The methods of the TAppletApplication class are listed here.

Protected Methods

	Name	Description
	DoCount (see page 6502)	This is DoCount, a member of class TAppletApplication.
	DoExit (see page 6503)	This is DoExit, a member of class TAppletApplication.
	DoHandleException (see page 6503)	This is DoHandleException, a member of class TAppletApplication.
	DoInit (see page 6503)	This is DoInit, a member of class TAppletApplication.
	DoSetup (see page 6503)	This is DoSetup, a member of class TAppletApplication.

Public Methods

	Name	Description
V	CreateForm (see page 6502)	Creates the Applet object or objects for the applet application.
	HandleException (see page 6503)	Responds to exceptions that are not caught by a try block in the application.
V	Initialize (see page 6504)	Sets the initial state of the application and hides any associated MainForm.
V	Run (see page 6504)	Sets the Applications Exception handler.

35.1.2.4.1 TAppletApplication.CreateForm Method

Creates the Applet object or objects for the applet application.

Pascal

```
procedure CreateForm(InstanceClass: TComponentClass; var Reference); virtual;
```

C++

```
virtual __fastcall CreateForm(TComponentClass InstanceClass,  Reference);
```

Description

Do not call CreateForm in an application.The module manager calls CreateForm internally to register the applet objects with the applet application.

CreateForm instantiates a TApplet object for each applet contained in the applet application. CreateForm is called automatically, for the first applet module, when the applet application is first created. Subsequently, CreateForm is called for each new applet module added to the application. To add a new applet module, select File | New | Other from the menu of the IDE main window and choose Applet Module in the New Items dialog.

35.1.2.4.2 TAppletApplication.DoCount Method

Pascal

```
procedure DoCount(var AppCount: Integer); dynamic;
```

C++

```
__fastcall DoCount(int AppCount);
```

Description

This is DoCount, a member of class TAppletApplication.

35.1.2.4.3 TAppletApplication.DoExit Method

Pascal

```
procedure DoExit; dynamic;
```

C++

```
__fastcall DoExit();
```

Description

This is DoExit, a member of class TAppletApplication.

35.1.2.4.4 TAppletApplication.DoHandleException Method

Pascal

```
procedure DoHandleException(Sender: TObject; E: Exception); dynamic;
```

C++

```
__fastcall DoHandleException(TObject * Sender, Exception E);
```

Description

This is DoHandleException, a member of class TAppletApplication.

35.1.2.4.5 TAppletApplication.DoInit Method

Pascal

```
procedure DoInit(var AppInitOK: Boolean); dynamic;
```

C++

```
__fastcall DoInit(Boolean AppInitOK);
```

Description

This is DoInit, a member of class TAppletApplication.

35.1.2.4.6 TAppletApplication.DoSetup Method

Pascal

```
procedure DoSetup; dynamic;
```

C++

```
__fastcall DoSetup();
```

Description

This is DoSetup, a member of class TAppletApplication.

35.1.2.4.7 TAppletApplication.HandleException Method

Responds to exceptions that are not caught by a try block in the application.

Pascal

```
procedure HandleException(Sender: TObject);
```

C++

```
__fastcall HandleException(TObject * Sender);
```

Description

HandleException is called automatically when an exception is not caught by any of the try blocks in the application code. It calls the global ExceptObject function to obtain the exception object.

If ExceptObject is Exception or one of its descendants, HandleException generates an OnException (see page 6507) event.

If ExceptObject is not an Exception descendant, or if there is no OnException (see page 6507) event handler, HandleException calls the global ShowException method to display a message box that indicates what exception occurred.

Sender is the object that raised the exception.

See Also

OnException (see page 6507)

ExceptObject (see page 22452)

ShowException (see page 22941)

35.1.2.4.8 TAppletApplication.Initialize Method

Sets the initial state of the application and hides any associated MainForm.

Pascal

```
procedure Initialize; virtual;
```

C++

```
virtual __fastcall Initialize();
```

Description

Initialize follows the standard (default) VCL application initialization process.

35.1.2.4.9 TAppletApplication.Run Method

Sets the Applications Exception handler.

Pascal

```
procedure Run; virtual;
```

C++

```
virtual __fastcall Run();
```

Description

Run sets the Application (see page 6526)'s Exception handler.

35.1.2.5 TAppletApplication Properties

The properties of the TAppletApplication class are listed here.

Public Properties

	Name	Description
R	ControlPanelHandle (see page 6505)	Indicates the window handle for the controlling application.

	ModuleCount (see page 6505)	Indicates the number of AppletModules available.
	Modules (see page 6505)	Lists the available modules in the Applet application.
	OnExit (see page 6506)	Occurs when the Application (see page 6526) should unload.
	OnSetup (see page 6506)	Occurs when the applet is invoked from the command line during system installation.

35.1.2.5.1 TAppletApplication.ControlPanelHandle Property

Indicates the window handle for the controlling application.

Pascal

```
property ControlPanelHandle: THandle;
```

C++

```
__property THandle ControlPanelHandle;
```

Description

ControlPanelHandle is the handle to the Control Panel or the controlling application (the application which invoked the CplApplet function).

35.1.2.5.2 TAppletApplication.ModuleCount Property

Indicates the number of AppletModules available.

Pascal

```
property ModuleCount: Integer;
```

C++

```
__property int ModuleCount;
```

Description

ModuleCount is the number of AppletModules that are available to the controlling application from this AppletApplication.

See Also

Modules (see page 6505)

35.1.2.5.3 TAppletApplication.Modules Property

Lists the available modules in the Applet application.

Pascal

```
property Modules [Index: Integer]: TAppletModule;
```

C++

```
__property TAppletModule Modules[int Index];
```

Description

Modules is a list of the TAppletModule (see page 6507) objects that are available to the AppletApplication.

See Also

ModuleCount (see page 6505)

35.1.2.5.4 TAppletApplication.OnExit Property

Occurs when the Application (see page 6526) should unload.

Pascal

```
property OnExit: TExitEvent;
```

C++

```
__property TExitEvent OnExit;
```

Description

The OnExit event occurs when the Application (see page 6526) receives a CPL_EXIT message from the controlling application. This is a recommended spot to do non-applet specific clean-up.

35.1.2.5.5 TAppletApplication.OnSetup Property

Occurs when the applet is invoked from the command line during system installation.

Pascal

```
property OnSetup: TSetupEvent;
```

C++

```
__property TSetupEvent OnSetup;
```

Description

This event occurs when the CPL_SETUP message is received from the controlling application. This message is only sent when an applet is invoked from the Command line during system installation.

35.1.2.6 TAppletApplication Events

The events of the TAppletApplication class are listed here.

Public Events

	Name	Description
	OnCount (see page 6506)	Occurs when the control panel requests a count of the AppletModules.
	OnException (see page 6507)	Occurs when an exception raised by the application is not handled.
	OnInit (see page 6507)	Occurs when the Application (see page 6526) is first initialized.

35.1.2.6.1 TAppletApplication.OnCount Event

Occurs when the control panel requests a count of the AppletModules.

Pascal

```
property OnCount: TCountEvent;
```

C++

```
__property TCountEvent OnCount;
```

Description

The OnCount event occurs when the Application (see page 6526) receives a CPL_GETCOUNT message from the controlling application. It must return the number of AppletModules that are available. The default is to return ModuleCount (see page 6505), but it may be overridden by changing the value of the AppCount parameter of the event handler.

35.1.2.6.2 TAppletApplication.OnException Event

Occurs when an exception raised by the application is not handled.

Pascal

```
property OnException: TOnAppletExceptionEvent;
```

C++

```
__property TOnAppletExceptionEvent OnException;
```

Description

Use OnException to change the default behavior that occurs when an exception is not handled by application code. If an exception passes through the try blocks in the application code, the application automatically calls the HandleException (see page 6503) method. The HandleException (see page 6503) method generates an OnException event when it encounters an exception that descends from Exception. If there is no OnException event handler, or if the Exception object does not descend from Exception, the applet displays a message box indicating an error occurred.

See Also

HandleException (see page 6503)

35.1.2.6.3 TAppletApplication.OnInit Event

Occurs when the Application (see page 6526) is first initialized.

Pascal

```
property OnInit: TInitEvent;
```

C++

```
__property TInitEvent OnInit;
```

Description

The OnInit event occurs when the Application (see page 6526) receives a CPL_INIT message from the controlling application. The application must return either true or false to determine whether the controlling application should attempt to continue loading this module. The default is to return true, but this value may be overridden by assigning a value to the AppInitOK parameter of the event handler.

35.1.3 TAppletModule Class

TAppletModule represents a single Control Panel Applet in a control panel applet application.

Class Hierarchy

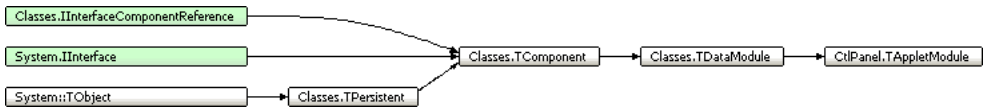

Pascal

```
TAppletModule = class(TDataModule);
```

C++

```
class TAppletModule : public TDataModule;
```

File

CtlPanel

Description

TAppletModule encapsulates the functionality of a Control Panel Applet. Its properties specify how the icon for the applet appears in the control panel, including the help that appears when users press F1. The events of TAppletModule let you specify what the applet does when users start or stop it, either by double-clicking in the control panel or by launching the applet programmatically.

See Also

TAppletApplication (see page 6497)

EAppletException (see page 6497)

35.1.3.1 TAppletModule Members

The following tables list the members exposed by TAppletModule.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoActivate (see page 6512)	This is DoActivate, a member of class TAppletModule.
V	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
V	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
	DoInquire (see page 6512)	This is DoInquire, a member of class TAppletModule.
	DoNewInquire (see page 6512)	This is DoNewInquire, a member of class TAppletModule.
	DoStartWParms (see page 6513)	This is DoStartWParms, a member of class TAppletModule.
	DoStop (see page 6513)	This is DoStop, a member of class TAppletModule.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 2829)	Generates an OnCreate (see page 2833) event.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 2830)	Generates an OnDestroy (see page 2834) event.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 6511)	Instantiates a TAppletModule (see page 6507) object.
V	CreateNew (see page 2828)	Creates an instance of a data module and registers with the global screen object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 6511)	Disposes of an instance of a TAppletModule (see page 6507) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Data (see page 6514)	Specifies user-defined data.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnActivate (see page 6516)	Occurs when the applet's icon is double clicked.
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.
	OnInquire (see page 6516)	Occurs when the AppletModule is queried for its information.
	OnNewInquire (see page 6517)	Occurs when the AppletModule is queried for its information.
	OnStartWParms (see page 6517)	Occurs when the AppletModule is invoked from RunDLL.
	OnStop (see page 6517)	Occurs when the AppletModule receives a request to shutdown.

Published Properties

	Name	Description
	AppletIcon (see page 6513)	Indicates the icon for the AppletModule.
	Caption (see page 6514)	Indicates the caption of the applet module.
	Help (see page 6514)	Indicates a help string that appears in the control panel for this applet.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.
	ResidIcon (see page 6515)	Indicates the resource ID of the icon associated with the applet module.
	ResidInfo (see page 6515)	Indicates the resource ID of the help string.
	ResidName (see page 6515)	Indicates the resource ID of the caption string.
	Tag (see page 2808)	Stores an integer value as part of a component.

35.1.3.2 TAppletModule.Create Constructor

Instantiates a TAppletModule (see page 6507) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TAppletModule(TComponent * AOwner);
```

Description

Create constructs a TAppletModule (see page 6507) instance and adds itself to the list of Modules in the TAppletApplication (see page 6497) specified as the value of AOwner.

35.1.3.3 TAppletModule.Destroy Destructor

Disposes of an instance of a TAppletModule (see page 6507) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAppletModule();
```

Description

The destructor destroys the specified instance of a TAppletModule (see page 6507) and removes it from the list of applet modules in the owning TAppletApplication (see page 6497).

Typically, applications do not call Destroy, because the owning TAppletApplication (see page 6497) is responsible for freeing them. However, if an Owner is not specified, do not call Destroy directly. Rather, call the Free method.

See Also

Free (see page 22300)

35.1.3.4 TAppletModule Methods

The methods of the TAppletModule class are listed here.

Protected Methods

	Name	Description
	DoActivate (see page 6512)	This is DoActivate, a member of class TAppletModule.
	DoInquire (see page 6512)	This is DoInquire, a member of class TAppletModule.
	DoNewInquire (see page 6512)	This is DoNewInquire, a member of class TAppletModule.
	DoStartWParms (see page 6513)	This is DoStartWParms, a member of class TAppletModule.
	DoStop (see page 6513)	This is DoStop, a member of class TAppletModule.

35.1.3.4.1 TAppletModule.DoActivate Method

Pascal

```
procedure DoActivate(Data: LongInt); dynamic;
```

C++

```
__fastcall DoActivate(LongInt Data);
```

Description

This is DoActivate, a member of class TAppletModule.

35.1.3.4.2 TAppletModule.DoInquire Method

Pascal

```
procedure DoInquire(var ACPLInfo: TCPLInfo); dynamic;
```

C++

```
__fastcall DoInquire(TCPLInfo ACPLInfo);
```

Description

This is DoInquire, a member of class TAppletModule.

35.1.3.4.3 TAppletModule.DoNewInquire Method

Pascal

```
procedure DoNewInquire(var ANewCPLInfo: TNewCPLInfo); dynamic;
```

C++

```
__fastcall DoNewInquire(TNewCPLInfo ANewCPLInfo);
```

Description

This is DoNewInquire, a member of class TAppletModule.

35.1.3.4.4 TAppletModule.DoStartWParms Method

Pascal

```
procedure DoStartWParms(Params: string); dynamic;
```

C++

```
__fastcall DoStartWParms(AnsiString Params);
```

Description

This is DoStartWParms, a member of class TAppletModule.

35.1.3.4.5 TAppletModule.DoStop Method

Pascal

```
procedure DoStop(Data: LongInt); dynamic;
```

C++

```
__fastcall DoStop(LongInt Data);
```

Description

This is DoStop, a member of class TAppletModule.

35.1.3.5 TAppletModule Properties

The properties of the TAppletModule class are listed here.

Public Properties

	Name	Description
	Data (see page 6514)	Specifies user-defined data.

Published Properties

	Name	Description
	AppletIcon (see page 6513)	Indicates the icon for the AppletModule.
	Caption (see page 6514)	Indicates the caption of the applet module.
	Help (see page 6514)	Indicates a help string that appears in the control panel for this applet.
	ResidIcon (see page 6515)	Indicates the resource ID of the icon associated with the applet module.
	ResidInfo (see page 6515)	Indicates the resource ID of the help string.
	ResidName (see page 6515)	Indicates the resource ID of the caption string.

35.1.3.5.1 TAppletModule.AppletIcon Property

Indicates the icon for the AppletModule.

Pascal

```
property AppletIcon: TIcon;
```

C++

```
__property TIcon AppletIcon;
```

Description

AppletIcon contains the icon that is associated with this Applet Module.

See Also

Caption (see page 6514)

ResidIcon (see page 6515)

35.1.3.5.2 TAppletModule.Caption Property

Indicates the caption of the applet module.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption is the string that shows in the Control Panel under the icon of the Applet.

See Also

AppletIcon (see page 6513)

Help (see page 6514)

ResidName (see page 6515)

35.1.3.5.3 TAppletModule.Data Property

Specifies user-defined data.

Pascal

```
property Data: LongInt;
```

C++

```
__property LongInt Data;
```

Description

Data is a user-defined value that is passed to the controlling application on the OnInquire (see page 6516)/OnNewInquire (see page 6517) events.

See Also

OnInquire (see page 6516)

OnNewInquire (see page 6517)

35.1.3.5.4 TAppletModule.Help Property

Indicates a help string that appears in the control panel for this applet.

Pascal

```
property Help: string;
```

C++

```
__property AnsiString Help;
```

Description

Help is the string that is visible in the status bar of the Control Panel when a particular Applet is selected.

See Also

Caption (see page 6514)

ResidInfo (see page 6515)

35.1.3.5.5 TAppletModule.ResidIcon Property

Indicates the resource ID of the icon associated with the applet module.

Pascal

```
property ResidIcon: Integer;
```

C++

```
__property int ResidIcon;
```

Description

ResidIcon is the resource ID of the icon associated with the applet module. This property is mutually exclusive with the AppletIcon (see page 6513) property.

See Also

AppletIcon (see page 6513)

35.1.3.5.6 TAppletModule.ResidInfo Property

Indicates the resource ID of the help string.

Pascal

```
property ResidInfo: Integer;
```

C++

```
__property int ResidInfo;
```

Description

ResidInfo is the resource ID of the help string associated with the applet module. This property is mutually exclusive with the Help (see page 6514) property.

See Also

Help (see page 6514)

35.1.3.5.7 TAppletModule.ResidName Property

Indicates the resource ID of the caption string.

Pascal

```
property ResidName: Integer;
```

C++

```
__property int ResidName;
```

Description

ResidName is the resource ID of the caption string associated with the applet module. This property is mutually exclusive with the Caption (see page 6514) property.

See Also

Caption (see page 6514)

35.1.3.6 TAppletModule Events

The events of the TAppletModule class are listed here.

Published Events

	Name	Description
	OnActivate (see page 6516)	Occurs when the applet's icon is double clicked.
	OnInquire (see page 6516)	Occurs when the AppletModule is queried for its information.
	OnNewInquire (see page 6517)	Occurs when the AppletModule is queried for its information.
	OnStartWParms (see page 6517)	Occurs when the AppletModule is invoked from RunDLL.
	OnStop (see page 6517)	Occurs when the AppletModule receives a request to shutdown.

35.1.3.6.1 TAppletModule.OnActivate Event

Occurs when the applet's icon is double clicked.

Pascal

```
property OnActivate: TActivateEvent;
```

C++

```
__property TActivateEvent OnActivate;
```

Description

OnActivate is invoked when the Applet icon is double clicked in the Control Panel.

See Also

OnInquire (see page 6516)

OnNewInquire (see page 6517)

OnStartWParms (see page 6517)

35.1.3.6.2 TAppletModule.OnInquire Event

Occurs when the AppletModule is queried for its information.

Pascal

```
property OnInquire: TInquireEvent;
```

C++

```
__property TInquireEvent OnInquire;
```

Description

OnInquire occurs when the controlling application requests each applet's information such as its icon, caption, Help (see page 6514) and Data (see page 6514) values. This event populates a structure with the resource ID's which are changed in this event (such as idIcon, idName, idInfo and lData). These values default to the corresponding module properties.

See Also

Data (see page 6514)

OnNewInquire (see page 6517)

35.1.3.6.3 TAppletModule.OnNewInquire Event

Occurs when the AppletModule is queried for its information.

Pascal

```
property OnNewInquire: TNewInquireEvent;
```

C++

```
__property TNewInquireEvent OnNewInquire;
```

Description

OnNewInquire occurs when the controlling application requests each applet's information such as its Icon, Caption (see page 6514), Help (see page 6514), and Data (see page 6514) Values. This event populates a structure with the values that are changed in this event (such as hIcon, AppletName, AppletInfo, and lData). HIcon defaults to the handle of the AppletIcon (see page 6513) property. AppletName defaults to Caption (see page 6514), AppletInfo defaults to Help (see page 6514), and lData defaults to the Data (see page 6514) property of the AppletModule.

See Also

Data (see page 6514)

OnInquire (see page 6516)

35.1.3.6.4 TAppletModule.OnStartWParms Event

Occurs when the AppletModule is invoked from RunDLL.

Pascal

```
property OnStartWParms: TStartWParmsEvent;
```

C++

```
__property TStartWParmsEvent OnStartWParms;
```

Description

OnStartWParms parallels the OnActivate (see page 6516) event. Where it differs is that the invocation is coming from RunDLL. Params may contain extra directions for execution.

See Also

OnActivate (see page 6516)

35.1.3.6.5 TAppletModule.OnStop Event

Occurs when the AppletModule receives a request to shutdown.

Pascal

```
property OnStop: TStopEvent;
```

C++

```
__property TStopEvent OnStop;
```

Description

Write an OnStop event handler to perform Applet-specific clean-up. OnStop occurs when the applet module receives a request to shutdown.

See Also

OnStartWParms (see page 6517)

OnActivate (see page 6516)

OnInquire (see page 6516)

OnNewInquire (see page 6517)

35.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CPlApplet (see page 6519)	This is function CtlPanel.CPlApplet.

35.2.1 CtlPanel.CPlApplet Function

Pascal

```
function CPlApplet(hwndCPl: THandle; uMsg: DWORD; lParam1: Longint; lParam2: Longint):
Longint; stdcall;
```

C++

```
__stdcall Longint CPlApplet(THandle hwndCPl, DWORD uMsg, Longint lParam1, Longint lParam2);
```

File

CtlPanel

Description

This is function CtlPanel.CPlApplet.

35.3 Types

The following table lists types in this documentation.

Types

Name	Description
TActivateEvent (see page 6520)	TActivateEvent is the type of the OnActivate event handler.
TAppletModuleClass (see page 6520)	This is type CtlPanel.TAppletModuleClass.
TCountEvent (see page 6521)	TCountEvent is the type of the OnCount event handler.
TCPLAppletClass (see page 6521)	This is type CtlPanel.TCPLAppletClass.
TDataModuleClass (see page 6522)	This is type CtlPanel.TDataModuleClass.
TExitEvent (see page 6522)	TExitEvent is the type for event handlers that respond when a control panel application is about to shut down.
TInitEvent (see page 6522)	TInitEvent is the type of the OnInit event handler.
TInquireEvent (see page 6523)	TInquireEvent is the type of the OnInquire event handler.
TNewInquireEvent (see page 6523)	TNewInquireEvent is the type of the OnNewInquire event handler.
TOnAppletExceptionEvent (see page 6524)	TOnAppletExceptionEvent is the type for event handlers that respond when a control panel applet encounters an exception.
TSetupEvent (see page 6524)	TSetupEvent is the type for event handlers that respond when a control panel applet is invoked from the command line during system installation.
TStartWParmsEvent (see page 6524)	TStartWParmsEvent is the type of the OnStartWParms event handler.
TStopEvent (see page 6525)	TStopEvent is the type of the OnStop event handler.

35.3.1 CtlPanel.TActivateEvent Type

TActivateEvent is the type of the OnActivate event handler.

Pascal

```
TActivateEvent = procedure (Sender: TObject; Data: LongInt) of object;
```

C++

```
(Sender: TObject; Data: LongInt) ( TActivateEvent)();
```

File

CtlPanel

Description

TActivateEvent lets you respond when the user first activates a control panel applet.

Sender is the Applet Module which was clicked.

Data is the value that was set in the Inquire events (OnInquire or OnNewInquire).

35.3.2 CtlPanel.TAppletModuleClass Type

Pascal

```
TAppletModuleClass = class of TAppletModule;
```

C++

```
TAppletModule TAppletModuleClass;
```

File

CtlPanel

Description

This is type CtlPanel.TAppletModuleClass.

35.3.3 CtlPanel.TCountEvent Type

TCountEvent is the type of the OnCount event handler.

Pascal

```
TCountEvent = procedure (Sender: TObject; var AppCount: Integer) of object;
```

C++

```
(Sender: TObject; var AppCount: Integer) ( TCountEvent)();
```

File

CtlPanel

Description

TCountEvent lets you override the indication a control panel application gives for number of applets it makes available.

Sender is the control panel application object.

AppCount returns the number of modules. On entry, this is the number of TAppletModule (see page 6507) instances in the application. The event handler can change this amount.

35.3.4 CtlPanel.TCPLAppletClass Type

Pascal

```
TCPLAppletClass = class of TAppletModule;
```

C++

```
TAppletModule TCPLAppletClass;
```

File

CtlPanel

Description

This is type CtlPanel.TCPLAppletClass.

35.3.5 CtlPanel.TDataModuleClass Type

Pascal

```
TDataModuleClass = class of TDataModule;
```

C++

```
TDataModule TDataModuleClass;
```

File

CtlPanel

Description

This is type CtlPanel.TDataModuleClass.

35.3.6 CtlPanel.TExitEvent Type

TExitEvent is the type for event handlers that respond when a control panel application is about to shut down.

Pascal

```
TExitEvent = TNotifyEvent;
```

C++

```
TNotifyEvent TExitEvent;
```

File

CtlPanel

Description

TExitEvent maps to the TNotifyEvent type. It has a single parameter, Sender, which is the TAppletApplication (see page 6497) instance.

35.3.7 CtlPanel.TInitEvent Type

TInitEvent is the type of the OnInit event handler.

Pascal

```
TInitEvent = procedure (Sender: TObject; var AppInitOK: Boolean) of object;
```

C++

```
(Sender: TObject; var AppInitOK: Boolean) ( TInitEvent)();
```

File

CtlPanel

Description

TInitEvent lets you indicate whether the controlling application should finish loading a control panel application.

Sender is the control panel application object.

AppInitOK returns true if the controlling application should continue loading the application, false if the controlling application should abort this application.

35.3.8 CtlPanel.TInquireEvent Type

TInquireEvent is the type of the OnInquire event handler.

Pascal

```
TInquireEvent = procedure (Sender: TObject; var idIcon: Integer; var idName: Integer; var
idInfo: Integer; var lData: Integer) of object;
```

C++

```
(Sender: TObject; var idIcon: Integer; var idName: Integer; var idInfo: Integer; var lData:
Integer) ( TInquireEvent)();
```

File

CtlPanel

Description

TInquireEvent lets you provide information about a control panel applet.

Sender is the Applet Module about which you are providing information.

idIcon is the resource ID for the icon of the applet.

idName is the resource ID for the applet's name as it appears in the control panel.

lData is a user-defined value that you want to have passed to the applet's OnActivate event handler.

35.3.9 CtlPanel.TNewInquireEvent Type

TNewInquireEvent is the type of the OnNewInquire event handler.

Pascal

```
TNewInquireEvent = procedure (Sender: TObject; var lData: Integer; var hIcon: HICON; var
AppletName: string; var AppletInfo: string) of object;
```

C++

```
(Sender: TObject; var lData: Integer; var hIcon: HICON; var AppletName: string; var
AppletInfo: string) ( TNewInquireEvent)();
```

File

CtlPanel

Description

TNewInquireEvent lets you provide information about a control panel applet.

Sender is the Applet Module about which you are providing information.

lData is a user-defined value that you want to have passed to the applet's OnActivate event handler.

hIcon is the handle to the icon for the applet.

AppletName is the applet's name as it appears in the control panel.

AppletInfo is the help string that describes the applet in the control panel.

35.3.10 CtlPanel.TOnAppletExceptionEvent Type

TOnAppletExceptionEvent is the type for event handlers that respond when a control panel applet encounters an exception.

Pascal

```
TOnAppletExceptionEvent = procedure (Sender: TObject; E: Exception) of object;
```

C++

```
(Sender: TObject; E: Exception) ( TOnAppletExceptionEvent)();
```

File

CtlPanel

Description

TOnAppletExceptionEvent lets you customize the response to exceptions that occur in a control panel application.

Sender is the object that raised the exception.

E is the exception object that describes the exception.

35.3.11 CtlPanel.TSetupEvent Type

TSetupEvent is the type for event handlers that respond when a control panel applet is invoked from the command line during system installation.

Pascal

```
TSetupEvent = TNotifyEvent;
```

C++

```
TNotifyEvent TSetupEvent;
```

File

CtlPanel

Description

TSetupEvent maps to the TNotifyEvent type. It has a single parameter, Sender, which is the TAppletApplication (see page 6497) instance.

35.3.12 CtlPanel.TStartWParmsEvent Type

TStartWParmsEvent is the type of the OnStartWParms event handler.

Pascal

```
TStartWParmsEvent = procedure (Sender: TObject; Params: string) of object;
```

C++

```
(Sender: TObject; Params: string) ( TStartWParmsEvent)();
```

File

CtlPanel

Description

TStartWParmsEvent lets you respond when a control panel applet first starts up.

Sender is the Applet Module that is starting up.

Params is the parameter string that is passed to RunDLL.

35.3.13 CtlPanel.TStopEvent Type

TStopEvent is the type of the OnStop event handler.

Pascal

```
TStopEvent = procedure (Sender: TObject; Data: LongInt) of object;
```

C++

```
(Sender: TObject; Data: LongInt) ( TStopEvent)();
```

File

CtlPanel

Description

TStopEvent lets you respond when a control panel applet receives a shutdown request.

Sender is the Applet Module that received the request to shut down.

Data is the value that was set in the Inquire events (OnInquire or OnNewInquire).

35.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
Application (see page 6526)	Represents the application for a control panel applet project.

35.4.1 CtlPanel.Application Variable

Represents the application for a control panel applet project.

Pascal

```
Application: TAppletApplication = nil;
```

C++

```
TAppletApplication Application = nil;
```

File

CtlPanel

Description

When a new control panel project is created, the IDE constructs an application object and assigns it to Application. If the project represents a control panel applet, Application is initialized to a TAppletApplication (see page 6497) object.

See Also

,

36 CustomizeDlg Namespace

This is namespace CustomizeDlg.

Topics

Name	Description
Classes (see page 6528)	The following table lists classes in this documentation.

Classes

	Name	Description
	TCustomizeDlg (see page 6528)	TCustomizeDlg is the customization dialog for action bands.
	TCustomizeFrm (see page 6535)	This is class CustomizeDlg.TCustomizeFrm.

36.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCustomizeDlg (see page 6528)	TCustomizeDlg is the customization dialog for action bands.
	TCustomizeFrm (see page 6535)	This is class CustomizeDlg.TCustomizeFrm.

36.1.1 TCustomizeDlg Class

TCustomizeDlg is the customization dialog for action bands.

Class Hierarchy

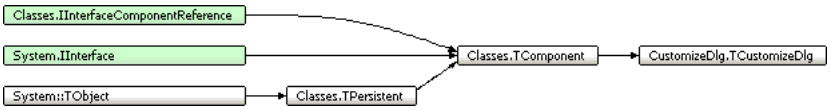

Pascal

```
TCustomizeDlg = class(TComponent);
```

C++

```
class TCustomizeDlg : public TComponent;
```

File

CustomizeDlg

Description

TCustomizeDlg is the customization dialog for action bands. This dialog appears when a user indicates a wish to customize user interface elements using a control provided by the application developer. The dialog is also used in the IDE when action bands for an application are first being created.

See Also

Creating Customizable Toolbars and Menus

36.1.1.1 TCustomizeDlg Members

The following tables list the members exposed by TCustomizeDlg.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.

	_Release (see page 2769)	Called when an application releases a component interface.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 6531)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetupDlg (see page 6532)	This is SetupDlg, a member of class TCustomizeDlg.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	Show (see page 6532)	Displays the customization dialog.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CustomizeForm (see page 6533)	Provides access to the form that implements the customize dialog.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnClose (see page 6534)	Occurs when the dialog closes.
	OnShow (see page 6534)	Occurs when the dialog opens.

Published Properties

	Name	Description
	ActionManager (see page 6533)	Indicates the action manager associated with the customization dialog.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	StayOnTop (see page 6533)	Specifies whether the customization dialog can be placed behind other windows.
	Tag (see page 2808)	Stores an integer value as part of a component.

36.1.1.2 TCustomizeDlg Methods

The methods of the TCustomizeDlg class are listed here.

Protected Methods

	Name	Description
V	Notification (see page 6531)	Forwards notification messages to all owned components.
V	SetupDlg (see page 6532)	This is SetupDlg, a member of class TCustomizeDlg.

Public Methods

	Name	Description
	Show (see page 6532)	Displays the customization dialog.

36.1.1.2.1 TCustomizeDlg.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification

to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

36.1.1.2.2 TCustomizeDlg.SetupDlg Method

Pascal

```
procedure SetupDlg; virtual;
```

C++

```
virtual __fastcall SetupDlg();
```

Description

This is SetupDlg, a member of class TCustomizeDlg.

36.1.1.2.3 TCustomizeDlg.Show Method

Displays the customization dialog.

Pascal

```
procedure Show;
```

C++

```
__fastcall Show();
```

Description

Call Show to display the customization dialog. Show causes the form that the CustomizeForm (see page 6533) property specifies to become visible. Immediately before the dialog appears to the user, TCustomizeDlg (see page 6528) receives an OnShow (see page 6534) event.

See Also

OnShow (see page 6534)

CustomizeForm (see page 6533)

36.1.1.3 TCustomizeDlg Properties

The properties of the TCustomizeDlg class are listed here.

Public Properties

	Name	Description
R	CustomizeForm (see page 6533)	Provides access to the form that implements the customize dialog.

Published Properties

	Name	Description
	ActionManager (see page 6533)	Indicates the action manager associated with the customization dialog.
	StayOnTop (see page 6533)	Specifies whether the customization dialog can be placed behind other windows.

36.1.1.3.1 TCustomizeDlg.ActionManager Property

Indicates the action manager associated with the customization dialog.

Pascal

```
property ActionManager: TCustomActionManager;
```

C++

```
__property TCustomActionManager ActionManager;
```

Description

At design time, set ActionManager to refer to your application's action manager object. The action manager maintains a logical description of the user interface elements as a hierarchy of action clients. The customization dialog operates on this hierarchy to change how user interface elements are organized or to hide and show individual elements.

See Also

TActionManager (see page 432)

ActionBars (see page 553)

36.1.1.3.2 TCustomizeDlg.CustomizeForm Property

Provides access to the form that implements the customize dialog.

Pascal

```
property CustomizeForm: TCustomizeFrm;
```

C++

```
__property TCustomizeFrm CustomizeForm;
```

Description

Read CustomizeForm to access the form that TCustomizeDlg (see page 6528) displays when you call the Show (see page 6532) method.

See Also

Show (see page 6532)

36.1.1.3.3 TCustomizeDlg.StayOnTop Property

Specifies whether the customization dialog can be placed behind other windows.

Pascal

```
property StayOnTop: Boolean;
```

C++

```
__property Boolean StayOnTop;
```

Description

If you want the customization dialog to stay on top of all windows while it is visible, set StayOnTop to true. Otherwise, it should be set to false.

36.1.1.4 TCustomizeDlg Events

The events of the TCustomizeDlg class are listed here.

Published Events

	Name	Description
	OnClose (see page 6534)	Occurs when the dialog closes.
	OnShow (see page 6534)	Occurs when the dialog opens.

36.1.1.4.1 TCustomizeDlg.OnClose Event

Occurs when the dialog closes.

Pascal

```
property OnClose: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClose;
```

Description

Write an OnClose event handler if you want special actions taken when the customization dialog closes.

See Also

OnShow (see page 6534)

36.1.1.4.2 TCustomizeDlg.OnShow Event

Occurs when the dialog opens.

Pascal

```
property OnShow: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnShow;
```

Description

Write an OnShow event handler if you want special actions taken immediately before the customization dialog opens.

See Also

OnClose (see page 6534)

Show (see page 6532)

36.1.2 TCustomizeFrm Class

Class Hierarchy

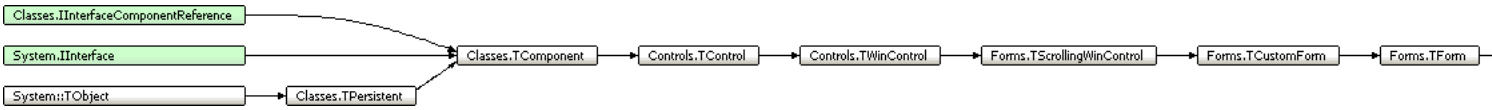

Pascal

```
TCustomizeFrm = class(TForm);
```

C++

```
class TCustomizeFrm : public TForm;
```

File

CustomizeDlg

Description

This is class CustomizeDlg.TCustomizeFrm.

36.1.2.1 TCustomizeFrm Members

The following tables list the members exposed by TCustomizeFrm.

Protected Events

	Name	Description
	OnActivate (see page 13943)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnCanResize (see page 13947)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnClick (see page 13947)	Occurs when the user clicks the control.
	OnClose (see page 13949)	Occurs when the form closes.
	OnCloseQuery (see page 13950)	Occurs when close is attempted.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 13951)	Occurs when the form is created.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDblClick (see page 13953)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 13953)	Occurs when the form loses focus.
	OnDestroy (see page 13953)	Occurs when the form is destroyed.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragDrop (see page 13954)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDragOver (see page 13957)	Occurs when the user drags an object over a control.

	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnHelp (see page 13959)	Occurs when the form receives a request for help.
	OnHide (see page 13960)	Occurs when the form is hidden (that is, when its Visible (see page 13940) property is set to false).
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyDown (see page 13962)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyPress (see page 13964)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnKeyUp (see page 13966)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseActivate (see page 13967)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseDown (see page 13968)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseMove (see page 13971)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseUp (see page 13973)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnPaint (see page 13975)	Occurs when the form is redrawn.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnResize (see page 13976)	Occurs immediately after the control is resized.
	OnShortCut (see page 13977)	Occurs when the user presses a key (before the OnKeyDown (see page 13962) event).
	OnShow (see page 13977)	Occurs when the form is shown (that is, when its Visible (see page 13940) property is set to true).
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.

	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFormState (see page 13863)	This is FFormState, a member of class TCustomForm.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	Activate (see page 13865)	Activation event dispatcher.
	ActiveChanged (see page 13865)	Active (see page 13896) state event dispatcher.
	AddAction (see page 6567)	This is AddAction, a member of class TCustomizeFrm.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 14324)	Adjusts the ClientRect property to exclude the scroll bars.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 13866)	Aligns all child controls of the form that fall within a specified rectangle.
V	AlignControls (see page 14325)	Aligns the controls in the scrolling windowed control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
V	AutoScrollEnabled (see page 14325)	Indicates whether automatic scrolling is enabled.
V	AutoScrollInView (see page 14325)	Scrolls a control into the visible area of the scrolling windowed control.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
V	BeginAutoDrag (see page 13867)	Responds when the user starts dragging the form if DragMode is dmAutomatic.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
V	ChangeScale (see page 13867)	Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.
V	ChangeScale (see page 14326)	Resizes the scrolling windowed control by a specific ratio.
V	ClearCatList (see page 6568)	This is ClearCatList, a member of class TCustomizeFrm.
	Click (see page 5743)	Respond to user click.
	CloseModal (see page 13868)	This is CloseModal, a member of class TCustomForm.
	CMVisiblechanged (see page 6568)	This is CMVisiblechanged, a member of class TCustomizeFrm.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.

	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 13868)	Initializes the window-creation parameter record when the form window is created.
V	CreateParams (see page 14326)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWindowHandle (see page 13869)	Creates a form window given a set of window-creation parameters.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CreateWnd (see page 13869)	Creates the form window.
V	CreateWnd (see page 14327)	Creates a Windows control corresponding to the TScrollingWinControl (see page 14312) object.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	Deactivate (see page 13870)	Generates an OnDeactivate (see page 13953) event.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
V	DefineProperties (see page 13871)	Designates methods for storing a form's unpublished data on a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyHandle (see page 13871)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWindowHandle (see page 13872)	Destroys the window created in the CreateWindowHandle (see page 13869) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoClose (see page 13873)	Form closing event dispatcher.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
V	DoCreate (see page 13873)	Form creation event dispatcher.
V	DoDestroy (see page 13873)	Form destruction event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
V	DoDock (see page 13874)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
V	DoFlipChildren (see page 14327)	Flips the control's children.
	DoHide (see page 13874)	Form hide event dispatcher.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.

	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoShow (see page 13875)	Form show event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindActionBars (see page 6568)	This is FindActionBars, a member of class TCustomizeFrm.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorderIconStyles (see page 13875)	This is GetBorderIconStyles, a member of class TCustomForm.
	GetBorderStyles (see page 13876)	This is GetBorderStyles, a member of class TCustomForm.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetChildren (see page 13876)	Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetClientRect (see page 13876)	Returns the value of the ClientRect property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloating (see page 13877)	Returns the value of the Floating property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetOwnerWindow (see page 13878)	This is GetOwnerWindow, a member of class TCustomForm.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event

	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 13878)	Handles exceptions that are raised by the OnCreate (see page 13951) event handler.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsDupShortCut (see page 6569)	This is IsDupShortCut, a member of class TCustomizeFrm.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 13880)	Performs fix-ups when the form is first loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 6570)	Responds to notifications that objects are about to be inserted or removed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 13882)	Generates an OnPaint (see page 13975) event.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
V	PaintWindow (see page 13882)	Renders the image of the form.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
V	PaletteChanged (see page 13883)	Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	QueryInterface (see page 13884)	Returns a reference to a specified interface if that interface is supported by the form.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
V	ReadState (see page 13884)	Reads the form's published properties, stored data, and owned components from a reader object.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
V	RequestAlign (see page 13885)	Instructs the form's parent (if any) to reposition the form, enforcing its Align property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.

	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
V	Resizing (see page 13885)	Makes any required adjustments when the form changes size.
V	Resizing (see page 14328)	This is Resizing, a member of class TScrollingWinControl.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActiveActionList (see page 6572)	This is SetActiveActionList, a member of class TCustomizeFrm.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
V	SetChildOrder (see page 13886)	Changes the order in which Child appears when GetChildren (see page 13876) is called.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParent (see page 13887)	Sets the Parent property.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
V	SetParentBiDiMode (see page 13887)	Sets the ParentBiDiMode property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
V	UpdateActions (see page 13891)	Updates all actions associated with the form.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateDialog (see page 6573)	This is UpdateDialog, a member of class TCustomizeFrm.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.

V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	UpdateWindowState (see page 13892)	Updates the WindowState (see page 13942) property to reflect the current state of the form.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	ValidateRename (see page 13892)	Determines if a form can rename one of its owned components.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	VisibleChanging (see page 13893)	Responds just before the form changes the value of its Visible (see page 13940) property.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WndProc (see page 13893)	Provides specific message responses for the form.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
R	ActiveMDIChild (see page 13897)	Specifies the MDI child that has focus.
	AlphaBlend (see page 13898)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 13898)	Specifies the degree of translucency on a translucent form.
	AutoScroll (see page 13899)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoScroll (see page 14329)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderIcons (see page 13899)	Specifies which icons appear on the title bar of the form.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
R	ClientHandle (see page 13904)	Provides access to the handle of the form's internal MDI client window.
W	ClientHeight (see page 13905)	Specifies the height (in pixels) of the form's client area.
W	ClientWidth (see page 13905)	Specifies the width (in pixels) of the form's client area.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Ctl3D (see page 13909)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultMonitor (see page 13909)	Specifies the monitor on which the form appears.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.

	FormStyle (see page 13912)	Determines the form's style.
	Height (see page 13915)	Specifies the vertical size of the control in pixels.
	HorzScrollBar (see page 13916)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 13917)	Specifies the icon that appears when the form is minimized.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	MDIChildCount (see page 13921)	Specifies the number of open MDI child forms.
R	MDIChildren (see page 13922)	Provides indexed access to all MDI child forms.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ObjectMenuItem (see page 13926)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 13926)	Specifies when OnCreate (see page 13951) and OnDestroy (see page 13953) events occur.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentFont (see page 13927)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PixelsPerInch (see page 13928)	Represents the proportion of the font on the system on which the form was designed.
R	PopupChildren (see page 13928)	This is PopupChildren, a member of class TCustomForm.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	PopupMenu (see page 13928)	Identifies the pop-up menu associated with the control.
	Position (see page 13933)	Represents the size and placement of the form.
	PrintScale (see page 13934)	Represents the proportions of a printed form.
	Scaled (see page 13935)	Specifies whether the form is sized according to the value of the PixelsPerInch (see page 13928) property.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	TileMode (see page 13936)	Represents how MDI child forms are arranged when the Tile method is called.
	TransparentColor (see page 13938)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 13939)	Indicates the color on the form that appears transparent when TransparentColor (see page 13938) is true.
	VertScrollBar (see page 13939)	Represents the vertical scroll bar for the scrolling windowed control.
W	Visible (see page 13940)	Indicates whether the form is visible.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	Width (see page 13941)	Specifies the horizontal size of the control or form in pixels.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowMenu (see page 13941)	Specifies the Window menu for an MDI parent form.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Fields

	Name	Description
	ActionBarList (see page 6554)	This is ActionBarList, a member of class TCustomizeFrm.
	ActionImages (see page 6554)	This is ActionImages, a member of class TCustomizeFrm.
	ActionList1 (see page 6554)	This is ActionList1, a member of class TCustomizeFrm.
	ActionsActionsLbl (see page 6554)	This is ActionsActionsLbl, a member of class TCustomizeFrm.
	ActionsCatLbl (see page 6554)	This is ActionsCatLbl, a member of class TCustomizeFrm.

	ActionsList (see page 6555)	This is ActionsList, a member of class TCustomizeFrm.
	ActionsTab (see page 6555)	This is ActionsTab, a member of class TCustomizeFrm.
	ApplyToAllActn (see page 6555)	This is ApplyToAllActn, a member of class TCustomizeFrm.
	ApplyToAllChk (see page 6555)	This is ApplyToAllChk, a member of class TCustomizeFrm.
	CaptionOptionsCombo (see page 6556)	This is CaptionOptionsCombo, a member of class TCustomizeFrm.
	CaptionOptionsGrp (see page 6556)	This is CaptionOptionsGrp, a member of class TCustomizeFrm.
	CatList (see page 6556)	This is CatList, a member of class TCustomizeFrm.
	CloseActn (see page 6556)	This is CloseActn, a member of class TCustomizeFrm.
	CloseBtn (see page 6556)	This is CloseBtn, a member of class TCustomizeFrm.
	CloseItem (see page 6557)	This is CloseItem, a member of class TCustomizeFrm.
	CloseMenu (see page 6557)	This is CloseMenu, a member of class TCustomizeFrm.
	ComboPanel (see page 6557)	This is ComboPanel, a member of class TCustomizeFrm.
	DescGroupBox (see page 6557)	This is DescGroupBox, a member of class TCustomizeFrm.
	FullMenusActn (see page 6558)	This is FullMenusActn, a member of class TCustomizeFrm.
	HintLbl (see page 6558)	This is HintLbl, a member of class TCustomizeFrm.
	InfoLbl (see page 6558)	This is InfoLbl, a member of class TCustomizeFrm.
	Label1 (see page 6558)	This is Label1, a member of class TCustomizeFrm.
	Label2 (see page 6558)	This is Label2, a member of class TCustomizeFrm.
	Label4 (see page 6559)	This is Label4, a member of class TCustomizeFrm.
	LargeIconsActn (see page 6559)	This is LargeIconsActn, a member of class TCustomizeFrm.
	LargeIconsChk (see page 6559)	This is LargeIconsChk, a member of class TCustomizeFrm.
	ListCombo (see page 6559)	This is ListCombo, a member of class TCustomizeFrm.
	ListPanel (see page 6560)	This is ListPanel, a member of class TCustomizeFrm.
	MenuAnimationStyles (see page 6560)	This is MenuAnimationStyles, a member of class TCustomizeFrm.
	OptionsBevel1 (see page 6560)	This is OptionsBevel1, a member of class TCustomizeFrm.
	OptionsBevel2 (see page 6560)	This is OptionsBevel2, a member of class TCustomizeFrm.
	OptionsTab (see page 6560)	This is OptionsTab, a member of class TCustomizeFrm.
	OtherLbl (see page 6561)	This is OtherLbl, a member of class TCustomizeFrm.
	PersonalizeLbl (see page 6561)	This is PersonalizeLbl, a member of class TCustomizeFrm.
	RecentlyUsedActn (see page 6561)	This is RecentlyUsedActn, a member of class TCustomizeFrm.
	RecentlyUsedChk (see page 6561)	This is RecentlyUsedChk, a member of class TCustomizeFrm.
	ResetActn (see page 6562)	This is ResetActn, a member of class TCustomizeFrm.
	ResetBtn (see page 6562)	This is ResetBtn, a member of class TCustomizeFrm.
	ResetUsageBtn (see page 6562)	This is ResetUsageBtn, a member of class TCustomizeFrm.
	ResetUsageDataActn (see page 6562)	This is ResetUsageDataActn, a member of class TCustomizeFrm.
	SeparatorBtn (see page 6562)	This is SeparatorBtn, a member of class TCustomizeFrm.
	ShortCutTipsChk (see page 6563)	This is ShortCutTipsChk, a member of class TCustomizeFrm.
	ShowHintsActn (see page 6563)	This is ShowHintsActn, a member of class TCustomizeFrm.
	ShowShortCutsInTipsActn (see page 6563)	This is ShowShortCutsInTipsActn, a member of class TCustomizeFrm.
	ShowTipsChk (see page 6563)	This is ShowTipsChk, a member of class TCustomizeFrm.
	Tabs (see page 6564)	This is Tabs, a member of class TCustomizeFrm.
	ToolbarsLbl (see page 6564)	This is ToolbarsLbl, a member of class TCustomizeFrm.
	ToolbarsTab (see page 6564)	This is ToolbarsTab, a member of class TCustomizeFrm.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActionBarListClick (see page 6565)	This is ActionBarListClick, a member of class TCustomizeFrm.
	ActionBarListClickCheck (see page 6565)	This is ActionBarListClickCheck, a member of class TCustomizeFrm.
	ActionsListClick (see page 6566)	This is ActionsListClick, a member of class TCustomizeFrm.
	ActionsListData (see page 6566)	This is ActionsListData, a member of class TCustomizeFrm.
	ActionsListDrawItem (see page 6566)	This is ActionsListDrawItem, a member of class TCustomizeFrm.
	ActionsListMeasureItem (see page 6566)	This is ActionsListMeasureItem, a member of class TCustomizeFrm.

	ActionsListStartDrag (see page 6566)	This is ActionsListStartDrag, a member of class TCustomizeFrm.
V	AfterConstruction (see page 13866)	Construction event dispatcher.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	ApplyToAllActnUpdate (see page 6567)	This is ApplyToAllActnUpdate, a member of class TCustomizeFrm.
	ArrangeIcons (see page 14006)	Arranges the icons of minimized MDI child forms.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 13866)	Destruction event dispatcher.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	CaptionOptionsComboChange (see page 6567)	This is CaptionOptionsComboChange, a member of class TCustomizeFrm.
	Cascade (see page 14007)	Arranges MDI child forms so they overlap.
	CatListClick (see page 6567)	This is CatListClick, a member of class TCustomizeFrm.
	CatListStartDrag (see page 6568)	This is CatListStartDrag, a member of class TCustomizeFrm.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Close (see page 13867)	Closes the form.
	CloseBtnClick (see page 6568)	This is CloseBtnClick, a member of class TCustomizeFrm.
V	CloseQuery (see page 13868)	Close (see page 13867) attempt event dispatcher.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 13861)	Creates and initializes a new TForm (see page 13992) object.
V	Create (see page 14323)	Creates and initializes a new TScrollingWinControl (see page 14312) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	CreateNew (see page 13862)	Creates and initializes a new form.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 13870)	Provides message handling for all messages that the form does not fully process.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DefocusControl (see page 13871)	Removes focus from a control on the form.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 13862)	Removes the form from memory.
V	Destroy (see page 14323)	Removes the TScrollingWinControl (see page 14312) object from memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	DisableAutoRange (see page 14327)	Disables automatic scrolling.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
V	Dock (see page 13872)	Handles automatic docking of the form into other windows.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EnableAutoRange (see page 14328)	Reenables automatic scrolling.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	FocusControl (see page 13875)	Sets focus to a control on the form.
	Focused (see page 6150)	Determines whether the control has input focus.
	FormClose (see page 6569)	This is FormClose, a member of class TCustomizeFrm.
	FormCreate (see page 6569)	This is FormCreate, a member of class TCustomizeFrm.
	FormResize (see page 6569)	This is FormResize, a member of class TCustomizeFrm.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFormImage (see page 13877)	Returns a bitmap of the form.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	Hide (see page 13879)	Hides the form.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	IsShortCut (see page 13880)	Processes shortcut keys when the form has focus.
	LargeIconsActnExecute (see page 6570)	This is LargeIconsActnExecute, a member of class TCustomizeFrm.
	LargeIconsActnUpdate (see page 6570)	This is LargeIconsActnUpdate, a member of class TCustomizeFrm.
	ListComboSelect (see page 6570)	This is ListComboSelect, a member of class TCustomizeFrm.
	MakeFullyVisible (see page 13881)	Ensures that the form is fully visible on a specified monitor.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MenuAnimationStylesChange (see page 6570)	This is MenuAnimationStylesChange, a member of class TCustomizeFrm.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MouseWheelHandler (see page 13881)	Directs mouse wheel messages to the appropriate control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 14008)	Activates the next child form in the form sequence.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	Previous (see page 14009)	Activates the previous MDI child form in the form sequence.
	Print (see page 13883)	Prints the form.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	RecentlyUsedActnExecute (see page 6571)	This is RecentlyUsedActnExecute, a member of class TCustomizeFrm.
	RecentlyUsedActnUpdate (see page 6571)	This is RecentlyUsedActnUpdate, a member of class TCustomizeFrm.
	RecreateAsPopup (see page 13884)	This is RecreateAsPopup, a member of class TCustomForm.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	Release (see page 13885)	Destroys the form and frees its associated memory.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	ResetActnExecute (see page 6571)	This is ResetActnExecute, a member of class TCustomizeFrm.
	ResetActnUpdate (see page 6572)	This is ResetActnUpdate, a member of class TCustomizeFrm.
	ResetUsageDataActnExecute (see page 6572)	This is ResetUsageDataActnExecute, a member of class TCustomizeFrm.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.

	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollInView (see page 14328)	Scrolls a control into the visible area of the scrolling windowed control.
	SendCancelMode (see page 13886)	Cancels modes on the form.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SeparatorBtnStartDrag (see page 6572)	This is SeparatorBtnStartDrag, a member of class TCustomizeFrm.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
V	SetFocus (see page 13886)	Sets focus to the form.
V	SetFocusedControl (see page 13887)	Sets focus to a control on the form.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Show (see page 13888)	Shows the form.
	ShowHintsActnExecute (see page 6572)	This is ShowHintsActnExecute, a member of class TCustomizeFrm.
	ShowHintsActnUpdate (see page 6573)	This is ShowHintsActnUpdate, a member of class TCustomizeFrm.
V	ShowModal (see page 13890)	Shows a form as a modal dialog.
	ShowShortCutsInTipsActnExecute (see page 6573)	This is ShowShortCutsInTipsActnExecute, a member of class TCustomizeFrm.
	Tile (see page 14009)	Arranges MDI child forms so that they are all the same size.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UpdateGlassFrame (see page 13892)	This is UpdateGlassFrame, a member of class TCustomForm.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.
V	WantChildKey (see page 13893)	Indicates whether the form processes keyboard input for a control it owns.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Action (see page 13895)	Designates the action associated with the control.
	ActionManager (see page 6573)	This is ActionManager, a member of class TCustomizeFrm.
R	Active (see page 13896)	Specifies whether the form has focus.
	ActiveActionList (see page 6574)	This is ActiveActionList, a member of class TCustomizeFrm.
	ActiveControl (see page 13896)	Specifies the control that has focus on the form.
	ActiveMDIChild (see page 14013)	Specifies the MDI child that has focus.
	ActiveOleControl (see page 13897)	Specifies the OLE control on the form that reacts to changes in focus.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 13900)	Specifies the appearance and behavior of the form border.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 13900)	Provides access to the drawing area of the form.
	Caption (see page 13902)	Specifies a text string that identifies the control to the user.
	ClientHandle (see page 14024)	Provides access to the handle of the form's internal MDI client window.

	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 13908)	Specifies the background color of the control.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designer (see page 13910)	Specifies the designer interface for the form.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockManager (see page 14029)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropTarget (see page 13910)	Specifies whether the form is the target of a drag-and-drop operation.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Font (see page 13910)	Controls the attributes of text written on or in the control.
R	FormState (see page 13911)	Indicates transitional state information about the form.
	GlassFrame (see page 13914)	This is GlassFrame, a member of class TCustomForm.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HelpFile (see page 13916)	Specifies the name of the file the form uses to display Help.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	KeyPreview (see page 13918)	Specifies whether the form should receive keyboard events before the active control.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	MDIChildCount (see page 14044)	Specifies the number of open MDI child forms.
	MDIChildren (see page 14044)	Provides indexed access to all MDI child forms.
	Menu (see page 13923)	Specifies the form's main menu.
	ModalResult (see page 13924)	Represents the return value of a form that is used as a modal dialog.
R	Monitor (see page 13926)	Provides access to the monitor on which the form appears.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.

	OleFormObject (see page 13927)	Specifies the IOleForm (see page 14340) interface for an in-place OLE object contained in the form.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PopupMode (see page 13933)	This is PopupMode, a member of class TCustomForm.
	PopupParent (see page 13933)	Sets an order for stacked forms that users cannot change.
	ScreenSnap (see page 13935)	Specifies whether form snaps to edge of screen.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	SnapBuffer (see page 13936)	Specifies distance for screen snap.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TileMode (see page 14059)	Represents how MDI child forms are arranged when the Tile method is called.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.
	WindowState (see page 13942)	Represents how the form appears on the screen.

Published Events

	Name	Description
	OnActivate (see page 14065)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 14069)	Occurs when an object with custom alignment is aligned.
	OnAlignInsertBefore (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14069)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 14070)	Occurs when an attempt is made to resize the control.
	OnClick (see page 14070)	Occurs when the user clicks the control.
	OnClose (see page 14072)	Occurs when the form closes.
	OnCloseQuery (see page 14073)	Occurs when close is attempted.
	OnConstrainedResize (see page 14074)	Adjust resize constraints.
	OnContextPopup (see page 14075)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 14076)	Occurs when the form is created.
	OnDblClick (see page 14077)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 14078)	Occurs when the form loses focus.
	OnDestroy (see page 14078)	Occurs when the form is destroyed.
	OnDockDrop (see page 14079)	Occurs when another control is docked to the control.
	OnDockOver (see page 14081)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 14083)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 14085)	Occurs when the user drags an object over a control.
	OnEndDock (see page 14088)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnGetSiteInfo (see page 14088)	Returns the control's docking information.
	OnHelp (see page 14090)	Occurs when the form receives a request for help.
	OnHide (see page 14091)	Occurs when the form is hidden (that is, when its Visible property is set to false).
	OnKeyDown (see page 14093)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 14095)	Occurs when a key is pressed.

	OnKeyUp (see page 14096)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 14098)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 14099)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 14101)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 14102)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 14102)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 14104)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 14107)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 14107)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 14107)	Occurs when the mouse wheel is rotated upward.
	OnPaint (see page 14108)	Occurs when the form is redrawn.
	OnResize (see page 14109)	Occurs immediately after the control is resized.
	OnShortCut (see page 14109)	Occurs when the user presses a key (before the OnKeyDown event).
	OnShow (see page 14110)	Occurs when the form is shown (that is, when its Visible property is set to true).
	OnStartDock (see page 14111)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnUnDock (see page 14111)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Action (see page 14011)	Designates the action associated with the control.
	ActiveControl (see page 14012)	Specifies the control that has focus on the form.
	Align (see page 14013)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AlphaBlend (see page 14017)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 14017)	Specifies the degree of translucency on a translucent form.
	Anchors (see page 14017)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 14018)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 14018)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BiDiMode (see page 14019)	Specifies the bi-directional mode for the control.
	BorderIcons (see page 14019)	Specifies which icons appear on the title bar of the form.
	BorderStyle (see page 14020)	Specifies the appearance and behavior of the form border.
	BorderWidth (see page 14021)	Specifies the width of the control's border.
	Caption (see page 14021)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 14024)	Specifies the height (in pixels) of the form's client area.
	ClientWidth (see page 14025)	Specifies the width (in pixels) of the form's client area.
	Color (see page 14027)	Specifies the background color of the control.
	Constraints (see page 14028)	Specifies the size constraints for the control.
	Ctl3D (see page 14028)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DefaultMonitor (see page 14029)	Specifies the monitor on which the form appears.
	DockSite (see page 14031)	Specifies whether the control can be the target of drag-and-dock operations.
	DragKind (see page 14032)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 14033)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 14034)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 14035)	Controls the attributes of text written on or in the control.
	FormStyle (see page 14036)	Determines the form's style.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	Height (see page 14038)	Specifies the vertical size of the control in pixels.

	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpFile (see page 14039)	Specifies the name of the file the form uses to display Help.
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HorzScrollBar (see page 14040)	Represents the horizontal scroll bar for the scrolling windowed control.
	HorzScrollBar (see page 14329)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 14041)	Specifies the icon that appears when the form is minimized.
	KeyPreview (see page 14041)	Specifies whether the form should receive keyboard events before the active control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Left (see page 13920)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Menu (see page 14045)	Specifies the form's main menu.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectMenuItem (see page 14047)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 14047)	Specifies when OnCreate and OnDestroy events occur.
	Padding (see page 14047)	Specifies the padding of a control.
	ParentBiDiMode (see page 14048)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 14048)	Determines where a control looks for its font information.
	PixelsPerInch (see page 14049)	Represents the proportion of the font on the system on which the form was designed.
	PopupMenu (see page 14049)	Identifies the pop-up menu associated with the control.
	PopupMode (see page 14054)	Controls (see page 5711) how the top-level form behaves with respect to Window's WS_POPUP style.
	PopupParent (see page 14054)	Sets an order for stacked forms that users cannot change.
	Position (see page 14055)	Represents the size and placement of the form.
	PrintScale (see page 14056)	Represents the proportions of a printed form.
	Scaled (see page 14056)	Specifies whether the form is sized according to the value of the PixelsPerInch property.
	ScreenSnap (see page 14056)	Specifies whether form snaps to edge of screen.
	ShowHint (see page 14057)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SnapBuffer (see page 14058)	Specifies distance for screen snap.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Top (see page 13937)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	TransparentColor (see page 14060)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 14060)	Indicates the color on the form that appears transparent when TransparentColor is true.
	UseDockManager (see page 14060)	Specifies whether the docking manager is used in drag-and-dock operations.
	VertScrollBar (see page 14061)	Represents the vertical scroll bar for the scrolling windowed control.
	VertScrollBar (see page 14330)	Represents the vertical scroll bar for the scrolling windowed control.
	Visible (see page 14062)	Indicates whether the form is visible.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	Width (see page 14063)	Specifies the horizontal size of the control or form in pixels.
	WindowMenu (see page 14063)	Specifies the Window menu for an MDI parent form.
	WindowState (see page 14063)	Represents how the form appears on the screen.

36.1.2.2 TCustomizeFrm Fields

The fields of the TCustomizeFrm class are listed here.

Public Fields

	Name	Description
	ActionBarList (see page 6554)	This is ActionBarList, a member of class TCustomizeFrm.
	ActionImages (see page 6554)	This is ActionImages, a member of class TCustomizeFrm.
	ActionList1 (see page 6554)	This is ActionList1, a member of class TCustomizeFrm.
	ActionsActionsLbl (see page 6554)	This is ActionsActionsLbl, a member of class TCustomizeFrm.
	ActionsCatLbl (see page 6554)	This is ActionsCatLbl, a member of class TCustomizeFrm.
	ActionsList (see page 6555)	This is ActionsList, a member of class TCustomizeFrm.
	ActionsTab (see page 6555)	This is ActionsTab, a member of class TCustomizeFrm.
	ApplyToAllActn (see page 6555)	This is ApplyToAllActn, a member of class TCustomizeFrm.
	ApplyToAllChk (see page 6555)	This is ApplyToAllChk, a member of class TCustomizeFrm.
	CaptionOptionsCombo (see page 6556)	This is CaptionOptionsCombo, a member of class TCustomizeFrm.
	CaptionOptionsGrp (see page 6556)	This is CaptionOptionsGrp, a member of class TCustomizeFrm.
	CatList (see page 6556)	This is CatList, a member of class TCustomizeFrm.
	CloseActn (see page 6556)	This is CloseActn, a member of class TCustomizeFrm.
	CloseBtn (see page 6556)	This is CloseBtn, a member of class TCustomizeFrm.
	CloseItem (see page 6557)	This is CloseItem, a member of class TCustomizeFrm.
	CloseMenu (see page 6557)	This is CloseMenu, a member of class TCustomizeFrm.
	ComboPanel (see page 6557)	This is ComboPanel, a member of class TCustomizeFrm.
	DescGroupBox (see page 6557)	This is DescGroupBox, a member of class TCustomizeFrm.
	FullMenusActn (see page 6558)	This is FullMenusActn, a member of class TCustomizeFrm.
	HintLbl (see page 6558)	This is HintLbl, a member of class TCustomizeFrm.
	InfoLbl (see page 6558)	This is InfoLbl, a member of class TCustomizeFrm.
	Label1 (see page 6558)	This is Label1, a member of class TCustomizeFrm.
	Label2 (see page 6558)	This is Label2, a member of class TCustomizeFrm.
	Label4 (see page 6559)	This is Label4, a member of class TCustomizeFrm.
	LargeIconsActn (see page 6559)	This is LargeIconsActn, a member of class TCustomizeFrm.
	LargeIconsChk (see page 6559)	This is LargeIconsChk, a member of class TCustomizeFrm.
	ListCombo (see page 6559)	This is ListCombo, a member of class TCustomizeFrm.
	ListPanel (see page 6560)	This is ListPanel, a member of class TCustomizeFrm.
	MenuAnimationStyles (see page 6560)	This is MenuAnimationStyles, a member of class TCustomizeFrm.
	OptionsBevel1 (see page 6560)	This is OptionsBevel1, a member of class TCustomizeFrm.
	OptionsBevel2 (see page 6560)	This is OptionsBevel2, a member of class TCustomizeFrm.
	OptionsTab (see page 6560)	This is OptionsTab, a member of class TCustomizeFrm.
	OtherLbl (see page 6561)	This is OtherLbl, a member of class TCustomizeFrm.
	PersonalizeLbl (see page 6561)	This is PersonalizeLbl, a member of class TCustomizeFrm.
	RecentlyUsedActn (see page 6561)	This is RecentlyUsedActn, a member of class TCustomizeFrm.
	RecentlyUsedChk (see page 6561)	This is RecentlyUsedChk, a member of class TCustomizeFrm.
	ResetActn (see page 6562)	This is ResetActn, a member of class TCustomizeFrm.
	ResetBtn (see page 6562)	This is ResetBtn, a member of class TCustomizeFrm.
	ResetUsageBtn (see page 6562)	This is ResetUsageBtn, a member of class TCustomizeFrm.
	ResetUsageDataActn (see page 6562)	This is ResetUsageDataActn, a member of class TCustomizeFrm.
	SeparatorBtn (see page 6562)	This is SeparatorBtn, a member of class TCustomizeFrm.
	ShortCutTipsChk (see page 6563)	This is ShortCutTipsChk, a member of class TCustomizeFrm.
	ShowHintsActn (see page 6563)	This is ShowHintsActn, a member of class TCustomizeFrm.
	ShowShortCutsInTipsActn (see page 6563)	This is ShowShortCutsInTipsActn, a member of class TCustomizeFrm.
	ShowTipsChk (see page 6563)	This is ShowTipsChk, a member of class TCustomizeFrm.
	Tabs (see page 6564)	This is Tabs, a member of class TCustomizeFrm.
	ToolbarsLbl (see page 6564)	This is ToolbarsLbl, a member of class TCustomizeFrm.

	ToolbarsTab (see page 6564)	This is ToolbarsTab, a member of class TCustomizeFrm.

36.1.2.2.1 TCustomizeFrm.ActionBarList Field

Pascal

```
ActionBarList: TCheckListBox;
```

C++

```
TCheckListBox ActionBarList;
```

Description

This is ActionBarList, a member of class TCustomizeFrm.

36.1.2.2.2 TCustomizeFrm.ActionImages Field

Pascal

```
ActionImages: TImageList;
```

C++

```
TImageList ActionImages;
```

Description

This is ActionImages, a member of class TCustomizeFrm.

36.1.2.2.3 TCustomizeFrm.ActionList1 Field

Pascal

```
ActionList1: TActionList;
```

C++

```
TActionList ActionList1;
```

Description

This is ActionList1, a member of class TCustomizeFrm.

36.1.2.2.4 TCustomizeFrm.ActionsActionsLbl Field

Pascal

```
ActionsActionsLbl: TLabel;
```

C++

```
TLabel ActionsActionsLbl;
```

Description

This is ActionsActionsLbl, a member of class TCustomizeFrm.

36.1.2.2.5 TCustomizeFrm.ActionsCatLbl Field

Pascal

```
ActionsCatLbl: TLabel;
```

C++

```
TLabel ActionsCatLbl;
```

Description

This is ActionsCatLbl, a member of class TCustomizeFrm.

36.1.2.2.6 TCustomizeFrm.ActionsList Field

Pascal

```
ActionsList: TListBox;
```

C++

```
TListBox ActionsList;
```

Description

This is ActionsList, a member of class TCustomizeFrm.

36.1.2.2.7 TCustomizeFrm.ActionsTab Field

Pascal

```
ActionsTab: TTabSheet;
```

C++

```
TTabSheet ActionsTab;
```

Description

This is ActionsTab, a member of class TCustomizeFrm.

36.1.2.2.8 TCustomizeFrm.ApplyToAllActn Field

Pascal

```
ApplyToAllActn: TAction;
```

C++

```
TAction ApplyToAllActn;
```

Description

This is ApplyToAllActn, a member of class TCustomizeFrm.

36.1.2.2.9 TCustomizeFrm.ApplyToAllChk Field

Pascal

```
ApplyToAllChk: TCheckBox;
```

C++

```
TCheckBox ApplyToAllChk;
```

Description

This is ApplyToAllChk, a member of class TCustomizeFrm.

36.1.2.2.10 TCustomizeFrm.CaptionOptionsCombo Field

Pascal

```
CaptionOptionsCombo: TComboBox;
```

C++

```
TComboBox CaptionOptionsCombo;
```

Description

This is CaptionOptionsCombo, a member of class TCustomizeFrm.

36.1.2.2.11 TCustomizeFrm.CaptionOptionsGrp Field

Pascal

```
CaptionOptionsGrp: TGroupBox;
```

C++

```
TGroupBox CaptionOptionsGrp;
```

Description

This is CaptionOptionsGrp, a member of class TCustomizeFrm.

36.1.2.2.12 TCustomizeFrm.CatList Field

Pascal

```
CatList: TListBox;
```

C++

```
TListBox CatList;
```

Description

This is CatList, a member of class TCustomizeFrm.

36.1.2.2.13 TCustomizeFrm.CloseActn Field

Pascal

```
CloseActn: TAction;
```

C++

```
TAction CloseActn;
```

Description

This is CloseActn, a member of class TCustomizeFrm.

36.1.2.2.14 TCustomizeFrm.CloseBtn Field

Pascal

```
CloseBtn: TButton;
```

C++

```
TButton CloseBtn;
```

Description

This is CloseBtn, a member of class TCustomizeFrm.

36.1.2.2.15 TCustomizeFrm.CloseItem Field

Pascal

```
CloseItem: TMenuItem;
```

C++

```
TMenuItem CloseItem;
```

Description

This is CloseItem, a member of class TCustomizeFrm.

36.1.2.2.16 TCustomizeFrm.CloseMenu Field

Pascal

```
CloseMenu: TPopupMenu;
```

C++

```
TPopupMenu CloseMenu;
```

Description

This is CloseMenu, a member of class TCustomizeFrm.

36.1.2.2.17 TCustomizeFrm.ComboPanel Field

Pascal

```
ComboPanel: TPanel;
```

C++

```
TPanel ComboPanel;
```

Description

This is ComboPanel, a member of class TCustomizeFrm.

36.1.2.2.18 TCustomizeFrm.DescGroupBox Field

Pascal

```
DescGroupBox: TGroupBox;
```

C++

```
TGroupBox DescGroupBox;
```

Description

This is DescGroupBox, a member of class TCustomizeFrm.

36.1.2.2.19 TCustomizeFrm.FullMenusActn Field

Pascal

```
FullMenusActn: TAction;
```

C++

```
TAction FullMenusActn;
```

Description

This is FullMenusActn, a member of class TCustomizeFrm.

36.1.2.2.20 TCustomizeFrm.HintLbl Field

Pascal

```
HintLbl: TLabel;
```

C++

```
TLabel HintLbl;
```

Description

This is HintLbl, a member of class TCustomizeFrm.

36.1.2.2.21 TCustomizeFrm.InfoLbl Field

Pascal

```
InfoLbl: TLabel;
```

C++

```
TLabel InfoLbl;
```

Description

This is InfoLbl, a member of class TCustomizeFrm.

36.1.2.2.22 TCustomizeFrm.Label1 Field

Pascal

```
Label1: TLabel;
```

C++

```
TLabel Label1;
```

Description

This is Label1, a member of class TCustomizeFrm.

36.1.2.2.23 TCustomizeFrm.Label2 Field

Pascal

```
Label2: TLabel;
```

C++

```
TLabel Label2;
```

Description

This is Label2, a member of class TCustomizeFrm.

36.1.2.2.24 TCustomizeFrm.Label4 Field

Pascal

```
Label4: TLabel;
```

C++

```
TLabel Label4;
```

Description

This is Label4, a member of class TCustomizeFrm.

36.1.2.2.25 TCustomizeFrm.LargeIconsActn Field

Pascal

```
LargeIconsActn: TAction;
```

C++

```
TAction LargeIconsActn;
```

Description

This is LargeIconsActn, a member of class TCustomizeFrm.

36.1.2.2.26 TCustomizeFrm.LargeIconsChk Field

Pascal

```
LargeIconsChk: TCheckBox;
```

C++

```
TCheckBox LargeIconsChk;
```

Description

This is LargeIconsChk, a member of class TCustomizeFrm.

36.1.2.2.27 TCustomizeFrm.ListCombo Field

Pascal

```
ListCombo: TComboBox;
```

C++

```
TComboBox ListCombo;
```

Description

This is ListCombo, a member of class TCustomizeFrm.

36.1.2.2.28 TCustomizeFrm.ListPanel Field

Pascal

```
ListPanel: TPanel;
```

C++

```
TPanel ListPanel;
```

Description

This is ListPanel, a member of class TCustomizeFrm.

36.1.2.2.29 TCustomizeFrm.MenuAnimationStyles Field

Pascal

```
MenuAnimationStyles: TComboBox;
```

C++

```
TComboBox MenuAnimationStyles;
```

Description

This is MenuAnimationStyles, a member of class TCustomizeFrm.

36.1.2.2.30 TCustomizeFrm.OptionsBevel1 Field

Pascal

```
OptionsBevel1: TBevel;
```

C++

```
TBevel OptionsBevel1;
```

Description

This is OptionsBevel1, a member of class TCustomizeFrm.

36.1.2.2.31 TCustomizeFrm.OptionsBevel2 Field

Pascal

```
OptionsBevel2: TBevel;
```

C++

```
TBevel OptionsBevel2;
```

Description

This is OptionsBevel2, a member of class TCustomizeFrm.

36.1.2.2.32 TCustomizeFrm.OptionsTab Field

Pascal

```
OptionsTab: TTabSheet;
```

C++

```
TTabSheet OptionsTab;
```

Description

This is OptionsTab, a member of class TCustomizeFrm.

36.1.2.2.33 TCustomizeFrm.OtherLbl Field

Pascal

```
OtherLbl: TLabel;
```

C++

```
TLabel OtherLbl;
```

Description

This is OtherLbl, a member of class TCustomizeFrm.

36.1.2.2.34 TCustomizeFrm.PersonalizeLbl Field

Pascal

```
PersonalizeLbl: TLabel;
```

C++

```
TLabel PersonalizeLbl;
```

Description

This is PersonalizeLbl, a member of class TCustomizeFrm.

36.1.2.2.35 TCustomizeFrm.RecentlyUsedActn Field

Pascal

```
RecentlyUsedActn: TAction;
```

C++

```
TAction RecentlyUsedActn;
```

Description

This is RecentlyUsedActn, a member of class TCustomizeFrm.

36.1.2.2.36 TCustomizeFrm.RecentlyUsedChk Field

Pascal

```
RecentlyUsedChk: TCheckBox;
```

C++

```
TCheckBox RecentlyUsedChk;
```

Description

This is RecentlyUsedChk, a member of class TCustomizeFrm.

36.1.2.2.37 TCustomizeFrm.ResetActn Field

Pascal

```
ResetActn: TAction;
```

C++

```
TAction ResetActn;
```

Description

This is ResetActn, a member of class TCustomizeFrm.

36.1.2.2.38 TCustomizeFrm.ResetBtn Field

Pascal

```
ResetBtn: TButton;
```

C++

```
TButton ResetBtn;
```

Description

This is ResetBtn, a member of class TCustomizeFrm.

36.1.2.2.39 TCustomizeFrm.ResetUsageBtn Field

Pascal

```
ResetUsageBtn: TButton;
```

C++

```
TButton ResetUsageBtn;
```

Description

This is ResetUsageBtn, a member of class TCustomizeFrm.

36.1.2.2.40 TCustomizeFrm.ResetUsageDataActn Field

Pascal

```
ResetUsageDataActn: TAction;
```

C++

```
TAction ResetUsageDataActn;
```

Description

This is ResetUsageDataActn, a member of class TCustomizeFrm.

36.1.2.2.41 TCustomizeFrm.SeparatorBtn Field

Pascal

```
SeparatorBtn: TButton;
```

C++

```
TButton SeparatorBtn;
```

Description

This is SeparatorBtn, a member of class TCustomizeFrm.

36.1.2.2.42 TCustomizeFrm.ShortCutTipsChk Field

Pascal

```
ShortCutTipsChk: TCheckBox;
```

C++

```
TCheckBox ShortCutTipsChk;
```

Description

This is ShortCutTipsChk, a member of class TCustomizeFrm.

36.1.2.2.43 TCustomizeFrm.ShowHintsActn Field

Pascal

```
ShowHintsActn: TAction;
```

C++

```
TAction ShowHintsActn;
```

Description

This is ShowHintsActn, a member of class TCustomizeFrm.

36.1.2.2.44 TCustomizeFrm.ShowShortCutsInTipsActn Field

Pascal

```
ShowShortCutsInTipsActn: TAction;
```

C++

```
TAction ShowShortCutsInTipsActn;
```

Description

This is ShowShortCutsInTipsActn, a member of class TCustomizeFrm.

36.1.2.2.45 TCustomizeFrm.ShowTipsChk Field

Pascal

```
ShowTipsChk: TCheckBox;
```

C++

```
TCheckBox ShowTipsChk;
```

Description

This is ShowTipsChk, a member of class TCustomizeFrm.

36.1.2.2.46 TCustomizeFrm.Tabs Field

Pascal

```
Tabs: TPageControl;
```

C++

```
TPageControl Tabs;
```

Description

This is Tabs, a member of class TCustomizeFrm.

36.1.2.2.47 TCustomizeFrm.ToolbarsLbl Field

Pascal

```
ToolbarsLbl: TLabel;
```

C++

```
TLabel ToolbarsLbl;
```

Description

This is ToolbarsLbl, a member of class TCustomizeFrm.

36.1.2.2.48 TCustomizeFrm.ToolbarsTab Field

Pascal

```
ToolbarsTab: TTabSheet;
```

C++

```
TTabSheet ToolbarsTab;
```

Description

This is ToolbarsTab, a member of class TCustomizeFrm.

36.1.2.3 TCustomizeFrm Methods

The methods of the TCustomizeFrm class are listed here.

Protected Methods

	Name	Description
	AddAction (see page 6567)	This is AddAction, a member of class TCustomizeFrm.
V	ClearCatList (see page 6568)	This is ClearCatList, a member of class TCustomizeFrm.
	CMVisiblechanged (see page 6568)	This is CMVisiblechanged, a member of class TCustomizeFrm.
	FindActionBars (see page 6568)	This is FindActionBars, a member of class TCustomizeFrm.
	IsDupShortCut (see page 6569)	This is IsDupShortCut, a member of class TCustomizeFrm.
V	Notification (see page 6570)	Responds to notifications that objects are about to be inserted or removed.
	SetActiveActionList (see page 6572)	This is SetActiveActionList, a member of class TCustomizeFrm.
V	UpdateDialog (see page 6573)	This is UpdateDialog, a member of class TCustomizeFrm.

Public Methods

	Name	Description
	ActionBarListClick (see page 6565)	This is ActionBarListClick, a member of class TCustomizeFrm.

	ActionBarListClickCheck (see page 6565)	This is ActionBarListClickCheck, a member of class TCustomizeFrm.
	ActionsListClick (see page 6566)	This is ActionsListClick, a member of class TCustomizeFrm.
	ActionsListData (see page 6566)	This is ActionsListData, a member of class TCustomizeFrm.
	ActionsListDrawItem (see page 6566)	This is ActionsListDrawItem, a member of class TCustomizeFrm.
	ActionsListMeasureItem (see page 6566)	This is ActionsListMeasureItem, a member of class TCustomizeFrm.
	ActionsListStartDrag (see page 6566)	This is ActionsListStartDrag, a member of class TCustomizeFrm.
	ApplyToAllActnUpdate (see page 6567)	This is ApplyToAllActnUpdate, a member of class TCustomizeFrm.
	CaptionOptionsComboChange (see page 6567)	This is CaptionOptionsComboChange, a member of class TCustomizeFrm.
	CatListClick (see page 6567)	This is CatListClick, a member of class TCustomizeFrm.
	CatListStartDrag (see page 6568)	This is CatListStartDrag, a member of class TCustomizeFrm.
	CloseBtnClick (see page 6568)	This is CloseBtnClick, a member of class TCustomizeFrm.
	FormClose (see page 6569)	This is FormClose, a member of class TCustomizeFrm.
	FormCreate (see page 6569)	This is FormCreate, a member of class TCustomizeFrm.
	FormResize (see page 6569)	This is FormResize, a member of class TCustomizeFrm.
	LargeIconsActnExecute (see page 6570)	This is LargeIconsActnExecute, a member of class TCustomizeFrm.
	LargeIconsActnUpdate (see page 6570)	This is LargeIconsActnUpdate, a member of class TCustomizeFrm.
	ListComboSelect (see page 6570)	This is ListComboSelect, a member of class TCustomizeFrm.
	MenuAnimationStylesChange (see page 6570)	This is MenuAnimationStylesChange, a member of class TCustomizeFrm.
	RecentlyUsedActnExecute (see page 6571)	This is RecentlyUsedActnExecute, a member of class TCustomizeFrm.
	RecentlyUsedActnUpdate (see page 6571)	This is RecentlyUsedActnUpdate, a member of class TCustomizeFrm.
	ResetActnExecute (see page 6571)	This is ResetActnExecute, a member of class TCustomizeFrm.
	ResetActnUpdate (see page 6572)	This is ResetActnUpdate, a member of class TCustomizeFrm.
	ResetUsageDataActnExecute (see page 6572)	This is ResetUsageDataActnExecute, a member of class TCustomizeFrm.
	SeparatorBtnStartDrag (see page 6572)	This is SeparatorBtnStartDrag, a member of class TCustomizeFrm.
	ShowHintsActnExecute (see page 6572)	This is ShowHintsActnExecute, a member of class TCustomizeFrm.
	ShowHintsActnUpdate (see page 6573)	This is ShowHintsActnUpdate, a member of class TCustomizeFrm.
	ShowShortCutsInTipsActnExecute (see page 6573)	This is ShowShortCutsInTipsActnExecute, a member of class TCustomizeFrm.

36.1.2.3.1 TCustomizeFrm.ActionBarListClick Method

Pascal

```
procedure ActionBarListClick(Sender: TObject);
```

C++

```
__fastcall ActionBarListClick(TObject * Sender);
```

Description

This is ActionBarListClick, a member of class TCustomizeFrm.

36.1.2.3.2 TCustomizeFrm.ActionBarListClickCheck Method

Pascal

```
procedure ActionBarListClickCheck(Sender: TObject);
```

C++

```
__fastcall ActionBarListClickCheck(TObject * Sender);
```

Description

This is ActionBarListClickCheck, a member of class TCustomizeFrm.

36.1.2.3.3 TCustomizeFrm.ActionsListClick Method

Pascal

```
procedure ActionsListClick(Sender: TObject);
```

C++

```
__fastcall ActionsListClick(TObject * Sender);
```

Description

This is ActionsListClick, a member of class TCustomizeFrm.

36.1.2.3.4 TCustomizeFrm.ActionsListData Method

Pascal

```
procedure ActionsListData(Control: TWinControl; Index: Integer; var Data: string);
```

C++

```
__fastcall ActionsListData(TWinControl * Control, int Index, AnsiString Data);
```

Description

This is ActionsListData, a member of class TCustomizeFrm.

36.1.2.3.5 TCustomizeFrm.ActionsListDrawItem Method

Pascal

```
procedure ActionsListDrawItem(Control: TWinControl; Index: Integer; Rect: TRect; State:
TOwnerDrawState);
```

C++

```
__fastcall ActionsListDrawItem(TWinControl * Control, int Index, TRect Rect, TOwnerDrawState
State);
```

Description

This is ActionsListDrawItem, a member of class TCustomizeFrm.

36.1.2.3.6 TCustomizeFrm.ActionsListMeasureItem Method

Pascal

```
procedure ActionsListMeasureItem(Control: TWinControl; Index: Integer; var Height: Integer);
```

C++

```
__fastcall ActionsListMeasureItem(TWinControl * Control, int Index, int Height);
```

Description

This is ActionsListMeasureItem, a member of class TCustomizeFrm.

36.1.2.3.7 TCustomizeFrm.ActionsListStartDrag Method

Pascal

```
procedure ActionsListStartDrag(Sender: TObject; var DragObject: TDragObject);
```

C++

```
__fastcall ActionsListStartDrag(TObject * Sender, TDragObject DragObject);
```

Description

This is ActionsListStartDrag, a member of class TCustomizeFrm.

36.1.2.3.8 TCustomizeFrm.AddAction Method

Pascal

```
function AddAction(AnAction: TContainedAction): string;
```

C++

```
__fastcall AnsiString AddAction(TContainedAction AnAction);
```

Description

This is AddAction, a member of class TCustomizeFrm.

36.1.2.3.9 TCustomizeFrm.ApplyToAllActnUpdate Method

Pascal

```
procedure ApplyToAllActnUpdate(Sender: TObject);
```

C++

```
__fastcall ApplyToAllActnUpdate(TObject * Sender);
```

Description

This is ApplyToAllActnUpdate, a member of class TCustomizeFrm.

36.1.2.3.10 TCustomizeFrm.CaptionOptionsComboChange Method

Pascal

```
procedure CaptionOptionsComboChange(Sender: TObject);
```

C++

```
__fastcall CaptionOptionsComboChange(TObject * Sender);
```

Description

This is CaptionOptionsComboChange, a member of class TCustomizeFrm.

36.1.2.3.11 TCustomizeFrm.CatListClick Method

Pascal

```
procedure CatListClick(Sender: TObject);
```

C++

```
__fastcall CatListClick(TObject * Sender);
```

Description

This is CatListClick, a member of class TCustomizeFrm.

36.1.2.3.12 TCustomizeFrm.CatListStartDrag Method

Pascal

```
procedure CatListStartDrag(Sender: TObject; var DragObject: TDragObject);
```

C++

```
__fastcall CatListStartDrag(TObject * Sender, TDragObject DragObject);
```

Description

This is CatListStartDrag, a member of class TCustomizeFrm.

36.1.2.3.13 TCustomizeFrm.ClearCatList Method

Pascal

```
procedure ClearCatList; virtual;
```

C++

```
virtual __fastcall ClearCatList();
```

Description

This is ClearCatList, a member of class TCustomizeFrm.

36.1.2.3.14 TCustomizeFrm.CloseBtnClick Method

Pascal

```
procedure CloseBtnClick(Sender: TObject);
```

C++

```
__fastcall CloseBtnClick(TObject * Sender);
```

Description

This is CloseBtnClick, a member of class TCustomizeFrm.

36.1.2.3.15 TCustomizeFrm.CMVisiblechanged Method

Pascal

```
procedure CMVisiblechanged(var Message: TMessage); message CM_VISIBLECHANGED;
```

C++

```
__fastcall CMVisiblechanged(TMessage Message);
```

Description

This is CMVisiblechanged, a member of class TCustomizeFrm.

36.1.2.3.16 TCustomizeFrm.FindActionBars Method

Pascal

```
procedure FindActionBars;
```

C++

```
__fastcall FindActionBars();
```

Description

This is FindActionBars, a member of class TCustomizeFrm.

36.1.2.3.17 TCustomizeFrm.FormClose Method

Pascal

```
procedure FormClose(Sender: TObject; var Action: TCloseAction);
```

C++

```
__fastcall FormClose(TObject * Sender, TCloseAction Action);
```

Description

This is FormClose, a member of class TCustomizeFrm.

36.1.2.3.18 TCustomizeFrm.FormCreate Method

Pascal

```
procedure FormCreate(Sender: TObject);
```

C++

```
__fastcall FormCreate(TObject * Sender);
```

Description

This is FormCreate, a member of class TCustomizeFrm.

36.1.2.3.19 TCustomizeFrm.FormResize Method

Pascal

```
procedure FormResize(Sender: TObject);
```

C++

```
__fastcall FormResize(TObject * Sender);
```

Description

This is FormResize, a member of class TCustomizeFrm.

36.1.2.3.20 TCustomizeFrm.IsDupShortCut Method

Pascal

```
function IsDupShortCut(AShortCut: TShortCut; var Action: TContainedAction): Boolean;
```

C++

```
__fastcall Boolean IsDupShortCut(TShortCut AShortCut, TContainedAction Action);
```

Description

This is IsDupShortCut, a member of class TCustomizeFrm.

36.1.2.3.21 TCustomizeFrm.LargeIconsActnExecute Method

Pascal

```
procedure LargeIconsActnExecute(Sender: TObject);
```

C++

```
__fastcall LargeIconsActnExecute(TObject * Sender);
```

Description

This is LargeIconsActnExecute, a member of class TCustomizeFrm.

36.1.2.3.22 TCustomizeFrm.LargeIconsActnUpdate Method

Pascal

```
procedure LargeIconsActnUpdate(Sender: TObject);
```

C++

```
__fastcall LargeIconsActnUpdate(TObject * Sender);
```

Description

This is LargeIconsActnUpdate, a member of class TCustomizeFrm.

36.1.2.3.23 TCustomizeFrm.ListComboSelect Method

Pascal

```
procedure ListComboSelect(Sender: TObject);
```

C++

```
__fastcall ListComboSelect(TObject * Sender);
```

Description

This is ListComboSelect, a member of class TCustomizeFrm.

36.1.2.3.24 TCustomizeFrm.MenuAnimationStylesChange Method

Pascal

```
procedure MenuAnimationStylesChange(Sender: TObject);
```

C++

```
__fastcall MenuAnimationStylesChange(TObject * Sender);
```

Description

This is MenuAnimationStylesChange, a member of class TCustomizeFrm.

36.1.2.3.25 TCustomizeFrm.Notification Method

Responds to notifications that objects are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, forms pass along the notification to their owned components, if any. In addition, Notification makes internal adjustments if the main menu is added or deleted.

A form can, if needed, override Notification to make other adjustments when components are inserted or removed. In particular, if a form has object fields or properties that contains references to other components, it might check the notifications of component removals and invalidate those references as needed.

See Also

InsertComponent (see page 2780)

36.1.2.3.26 TCustomizeFrm.RecentlyUsedActnExecute Method

Pascal

```
procedure RecentlyUsedActnExecute(Sender: TObject);
```

C++

```
__fastcall RecentlyUsedActnExecute(TObject * Sender);
```

Description

This is RecentlyUsedActnExecute, a member of class TCustomizeFrm.

36.1.2.3.27 TCustomizeFrm.RecentlyUsedActnUpdate Method

Pascal

```
procedure RecentlyUsedActnUpdate(Sender: TObject);
```

C++

```
__fastcall RecentlyUsedActnUpdate(TObject * Sender);
```

Description

This is RecentlyUsedActnUpdate, a member of class TCustomizeFrm.

36.1.2.3.28 TCustomizeFrm.ResetActnExecute Method

Pascal

```
procedure ResetActnExecute(Sender: TObject);
```

C++

```
__fastcall ResetActnExecute(TObject * Sender);
```

Description

This is ResetActnExecute, a member of class TCustomizeFrm.

36.1.2.3.29 TCustomizeFrm.ResetActnUpdate Method

Pascal

```
procedure ResetActnUpdate(Sender: TObject);
```

C++

```
__fastcall ResetActnUpdate(TObject * Sender);
```

Description

This is ResetActnUpdate, a member of class TCustomizeFrm.

36.1.2.3.30 TCustomizeFrm.ResetUsageDataActnExecute Method

Pascal

```
procedure ResetUsageDataActnExecute(Sender: TObject);
```

C++

```
__fastcall ResetUsageDataActnExecute(TObject * Sender);
```

Description

This is ResetUsageDataActnExecute, a member of class TCustomizeFrm.

36.1.2.3.31 TCustomizeFrm.SeparatorBtnStartDrag Method

Pascal

```
procedure SeparatorBtnStartDrag(Sender: TObject; var DragObject: TDragObject);
```

C++

```
__fastcall SeparatorBtnStartDrag(TObject * Sender, TDragObject DragObject);
```

Description

This is SeparatorBtnStartDrag, a member of class TCustomizeFrm.

36.1.2.3.32 TCustomizeFrm.SetActiveActionList Method

Pascal

```
procedure SetActiveActionList(const Value: TCustomActionList);
```

C++

```
__fastcall SetActiveActionList(const TCustomActionList Value);
```

Description

This is SetActiveActionList, a member of class TCustomizeFrm.

36.1.2.3.33 TCustomizeFrm.ShowHintsActnExecute Method

Pascal

```
procedure ShowHintsActnExecute(Sender: TObject);
```

C++

```
__fastcall ShowHintsActnExecute(TObject * Sender);
```

Description

This is ShowHintsActnExecute, a member of class TCustomizeFrm.

36.1.2.3.34 TCustomizeFrm.ShowHintsActnUpdate Method

Pascal

```
procedure ShowHintsActnUpdate(Sender: TObject);
```

C++

```
__fastcall ShowHintsActnUpdate(TObject * Sender);
```

Description

This is ShowHintsActnUpdate, a member of class TCustomizeFrm.

36.1.2.3.35 TCustomizeFrm.ShowShortCutsInTipsActnExecute Method

Pascal

```
procedure ShowShortCutsInTipsActnExecute(Sender: TObject);
```

C++

```
__fastcall ShowShortCutsInTipsActnExecute(TObject * Sender);
```

Description

This is ShowShortCutsInTipsActnExecute, a member of class TCustomizeFrm.

36.1.2.3.36 TCustomizeFrm.UpdateDialog Method

Pascal

```
procedure UpdateDialog; virtual;
```

C++

```
virtual __fastcall UpdateDialog();
```

Description

This is UpdateDialog, a member of class TCustomizeFrm.

36.1.2.4 TCustomizeFrm Properties

The properties of the TCustomizeFrm class are listed here.

Public Properties

	Name	Description
	ActionManager (see page 6573)	This is ActionManager, a member of class TCustomizeFrm.
	ActiveActionList (see page 6574)	This is ActiveActionList, a member of class TCustomizeFrm.

36.1.2.4.1 TCustomizeFrm.ActionManager Property

Pascal

```
property ActionManager: TCustomActionManager;
```

C++

```
__property TCustomActionManager ActionManager;
```

Description

This is ActionManager, a member of class TCustomizeFrm.

36.1.2.4.2 TCustomizeFrm.ActiveActionList Property

Pascal

```
property ActiveActionList: TCustomActionList;
```

C++

```
__property TCustomActionList ActiveActionList;
```

Description

This is ActiveActionList, a member of class TCustomizeFrm.

37 DataBkr Namespace

This is namespace DataBkr.

Topics

Name	Description
Classes (see page 6576)	The following table lists classes in this documentation.
Functions (see page 6600)	The following table lists functions in this documentation.

Classes

	Name	Description
	TCRemoteDataModule (see page 6576)	TCRemoteDataModule maintains a list of available providers in an application server.
	TRemoteDataModule (see page 6585)	TRemoteDataModule encapsulates the objects and interfaces of an application server in a multi-tiered database application.

Functions

	Name	Description
	DisableSocketTransport (see page 6600)	Unregisters a remote data module so clients can't access it using a Socket connection.
	DisableWebTransport (see page 6601)	Unregisters a remote data module so clients can't access it using a Web connection.
	EnableSocketTransport (see page 6601)	Registers a remote data module so clients can access it using a Socket connection.
	EnableWebTransport (see page 6602)	Registers a remote data module so clients can access it using a Web connection.
	RegisterPooled (see page 6602)	Registers a remote data module so that instances are pooled.
	UnregisterPooled (see page 6603)	Removes the object pooling registry entries for a remote data module.

37.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCRemoteDataModule (see page 6576)	TCRemoteDataModule maintains a list of available providers in an application server.
	TRemoteDataModule (see page 6585)	TRemoteDataModule encapsulates the objects and interfaces of an application server in a multi-tiered database application.

37.1.1 TCRemoteDataModule Class

TCRemoteDataModule maintains a list of available providers in an application server.

Class Hierarchy

Pascal

```
TCRemoteDataModule = class(TDataModule);
```

C++

```
class TCRemoteDataModule : public TDataModule;
```

File

DataBkr

Description

TCRemoteDataModule is the base class for remote data modules that appear in the application servers of multi-tiered database applications that are written in C++. It manages a list of provider components, which handle providing data to client applications and applying updates received from those clients.

Remote data modules act as the central repository for all objects in the application server. They can contain any nonvisual components. Typically these are data access components, such as datasets and the provider components that handle communication between these datasets and client applications. Remote data modules can also contain other nonvisual components, such as TTimer, or TImageList.

At design time a remote data module provides a visual container into which a developer can place nonvisual components, set their properties, and write event handlers for them. In the unit file for the remote data module a developer can also code any business rules that are to be centralized in the middle tier of a multi-tiered application.

To create a remote data module at design time, choose File|New, and select Remote Data Module from the multi-tiered page of the new items dialog. The Remote Data Module Wizard then creates a new descendant of TCRemoteDataModule that supports an interface that descends from IAppServer.

The interface itself is handled by a companion implementation object. This object descends from REMOTEDATAMODULE_IMPL(), which is a macro defined in Atlvcl.h. The implementation object uses the ATL to provide

Automation support for the IAppServer interface, which client applications use to communicate with the providers in the remote data module.

Note: TCRemoteDataModule does not provide support for the IAppServer interface. This support is introduced by the TRemoteDataModule (see page 6585) class, a descendant of TCRemoteDataModule.

See Also

IAppServer (see page 16450)

TDataSetProvider (see page 18508)

RemoteDataModule

37.1.1.1 TCRemoteDataModule Members

The following tables list the members exposed by TCRemoteDataModule.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
V	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.

	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 2829)	Generates an OnCreate (see page 2833) event.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 2830)	Generates an OnDestroy (see page 2834) event.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CRDMGetProviderNames (see page 6581)	Returns a list of all the available providers on the remote data module.
	Create (see page 6580)	The Create method creates a remote data module instance.
	CreateNew (see page 2828)	Creates an instance of a data module and registers with the global screen object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 6580)	Frees a remote data module instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetProvider (see page 6581)	Returns a registered provider component given its name.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	Lock (see page 6582)	Locks the remote data module so that it can't be locked by other threads.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
V	RegisterProvider (see page 6582)	Adds a provider to the list returned by CRDMGetProviderNames (see page 6581).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	UnLock (see page 6583)	This is UnLock, a member of class TCRemoteDataModule.
V	UnRegisterProvider (see page 6583)	Removes a provider from the list returned by AS_GetProviderNames.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	ProviderCount (see page 6584)	Indicates the number of providers registered with the remote data module.
R	Providers (see page 6584)	Lists the provider components that are contained in the remote data module.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.
	Tag (see page 2808)	Stores an integer value as part of a component.

37.1.1.2 TCRemoteDataModule.Create Constructor

The Create method creates a remote data module instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCRemoteDataModule(TComponent * AOwner);
```

Description

Do not use the contstructor to instantiate TCRemoteDataModule (see page 6576). Instead, add a remote data module to a server application using the remote data module wizard on the multi-tier page of the new items dialog.

AOwner specifies the component that is responsible for freeing the remote data module (typically the application).

See Also

Create (see page 2827)

37.1.1.3 TCRemoteDataModule.Destroy Destructor

Frees a remote data module instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCRemoteDataModule();
```

Description

Most applications do not need to call the destructor for TCRemoteDataModule (see page 6576). The remote data module is freed by its owner when the application shuts down.

If you must destroy a remote data module at runtime,

37.1.1.4 TCRemoteDataModule Methods

The methods of the TCRemoteDataModule class are listed here.

Public Methods

	Name	Description
	CRDMGetProviderNames (see page 6581)	Returns a list of all the available providers on the remote data module.
V	GetProvider (see page 6581)	Returns a registered provider component given its name.
V	Lock (see page 6582)	Locks the remote data module so that it can't be locked by other threads.
V	RegisterProvider (see page 6582)	Adds a provider to the list returned by CRDMGetProviderNames (see page 6581).
V	UnLock (see page 6583)	This is UnLock, a member of class TCRemoteDataModule.
V	UnRegisterProvider (see page 6583)	Removes a provider from the list returned by AS_GetProviderNames.

37.1.1.4.1 TCRemoteDataModule.CRDMGetProviderNames Method

Returns a list of all the available providers on the remote data module.

Pascal

```
function CRDMGetProviderNames: OleVariant;
```

C++

```
__fastcall OleVariant CRDMGetProviderNames();
```

Description

Call CRDMGetProviderNames for a list of all providers available from a remote data module. These values indicate the possible values for the ProviderName property of a client dataset that connects to the remote data module. When calling any of the remote data modules through the IAppServer interface, pass one of these names to indicate the provider that is the target of the method call.

CRDMGetProviderNames returns the names of the providers in a Variant array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

CRDMGetProviderNames provides the underlying implementation for the IAppServer method AS_GetProviderNames.

Warning: CRDMGetProviderNames returns a list that can include providers that are not exported. If a provider in the list has its Exported property set to false, trying to use that provider over an IAppServer interface results in an exception.

See Also

ProviderName (see page 8492)

GetProvider (see page 6581)

IAppServer (see page 16450)

Exported (see page 18474)

RegisterProvider (see page 6582)

37.1.1.4.2 TCRemoteDataModule.GetProvider Method

Returns a registered provider component given its name.

Pascal

```
function GetProvider(const ProviderName: string): TCustomProvider; virtual;
```

C++

```
virtual __fastcall TCustomProvider GetProvider(const AnsiString ProviderName);
```

Description

GetProvider returns the reference to a provider given its name. ProviderName specifies the name of a provider that is returned. If ProviderName is not one of the names returned by CRDMGetProviderNames (see page 6581), or if the provider's Exported property is false, GetProvider raises an exception.

See Also

Providers (see page 6584)

Exported (see page 18474)

CRDMGetProviderNames (see page 6581)

RegisterProvider (see page 6582)

37.1.1.4.3 TCRemoteDataModule.Lock Method

Locks the remote data module so that it can't be locked by other threads.

Pascal

```
procedure Lock; virtual;
```

C++

```
virtual __fastcall Lock();
```

Description

TCRemoteDataModule (see page 6576) descendants use Lock internally to ensure that all IAppServer calls are thread-safe. When implementing a remote data module that supports the free-threading model, call Lock before accessing any instance data, including the providers contained in the data module. When finished with the block of code that must be protected from thread conflicts, call the Unlock method to release the lock.

Lock uses a critical section to guard against thread conflicts. This means that any code which accesses the remote data module's properties or contained objects must call Lock or it will introduce thread conflicts.

See Also

Providers (see page 6584)

Unlock (see page 6583)

IAppServer (see page 16450)

TCriticalSection (see page 22058)

37.1.1.4.4 TCRemoteDataModule.RegisterProvider Method

Adds a provider to the list returned by CRDMGetProviderNames (see page 6581).

Pascal

```
procedure RegisterProvider(Value: TCustomProvider); virtual;
```

C++

```
virtual __fastcall RegisterProvider(TCustomProvider Value);
```

Description

Call RegisterProvider to make a provider available through a remote data module's IAppServer interface. Pass a reference to the provider as the Value parameter. Once a provider is registered, its availability to client applications can be turned on or off using its Exported property. However, even the names of unexported providers are returned by CRDMGetProviderNames (see page

6581).

There is no need to call RegisterProvider for providers that are added at design time or when the remote data module is assigned as an Owner in the provider's constructor. These are registered automatically.

Before a registered provider is destroyed, it must be unregistered using the UnRegisterProvider (see page 6583) method. Providers (see page 6584) that have the remote data module as an Owner are unregistered automatically from their destructors.

See Also

UnRegisterProvider (see page 6583)

CRDMGetProviderNames (see page 6581)

IAppServer (see page 16450)

Create (see page 18454)

Exported (see page 18474)

37.1.1.4.5 TCRemoteDataModule.UnLock Method

Pascal

```
procedure UnLock; virtual;
```

C++

```
virtual __fastcall UnLock();
```

Description

This is UnLock, a member of class TCRemoteDataModule.

37.1.1.4.6 TCRemoteDataModule.UnRegisterProvider Method

Removes a provider from the list returned by AS_GetProviderNames.

Pascal

```
procedure UnRegisterProvider(Value: TCustomProvider); virtual;
```

C++

```
virtual __fastcall UnRegisterProvider(TCustomProvider Value);
```

Description

Call UnRegisterProvider to remove a provider from the remote data module's list of available providers. Pass a reference to the provider that should be removed as the Value parameter. When a provider is not registered, it is not returned by the CRDMGetProviderNames (see page 6581) method, which can be accessed through the IAppServer interface.

To temporarily disable a provider, use the provider's Exported property. Unexported providers are listed by CRDMGetProviderNames (see page 6581) but can't be accessed through any IAppServer method calls.

Before a registered provider is destroyed, it must be unregistered using the UnRegisterProvider method. Providers (see page 6584) that have the remote data module as an Owner are unregistered automatically from their destructors.

See Also

RegisterProvider (see page 6582)

CRDMGetProviderNames (see page 6581)

IAppServer (see page 16450)

Destroy (see page 18454)

Exported (see page 18474)

37.1.1.5 TCRemoteDataModule Properties

The properties of the TCRemoteDataModule class are listed here.

Public Properties

	Name	Description
R	ProviderCount (see page 6584)	Indicates the number of providers registered with the remote data module.
R	Providers (see page 6584)	Lists the provider components that are contained in the remote data module.

37.1.1.5.1 TCRemoteDataModule.ProviderCount Property

Indicates the number of providers registered with the remote data module.

Pascal

```
property ProviderCount: integer;
```

C++

```
__property int ProviderCount;
```

Description

ProviderCount is the number of providers listed by the Providers (see page 6584) property.

See Also

Providers (see page 6584)

RegisterProvider (see page 6582)

37.1.1.5.2 TCRemoteDataModule.Providers Property

Lists the provider components that are contained in the remote data module.

Pascal

```
property Providers [const ProviderName:string]: TCustomProvider;
```

C++

```
__property TCustomProvider Providers[AnsiString const ProviderName];
```

Description

Use Providers to access a provider component from the remote data module given its name. The remote data module uses Providers to locate the target of method calls made using the IAppServer interface.

If the value passed as is not one of the names returned by AS_GetProviderNames, or if the provider's Exported property is false, reading Providers raises an exception.

See Also

AS_GetProviderNames (see page 16454)

Exported (see page 18474)

ProviderCount (see page 6584)

37.1.2 TRemoteDataModule Class

TRemoteDataModule encapsulates the objects and interfaces of an application server in a multi-tiered database application.

Class Hierarchy

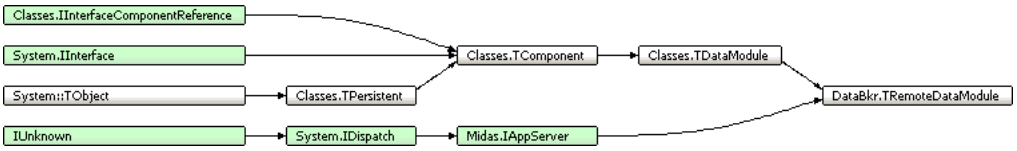

Pascal

```
TRemoteDataModule = class(TDataModule, IAppServer);
```

C++

```
class TRemoteDataModule : public TDataModule, public IAppServer;
```

File

DataBkr

Description

Use a TRemoteDataModule object as the central repository for all objects in an application server that is written in Delphi. Remote data modules can contain any nonvisual components. Typically these are data access components, such as datasets and the provider components that handle communication between these datasets and client applications. Remote data modules can also contain other nonvisual components, such as TTimer, or TImageList.

At design time a TRemoteDataModule object provides a visual container into which a developer can place nonvisual components, set their properties, and write event handlers for them. In the unit file for the remote data module a developer can also code any business rules that are to be centralized in the middle tier of a multi-tiered application.

Remote data modules are dual-interface Automation servers that implement the IAppServer interface.Connection components on client applications are designed to look for this interface.

To create a remote data module at design time, choose File|New|Other, and select Remote Data Module from the multi-tiered page of the new items dialog. If you are using Delphi, the Remote Data Module Wizard then creates a new descendant of TRemoteDataModule that supports an interface that descends from IAppServer.

See Also

IAppServer (see page 16450)

37.1.2.1 TRemoteDataModule Members

The following tables list the members exposed by TRemoteDataModule.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AS_ApplyUpdates (see page 6590)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 6591)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 6591)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 6592)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 6593)	Returns a list of all the available providers on the remote data module.
	AS_GetRecords (see page 6593)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 6594)	Returns information from a specified record of the provider's dataset.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
V	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetProvider (see page 6595)	This is GetProvider, a member of class TRemoteDataModule.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 6597)	Registers the remote data module in the system registry as an automation server.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 2829)	Generates an OnCreate (see page 2833) event.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 2830)	Generates an OnDestroy (see page 2834) event.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 6589)	Creates a remote data module instance.
V	CreateNew (see page 2828)	Creates an instance of a data module and registers with the global screen object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 6589)	Frees a remote data module instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	Lock (see page 6595)	Locks the remote data module so that it can't be locked by other threads.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
V	RegisterProvider (see page 6596)	Adds a provider to the list returned by AS_GetProviderNames (see page 6593).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Unlock (see page 6596)	Unlocks the remote data module after it was locked by a call to the Lock (see page 6595) method.
V	UnRegisterProvider (see page 6597)	Removes a provider from the list returned by AS_GetProviderNames (see page 6593).
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	ProviderCount (see page 6599)	Indicates the number of providers registered with the remote data module.
R	Providers (see page 6599)	This is Providers, a member of class TRemoteDataModule.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.

	Tag (see page 2808)	Stores an integer value as part of a component.

37.1.2.2 TRemoteDataModule.Create Constructor

Creates a remote data module instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TRemoteDataModule(TComponent * AOwner);
```

Description

Most applications do not need to call the constructor for TRemoteDataModule (see page 6585). Instead, add a remote data module to a server application using the remote data module wizard on the multi-tier page of the new items dialog.

AOwner specifies the component that is responsible for freeing the remote data module (typically the application).

See Also

Create (see page 2827)

37.1.2.3 TRemoteDataModule.Destroy Destructor

Frees a remote data module instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRemoteDataModule();
```

Description

Most applications do not need to call the destructor for TRemoteDataModule (see page 6585). The remote data module is freed by its owner when the application shuts down.

If you must destroy a remote data module at runtime, call the Free method instead.

See Also

Free (see page 22300)

37.1.2.4 TRemoteDataModule Methods

The methods of the TRemoteDataModule class are listed here.

Protected Methods

	Name	Description
	AS_ApplyUpdates (see page 6590)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 6591)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 6591)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 6592)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 6593)	Returns a list of all the available providers on the remote data module.
	AS_GetRecords (see page 6593)	Returns a data packet that contains the specified data.

	AS_RowRequest (see page 6594)	Returns information from a specified record of the provider's dataset.
	GetProvider (see page 6595)	This is GetProvider, a member of class TRemoteDataModule.
	UpdateRegistry (see page 6597)	Registers the remote data module in the system registry as an automation server.

Public Methods

	Name	Description
	Lock (see page 6595)	Locks the remote data module so that it can't be locked by other threads.
	RegisterProvider (see page 6596)	Adds a provider to the list returned by AS_GetProviderNames (see page 6593).
	Unlock (see page 6596)	Unlocks the remote data module after it was locked by a call to the Lock (see page 6595) method.
	UnRegisterProvider (see page 6597)	Removes a provider from the list returned by AS_GetProviderNames (see page 6593).

37.1.2.4.1 TRemoteDataModule.AS_ApplyUpdates Method

Applies updates received from a client dataset using a specified provider.

Pascal

```
function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_ApplyUpdates(const BSTR ProviderName, OleVariant Delta, int
MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

AS_AS_ApplyUpdates writes client updates to a dataset or database server. ProviderName specifies the name of the provider component that manages the update operation. Delta is a Variant containing the updated, inserted, and deleted records to write to the database. MaxErrors indicates the maximum number of errors to permit before rolling back the update operation; ErrorCount returns the actual number of errors encountered during the update operation. OwnerData is information supplied by the client application in a BeforeAS_ApplyUpdates event handler and returned to the client application in an AfterAS_ApplyUpdates event handler. It is passed to the provider's BeforeAS_ApplyUpdates event handler and returned from the provider's AfterAS_ApplyUpdates event handler.

AS_AS_ApplyUpdates The AS_ApplyUpdates_result parameter returns a Variant that is a data packet containing all records that could not be applied.

Note: Applications can only call the protected AS_AS_ApplyUpdates method using the IAppServer interface. To apply updates directly in code from the application server, use the provider component's AS_ApplyUpdates method.

See Also

ApplyUpdates (see page 8394)

applyupdates (see page 18459)

BeforeApplyUpdates (see page 18479)

AfterApplyUpdates (see page 18475)

BeforeApplyUpdates (see page 8511)

AfterApplyUpdates (see page 8497)

providers (see page 6599)

IAppServer (see page 16450)

37.1.2.4.2 TRemoteDataModule.AS_DataRequest Method

Generates an OnAS_DataRequest event on the specified provider component.

Pascal

```
function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

Use AS_AS_DataRequest to invoke an OnAS_DataRequest event handler of the provider named by ProviderName. This method is provided to allow application developers to customize the communication between a client application and a provider component. There is no predefined meaning for the Data parameter or the return value AS_DataRequest_result parameter, which is returned to the caller.

Note: Applications can only call the protected AS_AS_DataRequest method using the IAppServer interface.

See Also

OnDataRequest (see page 18482)

providers (see page 6599)

DataRequest (see page 8404)

IAppServer (see page 16450)

37.1.2.4.3 TRemoteDataModule.AS_Execute Method

AS_Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); safecall;
```

C++

```
__fastcall AS_Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant Params,
OleVariant OwnerData);
```

Description

Use AS_AS_Execute to pass an execute command to a remote provider, using a specified set of parameters. AS_AS_Execute calls the AS_Execute method of the specified provider after assigning any parameters. Output parameters that result from executing a query or stored procedure are returned in the Params parameter.

The provider's AS_Execute method generates a BeforeAS_Execute event, passes the command on to its associated dataset, and generates an AfterAS_Execute event. The dataset must represent a query or stored procedure that does not return a cursor.

The ProviderName parameter specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the query, or the name of a stored procedure that replaces the dataset's stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

The Params parameter encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams routine. To decode Params into a TParams object, use the Unpackparams routine.

The OwnerData parameter contains custom information that appears as an argument to the provider's BeforeAS_Execute and AfterAS_Execute event handlers. This information originates in a client dataset's BeforeAS_Execute event handler. The value returned in OwnerData is passed to the client dataset's AfterAS_Execute event handler.

See Also

DataSet (see page 18529)

beforeexecute (see page 18479)

AfterExecute (see page 18476)

Params (see page 8489)

Execute (see page 8413)

Execute (see page 18466)

BeforeExecute (see page 8514)

AfterExecute (see page 8504)

Options (see page 18440)

PackageParams (see page 8534)

UnpackParams (see page 8534)

Communicating with the Client Dataset

37.1.2.4.4 TRemoteDataModule.AS_GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

Use AS_AS_GetParams to retrieve the current parameter values from the dataset associated with a provider.

The ProviderName parameter specifies the name of the provider component whose dataset has the parameters.

The OwnerData represents custom information that originates in a client dataset's BeforeAS_GetParams event handler and returns information that is passed to the client dataset's AfterAS_GetParams event handler.

AS_GetParams_result (C++) or AS_AS_GetParams (Delphi) returns the parameters, encoded as a Variant array, where each element is a 2-element Variant array containing a parameter name and a parameter value. These parameters can be converted to a TParams object using the UnpackParams procedure. If the dataset has no parameters, or if the provider does not support parameter fetching, AS_GetParams_result (C++) or AS_AS_GetParams (Delphi) returns a Null Variant.

Note: AS_AS_GetParams should not be used to retrieve output parameters from stateless application servers, because parameter values may be changed by other applications. When writing a stateless application server, retrieve parameter values using the AS_Execute method instead.

See Also

DataSet (see page 18529)

Params (see page 8489)

FetchParams (see page 8415)

GetParams (see page 18467)

BeforeGetParams (see page 18480)

aftergetparams (see page 18477)

AS_Execute (see page 16453)

UnpackParams (see page 8534)

Communicating with the Client Dataset

37.1.2.4.5 TRemoteDataModule.AS_GetProviderNames Method

Returns a list of all the available providers on the remote data module.

Pascal

```
function AS_GetProviderNames: OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetProviderNames();
```

Description

Call AS_AS_GetProviderNames for a list of all registered providers available from a remote data module. These values indicate the possible values for the ProviderName property of a client dataset that connects to the remote data module. When calling any of the remote data modules through the IAppServer interface, pass one of these names to indicate the provider that is the target of the method call.

AS_AS_GetProviderNames returns the names of the providers in a Variant array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

Note: Applications can only call the protected AS_AS_GetProviderNames method using the IAppServer interface.

See Also

ProviderName (see page 8492)

providers (see page 6599)

IAppServer (see page 16450)

Exported (see page 18474)

37.1.2.4.6 TRemoteDataModule.AS_GetRecords Method

Returns a data packet that contains the specified data.

Pascal

```
function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetRecords(const BSTR ProviderName, int Count, int RecsOut, int
Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

Call AS_AS_GetRecords to retrieve a specified number of records. AS_AS_GetRecords returns the requested records, starting

with the current record of the provider's dataset. When working with a stateless remote data module, you may need to reposition the cursor or re-execute a query or stored procedure in the provider's BeforeAS_GetRecords event handler.

ProviderName gives the name of the provider component that provides the records.

Count indicates the number or type of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only Count records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants. For example, the following value requests metadata along with a data packet encoded in XML:

```
MetaDataOption + XMLOption
MetaDataOption || XMLOption
```

CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a table or stored procedure that replaces the provider's underlying database table or stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

Params is a Variant containing any parameters that should be passed to the provider's dataset before it executes to generate the requested data. It returns any output parameters.

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_GetRecords event handler. This information is passed to the provider's BeforeAS_GetRecords event handler. OwnerData returns information supplied by the provider's AfterAS_GetRecords event handler.

Records are returned as a data packet in a Variant parameter, AS_GetRecords_result.

Note: Applications can only call the protected AS_AS_GetRecords method using the IAppServer interface. To fetch records directly in code from the application server, use the provider component's AS_GetRecords method.

See Also

Data (see page 8467)

Params (see page 8489)

GetNextPacket (see page 8423)

GetRecords (see page 18468)

BeforeGetRecords (see page 18480)

AfterGetRecords (see page 18477)

providers (see page 6599)

IAppServer (see page 16450)

37.1.2.4.7 TRemoteDataModule.AS_RowRequest Method

Returns information from a specified record of the provider's dataset.

Pascal

```
function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_RowRequest(const BSTR ProviderName, OleVariant Row, int RequestType,
OleVariant OwnerData);
```

Description

Client datasets call AS_AS_RowRequest automatically to implement their FetchBlobs, FetchDetails , or RefreshRecord method.

The ProviderName parameter indicates the provider associated with the dataset from which information should be fetched.

The Row parameter is an OleVariant that describes the current record on the client dataset.

The RequestType parameter indicates the type of information required. It is an integer version of the TFetchOptions type. (An integer because the value may be marshaled to a remote application server). To create a value for RequestType, take the corresponding TFetchOptions value, cast it to a Byte, and cast the result to an Integer:

```
Integer(Byte([foBlobs, foDetails]));
(int) (Byte) (TFetchOptions() << foBlobs << foDetails);
```

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_RowRequest event handler. This information is passed to the provider's BeforeAS_RowRequest event handler. OwnerData returns custom information supplied by the provider's AfterAS_RowRequest event handler.

The requested data is returned as a delta packet.

Note: Applications can only call the protected AS_AS_RowRequest method using the IAppServer interface.

See Also

RowRequest (see page 18472)

BeforeRowRequest (see page 18482)

AfterRowRequest (see page 18478)

FetchBlobs (see page 8414)

FetchDetails (see page 8414)

RefreshRecord (see page 8442)

providers (see page 6599)

IAppServer (see page 16450)

37.1.2.4.8 TRemoteDataModule.GetProvider Method

Pascal

```
function GetProvider(const ProviderName: string): TCustomProvider; virtual;
```

C++

```
virtual __fastcall TCustomProvider GetProvider(const AnsiString ProviderName);
```

Description

This is GetProvider, a member of class TRemoteDataModule.

37.1.2.4.9 TRemoteDataModule.Lock Method

Locks the remote data module so that it can't be locked by other threads.

Pascal

```
procedure Lock; virtual;
```

C++

```
virtual __fastcall Lock();
```

Description

Remote data modules use Lock internally to ensure that all IAppServer calls are thread-safe. When implementing a remote data module descendant that supports the free-threading model, call Lock before accessing any instance data, including the providers contained in the data module. When finished with the block of code that must be protected from thread conflicts, call the Unlock (see page 6596) method to release the lock.

Lock uses a critical section to guard against thread conflicts. This means that any code which accesses the remote data module's properties or contained objects must call Lock or it will introduce thread conflicts.

See Also

providers (see page 6599)

Unlock (see page 6596)

IAppServer (see page 16450)

TCriticalSection (see page 22058)

37.1.2.4.10 TRemoteDataModule.RegisterProvider Method

Adds a provider to the list returned by AS_GetProviderNames (see page 6593).

Pascal

```
procedure RegisterProvider(Value: TCustomProvider); virtual;
```

C++

```
virtual __fastcall RegisterProvider(TCustomProvider Value);
```

Description

Call RegisterProvider to make a provider available through the remote data module's IAppServer interface. Pass a reference to the provider as the Value parameter. Once a provider is registered, its availability to client applications can be turned on or off using its Exported property.

There is no need to call RegisterProvider for providers that are added at design time or when the remote data module is assigned as an Owner in the provider's constructor. These are registered automatically.

Before a registered provider is destroyed, it must be unregistered using the UnRegisterProvider (see page 6597) method. Providers (see page 6599) that have the remote data module as an Owner are unregistered automatically from their destructors.

See Also

UnRegisterProvider (see page 6597)

AS_GetProviderNames (see page 6593)

IAppServer (see page 16450)

Create (see page 18454)

Exported (see page 18474)

37.1.2.4.11 TRemoteDataModule.Unlock Method

Unlocks the remote data module after it was locked by a call to the Lock (see page 6595) method.

Pascal

```
procedure Unlock; virtual;
```

C++

```
virtual __fastcall Unlock();
```

Description

Use Unlock to release the critical section that was entered by a call to the Lock (see page 6595) method. Every call to the Lock (see page 6595) method must be paired with a corresponding call to Unlock.

See Also

Lock (see page 6595)

37.1.2.4.12 TRemoteDataModule.UnRegisterProvider Method

Removes a provider from the list returned by AS_GetProviderNames (see page 6593).

Pascal

```
procedure UnRegisterProvider(Value: TCustomProvider); virtual;
```

C++

```
virtual __fastcall UnRegisterProvider(TCustomProvider Value);
```

Description

Call UnRegisterProvider to remove a provider from the remote data module's list of available providers. Pass a reference to the provider that should be removed as the Value parameter. When a provider is not registered, it is not returned by the AS_GetProviderNames (see page 6593) method, which can be accessed through the IAppServer interface.

To temporarily disable a provider, use the provider's Exported property. Unexported providers can't be accessed through any IAppServer method calls.

Before a registered provider is destroyed, it must be unregistered using the UnRegisterProvider method. Providers (see page 6599) that have the remote data module as an Owner are unregistered automatically from their destructors.

See Also

RegisterProvider (see page 6596)

AS_GetProviderNames (see page 6593)

IAppServer (see page 16450)

Destroy (see page 18454)

Exported (see page 18474)

37.1.2.4.13 TRemoteDataModule.UpdateRegistry Method

Registers the remote data module in the system registry as an automation server.

Pascal

```
class procedure UpdateRegistry(Register: Boolean; const ClassID: string; const ProgID:
string); override;
```

C++

```
virtual __fastcall UpdateRegistry(Boolean Register, const AnsiString ClassID, const AnsiString
ProgID);
```

Description

The protected UpdateRegistry is called automatically when you register or unregister the application server as an automation server.

The Register parameter is true when registering an application server, false when unregistering.

The ClassID and ProgID parameters specify the server's ClassID and ProgID respectively.

Note: If your application server pools remote data modules, you must override this method to call the global RegisterPooled (see page 6602) procedure when the Register parameter is true and UnregisterPooled (see page 6603) when the Register parameter is false.

Delphi Examples:

```
{
The following UpdateRegistry method adjusts a pooled remote
data module accessed from a
TWebConnection component. This allows the resources of the
data module to be freed when it is not in use, while
optimizing the use of existing remote data modules by letting
clients share them.  The UpdateRegistry method also comments
out the calls to DisableSocketTransport and EnableSocket
transport that are added by the Remote Data Module Wizard.
This prevents socket-based connections, which do not provide
for a secure connection, from using the remote data module.
}
class procedure TPooledRDM.UpdateRegistry(Register: Boolean; const ClassID, ProgID: string);
begin
  if Register then
  begin
    inherited UpdateRegistry(Register, ClassID, ProgID);
    EnableSocketTransport(ClassID);
    EnableWebTransport(ClassID);
  end else
  begin
    DisableSocketTransport(ClassID);
    DisableWebTransport(ClassID);
    inherited UpdateRegistry(Register, ClassID, ProgID);
  end;
end;
```

See Also

RegisterPooled (see page 6602)

UnregisterPooled (see page 6603)

Pooling Remote Data Modules

37.1.2.5 TRemoteDataModule Properties

The properties of the TRemoteDataModule class are listed here.

Public Properties

	Name	Description
R	ProviderCount (see page 6599)	Indicates the number of providers registered with the remote data module.
R	Providers (see page 6599)	This is Providers, a member of class TRemoteDataModule.

37.1.2.5.1 TRemoteDataModule.ProviderCount Property

Indicates the number of providers registered with the remote data module.

Pascal

```
property ProviderCount: integer;
```

C++

```
__property int ProviderCount;
```

Description

ProviderCount is the number of providers that can be accessed using the Providers (see page 6599) property.

See Also

providers (see page 6599)

RegisterProvider (see page 6596)

37.1.2.5.2 TRemoteDataModule.Providers Property

Pascal

```
property Providers [const ProviderName: string]: TCustomProvider;
```

C++

```
__property TCustomProvider Providers[AnsiString const ProviderName];
```

Description

This is Providers, a member of class TRemoteDataModule.

37.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DisableSocketTransport (see page 6600)	Unregisters a remote data module so clients can't access it using a Socket connection.
	DisableWebTransport (see page 6601)	Unregisters a remote data module so clients can't access it using a Web connection.
	EnableSocketTransport (see page 6601)	Registers a remote data module so clients can access it using a Socket connection.
	EnableWebTransport (see page 6602)	Registers a remote data module so clients can access it using a Web connection.
	RegisterPooled (see page 6602)	Registers a remote data module so that instances are pooled.
	UnregisterPooled (see page 6603)	Removes the object pooling registry entries for a remote data module.

37.2.1 DataBkr.DisableSocketTransport Function

Unregisters a remote data module so clients can't access it using a Socket connection.

Pascal

```
procedure DisableSocketTransport(const ClassID: string);
```

C++

```
DisableSocketTransport(const AnsiString ClassID);
```

File

DataBkr

Description

When working in Delphi, the remote data module Wizard automatically adds a call to DisableSocketTransport to the overridden UpdateRegistry method of the new remote data module class. This removes the registry entries that Scktsrvr.exe uses to check whether clients can access the application server. The registry entries are added by a call to EnableSocketTransport (see page 6601) (also added to UpdateRegistry).

To make an application server unavailable to Socket-based connections, remove these two calls from UpdateRegistry. The application server can still receive requests over connections formed using other protocols.

ClassID is the class ID of the remote data module. This value is passed as a parameter to the UpdateRegistry method that calls DisableSocketTransport.

See Also

EnableSocketTransport (see page 6601)

TRemoteDataModule (see page 6585)

37.2.2 DataBkr.DisableWebTransport Function

Unregisters a remote data module so clients can't access it using a Web connection.

Pascal

```
procedure DisableWebTransport(const ClassID: string);
```

C++

```
DisableWebTransport(const AnsiString ClassID);
```

File

DataBkr

Description

When working in Delphi, the remote data module Wizard automatically adds a call to DisableWebTransport to the overridden UpdateRegistry method of the new remote data module class. This removes the registry entries that httpsrvr.dll uses to check whether clients can access the application server. The registry entries are added by a call to EnableWebTransport (see page 6602) (also added to UpdateRegistry).

To make an application server unavailable to HTTP-based connections, remove these two calls from UpdateRegistry. The application server can still receive requests over connections formed using other protocols.

ClassID is the class ID of the remote data module. This value is passed as a parameter to the UpdateRegistry method that calls DisableSocketTransport (see page 6600).

See Also

EnableWebTransport (see page 6602)

TRemoteDataModule (see page 6585)

37.2.3 DataBkr.EnableSocketTransport Function

Registers a remote data module so clients can access it using a Socket connection.

Pascal

```
procedure EnableSocketTransport(const ClassID: string);
```

C++

```
EnableSocketTransport(const AnsiString ClassID);
```

File

DataBkr

Description

When working in Delphi, the remote data module Wizard automatically adds a call to EnableSocketTransport to the overridden UpdateRegistry method of the new remote data module class. This adds the registry entries that Scktsrvr.exe uses to check whether clients can access the application server. The registry entries are removed by a call to DisableSocketTransport (see page 6600) (also added to UpdateRegistry).

To make an application server unavailable to Socket-based connections, remove these two calls from UpdateRegistry. The

application server can still receive requests over connections formed using other protocols.

ClassID is the class ID of the remote data module. This value is passed as a parameter to the UpdateRegistry method that calls EnableSocketTransport.

See Also

DisableSocketTransport (see page 6600)

TRemoteDataModule (see page 6585)

Using Socket Connections

37.2.4 DataBkr.EnableWebTransport Function

Registers a remote data module so clients can access it using a Web connection.

Pascal

```
procedure EnableWebTransport(const ClassID: string);
```

C++

```
EnableWebTransport(const AnsiString ClassID);
```

File

DataBkr

Description

When working in Delphi, the remote data module Wizard automatically adds a call to EnableWebTransport to the overridden UpdateRegistry method of the new remote data module class. This adds the registry entries that httpsrvr.dll uses to check whether clients can access the application server. The registry entries are removed by a call to DisableWebTransport (see page 6601) (also added to UpdateRegistry).

To make an application server unavailable to HTTP-based connections, remove these two calls from UpdateRegistry. The application server can still receive requests over connections formed using other protocols.

ClassID is the class ID of the remote data module. This value is passed as a parameter to the UpdateRegistry method that calls DisableSocketTransport (see page 6600).

See Also

DisableWebTransport (see page 6601)

TRemoteDataModule (see page 6585)

Using Web Connections

37.2.5 DataBkr.RegisterPooled Function

Registers a remote data module so that instances are pooled.

Pascal

```
procedure RegisterPooled(const ClassID: string; Max: Integer; Timeout: Integer; Singleton:
Boolean = False);
```

C++

```
RegisterPooled(const AnsiString ClassID, int Max, int Timeout, Boolean Singleton = False);
```

File

DataBkr

Description

In Delphi applications, to enable remote data module instances to be pooled, call RegisterPooled from the UpdateRegistry method of a remote data module class. When data module instances are pooled, the server maintains a cache of remote data module instances. Each client request is serviced by the first available instance from that cache. Because instances are shared, the remote data module can't rely on persistent state information.

ClassID is the class ID of the remote data module. This value is passed as a parameter to the UpdateRegistry method that calls RegisterPooled.

Max indicates the maximum number of instances in the pool. If client applications request more than this maximum number, the application server raises a "Server too busy" exception.

Timeout indicates the number of minutes a remote data module instance can sit idle in the object pool. If an instance receives no client requests for more than Timeout minutes, it is automatically freed. The server checks every 6 minutes, so an object may persist in an idle state up to 6 minutes longer than Timeout. Setting Timeout to 0 indicates an infinite timeout period (that is, unused objects are not freed until the application server shuts down).

Singleton indicates that all client calls go to the same remote data module instance. There is only a single instance rather than a pool of available instances. Only set Singleton to true if the remote data module is free-threaded.

Note: When Singleton is true, the values of Max and Timeout are ignored because there is only one instance with no timeout period.

See Also

UnregisterPooled (see page 6603)

TRemoteDataModule (see page 6585)

Pooling Remote Data Modules

37.2.6 DataBkr.UnregisterPooled Function

Removes the object pooling registry entries for a remote data module.

Pascal

```
procedure UnregisterPooled(const ClassID: string);
```

C++

```
UnregisterPooled(const AnsiString ClassID);
```

File

DataBkr

Description

When remote data module instances are pooled in a Delphi application, call UnregisterPooled from the UpdateRegistry method of a remote data module class. This unregisters the object pooling at the same time the remote data module is unregistered as an automation server.

ClassID is the class ID of the remote data module. This value is passed as a parameter to the UpdateRegistry method that calls UnregisterPooled.

See Also

RegisterPooled (see page 6602)

TRemoteDataModule (see page 6585)

Pooling Remote Data Modules

38 DateUtils Namespace

This is namespace DateUtils.

Topics

Name	Description
Functions (see page 6610)	The following table lists functions in this documentation.
Variables (see page 6717)	The following table lists variables in this documentation.
Constants (see page 6718)	The following table lists constants in this documentation.

Constants

Name	Description
DayFriday (see page 6718)	This is constant DateUtils.DayFriday.
DayMonday (see page 6718)	This is constant DateUtils.DayMonday.
DaySaturday (see page 6719)	This is constant DateUtils.DaySaturday.
DaysPerWeek (see page 6719)	This is constant DateUtils.DaysPerWeek.
DaysPerYear (see page 6719)	This is constant DateUtils.DaysPerYear.
DaySunday (see page 6720)	This is constant DateUtils.DaySunday.
DayThursday (see page 6720)	This is constant DateUtils.DayThursday.
DayTuesday (see page 6720)	This is constant DateUtils.DayTuesday.
DayWednesday (see page 6721)	This is constant DateUtils.DayWednesday.
MonthsPerYear (see page 6721)	This is constant DateUtils.MonthsPerYear.
OneHour (see page 6721)	Delphi inverse time unit constants.
OneMillisecond (see page 6722)	This is constant DateUtils.OneMillisecond.
OneMinute (see page 6722)	This is constant DateUtils.OneMinute.
OneSecond (see page 6722)	This is constant DateUtils.OneSecond.
RecodeLeaveFieldAsIs (see page 6723)	This is constant DateUtils.RecodeLeaveFieldAsIs.
WeeksPerFortnight (see page 6723)	This is constant DateUtils.WeeksPerFortnight.
YearsPerCentury (see page 6723)	This is constant DateUtils.YearsPerCentury.
YearsPerDecade (see page 6724)	This is constant DateUtils.YearsPerDecade.
YearsPerMillennium (see page 6724)	This is constant DateUtils.YearsPerMillennium.

Functions

	Name	Description
	CompareDate (see page 6614)	Indicates the relationship between the date portions of two TDateTime values.
	CompareDateTime (see page 6614)	Indicates the relationship between two TDateTime values.
	CompareTime (see page 6615)	Indicates the relationship between the time portions of two TDateTime values.

	DateOf (see page 6616)	Strips the time portion from a TDateTime value.
	DateTimeToJulianDate (see page 6616)	Converts a TDateTime value into a Julian date.
	DateTimeToModifiedJulianDate (see page 6617)	Converts a TDateTime value into a modified Julian date.
	DateTimeToUnix (see page 6617)	Converts a TDateTime value into a Unix/Linux-style date-and-time value.
	DayOf (see page 6618)	Returns the day of the month represented by a TDateTime value.
	DayOfTheMonth (see page 6618)	Returns the day of the month represented by a TDateTime value.
	DayOfTheWeek (see page 6619)	Returns the day of the week represented by a TDateTime value.
	DayOfTheYear (see page 6620)	Returns the number of days between a specified TDateTime value and December 31 of the previous year.
	DaysBetween (see page 6621)	Returns the number of whole days between two specified TDateTime values.
	DaysInAMonth (see page 6622)	Returns the number of days in a specified month of a specified year.
	DaysInAYear (see page 6622)	Returns the number of days in a specified year.
	DaysInMonth (see page 6623)	Returns the number of days in the month of a specified TDateTime value.
	DaysInYear (see page 6623)	Returns the number of days in the year of a specified TDateTime value.
	DaySpan (see page 6624)	Returns the number of days (including fractional days) between two specified TDateTime values.
	DecodeDateDay (see page 6624)	Returns the year and day of the year for a specified TDateTime.
	DecodeDateMonthWeek (see page 6625)	Returns the year, month, week of the month, and day of the week for a specified TDateTime.
	DecodeDateTime (see page 6626)	Returns Year, Month, Day, Hour, Minute, Second, and Millisecond values for a TDateTime.
	DecodeDateWeek (see page 6627)	Returns the year, week of the year, and day of the week for a specified TDateTime value object.
	DecodeDayOfWeekInMonth (see page 6628)	For a given TDateTime value, returns the year, month, day of the week, and the count of that day of the week in the month.
	EncodeDateDay (see page 6629)	Returns a TDateTime that represents a specified day of the year for a specified year.
	EncodeDateMonthWeek (see page 6629)	Returns a TDateTime that represents a specified day of a specified week in a specified month and year.
	EncodeDateTime (see page 6630)	Returns a TDateTime that represents a specified year, month, day, hour, minute, second, and millisecond.
	EncodeDateWeek (see page 6631)	Returns a TDateTime that represents a specified day of a specified week in a specified year.
	EncodeDayOfWeekInMonth (see page 6632)	Returns a TDateTime that represents a specified occurrence of a day of the week within a specified month and year.
	EndOfADay (see page 6633)	Returns a TDateTime that represents the last millisecond of a specified day.
	EndOfADay (see page 6633)	Returns a TDateTime that represents the last millisecond of a specified day.
	EndOfAMonth (see page 6634)	Returns a TDateTime that represents the last millisecond of the last day of a specified month.
	EndOfAWeek (see page 6635)	Returns a TDateTime object value that represents the last millisecond of a specified day of a specified week.
	EndOfAYear (see page 6636)	Returns a TDateTime that represents the last millisecond of a specified year.
	EndOfTheDay (see page 6636)	Returns a TDateTime that represents the last millisecond of the day identified by a specified TDateTime.
	EndOfTheMonth (see page 6637)	Returns a TDateTime that represents the last millisecond of the last day of the month identified by a specified TDateTime.
	EndOfTheWeek (see page 6637)	Returns a TDateTime that represents the last millisecond of the last day of the week identified by a specified TDateTime.
	EndOfTheYear (see page 6638)	Returns a TDateTime that represents the last millisecond of the last day of the year identified by a specified TDateTime.
	HourOf (see page 6638)	Returns the hour of the day represented by a TDateTime value.
	HourOfTheDay (see page 6639)	Returns the hour of the day represented by a TDateTime value.
	HourOfTheMonth (see page 6640)	Returns the number of hours between a specified TDateTime value and 12:00 AM of the first day of the month.
	HourOfTheWeek (see page 6641)	Returns the number of hours between a specified TDateTime value and 12:00 AM of the first day of the week.
	HourOfTheYear (see page 6641)	Returns the number of hours between a specified TDateTime value and 12:00 AM of the first day of the year.
	HoursBetween (see page 6642)	Returns the number of whole hours between two specified TDateTime values.
	HourSpan (see page 6643)	Returns the number of Hours (including fractional Hours) between two specified TDateTime values.

	IncDay (see page 6643)	Returns a date shifted by a specified number of days.
	IncHour (see page 6644)	Returns a date/time value shifted by a specified number of hours.
	IncMilliSecond (see page 6645)	Returns a date/time value shifted by a specified number of milliseconds.
	IncMinute (see page 6645)	Returns a date/time value shifted by a specified number of minutes.
	IncSecond (see page 6646)	Returns a date/time value shifted by a specified number of seconds.
	IncWeek (see page 6647)	Returns a date shifted by a specified number of weeks.
	IncYear (see page 6647)	Returns a date shifted by a specified number of years.
	InvalidDateDayError (see page 6648)	This is function DateUtils.InvalidDateDayError.
	InvalidDateMonthWeekError (see page 6648)	This is function DateUtils.InvalidDateMonthWeekError.
	InvalidDateTimeError (see page 6649)	This is function DateUtils.InvalidDateTimeError.
	InvalidDateWeekError (see page 6649)	This is function DateUtils.InvalidDateWeekError.
	InvalidDayOfWeekInMonthError (see page 6650)	This is function DateUtils.InvalidDayOfWeekInMonthError.
	IsInLeapYear (see page 6650)	Indicates whether a specified TDateTime value occurs in a leap year.
	IsPM (see page 6650)	Indicates whether the time portion of a specified TDateTime value occurs after noon.
	IsSameDay (see page 6651)	Indicates whether a specified TDateTime value occurs on a the same day as a criterion date.
	IsToday (see page 6651)	Indicates whether a specified TDateTime value occurs on the current date.
	IsValidDate (see page 6652)	Indicates whether a specified year, month, and day represent a valid date.
	IsValidDateDay (see page 6652)	Indicates whether a specified year and day of the year represent a valid date.
	IsValidDateMonthWeek (see page 6653)	Indicates whether a specified year, month, week of the month, and day of the week represent a valid date.
	IsValidDateTime (see page 6654)	Indicates whether a specified year, month, day, hour, minute, second, and millisecond represent a valid date and time.
	IsValidDateWeek (see page 6654)	Indicates whether a specified year, week of the year, and day of the week represent a valid date.
	IsValidTime (see page 6655)	Indicates whether a specified hour, minute, second, and millisecond represent a valid date and time.
	JulianDateToDateTime (see page 6656)	Converts a Julian date to a TDateTime value.
	MilliSecondOf (see page 6656)	Returns the millisecond of the second represented by a TDateTime value.
	MilliSecondOfTheDay (see page 6657)	Returns the number of milliseconds between a specified TDateTime value and the beginning of the same day.
	MilliSecondOfTheHour (see page 6658)	Returns the number of milliseconds between a specified TDateTime value and the beginning of the same hour.
	MilliSecondOfTheMinute (see page 6659)	Returns the number of milliseconds between a specified TDateTime value and the beginning of the same minute.
	MilliSecondOfTheMonth (see page 6659)	Returns the number of milliseconds between a specified TDateTime value and the beginning of the month.
	MilliSecondOfTheSecond (see page 6660)	Returns the millisecond of the second represented by a TDateTime value.
	MilliSecondOfTheWeek (see page 6661)	Returns the number of milliseconds between a specified TDateTime value and 12:00:00:00 AM of the first day of the week.
	MilliSecondOfTheYear (see page 6661)	Returns the number of milliseconds between a specified TDateTime value and 12:00:00:00 AM of the first day of the year.
	MilliSecondsBetween (see page 6662)	Returns the number of milliseconds between two specified TDateTime values.
	MilliSecondSpan (see page 6663)	Returns the number of milliseconds between two specified TDateTime values.
	MinuteOf (see page 6663)	Returns the minute of the hour represented by a TDateTime value.
	MinuteOfTheDay (see page 6664)	Returns the number of minutes between a specified TDateTime value and 12:00 AM of the same day.
	MinuteOfTheHour (see page 6665)	Returns the number of minutes between a specified TDateTime value and the beginning of the same hour.
	MinuteOfTheMonth (see page 6666)	Returns the number of minutes between a specified TDateTime value and 12:00 AM of the first day of the month.
	MinuteOfTheWeek (see page 6666)	Returns the number of minutes between a specified TDateTime value and 12:00 AM of the first day of the week.
	MinuteOfTheYear (see page 6667)	Returns the number of minutes between a specified TDateTime value and 12:00 AM of the first day of the year.
	MinutesBetween (see page 6668)	Returns the number of minutes between two specified TDateTime values.
	MinuteSpan (see page 6668)	Returns the number of minutes, including fractions thereof, between two specified TDateTime values.
	ModifiedJulianDateToDateTime (see page 6669)	Converts a modified Julian date to a TDateTime value.

	MonthOf (see page 6670)	Returns the month of the year represented by a TDateTime value.
	MonthOfTheYear (see page 6670)	Returns the month of the year represented by a TDateTime value.
	MonthsBetween (see page 6671)	Returns the approximate number of months between two specified TDateTime values.
	MonthSpan (see page 6672)	Returns the approximate number of months, including fractions, between two specified TDateTime values.
	NthDayOfWeek (see page 6672)	Returns which occurrence of its weekday a specified TDateTime value represents.
	RecodeDate (see page 6673)	Replaces the date portion of a specified TDateTime value.
	RecodeDateTime (see page 6674)	Selectively replaces parts of a specified TDateTime value.
	RecodeDay (see page 6675)	Replaces the day of the month for a specified TDateTime value.
	RecodeHour (see page 6675)	Replaces the hour of the day for a specified TDateTime value.
	RecodeMilliSecond (see page 6676)	Replaces the millisecond of the second for a specified TDateTime value.
	RecodeMinute (see page 6677)	Replaces the minute of the hour for a specified TDateTime value.
	RecodeMonth (see page 6678)	Replaces the month of the year for a specified TDateTime value.
	RecodeSecond (see page 6678)	Replaces the second of the minute for a specified TDateTime value.
	RecodeTime (see page 6679)	Replaces the time portion of a specified TDateTime value.
	RecodeYear (see page 6680)	Replaces the year for a specified TDateTime value.
	SameDate (see page 6681)	Indicates whether two TDateTime values represent the same year, month, and day.
	SameDateTime (see page 6681)	Indicates whether two TDateTime values represent the same year, month, day, hour, minute, second, and millisecond.
	SameTime (see page 6682)	Indicates whether two TDateTime values represent the same time of day, ignoring the date portion.
	SecondOf (see page 6682)	Returns the second of the minute represented by a TDateTime value.
	SecondOfTheDay (see page 6683)	Returns the number of seconds between a specified TDateTime value and 12:00:00 AM of the same day.
	SecondOfTheHour (see page 6684)	Returns the number of seconds between a specified TDateTime value and the beginning of the same hour.
	SecondOfTheMinute (see page 6685)	Returns the number of seconds between a specified TDateTime value and the beginning of the same minute.
	SecondOfTheMonth (see page 6685)	Returns the number of seconds between a specified TDateTime value and 12:00:00 AM on the first day of the month.
	SecondOfTheWeek (see page 6686)	Returns the number of seconds between a specified TDateTime value and 12:00:00 AM of the first day of the week.
	SecondOfTheYear (see page 6687)	Returns the number of seconds between a specified TDateTime value and 12:00:00 AM on the first day of the year.
	SecondsBetween (see page 6687)	Returns the number of seconds between two specified TDateTime values.
	SecondSpan (see page 6688)	Returns the number of seconds (including fractions thereof) between two specified TDateTime values.
	StartOfADay (see page 6689)	Returns a TDateTime that represents 12:00:00:00 AM on a specified day.
	StartOfADay (see page 6689)	Returns a TDateTime that represents 12:00:00:00 AM on a specified day.
	StartOfAMonth (see page 6690)	Returns a TDateTime that represents 12:00:00:00 AM on the first day of a specified month.
	StartOfAWeek (see page 6690)	Returns a TDateTime that represents 1the first moment on a specified day of a specified week.
	StartOfAYear (see page 6691)	Returns a TDateTime that represents the first moment on the first day of a specified year.
	StartOfTheDay (see page 6692)	Returns a TDateTime that represents 12:00:00:00 AM on the day identified by a specified TDateTime.
	StartOfTheMonth (see page 6692)	Returns a TDateTime that represents 12:00:00:00 AM on the first day of the month identified by a specified TDateTime.
	StartOfTheWeek (see page 6693)	Returns a TDateTime that represents 12:00:00:00 AM on the first day of the week identified by a specified TDateTime.
	StartOfTheYear (see page 6693)	Returns a TDateTime that represents 12:00:00:00 AM on the first day of the year identified by a specified TDateTime.
	TimeOf (see page 6694)	Strips the date portion from a TDateTime value.
	Today (see page 6694)	Returns a TDateTime value that represents the current date.
	Tomorrow (see page 6695)	Returns a TDateTime value that represents the following day.
	TryEncodeDateDay (see page 6695)	Calculates the TDateTime value that represents a specified day of the year for a specified year.

	TryEncodeDateMonthWeek (see page 6696)	Calculates the TDateTime value that represents a specified day of a specified week in a specified month and year.
	TryEncodeDateTime (see page 6697)	Calculates the TDateTime value that represents a specified year, month, day, hour, minute, second, and millisecond.
	TryEncodeDateWeek (see page 6698)	Calculates the TDateTime value that represents a specified day of a specified week in a specified year.
	TryEncodeDayOfWeekInMonth (see page 6699)	Calculates a TDateTime that represents a specified occurrence of a day of the week within a specified month and year.
	TryJulianDateToDateTime (see page 6700)	Converts a Julian date to a TDateTime value.
	TryModifiedJulianDateToDateTime (see page 6701)	Converts a modified Julian date to a TDateTime value.
	TryRecodeDateTime (see page 6702)	Selectively replaces parts of a specified TDateTime value.
	UnixToDateTime (see page 6703)	Converts a Unix/Linux style date-and-time value to a TDateTime value.
	WeekOf (see page 6703)	Returns the week of the year represented by a TDateTime value.
	WeekOfTheMonth (see page 6704)	Returns the week of the month represented by a TDateTime value.
	WeekOfTheMonth (see page 6704)	Returns the week of the month represented by a TDateTime value.
	WeekOfTheYear (see page 6705)	Returns the week of the year represented by a TDateTime value.
	WeekOfTheYear (see page 6705)	Returns the week of the year represented by a TDateTime value.
	WeeksBetween (see page 6706)	Returns the number of whole weeks between two specified TDateTime values.
	WeeksInAYear (see page 6706)	Returns the number of weeks in a specified year.
	WeeksInYear (see page 6707)	Returns the number of weeks in the year of a specified TDateTime value.
	WeekSpan (see page 6708)	Returns the number of weeks (including fractional weeks) between two specified TDateTime values.
	WithinPastDays (see page 6708)	Indicates whether two dates are within a specified number of days of each other.
	WithinPastHours (see page 6709)	Indicates whether two date/time values are within a specified number of hours of each other.
	WithinPastMilliSeconds (see page 6710)	Indicates whether two date/time values are within a specified number of milliseconds of each other.
	WithinPastMinutes (see page 6710)	Indicates whether two date/time values are within a specified number of minutes of each other.
	WithinPastMonths (see page 6711)	Indicates whether two date/time values are within a specified number of months of each other.
	WithinPastSeconds (see page 6712)	Indicates whether two date/time values are within a specified number of seconds of each other.
	WithinPastWeeks (see page 6712)	Indicates whether two date/time values are within a specified number of weeks of each other.
	WithinPastYears (see page 6713)	Indicates whether two date/time values are within a specified number of years of each other.
	YearOf (see page 6714)	Returns the year represented by a TDateTime value.
	YearsBetween (see page 6714)	Returns the approximate number of years between two specified TDateTime values.
	YearSpan (see page 6715)	Returns the approximate number of years (including fractions thereof) between two specified TDateTime values.
	Yesterday (see page 6716)	Returns a TDateTime value that represents the preceding day.

Variables

Name	Description
ApproxDaysPerMonth (see page 6717)	This is variable DateUtils.ApproxDaysPerMonth.
ApproxDaysPerYear (see page 6717)	This is variable DateUtils.ApproxDaysPerYear.

38.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CompareDate (see page 6614)	Indicates the relationship between the date portions of two TDateTime values.
	CompareDateTime (see page 6614)	Indicates the relationship between two TDateTime values.
	CompareTime (see page 6615)	Indicates the relationship between the time portions of two TDateTime values.
	DateOf (see page 6616)	Strips the time portion from a TDateTime value.
	DateTimeToJulianDate (see page 6616)	Converts a TDateTime value into a Julian date.
	DateTimeToModifiedJulianDate (see page 6617)	Converts a TDateTime value into a modified Julian date.
	DateTimeToUnix (see page 6617)	Converts a TDateTime value into a Unix/Linux-style date-and-time value.
	DayOf (see page 6618)	Returns the day of the month represented by a TDateTime value.
	DayOfTheMonth (see page 6618)	Returns the day of the month represented by a TDateTime value.
	DayOfTheWeek (see page 6619)	Returns the day of the week represented by a TDateTime value.
	DayOfTheYear (see page 6620)	Returns the number of days between a specified TDateTime value and December 31 of the previous year.
	DaysBetween (see page 6621)	Returns the number of whole days between two specified TDateTime values.
	DaysInAMonth (see page 6622)	Returns the number of days in a specified month of a specified year.
	DaysInAYear (see page 6622)	Returns the number of days in a specified year.
	DaysInMonth (see page 6623)	Returns the number of days in the month of a specified TDateTime value.
	DaysInYear (see page 6623)	Returns the number of days in the year of a specified TDateTime value.
	DaySpan (see page 6624)	Returns the number of days (including fractional days) between two specified TDateTime values.
	DecodeDateDay (see page 6624)	Returns the year and day of the year for a specified TDateTime.
	DecodeDateMonthWeek (see page 6625)	Returns the year, month, week of the month, and day of the week for a specified TDateTime.
	DecodeDateTime (see page 6626)	Returns Year, Month, Day, Hour, Minute, Second, and Millisecond values for a TDateTime.
	DecodeDateWeek (see page 6627)	Returns the year, week of the year, and day of the week for a specified TDateTime value object.
	DecodeDayOfWeekInMonth (see page 6628)	For a given TDateTime value, returns the year, month, day of the week, and the count of that day of the week in the month.
	EncodeDateDay (see page 6629)	Returns a TDateTime that represents a specified day of the year for a specified year.
	EncodeDateMonthWeek (see page 6629)	Returns a TDateTime that represents a specified day of a specified week in a specified month and year.
	EncodeDateTime (see page 6630)	Returns a TDateTime that represents a specified year, month, day, hour, minute, second, and millisecond.
	EncodeDateWeek (see page 6631)	Returns a TDateTime that represents a specified day of a specified week in a specified year.
	EncodeDayOfWeekInMonth (see page 6632)	Returns a TDateTime that represents a specified occurrence of a day of the week within a specified month and year.
	EndOfADay (see page 6633)	Returns a TDateTime that represents the last millisecond of a specified day.
	EndOfADay (see page 6633)	Returns a TDateTime that represents the last millisecond of a specified day.
	EndOfAMonth (see page 6634)	Returns a TDateTime that represents the last millisecond of the last day of a specified month.
	EndOfAWeek (see page 6635)	Returns a TDateTime object value that represents the last millisecond of a specified day of a specified week.
	EndOfAYear (see page 6636)	Returns a TDateTime that represents the last millisecond of a specified year.
	EndOfTheDay (see page 6636)	Returns a TDateTime that represents the last millisecond of the day identified by a specified TDateTime.
	EndOfTheMonth (see page 6637)	Returns a TDateTime that represents the last millisecond of the last day of the month identified by a specified TDateTime.

	EndOfTheWeek (see page 6637)	Returns a TDateTime that represents the last millisecond of the last day of the week identified by a specified TDateTime.
	EndOfTheYear (see page 6638)	Returns a TDateTime that represents the last millisecond of the last day of the year identified by a specified TDateTime.
	HourOf (see page 6638)	Returns the hour of the day represented by a TDateTime value.
	HourOfTheDay (see page 6639)	Returns the hour of the day represented by a TDateTime value.
	HourOfTheMonth (see page 6640)	Returns the number of hours between a specified TDateTime value and 12:00 AM of the first day of the month.
	HourOfTheWeek (see page 6641)	Returns the number of hours between a specified TDateTime value and 12:00 AM of the first day of the week.
	HourOfTheYear (see page 6641)	Returns the number of hours between a specified TDateTime value and 12:00 AM of the first day of the year.
	HoursBetween (see page 6642)	Returns the number of whole hours between two specified TDateTime values.
	HourSpan (see page 6643)	Returns the number of Hours (including fractional Hours) between two specified TDateTime values.
	IncDay (see page 6643)	Returns a date shifted by a specified number of days.
	IncHour (see page 6644)	Returns a date/time value shifted by a specified number of hours.
	IncMilliSecond (see page 6645)	Returns a date/time value shifted by a specified number of milliseconds.
	IncMinute (see page 6645)	Returns a date/time value shifted by a specified number of minutes.
	IncSecond (see page 6646)	Returns a date/time value shifted by a specified number of seconds.
	IncWeek (see page 6647)	Returns a date shifted by a specified number of weeks.
	IncYear (see page 6647)	Returns a date shifted by a specified number of years.
	InvalidDateDayError (see page 6648)	This is function DateUtils.InvalidDateDayError.
	InvalidDateMonthWeekError (see page 6648)	This is function DateUtils.InvalidDateMonthWeekError.
	InvalidDateTimeError (see page 6649)	This is function DateUtils.InvalidDateTimeError.
	InvalidDateWeekError (see page 6649)	This is function DateUtils.InvalidDateWeekError.
	InvalidDayOfWeekInMonthError (see page 6650)	This is function DateUtils.InvalidDayOfWeekInMonthError.
	IsInLeapYear (see page 6650)	Indicates whether a specified TDateTime value occurs in a leap year.
	IsPM (see page 6650)	Indicates whether the time portion of a specified TDateTime value occurs after noon.
	IsSameDay (see page 6651)	Indicates whether a specified TDateTime value occurs on a the same day as a criterion date.
	IsToday (see page 6651)	Indicates whether a specified TDateTime value occurs on the current date.
	IsValidDate (see page 6652)	Indicates whether a specified year, month, and day represent a valid date.
	IsValidDateDay (see page 6652)	Indicates whether a specified year and day of the year represent a valid date.
	IsValidDateMonthWeek (see page 6653)	Indicates whether a specified year, month, week of the month, and day of the week represent a valid date.
	IsValidDateTime (see page 6654)	Indicates whether a specified year, month, day, hour, minute, second, and millisecond represent a valid date and time.
	IsValidDateWeek (see page 6654)	Indicates whether a specified year, week of the year, and day of the week represent a valid date.
	IsValidTime (see page 6655)	Indicates whether a specified hour, minute, second, and millisecond represent a valid date and time.
	JulianDateToDateTime (see page 6656)	Converts a Julian date to a TDateTime value.
	MilliSecondOf (see page 6656)	Returns the millisecond of the second represented by a TDateTime value.
	MilliSecondOfTheDay (see page 6657)	Returns the number of milliseconds between a specified TDateTime value and the beginning of the same day.
	MilliSecondOfTheHour (see page 6658)	Returns the number of milliseconds between a specified TDateTime value and the beginning of the same hour.
	MilliSecondOfTheMinute (see page 6659)	Returns the number of milliseconds between a specified TDateTime value and the beginning of the same minute.
	MilliSecondOfTheMonth (see page 6659)	Returns the number of milliseconds between a specified TDateTime value and the beginning of the month.
	MilliSecondOfTheSecond (see page 6660)	Returns the millisecond of the second represented by a TDateTime value.
	MilliSecondOfTheWeek (see page 6661)	Returns the number of milliseconds between a specified TDateTime value and 12:00:00:00 AM of the first day of the week.
	MilliSecondOfTheYear (see page 6661)	Returns the number of milliseconds between a specified TDateTime value and 12:00:00:00 AM of the first day of the year.
	MilliSecondsBetween (see page 6662)	Returns the number of milliseconds between two specified TDateTime values.
	MilliSecondSpan (see page 6663)	Returns the number of milliseconds between two specified TDateTime values.

	MinuteOf (see page 6663)	Returns the minute of the hour represented by a TDateTime value.
	MinuteOfTheDay (see page 6664)	Returns the number of minutes between a specified TDateTime value and 12:00 AM of the same day.
	MinuteOfTheHour (see page 6665)	Returns the number of minutes between a specified TDateTime value and the beginning of the same hour.
	MinuteOfTheMonth (see page 6666)	Returns the number of minutes between a specified TDateTime value and 12:00 AM of the first day of the month.
	MinuteOfTheWeek (see page 6666)	Returns the number of minutes between a specified TDateTime value and 12:00 AM of the first day of the week.
	MinuteOfTheYear (see page 6667)	Returns the number of minutes between a specified TDateTime value and 12:00 AM of the first day of the year.
	MinutesBetween (see page 6668)	Returns the number of minutes between two specified TDateTime values.
	MinuteSpan (see page 6668)	Returns the number of minutes, including fractions thereof, between two specified TDateTime values.
	ModifiedJulianDateToDateTime (see page 6669)	Converts a modified Julian date to a TDateTime value.
	MonthOf (see page 6670)	Returns the month of the year represented by a TDateTime value.
	MonthOfTheYear (see page 6670)	Returns the month of the year represented by a TDateTime value.
	MonthsBetween (see page 6671)	Returns the approximate number of months between two specified TDateTime values.
	MonthSpan (see page 6672)	Returns the approximate number of months, including fractions, between two specified TDateTime values.
	NthDayOfWeek (see page 6672)	Returns which occurrence of its weekday a specified TDateTime value represents.
	RecodeDate (see page 6673)	Replaces the date portion of a specified TDateTime value.
	RecodeDateTime (see page 6674)	Selectively replaces parts of a specified TDateTime value.
	RecodeDay (see page 6675)	Replaces the day of the month for a specified TDateTime value.
	RecodeHour (see page 6675)	Replaces the hour of the day for a specified TDateTime value.
	RecodeMilliSecond (see page 6676)	Replaces the millisecond of the second for a specified TDateTime value.
	RecodeMinute (see page 6677)	Replaces the minute of the hour for a specified TDateTime value.
	RecodeMonth (see page 6678)	Replaces the month of the year for a specified TDateTime value.
	RecodeSecond (see page 6678)	Replaces the second of the minute for a specified TDateTime value.
	RecodeTime (see page 6679)	Replaces the time portion of a specified TDateTime value.
	RecodeYear (see page 6680)	Replaces the year for a specified TDateTime value.
	SameDate (see page 6681)	Indicates whether two TDateTime values represent the same year, month, and day.
	SameDateTime (see page 6681)	Indicates whether two TDateTime values represent the same year, month, day, hour, minute, second, and millisecond.
	SameTime (see page 6682)	Indicates whether two TDateTime values represent the same time of day, ignoring the date portion.
	SecondOf (see page 6682)	Returns the second of the minute represented by a TDateTime value.
	SecondOfTheDay (see page 6683)	Returns the number of seconds between a specified TDateTime value and 12:00:00 AM of the same day.
	SecondOfTheHour (see page 6684)	Returns the number of seconds between a specified TDateTime value and the beginning of the same hour.
	SecondOfTheMinute (see page 6685)	Returns the number of seconds between a specified TDateTime value and the beginning of the same minute.
	SecondOfTheMonth (see page 6685)	Returns the number of seconds between a specified TDateTime value and 12:00:00 AM on the first day of the month.
	SecondOfTheWeek (see page 6686)	Returns the number of seconds between a specified TDateTime value and 12:00:00 AM of the first day of the week.
	SecondOfTheYear (see page 6687)	Returns the number of seconds between a specified TDateTime value and 12:00:00 AM on the first day of the year.
	SecondsBetween (see page 6687)	Returns the number of seconds between two specified TDateTime values.
	SecondSpan (see page 6688)	Returns the number of seconds (including fractions thereof) between two specified TDateTime values.
	StartOfADay (see page 6689)	Returns a TDateTime that represents 12:00:00:00 AM on a specified day.
	StartOfADay (see page 6689)	Returns a TDateTime that represents 12:00:00:00 AM on a specified day.
	StartOfAMonth (see page 6690)	Returns a TDateTime that represents 12:00:00:00 AM on the first day of a specified month.
	StartOfAWeek (see page 6690)	Returns a TDateTime that represents 1the first moment on a specified day of a specified week.

	StartOfAYear (see page 6691)	Returns a TDateTime that represents the first moment on the first day of a specified year.
	StartOfTheDay (see page 6692)	Returns a TDateTime that represents 12:00:00:00 AM on the day identified by a specified TDateTime.
	StartOfTheMonth (see page 6692)	Returns a TDateTime that represents 12:00:00:00 AM on the first day of the month identified by a specified TDateTime.
	StartOfTheWeek (see page 6693)	Returns a TDateTime that represents 12:00:00:00 AM on the first day of the week identified by a specified TDateTime.
	StartOfTheYear (see page 6693)	Returns a TDateTime that represents 12:00:00:00 AM on the first day of the year identified by a specified TDateTime.
	TimeOf (see page 6694)	Strips the date portion from a TDateTime value.
	Today (see page 6694)	Returns a TDateTime value that represents the current date.
	Tomorrow (see page 6695)	Returns a TDateTime value that represents the following day.
	TryEncodeDateDay (see page 6695)	Calculates the TDateTime value that represents a specified day of the year for a specified year.
	TryEncodeDateMonthWeek (see page 6696)	Calculates the TDateTime value that represents a specified day of a specified week in a specified month and year.
	TryEncodeDateTime (see page 6697)	Calculates the TDateTime value that represents a specified year, month, day, hour, minute, second, and millisecond.
	TryEncodeDateWeek (see page 6698)	Calculates the TDateTime value that represents a specified day of a specified week in a specified year.
	TryEncodeDayOfWeekInMonth (see page 6699)	Calculates a TDateTime that represents a specified occurrence of a day of the week within a specified month and year.
	TryJulianDateToDateTime (see page 6700)	Converts a Julian date to a TDateTime value.
	TryModifiedJulianDateToDateTime (see page 6701)	Converts a modified Julian date to a TDateTime value.
	TryRecodeDateTime (see page 6702)	Selectively replaces parts of a specified TDateTime value.
	UnixToDateTime (see page 6703)	Converts a Unix/Linux style date-and-time value to a TDateTime value.
	WeekOf (see page 6703)	Returns the week of the year represented by a TDateTime value.
	WeekOfTheMonth (see page 6704)	Returns the week of the month represented by a TDateTime value.
	WeekOfTheMonth (see page 6704)	Returns the week of the month represented by a TDateTime value.
	WeekOfTheYear (see page 6705)	Returns the week of the year represented by a TDateTime value.
	WeekOfTheYear (see page 6705)	Returns the week of the year represented by a TDateTime value.
	WeeksBetween (see page 6706)	Returns the number of whole weeks between two specified TDateTime values.
	WeeksInAYear (see page 6706)	Returns the number of weeks in a specified year.
	WeeksInYear (see page 6707)	Returns the number of weeks in the year of a specified TDateTime value.
	WeekSpan (see page 6708)	Returns the number of weeks (including fractional weeks) between two specified TDateTime values.
	WithinPastDays (see page 6708)	Indicates whether two dates are within a specified number of days of each other.
	WithinPastHours (see page 6709)	Indicates whether two date/time values are within a specified number of hours of each other.
	WithinPastMilliSeconds (see page 6710)	Indicates whether two date/time values are within a specified number of milliseconds of each other.
	WithinPastMinutes (see page 6710)	Indicates whether two date/time values are within a specified number of minutes of each other.
	WithinPastMonths (see page 6711)	Indicates whether two date/time values are within a specified number of months of each other.
	WithinPastSeconds (see page 6712)	Indicates whether two date/time values are within a specified number of seconds of each other.
	WithinPastWeeks (see page 6712)	Indicates whether two date/time values are within a specified number of weeks of each other.
	WithinPastYears (see page 6713)	Indicates whether two date/time values are within a specified number of years of each other.
	YearOf (see page 6714)	Returns the year represented by a TDateTime value.
	YearsBetween (see page 6714)	Returns the approximate number of years between two specified TDateTime values.
	YearSpan (see page 6715)	Returns the approximate number of years (including fractions thereof) between two specified TDateTime values.
	Yesterday (see page 6716)	Returns a TDateTime value that represents the preceding day.

38.1.1 DateUtils.CompareDate Function

Indicates the relationship between the date portions of two TDateTime values.

Pascal

```
function CompareDate(const A: TDateTime; const B: TDateTime): TValueRelationship;
```

C++

```
TValueRelationship CompareDate(const TDateTime A, const TDateTime B);
```

File

DateUtils

Description

Call CompareDate to compare the two TDateTime values specified by A and B. CompareDate returns

LessThanValue if A occurs on a day prior to the day specified by B.

EqualsValue if A occurs on the same day as B, ignoring the time of day.

GreaterThanValue if A occurs on a day that follows the day specified by B.

See Also

SameDate (see page 6681)

ConvUnitCompareValue

CompareValue

CompareDateTime (see page 6614)

CompareTime (see page 6615)

IsSameDay (see page 6651)

38.1.2 DateUtils.CompareDateTime Function

Indicates the relationship between two TDateTime values.

Pascal

```
function CompareDateTime(const A: TDateTime; const B: TDateTime): TValueRelationship;
```

C++

```
TValueRelationship CompareDateTime(const TDateTime A, const TDateTime B);
```

File

DateUtils

Description

Call CompareDateTime to compare the two TDateTime values specified by A and B. CompareDateTime returns

LessThanValue if A is earlier than B.

EqualsValue if A is the same as B.

GreaterThanValue if A is later than B.

Note: CompareDateTime differs from a direct comparison of the corresponding double precision values in that two TDateTime values are considered the same if they have the same value down to the millisecond. It is possible to create two TDateTime values that differ numerically when viewed as doubles, but which represent the same year, month, day, hour, minute, second, and millisecond.

See Also

SameDateTime (see page 6681)

ConvUnitCompareValue

CompareValue

CompareDate (see page 6614)

CompareTime (see page 6615)

IsSameDay (see page 6651)

38.1.3 DateUtils.CompareTime Function

Indicates the relationship between the time portions of two TDateTime values.

Pascal

```
function CompareTime(const A: TDateTime; const B: TDateTime): TValueRelationship;
```

C++

```
TValueRelationship CompareTime(const TDateTime A, const TDateTime B);
```

File

DateUtils

Description

Call CompareTime to compare the two TDateTime values specified by A and B. CompareTime returns

LessThanValue if A occurs earlier in the day than B (even if A occurs on a later day than B).

EqualsValue if A occurs at the same time of day as B, ignoring of the date portion of the two values.

GreaterThanValue if A occurs later in the day than B (even if A occurs on an earlier day than B).

See Also

SameTime (see page 6682)

ConvUnitCompareValue

CompareValue

CompareDateTime (see page 6614)

CompareDate (see page 6614)

38.1.4 DateUtils.DateOf Function

Strips the time portion from a TDateTime value.

Pascal

```
function DateOf(const AValue: TDateTime): TDateTime;
```

C++

```
TDateTime DateOf(const TDateTime AValue);
```

File

DateUtils

Description

Call DateOf to convert a TDateTime value to a TDateTime value that includes only the date information (sets the time portion to 0, which means midnight).

See Also

DecodeDate

ReplaceTime

TimeOf (see page 6694)

38.1.5 DateUtils.DateTimeToJulianDate Function

Converts a TDateTime value into a Julian date.

Pascal

```
function DateTimeToJulianDate(const AValue: TDateTime): Double;
```

C++

```
Double DateTimeToJulianDate(const TDateTime AValue);
```

File

DateUtils

Description

Call DateTimeToJulianDate to convert a TDateTime value the corresponding Julian date.

The Julian date is the number of days, including fractional days, since 4713 BC January 1, Greenwich noon. Julian dates are used by astronomers, geophysicist, and others who need an unambiguous dating system based on continuing day counts.

See Also

DateTimeToModifiedJulianDate (see page 6617)

JulianDateToDateTime (see page 6656)

ModifiedJulianDateToDateTime (see page 6669)

Convert

TryJulianDateToDateTime (see page 6700)

TryModifiedJulianDateToDateTime (see page 6701)

38.1.6 DateUtils.DateTimeToModifiedJulianDate Function

Converts a TDateTime value into a modified Julian date.

Pascal

```
function DateTimeToModifiedJulianDate(const AValue: TDateTime): Double;
```

C++

```
Double DateTimeToModifiedJulianDate(const TDateTime AValue);
```

File

DateUtils

Description

Call DateTimeToModifiedJulianDate to convert a TDateTime value the corresponding modified Julian date.

The modified Julian date is the number of days, including fractional days, since Greenwich midnight on November 17, 1858. Modified Julian dates are based on Julian dates, but adjusted to use midnight rather than noon as a starting point and to use a more recent date as a starting point.

See Also

DateTimeToJulianDate (see page 6616)

JulianDateToDateTime (see page 6656)

ModifiedJulianDateToDateTime (see page 6669)

Convert

TryJulianDateToDateTime (see page 6700)

TryModifiedJulianDateToDateTime (see page 6701)

38.1.7 DateUtils.DateTimeToUnix Function

Converts a TDateTime value into a Unix/Linux-style date-and-time value.

Pascal

```
function DateTimeToUnix(const AValue: TDateTime): Int64;
```

C++

```
Int64 DateTimeToUnix(const TDateTime AValue);
```

File

DateUtils

Description

Call DateTimeToUnix to convert a TDateTime value the corresponding Unix/Linux encoding of the same date and time.

Unix/Linux date-and-time values are encoded as the number of seconds since midnight at the start of January 1, 1970.

See Also

UnixToDateTime (see page 6703)

Convert

38.1.8 DateUtils.DayOf Function

Returns the day of the month represented by a TDateTime value.

Pascal

```
function DayOf(const AValue: TDateTime): Word;
```

C++

```
Word DayOf(const TDateTime AValue);
```

File

DateUtils

Description

Call DayOf to obtain the day of the month represented by a specified TDateTime value. DayOf returns a value between 1 and 31.

Note: DayOf returns the same value as the DayOfTheMonth (see page 6618) function.

See Also

DecodeDate

DayOfTheYear (see page 6620)

DayOfTheMonth (see page 6618)

HourOf (see page 6638)

MilliSecondOf (see page 6656)

MinuteOf (see page 6663)

MonthOf (see page 6670)

RecodeDateDay

SecondOf (see page 6682)

WeekOf (see page 6703)

YearOf (see page 6714)

38.1.9 DateUtils.DayOfTheMonth Function

Returns the day of the month represented by a TDateTime value.

Pascal

```
function DayOfTheMonth(const AValue: TDateTime): Word;
```

C++

```
Word DayOfTheMonth(const TDateTime AValue);
```

File

DateUtils

Description

Call DayOfTheMonth to obtain the day of the month represented by a specified TDateTime value. DayOfTheMonth returns a value between 1 and 31.

Note: DayOfTheMonth returns the same value as the DayOf (see page 6618) function.

See Also

DecodeDate

DayOfTheYear (see page 6620)

DayOfTheWeek (see page 6619)

DayOf (see page 6618)

DaysInAMonth (see page 6622)

HourOfTheMonth (see page 6640)

MilliSecondOfTheMonth (see page 6659)

MinuteOfTheMonth (see page 6666)

SecondOfTheMonth (see page 6685)

WeekOfTheMonth (see page 6704)

StartOfTheMonth (see page 6692)

DaysBetween (see page 6621)

38.1.10 DateUtils.DayOfTheWeek Function

Returns the day of the week represented by a TDateTime value.

Pascal

```
function DayOfTheWeek(const AValue: TDateTime): Word;
```

C++

```
Word DayOfTheWeek(const TDateTime AValue);
```

File

DateUtils

Description

Call DayOfTheWeek to obtain the day of the week represented by a specified TDateTime value. DayOfTheWeek returns a value between 1 and 7, where 1 indicates Monday and 7 indicates Sunday.

Note: DayOfTheWeek is ISO 8601 compliant (where Monday is considered the first day of the week). To obtain the day of the week where Sunday is considered the first day of the week, use the DayOfWeek function instead.

Tip: To make the return value more readable, use the Day of week constants.

See Also

DecodeDate

DayOfTheYear (see page 6620)

DayOfTheMonth (see page 6618)

DayOfWeek

HourOfTheWeek (see page 6641)

MilliSecondOfTheWeek (see page 6661)

MinuteOfTheWeek (see page 6666)

SecondOfTheWeek (see page 6686)

StartOfTheWeek (see page 6693)

DaysBetween (see page 6621)

DecodeDateWeek (see page 6627)

DecodeDayOfWeekInMonth (see page 6628)

38.1.11 DateUtils.DayOfTheYear Function

Returns the number of days between a specified TDateTime value and December 31 of the previous year.

Pascal

```
function DayOfTheYear(const AValue: TDateTime): Word;
```

C++

```
Word DayOfTheYear(const TDateTime AValue);
```

File

DateUtils

Description

Call DayOfTheYear to obtain the ordinal position of a TDateTime value's day within its year. Thus, a TDateTime value that refers to January 1 generates a result of 1, dates on January 2 return 2, dates on February 1 return 32, and so on.

See Also

DayOfTheMonth (see page 6618)

DayOfTheWeek (see page 6619)

DaysInYear (see page 6623)

DaysInAMonth (see page 6622)

HourOfTheYear (see page 6641)

MilliSecondOfTheYear (see page 6661)

MinuteOfTheYear (see page 6667)

MonthOfTheYear (see page 6670)

SecondOfTheYear (see page 6687)

WeekOfTheYear (see page 6705)

StartOfTheYear (see page 6693)

DaysBetween (see page 6621)

DecodeDateDay (see page 6624)

38.1.12 DateUtils.DaysBetween Function

Returns the number of whole days between two specified TDateTime values.

Pascal

```
function DaysBetween(const ANow: TDateTime; const AThen: TDateTime): Integer;
```

C++

```
int DaysBetween(const TDateTime ANow, const TDateTime AThen);
```

File

DateUtils

Description

Call DaysBetween to obtain the difference, in days, between two TDateTime values.

DaysBetween counts only whole days. Thus, DaysBetween reports the difference between Dec 31, 1999 11:59 PM and Jan 1, 2000 11:58 PM as 0 because the difference is one minute short of an entire day.

See Also

DaySpan (see page 6624)

HoursBetween (see page 6642)

MillisecondsBetween

MinutesBetween (see page 6668)

MonthsBetween (see page 6671)

SecondsBetween (see page 6687)

WeeksBetween (see page 6706)

YearsBetween (see page 6714)

WithinPastDays (see page 6708)

38.1.13 DateUtils.DaysInAMonth Function

Returns the number of days in a specified month of a specified year.

Pascal

```
function DaysInAMonth(const AYear: Word; const AMonth: Word): Word;
```

C++

```
Word DaysInAMonth(const Word AYear, const Word AMonth);
```

File

DateUtils

Description

Call DaysInAMonth to obtain the number of days in the specified month of the specified year.

AYear is a year between 1 and 9999 (inclusive).

AMonth is a month between 1 and 12 (inclusive).

See Also

DaysInAYear (see page 6622)

DaysInMonth (see page 6623)

DaysBetween (see page 6621)

DayOfTheMonth (see page 6618)

IsLeapYear

38.1.14 DateUtils.DaysInAYear Function

Returns the number of days in a specified year.

Pascal

```
function DaysInAYear(const AYear: Word): Word;
```

C++

```
Word DaysInAYear(const Word AYear);
```

File

DateUtils

Description

Call DaysInAYear to obtain the number of days in the year specified by AYear. AYear is a year between 1 and 9999 (inclusive).

See Also

DaysInYear (see page 6623)

DaysInAMonth (see page 6622)

DaysBetween (see page 6621)

DayOfTheYear (see page 6620)

IsLeapYear

WeeksInAYear (see page 6706)

38.1.15 DateUtils.DaysInMonth Function

Returns the number of days in the month of a specified TDateTime value.

Pascal

```
function DaysInMonth(const AValue: TDateTime): Word;
```

C++

```
Word DaysInMonth(const TDateTime AValue);
```

File

DateUtils

Description

Call DaysInMonth to obtain the number of days in the month of the TDateTime value specified by AValue.

See Also

DaysInYear (see page 6623)

DaysInAMonth (see page 6622)

DaysBetween (see page 6621)

DayOfTheMonth (see page 6618)

IsInLeapYear (see page 6650)

StartOfTheMonth (see page 6692)

EndOfTheMonth (see page 6637)

38.1.16 DateUtils.DaysInYear Function

Returns the number of days in the year of a specified TDateTime value.

Pascal

```
function DaysInYear(const AValue: TDateTime): Word;
```

C++

```
Word DaysInYear(const TDateTime AValue);
```

File

DateUtils

Description

Call DaysInYear to obtain the number of days in the year of the TDateTime value specified by AValue.

See Also

DaysInAYear (see page 6622)

DaysInMonth (see page 6623)

DaysBetween (see page 6621)

DayOfTheYear (see page 6620)

IsInLeapYear (see page 6650)

StartOfTheYear (see page 6693)

EndOfTheYear (see page 6638)

WeeksInYear (see page 6707)

38.1.17 DateUtils.DaySpan Function

Returns the number of days (including fractional days) between two specified TDateTime values.

Pascal

```
function DaySpan(const ANow: TDateTime; const AThen: TDateTime): Double;
```

C++

```
Double DaySpan(const TDateTime ANow, const TDateTime AThen);
```

File

DateUtils

Description

Call DaySpan to obtain the difference, in days, between two TDateTime values. Unlike the DaysBetween (see page 6621) function, which only counts whole days, DaySpan reports incomplete days as a fraction of an entire day.

See Also

DaysBetween (see page 6621)

HourSpan (see page 6643)

MillisecondSpan

MinuteSpan (see page 6668)

MonthSpan (see page 6672)

SecondSpan (see page 6688)

WeekSpan (see page 6708)

YearSpan (see page 6715)

38.1.18 DateUtils.DecodeDateDay Function

Returns the year and day of the year for a specified TDateTime.

Pascal

```
procedure DecodeDateDay(const AValue: TDateTime; out AYear: Word; out ADayOfYear: Word);
```

C++

```
DecodeDateDay(const TDateTime AValue, Word AYear, Word ADayOfYear);
```

File

DateUtils

Description

DecodeDateDay returns the day of the year for a specified date/time value, along with the year in which that day occurs.

AValue is the date/time value about which you want information.

AYear returns the year that AValue represents.

ADayOfYear returns the day within AYear that AValue represents, where 1 is January 1.

See Also

DecodeDateTime (see page 6626)

YearOf (see page 6714)

DecodeDate

DayOfTheYear (see page 6620)

DecodeDateWeek (see page 6627)

EncodeDateDay (see page 6629)

DecodeDateMonthWeek (see page 6625)

DecodeDayOfWeekInMonth (see page 6628)

38.1.19 DateUtils.DecodeDateMonthWeek Function

Returns the year, month, week of the month, and day of the week for a specified TDateTime.

Pascal

```
procedure DecodeDateMonthWeek(const AValue: TDateTime; out AYear: Word; out AMonth: Word; out
AWeekOfMonth: Word; out ADayOfWeek: Word);
```

C++

```
DecodeDateMonthWeek(const TDateTime AValue, Word AYear, Word AMonth, Word AWeekOfMonth, Word
ADayOfWeek);
```

File

DateUtils

Description

DecodeDateMonthWeek returns the day of the week and week of the month for a specified date/time value, along with the month and year in which that day and week occur.

AValue is the date/time value about which you want information.

AYear returns the year that AValue represents.

AMonth returns the month that AValue represents, where 1 is January and 12 is December.

AWeekOfMonth returns the week within AMonth that AValue represents, where 1 is the first week with four or more days.

ADayOfWeek returns the day within AWeekOfMonth that AValue represents, where 1 Monday and 7 is Sunday.

Note: The definitions for AWeekOfMonth and ADayOfWeek reflect the ISO 8601 standard.

Tip: To make the ADayOfWeek value more readable, use the Day of week constants.

See Also

DecodeDateTime (see page 6626)

YearOf (see page 6714)

MonthOf (see page 6670)

WeekOfTheMonth (see page 6704)

DecodeDate

DayOfTheWeek (see page 6619)

EncodeDateMonthWeek (see page 6629)

DecodeDateWeek (see page 6627)

DecodeDayOfWeekInMonth (see page 6628)

NthDayOfWeek (see page 6672)

38.1.20 DateUtils.DecodeDateTime Function

Returns Year, Month, Day, Hour, Minute, Second, and Millisecond values for a TDateTime.

Pascal

```
procedure DecodeDateTime(const AValue: TDateTime; out AYear: Word; out AMonth: Word; out ADay:
Word; out AHour: Word; out AMinute: Word; out ASecond: Word; out AMilliSecond: Word);
```

C++

```
DecodeDateTime(const TDateTime AValue, Word AYear, Word AMonth, Word ADay, Word AHour, Word
AMinute, Word ASecond, Word AMilliSecond);
```

File

DateUtils

Description

The DecodeDateTime procedure breaks the value specified as the AValue parameter into Year, Month, Day of the month, Hour, Minute, Second, and MilliSecond values and returns these as the AYear, AMonth, ADay, AHour, AMinute, ASecond, and AMilliSecond parameters.

See Also

DecodeTime

DayOf (see page 6618)

YearOf (see page 6714)

MonthOf (see page 6670)

DecodeDate

HourOf (see page 6638)

MinuteOf (see page 6663)

SecondOf (see page 6682)

MilliSecondOf (see page 6656)

DecodeDateDay (see page 6624)

DecodeDateWeek (see page 6627)

EncodeDateTime (see page 6630)

38.1.21 DateUtils.DecodeDateWeek Function

Returns the year, week of the year, and day of the week for a specified TDateTime value object.

Pascal

```
procedure DecodeDateWeek(const AValue: TDateTime; out AYear: Word; out AWeekOfYear: Word; out
ADayOfWeek: Word);
```

C++

```
DecodeDateWeek(const TDateTime AValue, Word AYear, Word AWeekOfYear, Word ADayOfWeek);
```

File

DateUtils

Description

DecodeDateWeek returns the day of the week and week of the year for a specified date/time value, along with the year in which that day and week occur.

AValue is the date/time value about which you want information.

AYear returns the year that AValue represents.

AWeekOfYear returns the week within AYear that AValue represents, where 1 is the first week with four or more days.

ADayOfWeek returns the day within AWeekOfYear that AValue represents, where 1 Monday and 7 is Sunday.

Note: The definitions for AWeekOfYear and ADayOfWeek follow the ISO 8601 standard.

Tip: To make the ADayOfWeek value more readable, use the Day of week constants.

See Also

DecodeDateTime (see page 6626)

YearOf (see page 6714)

WeekOfTheYear (see page 6705)

DecodeDate

DayOfTheWeek (see page 6619)

EncodeDateWeek (see page 6631)

DecodeDateMonthWeek (see page 6625)

DecodeDayOfWeekInMonth (see page 6628)

38.1.22 DateUtils.DecodeDayOfWeekInMonth Function

For a given TDateTime value, returns the year, month, day of the week, and the count of that day of the week in the month.

Pascal

```
procedure DecodeDayOfWeekInMonth(const AValue: TDateTime; out AYear: Word; out AMonth: Word;
out ANthDayOfWeek: Word; out ADayOfWeek: Word);
```

C++

```
DecodeDayOfWeekInMonth(const TDateTime AValue, Word AYear, Word AMonth, Word ANthDayOfWeek,
Word ADayOfWeek);
```

File

DateUtils

Description

For a given TDateTime value, DecodeDayOfWeekInMonth returns the day of the week and, for that day of the week, which occurrence it is within the month.

AValue is the date/time value about which you want information.

AYear returns the year that AValue represents.

AMonth returns the month in which AValue occurs. This value can be from 1 to 12, inclusive.

ANthDayOfWeek indicates the count for the day of the week represented by AValue. For example, if AValue represents the 2nd Tuesday of the month, ANthDayOfWeek returns 2. Note that this value may differ from the value that the WeekOfTheMonth (see page 6704) function returns, because DecodeDayOfWeekInMonth counts every occurrence of the given weekday, while WeekOfTheMonth (see page 6704) only counts a week if it includes 4 or more days in the month. Thus, for example, if AValue represents a Saturday that is the first day of a month, DecodeDayOfWeekInMonth returns 1 as the value of ANthDayOfWeek, while WeekOfTheMonth (see page 6704) returns 5 (or maybe 4), indicating the last week of the previous month.

ADayOfWeek returns the day of the week that AValue represents, where 1 Monday and 7 is Sunday.

Note: The definition for ADayOfWeek follows the ISO 8601 standard.

Tip: To make the ADayOfWeek value more readable, use the Day of week constants.

See Also

DecodeDateTime (see page 6626)

YearOf (see page 6714)

MonthOf (see page 6670)

EncodeDayOfWeekInMonth (see page 6632)

WeekOfTheMonth (see page 6704)

DecodeDate

DecodeDateMonthWeek (see page 6625)

DecodeDayOfWeekInMonth

38.1.23 DateUtils.EncodeDateDay Function

Returns a TDateTime that represents a specified day of the year for a specified year.

Pascal

```
function EncodeDateDay(const AYear: Word; const ADayOfYear: Word): TDateTime;
```

C++

```
TDateTime EncodeDateDay(const Word AYear, const Word ADayOfYear);
```

File

DateUtils

Description

EncodeDateDay returns a TDateTime for the date specified by the AYear and ADayOfYear parameters. The time portion of the return value is 0 (Midnight at the start of the specified day).

AYear is the year. It can be any value between 1 and 9999 (inclusive).

ADayOfYear is the day of the year, where 1 is January , 2 is January 2, 32 is February 1, and so on.

If AYear is not within range, or if ADayOfYear is less than 1 or greater than the number of days in AYear, EncodeDateDay raises an EConvertError exception.

See Also

EncodeDateTime (see page 6630)

YearOf (see page 6714)

EncodeDate

DayOfTheYear (see page 6620)

EncodeDateWeek (see page 6631)

DecodeDateDay (see page 6624)

EConvertError

TryEncodeDateDay (see page 6695)

EncodeDateMonthWeek (see page 6629)

38.1.24 DateUtils.EncodeDateMonthWeek Function

Returns a TDateTime that represents a specified day of a specified week in a specified month and year.

Pascal

```
function EncodeDateMonthWeek(const AYear: Word; const AMonth: Word; const AWeekOfMonth: Word;
const ADayOfWeek: Word): TDateTime;
```

C++

```
TDateTime EncodeDateMonthWeek(const Word AYear, const Word AMonth, const Word AWeekOfMonth,
const Word ADayOfWeek);
```

File

DateUtils

Description

EncodeDateMonthWeek returns a TDateTime for the date specified by the AYear, AMonth, AWeekOfMonth, and ADayOfWeek parameters. The time portion of the return value is 0 (Midnight at the start of the specified day).

AYear is the year. It can be any value between 1 and 9999 (inclusive).

AMonth is the month. It can be any value between 1 and 12 (inclusive).

AWeekOfMonth is the week within that month, where 1 is the first week that includes four or more days. Note that if the first calendar day of the month is a Friday, Saturday, or Sunday, then those three days must be expressed using AMonth set to the previous month and AWeekOfMonth set to the number of weeks in the previous month. Similarly, if the last calendar day of the month is a Monday, Tuesday, or Wednesday, then those three days are expressed with AMonth set to the following month and AWeekOfMonth set to 1.

ADayOfWeek is the day of the week, where 1 Monday, 2 is Tuesday, and so on.

Tip: To make the ADayOfWeek value more readable, use the Day of week constants.

If any of the parameters are not in range, EncodeDateMonthWeek raises an EConvertError exception.

Note: The definitions for AWeekOfMonth and ADayOfWeek reflect the ISO 8601 standard.

See Also

EncodeDateTime (see page 6630)

YearOf (see page 6714)

MonthOf (see page 6670)

EncodeDate

DayOfTheWeek (see page 6619)

DecodeDateMonthWeek (see page 6625)

EncodeDateDay (see page 6629)

EncodeDateWeek (see page 6631)

EConvertError

TryEncodeDateWeek (see page 6698)

EndOfAWeek (see page 6635)

38.1.25 DateUtils.EncodeDateTime Function

Returns a TDateTime that represents a specified year, month, day, hour, minute, second, and millisecond.

Pascal

```
function EncodeDateTime(const AYear: Word; const AMonth: Word; const ADay: Word; const AHour:
```

```
Word; const AMinute: Word; const ASecond: Word; const AMilliSecond: Word): TDateTime;
```

C++

```
TDateTime EncodeDateTime(const Word AYear, const Word AMonth, const Word ADay, const Word
AHour, const Word AMinute, const Word ASecond, const Word AMilliSecond);
```

File

DateUtils

Description

EncodeDateTime returns a TDateTime from the values specified as the AYear, AMonth, ADay, AHour, AMinute, ASecond, and AMilliSecond parameters.

The year must be between 1 and 9999.

Valid month values are 1 through 12.

Valid day values are 1 through 28, 29, 30, or 31, depending on the month value. For example, the possible day values for month 2 (February) are 1 through 28 or 1 through 29, depending on whether or not the year value specifies a leap year.

Valid hour values are 0 through 24. (If the specified hour is 24, the minute, second, and millisecond values should all be 0, and the resulting TDateTime value represents midnight at the end of the specified day and the beginning of the next day).

Valid minute values are 0 through 59.

Valid second values are 0 through 59.

Valid millisecond values are 0 through 999.

If the specified values are not within range, EncodeDateTime raises an EConvertError exception.

See Also

EncodeTime

EncodeDate

YearOf (see page 6714)

MonthOf (see page 6670)

DayOf (see page 6618)

HourOf (see page 6638)

MinuteOf (see page 6663)

SecondOf (see page 6682)

MillisecondOf

EConvertError

EncodeDateDay (see page 6629)

EncodeDateWeek (see page 6631)

TryEncodeDateTime (see page 6697)

38.1.26 DateUtils.EncodeDateWeek Function

Returns a TDateTime that represents a specified day of a specified week in a specified year.

Pascal

```
function EncodeDateWeek(const AYear: Word; const AWeekOfYear: Word; const ADayOfWeek: Word =
1): TDateTime;
```

C++

```
TDateTime EncodeDateWeek(const Word AYear, const Word AWeekOfYear, const Word ADayOfWeek = 1);
```

File

DateUtils

Description

EncodeDateWeek returns a TDateTime for the date specified by the AYear, AWeekOfYear, and ADayOfWeek parameters. The time portion of the return value is 0 (Midnight at the start of the specified day).

AYear is the year. It can be any value between 1 and 9999 (inclusive).

AWeekOfYear is the week within that year, where 1 is the first week that includes four or more days. Note that if the first calendar day of the year is a Friday, Saturday, or Sunday, then those three days must be expressed using AYear set to the previous year and AWeekOfYear set to the number of weeks in the previous year. Similarly, if the last calendar day of the year is a Monday, Tuesday, or Wednesday, then those three days are expressed with AYear set to the following year and AWeekOfYear set to 1.

ADayOfWeek is the day of the week, where 1 Monday, 2 is Tuesday, and so on.

Tip: To make the ADayOfWeek value more readable, use the Day of week constants.

If AYear is not within range, AWeekOfYear is less than 1 or greater than the number of weeks in AYear, or ADayOfWeek is less than 1 or greater than 7, EncodeDateWeek raises an EConvertError exception.

Note: The definitions for AWeekOfYear and ADayOfWeek follow the ISO 8601 standard.

See Also

EncodeDateTime (see page 6630)

YearOf (see page 6714)

EncodeDate

DayOfTheWeek (see page 6619)

DecodeDateWeek (see page 6627)

EncodeDateDay (see page 6629)

EConvertError

TryEncodeDateWeek (see page 6698)

EndOfAWeek (see page 6635)

EncodeDateMonthWeek (see page 6629)

38.1.27 DateUtils.EncodeDayOfWeekInMonth Function

Returns a TDateTime that represents a specified occurrence of a day of the week within a specified month and year.

Pascal

```
function EncodeDayOfWeekInMonth(const AYear: Word; const AMonth: Word; const ANthDayOfWeek:
Word; const ADayOfWeek: Word): TDateTime;
```

C++

```
TDateTime EncodeDayOfWeekInMonth(const Word AYear, const Word AMonth, const Word
ANthDayOfWeek, const Word ADayOfWeek);
```

File

DateUtils

Description

EncodeDayOfWeekInMonth returns a TDateTime for the date specified by the AYear, AMonth, ANthDayOfWeek, and ADayOfWeek parameters. The time portion of the return value is 0 (Midnight at the start of the specified day).

AYear is the year. It can be any value between 1 and 9999 (inclusive).

AMonth is the month. It can be any value between 1 and 12 (inclusive).

ANthDayOfWeek indicates the occurrence for the day of the week represented by ADayOfWeek. It can be any value between 1 and 5 (inclusive), as long as this value specifies a valid date. For example, if a month does not include 5 Mondays, setting ANthDayOfWeek to 5 and ADayOfWeek to 1 causes EncodeDayOfWeekInMonth to raise an exception.

ADayOfWeek is the day of the week, where 1 Monday and 7 is Sunday.

Note: The definition for ADayOfWeek follows the ISO 8601 standard.

Tip: To make the ADayOfWeek value more readable, use the Day of week constants.

See Also

EncodeDateTime (see page 6630)

YearOf (see page 6714)

MonthOf (see page 6670)

DecodeDayOfWeekInMonth (see page 6628)

EncodeDate

EncodeDateMonthWeek (see page 6629)

WeekOfTheMonth (see page 6704)

TryEncodeDayOfWeekInMonth (see page 6699)

38.1.28 DateUtils.EndOfADay Function

Returns a TDateTime that represents the last millisecond of a specified day.

Pascal

```
function EndOfADay(const AYear: Word; const AMonth: Word; const ADay: Word): TDateTime;
overload;
function EndOfADay(const AYear: Word; const ADayOfYear: Word): TDateTime; overload;
```

C++

```
TDateTime EndOfADay(const Word AYear, const Word AMonth, const Word ADay);
TDateTime EndOfADay(const Word AYear, const Word ADayOfYear);
```

File

DateUtils

Description

EndOfADay returns the last expressible moment (11:59:59.999 PM) of a specified day.

The AYear parameter specifies the year of the desired day.

The ADayOfYear parameter specifies the desired day as a day of the year, where 1 is January 1, 2 is January 2, 32 is February 1, and so on.

The AMonth and ADay parameters specify the desired day as a month of the year and day of the month. AMonth can range from 1 to 12. ADay can range from 1 to 28, 29, 30, or 31, depending on the values of AYear and AMonth.

If the parameters do not specify a valid date, EndOfADay raises an EConvertError exception.

See Also

EndOfTheDay (see page 6636)

EndOfAMonth (see page 6634)

EndOfAYear (see page 6636)

EndOfAWeek (see page 6635)

ReplaceTime

StartOfADay (see page 6689)

EncodeDateDay (see page 6629)

EncodeDate

38.1.29 DateUtils.EndOfAMonth Function

Returns a TDateTime that represents the last millisecond of the last day of a specified month.

Pascal

```
function EndOfAMonth(const AYear: Word; const AMonth: Word): TDateTime;
```

C++

```
TDateTime EndOfAMonth(const Word AYear, const Word AMonth);
```

File

DateUtils

Description

EndOfAMonth returns the last expressible moment (11:59:59.999 PM) of the last day of a specified month.

The AYear parameter specifies the year of the desired month.

The AMonth parameter specifies the month. It can range from 1 to 12.

If the parameters do not specify a valid month, EndOfAMonth raises an EConvertError exception.

See Also

EndOfTheMonth (see page 6637)

EndOfADay (see page 6633)

EndOfAYear (see page 6636)

EndOfAWeek (see page 6635)

ReplaceTime

StartOfAMonth (see page 6690)

EncodeDate

38.1.30 DateUtils.EndOfAWeek Function

Returns a TDateTime object value that represents the last millisecond of a specified day of a specified week.

Pascal

```
function EndOfAWeek(const AYear: Word; const AWeekOfYear: Word; const ADayOfWeek: Word = 7):
TDateTime;
```

C++

```
TDateTime EndOfAWeek(const Word AYear, const Word AWeekOfYear, const Word ADayOfWeek = 7);
```

File

DateUtils

Description

EndOfAWeek returns the last expressible moment (11:59:59.999 PM) of the specified day of the specified week.

The AYear parameter specifies the year of the desired day.

The AWeekOfYear parameter specifies the week of the year, where 1 is the first week in AYear that includes four or more days.

The ADayOfWeek parameter indicates the desired day in the specified week, where 1 is Monday, 2 is Tuesday, and so on.

Tip: To make the ADayOfWeek value more readable, use the Day of week constants.

If the parameters do not specify a valid date, EndOfAWeek raises an EConvertError exception.

Note: The definitions for AWeekOfYear and ADayOfWeek follow the ISO 8601 standard.

See Also

EndOfTheWeek (see page 6637)

EndOfADay (see page 6633)

EndOfAYear (see page 6636)

EndOfAMonth (see page 6634)

ReplaceTime

StartOfAWeek (see page 6690)

EncodeDateWeek (see page 6631)

38.1.31 DateUtils.EndOfAYear Function

Returns a TDateTime that represents the last millisecond of a specified year.

Pascal

```
function EndOfAYear(const AYear: Word): TDateTime;
```

C++

```
TDateTime EndOfAYear(const Word AYear);
```

File

DateUtils

Description

EndOfAYear returns the last expressible moment (December 31 at 11:59:59.999 PM) of the year specified by the AYear parameter.

See Also

EndOfTheYear (see page 6638)

EndOfADay (see page 6633)

EndOfAWeek (see page 6635)

EndOfAMonth (see page 6634)

ReplaceTime

StartOfAYear (see page 6691)

38.1.32 DateUtils.EndOfTheDay Function

Returns a TDateTime that represents the last millisecond of the day identified by a specified TDateTime.

Pascal

```
function EndOfTheDay(const AValue: TDateTime): TDateTime;
```

C++

```
TDateTime EndOfTheDay(const TDateTime AValue);
```

File

DateUtils

Description

EndOfTheDay returns the last expressible moment of the same day as the TDateTime specified by AValue. That is, it replaces the time portion of AValue with 11:59:59.999 PM and returns the result.

See Also

StartOfTheDay (see page 6692)

EndOfTheMonth (see page 6637)

EndOfTheYear (see page 6638)

EndOfTheWeek (see page 6637)

ReplaceTime

EndOfADay (see page 6633)

38.1.33 DateUtils.EndOfTheMonth Function

Returns a TDateTime that represents the last millisecond of the last day of the month identified by a specified TDateTime.

Pascal

```
function EndOfTheMonth(const AValue: TDateTime): TDateTime;
```

C++

```
TDateTime EndOfTheMonth(const TDateTime AValue);
```

File

DateUtils

Description

EndOfTheMonth returns the last expressible moment of the same month as the TDateTime specified by AValue. That is, it replaces the time portion of AValue with 11:59:59.999 PM, changes the day to the last day of the month, and returns the result.

See Also

StartOfTheMonth (see page 6692)

EndOfTheDay (see page 6636)

EndOfTheYear (see page 6638)

EndOfTheWeek (see page 6637)

EndOfAMonth (see page 6634)

38.1.34 DateUtils.EndOfTheWeek Function

Returns a TDateTime that represents the last millisecond of the last day of the week identified by a specified TDateTime.

Pascal

```
function EndOfTheWeek(const AValue: TDateTime): TDateTime;
```

C++

```
TDateTime EndOfTheWeek(const TDateTime AValue);
```

File

DateUtils

Description

EndOfTheWeek returns the last expressible moment of the same week as the TDateTime specified by AValue. That is, it replaces the time portion of AValue with 11:59:59.999 PM, changes the day to the last day of the week, and returns the result.

Note: EndOfTheWeek defines the week of AValue according to the ISO 8601 standard. That is, the week starts on Monday and ends on Sunday.

See Also

StartOfTheWeek (see page 6693)

EndOfTheDay (see page 6636)

EndOfTheYear (see page 6638)

EndOfAWeek (see page 6635)

EndOfTheMonth (see page 6637)

38.1.35 DateUtils.EndOfTheYear Function

Returns a TDateTime that represents the last millisecond of the last day of the year identified by a specified TDateTime.

Pascal

```
function EndOfTheYear(const AValue: TDateTime): TDateTime;
```

C++

```
TDateTime EndOfTheYear(const TDateTime AValue);
```

File

DateUtils

Description

EndOfTheYear returns the last expressible moment of the same year as the TDateTime specified by AValue. That is, it replaces the time portion of AValue with 11:59:59.999 PM, changes the day to December 31, and returns the result.

See Also

StartOfTheYear (see page 6693)

EndOfTheDay (see page 6636)

EndOfTheMonth (see page 6637)

EndOfTheWeek (see page 6637)

EndOfAYear (see page 6636)

38.1.36 DateUtils.HourOf Function

Returns the hour of the day represented by a TDateTime value.

Pascal

```
function HourOf(const AValue: TDateTime): Word;
```

C++

```
Word HourOf(const TDateTime AValue);
```

File

DateUtils

Description

Call HourOf to obtain the hour of the day represented by a specified TDateTime value. HourOf returns a value between 0 and 23.

Note: HourOf returns the same value as the HourOfTheDay (see page 6639) function.

See Also

DecodeTime

HourOfTheDay (see page 6639)

HourOfTheYear (see page 6641)

HourOfTheMonth (see page 6640)

HourOfTheWeek (see page 6641)

DayOf (see page 6618)

MilliSecondOf (see page 6656)

MinuteOf (see page 6663)

MonthOf (see page 6670)

RecodeDateHour

SecondOf (see page 6682)

WeekOf (see page 6703)

YearOf (see page 6714)

38.1.37 DateUtils.HourOfTheDay Function

Returns the hour of the day represented by a TDateTime value.

Pascal

```
function HourOfTheDay(const AValue: TDateTime): Word;
```

C++

```
Word HourOfTheDay(const TDateTime AValue);
```

File

DateUtils

Description

Call HourOfTheDay to obtain the hour of the day represented by a specified TDateTime value. HourOf (see page 6638) returns a value between 0 and 23.

Note: HourOfTheDay returns the same value as the HourOf (see page 6638) function.

See Also

HourOf (see page 6638)

HourOfTheYear (see page 6641)

HourOfTheMonth (see page 6640)

HourOfTheWeek (see page 6641)

HourOfTheDay

MilliSecondOfTheDay (see page 6657)

MinuteOfTheDay (see page 6664)

SecondOfTheDay (see page 6683)

HoursBetween (see page 6642)

38.1.38 DateUtils.HourOfTheMonth Function

Returns the number of hours between a specified TDateTime value and 12:00 AM of the first day of the month.

Pascal

```
function HourOfTheMonth(const AValue: TDateTime): Word;
```

C++

```
Word HourOfTheMonth(const TDateTime AValue);
```

File

DateUtils

Description

Call HourOfTheMonth to obtain the number of complete hours between AValue and 12:00 AM on the first day of the month that includes AValue.

See Also

HourOfTheDay (see page 6639)

HourOfTheWeek (see page 6641)

HourOfTheYear (see page 6641)

DayOfTheMonth (see page 6618)

MilliSecondOfTheMonth (see page 6659)

MinuteOfTheMonth (see page 6666)

SecondOfTheMonth (see page 6685)

WeekOfTheMonth (see page 6704)

StartOfTheMonth (see page 6692)

HoursBetween (see page 6642)

38.1.39 DateUtils.HourOfTheWeek Function

Returns the number of hours between a specified TDateTime value and 12:00 AM of the first day of the week.

Pascal

```
function HourOfTheWeek(const AValue: TDateTime): Word;
```

C++

```
Word HourOfTheWeek(const TDateTime AValue);
```

File

DateUtils

Description

Call HourOfTheWeek to obtain the number of complete hours between AValue and 12:00 AM on Monday of the week that includes AValue.

Note: HourOfTheWeek defines the week of AValue according to the ISO 8601 standard. That is, the week starts on Monday and ends on Sunday.

See Also

HourOfTheDay (see page 6639)

HourOfTheMonth (see page 6640)

HourOfTheYear (see page 6641)

DayOfWeek

MilliSecondOfTheWeek (see page 6661)

MinuteOfTheWeek (see page 6666)

SecondOfTheWeek (see page 6686)

StartOfTheWeek (see page 6693)

HoursBetween (see page 6642)

38.1.40 DateUtils.HourOfTheYear Function

Returns the number of hours between a specified TDateTime value and 12:00 AM of the first day of the year.

Pascal

```
function HourOfTheYear(const AValue: TDateTime): Word;
```

C++

```
Word HourOfTheYear(const TDateTime AValue);
```

File

DateUtils

Description

Call HourOfTheYear to obtain the number of complete hours between AValue and 12:00 AM on Jan 1 of the year that includes AValue.

See Also

HourOfTheDay (see page 6639)

HourOfTheWeek (see page 6641)

HourOfTheMonth (see page 6640)

DayOfTheYear (see page 6620)

MilliSecondOfTheYear (see page 6661)

MinuteOfTheYear (see page 6667)

MonthOfTheYear (see page 6670)

SecondOfTheYear (see page 6687)

WeekOfTheYear (see page 6705)

StartOfTheYear (see page 6693)

HoursBetween (see page 6642)

38.1.41 DateUtils.HoursBetween Function

Returns the number of whole hours between two specified TDateTime values.

Pascal

```
function HoursBetween(const ANow: TDateTime; const AThen: TDateTime): Int64;
```

C++

```
Int64 HoursBetween(const TDateTime ANow, const TDateTime AThen);
```

File

DateUtils

Description

Call HoursBetween to obtain the difference, in hours, between two TDateTime values. HoursBetween counts only entire hours. Thus, HoursBetween reports the difference between 9:00 AM and 9:59:59 AM as 0 because the difference is one second short of an entire hour.

See Also

HourSpan (see page 6643)

DaysBetween (see page 6621)

MillisecondsBetween

MinutesBetween (see page 6668)

MonthsBetween (see page 6671)

SecondsBetween (see page 6687)

WeeksBetween (see page 6706)

YearsBetween (see page 6714)

WithinPastHours (see page 6709)

38.1.42 DateUtils.HourSpan Function

Returns the number of Hours (including fractional Hours) between two specified TDateTime values.

Pascal

```
function HourSpan(const ANow: TDateTime; const AThen: TDateTime): Double;
```

C++

```
Double HourSpan(const TDateTime ANow, const TDateTime AThen);
```

File

DateUtils

Description

Call HourSpan to obtain the difference, in hours, between two TDateTime values. Unlike the HoursBetween (see page 6642) function, which only counts entire hours, HourSpan reports incomplete hours as a fraction of an entire hour.

See Also

HoursBetween (see page 6642)

DaySpan (see page 6624)

MillisecondSpan

MinuteSpan (see page 6668)

MonthSpan (see page 6672)

SecondSpan (see page 6688)

WeekSpan (see page 6708)

YearSpan (see page 6715)

38.1.43 DateUtils.IncDay Function

Returns a date shifted by a specified number of days.

Pascal

```
function IncDay(const AValue: TDateTime; const ANumberOfDays: Integer = 1): TDateTime;
```

C++

```
TDateTime IncDay(const TDateTime AValue, const int ANumberOfDays = 1);
```

File

DateUtils

Description

IncDay returns the value of the AValue parameter, incremented by ANumberOfDays days. ANumberOfDays can be negative, to return a date N days previous.

The time of day specified by the AValue parameter is copied to the result.

See Also

DecodeDate

EncodeDate

ReplaceDate

IncMonth

IncHour (see page 6644)

IncMilliSecond (see page 6645)

IncMinute (see page 6645)

IncSecond (see page 6646)

IncWeek (see page 6647)

IncYear (see page 6647)

38.1.44 DateUtils.IncHour Function

Returns a date/time value shifted by a specified number of hours.

Pascal

```
function IncHour(const AValue: TDateTime; const ANumberOfHours: Int64 = 1): TDateTime;
```

C++

```
TDateTime IncHour(const TDateTime AValue, const Int64 ANumberOfHours = 1);
```

File

DateUtils

Description

IncHour returns the value of the AValue parameter, incremented by ANumberOfHours hours. ANumberOfHours can be negative, to return a date and time N hours prior to AValue.

See Also

DecodeDate

EncodeDate

ReplaceDate

IncMonth

IncDay (see page 6643)

IncMilliSecond (see page 6645)

IncMinute (see page 6645)

IncSecond (see page 6646)

IncWeek (see page 6647)

IncYear (see page 6647)

38.1.45 DateUtils.IncMilliSecond Function

Returns a date/time value shifted by a specified number of milliseconds.

Pascal

```
function IncMilliSecond(const AValue: TDateTime; const ANumberOfMilliSeconds: Int64 = 1):
TDateTime;
```

C++

```
TDateTime IncMilliSecond(const TDateTime AValue, const Int64 ANumberOfMilliSeconds = 1);
```

File

DateUtils

Description

IncMilliSecond returns the value of the AValue parameter, incremented by ANumberOfMilliSeconds milliseconds. ANumberOfMilliSeconds can be negative, to return a date and time N milliseconds prior to AValue.

See Also

DecodeDate

EncodeDate

ReplaceDate

IncMonth

IncDay (see page 6643)

IncHour (see page 6644)

IncMinute (see page 6645)

IncSecond (see page 6646)

IncWeek (see page 6647)

IncYear (see page 6647)

38.1.46 DateUtils.IncMinute Function

Returns a date/time value shifted by a specified number of minutes.

Pascal

```
function IncMinute(const AValue: TDateTime; const ANumberOfMinutes: Int64 = 1): TDateTime;
```

C++

```
TDateTime IncMinute(const TDateTime AValue, const Int64 ANumberOfMinutes = 1);
```

File

DateUtils

Description

IncMinute returns the value of the AValue parameter, incremented by ANumberOfMinutes minutes. ANumberOfMinutes can be negative, to return a date and time N minutes prior to AValue.

See Also

DecodeDate

EncodeDate

ReplaceDate

IncMonth

IncDay (see page 6643)

IncHour (see page 6644)

IncMilliSecond (see page 6645)

IncSecond (see page 6646)

IncWeek (see page 6647)

IncYear (see page 6647)

38.1.47 DateUtils.IncSecond Function

Returns a date/time value shifted by a specified number of seconds.

Pascal

```
function IncSecond(const AValue: TDateTime; const ANumberOfSeconds: Int64 = 1): TDateTime;
```

C++

```
TDateTime IncSecond(const TDateTime AValue, const Int64 ANumberOfSeconds = 1);
```

File

DateUtils

Description

IncSecond returns the value of the AValue parameter, incremented by ANumberOfSeconds seconds. ANumberOfSeconds can be negative, to return a date and time N seconds prior to AValue.

See Also

DecodeDate

EncodeDate

ReplaceDate

IncMonth

IncDay (see page 6643)

IncHour (see page 6644)

IncMilliSecond (see page 6645)

IncMinute (see page 6645)

IncWeek (see page 6647)

IncYear (see page 6647)

38.1.48 DateUtils.IncWeek Function

Returns a date shifted by a specified number of weeks.

Pascal

```
function IncWeek(const AValue: TDateTime; const ANumberOfWeeks: Integer = 1): TDateTime;
```

C++

```
TDateTime IncWeek(const TDateTime AValue, const int ANumberOfWeeks = 1);
```

File

DateUtils

Description

IncWeek returns the value of the AValue parameter, incremented by ANumberOfWeeks weeks. ANumberOfWeeks can be negative, to return a date N weeks previous.

The time of day specified by the AValue parameter is copied to the result.

See Also

DecodeDate

EncodeDate

ReplaceDate

IncMonth

IncHour (see page 6644)

IncMilliSecond (see page 6645)

IncMinute (see page 6645)

IncSecond (see page 6646)

IncDay (see page 6643)

IncYear (see page 6647)

38.1.49 DateUtils.IncYear Function

Returns a date shifted by a specified number of years.

Pascal

```
function IncYear(const AValue: TDateTime; const ANumberOfYears: Integer = 1): TDateTime;
```

C++

```
TDateTime IncYear(const TDateTime AValue, const int ANumberOfYears = 1);
```

File

DateUtils

Description

IncYear returns the value of the AValue parameter, incremented by ANumberOfYears years. ANumberOfYears can be negative, to return a date N years previous.

The time of day specified by the AValue parameter is copied to the result.

See Also

DecodeDate

EncodeDate

ReplaceDate

IncMonth

IncHour (see page 6644)

IncMilliSecond (see page 6645)

IncMinute (see page 6645)

IncSecond (see page 6646)

IncDay (see page 6643)

IncWeek (see page 6647)

38.1.50 DateUtils.InvalidDateDayError Function

Pascal

```
procedure InvalidDateDayError(const AYear: Word; const ADayOfYear: Word);
```

C++

```
InvalidDateDayError(const Word AYear, const Word ADayOfYear);
```

File

DateUtils

Description

This is function DateUtils.InvalidDateDayError.

38.1.51 DateUtils.InvalidDateMonthWeekError Function

Pascal

```
procedure InvalidDateMonthWeekError(const AYear: Word; const AMonth: Word; const AWeekOfMonth:
Word; const ADayOfWeek: Word);
```

C++

```
InvalidDateMonthWeekError(const Word AYear, const Word AMonth, const Word AWeekOfMonth, const
Word ADayOfWeek);
```

File

DateUtils

Description

This is function DateUtils.InvalidDateMonthWeekError.

38.1.52 DateUtils.InvalidDateTimeError Function

Pascal

```
procedure InvalidDateTimeError(const AYear: Word; const AMonth: Word; const ADay: Word; const
AHour: Word; const AMinute: Word; const ASecond: Word; const AMilliSecond: Word; const
ABaseDate: TDateTime = 0);
```

C++

```
InvalidDateTimeError(const Word AYear, const Word AMonth, const Word ADay, const Word AHour,
const Word AMinute, const Word ASecond, const Word AMilliSecond, const TDateTime ABaseDate =
0);
```

File

DateUtils

Description

This is function DateUtils.InvalidDateTimeError.

38.1.53 DateUtils.InvalidDateWeekError Function

Pascal

```
procedure InvalidDateWeekError(const AYear: Word; const AWeekOfYear: Word; const ADayOfWeek:
Word);
```

C++

```
InvalidDateWeekError(const Word AYear, const Word AWeekOfYear, const Word ADayOfWeek);
```

File

DateUtils

Description

This is function DateUtils.InvalidDateWeekError.

38.1.54 DateUtils.InvalidDayOfWeekInMonthError Function

Pascal

```
procedure InvalidDayOfWeekInMonthError(const AYear: Word; const AMonth: Word; const
ANthDayOfWeek: Word; const ADayOfWeek: Word);
```

C++

```
InvalidDayOfWeekInMonthError(const Word AYear, const Word AMonth, const Word ANthDayOfWeek,
const Word ADayOfWeek);
```

File

DateUtils

Description

This is function DateUtils.InvalidDayOfWeekInMonthError.

38.1.55 DateUtils.IsInLeapYear Function

Indicates whether a specified TDateTime value occurs in a leap year.

Pascal

```
function IsInLeapYear(const AValue: TDateTime): Boolean;
```

C++

```
Boolean IsInLeapYear(const TDateTime AValue);
```

File

DateUtils

Description

Call IsInLeapYear to determine whether the year of the date specified by AValue occurs in a leap year.

See Also

Date

IsLeapYear

38.1.56 DateUtils.IsPM Function

Indicates whether the time portion of a specified TDateTime value occurs after noon.

Pascal

```
function IsPM(const AValue: TDateTime): Boolean;
```

C++

```
Boolean IsPM(const TDateTime AValue);
```

File

DateUtils

Description

IsPM returns true if the time portion of AValue occurs on or after 12:00 noon and before 12:00 midnight.

See Also

HourOf (see page 6638)

38.1.57 DateUtils.IsSameDay Function

Indicates whether a specified TDateTime value occurs on a the same day as a criterion date.

Pascal

```
function IsSameDay(const AValue: TDateTime; const ABasis: TDateTime): Boolean;
```

C++

```
Boolean IsSameDay(const TDateTime AValue, const TDateTime ABasis);
```

File

DateUtils

Description

IsSameDay returns true if AValue occurs on the same day as ABasis. The time portions of AValue and ABasis can differ.

IsSameDay returns false if the date portions of AValue and ABasis differ.

See Also

IsToday (see page 6651)

CompareDate (see page 6614)

CompareDateTime (see page 6614)

SameDate (see page 6681)

SameDateTime (see page 6681)

38.1.58 DateUtils.IsToday Function

Indicates whether a specified TDateTime value occurs on the current date.

Pascal

```
function IsToday(const AValue: TDateTime): Boolean;
```

C++

```
Boolean IsToday(const TDateTime AValue);
```

File

DateUtils

Description

IsToday returns true if AValue occurs on the current day.

IsToday returns false if AValue represents a time on any other day.

See Also

IsSameDay (see page 6651)

Today (see page 6694)

Date

38.1.59 DateUtils.IsValidDate Function

Indicates whether a specified year, month, and day represent a valid date.

Pascal

```
function IsValidDate(const AYear: Word; const AMonth: Word; const ADay: Word): Boolean;
```

C++

```
Boolean IsValidDate(const Word AYear, const Word AMonth, const Word ADay);
```

File

DateUtils

Description

IsValidDate returns true if

AYear falls in the range from 1 to 9999 inclusive.

AMonth falls in the range from 1 to 12 inclusive.

ADay falls in the range from1 to the number of days in the specified month.

Otherwise, IsValidDate returns false.

See Also

IsValidDateDay (see page 6652)

IsValidDateTime (see page 6654)

IsValidDateWeek (see page 6654)

IsValidTime (see page 6655)

38.1.60 DateUtils.IsValidDateDay Function

Indicates whether a specified year and day of the year represent a valid date.

Pascal

```
function IsValidDateDay(const AYear: Word; const ADayOfYear: Word): Boolean;
```

C++

```
Boolean IsValidDateDay(const Word AYear, const Word ADayOfYear);
```

File

DateUtils

Description

IsValidDateDay returns true if

AYear falls in the range from 1 to 9999 inclusive.

ADay falls in the range from1 to the number of days in the specified year.

Otherwise, IsValidDateDay returns false.

See Also

IsValidDate (see page 6652)

IsValidDateTime (see page 6654)

IsValidDateWeek (see page 6654)

IsValidTime (see page 6655)

38.1.61 DateUtils.IsValidDateMonthWeek Function

Indicates whether a specified year, month, week of the month, and day of the week represent a valid date.

Pascal

```
function IsValidDateMonthWeek(const AYear: Word; const AMonth: Word; const AWeekOfMonth: Word;
const ADayOfWeek: Word): Boolean;
```

C++

```
Boolean IsValidDateMonthWeek(const Word AYear, const Word AMonth, const Word AWeekOfMonth,
const Word ADayOfWeek);
```

File

DateUtils

Description

IsValidDateMonthWeek returns true if

AYear falls in the range from 1 to 9999 inclusive.

AMonth falls in the range from 1 to 12 inclusive.

AWeekOfMonth falls in the range from1 to the number of weeks in the specified month.

ADayOfWeek falls in the range from 1 to 7.

Otherwise, IsValidDateMonthWeek returns false.

See Also

IsValidDate (see page 6652)

IsValidDateTime (see page 6654)

IsValidDateDay (see page 6652)

IsValidTime (see page 6655)

38.1.62 DateUtils.IsValidDateTime Function

Indicates whether a specified year, month, day, hour, minute, second, and millisecond represent a valid date and time.

Pascal

```
function IsValidDateTime(const AYear: Word; const AMonth: Word; const ADay: Word; const AHour:
Word; const AMinute: Word; const ASecond: Word; const AMilliSecond: Word): Boolean;
```

C++

```
Boolean IsValidDateTime(const Word AYear, const Word AMonth, const Word ADay, const Word
AHour, const Word AMinute, const Word ASecond, const Word AMilliSecond);
```

File

DateUtils

Description

IsValidDateTime returns true if

AYear falls in the range from 1 to 9999 inclusive.

AMonth falls in the range from 1 to 12 inclusive.

ADay falls in the range from1 to the number of days in the specified month.

AHour falls in the range from 0 to 24, and if AHour is 24, then AMinute, ASecond, and AMilliSecond must all be 0.

AMinute falls in the range from 0 to 59 inclusive.

ASecond falls in the range from 0 to 59 inclusive.

AMilliSecond falls in the range from 0 to 999 inclusive.

Otherwise, IsValidDateTime returns false.

See Also

IsValidDate (see page 6652)

IsValidDateDay (see page 6652)

IsValidDateWeek (see page 6654)

IsValidTime (see page 6655)

38.1.63 DateUtils.IsValidDateWeek Function

Indicates whether a specified year, week of the year, and day of the week represent a valid date.

Pascal

```
function IsValidDateWeek(const AYear: Word; const AWeekOfYear: Word; const ADayOfWeek: Word):
Boolean;
```

C++

```
Boolean IsValidDateWeek(const Word AYear, const Word AWeekOfYear, const Word ADayOfWeek);
```

File

DateUtils

Description

IsValidDateWeek returns true if

AYear falls in the range from 1 to 9999 inclusive.

AWeekOfYear falls in the range from1 to the number of weeks in the specified year.

ADayOfWeek falls in the range from 1 to 7.

Otherwise, IsValidDateWeek returns false.

See Also

IsValidDate (see page 6652)

IsValidDateTime (see page 6654)

IsValidDateDay (see page 6652)

IsValidTime (see page 6655)

38.1.64 DateUtils.IsValidTime Function

Indicates whether a specified hour, minute, second, and millisecond represent a valid date and time.

Pascal

```
function IsValidTime(const AHour: Word; const AMinute: Word; const ASecond: Word; const
AMilliSecond: Word): Boolean;
```

C++

```
Boolean IsValidTime(const Word AHour, const Word AMinute, const Word ASecond, const Word
AMilliSecond);
```

File

DateUtils

Description

IsValidTime returns true if

AHour falls in the range from 0 to 24, and if AHour is 24, then AMinute, ASecond, and AMilliSecond must all be 0.

AMinute falls in the range from 0 to 59 inclusive.

ASecond falls in the range from 0 to 59 inclusive.

AMilliSecond falls in the range from 0 to 999 inclusive.

Otherwise, IsValidTime returns false.

See Also

IsValidDate (see page 6652)

IsValidDateDay (see page 6652)

IsValidDateWeek (see page 6654)

IsValidDateTime (see page 6654)

38.1.65 DateUtils.JulianDateToDateTime Function

Converts a Julian date to a TDateTime value.

Pascal

```
function JulianDateToDateTime(const AValue: Double): TDateTime;
```

C++

```
TDateTime JulianDateToDateTime(const Double AValue);
```

File

DateUtils

Description

Call JulianDateToDateTime to convert a Julian date to the corresponding TDateTime value.

The Julian date is the number of days, including fractional days, since 4713 BC January 1, Greenwich noon. Julian dates are used by astronomers and others who need an unambiguous dating system based on continuing day counts.

Note: If JulianDateToDateTime can't convert the date specified by AValue, it raises an EConvertError exception.

See Also

DateTimeToModifiedJulianDate (see page 6617)

DateTimeToJulianDate (see page 6616)

ModifiedJulianDateToDateTime (see page 6669)

Convert

TryJulianDateToDateTime (see page 6700)

TryModifiedJulianDateToDateTime (see page 6701)

EConvertError

38.1.66 DateUtils.MilliSecondOf Function

Returns the millisecond of the second represented by a TDateTime value.

Pascal

```
function MilliSecondOf(const AValue: TDateTime): Word;
```

C++

```
Word MilliSecondOf(const TDateTime AValue);
```

File

DateUtils

Description

Call MilliSecondOf to obtain the millisecond of the second represented by a specified TDateTime value. MilliSecondOf returns a value between 0 and 999.

Note: MilliSecondOf returns the same value as the MilliSecondOfTheSecond (see page 6660) function.

See Also

DecodeTime

MilliSecondOfTheSecond (see page 6660)

MilliSecondOfTheDay (see page 6657)

MilliSecondOfTheHour (see page 6658)

MilliSecondOfTheMinute (see page 6659)

MilliSecondOfTheYear (see page 6661)

MilliSecondOfTheMonth (see page 6659)

MilliSecondOfTheWeek (see page 6661)

DayOf (see page 6618)

HourOf (see page 6638)

MinuteOf (see page 6663)

MonthOf (see page 6670)

RecodeDateMilliSecond

SecondOf (see page 6682)

WeekOf (see page 6703)

YearOf (see page 6714)

38.1.67 DateUtils.MilliSecondOfTheDay Function

Returns the number of milliseconds between a specified TDateTime value and the beginning of the same day.

Pascal

```
function MilliSecondOfTheDay(const AValue: TDateTime): LongWord;
```

C++

```
LongWord MilliSecondOfTheDay(const TDateTime AValue);
```

File

DateUtils

Description

Call MilliSecondOfTheDay to obtain the number of milliseconds between AValue and the beginning (midnight) on the same day.

See Also

MilliSecondOfTheSecond (see page 6660)

MilliSecondOfTheHour (see page 6658)

MilliSecondOfTheMinute (see page 6659)

MilliSecondOfTheYear (see page 6661)

MilliSecondOfTheMonth (see page 6659)

MilliSecondOfTheWeek (see page 6661)

HourOfTheDay (see page 6639)

MinuteOfTheDay (see page 6664)

SecondOfTheDay (see page 6683)

MilliSecondsBetween (see page 6662)

StartOfTheDay (see page 6692)

38.1.68 DateUtils.MilliSecondOfTheHour Function

Returns the number of milliseconds between a specified TDateTime value and the beginning of the same hour.

Pascal

```
function MilliSecondOfTheHour(const AValue: TDateTime): LongWord;
```

C++

```
LongWord MilliSecondOfTheHour(const TDateTime AValue);
```

File

DateUtils

Description

Call MilliSecondOfTheHour to obtain the number of milliseconds between AValue and the start of the same hour on the same day.

See Also

MilliSecondOfTheSecond (see page 6660)

MilliSecondOfTheDay (see page 6657)

MilliSecondOfTheMinute (see page 6659)

MilliSecondOfTheYear (see page 6661)

MilliSecondOfTheMonth (see page 6659)

MilliSecondOfTheWeek (see page 6661)

MinuteOfTheHour (see page 6665)

SecondOfTheHour (see page 6684)

MilliSecondsBetween (see page 6662)

38.1.69 DateUtils.MilliSecondOfTheMinute Function

Returns the number of milliseconds between a specified TDateTime value and the beginning of the same minute.

Pascal

```
function MilliSecondOfTheMinute(const AValue: TDateTime): LongWord;
```

C++

```
LongWord MilliSecondOfTheMinute(const TDateTime AValue);
```

File

DateUtils

Description

Call MilliSecondOfTheMinute to obtain the number of milliseconds between AValue and the start of the same minute on the same day.

See Also

MilliSecondOfTheSecond (see page 6660)

MilliSecondOfTheDay (see page 6657)

MilliSecondOfTheHour (see page 6658)

MilliSecondOfTheYear (see page 6661)

MilliSecondOfTheMonth (see page 6659)

MilliSecondOfTheWeek (see page 6661)

SecondOfTheMinute (see page 6685)

MilliSecondsBetween (see page 6662)

38.1.70 DateUtils.MilliSecondOfTheMonth Function

Returns the number of milliseconds between a specified TDateTime value and the beginning of the month.

Pascal

```
function MilliSecondOfTheMonth(const AValue: TDateTime): LongWord;
```

C++

```
LongWord MilliSecondOfTheMonth(const TDateTime AValue);
```

File

DateUtils

Description

Call MilliSecondOfTheMonth to obtain the number of milliseconds between the beginning (Midnight on the first day) of the month that contains AValue and AValue itself.

See Also

MilliSecondOfTheDay (see page 6657)

MilliSecondOfTheWeek (see page 6661)

MilliSecondOfTheYear (see page 6661)

MilliSecondOfTheSecond (see page 6660)

MilliSecondOfTheMinute (see page 6659)

MilliSecondOfTheHour (see page 6658)

DayOfTheMonth (see page 6618)

HourOfTheMonth (see page 6640)

MinuteOfTheMonth (see page 6666)

SecondOfTheMonth (see page 6685)

WeekOfTheMonth (see page 6704)

StartOfTheMonth (see page 6692)

MilliSecondsBetween (see page 6662)

38.1.71 DateUtils.MilliSecondOfTheSecond Function

Returns the millisecond of the second represented by a TDateTime value.

Pascal

```
function MilliSecondOfTheSecond(const AValue: TDateTime): Word;
```

C++

```
Word MilliSecondOfTheSecond(const TDateTime AValue);
```

File

DateUtils

Description

Call MilliSecondOfTheSecond to obtain the millisecond of the second represented by a specified TDateTime value. MilliSecondOfTheSecond returns a value between 0 and 999.

Note: MilliSecondOfTheSecond returns the same value as the MilliSecondOf (see page 6656) function.

See Also

MilliSecondOf (see page 6656)

MilliSecondOfTheDay (see page 6657)

MilliSecondOfTheHour (see page 6658)

MilliSecondOfTheMinute (see page 6659)

MilliSecondOfTheYear (see page 6661)

MilliSecondOfTheMonth (see page 6659)

MilliSecondOfTheWeek (see page 6661)

MilliSecondsBetween (see page 6662)

38.1.72 DateUtils.MilliSecondOfTheWeek Function

Returns the number of milliseconds between a specified TDateTime value and 12:00:00:00 AM of the first day of the week.

Pascal

```
function MilliSecondOfTheWeek(const AValue: TDateTime): LongWord;
```

C++

```
LongWord MilliSecondOfTheWeek(const TDateTime AValue);
```

File

DateUtils

Description

Call MilliSecondOfTheWeek to obtain the number of milliseconds between AValue and 12:00:00:00 AM on Monday of the week that includes AValue.

Note: MilliSecondOfTheWeek defines the week of AValue according to the ISO 8601 standard. That is, the week starts on Monday and ends on Sunday.

See Also

MilliSecondOfTheDay (see page 6657)

MilliSecondOfTheMonth (see page 6659)

MilliSecondOfTheYear (see page 6661)

MilliSecondOfTheSecond (see page 6660)

MilliSecondOfTheHour (see page 6658)

MilliSecondOfTheMinute (see page 6659)

DayOfWeek

HourOfTheWeek (see page 6641)

MinuteOfTheWeek (see page 6666)

SecondOfTheWeek (see page 6686)

StartOfTheWeek (see page 6693)

MilliSecondsBetween (see page 6662)

38.1.73 DateUtils.MilliSecondOfTheYear Function

Returns the number of milliseconds between a specified TDateTime value and 12:00:00:00 AM of the first day of the year.

Pascal

```
function MilliSecondOfTheYear(const AValue: TDateTime): Int64;
```

C++

```
Int64 MilliSecondOfTheYear(const TDateTime AValue);
```

File

DateUtils

Description

Call MilliSecondOfTheYear to obtain the number of milliseconds between AValue and 12:00:00:00 AM on Jan 1 of the year that includes AValue.

See Also

MilliSecondOfTheSecond (see page 6660)

MilliSecondOfTheMinute (see page 6659)

MilliSecondOfTheHour (see page 6658)

MilliSecondOfTheDay (see page 6657)

MilliSecondOfTheWeek (see page 6661)

MilliSecondOfTheMonth (see page 6659)

DayOfTheYear (see page 6620)

HourOfTheYear (see page 6641)

MinuteOfTheYear (see page 6667)

MonthOfTheYear (see page 6670)

SecondOfTheYear (see page 6687)

WeekOfTheYear (see page 6705)

StartOfTheYear (see page 6693)

MilliSecondsBetween (see page 6662)

38.1.74 DateUtils.MilliSecondsBetween Function

Returns the number of milliseconds between two specified TDateTime values.

Pascal

```
function MilliSecondsBetween(const ANow: TDateTime; const AThen: TDateTime): Int64;
```

C++

```
Int64 MilliSecondsBetween(const TDateTime ANow, const TDateTime AThen);
```

File

DateUtils

Description

Call MilliSecondsBetween to obtain the difference, in milliseconds, between two TDateTime values.

See Also

MilliSecondSpan (see page 6663)

DaysBetween (see page 6621)

HoursBetween (see page 6642)

MinutesBetween (see page 6668)

MonthsBetween (see page 6671)

SecondsBetween (see page 6687)

WeeksBetween (see page 6706)

YearsBetween (see page 6714)

WithinPastMilliSeconds (see page 6710)

38.1.75 DateUtils.MilliSecondSpan Function

Returns the number of milliseconds between two specified TDateTime values.

Pascal

```
function MilliSecondSpan(const ANow: TDateTime; const AThen: TDateTime): Double;
```

C++

```
Double MilliSecondSpan(const TDateTime ANow, const TDateTime AThen);
```

File

DateUtils

Description

Call MilliSecondSpan to obtain the difference, in milliseconds, between two TDateTime values. MilliSecondSpan returns the same value as MilliSecondsBetween (see page 6662), but as a floating point type instead of an integer.

See Also

MilliSecondsBetween (see page 6662)

DaySpan (see page 6624)

HourSpan (see page 6643)

MinuteSpan (see page 6668)

MonthSpan (see page 6672)

SecondSpan (see page 6688)

WeekSpan (see page 6708)

YearSpan (see page 6715)

38.1.76 DateUtils.MinuteOf Function

Returns the minute of the hour represented by a TDateTime value.

Pascal

```
function MinuteOf(const AValue: TDateTime): Word;
```

C++

```
Word MinuteOf(const TDateTime AValue);
```

File

DateUtils

Description

Call MinuteOf to obtain the minute of the hour represented by a specified TDateTime value. MinuteOf returns a value between 0 and 59.

Note: MinuteOf returns the same value as the MinuteOfTheHour (see page 6665) function.

See Also

DecodeTime

MinuteOfTheDay (see page 6664)

MinuteOfTheHour (see page 6665)

MinuteOfTheYear (see page 6667)

MinuteOfTheMonth (see page 6666)

MinuteOfTheWeek (see page 6666)

DayOf (see page 6618)

HourOf (see page 6638)

MilliSecondOf (see page 6656)

MonthOf (see page 6670)

RecodeDateMinute

SecondOf (see page 6682)

WeekOf (see page 6703)

YearOf (see page 6714)

38.1.77 DateUtils.MinuteOfTheDay Function

Returns the number of minutes between a specified TDateTime value and 12:00 AM of the same day.

Pascal

```
function MinuteOfTheDay(const AValue: TDateTime): Word;
```

C++

```
Word MinuteOfTheDay(const TDateTime AValue);
```

File

DateUtils

Description

Call MinuteOfTheDay to obtain the number of minutes between AValue and 12:00 AM on the same day.

See Also

MinuteOfTheHour (see page 6665)

MinuteOfTheYear (see page 6667)

MinuteOfTheMonth (see page 6666)

MinuteOfTheWeek (see page 6666)

HourOfTheDay (see page 6639)

MilliSecondOfTheDay (see page 6657)

SecondOfTheDay (see page 6683)

MinutesBetween (see page 6668)

StartOfTheDay (see page 6692)

38.1.78 DateUtils.MinuteOfTheHour Function

Returns the number of minutes between a specified TDateTime value and the beginning of the same hour.

Pascal

```
function MinuteOfTheHour(const AValue: TDateTime): Word;
```

C++

```
Word MinuteOfTheHour(const TDateTime AValue);
```

File

DateUtils

Description

Call MinuteOfTheHour to obtain the number of minutes between AValue and the start of the same hour on the same day.

Note: MinuteOfTheHour returns the same value as the MinuteOf (see page 6663) function.

See Also

MinuteOfTheDay (see page 6664)

MinuteOfTheYear (see page 6667)

MinuteOfTheMonth (see page 6666)

MinuteOfTheWeek (see page 6666)

MinuteOfTheHour

SecondOfTheHour (see page 6684)

MinutesBetween (see page 6668)

38.1.79 DateUtils.MinuteOfTheMonth Function

Returns the number of minutes between a specified TDateTime value and 12:00 AM of the first day of the month.

Pascal

```
function MinuteOfTheMonth(const AValue: TDateTime): Word;
```

C++

```
Word MinuteOfTheMonth(const TDateTime AValue);
```

File

DateUtils

Description

Call MinuteOfTheMonth to obtain the number of minutes between AValue and 12:00 AM on the first day of the month that includes AValue.

See Also

MinuteOfTheDay (see page 6664)

MinuteOfTheWeek (see page 6666)

MinuteOfTheYear (see page 6667)

MinuteOfTheHour (see page 6665)

DayOfTheMonth (see page 6618)

HourOfTheMonth (see page 6640)

MilliSecondOfTheMonth (see page 6659)

SecondOfTheMonth (see page 6685)

WeekOfTheMonth (see page 6704)

StartOfTheMonth (see page 6692)

MinutesBetween (see page 6668)

38.1.80 DateUtils.MinuteOfTheWeek Function

Returns the number of minutes between a specified TDateTime value and 12:00 AM of the first day of the week.

Pascal

```
function MinuteOfTheWeek(const AValue: TDateTime): Word;
```

C++

```
Word MinuteOfTheWeek(const TDateTime AValue);
```

File

DateUtils

Description

Call MinuteOfTheWeek to obtain the number of minutes between AValue and 12:00 AM on Monday of the week that includes AValue.

Note: MinuteOfTheWeek defines the week of AValue according to the ISO 8601 standard. That is, the week starts on Monday and ends on Sunday.

See Also

MinuteOfTheDay (see page 6664)

MinuteOfTheMonth (see page 6666)

MinuteOfTheYear (see page 6667)

MinuteOfTheHour (see page 6665)

DayOfWeek

MilliSecondOfTheWeek (see page 6661)

SecondOfTheWeek (see page 6686)

HourOfTheWeek (see page 6641)

StartOfTheWeek (see page 6693)

MinutesBetween (see page 6668)

38.1.81 DateUtils.MinuteOfTheYear Function

Returns the number of minutes between a specified TDateTime value and 12:00 AM of the first day of the year.

Pascal

```
function MinuteOfTheYear(const AValue: TDateTime): LongWord;
```

C++

```
LongWord MinuteOfTheYear(const TDateTime AValue);
```

File

DateUtils

Description

Call MinuteOfTheYear to obtain the number of minutes between AValue and 12:00 AM on Jan 1 of the year that includes AValue.

See Also

MinuteOfTheHour (see page 6665)

MinuteOfTheDay (see page 6664)

MinuteOfTheWeek (see page 6666)

MinuteOfTheMonth (see page 6666)

DayOfTheYear (see page 6620)

HourOfTheYear (see page 6641)

MilliSecondOfTheYear (see page 6661)

MonthOfTheYear (see page 6670)

SecondOfTheYear (see page 6687)

WeekOfTheYear (see page 6705)

StartOfTheYear (see page 6693)

MinutesBetween (see page 6668)

38.1.82 DateUtils.MinutesBetween Function

Returns the number of minutes between two specified TDateTime values.

Pascal

```
function MinutesBetween(const ANow: TDateTime; const AThen: TDateTime): Int64;
```

C++

```
Int64 MinutesBetween(const TDateTime ANow, const TDateTime AThen);
```

File

DateUtils

Description

Call MinutesBetween to obtain the difference, in minutes, between two TDateTime values. MinutesBetween counts only entire minutes. Thus, MinutesBetween reports the difference between 9:00:00 AM and 9:00:59:999 AM as 0 because the difference is one millisecond short of an entire minute.

See Also

MinuteSpan (see page 6668)

DaysBetween (see page 6621)

HoursBetween (see page 6642)

MilliSecondsBetween (see page 6662)

MonthsBetween (see page 6671)

SecondsBetween (see page 6687)

WeeksBetween (see page 6706)

YearsBetween (see page 6714)

WithinPastMinutes (see page 6710)

38.1.83 DateUtils.MinuteSpan Function

Returns the number of minutes, including fractions thereof, between two specified TDateTime values.

Pascal

```
function MinuteSpan(const ANow: TDateTime; const AThen: TDateTime): Double;
```

C++

```
Double MinuteSpan(const TDateTime ANow, const TDateTime AThen);
```

File

DateUtils

Description

Call MinuteSpan to obtain the difference, in minutes, between two TDateTime values. Unlike the MinutesBetween (see page 6668) function, which only counts entire minutes, MinuteSpan reports incomplete minutes as a fraction of an entire minute.

See Also

MinutesBetween (see page 6668)

DaySpan (see page 6624)

HourSpan (see page 6643)

MilliSecondSpan (see page 6663)

MonthSpan (see page 6672)

SecondSpan (see page 6688)

WeekSpan (see page 6708)

YearSpan (see page 6715)

38.1.84 DateUtils.ModifiedJulianDateToDateTime Function

Converts a modified Julian date to a TDateTime value.

Pascal

```
function ModifiedJulianDateToDateTime(const AValue: Double): TDateTime;
```

C++

```
TDateTime ModifiedJulianDateToDateTime(const Double AValue);
```

File

DateUtils

Description

Call ModifiedJulianDateToDateTime to convert a modified Julian date to the corresponding TDateTime value.

The modified Julian date is the number of days, including fractional days, since Greenwich midnight on November 17, 1858. Modified Julian dates are based on Julian dates, but adjusted to use midnight rather than noon as a starting point and to use a more recent date as a starting point.

Note: If ModifiedJulianDateToDateTime can't convert the date specified by AValue, it raises an EConvertError exception.

See Also

DateTimeToModifiedJulianDate (see page 6617)

DateTimeToJulianDate (see page 6616)

JulianDateToDateTime (see page 6656)

Convert

TryJulianDateToDateTime (see page 6700)

TryModifiedJulianDateToDateTime (see page 6701)

EConvertError

38.1.85 DateUtils.MonthOf Function

Returns the month of the year represented by a TDateTime value.

Pascal

```
function MonthOf(const AValue: TDateTime): Word;
```

C++

```
Word MonthOf(const TDateTime AValue);
```

File

DateUtils

Description

Call MonthOf to obtain the month of the year represented by a specified TDateTime value. MonthOf returns a value between 1 and 12.

Note: MonthOf returns the same value as the MonthOfTheYear (see page 6670) function.

See Also

DecodeDate

MonthOfTheYear (see page 6670)

DayOf (see page 6618)

MilliSecondOf (see page 6656)

MinuteOf (see page 6663)

HourOf (see page 6638)

RecodeDateMonth

SecondOf (see page 6682)

WeekOf (see page 6703)

YearOf (see page 6714)

38.1.86 DateUtils.MonthOfTheYear Function

Returns the month of the year represented by a TDateTime value.

Pascal

```
function MonthOfTheYear(const AValue: TDateTime): Word;
```

C++

```
Word MonthOfTheYear(const TDateTime AValue);
```

File

DateUtils

Description

Call MonthOfTheYear to obtain the month of the year represented by a specified TDateTime value. MonthOfTheYear returns a value between 1 and 12.

Note: MonthOfTheYear returns the same value as the MonthOf (see page 6670) function.

See Also

MonthOf (see page 6670)

DayOfTheYear (see page 6620)

MilliSecondOfTheYear (see page 6661)

MinuteOfTheYear (see page 6667)

HourOfTheYear (see page 6641)

SecondOfTheYear (see page 6687)

WeekOfTheYear (see page 6705)

38.1.87 DateUtils.MonthsBetween Function

Returns the approximate number of months between two specified TDateTime values.

Pascal

```
function MonthsBetween(const ANow: TDateTime; const AThen: TDateTime): Integer;
```

C++

```
int MonthsBetween(const TDateTime ANow, const TDateTime AThen);
```

File

DateUtils

Description

Call MonthsBetween to obtain the difference, in months, between two TDateTime values. Because months are not all the same length, MonthsBetween returns an approximation based on an assumption of 30.4375 days per month. Fractional months are not counted. Thus, for example, MonthsBetween reports the difference between Feb 1 and Mar 1 as 0.

See Also

MonthSpan (see page 6672)

DaysBetween (see page 6621)

MillisecondsBetween

MinutesBetween (see page 6668)

HoursBetween (see page 6642)

SecondsBetween (see page 6687)

WeeksBetween (see page 6706)

YearsBetween (see page 6714)

WithinPastMonths (see page 6711)

38.1.88 DateUtils.MonthSpan Function

Returns the approximate number of months, including fractions, between two specified TDateTime values.

Pascal

```
function MonthSpan(const ANow: TDateTime; const AThen: TDateTime): Double;
```

C++

```
Double MonthSpan(const TDateTime ANow, const TDateTime AThen);
```

File

DateUtils

Description

Call MonthSpan to obtain the difference, in months, between two TDateTime values. Because months are not all the same length, MonthsBetween (see page 6671) returns an approximation based on an assumption of 30.4375 days per month. Unlike the MonthsBetween (see page 6671) function, which does not report fractional "months", MonthSpan reports incomplete months as a fraction of an entire month.

See Also

MonthsBetween (see page 6671)

DaySpan (see page 6624)

MillisecondSpan

MinuteSpan (see page 6668)

HourSpan (see page 6643)

SecondSpan (see page 6688)

WeekSpan (see page 6708)

YearSpan (see page 6715)

38.1.89 DateUtils.NthDayOfWeek Function

Returns which occurrence of its weekday a specified TDateTime value represents.

Pascal

```
function NthDayOfWeek(const AValue: TDateTime): Word;
```

C++

```
Word NthDayOfWeek(const TDateTime AValue);
```

File

DateUtils

Description

NthDayOfWeek indicates which occurrence of a weekday within a month the date and time specified by AValue represents. For example, if AValue represents the 2nd Tuesday of the month, NthDayOfWeek returns 2.

Note that this value may differ from the value that the WeekOfTheMonth (see page 6704) function returns, because NthDayOfWeek counts every occurrence of the given weekday, while WeekOfTheMonth (see page 6704) only counts a week if it includes 4 or more days in the month. Thus, for example, if AValue represents a Saturday that is the first day of a month, NthDayOfWeek returns 1, while WeekOfTheMonth (see page 6704) returns 5 (or maybe 4), indicating the last week of the previous month.

Note: To obtain the year, month, and day of the week, as well as which occurrence of that day of the week the specified TDateTime represents, use the DecodeDayOfWeekInMonth (see page 6628) function.

See Also

WeekOfTheMonth (see page 6704)

DecodeDayOfWeekInMonth (see page 6628)

38.1.90 DateUtils.RecodeDate Function

Replaces the date portion of a specified TDateTime value.

Pascal

```
function RecodeDate(const AValue: TDateTime; const AYear: Word; const AMonth: Word; const
ADay: Word): TDateTime;
```

C++

```
TDateTime RecodeDate(const TDateTime AValue, const Word AYear, const Word AMonth, const Word
ADay);
```

File

DateUtils

Description

Call RecodeDate to convert the date/time value specified by AValue with the value obtained by changing the year to AYear, the month to AMonth, and the day of the month to ADay.

AYear must fall between 1 and 9999 (inclusive).

AMonth can range from 1 to 12.

Legal values for ADay depend on the month represented by AMonth. (For example, if AMonth is 1, ADay can range from 1 to 31; if AMonth is 2, ADay can range from 1 to 28 or 1 to 29, depending on the year; and so on.)

If the values provided for AYear, AMonth, and ADay do not represent a valid date, RecodeDate raises an EConvertError exception.

See Also

ReplaceDate

RecodeDateMilliSecond

RecodeDateSecond

RecodeDateMinute

RecodeDateHour

RecodeDateMonth

RecodeDateYear

DayOf (see page 6618)

EConvertError

RecodeTime (see page 6679)

38.1.91 DateUtils.RecodeDateTime Function

Selectively replaces parts of a specified TDateTime value.

Pascal

```
function RecodeDateTime(const AValue: TDateTime; const AYear: Word; const AMonth: Word; const
ADay: Word; const AHour: Word; const AMinute: Word; const ASecond: Word; const AMilliSecond:
Word): TDateTime;
```

C++

```
TDateTime RecodeDateTime(const TDateTime AValue, const Word AYear, const Word AMonth, const
Word ADay, const Word AHour, const Word AMinute, const Word ASecond, const Word AMilliSecond);
```

File

DateUtils

Description

Call RecodeDateTime to convert the date/time value specified by AValue with the value obtained by changing the year to AYear, the month to AMonth, the day of the month to ADay, the hour to AHour, the minute to AMinute, the second to ASecond, and the millisecond to AMilliSecond. Any one of these parameters can be set to RecodeLeaveFieldAsIs (see page 6723) which tells RecodeDateTime not change the existing portion of the TDateTime value.

AYear can range from 1 to 9999 (or be RecodeLeaveFieldAsIs (see page 6723)).

AMonth can range from 1 to 12 (or be RecodeLeaveFieldAsIs (see page 6723)).

Legal values for ADay depend on the month represented by AMonth. (For example, if AMonth is 1, ADay can range from 1 to 31; if AMonth is 2, ADay can range from 1 to 28 or 1 to 29, depending on the year; and so on.) ADay can also be RecodeLeaveFieldAsIs (see page 6723).

AHour can range from 0 to 24 (or be RecodeLeaveFieldAsIs (see page 6723)). If AHour is 24, AMinute, ASecond, and AMilliSecond must all be 0.

AMinute can range from 0 to 59 (or be RecodeLeaveFieldAsIs (see page 6723)).

ASecond can range from 0 to 59 (or be RecodeLeaveFieldAsIs (see page 6723)).

AMilliSecond can range from 0 to 999 (or be RecodeLeaveFieldAsIs (see page 6723)).

If the values do not result in a valid date and time, RecodeDateTime raises an EConvertError exception.

See Also

ReplaceDate

RecodeTime (see page 6679)

RecodeDate (see page 6673)

EConvertError

ReplaceTime

EncodeDateTime (see page 6630)

TryRecodeDateTime (see page 6702)

38.1.92 DateUtils.RecodeDay Function

Replaces the day of the month for a specified TDateTime value.

Pascal

```
function RecodeDay(const AValue: TDateTime; const ADay: Word): TDateTime;
```

C++

```
TDateTime RecodeDay(const TDateTime AValue, const Word ADay);
```

File

DateUtils

Description

Call RecodeDay to convert the date/time value specified by AValue with the value obtained by changing the day of the month to ADay. Legal values for ADay depend on the month represented by AValue. (For example, if AValue is in January, ADay can range from 1 to 31; if AValue is in February, ADay can range from 1 to 28 or 1 to 29, depending on the year; and so on.)

If the value provided for ADay does not represent a valid day in the month identified by AValue, RecodeDay raises an EConvertError exception.

See Also

ReplaceDate

RecodeDateMilliSecond

RecodeDateSecond

RecodeDateMinute

RecodeDateHour

RecodeDateMonth

RecodeDateYear

DayOf (see page 6618)

EConvertError

38.1.93 DateUtils.RecodeHour Function

Replaces the hour of the day for a specified TDateTime value.

Pascal

```
function RecodeHour(const AValue: TDateTime; const AHour: Word): TDateTime;
```

C++

```
TDateTime RecodeHour(const TDateTime AValue, const Word AHour);
```

File

DateUtils

Description

Call RecodeHour to convert the date/time value specified by AValue with the value obtained by changing the hour of the day to AHour. Legal values for AHour range from 0 to 24. If AHour is 24, the minutes, seconds, and milliseconds of AValue must all be 0.

If the value provided for AHour is not in the legal range, RecodeHour raises an EConvertError exception.

See Also

ReplaceTime

RecodeDateMilliSecond

RecodeDateSecond

RecodeDateMinute

RecodeDateDay

RecodeDateMonth

RecodeDateYear

HourOf (see page 6638)

EConvertError

38.1.94 DateUtils.RecodeMilliSecond Function

Replaces the millisecond of the second for a specified TDateTime value.

Pascal

```
function RecodeMilliSecond(const AValue: TDateTime; const AMilliSecond: Word): TDateTime;
```

C++

```
TDateTime RecodeMilliSecond(const TDateTime AValue, const Word AMilliSecond);
```

File

DateUtils

Description

Call RecodeDateMilliSecond to convert the date/time value specified by AValue with the value obtained by changing the milliseconds of the second to AMilliSecond. Legal values for AMilliSecond range from 0 to 999.

If the value provided for AMilliSecond is not in the legal range, RecodeDateMilliSecond raises an EConvertError exception.

See Also

ReplaceTime

RecodeDateHour

RecodeDateSecond

RecodeDateMinute

RecodeDateDay

RecodeDateMonth

RecodeDateYear

MilliSecondOf (see page 6656)

EConvertError

38.1.95 DateUtils.RecodeMinute Function

Replaces the minute of the hour for a specified TDateTime value.

Pascal

```
function RecodeMinute(const AValue: TDateTime; const AMinute: Word): TDateTime;
```

C++

```
TDateTime RecodeMinute(const TDateTime AValue, const Word AMinute);
```

File

DateUtils

Description

Call RecodeMinute to convert the date/time value specified by AValue with the value obtained by changing the minute of the hour to AMinute. Legal values for AMinute range from 0 to 59.

If the value provided for AMinute is not in the legal range, RecodeMinute raises an EConvertError exception.

See Also

ReplaceTime

RecodeDateMilliSecond

RecodeDateSecond

RecodeDateHour

RecodeDateDay

RecodeDateMonth

RecodeDateYear

MinuteOf (see page 6663)

EConvertError

38.1.96 DateUtils.RecodeMonth Function

Replaces the month of the year for a specified TDateTime value.

Pascal

```
function RecodeMonth(const AValue: TDateTime; const AMonth: Word): TDateTime;
```

C++

```
TDateTime RecodeMonth(const TDateTime AValue, const Word AMonth);
```

File

DateUtils

Description

Call RecodeMonth to convert the date/time value specified by AValue with the value obtained by changing the month of the year to AMonth. Legal values for AMonth range from 1 to 12.

If the value provided for AMonth does not represent a valid month, or if the change results in an invalid day of the month (for example trying to change Jan 31 to Feb 31), RecodeMonth raises an EConvertError exception.

See Also

ReplaceDate

RecodeDateMilliSecond

RecodeDateSecond

RecodeDateMinute

RecodeDateHour

RecodeDateDay

RecodeDateYear

MonthOf (see page 6670)

EConvertError

38.1.97 DateUtils.RecodeSecond Function

Replaces the second of the minute for a specified TDateTime value.

Pascal

```
function RecodeSecond(const AValue: TDateTime; const ASecond: Word): TDateTime;
```

C++

```
TDateTime RecodeSecond(const TDateTime AValue, const Word ASecond);
```

File

DateUtils

Description

Call RecodeSecond to convert the date/time value specified by AValue with the value obtained by changing the second of the minute to ASecond. Legal values for ASecond range from 0 to 59.

If the value provided for ASecond is not in the legal range, RecodeSecond raises an EConvertError exception.

See Also

ReplaceTime

RecodeDateMilliSecond

RecodeDateMinute

RecodeDateHour

RecodeDateDay

RecodeDateMonth

RecodeDateYear

SecondOf (see page 6682)

EConvertError

38.1.98 DateUtils.RecodeTime Function

Replaces the time portion of a specified TDateTime value.

Pascal

```
function RecodeTime(const AValue: TDateTime; const AHour: Word; const AMinute: Word; const
ASecond: Word; const AMilliSecond: Word): TDateTime;
```

C++

```
TDateTime RecodeTime(const TDateTime AValue, const Word AHour, const Word AMinute, const Word
ASecond, const Word AMilliSecond);
```

File

DateUtils

Description

Call RecodeTime to convert the date/time value specified by AValue with the value obtained by changing the hour to AHour, the minute to AMinute, the second to ASecond, and the milliseconds to AMilliSecond.

AHour can range from 0 to 24. If AHour is 24, AMinute, ASecond, and AMilliSecond must all be 0.

AMinute can range from 0 to 59.

ASecond can range from 0 to 59.

AMilliSecond can range from 0 to 999.

If the values provided for AHour, AMinute, ASecond, and AMilliSecond do not fall in the valid range, RecodeTime raises an EConvertError exception.

See Also

ReplaceTime

RecodeDateMilliSecond

RecodeDateSecond

RecodeDateMinute

RecodeDateHour

RecodeDateMonth

RecodeDateYear

EConvertError

RecodeDate (see page 6673)

38.1.99 DateUtils.RecodeYear Function

Replaces the year for a specified TDateTime value.

Pascal

```
function RecodeYear(const AValue: TDateTime; const AYear: Word): TDateTime;
```

C++

```
TDateTime RecodeYear(const TDateTime AValue, const Word AYear);
```

File

DateUtils

Description

Call RecodeYear to convert the date/time value specified by AValue with the value obtained by changing the year to AYear. Legal values for AYear range from 1 to 9999.

If the value provided for AYear is not in the legal range, or if the change results in an invalid day of the month (changing Feb 29 on a leap year to a non-leap year), RecodeYear raises an EConvertError exception.

See Also

ReplaceDate

RecodeDateMilliSecond

RecodeDateSecond

RecodeDateMinute

RecodeDateHour

RecodeDateDay

RecodeDateYear

MonthOf (see page 6670)

EConvertError

38.1.100 DateUtils.SameDate Function

Indicates whether two TDateTime values represent the same year, month, and day.

Pascal

```
function SameDate(const A: TDateTime; const B: TDateTime): Boolean;
```

C++

```
Boolean SameDate(const TDateTime A, const TDateTime B);
```

File

DateUtils

Description

Call SameDate to determine whether the two TDateTime values specified by A and B represent the same date, ignoring the time portion. SameDate returns true if the two specified values occur on the same day, false otherwise.

See Also

CompareDate (see page 6614)

ConvUnitSameValue

SameValue

SameDateTime (see page 6681)

SameTime (see page 6682)

IsSameDay (see page 6651)

38.1.101 DateUtils.SameDateTime Function

Indicates whether two TDateTime values represent the same year, month, day, hour, minute, second, and millisecond.

Pascal

```
function SameDateTime(const A: TDateTime; const B: TDateTime): Boolean;
```

C++

```
Boolean SameDateTime(const TDateTime A, const TDateTime B);
```

File

DateUtils

Description

Call SameDateTime to determine whether the two TDateTime values specified by A and B represent the same time, down to the millisecond. SameDateTime returns true if the two specified values represent the same time, false otherwise.

Note: It is possible to create two TDateTime values that differ numerically when viewed as floating point types, but which represent the same year, month, day, hour, minute, second, and millisecond. SameDateTime allows applications to determine when two TDateTime values are the same even when their numeric values differ.

See Also

CompareDateTime (see page 6614)

ConvUnitSameValue

SameValue

SameDate (see page 6681)

SameTime (see page 6682)

IsSameDay (see page 6651)

38.1.102 DateUtils.SameTime Function

Indicates whether two TDateTime values represent the same time of day, ignoring the date portion.

Pascal

```
function SameTime(const A: TDateTime; const B: TDateTime): Boolean;
```

C++

```
Boolean SameTime(const TDateTime A, const TDateTime B);
```

File

DateUtils

Description

Call SameTime to determine whether the two TDateTime values specified by A and B represent the same time of day, ignoring the date portion. SameTime returns true if the two specified values occur on the same hour, minute, second, and millisecond, false otherwise.

Note: It is possible to create two TDateTime values whose time portions differ numerically when viewed as floating point types, but which represent the same hour, minute, second, and millisecond. SameTime allows applications to determine when two TDateTime values represent the same time, even when the fractional parts of the values differ.

See Also

CompareTime (see page 6615)

ConvUnitSameValue

SameValue

SameDateTime (see page 6681)

Date

38.1.103 DateUtils.SecondOf Function

Returns the second of the minute represented by a TDateTime value.

Pascal

```
function SecondOf(const AValue: TDateTime): Word;
```

C++

```
Word SecondOf(const TDateTime AValue);
```

File

DateUtils

Description

Call SecondOf to obtain the second of the minute represented by a specified TDateTime value. SecondOf returns a value between 0 and 59.

Note: SecondOf returns the same value as the SecondOfTheMinute (see page 6685) function.

See Also

DecodeTime

SecondOfTheDay (see page 6683)

SecondOfTheHour (see page 6684)

SecondOfTheMinute (see page 6685)

SecondOfTheYear (see page 6687)

SecondOfTheMonth (see page 6685)

SecondOfTheWeek (see page 6686)

DayOf (see page 6618)

HourOf (see page 6638)

MinuteOf (see page 6663)

MonthOf (see page 6670)

RecodeDateSecond

MilliSecondOf (see page 6656)

WeekOf (see page 6703)

YearOf (see page 6714)

38.1.104 DateUtils.SecondOfTheDay Function

Returns the number of seconds between a specified TDateTime value and 12:00:00 AM of the same day.

Pascal

```
function SecondOfTheDay(const AValue: TDateTime): LongWord;
```

C++

```
LongWord SecondOfTheDay(const TDateTime AValue);
```

File

DateUtils

Description

Call SecondOfTheDay to obtain the number of seconds between AValue and 12:00:00 AM on the same day.

See Also

SecondOfTheHour (see page 6684)

SecondOfTheMinute (see page 6685)

SecondOfTheYear (see page 6687)

SecondOfTheMonth (see page 6685)

SecondOfTheWeek (see page 6686)

HourOfTheDay (see page 6639)

MinuteOfTheDay (see page 6664)

MilliSecondOfTheDay (see page 6657)

SecondsBetween (see page 6687)

StartOfTheDay (see page 6692)

38.1.105 DateUtils.SecondOfTheHour Function

Returns the number of seconds between a specified TDateTime value and the beginning of the same hour.

Pascal

```
function SecondOfTheHour(const AValue: TDateTime): Word;
```

C++

```
Word SecondOfTheHour(const TDateTime AValue);
```

File

DateUtils

Description

Call SecondOfTheHour to obtain the number of seconds between AValue and the start of the same hour on the same day.

See Also

SecondOfTheDay (see page 6683)

SecondOfTheMinute (see page 6685)

SecondOfTheYear (see page 6687)

SecondOfTheMonth (see page 6685)

SecondOfTheWeek (see page 6686)

MinuteOfTheHour (see page 6665)

MilliSecondOfTheHour (see page 6658)

SecondsBetween (see page 6687)

38.1.106 DateUtils.SecondOfTheMinute Function

Returns the number of seconds between a specified TDateTime value and the beginning of the same minute.

Pascal

```
function SecondOfTheMinute(const AValue: TDateTime): Word;
```

C++

```
Word SecondOfTheMinute(const TDateTime AValue);
```

File

DateUtils

Description

Call SecondOfTheMinute to obtain the number of seconds between AValue and the start of the same minute on the same day.

Note: SecondOfTheMinute returns the same value as the SecondOf (see page 6682) function.

See Also

SecondOfTheDay (see page 6683)

SecondOfTheHour (see page 6684)

SecondOfTheYear (see page 6687)

SecondOfTheMonth (see page 6685)

SecondOfTheWeek (see page 6686)

MilliSecondOfTheMinute (see page 6659)

SecondsBetween (see page 6687)

38.1.107 DateUtils.SecondOfTheMonth Function

Returns the number of seconds between a specified TDateTime value and 12:00:00 AM on the first day of the month.

Pascal

```
function SecondOfTheMonth(const AValue: TDateTime): LongWord;
```

C++

```
LongWord SecondOfTheMonth(const TDateTime AValue);
```

File

DateUtils

Description

Call SecondOfTheMonth to obtain the number of seconds between AValue and 12:00:00 AM on the first day of the month that includes AValue.

See Also

SecondOfTheDay (see page 6683)

SecondOfTheWeek (see page 6686)

SecondOfTheYear (see page 6687)

SecondOfTheMinute (see page 6685)

SecondOfTheHour (see page 6684)

DayOfTheMonth (see page 6618)

HourOfTheMonth (see page 6640)

MinuteOfTheMonth (see page 6666)

MilliSecondOfTheMonth (see page 6659)

WeekOfTheMonth (see page 6704)

StartOfTheMonth (see page 6692)

SecondsBetween (see page 6687)

38.1.108 DateUtils.SecondOfTheWeek Function

Returns the number of seconds between a specified TDateTime value and 12:00:00 AM of the first day of the week.

Pascal

```
function SecondOfTheWeek(const AValue: TDateTime): LongWord;
```

C++

```
LongWord SecondOfTheWeek(const TDateTime AValue);
```

File

DateUtils

Description

Call SecondOfTheWeek to obtain the number of seconds between AValue and 12:00:00 AM on Monday of the week that includes AValue.

Note: SecondOfTheWeek defines the week of AValue according to the ISO 8601 standard. That is, the week starts on Monday and ends on Sunday.

See Also

SecondOfTheDay (see page 6683)

SecondOfTheMonth (see page 6685)

SecondOfTheYear (see page 6687)

SecondOfTheHour (see page 6684)

SecondOfTheMinute (see page 6685)

DayOfWeek

HourOfTheWeek (see page 6641)

MinuteOfTheWeek (see page 6666)

MilliSecondOfTheWeek (see page 6661)

StartOfTheWeek (see page 6693)

SecondsBetween (see page 6687)

38.1.109 DateUtils.SecondOfTheYear Function

Returns the number of seconds between a specified TDateTime value and 12:00:00 AM on the first day of the year.

Pascal

```
function SecondOfTheYear(const AValue: TDateTime): LongWord;
```

C++

```
LongWord SecondOfTheYear(const TDateTime AValue);
```

File

DateUtils

Description

Call SecondOfTheYear to obtain the number of seconds between AValue and 12:00:00 AM on Jan 1 of the year that includes AValue.

See Also

SecondOfTheMinute (see page 6685)

SecondOfTheHour (see page 6684)

SecondOfTheDay (see page 6683)

SecondOfTheWeek (see page 6686)

SecondOfTheMonth (see page 6685)

DayOfTheYear (see page 6620)

HourOfTheYear (see page 6641)

MinuteOfTheYear (see page 6667)

MonthOfTheYear (see page 6670)

MilliSecondOfTheYear (see page 6661)

WeekOfTheYear (see page 6705)

StartOfTheYear (see page 6693)

SecondsBetween (see page 6687)

38.1.110 DateUtils.SecondsBetween Function

Returns the number of seconds between two specified TDateTime values.

Pascal

```
function SecondsBetween(const ANow: TDateTime; const AThen: TDateTime): Int64;
```

C++

```
Int64 SecondsBetween(const TDateTime ANow, const TDateTime AThen);
```

File

DateUtils

Description

Call SecondsBetween to obtain the difference, in seconds, between two TDateTime values. SecondsBetween counts only entire seconds. Thus, SecondsBetween reports the difference between 9:00:00 AM and 9:00:00:999 AM as 0 because the difference is one millisecond short of an entire second.

See Also

SecondSpan (see page 6688)

DaysBetween (see page 6621)

MillisecondsBetween

MinutesBetween (see page 6668)

HoursBetween (see page 6642)

MonthsBetween (see page 6671)

WeeksBetween (see page 6706)

YearsBetween (see page 6714)

WithinPastSeconds (see page 6712)

38.1.111 DateUtils.SecondSpan Function

Returns the number of seconds (including fractions thereof) between two specified TDateTime values.

Pascal

```
function SecondSpan(const ANow: TDateTime; const AThen: TDateTime): Double;
```

C++

```
Double SecondSpan(const TDateTime ANow, const TDateTime AThen);
```

File

DateUtils

Description

Call SecondSpan to obtain the difference, in seconds, between two TDateTime values. Unlike the SecondsBetween (see page 6687) function, which only counts entire seconds, SecondSpan reports incomplete seconds as a fraction of an entire second.

See Also

SecondsBetween (see page 6687)

DaySpan (see page 6624)

MillisecondSpan

MinuteSpan (see page 6668)

HourSpan (see page 6643)

MonthSpan (see page 6672)

WeekSpan (see page 6708)

YearSpan (see page 6715)

38.1.112 DateUtils.StartOfADay Function

Returns a TDateTime that represents 12:00:00:00 AM on a specified day.

Pascal

```
function StartOfADay(const AYear: Word; const AMonth: Word; const ADay: Word): TDateTime;
overload;
function StartOfADay(const AYear: Word; const ADayOfYear: Word): TDateTime; overload;
```

C++

```
TDateTime StartOfADay(const Word AYear, const Word AMonth, const Word ADay);
TDateTime StartOfADay(const Word AYear, const Word ADayOfYear);
```

File

DateUtils

Description

StartOfADay returns the first expressible moment (12:00:000 AM) of a specified day.

The AYear parameter specifies the year of the desired day.

The ADayOfYear parameter specifies the desired day as a day of the year, where 1 is January 1, 2 is January 2, 32 is February 1, and so on.

The AMonth and ADay parameters specify the desired day as a month of the year and day of the month. AMonth can range from 1 to 12. ADay can range from 1 to 28, 29, 30, or 31, depending on the values of AYear and AMonth.

If the parameters do not specify a valid date, StartOfADay raises throws an EConvertError exception.

See Also

StartOfTheDay (see page 6692)

StartOfAMonth (see page 6690)

StartOfAYear (see page 6691)

StartOfAWeek (see page 6690)

ReplaceTime

EndOfADay (see page 6633)

EncodeDateDay (see page 6629)

EncodeDate

38.1.113 DateUtils.StartOfAMonth Function

Returns a TDateTime that represents 12:00:00:00 AM on the first day of a specified month.

Pascal

```
function StartOfAMonth(const AYear: Word; const AMonth: Word): TDateTime;
```

C++

```
TDateTime StartOfAMonth(const Word AYear, const Word AMonth);
```

File

DateUtils

Description

StartOfAMonth returns the first expressible moment (12:00:000 AM) of the first day of a specified month.

The AYear parameter specifies the year of the desired month.

The AMonth parameter specifies the month. It can range from 1 to 12.

If the parameters do not specify a valid month, StartOfAMonth raises an EConvertError exception.

See Also

StartOfTheMonth (see page 6692)

StartOfADay (see page 6689)

StartOfAYear (see page 6691)

StartOfAWeek (see page 6690)

ReplaceTime

EndOfAMonth (see page 6634)

EncodeDate

38.1.114 DateUtils.StartOfAWeek Function

Returns a TDateTime that represents 1the first moment on a specified day of a specified week.

Pascal

```
function StartOfAWeek(const AYear: Word; const AWeekOfYear: Word; const ADayOfWeek: Word = 1):
TDateTime;
```

C++

```
TDateTime StartOfAWeek(const Word AYear, const Word AWeekOfYear, const Word ADayOfWeek = 1);
```

File

DateUtils

Description

StartOfAWeek returns the first expressible moment (12:00:00.000 AM) of the specified day of the specified week.

The AYear parameter specifies the year of the desired day.

The AWeekOfYear parameter specifies the week of the year, where 1 is the first week in AYear that includes four or more days.

The ADayOfWeek parameter indicates the desired day in the specified week, where 1 is Monday, 2 is Tuesday, and so on.

If the parameters do not specify a valid date, StartOfAWeek raises an EConvertError exception.

Note: The definitions for AWeekOfYear and ADayOfWeek follow the ISO 8601 standard.

See Also

StartOfTheWeek (see page 6693)

StartOfADay (see page 6689)

StartOfAYear (see page 6691)

StartOfAMonth (see page 6690)

ReplaceTime

EndOfAWeek (see page 6635)

EncodeDateWeek (see page 6631)

38.1.115 DateUtils.StartOfAYear Function

Returns a TDateTime that represents the first moment on the first day of a specified year.

Pascal

```
function StartOfAYear(const AYear: Word): TDateTime;
```

C++

```
TDateTime StartOfAYear(const Word AYear);
```

File

DateUtils

Description

StartOfAYear returns the first expressible moment (January 1 at 12:00:00.000 AM) of the year specified by the AYear parameter.

See Also

StartOfTheYear (see page 6693)

StartOfADay (see page 6689)

StartOfAWeek (see page 6690)

StartOfAMonth (see page 6690)

ReplaceTime

EndOfAYear (see page 6636)

38.1.116 DateUtils.StartOfTheDay Function

Returns a TDateTime that represents 12:00:00:00 AM on the day identified by a specified TDateTime.

Pascal

```
function StartOfTheDay(const AValue: TDateTime): TDateTime;
```

C++

```
TDateTime StartOfTheDay(const TDateTime AValue);
```

File

DateUtils

Description

StartOfTheDay returns the first expressible moment of the same day as the TDateTime specified by AValue. That is, it replaces the time portion of AValue with 0 and returns the result.

See Also

EndOfTheDay (see page 6636)

StartOfTheMonth (see page 6692)

StartOfTheYear (see page 6693)

StartOfTheWeek (see page 6693)

ReplaceTime

StartOfADay (see page 6689)

38.1.117 DateUtils.StartOfTheMonth Function

Returns a TDateTime that represents 12:00:00:00 AM on the first day of the month identified by a specified TDateTime.

Pascal

```
function StartOfTheMonth(const AValue: TDateTime): TDateTime;
```

C++

```
TDateTime StartOfTheMonth(const TDateTime AValue);
```

File

DateUtils

Description

StartOfTheMonth returns the first expressible moment of the same month as the TDateTime specified by AValue. That is, it replaces the time portion of AValue with 0, changes the day to 1, and returns the result.

See Also

EndOfTheMonth (see page 6637)

StartOfTheDay (see page 6692)

StartOfTheYear (see page 6693)

StartOfTheWeek (see page 6693)

StartOfAMonth (see page 6690)

38.1.118 DateUtils.StartOfTheWeek Function

Returns a TDateTime that represents 12:00:00:00 AM on the first day of the week identified by a specified TDateTime.

Pascal

```
function StartOfTheWeek(const AValue: TDateTime): TDateTime;
```

C++

```
TDateTime StartOfTheWeek(const TDateTime AValue);
```

File

DateUtils

Description

StartOfTheWeek returns the first expressible moment of the same week as the TDateTime specified by AValue. That is, it replaces the time portion of AValue with 0, changes the day to Monday, and returns the result.

Note: StartOfTheWeek defines the week of AValue according to the ISO 8601 standard. That is, the week starts on Monday and ends on Sunday.

See Also

EndOfTheMonth (see page 6637)

StartOfTheDay (see page 6692)

StartOfTheYear (see page 6693)

StartOfTheWeek

StartOfAWeek (see page 6690)

38.1.119 DateUtils.StartOfTheYear Function

Returns a TDateTime that represents 12:00:00:00 AM on the first day of the year identified by a specified TDateTime.

Pascal

```
function StartOfTheYear(const AValue: TDateTime): TDateTime;
```

C++

```
TDateTime StartOfTheYear(const TDateTime AValue);
```

File

DateUtils

Description

StartOfTheYear returns the first expressible moment of the same year as the TDateTime specified by AValue. That is, it replaces the time portion of AValue with 0, changes the day to January 1, and returns the result.

See Also

EndOfTheYear (see page 6638)

StartOfTheDay (see page 6692)

StartOfTheMonth (see page 6692)

StartOfTheWeek (see page 6693)

StartOfAYear (see page 6691)

38.1.120 DateUtils.TimeOf Function

Strips the date portion from a TDateTime value.

Pascal

```
function TimeOf(const AValue: TDateTime): TDateTime;
```

C++

```
TDateTime TimeOf(const TDateTime AValue);
```

File

DateUtils

Description

Call TimeOf to convert a TDateTime value to a TDateTime value that includes only the time information (sets the date portion to 0, which means 12/30/1899).

See Also

DecodeTime

ReplaceDate

DateOf (see page 6616)

38.1.121 DateUtils.Today Function

Returns a TDateTime value that represents the current date.

Pascal

```
function Today: TDateTime;
```

C++

```
TDateTime Today();
```

File

DateUtils

Description

Today returns a TDateTime value with the date portion set to the current date and the time portion set to 0.

See Also

IsToday (see page 6651)

Today

Date

Yesterday (see page 6716)

Tomorrow (see page 6695)

38.1.122 DateUtils.Tomorrow Function

Returns a TDateTime value that represents the following day.

Pascal

```
function Tomorrow: TDateTime;
```

C++

```
TDateTime Tomorrow();
```

File

DateUtils

Description

Tomorrow returns a TDateTime value with the date portion set to the day following the current date and the time portion set to 0.

See Also

Today (see page 6694)

Date

Yesterday (see page 6716)

38.1.123 DateUtils.TryEncodeDateDay Function

Calculates the TDateTime value that represents a specified day of the year for a specified year.

Pascal

```
function TryEncodeDateDay(const AYear: Word; const ADayOfYear: Word; out AValue: TDateTime):
Boolean;
```

C++

```
Boolean TryEncodeDateDay(const Word AYear, const Word ADayOfYear, TDateTime AValue);
```

File

DateUtils

Description

TryEncodeDateDay calculates the TDateTime value for the date specified by the AYear and ADayOfYear parameters.

AYear is the year. It can be any value between 1 and 9999 (inclusive).

ADayOfYear is the day of the year, where 1 is January , 2 is January 2, 32 is February 1, and so on.

AValue returns the specified date as a TDateTime value. The time portion of AValue is 0 (Midnight at the start of the specified day).

TryEncodeDateDay returns T true if AYear and ADayOfYear represent a valid date, and F false if ADayOfYear is less than 1 or greater than the number of days in AYear.

See Also

TryEncodeDateTime (see page 6697)

YearOf (see page 6714)

TryEncodeTime

DayOfTheYear (see page 6620)

TryEncodeDateWeek (see page 6698)

DecodeDateDay (see page 6624)

EncodeDateDay (see page 6629)

TryEncodeDate

TryEncodeDateMonthWeek (see page 6696)

38.1.124 DateUtils.TryEncodeDateMonthWeek Function

Calculates the TDateTime value that represents a specified day of a specified week in a specified month and year.

Pascal

```
function TryEncodeDateMonthWeek(const AYear: Word; const AMonth: Word; const AWeekOfMonth:
Word; const ADayOfWeek: Word; var AValue: TDateTime): Boolean;
```

C++

```
Boolean TryEncodeDateMonthWeek(const Word AYear, const Word AMonth, const Word AWeekOfMonth,
const Word ADayOfWeek, TDateTime AValue);
```

File

DateUtils

Description

TryEncodeDateMonthWeek calculates the TDateTime value for the date specified by the AYear, AMonth, AWeekOfMonth, and ADayOfWeek parameters.

AYear is the year. It can be any value between 1 and 9999 (inclusive).

AMonth is the year. It can be any value between 1 and 12 (inclusive).

AWeekOfMonth is the week within that month, where 1 is the first week that includes four or more days. Note that if the first calendar day of the month is a Friday, Saturday, or Sunday, then those three days must be expressed using AMonth set to the previous month and AWeekOfMonth set to the number of weeks in the previous month. Similarly, if the last calendar day of the month is a Monday, Tuesday, or Wednesday, then those three days are expressed with AMonth set to the following month and AWeekOfMonth set to 1.

ADayOfWeek is the day of the week, where 1 Monday, 2 is Tuesday, and so on.

AValue returns the specified date as a TDateTime value. The time portion of AValue is 0 (Midnight at the start of the specified day).

TryEncodeDateMonthWeek returns true if AYear, AMonth, AWeekOfMonth, and ADayOfWeek represent a valid date, and false if any of these parameters are not in the valid range.

Note: The definitions for AWeekOfYear and ADayOfWeek follow the ISO 8601 standard.

Tip: To make the ADayOfWeek value more readable, use the Day of week constants.

See Also

TryEncodeDateTime (see page 6697)

YearOf (see page 6714)

TryEncodeDate

DayOfTheWeek (see page 6619)

DecodeDateMonthWeek (see page 6625)

TryEncodeDateDay (see page 6695)

EncodeDateMonthWeek (see page 6629)

EndOfAWeek (see page 6635)

TryEncodeTime

TryEncodeDateWeek (see page 6698)

38.1.125 DateUtils.TryEncodeDateTime Function

Calculates the TDateTime value that represents a specified year, month, day, hour, minute, second, and millisecond.

Pascal

```
function TryEncodeDateTime(const AYear: Word; const AMonth: Word; const ADay: Word; const
AHour: Word; const AMinute: Word; const ASecond: Word; const AMilliSecond: Word; out AValue:
TDateTime): Boolean;
```

C++

```
Boolean TryEncodeDateTime(const Word AYear, const Word AMonth, const Word ADay, const Word
AHour, const Word AMinute, const Word ASecond, const Word AMilliSecond, TDateTime AValue);
```

File

DateUtils

Description

TryEncodeDateTime calculates the TDateTime value for the values specified as the AYear, AMonth, ADay, AHour, AMinute, ASecond, and AMilliSecond parameters.

The year must be between 1 and 9999.

Valid month values are 1 through 12.

Valid day values are 1 through 28, 29, 30, or 31, depending on the month value. For example, the possible day values for month 2 (February) are 1 through 28 or 1 through 29, depending on whether or not the year value specifies a leap year.

Valid hour values are 0 through 24. (If the specified hour is 24, the minute, second, and millisecond values should all be 0, and the resulting TDateTime value represents midnight at the end of the specified day and the beginning of the next day).

Valid minute values are 0 through 59.

Valid second values are 0 through 59.

Valid millisecond values are 0 through 999.

AValue returns the specified date and time as a TDateTime value.

TryEncodeDateDay (see page 6695) returns true if the parameters are all within range, and false otherwise.

See Also

TryEncodeTime

TryEncodeDateDay (see page 6695)

YearOf (see page 6714)

MonthOf (see page 6670)

DayOf (see page 6618)

HourOf (see page 6638)

MinuteOf (see page 6663)

SecondOf (see page 6682)

MillisecondOf

TryEncodeDate

TryEncodeDateWeek (see page 6698)

EncodeDateTime (see page 6630)

TryEncodeDateMonthWeek (see page 6696)

38.1.126 DateUtils.TryEncodeDateWeek Function

Calculates the TDateTime value that represents a specified day of a specified week in a specified year.

Pascal

```
function TryEncodeDateWeek(const AYear: Word; const AWeekOfYear: Word; out AValue: TDateTime;
const ADayOfWeek: Word = 1): Boolean;
```

C++

```
Boolean TryEncodeDateWeek(const Word AYear, const Word AWeekOfYear, TDateTime AValue, const
Word ADayOfWeek = 1);
```

File

DateUtils

Description

TryEncodeDateWeek calculates the TDateTime value for the date specified by the AYear, AWeekOfYear, and ADayOfWeek parameters.

AYear is the year. It can be any value between 1 and 9999 (inclusive).

AWeekOfYear is the week within that year, where 1 is the first week that includes four or more days. Note that if the first calendar day of the year is a Friday, Saturday, or Sunday, then those three days must be expressed using AYear set to the previous year and AWeekOfYear set to the number of weeks in the previous year. Similarly, if the last calendar day of the year is a Monday, Tuesday, or Wednesday, then those three days are expressed with AYear set to the following year and AWeekOfYear set to 1.

ADayOfWeek is the day of the week, where 1 Monday, 2 is Tuesday, and so on.

AValue returns the specified date as a TDateTime value. The time portion of AValue is 0 (Midnight at the start of the specified day).

TryEncodeDateWeek returns true if AYear, AWeekOfYear, and ADayOfWeek represent a valid date, and false if any of these parameters are not in the valid range.

Note: The definitions for AWeekOfYear and ADayOfWeek follow the ISO 8601 standard.

Tip: To make the ADayOfWeek value more readable, use the Day of week constants.

See Also

TryEncodeDateTime (see page 6697)

YearOf (see page 6714)

TryEncodeDate

DayOfTheWeek (see page 6619)

DecodeDateWeek (see page 6627)

TryEncodeDateDay (see page 6695)

EncodeDateWeek (see page 6631)

EndOfAWeek (see page 6635)

TryEncodeTime

TryEncodeDateMonthWeek (see page 6696)

38.1.127 DateUtils.TryEncodeDayOfWeekInMonth Function

Calculates a TDateTime that represents a specified occurrence of a day of the week within a specified month and year.

Pascal

```
function TryEncodeDayOfWeekInMonth(const AYear: Word; const AMonth: Word; const ANthDayOfWeek:
Word; const ADayOfWeek: Word; out AValue: TDateTime): Boolean;
```

C++

```
Boolean TryEncodeDayOfWeekInMonth(const Word AYear, const Word AMonth, const Word
ANthDayOfWeek, const Word ADayOfWeek, TDateTime AValue);
```

File

DateUtils

Description

TryEncodeDayOfWeekInMonth returns a TDateTime for the date specified by the AYear, AMonth, ANthDayOfWeek, and

ADayOfWeek parameters. The time portion of the return value is 0 (Midnight at the start of the specified day).

AYear is the year. It can be any value between 1 and 9999 (inclusive).

AMonth is the month. It can be any value between 1 and 12 (inclusive).

ANthDayOfWeek indicates the occurrence for the day of the week represented by ADayOfWeek. It can be any value between 1 and 5 (inclusive), as long as this value specifies a valid date. For example, if a month does not include 5 Mondays, setting ANthDayOfWeek to 5 and ADayOfWeek to 1 causes TryEncodeDayOfWeekInMonth to return F false.

ADayOfWeek is the day of the week, where 1 Monday and 7 is Sunday.

TryEncodeDayOfWeekInMonth returns true if AYear, AMonth, ANthDayOfWeek, and ADayOfWeek represent a valid date, and false if this combination does not correspond to a valid date.

Note: The definition for ADayOfWeek follows the ISO 8601 standard.

Tip: To make the ADayOfWeek value more readable, use the Day of week constants.

See Also

TryEncodeDateTime (see page 6697)

YearOf (see page 6714)

MonthOf (see page 6670)

EncodeDayOfWeekInMonth (see page 6632)

TryEncodeDate

TryEncodeDateMonthWeek (see page 6696)

WeekOfTheMonth (see page 6704)

DecodeDayOfWeekInMonth (see page 6628)

38.1.128 DateUtils.TryJulianDateToDateTime Function

Converts a Julian date to a TDateTime value.

Pascal

```
function TryJulianDateToDateTime(const AValue: Double; out ADateTime: TDateTime): Boolean;
```

C++

```
Boolean TryJulianDateToDateTime(const Double AValue, TDateTime ADateTime);
```

File

DateUtils

Description

Call TryJulianDateToDateTime to convert a Julian date to the corresponding TDateTime value.

AValue is the Julian date to convert. The Julian date is the number of days, including fractional days, since 4713 BC January 1, Greenwich noon. Julian dates are used by astronomers, geophysicist, and others who need an unambiguous dating system based on continuing day counts.

ADateTime returns the corresponding TDateTime value.

TryJulianDateToDateTime returns true if it can successfully convert AValue to a TDateTime value. It returns false if it can't convert AValue into a TDateTime value.

See Also

DateTimeToModifiedJulianDate (see page 6617)

DateTimeToJulianDate (see page 6616)

TryModifiedJulianDateToDateTime (see page 6701)

Convert

JulianDateToDateTime (see page 6656)

ModifiedJulianDateToDateTime (see page 6669)

38.1.129 DateUtils.TryModifiedJulianDateToDateTime Function

Converts a modified Julian date to a TDateTime value.

Pascal

```
function TryModifiedJulianDateToDateTime(const AValue: Double; out ADateTime: TDateTime):
Boolean;
```

C++

```
Boolean TryModifiedJulianDateToDateTime(const Double AValue, TDateTime ADateTime);
```

File

DateUtils

Description

Call TryModifiedJulianDateToDateTime to convert a modified Julian date to the corresponding TDateTime value.

AValue is the modified Julian date to convert. The modified Julian date is the number of days, including fractional days, since Greenwich midnight on November 17, 1858. Modified Julian dates are based on Julian dates, but adjusted to use midnight rather than noon as a starting point and to use a more recent date as a starting point.

ADateTime returns the corresponding TDateTime value.

TryModifiedJulianDateToDateTime returns true if it can successfully convert AValue to a TDateTime value. It returns false if it can't convert AValue into a TDateTime value.

See Also

DateTimeToModifiedJulianDate (see page 6617)

DateTimeToJulianDate (see page 6616)

JulianDateToDateTime (see page 6656)

Convert

TryJulianDateToDateTime (see page 6700)

ModifiedJulianDateToDateTime (see page 6669)

38.1.130 DateUtils.TryRecodeDateTime Function

Selectively replaces parts of a specified TDateTime value.

Pascal

```
function TryRecodeDateTime(const AValue: TDateTime; const AYear: Word; const AMonth: Word;
const ADay: Word; const AHour: Word; const AMinute: Word; const ASecond: Word; const
AMilliSecond: Word; out AResult: TDateTime): Boolean;
```

C++

```
Boolean TryRecodeDateTime(const TDateTime AValue, const Word AYear, const Word AMonth, const
Word ADay, const Word AHour, const Word AMinute, const Word ASecond, const Word AMilliSecond,
TDateTime AResult);
```

File

DateUtils

Description

Call TryRecodeDateTime to convert the date/time value specified by AValue with the value obtained by changing the year to AYear, the month to AMonth, the day of the month to ADay, the hour to AHour, the minute to AMinute, the second to ASecond, and the millisecond to AMilliSecond. Any one of these parameters can be set to RecodeLeaveFieldAsIs (see page 6723), which tells TryRecodeDateTime not change the existing portion of the TDateTime value.

AYear can range from 1 to 9999 (or be RecodeLeaveFieldAsIs (see page 6723)).

AMonth can range from 1 to 12 (or be RecodeLeaveFieldAsIs (see page 6723)).

Legal values for ADay depend on the month represented by AMonth. (For example, if AMonth is 1, ADay can range from 1 to 31; if AMonth is 2, ADay can range from 1 to 28 or 1 to 29, depending on the year; and so on.) ADay can also be RecodeLeaveFieldAsIs (see page 6723).

AHour can range from 0 to 24 (or be RecodeLeaveFieldAsIs (see page 6723)). If AHour is 24, AMinute, ASecond, and AMilliSecond must all be 0.

AMinute can range from 0 to 59 (or be RecodeLeaveFieldAsIs (see page 6723)).

ASecond can range from 0 to 59 (or be RecodeLeaveFieldAsIs (see page 6723)).

AMilliSecond can range from 0 to 999 (or be RecodeLeaveFieldAsIs (see page 6723)).

AResult returns the results of making the specified changes to AValue.

TryRecodeDateTime returns true if the parameters are all in the valid range, false otherwise.

See Also

ReplaceDate

RecodeTime (see page 6679)

RecodeDate (see page 6673)

EConvertError

ReplaceTime

TryEncodeDateTime (see page 6697)

RecodeDateTime (see page 6674)

38.1.131 DateUtils.UnixToDateTime Function

Converts a Unix/Linux style date-and-time value to a TDateTime value.

Pascal

```
function UnixToDateTime(const AValue: Int64): TDateTime;
```

C++

```
TDateTime UnixToDateTime(const Int64 AValue);
```

File

DateUtils

Description

Call UnixToDateTime to convert the Unix/Linux encoding of a date and time value into the corresponding TDateTime.

Unix date-and-time values are encoded as the number of seconds since midnight at the start of January 1, 1970.

See Also

DateTimeToUnix (see page 6617)

Convert

38.1.132 DateUtils.WeekOf Function

Returns the week of the year represented by a TDateTime value.

Pascal

```
function WeekOf(const AValue: TDateTime): Word;
```

C++

```
Word WeekOf(const TDateTime AValue);
```

File

DateUtils

Description

Call WeekOf to obtain the week of the year represented by a specified TDateTime value. WeekOf returns a value between 1 and 53.

WeekOf uses the ISO 8601 standard to define the week of the year. That is, a week is defined as running from Monday through Sunday, and the first week of the year is defined as the one the first calendar week of the year is the one that includes the first Thursday of the year (the first week that includes four or more days in the year). This means that if the first calendar day of the year is a Friday, Saturday, or Sunday, then for the first three, two, or one days of the calendar year, WeekOf returns the last week of the previous year. Similarly, if the last calendar day of the year is a Monday, Tuesday, or Wednesday, then for the last one, two, or three days of the calendar year, WeekOf returns 1 (the first week of the next calendar year).

Note: WeekOf returns the same value as the WeekOfTheYear (see page 6705) function.

See Also

WeekOfTheYear (see page 6705)

WeekOfTheMonth (see page 6704)

HourOf (see page 6638)

MilliSecondOf (see page 6656)

MinuteOf (see page 6663)

MonthOf (see page 6670)

RecodeDate (see page 6673)

SecondOf (see page 6682)

DayOf (see page 6618)

YearOf (see page 6714)

38.1.133 DateUtils.WeekOfTheMonth Function

Returns the week of the month represented by a TDateTime value.

Pascal

```
function WeekOfTheMonth(const AValue: TDateTime): Word; overload;
function WeekOfTheMonth(const AValue: TDateTime; var AYear: Word; var AMonth: Word): Word;
overload;
```

C++

```
Word WeekOfTheMonth(const TDateTime AValue);
Word WeekOfTheMonth(const TDateTime AValue, Word AYear, Word AMonth);
```

File

DateUtils

Description

Call WeekOfTheMonth to obtain the week of the month represented by a specified TDateTime value. WeekOfTheMonth returns a value between 1 and 6.

AYear and AMonth return the year and month in which WeekOfTheMonth occurs. Note that these values may not be the same as the year and month of AValue. This is because the first week of a month is defined as the first week with four or more days in that month. Thus, if the first day of the month is a Friday, Saturday, or Sunday, the first one, two, or three days of the month are defined as belonging to the last week of the previous month. Similarly, if the last day of the month is a Monday, Tuesday, or Wednesday, then the last one, two, or three days of the month are defined as belonging to the first week of the next month.

Tip: To count weeks based on the weekday of the date/time value, use the NthDayOfWeek (see page 6672) function instead. For example, if AValue represents a Saturday that is the first day of a month, WeekOfTheMonth returns 5 (or maybe 4), indicating the last week of the previous month, while NthDayOfWeek (see page 6672) returns 1 AValue is the first Saturday in the month.

Note: WeekOfTheMonth uses the ISO 8601 standard definition of a week. That is, a week is considered to start on a Monday and end on a Sunday.

See Also

WeekOfTheYear (see page 6705)

WeekOf (see page 6703)

HourOfTheMonth (see page 6640)

MilliSecondOfTheMonth (see page 6659)

MinuteOfTheMonth (see page 6666)

SecondOfTheMonth (see page 6685)

DayOfTheMonth (see page 6618)

StartOfTheMonth (see page 6692)

WeeksBetween (see page 6706)

38.1.134 DateUtils.WeekOfTheYear Function

Returns the week of the year represented by a TDateTime value.

Pascal

```
function WeekOfTheYear(const AValue: TDateTime): Word; overload;
function WeekOfTheYear(const AValue: TDateTime; var AYear: Word): Word; overload;
```

C++

```
Word WeekOfTheYear(const TDateTime AValue);
Word WeekOfTheYear(const TDateTime AValue, Word AYear);
```

File

DateUtils

Description

Call WeekOfTheYear to obtain the week of the year represented by a specified TDateTime value. WeekOfTheYear returns a value between 1 and 53.

AYear returns the year in which the specified week occurs. Note that this may not be the same as the year of AValue. This is because the first week of a year is defined as the first week with four or more days in that year. This means that if the first calendar day of the year is a Friday, Saturday, or Sunday, then for the first three, two, or one days of the calendar year, WeekOfTheYear returns the last week of the previous year. Similarly, if the last calendar day of the year is a Monday, Tuesday, or Wednesday, then for the last one, two, or three days of the calendar year, WeekOfTheYear returns 1 (the first week of the next calendar year).

Note: WeekOfTheYear uses the ISO 8601 standard definition of a week. That is, a week is considered to start on a Monday and end on a Sunday.

Note: WeekOfTheYear returns the same value as the WeekOf (see page 6703) function.

See Also

WeekOfTheMonth (see page 6704)

HourOfTheYear (see page 6641)

MilliSecondOfTheYear (see page 6661)

MinuteOfTheYear (see page 6667)

MonthOfTheYear (see page 6670)

SecondOfTheYear (see page 6687)

WeekOf (see page 6703)

StartOfTheYear (see page 6693)

WeeksBetween (see page 6706)

WeeksInYear (see page 6707)

38.1.135 DateUtils.WeeksBetween Function

Returns the number of whole weeks between two specified TDateTime values.

Pascal

```
function WeeksBetween(const ANow: TDateTime; const AThen: TDateTime): Integer;
```

C++

```
int WeeksBetween(const TDateTime ANow, const TDateTime AThen);
```

File

DateUtils

Description

Call WeeksBetween to obtain the difference, in weeks, between two TDateTime values. WeeksBetween counts only whole Weeks. Thus, WeeksBetween reports the difference between January 1 at 12:00 AM and Jan 6 at 11:58 PM as 0 because the difference is one minute short of an entire week.

See Also

WeekSpan (see page 6708)

HoursBetween (see page 6642)

MillisecondsBetween

MinutesBetween (see page 6668)

MonthsBetween (see page 6671)

SecondsBetween (see page 6687)

DaysBetween (see page 6621)

YearsBetween (see page 6714)

WithinPastWeeks (see page 6712)

38.1.136 DateUtils.WeeksInAYear Function

Returns the number of weeks in a specified year.

Pascal

```
function WeeksInAYear(const AYear: Word): Word;
```

C++

```
Word WeeksInAYear(const Word AYear);
```

File

DateUtils

Description

Call WeeksInAYear to obtain the number of weeks in the year specified by AYear. AYear is a year between 1 and 9999 (inclusive).

Note: WeeksInAYear defines the first week of the year according to the ISO 8601 standard. That is, the first week of the year is the one that includes the first Thursday of the year (the first week that has 4 or more days in the year). This means that WeeksInAYear always returns either 52 or 53.

See Also

WeeksInYear (see page 6707)

WeeksBetween (see page 6706)

WeekOfTheYear (see page 6705)

IsLeapYear

38.1.137 DateUtils.WeeksInYear Function

Returns the number of weeks in the year of a specified TDateTime value.

Pascal

```
function WeeksInYear(const AValue: TDateTime): Word;
```

C++

```
Word WeeksInYear(const TDateTime AValue);
```

File

DateUtils

Description

Call WeeksInYear to obtain the number of weeks in the year of the TDateTime value specified by AValue.

Note: WeeksInYear defines the first week of the year according to the ISO 8601 standard. That is, the first week of the year is the one that includes the first Thursday of the year (the first week that has 4 or more days in the year). This means that WeeksInYear always returns either 52 or 53.

See Also

WeeksInAYear (see page 6706)

WeeksBetween (see page 6706)

WeekOfTheYear (see page 6705)

IsInLeapYear (see page 6650)

StartOfTheYear (see page 6693)

EndOfTheYear (see page 6638)

DaysInYear (see page 6623)

38.1.138 DateUtils.WeekSpan Function

Returns the number of weeks (including fractional weeks) between two specified TDateTime values.

Pascal

```
function WeekSpan(const ANow: TDateTime; const AThen: TDateTime): Double;
```

C++

```
Double WeekSpan(const TDateTime ANow, const TDateTime AThen);
```

File

DateUtils

Description

Call WeekSpan to obtain the difference, in weeks, between two TDateTime values. Unlike the WeeksBetween (see page 6706) function, which only counts whole weeks, WeekSpan reports incomplete weeks as a fraction of an entire week.

See Also

WeeksBetween (see page 6706)

HourSpan (see page 6643)

MillisecondSpan

MinuteSpan (see page 6668)

MonthSpan (see page 6672)

SecondSpan (see page 6688)

DaySpan (see page 6624)

YearSpan (see page 6715)

38.1.139 DateUtils.WithinPastDays Function

Indicates whether two dates are within a specified number of days of each other.

Pascal

```
function WithinPastDays(const ANow: TDateTime; const AThen: TDateTime; const ADays: Integer):
Boolean;
```

C++

```
Boolean WithinPastDays(const TDateTime ANow, const TDateTime AThen, const int ADays);
```

File

DateUtils

Description

Call WithinPastDays to determine whether the date specified by ANow is within ADays days of the date specified by AThen.

Note: WithinPastDays uses the DaysBetween (see page 6621) function, which means that fractional days do not count. If ANow and AThen are 2 and a half days apart, calling WithinPastDays with ADays set to 2 returns true.

See Also

DaysBetween (see page 6621)

WithinPastHours (see page 6709)

WithinPastMilliSeconds (see page 6710)

WithinPastSeconds (see page 6712)

WithinPastMinutes (see page 6710)

WithinPastWeeks (see page 6712)

WithinPastMonths (see page 6711)

WithinPastYears (see page 6713)

38.1.140 DateUtils.WithinPastHours Function

Indicates whether two date/time values are within a specified number of hours of each other.

Pascal

```
function WithinPastHours(const ANow: TDateTime; const AThen: TDateTime; const AHours: Int64):
Boolean;
```

C++

```
Boolean WithinPastHours(const TDateTime ANow, const TDateTime AThen, const Int64 AHours);
```

File

DateUtils

Description

Call WithinPastHours to determine whether the date and time specified by ANow is within AHours hours of the date and time specified by AThen.

Note: WithinPastHours uses the HoursBetween (see page 6642) function, which means that fractional hours do not count. If ANow and AThen are 2 and a half hours apart, calling WithinPastHours with AHours set to 2 returns true.

See Also

HoursBetween (see page 6642)

WithinPastDays (see page 6708)

WithinPastMilliSeconds (see page 6710)

WithinPastSeconds (see page 6712)

WithinPastMinutes (see page 6710)

WithinPastWeeks (see page 6712)

WithinPastMonths (see page 6711)

WithinPastYears (see page 6713)

38.1.141 DateUtils.WithinPastMilliSeconds Function

Indicates whether two date/time values are within a specified number of milliseconds of each other.

Pascal

```
function WithinPastMilliSeconds(const ANow: TDateTime; const AThen: TDateTime; const
AMilliSeconds: Int64): Boolean;
```

C++

```
Boolean WithinPastMilliSeconds(const TDateTime ANow, const TDateTime AThen, const Int64
AMilliSeconds);
```

File

DateUtils

Description

Call WithinPastMilliSeconds to determine whether the date and time specified by ANow is within AMilliSeconds milliseconds of the date and time specified by AThen.

See Also

MilliSecondsBetween (see page 6662)

WithinPastDays (see page 6708)

WithinPastHours (see page 6709)

WithinPastSeconds (see page 6712)

WithinPastMinutes (see page 6710)

WithinPastWeeks (see page 6712)

WithinPastMonths (see page 6711)

WithinPastYears (see page 6713)

38.1.142 DateUtils.WithinPastMinutes Function

Indicates whether two date/time values are within a specified number of minutes of each other.

Pascal

```
function WithinPastMinutes(const ANow: TDateTime; const AThen: TDateTime; const AMinutes:
Int64): Boolean;
```

C++

```
Boolean WithinPastMinutes(const TDateTime ANow, const TDateTime AThen, const Int64 AMinutes);
```

File

DateUtils

Description

Call WithinPastMinutes to determine whether the date and time specified by ANow is within AMinutes minutes of the date and time specified by AThen.

Note: WithinPastMinutes uses the MinutesBetween (see page 6668) function, which means that fractional minutes do not count. If ANow and AThen are 2 and a half minutes apart, calling WithinPastMinutes with AMinutes set to 2 returns true.

See Also

MinutesBetween (see page 6668)

WithinPastDays (see page 6708)

WithinPastMilliSeconds (see page 6710)

WithinPastSeconds (see page 6712)

WithinPastHours (see page 6709)

WithinPastWeeks (see page 6712)

WithinPastMonths (see page 6711)

WithinPastYears (see page 6713)

38.1.143 DateUtils.WithinPastMonths Function

Indicates whether two date/time values are within a specified number of months of each other.

Pascal

```
function WithinPastMonths(const ANow: TDateTime; const AThen: TDateTime; const AMonths:
Integer): Boolean;
```

C++

```
Boolean WithinPastMonths(const TDateTime ANow, const TDateTime AThen, const int AMonths);
```

File

DateUtils

Description

Call WithinPastMonths to determine whether the date specified by ANow is within AMonths months of the date specified by AThen.

Note: WithinPastMonths uses the MonthsBetween (see page 6671) function, which means that fractional months do not count. If ANow and AThen are 2 and a half months apart, calling WithinPastMonths with AMonths set to 2 returns true.

See Also

MonthsBetween (see page 6671)

WithinPastDays (see page 6708)

WithinPastMilliSeconds (see page 6710)

WithinPastSeconds (see page 6712)

WithinPastHours (see page 6709)

WithinPastWeeks (see page 6712)

WithinPastMinutes (see page 6710)

WithinPastYears (see page 6713)

38.1.144 DateUtils.WithinPastSeconds Function

Indicates whether two date/time values are within a specified number of seconds of each other.

Pascal

```
function WithinPastSeconds(const ANow: TDateTime; const AThen: TDateTime; const ASeconds:
Int64): Boolean;
```

C++

```
Boolean WithinPastSeconds(const TDateTime ANow, const TDateTime AThen, const Int64 ASeconds);
```

File

DateUtils

Description

Call WithinPastSeconds to determine whether the date and time specified by ANow is within ASeconds seconds of the date and time specified by AThen.

Note: WithinPastSeconds uses the SecondsBetween (see page 6687) function, which means that fractional seconds do not count. If ANow and AThen are 2 and a half seconds apart, calling WithinPastSeconds with ASeconds set to 2 returns true.

See Also

SecondsBetween (see page 6687)

WithinPastDays (see page 6708)

WithinPastMilliSeconds (see page 6710)

WithinPastMinutes (see page 6710)

WithinPastHours (see page 6709)

WithinPastWeeks (see page 6712)

WithinPastMonths (see page 6711)

WithinPastYears (see page 6713)

38.1.145 DateUtils.WithinPastWeeks Function

Indicates whether two date/time values are within a specified number of weeks of each other.

Pascal

```
function WithinPastWeeks(const ANow: TDateTime; const AThen: TDateTime; const AWeeks:
Integer): Boolean;
```

C++

```
Boolean WithinPastWeeks(const TDateTime ANow, const TDateTime AThen, const int AWeeks);
```

File

DateUtils

Description

Call WithinPastWeeks to determine whether the date specified by ANow is within AWeeks weeks of the date specified by AThen.

Note: WithinPastWeeks uses the WeeksBetween (see page 6706) function, which means that fractional weeks do not count. If ANow and AThen are 2 and a half weeks apart, calling WithinPastWeeks with AWeeks set to 2 returns true.

See Also

WeeksBetween (see page 6706)

WithinPastDays (see page 6708)

WithinPastMilliSeconds (see page 6710)

WithinPastSeconds (see page 6712)

WithinPastHours (see page 6709)

WithinPastMinutes (see page 6710)

WithinPastMonths (see page 6711)

WithinPastYears (see page 6713)

38.1.146 DateUtils.WithinPastYears Function

Indicates whether two date/time values are within a specified number of years of each other.

Pascal

```
function WithinPastYears(const ANow: TDateTime; const AThen: TDateTime; const AYears:
Integer): Boolean;
```

C++

```
Boolean WithinPastYears(const TDateTime ANow, const TDateTime AThen, const int AYears);
```

File

DateUtils

Description

Call WithinPastYears to determine whether the date specified by ANow is within AYears years of the date specified by AThen.

Note: WithinPastYears uses the YearsBetween (see page 6714) function, which means that fractional years do not count. If ANow and AThen are 2 and a half years apart, calling WithinPastYears with AYears set to 2 returns true.

See Also

YearsBetween (see page 6714)

WithinPastDays (see page 6708)

WithinPastMilliSeconds (see page 6710)

WithinPastSeconds (see page 6712)

WithinPastHours (see page 6709)

WithinPastWeeks (see page 6712)

WithinPastMonths (see page 6711)

WithinPastMinutes (see page 6710)

38.1.147 DateUtils.YearOf Function

Returns the year represented by a TDateTime value.

Pascal

```
function YearOf(const AValue: TDateTime): Word;
```

C++

```
Word YearOf(const TDateTime AValue);
```

File

DateUtils

Description

Call YearOf to obtain the year represented by a specified TDateTime value. YearOf returns a value between 1 and 9999.

See Also

DecodeDate

DayOf (see page 6618)

MilliSecondOf (see page 6656)

MinuteOf (see page 6663)

HourOf (see page 6638)

RecodeDateYear

SecondOf (see page 6682)

WeekOf (see page 6703)

MonthOf (see page 6670)

38.1.148 DateUtils.YearsBetween Function

Returns the approximate number of years between two specified TDateTime values.

Pascal

```
function YearsBetween(const ANow: TDateTime; const AThen: TDateTime): Integer;
```

C++

```
int YearsBetween(const TDateTime ANow, const TDateTime AThen);
```

File

DateUtils

Description

Call YearsBetween to obtain the difference, in years, between two TDateTime values. Because years are not all the same length (e.g. leap years), YearsBetween returns an approximation based on an assumption of 365.25 days per year. Fractional years are

not counted. Thus, for example, YearsBetween reports the difference between Jan 1 and Dec 31 as 0 on non-leap years and 1 on leap years.

See Also

YearSpan (see page 6715)

DaysBetween (see page 6621)

MillisecondsBetween

MinutesBetween (see page 6668)

HoursBetween (see page 6642)

SecondsBetween (see page 6687)

WeeksBetween (see page 6706)

MonthsBetween (see page 6671)

WithinPastYears (see page 6713)

38.1.149 DateUtils.YearSpan Function

Returns the approximate number of years (including fractions thereof) between two specified TDateTime values.

Pascal

```
function YearSpan(const ANow: TDateTime; const AThen: TDateTime): Double;
```

C++

```
Double YearSpan(const TDateTime ANow, const TDateTime AThen);
```

File

DateUtils

Description

Call YearSpan to obtain the difference, in years, between two TDateTime values. Because years are not all the same length (e.g. leap years), YearsBetween (see page 6714) returns an approximation based on an assumption of 365.25 days per year. Unlike the YearsBetween (see page 6714) function, which does not report fractional years, YearSpan reports incomplete years as a fraction of a 365.25-day year.

See Also

YearsBetween (see page 6714)

DaySpan (see page 6624)

MillisecondSpan

MinuteSpan (see page 6668)

HourSpan (see page 6643)

SecondSpan (see page 6688)

WeekSpan (see page 6708)

MonthSpan (see page 6672)

38.1.150 DateUtils.Yesterday Function

Returns a TDateTime value that represents the preceding day.

Pascal

```
function Yesterday: TDateTime;
```

C++

```
TDateTime Yesterday();
```

File

DateUtils

Description

Yesterday returns a TDateTime value with the date portion set to the day before the current date and the time portion set to 0.

See Also

Today (see page 6694)

Date

Tomorrow (see page 6695)

38.2 Variables

The following table lists variables in this documentation.

Variables

Name	Description
ApproxDaysPerMonth (see page 6717)	This is variable DateUtils.ApproxDaysPerMonth.
ApproxDaysPerYear (see page 6717)	This is variable DateUtils.ApproxDaysPerYear.

38.2.1 DateUtils.ApproxDaysPerMonth Variable

Pascal

```
ApproxDaysPerMonth: Double = 30.4375;
```

C++

```
Double ApproxDaysPerMonth = 30.4375;
```

File

DateUtils

Description

This is variable DateUtils.ApproxDaysPerMonth.

38.2.2 DateUtils.ApproxDaysPerYear Variable

Pascal

```
ApproxDaysPerYear: Double = 365.25;
```

C++

```
Double ApproxDaysPerYear = 365.25;
```

File

DateUtils

Description

This is variable DateUtils.ApproxDaysPerYear.

38.3 Constants

The following table lists constants in this documentation.

Constants

Name	Description
DayFriday (see page 6718)	This is constant DateUtils.DayFriday.
DayMonday (see page 6718)	This is constant DateUtils.DayMonday.
DaySaturday (see page 6719)	This is constant DateUtils.DaySaturday.
DaysPerWeek (see page 6719)	This is constant DateUtils.DaysPerWeek.
DaysPerYear (see page 6719)	This is constant DateUtils.DaysPerYear.
DaySunday (see page 6720)	This is constant DateUtils.DaySunday.
DayThursday (see page 6720)	This is constant DateUtils.DayThursday.
DayTuesday (see page 6720)	This is constant DateUtils.DayTuesday.
DayWednesday (see page 6721)	This is constant DateUtils.DayWednesday.
MonthsPerYear (see page 6721)	This is constant DateUtils.MonthsPerYear.
OneHour (see page 6721)	Delphi inverse time unit constants.
OneMillisecond (see page 6722)	This is constant DateUtils.OneMillisecond.
OneMinute (see page 6722)	This is constant DateUtils.OneMinute.
OneSecond (see page 6722)	This is constant DateUtils.OneSecond.
RecodeLeaveFieldAsIs (see page 6723)	This is constant DateUtils.RecodeLeaveFieldAsIs.
WeeksPerFortnight (see page 6723)	This is constant DateUtils.WeeksPerFortnight.
YearsPerCentury (see page 6723)	This is constant DateUtils.YearsPerCentury.
YearsPerDecade (see page 6724)	This is constant DateUtils.YearsPerDecade.
YearsPerMillennium (see page 6724)	This is constant DateUtils.YearsPerMillennium.

38.3.1 DateUtils.DayFriday Constant

Pascal

```
DayFriday = 5;
```

C++

```
DayFriday = 5;
```

File

DateUtils

Description

This is constant DateUtils.DayFriday.

38.3.2 DateUtils.DayMonday Constant

Pascal

```
DayMonday = 1;
```

C++

```
DayMonday = 1;
```

File

DateUtils

Description

This is constant DateUtils.DayMonday.

38.3.3 DateUtils.DaySaturday Constant

Pascal

```
DaySaturday = 6;
```

C++

```
DaySaturday = 6;
```

File

DateUtils

Description

This is constant DateUtils.DaySaturday.

38.3.4 DateUtils.DaysPerWeek Constant

Pascal

```
DaysPerWeek = 7;
```

C++

```
DaysPerWeek = 7;
```

File

DateUtils

Description

This is constant DateUtils.DaysPerWeek.

38.3.5 DateUtils.DaysPerYear Constant

Pascal

```
DaysPerYear: array [Boolean] of Word = (365, 366);
```

C++

```
array [Boolean] of Word DaysPerYear = (365, 366);
```

File

DateUtils

Description

This is constant DateUtils.DaysPerYear.

38.3.6 DateUtils.DaySunday Constant

Pascal

```
DaySunday = 7;
```

C++

```
DaySunday = 7;
```

File

DateUtils

Description

This is constant DateUtils.DaySunday.

38.3.7 DateUtils.DayThursday Constant

Pascal

```
DayThursday = 4;
```

C++

```
DayThursday = 4;
```

File

DateUtils

Description

This is constant DateUtils.DayThursday.

38.3.8 DateUtils.DayTuesday Constant

Pascal

```
DayTuesday = 2;
```

C++

```
DayTuesday = 2;
```

File

DateUtils

Description

This is constant DateUtils.DayTuesday.

38.3.9 DateUtils.DayWednesday Constant

Pascal

```
DayWednesday = 3;
```

C++

```
DayWednesday = 3;
```

File

DateUtils

Description

This is constant DateUtils.DayWednesday.

38.3.10 DateUtils.MonthsPerYear Constant

Pascal

```
MonthsPerYear = 12;
```

C++

```
MonthsPerYear = 12;
```

File

DateUtils

Description

This is constant DateUtils.MonthsPerYear.

38.3.11 DateUtils.OneHour Constant

Delphi inverse time unit constants.

Pascal

```
OneHour = 1 / HoursPerDay;
```

C++

```
OneHour = 1 / HoursPerDay;
```

File

DateUtils

Description

These constants are inverses of the time constants defined in the SysUtils (see page 22709) unit.

See Also

Time

38.3.12 DateUtils.OneMillisecond Constant

Pascal

```
OneMillisecond = 1 / MSecsPerDay;
```

C++

```
OneMillisecond = 1 / MSecsPerDay;
```

File

DateUtils

Description

This is constant DateUtils.OneMillisecond.

38.3.13 DateUtils.OneMinute Constant

Pascal

```
OneMinute = 1 / MinsPerDay;
```

C++

```
OneMinute = 1 / MinsPerDay;
```

File

DateUtils

Description

This is constant DateUtils.OneMinute.

38.3.14 DateUtils.OneSecond Constant

Pascal

```
OneSecond = 1 / SecsPerDay;
```

C++

```
OneSecond = 1 / SecsPerDay;
```

File

DateUtils

Description

This is constant DateUtils.OneSecond.

38.3.15 DateUtils.RecodeLeaveFieldAsIs Constant

Pascal

```
RecodeLeaveFieldAsIs = High(Word);
```

C++

```
RecodeLeaveFieldAsIs = High(Word);
```

File

DateUtils

Description

This is constant DateUtils.RecodeLeaveFieldAsIs.

38.3.16 DateUtils.WeeksPerFortnight Constant

Pascal

```
WeeksPerFortnight = 2;
```

C++

```
WeeksPerFortnight = 2;
```

File

DateUtils

Description

This is constant DateUtils.WeeksPerFortnight.

38.3.17 DateUtils.YearsPerCentury Constant

Pascal

```
YearsPerCentury = 100;
```

C++

```
YearsPerCentury = 100;
```

File

DateUtils

Description

This is constant DateUtils.YearsPerCentury.

38.3.18 DateUtils.YearsPerDecade Constant

Pascal

```
YearsPerDecade = 10;
```

C++

```
YearsPerDecade = 10;
```

File

DateUtils

Description

This is constant DateUtils.YearsPerDecade.

38.3.19 DateUtils.YearsPerMillennium Constant

Pascal

```
YearsPerMillennium = 1000;
```

C++

```
YearsPerMillennium = 1000;
```

File

DateUtils

Description

This is constant DateUtils.YearsPerMillennium.

39 DB Namespace

This is namespace DB.

Topics

Name	Description
Classes (see page 6730)	The following table lists classes in this documentation.
Interfaces (see page 7569)	The following table lists interfaces in this documentation.
Functions (see page 7599)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 7604)	The following table lists structs, records, enums in this documentation.
Types (see page 7625)	The following table lists types in this documentation.
Variables (see page 7641)	The following table lists variables in this documentation.
Constants (see page 7645)	The following table lists constants in this documentation.

Classes

	Name	Description
	EDatabaseError (see page 6731)	EDatabaseError is the exception class for database errors.
	EUpdateError (see page 6732)	EUpdateError is the exception class for dataset provider update errors.
	TADTField (see page 6736)	A TADTField object represents an ADT (Abstract Data Type) field in a dataset.
	TAggregateField (see page 6743)	TAggregate field represents a maintained aggregate in a client dataset.
	TArrayField (see page 6760)	TArrayField represents an Array field in a dataset.
	TAutoIncField (see page 6768)	TAutoIncField is a persistent field object for an autoincrement field in a dataset.
	TBCDField (see page 6776)	TBCDField represents a binary-coded decimal (BCD) field in a dataset.
	TBinaryField (see page 6791)	TBinaryField represents an untyped binary field in a dataset.
	TBlobField (see page 6801)	TBlobField represents a field in a dataset that holds a reference to a binary large object (BLOB).
	TBooleanField (see page 6822)	TBooleanField represents a field containing Boolean values.
	TBytesField (see page 6833)	TBytesField represents a bytes field in a dataset.
	TCheckConstraint (see page 6841)	TCheckConstraint represents a record-level constraint on a dataset component.
	TCheckConstraints (see page 6846)	TCheckConstraints contains a collection of check constraints for a dataset component.
	TCurrencyField (see page 6850)	TCurrencyField represents a field that contains currency (see page 6857) values in a dataset.
	TCustomConnection (see page 6858)	TCustomConnection is the base class for components that connect datasets to a remote source of data.
	TDataLink (see page 6876)	TDataLink is a helper class used by data-aware objects to coordinate the actions of TDataSource (see page 7070) and TDataSet (see page 6894), and to respond to data events.

	TDataSet (see page 6894)	TDataSet is the base class for all dataset components that represent data in rows and columns.
	TDataSetDesigner (see page 7056)	TDataSetDesigner is the base class for all dataset designers.
	TDataSetField (see page 7060)	TDataSetField provides access to nested datasets.
	TDataSource (see page 7070)	TDataSource provides an interface between a dataset component and data-aware controls on a form.
	TDateField (see page 7083)	TDateField represents a date field in a dataset.
	TDateTimeField (see page 7090)	TDateTimeField represents a date-time field in a dataset.
	TDefCollection (see page 7104)	TDefCollection holds the definition objects (TNamedItem (see page 7357)) that represent the specifications for database entities such as fields or indexes.
	TDetailDataLink (see page 7111)	TDetailDataLink is the ancestor to datalink classes that manage the relationships between datasets that are linked in master/detail relationships.
	TField (see page 7115)	TField is the common ancestor of all the field components.
	TFieldDef (see page 7184)	TFieldDef is a field definition that corresponds to a physical field of a record in a table underlying a dataset.
	TFieldDefList (see page 7195)	TFieldDefList lists the field definition (TFieldDef (see page 7184)) objects of a dataset in a flattened form.
	TFieldDefs (see page 7198)	TFieldDefs holds the field definition (TFieldDef (see page 7184)) objects that represent the physical fields underlying a dataset.
	TFieldList (see page 7207)	TFieldList lists the fields of a dataset, flattening out the structure of any object fields.
	TFields (see page 7211)	TFields holds the field (TField (see page 7115)) objects that represent the physical fields underlying a dataset or an object field such as TADTField (see page 6736) and TArrayField (see page 6760).
	TFieldsEnumerator (see page 7222)	Enumerates database fields.
	TFlatList (see page 7225)	TFlatList is an abstract base class for objects that represent a flattened list of fields or field definitions.
	TFloatField (see page 7229)	TFloatField represents a field that contains floating-point values in a dataset.
	TFMTBCDField (see page 7242)	TFMTBCDField represents a binary-coded decimal (BCD) field in a dataset.
	TGraphicField (see page 7257)	TGraphicField represents a graphics field in a dataset.
	TGuidField (see page 7265)	TGuidField represents a guid field in a dataset.
	TIDispatchField (see page 7274)	TIDispatchField represents an IDispatch field in a dataset.
	TIndexDef (see page 7283)	TIndexDef describes an index in a database table.
	TIndexDefs (see page 7292)	TIndexDefs holds the list of available index definitions for a table.
	TIntegerField (see page 7301)	TIntegerField represents a signed 32-bit integer field in a dataset.
	TInterfaceField (see page 7314)	TInterfaceField represents an interface field in a dataset.
	TLargeintField (see page 7323)	A TLargeintField object represents a large integer field in a dataset.
	TLookupList (see page 7336)	TLookupList maintains a list of lookup values associated with a set of key field values.
	TMasterDataLink (see page 7339)	TMasterDataLink allows a dataset to establish a master/detail relationship.
	TMemoField (see page 7348)	TMemoField represents a memo field in a dataset.
	TNamedItem (see page 7357)	TNamedItem is the base class for definitions used by datasets.
	TNumericField (see page 7361)	TNumericField is the common ancestor of all the numeric fields in a dataset.
	TObjectField (see page 7372)	TObjectField is the common ancestor of TADTField (see page 6736), TArrayField (see page 6760), TDataSetField (see page 7060), and TReferenceField (see page 7452) components in a dataset.
	TParam (see page 7386)	TParam represents a field parameter for a query or stored procedure.
	TParams (see page 7434)	TParams manages a list of field parameters.
	TReferenceField (see page 7452)	TReferenceField represents a REF field in a dataset.
	TSmallintField (see page 7461)	TSmallIntField represents a signed 16-bit integer field in a dataset.
	TSQLTimeStampField (see page 7469)	TSQLTimeStampField represents a date-time field in a DBExpress dataset.
	TStringField (see page 7483)	TStringField represents a string field in a dataset.
	TTimeField (see page 7499)	TTimeField represents a time field in a dataset.
	TVarBytesField (see page 7507)	TVarBytesField represents a variable-length untyped binary field in a dataset.
	TVariantField (see page 7514)	TVariantField represents a variant field in a dataset.
	TWideDataSet (see page 7526)	Base class for all dataset components that represent Unicode data in rows and columns.
	TWideMemoField (see page 7541)	TWideMemoField represents a memo field in a dataset.
	TWideStringField (see page 7549)	TWideStringField represents a large string field in a dataset.

	TWordField (see page 7560)	TWordField represents an unsigned 16-bit integer field in a dataset.

Constants

Name	Description
dsEditModes (see page 7645)	This is constant DB.dsEditModes.
dsMaxStringSize (see page 7645)	This is constant DB.dsMaxStringSize.
dsWriteModes (see page 7646)	This is constant DB.dsWriteModes.
FieldTypeNames (see page 7646)	This is constant DB.FieldTypeNames.
FieldTypeVarMap (see page 7646)	This is constant DB.FieldTypeVarMap.
ftFixedSizeTypes (see page 7647)	This is constant DB.ftFixedSizeTypes.
ftNonTextTypes (see page 7647)	This is constant DB.ftNonTextTypes.
ObjectFieldTypes (see page 7648)	This is constant DB.ObjectFieldTypes.
RegisterFieldsProc (see page 7648)	This is constant DB.RegisterFieldsProc.

Functions

	Name	Description
	BuffersEqual (see page 7599)	This is function DB.BuffersEqual.
	DatabaseError (see page 7599)	Creates and raises an EDatabaseError (see page 6731) exception.
	DatabaseErrorFmt (see page 7600)	Creates and raises an EDatabaseError (see page 6731) exception with a formatted error message.
	DisposeMem (see page 7600)	This is function DB.DisposeMem.
	ExtractFieldName (see page 7601)	Returns the next name in a semicolon-delimited string of field names.
	ExtractFieldName (see page 7601)	Returns the next name in a semicolon-delimited string of field names.
	GetFieldProperty (see page 7602)	Returns a TField (see page 7115) object for a dataset and data control.
	RegisterFields (see page 7602)	Registers a set of field components so that they appear in the Dataset designer.
	VarTypeToDataType (see page 7603)	Returns the field type value that most closely corresponds to a Variant type.

Interfaces

	Name	Description
	IDBApplication (see page 7569)	Interface for database application.
	IDBScreen (see page 7571)	Interface to display device.
	IDBSession (see page 7573)	IDBSession is the interface for database sessions.
	IProviderSupport (see page 7576)	IProviderSupport is the interface that datasets use to enable a provider to represent them.
	IProviderSupport2 (see page 7587)	Interface datasets use to enable providers to represent them with Unicode support.

Structs, Records, Enums

	Name	Description
	TAutoRefreshFlag (see page 7604)	TAutoRefreshFlag Indicates whether the server provides field values, and if so, why.
	TBlobStreamMode (see page 7605)	TBlobStreamMode indicates how a BLOB stream can be used.
	TBookmarkFlag (see page 7606)	This is record DB.TBookmarkFlag.
	TDataAction (see page 7606)	TDataAction indicates how an application should respond to a database error condition.
	TDataEvent (see page 7607)	TDataEvent identifies an event that occurs to a dataset.
	TDataSetState (see page 7608)	TDataSetState indicates the current operating mode of a dataset component.
	TDateTimeRec (see page 7610)	This is record DB.TDateTimeRec.
	TDBScreenCursor (see page 7610)	This is record DB.TDBScreenCursor.
	TFieldAttribute (see page 7611)	TFieldAttribute and TFieldAttributes (see page 7632) indicate various attributes of a field.
	TFieldKind (see page 7611)	TFieldKind indicates what type of information a field component represents.
	TFieldType (see page 7612)	TFieldType is a set of values that parallel the data types of fields in tables.
	TFilterOption (see page 7615)	TFilterOption and TFilterOptions (see page 7635) influence how filter strings are applied.
	TGetMode (see page 7615)	This is record DB.TGetMode.
	TGetResult (see page 7616)	This is record DB.TGetResult.

	TGroupPosInd (see page 7616)	TGroupPosInd and TGroupPosInds (see page 7636) indicate the position of the current record within its group.
	TIndexOption (see page 7617)	This is record DB.TIndexOption.
	TLocateOption (see page 7618)	This is record DB.TLocateOption.
	TLookupListEntry (see page 7618)	This is record DB.TLookupListEntry.
	TPacketAttribute (see page 7618)	This is record DB.TPacketAttribute.
	TParamType (see page 7619)	TParamType and TParamTypes (see page 7637) represent the use of parameters on datasets that represent queries or stored procedures.
	TProviderFlag (see page 7620)	TProviderFlag and TProviderFlags (see page 7638) indicate how a provider uses a field value when applying updates.
	TPSCommandType (see page 7621)	This is record DB.TPSCommandType.
	TUpdateAction (see page 7621)	TUpdateAction Indicates how an a dataset should proceed after an event handler has processed an update.
	TUpdateKind (see page 7622)	TUpdateKind defines the values for the UpdateKind parameter of OnUpdateError and OnUpdateRecord event handlers.
	TUpdateMode (see page 7623)	TUpdateMode indicates how to locate records so that they can be updated.
	TUpdateStatus (see page 7623)	TUpdateStatus indicates the update status of a record.

Types

Name	Description
Largeint (see page 7626)	This is type DB.Largeint.
PBookmarkFlag (see page 7626)	This is type DB.PBookmarkFlag.
PLookupListEntry (see page 7626)	This is type DB.PLookupListEntry.
PPacketAttribute (see page 7627)	This is type DB.PPacketAttribute.
TBlobByteData (see page 7627)	TBlobByteData stores the contents of a Binary Large Object (BLOB) field.
TBlobData (see page 7627)	This is type DB.TBlobData.
TBlobType (see page 7628)	TBlobType indicates the type of a BLOB field.
TBookmark (see page 7628)	TBookmark identifies a record in a dataset for later navigation.
TBookmarkStr (see page 7629)	TBookmarkStr is a bookmark, represented as a string.
TBufferList (see page 7629)	This is type DB.TBufferList.
TConnectChangeEvent (see page 7630)	This is type DB.TConnectChangeEvent.
TDataChangeEvent (see page 7630)	This is type DB.TDataChangeEvent.
TDataOperation (see page 7630)	This is type DB.TDataOperation.
TDataSetClass (see page 7631)	TDataSetClass is the class type of a TDataSet (see page 6894) descendant.
TDataSetErrorEvent (see page 7631)	This is type DB.TDataSetErrorEvent.
TDataSetNotifyEvent (see page 7631)	This is type DB.TDataSetNotifyEvent.
TDateTimeAlias (see page 7632)	This is type DB.TDateTimeAlias.
TDefUpdateMethod (see page 7632)	This is type DB.TDefUpdateMethod.
TFieldAttributes (see page 7632)	This is type DB.TFieldAttributes.
TFieldChars (see page 7633)	TFieldChars indicates the set of characters that are valid for field.
TFieldClass (see page 7633)	TFieldClass is the class type of a TField (see page 7115) descendant.
TFieldGetTextEvent (see page 7633)	This is type DB.TFieldGetTextEvent.
TFieldKinds (see page 7634)	This is type DB.TFieldKinds.
TFieldNotifyEvent (see page 7634)	This is type DB.TFieldNotifyEvent.
TFieldRef (see page 7634)	This is type DB.TFieldRef.
TFieldSetTextEvent (see page 7635)	This is type DB.TFieldSetTextEvent.
TFilterOptions (see page 7635)	This is type DB.TFilterOptions.
TFilterRecordEvent (see page 7635)	This is type DB.TFilterRecordEvent.
TGroupPosInds (see page 7636)	This is type DB.TGroupPosInds.
TIndexOptions (see page 7636)	TIndexOptions describes the attributes of an index.
TLocateOptions (see page 7637)	TLocateOptions defines the values for the Options parameter of the Locate method of a dataset.
TParamTypes (see page 7637)	This is type DB.TParamTypes.
TProviderFlags (see page 7638)	This is type DB.TProviderFlags.
TResyncMode (see page 7638)	TResyncMode is a set of flags that indicate operations a dataset should perform when resynchronizing data.
TSQLCommandType (see page 7638)	This is type DB.TSQLCommandType.

TUpdateErrorEvent (see page 7639)	This is type DB.TUpdateErrorEvent.
TUpdateRecordEvent (see page 7639)	This is type DB.TUpdateRecordEvent.
TUpdateRecordTypes (see page 7639)	This is type DB.TUpdateRecordTypes.
TUpdateStatusSet (see page 7640)	This is type DB.TUpdateStatusSet.

Variables

Name	Description
DBApplication (see page 7641)	This is variable DB.DBApplication.
DBScreen (see page 7641)	This is variable DB.DBScreen.
DefaultFieldClasses (see page 7642)	This is variable DB.DefaultFieldClasses.
LoginDialogExProc (see page 7642)	This is variable DB.LoginDialogExProc.
LoginDialogProc (see page 7642)	This is variable DB.LoginDialogProc.
PasswordDialog (see page 7643)	Displays a dialog box that prompts the user for the password of a local password-protected table.
RemoteLoginDialogProc (see page 7643)	This is variable DB.RemoteLoginDialogProc.
ScreenCursorProc (see page 7644)	This is variable DB.ScreenCursorProc.

39.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EDatabaseError (see page 6731)	EDatabaseError is the exception class for database errors.
	EUpdateError (see page 6732)	EUpdateError is the exception class for dataset provider update errors.
	TADTField (see page 6736)	A TADTField object represents an ADT (Abstract Data Type) field in a dataset.
	TAggregateField (see page 6743)	TAggregate field represents a maintained aggregate in a client dataset.
	TArrayField (see page 6760)	TArrayField represents an Array field in a dataset.
	TAutoIncField (see page 6768)	TAutoIncField is a persistent field object for an autoincrement field in a dataset.
	TBCDField (see page 6776)	TBCDField represents a binary-coded decimal (BCD) field in a dataset.
	TBinaryField (see page 6791)	TBinaryField represents an untyped binary field in a dataset.
	TBlobField (see page 6801)	TBlobField represents a field in a dataset that holds a reference to a binary large object (BLOB).
	TBooleanField (see page 6822)	TBooleanField represents a field containing Boolean values.
	TBytesField (see page 6833)	TBytesField represents a bytes field in a dataset.
	TCheckConstraint (see page 6841)	TCheckConstraint represents a record-level constraint on a dataset component.
	TCheckConstraints (see page 6846)	TCheckConstraints contains a collection of check constraints for a dataset component.
	TCurrencyField (see page 6850)	TCurrencyField represents a field that contains currency (see page 6857) values in a dataset.
	TCustomConnection (see page 6858)	TCustomConnection is the base class for components that connect datasets to a remote source of data.
	TDataLink (see page 6876)	TDataLink is a helper class used by data-aware objects to coordinate the actions of TDataSource (see page 7070) and TDataSet (see page 6894), and to respond to data events.
	TDataSet (see page 6894)	TDataSet is the base class for all dataset components that represent data in rows and columns.
	TDataSetDesigner (see page 7056)	TDataSetDesigner is the base class for all dataset designers.
	TDataSetField (see page 7060)	TDataSetField provides access to nested datasets.
	TDataSource (see page 7070)	TDataSource provides an interface between a dataset component and data-aware controls on a form.
	TDateField (see page 7083)	TDateField represents a date field in a dataset.
	TDateTimeField (see page 7090)	TDateTimeField represents a date-time field in a dataset.
	TDefCollection (see page 7104)	TDefCollection holds the definition objects (TNamedItem (see page 7357)) that represent the specifications for database entities such as fields or indexes.
	TDetailDataLink (see page 7111)	TDetailDataLink is the ancestor to datalink classes that manage the relationships between datasets that are linked in master/detail relationships.
	TField (see page 7115)	TField is the common ancestor of all the field components.
	TFieldDef (see page 7184)	TFieldDef is a field definition that corresponds to a physical field of a record in a table underlying a dataset.
	TFieldDefList (see page 7195)	TFieldDefList lists the field definition (TFieldDef (see page 7184)) objects of a dataset in a flattened form.
	TFieldDefs (see page 7198)	TFieldDefs holds the field definition (TFieldDef (see page 7184)) objects that represent the physical fields underlying a dataset.
	TFieldList (see page 7207)	TFieldList lists the fields of a dataset, flattening out the structure of any object fields.
	TFields (see page 7211)	TFields holds the field (TField (see page 7115)) objects that represent the physical fields underlying a dataset or an object field such as TADTField (see page 6736) and TArrayField (see page 6760).
	TFieldsEnumerator (see page 7222)	Enumerates database fields.
	TFlatList (see page 7225)	TFlatList is an abstract base class for objects that represent a flattened list of fields or field definitions.

	TFloatField (see page 7229)	TFloatField represents a field that contains floating-point values in a dataset.
	TFMTBCDField (see page 7242)	TFMTBCDField represents a binary-coded decimal (BCD) field in a dataset.
	TGraphicField (see page 7257)	TGraphicField represents a graphics field in a dataset.
	TGuidField (see page 7265)	TGuidField represents a guid field in a dataset.
	TIDispatchField (see page 7274)	TIDispatchField represents an IDispatch field in a dataset.
	TIndexDef (see page 7283)	TIndexDef describes an index in a database table.
	TIndexDefs (see page 7292)	TIndexDefs holds the list of available index definitions for a table.
	TIntegerField (see page 7301)	TIntegerField represents a signed 32-bit integer field in a dataset.
	TInterfaceField (see page 7314)	TInterfaceField represents an interface field in a dataset.
	TLargeintField (see page 7323)	A TLargeintField object represents a large integer field in a dataset.
	TLookupList (see page 7336)	TLookupList maintains a list of lookup values associated with a set of key field values.
	TMasterDataLink (see page 7339)	TMasterDataLink allows a dataset to establish a master/detail relationship.
	TMemoField (see page 7348)	TMemoField represents a memo field in a dataset.
	TNamedItem (see page 7357)	TNamedItem is the base class for definitions used by datasets.
	TNumericField (see page 7361)	TNumericField is the common ancestor of all the numeric fields in a dataset.
	TObjectField (see page 7372)	TObjectField is the common ancestor of TADTField (see page 6736), TArrayField (see page 6760), TDataSetField (see page 7060), and TReferenceField (see page 7452) components in a dataset.
	TParam (see page 7386)	TParam represents a field parameter for a query or stored procedure.
	TParams (see page 7434)	TParams manages a list of field parameters.
	TReferenceField (see page 7452)	TReferenceField represents a REF field in a dataset.
	TSmallintField (see page 7461)	TSmallIntField represents a signed 16-bit integer field in a dataset.
	TSQLTimeStampField (see page 7469)	TSQLTimeStampField represents a date-time field in a DBExpress dataset.
	TStringField (see page 7483)	TStringField represents a string field in a dataset.
	TTimeField (see page 7499)	TTimeField represents a time field in a dataset.
	TVarBytesField (see page 7507)	TVarBytesField represents a variable-length untyped binary field in a dataset.
	TVariantField (see page 7514)	TVariantField represents a variant field in a dataset.
	TWideDataSet (see page 7526)	Base class for all dataset components that represent Unicode data in rows and columns.
	TWideMemoField (see page 7541)	TWideMemoField represents a memo field in a dataset.
	TWideStringField (see page 7549)	TWideStringField represents a large string field in a dataset.
	TWordField (see page 7560)	TWordField represents an unsigned 16-bit integer field in a dataset.

39.1.1 EDatabaseError Class

EDatabaseError is the exception class for database errors.

Class Hierarchy

Pascal

```
EDatabaseError = class(Exception);
```

C++

```
class EDatabaseError : public Exception;
```

File

DB

Description

EDatabaseError is raised when a component detects a database error. Use EDatabaseError in an exception handling block or to create a database exception.

See Also

TDataSet (see page 6894)

DatabaseError

39.1.2 EUpdateError Class

EUpdateError is the exception class for dataset provider update errors.

Class Hierarchy

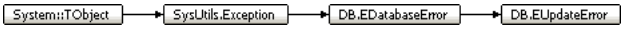

Pascal

```
EUpdateError = class(EDatabaseError);
```

C++

```
class EUpdateError : public EDatabaseError;
```

File

DB

Description

EUpdateError is raised if an error occurs when updates are applied by a provider component. raise this exception class for update errors when implementing the IProviderSupport (see page 7576) interface in a custom dataset.

See Also

TBaseProvider (see page 18423)

IProviderSupport (see page 7576)

39.1.2.1 EUpdateError Members

The following tables list the members exposed by EUpdateError.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6733)	Creates and initializes an instance of EUpdateError (see page 6732).
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.

	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 6734)	Destroys an instance of EUpdateError (see page 6732).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Context (see page 6734)	Contains context information from the error stack.
R	ErrorCode (see page 6735)	Indicates the error code that this exception represents.
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.
R	OriginalException (see page 6735)	Indicates the exception object that this update error represents.
R	PreviousError (see page 6736)	Indicates the previous error code for this update process.

39.1.2.2 EUpdateError.Create Constructor

Creates and initializes an instance of EUpdateError (see page 6732).

Pascal

```
constructor Create(NativeError: string; Context: string; ErrCode: Integer; PrevError: Integer;
E: Exception);
```

C++

```
__fastcall EUpdateError(AnsiString NativeError, AnsiString Context, int ErrCode, int
PrevError, Exception E);
```

Description

Create generates an EUpdateError (see page 6732) object to represent another error that is caught while applying updates.

NativeError specifies the error message for the exception.

Context (see page 6734) is a string providing additional context information. It becomes the value of the Context (see page

6734) property.

ErrCode and PrevError specify error codes for this exception or the previous exception in this update operation. If the current exception or previous exception does not have an error code, the corresponding argument is 0. The ErrCode and PrevError parameters initialize the ErrorCode (see page 6735) and PreviousError (see page 6736) properties.

Exception specifies the exception that was caught (which this EUpdateError (see page 6732) represents). It becomes the value of the OriginalException (see page 6735) property.

See Also

Context (see page 6734)

ErrorCode (see page 6735)

PreviousError (see page 6736)

EDBEngineError (see page 9854)

OriginalException (see page 6735)

39.1.2.3 EUpdateError.Destroy Destructor

Destroys an instance of EUpdateError (see page 6732).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~EUpdateError();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which checks that the EUpdateError (see page 6732) reference is not nil before calling Destroy.

See Also

Create (see page 6733)

Free (see page 22300)

39.1.2.4 EUpdateError Properties

The properties of the EUpdateError class are listed here.

Public Properties

	Name	Description
R	Context (see page 6734)	Contains context information from the error stack.
R	ErrorCode (see page 6735)	Indicates the error code that this exception represents.
R	OriginalException (see page 6735)	Indicates the exception object that this update error represents.
R	PreviousError (see page 6736)	Indicates the previous error code for this update process.

39.1.2.4.1 EUpdateError.Context Property

Contains context information from the error stack.

Pascal

```
property Context: string;
```

C++

```
__property AnsiString Context;
```

Description

Context contains additional information returned with the error message. If there is no additional context information, Context is an empty string.

See Also

ErrorCode (see page 6735)

39.1.2.4.2 EUpdateError.ErrorCode Property

Indicates the error code that this exception represents.

Pascal

```
property ErrorCode: Integer;
```

C++

```
__property int ErrorCode;
```

Description

EUpdateError (see page 6732) objects represent exceptions that occur when updating records. These exceptions can be generated in response to error codes from the data access mechanism (such as the BDE or ADO) or by other exceptions that are caught by resolver components. If the update error represents an error code returned by the data access mechanism, ErrorCode is that error code. Otherwise, ErrorCode is 0.

Note: DBExpress does not return error codes. This property merely provides a mechanism for using custom data-access components that generate error codes rather than exceptions.

Note: If ErrorCode is 0, use the OriginalException (see page 6735) property, which provides access to the exception that was caught.

See Also

OriginalException (see page 6735)

PreviousError (see page 6736)

39.1.2.4.3 EUpdateError.OriginalException Property

Indicates the exception object that this update error represents.

Pascal

```
property OriginalException: Exception;
```

C++

```
__property Exception OriginalException;
```

Description

EUpdateError (see page 6732) objects represent exceptions that occur when updating records. OriginalException is the exception object that is caught during the update process. Use this value to obtain more information about the problem that

occurred.

Note: If the EUpdateError (see page 6732) object represents an EDBEngineError, the ErrorCode (see page 6735) property provides immediate access to its BDE error code without going through the OriginalException property.

See Also

ErrorCode (see page 6735)

39.1.2.4.4 EUpdateError.PreviousError Property

Indicates the previous error code for this update process.

Pascal

```
property PreviousError: Integer;
```

C++

```
__property int PreviousError;
```

Description

PreviousError is the error code of the last update error in this update operation. If the last update error did not have an error code, PreviousError is 0.

See Also

ErrorCode (see page 6735)

39.1.3 TADTField Class

A TADTField object represents an ADT (Abstract Data Type) field in a dataset.

Class Hierarchy

Pascal

```
TADTField = class(TObjectField);
```

C++

```
class TADTField : public TObjectField;
```

File

DB

Description

ADT fields are database fields that contain Abstract Data Type (ADT) values. ADTs are user defined types created on the server, and are similar to records (Delphi) or structures (C++). An ADT can contain most scalar field types, including array fields, reference fields, nested dataset fields, or nested ADTs.

See Also

TField (see page 7115)

39.1.3.1 TADTField Members

The following tables list the members exposed by TADTField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CheckTypeSize (see page 7380)	This is CheckTypeSize, a member of class TObjectField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 7380)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	FreeBuffers (see page 7380)	This is FreeBuffers, a member of class TObjectField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
V	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
V	GetAsString (see page 7381)	This is GetAsString, a member of class TObjectField.
V	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
V	GetAsVariant (see page 7381)	This is GetAsVariant, a member of class TObjectField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
V	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
V	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
V	GetDefaultWidth (see page 7381)	This is GetDefaultWidth, a member of class TObjectField.

	GetFieldCount (see page 7381)	This is GetFieldCount, a member of class TObjectField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetFields (see page 7382)	This is GetFields, a member of class TObjectField.
	GetFieldValue (see page 7382)	This is GetFieldValue, a member of class TObjectField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetHasConstraints (see page 7382)	This is GetHasConstraints, a member of class TObjectField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 6743)	This is GetSize, a member of class TADTField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 7382)	This is SetChildOrder, a member of class TObjectField.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataSet (see page 7383)	This is SetDataSet, a member of class TObjectField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetFieldKind (see page 7383)	This is SetFieldKind, a member of class TObjectField.
	SetFieldValue (see page 7383)	This is SetFieldValue, a member of class TObjectField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetParentField (see page 7383)	This is SetParentField, a member of class TObjectField.

	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetUnNamed (see page 7383)	This is SetUnNamed, a member of class TObjectField.
	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
	SetVarValue (see page 7384)	This is SetVarValue, a member of class TObjectField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7126)	Sets the value of the field to NULL.
	Create (see page 6742)	Creates an instance of TADTField (see page 6736).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	Destroy (see page 7379)	Destroys an instance of TObjectField (see page 7372).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetChildren (see page 7381)	This is GetChildren, a member of class TObjectField.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.

	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldCount (see page 7384)	Indicates the number of child fields that comprise the object field.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	Fields (see page 7384)	Specifies the TFields (see page 7211) object that contains an array of child fields.
	FieldValues (see page 7385)	Provides access to the values for all child fields in the object field.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
R	UnNamed (see page 7386)	Indicates whether an object field in a nested table has a name associated with it.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7180)	Represents the data in a field component.

	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectType (see page 7385)	Specifies the classification of the field object in Oracle.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.3.2 TADTField.Create Constructor

Creates an instance of TADTField (see page 6736).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TADTField(TComponent * AOwner);
```

Description

Call Create to create and initialize an instance of TADTField (see page 6736). Create calls the inherited constructor and then sets the DataType (see page 7159) to ftADT.

See Also

Create (see page 7121)

39.1.3.3 TADTField Methods

The methods of the TADTField class are listed here.

Protected Methods

	Name	Description
	GetSize (see page 6743)	This is GetSize, a member of class TADTField.

39.1.3.3.1 TADTField.GetSize Method

Pascal

```
function GetSize: Integer; override;
```

C++

```
virtual __fastcall int GetSize();
```

Description

This is GetSize, a member of class TADTField.

39.1.4 TAggregateField Class

TAggregate field represents a maintained aggregate in a client dataset.

Class Hierarchy

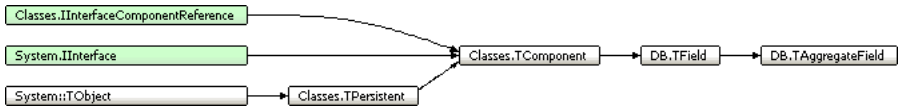

Pascal

```
TAggregateField = class(TField);
```

C++

```
class TAggregateField : public TField;
```

File

DB

Description

TAggregateField represents an aggregate value from a client dataset in a form that can be used by data-aware controls. Use the Fields Editor on a ClientDataSet to create a persistent aggregate field to which a TDBText control can bind.

As with TAggregate, TAggregateField has an Expression (see page 6756) property that represents a calculation summarizing

the data in a set of records. When using aggregate fields, you do not need to explicitly create the corresponding aggregate object in the client dataset: the Fields editor creates that object when you create a persistent aggregate field.

See Also

TAggregates (see page 8280)

TAggregate (see page 8268)

TCustomClientDataSet (see page 8373)

TDataSet (see page 6894)

TClientDataSet (see page 8289)

39.1.4.1 TAggregateField Members

The following tables list the members exposed by TAggregateField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
V	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsString (see page 6750)	This is GetAsString, a member of class TAggregateField.
V	GetAsVariant (see page 6750)	This is GetAsVariant, a member of class TAggregateField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.

	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
V	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
V	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
V	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
V	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetSize (see page 7135)	This is GetSize, a member of class TField.
V	GetText (see page 6751)	This is GetText, a member of class TAggregateField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 7138)	This is ReadState, a member of class TField.
	Reset (see page 6751)	This is Reset, a member of class TAggregateField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
V	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
V	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
V	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
V	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
V	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
V	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
V	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
V	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
V	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
V	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayFormat (see page 6751)	This is SetDisplayFormat, a member of class TAggregateField.
V	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
V	SetSize (see page 7143)	This is SetSize, a member of class TField.

	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7126)	Sets the value of the field to NULL.
	Create (see page 6749)	Creates an instance of TAggregateField (see page 6743).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).

	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
	Handle (see page 6758)	Points to the aggregate handle.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	ResultType (see page 6759)	Indicates the data type of the aggregate field.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7180)	Represents the data in a field component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Active (see page 6752)	Indicates whether the aggregate should be calculated.
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	currency (see page 6752)	Determines whether the value in the field should be formatted as a currency value.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayFormat (see page 6753)	Determines how numeric and time values are formatted for display in a data-aware control.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayName (see page 6756)	Represents the name of the field for display purposes.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	Expression (see page 6756)	Specifies the formula used to calculate the aggregate value.
	FieldKind (see page 6757)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
	GroupingLevel (see page 6758)	Specifies the set of records summarized by this aggregate field.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	IndexName (see page 6758)	Specifies the index that is used to define subgroups of records.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	Precision (see page 6759)	Determines the precision used in formatting floating-point values.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 6760)	Determines whether the field appears in a data grid.

39.1.4.2 TAggregateField.Create Constructor

Creates an instance of TAggregateField (see page 6743).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TAggregateField(TComponent * AOwner);
```

Description

Call Create to create and initialize an instance of TAggregateField (see page 6743) at runtime.

It is seldom necessary to call Create directly, because an aggregate field component is instantiated automatically when an aggregate field is added through the Fields Editor of a client dataset.

Create initializes DataType (see page 7159) to ftUnknown, FieldKind (see page 6757) to fkAggregate, Precision (see page 6759) to 15, and Visible (see page 6760) to false.

See Also

Create (see page 7121)

DataType (see page 7159)

FieldKind (see page 7164)

Precision (see page 6759)

Visible (see page 7181)

39.1.4.3 TAggregateField Methods

The methods of the TAggregateField class are listed here.

Protected Methods

	Name	Description
	GetAsString (see page 6750)	This is GetAsString, a member of class TAggregateField.
	GetAsVariant (see page 6750)	This is GetAsVariant, a member of class TAggregateField.
	GetText (see page 6751)	This is GetText, a member of class TAggregateField.
	Reset (see page 6751)	This is Reset, a member of class TAggregateField.
	SetDisplayFormat (see page 6751)	This is SetDisplayFormat, a member of class TAggregateField.

39.1.4.3.1 TAggregateField.GetAsString Method

Pascal

```
function GetAsString: string; override;
```

C++

```
virtual __fastcall AnsiString GetAsString();
```

Description

This is GetAsString, a member of class TAggregateField.

39.1.4.3.2 TAggregateField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TAggregateField.

39.1.4.3.3 TAggregateField.GetText Method

Pascal

```
procedure GetText(var Text: string; DisplayText: Boolean); override;
```

C++

```
virtual __fastcall GetText(AnsiString Text, Boolean DisplayText);
```

Description

This is GetText, a member of class TAggregateField.

39.1.4.3.4 TAggregateField.Reset Method

Pascal

```
procedure Reset;
```

C++

```
__fastcall Reset();
```

Description

This is Reset, a member of class TAggregateField.

39.1.4.3.5 TAggregateField.SetDisplayFormat Method

Pascal

```
procedure SetDisplayFormat(const Value: string);
```

C++

```
__fastcall SetDisplayFormat(const AnsiString Value);
```

Description

This is SetDisplayFormat, a member of class TAggregateField.

39.1.4.4 TAggregateField Properties

The properties of the TAggregateField class are listed here.

Public Properties

	Name	Description
	Handle (see page 6758)	Points to the aggregate handle.
	ResultType (see page 6759)	Indicates the data type of the aggregate field.

Published Properties

	Name	Description
	Active (see page 6752)	Indicates whether the aggregate should be calculated.
	currency (see page 6752)	Determines whether the value in the field should be formatted as a currency value.
	DisplayFormat (see page 6753)	Determines how numeric and time values are formatted for display in a data-aware control.
	DisplayName (see page 6756)	Represents the name of the field for display purposes.
	Expression (see page 6756)	Specifies the formula used to calculate the aggregate value.

	FieldKind (see page 6757)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	GroupingLevel (see page 6758)	Specifies the set of records summarized by this aggregate field.
	IndexName (see page 6758)	Specifies the index that is used to define subgroups of records.
	Precision (see page 6759)	Determines the precision used in formatting floating-point values.
	Visible (see page 6760)	Determines whether the field appears in a data grid.

39.1.4.4.1 TAggregateField.Active Property

Indicates whether the aggregate should be calculated.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Use the Active property to indicate that the aggregate field's value should be maintained when possible. If Active is false, the value of the aggregate is never calculated.

If Active is true, the aggregate field's value is calculated unless

It summarizes records over a group not defined by the current index.

The client dataset's AggregatesActive property is false.

Note: To determine whether the aggregate that corresponds to the field is currently maintained, read its InUse property.

See Also

AggregatesActive (see page 8461)

InUse (see page 8278)

39.1.4.4.2 TAggregateField.currency Property

Determines whether the value in the field should be formatted as a currency value.

Pascal

```
property currency: Boolean;
```

C++

```
__property Boolean currency;
```

Description

Use currency to specify the formatting of the field if its ResultType (see page 6759) is a floating-point type (ftFloat or ftCurrency).

Floating-point values are formatted for display using the DisplayFormat (see page 6753) property if it is assigned. If DisplayFormat (see page 6753) is not assigned, currency determines how the field is formatted for display. If currency is true, the value is formatted for display using the FloatToText function with the ffCurrency formatting code. If currency is false, the value is formatted with the ffGeneral format.

See Also

DisplayFormat (see page 6753)

DisplayText (see page 7162)

FloatToText (see page 22892)

FloatToTextFmt (see page 22893)

Text (see page 7179)

ResultType (see page 6759)

39.1.4.4.3 TAggregateField.DisplayFormat Property

Determines how numeric and time values are formatted for display in a data-aware control.

Pascal

```
property DisplayFormat: string;
```

C++

```
__property AnsiString DisplayFormat;
```

Description

Use the DisplayFormat property to override automatic formatting of a field for display purposes when ResultType (see page 6759) is ftFloat, ftCurrency, ftDate, ftTime, or ftDateTime.

When ResultType (see page 6759) is ftFloat or ftCurrency, the value of DisplayFormat is a string that encodes the formatting of data using the specifiers in the following table:

Specifier	Represents
0	Digit placeholder. If the value being formatted has a digit in the position where the "0" appears in the format string, then that digit is copied to the output string. Otherwise, a "0" is stored in that position in the output string.
#	Digit placeholder. If the value being formatted has a digit in the position where the "#" appears in the format string, then that digit is copied to the output string. Otherwise, nothing is stored in that position in the output string.
.	Decimal point. The first "." character in the format string determines the location of the decimal separator in the formatted value; any additional "." characters are ignored. The actual character used as a the decimal separator in the output string is determined by the DecimalSeparator global variable. The default value of DecimalSeparator is specified in the Number Format of the Regional Settings section in the Windows Control Panel.
,	Thousand separator. If the format string contains one or more "," characters, the output will have thousand separators inserted between each group of three digits to the left of the decimal point. The placement and number of "," characters in the format string does not affect the output, except to indicate that thousand separators are wanted. The actual character used as a the thousand separator in the output is determined by the ThousandSeparator global variable. The default value of ThousandSeparator is specified in the Number Format of the Regional Settings section in the Windows Control Panel.
E+	Scientific notation. If any of the strings "E+", "E-", "e+", or "e-" are contained in the format string, the number is formatted using scientific notation. A group of up to four "0" characters can immediately follow the "E+", "E-", "e+", or "e-" to determine the minimum number of digits in the exponent. The "E+" and "e+" formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The "E-" and "e-" formats output a sign character only for negative exponents.
'xx'/"xx"	Characters enclosed in single or double quotes are output as-is, and do not affect formatting.
;	Separates sections for positive, negative, and zero numbers in the format string.
c	The date using the format given by the ShortDateFormat global variable, followed by the time using the format given by the LongTimeFormat global variable. The time is not displayed if the fractional part of the DateTime value is zero.
d	The day as a number without a leading zero (1-31).
dd	The day as a number with a leading zero (01-31).
ddd	The day as an abbreviation (Sun-Sat) using the strings given by the ShortDayNames global variable.
dddd	The day as a full name (Sunday-Saturday) using the strings given by the LongDayNames global variable.

ddddd	The date using the format given by the ShortDateFormat global variable.
dddddd	The date using the format given by the LongDateFormat global variable.
m	The month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mm	The month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mmm	The month as an abbreviation (Jan-Dec) using the strings given by the ShortMonthNames global variable.
mmmm	The month as a full name (January-December) using the strings given by the LongMonthNames global variable.
yy	The year as a two-digit number (00-99).
yyyy	The year as a four-digit number (0000-9999).
h	The hour without a leading zero (0-23).
hh	The hour with a leading zero (00-23).
n	The minute without a leading zero (0-59).
nn	The minute with a leading zero (00-59).
s	The second without a leading zero (0-59).
ss	The second with a leading zero (00-59).
t	The time using the format given by the ShortTimeFormat global variable.
tt	The time using the format given by the LongTimeFormat global variable.
am/pm	The time using the 12-hour clock for the preceding h or hh specifier, followed by "am" for any hour before noon, or "pm" for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	The time using the 12-hour clock for the preceding h or hh specifier, followed by "a" for any hour before noon, or "p" for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm	The time using the 12-hour clock for the preceding h or hh specifier, followed by the contents of the TimeAMString global variable for any hour before noon, or the contents of the TimePMString global variable for any hour after noon.
/	The date separator character given by the DateSeparator global variable.
:	The time separator character given by the TimeSeparator global variable.
'xx'/"xx"	Characters enclosed in single or double quotes are displayed as-is, with no formatting changes.

The locations of the leftmost "0" before the decimal point in the format string and the rightmost "0" after the decimal point in the format string determine the range of digits that are always present in the output string.

The number being formatted is always rounded to as many decimal places as there are digit placeholders ("0" or "#") to the right of the decimal point. If the format string contains no decimal point, the value being formatted is rounded to the nearest whole number.

If the number being formatted has more digits to the left of the decimal separator than there are digit placeholders to the left of the "." character in the format string, the extra digits are output before the first digit placeholder.

To allow different formats for positive, negative, and zero values, the format string can contain between one and three sections separated by semicolons.

One section: The format string applies to all values.

Two sections: The first section applies to positive values and zeros, and the second section applies to negative values.

Three sections: The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If the section for negative values or the section for zero values is empty, that is, if there is nothing between the semicolons that delimit the section, the section for positive values is used instead.

If the section for positive values is empty, or if the entire format string is empty, the value is formatted using general floating-point formatting with 15 significant digits. General floating-point formatting is also used if the value has more than 18 digits to the left of the decimal point and the format string does not specify scientific notation.

When ResultType (see page 6759) is ftDate, ftTime, or ftDateTime, the value of DisplayFormat is a string that encodes the formatting of data using the specifiers in the following table:

Specifier	Displays
c	The date using the format given by the ShortDateFormat global variable, followed by the time using the format given by the LongTimeFormat global variable. The time is not displayed if the fractional part of the DateTime value is zero.
d	The day as a number without a leading zero (1-31).
dd	The day as a number with a leading zero (01-31).
ddd	The day as an abbreviation (Sun-Sat) using the strings given by the ShortDayNames global variable.
dddd	The day as a full name (Sunday-Saturday) using the strings given by the LongDayNames global variable.
ddddd	The date using the format given by the ShortDateFormat global variable.
dddddd	The date using the format given by the LongDateFormat global variable.
m	The month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mm	The month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mmm	The month as an abbreviation (Jan-Dec) using the strings given by the ShortMonthNames global variable.
mmmm	The month as a full name (January-December) using the strings given by the LongMonthNames global variable.
yy	The year as a two-digit number (00-99).
yyyy	The year as a four-digit number (0000-9999).
h	The hour without a leading zero (0-23).
hh	The hour with a leading zero (00-23).
n	The minute without a leading zero (0-59).
nn	The minute with a leading zero (00-59).
s	The second without a leading zero (0-59).
ss	The second with a leading zero (00-59).
t	The time using the format given by the ShortTimeFormat global variable.
tt	The time using the format given by the LongTimeFormat global variable.
am/pm	The time using the 12-hour clock for the preceding h or hh specifier, followed by "am" for any hour before noon, or "pm" for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	The time using the 12-hour clock for the preceding h or hh specifier, followed by "a" for any hour before noon, or "p" for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm	The time using the 12-hour clock for the preceding h or hh specifier, followed by the contents of the TimeAMString global variable for any hour before noon, or the contents of the TimePMString global variable for any hour after noon.

/	The date separator character given by the DateSeparator global variable.
:	The time separator character given by the TimeSeparator global variable.
'xx'/"xx"	Characters enclosed in single or double quotes are displayed as-is, with no formatting changes.

See Also

AsString (see page 7151)

DisplayText (see page 7162)

OnGetText (see page 7182)

FormatFloat (see page 22904)

ResultType (see page 6759)

39.1.4.4.4 TAggregateField.DisplayName Property

Represents the name of the field for display purposes.

Pascal

```
property DisplayName: WideString;
```

C++

```
__property BSTR DisplayName;
```

Description

Set DisplayName to give the aggregate a name that can be used to label values displayed to the user.

Note: When you create an aggregate field in the Fields editor, DisplayName is the same as the AggregateName property of the corresponding TAggregate object. However, if you change DisplayName, the aggregate's AggregateName is not updated as well.

See Also

AggregateName (see page 8275)

39.1.4.4.5 TAggregateField.Expression Property

Specifies the formula used to calculate the aggregate value.

Pascal

```
property Expression: String;
```

C++

```
__property AnsiString Expression;
```

Description

Set Expression to indicate how aggregate values should be calculated. Expression contains an expression that includes one or more of the summary operators in the following table:

Operator	Use
Sum	Totals the values for a numeric field or expression.
Avg	Computes the average value for a numeric or date-time field or expression.

Count	Specifies the number of non-blank values for a field or expression. Use count(*) to count the number of records in a dataset or subgroup.
Min	Indicates the minimum value for a string, numeric, or date-time field or expression.
Max	Indicates the maximum value for a string, numeric, or date-time field or expression.
Sum(Qty * Price)	{legal -- summary of an expression on fields }
Max(Field1) - Max(Field2)	{legal -- expression on summaries }
Avg(DiscountRate) * 100	{ legal -- expression of summary and constant }
Min(Sum(Field1))	{ illegal -- nested summaries }
Count(Field1) - Field2	{ illegal -- expression of summary and field }

The summary operators act on field values or on expressions built from field values using the same operators you use to create filters. (You can't nest summary operators, however.) You can create expressions by using operators on summarized values with other summarized values, or on summarized values and constants. However, you can't combine summarized values with field values, because such expressions are ambiguous (there is no indication of which record should supply the field value.) These rules are illustrated in the following expressions:

Note: To specify the set of records summarized by this expression, use the GroupingLevel (see page 6758) property.

See Also

GroupingLevel (see page 6758)

Filter (see page 8476)

39.1.4.4.6 TAggregateField.FieldKind Property

Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.

Pascal

```
property FieldKind: TFieldKind;
```

C++

```
__property TFieldKind FieldKind;
```

Description

Use FieldKind to determine if a field is a data field, a calculated field, a lookup field, or an aggregate field. The value of FieldKind can be changed programmatically, but in practice FieldKind is set at design time when creating Field components with the Fields editor.

Note: Fields calculated by SQL servers or the Borland Database Engine to display the results of a query that returns a live dataset

have a FieldKind of fkInternalCalc, not fkCalculated. This is because the field values are stored in the dataset. Calculated fields in a client dataset that are calculated in an OnCalcFields event handler but stored in the dataset also have a FieldKind of fkInternalCalc instead of fkCalculated. Unlike regular calculated fields, internally calculated fields can be used in filter expressions. They can be edited, but the changes are discarded. To prevent editing, set the ReadOnly property to true.

Note: The FieldKind property of a field is only stored with the field object if the value is fkInternalCalc.

See Also

Calculated (see page 7154)

Lookup (see page 7170)

ReadOnly (see page 7177)

39.1.4.4.7 TAggregateField.GroupingLevel Property

Specifies the set of records summarized by this aggregate field.

Pascal

```
property GroupingLevel: Integer;
```

C++

```
__property int GroupingLevel;
```

Description

Use GroupingLevel to indicate the groups of records over which the field's aggregate is calculated. When GroupingLevel is 0 (the default), the aggregate summarizes all the records in the client dataset. When GroupingLevel is greater than 0, the aggregate summarizes each group of records that has the same values on the first GroupingLevel fields of the index.

GroupingLevel can be any value from 0 to the number of fields in the index specified by IndexName (see page 6758).

See Also

TAggregate (see page 8268)

IndexName (see page 6758)

Expression (see page 6756)

39.1.4.4.8 TAggregateField.Handle Property

Points to the aggregate handle.

Pascal

```
property Handle: Pointer;
```

C++

```
__property void * Handle;
```

Description

TAggregateField (see page 6743) uses Handle internally to obtain its aggregated values. Most applications do not need to use this value.

Component writers can use Handle, if necessary, with the routines declared in the dsintf unit.

See Also

AggHandle (see page 8274)

39.1.4.4.9 TAggregateField.IndexName Property

Specifies the index that is used to define subgroups of records.

Pascal

```
property IndexName: String;
```

C++

```
__property AnsiString IndexName;
```

Description

Set IndexName to specify an index that sorts the records of the client dataset into subgroups. Each subgroup is defined as a set of records that all have the same values on a set of fields. This defining set of fields is the first GroupingLevel (see page 6758) fields of the index.

IndexName is required for any aggregate that does not summarize all the records in the client dataset (GroupingLevel (see page 6758) = 0). When the index associated with an aggregate is not the current index of the client dataset, its value is unavailable.

See Also

DataSet (see page 7157)

GroupingLevel (see page 6758)

IndexName (see page 8483)

InUse (see page 8278)

39.1.4.4.10 TAggregateField.Precision Property

Determines the precision used in formatting floating-point values.

Pascal

```
property Precision: Integer;
```

C++

```
__property int Precision;
```

Description

When the aggregate field represents a floating-point value (ResultType (see page 6759) is ftFloat or ftCurrency), use Precision to set the number of digits beyond the decimal point that appear before rounding begins. Precision affects the format of the floating-point value when it is displayed as a text string, not the actual value that is stored in the field.

The default value of Precision is 15 decimal places.

Note: Precision must be greater than or equal to 2. Trying to set Precision to a value less than 2 changes the value of Precision to 2.

See Also

DisplayFormat (see page 6753)

ResultType (see page 6759)

39.1.4.4.11 TAggregateField.ResultType Property

Indicates the data type of the aggregate field.

Pascal

```
property ResultType: TFieldType;
```

C++

```
__property TFieldType ResultType;
```

Description

Read ResultType to determine the type of data that the aggregate produces.

See Also

Expression (see page 6756)

39.1.4.4.12 TAggregateField.Visible Property

Determines whether the field appears in a data grid.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

To hide a field in a data grid, set Visible to false. If Visible is true, the field appears in the grid.

Note: The Visible property has no effect if the grid has an explicit Columns value for the field. It also has no effect on controls other than TDBGrid, such as TDBEdit.

See Also

Columns (see page 9653)

39.1.5 TArrayField Class

TArrayField represents an Array field in a dataset.

Class Hierarchy

Pascal

```
TArrayField = class(TObjectField);
```

C++

```
class TArrayField : public TObjectField;
```

File

DB

Description

Array fields consist of a set of subfields of the same type. The field type of all subfields can be scalar (for example, float, string), or non-scalar (an ADT), but an array field of arrays is not permitted. The SparseArrays property of TDataSet (see page 6894) determines whether a unique TField (see page 7115) object is created for each element of the array field.

See Also

TField (see page 7115)

39.1.5.1 TArrayField Members

The following tables list the members exposed by TArrayField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 6767)	This is Bind, a member of class TArrayField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CheckTypeSize (see page 7380)	This is CheckTypeSize, a member of class TObjectField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 7380)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	FreeBuffers (see page 7380)	This is FreeBuffers, a member of class TObjectField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
V	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
V	GetAsString (see page 7381)	This is GetAsString, a member of class TObjectField.
V	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
V	GetAsVariant (see page 7381)	This is GetAsVariant, a member of class TObjectField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
V	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
V	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
V	GetDefaultWidth (see page 7381)	This is GetDefaultWidth, a member of class TObjectField.

	GetFieldCount (see page 7381)	This is GetFieldCount, a member of class TObjectField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetFields (see page 7382)	This is GetFields, a member of class TObjectField.
	GetFieldValue (see page 7382)	This is GetFieldValue, a member of class TObjectField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetHasConstraints (see page 7382)	This is GetHasConstraints, a member of class TObjectField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 7382)	This is SetChildOrder, a member of class TObjectField.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataSet (see page 7383)	This is SetDataSet, a member of class TObjectField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetFieldKind (see page 7383)	This is SetFieldKind, a member of class TObjectField.
	SetFieldValue (see page 7383)	This is SetFieldValue, a member of class TObjectField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetParentField (see page 7383)	This is SetParentField, a member of class TObjectField.

	SetSize (see page 6767)	This is SetSize, a member of class TArrayField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetUnNamed (see page 7383)	This is SetUnNamed, a member of class TObjectField.
	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
	SetVarValue (see page 7384)	This is SetVarValue, a member of class TObjectField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7126)	Sets the value of the field to NULL.
	Create (see page 6766)	Creates an instance of TArrayField (see page 6760).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	Destroy (see page 7379)	Destroys an instance of TObjectField (see page 7372).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetChildren (see page 7381)	This is GetChildren, a member of class TObjectField.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.

	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldCount (see page 7384)	Indicates the number of child fields that comprise the object field.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	Fields (see page 7384)	Specifies the TFields (see page 7211) object that contains an array of child fields.
	FieldValues (see page 7385)	Provides access to the values for all child fields in the object field.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 6768)	The number of elements in the array field.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
R	UnNamed (see page 7386)	Indicates whether an object field in a nested table has a name associated with it.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7180)	Represents the data in a field component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectType (see page 7385)	Specifies the classification of the field object in Oracle.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.5.2 TArrayField.Create Constructor

Creates an instance of TArrayField (see page 6760).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TArrayField(TComponent * AOwner);
```

Description

Most applications do not need to explicitly create TArrayField (see page 6760) instances. These are created automatically by the dataset.

Create calls the inherited TField (see page 7115) constructor, sets the DataType (see page 7159) to ftArray, and sets the Size (see page 6768) to 10. The AOwner parameter specifies the owner of the field (typically, the dataset). The owner is responsible for freeing the field component.

See Also

Create (see page 7121)

Owner (see page 2807)

39.1.5.3 TArrayField Methods

The methods of the TArrayField class are listed here.

Protected Methods

	Name	Description
	Bind (see page 6767)	This is Bind, a member of class TArrayField.
	SetSize (see page 6767)	This is SetSize, a member of class TArrayField.

39.1.5.3.1 TArrayField.Bind Method

Pascal

```
procedure Bind(Binding: Boolean); override;
```

C++

```
virtual __fastcall Bind(Boolean Binding);
```

Description

This is Bind, a member of class TArrayField.

39.1.5.3.2 TArrayField.SetSize Method

Pascal

```
procedure SetSize(Value: Integer); override;
```

C++

```
virtual __fastcall SetSize(int Value);
```

Description

This is SetSize, a member of class TArrayField.

39.1.5.4 TArrayField Properties

The properties of the TArrayField class are listed here.

Public Properties

	Name	Description
	Size (see page 6768)	The number of elements in the array field.

39.1.5.4.1 TArrayField.Size Property

The number of elements in the array field.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

Use Size property to get or set the number of elements in the array field.

39.1.6 TAutoIncField Class

TAutoIncField is a persistent field object for an autoincrement field in a dataset.

Class Hierarchy

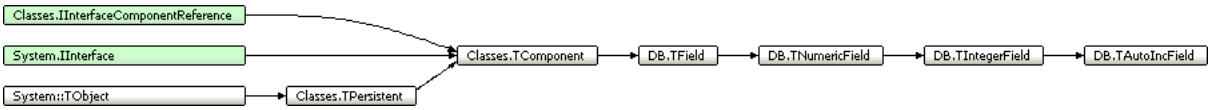

Pascal

```
TAutoIncField = class(TIntegerField);
```

C++

```
class TAutoIncField : public TIntegerField;
```

File

DB

Description

TAutoIncField provides access to an autoincrement field in a table. Autoincrement fields can hold values in the range -2,147,483,648 to 2,147,483,647, and are used as a counter for the records in the dataset. Each record is given a unique value in an autoincrement field, with each record receiving the next highest integer from that of the previously inserted record. Autoincrement fields are commonly used to provide a unique primary key value for a record.

Note: Auto-incrementing fields are a function of the database back-end. An application should not attempt to set a field value for an autoincrement field (with or without a TAutoIncField object).

If you use the Fields editor at design time to create a persistent field component for the auto-incrementing field, you can access it by name at runtime. When using dynamic field components, you can access the TAutoIncField instance using the dataset's Fields property or FieldByName method.

See Also

TDataSet (see page 6894)

TField (see page 7115)

TField (see page 7115)

TDataSet (see page 6894)

TDataSet (see page 6894)

TIntegerField (see page 7301)

39.1.6.1 TAutoIncField Members

The following tables list the members exposed by TAutoIncField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
V	GetAsFloat (see page 7310)	This is GetAsFloat, a member of class TIntegerField.
V	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
V	GetAsInteger (see page 7310)	This is GetAsInteger, a member of class TIntegerField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
V	GetAsString (see page 7311)	This is GetAsString, a member of class TIntegerField.
V	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
V	GetAsVariant (see page 7311)	This is GetAsVariant, a member of class TIntegerField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.

	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
V	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
V	GetDataSize (see page 7311)	This is GetDataSize, a member of class TIntegerField.
V	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
V	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
V	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetSize (see page 7135)	This is GetSize, a member of class TField.
V	GetText (see page 7135)	This is GetText, a member of class TField.
V	GetText (see page 7311)	This is GetText, a member of class TIntegerField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 7312)	This is GetValue, a member of class TIntegerField.
V	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	RangeError (see page 7368)	This is RangeError, a member of class TNumericField.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
V	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
V	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
V	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
V	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
V	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
V	SetAsFloat (see page 7312)	This is SetAsFloat, a member of class TIntegerField.
V	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
V	SetAsInteger (see page 7312)	This is SetAsInteger, a member of class TIntegerField.
V	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
V	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
V	SetAsString (see page 7312)	This is SetAsString, a member of class TIntegerField.
V	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
V	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayFormat (see page 7368)	This is SetDisplayFormat, a member of class TNumericField.
	SetEditFormat (see page 7368)	This is SetEditFormat, a member of class TNumericField.

	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
	SetVarValue (see page 7312)	This is SetVarValue, a member of class TIntegerField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7126)	Sets the value of the field to NULL.
	Create (see page 6775)	Creates an instance of TAutoIncField (see page 6768).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.

	Value (see page 7180)	Represents the data in a field component.
	Value (see page 7314)	Represents the value of the integer field.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	Alignment (see page 7369)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayFormat (see page 7369)	Determines how a numeric field's value is formatted for display in a data-aware control.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	EditFormat (see page 7371)	Determines how a numeric field's value is formatted when it is being edited in a data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	MaxValue (see page 7313)	Limits the maximum value in the integer field.
	MinValue (see page 7313)	Limits the minimum value in the integer field.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.6.2 TAutoIncField.Create Constructor

Creates an instance of TAutoIncField (see page 6768).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TAutoIncField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TAutoIncField (see page 6768). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

After calling the inherited constructor, Create sets the DataType (see page 7159) property to ftAutoInc. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TAutoIncField (see page 6768). After instantiating a TAutoIncField (see page 6768), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TAutoIncField (see page 6768) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TAutoIncField (see page 6768) with a dataset component by setting its DataSet (see page 7157) property to the name of the dataset component.

The example below creates a TAutoIncField (see page 6768) object for a field named ID accessed through a TSQLQuery named SQLQuery1.

```
var
  T: TAutoIncField;
begin
  SQLQuery1.Close;
  T := TAutoIncField.Create(SQLQuery1);
  T.FieldName := 'ID';
  T.Name := SQLQuery1.Name + T.FieldName;
  T.Index := SQLQuery1.FieldCount;
  T.DataSet := SQLQuery1;
  SQLQuery1.FieldDefs.UpDate;
  SQLQuery1.Open;
end;
TAutoIncField *T;
SQLQuery1->Close();
T = new TAutoIncField(SQLQuery1);
T->FieldName = "ID";
T->Name = SQLQuery1->Name + T->FieldName;
T->Index = SQLQuery1->FieldCount;
T->DataSet = SQLQuery1;
SQLQuery1->FieldDefs->UpDate();
SQLQuery1->Open();
```

See Also

DataType (see page 7159)

Create (see page 7309)

Owner (see page 2807)

39.1.7 TBCDField Class

TBCDField represents a binary-coded decimal (BCD) field in a dataset.

Class Hierarchy

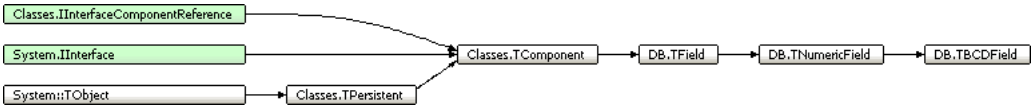

Pascal

```
TBCDField = class(TNumericField);
```

C++

```
class TBCDField : public TNumericField;
```

File

DB

Description

TBCDField encapsulates the fundamental behavior common to binary-coded decimal (BCD) fields. BCD values provide greater precision and accuracy than floating-point numbers. BCD fields are often used for storing and manipulating monetary values.

The IDE uses two different field types for representing BCD fields: TBCDField and TFMTBCDField (see page 7242). TBCDField uses the Currency (Delphi) or System::Currency (see page 22145) (C++) type to manipulate BCD values. This is faster than storing and manipulating the value using a true BCD type, but limits the precision of the BCD values it can support to 4 decimal places and 20 significant digits.

TBCDField converts the data from a BCD value to a Currency value when it fetches the data from the database table, and converts it from a Currency value to a binary-coded decimal value when it posts the data. If the underlying database table contains a value that requires greater precision, TBCDField raises an exception. If your application requires BCD values with more than 4 decimal places or 20 significant digits, you should use TFMTBCDField (see page 7242) instead. TFMTBCDField (see page 7242) is a true BCD, with the precision of the binary-coded decimal type (TBCD) but with somewhat slower performance.

If you use the Fields editor at design time to create a persistent field component for the BCD field, you can access it by name at runtime. When using dynamic field components, you can access the TBCDField instance using the dataset's Fields property or FieldByName method.

See Also

TDataSet (see page 6894)

TDataSet (see page 6894)

TDataSet (see page 6894)

TFloatField (see page 7229)

TFMTBCDField (see page 7242)

TBcd

Using Default Formatting for Numeric

39.1.7.1 TBCDField Members

The following tables list the members exposed by TBCDField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Bind (see page 7126)	This is Bind, a member of class TField.
	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
	CheckTypeSize (see page 6784)	This is CheckTypeSize, a member of class TBCDField.
	CopyData (see page 6784)	This is CopyData, a member of class TBCDField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
	GetAsBCD (see page 6784)	This is GetAsBCD, a member of class TBCDField.
	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
	GetAsCurrency (see page 6784)	This is GetAsCurrency, a member of class TBCDField.
	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
	GetAsFloat (see page 6785)	This is GetAsFloat, a member of class TBCDField.
	GetAsInteger (see page 6785)	This is GetAsInteger, a member of class TBCDField.
	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
	GetAsString (see page 6785)	This is GetAsString, a member of class TBCDField.
	GetAsVariant (see page 6785)	This is GetAsVariant, a member of class TBCDField.
	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
	GetDataSize (see page 6785)	This is GetDataSize, a member of class TBCDField.
	GetDefaultWidth (see page 6786)	This is GetDefaultWidth, a member of class TBCDField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.

	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 6786)	This is GetText, a member of class TBCDField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 6786)	This is GetValue, a member of class TBCDField.
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	RangeError (see page 7368)	This is RangeError, a member of class TNumericField.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 6786)	This is SetAsBCD, a member of class TBCDField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 6787)	This is SetAsCurrency, a member of class TBCDField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 6787)	This is SetAsFloat, a member of class TBCDField.
	SetAsInteger (see page 6787)	This is SetAsInteger, a member of class TBCDField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 6787)	This is SetAsString, a member of class TBCDField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayFormat (see page 7368)	This is SetDisplayFormat, a member of class TNumericField.
	SetEditFormat (see page 7368)	This is SetEditFormat, a member of class TNumericField.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 6787)	This is SetVarValue, a member of class TBCDField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.

	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 7126)	Sets the value of the field to NULL.
V	Create (see page 6782)	Create creates an instance of TBCDField (see page 6776).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.

	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.

R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 6790)	Represents the value of the BCD field as a Currency value.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	Alignment (see page 7369)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	currency (see page 6788)	This is currency, a member of class TBCDField.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.

	DisplayFormat (see page 7369)	Determines how a numeric field's value is formatted for display in a data-aware control.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	EditFormat (see page 7371)	Determines how a numeric field's value is formatted when it is being edited in a data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	MaxValue (see page 6788)	Limits the maximum value in the binary-coded decimal field.
	MinValue (see page 6789)	Limits the minimum value in the binary-coded decimal field.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	Precision (see page 6789)	Specifies the number of digits of accuracy used to store the field value.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Size (see page 6789)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.7.2 TBCDField.Create Constructor

Create creates an instance of TBCDField (see page 6776).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBCDField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TBCDField (see page 6776). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

After calling the inherited constructor, Create sets the DataType (see page 7159) property to ftBCD and initializes Size (see page 6789) to 4. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TBCDField (see page 6776). After instantiating a TBCDField (see page 6776), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TBCDField (see page 6776) a unique identifier in the

Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TBCDField (see page 6776) with a dataset component by setting its DataSet (see page 7157) property to the name of the dataset component. Specify the precision of the TBCDField (see page 6776) by setting the Precision (see page 6789) property.

The example below creates a TBCDField (see page 6776) object for a field named Amount accessed through a TSQLQuery named SQLQuery1.

```
var
  T: TBCDField;
begin
  SQLQuery1.Close;
  T := TBCDField.Create(SQLQuery1);
  T.FieldName := 'Amount';
  T.Name := SQLQuery1.Name + T.FieldName;
  T.Index := SQLQuery1.FieldCount;
  T.DataSet := SQLQuery1;
  T.Precision := 2;
  SQLQuery1.FieldDefs.UpDate;
  SQLQuery1.Open;
end;
SQLQuery1->Close();
TBCDField *T = new TBCDField(SQLQuery1);
T->FieldName = "Amount";
T->Name = SQLQuery1->Name + T->FieldName;
T->Index = SQLQuery1->FieldCount;
T->DataSet = SQLQuery1;
SQLQuery1->FieldDefs->UpDate();
SQLQuery1->Open();
```

See Also

Destroy (see page 7122)

DataType (see page 7159)

Size (see page 7178)

Create (see page 7367)

Owner (see page 2807)

39.1.7.3 TBCDField Methods

The methods of the TBCDField class are listed here.

Protected Methods

	Name	Description
	CheckTypeSize (see page 6784)	This is CheckTypeSize, a member of class TBCDField.
	CopyData (see page 6784)	This is CopyData, a member of class TBCDField.
	GetAsBCD (see page 6784)	This is GetAsBCD, a member of class TBCDField.
	GetAsCurrency (see page 6784)	This is GetAsCurrency, a member of class TBCDField.
	GetAsFloat (see page 6785)	This is GetAsFloat, a member of class TBCDField.
	GetAsInteger (see page 6785)	This is GetAsInteger, a member of class TBCDField.
	GetAsString (see page 6785)	This is GetAsString, a member of class TBCDField.
	GetAsVariant (see page 6785)	This is GetAsVariant, a member of class TBCDField.
	GetDataSize (see page 6785)	This is GetDataSize, a member of class TBCDField.
	GetDefaultWidth (see page 6786)	This is GetDefaultWidth, a member of class TBCDField.
	GetText (see page 6786)	This is GetText, a member of class TBCDField.
	GetValue (see page 6786)	This is GetValue, a member of class TBCDField.

	SetAsBCD (see page 6786)	This is SetAsBCD, a member of class TBCDField.
	SetAsCurrency (see page 6787)	This is SetAsCurrency, a member of class TBCDField.
	SetAsFloat (see page 6787)	This is SetAsFloat, a member of class TBCDField.
	SetAsInteger (see page 6787)	This is SetAsInteger, a member of class TBCDField.
	SetAsString (see page 6787)	This is SetAsString, a member of class TBCDField.
	SetVarValue (see page 6787)	This is SetVarValue, a member of class TBCDField.

39.1.7.3.1 TBCDField.CheckTypeSize Method

Pascal

```
class procedure CheckTypeSize(Value: Integer); override;
```

C++

```
virtual __fastcall CheckTypeSize(int Value);
```

Description

This is CheckTypeSize, a member of class TBCDField.

39.1.7.3.2 TBCDField.CopyData Method

Pascal

```
procedure CopyData(Source: Pointer; Dest: Pointer); override;
```

C++

```
virtual __fastcall CopyData(void * Source, void * Dest);
```

Description

This is CopyData, a member of class TBCDField.

39.1.7.3.3 TBCDField.GetAsBCD Method

Pascal

```
function GetAsBCD: TBcd; override;
```

C++

```
virtual __fastcall TBcd GetAsBCD();
```

Description

This is GetAsBCD, a member of class TBCDField.

39.1.7.3.4 TBCDField.GetAsCurrency Method

Pascal

```
function GetAsCurrency: Currency; override;
```

C++

```
virtual __fastcall Currency GetAsCurrency();
```

Description

This is GetAsCurrency, a member of class TBCDField.

39.1.7.3.5 TBCDField.GetAsFloat Method

Pascal

```
function GetAsFloat: Double; override;
```

C++

```
virtual __fastcall Double GetAsFloat();
```

Description

This is GetAsFloat, a member of class TBCDField.

39.1.7.3.6 TBCDField.GetAsInteger Method

Pascal

```
function GetAsInteger: Longint; override;
```

C++

```
virtual __fastcall Longint GetAsInteger();
```

Description

This is GetAsInteger, a member of class TBCDField.

39.1.7.3.7 TBCDField.GetAsString Method

Pascal

```
function GetAsString: string; override;
```

C++

```
virtual __fastcall AnsiString GetAsString();
```

Description

This is GetAsString, a member of class TBCDField.

39.1.7.3.8 TBCDField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TBCDField.

39.1.7.3.9 TBCDField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; override;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TBCDField.

39.1.7.3.10 TBCDField.GetDefaultWidth Method

Pascal

```
function GetDefaultWidth: Integer; override;
```

C++

```
virtual __fastcall int GetDefaultWidth();
```

Description

This is GetDefaultWidth, a member of class TBCDField.

39.1.7.3.11 TBCDField.GetText Method

Pascal

```
procedure GetText(var Text: string; DisplayText: Boolean); override;
```

C++

```
virtual __fastcall GetText(AnsiString Text, Boolean DisplayText);
```

Description

This is GetText, a member of class TBCDField.

39.1.7.3.12 TBCDField.GetValue Method

Pascal

```
function GetValue(var Value: Currency): Boolean;
```

C++

```
__fastcall Boolean GetValue(Currency Value);
```

Description

This is GetValue, a member of class TBCDField.

39.1.7.3.13 TBCDField.SetAsBCD Method

Pascal

```
procedure SetAsBCD(const Value: TBcd); override;
```

C++

```
virtual __fastcall SetAsBCD(const TBcd Value);
```

Description

This is SetAsBCD, a member of class TBCDField.

39.1.7.3.14 TBCDField.SetAsCurrency Method

Pascal

```
procedure SetAsCurrency(Value: Currency); override;
```

C++

```
virtual __fastcall SetAsCurrency(Currency Value);
```

Description

This is SetAsCurrency, a member of class TBCDField.

39.1.7.3.15 TBCDField.SetAsFloat Method

Pascal

```
procedure SetAsFloat(Value: Double); override;
```

C++

```
virtual __fastcall SetAsFloat(Double Value);
```

Description

This is SetAsFloat, a member of class TBCDField.

39.1.7.3.16 TBCDField.SetAsInteger Method

Pascal

```
procedure SetAsInteger(Value: Longint); override;
```

C++

```
virtual __fastcall SetAsInteger(Longint Value);
```

Description

This is SetAsInteger, a member of class TBCDField.

39.1.7.3.17 TBCDField.SetAsString Method

Pascal

```
procedure SetAsString(const Value: string); override;
```

C++

```
virtual __fastcall SetAsString(const AnsiString Value);
```

Description

This is SetAsString, a member of class TBCDField.

39.1.7.3.18 TBCDField.SetVarValue Method

Pascal

```
procedure SetVarValue(const Value: Variant); override;
```

C++

```
virtual __fastcall SetVarValue(const Variant Value);
```

Description

This is SetVarValue, a member of class TBCDField.

39.1.7.4 TBCDField Properties

The properties of the TBCDField class are listed here.

Public Properties

	Name	Description
	Value (see page 6790)	Represents the value of the BCD field as a Currency value.

Published Properties

	Name	Description
	currency (see page 6788)	This is currency, a member of class TBCDField.
	MaxValue (see page 6788)	Limits the maximum value in the binary-coded decimal field.
	MinValue (see page 6789)	Limits the minimum value in the binary-coded decimal field.
	Precision (see page 6789)	Specifies the number of digits of accuracy used to store the field value.
	Size (see page 6789)	Indicates the size used in the definition of the physical database field for data types that support different sizes.

39.1.7.4.1 TBCDField.currency Property

Pascal

```
property currency: Boolean;
```

C++

```
__property Boolean currency;
```

Description

This is currency, a member of class TBCDField.

39.1.7.4.2 TBCDField.MaxValue Property

Limits the maximum value in the binary-coded decimal field.

Pascal

```
property MaxValue: Currency;
```

C++

```
__property Currency MaxValue;
```

Description

Use MaxValue to get or set the maximum value that can be entered into the field. If a value greater than MaxValue is entered, TBCDField (see page 6776) raises an EDatabaseError (see page 6731) exception.

Newly created binary-coded decimal (BCD) fields set MaxValue to the maximum value that can be expressed by the Currency data type.

See Also

MinValue (see page 6789)

Value (see page 6790)

39.1.7.4.3 TBCDField.MinValue Property

Limits the minimum value in the binary-coded decimal field.

Pascal

```
property MinValue: Currency;
```

C++

```
__property Currency MinValue;
```

Description

Use MinValue to get or set the minimum value that can be entered into the field. If a value smaller than MinValue is entered, TBCDField (see page 6776) raises an EDatabaseError (see page 6731) exception.

Newly created integer fields set MinValue to the minimum value that can be expressed by the Currency data type.

See Also

MaxValue (see page 6788)

Value (see page 6790)

39.1.7.4.4 TBCDField.Precision Property

Specifies the number of digits of accuracy used to store the field value.

Pascal

```
property Precision: Integer;
```

C++

```
__property int Precision;
```

Description

Precision can be a value between 0 and 32. However, because TBCDField (see page 6776) uses the Currency type to work with data from BCD fields, TBCDField (see page 6776) limits the precision of the BCD values it can support to 4 decimal places and 20 significant digits. The underlying database table may include values that require greater precision, and setting precision to a value greater than 20 allows you to create a table with higher precision.

See Also

AsCurrency

SetData (see page 7141)

39.1.7.4.5 TBCDField.Size Property

Indicates the size used in the definition of the physical database field for data types that support different sizes.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

The interpretation of Size depends on the data type. The meaning of Size for data types that use it is given in the following table.

For all other data types, Size is not used and its value is always 0.

DataType	Meaning of Size
ftBCD	Size is the number of digits after the decimal place.
ftString	Size is the maximum number of characters in the string.
ftVarBytes	Size is the maximum number of bytes of data, not counting the first two bytes which indicate the actual number of bytes of data that were used.
ftBytes	Size is the maximum number of bytes of data.
ftBlob	Size is the number of bytes from the BLOB that are stored in the record buffer.
ftDBaseOle	Size is the number of bytes from the dBASE OLE BLOB that are stored in the record buffer.
ftFMTBCD	Size is the number of digits after the decimal place.
ftFmtMemo	Size is the number of bytes from the memo that are stored in the record buffer.
ftGraphic	Size is the number of bytes from the image that are stored in the record buffer.
ftMemo	Size is the number of characters from the memo that are stored in the record buffer.
ftParadoxOle	Size is the number of bytes from the Paradox OLE BLOB that are stored in the record buffer.
ftTypedBinary	Size is the number of bytes from the BLOB that are stored in the record buffer.
ftADT	Size is the total number of fields contained in the ADT field, including the fields of child fields that are of type ADT.
ftArray	Size is the number of elements in the array.

See Also

DataSize (see page 7157)

DataType (see page 7159)

39.1.7.4.6 TBCDField.Value Property

Represents the value of the BCD field as a Currency value.

Pascal

```
property Value: Currency;
```

C++

```
__property Currency Value;
```

Description

Unlike other field types, the Value of a binary-coded decimal (BCD) field does not correspond to the physical format of the data that is stored in the underlying database table. The IDE does not have a native type for BCD. Therefore, TBCDField (see page 6776) converts the data from a BCD value to a Currency value when it fetches the data from the database table, and converts it from a Currency value to a BCD value when it posts the data. For BCD fields, Value is the same as the AsCurrency (see page 7148) property.

Note: Because TBCDField (see page 6776) works with its data as a Currency value, simply getting the value of the field and posting that value back to the database table can alter the contents of the field. That is, the line below can alter the contents of the field in the database table.

```
MyBCDField.Value := MyBCDField.Value;
MyBCDField->Value = MyBCDField->Value;
```

See Also

AsCurrency

SetData (see page 7141)

39.1.8 TBinaryField Class

TBinaryField represents an untyped binary field in a dataset.

Class Hierarchy

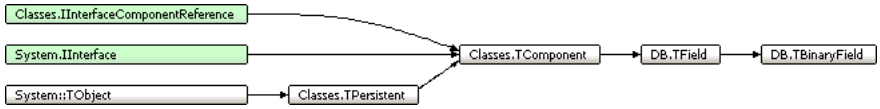

Pascal

```
TBinaryField = class(TField);
```

C++

```
class TBinaryField : public TField;
```

File

DB

Description

TBinaryField encapsulates the fundamental behavior common to untyped binary fields. In addition to the properties, methods, and events common to all fields that are introduced by the TField (see page 7115) class, TBinaryField reimplements the properties that are used to read and write binary data from or to the underlying database table.

If you use the Fields editor at design time to create a persistent field component for the binary field, you can access it by name at runtime. When using dynamic field components, you can access the TBinaryField instance using the dataset's Fields property or FieldByName method.

See Also

TDataSet (see page 6894)

TField (see page 7115)

TDataSet (see page 6894)

TDataSet (see page 6894)

TBytesField (see page 6833)

TVarBytesField (see page 7507)

TMemoField (see page 7348)

TBlobField (see page 6801)

39.1.8.1 TBinaryField Members

The following tables list the members exposed by TBinaryField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 6798)	This is CheckTypeSize, a member of class TBinaryField.
V	CopyData (see page 6798)	This is CopyData, a member of class TBinaryField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
V	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsString (see page 6799)	This is GetAsString, a member of class TBinaryField.
V	GetAsVariant (see page 6799)	This is GetAsVariant, a member of class TBinaryField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
V	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
V	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
V	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
V	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetSize (see page 7135)	This is GetSize, a member of class TField.
V	GetText (see page 6799)	This is GetText, a member of class TBinaryField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).

	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 6799)	This is SetAsString, a member of class TBinaryField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 6799)	This is SetText, a member of class TBinaryField.
	SetVarValue (see page 6800)	This is SetVarValue, a member of class TBinaryField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7126)	Sets the value of the field to NULL.
	Create (see page 6797)	Creates an instance of TBinaryField (see page 6791).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.

	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7180)	Represents the data in a field component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.

	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Size (see page 6800)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.8.2 TBinaryField.Create Constructor

Creates an instance of TBinaryField (see page 6791).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBinaryField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TBinaryField (see page 6791). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TBinaryField (see page 6791). After instantiating a TBinaryField (see page 6791), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TBinaryField (see page 6791) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TBinaryField (see page 6791) with a dataset component by setting its DataSet (see page 7157) property to the name of the dataset component.

The example below creates a TBinaryField (see page 6791) object for a field named RawData accessed through a TSQLQuery named SQLQuery1.

```
var
  T: TBinaryField;
begin
  SQLQuery1.Close;
  T := TBinaryFieldCreate(SQLQuery1);
  T.FieldName := 'RawData';
  T.Name := SQLQuery1.Name + T.FieldName;
  T.Index := SQLQuery1.FieldCount;
  T.DataSet := SQLQuery1;
  SQLQuery1.FieldDefs.UpDate;
  SQLQuery1.Open;
end;
TBinaryField *T;
SQLQuery1->Close();
```

```
T = new TBinaryField(SQLQuery1);
T->FieldName = "RawData";
T->Name = SQLQuery1->Name + T->FieldName;
T->Index = SQLQuery1->FieldCount;
T->DataSet = SQLQuery1;
SQLQuery1->FieldDefs->UpDate();
SQLQuery1->Open();
```

See Also

Create (see page 7121)

Destroy (see page 7122)

39.1.8.3 TBinaryField Methods

The methods of the TBinaryField class are listed here.

Protected Methods

	Name	Description
	CheckTypeSize (see page 6798)	This is CheckTypeSize, a member of class TBinaryField.
	CopyData (see page 6798)	This is CopyData, a member of class TBinaryField.
	GetAsString (see page 6799)	This is GetAsString, a member of class TBinaryField.
	GetAsVariant (see page 6799)	This is GetAsVariant, a member of class TBinaryField.
	GetText (see page 6799)	This is GetText, a member of class TBinaryField.
	SetAsString (see page 6799)	This is SetAsString, a member of class TBinaryField.
	SetText (see page 6799)	This is SetText, a member of class TBinaryField.
	SetVarValue (see page 6800)	This is SetVarValue, a member of class TBinaryField.

39.1.8.3.1 TBinaryField.CheckTypeSize Method

Pascal

```
class procedure CheckTypeSize(Value: Integer); override;
```

C++

```
virtual __fastcall CheckTypeSize(int Value);
```

Description

This is CheckTypeSize, a member of class TBinaryField.

39.1.8.3.2 TBinaryField.CopyData Method

Pascal

```
procedure CopyData(Source: Pointer; Dest: Pointer); override;
```

C++

```
virtual __fastcall CopyData(void * Source, void * Dest);
```

Description

This is CopyData, a member of class TBinaryField.

39.1.8.3.3 TBinaryField.GetAsString Method

Pascal

```
function GetAsString: string; override;
```

C++

```
virtual __fastcall AnsiString GetAsString();
```

Description

This is GetAsString, a member of class TBinaryField.

39.1.8.3.4 TBinaryField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TBinaryField.

39.1.8.3.5 TBinaryField.GetText Method

Pascal

```
procedure GetText(var Text: string; DisplayText: Boolean); override;
```

C++

```
virtual __fastcall GetText(AnsiString Text, Boolean DisplayText);
```

Description

This is GetText, a member of class TBinaryField.

39.1.8.3.6 TBinaryField.SetAsString Method

Pascal

```
procedure SetAsString(const Value: string); override;
```

C++

```
virtual __fastcall SetAsString(const AnsiString Value);
```

Description

This is SetAsString, a member of class TBinaryField.

39.1.8.3.7 TBinaryField.SetText Method

Pascal

```
procedure SetText(const Value: string); override;
```

C++

```
virtual __fastcall SetText(const AnsiString Value);
```

Description

This is SetText, a member of class TBinaryField.

39.1.8.3.8 TBinaryField.SetVarValue Method

Pascal

```
procedure SetVarValue(const Value: Variant); override;
```

C++

```
virtual __fastcall SetVarValue(const Variant Value);
```

Description

This is SetVarValue, a member of class TBinaryField.

39.1.8.4 TBinaryField Properties

The properties of the TBinaryField class are listed here.

Published Properties

	Name	Description
	Size (see page 6800)	Indicates the size used in the definition of the physical database field for data types that support different sizes.

39.1.8.4.1 TBinaryField.Size Property

Indicates the size used in the definition of the physical database field for data types that support different sizes.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

The interpretation of Size depends on the data type. The meaning of Size for data types that use it is given in the following table. For all other data types, Size is not used and its value is always 0.

DataType	Meaning of Size
ftBCD	Size is the number of digits after the decimal place.
ftString	Size is the maximum number of characters in the string.
ftVarBytes	Size is the maximum number of bytes of data, not counting the first two bytes which indicate the actual number of bytes of data that were used.
ftBytes	Size is the maximum number of bytes of data.
ftBlob	Size is the number of bytes from the BLOB that are stored in the record buffer.
ftDBaseOle	Size is the number of bytes from the dBASE OLE BLOB that are stored in the record buffer.
ftFMTBCD	Size is the number of digits after the decimal place.
ftFmtMemo	Size is the number of bytes from the memo that are stored in the record buffer.
ftGraphic	Size is the number of bytes from the image that are stored in the record buffer.

ftMemo	Size is the number of characters from the memo that are stored in the record buffer.
ftParadoxOle	Size is the number of bytes from the Paradox OLE BLOB that are stored in the record buffer.
ftTypedBinary	Size is the number of bytes from the BLOB that are stored in the record buffer.
ftADT	Size is the total number of fields contained in the ADT field, including the fields of child fields that are of type ADT.
ftArray	Size is the number of elements in the array.

See Also

DataSize (see page 7157)

DataType (see page 7159)

39.1.9 TBlobField Class

TBlobField represents a field in a dataset that holds a reference to a binary large object (BLOB).

Class Hierarchy

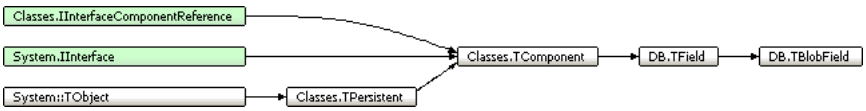

Pascal

```
TBlobField = class(TField);
```

C++

```
class TBlobField : public TField;
```

File

DB

Description

TBlobField encapsulates the fundamental behavior common to binary large object (BLOB) fields. BLOB fields are database fields that contain raw binary data of arbitrary length. BLOB fields can represent different arbitrarily large data types. These data types are distinguished in the header of the binary data.

In addition to the field types supported directly, TBlobField is the direct ancestor of two BLOB field components: TMemoField (see page 7348) (ftMemo) and TGraphicField (see page 7257) (ftGraphic). These descendants represent BLOB fields that have headers peculiar to memos or graphic data, respectively.

TBlobField introduces new methods to support streaming data to and from the BLOB field and to support copying raw binary data between the BLOB field and a binary file. You can also use the stream returned by a dataset's CreateBlobStream method to read or write the data managed by a BLOB field.

If you use the Fields editor at design time to create a persistent field component for the BLOB field, you can access it by name at runtime. When using dynamic field components, you can access the TBlobField instance using the dataset's Fields property or FieldByName method.

See Also

TBinaryField (see page 6791)

TBlobStream (see page 9958)

TADOBlobStream (see page 979)

TSQLBlobStream (see page 20213)

TClientBlobStream (see page 8285)

TGraphicField (see page 7257)

TMemoField (see page 7348)

TDataSet (see page 6894)

TField (see page 7115)

TBytesField (see page 6833)

TVarBytesField (see page 7507)

TMemoField (see page 7348)

TGraphicField (see page 7257)

TDataSet (see page 6894)

39.1.9.1 TBlobField Members

The following tables list the members exposed by TBlobField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 6810)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 6811)	This is FreeBuffers, a member of class TBlobField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.

	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
	GetAsString (see page 6811)	This is GetAsString, a member of class TBlobField.
	GetAsVariant (see page 6811)	This is GetAsVariant, a member of class TBlobField.
	GetAsWideString (see page 6812)	This is GetAsWideString, a member of class TBlobField.
	GetBlobSize (see page 6812)	This is GetBlobSize, a member of class TBlobField.
	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClassDesc (see page 6812)	This is GetClassDesc, a member of class TBlobField.
	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIsNull (see page 6812)	This is GetIsNull, a member of class TBlobField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 6812)	This is GetText, a member of class TBlobField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 6817)	This is SetAsString, a member of class TBlobField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 6817)	This is SetAsWideString, a member of class TBlobField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 6818)	This is SetText, a member of class TBlobField.
	SetVarValue (see page 6818)	This is SetVarValue, a member of class TBlobField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 6809)	Copies a value to the BLOB field.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 6810)	Clears the current value in the BLOB field.
	Create (see page 6808)	Creates an instance of a BLOB field component.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.

	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 6813)	Indicates that this is a BLOB field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	LoadFromFile (see page 6814)	Loads BLOB data from a file into the field.
	LoadFromStream (see page 6815)	Loads BLOB data from a stream into the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 6816)	Saves the contents of the BLOB field to a file.
	SaveToStream (see page 6816)	Saves the contents of the BLOB field to a stream.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 6817)	Specifies the type of the BLOB field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
R	BlobSize (see page 6819)	Indicates the number of bytes of data contained in the blob field on the current record.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	Modified (see page 6820)	Indicates whether a the value for the BLOB field was changed since it was last read from the underlying database.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.

	Transliterate (see page 6821)	Determines whether any text in the BLOB field should be converted to and from the character set identified by the dataset.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 6822)	Represents the data in the BLOB field.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	BlobType (see page 6819)	Identifies the type of the BLOB field associated with a BLOB field object.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
	GraphicHeader (see page 6820)	Indicates whether the field's value includes a Paradox graphic BLOB header.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Size (see page 6820)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.9.2 TBlobField.Create Constructor

Creates an instance of a BLOB field component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBlobField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TBlobField (see page 6801). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create initializes the DataType (see page 7159) property to ftBlob. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TBlobField (see page 6801). After instantiating a TBlobField (see page 6801), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TBlobField (see page 6801) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TBlobField (see page 6801) with a dataset component by setting its DataSet (see page 7157) property to the name of the dataset component.

The example below creates a TBlobField (see page 6801) object for a field named Images accessed through a TSQLDataSet named SQLDataSet1.

```
var
  T: TBlobField;
begin
  SQLDataSet1.Close;
  T := TBlobField.Create(SQLDataSet1);
  T.FieldName := 'Images';
  T.Name := SQLDataSet1.Name + T.FieldName;
  T.Index := SQLDataSet1.FieldCount;
  T.DataSet := SQLDataSet1;
  SQLDataSet1.FieldDefs.UpDate;
  SQLDataSet1.Open;
end;
SQLDataSet1->Close();
TBlobField *T = new TBlobField(SQLDataSet1);
T->FieldName = "Images";
T->Name = SQLDataSet1->Name + T->FieldName;
T->Index = SQLDataSet1->FieldCount;
T->DataSet = SQLDataSet1;
SQLDataSet1->FieldDefs->UpDate();
SQLDataSet1->Open();
```

See Also

Destroy (see page 7122)

BlobType (see page 6819)

DataType (see page 7159)

Create (see page 7121)

39.1.9.3 TBlobField Methods

The methods of the TBlobField class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 6810)	Copies the properties of an object to a destination object.
	FreeBuffers (see page 6811)	This is FreeBuffers, a member of class TBlobField.
	GetAsString (see page 6811)	This is GetAsString, a member of class TBlobField.
	GetAsVariant (see page 6811)	This is GetAsVariant, a member of class TBlobField.
	GetAsWideString (see page 6812)	This is GetAsWideString, a member of class TBlobField.
	GetBlobSize (see page 6812)	This is GetBlobSize, a member of class TBlobField.
	GetClassDesc (see page 6812)	This is GetClassDesc, a member of class TBlobField.
	GetIsNull (see page 6812)	This is GetIsNull, a member of class TBlobField.
	GetText (see page 6812)	This is GetText, a member of class TBlobField.
	SetAsString (see page 6817)	This is SetAsString, a member of class TBlobField.
	SetAsWideString (see page 6817)	This is SetAsWideString, a member of class TBlobField.
	SetText (see page 6818)	This is SetText, a member of class TBlobField.
	SetVarValue (see page 6818)	This is SetVarValue, a member of class TBlobField.

Public Methods

	Name	Description
	Assign (see page 6809)	Copies a value to the BLOB field.
	Clear (see page 6810)	Clears the current value in the BLOB field.
	IsBlob (see page 6813)	Indicates that this is a BLOB field.
	LoadFromFile (see page 6814)	Loads BLOB data from a file into the field.
	LoadFromStream (see page 6815)	Loads BLOB data from a stream into the field.
	SaveToFile (see page 6816)	Saves the contents of the BLOB field to a file.
	SaveToStream (see page 6816)	Saves the contents of the BLOB field to a stream.
	SetFieldType (see page 6817)	Specifies the type of the BLOB field.

39.1.9.3.1 TBlobField.Assign Method

Copies a value to the BLOB field.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use Assign to copy data to the BLOB field. Assign copies the Value (see page 6822) of a BLOB field from

Another BLOB field

A TStrings object

A TPicture or TBitmap object, if the BlobType (see page 6819) is ftGraphic or ftTypedBinary

Any object that supports TBlobField (see page 6801) in its AssignTo (see page 6810) method

In the example below, the Assign method is used to copy the bitmap from a TImage component into a BLOB field through its TBlobField (see page 6801) field object:

```
if not (ClientDataSet1.State in [dsInsert, dsEdit]) then
```

```
  ClientDataSet1.Insert;
ClientDataSet1Images.Assign(Image1.Picture);
ClientDataSet1.Post;
if ((ClientDataSet1->State != dsInsert) &&
(ClientDataSet1->State != dsEdit))
  ClientDataSet1->Insert();
ClientDataSet1Images->Assign(Image1->Picture);
ClientDataSet1->Post();
```

See Also

TBitmap (see page 14396)

TPicture (see page 14584)

TStrings (see page 3029)

AssignTo (see page 2931)

Value (see page 6822)

39.1.9.3.2 TBlobField.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign (see page 2930)

EConvertError (see page 22724)

39.1.9.3.3 TBlobField.Clear Method

Clears the current value in the BLOB field.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Use Clear to delete the current value in the field and leave the field without a value.

```
Query1Images.Clear;
```

See Also

IsNull

Required (see page 7177)

Value (see page 6822)

39.1.9.3.4 TBlobField.FreeBuffers Method

Pascal

```
procedure FreeBuffers; override;
```

C++

```
virtual __fastcall FreeBuffers();
```

Description

This is FreeBuffers, a member of class TBlobField.

39.1.9.3.5 TBlobField.GetAsString Method

Pascal

```
function GetAsString: string; override;
```

C++

```
virtual __fastcall AnsiString GetAsString();
```

Description

This is GetAsString, a member of class TBlobField.

39.1.9.3.6 TBlobField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TBlobField.

39.1.9.3.7 TBlobField.GetAsWideString Method

Pascal

```
function GetAsWideString: WideString; override;
```

C++

```
virtual __fastcall BSTR GetAsWideString();
```

Description

This is GetAsWideString, a member of class TBlobField.

39.1.9.3.8 TBlobField.GetBlobSize Method

Pascal

```
function GetBlobSize: Integer; virtual;
```

C++

```
virtual __fastcall int GetBlobSize();
```

Description

This is GetBlobSize, a member of class TBlobField.

39.1.9.3.9 TBlobField.GetClassDesc Method

Pascal

```
function GetClassDesc: string; override;
```

C++

```
virtual __fastcall AnsiString GetClassDesc();
```

Description

This is GetClassDesc, a member of class TBlobField.

39.1.9.3.10 TBlobField.GetIsNull Method

Pascal

```
function GetIsNull: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsNull();
```

Description

This is GetIsNull, a member of class TBlobField.

39.1.9.3.11 TBlobField.GetText Method

Pascal

```
procedure GetText(var Text: string; DisplayText: Boolean); override;
```

C++

```
virtual __fastcall GetText(AnsiString Text, Boolean DisplayText);
```

Description

This is GetText, a member of class TBlobField.

39.1.9.3.12 TBlobField.IsBlob Method

Indicates that this is a BLOB field.

Pascal

```
class function IsBlob: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsBlob();
```

Description

Use IsBlob when working with a TField (see page 7115) object to determine whether it represents BLOB data. For example, before using the GetData (see page 7131) or SetData (see page 7141) method (which does not work with BLOB fields), call IsBlob to determine whether the method can be used. The IsBlob method for TBlobField (see page 6801) always returns true.

C++ Examples:

```
/*
This example requires a button, a test edit, and a populated
ClientDataSet.  Pipe the ClientDataSet through a DataSource
to a DGGrid or DBNavigator to control the current field.
Cast the data correctly according to the field type when
assigning to the test edit.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // Retrieve the "raw" data from Field1
  if (!CDS->Fields->Fields[0]->IsBlob(__classid(TField))) // GetData does not work for BLOB
fields
  {
    void *MyBuffer = malloc(CDS->Fields->Fields[0]->DataSize);
    try
    {
      if (!CDS->Fields->Fields[0]->GetData(MyBuffer))
        MessageDlg(CDS->Fields->Fields[0]->DisplayName + "is blank.", mtInformation,
TMsgDlgButtons() << mbOK,0);
      else
        // Do something with the data;
        Edit1->Text = (char *)MyBuffer; // for a stringfield
    }
    __finally
    {
      free(MyBuffer);
    }
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CDS->LoadFromFile("../CDS.XML");
}
```

Delphi Examples:

```
{
This example requires a button, a test edit, and a populated
ClientDataSet.  Pipe the ClientDataSet through a DataSource
to a DGGrid or DBNavigator to control the current field.
Cast the data correctly according to the field type when
```

```
  assigning to the test edit.
  }

  {$IFNDEF UNICODE}
  uses SwSystem;
  {$ENDIF}

  procedure TForm1.Button1Click(Sender: TObject);
  var MyBuffer: Pointer;
  begin
  { Retrieve the "raw" data from Field1 }
  with CDS.Fields[0] do
    begin
      if not IsBlob then { this does not work for BLOB fields }
      begin
        { Allocate space }
        MyBuffer:= GetMemory(DataSize);
        try
          if not GetData(MyBuffer) then
            MessageDlg(DisplayName + ' is NULL', mtInformation, [mbOK], 0)
          else
            { Do something with the data };
            Edit1.Text:= string(MyBuffer); // for a stringfield
        finally
          { Free the space }
          FreeMem(MyBuffer, DataSize);
        end;
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
  {$IFDEF UNICODE}
    CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
  {$ELSE}
    CDS.LoadFromFile(gsAppPath + 'CDS.XML');
  {$ENDIF}
  end;
```

See Also

GetData (see page 7131)

SetData (see page 7141)

39.1.9.3.13 TBlobField.LoadFromFile Method

Loads BLOB data from a file into the field.

Pascal

```
procedure LoadFromFile(const FileName: string);
```

C++

```
__fastcall LoadFromFile(const AnsiString FileName);
```

Description

Use LoadFromFile to load the contents of a file into a BLOB field. Specify the name of the file to load into the field as the value of the FileName parameter.

```
if not (Query.State in [dsInsert, dsEdit]) then Query1.Insert;
Query1Images.LoadFromFile('c:\Windows\Waves.bmp');
Query1.Post;
```

See Also

LoadFromStream (see page 6815)

SaveToFile (see page 6816)

CreateBlobStream (see page 6922)

39.1.9.3.14 TBlobField.LoadFromStream Method

Loads BLOB data from a stream into the field.

Pascal

```
procedure LoadFromStream(Stream: TStream);
```

C++

```
__fastcall LoadFromStream(TStream Stream);
```

Description

Use LoadFromStream to copy the contents of a stream into the BLOB field. Specify the stream from which the field's value is copied as the value of the Stream parameter.

Note: The Stream parameter is typically not a BLOB stream. BLOB streams (returned by the dataset's CreateBlobStream method) provide a completely separate mechanism for streaming data into a BLOB field.

```
var
  MS: TMemoryStream;
begin
if not (ClientDataSet1.State in [dsInsert, dsEdit]) then
 ClientDataSet1.Insert;
  MS := TMemoryStream.Create();
try
Image1.Picture.Bitmap.SaveToStream(MS);
    ClientDataSet1Images.LoadFromStream(MS);
finally
    MS.Free;
end;
  ClientDataSet1.Post;
end;
if ((ClientDataSet1->State != dsInsert) &&
(ClientDataSet1->State != dsEdit))
  ClientDataSet1->Insert();
TMemoryStream *pMS = new TMemoryStream;
try
{
  Image1->Picture->Bitmap->SaveToStream(pMS);
  ClientDataSet1Images->LoadFromStream(pMS);
}
__finally
{
delete pMS;
}
ClientDataSet1->Post();
```

See Also

LoadFromFile (see page 6814)

SaveToStream (see page 6816)

CreateBlobStream (see page 6922)

39.1.9.3.15 TBlobField.SaveToFile Method

Saves the contents of the BLOB field to a file.

Pascal

```
procedure SaveToFile(const FileName: string);
```

C++

```
__fastcall SaveToFile(const AnsiString FileName);
```

Description

Use SaveToFile to save the contents of the BLOB field to a file. Specify the name of the file as the value of the FileName parameter.

```
Query1Files.SaveToFile('c:\Data\' + Query1FileName.AsString);
```

See Also

LoadFromFile (see page 6814)

SaveToStream (see page 6816)

39.1.9.3.16 TBlobField.SaveToStream Method

Saves the contents of the BLOB field to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream);
```

C++

```
__fastcall SaveToStream(TStream Stream);
```

Description

Use SaveToStream to copy the contents of a BLOB field to a stream. Specify the name of the stream to which the field's value is saved as the value of the Stream parameter.

Note: The Stream parameter is typically not a BLOB stream. BLOB streams (returned by the dataset's CreateBlobStream method) provide a completely separate mechanism for streaming data from a BLOB field.

```
var
  MS: TMemoryStream;
begin
  MS := TMemoryStream.Create;
try
    SQLDataSet1Images.SaveToStream(MS);
    Image1.Picture.Bitmap.LoadFromStream(MS);
finally
    MS.Free;
end;
end;
TMemoryStream *pMS = new TMemoryStream;
try
{
  SQLDataSet1Images->SaveToStream(pMS);
  Image1->Picture->Bitmap->LoadFromStream(pMS);
}
__finally
{
```

```
delete pMS;
}
```

See Also

LoadFromStream (see page 6815)

SaveToFile (see page 6816)

CreateBlobStream (see page 6922)

39.1.9.3.17 TBlobField.SetAsString Method

Pascal

```
procedure SetAsString(const Value: string); override;
```

C++

```
virtual __fastcall SetAsString(const AnsiString Value);
```

Description

This is SetAsString, a member of class TBlobField.

39.1.9.3.18 TBlobField.SetAsWideString Method

Pascal

```
procedure SetAsWideString(const Value: WideString); override;
```

C++

```
virtual __fastcall SetAsWideString(const BSTR Value);
```

Description

This is SetAsWideString, a member of class TBlobField.

39.1.9.3.19 TBlobField.SetFieldType Method

Specifies the type of the BLOB field.

Pascal

```
procedure SetFieldType(Value: TFieldType); override;
```

C++

```
virtual __fastcall SetFieldType(TFieldType Value);
```

Description

Use SetFieldType to change the field type of a BLOB field, overriding the default type.

Normally, the type of the physical database field is checked and a TFieldType (see page 7612) is selected that matches most closely. BLOB fields can indicate more explicitly what kind of BLOB is in the database. Specify the BLOB type as the value of the Value (see page 6822) parameter.

The following table contains the possible values for BLOB fields. Calling SetFieldType with any other data type does nothing.

Value (see page 6822)	Description
ftBlob	Binary Large Object field
ftMemo	Text (see page 7179) memo field

ftGraphic	Bitmap field
ftFmtMemo	Formatted text memo field
ftParadoxOle	Paradox OLE field
ftDBaseOle	dBASE OLE field
ftTypedBinary	Typed binary field
ftOraBlob	BLOB fields in Oracle 8 tables
ftOraClob	CLOB fields in Oracle 8 tables

Note: Calling SetFieldType does the same thing as setting the BlobType (see page 6819) property.

See Also

BlobType (see page 6819)

DataType (see page 7159)

39.1.9.3.20 TBlobField.SetText Method

Pascal

```
procedure SetText(const Value: string); override;
```

C++

```
virtual __fastcall SetText(const AnsiString Value);
```

Description

This is SetText, a member of class TBlobField.

39.1.9.3.21 TBlobField.SetVarValue Method

Pascal

```
procedure SetVarValue(const Value: Variant); override;
```

C++

```
virtual __fastcall SetVarValue(const Variant Value);
```

Description

This is SetVarValue, a member of class TBlobField.

39.1.9.4 TBlobField Properties

The properties of the TBlobField class are listed here.

Public Properties

	Name	Description
R	BlobSize (see page 6819)	Indicates the number of bytes of data contained in the blob field on the current record.
	Modified (see page 6820)	Indicates whether a the value for the BLOB field was changed since it was last read from the underlying database.
	Transliterate (see page 6821)	Determines whether any text in the BLOB field should be converted to and from the character set identified by the dataset.
	Value (see page 6822)	Represents the data in the BLOB field.

Published Properties

	Name	Description
	BlobType (see page 6819)	Identifies the type of the BLOB field associated with a BLOB field object.
	GraphicHeader (see page 6820)	Indicates whether the field's value includes a Paradox graphic BLOB header.
	Size (see page 6820)	Indicates the size used in the definition of the physical database field for data types that support different sizes.

39.1.9.4.1 TBlobField.BlobSize Property

Indicates the number of bytes of data contained in the blob field on the current record.

Pascal

```
property BlobSize: Integer;
```

C++

```
__property int BlobSize;
```

Description

Read BlobSize to determine the amount of memory required to store the contents of the BLOB field. Use BlobSize instead of the DataSize (see page 7157) property, which always returns 0 for BLOB fields.

Note: Unlike the DataSize (see page 7157) property, which specifies the size for field data on any record, BlobSize gives the size of the BLOB data for the current record only.

In the Delphi example below, a handler for the AfterScroll event of a TQuery, the value of the BlobSize property is displayed in the Caption of a form.

```
procedure TForm1.Query1AfterScroll(DataSet: TDataSet);
begin
  Caption := 'BLOB size: ' +
    IntToStr(TBlobField(Query1.FieldByName('BITMAP')).BlobSize);
end;
```

See Also

DataSize (see page 7157)

39.1.9.4.2 TBlobField.BlobType Property

Identifies the type of the BLOB field associated with a BLOB field object.

Pascal

```
property BlobType: TBlobType;
```

C++

```
__property TBlobType BlobType;
```

Description

Use BlobType to set or read the data type for the BLOB data associated with the field component.

Specify BlobType when using a TBlobField (see page 6801) object to represent a BLOB field type that isn't directly supported (such as a formatted memo BLOB). BlobType ensures that the BLOB type in the dataset matches the expected field type.

See Also

DataType (see page 7159)

SetFieldType (see page 6817)

39.1.9.4.3 TBlobField.GraphicHeader Property

Indicates whether the field's value includes a Paradox graphic BLOB header.

Pascal

```
property GraphicHeader: Boolean;
```

C++

```
__property Boolean GraphicHeader;
```

Description

Use GraphicHeader to specify whether the BLOB field represents a Paradox graphic field. TBlobField (see page 6801) checks GraphicHeader to determine whether it must write out a Paradox graphic BLOB header when writing the field's data to the dataset.

GraphicHeader is ignored unless the DataType (see page 7159) property is ftGraphic or ftTypedBinary.

See Also

DataType (see page 7159)

SetData (see page 7141)

39.1.9.4.4 TBlobField.Modified Property

Indicates whether a the value for the BLOB field was changed since it was last read from the underlying database.

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

If the value of the BLOB field is set by using the properties or methods of TBlobField (see page 6801), the BLOB stream returned by a dataset's CreateBlobStream method, or a data-aware control, Modified is automatically set to true. Set Modified to true when changing the value of a BLOB field using direct Borland (see page 1761) Database Engine (BDE) API calls.

BLOB fields use the Modified property to trigger data events and to indicate when the dataset may need BDE needs to clean up any buffers used to store BLOB data.

See Also

AsString

AsVariant

CreateBlobStream (see page 6922)

Value (see page 6822)

39.1.9.4.5 TBlobField.Size Property

Indicates the size used in the definition of the physical database field for data types that support different sizes.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

The interpretation of Size depends on the data type. The meaning of Size for data types that use it is given in the following table. For all other data types, Size is not used and its value is always 0.

DataType	Meaning of Size
ftBCD	Size is the number of digits after the decimal place.
ftString	Size is the maximum number of characters in the string.
ftVarBytes	Size is the maximum number of bytes of data, not counting the first two bytes which indicate the actual number of bytes of data that were used.
ftBytes	Size is the maximum number of bytes of data.
ftBlob	Size is the number of bytes from the BLOB that are stored in the record buffer.
ftDBaseOle	Size is the number of bytes from the dBASE OLE BLOB that are stored in the record buffer.
ftFMTBCD	Size is the number of digits after the decimal place.
ftFmtMemo	Size is the number of bytes from the memo that are stored in the record buffer.
ftGraphic	Size is the number of bytes from the image that are stored in the record buffer.
ftMemo	Size is the number of characters from the memo that are stored in the record buffer.
ftParadoxOle	Size is the number of bytes from the Paradox OLE BLOB that are stored in the record buffer.
ftTypedBinary	Size is the number of bytes from the BLOB that are stored in the record buffer.
ftADT	Size is the total number of fields contained in the ADT field, including the fields of child fields that are of type ADT.
ftArray	Size is the number of elements in the array.

See Also

DataSize (see page 7157)

DataType (see page 7159)

39.1.9.4.6 TBlobField.Transliterate Property

Determines whether any text in the BLOB field should be converted to and from the character set identified by the dataset.

Pascal

```
property Transliterate: Boolean;
```

C++

```
__property Boolean Transliterate;
```

Description

Use Transliterate when the physical database table identified by the dataset does not use an ANSI language driver and the data may contain extended ASCII characters uses a different character set than UTF-8 (on Linux) or ANSI (on Windows) and the BLOB field contains text.

When Transliterate is true, BLOB fields call the dataset's Translate method to translate ANSI UTF-8 characters to the character set used by the dataset when the value is changed using the Value (see page 6822) property, the AsString (see page 7151) property, or the AsVariant (see page 7152) property. When these properties are read to obtain the value of the BLOB field, the

Translate method translates the characters in the database table to the UTF-8 (Linux) or ANSI (Windows) character set.

Note: BLOB streams (returned by the dataset's CreateBlobStream method) also check the Transliterate property when reading from or writing to a BLOB field.

See Also

AnsiToNative (see page 10244)

AsString

AsVariant

DataSet (see page 7157)

NativeToAnsi (see page 10249)

Locale (see page 9934)

Value (see page 6822)

Translate (see page 7001)

CreateBlobStream (see page 6922)

39.1.9.4.7 TBlobField.Value Property

Represents the data in the BLOB field.

Pascal

```
property Value: string;
```

C++

```
__property AnsiString Value;
```

Description

Use Value to read data directly from and write data directly to a BLOB field component at runtime. For BLOB fields, the Value property is equivalent to the AsString (see page 7151) property.

Note: The string data (Delphi) or AnsiString (C++) type can store binary data as well as text. Thus, even BLOB fields for nontextual BLOB types such as ftGraphic or ftTypedBinary can get or set their data using a string (Delphi) or an AnsiString (C++).

See Also

AsString

BlobType (see page 6819)

39.1.10 TBooleanField Class

TBooleanField represents a field containing Boolean values.

Class Hierarchy

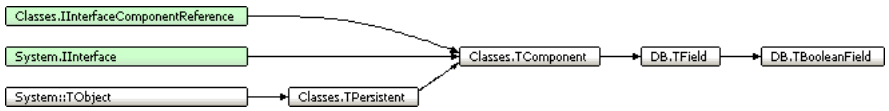

Pascal

```
TBooleanField = class(TField);
```

C++

```
class TBooleanField : public TField;
```

File

DB

Description

TBooleanField encapsulates the fundamental behavior common to Boolean data. Boolean fields can hold values of true or false.

In addition to the properties, methods, and events common to all fields that are introduced by the TField (see page 7115) class, TBooleanField introduces new properties to convert between boolean values and other data types, and to provide display strings that represent various interpretations of the data.

If you use the Fields editor at design time to create a persistent field component for the boolean field, you can access it by name at runtime. When using dynamic field components, you can access the TBooleanField instance using the dataset's Fields property or FieldByName method.

See Also

TDataSet (see page 6894)

TDataSet (see page 6894)

TDataSet (see page 6894)

TField (see page 7115)

39.1.10.1 TBooleanField Members

The following tables list the members exposed by TBooleanField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
	GetAsBoolean (see page 6830)	This is GetAsBoolean, a member of class TBooleanField.
	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
	GetAsString (see page 6830)	This is GetAsString, a member of class TBooleanField.
	GetAsVariant (see page 6830)	This is GetAsVariant, a member of class TBooleanField.
	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
	GetDataSize (see page 6830)	This is GetDataSize, a member of class TBooleanField.
	GetDefaultWidth (see page 6831)	This is GetDefaultWidth, a member of class TBooleanField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 6831)	This is SetAsBoolean, a member of class TBooleanField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 6831)	This is SetAsString, a member of class TBooleanField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.

	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 6831)	This is SetVarValue, a member of class TBooleanField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7126)	Sets the value of the field to NULL.
	Create (see page 6828)	Instantiates a Boolean field object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	Destroy (see page 22287)	Disposes of an object instance.

	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.

	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 6833)	Represents the value of the data in the Boolean field.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayValues (see page 6832)	Controls (see page 5711) how the Boolean field is translated to and from display format.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.10.2 TBooleanField.Create Constructor

Instantiates a Boolean field object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBooleanField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TBooleanField (see page 6822). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftBoolean and initializes the strings associated with Boolean values to 'True' and 'False'. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TBooleanField (see page 6822). After instantiating a TBooleanField (see page 6822), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TBooleanField (see page 6822) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TBooleanField (see page 6822) with a dataset component by setting its DataSet (see page 7157) property to the name of the dataset component.

The example below creates a TBooleanField (see page 6822) object for a field named PaidInFull accessed through a TClientDataSet named ClientDataSet1.

```
var
  T: TBooleanField;
begin
  ClientDataSet1.Close;
  T := TBooleanField.Create(ClientDataSet1);
  T.FieldName := 'PaidInFull';
  T.Name := ClientDataSet1.Name + T.FieldName;
  T.Index := ClientDataSet1.FieldCount;
  T.DataSet := ClientDataSet1;
  ClientDataSet1.FieldDefs.UpDate;
  ClientDataSet1.Open;
end;
ClientDataSet1->Close();
TBooleanField *T = new TBooleanField(ClientDataSet1);
T->FieldName = "PaidInFull";
T->Name = ClientDataSet1->Name + T->FieldName;
T->Index = ClientDataSet1->FieldCount;
T->DataSet = ClientDataSet1;
ClientDataSet1->FieldDefs->UpDate();
ClientDataSet1->Open();
```

See Also

Destroy (see page 7122)

Create (see page 7121)

DataType (see page 7159)

39.1.10.3 TBooleanField Methods

The methods of the TBooleanField class are listed here.

Protected Methods

	Name	Description
	GetAsBoolean (see page 6830)	This is GetAsBoolean, a member of class TBooleanField.
	GetAsString (see page 6830)	This is GetAsString, a member of class TBooleanField.
	GetAsVariant (see page 6830)	This is GetAsVariant, a member of class TBooleanField.
	GetDataSize (see page 6830)	This is GetDataSize, a member of class TBooleanField.
	GetDefaultWidth (see page 6831)	This is GetDefaultWidth, a member of class TBooleanField.
	SetAsBoolean (see page 6831)	This is SetAsBoolean, a member of class TBooleanField.
	SetAsString (see page 6831)	This is SetAsString, a member of class TBooleanField.
	SetVarValue (see page 6831)	This is SetVarValue, a member of class TBooleanField.

39.1.10.3.1 TBooleanField.GetAsBoolean Method

Pascal

```
function GetAsBoolean: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetAsBoolean();
```

Description

This is GetAsBoolean, a member of class TBooleanField.

39.1.10.3.2 TBooleanField.GetAsString Method

Pascal

```
function GetAsString: string; override;
```

C++

```
virtual __fastcall AnsiString GetAsString();
```

Description

This is GetAsString, a member of class TBooleanField.

39.1.10.3.3 TBooleanField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TBooleanField.

39.1.10.3.4 TBooleanField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; override;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TBooleanField.

39.1.10.3.5 TBooleanField.GetDefaultWidth Method

Pascal

```
function GetDefaultWidth: Integer; override;
```

C++

```
virtual __fastcall int GetDefaultWidth();
```

Description

This is GetDefaultWidth, a member of class TBooleanField.

39.1.10.3.6 TBooleanField.SetAsBoolean Method

Pascal

```
procedure SetAsBoolean(Value: Boolean); override;
```

C++

```
virtual __fastcall SetAsBoolean(Boolean Value);
```

Description

This is SetAsBoolean, a member of class TBooleanField.

39.1.10.3.7 TBooleanField.SetAsString Method

Pascal

```
procedure SetAsString(const Value: string); override;
```

C++

```
virtual __fastcall SetAsString(const AnsiString Value);
```

Description

This is SetAsString, a member of class TBooleanField.

39.1.10.3.8 TBooleanField.SetVarValue Method

Pascal

```
procedure SetVarValue(const Value: Variant); override;
```

C++

```
virtual __fastcall SetVarValue(const Variant Value);
```

Description

This is SetVarValue, a member of class TBooleanField.

39.1.10.4 TBooleanField Properties

The properties of the TBooleanField class are listed here.

Public Properties

	Name	Description
	Value (see page 6833)	Represents the value of the data in the Boolean field.

Published Properties

	Name	Description
	DisplayValues (see page 6832)	Controls (see page 5711) how the Boolean field is translated to and from display format.

39.1.10.4.1 TBooleanField.DisplayValues Property

Controls (see page 5711) how the Boolean field is translated to and from display format.

Pascal

```
property DisplayValues: string;
```

C++

```
__property AnsiString DisplayValues;
```

Description

Use DisplayValues to specify strings the field uses to represent Boolean values. Use any pair of phrases, separated by a semicolon. For example, to have the true and false values correspond to the letters T and F, respectively, set DisplayValues as follows:

On Windows:

```
Table1LogicalField.DisplayValues := 'T;F';
```

On cross-platform:

```
ClientDataSet1LogicalField.DisplayValues := 'T;F';
```

On Windows:

```
Table1LogicalField->DisplayValues = "T;F";
```

On cross-platform:

```
ClientDataSet1LogicalField->DisplayValues = "T;F";
```

Similarly, to set the values of true and false to the strings Yes and No, set DisplayValues to:

On Windows:

```
Table1LogicalField.DisplayValues := 'Yes;No';
```

On cross-platform:

```
ClientDataSet1LogicalField.DisplayValues := 'Yes;No';
```

On Windows:

```
Table1LogicalField ->DisplayValues = "Yes;No";
```

On cross-platform:

```
ClientDataSet1LogicalField ->DisplayValues = "Yes;No";
```

The string associated with true or false can be an empty string. To set the value for true to an empty string, set DisplayValues to a string that begins with a semicolon. For example, to associate false with the string Fail, and true with an empty string, set DisplayValues to:

On Windows:

```
Table1LogicalField.DisplayValues := ';Fail';
```

On cross-platform:

```
ClientDataSet1LogicalField.DisplayValues := ';Fail';
```

On Windows:

```
Table1LogicalField ->DisplayValues = ";Fail";
```

On cross-platform:

```
ClientDataSet1LogicalField ->DisplayValues = ";Fail";
```

To associate false with an empty string, set DisplayValues to the string for true, with no semicolon at all.

The strings associated with true and false by DisplayValues appear in data-aware controls when they display the data for a Boolean field. If one of the strings is an empty string, boolean values associated with that string appear blank in data-aware controls.

These strings are also used when getting or setting the Boolean field's AsString (see page 7151) property.

See Also

AsString

39.1.10.4.2 TBooleanField.Value Property

Represents the value of the data in the Boolean field.

Pascal

```
property Value: Boolean;
```

C++

```
__property Boolean Value;
```

Description

Use Value to read data from and write data to a Boolean field object at runtime. Using Value is the same as using the AsBoolean (see page 7147) property. Use Value when you know the field component is a Boolean field. Use AsBoolean (see page 7147) when working with a generic field component.

See Also

AsBoolean

39.1.11 TBytesField Class

TBytesField represents a bytes field in a dataset.

Class Hierarchy

Pascal

```
TBytesField = class(TBinaryField);
```

C++

```
class TBytesField : public TBinaryField;
```

File

DB

Description

TBytesField encapsulates the fundamental behavior common to bytes fields. A bytes field value is a set of unformatted bytes of fixed size.

If you use the Fields editor at design time to create a persistent field component for the bytes field, you can access it by name at runtime. When using dynamic field components, you can access the TBytesField instance using the dataset's Fields property or FieldByName method.

See Also

TBinaryField (see page 6791)

TDataSet (see page 6894)

TGraphicField (see page 7257)

TMemoField (see page 7348)

TDataSet (see page 6894)

TField (see page 7115)

TDataSet (see page 6894)

TDataSet (see page 6894)

TBlobField (see page 6801)

TVarBytesField (see page 7507)

TMemoField (see page 7348)

TGraphicField (see page 7257)

Defining a Data Field

Defining a Lookup Field

Defining a Calculated Field

39.1.11.1 TBytesField Members

The following tables list the members exposed by TBytesField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.

	Bind (see page 7126)	This is Bind, a member of class TField.
	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
	CheckTypeSize (see page 6798)	This is CheckTypeSize, a member of class TBinaryField.
	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
	CopyData (see page 6798)	This is CopyData, a member of class TBinaryField.
	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
	GetAsString (see page 6799)	This is GetAsString, a member of class TBinaryField.
	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
	GetAsVariant (see page 6799)	This is GetAsVariant, a member of class TBinaryField.
	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
	GetDataSize (see page 6841)	This is GetDataSize, a member of class TBytesField.
	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 6799)	This is GetText, a member of class TBinaryField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.

	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
V	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
V	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
V	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
V	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
V	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
V	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
V	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
V	SetAsString (see page 6799)	This is SetAsString, a member of class TBinaryField.
V	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
V	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
V	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
V	SetSize (see page 7143)	This is SetSize, a member of class TField.
V	SetText (see page 6799)	This is SetText, a member of class TBinaryField.
V	SetText (see page 7144)	This is SetText, a member of class TField.
V	SetVarValue (see page 6800)	This is SetVarValue, a member of class TBinaryField.
V	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
V	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.

	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 7126)	Sets the value of the field to NULL.
V	Create (see page 6840)	Creates an instance of TBytesField (see page 6833).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.

R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7180)	Represents the data in a field component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.

	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Size (see page 6800)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.11.2 TBytesField.Create Constructor

Creates an instance of TBytesField (see page 6833).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBytesField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TBytesField (see page 6833). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftBytes. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TBytesField (see page 6833). After instantiating a TBytesField (see page 6833), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TBytesField (see page 6833) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TBytesField (see page 6833) with a dataset component by setting its DataSet (see page 7157) property to the name of the dataset component.

The example below creates a TBytesField (see page 6833) object for a field named BytesField accessed through a TSQLDataSet named SQLDataSet1.

```
var
  T: TBytesField;
begin
  SQLDataSet1.Close;
  T := TBytesField.Create(SQLDataSet1);
  T.FieldName := 'BytesField';
  T.Name := SQLDataSet1.Name + T.FieldName;
  T.Index := SQLDataSet1.FieldCount;
  T.DataSet := SQLDataSet1;
  SQLDataSet1.FieldDefs.UpDate;
  SQLDataSet1.Open;
end;
SQLDataSet1->Close();
TBytesField *T = new TBytesField(SQLDataSet1);
T->FieldName = "BytesField";
T->Name = SQLDataSet1->Name + T->FieldName;
```

```
T->Index = SQLDataSet1->FieldCount;
T->DataSet = SQLDataSet1;
SQLDataSet1->FieldDefs->UpDate();
SQLDataSet1->Open();
```

See Also

Destroy (see page 7122)

DataType (see page 7159)

Create (see page 7121)

Owner (see page 2807)

39.1.11.3 TBytesField Methods

The methods of the TBytesField class are listed here.

Protected Methods

	Name	Description
	GetDataSize (see page 6841)	This is GetDataSize, a member of class TBytesField.

39.1.11.3.1 TBytesField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; override;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TBytesField.

39.1.12 TCheckConstraint Class

TCheckConstraint represents a record-level constraint on a dataset component.

Class Hierarchy

System::TObject → Classes.TPersistent → Classes.TCollectionItem → DB.TCheckConstraint

Pascal

```
TCheckConstraint = class(TCollectionItem);
```

C++

```
class TCheckConstraint : public TCollectionItem;
```

File

DB

Description

BDE-enabled and C client dataset components implement a Constraints property that uses check constraints to represent record-level constraints. These constraints can be derived from the server or the data dictionary, and can be customized in

applications. Each TCheckConstraint object represents an individual constraint associated with a dataset.

See Also

Constraining Data Values

39.1.12.1 TCheckConstraint Members

The following tables list the members exposed by TCheckConstraint.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 6843)	Assigns one constraint to another.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2757)	Creates and initializes a TCollectionItem (see page 2755) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetDisplayName (see page 6843)	Retrieves the current text for a constraint.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	CustomConstraint (see page 6844)	Contains the SQL text representation of a constraint defined on a record by an application.
	ErrorMessage (see page 6845)	Contains the text of an error message to display if data entered in a record by a user violates the constraint condition.
	FromDictionary (see page 6845)	Indicates whether to fetch a check constraint from the Data Dictionary. Reserved for future use.
	ImportedConstraint (see page 6845)	Contains the SQL text for a check constraint imported from the Data Dictionary or a remote SQL database server.

39.1.12.2 TCheckConstraint Methods

The methods of the TCheckConstraint class are listed here.

Public Methods

	Name	Description
V	Assign (see page 6843)	Assigns one constraint to another.
V	GetDisplayName (see page 6843)	Retrieves the current text for a constraint.

39.1.12.2.1 TCheckConstraint.Assign Method

Assigns one constraint to another.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the SQL that defines the constraint of another TCheckConstraint (see page 6841) object into this one. Source is the TCheckConstraint (see page 6841) from which to read text.

39.1.12.2.2 TCheckConstraint.GetDisplayName Method

Retrieves the current text for a constraint.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

Call GetDisplayName to retrieve the current text for a check constraint. GetDisplayName returns the current value of the ImportedConstraint (see page 6845) property, unless it is an empty string. In that case it returns the current value of the CustomConstraint (see page 6844) property. If it, too, is an empty string, GetDisplayName returns the class name of the object.

See Also

CustomConstraint (see page 6844)

ImportedConstraint (see page 6845)

39.1.12.3 TCheckConstraint Properties

The properties of the TCheckConstraint class are listed here.

Published Properties

	Name	Description
	CustomConstraint (see page 6844)	Contains the SQL text representation of a constraint defined on a record by an application.
	ErrorMessage (see page 6845)	Contains the text of an error message to display if data entered in a record by a user violates the constraint condition.
	FromDictionary (see page 6845)	Indicates whether to fetch a check constraint from the Data Dictionary. Reserved for future use.
	ImportedConstraint (see page 6845)	Contains the SQL text for a check constraint imported from the Data Dictionary or a remote SQL database server.

39.1.12.3.1 TCheckConstraint.CustomConstraint Property

Contains the SQL text representation of a constraint defined on a record by an application.

Pascal

```
property CustomConstraint: string;
```

C++

```
__property AnsiString CustomConstraint;
```

Description

Use CustomConstraint to store the SQL text that defines a check constraint defined by the application for a dataset component. Custom constraints are local to the application and do not get applied by the server when an application writes its changes to a database.

Note: Custom constraints are SQL expressions. Because of this, always use a period (.) as a floating-point separator. This may be inconsistent with international Windows

settings.

See Also

FromDictionary (see page 6845)

ImportedConstraint (see page 6845)

Constraining Data Values

39.1.12.3.2 TCheckConstraint.ErrorMessage Property

Contains the text of an error message to display if data entered in a record by a user violates the constraint condition.

Pascal

```
property ErrorMessage: string;
```

C++

```
__property AnsiString ErrorMessage;
```

Description

Use ErrorMessage to set the error message string to display in an exception message dialog if a user enters data in a record that violates a check constraint condition.

See Also

CustomConstraint (see page 6844)

ImportedConstraint (see page 6845)

39.1.12.3.3 TCheckConstraint.FromDictionary Property

Indicates whether to fetch a check constraint from the Data Dictionary. Reserved for future use.

Pascal

```
property FromDictionary: Boolean;
```

C++

```
__property Boolean FromDictionary;
```

Description

FromDictionary is only implemented for the Windows product, which has a data dictionary.

Use FromDictionary to indicate whether this check constraint is derived from the Data Dictionary (True) (true) , or is local to the application (False) (false) .

See Also

CustomConstraint (see page 6844)

ImportedConstraint (see page 6845)

39.1.12.3.4 TCheckConstraint.ImportedConstraint Property

Contains the SQL text for a check constraint imported from the Data Dictionary or a remote SQL database server.

Pascal

```
property ImportedConstraint: string;
```

C++

```
__property AnsiString ImportedConstraint;
```

Description

ImportedConstraint stores the SQL text for a check constraint that is imported from the server or Data Dictionary. Although an application can set this property at runtime, ordinarily the text in ImportedConstraint is automatically provided to the application.

Note:

See Also

CustomConstraint (see page 6844)

FromDictionary (see page 6845)

39.1.13 TCheckConstraints Class

TCheckConstraints contains a collection of check constraints for a dataset component.

Class Hierarchy

Pascal

```
TCheckConstraints = class(TCollection);
```

C++

```
class TCheckConstraints : public TCollection;
```

File

DB

Description

BDE-enabled and c Client dataset components support check constraints that define the values that can be entered into a single record. The TCheckConstraints object maintains the collection of check constraints that apply to a given dataset.

See Also

TDataSet (see page 6894)

39.1.13.1 TCheckConstraints Members

The following tables list the members exposed by TCheckConstraints.

Protected Methods

	Name	Description
	Added (see page 2736)	Responds when items are added to the collection.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
	GetOwner (see page 6849)	Returns the owner of an object.
	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.

V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 6848)	Adds a check constraint to the list maintained by this collection object.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 6848)	Instantiates an instance of a check constraint collection object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 6849)	Maintains an indexed array of check constraints.

39.1.13.2 TCheckConstraints.Create Constructor

Instantiates an instance of a check constraint collection object.

Pascal

```
constructor Create(Owner: TPersistent);
```

C++

```
__fastcall TCheckConstraints(TPersistent * Owner);
```

Description

Most applications do not need to instantiate TCheckConstraints (see page 6846). This object is created by the dataset that uses it to implement its Constraints property.

Call Create to instantiate a check constraint collection at runtime when you need a collection object and it has not been created automatically for you. Create calls its inherited constructor, and sets the collection's owner object to Owner.

See Also

Constraints (see page 7013)

39.1.13.3 TCheckConstraints Methods

The methods of the TCheckConstraints class are listed here.

Protected Methods

	Name	Description
V	GetOwner (see page 6849)	Returns the owner of an object.

Public Methods

	Name	Description
	Add (see page 6848)	Adds a check constraint to the list maintained by this collection object.

39.1.13.3.1 TCheckConstraints.Add Method

Adds a check constraint to the list maintained by this collection object.

Pascal

```
function Add: TCheckConstraint;
```

C++

```
__fastcall TCheckConstraint Add();
```

Description

Call Add to add another check constraint to those known to the dataset component that owns this check constraint object. After adding a check constraint, set its properties to provide the SQL that defines the constraint.

See Also

Items (see page 6849)

39.1.13.3.2 TCheckConstraints.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

39.1.13.4 TCheckConstraints Properties

The properties of the TCheckConstraints class are listed here.

Public Properties

	Name	Description
	Items (see page 6849)	Maintains an indexed array of check constraints.

39.1.13.4.1 TCheckConstraints.Items Property

Maintains an indexed array of check constraints.

Pascal

```
property Items [Index: Integer]: TCheckConstraint;
```

C++

```
__property TCheckConstraint Items[int Index];
```

Description

Use Items to access individual check constraints that apply to the dataset component that owns this check constraint collection.

See Also

Add (see page 6848)

operator_sb

39.1.14 TCurrencyField Class

TCurrencyField represents a field that contains currency (see page 6857) values in a dataset.

Class Hierarchy

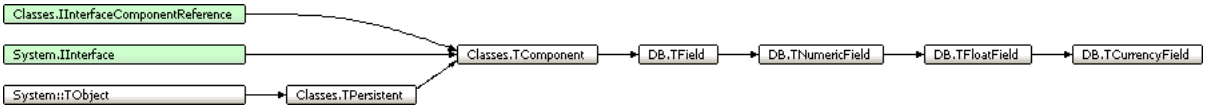

Pascal

```
TCurrencyField = class(TFloatField);
```

C++

```
class TCurrencyField : public TFloatField;
```

File

DB

Description

TCurrencyField encapsulates the fundamental behavior common to currency (see page 6857) fields. TCurrencyField differs from its immediate ancestor TFloatField (see page 7229) only in having a DataType (see page 7159) of ftCurrency, and in formatting the value using a currency (see page 6857) format (one that represents monetary values) by default. Currency fields can hold values in the range from (positive or negative) 5.0 * 10^-324 to 1.7 * 10^308 with an accuracy of 15 digits.

Do not confuse TCurrencyField with the Currency data type. Currency fields use the double data type to store and manipulate their values. This data type is the format used by the physical database tables for currency (see page 6857) fields. The TBCDField (see page 6776) class uses the Currency data type to store and manipulate its value.

If you use the Fields editor at design time to create a persistent field component for the currency (see page 6857) field, you can access it by name at runtime. When using dynamic field components, you can access the TCurrencyField instance using the dataset's Fields property or FieldByName method.

See Also

Using Default Formatting for Numeric

39.1.14.1 TCurrencyField Members

The following tables list the members exposed by TCurrencyField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
V	GetAsFloat (see page 7237)	This is GetAsFloat, a member of class TFloatField.
V	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
V	GetAsInteger (see page 7237)	This is GetAsInteger, a member of class TFloatField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
V	GetAsString (see page 7238)	This is GetAsString, a member of class TFloatField.
V	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
V	GetAsVariant (see page 7238)	This is GetAsVariant, a member of class TFloatField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
V	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
V	GetDataSize (see page 7238)	This is GetDataSize, a member of class TFloatField.
V	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
V	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
V	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetSize (see page 7135)	This is GetSize, a member of class TField.
V	GetText (see page 7135)	This is GetText, a member of class TField.
V	GetText (see page 7238)	This is GetText, a member of class TFloatField.

	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	RangeError (see page 7368)	This is RangeError, a member of class TNumericField.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsFloat (see page 7239)	This is SetAsFloat, a member of class TFloatField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsInteger (see page 7239)	This is SetAsInteger, a member of class TFloatField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsString (see page 7239)	This is SetAsString, a member of class TFloatField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayFormat (see page 7368)	This is SetDisplayFormat, a member of class TNumericField.
	SetEditFormat (see page 7368)	This is SetEditFormat, a member of class TNumericField.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
	SetVarValue (see page 7239)	This is SetVarValue, a member of class TFloatField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 7126)	Sets the value of the field to NULL.
V	Create (see page 6856)	Creates an instance of TCurrencyField (see page 6850).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.

	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7180)	Represents the data in a field component.
	Value (see page 7242)	Specifies the value of the floating-point field.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	Alignment (see page 7369)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.

	currency (see page 6857)	Determines whether the value in the field should be formatted as a currency value.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayFormat (see page 7369)	Determines how a numeric field's value is formatted for display in a data-aware control.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	EditFormat (see page 7371)	Determines how a numeric field's value is formatted when it is being edited in a data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	MaxValue (see page 7240)	Limits the maximum value in the floating-point field.
	MinValue (see page 7241)	Limits the minimum value in the floating-point field.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	Precision (see page 7241)	Determines the precision used in formatting the value in a floating-point field.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.14.2 TCurrencyField.Create Constructor

Creates an instance of TCurrencyField (see page 6850).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCurrencyField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TCurrencyField (see page 6850). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftCurrency and the Currency property to true. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TCurrencyField (see page 6850). After instantiating a TCurrencyField (see page 6850), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TCurrencyField (see page 6850) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TCurrencyField (see page 6850) with a dataset component by setting its DataSet (see page 7157) property to the name of the dataset component.

The example below creates a TCurrencyField (see page 6850) object for a field named Amount accessed through a TSQLDataSet named SQLDataSet1.

```
var
  T: TCurrencyField;
begin
  SQLDataSet1.Close;
  T := TCurrencyField.Create(SQLDataSet1);
  T.FieldName := 'Amount';
  T.Name := SQLDataSet1.Name + T.FieldName;
  T.Index := SQLDataSet1.FieldCount;
  T.DataSet := SQLDataSet1;
  SQLDataSet1.FieldDefs.UpDate;
  SQLDataSet1.Open;
end;
SQLDataSet1->Close();
TCurrencyField *T = new TCurrencyField(SQLDataSet1);
T->FieldName = "Amount";
T->Name = SQLDataSet1->Name + T->FieldName;
T->Index = SQLDataSet1->FieldCount;
T->DataSet = SQLDataSet1;
SQLDataSet1->FieldDefs->UpDate();
SQLDataSet1->Open();
```

See Also

Destroy (see page 7122)

currency (see page 7240)

DataType (see page 7159)

Create (see page 7367)

Owner (see page 2807)

39.1.14.3 TCurrencyField Properties

The properties of the TCurrencyField class are listed here.

Published Properties

	Name	Description
	currency (see page 6857)	Determines whether the value in the field should be formatted as a currency value.

39.1.14.3.1 TCurrencyField.currency Property

Determines whether the value in the field should be formatted as a currency value.

Pascal

```
property currency: Boolean;
```

C++

```
__property Boolean currency;
```

Description

Use the currency property to specify the formatting of the field for viewing and editing in a data-aware control.

The field is formatted for display using the DisplayFormat property if it is assigned. If DisplayFormat is not assigned, currency determines how the field is formatted for display. If currency is true, the value is formatted for display using the FloatToText function with the ffCurrency formatting code. If currency is false, the value is formatted with the ffGeneral format.

The field is formatted for editing purposes using the EditFormat (or DisplayFormat) property if it is assigned. If neither EditFormat nor DisplayFormat is assigned, currency determines how the field is formatted for editing. If currency is true, the value is formatted for editing using the FloatToText function with the ffFixed formatting code. If currency is false, the value is formatted with the ffGeneral format.

See Also

DisplayFormat (see page 7369)

DisplayText (see page 7162)

EditFormat (see page 7371)

FloatToText (see page 22892)

FloatToTextFmt (see page 22893)

Text (see page 7179)

39.1.15 TCustomConnection Class

TCustomConnection is the base class for components that connect datasets to a remote source of data.

Class Hierarchy

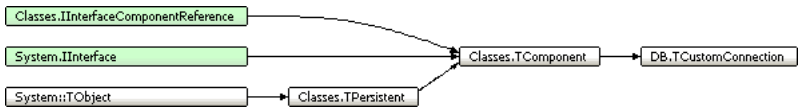

Pascal

```
TCustomConnection = class(TComponent);
```

C++

```
class TCustomConnection : public TComponent;
```

File

DB

Description

Use TCustomConnection as a base class for components that represent a remote source of the data contained in a dataset. This source can be a remote database server, the middle tier of a multi-tiered database application, an interface such as the one used by ActiveX Data Objects (ADO), or any other driver that provides the information represented in a dataset. TCustomConnection introduces:

Methods for connecting to and disconnecting from the source of data.

Login support for providing a user name and password when establishing a connection.

Properties for keeping track of the datasets that use the connection component.

Do not create instances of TCustomConnection. To add a component that represents the connection to a source of data, use a TCustomConnection descendant such as TDatabase, TADOConnection, TSQLConnection, or one of the DataSnap connection components such as TDCOMConnection TSQLConnection or TSOAPConnection.

See Also

TDatabase (see page 9964)

TDCOMConnection (see page 16008)

TSocketConnection (see page 18928)

TWebConnection (see page 18968)

TADOConnection (see page 1001)

TDataSet (see page 6894)

TIBDatabase

TSQLConnection (see page 20217)

TSOAPConnection

Connecting to Databases: Overview

39.1.15.1 TCustomConnection Members

The following tables list the members exposed by TCustomConnection.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoConnect (see page 6864)	Provides the interface for a method that opens a connection.
V	DoDisconnect (see page 6864)	Provides the interface for a method that terminates the connection.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetConnected (see page 6865)	Returns the value of the Connected (see page 6868) property.
V	GetDataSet (see page 6865)	Returns a dataset, given its index in the DataSets (see page 6870) property array.
V	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 6866)	Initializes the connection component after all objects in the form or data module have been loaded.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RegisterClient (see page 6866)	Registers an object that uses the connection component.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnected (see page 6867)	Changes the value of the Connected (see page 6868) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	UnRegisterClient (see page 6868)	Removes a registered client so that it is no longer associated with the connection component.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.

Public Events

	Name	Description
	AfterConnect (see page 6874)	Occurs after a connection is established.
	AfterDisconnect (see page 6875)	Occurs after the connection closes.
	BeforeConnect (see page 6875)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 6875)	Occurs immediately before the connection closes.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.

	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 6863)	Closes the connection.
	Create (see page 6862)	Creates an instance of TCustomConnection (see page 6858).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 6863)	Destroys the instance of a connection component.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 6866)	Opens the connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)

V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Connected (see page 6868)	Determines whether a connection has been established to the remote source of data.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 6870)	Lists the datasets currently using this connection component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	LoginPrompt (see page 6872)	Specifies whether a login dialog appears immediately before opening a new connection.
	OnLogin (see page 6873)	Occurs when a communication channel opens to the server.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

39.1.15.2 TCustomConnection.Create Constructor

Creates an instance of TCustomConnection (see page 6858).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomConnection(TComponent * AOwner);
```

Description

Call Create to instantiate a connection component at runtime. Connection components added to forms or data modules at design time are created automatically.

The Owner parameter specifies another component (usually a form or data module) that is responsible for freeing the connection component.

See Also

Owner (see page 2807)

39.1.15.3 TCustomConnection.Destroy Destructor

Destroys the instance of a connection component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomConnection();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which verifies that the reference to the connection component is not nil before calling Destroy

Destroy closes all active datasets and closes its connection before freeing all internal resources.

See Also

Create (see page 6862)

Free (see page 22300)

39.1.15.4 TCustomConnection Methods

The methods of the TCustomConnection class are listed here.

Protected Methods

	Name	Description
	DoConnect (see page 6864)	Provides the interface for a method that opens a connection.
	DoDisconnect (see page 6864)	Provides the interface for a method that terminates the connection.
	GetConnected (see page 6865)	Returns the value of the Connected (see page 6868) property.
	GetDataSet (see page 6865)	Returns a dataset, given its index in the DataSets (see page 6870) property array.
	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.
	Loaded (see page 6866)	Initializes the connection component after all objects in the form or data module have been loaded.
	RegisterClient (see page 6866)	Registers an object that uses the connection component.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.
	SetConnected (see page 6867)	Changes the value of the Connected (see page 6868) property.
	UnRegisterClient (see page 6868)	Removes a registered client so that it is no longer associated with the connection component.

Public Methods

	Name	Description
	Close (see page 6863)	Closes the connection.
	Open (see page 6866)	Opens the connection.

39.1.15.4.1 TCustomConnection.Close Method

Closes the connection.

Pascal

```
procedure Close;
```

C++

```
__fastcall Close();
```

Description

Call Close to disconnect from the remote source of database information. Before the connection component is deactivated, all associated datasets are closed. Calling Close is the same as setting the Connected (see page 6868) property to false.

In most cases, closing a connection frees system resources allocated to the connection.

Note: If a previously active connection is closed and then reopened, any associated datasets must be individually reopened; reopening the connection does not automatically reopen associated datasets.

See Also

Connected (see page 6868)

Disconnecting from a Database Server

39.1.15.4.2 TCustomConnection.DoConnect Method

Provides the interface for a method that opens a connection.

Pascal

```
procedure DoConnect; virtual;
```

C++

```
virtual __fastcall DoConnect();
```

Description

The Connected (see page 6868) property uses DoConnect to establish a connection. Descendant classes override the DoConnect method to establish their connection as appropriate. As implemented in TCustomConnection (see page 6858), DoConnect does nothing.

See Also

Connected (see page 6868)

DoDisconnect (see page 6864)

39.1.15.4.3 TCustomConnection.DoDisconnect Method

Provides the interface for a method that terminates the connection.

Pascal

```
procedure DoDisconnect; virtual;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

The Connected (see page 6868) property uses DoDisconnect to close a connection. Descendant classes override the DoDisconnect method to drop a connection. As implemented in TCustomConnection (see page 6858), DoDisconnect does nothing.

See Also

Connected (see page 6868)

DoConnect (see page 6864)

39.1.15.4.4 TCustomConnection.GetConnected Method

Returns the value of the Connected (see page 6868) property.

Pascal

```
function GetConnected: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected read implementation of the Connected (see page 6868) property. Descendant classes override this method to return true if a connection is open and false otherwise. As defined in TCustomConnection (see page 6858), GetConnected always returns false.

See Also

Connected (see page 6868)

SetConnected (see page 6867)

39.1.15.4.5 TCustomConnection.GetDataSet Method

Returns a dataset, given its index in the DataSets (see page 6870) property array.

Pascal

```
function GetDataSet(Index: Integer): TDataSet; virtual;
```

C++

```
virtual __fastcall TDataSet GetDataSet(int Index);
```

Description

GetDataSet is the protected read implementation of the DataSets (see page 6870) property.

See Also

DataSets (see page 6870)

39.1.15.4.6 TCustomConnection.GetDataSetCount Method

Returns the value of the DataSetCount (see page 6869) property.

Pascal

```
function GetDataSetCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetDataSetCount();
```

Description

GetDataSetCount is the protected read implementation of the DataSetCount (see page 6869) property.

See Also

DataSetCount (see page 6869)

39.1.15.4.7 TCustomConnection.Loaded Method

Initializes the connection component after all objects in the form or data module have been loaded.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. It is called automatically after the form or data module that contains the connection component has loaded all its owned objects. TCustomConnection (see page 6858) overrides the inherited method to establish a connection if the Connected (see page 6868) property is true.

See Also

StreamedConnected (see page 6874)

Loaded (see page 2782)

39.1.15.4.8 TCustomConnection.Open Method

Opens the connection.

Pascal

```
procedure Open; overload;
```

C++

```
__fastcall Open();
```

Description

Call Open to establish a connection to the remote source of database information. Open sets the Connected (see page 6868) property to true.

See Also

Connected (see page 6868)

Connecting to a Database Server

39.1.15.4.9 TCustomConnection.RegisterClient Method

Registers an object that uses the connection component.

Pascal

```
procedure RegisterClient(Client: TObject; Event: TConnectChangeEvent = nil); virtual;
```

C++

```
virtual __fastcall RegisterClient(TObject * Client, TConnectChangeEvent Event = nil);
```

Description

RegisterClient is used internally to associate objects with the connection component. Registered components can supply an event handler that responds when the connection is created or terminated.

Client is the object to register. If Client is a dataset, it is added to the DataSets (see page 6870) property.

Event is an event handler that is called after the connection is established or before it is terminated.

Datasets call this method when they start using the connection component. For BDE-enabled datasets, ADO datasets, and dbExpress datasets, this is when they are activated. For client datasets, this is when the RemoteServer property is assigned.

See Also

DataSets (see page 6870)

UnRegisterClient (see page 6868)

39.1.15.4.10 TCustomConnection.SendConnectEvent Method

Informs all registered clients that use the connection component when a connection is opened or closed.

Pascal

```
procedure SendConnectEvent(Connecting: Boolean);
```

C++

```
__fastcall SendConnectEvent(Boolean Connecting);
```

Description

SendConnectEvent is used internally to ensure that registered clients are informed when their connection is opened or closed. It is called automatically when the Connected (see page 6868) property changes.

SendConnectEvent iterates through all registered clients. For each client that supplied an event handler to the RegisterClient (see page 6866) method, SendConnectEvent calls that event handler to inform the client that a connection has just been opened (when Connecting is true) or that the connection is about to close (when Connecting is false). If the client is a dataset, it is informed about the change even if it did not supply an event handler to RegisterClient (see page 6866).

See Also

DataSets (see page 6870)

Connected (see page 6868)

RegisterClient (see page 6866)

39.1.15.4.11 TCustomConnection.SetConnected Method

Changes the value of the Connected (see page 6868) property.

Pascal

```
procedure SetConnected(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetConnected(Boolean Value);
```

Description

SetConnected is the protected write implementation of the Connected (see page 6868) property. Most descendants of TCustomConnection (see page 6858) do not override this method to change the implementation of Connected (see page 6868). Instead, they override the DoConnect (see page 6864) and DoDisconnect (see page 6864) methods to change the way the connection is opened and closed.

See Also

Connected (see page 6868)

DoConnect (see page 6864)

DoDisconnect (see page 6864)

39.1.15.4.12 TCustomConnection.UnRegisterClient Method

Removes a registered client so that it is no longer associated with the connection component.

Pascal

```
procedure UnRegisterClient(Client: TObject); virtual;
```

C++

```
virtual __fastcall UnRegisterClient(TObject * Client);
```

Description

UnRegisterClient is used internally to remove objects from the internal list of objects that are informed when a connection is created or terminated.

Client is the object that is removed from the list. It should be an object that was previously registered by a call to RegisterClient (see page 6866). If Client is a dataset, it is removed from the DataSets (see page 6870) property list.

Datasets call this method when they stop using the connection component. For BDE-enabled datasets, ADO datasets, and dbExpress datasets, this is when they are deactivated. For client datasets, this is when the RemoteServer property is changed.

See Also

DataSets (see page 6870)

RegisterClient (see page 6866)

39.1.15.5 TCustomConnection Properties

The properties of the TCustomConnection class are listed here.

Protected Properties

	Name	Description
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.

Public Properties

	Name	Description
	Connected (see page 6868)	Determines whether a connection has been established to the remote source of data.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 6870)	Lists the datasets currently using this connection component.
	LoginPrompt (see page 6872)	Specifies whether a login dialog appears immediately before opening a new connection.
	OnLogin (see page 6873)	Occurs when a communication channel opens to the server.

39.1.15.5.1 TCustomConnection.Connected Property

Determines whether a connection has been established to the remote source of data.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Set Connected to true to open the connection. Set Connected to false to terminate the connection.

Setting Connected to true generates a BeforeConnect (see page 6875) event, calls the protected DoConnect (see page 6864) method to establish the connection, and generates an AfterConnect (see page 6874) event. In addition, when setting Connected to true, TCustomConnection (see page 6858) may display a login dialog, depending on the value of LoginPrompt (see page 6872).

Setting Connected to false generates a BeforeDisconnect (see page 6875) event, calls the protected DoConnect (see page 6864) method to drop the connection, and generates an AfterDisconnect (see page 6875) event.

When deriving custom connection components from TCustomConnection (see page 6858), override GetConnected (see page 6865) to return true when a connection is established, and override DoConnect (see page 6864) and DoDisconnect (see page 6864) to create and drop the connection.

See Also

DataSets (see page 6870)

Open (see page 6866)

Close (see page 6863)

DoConnect (see page 6864)

DoDisconnect (see page 6864)

BeforeConnect (see page 6875)

AfterConnect (see page 6874)

BeforeDisconnect (see page 6875)

AfterDisconnect (see page 6875)

GetConnected (see page 6865)

LoginPrompt (see page 6872)

Disconnecting from a Database Server

Connecting to a Database Server

39.1.15.5.2 TCustomConnection.DataSetCount Property

Indicates the number of datasets associated with the connection component.

Pascal

```
property DataSetCount: Integer;
```

C++

```
__property int DataSetCount;
```

Description

Use DataSetCount to determine the number of datasets listed by the DataSets (see page 6870) property. Depending on the TCustomConnection (see page 6858) descendant, DataSets (see page 6870) may include only the active datasets that use the connection component (as is the case with TDatabase and TSQLConnection), or it may list all datasets, regardless of whether they are active (as is the case with TADODatabase and DataSnap connection components TSOAPConnection). If DataSets (see page 6870) includes only active datasets, the value of DataSetCount changes when datasets are opened and closed.

Use DataSetCount as an upper bound when iterating through the DataSets (see page 6870) property.

Delphi Examples:

```
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
  RemoteServer1.Connected := False;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].DisableControls;
  end;
end;

procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
```

See Also

DataSets (see page 6870)

Active (see page 7004)

Working with Associated Datasets

39.1.15.5.3 TCustomConnection.DataSets Property

Lists the datasets currently using this connection component.

Pascal

```
property DataSets [Index: Integer]: TDataSet;
```

C++

```
__property TDataSet DataSets[int Index];
```

Description

Use DataSets to access the datasets that are currently using the connection component. Depending on the TCustomConnection (see page 6858) descendant, DataSets may include only the active datasets that use the connection component (as is the case with TDatabase and TSQLConnection), or it may list all datasets, regardless of whether they are active (as is the case with TADODatabase and DataSnap connection components TSOAPConnection). If DataSets includes only active datasets, its value changes when datasets are opened and closed.

Note: If a client dataset with an internal provider (for example TSQLClientDataSet or TBDEClientDataSet

) uses the connection component to connect to a database server, DataSets lists the internal dataset that the client dataset creates to access data, not the client dataset itself.

Delphi Examples:

```
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
  RemoteServer1.Connected := False;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].DisableControls;
  end;
end;

procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
```

See Also

DataSetCount (see page 6869)

Active (see page 7004)

Working with Associated Datasets

39.1.15.5.4 TCustomConnection.LoginPrompt Property

Specifies whether a login dialog appears immediately before opening a new connection.

Pascal

```
property LoginPrompt: Boolean;
```

C++

```
__property Boolean LoginPrompt;
```

Description

Set LoginPrompt to true to provide login support when establishing a connection. LoginPrompt controls two things: the occurrence of the OnLogin (see page 6873) event, and the appearance of a default login dialog that prompts users for a name and password when you include DBLogDlg.hpp in your unit (C++) or add DBLogDlg (see page 9775) to your uses clause (Delphi). When the OnLogin (see page 6873) event occurs and when the default login dialog appears depend on the type of connection component:

For TDatabase, TIBDatabase, and TSQLConnection, the dialog appears after the BeforeConnect (see page 6875) event and before the AfterConnect (see page 6874) event, unless you supply an OnLogin (see page 6873) event handler. If there is an OnLogin (see page 6873) event handler, that event occurs in place of the login dialog, and there is no need to include DBLogDlg.hpp in your unit (C++) or add DBLogDlg (see page 9775) to your uses clause (Delphi). If correct values for the user name and password are not supplied in the dialog or by the OnLogin (see page 6873) event handler, the connection fails. The OnLogin (see page 6873) event does not fire unless LoginPrompt is set to true.

For TADOConnection components, the dialog appears after the OnWillConnect event and before the BeforeConnect (see page 6875) event. If there is an OnLogin (see page 6873) event handler, that event occurs after the login dialog. If you do not include DBLogDlg.hpp in your unit (C++) or add DBLogDlg (see page 9775) to your uses clause (Delphi), the OnLogin (see page 6873) event occurs but there is no default login dialog. If correct values for the user name and password are not supplied in the dialog or by the OnLogin (see page 6873) event handler, the connection fails. The OnLogin (see page 6873) event does not fire unless LoginPrompt is set to true.

For DataSnap SOAP connection components, the dialog appears after the OnGetUsername event and before the BeforeConnect (see page 6875), AfterConnect (see page 6874), and OnLogin (see page 6873) events. If the user cancels from the login dialog, no attempt is made to open a connection.

When LoginPrompt is false, the application must supply user name and password values programmatically:

For TDatabase, the user name and password can be supplied as USER NAME and PASSWORD parameters in the Params property.

For TADOConnection, the user name and password can be supplied as the ConnectionString property.

For TSQLConnection, the user name, password, and database can be supplied as UserName, Password, and Database parameters in the Params property, or provided as connection parameters associated with the connection name.

For DataSnap SOAP connection components, there is no built-in use for the user name and password supplied by the login dialog. (The UserName and Password properties of TWebConnection are unrelated).

Warning: Storing hard-coded user name and password entries as property values or in code for an OnLogin (see page 6873) event handler can compromise server security.

See Also

OnLogin (see page 6873)

BeforeConnect (see page 6875)

AfterConnect (see page 6874)

OnGetUserName (see page 8533)

OnLogin (see page 20259)

OnLogin (see page 9994)

Controlling Server Login

39.1.15.5.5 TCustomConnection.OnLogin Property

Occurs when a communication channel opens to the server.

Pascal

```
property OnLogin: TLoginEvent;
```

C++

```
__property TLoginEvent OnLogin;
```

Description

OnLogin occurs when connections are opened if the LoginPrompt (see page 6872) property is true.

For DataSnap SOAP connection components, the following sequence of events occurs when opening a connection to an application server:

1. An OnGetUsername event occurs, where the application can supply an initial user name for the login dialog.
2. The login dialog appears, where the user can supply a user name and password.
3. A BeforeConnect (see page 6875) event occurs, the connection is opened, and an AfterConnect (see page 6874) event occurs.
4. The OnLogin event occurs, where you can write code to log in to the application server using the username and password that were provided in the remote login dialog.

   Note: Do not use the OnLogin event if the connection component provides a different scheme for logging in. For example, TWebConnection uses the Username and Password properties instead.

   For ADO connection components, the following sequence of events occurs when opening a connection to an application server:

1. A BeforeConnect (see page 6875) event occurs.

2. An OnWillConnect event occurs, where the application can supply an initial user name and password for the login dialog, as well as the connection options and status.

3. The login dialog appears, where the user can supply a user name and password.

4. The OnLogin event occurs, where you can write code to respond to the user-supplied values. (For example, you could convert an application-specific user name and password to database user name and password).

5. The connection is opened, and an AfterConnect (see page 6874) event occurs.

Note: Some

C connection components that do not connect to an application server in a multitiered database application replace the OnLogin event defined in TCustomConnection (see page 6858) with a different event that occurs before a connection is established.

See Also

Connected (see page 6868)

Open (see page 6866)

OnGetUserName (see page 8533)

BeforeConnect (see page 6875)

AfterConnect (see page 6874)

LoginPrompt (see page 6872)

OnWillConnect (see page 1039)

39.1.15.5.6 TCustomConnection.StreamedConnected Property

Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.

Pascal

```
property StreamedConnected: Boolean;
```

C++

```
__property Boolean StreamedConnected;
```

Description

StreamedConnected is used internally to ensure that the connection component does not open a connection until all objects on the form or data module are loaded into memory. This property is set instead of the Connected (see page 6868) property when loading the connection component. In the Loaded (see page 6866) method, after all objects in the form or data module have been loaded into memory, the value of StreamedConnected is used to set the Connected (see page 6868) property.

See Also

Connected (see page 6868)

Loaded (see page 6866)

39.1.15.6 TCustomConnection Events

The events of the TCustomConnection class are listed here.

Public Events

	Name	Description
	AfterConnect (see page 6874)	Occurs after a connection is established.
	AfterDisconnect (see page 6875)	Occurs after the connection closes.
	BeforeConnect (see page 6875)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 6875)	Occurs immediately before the connection closes.

39.1.15.6.1 TCustomConnection.AfterConnect Event

Occurs after a connection is established.

Pascal

```
property AfterConnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterConnect;
```

Description

Write an AfterConnect event handler to take application-specific actions immediately after the connection component opens a

connection to the remote source of database information.

See Also

BeforeConnect (see page 6875)

Connected (see page 6868)

Connecting to a Database Server

39.1.15.6.2 TCustomConnection.AfterDisconnect Event

Occurs after the connection closes.

Pascal

```
property AfterDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterDisconnect;
```

Description

Write an AfterDisconnect event handler to take application-specific actions after the connection component drops a connection.

See Also

BeforeDisconnect (see page 6875)

Connected (see page 6868)

Disconnecting from a Database Server

39.1.15.6.3 TCustomConnection.BeforeConnect Event

Occurs immediately before establishing a connection.

Pascal

```
property BeforeConnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeConnect;
```

Description

Write a BeforeConnect event handler to take application-specific actions before the connection component opens a connection to the remote source of database information.

See Also

AfterConnect (see page 6874)

Connected (see page 6868)

Connecting to a Database Server

39.1.15.6.4 TCustomConnection.BeforeDisconnect Event

Occurs immediately before the connection closes.

Pascal

```
property BeforeDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeDisconnect;
```

Description

Write a BeforeDisconnect event handler to take application-specific actions before dropping a connection.

Delphi Examples:

```
{
This BeforeDisconnect event handler makes sure that the
client dataset has applied all its pending updates before
terminating the connection to the server.
}
procedure TForm1.RemoteServer1BeforeDisconnect(Sender: TObject);
begin
  if (ClientDataSet1.ChangeCount > 0) then
    ClientDataSet1.ApplyUpdates(-1);
end;
```

See Also

AfterDisconnect (see page 6875)

Connected (see page 6868)

Disconnecting from a Database Server

39.1.16 TDataLink Class

TDataLink is a helper class used by data-aware objects to coordinate the actions of TDataSource (see page 7070) and TDataSet (see page 6894), and to respond to data events.

Class Hierarchy

Pascal

```
TDataLink = class(TPersistent);
```

C++

```
class TDataLink : public TPersistent;
```

File

DB

Description

Use TDataLink or one of its descendants in any data-aware object that implements a DataSource (see page 6891) property to represent its link to a dataset or that needs to respond to data events.

The constructor of the data-aware object should call the constructor of TDataLink, and initialize any relevant properties. The data-aware object can then link to a TDataSource (see page 7070) by using the DataSource (see page 6891) property of the TDataLink.

Data-aware objects that link to a single field in a dataset should use a TFieldDataLink instead.

See Also

TFieldDataLink (see page 9487)

TDataSet (see page 6894)

TDataSource (see page 7070)

39.1.16.1 TDataLink Members

The following tables list the members exposed by TDataLink.

Protected Methods

	Name	Description
	ActiveChanged (see page 6880)	Responds to changes in the Active (see page 6888) property.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CheckBrowseMode (see page 6880)	Allows the data-aware object to respond just before an event that changes the dataset state.
	DataEvent (see page 6881)	Responds to various events that occur while working with data.
	DataSetChanged (see page 6881)	Responds to changes in the dataset.
	DataSetScrolled (see page 6882)	Allows a response to scrolling the representation of the dataset within the data-aware object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EditingChanged (see page 6883)	Responds to changes in the editing state of the data source.
	FocusControl (see page 6884)	Provides the interface for a method to force focus to the data-aware component associated with the TDataLink (see page 6876).
	GetActiveRecord (see page 6884)	This is GetActiveRecord, a member of class TDataLink.
	GetBOF (see page 6884)	This is GetBOF, a member of class TDataLink.
	GetBufferCount (see page 6884)	This is GetBufferCount, a member of class TDataLink.
	GetEOF (see page 6885)	This is GetEOF, a member of class TDataLink.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRecordCount (see page 6885)	This is GetRecordCount, a member of class TDataLink.
	LayoutChanged (see page 6885)	Responds to changes in the representation of the data by the data-aware object.
	MoveBy (see page 6885)	Repositions the dataset to another record relative to its active record.
	RecordChanged (see page 6886)	Responds to changes in the contents of the current record or field of the dataset.
	SetActiveRecord (see page 6886)	This is SetActiveRecord, a member of class TDataLink.
	SetBufferCount (see page 6887)	This is SetBufferCount, a member of class TDataLink.
	UpdateData (see page 6887)	Provides an interface for writing edits to a record in the dataset.

Protected Properties

	Name	Description
	VisualControl (see page 6894)	Indicates whether a visual control uses the datalink.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6878)	Creates an instance of TDataLink (see page 6876).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

V	Destroy (see page 6879)	Destroys an instance of TDataLink (see page 6876).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6882)	Attempts to drive the data source into an editing state if it is not already in one.
	ExecuteAction (see page 6883)	Invokes an action with the datasource as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	UpdateAction (see page 6887)	Updates an action to reflect the current state of the TDataLink (see page 6876).
	UpdateRecord (see page 6888)	Responds to notifications that edits should be posted to the dataset.

Public Properties

	Name	Description
R	Active (see page 6888)	Specifies whether the dataset that TDataLink (see page 6876) manages is active.
	ActiveRecord (see page 6889)	Specifies the index of the current record within the internal record buffer maintained by the dataset for the Owner of the TDataLink (see page 6876) object.
R	BOF (see page 6889)	Indicates whether the first record in a dataset is active.
	BufferCount (see page 6890)	Specifies the number of records cached by the dataset for the Owner of the TDataLink (see page 6876) object.
R	DataSet (see page 6890)	Specifies the TDataSet (see page 6894) object that represents the data that this TDataLink (see page 6876) is tracking.
	DataSource (see page 6891)	Specifies the TDataSource (see page 7070) object that the Owner of the TDataLink (see page 6876) uses to connect to the dataset.
	DataSourceFixed (see page 6891)	Indicates whether the DataSource (see page 6891) property can be set.
R	Editing (see page 6892)	Indicates whether the DataSource (see page 6891) is in edit mode.
R	Eof (see page 6892)	Indicates whether the last record in a dataset is active.
	ReadOnly (see page 6893)	Indicates whether the dataset for this TDataLink (see page 6876) can be modified.
R	RecordCount (see page 6893)	Indicates the number of records in the internal record buffer maintained by the dataset.

39.1.16.2 TDataLink.Create Constructor

Creates an instance of TDataLink (see page 6876).

Pascal

```
constructor Create;
```

C++

```
__fastcall TDataLink();
```

Description

Create is called from the constructor of any data-aware object that uses a TDataLink (see page 6876) to implement its DataSource (see page 6891) property.

After calling the inherited constructor, Create initializes the BufferCount (see page 6890) property to 1. Data-aware objects that use a TDataLink (see page 6876) object to manage their link to a DataSource (see page 6891) should change the BufferCount (see page 6890) property to the number or records they represent, after calling the inherited constructor.

See Also

BufferCount (see page 6890)

39.1.16.3 TDataLink.Destroy Destructor

Destroys an instance of TDataLink (see page 6876).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataLink();
```

Description

The TDataLink (see page 6876) object should be destroyed in the destructor of its Owner, where that Owner calls Create (see page 6878) from its constructor.

Do not call Destroy directly in an application. Instead, call the Free method. Free checks that the TDataLink (see page 6876) reference is not nil and only then calls Destroy.

Before calling the inherited destructor, Destroy removes any reference to the TDataLink (see page 6876) from the data source object.

See Also

DataSource (see page 6891)

Free (see page 22300)

39.1.16.4 TDataLink Methods

The methods of the TDataLink class are listed here.

Protected Methods

	Name	Description
	ActiveChanged (see page 6880)	Responds to changes in the Active (see page 6888) property.
	CheckBrowseMode (see page 6880)	Allows the data-aware object to respond just before an event that changes the dataset state.
	DataEvent (see page 6881)	Responds to various events that occur while working with data.
	DataSetChanged (see page 6881)	Responds to changes in the dataset.
	DataSetScrolled (see page 6882)	Allows a response to scrolling the representation of the dataset within the data-aware object.
	EditingChanged (see page 6883)	Responds to changes in the editing state of the data source.
	FocusControl (see page 6884)	Provides the interface for a method to force focus to the data-aware component associated with the TDataLink (see page 6876).
	GetActiveRecord (see page 6884)	This is GetActiveRecord, a member of class TDataLink.

	GetBOF (see page 6884)	This is GetBOF, a member of class TDataLink.
	GetBufferCount (see page 6884)	This is GetBufferCount, a member of class TDataLink.
	GetEOF (see page 6885)	This is GetEOF, a member of class TDataLink.
	GetRecordCount (see page 6885)	This is GetRecordCount, a member of class TDataLink.
	LayoutChanged (see page 6885)	Responds to changes in the representation of the data by the data-aware object.
	MoveBy (see page 6885)	Repositions the dataset to another record relative to its active record.
	RecordChanged (see page 6886)	Responds to changes in the contents of the current record or field of the dataset.
	SetActiveRecord (see page 6886)	This is SetActiveRecord, a member of class TDataLink.
	SetBufferCount (see page 6887)	This is SetBufferCount, a member of class TDataLink.
	UpdateData (see page 6887)	Provides an interface for writing edits to a record in the dataset.

Public Methods

	Name	Description
	Edit (see page 6882)	Attempts to drive the data source into an editing state if it is not already in one.
	ExecuteAction (see page 6883)	Invokes an action with the datasource as its target.
	UpdateAction (see page 6887)	Updates an action to reflect the current state of the TDataLink (see page 6876).
	UpdateRecord (see page 6888)	Responds to notifications that edits should be posted to the dataset.

39.1.16.4.1 TDataLink.ActiveChanged Method

Responds to changes in the Active (see page 6888) property.

Pascal

```
procedure ActiveChanged; virtual;
```

C++

```
virtual __fastcall ActiveChanged();
```

Description

The ActiveChanged method defined by TDataLink (see page 6876) merely provides an interface for a method that can respond to changes in the Active (see page 6888) property. Derived objects that do not need to respond to such changes can allow the inherited method to ignore them.

See Also

Active (see page 6888)

39.1.16.4.2 TDataLink.CheckBrowseMode Method

Allows the data-aware object to respond just before an event that changes the dataset state.

Pascal

```
procedure CheckBrowseMode; virtual;
```

C++

```
virtual __fastcall CheckBrowseMode();
```

Description

The CheckBrowseMode method defined by TDataLink (see page 6876) merely provides an interface for a method that can update any state information just before a change to the dataset occurs. Derived objects that do not need to respond to these events can allow the inherited method to ignore them.

See Also

DataSet (see page 6890)

39.1.16.4.3 TDataLink.DataEvent Method

Responds to various events that occur while working with data.

Pascal

```
procedure DataEvent(Event: TDataEvent; Info: Longint); virtual;
```

C++

```
virtual __fastcall DataEvent(TDataEvent Event, Longint Info);
```

Description

DataEvent is called automatically when various data events occur. Event indicates the event that just occurred. Info provides additional context information for some events.

DataEvent dispatches these events by calling the appropriate methods, as indicated in the following table.

Data event	Info	Method called
deFieldChange | deRecordChange	The field that changed	RecordChanged (see page 6886)
deDataSetChange	Current index into the record buffer	DataSetChanged (see page 6881)
deDataSetScroll	Current index into the record buffer	DataSetScrolled (see page 6882)
deLayoutChange	Current index into the record buffer	LayoutChanged (see page 6885)
deUpdateRecord	UpdateRecord (see page 6888)	
deUpdateState	EditingChanged (see page 6883) or ActiveChanged (see page 6880)	
deCheckBrowseMode	CheckBrowseMode (see page 6880)	
deFocusControl	FocusControl (see page 6884)	

See Also

DataSetScrolled (see page 6882)

LayoutChanged (see page 6885)

RecordChanged (see page 6886)

DataSetChanged (see page 6881)

UpdateRecord (see page 6888)

CheckBrowseMode (see page 6880)

FocusControl (see page 6884)

ActiveChanged (see page 6880)

EditingChanged (see page 6883)

39.1.16.4.4 TDataLink.DataSetChanged Method

Responds to changes in the dataset.

Pascal

```
procedure DataSetChanged; virtual;
```

C++

```
virtual __fastcall DataSetChanged();
```

Description

DataSetChanged responds to changes to the contents of the dataset. Anything that changes the contents of the dataset, whether it is editing the data, inserting or deleting records, or changing the key triggers this method. Changes specific to the representation of the data within the data-aware object, such as scrolling the dataset or changing the layout of data elements within the object, also trigger this method. DataSetChanged simply calls RecordChanged (see page 6886). Derived classes can override this procedure to make additional adjustments to changes in the dataset.

See Also

DataSetScrolled (see page 6882)

LayoutChanged (see page 6885)

RecordChanged (see page 6886)

39.1.16.4.5 TDataLink.DataSetScrolled Method

Allows a response to scrolling the representation of the dataset within the data-aware object.

Pascal

```
procedure DataSetScrolled(Distance: Integer); virtual;
```

C++

```
virtual __fastcall DataSetScrolled(int Distance);
```

Description

DataSetScrolled ignores the Distance parameter and calls DataSetChanged (see page 6881). Derived classes can override this procedure to make additional adjustments when the dataset scrolls.

See Also

DataSetChanged (see page 6881)

39.1.16.4.6 TDataLink.Edit Method

Attempts to drive the data source into an editing state if it is not already in one.

Pascal

```
function Edit: Boolean;
```

C++

```
__fastcall Boolean Edit();
```

Description

Use Edit to try to ensure that the current record can be modified. A return value of true indicates that the data source was already in an editing state, or that it was successfully changed to allow editing. A return value of false indicates that the data source could not be changed to allow editing. For example, if the ReadOnly (see page 6893) property is true, Edit will not allow editing and will return false.

See Also

ActiveRecord (see page 6889)

DataSource (see page 6891)

Editing (see page 6892)

ReadOnly (see page 6893)

39.1.16.4.7 TDataLink.EditingChanged Method

Responds to changes in the editing state of the data source.

Pascal

```
procedure EditingChanged; virtual;
```

C++

```
virtual __fastcall EditingChanged();
```

Description

The EditingChanged method defined by TDataLink (see page 6876) merely provides an interface for a method that can respond to changes in the editing state of the data source. EditingChanged is called immediately after the Editing (see page 6892) property has changed its value. Derived objects that do not need to respond to such changes can allow the inherited method to ignore them.

See Also

Edit (see page 6882)

Editing (see page 6892)

39.1.16.4.8 TDataLink.ExecuteAction Method

Invokes an action with the datasource as its target.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; dynamic;
```

C++

```
__fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

ExecuteAction is called automatically when the user invokes an action with the Owner as a target. The Action parameter specifies the action that was invoked. ExecuteAction checks whether the associated datasource can handle the action, and if so, forwards the action to the datasource.

ExecuteAction returns true if the action was successfully dispatched, and false if the TDataLink (see page 6876) could not handle the action.

See Also

UpdateAction (see page 6887)

DataSource (see page 6891)

39.1.16.4.9 TDataLink.FocusControl Method

Provides the interface for a method to force focus to the data-aware component associated with the TDataLink (see page 6876).

Pascal

```
procedure FocusControl(Field: TFieldRef); virtual;
```

C++

```
virtual __fastcall FocusControl(TFieldRef Field);
```

Description

The FocusControl method defined by TDataLink (see page 6876) merely provides an interface for a method that can force the input focus to a data-aware object associated with the TDataLink (see page 6876). Derived objects that do not need this capability can allow the inherited method to ignore the request.

See Also

TabStop

SetFocus

39.1.16.4.10 TDataLink.GetActiveRecord Method

Pascal

```
function GetActiveRecord: Integer; virtual;
```

C++

```
virtual __fastcall int GetActiveRecord();
```

Description

This is GetActiveRecord, a member of class TDataLink.

39.1.16.4.11 TDataLink.GetBOF Method

Pascal

```
function GetBOF: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetBOF();
```

Description

This is GetBOF, a member of class TDataLink.

39.1.16.4.12 TDataLink.GetBufferCount Method

Pascal

```
function GetBufferCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetBufferCount();
```

Description

This is GetBufferCount, a member of class TDataLink.

39.1.16.4.13 TDataLink.GetEOF Method

Pascal

```
function GetEOF: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetEOF();
```

Description

This is GetEOF, a member of class TDataLink.

39.1.16.4.14 TDataLink.GetRecordCount Method

Pascal

```
function GetRecordCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetRecordCount();
```

Description

This is GetRecordCount, a member of class TDataLink.

39.1.16.4.15 TDataLink.LayoutChanged Method

Responds to changes in the representation of the data by the data-aware object.

Pascal

```
procedure LayoutChanged; virtual;
```

C++

```
virtual __fastcall LayoutChanged();
```

Description

The LayoutChanged method simply calls the DataSetChanged (see page 6881) method. LayoutChanged is intended to provide an interface for a method that can respond after changes in the representation of the data by the data-aware object. An example of such a change would be changing the number or order of the columns in a TCustomDBGrid.

See Also

DataSetChanged (see page 6881)

39.1.16.4.16 TDataLink.MoveBy Method

Repositions the dataset to another record relative to its active record.

Pascal

```
function MoveBy(Distance: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int MoveBy(int Distance);
```

Description

The MoveBy record changes the active record in the associated dataset, based on the Distance parameter. If Distance is greater than 0, MoveBy causes the dataset to move forward by that number of records. If Distance is less than 0, MoveBy causes the dataset to move backward. MoveBy returns the number of records actually moved (which may be less than the absolute value of Distance if the dataset encountered the first or last record before moving the indicated number of records).

MoveBy calls the MoveBy method of the associated dataset.

See Also

MoveBy (see page 6971)

DataSet (see page 6890)

39.1.16.4.17 TDataLink.RecordChanged Method

Responds to changes in the contents of the current record or field of the dataset.

Pascal

```
procedure RecordChanged(Field: TField); virtual;
```

C++

```
virtual __fastcall RecordChanged(TField Field);
```

Description

The RecordChanged method defined by TDataLink (see page 6876) merely provides an interface for a method that can respond to changes to the contents of the current record. RecordChanged is called after changes have been posted to the current record in the dataset, and from DataSetChanged (see page 6881).

The Field parameter indicates which field of the current record has changed in value. If Field is nil (Delphi) or NULL (C++), any number of fields within the current record may have changed.

Derived objects that do not need to respond to such changes can allow the inherited method to ignore them.

See Also

ActiveRecord (see page 6889)

DataSetChanged (see page 6881)

OnDataChange (see page 7080)

OnChange (see page 7181)

39.1.16.4.18 TDataLink.SetActiveRecord Method

Pascal

```
procedure SetActiveRecord(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetActiveRecord(int Value);
```

Description

This is SetActiveRecord, a member of class TDataLink.

39.1.16.4.19 TDataLink.SetBufferCount Method

Pascal

```
procedure SetBufferCount(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetBufferCount(int Value);
```

Description

This is SetBufferCount, a member of class TDataLink.

39.1.16.4.20 TDataLink.UpdateAction Method

Updates an action to reflect the current state of the TDataLink (see page 6876).

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; dynamic;
```

C++

```
__fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

UpdateAction is called automatically when the application is idle to give the TDataLink (see page 6876) an opportunity to update associated actions. This allows actions to be grayed, checked, or to otherwise reflect the status of the TDataLink (see page 6876) they target.

The Action parameter specifies the action component that should be updated. UpdateAction returns true if the action component now reflects the state of the component, and false if it did not know how to update the action.

See Also

ExecuteAction (see page 6883)

39.1.16.4.21 TDataLink.UpdateData Method

Provides an interface for writing edits to a record in the dataset.

Pascal

```
procedure UpdateData; virtual;
```

C++

```
virtual __fastcall UpdateData();
```

Description

The UpdateData method defined by TDataLink (see page 6876) merely provides an interface for a method that writes updates to the current record in the dataset while avoiding calls to RecordChanged (see page 6886) that posting the changes may trigger. Derived objects that do not need to write edits to the dataset can allow the inherited method to do nothing.

See Also

RecordChanged (see page 6886)

OnUpdateData (see page 7082)

UpdateRecord (see page 6888)

39.1.16.4.22 TDataLink.UpdateRecord Method

Responds to notifications that edits should be posted to the dataset.

Pascal

```
procedure UpdateRecord;
```

C++

```
__fastcall UpdateRecord();
```

Description

UpdateRecord calls UpdateData (see page 6887) after guarding against calls to RecordChanged (see page 6886), which may arise while UpdateData (see page 6887) is executing.

See Also

RecordChanged (see page 6886)

OnUpdateRecord (see page 9957)

UpdateData (see page 6887)

39.1.16.5 TDataLink Properties

The properties of the TDataLink class are listed here.

Protected Properties

	Name	Description
	VisualControl (see page 6894)	Indicates whether a visual control uses the datalink.

Public Properties

	Name	Description
R	Active (see page 6888)	Specifies whether the dataset that TDataLink (see page 6876) manages is active.
	ActiveRecord (see page 6889)	Specifies the index of the current record within the internal record buffer maintained by the dataset for the Owner of the TDataLink (see page 6876) object.
R	BOF (see page 6889)	Indicates whether the first record in a dataset is active.
	BufferCount (see page 6890)	Specifies the number of records cached by the dataset for the Owner of the TDataLink (see page 6876) object.
R	DataSet (see page 6890)	Specifies the TDataSet (see page 6894) object that represents the data that this TDataLink (see page 6876) is tracking.
	DataSource (see page 6891)	Specifies the TDataSource (see page 7070) object that the Owner of the TDataLink (see page 6876) uses to connect to the dataset.
	DataSourceFixed (see page 6891)	Indicates whether the DataSource (see page 6891) property can be set.
R	Editing (see page 6892)	Indicates whether the DataSource (see page 6891) is in edit mode.
R	Eof (see page 6892)	Indicates whether the last record in a dataset is active.
	ReadOnly (see page 6893)	Indicates whether the dataset for this TDataLink (see page 6876) can be modified.
R	RecordCount (see page 6893)	Indicates the number of records in the internal record buffer maintained by the dataset.

39.1.16.5.1 TDataLink.Active Property

Specifies whether the dataset that TDataLink (see page 6876) manages is active.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Read Active to determine whether the dataset for this TDataLink (see page 6876) is Active. When Active is false, the dataset is closed, and the dataset cannot read data from or write data to the database. When Active is true, data can be read from and written to the dataset.

See Also

ActiveChanged (see page 6880)

DataSet (see page 6890)

Active (see page 7004)

39.1.16.5.2 TDataLink.ActiveRecord Property

Specifies the index of the current record within the internal record buffer maintained by the dataset for the Owner of the TDataLink (see page 6876) object.

Pascal

```
property ActiveRecord: Integer;
```

C++

```
__property int ActiveRecord;
```

Description

Use ActiveRecord to discover or set the current record in the set of one or more records managed by the dataset. The set of records managed by the dataset corresponds to the number of records from the dataset visible at one time. For example, when the TDataLink (see page 6876) object is owned by a data-aware grid, the set of records managed by the dataset corresponds to the number of rows shown by the grid, and the ActiveRecord represents the current row.

Note: Unidirectional datasets do not buffer records. When the data link is used with a unidirectional dataset, ActiveRecord is always 0.

See Also

BufferCount (see page 6890)

DataSet (see page 6890)

DataSource (see page 6891)

39.1.16.5.3 TDataLink.BOF Property

Indicates whether the first record in a dataset is active.

Pascal

```
property BOF: Boolean;
```

C++

```
__property Boolean BOF;
```

Description

Inspect BOF to determine if the dataset component associated with the TDataLink (see page 6876) is positioned on its first record. If BOF is true, the active record is unequivocally the first row in the dataset. BOF is also true when:

A dataset is first opened.

A call is made to the dataset's First method.

A call is made to the dataset's Prior method, and the method fails because the first row is already active.

A call is made to the dataset's SetRange, resulting in an empty result set.

BOF is false in all other cases. An application should assume BOF is false unless one of the conditions above is met and the application tests the property directly.

Tip: If both Eof (see page 6892) and BOF are true, a dataset or range is empty.

See Also

Bof (see page 7008)

Eof (see page 6892)

39.1.16.5.4 TDataLink.BufferCount Property

Specifies the number of records cached by the dataset for the Owner of the TDataLink (see page 6876) object.

Pascal

```
property BufferCount: Integer;
```

C++

```
__property int BufferCount;
```

Description

Set BufferCount to tell the dataset how many records to buffer together to service the owner of the TDataLink (see page 6876) object. For example, a data-aware grid sets BufferCount to the number of rows visible in the grid.

Read BufferCount to iterate through all the records buffered by the dataset. For example, a data-aware object can paint itself by cycling through the set of records, moving the ActiveRecord (see page 6889) to visit each record in turn.

Note: Unidirectional datasets do not buffer records. Do not use BufferCount when the datalink is associated with a unidirectional dataset.

See Also

ActiveRecord (see page 6889)

DataSet (see page 6890)

DataSource (see page 6891)

RecordCount (see page 6893)

39.1.16.5.5 TDataLink.DataSet Property

Specifies the TDataSet (see page 6894) object that represents the data that this TDataLink (see page 6876) is tracking.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Read DataSet to determine the dataset that this TDataLink (see page 6876) helps manage. The TDataLink (see page 6876) responds to data events that are received by the DataSet, and makes changes to the dataset on behalf of the object that owns the TDataLink (see page 6876) object.

See Also

ActiveChanged (see page 6880)

ActiveRecord (see page 6889)

BufferCount (see page 6890)

CheckBrowseMode (see page 6880)

DataSetChanged (see page 6881)

DataSetScrolled (see page 6882)

EditingChanged (see page 6883)

RecordChanged (see page 6886)

RecordCount (see page 6893)

UpdateData (see page 6887)

UpdateRecord (see page 6888)

39.1.16.5.6 TDataLink.DataSource Property

Specifies the TDataSource (see page 7070) object that the Owner of the TDataLink (see page 6876) uses to connect to the dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Use DataSource to determine or set the TDataSource (see page 7070) object that connects to the dataset. TDataLink (see page 6876) responds to data events that the DataSource receives.

Note: To implement the DataSource property of a data-aware object which owns a TDataLink (see page 6876) or its descendant, create a method that gets or sets the DataSource property of the TDataLink (see page 6876) derived member.

See Also

DataSourceFixed (see page 6891)

39.1.16.5.7 TDataLink.DataSourceFixed Property

Indicates whether the DataSource (see page 6891) property can be set.

Pascal

```
property DataSourceFixed: Boolean;
```

C++

```
__property Boolean DataSourceFixed;
```

Description

Use DataSourceFixed to determine whether the DataSource (see page 6891) property can be changed. By default, DataSourceFixed is false, allowing the DataSource (see page 6891) property to be changed.

When there is a chance that an asynchronous event might change the DataSource (see page 6891) before an operation completes, set DataSourceFixed to true before performing the operation to ensure that the DataSource (see page 6891) is constant throughout the operation. After the data operation is complete, DataSourceFixed can be set back to false.

Set DataSourceFixed to prevent the DataSource (see page 6891) of a data-aware object that owns the TDataLink (see page 6876) from being changed once it is set.

See Also

DataSource (see page 6891)

39.1.16.5.8 TDataLink.Editing Property

Indicates whether the DataSource (see page 6891) is in edit mode.

Pascal

```
property Editing: Boolean;
```

C++

```
__property Boolean Editing;
```

Description

Read Editing to determine whether the DataSource (see page 6891) is inserting a new record into the dataset, or modifying the current record. To change the dataset into edit mode, and hence the Editing property to true, use the Edit (see page 6882) method.

See Also

Edit (see page 6882)

EditingChanged (see page 6883)

Edit (see page 7076)

39.1.16.5.9 TDataLink.Eof Property

Indicates whether the last record in a dataset is active.

Pascal

```
property Eof: Boolean;
```

C++

```
__property Boolean Eof;
```

Description

Inspect Eof to determine if the dataset component associated with the TDataLink (see page 6876) is positioned on its last record. If Eof is true, the active record is unequivocally the last row in the dataset. Eof is true when:

The associated dataset is empty.

A call is made to the dataset's Last method and the dataset is not unidirectional.

A call is made to the dataset's Next method, and the method fails because the last row is already active.

A call to the dataset's SetRange method is made resulting in an empty result set.

Eof is false in all other cases. An application should assume Eof is false unless one of the conditions above is met and the application tests the property directly.

Tip: If both Eof and BOF (see page 6889) are true, a dataset or range is empty.

See Also

Eof (see page 7015)

BOF (see page 6889)

39.1.16.5.10 TDataLink.ReadOnly Property

Indicates whether the dataset for this TDataLink (see page 6876) can be modified.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to determine whether the dataset for this TDataLink (see page 6876) can be modified. Setting ReadOnly to true causes any subsequent calls to the Edit (see page 6882) method to fail, and changes the DataSource (see page 6891) out of an editing state if it is in one. Setting ReadOnly to false enables changes to the data in the dataset.

See Also

Edit (see page 6882)

Editing (see page 6892)

39.1.16.5.11 TDataLink.RecordCount Property

Indicates the number of records in the internal record buffer maintained by the dataset.

Pascal

```
property RecordCount: Integer;
```

C++

```
__property int RecordCount;
```

Description

Read RecordCount to determine the number of actual records in the buffer maintained by the dataset for the Owner of this TDataLink (see page 6876).

In most cases, RecordCount is the same as BufferCount (see page 6890). RecordCount indicates the number of records actually in the record buffer of the dataset, as opposed to the number of records that the dataset can possibly maintain. For example, a data-aware grid with four rows could have one record actually showing data. In this case, BufferCount (see page 6890) would be 4, while RecordCount would be 1.

See Also

BufferCount (see page 6890)

DataSet (see page 6890)

39.1.16.5.12 TDataLink.VisualControl Property

Indicates whether a visual control uses the datalink.

Pascal

```
property VisualControl: Boolean;
```

C++

```
__property Boolean VisualControl;
```

Description

Descendants of TDataLink (see page 6876) set VisualControl to indicate whether the control that uses the link is a visual control. A value of true indicates a visual control.

See Also

TfieldDataLink

TNavDataLink (see page 9513)

TDataSourceLink (see page 8577)

TListSourceLink (see page 9498)

39.1.17 TDataSet Class

TDataSet is the base class for all dataset components that represent data in rows and columns.

Class Hierarchy

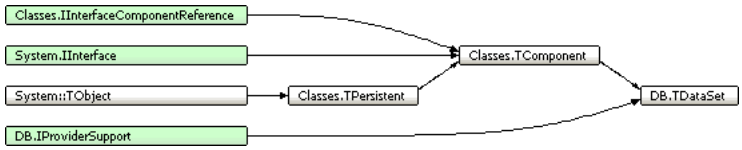

Pascal

```
TDataSet = class(TComponent, IProviderSupport);
```

C++

```
class TDataSet : public TComponent, public IProviderSupport;
```

File

DB

Description

TDataSet introduces the basic properties, events, and methods for working with data.

Many of these properties, events, and methods are abstract (Delphi) or pure virtual (C++) in TDataSet. Abstract or pure virtual declarations are declarations without implementations. At the TDataSet level they cannot be used or accessed. Developers must

use or derive descendants of TDataSet that redeclare and implement these abstract or pure virtual methods. Many of the other TDataSet methods are declared and implemented in TDataSet as virtual methods, but the implementations are merely stubs that are reimplemented in descendants.

TDataSet has two several descendants: TBDEDataSet, TCustomADONETConnector, TIBCustomDataSet, TCustomSQLDataSet, TCustomClientDataSet and TWideDataSet (see page 7526).

TBDEDataSet is the base class for datasets that access their data using the Borland (see page 1761) Database Engine (BDE). TBDEDataSet descendants include TTable, TQuery, and TStoredProc. Developers who create custom dataset components that use the BDE derive them from TBDEDataSet, TDBDataSet, TQuery, TStoredProc, or TTable.

TCustomADONETConnector is the base class for datasets that access their data using ActiveX Data Objects (ADO). TCustomADONETConnector descendants include TADODataSet, TADOTable, TADOQuery, and TADOStoredProc. Developers who create custom dataset components that use ADO derive them from CustomADODataSet.

TIBCustomDataSet is the base class for datasets that directly access the data in InterBase tables. TIBCustomDataSet descendants include TIBDataSet, TIBTable, TIBQuery, and TIBStoredProc. Developers who create custom dataset components that directly access data in an InterBase database derive from TIBCustomDataSet.

TCustomSQLDataSet is the base class for unidirectional datasets. Unidirectional datasets are read-only datasets that permit only forward navigation. TCustomSQLDataSet descendants include the dbExpress datasets TSQLDataSet, TSQLQuery, TSQLTable, and TSQLStoredProc. Developers who create custom dataset components that use dbExpress to access their data derive from TCustomSQLDataSet.

TCustomClientDataSet is the base class for in-memory datasets. Client datasets can work with data from files on disk or with data provided by another component via a provider. They cache that data in memory, maintain a record of any changes in a change log, and apply cached updates at a later point back to the source of the data. Developers who create custom datasets that store their data in an in-memory cache derive from TCustomClientDataSet.

TWideDataSet (see page 7526) provides dataset support for wide strings.

Developer's can also derive custom dataset components directly from TDataSet, providing their own mechanisms for accessing and manipulating the data.

See Also

TClientDataSet (see page 8289)

TWideDataSet (see page 7526)

TSQLTable (see page 20337)

TSQLQuery (see page 20298)

TSQLStoredProc (see page 20318)

TSQLDataSet (see page 20265)

TBDEDataSet (see page 9878)

TDBDataSet (see page 10000)

TQuery (see page 10059)

TStoredProc (see page 10139)

TTable (see page 10168)

TADODataSet (see page 1041)

TADOTable (see page 1125)

TADOQuery (see page 1084)

Marking and Returning to Records

Understanding Datasets: Overview

Distributing Database Applications

Connecting to Databases with TDatabase

39.1.17.1 TDataSet Members

The following tables list the members exposed by TDataSet.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
V	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
V	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
V	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
V	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
V	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
V	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
V	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
V	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
V	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
V	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
V	DataEvent (see page 6924)	This is DataEvent, a member of class TDataSet.
V	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
V	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
V	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
V	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
V	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
V	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
V	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
V	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
V	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.

	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.
	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.

	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	OpenCursor (see page 6976)	This is OpenCursor, a member of class TDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 6982)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandType (see page 6982)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.

	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
V	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
V	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
V	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
V	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
V	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
V	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
V	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
V	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
V	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
V	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
V	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
V	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
V	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.

	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
	Reserved (see page 7035)	For internal use only.

Public Events

	Name	Description
	AfterCancel (see page 7038)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 7040)	Occurs after an application closes a dataset.
	AfterDelete (see page 7041)	Occurs after an application deletes a record.
	AfterEdit (see page 7044)	Occurs after an application starts editing a record.
	AfterInsert (see page 7044)	Occurs after an application inserts a new record.
	AfterOpen (see page 7045)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 7046)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 7047)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 7047)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 7048)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 7048)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 7049)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 7049)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 7050)	Occurs before an application enters insert mode.
	BeforeOpen (see page 7051)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 7052)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 7052)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 7053)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 7053)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 7054)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 7054)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 7056)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 7056)	Occurs when an application attempts to modify or insert a record and an exception is raised.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.

	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Close (see page 6920)	Closes a dataset.
	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
	Create (see page 6904)	Creates an instance of a TDataSet (see page 6894) component.
	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
	Destroy (see page 6904)	Destroys the TDataSet (see page 6894) instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.

V	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
V	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
V	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	Last (see page 6968)	Moves to the last record in the dataset.
V	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	Open (see page 6976)	Opens the dataset.
V	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.

Public Properties

	Name	Description
	Active (see page 7004)	Specifies whether or not a dataset is open.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
R	DataSource (see page 7014)	Represents the data source of another dataset that supplies values to the dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
	FieldDefs (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	Modified (see page 7029)	Indicates whether the active record is modified.
	ObjectView (see page 7031)	Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

39.1.17.2 TDataSet.Create Constructor

Creates an instance of a TDataSet (see page 6894) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDataSet(TComponent * AOwner);
```

Description

Call Create to instantiate a dataset component at runtime. TDataSet (see page 6894) contains abstract methods, and is not intended to be used in applications. Instead, applications instantiate dataset descendants such as TClientDataSet, TADODataSet, TTable, TQuery, TStoredProc, or TSQLDataSet.

AOwner is the component, typically a form or data module, that is responsible for freeing the dataset. It becomes the value of the Owner property.

Create creates the auxiliary objects such as empty field and field definition lists and initializes default property values (such as setting AutoCalcFields (see page 7006) to true).

See Also

AutoCalcFields (see page 7006)

Destroy (see page 6904)

Owner (see page 2807)

39.1.17.3 TDataSet.Destroy Destructor

Destroys the TDataSet (see page 6894) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataSet();
```

Description

Do not call Destroy directly in an application. Usually destruction of datasets is handled automatically by Delphi. If an application creates its own instances of a dataset, however, and does not assign an Owner that is responsible for freeing the dataset, then the application should call Free, which checks that the dataset reference is not nil before calling Destroy.

See Also

Designer (see page 7015)

FieldDefs (see page 7019)

Create (see page 6904)

Free (see page 22300)

Owner (see page 2807)

39.1.17.4 TDataSet Methods

The methods of the TDataSet class are listed here.

Protected Methods

	Name	Description
	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
	DataEvent (see page 6924)	This is DataEvent, a member of class TDataSet.
	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.

V	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
V	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
V	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
V	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
V	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
V	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
V	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
V	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
V	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
V	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
V	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
V	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
V	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
V	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
V	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
V	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
V	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
A	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
V	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
V	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
V	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
V	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
V	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
V	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
V	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
A	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.
V	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
V	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
V	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
V	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
A	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
A	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
V	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
V	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
V	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
A	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
V	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
V	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
V	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
A	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
V	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
V	OpenCursor (see page 6976)	This is OpenCursor, a member of class TDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
V	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
V	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
V	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
V	PSGetCommandText (see page 6982)	Gets the SQL command to be executed by PSExecute.
V	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
V	PSGetCommandType (see page 6982)	Gets the command type to be executed by PSExecute.
V	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.

	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.

Public Methods

	Name	Description
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.

	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
V	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
V	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Close (see page 6920)	Closes a dataset.
V	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
V	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	FieldByName (see page 6936)	Finds a field based on its name.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
V	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
V	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
V	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
V	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
V	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
V	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
V	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
V	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
V	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	Last (see page 6968)	Moves to the last record in the dataset.
V	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
	Next (see page 6973)	Moves to the next record in the dataset.
	Open (see page 6976)	Opens the dataset.
V	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
	Prior (see page 6979)	Moves to the previous record in the dataset.

	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
	SetFields (see page 6997)	Sets the values for all fields in a record
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.

39.1.17.4.1 TDataSet.ActivateBuffers Method

Pascal

```
procedure ActivateBuffers; virtual;
```

C++

```
virtual __fastcall ActivateBuffers();
```

Description

This is ActivateBuffers, a member of class TDataSet.

39.1.17.4.2 TDataSet.ActiveBuffer Method

Returns a pointer to the buffer for the active record.

Pascal

```
function ActiveBuffer: PChar;
```

C++

```
__fastcall const char * ActiveBuffer();
```

Description

ActiveBuffer is used internally by many dataset methods to ensure that the active buffer points to the buffer for the active record. If an application uses existing dataset methods, the active buffer is always set correctly, so there is usually no need to call ActiveBuffer directly.

Note: Unidirectional datasets do not buffer records. On Unidirectional datasets, ActiveBuffer returns a meaningless value.

ActiveBuffer is used by bookmarking methods to index into the active record buffer to retrieve bookmark information.

Applications that provide custom dataset routines may need to call ActiveBuffer to access the buffer data.

See Also

RecordSize (see page 7035)

Next (see page 6973)

Prior (see page 6979)

39.1.17.4.3 TDataSet.AllocRecordBuffer Method

Pascal

```
function AllocRecordBuffer: PChar; virtual;
```

C++

```
virtual __fastcall const char * AllocRecordBuffer();
```

Description

This is AllocRecordBuffer, a member of class TDataSet.

39.1.17.4.4 TDataSet.Append Method

Adds a new, empty record to the end of the dataset.

Pascal

```
procedure Append;
```

C++

```
__fastcall Append();
```

Description

For datasets that permit editing, call Append to:

1Open (see page 6976) a new, empty record at the end of the dataset.

2Set the active record to the new record.

After a call to Append, an application can enable users to enter data in the fields of the record, and can then post those changes to the database or change log using Post (see page 6977) (Client datasets can subsequently apply updates in the change log back to the database by calling ApplyUpdates.) A newly appended record is posted to the database in one of three ways:

For indexed Paradox and dBASE tables, the record is inserted into the dataset in a position based on its index.

For unindexed Paradox and dBASE tables, the record is added to the end of the dataset.

For SQL databases, the physical location of the appended record is implementation-specific. For indexed tables, the index is updated with the new record information.

After the new record is applied back to the database server, its physical location is database-specific. For indexed tables, the index is updated with the new record information.

Delphi Examples:

```
{
This example appends a new record to a table or client
dataset when the user clicks a button. The two fields
Field1 and Field2 are filled from the contents of two edit
controls.  This example requires a populated TTable or
TDataSet with the appropriate field values.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Customers.Append;
  Customers.FieldValues['Field1'] := StrToInt(Edit1.text);
  Customers.FieldValues['Field2'] := Edit2.text;
  Customers.Post;
end;
```

See Also

AppendRecord (see page 6911)

ApplyUpdates (see page 8394)

Insert (see page 6959)

Post (see page 6977)

CanModify (see page 7012)

Adding New Records

Posting Data

39.1.17.4.5 TDataSet.AppendRecord Method

Adds a new, populated record to the end of the dataset and posts it.

Pascal

```
procedure AppendRecord(const Values: array of const);
```

C++

```
__fastcall AppendRecord(const array of const Values);
```

Description

Call AppendRecord to create a new, empty record at the end of the dataset, populate it with the field values in Values, and post the values to the database or change log.

In C++, the Values parameter is an array of field values. The Values_Size parameter is the index of the last value (one less than the total number of values).

A newly appended record is posted to the database in one of three ways:

For indexed Paradox and dBASE tables, the record is inserted into the dataset in a position based on its index.

For unindexed Paradox and dBASE tables, the record is added to the end of the dataset.

For SQL databases, the physical location of the append is implementation-specific. For indexed tables, the index is updated with the new record information.

The newly appended record becomes the active record.

C++ Examples:

```
/*
Reads through all records in the Customers table.
Updates the ProgressBar accordingly.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ProgressBar1->Min = 0;
  ProgressBar1->Max = Customers->RecordCount;
  Customers->First();
  for (int i = ProgressBar1->Min; i <= ProgressBar1->Max; i++)
  {
    ProgressBar1->Position = i;
    Customers->Next();
    // do something with the current record
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Customers = new TTable(Form1);
  Customers->Active = false; // The Table component must not be active
  Customers->DatabaseName = "DBDEMOS";
  Customers->TableType = ttParadox;
  Customers->TableName = "CustInfo";
  Customers->Active = False;
  if (Customers->Exists) // Don't overwrite an existing table
  {
    Customers->Close();
    Customers->DeleteTable();
```

```
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Customers->FieldDefs->Clear();
  TFieldDef *pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Customers->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Customers->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Customers->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Customers->CreateTable();
  Customers->Active = True;
  for (int i = 1; i <= 20; i++)
    Customers->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  DS2->DataSet = Customers;
  DBGrid2->DataSource->DataSet = Customers;
  Customers->Active = True;
}
```

Delphi Examples:

```
{
This statement appends a record to the MyCustInfo data. Note
that Nulls are entered for some of the values, but are not
required for missing values at the end of the array argument,
i.e. after the Field3 field.  InsertRecord behaves differently
depending on the indexing of the table.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Customers.InsertRecord([Edit1.Text, Edit2.Text, Null, Edit3.Text]);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Customers:= TTable.Create(Form1);
  with Customers do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'MyCustInfo';
    ReadOnly:= False;

//    Don't overwrite an existing table
    if (not Customers.Exists) then CreateMyTable
    else
    begin
      if (Customers.Exists AND
        (MessageDlg('MyCustInfo table already exists.  Do you want to rebuild it?',
mtConfirmation, [mbYes, mbNo], 0) = mrYes)) then
      begin
        Customers.Close;
        Customers.DeleteTable;
        CreateMyTable;
      end;
    end;
  end;
```

```
    DS2.DataSet:= Customers;
    DBGrid2.DataSource.DataSet:= Customers;
    Customers.Active:= True;
  end;

  procedure CreateMyTable();
  var
    i: Integer;
  begin
  //      The Table component must not be active
  //      First, describe the type of table and give
  //      it a name
  //      Next, describe the fields in the table
    with Customers do
    begin
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
          with AddFieldDef do
          begin
            Name := 'Field3';
            DataType := ftString;
            Size := 30;
          end;
          with AddFieldDef do
          begin
            Name := 'Field4';
            DataType := ftString;
            Size := 30;
          end;
        end;

  //    Take out this part to remove indexing
  //    Next, describe any indexes
        with IndexDefs do
        begin
          Clear;
  //        The 1st index has no name because it is
  //        a Paradox primary key
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;

  //      Call the CreateTable method to create the table
```

```
      CreateTable;
      Customers.Active:= True;
      for i := 1 to 20 do
        Customers.AppendRecord([i*111, i*222, i*333, i*444]);
    end;
end;
{
Reads through all records in the Customers table.
Updates the ProgressBar accordingly.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
begin
  with ProgressBar1 do
  begin
    Min := 0;
    Max := Customers.RecordCount;
    Customers.First;
    for i := Min to Max do
    begin
      Position := i;
      Customers.Next;
      // do something with the current record
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Customers:= TTable.Create(Form1);
  with Customers do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Active := False;
//  Overwrite any existing table
    if Customers.Exists then
    begin
      Customers.Close;
      Customers.DeleteTable;
    end;
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
```

```
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Customers.Active:= True;
        for i := 1 to 100 do
          Customers.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Customers;
    DBGrid2.DataSource.DataSet:= Customers;
    Customers.Active:= True;
  end;
```

See Also

Append (see page 6910)

InsertRecord (see page 6961)

Post (see page 6977)

SetFields (see page 6997)

Modifying Entire Records

39.1.17.4.6 TDataSet.BindFields Method

Pascal

```
procedure BindFields(Binding: Boolean);
```

C++

```
__fastcall BindFields(Boolean Binding);
```

Description

This is BindFields, a member of class TDataSet.

39.1.17.4.7 TDataSet.BlockReadNext Method

Pascal

```
procedure BlockReadNext; virtual;
```

C++

```
virtual __fastcall BlockReadNext();
```

Description

This is BlockReadNext, a member of class TDataSet.

39.1.17.4.8 TDataSet.BookmarkAvailable Method

Pascal

```
function BookmarkAvailable: Boolean;
```

C++

```
__fastcall Boolean BookmarkAvailable();
```

Description

This is BookmarkAvailable, a member of class TDataSet.

39.1.17.4.9 TDataSet.BookmarkValid Method

Indicates the validity of a specified bookmark.

Pascal

```
function BookmarkValid(Bookmark: TBookmark): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean BookmarkValid(TBookmark Bookmark);
```

Description

This function returns false in all cases, indicating that any bookmark is invalid because the TDataSet (see page 6894) class does not implement support for bookmarks.

Descendant classes that implement bookmark support override BookmarkValid to test bookmarks for validity and indicate when a bookmark is valid by returning true.

See Also

CompareBookmarks (see page 6921)

TTable (see page 10168)

Marking and Returning to Records

39.1.17.4.10 TDataSet.CalculateFields Method

Pascal

```
procedure CalculateFields(Buffer: PChar); virtual;
```

C++

```
virtual __fastcall CalculateFields(const char * Buffer);
```

Description

This is CalculateFields, a member of class TDataSet.

39.1.17.4.11 TDataSet.Cancel Method

Cancels modifications to the active record if those changes are not yet posted.

Pascal

```
procedure Cancel; virtual;
```

C++

```
virtual __fastcall Cancel();
```

Description

Call Cancel to undo modifications made to one or more fields belonging to the active record. As long as those changes are not already posted, Cancel returns the record to its previous state, and sets the dataset state to dsBrowse.

Typically Cancel is used to back out of changes in response to user request, or in field validation routines that back out illegal field values. The TDBNavigator object contains a Cancel button that calls Cancel.

Note: If the dataset is not in an editing state (dsEdit or dsInsert), Cancel does nothing.

Delphi Examples:

```
{
The following fragment prompts the user to confirm changes
to a record; if the user clicks Yes, the record is posted to
the table, otherwise the changes are cancelled.
}
procedure TForm1.Button2Click(Sender: TObject);
begin
if MessageDlg('Update Record?', mtConfirmation, [mbYes, mbNo], 0) =
              mrYes then
  Table1.Post
else
  Table1.Cancel;
end;
```

See Also

State (see page 7036)

Post (see page 6977)

TDBNavigator (see page 9230)

Canceling Changes

Editing Records

Connecting to Databases with TDatabase

39.1.17.4.12 TDataSet.CheckActive Method

Pascal

```
procedure CheckActive; virtual;
```

C++

```
virtual __fastcall CheckActive();
```

Description

This is CheckActive, a member of class TDataSet.

39.1.17.4.13 TDataSet.CheckBiDirectional Method

Pascal

```
procedure CheckBiDirectional;
```

C++

```
__fastcall CheckBiDirectional();
```

Description

This is CheckBiDirectional, a member of class TDataSet.

39.1.17.4.14 TDataSet.CheckBrowseMode Method

Automatically posts or cancels data changes when the active record changes.

Pascal

```
procedure CheckBrowseMode;
```

C++

```
__fastcall CheckBrowseMode();
```

Description

CheckBrowseMode is used internally by many dataset methods to ensure that modifications to the active record are posted when a dataset's state is dsEdit, dsInsert, or dsSetKey and a method switches to a different record.

If State (see page 7036) is dsEdit or dsInsert, CheckBrowseMode calls UpdateRecord (see page 7002), and, if the Modified (see page 7029) property for the dataset is true, calls Post (see page 6977). If Modified (see page 7029) is false, CheckBrowseMode calls Cancel (see page 6916).

If State (see page 7036) is dsSetKey, CheckBrowseMode calls Post (see page 6977).

If State (see page 7036) is dsInactive, CheckBrowseMode raises an exception.

If an application uses existing dataset methods, CheckBrowseMode is always called when necessary, so there is usually no need to call CheckBrowseMode directly.

Applications that provide custom dataset routines may need to call CheckBrowseMode inside those routines to guarantee that changes are posted when switching to a different record.

See Also

State (see page 7036)

Cancel (see page 6916)

Edit (see page 6932)

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Post (see page 6977)

Prior (see page 6979)

SetKey (see page 10208)

SetKey (see page 8449)

39.1.17.4.15 TDataSet.CheckFieldCompatibility Method

Pascal

```
procedure CheckFieldCompatibility(Field: TField; FieldDef: TFieldDef); virtual;
```

C++

```
virtual __fastcall CheckFieldCompatibility(TField Field, TFieldDef FieldDef);
```

Description

This is CheckFieldCompatibility, a member of class TDataSet.

39.1.17.4.16 TDataSet.CheckInactive Method

Pascal

```
procedure CheckInactive; virtual;
```

C++

```
virtual __fastcall CheckInactive();
```

Description

This is CheckInactive, a member of class TDataSet.

39.1.17.4.17 TDataSet.ClearBuffers Method

Pascal

```
procedure ClearBuffers; virtual;
```

C++

```
virtual __fastcall ClearBuffers();
```

Description

This is ClearBuffers, a member of class TDataSet.

39.1.17.4.18 TDataSet.ClearCalcFields Method

Pascal

```
procedure ClearCalcFields(Buffer: PChar); virtual;
```

C++

```
virtual __fastcall ClearCalcFields(const char * Buffer);
```

Description

This is ClearCalcFields, a member of class TDataSet.

39.1.17.4.19 TDataSet.ClearFields Method

Clears the contents of all fields for the active record.

Pascal

```
procedure ClearFields;
```

C++

```
__fastcall ClearFields();
```

Description

Call ClearFields to erase the current contents of all fields in the active record. If the dataset is not in either dsInsert or dsEdit state, ClearFields raises an exception.

If a SetKey operation is not under way, ClearFields recalculates all calculated fields, and generates an OnDataChange event on the data source component associated with the dataset.

See Also

CheckBrowseMode (see page 6918)

SetKey (see page 10208)

OnDataChange (see page 7080)

SetKey (see page 8449)

39.1.17.4.20 TDataSet.Close Method

Closes a dataset.

Pascal

```
procedure Close;
```

C++

```
__fastcall Close();
```

Description

Call Close to set the Active (see page 7004) property of a dataset to false. When Active (see page 7004) is false, the dataset is closed; it cannot read or write data and data-aware controls can't use it to fetch data or post edits.

An application must close the dataset before changing properties that affect the status of the database or the controls that display data in an application. For example, to change the DataSource (see page 7014) property for a dataset, the dataset must be closed. Closing the dataset puts it into the dsInactive state.

See Also

Active (see page 7004)

DataSource (see page 7014)

State (see page 7036)

AfterClose (see page 7040)

BeforeClose (see page 7048)

Open (see page 6976)

Opening and Closing Datasets

Posting Data

39.1.17.4.21 TDataSet.CloseBlob Method

Pascal

```
procedure CloseBlob(Field: TField); virtual;
```

C++

```
virtual __fastcall CloseBlob(TField Field);
```

Description

This is CloseBlob, a member of class TDataSet.

39.1.17.4.22 TDataSet.CloseCursor Method

Pascal

```
procedure CloseCursor; virtual;
```

C++

```
virtual __fastcall CloseCursor();
```

Description

This is CloseCursor, a member of class TDataSet.

39.1.17.4.23 TDataSet.CompareBookmarks Method

Implements a virtual function to compare two bookmarks.

Pascal

```
function CompareBookmarks(Bookmark1: TBookmark; Bookmark2: TBookmark): Integer; virtual;
```

C++

```
virtual __fastcall int CompareBookmarks(TBookmark Bookmark1, TBookmark Bookmark2);
```

Description

As implemented in TDataSet (see page 6894), CompareBookmarks always returns 0, indicating no difference between the bookmarks. This is because TDataSet (see page 6894) does not implement support for bookmarks.

Descendant classes that provide bookmark support override this method to return a value less than 0 if Bookmark1 is less than Bookmark2, 0 if the bookmarks are identical, and a value greater than 0 if Bookmark1 is greater than Bookmark2.

See Also

BookmarkValid (see page 6916)

TTable (see page 10168)

TClientDataSet (see page 8289)

Marking and Returning to Records

39.1.17.4.24 TDataSet.ControlsDisabled Method

Indicates whether data-aware controls update their display to reflect changes to the dataset.

Pascal

```
function ControlsDisabled: Boolean;
```

C++

```
__fastcall Boolean ControlsDisabled();
```

Description

Call ControlsDisabled to ascertain whether the updating of data display in data-aware controls is currently disabled. If ControlsDisabled is true, controls are currently disabled. ControlsDisabled is true as long as the reference count that keeps track of disabling for the dataset is greater than zero. This count is incremented every time the DisableControls (see page 6925) method is called and decremented when EnableControls (see page 6934) is called. Applications call DisableControls (see page 6925) to improve performance and prevent constant updates during automated iterations through records in the dataset.

In complex applications, when controls may be disabled multiple times by different processes, you can use ControlsDisabled as a check in a procedure to reenable controls should each call to DisableControls (see page 6925) not be paired with a subsequent call to EnableControls (see page 6934). For example,

```
procedure ReEnableControls (DataSet: TDataSet);
begin
while DataSet.ControlsDisabled do
  DataSet.EnableControls;
end;
void __fastcall TForm1::ReEnableControls(TDataSet *DataSet)
{
while (DataSet->ControlsDisabled())
    DataSet->EnableControls();
}
```

See Also

DisableControls (see page 6925)

EnableControls (see page 6934)

39.1.17.4.25 TDataSet.CopyFields Method

Pascal

```
function CopyFields(Source: TDataSet): Integer;
```

C++

```
__fastcall int CopyFields(TDataSet Source);
```

Description

This is CopyFields, a member of class TDataSet.

39.1.17.4.26 TDataSet.CreateBlobStream Method

Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.

Pascal

```
function CreateBlobStream(Field: TField; Mode: TBlobStreamMode): TStream; virtual;
```

C++

```
virtual __fastcall TStream CreateBlobStream(TField Field, TBlobStreamMode Mode);
```

Description

Call CreateBlobStream to obtain a stream for reading and writing the value of the field specified by the Field parameter. The Mode parameter indicates whether the stream will be used for reading the field's value (bmRead), writing the field's value (bmWrite), or modifying the field's value (bmReadWrite).

Blob streams are created in a specific mode for a specific field on a specific record. Applications create a new blob stream every time the record in the dataset changes: do not reuse an existing blob stream.

As implemented in TDataSet (see page 6894), CreateBlobStream always returns nil (Delphi) or NULL (C++). Descendants of TDataSet (see page 6894) override this method to create the TStream descendant that reads and writes BLOB data in the format that dataset type uses to store BLOB fields.

Tip: It is preferable to call CreateBlobStream rather than creating a blob stream directly in code. This ensures that the stream is appropriate to the dataset, and may also ensure that datasets that do not always store BLOB data in memory fetch the blob data before creating the stream.

39.1.17.4.27 TDataSet.CreateFields Method

Pascal

```
procedure CreateFields; virtual;
```

C++

```
virtual __fastcall CreateFields();
```

Description

This is CreateFields, a member of class TDataSet.

39.1.17.4.28 TDataSet.CreateNestedDataSet Method

Pascal

```
function CreateNestedDataSet(DataSetField: TDataSetField): TDataSet; virtual;
```

C++

```
virtual __fastcall TDataSet CreateNestedDataSet(TDataSetField DataSetField);
```

Description

This is CreateNestedDataSet, a member of class TDataSet.

39.1.17.4.29 TDataSet.CursorPosChanged Method

Marks the internal cursor position as invalid.

Pascal

```
procedure CursorPosChanged;
```

C++

```
__fastcall CursorPosChanged();
```

Description

CursorPosChanged is an internal method that invalidates the variable that tracks the physical cursor position relative to the logical cursor position. CurPosChanged is called by the Locate (see page 6970) and Lookup (see page 6970) methods prior to searching for a requested record. These methods, if successful, reposition the current record at the first matching record found.

See Also

Locate (see page 6970)

Lookup (see page 6970)

39.1.17.4.30 TDataSet.DataConvert Method

Pascal

```
procedure DataConvert(Field: TField; Source: Pointer; Dest: Pointer; ToNative: Boolean);
virtual;
```

C++

```
virtual __fastcall DataConvert(TField Field, void * Source, void * Dest, Boolean ToNative);
```

Description

This is DataConvert, a member of class TDataSet.

39.1.17.4.31 TDataSet.DataEvent Method

Pascal

```
procedure DataEvent(Event: TDataEvent; Info: Longint); virtual;
```

C++

```
virtual __fastcall DataEvent(TDataEvent Event, Longint Info);
```

Description

This is DataEvent, a member of class TDataSet.

39.1.17.4.32 TDataSet.DefChanged Method

Pascal

```
procedure DefChanged(Sender: TObject); virtual;
```

C++

```
virtual __fastcall DefChanged(TObject * Sender);
```

Description

This is DefChanged, a member of class TDataSet.

39.1.17.4.33 TDataSet.Delete Method

Deletes the active record and positions the dataset on the next record.

Pascal

```
procedure Delete;
```

C++

```
__fastcall Delete();
```

Description

Call Delete to remove the active record from the database. If the dataset is inactive, Delete raises an exception. Otherwise, Delete:

- Verifies that the dataset is not empty (and raises an exception if it is).
- Calls CheckBrowseMode (see page 6918) to post any pending changes to a prior record if necessary.
- Calls the BeforeDelete (see page 7049) event handler.

- Deletes the record.
- Frees any buffers allocated for the record.
- Puts the dataset into dsBrowse mode.
- Resynchronizes the dataset to make the next undeleted record active. If the record deleted was the last record in the dataset, then the previous record becomes the current record.
- Calls the AfterDelete (see page 7041) event handler.

See Also

State (see page 7036)

AfterDelete (see page 7041)

BeforeDelete (see page 7049)

CheckBrowseMode (see page 6918)

Edit (see page 6932)

Insert (see page 6959)

Post (see page 6977)

Deleting Records

39.1.17.4.34 TDataSet.DestroyFields Method

Pascal

```
procedure DestroyFields; virtual;
```

C++

```
virtual __fastcall DestroyFields();
```

Description

This is DestroyFields, a member of class TDataSet.

39.1.17.4.35 TDataSet.DisableControls Method

Disables data display in data-aware controls associated with the dataset.

Pascal

```
procedure DisableControls;
```

C++

```
__fastcall DisableControls();
```

Description

Call DisableControls prior to iterating through a large number of records in the dataset to prevent data-aware controls from updating every time the active record changes. Disabling controls prevents flicker and speeds performance because data does not need to be written to the display.

If controls are not already disabled, DisableControls records the current state of the dataset, broadcasts the state change to all associated data-aware controls and detail datasets, and increments the dataset's disabled count variable. Otherwise, DisableControls just increments the disabled count variable.

The disabled count is used internally to determine whether to display data in data-aware controls. When the disable count variable is greater than zero, data is not updated.

If the dataset is the master of a master/detail relationship, calling DisableControls also disables the master/detail relationship. Setting BlockReadSize (see page 7007) instead of calling DisableControls updates the detail datasets as you scroll through the dataset, but does not update data-aware controls.

Note: Calls to DisableControls can be nested. Only when all calls to DisableControls is matched to a corresponding call to EnableControls (see page 6934) does the dataset update data controls and detail datasets.

Delphi Examples:

```
{
Usually DisableControls is called within the context of a
try...finally block that reenables the controls even if an
exception occurs. For example:
}
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
begin
  ListBox1.Clear;
  ListBox1.Items[0]:= 'Destination Airports:';
  with Flights do
  begin
    DisableControls;
    try
      First;
      i:= 1;
      ListBox1.Items[0]:= 'Destination Airports:';
      while not Eof do
      begin
        ListBox1.Items[i]:= Fields[2].Value;;
        i:= i + 1;
        Next;
      end;
    finally
      EnableControls;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Flights:= TTable.Create(Form1);
  with Flights do
  begin
    TableName := 'Flights';
    DatabaseName := 'Flights';

  end;
  DS2.DataSet:= Flights;
  DBGrid2.DataSource.DataSet:=
    c:\Documents and Settings\All Users\Documents\RAD
Studio\5.0\Demos\IntraWeb\Win32\FlightInformation\Data;
  Flights.Active:= True;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
    I: Integer;
  begin
    if not RemoteServer1.Connected then
      RemoteServer1.Connected := True;
    ClientDataSet1.Close;
    with RemoteServer1 do
    begin
      for I := 0 to DataSetCount - 1 do
        DataSets[I].EnableControls;
    end;
    ClientDataSet1.Open;
  end;

  procedure TForm1.Button3Click(Sender: TObject);
  var
    I: Integer;
  begin
    ClientDataSet1.Close;
    RemoteServer1.Connected := False;
    with RemoteServer1 do
    begin
      for I := 0 to DataSetCount - 1 do
        DataSets[I].DisableControls;
    end;
  end;

  procedure TForm1.CheckButtonActive(DataSet: TDataSet);
  begin
    Button1.Enabled := not ClientDataSet1.Active;
    Button2.Enabled := ClientDataSet1.Active;
    Button3.Enabled := ClientDataSet1.Active;
  end;
```

See Also

ControlsDisabled (see page 6921)

EnableControls (see page 6934)

BlockReadSize (see page 7007)

Disabling and Enabling Data Display

Enabling and Disabling the Data Source

39.1.17.4.36 TDataSet.DoAfterCancel Method

Pascal

```
procedure DoAfterCancel; virtual;
```

C++

```
virtual __fastcall DoAfterCancel();
```

Description

This is DoAfterCancel, a member of class TDataSet.

39.1.17.4.37 TDataSet.DoAfterClose Method

Pascal

```
procedure DoAfterClose; virtual;
```

C++

```
virtual __fastcall DoAfterClose();
```

Description

This is DoAfterClose, a member of class TDataSet.

39.1.17.4.38 TDataSet.DoAfterDelete Method

Pascal

```
procedure DoAfterDelete; virtual;
```

C++

```
virtual __fastcall DoAfterDelete();
```

Description

This is DoAfterDelete, a member of class TDataSet.

39.1.17.4.39 TDataSet.DoAfterEdit Method

Pascal

```
procedure DoAfterEdit; virtual;
```

C++

```
virtual __fastcall DoAfterEdit();
```

Description

This is DoAfterEdit, a member of class TDataSet.

39.1.17.4.40 TDataSet.DoAfterInsert Method

Pascal

```
procedure DoAfterInsert; virtual;
```

C++

```
virtual __fastcall DoAfterInsert();
```

Description

This is DoAfterInsert, a member of class TDataSet.

39.1.17.4.41 TDataSet.DoAfterOpen Method

Pascal

```
procedure DoAfterOpen; virtual;
```

C++

```
virtual __fastcall DoAfterOpen();
```

Description

This is DoAfterOpen, a member of class TDataSet.

39.1.17.4.42 TDataSet.DoAfterPost Method

Pascal

```
procedure DoAfterPost; virtual;
```

C++

```
virtual __fastcall DoAfterPost();
```

Description

This is DoAfterPost, a member of class TDataSet.

39.1.17.4.43 TDataSet.DoAfterRefresh Method

Pascal

```
procedure DoAfterRefresh; virtual;
```

C++

```
virtual __fastcall DoAfterRefresh();
```

Description

This is DoAfterRefresh, a member of class TDataSet.

39.1.17.4.44 TDataSet.DoAfterScroll Method

Pascal

```
procedure DoAfterScroll; virtual;
```

C++

```
virtual __fastcall DoAfterScroll();
```

Description

This is DoAfterScroll, a member of class TDataSet.

39.1.17.4.45 TDataSet.DoBeforeCancel Method

Pascal

```
procedure DoBeforeCancel; virtual;
```

C++

```
virtual __fastcall DoBeforeCancel();
```

Description

This is DoBeforeCancel, a member of class TDataSet.

39.1.17.4.46 TDataSet.DoBeforeClose Method

Pascal

```
procedure DoBeforeClose; virtual;
```

C++

```
virtual __fastcall DoBeforeClose();
```

Description

This is DoBeforeClose, a member of class TDataSet.

39.1.17.4.47 TDataSet.DoBeforeDelete Method

Pascal

```
procedure DoBeforeDelete; virtual;
```

C++

```
virtual __fastcall DoBeforeDelete();
```

Description

This is DoBeforeDelete, a member of class TDataSet.

39.1.17.4.48 TDataSet.DoBeforeEdit Method

Pascal

```
procedure DoBeforeEdit; virtual;
```

C++

```
virtual __fastcall DoBeforeEdit();
```

Description

This is DoBeforeEdit, a member of class TDataSet.

39.1.17.4.49 TDataSet.DoBeforeInsert Method

Pascal

```
procedure DoBeforeInsert; virtual;
```

C++

```
virtual __fastcall DoBeforeInsert();
```

Description

This is DoBeforeInsert, a member of class TDataSet.

39.1.17.4.50 TDataSet.DoBeforeOpen Method

Pascal

```
procedure DoBeforeOpen; virtual;
```

C++

```
virtual __fastcall DoBeforeOpen();
```

Description

This is DoBeforeOpen, a member of class TDataSet.

39.1.17.4.51 TDataSet.DoBeforePost Method

Pascal

```
procedure DoBeforePost; virtual;
```

C++

```
virtual __fastcall DoBeforePost();
```

Description

This is DoBeforePost, a member of class TDataSet.

39.1.17.4.52 TDataSet.DoBeforeRefresh Method

Pascal

```
procedure DoBeforeRefresh; virtual;
```

C++

```
virtual __fastcall DoBeforeRefresh();
```

Description

This is DoBeforeRefresh, a member of class TDataSet.

39.1.17.4.53 TDataSet.DoBeforeScroll Method

Pascal

```
procedure DoBeforeScroll; virtual;
```

C++

```
virtual __fastcall DoBeforeScroll();
```

Description

This is DoBeforeScroll, a member of class TDataSet.

39.1.17.4.54 TDataSet.DoOnCalcFields Method

Pascal

```
procedure DoOnCalcFields; virtual;
```

C++

```
virtual __fastcall DoOnCalcFields();
```

Description

This is DoOnCalcFields, a member of class TDataSet.

39.1.17.4.55 TDataSet.DoOnNewRecord Method

Pascal

```
procedure DoOnNewRecord; virtual;
```

C++

```
virtual __fastcall DoOnNewRecord();
```

Description

This is DoOnNewRecord, a member of class TDataSet.

39.1.17.4.56 TDataSet.Edit Method

Enables editing of data in the dataset.

Pascal

```
procedure Edit;
```

C++

```
__fastcall Edit();
```

Description

Call Edit to permit editing of the active record in a dataset. Edit determines the current state of the dataset. If the dataset is empty, Edit calls Insert (see page 6959). Otherwise Edit

Calls CheckBrowseMode (see page 6918) to post any pending changes to a prior record if necessary.

Checks the CanModify (see page 7012) property and raises an exception if the dataset can't be edited.

Calls the BeforeEdit (see page 7049) event handler.

Retrieves the record.

Puts the dataset into dsEdit state, enabling the application or user to modify fields in the record.

Broadcasts the state change to associated controls.

Calls the AfterEdit (see page 7044) event handler.

Delphi Examples:

```
{
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
{
The following example copies the data in the Notes field of
Table1 or SQLDataSet1 to the Remarks field of ClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Stream1: TBlobStream;
```

```
    Stream2: TStream;
    blobType : TBlobType;
  begin
    blobType := Table1Notes.BlobType;
    Stream1 := TBlobStream.Create(Table1Notes, bmRead);
    try
      CDS2.Edit;
      { here's a different way to create a blob stream }
      Stream2 := CDS2.CreateBlobStream(CDS2.FieldByName('Remarks'), bmReadWrite);
      try
        Stream2.CopyFrom(Stream1, Stream1.Size);
//      CDS2.Post;
//      CDS2.Active := True;
      finally
        Stream2.Free;
      end;
    finally
      Stream1.Free;
    end;
  end;
  {
  This example uses a button to copy the value of a field in
  the previous record into the corresponding field in the
  current record.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
     SavePlace: TBookmark;
     PrevValue: Variant;
  begin
     with Customers do
     begin
      { get a bookmark so that we can return to the same record }
      SavePlace := GetBookmark;
      try
        { move to prior record}
        FindPrior;
        { get the value }
        PrevValue := Fields[1].Value;
        {Move back to the bookmark
        this may not be the next record anymore
        if something else is changing the dataset asynchronously }
        GotoBookmark(SavePlace);
        { Set the value }
        Edit;
        Fields[1].Value := PrevValue;
        { Free the bookmark }
      finally
        FreeBookmark(SavePlace);
      end;
    end;
  end;

  {
  To ensure that the button is disabled when there is no
  previous record, the OnDataChange event of the DataSource
  detects when the user moves to the beginning of file (BOF
  property becomes true), and disables the button.  Detection
  occurs on scrolling and editing, not selection with the mouse.
  }
  procedure TForm1.DS2DataChange(Sender: TObject; Field: TField);
  begin
    if Customers.Bof then
      Button1.Enabled := False
    else
      Button1.Enabled := True;
```

```
end;
```

See Also

State (see page 7036)

AfterEdit (see page 7044)

BeforeEdit (see page 7049)

CheckBrowseMode (see page 6918)

Delete (see page 6924)

Insert (see page 6959)

CanModify (see page 7012)

Enabling Editing in Controls On User Entry

Editing Records

Modifying Entire Records

39.1.17.4.57 TDataSet.EnableControls Method

Re-enables data display in data-aware controls associated with the dataset.

Pascal

```
procedure EnableControls;
```

C++

```
__fastcall EnableControls();
```

Description

Call EnableControls to permit data display in data-aware controls after a prior call to DisableControls (see page 6925). EnableControls decrements the disabled count variable for the dataset if it is not already zero. If the disable count variable is zero, EnableControls updates the current state of the dataset, if necessary, and then tells associated controls to re-enable display.

Delphi Examples:

```
{
Usually DisableControls is called within the context of a
try...finally block that reenables the controls even if an
exception occurs. For example:
}
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
begin
  ListBox1.Clear;
  ListBox1.Items[0]:= 'Destination Airports:';
  with Flights do
  begin
    DisableControls;
    try
      First;
      i:= 1;
      ListBox1.Items[0]:= 'Destination Airports:';
      while not Eof do
      begin
        ListBox1.Items[i]:= Fields[2].Value;;
        i:= i + 1;
```

```
        Next;
      end;
    finally
      EnableControls;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Flights:= TTable.Create(Form1);
  with Flights do
  begin
    TableName := 'Flights';
    DatabaseName := 'Flights';

  end;
  DS2.DataSet:= Flights;
  DBGrid2.DataSource.DataSet:=
    c:\Documents and Settings\All Users\Documents\RAD
Studio\5.0\Demos\IntraWeb\Win32\FlightInformation\Data;
  Flights.Active:= True;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
  RemoteServer1.Connected := False;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].DisableControls;
  end;
end;

procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
```

See Also

ControlsDisabled (see page 6921)

DisableControls (see page 6925)

Disabling and Enabling Data Display

Enabling and Disabling the Data Source

39.1.17.4.58 TDataSet.FieldByName Method

Finds a field based on its name.

Pascal

```
function FieldByName(const FieldName: WideString): TField;
```

C++

```
__fastcall TField FieldByName(const BSTR FieldName);
```

Description

Call FieldByName to retrieve field information for a field given its name. FieldName is the name of an existing field. FieldByName returns the TField (see page 7115) component that represents the specified field. If the specified field does not exist, FieldByName raises an EDatabaseError (see page 6731) exception.

FieldName can be the name of a simple field, the name of a subfield of an object field that has been qualified by the parent field's name, or the name of an aggregated field. Because of this flexibility, it is often preferable to use FieldByName rather than the Fields (see page 7023) property or the AggFields (see page 7006) property.

An application can directly access specific properties and methods of the field returned by FieldByName. For example, the following statement determines if a specified field is a calculated field:

```
if Customers.FieldByName('FullName').Calculated then
  MessageDlg(Format('%s is a calculated field. ', ['FullName']), mtInformation, [mbOK], 0);
if (Customers->FieldByName("FullName")->Calculated)
  Application.ShowMessage("This is a calculated field", "FullName", MB_OK);
```

FieldByName is especially useful at design time for developers who are creating database applications, but do not have access to the underlying table and therefore cannot use persistent field components.

Tip: To retrieve or set the value for a specific field, call the default dataset method FieldValues (see page 7025) instead of FieldByName.

Delphi Examples:

```
{
The following example copies the data in the Notes field of
Table1 or SQLDataSet1 to the Remarks field of ClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Stream1: TBlobStream;
  Stream2: TStream;
  blobType : TBlobType;
begin
  blobType := Table1Notes.BlobType;
  Stream1 := TBlobStream.Create(Table1Notes, bmRead);
  try
    CDS2.Edit;
    { here’s a different way to create a blob stream }
```

```
    Stream2 := CDS2.CreateBlobStream(CDS2.FieldByName('Remarks'), bmReadWrite);
    try
      Stream2.CopyFrom(Stream1, Stream1.Size);
//    CDS2.Post;
//    CDS2.Active := True;
    finally
      Stream2.Free;
    end;
  finally
    Stream1.Free;
  end;
end;
{
This example uses the BeforeInsert event to do data
validation. If the StrToInt function raises an exception,
the edit control’s contents are set to a valid value so
that the assignment to the INTEGER field in the dataset
will succeed.  Continuing after the exception causes a
'0' to be placed in the PORTA field.
DataSet design:
This example requires a ClientDataSet, a DBGrid, a
DataSource and a DBNavigator.  Name the ClientDataSet CDS.
Set the DataSet property in the DataSource to CDS and name
that DataSource DS.  Set the DataSource in the DBNavigator
to DS the DataSource in the DBGrid to DS.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
type to Integer.  The file used to load CDS must have an
attribute name of HOST with an attribute type of string and
an attribute name of PORTA with an attribute type of integer.
Add DB, DBCtrls, Grids, DBGrids and DBClient to the uses clause.
}
{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button3Click(Sender: TObject);
begin
  CDS.Insert;
  CDS.FieldByName('PORTA').AsInteger := StrToInt(Edit1.Text);
  CDS.Post;
end;

procedure TForm1.CDSBeforeInsert(DataSet: TDataSet);
begin
  try
  { Make sure edit field can be converted to integer --
   this will raise an exception if it can’t }
    StrToInt(Edit1.Text);
  except
    Edit1.Text := '0';
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
```

```
end;
```

See Also

Fields (see page 7023)

FindField (see page 6938)

FieldValues (see page 7025)

AggFields (see page 7006)

ObjectView (see page 7031)

Specifying Ranges

Accessing Field Values with a Dataset'sFieldByName Method

39.1.17.4.59 TDataSet.FieldByNumber Method

Pascal

```
function FieldByNumber(FieldNo: Integer): TField;
```

C++

```
__fastcall TField FieldByNumber(int FieldNo);
```

Description

This is FieldByNumber, a member of class TDataSet.

39.1.17.4.60 TDataSet.FindField Method

Searches for a specified field in the dataset.

Pascal

```
function FindField(const FieldName: WideString): TField;
```

C++

```
__fastcall TField FindField(const BSTR FieldName);
```

Description

Call FindField to determine if a specified field component exists in a dataset. FieldName is the name of the field for which to search. This name can be the the name of a simple field, the name of a subfield of an object field that has been qualified by the parent field's name, or the name of an aggregated field.

If FindField finds a field with a matching name, it returns the TField (see page 7115) component for the specified field. Otherwise it returns nil (Delphi) or NULL (C++).

FindField is the same as the FieldByName (see page 6936) method, except that it returns nil (Delphi) or NULL (C++) rather than raising an exception when the field is not found.

Delphi Examples:

```
{
This example uses a button to copy the value of a field in
the previous record into the corresponding field in the
current record.  The field to copy is specified by using
FindField and the name of the field.
}
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
    SavePlace: TBookmark;
    PrevValue: Variant;
  begin
    with Customers do
    begin
     { get a bookmark so that we can return to the same record }
     SavePlace := GetBookmark;
     try
       { move to prior record}
       FindPrior;
       { get the value }
       { This is the safe way to get 'CustNo' field }
       PrevValue := FindField('Field2').Value;
       { This is *not* the safe way to change 'CustNo' field }
//     PrevValue := Fields[1].Value;
       { Move back to the bookmark
       this may not be the next record anymore
       if something else is changing the dataset asynchronously }
       GotoBookmark(SavePlace);
       { Set the value }
       Edit;
       { This is the safe way to change 'CustNo' field }
       FindField('Field2').AsString := PrevValue;
       { This is *not* the safe way to change 'CustNo' field }
//       Fields[1].AsString := PrevValue;
       { Free the bookmark }
     finally
       FreeBookmark(SavePlace);
     end;
   end;
  end;
```

See Also

Fields (see page 7023)

FieldByName (see page 6936)

AggFields (see page 7006)

Obtaining Aggregate Values

39.1.17.4.61 TDataSet.FindFirst Method

Implements a virtual method for positioning the dataset on the first record, respecting any filters.

Pascal

```
function FindFirst: Boolean;
```

C++

```
__fastcall Boolean FindFirst();
```

Description

This function returns false, indicating that the active record was not successfully changed. Descendant classes override FindFirst to make the first record active, honoring any filters that are in effect. In descendant classes, FindFirst returns true if the active record is successfully changed.

See Also

Filter (see page 7026)

Filtered (see page 7027)

OnFilterRecord (see page 7055)

FindLast (see page 6940)

FindNext (see page 6940)

FindPrior (see page 6941)

First (see page 6942)

39.1.17.4.62 TDataSet.FindLast Method

Implements a virtual method for positioning the dataset on the last record, respecting any filters.

Pascal

```
function FindLast: Boolean;
```

C++

```
__fastcall Boolean FindLast();
```

Description

This function returns false, indicating that the active record was not successfully changed. Descendant classes override FindLast to make the last record active, honoring any filters that are in effect. In descendants, FindLast returns true if the active record is successfully changed.

See Also

Filter (see page 7026)

Filtered (see page 7027)

OnFilterRecord (see page 7055)

FindFirst (see page 6939)

FindNext (see page 6940)

FindPrior (see page 6941)

Last (see page 6968)

39.1.17.4.63 TDataSet.FindNext Method

Implements a virtual method for positioning the dataset on the next record, respecting any filters.

Pascal

```
function FindNext: Boolean;
```

C++

```
__fastcall Boolean FindNext();
```

Description

This function returns false, indicating that the active record was not successfully changed. Descendant classes override FindNext to move to the next record of the dataset, honoring any filters that are in effect. In descendants, FindNext returns true if the active record is successfully changed.

See Also

Filter (see page 7026)

Filtered (see page 7027)

OnFilterRecord (see page 7055)

FindFirst (see page 6939)

FindLast (see page 6940)

FindPrior (see page 6941)

Next (see page 6973)

Navigating Records in a Filtered Dataset

39.1.17.4.64 TDataSet.FindPrior Method

Implements a virtual method for positioning the dataset on the previous record, respecting any filters.

Pascal

```
function FindPrior: Boolean;
```

C++

```
__fastcall Boolean FindPrior();
```

Description

This function returns false, indicating that the active record was not successfully changed. Descendant classes override FindPrior to move to the previous record of the dataset, honoring any filters that are in effect. In descendants, FindPrior returns true if the active record is successfully changed.

Delphi Examples:

```
{
This example uses a button to copy the value of a field in
the previous record into the corresponding field in the
current record.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   SavePlace: TBookmark;
   PrevValue: Variant;
begin
   with Customers do
   begin
    { get a bookmark so that we can return to the same record }
    SavePlace := GetBookmark;
    try
      { move to prior record}
      FindPrior;
      { get the value }
      PrevValue := Fields[1].Value;
      {Move back to the bookmark
      this may not be the next record anymore
      if something else is changing the dataset asynchronously }
      GotoBookmark(SavePlace);
      { Set the value }
      Edit;
      Fields[1].Value := PrevValue;
      { Free the bookmark }
    finally
      FreeBookmark(SavePlace);
    end;
  end;
end;

{
To ensure that the button is disabled when there is no
previous record, the OnDataChange event of the DataSource
```

```
  detects when the user moves to the beginning of file (BOF
  property becomes true), and disables the button.  Detection
  occurs on scrolling and editing, not selection with the mouse.
  }
  procedure TForm1.DS2DataChange(Sender: TObject; Field: TField);
  begin
    if Customers.Bof then
      Button1.Enabled := False
    else
      Button1.Enabled := True;
  end;
```

See Also

Filter (see page 7026)

Filtered (see page 7027)

OnFilterRecord (see page 7055)

FindFirst (see page 6939)

FindLast (see page 6940)

FindNext (see page 6940)

Prior (see page 6979)

39.1.17.4.65 TDataSet.FindRecord Method

Pascal

```
function FindRecord(Restart: Boolean; GoForward: Boolean): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean FindRecord(Boolean Restart, Boolean GoForward);
```

Description

This is FindRecord, a member of class TDataSet.

39.1.17.4.66 TDataSet.First Method

Moves to the first record in the dataset.

Pascal

```
procedure First;
```

C++

```
__fastcall First();
```

Description

Call First to make the first record in the dataset active. First posts any changes to the active record and:

Clears the record buffers.

Fetches the first record and makes it the active record.

Fetches any additional records required for display, such as those needed to fill out a grid control.

Sets the Bof (see page 7008) property to true.

Broadcasts the record change so that data controls and linked detail sets can update.

Note: TDataSet (see page 6894) uses internal, protected methods to reposition the active record and to fetch additional records required for display. In TDataSet (see page 6894), these internal methods are empty stubs. Descendant classes implement these methods to enable the First method to work.

C++ Examples:

```
/*
Reads through all records in the Customers table.
Updates the ProgressBar accordingly.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ProgressBar1->Min = 0;
  ProgressBar1->Max = Customers->RecordCount;
  Customers->First();
  for (int i = ProgressBar1->Min; i <= ProgressBar1->Max; i++)
  {
    ProgressBar1->Position = i;
    Customers->Next();
    // do something with the current record
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Customers = new TTable(Form1);
  Customers->Active = false; // The Table component must not be active
  Customers->DatabaseName = "DBDEMOS";
  Customers->TableType = ttParadox;
  Customers->TableName = "CustInfo";
  Customers->Active = False;
  if (Customers->Exists) // Don't overwrite an existing table
  {
    Customers->Close();
    Customers->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Customers->FieldDefs->Clear();
  TFieldDef *pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Customers->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Customers->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Customers->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Customers->CreateTable();
  Customers->Active = True;
  for (int i = 1; i <= 20; i++)
    Customers->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  DS2->DataSet = Customers;
  DBGrid2->DataSource->DataSet = Customers;
  Customers->Active = True;
}
```

Delphi Examples:

```
{
Reads through all records in the Customers table.
Updates the ProgressBar accordingly.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
begin
  with ProgressBar1 do
  begin
    Min := 0;
    Max := Customers.RecordCount;
    Customers.First;
    for i := Min to Max do
    begin
      Position := i;
      Customers.Next;
      // do something with the current record
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Customers:= TTable.Create(Form1);
  with Customers do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Active := False;
//  Overwrite any existing table
    if Customers.Exists then
    begin
      Customers.Close;
      Customers.DeleteTable;
    end;
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
```

```
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Customers.Active:= True;
        for i := 1 to 100 do
          Customers.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Customers;
    DBGrid2.DataSource.DataSet:= Customers;
    Customers.Active:= True;
  end;
```

See Also

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Prior (see page 6979)

Posting Data

Using the Eof and Bof Properties

Using the First and Last Methods

Using Unidirectional Datasets

39.1.17.4.67 TDataSet.FreeBookmark Method

Frees the resources allocated for a specified bookmark.

Pascal

```
procedure FreeBookmark(Bookmark: TBookmark); virtual;
```

C++

```
virtual __fastcall FreeBookmark(TBookmark Bookmark);
```

Description

Call FreeBookmark to free an existing bookmark before reassigning it. FreeBookmark releases the memory allocated for a specified bookmark when the bookmark is no longer needed.

Delphi Examples:

```
{
This example uses a button to copy the value of a field in
the previous record into the corresponding field in the
```

```
current record.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   SavePlace: TBookmark;
   PrevValue: Variant;
begin
   with Customers do
   begin
    { get a bookmark so that we can return to the same record }
    SavePlace := GetBookmark;
    try
      { move to prior record}
      FindPrior;
      { get the value }
      PrevValue := Fields[1].Value;
      {Move back to the bookmark
      this may not be the next record anymore
      if something else is changing the dataset asynchronously }
      GotoBookmark(SavePlace);
      { Set the value }
      Edit;
      Fields[1].Value := PrevValue;
      { Free the bookmark }
    finally
      FreeBookmark(SavePlace);
    end;
  end;
end;

{
To ensure that the button is disabled when there is no
previous record, the OnDataChange event of the DataSource
detects when the user moves to the beginning of file (BOF
property becomes true), and disables the button.  Detection
occurs on scrolling and editing, not selection with the mouse.
}
procedure TForm1.DS2DataChange(Sender: TObject; Field: TField);
begin
  if Customers.Bof then
    Button1.Enabled := False
  else
    Button1.Enabled := True;
end;
```

See Also

Bookmark (see page 7009)

GetBookmark (see page 6948)

GotoBookmark (see page 6957)

Marking and Returning to Records

39.1.17.4.68 TDataSet.FreeFieldBuffers Method

Pascal

```
procedure FreeFieldBuffers; virtual;
```

C++

```
virtual __fastcall FreeFieldBuffers();
```

Description

This is FreeFieldBuffers, a member of class TDataSet.

39.1.17.4.69 TDataSet.FreeRecordBuffer Method

Pascal

```
procedure FreeRecordBuffer(var Buffer: PChar); virtual;
```

C++

```
virtual __fastcall FreeRecordBuffer(const char * Buffer);
```

Description

This is FreeRecordBuffer, a member of class TDataSet.

39.1.17.4.70 TDataSet.GetAggRecordCount Method

Pascal

```
function GetAggRecordCount(Grp: TGroupPosInd): Integer; virtual;
```

C++

```
virtual __fastcall int GetAggRecordCount(TGroupPosInd Grp);
```

Description

This is GetAggRecordCount, a member of class TDataSet.

39.1.17.4.71 TDataSet.GetAggregateValue Method

Pascal

```
function GetAggregateValue(Field: TField): Variant; virtual;
```

C++

```
virtual __fastcall Variant GetAggregateValue(TField Field);
```

Description

This is GetAggregateValue, a member of class TDataSet.

39.1.17.4.72 TDataSet.GetBlobFieldData Method

Reads BLOB data into a buffer.

Pascal

```
function GetBlobFieldData(FieldNo: Integer; var Buffer: TBlobByteData): Integer; virtual;
```

C++

```
virtual __fastcall int GetBlobFieldData(int FieldNo, TBlobByteData Buffer);
```

Description

GetBlobFieldData reads blob data from the field specified by FieldNo into a Buffer. The buffer is a dynamic array of bytes, so that it can grow to accommodate the size of the BLOB data. GetBlobFieldData returns the size of the buffer.

GetBlobFieldData is intended primarily for internal use. Applications should use the BLOB stream returned by the CreateBlobStream (see page 6922) method to access BLOB data.

See Also

BlobFieldCount (see page 7007)

GetFieldData (see page 6952)

CreateBlobStream (see page 6922)

39.1.17.4.73 TDataSet.GetBookmark Method

Allocates a bookmark for the active record in the dataset.

Pascal

```
function GetBookmark: TBookmark; virtual;
```

C++

```
virtual __fastcall TBookmark GetBookmark();
```

Description

Call GetBookmark to establish a bookmark for the active record in the dataset. Establishing a bookmark for a record enables a dataset to return to that record at any time while the bookmark exists.

GetBookmark requires that a variable for the bookmark (for Delphi, of type TBookmark (see page 7628)) already be declared in the application. Use GetBookmark to assign a value to the variable. This value can then be referenced by subsequent calls to GotoBookmark (see page 6957) and FreeBookmark (see page 6945).

Applications that create bookmarks with GetBookmark must subsequently release the system resource allocated to them by calling FreeBookmark (see page 6945) when the bookmarks are no longer needed.

Note: GetBookmark relies on a protected method to obtain the bookmark value. TDataSet (see page 6894) descendants implement this method to provide their own type of bookmark support. Unidirectional datasets do not support bookmarks, and so do not return a meaningful value.

Delphi Examples:

```
{
This example uses a button to copy the value of a field in
the previous record into the corresponding field in the
current record.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   SavePlace: TBookmark;
   PrevValue: Variant;
begin
   with Customers do
   begin
    { get a bookmark so that we can return to the same record }
    SavePlace := GetBookmark;
    try
      { move to prior record}
      FindPrior;
      { get the value }
      PrevValue := Fields[1].Value;
      {Move back to the bookmark
      this may not be the next record anymore
      if something else is changing the dataset asynchronously }
      GotoBookmark(SavePlace);
      { Set the value }
      Edit;
```

```
        Fields[1].Value := PrevValue;
        { Free the bookmark }
      finally
        FreeBookmark(SavePlace);
      end;
    end;
  end;

  {
  To ensure that the button is disabled when there is no
  previous record, the OnDataChange event of the DataSource
  detects when the user moves to the beginning of file (BOF
  property becomes true), and disables the button.  Detection
  occurs on scrolling and editing, not selection with the mouse.
  }
  procedure TForm1.DS2DataChange(Sender: TObject; Field: TField);
  begin
    if Customers.Bof then
      Button1.Enabled := False
    else
      Button1.Enabled := True;
  end;
```

See Also

FreeBookmark (see page 6945)

GotoBookmark (see page 6957)

Marking and Returning to Records

39.1.17.4.74 TDataSet.GetBookmarkData Method

Pascal

```
procedure GetBookmarkData(Buffer: PChar; Data: Pointer); virtual;
```

C++

```
virtual __fastcall GetBookmarkData(const char * Buffer, void * Data);
```

Description

This is GetBookmarkData, a member of class TDataSet.

39.1.17.4.75 TDataSet.GetBookmarkFlag Method

Pascal

```
function GetBookmarkFlag(Buffer: PChar): TBookmarkFlag; virtual;
```

C++

```
virtual __fastcall TBookmarkFlag GetBookmarkFlag(const char * Buffer);
```

Description

This is GetBookmarkFlag, a member of class TDataSet.

39.1.17.4.76 TDataSet.GetBookmarkStr Method

Pascal

```
function GetBookmarkStr: TBookmarkStr; virtual;
```

C++

```
virtual __fastcall TBookmarkStr GetBookmarkStr();
```

Description

This is GetBookmarkStr, a member of class TDataSet.

39.1.17.4.77 TDataSet.GetCalcFields Method

Pascal

```
procedure GetCalcFields(Buffer: PChar); virtual;
```

C++

```
virtual __fastcall GetCalcFields(const char * Buffer);
```

Description

This is GetCalcFields, a member of class TDataSet.

39.1.17.4.78 TDataSet.GetCanModify Method

Pascal

```
function GetCanModify: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetCanModify();
```

Description

This is GetCanModify, a member of class TDataSet.

39.1.17.4.79 TDataSet.GetChildren Method

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

This is GetChildren, a member of class TDataSet.

39.1.17.4.80 TDataSet.GetCurrentRecord Method

Implements a virtual function to retrieve the current record in a buffer.

Pascal

```
function GetCurrentRecord(Buffer: PChar): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetCurrentRecord(const char * Buffer);
```

Description

This function always returns false, indicating failure. For TDataSet (see page 6894) descendants that override this method to retrieve the value of the current record, GetCurrentRecord returns true if Buffer is successfully filled with the value of the current

record.

See Also

ActiveBuffer (see page 6909)

CurrentRecord (see page 7013)

39.1.17.4.81 TDataSet.GetDataSource Method

Pascal

```
function GetDataSource: TDataSource; virtual;
```

C++

```
virtual __fastcall TDataSource GetDataSource();
```

Description

This is GetDataSource, a member of class TDataSet.

39.1.17.4.82 TDataSet.GetDetailDataSets Method

Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.

Pascal

```
procedure GetDetailDataSets(List: TList); virtual;
```

C++

```
virtual __fastcall GetDetailDataSets(TList List);
```

Description

Datasets can represent master/detail relationships in two ways: as linked cursors or as nested dataset fields. GetDetailDataSets lists all detail datasets of the active record into List if they are not the value of a nested dataset field. To obtain a list of the detail datasets that are the values of nested dataset fields, use the NestedDataSets (see page 7030) property instead.

See Also

TDetailDataLink (see page 7111)

GetDetailLinkFields (see page 6951)

NestedDataSets (see page 7030)

39.1.17.4.83 TDataSet.GetDetailLinkFields Method

Introduces a method to list the field components that link this dataset as a detail of a master dataset.

Pascal

```
procedure GetDetailLinkFields(MasterFields: TList; DetailFields: TList); virtual;
```

C++

```
virtual __fastcall GetDetailLinkFields(TList MasterFields, TList DetailFields);
```

Description

As implemented in TDataSet (see page 6894), GetDetailLinkFields does nothing. Descendants override this method to fill the two lists with field components that define a master-detail relationship between this dataset and another (master) dataset. The MasterFields list is filled with fields from the master table whose values must equal the values of the fields in the DetailFields list.

The DetailFields list is filled with fields from the calling dataset.

39.1.17.4.84 TDataSet.GetFieldClass Method

Pascal

```
function GetFieldClass(FieldType: TFieldType): TFieldClass; virtual;
```

C++

```
virtual __fastcall TFieldClass GetFieldClass(TFieldType FieldType);
```

Description

This is GetFieldClass, a member of class TDataSet.

39.1.17.4.85 GetFieldData Method

This is the overview for the GetFieldData method overload.

Overload List

	Name	Description
	TDataSet.GetFieldData (Integer, Pointer) (see page 6952)	Retrieves the current value of a field into a buffer.
	TDataSet.GetFieldData (TField, Pointer) (see page 6952)	Retrieves the current value of a field into a buffer.
	TDataSet.GetFieldData (TField, Pointer, Boolean) (see page 6952)	Retrieves the current value of a field into a buffer.

39.1.17.4.85.1 TDataSet.GetFieldData Method (Integer, Pointer)

Retrieves the current value of a field into a buffer.

Pascal

```
function GetFieldData(Field: TField; Buffer: Pointer): Boolean; virtual; overload;
function GetFieldData(FieldNo: Integer; Buffer: Pointer): Boolean; virtual; overload;
function GetFieldData(Field: TField; Buffer: Pointer; NativeFormat: Boolean): Boolean;
virtual; overload;
```

C++

```
virtual __fastcall Boolean GetFieldData(TField Field, void * Buffer);
virtual __fastcall Boolean GetFieldData(int FieldNo, void * Buffer);
virtual __fastcall Boolean GetFieldData(TField Field, void * Buffer, Boolean NativeFormat);
```

Description

Most applications do not need to call GetFieldData. TField (see page 7115) objects call this method to implement their GetData method.

The Field or FieldNo parameter indicates the field whose data should be fetched. Field specifies the component itself, while FieldNo indicates its field number. The Buffer parameter is a memory buffer with sufficient space to accept the value of the field as it exists in the database (unformatted and untranslated). NativeFormat indicates whether the dataset fetches the field in the IDE's native format for the field type. When NativeFormat is false, the dataset must convert the field value to the native type. This allows the field to handle data from different types of datasets (ADO-based, BDE-based, and so on) in a uniform manner.

GetFieldData returns a value that indicates whether the data was successfully fetched.

As implemented in TDataSet (see page 6894), GetFieldData always returns false, indicating that no data was fetched from the

specified field. Descendants override this method to fetch data in whatever way is appropriate to the implementation of the dataset.

See Also

GetFieldList (see page 6953)

GetData (see page 7131)

39.1.17.4.86 TDataSet.GetFieldFullName Method

Pascal

```
function GetFieldFullName(Field: TField): string; virtual;
```

C++

```
virtual __fastcall AnsiString GetFieldFullName(TField Field);
```

Description

This is GetFieldFullName, a member of class TDataSet.

39.1.17.4.87 TDataSet.GetFieldList Method

Retrieves a specified set of field objects into a list.

Pascal

```
procedure GetFieldList(List: TList; const FieldNames: WideString);
```

C++

```
__fastcall GetFieldList(TList List, const BSTR FieldNames);
```

Description

Call GetFieldList to copy a specified set of field objects into a list object created and maintained by the application.

List is the TList object into which to copy the field objects. FieldNames is a string containing the name of the fields to copy. Each field name in the string must be separated from other field names with a semicolon. GetFieldList builds a list that contains only the field objects for which it finds a matching name in the dataset's list of field objects.

Note: Applications do not normally call GetFieldList to copy field objects. Field objects are directly accessible through the dataset itself. In some cases, however, it can be useful to work with a copy of a field object or its data instead of working on the actual object in the dataset.

See Also

FieldDefs (see page 7019)

Fields (see page 7023)

FieldByName (see page 6936)

FindField (see page 6938)

GetFieldNames (see page 6954)

39.1.17.4.88 GetFieldNames Method

This is the overview for the GetFieldNames method overload.

Overload List

	Name	Description
	TDataSet.GetFieldNames (TStrings) (see page 6954)	Retrieves a list of names for all fields in a dataset.
	TDataSet.GetFieldNames (TWideStrings) (see page 6954)	Retrieves a list of names for all fields in a dataset.

39.1.17.4.88.1 TDataSet.GetFieldNames Method (TStrings)

Retrieves a list of names for all fields in a dataset.

Pascal

```
procedure GetFieldNames(List: TStrings); virtual; overload; deprecated;
procedure GetFieldNames(List: TWideStrings); virtual; overload;
```

C++

```
virtual __fastcall GetFieldNames(TStrings List);
virtual __fastcall GetFieldNames(TWideStrings List);
```

Description

Call GetFieldNames to get a list of names for all fields in a dataset. List is a TStrings object created and maintained by the application. For example, to load a list box with the field names of a table,

```
SQLDataSet1.GetFieldNames(ListBox1.Items);
SQLDataSet1->GetFieldNames(ListBox1->Items);
```

Retrieving a list of field names is especially useful for applications that work with datasets whose field objects are created dynamically at runtime. By retrieving a list of field names, the application can be restricted to working only with fields that exist at runtime.

Note: GetFieldNames does not add the names of any aggregated fields to the list.

See Also

FieldDefs (see page 7019)

Fields (see page 7023)

GetFieldList (see page 6953)

39.1.17.4.89 TDataSet.GetIndexDefs Method

Pascal

```
function GetIndexDefs(IndexDefs: TIndexDefs; IndexTypes: TIndexOptions): TIndexDefs;
```

C++

```
__fastcall TIndexDefs GetIndexDefs(TIndexDefs IndexDefs, TIndexOptions IndexTypes);
```

Description

This is GetIndexDefs, a member of class TDataSet.

39.1.17.4.90 TDataSet.GetIsIndexField Method

Pascal

```
function GetIsIndexField(Field: TField): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetIsIndexField(TField Field);
```

Description

This is GetIsIndexField, a member of class TDataSet.

39.1.17.4.91 TDataSet.GetNextRecord Method

Pascal

```
function GetNextRecord: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetNextRecord();
```

Description

This is GetNextRecord, a member of class TDataSet.

39.1.17.4.92 TDataSet.GetNextRecords Method

Pascal

```
function GetNextRecords: Integer; virtual;
```

C++

```
virtual __fastcall int GetNextRecords();
```

Description

This is GetNextRecords, a member of class TDataSet.

39.1.17.4.93 TDataSet.GetPriorRecord Method

Pascal

```
function GetPriorRecord: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetPriorRecord();
```

Description

This is GetPriorRecord, a member of class TDataSet.

39.1.17.4.94 TDataSet.GetPriorRecords Method

Pascal

```
function GetPriorRecords: Integer; virtual;
```

C++

```
virtual __fastcall int GetPriorRecords();
```

Description

This is GetPriorRecords, a member of class TDataSet.

39.1.17.4.95 TDataSet.GetRecNo Method

Pascal

```
function GetRecNo: Integer; virtual;
```

C++

```
virtual __fastcall int GetRecNo();
```

Description

This is GetRecNo, a member of class TDataSet.

39.1.17.4.96 TDataSet.GetRecord Method

Pascal

```
function GetRecord(Buffer: PChar; GetMode: TGetMode; DoCheck: Boolean): TGetResult; virtual;
abstract;
```

C++

```
virtual __fastcall TGetResult GetRecord(const char * Buffer, TGetMode GetMode, Boolean
DoCheck) = 0;
```

Description

This is GetRecord, a member of class TDataSet.

39.1.17.4.97 TDataSet.GetRecordCount Method

Pascal

```
function GetRecordCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetRecordCount();
```

Description

This is GetRecordCount, a member of class TDataSet.

39.1.17.4.98 TDataSet.GetRecordSize Method

Pascal

```
function GetRecordSize: Word; virtual;
```

C++

```
virtual __fastcall Word GetRecordSize();
```

Description

This is GetRecordSize, a member of class TDataSet.

39.1.17.4.99 TDataSet.GetStateFieldValue Method

Pascal

```
function GetStateFieldValue(State: TDataSetState; Field: TField): Variant; virtual;
```

C++

```
virtual __fastcall Variant GetStateFieldValue(TDataSetState State, TField Field);
```

Description

This is GetStateFieldValue, a member of class TDataSet.

39.1.17.4.100 TDataSet.GotoBookmark Method

Implements a virtual method to position the dataset on the record to which a specified bookmark points.

Pascal

```
procedure GotoBookmark(Bookmark: TBookmark);
```

C++

```
__fastcall GotoBookmark(TBookmark Bookmark);
```

Description

GotoBookmark calls an internal method that is an empty stub in TDataSet (see page 6894). Descendants of TDataSet (see page 6894) redeclare and implement the internal method so that GotoBookmark makes the record identified by the Bookmark (see page 7009) parameter active.

Note: Unidirectional datasets do not support bookmarks. Calling GotoBookmark on a unidirectional dataset does not change the active record.

Delphi Examples:

```
{
This example uses a button to copy the value of a field in
the previous record into the corresponding field in the
current record.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   SavePlace: TBookmark;
   PrevValue: Variant;
begin
   with Customers do
   begin
    { get a bookmark so that we can return to the same record }
    SavePlace := GetBookmark;
    try
      { move to prior record}
      FindPrior;
      { get the value }
      PrevValue := Fields[1].Value;
      {Move back to the bookmark
      this may not be the next record anymore
      if something else is changing the dataset asynchronously }
      GotoBookmark(SavePlace);
      { Set the value }
      Edit;
      Fields[1].Value := PrevValue;
      { Free the bookmark }
    finally
      FreeBookmark(SavePlace);
    end;
  end;
end;
```

```
{
To ensure that the button is disabled when there is no
previous record, the OnDataChange event of the DataSource
detects when the user moves to the beginning of file (BOF
property becomes true), and disables the button.  Detection
occurs on scrolling and editing, not selection with the mouse.
}
procedure TForm1.DS2DataChange(Sender: TObject; Field: TField);
begin
  if Customers.Bof then
    Button1.Enabled := False
  else
    Button1.Enabled := True;
end;
{
The following example copies the selected rows in a db grid
to a list box.  Set the db grid Options dgRowSelect,
dgAlwaysShowSelection and dgMultiSelect to True.  Make a
multiple selecton using the CNTL key.  This example requires
a TDataSet associated with a TDataSource and a TDBGrid.}
procedure TForm1.Button2Click(Sender: TObject);
var
  i, j: Integer;
  s: string;
begin
  if DBGrid2.SelectedRows.Count>0 then
    with DBGrid2.DataSource.DataSet do
      for i:=0 to DBGrid2.SelectedRows.Count-1 do
      begin
        GotoBookmark(pointer(DBGrid2.SelectedRows.Items[i]));
        for j := 0 to FieldCount-1 do
        begin
          if (j>0) then s:=s+', ';
          s:=s+Fields[j].AsString;
        end;
        Listbox1.Items.Add(s);
        s:= '';
      end;
end;
```

See Also

Bookmark (see page 7009)

FreeBookmark (see page 6945)

GetBookmark (see page 6948)

39.1.17.4.101 TDataSet.InitFieldDefs Method

Pascal

```
procedure InitFieldDefs; virtual;
```

C++

```
virtual __fastcall InitFieldDefs();
```

Description

This is InitFieldDefs, a member of class TDataSet.

39.1.17.4.102 TDataSet.InitFieldDefsFromFields Method

Pascal

```
procedure InitFieldDefsFromFields;
```

C++

```
__fastcall InitFieldDefsFromFields();
```

Description

This is InitFieldDefsFromFields, a member of class TDataSet.

39.1.17.4.103 TDataSet.InitRecord Method

Pascal

```
procedure InitRecord(Buffer: PChar); virtual;
```

C++

```
virtual __fastcall InitRecord(const char * Buffer);
```

Description

This is InitRecord, a member of class TDataSet.

39.1.17.4.104 TDataSet.Insert Method

Inserts a new, empty record in the dataset.

Pascal

```
procedure Insert;
```

C++

```
__fastcall Insert();
```

Description

Call Insert to:

1Open (see page 6976) a new, empty record in the dataset.

2Set the active record to the new record.

After a call to Insert, an application can allow users to enter data in the fields of the record, and then post those changes to the database or change log using Post (see page 6977). (Client datasets can subsequently apply updates in the change log back to the database by calling ApplyUpdates.) A newly inserted record is posted in one of three ways:

For Paradox tables with primary indexes, the record is inserted into the dataset in a position based on its index.

For Paradox tables without primary indexes, the record is inserted into the dataset at the current position.

For dBASE, FoxPro, and Access tables, the record is physically appended to the dataset at the end. If an index happens to be active, the new record may appear in a position relative to the index, but the record is still actually stored at the end of the table.

For SQL databases, the physical location of the insert is implementation-specific. For indexed tables, the index is updated with the new record information.

After the new record is applied back to the database server, its physical location is database-specific. For indexed tables, the index is updated with the new record information.

Delphi Examples:

```
{
This example uses the BeforeInsert event to do data
validation. If the StrToInt function raises an exception,
the edit control’s contents are set to a valid value so
that the assignment to the INTEGER field in the dataset
will succeed.  Continuing after the exception causes a
'0' to be placed in the PORTA field.
DataSet design:
This example requires a ClientDataSet, a DBGrid, a
DataSource and a DBNavigator.  Name the ClientDataSet CDS.
Set the DataSet property in the DataSource to CDS and name
that DataSource DS.  Set the DataSource in the DBNavigator
to DS the DataSource in the DBGrid to DS.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
type to Integer.  The file used to load CDS must have an
attribute name of HOST with an attribute type of string and
an attribute name of PORTA with an attribute type of integer.
Add DB, DBCtrls, Grids, DBGrids and DBClient to the uses clause.
}
{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button3Click(Sender: TObject);
begin
  CDS.Insert;
  CDS.FieldByName('PORTA').AsInteger := StrToInt(Edit1.Text);
  CDS.Post;
end;

procedure TForm1.CDSBeforeInsert(DataSet: TDataSet);
begin
  try
  { Make sure edit field can be converted to integer --
   this will raise an exception if it can’t }
    StrToInt(Edit1.Text);
  except
    Edit1.Text := '0';
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;
```

See Also

Append (see page 6910)

ApplyUpdates (see page 8394)

InsertRecord (see page 6961)

Post (see page 6977)

Adding New Records

Posting Data

39.1.17.4.105 TDataSet.InsertRecord Method

Inserts a new, populated record to the dataset and posts it.

Pascal

```
procedure InsertRecord(const Values: array of const);
```

C++

```
__fastcall InsertRecord(const array of const Values);
```

Description

Call InsertRecord to create a new, empty record in the dataset, populate it with the field values in Values, and post the values to the database or change log. In C++, the Values_Size parameter indicates the index of the last value in the Values array (one less than the number of values).

A newly inserted record is posted to the database in one of three ways:

For indexed Paradox and dBASE tables, the record is inserted into the dataset in a position based on its index.

For unindexed Paradox tables, the record is inserted into the dataset at the current position.

For unindexed dBASE, FoxPro, and Access tables, the record is inserted into the dataset at the end.

For SQL databases, the physical location of the insert is implementation-specific. For indexed tables, the index is updated with the new record information.

The newly inserted record becomes the active record.

Delphi Examples:

```
{
This statement inserts a record to the MyCustInfo data. Note
that Nulls are entered for some of the values, but are not
required for missing values at the end of the array argument,
i.e. after the Field3 field.  InsertRecord behaves differently
depending on the indexing of the table.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Customers.InsertRecord([Edit1.Text, Edit2.Text, Null, Edit3.Text]);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Customers:= TTable.Create(Form1);
  with Customers do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'MyCustInfo';
    ReadOnly:= False;

//    Don't overwrite an existing table
    if (not Customers.Exists) then CreateMyTable
    else
    begin
      if (Customers.Exists AND
        (MessageDlg('MyCustInfo table already exists.  Do you want to rebuild it?',
mtConfirmation, [mbYes, mbNo], 0) = mrYes)) then
```

```
        begin
          Customers.Close;
          Customers.DeleteTable;
          CreateMyTable;
        end;
      end;
    end;
    DS2.DataSet:= Customers;
    DBGrid2.DataSource.DataSet:= Customers;
    Customers.Active:= True;
  end;

  procedure CreateMyTable();
  var
    i: Integer;
  begin
  //      The Table component must not be active
  //      First, describe the type of table and give
  //      it a name
  //      Next, describe the fields in the table
    with Customers do
    begin
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
          with AddFieldDef do
          begin
            Name := 'Field3';
            DataType := ftString;
            Size := 30;
          end;
          with AddFieldDef do
          begin
            Name := 'Field4';
            DataType := ftString;
            Size := 30;
          end;
        end;

  //    Take out this part to remove indexing
  //    Next, describe any indexes
        with IndexDefs do
        begin
          Clear;
  //        The 1st index has no name because it is
  //        a Paradox primary key
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
```

```
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;

//      Call the CreateTable method to create the table
        CreateTable;
        Customers.Active:= True;
        for i := 1 to 20 do
          Customers.AppendRecord([i*111, i*222, i*333, i*444]);
      end;
end;
```

See Also

AppendRecord (see page 6911)

Insert (see page 6959)

Post (see page 6977)

SetFields (see page 6997)

Modifying Entire Records

39.1.17.4.106 TDataSet.InternalAddRecord Method

Pascal

```
procedure InternalAddRecord(Buffer: Pointer; Append: Boolean); virtual;
```

C++

```
virtual __fastcall InternalAddRecord(void * Buffer, Boolean Append);
```

Description

This is InternalAddRecord, a member of class TDataSet.

39.1.17.4.107 TDataSet.InternalCancel Method

Pascal

```
procedure InternalCancel; virtual;
```

C++

```
virtual __fastcall InternalCancel();
```

Description

This is InternalCancel, a member of class TDataSet.

39.1.17.4.108 TDataSet.InternalClose Method

Pascal

```
procedure InternalClose; virtual; abstract;
```

C++

```
virtual __fastcall InternalClose() = 0;
```

Description

This is InternalClose, a member of class TDataSet.

39.1.17.4.109 TDataSet.InternalDelete Method

Pascal

```
procedure InternalDelete; virtual;
```

C++

```
virtual __fastcall InternalDelete();
```

Description

This is InternalDelete, a member of class TDataSet.

39.1.17.4.110 TDataSet.InternalEdit Method

Pascal

```
procedure InternalEdit; virtual;
```

C++

```
virtual __fastcall InternalEdit();
```

Description

This is InternalEdit, a member of class TDataSet.

39.1.17.4.111 TDataSet.InternalFirst Method

Pascal

```
procedure InternalFirst; virtual;
```

C++

```
virtual __fastcall InternalFirst();
```

Description

This is InternalFirst, a member of class TDataSet.

39.1.17.4.112 TDataSet.InternalGotoBookmark Method

Pascal

```
procedure InternalGotoBookmark(Bookmark: Pointer); virtual;
```

C++

```
virtual __fastcall InternalGotoBookmark(void * Bookmark);
```

Description

This is InternalGotoBookmark, a member of class TDataSet.

39.1.17.4.113 TDataSet.InternalHandleException Method

Pascal

```
procedure InternalHandleException; virtual; abstract;
```

C++

```
virtual __fastcall InternalHandleException() = 0;
```

Description

This is InternalHandleException, a member of class TDataSet.

39.1.17.4.114 TDataSet.InternalInitFieldDefs Method

Pascal

```
procedure InternalInitFieldDefs; virtual; abstract;
```

C++

```
virtual __fastcall InternalInitFieldDefs() = 0;
```

Description

This is InternalInitFieldDefs, a member of class TDataSet.

39.1.17.4.115 TDataSet.InternalInitRecord Method

Pascal

```
procedure InternalInitRecord(Buffer: PChar); virtual;
```

C++

```
virtual __fastcall InternalInitRecord(const char * Buffer);
```

Description

This is InternalInitRecord, a member of class TDataSet.

39.1.17.4.116 TDataSet.InternalInsert Method

Pascal

```
procedure InternalInsert; virtual;
```

C++

```
virtual __fastcall InternalInsert();
```

Description

This is InternalInsert, a member of class TDataSet.

39.1.17.4.117 TDataSet.InternalLast Method

Pascal

```
procedure InternalLast; virtual;
```

C++

```
virtual __fastcall InternalLast();
```

Description

This is InternalLast, a member of class TDataSet.

39.1.17.4.118 TDataSet.InternalOpen Method

Pascal

```
procedure InternalOpen; virtual; abstract;
```

C++

```
virtual __fastcall InternalOpen() = 0;
```

Description

This is InternalOpen, a member of class TDataSet.

39.1.17.4.119 TDataSet.InternalPost Method

Pascal

```
procedure InternalPost; virtual;
```

C++

```
virtual __fastcall InternalPost();
```

Description

This is InternalPost, a member of class TDataSet.

39.1.17.4.120 TDataSet.InternalRefresh Method

Pascal

```
procedure InternalRefresh; virtual;
```

C++

```
virtual __fastcall InternalRefresh();
```

Description

This is InternalRefresh, a member of class TDataSet.

39.1.17.4.121 TDataSet.InternalSetToRecord Method

Pascal

```
procedure InternalSetToRecord(Buffer: PChar); virtual;
```

C++

```
virtual __fastcall InternalSetToRecord(const char * Buffer);
```

Description

This is InternalSetToRecord, a member of class TDataSet.

39.1.17.4.122 TDataSet.IsCursorOpen Method

Pascal

```
function IsCursorOpen: Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean IsCursorOpen() = 0;
```

Description

This is IsCursorOpen, a member of class TDataSet.

39.1.17.4.123 TDataSet.IsEmpty Method

Indicates whether the dataset contains no records.

Pascal

```
function IsEmpty: Boolean;
```

C++

```
__fastcall Boolean IsEmpty();
```

Description

Call IsEmpty to determine if a dataset has records. IsEmpty returns true if the dataset does not contain any records. Otherwise it returns false.

See Also

Bof (see page 7008)

Eof (see page 7015)

39.1.17.4.124 TDataSet.IsLinkedTo Method

Indicates whether a dataset is linked to a specified data source.

Pascal

```
function IsLinkedTo(DataSource: TDataSource): Boolean;
```

C++

```
__fastcall Boolean IsLinkedTo(TDataSource DataSource);
```

Description

Call IsLinkedTo to verify that a dataset provides data through a specific data source. DataSource (see page 7014) is the data source component to test.

Note: IsLinkedTo is mainly provided for developers deriving custom components based on TDataSet (see page 6894). It is not intended or needed for general data access.

If the datasource provides data from the dataset or one of its nested dataset fields (or a nested dataset nested in a dataset field...), IsLinkedTo returns true. If the datasource provides data from some other dataset, or if the data source does not already have a dataset of its own, IsLinkedTo returns false.

Note: IsLinkedTo is mainly provided for developers deriving custom components based on TDataSet (see page 6894). It is not intended or needed for general data access.

See Also

DataSource (see page 7014)

39.1.17.4.125 TDataSet.IsSequenced Method

Indicates whether the underlying database table uses record numbers to indicate the order of records.

Pascal

```
function IsSequenced: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsSequenced();
```

Description

Use IsSequenced to determine whether the underlying database table supports sequence numbers, or whether these are computed by the dataset component. When IsSequenced returns true, applications can safely use the RecNo (see page 7031) property to navigate to records in the dataset.

As implemented in TDataSet (see page 6894), IsSequenced always returns true. Descendants of TDataSet (see page 6894) reimplement this method to return a value that depends on the underlying table type.

See Also

RecNo (see page 7031)

RecordCount (see page 7032)

39.1.17.4.126 TDataSet.Last Method

Moves to the last record in the dataset.

Pascal

```
procedure Last;
```

C++

```
__fastcall Last();
```

Description

Call Last to make the last record in the dataset active. If the dataset is not unidirectional Last posts any changes to the active record and

Clears the record buffers.

Fetches the last record and makes it the active record.

Fetches any additional records required for display, such as those needed to fill out a grid control.

Sets the Eof (see page 7015) property to true.

Broadcasts the record change so that data controls and linked detail sets can update.

For unidirectional datasets, Last raises an EDatabaseError (see page 6731) exception.

Note: TDataSet (see page 6894) uses internal, protected methods to reposition the active record and to fetch additional records required for display. In TDataSet (see page 6894), these internal methods are empty stubs. Descendant classes implement these methods to enable the Last method to work.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
```

```
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
value of the first field for the last record to the server so
that the server can return the correct records. The complete
project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
var
  LastValue: OleVariant;
  CDSClone: TClientDataSet;
begin
  if ClientDataSet1.Active then
  begin
    CDSClone := TClientDataSet.Create(Form1);
    try
      CDSClone.CloneCursor(ClientDataSet1, True);
      { turn off FetchOnDemand so that the clone only fetches
        the last LOCAL record }
      CDSClone.FetchOnDemand := False;
      CDSClone.Last;
      LastValue := CDSClone.Fields[0].AsString;
      CDSClone.Close;
    finally
      CDSClone.Free;
    end;
  end else
    LastValue := NULL;
  OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
  ClientDataSet1.GetNextPacket;
end;
```

See Also

Adding New Records

Posting Data

Using the First and Last Methods

39.1.17.4.127 TDataSet.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState (see page 2804)

Create (see page 2766)

TReader (see page 2933)

39.1.17.4.128 TDataSet.Locate Method

Implements a virtual method for searching a dataset for a specified record and making it the active record.

Pascal

```
function Locate(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions):
Boolean; virtual;
```

C++

```
virtual __fastcall Boolean Locate(const AnsiString KeyFields, const Variant KeyValues,
TLocateOptions Options);
```

Description

This function:

Checks whether the dataset is unidirectional, and if so, raises an EDatabaseError (see page 6731) exception.

Returns false, indicating that a matching record was not found and the active record was not changed.

Descendant classes that are not unidirectional override this method so that it locates the record where the fields identified by the semicolon-separated list of fields in KeyFields have the values specified by the Variant or Variant array KeyValues. Options indicates whether the search is case insensitive and whether partial matches are supported. Locate returns true if a record is found that matches the specified criteria and that record is now active.

See Also

FindField (see page 6938)

Lookup (see page 6970)

Using Indexes to Search for Records

Using Locate

39.1.17.4.129 TDataSet.Lookup Method

Implements a virtual method to retrieve field values from a record that matches specified search values.

Pascal

```
function Lookup(const KeyFields: string; const KeyValues: Variant; const ResultFields:
string): Variant; virtual;
```

C++

```
virtual __fastcall Variant Lookup(const AnsiString KeyFields, const Variant KeyValues, const
AnsiString ResultFields);
```

Description

This function:

Checks whether the dataset is unidirectional, and if so, raises an EDatabaseError (see page 6731) exception.

Returns a Variant with the value false, indicating that a matching record was not found.

Descendant classes that are not unidirectional override this method so that it locates the record where the fields identified by the comma-delimited string KeyFields have the values specified by the Variant or Variant array KeyValues. In classes that implement Lookup, it returns a Variant or Variant array that contains the value or values of the fields specified by the comma-delimited string ResultFields on the specified record.

See Also

Locate (see page 6970)

Using Indexes to Search for Records

Using Locate

Using Lookup

39.1.17.4.130 TDataSet.MoveBy Method

Moves to another record relative to the active record in the dataset.

Pascal

```
function MoveBy(Distance: Integer): Integer;
```

C++

```
__fastcall int MoveBy(int Distance);
```

Description

Call MoveBy to move the active record by the number of records specified by Distance. A positive value for Distance indicates forward progress through the dataset, while a negative value indicates backward progress. For example, the following statement moves backward through the dataset by 10 records:

```
MoveBy(-10);
DataSet1->MoveBy(-10);
```

MoveBy posts any changes to the active record and

Sets the Bof (see page 7008) and Eof (see page 7015) properties to false.

If Distance is positive, repeatedly fetches Distance subsequent records (if possible), and makes the last record fetched active. If an attempt is made to move past the end of the file, MoveBy sets Eof (see page 7015) to true.

If Distance is negative, repeatedly fetches the appropriate number of previous records (if possible), and makes the last record fetched active. If an attempt is made to move past the start of the file, MoveBy sets Bof (see page 7008) to true. If the dataset is unidirectional, the dataset raises an EDatabaseError (see page 6731) exception when MoveBy tries to fetch a prior record.

Broadcasts information about the record change so that data-aware controls and linked datasets can update.

Returns the number of records moved. In most cases, Result is the absolute value of Distance, but if MoveBy encounters the beginning-of-file or end-of-file before moving Distance records, Result will be less than the absolute value of Distance.

C++ Examples:

```
/*
The following example enables the user to move the current
selected cell in a db grid.  The Up and Down buttons have
their OnClick events assigned to the UpDownClick procedure.
The Left and Right buttons have their OnClick events
assigned to the LeftRightClick procedure.  The Up and Left
buttons have their Tag property set to -1, while the Down
and Right buttons have their Tag property set to 1.  MoveBy
does not move the cell in the db grid, it only moves the
selection.  The data set is not edited.  Notice that the
Up button and the Down button are separate TButtons, not
one TUpDown.  The tags must be set different.
*/
void __fastcall TForm1::UpDownClick(TObject *Sender)
{
  CDS2->MoveBy(((TComponent *)Sender)->Tag);
  DBGrid2->SetFocus();
}

void __fastcall TForm1::LeftRightClick(TObject *Sender)
{
  DBGrid2->SelectedIndex = DBGrid2->SelectedIndex + ((TComponent *)Sender)->Tag;
  DBGrid2->SetFocus();
}
```

Delphi Examples:

```
{
The following example enables the user to move the current
selected cell in a db grid.  The Up and Down buttons have
their OnClick events assigned to the UpDownClick procedure.
The Left and Right buttons have their OnClick events
assigned to the LeftRightClick procedure.  The Up and Left
buttons have their Tag property set to -1, while the Down
and Right buttons have their Tag property set to 1.  MoveBy
does not move the cell in the db grid, it only moves the
selection.  The data set is not edited.  Notice that the
Up button and the Down button are separate TButtons, not
one TUpDown.  The tags must be set different.
}
procedure TForm1.UpDownClick(Sender: TObject);
begin
  CDS.MoveBy(TComponent(Sender).Tag);
  DBGrid1.SetFocus;
end;

procedure TForm1.LeftRightClick(Sender: TObject);
begin
  DBGrid1.SelectedIndex := DBGrid1.SelectedIndex + TComponent(Sender).Tag;
  DBGrid1.SetFocus;
end;
```

See Also

First (see page 6942)

Last (see page 6968)

Next (see page 6973)

Prior (see page 6979)

Using the MoveBy Method

39.1.17.4.131 TDataSet.Next Method

Moves to the next record in the dataset.

Pascal

```
procedure Next;
```

C++

```
__fastcall Next();
```

Description

Call Next to move to the next record in the dataset, making it the active record. Next posts any changes to the active record and

Sets the Bof (see page 7008) and Eof (see page 7015) properties to false.

Fetches the next record and makes it the active record.

If the dataset is not unidirectional, fetches any additional records required for display, such as those needed to fill out a grid control.

Sets the Eof (see page 7015) property to true if the last record in the dataset was already active.

Broadcasts the record change so that data controls and linked detail sets can update.

Note: TDataSet (see page 6894) uses internal, protected methods to move the active record and to fetch additional records required for display. In TDataSet (see page 6894), these internal methods are empty stubs. Descendant classes implement these methods to enable the Next method to work.

C++ Examples:

```
/*
Reads through all records in the Customers table.
Updates the ProgressBar accordingly.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ProgressBar1->Min = 0;
  ProgressBar1->Max = Customers->RecordCount;
  Customers->First();
  for (int i = ProgressBar1->Min; i <= ProgressBar1->Max; i++)
  {
    ProgressBar1->Position = i;
    Customers->Next();
    // do something with the current record
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Customers = new TTable(Form1);
  Customers->Active = false; // The Table component must not be active
  Customers->DatabaseName = "DBDEMOS";
  Customers->TableType = ttParadox;
  Customers->TableName = "CustInfo";
  Customers->Active = False;
  if (Customers->Exists) // Don't overwrite an existing table
  {
    Customers->Close();
    Customers->DeleteTable();
  }
  // The Table component must not be active
```

```
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Customers->FieldDefs->Clear();
  TFieldDef *pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Customers->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Customers->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Customers->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Customers->CreateTable();
  Customers->Active = True;
  for (int i = 1; i <= 20; i++)
    Customers->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  DS2->DataSet = Customers;
  DBGrid2->DataSource->DataSet = Customers;
  Customers->Active = True;
}
```

Delphi Examples:

```
{
Reads through all records in the Customers table.
Updates the ProgressBar accordingly.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
begin
  with ProgressBar1 do
  begin
    Min := 0;
    Max := Customers.RecordCount;
    Customers.First;
    for i := Min to Max do
    begin
      Position := i;
      Customers.Next;
      // do something with the current record
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Customers:= TTable.Create(Form1);
  with Customers do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Active := False;
//  Overwrite any existing table
    if Customers.Exists then
    begin
      Customers.Close;
      Customers.DeleteTable;
```

```
      end;
      begin
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
        { Next, describe the fields in the table }
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Customers.Active:= True;
        for i := 1 to 100 do
          Customers.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Customers;
    DBGrid2.DataSource.DataSet:= Customers;
    Customers.Active:= True;
  end;
```

See Also

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Prior (see page 6979)

Adding New Records

Posting Data

Using the Eof and Bof Properties

Using the Next and Prior Methods

Using Unidirectional Datasets

39.1.17.4.132 TDataSet.Open Method

Opens the dataset.

Pascal

```
procedure Open;
```

C++

```
__fastcall Open();
```

Description

Call Open to set the Active (see page 7004) property for the dataset to true. When Active (see page 7004) is true, data can be populated with data. It can read data from a database or other source (such as a provider). Depending on the CanModify (see page 7012) property, active datasets can post changes.

Setting Active (see page 7004) to true:

Triggers the BeforeOpen (see page 7051) event handler if one is defined for the dataset.

Sets the dataset state to dsBrowse.

Establishes a way to fetch data (typically by opening a cursor).

Triggers the After Open event handler if one is defined for the dataset.

If an error occurs during the dataset open, dataset state is set to dsInactive, and any cursor is closed.

See Also

DataSource (see page 7014)

State (see page 7036)

AfterOpen (see page 7045)

BeforeOpen (see page 7051)

Close (see page 6920)

CanModify (see page 7012)

Opening and Closing Datasets

Fetching the Data

39.1.17.4.133 TDataSet.OpenCursor Method

Pascal

```
procedure OpenCursor(InfoQuery: Boolean = False); virtual;
```

C++

```
virtual __fastcall OpenCursor(Boolean InfoQuery = False);
```

Description

This is OpenCursor, a member of class TDataSet.

39.1.17.4.134 TDataSet.OpenCursorComplete Method

Pascal

```
procedure OpenCursorComplete;
```

C++

```
__fastcall OpenCursorComplete();
```

Description

This is OpenCursorComplete, a member of class TDataSet.

39.1.17.4.135 TDataSet.OpenParentDataSet Method

Pascal

```
procedure OpenParentDataSet(ParentDataSet: TDataSet);
```

C++

```
__fastcall OpenParentDataSet(TDataSet ParentDataSet);
```

Description

This is OpenParentDataSet, a member of class TDataSet.

39.1.17.4.136 TDataSet.Post Method

Implements a virtual method to write a modified record to the database or change log.

Pascal

```
procedure Post; virtual;
```

C++

```
virtual __fastcall Post();
```

Description

TDataSet (see page 6894) implements a virtual method to write a modified record to the database or change log. Dataset methods that change the dataset state, such as Edit (see page 6932), Insert (see page 6959), or Append (see page 6910), or that move from one record to another, such as First (see page 6942), Last (see page 6968), Next (see page 6973), and Prior (see page 6979) automatically call Post.

Different types of datasets handle posting differently:

BDE-enabled datasets post records directly to the database server unless CachedUpdates is true. When caching updates, BDE-enabled datasets post records to an internal change log until they are applied to the database by calling ApplyUpdates.

ADO and InterBaseExpress datasets post records directly to the database server.

Client datasets post records to an internal change log until they are applied to the database by calling ApplyUpdates or merged with the client dataset's data by calling MergeChangeLog.

Unidirectional datasets are read-only and do not post records at all.

Designers of custom datasets can choose whether to implement Post by writing records to the database server or to an internal change log.

Delphi Examples:

```
{
```

```
This example appends a new record to a table or client
dataset when the user clicks a button. The two fields
Field1 and Field2 are filled from the contents of two edit
controls.  This example requires a populated TTable or
TDataSet with the appropriate field values.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Customers.Append;
  Customers.FieldValues['Field1'] := StrToInt(Edit1.text);
  Customers.FieldValues['Field2'] := Edit2.text;
  Customers.Post;
end;
{
The following fragment prompts the user to confirm changes
to a record; if the user clicks Yes, the record is posted to
the table, otherwise the changes are cancelled.
}
procedure TForm1.Button2Click(Sender: TObject);
begin
if MessageDlg('Update Record?', mtConfirmation, [mbYes, mbNo], 0) =
              mrYes then
  Table1.Post
else
  Table1.Cancel;
end;
```

See Also

State (see page 7036)

AfterPost (see page 7046)

BeforePost (see page 7052)

Append (see page 6910)

Edit (see page 6932)

First (see page 6942)

Insert (see page 6959)

Last (see page 6968)

Next (see page 6973)

Prior (see page 6979)

Resync (see page 6994)

UpdateRecord (see page 7002)

CachedUpdates (see page 9932)

ApplyUpdates (see page 9894)

ApplyUpdates (see page 8394)

MergeChangeLog (see page 8437)

Responding to Changes Mediated by the Data Source

Adding New Records

Calculating Fields

Canceling Changes

Editing Records

Modifying Entire Records

Posting Data

39.1.17.4.137 TDataSet.Prior Method

Moves to the previous record in the dataset.

Pascal

```
procedure Prior;
```

C++

```
__fastcall Prior();
```

Description

Call Prior to move to the previous record in the dataset, making it the active record. Prior posts any changes to the active record and

Sets the Bof (see page 7008) and Eof (see page 7015) properties to false.

Fetches the previous record and makes it the active record. If the dataset is unidirectional, it raises an EDatabaseError (see page 6731) exception at this point.

Fetches any additional records required for display, such as those needed to fill out a grid control.

Sets the Bof (see page 7008) property to true if the first record in the dataset was already active.

Broadcasts the record change so that data controls and linked detail sets can update.

Note: TDataSet (see page 6894) uses internal, protected methods to move the active record and to fetch additional records required for display. In TDataSet (see page 6894), these internal methods are empty stubs. Descendant classes implement these methods to enable the Prior method to work.

See Also

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Posting Data

Using the Eof and Bof Properties

Using the Next and Prior Methods

39.1.17.4.138 TDataSet.PSEndTransaction Method

Ends a transaction that was started by a call to PSStartTransaction.

Pascal

```
procedure PSEndTransaction(Commit: Boolean); virtual;
```

C++

```
virtual __fastcall PSEndTransaction(Boolean Commit);
```

Description

The provider component applies updates within a transaction, if possible. To do so, it uses PSEndTransaction in conjunction with the PSStartTransaction method.

The Commit parameter indicates whether the dataset should commit the current transaction (Commit is true), or roll it back (Commit is false).

Note: TDataSetProvider only starts a transaction (Using PSStartTransaction) if the PSInTransaction method indicates that there is no transaction already established. That is, it does not assume nested transaction support is available. Thus, there is no need to ensure that PSEndTransaction does not terminate a transaction that was not started by the provider.

See Also

PSStartTransaction (see page 7586)

PSInTransaction (see page 7583)

applyupdates (see page 18459)

39.1.17.4.139 TDataSet.PSExecute Method

Executes the SQL command associated with the dataset.

Pascal

```
procedure PSExecute; virtual;
```

C++

```
virtual __fastcall PSExecute();
```

Description

The provider component calls PSExecute to implement its Execute method. If the dataset represents a query or stored procedure that does not return a cursor, this method executes the query or stored procedure. If the provider Options includes poAllowCommandText and the dataset supports the PSSetCommandText method, the statement specified by PSSetCommandText may be executed instead.

If there is not statement to execute, PSExecute raises an EDatabaseError exception.

See Also

PSSetCommandText (see page 7585)

Execute (see page 18466)

Options (see page 18440)

39.1.17.4.140 PSExecuteStatement Method

This is the overview for the PSExecuteStatement method overload.

Overload List

	Name	Description
	TDataSet.PSExecuteStatement (string, TParams, Pointer) (see page 6981)	Executes a specified SQL command.
	TDataSet.PSExecuteStatement (WideString, TParams, Pointer) (see page 6981)	This is PSExecuteStatement, a member of class TDataSet.

39.1.17.4.140.1 TDataSet.PSExecuteStatement Method (WideString, TParams, Pointer)

Pascal

```
function PSExecuteStatement(const ASQL: WideString; AParams: TParams; ResultSet: Pointer =
nil): Integer; virtual; overload;
```

C++

```
virtual __fastcall int PSExecuteStatement(const BSTR ASQL, TParams AParams, void * ResultSet =
nil);
```

Description

This is PSExecuteStatement, a member of class TDataSet.

39.1.17.4.140.2 TDataSet.PSExecuteStatement Method (string, TParams, Pointer)

Executes a specified SQL command.

Pascal

```
function PSExecuteStatement(const ASQL: string; AParams: TParams; ResultSet: Pointer = nil):
Integer; virtual; overload; deprecated;
```

C++

```
virtual __fastcall int PSExecuteStatement(const AnsiString ASQL, TParams AParams, void *
ResultSet = nil);
```

Description

If the value its ResolveToDataSet property is false, a provider component calls PSExecuteStatement (indirectly) to execute SQL statements that it generates for applying updates. Usually, the dataset passes these statements on to its database server for execution.

The ASQL parameter specifies the SQL command to execute.

AParams contains any parameter values that should be applied to ASQL before execution.

ResultSet allows PSExecuteStatement to return a dataset component that contains the results of a SELECT statement. If ResultSet is not nil (Delphi) or NULL (C++), PSExecuteStatement creates a new dataset component as the target of this pointer and fills it with the result set. The caller is responsible for freeing the dataset.

PSExecuteStatement returns the number of rows affected by executing ASQL.

See Also

PSUpdateRecord (see page 7586)

applyupdates (see page 18459)

ResolveToDataSet (see page 18531)

39.1.17.4.141 TDataSet.PSGetAttributes Method

Returns information that is stored in data packets as Name/Value pairs.

Pascal

```
procedure PSGetAttributes(List: TList); virtual;
```

C++

```
virtual __fastcall PSGetAttributes(TList List);
```

Description

The provider component calls PSGetAttributes to add information about the dataset to the metadata stored with generated data packets. This information is stored as Name/Value pairs that are added to the list specified by the List parameter.

See Also

DoGetProviderAttributes (see page 18521)

39.1.17.4.142 TDataSet.PSGetCommandText Method

Gets the SQL command to be executed by PSExecute.

Pascal

```
function PSGetCommandText: string; virtual; deprecated;
```

C++

```
virtual __fastcall AnsiString PSGetCommandText();
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

PSSetCommandText (see page 7585)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

39.1.17.4.143 TDataSet.PSGetCommandTextW Method

Pascal

```
function PSGetCommandTextW: WideString; virtual;
```

C++

```
virtual __fastcall BSTR PSGetCommandTextW();
```

Description

This is PSGetCommandTextW, a member of class TDataSet.

39.1.17.4.144 TDataSet.PSGetCommandType Method

Gets the command type to be executed by PSExecute.

Pascal

```
function PSGetCommandType: TPSCommandType; virtual;
```

C++

```
virtual __fastcall TPSCommandType PSGetCommandType();
```

Description

PSGetCommandType gets the command type executed by PSExecute. The type may be a type of SQL command or other command types, such as stored procedures.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

PSGetCommandText (see page 7579)

Execute (see page 18466)

Options (see page 18440)

39.1.17.4.145 TDataSet.PSGetDefaultOrder Method

Returns the definition of an index that imposes the default order on the data included in data packets.

Pascal

```
function PSGetDefaultOrder: TIndexDef; virtual;
```

C++

```
virtual __fastcall TIndexDef PSGetDefaultOrder();
```

Description

The provider component calls PSGetDefaultOrder to generate a default index, called DEFAULT_ORDER, for the records that appear in data packets. This index defines the default sort order for records when they appear in the client dataset on the client.

PSGetDefaultOrder creates a TIndexDef object to describe the default index. The provider that calls PSGetDefaultOrder subsequently frees this TIndexDef object.

See Also

CreateDataPacket (see page 18518)

39.1.17.4.146 TDataSet.PSGetIndexDefs Method

Returns the definitions of all specified indexes defined for the dataset.

Pascal

```
function PSGetIndexDefs(IndexTypes: TIndexOptions): TIndexDefs; virtual;
```

C++

```
virtual __fastcall TIndexDefs PSGetIndexDefs(TIndexOptions IndexTypes);
```

Description

The provider component calls PSGetIndexDefs to locate an index that contains information it needs. For example, it uses PSGetIndexDefs to locate an index that contains the key fields returned by PSGetKeyFields. This information allows the provider to locate the records that need to change when it applies updates. The provider also calls this method to locate indexes that it adds to the metadata of data packets.

IndexTypes indicates the options required of the returned indexes. PSGetIndexDefs returns all indexes whose Options property

includes the specified flags.

PSGetIndexDefs returns a TIndexDefs object that contains all the indexes of the dataset that match the IndexTypes parameter.

See Also

Options (see page 7291)

PSGetKeyFields (see page 7581)

PSGetDefaultOrder (see page 7580)

39.1.17.4.147 TDataSet.PSGetKeyFields Method

Returns the names of all fields required to uniquely identify records in the data packet.

Pascal

```
function PSGetKeyFields: string; virtual; deprecated;
```

C++

```
virtual __fastcall AnsiString PSGetKeyFields();
```

Description

The provider component calls PSGetKeyFields to determine what information is required to uniquely identify records in the data packet. This information allows it to locate the records that need to change when it applies updates.

PSGetKeyFields returns a semicolon-delimited string that lists the names of the key fields of the dataset. By default (as implemented by TDataSet), these are the fields with pfInKey included in their ProviderFlags property.

See Also

PSGetDefaultOrder (see page 7580)

ProviderFlags (see page 7176)

PSGetIndexDefs (see page 7581)

39.1.17.4.148 TDataSet.PSGetKeyFieldsW Method

Pascal

```
function PSGetKeyFieldsW: WideString; virtual;
```

C++

```
virtual __fastcall BSTR PSGetKeyFieldsW();
```

Description

This is PSGetKeyFieldsW, a member of class TDataSet.

39.1.17.4.149 TDataSet.PSGetParams Method

Returns the current parameter values of the dataset.

Pascal

```
function PSGetParams: TParams; virtual;
```

C++

```
virtual __fastcall TParams PSGetParams();
```

Description

The provider component calls PSGetParams to implement its GetParams method.

PSGetParams returns a TParams object that describes the current values of all parameters of the dataset. If the dataset does not have any parameters, PSGetParams returns nil (Delphi) or NULL (C++).

Note: The PSGetParams method should not create a new instance of TParams for the return value. The provider that calls PSGetParams does not free the returned value.

See Also

GetParams (see page 18467)

39.1.17.4.150 TDataSet.PSGetQuoteChar Method

Returns the character or characters to use in generated SQL statements to enclose quoted strings.

Pascal

```
function PSGetQuoteChar: string; virtual; deprecated;
```

C++

```
virtual __fastcall AnsiString PSGetQuoteChar();
```

Description

The provider component calls PSGetQuoteChar (indirectly) when generating the dynamic SQL statements that apply updates. These statements are subsequently executed by a call to PSExecuteStatement.

PSGetQuoteChar returns the character or characters that delimit quoted strings in the generated SQL. This string appears both before and after quoted strings.

See Also

applyupdates (see page 18459)

PSExecuteStatement (see page 7579)

ResolveToDataSet (see page 18531)

39.1.17.4.151 TDataSet.PSGetQuoteCharW Method

Pascal

```
function PSGetQuoteCharW: WideString; virtual;
```

C++

```
virtual __fastcall BSTR PSGetQuoteCharW();
```

Description

This is PSGetQuoteCharW, a member of class TDataSet.

39.1.17.4.152 TDataSet.PSGetTableName Method

Returns the name of the dataset table that appears in generated SQL statements.

Pascal

```
function PSGetTableName: string; virtual; deprecated;
```

C++

```
virtual __fastcall AnsiString PSGetTableName();
```

Description

PSGetTableName returns the name of the underlying database table against which updates are applied when the provider's ResolveToDataSet property is false. This value is used for generating the SQL statements that apply updates. The generated SQL statements are subsequently executed by a call to PSExecuteStatement.

If PSGetTableName does not return a table name, the provider must have an OnGetTableName event handler to supply this value.

See Also

applyupdates (see page 18459)

PSExecuteStatement (see page 7579)

ResolveToDataSet (see page 18531)

ongettablename (see page 18542)

39.1.17.4.153 TDataSet.PSGetTableNameW Method

Pascal

```
function PSGetTableNameW: WideString; virtual;
```

C++

```
virtual __fastcall BSTR PSGetTableNameW();
```

Description

This is PSGetTableNameW, a member of class TDataSet.

39.1.17.4.154 TDataSet.PSGetUpdateException Method

Generates an EUpdateError object based on another exception object.

Pascal

```
function PSGetUpdateException(E: Exception; Prev: EUpdateError): EUpdateError; virtual;
```

C++

```
virtual __fastcall EUpdateError PSGetUpdateException(Exception E, EUpdateError Prev);
```

Description

The provider component calls PSGetUpdateException when the exception E occurs while applying updates. This method allows the provider to create a single EUpdateError object that encapsulates all the exceptions that occur in an update operation.

PSGetUpdateError determines the error message, context string, and error code from the exception specified by E and creates a new EUpdateError exception that includes that information as well as the information about a previous error as specified by Prev. The context string and error code for E may be specific to the Exception class associated with the dataset. By default, TDataSet generates an EUpdateError with an error code of 1 and an empty context string.

See Also

Create (see page 6733)

39.1.17.4.155 TDataSet.PSInTransaction Method

Indicates whether actions against the dataset occur in the context of a transaction.

Pascal

```
function PSInTransaction: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean PSInTransaction();
```

Description

The provider component applies updates within a transaction, if possible. To do so, it first checks whether a transaction is already underway, and if not, generates one using PSStartTransaction. To determine whether a transaction is already underway, the provider calls PSInTransaction.

PSInTransaction returns true if there is a transaction underway, and false otherwise.

See Also

PSStartTransaction (see page 7586)

PSEndTransaction (see page 7578)

39.1.17.4.156 TDataSet.PSIsSQLBased Method

Indicates whether the dataset includes native SQL support.

Pascal

```
function PSIsSQLBased: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean PSIsSQLBased();
```

Description

The provider calls PSIsSQLBased to determine whether the dataset supports some form of native SQL. This information is used when generating SQL commands that are executed by PSExecuteStatement.

Even if the dataset is not SQL-based, it may still allow the provider to execute SQL statements for applying updates. For example, BDE-enabled local databases such as Paradox and DBASE provide SQL support via the BDE's local SQL engine, but are not SQL-based. However, table and field names are not specified in the same way for the local SQL engine as they are for native SQL. The provider uses PSIsSQLBased to determine how these names should appear in generated SQL.

Note: The PSIsSQLSupported method indicates whether the dataset can execute SQL commands at all, regardless of whether the dataset is SQL-based.

See Also

PSIsSQLSupported (see page 7584)

PSExecuteStatement (see page 7579)

39.1.17.4.157 TDataSet.PSIsSQLSupported Method

Indicates whether the dataset can execute SQL statements.

Pascal

```
function PSIsSQLSupported: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean PSIsSQLSupported();
```

Description

PSIsSQLSupported indicates whether the provider can call PSExecuteStatement to execute an SQL statement. When PSIsSQLSupported returns false, calling PSExecuteStatement generates an exception. The provider can still work with a dataset that returns false from PSIsSQLSupported, but it must set the ResolveToDataSet property to true, or provide data on a read-only basis.

See Also

PSIsSQLBased (see page 7584)

PSExecuteStatement (see page 7579)

ResolveToDataSet (see page 18531)

39.1.17.4.158 TDataSet.PSReset Method

Resets the dataset so that it is positioned on the first record.

Pascal

```
procedure PSReset; virtual;
```

C++

```
virtual __fastcall PSReset();
```

Description

The provider calls PSReset to reposition the dataset on the first record before fetching records.

See Also

GetRecords (see page 18468)

AS_GetRecords (see page 16455)

39.1.17.4.159 PSSetCommandText Method

This is the overview for the PSSetCommandText method overload.

Overload List

	Name	Description
	TDataSet.PSSetCommandText (string) (see page 6989)	Assigns a specified SQL command to be executed by PSExecute.
	TDataSet.PSSetCommandText (WideString) (see page 6988)	This is PSSetCommandText, a member of class TDataSet.

39.1.17.4.159.1 TDataSet.PSSetCommandText Method (WideString)

Pascal

```
procedure PSSetCommandText(const CommandText: WideString); virtual; overload;
```

C++

```
virtual __fastcall PSSetCommandText(const BSTR CommandText);
```

Description

This is PSSetCommandText, a member of class TDataSet.

39.1.17.4.159.2 TDataSet.PSSetCommandText Method (string)

Assigns a specified SQL command to be executed by PSExecute.

Pascal

```
procedure PSSetCommandText(const CommandText: string); virtual; overload; deprecated;
```

C++

```
virtual __fastcall PSSetCommandText(const AnsiString CommandText);
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

The value of the CommandText parameter replaces the dataset's SQL statement (if it has one), or replaces the name of an underlying database table or stored procedure.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

39.1.17.4.160 TDataSet.PSSetParams Method

Assigns the specified parameter values to the dataset.

Pascal

```
procedure PSSetParams(AParams: TParams); virtual;
```

C++

```
virtual __fastcall PSSetParams(TParams AParams);
```

Description

PSSetParams assigns the parameter values specified by AParams to the parameters of the dataset.

See Also

Params (see page 8489)

PSGetParams (see page 7582)

39.1.17.4.161 TDataSet.PSStartTransaction Method

Starts a transaction that encapsulates all activity until a call to PSEndTransaction.

Pascal

```
procedure PSStartTransaction; virtual;
```

C++

```
virtual __fastcall PSStartTransaction();
```

Description

The provider component applies updates within a transaction, if possible. To do so, it calls PSStartTransaction to start a transaction before applying updates, and calls PSEndTransaction when all updates are applied or too many errors are encountered.

Note: TDataSetProvider only starts a transaction (Using PSStartTransaction) if the PSInTransaction method indicates that there is no transaction already established. That is, it does not assume nested transaction support is available.

See Also

PSEndTransaction (see page 7578)

PSInTransaction (see page 7583)

applyupdates (see page 18459)

39.1.17.4.162 TDataSet.PSUpdateRecord Method

Applies a single update to the underlying database table or tables.

Pascal

```
function PSUpdateRecord(UpdateKind: TUpdateKind; Delta: TDataSet): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean PSUpdateRecord(TUpdateKind UpdateKind, TDataSet Delta);
```

Description

Before the provider generates SQL statements to apply an update, it calls PSUpdateRecord to let the dataset apply the update in some other fashion. For example, BDE-enabled datasets generate an OnUpdateRecord to allow the application to apply the update and, if the OnUpdateRecord event handler does not apply the update, uses any associated TUpdateSQL object to apply the update.

UpdateKind indicates whether the update is an insertion (ukInsert), deletion (ukDelete), or modification (ukModify).

Delta is the delta datapacket. The current record represents the record to be updated.

PSUpdateRecord returns true if it updates the record, false otherwise. If PSUpdateRecord returns false, the provider generates dynamic SQL to perform the update and calls PSExecuteStatement.

Note: PSUpdateRecord is only called if the provider's ResolveToDataSet property is false.

See Also

OnUpdateRecord (see page 9957)

TUpdateSQL (see page 10229)

applyupdates (see page 18459)

ResolveToDataSet (see page 18531)

PSExecuteStatement (see page 7579)

39.1.17.4.163 TDataSet.Refresh Method

Re-fetches data from the database to update a dataset's view of data.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh to ensure that an application has the latest data from a database. For example, when an application turns off filtering for a dataset, it should immediately call Refresh to display all records in the dataset, not just those that used to meet the filter condition.

Note: The Refresh method does not work for all TDataSet (see page 6894) descendants. In particular, TQuery components do not support the Refresh method if the query is not "live". To refresh a static TQuery, close and reopen the dataset.

TDataSet (see page 6894) generates a BeforeRefresh (see page 7052) event before refreshing the records and an AfterRefresh (see page 7047) event afterwards.

Note: Most datasets try to maintain the current record position when you call refresh. However, this is not always possible. For example, the current record may have been deleted from the server by another user. Unidirectional datasets have no mechanism for locating the current record after a refresh, and always move back to the first record.

Warning: Unidirectional datasets refresh the data by closing and reopening the cursor. This can have unintended side effects if, for example, you have code in the BeforeClose (see page 7048), AfterClose (see page 7040), BeforeOpen (see page 7051), or AfterOpen (see page 7045) event handlers.

Delphi Examples:

```
{
The following example sets a range for a dataset. The form
requires two edit boxes, a data source, a client dataset or
table, a db grid, and a button.  Make sure that the Table1
info in FormActivate agrees with the table used.  Also, the
caption for the button must be initialized to '&ApplyRange'.
}

procedure TForm1.FormActivate(Sender: TObject);
begin
  Table1.DatabaseName := 'DBDEMOS';
  Table1.TableName := 'CustInfo';
  Table1.Active := True;
  Table1.IndexName := '';
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  if Button2.Caption = '&Apply Range' then
    begin
      Table1.SetRange([Edit1.Text],[Edit2.Text]);
      Button2.Caption := '&Drop Range';
    end
  else
    begin
      Table1.CancelRange;
```

```
      Table1.Refresh;
      Button2.Caption := '&Apply Range';
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
    else
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
      Table1.Active:= True;
      for i := 1 to 20 do
        Table1.AppendRecord([i*111, i*222]);
    end;
```

```
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.Active:= False;
  end;
```

See Also

CheckBrowseMode (see page 6918)

Resync (see page 6994)

UpdateCursorPos (see page 7001)

BeforeRefresh (see page 7052)

AfterRefresh (see page 7047)

Refreshing Data Display

Refreshing Records

39.1.17.4.164 TDataSet.RefreshInternalCalcFields Method

Pascal

```
procedure RefreshInternalCalcFields(Buffer: PChar); virtual;
```

C++

```
virtual __fastcall RefreshInternalCalcFields(const char * Buffer);
```

Description

This is RefreshInternalCalcFields, a member of class TDataSet.

39.1.17.4.165 TDataSet.ResetAggField Method

Pascal

```
procedure ResetAggField(Field: TField); virtual;
```

C++

```
virtual __fastcall ResetAggField(TField Field);
```

Description

This is ResetAggField, a member of class TDataSet.

39.1.17.4.166 TDataSet.RestoreState Method

Pascal

```
procedure RestoreState(const Value: TDataSetState);
```

C++

```
__fastcall RestoreState(const TDataSetState Value);
```

Description

This is RestoreState, a member of class TDataSet.

39.1.17.4.167 TDataSet.Resync Method

Re-fetches the active record and the records that precede and follow it.

Pascal

```
procedure Resync(Mode: TResyncMode); virtual;
```

C++

```
virtual __fastcall Resync(TResyncMode Mode);
```

Description

Resync is used internally by other dataset methods to resynchronize the dataset with underlying physical data when making calls that may change the internal cursor position. Applications should use the Refresh (see page 6991) method instead of calling Resync.

Mode indicates optional processing that Resync should handle. If Mode contains rmExact, Resync raises an exception if Resync is called when the cursor is not positioned on a valid record. If Mode contains rmCenter, Resync positions the active record in the center of all buffered records.

Regardless of Mode, Resync also activates the buffers for the active record, retrieves prior and subsequent records for display purposes, and triggers a dataset change event.

See Also

Refresh (see page 6991)

39.1.17.4.168 TDataSet.SetActive Method

Pascal

```
procedure SetActive(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetActive(Boolean Value);
```

Description

This is SetActive, a member of class TDataSet.

39.1.17.4.169 TDataSet.SetBlockReadSize Method

Pascal

```
procedure SetBlockReadSize(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetBlockReadSize(int Value);
```

Description

This is SetBlockReadSize, a member of class TDataSet.

39.1.17.4.170 TDataSet.SetBookmarkData Method

Pascal

```
procedure SetBookmarkData(Buffer: PChar; Data: Pointer); virtual;
```

C++

```
virtual __fastcall SetBookmarkData(const char * Buffer, void * Data);
```

Description

This is SetBookmarkData, a member of class TDataSet.

39.1.17.4.171 TDataSet.SetBookmarkFlag Method

Pascal

```
procedure SetBookmarkFlag(Buffer: PChar; Value: TBookmarkFlag); virtual;
```

C++

```
virtual __fastcall SetBookmarkFlag(const char * Buffer, TBookmarkFlag Value);
```

Description

This is SetBookmarkFlag, a member of class TDataSet.

39.1.17.4.172 TDataSet.SetBookmarkStr Method

Pascal

```
procedure SetBookmarkStr(const Value: TBookmarkStr); virtual;
```

C++

```
virtual __fastcall SetBookmarkStr(const TBookmarkStr Value);
```

Description

This is SetBookmarkStr, a member of class TDataSet.

39.1.17.4.173 TDataSet.SetBufListSize Method

Pascal

```
procedure SetBufListSize(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetBufListSize(int Value);
```

Description

This is SetBufListSize, a member of class TDataSet.

39.1.17.4.174 TDataSet.SetChildOrder Method

Pascal

```
procedure SetChildOrder(Component: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Component, int Order);
```

Description

This is SetChildOrder, a member of class TDataSet.

39.1.17.4.175 TDataSet.SetCurrentRecord Method

Pascal

```
procedure SetCurrentRecord(Index: Integer); virtual;
```

C++

```
virtual __fastcall SetCurrentRecord(int Index);
```

Description

This is SetCurrentRecord, a member of class TDataSet.

39.1.17.4.176 TDataSet.SetDataSetField Method

Pascal

```
procedure SetDataSetField(const Value: TDataSetField); virtual;
```

C++

```
virtual __fastcall SetDataSetField(const TDataSetField Value);
```

Description

This is SetDataSetField, a member of class TDataSet.

39.1.17.4.177 TDataSet.SetDefaultFields Method

Pascal

```
procedure SetDefaultFields(const Value: Boolean);
```

C++

```
__fastcall SetDefaultFields(const Boolean Value);
```

Description

This is SetDefaultFields, a member of class TDataSet.

39.1.17.4.178 SetFieldData Method

This is the overview for the SetFieldData method overload.

Overload List

	Name	Description
	TDataSet.SetFieldData (TField, Pointer) (see page 6996)	This is SetFieldData, a member of class TDataSet.
	TDataSet.SetFieldData (TField, Pointer, Boolean) (see page 6997)	This is SetFieldData, a member of class TDataSet.

39.1.17.4.178.1 TDataSet.SetFieldData Method (TField, Pointer)

Pascal

```
procedure SetFieldData(Field: TField; Buffer: Pointer); virtual; overload;
```

C++

```
virtual __fastcall SetFieldData(TField Field, void * Buffer);
```

Description

This is SetFieldData, a member of class TDataSet.

39.1.17.4.178.2 TDataSet.SetFieldData Method (TField, Pointer, Boolean)

Pascal

```
procedure SetFieldData(Field: TField; Buffer: Pointer; NativeFormat: Boolean); virtual;
overload;
```

C++

```
virtual __fastcall SetFieldData(TField Field, void * Buffer, Boolean NativeFormat);
```

Description

This is SetFieldData, a member of class TDataSet.

39.1.17.4.179 TDataSet.SetFields Method

Sets the values for all fields in a record

Pascal

```
procedure SetFields(const Values: array of const);
```

C++

```
__fastcall SetFields(const array of const Values);
```

Description

Call SetFields to set values for some or all fields in the active record at the same time.

Values contains the values to insert into each field. Values are assigned to the record based on the order of columns in the table or tables underlying the dataset. These values can be literals, variables, NULL, or nil (Delphi). If Values contains fewer values than there are fields in the record, all records for which values are not provided are assigned a NULL value. A NULL value overwrites any existing value in such fields.

In C++, Values_Size is the index of the last value in the Values array (one less than the total number of values).

Before calling SetFields, call Edit (see page 6932) to put the dataset into dsEdit state. After calling SetFields, call Post (see page 6977) to write the changes to the database or change log.

Note: To set values for some fields while retaining existing values for others, pass nil (Delphi) or NULL (C++) (explicitly casting it to void * in C++) for each field that should not change.

See Also

AppendRecord (see page 6911)

InsertRecord (see page 6961)

Modifying Entire Records

39.1.17.4.180 TDataSet.SetFiltered Method

Pascal

```
procedure SetFiltered(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetFiltered(Boolean Value);
```

Description

This is SetFiltered, a member of class TDataSet.

39.1.17.4.181 TDataSet.SetFilterOptions Method

Pascal

```
procedure SetFilterOptions(Value: TFilterOptions); virtual;
```

C++

```
virtual __fastcall SetFilterOptions(TFilterOptions Value);
```

Description

This is SetFilterOptions, a member of class TDataSet.

39.1.17.4.182 TDataSet.SetFilterText Method

Pascal

```
procedure SetFilterText(const Value: string); virtual;
```

C++

```
virtual __fastcall SetFilterText(const AnsiString Value);
```

Description

This is SetFilterText, a member of class TDataSet.

39.1.17.4.183 TDataSet.SetFound Method

Pascal

```
procedure SetFound(const Value: Boolean);
```

C++

```
__fastcall SetFound(const Boolean Value);
```

Description

This is SetFound, a member of class TDataSet.

39.1.17.4.184 TDataSet.SetModified Method

Pascal

```
procedure SetModified(Value: Boolean);
```

C++

```
__fastcall SetModified(Boolean Value);
```

Description

This is SetModified, a member of class TDataSet.

39.1.17.4.185 TDataSet.SetName Method

Sets the value of the Name property.

Pascal

```
procedure SetName(const Value: TComponentName); override;
```

C++

```
virtual __fastcall SetName(const TComponentName Value);
```

Description

SetName is the virtual property setter for the Name property. It calls the ChangeName method, which performs the actual name change.

ChangeName is not virtual; override SetName instead when modifying the behavior of the Name property.

Warning: Using SetName to change the name of a component at runtime causes any references to the old name to become undefined. Any subsequent code that uses the old name will cause an exception.

See Also

ChangeName (see page 2770)

Name (see page 2805)

ValidateRename (see page 2792)

39.1.17.4.186 TDataSet.SetObjectView Method

Pascal

```
procedure SetObjectView(const Value: Boolean);
```

C++

```
__fastcall SetObjectView(const Boolean Value);
```

Description

This is SetObjectView, a member of class TDataSet.

39.1.17.4.187 TDataSet.SetOnFilterRecord Method

Pascal

```
procedure SetOnFilterRecord(const Value: TFilterRecordEvent); virtual;
```

C++

```
virtual __fastcall SetOnFilterRecord(const TFilterRecordEvent Value);
```

Description

This is SetOnFilterRecord, a member of class TDataSet.

39.1.17.4.188 TDataSet.SetRecNo Method

Pascal

```
procedure SetRecNo(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetRecNo(int Value);
```

Description

This is SetRecNo, a member of class TDataSet.

39.1.17.4.189 TDataSet.SetState Method

Pascal

```
procedure SetState(Value: TDataSetState);
```

C++

```
__fastcall SetState(TDataSetState Value);
```

Description

This is SetState, a member of class TDataSet.

39.1.17.4.190 TDataSet.SetStateFieldValue Method

Pascal

```
procedure SetStateFieldValue(State: TDataSetState; Field: TField; const Value: Variant);
virtual;
```

C++

```
virtual __fastcall SetStateFieldValue(TDataSetState State, TField Field, const Variant Value);
```

Description

This is SetStateFieldValue, a member of class TDataSet.

39.1.17.4.191 TDataSet.SetTempState Method

Pascal

```
function SetTempState(const Value: TDataSetState): TDataSetState;
```

C++

```
__fastcall TDataSetState SetTempState(const TDataSetState Value);
```

Description

This is SetTempState, a member of class TDataSet.

39.1.17.4.192 TDataSet.SetUniDirectional Method

Pascal

```
procedure SetUniDirectional(const Value: Boolean);
```

C++

```
__fastcall SetUniDirectional(const Boolean Value);
```

Description

This is SetUniDirectional, a member of class TDataSet.

39.1.17.4.193 TDataSet.TempBuffer Method

Pascal

```
function TempBuffer: PChar;
```

C++

```
__fastcall const char * TempBuffer();
```

Description

This is TempBuffer, a member of class TDataSet.

39.1.17.4.194 TDataSet.Translate Method

Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.

Pascal

```
function Translate(Src: PChar; Dest: PChar; ToOem: Boolean): Integer; virtual;
```

C++

```
virtual __fastcall int Translate(const char * Src, const char * Dest, Boolean ToOem);
```

Description

As implemented by TDataSet (see page 6894), Translate copies a source string to a destination string, and ignores the value passed in ToOem. Translate provides a fallback method for derived dataset objects that do not reimplement Translate.

In BDE-enabled datasets, Translate converts the source string from ANSI (the native character mapping for the VCL) to OEM (the native character mapping for the BDE) when ToOem is true, and reverses that translation when ToOem is false.

Custom datasets can use Translate to convert strings between ANSI UTF-8 (the native character mapping for dataset components) and any character mapping used by the underlying data access mechanism. When ToOem is true, Translate converts from ANSI to the underlying data access mapping. When ToOem is false, Translate reverses that conversion.

39.1.17.4.195 TDataSet.UpdateBufferCount Method

Pascal

```
procedure UpdateBufferCount;
```

C++

```
__fastcall UpdateBufferCount();
```

Description

This is UpdateBufferCount, a member of class TDataSet.

39.1.17.4.196 TDataSet.UpdateCursorPos Method

Positions the cursor on the active record.

Pascal

```
procedure UpdateCursorPos;
```

C++

```
__fastcall UpdateCursorPos();
```

Description

UpdateCursorPos is an internal routine used by many dataset methods to ensure that the physical cursor is positioned on the active record. Normally an application should not need to call UpdateCursorPos. Typically UpdateCursorPos is called to ensure that the physical cursor position matches the logical cursor position.

See Also

RecNo (see page 7031)

CursorPosChanged (see page 6923)

Refresh (see page 6991)

Resync (see page 6994)

CurrentRecord (see page 7013)

39.1.17.4.197 TDataSet.UpdateIndexDefs Method

Pascal

```
procedure UpdateIndexDefs; virtual;
```

C++

```
virtual __fastcall UpdateIndexDefs();
```

Description

This is UpdateIndexDefs, a member of class TDataSet.

39.1.17.4.198 TDataSet.UpdateRecord Method

Ensures that data-aware controls and detail datasets reflect record updates.

Pascal

```
procedure UpdateRecord;
```

C++

```
__fastcall UpdateRecord();
```

Description

UpdateRecord is used internally by some dataset methods to inform data-aware controls of updates and trigger an OnUpdateRecord event if cached updates are enabled. Applications should not need to call UpdateRecord directly unless they provide custom dataset methods that bypass TDataSet (see page 6894) methods.

See Also

OnUpdateRecord (see page 9957)

39.1.17.4.199 TDataSet.UpdateStatus Method

Reports the update status for the active record.

Pascal

```
function UpdateStatus: TUpdateStatus; virtual;
```

C++

```
virtual __fastcall TUpdateStatus UpdateStatus();
```

Description

As implemented in TDataSet (see page 6894), UpdateStatus always returns usUnmodified. Descendant classes override this method to reflect whether the active record has been modified, and if so, how.

39.1.17.5 TDataSet Properties

The properties of the TDataSet class are listed here.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
	Reserved (see page 7035)	For internal use only.

Public Properties

	Name	Description
	Active (see page 7004)	Specifies whether or not a dataset is open.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
R	DataSource (see page 7014)	Represents the data source of another dataset that supplies values to the dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
	FieldDefs (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.

	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	Modified (see page 7029)	Indicates whether the active record is modified.
	ObjectView (see page 7031)	Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.

39.1.17.5.1 TDataSet.Active Property

Specifies whether or not a dataset is open.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Use Active to determine or set whether a dataset is populated with data. When Active is false, the dataset is closed; the dataset cannot read or write data and data-aware controls can not use it to fetch data or post edits. When Active is true, the dataset can be populated with data. It can read data from a database or other source (such as a provider). Depending on the CanModify (see page 7012) property, active datasets can post changes.

Setting Active to true:

1. Generates a BeforeOpen (see page 7051) event.
2. Sets the dataset state to dsBrowse.
3. Establishes a way to fetch data (typically by opening a cursor).
4. Generates an AfterOpen (see page 7045) event.

If an error occurs while opening the dataset, dataset state is set to dsInactive, and any cursor is closed.

Setting Active to false:

1. Triggers a BeforeClose (see page 7048) event.
2. Sets the State (see page 7036) property to dsInactive.
3. Closes the cursor.
4. Triggers an AfterClose (see page 7040) event.

An application must set Active to false before changing other properties that affect the status of a database or the controls that display data in an application.

Note: Calling the Open (see page 6976) method sets Active to true; calling the Close (see page 6920) method sets Active to false.

Delphi Examples:

```
{
```

```
  This example requires a database with the following name or
  in the form of a *.DB file in the following path.  The table
  should be closed when changing the database.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Customers.Active := False;
    try
    { First try to use an alias }
      Customers.DatabaseName := 'DBDEMOS';
      Customers.Active := True;
    except
      on EDatabaseError do
      begin
        { If that fails, try to use the drive and directory }
        Customers.TableName := 'Flights';
        Customers.DatabaseName :=
          'c:\Documents and Settings\All Users\Documents\RAD
  Studio\5.0\Demos\IntraWeb\Win32\FlightInformation\Data';
        Customers.Active := True;
      end;
    end;
  end;
```

See Also

DataSource (see page 7014)

State (see page 7036)

AfterOpen (see page 7045)

BeforeOpen (see page 7051)

Close (see page 6920)

Open (see page 6976)

CanModify (see page 7012)

BeforeClose (see page 7048)

AfterClose (see page 7040)

Using Datasets with Decision Support Components

Making the Table a Detail of Another Dataset

Opening and Closing Datasets

Fetching the Data

39.1.17.5.2 TDataSet.ActiveRecord Property

Indicates the index of the active record in the internal cache of records.

Pascal

```
property ActiveRecord: Integer;
```

C++

```
__property int ActiveRecord;
```

Description

Most TDataSet (see page 6894) descendants maintain an internal cache of records from the underlying database table. This cache is used, for example, when the dataset is used to populate a data-aware grid, in which case it includes a record for each

row in the grid. ActiveRecord indicates the offset in this cache of the record buffer for the active record. This active record is the value returned by the public ActiveBuffer (see page 6909) method.

Note: Unidirectional datasets never cache records. For unidirectional datasets, the ActiveRecord property is meaningless (and always 0).

See Also

CurrentRecord (see page 7013)

RecNo (see page 7031)

Bookmark (see page 7009)

ActiveBuffer (see page 6909)

BufferCount (see page 6890)

Buffers (see page 7011)

IsUniDirectional (see page 7029)

39.1.17.5.3 TDataSet.AggFields Property

Contains any aggregate fields.

Pascal

```
property AggFields: TFields;
```

C++

```
__property TFields AggFields;
```

Description

Some TDataSet (see page 6894) descendants, such as TClientDataSet, support aggregate fields. Aggregate fields summarize data over groups of records from the dataset (as opposed to calculated fields, which can only summarize data over the fields of a single record). AggFields is a collection of all aggregate fields that are supported by the dataset.

Note: The Fields (see page 7023) property is a collection of all the dataset's fields that are not aggregate fields. Thus, AggFields and Fields (see page 7023) are mutually exclusive collections of the dataset's fields. These two properties contain all of the dataset's fields between them.

See Also

TAggregateField (see page 6743)

Aggregates (see page 8461)

Fields (see page 7023)

FieldValues (see page 7025)

Obtaining Aggregate Values

39.1.17.5.4 TDataSet.AutoCalcFields Property

Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.

Pascal

```
property AutoCalcFields: Boolean;
```

C++

```
__property Boolean AutoCalcFields;
```

Description

Set AutoCalcFields to control when the OnCalcFields (see page 7053) event is triggered to update calculated fields and when lookup fields are calculated.

A calculated field is one that derives its value from the values of one or more fields in the active record, sometimes with additional processing. Lookup (see page 6970) fields are fields whose values come from a secondary dataset or lookup cache.

Note: Unidirectional datasets support calculated fields but not unidirectional fields. For unidirectional datasets, OnCalcFields (see page 7053) only controls when calculated fields are updated.

When AutoCalcFields is true (the default), Lookup (see page 6970) fields are recalculated and OnCalcFields (see page 7053) is triggered when:

The dataset is opened.

The dataset is put into dsEdit state.

Focus moves from one visual control to another, or from one column to another in a data-aware grid and modifications have been made to the record.

When AutoCalcFields is false, Lookup (see page 6970) fields are recalculated and the OnCalcFields (see page 7053) event occurs only when

The dataset is opened.

The dataset is put into dsEdit state.

A record is retrieved from a database.

If an application permits users to change data, OnCalcFields (see page 7053) is frequently triggered. In these cases an application may set AutoCalcFields to false to reduce the frequency with which the OnCalcFields (see page 7053) event occurs and with which lookup values are fetched.

See Also

OnCalcFields (see page 7053)

Calculating Fields

39.1.17.5.5 TDataSet.BlobFieldCount Property

Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.

Pascal

```
property BlobFieldCount: Integer;
```

C++

```
__property int BlobFieldCount;
```

Description

Descendants of TDataSet (see page 6894) use BlobFieldCount when determining how to handle BLOB information.

39.1.17.5.6 TDataSet.BlockReadSize Property

Determines how many record buffers are read in each block.

Pascal

```
property BlockReadSize: Integer;
```

C++

```
__property int BlockReadSize;
```

Description

Setting BlockReadSize to a value greater than 0 puts the dataset in the dsBlockRead state. It remains in that state until you set BlockReadSize to 0 or change the dataset's state in some other manner (such as putting it in edit mode).

For all TDataSet (see page 6894) descendants, data-aware controls are not updated and data events are not triggered when the dataset is in the dsBlockRead state. Thus, you can set BlockReadSize to a positive value to scan through the entire dataset quickly without updating data-aware controls.

For BDE-enabled datasets, setting BlockReadSize also causes the dataset to fetch database information in blocks of BlockReadSize records, which it then buffers. This minimizes traffic between your application and the database server.

See Also

State (see page 7036)

Next (see page 6973)

39.1.17.5.7 TDataSet.Bof Property

Indicates whether the first record in the dataset is active.

Pascal

```
property Bof: Boolean;
```

C++

```
__property Boolean Bof;
```

Description

Test Bof (beginning of file) to determine if the dataset is positioned at the first record. If Bof is true, the active record is unequivocally the first row in the dataset. Bof is true when an application.

Opens a dataset.

Calls a dataset's First (see page 6942) method.

Call a dataset's Prior (see page 6979) method, and the method fails because the first row is already active.

Calls SetRange on an empty range or dataset.

Bof is false in all other cases.

Delphi Examples:

```
{
This example uses a button to copy the value of a field in
the previous record into the corresponding field in the
current record.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   SavePlace: TBookmark;
   PrevValue: Variant;
begin
   with Customers do
   begin
```

```
      { get a bookmark so that we can return to the same record }
      SavePlace := GetBookmark;
      try
        { move to prior record}
        FindPrior;
        { get the value }
        PrevValue := Fields[1].Value;
        {Move back to the bookmark
        this may not be the next record anymore
        if something else is changing the dataset asynchronously }
        GotoBookmark(SavePlace);
        { Set the value }
        Edit;
        Fields[1].Value := PrevValue;
        { Free the bookmark }
      finally
        FreeBookmark(SavePlace);
      end;
    end;
  end;

  {
  To ensure that the button is disabled when there is no
  previous record, the OnDataChange event of the DataSource
  detects when the user moves to the beginning of file (BOF
  property becomes true), and disables the button.  Detection
  occurs on scrolling and editing, not selection with the mouse.
  }
  procedure TForm1.DS2DataChange(Sender: TObject; Field: TField);
  begin
    if Customers.Bof then
      Button1.Enabled := False
    else
      Button1.Enabled := True;
  end;
```

See Also

Eof (see page 7015)

First (see page 6942)

Prior (see page 6979)

SetRange (see page 10209)

SetRange (see page 8452)

Using the Eof and Bof Properties

Using the First and Last Methods

Using the MoveBy Method

Using the Next and Prior Methods

39.1.17.5.8 TDataSet.Bookmark Property

Specifies the current bookmark in the dataset.

Pascal

```
property Bookmark: TBookmarkStr;
```

C++

```
__property TBookmarkStr Bookmark;
```

Description

Bookmark gets or sets the current bookmark in a dataset. A bookmark marks a location in a dataset so that an application can easily return to that location quickly.

An application can read Bookmark to retrieve the bookmark associated with the current record, and it can change the current record in the dataset by assigning a saved bookmark value to this property.

Note: Unidirectional datasets do not support bookmarks.

See Also

GetBookmark (see page 6948)

GotoBookmark (see page 6957)

FreeBookmark (see page 6945)

Marking and Returning to Records

39.1.17.5.9 TDataSet.BookmarkSize Property

Indicates the number of bytes used to represent a bookmark.

Pascal

```
property BookmarkSize: Integer;
```

C++

```
__property int BookmarkSize;
```

Description

Bookmarks mark a location in a dataset so that an application can easily return to that location quickly. Descendants of TDataSet (see page 6894) can use different mechanisms for storing enough information to identify records when applications set bookmarks. BookmarkSize indicates the number of bytes required to specify bookmark information for a particular dataset.

Note: Unidirectional datasets do not support bookmarks.

See Also

Bookmark (see page 7009)

GetBookmark (see page 6948)

FreeBookmark (see page 6945)

39.1.17.5.10 TDataSet.BufferCount Property

Indicates the number of records in the internal cache.

Pascal

```
property BufferCount: Integer;
```

C++

```
__property int BufferCount;
```

Description

All TDataSet (see page 6894) descendants except for unidirectional datasets maintain an internal cache of records from the underlying database tables. For example, if the dataset is used to populate a data-aware grid, the cache includes a record for each row in the grid. BufferCount indicates the size of this cache.

BufferCount is read-only. An associated TDataLink (see page 6876) object determines its value.

See Also

CurrentRecord (see page 7013)

ActiveBuffer (see page 6909)

BufferCount (see page 6890)

ActiveRecord (see page 7005)

Buffers (see page 7011)

39.1.17.5.11 TDataSet.Buffers Property

Provides indexed access to the record buffers in the internal cache.

Pascal

```
property Buffers [Index: Integer]: PChar;
```

C++

```
__property const char * Buffers[int Index];
```

Description

All TDataSet (see page 6894) descendants except for unidirectional datasets maintain an internal cache of records from the underlying database table. For example, if the dataset is used to populate a data-aware grid, the cache includes a record for each row in the grid.

For datasets that cache records, each entry in Buffers points to a record buffer in the internal cache. The BufferCount (see page 7010) property specifies the total number of record buffers in the cache.

See Also

BufferCount (see page 7010)

CurrentRecord (see page 7013)

39.1.17.5.12 TDataSet.CalcBuffer Property

Points to the record buffer used during an OnCalcFields (see page 7053) event.

Pascal

```
property CalcBuffer: PChar;
```

C++

```
__property const char * CalcBuffer;
```

Description

Datasets use CalcBuffer to retrieve and store values when in the dsCalcFields state. The values of lookup fields and calculated fields are set in CalcBuffer, based on data field values retrieved from this buffer.

Most applications do not need to explicitly reference CalcBuffer. It is used automatically when the dataset is in the dsCalcFields state.

See Also

OnCalcFields (see page 7053)

State (see page 7036)

39.1.17.5.13 TDataSet.CalcFieldsSize Property

Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.

Pascal

```
property CalcFieldsSize: Integer;
```

C++

```
__property int CalcFieldsSize;
```

Description

TDataSet (see page 6894) uses CalcFieldsSize to determine whether there are any calculated fields or lookup fields to calculate. Descendants of TDataSet (see page 6894) use this property when locating fields in a record buffer.

See Also

CalcBuffer (see page 7011)

39.1.17.5.14 TDataSet.CanModify Property

Indicates whether the dataset permits write access to data.

Pascal

```
property CanModify: Boolean;
```

C++

```
__property Boolean CanModify;
```

Description

As implemented in TDataSet (see page 6894), this property is false if the dataset is unidirectional, and true otherwise. Descendant classes reimplement this property as follows

For unidirectional datasets, CanModify is always false.

For client datasets, CanModify reflects the value of the ReadOnly property and whether a data provider supplied a read-only data packet.

Other datasets reflect whether the dataset has write access to data.

See Also

DataSource (see page 7014)

AutoEdit (see page 7077)

ReadOnly (see page 8493)

Editing Data in a Control

Editing in the Grid

Adding New Records

Controlling Read/Write Access to Tables

Editing Records

Using Unidirectional Datasets

39.1.17.5.15 TDataSet.Constraints Property

Specifies record-level constraints that must be met when editing the data.

Pascal

```
property Constraints: TCheckConstraints;
```

C++

```
__property TCheckConstraints Constraints;
```

Description

Use Constraints to read or add record-level constraints to the dataset. Record-level constraints usually impose relationships between the fields in a single record. Constraints that limit the value of a single field should be added to the individual field components.

See Also

TCheckConstraint (see page 6841)

CustomConstraint (see page 7156)

ImportedConstraint (see page 7167)

Constraining Data Values

39.1.17.5.16 TDataSet.CurrentRecord Property

Indicates the index of the current record in the internal cache of record buffers.

Pascal

```
property CurrentRecord: Integer;
```

C++

```
__property int CurrentRecord;
```

Description

Most TDataSet (see page 6894) descendants maintain an internal cache of records from the underlying database table. For example, if the dataset is used to populate a data-aware grid, the cache includes a record for each row in the grid. CurrentRecord indicates the offset in this cache of the record buffer for the current record.

The current record is the record currently in use by the dataset. This can differ from the active record (which the end user sees as active) when the dataset performs operations involving other records. For example, when painting a data-aware grid, CurrentRecord cycles through all the visible records, while ActiveRecord (see page 7005) remains constant.

Note: For unidirectional datasets, CurrentRecord is meaningless.

See Also

GetCurrentRecord (see page 6950)

ActiveRecord (see page 7005)

39.1.17.5.17 TDataSet.DataSetField Property

Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.

Pascal

```
property DataSetField: TDataSetField;
```

C++

```
__property TDataSetField DataSetField;
```

Description

Use DataSetField when the dataset is the nested detail set of another dataset, such as an Oracle8 nested dataset. Read DataSetField to access the data set field component in the master dataset that includes this dataset as a detail. Set DataSetField to a field component in the master dataset to assign this dataset as the value of that field.

Note: When setting the DataSetField property, the dataset must be of (or a descendant of) the class specified by the master dataset's NestedDataSetClass (see page 7030) property. Otherwise, setting DataSetField raises an exception. Further, setting DataSetField to a TDataSetField (see page 7060) that is contained in the same dataset raises an exception (A dataset can't be nested in itself).

See Also

TNestedTable (see page 10040)

NestedDataSet (see page 7070)

NestedDataSetClass (see page 7030)

Using Nested Detail Tables

Working with DataSet Fields

Working with Reference Fields

39.1.17.5.18 TDataSet.DataSource Property

Represents the data source of another dataset that supplies values to the dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

As implemented in TDataSet (see page 6894), the DataSource property is always nil (Delphi) or NULL (C++).

Descendant dataset classes that support specifying a data source redeclare and implement methods for getting and setting the DataSource property.

See Also

Active (see page 7004)

IsLinkedTo (see page 6967)

Establishing Master/detail Relationships Using Parameters

39.1.17.5.19 TDataSet.DefaultFields Property

Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.

Pascal

```
property DefaultFields: Boolean;
```

C++

```
__property Boolean DefaultFields;
```

Description

Read DefaultFields to determine whether a dataset uses dynamically generated field components or persistent field components. If DefaultFields is true, the dataset uses dynamically allocated field components. If DefaultFields is false, the dataset uses persistent field components.

Unless persistent field components are assigned to a dataset at design time using the Fields (see page 7023) editor, the dataset creates dynamic field components based on the structure of its underlying data when it is opened.

See Also

Active (see page 7004)

Fields (see page 7023)

Open (see page 6976)

39.1.17.5.20 TDataSet.Designer Property

Returns a pointer to the Dataset designer for the dataset.

Pascal

```
property Designer: TDataSetDesigner;
```

C++

```
__property TDataSetDesigner Designer;
```

Description

Use Designer to determine whether the dataset is currently being designed and, if so, to access the dataset designer. When a dataset designer, such as the Fields (see page 7023) editor, is created for a dataset at design time, it sets the Designer property to itself. When the dataset designer is destroyed, the Designer property is set back to nil (Delphi) or NULL (C++). A Dataset designer enables a developer to specify the fields that belong to a dataset at design time.

See Also

DefaultFields (see page 7014)

Fields (see page 7023)

39.1.17.5.21 TDataSet.Eof Property

Indicates whether a dataset is positioned at the last record.

Pascal

```
property Eof: Boolean;
```

C++

```
__property Boolean Eof;
```

Description

Test Eof (end-of-file) to determine if the active record in a dataset is the last record. If Eof is true, the current record is unequivocally the last row in the dataset. Eof is true when an application:

Opens an empty dataset.

Calls a dataset's Last (see page 6968) method. (Unless it is a unidirectional dataset)

Call a dataset's Next (see page 6973) method, and the method fails because the current record is already the last row in the dataset.

Calls SetRange on an empty range or dataset.

Eof is false in all other cases.

Tip: If both Eof and Bof (see page 7008) are true, the dataset or range is empty.

Delphi Examples:

```
{
Usually DisableControls is called within the context of a
try...finally block that reenables the controls even if an
exception occurs. For example:
}
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
begin
  ListBox1.Clear;
  ListBox1.Items[0]:= 'Destination Airports:';
  with Flights do
  begin
    DisableControls;
    try
      First;
      i:= 1;
      ListBox1.Items[0]:= 'Destination Airports:';
      while not Eof do
      begin
        ListBox1.Items[i]:= Fields[2].Value;;
        i:= i + 1;
        Next;
      end;
    finally
      EnableControls;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Flights:= TTable.Create(Form1);
  with Flights do
  begin
    TableName := 'Flights';
    DatabaseName := 'Flights';

  end;
  DS2.DataSet:= Flights;
  DBGrid2.DataSource.DataSet:=
    c:\Documents and Settings\All Users\Documents\RAD
Studio\5.0\Demos\IntraWeb\Win32\FlightInformation\Data;
  Flights.Active:= True;
end;
```

See Also

Bof (see page 7008)

Last (see page 6968)

Next (see page 6973)

SetRange (see page 10209)

SetRange (see page 8452)

Using the Eof and Bof Properties

Using the First and Last Methods

Using the MoveBy Method

Using the Next and Prior Methods

39.1.17.5.22 TDataSet.FieldCount Property

Indicates the number of field components associated with the dataset.

Pascal

```
property FieldCount: Integer;
```

C++

```
__property int FieldCount;
```

Description

Examine FieldCount to determine the number of fields listed by the Fields (see page 7023) property. For datasets with dynamically created fields, FieldCount may differ each time a dataset is opened. For datasets with persistent fields, FieldCount should be unchanged each time a dataset is open.

Note: FieldCount includes only the fields listed by the Fields (see page 7023) property. Any aggregated fields listed by the AggFields (see page 7006) property are not included in the count.

Delphi Examples:

```
{
This example displays a message box with the names of all
fields in a table.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   i: Integer;
   Info: String;
begin
   Info := 'The fields of table ' + Table1.TableName +
           ' are:'#13#10#13#10;
   for i := 0 to Table1.FieldCount - 1 do
      Info := Info + Table1.Fields[i].FieldName + #13#10;
  ShowMessage(Info);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
```

```
      DatabaseName := 'DBDEMOS';
      TableType := ttParadox;
      TableName := 'CustInfo';
      Table1.Active := False;
      { Don't overwrite an existing table }
      if not Table1.Exists then
      begin
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
        { Next, describe the fields in the table }
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.Active:= True;
  end;
```

See Also

DefaultFields (see page 7014)

Designer (see page 7015)

Fields (see page 7023)

39.1.17.5.23 TDataSet.FieldDefList Property

Points to the list of field definitions for the dataset.

Pascal

```
property FieldDefList: TFieldDefList;
```

C++

```
__property TFieldDefList FieldDefList;
```

Description

FieldDefList points to an internal list of field definitions for the dataset. FieldDefList represents a flattened view of the data, meaning that object fields in the dataset may be represented by several simple field definitions that represent the constituents of the object field. To determine the definitions in a hierarchical view, use FieldDefs (see page 7019) instead.

To access fields and field values in a dataset, use the Fields (see page 7023), AggFields (see page 7006), and FieldValues (see page 7025) properties, and the FieldByName (see page 6936) method.

See Also

FieldDefs (see page 7019)

ObjectView (see page 7031)

TObjectField (see page 7372)

Fields (see page 7023)

AggFields (see page 7006)

FieldValues (see page 7025)

FieldByName (see page 6936)

39.1.17.5.24 TDataSet.FieldDefs Property

Points to the list of field definitions for the dataset.

Pascal

```
property FieldDefs: TFieldDefs;
```

C++

```
__property TFieldDefs FieldDefs;
```

Description

FieldDefs lists the field definitions for a dataset. While an application can examine FieldDefs to explore the field definitions for a dataset, it should not change these definitions unless creating a new table with CreateTable or CreateDataSet.

To access fields and field values in a dataset, use the Fields (see page 7023), AggFields (see page 7006), and FieldValues (see page 7025) properties, and the FieldsByName method.

Note: If the dataset includes object field descendants, FieldDefs represents a hierarchical view of the data, meaning that the definitions include object field definitions. To determine the definitions in a flattened view, use FieldDefList (see page 7019) instead.

C++ Examples:

```
/*
The following code creates and activates a client dataset in
the form's OnCreate event handler.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TFieldDefs *pDefs = CDS2->FieldDefs;
  TFieldDef *pDef = pDefs->AddFieldDef();
  pDef->DataType = ftInteger;
  pDef->Name = "Field1";

  pDef = pDefs->AddFieldDef();
  pDef->DataType = ftString;
  pDef->Size = 10;
  pDef->Name = "Field2";

  TIndexDef *pIDef = CDS2->IndexDefs->AddIndexDef();
  pIDef->Fields = "Field1";
  pIDef->Name = "IntIndex";

  CDS2->CreateDataSet();
}
```

Delphi Examples:

```
{
The following code creates and activates a client dataset in
the form's OnCreate event handler.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  with CDS2 do
  begin
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftInteger;
      Name := 'Field1';
    end;
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftString;
      Size := 10;
      Name := 'Field2';
    end;
    with IndexDefs.AddIndexDef do
    begin
      Fields := 'Field1';
      Name := 'IntIndex';
    end;
    CreateDataSet;
  end;
end;
{ The following example shows how to create a table. }
procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Close;
  Table1.DeleteTable;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
```

```
      TableType := ttParadox;
      TableName := 'CustInfo';
      Table1.Active := False;
      { Don't overwrite an existing table }
      if Table1.Exists then
        MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
      else
      begin
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
        { Next, describe the fields in the table }
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.Active:= True;
  end;
```

See Also

CreateTable (see page 10189)

Fields (see page 7023)

FieldValues (see page 7025)

ClearFields (see page 6919)

GetFieldList (see page 6953)

GetFieldNames (see page 6954)

FieldByName (see page 6936)

FindField (see page 6938)

SetFields (see page 6997)

ObjectView (see page 7031)

AggFields (see page 7006)

CreateDataSet (see page 8401)

Creating and Deleting Tables

39.1.17.5.25 TDataSet.FieldList Property

Lists the field components of a dataset.

Pascal

```
property FieldList: TFieldList;
```

C++

```
__property TFieldList FieldList;
```

Description

FieldList contains the names of all field components in the dataset. The fields are stored sequentially, or flattened out, meaning any child fields of an object field are stored as siblings in the TFieldList (see page 7207) . ::Fields (see page 7023) array.

See Also

Fields (see page 7023)

39.1.17.5.26 TDataSet.FieldNoOfs Property

Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.

Pascal

```
property FieldNoOfs: Integer;
```

C++

```
__property int FieldNoOfs;
```

Description

Add FieldNoOfs to a field index to obtain its FieldNo property.

See Also

Fields (see page 7023)

FieldList (see page 7022)

FieldNo (see page 7166)

39.1.17.5.27 TDataSet.Fields Property

Lists all non-aggregate field components of the dataset.

Pascal

```
property Fields: TFields;
```

C++

```
__property TFields Fields;
```

Description

Use Fields to access field components. If fields are generated dynamically at runtime, the order of field components in Fields corresponds directly to the order of columns in the table or tables underlying a dataset. If a dataset uses persistent fields, then the order of field components corresponds to the ordering of fields specified in the Fields editor at design time.

When ObjectView (see page 7031) is true, the fields are stored hierarchically, meaning any child fields of an object field are referenced by the object field and don't appear sequentially after the object field in the TFields (see page 7211) . :: Fields array. When ObjectView (see page 7031) is false, the fields are stored sequentially, or flattened out, meaning any child fields of an object field are stored sequentially in the TFields (see page 7211) . :: Fields array.

Accessing fields with the Fields property is useful for applications that:

Iterate over some or all fields in a dataset.

Work with underlying tables whose internal data structure is unknown at runtime.

If an application knows the data types of individual fields, then it can read or write individual field values through the Fields property. For example, the following statement assigns a field value to the Text property of an edit box:

```
Edit1.Text := CustTable.Fields.Fields[6].AsString;
Edit1->Text = CustTable->Fields->Fields[0]->AsString;
```

Note: The preferred method for retrieving and assigning field values is to use persistent fields or the FieldByName (see page 6936) method.

Note: The AggFields (see page 7006) property is a collection of all the dataset's aggregated fields. AggFields (see page 7006) and Fields are mutually exclusive collections of the dataset's fields. These two properties contain all of the dataset's fields between them.

Delphi Examples:

```
{
This example displays a message box with the names of all
fields in a table.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   i: Integer;
   Info: String;
begin
   Info := 'The fields of table ' + Table1.TableName +
           ' are:'#13#10#13#10;
   for i := 0 to Table1.FieldCount - 1 do
      Info := Info + Table1.Fields[i].FieldName + #13#10;
  ShowMessage(Info);
end;

procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if not Table1.Exists then
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
    end;
  end;
  DS2.DataSet:= Table1;
  DBGrid2.DataSource.DataSet:= Table1;
  Table1.Active:= True;
end;
```

See Also

FieldByName (see page 6936)

Edit (see page 6932)

Post (see page 6977)

TEdit (see page 21402)

FieldList (see page 7022)

AggFields (see page 7006)

FieldDefs (see page 7019)

FieldCount (see page 7017)

Executing a Search with Goto Methods

Modifying a Range

Specifying Ranges

Accessing Field Values with a Dataset's Fields Property

Working with Reference Fields

39.1.17.5.28 TDataSet.FieldValues Property

Provides access to the values for all fields in the active record for the dataset

Pascal

```
property FieldValues [const FieldName: string]: Variant;
```

C++

```
__property Variant FieldValues[AnsiString const FieldName];
```

Description

Use FieldValues to read and write values for fields in a dataset. FieldName is the name of a field to read from or write to.

FieldName can represent simple field names, qualified field names for subfields of an object field, or aggregated fields such as can be found in the AggFields (see page 7006) property. Because of this flexibility, it is often preferable to use the FieldValues property (or the FieldByName (see page 6936) method) rather than the Fields (see page 7023), FieldList (see page 7022), or AggFields (see page 7006) properties, all of which present a more limited selection of the dataset's fields.

FieldValues accepts and returns a Variant, so it can handle and convert fields of any type. Because FieldValues is the default property for TDataSet (see page 6894), you can omit the property name when referencing this property. For example, the following statements are semantically identical and write the value from an edit box into an integer field:

```
Customers.FieldValues['CustNo'] := Edit1.Text;
Customers['CustNo'] := Edit1.Text;
Customers->FieldValues["CustNo"] = Edit1->Text;
```

Note: Because FieldValues always uses Variants (see page 23567), it may be a somewhat slower method of accessing data, than using a field's native format (for example, using a field's AsXXX property), especially in applications that process large amounts of data.

Delphi Examples:

```
{
This example appends a new record to a table or client
dataset when the user clicks a button. The two fields
Field1 and Field2 are filled from the contents of two edit
controls.  This example requires a populated TTable or
TDataSet with the appropriate field values.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
  Customers.Append;
  Customers.FieldValues['Field1'] := StrToInt(Edit1.text);
  Customers.FieldValues['Field2'] := Edit2.text;
  Customers.Post;
end;
```

See Also

Fields (see page 7023)

FieldByName (see page 6936)

Edit (see page 6932)

Post (see page 6977)

AggFields (see page 7006)

operator_sb

39.1.17.5.29 TDataSet.Filter Property

Specifies the text of the current filter for a dataset.

Pascal

```
property Filter: string;
```

C++

```
__property AnsiString Filter;
```

Description

Use Filter to specify a dataset filter. When filtering is applied to a dataset, only those records that meet a filter's conditions are available. Filter describes the filter condition. For example, the following filter condition displays only those records where the State (see page 7036) field is 'CA' or 'MA':

```
State = 'CA' or State = 'MA'
```

When a filter is set, Blank records do not appear unless explicitly included in the filter. For example:

```
State <> 'CA' or State = NULL
```

When a field name contains spaces, you must enclose the field name in brackets. For example:

```
[Home State] = 'CA' or [Home State] = 'MA'
```

Filter expressions on remote SQL tables and on client datasets support field comparisons. For example:

```
Field1 > Field2
```

Field comparisons are not supported against local tables (Paradox, dBASE, Access, FoxPro).

For a complete list of operators you can use in filter strings, see SettingTheFilterProperty.

Note: Filters are not supported on unidirectional datasets. If you set the Filter property of a unidirectional dataset, it raises an exception.

The FilterOptions (see page 7027) property controls case sensitivity and filtering on partial comparisons.

Tip: Applications can set Filter at runtime to change the dataset's filtering condition (for example, in response to user input).

See Also

Filtered (see page 7027)

FilterOptions (see page 7027)

OnFilterRecord (see page 7055)

Locate (see page 6970)

Lookup (see page 6970)

Creating Filters

Navigating Records in a Filtered Dataset

Setting the Filter Property

Displaying and Editing a Subset of Data Using Filters

39.1.17.5.30 TDataSet.Filtered Property

Specifies whether or not filtering is active for a dataset.

Pascal

```
property Filtered: Boolean;
```

C++

```
__property Boolean Filtered;
```

Description

Check Filtered to determine whether or not dataset filtering is in effect. If Filtered is true, then filtering is active. To apply filter conditions specified in the Filter (see page 7026) property or the OnFilterRecord (see page 7055) event handler, set Filtered to true. Because unidirectional datasets do not support filters, when you set Filtered to true on a unidirectional dataset, it raises an exception.

Note: When filtering is enabled, user edits to a record may mean that the record no longer meets a filter's test condition. The next time an attempt is made to the read the record from the dataset while the filter is in effect, the record is not visible. If that happens, the next record that passes the filter condition becomes the active record.

See Also

Filter (see page 7026)

FilterOptions (see page 7027)

OnFilterRecord (see page 7055)

Locate (see page 6970)

Lookup (see page 6970)

Enabling and Disabling Filtering

Setting the Filter Property

39.1.17.5.31 TDataSet.FilterOptions Property

Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.

Pascal

```
property FilterOptions: TFilterOptions;
```

C++

```
__property TFilterOptions FilterOptions;
```

Description

Set FilterOptions to specify whether or not filtering is case insensitive when filtering on string or character fields, and whether or not partial comparisons for matching filter conditions is allowed.

Note: Because unidirectional datasets do not support filters, when you set the FilterOptions property of a unidirectional dataset, it raises an exception.

By default, FilterOptions is set to an empty set. For filters based on string fields, set FilterOptions to foCaseInsensitive to catch all variations on a string regardless of capitalization.

When a string in a filter ends with an asterisk (*), it can be used to match partial strings. To disable matching of partial strings and to treat the asterisk as a literal character in string comparisons, set FilterOptions to include foNoPartialCompare.

See Also

Filter (see page 7026)

Filtered (see page 7027)

OnFilterRecord (see page 7055)

Locate (see page 6970)

Lookup (see page 6970)

Setting Filter Options

39.1.17.5.32 TDataSet.Found Property

Indicates whether or not moving to a different record is successful.

Pascal

```
property Found: Boolean;
```

C++

```
__property Boolean Found;
```

Description

Check the status of Found to determine if a call to FindFirst (see page 6939), FindLast (see page 6940), FindNext (see page 6940) or, FindPrior (see page 6941) succeeds. If Found is true, success is indicated. If false, the move to a different record failed.

See Also

Filter (see page 7026)

FindFirst (see page 6939)

FindLast (see page 6940)

FindNext (see page 6940)

FindPrior (see page 6941)

Navigating Records in a Filtered Dataset

39.1.17.5.33 TDataSet.InternalCalcFields Property

Indicates whether the dataset includes internally calculated fields.

Pascal

```
property InternalCalcFields: Boolean;
```

C++

```
__property Boolean InternalCalcFields;
```

Description

Read InternalCalcFields to discover if the dataset includes fields whose values are provided by the data source but that are not a fields in the physical database table or query. Internally calculated fields are calculated by SQL servers or the Borland (see page 1761) Database Engine in a live query view.

On TClientDataSet, internally calculated fields are stored with the rest of the data, allowing the developer to sort and filter on the calculated fields. However, internally calculated fields on client datasets are not updated when the data they depend on is updated.

Note: Do not confuse internally calculated fields with calculated fields, which are computed in an OnCalcFields (see page 7053) event.

See Also

InternalCalcField (see page 7192)

39.1.17.5.34 TDataSet.IsUniDirectional Property

Indicates whether the dataset is unidirectional.

Pascal

```
property IsUniDirectional: Boolean;
```

C++

```
__property Boolean IsUniDirectional;
```

Description

IsUniDirectional indicates whether the dataset is a descendant of TCustomSQLDataSet. Unidirectional datasets raise exceptions on all navigation methods except for First (see page 6942) and Next (see page 6973). They do not support filters, bookmarks, lookup fields, or any other features that require an internal record buffer. You can't use a unidirectional dataset as the source to a data-aware grid.

See Also

TCustomSQLDataSet (see page 20150)

39.1.17.5.35 TDataSet.Modified Property

Indicates whether the active record is modified.

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

Check Modified to determine if the active record is modified. If Modified is true, the active record is modified. If false, the active record is not modified.

Note: In general, an application need not check the status of Modified. Properties, events, and methods of TDataSet (see page 6894) and its descendants that modify records generally check this status automatically and take appropriate actions based on its value.

See Also

State (see page 7036)

Delete (see page 6924)

Edit (see page 6932)

Insert (see page 6959)

Post (see page 6977)

39.1.17.5.36 TDataSet.NestedDataSetClass Property

Indicates the class type to use when creating nested datasets.

Pascal

```
property NestedDataSetClass: TDataSetClass;
```

C++

```
__property TDataSetClass NestedDataSetClass;
```

Description

TDataSet (see page 6894) descendants use NestedDataSetClass to specify the class that can be used for nested datasets. By default, this value is the same as the class of the dataset instance.

NestedDataSetClass is used to check that nested detail sets are the appropriate type when they are assigned to a data set field (by assigning a value to their DataSetField (see page 7013) property).

See Also

NestedDataSets (see page 7030)

DataSetField (see page 7013)

39.1.17.5.37 TDataSet.NestedDataSets Property

Lists all nested datasets.

Pascal

```
property NestedDataSets: TList;
```

C++

```
__property TList NestedDataSets;
```

Description

NestedDataSets is a list of the nested detail sets belonging to the data set. Each entry in the list is a dataset of the type specified by NestedDataSetClass (see page 7030).

Do not use NestedDataSets to assign values to dataset fields. Instead, set the DataSetField (see page 7013) property of the detail set to reference the appropriate field component.

See Also

NestedDataSetClass (see page 7030)

TDataSetField (see page 7060)

DataSetField (see page 7013)

39.1.17.5.38 TDataSet.ObjectView Property

Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.

Pascal

```
property ObjectView: Boolean;
```

C++

```
__property Boolean ObjectView;
```

Description

ObjectView affects the way the Fields (see page 7023) property stores object fields and the way data-aware grids display ADT and array fields.

When ObjectView is true, the fields are stored hierarchically in the Fields (see page 7023) property, meaning any child fields of an object field are referenced by the object field and don't appear sequentially after the object field in the TFields (see page 7211):: .Fields (see page 7023) array. When ObjectView is false, the fields are stored sequentially in the Fields (see page 7023) property, meaning any child fields of an object field are stored after the object field as siblings in the Fields (see page 7023) array.

When ObjectView is false, object field types, such as TADTField (see page 6736), are not created. This switch is provided for increased compatibility with older data-aware controls, which may not handle object field types properly. The default is false for TBDEDataSet, and true for TClientDataSet.

Note: If the ObjectView property of the dataset provided to a TClientDataSet is false, then the TClientDataSet's ObjectView property always acts as false, since the data received is already flattened.

See Also

FieldDefList (see page 7019)

Fields (see page 7023)

Displaying ADT and Array Fields

Using Nested Detail Tables

Using Object Fields

Working with ADT Fields

Working with Array Fields

39.1.17.5.39 TDataSet.RecNo Property

Indicates the active record in the dataset.

Pascal

```
property RecNo: Integer;
```

C++

```
__property int RecNo;
```

Description

As implemented in TDataSet (see page 6894), the value of RecNo is always -1. Ordinarily an application does not access

RecNo at the TDataSet (see page 6894) level. Instead a re-implemented RecNo property in a descendant class such as TTable TClientDataSet is accessed. RecNo provides a fallback property for derived datasets that do not support record numbers.

Delphi Examples:

```
{
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

RecordCount (see page 7032)

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Prior (see page 6979)

39.1.17.5.40 TDataSet.RecordCount Property

Indicates the total number of records associated with the dataset.

Pascal

```
property RecordCount: Integer;
```

C++

```
__property int RecordCount;
```

Description

As implemented in TDataSet (see page 6894), RecordCount is always -1. Ordinarily an application does not access RecordCount at the TDataSet (see page 6894) level. Instead a redeclared and implemented RecordCount property in a descendant class is accessed. RecordCount provides a fallback property for derived dataset classes that do not reimplement the property access method.

C++ Examples:

```
/*
Reads through all records in the Customers table.
```

```
Updates the ProgressBar accordingly.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ProgressBar1->Min = 0;
  ProgressBar1->Max = Customers->RecordCount;
  Customers->First();
  for (int i = ProgressBar1->Min; i <= ProgressBar1->Max; i++)
  {
    ProgressBar1->Position = i;
    Customers->Next();
    // do something with the current record
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Customers = new TTable(Form1);
  Customers->Active = false; // The Table component must not be active
  Customers->DatabaseName = "DBDEMOS";
  Customers->TableType = ttParadox;
  Customers->TableName = "CustInfo";
  Customers->Active = False;
  if (Customers->Exists) // Don't overwrite an existing table
  {
    Customers->Close();
    Customers->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Customers->FieldDefs->Clear();
  TFieldDef *pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Customers->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Customers->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Customers->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Customers->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Customers->CreateTable();
  Customers->Active = True;
  for (int i = 1; i <= 20; i++)
    Customers->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  DS2->DataSet = Customers;
  DBGrid2->DataSource->DataSet = Customers;
  Customers->Active = True;
}
```

Delphi Examples:

```
{
Reads through all records in the Customers table.
Updates the ProgressBar accordingly.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
begin
  with ProgressBar1 do
```

```
    begin
      Min := 0;
      Max := Customers.RecordCount;
      Customers.First;
      for i := Min to Max do
      begin
        Position := i;
        Customers.Next;
        // do something with the current record
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    Customers:= TTable.Create(Form1);
    with Customers do
    begin
      DatabaseName := 'DBDEMOS';
      TableType := ttParadox;
      TableName := 'CustInfo';
      Active := False;
  //  Overwrite any existing table
      if Customers.Exists then
      begin
        Customers.Close;
        Customers.DeleteTable;
      end;
      begin
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
        { Next, describe the fields in the table }
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
```

```
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Customers.Active:= True;
        for i := 1 to 100 do
          Customers.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Customers;
    DBGrid2.DataSource.DataSet:= Customers;
    Customers.Active:= True;
  end;
```

See Also

RecNo (see page 7031)

RecordSize (see page 7035)

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Prior (see page 6979)

39.1.17.5.41 TDataSet.RecordSize Property

Indicates the size of a record in the dataset.

Pascal

```
property RecordSize: Word;
```

C++

```
__property Word RecordSize;
```

Description

RecordSize indicates the size needed for internal record buffers. In TDataSet (see page 6894), the value of RecordSize is always 0 because TDataSet (see page 6894) does not implement record buffering. Descendant classes that buffer records override the property getter to return the number of bytes in a record buffer.

See Also

RecordCount (see page 7032)

RecNo (see page 7031)

Open (see page 6976)

39.1.17.5.42 TDataSet.Reserved Property

For internal use only.

Pascal

```
property Reserved: Pointer;
```

C++

```
__property void * Reserved;
```

Description

Do not use the Reserved property. It is for internal use only.

39.1.17.5.43 TDataSet.SparseArrays Property

Determines whether a unique TField (see page 7115) object is created for each element of an array field.

Pascal

```
property SparseArrays: Boolean;
```

C++

```
__property Boolean SparseArrays;
```

Description

When opening a dataset with an array field, SparseArrays determines whether a unique field component is created for each element of the array field. The default is false, where a TField (see page 7115) descendant is created for each element of the array field. If you plan to bind data-aware controls to elements of the array field, SparseArrays must be set to false. Setting SparseArrays to true conserves memory, but does not allow applications to reference individual elements of the array field.

See Also

ObjectView (see page 7031)

TField (see page 7115)

Working with Array Fields

39.1.17.5.44 TDataSet.State Property

Indicates the current operating mode of the dataset.

Pascal

```
property State: TDataSetState;
```

C++

```
__property TDataSetState State;
```

Description

Examine State to determine the current operating mode of the dataset. State determines what can be done with data in a dataset, such as editing existing records or inserting new ones. The dataset state constantly changes as an application processes data.

Opening a dataset changes State from dsInactive to dsBrowse. An application can call Edit (see page 6932) to put a dataset into dsEdit state, or call Insert (see page 6959) to put a dataset into dsInsert state. If a dataset is a TTable or TClientDataSet object, an application can call SetKey or SetRange to put the dataset into dsSetKey state.

Posting or canceling edits, insertions, or deletions, changes State from its current state to dsBrowse. Closing a dataset changes its state to dsInactive.

Some states, such as dsCalcFields, dsFilter, dsNewValue, dsOldValue, and dsCurValue cannot be seen or set directly by an application. These states are automatically set when OnCalcFields (see page 7053) and OnFilterRecord (see page 7055) events occur, or when an application accesses certain field properties.

See Also

Active (see page 7004)

CachedUpdates (see page 9932)

Filtered (see page 7027)

OnCalcFields (see page 7053)

OnFilterRecord (see page 7055)

Close (see page 6920)

Edit (see page 6932)

Insert (see page 6959)

Open (see page 6976)

TField (see page 7115)

SetKey (see page 10208)

SetRange (see page 10209)

Responding to Changes Mediated by the Data Source

Determining Dataset States

Using Internally Calculated Fields in Client Datasets

Programming a Calculated Field

39.1.17.6 TDataSet Events

The events of the TDataSet class are listed here.

Public Events

	Name	Description
	AfterCancel (see page 7038)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 7040)	Occurs after an application closes a dataset.
	AfterDelete (see page 7041)	Occurs after an application deletes a record.
	AfterEdit (see page 7044)	Occurs after an application starts editing a record.
	AfterInsert (see page 7044)	Occurs after an application inserts a new record.
	AfterOpen (see page 7045)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 7046)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 7047)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 7047)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 7048)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 7048)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 7049)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 7049)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 7050)	Occurs before an application enters insert mode.
	BeforeOpen (see page 7051)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 7052)	Occurs before an application posts changes for the active record to the database or change log.

	BeforeRefresh (see page 7052)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 7053)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 7053)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 7054)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 7054)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 7056)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 7056)	Occurs when an application attempts to modify or insert a record and an exception is raised.

39.1.17.6.1 TDataSet.AfterCancel Event

Occurs after an application completes a request to cancel modifications to the active record.

Pascal

```
property AfterCancel: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterCancel;
```

Description

Write an AfterCancel event handler to take specific action after an application cancels changes to the active record. AfterCancel is called by the Cancel (see page 6916) method after it updates the current position, releases the lock on the active record if necessary, and sets the dataset state to dsBrowse. If an application requires additional processing before returning control to a user after a Cancel (see page 6916) event, code it in the AfterCancel event.

Delphi Examples:

```
{
This example updates the form's status bar with a message
when an AfterCancel event occurs.  Set the StatusBar
SimplePanel property to True.
}
procedure TForm1.Table1AfterCancel(DataSet: TDataSet);
begin
  StatusBar1.SimpleText := 'Record changes cancelled for ' + DataSet.Name;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Table1.Cancel;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  Table1.Post
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Name := 'CustInfo';
```

```
      Table1.Active := False;
      { Don't overwrite an existing table }
      if Table1.Exists then
      begin
          Table1.Close;
          Table1.DeleteTable;
      end;
      begin
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
        { Next, describe the fields in the table }
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.AfterCancel:= Table1AfterCancel;
    Table1.Active:= True;
  end;
```

See Also

BeforeCancel (see page 7048)

Cancel (see page 6916)

State (see page 7036)

39.1.17.6.2 TDataSet.AfterClose Event

Occurs after an application closes a dataset.

Pascal

```
property AfterClose: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterClose;
```

Description

Write an AfterClose event handler to take specific action immediately after an application closes a dataset. For example, as a security measure, an application might clear a PASSWORD entry from the Params property of a TSQLConnection component when the dataset is closed.

AfterClose is called after a dataset is closed and the dataset state is set to dsInactive.

Delphi Examples:

```
{
This example adds an entry to a memo with a message when an
AfterOpen event or an AfterClose event occurs.  Populate the
ClientDataSet events with CheckButtonActive.
}
procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  if (Button1.Enabled) then
    Memo2.Lines.Add('Now that the dataset is open, record information is available.')
  else
    Memo2.Lines.Add('Now that the dataset is closed, record information is not available.');
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
  RemoteServer1.Connected := False;
  with RemoteServer1 do
  begin
```

```
    for I := 0 to DataSetCount - 1 do
      DataSets[I].DisableControls;
  end;
end;

procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
```

See Also

Active (see page 7004)

State (see page 7036)

BeforeClose (see page 7048)

Close (see page 6920)

Params (see page 20253)

Opening and Closing Datasets

39.1.17.6.3 TDataSet.AfterDelete Event

Occurs after an application deletes a record.

Pascal

```
property AfterDelete: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterDelete;
```

Description

Write an AfterDelete event handler to take specific action immediately after an application deletes the active record in a dataset. AfterDelete is called by Delete (see page 6924) after it deletes the record, sets the dataset state to dsBrowse, and repositions the current record.

C++ Examples:

```
/*
This example displays a message on the form's status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
*/
void __fastcall TForm1::Table1AfterDelete(TDataSet *DataSet)
{
  StatusBar1->SimpleText = Format(
    "There are now %d records in the table",
    ARRAYOFCONST(((int)DataSet->RecordCount)));
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Table1->Delete();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Table1 = new TTable(Form1);
  Table1->Active = false; // The Table component must not be active
```

```
  Table1->DatabaseName = "DBDEMOS";
  Table1->TableType = ttParadox;
  Table1->TableName = "CustInfo";
  Table1->Active = False;
  if (!Table1->Exists) // Don't overwrite an existing table
  {
    Table1->Close();
    Table1->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Table1->FieldDefs->Clear();
  TFieldDef *pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Table1->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Table1->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Table1->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Table1->CreateTable();
  Table1->Active = True;
  for (int i = 1; i <= 20; i++)
    Table1->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  DS2->DataSet = Table1;
  DBGrid2->DataSource->DataSet = Table1;
  Table1->AfterDelete = Table1AfterDelete;
  Table1->Active = True;
}
```

Delphi Examples:

```
{
This example displays a message on the form’s status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
}
procedure TForm1.Table1AfterDelete(DataSet: TDataSet);
begin
  StatusBar1.SimpleText := Format('There are now %d records in the table',
[DataSet.RecordCount]);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Delete;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Name := 'CustInfo';
```

```
      Table1.Active := False;
      { Don't overwrite an existing table }
      if Table1.Exists then
      begin
          Table1.Close;
          Table1.DeleteTable;
      end;
      begin
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
        { Next, describe the fields in the table }
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.AfterDelete:= Table1AfterDelete;
    Table1.Active:= True;
  end;
```

See Also

State (see page 7036)

BeforeDelete (see page 7049)

Delete (see page 6924)

39.1.17.6.4 TDataSet.AfterEdit Event

Occurs after an application starts editing a record.

Pascal

```
property AfterEdit: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterEdit;
```

Description

Write an AfterEdit event handler to take specific action immediately after dataset enters edit mode. AfterEdit is called by Edit (see page 6932) after it enables editing of a record, recalculates calculated fields, and calls the data event handler to process a record change.

Delphi Examples:

```
{
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

BeforeEdit (see page 7049)

Edit (see page 6932)

39.1.17.6.5 TDataSet.AfterInsert Event

Occurs after an application inserts a new record.

Pascal

```
property AfterInsert: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterInsert;
```

Description

Write an AfterInsert event handler to take specific action immediately after an application inserts a record. The Insert (see page 6959) and Append (see page 6910) methods generate an AfterInsert event after inserting or appending a new record.

Delphi Examples:

```
{
This example updates a memo with a message when an
AfterInsert event or a BeforeInsert event occurs.
}
procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After inserting a record');
end;

procedure TForm1.ClientDataSet1BeforeInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before inserting a record');
end;
```

See Also

BeforeInsert (see page 7050)

Append (see page 6910)

Insert (see page 6959)

39.1.17.6.6 TDataSet.AfterOpen Event

Occurs after an application completes opening a dataset and before any data access occurs.

Pascal

```
property AfterOpen: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterOpen;
```

Description

Write an AfterOpen event handler to take specific action immediately after an application opens the dataset. AfterOpen is called after the dataset establishes access to its data and the dataset is put into dsBrowse state. For example, an AfterOpen event handler might check an ini file to determine the last record touched in the dataset the previous time the application ran, and position the dataset at that record.

Delphi Examples:

```
{
This example adds an entry to a memo with a message when an
AfterOpen event or an AfterClose event occurs.  Populate the
ClientDataSet events with CheckButtonActive.
}
procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  if (Button1.Enabled) then
    Memo2.Lines.Add('Now that the dataset is open, record information is available.')
  else
    Memo2.Lines.Add('Now that the dataset is closed, record information is not available.');
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
  RemoteServer1.Connected := False;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].DisableControls;
  end;
end;

procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
```

See Also

Active (see page 7004)

State (see page 7036)

BeforeOpen (see page 7051)

Open (see page 6976)

Opening and Closing Datasets

39.1.17.6.7 TDataSet.AfterPost Event

Occurs after an application writes the active record to the database or change log and returns to browse state.

Pascal

```
property AfterPost: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterPost;
```

Description

Write an AfterPost event handler to take specific action immediately after an application posts a change to the active record. AfterPost is called after a modification or insertion is made to a record.

Delphi Examples:

```
{
```

```
This example updates the form's status bar with a message
when an AfterPost event occurs.
}
procedure TForm1.ClientDataSet1AfterPost(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before posting a record');
end;
```

See Also

BeforePost (see page 7052)

Post (see page 6977)

39.1.17.6.8 TDataSet.AfterRefresh Event

Occurs after an application refreshes the data in the dataset.

Pascal

```
property AfterRefresh: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterRefresh;
```

Description

Write an AfterRefresh event handler to take specific action immediately after an application has updated the records in the dataset. AfterRefresh is generated by calls to the Refresh (see page 6991) method.

See Also

BeforeRefresh (see page 7052)

Refresh (see page 6991)

39.1.17.6.9 TDataSet.AfterScroll Event

Occurs after an application scrolls from one record to another.

Pascal

```
property AfterScroll: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterScroll;
```

Description

Write an AfterScroll event handler to take specific action immediately after an application scrolls to another record as a result of a call to the First (see page 6942), Last (see page 6968), MoveBy (see page 6971), Next (see page 6973), Prior (see page 6979), FindKey, FindFirst (see page 6939), FindNext (see page 6940), FindLast (see page 6940), FindPrior (see page 6941), and Locate (see page 6970) methods. AfterScroll is called after all other events triggered by these methods and any other methods that switch from record to record in the dataset.

See Also

BeforeScroll (see page 7053)

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Prior (see page 6979)

Navigating Datasets

39.1.17.6.10 TDataSet.BeforeCancel Event

Occurs before an application executes a request to cancel changes to the active record.

Pascal

```
property BeforeCancel: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeCancel;
```

Description

Write a BeforeCancel event to take specific action before an application carries out a request to cancel changes. BeforeCancel is called by the Cancel (see page 6916) method before it cancels a dataset operation such as Edit (see page 6932), Insert (see page 6959), or Delete (see page 6924).

An application might use the BeforeCancel event to record a user's changes in an undo buffer.

See Also

AfterCancel (see page 7038)

Delete (see page 6924)

Edit (see page 6932)

Insert (see page 6959)

39.1.17.6.11 TDataSet.BeforeClose Event

Occurs immediately before the dataset closes.

Pascal

```
property BeforeClose: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeClose;
```

Description

Write a BeforeClose event to take specific action before an application closes a dataset. Calling Close (see page 6920) or setting the Active (see page 7004) property to false results in a call to the BeforeClose event handler.

See Also

Active (see page 7004)

AfterClose (see page 7040)

Close (see page 6920)

Opening and Closing Datasets

Posting Data

39.1.17.6.12 TDataSet.BeforeDelete Event

Occurs before an application attempts to delete the active record.

Pascal

```
property BeforeDelete: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeDelete;
```

Description

Write a BeforeDelete event handler to take specific action before an application deletes the active record. BeforeDelete is called by Delete (see page 6924) before it actually deletes a record.

In Delphi, making use of this event an application might, for example, display a dialog box asking for confirmation before deleting the record. On denial of confirmation, the application could abort the deletion by calling the Abort procedure.

See Also

AfterDelete (see page 7041)

Delete (see page 6924)

Abort (see page 22786)

39.1.17.6.13 TDataSet.BeforeEdit Event

Occurs before an application enters edit mode for the active record.

Pascal

```
property BeforeEdit: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeEdit;
```

Description

Write a BeforeEdit event handler to take specific action before an application enables editing of the active record. For example, an application that keeps a log of database edits could use the BeforeEdit event to record the edit request, time, and user before entering edit state.

Delphi Examples:

```
{
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;
```

```
procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

State (see page 7036)

AfterEdit (see page 7044)

Edit (see page 6932)

39.1.17.6.14 TDataSet.BeforeInsert Event

Occurs before an application enters insert mode.

Pascal

```
property BeforeInsert: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeInsert;
```

Description

Write a BeforeInsert event handler to take specific action before an application inserts or appends a new record. The Insert (see page 6959) or Append (see page 6910) method generates a BeforeInsert method before it sets the dataset into dsInsert state.

Delphi Examples:

```
{
This example updates a memo with a message when an
AfterInsert event or a BeforeInsert event occurs.
}
procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After inserting a record');
end;

procedure TForm1.ClientDataSet1BeforeInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before inserting a record');
end;
{
This example uses the BeforeInsert event to do data
validation. If the StrToInt function raises an exception,
the edit control’s contents are set to a valid value so
that the assignment to the INTEGER field in the dataset
will succeed.  Continuing after the exception causes a
'0' to be placed in the PORTA field.
DataSet design:
This example requires a ClientDataSet, a DBGrid, a
DataSource and a DBNavigator.  Name the ClientDataSet CDS.
Set the DataSet property in the DataSource to CDS and name
that DataSource DS.  Set the DataSource in the DBNavigator
to DS the DataSource in the DBGrid to DS.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
type to Integer.  The file used to load CDS must have an
```

```
  attribute name of HOST with an attribute type of string and
  an attribute name of PORTA with an attribute type of integer.
  Add DB, DBCtrls, Grids, DBGrids and DBClient to the uses clause.
  }
  {$IFNDEF UNICODE}
  uses SwSystem;
  {$ENDIF}

  procedure TForm1.Button3Click(Sender: TObject);
  begin
    CDS.Insert;
    CDS.FieldByName('PORTA').AsInteger := StrToInt(Edit1.Text);
    CDS.Post;
  end;

  procedure TForm1.CDSBeforeInsert(DataSet: TDataSet);
  begin
    try
    { Make sure edit field can be converted to integer --
     this will raise an exception if it can't }
      StrToInt(Edit1.Text);
    except
      Edit1.Text := '0';
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
  {$IFDEF UNICODE}
    CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
  {$ELSE}
    CDS.LoadFromFile(gsAppPath + 'CDS.XML');
  {$ENDIF}
  end;
```

See Also

State (see page 7036)

AfterInsert (see page 7044)

Append (see page 6910)

Insert (see page 6959)

39.1.17.6.15 TDataSet.BeforeOpen Event

Occurs before an application executes a request to open a dataset.

Pascal

```
property BeforeOpen: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeOpen;
```

Description

Write a BeforeOpen event handler to take specific action before an application opens a dataset for viewing or editing. BeforeOpen is triggered when an application sets the Active (see page 7004) property to true for a dataset or an application calls Open (see page 6976).

See Also

Active (see page 7004)

AfterOpen (see page 7045)

Open (see page 6976)

Opening and Closing Datasets

39.1.17.6.16 TDataSet.BeforePost Event

Occurs before an application posts changes for the active record to the database or change log.

Pascal

```
property BeforePost: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforePost;
```

Description

Write a BeforePost event handler to take specific action before an application posts dataset changes. BeforePost is triggered when an application calls the Post (see page 6977) method. Post (see page 6977) checks to make sure all required fields are present, then calls BeforePost before posting the record.

An application might use BeforePost to perform validity checks on data changes before committing them. If it encountered a validity problem, it could call Abort to cancel the Post (see page 6977) operation (Delphi) or throw an exception (C++).

Delphi Examples:

```
{
This example checks for a valid entry in a TDBEdit control
and calls the Abort procedure if the control is empty; Abort
cancels the post before it happens.
}
procedure TForm1.ClientDataSet1BeforePost(DataSet: TDataSet);
begin
  if DBEdit1.Text = '' then
    Abort;
  DataSet.Fields[2].AsString := DBEdit1.Text;
end;
```

See Also

AfterPost (see page 7046)

Post (see page 6977)

Abort (see page 22786)

Working with Field Component Methods at Runtime

39.1.17.6.17 TDataSet.BeforeRefresh Event

Occurs immediately before an application refreshes the data in the dataset.

Pascal

```
property BeforeRefresh: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeRefresh;
```

Description

Write a BeforeRefresh event handler to take specific action immediately before an application updates the records in the dataset.

BeforeRefresh is generated by calls to the Refresh (see page 6991) method.

See Also

AfterRefresh (see page 7047)

Refresh (see page 6991)

39.1.17.6.18 TDataSet.BeforeScroll Event

Occurs before an application scrolls from one record to another.

Pascal

```
property BeforeScroll: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeScroll;
```

Description

Write a BeforeScroll event handler to take specific action immediately before an application scrolls to another record as a result of a call to the First (see page 6942), Last (see page 6968), MoveBy (see page 6971), Next (see page 6973), Prior (see page 6979), FindKey, FindFirst (see page 6939), FindNext (see page 6940), FindLast (see page 6940), FindPrior (see page 6941), and Locate (see page 6970) methods. BeforeScroll is called before all other events triggered by these methods and any other methods that switch from record to record in the dataset.

See Also

AfterScroll (see page 7047)

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Prior (see page 6979)

Navigating Datasets

39.1.17.6.19 TDataSet.OnCalcFields Event

Occurs when an application recalculates calculated fields.

Pascal

```
property OnCalcFields: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent OnCalcFields;
```

Description

Write an OnCalcFields event handler to take specific action when an application recalculates calculated fields. A calculated field is one that derives its value from the values in one or more fields in the dataset, sometimes with additional processing.

OnCalcFields is triggered when:

A dataset is opened.

A dataset is put into dsEdit state.

A record is retrieved from a database.

When the AutoCalcFields (see page 7006) property is true, OnCalcFields is also triggered when:

Focus moves from one visual control to another, or from one column to another is a data-aware grid control and modifications were made to the record.

Note: When the AutoCalcFields (see page 7006) property is true, an OnCalcFields event handler should not modify the dataset (or a linked dataset if it is part of a master-detail relationship), because such modifications retrigger the OnCalcField event, leading to infinite recursion.

If an application permits users to change data, OnCalcFields is frequently triggered. To reduce the frequency with which OnCalcFields occurs, set AutoCalcFields (see page 7006) to false.

Warning: When the dataset is the master table of a master-detail relationship, OnCalcFields occurs before detail sets have been synchronized with the master table.

See Also

Active (see page 7004)

AutoCalcFields (see page 7006)

Open (see page 6976)

Post (see page 6977)

Calculating Fields

Using Internally Calculated Fields in Client Datasets

Defining a Calculated Field

39.1.17.6.20 TDataSet.OnDeleteError Event

Occurs when an application attempts to delete a record and an exception is raised.

Pascal

```
property OnDeleteError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnDeleteError;
```

Description

Write an OnDeleteError event handler to handle exceptions that occur when an attempt to delete a record fails.

See Also

OnEditError (see page 7054)

OnPostError (see page 7056)

Delete (see page 6924)

39.1.17.6.21 TDataSet.OnEditError Event

Occurs when an application attempts to modify or insert a record and an exception is raised.

Pascal

```
property OnEditError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnEditError;
```

Description

Write an OnEditError event handler to handle exceptions that occur when an attempt to edit a record fails.

See Also

OnDeleteError (see page 7054)

OnPostError (see page 7056)

Edit (see page 6932)

Insert (see page 6959)

39.1.17.6.22 TDataSet.OnFilterRecord Event

Occurs each time a different record in the dataset becomes the active record and filtering is enabled.

Pascal

```
property OnFilterRecord: TFilterRecordEvent;
```

C++

```
__property TFilterRecordEvent OnFilterRecord;
```

Description

Write an OnFilterRecord event handler to specify for each record in a dataset whether it should be visible to the application. To indicate that a record passes the filter condition, the OnFilterRecord event handler must set the Accept parameter to true. To exclude a record, set the Accept parameter to false. Most TDataSet (see page 6894) descendants (such as ADO datasets or BDE-based datasets) initialize Accept to true before calling the OnFilterRecord event handler.

Warning: Do not supply an OnFilterRecord event handler to a unidirectional dataset. Unidirectional datasets do not support filters, and assigning an OnFilterRecord event handler causes the dataset to raise an exception.

Filtering is enabled if the Filtered (see page 7027) property is true. When an application is processing a filter, the State (see page 7036) property for the dataset is dsFilter.

Use an OnFilterRecord event handler to filter records using a criterion that can't be implemented using the Filter (see page 7026) property. For example, using the Filter (see page 7026) property, field comparisons are not supported against local tables (Paradox, dBASE, Access, FoxPro), but an OnFilterRecord event handler can implement any criterion at all.

Tip: Be sure that any interactions between the Filter (see page 7026) property and the OnFilterRecord event handler do not result in an empty filter set when they are used simultaneously in an application.

See Also

Filter (see page 7026)

Filtered (see page 7027)

FilterOptions (see page 7027)

State (see page 7036)

Creating Filters

Navigating Records in a Filtered Dataset

Displaying and Editing a Subset of Data Using Filters

Writing an OnFilterRecord Event Handler

39.1.17.6.23 TDataSet.OnNewRecord Event

Occurs when an application inserts or appends a new dataset record.

Pascal

```
property OnNewRecord: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent OnNewRecord;
```

Description

Write an OnNewRecord event handler to take specific actions as an application inserts or appends a new record. OnNewRecord is called as part of the insert or append process. An application might use the OnNewRecord event to set initial values for a record or as a way of implementing cascading insertions in related datasets.

See Also

AfterInsert (see page 7044)

BeforeInsert (see page 7050)

39.1.17.6.24 TDataSet.OnPostError Event

Occurs when an application attempts to modify or insert a record and an exception is raised.

Pascal

```
property OnPostError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnPostError;
```

Description

Write an OnPostError event handler to handle exceptions that occur when an attempt to post a record fails.

See Also

OnDeleteError (see page 7054)

OnEditError (see page 7054)

Post (see page 6977)

39.1.18 TDataSetDesigner Class

TDataSetDesigner is the base class for all dataset designers.

Class Hierarchy

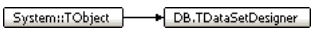

Pascal

```
TDataSetDesigner = class(TObject);
```

C++

```
class TDataSetDesigner : public TObject;
```

File

DB

Description

Use TDataSetDesigner when defining a design-time editor, such as the Fields editor, that enables a developer to specify the fields that belong to a dataset at design time. The dataset designer is specified as the designer property of a dataset that uses it to enable design-time specification of its structure.

See Also

Tdataset

RegisterFields

39.1.18.1 TDataSetDesigner Members

The following tables list the members exposed by TDataSetDesigner.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDesign (see page 7059)	Ensures that the dataset is not active while design changes are made.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 7058)	Creates and initializes an instance of TDataSetDesigner (see page 7056).
	DataEvent (see page 7059)	Provides the interface for a method that can respond to data events.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 7058)	Disposes of an instance of TDataSetDesigner (see page 7056).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndDesign (see page 7059)	Reactivates the dataset if it was active when designing began.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	DataSet (see page 7060)	Provides a reference to the dataset currently being designed.

39.1.18.2 TDataSetDesigner.Create Constructor

Creates and initializes an instance of TDataSetDesigner (see page 7056).

Pascal

```
constructor Create(DataSet: TDataSet);
```

C++

```
__fastcall TDataSetDesigner(TDataSet DataSet);
```

Description

Call Create from the component editor of a specialized dataset to create an instance of the dataset designer. Create initializes the DataSet (see page 7060) property of the dataset designer and the Designer property of the dataset.

See Also

TComponentEditor (see page 10927)

DataSet (see page 7060)

Designer (see page 7015)

Destroy (see page 7058)

39.1.18.3 TDataSetDesigner.Destroy Destructor

Disposes of an instance of TDataSetDesigner (see page 7056).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataSetDesigner();
```

Description

Do not call the Destroy method directly. Instead, when freeing the dataset designer from a component editor, use the Free method. Free checks that the dataset designer reference is not nil, and only then calls Destroy.

Destroy sets the Designer property of the associated dataset to nil (Delphi) or NULL (C++) before destroying the instance of TDataSetDesigner (see page 7056).

See Also

Free (see page 22300)

TComponentEditor (see page 10927)

Create (see page 7058)

39.1.18.4 TDataSetDesigner Methods

The methods of the TDataSetDesigner class are listed here.

Public Methods

	Name	Description
	BeginDesign (see page 7059)	Ensures that the dataset is not active while design changes are made.
	DataEvent (see page 7059)	Provides the interface for a method that can respond to data events.
	EndDesign (see page 7059)	Reactivates the dataset if it was active when designing began.

39.1.18.4.1 TDataSetDesigner.BeginDesign Method

Ensures that the dataset is not active while design changes are made.

Pascal

```
procedure BeginDesign;
```

C++

```
__fastcall BeginDesign();
```

Description

BeginDesign is called before changing the structure of the associated dataset. BeginDesign deactivates the dataset and disables any associated controls until the EndDesign (see page 7059) method is called.

See Also

Active (see page 7004)

EndDesign (see page 7059)

DataSet (see page 7060)

39.1.18.4.2 TDataSetDesigner.DataEvent Method

Provides the interface for a method that can respond to data events.

Pascal

```
procedure DataEvent(Event: TDataEvent; Info: Longint); virtual;
```

C++

```
virtual __fastcall DataEvent(TDataEvent Event, Longint Info);
```

Description

Call DataEvent to pass information about data events to the dataset designer. The Event parameter specifies the type of event that is occurring. Use the Info parameter to specify additional information about the event. The value of the Info parameter depends on the type of data event.

The DataEvent method for TDataSetDesigner (see page 7056) does nothing when informed of a data event. Override this method to allow the data set designer to respond to different types of events.

39.1.18.4.3 TDataSetDesigner.EndDesign Method

Reactivates the dataset if it was active when designing began.

Pascal

```
procedure EndDesign;
```

C++

```
__fastcall EndDesign();
```

Description

Call EndDesign when changes to the dataset are complete. EndDesign re-enables all controls that use the dataset and resets the Active property of the dataset to the value it help when BeginDesign (see page 7059) was called.

See Also

Active (see page 7004)

BeginDesign (see page 7059)

DataSet (see page 7060)

39.1.18.5 TDataSetDesigner Properties

The properties of the TDataSetDesigner class are listed here.

Public Properties

	Name	Description
R	DataSet (see page 7060)	Provides a reference to the dataset currently being designed.

39.1.18.5.1 TDataSetDesigner.DataSet Property

Provides a reference to the dataset currently being designed.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Use DataSet to access the dataset, if any, that is being designed. When the user specifies changes in the dataset designer, the changes can be made to the dataset specified by the DataSet property.

See Also

TDataSet (see page 6894)

39.1.19 TDataSetField Class

TDataSetField provides access to nested datasets.

Class Hierarchy

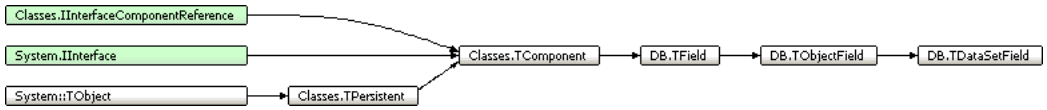

Pascal

```
TDataSetField = class(TObjectField);
```

C++

```
class TDataSetField : public TObjectField;
```

File

DB

Description

TDataSetField represents a nested dataset field in a database. This nested dataset can result from an object relational database (such as the nested datasets in Oracle 8 databases) or from a provider that packages data for the master table of a master/detail relationship.

TDataSetField introduces a new property NestedDataSet (see page 7070) for referencing nested datasets. It also introduces the Fields (see page 7384) property for accessing the fields in the nested dataset. When assigning data to a data set field, use the DataSetField property of a dataset component that is assigned as the value of the dataset field.

See Also

TDataSet (see page 6894)

Working with DataSet Fields

Working with Reference Fields

39.1.19.1 TDataSetField Members

The following tables list the members exposed by TDataSetField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7069)	This is Bind, a member of class TDataSetField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CheckTypeSize (see page 7380)	This is CheckTypeSize, a member of class TObjectField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 7380)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
	FreeBuffers (see page 7380)	This is FreeBuffers, a member of class TObjectField.
	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
	GetAsString (see page 7381)	This is GetAsString, a member of class TObjectField.
	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
	GetAsVariant (see page 7381)	This is GetAsVariant, a member of class TObjectField.
	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
	GetCanModify (see page 7069)	This is GetCanModify, a member of class TDataSetField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
	GetDefaultWidth (see page 7381)	This is GetDefaultWidth, a member of class TObjectField.
	GetFieldCount (see page 7381)	This is GetFieldCount, a member of class TObjectField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetFields (see page 7069)	This is GetFields, a member of class TDataSetField.
	GetFieldValue (see page 7382)	This is GetFieldValue, a member of class TObjectField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetHasConstraints (see page 7382)	This is GetHasConstraints, a member of class TObjectField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.

	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 7382)	This is SetChildOrder, a member of class TObjectField.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataSet (see page 7383)	This is SetDataSet, a member of class TObjectField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetFieldKind (see page 7383)	This is SetFieldKind, a member of class TObjectField.
	SetFieldValue (see page 7383)	This is SetFieldValue, a member of class TObjectField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetParentField (see page 7383)	This is SetParentField, a member of class TObjectField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetUnNamed (see page 7383)	This is SetUnNamed, a member of class TObjectField.
	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
	SetVarValue (see page 7384)	This is SetVarValue, a member of class TObjectField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 7068)	Copies the Value property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 7126)	Sets the value of the field to NULL.
V	Create (see page 7067)	Creates and initializes a TDataSetField (see page 7060) instance.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 7067)	Destroys an instance of TObjectField.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetChildren (see page 7381)	This is GetChildren, a member of class TObjectField.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.

	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldCount (see page 7384)	Indicates the number of child fields that comprise the object field.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	Fields (see page 7384)	Specifies the TFields (see page 7211) object that contains an array of child fields.
	FieldValues (see page 7385)	Provides access to the values for all child fields in the object field.

R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
R	NestedDataSet (see page 7070)	Indicates the dataset that is nested in the dataset field.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
R	UnNamed (see page 7386)	Indicates whether an object field in a nested table has a name associated with it.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7180)	Represents the data in a field component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	IncludeObjectField (see page 7069)	Indicates whether this field can be represented as a persistent field in the nested dataset. Reserved for future use.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.

	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectType (see page 7385)	Specifies the classification of the field object in Oracle.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.19.2 TDataSetField.Create Constructor

Creates and initializes a TDataSetField (see page 7060) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDataSetField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TDataSetField (see page 7060). Instead, the field components are created automatically, as persistent field components defined in the Fields (see page 7384) editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftDataSet. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TDataSetField (see page 7060). After instantiating a TDataSetField (see page 7060), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TDataSetField (see page 7060) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Specify the dataset that contains the dataset field by setting its DataSet (see page 7157) property.

39.1.19.3 TDataSetField.Destroy Destructor

Destroys an instance of TObjectField.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataSetField();
```

Description

Do not call Destroy directly in an application. The destructor is called automatically by the dataset that owns the object field component, and applications never need to call it explicitly.

If an application creates a descendant of TObjectField without an owner, it should call Free instead of Destroy. Free checks that the field reference is not nil, and only then calls Destroy.

Destroy closes the dataset, deletes the instance of TObjectField from the dataset, and calls the inherited destructor.

See Also

Free (see page 22300)

39.1.19.4 TDataSetField Methods

The methods of the TDataSetField class are listed here.

Protected Methods

	Name	Description
	Bind (see page 7069)	This is Bind, a member of class TDataSetField.
	GetCanModify (see page 7069)	This is GetCanModify, a member of class TDataSetField.
	GetFields (see page 7069)	This is GetFields, a member of class TDataSetField.

Public Methods

	Name	Description
	Assign (see page 7068)	Copies the Value property from another field component or assigns another object to the field component.

39.1.19.4.1 TDataSetField.Assign Method

Copies the Value property from another field component or assigns another object to the field component.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use Assign to copy the Value property from one field to another. The fields must have compatible data types.

When Source is another field object, Assign calls the AssignValue method, using the Source field's Value property. Calling Assign with the Source parameter set to nil (Delphi) or NULL (C++) assigns a blank value to the field component. Calling Assign with any other object type calls the inherited method, which allows assignment from objects that implement a TField object in their AssignTo method.

See Also

AssignValue (see page 7124)

Clear (see page 7126)

DataType (see page 7159)

AssignTo (see page 2931)

Value (see page 7180)

39.1.19.4.2 TDataSetField.Bind Method

Pascal

```
procedure Bind(Binding: Boolean); override;
```

C++

```
virtual __fastcall Bind(Boolean Binding);
```

Description

This is Bind, a member of class TDataSetField.

39.1.19.4.3 TDataSetField.GetCanModify Method

Pascal

```
function GetCanModify: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetCanModify();
```

Description

This is GetCanModify, a member of class TDataSetField.

39.1.19.4.4 TDataSetField.GetFields Method

Pascal

```
function GetFields: TFields; override;
```

C++

```
virtual __fastcall TFields GetFields();
```

Description

This is GetFields, a member of class TDataSetField.

39.1.19.5 TDataSetField Properties

The properties of the TDataSetField class are listed here.

Public Properties

	Name	Description
R	NestedDataSet (see page 7070)	Indicates the dataset that is nested in the dataset field.

Published Properties

	Name	Description
	IncludeObjectField (see page 7069)	Indicates whether this field can be represented as a persistent field in the nested dataset. Reserved for future use.

39.1.19.5.1 TDataSetField.IncludeObjectField Property

Indicates whether this field can be represented as a persistent field in the nested dataset. Reserved for future use.

Pascal

```
property IncludeObjectField: Boolean;
```

C++

```
__property Boolean IncludeObjectField;
```

Description

The IncludeObjectField property is not currently implemented.

In BDE-enabled datasets, set this property to true to allow the nested dataset that is the value of this field to access this parent dataset field. When this property is true, a persistent field can be created for this dataset field in the detail dataset using the Fields (see page 7384) editor. The name of this dataset field is then displayed above the child fields in a grid associated with the nested dataset.

The default value is false.

See Also

NestedDataSet (see page 7070)

ParentField (see page 7176)

DataSetField (see page 7013)

39.1.19.5.2 TDataSetField.NestedDataSet Property

Indicates the dataset that is nested in the dataset field.

Pascal

```
property NestedDataSet: TDataSet;
```

C++

```
__property TDataSet NestedDataSet;
```

Description

Read NestedDataSet to access the nested dataset of the dataset field.

See Also

DataSetField (see page 7013)

Fields

Working with DataSet Fields

39.1.20 TDataSource Class

TDataSource provides an interface between a dataset component and data-aware controls on a form.

Class Hierarchy

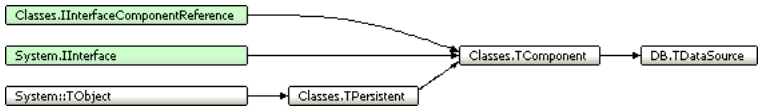

Pascal

```
TDataSource = class(TComponent);
```

C++

```
class TDataSource : public TComponent;
```

File

DB

Description

Use TDataSource to

provide a conduit between a dataset and data-aware controls on a form that enable display, navigation, and editing of the data underlying the dataset.

link two datasets in a master/detail relationship.

All datasets must be associated with a data source component if their data is to be displayed and manipulated in data-aware controls. Similarly, each data-aware control needs to be associated with a data source component in order for the control to receive and manipulate data.

Data source components also link datasets in master-detail relationships.

Delphi Examples:

```
{
This example requires two ClientDataSets, two DBGrids, two
DataSources and two DBNavigators.  Name the two
ClientDataSets CDS and CDS2. Set the DataSet property in the
first DataSource to CDS and name that DataSource DS.  Set the
DataSet property in the second DataSource to CDS2 and name
that DataSource DS2.  Set the DataSource in the first
DBNavigator to DS and the DataSource in the second DBNavigator
to DS2.  Set the DataSource in the first DBGrid to DS and the
DataSource in the second DBGrid to DS2.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
type to Integer.  The file used to load CDS must have an
attribute name of HOST with an attribute type of string and
an attribute name of PORTA with an attribute type of integer.
Add DB, DBCtrls, Grids, DBGrids, DBClient and SwSystem to the
uses clause.  "LoadFormFile" to start with the file, or
"CreateDataSet" to start from scratch.  You must "CloseDataSet"
after a "LoadFromFile" before you can do another "CreateDataSet".
}
{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button1Click(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.SaveToFile(GetCurrentDir + '\CDS.XML', dfXML);
{$ELSE}
  CDS.SaveToFile(gsAppPath + 'CDS.XML', dfXML);
{$ENDIF}
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  CDS.CreateDataSet;
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  CDS2.Data := CDS.Data;
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  CDS.Close;
end;
```

See Also

TDataSet (see page 6894)

Associating a Data Control with a Dataset

Using Data-aware ActiveX Controls

VCL Overview

39.1.20.1 TDataSource Members

The following tables list the members exposed by TDataSource.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.

	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	DataLinks (see page 7078)	This is DataLinks, a member of class TDataSource.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 7075)	Creates an instance of a TDataSource (see page 7070) component.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 7075)	Destroys the instance of a data source component.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.

V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 7076)	Puts the associated dataset into edit mode, if possible.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 7076)	Checks whether a data source is linked to a specified dataset.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.

R	State (see page 7080)	Reads the current state of the dataset component associated with a data source.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnDataChange (see page 7080)	Occurs when the data in a record has changed, either due to field edits or moving the cursor to a new record.
	OnStateChange (see page 7082)	Occurs when the state of a data source component's dataset changes.
	OnUpdateData (see page 7082)	Occurs when the data in the current record is about to be updated.

Published Properties

	Name	Description
	AutoEdit (see page 7077)	Determines if a data source component automatically calls a dataset's Edit (see page 7076) method when a data-aware control associated with the data source receives focus.
	DataSet (see page 7078)	Specifies the dataset for which the data source component serves as a conduit to data-aware controls or other datasets.
	Enabled (see page 7079)	Determines if the data-aware controls associated with the data source component display data.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

39.1.20.2 TDataSource.Create Constructor

Creates an instance of a TDataSource (see page 7070) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDataSource(TComponent * AOwner);
```

Description

Call Create to instantiate a TDataSource (see page 7070) component at runtime. Data source components added to a form or data module at design time are automatically created.

Create instantiates a data link list, and sets the Enabled (see page 7079) and AutoEdit (see page 7077) properties of the component to true.

See Also

AutoEdit (see page 7077)

Enabled (see page 7079)

Destroy (see page 7075)

39.1.20.3 TDataSource.Destroy Destructor

Destroys the instance of a data source component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataSource();
```

Description

Do not call Destroy directly. Instead, call Free, which checks that the data source reference is not nil before calling Destroy.

Destroy sets OnStateChange (see page 7082) to nil (Delphi) or NULL (C++), sets the dataset for the component to nil (Delphi) or NULL (C++), and frees the data link list before calling the inherited destructor for the component.

See Also

Create (see page 7075)

DataSet (see page 7078)

OnStateChange (see page 7082)

Create (see page 7075)

Free (see page 22300)

39.1.20.4 TDataSource Methods

The methods of the TDataSource class are listed here.

Public Methods

	Name	Description
	Edit (see page 7076)	Puts the associated dataset into edit mode, if possible.
	IsLinkedTo (see page 7076)	Checks whether a data source is linked to a specified dataset.

39.1.20.4.1 TDataSource.Edit Method

Puts the associated dataset into edit mode, if possible.

Pascal

```
procedure Edit;
```

C++

```
__fastcall Edit();
```

Description

Call Edit to enable a data source component to control a user's ability to edit a dataset associated with the data source. Edit checks that the AutoEdit (see page 7077) property of the data source is true and that its State (see page 7080) property is dsBrowse before calling the dataset's Edit method. Checking these properties before calling the dataset's Edit method ensures that the dataset is in a state that supports editing, and that the data source allows editing as well.

See Also

AutoEdit (see page 7077)

State (see page 7080)

Edit (see page 6932)

39.1.20.4.2 TDataSource.IsLinkedTo Method

Checks whether a data source is linked to a specified dataset.

Pascal

```
function IsLinkedTo(DataSet: TDataSet): Boolean;
```

C++

```
__fastcall Boolean IsLinkedTo(TDataSet DataSet);
```

Description

Call IsLinkedTo to verify that a data source is linked to a specific dataset. DataSet (see page 7078) is the name of the dataset to which the data source should be linked.

If the data source already uses the specified dataset (or its parent dataset if the specified dataset is nested in another dataset), IsLinkedTo returns true. If the data source does not use the specified dataset IsLinkedTo returns false.

Note: IsLinkedTo is a low-level method mainly of interest to component developers who need to detect circular data references. Otherwise IsLinkedTo should not be called directly.

See Also

DataSet (see page 7078)

IsLinkedTo (see page 6967)

39.1.20.5 TDataSource Properties

The properties of the TDataSource class are listed here.

Protected Properties

	Name	Description
R	DataLinks (see page 7078)	This is DataLinks, a member of class TDataSource.

Public Properties

	Name	Description
R	State (see page 7080)	Reads the current state of the dataset component associated with a data source.

Published Properties

	Name	Description
	AutoEdit (see page 7077)	Determines if a data source component automatically calls a dataset's Edit (see page 7076) method when a data-aware control associated with the data source receives focus.
	DataSet (see page 7078)	Specifies the dataset for which the data source component serves as a conduit to data-aware controls or other datasets.
	Enabled (see page 7079)	Determines if the data-aware controls associated with the data source component display data.

39.1.20.5.1 TDataSource.AutoEdit Property

Determines if a data source component automatically calls a dataset's Edit (see page 7076) method when a data-aware control associated with the data source receives focus.

Pascal

```
property AutoEdit: Boolean;
```

C++

```
__property Boolean AutoEdit;
```

Description

AutoEdit is true by default. If AutoEdit is true, then when a user attempts to modify the data displayed by the control the data source calls the underlying dataset's Edit (see page 7076) method.

Set AutoEdit to false to protect data from unintentional modification. Even if AutoEdit is false, an application can explicitly call a dataset's Edit (see page 7076) method to permit data modification.

See Also

DataSet (see page 7078)

Enabled (see page 7079)

OnDataChange (see page 7080)

OnStateChange (see page 7082)

Edit (see page 7076)

Edit (see page 6932)

Enabling Editing in Controls On User Entry

39.1.20.5.2 TDataSource.DataLinks Property

Pascal

```
property DataLinks: TList;
```

C++

```
__property TList DataLinks;
```

Description

This is DataLinks, a member of class TDataSource.

39.1.20.5.3 TDataSource.DataSet Property

Specifies the dataset for which the data source component serves as a conduit to data-aware controls or other datasets.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Set DataSet to the name of an existing dataset component either at design time, or at runtime. By changing the value of DataSet at runtime an application can effectively use the same data-aware controls to display and edit data in different datasets.

DataSource.DataSet := ClientDataSet1;

DataSource->DataSet = ClientDataSet1;

Note: To link a dataset that resides in a data module to a form at design-time, choose File | Use unit (Delphi) or File | Include unit header (C++).

Delphi Examples:

```
{
The following example copies the selected rows in a db grid
to a list box.  Set the db grid Options dgRowSelect,
dgAlwaysShowSelection and dgMultiSelect to True.  Make a
multiple selecton using the CNTL key.  This example requires
a TDataSet associated with a TDataSource and a TDBGrid.}
procedure TForm1.Button2Click(Sender: TObject);
var
  i, j: Integer;
  s: string;
begin
  if DBGrid2.SelectedRows.Count>0 then
    with DBGrid2.DataSource.DataSet do
      for i:=0 to DBGrid2.SelectedRows.Count-1 do
      begin
        GotoBookmark(pointer(DBGrid2.SelectedRows.Items[i]));
        for j := 0 to FieldCount-1 do
        begin
          if (j>0) then s:=s+', ';
          s:=s+Fields[j].AsString;
        end;
        Listbox1.Items.Add(s);
        s:= '';
      end;
end;
```

See Also

AutoEdit (see page 7077)

Enabled (see page 7079)

Associating a Data Control with a Dataset

39.1.20.5.4 TDataSource.Enabled Property

Determines if the data-aware controls associated with the data source component display data.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to control whether data-aware controls connected to a data source display data. If Enabled is true (the default), data are displayed. If Enabled is false, all controls associated with the data source are blank.

Note: While an application can set Enabled to false to blank out data-aware controls during processing of multiple records, a better choice is to use the TDataSet (see page 6894) EnableControls and DisableControls methods. Using these methods prevents the controls from blanking out during processing.

See Also

State (see page 7080)

DisableControls (see page 6925)

EnableControls (see page 6934)

Editing Data in a Control

Enabling and Disabling the Data Source

39.1.20.5.5 TDataSource.State Property

Reads the current state of the dataset component associated with a data source.

Pascal

```
property State: TDataSetState;
```

C++

```
__property TDataSetState State;
```

Description

Examine State to determine the current state of the dataset associated with a data source component. The state of a dataset indicates the action, if any, taking place against the dataset.

The possible values of State correspond to the values for a dataset's State property and are identical to them except that when Enabled (see page 7079) is false or DataSet (see page 7078) is unassigned, State is dsInactive regardless of the actual state of the dataset.

See Also

DataSet (see page 7078)

Enabled (see page 7079)

Edit (see page 7076)

State (see page 7036)

OnCalcFields (see page 7053)

OnFilterRecord (see page 7055)

SetRange (see page 10209)

39.1.20.6 TDataSource Events

The events of the TDataSource class are listed here.

Published Events

	Name	Description
	OnDataChange (see page 7080)	Occurs when the data in a record has changed, either due to field edits or moving the cursor to a new record.
	OnStateChange (see page 7082)	Occurs when the state of a data source component's dataset changes.
	OnUpdateData (see page 7082)	Occurs when the data in the current record is about to be updated.

39.1.20.6.1 TDataSource.OnDataChange Event

Occurs when the data in a record has changed, either due to field edits or moving the cursor to a new record.

Pascal

```
property OnDataChange: TDataChangeEvent;
```

C++

```
__property TDataChangeEvent OnDataChange;
```

Description

Write an OnDataChange event handler to take specific actions when a field in the current record has been edited and the

application moves to another field, or when the current record in the associated dataset changes. OnDataChange is especially useful in applications that must synchronize data display in controls that are not data-aware. This event is typically used to make sure the control reflects the current field values in the dataset, because it is triggered by all changes.

Methods that can trigger this event include the Next or Prior methods for the dataset. Data-aware controls notify a data source of a data change when:

Scrolling to a new record.

Modifications to a field's data.

Delphi Examples:

```
{
This example uses a button to copy the value of a field in
the previous record into the corresponding field in the
current record.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   SavePlace: TBookmark;
   PrevValue: Variant;
begin
   with Customers do
   begin
    { get a bookmark so that we can return to the same record }
    SavePlace := GetBookmark;
    try
      { move to prior record}
      FindPrior;
      { get the value }
      PrevValue := Fields[1].Value;
      {Move back to the bookmark
      this may not be the next record anymore
      if something else is changing the dataset asynchronously }
      GotoBookmark(SavePlace);
      { Set the value }
      Edit;
      Fields[1].Value := PrevValue;
      { Free the bookmark }
    finally
      FreeBookmark(SavePlace);
    end;
  end;
end;

{
To ensure that the button is disabled when there is no
previous record, the OnDataChange event of the DataSource
detects when the user moves to the beginning of file (BOF
property becomes true), and disables the button.  Detection
occurs on scrolling and editing, not selection with the mouse.
}
procedure TForm1.DS2DataChange(Sender: TObject; Field: TField);
begin
  if Customers.Bof then
    Button1.Enabled := False
  else
    Button1.Enabled := True;
end;
```

See Also

OnUpdateData (see page 7082)

OnStateChange (see page 7082)

Responding to Changes Mediated by the Data Source

Displaying

39.1.20.6.2 TDataSource.OnStateChange Event

Occurs when the state of a data source component's dataset changes.

Pascal

```
property OnStateChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnStateChange;
```

Description

Write an OnStateChange event handler to take specific actions when the State (see page 7080) property changes for a dataset associated with a data source component.

During the course of a normal connection to a database, a dataset's state changes frequently. For example, each time a user starts editing a field in a data-aware control the dataset's State (see page 7080) property is changed from dsBrowse to dsEdit if the State (see page 7080) is not already dsEdit. An OnStateChange event handler can respond to changes in state by taking actions such as disabling or enabling menu items or buttons.

See Also

OnUpdateData (see page 7082)

State (see page 7036)

Displaying

39.1.20.6.3 TDataSource.OnUpdateData Event

Occurs when the data in the current record is about to be updated.

Pascal

```
property OnUpdateData: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdateData;
```

Description

Write an OnUpdateData event handler to take specific actions prior to posting data changes to the database. If an application posts changes to the database and an OnUpdateData event handler exists, the handler is called before the dataset's Post method is executed. An OnUpdateData event handler might be useful for performing additional data processing or validation before posting.

See Also

OnDataChange (see page 7080)

OnStateChange (see page 7082)

BeforePost (see page 7052)

39.1.21 TDateField Class

TDateField represents a date field in a dataset.

Class Hierarchy

Pascal

```
TDateField = class(TDateTimeField);
```

C++

```
class TDateField : public TDateTimeField;
```

File

DB

Description

TDateField encapsulates the fundamental behavior common to Date fields. Date fields contain date values (without any time value), which they manage using the TDateTime type (Delphi) or class (C++).

If you use the Fields editor at design time to create a persistent field component for the date field, you can access it by name at runtime. When using dynamic field components, you can access the TDateField instance using the dataset's Fields property or FieldByName method.

See Also

TTimeField (see page 7499)

TDateTimeField (see page 7090)

TSQLTimeStampField (see page 7469)

TDataSet (see page 6894)

TDataSet (see page 6894)

TDataSet (see page 6894)

TField (see page 7115)

Using Default Formatting for Numeric

Controlling and Masking User Input

39.1.21.1 TDateField Members

The following tables list the members exposed by TDateField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7098)	This is CopyData, a member of class TDateTimeField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7098)	This is GetAsDateTime, a member of class TDateTimeField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsFloat (see page 7099)	This is GetAsFloat, a member of class TDateTimeField.
V	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
V	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsString (see page 7099)	This is GetAsString, a member of class TDateTimeField.
V	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
V	GetAsVariant (see page 7099)	This is GetAsVariant, a member of class TDateTimeField.
V	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
V	GetDataSize (see page 7090)	This is GetDataSize, a member of class TDateField.
V	GetDefaultWidth (see page 7100)	This is GetDefaultWidth, a member of class TDateTimeField.
V	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
V	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
V	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetSize (see page 7135)	This is GetSize, a member of class TField.
V	GetText (see page 7100)	This is GetText, a member of class TDateTimeField.

	GetText (see page 7135)	This is GetText, a member of class TField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7100)	This is SetAsDateTime, a member of class TDateTimeField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7100)	This is SetAsFloat, a member of class TDateTimeField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7100)	This is SetAsString, a member of class TDateTimeField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 7101)	This is SetVarValue, a member of class TDateTimeField.
	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.

	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7126)	Sets the value of the field to NULL.
	Create (see page 7089)	Creates an instance of TDateField (see page 7083).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.

	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7103)	Represents the value of the date in the date-time field.
	Value (see page 7180)	Represents the data in a field component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.

	DisplayFormat (see page 7101)	Controls (see page 5711) the formatting of the date-time field for display purposes.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	EditMask (see page 7102)	Contains a mask that restricts the data that can be entered into a data field.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.21.2 TDateField.Create Constructor

Creates an instance of TDateField (see page 7083).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDateField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TDateField (see page 7083). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftDate. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TDateField (see page 7083). After instantiating a TDateField (see page 7083), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TDateField (see page 7083) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TDateField (see page 7083) with a dataset component by setting its DataSet (see page 7157) property.

The example below creates a TDateField (see page 7083) object for a field named LastContact accessed through a TClientDataSet named ClientDataSet1.

```
var
  T: TDateField;
begin
  ClientDataSet1.Close;
  T := TDateField.Create(ClientDataSet1);
  T.FieldName := 'LastContact';
  T.Name := ClientDataSet1.Name + T.FieldName;
  T.Index := ClientDataSet1.FieldCount;
  T.DataSet := ClientDataSet1;
  T.DisplayFormat := 'mm/dd/yyyy';
  ClientDataSet1.FieldDefs.UpDate;
  ClientDataSet1.Open;
end;
ClientDataSet1->Close();
TDateField *T = new TDateField(ClientDataSet1);
T->FieldName = "LastContact";
T->Name = ClientDataSet1->Name + T->FieldName;
T->Index = ClientDataSet1->FieldCount;
T->DataSet = ClientDataSet1;
T->DisplayFormat = "mm//dd//yyyy";
ClientDataSet1->FieldDefs->UpDate();
ClientDataSet1->Open();
```

See Also

Destroy (see page 7122)

DataType (see page 7159)

Create (see page 7121)

Owner (see page 2807)

39.1.21.3 TDateField Methods

The methods of the TDateField class are listed here.

Protected Methods

	Name	Description
	GetDataSize (see page 7090)	This is GetDataSize, a member of class TDateField.

39.1.21.3.1 TDateField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; override;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TDateField.

39.1.22 TDateTimeField Class

TDateTimeField represents a date-time field in a dataset.

Class Hierarchy

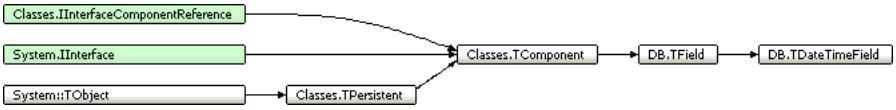

Pascal

```
TDateTimeField = class(TField);
```

C++

```
class TDateTimeField : public TField;
```

File

DB

Description

TDateTimeField encapsulates the fundamental behavior common to Date-time fields. Date-time fields contain combined date and time information, sometimes called timestamps. TDateTimeField introduces new properties to convert between date-time values and other data types, and to format the value of a date-time field for display.

Note: Date-time values accessed through dbExpress use TSQLTimeStampField (see page 7469) instead of TDateTimeField

If you use the Fields editor at design time to create a persistent field component for the date-time field, you can access it by name at runtime. When using dynamic field components, you can access the TDateTimeField instance using the dataset's Fields property or FieldByName method.

See Also

TTimeField (see page 7499)

TDateField (see page 7083)

TSQLTimeStampField (see page 7469)

TDataSet (see page 6894)

TDataSet (see page 6894)

TDataSet (see page 6894)

TField (see page 7115)

Using Default Formatting for Numeric

Converting Field Values

Controlling and Masking User Input

39.1.22.1 TDateTimeField Members

The following tables list the members exposed by TDateTimeField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7098)	This is CopyData, a member of class TDateTimeField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7098)	This is GetAsDateTime, a member of class TDateTimeField.
V	GetAsFloat (see page 7099)	This is GetAsFloat, a member of class TDateTimeField.
V	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsString (see page 7099)	This is GetAsString, a member of class TDateTimeField.
V	GetAsVariant (see page 7099)	This is GetAsVariant, a member of class TDateTimeField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
V	GetDataSize (see page 7099)	This is GetDataSize, a member of class TDateTimeField.
V	GetDefaultWidth (see page 7100)	This is GetDefaultWidth, a member of class TDateTimeField.
V	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
V	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetSize (see page 7135)	This is GetSize, a member of class TField.
V	GetText (see page 7100)	This is GetText, a member of class TDateTimeField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.

	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7100)	This is SetAsDateTime, a member of class TDateTimeField.
	SetAsFloat (see page 7100)	This is SetAsFloat, a member of class TDateTimeField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7100)	This is SetAsString, a member of class TDateTimeField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 7101)	This is SetVarValue, a member of class TDateTimeField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.

	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 7126)	Sets the value of the field to NULL.
V	Create (see page 7097)	Creates and initializes a TDateTimeField (see page 7090) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.

R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7103)	Represents the value of the date in the date-time field.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayFormat (see page 7101)	Controls (see page 5711) the formatting of the date-time field for display purposes.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	EditMask (see page 7102)	Contains a mask that restricts the data that can be entered into a data field.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.

	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.22.2 TDateTimeField.Create Constructor

Creates and initializes a TDateTimeField (see page 7090) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDateTimeField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TDateTimeField (see page 7090). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftDateTime. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TDateTimeField (see page 7090). After instantiating a TDateTimeField (see page 7090), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TDateTimeField (see page 7090) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TDateTimeField (see page 7090) with a dataset component by setting its DataSet (see page 7157) property to the name of the dataset component. Optionally set the DisplayFormat (see page 7101) property to the desired date and time format.

The example below creates a TDateTimeField (see page 7090) object for a field named CheckOut accessed through a TClientDataSet named ClientDataSet1.

```
var
  T: TDateTimeField;
begin
  ClientDataSet1.Close;
  T := TDateTimeField.Create(ClientDataSet1);
  T.FieldName := 'CheckOut';
  T.Name := ClientDataSet1.Name + T.FieldName;
  T.Index := ClientDataSet1.FieldCount;
  T.DataSet := ClientDataSet1;
  T.DisplayFormat := 'mm/dd/yyyy hh:nn:ss ampm';
  ClientDataSet1.FieldDefs.UpDate;
  ClientDataSet1.Open;
end;
ClientDataSet1->Close();
TDateTimeField *T = new TDateTimeField(ClientDataSet1);
```

```
T->FieldName = "CheckOut";
T->Name = ClientDataSet1->Name + T->FieldName;
T->Index = ClientDataSet1->FieldCount;
T->DataSet = ClientDataSet1;
T->DisplayFormat = "mm//dd//yyyy hh:nn:ss ampm";
ClientDataSet1->FieldDefs->UpDate();
ClientDataSet1->Open();
```

See Also

DataType (see page 7159)

Create (see page 7121)

Owner (see page 2807)

39.1.22.3 TDateTimeField Methods

The methods of the TDateTimeField class are listed here.

Protected Methods

	Name	Description
	CopyData (see page 7098)	This is CopyData, a member of class TDateTimeField.
	GetAsDateTime (see page 7098)	This is GetAsDateTime, a member of class TDateTimeField.
	GetAsFloat (see page 7099)	This is GetAsFloat, a member of class TDateTimeField.
	GetAsString (see page 7099)	This is GetAsString, a member of class TDateTimeField.
	GetAsVariant (see page 7099)	This is GetAsVariant, a member of class TDateTimeField.
	GetDataSize (see page 7099)	This is GetDataSize, a member of class TDateTimeField.
	GetDefaultWidth (see page 7100)	This is GetDefaultWidth, a member of class TDateTimeField.
	GetText (see page 7100)	This is GetText, a member of class TDateTimeField.
	SetAsDateTime (see page 7100)	This is SetAsDateTime, a member of class TDateTimeField.
	SetAsFloat (see page 7100)	This is SetAsFloat, a member of class TDateTimeField.
	SetAsString (see page 7100)	This is SetAsString, a member of class TDateTimeField.
	SetVarValue (see page 7101)	This is SetVarValue, a member of class TDateTimeField.

39.1.22.3.1 TDateTimeField.CopyData Method

Pascal

```
procedure CopyData(Source: Pointer; Dest: Pointer); override;
```

C++

```
virtual __fastcall CopyData(void * Source, void * Dest);
```

Description

This is CopyData, a member of class TDateTimeField.

39.1.22.3.2 TDateTimeField.GetAsDateTime Method

Pascal

```
function GetAsDateTime: TDateTime; override;
```

C++

```
virtual __fastcall TDateTime GetAsDateTime();
```

Description

This is GetAsDateTime, a member of class TDateTimeField.

39.1.22.3.3 TDateTimeField.GetAsFloat Method

Pascal

```
function GetAsFloat: Double; override;
```

C++

```
virtual __fastcall Double GetAsFloat();
```

Description

This is GetAsFloat, a member of class TDateTimeField.

39.1.22.3.4 TDateTimeField.GetAsString Method

Pascal

```
function GetAsString: string; override;
```

C++

```
virtual __fastcall AnsiString GetAsString();
```

Description

This is GetAsString, a member of class TDateTimeField.

39.1.22.3.5 TDateTimeField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TDateTimeField.

39.1.22.3.6 TDateTimeField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; override;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TDateTimeField.

39.1.22.3.7 TDateTimeField.GetDefaultWidth Method

Pascal

```
function GetDefaultWidth: Integer; override;
```

C++

```
virtual __fastcall int GetDefaultWidth();
```

Description

This is GetDefaultWidth, a member of class TDateTimeField.

39.1.22.3.8 TDateTimeField.GetText Method

Pascal

```
procedure GetText(var Text: string; DisplayText: Boolean); override;
```

C++

```
virtual __fastcall GetText(AnsiString Text, Boolean DisplayText);
```

Description

This is GetText, a member of class TDateTimeField.

39.1.22.3.9 TDateTimeField.SetAsDateTime Method

Pascal

```
procedure SetAsDateTime(Value: TDateTime); override;
```

C++

```
virtual __fastcall SetAsDateTime(TDateTime Value);
```

Description

This is SetAsDateTime, a member of class TDateTimeField.

39.1.22.3.10 TDateTimeField.SetAsFloat Method

Pascal

```
procedure SetAsFloat(Value: Double); override;
```

C++

```
virtual __fastcall SetAsFloat(Double Value);
```

Description

This is SetAsFloat, a member of class TDateTimeField.

39.1.22.3.11 TDateTimeField.SetAsString Method

Pascal

```
procedure SetAsString(const Value: string); override;
```

C++

```
virtual __fastcall SetAsString(const AnsiString Value);
```

Description

This is SetAsString, a member of class TDateTimeField.

39.1.22.3.12 TDateTimeField.SetVarValue Method

Pascal

```
procedure SetVarValue(const Value: Variant); override;
```

C++

```
virtual __fastcall SetVarValue(const Variant Value);
```

Description

This is SetVarValue, a member of class TDateTimeField.

39.1.22.4 TDateTimeField Properties

The properties of the TDateTimeField class are listed here.

Public Properties

	Name	Description
	Value (see page 7103)	Represents the value of the date in the date-time field.

Published Properties

	Name	Description
	DisplayFormat (see page 7101)	Controls (see page 5711) the formatting of the date-time field for display purposes.
	EditMask (see page 7102)	Contains a mask that restricts the data that can be entered into a data field.

39.1.22.4.1 TDateTimeField.DisplayFormat Property

Controls (see page 5711) the formatting of the date-time field for display purposes.

Pascal

```
property DisplayFormat: string;
```

C++

```
__property AnsiString DisplayFormat;
```

Description

Use DisplayFormat to specify the formatting string to use for formatting the value in a date-time field when the field's value appears as a string. If DisplayFormat is not assigned a string, the value is formatted according to the default date-time format specified by the Windows Control Panel.

Note: DisplayFormat does not affect the value of the AsString (see page 7151) property.

Construct a DisplayFormat string using these format specifiers:

Specifier	Displays
c	The date using the format given by the ShortDateFormat global variable, followed by the time using the format given by the LongTimeFormat global variable. The time is not displayed if the fractional part of the DateTime value is zero.
d	The day as a number without a leading zero (1-31).
dd	The day as a number with a leading zero (01-31).

ddd	The day as an abbreviation (Sun-Sat) using the strings given by the ShortDayNames global variable.
dddd	The day as a full name (Sunday-Saturday) using the strings given by the LongDayNames global variable.
ddddd	The date using the format given by the ShortDateFormat global variable.
dddddd	The date using the format given by the LongDateFormat global variable.
m	The month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mm	The month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mmm	The month as an abbreviation (Jan-Dec) using the strings given by the ShortMonthNames global variable.
mmmm	The month as a full name (January-December) using the strings given by the LongMonthNames global variable.
yy	The year as a two-digit number (00-99).
yyyy	The year as a four-digit number (0000-9999).
h	The hour without a leading zero (0-23).
hh	The hour with a leading zero (00-23).
n	The minute without a leading zero (0-59).
nn	The minute with a leading zero (00-59).
s	The second without a leading zero (0-59).
ss	The second with a leading zero (00-59).
t	The time using the format given by the ShortTimeFormat global variable.
tt	The time using the format given by the LongTimeFormat global variable.
am/pm	The time using the 12-hour clock for the preceding h or hh specifier, followed by "am" for any hour before noon, or "pm" for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	The time using the 12-hour clock for the preceding h or hh specifier, followed by "a" for any hour before noon, or "p" for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm	The time using the 12-hour clock for the preceding h or hh specifier, followed by the contents of the TimeAMString global variable for any hour before noon, or the contents of the TimePMString global variable for any hour after noon.
/	The date separator character given by the DateSeparator global variable.
:	The time separator character given by the TimeSeparator global variable.
'xx'/"xx"	Characters enclosed in single or double quotes are displayed as-is, with no formatting changes.

Format specifiers may be written in uppercase or lowercase letters; both produce the same result.

If the string given by the Format parameter is empty, the date and time value is formatted as if a c format specifier had been given.

See Also

AsString

39.1.22.4.2 TDateTimeField.EditMask Property

Contains a mask that restricts the data that can be entered into a data field.

Pascal

```
property EditMask: TEditMask;
```

C++

```
__property TEditMask EditMask;
```

Description

Use EditMask to restrict the characters a user can enter. If the user attempts to enter a character that is not valid, the character is not accepted. Validation using the EditMask property is performed on a character-by-character basis. Use the OnValidate event instead to validate the entire input, just before it is posted.

Setting EditMask to an empty string removes the mask.

Note: For a description of the format required by EditMask, see the

TEditMask type.

See Also

DisplayText (see page 7162)

IsValidChar (see page 7136)

OnValidate (see page 7183)

Validate (see page 7144)

Value (see page 7180)

39.1.22.4.3 TDateTimeField.Value Property

Represents the value of the date in the date-time field.

Pascal

```
property Value: TDateTime;
```

C++

```
__property TDateTime Value;
```

Description

Use Value to read data directly from and write data directly to a date-time field component at runtime. TDateTimeField (see page 7090) components store and manipulate their data as TDateTime values. Thus, for date-time fields, the Value property is the same as the AsDateTime (see page 7148) property.

```
var
  DT: TDateTime;
begin
  DT := Table1DateField.Value;
  ShowMessage(DateTimeToStr(DT));
end;
```

See Also

AsDateTime

39.1.23 TDefCollection Class

TDefCollection holds the definition objects (TNamedItem (see page 7357)) that represent the specifications for database entities such as fields or indexes.

Class Hierarchy

Pascal

```
TDefCollection = class(TOwnedCollection);
```

C++

```
class TDefCollection : public TOwnedCollection;
```

File

DB

Description

TDefCollection is used as a base class for collections of definition objects such as index or field definitions. TTable objects and TClientDataSet objects use these definition objects to specify the attributes of a database entity before creating the underlying table or dataset.

Use the properties and methods of TDefCollection to:

Access a specific definition object.

Add or delete definition objects from the list.

Find (see page 7107) out how many definition objects are defined.

Copy a set of definitions to another dataset.

See Also

TFieldDefs (see page 7198)

TNamedItem (see page 7357)

TIndexDefs (see page 7292)

TTable (see page 10168)

TClientDataSet (see page 8289)

39.1.23.1 TDefCollection Members

The following tables list the members exposed by TDefCollection.

Protected Events

	Name	Description
	OnUpdate (see page 7111)	This is OnUpdate, a member of class TDefCollection.

Protected Methods

	Name	Description
	Added (see page 2736)	Responds when items are added to the collection.

V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	DoUpdate (see page 7107)	This is DoUpdate, a member of class TDefCollection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2915)	Returns the owner of an object.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 7109)	Initializes the name of a newly inserted collection item.
V	Update (see page 7109)	Updates the collection to reflect changes to its items.
	UpdateDefs (see page 7109)	This is UpdateDefs, a member of class TDefCollection.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 2736)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 7106)	Instantiates a TDefCollection (see page 7104) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Find (see page 7107)	Locates a definition object in the Items array from its name.
	FindItemID (see page 2743)	Returns the item with the specified ID.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItemNames (see page 7108)	This is the overview for the GetItemNames method overload.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	IndexOf (see page 7108)	Finds the index number of a definition in the Items array from its name.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	DataSet (see page 7110)	Indicates the dataset that uses the TDefCollection (see page 7104) object to manage its definition objects.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 2751)	Lists the items in the collection.
	Updated (see page 7110)	Indicates whether the definition objects in the Items array match the corresponding instantiated objects in the dataset.

39.1.23.2 TDefCollection.Create Constructor

Instantiates a TDefCollection (see page 7104) object.

Pascal

```
constructor Create(ADataSet: TDataSet; AOwner: TPersistent; AClass: TCollectionItemClass);
```

C++

```
__fastcall TDefCollection(TDataSet ADataSet, TPersistent * AOwner, TCollectionItemClass
AClass);
```

Description

Do not call the Create method. TDefCollection (see page 7104) objects are created by the dataset that uses them to manage definition objects. The ADataset parameter specifies the dataset that creates the TDefCollection (see page 7104) object. It becomes the value of the Dataset property. The AOwner parameter indicates the object on whose behalf the definition is created. The AClass parameter specifies a descendant class of TNamedItem (see page 7357), and indicates the type of definition objects in the Items property array.

See Also

DataSet (see page 7110)

Items (see page 2751)

39.1.23.3 TDefCollection Methods

The methods of the TDefCollection class are listed here.

Protected Methods

	Name	Description
	DoUpdate (see page 7107)	This is DoUpdate, a member of class TDefCollection.
V	SetItemName (see page 7109)	Initializes the name of a newly inserted collection item.
V	Update (see page 7109)	Updates the collection to reflect changes to its items.
	UpdateDefs (see page 7109)	This is UpdateDefs, a member of class TDefCollection.

Public Methods

	Name	Description
	Find (see page 7107)	Locates a definition object in the Items array from its name.
	GetItemNames (see page 7108)	This is the overview for the GetItemNames method overload.
	IndexOf (see page 7108)	Finds the index number of a definition in the Items array from its name.

39.1.23.3.1 TDefCollection.DoUpdate Method

Pascal

```
procedure DoUpdate(Sender: TObject);
```

C++

```
__fastcall DoUpdate(TObject * Sender);
```

Description

This is DoUpdate, a member of class TDefCollection.

39.1.23.3.2 TDefCollection.Find Method

Locates a definition object in the Items array from its name.

Pascal

```
function Find(const AName: WideString): TNamedItem;
```

C++

```
__fastcall TNamedItem Find(const BSTR AName);
```

Description

Call Find to obtain information about a particular definition object. Specify the name of the definition object as the value of the Name parameter.

See Also

IndexOf (see page 7108)

Items (see page 2751)

Name (see page 7359)

39.1.23.3.3 GetItemNames Method

This is the overview for the GetItemNames method overload.

Overload List

	Name	Description
	TDefCollection.GetItemNames (TStrings) (see page 7108)	Fills a TStrings object with the names of all definitions in the list.
	TDefCollection.GetItemNames (TWideStrings) (see page 7108)	Fills a TStrings object with the names of all definitions in the list.

39.1.23.3.3.1 TDefCollection.GetItemNames Method (TStrings)

Fills a TStrings object with the names of all definitions in the list.

Pascal

```
procedure GetItemNames(List: TWideStrings); overload;
procedure GetItemNames(List: TStrings); overload;
```

C++

```
__fastcall GetItemNames(TWideStrings List);
__fastcall GetItemNames(TStrings List);
```

Description

Call GetItemNames to retrieve the names of all definitions managed by the TDefCollection (see page 7104) object. Pass an empty string list as the List parameter. GetItemNames fills the string list with the names of all definitions.

See Also

Name (see page 7359)

39.1.23.3.4 TDefCollection.IndexOf Method

Finds the index number of a definition in the Items array from its name.

Pascal

```
function IndexOf(const AName: WideString): Integer;
```

C++

```
__fastcall int IndexOf(const BSTR AName);
```

Description

Use the IndexOf method to locate the position of a definition in the Items property array. IndexOf returns a value from zero (the first position) to Count - 1. Specify the name of the definition as the Name parameter.

If Items contains no definition with the specified Name, the return value is -1.

See Also

Count (see page 2749)

Find (see page 7107)

Items (see page 2751)

Name (see page 7359)

39.1.23.3.5 TDefCollection.SetItemName Method

Initializes the name of a newly inserted collection item.

Pascal

```
procedure SetItemName(AItem: TCollectionItem); override;
```

C++

```
virtual __fastcall SetItemName(TCollectionItem AItem);
```

Description

The Insert method calls SetItemName to initialize the Name property of items when it inserts them into the collection.

As implemented in TCollection, SetItemName does nothing. Some TCollection descendants override this method to provide collection items with default names.

See Also

Insert (see page 2746)

Name (see page 2805)

39.1.23.3.6 TDefCollection.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(AItem: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem AItem);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed (see page 2740)

Changed (see page 2758)

39.1.23.3.7 TDefCollection.UpdateDefs Method

Pascal

```
procedure UpdateDefs(AMethod: TDefUpdateMethod);
```

C++

```
__fastcall UpdateDefs(TDefUpdateMethod AMethod);
```

Description

This is UpdateDefs, a member of class TDefCollection.

39.1.23.4 TDefCollection Properties

The properties of the TDefCollection class are listed here.

Public Properties

	Name	Description
R	DataSet (see page 7110)	Indicates the dataset that uses the TDefCollection (see page 7104) object to manage its definition objects.
	Updated (see page 7110)	Indicates whether the definition objects in the Items array match the corresponding instantiated objects in the dataset.

39.1.23.4.1 TDefCollection.DataSet Property

Indicates the dataset that uses the TDefCollection (see page 7104) object to manage its definition objects.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Use DataSet to access the dataset that uses the definitions in the Items array.

See Also

FieldDefs (see page 7019)

IndexDefs (see page 8480)

Items (see page 2751)

39.1.23.4.2 TDefCollection.Updated Property

Indicates whether the definition objects in the Items array match the corresponding instantiated objects in the dataset.

Pascal

```
property Updated: Boolean;
```

C++

```
__property Boolean Updated;
```

Description

Use Updated to determine whether the definitions managed by the TDefCollection (see page 7104) object match the corresponding entities (fields or indexes) of the dataset. Updated is set to true when the definitions are retrieved from the dataset at design time, or when the definitions are used to create the corresponding entities in the dataset. Updated is set to false when the properties of the definition objects are changed.

See Also

DataSet (see page 7110)

Items (see page 2751)

Update (see page 7205)

Update (see page 7299)

39.1.23.5 TDefCollection Events

The events of the TDefCollection class are listed here.

Protected Events

	Name	Description
	OnUpdate (see page 7111)	This is OnUpdate, a member of class TDefCollection.

39.1.23.5.1 TDefCollection.OnUpdate Event

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

This is OnUpdate, a member of class TDefCollection.

39.1.24 TDetailDataLink Class

TDetailDataLink is the ancestor to datalink classes that manage the relationships between datasets that are linked in master/detail relationships.

Class Hierarchy

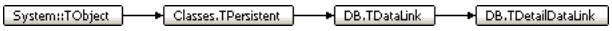

Pascal

```
TDetailDataLink = class(TDataLink);
```

C++

```
class TDetailDataLink : public TDataLink;
```

File

DB

Description

TDetailDataLink descendants are used to create master/detail relationships. They underlie the DataSource (see page 6891) property of dataset components.

TDetailDataLink has two descendants,

TMasterDataLink (see page 7339), which is used by BDE-enabled datasets, ADO-based datasets, and client datasets.

TSQLDataLink, which is used by unidirectional datasets.

See Also

TMasterDataLink (see page 7339)

TSQLDataLink (see page 20260)

TDataSet (see page 6894)

39.1.24.1 TDetailDataLink Members

The following tables list the members exposed by TDetailDataLink.

Protected Methods

	Name	Description
	ActiveChanged (see page 6880)	Responds to changes in the Active (see page 6888) property.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CheckBrowseMode (see page 6880)	Allows the data-aware object to respond just before an event that changes the dataset state.
	DataEvent (see page 6881)	Responds to various events that occur while working with data.
	DataSetChanged (see page 6881)	Responds to changes in the dataset.
	DataSetScrolled (see page 6882)	Allows a response to scrolling the representation of the dataset within the data-aware object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EditingChanged (see page 6883)	Responds to changes in the editing state of the data source.
	FocusControl (see page 6884)	Provides the interface for a method to force focus to the data-aware component associated with the TDataLink (see page 6876).
	GetActiveRecord (see page 6884)	This is GetActiveRecord, a member of class TDataLink.
	GetBOF (see page 6884)	This is GetBOF, a member of class TDataLink.
	GetBufferCount (see page 6884)	This is GetBufferCount, a member of class TDataLink.
	GetDetailDataSet (see page 7114)	Returns the value of the DetailDataSet (see page 7114) property.
	GetEOF (see page 6885)	This is GetEOF, a member of class TDataLink.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRecordCount (see page 6885)	This is GetRecordCount, a member of class TDataLink.
	LayoutChanged (see page 6885)	Responds to changes in the representation of the data by the data-aware object.
	MoveBy (see page 6885)	Repositions the dataset to another record relative to its active record.
	RecordChanged (see page 6886)	Responds to changes in the contents of the current record or field of the dataset.
	SetActiveRecord (see page 6886)	This is SetActiveRecord, a member of class TDataLink.
	SetBufferCount (see page 6887)	This is SetBufferCount, a member of class TDataLink.
	UpdateData (see page 6887)	Provides an interface for writing edits to a record in the dataset.

Protected Properties

	Name	Description
	VisualControl (see page 6894)	Indicates whether a visual control uses the datalink.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6878)	Creates an instance of TDataLink (see page 6876).

	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 6879)	Destroys an instance of TDataLink (see page 6876).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6882)	Attempts to drive the data source into an editing state if it is not already in one.
	ExecuteAction (see page 6883)	Invokes an action with the datasource as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	UpdateAction (see page 6887)	Updates an action to reflect the current state of the TDataLink (see page 6876).
	UpdateRecord (see page 6888)	Responds to notifications that edits should be posted to the dataset.

Public Properties

	Name	Description
R	Active (see page 6888)	Specifies whether the dataset that TDataLink (see page 6876) manages is active.
	ActiveRecord (see page 6889)	Specifies the index of the current record within the internal record buffer maintained by the dataset for the Owner of the TDataLink (see page 6876) object.
R	BOF (see page 6889)	Indicates whether the first record in a dataset is active.
	BufferCount (see page 6890)	Specifies the number of records cached by the dataset for the Owner of the TDataLink (see page 6876) object.
R	DataSet (see page 6890)	Specifies the TDataSet (see page 6894) object that represents the data that this TDataLink (see page 6876) is tracking.
	DataSource (see page 6891)	Specifies the TDataSource (see page 7070) object that the Owner of the TDataLink (see page 6876) uses to connect to the dataset.
	DataSourceFixed (see page 6891)	Indicates whether the DataSource (see page 6891) property can be set.
R	DetailDataSet (see page 7114)	Indicates the detail dataset in the master/detail relationship.
R	Editing (see page 6892)	Indicates whether the DataSource (see page 6891) is in edit mode.
R	Eof (see page 6892)	Indicates whether the last record in a dataset is active.
	ReadOnly (see page 6893)	Indicates whether the dataset for this TDataLink (see page 6876) can be modified.
R	RecordCount (see page 6893)	Indicates the number of records in the internal record buffer maintained by the dataset.

39.1.24.2 TDetailDataLink Methods

The methods of the TDetailDataLink class are listed here.

Protected Methods

	Name	Description
	GetDetailDataSet (see page 7114)	Returns the value of the DetailDataSet (see page 7114) property.

39.1.24.2.1 TDetailDataLink.GetDetailDataSet Method

Returns the value of the DetailDataSet (see page 7114) property.

Pascal

```
function GetDetailDataSet: TDataSet; virtual;
```

C++

```
virtual __fastcall TDataSet GetDetailDataSet();
```

Description

GetDetailDataSet is the protected read implementation of the DetailDataSet (see page 7114) property. As implemented in TDetailDataLink (see page 7111), GetDetailDataSet always returns nil (Delphi) or NULL (C++). Descendants override this method to return the dataset that uses the data link to link to a master dataset.

See Also

DetailDataSet (see page 7114)

39.1.24.3 TDetailDataLink Properties

The properties of the TDetailDataLink class are listed here.

Public Properties

	Name	Description
	DetailDataSet (see page 7114)	Indicates the detail dataset in the master/detail relationship.

39.1.24.3.1 TDetailDataLink.DetailDataSet Property

Indicates the detail dataset in the master/detail relationship.

Pascal

```
property DetailDataSet: TDataSet;
```

C++

```
__property TDataSet DetailDataSet;
```

Description

Read DetailDataSet to access the detail dataset. This is the dataset that uses a DetailDataLink to access its master dataset. The detail dataset is the detail datalink's owner, and uses the datalink to implement its DataSource (see page 6891) property.

Note: Do not confuse the DetailDataSet property with the DataSet (see page 6890) property. The DataSet (see page 6890) property is the dataset whose data events the detail datalink tracks. That is, DataSet (see page 6890) is the master dataset while DetailDataSet is the detail set.

See Also

DataSet (see page 6890)

DataSource (see page 7014)

39.1.25 TField Class

TField is the common ancestor of all the field components.

Class Hierarchy

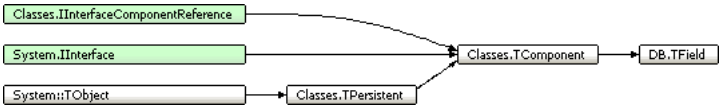

Pascal

```
TField = class(TComponent);
```

C++

```
class TField : public TComponent;
```

File

DB

Description

TField encapsulates the fundamental behavior common to all field components. It introduces the properties, events, and methods that are used to:

Change (see page 7126) the value of a field in a dataset.

Convert the value of a field from one data type to another.

Validate (see page 7144) data that the user enters for a field.

Define how the data in the field appears as it is displayed or edited.

Calculate the value of a field from code written in the OnCalcFields event of the dataset.

Look up the field's value from another dataset.

Do not create instances of TField. TField descendants are created automatically each time a dataset is activated. These descendants can be dynamic (the default) or persistent. Dynamic field components reflect the columns in the underlying metadata at the time a dataset is opened. Persistent field components are created at design time using the Fields editor, which specifies the fields in the dataset, their properties, and their ordering.

Creating persistent field components guarantees that each time an application runs, it uses and displays the same columns, in the same order, even if the physical structure of the underlying database changes. If a column on which a persistent field component is based is deleted or changed, the IDE generates an exception rather than opening the dataset against a nonexistent column or mismatched data. If this happens, remove the field component for the nonexistent field using the Fields editor.

A field in a dataset is always treated as one of the following TField descendant classes below:

TADTField (see page 6736)	TDateField (see page 7083)	TReferenceField (see page 7452)	TAggregateField (see page 6743)

TDateTimeField (see page 7090)	TSmallIntField	TArrayField (see page 6760)	TFloatField (see page 7229)
TSQLTimeStampField (see page 7469)	TAutoIncField (see page 6768)	TFMTBCDField (see page 7242)	TStringField (see page 7483)
TBCDField (see page 6776)	TGraphicField (see page 7257)	TTimeField (see page 7499)	TBinaryField (see page 6791)
TGuidField (see page 7265)	TVarBytesField (see page 7507)	TBlobField (see page 6801)	TIDispatchField (see page 7274)
TVariantField (see page 7514)	TBooleanField (see page 6822)	TIntegerField (see page 7301)	TWideStringField (see page 7549)
TBytesField (see page 6833)	TInterfaceField (see page 7314)	TWordField (see page 7560)	TCurrencyField (see page 6850)
TLargeintField (see page 7323)	TDataSetField (see page 7060)	TMemoField (see page 7348)	

See Also

TDataSet (see page 6894)

TFieldDef (see page 7184)

TFieldDefs (see page 7198)

Understanding Datasets: Overview

Working with Array Fields

Working with Field Component Methods at Runtime

Dynamic Field Components

Working with Field Components: Overview

39.1.25.1 TField Members

The following tables list the members exposed by TField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.

	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.

	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
V	SetSize (see page 7143)	This is SetSize, a member of class TField.
V	SetText (see page 7144)	This is SetText, a member of class TField.
V	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
V	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 7126)	Sets the value of the field to NULL.
V	Create (see page 7121)	Creates and initializes a field component.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.

	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7180)	Represents the data in a field component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.25.2 TField.Create Constructor

Creates and initializes a field component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TField(TComponent * AOwner);
```

Description

Do not create TField (see page 7115) instances. TField (see page 7115) is intended only as a base class, and should not be

directly instantiated. The constructor for descendant classes is usually called by the dataset when it is opened.

After calling the inherited constructor, TField (see page 7115) initializes the Visible (see page 7181) property to true.

See Also

Visible (see page 7181)

Destroy (see page 7122)

39.1.25.3 TField.Destroy Destructor

Destroys an instance of TField (see page 7115).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TField();
```

Description

Do not call Destroy directly in an application. Destroy is called automatically by the dataset that owns the field component, and applications never need to call it explicitly. If an application creates a descendant of TField (see page 7115) without an owner, it should call Free instead of Destroy, to guard against nil values.

Destroy closes the dataset, deletes the instance of TField (see page 7115) from the dataset, and calls the inherited destructor.

See Also

DataSet (see page 7157)

Free (see page 22300)

Destroy (see page 2766)

Create (see page 7121)

39.1.25.4 TField Methods

The methods of the TField class are listed here.

Protected Methods

	Name	Description
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.

	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.

Public Methods

	Name	Description
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	Clear (see page 7126)	Sets the value of the field to NULL.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.

	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	SetData (see page 7141)	Assigns unformatted data to the field.
	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

39.1.25.4.1 TField.AccessError Method

Pascal

```
function AccessError(const TypeName: string): EDatabaseError; dynamic;
```

C++

```
__fastcall EDatabaseError AccessError(const AnsiString TypeName);
```

Description

This is AccessError, a member of class TField.

39.1.25.4.2 TField.Assign Method

Copies the Value (see page 7180) property from another field component or assigns another object to the field component.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use Assign to copy the Value (see page 7180) property from one field to another. The fields must have compatible data types.

When Source is another field object, Assign calls the AssignValue (see page 7124) method, using the Source field's Value (see page 7180) property. Calling Assign with the Source parameter set to nil (Delphi) or NULL (C++) assigns a blank value to the field component. Calling Assign with any other object type calls the inherited method, which allows assignment from objects that implement a TField (see page 7115) object in their AssignTo method.

See Also

AssignValue (see page 7124)

Clear (see page 7126)

DataType (see page 7159)

AssignTo (see page 2931)

Value (see page 7180)

39.1.25.4.3 TField.AssignValue Method

Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.

Pascal

```
procedure AssignValue(const Value: TVarRec);
```

C++

```
__fastcall AssignValue(const TVarRec Value);
```

Description

Use AssignValue to assign a value to a field and, in Delphi, that comes from an array of const. AssignValue uses an As? property for the assignment based on the type of the Value (see page 7180) parameter:

Value (see page 7180)	As... property used
vtInteger	AsInteger (see page 7149)
vtBoolean	AsBoolean (see page 7147)
vtChar	AsString (see page 7151)
vtExtended	AsFloat (see page 7149)
vtString	AsString (see page 7151)
vtPChar	AsString (see page 7151)
vtAnsiString	AsString (see page 7151)
vtCurrency	AsCurrency (see page 7148)
vtVariant	AsVariant (see page 7152)

If Value (see page 7180) is not one of the types in the table, but is of type TObject or a TObject descendant, AssignValue uses the Assign (see page 7124) method to assign the value to the field.

AssignValue exists primarily for internal use. In C++ applications and components, use the TField::Value (see page 7180) property instead. In most Delphi situations, you should use the data type-specific assignment properties like AsString (see page 7151) and AsInteger (see page 7149). In cases where a component or an application must be able to make field assignments from untyped arrays, the syntax for AssignValue looks like that below. In this example, a custom data class uses AssignValue in a member method named SetFieldValues to assign values to multiple fields. The values for the field assignments come from an untyped array.

```
procedure TCustomDataClass.SetFieldValues(Values: array of const);
var
  i: Integer;
begin
for i := 0 to High(Values) do
    Fields[i].AssignValue(Values[i]);
end;
```

See Also

AsBoolean (see page 7147)

AsCurrency (see page 7148)

AsFloat (see page 7149)

AsInteger (see page 7149)

Assign (see page 7124)

AsString (see page 7151)

39.1.25.4.4 TField.Bind Method

Pascal

```
procedure Bind(Binding: Boolean); virtual;
```

C++

```
virtual __fastcall Bind(Boolean Binding);
```

Description

This is Bind, a member of class TField.

39.1.25.4.5 TField.Change Method

Pascal

```
procedure Change; virtual;
```

C++

```
virtual __fastcall Change();
```

Description

This is Change, a member of class TField.

39.1.25.4.6 TField.CheckInactive Method

Pascal

```
procedure CheckInactive;
```

C++

```
__fastcall CheckInactive();
```

Description

This is CheckInactive, a member of class TField.

39.1.25.4.7 TField.CheckTypeSize Method

Pascal

```
class procedure CheckTypeSize(Value: Integer); virtual;
```

C++

```
virtual __fastcall CheckTypeSize(int Value);
```

Description

This is CheckTypeSize, a member of class TField.

39.1.25.4.8 TField.Clear Method

Sets the value of the field to NULL.

Pascal

```
procedure Clear; virtual;
```

C++

```
virtual __fastcall Clear();
```

Description

Call the Clear method to give the field a blank value (a NULL value in SQL terms).

See Also

IsNull (see page 7169)

Required (see page 7177)

39.1.25.4.9 TField.CopyData Method

Pascal

```
procedure CopyData(Source: Pointer; Dest: Pointer); virtual;
```

C++

```
virtual __fastcall CopyData(void * Source, void * Dest);
```

Description

This is CopyData, a member of class TField.

39.1.25.4.10 TField.DataChanged Method

Pascal

```
procedure DataChanged;
```

C++

```
__fastcall DataChanged();
```

Description

This is DataChanged, a member of class TField.

39.1.25.4.11 TField.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor (see page 2839)

DefineBinaryProperty (see page 2837)

DefineProperties (see page 2931)

DefineProperty (see page 2838)

39.1.25.4.12 TField.FocusControl Method

Forces a form's focus to the first data-aware component associated with the field object.

Pascal

```
procedure FocusControl;
```

C++

```
__fastcall FocusControl();
```

Description

Call FocusControl to switch the focus of the form to a data-aware control associated with the field. Because it is possible that the field's value can change even if its associated data-aware control does not have the focus, calling FocusControl can aid in drawing attention to a new value when it fails to be validated.

See Also

OnValidate (see page 7183)

Validate (see page 7144)

Working with Field Component Methods at Runtime

39.1.25.4.13 TField.FreeBuffers Method

Pascal

```
procedure FreeBuffers; virtual;
```

C++

```
virtual __fastcall FreeBuffers();
```

Description

This is FreeBuffers, a member of class TField.

39.1.25.4.14 TField.GetAsBCD Method

Pascal

```
function GetAsBCD: TBcd; virtual;
```

C++

```
virtual __fastcall TBcd GetAsBCD();
```

Description

This is GetAsBCD, a member of class TField.

39.1.25.4.15 TField.GetAsBoolean Method

Pascal

```
function GetAsBoolean: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetAsBoolean();
```

Description

This is GetAsBoolean, a member of class TField.

39.1.25.4.16 TField.GetAsByteArray Method

Pascal

```
function GetAsByteArray: Variant; virtual;
```

C++

```
virtual __fastcall Variant GetAsByteArray();
```

Description

This is GetAsByteArray, a member of class TField.

39.1.25.4.17 TField.GetAsCurrency Method

Pascal

```
function GetAsCurrency: Currency; virtual;
```

C++

```
virtual __fastcall Currency GetAsCurrency();
```

Description

This is GetAsCurrency, a member of class TField.

39.1.25.4.18 TField.GetAsDateTime Method

Pascal

```
function GetAsDateTime: TDateTime; virtual;
```

C++

```
virtual __fastcall TDateTime GetAsDateTime();
```

Description

This is GetAsDateTime, a member of class TField.

39.1.25.4.19 TField.GetAsFloat Method

Pascal

```
function GetAsFloat: Double; virtual;
```

C++

```
virtual __fastcall Double GetAsFloat();
```

Description

This is GetAsFloat, a member of class TField.

39.1.25.4.20 TField.GetAsInteger Method

Pascal

```
function GetAsInteger: Longint; virtual;
```

C++

```
virtual __fastcall Longint GetAsInteger();
```

Description

This is GetAsInteger, a member of class TField.

39.1.25.4.21 TField.GetAsSQLTimeStamp Method

Pascal

```
function GetAsSQLTimeStamp: TSQLTimeStamp; virtual;
```

C++

```
virtual __fastcall TSQLTimeStamp GetAsSQLTimeStamp();
```

Description

This is GetAsSQLTimeStamp, a member of class TField.

39.1.25.4.22 TField.GetAsString Method

Pascal

```
function GetAsString: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetAsString();
```

Description

This is GetAsString, a member of class TField.

39.1.25.4.23 TField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; virtual;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TField.

39.1.25.4.24 TField.GetAsWideString Method

Pascal

```
function GetAsWideString: WideString; virtual;
```

C++

```
virtual __fastcall BSTR GetAsWideString();
```

Description

This is GetAsWideString, a member of class TField.

39.1.25.4.25 TField.GetCanModify Method

Pascal

```
function GetCanModify: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetCanModify();
```

Description

This is GetCanModify, a member of class TField.

39.1.25.4.26 TField.GetClassDesc Method

Pascal

```
function GetClassDesc: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetClassDesc();
```

Description

This is GetClassDesc, a member of class TField.

39.1.25.4.27 TField.GetData Method

Returns the unformatted data for the field.

Pascal

```
function GetData(Buffer: Pointer; NativeFormat: Boolean = True): Boolean;
```

C++

```
__fastcall Boolean GetData(void * Buffer, Boolean NativeFormat = True);
```

Description

Call GetData to return the data in a field to a buffer. GetData is intended primarily for internal use.

Buffer must have sufficient space allocated for the data. Use the DataSize (see page 7157) property to determine the space required.

Unlike the DisplayText (see page 7162), Text (see page 7179), or AsXXX properties, GetData does not translate the data into a specific type. Instead, it uses a type that reflects the underlying database information unless you set NativeFormat to false. NativeFormat indicates whether the raw database information must be translated into the underlying type associated with the specific type of database field. When NativeFormat is true, the dataset does not translate the raw database information. When

NativeFormat is false, the dataset must translate the field value into the type associated with the field.

If the data is NULL, GetData returns false and no data is transferred to Buffer. Otherwise, GetData returns true.

Note: GetData cannot be used to read data from BLOB or memo fields. To read BLOB data, use the stream returned by the dataset's CreateBlobStream method or the BLOB field's SaveToStream method.

C++ Examples:

```
/*
This example requires a button, a test edit, and a populated
ClientDataSet.  Pipe the ClientDataSet through a DataSource
to a DGGrid or DBNavigator to control the current field.
Cast the data correctly according to the field type when
assigning to the test edit.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // Retrieve the "raw" data from Field1
  if (!CDS->Fields->Fields[0]->IsBlob(__classid(TField))) // GetData does not work for BLOB
fields
  {
    void *MyBuffer = malloc(CDS->Fields->Fields[0]->DataSize);
    try
    {
      if (!CDS->Fields->Fields[0]->GetData(MyBuffer))
        MessageDlg(CDS->Fields->Fields[0]->DisplayName + "is blank.", mtInformation,
TMsgDlgButtons() << mbOK,0);
      else
        // Do something with the data;
        Edit1->Text = (char *)MyBuffer; // for a stringfield
    }
    __finally
    {
      free(MyBuffer);
    }
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CDS->LoadFromFile("../CDS.XML");
}
```

Delphi Examples:

```
{
This example requires a button, a test edit, and a populated
ClientDataSet.  Pipe the ClientDataSet through a DataSource
to a DGGrid or DBNavigator to control the current field.
Cast the data correctly according to the field type when
assigning to the test edit.
}

{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button1Click(Sender: TObject);
var MyBuffer: Pointer;
begin
{ Retrieve the "raw" data from Field1 }
with CDS.Fields[0] do
  begin
    if not IsBlob then { this does not work for BLOB fields }
```

```
      begin
        { Allocate space }
        MyBuffer:= GetMemory(DataSize);
        try
          if not GetData(MyBuffer) then
            MessageDlg(DisplayName + ' is NULL', mtInformation, [mbOK], 0)
          else
            { Do something with the data };
            Edit1.Text:= string(MyBuffer); // for a stringfield
        finally
          { Free the space }
          FreeMem(MyBuffer, DataSize);
        end;
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
  {$IFDEF UNICODE}
    CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
  {$ELSE}
    CDS.LoadFromFile(gsAppPath + 'CDS.XML');
  {$ENDIF}
  end;
```

See Also

DataSize (see page 7157)

DisplayText (see page 7162)

IsBlob (see page 7136)

SetData (see page 7141)

Text (see page 7179)

CreateBlobStream (see page 6922)

SaveToStream (see page 6816)

39.1.25.4.28 TField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; virtual;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TField.

39.1.25.4.29 TField.GetDefaultWidth Method

Pascal

```
function GetDefaultWidth: Integer; virtual;
```

C++

```
virtual __fastcall int GetDefaultWidth();
```

Description

This is GetDefaultWidth, a member of class TField.

39.1.25.4.30 TField.GetFieldNo Method

Pascal

```
function GetFieldNo: Integer; virtual;
```

C++

```
virtual __fastcall int GetFieldNo();
```

Description

This is GetFieldNo, a member of class TField.

39.1.25.4.31 TField.GetHasConstraints Method

Pascal

```
function GetHasConstraints: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetHasConstraints();
```

Description

This is GetHasConstraints, a member of class TField.

39.1.25.4.32 TField.GetIsNull Method

Pascal

```
function GetIsNull: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetIsNull();
```

Description

This is GetIsNull, a member of class TField.

39.1.25.4.33 TField.GetParentComponent Method

Returns the component which directs the loading and saving of the field.

Pascal

```
function GetParentComponent: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetParentComponent();
```

Description

The streaming system that loads and saves components calls GetParentComponent to identify the component that uses this field to represent one of its parts. If the field represents a subfield in an object field, GetParentComponent returns the value of ParentField (see page 7176). Otherwise, GetParentComponent returns the value of DataSet (see page 7157).

See Also

HasParent (see page 7135)

GetChildren

DataSet (see page 7157)

ParentField (see page 7176)

39.1.25.4.34 TField.GetSize Method

Pascal

```
function GetSize: Integer; virtual;
```

C++

```
virtual __fastcall int GetSize();
```

Description

This is GetSize, a member of class TField.

39.1.25.4.35 TField.GetText Method

Pascal

```
procedure GetText(var Text: String; DisplayText: Boolean); virtual;
```

C++

```
virtual __fastcall GetText(AnsiString Text, Boolean DisplayText);
```

Description

This is GetText, a member of class TField.

39.1.25.4.36 TField.GetWideText Method

Pascal

```
procedure GetWideText(var Text: WideString; DisplayText: Boolean); virtual;
```

C++

```
virtual __fastcall GetWideText(BSTR Text, Boolean DisplayText);
```

Description

This is GetWideText, a member of class TField.

39.1.25.4.37 TField.HasParent Method

Returns true, indicating that field components act as part of a larger composite.

Pascal

```
function HasParent: Boolean; override;
```

C++

```
virtual __fastcall Boolean HasParent();
```

Description

The streaming system that loads and saves components calls HasParent to determine when a component acts as part of a composite. The component that represents the entire composite is returned by the GetParentComponent (see page 7134) method. The parent component is responsible for directing the loading and saving of its subcomponents.

See Also

GetParentComponent (see page 7134)

GetChildren

39.1.25.4.38 TField.IsBlob Method

Indicates whether the field represents the data in a binary large object (BLOB) field.

Pascal

```
class function IsBlob: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsBlob();
```

Description

Call IsBlob to determine whether the field represents BLOB data. For example, before using the GetData (see page 7131) or SetData (see page 7141) method, call IsBlob to determine whether the method can be used. The IsBlob method for TField (see page 7115) always returns false. Descendants of TField (see page 7115) that implement blob fields override this method to indicate that they represent BLOB data.

See Also

GetData (see page 7131)

SetData (see page 7141)

39.1.25.4.39 TField.IsValidChar Method

Indicates whether a particular character is valid for the field.

Pascal

```
function IsValidChar(InputChar: Char): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsValidChar(Char InputChar);
```

Description

Call IsValidChar to determine if the character specified by InputChar can appear in the text representation of the field's value. Many data-aware controls use IsValidChar to check characters entered by the user to determine if they are valid. IsValidChar returns true if InputChar can appear in the field. IsValidChar returns false if InputChar is not a valid character for the field type.

The valid characters for a field object are given by the ValidChars (see page 7179) property. By default, field objects accept all characters as valid. For each DataType (see page 7159) that restricts the set of valid characters, the default valid characters are listed below:

DataType (see page 7159)	Valid characters
ftSmallint	Digits 0 to 9, plus sign (+), and minus sign (-).
ftWord	Digits 0 to 9, plus sign (+), and minus sign (-)
ftAutoInc	Digits 0 to 9, plus sign (+), and minus sign (-)
ftInteger	Digits 0 to 9, plus sign (+), and minus sign (-)
ftLargeInt	Digits 0 to 9, plus sign (+), and minus sign (-)
ftCurrency	Digits 0 to 9, plus sign (+), minus sign (-), the letter E (E or e) and the decimal separator from the regional settings in the Windows control panel.
ftFloat	Digits 0 to 9, plus sign (+), minus sign (-), the letter E (E or e) and the decimal separator from the regional settings in the Windows control panel
ftBCD	Digits 0 to 9, plus sign (+), minus sign (-), and the decimal separator from the regional settings in the Windows control panel.

Note: For some datatypes, such as ftGraphic, IsValidChar returns true for all characters although not every character is valid for the field. IsValidChar provides only a rough validation of input characters. More refined character-by-character validation can be provided by an edit mask. Also, the value for a field is validated as a whole when it is written to the database record.

See Also

DataType (see page 7159)

EditMask (see page 7163)

OnValidate (see page 7183)

ValidChars (see page 7179)

39.1.25.4.40 TField.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

39.1.25.4.41 TField.PropertyChanged Method

Pascal

```
procedure PropertyChanged(LayoutAffected: Boolean);
```

C++

```
__fastcall PropertyChanged(Boolean LayoutAffected);
```

Description

This is PropertyChanged, a member of class TField.

39.1.25.4.42 TField.ReadState Method

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

This is ReadState, a member of class TField.

39.1.25.4.43 TField.RefreshLookupList Method

Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).

Pascal

```
procedure RefreshLookupList;
```

C++

```
__fastcall RefreshLookupList();
```

Description

Call RefreshLookupList to load the lookup cache when setting the LookupCache (see page 7170) property at runtime or when the values of the LookupResultField (see page 7173) or the LookupKeyFields (see page 7172) in the LookupDataSet (see page 7171) may have changed.

See Also

LookupCache (see page 7170)

LookupDataSet (see page 7171)

LookupKeyFields (see page 7172)

LookupList (see page 7172)

LookupResultField (see page 7173)

Defining a Lookup Field

39.1.25.4.44 TField.SetAsBCD Method

Pascal

```
procedure SetAsBCD(const Value: TBcd); virtual;
```

C++

```
virtual __fastcall SetAsBCD(const TBcd Value);
```

Description

This is SetAsBCD, a member of class TField.

39.1.25.4.45 TField.SetAsBoolean Method

Pascal

```
procedure SetAsBoolean(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetAsBoolean(Boolean Value);
```

Description

This is SetAsBoolean, a member of class TField.

39.1.25.4.46 TField.SetAsByteArray Method

Pascal

```
procedure SetAsByteArray(const Value: Variant); virtual;
```

C++

```
virtual __fastcall SetAsByteArray(const Variant Value);
```

Description

This is SetAsByteArray, a member of class TField.

39.1.25.4.47 TField.SetAsCurrency Method

Pascal

```
procedure SetAsCurrency(Value: Currency); virtual;
```

C++

```
virtual __fastcall SetAsCurrency(Currency Value);
```

Description

This is SetAsCurrency, a member of class TField.

39.1.25.4.48 TField.SetAsDateTime Method

Pascal

```
procedure SetAsDateTime(Value: TDateTime); virtual;
```

C++

```
virtual __fastcall SetAsDateTime(TDateTime Value);
```

Description

This is SetAsDateTime, a member of class TField.

39.1.25.4.49 TField.SetAsFloat Method

Pascal

```
procedure SetAsFloat(Value: Double); virtual;
```

C++

```
virtual __fastcall SetAsFloat(Double Value);
```

Description

This is SetAsFloat, a member of class TField.

39.1.25.4.50 TField.SetAsInteger Method

Pascal

```
procedure SetAsInteger(Value: Longint); virtual;
```

C++

```
virtual __fastcall SetAsInteger(Longint Value);
```

Description

This is SetAsInteger, a member of class TField.

39.1.25.4.51 TField.SetAsSQLTimeStamp Method

Pascal

```
procedure SetAsSQLTimeStamp(const Value: TSQLTimeStamp); virtual;
```

C++

```
virtual __fastcall SetAsSQLTimeStamp(const TSQLTimeStamp Value);
```

Description

This is SetAsSQLTimeStamp, a member of class TField.

39.1.25.4.52 TField.SetAsString Method

Pascal

```
procedure SetAsString(const Value: string); virtual;
```

C++

```
virtual __fastcall SetAsString(const AnsiString Value);
```

Description

This is SetAsString, a member of class TField.

39.1.25.4.53 TField.SetAsVariant Method

Pascal

```
procedure SetAsVariant(const Value: Variant); virtual;
```

C++

```
virtual __fastcall SetAsVariant(const Variant Value);
```

Description

This is SetAsVariant, a member of class TField.

39.1.25.4.54 TField.SetAsWideString Method

Pascal

```
procedure SetAsWideString(const Value: WideString); virtual;
```

C++

```
virtual __fastcall SetAsWideString(const BSTR Value);
```

Description

This is SetAsWideString, a member of class TField.

39.1.25.4.55 TField.SetAutoGenerateValue Method

Pascal

```
procedure SetAutoGenerateValue(const Value: TAutoRefreshFlag);
```

C++

```
__fastcall SetAutoGenerateValue(const TAutoRefreshFlag Value);
```

Description

This is SetAutoGenerateValue, a member of class TField.

39.1.25.4.56 TField.SetData Method

Assigns unformatted data to the field.

Pascal

```
procedure SetData(Buffer: Pointer; NativeFormat: Boolean = True);
```

C++

```
__fastcall SetData(void * Buffer, Boolean NativeFormat = True);
```

Description

Use SetData to assign data to a field in the format that is used by the underlying database table. Ultimately, any method of setting the data for the field component calls SetData, once the data has been parsed into the format expected by the database table.

SetData calls the OnValidate (see page 7183) event handler before writing the data to the current record buffer. If the data is successfully written, SetData calls the OnChange (see page 7181) event handler.

Buffer contains the field value in the field components underlying type. NativeFormat indicates whether the value in Buffer must be translated before the dataset assigns its value to its internal record.

Note: SetData cannot be used to assign data to BLOB or memo fields. To write BLOB data, use the stream returned by the dataset's CreateBlobStream method or the BLOB field's LoadFromStream method.

See Also

GetData (see page 7131)

IsBlob (see page 7136)

OnChange (see page 7181)

OnValidate (see page 7183)

CreateBlobStream (see page 6922)

LoadFromStream (see page 6815)

39.1.25.4.57 TField.SetDataSet Method

Pascal

```
procedure SetDataSet(ADataSet: TDataSet); virtual;
```

C++

```
virtual __fastcall SetDataSet(TDataSet ADataSet);
```

Description

This is SetDataSet, a member of class TField.

39.1.25.4.58 TField.SetDataType Method

Pascal

```
procedure SetDataType(Value: TFieldType);
```

C++

```
__fastcall SetDataType(TFieldType Value);
```

Description

This is SetDataType, a member of class TField.

39.1.25.4.59 TField.SetFieldKind Method

Pascal

```
procedure SetFieldKind(Value: TFieldKind); virtual;
```

C++

```
virtual __fastcall SetFieldKind(TFieldKind Value);
```

Description

This is SetFieldKind, a member of class TField.

39.1.25.4.60 TField.SetFieldType Method

Provides the interface for a method that can set the DataType (see page 7159) for the field.

Pascal

```
procedure SetFieldType(Value: TFieldType); virtual;
```

C++

```
virtual __fastcall SetFieldType(TFieldType Value);
```

Description

As implemented in TField (see page 7115), SetFieldType does nothing. Descendant classes override SetFieldType to set the DataType (see page 7159) for the field component. SetFieldType is useful for derived classes that can support more than one underlying data type. For example, a TBlobField (see page 6801) may support ftBlob, ftFmtMemo, ftParadoxOle, ftDBaseOle, or ftTypedBinary.

See Also

DataType (see page 7159)

TBlobField (see page 6801)

39.1.25.4.61 TField.SetParentComponent Method

Pascal

```
procedure SetParentComponent(AParent: TComponent); override;
```

C++

```
virtual __fastcall SetParentComponent(TComponent * AParent);
```

Description

This is SetParentComponent, a member of class TField.

39.1.25.4.62 TField.SetParentField Method

Pascal

```
procedure SetParentField(AField: TObjectField); virtual;
```

C++

```
virtual __fastcall SetParentField(TObjectField AField);
```

Description

This is SetParentField, a member of class TField.

39.1.25.4.63 TField.SetSize Method

Pascal

```
procedure SetSize(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetSize(int Value);
```

Description

This is SetSize, a member of class TField.

39.1.25.4.64 TField.SetText Method

Pascal

```
procedure SetText(const Value: string); virtual;
```

C++

```
virtual __fastcall SetText(const AnsiString Value);
```

Description

This is SetText, a member of class TField.

39.1.25.4.65 TField.SetVarValue Method

Pascal

```
procedure SetVarValue(const Value: Variant); virtual;
```

C++

```
virtual __fastcall SetVarValue(const Variant Value);
```

Description

This is SetVarValue, a member of class TField.

39.1.25.4.66 TField.SetWideText Method

Pascal

```
procedure SetWideText(const Value: WideString); virtual;
```

C++

```
virtual __fastcall SetWideText(const BSTR Value);
```

Description

This is SetWideText, a member of class TField.

39.1.25.4.67 TField.Validate Method

Generates an OnValidate (see page 7183) event.

Pascal

```
procedure Validate(Buffer: Pointer);
```

C++

```
__fastcall Validate(void * Buffer);
```

Description

Call Validate to validate the unformatted data pointed to by the Buffer parameter. Validate is called by the dataset when writing new values to the current record.

The value of Buffer does not affect the value of the field outside the OnValidate (see page 7183) event handler.

See Also

OnValidate (see page 7183)

Value (see page 7180)

39.1.25.5 TField Properties

The properties of the TField class are listed here.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7180)	Represents the data in a field component.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.

	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.25.5.1 TField.Alignment Property

Determines how the field's data is displayed within a data-aware control.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to specify the position of the field's data within data-aware controls. These are the possible values:

Value (see page 7180)	Meaning
taLeftJustify	Align data to the left side of the control
taCenter	Center data horizontally in the control
taRightJustify	Align data to the right side of the control

See Also

DisplayLabel (see page 7160)

DisplayName (see page 7160)

DisplayText (see page 7162)

DisplayWidth (see page 7162)

TDBEdit (see page 8735)

39.1.25.5.2 TField.AsBCD Property

Represents the field's value as a TBcd value.

Pascal

```
property AsBCD: TBcd;
```

C++

```
__property TBcd AsBCD;
```

Description

Use AsBCD to read the value of the field's data into a TBcd or assign a TBcd value to the contents of the field.

As implemented by TField (see page 7115), reading AsBCD returns the value of AsCurrency (see page 7148) converted to a TBcd value, and setting AsBCD sets the AsCurrency (see page 7148) property after converting a supplied TBcd value to the Currency type. Some TField (see page 7115) descendants override AsBCD to perform a more direct conversion.

Note: If the field does not support Currency values, the field component raises an exception when an attempt is made to get or set the AsBCD or AsCurrency (see page 7148) property.

See Also

DataType (see page 7159)

Value (see page 7180)

BCDToCurr (see page 13637)

CurrToBCD (see page 13640)

39.1.25.5.3 TField.AsBoolean Property

Represents the field's value as a boolean value.

Pascal

```
property AsBoolean: Boolean;
```

C++

```
__property Boolean AsBoolean;
```

Description

Use AsBoolean to read the value of the field's data into a boolean or assign a boolean value to the contents of the field. TField (see page 7115) does not support boolean values, and raises an exception when an attempt is made to get or set the AsBoolean property.

Descendants of TField (see page 7115) that represent boolean fields or that support conversions between the field's Value (see page 7180) property and a boolean value, override AsBoolean to read and write the value of the field as a boolean value.

See Also

DataType (see page 7159)

Value (see page 7180)

39.1.25.5.4 TField.AsCurrency Property

Represents the field's value as a Currency value.

Pascal

```
property AsCurrency: Currency;
```

C++

```
__property Currency AsCurrency;
```

Description

Use AsCurrency to read the value of the field's data into an object or variable of type Currency or to assign a Currency value to the contents of the field.

As implemented by TField (see page 7115), reading AsCurrency returns the value of AsFloat (see page 7149) converted to a Currency value, and setting AsCurrency sets the AsFloat (see page 7149) property after converting a supplied Currency value to a double. Some TField (see page 7115) descendants override AsCurrency to work directly with Currency values rather than implicitly converting between the Currency type and doubles.

Note: If the field does not support double values, the field component raises an exception when an attempt is made to get or set the AsFloat (see page 7149) or AsCurrency property.

See Also

AsFloat (see page 7149)

DataType (see page 7159)

Value (see page 7180)

39.1.25.5.5 TField.AsDateTime Property

Represents the field's value as a TDateTime value.

Pascal

```
property AsDateTime: TDateTime;
```

C++

```
__property TDateTime AsDateTime;
```

Description

Use AsDateTime to read the value of the field's data into an object or variable of type TDateTime, or to assign a TDateTime value to the contents of the field. TField (see page 7115) does not support TDateTime values, and raises an exception when an attempt is made to get or set the AsDateTime property.

Descendants of TField (see page 7115) that represent datetime (timestamp) fields or that support conversions between the field's Value (see page 7180) property and a TDateTime value, override AsDateTime to read and write the value of the field as a TDateTime value.

See Also

DataType (see page 7159)

Value (see page 7180)

AsSQLTimeStamp (see page 7150)

39.1.25.5.6 TField.AsFloat Property

Represents the field's value as a double value.

Pascal

```
property AsFloat: Double;
```

C++

```
__property Double AsFloat;
```

Description

Use AsFloat to read the value of the field's data into a double, or to assign a Double value to the contents of the field. TField (see page 7115) does not support floating point values, and raises an exception when an attempt is made to get or set the AsFloat property.

Descendants of TField (see page 7115) that represent floating point numeric fields or that support conversions between the field's Value (see page 7180) property and a double value, override AsFloat to read and write the value of the field as a double.

See Also

AsCurrency (see page 7148)

DataType (see page 7159)

Value (see page 7180)

39.1.25.5.7 TField.AsInteger Property

Represents the field's value as a 32-bit integer.

Pascal

```
property AsInteger: Longint;
```

C++

```
__property Longint AsInteger;
```

Description

Use AsInteger to read the value of the field's data into an integer, or to assign an integer value to the contents of the field. TField (see page 7115) does not support integer values, and raises an exception when an attempt is made to get or set the AsInteger property.

Descendants of TField (see page 7115) that represent integer fields or that support conversions between the field's Value (see page 7180) property and an integer value, override AsInteger to read and write the value of the field as an integer value.

Delphi Examples:

```
{
This example uses the BeforeInsert event to do data
validation. If the StrToInt function raises an exception,
the edit control's contents are set to a valid value so
that the assignment to the INTEGER field in the dataset
will succeed.  Continuing after the exception causes a
'0' to be placed in the PORTA field.
DataSet design:
This example requires a ClientDataSet, a DBGrid, a
DataSource and a DBNavigator.  Name the ClientDataSet CDS.
Set the DataSet property in the DataSource to CDS and name
that DataSource DS.  Set the DataSource in the DBNavigator
to DS the DataSource in the DBGrid to DS.  Right click on CDS
```

```
  and select the Fields Editor, then right click on the Field
  Editor form and create two fields by selecting "new field".
  In the field configuration window, set the name to HOST, the
  Component value will automatically be set to CDSHOST.  Set
  the filed type to String.  In the field configuration window
  for the second field, set the name to PORTA, the Component
  value will automatically be set to CDSPORTA.  Set the field
  type to Integer.  The file used to load CDS must have an
  attribute name of HOST with an attribute type of string and
  an attribute name of PORTA with an attribute type of integer.
  Add DB, DBCtrls, Grids, DBGrids and DBClient to the uses clause.
  }
  {$IFNDEF UNICODE}
  uses SwSystem;
  {$ENDIF}

  procedure TForm1.Button3Click(Sender: TObject);
  begin
    CDS.Insert;
    CDS.FieldByName('PORTA').AsInteger := StrToInt(Edit1.Text);
    CDS.Post;
  end;

  procedure TForm1.CDSBeforeInsert(DataSet: TDataSet);
  begin
    try
    { Make sure edit field can be converted to integer --
     this will raise an exception if it can't }
      StrToInt(Edit1.Text);
    except
      Edit1.Text := '0';
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
  {$IFDEF UNICODE}
    CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
  {$ELSE}
    CDS.LoadFromFile(gsAppPath + 'CDS.XML');
  {$ENDIF}
  end;
```

See Also

AsFloat (see page 7149)

DataType (see page 7159)

Value (see page 7180)

39.1.25.5.8 TField.AsSQLTimeStamp Property

Represents the field's value as a TSQLTimeStamp.

Pascal

```
property AsSQLTimeStamp: TSQLTimeStamp;
```

C++

```
__property TSQLTimeStamp AsSQLTimeStamp;
```

Description

Use AsSQLTimeStamp to read the value of the field's data into a TSQLTimeStamp record (Delphi) or structure (C++), or to assign a TSQLTimeStamp value to the contents of the field. TField (see page 7115) does not support TSQLTimeStamp

values, and raises an exception when an attempt is made to get or set the AsSQLTimeStamp property.

Descendants of TField (see page 7115) that represent SQL timestamp fields or that support conversions between the field's Value (see page 7180) property and TSQLTimeStamp, override AsSQLTimeStamp to read and write the value of the field as a TSQLTimeStamp record (Delphi) or structure (C++).

See Also

DataType (see page 7159)

Value (see page 7180)

AsDateTime (see page 7148)

39.1.25.5.9 TField.AsString Property

Represents the field's value as a string (Delphi) or an AnsiString (C++).

Pascal

```
property AsString: string;
```

C++

```
__property AnsiString AsString;
```

Description

Reading the AsString property returns the abbreviated class name of the field component enclosed in parentheses. The abbreviated class name is built by stripping the leading 'T' from the class name and removing the trailing word 'FIELD', except in the case of TField (see page 7115) where that would leave an empty string. Thus, the abbreviated class name of TField (see page 7115) is "FIELD", the abbreviated class name of TVarBytesField (see page 7507) is "VarBytes", and so on.

Trying to set the AsString property raises an exception.

Descendants of TField (see page 7115) that represent string fields or that support conversions between the field's Value (see page 7180) property and a string override AsString to read and write the value of the field as a different, more meaningful, string.

Delphi Examples:

```
{
This example uses a button to copy the value of a field in
the previous record into the corresponding field in the
current record.  The field to copy is specified by using
FindField and the name of the field.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   SavePlace: TBookmark;
   PrevValue: Variant;
begin
   with Customers do
   begin
    { get a bookmark so that we can return to the same record }
    SavePlace := GetBookmark;
    try
      { move to prior record}
      FindPrior;
      { get the value }
      { This is the safe way to get 'CustNo' field }
      PrevValue := FindField('Field2').Value;
      { This is *not* the safe way to change 'CustNo' field }
//    PrevValue := Fields[1].Value;
      { Move back to the bookmark
```

```
        this may not be the next record anymore
        if something else is changing the dataset asynchronously }
        GotoBookmark(SavePlace);
        { Set the value }
        Edit;
        { This is the safe way to change 'CustNo' field }
        FindField('Field2').AsString := PrevValue;
        { This is *not* the safe way to change 'CustNo' field }
//        Fields[1].AsString := PrevValue;
        { Free the bookmark }
      finally
        FreeBookmark(SavePlace);
      end;
    end;
  end;
```

See Also

DataType (see page 7159)

Value (see page 7180)

39.1.25.5.10 TField.AsVariant Property

Represents the Value (see page 7180) of the field as a Variant.

Pascal

```
property AsVariant: Variant;
```

C++

```
__property Variant AsVariant;
```

Description

Use AsVariant to read the value of the field's data into a Variant, or to assign a Variant value to the contents of the field. TField (see page 7115) does not support Variant values, and raises an exception when an attempt is made to read the AsVariant property. TField (see page 7115) also raises an exception when an attempt is made to set the AsVariant property, unless the Variant is of type varNull, in which case TField (see page 7115) calls the Clear (see page 7126) method.

Descendants of TField (see page 7115) implement the AsVariant method to get or set the value of the field as a Variant.

See Also

Clear (see page 7126)

Value (see page 7180)

39.1.25.5.11 TField.AsWideString Property

Pascal

```
property AsWideString: WideString;
```

C++

```
__property BSTR AsWideString;
```

Description

This is AsWideString, a member of class TField.

39.1.25.5.12 TField.AttributeSet Property

Specifies the name of an attribute set in the data dictionary to apply to the field component.

Pascal

```
property AttributeSet: string;
```

C++

```
__property AnsiString AttributeSet;
```

Description

An attribute set defines the display and format properties of a field component that are displayed in a data-aware control at design time. It also defines any constraint and default properties of the field component. Attribute sets are only available on Windows.

Attribute sets are stored in the data dictionary by name so that at design time developers can create and apply the same set of attributes to different field components in one or more datasets. When an attribute set is assigned to a field component at design time and that field component is owned by a TQuery or TStoredProc dataset, AttributeSet contains the name of the assigned attribute set.

See Also

TDataSet (see page 6894)

TQuery (see page 10059)

TStoredProc (see page 10139)

39.1.25.5.13 TField.AutoGenerateValue Property

Indicates whether the field values can be generated by the server rather than entered by the user.

Pascal

```
property AutoGenerateValue: TAutoRefreshFlag;
```

C++

```
__property TAutoRefreshFlag AutoGenerateValue;
```

Description

Use AutoGenerateValue to indicate when field values can be generated by the database. Such fields include autoincrement fields and fields with default values.

Some database drivers can't determine which fields can have values supplied automatically by the server. This can lead to problems when posting edits. For these cases, applications can provide this information by setting the AutoGenerateValue property. Use this property to indicate whether the field is an autoincrement field (arAutoInc), a field with a default value (arDefault), or a field whose value is not automatically generated (arNone).

AutoGenerateValue is not always required. Fields with an AutoGenerateValue property of arNone may still be updated if the driver supplies metadata that indicates it has a default value or an autoincrement value. However, it is good practice to supply this information because not all drivers supply this metadata.

If you do use AutoGenerateValue for any fields that have values generated by the database, you must set AutoGenerateValue to arAutoInc or arDefault on every field whose value is automatically generated by the database. This is because when records are posted, the application immediately executes a query to fetch the automatically-generated values. If you leave an automatically-generated field with AutoGenerateValue set to arNone, this query looks for a record with a blank value on that field, which will fail to locate the correct record.

Warning: AutoGenerateValue can fail in some cases when the automatically-generated field value is on a key field. This is because the query that re-fetches the new field value can't always uniquely locate the record that was just posted.

Note: Trying to change the AutoGenerateValue property when the field's dataset is open generates an exception. Thus, AutoGenerateValue only works with persistent field components, which remain when the dataset is closed.

See Also

AutoRefresh (see page 10021)

Refresh (see page 6991)

Value (see page 7180)

39.1.25.5.14 TField.Calculated Property

Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.

Pascal

```
property Calculated: Boolean;
```

C++

```
__property Boolean Calculated;
```

Description

Use Calculated to determine if the field is a calculated field. When Calculated is true and the OnCalcFields event occurs, the OnCalcFields event handler updates the value of the field. The default value is false.

The value of a Calculated field is not stored in or retrieved from the physical tables underlying a dataset. Instead, calculated fields are calculated for each record in the table by the dataset's OnCalcFields event handler, which typically uses expressions involving values from other fields in the record to generate a value for each calculated field.

For example, a dataset might have non-calculated fields for Quantity and UnitPrice, and a calculated field for ExtendedPrice, which would be calculated by multiplying the values of the Quantity and UnitPrice fields in an OnCalcFields event handler.

Note: Do not change the setting of Calculated in application code. Calculated is automatically set to true by the Fields editor at design time if a field's value is specified as calculated.

Note: Do not confuse the Calculated property with the InternalCalcField property of a field definition. The value of a calculated field is computed in the OnCalcFields event handler. The value of an internally calculated field is computed by the SQL server or the Borland (see page 1761) Database Engine

.

See Also

OnCalcFields (see page 7053)

InternalCalcField (see page 7192)

Setting Display and Edit Properties at Design Time

39.1.25.5.15 TField.CanModify Property

Specifies whether a field can be modified.

Pascal

```
property CanModify: Boolean;
```

C++

```
__property Boolean CanModify;
```

Description

Check CanModify to determine if a field's data can be edited. CanModify is true only if the ReadOnly (see page 7177) property is false, and the dataset for this field component allows write access to the corresponding database field.

See Also

DataSet (see page 7157)

ReadOnly (see page 7177)

CanModify (see page 7012)

39.1.25.5.16 TField.ConstraintErrorMessage Property

Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.

Pascal

```
property ConstraintErrorMessage: string;
```

C++

```
__property AnsiString ConstraintErrorMessage;
```

Description

Set ConstraintErrorMessage to specify the message that appears when the user-supplied value for the field does not meet the constraints imposed by the server or specified in the CustomConstraint (see page 7156) property.

See Also

CustomConstraint (see page 7156)

ImportedConstraint (see page 7167)

39.1.25.5.17 TField.CurValue Property

Represents the current value of the field component including changes made by other users of the database.

Pascal

```
property CurValue: Variant;
```

C++

```
__property Variant CurValue;
```

Description

Use CurValue to examine the value of a field when a problem occurs while posting a value to the database using a provider. If the current field value causes a problem, such as a key violation, when posting the value, an event is generated to allow applications to respond to the problem. Provider (see page 18419) components generate an OnUpdateError event. If a provider returns problem records to the client dataset, the client dataset generates an OnReconcileError event. In the OnUpdateError or OnReconcileError event handler, NewValue (see page 7174) is the unposted value that caused the problem, OldValue (see page 7175) is the value that was originally assigned to the field before any edits were made, and

CurValue is the value that is currently assigned to the field. CurValue may differ from OldValue (see page 7175) if another user changed the value of the field after OldValue (see page 7175) was read.

Note: CurValue is only supported when the dataset is a TClientDataSet. In the provider's OnUpdateError event, a temporary client dataset containing fields with a CurValue property is passed to the event handler.

See Also

NewValue (see page 7174)

OldValue (see page 7175)

OnUpdateError (see page 18444)

OnReconcileError (see page 8524)

TClientDataSet (see page 8289)

Reconciling Update Errors

Resolving Update Errors On the Provider

39.1.25.5.18 TField.CustomConstraint Property

Specifies an SQL string the enforces any application-specific constraints on the field's value.

Pascal

```
property CustomConstraint: string;
```

C++

```
__property AnsiString CustomConstraint;
```

Description

Set CustomConstraint to limit the values that the user can enter into a field. CustomConstraint must be a valid SQL search expression such as

```
x > 0 and x < 100
```

The name used to refer to the field's value can be any string that is not a reserved SQL keyword, as long as it is used consistently throughout the constraint expression.

Custom constraints are imposed in addition to any constraints to the field's value that come from the server. To see the constraints imposed by the server, read the ImportedConstraint (see page 7167) property.

Note: When setting the CustomConstraint property, be sure to set the ConstraintErrorMessage (see page 7155) property as well, so that a message can appear when the user enters a value that violates the custom constraint.

Note: Custom constraints are SQL expressions. Because of this, always use a period (.) as a floating-point separator. This may be inconsistent with international Windows

environment settings.

See Also

ConstraintErrorMessage (see page 7155)

ImportedConstraint (see page 7167)

Constraining Data Values

39.1.25.5.19 TField.DataSet Property

Identifies the dataset to which a field component belongs.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Check the value of DataSet to determine the dataset that uses the field component to represent one of its fields. Applications should not directly assign the DataSet property of a field. It is assigned automatically when the field is created from a TFieldDef (see page 7184) object.

See Also

CreateField (see page 7188)

39.1.25.5.20 TField.DataSize Property

Indicates the amount of memory needed to store a field component's value.

Pascal

```
property DataSize: Integer;
```

C++

```
__property int DataSize;
```

Description

Check DataSize to determine the number of bytes required to store a field component's value. Use DataSize to determine the required size of a buffer for working with the field's value in native format. For example, use DataSize to determine the buffer size needed by the GetData (see page 7131) and SetData (see page 7141) methods.

The value of DataSize for TField (see page 7115) is 0. Most descendants of TField (see page 7115) override this property to specify the size required by the specific field type they represent. BLOB fields, however, represent data of indeterminate size. They do not override the inherited method, so the value of DataSize for BLOB fields is 0. A value of zero should not be interpreted as meaning the field value requires no memory. A value of zero indicates that the memory required for the field is indeterminate.

C++ Examples:

```
/*
This example requires a button, a test edit, and a populated
ClientDataSet.  Pipe the ClientDataSet through a DataSource
to a DGGrid or DBNavigator to control the current field.
Cast the data correctly according to the field type when
assigning to the test edit.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // Retrieve the "raw" data from Field1
  if (!CDS->Fields->Fields[0]->IsBlob(__classid(TField))) // GetData does not work for BLOB
fields
  {
    void *MyBuffer = malloc(CDS->Fields->Fields[0]->DataSize);
    try
    {
```

```
      if (!CDS->Fields->Fields[0]->GetData(MyBuffer))
        MessageDlg(CDS->Fields->Fields[0]->DisplayName + "is blank.", mtInformation,
TMsgDlgButtons() << mbOK,0);
      else
        // Do something with the data;
        Edit1->Text = (char *)MyBuffer; // for a stringfield
    }
    __finally
    {
      free(MyBuffer);
    }
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CDS->LoadFromFile("../CDS.XML");
}
```

Delphi Examples:

```
{
This example requires a button, a test edit, and a populated
ClientDataSet.  Pipe the ClientDataSet through a DataSource
to a DGGrid or DBNavigator to control the current field.
Cast the data correctly according to the field type when
assigning to the test edit.
}

{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button1Click(Sender: TObject);
var MyBuffer: Pointer;
begin
{ Retrieve the "raw" data from Field1 }
with CDS.Fields[0] do
  begin
    if not IsBlob then { this does not work for BLOB fields }
    begin
      { Allocate space }
      MyBuffer:= GetMemory(DataSize);
      try
        if not GetData(MyBuffer) then
          MessageDlg(DisplayName + ' is NULL', mtInformation, [mbOK], 0)
        else
          { Do something with the data };
          Edit1.Text:= string(MyBuffer); // for a stringfield
      finally
        { Free the space }
        FreeMem(MyBuffer, DataSize);
      end;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;
```

See Also

DataType (see page 7159)

GetData (see page 7131)

SetData (see page 7141)

39.1.25.5.21 TField.DataType Property

Identifies the data type of the field component.

Pascal

```
property DataType: TFieldType;
```

C++

```
__property TFieldType DataType;
```

Description

Read the DataType property to learn the type of the data the field contains.

See Also

DataSize (see page 7157)

IsValidChar (see page 7136)

39.1.25.5.22 TField.DefaultExpression Property

Specifies an SQL expression that is assigned to the field if the user does not provide a value.

Pascal

```
property DefaultExpression: string;
```

C++

```
__property AnsiString DefaultExpression;
```

Description

Set DefaultExpression to provide a default value for a field. DefaultExpression can be any valid SQL value expression that does not refer to field values. If the expression contains literals other than numeric values, they must appear in quotes. For example, a default value of noon for a time field is

```
'12:00:00'
```

including the quotes around the literal value.

Note: If you supply a value for DefaultExpression, this value takes precedence over any default values for the field implemented on the database server. This is because DefaultExpression is applied to the field in your application before any edits are sent to the server.

See Also

Value (see page 7180)

Constraining Data Values

Setting a Default Value for a Field

39.1.25.5.23 TField.DisplayLabel Property

Contains the text to display in the corresponding column heading of a data grid.

Pascal

```
property DisplayLabel: WideString;
```

C++

```
__property BSTR DisplayLabel;
```

Description

Use DisplayLabel to assign column headings to a data grid. The column headings of the data grid use the DisplayName (see page 7160) property of the field whose value they represent. Setting DisplayLabel changes the read-only DisplayName (see page 7160) property from FieldName (see page 7164) to the string specified as DisplayLabel.

See Also

DisplayName (see page 7160)

FieldName (see page 7164)

TDBGrid (see page 9668)

Creating a Customized Grid

39.1.25.5.24 TField.DisplayName Property

Represents the name of the field for display purposes.

Pascal

```
property DisplayName: WideString;
```

C++

```
__property BSTR DisplayName;
```

Description

Use DisplayName when displaying a string that represents the field to the user. For example, when raising an exception concerning the field, format an error string that includes the DisplayName to refer to the field. DisplayName is a read-only property. To change the value of DisplayName, use the DisplayLabel (see page 7160) property.

If DisplayLabel (see page 7160) is defined, DisplayName is the same as DisplayLabel (see page 7160). Otherwise, DisplayName is the same as the FieldName (see page 7164) property.

C++ Examples:

```
/*
This example requires a button, a test edit, and a populated
ClientDataSet.  Pipe the ClientDataSet through a DataSource
to a DGGrid or DBNavigator to control the current field.
Cast the data correctly according to the field type when
assigning to the test edit.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // Retrieve the "raw" data from Field1
  if (!CDS->Fields->Fields[0]->IsBlob(__classid(TField))) // GetData does not work for BLOB
fields
  {
    void *MyBuffer = malloc(CDS->Fields->Fields[0]->DataSize);
```

```
    try
    {
      if (!CDS->Fields->Fields[0]->GetData(MyBuffer))
        MessageDlg(CDS->Fields->Fields[0]->DisplayName + "is blank.", mtInformation,
TMsgDlgButtons() << mbOK,0);
      else
        // Do something with the data;
        Edit1->Text = (char *)MyBuffer; // for a stringfield
    }
    __finally
    {
      free(MyBuffer);
    }
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CDS->LoadFromFile("../CDS.XML");
}
```

Delphi Examples:

```
{
This example requires a button, a test edit, and a populated
ClientDataSet.  Pipe the ClientDataSet through a DataSource
to a DGGrid or DBNavigator to control the current field.
Cast the data correctly according to the field type when
assigning to the test edit.
}

{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button1Click(Sender: TObject);
var MyBuffer: Pointer;
begin
{ Retrieve the "raw" data from Field1 }
with CDS.Fields[0] do
  begin
    if not IsBlob then { this does not work for BLOB fields }
    begin
      { Allocate space }
      MyBuffer:= GetMemory(DataSize);
      try
        if not GetData(MyBuffer) then
          MessageDlg(DisplayName + ' is NULL', mtInformation, [mbOK], 0)
        else
          { Do something with the data };
          Edit1.Text:= string(MyBuffer); // for a stringfield
      finally
        { Free the space }
        FreeMem(MyBuffer, DataSize);
      end;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;
```

See Also

DisplayLabel (see page 7160)

FieldName (see page 7164)

39.1.25.5.25 TField.DisplayText Property

Represents the field's value as it is displayed in a data-aware control.

Pascal

```
property DisplayText: string;
```

C++

```
__property AnsiString DisplayText;
```

Description

DisplayText is a read-only string representation of a field's value for displaying in data-aware controls. It represents the field's value when it is not being edited. When the field is being edited, use the Text (see page 7179) property.

If the field has an OnGetText (see page 7182) event handler, DisplayText is the value returned in the Text (see page 7179) parameter of the OnGetText (see page 7182) event handler when its DisplayText parameter is true. Otherwise, DisplayText is the value of the AsString (see page 7151) property.

See Also

AsString (see page 7151)

OnGetText (see page 7182)

Text (see page 7179)

Value (see page 7180)

39.1.25.5.26 TField.DisplayWidth Property

Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set DisplayWidth to control how many characters are used to calculate the column width for the field when it is displayed by a TDBLookupListBox object or a descendant of TCustomDBGrid. The width of columns is determined by the average character width. If the data-aware control that displays the field is not using a fixed pitch font, DisplayWidth may indicate more or fewer characters than will fit in the column for any given string.

Note: The DisplayWidth property has no effect if there is an explicit Columns value for the field in a TDBGrid object.

See Also

DisplayText (see page 7162)

TCustomDBGrid (see page 9608)

TDBLookupListBox (see page 9070)

Using a Grid Control in Its Default State

Working with Field Components: Overview

39.1.25.5.27 TField.EditMask Property

Contains a mask that restricts the data that can be entered into a data field.

Pascal

```
property EditMask: TEditMask;
```

C++

```
__property TEditMask EditMask;
```

Description

Use EditMask to restrict the characters a user can enter. If the user attempts to enter a character that is not valid, the character is not accepted. Validation using the EditMask property is performed on a character-by-character basis. Use the OnValidate (see page 7183) event instead to validate the entire input, just before it is posted.

Setting EditMask to an empty string removes the mask.

Note: For a description of the format required by EditMask, see the

TEditMask type.

See Also

DisplayText (see page 7162)

IsValidChar (see page 7136)

OnValidate (see page 7183)

Validate (see page 7144)

Value (see page 7180)

39.1.25.5.28 TField.EditMaskPtr Property

Provides read-only access to the EditMask (see page 7163) property.

Pascal

```
property EditMaskPtr: TEditMask;
```

C++

```
__property TEditMask EditMaskPtr;
```

Description

Use EditMaskPtr when reading the EditMask (see page 7163) property. Because EditMaskPtr is a read-only property, EditMaskPtr guards against accidentally changing the value of the mask in code that should only read it.

Note: For a description of the format required by EditMask (see page 7163), see the

TEditMask type.

See Also

EditMask (see page 7163)

39.1.25.5.29 TField.FieldKind Property

Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.

Pascal

```
property FieldKind: TFieldKind;
```

C++

```
__property TFieldKind FieldKind;
```

Description

Use FieldKind to determine if a field is a data field, a calculated field, a lookup field, or an aggregate field. The value of FieldKind can be changed programmatically, but in practice FieldKind is set at design time when creating Field components with the Fields editor.

Note: Fields calculated by SQL servers or the Borland (see page 1761) Database Engine to display the results of a query that returns a live dataset

have a FieldKind of fkInternalCalc, not fkCalculated. This is because the field values are stored in the dataset. Calculated (see page 7154) fields in a client dataset that are calculated in an OnCalcFields event handler but stored in the dataset also have a FieldKind of fkInternalCalc instead of fkCalculated. Unlike regular calculated fields, internally calculated fields can be used in filter expressions. They can be edited, but the changes are discarded. To prevent editing, set the ReadOnly (see page 7177) property to true.

Note: The FieldKind property of a field is only stored with the field object if the value is fkInternalCalc.

See Also

Calculated (see page 7154)

Lookup (see page 7170)

ReadOnly (see page 7177)

39.1.25.5.30 TField.FieldName Property

Indicates the name of the physical column in the underlying table or query result to which a field component is bound.

Pascal

```
property FieldName: WideString;
```

C++

```
__property BSTR FieldName;
```

Description

When creating a field component, the TFieldDef (see page 7184) object uses FieldName to specify which field in the underlying dataset the field component represents. FieldName is used when displaying references to the field to users, unless a DisplayLabel (see page 7160) has been set. For calculated fields, supply a FieldName when defining the field. For non-calculated fields, an EDatabaseError (see page 6731) exception is raised if FieldName is not a column name in the underlying table.

Delphi Examples:

```
{
This example displays a message box with the names of all
fields in a table.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
   i: Integer;
   Info: String;
begin
   Info := 'The fields of table ' + Table1.TableName +
           ' are:'#13#10#13#10;
   for i := 0 to Table1.FieldCount - 1 do
      Info := Info + Table1.Fields[i].FieldName + #13#10;
  ShowMessage(Info);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if not Table1.Exists then
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
    end;
  end;
```

```
  DS2.DataSet:= Table1;
  DBGrid2.DataSource.DataSet:= Table1;
  Table1.Active:= True;
end;
```

See Also

DisplayLabel (see page 7160)

DisplayName (see page 7160)

EDatabaseError (see page 6731)

Origin (see page 7176)

CreateField (see page 7188)

FullName (see page 7166)

Defining New Persistent Fields

39.1.25.5.31 TField.FieldNo Property

Indicates the ordinal position of the field's column in the underlying table or query result.

Pascal

```
property FieldNo: Integer;
```

C++

```
__property int FieldNo;
```

Description

Read FieldNo to learn a field component's ordinal position in the table underlying the dataset, where 1 is the first field. Use the FieldNo property to specify the field only in direct API calls to the Borland (see page 1761) Database Engine.

Note: Do not confuse FieldNo, which is the field component's ordinal position in the physical database table, with Index (see page 7168), the field component's position in the dataset's Fields property.

See Also

FieldName (see page 7164)

Index (see page 7168)

39.1.25.5.32 TField.FullName Property

Indicates the field name qualified by the full name of the parent field, if any.

Pascal

```
property FullName: WideString;
```

C++

```
__property BSTR FullName;
```

Description

Inspect FullName to determine the field's name when it may be a subfield in the flattened version of a more complex object field. FullName prefixes the FieldName (see page 7164) property with the full name of the more complex object field of which it is a part.

See Also

FieldName (see page 7164)

ParentField (see page 7176)

39.1.25.5.33 TField.HasConstraints Property

Indicates whether field values are constrained.

Pascal

```
property HasConstraints: Boolean;
```

C++

```
__property Boolean HasConstraints;
```

Description

Read HasConstraints to determine whether the field component imposes custom constraints, imported constraints, or a default expression. HasConstraints is true if the CustomConstraint (see page 7156), DefaultExpression (see page 7159), or ImportedConstraint (see page 7167) property is set. Otherwise, HasConstraints is false, indicating that no constraints on field values are implemented.

Field values can still be limited when HasConstraints is false. For example, an edit mask or OnValidate (see page 7183) event handler can constrain field values.

See Also

CustomConstraint (see page 7156)

DefaultExpression (see page 7159)

EditMask (see page 7163)

ImportedConstraint (see page 7167)

OnValidate (see page 7183)

39.1.25.5.34 TField.ImportedConstraint Property

Represents an SQL clause that enforces constraints stored on the server.

Pascal

```
property ImportedConstraint: string;
```

C++

```
__property AnsiString ImportedConstraint;
```

Description

Read ImportedConstraint to determine the limits imposed by the server on what values are valid for the field. ImportedConstraint is an SQL search expression such as

```
Value > 0 and Value < 100
```

Do not change the value of ImportedConstraint. To add additional constraints on the field value, use the CustomConstraint (see page 7156) property instead. Custom constraints are imposed in addition to the imported constraints. If the server constraints change, the value of ImportedConstraint will be changed but constraints introduced in the CustomConstraint (see page 7156) property will persist.

Removing constraints from the ImportedConstraint property will not change the validity of field values that violate those

constraints. Removing constraints results in the constraints being checked by the server instead of locally. When constraints are checked locally, the error message supplied as the ConstraintErrorMessage (see page 7155) property is displayed when violations are posted, instead of an error message from the server when updates are applied.

Note: On Windows, ImportedConstraint is only set for fields of datasets that import constraints. These include client datasets (when they receive imported constraints in data packets) and BDE-based datasets.

Note: On Linux, the IDE does not provide any mechanism for importing server constraints. However, client datasets can receive imported constraints in data packets from an application server running on another platform.

See Also

ConstraintErrorMessage (see page 7155)

CustomConstraint (see page 7156)

39.1.25.5.35 TField.Index Property

Specifies the field component's index in the Fields property of the dataset.

Pascal

```
property Index: Integer;
```

C++

```
__property int Index;
```

Description

Use the Index property to:

Find the position of the field in the dataset.

Change (see page 7126) the order of a field's position in the dataset by changing the value of Index. Changing the Index value affects the order in which fields are displayed in data grids, but not the position of the fields in physical database tables.

Note: The field order can also be changed by dragging and dropping fields in the Fields editor at design time.

See Also

FieldNo (see page 7166)

Fields (see page 7023)

TDBGrid (see page 9668)

39.1.25.5.36 TField.IsIndexField Property

Indicates whether a field is an index field.

Pascal

```
property IsIndexField: Boolean;
```

C++

```
__property Boolean IsIndexField;
```

Description

Inspect the IsIndexField to determine if a field is used in the dataset's current index. If IsIndexField is true, the field is one of the fields that makes up the dataset's current index. If IsIndexField is false, the field is not an index field. A field component must be

an index field to serve as one of the fields in the LookupKeyFields (see page 7172) property of another field component.

See Also

LookupKeyFields (see page 7172)

39.1.25.5.37 TField.IsNull Property

Indicates whether the field has a value assigned to it.

Pascal

```
property IsNull: Boolean;
```

C++

```
__property Boolean IsNull;
```

Description

Inspect IsNull to ascertain whether the field contains a value. If IsNull is true, the field is blank. If IsNull is false, the field has a value.

See Also

Clear (see page 7126)

39.1.25.5.38 TField.KeyFields Property

Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.

Pascal

```
property KeyFields: WideString;
```

C++

```
__property BSTR KeyFields;
```

Description

Set KeyFields only on lookup fields (fields where FieldKind (see page 7164) is fkLookup and Lookup (see page 7170) is true). KeyFields specifies the field or fields to match when a lookup occurs. To use more than one field, separate each field name with a semicolon. The specified fields must be of the same types as the corresponding fields in LookupKeyFields (see page 7172), or the lookup fails.

The values of the KeyFields in the DataSet (see page 7157) are matched with the values of the LookupKeyFields (see page 7172) in the LookupDataSet (see page 7171) to determine a particular record of the lookup table. The value of the LookupResultField (see page 7173) on that record becomes the Value (see page 7180) property for the field component.

See Also

DataSet (see page 7157)

FieldKind (see page 7164)

Lookup (see page 7170)

LookupDataSet (see page 7171)

LookupKeyFields (see page 7172)

LookupResultField (see page 7173)

Value (see page 7180)

Defining a Lookup Field

39.1.25.5.39 TField.Lookup Property

Determines whether the field is specified as a lookup field.

Pascal

```
property Lookup: Boolean;
```

C++

```
__property Boolean Lookup;
```

Description

Lookup is provided for backward compatibility. To determine if a field is a lookup field, use the FieldKind (see page 7164) property instead Lookup indicates whether FieldKind (see page 7164) is fkLookup. When FieldKind (see page 7164) is fkLookup, Lookup is true. When FieldKind (see page 7164) is any other value, Lookup is false. Changing Lookup to false implicitly sets FieldKind (see page 7164) to fkData. Note that lookup fields do not work if the field is in a unidirectional dataset.

If the field is a lookup field, the KeyFields (see page 7169) property indicates which fields in the dataset must match the LookupKeyFields (see page 7172) in the LookupDataSet (see page 7171), to identify a record in the LookupDataSet (see page 7171). The value of the LookupResultField (see page 7173) for that record becomes the Value (see page 7180) of the field component.

See Also

FieldKind (see page 7164)

KeyFields (see page 7169)

LookupCache (see page 7170)

LookupDataSet (see page 7171)

LookupKeyFields (see page 7172)

LookupResultField (see page 7173)

Value (see page 7180)

39.1.25.5.40 TField.LookupCache Property

Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.

Pascal

```
property LookupCache: Boolean;
```

C++

```
__property Boolean LookupCache;
```

Description

Set LookupCache to true to cache the values of a lookup field when the LookupDataSet (see page 7171) is unlikely to change and the number of distinct lookup values is small. Caching lookup values can speed performance, because the lookup values for every set of LookupKeyFields (see page 7172) values are preloaded when the DataSet (see page 7157) is opened. When the current record in the DataSet (see page 7157) changes, the field object can locate its Value (see page 7180) in the cache, rather than accessing the LookupDataSet (see page 7171). This performance improvement is especially dramatic if the LookupDataSet (see page 7171) is on a network where access is slow.

If every record of DataSet (see page 7157) has different values for KeyFields (see page 7169), the overhead of locating values in the cache can be greater than any performance benefit provided by the cache. The overhead of locating values in the cache increases with the number of distinct values that can be taken by KeyFields (see page 7169).

If LookupDataSet (see page 7171) is volatile, caching lookup values can lead to inaccurate results. Call RefreshLookupList (see page 7138) to update the values in the lookup cache. RefreshLookupList (see page 7138) regenerates the LookupList (see page 7172) property, which contains the value of the LookupResultField (see page 7173) for every set of LookupKeyFields (see page 7172) values.

When setting LookupCache at runtime, call RefreshLookupList (see page 7138) to initialize the cache.

Note: LookupCache is only meaningful if the value of FieldKind (see page 7164) is fkLookup.

See Also

DataSet (see page 7157)

FieldKind (see page 7164)

KeyFields (see page 7169)

LookupDataSet (see page 7171)

LookupKeyFields (see page 7172)

LookupList (see page 7172)

LookupResultField (see page 7173)

RefreshLookupList (see page 7138)

Defining a Lookup Field

39.1.25.5.41 TField.LookupDataSet Property

Identifies the dataset used to look up field values.

Pascal

```
property LookupDataSet: TDataSet;
```

C++

```
__property TDataSet LookupDataSet;
```

Description

Use LookupDataSet to specify the dataset to use for looking up field values in a field with a FieldKind (see page 7164) of fkLookup.

Note: LookupDataSet can't refer to a unidirectional dataset.

When a field component is a lookup field, the Value (see page 7180) of the field component is the value of the LookupResultField (see page 7173) in the LookupDataSet. The record in the LookupDataSet that contains the correct value is found by matching the LookupKeyFields (see page 7172) in the LookupDataSet with the current values of the KeyFields (see page 7169) in the DataSet (see page 7157).

See Also

FieldKind (see page 7164)

KeyFields (see page 7169)

LookupKeyFields (see page 7172)

LookupResultField (see page 7173)

Value (see page 7180)

DataSet (see page 7157)

Defining a Lookup Field

39.1.25.5.42 TField.LookupKeyFields Property

Identifies the field or fields in the lookup dataset to match when doing a lookup.

Pascal

```
property LookupKeyFields: WideString;
```

C++

```
__property BSTR LookupKeyFields;
```

Description

Only use LookupKeyFields on field components with a FieldKind (see page 7164) of fkLookup. LookupKeyFields are the fields in the index of the LookupDataSet (see page 7171) whose values must match the KeyFields (see page 7169) in the field component's DataSet (see page 7157).

To specify more than one field, separate each field name with a semicolon. The fields specified in LookupKeyFields must be of the same types as the corresponding fields in KeyFields (see page 7169), or the lookup can't work.

Matching the values of the LookupKeyFields in the LookupDataSet (see page 7171) with the values of the KeyFields (see page 7169) in the DataSet (see page 7157) determines a specific record in the LookupDataSet (see page 7171). The Value (see page 7180) of the field component is the value of the LookupResultField (see page 7173) in the indicated record.

See Also

FieldKind (see page 7164)

KeyFields (see page 7169)

Lookup (see page 7170)

LookupDataSet (see page 7171)

LookupResultField (see page 7173)

Value (see page 7180)

DataSet (see page 7157)

Defining a Lookup Field

39.1.25.5.43 TField.LookupList Property

Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.

Pascal

```
property LookupList: TLookupList;
```

C++

```
__property TLookupList LookupList;
```

Description

Inspect LookupList to find the cached value assigned to the field for a given set of values on the fields specified by KeyFields (see page 7169). The value of LookupList is set when the dataset is open or the RefreshLookupList (see page 7138) method is called.

Tip: Applications can also fill LookupList programmatically to provide lookup values as long as the LookupDataSet (see page 7171) property is not set.

The ValueOfKey method of the LookupList returns the value of the LookupResultField (see page 7173) for a set of values on the LookupKeyFields (see page 7172). This allows the value of the lookup field object to be computed as follows:

```
Value := LookupList.ValueOfKey(DataSet.FieldValues[KeyFields]);
Value = LookupList->ValueOfKey(DataSet->FieldValues[KeyFields]);
```

Note: LookupList is only meaningful when the value of FieldKind (see page 7164) is fklookup and the LookupCache (see page 7170) property is true.

See Also

DataSet (see page 7157)

FieldKind (see page 7164)

KeyFields (see page 7169)

LookupCache (see page 7170)

LookupDataSet (see page 7171)

LookupKeyFields (see page 7172)

LookupResultField (see page 7173)

FieldValues (see page 7025)

ValueOfKey (see page 7339)

39.1.25.5.44 TField.LookupResultField Property

Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.

Pascal

```
property LookupResultField: WideString;
```

C++

```
__property BSTR LookupResultField;
```

Description

Use LookupResultField to specify a single field in the LookupDataSet (see page 7171) whose value becomes the value of the field component. The field component must have a FieldKind (see page 7164) of fkLookup, and the DataType (see page 7159) of the field component must be the same as the data type of LookupResultField.

When a field component is a lookup field, the Value (see page 7180) of the field component is the value of the LookupResultField in the LookupDataSet (see page 7171). The record in the LookupDataSet (see page 7171) that contains the correct value is found by matching the LookupKeyFields (see page 7172) in the LookupDataSet (see page 7171) with the current values of the KeyFields (see page 7169) in the DataSet (see page 7157).

See Also

DataSet (see page 7157)

DataType (see page 7159)

FieldKind (see page 7164)

KeyFields (see page 7169)

LookupDataSet (see page 7171)

LookupKeyFields (see page 7172)

Value (see page 7180)

Defining a Lookup Field

39.1.25.5.45 TField.NewValue Property

Represents the current value of the field component including pending cached updates.

Pascal

```
property NewValue: Variant;
```

C++

```
__property Variant NewValue;
```

Description

Use NewValue to examine or change the current value of a field when in the process of applying multiple updates. If the current field value is causing a problem, such as a key violation, when applying updates, datasets generate an OnUpdateError event. Similarly, provider components generate an OnUpdateError event when problems occur posting records from a client, and client datasets generate an OnReconcileError event when informed of problems by the provider. In the event handler, assign a new value to NewValue to correct the problem.

NewValue is the same as Value (see page 7180), except when errors are encountered while posting records. Setting NewValue in an OnUpdateError event handler, an OnUpdateRecord event handler, or an OnReconcileError event handler causes NewValue to differ from Value (see page 7180) until the records have finished being applied to the underlying database table.

Note: The NewValue property is only usable when the data is accessed using a TClientDataSet component or cached updates is enabled

.

See Also

CurValue (see page 7155)

OldValue (see page 7175)

OnReconcileError (see page 8524)

CachedUpdates (see page 9932)

OnUpdateError (see page 9956)

OnUpdateRecord (see page 9957)

TClientDataSet (see page 8289)

Reconciling Update Errors

Resolving Update Errors On the Provider

39.1.25.5.46 TField.Offset Property

Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).

Pascal

```
property Offset: Integer;
```

C++

```
__property int Offset;
```

Description

Read Offset to determine the extra space in the record buffer of a TClientDataSet object allocated for cached information on calculated fields or BLOB fields. Offset should only be used by component writers when developing new descendants of TClientDataSet.

See Also

Calculated (see page 7154)

IsBlob (see page 7136)

TClientDataSet (see page 8289)

39.1.25.5.47 TField.OldValue Property

Represents the original value of the field (as a Variant).

Pascal

```
property OldValue: Variant;
```

C++

```
__property Variant OldValue;
```

Description

Read the OldValue property to examine or retrieve the original value of the field that was obtained from the dataset before any edits were posted. For example, in Delphi the following line replaces current pending changes with a field's original value:

```
NewValue :=OldValue;
```

Once records are applied successfully to the database, the old field value cannot be retrieved.

Note: the OldValue property is only usable when the data is accessed using a TClientDataSet component or cached updates is enabled

.

See Also

CurValue (see page 7155)

NewValue (see page 7174)

CachedUpdates (see page 9932)

TClientDataSet (see page 8289)

OnReconcileError (see page 8524)

OnUpdateError (see page 9956)

OnUpdateRecord (see page 9957)

Reconciling Update Errors

Resolving Update Errors On the Provider

39.1.25.5.48 TField.Origin Property

Indicates the name of the field in its original database table.

Pascal

```
property Origin: string;
```

C++

```
__property AnsiString Origin;
```

Description

Origin is only assigned at design time by the Fields editor, and only when the field component is used by a TQuery object. The Origin property distinguishes the name of the field as it appears in its dataset from the name of the field in the base table on which it is based. For example, in a query that uses the following SQL statement, the value of Origin is CUSTOMER.CUSTNO.

```
SELECT CUSTNO AS ID FROM CUSTOMER
```

In this case, the FieldName (see page 7164) property is ID.

Note: Origin is not available on Linux. The Origin property is only implemented for BDE-enabled datasets, which appear only in the Windows product.

See Also

FieldName (see page 7164)

39.1.25.5.49 TField.ParentField Property

Specifies the TObjectField (see page 7372) object from which this field is derived.

Pascal

```
property ParentField: TObjectField;
```

C++

```
__property TObjectField ParentField;
```

Description

Use ParentField when the field represents a subfield in the flattened version of a more complex object field. ParentField represents the object field (an Oracle8 array field or ADT).

When ParentField is nil (Delphi) or NULL (C++), the field does not represent part of a composite field.

See Also

ObjectView (see page 7031)

39.1.25.5.50 TField.ProviderFlags Property

Specifies how a provider uses the field value when applying updates.

Pascal

```
property ProviderFlags: TProviderFlags;
```

C++

```
__property TProviderFlags ProviderFlags;
```

Description

ProviderFlags is only relevant for fields that appear in the data packets supplied to a provider component's event handlers. This property is a set of flags that determine how the field is used when applying updates or resolving update errors.

See Also

TDataSetProvider (see page 18508)

UpdateMode (see page 10028)

UpdateMode (see page 18441)

CurValue (see page 7155)

OldValue (see page 7175)

NewValue (see page 7174)

Influencing How Updates Are Applied

Specifying What Fields Appear in Data Packets

39.1.25.5.51 TField.ReadOnly Property

Determines whether the field can be modified.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set ReadOnly to true to prevent a field from being modified. In a data grid, tabbing from field to field skips over ReadOnly fields.

To learn whether a field can be modified, use the CanModify (see page 7154) property instead. CanModify (see page 7154) reflects not only the value of the ReadOnly property, but also any restrictions imposed by the underlying dataset.

See Also

CanModify (see page 7154)

39.1.25.5.52 TField.Required Property

Specifies whether a nonblank value for a field is required.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;
```

Description

Use Required to find out if a field requires a value or if the field can be blank.

If a field is created with the Fields editor, this property is set based on the underlying table. Applications that set Required to true

for fields that must have values (for example, a password or part number), but for which the underlying table does not require the field, must write an OnValidate (see page 7183) event handler to enforce the property.

When the Required property reflects a property of the underlying database table, trying to post apply a null value causes an exception to be raised. Applications that set the Required property to true when the underlying table does not require the field, should raise an EDatabaseError (see page 6731) exception on null values in the OnValidate (see page 7183) event handler in order to achieve the same result.

See Also

OnValidate (see page 7183)

Validate (see page 7144)

39.1.25.5.53 TField.Size Property

Indicates the size used in the definition of the physical database field for data types that support different sizes.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

The interpretation of Size depends on the data type. The meaning of Size for data types that use it is given in the following table. For all other data types, Size is not used and its value is always 0.

DataType (see page 7159)	Meaning of Size
ftBCD	Size is the number of digits after the decimal place.
ftString	Size is the maximum number of characters in the string.
ftVarBytes	Size is the maximum number of bytes of data, not counting the first two bytes which indicate the actual number of bytes of data that were used.
ftBytes	Size is the maximum number of bytes of data.
ftBlob	Size is the number of bytes from the BLOB that are stored in the record buffer.
ftDBaseOle	Size is the number of bytes from the dBASE OLE BLOB that are stored in the record buffer.
ftFMTBCD	Size is the number of digits after the decimal place.
ftFmtMemo	Size is the number of bytes from the memo that are stored in the record buffer.
ftGraphic	Size is the number of bytes from the image that are stored in the record buffer.
ftMemo	Size is the number of characters from the memo that are stored in the record buffer.
ftParadoxOle	Size is the number of bytes from the Paradox OLE BLOB that are stored in the record buffer.
ftTypedBinary	Size is the number of bytes from the BLOB that are stored in the record buffer.
ftADT	Size is the total number of fields contained in the ADT field, including the fields of child fields that are of type ADT.
ftArray	Size is the number of elements in the array.

See Also

DataSize (see page 7157)

DataType (see page 7159)

39.1.25.5.54 TField.Text Property

Contains the string to display in a data-aware control when the field is in edit mode.

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

Data-aware controls rely on the Text property to provide the editing format for each field. For example, by default the Text property of a currency field omits the thousands separators and currency symbol.

Text can differ from the DisplayText (see page 7162) property if the field uses a different string representation when the value is being edited. To implement two different string representations of a field's value, use the OnGetText (see page 7182) event handler. If an OnGetText (see page 7182) event handler is assigned, Text is the value returned in the Text parameter of the event handler when the DisplayText (see page 7162) parameter is false. If there is no OnGetText (see page 7182) event handler, Text is the value of the AsString (see page 7151) property.

See Also

AsString (see page 7151)

DisplayText (see page 7162)

OnGetText (see page 7182)

Value (see page 7180)

39.1.25.5.55 TField.ValidChars Property

Specifies the characters that can be included in the text representation of the field's value.

Pascal

```
property ValidChars: TFieldChars;
```

C++

```
__property TFieldChars ValidChars;
```

Description

Set ValidChars to restrict the characters that can be validly entered as part of the text representation of the field's value. ValidChars is used by the IsValidChar (see page 7136) method to determine whether a character typed into a data-aware edit control can make up part of the field's value. Unlike the EditMask (see page 7163) property which specifies which characters can appear in specific positions, ValidChars limits the characters that can appear anywhere in the field's value.

TField (see page 7115) initializes ValidChars to include all characters. Numeric descendants of TField (see page 7115) change the default value to include only the characters that can make up a numeric value. Applications can change this value to restrict the character set for a field in other ways or to allow characters that are not permitted by default.

See Also

EditMask (see page 7163)

IsValidChar (see page 7136)

39.1.25.5.56 TField.Value Property

Represents the data in a field component.

Pascal

```
property Value: Variant;
```

C++

```
__property Variant Value;
```

Description

Use Value to read data directly from and write data directly to a field component at runtime. For example, use the Value property to affect the contents of a field that is not Visible (see page 7181).

Delphi Examples:

```
{
This example uses a button to copy the value of a field in
the previous record into the corresponding field in the
current record.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   SavePlace: TBookmark;
   PrevValue: Variant;
begin
   with Customers do
   begin
    { get a bookmark so that we can return to the same record }
    SavePlace := GetBookmark;
    try
      { move to prior record}
      FindPrior;
      { get the value }
      PrevValue := Fields[1].Value;
      {Move back to the bookmark
      this may not be the next record anymore
      if something else is changing the dataset asynchronously }
      GotoBookmark(SavePlace);
      { Set the value }
      Edit;
      Fields[1].Value := PrevValue;
      { Free the bookmark }
    finally
      FreeBookmark(SavePlace);
    end;
  end;
end;

{
To ensure that the button is disabled when there is no
previous record, the OnDataChange event of the DataSource
detects when the user moves to the beginning of file (BOF
property becomes true), and disables the button.  Detection
occurs on scrolling and editing, not selection with the mouse.
}
procedure TForm1.DS2DataChange(Sender: TObject; Field: TField);
begin
  if Customers.Bof then
    Button1.Enabled := False
  else
    Button1.Enabled := True;
end;
```

See Also

Text (see page 7179)

DisplayText (see page 7162)

NewValue (see page 7174)

Visible (see page 7181)

39.1.25.5.57 TField.Visible Property

Determines whether the field appears in a data grid.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

To hide a field in a data grid, set Visible to false. If Visible is true, the field appears in the grid.

Note: The Visible property has no effect if the grid has an explicit Columns value for the field. It also has no effect on controls other than TDBGrid, such as TDBEdit.

See Also

Columns (see page 9653)

39.1.25.6 TField Events

The events of the TField class are listed here.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

39.1.25.6.1 TField.OnChange Event

Occurs immediately after the data in the field is written to the record buffer.

Pascal

```
property OnChange: TFieldNotifyEvent;
```

C++

```
__property TFieldNotifyEvent OnChange;
```

Description

Write an OnChange event handler to respond to any changes in the field's data. OnChange allows a response once the data has been successfully written to the record buffer.

When writing the value of a field to the current record buffer, the following steps occur:

1The OnValidate (see page 7183) event handler is called to validate the data.

2If the OnValidate (see page 7183) event handler accepts the current Value (see page 7180), the data is written to the record buffer.

3If writing the data does not raise an exception, the OnChange event handler is called to allow a response to the change.

See Also

OnValidate (see page 7183)

AfterPost (see page 7046)

Value (see page 7180)

39.1.25.6.2 TField.OnGetText Event

Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.

Pascal

```
property OnGetText: TFieldGetTextEvent;
```

C++

```
__property TFieldGetTextEvent OnGetText;
```

Description

Write an OnGetText handler to perform custom processing for the DisplayText (see page 7162) and Text (see page 7179) properties. Use OnGetText to format the Value (see page 7180) of the field differently when it must be edited from the format used when simply displaying the value, or use OnGetText to display something other than the field's value when it is displayed.

If there is no OnGetText handler, the value of the DisplayText (see page 7162) and Text (see page 7179) properties is simply the value of the AsString (see page 7151) property.

See Also

AsString (see page 7151)

DisplayText (see page 7162)

OnSetText (see page 7182)

Text (see page 7179)

Value (see page 7180)

Handling Events

Using Default Formatting for Numeric

39.1.25.6.3 TField.OnSetText Event

Occurs when the Text (see page 7179) property is assigned a value.

Pascal

```
property OnSetText: TFieldSetTextEvent;
```

C++

```
__property TFieldSetTextEvent OnSetText;
```

Description

Write an OnSetText event handler to specify processing that occurs whenever Text (see page 7179) receives a new value. If there is an OnGetText (see page 7182) event handler that formats the Value (see page 7180) of the field in some special way for editing purposes, OnSetText should reverse the process when parsing the edited string.

If there is no OnSetText handler, the Text (see page 7179) property is set by setting the AsString (see page 7151) property.

See Also

AsString (see page 7151)

OnGetText (see page 7182)

Text (see page 7179)

Value (see page 7180)

Handling Events

Using Default Formatting for Numeric

39.1.25.6.4 TField.OnValidate Event

Occurs just before the data is written to the record buffer.

Pascal

```
property OnValidate: TFieldNotifyEvent;
```

C++

```
__property TFieldNotifyEvent OnValidate;
```

Description

Write an OnValidate event handler to validate changes made to the data in the field, just before the data is written to the current record buffer. The EditMask (see page 7163) property allows validation of the data on a character by character basis while it is being entered by the user. OnValidate allows an application to validate the data as a whole.

When the value of a field component is assigned programmatically, validation by the EditMask (see page 7163) is bypassed, since there is no data-aware control to enforce adherence to the mask. OnValidate allows an application to validate such data before it is posted to the database table.

To reject the current value of the field from the OnValidate event handler, raise an exception.

When writing the value of a field to the current record buffer, the following steps occur:

1The OnValidate event handler is called to validate the data.

2If the OnValidate event handler does not raise an exception, the data is written to the current record buffer.

3If writing the data does not raise an exception, the OnChange (see page 7181) event handler is called to allow a response to the change.

See Also

EditMask (see page 7163)

IsValidChar (see page 7136)

OnChange (see page 7181)

BeforePost (see page 7052)

Validate (see page 7144)

Value (see page 7180)

Handling Events

39.1.26 TFieldDef Class

TFieldDef is a field definition that corresponds to a physical field of a record in a table underlying a dataset.

Class Hierarchy

Pascal

```
TFieldDef = class(TNamedItem);
```

C++

```
class TFieldDef : public TNamedItem;
```

File

DB

Description

A TFieldDef object contains the definition of one field in a table. The definition for a field includes such attributes as the field's name, data type, and size. TFieldDef objects are typically used in collections of such objects, such as the FieldDefs property of the TDataSet (see page 6894) component.

There are two primary reasons for working with TFieldDef objects:

To obtain information about field types in a dataset without opening the dataset.

To specify field definitions for a new table.

When using an existing table, a field definition is automatically created for each field in the dataset that comes from the underlying database. Inspect the properties of TFieldDef to retrieve information about specific fields in the dataset.

When creating new tables, such as with the CreateTable method of TTableor the CreateDataSet method of TClientDataSet, TFieldDef objects supply the definitions for the new fields that will comprise the new table.

A field definition has a corresponding TField (see page 7115) object, but not all TField (see page 7115) objects have a corresponding field definition. For example, calculated fields do not have field definition objects.

See Also

TFieldDefs (see page 7198)

TFieldDefList (see page 7195)

TDataSet (see page 6894)

TTable (see page 10168)

TCustomClientDataSet (see page 8373)

Creating and Deleting Tables

39.1.26.1 TFieldDef Members

The following tables list the members exposed by TFieldDef.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 7189)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
	GetDisplayName (see page 7359)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetDisplayName (see page 7359)	This is SetDisplayName, a member of class TNamedItem.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AddChild (see page 7187)	Adds a new TFieldDef (see page 7184) object to the ChildDefs (see page 7191) array.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 7188)	Copies the properties of one field definition to another.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 7186)	Creates and initializes a TFieldDef (see page 7184) instance.
	CreateField (see page 7188)	Creates a TField (see page 7115) component that matches the specifications of the TFieldDef (see page 7184) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 7187)	Destroys the TFieldDef (see page 7184) instance and frees its memory.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	HasChildDefs (see page 7189)	Indicates whether the field of the TFieldDef (see page 7184) object has any child field definitions.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	FieldClass (see page 7192)	Determines the class of the field object that corresponds to this field definition object.
	FieldNo (see page 7192)	Identifies the physical field number used to reference the field.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
	InternalCalcField (see page 7192)	Determines if the field is calculated by the underlying database.
R	ParentDef (see page 7193)	The ParentDef property references the parent field definition for the field definition, if it exists.
	Required (see page 7194)	Indicates whether a value for the physical field in the underlying table is required.

Published Properties

	Name	Description
	Attributes (see page 7190)	Identifies attributes of the field associated with the field def.
	ChildDefs (see page 7191)	Specifies the TFieldDefs (see page 7198) object that contains an array of child fields.
	DataType (see page 7191)	Determines the type of a physical field.
	Name (see page 7359)	Specifies the name of the database entity defined by the TNamedItem (see page 7357) object.
	Precision (see page 7193)	Specifies the number of digits of accuracy used to store the field value.
	Size (see page 7194)	Reports the size associated with the field in the physical database table.

39.1.26.2 TFieldDef.Create Constructor

Creates and initializes a TFieldDef (see page 7184) instance.

Pascal

```
constructor Create(Owner: TFieldDefs; const Name: WideString; DataType: TFieldType; Size:
Integer; Required: Boolean; FieldNo: Integer); overload; reintroduce;
```

C++

```
__fastcall TFieldDef(TFieldDefs Owner, const BSTR Name, TFieldType DataType, int Size, Boolean
Required, int FieldNo);
```

Description

Applications should not call Create directly. A TFieldDef (see page 7184) is created automatically by the dataset for each field in an already existing table. When composing field definitions preparatory to creating a new dataset, the TFieldDefs.AddFieldDef method (or the older TFieldDefs.Add method) creates the needed instance of TFieldDef (see page 7184).

See Also

CreateField (see page 7188)

Add (see page 7202)

AddFieldDef (see page 7202)

CreateTable (see page 10189)

CreateDataSet (see page 8401)

39.1.26.3 TFieldDef.Destroy Destructor

Destroys the TFieldDef (see page 7184) instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TFieldDef();
```

Description

Destroy calls the inherited Destroy method, and then frees and child field definition objects.

An application will seldom need to destroy a TFieldDef (see page 7184) object using this method. Instead, call the Free method, inherited from TObject.

See Also

Create (see page 7186)

Free (see page 22300)

39.1.26.4 TFieldDef Methods

The methods of the TFieldDef class are listed here.

Protected Methods

	Name	Description
	DefineProperties (see page 7189)	Provides an interface for a method that reads and writes otherwise unpublished data.

Public Methods

	Name	Description
	AddChild (see page 7187)	Adds a new TFieldDef (see page 7184) object to the ChildDefs (see page 7191) array.
	Assign (see page 7188)	Copies the properties of one field definition to another.
	CreateField (see page 7188)	Creates a TField (see page 7115) component that matches the specifications of the TFieldDef (see page 7184) object.
	HasChildDefs (see page 7189)	Indicates whether the field of the TFieldDef (see page 7184) object has any child field definitions.

39.1.26.4.1 TFieldDef.AddChild Method

Adds a new TFieldDef (see page 7184) object to the ChildDefs (see page 7191) array.

Pascal

```
function AddChild: TFieldDef;
```

C++

```
__fastcall TFieldDef AddChild();
```

Description

Use AddChild to add a new field definition for a child field of the TFieldDef (see page 7184) object's field. AddChild returns the new field definition for the subfield.

Typically, applications never need to call AddChild because you can't create tables that include object fields.

See Also

ChildDefs (see page 7191)

ParentDef (see page 7193)

HasChildDefs (see page 7189)

39.1.26.4.2 TFieldDef.Assign Method

Copies the properties of one field definition to another.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy a field definition. Assign checks whether the Source parameter specifies another field definition, and if so, copies the values of the Name (see page 7359), DataType (see page 7191), Size (see page 7194), Precision (see page 7193), and InternalCalcField (see page 7192) properties. Note that the value of FieldNo (see page 7192) is not changed.

If the source is not a field definition, Assign calls the inherited method, which assigns properties from any object that specifies how to copy to a field definition in its AssignTo method.

See Also

DataType (see page 7191)

FieldNo (see page 7192)

Name (see page 7359)

Required (see page 7194)

Size (see page 7194)

InternalCalcField (see page 7192)

Precision (see page 7193)

Assign (see page 2930)

Assign (see page 2930)

AssignTo (see page 2931)

39.1.26.4.3 TFieldDef.CreateField Method

Creates a TField (see page 7115) component that matches the specifications of the TFieldDef (see page 7184) object.

Pascal

```
function CreateField(Owner: TComponent; ParentField: TObjectField = nil; const FieldName:
```

```
WideString = ''; CreateChildren: Boolean = True): TField;
```

C++

```
__fastcall TField CreateField(TComponent * Owner, TObjectField ParentField = nil, const BSTR
FieldName = '', Boolean CreateChildren = True);
```

Description

A dataset object calls CreateField to create the field components from the field definition objects. Applications need never call CreateField directly.

See Also

FieldClass (see page 7192)

TDataSet (see page 6894)

TField (see page 7115)

39.1.26.4.4 TFieldDef.DefineProperties Method

Provides an interface for a method that reads and writes otherwise unpublished data.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

Descendants of TPersistent override DefineProperties to designate a method for storing the object's unpublished data to a stream such as a form file. By default, writing an object to a stream writes the values of all its published properties, and reading the object in reads those values and assigns them to the properties. Objects can also specify methods that read and write data other than published properties by overriding the DefineProperties method.

When overriding DefineProperties, consider including some or all of the following:

A call to the inherited method

Calls to the filer object's DefineProperty method

Calls to the filer object's DefineBinaryProperty method

DefineProperties is virtual, so descendant classes can override it as necessary but are not required to do so.

See Also

DefineProperties (see page 2771)

DefineBinaryProperty (see page 2837)

DefineProperty (see page 2838)

39.1.26.4.5 TFieldDef.HasChildDefs Method

Indicates whether the field of the TFieldDef (see page 7184) object has any child field definitions.

Pascal

```
function HasChildDefs: Boolean;
```

C++

```
__fastcall Boolean HasChildDefs();
```

Description

If the TFieldDef (see page 7184) object defines an object field, such as TADTField (see page 6736) or TArrayField (see page 6760), HasChildDefs returns true.

See Also

ParentDef (see page 7193)

AddChild (see page 7187)

ChildDefs (see page 7191)

39.1.26.5 TFieldDef Properties

The properties of the TFieldDef class are listed here.

Public Properties

	Name	Description
R	FieldClass (see page 7192)	Determines the class of the field object that corresponds to this field definition object.
	FieldNo (see page 7192)	Identifies the physical field number used to reference the field.
	InternalCalcField (see page 7192)	Determines if the field is calculated by the underlying database.
R	ParentDef (see page 7193)	The ParentDef property references the parent field definition for the field definition, if it exists.
	Required (see page 7194)	Indicates whether a value for the physical field in the underlying table is required.

Published Properties

	Name	Description
	Attributes (see page 7190)	Identifies attributes of the field associated with the field def.
	ChildDefs (see page 7191)	Specifies the TFieldDefs (see page 7198) object that contains an array of child fields.
	DataType (see page 7191)	Determines the type of a physical field.
	Precision (see page 7193)	Specifies the number of digits of accuracy used to store the field value.
	Size (see page 7194)	Reports the size associated with the field in the physical database table.

39.1.26.5.1 TFieldDef.Attributes Property

Identifies attributes of the field associated with the field def.

Pascal

```
property Attributes: TFieldAttributes;
```

C++

```
__property TFieldAttributes Attributes;
```

Description

Attributes lists various attributes that are used to describe the role of the field that the TFieldDef (see page 7184) object defines.

Note: In Delphi, faReadonly value has the same name as a constant is in the SysUtils (see page 22709) unit. If both the SysUtils (see page 22709) and the Db units are used in your source files, you must disambiguate by specifying the unit to qualify the use of faReadonly. That is,write Db.faReadonly. In C++, this is not a problem because the constant in the SysUtils (

see page 22709) unit differs in case (faReadOnly).

See Also

UnNamed (see page 7386)

ReadOnly (see page 7177)

Required (see page 7194)

HiddenFields (see page 7206)

39.1.26.5.2 TFieldDef.ChildDefs Property

Specifies the TFieldDefs (see page 7198) object that contains an array of child fields.

Pascal

```
property ChildDefs: TFieldDefs;
```

C++

```
__property TFieldDefs ChildDefs;
```

Description

If the field that the TFieldDef (see page 7184) object represents is an object field such as TADTField (see page 6736) or TArrayField (see page 6760), child fields of the object field are stored in the TFieldDefs (see page 7198) array referenced by the ChildDefs property.

Note: You can't create tables that contain object fields. ChildDefs is only used in field definitions that reflect an existing dataset, not field definitions that are used to create a new dataset.

See Also

AddChild (see page 7187)

ParentDef (see page 7193)

TADTField (see page 6736)

TArrayField (see page 6760)

39.1.26.5.3 TFieldDef.DataType Property

Determines the type of a physical field.

Pascal

```
property DataType: TFieldType;
```

C++

```
__property TFieldType DataType;
```

Description

Use DataType to find out what type of data the field contains. When adding field definitions to a dataset, set DataType to specify what type of field is being defined.

See Also

FieldClass (see page 7192)

Size (see page 7194)

39.1.26.5.4 TFieldDef.FieldClass Property

Determines the class of the field object that corresponds to this field definition object.

Pascal

```
property FieldClass: TFieldClass;
```

C++

```
__property TFieldClass FieldClass;
```

Description

Inspect the read-only FieldClass property to find out the class of field object to which this field definition corresponds. The FieldClass is determined by the DataType (see page 7191) of the field. For example, if the DataType (see page 7191) is ftString, the FieldClass is TStringField (see page 7483) (Delphi) or __classid(TStringField (see page 7483)) (C++).

The field object created from a field definition by the CreateField (see page 7188) method is an instance of the FieldClass.

See Also

CreateField (see page 7188)

DataType (see page 7191)

TField (see page 7115)

ClassName (see page 22290)

39.1.26.5.5 TFieldDef.FieldNo Property

Identifies the physical field number used to reference the field.

Pascal

```
property FieldNo: Integer;
```

C++

```
__property int FieldNo;
```

Description

Use FieldNo to find out where the physical field the field definition references is in the set of fields in the table. For example, if the value of FieldNo is 2, then the field is the second field in the underlying database table.

When adding field definitions to a dataset, set FieldNo to specify the location at which to create the field.

See Also

Name (see page 7359)

FieldNo (see page 7166)

39.1.26.5.6 TFieldDef.InternalCalcField Property

Determines if the field is calculated by the underlying database.

Pascal

```
property InternalCalcField: Boolean;
```

C++

```
__property Boolean InternalCalcField;
```

Description

Read InternalCalcField to find out if the field represents a value provided by the underlying database but is not a field in the physical database table. Internally calculated fields are calculated by SQL servers or the Borland (see page 1761) Database Engine in a live query view and stored in the dataset like data fields.

For example, in the following SQL statement, the dataset includes a field for the calculated value ONHAND * COST.

```
SELECT PartNo, Description, OnHand, Cost, (OnHand * Cost)
FROM PARTS
```

When the TQuery object has its RequestLive property set to true, the field definition for that calculated value will have InternalCalcField set to true.

InternalCalcField is a read-only property.

See Also

RequestLive (see page 10088)

FieldKind (see page 7164)

Using Internally Calculated Fields in Client Datasets

39.1.26.5.7 TFieldDef.ParentDef Property

The ParentDef property references the parent field definition for the field definition, if it exists.

Pascal

```
property ParentDef: TFieldDef;
```

C++

```
__property TFieldDef ParentDef;
```

Description

If the TFieldDef (see page 7184) object is the field definition of the child field of an object field, such as an ADT field or an array field, then ParentDef references the parent object's field definition. If the TFieldDef (see page 7184) object does not define the child field of an object field, ParentDef is nil (Delphi) or NULL (C++)..

Note: You can't create tables that contain object fields. ParentDef is only used in field definitions that reflect an existing dataset, not field definitions that are used to create a new dataset.

See Also

ParentField (see page 7176)

AddChild (see page 7187)

HasChildDefs (see page 7189)

39.1.26.5.8 TFieldDef.Precision Property

Specifies the number of digits of accuracy used to store the field value.

Pascal

```
property Precision: Integer;
```

C++

```
__property int Precision;
```

Description

When defining a BCD field, set Precision to the total number of digits stored for each field value. The value of Precision is ignored unless DataType (see page 7191) is ftBCD.

See Also

DataType (see page 7191)

Size (see page 7194)

39.1.26.5.9 TFieldDef.Required Property

Indicates whether a value for the physical field in the underlying table is required.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;
```

Description

Use Required to find out if a field in a dataset must have a value. If Required is true, the field must have a value. If Required is false, the field can be left blank.

When adding field definitions to a dataset, set Required to specify whether blank values for the new field should be prohibited.

See Also

Required (see page 7177)

39.1.26.5.10 TFieldDef.Size Property

Reports the size associated with the field in the physical database table.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

Use Size to determine the size associated with the corresponding physical field.

Size is meaningful only for a field definition object with one of the following TFieldType (see page 7612) values: ftString, ftBCD, ftBytes, ftVarBytes, ftBlob, ftMemo or ftGraphic. For string and byte fields, Size is the number of bytes reserved in the table for the field. For a BCD field, Size is the number of digits following the decimal point. For a BLOB, memo, or graphic field, Size is the number of bytes from the field's value that are stored in the actual database table.

See Also

DataType (see page 7191)

Size (see page 7178)

39.1.27 TFieldDefList Class

TFieldDefList lists the field definition (TFieldDef (see page 7184)) objects of a dataset in a flattened form.

Class Hierarchy

Pascal

```
TFieldDefList = class(TFlatList);
```

C++

```
class TFieldDefList : public TFlatList;
```

File

DB

Description

TFieldDefList is used by a dataset to implement its FieldDefList property. This object lists field definitions in a flattened format.

With object field types, such as TADTField (see page 6736) and TArrayField (see page 6760), there are two ways to represent the field definitions: hierarchically or flattened. When represented hierarchically, object field definitions contain a set of child definitions for the constituent subfields. When represented as a flattened set, child field definitions appear as siblings sequentially after the parent field definition.

TFieldDefList stores the flattened field definitions of a dataset. This is a linear representation of the data, and is in contrast to TFieldDefs (see page 7198) which has a hierarchical representation of the data.

TFieldDefList is a descendant of TStringList. The strings in this string list are the names of fields. The objects that correspond to those strings are the TFieldDef (see page 7184) objects.

See Also

TDataSet (see page 6894)

TFieldDefs (see page 7198)

TFieldDef (see page 7184)

TADTField (see page 6736)

TArrayField (see page 6760)

39.1.27.1 TFieldDefList Members

The following tables list the members exposed by TFieldDefList.

Protected Methods

	Name	Description
	FindItem (see page 7227)	This is FindItem, a member of class TFlatList.
V	GetCount (see page 7227)	This is GetCount, a member of class TFlatList.
V	GetUpdated (see page 7197)	This is GetUpdated, a member of class TFieldDefList.
	ListChanging (see page 7227)	This is ListChanging, a member of class TFlatList.
V	UpdateList (see page 7197)	This is UpdateList, a member of class TFieldDefList.

Protected Properties

	Name	Description
	Locked (see page 7229)	This is Locked, a member of class TFlatList.
	Updated (see page 7229)	This is Updated, a member of class TFlatList.

Public Methods

	Name	Description
	Create (see page 7226)	Creates an instance of TFlatList (see page 7225).
	FieldByName (see page 7196)	Returns a field definition object given its field name.
	Find (see page 7197)	Returns a field definition object given its field name.
	Update (see page 7228)	Refreshes the strings and objects to reflect the current state of the physical fields underlying the dataset.

Public Properties

	Name	Description
R	DataSet (see page 7228)	Identifies the dataset to which the TFlatList (see page 7225) object belongs.
R	FieldDefs (see page 7198)	Lists the field definitions for the dataset associated with the TFieldDefList (see page 7195) object.

39.1.27.2 TFieldDefList Methods

The methods of the TFieldDefList class are listed here.

Protected Methods

	Name	Description
V	GetUpdated (see page 7197)	This is GetUpdated, a member of class TFieldDefList.
V	UpdateList (see page 7197)	This is UpdateList, a member of class TFieldDefList.

Public Methods

	Name	Description
	FieldByName (see page 7196)	Returns a field definition object given its field name.
	Find (see page 7197)	Returns a field definition object given its field name.

39.1.27.2.1 TFieldDefList.FieldByName Method

Returns a field definition object given its field name.

Pascal

```
function FieldByName(const Name: WideString): TFieldDef;
```

C++

```
__fastcall TFieldDef FieldByName(const BSTR Name);
```

Description

Call FieldByName to obtain information about a particular field definition object. Specify the name of the field as the value of the Name parameter. If Name is not found, FieldByName raises an EDatabaseError (see page 6731) exception with the error message "Field not found".

Note: FieldByName differs from the Find (see page 7197) method only when the named field definition is not in the list. When the field definition is not found, FieldByName raises an exception, while Find (see page 7197) returns nil (Delphi) or NULL (C++).

See Also

FieldDefs (see page 7198)

Find (see page 7197)

39.1.27.2.2 TFieldDefList.Find Method

Returns a field definition object given its field name.

Pascal

```
function Find(const Name: WideString): TFieldDef; reintroduce;
```

C++

```
__fastcall TFieldDef Find(const BSTR Name);
```

Description

Call Find to obtain information about a particular field definition object. Specify the name of the field as the value of the Name parameter. If Find finds a field with a matching name, it returns the TFieldDef (see page 7184) component for the specified Name. Otherwise it returns nil (Delphi) or NULL (C++).

Note: Find differs from the FieldByName (see page 7196) method only when the named field definition is not in the list. When the field definition is not found, Find returns nil (Delphi) or NULL (C++), while FieldByName (see page 7196) raises an exception.

See Also

FieldDefs (see page 7198)

FieldByName (see page 7196)

39.1.27.2.3 TFieldDefList.GetUpdated Method

Pascal

```
function GetUpdated: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetUpdated();
```

Description

This is GetUpdated, a member of class TFieldDefList.

39.1.27.2.4 TFieldDefList.UpdateList Method

Pascal

```
procedure UpdateList; override;
```

C++

```
virtual __fastcall UpdateList();
```

Description

This is UpdateList, a member of class TFieldDefList.

39.1.27.3 TFieldDefList Properties

The properties of the TFieldDefList class are listed here.

Public Properties

	Name	Description
R	FieldDefs (see page 7198)	Lists the field definitions for the dataset associated with the TFieldDefList (see page 7195) object.

39.1.27.3.1 TFieldDefList.FieldDefs Property

Lists the field definitions for the dataset associated with the TFieldDefList (see page 7195) object.

Pascal

```
property FieldDefs [Index: Integer]: TFieldDef;
```

C++

```
__property TFieldDef FieldDefs[int Index];
```

Description

FieldDefs points to a internal list of field definitions for a dataset. A TFieldDefList (see page 7195) with object field types, such as TADTField (see page 6736) and TArrayField (see page 6760), lists child TFieldDef (see page 7184) objects sequentially after the parent TFieldDef (see page 7184).

See Also

Find (see page 7197)

DataSet (see page 7228)

operator_sb

39.1.28 TFieldDefs Class

TFieldDefs holds the field definition (TFieldDef (see page 7184)) objects that represent the physical fields underlying a dataset.

Class Hierarchy

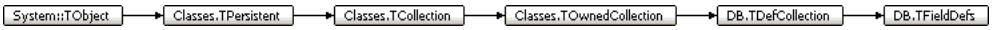

Pascal

```
TFieldDefs = class(TDefCollection);
```

C++

```
class TFieldDefs : public TDefCollection;
```

File

DB

Description

Datasets use TFieldDefs to manage the field definitions they use to create field objects that correspond to fields in a database table. Every field definition listed by TFieldDefs has a corresponding TField (see page 7115) object, but not all TField (see page 7115) objects have a corresponding field definition. For example, calculated fields do not have field definitions.

TTable and TClientDataSet objects also use TFieldDefs when creating a new table.

With object field types, such as TADTField (see page 6736) and TArrayField (see page 6760), there are two ways to represent the field definitions: hierarchically or flattened. When represented hierarchically, object field definitions contain a set of child definitions for the constituent subfields. When represented as a flattened set, child field definitions appear as siblings

sequentially after the parent field definition.

TFieldDefs stores field definitions hierarchically. This is in contrast to TFieldDefList (see page 7195), which lists the child TFieldDef (see page 7184) objects of object fields sequentially after the parent TFieldDef (see page 7184).

Use the properties and methods of TFieldDefs to:

Access a specific field definition.

Add (see page 7202) or delete field definitions from the list (when creating new tables).

Find (see page 7204) out how many fields are defined.

Copy a set of field definitions to another dataset.

See Also

TDataSet (see page 6894)

TFieldDef (see page 7184)

TDataSet (see page 6894)

TTable (see page 10168)

TFieldDefList (see page 7195)

TTable (see page 10168)

TCustomClientDataSet (see page 8373)

TADTField (see page 6736)

TArrayField (see page 6760)

39.1.28.1 TFieldDefs Members

The following tables list the members exposed by TFieldDefs.

Protected Events

	Name	Description
	OnUpdate (see page 7111)	This is OnUpdate, a member of class TDefCollection.

Protected Methods

	Name	Description
	Added (see page 2736)	Responds when items are added to the collection.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
	ChildDefUpdate (see page 7204)	This is ChildDefUpdate, a member of class TFieldDefs.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	Deleting (see page 2741)	Responds when items are deleted from the collection.
	DoUpdate (see page 7107)	This is DoUpdate, a member of class TDefCollection.
	FieldDefUpdate (see page 7204)	This is FieldDefUpdate, a member of class TFieldDefs.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
	GetOwner (see page 2915)	Returns the owner of an object.
	GetOwner (see page 2932)	Returns the owner of an object.

V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 7205)	Initializes the name of a newly inserted collection item.
	UpdateDefs (see page 7109)	This is UpdateDefs, a member of class TDefCollection.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 7202)	Creates a new field definition object and adds it to the Items (see page 7206) property.
	AddFieldDef (see page 7202)	Creates a new field definition object and adds it to the Items (see page 7206) property of this TFieldDefs (see page 7198) object.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 7201)	Creates an instance of TFieldDefs (see page 7198).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Find (see page 7204)	Returns a field definition, given the name of the field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItemNames (see page 7108)	This is the overview for the GetItemNames method overload.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	IndexOf (see page 7108)	Finds the index number of a definition in the Items array from its name.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Update (see page 7205)	Refreshes the field definitions in the Items (see page 7206) property array to reflect the current state of the physical fields underlying the dataset.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	DataSet (see page 7110)	Indicates the dataset that uses the TDefCollection (see page 7104) object to manage its definition objects.
	HiddenFields (see page 7206)	Determines whether to show the extra hidden columns that may be included to ensure unique records.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 7206)	Lists the field definitions that describe each physical field in the dataset.
R	ParentDef (see page 7207)	References the field definition of the parent field, if there is one.
	Updated (see page 7110)	Indicates whether the definition objects in the Items array match the corresponding instantiated objects in the dataset.

39.1.28.2 TFieldDefs.Create Constructor

Creates an instance of TFieldDefs (see page 7198).

Pascal

```
constructor Create(AOwner: TPersistent);
```

C++

```
__fastcall TFieldDefs(TPersistent * AOwner);
```

Description

Applications should not need to instantiate TFieldDefs (see page 7198). TFieldDefs (see page 7198) objects are created automatically by dataset components. An example of such a default TFieldDefs (see page 7198) object is the FieldDefs property of the TDataSet (see page 6894) component or the ChildDefs property of a TFieldDef (see page 7184) object.

The AOwner parameter specifies the object that uses TFieldDefs (see page 7198) to implement one of its properties (the dataset or field definition object).

See Also

FieldDefs (see page 7019)

ChildDefs (see page 7191)

39.1.28.3 TFieldDefs Methods

The methods of the TFieldDefs class are listed here.

Protected Methods

	Name	Description
	ChildDefUpdate (see page 7204)	This is ChildDefUpdate, a member of class TFieldDefs.
	FieldDefUpdate (see page 7204)	This is FieldDefUpdate, a member of class TFieldDefs.
V	SetItemName (see page 7205)	Initializes the name of a newly inserted collection item.

Public Methods

	Name	Description
	Add (see page 7202)	Creates a new field definition object and adds it to the Items (see page 7206) property.
	AddFieldDef (see page 7202)	Creates a new field definition object and adds it to the Items (see page 7206) property of this TFieldDefs (see page 7198) object.
	Find (see page 7204)	Returns a field definition, given the name of the field.
	Update (see page 7205)	Refreshes the field definitions in the Items (see page 7206) property array to reflect the current state of the physical fields underlying the dataset.

39.1.28.3.1 TFieldDefs.Add Method

Creates a new field definition object and adds it to the Items (see page 7206) property.

Pascal

```
procedure Add(const Name: string; DataType: TFieldType; Size: Integer = 0; Required: Boolean =
False);
```

C++

```
__fastcall Add(const AnsiString Name, TFieldType DataType, int Size = 0, Boolean Required =
False);
```

Description

Add is provided for backward compatibility. The recommended way to add new field definitions to the Items (see page 7206) property array is using the AddFieldDef (see page 7202) method.

Add uses the values passed in the Name, DataType, Size, and Required parameters and assigns them to the respective properties of the new field definition object.

If a field definition with same name already exists, Add raises an EDatabaseError (see page 6731) exception.

See Also

Items (see page 7206)

DataType (see page 7191)

Name (see page 7359)

Required (see page 7194)

Size (see page 7194)

AddFieldDef (see page 7202)

39.1.28.3.2 TFieldDefs.AddFieldDef Method

Creates a new field definition object and adds it to the Items (see page 7206) property of this TFieldDefs (see page 7198) object.

Pascal

```
function AddFieldDef: TFieldDef;
```

C++

```
__fastcall TFieldDef AddFieldDef();
```

Description

Call AddFieldDef to add field definitions when creating a client dataset's data or a database table. AddFieldDef returns the new TFieldDef (see page 7184) object that is added to the collection of field definitions. After calling AddFieldDef, set the properties of the newly added TFieldDef (see page 7184) object to specify the name, type, and size of field that should be created.

Note: AddFieldDef is only valid in the context of creating field definitions for a table to be created with a subsequent call to the CreateTable

or CreateDataSet method. It cannot be used to add fields to an already-existing table.

C++ Examples:

```
/*
The following code creates and activates a client dataset in
the form’s OnCreate event handler.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TFieldDefs *pDefs = CDS2->FieldDefs;
  TFieldDef *pDef = pDefs->AddFieldDef();
  pDef->DataType = ftInteger;
  pDef->Name = "Field1";

  pDef = pDefs->AddFieldDef();
  pDef->DataType = ftString;
  pDef->Size = 10;
  pDef->Name = "Field2";

  TIndexDef *pIDef = CDS2->IndexDefs->AddIndexDef();
  pIDef->Fields = "Field1";
  pIDef->Name = "IntIndex";

  CDS2->CreateDataSet();
}
```

Delphi Examples:

```
{
The following code creates and activates a client dataset in
the form’s OnCreate event handler.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  with CDS2 do
  begin
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftInteger;
      Name := 'Field1';
    end;
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftString;
      Size := 10;
      Name := 'Field2';
    end;
    with IndexDefs.AddIndexDef do
    begin
      Fields := 'Field1';
      Name := 'IntIndex';
    end;
```

```
      CreateDataSet;
    end;
  end;
```

See Also

AddChild (see page 7187)

Items (see page 7206)

CreateTable (see page 10189)

CreateDataSet (see page 8401)

Creating and Deleting Tables

39.1.28.3.3 TFieldDefs.ChildDefUpdate Method

Pascal

```
procedure ChildDefUpdate(Sender: TObject);
```

C++

```
__fastcall ChildDefUpdate(TObject * Sender);
```

Description

This is ChildDefUpdate, a member of class TFieldDefs.

39.1.28.3.4 TFieldDefs.FieldDefUpdate Method

Pascal

```
procedure FieldDefUpdate(Sender: TObject);
```

C++

```
__fastcall FieldDefUpdate(TObject * Sender);
```

Description

This is FieldDefUpdate, a member of class TFieldDefs.

39.1.28.3.5 TFieldDefs.Find Method

Returns a field definition, given the name of the field.

Pascal

```
function Find(const Name: string): TFieldDef;
```

C++

```
__fastcall TFieldDef Find(const AnsiString Name);
```

Description

Call Find to obtain information about a particular field definition. Specify the name of the field as the value of the Name parameter.

If no matching field is found, Find raises an exception. The IndexOf (see page 7108) method (inherited from TDefCollection (see page 7104)) can be used to determine if the field exists first before calling Find. If IndexOf (see page 7108) returns a value greater than –1, the TFieldDef (see page 7184) exists and Find can be used.

See Also

Items (see page 7206)

TFieldDef (see page 7184)

Name (see page 7359)

IndexOf (see page 7108)

39.1.28.3.6 TFieldDefs.SetItemName Method

Initializes the name of a newly inserted collection item.

Pascal

```
procedure SetItemName(AItem: TCollectionItem); override;
```

C++

```
virtual __fastcall SetItemName(TCollectionItem AItem);
```

Description

The Insert method calls SetItemName to initialize the Name property of items when it inserts them into the collection.

As implemented in TCollection, SetItemName does nothing. Some TCollection descendants override this method to provide collection items with default names.

See Also

Insert (see page 2746)

Name (see page 2805)

39.1.28.3.7 TFieldDefs.Update Method

Refreshes the field definitions in the Items (see page 7206) property array to reflect the current state of the physical fields underlying the dataset.

Pascal

```
procedure Update; reintroduce;
```

C++

```
__fastcall Update();
```

Description

Call Update to refresh the field definitions when the structure of the underlying dataset may have changed. Update updates the field definitions without opening the dataset.

Before updating, the Updated (see page 7110) property is checked to determine if the list is already up to date. You can force the list to be regenerated by setting the Updated (see page 7110) property to false.

See Also

Updated (see page 7110)

Items (see page 7206)

TDataSet (see page 6894)

TFieldDef (see page 7184)

39.1.28.4 TFieldDefs Properties

The properties of the TFieldDefs class are listed here.

Public Properties

	Name	Description
	HiddenFields (see page 7206)	Determines whether to show the extra hidden columns that may be included to ensure unique records.
	Items (see page 7206)	Lists the field definitions that describe each physical field in the dataset.
R	ParentDef (see page 7207)	References the field definition of the parent field, if there is one.

39.1.28.4.1 TFieldDefs.HiddenFields Property

Determines whether to show the extra hidden columns that may be included to ensure unique records.

Pascal

```
property HiddenFields: Boolean;
```

C++

```
__property Boolean HiddenFields;
```

Description

When the user performs a live query, the BDE returns extra hidden columns in the query result set. Similarly, pProviders sometimes include extra hidden columns in data packets. These extra columns ensure that each record contains enough information to locate the corresponding record in the underlying database tables.

Setting HiddenFields to true and calling TFieldDefs (see page 7198) . ::Update (see page 7205) allows these columns to be seen. HiddenFields is primarily for internal use.

See Also

Update (see page 7205)

39.1.28.4.2 TFieldDefs.Items Property

Lists the field definitions that describe each physical field in the dataset.

Pascal

```
property Items [Index: Integer]: TFieldDef;
```

C++

```
__property TFieldDef Items[int Index];
```

Description

Use Items to access a particular field definition. Specify the field definition to access with the Index parameter. Index is an integer identifying the field definition's position in the list of field definitions, in the range 0 to Count - 1.

For example, the following code displays the name of the first field definition in a label:

On Windows:

```
Label1.Caption := Table1.FieldDefs.Items[0].DisplayName;
```

On cross-platform:

```
Label1.Caption := ClientDataSet1.FieldDefs.Items[0].DisplayName;
```

On Windows:

```
Label1->Caption = Table1->FieldDefs->Items[0]->DisplayName;
```

On cross-platform:

```
Label1->Caption = ClientDataSet1->FieldDefs->Items[0]->DisplayName;
```

See Also

Add (see page 7202)

AddFieldDef (see page 7202)

Clear (see page 2741)

Count (see page 2749)

operator_sb

39.1.28.4.3 TFieldDefs.ParentDef Property

References the field definition of the parent field, if there is one.

Pascal

```
property ParentDef: TFieldDef;
```

C++

```
__property TFieldDef ParentDef;
```

Description

When a TFieldDefs (see page 7198) object represents the data members of an object field type (such as TADTField (see page 6736)), ParentDef specifies the field definition for that object field. Otherwise, ParentDef is nil (Delphi) or NULL (C++).

See Also

ChildDefs (see page 7191)

ParentField (see page 7176)

AddFieldDef (see page 7202)

HasChildDefs (see page 7189)

39.1.29 TFieldList Class

TFieldList lists the fields of a dataset, flattening out the structure of any object fields.

Class Hierarchy

Pascal

```
TFieldList = class(TFlatList);
```

C++

```
class TFieldList : public TFlatList;
```

File

DB

Description

TFieldList is used by a dataset to implement its FieldList property. It lists the fields of the dataset in a flattened format.

With object field types, such as TADTField (see page 6736) and TArrayField (see page 6760), there are two ways to represent the field components: hierarchically or flattened. When represented hierarchically, object fields contain a set of child field components for the constituent subfields. When represented as a flattened set, child fields appear as siblings sequentially after the parent field.

TFieldList is a descendant of TStringList. The strings in this string list are the names of fields. The objects that correspond to those strings are the TField (see page 7115) objects.

See Also

TDataSet (see page 6894)

TFields (see page 7211)

TField (see page 7115)

TADTField (see page 6736)

TArrayField (see page 6760)

TObjectField (see page 7372)

39.1.29.1 TFieldList Members

The following tables list the members exposed by TFieldList.

Protected Methods

	Name	Description
	FindItem (see page 7227)	This is FindItem, a member of class TFlatList.
V	GetCount (see page 7227)	This is GetCount, a member of class TFlatList.
V	GetUpdated (see page 7227)	This is GetUpdated, a member of class TFlatList.
	ListChanging (see page 7227)	This is ListChanging, a member of class TFlatList.
V	UpdateList (see page 7210)	This is UpdateList, a member of class TFieldList.

Protected Properties

	Name	Description
	Locked (see page 7229)	This is Locked, a member of class TFlatList.
	Updated (see page 7229)	This is Updated, a member of class TFlatList.

Public Methods

	Name	Description
	Create (see page 7226)	Creates an instance of TFlatList (see page 7225).
	FieldByName (see page 7209)	Returns a field, given its name.
	Find (see page 7209)	Returns a field component given the field name.
	Update (see page 7228)	Refreshes the strings and objects to reflect the current state of the physical fields underlying the dataset.

Public Properties

	Name	Description
R	DataSet (see page 7228)	Identifies the dataset to which the TFlatList (see page 7225) object belongs.

R	Fields (see page 7210)	Lists the fields in the dataset.

39.1.29.2 TFieldList Methods

The methods of the TFieldList class are listed here.

Protected Methods

	Name	Description
V	UpdateList (see page 7210)	This is UpdateList, a member of class TFieldList.

Public Methods

	Name	Description
	FieldByName (see page 7209)	Returns a field, given its name.
	Find (see page 7209)	Returns a field component given the field name.

39.1.29.2.1 TFieldList.FieldByName Method

Returns a field, given its name.

Pascal

```
function FieldByName(const Name: string): TField;
```

C++

```
__fastcall TField FieldByName(const AnsiString Name);
```

Description

Call FieldByName to retrieve field information for a field when only the field's name is known. FieldName is the name of an existing field. FieldByName returns the TField (see page 7115) component for the specified field. An application can access specific properties and methods of the field directly.

If the specified field is not found, FieldByName raises an EDatabaseError (see page 6731) exception.

Note: FieldByName differs from the Find (see page 7209) method only when the named field is not in the list. When the field is not found, FieldByName raises an exception, while Find (see page 7209) returns nil (Delphi) or NULL (C++).

See Also

Fields (see page 7210)

Find (see page 7209)

39.1.29.2.2 TFieldList.Find Method

Returns a field component given the field name.

Pascal

```
function Find(const Name: WideString): TField; reintroduce;
```

C++

```
__fastcall TField Find(const BSTR Name);
```

Description

Call Find to determine if a specified field component is referenced in the TFieldList (see page 7207) object. FieldName is the name of the field for which to search. If Find finds a field with a matching name, it returns the TField (see page 7115) component for the specified field. Otherwise it returns nil (Delphi) or NULL (C++).

Note: Find differs from the FieldByName (see page 7209) method only when the named field is not in the list. When the field is not found, Find returns nil (Delphi) or NULL (C++), while FieldByName (see page 7209) raises an exception.

See Also

Fields (see page 7210)

FieldByName (see page 7209)

39.1.29.2.3 TFieldList.UpdateList Method

Pascal

```
procedure UpdateList; override;
```

C++

```
virtual __fastcall UpdateList();
```

Description

This is UpdateList, a member of class TFieldList.

39.1.29.3 TFieldList Properties

The properties of the TFieldList class are listed here.

Public Properties

	Name	Description
R	Fields (see page 7210)	Lists the fields in the dataset.

39.1.29.3.1 TFieldList.Fields Property

Lists the fields in the dataset.

Pascal

```
property Fields [Index: Integer]: TField;
```

C++

```
__property TField Fields[int Index];
```

Description

Use Fields to obtain a pointer to a specific field component. The Index parameter indicates the index of the field, where 0 is the index of the first field, 1 is the index of the second field, and so on.

A TFieldList (see page 7207) with object field types, such as TADTField (see page 6736) and TArrayField (see page 6760), lists child TField (see page 7115) objects sequentially after the parent TField (see page 7115).

Use Fields with the Count property to iterate through all of the fields in the list.

See Also

Count

ParentField (see page 7176)

operator_sb

39.1.30 TFields Class

TFields holds the field (TField (see page 7115)) objects that represent the physical fields underlying a dataset or an object field such as TADTField (see page 6736) and TArrayField (see page 6760).

Class Hierarchy

Pascal

```
TFields = class(TObject);
```

C++

```
class TFields : public TObject;
```

File

DB

Description

TFields is used by a dataset to manage the field objects that correspond to fields in the database table and by object fields to manage child fields.

When TDataSet (see page 6894). ::ObjectView is true, the fields are stored hierarchically, meaning any child fields of an object field are referenced by the object field and do not appear sequentially after the object field in the Fields (see page 7221) array. When TDataSet (see page 6894). ::ObjectView is false, the fields are stored sequentially, or flattened out, meaning any child fields of an object field are stored sequentially in the Fields (see page 7221) array.

Use the properties and methods of TFields to:

Access a specific field.

Add (see page 7214) or delete persistent field components from the list.

Find out how many fields there are.

See Also

TField (see page 7115)

TDataSet (see page 6894)

TObjectField (see page 7372)

39.1.30.1 TFields Members

The following tables list the members exposed by TFields.

Protected Events

	Name	Description
	OnChange (see page 7222)	This is OnChange, a member of class TFields.

Protected Methods

	Name	Description
	Changed (see page 7214)	This is Changed, a member of class TFields.
	CheckFieldKind (see page 7215)	This is CheckFieldKind, a member of class TFields.

	GetCount (see page 7217)	This is GetCount, a member of class TFields.
	GetField (see page 7218)	This is GetField, a member of class TFields.
	SetField (see page 7220)	This is SetField, a member of class TFields.
	SetFieldIndex (see page 7220)	This is SetFieldIndex, a member of class TFields.

Protected Properties

	Name	Description
	SparseFields (see page 7221)	This is SparseFields, a member of class TFields.
	ValidFieldKinds (see page 7222)	This is ValidFieldKinds, a member of class TFields.

Public Methods

	Name	Description
	Add (see page 7214)	Inserts a new field to the end of the Fields (see page 7221) array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckFieldName (see page 7215)	Checks if a field name already exists in the TFields (see page 7211) object.
	CheckFieldNames (see page 7215)	Checks if fields exist in the TFields (see page 7211) object.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7216)	Deletes all fields from the TFields (see page 7211) object.
	Create (see page 7213)	Creates an instance of TFields (see page 7211).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 7213)	Destroys an instance of TFields (see page 7211).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 7216)	Returns a field given its field name.
	FieldByNumber (see page 7217)	Finds a field based on a specified field number.
	FindField (see page 7217)	Returns a field given its field name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 7218)	This is GetEnumerator, a member of class TFields.
	GetFieldNames (see page 7218)	This is the overview for the GetFieldNames method overload.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	IndexOf (see page 7219)	Finds the index of a field object in the Fields (see page 7221) array from the field object.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Remove (see page 7219)	Deletes the reference to the Field parameter from the Fields (see page 7221) array.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Count (see page 7220)	Indicates the number of fields in the fields object.
R	DataSet (see page 7221)	Identifies the dataset to which a TFields (see page 7211) object belongs.
	Fields (see page 7221)	Lists the field references that are managed by the TFields (see page 7211) object.

39.1.30.2 TFields.Create Constructor

Creates an instance of TFields (see page 7211).

Pascal

```
constructor Create(ADataSet: TDataSet);
```

C++

```
__fastcall TFields(TDataSet ADataSet);
```

Description

Most applications do not need to create a TFields (see page 7211) object. These objects are created by the dataset or object field that uses them.

See Also

Fields (see page 7221)

Destroy (see page 7213)

39.1.30.3 TFields.Destroy Destructor

Destroys an instance of TFields (see page 7211).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TFields();
```

Description

Most applications do not explicitly free a TFields (see page 7211) object. TFields (see page 7211) instances are freed by the dataset or object field that uses them.

Destroy frees the memory used to store the list of fields, including the field components themselves.

See Also

Fields (see page 7221)

Create (see page 7213)

Free (see page 22300)

39.1.30.4 TFields Methods

The methods of the TFields class are listed here.

Protected Methods

	Name	Description
	Changed (see page 7214)	This is Changed, a member of class TFields.
	CheckFieldKind (see page 7215)	This is CheckFieldKind, a member of class TFields.
	GetCount (see page 7217)	This is GetCount, a member of class TFields.
	GetField (see page 7218)	This is GetField, a member of class TFields.
	SetField (see page 7220)	This is SetField, a member of class TFields.
	SetFieldIndex (see page 7220)	This is SetFieldIndex, a member of class TFields.

Public Methods

	Name	Description
	Add (see page 7214)	Inserts a new field to the end of the Fields (see page 7221) array.
	CheckFieldName (see page 7215)	Checks if a field name already exists in the TFields (see page 7211) object.
	CheckFieldNames (see page 7215)	Checks if fields exist in the TFields (see page 7211) object.
	Clear (see page 7216)	Deletes all fields from the TFields (see page 7211) object.
	FieldByName (see page 7216)	Returns a field given its field name.
	FieldByNumber (see page 7217)	Finds a field based on a specified field number.
	FindField (see page 7217)	Returns a field given its field name.
	GetEnumerator (see page 7218)	This is GetEnumerator, a member of class TFields.
	GetFieldNames (see page 7218)	This is the overview for the GetFieldNames method overload.
	IndexOf (see page 7219)	Finds the index of a field object in the Fields (see page 7221) array from the field object.
	Remove (see page 7219)	Deletes the reference to the Field parameter from the Fields (see page 7221) array.

39.1.30.4.1 TFields.Add Method

Inserts a new field to the end of the Fields (see page 7221) array.

Pascal

```
procedure Add(Field: TField);
```

C++

```
__fastcall Add(TField Field);
```

Description

Applications do not normally call the Add method directly. When field objects are created, Add is called automatically by the dataset.

See Also

Fields (see page 7221)

Remove (see page 7219)

39.1.30.4.2 TFields.Changed Method

Pascal

```
procedure Changed;
```

C++

```
__fastcall Changed();
```

Description

This is Changed, a member of class TFields.

39.1.30.4.3 TFields.CheckFieldKind Method

Pascal

```
procedure CheckFieldKind(FieldKind: TFieldKind; Field: TField);
```

C++

```
__fastcall CheckFieldKind(TFieldKind FieldKind, TField Field);
```

Description

This is CheckFieldKind, a member of class TFields.

39.1.30.4.4 TFields.CheckFieldName Method

Checks if a field name already exists in the TFields (see page 7211) object.

Pascal

```
procedure CheckFieldName(const FieldName: WideString);
```

C++

```
__fastcall CheckFieldName(const BSTR FieldName);
```

Description

CheckFieldName checks for the field specified by FieldName in the Fields (see page 7221) property array. If the field with the specified name is already listed, CheckFieldName raises an EDatabaseError (see page 6731) exception with a duplicate field name error message.

See Also

Fields (see page 7221)

GetFieldNames (see page 7218)

EDatabaseError (see page 6731)

39.1.30.4.5 TFields.CheckFieldNames Method

Checks if fields exist in the TFields (see page 7211) object.

Pascal

```
procedure CheckFieldNames(const FieldNames: WideString);
```

C++

```
__fastcall CheckFieldNames(const BSTR FieldNames);
```

Description

CheckFieldNames checks for the set of fields specified by FieldNames in the Fields (see page 7221) property array. FieldNames lists the names of the fields to check, separated by semicolons. If any of the fields in FieldNames are not listed, CheckFieldNames raises an EDatabaseError (see page 6731) exception with a field not found error message.

See Also

CheckFieldName (see page 7215)

EDatabaseError (see page 6731)

GetFieldNames (see page 7218)

Fields (see page 7221)

39.1.30.4.6 TFields.Clear Method

Deletes all fields from the TFields (see page 7211) object.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Applications do not normally call the Clear method directly. Clear empties the Fields (see page 7221) property array, frees the memory used to store the array and sets the DataSet (see page 7221) property to nil (Delphi) or NULL (C++).

See Also

Remove (see page 7219)

Fields (see page 7221)

DataSet (see page 7221)

39.1.30.4.7 TFields.FieldByName Method

Returns a field given its field name.

Pascal

```
function FieldByName(const FieldName: WideString): TField;
```

C++

```
__fastcall TField FieldByName(const BSTR FieldName);
```

Description

Call FieldByName to retrieve field information for a field when only the field's name is known. FieldName is the name of an existing field. FieldByName returns the TField (see page 7115) component for the specified field. If the field can not be found, an exception is raised.

When the Fields (see page 7221) array contains ADT fields, you can use FieldByName to access the subfields of those ADT fields. To access subfields, specify FieldName as the name of the subfield qualified by the name of the ADT field, as in the following:

```
CityEdit.Text := Customer.FieldByName('Address.City').AsString;
CityEdit->Text = Customer->FieldByName("Address.City")->AsString;
```

FieldByName is especially useful when you do not have access to the underlying table and cannot use persistent field components.

Note: FieldByName differs from the FindField (see page 7217) method only when the named field is not in the list. When the field is not found, FieldByName raises an exception, while FindField (see page 7217) returns nil (Delphi) or NULL (C++).

See Also

FieldByNumber (see page 7217)

Fields (see page 7221)

FieldName (see page 7164)

FindField (see page 7217)

39.1.30.4.8 TFields.FieldByNumber Method

Finds a field based on a specified field number.

Pascal

```
function FieldByNumber(FieldNo: Integer): TField;
```

C++

```
__fastcall TField FieldByNumber(int FieldNo);
```

Description

Call FieldByNumber to retrieve field information for a field given its field number. Note that the field number is the position of the field in the underlying database table, not the index of the field in the TFields (see page 7211) object.

See Also

FieldByName (see page 7216)

Fields (see page 7221)

FieldNo (see page 7166)

39.1.30.4.9 TFields.FindField Method

Returns a field given its field name.

Pascal

```
function FindField(const FieldName: WideString): TField;
```

C++

```
__fastcall TField FindField(const BSTR FieldName);
```

Description

Call FindField to determine if a specified field component is referenced in the TFields (see page 7211) object. FieldName is the name of the field for which to search. If FindField finds a field with a matching name, it returns the TField (see page 7115) component for the specified field. Otherwise it returns nil (Delphi) or NULL (C++).

Note: FindField differs from the FieldByName (see page 7216) method only when the named field is not in the list. When the field is not found, FindField returns nil (Delphi) or NULL (C++), while FieldByName (see page 7216) raises an exception.

See Also

FieldByName (see page 7216)

Fields (see page 7221)

FieldName (see page 7164)

39.1.30.4.10 TFields.GetCount Method

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

This is GetCount, a member of class TFields.

39.1.30.4.11 TFields.GetEnumerator Method

Pascal

```
function GetEnumerator: TFieldsEnumerator;
```

C++

```
__fastcall TFieldsEnumerator GetEnumerator();
```

Description

This is GetEnumerator, a member of class TFields.

39.1.30.4.12 TFields.GetField Method

Pascal

```
function GetField(Index: Integer): TField;
```

C++

```
__fastcall TField GetField(int Index);
```

Description

This is GetField, a member of class TFields.

39.1.30.4.13 GetFieldNames Method

This is the overview for the GetFieldNames method overload.

Overload List

	Name	Description
	TFields.GetFieldNames (TStrings) (see page 7218)	Retrieves a list of names for all fields in a fields object.
	TFields.GetFieldNames (TWideStrings) (see page 7218)	Retrieves a list of names for all fields in a fields object.

39.1.30.4.13.1 TFields.GetFieldNames Method (TStrings)

Retrieves a list of names for all fields in a fields object.

Pascal

```
procedure GetFieldNames(List: TStrings); overload;
procedure GetFieldNames(List: TWideStrings); overload;
```

C++

```
__fastcall GetFieldNames(TStrings List);
__fastcall GetFieldNames(TWideStrings List);
```

Description

Call GetFieldNames to fill a list with the field names for all fields in the fields object. List is a TStrings descendant created and maintained by the application.

Retrieving a list of field names is especially useful for applications that work with datasets whose field objects are created dynamically at runtime. By retrieving a list of field names, the application can be restricted to working only with fields that exist at runtime.

See Also

FieldByName (see page 7216)

Fields (see page 7221)

39.1.30.4.14 TFields.IndexOf Method

Finds the index of a field object in the Fields (see page 7221) array from the field object.

Pascal

```
function IndexOf(Field: TField): Integer;
```

C++

```
__fastcall int IndexOf(TField Field);
```

Description

Call IndexOf to get the index for a field in the Fields (see page 7221) array. Specify the field as the Field parameter.

The first field in the array has index 0, the second field has index 1, and so on. If a field is not in the Fields (see page 7221) array, IndexOf returns -1.

See Also

FindField (see page 7217)

Fields (see page 7221)

39.1.30.4.15 TFields.Remove Method

Deletes the reference to the Field parameter from the Fields (see page 7221) array.

Pascal

```
procedure Remove(Field: TField);
```

C++

```
__fastcall Remove(TField Field);
```

Description

Applications do not normally call the Remove method directly. After a field is removed, all of the items that follow it are moved up in index position and the Count (see page 7220) is reduced by one.

Note: Unlike the Clear (see page 7216) method, which frees the field components when it removes them, Remove does not free the specified field component.

See Also

Add (see page 7214)

Fields (see page 7221)

39.1.30.4.16 TFields.SetField Method

Pascal

```
procedure SetField(Index: Integer; Value: TField);
```

C++

```
__fastcall SetField(int Index, TField Value);
```

Description

This is SetField, a member of class TFields.

39.1.30.4.17 TFields.SetFieldIndex Method

Pascal

```
procedure SetFieldIndex(Field: TField; Value: Integer);
```

C++

```
__fastcall SetFieldIndex(TField Field, int Value);
```

Description

This is SetFieldIndex, a member of class TFields.

39.1.30.5 TFields Properties

The properties of the TFields class are listed here.

Protected Properties

	Name	Description
	SparseFields (see page 7221)	This is SparseFields, a member of class TFields.
	ValidFieldKinds (see page 7222)	This is ValidFieldKinds, a member of class TFields.

Public Properties

	Name	Description
R	Count (see page 7220)	Indicates the number of fields in the fields object.
R	DataSet (see page 7221)	Identifies the dataset to which a TFields (see page 7211) object belongs.
	Fields (see page 7221)	Lists the field references that are managed by the TFields (see page 7211) object.

39.1.30.5.1 TFields.Count Property

Indicates the number of fields in the fields object.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use Count to determine the number of fields referenced by the TFields (see page 7211) object.

See Also

Fields (see page 7221)

39.1.30.5.2 TFields.DataSet Property

Identifies the dataset to which a TFields (see page 7211) object belongs.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Check the value of the DataSet property to determine the dataset that is associated with the TFields (see page 7211) object. Applications should not directly assign the DataSet property of a field. It is assigned automatically when the field is created from a TFieldDef (see page 7184) object.

See Also

Fields (see page 7221)

TFieldDef (see page 7184)

Fields (see page 7023)

39.1.30.5.3 TFields.Fields Property

Lists the field references that are managed by the TFields (see page 7211) object.

Pascal

```
property Fields [Index: Integer]: TField;
```

C++

```
__property TField Fields[int Index];
```

Description

Use Fields to obtain a pointer to a specific field. The Index parameter indicates the index of the field, where 0 is the index of the first field, 1 is the index of the second field, and so on.

Set Fields to assign the properties of another field component to one of the fields in the list.

Use Fields with the Count (see page 7220) property to iterate through all of the fields in the list.

See Also

Count (see page 7220)

operator_sb

39.1.30.5.4 TFields.SparseFields Property

Pascal

```
property SparseFields: Integer;
```

C++

```
__property int SparseFields;
```

Description

This is SparseFields, a member of class TFields.

39.1.30.5.5 TFields.ValidFieldKinds Property

Pascal

```
property ValidFieldKinds: TFieldKinds;
```

C++

```
__property TFieldKinds ValidFieldKinds;
```

Description

This is ValidFieldKinds, a member of class TFields.

39.1.30.6 TFields Events

The events of the TFields class are listed here.

Protected Events

	Name	Description
	OnChange (see page 7222)	This is OnChange, a member of class TFields.

39.1.30.6.1 TFields.OnChange Event

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

This is OnChange, a member of class TFields.

39.1.31 TFieldsEnumerator Class

Enumerates database fields.

Class Hierarchy

DB.TFieldsEnumerator

Pascal

```
TFieldsEnumerator = class;
```

C++

```
class TFieldsEnumerator;
```

File

DB

Description

This class enumerates the TField (see page 7115) objects in a TFields (see page 7211) object.

See Also

TField (see page 7115)

TFields (see page 7211)

39.1.31.1 TFieldsEnumerator Members

The following tables list the members exposed by TFieldsEnumerator.

Public Methods

	Name	Description
	Create (see page 7223)	Creates an instance of TFieldsEnumerator (see page 7222).
	GetCurrent (see page 7224)	Gets current field object being enumerated.
	MoveNext (see page 7224)	Move to next field in field enumeration.

Public Properties

	Name	Description
R	Current (see page 7224)	Current field object being enumerated.

39.1.31.2 TFieldsEnumerator.Create Constructor

Creates an instance of TFieldsEnumerator (see page 7222).

Pascal

```
constructor Create(AFields: TFields);
```

C++

```
__fastcall TFieldsEnumerator(TFields AFields);
```

Description

This method creates an instance of TFieldsEnumerator (see page 7222), suitable for enumerating all the TField (see page 7115) objects specified in the **AFields** parameter.

See Also

TField (see page 7115)

TFields (see page 7211)

39.1.31.3 TFieldsEnumerator Methods

The methods of the TFieldsEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 7224)	Gets current field object being enumerated.

	MoveNext (see page 7224)	Move to next field in field enumeration.

39.1.31.3.1 TFieldsEnumerator.GetCurrent Method

Gets current field object being enumerated.

Pascal

```
function GetCurrent: TField;
```

C++

```
__fastcall TField GetCurrent();
```

Description

GetCurrent gets the current TField (see page 7115) instance in the TFields (see page 7211) being enumerated.

See Also

TField (see page 7115)

TFields (see page 7211)

Current (see page 7224)

39.1.31.3.2 TFieldsEnumerator.MoveNext Method

Move to next field in field enumeration.

Pascal

```
function MoveNext: Boolean;
```

C++

```
__fastcall Boolean MoveNext();
```

Description

MoveNext moves to the next TField (see page 7115) instance in the TFields (see page 7211) being enumerated.

See Also

TField (see page 7115)

TFields (see page 7211)

39.1.31.4 TFieldsEnumerator Properties

The properties of the TFieldsEnumerator class are listed here.

Public Properties

	Name	Description
R	Current (see page 7224)	Current field object being enumerated.

39.1.31.4.1 TFieldsEnumerator.Current Property

Current field object being enumerated.

Pascal

```
property Current: TField;
```

C++

```
__property TField Current;
```

Description

Current contains the current TField (see page 7115) instance in the TFields (see page 7211) being enumerated.

See Also

TField (see page 7115)

TFields (see page 7211)

GetCurrent (see page 7224)

39.1.32 TFlatList Class

TFlatList is an abstract base class for objects that represent a flattened list of fields or field definitions.

Class Hierarchy

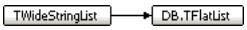

Pascal

```
TFlatList = class(TWideStringList);
```

C++

```
class TFlatList : public TWideStringList;
```

File

DB

Description

Derive a class from TFlatList to store and manipulate a list of fields or field definitions in an iterative manner. When fields are stored iteratively, the child fields of object fields such as ADTs and Arrays, appear as siblings after the parent object field rather than nested within the parent field.

Descendants of TFlatList include TFieldList (see page 7207) and TFieldDefList (see page 7195).

TFlatList is a descendant of TStringList. It uses the string list to maintain its list of fields or field definitions, where the strings in the list are the field names and the associated objects are TField (see page 7115) or TFieldDef (see page 7184) objects.

See Also

TDataSet (see page 6894)

TField (see page 7115)

TFieldDef (see page 7184)

39.1.32.1 TFlatList Members

The following tables list the members exposed by TFlatList.

Protected Methods

	Name	Description
	FindItem (see page 7227)	This is FindItem, a member of class TFlatList.
V	GetCount (see page 7227)	This is GetCount, a member of class TFlatList.
V	GetUpdated (see page 7227)	This is GetUpdated, a member of class TFlatList.
	ListChanging (see page 7227)	This is ListChanging, a member of class TFlatList.
A	UpdateList (see page 7228)	This is UpdateList, a member of class TFlatList.

Protected Properties

	Name	Description
	Locked (see page 7229)	This is Locked, a member of class TFlatList.
	Updated (see page 7229)	This is Updated, a member of class TFlatList.

Public Methods

	Name	Description
	Create (see page 7226)	Creates an instance of TFlatList (see page 7225).
	Update (see page 7228)	Refreshes the strings and objects to reflect the current state of the physical fields underlying the dataset.

Public Properties

	Name	Description
R	DataSet (see page 7228)	Identifies the dataset to which the TFlatList (see page 7225) object belongs.

39.1.32.2 TFlatList.Create Constructor

Creates an instance of TFlatList (see page 7225).

Pascal

```
constructor Create(ADataSet: TDataSet);
```

C++

```
__fastcall TFlatList(TDataSet ADataSet);
```

Description

Applications should not call the Create method. TFlatList (see page 7225) objects are created by the dataset that uses them to manage information about its fields. The ADataset parameter specifies the dataset that creates the TFlatList (see page 7225) object. It becomes the value of the Dataset property.

See Also

DataSet (see page 7228)

TFieldDefList (see page 7195)

TFieldList (see page 7207)

39.1.32.3 TFlatList Methods

The methods of the TFlatList class are listed here.

Protected Methods

	Name	Description
	FindItem (see page 7227)	This is FindItem, a member of class TFlatList.
V	GetCount (see page 7227)	This is GetCount, a member of class TFlatList.

	GetUpdated (see page 7227)	This is GetUpdated, a member of class TFlatList.
	ListChanging (see page 7227)	This is ListChanging, a member of class TFlatList.
	UpdateList (see page 7228)	This is UpdateList, a member of class TFlatList.

Public Methods

	Name	Description
	Update (see page 7228)	Refreshes the strings and objects to reflect the current state of the physical fields underlying the dataset.

39.1.32.3.1 TFlatList.FindItem Method

Pascal

```
function FindItem(const Name: WideString; MustExist: Boolean): TObject;
```

C++

```
__fastcall TObject * FindItem(const BSTR Name, Boolean MustExist);
```

Description

This is FindItem, a member of class TFlatList.

39.1.32.3.2 TFlatList.GetCount Method

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

This is GetCount, a member of class TFlatList.

39.1.32.3.3 TFlatList.GetUpdated Method

Pascal

```
function GetUpdated: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetUpdated();
```

Description

This is GetUpdated, a member of class TFlatList.

39.1.32.3.4 TFlatList.ListChanging Method

Pascal

```
procedure ListChanging(Sender: TObject);
```

C++

```
__fastcall ListChanging(TObject * Sender);
```

Description

This is ListChanging, a member of class TFlatList.

39.1.32.3.5 TFlatList.Update Method

Refreshes the strings and objects to reflect the current state of the physical fields underlying the dataset.

Pascal

```
procedure Update;
```

C++

```
__fastcall Update();
```

Description

Call Update to refresh the fields or field definitions when the structure of the underlying dataset may have changed. Update updates the field names listed by Strings and the corresponding objects without opening the dataset.

See Also

DataSet (see page 7228)

Strings

Objects

39.1.32.3.6 TFlatList.UpdateList Method

Pascal

```
procedure UpdateList; virtual; abstract;
```

C++

```
virtual __fastcall UpdateList() = 0;
```

Description

This is UpdateList, a member of class TFlatList.

39.1.32.4 TFlatList Properties

The properties of the TFlatList class are listed here.

Protected Properties

	Name	Description
	Locked (see page 7229)	This is Locked, a member of class TFlatList.
	Updated (see page 7229)	This is Updated, a member of class TFlatList.

Public Properties

	Name	Description
R	DataSet (see page 7228)	Identifies the dataset to which the TFlatList (see page 7225) object belongs.

39.1.32.4.1 TFlatList.DataSet Property

Identifies the dataset to which the TFlatList (see page 7225) object belongs.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Check the value of the DataSet property to determine the dataset that uses the TFlatList (see page 7225) object to manage information about its fields. Applications should not directly assign the DataSet property of a TFlatList (see page 7225) object. It is assigned automatically when a TFlatList (see page 7225) object descendant is created.

See Also

Create (see page 7226)

39.1.32.4.2 TFlatList.Locked Property

Pascal

```
property Locked: Boolean;
```

C++

```
__property Boolean Locked;
```

Description

This is Locked, a member of class TFlatList.

39.1.32.4.3 TFlatList.Updated Property

Pascal

```
property Updated: Boolean;
```

C++

```
__property Boolean Updated;
```

Description

This is Updated, a member of class TFlatList.

39.1.33 TFloatField Class

TFloatField represents a field that contains floating-point values in a dataset.

Class Hierarchy

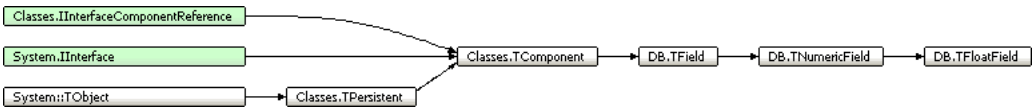

Pascal

```
TFloatField = class(TNumericField);
```

C++

```
class TFloatField : public TNumericField;
```

File

DB

Description

TFloatField encapsulates the fundamental behavior common to numeric data. Floating-point fields can hold values in the range from (positive or negative) 5.0 * 10^-324 to 1.7 * 10^308 with an accuracy of 15 digits. TFloatField introduces new properties to convert between integer values and other data types, to control the precision of numbers that the field will permit, and to format the value for display purposes.

If you use the Fields editor at design time to create a persistent field component for the float field, you can access it by name at runtime. When using dynamic field components, you can access the TFloatField instance using the dataset's Fields property or FieldByName method.

See Also

TCurrencyField (see page 6850)

TDataSet (see page 6894)

TDataSet (see page 6894)

TDataSet (see page 6894)

TField (see page 7115)

TAutoIncField (see page 6768)

TSmallintField (see page 7461)

TWordField (see page 7560)

Setting Display and Edit Properties at Design Time

Using Default Formatting for Numeric

Working with Field Components: Overview

39.1.33.1 TFloatField Members

The following tables list the members exposed by TFloatField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.

	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
	GetAsFloat (see page 7237)	This is GetAsFloat, a member of class TFloatField.
	GetAsInteger (see page 7237)	This is GetAsInteger, a member of class TFloatField.
	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
	GetAsString (see page 7238)	This is GetAsString, a member of class TFloatField.
	GetAsVariant (see page 7238)	This is GetAsVariant, a member of class TFloatField.
	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
	GetDataSize (see page 7238)	This is GetDataSize, a member of class TFloatField.
	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7238)	This is GetText, a member of class TFloatField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	RangeError (see page 7368)	This is RangeError, a member of class TNumericField.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.

	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7239)	This is SetAsFloat, a member of class TFloatField.
	SetAsInteger (see page 7239)	This is SetAsInteger, a member of class TFloatField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7239)	This is SetAsString, a member of class TFloatField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayFormat (see page 7368)	This is SetDisplayFormat, a member of class TNumericField.
	SetEditFormat (see page 7368)	This is SetEditFormat, a member of class TNumericField.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 7239)	This is SetVarValue, a member of class TFloatField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 7126)	Sets the value of the field to NULL.
V	Create (see page 7236)	Creates an instance of TFloatField (see page 7229).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.

R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7242)	Specifies the value of the floating-point field.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	Alignment (see page 7369)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	currency (see page 7240)	Determines whether the value in the field should be formatted as a currency value.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayFormat (see page 7369)	Determines how a numeric field's value is formatted for display in a data-aware control.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	EditFormat (see page 7371)	Determines how a numeric field's value is formatted when it is being edited in a data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.

	MaxValue (see page 7240)	Limits the maximum value in the floating-point field.
	MinValue (see page 7241)	Limits the minimum value in the floating-point field.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	Precision (see page 7241)	Determines the precision used in formatting the value in a floating-point field.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.33.2 TFloatField.Create Constructor

Creates an instance of TFloatField (see page 7229).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TFloatField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TFloatField (see page 7229). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftFloat and Precision (see page 7241) to 15. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TFloatField (see page 7229). After instantiating a TFloatField (see page 7229), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TFloatField (see page 7229) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TFloatField (see page 7229) with a dataset component by setting its DataSet (see page 7157) property to the name of the dataset component. Optionally set the Precision (see page 7241) property to the desired degree of precision.

The example below creates a TFloatField (see page 7229) object for a field named Amount accessed through a TSQLDataSet named SQLDataSet1.

```
var
  T: TFloatField;
begin
  SQLDataSet1.Close;
  T := TFloatField.Create(SQLDataSet1);
  T.Precision := 2;
  T.FieldName := 'Amount';
  T.Name := SQLDataSet1.Name + T.FieldName;
  T.Index := SQLDataSet1.FieldCount;
  T.DataSet := SQLDataSet1;
  SQLDataSet1.FieldDefs.UpDate;
  SQLDataSet1.Open;
end;
SQLDataSet1->Close();
TFloatField *T = new TFloatField(SQLDataSet1);
T->Precision = 2;
```

```
T->FieldName = "Amount";
T->Name = SQLDataSet1->Name + T->FieldName;
T->Index = SQLDataSet1->FieldCount;
T->DataSet = SQLDataSet1;
SQLDataSet1->FieldDefs->UpDate();
SQLDataSet1->Open();
```

See Also

Destroy (see page 7122)

Create (see page 7367)

DataType (see page 7159)

Precision (see page 7241)

Owner (see page 2807)

39.1.33.3 TFloatField Methods

The methods of the TFloatField class are listed here.

Protected Methods

	Name	Description
	GetAsFloat (see page 7237)	This is GetAsFloat, a member of class TFloatField.
	GetAsInteger (see page 7237)	This is GetAsInteger, a member of class TFloatField.
	GetAsString (see page 7238)	This is GetAsString, a member of class TFloatField.
	GetAsVariant (see page 7238)	This is GetAsVariant, a member of class TFloatField.
	GetDataSize (see page 7238)	This is GetDataSize, a member of class TFloatField.
	GetText (see page 7238)	This is GetText, a member of class TFloatField.
	SetAsFloat (see page 7239)	This is SetAsFloat, a member of class TFloatField.
	SetAsInteger (see page 7239)	This is SetAsInteger, a member of class TFloatField.
	SetAsString (see page 7239)	This is SetAsString, a member of class TFloatField.
	SetVarValue (see page 7239)	This is SetVarValue, a member of class TFloatField.

39.1.33.3.1 TFloatField.GetAsFloat Method

Pascal

```
function GetAsFloat: Double; override;
```

C++

```
virtual __fastcall Double GetAsFloat();
```

Description

This is GetAsFloat, a member of class TFloatField.

39.1.33.3.2 TFloatField.GetAsInteger Method

Pascal

```
function GetAsInteger: Longint; override;
```

C++

```
virtual __fastcall Longint GetAsInteger();
```

Description

This is GetAsInteger, a member of class TFloatField.

39.1.33.3.3 TFloatField.GetAsString Method

Pascal

```
function GetAsString: string; override;
```

C++

```
virtual __fastcall AnsiString GetAsString();
```

Description

This is GetAsString, a member of class TFloatField.

39.1.33.3.4 TFloatField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TFloatField.

39.1.33.3.5 TFloatField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; override;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TFloatField.

39.1.33.3.6 TFloatField.GetText Method

Pascal

```
procedure GetText(var Text: string; DisplayText: Boolean); override;
```

C++

```
virtual __fastcall GetText(AnsiString Text, Boolean DisplayText);
```

Description

This is GetText, a member of class TFloatField.

39.1.33.3.7 TFloatField.SetAsFloat Method

Pascal

```
procedure SetAsFloat(Value: Double); override;
```

C++

```
virtual __fastcall SetAsFloat(Double Value);
```

Description

This is SetAsFloat, a member of class TFloatField.

39.1.33.3.8 TFloatField.SetAsInteger Method

Pascal

```
procedure SetAsInteger(Value: Longint); override;
```

C++

```
virtual __fastcall SetAsInteger(Longint Value);
```

Description

This is SetAsInteger, a member of class TFloatField.

39.1.33.3.9 TFloatField.SetAsString Method

Pascal

```
procedure SetAsString(const Value: string); override;
```

C++

```
virtual __fastcall SetAsString(const AnsiString Value);
```

Description

This is SetAsString, a member of class TFloatField.

39.1.33.3.10 TFloatField.SetVarValue Method

Pascal

```
procedure SetVarValue(const Value: Variant); override;
```

C++

```
virtual __fastcall SetVarValue(const Variant Value);
```

Description

This is SetVarValue, a member of class TFloatField.

39.1.33.4 TFloatField Properties

The properties of the TFloatField class are listed here.

Public Properties

	Name	Description
	Value (see page 7242)	Specifies the value of the floating-point field.

Published Properties

	Name	Description
	currency (see page 7240)	Determines whether the value in the field should be formatted as a currency value.
	MaxValue (see page 7240)	Limits the maximum value in the floating-point field.
	MinValue (see page 7241)	Limits the minimum value in the floating-point field.
	Precision (see page 7241)	Determines the precision used in formatting the value in a floating-point field.

39.1.33.4.1 TFloatField.currency Property

Determines whether the value in the field should be formatted as a currency value.

Pascal

```
property currency: Boolean;
```

C++

```
__property Boolean currency;
```

Description

Use the currency property to specify the formatting of the field for viewing and editing in a data-aware control.

The field is formatted for display using the DisplayFormat (see page 7369) property if it is assigned. If DisplayFormat (see page 7369) is not assigned, currency determines how the field is formatted for display. If currency is true, the value is formatted for display using the FloatToText function with the ffCurrency formatting code. If currency is false, the value is formatted with the ffGeneral format.

The field is formatted for editing purposes using the EditFormat (see page 7371) (or DisplayFormat (see page 7369)) property if it is assigned. If neither EditFormat (see page 7371) nor DisplayFormat (see page 7369) is assigned, currency determines how the field is formatted for editing. If currency is true, the value is formatted for editing using the FloatToText function with the ffFixed formatting code. If currency is false, the value is formatted with the ffGeneral format.

See Also

DisplayFormat (see page 7369)

DisplayText (see page 7162)

EditFormat (see page 7371)

FloatToText (see page 22892)

FloatToTextFmt (see page 22893)

Text (see page 7179)

39.1.33.4.2 TFloatField.MaxValue Property

Limits the maximum value in the floating-point field.

Pascal

```
property MaxValue: Double;
```

C++

```
__property Double MaxValue;
```

Description

Use MaxValue to get or set the maximum value that can be entered into the field. If a value greater than MaxValue is entered, an EDatabaseError (see page 6731) exception is raised.

See Also

EDatabaseError (see page 6731)

MinValue (see page 7241)

39.1.33.4.3 TFloatField.MinValue Property

Limits the minimum value in the floating-point field.

Pascal

```
property MinValue: Double;
```

C++

```
__property Double MinValue;
```

Description

Use MinValue to get or set the minimum value that can be entered into the field. If a value smaller than MinValue is entered, an EDatabaseError (see page 6731) exception is raised.

See Also

EDatabaseError (see page 6731)

MaxValue (see page 7240)

39.1.33.4.4 TFloatField.Precision Property

Determines the precision used in formatting the value in a floating-point field.

Pascal

```
property Precision: Integer;
```

C++

```
__property int Precision;
```

Description

Use Precision to set the number of digits beyond the decimal point that the value should be formatted to before rounding begins. Precision affects the format of the floating-point value when it is displayed as a text string, not the actual floating-point value that is stored in the field.

The default value of Precision is 15 decimal places.

Note: Precision must be greater than or equal to 2. Trying to set Precision to a value less than 2 changes the value of Precision to 2.

See Also

AsString

DisplayFormat (see page 7369)

Working with Field Components: Overview

39.1.33.4.5 TFloatField.Value Property

Specifies the value of the floating-point field.

Pascal

```
property Value: Double;
```

C++

```
__property Double Value;
```

Description

Use Value to read data from and write data to a floating-point component at runtime. Using the Value property is the same as using the AsFloat (see page 7149) property. Use Value when you know the field component is a floating-point field. Use AsFloat (see page 7149) when working with a generic field component.

See Also

AsFloat

39.1.34 TFMTBCDField Class

TFMTBCDField represents a binary-coded decimal (BCD) field in a dataset.

Class Hierarchy

Pascal

```
TFMTBCDField = class(TNumericField);
```

C++

```
class TFMTBCDField : public TNumericField;
```

File

DB

Description

TFMTBCDField encapsulates the fundamental behavior common to binary-coded decimal (BCD) fields. BCD values provide greater precision and accuracy than floating-point numbers. BCD fields are often used for storing and manipulating monetary values.

The IDE uses two different field types for representing BCD fields: TFMTBCDField and TBCDField (see page 6776). TFMTBCDField uses a true BCD value (TBCD) to store and manipulate BCD values. This gives greater precision and accuracy than the Currency type used by TBCDField (see page 6776) objects, but results in somewhat slower performance. If your application does not require more than 4 decimal places or 20 significant digits, you may want to use TBCDField (see page 6776) instead to take advantage of its better performance.

If you use the Fields editor at design time to create a persistent field component for the BCD field, you can access it by name at runtime. When using dynamic field components, you can access the TFMTBCDField instance using the dataset's Fields property

or FieldByName method.

See Also

TDataSet (see page 6894)

TDataSet (see page 6894)

TDataSet (see page 6894)

TFloatField (see page 7229)

TBCDField (see page 6776)

TBcd

Using Default Formatting for Numeric

39.1.34.1 TFMTBCDField Members

The following tables list the members exposed by TFMTBCDField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7250)	This is CheckTypeSize, a member of class TFMTBCDField.
V	CopyData (see page 7250)	This is CopyData, a member of class TFMTBCDField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7250)	This is GetAsBCD, a member of class TFMTBCDField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7251)	This is GetAsCurrency, a member of class TFMTBCDField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsFloat (see page 7251)	This is GetAsFloat, a member of class TFMTBCDField.
V	GetAsInteger (see page 7251)	This is GetAsInteger, a member of class TFMTBCDField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsString (see page 7251)	This is GetAsString, a member of class TFMTBCDField.
V	GetAsVariant (see page 7252)	This is GetAsVariant, a member of class TFMTBCDField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.

	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
	GetDataSize (see page 7252)	This is GetDataSize, a member of class TFMTBCDField.
	GetDefaultWidth (see page 7252)	This is GetDefaultWidth, a member of class TFMTBCDField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7252)	This is GetText, a member of class TFMTBCDField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 7252)	This is GetValue, a member of class TFMTBCDField.
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	RangeError (see page 7368)	This is RangeError, a member of class TNumericField.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7253)	This is SetAsBCD, a member of class TFMTBCDField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7253)	This is SetAsCurrency, a member of class TFMTBCDField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7253)	This is SetAsFloat, a member of class TFMTBCDField.
	SetAsInteger (see page 7253)	This is SetAsInteger, a member of class TFMTBCDField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7254)	This is SetAsString, a member of class TFMTBCDField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayFormat (see page 7368)	This is SetDisplayFormat, a member of class TNumericField.
	SetEditFormat (see page 7368)	This is SetEditFormat, a member of class TNumericField.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property

	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 7254)	This is SetVarValue, a member of class TFMTBCDField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7126)	Sets the value of the field to NULL.
	Create (see page 7248)	Create creates an instance of TFMTBCDField (see page 7242).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.

	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7257)	Represents the value of the BCD field as a TBcd value.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.

	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	Alignment (see page 7369)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	currency (see page 7254)	Determines whether the value in the field should be formatted as a currency value.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayFormat (see page 7369)	Determines how a numeric field's value is formatted for display in a data-aware control.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	EditFormat (see page 7371)	Determines how a numeric field's value is formatted when it is being edited in a data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	MaxValue (see page 7255)	Limits the maximum value in the binary-coded decimal field.
	MinValue (see page 7255)	Limits the minimum value in the binary-coded decimal field.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	Precision (see page 7256)	Specifies the number of digits of accuracy used to store the field value.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Size (see page 7256)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.34.2 TFMTBCDField.Create Constructor

Create creates an instance of TFMTBCDField (see page 7242).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TFMTBCDField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TFMTBCDField (see page 7242). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TFMTBCDField (see page 7242). After instantiating a TFMTBCDField (see page 7242), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TFMTBCDField (see page 7242) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TFMTBCDField (see page 7242) with a dataset component by setting its DataSet (see page 7157) property to the name of the dataset component. Specify the precision of the TFMTBCDField (see page 7242) by setting the Precision (see page 7256) property.

The example below creates a TFMTBCDField (see page 7242) object for a field named Amount accessed through a TSQLQuery named SQLQuery1.

```
var
  T: TFMTBCDField;
begin
  SQLQuery1.Close;
  T := TFMTBCDField.Create(SQLQuery1);
  T.FieldName := 'Amount';
  T.Name := SQLQuery1.Name + T.FieldName;
  T.Index := SQLQuery1.FieldCount;
  T.DataSet := SQLQuery1;
  T.Precision := 20;
  SQLQuery1.FieldDefs.UpDate;
  SQLQuery1.Open;
end;
SQLQuery1->Close();
TFMTBCDField *T = new TFMTBCDField(SQLQuery1);
T->FieldName = "Amount";
T->Name = SQLQuery1->Name + T->FieldName;
T->Index = SQLQuery1->FieldCount;
T->DataSet = SQLQuery1;
T->Precision = 20;
SQLQuery1->FieldDefs->UpDate();
SQLQuery1->Open();
```

See Also

DataType (see page 7159)

Size (see page 7178)

Create (see page 7367)

Owner (see page 2807)

39.1.34.3 TFMTBCDField Methods

The methods of the TFMTBCDField class are listed here.

Protected Methods

	Name	Description
	CheckTypeSize (see page 7250)	This is CheckTypeSize, a member of class TFMTBCDField.
	CopyData (see page 7250)	This is CopyData, a member of class TFMTBCDField.
	GetAsBCD (see page 7250)	This is GetAsBCD, a member of class TFMTBCDField.
	GetAsCurrency (see page 7251)	This is GetAsCurrency, a member of class TFMTBCDField.
	GetAsFloat (see page 7251)	This is GetAsFloat, a member of class TFMTBCDField.
	GetAsInteger (see page 7251)	This is GetAsInteger, a member of class TFMTBCDField.
	GetAsString (see page 7251)	This is GetAsString, a member of class TFMTBCDField.
	GetAsVariant (see page 7252)	This is GetAsVariant, a member of class TFMTBCDField.
	GetDataSize (see page 7252)	This is GetDataSize, a member of class TFMTBCDField.
	GetDefaultWidth (see page 7252)	This is GetDefaultWidth, a member of class TFMTBCDField.
	GetText (see page 7252)	This is GetText, a member of class TFMTBCDField.
	GetValue (see page 7252)	This is GetValue, a member of class TFMTBCDField.
	SetAsBCD (see page 7253)	This is SetAsBCD, a member of class TFMTBCDField.
	SetAsCurrency (see page 7253)	This is SetAsCurrency, a member of class TFMTBCDField.
	SetAsFloat (see page 7253)	This is SetAsFloat, a member of class TFMTBCDField.
	SetAsInteger (see page 7253)	This is SetAsInteger, a member of class TFMTBCDField.
	SetAsString (see page 7254)	This is SetAsString, a member of class TFMTBCDField.
	SetVarValue (see page 7254)	This is SetVarValue, a member of class TFMTBCDField.

39.1.34.3.1 TFMTBCDField.CheckTypeSize Method

Pascal

```
class procedure CheckTypeSize(Value: Integer); override;
```

C++

```
virtual __fastcall CheckTypeSize(int Value);
```

Description

This is CheckTypeSize, a member of class TFMTBCDField.

39.1.34.3.2 TFMTBCDField.CopyData Method

Pascal

```
procedure CopyData(Source: Pointer; Dest: Pointer); override;
```

C++

```
virtual __fastcall CopyData(void * Source, void * Dest);
```

Description

This is CopyData, a member of class TFMTBCDField.

39.1.34.3.3 TFMTBCDField.GetAsBCD Method

Pascal

```
function GetAsBCD: TBcd; override;
```

C++

```
virtual __fastcall TBcd GetAsBCD();
```

Description

This is GetAsBCD, a member of class TFMTBCDField.

39.1.34.3.4 TFMTBCDField.GetAsCurrency Method

Pascal

```
function GetAsCurrency: Currency; override;
```

C++

```
virtual __fastcall Currency GetAsCurrency();
```

Description

This is GetAsCurrency, a member of class TFMTBCDField.

39.1.34.3.5 TFMTBCDField.GetAsFloat Method

Pascal

```
function GetAsFloat: Double; override;
```

C++

```
virtual __fastcall Double GetAsFloat();
```

Description

This is GetAsFloat, a member of class TFMTBCDField.

39.1.34.3.6 TFMTBCDField.GetAsInteger Method

Pascal

```
function GetAsInteger: Longint; override;
```

C++

```
virtual __fastcall Longint GetAsInteger();
```

Description

This is GetAsInteger, a member of class TFMTBCDField.

39.1.34.3.7 TFMTBCDField.GetAsString Method

Pascal

```
function GetAsString: string; override;
```

C++

```
virtual __fastcall AnsiString GetAsString();
```

Description

This is GetAsString, a member of class TFMTBCDField.

39.1.34.3.8 TFMTBCDField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TFMTBCDField.

39.1.34.3.9 TFMTBCDField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; override;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TFMTBCDField.

39.1.34.3.10 TFMTBCDField.GetDefaultWidth Method

Pascal

```
function GetDefaultWidth: Integer; override;
```

C++

```
virtual __fastcall int GetDefaultWidth();
```

Description

This is GetDefaultWidth, a member of class TFMTBCDField.

39.1.34.3.11 TFMTBCDField.GetText Method

Pascal

```
procedure GetText(var Text: string; DisplayText: Boolean); override;
```

C++

```
virtual __fastcall GetText(AnsiString Text, Boolean DisplayText);
```

Description

This is GetText, a member of class TFMTBCDField.

39.1.34.3.12 TFMTBCDField.GetValue Method

Pascal

```
function GetValue(var Value: TBcd): Boolean;
```

C++

```
__fastcall Boolean GetValue(TBcd Value);
```

Description

This is GetValue, a member of class TFMTBCDField.

39.1.34.3.13 TFMTBCDField.SetAsBCD Method

Pascal

```
procedure SetAsBCD(const Value: TBcd); override;
```

C++

```
virtual __fastcall SetAsBCD(const TBcd Value);
```

Description

This is SetAsBCD, a member of class TFMTBCDField.

39.1.34.3.14 TFMTBCDField.SetAsCurrency Method

Pascal

```
procedure SetAsCurrency(Value: Currency); override;
```

C++

```
virtual __fastcall SetAsCurrency(Currency Value);
```

Description

This is SetAsCurrency, a member of class TFMTBCDField.

39.1.34.3.15 TFMTBCDField.SetAsFloat Method

Pascal

```
procedure SetAsFloat(Value: Double); override;
```

C++

```
virtual __fastcall SetAsFloat(Double Value);
```

Description

This is SetAsFloat, a member of class TFMTBCDField.

39.1.34.3.16 TFMTBCDField.SetAsInteger Method

Pascal

```
procedure SetAsInteger(Value: Longint); override;
```

C++

```
virtual __fastcall SetAsInteger(Longint Value);
```

Description

This is SetAsInteger, a member of class TFMTBCDField.

39.1.34.3.17 TFMTBCDField.SetAsString Method

Pascal

```
procedure SetAsString(const Value: string); override;
```

C++

```
virtual __fastcall SetAsString(const AnsiString Value);
```

Description

This is SetAsString, a member of class TFMTBCDField.

39.1.34.3.18 TFMTBCDField.SetVarValue Method

Pascal

```
procedure SetVarValue(const Value: Variant); override;
```

C++

```
virtual __fastcall SetVarValue(const Variant Value);
```

Description

This is SetVarValue, a member of class TFMTBCDField.

39.1.34.4 TFMTBCDField Properties

The properties of the TFMTBCDField class are listed here.

Public Properties

	Name	Description
	Value (see page 7257)	Represents the value of the BCD field as a TBcd value.

Published Properties

	Name	Description
	currency (see page 7254)	Determines whether the value in the field should be formatted as a currency value.
	MaxValue (see page 7255)	Limits the maximum value in the binary-coded decimal field.
	MinValue (see page 7255)	Limits the minimum value in the binary-coded decimal field.
	Precision (see page 7256)	Specifies the number of digits of accuracy used to store the field value.
	Size (see page 7256)	Indicates the size used in the definition of the physical database field for data types that support different sizes.

39.1.34.4.1 TFMTBCDField.currency Property

Determines whether the value in the field should be formatted as a currency value.

Pascal

```
property currency: Boolean;
```

C++

```
__property Boolean currency;
```

Description

Use the currency property to influence how the value in the field should be formatted for viewing and editing in a data-aware

control.

For display purposes, the field is formatted using the DisplayFormat (see page 7369) property if it is assigned. Similarly, for editing purposes, the field is formatted using the EditFormat (see page 7371) (or DisplayFormat (see page 7369)) property if it is assigned.

If DisplayFormat (see page 7369) or EditFormat (see page 7371) is not assigned, C currency determines how the field is formatted. If currency is true, the value is formatted to represent a monetary value. If currency is false, it is formatted to reflect the precision and decimal places of the BCD type.

See Also

DisplayFormat (see page 7369)

DisplayText (see page 7162)

EditFormat (see page 7371)

FloatToText (see page 22892)

FloatToTextFmt (see page 22893)

Text (see page 7179)

TFloatFormat (see page 22999)

39.1.34.4.2 TFMTBCDField.MaxValue Property

Limits the maximum value in the binary-coded decimal field.

Pascal

```
property MaxValue: String;
```

C++

```
__property AnsiString MaxValue;
```

Description

Use MaxValue to get or set the maximum value that can be entered into the field. MaxValue is the string representation of the maximum value.

If a value greater than MaxValue is entered, TFMTBCDField (see page 7242) raises an EDatabaseError (see page 6731) exception.

See Also

MinValue (see page 7255)

Value (see page 7257)

39.1.34.4.3 TFMTBCDField.MinValue Property

Limits the minimum value in the binary-coded decimal field.

Pascal

```
property MinValue: String;
```

C++

```
__property AnsiString MinValue;
```

Description

Use MinValue to get or set the minimum value that can be entered into the field. MinValue is the string representation of the minimum value.

If a value smaller than MinValue is entered, TFMTBCDField (see page 7242) raises an EDatabaseError (see page 6731) exception.

See Also

MaxValue (see page 7255)

Value (see page 7257)

39.1.34.4.4 TFMTBCDField.Precision Property

Specifies the number of digits of accuracy used to store the field value.

Pascal

```
property Precision: Integer;
```

C++

```
__property int Precision;
```

Description

Precision can be a value between 0 and 32. Some database servers support more digits than 32, but these can't be handled by client datasets, and so TFMTBCDField (see page 7242) limits the value of Precision to be no greater than 32.

See Also

AsCurrency

SetData (see page 7141)

39.1.34.4.5 TFMTBCDField.Size Property

Indicates the size used in the definition of the physical database field for data types that support different sizes.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

The interpretation of Size depends on the data type. The meaning of Size for data types that use it is given in the following table. For all other data types, Size is not used and its value is always 0.

DataType	Meaning of Size
ftBCD	Size is the number of digits after the decimal place.
ftString	Size is the maximum number of characters in the string.
ftVarBytes	Size is the maximum number of bytes of data, not counting the first two bytes which indicate the actual number of bytes of data that were used.
ftBytes	Size is the maximum number of bytes of data.
ftBlob	Size is the number of bytes from the BLOB that are stored in the record buffer.

ftDBaseOle	Size is the number of bytes from the dBASE OLE BLOB that are stored in the record buffer.
ftFMTBCD	Size is the number of digits after the decimal place.
ftFmtMemo	Size is the number of bytes from the memo that are stored in the record buffer.
ftGraphic	Size is the number of bytes from the image that are stored in the record buffer.
ftMemo	Size is the number of characters from the memo that are stored in the record buffer.
ftParadoxOle	Size is the number of bytes from the Paradox OLE BLOB that are stored in the record buffer.
ftTypedBinary	Size is the number of bytes from the BLOB that are stored in the record buffer.
ftADT	Size is the total number of fields contained in the ADT field, including the fields of child fields that are of type ADT.
ftArray	Size is the number of elements in the array.

See Also

DataSize (see page 7157)

DataType (see page 7159)

39.1.34.4.6 TFMTBCDField.Value Property

Represents the value of the BCD field as a TBcd value.

Pascal

```
property Value: TBcd;
```

C++

```
__property TBcd Value;
```

Description

Use Value to read data from and write data to a BCD field. Using the Value property is the same as using the AsBCD (see page 7147) property.

If a value assigned to the field is not within the range established by the MinValue (see page 7255) and the MaxValue (see page 7255) properties, an EDatabaseError (see page 6731) exception is raised.

See Also

AsBCD

SetData (see page 7141)

39.1.35 TGraphicField Class

TGraphicField represents a graphics field in a dataset.

Class Hierarchy

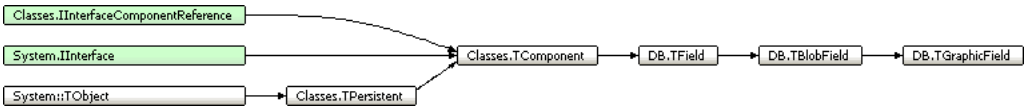

Pascal

```
TGraphicField = class(TBlobField);
```

C++

```
class TGraphicField : public TBlobField;
```

File

DB

Description

TGraphicField encapsulates the fundamental behavior common to fields that hold graphic data. Graphics (see page 14387) fields are a form of binary large object (BLOB) field where the data includes a BLOB header describing the encoding of the graphical value.

If you use the Fields editor at design time to create a persistent field component for the graphics field, you can access it by name at runtime. When using dynamic field components, you can access the TGraphicField instance using the dataset's Fields property or FieldByName method.

See Also

TImage (see page 12777)

TMemoField (see page 7348)

TPicture (see page 14584)

39.1.35.1 TGraphicField Members

The following tables list the members exposed by TGraphicField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 6810)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 6811)	This is FreeBuffers, a member of class TBlobField.

	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
	GetAsString (see page 6811)	This is GetAsString, a member of class TBlobField.
	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
	GetAsVariant (see page 6811)	This is GetAsVariant, a member of class TBlobField.
	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
	GetAsWideString (see page 6812)	This is GetAsWideString, a member of class TBlobField.
	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
	GetBlobSize (see page 6812)	This is GetBlobSize, a member of class TBlobField.
	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClassDesc (see page 6812)	This is GetClassDesc, a member of class TBlobField.
	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsNull (see page 6812)	This is GetIsNull, a member of class TBlobField.
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 6812)	This is GetText, a member of class TBlobField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.

	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 6817)	This is SetAsString, a member of class TBlobField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 6817)	This is SetAsWideString, a member of class TBlobField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 6818)	This is SetText, a member of class TBlobField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 6818)	This is SetVarValue, a member of class TBlobField.
	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 6809)	Copies a value to the BLOB field.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.

	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 6810)	Clears the current value in the BLOB field.
V	Clear (see page 7126)	Sets the value of the field to NULL.
V	Create (see page 7264)	Creates an instance of TGraphicField (see page 7257).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 6813)	Indicates that this is a BLOB field.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	LoadFromFile (see page 6814)	Loads BLOB data from a file into the field.
	LoadFromStream (see page 6815)	Loads BLOB data from a stream into the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 6816)	Saves the contents of the BLOB field to a file.
	SaveToStream (see page 6816)	Saves the contents of the BLOB field to a stream.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 6817)	Specifies the type of the BLOB field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
R	BlobSize (see page 6819)	Indicates the number of bytes of data contained in the blob field on the current record.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.

	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	Modified (see page 6820)	Indicates whether a the value for the BLOB field was changed since it was last read from the underlying database.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	Transliterate (see page 6821)	Determines whether any text in the BLOB field should be converted to and from the character set identified by the dataset.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 6822)	Represents the data in the BLOB field.
	Value (see page 7180)	Represents the data in a field component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	BlobType (see page 6819)	Identifies the type of the BLOB field associated with a BLOB field object.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.

	GraphicHeader (see page 6820)	Indicates whether the field's value includes a Paradox graphic BLOB header.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Size (see page 6820)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.35.2 TGraphicField.Create Constructor

Creates an instance of TGraphicField (see page 7257).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TGraphicField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TGraphicField (see page 7257). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftGraphic. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TGraphicField (see page 7257). After instantiating a TGraphicField (see page 7257), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TGraphicField (see page 7257) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TGraphicField (see page 7257) with a dataset component by setting its DataSet (see page 7157) property to the name of the dataset component.

The example below creates a TGraphicField (see page 7257) object for a field named Pics accessed through a TSQLDataSet named SQLDataSet1.

```
var
  T: TGraphicField;
begin
  SQLDataSet1.Close;
  T := TGraphicField.Create(SQLDataSet1);
  T.FieldName := 'Pics';
  T.Name := SQLDataSet1.Name + T.FieldName;
```

```
    T.Index := SQLDataSet1.FieldCount;
    T.DataSet := SQLDataSet1;
    SQLDataSet1.FieldDefs.UpDate;
    SQLDataSet1.Open;
  end;
  TGraphicField *T;
  SQLDataSet1->Close();
  T = new TGraphicField(SQLDataSet1);
  T->FieldName = "Pics";
  T->Name = SQLDataSet1->Name + T->FieldName;
  T->Index = SQLDataSet1->FieldCount;
  T->DataSet = SQLDataSet1;
  SQLDataSet1->FieldDefs->UpDate();
  SQLDataSet1->Open();
```

See Also

Destroy (see page 7122)

DataType (see page 7159)

Create (see page 7121)

Owner (see page 2807)

39.1.36 TGuidField Class

TGuidField represents a guid field in a dataset.

Class Hierarchy

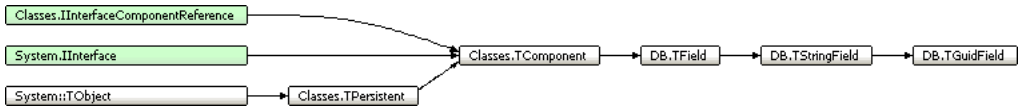

Pascal

```
TGuidField = class(TStringField);
```

C++

```
class TGuidField : public TStringField;
```

File

DB

Description

TGuidField encapsulates the fundamental behavior common to fields storing Globally Unique Identifiers (GUIDs). In addition to the properties, methods, and events common to all fields that are introduced by the TField (see page 7115) class, TGuidField introduces the AsGuid (see page 7274) property for setting and getting the field's value as a TGUID data type.

If you use the Fields editor at design time to create a persistent field component for the guid field, you can access it by name at runtime. When using dynamic field components, you can access the TGuidField instance using the dataset's Fields property or FieldByName method.

Maximum/minimum data storage limits?

See Also

TDataSet (see page 6894)

TMemoField (see page 7348)

TDataSet (see page 6894)

TDataSet (see page 6894)

39.1.36.1 TGuidField Members

The following tables list the members exposed by TGuidField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7273)	This is CheckTypeSize, a member of class TGuidField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBCD (see page 7492)	This is GetAsBCD, a member of class TStringField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsBoolean (see page 7492)	This is GetAsBoolean, a member of class TStringField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsDateTime (see page 7492)	This is GetAsDateTime, a member of class TStringField.
V	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
V	GetAsFloat (see page 7492)	This is GetAsFloat, a member of class TStringField.
	GetAsGuid (see page 7273)	This is GetAsGuid, a member of class TGuidField.
V	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
V	GetAsInteger (see page 7493)	This is GetAsInteger, a member of class TStringField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsSQLTimeStamp (see page 7493)	This is GetAsSQLTimeStamp, a member of class TStringField.
V	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
V	GetAsString (see page 7493)	This is GetAsString, a member of class TStringField.
V	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
V	GetAsVariant (see page 7493)	This is GetAsVariant, a member of class TStringField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.

	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
	GetDataSize (see page 7493)	This is GetDataSize, a member of class TStringField.
	GetDefaultWidth (see page 7273)	This is GetDefaultWidth, a member of class TGuidField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetText (see page 7494)	This is GetText, a member of class TStringField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 7494)	This is GetValue, a member of class TStringField.
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBCD (see page 7494)	This is SetAsBCD, a member of class TStringField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsBoolean (see page 7495)	This is SetAsBoolean, a member of class TStringField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsDateTime (see page 7495)	This is SetAsDateTime, a member of class TStringField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsFloat (see page 7495)	This is SetAsFloat, a member of class TStringField.
	SetAsGuid (see page 7273)	This is SetAsGuid, a member of class TGuidField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsInteger (see page 7495)	This is SetAsInteger, a member of class TStringField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsSQLTimeStamp (see page 7495)	This is SetAsSQLTimeStamp, a member of class TStringField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsString (see page 7496)	This is SetAsString, a member of class TStringField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
	SetVarValue (see page 7496)	This is SetVarValue, a member of class TStringField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7126)	Sets the value of the field to NULL.
	Create (see page 7272)	Creates an instance of a TGuidField (see page 7265) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsGuid (see page 7274)	Represents the value of the field as a Globally Unique Identifier (GUID).
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.

	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7180)	Represents the data in a field component.
	Value (see page 7499)	Represents the data in the string field.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	EditMask (see page 7496)	Contains a mask that restricts the data that can be entered into a data field.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
	FixedChar (see page 7497)	Specifies whether the represented field is of Char type.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Size (see page 7497)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Transliterate (see page 7498)	Determines whether text should be converted to and from the character set identified by the dataset.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.36.2 TGuidField.Create Constructor

Creates an instance of a TGuidField (see page 7265) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TGuidField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TGuidField (see page 7265). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftGuid. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TGuidField (see page 7265). After instantiating a TGuidField (see page 7265), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TGuidField (see page 7265) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TGuidField (see page 7265) with a dataset component by setting its DataSet (see page 7157) property.

The example below creates a TGuidField (see page 7265) object for a field named Guid accessed through a TQuery named Query1.

```
var
  T: TGuidField;
begin
  Query1.Close;
  T := TGuidField.Create(Query1);
  T.FieldName := 'Guid';
  T.Name := Query1.Name + T.FieldName;
  T.Index := Query1.FieldCount;
  T.DataSet := Query1;
  Query1.FieldDefs.UpDate;
  Query1.Open;
end;
TGuidField *T;
Query1->Close();
T = new TGuidField(Query1);
T->FieldName = "Guid";
T->Name = Query1->Name + T->FieldName;
T->Index = Query1->FieldCount;
T->DataSet = Query1;
Query1->FieldDefs->UpDate();
Query1->Open();
```

See Also

Destroy (see page 7122)

DataType (see page 7159)

Size (see page 7178)

Owner (see page 2807)

39.1.36.3 TGuidField Methods

The methods of the TGuidField class are listed here.

Protected Methods

	Name	Description
	CheckTypeSize (see page 7273)	This is CheckTypeSize, a member of class TGuidField.
	GetAsGuid (see page 7273)	This is GetAsGuid, a member of class TGuidField.
	GetDefaultWidth (see page 7273)	This is GetDefaultWidth, a member of class TGuidField.
	SetAsGuid (see page 7273)	This is SetAsGuid, a member of class TGuidField.

39.1.36.3.1 TGuidField.CheckTypeSize Method

Pascal

```
class procedure CheckTypeSize(Value: Integer); override;
```

C++

```
virtual __fastcall CheckTypeSize(int Value);
```

Description

This is CheckTypeSize, a member of class TGuidField.

39.1.36.3.2 TGuidField.GetAsGuid Method

Pascal

```
function GetAsGuid: TGUID;
```

C++

```
__fastcall TGUID GetAsGuid();
```

Description

This is GetAsGuid, a member of class TGuidField.

39.1.36.3.3 TGuidField.GetDefaultWidth Method

Pascal

```
function GetDefaultWidth: Integer; override;
```

C++

```
virtual __fastcall int GetDefaultWidth();
```

Description

This is GetDefaultWidth, a member of class TGuidField.

39.1.36.3.4 TGuidField.SetAsGuid Method

Pascal

```
procedure SetAsGuid(const Value: TGUID);
```

C++

```
__fastcall SetAsGuid(const TGUID Value);
```

Description

This is SetAsGuid, a member of class TGuidField.

39.1.36.4 TGuidField Properties

The properties of the TGuidField class are listed here.

Public Properties

	Name	Description
	AsGuid (see page 7274)	Represents the value of the field as a Globally Unique Identifier (GUID).

39.1.36.4.1 TGuidField.AsGuid Property

Represents the value of the field as a Globally Unique Identifier (GUID).

Pascal

```
property AsGuid: TGUID;
```

C++

```
__property TGUID AsGuid;
```

Description

Use AsGuid to

Assign (see page 7124) the GUID field value to a TGUID variable (Delphi) or GUID variable (C++).

Assign (see page 7124) a TGUID value (Delphi) or GUID value (C++) as the value of the field.

AsGuid uses the StringToGUID and GUIDToString functions internally to converts between the internal representation of the GUID as a string and a TGUID type.

See Also

StringToGUID (see page 22952)

GuidToString

39.1.37 TIDispatchField Class

TIDispatchField represents an IDispatch field in a dataset.

Class Hierarchy

Pascal

```
TIDispatchField = class(TInterfaceField);
```

C++

```
class TIDispatchField : public TInterfaceField;
```

File

DB

Description

TIDispatchField encapsulates the fundamental behavior common to fields that contain pointers to IDispatch interfaces as data.

If you use the Fields editor at design time to create a persistent field component for the IDispatch field, you can access it by name at runtime. When using dynamic field components, you can access the TIDispatchField instance using the dataset's Fields property or FieldByName method.

Maximum/minimum capacities?

See Also

TDataSet (see page 6894)

TMemoField (see page 7348)

TDataSet (see page 6894)

TDataSet (see page 6894)

39.1.37.1 TIDispatchField Members

The following tables list the members exposed by TIDispatchField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CheckTypeSize (see page 7321)	This is CheckTypeSize, a member of class TInterfaceField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.

	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
	GetAsVariant (see page 7321)	This is GetAsVariant, a member of class TInterfaceField.
	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 7282)	This is GetValue, a member of class TIDispatchField.
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetValue (see page 7282)	This is SetValue, a member of class TIDispatchField.
	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
	SetVarValue (see page 7322)	This is SetVarValue, a member of class TInterfaceField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7126)	Sets the value of the field to NULL.
	Create (see page 7280)	Creates and initializes an IDispatch field component.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.

	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7282)	Represents the data in an IDispatch field component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.37.2 TIDispatchField.Create Constructor

Creates and initializes an IDispatch field component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TIDispatchField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TIDispatchField (see page 7274). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftIDispatch. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TIDispatchField (see page 7274). After instantiating a TIDispatchField (see page 7274), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TIDispatchField (see page 7274) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TIDispatchField (see page 7274) with a dataset component by setting its DataSet (see page 7157) property.

The example below creates a TIDispatchField (see page 7274) object for a field named IDispatch accessed through a TQuery named Query1.

```
var
  T: TIDispatchField;
begin
  Query1.Close;
  T := TIDispatchField.Create(Query1);
  T.FieldName := 'IDispatch';
  T.Name := Query1.Name + T.FieldName;
  T.Index := Query1.FieldCount;
  T.DataSet := Query1;
  Query1.FieldDefs.UpDate;
  Query1.Open;
end;
TIDispatchField *T;
Query1->Close();
T = new TIDispatchField(Query1);
T->FieldName = "IDispatch";
T->Name = Query1->Name + T->FieldName;
T->Index = Query1->FieldCount;
T->DataSet = Query1;
Query1->FieldDefs->UpDate();
Query1->Open();
```

See Also

Visible (see page 7181)

Destroy (see page 7122)

Owner (see page 2807)

39.1.37.3 TIDispatchField Methods

The methods of the TIDispatchField class are listed here.

Protected Methods

	Name	Description
	GetValue (see page 7282)	This is GetValue, a member of class TIDispatchField.
	SetValue (see page 7282)	This is SetValue, a member of class TIDispatchField.

39.1.37.3.1 TIDispatchField.GetValue Method

Pascal

```
function GetValue: IDispatch;
```

C++

```
__fastcall IDispatch GetValue();
```

Description

This is GetValue, a member of class TIDispatchField.

39.1.37.3.2 TIDispatchField.SetValue Method

Pascal

```
procedure SetValue(const Value: IDispatch);
```

C++

```
__fastcall SetValue(const IDispatch Value);
```

Description

This is SetValue, a member of class TIDispatchField.

39.1.37.4 TIDispatchField Properties

The properties of the TIDispatchField class are listed here.

Public Properties

	Name	Description
	Value (see page 7282)	Represents the data in an IDispatch field component.

39.1.37.4.1 TIDispatchField.Value Property

Represents the data in an IDispatch field component.

Pascal

```
property Value: IDispatch;
```

C++

```
__property IDispatch Value;
```

Description

Use Value to read data directly from and write data directly to a field component at runtime. The Value property can affect the contents of a field that is not Visible (see page 7181).

See Also

Text (see page 7179)

DisplayText (see page 7162)

NewValue (see page 7174)

Visible (see page 7181)

AsVariant

39.1.38 TIndexDef Class

TIndexDef describes an index in a database table.

Class Hierarchy

Pascal

```
TIndexDef = class(TNamedItem);
```

C++

```
class TIndexDef : public TNamedItem;
```

File

DB

Description

Use the properties and methods of an index definition to:

Set the attributes of a new index in a table created with TTable . :: CreateTable or TClientDataSet. ::CreateDataSet.

Determine the name of an index.

Identify the fields that make up the index, or the dBASE expression that orders the index.

Determine the characteristics of the index.

Note: you cannot use TIndexDef objects in a dataset component's IndexDefs property to add indexes to already existing tables. To do that, use methods such as TTable

. :: AddIndex and TClientDataSet. ::AddIndex.

See Also

TIndexDefs (see page 7292)

IndexDefs

TTable (see page 10168)

AddIndex

TCustomClientDataSet (see page 8373)

Creating and Deleting Tables

39.1.38.1 TIndexDef Members

The following tables list the members exposed by TIndexDef.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 7286)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetDisplayName (see page 7359)	This is SetDisplayName, a member of class TNamedItem.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 7285)	Copies the properties of one index definition to another.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 7285)	Creates an instance of TIndexDef (see page 7283).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	FieldExpression (see page 7289)	Provides access to either a list of field names or an index expression.
R	ID (see page 2761)	A unique, permanent index for the item.

	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	CaseInsFields (see page 7287)	Specifies which fields of the index are case-insensitive.
	DescFields (see page 7288)	Specifies the fields of the index that are sorted in descending order.
	Expression (see page 7289)	Contains a dBASE key expression.
	Fields (see page 7290)	Identifies the fields that comprise the index.
	GroupingLevel (see page 7291)	Indicates the depth of grouping provided by the index.
	Name (see page 7359)	Specifies the name of the database entity defined by the TNamedItem (see page 7357) object.
	Options (see page 7291)	Describes the characteristics of the index.
	Source (see page 7292)	Contains the name of a dBASE maintained index.

39.1.38.2 TIndexDef.Create Constructor

Creates an instance of TIndexDef (see page 7283).

Pascal

```
constructor Create(Owner: TIndexDefs; const Name: string; const Fields: string; Options:
TIndexOptions); overload; reintroduce;
```

C++

```
__fastcall TIndexDef(TIndexDefs Owner, const AnsiString Name, const AnsiString Fields,
TIndexOptions Options);
```

Description

Do not explicitly create index definition objects. A TIndexDef (see page 7283) is created automatically by the dataset for each existing index of the table. When creating new indexes, use the TIndexDefs.AddIndexDef method.

See Also

Create (see page 2757)

AddIndex (see page 10184)

Add (see page 7295)

AddIndexDef (see page 7296)

39.1.38.3 TIndexDef Methods

The methods of the TIndexDef class are listed here.

Protected Methods

	Name	Description
	GetDisplayName (see page 7286)	Returns the name of the collection item as it appears in the collection editor.

Public Methods

	Name	Description
	Assign (see page 7285)	Copies the properties of one index definition to another.

39.1.38.3.1 TIndexDef.Assign Method

Copies the properties of one index definition to another.

Pascal

```
procedure Assign(ASource: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * ASource);
```

Description

Call Assign to copy another index definition. Assign checks whether the ASource parameter is another index definition, and, if so, copies the values of the Options (see page 7291), Name (see page 7359), Source (see page 7292), Expression (see page 7289), and Fields (see page 7290) properties.

If the source is not an index definition, Assign calls the inherited (Delphi) or parent class's (C++) method, which assigns properties from any object that specifies how to copy to an index definition in its AssignTo method.

See Also

Expression (see page 7289)

Fields (see page 7290)

Name (see page 7359)

Options (see page 7291)

Source (see page 7292)

AssignTo (see page 2931)

39.1.38.3.2 TIndexDef.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName (see page 2761)

GetNamePath (see page 2759)

39.1.38.4 TIndexDef Properties

The properties of the TIndexDef class are listed here.

Public Properties

	Name	Description
R	FieldExpression (see page 7289)	Provides access to either a list of field names or an index expression.

Published Properties

	Name	Description
	CaseInsFields (see page 7287)	Specifies which fields of the index are case-insensitive.
	DescFields (see page 7288)	Specifies the fields of the index that are sorted in descending order.
	Expression (see page 7289)	Contains a dBASE key expression.
	Fields (see page 7290)	Identifies the fields that comprise the index.
	GroupingLevel (see page 7291)	Indicates the depth of grouping provided by the index.
	Options (see page 7291)	Describes the characteristics of the index.
	Source (see page 7292)	Contains the name of a dBASE maintained index.

39.1.38.4.1 TIndexDef.CaseInsFields Property

Specifies which fields of the index are case-insensitive.

Pascal

```
property CaseInsFields: string;
```

C++

```
__property AnsiString CaseInsFields;
```

Description

Set CaseInsFields to a string that lists the names of fields in the index, separated by semicolons. The ordering imposed by the index on the fields specified in CaseInsFields is case-insensitive. Fields (see page 7290) in the index definition but not in the CaseInsFields list use case sensitive ordering (the default for the database type). It is possible that a single index can have fields using both case-insensitive and case-sensitive ordering.

For a field to be included in CaseInsFields, the field must be included in the fields on which the index is based. These fields are specified in the Fields (see page 7290) property of the TIndexDef (see page 7283) object.

Note: Not all database servers support both case-insensitive and case-sensitive field orderings in the same index. Consult the documentation for the particular database server used to determine whether this is actually supported. Some database systems require an index be designated case-insensitive as a whole (all of its base fields).

In the example below, the CaseInsFields property is given a list of two table fields for the index being created: LastName and Company.

```
with Table1 do begin
  ...
  with IndexDefs do begin
    with AddIndexDef do begin
      Name := 'MultiIndex'
      Fields := 'LastName;Company;State'
      Options := [ixUnique];
    end;
    Items[IndexDefs.Count – 1].CaseInFields := 'LastName;Company';
  end;
  ...
  CreateTable;
end;
```

...

```
  Table1->IndexDefs->Add("MultiIndex", "LastName;Company;State",
    TIndexOptions() << ixUnique);
  Table1->IndexDefs->Items[Table1->IndexDefs->Count-1]->CaseInsFields = "LastName;Company";
  ...
```

See Also

TIndexDefs (see page 7292)

Options (see page 7291)

IndexDefs (see page 10219)

AddIndexDef (see page 7296)

Fields (see page 7290)

39.1.38.4.2 TIndexDef.DescFields Property

Specifies the fields of the index that are sorted in descending order.

Pascal

```
property DescFields: string;
```

C++

```
__property AnsiString DescFields;
```

Description

Set DescFields to a string that lists the names of fields in the index, separated by semicolons. The ordering imposed by the index on the fields specified in DescFields is descending. Fields (see page 7290) in the index definition but not in the DescFields list use the default ascending order. It is possible for a single index to have fields using both ascending and descending ordering.

For a field to be included in DescFields, the field must be included in the fields on which the index is based. These fields are specified in the Fields (see page 7290) property of the TIndexDef (see page 7283) object.

Note: Not all database servers support both ascending and descending field orderings in the same index. Consult the documentation for the particular database server to determine whether this is actually supported.

In the example below, the DescFields property is given a list of two table fields for the index being created: TransDate and Company.

```
with Table1 do begin
  ...
  with IndexDefs do begin
    with AddIndexDef do begin
      Name := 'MultiIndex'
      Fields := 'TransDate;Company;State'
      Options := [ixUnique];
    end;
    Items[IndexDefs.Count – 1].DescInFields := 'TransDate;Company';
  end;
  ...
  CreateTable;
end;
...
Table1->IndexDefs->Add("MultiIndex","TransDate;Company;State",
  TIndexOptions() << ixUnique);
Table1->IndexDefs->Items[Table1->IndexDefs->Count-1]->DescFields = "TransDate;Company";
...
```

See Also

Options (see page 7291)

TIndexDefs (see page 7292)

IndexDefs (see page 10219)

AddIndexDef (see page 7296)

Fields (see page 7290)

39.1.38.4.3 TIndexDef.Expression Property

Contains a dBASE key expression.

Pascal

```
property Expression: string;
```

C++

```
__property AnsiString Expression;
```

Description

Read Expression to learn the key expression of a dBASE expression index. This property is only used on the Windows product, where there are components that can represent dBASE tables.

A key expression for a dBASE table is a field name, or a combination of field names, functions, and operators that specifies how the index orders records in the table. If the table is not a dBASE table or the index is not an expression index, Expression is an empty string and the Fields (see page 7290) property contains the field(s) for the index. When the index is a dBASE expression index, Expression contains a value and the Fields (see page 7290) property contains an empty string.

Note: For more information about key expressions, see your dBASE documentation.

See Also

Fields (see page 7290)

Options (see page 7291)

39.1.38.4.4 TIndexDef.FieldExpression Property

Provides access to either a list of field names or an index expression.

Pascal

```
property FieldExpression: string;
```

C++

```
__property AnsiString FieldExpression;
```

Description

FieldExpression provides access to the private data member on TIndexDef (see page 7283) which contains either a list of field names or an index expression. This data member is used to return the value for the Fields (see page 7290) property or the Expression (see page 7289) property depending on whether ixExpression is in Options (see page 7291).

This property is only used in the Windows product, where there are components that can represent dBASE tables. This property is used by TBDEDataSet but normally d Developers should use the Fields (see page 7290) or Expression (see page 7289) properties instead.

See Also

Expression (see page 7289)

Fields (see page 7290)

Options (see page 7291)

39.1.38.4.5 TIndexDef.Fields Property

Identifies the fields that comprise the index.

Pascal

```
property Fields: WideString;
```

C++

```
__property BSTR Fields;
```

Description

When creating an index, set Fields to a list of fields separated by semicolons (no semicolon is required if there is only one field in the index). The order of field names in the string determines the order of fields in the index.

When inspecting already-existing indexes, read Fields to determine the fields that make up the index. The string returned can be either the names of the fields, or the numbers of the field. When numbers are used, the numbers correspond to the physical field numbers in the table. Multiple fields in the string are separated by semicolons (;).

Note:

C++ Examples:

```
/*
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
*/
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2->Active = False;
  // Get the current available indices
  CDS2->IndexDefs->Update();
  // Find a field named "Field2"
  for (int I = 0; I < CDS2->IndexDefs->Count; I++)
    if (CDS2->IndexDefs->Items[I]->Fields == "Field2")
    {
      // set that index as the current index for the dataset}
      CDS2->IndexName = CDS2->IndexDefs->Items[I]->Name;
    }
  CDS2->Active = True;
}
```

Delphi Examples:

```
{
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
}
procedure TForm1.Button3Click(Sender: TObject);
var
  I : Integer;
begin
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2.Active := False;
  { Get the current available indices }
  CDS2.IndexDefs.Update;
  { Find a field named 'Field2' }
  for I := 0 to CDS2.IndexDefs.Count - 1 do
    if CDS2.IndexDefs.Items[I].Fields = 'Field2' then
```

```
      begin
        { set that index as the current index for the dataset}
        CDS2.IndexName := CDS2.IndexDefs.Items[I].Name;
      end;
    CDS2.Active := True;
  end;
```

See Also

Expression (see page 7289)

Options (see page 7291)

FieldNo (see page 7166)

FieldName (see page 7164)

Add (see page 7295)

39.1.38.4.6 TIndexDef.GroupingLevel Property

Indicates the depth of grouping provided by the index.

Pascal

```
property GroupingLevel: Integer;
```

C++

```
__property int GroupingLevel;
```

Description

Grouping level specifies the level of grouping information maintained by the index. Grouping is only supported on client datasets.

See Also

GroupingLevel (see page 8277)

39.1.38.4.7 TIndexDef.Options Property

Describes the characteristics of the index.

Pascal

```
property Options: TIndexOptions;
```

C++

```
__property TIndexOptions Options;
```

Description

When creating a new index, use Options to specify the attributes of the index. Options can contain zero or more of the TIndexOption (see page 7617) constants ixPrimary, ixUnique, ixDescending, ixCaseInsensitive, and ixExpression.

When inspecting the definitions of existing indexes, read Options to determine the option(s) used to create the index.

Note: Different database drivers may support only a subset of these options. For example, with dBASE tables, ixCaseInsensitive is not supported. Including an option that is not valid for a table will raise an exception.

Note: Primary indexes are, by definition unique. If setting the Options value ixPrimary, you need not also use ixUnique.

See Also

Expression (see page 7289)

39.1.38.4.8 TIndexDef.Source Property

Contains the name of a dBASE maintained index.

Pascal

```
property Source: string;
```

C++

```
__property AnsiString Source;
```

Description

When the table uses dBASE .MDX indexes, Name (see page 7359) indicates the index tag name and Source is the name of the index file. Source is applicable to dBASE table indexes only.

Source is not used on the Linux platform. This property is available for the Windows product, where there are components that represent dBASE tables.

See Also

Name (see page 7359)

39.1.39 TIndexDefs Class

TIndexDefs holds the list of available index definitions for a table.

Class Hierarchy

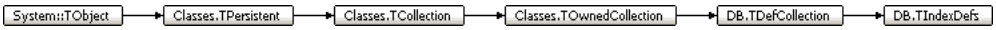

Pascal

```
TIndexDefs = class(TDefCollection);
```

C++

```
class TIndexDefs : public TDefCollection;
```

File

DB

Description

TTable and TClientDataSet components use TIndexDefs to implement their IndexDefs property. These index definitions can be used to obtain information about the indexes defined for the underlying data or, when the component is used to create a table, to create the indexes of that table.

Use the properties and methods of TIndexDefs to:

Access a specific index definition.

Determine how many index definitions this TIndexDefs object contains.

Add (see page 7295) or delete index definitions.

Copy all index definitions from one TIndexDefs to another.

Note: you cannot use TIndexDef (see page 7283) objects in a dataset component's IndexDefs property to add indexes to already existing tables. To do that, use methods such as AddIndex.

See Also

TIndexDef (see page 7283)

TCustomClientDataSet (see page 8373)

IndexDefs

AddIndex

39.1.39.1 TIndexDefs Members

The following tables list the members exposed by TIndexDefs.

Protected Events

	Name	Description
	OnUpdate (see page 7111)	This is OnUpdate, a member of class TDefCollection.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	DoUpdate (see page 7107)	This is DoUpdate, a member of class TDefCollection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2915)	Returns the owner of an object.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	SetItemName (see page 7109)	Initializes the name of a newly inserted collection item.
	UpdateDefs (see page 7109)	This is UpdateDefs, a member of class TDefCollection.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 7295)	Creates a new index definition and adds it to the Items (see page 7300) property.
	AddIndexDef (see page 7296)	Creates a new index definition object and adds it to the Items (see page 7300) property.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 7295)	Creates an instance of TIndexDefs (see page 7292).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
	Destroy (see page 2735)	Destroys the collection and each item in it.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Find (see page 7297)	Returns the TIndexDef (see page 7283) object for the index definition with a specified name.
	FindIndexForFields (see page 7298)	Locates an index definition in the Items (see page 7300) array.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetIndexForFields (see page 7298)	Locates an index definition in the Items (see page 7300) array.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItemNames (see page 7108)	This is the overview for the GetItemNames method overload.
	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	IndexOf (see page 7108)	Finds the index number of a definition in the Items array from its name.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Update (see page 7299)	Refreshes index definitions in Items (see page 7300) to reflect the current state of the dataset.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	DataSet (see page 7110)	Indicates the dataset that uses the TDefCollection (see page 7104) object to manage its definition objects.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 7300)	Lists the index definitions that describe each index of a dataset.
	Updated (see page 7110)	Indicates whether the definition objects in the Items array match the corresponding instantiated objects in the dataset.

39.1.39.2 TIndexDefs.Create Constructor

Creates an instance of TIndexDefs (see page 7292).

Pascal

```
constructor Create(ADataSet: TDataSet);
```

C++

```
__fastcall TIndexDefs(TDataSet ADataSet);
```

Description

Applications should not instantiate TIndexDefs (see page 7292). TIndexDefs (see page 7292) objects are created automatically by the dataset that uses them (even if the dataset currently has no indexes).

See Also

Items (see page 7300)

IndexDefs (see page 10219)

39.1.39.3 TIndexDefs Methods

The methods of the TIndexDefs class are listed here.

Public Methods

	Name	Description
	Add (see page 7295)	Creates a new index definition and adds it to the Items (see page 7300) property.
	AddIndexDef (see page 7296)	Creates a new index definition object and adds it to the Items (see page 7300) property.
	Find (see page 7297)	Returns the TIndexDef (see page 7283) object for the index definition with a specified name.
	FindIndexForFields (see page 7298)	Locates an index definition in the Items (see page 7300) array.
	GetIndexForFields (see page 7298)	Locates an index definition in the Items (see page 7300) array.
	Update (see page 7299)	Refreshes index definitions in Items (see page 7300) to reflect the current state of the dataset.

39.1.39.3.1 TIndexDefs.Add Method

Creates a new index definition and adds it to the Items (see page 7300) property.

Pascal

```
procedure Add(const Name: string; const Fields: string; Options: TIndexOptions);
```

C++

```
__fastcall Add(const AnsiString Name, const AnsiString Fields, TIndexOptions Options);
```

Description

Add is provided for backward compatibility. The recommended way to add new index definitions before creating a table is to use the AddIndexDef (see page 7296) method. To add an index to an existing table, use the AddIndex method of the dataset instead. To change the array of indexes when a new index has been added to the dataset, use the Update (see page 7299) method instead.

Add creates an index definition and assigns the values passed in the Name, Fields, and Options parameters to the TIndexDef (see page 7283) properties of the same names. If an index definition with same name already exists, Add raises an EDatabaseError (see page 6731) exception.

See Also

Items (see page 7300)

TIndexDef (see page 7283)

Fields (see page 7290)

Name (see page 7359)

Options (see page 7291)

AddIndex (see page 10184)

Update (see page 7299)

AddIndexDef (see page 7296)

39.1.39.3.2 TIndexDefs.AddIndexDef Method

Creates a new index definition object and adds it to the Items (see page 7300) property.

Pascal

```
function AddIndexDef: TIndexDef;
```

C++

```
__fastcall TIndexDef AddIndexDef();
```

Description

Call AddIndexDef to add index definitions when creating a database table. AddIndexDef returns the new TIndexDef (see page 7283) object that was added to the IndexDefs collection of index definitions. After calling AddIndexDef, set the properties of the newly added TIndexDef (see page 7283) object to specify the name, type, and options of index that will be created.

Note: AddIndexDef is only valid in the context of creating index definitions for a table to be created with a subsequent call to the CreateTable

or CreateDataSet method. It cannot be used to add indexes to an already-existing table.

C++ Examples:

```
/*
The following code creates and activates a client dataset in
the form's OnCreate event handler.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TFieldDefs *pDefs = CDS2->FieldDefs;
```

```
  TFieldDef *pDef = pDefs->AddFieldDef();
  pDef->DataType = ftInteger;
  pDef->Name = "Field1";

  pDef = pDefs->AddFieldDef();
  pDef->DataType = ftString;
  pDef->Size = 10;
  pDef->Name = "Field2";

  TIndexDef *pIDef = CDS2->IndexDefs->AddIndexDef();
  pIDef->Fields = "Field1";
  pIDef->Name = "IntIndex";

  CDS2->CreateDataSet();
}
```

Delphi Examples:

```
{
The following code creates and activates a client dataset in
the form's OnCreate event handler.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  with CDS2 do
  begin
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftInteger;
      Name := 'Field1';
    end;
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftString;
      Size := 10;
      Name := 'Field2';
    end;
    with IndexDefs.AddIndexDef do
    begin
      Fields := 'Field1';
      Name := 'IntIndex';
    end;
    CreateDataSet;
  end;
end;
```

See Also

Items (see page 7300)

CreateTable (see page 10189)

CreateDataSet (see page 8401)

Creating and Deleting Tables

39.1.39.3.3 TIndexDefs.Find Method

Returns the TIndexDef (see page 7283) object for the index definition with a specified name.

Pascal

```
function Find(const Name: string): TIndexDef;
```

C++

```
__fastcall TIndexDef Find(const AnsiString Name);
```

Description

Call Find to obtain information about a particular index definition. Specify the name of the index as the value of the Name parameter.

Find raises an EDatabaseError (see page 6731) exception if there is no index definition with the given name.

Note: To locate an index definition based on the indexed fields rather than the name of the index, use the FindIndexForFields (see page 7298) method.

See Also

FindIndexForFields (see page 7298)

Items (see page 7300)

IndexOf (see page 7108)

39.1.39.3.4 TIndexDefs.FindIndexForFields Method

Locates an index definition in the Items (see page 7300) array.

Pascal

```
function FindIndexForFields(const Fields: string): TIndexDef;
```

C++

```
__fastcall TIndexDef FindIndexForFields(const AnsiString Fields);
```

Description

Use FindIndexForFields to search for the index that starts with the fields specified in the Fields parameter. If the index is constructed using more than one field, separate the fields with semicolons.

FindIndexForFields returns the index definition that exactly matches the fields specified (in the order specified), if one exists. If no exact match is found, FindIndexForFields returns the first index that begins with the indicated fields. If no match is found, FindIndexForFields raises an EDatabaseError (see page 6731) exception.

Note: FindIndexForFields can only locate ascending indexes. To include descending indexes in the search, use GetIndexForFields (see page 7298).

Note: To restrict the search to case-insensitive indexes, use the GetIndexForFields (see page 7298) method.

See Also

GetIndexForFields (see page 7298)

Find (see page 7297)

Items (see page 7300)

Fields (see page 7290)

39.1.39.3.5 TIndexDefs.GetIndexForFields Method

Locates an index definition in the Items (see page 7300) array.

Pascal

```
function GetIndexForFields(const Fields: string; CaseInsensitive: Boolean): TIndexDef;
```

C++

```
__fastcall TIndexDef GetIndexForFields(const AnsiString Fields, Boolean CaseInsensitive);
```

Description

Use GetIndexForFields to search for the index that starts with the fields specified in the Fields parameter. If more than one field comprises the index, separate the fields with semicolons. Set the CaseInsensitive parameter to true to restrict the search to only the case insensitive indexes in the Items (see page 7300) array.

GetIndexForFields returns the index definition that exactly matches the specified fields in the order specified, if it exists. If no exact match is found, GetIndexForFields returns the first index that begins with the indicated fields. If no match can be found, GetIndexForFields returns nil (Delphi) or NULL (C++).

Note: When CaseInsensitive is true, descending indexes are considered even if they are case-sensitive.

See Also

FindIndexForFields (see page 7298)

Find (see page 7297)

Items (see page 7300)

Fields (see page 7290)

39.1.39.3.6 TIndexDefs.Update Method

Refreshes index definitions in Items (see page 7300) to reflect the current state of the dataset.

Pascal

```
procedure Update; reintroduce;
```

C++

```
__fastcall Update();
```

Description

Call Update when the set of indexes for the dataset may have changed. Update refreshes the index definitions without opening the dataset.

C++ Examples:

```
/*
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
*/
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2->Active = False;
  // Get the current available indices
  CDS2->IndexDefs->Update();
  // Find a field named "Field2"
  for (int I = 0; I < CDS2->IndexDefs->Count; I++)
    if (CDS2->IndexDefs->Items[I]->Fields == "Field2")
    {
      // set that index as the current index for the dataset}
      CDS2->IndexName = CDS2->IndexDefs->Items[I]->Name;
    }
  CDS2->Active = True;
}
```

Delphi Examples:

```
{
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
}
procedure TForm1.Button3Click(Sender: TObject);
var
  I : Integer;
begin
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2.Active := False;
  { Get the current available indices }
  CDS2.IndexDefs.Update;
  { Find a field named 'Field2' }
  for I := 0 to CDS2.IndexDefs.Count - 1 do
    if CDS2.IndexDefs.Items[I].Fields = 'Field2' then
    begin
      { set that index as the current index for the dataset}
      CDS2.IndexName := CDS2.IndexDefs.Items[I].Name;
    end;
  CDS2.Active := True;
end;
```

See Also

Items (see page 7300)

TIndexDef (see page 7283)

39.1.39.4 TIndexDefs Properties

The properties of the TIndexDefs class are listed here.

Public Properties

	Name	Description
	Items (see page 7300)	Lists the index definitions that describe each index of a dataset.

39.1.39.4.1 TIndexDefs.Items Property

Lists the index definitions that describe each index of a dataset.

Pascal

```
property Items [Index: Integer]: TIndexDef;
```

C++

```
__property TIndexDef Items[int Index];
```

Description

Use Items to access a particular index definition. Specify the index definition to access with the Index parameter. Index identifies the index definition's position in the list of index definitions, in the range 0 to Count - 1.

```
IdxFldsVar := Table1.IndexDefs.Items[i].Fields;
IdxFldsVar = Table1->IndexDefs->Items[i]->Fields;
```

C++ Examples:

```
/*
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
```

```
*/
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2->Active = False;
  // Get the current available indices
  CDS2->IndexDefs->Update();
  // Find a field named "Field2"
  for (int I = 0; I < CDS2->IndexDefs->Count; I++)
    if (CDS2->IndexDefs->Items[I]->Fields == "Field2")
    {
      // set that index as the current index for the dataset}
      CDS2->IndexName = CDS2->IndexDefs->Items[I]->Name;
    }
  CDS2->Active = True;
}
```

Delphi Examples:

```
{
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
}
procedure TForm1.Button3Click(Sender: TObject);
var
  I : Integer;
begin
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2.Active := False;
  { Get the current available indices }
  CDS2.IndexDefs.Update;
  { Find a field named 'Field2' }
  for I := 0 to CDS2.IndexDefs.Count - 1 do
    if CDS2.IndexDefs.Items[I].Fields = 'Field2' then
    begin
      { set that index as the current index for the dataset}
      CDS2.IndexName := CDS2.IndexDefs.Items[I].Name;
    end;
  CDS2.Active := True;
end;
```

See Also

AddIndexDef (see page 7296)

Clear (see page 2741)

Count (see page 2749)

operator_sb

39.1.40 TIntegerField Class

TIntegerField represents a signed 32-bit integer field in a dataset.

Class Hierarchy

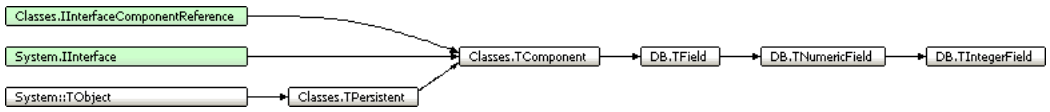

Pascal

```
TIntegerField = class(TNumericField);
```

C++

```
class TIntegerField : public TNumericField;
```

File

DB

Description

TIntegerField encapsulates the fundamental behavior common to fields that contain integer data. Integer fields can hold values in the range -2,147,483,648 to 2,147,483,647, and are frequently used to hold values that are large, signed whole numbers. In addition to the properties, methods, and events common to all fields that are introduced by the TField (see page 7115) class, TIntegerField introduces new properties to convert between integer values and other data types, and to limit the range of valid integers that the field will permit.

If you use the Fields editor at design time to create a persistent field component for the integer field, you can access it by name at runtime. When using dynamic field components, you can access the TIntegerField instance using the dataset's Fields property or FieldByName method.

TIntegerField is the direct ancestor of three other field types that represent whole numbers: TAutoIncField (see page 6768), TSmallIntField, and TWordField (see page 7560).

Delphi Examples:

```
{
This example requires two ClientDataSets, two DBGrids, two
DataSources and two DBNavigators.  Name the two
ClientDataSets CDS and CDS2. Set the DataSet property in the
first DataSource to CDS and name that DataSource DS.  Set the
DataSet property in the second DataSource to CDS2 and name
that DataSource DS2.  Set the DataSource in the first
DBNavigator to DS and the DataSource in the second DBNavigator
to DS2.  Set the DataSource in the first DBGrid to DS and the
DataSource in the second DBGrid to DS2.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
type to Integer.  The file used to load CDS must have an
attribute name of HOST with an attribute type of string and
an attribute name of PORTA with an attribute type of integer.
Add DB, DBCtrls, Grids, DBGrids, DBClient and SwSystem to the
uses clause.  "LoadFormFile" to start with the file, or
"CreateDataSet" to start from scratch.  You must "CloseDataSet"
after a "LoadFromFile" before you can do another "CreateDataSet".
}
{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button1Click(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.SaveToFile(GetCurrentDir + '\CDS.XML', dfXML);
{$ELSE}
  CDS.SaveToFile(gsAppPath + 'CDS.XML', dfXML);
{$ENDIF}
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  CDS.CreateDataSet;
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  CDS2.Data := CDS.Data;
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  CDS.Close;
end;
```

See Also

TDataSet (see page 6894)

TDataSet (see page 6894)

TDataSet (see page 6894)

TField (see page 7115)

TAutoIncField (see page 6768)

TSmallintField (see page 7461)

TWordField (see page 7560)

Defining a Data Field

Using Default Formatting for Numeric

39.1.40.1 TIntegerField Members

The following tables list the members exposed by TIntegerField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.

	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsFloat (see page 7310)	This is GetAsFloat, a member of class TIntegerField.
V	GetAsInteger (see page 7310)	This is GetAsInteger, a member of class TIntegerField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsString (see page 7311)	This is GetAsString, a member of class TIntegerField.
V	GetAsVariant (see page 7311)	This is GetAsVariant, a member of class TIntegerField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
V	GetDataSize (see page 7311)	This is GetDataSize, a member of class TIntegerField.
V	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
V	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
V	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetSize (see page 7135)	This is GetSize, a member of class TField.
V	GetText (see page 7311)	This is GetText, a member of class TIntegerField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 7312)	This is GetValue, a member of class TIntegerField.
V	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	RangeError (see page 7368)	This is RangeError, a member of class TNumericField.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 7138)	This is ReadState, a member of class TField.

	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
V	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
V	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
V	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
V	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
V	SetAsFloat (see page 7312)	This is SetAsFloat, a member of class TIntegerField.
V	SetAsInteger (see page 7312)	This is SetAsInteger, a member of class TIntegerField.
V	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
V	SetAsString (see page 7312)	This is SetAsString, a member of class TIntegerField.
V	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
V	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayFormat (see page 7368)	This is SetDisplayFormat, a member of class TNumericField.
	SetEditFormat (see page 7368)	This is SetEditFormat, a member of class TNumericField.
V	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
V	SetSize (see page 7143)	This is SetSize, a member of class TField.
V	SetText (see page 7144)	This is SetText, a member of class TField.
V	SetVarValue (see page 7312)	This is SetVarValue, a member of class TIntegerField.
V	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 7126)	Sets the value of the field to NULL.
V	Create (see page 7309)	Creates an instance of TIntegerField (see page 7301).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.

R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7314)	Represents the value of the integer field.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	Alignment (see page 7369)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayFormat (see page 7369)	Determines how a numeric field's value is formatted for display in a data-aware control.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	EditFormat (see page 7371)	Determines how a numeric field's value is formatted when it is being edited in a data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.

	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	MaxValue (see page 7313)	Limits the maximum value in the integer field.
	MinValue (see page 7313)	Limits the minimum value in the integer field.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.40.2 TIntegerField.Create Constructor

Creates an instance of TIntegerField (see page 7301).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TIntegerField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TIntegerField (see page 7301). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftInteger, MinValue (see page 7313) to -2147483648, and MaxValue (see page 7313) to 2147483647. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TIntegerField (see page 7301). After instantiating a TIntegerField (see page 7301), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TIntegerField (see page 7301) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TIntegerField (see page 7301) with a dataset component by setting its DataSet (see page 7157) property.

The example below creates a TIntegerField (see page 7301) object for a field named Quantity accessed through a TSQLDataSet named SQLDataSet1.

```
var
  T: TIntegerField;
begin
  SQLDataSet1.Close;
  T := TIntegerField.Create(SQLDataSet1);
  T.FieldName := 'Quantity';
  T.Name := SQLDataSet1.Name + T.FieldName;
  T.Index := SQLDataSet1.FieldCount;
  T.DataSet := SQLDataSet1;
  SQLDataSet1.FieldDefs.UpDate;
  SQLDataSet1.Open;
end;
SQLDataSet1->Close();
TIntegerField *T = new TIntegerField(SQLDataSet1);
T->FieldName = "Quantity";
```

```
T->Name = SQLDataSet1->Name + T->FieldName;
T->Index = SQLDataSet1->FieldCount;
T->DataSet = SQLDataSet1;
SQLDataSet1->FieldDefs->UpDate();
SQLDataSet1->Open();
```

See Also

Destroy (see page 7122)

DataType (see page 7159)

MinValue (see page 7313)

MaxValue (see page 7313)

Owner (see page 2807)

39.1.40.3 TIntegerField Methods

The methods of the TIntegerField class are listed here.

Protected Methods

	Name	Description
	GetAsFloat (see page 7310)	This is GetAsFloat, a member of class TIntegerField.
	GetAsInteger (see page 7310)	This is GetAsInteger, a member of class TIntegerField.
	GetAsString (see page 7311)	This is GetAsString, a member of class TIntegerField.
	GetAsVariant (see page 7311)	This is GetAsVariant, a member of class TIntegerField.
	GetDataSize (see page 7311)	This is GetDataSize, a member of class TIntegerField.
	GetText (see page 7311)	This is GetText, a member of class TIntegerField.
	GetValue (see page 7312)	This is GetValue, a member of class TIntegerField.
	SetAsFloat (see page 7312)	This is SetAsFloat, a member of class TIntegerField.
	SetAsInteger (see page 7312)	This is SetAsInteger, a member of class TIntegerField.
	SetAsString (see page 7312)	This is SetAsString, a member of class TIntegerField.
	SetVarValue (see page 7312)	This is SetVarValue, a member of class TIntegerField.

39.1.40.3.1 TIntegerField.GetAsFloat Method

Pascal

```
function GetAsFloat: Double; override;
```

C++

```
virtual __fastcall Double GetAsFloat();
```

Description

This is GetAsFloat, a member of class TIntegerField.

39.1.40.3.2 TIntegerField.GetAsInteger Method

Pascal

```
function GetAsInteger: Longint; override;
```

C++

```
virtual __fastcall Longint GetAsInteger();
```

Description

This is GetAsInteger, a member of class TIntegerField.

39.1.40.3.3 TIntegerField.GetAsString Method

Pascal

```
function GetAsString: string; override;
```

C++

```
virtual __fastcall AnsiString GetAsString();
```

Description

This is GetAsString, a member of class TIntegerField.

39.1.40.3.4 TIntegerField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TIntegerField.

39.1.40.3.5 TIntegerField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; override;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TIntegerField.

39.1.40.3.6 TIntegerField.GetText Method

Pascal

```
procedure GetText(var Text: string; DisplayText: Boolean); override;
```

C++

```
virtual __fastcall GetText(AnsiString Text, Boolean DisplayText);
```

Description

This is GetText, a member of class TIntegerField.

39.1.40.3.7 TIntegerField.GetValue Method

Pascal

```
function GetValue(var Value: Longint): Boolean;
```

C++

```
__fastcall Boolean GetValue(Longint Value);
```

Description

This is GetValue, a member of class TIntegerField.

39.1.40.3.8 TIntegerField.SetAsFloat Method

Pascal

```
procedure SetAsFloat(Value: Double); override;
```

C++

```
virtual __fastcall SetAsFloat(Double Value);
```

Description

This is SetAsFloat, a member of class TIntegerField.

39.1.40.3.9 TIntegerField.SetAsInteger Method

Pascal

```
procedure SetAsInteger(Value: Longint); override;
```

C++

```
virtual __fastcall SetAsInteger(Longint Value);
```

Description

This is SetAsInteger, a member of class TIntegerField.

39.1.40.3.10 TIntegerField.SetAsString Method

Pascal

```
procedure SetAsString(const Value: string); override;
```

C++

```
virtual __fastcall SetAsString(const AnsiString Value);
```

Description

This is SetAsString, a member of class TIntegerField.

39.1.40.3.11 TIntegerField.SetVarValue Method

Pascal

```
procedure SetVarValue(const Value: Variant); override;
```

C++

```
virtual __fastcall SetVarValue(const Variant Value);
```

Description

This is SetVarValue, a member of class TIntegerField.

39.1.40.4 TIntegerField Properties

The properties of the TIntegerField class are listed here.

Public Properties

	Name	Description
	Value (see page 7314)	Represents the value of the integer field.

Published Properties

	Name	Description
	MaxValue (see page 7313)	Limits the maximum value in the integer field.
	MinValue (see page 7313)	Limits the minimum value in the integer field.

39.1.40.4.1 TIntegerField.MaxValue Property

Limits the maximum value in the integer field.

Pascal

```
property MaxValue: Longint;
```

C++

```
__property Longint MaxValue;
```

Description

Use MaxValue to get or set the maximum value that can be entered into the field. If a value greater than MaxValue is entered, an EDatabaseError (see page 6731) exception is raised.

If the value of MaxValue is 0, the integer field can have a value up to the maximum value that can be represented by the native database type. For 32-bit integer fields, this value is 2147483647.

See Also

MinValue (see page 7313)

39.1.40.4.2 TIntegerField.MinValue Property

Limits the minimum value in the integer field.

Pascal

```
property MinValue: Longint;
```

C++

```
__property Longint MinValue;
```

Description

Use MinValue to get or set the minimum value that can be entered into the field. If a value smaller than MinValue is entered, an EDatabaseError (see page 6731) exception is raised.

If the value of MinValue is 0, the integer field can have a value as small as the native database type can represent. For 32-bit integer fields, this minimum value is -2147483648.

See Also

MaxValue (see page 7313)

39.1.40.4.3 TIntegerField.Value Property

Represents the value of the integer field.

Pascal

```
property Value: Longint;
```

C++

```
__property Longint Value;
```

Description

Use Value to read data directly from and write data directly to an integer field. Using the Value property is the same as using the AsInteger (see page 7149) property. Use Value when you know the field component is a 32-bit integer field. Use AsInteger (see page 7149) when working with a generic field component.

See Also

AsInteger

39.1.41 TInterfaceField Class

TInterfaceField represents an interface field in a dataset.

Class Hierarchy

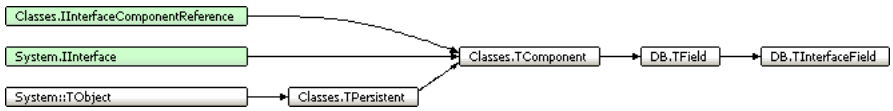

Pascal

```
TInterfaceField = class(TField);
```

C++

```
class TInterfaceField : public TField;
```

File

DB

Description

TInterfaceField encapsulates the fundamental behavior common to fields that contain pointers to interfaces (IUnknown) as data. TInterfaceField reimplements the Value (see page 7322) and AsVariant (see page 7152) properties to adapt them to accessing fields containing interfaces.

If you use the Fields editor at design time to create a persistent field component for the interface field, you can access it by name at runtime. When using dynamic field components, you can access the TInterfaceField instance using the dataset's Fields property or FieldByName method.

See Also

TDataSet (see page 6894)

TMemoField (see page 7348)

TDataSet (see page 6894)

TDataSet (see page 6894)

39.1.41.1 TInterfaceField Members

The following tables list the members exposed by TInterfaceField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7321)	This is CheckTypeSize, a member of class TInterfaceField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
V	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
V	GetAsVariant (see page 7321)	This is GetAsVariant, a member of class TInterfaceField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
V	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
V	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
V	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
V	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 7322)	This is GetValue, a member of class TInterfaceField.
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetValue (see page 7322)	This is SetValue, a member of class TInterfaceField.
	SetVarValue (see page 7322)	This is SetVarValue, a member of class TInterfaceField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 7126)	Sets the value of the field to NULL.
V	Create (see page 7320)	Creates and initializes an interface field component.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.

R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7322)	Represents the data in an interface field component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.

	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.41.2 TInterfaceField.Create Constructor

Creates and initializes an interface field component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TInterfaceField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TInterfaceField (see page 7314). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftInterface. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TIntegerField (see page 7301). After instantiating a TInterfaceField (see page 7314), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TInterfaceField (see page 7314) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TInterfaceField (see page 7314) with a dataset component by setting its DataSet (see page 7157) property to the name of the dataset component.

The example below creates a TInterfaceField (see page 7314) object for a field named Interface accessed through a TQuery named Query1.

```
var
  T: TInterfaceField;
begin
  Query1.Close;
  T := TInterfaceField.Create(Query1);
  T.FieldName := 'Interface';
  T.Name := Query1.Name + T.FieldName;
  T.Index := Query1.FieldCount;
  T.DataSet := Query1;
  Query1.FieldDefs.UpDate;
  Query1.Open;
end;
TInterfaceField *T;
Query1->Close();
T = new TInterfaceField(Query1);
T->FieldName = "Interface";
T->Name = Query1->Name + T->FieldName;
T->Index = Query1->FieldCount;
T->DataSet = Query1;
Query1->FieldDefs->UpDate();
Query1->Open();
```

See Also

Visible (see page 7181)

Owner (see page 2807)

39.1.41.3 TInterfaceField Methods

The methods of the TInterfaceField class are listed here.

Protected Methods

	Name	Description
	CheckTypeSize (see page 7321)	This is CheckTypeSize, a member of class TInterfaceField.
	GetAsVariant (see page 7321)	This is GetAsVariant, a member of class TInterfaceField.
	GetValue (see page 7322)	This is GetValue, a member of class TInterfaceField.
	SetValue (see page 7322)	This is SetValue, a member of class TInterfaceField.
	SetVarValue (see page 7322)	This is SetVarValue, a member of class TInterfaceField.

39.1.41.3.1 TInterfaceField.CheckTypeSize Method

Pascal

```
class procedure CheckTypeSize(Value: Integer); override;
```

C++

```
virtual __fastcall CheckTypeSize(int Value);
```

Description

This is CheckTypeSize, a member of class TInterfaceField.

39.1.41.3.2 TInterfaceField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TInterfaceField.

39.1.41.3.3 TInterfaceField.GetValue Method

Pascal

```
function GetValue: IUnknown;
```

C++

```
__fastcall IUnknown GetValue();
```

Description

This is GetValue, a member of class TInterfaceField.

39.1.41.3.4 TInterfaceField.SetValue Method

Pascal

```
procedure SetValue(const Value: IUnknown);
```

C++

```
__fastcall SetValue(const IUnknown Value);
```

Description

This is SetValue, a member of class TInterfaceField.

39.1.41.3.5 TInterfaceField.SetVarValue Method

Pascal

```
procedure SetVarValue(const Value: Variant); override;
```

C++

```
virtual __fastcall SetVarValue(const Variant Value);
```

Description

This is SetVarValue, a member of class TInterfaceField.

39.1.41.4 TInterfaceField Properties

The properties of the TInterfaceField class are listed here.

Public Properties

	Name	Description
	Value (see page 7322)	Represents the data in an interface field component.

39.1.41.4.1 TInterfaceField.Value Property

Represents the data in an interface field component.

Pascal

```
property Value: IUnknown;
```

C++

```
__property IUnknown Value;
```

Description

Use Value to read data directly from and write data directly to a field component at runtime. The Value property can affect the contents of a field even if it is not Visible (see page 7181).

See Also

Text (see page 7179)

DisplayText (see page 7162)

AsVariant

Visible (see page 7181)

39.1.42 TLargeintField Class

A TLargeintField object represents a large integer field in a dataset.

Class Hierarchy

Pascal

```
TLargeintField = class(TNumericField);
```

C++

```
class TLargeintField : public TNumericField;
```

File

DB

Description

TLargeintField fields are database fields that contain large (64-bit) integer values.

If you use the Fields editor at design time to create a persistent field component for the large integer field, you can access it by name at runtime. When using dynamic field components, you can access the TIntegerField (see page 7301) instance using the dataset's Fields property or FieldByName method.

See Also

TDataSet (see page 6894)

TDataSet (see page 6894)

TDataSet (see page 6894)

TField (see page 7115)

TAutoIncField (see page 6768)

TSmallintField (see page 7461)

TWordField (see page 7560)

39.1.42.1 TLargeintField Members

The following tables list the members exposed by TLargeintField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsFloat (see page 7331)	This is GetAsFloat, a member of class TLargeintField.
V	GetAsInteger (see page 7331)	This is GetAsInteger, a member of class TLargeintField.
	GetAsLargeint (see page 7331)	This is GetAsLargeint, a member of class TLargeintField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsString (see page 7331)	This is GetAsString, a member of class TLargeintField.
V	GetAsVariant (see page 7331)	This is GetAsVariant, a member of class TLargeintField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
V	GetDataSize (see page 7332)	This is GetDataSize, a member of class TLargeintField.
V	GetDefaultWidth (see page 7332)	This is GetDefaultWidth, a member of class TLargeintField.
V	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
V	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7332)	This is GetText, a member of class TLargeintField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 7332)	This is GetValue, a member of class TLargeintField.
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	RangeError (see page 7368)	This is RangeError, a member of class TNumericField.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7333)	This is SetAsFloat, a member of class TLargeintField.
	SetAsInteger (see page 7333)	This is SetAsInteger, a member of class TLargeintField.
	SetAsLargeint (see page 7333)	This is SetAsLargeint, a member of class TLargeintField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7333)	This is SetAsString, a member of class TLargeintField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayFormat (see page 7368)	This is SetDisplayFormat, a member of class TNumericField.
	SetEditFormat (see page 7368)	This is SetEditFormat, a member of class TNumericField.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 7333)	This is SetVarValue, a member of class TLargeintField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7126)	Sets the value of the field to NULL.
	Create (see page 7329)	Create creates an instance of TLargeintField (see page 7323).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsLargeInt (see page 7334)	Represents the field value as a large integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.

	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7335)	Represents the value of the large integer field.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	Alignment (see page 7369)	Determines how the field's data is displayed within a data-aware control.

	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayFormat (see page 7369)	Determines how a numeric field's value is formatted for display in a data-aware control.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	EditFormat (see page 7371)	Determines how a numeric field's value is formatted when it is being edited in a data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	MaxValue (see page 7335)	Limits the maximum value in the large integer field.
	MinValue (see page 7335)	Limits the minimum value in the large integer field.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.42.2 TLargeintField.Create Constructor

Create creates an instance of TLargeintField (see page 7323).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TLargeintField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TLargeintField (see page 7323). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftLargeint, and valid characters to '+', '-', and '0'..'9'. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TLargeintField (see page 7323). After instantiating a TLargeintField (see page 7323), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TLargeintField (see page 7323) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TLargeintField (see page 7323) with a dataset component by setting its DataSet (see page 7157) property.

The example below creates a TLargeintField (see page 7323) object for a field named Population accessed through a TSQLDataSet named SQLDataSet1.

```
var
  T: TLargeintField;
begin
  SQLDataSet1.Close;
  T := TLargeintField.Create(SQLDataSet1);
  T.FieldName := 'Population';
  T.Name := SQLDataSet1.Name + T.FieldName;
  T.Index := SQLDataSet1.FieldCount;
  T.DataSet := SQLDataSet1;
  SQLDataSet1.FieldDefs.UpDate;
  SQLDataSet1.Open;
end;
SQLDataSet1->Close();
TLargeintField *T = new TLargeintField(SQLDataSet1);
T->FieldName = "Population";
T->Name = SQLDataSet1->Name + T->FieldName;
T->Index = SQLDataSet1->FieldCount;
T->DataSet = SQLDataSet1;
SQLDataSet1->FieldDefs->UpDate();
SQLDataSet1->Open();
```

See Also

Create (see page 7121)

Owner (see page 2807)

DataType (see page 7159)

Destroy (see page 7122)

39.1.42.3 TLargeintField Methods

The methods of the TLargeintField class are listed here.

Protected Methods

	Name	Description
	GetAsFloat (see page 7331)	This is GetAsFloat, a member of class TLargeintField.
	GetAsInteger (see page 7331)	This is GetAsInteger, a member of class TLargeintField.
	GetAsLargeint (see page 7331)	This is GetAsLargeint, a member of class TLargeintField.
	GetAsString (see page 7331)	This is GetAsString, a member of class TLargeintField.
	GetAsVariant (see page 7331)	This is GetAsVariant, a member of class TLargeintField.
	GetDataSize (see page 7332)	This is GetDataSize, a member of class TLargeintField.
	GetDefaultWidth (see page 7332)	This is GetDefaultWidth, a member of class TLargeintField.
	GetText (see page 7332)	This is GetText, a member of class TLargeintField.
	GetValue (see page 7332)	This is GetValue, a member of class TLargeintField.
	SetAsFloat (see page 7333)	This is SetAsFloat, a member of class TLargeintField.
	SetAsInteger (see page 7333)	This is SetAsInteger, a member of class TLargeintField.
	SetAsLargeint (see page 7333)	This is SetAsLargeint, a member of class TLargeintField.

	SetAsString (see page 7333)	This is SetAsString, a member of class TLargeintField.
	SetVarValue (see page 7333)	This is SetVarValue, a member of class TLargeintField.

39.1.42.3.1 TLargeintField.GetAsFloat Method

Pascal

```
function GetAsFloat: Double; override;
```

C++

```
virtual __fastcall Double GetAsFloat();
```

Description

This is GetAsFloat, a member of class TLargeintField.

39.1.42.3.2 TLargeintField.GetAsInteger Method

Pascal

```
function GetAsInteger: Longint; override;
```

C++

```
virtual __fastcall Longint GetAsInteger();
```

Description

This is GetAsInteger, a member of class TLargeintField.

39.1.42.3.3 TLargeintField.GetAsLargeint Method

Pascal

```
function GetAsLargeint: Largeint;
```

C++

```
__fastcall Largeint GetAsLargeint();
```

Description

This is GetAsLargeint, a member of class TLargeintField.

39.1.42.3.4 TLargeintField.GetAsString Method

Pascal

```
function GetAsString: string; override;
```

C++

```
virtual __fastcall AnsiString GetAsString();
```

Description

This is GetAsString, a member of class TLargeintField.

39.1.42.3.5 TLargeintField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TLargeintField.

39.1.42.3.6 TLargeintField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; override;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TLargeintField.

39.1.42.3.7 TLargeintField.GetDefaultWidth Method

Pascal

```
function GetDefaultWidth: Integer; override;
```

C++

```
virtual __fastcall int GetDefaultWidth();
```

Description

This is GetDefaultWidth, a member of class TLargeintField.

39.1.42.3.8 TLargeintField.GetText Method

Pascal

```
procedure GetText(var Text: string; DisplayText: Boolean); override;
```

C++

```
virtual __fastcall GetText(AnsiString Text, Boolean DisplayText);
```

Description

This is GetText, a member of class TLargeintField.

39.1.42.3.9 TLargeintField.GetValue Method

Pascal

```
function GetValue(var Value: Largeint): Boolean;
```

C++

```
__fastcall Boolean GetValue(Largeint Value);
```

Description

This is GetValue, a member of class TLargeintField.

39.1.42.3.10 TLargeintField.SetAsFloat Method

Pascal

```
procedure SetAsFloat(Value: Double); override;
```

C++

```
virtual __fastcall SetAsFloat(Double Value);
```

Description

This is SetAsFloat, a member of class TLargeintField.

39.1.42.3.11 TLargeintField.SetAsInteger Method

Pascal

```
procedure SetAsInteger(Value: Longint); override;
```

C++

```
virtual __fastcall SetAsInteger(Longint Value);
```

Description

This is SetAsInteger, a member of class TLargeintField.

39.1.42.3.12 TLargeintField.SetAsLargeint Method

Pascal

```
procedure SetAsLargeint(Value: Largeint);
```

C++

```
__fastcall SetAsLargeint(Largeint Value);
```

Description

This is SetAsLargeint, a member of class TLargeintField.

39.1.42.3.13 TLargeintField.SetAsString Method

Pascal

```
procedure SetAsString(const Value: string); override;
```

C++

```
virtual __fastcall SetAsString(const AnsiString Value);
```

Description

This is SetAsString, a member of class TLargeintField.

39.1.42.3.14 TLargeintField.SetVarValue Method

Pascal

```
procedure SetVarValue(const Value: Variant); override;
```

C++

```
virtual __fastcall SetVarValue(const Variant Value);
```

Description

This is SetVarValue, a member of class TLargeintField.

39.1.42.4 TLargeintField Properties

The properties of the TLargeintField class are listed here.

Public Properties

	Name	Description
	AsLargeInt (see page 7334)	Represents the field value as a large integer.
	Value (see page 7335)	Represents the value of the large integer field.

Published Properties

	Name	Description
	MaxValue (see page 7335)	Limits the maximum value in the large integer field.
	MinValue (see page 7335)	Limits the minimum value in the large integer field.

39.1.42.4.1 TLargeintField.AsLargeInt Property

Represents the field value as a large integer.

Pascal

```
property AsLargeInt: LargeInt;
```

C++

```
__property LargeInt AsLargeInt;
```

Description

Use AsLargeInt to:

Assign (see page 7124) the large integer field value to a 64-bit integer variable.

Assign (see page 7124) a 64-bit integer value to the large integer field.

For example, this code assigns a 64-bit integer value to a large integer field:

```
MyLargeIntegerField.AsLargeInt := 2147483648;
MyLargeIntegerField->AsLargeInt = 2147483648;
```

To assign a large integer field value to a 64-bit integer, use code similar to this:

```
LargeIntegerVariable := MyLargeIntegerField.AsLargeInt;
LargeIntVariable = MyLargeIntegerField->AsLargeInt;
```

If a value assigned to the field is not within the range established by the MinValue (see page 7335) and MaxValue (see page 7335) properties, an EDatabaseError (see page 6731) exception is raised.

See Also

Value (see page 7335)

MaxValue (see page 7335)

MinValue (see page 7335)

39.1.42.4.2 TLargeintField.MaxValue Property

Limits the maximum value in the large integer field.

Pascal

```
property MaxValue: Largeint;
```

C++

```
__property Largeint MaxValue;
```

Description

Use MaxValue to get or set the maximum value that can be entered into the field. If a value greater than MaxValue is entered, an EDatabaseError (see page 6731) exception is raised.

If the value of MaxValue is 0, the large integer field can have a value up to the maximum value that can be represented by the native database type. For 64-bit integer fields, this value is 2 ^63 - 1.

See Also

MinValue (see page 7335)

39.1.42.4.3 TLargeintField.MinValue Property

Limits the minimum value in the large integer field.

Pascal

```
property MinValue: Largeint;
```

C++

```
__property Largeint MinValue;
```

Description

Use MinValue to get or set the minimum value that can be entered into the field. If a value smaller than MinValue is entered, an EDatabaseError (see page 6731) exception is raised.

If the value of MinValue is 0, the large integer field can have a value as small as the native database type can represent. For 64-bit integer fields, this minimum value is -2 ^63

See Also

MaxValue (see page 7335)

39.1.42.4.4 TLargeintField.Value Property

Represents the value of the large integer field.

Pascal

```
property Value: Largeint;
```

C++

```
__property Largeint Value;
```

Description

Use Value to read data directly from and write data directly to a large integer field. Using the Value property is the same as using the AsLargeInt (see page 7334) property.

If a value assigned to the field is not within the range established by the MinValue (see page 7335) and the MaxValue (see page 7335) properties, an EDatabaseError (see page 6731) exception is raised.

See Also

AsLargeInt (see page 7334)

AsVariant

MaxValue (see page 7335)

MinValue (see page 7335)

39.1.43 TLookupList Class

TLookupList maintains a list of lookup values associated with a set of key field values.

Class Hierarchy

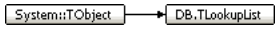

Pascal

```
TLookupList = class(TObject);
```

C++

```
class TLookupList : public TObject;
```

File

DB

Description

TField (see page 7115) uses TLookupList to implement a cache of lookup values. Each value in the lookup list is associated with a set of values on the key fields in the lookup data set. Lookup fields match the values of the fields identified by their KeyFields property with the values of the key fields from the lookup data set. When these values match, the value of the lookup field is set to the associated value from TLookupList.

Use TLookupList to generate a lookup list programmatically when there is no actual lookup dataset. Use the methods of the field's LookupList property to fill the lookup list with values and set the LookupCache property of the field to true. If no other lookup properties of the field are set, the field will use the supplied lookup list without overwriting it with values from a lookup dataset.

See Also

TField (see page 7115)

TField (see page 7115)

TField (see page 7115)

TField (see page 7115)

39.1.43.1 TLookupList Members

The following tables list the members exposed by TLookupList.

Public Methods

	Name	Description
	Add (see page 7338)	Adds a lookup value to the list for a given key value.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7339)	Empties the lookup list and frees the associated memory.
	Create (see page 7337)	Creates an instance of TLookupList (see page 7336).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 7338)	Destroys an instance of TLookupList (see page 7336).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ValueOfKey (see page 7339)	Returns the lookup value associated with a set of key field values.

39.1.43.2 TLookupList.Create Constructor

Creates an instance of TLookupList (see page 7336).

Pascal

```
constructor Create;
```

C++

```
__fastcall TLookupList();
```

Description

Applications should not need to call Create. Instances of TLookupList (see page 7336) are created by the field component that uses the list for its lookup cache. To obtain one of these instances, read the LookupList property of the field.

See Also

Create (see page 22287)

Destroy (see page 7338)

39.1.43.3 TLookupList.Destroy Destructor

Destroys an instance of TLookupList (see page 7336).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TLookupList();
```

Description

Do not call Destroy directly in an application. The lookup field that uses TLookupList (see page 7336) destroys the lookup list when it is done.

Destroy frees all memory used to store the list of lookup values.

See Also

Free (see page 22300)

Create (see page 7337)

39.1.43.4 TLookupList Methods

The methods of the TLookupList class are listed here.

Public Methods

	Name	Description
	Add (see page 7338)	Adds a lookup value to the list for a given key value.
	Clear (see page 7339)	Empties the lookup list and frees the associated memory.
	ValueOfKey (see page 7339)	Returns the lookup value associated with a set of key field values.

39.1.43.4.1 TLookupList.Add Method

Adds a lookup value to the list for a given key value.

Pascal

```
procedure Add(const AKey: Variant; const AValue: Variant);
```

C++

```
__fastcall Add(const Variant AKey, const Variant AValue);
```

Description

Lookup fields call Add when building the lookup cache to associate lookup values with the values of the key fields in the lookup dataset. The AKey parameter is a variant or variant array that represents the value or values of the key fields. AValue is a variant that represents the value of the lookup field associated with the key field values.

Use Add to build a lookup list from scratch when there is no actual lookup dataset.

See Also

LookupCache (see page 7170)

LookupList (see page 7172)

RefreshLookupList (see page 7138)

ValueOfKey (see page 7339)

39.1.43.4.2 TLookupList.Clear Method

Empties the lookup list and frees the associated memory.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the current values in the lookup list when building a lookup list programmatically. Lookup fields call Clear automatically before regenerating the lookup cache if the list represents the values in a lookup dataset.

See Also

RefreshLookupList (see page 7138)

39.1.43.4.3 TLookupList.ValueOfKey Method

Returns the lookup value associated with a set of key field values.

Pascal

```
function ValueOfKey(const AKey: Variant): Variant;
```

C++

```
__fastcall Variant ValueOfKey(const Variant AKey);
```

Description

Call ValueOfKey to look up the value for a lookup field that has the set of values specified by the AKey parameter on the fields identified by its KeyFields property. AKey is a Variant or Variant array that contains the values on the key fields in the lookup data set that uniquely specify a lookup field value.

See Also

KeyFields (see page 7169)

LookupList (see page 7172)

39.1.44 TMasterDataLink Class

TMasterDataLink allows a dataset to establish a master/detail relationship.

Class Hierarchy

Pascal

```
TMasterDataLink = class(TDetailDataLink);
```

C++

```
class TMasterDataLink : public TDetailDataLink;
```

File

DB

Description

Use TMasterDataLink as a private member of a custom dataset class that can represent the detail records for another dataset. TMasterDataLink maintains a list of field components from the master dataset, and passes on data events to the dataset that owns it. In the constructor of the dataset that uses TMasterDataLink to link to another data source, create the TMasterDataLink object and set any event handlers to respond to data events.

Do not confuse the dataset that is the value of the TMasterDataLink object's DataSet (see page 6890) property with the dataset that uses TMasterDataLink to link to a master dataset. The DataSet (see page 6890) property is the master dataset, while the dataset that owns the TMasterDataLink object is the detail set.

TClientDataSet, ADO-based datasets, and TTable objects already include a TMasterDataLink object.

Note: Unidirectional datasets and TQuery objects

use their own dedicated datalink classes to perform the same types of tasks.

See Also

TClientDataSet (see page 8289)

TTable (see page 10168)

TSQLDataLink (see page 20260)

39.1.44.1 TMasterDataLink Members

The following tables list the members exposed by TMasterDataLink.

Protected Methods

	Name	Description
	ActiveChanged (see page 7343)	Regenerates the Fields (see page 7346) property and triggers an OnMasterChange (see page 7347) or OnMasterDisable (see page 7347) event.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CheckBrowseMode (see page 7344)	Informs the dataset that owns the TMasterDataLink (see page 7339) object when the current record in the source dataset is about to change.
	DataEvent (see page 6881)	Responds to various events that occur while working with data.
	DataSetChanged (see page 6881)	Responds to changes in the dataset.
	DataSetScrolled (see page 6882)	Allows a response to scrolling the representation of the dataset within the data-aware object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EditingChanged (see page 6883)	Responds to changes in the editing state of the data source.
	FocusControl (see page 6884)	Provides the interface for a method to force focus to the data-aware component associated with the TDataLink (see page 6876).
	GetActiveRecord (see page 6884)	This is GetActiveRecord, a member of class TDataLink.
	GetBOF (see page 6884)	This is GetBOF, a member of class TDataLink.
	GetBufferCount (see page 6884)	This is GetBufferCount, a member of class TDataLink.
	GetDetailDataSet (see page 7344)	Returns the value of the DetailDataSet (see page 7114) property.
	GetEOF (see page 6885)	This is GetEOF, a member of class TDataLink.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRecordCount (see page 6885)	This is GetRecordCount, a member of class TDataLink.
	LayoutChanged (see page 7344)	Informs the dataset that owns the TMasterDataLink (see page 7339) object after the data source has changed the number of records buffered.

	MoveBy (see page 6885)	Repositions the dataset to another record relative to its active record.
	RecordChanged (see page 7345)	Generates an OnMasterChange (see page 7347) event when the contents of one of the fields in the current record changes.
	SetActiveRecord (see page 6886)	This is SetActiveRecord, a member of class TDataLink.
	SetBufferCount (see page 6887)	This is SetBufferCount, a member of class TDataLink.
	UpdateData (see page 6887)	Provides an interface for writing edits to a record in the dataset.

Protected Properties

	Name	Description
	VisualControl (see page 6894)	Indicates whether a visual control uses the datalink.

Public Events

	Name	Description
	OnMasterChange (see page 7347)	Occurs after the Fields (see page 7346) property changes.
	OnMasterDisable (see page 7347)	Occurs when the source dataset becomes inactive.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 7342)	Creates an instance of TMasterDataLink (see page 7339).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 7342)	Destroys an instance of TMasterDataLink (see page 7339).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6882)	Attempts to drive the data source into an editing state if it is not already in one.
	ExecuteAction (see page 6883)	Invokes an action with the datasource as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	UpdateAction (see page 6887)	Updates an action to reflect the current state of the TDataLink (see page 6876).
	UpdateRecord (see page 6888)	Responds to notifications that edits should be posted to the dataset.

Public Properties

	Name	Description
R	Active (see page 6888)	Specifies whether the dataset that TDataLink (see page 6876) manages is active.
	ActiveRecord (see page 6889)	Specifies the index of the current record within the internal record buffer maintained by the dataset for the Owner of the TDataLink (see page 6876) object.
R	BOF (see page 6889)	Indicates whether the first record in a dataset is active.
	BufferCount (see page 6890)	Specifies the number of records cached by the dataset for the Owner of the TDataLink (see page 6876) object.
R	DataSet (see page 6890)	Specifies the TDataSet (see page 6894) object that represents the data that this TDataLink (see page 6876) is tracking.
	DataSource (see page 6891)	Specifies the TDataSource (see page 7070) object that the Owner of the TDataLink (see page 6876) uses to connect to the dataset.
	DataSourceFixed (see page 6891)	Indicates whether the DataSource (see page 6891) property can be set.
R	DetailDataSet (see page 7114)	Indicates the detail dataset in the master/detail relationship.
R	Editing (see page 6892)	Indicates whether the DataSource (see page 6891) is in edit mode.
R	Eof (see page 6892)	Indicates whether the last record in a dataset is active.
	FieldNames (see page 7345)	Names the fields from the master dataset that are used to define the link to the detail dataset.
R	Fields (see page 7346)	Lists all the field components from the master dataset that are used to establish a link to the detail dataset.
	ReadOnly (see page 6893)	Indicates whether the dataset for this TDataLink (see page 6876) can be modified.
R	RecordCount (see page 6893)	Indicates the number of records in the internal record buffer maintained by the dataset.

39.1.44.2 TMasterDataLink.Create Constructor

Creates an instance of TMasterDataLink (see page 7339).

Pascal

```
constructor Create(DataSet: TDataSet);
```

C++

```
__fastcall TMasterDataLink(TDataSet DataSet);
```

Description

Call Create from the constructor of the dataset that uses TMasterDataLink (see page 7339) to link to a master dataset. The dataset that uses the TMasterDataLink (see page 7339) passes itself as the DataSet (see page 6890) parameter.

Create associates the instance of TMasterDataLink (see page 7339) with the dataset that owns it, and creates the helper objects it uses to manage the list of field components.

See Also

Fields (see page 7346)

Create (see page 6878)

Destroy (see page 6879)

39.1.44.3 TMasterDataLink.Destroy Destructor

Destroys an instance of TMasterDataLink (see page 7339).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMasterDataLink();
```

Description

Applications do not free instances of TMasterDataLink (see page 7339). The dataset that owns the TMasterDataLink (see page 7339) instance frees the datalink from its own destructor.

Component writers call the Free method from the destructor of the dataset that owns the TMasterDataLink (see page 7339) object. Free checks that the TMasterDataLink (see page 7339) reference is not already freed, and only then calls Destroy.

Destroy frees up the memory used to manage the list of field components.

See Also

Create (see page 7342)

Free (see page 22300)

39.1.44.4 TMasterDataLink Methods

The methods of the TMasterDataLink class are listed here.

Protected Methods

	Name	Description
	ActiveChanged (see page 7343)	Regenerates the Fields (see page 7346) property and triggers an OnMasterChange (see page 7347) or OnMasterDisable (see page 7347) event.
	CheckBrowseMode (see page 7344)	Informs the dataset that owns the TMasterDataLink (see page 7339) object when the current record in the source dataset is about to change.
	GetDetailDataSet (see page 7344)	Returns the value of the DetailDataSet (see page 7114) property.
	LayoutChanged (see page 7344)	Informs the dataset that owns the TMasterDataLink (see page 7339) object after the data source has changed the number of records buffered.
	RecordChanged (see page 7345)	Generates an OnMasterChange (see page 7347) event when the contents of one of the fields in the current record changes.

39.1.44.4.1 TMasterDataLink.ActiveChanged Method

Regenerates the Fields (see page 7346) property and triggers an OnMasterChange (see page 7347) or OnMasterDisable (see page 7347) event.

Pascal

```
procedure ActiveChanged; override;
```

C++

```
virtual __fastcall ActiveChanged();
```

Description

Applications can not call the protected ActiveChanged procedure. Changes in the Active (see page 6888) property or the FieldNames (see page 7345) property trigger this procedure. When the source dataset becomes inactive, ActiveChanged empties the Fields (see page 7346) property and generates an OnMasterDisable (see page 7347) event. When the dataset becomes active, ActiveChanged regenerates Fields (see page 7346) from the source dataset and generates an OnMasterChange (see page 7347) event.

See Also

Active (see page 6888)

Fields (see page 7346)

OnMasterChange (see page 7347)

OnMasterDisable (see page 7347)

FieldNames (see page 7345)

39.1.44.4.2 TMasterDataLink.CheckBrowseMode Method

Informs the dataset that owns the TMasterDataLink (see page 7339) object when the current record in the source dataset is about to change.

Pascal

```
procedure CheckBrowseMode; override;
```

C++

```
virtual __fastcall CheckBrowseMode();
```

Description

Applications can not call the protected CheckBrowseMode procedure. CheckBrowseMode is called immediately before the current record in the source dataset changes. It calls the CheckBrowseMode method of the dataset that owns the TMasterDataLink (see page 7339) object, so that it can post any pending edits before the record changes.

See Also

OnMasterChange (see page 7347)

CheckBrowseMode (see page 6918)

39.1.44.4.3 TMasterDataLink.GetDetailDataSet Method

Returns the value of the DetailDataSet (see page 7114) property.

Pascal

```
function GetDetailDataSet: TDataSet; override;
```

C++

```
virtual __fastcall TDataSet GetDetailDataSet();
```

Description

GetDetailDataSet is the protected read implementation of the DetailDataSet (see page 7114) property.

See Also

DetailDataSet

39.1.44.4.4 TMasterDataLink.LayoutChanged Method

Informs the dataset that owns the TMasterDataLink (see page 7339) object after the data source has changed the number of records buffered.

Pascal

```
procedure LayoutChanged; override;
```

C++

```
virtual __fastcall LayoutChanged();
```

Description

Applications can not call the protected LayoutChanged method. LayoutChanged is called automatically when the source dataset changes to reflect a change in the layout of a control linked to that source dataset. LayoutChanged regenerates the Fields (see page 7346) property and triggers an OnMasterChange (see page 7347) event.

See Also

Fields (see page 7346)

OnMasterChange (see page 7347)

39.1.44.4.5 TMasterDataLink.RecordChanged Method

Generates an OnMasterChange (see page 7347) event when the contents of one of the fields in the current record changes.

Pascal

```
procedure RecordChanged(Field: TField); override;
```

C++

```
virtual __fastcall RecordChanged(TField Field);
```

Description

Applications can not call the protected RecordChanged method. It is called automatically when the contents of the current record change. The Field parameter indicates which field in the current record has changed. If Field is nil (Delphi) or NULL (C++), the entire record has changed.

If the Field parameter is nil (Delphi) or NULL (C++), or if the Field parameter is one of the fields listed in the Fields (see page 7346) property, RecordChanged triggers an OnMasterChange (see page 7347) event.

See Also

Fields (see page 7346)

OnMasterChange (see page 7347)

39.1.44.5 TMasterDataLink Properties

The properties of the TMasterDataLink class are listed here.

Public Properties

	Name	Description
	FieldNames (see page 7345)	Names the fields from the master dataset that are used to define the link to the detail dataset.
R	Fields (see page 7346)	Lists all the field components from the master dataset that are used to establish a link to the detail dataset.

39.1.44.5.1 TMasterDataLink.FieldNames Property

Names the fields from the master dataset that are used to define the link to the detail dataset.

Pascal

```
property FieldNames: string;
```

C++

```
__property AnsiString FieldNames;
```

Description

Set FieldNames to the fields in the master dataset that supply values that must be matched by fields in the detail dataset. Each field name in the string must be separated from other field names with a semicolon.

Setting FieldNames causes TMasterDataLink (see page 7339) to regenerate the Fields (see page 7346) property from the field components of the master dataset. Once the Fields (see page 7346) property has been regenerated, TMasterDataLink (see page 7339) triggers an OnMasterChange (see page 7347) event.

See Also

DataSet (see page 6890)

Fields (see page 7346)

OnMasterChange (see page 7347)

39.1.44.5.2 TMasterDataLink.Fields Property

Lists all the field components from the master dataset that are used to establish a link to the detail dataset.

Pascal

```
property Fields: TList;
```

C++

```
__property TList Fields;
```

Description

Use Fields to access individual link field components from the master data set. Fields lists the field components in the master dataset named by the FieldNames (see page 7345) property. Datasets use this property to ascertain the current values of the link fields.

Fields is a read-only property. Fields is generated by locating the field components referenced by the FieldNames (see page 7345) property in the master dataset.

See Also

DataSet (see page 6890)

FieldNames (see page 7345)

TField (see page 7115)

39.1.44.6 TMasterDataLink Events

The events of the TMasterDataLink class are listed here.

Public Events

	Name	Description
	OnMasterChange (see page 7347)	Occurs after the Fields (see page 7346) property changes.
	OnMasterDisable (see page 7347)	Occurs when the source dataset becomes inactive.

39.1.44.6.1 TMasterDataLink.OnMasterChange Event

Occurs after the Fields (see page 7346) property changes.

Pascal

```
property OnMasterChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMasterChange;
```

Description

Write an OnMasterChange event handler to update the dataset that owns the TMasterSourceLink to reflect any changes that occurred in the source (master) dataset or the FieldNames (see page 7345) property. The Fields (see page 7346) property reflects the new state of the source (master) dataset and the FieldNames (see page 7345) property. If FieldNames (see page 7345) refers to any fields that disappeared from the source dataset, the Fields (see page 7346) property contains undefined field components for those fields.

Typically, the OnMasterChange event handler updates the detail dataset to reflect those details that correspond to the master field values obtained from the Fields (see page 7346) property.

See Also

FieldNames (see page 7345)

Fields (see page 7346)

OnMasterDisable (see page 7347)

39.1.44.6.2 TMasterDataLink.OnMasterDisable Event

Occurs when the source dataset becomes inactive.

Pascal

```
property OnMasterDisable: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMasterDisable;
```

Description

Write an OnMasterDisable event handler to respond when the source (master) dataset becomes inactive. When the source dataset is inactive, the field components in the Fields (see page 7346) property can not be used to obtain the data from records in the source dataset. Thus, only information cached by the detail dataset that owns the TMasterDataLink (see page 7339) can be used after OnMasterDisable until the next OnMasterChange (see page 7347) event occurs.

See Also

Active (see page 6888)

Fields (see page 7346)

OnMasterChange (see page 7347)

39.1.45 TMemoField Class

TMemoField represents a memo field in a dataset.

Class Hierarchy

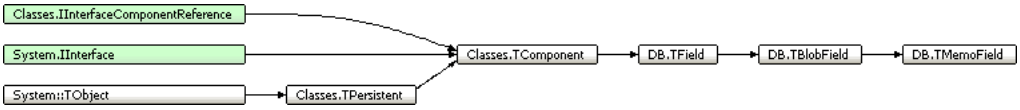

Pascal

```
TMemoField = class(TBlobField);
```

C++

```
class TMemoField : public TBlobField;
```

File

DB

Description

TMemoField encapsulates the fundamental behavior common to fields that contain text data or arbitrary length. Memo fields are a form of binary large object (BLOB) field where the data consists of simple text.

If you use the Fields editor at design time to create a persistent field component for the memo field, you can access it by name at runtime. When using dynamic field components, you can access the TMemoField instance using the dataset's Fields property or FieldByName method.

See Also

TDataSet (see page 6894)

TGraphicField (see page 7257)

TDBMemo (see page 9149)

TBlobField (see page 6801)

TDataSet (see page 6894)

TField (see page 7115)

39.1.45.1 TMemoField Members

The following tables list the members exposed by TMemoField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.

	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 6810)	Copies the properties of an object to a destination object.
	Bind (see page 7126)	This is Bind, a member of class TField.
	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	FreeBuffers (see page 6811)	This is FreeBuffers, a member of class TBlobField.
	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
	GetAsString (see page 6811)	This is GetAsString, a member of class TBlobField.
	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
	GetAsVariant (see page 6811)	This is GetAsVariant, a member of class TBlobField.
	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
	GetAsWideString (see page 7355)	This is GetAsWideString, a member of class TMemoField.
	GetBlobSize (see page 6812)	This is GetBlobSize, a member of class TBlobField.
	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClassDesc (see page 6812)	This is GetClassDesc, a member of class TBlobField.
	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsNull (see page 6812)	This is GetIsNull, a member of class TBlobField.
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 6812)	This is GetText, a member of class TBlobField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 6817)	This is SetAsString, a member of class TBlobField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7356)	This is SetAsWideString, a member of class TMemoField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 6818)	This is SetText, a member of class TBlobField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 6818)	This is SetVarValue, a member of class TBlobField.
	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.

	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	Assign (see page 6809)	Copies a value to the BLOB field.
V	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 6810)	Clears the current value in the BLOB field.
V	Clear (see page 7126)	Sets the value of the field to NULL.
V	Create (see page 7354)	Creates an instance of TMemoField (see page 7348).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 6813)	Indicates that this is a BLOB field.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	LoadFromFile (see page 6814)	Loads BLOB data from a file into the field.
	LoadFromStream (see page 6815)	Loads BLOB data from a stream into the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 6816)	Saves the contents of the BLOB field to a file.
	SaveToStream (see page 6816)	Saves the contents of the BLOB field to a stream.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 6817)	Specifies the type of the BLOB field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
R	BlobSize (see page 6819)	Indicates the number of bytes of data contained in the blob field on the current record.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.

R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	Modified (see page 6820)	Indicates whether a the value for the BLOB field was changed since it was last read from the underlying database.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 6822)	Represents the data in the BLOB field.
	Value (see page 7180)	Represents the data in a field component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.

	BlobType (see page 6819)	Identifies the type of the BLOB field associated with a BLOB field object.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
	GraphicHeader (see page 6820)	Indicates whether the field's value includes a Paradox graphic BLOB header.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Size (see page 6820)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Transliterate (see page 7356)	Determines whether any text in the BLOB field should be converted to and from the character set identified by the dataset.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.45.2 TMemoField.Create Constructor

Creates an instance of TMemoField (see page 7348).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TMemoField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TMemoField (see page 7348). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftMemo, and initializes the Transliterate (see page 7356) property to true. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of

TMemoField (see page 7348). After instantiating a TMemoField (see page 7348), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TMemoField (see page 7348) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TMemoField (see page 7348) with a dataset component by setting its DataSet (see page 7157) property.

The example below creates a TMemoField (see page 7348) object for a field named Notes accessed through a TQuery named Query1.

```
var
  T: TMemoField;
begin
  Query1.Close;
  T := TMemoField.Create(Query1);
  T.FieldName := 'Notes';
  T.Name := Query1.Name + T.FieldName;
  T.Index := Query1.FieldCount;
  T.DataSet := Query1;
  Query1.FieldDefs.UpDate;
  Query1.Open;
end;
TMemoField *T;
Query1->Close();
T = new TMemoField(Query1);
T->FieldName = "Notes";
T->Name = Query1->Name + T->FieldName;
T->Index = Query1->FieldCount;
T->DataSet = Query1;
Query1->FieldDefs->UpDate();
Query1->Open();
```

See Also

Destroy (see page 7122)

DataType (see page 7159)

Create (see page 7121)

Transliterate (see page 6821)

Owner (see page 2807)

39.1.45.3 TMemoField Methods

The methods of the TMemoField class are listed here.

Protected Methods

	Name	Description
	GetAsWideString (see page 7355)	This is GetAsWideString, a member of class TMemoField.
	SetAsWideString (see page 7356)	This is SetAsWideString, a member of class TMemoField.

39.1.45.3.1 TMemoField.GetAsWideString Method

Pascal

```
function GetAsWideString: WideString; override;
```

C++

```
virtual __fastcall BSTR GetAsWideString();
```

Description

This is GetAsWideString, a member of class TMemoField.

39.1.45.3.2 TMemoField.SetAsWideString Method

Pascal

```
procedure SetAsWideString(const Value: WideString); override;
```

C++

```
virtual __fastcall SetAsWideString(const BSTR Value);
```

Description

This is SetAsWideString, a member of class TMemoField.

39.1.45.4 TMemoField Properties

The properties of the TMemoField class are listed here.

Published Properties

	Name	Description
	Transliterate (see page 7356)	Determines whether any text in the BLOB field should be converted to and from the character set identified by the dataset.

39.1.45.4.1 TMemoField.Transliterate Property

Determines whether any text in the BLOB field should be converted to and from the character set identified by the dataset.

Pascal

```
property Transliterate: Boolean;
```

C++

```
__property Boolean Transliterate;
```

Description

Use Transliterate when the physical database table identified by the dataset does not use an ANSI language driver and the data may contain extended ASCII characters uses a different character set than UTF-8 (on Linux) or ANSI (on Windows) and the BLOB field contains text.

When Transliterate is true, BLOB fields call the dataset's Translate method to translate ANSI UTF-8 characters to the character set used by the dataset when the value is changed using the Value property, the AsString property, or the AsVariant property. When these properties are read to obtain the value of the BLOB field, the Translate method translates the characters in the database table to the UTF-8 (Linux) or ANSI (Windows) character set.

Note: BLOB streams (returned by the dataset's CreateBlobStream method) also check the Transliterate property when reading from or writing to a BLOB field.

See Also

AnsiToNative (see page 10244)

AsString

AsVariant

DataSet (see page 7157)

NativeToAnsi (see page 10249)

Locale (see page 9934)

Value (see page 6822)

Translate (see page 7001)

CreateBlobStream (see page 6922)

39.1.46 TNamedItem Class

TNamedItem is the base class for definitions used by datasets.

Class Hierarchy

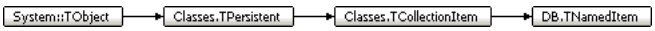

Pascal

```
TNamedItem = class(TCollectionItem);
```

C++

```
class TNamedItem : public TCollectionItem;
```

File

DB

Description

Use TNamedItem as a base class when defining a class for definitions that are maintained in a collection by a dataset. Sets of TNamedItem objects are maintained by a descendant of TDefCollection (see page 7104) that is specific to the type of object defined.

Definition objects are used by datasets to specify the properties of database entities such as fields or indexes.

See Also

TFieldDef (see page 7184)

TIndexDef (see page 7283)

TDefCollection (see page 7104)

39.1.46.1 TNamedItem Members

The following tables list the members exposed by TNamedItem.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 7359)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.

	SetDisplayName (see page 7359)	This is SetDisplayName, a member of class TNamedItem.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2757)	Creates and initializes a TCollectionItem (see page 2755) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	Name (see page 7359)	Specifies the name of the database entity defined by the TNamedItem (see page 7357) object.

39.1.46.2 TNamedItem Methods

The methods of the TNamedItem class are listed here.

Protected Methods

	Name	Description
	GetDisplayName (see page 7359)	Returns the name of the collection item as it appears in the collection editor.
	SetDisplayName (see page 7359)	This is SetDisplayName, a member of class TNamedItem.

39.1.46.2.1 TNamedItem.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName (see page 2761)

GetNamePath (see page 2759)

39.1.46.2.2 TNamedItem.SetDisplayName Method

Pascal

```
procedure SetDisplayName(const Value: WideString); virtual; overload;
```

C++

```
virtual __fastcall SetDisplayName(const BSTR Value);
```

Description

This is SetDisplayName, a member of class TNamedItem.

39.1.46.3 TNamedItem Properties

The properties of the TNamedItem class are listed here.

Published Properties

	Name	Description
	Name (see page 7359)	Specifies the name of the database entity defined by the TNamedItem (see page 7357) object.

39.1.46.3.1 TNamedItem.Name Property

Specifies the name of the database entity defined by the TNamedItem (see page 7357) object.

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

Use Name to specify the name that the underlying database table uses to refer to the defined object. For example, if the TNamedItem (see page 7357) object is a field definition, the value of Name is the name of the corresponding field in the underlying database table.

Note: For dBASE maintained indexes, Name refers to the tag name of the index. For Paradox tables, the Name of the primary key is an empty string.

C++ Examples:

```
/*
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
*/
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2->Active = False;
  // Get the current available indices
  CDS2->IndexDefs->Update();
  // Find a field named "Field2"
  for (int I = 0; I < CDS2->IndexDefs->Count; I++)
    if (CDS2->IndexDefs->Items[I]->Fields == "Field2")
    {
      // set that index as the current index for the dataset}
      CDS2->IndexName = CDS2->IndexDefs->Items[I]->Name;
    }
  CDS2->Active = True;
}
```

Delphi Examples:

```
{
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
}
procedure TForm1.Button3Click(Sender: TObject);
var
  I : Integer;
begin
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2.Active := False;
  { Get the current available indices }
  CDS2.IndexDefs.Update;
  { Find a field named 'Field2' }
  for I := 0 to CDS2.IndexDefs.Count - 1 do
    if CDS2.IndexDefs.Items[I].Fields = 'Field2' then
    begin
      { set that index as the current index for the dataset}
      CDS2.IndexName := CDS2.IndexDefs.Items[I].Name;
    end;
  CDS2.Active := True;
end;
```

See Also

DisplayName

Source (see page 7292)

IndexName (see page 10222)

39.1.47 TNumericField Class

TNumericField is the common ancestor of all the numeric fields in a dataset.

Class Hierarchy

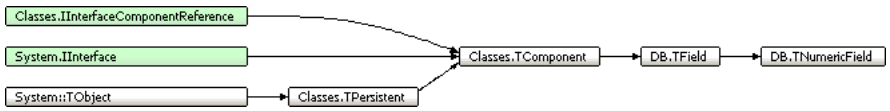

Pascal

```
TNumericField = class(TField);
```

C++

```
class TNumericField : public TField;
```

File

DB

Description

TNumericField encapsulates the fundamental behavior common to all numeric field components. In addition to the properties, events, and methods common to all field components introduced by TField (see page 7115), TNumericField introduces new properties for formatting numeric data for display or editing purposes.

Do not instantiate objects of type TNumericField. Instead, use one of its descendant classes such as TIntegerField (see page 7301) and TFloatField (see page 7229).

See Also

TBCDField (see page 6776)

TDataSet (see page 6894)

TDataSet (see page 6894)

TDataSet (see page 6894)

TField (see page 7115)

TAutoIncField (see page 6768)

TSmallintField (see page 7461)

TIntegerField (see page 7301)

TWordField (see page 7560)

39.1.47.1 TNumericField Members

The following tables list the members exposed by TNumericField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
V	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
V	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
V	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
V	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
V	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
V	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.

	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	RangeError (see page 7368)	This is RangeError, a member of class TNumericField.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayFormat (see page 7368)	This is SetDisplayFormat, a member of class TNumericField.
	SetEditFormat (see page 7368)	This is SetEditFormat, a member of class TNumericField.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.

	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7126)	Sets the value of the field to NULL.
	Create (see page 7367)	Creates an instance of a TNumericField (see page 7361) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.

	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7180)	Represents the data in a field component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7369)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayFormat (see page 7369)	Determines how a numeric field's value is formatted for display in a data-aware control.

	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	EditFormat (see page 7371)	Determines how a numeric field's value is formatted when it is being edited in a data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.47.2 TNumericField.Create Constructor

Creates an instance of a TNumericField (see page 7361) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TNumericField(TComponent * AOwner);
```

Description

Do not call the create method for TNumericField (see page 7361) directly. TNumericField (see page 7361) is an abstract class and its constructor should only be called as an inherited method from the constructor of a derived class.

Create changes the Alignment (see page 7369) property to taRightJustify.

See Also

Alignment (see page 7146)

Create (see page 7121)

39.1.47.3 TNumericField Methods

The methods of the TNumericField class are listed here.

Protected Methods

	Name	Description
	RangeError (see page 7368)	This is RangeError, a member of class TNumericField.
	SetDisplayFormat (see page 7368)	This is SetDisplayFormat, a member of class TNumericField.
	SetEditFormat (see page 7368)	This is SetEditFormat, a member of class TNumericField.

39.1.47.3.1 TNumericField.RangeError Method

Pascal

```
procedure RangeError(Value: Extended; Min: Extended; Max: Extended);
```

C++

```
__fastcall RangeError(Extended Value, Extended Min, Extended Max);
```

Description

This is RangeError, a member of class TNumericField.

39.1.47.3.2 TNumericField.SetDisplayFormat Method

Pascal

```
procedure SetDisplayFormat(const Value: string);
```

C++

```
__fastcall SetDisplayFormat(const AnsiString Value);
```

Description

This is SetDisplayFormat, a member of class TNumericField.

39.1.47.3.3 TNumericField.SetEditFormat Method

Pascal

```
procedure SetEditFormat(const Value: string);
```

C++

```
__fastcall SetEditFormat(const AnsiString Value);
```

Description

This is SetEditFormat, a member of class TNumericField.

39.1.47.4 TNumericField Properties

The properties of the TNumericField class are listed here.

Published Properties

	Name	Description
	Alignment (see page 7369)	Determines how the field's data is displayed within a data-aware control.
	DisplayFormat (see page 7369)	Determines how a numeric field's value is formatted for display in a data-aware control.
	EditFormat (see page 7371)	Determines how a numeric field's value is formatted when it is being edited in a data-aware control.

39.1.47.4.1 TNumericField.Alignment Property

Determines how the field's data is displayed within a data-aware control.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to specify the position of the field's data within data-aware controls. These are the possible values:

Value	Meaning
taLeftJustify	Align data to the left side of the control
taCenter	Center data horizontally in the control
taRightJustify	Align data to the right side of the control

See Also

DisplayLabel (see page 7160)

DisplayName (see page 7160)

DisplayText (see page 7162)

DisplayWidth (see page 7162)

TDBEdit (see page 8735)

39.1.47.4.2 TNumericField.DisplayFormat Property

Determines how a numeric field's value is formatted for display in a data-aware control.

Pascal

```
property DisplayFormat: string;
```

C++

```
__property AnsiString DisplayFormat;
```

Description

Use the DisplayFormat property to override automatic formatting of a field for display purposes.

Note: DisplayFormat affects only the display of the field's data. It does not affect the validity of strings assigned to a numeric field using the AsString (see page 7151) property.

The value of DisplayFormat is a string that encodes the formatting of numeric data using the specifiers in the following table:

Specifier	Represents
0	Digit placeholder. If the value being formatted has a digit in the position where the "0" appears in the format string, then that digit is copied to the output string. Otherwise, a "0" is stored in that position in the output string.
#	Digit placeholder. If the value being formatted has a digit in the position where the "#" appears in the format string, then that digit is copied to the output string. Otherwise, nothing is stored in that position in the output string.

.	Decimal point. The first "." character in the format string determines the location of the decimal separator in the formatted value; any additional "." characters are ignored. The actual character used as a the decimal separator in the output string is determined by the DecimalSeparator global variable. The default value of DecimalSeparator is specified in the Number Format of the Regional Settings section in the Windows Control Panel.
,	Thousand separator. If the format string contains one or more "," characters, the output will have thousand separators inserted between each group of three digits to the left of the decimal point. The placement and number of "," characters in the format string does not affect the output, except to indicate that thousand separators are wanted. The actual character used as a the thousand separator in the output is determined by the ThousandSeparator global variable. The default value of ThousandSeparator is specified in the Number Format of the Regional Settings section in the Windows Control Panel.
E+	Scientific notation. If any of the strings "E+", "E-", "e+", or "e-" are contained in the format string, the number is formatted using scientific notation. A group of up to four "0" characters can immediately follow the "E+", "E-", "e+", or "e-" to determine the minimum number of digits in the exponent. The "E+" and "e+" formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The "E-" and "e-" formats output a sign character only for negative exponents.
'xx'/"xx"	Characters enclosed in single or double quotes are output as-is, and do not affect formatting.
;	Separates sections for positive, negative, and zero numbers in the format string.

The locations of the leftmost "0" before the decimal point in the format string and the rightmost "0" after the decimal point in the format string determine the range of digits that are always present in the output string.

The number being formatted is always rounded to as many decimal places as there are digit placeholders ("0" or "#") to the right of the decimal point. If the format string contains no decimal point, the value being formatted is rounded to the nearest whole number.

If the number being formatted has more digits to the left of the decimal separator than there are digit placeholders to the left of the "." character in the format string, the extra digits are output before the first digit placeholder.

The following table shows the effect of various format strings:

DisplayFormat	Value (see page 7180)	Result	Comment

To allow different formats for positive, negative, and zero values, the format string can contain between one and three sections separated by semicolons.

One section: The format string applies to all values.

Two sections: The first section applies to positive values and zeros, and the second section applies to negative values.

Three sections: The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If the section for negative values or the section for zero values is empty, that is, if there is nothing between the semicolons that delimit the section, the section for positive values is used instead.

If the section for positive values is empty, or if the entire format string is empty, the value is formatted using general floating-point formatting with 15 significant digits. General floating-point formatting is also used if the value has more than 18 digits to the left of the decimal point and the format string does not specify scientific notation.

See Also

AsString (see page 7151)

DisplayText (see page 7162)

EditFormat (see page 7371)

OnGetText (see page 7182)

FormatFloat (see page 22904)

39.1.47.4.3 TNumericField.EditFormat Property

Determines how a numeric field's value is formatted when it is being edited in a data-aware control.

Pascal

```
property EditFormat: string;
```

C++

```
__property AnsiString EditFormat;
```

Description

Use the EditFormat property to override the formatting of a field when the field's value is being edited in a data-aware control.

When EditFormat is unassigned, but the DisplayFormat (see page 7369) property has a value, the DisplayFormat (see page 7369) string is used.

The value of EditFormat is a string that encodes the formatting of numeric data using the specifiers in the following table:

Specifier	Represents
0	Digit placeholder. If the value being formatted has a digit in the position where the "0" appears in the format string, then that digit is copied to the output string. Otherwise, a "0" is stored in that position in the output string.
#	Digit placeholder. If the value being formatted has a digit in the position where the "#" appears in the format string, then that digit is copied to the output string. Otherwise, nothing is stored in that position in the output string.
.	Decimal point. The first "." character in the format string determines the location of the decimal separator in the formatted value; any additional "." characters are ignored. The actual character used as a the decimal separator in the output string is determined by the DecimalSeparator global variable. The default value of DecimalSeparator is specified in the Number Format of the Regional Settings section in the Windows Control Panel.
,	Thousand separator. If the format string contains one or more "," characters, the output will have thousand separators inserted between each group of three digits to the left of the decimal point. The placement and number of "," characters in the format string does not affect the output, except to indicate that thousand separators are wanted. The actual character used as a the thousand separator in the output is determined by the ThousandSeparator global variable. The default value of ThousandSeparator is specified in the Number Format of the Regional Settings section in the Windows Control Panel.
E+	Scientific notation. If any of the strings "E+", "E-", "e+", or "e-" are contained in the format string, the number is formatted using scientific notation. A group of up to four "0" characters can immediately follow the "E+", "E-", "e+", or "e-" to determine the minimum number of digits in the exponent. The "E+" and "e+" formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The "E-" and "e-" formats output a sign character only for negative exponents.
'xx'/"xx"	Characters enclosed in single or double quotes are output as-is, and do not affect formatting.
;	Separates sections for positive, negative, and zero numbers in the format string.

The locations of the leftmost "0" before the decimal point in the format string and the rightmost "0" after the decimal point in the format string determine the range of digits that are always present in the output string.

The number being formatted is always rounded to as many decimal places as there are digit placeholders ("0" or "#") to the right of the decimal point. If the format string contains no decimal point, the value being formatted is rounded to the nearest whole number.

If the number being formatted has more digits to the left of the decimal separator than there are digit placeholders to the left of the "." character in the format string, the extra digits are output before the first digit placeholder.

To allow different formats for positive, negative, and zero values, the format string can contain between one and three sections separated by semicolons.

One section: The format string applies to all values.

Two sections: The first section applies to positive values and zeros, and the second section applies to negative values.

Three sections: The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If the section for negative values or the section for zero values is empty, that is if there is nothing between the semicolons that delimit the section, the section for positive values is used instead.

If the section for positive values is empty, or if the entire format string is empty, the value is formatted using general floating-point formatting with 15 significant digits. General floating-point formatting is also used if the value has more than 18 digits to the left of the decimal point and the format string does not specify scientific notation.

Note: EditFormat affects only the display of the field's data when it is being edited. It does not affect the validity of strings assigned to a numeric field using the AsString (see page 7151) property, nor does it affect the display of the field's data when it is not being edited.

See Also

DisplayFormat (see page 7369)

OnGetText (see page 7182)

Text (see page 7179)

Controlling Grid Drawing

Using Default Formatting for Numeric

39.1.48 TObjectField Class

TObjectField is the common ancestor of TADTField (see page 6736), TArrayField (see page 6760), TDataSetField (see page 7060), and TReferenceField (see page 7452) components in a dataset.

Class Hierarchy

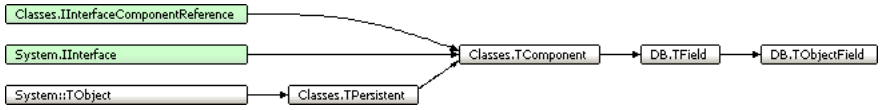

Pascal

```
TObjectField = class(TField);
```

C++

```
class TObjectField : public TField;
```

File

DB

Description

TObjectField encapsulates the fundamental behavior common to fields that are a composite of simpler types. These include ADT fields and array fields, which are built up of subfields, and dataset fields and reference fields, which refer to another dataset. TObjectField introduces the Fields (see page 7384) property and the FieldValues (see page 7385) property for accessing the subfields that make up the composite (or that reside in the dataset that is the field's value).

Do not create instances of TObjectField. It is intended only as a base class.

See Also

TADTField (see page 6736)

TArrayField (see page 6760)

TDataSetField (see page 7060)

TReferenceField (see page 7452)

39.1.48.1 TObjectField Members

The following tables list the members exposed by TObjectField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7380)	This is CheckTypeSize, a member of class TObjectField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 7380)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7380)	This is FreeBuffers, a member of class TObjectField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
V	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsString (see page 7381)	This is GetAsString, a member of class TObjectField.
V	GetAsVariant (see page 7381)	This is GetAsVariant, a member of class TObjectField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
V	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
V	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
V	GetDefaultWidth (see page 7381)	This is GetDefaultWidth, a member of class TObjectField.
	GetFieldCount (see page 7381)	This is GetFieldCount, a member of class TObjectField.
V	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.

	GetFields (see page 7382)	This is GetFields, a member of class TObjectField.
	GetFieldValue (see page 7382)	This is GetFieldValue, a member of class TObjectField.
	GetHasConstraints (see page 7382)	This is GetHasConstraints, a member of class TObjectField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 7382)	This is SetChildOrder, a member of class TObjectField.
	SetDataSet (see page 7383)	This is SetDataSet, a member of class TObjectField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7383)	This is SetFieldKind, a member of class TObjectField.
	SetFieldValue (see page 7383)	This is SetFieldValue, a member of class TObjectField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7383)	This is SetParentField, a member of class TObjectField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetUnNamed (see page 7383)	This is SetUnNamed, a member of class TObjectField.
	SetVarValue (see page 7384)	This is SetVarValue, a member of class TObjectField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7126)	Sets the value of the field to NULL.
	Create (see page 7378)	Creates and initializes a TObjectField (see page 7372) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 7379)	Destroys an instance of TObjectField (see page 7372).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetChildren (see page 7381)	This is GetChildren, a member of class TObjectField.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.

R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldCount (see page 7384)	Indicates the number of child fields that comprise the object field.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	Fields (see page 7384)	Specifies the TFields (see page 7211) object that contains an array of child fields.
	FieldValues (see page 7385)	Provides access to the values for all child fields in the object field.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
R	UnNamed (see page 7386)	Indicates whether an object field in a nested table has a name associated with it.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7180)	Represents the data in a field component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectType (see page 7385)	Specifies the classification of the field object in Oracle.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.48.2 TObjectField.Create Constructor

Creates and initializes a TObjectField (see page 7372) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TObjectField(TComponent * AOwner);
```

Description

Do not create TObjectField (see page 7372) instances. TObjectField (see page 7372) is an abstract base class that should not be directly instantiated. The constructor for descendant classes is usually called indirectly by using the Fields (see page 7384) editor at design time to create a persistent field component. Dataset objects also call field constructors automatically to create dynamic field components.

Create creates a TFields (see page 7211) object to manage the child fields of the object field.

See Also

Destroy (see page 7379)

39.1.48.3 TObjectField.Destroy Destructor

Destroys an instance of TObjectField (see page 7372).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TObjectField();
```

Description

Do not call Destroy directly in an application. The destructor is called automatically by the dataset that owns the object field component, and applications never need to call it explicitly.

If an application creates a descendant of TObjectField (see page 7372) without an owner, it should call Free instead of Destroy. Free checks that the field reference is not nil, and only then calls Destroy.

Destroy closes the dataset, deletes the instance of TObjectField (see page 7372) from the dataset, and calls the inherited destructor.

See Also

Free (see page 22300)

39.1.48.4 TObjectField Methods

The methods of the TObjectField class are listed here.

Protected Methods

	Name	Description
	CheckTypeSize (see page 7380)	This is CheckTypeSize, a member of class TObjectField.
	DefineProperties (see page 7380)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	FreeBuffers (see page 7380)	This is FreeBuffers, a member of class TObjectField.
	GetAsString (see page 7381)	This is GetAsString, a member of class TObjectField.
	GetAsVariant (see page 7381)	This is GetAsVariant, a member of class TObjectField.
	GetDefaultWidth (see page 7381)	This is GetDefaultWidth, a member of class TObjectField.
	GetFieldCount (see page 7381)	This is GetFieldCount, a member of class TObjectField.
	GetFields (see page 7382)	This is GetFields, a member of class TObjectField.
	GetFieldValue (see page 7382)	This is GetFieldValue, a member of class TObjectField.
	GetHasConstraints (see page 7382)	This is GetHasConstraints, a member of class TObjectField.
	SetChildOrder (see page 7382)	This is SetChildOrder, a member of class TObjectField.
	SetDataSet (see page 7383)	This is SetDataSet, a member of class TObjectField.
	SetFieldKind (see page 7383)	This is SetFieldKind, a member of class TObjectField.
	SetFieldValue (see page 7383)	This is SetFieldValue, a member of class TObjectField.
	SetParentField (see page 7383)	This is SetParentField, a member of class TObjectField.
	SetUnNamed (see page 7383)	This is SetUnNamed, a member of class TObjectField.
	SetVarValue (see page 7384)	This is SetVarValue, a member of class TObjectField.

Public Methods

	Name	Description
	GetChildren (see page 7381)	This is GetChildren, a member of class TObjectField.

39.1.48.4.1 TObjectField.CheckTypeSize Method

Pascal

```
class procedure CheckTypeSize(Value: Integer); override;
```

C++

```
virtual __fastcall CheckTypeSize(int Value);
```

Description

This is CheckTypeSize, a member of class TObjectField.

39.1.48.4.2 TObjectField.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor (see page 2839)

DefineBinaryProperty (see page 2837)

DefineProperties (see page 2931)

DefineProperty (see page 2838)

39.1.48.4.3 TObjectField.FreeBuffers Method

Pascal

```
procedure FreeBuffers; override;
```

C++

```
virtual __fastcall FreeBuffers();
```

Description

This is FreeBuffers, a member of class TObjectField.

39.1.48.4.4 TObjectField.GetAsString Method

Pascal

```
function GetAsString: string; override;
```

C++

```
virtual __fastcall AnsiString GetAsString();
```

Description

This is GetAsString, a member of class TObjectField.

39.1.48.4.5 TObjectField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TObjectField.

39.1.48.4.6 TObjectField.GetChildren Method

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

This is GetChildren, a member of class TObjectField.

39.1.48.4.7 TObjectField.GetDefaultWidth Method

Pascal

```
function GetDefaultWidth: Integer; override;
```

C++

```
virtual __fastcall int GetDefaultWidth();
```

Description

This is GetDefaultWidth, a member of class TObjectField.

39.1.48.4.8 TObjectField.GetFieldCount Method

Pascal

```
function GetFieldCount: Integer;
```

C++

```
__fastcall int GetFieldCount();
```

Description

This is GetFieldCount, a member of class TObjectField.

39.1.48.4.9 TObjectField.GetFields Method

Pascal

```
function GetFields: TFields; virtual;
```

C++

```
virtual __fastcall TFields GetFields();
```

Description

This is GetFields, a member of class TObjectField.

39.1.48.4.10 TObjectField.GetFieldValue Method

Pascal

```
function GetFieldValue(Index: Integer): Variant; virtual;
```

C++

```
virtual __fastcall Variant GetFieldValue(int Index);
```

Description

This is GetFieldValue, a member of class TObjectField.

39.1.48.4.11 TObjectField.GetHasConstraints Method

Pascal

```
function GetHasConstraints: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetHasConstraints();
```

Description

This is GetHasConstraints, a member of class TObjectField.

39.1.48.4.12 TObjectField.SetChildOrder Method

Pascal

```
procedure SetChildOrder(Component: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Component, int Order);
```

Description

This is SetChildOrder, a member of class TObjectField.

39.1.48.4.13 TObjectField.SetDataSet Method

Pascal

```
procedure SetDataSet(ADataSet: TDataSet); override;
```

C++

```
virtual __fastcall SetDataSet(TDataSet ADataSet);
```

Description

This is SetDataSet, a member of class TObjectField.

39.1.48.4.14 TObjectField.SetFieldKind Method

Pascal

```
procedure SetFieldKind(Value: TFieldKind); override;
```

C++

```
virtual __fastcall SetFieldKind(TFieldKind Value);
```

Description

This is SetFieldKind, a member of class TObjectField.

39.1.48.4.15 TObjectField.SetFieldValue Method

Pascal

```
procedure SetFieldValue(Index: Integer; const Value: Variant); virtual;
```

C++

```
virtual __fastcall SetFieldValue(int Index, const Variant Value);
```

Description

This is SetFieldValue, a member of class TObjectField.

39.1.48.4.16 TObjectField.SetParentField Method

Pascal

```
procedure SetParentField(AField: TObjectField); override;
```

C++

```
virtual __fastcall SetParentField(TObjectField AField);
```

Description

This is SetParentField, a member of class TObjectField.

39.1.48.4.17 TObjectField.SetUnNamed Method

Pascal

```
procedure SetUnNamed(Value: Boolean);
```

C++

```
__fastcall SetUnNamed(Boolean Value);
```

Description

This is SetUnNamed, a member of class TObjectField.

39.1.48.4.18 TObjectField.SetVarValue Method

Pascal

```
procedure SetVarValue(const Value: Variant); override;
```

C++

```
virtual __fastcall SetVarValue(const Variant Value);
```

Description

This is SetVarValue, a member of class TObjectField.

39.1.48.5 TObjectField Properties

The properties of the TObjectField class are listed here.

Public Properties

	Name	Description
R	FieldCount (see page 7384)	Indicates the number of child fields that comprise the object field.
R	Fields (see page 7384)	Specifies the TFields (see page 7211) object that contains an array of child fields.
	FieldValues (see page 7385)	Provides access to the values for all child fields in the object field.
R	UnNamed (see page 7386)	Indicates whether an object field in a nested table has a name associated with it.

Published Properties

	Name	Description
	ObjectType (see page 7385)	Specifies the classification of the field object in Oracle.

39.1.48.5.1 TObjectField.FieldCount Property

Indicates the number of child fields that comprise the object field.

Pascal

```
property FieldCount: Integer;
```

C++

```
__property int FieldCount;
```

Description

Examine FieldCount to determine the number of child fields associated with the object field. This is the number of fields managed by the Fields (see page 7384) property.

See Also

Fields (see page 7384)

FieldValues (see page 7385)

39.1.48.5.2 TObjectField.Fields Property

Specifies the TFields (see page 7211) object that contains an array of child fields.

Pascal

```
property Fields: TFields;
```

C++

```
__property TFields Fields;
```

Description

Child fields of the object field are maintained in a TFields (see page 7211) object specified by the Fields property. Use this property to access individual subfield components.

See Also

Working with ADT Fields

Working with Array Fields

39.1.48.5.3 TObjectField.FieldValues Property

Provides access to the values for all child fields in the object field.

Pascal

```
property FieldValues [Index: Integer]: Variant;
```

C++

```
__property Variant FieldValues[int Index];
```

Description

Use FieldValues to get or set the values of the object field's subfields. FieldValues accepts and returns a Variant, so it can handle and convert fields of any type. For example, the following statements are syntactically identical and write the value from an edit box into an ADT field with a child string field called STREET:

```
DataSet1.FieldByName('ADDRESS.STREET').Value := Edit1.Text;
TADTField(DataSet1.FieldByName('ADDRESS')).FieldValues[0] := Edit1.Text;
Table1->FieldByName("ADDRESS.STREET")->Value = Edit1->Text;
((TADTField*)Table1->FieldByName("ADDRESS"))->FieldValues[0] = Edit1->Text;
```

The next statements reads a string value from the first child field of the object field ADDRESS into an edit box:

```
Edit1.Text := DataSet1.FieldByName('ADDRESS').FieldValues[0];
Edit1->Text = ((TADTField*)DataSet1->FieldByName("ADDRESS"))->FieldValues[0];
```

Note: Because FieldValues always uses Variants (see page 23567), it may be slower than directly accessing a subfield and using its native format (i.e., using its AsXXX property), especially in applications that process large amounts of data.

See Also

Value

Fields (see page 7384)

operator_sb

Working with ADT Fields

39.1.48.5.4 TObjectField.ObjectType Property

Specifies the classification of the field object in Oracle.

Pascal

```
property ObjectType: WideString;
```

C++

```
__property BSTR ObjectType;
```

Description

In Oracle, objects are classified into object types.

39.1.48.5.5 TObjectField.UnNamed Property

Indicates whether an object field in a nested table has a name associated with it.

Pascal

```
property UnNamed: Boolean;
```

C++

```
__property Boolean UnNamed;
```

Description

UnNamed is set to true when the object field in a nested table has no name associated with it.

39.1.49 TParam Class

TParam represents a field parameter for a query or stored procedure.

Class Hierarchy

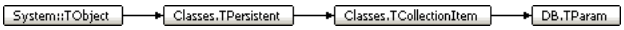

Pascal

```
TParam = class(TCollectionItem);
```

C++

```
class TParam : public TCollectionItem;
```

File

DB

Description

Use the properties of a TParam to set the value of a parameter that represents the value of a field. Datasets that represent queries and stored procedures create TParam objects to represent the parameters of their query or stored procedures. Client datasets use TParam objects to pass parameter values to a source dataset.

Note: ADO-based datasets do not use TParam objects to represent parameter values. They use the TParameter class instead.

TParam shares many properties with TField (see page 7115), as both describe the value of a field in a dataset. However, a TField (see page 7115) object has several properties to describe the field binding, and how the field is displayed, edited, or calculated that are not needed in a TParam object. Conversely, TParam includes properties that indicate how the field value is passed as a parameter.

See Also

TField (see page 7115)

TParams (see page 7434)

TCustomClientDataSet (see page 8373)

TCustomSQLDataSet (see page 20150)

TQuery (see page 10059)

TStoredProc (see page 10139)

TXMLBroker (see page 26049)

Using Parameters in Queries

39.1.49.1 TParam Members

The following tables list the members exposed by TParam.

Protected Methods

	Name	Description
	AssignParam (see page 7393)	This is AssignParam, a member of class TParam.
V	AssignTo (see page 7394)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetAsBCD (see page 7395)	This is GetAsBCD, a member of class TParam.
	GetAsBoolean (see page 7395)	This is GetAsBoolean, a member of class TParam.
	GetAsCurrency (see page 7395)	This is GetAsCurrency, a member of class TParam.
	GetAsDateTime (see page 7395)	This is GetAsDateTime, a member of class TParam.
	GetAsFloat (see page 7395)	This is GetAsFloat, a member of class TParam.
	GetAsFMTBCD (see page 7396)	This is GetAsFMTBCD, a member of class TParam.
	GetAsInteger (see page 7396)	This is GetAsInteger, a member of class TParam.
	GetAsMemo (see page 7396)	This is GetAsMemo, a member of class TParam.
	GetAsSQLTimeStamp (see page 7396)	This is GetAsSQLTimeStamp, a member of class TParam.
	GetAsString (see page 7397)	This is GetAsString, a member of class TParam.
	GetAsVariant (see page 7397)	This is GetAsVariant, a member of class TParam.
	GetAsWideString (see page 7397)	This is GetAsWideString, a member of class TParam.
V	GetDisplayName (see page 7399)	Returns the name of the collection item as it appears in the collection editor.
	GetIsNull (see page 7400)	This is GetIsNull, a member of class TParam.
V	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	IsEqual (see page 7400)	This is IsEqual, a member of class TParam.
	SetAsBCD (see page 7401)	This is SetAsBCD, a member of class TParam.
	SetAsBlob (see page 7401)	This is SetAsBlob, a member of class TParam.
	SetAsBoolean (see page 7401)	This is SetAsBoolean, a member of class TParam.
	SetAsCurrency (see page 7401)	This is SetAsCurrency, a member of class TParam.
	SetAsDate (see page 7402)	This is SetAsDate, a member of class TParam.
	SetAsDateTime (see page 7402)	This is SetAsDateTime, a member of class TParam.
	SetAsFloat (see page 7402)	This is SetAsFloat, a member of class TParam.
	SetAsFMTBCD (see page 7402)	This is SetAsFMTBCD, a member of class TParam.
	SetAsInteger (see page 7403)	This is SetAsInteger, a member of class TParam.
	SetAsMemo (see page 7403)	This is SetAsMemo, a member of class TParam.
	SetAsSmallInt (see page 7403)	This is SetAsSmallInt, a member of class TParam.
	SetAsSQLTimeStamp (see page 7403)	This is SetAsSQLTimeStamp, a member of class TParam.
	SetAsString (see page 7403)	This is SetAsString, a member of class TParam.
	SetAsTime (see page 7404)	This is SetAsTime, a member of class TParam.
	SetAsVariant (see page 7404)	This is SetAsVariant, a member of class TParam.
	SetAsWideString (see page 7404)	This is SetAsWideString, a member of class TParam.

	SetAsWord (see page 7404)	This is SetAsWord, a member of class TParam.
V	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDataType (see page 7405)	This is SetDataType, a member of class TParam.
V	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
V	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.
	SetText (see page 7406)	This is SetText, a member of class TParam.

Protected Properties

	Name	Description
R	DataSet (see page 7427)	This is DataSet, a member of class TParam.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 7391)	Copies the Name (see page 7430) and Value (see page 7433) of another TParam (see page 7386) object.
	AssignField (see page 7392)	Assigns the DataType (see page 7427), Name (see page 7430), and Value (see page 7433) properties from a TField (see page 7115) object.
	AssignFieldValue (see page 7393)	Assigns the DataType (see page 7427) and Value (see page 7433) properties from a TField (see page 7115) object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7394)	Sets the Value (see page 7433) of the parameter to NULL.
V	Create (see page 7389)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetData (see page 7397)	Fills the buffer with the Value (see page 7433) of the parameter in the native format of a database field of the appropriate type.
	GetDataSize (see page 7398)	Returns the number of bytes needed to hold the parameter's value in the format used by the corresponding type of database field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LoadFromFile (see page 7400)	Loads the value of a BLOB parameter from the specified file.
	LoadFromStream (see page 7400)	Loads the value of a BLOB parameter from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetBlobData (see page 7404)	Copies a new value for the parameter in physical database format from Buffer.
	SetData (see page 7405)	Copies a new value for the parameter in physical database format from Buffer.

Public Properties

	Name	Description
	AsBCD (see page 7407)	Specifies the value of the parameter when it represents a binary-coded decimal (BCD) field with less than 4 decimal places or 20 significant digits.
	AsBlob (see page 7408)	Specifies the value of the parameter when it represents a binary large object (BLOB) field.
	AsBoolean (see page 7409)	Specifies the value of the parameter when it represents a boolean field.
	AsCurrency (see page 7410)	Specifies the value of the parameter when it represents a field of type Currency.
	AsDate (see page 7412)	Specifies the value of the parameter when it represents a date field.
	AsDateTime (see page 7413)	Specifies the value of the parameter when it represents a date-time field.
	AsFloat (see page 7415)	Specifies the value of the parameter when it represents a float field.
	AsFMTBCD (see page 7416)	Specifies the value of the parameter when it represents a binary-coded decimal (BCD) field with more than 4 decimal places or 20 significant digits
	AsInteger (see page 7417)	Specifies the value of the parameter when it represents an integer field.
	AsMemo (see page 7419)	Specifies the value of the parameter when it represents a memo field.
	AsSmallInt (see page 7419)	Specifies the value of the parameter when it represents a small integer field.
	AsSQLTimeStamp (see page 7421)	Specifies the value of the parameter when it represents an SQL timestamp field.
	AsString (see page 7421)	Specifies the value of the parameter when it represents a string field.
	AsTime (see page 7423)	Specifies the value of the parameter when it represents a time field.
	AsWideString (see page 7424)	This is AsWideString, a member of class TParam.
	AsWord (see page 7425)	Specifies the value of the parameter when it represents a word field.
	Bound (see page 7426)	Indicates whether a value (NULL or otherwise) has been assigned to the parameter.
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
R	IsNull (see page 7429)	Indicates whether the value assigned to the parameter is NULL (blank).
	NativeStr (see page 7431)	Indicates the value of the parameter as a string, using the character set of the database table.
	Text (see page 7433)	Represents the value of the parameter as a string.

Published Properties

	Name	Description
	DataType (see page 7427)	Indicates the type of field whose value the parameter represents.
	Name (see page 7430)	Specifies the name of the parameter.
	NumericScale (see page 7431)	Specifies the number of decimal places for the parameter.
	ParamType (see page 7431)	Indicates the type of the parameter the TParam (see page 7386) represents.
	Precision (see page 7432)	Specifies the number of digits allowed for a numeric parameter.
	Size (see page 7432)	Specifies the number of characters in a string-type parameter.
	Value (see page 7433)	Represents the value of the parameter as a Variant.

39.1.49.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TParam.Create (TCollection) (see page 7390)	Creates an instance of TParam (see page 7386) and inserts it in a parameter list.
	TParam.Create (TParams, TParamType) (see page 7390)	Creates an instance of TParam (see page 7386) and inserts it in a parameter list.

39.1.49.2.1 TParam.Create Constructor (TCollection)

Creates an instance of TParam (see page 7386) and inserts it in a parameter list.

Pascal

```
constructor Create(Collection: TCollection); override; overload;
constructor Create(AParams: TParams; AParamType: TParamType); overload; reintroduce;
```

C++

```
virtual __fastcall TParam(TCollection Collection);
__fastcall TParam(TParams AParams, TParamType AParamType);
```

Description

Call Create to instantiate a TParam (see page 7386) object. TParam (see page 7386) objects are usually created by the dataset that uses them to represent its field parameters.

AParams is the TParams (see page 7434) object to which the parameter belongs.

AParamType indicates how the parameter is to be used. It becomes the value of the ParamType (see page 7431) property.

Collection is the TParams (see page 7434) object to which the parameter belongs.

See Also

Bound (see page 7426)

DataType (see page 7427)

ParamType (see page 7431)

TParams (see page 7434)

39.1.49.3 TParam Methods

The methods of the TParam class are listed here.

Protected Methods

	Name	Description
	AssignParam (see page 7393)	This is AssignParam, a member of class TParam.
	AssignTo (see page 7394)	Copies the properties of an object to a destination object.
	GetAsBCD (see page 7395)	This is GetAsBCD, a member of class TParam.
	GetAsBoolean (see page 7395)	This is GetAsBoolean, a member of class TParam.
	GetAsCurrency (see page 7395)	This is GetAsCurrency, a member of class TParam.
	GetAsDateTime (see page 7395)	This is GetAsDateTime, a member of class TParam.
	GetAsFloat (see page 7395)	This is GetAsFloat, a member of class TParam.
	GetAsFMTBCD (see page 7396)	This is GetAsFMTBCD, a member of class TParam.
	GetAsInteger (see page 7396)	This is GetAsInteger, a member of class TParam.

	GetAsMemo (see page 7396)	This is GetAsMemo, a member of class TParam.
	GetAsSQLTimeStamp (see page 7396)	This is GetAsSQLTimeStamp, a member of class TParam.
	GetAsString (see page 7397)	This is GetAsString, a member of class TParam.
	GetAsVariant (see page 7397)	This is GetAsVariant, a member of class TParam.
	GetAsWideString (see page 7397)	This is GetAsWideString, a member of class TParam.
	GetDisplayName (see page 7399)	Returns the name of the collection item as it appears in the collection editor.
	GetIsNull (see page 7400)	This is GetIsNull, a member of class TParam.
	IsEqual (see page 7400)	This is IsEqual, a member of class TParam.
	SetAsBCD (see page 7401)	This is SetAsBCD, a member of class TParam.
	SetAsBlob (see page 7401)	This is SetAsBlob, a member of class TParam.
	SetAsBoolean (see page 7401)	This is SetAsBoolean, a member of class TParam.
	SetAsCurrency (see page 7401)	This is SetAsCurrency, a member of class TParam.
	SetAsDate (see page 7402)	This is SetAsDate, a member of class TParam.
	SetAsDateTime (see page 7402)	This is SetAsDateTime, a member of class TParam.
	SetAsFloat (see page 7402)	This is SetAsFloat, a member of class TParam.
	SetAsFMTBCD (see page 7402)	This is SetAsFMTBCD, a member of class TParam.
	SetAsInteger (see page 7403)	This is SetAsInteger, a member of class TParam.
	SetAsMemo (see page 7403)	This is SetAsMemo, a member of class TParam.
	SetAsSmallInt (see page 7403)	This is SetAsSmallInt, a member of class TParam.
	SetAsSQLTimeStamp (see page 7403)	This is SetAsSQLTimeStamp, a member of class TParam.
	SetAsString (see page 7403)	This is SetAsString, a member of class TParam.
	SetAsTime (see page 7404)	This is SetAsTime, a member of class TParam.
	SetAsVariant (see page 7404)	This is SetAsVariant, a member of class TParam.
	SetAsWideString (see page 7404)	This is SetAsWideString, a member of class TParam.
	SetAsWord (see page 7404)	This is SetAsWord, a member of class TParam.
	SetDataType (see page 7405)	This is SetDataType, a member of class TParam.
	SetText (see page 7406)	This is SetText, a member of class TParam.

Public Methods

	Name	Description
	Assign (see page 7391)	Copies the Name (see page 7430) and Value (see page 7433) of another TParam (see page 7386) object.
	AssignField (see page 7392)	Assigns the DataType (see page 7427), Name (see page 7430), and Value (see page 7433) properties from a TField (see page 7115) object.
	AssignFieldValue (see page 7393)	Assigns the DataType (see page 7427) and Value (see page 7433) properties from a TField (see page 7115) object.
	Clear (see page 7394)	Sets the Value (see page 7433) of the parameter to NULL.
	GetData (see page 7397)	Fills the buffer with the Value (see page 7433) of the parameter in the native format of a database field of the appropriate type.
	GetDataSize (see page 7398)	Returns the number of bytes needed to hold the parameter's value in the format used by the corresponding type of database field.
	LoadFromFile (see page 7400)	Loads the value of a BLOB parameter from the specified file.
	LoadFromStream (see page 7400)	Loads the value of a BLOB parameter from a stream.
	SetBlobData (see page 7404)	Copies a new value for the parameter in physical database format from Buffer.
	SetData (see page 7405)	Copies a new value for the parameter in physical database format from Buffer.

39.1.49.3.1 TParam.Assign Method

Copies the Name (see page 7430) and Value (see page 7433) of another TParam (see page 7386) object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use Assign to copy from another parameter or from a field component. If the Source parameter is another parameter, Assign transfers the value of

The DataType (see page 7427) property

The Bound (see page 7426) property

The Value (see page 7433) property

The Name (see page 7430) property

The ParamType (see page 7431) property only if it is currently ftUnknown.

If the Source parameter is a field component, Assign transfers the value of

The DataType (see page 7427) property

The Value (see page 7433) property

The Name (see page 7430) property (from the FieldName of the field)

When assigning values from a field, Assign sets the Bound (see page 7426) property to true.

Note: Assign does not alter the use of a parameter, only the identifier and the value.

See Also

AssignField (see page 7392)

Bound (see page 7426)

DataType (see page 7427)

Name (see page 7430)

ParamType (see page 7431)

TField (see page 7115)

Value (see page 7433)

39.1.49.3.2 TParam.AssignField Method

Assigns the DataType (see page 7427), Name (see page 7430), and Value (see page 7433) properties from a TField (see page 7115) object.

Pascal

```
procedure AssignField(Field: TField);
```

C++

```
__fastcall AssignField(TField Field);
```

Description

Use AssignField to set a parameter to represent a particular TField (see page 7115) object. AssignField sets the Bound (see page 7426) property to true.

Unlike AssignFieldValue (see page 7393), AssignField names the parameter after the field object as well as taking its value.

See Also

Assign (see page 7391)

AssignFieldValue (see page 7393)

Bound (see page 7426)

DataType (see page 7427)

Name (see page 7430)

DataType (see page 7159)

FieldName (see page 7164)

Value (see page 7180)

Value (see page 7433)

39.1.49.3.3 TParam.AssignFieldValue Method

Assigns the DataType (see page 7427) and Value (see page 7433) properties from a TField (see page 7115) object.

Pascal

```
procedure AssignFieldValue(Field: TField; const Value: Variant);
```

C++

```
__fastcall AssignFieldValue(TField Field, const Variant Value);
```

Description

Use AssignFieldValue to set the Value (see page 7433) property to the value passed as the Value (see page 7433) parameter. AssignFieldValue assumes that the Value (see page 7433) represents a value from a field like the Field parameter, and assigns the DataType (see page 7427) property accordingly. AssignFieldValue sets the Bound (see page 7426) property to true.

Unlike AssignField (see page 7392), AssignFieldValue does not name the parameter after the field object.

See Also

AssignField (see page 7392)

Bound (see page 7426)

DataType (see page 7427)

DataType (see page 7159)

FieldName (see page 7164)

Value (see page 7180)

Value (see page 7433)

39.1.49.3.4 TParam.AssignParam Method

Pascal

```
procedure AssignParam(Param: TParam);
```

C++

```
__fastcall AssignParam(TParam Param);
```

Description

This is AssignParam, a member of class TParam.

39.1.49.3.5 TParam.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign (see page 2930)

EConvertError (see page 22724)

39.1.49.3.6 TParam.Clear Method

Sets the Value (see page 7433) of the parameter to NULL.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Use Clear to assign a NULL value to a parameter. Calling Clear neither sets nor clears the Bound (see page 7426) property. When assigning a NULL value to a parameter, set the Bound (see page 7426) property as well as calling Clear.

See Also

Bound (see page 7426)

IsNull (see page 7429)

Value (see page 7433)

39.1.49.3.7 TParam.GetAsBCD Method

Pascal

```
function GetAsBCD: Currency;
```

C++

```
__fastcall Currency GetAsBCD();
```

Description

This is GetAsBCD, a member of class TParam.

39.1.49.3.8 TParam.GetAsBoolean Method

Pascal

```
function GetAsBoolean: Boolean;
```

C++

```
__fastcall Boolean GetAsBoolean();
```

Description

This is GetAsBoolean, a member of class TParam.

39.1.49.3.9 TParam.GetAsCurrency Method

Pascal

```
function GetAsCurrency: Currency;
```

C++

```
__fastcall Currency GetAsCurrency();
```

Description

This is GetAsCurrency, a member of class TParam.

39.1.49.3.10 TParam.GetAsDateTime Method

Pascal

```
function GetAsDateTime: TDateTime;
```

C++

```
__fastcall TDateTime GetAsDateTime();
```

Description

This is GetAsDateTime, a member of class TParam.

39.1.49.3.11 TParam.GetAsFloat Method

Pascal

```
function GetAsFloat: Double;
```

C++

```
__fastcall Double GetAsFloat();
```

Description

This is GetAsFloat, a member of class TParam.

39.1.49.3.12 TParam.GetAsFMTBCD Method

Pascal

```
function GetAsFMTBCD: TBcd;
```

C++

```
__fastcall TBcd GetAsFMTBCD();
```

Description

This is GetAsFMTBCD, a member of class TParam.

39.1.49.3.13 TParam.GetAsInteger Method

Pascal

```
function GetAsInteger: Longint;
```

C++

```
__fastcall Longint GetAsInteger();
```

Description

This is GetAsInteger, a member of class TParam.

39.1.49.3.14 TParam.GetAsMemo Method

Pascal

```
function GetAsMemo: string;
```

C++

```
__fastcall AnsiString GetAsMemo();
```

Description

This is GetAsMemo, a member of class TParam.

39.1.49.3.15 TParam.GetAsSQLTimeStamp Method

Pascal

```
function GetAsSQLTimeStamp: TSQLTimeStamp;
```

C++

```
__fastcall TSQLTimeStamp GetAsSQLTimeStamp();
```

Description

This is GetAsSQLTimeStamp, a member of class TParam.

39.1.49.3.16 TParam.GetAsString Method

Pascal

```
function GetAsString: string;
```

C++

```
__fastcall AnsiString GetAsString();
```

Description

This is GetAsString, a member of class TParam.

39.1.49.3.17 TParam.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant;
```

C++

```
__fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TParam.

39.1.49.3.18 TParam.GetAsWideString Method

Pascal

```
function GetAsWideString: WideString;
```

C++

```
__fastcall BSTR GetAsWideString();
```

Description

This is GetAsWideString, a member of class TParam.

39.1.49.3.19 TParam.GetData Method

Fills the buffer with the Value (see page 7433) of the parameter in the native format of a database field of the appropriate type.

Pascal

```
procedure GetData(Buffer: Pointer);
```

C++

```
__fastcall GetData(void * Buffer);
```

Description

Use GetData to obtain the value of a parameter in the format ready to be written directly to the underlying database. That format is the format used by physical database fields corresponding to DataType (see page 7427). Buffer must have enough space to hold the information. Use the GetDataSize (see page 7398) method to determine the necessary size.

The native format for a database field may not be the same as the datatype used by the corresponding field object. Use the As... properties to get the value in the type used by the corresponding field object. Use GetData to get the value in the native database format.

For example, when DataType (see page 7427) is ftBCD, GetData retrieves the value in binary-coded decimal, even though the

TBCDField (see page 6776) object uses Currency to store and manipulate its values. AsBCD (see page 7407) gets the value in the Currency type.

See Also

DataType (see page 7427)

GetDataSize (see page 7398)

SetData (see page 7405)

GetData (see page 7131)

SetData (see page 7141)

Value (see page 7433)

39.1.49.3.20 TParam.GetDataSize Method

Returns the number of bytes needed to hold the parameter's value in the format used by the corresponding type of database field.

Pascal

```
function GetDataSize: Integer;
```

C++

```
__fastcall int GetDataSize();
```

Description

Call GetDataSize to determine the number of bytes used to represent the parameter's value. GetDataSize indicates the buffer size needed before calling the GetData (see page 7397) method. The value returned by GetDataSize depends on the Value (see page 7433) of the parameter. For some values of DataType (see page 7427), this is always the same:

DataType (see page 7427)	Value (see page 7433) of DataSize
ftBoolean	2
ftSmallint	2
ftWord	2
ftAutoInc	4
ftDate	4
ftInteger	4
ftTime	4
ftCurrency	8
ftDateTime	8
ftFloat	8
ftTimeStamp	16
ftBCD	34
ftString	length of the string + 1
ftFixedChar	length of the string + 1
ftVarBytes	If the value is a Variant array, the number of elements in the array, else 0
ftBytes	If the value is a Variant array, the number of elements in the array, else 0

ftBlob	number of bytes of blob data
ftFmtMemo	number of bytes of blob data
ftGraphic	number of bytes of blob data
ftMemo	length of memo string + 1
ftDBaseOle	number of bytes of blob data
ftParadoxOle	number of bytes of blob data
ftOraBlob	number of bytes of blob data
ftOraClob	number of bytes of blob data
ftTypedBinary	number of bytes of data
ftADT	0
ftArray	0
ftDataSet	0
ftReference	0
ftCursor	0
ftUnknown	raises an EDatabaseError (see page 6731) exception

Note: The size returned by GetDataSize is the size required for the physical database representation of the Value (see page 7433). The field object that corresponds to that physical database field may store its data in another format requiring a different number of bytes.

See Also

DataType (see page 7427)

GetData (see page 7397)

DataSize (see page 7157)

Value (see page 7433)

39.1.49.3.21 TParam.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName (see page 2761)

GetNamePath (see page 2759)

39.1.49.3.22 TParam.GetIsNull Method

Pascal

```
function GetIsNull: Boolean;
```

C++

```
__fastcall Boolean GetIsNull();
```

Description

This is GetIsNull, a member of class TParam.

39.1.49.3.23 TParam.IsEqual Method

Pascal

```
function IsEqual(Value: TParam): Boolean;
```

C++

```
__fastcall Boolean IsEqual(TParam Value);
```

Description

This is IsEqual, a member of class TParam.

39.1.49.3.24 TParam.LoadFromFile Method

Loads the value of a BLOB parameter from the specified file.

Pascal

```
procedure LoadFromFile(const FileName: string; BlobType: TBlobType);
```

C++

```
__fastcall LoadFromFile(const AnsiString FileName, TBlobType BlobType);
```

Description

Use LoadFromFile to set the value of a BLOB parameter from a value stored in the file specified by the FileName parameter. The DataType (see page 7427) property is set to the value passed as the BlobType parameter.

```
Query2.ParamByName('Notes').LoadFromFile('c:\Files\Note1234.txt', ftMemo);
```

See Also

DataType (see page 7427)

LoadFromStream (see page 7400)

39.1.49.3.25 TParam.LoadFromStream Method

Loads the value of a BLOB parameter from a stream.

Pascal

```
procedure LoadFromStream(Stream: TStream; BlobType: TBlobType);
```

C++

```
__fastcall LoadFromStream(TStream Stream, TBlobType BlobType);
```

Description

Use LoadFromStream to set the value of a BLOB parameter from a value read from the stream specified by the Stream parameter. For example, use LoadFromStream to set the value of a BLOB parameter directly from a BLOB stream. The DataType (see page 7427) property is set to the value passed as the BlobType parameter.

See Also

DataType (see page 7427)

LoadFromFile (see page 7400)

CreateBlobStream (see page 6922)

TStream (see page 2973)

39.1.49.3.26 TParam.SetAsBCD Method

Pascal

```
procedure SetAsBCD(const Value: Currency);
```

C++

```
__fastcall SetAsBCD(const Currency Value);
```

Description

This is SetAsBCD, a member of class TParam.

39.1.49.3.27 TParam.SetAsBlob Method

Pascal

```
procedure SetAsBlob(const Value: TBlobData);
```

C++

```
__fastcall SetAsBlob(const TBlobData Value);
```

Description

This is SetAsBlob, a member of class TParam.

39.1.49.3.28 TParam.SetAsBoolean Method

Pascal

```
procedure SetAsBoolean(Value: Boolean);
```

C++

```
__fastcall SetAsBoolean(Boolean Value);
```

Description

This is SetAsBoolean, a member of class TParam.

39.1.49.3.29 TParam.SetAsCurrency Method

Pascal

```
procedure SetAsCurrency(const Value: Currency);
```

C++

```
__fastcall SetAsCurrency(const Currency Value);
```

Description

This is SetAsCurrency, a member of class TParam.

39.1.49.3.30 TParam.SetAsDate Method

Pascal

```
procedure SetAsDate(const Value: TDateTime);
```

C++

```
__fastcall SetAsDate(const TDateTime Value);
```

Description

This is SetAsDate, a member of class TParam.

39.1.49.3.31 TParam.SetAsDateTime Method

Pascal

```
procedure SetAsDateTime(const Value: TDateTime);
```

C++

```
__fastcall SetAsDateTime(const TDateTime Value);
```

Description

This is SetAsDateTime, a member of class TParam.

39.1.49.3.32 TParam.SetAsFloat Method

Pascal

```
procedure SetAsFloat(const Value: Double);
```

C++

```
__fastcall SetAsFloat(const Double Value);
```

Description

This is SetAsFloat, a member of class TParam.

39.1.49.3.33 TParam.SetAsFMTBCD Method

Pascal

```
procedure SetAsFMTBCD(const Value: TBcd);
```

C++

```
__fastcall SetAsFMTBCD(const TBcd Value);
```

Description

This is SetAsFMTBCD, a member of class TParam.

39.1.49.3.34 TParam.SetAsInteger Method

Pascal

```
procedure SetAsInteger(Value: Longint);
```

C++

```
__fastcall SetAsInteger(Longint Value);
```

Description

This is SetAsInteger, a member of class TParam.

39.1.49.3.35 TParam.SetAsMemo Method

Pascal

```
procedure SetAsMemo(const Value: string);
```

C++

```
__fastcall SetAsMemo(const AnsiString Value);
```

Description

This is SetAsMemo, a member of class TParam.

39.1.49.3.36 TParam.SetAsSmallInt Method

Pascal

```
procedure SetAsSmallInt(Value: LongInt);
```

C++

```
__fastcall SetAsSmallInt(LongInt Value);
```

Description

This is SetAsSmallInt, a member of class TParam.

39.1.49.3.37 TParam.SetAsSQLTimeStamp Method

Pascal

```
procedure SetAsSQLTimeStamp(const Value: TSQLTimeStamp);
```

C++

```
__fastcall SetAsSQLTimeStamp(const TSQLTimeStamp Value);
```

Description

This is SetAsSQLTimeStamp, a member of class TParam.

39.1.49.3.38 TParam.SetAsString Method

Pascal

```
procedure SetAsString(const Value: string);
```

C++

```
__fastcall SetAsString(const AnsiString Value);
```

Description

This is SetAsString, a member of class TParam.

39.1.49.3.39 TParam.SetAsTime Method

Pascal

```
procedure SetAsTime(const Value: TDateTime);
```

C++

```
__fastcall SetAsTime(const TDateTime Value);
```

Description

This is SetAsTime, a member of class TParam.

39.1.49.3.40 TParam.SetAsVariant Method

Pascal

```
procedure SetAsVariant(const Value: Variant);
```

C++

```
__fastcall SetAsVariant(const Variant Value);
```

Description

This is SetAsVariant, a member of class TParam.

39.1.49.3.41 TParam.SetAsWideString Method

Pascal

```
procedure SetAsWideString(const Value: WideString);
```

C++

```
__fastcall SetAsWideString(const BSTR Value);
```

Description

This is SetAsWideString, a member of class TParam.

39.1.49.3.42 TParam.SetAsWord Method

Pascal

```
procedure SetAsWord(Value: LongInt);
```

C++

```
__fastcall SetAsWord(LongInt Value);
```

Description

This is SetAsWord, a member of class TParam.

39.1.49.3.43 TParam.SetBlobData Method

Copies a new value for the parameter in physical database format from Buffer.

Pascal

```
procedure SetBlobData(Buffer: Pointer; Size: Integer);
```

C++

```
__fastcall SetBlobData(void * Buffer, int Size);
```

Description

Use SetBlobData to set the value of a parameter from a buffer that contains data from a Binary Large Object (BLOB) field. SetBlobData copies the number of bytes specified by the Size (see page 7432) parameter from the buffer specified by the Buffer parameter, and sets the DataType (see page 7427) property to ftBlob.

See Also

AsBlob (see page 7408)

DataType (see page 7427)

SetData (see page 7405)

39.1.49.3.44 TParam.SetData Method

Copies a new value for the parameter in physical database format from Buffer.

Pascal

```
procedure SetData(Buffer: Pointer);
```

C++

```
__fastcall SetData(void * Buffer);
```

Description

Use SetData to set the value of a parameter from a buffer that contains data in the physical database format. The GetData (see page 7397) method of a TField (see page 7115) object fetches data in the physical database format. Borland (see page 1761) Database Engine API calls can also provide data in physical database format.

Note: When the DataType (see page 7427) property is ftUnknown, ftBytes, ftVarBytes, ftAutoInc, ftBlob, ftGraphic, ftFmtMemo, ftParadoxOle, ftDBaseOle, or ftTypedBinary, SetData raises an EDatabaseError (see page 6731) exception. For BLOB fields, use the SetBlobData (see page 7404) method instead.

See Also

GetData (see page 7397)

SetBlobData (see page 7404)

GetData (see page 7131)

Value (see page 7433)

39.1.49.3.45 TParam.SetDataType Method

Pascal

```
procedure SetDataType(Value: TFieldType);
```

C++

```
__fastcall SetDataType(TFieldType Value);
```

Description

This is SetDataType, a member of class TParam.

39.1.49.3.46 TParam.SetText Method

Pascal

```
procedure SetText(const Value: string);
```

C++

```
__fastcall SetText(const AnsiString Value);
```

Description

This is SetText, a member of class TParam.

39.1.49.4 TParam Properties

The properties of the TParam class are listed here.

Protected Properties

	Name	Description
R	DataSet (see page 7427)	This is DataSet, a member of class TParam.

Public Properties

	Name	Description
	AsBCD (see page 7407)	Specifies the value of the parameter when it represents a binary-coded decimal (BCD) field with less than 4 decimal places or 20 significant digits.
	AsBlob (see page 7408)	Specifies the value of the parameter when it represents a binary large object (BLOB) field.
	AsBoolean (see page 7409)	Specifies the value of the parameter when it represents a boolean field.
	AsCurrency (see page 7410)	Specifies the value of the parameter when it represents a field of type Currency.
	AsDate (see page 7412)	Specifies the value of the parameter when it represents a date field.
	AsDateTime (see page 7413)	Specifies the value of the parameter when it represents a date-time field.
	AsFloat (see page 7415)	Specifies the value of the parameter when it represents a float field.
	AsFMTBCD (see page 7416)	Specifies the value of the parameter when it represents a binary-coded decimal (BCD) field with more than 4 decimal places or 20 significant digits
	AsInteger (see page 7417)	Specifies the value of the parameter when it represents an integer field.
	AsMemo (see page 7419)	Specifies the value of the parameter when it represents a memo field.
	AsSmallInt (see page 7419)	Specifies the value of the parameter when it represents a small integer field.
	AsSQLTimeStamp (see page 7421)	Specifies the value of the parameter when it represents an SQL timestamp field.
	AsString (see page 7421)	Specifies the value of the parameter when it represents a string field.
	AsTime (see page 7423)	Specifies the value of the parameter when it represents a time field.
	AsWideString (see page 7424)	This is AsWideString, a member of class TParam.
	AsWord (see page 7425)	Specifies the value of the parameter when it represents a word field.
	Bound (see page 7426)	Indicates whether a value (NULL or otherwise) has been assigned to the parameter.
R	IsNull (see page 7429)	Indicates whether the value assigned to the parameter is NULL (blank).
	NativeStr (see page 7431)	Indicates the value of the parameter as a string, using the character set of the database table.
	Text (see page 7433)	Represents the value of the parameter as a string.

Published Properties

	Name	Description
	DataType (see page 7427)	Indicates the type of field whose value the parameter represents.

	Name (see page 7430)	Specifies the name of the parameter.
	NumericScale (see page 7431)	Specifies the number of decimal places for the parameter.
	ParamType (see page 7431)	Indicates the type of the parameter the TParam (see page 7386) represents.
	Precision (see page 7432)	Specifies the number of digits allowed for a numeric parameter.
	Size (see page 7432)	Specifies the number of characters in a string-type parameter.
	Value (see page 7433)	Represents the value of the parameter as a Variant.

39.1.49.4.1 TParam.AsBCD Property

Specifies the value of the parameter when it represents a binary-coded decimal (BCD) field with less than 4 decimal places or 20 significant digits.

Pascal

```
property AsBCD: Currency;
```

C++

```
__property Currency AsBCD;
```

Description

Set AsBCD to assign the value for a BCD field to the parameter when the BCD field can be converted to the Currency type without a loss of precision. TBCDField (see page 6776) objects use the Currency type to represent the value of BCD fields, so AsBCD takes a Currency value as well. Setting AsBCD sets the DataType (see page 7427) property to ftBCD.

Read the AsBCD property to determine the value that was assigned to an output parameter, represented as a Currency type. The value of the parameter is converted to a Currency value if possible.

Tip: If the BCD field requires greater precision than allowed by the Currency type, use a TFMTBCDField (see page 7242) object and the AsFMTBCD (see page 7416) property instead.

Delphi Examples:

```
{
This example fills in the parameters of a client dataset
from the entries of a list box.
}
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
  ListItem: string;
begin
  { Call FetchParams to ensure parameters reflect server metadata }
  ClientDataSet1.FetchParams;
  with ClientDataSet1.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
```

```
              Items[I].AsBoolean := True
            else
              Items[I].AsBoolean := False;
          end;
        ftFloat:
          Items[I].AsFloat := StrToFloat(ListItem);
        ftCurrency:
          Items[I].AsCurrency := StrToFloat(ListItem);
        ftBCD:
          Items[I].AsBCD := StrToCurr(ListItem);
        ftDate:
          Items[I].AsDate := StrToDate(ListItem);
        ftTime:
          Items[I].AsTime := StrToTime(ListItem);
        ftDateTime:
          Items[I].AsDateTime := StrToDateTime(ListItem);
      end;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Here is a string');
  ListBox1.Items.Add('4');
  ListBox1.Items.Add('3456789');
  ListBox1.Items.Add('34');
  ListBox1.Items.Add('0');
  ListBox1.Items.Add('12.56');
  ListBox1.Items.Add('4.23');
  ListBox1.Items.Add('99.5');
  ListBox1.Items.Add('4/12/53');
  ListBox1.Items.Add('7:45:00');
  ListBox1.Items.Add('4/12/53 7:45:00');

end;
```

See Also

DataType (see page 7427)

TBCDField (see page 6776)

AsFMTBCD (see page 7416)

TFMTBCDField (see page 7242)

39.1.49.4.2 TParam.AsBlob Property

Specifies the value of the parameter when it represents a binary large object (BLOB) field.

Pascal

```
property AsBlob: TBlobData;
```

C++

```
__property TBlobData AsBlob;
```

Description

Set AsBlob to assign the value for a Blob field to the parameter. AsBlob takes a TBlobData (see page 7627) (Delphi) or an AnsiString (C++) value, which can act as an untyped string of bytes. Setting AsBlob sets the DataType (see page 7427) property to ftBlob.

For example, if the variable Buffer is of type PChar and contains binary data read from a file, the contents of Buffer are directly

assigned to the BLOB parameter using the AsBlob property:

```
Query1.Params[0].AsBlob := Buffer;
```

Note: Applications seldom need to read AsBlob because Blob fields can't be used as output parameters.

See Also

DataType (see page 7427)

TBlobField (see page 6801)

39.1.49.4.3 TParam.AsBoolean Property

Specifies the value of the parameter when it represents a boolean field.

Pascal

```
property AsBoolean: Boolean;
```

C++

```
__property Boolean AsBoolean;
```

Description

Set AsBoolean to assign the value for a boolean field to the parameter. Setting AsBoolean sets the DataType (see page 7427) property to ftBoolean.

Read the AsBoolean property to determine the value that was assigned to an output parameter, represented as a Boolean (Delphi) or bool (C++). The value of the parameter is converted to a Boolean or bool value if possible.

Delphi Examples:

```
{
This example fills in the parameters of a client dataset
from the entries of a list box.
}
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
  ListItem: string;
begin
  { Call FetchParams to ensure parameters reflect server metadata }
  ClientDataSet1.FetchParams;
  with ClientDataSet1.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
              Items[I].AsBoolean := True
            else
              Items[I].AsBoolean := False;
          end;
        ftFloat:
```

```
          Items[I].AsFloat := StrToFloat(ListItem);
        ftCurrency:
          Items[I].AsCurrency := StrToFloat(ListItem);
        ftBCD:
          Items[I].AsBCD := StrToCurr(ListItem);
        ftDate:
          Items[I].AsDate := StrToDate(ListItem);
        ftTime:
          Items[I].AsTime := StrToTime(ListItem);
        ftDateTime:
          Items[I].AsDateTime := StrToDateTime(ListItem);
      end;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Here is a string');
  ListBox1.Items.Add('4');
  ListBox1.Items.Add('3456789');
  ListBox1.Items.Add('34');
  ListBox1.Items.Add('0');
  ListBox1.Items.Add('12.56');
  ListBox1.Items.Add('4.23');
  ListBox1.Items.Add('99.5');
  ListBox1.Items.Add('4/12/53');
  ListBox1.Items.Add('7:45:00');
  ListBox1.Items.Add('4/12/53 7:45:00');

end;
```

See Also

DataType (see page 7427)

TBooleanField (see page 6822)

39.1.49.4.4 TParam.AsCurrency Property

Specifies the value of the parameter when it represents a field of type Currency.

Pascal

```
property AsCurrency: Currency;
```

C++

```
__property Currency AsCurrency;
```

Description

Set AsCurrency to assign the value for a parameter of type Currency. Setting AsCurrency sets the DataType (see page 7427) property to ftCurrency.

Read the AsCurrency property to determine the value that was assigned to an output parameter, represented as a Currency type. The value of the parameter is converted if possible.

Delphi Examples:

```
{
This example fills in the parameters of a client dataset
from the entries of a list box.
}
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
```

```
    ListItem: string;
  begin
    { Call FetchParams to ensure parameters reflect server metadata }
    ClientDataSet1.FetchParams;
    with ClientDataSet1.Params do
    begin
      for I := 0 to Count - 1 do
      begin
        ListItem := ListBox1.Items[I];
        case Items[I].DataType of
          ftString:
            Items[I].AsString := ListItem;
          ftSmallInt:
            Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
          ftInteger:
            Items[I].AsInteger := StrToIntDef(ListItem, 0);
          ftWord:
            Items[I].AsWord := StrToIntDef(ListItem, 0);
          ftBoolean:
            begin
              if ListItem = 'True' then
                Items[I].AsBoolean := True
              else
                Items[I].AsBoolean := False;
            end;
          ftFloat:
            Items[I].AsFloat := StrToFloat(ListItem);
          ftCurrency:
            Items[I].AsCurrency := StrToFloat(ListItem);
          ftBCD:
            Items[I].AsBCD := StrToCurr(ListItem);
          ftDate:
            Items[I].AsDate := StrToDate(ListItem);
          ftTime:
            Items[I].AsTime := StrToTime(ListItem);
          ftDateTime:
            Items[I].AsDateTime := StrToDateTime(ListItem);
        end;
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Here is a string');
    ListBox1.Items.Add('4');
    ListBox1.Items.Add('3456789');
    ListBox1.Items.Add('34');
    ListBox1.Items.Add('0');
    ListBox1.Items.Add('12.56');
    ListBox1.Items.Add('4.23');
    ListBox1.Items.Add('99.5');
    ListBox1.Items.Add('4/12/53');
    ListBox1.Items.Add('7:45:00');
    ListBox1.Items.Add('4/12/53 7:45:00');

  end;
```

See Also

AsBCD (see page 7407)

AsFloat (see page 7415)

DataType (see page 7427)

TCurrencyField (see page 6850)

39.1.49.4.5 TParam.AsDate Property

Specifies the value of the parameter when it represents a date field.

Pascal

```
property AsDate: TDateTime;
```

C++

```
__property TDateTime AsDate;
```

Description

Set AsDate to assign the value for a date field to the parameter. Setting AsDate sets the DataType (see page 7427) property to ftDate.

Read the AsDate property to determine the value that was assigned to an output parameter, represented as a TDateTime. The value of the parameter is converted to a DateTime value if possible. If the DataType (see page 7427) for the parameter is ftDate, only the date portion of the TDateTime value is valid. If a TDateTime value is assigned to a parameter via the AsDate property and the TDateTime value has a nonzero time, the time is ignored and only the date is assigned. If AsDate is used to get the value for a TDateTime container, the date from the parameter is stored but the TDateTime gets a zero time value.

Note: It is not possible to store a NULL value directly to a parameter using AsDate. Use the TParam (see page 7386) .

:: Clear (see page 7394) method instead.

Delphi Examples:

```
{
This example fills in the parameters of a client dataset
from the entries of a list box.
}
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
  ListItem: string;
begin
  { Call FetchParams to ensure parameters reflect server metadata }
  ClientDataSet1.FetchParams;
  with ClientDataSet1.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
              Items[I].AsBoolean := True
            else
              Items[I].AsBoolean := False;
          end;
        ftFloat:
          Items[I].AsFloat := StrToFloat(ListItem);
        ftCurrency:
          Items[I].AsCurrency := StrToFloat(ListItem);
```

```
          ftBCD:
            Items[I].AsBCD := StrToCurr(ListItem);
          ftDate:
            Items[I].AsDate := StrToDate(ListItem);
          ftTime:
            Items[I].AsTime := StrToTime(ListItem);
          ftDateTime:
            Items[I].AsDateTime := StrToDateTime(ListItem);
        end;
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Here is a string');
    ListBox1.Items.Add('4');
    ListBox1.Items.Add('3456789');
    ListBox1.Items.Add('34');
    ListBox1.Items.Add('0');
    ListBox1.Items.Add('12.56');
    ListBox1.Items.Add('4.23');
    ListBox1.Items.Add('99.5');
    ListBox1.Items.Add('4/12/53');
    ListBox1.Items.Add('7:45:00');
    ListBox1.Items.Add('4/12/53 7:45:00');

  end;
```

See Also

AsDateTime (see page 7413)

DataType (see page 7427)

TDateField (see page 7083)

TDate

TDateTime (see page 22612)

39.1.49.4.6 TParam.AsDateTime Property

Specifies the value of the parameter when it represents a date-time field.

Pascal

```
property AsDateTime: TDateTime;
```

C++

```
__property TDateTime AsDateTime;
```

Description

Set AsDateTime to assign the value for a date-time field to the parameter. Setting AsDateTime sets the DataType (see page 7427) property to ftDateTime.

Read the AsDateTime property to determine the value that was assigned to an output parameter, represented as a DateTime. The value of the parameter is converted to a DateTime value if possible. If a TDate value is assigned to a parameter via the AsDateTime property, the date portion of the TDate value is assigned to the parameter and the time portion is assigned a zero value. If AsDateTime is used to get the value for a TDate container, the date from the parameter is stored but the time value returned by AsDateTime is ignored.

Note: It is not possible to store a NULL value directly to a parameter using AsDateTime. Use the TParam (see page 7386) .

:: Clear (see page 7394) method instead.

Delphi Examples:

```
{
This example fills in the parameters of a client dataset
from the entries of a list box.
}
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
  ListItem: string;
begin
  { Call FetchParams to ensure parameters reflect server metadata }
  ClientDataSet1.FetchParams;
  with ClientDataSet1.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
              Items[I].AsBoolean := True
            else
              Items[I].AsBoolean := False;
          end;
        ftFloat:
          Items[I].AsFloat := StrToFloat(ListItem);
        ftCurrency:
          Items[I].AsCurrency := StrToFloat(ListItem);
        ftBCD:
          Items[I].AsBCD := StrToCurr(ListItem);
        ftDate:
          Items[I].AsDate := StrToDate(ListItem);
        ftTime:
          Items[I].AsTime := StrToTime(ListItem);
        ftDateTime:
          Items[I].AsDateTime := StrToDateTime(ListItem);
      end;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Here is a string');
  ListBox1.Items.Add('4');
  ListBox1.Items.Add('3456789');
  ListBox1.Items.Add('34');
  ListBox1.Items.Add('0');
  ListBox1.Items.Add('12.56');
  ListBox1.Items.Add('4.23');
  ListBox1.Items.Add('99.5');
  ListBox1.Items.Add('4/12/53');
  ListBox1.Items.Add('7:45:00');
  ListBox1.Items.Add('4/12/53 7:45:00');
```

```
end;
```

See Also

DataType (see page 7427)

TDateTimeField (see page 7090)

TDate

TDateTime (see page 22612)

39.1.49.4.7 TParam.AsFloat Property

Specifies the value of the parameter when it represents a float field.

Pascal

```
property AsFloat: Double;
```

C++

```
__property Double AsFloat;
```

Description

Set AsFloat to assign the value for a float field to the parameter. Setting AsFloat sets the DataType (see page 7427) property to ftFloat.

Read the AsFloat property to determine the value that was assigned to an output parameter, represented as a double. The value of the parameter is converted to a double value if possible.

Delphi Examples:

```
{
This example fills in the parameters of a client dataset
from the entries of a list box.
}
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
  ListItem: string;
begin
  { Call FetchParams to ensure parameters reflect server metadata }
  ClientDataSet1.FetchParams;
  with ClientDataSet1.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
              Items[I].AsBoolean := True
            else
              Items[I].AsBoolean := False;
          end;
```

```
          ftFloat:
            Items[I].AsFloat := StrToFloat(ListItem);
          ftCurrency:
            Items[I].AsCurrency := StrToFloat(ListItem);
          ftBCD:
            Items[I].AsBCD := StrToCurr(ListItem);
          ftDate:
            Items[I].AsDate := StrToDate(ListItem);
          ftTime:
            Items[I].AsTime := StrToTime(ListItem);
          ftDateTime:
            Items[I].AsDateTime := StrToDateTime(ListItem);
        end;
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Here is a string');
    ListBox1.Items.Add('4');
    ListBox1.Items.Add('3456789');
    ListBox1.Items.Add('34');
    ListBox1.Items.Add('0');
    ListBox1.Items.Add('12.56');
    ListBox1.Items.Add('4.23');
    ListBox1.Items.Add('99.5');
    ListBox1.Items.Add('4/12/53');
    ListBox1.Items.Add('7:45:00');
    ListBox1.Items.Add('4/12/53 7:45:00');

  end;
```

See Also

DataType (see page 7427)

TFloatField (see page 7229)

39.1.49.4.8 TParam.AsFMTBCD Property

Specifies the value of the parameter when it represents a binary-coded decimal (BCD) field with more than 4 decimal places or 20 significant digits

Pascal

```
property AsFMTBCD: TBcd;
```

C++

```
__property TBcd AsFMTBCD;
```

Description

Set AsFMTBCD to assign the value for a BCD field to the parameter when the BCD field is represented by a TFMTBCDField (see page 7242) object. TFMTBCDField (see page 7242) objects and the AsFMTBCD property use a true BCD type to represent the value of BCD fields. This is more accurate but slower than the Currency type used by TBCDField (see page 6776) and the AsBCD (see page 7407) property. Setting AsFMTBCD sets the DataType (see page 7427) property to ftFMTBCD.

Read the AsFMTBCD property to determine the value that was assigned to an output parameter, represented as a TBcd type. The value of the parameter is converted to a TBcd value if possible.

Tip: If the BCD field uses less than 4 decimal places or 20 significant digits, you can achieve better performance by using a TBCDField (see page 6776) object and the AsBCD (see page 7407) property instead.

See Also

DataType (see page 7427)

TBCDField (see page 6776)

AsBCD (see page 7407)

TFMTBCDField (see page 7242)

39.1.49.4.9 TParam.AsInteger Property

Specifies the value of the parameter when it represents an integer field.

Pascal

```
property AsInteger: LongInt;
```

C++

```
__property LongInt AsInteger;
```

Description

Set AsInteger to assign the value for an integer field to the parameter. Setting AsInteger sets the DataType (see page 7427) property to ftInteger.

Read the AsInteger property to determine the value that was assigned to an output parameter, represented as a 32-bit integer. The value of the parameter is converted to a 32-bit integer if possible.

C++ Examples:

```
/*
This example requires a TClientDataSet and a button on a form.
Put swSystem in the uses clause for gsAppPath.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
for (int I = 0; I < CDS->Params->Count; I++)
{
  if (CDS->Params->Items[I]->IsNull &&
      CDS->Params->Items[I]->DataType == ftInteger)
    CDS->Params->Items[I]->AsInteger = -1;
}}
```

```
#include <swSystem.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CDS->LoadFromFile(gsAppPath + "../CDS.XML");
  CDS->Params->CreateParam(ftInteger, "StateParam", ptInput);
  CDS->Params->CreateParam(ftInteger, "MyParam", ptInput);
}
```

Delphi Examples:

```
{
This example fills in the parameters of a client dataset
from the entries of a list box.
}
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
  ListItem: string;
begin
  { Call FetchParams to ensure parameters reflect server metadata }
```

```
    ClientDataSet1.FetchParams;
    with ClientDataSet1.Params do
    begin
      for I := 0 to Count - 1 do
      begin
        ListItem := ListBox1.Items[I];
        case Items[I].DataType of
          ftString:
            Items[I].AsString := ListItem;
          ftSmallInt:
            Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
          ftInteger:
            Items[I].AsInteger := StrToIntDef(ListItem, 0);
          ftWord:
            Items[I].AsWord := StrToIntDef(ListItem, 0);
          ftBoolean:
            begin
              if ListItem = 'True' then
                Items[I].AsBoolean := True
              else
                Items[I].AsBoolean := False;
            end;
          ftFloat:
            Items[I].AsFloat := StrToFloat(ListItem);
          ftCurrency:
            Items[I].AsCurrency := StrToFloat(ListItem);
          ftBCD:
            Items[I].AsBCD := StrToCurr(ListItem);
          ftDate:
            Items[I].AsDate := StrToDate(ListItem);
          ftTime:
            Items[I].AsTime := StrToTime(ListItem);
          ftDateTime:
            Items[I].AsDateTime := StrToDateTime(ListItem);
        end;
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Here is a string');
    ListBox1.Items.Add('4');
    ListBox1.Items.Add('3456789');
    ListBox1.Items.Add('34');
    ListBox1.Items.Add('0');
    ListBox1.Items.Add('12.56');
    ListBox1.Items.Add('4.23');
    ListBox1.Items.Add('99.5');
    ListBox1.Items.Add('4/12/53');
    ListBox1.Items.Add('7:45:00');
    ListBox1.Items.Add('4/12/53 7:45:00');

  end;
  {
  This example requires a TClientDataSet and a button on a form.
  Put swSystem in the uses clause for gsAppPath.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    I : Integer;
  begin
  { Assign -1 to any integer parameter which does not have a value. }
  for I := 0 to CDS.Params.Count - 1 do
    if (CDS.Params.Items[I].IsNull) and
       (CDS.Params.Items[I].DataType = ftInteger) then
```

```
    { Items is the default property, so you can omit its name }
    CDS.Params[I].AsInteger := -1;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
  CDS.Params.CreateParam(ftInteger, 'StateParam', ptInput);
  CDS.Params.CreateParam(ftInteger, 'MyParam', ptInput);
end;
```

See Also

DataType (see page 7427)

TIntegerField (see page 7301)

39.1.49.4.10 TParam.AsMemo Property

Specifies the value of the parameter when it represents a memo field.

Pascal

```
property AsMemo: string;
```

C++

```
__property AnsiString AsMemo;
```

Description

Set AsMemo to assign the value for a memo field to the parameter. Setting AsMemo sets the DataType (see page 7427) property to ftMemo.

A literal string or a null-terminated string may be directly assigned to the parameter for a memo column using the AsMemo property. For example:

```
SQLQuery1.Params[0].AsMemo := 'Some text in a long Pascal String';
SQLQuery2->ParamByName("Notes")->AsString = "Some string value";
```

It is seldom necessary to read AsMemo, because memo fields can not be used as output parameters.

Note: Some servers support string field types that are too long to fit in the record buffer. These fields are represented as memo fields, although they are not actually Blob fields. When working with such "pseudo-Blob" fields, use the AsString (see page 7421) property instead.

See Also

AsString (see page 7421)

DataType (see page 7427)

TMemoField (see page 7348)

39.1.49.4.11 TParam.AsSmallInt Property

Specifies the value of the parameter when it represents a small integer field.

Pascal

```
property AsSmallInt: LongInt;
```

C++

```
__property LongInt AsSmallInt;
```

Description

Set AsSmallInt to assign the value for a small integer field to the parameter. Setting AsSmallInt sets the DataType (see page 7427) property to ftSmallint.

Read the AsSmallInt property to determine the value that was assigned to an output parameter, represented as a 32-bit integer. The value of the parameter is converted to a LongInt value if possible.

Note: AsSmallInt works with a 32-bit integer. Small integer field objects store and manipulate their data as 32-bit integers even though the values in the underlying database are 16-bit integers.

Delphi Examples:

```
{
This example fills in the parameters of a client dataset
from the entries of a list box.
}
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
  ListItem: string;
begin
  { Call FetchParams to ensure parameters reflect server metadata }
  ClientDataSet1.FetchParams;
  with ClientDataSet1.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
              Items[I].AsBoolean := True
            else
              Items[I].AsBoolean := False;
          end;
        ftFloat:
          Items[I].AsFloat := StrToFloat(ListItem);
        ftCurrency:
          Items[I].AsCurrency := StrToFloat(ListItem);
        ftBCD:
          Items[I].AsBCD := StrToCurr(ListItem);
        ftDate:
          Items[I].AsDate := StrToDate(ListItem);
        ftTime:
          Items[I].AsTime := StrToTime(ListItem);
        ftDateTime:
          Items[I].AsDateTime := StrToDateTime(ListItem);
      end;
    end;
  end;
```

```
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Here is a string');
  ListBox1.Items.Add('4');
  ListBox1.Items.Add('3456789');
  ListBox1.Items.Add('34');
  ListBox1.Items.Add('0');
  ListBox1.Items.Add('12.56');
  ListBox1.Items.Add('4.23');
  ListBox1.Items.Add('99.5');
  ListBox1.Items.Add('4/12/53');
  ListBox1.Items.Add('7:45:00');
  ListBox1.Items.Add('4/12/53 7:45:00');

end;
```

See Also

DataType (see page 7427)

TSmallIntField

39.1.49.4.12 TParam.AsSQLTimeStamp Property

Specifies the value of the parameter when it represents an SQL timestamp field.

Pascal

```
property AsSQLTimeStamp: TSQLTimeStamp;
```

C++

```
__property TSQLTimeStamp AsSQLTimeStamp;
```

Description

Set AsSQLTimeStamp to assign the value for an SQL timestamp field to the parameter. Setting AsSQLTimeStamp sets the DataType (see page 7427) property to ftTimeStamp.

Read the AsSQLTimeStamp property to determine the value that was assigned to an output parameter, represented as an SQLTimeStamp record (Delphi) or structure (C++). The value of the parameter is converted to an SQLTimeStamp record or structure if possible.

See Also

DataType (see page 7427)

TSQLTimeStampField (see page 7469)

39.1.49.4.13 TParam.AsString Property

Specifies the value of the parameter when it represents a string field.

Pascal

```
property AsString: string;
```

C++

```
__property AnsiString AsString;
```

Description

Set AsString to assign the value for a string field to the parameter. Setting AsString sets the DataType (see page 7427)

property to ftString.

Read the AsString property to determine the value that was assigned to an output parameter, represented as a string. The value of the parameter is converted to a string.

Note: Some servers support string field types that are too long to fit in the record buffer. These fields are represented as memo fields, although they are not actually Blob fields. When working with such "pseudo-Blob" fields, use the AsString property rather than the AsMemo (see page 7419) property.

Delphi Examples:

```
{
This example fills in the parameters of a client dataset
from the entries of a list box.
}
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
  ListItem: string;
begin
  { Call FetchParams to ensure parameters reflect server metadata }
  ClientDataSet1.FetchParams;
  with ClientDataSet1.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
              Items[I].AsBoolean := True
            else
              Items[I].AsBoolean := False;
          end;
        ftFloat:
          Items[I].AsFloat := StrToFloat(ListItem);
        ftCurrency:
          Items[I].AsCurrency := StrToFloat(ListItem);
        ftBCD:
          Items[I].AsBCD := StrToCurr(ListItem);
        ftDate:
          Items[I].AsDate := StrToDate(ListItem);
        ftTime:
          Items[I].AsTime := StrToTime(ListItem);
        ftDateTime:
          Items[I].AsDateTime := StrToDateTime(ListItem);
      end;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Here is a string');
  ListBox1.Items.Add('4');
```

```
    ListBox1.Items.Add('3456789');
    ListBox1.Items.Add('34');
    ListBox1.Items.Add('0');
    ListBox1.Items.Add('12.56');
    ListBox1.Items.Add('4.23');
    ListBox1.Items.Add('99.5');
    ListBox1.Items.Add('4/12/53');
    ListBox1.Items.Add('7:45:00');
    ListBox1.Items.Add('4/12/53 7:45:00');

  end;
```

See Also

AsMemo (see page 7419)

DataType (see page 7427)

TStringField (see page 7483)

39.1.49.4.14 TParam.AsTime Property

Specifies the value of the parameter when it represents a time field.

Pascal

```
property AsTime: TDateTime;
```

C++

```
__property TDateTime AsTime;
```

Description

Set AsTime to assign the value for a time field to the parameter. Setting AsTime sets the DataType (see page 7427) property to ftTime.

Read the AsTime property to determine the value that was assigned to the parameter, represented as a DateTime. The value of the parameter is converted to a DateTime value if possible. If the DataType (see page 7427) for the parameter is ftTime, only the time portion of the TDateTime value is valid.

Note: It is not possible to store a NULL value directly to a parameter using AsTime. Use the TParam (see page 7386) .

:: Clear (see page 7394) method instead.

Delphi Examples:

```
{
This example fills in the parameters of a client dataset
from the entries of a list box.
}
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
  ListItem: string;
begin
  { Call FetchParams to ensure parameters reflect server metadata }
  ClientDataSet1.FetchParams;
  with ClientDataSet1.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
```

```
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
              Items[I].AsBoolean := True
            else
              Items[I].AsBoolean := False;
          end;
        ftFloat:
          Items[I].AsFloat := StrToFloat(ListItem);
        ftCurrency:
          Items[I].AsCurrency := StrToFloat(ListItem);
        ftBCD:
          Items[I].AsBCD := StrToCurr(ListItem);
        ftDate:
          Items[I].AsDate := StrToDate(ListItem);
        ftTime:
          Items[I].AsTime := StrToTime(ListItem);
        ftDateTime:
          Items[I].AsDateTime := StrToDateTime(ListItem);
      end;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Here is a string');
  ListBox1.Items.Add('4');
  ListBox1.Items.Add('3456789');
  ListBox1.Items.Add('34');
  ListBox1.Items.Add('0');
  ListBox1.Items.Add('12.56');
  ListBox1.Items.Add('4.23');
  ListBox1.Items.Add('99.5');
  ListBox1.Items.Add('4/12/53');
  ListBox1.Items.Add('7:45:00');
  ListBox1.Items.Add('4/12/53 7:45:00');

end;
```

See Also

AsDateTime (see page 7413)

DataType (see page 7427)

TTimeField (see page 7499)

Clear (see page 7394)

39.1.49.4.15 TParam.AsWideString Property

Pascal

```
property AsWideString: WideString;
```

C++

```
__property BSTR AsWideString;
```

Description

This is AsWideString, a member of class TParam.

39.1.49.4.16 TParam.AsWord Property

Specifies the value of the parameter when it represents a word field.

Pascal

```
property AsWord: LongInt;
```

C++

```
__property LongInt AsWord;
```

Description

Set AsWord to assign the value for a word field to the parameter. Setting AsWord sets the DataType (see page 7427) property to ftWord.

Read the AsWord property to determine the value that was assigned to the parameter, represented as a 32-bit integer. The value of the parameter is converted to a LongInt value if possible.

Note: AsWord works with a 32-bit integer. Word field objects store and manipulate their data as 32-bit integers even though the values in the underlying database are unsigned 16-bit values.

Delphi Examples:

```
{
This example fills in the parameters of a client dataset
from the entries of a list box.
}
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
  ListItem: string;
begin
  { Call FetchParams to ensure parameters reflect server metadata }
  ClientDataSet1.FetchParams;
  with ClientDataSet1.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
              Items[I].AsBoolean := True
            else
              Items[I].AsBoolean := False;
          end;
        ftFloat:
          Items[I].AsFloat := StrToFloat(ListItem);
```

```
          ftCurrency:
            Items[I].AsCurrency := StrToFloat(ListItem);
          ftBCD:
            Items[I].AsBCD := StrToCurr(ListItem);
          ftDate:
            Items[I].AsDate := StrToDate(ListItem);
          ftTime:
            Items[I].AsTime := StrToTime(ListItem);
          ftDateTime:
            Items[I].AsDateTime := StrToDateTime(ListItem);
        end;
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Here is a string');
    ListBox1.Items.Add('4');
    ListBox1.Items.Add('3456789');
    ListBox1.Items.Add('34');
    ListBox1.Items.Add('0');
    ListBox1.Items.Add('12.56');
    ListBox1.Items.Add('4.23');
    ListBox1.Items.Add('99.5');
    ListBox1.Items.Add('4/12/53');
    ListBox1.Items.Add('7:45:00');
    ListBox1.Items.Add('4/12/53 7:45:00');

  end;
```

See Also

DataType (see page 7427)

TWordField (see page 7560)

39.1.49.4.17 TParam.Bound Property

Indicates whether a value (NULL or otherwise) has been assigned to the parameter.

Pascal

```
property Bound: Boolean;
```

C++

```
__property Boolean Bound;
```

Description

Read Bound to determine whether a value has been assigned to the parameter. Whenever a value is assigned to the TParam (see page 7386) object, Bound is automatically set to true. Set Bound to false to undo the setting of a value. The Clear (see page 7394) method replaces the value of the parameter with NULL, but does not set Bound to false. However, if the Clear (see page 7394) method is used to bind the parameter to a NULL value, Bound must be separately set to true.

Datasets that represent queries and stored procedures use the value of Bound to determine whether to assign a default value for the parameter. If Bound is false, datasets that represent queries attempt to assign a value from the dataset indicated by their DataSource property. Similarly, when Bound is false, datasets that represent stored procedures attempt to supply a value directly from the server.

See Also

Clear (see page 7394)

IsNull (see page 7429)

TQuery (see page 10059)

TStoredProc (see page 10139)

39.1.49.4.18 TParam.DataSet Property

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

This is DataSet, a member of class TParam.

39.1.49.4.19 TParam.DataType Property

Indicates the type of field whose value the parameter represents.

Pascal

```
property DataType: TFieldType;
```

C++

```
__property TFieldType DataType;
```

Description

DataType is set automatically when a value is assigned to the parameter. Do not set DataType for bound fields, as that may cause the assigned value to be misinterpreted.

Read DataType to discover the type of data that was assigned to the parameter. Each possible value of DataType corresponds to a type of database field.

C++ Examples:

```
/*
This example requires a TClientDataSet and a button on a form.
Put swSystem in the uses clause for gsAppPath.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
for (int I = 0; I < CDS->Params->Count; I++)
{
  if (CDS->Params->Items[I]->IsNull &&
      CDS->Params->Items[I]->DataType == ftInteger)
    CDS->Params->Items[I]->AsInteger = -1;
}}

#include <swSystem.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CDS->LoadFromFile(gsAppPath + "../CDS.XML");
  CDS->Params->CreateParam(ftInteger, "StateParam", ptInput);
  CDS->Params->CreateParam(ftInteger, "MyParam", ptInput);
}
```

Delphi Examples:

```
{
This example fills in the parameters of a client dataset
```

```
from the entries of a list box.
}
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
  ListItem: string;
begin
  { Call FetchParams to ensure parameters reflect server metadata }
  ClientDataSet1.FetchParams;
  with ClientDataSet1.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
              Items[I].AsBoolean := True
            else
              Items[I].AsBoolean := False;
          end;
        ftFloat:
          Items[I].AsFloat := StrToFloat(ListItem);
        ftCurrency:
          Items[I].AsCurrency := StrToFloat(ListItem);
        ftBCD:
          Items[I].AsBCD := StrToCurr(ListItem);
        ftDate:
          Items[I].AsDate := StrToDate(ListItem);
        ftTime:
          Items[I].AsTime := StrToTime(ListItem);
        ftDateTime:
          Items[I].AsDateTime := StrToDateTime(ListItem);
      end;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Here is a string');
  ListBox1.Items.Add('4');
  ListBox1.Items.Add('3456789');
  ListBox1.Items.Add('34');
  ListBox1.Items.Add('0');
  ListBox1.Items.Add('12.56');
  ListBox1.Items.Add('4.23');
  ListBox1.Items.Add('99.5');
  ListBox1.Items.Add('4/12/53');
  ListBox1.Items.Add('7:45:00');
  ListBox1.Items.Add('4/12/53 7:45:00');

end;
{
This example requires a TClientDataSet and a button on a form.
Put swSystem in the uses clause for gsAppPath.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I : Integer;
begin
{ Assign -1 to any integer parameter which does not have a value. }
for I := 0 to CDS.Params.Count - 1 do
  if (CDS.Params.Items[I].IsNull) and
     (CDS.Params.Items[I].DataType = ftInteger) then
    { Items is the default property, so you can omit its name }
    CDS.Params[I].AsInteger := -1;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
  CDS.Params.CreateParam(ftInteger, 'StateParam', ptInput);
  CDS.Params.CreateParam(ftInteger, 'MyParam', ptInput);
end;
```

See Also

Working with Stored Procedure Parameters

Supplying Parameters at Design Time

39.1.49.4.20 TParam.IsNull Property

Indicates whether the value assigned to the parameter is NULL (blank).

Pascal

```
property IsNull: Boolean;
```

C++

```
__property Boolean IsNull;
```

Description

Inspect IsNull to discover if the value of the parameter is NULL, indicating the value of a blank field. NULL values can arise in the following ways:

Assigning the value of another, NULL, parameter.

Assigning the value of a blank field.

Calling the Clear (see page 7394) method.

Note: NULL parameters are not the same as unbound parameters. Unbound parameters have not had a value assigned. NULL parameters have a NULL value. NULL parameters may be bound or unbound.

C++ Examples:

```
/*
This example requires a TClientDataSet and a button on a form.
Put swSystem in the uses clause for gsAppPath.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
for (int I = 0; I < CDS->Params->Count; I++)
{
  if (CDS->Params->Items[I]->IsNull &&
      CDS->Params->Items[I]->DataType == ftInteger)
    CDS->Params->Items[I]->AsInteger = -1;
}}
```

```
#include <swSystem.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CDS->LoadFromFile(gsAppPath + "../CDS.XML");
  CDS->Params->CreateParam(ftInteger, "StateParam", ptInput);
  CDS->Params->CreateParam(ftInteger, "MyParam", ptInput);
}
```

Delphi Examples:

```
{
This example requires a TClientDataSet and a button on a form.
Put swSystem in the uses clause for gsAppPath.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I : Integer;
begin
{ Assign -1 to any integer parameter which does not have a value. }
for I := 0 to CDS.Params.Count - 1 do
  if (CDS.Params.Items[I].IsNull) and
     (CDS.Params.Items[I].DataType = ftInteger) then
    { Items is the default property, so you can omit its name }
    CDS.Params[I].AsInteger := -1;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
  CDS.Params.CreateParam(ftInteger, 'StateParam', ptInput);
  CDS.Params.CreateParam(ftInteger, 'MyParam', ptInput);
end;
```

See Also

Assign (see page 7391)

Bound (see page 7426)

Clear (see page 7394)

39.1.49.4.21 TParam.Name Property

Specifies the name of the parameter.

Pascal

```
property Name: Widestring;
```

C++

```
__property BSTR Name;
```

Description

Use Name to identify a particular parameter within a TParams (see page 7434) object. Name is the name of the parameter, not the name of the field whose value the parameter represents.

See Also

ParamByName (see page 7447)

Working with Stored Procedure Parameters

39.1.49.4.22 TParam.NativeStr Property

Indicates the value of the parameter as a string, using the character set of the database table.

Pascal

```
property NativeStr: string;
```

C++

```
__property AnsiString NativeStr;
```

Description

Use NativeStr to obtain a string version of the parameter value that uses the character set (code page) associated with the underlying database table.

See Also

Value (see page 7433)

AnsiToNative (see page 10244)

Translate (see page 7001)

39.1.49.4.23 TParam.NumericScale Property

Specifies the number of decimal places for the parameter.

Pascal

```
property NumericScale: Integer;
```

C++

```
__property int NumericScale;
```

Description

Set NumericScale to indicate the maximum number of digits that can appear to the right of the decimal place in a parameter's value (the scale).

Note: NumericScale only applies to parameters used by dbExpress datasets.

Note: The Precision (see page 7432) property controls the total number of digits allowed.

See Also

Precision (see page 7432)

Working with Stored Procedure Parameters

39.1.49.4.24 TParam.ParamType Property

Indicates the type of the parameter the TParam (see page 7386) represents.

Pascal

```
property ParamType: TParamType;
```

C++

```
__property TParamType ParamType;
```

Description

Objects that use TParam (see page 7386) objects to represent field parameters set ParamType to indicate how the parameter is used. ParamType must be one of the following values:

Value (see page 7433)	Description
ptUnknown	Unknown or undetermined. Before executing a stored procedure, the application must set parameters of this type to another kind.
ptInput	Used to input a field value. Identifies a parameter used to pass values to a query or stored procedure for processing.
ptOutput	Used to output a field value. Identifies a parameter used by a stored procedure to return values to an application.
ptInputOutput	Used for both input and output.
ptResult	Used as a return value. Identifies a parameter used by a stored procedure to return an error or status value. A stored procedure can only have one parameter of type ptResult.

See Also

Working with Stored Procedure Parameters

Supplying Parameters at Design Time

39.1.49.4.25 TParam.Precision Property

Specifies the number of digits allowed for a numeric parameter.

Pascal

```
property Precision: Integer;
```

C++

```
__property int Precision;
```

Description

Set Precision to indicate the maximum number of digits allowed in the parameter, including both significant and fractional.

Note: Precision only applies to parameters used by dbExpress datasets.

Note: The NumericScale (see page 7431) property controls the total number of decimal digits allowed.

See Also

NumericScale (see page 7431)

Size (see page 7432)

Working with Stored Procedure Parameters

39.1.49.4.26 TParam.Size Property

Specifies the number of characters in a string-type parameter.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

Set Size to indicate the maximum number of characters allowed in the parameter. Size only pertains to string-type parameters. For numeric type parameters, use the NumericScale (see page 7431) and Precision (see page 7432) properties. For other parameter types, Size should be zero.

Note: Size only applies to parameters used by dbExpress datasets.

See Also

Precision (see page 7432)

NumericScale (see page 7431)

Working with Stored Procedure Parameters

39.1.49.4.27 TParam.Text Property

Represents the value of the parameter as a string.

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

Set the Text property to assign the value of the parameter to a string without changing the DataType (see page 7427). Unlike the AsString (see page 7421) property, which sets the value to a string and changes the DataType (see page 7427), setting the text property converts the string to the DataType (see page 7427) of the parameter, and sets the value accordingly. Thus, use AsString (see page 7421) to treat the parameter as representing the value of a string field. Use Text instead when assigning a value that is in string form, when making no assumptions about the field type. For example, Text is useful for assigning user data that was input using an edit control.

Reading the Text property is the same as reading the AsString (see page 7421) property.

See Also

AsString (see page 7421)

DataType (see page 7427)

TStringField (see page 7483)

39.1.49.4.28 TParam.Value Property

Represents the value of the parameter as a Variant.

Pascal

```
property Value: Variant;
```

C++

```
__property Variant Value;
```

Description

Use Value in generic code that manipulates the values of parameters without needing to know the field type the parameters represent.

See Also

DataType (see page 7427)

Text (see page 7433)

Working with Stored Procedure Parameters

Supplying Parameters at Design Time

39.1.50 TParams Class

TParams manages a list of field parameters.

Class Hierarchy

Pascal

```
TParams = class(TCollection);
```

C++

```
class TParams : public TCollection;
```

File

DB

Description

Datasets that represent queries and stored procedures use TParams objects to represent the parameters of their query or stored procedures. Client datasets use TParams to represent parameters they pass to a source dataset.

Note: ADO-based datasets do not use TParams objects to represent parameters. They use the TParameters class instead.

Use the properties and methods of TParams to:

Access a specific field parameter.

Add or delete field parameters from the list.

Get or set the values of individual field parameters.

Compare field parameters.

Iterate through all field parameters.

Copy a set of field parameters to another parameter list.

See Also

TParam (see page 7386)

TCustomClientDataSet (see page 8373)

TCustomSQLDataSet (see page 20150)

TQuery (see page 10059)

TStoredProc (see page 10139)

TXMLBroker (see page 26049)

TParameters (see page 1261)

Sending Commands to the Server

39.1.50.1 TParams Members

The following tables list the members exposed by TParams.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 7438)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 7444)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetDataSet (see page 7445)	This is GetDataSet, a member of class TParams.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 7445)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 7448)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 2736)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	AddParam (see page 7438)	Inserts a new field parameter into the Items (see page 7449) list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
	AssignValues (see page 7439)	Assigns new values to the parameters of the Items (see page 7449) property.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 7437)	This is the overview for the Create constructor overload.
	CreateParam (see page 7441)	Creates a new field parameter object and inserts it into the Items (see page 7449) list.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	FindParam (see page 7445)	Locates the parameter with a given name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParamList (see page 7446)	Fills a TList object with the TParam (see page 7386) objects specified by name.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEqual (see page 7446)	Indicates whether two TParams (see page 7434) objects have identical Items (see page 7449).
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
	ParamByName (see page 7447)	Locates the parameter with a given name.
	ParseSQL (see page 7447)	Locates the parameters in an SQL statement and replaces them with question marks.
	RemoveParam (see page 7448)	Removes the specified parameter from the Items (see page 7449) property of a TParams (see page 7434) object.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 7449)	Lists the individual field parameters of a TParams (see page 7434) object.
	ParamValues (see page 7451)	Lists the values of individual field parameters that are specified by name.

39.1.50.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TParams.Create () (see page 7437)	Creates an instance of TParams (see page 7434).
	TParams.Create (TPersistent) (see page 7437)	Creates an instance of TParams (see page 7434).

39.1.50.2.1 TParams.Create Constructor ()

Creates an instance of TParams (see page 7434).

Pascal

```
constructor Create(Owner: TPersistent); overload;
constructor Create; overload;
```

C++

```
__fastcall TParams(TPersistent * Owner);
__fastcall TParams();
```

Description

Call Create to instantiate a TParams (see page 7434) object. Most applications will not need to create a TParams (see page 7434) list, as it is created by the dataset that uses the TParams (see page 7434) object.

See Also

Params (see page 10087)

Params (see page 10165)

Params (see page 8489)

Params (see page 20197)

Params (see page 26073)

39.1.50.3 TParams Methods

The methods of the TParams class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 7438)	Copies the properties of an object to a destination object.
	DefineProperties (see page 7444)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDataSet (see page 7445)	This is GetDataSet, a member of class TParams.
	GetOwner (see page 7445)	Returns the owner of an object.
	Update (see page 7448)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	AddParam (see page 7438)	Inserts a new field parameter into the Items (see page 7449) list.

	AssignValues (see page 7439)	Assigns new values to the parameters of the Items (see page 7449) property.
	CreateParam (see page 7441)	Creates a new field parameter object and inserts it into the Items (see page 7449) list.
	FindParam (see page 7445)	Locates the parameter with a given name.
	GetParamList (see page 7446)	Fills a TList object with the TParam (see page 7386) objects specified by name.
	IsEqual (see page 7446)	Indicates whether two TParams (see page 7434) objects have identical Items (see page 7449).
	ParamByName (see page 7447)	Locates the parameter with a given name.
	ParseSQL (see page 7447)	Locates the parameters in an SQL statement and replaces them with question marks.
	RemoveParam (see page 7448)	Removes the specified parameter from the Items (see page 7449) property of a TParams (see page 7434) object.

39.1.50.3.1 TParams.AddParam Method

Inserts a new field parameter into the Items (see page 7449) list.

Pascal

```
procedure AddParam(Value: TParam);
```

C++

```
__fastcall AddParam(TParam Value);
```

Description

Use AddParam to add an instantiated parameter to the Items (see page 7449) list. To create a field parameter as well as add it to the list, use CreateParam (see page 7441) instead.

Note: Parameters for datasets that represent queries are only created by specifying parameters in the SQL statement. The TParam (see page 7386) objects in such a dataset's1` Params property are created automatically when parameter tokens are added to the SQL statement.

See Also

CreateParam (see page 7441)

RemoveParam (see page 7448)

TParam (see page 7386)

39.1.50.3.2 TParams.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign (see page 2930)

EConvertError (see page 22724)

39.1.50.3.3 TParams.AssignValues Method

Assigns new values to the parameters of the Items (see page 7449) property.

Pascal

```
procedure AssignValues(Value: TParams);
```

C++

```
__fastcall AssignValues(TParams Value);
```

Description

Use AssignValues to update the values of the field parameters in the Items (see page 7449) list to match the values of the corresponding field parameters in another TParams (see page 7434) object. For each entry in Items (see page 7449), the AssignValues method attempts to find a parameter with the same Name property in Value. If successful, the parameter information (type and current data) from the Value parameter is assigned to the Items (see page 7449) entry. Entries in Items (see page 7449) for which no match is found are left unchanged.

Delphi Examples:

```
{
  This example saves params from a ClientDataSet, alters them
  and restores them.
}
procedure TForm1.SaveParamsClick(Sender: TObject);
begin
  { Save the parameters for the TDataSet }
  SavedParams.Assign(CDS.Params);
end;

procedure TForm1.AlterParamsClick(Sender: TObject);
var
  I : Integer;
  ListItem: string;
begin
  CDS.Params.CreateParam(ftString, 'stringParam', ptInput);  // you can also edit the params
in the CDS dynamically
  CDS.Params.CreateParam(ftSmallInt, 'smallIntParam', ptInput);
  CDS.Params.CreateParam(ftInteger, 'intParam', ptInput);
  CDS.Params.CreateParam(ftWord, 'wordParam', ptInput);
  CDS.Params.CreateParam(ftBoolean, 'boolParam', ptInput);
  CDS.Params.CreateParam(ftFloat, 'floatParam', ptInput);
  CDS.Params.CreateParam(ftCurrency, 'currencyParam', ptInput);
  CDS.Params.CreateParam(ftBCD, 'bcdParam', ptInput);
  CDS.Params.CreateParam(ftDate, 'dateParam', ptInput);
```

```
    CDS.Params.CreateParam(ftTime, 'timeParam', ptInput);
    CDS.Params.CreateParam(ftDateTime, 'dateTimeParam', ptInput);
    { Call FetchParams to ensure parameters reflect server metadata }
    CDS.FetchParams;
    with CDS.Params do
    begin
      for I := 0 to Count - 1 do
      begin
        ListItem := ListBox1.Items[I];
        case Items[I].DataType of
          ftString:
            Items[I].AsString := ListItem;
          ftSmallInt:
            Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
          ftInteger:
            Items[I].AsInteger := StrToIntDef(ListItem, 0);
          ftWord:
            Items[I].AsWord := StrToIntDef(ListItem, 0);
          ftBoolean:
            begin
              if ListItem = 'True' then
                Items[I].AsBoolean := True
              else
                Items[I].AsBoolean := False;
            end;
          ftFloat:
            Items[I].AsFloat := StrToFloat(ListItem);
          ftCurrency:
            Items[I].AsCurrency := StrToFloat(ListItem);
          ftBCD:
            Items[I].AsBCD := StrToCurr(ListItem);
          ftDate:
            Items[I].AsDate := StrToDate(ListItem);
          ftTime:
            Items[I].AsTime := StrToTime(ListItem);
          ftDateTime:
            Items[I].AsDateTime := StrToDateTime(ListItem);
        end;
      end;
    end;
  end;

  procedure TForm1.RestoreParamsClick(Sender: TObject);
  begin
    { Restore the parameters to TDataSet }
    CDS.Params.AssignValues(SavedParams);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    { Initialize SavedParams }
    SavedParams := TParams.Create;
    ListBox1.Items.Add('Here is a string');
    ListBox1.Items.Add('4');
    ListBox1.Items.Add('3456789');
    ListBox1.Items.Add('34');
    ListBox1.Items.Add('0');
    ListBox1.Items.Add('12.56');
    ListBox1.Items.Add('4.23');
    ListBox1.Items.Add('99.5');
    ListBox1.Items.Add('4/12/53');
    ListBox1.Items.Add('7:45:00');
    ListBox1.Items.Add('4/12/53 7:45:00');
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
```

```
begin
  SavedParams.Free;
end;
{
  This example saves params from a ClientDataSet, alters them
  and restores them.
}
procedure TForm1.SaveParamsClick(Sender: TObject);
begin
  { Save the parameters for the TDataSet }
  SavedParams.Assign(CDS.Params);
end;

procedure TForm1.AlterParamsClick(Sender: TObject);
var
  I : Integer;
begin
  CDS.Params.CreateParam(ftInteger, 'StateParam', ptInput);
  CDS.Params.CreateParam(ftInteger, 'MyParam', ptInput);
  for I := 0 to CDS.Params.Count - 1 do
    if (CDS.Params.Items[I].IsNull) and
       (CDS.Params.Items[I].DataType = ftInteger) then
      { Items is the default property, so you can omit its name }
      CDS.Params[I].AsInteger := -1;
end;

procedure TForm1.RestoreParamsClick(Sender: TObject);
begin
  { Restore the parameters to TDataSet }
  CDS.Params.AssignValues(SavedParams);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { Initialize SavedParams }
  SavedParams := TParams.Create;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  SavedParams.Free;
end;
```

See Also

Assign (see page 2736)

Items (see page 7449)

ParamValues (see page 7451)

DataType (see page 7427)

Name (see page 7430)

Value (see page 7433)

39.1.50.3.4 TParams.CreateParam Method

Creates a new field parameter object and inserts it into the Items (see page 7449) list.

Pascal

```
function CreateParam(FldType: TFieldType; const ParamName: string; ParamType: TParamType):
TParam;
```

C++

```
__fastcall TParam CreateParam(TFieldType FldType, const AnsiString ParamName, TParamType
ParamType);
```

Description

Call CreateParam to instantiate a new TParam (see page 7386) object and add it to the Items (see page 7449) managed by this TParams (see page 7434) object. Specify the field type of the new parameter as the value of FldType, specify the name of the parameter as the value of ParamName, and specify the type of parameter as the value of ParamType.

Note: Parameters for datasets that represent queries are only created by specifying parameters in the SQL statement. The TParam (see page 7386) objects in such as dataset's Params property are created automatically when parameter tokens are added to the SQL statement.

Delphi Examples:

```
{
  This example saves params from a ClientDataSet, alters them
  and restores them.
}
procedure TForm1.SaveParamsClick(Sender: TObject);
begin
  { Save the parameters for the TDataSet }
  SavedParams.Assign(CDS.Params);
end;

procedure TForm1.AlterParamsClick(Sender: TObject);
var
  I : Integer;
  ListItem: string;
begin
  CDS.Params.CreateParam(ftString, 'stringParam', ptInput);  // you can also edit the params
in the CDS dynamically
  CDS.Params.CreateParam(ftSmallInt, 'smallIntParam', ptInput);
  CDS.Params.CreateParam(ftInteger, 'intParam', ptInput);
  CDS.Params.CreateParam(ftWord, 'wordParam', ptInput);
  CDS.Params.CreateParam(ftBoolean, 'boolParam', ptInput);
  CDS.Params.CreateParam(ftFloat, 'floatParam', ptInput);
  CDS.Params.CreateParam(ftCurrency, 'currencyParam', ptInput);
  CDS.Params.CreateParam(ftBCD, 'bcdParam', ptInput);
  CDS.Params.CreateParam(ftDate, 'dateParam', ptInput);
  CDS.Params.CreateParam(ftTime, 'timeParam', ptInput);
  CDS.Params.CreateParam(ftDateTime, 'dateTimeParam', ptInput);
  { Call FetchParams to ensure parameters reflect server metadata }
  CDS.FetchParams;
  with CDS.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
              Items[I].AsBoolean := True
```

```
            else
              Items[I].AsBoolean := False;
          end;
        ftFloat:
          Items[I].AsFloat := StrToFloat(ListItem);
        ftCurrency:
          Items[I].AsCurrency := StrToFloat(ListItem);
        ftBCD:
          Items[I].AsBCD := StrToCurr(ListItem);
        ftDate:
          Items[I].AsDate := StrToDate(ListItem);
        ftTime:
          Items[I].AsTime := StrToTime(ListItem);
        ftDateTime:
          Items[I].AsDateTime := StrToDateTime(ListItem);
      end;
    end;
  end;
end;

procedure TForm1.RestoreParamsClick(Sender: TObject);
begin
  { Restore the parameters to TDataSet }
  CDS.Params.AssignValues(SavedParams);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { Initialize SavedParams }
  SavedParams := TParams.Create;
  ListBox1.Items.Add('Here is a string');
  ListBox1.Items.Add('4');
  ListBox1.Items.Add('3456789');
  ListBox1.Items.Add('34');
  ListBox1.Items.Add('0');
  ListBox1.Items.Add('12.56');
  ListBox1.Items.Add('4.23');
  ListBox1.Items.Add('99.5');
  ListBox1.Items.Add('4/12/53');
  ListBox1.Items.Add('7:45:00');
  ListBox1.Items.Add('4/12/53 7:45:00');
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  SavedParams.Free;
end;
{
  This example saves params from a ClientDataSet, alters them
  and restores them.
}
procedure TForm1.SaveParamsClick(Sender: TObject);
begin
  { Save the parameters for the TDataSet }
  SavedParams.Assign(CDS.Params);
end;

procedure TForm1.AlterParamsClick(Sender: TObject);
var
  I : Integer;
begin
  CDS.Params.CreateParam(ftInteger, 'StateParam', ptInput);
  CDS.Params.CreateParam(ftInteger, 'MyParam', ptInput);
  for I := 0 to CDS.Params.Count - 1 do
    if (CDS.Params.Items[I].IsNull) and
       (CDS.Params.Items[I].DataType = ftInteger) then
```

```
      { Items is the default property, so you can omit its name }
      CDS.Params[I].AsInteger := -1;
end;

procedure TForm1.RestoreParamsClick(Sender: TObject);
begin
  { Restore the parameters to TDataSet }
  CDS.Params.AssignValues(SavedParams);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { Initialize SavedParams }
  SavedParams := TParams.Create;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  SavedParams.Free;
end;
```

See Also

Items (see page 7449)

DataType (see page 7427)

Name (see page 7430)

ParamType (see page 7431)

Passing Parameters to the Source Dataset

39.1.50.3.5 TParams.DefineProperties Method

Provides an interface for a method that reads and writes otherwise unpublished data.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

Descendants of TPersistent override DefineProperties to designate a method for storing the object's unpublished data to a stream such as a form file. By default, writing an object to a stream writes the values of all its published properties, and reading the object in reads those values and assigns them to the properties. Objects can also specify methods that read and write data other than published properties by overriding the DefineProperties method.

When overriding DefineProperties, consider including some or all of the following:

A call to the inherited method

Calls to the filer object's DefineProperty method

Calls to the filer object's DefineBinaryProperty method

DefineProperties is virtual, so descendant classes can override it as necessary but are not required to do so.

See Also

DefineProperties (see page 2771)

DefineBinaryProperty (see page 2837)

DefineProperty (see page 2838)

39.1.50.3.6 TParams.FindParam Method

Locates the parameter with a given name.

Pascal

```
function FindParam(const Value: WideString): TParam;
```

C++

```
__fastcall TParam FindParam(const BSTR Value);
```

Description

Use FindParam to find a parameter with the name passed in Value. If a match is found, FindParam returns the parameter. Otherwise, it returns nil (Delphi) or NULL (C++). Use this method rather than a direct reference to the Items (see page 7449) property to avoid depending on the order of the entries.

To locate more than one parameter at a time, by name, use the GetParamList (see page 7446) method instead. To get only the value of a named parameter, use the ParamValues (see page 7451) property.

See Also

Items (see page 7449)

ParamByName (see page 7447)

Name (see page 7430)

GetParamList (see page 7446)

ParamValues (see page 7451)

39.1.50.3.7 TParams.GetDataSet Method

Pascal

```
function GetDataSet: TDataSet;
```

C++

```
__fastcall TDataSet GetDataSet();
```

Description

This is GetDataSet, a member of class TParams.

39.1.50.3.8 TParams.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent,

GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

39.1.50.3.9 TParams.GetParamList Method

Fills a TList object with the TParam (see page 7386) objects specified by name.

Pascal

```
procedure GetParamList(List: TList; const ParamNames: WideString);
```

C++

```
__fastcall GetParamList(TList List, const BSTR ParamNames);
```

Description

Use GetParamList to obtain a subset of the field parameters in the Items (see page 7449) list that have been identified by name. ParamNames is a string giving the names of all field parameters of interest. Multiple parameter names are separated by semicolons (;).

GetParamList adds the field parameters to List in the order they appear in ParamNames. If ParamNames includes a name that does not match any of the field parameters in Items (see page 7449), an exception is raised.

See Also

Items (see page 7449)

ParamByName (see page 7447)

TList (see page 2868)

Name (see page 7430)

39.1.50.3.10 TParams.IsEqual Method

Indicates whether two TParams (see page 7434) objects have identical Items (see page 7449).

Pascal

```
function IsEqual(Value: TParams): Boolean;
```

C++

```
__fastcall Boolean IsEqual(TParams Value);
```

Description

IsEqual returns true if the Items (see page 7449) property of the Value parameter exactly matches the Items (see page 7449) property. To match, each Items (see page 7449) list must have the same number of field parameters, in the same order, which match on name, datatype, parameter type, and value.

See Also

Items (see page 7449)

DataType (see page 7427)

Name (see page 7430)

ParamType (see page 7431)

Value (see page 7433)

39.1.50.3.11 TParams.ParamByName Method

Locates the parameter with a given name.

Pascal

```
function ParamByName(const Value: string): TParam;
```

C++

```
__fastcall TParam ParamByName(const AnsiString Value);
```

Description

Use ParamByName to find a parameter with the name passed in Value. If a match is found, ParamByName returns the parameter. Otherwise, an exception is raised. Use this method rather than a direct reference to the Items (see page 7449) property to avoid depending on the order of the entries.

To locate a parameter by name without raising an exception if the parameter is not found, use the FindParam (see page 7445) method.

To locate more than one parameter at a time, by name, use the GetParamList (see page 7446) method instead. To get only the value of a named parameter, use the ParamValues (see page 7451) property.

See Also

GetParamList (see page 7446)

Items (see page 7449)

ParamValues (see page 7451)

Name (see page 7430)

FindParam (see page 7445)

39.1.50.3.12 TParams.ParseSQL Method

Locates the parameters in an SQL statement and replaces them with question marks.

Pascal

```
function ParseSQL(SQL: WideString; DoCreate: Boolean): WideString;
```

C++

```
__fastcall BSTR ParseSQL(BSTR SQL, Boolean DoCreate);
```

Description

Call ParseSQL to replace all the parameters in the SQL statement passed as SQL with question marks (?). ParseSQL recognizes parameters by a preceding colon (:). Double colons (::) or colons contained in quoted strings are parsed as literals, and the immediately following name is not replaced with a question mark. The converted SQL statement is returned.

If the DoCreate parameter is true, all parameters in the Items (see page 7449) property are replaced with a set of TParam (see page 7386) objects whose names are the same as the strings removed from the SQL statement.

For example, given the SQL string

```
SELECT * FROM EMPLOYEES
WHERE (ID = :EMP_ID) AND (NAME = :EMP_NAME)
```

ParseSQL returns the string

```
SELECT *
FROM EMPLOYEES
WHERE (ID = ?) AND (NAME = ?)
```

and, if DoCreate is true, the Items (see page 7449) property contains two TParam (see page 7386) objects with the names EMP_ID and EMP_NAME.

See Also

Items (see page 7449)

Name (see page 7430)

39.1.50.3.13 TParams.RemoveParam Method

Removes the specified parameter from the Items (see page 7449) property of a TParams (see page 7434) object.

Pascal

```
procedure RemoveParam(Value: TParam);
```

C++

```
__fastcall RemoveParam(TParam Value);
```

Description

Use RemoveParam to remove a field parameter from the Items (see page 7449) list. RemoveParam takes Value out of the list but does not free it.

Note: Parameters for datasets that represent queries are only created and removed by specifying parameters in the SQL statement. The TParam (see page 7386) objects in such a dataset's Params property are deleted automatically when the corresponding parameter tokens are removed from the SQL statement.

See Also

AddParam (see page 7438)

Clear (see page 2741)

CreateParam (see page 7441)

TParam (see page 7386)

39.1.50.3.14 TParams.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed (see page 2740)

Changed (see page 2758)

39.1.50.4 TParams Properties

The properties of the TParams class are listed here.

Public Properties

	Name	Description
	Items (see page 7449)	Lists the individual field parameters of a TParams (see page 7434) object.
	ParamValues (see page 7451)	Lists the values of individual field parameters that are specified by name.

39.1.50.4.1 TParams.Items Property

Lists the individual field parameters of a TParams (see page 7434) object.

Pascal

```
property Items [Index: Integer]: TParam;
```

C++

```
__property TParam Items[int Index];
```

Description

Use Items to iterate through all the field parameters. Index identifies the index in the range 0 to Count - 1. Items can reference a particular parameter by its index, but the ParamByName (see page 7447) method is preferred when working with named parameters, because it does not depend on the order of the parameters.

C++ Examples:

```
/*
This example requires a TClientDataSet and a button on a form.
Put swSystem in the uses clause for gsAppPath.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
for (int I = 0; I < CDS->Params->Count; I++)
{
  if (CDS->Params->Items[I]->IsNull &&
      CDS->Params->Items[I]->DataType == ftInteger)
    CDS->Params->Items[I]->AsInteger = -1;
}}
```

```
#include <swSystem.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  CDS->LoadFromFile(gsAppPath + "../CDS.XML");
  CDS->Params->CreateParam(ftInteger, "StateParam", ptInput);
  CDS->Params->CreateParam(ftInteger, "MyParam", ptInput);
}
```

Delphi Examples:

```
{
This example fills in the parameters of a client dataset
from the entries of a list box.
}
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
  ListItem: string;
begin
  { Call FetchParams to ensure parameters reflect server metadata }
  ClientDataSet1.FetchParams;
  with ClientDataSet1.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
              Items[I].AsBoolean := True
            else
              Items[I].AsBoolean := False;
          end;
        ftFloat:
          Items[I].AsFloat := StrToFloat(ListItem);
        ftCurrency:
          Items[I].AsCurrency := StrToFloat(ListItem);
        ftBCD:
          Items[I].AsBCD := StrToCurr(ListItem);
        ftDate:
          Items[I].AsDate := StrToDate(ListItem);
        ftTime:
          Items[I].AsTime := StrToTime(ListItem);
        ftDateTime:
          Items[I].AsDateTime := StrToDateTime(ListItem);
      end;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Here is a string');
  ListBox1.Items.Add('4');
  ListBox1.Items.Add('3456789');
  ListBox1.Items.Add('34');
  ListBox1.Items.Add('0');
  ListBox1.Items.Add('12.56');
  ListBox1.Items.Add('4.23');
  ListBox1.Items.Add('99.5');
```

```
    ListBox1.Items.Add('4/12/53');
    ListBox1.Items.Add('7:45:00');
    ListBox1.Items.Add('4/12/53 7:45:00');

  end;
  {
  This example requires a TClientDataSet and a button on a form.
  Put swSystem in the uses clause for gsAppPath.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    I : Integer;
  begin
  { Assign -1 to any integer parameter which does not have a value. }
  for I := 0 to CDS.Params.Count - 1 do
    if (CDS.Params.Items[I].IsNull) and
       (CDS.Params.Items[I].DataType = ftInteger) then
      { Items is the default property, so you can omit its name }
      CDS.Params[I].AsInteger := -1;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    CDS.LoadFromFile(gsAppPath + 'CDS.XML');
    CDS.Params.CreateParam(ftInteger, 'StateParam', ptInput);
    CDS.Params.CreateParam(ftInteger, 'MyParam', ptInput);
  end;
```

See Also

Count (see page 2749)

ParamByName (see page 7447)

operator_sb

39.1.50.4.2 TParams.ParamValues Property

Lists the values of individual field parameters that are specified by name.

Pascal

```
property ParamValues [const ParamName: string]: Variant;
```

C++

```
__property Variant ParamValues[AnsiString const ParamName];
```

Description

Use ParamValues to get or set the values of individual named field parameters. ParamName is a string containing the names of the individual field parameters of interest. If ParamValues is used to access more than one field parameter, the names are separated by semicolons (;).

Setting ParamValues sets the Value property for each parameter listed in the ParamName string. Specify the values as Variants (see page 23567), in order, in a Variant array.

Getting ParamValues retrieves a Variant array of Variants (see page 23567), each of which represents the value of one of the named parameters.

If ParamName includes a name that does not match any of the field parameters in Items (see page 7449), an exception is raised.

See Also

AssignValues (see page 7439)

GetParamList (see page 7446)

ParamByName (see page 7447)

Name (see page 7430)

Value (see page 7433)

39.1.51 TReferenceField Class

TReferenceField represents a REF field in a dataset.

Class Hierarchy

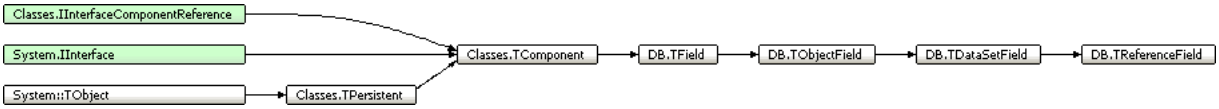

Pascal

```
TReferenceField = class(TDataSetField);
```

C++

```
class TReferenceField : public TDataSetField;
```

File

DB

Description

Reference fields store a pointer or reference to another persistent object. TReferenceField introduces properties for working with references contained in other tables and for working directly with the reference field values.

If you use the Fields (see page 7384) editor at design time to create a persistent field component for the reference field, you can access it by name at runtime. When using dynamic field components, you can access the TReferenceFieldField instance using the dataset's Fields (see page 7384) property or FieldByName method.

See Also

TDataSEtField

TDataSEt

TDataSet (see page 6894)

39.1.51.1 TReferenceField Members

The following tables list the members exposed by TReferenceField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.

	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7069)	This is Bind, a member of class TDataSetField.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CheckTypeSize (see page 7380)	This is CheckTypeSize, a member of class TObjectField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 7380)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	FreeBuffers (see page 7380)	This is FreeBuffers, a member of class TObjectField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
V	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
V	GetAsString (see page 7381)	This is GetAsString, a member of class TObjectField.
V	GetAsVariant (see page 7459)	This is GetAsVariant, a member of class TReferenceField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetCanModify (see page 7069)	This is GetCanModify, a member of class TDataSetField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
V	GetDataSize (see page 7460)	This is GetDataSize, a member of class TReferenceField.
V	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
V	GetDefaultWidth (see page 7381)	This is GetDefaultWidth, a member of class TObjectField.
	GetFieldCount (see page 7381)	This is GetFieldCount, a member of class TObjectField.
V	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
V	GetFields (see page 7069)	This is GetFields, a member of class TDataSetField.
V	GetFields (see page 7382)	This is GetFields, a member of class TObjectField.
V	GetFieldValue (see page 7382)	This is GetFieldValue, a member of class TObjectField.
V	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
V	GetHasConstraints (see page 7382)	This is GetHasConstraints, a member of class TObjectField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
V	GetOwner (see page 2778)	Returns the owner of a component.

	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 7382)	This is SetChildOrder, a member of class TObjectField.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataSet (see page 7383)	This is SetDataSet, a member of class TObjectField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetFieldKind (see page 7383)	This is SetFieldKind, a member of class TObjectField.
	SetFieldValue (see page 7383)	This is SetFieldValue, a member of class TObjectField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetParentField (see page 7383)	This is SetParentField, a member of class TObjectField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetUnNamed (see page 7383)	This is SetUnNamed, a member of class TObjectField.
	SetVarValue (see page 7460)	This is SetVarValue, a member of class TReferenceField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 7459)	Copies the Value property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 7126)	Sets the value of the field to NULL.
V	Create (see page 7458)	Creates an instance of TReferenceField (see page 7452).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 7067)	Destroys an instance of TObjectField.
V	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
V	Destroy (see page 7379)	Destroys an instance of TObjectField (see page 7372).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetChildren (see page 7381)	This is GetChildren, a member of class TObjectField.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.

R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldCount (see page 7384)	Indicates the number of child fields that comprise the object field.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	Fields (see page 7384)	Specifies the TFields (see page 7211) object that contains an array of child fields.
	FieldValues (see page 7385)	Provides access to the values for all child fields in the object field.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
R	NestedDataSet (see page 7070)	Indicates the dataset that is nested in the dataset field.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
R	UnNamed (see page 7386)	Indicates whether an object field in a nested table has a name associated with it.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7180)	Represents the data in a field component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	IncludeObjectField (see page 7069)	Indicates whether this field can be represented as a persistent field in the nested dataset. Reserved for future use.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectType (see page 7385)	Specifies the classification of the field object in Oracle.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	ReferenceTableName (see page 7460)	Names the table that contains the reference associated with the reference field.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Size (see page 7460)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.51.2 TReferenceField.Create Constructor

Creates an instance of TReferenceField (see page 7452).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TReferenceField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TReferenceField (see page 7452). Instead, the field components are created automatically, as persistent field components defined in the Fields (see page 7384) editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) to ftReference.

39.1.51.3 TReferenceField Methods

The methods of the TReferenceField class are listed here.

Protected Methods

	Name	Description
	GetAsVariant (see page 7459)	This is GetAsVariant, a member of class TReferenceField.
	GetDataSize (see page 7460)	This is GetDataSize, a member of class TReferenceField.
	SetVarValue (see page 7460)	This is SetVarValue, a member of class TReferenceField.

Public Methods

	Name	Description
	Assign (see page 7459)	Copies the Value property from another field component or assigns another object to the field component.

39.1.51.3.1 TReferenceField.Assign Method

Copies the Value property from another field component or assigns another object to the field component.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use Assign to copy the Value property from one field to another. The fields must have compatible data types.

When Source is another field object, Assign calls the AssignValue method, using the Source field's Value property. Calling Assign with the Source parameter set to nil (Delphi) or NULL (C++) assigns a blank value to the field component. Calling Assign with any other object type calls the inherited method, which allows assignment from objects that implement a TField object in their AssignTo method.

See Also

AssignValue (see page 7124)

Clear (see page 7126)

DataType (see page 7159)

AssignTo (see page 2931)

Value (see page 7180)

39.1.51.3.2 TReferenceField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TReferenceField.

39.1.51.3.3 TReferenceField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; override;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TReferenceField.

39.1.51.3.4 TReferenceField.SetVarValue Method

Pascal

```
procedure SetVarValue(const Value: Variant); override;
```

C++

```
virtual __fastcall SetVarValue(const Variant Value);
```

Description

This is SetVarValue, a member of class TReferenceField.

39.1.51.4 TReferenceField Properties

The properties of the TReferenceField class are listed here.

Published Properties

	Name	Description
	ReferenceTableName (see page 7460)	Names the table that contains the reference associated with the reference field.
	Size (see page 7460)	Indicates the size used in the definition of the physical database field for data types that support different sizes.

39.1.51.4.1 TReferenceField.ReferenceTableName Property

Names the table that contains the reference associated with the reference field.

Pascal

```
property ReferenceTableName: string;
```

C++

```
__property AnsiString ReferenceTableName;
```

Description

When ReferenceTableName is set, edits to the referenced data can be posted on the table of the reference field.

39.1.51.4.2 TReferenceField.Size Property

Indicates the size used in the definition of the physical database field for data types that support different sizes.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

The interpretation of Size depends on the data type. The meaning of Size for data types that use it is given in the following table. For all other data types, Size is not used and its value is always 0.

DataType	Meaning of Size
ftBCD	Size is the number of digits after the decimal place.
ftString	Size is the maximum number of characters in the string.
ftVarBytes	Size is the maximum number of bytes of data, not counting the first two bytes which indicate the actual number of bytes of data that were used.
ftBytes	Size is the maximum number of bytes of data.
ftBlob	Size is the number of bytes from the BLOB that are stored in the record buffer.
ftDBaseOle	Size is the number of bytes from the dBASE OLE BLOB that are stored in the record buffer.
ftFMTBCD	Size is the number of digits after the decimal place.
ftFmtMemo	Size is the number of bytes from the memo that are stored in the record buffer.
ftGraphic	Size is the number of bytes from the image that are stored in the record buffer.
ftMemo	Size is the number of characters from the memo that are stored in the record buffer.
ftParadoxOle	Size is the number of bytes from the Paradox OLE BLOB that are stored in the record buffer.
ftTypedBinary	Size is the number of bytes from the BLOB that are stored in the record buffer.
ftADT	Size is the total number of fields contained in the ADT field, including the fields of child fields that are of type ADT.
ftArray	Size is the number of elements in the array.

See Also

DataSize (see page 7157)

DataType (see page 7159)

39.1.52 TSmallintField Class

TSmallIntField represents a signed 16-bit integer field in a dataset.

Class Hierarchy

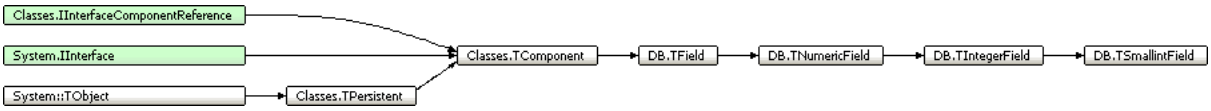

Pascal

```
TSmallintField = class(TIntegerField);
```

C++

```
class TSmallintField : public TIntegerField;
```

File

DB

Description

TSmallIntField encapsulates the fundamental behavior common to small-integer fields. Small-integer fields can hold values in the range -32768 to 32767, and are used to hold values that are signed 16-bit whole numbers.

If you use the Fields editor at design time to create a persistent field component for the small integer field, you can access it by name at runtime. When using dynamic field components, you can access the TSmallIntField instance using the dataset's Fields property or FieldByName method.

See Also

TDataSet (see page 6894)

TDataSet (see page 6894)

TDataSet (see page 6894)

TField (see page 7115)

TIntegerField (see page 7301)

TWordField (see page 7560)

Defining a Data Field

Using Default Formatting for Numeric

39.1.52.1 TSmallintField Members

The following tables list the members exposed by TSmallintField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
	GetAsFloat (see page 7310)	This is GetAsFloat, a member of class TIntegerField.
	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
	GetAsInteger (see page 7310)	This is GetAsInteger, a member of class TIntegerField.
	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
	GetAsString (see page 7311)	This is GetAsString, a member of class TIntegerField.
	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
	GetAsVariant (see page 7311)	This is GetAsVariant, a member of class TIntegerField.
	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
	GetDataSize (see page 7469)	This is GetDataSize, a member of class TSmallintField.
	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetText (see page 7311)	This is GetText, a member of class TIntegerField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 7312)	This is GetValue, a member of class TIntegerField.
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	RangeError (see page 7368)	This is RangeError, a member of class TNumericField.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.

	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsFloat (see page 7312)	This is SetAsFloat, a member of class TIntegerField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsInteger (see page 7312)	This is SetAsInteger, a member of class TIntegerField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsString (see page 7312)	This is SetAsString, a member of class TIntegerField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayFormat (see page 7368)	This is SetDisplayFormat, a member of class TNumericField.
	SetEditFormat (see page 7368)	This is SetEditFormat, a member of class TNumericField.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
	SetVarValue (see page 7312)	This is SetVarValue, a member of class TIntegerField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 7126)	Sets the value of the field to NULL.
V	Create (see page 7468)	Creates an instance of a TSmallIntField object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.

R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7180)	Represents the data in a field component.
	Value (see page 7314)	Represents the value of the integer field.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	Alignment (see page 7369)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayFormat (see page 7369)	Determines how a numeric field's value is formatted for display in a data-aware control.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	EditFormat (see page 7371)	Determines how a numeric field's value is formatted when it is being edited in a data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.

	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	MaxValue (see page 7313)	Limits the maximum value in the integer field.
	MinValue (see page 7313)	Limits the minimum value in the integer field.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.52.2 TSmallintField.Create Constructor

Creates an instance of a TSmallIntField object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSmallintField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TSmallIntField. Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftSmallInt, MinValue (see page 7313) to -32768, and MaxValue (see page 7313) to 32767. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TSmallIntField. After instantiating a TSmallIntField, associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TSmallIntField a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TSmallIntField with a dataset component by setting its DataSet (see page 7157) property.

The example below creates a TSmallIntField object for a field named Children accessed through a TSQLDataSet named SQLDataSet1.

```
var
  T: TSmallintField;
begin
  SQLDataSet1.Close;
  T := TSmallintField.Create(SQLDataSet1);
  T.FieldName := 'Children';
  T.Name := SQLDataSet1.Name + T.FieldName;
  T.Index := SQLDataSet1.FieldCount;
  T.DataSet := SQLDataSet1;
  SQLDataSet1.FieldDefs.UpDate;
  SQLDataSet1.Open;
end;
SQLDataSet1->Close();
TSmallintField *T = new TSmallintField(SQLDataSet1);
```

```
T->FieldName = "Children";
T->Name = SQLDataSet1->Name + T->FieldName;
T->Index = SQLDataSet1->FieldCount;
T->DataSet = SQLDataSet1;
SQLDataSet1->FieldDefs->UpDate();
SQLDataSet1->Open();
```

See Also

DataType (see page 7159)

MaxValue (see page 7313)

MinValue (see page 7313)

Create (see page 7121)

Owner (see page 2807)

39.1.52.3 TSmallintField Methods

The methods of the TSmallintField class are listed here.

Protected Methods

	Name	Description
	GetDataSize (see page 7469)	This is GetDataSize, a member of class TSmallintField.

39.1.52.3.1 TSmallintField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; override;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TSmallintField.

39.1.53 TSQLTimeStampField Class

TSQLTimeStampField represents a date-time field in a DBExpress dataset.

Class Hierarchy

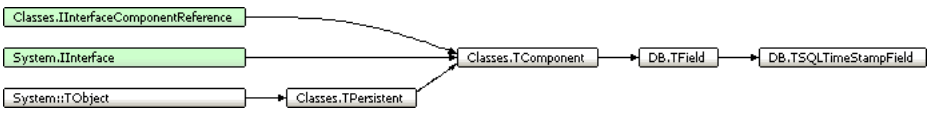

Pascal

```
TSQLTimeStampField = class(TField);
```

C++

```
class TSQLTimeStampField : public TField;
```

File

DB

Description

TSQLTimeStampField encapsulates the fundamental behavior common to Date-time fields. Date-time fields contain combined date and time information, sometimes called timestamps. TSQLTimeStampField uses the TSQLTimeStamp type to work with date-time values internally. This is the type used by DBExpress drivers for representing date and time values.

If you use the Fields editor at design time to create a persistent field component for the SQL timestamp field, you can access it by name at runtime. When using dynamic field components, you can access the TSQLTimeStampField instance using the dataset's Fields property or FieldByName method.

See Also

TTimeField (see page 7499)

TDateField (see page 7083)

TDateTimeField (see page 7090)

TDataSet (see page 6894)

TDataSet (see page 6894)

TDataSet (see page 6894)

Using Default Formatting for Numeric

Controlling and Masking User Input

39.1.53.1 TSQLTimeStampField Members

The following tables list the members exposed by TSQLTimeStampField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7477)	This is CopyData, a member of class TSQLTimeStampField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
	GetAsDateTime (see page 7477)	This is GetAsDateTime, a member of class TSQLTimeStampField.
	GetAsFloat (see page 7477)	This is GetAsFloat, a member of class TSQLTimeStampField.
	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
	GetAsSQLTimeStamp (see page 7478)	This is GetAsSQLTimeStamp, a member of class TSQLTimeStampField.
	GetAsString (see page 7478)	This is GetAsString, a member of class TSQLTimeStampField.
	GetAsVariant (see page 7478)	This is GetAsVariant, a member of class TSQLTimeStampField.
	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
	GetDataSize (see page 7478)	This is GetDataSize, a member of class TSQLTimeStampField.
	GetDefaultWidth (see page 7478)	This is GetDefaultWidth, a member of class TSQLTimeStampField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7479)	This is GetText, a member of class TSQLTimeStampField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7479)	This is SetAsDateTime, a member of class TSQLTimeStampField.
	SetAsFloat (see page 7479)	This is SetAsFloat, a member of class TSQLTimeStampField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7479)	This is SetAsSQLTimeStamp, a member of class TSQLTimeStampField.
	SetAsString (see page 7480)	This is SetAsString, a member of class TSQLTimeStampField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.

	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 7480)	This is SetVarValue, a member of class TSQLTimeStampField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7126)	Sets the value of the field to NULL.
	Create (see page 7475)	Creates and initializes a TSQLTimeStampField (see page 7469) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.

	Value (see page 7482)	Represents the value of the data in the SQL timestamp field.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayFormat (see page 7480)	Controls (see page 5711) the formatting of the SQL timestamp field for display purposes.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	EditMask (see page 7482)	Contains a mask that restricts the data that can be entered into a data field.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.53.2 TSQLTimeStampField.Create Constructor

Creates and initializes a TSQLTimeStampField (see page 7469) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSQLTimeStampField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TSQLTimeStampField (see page 7469). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftTimeStamp. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TSQLTimeStampField (see page 7469). After instantiating a TSQLTimeStampField (see page 7469), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TSQLTimeStampField (see page 7469) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TSQLTimeStampField (see page 7469) with a dataset component by setting its DataSet (see page 7157) property to the name of the dataset component. Optionally set the DisplayFormat (see page 7480) property to the desired date and time format.

The example below creates a TSQLTimeStampField (see page 7469) object for a field named CheckOut accessed through a TSQLDataSet named SQLDataSet1.

```
var
  T: TSQLTimeStampField;
begin
  SQLDataSet1.Close;
  T := TSQLTimeStampField.Create(SQLDataSet1);
  T.FieldName := 'CheckOut';
  T.Name := SQLDataSet1.Name + T.FieldName;
  T.Index := SQLDataSet1.FieldCount;
  T.DataSet := SQLDataSet1;
  T.DisplayFormat := 'mm/dd/yyyy hh:nn:ss ampm';
  SQLDataSet1.FieldDefs.UpDate;
  SQLDataSet1.Open;
end;
SQLDataSet1->Close();
TSQLTimeStampField *T = new TSQLTimeStampField(SQLDataSet1);
T->FieldName = "CheckOut";
T->Name = SQLDataSet1->Name + T->FieldName;
T->Index = SQLDataSet1->FieldCount;
T->DataSet = SQLDataSet1;
T->DisplayFormat = "mm//dd//yyyy hh:nn:ss ampm";
SQLDataSet1->FieldDefs->UpDate();
SQLDataSet1->Open();
```

See Also

DataType (see page 7159)

Create (see page 7121)

Owner (see page 2807)

39.1.53.3 TSQLTimeStampField Methods

The methods of the TSQLTimeStampField class are listed here.

Protected Methods

	Name	Description
	CopyData (see page 7477)	This is CopyData, a member of class TSQLTimeStampField.
	GetAsDateTime (see page 7477)	This is GetAsDateTime, a member of class TSQLTimeStampField.
	GetAsFloat (see page 7477)	This is GetAsFloat, a member of class TSQLTimeStampField.
	GetAsSQLTimeStamp (see page 7478)	This is GetAsSQLTimeStamp, a member of class TSQLTimeStampField.
	GetAsString (see page 7478)	This is GetAsString, a member of class TSQLTimeStampField.
	GetAsVariant (see page 7478)	This is GetAsVariant, a member of class TSQLTimeStampField.
	GetDataSize (see page 7478)	This is GetDataSize, a member of class TSQLTimeStampField.
	GetDefaultWidth (see page 7478)	This is GetDefaultWidth, a member of class TSQLTimeStampField.
	GetText (see page 7479)	This is GetText, a member of class TSQLTimeStampField.
	SetAsDateTime (see page 7479)	This is SetAsDateTime, a member of class TSQLTimeStampField.
	SetAsFloat (see page 7479)	This is SetAsFloat, a member of class TSQLTimeStampField.
	SetAsSQLTimeStamp (see page 7479)	This is SetAsSQLTimeStamp, a member of class TSQLTimeStampField.
	SetAsString (see page 7480)	This is SetAsString, a member of class TSQLTimeStampField.
	SetVarValue (see page 7480)	This is SetVarValue, a member of class TSQLTimeStampField.

39.1.53.3.1 TSQLTimeStampField.CopyData Method

Pascal

```
procedure CopyData(Source: Pointer; Dest: Pointer); override;
```

C++

```
virtual __fastcall CopyData(void * Source, void * Dest);
```

Description

This is CopyData, a member of class TSQLTimeStampField.

39.1.53.3.2 TSQLTimeStampField.GetAsDateTime Method

Pascal

```
function GetAsDateTime: TDateTime; override;
```

C++

```
virtual __fastcall TDateTime GetAsDateTime();
```

Description

This is GetAsDateTime, a member of class TSQLTimeStampField.

39.1.53.3.3 TSQLTimeStampField.GetAsFloat Method

Pascal

```
function GetAsFloat: Double; override;
```

C++

```
virtual __fastcall Double GetAsFloat();
```

Description

This is GetAsFloat, a member of class TSQLTimeStampField.

39.1.53.3.4 TSQLTimeStampField.GetAsSQLTimeStamp Method

Pascal

```
function GetAsSQLTimeStamp: TSQLTimeStamp; override;
```

C++

```
virtual __fastcall TSQLTimeStamp GetAsSQLTimeStamp();
```

Description

This is GetAsSQLTimeStamp, a member of class TSQLTimeStampField.

39.1.53.3.5 TSQLTimeStampField.GetAsString Method

Pascal

```
function GetAsString: string; override;
```

C++

```
virtual __fastcall AnsiString GetAsString();
```

Description

This is GetAsString, a member of class TSQLTimeStampField.

39.1.53.3.6 TSQLTimeStampField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TSQLTimeStampField.

39.1.53.3.7 TSQLTimeStampField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; override;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TSQLTimeStampField.

39.1.53.3.8 TSQLTimeStampField.GetDefaultWidth Method

Pascal

```
function GetDefaultWidth: Integer; override;
```

C++

```
virtual __fastcall int GetDefaultWidth();
```

Description

This is GetDefaultWidth, a member of class TSQLTimeStampField.

39.1.53.3.9 TSQLTimeStampField.GetText Method

Pascal

```
procedure GetText(var Text: string; DisplayText: Boolean); override;
```

C++

```
virtual __fastcall GetText(AnsiString Text, Boolean DisplayText);
```

Description

This is GetText, a member of class TSQLTimeStampField.

39.1.53.3.10 TSQLTimeStampField.SetAsDateTime Method

Pascal

```
procedure SetAsDateTime(Value: TDateTime); override;
```

C++

```
virtual __fastcall SetAsDateTime(TDateTime Value);
```

Description

This is SetAsDateTime, a member of class TSQLTimeStampField.

39.1.53.3.11 TSQLTimeStampField.SetAsFloat Method

Pascal

```
procedure SetAsFloat(Value: Double); override;
```

C++

```
virtual __fastcall SetAsFloat(Double Value);
```

Description

This is SetAsFloat, a member of class TSQLTimeStampField.

39.1.53.3.12 TSQLTimeStampField.SetAsSQLTimeStamp Method

Pascal

```
procedure SetAsSQLTimeStamp(const Value: TSQLTimeStamp); override;
```

C++

```
virtual __fastcall SetAsSQLTimeStamp(const TSQLTimeStamp Value);
```

Description

This is SetAsSQLTimeStamp, a member of class TSQLTimeStampField.

39.1.53.3.13 TSQLTimeStampField.SetAsString Method

Pascal

```
procedure SetAsString(const Value: string); override;
```

C++

```
virtual __fastcall SetAsString(const AnsiString Value);
```

Description

This is SetAsString, a member of class TSQLTimeStampField.

39.1.53.3.14 TSQLTimeStampField.SetVarValue Method

Pascal

```
procedure SetVarValue(const Value: Variant); override;
```

C++

```
virtual __fastcall SetVarValue(const Variant Value);
```

Description

This is SetVarValue, a member of class TSQLTimeStampField.

39.1.53.4 TSQLTimeStampField Properties

The properties of the TSQLTimeStampField class are listed here.

Public Properties

	Name	Description
	Value (see page 7482)	Represents the value of the data in the SQL timestamp field.

Published Properties

	Name	Description
	DisplayFormat (see page 7480)	Controls (see page 5711) the formatting of the SQL timestamp field for display purposes.
	EditMask (see page 7482)	Contains a mask that restricts the data that can be entered into a data field.

39.1.53.4.1 TSQLTimeStampField.DisplayFormat Property

Controls (see page 5711) the formatting of the SQL timestamp field for display purposes.

Pascal

```
property DisplayFormat: string;
```

C++

```
__property AnsiString DisplayFormat;
```

Description

Use DisplayFormat to specify the formatting string to use for formatting the value in a SQL timestamp field when the field's value appears as a string. If DisplayFormat is not assigned a string, the value is formatted according to the default date-time format specified by the Windows Control Panel.

Note: DisplayFormat does not affect the value of the AsString (see page 7151) property.

Construct a DisplayFormat string using these format specifiers:

Specifier	Displays
c	The date using the format given by the ShortDateFormat global variable, followed by the time using the format given by the LongTimeFormat global variable. The time is not displayed if the fractional part of the DateTime value is zero.
d	The day as a number without a leading zero (1-31).
dd	The day as a number with a leading zero (01-31).
ddd	The day as an abbreviation (Sun-Sat) using the strings given by the ShortDayNames global variable.
dddd	The day as a full name (Sunday-Saturday) using the strings given by the LongDayNames global variable.
ddddd	The date using the format given by the ShortDateFormat global variable.
dddddd	The date using the format given by the LongDateFormat global variable.
m	The month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mm	The month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mmm	The month as an abbreviation (Jan-Dec) using the strings given by the ShortMonthNames global variable.
mmmm	The month as a full name (January-December) using the strings given by the LongMonthNames global variable.
yy	The year as a two-digit number (00-99).
yyyy	The year as a four-digit number (0000-9999).
h	The hour without a leading zero (0-23).
hh	The hour with a leading zero (00-23).
n	The minute without a leading zero (0-59).
nn	The minute with a leading zero (00-59).
s	The second without a leading zero (0-59).
ss	The second with a leading zero (00-59).
t	The time using the format given by the ShortTimeFormat global variable.
tt	The time using the format given by the LongTimeFormat global variable.
am/pm	The time using the 12-hour clock for the preceding h or hh specifier, followed by "am" for any hour before noon, or "pm" for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	The time using the 12-hour clock for the preceding h or hh specifier, followed by "a" for any hour before noon, or "p" for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm	The time using the 12-hour clock for the preceding h or hh specifier, followed by the contents of the TimeAMString global variable for any hour before noon, or the contents of the TimePMString global variable for any hour after noon.
/	The date separator character given by the DateSeparator global variable.
:	The time separator character given by the TimeSeparator global variable.
'xx'/"xx"	Characters enclosed in single or double quotes are displayed as-is, with no formatting changes.

Format specifiers may be written in uppercase or lowercase letters; both produce the same result.

If the string given by the Format parameter is empty, the date and time value is formatted as if a c format specifier had been given.

See Also

AsString

39.1.53.4.2 TSQLTimeStampField.EditMask Property

Contains a mask that restricts the data that can be entered into a data field.

Pascal

```
property EditMask: TEditMask;
```

C++

```
__property TEditMask EditMask;
```

Description

Use EditMask to restrict the characters a user can enter. If the user attempts to enter a character that is not valid, the character is not accepted. Validation using the EditMask property is performed on a character-by-character basis. Use the OnValidate event instead to validate the entire input, just before it is posted.

Setting EditMask to an empty string removes the mask.

Note: For a description of the format required by EditMask, see the

TEditMask type.

See Also

DisplayText (see page 7162)

IsValidChar (see page 7136)

OnValidate (see page 7183)

Validate (see page 7144)

Value (see page 7180)

39.1.53.4.3 TSQLTimeStampField.Value Property

Represents the value of the data in the SQL timestamp field.

Pascal

```
property Value: TSQLTimeStamp;
```

C++

```
__property TSQLTimeStamp Value;
```

Description

Use Value to read data directly from and write data directly to a SQL timestamp field component at runtime. TSQLTimeStampField (see page 7469) components store and manipulate their data as TSQLTimeStamp values. Thus, for SQL timestamp fields, the Value property is the same as the AsSQLTimeStamp (see page 7150) property.

See Also

AsSQLTimeStamp

39.1.54 TStringField Class

TStringField represents a string field in a dataset.

Class Hierarchy

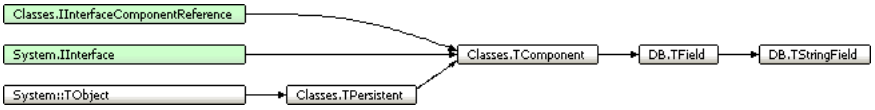

Pascal

```
TStringField = class(TField);
```

C++

```
class TStringField : public TField;
```

File

DB

Description

TStringField encapsulates the fundamental behavior common to fields that contain string data. A value of a string field is physically stored as a sequence of up to 8192 characters. However, some table types may only support string fields of smaller dimensions. Common uses for string fields are to store text, such as names and addresses.

If you use the Fields editor at design time to create a persistent field component for the string field, you can access it by name at runtime. When using dynamic field components, you can access the TStringField instance using the dataset's Fields property or FieldByName method.

Delphi Examples:

```
{
This example requires two ClientDataSets, two DBGrids, two
DataSources and two DBNavigators.  Name the two
ClientDataSets CDS and CDS2. Set the DataSet property in the
first DataSource to CDS and name that DataSource DS.  Set the
DataSet property in the second DataSource to CDS2 and name
that DataSource DS2.  Set the DataSource in the first
DBNavigator to DS and the DataSource in the second DBNavigator
to DS2.  Set the DataSource in the first DBGrid to DS and the
DataSource in the second DBGrid to DS2.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
type to Integer.  The file used to load CDS must have an
attribute name of HOST with an attribute type of string and
an attribute name of PORTA with an attribute type of integer.
Add DB, DBCtrls, Grids, DBGrids, DBClient and SwSystem to the
uses clause.  "LoadFormFile" to start with the file, or
"CreateDataSet" to start from scratch.  You must "CloseDataSet"
after a "LoadFromFile" before you can do another "CreateDataSet".
}
{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.SaveToFile(GetCurrentDir + '\CDS.XML', dfXML);
{$ELSE}
  CDS.SaveToFile(gsAppPath + 'CDS.XML', dfXML);
{$ENDIF}
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  CDS.CreateDataSet;
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  CDS2.Data := CDS.Data;
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  CDS.Close;
end;
```

See Also

TDataSet (see page 6894)

TMemoField (see page 7348)

TDataSet (see page 6894)

TDataSet (see page 6894)

Defining a Data Field

Setting Display and Edit Properties at Design Time

Defining a Lookup Field

Defining a Calculated Field

Controlling and Masking User Input

39.1.54.1 TStringField Members

The following tables list the members exposed by TStringField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7491)	This is CheckTypeSize, a member of class TStringField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7492)	This is GetAsBCD, a member of class TStringField.
V	GetAsBoolean (see page 7492)	This is GetAsBoolean, a member of class TStringField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7492)	This is GetAsDateTime, a member of class TStringField.
V	GetAsFloat (see page 7492)	This is GetAsFloat, a member of class TStringField.
V	GetAsInteger (see page 7493)	This is GetAsInteger, a member of class TStringField.
V	GetAsSQLTimeStamp (see page 7493)	This is GetAsSQLTimeStamp, a member of class TStringField.
V	GetAsString (see page 7493)	This is GetAsString, a member of class TStringField.
V	GetAsVariant (see page 7493)	This is GetAsVariant, a member of class TStringField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
V	GetDataSize (see page 7493)	This is GetDataSize, a member of class TStringField.
V	GetDefaultWidth (see page 7494)	This is GetDefaultWidth, a member of class TStringField.
V	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
V	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetSize (see page 7135)	This is GetSize, a member of class TField.
V	GetText (see page 7494)	This is GetText, a member of class TStringField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 7494)	This is GetValue, a member of class TStringField.
V	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.

	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7494)	This is SetAsBCD, a member of class TStringField.
	SetAsBoolean (see page 7495)	This is SetAsBoolean, a member of class TStringField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7495)	This is SetAsDateTime, a member of class TStringField.
	SetAsFloat (see page 7495)	This is SetAsFloat, a member of class TStringField.
	SetAsInteger (see page 7495)	This is SetAsInteger, a member of class TStringField.
	SetAsSQLTimeStamp (see page 7495)	This is SetAsSQLTimeStamp, a member of class TStringField.
	SetAsString (see page 7496)	This is SetAsString, a member of class TStringField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 7496)	This is SetVarValue, a member of class TStringField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.

	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 7126)	Sets the value of the field to NULL.
V	Create (see page 7490)	Creates an instance of a TStringField (see page 7483) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.

R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7499)	Represents the data in the string field.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	EditMask (see page 7496)	Contains a mask that restricts the data that can be entered into a data field.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
	FixedChar (see page 7497)	Specifies whether the represented field is of Char type.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.

	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Size (see page 7497)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Transliterate (see page 7498)	Determines whether text should be converted to and from the character set identified by the dataset.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.54.2 TStringField.Create Constructor

Creates an instance of a TStringField (see page 7483) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TStringField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TStringField (see page 7483). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftString, Size (see page 7497) to 20, and Transliterate (see page 7498) to true. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TStringField (see page 7483). After instantiating a TStringField (see page 7483), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TStringField (see page 7483) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TStringField (see page 7483) with a dataset component by setting its DataSet (see page 7157) property.

The example below creates a TStringField (see page 7483) object for a field named LastName accessed through a TSQLDataSet named SQLDataSet1.

```
var
  T: TStringField;
begin
  SQLDataSet1.Close;
  T := TStringField.Create(SQLDataSet1);
  T.FieldName := 'LastName';
  T.Name := SQLDataSet1.Name + T.FieldName;
  T.Index := SQLDataSet1.FieldCount;
  T.DataSet := SQLDataSet1;
  SQLDataSet1.FieldDefs.UpDate;
  SQLDataSet1.Open;
end;
SQLDataSet1->Close();
TStringField *T = new TStringField(SQLDataSet1);
```

```
T->FieldName = "LastName";
T->Name = SQLDataSet1->Name + T->FieldName;
T->Index = SQLDataSet1->FieldCount;
T->DataSet = SQLDataSet1;
SQLDataSet1->FieldDefs->UpDate();
SQLDataSet1->Open();
```

See Also

Destroy (see page 7122)

DataType (see page 7159)

Size (see page 7178)

Create (see page 7121)

Transliterate (see page 7498)

Owner (see page 2807)

39.1.54.3 TStringField Methods

The methods of the TStringField class are listed here.

Protected Methods

	Name	Description
	CheckTypeSize (see page 7491)	This is CheckTypeSize, a member of class TStringField.
	GetAsBCD (see page 7492)	This is GetAsBCD, a member of class TStringField.
	GetAsBoolean (see page 7492)	This is GetAsBoolean, a member of class TStringField.
	GetAsDateTime (see page 7492)	This is GetAsDateTime, a member of class TStringField.
	GetAsFloat (see page 7492)	This is GetAsFloat, a member of class TStringField.
	GetAsInteger (see page 7493)	This is GetAsInteger, a member of class TStringField.
	GetAsSQLTimeStamp (see page 7493)	This is GetAsSQLTimeStamp, a member of class TStringField.
	GetAsString (see page 7493)	This is GetAsString, a member of class TStringField.
	GetAsVariant (see page 7493)	This is GetAsVariant, a member of class TStringField.
	GetDataSize (see page 7493)	This is GetDataSize, a member of class TStringField.
	GetDefaultWidth (see page 7494)	This is GetDefaultWidth, a member of class TStringField.
	GetText (see page 7494)	This is GetText, a member of class TStringField.
	GetValue (see page 7494)	This is GetValue, a member of class TStringField.
	SetAsBCD (see page 7494)	This is SetAsBCD, a member of class TStringField.
	SetAsBoolean (see page 7495)	This is SetAsBoolean, a member of class TStringField.
	SetAsDateTime (see page 7495)	This is SetAsDateTime, a member of class TStringField.
	SetAsFloat (see page 7495)	This is SetAsFloat, a member of class TStringField.
	SetAsInteger (see page 7495)	This is SetAsInteger, a member of class TStringField.
	SetAsSQLTimeStamp (see page 7495)	This is SetAsSQLTimeStamp, a member of class TStringField.
	SetAsString (see page 7496)	This is SetAsString, a member of class TStringField.
	SetVarValue (see page 7496)	This is SetVarValue, a member of class TStringField.

39.1.54.3.1 TStringField.CheckTypeSize Method

Pascal

```
class procedure CheckTypeSize(Value: Integer); override;
```

C++

```
virtual __fastcall CheckTypeSize(int Value);
```

Description

This is CheckTypeSize, a member of class TStringField.

39.1.54.3.2 TStringField.GetAsBCD Method

Pascal

```
function GetAsBCD: TBcd; override;
```

C++

```
virtual __fastcall TBcd GetAsBCD();
```

Description

This is GetAsBCD, a member of class TStringField.

39.1.54.3.3 TStringField.GetAsBoolean Method

Pascal

```
function GetAsBoolean: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetAsBoolean();
```

Description

This is GetAsBoolean, a member of class TStringField.

39.1.54.3.4 TStringField.GetAsDateTime Method

Pascal

```
function GetAsDateTime: TDateTime; override;
```

C++

```
virtual __fastcall TDateTime GetAsDateTime();
```

Description

This is GetAsDateTime, a member of class TStringField.

39.1.54.3.5 TStringField.GetAsFloat Method

Pascal

```
function GetAsFloat: Double; override;
```

C++

```
virtual __fastcall Double GetAsFloat();
```

Description

This is GetAsFloat, a member of class TStringField.

39.1.54.3.6 TStringField.GetAsInteger Method

Pascal

```
function GetAsInteger: Longint; override;
```

C++

```
virtual __fastcall Longint GetAsInteger();
```

Description

This is GetAsInteger, a member of class TStringField.

39.1.54.3.7 TStringField.GetAsSQLTimeStamp Method

Pascal

```
function GetAsSQLTimeStamp: TSQLTimeStamp; override;
```

C++

```
virtual __fastcall TSQLTimeStamp GetAsSQLTimeStamp();
```

Description

This is GetAsSQLTimeStamp, a member of class TStringField.

39.1.54.3.8 TStringField.GetAsString Method

Pascal

```
function GetAsString: string; override;
```

C++

```
virtual __fastcall AnsiString GetAsString();
```

Description

This is GetAsString, a member of class TStringField.

39.1.54.3.9 TStringField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TStringField.

39.1.54.3.10 TStringField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; override;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TStringField.

39.1.54.3.11 TStringField.GetDefaultWidth Method

Pascal

```
function GetDefaultWidth: Integer; override;
```

C++

```
virtual __fastcall int GetDefaultWidth();
```

Description

This is GetDefaultWidth, a member of class TStringField.

39.1.54.3.12 TStringField.GetText Method

Pascal

```
procedure GetText(var Text: string; DisplayText: Boolean); override;
```

C++

```
virtual __fastcall GetText(AnsiString Text, Boolean DisplayText);
```

Description

This is GetText, a member of class TStringField.

39.1.54.3.13 TStringField.GetValue Method

Pascal

```
function GetValue(var Value: string): Boolean;
```

C++

```
__fastcall Boolean GetValue(AnsiString Value);
```

Description

This is GetValue, a member of class TStringField.

39.1.54.3.14 TStringField.SetAsBCD Method

Pascal

```
procedure SetAsBCD(const Value: TBcd); override;
```

C++

```
virtual __fastcall SetAsBCD(const TBcd Value);
```

Description

This is SetAsBCD, a member of class TStringField.

39.1.54.3.15 TStringField.SetAsBoolean Method

Pascal

```
procedure SetAsBoolean(Value: Boolean); override;
```

C++

```
virtual __fastcall SetAsBoolean(Boolean Value);
```

Description

This is SetAsBoolean, a member of class TStringField.

39.1.54.3.16 TStringField.SetAsDateTime Method

Pascal

```
procedure SetAsDateTime(Value: TDateTime); override;
```

C++

```
virtual __fastcall SetAsDateTime(TDateTime Value);
```

Description

This is SetAsDateTime, a member of class TStringField.

39.1.54.3.17 TStringField.SetAsFloat Method

Pascal

```
procedure SetAsFloat(Value: Double); override;
```

C++

```
virtual __fastcall SetAsFloat(Double Value);
```

Description

This is SetAsFloat, a member of class TStringField.

39.1.54.3.18 TStringField.SetAsInteger Method

Pascal

```
procedure SetAsInteger(Value: Longint); override;
```

C++

```
virtual __fastcall SetAsInteger(Longint Value);
```

Description

This is SetAsInteger, a member of class TStringField.

39.1.54.3.19 TStringField.SetAsSQLTimeStamp Method

Pascal

```
procedure SetAsSQLTimeStamp(const Value: TSQLTimeStamp); override;
```

C++

```
virtual __fastcall SetAsSQLTimeStamp(const TSQLTimeStamp Value);
```

Description

This is SetAsSQLTimeStamp, a member of class TStringField.

39.1.54.3.20 TStringField.SetAsString Method

Pascal

```
procedure SetAsString(const Value: string); override;
```

C++

```
virtual __fastcall SetAsString(const AnsiString Value);
```

Description

This is SetAsString, a member of class TStringField.

39.1.54.3.21 TStringField.SetVarValue Method

Pascal

```
procedure SetVarValue(const Value: Variant); override;
```

C++

```
virtual __fastcall SetVarValue(const Variant Value);
```

Description

This is SetVarValue, a member of class TStringField.

39.1.54.4 TStringField Properties

The properties of the TStringField class are listed here.

Public Properties

	Name	Description
	Value (see page 7499)	Represents the data in the string field.

Published Properties

	Name	Description
	EditMask (see page 7496)	Contains a mask that restricts the data that can be entered into a data field.
	FixedChar (see page 7497)	Specifies whether the represented field is of Char type.
	Size (see page 7497)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Transliterate (see page 7498)	Determines whether text should be converted to and from the character set identified by the dataset.

39.1.54.4.1 TStringField.EditMask Property

Contains a mask that restricts the data that can be entered into a data field.

Pascal

```
property EditMask: TEditMask;
```

C++

```
__property TEditMask EditMask;
```

Description

Use EditMask to restrict the characters a user can enter. If the user attempts to enter a character that is not valid, the character is not accepted. Validation using the EditMask property is performed on a character-by-character basis. Use the OnValidate event instead to validate the entire input, just before it is posted.

Setting EditMask to an empty string removes the mask.

Note: For a description of the format required by EditMask, see the

TEditMask type.

See Also

DisplayText (see page 7162)

IsValidChar (see page 7136)

OnValidate (see page 7183)

Validate (see page 7144)

Value (see page 7180)

39.1.54.4.2 TStringField.FixedChar Property

Specifies whether the represented field is of Char type.

Pascal

```
property FixedChar: Boolean;
```

C++

```
__property Boolean FixedChar;
```

Description

FixedChar is true when the field in the underlying database is a CHAR type as opposed to VARCHAR.

See Also

TFieldType (see page 7612)

39.1.54.4.3 TStringField.Size Property

Indicates the size used in the definition of the physical database field for data types that support different sizes.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

The interpretation of Size depends on the data type. The meaning of Size for data types that use it is given in the following table. For all other data types, Size is not used and its value is always 0.

DataType	Meaning of Size
ftBCD	Size is the number of digits after the decimal place.
ftString	Size is the maximum number of characters in the string.

ftVarBytes	Size is the maximum number of bytes of data, not counting the first two bytes which indicate the actual number of bytes of data that were used.
ftBytes	Size is the maximum number of bytes of data.
ftBlob	Size is the number of bytes from the BLOB that are stored in the record buffer.
ftDBaseOle	Size is the number of bytes from the dBASE OLE BLOB that are stored in the record buffer.
ftFMTBCD	Size is the number of digits after the decimal place.
ftFmtMemo	Size is the number of bytes from the memo that are stored in the record buffer.
ftGraphic	Size is the number of bytes from the image that are stored in the record buffer.
ftMemo	Size is the number of characters from the memo that are stored in the record buffer.
ftParadoxOle	Size is the number of bytes from the Paradox OLE BLOB that are stored in the record buffer.
ftTypedBinary	Size is the number of bytes from the BLOB that are stored in the record buffer.
ftADT	Size is the total number of fields contained in the ADT field, including the fields of child fields that are of type ADT.
ftArray	Size is the number of elements in the array.

See Also

DataSize (see page 7157)

DataType (see page 7159)

39.1.54.4.4 TStringField.Transliterate Property

Determines whether text should be converted to and from the character set identified by the dataset.

Pascal

```
property Transliterate: Boolean;
```

C++

```
__property Boolean Transliterate;
```

Description

Use Transliterate when the physical database table identified by the dataset does not use an ANSI language driver and the data may contain extended ASCII characters uses a different character set than UTF-8 (on Linux) or ANSI (on Windows).

When Transliterate is true, string fields call the dataset's Translate method to translate UTF-8 or ANSI characters to the character set used by the dataset when the string field value is set. When the string field value is read, the Translate method translates the characters in the database table to the UTF-8 or ANSI character set.

See Also

AnsiToNative (see page 10244)

DataSet (see page 7157)

NativeToAnsi (see page 10249)

Locale (see page 9934)

Translate (see page 7001)

39.1.54.4.5 TStringField.Value Property

Represents the data in the string field.

Pascal

```
property Value: string;
```

C++

```
__property AnsiString Value;
```

Description

Use Value to read data directly from and write data directly to a string field component at runtime. Using the Value property is the same as using the AsString (see page 7151) property. Use Value when you know the field component is a string field. Use AsString (see page 7151) when working with a generic field component.

When setting the Value property, TStringField (see page 7483) can convert the assigned string from UTF-8 or ANSI to the character set used by the dataset, if the Transliterate (see page 7498) property is true. Strings that have more characters than permitted by the value of the Size (see page 7497) property are truncated when setting Value.

See Also

AsString

Size (see page 7178)

Transliterate (see page 7498)

39.1.55 TTimeField Class

TTimeField represents a time field in a dataset.

Class Hierarchy

Pascal

```
TTimeField = class(TDateTimeField);
```

C++

```
class TTimeField : public TDateTimeField;
```

File

DB

Description

TTimeField encapsulates the fundamental behavior common to fields that contain time values. Time fields contain time values (without any date), which they manage using the TDateTime type (Delphi) or class (C++).

If you use the Fields editor at design time to create a persistent field component for the time field, you can access it by name at runtime. When using dynamic field components, you can access the TTimeField instance using the dataset's Fields property or FieldByName method.

See Also

TDataSet (see page 6894)

TDateField (see page 7083)

TDateTimeField (see page 7090)

TDataSet (see page 6894)

TSQLTimeStampField (see page 7469)

TDataSet (see page 6894)

Using Default Formatting for Numeric

Controlling and Masking User Input

39.1.55.1 TTimeField Members

The following tables list the members exposed by TTimeField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7098)	This is CopyData, a member of class TDateTimeField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7098)	This is GetAsDateTime, a member of class TDateTimeField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsFloat (see page 7099)	This is GetAsFloat, a member of class TDateTimeField.
V	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
V	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.

	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
	GetAsString (see page 7099)	This is GetAsString, a member of class TDateTimeField.
	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
	GetAsVariant (see page 7099)	This is GetAsVariant, a member of class TDateTimeField.
	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
	GetDataSize (see page 7507)	This is GetDataSize, a member of class TTimeField.
	GetDefaultWidth (see page 7100)	This is GetDefaultWidth, a member of class TDateTimeField.
	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7100)	This is GetText, a member of class TDateTimeField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7100)	This is SetAsDateTime, a member of class TDateTimeField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7100)	This is SetAsFloat, a member of class TDateTimeField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7100)	This is SetAsString, a member of class TDateTimeField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 7101)	This is SetVarValue, a member of class TDateTimeField.
	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7126)	Sets the value of the field to NULL.
	Create (see page 7505)	Creates and initializes an instance of TTimeField (see page 7499).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.

	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7103)	Represents the value of the date in the date-time field.
	Value (see page 7180)	Represents the data in a field component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayFormat (see page 7101)	Controls (see page 5711) the formatting of the date-time field for display purposes.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	EditMask (see page 7102)	Contains a mask that restricts the data that can be entered into a data field.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.55.2 TTimeField.Create Constructor

Creates and initializes an instance of TTimeField (see page 7499).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TTimeField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TTimeField (see page 7499). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftTime. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TTimeField (see page 7499). After instantiating a TTimeField (see page 7499), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TTimeField (see page 7499) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TTimeField (see page 7499) with a dataset component by setting its DataSet (see page 7157) property.

The example below creates a TTimeField (see page 7499) object for a field named CheckOutTime accessed through a TQuery named Query1.

```
var
  T: TimeField;
begin
  Query1.Close;
  T := TTimeField.Create(Query1);
  T.FieldName := 'CheckOutTime';
  T.Name := Query1.Name + T.FieldName;
  T.Index := Query1.FieldCount;
  T.DataSet := Query1;
  Query1.FieldDefs.UpDate;
  Query1.Open;
end;
```

See Also

Destroy (see page 7122)

DataType (see page 7159)

Create (see page 7121)

Owner (see page 2807)

39.1.55.3 TTimeField Methods

The methods of the TTimeField class are listed here.

Protected Methods

	Name	Description
	GetDataSize (see page 7507)	This is GetDataSize, a member of class TTimeField.

39.1.55.3.1 TTimeField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; override;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TTimeField.

39.1.56 TVarBytesField Class

TVarBytesField represents a variable-length untyped binary field in a dataset.

Class Hierarchy

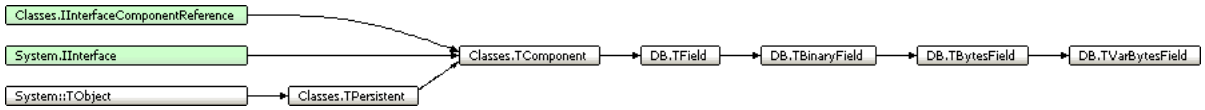

Pascal

```
TVarBytesField = class(TBytesField);
```

C++

```
class TVarBytesField : public TBytesField;
```

File

DB

Description

A variable bytes field value is a set of bytes of variable size, with the actual length of the value stored in the first two bytes.

If you use the Fields editor at design time to create a persistent field component for the variable bytes field, you can access it by name at runtime. When using dynamic field components, you can access the TVarBytesField instance using the dataset's Fields property or FieldByName method.

See Also

TDataSet (see page 6894)

Defining a Data Field

Defining a Lookup Field

Defining a Calculated Field

39.1.56.1 TVarBytesField Members

The following tables list the members exposed by TVarBytesField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 6798)	This is CheckTypeSize, a member of class TBinaryField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 6798)	This is CopyData, a member of class TBinaryField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
V	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsString (see page 6799)	This is GetAsString, a member of class TBinaryField.
V	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
V	GetAsVariant (see page 6799)	This is GetAsVariant, a member of class TBinaryField.
V	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
V	GetDataSize (see page 7514)	This is GetDataSize, a member of class TVarBytesField.
V	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
V	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
V	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetSize (see page 7135)	This is GetSize, a member of class TField.

	GetText (see page 6799)	This is GetText, a member of class TBinaryField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7514)	This is SetAsByteArray, a member of class TVarBytesField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 6799)	This is SetAsString, a member of class TBinaryField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 6799)	This is SetText, a member of class TBinaryField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 6800)	This is SetVarValue, a member of class TBinaryField.
	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.

	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7126)	Sets the value of the field to NULL.
	Create (see page 7513)	Creates an instance of TVarBytesField (see page 7507).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.

	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7180)	Represents the data in a field component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.

	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Size (see page 6800)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.56.2 TVarBytesField.Create Constructor

Creates an instance of TVarBytesField (see page 7507).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TVarBytesField(TComponent * AOwner);
```

Description

It is seldom necessary to call Create directly, because a VarBytes field object is instantiated automatically for all variable bytes fields in a dataset.

Create sets the DataType (see page 7159) to ftVarBytes and initializes the Size (see page 6800) property to 16.

See Also

Destroy (see page 7122)

DataType (see page 7159)

Size (see page 7178)

Create (see page 7121)

Owner (see page 2807)

39.1.56.3 TVarBytesField Methods

The methods of the TVarBytesField class are listed here.

Protected Methods

	Name	Description
	GetDataSize (see page 7514)	This is GetDataSize, a member of class TVarBytesField.
	SetAsByteArray (see page 7514)	This is SetAsByteArray, a member of class TVarBytesField.

39.1.56.3.1 TVarBytesField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; override;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TVarBytesField.

39.1.56.3.2 TVarBytesField.SetAsByteArray Method

Pascal

```
procedure SetAsByteArray(const Value: Variant); override;
```

C++

```
virtual __fastcall SetAsByteArray(const Variant Value);
```

Description

This is SetAsByteArray, a member of class TVarBytesField.

39.1.57 TVariantField Class

TVariantField represents a variant field in a dataset.

Class Hierarchy

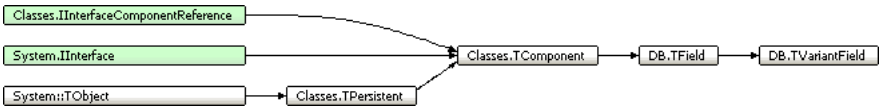

Pascal

```
TVariantField = class(TField);
```

C++

```
class TVariantField : public TField;
```

File

DB

Description

TVariantField encapsulates the fundamental behavior common to fields that contain Variant data.

If you use the Fields editor at design time to create a persistent field component for the variant field, you can access it by name at runtime. When using dynamic field components, you can access the TVariantField instance using the dataset's Fields property

or FieldByName method.

See Also

TDataSet (see page 6894)

TMemoField (see page 7348)

TDataSet (see page 6894)

TDataSet (see page 6894)

39.1.57.1 TVariantField Members

The following tables list the members exposed by TVariantField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7522)	This is CheckTypeSize, a member of class TVariantField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7522)	This is GetAsBCD, a member of class TVariantField.
V	GetAsBoolean (see page 7522)	This is GetAsBoolean, a member of class TVariantField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7522)	This is GetAsDateTime, a member of class TVariantField.
V	GetAsFloat (see page 7522)	This is GetAsFloat, a member of class TVariantField.
V	GetAsInteger (see page 7523)	This is GetAsInteger, a member of class TVariantField.
V	GetAsSQLTimeStamp (see page 7523)	This is GetAsSQLTimeStamp, a member of class TVariantField.
V	GetAsString (see page 7523)	This is GetAsString, a member of class TVariantField.
V	GetAsVariant (see page 7523)	This is GetAsVariant, a member of class TVariantField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.

	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
	GetDefaultWidth (see page 7524)	This is GetDefaultWidth, a member of class TVariantField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7524)	This is SetAsBCD, a member of class TVariantField.
	SetAsBoolean (see page 7524)	This is SetAsBoolean, a member of class TVariantField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7524)	This is SetAsDateTime, a member of class TVariantField.
	SetAsFloat (see page 7524)	This is SetAsFloat, a member of class TVariantField.
	SetAsInteger (see page 7525)	This is SetAsInteger, a member of class TVariantField.
	SetAsSQLTimeStamp (see page 7525)	This is SetAsSQLTimeStamp, a member of class TVariantField.
	SetAsString (see page 7525)	This is SetAsString, a member of class TVariantField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 7525)	This is SetVarValue, a member of class TVariantField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7126)	Sets the value of the field to NULL.
	Create (see page 7520)	Creates and initializes a variant field component.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.

R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7180)	Represents the data in a field component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.

	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.57.2 TVariantField.Create Constructor

Creates and initializes a variant field component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TVariantField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TVariantField (see page 7514). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftVariant. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TVariantField (see page 7514). After instantiating a TVariantField (see page 7514), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TVariantField (see page 7514) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TVariantField (see page 7514) with a dataset component by setting its DataSet (see page 7157) property to the name of the dataset component.

The example below creates a TInterfaceField (see page 7314) object for a field named UnknownData accessed through a TQuery named Query1.

```
var
  T: TInterfaceField;
begin
  Query1.Close;
  T := TInterfaceField.Create(Query1);
  T.FieldName := 'UnknownData';
  T.Name := Query1.Name + T.FieldName;
  T.Index := Query1.FieldCount;
  T.DataSet := Query1;
  Query1.FieldDefs.UpDate;
  Query1.Open;
end;
Query1->Close();
TInterfaceField *T = new TInterfaceField(Query1);
T->FieldName = "UnknownData";
T->Name = Query1->Name + T->FieldName;
T->Index = Query1->FieldCount;
T->DataSet = Query1;
Query1->FieldDefs->UpDate();
Query1->Open();
```

See Also

Owner (see page 2807)

Visible (see page 7181)

39.1.57.3 TVariantField Methods

The methods of the TVariantField class are listed here.

Protected Methods

	Name	Description
	CheckTypeSize (see page 7522)	This is CheckTypeSize, a member of class TVariantField.
	GetAsBCD (see page 7522)	This is GetAsBCD, a member of class TVariantField.
	GetAsBoolean (see page 7522)	This is GetAsBoolean, a member of class TVariantField.
	GetAsDateTime (see page 7522)	This is GetAsDateTime, a member of class TVariantField.
	GetAsFloat (see page 7522)	This is GetAsFloat, a member of class TVariantField.
	GetAsInteger (see page 7523)	This is GetAsInteger, a member of class TVariantField.
	GetAsSQLTimeStamp (see page 7523)	This is GetAsSQLTimeStamp, a member of class TVariantField.
	GetAsString (see page 7523)	This is GetAsString, a member of class TVariantField.
	GetAsVariant (see page 7523)	This is GetAsVariant, a member of class TVariantField.
	GetDefaultWidth (see page 7524)	This is GetDefaultWidth, a member of class TVariantField.
	SetAsBCD (see page 7524)	This is SetAsBCD, a member of class TVariantField.
	SetAsBoolean (see page 7524)	This is SetAsBoolean, a member of class TVariantField.
	SetAsDateTime (see page 7524)	This is SetAsDateTime, a member of class TVariantField.
	SetAsFloat (see page 7524)	This is SetAsFloat, a member of class TVariantField.
	SetAsInteger (see page 7525)	This is SetAsInteger, a member of class TVariantField.
	SetAsSQLTimeStamp (see page 7525)	This is SetAsSQLTimeStamp, a member of class TVariantField.
	SetAsString (see page 7525)	This is SetAsString, a member of class TVariantField.
	SetVarValue (see page 7525)	This is SetVarValue, a member of class TVariantField.

39.1.57.3.1 TVariantField.CheckTypeSize Method

Pascal

```
class procedure CheckTypeSize(Value: Integer); override;
```

C++

```
virtual __fastcall CheckTypeSize(int Value);
```

Description

This is CheckTypeSize, a member of class TVariantField.

39.1.57.3.2 TVariantField.GetAsBCD Method

Pascal

```
function GetAsBCD: TBcd; override;
```

C++

```
virtual __fastcall TBcd GetAsBCD();
```

Description

This is GetAsBCD, a member of class TVariantField.

39.1.57.3.3 TVariantField.GetAsBoolean Method

Pascal

```
function GetAsBoolean: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetAsBoolean();
```

Description

This is GetAsBoolean, a member of class TVariantField.

39.1.57.3.4 TVariantField.GetAsDateTime Method

Pascal

```
function GetAsDateTime: TDateTime; override;
```

C++

```
virtual __fastcall TDateTime GetAsDateTime();
```

Description

This is GetAsDateTime, a member of class TVariantField.

39.1.57.3.5 TVariantField.GetAsFloat Method

Pascal

```
function GetAsFloat: Double; override;
```

C++

```
virtual __fastcall Double GetAsFloat();
```

Description

This is GetAsFloat, a member of class TVariantField.

39.1.57.3.6 TVariantField.GetAsInteger Method

Pascal

```
function GetAsInteger: Longint; override;
```

C++

```
virtual __fastcall Longint GetAsInteger();
```

Description

This is GetAsInteger, a member of class TVariantField.

39.1.57.3.7 TVariantField.GetAsSQLTimeStamp Method

Pascal

```
function GetAsSQLTimeStamp: TSqlTimeStamp; override;
```

C++

```
virtual __fastcall TSqlTimeStamp GetAsSQLTimeStamp();
```

Description

This is GetAsSQLTimeStamp, a member of class TVariantField.

39.1.57.3.8 TVariantField.GetAsString Method

Pascal

```
function GetAsString: string; override;
```

C++

```
virtual __fastcall AnsiString GetAsString();
```

Description

This is GetAsString, a member of class TVariantField.

39.1.57.3.9 TVariantField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TVariantField.

39.1.57.3.10 TVariantField.GetDefaultWidth Method

Pascal

```
function GetDefaultWidth: Integer; override;
```

C++

```
virtual __fastcall int GetDefaultWidth();
```

Description

This is GetDefaultWidth, a member of class TVariantField.

39.1.57.3.11 TVariantField.SetAsBCD Method

Pascal

```
procedure SetAsBCD(const Value: TBcd); override;
```

C++

```
virtual __fastcall SetAsBCD(const TBcd Value);
```

Description

This is SetAsBCD, a member of class TVariantField.

39.1.57.3.12 TVariantField.SetAsBoolean Method

Pascal

```
procedure SetAsBoolean(Value: Boolean); override;
```

C++

```
virtual __fastcall SetAsBoolean(Boolean Value);
```

Description

This is SetAsBoolean, a member of class TVariantField.

39.1.57.3.13 TVariantField.SetAsDateTime Method

Pascal

```
procedure SetAsDateTime(Value: TDateTime); override;
```

C++

```
virtual __fastcall SetAsDateTime(TDateTime Value);
```

Description

This is SetAsDateTime, a member of class TVariantField.

39.1.57.3.14 TVariantField.SetAsFloat Method

Pascal

```
procedure SetAsFloat(Value: Double); override;
```

C++

```
virtual __fastcall SetAsFloat(Double Value);
```

Description

This is SetAsFloat, a member of class TVariantField.

39.1.57.3.15 TVariantField.SetAsInteger Method

Pascal

```
procedure SetAsInteger(Value: Longint); override;
```

C++

```
virtual __fastcall SetAsInteger(Longint Value);
```

Description

This is SetAsInteger, a member of class TVariantField.

39.1.57.3.16 TVariantField.SetAsSQLTimeStamp Method

Pascal

```
procedure SetAsSQLTimeStamp(const Value: TSqlTimeStamp); override;
```

C++

```
virtual __fastcall SetAsSQLTimeStamp(const TSqlTimeStamp Value);
```

Description

This is SetAsSQLTimeStamp, a member of class TVariantField.

39.1.57.3.17 TVariantField.SetAsString Method

Pascal

```
procedure SetAsString(const Value: string); override;
```

C++

```
virtual __fastcall SetAsString(const AnsiString Value);
```

Description

This is SetAsString, a member of class TVariantField.

39.1.57.3.18 TVariantField.SetVarValue Method

Pascal

```
procedure SetVarValue(const Value: Variant); override;
```

C++

```
virtual __fastcall SetVarValue(const Variant Value);
```

Description

This is SetVarValue, a member of class TVariantField.

39.1.58 TWideDataSet Class

Base class for all dataset components that represent Unicode data in rows and columns.

Class Hierarchy

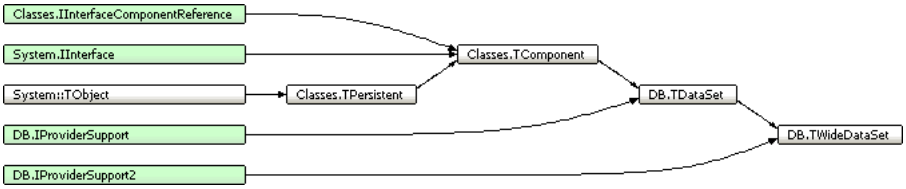

Pascal

```
TWideDataSet = class(TDataSet, IProviderSupport2);
```

C++

```
class TWideDataSet : public TDataSet, public IProviderSupport2;
```

File

DB

Description

TWideDataSet introduces the basic properties, events, and methods for working with Unicode data.

The TWideDataSet class extends TDataSet (see page 6894) to support the Unicode string type. TWideDataSet inherits the behavior of TDataSet (see page 6894) and TDataSet (see page 6894)'s IProviderSupport (see page 7576) interface, and TWideDataSet supports IProviderSupport2 (see page 7587).

BDE does not support a Unicode type, so TTable and TQuery are derived from TDataSet (see page 6894). However, TADODataSet and TSQLDataSet (dbExpress) derive from TWideDataSet, because ADO and dbExpress need to support Unicode data.

See Also

IProviderSupport2 (see page 7587)

TADODataSet (see page 1041)

TDataSet (see page 6894)

TSQLDataSet (see page 20265)

39.1.58.1 TWideDataSet Members

The following tables list the members exposed by TWideDataSet.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.

	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
V	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
V	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
V	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
V	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
V	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
V	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
V	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
V	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
V	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
V	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
V	DataEvent (see page 6924)	This is DataEvent, a member of class TDataSet.
V	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
V	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
V	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
V	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
V	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
V	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
V	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
V	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
V	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
V	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
V	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
V	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
V	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
V	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
V	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
V	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
V	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
V	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
V	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
V	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
V	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
V	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
V	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
V	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
V	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
V	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
V	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.

	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.
	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	OpenCursor (see page 6976)	This is OpenCursor, a member of class TDataSet.

	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
V	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
V	PSExecuteStatement (see page 7535)	This is the overview for the PSExecuteStatement method overload.
V	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
V	PSGetCommandText (see page 7537)	Gets the SQL command to be executed by PSExecute.
V	PSGetCommandTextW (see page 7537)	This is PSGetCommandTextW, a member of class TWideDataSet.
V	PSGetCommandType (see page 6982)	Gets the command type to be executed by PSExecute.
V	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
V	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
V	PSGetKeyFields (see page 7538)	Returns the names of all fields required to uniquely identify records in the data packet.
V	PSGetKeyFieldsW (see page 7538)	This is PSGetKeyFieldsW, a member of class TWideDataSet.
V	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
V	PSGetQuoteChar (see page 7538)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
V	PSGetQuoteCharW (see page 7539)	This is PSGetQuoteCharW, a member of class TWideDataSet.
V	PSGetTableName (see page 7539)	Returns the name of the dataset table that appears in generated SQL statements.
V	PSGetTableNameW (see page 7539)	This is PSGetTableNameW, a member of class TWideDataSet.
V	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
V	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
V	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
V	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
V	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
V	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
V	PSSetCommandText (see page 7540)	This is the overview for the PSSetCommandText method overload.
V	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.
V	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
V	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
V	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
V	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
V	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
V	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
V	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
V	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
V	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
V	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
V	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
V	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.

V	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
V	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
V	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
	Reserved (see page 7035)	For internal use only.

Public Events

	Name	Description
	AfterCancel (see page 7038)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 7040)	Occurs after an application closes a dataset.
	AfterDelete (see page 7041)	Occurs after an application deletes a record.

	AfterEdit (see page 7044)	Occurs after an application starts editing a record.
	AfterInsert (see page 7044)	Occurs after an application inserts a new record.
	AfterOpen (see page 7045)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 7046)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 7047)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 7047)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 7048)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 7048)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 7049)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 7049)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 7050)	Occurs before an application enters insert mode.
	BeforeOpen (see page 7051)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 7052)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 7052)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 7053)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 7053)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 7054)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 7054)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 7056)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 7056)	Occurs when an application attempts to modify or insert a record and an exception is raised.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
V	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Close (see page 6920)	Closes a dataset.
V	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.

	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 6904)	Creates an instance of a TDataSet (see page 6894) component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 6904)	Destroys the TDataSet (see page 6894) instance.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.

	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
V	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	Last (see page 6968)	Moves to the last record in the dataset.
V	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	Open (see page 6976)	Opens the dataset.
V	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSEndTransaction (see page 7589)	Ends a transaction that was started by a call to PSStartTransaction (see page 7597).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecute (see page 7590)	Executes the SQL command associated with the dataset.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 7591)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandType (see page 7580)	Gets the command type to be executed by PSExecute (see page 7578).
	PSGetCommandType (see page 7591)	Gets the command type to be executed by PSExecute (see page 7590).
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 7592)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 7592)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetParams (see page 7593)	Returns the current parameter values of the dataset.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSGetUpdateException (see page 7594)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 7595)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.

	PSIsSQLBased (see page 7595)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 7596)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 7596)	Resets the dataset so that it is positioned on the first record.
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 7597)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).
	PSStartTransaction (see page 7597)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7589).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 7598)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.

Public Properties

	Name	Description
	Active (see page 7004)	Specifies whether or not a dataset is open.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
R	DataSource (see page 7014)	Represents the data source of another dataset that supplies values to the dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.

	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
	FieldDefs (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	Modified (see page 7029)	Indicates whether the active record is modified.
	ObjectView (see page 7031)	Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

39.1.58.2 TWideDataSet Methods

The methods of the TWideDataSet class are listed here.

Protected Methods

	Name	Description
V	PSExecuteStatement (see page 7535)	This is the overview for the PSExecuteStatement method overload.
V	PSGetCommandText (see page 7537)	Gets the SQL command to be executed by PSExecute.
V	PSGetCommandTextW (see page 7537)	This is PSGetCommandTextW, a member of class TWideDataSet.
V	PSGetKeyFields (see page 7538)	Returns the names of all fields required to uniquely identify records in the data packet.
V	PSGetKeyFieldsW (see page 7538)	This is PSGetKeyFieldsW, a member of class TWideDataSet.
V	PSGetQuoteChar (see page 7538)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
V	PSGetQuoteCharW (see page 7539)	This is PSGetQuoteCharW, a member of class TWideDataSet.
V	PSGetTableName (see page 7539)	Returns the name of the dataset table that appears in generated SQL statements.
V	PSGetTableNameW (see page 7539)	This is PSGetTableNameW, a member of class TWideDataSet.
V	PSSetCommandText (see page 7540)	This is the overview for the PSSetCommandText method overload.

39.1.58.2.1 PSExecuteStatement Method

This is the overview for the PSExecuteStatement method overload.

Overload List

	Name	Description
	TWideDataSet.PSExecuteStatement (string, TParams, Pointer) (see page 7536)	Executes a specified SQL command.
	TWideDataSet.PSExecuteStatement (WideString, TParams, Pointer) (see page 7536)	Executes a specified SQL command.

39.1.58.2.1.1 TWideDataSet.PSExecuteStatement Method (WideString, TParams, Pointer)

Executes a specified SQL command.

Pascal

```
function PSExecuteStatement(const ASQL: WideString; AParams: TParams; ResultSet: Pointer =
nil): Integer; override; overload;
```

C++

```
virtual __fastcall int PSExecuteStatement(const BSTR ASQL, TParams AParams, void * ResultSet =
nil);
```

Description

If the value of its ResolveToDataSet property is false, a provider component calls PSExecuteStatement (indirectly) to execute SQL statements that it generates for applying updates. Usually, the dataset passes these statements on to its database server for execution.

The ASQL parameter specifies the SQL command to execute.

AParams contains any parameter values that should be applied to ASQL before execution.

ResultSet allows PSExecuteStatement to return a dataset component that contains the results of a SELECT statement. If ResultSet is not nil (Delphi) or NULL (C++), PSExecuteStatement creates a new dataset component as the target of this pointer and fills it with the result set. The caller is responsible for freeing the dataset.

PSExecuteStatement returns the number of rows affected by executing ASQL.

See Also

PSUpdateRecord (see page 7598)

applyupdates (see page 18459)

ResolveToDataSet (see page 18531)

39.1.58.2.1.2 TWideDataSet.PSExecuteStatement Method (string, TParams, Pointer)

Executes a specified SQL command.

Pascal

```
function PSExecuteStatement(const ASQL: string; AParams: TParams; ResultSet: Pointer = nil):
Integer; override; overload; deprecated;
```

C++

```
virtual __fastcall int PSExecuteStatement(const AnsiString ASQL, TParams AParams, void *
ResultSet = nil);
```

Description

If the value its ResolveToDataSet property is false, a provider component calls PSExecuteStatement (indirectly) to execute SQL

statements that it generates for applying updates. Usually, the dataset passes these statements on to its database server for execution.

The ASQL parameter specifies the SQL command to execute.

AParams contains any parameter values that should be applied to ASQL before execution.

ResultSet allows PSExecuteStatement to return a dataset component that contains the results of a SELECT statement. If ResultSet is not nil (Delphi) or NULL (C++), PSExecuteStatement creates a new dataset component as the target of this pointer and fills it with the result set. The caller is responsible for freeing the dataset.

PSExecuteStatement returns the number of rows affected by executing ASQL.

See Also

PSUpdateRecord (see page 7586)

applyupdates (see page 18459)

ResolveToDataSet (see page 18531)

39.1.58.2.2 TWideDataSet.PSGetCommandText Method

Gets the SQL command to be executed by PSExecute.

Pascal

```
function PSGetCommandText: string; override; deprecated;
```

C++

```
virtual __fastcall AnsiString PSGetCommandText();
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

PSSetCommandText (see page 7585)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

39.1.58.2.3 TWideDataSet.PSGetCommandTextW Method

Pascal

```
function PSGetCommandTextW: WideString; override;
```

C++

```
virtual __fastcall BSTR PSGetCommandTextW();
```

Description

This is PSGetCommandTextW, a member of class TWideDataSet.

39.1.58.2.4 TWideDataSet.PSGetKeyFields Method

Returns the names of all fields required to uniquely identify records in the data packet.

Pascal

```
function PSGetKeyFields: string; override; deprecated;
```

C++

```
virtual __fastcall AnsiString PSGetKeyFields();
```

Description

The provider component calls PSGetKeyFields to determine what information is required to uniquely identify records in the data packet. This information allows it to locate the records that need to change when it applies updates.

PSGetKeyFields returns a semicolon-delimited string that lists the names of the key fields of the dataset. By default (as implemented by TDataSet), these are the fields with pfInKey included in their ProviderFlags property.

See Also

PSGetDefaultOrder (see page 7580)

ProviderFlags (see page 7176)

PSGetIndexDefs (see page 7581)

39.1.58.2.5 TWideDataSet.PSGetKeyFieldsW Method

Pascal

```
function PSGetKeyFieldsW: WideString; override;
```

C++

```
virtual __fastcall BSTR PSGetKeyFieldsW();
```

Description

This is PSGetKeyFieldsW, a member of class TWideDataSet.

39.1.58.2.6 TWideDataSet.PSGetQuoteChar Method

Returns the character or characters to use in generated SQL statements to enclose quoted strings.

Pascal

```
function PSGetQuoteChar: string; override; deprecated;
```

C++

```
virtual __fastcall AnsiString PSGetQuoteChar();
```

Description

The provider component calls PSGetQuoteChar (indirectly) when generating the dynamic SQL statements that apply updates. These statements are subsequently executed by a call to PSExecuteStatement.

PSGetQuoteChar returns the character or characters that delimit quoted strings in the generated SQL. This string appears both before and after quoted strings.

See Also

applyupdates (see page 18459)

PSExecuteStatement (see page 7579)

ResolveToDataSet (see page 18531)

39.1.58.2.7 TWideDataSet.PSGetQuoteCharW Method

Pascal

```
function PSGetQuoteCharW: WideString; override;
```

C++

```
virtual __fastcall BSTR PSGetQuoteCharW();
```

Description

This is PSGetQuoteCharW, a member of class TWideDataSet.

39.1.58.2.8 TWideDataSet.PSGetTableName Method

Returns the name of the dataset table that appears in generated SQL statements.

Pascal

```
function PSGetTableName: string; override; deprecated;
```

C++

```
virtual __fastcall AnsiString PSGetTableName();
```

Description

PSGetTableName returns the name of the underlying database table against which updates are applied when the provider's ResolveToDataSet property is false. This value is used for generating the SQL statements that apply updates. The generated SQL statements are subsequently executed by a call to PSExecuteStatement.

If PSGetTableName does not return a table name, the provider must have an OnGetTableName event handler to supply this value.

See Also

applyupdates (see page 18459)

PSExecuteStatement (see page 7579)

ResolveToDataSet (see page 18531)

ongettablename (see page 18542)

39.1.58.2.9 TWideDataSet.PSGetTableNameW Method

Pascal

```
function PSGetTableNameW: WideString; override;
```

C++

```
virtual __fastcall BSTR PSGetTableNameW();
```

Description

This is PSGetTableNameW, a member of class TWideDataSet.

39.1.58.2.10 PSSetCommandText Method

This is the overview for the PSSetCommandText method overload.

Overload List

	Name	Description
	TWideDataSet.PSSetCommandText (string) (see page 7540)	Assigns a specified SQL command to be executed by PSExecute.
	TWideDataSet.PSSetCommandText (WideString) (see page 7540)	Assigns a specified SQL command to be executed by PSExecute.

39.1.58.2.10.1 TWideDataSet.PSSetCommandText Method (WideString)

Assigns a specified SQL command to be executed by PSExecute.

Pascal

```
procedure PSSetCommandText(const CommandText: WideString); override; overload;
```

C++

```
virtual __fastcall PSSetCommandText(const BSTR CommandText);
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

The value of the CommandText parameter replaces the dataset's SQL statement (if it has one), or replaces the name of an underlying database table or stored procedure.

See Also

CommandText (see page 8465)

PSExecute (see page 7590)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

39.1.58.2.10.2 TWideDataSet.PSSetCommandText Method (string)

Assigns a specified SQL command to be executed by PSExecute.

Pascal

```
procedure PSSetCommandText(const CommandText: string); override; overload; deprecated;
```

C++

```
virtual __fastcall PSSetCommandText(const AnsiString CommandText);
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

The value of the CommandText parameter replaces the dataset's SQL statement (if it has one), or replaces the name of an underlying database table or stored procedure.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

39.1.59 TWideMemoField Class

TWideMemoField represents a memo field in a dataset.

Class Hierarchy

Pascal

```
TWideMemoField = class(TBlobField);
```

C++

```
class TWideMemoField : public TBlobField;
```

File

DB

Description

TWideMemoField encapsulates the fundamental behavior common to fields that contain wide text data of arbitrary length. Memo fields are a form of binary large object (BLOB) field where the data consists of simple text.

See Also

TDataSet (see page 6894)

TGraphicField (see page 7257)

TDBMemo (see page 9149)

TBlobField (see page 6801)

TDataSet (see page 6894)

TField (see page 7115)

TMemoField (see page 7348)

39.1.59.1 TWideMemoField Members

The following tables list the members exposed by TWideMemoField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 6810)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 6811)	This is FreeBuffers, a member of class TBlobField.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
V	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsString (see page 7548)	This is GetAsString, a member of class TWideMemoField.
V	GetAsVariant (see page 6811)	This is GetAsVariant, a member of class TBlobField.
V	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
V	GetAsWideString (see page 6812)	This is GetAsWideString, a member of class TBlobField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetBlobSize (see page 6812)	This is GetBlobSize, a member of class TBlobField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClassDesc (see page 6812)	This is GetClassDesc, a member of class TBlobField.
V	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
V	GetDataSize (see page 7133)	This is GetDataSize, a member of class TField.
V	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
V	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
V	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetIsNull (see page 6812)	This is GetIsNull, a member of class TBlobField.
V	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.

	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 6812)	This is GetText, a member of class TBlobField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7548)	This is SetAsString, a member of class TWideMemoField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 6817)	This is SetAsWideString, a member of class TBlobField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 6818)	This is SetText, a member of class TBlobField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 6818)	This is SetVarValue, a member of class TBlobField.
	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	Assign (see page 6809)	Copies a value to the BLOB field.
V	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 6810)	Clears the current value in the BLOB field.
V	Clear (see page 7126)	Sets the value of the field to NULL.
V	Create (see page 7547)	Creates an instance of TWideMemoField (see page 7541).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsBlob (see page 6813)	Indicates that this is a BLOB field.
V	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	LoadFromFile (see page 6814)	Loads BLOB data from a file into the field.
	LoadFromStream (see page 6815)	Loads BLOB data from a stream into the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 6816)	Saves the contents of the BLOB field to a file.
	SaveToStream (see page 6816)	Saves the contents of the BLOB field to a stream.
	SetData (see page 7141)	Assigns unformatted data to the field.
V	SetFieldType (see page 6817)	Specifies the type of the BLOB field.
V	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
V	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.

	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
R	BlobSize (see page 6819)	Indicates the number of bytes of data contained in the blob field on the current record.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	Modified (see page 6820)	Indicates whether a the value for the BLOB field was changed since it was last read from the underlying database.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	Transliterate (see page 6821)	Determines whether any text in the BLOB field should be converted to and from the character set identified by the dataset.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7549)	Value of instance's text.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	BlobType (see page 6819)	Identifies the type of the BLOB field associated with a BLOB field object.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
	GraphicHeader (see page 6820)	Indicates whether the field's value includes a Paradox graphic BLOB header.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Size (see page 6820)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.59.2 TWideMemoField.Create Constructor

Creates an instance of TWideMemoField (see page 7541).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWideMemoField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TWideMemoField (see page 7541).

Create sets the DataType (see page 7159) property to ftWideMemo. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

See Also

Destroy (see page 7122)

DataType (see page 7159)

Create (see page 7121)

Transliterate (see page 6821)

Owner (see page 2807)

39.1.59.3 TWideMemoField Methods

The methods of the TWideMemoField class are listed here.

Protected Methods

	Name	Description
	GetAsString (see page 7548)	This is GetAsString, a member of class TWideMemoField.
	SetAsString (see page 7548)	This is SetAsString, a member of class TWideMemoField.

39.1.59.3.1 TWideMemoField.GetAsString Method

Pascal

```
function GetAsString: string; override;
```

C++

```
virtual __fastcall AnsiString GetAsString();
```

Description

This is GetAsString, a member of class TWideMemoField.

39.1.59.3.2 TWideMemoField.SetAsString Method

Pascal

```
procedure SetAsString(const Value: string); override;
```

C++

```
virtual __fastcall SetAsString(const AnsiString Value);
```

Description

This is SetAsString, a member of class TWideMemoField.

39.1.59.4 TWideMemoField Properties

The properties of the TWideMemoField class are listed here.

Public Properties

	Name	Description
	Value (see page 7549)	Value of instance's text.

39.1.59.4.1 TWideMemoField.Value Property

Value of instance's text.

Pascal

```
property Value: WideString;
```

C++

```
__property BSTR Value;
```

Description

Value reads and writes the instance's text as a WideString.

See Also

TMemoField (see page 7348)

39.1.60 TWideStringField Class

TWideStringField represents a large string field in a dataset.

Class Hierarchy

Pascal

```
TWideStringField = class(TStringField);
```

C++

```
class TWideStringField : public TStringField;
```

File

DB

Description

TWideStringField encapsulates the fundamental behavior common to wide string fields. Wide string fields are similar to simple string fields except that their storage capacity is virtually unlimited and the server specifically returns the field as wide string and not string. An example of a wide string field is the Unicode field in Microsoft SQL Server. In all other ways, wide string fields serve the same purpose as string fields.

If you use the Fields editor at design time to create a persistent field component for the wide string field, you can access it by name at runtime. When using dynamic field components, you can access the TWideStringField instance using the dataset's

Fields property or FieldByName method.

See Also

TStringField (see page 7483)

TDataSet (see page 6894)

TDataSet (see page 6894)

TDataSet (see page 6894)

39.1.60.1 TWideStringField Members

The following tables list the members exposed by TWideStringField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7557)	This is CheckTypeSize, a member of class TWideStringField.
V	CopyData (see page 7557)	This is CopyData, a member of class TWideStringField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBCD (see page 7492)	This is GetAsBCD, a member of class TStringField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsBoolean (see page 7492)	This is GetAsBoolean, a member of class TStringField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsDateTime (see page 7492)	This is GetAsDateTime, a member of class TStringField.
V	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
V	GetAsFloat (see page 7492)	This is GetAsFloat, a member of class TStringField.
V	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
V	GetAsInteger (see page 7493)	This is GetAsInteger, a member of class TStringField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsSQLTimeStamp (see page 7493)	This is GetAsSQLTimeStamp, a member of class TStringField.
V	GetAsString (see page 7557)	This is GetAsString, a member of class TWideStringField.
V	GetAsVariant (see page 7558)	This is GetAsVariant, a member of class TWideStringField.

	GetAsWideString (see page 7558)	This is GetAsWideString, a member of class TWideStringField.
	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
	GetDataSize (see page 7558)	This is GetDataSize, a member of class TWideStringField.
	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
	GetDefaultWidth (see page 7494)	This is GetDefaultWidth, a member of class TStringField.
	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetText (see page 7494)	This is GetText, a member of class TStringField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 7558)	This is GetValue, a member of class TWideStringField.
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBCD (see page 7494)	This is SetAsBCD, a member of class TStringField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsBoolean (see page 7495)	This is SetAsBoolean, a member of class TStringField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsDateTime (see page 7495)	This is SetAsDateTime, a member of class TStringField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsFloat (see page 7495)	This is SetAsFloat, a member of class TStringField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsInteger (see page 7495)	This is SetAsInteger, a member of class TStringField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsSQLTimeStamp (see page 7495)	This is SetAsSQLTimeStamp, a member of class TStringField.
	SetAsString (see page 7558)	This is SetAsString, a member of class TWideStringField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7559)	This is SetAsWideString, a member of class TWideStringField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.

	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
V	SetSize (see page 7143)	This is SetSize, a member of class TField.
V	SetText (see page 7144)	This is SetText, a member of class TField.
V	SetVarValue (see page 7559)	This is SetVarValue, a member of class TWideStringField.
V	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 7126)	Sets the value of the field to NULL.
V	Create (see page 7555)	Creates an instance of a TWideStringField (see page 7549) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.
	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.

	Value (see page 7559)	Represents the data in the string field.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	EditMask (see page 7496)	Contains a mask that restricts the data that can be entered into a data field.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
	FixedChar (see page 7497)	Specifies whether the represented field is of Char type.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Size (see page 7497)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Transliterate (see page 7498)	Determines whether text should be converted to and from the character set identified by the dataset.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.60.2 TWideStringField.Create Constructor

Creates an instance of a TWideStringField (see page 7549) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWideStringField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TWideStringField (see page 7549). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created automatically by the dataset.

Create sets the DataType (see page 7159) property to ftWideString, Size (see page 7497) to 20, and Transliterate (see page 7498) to true. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TWideStringField (see page 7549). After instantiating a TWideStringField (see page 7549), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TWideStringField (see page 7549) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TWideStringField (see page 7549) with a dataset component by setting its DataSet (see page 7157) property.

The example below creates a TWideStringField (see page 7549) object for a field named BookList accessed through a TQuery named Query1.

```
var
  T: TWideStringField;
begin
  Query1.Close;
  T := TWideStringField.Create(Query1);
  T.FieldName := 'BookList';
  T.Name := Query1.Name + T.FieldName;
  T.Index := Query1.FieldCount;
  T.DataSet := Query1;
  Query1.FieldDefs.UpDate;
  Query1.Open;
end;
TWideStringField *T;
Query1->Close();
T = new TWideStringField(Query1);
T->FieldName = "LastName";
T->Name = Query1->Name + T->FieldName;
T->Index = Query1->FieldCount;
T->DataSet = Query1;
Query1->FieldDefs->UpDate();
Query1->Open();
```

See Also

Destroy (see page 7122)

DataType (see page 7159)

Size (see page 7178)

Create (see page 7121)

Transliterate (see page 7498)

Owner (see page 2807)

39.1.60.3 TWideStringField Methods

The methods of the TWideStringField class are listed here.

Protected Methods

	Name	Description
	CheckTypeSize (see page 7557)	This is CheckTypeSize, a member of class TWideStringField.
	CopyData (see page 7557)	This is CopyData, a member of class TWideStringField.
	GetAsString (see page 7557)	This is GetAsString, a member of class TWideStringField.
	GetAsVariant (see page 7558)	This is GetAsVariant, a member of class TWideStringField.
	GetAsWideString (see page 7558)	This is GetAsWideString, a member of class TWideStringField.
	GetDataSize (see page 7558)	This is GetDataSize, a member of class TWideStringField.
	GetValue (see page 7558)	This is GetValue, a member of class TWideStringField.
	SetAsString (see page 7558)	This is SetAsString, a member of class TWideStringField.
	SetAsWideString (see page 7559)	This is SetAsWideString, a member of class TWideStringField.
	SetVarValue (see page 7559)	This is SetVarValue, a member of class TWideStringField.

39.1.60.3.1 TWideStringField.CheckTypeSize Method

Pascal

```
class procedure CheckTypeSize(Value: Integer); override;
```

C++

```
virtual __fastcall CheckTypeSize(int Value);
```

Description

This is CheckTypeSize, a member of class TWideStringField.

39.1.60.3.2 TWideStringField.CopyData Method

Pascal

```
procedure CopyData(Source: Pointer; Dest: Pointer); override;
```

C++

```
virtual __fastcall CopyData(void * Source, void * Dest);
```

Description

This is CopyData, a member of class TWideStringField.

39.1.60.3.3 TWideStringField.GetAsString Method

Pascal

```
function GetAsString: string; override;
```

C++

```
virtual __fastcall AnsiString GetAsString();
```

Description

This is GetAsString, a member of class TWideStringField.

39.1.60.3.4 TWideStringField.GetAsVariant Method

Pascal

```
function GetAsVariant: Variant; override;
```

C++

```
virtual __fastcall Variant GetAsVariant();
```

Description

This is GetAsVariant, a member of class TWideStringField.

39.1.60.3.5 TWideStringField.GetAsWideString Method

Pascal

```
function GetAsWideString: WideString; override;
```

C++

```
virtual __fastcall BSTR GetAsWideString();
```

Description

This is GetAsWideString, a member of class TWideStringField.

39.1.60.3.6 TWideStringField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; override;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TWideStringField.

39.1.60.3.7 TWideStringField.GetValue Method

Pascal

```
function GetValue(var Value: WideString): Boolean;
```

C++

```
__fastcall Boolean GetValue(BSTR Value);
```

Description

This is GetValue, a member of class TWideStringField.

39.1.60.3.8 TWideStringField.SetAsString Method

Pascal

```
procedure SetAsString(const Value: string); override;
```

C++

```
virtual __fastcall SetAsString(const AnsiString Value);
```

Description

This is SetAsString, a member of class TWideStringField.

39.1.60.3.9 TWideStringField.SetAsWideString Method

Pascal

```
procedure SetAsWideString(const Value: WideString); override;
```

C++

```
virtual __fastcall SetAsWideString(const BSTR Value);
```

Description

This is SetAsWideString, a member of class TWideStringField.

39.1.60.3.10 TWideStringField.SetVarValue Method

Pascal

```
procedure SetVarValue(const Value: Variant); override;
```

C++

```
virtual __fastcall SetVarValue(const Variant Value);
```

Description

This is SetVarValue, a member of class TWideStringField.

39.1.60.4 TWideStringField Properties

The properties of the TWideStringField class are listed here.

Public Properties

	Name	Description
	Value (see page 7559)	Represents the data in the string field.

39.1.60.4.1 TWideStringField.Value Property

Represents the data in the string field.

Pascal

```
property Value: WideString;
```

C++

```
__property BSTR Value;
```

Description

Use Value to read data directly from and write data directly to a string field component at runtime. Using the Value property is the same as using the AsString (see page 7151) property. Use Value when you know the field component is a wide string field. Use AsString (see page 7151) when working with a generic field component.

When setting the Value property, TWideStringField (see page 7549) converts the assigned string from UTF-8 or ANSI to the character set for the dataset if the Transliterate (see page 7498) property is true. Strings that have more characters than permitted by the value of the Size (see page 7497) property are truncated when setting Value.

```
Table1TheWideStringField.Value := WideStringVariable;
```

See Also

AsString

Size (see page 7178)

Transliterate (see page 7498)

39.1.61 TWordField Class

TWordField represents an unsigned 16-bit integer field in a dataset.

Class Hierarchy

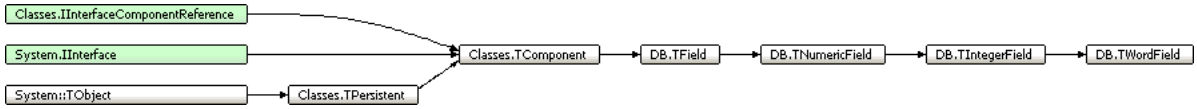

Pascal

```
TWordField = class(TIntegerField);
```

C++

```
class TWordField : public TIntegerField;
```

File

DB

Description

TWordField encapsulates the fundamental behavior common to fields that contain unsigned integer data. Word fields can hold values in the range 0 to 65535, and are used to hold values that are unsigned 16-bit whole numbers.

If you use the Fields editor at design time to create a persistent field component for the word field, you can access it by name at runtime. When using dynamic field components, you can access the TWordField instance using the dataset's Fields property or FieldByName method.

See Also

TDataSet (see page 6894)

TDataSet (see page 6894)

TDataSet (see page 6894)

TField (see page 7115)

TAutoIncField (see page 6768)

TSmallintField (see page 7461)

TIntegerField (see page 7301)

Using Default Formatting for Numeric

39.1.61.1 TWordField Members

The following tables list the members exposed by TWordField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AccessError (see page 7124)	This is AccessError, a member of class TField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Bind (see page 7126)	This is Bind, a member of class TField.
V	Change (see page 7126)	This is Change, a member of class TField.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 7126)	This is CheckInactive, a member of class TField.
V	CheckTypeSize (see page 7126)	This is CheckTypeSize, a member of class TField.
V	CopyData (see page 7127)	This is CopyData, a member of class TField.
	DataChanged (see page 7127)	This is DataChanged, a member of class TField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7127)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	FreeBuffers (see page 7128)	This is FreeBuffers, a member of class TField.
V	GetAsBCD (see page 7128)	This is GetAsBCD, a member of class TField.
V	GetAsBoolean (see page 7129)	This is GetAsBoolean, a member of class TField.
V	GetAsByteArray (see page 7129)	This is GetAsByteArray, a member of class TField.
V	GetAsCurrency (see page 7129)	This is GetAsCurrency, a member of class TField.
V	GetAsDateTime (see page 7129)	This is GetAsDateTime, a member of class TField.
V	GetAsFloat (see page 7129)	This is GetAsFloat, a member of class TField.
V	GetAsFloat (see page 7310)	This is GetAsFloat, a member of class TIntegerField.
V	GetAsInteger (see page 7130)	This is GetAsInteger, a member of class TField.
V	GetAsInteger (see page 7310)	This is GetAsInteger, a member of class TIntegerField.
V	GetAsSQLTimeStamp (see page 7130)	This is GetAsSQLTimeStamp, a member of class TField.
V	GetAsString (see page 7130)	This is GetAsString, a member of class TField.
V	GetAsString (see page 7311)	This is GetAsString, a member of class TIntegerField.
V	GetAsVariant (see page 7130)	This is GetAsVariant, a member of class TField.
V	GetAsVariant (see page 7311)	This is GetAsVariant, a member of class TIntegerField.
V	GetAsWideString (see page 7131)	This is GetAsWideString, a member of class TField.
V	GetCanModify (see page 7131)	This is GetCanModify, a member of class TField.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClassDesc (see page 7131)	This is GetClassDesc, a member of class TField.
V	GetDataSize (see page 7568)	This is GetDataSize, a member of class TWordField.
V	GetDefaultWidth (see page 7133)	This is GetDefaultWidth, a member of class TField.
V	GetFieldNo (see page 7134)	This is GetFieldNo, a member of class TField.
V	GetHasConstraints (see page 7134)	This is GetHasConstraints, a member of class TField.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIsNull (see page 7134)	This is GetIsNull, a member of class TField.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSize (see page 7135)	This is GetSize, a member of class TField.
	GetText (see page 7135)	This is GetText, a member of class TField.
	GetText (see page 7311)	This is GetText, a member of class TIntegerField.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 7312)	This is GetValue, a member of class TIntegerField.
	GetWideText (see page 7135)	This is GetWideText, a member of class TField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7137)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 7138)	This is PropertyChanged, a member of class TField.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	RangeError (see page 7368)	This is RangeError, a member of class TNumericField.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 7138)	This is ReadState, a member of class TField.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAsBCD (see page 7139)	This is SetAsBCD, a member of class TField.
	SetAsBoolean (see page 7139)	This is SetAsBoolean, a member of class TField.
	SetAsByteArray (see page 7139)	This is SetAsByteArray, a member of class TField.
	SetAsCurrency (see page 7139)	This is SetAsCurrency, a member of class TField.
	SetAsDateTime (see page 7139)	This is SetAsDateTime, a member of class TField.
	SetAsFloat (see page 7140)	This is SetAsFloat, a member of class TField.
	SetAsFloat (see page 7312)	This is SetAsFloat, a member of class TIntegerField.
	SetAsInteger (see page 7140)	This is SetAsInteger, a member of class TField.
	SetAsInteger (see page 7312)	This is SetAsInteger, a member of class TIntegerField.
	SetAsSQLTimeStamp (see page 7140)	This is SetAsSQLTimeStamp, a member of class TField.
	SetAsString (see page 7140)	This is SetAsString, a member of class TField.
	SetAsString (see page 7312)	This is SetAsString, a member of class TIntegerField.
	SetAsVariant (see page 7141)	This is SetAsVariant, a member of class TField.
	SetAsWideString (see page 7141)	This is SetAsWideString, a member of class TField.
	SetAutoGenerateValue (see page 7141)	This is SetAutoGenerateValue, a member of class TField.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 7142)	This is SetDataSet, a member of class TField.
	SetDataType (see page 7142)	This is SetDataType, a member of class TField.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayFormat (see page 7368)	This is SetDisplayFormat, a member of class TNumericField.
	SetEditFormat (see page 7368)	This is SetEditFormat, a member of class TNumericField.
	SetFieldKind (see page 7142)	This is SetFieldKind, a member of class TField.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentField (see page 7143)	This is SetParentField, a member of class TField.
	SetSize (see page 7143)	This is SetSize, a member of class TField.
	SetText (see page 7144)	This is SetText, a member of class TField.
	SetVarValue (see page 7144)	This is SetVarValue, a member of class TField.

	SetVarValue (see page 7312)	This is SetVarValue, a member of class TIntegerField.
	SetWideText (see page 7144)	This is SetWideText, a member of class TField.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 7124)	Copies the Value (see page 7180) property from another field component or assigns another object to the field component.
	AssignValue (see page 7124)	Sets the field value using one of the AsInteger (see page 7149), AsBoolean (see page 7147), AsString (see page 7151) or AsFloat (see page 7149) properties.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7126)	Sets the value of the field to NULL.
	Create (see page 7566)	Creates an instance of TWordField (see page 7560).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 7122)	Destroys an instance of TField (see page 7115).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FocusControl (see page 7128)	Forces a form's focus to the first data-aware component associated with the field object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetData (see page 7131)	Returns the unformatted data for the field.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 7134)	Returns the component which directs the loading and saving of the field.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 7135)	Returns true, indicating that field components act as part of a larger composite.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsBlob (see page 7136)	Indicates whether the field represents the data in a binary large object (BLOB) field.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsValidChar (see page 7136)	Indicates whether a particular character is valid for the field.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshLookupList (see page 7138)	Loads LookupList (see page 7172) with all the values for LookupResultField (see page 7173) that match distinct values of LookupKeyFields (see page 7172).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetData (see page 7141)	Assigns unformatted data to the field.
	SetFieldType (see page 7142)	Provides the interface for a method that can set the DataType (see page 7159) for the field.
	SetParentComponent (see page 7143)	This is SetParentComponent, a member of class TField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Validate (see page 7144)	Generates an OnValidate (see page 7183) event.

Public Properties

	Name	Description
	AsBCD (see page 7147)	Represents the field's value as a TBcd value.
	AsBoolean (see page 7147)	Represents the field's value as a boolean value.
	AsCurrency (see page 7148)	Represents the field's value as a Currency value.
	AsDateTime (see page 7148)	Represents the field's value as a TDateTime value.
	AsFloat (see page 7149)	Represents the field's value as a double value.
	AsInteger (see page 7149)	Represents the field's value as a 32-bit integer.
	AsSQLTimeStamp (see page 7150)	Represents the field's value as a TSQLTimeStamp.
	AsString (see page 7151)	Represents the field's value as a string (Delphi) or an AnsiString (C++).
	AsVariant (see page 7152)	Represents the Value (see page 7180) of the field as a Variant.

	AsWideString (see page 7152)	This is AsWideString, a member of class TField.
	AttributeSet (see page 7153)	Specifies the name of an attribute set in the data dictionary to apply to the field component.
	Calculated (see page 7154)	Determines whether the value of the field is calculated by the OnCalcFields event handler of its dataset.
R	CanModify (see page 7154)	Specifies whether a field can be modified.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurValue (see page 7155)	Represents the current value of the field component including changes made by other users of the database.
	DataSet (see page 7157)	Identifies the dataset to which a field component belongs.
R	DataSize (see page 7157)	Indicates the amount of memory needed to store a field component's value.
R	DataType (see page 7159)	Identifies the data type of the field component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 7160)	Represents the name of the field for display purposes.
R	DisplayText (see page 7162)	Represents the field's value as it is displayed in a data-aware control.
	EditMask (see page 7163)	Contains a mask that restricts the data that can be entered into a data field.
R	EditMaskPtr (see page 7163)	Provides read-only access to the EditMask (see page 7163) property.
R	FieldNo (see page 7166)	Indicates the ordinal position of the field's column in the underlying table or query result.
R	FullName (see page 7166)	Indicates the field name qualified by the full name of the parent field, if any.
R	IsIndexField (see page 7168)	Indicates whether a field is an index field.
R	IsNull (see page 7169)	Indicates whether the field has a value assigned to it.
	Lookup (see page 7170)	Determines whether the field is specified as a lookup field.
R	LookupList (see page 7172)	Indicates a cache of values from the LookupDataSet (see page 7171) indexed by a set of values from the KeyFields (see page 7169) property.
	NewValue (see page 7174)	Represents the current value of the field component including pending cached updates.
R	Offset (see page 7175)	Indicates the number of bytes added to the end of the current record buffer in the dataset for storing cached values (for calculated fields or BLOB fields).
R	OldValue (see page 7175)	Represents the original value of the field (as a Variant).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParentField (see page 7176)	Specifies the TObjectField (see page 7372) object from which this field is derived.
	Size (see page 7178)	Indicates the size used in the definition of the physical database field for data types that support different sizes.
	Text (see page 7179)	Contains the string to display in a data-aware control when the field is in edit mode.
	ValidChars (see page 7179)	Specifies the characters that can be included in the text representation of the field's value.
	Value (see page 7180)	Represents the data in a field component.
	Value (see page 7314)	Represents the value of the integer field.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 7181)	Occurs immediately after the data in the field is written to the record buffer.
	OnGetText (see page 7182)	Occurs when the DisplayText (see page 7162) or Text (see page 7179) properties of the field are referenced.
	OnSetText (see page 7182)	Occurs when the Text (see page 7179) property is assigned a value.
	OnValidate (see page 7183)	Occurs just before the data is written to the record buffer.

Published Properties

	Name	Description
	Alignment (see page 7146)	Determines how the field's data is displayed within a data-aware control.
	Alignment (see page 7369)	Determines how the field's data is displayed within a data-aware control.
	AutoGenerateValue (see page 7153)	Indicates whether the field values can be generated by the server rather than entered by the user.
	ConstraintErrorMessage (see page 7155)	Specifies a customized error message that appears when the user tries to set a value that violates the data constraints of the field.
	CustomConstraint (see page 7156)	Specifies an SQL string the enforces any application-specific constraints on the field's value.
	DefaultExpression (see page 7159)	Specifies an SQL expression that is assigned to the field if the user does not provide a value.
	DisplayFormat (see page 7369)	Determines how a numeric field's value is formatted for display in a data-aware control.
	DisplayLabel (see page 7160)	Contains the text to display in the corresponding column heading of a data grid.
	DisplayWidth (see page 7162)	Specifies the number of characters that should be used to display a field's value by a cooperating data-aware control.
	EditFormat (see page 7371)	Determines how a numeric field's value is formatted when it is being edited in a data-aware control.
	FieldKind (see page 7164)	Indicates whether a field represents a column in a dataset, a calculated field, or a lookup field.
	FieldName (see page 7164)	Indicates the name of the physical column in the underlying table or query result to which a field component is bound.
R	HasConstraints (see page 7167)	Indicates whether field values are constrained.
	ImportedConstraint (see page 7167)	Represents an SQL clause that enforces constraints stored on the server.
	Index (see page 7168)	Specifies the field component's index in the Fields property of the dataset.
	KeyFields (see page 7169)	Identifies the field or fields in the dataset that must be matched in a lookup dataset when doing a lookup.
	LookupCache (see page 7170)	Determines whether the values of a lookup field are cached or looked up dynamically every time the current record in the dataset changes.
	LookupDataSet (see page 7171)	Identifies the dataset used to look up field values.
	LookupKeyFields (see page 7172)	Identifies the field or fields in the lookup dataset to match when doing a lookup.
	LookupResultField (see page 7173)	Identifies the field from the lookup dataset whose value becomes the Value (see page 7180) property of the field component.
	MaxValue (see page 7313)	Limits the maximum value in the integer field.
	MinValue (see page 7313)	Limits the minimum value in the integer field.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Origin (see page 7176)	Indicates the name of the field in its original database table.
	ProviderFlags (see page 7176)	Specifies how a provider uses the field value when applying updates.
	ReadOnly (see page 7177)	Determines whether the field can be modified.
	Required (see page 7177)	Specifies whether a nonblank value for a field is required.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 7181)	Determines whether the field appears in a data grid.

39.1.61.2 TWordField.Create Constructor

Creates an instance of TWordField (see page 7560).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWordField(TComponent * AOwner);
```

Description

Most applications do not explicitly create instances of TWordField (see page 7560). Instead, the field components are created automatically, as persistent field components defined in the Fields editor at design time or as dynamic field components created

automatically by the dataset.

Create sets the DataType (see page 7159) property to ftWord, MinValue (see page 7313) to 0, and MaxValue (see page 7313) to 65535. The AOwner parameter specifies the component, typically a dataset, that becomes the new field's Owner. The Owner is responsible for freeing the component.

In the rare cases when you must create a persistent field component at runtime, call Create to create and initialize an instance of TWordField (see page 7560). After instantiating a TWordField (see page 7560), associate it with a specific field by setting its FieldName (see page 7164) property to the name of the field. Give the TWordField (see page 7560) a unique identifier in the Name property. Establish where the field appears in the collection of fields by providing an ordinal number in the Index (see page 7168) property. Associate the TWordField (see page 7560) with a dataset component by setting its DataSet (see page 7157) property.

The example below creates a TWordField (see page 7560) object for a field named Counter accessed through a TSQLDataSet named SQLDataSet1.

```
var
  T: TWordField;
begin
  SQLDataSet1.Close;
  T := TWordField.Create(SQLDataSet1);
  T.FieldName := 'Counter';
  T.Name := SQLDataSet1.Name + T.FieldName;
  T.Index := SQLDataSet1.FieldCount;
  T.DataSet := SQLDataSet1;
  SQLDataSet1.FieldDefs.UpDate;
  SQLDataSet1.Open;
end;
SQLDataSet1->Close();
TWordField *T = new TWordField(SQLDataSet1);
T->FieldName = "Counter";
T->Name = SQLDataSet1->Name + T->FieldName;
T->Index = SQLDataSet1->FieldCount;
T->DataSet = SQLDataSet1;
SQLDataSet1->FieldDefs->UpDate();
SQLDataSet1->Open();
```

See Also

DataType (see page 7159)

MaxValue (see page 7313)

MinValue (see page 7313)

Create (see page 7121)

Owner (see page 2807)

39.1.61.3 TWordField Methods

The methods of the TWordField class are listed here.

Protected Methods

	Name	Description
	GetDataSize (see page 7568)	This is GetDataSize, a member of class TWordField.

39.1.61.3.1 TWordField.GetDataSize Method

Pascal

```
function GetDataSize: Integer; override;
```

C++

```
virtual __fastcall int GetDataSize();
```

Description

This is GetDataSize, a member of class TWordField.

39.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IDBApplication (see page 7569)	Interface for database application.
	IDBScreen (see page 7571)	Interface to display device.
	IDBSession (see page 7573)	IDBSession is the interface for database sessions.
	IProviderSupport (see page 7576)	IProviderSupport is the interface that datasets use to enable a provider to represent them.
	IProviderSupport2 (see page 7587)	Interface datasets use to enable providers to represent them with Unicode support.

39.2.1 IDBApplication Interface

Interface for database application.

Class Hierarchy

DB.IDBApplication

Pascal

```
IDBApplication = interface;
```

C++

```
__interface IDBApplication;
```

File

DB

Description

IDBApplication is an interface for a database application.

39.2.1.1 IDBApplication Members

The following tables list the members exposed by IDBApplication.

Public Methods

	Name	Description
	GetTitle (see page 7570)	Get application title.
	ProcessMessages (see page 7570)	Allow an application to process queued messages.
	SetTitle (see page 7570)	Set application title.

Public Properties

	Name	Description
	Title (see page 7571)	Application title.

39.2.1.2 IDBApplication Methods

The methods of the IDBApplication class are listed here.

Public Methods

	Name	Description
	GetTitle (see page 7570)	Get application title.
	ProcessMessages (see page 7570)	Allow an application to process queued messages.
	SetTitle (see page 7570)	Set application title.

39.2.1.2.1 IDBApplication.GetTitle Method

Get application title.

Pascal

```
function GetTitle: string;
```

C++

```
__fastcall AnsiString GetTitle();
```

Description

GetTitle gets the title of a database application.

See Also

Title (see page 7571)

SetTitle (see page 7570)

39.2.1.2.2 IDBApplication.ProcessMessages Method

Allow an application to process queued messages.

Pascal

```
procedure ProcessMessages;
```

C++

```
__fastcall ProcessMessages();
```

Description

Call ProcessMessages to permit the application to process messages that are currently in the message queue. Neglecting message processing affects only the application calling ProcessMessages, not other applications. In lengthy operations, calling ProcessMessages periodically allows the application to respond to paint and other messages. .

39.2.1.2.3 IDBApplication.SetTitle Method

Set application title.

Pascal

```
procedure SetTitle(const Value: string);
```

C++

```
__fastcall SetTitle(const AnsiString Value);
```

Description

SetTitle sets the title of a database application.

See Also

Title (see page 7571)

GetTitle (see page 7570)

39.2.1.3 IDBApplication Properties

The properties of the IDBApplication class are listed here.

Public Properties

	Name	Description
	Title (see page 7571)	Application title.

39.2.1.3.1 IDBApplication.Title Property

Application title.

Pascal

```
property Title: string;
```

C++

```
__property AnsiString Title;
```

Description

Title allows reading and writing a database application's title.

See Also

GetTitle (see page 7570)

SetTitle (see page 7570)

39.2.2 IDBScreen Interface

Interface to display device.

Class Hierarchy

DB.IDBScreen

Pascal

```
IDBScreen = interface;
```

C++

```
__interface IDBScreen;
```

File

DB

Description

IDBScreen is an interface to a display device.

39.2.2.1 IDBScreen Members

The following tables list the members exposed by IDBScreen.

Public Methods

	Name	Description
	GetCursor (see page 7572)	Get screen cursor.
	SetCursor (see page 7572)	Set screen cursor.

Public Properties

	Name	Description
	Cursor (see page 7573)	Screen cursor.

39.2.2.2 IDBScreen Methods

The methods of the IDBScreen class are listed here.

Public Methods

	Name	Description
	GetCursor (see page 7572)	Get screen cursor.
	SetCursor (see page 7572)	Set screen cursor.

39.2.2.2.1 IDBScreen.GetCursor Method

Get screen cursor.

Pascal

```
function GetCursor: TDBScreenCursor;
```

C++

```
__fastcall TDBScreenCursor GetCursor();
```

Description

GetCursor gets the screen cursor.

See Also

Cursor (see page 7573)

SetCursor (see page 7572)

39.2.2.2.2 IDBScreen.SetCursor Method

Set screen cursor.

Pascal

```
procedure SetCursor(Cursor: TDBScreenCursor);
```

C++

```
__fastcall SetCursor(TDBScreenCursor Cursor);
```

Description

SetCursor sets the screen cursor.

See Also

Cursor (see page 7573)

GetCursor (see page 7572)

39.2.2.3 IDBScreen Properties

The properties of the IDBScreen class are listed here.

Public Properties

	Name	Description
	Cursor (see page 7573)	Screen cursor.

39.2.2.3.1 IDBScreen.Cursor Property

Screen cursor.

Pascal

```
property Cursor: TDBScreenCursor;
```

C++

```
__property TDBScreenCursor Cursor;
```

Description

Cursor allows reading and writing a screen cursor.

See Also

GetCursor (see page 7572)

SetCursor (see page 7572)

39.2.3 IDBSession Interface

IDBSession is the interface for database sessions.

Class Hierarchy

DB.IDBSession

Pascal

```
IDBSession = interface;
```

C++

```
__interface IDBSession;
```

File

DB

Description

IDBSession is a generic interface for logging into a database session. Components that allow an application to provide a password for opening password-protected tables or databases implement IDBSession. This allows the global PasswordDialog (see page 7643) function to work with any component that implements IDBSession to add or remove passwords.

Note: Component Method declarations that use IDBSession use the _di_IDBSession type instead. This type is a DelphiInterface wrapper around the IDBSession interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IDBSession >  _di_IDBSession;
```

See Also

TSession (see page 10093)

PasswordDialog

39.2.3.1 IDBSession Members

The following tables list the members exposed by IDBSession.

Public Methods

	Name	Description
	AddPassword (see page 7574)	Logs in to the session using a specified password.
	RemoveAllPasswords (see page 7575)	Removes all passwords from the session.
	RemovePassword (see page 7575)	Removes a specified password from the session.

39.2.3.2 IDBSession Methods

The methods of the IDBSession class are listed here.

Public Methods

	Name	Description
	AddPassword (see page 7574)	Logs in to the session using a specified password.
	RemoveAllPasswords (see page 7575)	Removes all passwords from the session.
	RemovePassword (see page 7575)	Removes a specified password from the session.

39.2.3.2.1 IDBSession.AddPassword Method

Logs in to the session using a specified password.

Pascal

```
procedure AddPassword(const APassword: string);
```

C++

```
__fastcall AddPassword(const AnsiString APassword);
```

Description

Call AddPassword to supply a password to a session. After calling AddPassword, the application can open any tables or databases protected by that password from within the session until the RemovePassword (see page 7575) or

RemoveAllPasswords (see page 7575) method retracts the password. If the password-protected table is closed, it can be reopened without resupplying the password.

APassword is the password to supply.

See Also

RemovePassword (see page 7575)

RemoveAllPasswords (see page 7575)

39.2.3.2.2 IDBSession.RemoveAllPasswords Method

Removes all passwords from the session.

Pascal

```
procedure RemoveAllPasswords;
```

C++

```
__fastcall RemoveAllPasswords();
```

Description

Use RemoveAllPasswords to delete all passwords that were previously supplied by a call to AddPassword (see page 7574). After a call to RemoveAllPasswords, no password-protected tables or databases can be reopened without resupplying a password using the AddPassword (see page 7574) method.

See Also

AddPassword (see page 7574)

RemovePassword (see page 7575)

39.2.3.2.3 IDBSession.RemovePassword Method

Removes a specified password from the session.

Pascal

```
procedure RemovePassword(const APassword: string);
```

C++

```
__fastcall RemovePassword(const AnsiString APassword);
```

Description

Use RemovePassword delete a specified password from the session. After a call to RemovePassword, any password-protected tables or databases that require the specified password can't be reopened until the password is supplied again by a call to the AddPassword (see page 7574) method.

APassword is the password to remove. It is a string that was previously supplied by the AddPassword (see page 7574) method.

See Also

AddPassword (see page 7574)

RemoveAllPasswords (see page 7575)

39.2.4 IProviderSupport Interface

IProviderSupport is the interface that datasets use to enable a provider to represent them.

Class Hierarchy

DB.IProviderSupport

Pascal

```
IProviderSupport = interface;
```

C++

```
__interface IProviderSupport;
```

File

DB

Description

TDataSetProvider uses the IProviderSupport interface to interact with its associated dataset. Any dataset that implements the IProviderSupport interface can be connected to a client dataset or XML broker via a provider. Once the provider connects the dataset that supports IProvider to a client dataset, that client dataset can obtain data from the dataset and apply updates back to it or its database server.

TDataSet (see page 6894), the base class for all datasets, includes support for IProviderSupport. This support appears as a set of protected methods. However, most of these methods are simple stubs that either do nothing or generate an exception. Most of the dataset classes (dbExpress datasets, BDE-enabled datasets, ADO-enabled datasets, and InterBase Express datasets) override these protected methods to implement the IProviderSupport interface in a more useful fashion. For datasets that simply rely on the implementation inherited from TDataSet (see page 6894) (for example, client datasets), the provider should set its ResolveToDataSet property to true.

Component writers that create their own custom descendants from TDataSet (see page 6894) must override all appropriate IProviderSupport methods if their datasets are to work in an application server or connected to a client dataset or XML broker via a local provider.

See Also

TDataSetProvider (see page 18508)

TDataSet (see page 6894)

Determining the Source of Data

39.2.4.1 IProviderSupport Members

The following tables list the members exposed by IProviderSupport.

Public Methods

	Name	Description
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.

	PSGetCommandText (see page 7579)	Gets the SQL command to be executed by PSExecute (see page 7578).
	PSGetCommandType (see page 7580)	Gets the command type to be executed by PSExecute (see page 7578).
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7585)	Assigns a specified SQL command to be executed by PSExecute (see page 7578).
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.

39.2.4.2 IProviderSupport Methods

The methods of the IProviderSupport class are listed here.

Public Methods

	Name	Description
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 7579)	Gets the SQL command to be executed by PSExecute (see page 7578).
	PSGetCommandType (see page 7580)	Gets the command type to be executed by PSExecute (see page 7578).
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7585)	Assigns a specified SQL command to be executed by PSExecute (see page 7578).
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.

	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.

39.2.4.2.1 IProviderSupport.PSEndTransaction Method

Ends a transaction that was started by a call to PSStartTransaction (see page 7586).

Pascal

```
procedure PSEndTransaction(Commit: Boolean);
```

C++

```
__fastcall PSEndTransaction(Boolean Commit);
```

Description

The provider component applies updates within a transaction, if possible. To do so, it uses PSEndTransaction in conjunction with the PSStartTransaction (see page 7586) method.

The Commit parameter indicates whether the dataset should commit the current transaction (Commit is true), or roll it back (Commit is false).

Note: TDataSetProvider only starts a transaction (Using PSStartTransaction (see page 7586)) if the PSInTransaction (see page 7583) method indicates that there is no transaction already established. That is, it does not assume nested transaction support is available. Thus, there is no need to ensure that PSEndTransaction does not terminate a transaction that was not started by the provider.

See Also

PSStartTransaction (see page 7586)

PSInTransaction (see page 7583)

applyupdates (see page 18459)

39.2.4.2.2 IProviderSupport.PSExecute Method

Executes the SQL command associated with the dataset.

Pascal

```
procedure PSExecute;
```

C++

```
__fastcall PSExecute();
```

Description

The provider component calls PSExecute to implement its Execute method. If the dataset represents a query or stored procedure that does not return a cursor, this method executes the query or stored procedure. If the provider Options includes poAllowCommandText and the dataset supports the PSSetCommandText (see page 7585) method, the statement specified by PSSetCommandText (see page 7585) may be executed instead.

If there is not statement to execute, PSExecute raises an EDatabaseError (see page 6731) exception.

See Also

PSSetCommandText (see page 7585)

Execute (see page 18466)

Options (see page 18440)

39.2.4.2.3 IProviderSupport.PSExecuteStatement Method

Executes a specified SQL command.

Pascal

```
function PSExecuteStatement(const ASQL: string; AParams: TParams; ResultSet: Pointer = nil):
Integer;
```

C++

```
__fastcall int PSExecuteStatement(const AnsiString ASQL, TParams AParams, void * ResultSet =
nil);
```

Description

If the value its ResolveToDataSet property is false, a provider component calls PSExecuteStatement (indirectly) to execute SQL statements that it generates for applying updates. Usually, the dataset passes these statements on to its database server for execution.

The ASQL parameter specifies the SQL command to execute.

AParams contains any parameter values that should be applied to ASQL before execution.

ResultSet allows PSExecuteStatement to return a dataset component that contains the results of a SELECT statement. If ResultSet is not nil (Delphi) or NULL (C++), PSExecuteStatement creates a new dataset component as the target of this pointer and fills it with the result set. The caller is responsible for freeing the dataset.

PSExecuteStatement returns the number of rows affected by executing ASQL.

See Also

PSUpdateRecord (see page 7586)

applyupdates (see page 18459)

ResolveToDataSet (see page 18531)

39.2.4.2.4 IProviderSupport.PSGetAttributes Method

Returns information that is stored in data packets as Name/Value pairs.

Pascal

```
procedure PSGetAttributes(List: TList);
```

C++

```
__fastcall PSGetAttributes(TList List);
```

Description

The provider component calls PSGetAttributes to add information about the dataset to the metadata stored with generated data packets. This information is stored as Name/Value pairs that are added to the list specified by the List parameter.

See Also

DoGetProviderAttributes (see page 18521)

39.2.4.2.5 IProviderSupport.PSGetCommandText Method

Gets the SQL command to be executed by PSExecute (see page 7578).

Pascal

```
function PSGetCommandText: string;
```

C++

```
__fastcall AnsiString PSGetCommandText();
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText (see page 7585), and subsequently executed by a call to PSExecute (see page 7578).

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

PSSetCommandText (see page 7585)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

39.2.4.2.6 IProviderSupport.PSGetCommandType Method

Gets the command type to be executed by PSExecute (see page 7578).

Pascal

```
function PSGetCommandType: TPSCommandType;
```

C++

```
__fastcall TPSCommandType PSGetCommandType();
```

Description

PSGetCommandType gets the command type executed by PSExecute (see page 7578). The type may be a type of SQL command or other command types, such as stored procedures.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

PSGetCommandText (see page 7579)

Execute (see page 18466)

Options (see page 18440)

39.2.4.2.7 IProviderSupport.PSGetDefaultOrder Method

Returns the definition of an index that imposes the default order on the data included in data packets.

Pascal

```
function PSGetDefaultOrder: TIndexDef;
```

C++

```
__fastcall TIndexDef PSGetDefaultOrder();
```

Description

The provider component calls PSGetDefaultOrder to generate a default index, called DEFAULT_ORDER, for the records that appear in data packets. This index defines the default sort order for records when they appear in the client dataset on the client.

PSGetDefaultOrder creates a TIndexDef (see page 7283) object to describe the default index. The provider that calls PSGetDefaultOrder subsequently frees this TIndexDef (see page 7283) object.

See Also

CreateDataPacket (see page 18518)

39.2.4.2.8 IProviderSupport.PSGetIndexDefs Method

Returns the definitions of all specified indexes defined for the dataset.

Pascal

```
function PSGetIndexDefs(IndexTypes: TIndexOptions = [ixPrimary..ixNonMaintained]): TIndexDefs;
```

C++

```
__fastcall TIndexDefs PSGetIndexDefs(TIndexOptions IndexTypes = [ixPrimary..ixNonMaintained]);
```

Description

The provider component calls PSGetIndexDefs to locate an index that contains information it needs. For example, it uses PSGetIndexDefs to locate an index that contains the key fields returned by PSGetKeyFields (see page 7581). This information allows the provider to locate the records that need to change when it applies updates. The provider also calls this method to locate indexes that it adds to the metadata of data packets.

IndexTypes indicates the options required of the returned indexes. PSGetIndexDefs returns all indexes whose Options property includes the specified flags.

PSGetIndexDefs returns a TIndexDefs (see page 7292) object that contains all the indexes of the dataset that match the IndexTypes parameter.

See Also

Options (see page 7291)

PSGetKeyFields (see page 7581)

PSGetDefaultOrder (see page 7580)

39.2.4.2.9 IProviderSupport.PSGetKeyFields Method

Returns the names of all fields required to uniquely identify records in the data packet.

Pascal

```
function PSGetKeyFields: string;
```

C++

```
__fastcall AnsiString PSGetKeyFields();
```

Description

The provider component calls PSGetKeyFields to determine what information is required to uniquely identify records in the data packet. This information allows it to locate the records that need to change when it applies updates.

PSGetKeyFields returns a semicolon-delimited string that lists the names of the key fields of the dataset. By default (as implemented by TDataSet (see page 6894)), these are the fields with pfInKey included in their ProviderFlags property.

See Also

PSGetDefaultOrder (see page 7580)

ProviderFlags (see page 7176)

PSGetIndexDefs (see page 7581)

39.2.4.2.10 IProviderSupport.PSGetParams Method

Returns the current parameter values of the dataset.

Pascal

```
function PSGetParams: TParams;
```

C++

```
__fastcall TParams PSGetParams();
```

Description

The provider component calls PSGetParams to implement its GetParams method.

PSGetParams returns a TParams (see page 7434) object that describes the current values of all parameters of the dataset. If the dataset does not have any parameters, PSGetParams returns nil (Delphi) or NULL (C++).

Note: The PSGetParams method should not create a new instance of TParams (see page 7434) for the return value. The provider that calls PSGetParams does not free the returned value.

See Also

GetParams (see page 18467)

39.2.4.2.11 IProviderSupport.PSGetQuoteChar Method

Returns the character or characters to use in generated SQL statements to enclose quoted strings.

Pascal

```
function PSGetQuoteChar: string;
```

C++

```
__fastcall AnsiString PSGetQuoteChar();
```

Description

The provider component calls PSGetQuoteChar (indirectly) when generating the dynamic SQL statements that apply updates. These statements are subsequently executed by a call to PSExecuteStatement (see page 7579).

PSGetQuoteChar returns the character or characters that delimit quoted strings in the generated SQL. This string appears both before and after quoted strings.

See Also

applyupdates (see page 18459)

PSExecuteStatement (see page 7579)

ResolveToDataSet (see page 18531)

39.2.4.2.12 IProviderSupport.PSGetTableName Method

Returns the name of the dataset table that appears in generated SQL statements.

Pascal

```
function PSGetTableName: string;
```

C++

```
__fastcall AnsiString PSGetTableName();
```

Description

PSGetTableName returns the name of the underlying database table against which updates are applied when the provider's ResolveToDataSet property is false. This value is used for generating the SQL statements that apply updates. The generated SQL statements are subsequently executed by a call to PSExecuteStatement (see page 7579).

If PSGetTableName does not return a table name, the provider must have an OnGetTableName event handler to supply this value.

See Also

applyupdates (see page 18459)

PSExecuteStatement (see page 7579)

ResolveToDataSet (see page 18531)

ongettablename (see page 18542)

39.2.4.2.13 IProviderSupport.PSGetUpdateException Method

Generates an EUpdateError (see page 6732) object based on another exception object.

Pascal

```
function PSGetUpdateException(E: Exception; Prev: EUpdateError): EUpdateError;
```

C++

```
__fastcall EUpdateError PSGetUpdateException(Exception E, EUpdateError Prev);
```

Description

The provider component calls PSGetUpdateException when the exception E occurs while applying updates. This method allows the provider to create a single EUpdateError (see page 6732) object that encapsulates all the exceptions that occur in an update operation.

PSGetUpdateError determines the error message, context string, and error code from the exception specified by E and creates a new EUpdateError (see page 6732) exception that includes that information as well as the information about a previous error as specified by Prev. The context string and error code for E may be specific to the Exception class associated with the dataset. By default, TDataSet (see page 6894) generates an EUpdateError (see page 6732) with an error code of 1 and an empty context string.

See Also

Create (see page 6733)

39.2.4.2.14 IProviderSupport.PSInTransaction Method

Indicates whether actions against the dataset occur in the context of a transaction.

Pascal

```
function PSInTransaction: Boolean;
```

C++

```
__fastcall Boolean PSInTransaction();
```

Description

The provider component applies updates within a transaction, if possible. To do so, it first checks whether a transaction is already underway, and if not, generates one using PSStartTransaction (see page 7586). To determine whether a transaction is already underway, the provider calls PSInTransaction.

PSInTransaction returns true if there is a transaction underway, and false otherwise.

See Also

PSStartTransaction (see page 7586)

PSEndTransaction (see page 7578)

39.2.4.2.15 IProviderSupport.PSIsSQLBased Method

Indicates whether the dataset includes native SQL support.

Pascal

```
function PSIsSQLBased: Boolean;
```

C++

```
__fastcall Boolean PSIsSQLBased();
```

Description

The provider calls PSIsSQLBased to determine whether the dataset supports some form of native SQL. This information is used when generating SQL commands that are executed by PSExecuteStatement (see page 7579).

Even if the dataset is not SQL-based, it may still allow the provider to execute SQL statements for applying updates. For example, BDE-enabled local databases such as Paradox and DBASE provide SQL support via the BDE's local SQL engine, but are not SQL-based. However, table and field names are not specified in the same way for the local SQL engine as they are for native SQL. The provider uses PSIsSQLBased to determine how these names should appear in generated SQL.

Note: The PSIsSQLSupported (see page 7584) method indicates whether the dataset can execute SQL commands at all, regardless of whether the dataset is SQL-based.

See Also

PSIsSQLSupported (see page 7584)

PSExecuteStatement (see page 7579)

39.2.4.2.16 IProviderSupport.PSIsSQLSupported Method

Indicates whether the dataset can execute SQL statements.

Pascal

```
function PSIsSQLSupported: Boolean;
```

C++

```
__fastcall Boolean PSIsSQLSupported();
```

Description

PSIsSQLSupported indicates whether the provider can call PSExecuteStatement (see page 7579) to execute an SQL statement. When PSIsSQLSupported returns false, calling PSExecuteStatement (see page 7579) generates an exception. The provider can still work with a dataset that returns false from PSIsSQLSupported, but it must set the ResolveToDataSet property to true, or provide data on a read-only basis.

See Also

PSIsSQLBased (see page 7584)

PSExecuteStatement (see page 7579)

ResolveToDataSet (see page 18531)

39.2.4.2.17 IProviderSupport.PSReset Method

Resets the dataset so that it is positioned on the first record.

Pascal

```
procedure PSReset;
```

C++

```
__fastcall PSReset();
```

Description

The provider calls PSReset to reposition the dataset on the first record before fetching records.

See Also

GetRecords (see page 18468)

AS_GetRecords (see page 16455)

39.2.4.2.18 IProviderSupport.PSSetCommandText Method

Assigns a specified SQL command to be executed by PSExecute (see page 7578).

Pascal

```
procedure PSSetCommandText(const CommandText: string);
```

C++

```
__fastcall PSSetCommandText(const AnsiString CommandText);
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute (see page 7578).

The value of the CommandText parameter replaces the dataset's SQL statement (if it has one), or replaces the name of an underlying database table or stored procedure.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

39.2.4.2.19 IProviderSupport.PSSetParams Method

Assigns the specified parameter values to the dataset.

Pascal

```
procedure PSSetParams(AParams: TParams);
```

C++

```
__fastcall PSSetParams(TParams AParams);
```

Description

PSSetParams assigns the parameter values specified by AParams to the parameters of the dataset.

See Also

Params (see page 8489)

PSGetParams (see page 7582)

39.2.4.2.20 IProviderSupport.PSStartTransaction Method

Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).

Pascal

```
procedure PSStartTransaction;
```

C++

```
__fastcall PSStartTransaction();
```

Description

The provider component applies updates within a transaction, if possible. To do so, it calls PSStartTransaction to start a transaction before applying updates, and calls PSEndTransaction (see page 7578) when all updates are applied or too many errors are encountered.

Note: TDataSetProvider only starts a transaction (Using PSStartTransaction) if the PSInTransaction (see page 7583) method indicates that there is no transaction already established. That is, it does not assume nested transaction support is available.

See Also

PSEndTransaction (see page 7578)

PSInTransaction (see page 7583)

applyupdates (see page 18459)

39.2.4.2.21 IProviderSupport.PSUpdateRecord Method

Applies a single update to the underlying database table or tables.

Pascal

```
function PSUpdateRecord(UpdateKind: TUpdateKind; Delta: TDataSet): Boolean;
```

C++

```
__fastcall Boolean PSUpdateRecord(TUpdateKind UpdateKind, TDataSet Delta);
```

Description

Before the provider generates SQL statements to apply an update, it calls PSUpdateRecord to let the dataset apply the update in some other fashion. For example, BDE-enabled datasets generate an OnUpdateRecord to allow the application to apply the update and, if the OnUpdateRecord event handler does not apply the update, uses any associated TUpdateSQL object to apply the update.

UpdateKind indicates whether the update is an insertion (ukInsert), deletion (ukDelete), or modification (ukModify).

Delta is the delta datapacket. The current record represents the record to be updated.

PSUpdateRecord returns true if it updates the record, false otherwise. If PSUpdateRecord returns false, the provider generates dynamic SQL to perform the update and calls PSExecuteStatement (see page 7579).

Note: PSUpdateRecord is only called if the provider's ResolveToDataSet property is false.

See Also

OnUpdateRecord (see page 9957)

TUpdateSQL (see page 10229)

applyupdates (see page 18459)

ResolveToDataSet (see page 18531)

PSExecuteStatement (see page 7579)

39.2.5 IProviderSupport2 Interface

Interface datasets use to enable providers to represent them with Unicode support.

Class Hierarchy

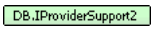

Pascal

```
IProviderSupport2 = interface;
```

C++

```
__interface IProviderSupport2;
```

File

DB

Description

IProviderSupport2 is similar to IProviderSupport (see page 7576) but provides Unicode support.

TWideDataSet (see page 7526) uses the IProviderSupport2 interface to interact with its associated dataset. Any dataset that implements the IProviderSupport2 interface can be connected to a client dataset or XML broker via a provider.

The TWideDataSet (see page 7526) class extends TDataSet (see page 6894) to support the Unicode string type. TWideDataSet (see page 7526) inherits the behavior of TDataSet (see page 6894) and TDataSet (see page 6894)'s IProviderSupport (see page 7576) interface, and TWideDataSet (see page 7526) supports IProviderSupport2.

BDE does not support a Unicode type, so TTable and TQuery are derived from TDataSet (see page 6894). However, TADODataSet and TSQLDataSet (dbExpress) derive from TWideDataSet (see page 7526), because ADO and dbExpress need to support Unicode data.

See Also

IProviderSupport (see page 7576)

TADODataSet (see page 1041)

TDataSet (see page 6894)

TSQLDataSet (see page 20265)

TWideDataSet (see page 7526)

Determining the Source of Data

39.2.5.1 IProviderSupport2 Members

The following tables list the members exposed by IProviderSupport2.

Public Methods

	Name	Description
	PSEndTransaction (see page 7589)	Ends a transaction that was started by a call to PSStartTransaction (see page 7597).
	PSExecute (see page 7590)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7590)	Executes a specified SQL command.
	PSGetAttributes (see page 7591)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 7591)	Gets the SQL command to be executed by PSExecute (see page 7590).
	PSGetCommandType (see page 7591)	Gets the command type to be executed by PSExecute (see page 7590).
	PSGetDefaultOrder (see page 7592)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7592)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7593)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7593)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7594)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7594)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7594)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7595)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7595)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7596)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7596)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7596)	Assigns a specified SQL command to be executed by PSExecute (see page 7590).
	PSSetParams (see page 7597)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7597)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7589).
	PSUpdateRecord (see page 7598)	Applies a single update to the underlying database table or tables.

39.2.5.2 IProviderSupport2 Methods

The methods of the IProviderSupport2 class are listed here.

Public Methods

	Name	Description
	PSEndTransaction (see page 7589)	Ends a transaction that was started by a call to PSStartTransaction (see page 7597).
	PSExecute (see page 7590)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7590)	Executes a specified SQL command.
	PSGetAttributes (see page 7591)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 7591)	Gets the SQL command to be executed by PSExecute (see page 7590).
	PSGetCommandType (see page 7591)	Gets the command type to be executed by PSExecute (see page 7590).
	PSGetDefaultOrder (see page 7592)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7592)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7593)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7593)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7594)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7594)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7594)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7595)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7595)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7596)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7596)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7596)	Assigns a specified SQL command to be executed by PSExecute (see page 7590).
	PSSetParams (see page 7597)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7597)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7589).
	PSUpdateRecord (see page 7598)	Applies a single update to the underlying database table or tables.

39.2.5.2.1 IProviderSupport2.PSEndTransaction Method

Ends a transaction that was started by a call to PSStartTransaction (see page 7597).

Pascal

```
procedure PSEndTransaction(Commit: Boolean);
```

C++

```
__fastcall PSEndTransaction(Boolean Commit);
```

Description

The provider component applies updates within a transaction, if possible. To do so, it uses PSEndTransaction in conjunction with the PSStartTransaction (see page 7597) method.

The Commit parameter indicates whether the dataset should commit the current transaction (Commit is true), or roll it back (Commit is false).

Note: TDataSetProvider only starts a transaction (Using PSStartTransaction (see page 7597)) if the PSInTransaction (see page 7595) method indicates that there is no transaction already established. That is, it does not assume nested transaction support is available. Thus, there is no need to ensure that PSEndTransaction does not terminate a transaction that was not started by the provider.

See Also

PSStartTransaction (see page 7597)

PSInTransaction (see page 7595)

applyupdates (see page 18459)

39.2.5.2.2 IProviderSupport2.PSExecute Method

Executes the SQL command associated with the dataset.

Pascal

```
procedure PSExecute;
```

C++

```
__fastcall PSExecute();
```

Description

The provider component calls PSExecute to implement its Execute method. If the dataset represents a query or stored procedure that does not return a cursor, this method executes the query or stored procedure. If the provider Options includes poAllowCommandText and the dataset supports the PSSetCommandText (see page 7596) method, the statement specified by PSSetCommandText (see page 7596) may be executed instead.

If there is no statement to execute, PSExecute raises an EDatabaseError (see page 6731) exception.

See Also

PSSetCommandText (see page 7596)

Execute (see page 18466)

Options (see page 18440)

39.2.5.2.3 IProviderSupport2.PSExecuteStatement Method

Executes a specified SQL command.

Pascal

```
function PSExecuteStatement(const ASQL: WideString; AParams: TParams; ResultSet: Pointer =
nil): Integer;
```

C++

```
__fastcall int PSExecuteStatement(const BSTR ASQL, TParams AParams, void * ResultSet = nil);
```

Description

If the value of its ResolveToDataSet property is false, a provider component calls PSExecuteStatement (indirectly) to execute SQL statements that it generates for applying updates. Usually, the dataset passes these statements on to its database server for execution.

The ASQL parameter specifies the SQL command to execute.

AParams contains any parameter values that should be applied to ASQL before execution.

ResultSet allows PSExecuteStatement to return a dataset component that contains the results of a SELECT statement. If ResultSet is not nil (Delphi) or NULL (C++), PSExecuteStatement creates a new dataset component as the target of this pointer and fills it with the result set. The caller is responsible for freeing the dataset.

PSExecuteStatement returns the number of rows affected by executing ASQL.

See Also

PSUpdateRecord (see page 7598)

applyupdates (see page 18459)

ResolveToDataSet (see page 18531)

39.2.5.2.4 IProviderSupport2.PSGetAttributes Method

Returns information that is stored in data packets as Name/Value pairs.

Pascal

```
procedure PSGetAttributes(List: TList);
```

C++

```
__fastcall PSGetAttributes(TList List);
```

Description

The provider component calls PSGetAttributes to add information about the dataset to the metadata stored with generated data packets. This information is stored as Name/Value pairs that are added to the list specified by the List parameter.

See Also

DoGetProviderAttributes (see page 18521)

39.2.5.2.5 IProviderSupport2.PSGetCommandText Method

Gets the SQL command to be executed by PSExecute (see page 7590).

Pascal

```
function PSGetCommandText: WideString;
```

C++

```
__fastcall BSTR PSGetCommandText();
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText (see page 7596), and subsequently executed by a call to PSExecute (see page 7590).

See Also

CommandText (see page 8465)

PSExecute (see page 7590)

PSSetCommandText (see page 7596)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

39.2.5.2.6 IProviderSupport2.PSGetCommandType Method

Gets the command type to be executed by PSExecute (see page 7590).

Pascal

```
function PSGetCommandType: TPSCommandType;
```

C++

```
__fastcall TPSCommandType PSGetCommandType();
```

Description

PSGetCommandType gets the command type executed by PSExecute (see page 7590). The type may be a type of SQL command or other command types, such as stored procedures.

See Also

CommandText (see page 8465)

PSExecute (see page 7590)

PSGetCommandText (see page 7591)

Execute (see page 18466)

Options (see page 18440)

39.2.5.2.7 IProviderSupport2.PSGetDefaultOrder Method

Returns the definition of an index that imposes the default order on the data included in data packets.

Pascal

```
function PSGetDefaultOrder: TIndexDef;
```

C++

```
__fastcall TIndexDef PSGetDefaultOrder();
```

Description

The provider component calls PSGetDefaultOrder to generate a default index, called DEFAULT_ORDER, for the records that appear in data packets. This index defines the default sort order for records when they appear in the client dataset on the client.

PSGetDefaultOrder creates a TIndexDef (see page 7283) object to describe the default index. The provider that calls PSGetDefaultOrder subsequently frees this TIndexDef (see page 7283) object.

See Also

CreateDataPacket (see page 18518)

39.2.5.2.8 IProviderSupport2.PSGetIndexDefs Method

Returns the definitions of all specified indexes defined for the dataset.

Pascal

```
function PSGetIndexDefs(IndexTypes: TIndexOptions = [ixPrimary..ixNonMaintained]): TIndexDefs;
```

C++

```
__fastcall TIndexDefs PSGetIndexDefs(TIndexOptions IndexTypes = [ixPrimary..ixNonMaintained]);
```

Description

The provider component calls PSGetIndexDefs to locate an index that contains information it needs. For example, it uses PSGetIndexDefs to locate an index that contains the key fields returned by PSGetKeyFields (see page 7593). This information allows the provider to locate the records that need to change when it applies updates. The provider also calls this method to locate indexes that it adds to the metadata of data packets.

IndexTypes indicates the options required of the returned indexes. PSGetIndexDefs returns all indexes whose Options property

includes the specified flags.

PSGetIndexDefs returns a TIndexDefs (see page 7292) object that contains all the indexes of the dataset that match the IndexTypes parameter.

See Also

Options (see page 7291)

PSGetKeyFields (see page 7593)

PSGetDefaultOrder (see page 7592)

39.2.5.2.9 IProviderSupport2.PSGetKeyFields Method

Returns the names of all fields required to uniquely identify records in the data packet.

Pascal

```
function PSGetKeyFields: WideString;
```

C++

```
__fastcall BSTR PSGetKeyFields();
```

Description

The provider component calls PSGetKeyFields to determine what information is required to uniquely identify records in the data packet. This information allows it to locate the records that need to change when it applies updates.

PSGetKeyFields returns a semicolon-delimited string that lists the names of the key fields of the dataset. By default (as implemented by TDataSet (see page 6894)), these are the fields with pfInKey included in their ProviderFlags property.

See Also

PSGetDefaultOrder (see page 7592)

ProviderFlags (see page 7176)

PSGetIndexDefs (see page 7592)

39.2.5.2.10 IProviderSupport2.PSGetParams Method

Returns the current parameter values of the dataset.

Pascal

```
function PSGetParams: TParams;
```

C++

```
__fastcall TParams PSGetParams();
```

Description

The provider component calls PSGetParams to implement its GetParams method.

PSGetParams returns a TParams (see page 7434) object that describes the current values of all parameters of the dataset. If the dataset does not have any parameters, PSGetParams returns nil (Delphi) or NULL (C++).

Note: The PSGetParams method should not create a new instance of TParams (see page 7434) for the return value. The provider that calls PSGetParams does not free the returned value.

See Also

GetParams (see page 18467)

39.2.5.2.11 IProviderSupport2.PSGetQuoteChar Method

Returns the character or characters to use in generated SQL statements to enclose quoted strings.

Pascal

```
function PSGetQuoteChar: WideString;
```

C++

```
__fastcall BSTR PSGetQuoteChar();
```

Description

The provider component calls PSGetQuoteChar (indirectly) when generating the dynamic SQL statements that apply updates. These statements are subsequently executed by a call to PSExecuteStatement (see page 7590).

PSGetQuoteChar returns the character or characters that delimit quoted strings in the generated SQL. This string appears both before and after quoted strings.

See Also

applyupdates (see page 18459)

PSExecuteStatement (see page 7590)

ResolveToDataSet (see page 18531)

39.2.5.2.12 IProviderSupport2.PSGetTableName Method

Returns the name of the dataset table that appears in generated SQL statements.

Pascal

```
function PSGetTableName: WideString;
```

C++

```
__fastcall BSTR PSGetTableName();
```

Description

PSGetTableName returns the name of the underlying database table against which updates are applied when the provider's ResolveToDataSet property is false. This value is used for generating the SQL statements that apply updates. The generated SQL statements are subsequently executed by a call to PSExecuteStatement (see page 7590).

If PSGetTableName does not return a table name, the provider must have an OnGetTableName event handler to supply this value.

See Also

applyupdates (see page 18459)

PSExecuteStatement (see page 7590)

ResolveToDataSet (see page 18531)

ongettablename (see page 18542)

39.2.5.2.13 IProviderSupport2.PSGetUpdateException Method

Generates an EUpdateError (see page 6732) object based on another exception object.

Pascal

```
function PSGetUpdateException(E: Exception; Prev: EUpdateError): EUpdateError;
```

C++

```
__fastcall EUpdateError PSGetUpdateException(Exception E, EUpdateError Prev);
```

Description

The provider component calls PSGetUpdateException when the exception E occurs while applying updates. This method allows the provider to create a single EUpdateError (see page 6732) object that encapsulates all the exceptions that occur in an update operation.

PSGetUpdateError determines the error message, context string, and error code from the exception specified by E and creates a new EUpdateError (see page 6732) exception that includes that information as well as the information about a previous error as specified by Prev. The context string and error code for E may be specific to the Exception class associated with the dataset. By default, TDataSet (see page 6894) generates an EUpdateError (see page 6732) with an error code of 1 and an empty context string.

See Also

Create (see page 6733)

39.2.5.2.14 IProviderSupport2.PSInTransaction Method

Indicates whether actions against the dataset occur in the context of a transaction.

Pascal

```
function PSInTransaction: Boolean;
```

C++

```
__fastcall Boolean PSInTransaction();
```

Description

The provider component applies updates within a transaction, if possible. To do so, it first checks whether a transaction is already underway, and if not, generates one using PSStartTransaction (see page 7597). To determine whether a transaction is already underway, the provider calls PSInTransaction.

PSInTransaction returns true if there is a transaction underway, and false otherwise.

See Also

PSStartTransaction (see page 7597)

PSEndTransaction (see page 7589)

39.2.5.2.15 IProviderSupport2.PSIsSQLBased Method

Indicates whether the dataset includes native SQL support.

Pascal

```
function PSIsSQLBased: Boolean;
```

C++

```
__fastcall Boolean PSIsSQLBased();
```

Description

The provider calls PSIsSQLBased to determine whether the dataset supports some form of native SQL. This information is used

when generating SQL commands that are executed by PSExecuteStatement (see page 7590).

Note: The PSIsSQLSupported (see page 7596) method indicates whether the dataset can execute SQL commands at all, regardless of whether the dataset is SQL-based.

See Also

PSIsSQLSupported (see page 7596)

PSExecuteStatement (see page 7590)

39.2.5.2.16 IProviderSupport2.PSIsSQLSupported Method

Indicates whether the dataset can execute SQL statements.

Pascal

```
function PSIsSQLSupported: Boolean;
```

C++

```
__fastcall Boolean PSIsSQLSupported();
```

Description

PSIsSQLSupported indicates whether the provider can call PSExecuteStatement (see page 7590) to execute an SQL statement. When PSIsSQLSupported returns false, calling PSExecuteStatement (see page 7590) generates an exception. The provider can still work with a dataset that returns false from PSIsSQLSupported, but it must set the ResolveToDataSet property to true, or provide data on a read-only basis.

See Also

PSIsSQLBased (see page 7595)

PSExecuteStatement (see page 7590)

ResolveToDataSet (see page 18531)

39.2.5.2.17 IProviderSupport2.PSReset Method

Resets the dataset so that it is positioned on the first record.

Pascal

```
procedure PSReset;
```

C++

```
__fastcall PSReset();
```

Description

The provider calls PSReset to reposition the dataset on the first record before fetching records.

See Also

GetRecords (see page 18468)

AS_GetRecords (see page 16455)

39.2.5.2.18 IProviderSupport2.PSSetCommandText Method

Assigns a specified SQL command to be executed by PSExecute (see page 7590).

Pascal

```
procedure PSSetCommandText(const CommandText: WideString);
```

C++

```
__fastcall PSSetCommandText(const BSTR CommandText);
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute (see page 7590).

The value of the CommandText parameter replaces the dataset's SQL statement (if it has one), or replaces the name of an underlying database table or stored procedure.

See Also

CommandText (see page 8465)

PSExecute (see page 7590)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

39.2.5.2.19 IProviderSupport2.PSSetParams Method

Assigns the specified parameter values to the dataset.

Pascal

```
procedure PSSetParams(AParams: TParams);
```

C++

```
__fastcall PSSetParams(TParams AParams);
```

Description

PSSetParams assigns the parameter values specified by AParams to the parameters of the dataset.

See Also

Params (see page 8489)

PSGetParams (see page 7593)

39.2.5.2.20 IProviderSupport2.PSStartTransaction Method

Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7589).

Pascal

```
procedure PSStartTransaction;
```

C++

```
__fastcall PSStartTransaction();
```

Description

The provider component applies updates within a transaction, if possible. To do so, it calls PSStartTransaction to start a transaction before applying updates, and calls PSEndTransaction (see page 7589) when all updates are applied or too many

errors are encountered.

Note: TDataSetProvider only starts a transaction (Using PSStartTransaction) if the PSInTransaction (see page 7595) method indicates that there is no transaction already established. That is, it does not assume nested transaction support is available.

See Also

PSEndTransaction (see page 7589)

PSInTransaction (see page 7595)

applyupdates (see page 18459)

39.2.5.2.21 IProviderSupport2.PSUpdateRecord Method

Applies a single update to the underlying database table or tables.

Pascal

```
function PSUpdateRecord(UpdateKind: TUpdateKind; Delta: TDataSet): Boolean;
```

C++

```
__fastcall Boolean PSUpdateRecord(TUpdateKind UpdateKind, TDataSet Delta);
```

Description

Before the provider generates SQL statements to apply an update, it calls PSUpdateRecord to let the dataset apply the update in some other fashion.

UpdateKind indicates whether the update is an insertion (ukInsert), deletion (ukDelete), or modification (ukModify).

Delta is the delta datapacket. The current record represents the record to be updated.

PSUpdateRecord returns true if it updates the record, false otherwise. If PSUpdateRecord returns false, the provider generates dynamic SQL to perform the update and calls PSExecuteStatement (see page 7590).

Note: PSUpdateRecord is only called if the provider's ResolveToDataSet property is false.

See Also

OnUpdateRecord (see page 9957)

TUpdateSQL (see page 10229)

applyupdates (see page 18459)

ResolveToDataSet (see page 18531)

PSExecuteStatement (see page 7590)

39.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	BuffersEqual (see page 7599)	This is function DB.BuffersEqual.
	DatabaseError (see page 7599)	Creates and raises an EDatabaseError (see page 6731) exception.
	DatabaseErrorFmt (see page 7600)	Creates and raises an EDatabaseError (see page 6731) exception with a formatted error message.
	DisposeMem (see page 7600)	This is function DB.DisposeMem.
	ExtractFieldName (see page 7601)	Returns the next name in a semicolon-delimited string of field names.
	ExtractFieldName (see page 7601)	Returns the next name in a semicolon-delimited string of field names.
	GetFieldProperty (see page 7602)	Returns a TField (see page 7115) object for a dataset and data control.
	RegisterFields (see page 7602)	Registers a set of field components so that they appear in the Dataset designer.
	VarTypeToDataType (see page 7603)	Returns the field type value that most closely corresponds to a Variant type.

39.3.1 DB.BuffersEqual Function

Pascal

```
function BuffersEqual(Buf1: Pointer; Buf2: Pointer; Size: Integer): Boolean;
```

C++

```
Boolean BuffersEqual(void * Buf1, void * Buf2, int Size);
```

File

DB

Description

This is function DB.BuffersEqual.

39.3.2 DB.DatabaseError Function

Creates and raises an EDatabaseError (see page 6731) exception.

Pascal

```
procedure DatabaseError(const Message: WideString; Component: TComponent = nil);
```

C++

```
DatabaseError(const BSTR Message, TComponent * Component = nil);
```

File

DB

Description

Call DatabaseError to raise an EDatabaseError (see page 6731) exception, using Message as the text for the exception. If a

component is provided as the second parameter, the message is prefixed by the name of the component to help in interpreting the error message. In C++, the second parameter can be NULL.

Calling DatabaseError rather than creating and raising the exception directly in code reduces the overall code size of the application.

See Also

DbiError

DatabaseErrorFmt

39.3.3 DB.DatabaseErrorFmt Function

Creates and raises an EDatabaseError (see page 6731) exception with a formatted error message.

Pascal

```
procedure DatabaseErrorFmt(const Message: WIdeString; const Args: array of const; Component:
TComponent = nil);
```

C++

```
DatabaseErrorFmt(const BSTR Message, const array of const Args, TComponent * Component = nil);
```

File

DB

Description

Call DatabaseError (see page 7599) to raise an EDatabaseError (see page 6731) exception. The error message for the exception is assembled from the format string specified by the Message parameter and applying the Args parameter. In C++, the Args_Size parameter specifies the index of the last argument in Args (one less than the number of arguments). If a component is provided as the final parameter, the message is prefixed by the name of the component to help in interpreting the error message.In C++, this parameter can be NULL.

Call DatabaseErrorFmt rather than creating and raising the exception directly in code to reduce the overall code size of the application.

See Also

DbiError

DatabaseError

39.3.4 DB.DisposeMem Function

Pascal

```
procedure DisposeMem(var Buffer; Size: Integer);
```

C++

```
DisposeMem( Buffer, int Size);
```

File

DB

Description

This is function DB.DisposeMem.

39.3.5 DB.ExtractFieldName Function

Returns the next name in a semicolon-delimited string of field names.

Pascal

```
function ExtractFieldName(const Fields: string; var Pos: Integer): string; overload;
deprecated;
function ExtractFieldName(const Fields: WideString; var Pos: Integer): WideString; overload;
```

C++

```
AnsiString ExtractFieldName(const AnsiString Fields, int Pos);
BSTR ExtractFieldName(const BSTR Fields, int Pos);
```

File

DB

Description

Use ExtractFieldName when parsing a semicolon-delimited string of names such as those used as parameters to many methods on database-specific objects.

Fields is a semicolon-delimited string of names.

Pos is the starting position of the next string in Fields. On exit from the function, Pos is updated to the starting position of the following string. If there is no following string, on exit, Pos is one more than the length of Fields.

ExtractFieldName returns a string with the single name that began at position Pos when the function was called.

ExtractFieldName is typically used in a loop, as follows:

Pos := 1;

```
while Pos <= Length(FieldNames) do
begin
  Name := ExtractFieldName(FieldNames, Pos);
  // do something with Name
end;
```

int Pos = 1;

```
while (Pos <= FieldNames.Length())
{
  AnsiString Name = ExtractFieldName(FieldNames, Pos);
  // do something with Name
}
```

See Also

TDataSet

TFields

39.3.6 DB.GetFieldProperty Function

Returns a TField (see page 7115) object for a dataset and data control.

Pascal

```
function GetFieldProperty(DataSet: TDataSet; Control: TComponent; const FieldName:
WideString): TField;
```

C++

```
TField GetFieldProperty(TDataSet DataSet, TComponent * Control, const BSTR FieldName);
```

File

DB

Description

GetFieldProperty returns a reference to a field given the dataset component that contains the field, the data-aware control that accesses the field, and the field name. GetFieldProperty returns a value of type TField (see page 7115).

DataSet is the dataset component containing the field.

Control is the data-aware control that accesses the field.

FieldName is the name of the field for which to return a TField (see page 7115).

See Also

TDataSet

TFields

39.3.7 DB.RegisterFields Function

Registers a set of field components so that they appear in the Dataset designer.

Pascal

```
procedure RegisterFields(const FieldClasses: array of TFieldClass);
```

C++

```
RegisterFields(const array of TFieldClass FieldClasses);
```

File

DB

Description

Call RegisterFields to allow custom descendants of TField (see page 7115) to appear in the field type drop down of the new field dialog box.

The FieldClasses parameter is an array of TField (see page 7115) descendants.In C++, the FieldClasses_Size is the index of the last class in the array (one less than the number of class types).

See Also

TField

TDataSetDesigner

TDataset

39.3.8 DB.VarTypeToDataType Function

Returns the field type value that most closely corresponds to a Variant type.

Pascal

```
function VarTypeToDataType(VarType: Integer): TFieldType;
```

C++

```
TFieldType VarTypeToDataType(int VarType);
```

File

DB

Description

Call VarTypeToDataType to infer the type of field component that should hold a value, given its Variant type. Use the VarType function to obtain a Variant type from a Variant.

See Also

VarType

39.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAutoRefreshFlag (see page 7604)	TAutoRefreshFlag Indicates whether the server provides field values, and if so, why.
	TBlobStreamMode (see page 7605)	TBlobStreamMode indicates how a BLOB stream can be used.
	TBookmarkFlag (see page 7606)	This is record DB.TBookmarkFlag.
	TDataAction (see page 7606)	TDataAction indicates how an application should respond to a database error condition.
	TDataEvent (see page 7607)	TDataEvent identifies an event that occurs to a dataset.
	TDataSetState (see page 7608)	TDataSetState indicates the current operating mode of a dataset component.
	TDBScreenCursor (see page 7610)	This is record DB.TDBScreenCursor.
	TFieldAttribute (see page 7611)	TFieldAttribute and TFieldAttributes (see page 7632) indicate various attributes of a field.
	TFieldKind (see page 7611)	TFieldKind indicates what type of information a field component represents.
	TFieldType (see page 7612)	TFieldType is a set of values that parallel the data types of fields in tables.
	TFilterOption (see page 7615)	TFilterOption and TFilterOptions (see page 7635) influence how filter strings are applied.
	TGetMode (see page 7615)	This is record DB.TGetMode.
	TGetResult (see page 7616)	This is record DB.TGetResult.
	TGroupPosInd (see page 7616)	TGroupPosInd and TGroupPosInds (see page 7636) indicate the position of the current record within its group.
	TIndexOption (see page 7617)	This is record DB.TIndexOption.
	TLocateOption (see page 7618)	This is record DB.TLocateOption.
	TParamType (see page 7619)	TParamType and TParamTypes (see page 7637) represent the use of parameters on datasets that represent queries or stored procedures.
	TProviderFlag (see page 7620)	TProviderFlag and TProviderFlags (see page 7638) indicate how a provider uses a field value when applying updates.
	TPSCommandType (see page 7621)	This is record DB.TPSCommandType.
	TUpdateAction (see page 7621)	TUpdateAction Indicates how an a dataset should proceed after an event handler has processed an update.
	TUpdateKind (see page 7622)	TUpdateKind defines the values for the UpdateKind parameter of OnUpdateError and OnUpdateRecord event handlers.
	TUpdateMode (see page 7623)	TUpdateMode indicates how to locate records so that they can be updated.
	TUpdateStatus (see page 7623)	TUpdateStatus indicates the update status of a record.

Records

	Name	Description
	TDateTimeRec (see page 7610)	This is record DB.TDateTimeRec.
	TLookupListEntry (see page 7618)	This is record DB.TLookupListEntry.
	TPacketAttribute (see page 7618)	This is record DB.TPacketAttribute.

39.4.1 DB.TAutoRefreshFlag Enumeration

TAutoRefreshFlag Indicates whether the server provides field values, and if so, why.

Pascal

```
TAutoRefreshFlag = (
```

```
  arNone,
  arAutoInc,
  arDefault
);
```

C++

```
enum TAutoRefreshFlag {
  arNone,
  arAutoInc,
  arDefault
};
```

File

DB

Description

TAutoRefreshFlag is the type of the AutoGenerateValue property of a field component. It indicates whether the server automatically supplies data for a field. TAutoRefreshFlag includes the following values:

Value	Meaning
arNone	The field does not have a default value and is not an autoincrement field. The server does not automatically supply values for this type of field.
arAutoInc	The field is an autoincrement field.
arDefault	The field has a default value maintained by the server.

See Also

TField

39.4.2 DB.TBlobStreamMode Enumeration

TBlobStreamMode indicates how a BLOB stream can be used.

Pascal

```
TBlobStreamMode = (
  bmRead,
  bmWrite,
  bmReadWrite
);
```

C++

```
enum TBlobStreamMode {
  bmRead,
  bmWrite,
  bmReadWrite
};
```

File

DB

Description

When a BLOB stream is first created, TBlobStreamMode is used to specify how that stream can be used. The following table lists the possible values.

Value	Meaning
bmRead	The stream is used to read from a BLOB field.
bmWrite	The stream is used to write from a BLOB field.
bmReadWrite	The stream is used to modify a BLOB field's value.

See Also

TBlobStream

TClientBlobStream

TSQLBlobStream

TADOBlobStream

TDataSet

39.4.3 DB.TBookmarkFlag Enumeration

Pascal

```
TBookmarkFlag = (
  bfCurrent,
  bfBOF,
  bfEOF,
  bfInserted
);
```

C++

```
enum TBookmarkFlag {
  bfCurrent,
  bfBOF,
  bfEOF,
  bfInserted
};
```

File

DB

Description

This is record DB.TBookmarkFlag.

39.4.4 DB.TDataAction Enumeration

TDataAction indicates how an application should respond to a database error condition.

Pascal

```
TDataAction = (
  daFail,
  daAbort,
  daRetry
);
```

C++

```
enum TDataAction {
  daFail,
  daAbort,
  daRetry
};
```

File

DB

Description

TDataAction is the type of a parameter in certain dataset event handlers that indicates how the dataset should proceed after the event handler exits. The following table summarizes the possible values:

Value	Meaning
daFail	Abort the operation that led to an error and display an error message.
daAbort	Abort the operation that led to an error without displaying a message.
daRetry	Repeat the operation that led to an error. The cause of the error should be corrected within the event handler before returning this value.

39.4.5 DB.TDataEvent Enumeration

TDataEvent identifies an event that occurs to a dataset.

Pascal

```
TDataEvent = (
  deFieldChange,
  deRecordChange,
  deDataSetChange,
  deDataSetScroll,
  deLayoutChange,
  deUpdateRecord,
  deUpdateState,
  deCheckBrowseMode,
  dePropertyChange,
  deFieldListChange,
  deFocusControl,
  deParentScroll,
  deConnectChange,
  deReconcileError,
  deDisabledStateChange
);
```

C++

```
enum TDataEvent {
  deFieldChange,
  deRecordChange,
  deDataSetChange,
  deDataSetScroll,
  deLayoutChange,
  deUpdateRecord,
  deUpdateState,
  deCheckBrowseMode,
  dePropertyChange,
  deFieldListChange,
```

```
    deFocusControl,
    deParentScroll,
    deConnectChange,
    deReconcileError,
    deDisabledStateChange
  };
```

File

DB

Description

The following table describes the possible values:

Value	Meaning
deFieldChange	A field value changed.
deRecordChange	The contents of the current record changed.
deDataSetChange	A change occurred that affects the entire dataset (such as the insertion or deletion of records, changes to the key, or edits).
deDataSetScroll	The set of displayed records was scrolled.
deLayoutChange	The layout of data in a data-aware control changed.
deUpdateRecord	Edits to the current record were posted.
deUpdateState	The state of the dataset changed.
deCheckBrowseMode	The state of the dataset is about to change.
dePropertyChange	A property of the dataset or one of its fields changed.
deFieldListChange	The list of fields in the dataset changed.
deFocusControl	Focus needs to shift to the representation of a specified field.
deParentScroll	The parent for which the dataset is a nested detail scrolled.
deConnectChange	The connection to a remote dataset changed.
deReconcileError	The dataset encountered reconcile errors when applying updates.

39.4.6 DB.TDataSetState Enumeration

TDataSetState indicates the current operating mode of a dataset component.

Pascal

```
  TDataSetState = (
    dsInactive,
    dsBrowse,
    dsEdit,
    dsInsert,
    dsSetKey,
    dsCalcFields,
    dsFilter,
    dsNewValue,
    dsOldValue,
    dsCurValue,
    dsBlockRead,
    dsInternalCalc,
    dsOpening
  );
```

C++

```
enum TDataSetState {
  dsInactive,
  dsBrowse,
  dsEdit,
  dsInsert,
  dsSetKey,
  dsCalcFields,
  dsFilter,
  dsNewValue,
  dsOldValue,
  dsCurValue,
  dsBlockRead,
  dsInternalCalc,
  dsOpening
};
```

File

DB

Description

TDataSetState represents the values that may be in the State property of a dataset component. TDataSetState indicates the current operating mode of that dataset component. The following table lists all possible values in the TDataSetState type and describes what they indicate in the State property:

Value	Meaning
dsInactive	Dataset is closed, so its data is unavailable.
dsBrowse	Data can be viewed, but not changed. This is the default state of an open dataset.
dsEdit	Active record can be modified.
dsInsert	The active record is a newly inserted buffer that has not been posted. This record can be modified and then either posted or discarded.
dsSetKey	TClientDataSet only. Record searching is enabled, or a SetRange operation is under way. A restricted set of data can be viewed, and no data can be edited or inserted.
dsCalcFields	An OnCalcFields event is in progress. Noncalculated fields cannot be edited, and new records cannot be inserted.
dsFilter	An OnFilterRecord event is in progress. A restricted set of data can be viewed. No data can edited or inserted.
dsNewValue	Temporary state used internally when a field component's NewValue property is accessed.
dsOldValue	Temporary state used internally when a field component's OldValue property is accessed.
dsCurValue	Temporary state used internally when a field component's CurValue property is accessed.
dsBlockRead	Data-aware controls are not updated and events are not triggered when moving to the next record.
dsInternalCalc	Temporary state used internally when values need to be calculated for a field that has a FieldKind of fkInternalCalc.
dsOpening	DataSet is in the process of opening but has not finished.

See Also

TDataSet

TDataSource

39.4.7 DB.TDateTimeRec Record

Pascal

```
TDateTimeRec = record
  case TFieldType of
    ftDate: (Date: Longint;);
    ftDateTime: (DateTime: TDateTimeAlias;);
    ftTime: (Time: Longint;);
end;
```

C++

```
struct TDateTimeRec {
  Longint Date;
  Longint Time;
  TDateTimeAlias DateTime;
};
```

File

DB

Description

This is record DB.TDateTimeRec.

39.4.8 DB.TDBScreenCursor Enumeration

Pascal

```
TDBScreenCursor = (
  dcrDefault,
  dcrHourGlass,
  dcrSQLWait,
  dcrOther
);
```

C++

```
enum TDBScreenCursor {
  dcrDefault,
  dcrHourGlass,
  dcrSQLWait,
  dcrOther
};
```

File

DB

Description

This is record DB.TDBScreenCursor.

39.4.9 DB.TFieldAttribute Enumeration

TFieldAttribute and TFieldAttributes (see page 7632) indicate various attributes of a field.

Pascal

```
TFieldAttribute = (
  faHiddenCol,
  faReadonly,
  faRequired,
  faLink,
  faUnNamed,
  faFixed
);
```

C++

```
enum TFieldAttribute {
  faHiddenCol,
  faReadonly,
  faRequired,
  faLink,
  faUnNamed,
  faFixed
};
```

File

DB

Description

TFieldAttributes (see page 7632) is a set that describes the attributes of a physical field in an underlying database table. It can contain zero or more of the following TFieldAttribute values:

Value	Meaning
faHiddenCol	Internal flag for hidden columns
faReadonly	Indicates whether the physical field in the underlying table is read-only.
faRequired	Indicates whether a value for the physical field in the underlying table is required.
faLink	Internal flag for nested datasets.
faUnNamed	Internal use only. Indicates a TObjectField (see page 7372) in a nested table that has no name associated with it.
faFixed	Indicates that the field has a fixed size.

39.4.10 DB.TFieldKind Enumeration

TFieldKind indicates what type of information a field component represents.

Pascal

```
TFieldKind = (
  fkData,
  fkCalculated,
  fkLookup,
```

```
    fkInternalCalc,
    fkAggregate
  );
```

C++

```
  enum TFieldKind {
    fkData,
    fkCalculated,
    fkLookup,
    fkInternalCalc,
    fkAggregate
  };
```

File

DB

Description

TFieldKind can take one of the following values:

Value	Description
fkData	Field represents a physical field in a database table
fkCalculated	Field is calculated in an OnCalcFields event handler
fkLookup	Field is a lookup field. (Not implemented for fields on unidirectional datasets)
fkInternalCalc	Field is calculated but values are stored in the dataset.
fkAggregate	Field represents a maintained aggregate in a client dataset.

See Also

TField

39.4.11 DB.TFieldType Enumeration

TFieldType is a set of values that parallel the data types of fields in tables.

Pascal

```
  TFieldType = (
    ftUnknown,
    ftString,
    ftSmallint,
    ftInteger,
    ftWord,
    ftBoolean,
    ftFloat,
    ftCurrency,
    ftBCD,
    ftDate,
    ftTime,
    ftDateTime,
    ftBytes,
    ftVarBytes,
    ftAutoInc,
    ftBlob,
    ftMemo,
    ftGraphic,
    ftFmtMemo,
    ftParadoxOle,
```

```
    ftDBaseOle,
    ftTypedBinary,
    ftCursor,
    ftFixedChar,
    ftWideString,
    ftLargeint,
    ftADT,
    ftArray,
    ftReference,
    ftDataSet,
    ftOraBlob,
    ftOraClob,
    ftVariant,
    ftInterface,
    ftIDispatch,
    ftGuid,
    ftTimeStamp,
    ftFMTBcd,
    ftFixedWideChar,
    ftWideMemo,
    ftOraTimeStamp,
    ftOraInterval
  );
```

C++

```
  enum TFieldType {
    ftUnknown,
    ftString,
    ftSmallint,
    ftInteger,
    ftWord,
    ftBoolean,
    ftFloat,
    ftCurrency,
    ftBCD,
    ftDate,
    ftTime,
    ftDateTime,
    ftBytes,
    ftVarBytes,
    ftAutoInc,
    ftBlob,
    ftMemo,
    ftGraphic,
    ftFmtMemo,
    ftParadoxOle,
    ftDBaseOle,
    ftTypedBinary,
    ftCursor,
    ftFixedChar,
    ftWideString,
    ftLargeint,
    ftADT,
    ftArray,
    ftReference,
    ftDataSet,
    ftOraBlob,
    ftOraClob,
    ftVariant,
    ftInterface,
    ftIDispatch,
    ftGuid,
    ftTimeStamp,
    ftFMTBcd,
    ftFixedWideChar,
    ftWideMemo,
```

```
    ftOraTimeStamp,
    ftOraInterval
  };
```

File

DB

Description

TFieldType type is the set of values for the DataType property field objects, field definition objects, and parameter objects. Classes (see page 2685) in which TFieldType values are used include TField (see page 7115) (and descendants), TFieldDef (see page 7184), TParam (see page 7386), TParameter, and TAggregate. TFieldType values are also used in field-related functions and methods like the Add method of TFieldDefs (see page 7198). The following table describes each value:

Value	Description
ftUnknown	Unknown or undetermined
ftString	Character or string field
ftSmallint	16-bit integer field
ftInteger	32-bit integer field
ftWord	16-bit unsigned integer field
ftBoolean	Boolean field
ftFloat	Floating-point numeric field
ftCurrency	Money field
ftBCD	Binary-Coded Decimal field that can be converted to Currency type without a loss of precision.
ftDate	Date field
ftTime	Time field
ftDateTime	Date and time field
ftBytes	Fixed number of bytes (binary storage)
ftVarBytes	Variable number of bytes (binary storage)
ftAutoInc	Auto-incrementing 32-bit integer counter field
ftBlob	Binary Large OBject field
ftMemo	Text memo field
ftGraphic	Bitmap field
ftFmtMemo	Formatted text memo field
ftParadoxOle	Paradox OLE field
ftDBaseOle	dBASE OLE field
ftTypedBinary	Typed binary field
ftCursor	Output cursor from an Oracle stored procedure (TParam (see page 7386) only)
ftFixedChar	Fixed character field
ftWideString	Wide string field
ftLargeint	Int64 large integer field
ftADT	Abstract Data Type field
ftArray	Array field

ftReference	REF field
ftDataSet	DataSet field
ftOraBlob	BLOB fields in Oracle 8 tables
ftOraClob	CLOB fields in Oracle 8 tables
ftVariant	Data of unknown or undetermined type
ftInterface	References to interfaces (IUnknown)
ftIDispatch	References to IDispatch interfaces
ftGuid	globally unique identifier (GUID) values
ftTimeStamp	Date and time field accessed through dbExpress
ftFMTBcd	Binary-Coded Decimal field that is too large for ftBCD.

39.4.12 DB.TFilterOption Enumeration

TFilterOption and TFilterOptions (see page 7635) influence how filter strings are applied.

Pascal

```
TFilterOption = (
  foCaseInsensitive,
  foNoPartialCompare
);
```

C++

```
enum TFilterOption {
  foCaseInsensitive,
  foNoPartialCompare
};
```

File

DB

Description

TFilterOptions (see page 7635) is a set that specifies how filter expressions are applied to string fields. It can contain zero or more of the following values:

Value	Meaning
foCaseInsensitive	Literal strings in the filter are compared to string-valued fields case-insensitively.
foNoPartialCompare	Asterisks (*) in the filter are treated as literal characters rather than as wildcards. When filter options does not include foNoPartialCompare, strings that end in an asterisk signify a partial string match with the asterisk matching any number of characters.

39.4.13 DB.TGetMode Enumeration

Pascal

```
TGetMode = (
```

```
    gmCurrent,
    gmNext,
    gmPrior
  );
```

C++

```
  enum TGetMode {
    gmCurrent,
    gmNext,
    gmPrior
  };
```

File

DB

Description

This is record DB.TGetMode.

39.4.14 DB.TGetResult Enumeration

Pascal

```
  TGetResult = (
    grOK,
    grBOF,
    grEOF,
    grError
  );
```

C++

```
  enum TGetResult {
    grOK,
    grBOF,
    grEOF,
    grError
  };
```

File

DB

Description

This is record DB.TGetResult.

39.4.15 DB.TGroupPosInd Enumeration

TGroupPosInd and TGroupPosInds (see page 7636) indicate the position of the current record within its group.

Pascal

```
  TGroupPosInd = (
    gbFirst,
    gbMiddle,
    gbLast
  );
```

C++

```
enum TGroupPosInd {
  gbFirst,
  gbMiddle,
  gbLast
};
```

File

DB

Description

TGroupPosInds (see page 7636) is a set that indicates where the current record sits within a group of records. It is a set because if the group contains only one record, the current record is both first and last.

The following table lists the possible TGroupPosInd values:

Value	Meaning
gbFirst	The record is the first record in the group.
gbMiddle	The record is neither the first nor the last in the group.
gbLast	The record is the last record in the group.

39.4.16 DB.TIndexOption Enumeration

Pascal

```
TIndexOption = (
  ixPrimary,
  ixUnique,
  ixDescending,
  ixCaseInsensitive,
  ixExpression,
  ixNonMaintained
);
```

C++

```
enum TIndexOption {
  ixPrimary,
  ixUnique,
  ixDescending,
  ixCaseInsensitive,
  ixExpression,
  ixNonMaintained
};
```

File

DB

Description

This is record DB.TIndexOption.

39.4.17 DB.TLocateOption Enumeration

Pascal

```
TLocateOption = (
  loCaseInsensitive,
  loPartialKey
);
```

C++

```
enum TLocateOption {
  loCaseInsensitive,
  loPartialKey
};
```

File

DB

Description

This is record DB.TLocateOption.

39.4.18 DB.TLookupListEntry Record

Pascal

```
TLookupListEntry = record
  Key: Variant;
  Value: Variant;
end;
```

C++

```
struct TLookupListEntry {
  Variant Key;
  Variant Value;
};
```

File

DB

Description

This is record DB.TLookupListEntry.

39.4.19 DB.TPacketAttribute Record

Pascal

```
TPacketAttribute = record
  Name: string;
  Value: OleVariant;
  IncludeInDelta: Boolean;
end;
```

C++

```
struct TPacketAttribute {
  AnsiString Name;
  OleVariant Value;
  Boolean IncludeInDelta;
};
```

File

DB

Description

This is record DB.TPacketAttribute.

39.4.20 DB.TParamType Enumeration

TParamType and TParamTypes (see page 7637) represent the use of parameters on datasets that represent queries or stored procedures.

Pascal

```
TParamType = (
  ptUnknown,
  ptInput,
  ptOutput,
  ptInputOutput,
  ptResult
);
```

C++

```
enum TParamType {
  ptUnknown,
  ptInput,
  ptOutput,
  ptInputOutput,
  ptResult
};
```

File

DB

Description

TParamType indicates how a query or stored procedure parameter is used. TParamTypes (see page 7637) is a set of parameter types that can be used to filter parameters so that an application need only work with those parameters that are of interest.

The DbClient unit defines a constant, AllParamTypes, of type TParamTypes (see page 7637) that represents any possible parameter type:

const AllParamTypes = [ptUnknown, ptInput, ptOutput, ptInputOutput, ptResult];

#define AllParamTypes TParamTypes (see page 7637)() << ptUnknown << ptInput << ptOutput << ptInputOutput << ptResult)

TParamType includes the following values:

Value	Description
ptUnknown	Unknown or undetermined. Before executing a stored procedure, the application must set parameters of this type to another kind.
ptInput	Used to input a field value. Identifies a parameter used to pass values to a query or stored procedure for processing.
ptOutput	Used to output a field value. Identifies a parameter used by a stored procedure to return values to an application.
ptInputOutput	Used for both input and output.
ptResult	Used as a return value. Identifies a parameter used by a stored procedure to return an error or status value. A stored procedure can only have one parameter of type ptResult.

39.4.21 DB.TProviderFlag Enumeration

TProviderFlag and TProviderFlags (see page 7638) indicate how a provider uses a field value when applying updates.

Pascal

```
TProviderFlag = (
  pfInUpdate,
  pfInWhere,
  pfInKey,
  pfHidden
);
```

C++

```
enum TProviderFlag {
  pfInUpdate,
  pfInWhere,
  pfInKey,
  pfHidden
};
```

File

DB

Description

TProviderFlags (see page 7638) is a set of TProviderFlag values. The following table lists the TProviderFlag values:

Value	Meaning
pfInUpdate	The field can be included in UPDATE statements. (That is, it can be modified)
pfInWhere	The field is included in the WHERE clause when the provider's UpdateMode is upWhereAll or upWhereChanged.
pfInKey	The field is included in the WHERE clause when the provider's UpdateMode is upWhereKeyOnly.
pfHidden	The field is included in data packets to ensure the uniqueness of records so that they can be correctly updated. It can't be seen or used by the receiving client dataset.

39.4.22 DB.TPSCommandType Enumeration

Pascal

```
TPSCommandType = (
  ctUnknown,
  ctQuery,
  ctTable,
  ctStoredProc,
  ctSelect,
  ctInsert,
  ctUpdate,
  ctDelete,
  ctDDL
);
```

C++

```
enum TPSCommandType {
  ctUnknown,
  ctQuery,
  ctTable,
  ctStoredProc,
  ctSelect,
  ctInsert,
  ctUpdate,
  ctDelete,
  ctDDL
};
```

File

DB

Description

This is record DB.TPSCommandType.

39.4.23 DB.TUpdateAction Enumeration

TUpdateAction Indicates how an a dataset should proceed after an event handler has processed an update.

Pascal

```
TUpdateAction = (
  uaFail,
  uaAbort,
  uaSkip,
  uaRetry,
  uaApplied
);
```

C++

```
enum TUpdateAction {
  uaFail,
  uaAbort,
  uaSkip,
  uaRetry,
  uaApplied
};
```

File

DB

Description

TUpdateAction is the type of a parameter on event handlers that intervene when applying cached updates (without using a client dataset and provider). The following table lists the possible values:

Value	Meaning
uaAbort	Abort the update operation without displaying an error message.
uaApplied	The event handler applied the update (not used in error handling event handlers). Free the current record update from the cache.
uaFail	Abort the update operation and display an error message.
uaRetry	Repeat the update operation that originally raised the error condition (not used in OnUpdateRecord event handlers).
uaSkip	Skip updating the current record and leave the unapplied changes in the cache.

39.4.24 DB.TUpdateKind Enumeration

TUpdateKind defines the values for the UpdateKind parameter of OnUpdateError and OnUpdateRecord event handlers.

Pascal

```
TUpdateKind = (
  ukModify,
  ukInsert,
  ukDelete
);
```

C++

```
enum TUpdateKind {
  ukModify,
  ukInsert,
  ukDelete
};
```

File

DB

Description

The TUpdateKind type defines the possible values for the UpdateKind parameter of OnUpdateError and OnUpdateRecord event handlers, as described in the following table:

Value	Description
ukModify	The update to the record is a modification to the record's contents.
ukInsert	The update is the insertion of a new record.
ukDelete	The update is the deletion of a record.

39.4.25 DB.TUpdateMode Enumeration

TUpdateMode indicates how to locate records so that they can be updated.

Pascal

```
TUpdateMode = (
  upWhereAll,
  upWhereChanged,
  upWhereKeyOnly
);
```

C++

```
enum TUpdateMode {
  upWhereAll,
  upWhereChanged,
  upWhereKeyOnly
};
```

File

DB

Description

TUpdateMode is the type of the UpdateMode property. UpdateMode specifies how the records are located when the ApplyUpdates method is called. TUpdateMode includes the following values:

Value	Meaning
upWhereAll	All columns (fields) are used to locate the record.
upWhereChanged	Only key field values and the original value of fields that have changed are used to find the record.
upWhereKeyOnly	Only key fields are used to find the record.

See Also

UpdateMode

ApplyUpdates

39.4.26 DB.TUpdateStatus Enumeration

TUpdateStatus indicates the update status of a record.

Pascal

```
TUpdateStatus = (
  usUnmodified,
  usModified,
  usInserted,
  usDeleted
);
```

C++

```
enum TUpdateStatus {
  usUnmodified,
  usModified,
```

```
    usInserted,
    usDeleted
  };
```

File

DB

Description

TUpdateStatus is the return type of the UpdateStatus method. UpdateStatus indicates what change, if any, is cached for the current record. TUpdateStatus includes the following values:

Value	Description
usUnmodified	The current record has no unapplied updates.
usModified	The current record has unapplied modifications.
usInserted	The current record has been inserted but the insertion was not applied.
usDeleted	The current record represents a deleted record, where the deletion has not yet been applied.

See Also

UpdateStatus

39.5 Types

The following table lists types in this documentation.

Types

Name	Description
Largeint (see page 7626)	This is type DB.Largeint.
PBookmarkFlag (see page 7626)	This is type DB.PBookmarkFlag.
PLookupListEntry (see page 7626)	This is type DB.PLookupListEntry.
PPacketAttribute (see page 7627)	This is type DB.PPacketAttribute.
TBlobByteData (see page 7627)	TBlobByteData stores the contents of a Binary Large Object (BLOB) field.
TBlobData (see page 7627)	This is type DB.TBlobData.
TBlobType (see page 7628)	TBlobType indicates the type of a BLOB field.
TBookmark (see page 7628)	TBookmark identifies a record in a dataset for later navigation.
TBookmarkStr (see page 7629)	TBookmarkStr is a bookmark, represented as a string.
TBufferList (see page 7629)	This is type DB.TBufferList.
TConnectChangeEvent (see page 7630)	This is type DB.TConnectChangeEvent.
TDataChangeEvent (see page 7630)	This is type DB.TDataChangeEvent.
TDataOperation (see page 7630)	This is type DB.TDataOperation.
TDataSetClass (see page 7631)	TDataSetClass is the class type of a TDataSet (see page 6894) descendant.
TDataSetErrorEvent (see page 7631)	This is type DB.TDataSetErrorEvent.
TDataSetNotifyEvent (see page 7631)	This is type DB.TDataSetNotifyEvent.
TDateTimeAlias (see page 7632)	This is type DB.TDateTimeAlias.
TDefUpdateMethod (see page 7632)	This is type DB.TDefUpdateMethod.
TFieldAttributes (see page 7632)	This is type DB.TFieldAttributes.
TFieldChars (see page 7633)	TFieldChars indicates the set of characters that are valid for field.
TFieldClass (see page 7633)	TFieldClass is the class type of a TField (see page 7115) descendant.
TFieldGetTextEvent (see page 7633)	This is type DB.TFieldGetTextEvent.
TFieldKinds (see page 7634)	This is type DB.TFieldKinds.
TFieldNotifyEvent (see page 7634)	This is type DB.TFieldNotifyEvent.
TFieldRef (see page 7634)	This is type DB.TFieldRef.
TFieldSetTextEvent (see page 7635)	This is type DB.TFieldSetTextEvent.
TFilterOptions (see page 7635)	This is type DB.TFilterOptions.
TFilterRecordEvent (see page 7635)	This is type DB.TFilterRecordEvent.
TGroupPosInds (see page 7636)	This is type DB.TGroupPosInds.
TIndexOptions (see page 7636)	TIndexOptions describes the attributes of an index.
TLocateOptions (see page 7637)	TLocateOptions defines the values for the Options parameter of the Locate method of a dataset.
TParamTypes (see page 7637)	This is type DB.TParamTypes.
TProviderFlags (see page 7638)	This is type DB.TProviderFlags.
TResyncMode (see page 7638)	TResyncMode is a set of flags that indicate operations a dataset should perform when resynchronizing data.
TSQLCommandType (see page 7638)	This is type DB.TSQLCommandType.
TUpdateErrorEvent (see page 7639)	This is type DB.TUpdateErrorEvent.
TUpdateRecordEvent (see page 7639)	This is type DB.TUpdateRecordEvent.
TUpdateRecordTypes (see page 7639)	This is type DB.TUpdateRecordTypes.
TUpdateStatusSet (see page 7640)	This is type DB.TUpdateStatusSet.

39.5.1 DB.Largeint Type

Pascal

```
Largeint = Int64;
```

C++

```
Int64 Largeint;
```

File

DB

Description

This is type DB.Largeint.

39.5.2 DB.PBookmarkFlag Type

Pascal

```
PBookmarkFlag = ^TBookmarkFlag;
```

C++

```
^TBookmarkFlag PBookmarkFlag;
```

File

DB

Description

This is type DB.PBookmarkFlag.

39.5.3 DB.PLookupListEntry Type

Pascal

```
PLookupListEntry = ^TLookupListEntry;
```

C++

```
^TLookupListEntry PLookupListEntry;
```

File

DB

Description

This is type DB.PLookupListEntry.

39.5.4 DB.PPacketAttribute Type

Pascal

```
PPacketAttribute = ^TPacketAttribute;
```

C++

```
^TPacketAttribute PPacketAttribute;
```

File

DB

Description

This is type DB.PPacketAttribute.

39.5.5 DB.TBlobByteData Type

TBlobByteData stores the contents of a Binary Large Object (BLOB) field.

Pascal

```
TBlobByteData = array of Byte;
```

C++

```
array of Byte TBlobByteData;
```

File

DB

Description

TBlobByteData is the type for buffers that store information from a BLOB field.

39.5.6 DB.TBlobData Type

Pascal

```
TBlobData = string;
```

C++

```
AnsiString TBlobData;
```

File

DB

Description

This is type DB.TBlobData.

39.5.7 DB.TBlobType Type

TBlobType indicates the type of a BLOB field.

Pascal

```
TBlobType = ftBlob..ftWideMemo;
```

C++

```
ftBlob..ftWideMemo TBlobType;
```

File

DB

Description

TBlobType represents a range of TFieldType (see page 7612) values. These are the possible values of BlobType:

Value	Description
ftBlob	Binary Large Object field
ftMemo	Text memo field
ftGraphic	Bitmap field
ftFmtMemo	Formatted text memo field
ftParadoxOle	Paradox OLE field
ftDBaseOle	dBASE OLE field
ftTypedBinary	Typed binary field
ftOraBlob	BLOB fields in Oracle 8 tables
ftOraClob	CLOB fields in Oracle 8 tables

See Also

TBlobField

TFieldType

39.5.8 DB.TBookmark Type

TBookmark identifies a record in a dataset for later navigation.

Pascal

```
TBookmark = Pointer;
```

C++

```
void * TBookmark;
```

File

DB

Description

TBookmark is the type of the Bookmark parameter in the GetBookmark, GotoBookmark, and FreeBookmark methods of a dataset component.

Note: Different types of datsets can implement bookmarks differently. Do not use a TBookmark from one type of dataset (for example, a client dataset) with methods on another type of dataset (for example, an InterBase Express dataset).

See Also

GetBookmark (see page 6948)

GotoBookmark (see page 6957)

FreeBookmark (see page 6945)

Bookmark (see page 7009)

Marking and Returning to Records

39.5.9 DB.TBookmarkStr Type

TBookmarkStr is a bookmark, represented as a string.

Pascal

```
TBookmarkStr = string;
```

C++

```
AnsiString TBookmarkStr;
```

File

DB

Description

TBookmarkStr is the value of a bookmark in a dataset. Although this is a string type, it is not necessarily a string of text. The string can be any string of bytes. The format depends on the type of dataset.

39.5.10 DB.TBufferList Type

Pascal

```
TBufferList = array of PChar;
```

C++

```
array of PChar TBufferList;
```

File

DB

Description

This is type DB.TBufferList.

39.5.11 DB.TConnectChangeEvent Type

Pascal

```
TConnectChangeEvent = procedure (Sender: TObject; Connecting: Boolean) of object;
```

C++

```
(Sender: TObject; Connecting: Boolean) ( TConnectChangeEvent)();
```

File

DB

Description

This is type DB.TConnectChangeEvent.

39.5.12 DB.TDataChangeEvent Type

Pascal

```
TDataChangeEvent = procedure (Sender: TObject; Field: TField) of object;
```

C++

```
(Sender: TObject; Field: TField) ( TDataChangeEvent)();
```

File

DB

Description

This is type DB.TDataChangeEvent.

39.5.13 DB.TDataOperation Type

Pascal

```
TDataOperation = procedure of object;
```

C++

```
( TDataOperation)();
```

File

DB

Description

This is type DB.TDataOperation.

39.5.14 DB.TDataSetClass Type

TDataSetClass is the class type of a TDataSet (see page 6894) descendant.

Pascal

```
TDataSetClass = class of TDataSet;
```

C++

```
TDataSet TDataSetClass;
```

File

DB

Description

Use TDataSetClass to specify the type of a TDataSet (see page 6894) descendant.

See Also

TDataSet

39.5.15 DB.TDataSetErrorEvent Type

Pascal

```
TDataSetErrorEvent = procedure (DataSet: TDataSet; E: EDatabaseError; var Action: TDataAction)
of object;
```

C++

```
(DataSet: TDataSet; E: EDatabaseError; var Action: TDataAction) ( TDataSetErrorEvent)();
```

File

DB

Description

This is type DB.TDataSetErrorEvent.

39.5.16 DB.TDataSetNotifyEvent Type

Pascal

```
TDataSetNotifyEvent = procedure (DataSet: TDataSet) of object;
```

C++

```
(DataSet: TDataSet) ( TDataSetNotifyEvent)();
```

File

DB

Description

This is type DB.TDataSetNotifyEvent.

39.5.17 DB.TDateTimeAlias Type

Pascal

```
TDateTimeAlias = type TDateTime;
```

C++

```
type TDateTime TDateTimeAlias;
```

File

DB

Description

This is type DB.TDateTimeAlias.

39.5.18 DB.TDefUpdateMethod Type

Pascal

```
TDefUpdateMethod = procedure of object;
```

C++

```
( TDefUpdateMethod)();
```

File

DB

Description

This is type DB.TDefUpdateMethod.

39.5.19 DB.TFieldAttributes Type

Pascal

```
TFieldAttributes = set of TFieldAttribute;
```

C++

```
TFieldAttribute TFieldAttributes;
```

File

DB

Description

This is type DB.TFieldAttributes.

39.5.20 DB.TFieldChars Type

TFieldChars indicates the set of characters that are valid for field.

Pascal

```
TFieldChars = set of Char;
```

C++

```
Char TFieldChars;
```

File

DB

Description

TFieldChars is a set of characters that can be used to create a string that is the value of a field.

39.5.21 DB.TFieldClass Type

TFieldClass is the class type of a TField (see page 7115) descendant.

Pascal

```
TFieldClass = class of TField;
```

C++

```
TField TFieldClass;
```

File

DB

Description

Use TFieldClass to specify the type of a TField (see page 7115) descendant.

See Also

TField

39.5.22 DB.TFieldGetTextEvent Type

Pascal

```
TFieldGetTextEvent = procedure (Sender: TField; var Text: string; DisplayText: Boolean) of
object;
```

C++

```
(Sender: TField; var Text: string; DisplayText: Boolean) ( TFieldGetTextEvent)();
```

File

DB

Description

This is type DB.TFieldGetTextEvent.

39.5.23 DB.TFieldKinds Type

Pascal

```
TFieldKinds = set of TFieldKind;
```

C++

```
TFieldKind TFieldKinds;
```

File

DB

Description

This is type DB.TFieldKinds.

39.5.24 DB.TFieldNotifyEvent Type

Pascal

```
TFieldNotifyEvent = procedure (Sender: TField) of object;
```

C++

```
(Sender: TField) ( TFieldNotifyEvent)();
```

File

DB

Description

This is type DB.TFieldNotifyEvent.

39.5.25 DB.TFieldRef Type

Pascal

```
TFieldRef = ^TField;
```

C++

```
^TField TFieldRef;
```

File

DB

Description

This is type DB.TFieldRef.

39.5.26 DB.TFieldSetTextEvent Type

Pascal

```
TFieldSetTextEvent = procedure (Sender: TField; const Text: string) of object;
```

C++

```
(Sender: TField; const Text: string) ( TFieldSetTextEvent)();
```

File

DB

Description

This is type DB.TFieldSetTextEvent.

39.5.27 DB.TFilterOptions Type

Pascal

```
TFilterOptions = set of TFilterOption;
```

C++

```
TFilterOption TFilterOptions;
```

File

DB

Description

This is type DB.TFilterOptions.

39.5.28 DB.TFilterRecordEvent Type

Pascal

```
TFilterRecordEvent = procedure (DataSet: TDataSet; var Accept: Boolean) of object;
```

C++

```
(DataSet: TDataSet; var Accept: Boolean) ( TFilterRecordEvent)();
```

File

DB

Description

This is type DB.TFilterRecordEvent.

39.5.29 DB.TGroupPosInds Type

Pascal

```
TGroupPosInds = set of TGroupPosInd;
```

C++

```
TGroupPosInd TGroupPosInds;
```

File

DB

Description

This is type DB.TGroupPosInds.

39.5.30 DB.TIndexOptions Type

TIndexOptions describes the attributes of an index.

Pascal

```
TIndexOptions = set of TIndexOption;
```

C++

```
TIndexOption TIndexOptions;
```

File

DB

Description

TIndexOptions is a set of attributes that applies to a specific index. A TIndexOptions value can include zero or more of the following values:

Value	Description
ixPrimary	The index is the primary index of the table. (Not applicable to dBASE tables).
ixUnique	Each value in the index is unique; there are no duplicates. (Not applicable to all table types.)
ixDescending	The index imposes a descending sort order. (Not applicable to all table types.)
ixExpression	The index is based on a dBASE key expression. (Only applicable to dBASE tables).
ixCaseInsensitive	The index sorts records case insensitively. (Not applicable to all table types.)
ixNonMaintained	The index is not automatically updated when the data is edited. (Not applicable to all table types.)

Warning: Attempting to create an index using options that are not applicable to the table type causes AddIndex to raise an exception.

Not all table types support use of all of the above index options. Even within a particular table type, any given option may only be supported in certain versions of a table type. See the BDE Administrator or Database Explorer utility for information on local table levels.

dBASE tables only support primary indexes and true unique indexes when the table level is 7 or higher (concurrent with Visual

dBASE 7). dBASE tables do not support case-insensitive or descending indexes at all. Use the ixExpression constant when the index is based on multiple fields or uses dBASE data manipulation language functions (that is, for expression indexes). The ixExpression constant is only applicable to dBASE tables.

Paradox tables support the ixDescending for secondary indexes if the table level is 7 or higher and the ixUnique constant if the table level is 5 or higher. The ixDescending and ixCaseInsensitive constants are not applicable to primary indexes.

For SQL-based table types, see the vendor-supplied documentation for the particular database system for details on what index options are applicable.

39.5.31 DB.TLocateOptions Type

TLocateOptions defines the values for the Options parameter of the Locate method of a dataset.

Pascal

```
TLocateOptions = set of TLocateOption;
```

C++

```
TLocateOption TLocateOptions;
```

File

DB

Description

TLocateOptions is the type of the Options parameter to a dataset's Locate method. The following table lists the possible values:

Value	Description
loCaseInsensitive	Key fields and key values are matched without regard to case.
loPartialKey	Key values can include only part of the matching key field value; for example, 'HAM' would match both 'HAMM' and 'HAMMER.'

39.5.32 DB.TParamTypes Type

Pascal

```
TParamTypes = set of TParamType;
```

C++

```
TParamType TParamTypes;
```

File

DB

Description

This is type DB.TParamTypes.

39.5.33 DB.TProviderFlags Type

Pascal

```
TProviderFlags = set of TProviderFlag;
```

C++

```
TProviderFlag TProviderFlags;
```

File

DB

Description

This is type DB.TProviderFlags.

39.5.34 DB.TResyncMode Type

TResyncMode is a set of flags that indicate operations a dataset should perform when resynchronizing data.

Pascal

```
TResyncMode = set of (rmExact, rmCenter);
```

C++

```
(rmExact, rmCenter) TResyncMode;
```

File

DB

Description

TResyncMode is a set of zero or more of the following values:

Value	Meaning
rmExact	The dataset should raise an exception when the resync operation is initiated when the cursor is not positioned on a valid record.
rmCenter	The resync operation should position the active record in the center of all buffered records.

39.5.35 DB.TSQLCommandType Type

Pascal

```
TSQLCommandType = ctQuery..ctStoredProc;
```

C++

```
ctQuery..ctStoredProc TSQLCommandType;
```

File

DB

Description

This is type DB.TSQLCommandType.

39.5.36 DB.TUpdateErrorEvent Type

Pascal

```
TUpdateErrorEvent = procedure (DataSet: TDataSet; E: EDatabaseError; UpdateKind: TUpdateKind;
var UpdateAction: TUpdateAction) of object;
```

C++

```
(DataSet: TDataSet; E: EDatabaseError; UpdateKind: TUpdateKind; var UpdateAction:
TUpdateAction) ( TUpdateErrorEvent)();
```

File

DB

Description

This is type DB.TUpdateErrorEvent.

39.5.37 DB.TUpdateRecordEvent Type

Pascal

```
TUpdateRecordEvent = procedure (DataSet: TDataSet; UpdateKind: TUpdateKind; var UpdateAction:
TUpdateAction) of object;
```

C++

```
(DataSet: TDataSet; UpdateKind: TUpdateKind; var UpdateAction: TUpdateAction) (
TUpdateRecordEvent)();
```

File

DB

Description

This is type DB.TUpdateRecordEvent.

39.5.38 DB.TUpdateRecordTypes Type

Pascal

```
TUpdateRecordTypes = set of (rtModified, rtInserted, rtDeleted, rtUnmodified);
```

C++

```
(rtModified, rtInserted, rtDeleted, rtUnmodified) TUpdateRecordTypes;
```

File

DB

Description

This is type DB.TUpdateRecordTypes.

39.5.39 DB.TUpdateStatusSet Type

Pascal

```
TUpdateStatusSet = set of TUpdateStatus;
```

C++

```
TUpdateStatus TUpdateStatusSet;
```

File

DB

Description

This is type DB.TUpdateStatusSet.

39.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
DBApplication (see page 7641)	This is variable DB.DBApplication.
DBScreen (see page 7641)	This is variable DB.DBScreen.
DefaultFieldClasses (see page 7642)	This is variable DB.DefaultFieldClasses.
LoginDialogExProc (see page 7642)	This is variable DB.LoginDialogExProc.
LoginDialogProc (see page 7642)	This is variable DB.LoginDialogProc.
PasswordDialog (see page 7643)	Displays a dialog box that prompts the user for the password of a local password-protected table.
RemoteLoginDialogProc (see page 7643)	This is variable DB.RemoteLoginDialogProc.
ScreenCursorProc (see page 7644)	This is variable DB.ScreenCursorProc.

39.6.1 DB.DBApplication Variable

Pascal

```
DBApplication: IDBApplication;
```

C++

```
IDBApplication DBApplication;
```

File

DB

Description

This is variable DB.DBApplication.

39.6.2 DB.DBScreen Variable

Pascal

```
DBScreen: IDBScreen;
```

C++

```
IDBScreen DBScreen;
```

File

DB

Description

This is variable DB.DBScreen.

39.6.3 DB.DefaultFieldClasses Variable

Pascal

```
DefaultFieldClasses: array[TFieldType] of TFieldClass = ( nil, TStringField, TSmallintField,
TIntegerField, TWordField, TBooleanField, TFloatField, TCurrencyField, TBCDField, TDateField,
TTimeField, TDateTimeField, TBytesField, TVarBytesField, TAutoIncField, TBlobField,
TMemoField, TGraphicField, TBlobField, TBlobField, TBlobField, TBlobField, nil, TStringField,
TWideStringField, TLargeIntField, TADTField, TArrayField, TReferenceField, TDataSetField,
TBlobField, TMemoField, TVariantField, TInterfaceField, TIDispatchField, TGuidField,
TSQLTimeStampField, TFMTBcdField, TWideStringField, TWideMemoField, TSQLTimeStampField,
TStringField);
```

C++

```
array[TFieldType] of TFieldClass DefaultFieldClasses = ( nil, TStringField, TSmallintField,
TIntegerField, TWordField, TBooleanField, TFloatField, TCurrencyField, TBCDField, TDateField,
TTimeField, TDateTimeField, TBytesField, TVarBytesField, TAutoIncField, TBlobField,
TMemoField, TGraphicField, TBlobField, TBlobField, TBlobField, TBlobField, nil, TStringField,
TWideStringField, TLargeIntField, TADTField, TArrayField, TReferenceField, TDataSetField,
TBlobField, TMemoField, TVariantField, TInterfaceField, TIDispatchField, TGuidField,
TSQLTimeStampField, TFMTBcdField, TWideStringField, TWideMemoField, TSQLTimeStampField,
TStringField);
```

File

DB

Description

This is variable DB.DefaultFieldClasses.

39.6.4 DB.LoginDialogExProc Variable

Pascal

```
LoginDialogExProc: function (const ADatabaseName: string; var AUserName, APassword: string;
NameReadOnly: Boolean): Boolean;
```

C++

```
function (const ADatabaseName: string; var AUserName, APassword: string; NameReadOnly:
Boolean): Boolean LoginDialogExProc;
```

File

DB

Description

This is variable DB.LoginDialogExProc.

39.6.5 DB.LoginDialogProc Variable

Pascal

```
LoginDialogProc: function (const ADatabaseName: string; var AUserName, APassword: string):
```

```
Boolean;
```

C++

```
function (const ADatabaseName: string; var AUserName, APassword: string): Boolean
LoginDialogProc;
```

File

DB

Description

This is variable DB.LoginDialogProc.

39.6.6 DB.PasswordDialog Variable

Displays a dialog box that prompts the user for the password of a local password-protected table.

Pascal

```
PasswordDialog: function (const ASession: IDBSession): Boolean;
```

C++

```
function (const ASession: IDBSession): Boolean PasswordDialog;
```

File

DB

Description

Use PasswordDialog to allow the user to enter a password for a password-protected Paradox or dBase table.

ASession is the interface of a component that represents the session onto which the user logs in. Typically, this is the interface of a TSession component.

If the user clicks OK in the dialog box, PasswordDialog returns true. Otherwise it returns false.

Note: When an application opens a Paradox or dBASE table that requires a password, the user is prompted to enter a password unless the Session already has a valid password for that table.

See Also

TSession

39.6.7 DB.RemoteLoginDialogProc Variable

Pascal

```
RemoteLoginDialogProc: function (var AUserName, APassword: string): Boolean;
```

C++

```
function (var AUserName, APassword: string): Boolean RemoteLoginDialogProc;
```

File

DB

Description

This is variable DB.RemoteLoginDialogProc.

39.6.8 DB.ScreenCursorProc Variable

Pascal

```
ScreenCursorProc: procedure (const CurIndex : integer);
```

C++

```
procedure (const CurIndex : integer) ScreenCursorProc;
```

File

DB

Description

This is variable DB.ScreenCursorProc.

39.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
dsEditModes (see page 7645)	This is constant DB.dsEditModes.
dsMaxStringSize (see page 7645)	This is constant DB.dsMaxStringSize.
dsWriteModes (see page 7646)	This is constant DB.dsWriteModes.
FieldTypeNames (see page 7646)	This is constant DB.FieldTypeNames.
FieldTypeVarMap (see page 7646)	This is constant DB.FieldTypeVarMap.
ftFixedSizeTypes (see page 7647)	This is constant DB.ftFixedSizeTypes.
ftNonTextTypes (see page 7647)	This is constant DB.ftNonTextTypes.
ObjectFieldTypes (see page 7648)	This is constant DB.ObjectFieldTypes.
RegisterFieldsProc (see page 7648)	This is constant DB.RegisterFieldsProc.

39.7.1 DB.dsEditModes Constant

Pascal

```
dsEditModes = [dsEdit, dsInsert, dsSetKey];
```

C++

```
dsEditModes = [dsEdit, dsInsert, dsSetKey];
```

File

DB

Description

This is constant DB.dsEditModes.

39.7.2 DB.dsMaxStringSize Constant

Pascal

```
dsMaxStringSize = 8192;
```

C++

```
dsMaxStringSize = 8192;
```

File

DB

Description

This is constant DB.dsMaxStringSize.

39.7.3 DB.dsWriteModes Constant

Pascal

```
dsWriteModes = [dsEdit, dsInsert, dsSetKey, dsCalcFields, dsFilter, dsNewValue,
dsInternalCalc];
```

C++

```
dsWriteModes = [dsEdit, dsInsert, dsSetKey, dsCalcFields, dsFilter, dsNewValue,
dsInternalCalc];
```

File

DB

Description

This is constant DB.dsWriteModes.

39.7.4 DB.FieldTypeNames Constant

Pascal

```
FieldTypeNames: array[TFieldType] of string = ( 'Unknown', 'String', 'SmallInt', 'Integer',
'Word', 'Boolean', 'Float', 'Currency', 'BCD', 'Date', 'Time', 'DateTime', 'Bytes',
'VarBytes', 'AutoInc', 'Blob', 'Memo', 'Graphic', 'FmtMemo', 'ParadoxOle', 'dBaseOle',
'TypedBinary', 'Cursor', 'FixedChar', 'WideString', 'LargeInt', 'ADT', 'Array', 'Reference',
'DataSet', 'HugeBlob', 'HugeClob', 'Variant', 'Interface', 'Dispatch', 'Guid', 'SQLTimeStamp',
'FMTBcdField', 'FixedWideChar', 'WideMemo', 'SQLTimeStamp', 'String');
```

C++

```
array[TFieldType] of string FieldTypeNames = ( 'Unknown', 'String', 'SmallInt', 'Integer',
'Word', 'Boolean', 'Float', 'Currency', 'BCD', 'Date', 'Time', 'DateTime', 'Bytes',
'VarBytes', 'AutoInc', 'Blob', 'Memo', 'Graphic', 'FmtMemo', 'ParadoxOle', 'dBaseOle',
'TypedBinary', 'Cursor', 'FixedChar', 'WideString', 'LargeInt', 'ADT', 'Array', 'Reference',
'DataSet', 'HugeBlob', 'HugeClob', 'Variant', 'Interface', 'Dispatch', 'Guid', 'SQLTimeStamp',
'FMTBcdField', 'FixedWideChar', 'WideMemo', 'SQLTimeStamp', 'String');
```

File

DB

Description

This is constant DB.FieldTypeNames.

39.7.5 DB.FieldTypeVarMap Constant

Pascal

```
FieldTypeVarMap: array[TFieldType] of Word = ( varEmpty, varString, varInteger, varInteger,
varInteger, varBoolean, varDouble, varCurrency, varCurrency, varDate, varDate, varDate,
varEmpty, varEmpty, varInteger, varEmpty, varString, varEmpty, varEmpty, varEmpty, varEmpty,
varEmpty, varEmpty, varString, varOleStr, varEmpty, varEmpty, varEmpty, varEmpty, varEmpty,
varEmpty, varEmpty, varVariant, varUnknown, varDispatch, varString, varEmpty, varEmpty,
```

```
varOleStr, varOleStr, varEmpty, varString);
```

C++

```
array[TFieldType] of Word FieldTypeVarMap = ( varEmpty, varString, varInteger, varInteger,
varInteger, varBoolean, varDouble, varCurrency, varCurrency, varDate, varDate, varDate,
varEmpty, varEmpty, varInteger, varEmpty, varString, varEmpty, varEmpty, varEmpty, varEmpty,
varEmpty, varEmpty, varString, varOleStr, varEmpty, varEmpty, varEmpty, varEmpty, varEmpty,
varEmpty, varEmpty, varVariant, varUnknown, varDispatch, varString, varEmpty, varEmpty,
varOleStr, varOleStr, varEmpty, varString);
```

File

DB

Description

This is constant DB.FieldTypeVarMap.

39.7.6 DB.ftFixedSizeTypes Constant

Pascal

```
ftFixedSizeTypes = [ftSmallint, ftInteger, ftWord, ftBoolean, ftFloat, ftCurrency, ftDate,
ftTime, ftDateTime, ftAutoInc, ftLargeint, ftTimeStamp];
```

C++

```
ftFixedSizeTypes = [ftSmallint, ftInteger, ftWord, ftBoolean, ftFloat, ftCurrency, ftDate,
ftTime, ftDateTime, ftAutoInc, ftLargeint, ftTimeStamp];
```

File

DB

Description

This is constant DB.ftFixedSizeTypes.

39.7.7 DB.ftNonTextTypes Constant

Pascal

```
ftNonTextTypes = [ftBytes, ftVarBytes, ftBlob, ftMemo, ftGraphic, ftFmtMemo, ftParadoxOle,
ftDBaseOle, ftTypedBinary, ftCursor, ftADT, ftArray, ftReference, ftDataSet];
```

C++

```
ftNonTextTypes = [ftBytes, ftVarBytes, ftBlob, ftMemo, ftGraphic, ftFmtMemo, ftParadoxOle,
ftDBaseOle, ftTypedBinary, ftCursor, ftADT, ftArray, ftReference, ftDataSet];
```

File

DB

Description

This is constant DB.ftNonTextTypes.

39.7.8 DB.ObjectFieldTypes Constant

Pascal

```
ObjectFieldTypes = [ftADT, ftArray, ftReference, ftDataSet];
```

C++

```
ObjectFieldTypes = [ftADT, ftArray, ftReference, ftDataSet];
```

File

DB

Description

This is constant DB.ObjectFieldTypes.

39.7.9 DB.RegisterFieldsProc Constant

Pascal

```
RegisterFieldsProc: procedure(const FieldClassess: array of TFieldClass) = nil;
```

C++

```
procedure(const FieldClassess: array of TFieldClass) RegisterFieldsProc = nil;
```

File

DB

Description

This is constant DB.RegisterFieldsProc.

40 DBActns Namespace

This is namespace DBActns.

Topics

Name	Description
Classes (see page 7650)	The following table lists classes in this documentation.

Classes

	Name	Description
	TDataSetAction (see page 7650)	TDataSetAction is the base class for action objects that act on dataset components.
	TDataSetCancel (see page 7657)	TDataSetCancel cancels the edits to the current record, restores record display to its condition prior to editing, and takes the dataset out of edit mode.
	TDataSetDelete (see page 7664)	TDataSetDelete deletes the current record, making the next record current.
	TDataSetEdit (see page 7671)	TDataSetEdit puts a target dataset into edit mode.
	TDataSetFirst (see page 7677)	TDataSetFirst moves a target dataset to its first record.
	TDataSetInsert (see page 7684)	TDataSetInsert inserts a new record into a target dataset.
	TDataSetLast (see page 7691)	TDataSetLast moves a target dataset to its last record.
	TDataSetNext (see page 7698)	TDataSetNext moves a target dataset to its next record.
	TDataSetPost (see page 7704)	TDataSetPost posts the edits in the current record of a target dataset.
	TDataSetPrior (see page 7711)	TDataSetPrior moves a target dataset to the previous record.
	TDataSetRefresh (see page 7718)	TDataSetRefresh refreshes the records in a target dataset.

40.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TDataSetAction (see page 7650)	TDataSetAction is the base class for action objects that act on dataset components.
	TDataSetCancel (see page 7657)	TDataSetCancel cancels the edits to the current record, restores record display to its condition prior to editing, and takes the dataset out of edit mode.
	TDataSetDelete (see page 7664)	TDataSetDelete deletes the current record, making the next record current.
	TDataSetEdit (see page 7671)	TDataSetEdit puts a target dataset into edit mode.
	TDataSetFirst (see page 7677)	TDataSetFirst moves a target dataset to its first record.
	TDataSetInsert (see page 7684)	TDataSetInsert inserts a new record into a target dataset.
	TDataSetLast (see page 7691)	TDataSetLast moves a target dataset to its last record.
	TDataSetNext (see page 7698)	TDataSetNext moves a target dataset to its next record.
	TDataSetPost (see page 7704)	TDataSetPost posts the edits in the current record of a target dataset.
	TDataSetPrior (see page 7711)	TDataSetPrior moves a target dataset to the previous record.
	TDataSetRefresh (see page 7718)	TDataSetRefresh refreshes the records in a target dataset.

40.1.1 TDataSetAction Class

TDataSetAction is the base class for action objects that act on dataset components.

Class Hierarchy

Pascal

```
TDataSetAction = class(TAction);
```

C++

```
class TDataSetAction : public TAction;
```

File

DBActns

Description

TDataSetAction implements the common behavior for actions that work with datasets. Each descendant class performs a specific database function. Do not use TDataSetAction directly in an application. Instead, use a TDataSetAction descendant such as TDataSetCancel (see page 7657), TDataSetDelete (see page 7664), TDataSetEdit (see page 7671), TDataSetFirst (see page 7677), TDataSetInsert (see page 7684), TDataSetLast (see page 7691), TDataSetNext (see page 7698), TDataSetPost (see page 7704), TDataSetPrior (see page 7711), or TDataSetRefresh (see page 7718).

TDataSetAction ensures that the action is only enabled to work with a TDataSource component as its target. This TDataSource is first located as the data source that connects an active data-aware control to its source dataset. TDataSetAction saves this data source as the DataSource (see page 7657) property, which descendants then use to perform their specific database

functions.

See Also

TDataSetCancel (see page 7657)

TDataSetDelete (see page 7664)

TDataSetEdit (see page 7671)

TDataSetFirst (see page 7677)

TDataSetInsert (see page 7684)

TDataSEtLast

TDataSetNext (see page 7698)

TDataSetPost (see page 7704)

TDataSetPrior (see page 7711)

TDataSetRefresh (see page 7718)

TDataSource (see page 7070)

40.1.1.1 TDataSetAction Members

The following tables list the members exposed by TDataSetAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDataSet (see page 7655)	This is GetDataSet, a member of class TDataSetAction.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 7656)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 7656)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DataSource (see page 7657)	Provides access to the target of the action.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.

	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

40.1.1.2 TDataSetAction Methods

The methods of the TDataSetAction class are listed here.

Protected Methods

	Name	Description
	GetDataSet (see page 7655)	This is GetDataSet, a member of class TDataSetAction.
	Notification (see page 7656)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	HandlesTarget (see page 7656)	Indicates whether the action can execute its function.

40.1.1.2.1 TDataSetAction.GetDataSet Method

Pascal

```
function GetDataSet(Target: TObject): TDataSet; virtual;
```

C++

```
virtual __fastcall TDataSet GetDataSet(TObject * Target);
```

Description

This is GetDataSet, a member of class TDataSetAction.

40.1.1.2.2 TDataSetAction.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the current target object.

HandlesTarget checks that the target of is a TDataSource component with a DataSet property that is not nil (Delphi) or NULL (C++). If so, it returns true. If not, it returns false.

See Also

ExecuteTarget (see page 2712)

TDataSource (see page 7070)

DataSet (see page 7078)

40.1.1.2.3 TDataSetAction.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TBasicAction overrides the Notification method to check whether the component that is the value of the ActionComponent property is about to be freed. If so, it sets the ActionComponent property to nil (Delphi) or NULL (C++).

See Also

ActionComponent (see page 2715)

40.1.1.3 TDataSetAction Properties

The properties of the TDataSetAction class are listed here.

Public Properties

	Name	Description
	DataSource (see page 7657)	Provides access to the target of the action.

40.1.1.3.1 TDataSetAction.DataSource Property

Provides access to the target of the action.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource is the data source that represents the dataset on which the action performs its function. You can set this property to indicate the dataset that is the target of the action.

If you do not explicitly set the DataSource property, it is obtained from an active data-aware control when the action fires.

See Also

DataSet (see page 7078)

40.1.2 TDataSetCancel Class

TDataSetCancel cancels the edits to the current record, restores record display to its condition prior to editing, and takes the dataset out of edit mode.

Class Hierarchy

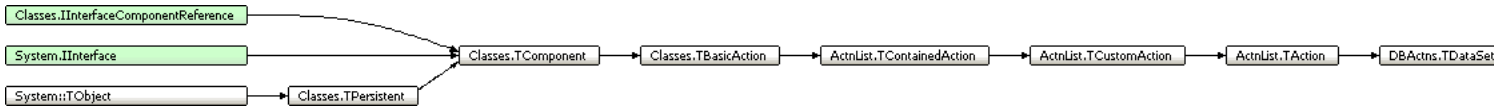

Pascal

```
TDataSetCancel = class(TDataSetAction);
```

C++

```
class TDataSetCancel : public TDataSetAction;
```

File

DBActns

Description

Add TDataSetCancel to an action list to allow users to cancel edits made to a dataset in the application. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to cancel edits, restore record display, and take the associated dataset out of edit mode.

TDataSetCancel is designed to work with a TDataSource component as its target. You can explicitly provide this target by setting the DataSource (see page 7663) property, or you can allow the action to infer the data source from an active data-aware control. TDataSetCancel automatically disables itself if it does not have a target data source attached to an active dataset that is in edit mode.

See Also

TDataSetDelete (see page 7664)

TDataSetEdit (see page 7671)

TDataSetFirst (see page 7677)

TDataSetInsert (see page 7684)

TDataSEtLast

TDataSetNext (see page 7698)

TDataSetPost (see page 7704)

TDataSetPrior (see page 7711)

TDataSetRefresh (see page 7718)

TDataSource (see page 7070)

40.1.2.1 TDataSetCancel Members

The following tables list the members exposed by TDataSetCancel.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDataSet (see page 7655)	This is GetDataSet, a member of class TDataSetAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.

	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7656)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.

	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 7662)	Cancels unposted edits and takes the dataset out of edit mode.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 7656)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 7663)	Enables or disables the action, depending on the state of the target dataset.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.

	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	DataSource (see page 7663)	Provides access to the target of the action.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

40.1.2.2 TDataSetCancel Methods

The methods of the TDataSetCancel class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 7662)	Cancels unposted edits and takes the dataset out of edit mode.
	UpdateTarget (see page 7663)	Enables or disables the action, depending on the state of the target dataset.

40.1.2.2.1 TDataSetCancel.ExecuteTarget Method

Cancels unposted edits and takes the dataset out of edit mode.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for

example, when the user selects a menu item or presses a tool button that is linked to this action).

ExecuteTarget checks that Target is a data source component that has an associated dataset. If the DataSource (see page 7663) property is set, Target must be the same data source.

If Target is a valid data source, ExecuteTarget calls the associated dataset's Cancel method to back out any unposted edits and take the dataset out of edit mode.

See Also

Cancel (see page 6916)

DataSource (see page 7657)

DataSet (see page 7078)

40.1.2.2.2 TDataSetCancel.UpdateTarget Method

Enables or disables the action, depending on the state of the target dataset.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TDataSetCancel (see page 7657), UpdateTarget checks whether the current target object (specified by the Target parameter) is a data source whose dataset is in edit mode. If so, it sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

40.1.2.3 TDataSetCancel Properties

The properties of the TDataSetCancel class are listed here.

Published Properties

	Name	Description
	DataSource (see page 7663)	Provides access to the target of the action.

40.1.2.3.1 TDataSetCancel.DataSource Property

Provides access to the target of the action.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource is the data source that represents the dataset on which the action performs its function. You can set this property to indicate the dataset that is the target of the action.

If you do not explicitly set the DataSource property, it is obtained from an active data-aware control when the action fires.

See Also

DataSet (see page 7078)

40.1.3 TDataSetDelete Class

TDataSetDelete deletes the current record, making the next record current.

Class Hierarchy

Pascal

```
TDataSetDelete = class(TDataSetAction);
```

C++

```
class TDataSetDelete : public TDataSetAction;
```

File

DBActns

Description

Add TDataSetDelete to an action list to allow users to delete records from a dataset. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to delete the current record of a dataset.

TDataSetDelete is designed to work with a TDataSource component as its target. You can explicitly provide this target by setting the DataSource (see page 7670) property, or you can allow the action to infer the data source from an active data-aware control. TDataSetDelete automatically disables itself if it does not have a target data source attached to an active, nonempty dataset that permits edits.

See Also

TDataSetCancel (see page 7657)

TDataSetEdit (see page 7671)

TDataSetFirst (see page 7677)

TDataSetInsert (see page 7684)

TDataSEtLast

TDataSetNext (see page 7698)

TDataSetPost (see page 7704)

TDataSetPrior (see page 7711)

TDataSetRefresh (see page 7718)

TDataSource (see page 7070)

40.1.3.1 TDataSetDelete Members

The following tables list the members exposed by TDataSetDelete.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDataSet (see page 7655)	This is GetDataSet, a member of class TDataSetAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 7656)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 7669)	Deletes the current record in the target dataset.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 7656)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)

V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 7670)	Enables or disables the action, depending on the state of the target dataset.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.

	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	DataSource (see page 7670)	Provides access to the target of the action.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

40.1.3.2 TDataSetDelete Methods

The methods of the TDataSetDelete class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 7669)	Deletes the current record in the target dataset.
	UpdateTarget (see page 7670)	Enables or disables the action, depending on the state of the target dataset.

40.1.3.2.1 TDataSetDelete.ExecuteTarget Method

Deletes the current record in the target dataset.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

ExecuteTarget checks that Target is a data source component that has an associated dataset. If the DataSource (see page 7670) property is set, Target must be the same data source.

If Target is a valid data source, ExecuteTarget calls the associated dataset's Delete method to delete the current record and make the following record current.

See Also

Delete (see page 6924)

DataSource (see page 7657)

DataSet (see page 7078)

40.1.3.2.2 TDataSetDelete.UpdateTarget Method

Enables or disables the action, depending on the state of the target dataset.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TDataSetDelete (see page 7664), UpdateTarget checks whether the current target object (specified by the Target parameter) is a data source whose dataset is not empty and that permits edits. If so, it sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

40.1.3.3 TDataSetDelete Properties

The properties of the TDataSetDelete class are listed here.

Published Properties

	Name	Description
	DataSource (see page 7670)	Provides access to the target of the action.

40.1.3.3.1 TDataSetDelete.DataSource Property

Provides access to the target of the action.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource is the data source that represents the dataset on which the action performs its function. You can set this property to indicate the dataset that is the target of the action.

If you do not explicitly set the DataSource property, it is obtained from an active data-aware control when the action fires.

See Also

DataSet (see page 7078)

40.1.4 TDataSetEdit Class

TDataSetEdit puts a target dataset into edit mode.

Class Hierarchy

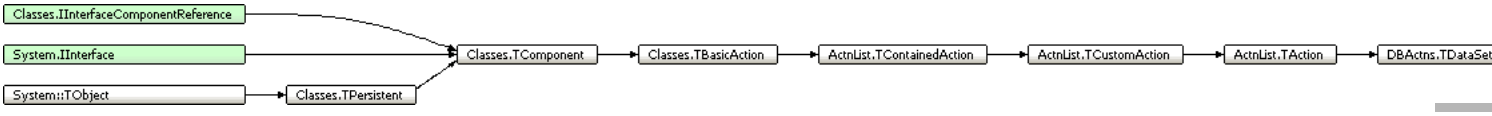

Pascal

```
TDataSetEdit = class(TDataSetAction);
```

C++

```
class TDataSetEdit : public TDataSetAction;
```

File

DBActns

Description

Add TDataSetEdit to an action list to allow users to put a dataset into edit mode. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to put a dataset into edit mode if it can.

TDataSetEdit is designed to work with a TDataSource component as its target. You can explicitly provide this target by setting the DataSource (see page 7677) property, or you can allow the action to infer the data source from an active data-aware control. TDataSetEdit automatically disables itself if it does not have a target data source attached to a dataset that permits edits.

See Also

TDataSetCancel (see page 7657)

TDataSetDelete (see page 7664)

TDataSetFirst (see page 7677)

TDataSetInsert (see page 7684)

TDataSEtLast

TDataSetNext (see page 7698)

TDataSetPost (see page 7704)

TDataSetPrior (see page 7711)

TDataSetRefresh (see page 7718)

TDataSource (see page 7070)

40.1.4.1 TDataSetEdit Members

The following tables list the members exposed by TDataSetEdit.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDataSet (see page 7655)	This is GetDataSet, a member of class TDataSetAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 7656)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.

	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 7676)	Puts the target dataset into edit mode.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 7656)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
	Update (see page 295)	Indicates whether the action updates itself.
	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

V	UpdateTarget (see page 7676)	Enables or disables the action, depending on the state of the target dataset.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	DataSource (see page 7677)	Provides access to the target of the action.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.

	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

40.1.4.2 TDataSetEdit Methods

The methods of the TDataSetEdit class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 7676)	Puts the target dataset into edit mode.
	UpdateTarget (see page 7676)	Enables or disables the action, depending on the state of the target dataset.

40.1.4.2.1 TDataSetEdit.ExecuteTarget Method

Puts the target dataset into edit mode.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

ExecuteTarget checks that Target is a data source component that has an associated dataset. If the DataSource (see page 7677) property is set, Target must be the same data source.

If Target is a valid data source, ExecuteTarget calls the associated dataset's Edit method to put the dataset into edit mode.

See Also

Edit (see page 6932)

DataSource (see page 7657)

DataSet (see page 7078)

40.1.4.2.2 TDataSetEdit.UpdateTarget Method

Enables or disables the action, depending on the state of the target dataset.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TDataSetEdit (see page 7671), UpdateTarget checks whether the current target object (specified by the Target parameter) is a data source whose dataset is permits edits and which is not already in edit mode. If so, it sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

40.1.4.3 TDataSetEdit Properties

The properties of the TDataSetEdit class are listed here.

Published Properties

	Name	Description
	DataSource (see page 7677)	Provides access to the target of the action.

40.1.4.3.1 TDataSetEdit.DataSource Property

Provides access to the target of the action.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource is the data source that represents the dataset on which the action performs its function. You can set this property to indicate the dataset that is the target of the action.

If you do not explicitly set the DataSource property, it is obtained from an active data-aware control when the action fires.

See Also

DataSet (see page 7078)

40.1.5 TDataSetFirst Class

TDataSetFirst moves a target dataset to its first record.

Class Hierarchy

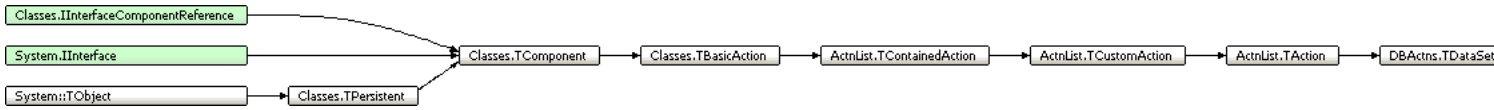

Pascal

```
TDataSetFirst = class(TDataSetAction);
```

C++

```
class TDataSetFirst : public TDataSetAction;
```

File

DBActns

Description

Add TDataSetFirst to an action list to allow users to navigate to the first record in a dataset. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the target dataset to move to the first record.

TDataSetFirst is designed to work with a TDataSource component as its target. You can explicitly provide this target by setting the DataSource (see page 7684) property, or you can allow the action to infer the data source from an active data-aware control. TDataSetFirst automatically disables itself if it does not have a target data source attached to an active dataset that is not already on the first record.

See Also

TDataSetCancel (see page 7657)

TDataSetDelete (see page 7664)

TDataSetEdit (see page 7671)

TDataSetInsert (see page 7684)

TDataSEtLast

TDataSetNext (see page 7698)

TDataSetPost (see page 7704)

TDataSetPrior (see page 7711)

TDataSetRefresh (see page 7718)

TDataSource (see page 7070)

TDataSEt

40.1.5.1 TDataSetFirst Members

The following tables list the members exposed by TDataSetFirst.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.

	AssignTo (see page 303)	Copies the properties of this action to a destination object.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDataSet (see page 7655)	This is GetDataSet, a member of class TDataSetAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7656)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 7683)	Navigates to the first record of the target dataset.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 7656)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 7683)	Enables or disables the action, depending on the state of the target dataset.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	DataSource (see page 7684)	Provides access to the target of the action.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

40.1.5.2 TDataSetFirst Methods

The methods of the TDataSetFirst class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 7683)	Navigates to the first record of the target dataset.
	UpdateTarget (see page 7683)	Enables or disables the action, depending on the state of the target dataset.

40.1.5.2.1 TDataSetFirst.ExecuteTarget Method

Navigates to the first record of the target dataset.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

ExecuteTarget checks that Target is a data source component that has an associated dataset. If the DataSource (see page 7684) property is set, Target must be the same data source.

If Target is a valid data source, ExecuteTarget calls the associated dataset's First method to move to the first record.

See Also

First (see page 6942)

DataSource (see page 7657)

DataSet (see page 7078)

40.1.5.2.2 TDataSetFirst.UpdateTarget Method

Enables or disables the action, depending on the state of the target dataset.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TDataSetFirst (see page 7677), UpdateTarget checks whether the current target object (specified by the Target parameter) is a data source whose dataset is active and which is not already on the first record. If so, it sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

40.1.5.3 TDataSetFirst Properties

The properties of the TDataSetFirst class are listed here.

Published Properties

	Name	Description
	DataSource (see page 7684)	Provides access to the target of the action.

40.1.5.3.1 TDataSetFirst.DataSource Property

Provides access to the target of the action.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource is the data source that represents the dataset on which the action performs its function. You can set this property to indicate the dataset that is the target of the action.

If you do not explicitly set the DataSource property, it is obtained from an active data-aware control when the action fires.

See Also

DataSet (see page 7078)

40.1.6 TDataSetInsert Class

TDataSetInsert inserts a new record into a target dataset.

Class Hierarchy

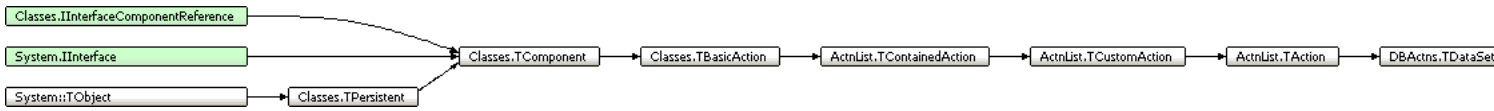

Pascal

```
TDataSetInsert = class(TDataSetAction);
```

C++

```
class TDataSetInsert : public TDataSetAction;
```

File

DBActns

Description

Add TDataSetInsert to an action list to allow users to insert new records in a dataset. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the target dataset to insert a new record above the current one and go into edit mode.

TDataSetInsert is designed to work with a TDataSource component as its target. You can explicitly provide this target by setting the DataSource (see page 7690) property, or you can allow the action to infer the data source from an active data-aware control. TDataSetInsert automatically disables itself if it does not have a target data source attached to an active dataset that permits edits.

See Also

TDataSetCancel (see page 7657)

TDataSetDelete (see page 7664)

TDataSetEdit (see page 7671)

TDataSetInsert

TDataSetFirst (see page 7677)

TDataSetNext (see page 7698)

TDataSetPost (see page 7704)

TDataSetPrior (see page 7711)

TDataSetRefresh (see page 7718)

TDataSource (see page 7070)

40.1.6.1 TDataSetInsert Members

The following tables list the members exposed by TDataSetInsert.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.

	GetDataSet (see page 7655)	This is GetDataSet, a member of class TDataSetAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7656)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 7689)	Inserts a new record above the current one in the target dataset.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.

	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 7656)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 7690)	Enables or disables the action, depending on the state of the target dataset.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.

	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	DataSource (see page 7690)	Provides access to the target of the action.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

40.1.6.2 TDataSetInsert Methods

The methods of the TDataSetInsert class are listed here.

Public Methods

	Name	Description
V	ExecuteTarget (see page 7689)	Inserts a new record above the current one in the target dataset.
V	UpdateTarget (see page 7690)	Enables or disables the action, depending on the state of the target dataset.

40.1.6.2.1 TDataSetInsert.ExecuteTarget Method

Inserts a new record above the current one in the target dataset.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

ExecuteTarget checks that Target is a data source component that has an associated dataset. If the DataSource (see page 7690) property is set, Target must be the same data source.

If Target is a valid data source, ExecuteTarget calls the associated dataset's Insert method to insert a new, empty, record above the current record and put the dataset in edit mode.

See Also

Insert (see page 6959)

DataSource (see page 7657)

DataSet (see page 7078)

40.1.6.2.2 TDataSetInsert.UpdateTarget Method

Enables or disables the action, depending on the state of the target dataset.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TDataSetInsert (see page 7684), UpdateTarget checks whether the current target object (specified by the Target parameter) is a data source whose dataset is active and which permits edits. If so, it sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

40.1.6.3 TDataSetInsert Properties

The properties of the TDataSetInsert class are listed here.

Published Properties

	Name	Description
	DataSource (see page 7690)	Provides access to the target of the action.

40.1.6.3.1 TDataSetInsert.DataSource Property

Provides access to the target of the action.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource is the data source that represents the dataset on which the action performs its function. You can set this property to indicate the dataset that is the target of the action.

If you do not explicitly set the DataSource property, it is obtained from an active data-aware control when the action fires.

See Also

DataSet (see page 7078)

40.1.7 TDataSetLast Class

TDataSetLast moves a target dataset to its last record.

Class Hierarchy

Pascal

```
TDataSetLast = class(TDataSetAction);
```

C++

```
class TDataSetLast : public TDataSetAction;
```

File

DBActns

Description

Add TDataSetLast to an action list to allow users to navigate to the last record in a dataset. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the target dataset to move to the last record.

TDataSetLast is designed to work with a TDataSource component as its target. You can explicitly provide this target by setting the DataSource (see page 7697) property, or you can allow the action to infer the data source from an active data-aware control. If you set DataSource (see page 7697), do not set it to a data source whose dataset is unidirectional. TDataSetLast automatically disables itself if it does not have a target data source attached to an active dataset that is not already on the last record.

See Also

TDataSetCancel (see page 7657)

TDataSetDelete (see page 7664)

TDataSetEdit (see page 7671)

TDataSetInsert (see page 7684)

TDataSetFirst (see page 7677)

TDataSetNext (see page 7698)

TDataSetPost (see page 7704)

TDataSetPrior (see page 7711)

TDataSetRefresh (see page 7718)

TDataSource (see page 7070)

40.1.7.1 TDataSetLast Members

The following tables list the members exposed by TDataSetLast.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDataSet (see page 7655)	This is GetDataSet, a member of class TDataSetAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 7656)	Responds when components are created or destroyed.

	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 7696)	Navigates to the last record of the target dataset.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 7656)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 7697)	Enables or disables the action, depending on the state of the target dataset.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	DataSource (see page 7697)	Provides access to the target of the action.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

40.1.7.2 TDataSetLast Methods

The methods of the TDataSetLast class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 7696)	Navigates to the last record of the target dataset.
	UpdateTarget (see page 7697)	Enables or disables the action, depending on the state of the target dataset.

40.1.7.2.1 TDataSetLast.ExecuteTarget Method

Navigates to the last record of the target dataset.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

ExecuteTarget checks that Target is a data source component that has an associated dataset. If the DataSource (see page 7697) property is set, Target must be the same data source.

If Target is a valid data source, ExecuteTarget calls the associated dataset's Last method to move to the last record.

See Also

Last (see page 6968)

DataSource (see page 7657)

DataSet (see page 7078)

40.1.7.2.2 TDataSetLast.UpdateTarget Method

Enables or disables the action, depending on the state of the target dataset.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TDataSetLast (see page 7691), UpdateTarget checks whether the current target object (specified by the Target parameter) is a data source whose dataset is active and which is not already on the last record. If so, it sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

40.1.7.3 TDataSetLast Properties

The properties of the TDataSetLast class are listed here.

Published Properties

	Name	Description
	DataSource (see page 7697)	Provides access to the target of the action.

40.1.7.3.1 TDataSetLast.DataSource Property

Provides access to the target of the action.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource is the data source that represents the dataset on which the action performs its function. You can set this property to indicate the dataset that is the target of the action.

If you do not explicitly set the DataSource property, it is obtained from an active data-aware control when the action fires.

See Also

DataSet (see page 7078)

40.1.8 TDataSetNext Class

TDataSetNext moves a target dataset to its next record.

Class Hierarchy

Pascal

```
TDataSetNext = class(TDataSetAction);
```

C++

```
class TDataSetNext : public TDataSetAction;
```

File

DBActns

Description

Add TDataSetNext to an action list to allow users to navigate to the next record in a dataset. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the target dataset to move to the last record.

TDataSetNext is designed to work with a TDataSource component as its target. You can explicitly provide this target by setting the DataSource (see page 7704) property, or you can allow the action to infer the data source from an active data-aware control. TDataSetNext automatically disables itself if it does not have a target data source attached to an active dataset.

See Also

TDataSetCancel (see page 7657)

TDataSetDelete (see page 7664)

TDataSetEdit (see page 7671)

TDataSetInsert (see page 7684)

TDataSetFirst (see page 7677)

TDataSetLast (see page 7691)

TDataSetPost (see page 7704)

TDataSetPrior (see page 7711)

TDataSetRefresh (see page 7718)

TDataSource (see page 7070)

40.1.8.1 TDataSetNext Members

The following tables list the members exposed by TDataSetNext.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDataSet (see page 7655)	This is GetDataSet, a member of class TDataSetAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 7656)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.

	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 7703)	Navigates to the next record of the target dataset.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 7656)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

V	UpdateTarget (see page 7703)	Enables or disables the action, depending on the state of the target dataset.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	DataSource (see page 7704)	Provides access to the target of the action.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.

	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

40.1.8.2 TDataSetNext Methods

The methods of the TDataSetNext class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 7703)	Navigates to the next record of the target dataset.
	UpdateTarget (see page 7703)	Enables or disables the action, depending on the state of the target dataset.

40.1.8.2.1 TDataSetNext.ExecuteTarget Method

Navigates to the next record of the target dataset.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

ExecuteTarget checks that Target is a data source component that has an associated dataset. If the DataSource (see page 7704) property is set, Target must be the same data source.

If Target is a valid data source, ExecuteTarget calls the associated dataset's Next method to move to the next record.

See Also

Next (see page 6973)

DataSource (see page 7657)

DataSet (see page 7078)

40.1.8.2.2 TDataSetNext.UpdateTarget Method

Enables or disables the action, depending on the state of the target dataset.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TDataSetNext (see page 7698), UpdateTarget checks whether the current target object (specified by the Target parameter) is a data source whose dataset is active and which is not on the last record. If so, it sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

40.1.8.3 TDataSetNext Properties

The properties of the TDataSetNext class are listed here.

Published Properties

	Name	Description
	DataSource (see page 7704)	Provides access to the target of the action.

40.1.8.3.1 TDataSetNext.DataSource Property

Provides access to the target of the action.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource is the data source that represents the dataset on which the action performs its function. You can set this property to indicate the dataset that is the target of the action.

If you do not explicitly set the DataSource property, it is obtained from an active data-aware control when the action fires.

See Also

DataSet (see page 7078)

40.1.9 TDataSetPost Class

TDataSetPost posts the edits in the current record of a target dataset.

Class Hierarchy

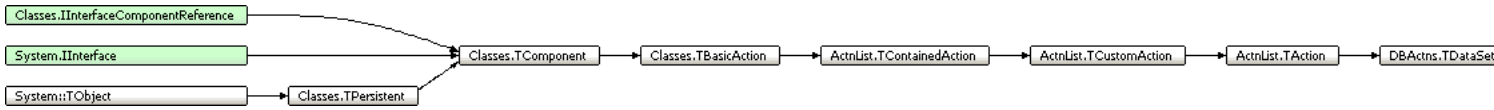

Pascal

```
TDataSetPost = class(TDataSetAction);
```

C++

```
class TDataSetPost : public TDataSetAction;
```

File

DBActns

Description

Add TDataSetPost to an action list to allow users to post dataset records. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the target dataset to post any edits made to the current record.

TDataSetPost is designed to work with a TDataSource component as its target. You can explicitly provide this target by setting the DataSource (see page 7711) property, or you can allow the action to infer the data source from an active data-aware control. TDataSetPost automatically disables itself if it does not have a target data source attached to a dataset that is in edit mode.

See Also

TDataSetCancel (see page 7657)

TDataSetDelete (see page 7664)

TDataSetEdit (see page 7671)

TDataSetPost

TDataSetFirst (see page 7677)

TDataSetNext (see page 7698)

TDataSetPrior (see page 7711)

TDataSetInsert (see page 7684)

TDataSetRefresh (see page 7718)

TDataSource (see page 7070)

40.1.9.1 TDataSetPost Members

The following tables list the members exposed by TDataSetPost.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.

	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDataSet (see page 7655)	This is GetDataSet, a member of class TDataSetAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7656)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 7710)	Posts the current record in the target dataset.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 7656)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 7710)	Enables or disables the action, depending on the state of the target dataset.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	DataSource (see page 7711)	Provides access to the target of the action.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

40.1.9.2 TDataSetPost Methods

The methods of the TDataSetPost class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 7710)	Posts the current record in the target dataset.
	UpdateTarget (see page 7710)	Enables or disables the action, depending on the state of the target dataset.

40.1.9.2.1 TDataSetPost.ExecuteTarget Method

Posts the current record in the target dataset.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

ExecuteTarget checks that Target is a data source component that has an associated dataset. If the DataSource (see page 7711) property is set, Target must be the same data source.

If Target is a valid data source, ExecuteTarget calls the associated dataset's Post method to post all edits to the current record.

See Also

Post (see page 6977)

DataSource (see page 7657)

DataSet (see page 7078)

40.1.9.2.2 TDataSetPost.UpdateTarget Method

Enables or disables the action, depending on the state of the target dataset.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TDataSetPost (see page 7704), UpdateTarget checks whether the current target object (specified by the Target parameter) is a data source whose dataset is in edit mode. If so, it sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

40.1.9.3 TDataSetPost Properties

The properties of the TDataSetPost class are listed here.

Published Properties

	Name	Description
	DataSource (see page 7711)	Provides access to the target of the action.

40.1.9.3.1 TDataSetPost.DataSource Property

Provides access to the target of the action.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource is the data source that represents the dataset on which the action performs its function. You can set this property to indicate the dataset that is the target of the action.

If you do not explicitly set the DataSource property, it is obtained from an active data-aware control when the action fires.

See Also

DataSet (see page 7078)

40.1.10 TDataSetPrior Class

TDataSetPrior moves a target dataset to the previous record.

Class Hierarchy

Pascal

```
TDataSetPrior = class(TDataSetAction);
```

C++

```
class TDataSetPrior : public TDataSetAction;
```

File

DBActns

Description

Add TDataSetPrior to an action list to allow users to navigate to the previous record in a dataset. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the target dataset to move to the previous record.

TDataSetPrior is designed to work with a TDataSource component as its target. You can explicitly provide this target by setting the DataSource (see page 7717) property, or you can allow the action to infer the data source from an active data-aware control. If you set DataSource (see page 7717), do not set it to a data source whose dataset is unidirectional. TDataSetPrior automatically disables itself if it does not have a target data source attached to an active dataset that is not on the first record.

See Also

TDataSetCancel (see page 7657)

TDataSetDelete (see page 7664)

TDataSetEdit (see page 7671)

TDataSetInsert (see page 7684)

TDataSetFirst (see page 7677)

TDataSetNext (see page 7698)

TDataSetPost (see page 7704)

TDataSetPrior

TDataSetRefresh (see page 7718)

TDataSource (see page 7070)

40.1.10.1 TDataSetPrior Members

The following tables list the members exposed by TDataSetPrior.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDataSet (see page 7655)	This is GetDataSet, a member of class TDataSetAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.

	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7656)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.

	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 7716)	Navigates to the previous record of the target dataset.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 7656)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 7717)	Enables or disables the action, depending on the state of the target dataset.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.

	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	DataSource (see page 7717)	Provides access to the target of the action.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

40.1.10.2 TDataSetPrior Methods

The methods of the TDataSetPrior class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 7716)	Navigates to the previous record of the target dataset.
	UpdateTarget (see page 7717)	Enables or disables the action, depending on the state of the target dataset.

40.1.10.2.1 TDataSetPrior.ExecuteTarget Method

Navigates to the previous record of the target dataset.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for

example, when the user selects a menu item or presses a tool button that is linked to this action).

ExecuteTarget checks that Target is a data source component that has an associated dataset. If the DataSource (see page 7717) property is set, Target must be the same data source.

If Target is a valid data source, ExecuteTarget calls the associated dataset's Prior method to move to the previous record.

See Also

Prior (see page 6979)

DataSource (see page 7657)

DataSet (see page 7078)

40.1.10.2.2 TDataSetPrior.UpdateTarget Method

Enables or disables the action, depending on the state of the target dataset.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TDataSetPrior (see page 7711), UpdateTarget checks whether the current target object (specified by the Target parameter) is a data source whose dataset is active and which is not on the first record. If so, it sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

40.1.10.3 TDataSetPrior Properties

The properties of the TDataSetPrior class are listed here.

Published Properties

	Name	Description
	DataSource (see page 7717)	Provides access to the target of the action.

40.1.10.3.1 TDataSetPrior.DataSource Property

Provides access to the target of the action.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource is the data source that represents the dataset on which the action performs its function. You can set this property to

indicate the dataset that is the target of the action.

If you do not explicitly set the DataSource property, it is obtained from an active data-aware control when the action fires.

See Also

DataSet (see page 7078)

40.1.11 TDataSetRefresh Class

TDataSetRefresh refreshes the records in a target dataset.

Class Hierarchy

Pascal

```
TDataSetRefresh = class(TDataSetAction);
```

C++

```
class TDataSetRefresh : public TDataSetAction;
```

File

DBActns

Description

Add TDataSetRefresh to an action list to allow users to refresh a dataset with the most current data. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the target dataset to refetch its data.

TDataSetRefresh is designed to work with a TDataSource component as its target. You can explicitly provide this target by setting the DataSource (see page 7724) property, or you can allow the action to infer the data source from an active data-aware control. TDataSetRefresh automatically disables itself if it does not have a target data source attached to an active dataset.

See Also

TDataSetCancel (see page 7657)

TDataSetDelete (see page 7664)

TDataSetEdit (see page 7671)

TDataSetRefresh

TDataSetFirst (see page 7677)

TDataSetNext (see page 7698)

TDataSetPrior (see page 7711)

TDataSetInsert (see page 7684)

TDataSetPost (see page 7704)

TDataSource (see page 7070)

40.1.11.1 TDataSetRefresh Members

The following tables list the members exposed by TDataSetRefresh.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDataSet (see page 7655)	This is GetDataSet, a member of class TDataSetAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 7656)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).

	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 7723)	Refreshes the data of a target dataset.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 7656)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 7723)	Enables or disables the action, depending on the state of the target dataset.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	DataSource (see page 7724)	Provides access to the target of the action.

	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

40.1.11.2 TDataSetRefresh Methods

The methods of the TDataSetRefresh class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 7723)	Refreshes the data of a target dataset.
	UpdateTarget (see page 7723)	Enables or disables the action, depending on the state of the target dataset.

40.1.11.2.1 TDataSetRefresh.ExecuteTarget Method

Refreshes the data of a target dataset.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

ExecuteTarget checks that Target is a data source component that has an associated dataset. If the DataSource (see page 7724) property is set, Target must be the same data source.

If Target is a valid data source, ExecuteTarget calls the associated dataset's Refresh method to refetch the most current data.

See Also

Refresh (see page 6991)

DataSource (see page 7657)

DataSet (see page 7078)

40.1.11.2.2 TDataSetRefresh.UpdateTarget Method

Enables or disables the action, depending on the state of the target dataset.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TDataSetRefresh (see page 7718), UpdateTarget checks whether the current target object (specified by the Target parameter) is a data source whose dataset is active and permits edits. If so, it sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

40.1.11.3 TDataSetRefresh Properties

The properties of the TDataSetRefresh class are listed here.

Published Properties

	Name	Description
	DataSource (see page 7724)	Provides access to the target of the action.

40.1.11.3.1 TDataSetRefresh.DataSource Property

Provides access to the target of the action.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource is the data source that represents the dataset on which the action performs its function. You can set this property to indicate the dataset that is the target of the action.

If you do not explicitly set the DataSource property, it is obtained from an active data-aware control when the action fires.

See Also

DataSet (see page 7078)

41 DBAdapt Namespace

This is namespace DBAdapt.

Topics

Name	Description
Classes (see page 7728)	The following table lists classes in this documentation.
Interfaces (see page 8102)	The following table lists interfaces in this documentation.
Functions (see page 8110)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 8111)	The following table lists structs, records, enums in this documentation.
Types (see page 8113)	The following table lists types in this documentation.
Variables (see page 8116)	The following table lists variables in this documentation.
Constants (see page 8117)	The following table lists constants in this documentation.

Classes

	Name	Description
	TBaseDataSetAdapterField (see page 7729)	TBaseDataSetAdapterField is the common base class for objects that represent an adapter field for a field in a database.
	TBaseDataSetAdapterImageField (see page 7755)	TBaseDataSetAdapteImageField is the base class for the TDataSetAdapterImageField.
	TBaseDataSetAdapterInputField (see page 7756)	TBaseDataSetAdapterInputField is the common base class for adapter field objects that represent a dataset field of a type users can edit directly.
	TCustomDataSetAdapter (see page 7766)	TCustomDataSetAdapter is the base class for adapters that represent the information from a dataset.
	TCustomDataSetAdapterAction (see page 7811)	TCustomDataSetAdapterAction is the base class for adapter actions that can be added to a dataset adapter.
	TCustomDataSetAdapterField (see page 7823)	TCustomDataSetAdapterField is the common base class for adapter field objects that represent a dataset field of a type whose value can be displayed as a simple text element.
	TCustomDataSetAdapterMemoField (see page 7837)	The TCustomDataSetAdapterMemoField is the ancestor of TDataSetAdapterMemoField (see page 7996).
	TCustomDataSetAdapterRowAction (see page 7846)	TCustomDataSetAdapterRowAction is the base class for adapter actions that operate on the records of a dataset.
	TCustomDataSetValuesList (see page 7855)	TCustomDataSetValuesList is the base class for components that represent a set of name/value pairs taken from a dataset.
	TDataSetAdapter (see page 7872)	TDataSetAdapter defines a scriptable interface to a TDataSet component and allows it to operate properly in a stateless environment.
	TDataSetAdapterApplyRowAction (see page 7890)	TDataSetAdapterApplyRowAction applies the updates from an HTTP request to a single record of an adapter's dataset.

	TDataSetAdapterBrowseRowAction (see page 7903)	TDataSetAdapterBrowseRowAction applies the updates from an HTTP request to a single record of an adapter's dataset and changes the dataset adapter to browse mode.
	TDataSetAdapterCancelRowAction (see page 7916)	TDataSetAdapterBrowseRowAction (see page 7903) cancels edits made in an HTML form and changes the dataset adapter to browse mode.
	TDataSetAdapterDataLink (see page 7930)	<insert text here>
	TDataSetAdapterDeleteRowAction (see page 7934)	TDataSetAdapterDeleteRowAction deletes the current row from a dataset, as specified in an HTTP request.
	TDataSetAdapterEditRowAction (see page 7947)	TDataSetAdapterDeleteRowAction (see page 7934) puts the parent dataset adapter into edit mode.
	TDataSetAdapterField (see page 7960)	TDataSetAdapterField is the an adapter field object that represents a database field whose value can be displayed as a simple text element.
	TDataSetAdapterFirstRowAction (see page 7972)	TDataSetAdapterNextRowAction (see page 8030) posts any edits made in an HTML form and moves the dataset adapter to the first record.
	TDataSetAdapterLastRowAction (see page 7984)	TDataSetAdapterLastRowAction posts any edits made in an HTML form and moves the dataset adapter to the last record.
	TDataSetAdapterMemoField (see page 7996)	The TDataSetAdapterMemoField represents a memo field in a dataset.
	TDataSetAdapterMoveRowAction (see page 8008)	TDataSetAdapterMoveRowAction is the base class for adapter actions that change the current record of a dataset.
	TDataSetAdapterNewRowAction (see page 8016)	TDataSetAdapterNewRowAction places the parent dataset adapter in insert mode.
	TDataSetAdapterNextRowAction (see page 8030)	TDataSetAdapterNextRowAction posts any edits made in an HTML form and moves the dataset adapter to the next record.
	TDataSetAdapterPrevRowAction (see page 8042)	TDataSetAdapterPrevRowAction posts any edits made in an HTML form and moves the dataset adapter to the prior record.
	TDataSetAdapterRefreshRowAction (see page 8054)	TDataSetAdapterRefreshRowAction refetches the current field values for the dataset record specified in the current HTTP request.
	TDataSetAdapterRowAction (see page 8066)	TDataSetAdapterRowAction is a general-purpose action for dataset adapters.
	TDataSetValuesList (see page 8079)	TDataSetValuesList is an adapter component that provides a list of name/value pairs by retrieving values from a TDataSet.
	TLocateParams (see page 8090)	TLocateParams is a collection of the key fields and their corresponding values that can be used to locate a record of a dataset adapter.
	TLocateParamsList (see page 8096)	TLocateParamsList is a collection of TLocateParams (see page 8090) objects.

Constants

Name	Description
AllActionParamOptions (see page 8117)	This is constant DBAdapt.AllActionParamOptions.
AllImageParamOptions (see page 8117)	This is constant DBAdapt.AllImageParamOptions.
DataSetAdapterModeNames (see page 8117)	This is constant DBAdapt.DataSetAdapterModeNames.

Functions

	Name	Description
	AddLocateParamsString (see page 8110)	This is function DBAdapt.AddLocateParamsString.

Interfaces

	Name	Description
	IAdapterDataSetComponent (see page 8102)	This is class DBAdapt.IAdapterDataSetComponent.
	IDataSetAdapterFieldClass (see page 8104)	This is class DBAdapt.IDataSetAdapterFieldClass.
	IGetDataSetKeyName (see page 8104)	This is class DBAdapt.IGetDataSetKeyName.
	IKeyParams (see page 8105)	IKeyParams lists the names and values of the fields a dataset adapter uses to uniquely identify database records.
	IPrepareDataSet (see page 8109)	This is class DBAdapt.IPrepareDataSet.

Structs, Records, Enums

	Name	Description
	TDataSetAdapterFieldFlag (see page 8111)	This is record DBAdapt.TDataSetAdapterFieldFlag.
	TDataSetAdapterMode (see page 8111)	This is record DBAdapt.TDataSetAdapterMode.
	TDataSetAdapterRequestParamOption (see page 8112)	This is record DBAdapt.TDataSetAdapterRequestParamOption.

Types

Name	Description
TDataAdapterFieldValidateValueEvent (see page 8113)	This is type DBAdapt.TDataAdapterFieldValidateValueEvent.
TDataSetAdapterFieldFlags (see page 8113)	This is type DBAdapt.TDataSetAdapterFieldFlags.
TDataSetAdapterFieldGetStringEvent (see page 8114)	This is type DBAdapt.TDataSetAdapterFieldGetStringEvent.
TDataSetAdapterFieldGetValueEvent (see page 8114)	This is type DBAdapt.TDataSetAdapterFieldGetValueEvent.
TDataSetAdapterFieldValueEvent (see page 8114)	This is type DBAdapt.TDataSetAdapterFieldValueEvent.
TDataSetAdapterImageFieldClass (see page 8115)	This is type DBAdapt.TDataSetAdapterImageFieldClass.
TDataSetAdapterModes (see page 8115)	This is type DBAdapt.TDataSetAdapterModes.
TDataSetAdapterRequestParamOptions (see page 8115)	This is type DBAdapt.TDataSetAdapterRequestParamOptions.

Variables

Name	Description
DataSetAdapterImageFieldClass (see page 8116)	This is variable DBAdapt.DataSetAdapterImageFieldClass.

41.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TBaseDataSetAdapterField (see page 7729)	TBaseDataSetAdapterField is the common base class for objects that represent an adapter field for a field in a database.
	TBaseDataSetAdapterImageField (see page 7755)	TBaseDataSetAdapteImageField is the base class for the TDataSetAdapterImageField.
	TBaseDataSetAdapterInputField (see page 7756)	TBaseDataSetAdapterInputField is the common base class for adapter field objects that represent a dataset field of a type users can edit directly.
	TCustomDataSetAdapter (see page 7766)	TCustomDataSetAdapter is the base class for adapters that represent the information from a dataset.
	TCustomDataSetAdapterAction (see page 7811)	TCustomDataSetAdapterAction is the base class for adapter actions that can be added to a dataset adapter.
	TCustomDataSetAdapterField (see page 7823)	TCustomDataSetAdapterField is the common base class for adapter field objects that represent a dataset field of a type whose value can be displayed as a simple text element.
	TCustomDataSetAdapterMemoField (see page 7837)	The TCustomDataSetAdapterMemoField is the ancestor of TDataSetAdapterMemoField (see page 7996).
	TCustomDataSetAdapterRowAction (see page 7846)	TCustomDataSetAdapterRowAction is the base class for adapter actions that operate on the records of a dataset.
	TCustomDataSetValuesList (see page 7855)	TCustomDataSetValuesList is the base class for components that represent a set of name/value pairs taken from a dataset.
	TDataSetAdapter (see page 7872)	TDataSetAdapter defines a scriptable interface to a TDataSet component and allows it to operate properly in a stateless environment.
	TDataSetAdapterApplyRowAction (see page 7890)	TDataSetAdapterApplyRowAction applies the updates from an HTTP request to a single record of an adapter's dataset.
	TDataSetAdapterBrowseRowAction (see page 7903)	TDataSetAdapterBrowseRowAction applies the updates from an HTTP request to a single record of an adapter's dataset and changes the dataset adapter to browse mode.
	TDataSetAdapterCancelRowAction (see page 7916)	TDataSetAdapterBrowseRowAction (see page 7903) cancels edits made in an HTML form and changes the dataset adapter to browse mode.
	TDataSetAdapterDataLink (see page 7930)	<insert text here>
	TDataSetAdapterDeleteRowAction (see page 7934)	TDataSetAdapterDeleteRowAction deletes the current row from a dataset, as specified in an HTTP request.
	TDataSetAdapterEditRowAction (see page 7947)	TDataSetAdapterDeleteRowAction (see page 7934) puts the parent dataset adapter into edit mode.
	TDataSetAdapterField (see page 7960)	TDataSetAdapterField is the an adapter field object that represents a database field whose value can be displayed as a simple text element.
	TDataSetAdapterFirstRowAction (see page 7972)	TDataSetAdapterNextRowAction (see page 8030) posts any edits made in an HTML form and moves the dataset adapter to the first record.
	TDataSetAdapterLastRowAction (see page 7984)	TDataSetAdapterLastRowAction posts any edits made in an HTML form and moves the dataset adapter to the last record.
	TDataSetAdapterMemoField (see page 7996)	The TDataSetAdapterMemoField represents a memo field in a dataset.
	TDataSetAdapterMoveRowAction (see page 8008)	TDataSetAdapterMoveRowAction is the base class for adapter actions that change the current record of a dataset.
	TDataSetAdapterNewRowAction (see page 8016)	TDataSetAdapterNewRowAction places the parent dataset adapter in insert mode.
	TDataSetAdapterNextRowAction (see page 8030)	TDataSetAdapterNextRowAction posts any edits made in an HTML form and moves the dataset adapter to the next record.
	TDataSetAdapterPrevRowAction (see page 8042)	TDataSetAdapterPrevRowAction posts any edits made in an HTML form and moves the dataset adapter to the prior record.
	TDataSetAdapterRefreshRowAction (see page 8054)	TDataSetAdapterRefreshRowAction refetches the current field values for the dataset record specified in the current HTTP request.
	TDataSetAdapterRowAction (see page 8066)	TDataSetAdapterRowAction is a general-purpose action for dataset adapters.

	TDataSetValuesList (see page 8079)	TDataSetValuesList is an adapter component that provides a list of name/value pairs by retrieving values from a TDataSet.
	TLocateParams (see page 8090)	TLocateParams is a collection of the key fields and their corresponding values that can be used to locate a record of a dataset adapter.
	TLocateParamsList (see page 8096)	TLocateParamsList is a collection of TLocateParams (see page 8090) objects.

41.1.1 TBaseDataSetAdapterField Class

TBaseDataSetAdapterField is the common base class for objects that represent an adapter field for a field in a database.

Class Hierarchy

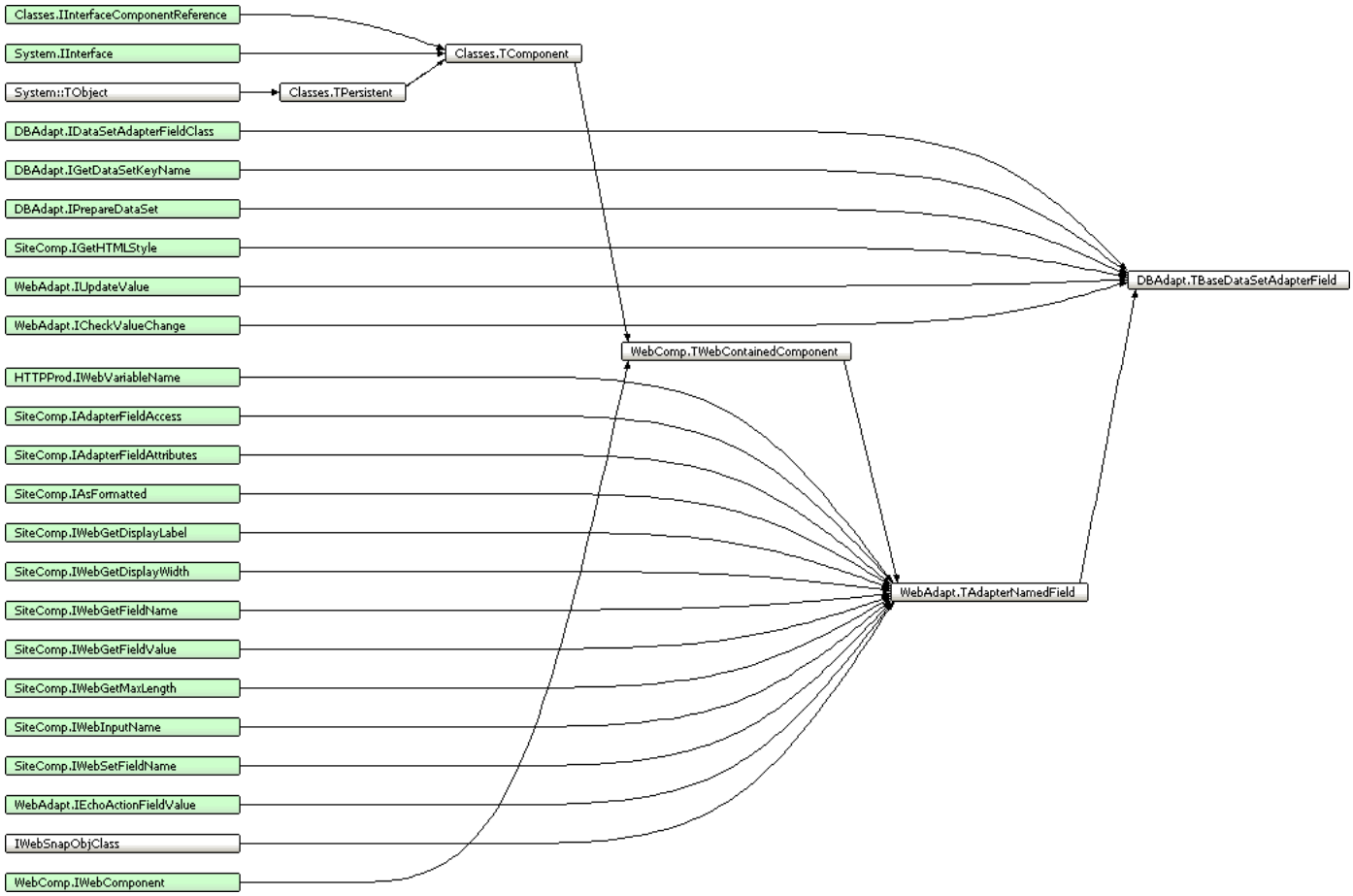

Pascal

```
TBaseDataSetAdapterField = class(TAdapterNamedField, IGetHTMLStyle, IUpdateValue,
ICheckValueChange, IDataSetAdapterFieldClass, IGetDataSetKeyName, IPrepareDataSet);
```

C++

```
class TBaseDataSetAdapterField : public TAdapterNamedField, public IGetHTMLStyle, public
IUpdateValue, public ICheckValueChange, public IDataSetAdapterFieldClass, public
IGetDataSetKeyName, public IPrepareDataSet;
```

File

DBAdapt

Description

TBaseDataSetAdapterField is the base class for all adapter field objects that represent database fields. TBaseDataSetAdapterField descendants are included as the fields of a dataset adapter (TDataSetAdapter (see page 7872)). Each TBaseDataSetAdapterField descendant represents a single field from a database table, and enables server-side script to refer to or update that field's value.

Do not use TBaseDataSetAdapterField directly. It is intended only as a base class, and the support for some actions (such as editing field values) consists of simple stubs that do nothing. Instead, use a TBaseDataSetAdapterField descendant that is appropriate to the type of the corresponding database field.

See Also

TDataSetAdapterImageField (see page 8138)

TDataSetAdapterField (see page 7960)

TDataSetAdapterMemoField (see page 7996)

TDataSetAdapter (see page 7872)

TField (see page 7115)

41.1.1.1 TBaseDataSetAdapterField Members

The following tables list the members exposed by TBaseDataSetAdapterField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckOrigValue (see page 7736)	Ensures that the current value of this field is the same as the original value.
	CheckValueChange (see page 7737)	Indicates whether the current value of this field is the same as the original value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 7738)	Indicates whether references to this field in HTML forms must be qualified with the name of the parent adapter.
	GetAdapter (see page 7738)	Returns the value of the Adapter (see page 7752) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7738)	Returns the dataset associated with this field's parent adapter.
V	GetDataSetFieldValue (see page 7739)	Returns the value of the associated dataset field.

	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 7739)	Returns a string value that suggests an HTML display style for this adapter field.
	GetDisplayStyleType (see page 7740)	Returns the HTML display style type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 7741)	Returns a string value that suggests an HTML editing style for this adapter field.
	GetInputStyleType (see page 7742)	Returns the style of the HTML control to use for editing this adapter field.
	GetKeyName (see page 7742)	Returns the name of the associated dataset field if it appears in the index (key) that the parent dataset adapter uses.
	GetKeyParams (see page 7743)	Returns a list of fields that the parent dataset adapter uses to locate records, along with their current values in that adapter's dataset.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetReadOnly (see page 7743)	Indicates whether the adapter field represents a read-only database field.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 7744)	Returns a string value that indicates what function a control that represents this field should perform.
	GetViewModeType (see page 7744)	Returns the view mode for the function a control that represents this field should perform.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 7745)	Provides the underlying implementation for the AsFormatted method.
	ImplCheckOrigValue (see page 7745)	Provides the underlying implementation for the CheckOrigValue (see page 7736) method.
	ImplCheckValueChange (see page 7746)	Provides the underlying implementation for the CheckValueChange (see page 7737) method.
	ImplGetDisplayLabel (see page 7747)	Provides the underlying implementation for the GetDisplayLabel method.
	ImplGetDisplayWidth (see page 7747)	Provides the underlying implementation for the GetDisplayWidth method.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 7747)	Provides the underlying implementation for the GetRequired method.
	ImplGetValue (see page 7748)	Provides the underlying implementation for the GetValue method.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 7748)	Provides the underlying implementation for the GetVisible method.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 7749)	Provides the underlying implementation for the SetFieldName method.
	ImplUpdateValue (see page 7749)	Provides the underlying implementation for the UpdateValue (see page 7751) method.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 7750)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	PrepareDataSet (see page 7750)	Responds when the parent dataset adapter prepares its dataset.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSetField (see page 7751)	Sets the value of the DataSetField (see page 7752) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 7751)	Updates the adapter field when the user assigns a new value.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7752)	Indicates the dataset adapter that uses this object to represent one of its fields.
	DataSetField (see page 7752)	Specifies the name of the dataset field that this adapter field represents.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 7754)	Occurs when the adapter field needs to supply its formatted value as a string that can appear in HTML forms.
	OnGetValue (see page 7754)	Occurs when the adapter field needs to supply its value as a Variant.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 7735)	Creates and initializes an instance of TBaseDataSetAdapterField (see page 7729).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 7737)	Returns the dataset field whose value this adapter field represents.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.

V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	FieldFlags (see page 7752)	Indicates various attributes of the adapter field.
	FieldModes (see page 7753)	Indicates what dataset adapter modes this field supports.

R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
R	KeyName (see page 8105)	This is KeyName, a member of class IGetDataSetKeyName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.1.2 TBaseDataSetAdapterField.Create Constructor

Creates and initializes an instance of TBaseDataSetAdapterField (see page 7729).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBaseDataSetAdapterField(TComponent * AOwner);
```

Description

The TBaseDataSetAdapterField (see page 7729) constructor allocates memory for the adapter field object and initializes its properties.

AOwner is the component that is responsible for starting the streaming process when a module containing the adapter field is loaded or saved. It becomes the value of the Owner property. Typically, AOwner is the Web module.

See Also

Owner (see page 2807)

41.1.1.3 TBaseDataSetAdapterField Methods

The methods of the TBaseDataSetAdapterField class are listed here.

Protected Methods

	Name	Description
	CheckOrigValue (see page 7736)	Ensures that the current value of this field is the same as the original value.
	CheckValueChange (see page 7737)	Indicates whether the current value of this field is the same as the original value.
V	FullyQualifyInputName (see page 7738)	Indicates whether references to this field in HTML forms must be qualified with the name of the parent adapter.
	GetAdapter (see page 7738)	Returns the value of the Adapter (see page 7752) property.

	GetDataSet (see page 7738)	Returns the dataset associated with this field's parent adapter.
	GetDataSetFieldValue (see page 7739)	Returns the value of the associated dataset field.
	GetDisplayStyle (see page 7739)	Returns a string value that suggests an HTML display style for this adapter field.
	GetDisplayStyleType (see page 7740)	Returns the HTML display style type for this adapter field.
	GetInputStyle (see page 7741)	Returns a string value that suggests an HTML editing style for this adapter field.
	GetInputStyleType (see page 7742)	Returns the style of the HTML control to use for editing this adapter field.
	GetKeyName (see page 7742)	Returns the name of the associated dataset field if it appears in the index (key) that the parent dataset adapter uses.
	GetKeyParams (see page 7743)	Returns a list of fields that the parent dataset adapter uses to locate records, along with their current values in that adapter's dataset.
	GetReadOnly (see page 7743)	Indicates whether the adapter field represents a read-only database field.
	GetViewMode (see page 7744)	Returns a string value that indicates what function a control that represents this field should perform.
	GetViewModeType (see page 7744)	Returns the view mode for the function a control that represents this field should perform.
	ImplAsFormatted (see page 7745)	Provides the underlying implementation for the AsFormatted method.
	ImplCheckOrigValue (see page 7745)	Provides the underlying implementation for the CheckOrigValue (see page 7736) method.
	ImplCheckValueChange (see page 7746)	Provides the underlying implementation for the CheckValueChange (see page 7737) method.
	ImplGetDisplayLabel (see page 7747)	Provides the underlying implementation for the GetDisplayLabel method.
	ImplGetDisplayWidth (see page 7747)	Provides the underlying implementation for the GetDisplayWidth method.
	ImplGetRequired (see page 7747)	Provides the underlying implementation for the GetRequired method.
	ImplGetValue (see page 7748)	Provides the underlying implementation for the GetValue method.
	ImplGetVisible (see page 7748)	Provides the underlying implementation for the GetVisible method.
	ImplSetFieldName (see page 7749)	Provides the underlying implementation for the SetFieldName method.
	ImplUpdateValue (see page 7749)	Provides the underlying implementation for the UpdateValue (see page 7751) method.
	Notification (see page 7750)	Forwards notification messages to all owned components.
	PrepareDataSet (see page 7750)	Responds when the parent dataset adapter prepares its dataset.
	SetDataSetField (see page 7751)	Sets the value of the DataSetField (see page 7752) property.
	UpdateValue (see page 7751)	Updates the adapter field when the user assigns a new value.

Public Methods

	Name	Description
	FindField (see page 7737)	Returns the dataset field whose value this adapter field represents.

41.1.1.3.1 TBaseDataSetAdapterField.CheckOrigValue Method

Ensures that the current value of this field is the same as the original value.

Pascal

```
procedure CheckOrigValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

```
__fastcall CheckOrigValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

Dataset adapter fields call CheckOrigValue internally when the field value must be the same as the original field value. The original value is stored as a hidden field on HTML forms when the FieldFlags (see page 7752) property includes ffInOrigValues. If the current value differs from the original value, CheckOrigValue raises an exception.

AActionRequest is interface of the action request object. CheckOrigValue can use this interface to obtain information about the current request (such as the value of the hidden field that contains the original value).

AFieldIndex is the index of this field in the set of field values that AActionRequest maintains.

CheckOrigValue calls the ImplCheckOrigValue (see page 7745) method, which performs the actual checking.

See Also

CheckValueChange (see page 7737)

ImplCheckOrigValue (see page 7745)

FieldFlags (see page 7752)

ActionOrigValue (see page 23943)

Value (see page 23945)

41.1.1.3.2 TBaseDataSetAdapterField.CheckValueChange Method

Indicates whether the current value of this field is the same as the original value.

Pascal

```
function CheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
```

C++

```
__fastcall Boolean CheckValueChange(IActionRequest AActionRequest, int AFieldIndex);
```

Description

CheckValueChange indicates whether the value of this adapter field has changed from its previous value.

AActionRequest is the interface of the action request object. CheckOrigValue (see page 7736) can use this interface to obtain information about the current request (such as the value of the hidden field that contains the original value).

AFieldIndex is the index of this field in the set of field values that AActionRequest maintains.

CheckValueChange returns true if the value has changed, false if it is the same as the original value.

CheckValueChange calls the ImplCheckValueChange (see page 7746) method, which performs the actual checking.

See Also

CheckOrigValue (see page 7736)

ImplCheckValueChange (see page 7746)

Value (see page 23945)

41.1.1.3.3 TBaseDataSetAdapterField.FindField Method

Returns the dataset field whose value this adapter field represents.

Pascal

```
function FindField: TField;
```

C++

```
__fastcall TField FindField();
```

Description

Call FindField to access the field component for the dataset field that this adapter makes available to server-side scripts.

FindField returns the field with the name specified by the DataSetField (see page 7752) property from the dataset that is associated with the parent adapter. If DataSetField (see page 7752) is an empty string, FindField returns nil (Delphi) or NULL (C++).

Note: Calling FindField can be more efficient than using the parent adapter to access the dataset and calling its FindField

method. This is because TBaseDataSetAdapterField (see page 7729) caches a reference to the field component when possible.

See Also

DataSetField (see page 7752)

FindField (see page 6938)

GetDataSet (see page 7738)

41.1.1.3.4 TBaseDataSetAdapterField.FullyQualifyInputName Method

Indicates whether references to this field in HTML forms must be qualified with the name of the parent adapter.

Pascal

```
function FullyQualifyInputName: Boolean; override;
```

C++

```
virtual __fastcall Boolean FullyQualifyInputName();
```

Description

The WebSnap application calls FullyQualifyInputName when generating an input name for this adapter field in an HTML document. Input names must be qualified by an adapter name when a single HTML form is used to input values for multiple adapters. This function returns true if the field name must be qualified with the name of the adapter to which it belongs, false otherwise.

FullyQualifyInputName calls the parent adapter's FullyQualifyInputNames method, which checks whether there are other dataset adapters in a master/detail relationship with it.

See Also

FullyQualifyInputNames (see page 7782)

Adapter (see page 7752)

41.1.1.3.5 TBaseDataSetAdapterField.GetAdapter Method

Returns the value of the Adapter (see page 7752) property.

Pascal

```
function GetAdapter: TCustomDataSetAdapter;
```

C++

```
__fastcall TCustomDataSetAdapter GetAdapter();
```

Description

GetAdapter is the protected read implementation of the Adapter (see page 7752) property.

See Also

Adapter (see page 7752)

41.1.1.3.6 TBaseDataSetAdapterField.GetDataSet Method

Returns the dataset associated with this field's parent adapter.

Pascal

```
function GetDataSet: TDataSet;
```

C++

```
__fastcall TDataSet GetDataSet();
```

Description

GetDataSet returns the value of the parent adapter's DataSet property. If the value of this field's Adapter (see page 7752) property is nil (Delphi) or NULL (C++), GetDataSet returns nil (Delphi) or NULL (C++).

See Also

FindField (see page 7737)

Adapter (see page 7752)

DataSet (see page 7807)

41.1.1.3.7 TBaseDataSetAdapterField.GetDataSetFieldValue Method

Returns the value of the associated dataset field.

Pascal

```
function GetDataSetFieldValue(Field: TField): Variant; virtual;
```

C++

```
virtual __fastcall Variant GetDataSetFieldValue(TField Field);
```

Description

GetDataSetFieldValue returns the value of the associated dataset field, given a field object. This method is called by the implementation of the Value property when there is no OnGetValue (see page 7754) event handler.

Field is the dataset field whose value this adapter field represents. It is the component that the FindField (see page 7737) method returns.

In TBaseDataSetAdapterField (see page 7729), GetDataSetFieldValue returns the Value property of the Field parameter. Descendant classes can override this method to derive the field value from the dataset field in some other way. For example, TCustomDataSetAdapterImageField uses the display text, because the field component's Value property is a binary encoding of the image. Similarly, if you create a custom descendant for encrypted fields, GetDataSetFieldValue could decrypt the value that is stored in the database table.

See Also

OnGetValue (see page 7754)

Value (see page 23945)

FindField (see page 7737)

Value (see page 7180)

41.1.1.3.8 TBaseDataSetAdapterField.GetDisplayStyle Method

Returns a string value that suggests an HTML display style for this adapter field.

Pascal

```
function GetDisplayStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall AnsiString GetDisplayStyle(const AnsiString AAdapterMode);
```

Description

GetDisplayStyle implements the DisplayStyle property of this adapter field. The DisplayStyle property can be used in server-side script to determine an appropriate control for displaying the field's value.

AAdapterMode indicates the current mode of the parent dataset adapter. It is one of the following values:

Value	Mode	The HTML control is used for
Insert	amInsert	Providing a field value on a newly inserted record.
Edit	amEdit	Modifying an existing field value.
Browse	amBrowse	Read-only viewing of an existing field value.
Query	amQuery	Entering a value to use for the field in a generated query.

GetDisplayStyle returns one of the following strings:

Value	Meaning
Text	Display the field as read-only text
Image	Display the field as an image
List	Display the field as a list of read only values

Note: Although GetDisplayStyle is not a virtual method, it uses the virtual GetDisplayStyleType (see page 7740) method to determine what value to return. To change the way the adapter field determines the display style, override the GetDisplayStyleType (see page 7740) method.

See Also

GetDisplayStyleType (see page 7740)

GetInputStyle (see page 7741)

TDataSetAdapterModes (see page 8115)

41.1.1.3.9 TBaseDataSetAdapterField.GetDisplayStyleType Method

Returns the HTML display style type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
virtual;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const AnsiString
AAdapterMode);
```

Description

GetDisplayStyleType returns the best way to display this field's value in HTML forms.

AAdapterMode indicates the current mode of the parent dataset adapter. It is one of the following values:

Value	Mode	Meaning
Insert	amInsert	The field value is to be used for entering a value in a newly inserted record.

Edit	amEdit	The field value is to be used for editing the current field value.
Browse	amBrowse	The field value is to be displayed in a read-only manner.
Query	amQuery	The field value represents a criterion to use in a generated query.

In TBaseDataSetAdapterField (see page 7729), GetDisplayStyleType always returns htmldText, indicating that the value should appear as read-only text. Descendant classes override this to allow other display styles when necessary.

See Also

GetDisplayStyle (see page 7739)

GetInputStyleType (see page 7742)

TDataSetAdapterModes (see page 8115)

41.1.1.3.10 TBaseDataSetAdapterField.GetInputStyle Method

Returns a string value that suggests an HTML editing style for this adapter field.

Pascal

```
function GetInputStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall AnsiString GetInputStyle(const AnsiString AAdapterMode);
```

Description

GetInputStyle implements the InputStyle property of this adapter field. The InputStyle property can be used in server-side script to determine an appropriate control for editing the field's value.

AAdapterMode indicates the current mode of the parent dataset adapter. It is one of the following values:

Value	Mode	The HTML control is used for
Insert	amInsert	Providing a field value on a newly inserted record.
Edit	amEdit	Modifying an existing field value.
Browse	amBrowse	Read-only viewing of an existing field value.
Query	amQuery	Entering a value to use for the field in a generated query.

GetInputStyle returns one of the following strings:

Value	Meaning
TextInput	Edit the field using a single line input field.
PasswordInput	Edit the field as a password field.
Select	Edit the field using a single selection list.
SelectMultiple	Edit the field using a multiple selection list.
Radio	Edit the field using radio buttons.
CheckBox	Edit the field using check boxes.
TextArea	Edit the field using a multi-line input field.
File	Edit the field using a file upload input field.

Note: Although GetInputStyle is not a virtual method, it uses the virtual GetInputStyleType (see page 7742) method to

determine what value to return. To change the way the adapter field determines the input style, override the GetInputStyleType (see page 7742) method.

See Also

GetInputStyleType (see page 7742)

GetDisplayStyle (see page 7739)

TDataSetAdapterModes (see page 8115)

41.1.1.3.11 TBaseDataSetAdapterField.GetInputStyleType Method

Returns the style of the HTML control to use for editing this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; virtual;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const AnsiString
AAdapterMode);
```

Description

GetInputStyleType returns the recommended type of control to use for editing this field's value in HTML forms.

AAdapterMode indicates the current mode of the parent dataset adapter. It is one of the following values:

Value	Mode	Meaning
Insert	amInsert	The control is to be used for entering a field value in a newly inserted record.
Edit	amEdit	The control is to be used for editing the current field value.
Browse	amBrowse	The control is to be used for read-only display.
Query	amQuery	The control is to be used for entering a criterion to use in a generated query.

In TBaseDataSetAdapterField (see page 7729), GetInputStyleType always returns htmliTextInput, indicating that the default control is a single-line edit control. Descendant classes override this to allow other control styles when appropriate.

See Also

GetInputStyle (see page 7741)

GetDisplayStyleType (see page 7740)

TDataSetAdapterModes (see page 8115)

41.1.1.3.12 TBaseDataSetAdapterField.GetKeyName Method

Returns the name of the associated dataset field if it appears in the index (key) that the parent dataset adapter uses.

Pascal

```
function GetKeyName: string;
```

C++

```
__fastcall AnsiString GetKeyName();
```

Description

The parent dataset adapter calls GetKeyName when assembling a list of the field names in the key it uses. GetKeyName returns the name of the underlying database field (the value of the DataSetField (see page 7752) property) if the FieldFlags (see

page 7752) property includes ffInKey. Otherwise, GetKeyName returns an empty string.

See Also

GetKeyParams (see page 7743)

DataSetField (see page 7752)

GetKeyFields (see page 7786)

FieldFlags (see page 7752)

41.1.1.3.13 TBaseDataSetAdapterField.GetKeyParams Method

Returns a list of fields that the parent dataset adapter uses to locate records, along with their current values in that adapter's dataset.

Pascal

```
procedure GetKeyParams(Params: TLocateParamsList);
```

C++

```
__fastcall GetKeyParams(TLocateParamsList Params);
```

Description

GetKeyParams fills out a TLocateParamsList (see page 8096) object with the current values of all fields that the parent dataset adapter uses to uniquely identify records, along with all similarly-used fields in any dataset adapter's linked to the parent dataset in a master/detail relationship.

Params is a collection of TLocateParams (see page 8090) objects, where each TLocateParams (see page 8090) object lists the key fields and their values for a single dataset adapter.

GetKeyParams calls the GetKeyParams method of the parent dataset adapter.

See Also

GetKeyName (see page 7742)

GetKeyParams (see page 7787)

GetKeyFields (see page 7786)

41.1.1.3.14 TBaseDataSetAdapterField.GetReadOnly Method

Indicates whether the adapter field represents a read-only database field.

Pascal

```
function GetReadOnly: Boolean;
```

C++

```
__fastcall Boolean GetReadOnly();
```

Description

GetReadOnly returns true if this adapter field represents a database field that is read-only, false otherwise (if the underlying database field is not read-only or if this adapter is not associated with a field object).

See Also

FindField (see page 7737)

ReadOnly (see page 7177)

41.1.1.3.15 TBaseDataSetAdapterField.GetViewMode Method

Returns a string value that indicates what function a control that represents this field should perform.

Pascal

```
function GetViewMode(const AAdapterMode: string): string;
```

C++

```
__fastcall AnsiString GetViewMode(const AnsiString AAdapterMode);
```

Description

GetViewMode implements the ViewMode property of this adapter field. The ViewMode property can be used in server-side script to determine an appropriate control for the field's value.

AAdapterMode indicates the current mode of the parent dataset adapter. It is one of the following values:

Value	Mode	The HTML control is used for
Insert	amInsert	Providing a field value on a newly inserted record.
Edit	amEdit	Modifying an existing field value.
Browse	amBrowse	Read-only viewing of an existing field value.
Query	amQuery	Entering a value to use for the field in a generated query.

GetViewMode returns one of the following strings:

Value	Meaning
Input	Display the adapter field as an input field, by default use the style returned by the GetInputStyle (see page 7741) method.
Display	Display the adapter field as a read-only field, by default use the style returned by GetDisplayStyle (see page 7739).
ToggleOnAccess	Display as an input field if the end-user has modify rights, otherwise displays as a read-only field.

Note: Although GetViewMode is not a virtual method, it uses the virtual GetViewModeType (see page 7744) method to determine what value to return. To change the way the adapter field determines the view mode, override the GetViewModeType (see page 7744) method.

See Also

GetInputStyle (see page 7741)

GetDisplayStyle (see page 7739)

TDataSetAdapterModes (see page 8115)

GetViewModeType (see page 7744)

41.1.1.3.16 TBaseDataSetAdapterField.GetViewModeType Method

Returns the view mode for the function a control that represents this field should perform.

Pascal

```
function GetViewModeType(const AAdapterMode: string): TAdapterDisplayViewModeType; virtual;
```

C++

```
virtual __fastcall TAdapterDisplayViewModeType GetViewModeType(const AnsiString AAdapterMode);
```

Description

GetViewModeType returns the recommended type of control (edit or display) to use for this field's value in HTML forms.

AAdapterMode indicates the current mode of the parent dataset adapter. It is one of the following values:

Value	Mode	Meaning
Insert	amInsert	The control is to be used for entering a field value in a newly inserted record.
Edit	amEdit	The control is to be used for editing the current field value.
Browse	amBrowse	The control is to be used for read-only display.
Query	amQuery	The control is to be used for entering a criterion to use in a generated query.

In TBaseDataSetAdapterField (see page 7729), GetViewModeType returns htmlvmDisplay if the associated field is read-only or AAdapterMode indicates that the parent adapter is in browse mode (amBrowse). Otherwise, it returns htmlvmInput.

See Also

GetInputStyle (see page 7741)

GetDisplayStyle (see page 7739)

TDataSetAdapterModes (see page 8115)

GetViewMode (see page 7744)

41.1.1.3.17 TBaseDataSetAdapterField.ImplAsFormatted Method

Provides the underlying implementation for the AsFormatted method.

Pascal

```
function ImplAsFormatted: string; override;
```

C++

```
virtual __fastcall AnsiString ImplAsFormatted();
```

Description

The AsFormatted method calls ImplAsFormatted to obtain a string to use as the value of the DisplayText property in server-side script.

ImplAsFormatted generates an OnGetDisplayText (see page 7754) event and returns the string the event handler returns. If there is no OnGetDisplayText (see page 7754) event handler, ImplAsFormatted returns the value of the associated dataset field's DisplayText property.

See Also

AsFormatted (see page 23931)

OnGetDisplayText (see page 7754)

DisplayText (see page 7162)

41.1.1.3.18 TBaseDataSetAdapterField.ImplCheckOrigValue Method

Provides the underlying implementation for the CheckOrigValue (see page 7736) method.

Pascal

```
procedure ImplCheckOrigValue(AActionRequest: IActionRequest; AFieldIndex: Integer); virtual;
```

C++

```
virtual __fastcall ImplCheckOrigValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The CheckOrigValue (see page 7736) method calls ImplCheckOrigValue to raise an exception if the current value of this field is not the same as the original value.

AActionRequest is interface of the action request object. ImplCheckOrigValue can use this interface to obtain information about the current request (such as the value of the hidden field that contains the original value).

AFieldIndex is the index of this field in the set of field values that AActionRequest maintains.

In TBaseDataSetAdapter, ImplCheckOrigValue does nothing (acting as if the current value always matches the original value). Descendant classes override this method if they can compare the current field value with the original values stored in the action request.

See Also

CheckOrigValue (see page 7736)

FieldFlags (see page 7752)

ActionOrigValue (see page 23943)

Value (see page 23945)

41.1.1.3.19 TBaseDataSetAdapterField.ImplCheckValueChange Method

Provides the underlying implementation for the CheckValueChange (see page 7737) method.

Pascal

```
function ImplCheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
virtual;
```

C++

```
virtual __fastcall Boolean ImplCheckValueChange(IActionRequest AActionRequest, int
AFieldIndex);
```

Description

The CheckValueChange (see page 7737) method calls ImplCheckValueChange to determine whether the value of this adapter field has changed from the value it held when building the form from which the current request originates.

AActionRequest is the interface of the action request object. CheckOrigValue (see page 7736) can use this interface to obtain information about the current request (such as the value of the hidden field that contains the original value).

AFieldIndex is the index of this field in the set of field values that AActionRequest maintains.

ImplCheckValueChange returns true if the value has changed. In TBaseDataSetAdapterField (see page 7729), ImplCheckValueChange always returns false, indicating no change to the field value. Descendant classes override this method if they can compare the current field value with the original values stored in the action request.

See Also

CheckValueChange (see page 7737)

Value (see page 23945)

41.1.1.3.20 TBaseDataSetAdapterField.ImplGetDisplayLabel Method

Provides the underlying implementation for the GetDisplayLabel method.

Pascal

```
function ImplGetDisplayLabel: string; override;
```

C++

```
virtual __fastcall AnsiString ImplGetDisplayLabel();
```

Description

GetDisplayLabel calls ImplGetDisplayLabel to obtain the label that can be used when generating HTML to display the value of this adapter field (in server-side script, the value of the DisplayLabel property).

ImplGetDisplayLabel calls FindField (see page 7737) to access the associated field component and returns the value of its DisplayLabel property. If FindField (see page 7737) returns nil (Delphi) or NULL (C++), ImplGetDisplayLabel returns an empty string.

See Also

DisplayLabel (see page 7160)

GetDisplayLabel (see page 23933)

FindField (see page 7737)

41.1.1.3.21 TBaseDataSetAdapterField.ImplGetDisplayWidth Method

Provides the underlying implementation for the GetDisplayWidth method.

Pascal

```
function ImplGetDisplayWidth: Integer; override;
```

C++

```
virtual __fastcall int ImplGetDisplayWidth();
```

Description

GetDisplayWidth calls ImplGetDisplayWidth to obtain the width (number of characters) to use for an HTML control that displays the value of this adapter field (in server-side script, the value of the DisplayWidth property).

ImplGetDisplayWidth calls FindField (see page 7737) to access the associated field component and returns the value of its DisplayWidth property. If FindField (see page 7737) returns nil (Delphi) or NULL (C++), ImplGetDisplayWidth returns –1, indicating that the default display width is undetermined.

See Also

DisplayWidth (see page 7162)

GetDisplayWidth (see page 23933)

FindField (see page 7737)

41.1.1.3.22 TBaseDataSetAdapterField.ImplGetRequired Method

Provides the underlying implementation for the GetRequired method.

Pascal

```
function ImplGetRequired: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplGetRequired();
```

Description

GetRequired calls ImplGetRequired to determine whether the end-user must supply a value for this field before submitting a form (in server-side script, the value of the Required property).

ImplGetRequired calls FindField (see page 7737) to access the associated field component and returns the value of its Required property. If FindField (see page 7737) returns nil (Delphi) or NULL (C++), ImplGetRequired returns false.

See Also

Required (see page 7177)

GetRequired (see page 23934)

FindField (see page 7737)

41.1.1.3.23 TBaseDataSetAdapterField.ImplGetValue Method

Provides the underlying implementation for the GetValue method.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

GetValue calls ImplGetValue to obtain the value of the adapter field as a Variant.

ImplGetValue first generates an OnGetValue (see page 7754) event, to allow an event handler to supply the value of the field. If there is no OnGetValue (see page 7754) event handler, ImplGetValue calls the GetDataSetFieldValue (see page 7739) method, which returns the value of the associated dataset field. If ImplGetValue can't locate the dataset field from which to obtain a value, it returns an unassigned Variant.

See Also

Value (see page 7180)

GetValue (see page 23935)

OnGetValue (see page 7754)

Value (see page 23945)

GetDataSetFieldValue (see page 7739)

41.1.1.3.24 TBaseDataSetAdapterField.ImplGetVisible Method

Provides the underlying implementation for the GetVisible method.

Pascal

```
function ImplGetVisible: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplGetVisible();
```

Description

GetVisible calls ImplGetVisible to determine whether this field should appear in HTML forms (in server-side script, the value of the Visible property).

ImplGetVisible first checks whether the FieldModes (see page 7753) property includes the current mode of the parent dataset adapter. If not, it returns false. If the current mode is supported, ImplGetVisible returns the value of the associated field component's Visible property.

See Also

Visible (see page 7181)

GetVisible (see page 23935)

FieldModes (see page 7753)

Mode (see page 7810)

41.1.1.3.25 TBaseDataSetAdapterField.ImplSetFieldName Method

Provides the underlying implementation for the SetFieldName method.

Pascal

```
procedure ImplSetFieldName(const Value: string); override;
```

C++

```
virtual __fastcall ImplSetFieldName(const AnsiString Value);
```

Description

The SetFieldName method calls ImplSetFieldName to assign the name by which server-side script can refer to this adapter field. ImplSetFieldName assigns the supplied name as the value of the DataSetField (see page 7752) property (binding this adapter field to a dataset field), and removes all illegal characters from the supplied name. Thus, the name for this field in server-side script may differ from the field-name only in omitting any illegal characters.

Value is the name of the field this adapter represents, and forms the basis of the resulting adapter field name.

See Also

DataSetField (see page 7752)

SetFieldName (see page 23942)

41.1.1.3.26 TBaseDataSetAdapterField.ImplUpdateValue Method

Provides the underlying implementation for the UpdateValue (see page 7751) method.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); virtual;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The UpdateValue (see page 7751) method calls ImplUpdateValue to respond when the user assigns a new value to this field.

AActionRequest is interface of the action request object. ImplUpdateValue can use this interface to obtain information about the current request (such as the value the user assigned to this field).

AFieldIndex is the index of this field in the set of field values that AActionRequest maintains.

ImplUpdateValue does nothing. Descendant classes that support field updates override this method to assign a new value to the associated database field.

See Also

UpdateValue (see page 7751)

Value (see page 23945)

41.1.1.3.27 TBaseDataSetAdapterField.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification calls the inherited method and performs additional cleanup actions when necessary. For more information, see the documentation for TComponent's Notification method.

See Also

Notification (see page 2783)

41.1.1.3.28 TBaseDataSetAdapterField.PrepareDataSet Method

Responds when the parent dataset adapter prepares its dataset.

Pascal

```
procedure PrepareDataSet;
```

C++

```
__fastcall PrepareDataSet();
```

Description

The parent dataset adapter calls PrepareDataSet from its own PrepareDataSet method. This allows the adapter field to reset itself as necessary to reflect the newly initialized state of the parent adapter's dataset.

In TBaseDataSetAdapterField (see page 7729), PrepareDataSet releases its cached instance of the associated field component. That way, the next call to FindField (see page 7737) checks the parent adapter's dataset for the field, in case the newly initialized dataset no longer includes the same field component.

See Also

PrepareDataSet (see page 7799)

FindField (see page 7737)

41.1.1.3.29 TBaseDataSetAdapterField.SetDataSetField Method

Sets the value of the DataSetField (see page 7752) property.

Pascal

```
procedure SetDataSetField(const Value: string);
```

C++

```
__fastcall SetDataSetField(const AnsiString Value);
```

Description

SetDataSetField is the protected write implementation of the DataSetField (see page 7752) property. This method assigns the name of the adapter field (as referenced from server-side script) as a side effect. The new name for the adapter field is the same as the new value of DataSetField (see page 7752), with any illegal characters removed.

Value is the new value of the DataSetField (see page 7752) property.

See Also

SetFieldName (see page 23942)

DataSetField (see page 7752)

41.1.1.3.30 TBaseDataSetAdapterField.UpdateValue Method

Updates the adapter field when the user assigns a new value.

Pascal

```
procedure UpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

```
__fastcall UpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

UpdateValue updates the associated database field to a value supplied in an incoming request, if possible.

AActionRequest is interface of the action request object. UpdateValue can use this interface to obtain information about the current request (such as the value the user assigned to this field).

AFieldIndex is the index of this field in the set of field values that AActionRequest maintains.

UpdateValue calls the ImplUpdateValue (see page 7749) method, which performs the actual update.

See Also

ImplUpdateValue (see page 7749)

Value (see page 23945)

41.1.1.4 TBaseDataSetAdapterField Properties

The properties of the TBaseDataSetAdapterField class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 7752)	Indicates the dataset adapter that uses this object to represent one of its fields.
	DataSetField (see page 7752)	Specifies the name of the dataset field that this adapter field represents.

Public Properties

	Name	Description
	FieldFlags (see page 7752)	Indicates various attributes of the adapter field.
	FieldModes (see page 7753)	Indicates what dataset adapter modes this field supports.

41.1.1.4.1 TBaseDataSetAdapterField.Adapter Property

Indicates the dataset adapter that uses this object to represent one of its fields.

Pascal

```
property Adapter: TCustomDataSetAdapter;
```

C++

```
__property TCustomDataSetAdapter Adapter;
```

Description

Use Adapter to access the parent adapter of this adapter field. This adapter is associated with the dataset whose database field this adapter field represents. This adapter field is one of the objects listed in its Data property.

See Also

Data (see page 24402)

TDataSet (see page 6894)

41.1.1.4.2 TBaseDataSetAdapterField.DataSetField Property

Specifies the name of the dataset field that this adapter field represents.

Pascal

```
property DataSetField: string;
```

C++

```
__property AnsiString DataSetField;
```

Description

DataSetField is the FieldName of the TField object whose value this adapter field represents. DataSetField can be the name of a simple field in a database table, the name of a subfield of an object field that has been qualified by the parent field's name, or the name of an aggregated field.

DataSetField is the same as the name of this adapter field in server-side script, except that the name of the adapter field omits any illegal characters (the only legal characters are the letters between a and z (or A and Z), digits, and the underline character).

If DataSetField is an empty string, this adapter field does not have a value, and the FindField (see page 7737) method returns nil (Delphi) or NULL (C++).

See Also

FieldName (see page 7164)

FindField (see page 7737)

41.1.1.4.3 TBaseDataSetAdapterField.FieldFlags Property

Indicates various attributes of the adapter field.

Pascal

```
property FieldFlags: TDataSetAdapterFieldFlags;
```

C++

```
__property TDataSetAdapterFieldFlags FieldFlags;
```

Description

FieldFlags indicates special uses of the value of this adapter field. This is a set that can indicate whether the value corresponds to a key field in the database (used to locate database records) and whether HTML forms store the original value of the field as a hidden field so that the application can detect changes to the field value.

See Also

IsIndexField (see page 7168)

DataSetField (see page 7752)

GetKeyName (see page 7742)

GetKeyFields (see page 7786)

41.1.1.4.4 TBaseDataSetAdapterField.FieldModes Property

Indicates what dataset adapter modes this field supports.

Pascal

```
property FieldModes: TDataSetAdapterModes;
```

C++

```
__property TDataSetAdapterModes FieldModes;
```

Description

FieldModes is a set of all the modes in which this adapter field can be used. If the parent dataset adapter is in a mode that is not included in this set, any forms generated using that adapter do not include a control to represent this field. Thus, for example, to make a field read-only, you can exclude amInsert and amEdit from FieldModes. Then, the field appears when the user is viewing or querying the dataset, but not when the user is editing or inserting a record.

See Also

CanModify (see page 7154)

Adapter (see page 7752)

Mode (see page 7810)

GetDisplayStyle (see page 7739)

GetInputStyle (see page 7741)

GetViewMode (see page 7744)

41.1.1.5 TBaseDataSetAdapterField Events

The events of the TBaseDataSetAdapterField class are listed here.

Public Events

	Name	Description
	OnGetDisplayText (see page 7754)	Occurs when the adapter field needs to supply its formatted value as a string that can appear in HTML forms.
	OnGetValue (see page 7754)	Occurs when the adapter field needs to supply its value as a Variant.

41.1.1.5.1 TBaseDataSetAdapterField.OnGetDisplayText Event

Occurs when the adapter field needs to supply its formatted value as a string that can appear in HTML forms.

Pascal

```
property OnGetDisplayText: TDataSetAdapterFieldGetStringEvent;
```

C++

```
__property TDataSetAdapterFieldGetStringEvent OnGetDisplayText;
```

Description

TBaseDataSetAdapterField (see page 7729) generates this event when supplying the read-only text representation of the field's value. This allows the WebSnap application to dynamically supply the formatted value of the field. Server-side script can access the resulting string using the DisplayText property.

Note: If you do not supply an OnGetDisplayText event handler, TBaseDataSetAdapterField (see page 7729) uses the field component's DisplayText property.

See Also

DisplayText (see page 7162)

Value (see page 23945)

OnGetValue (see page 7754)

41.1.1.5.2 TBaseDataSetAdapterField.OnGetValue Event

Occurs when the adapter field needs to supply its value as a Variant.

Pascal

```
property OnGetValue: TDataSetAdapterFieldGetValueEvent;
```

C++

```
__property TDataSetAdapterFieldGetValueEvent OnGetValue;
```

Description

TBaseDataSetAdapterField (see page 7729) generates an OnGetValue event when the WebSnap application reads its Value property. This allows the adapter field to dynamically supply the value for the field. For example, you can use this event to decrypt values that are stored in the database in an encrypted form.

Note: If you do not supply an OnGetValue event handler, TBaseDataSetAdapterField (see page 7729) uses the field component's Value property.

See Also

Value (see page 7180)

Value (see page 23945)

OnGetDisplayText (see page 7754)

41.1.2 TBaseDataSetAdapterImageField Class

TBaseDataSetAdapteImageField is the base class for the TDataSetAdapterImageField.

Class Hierarchy

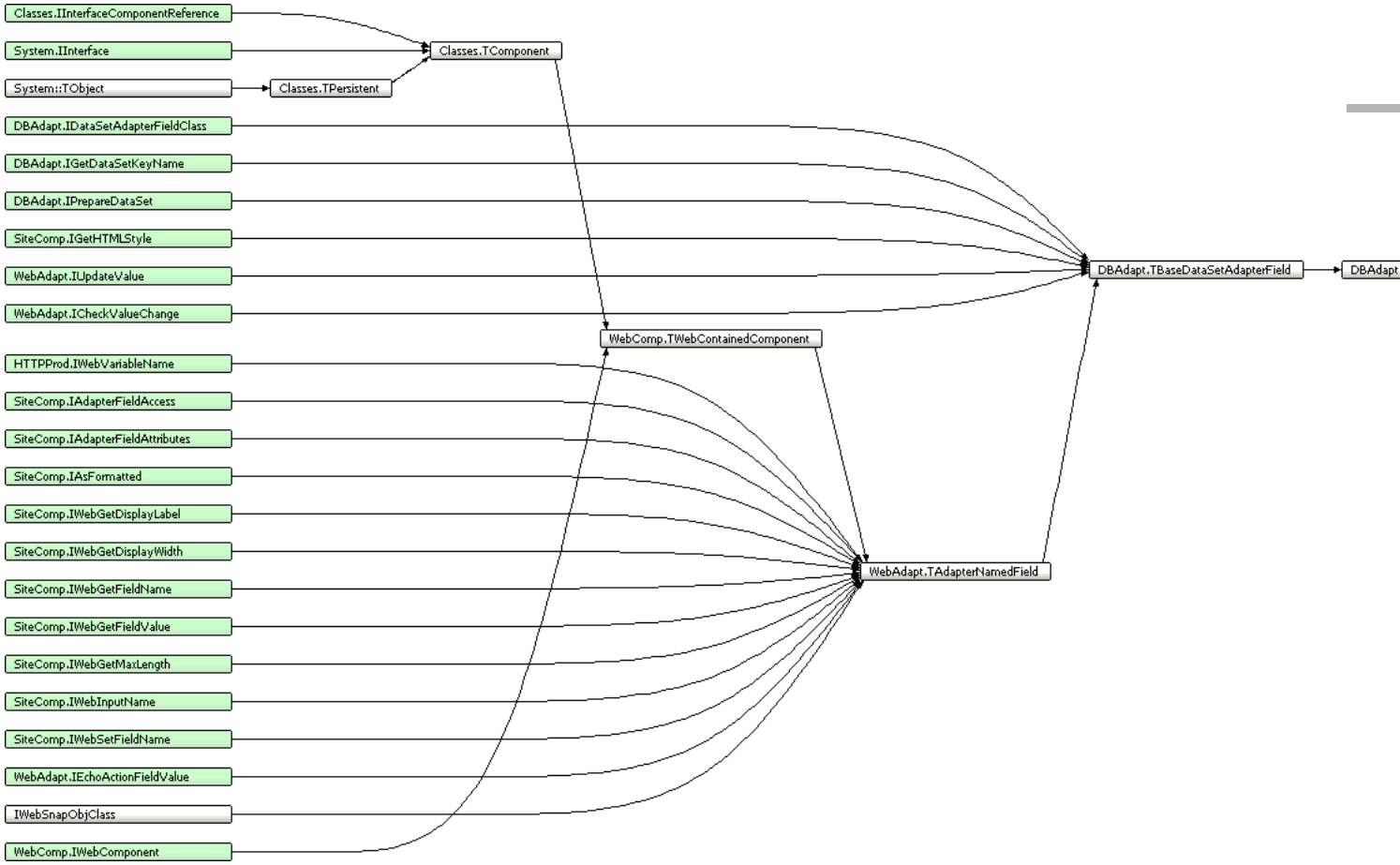

Pascal

```
TBaseDataSetAdapterImageField = class(TBaseDataSetAdapterField);
```

C++

```
class TBaseDataSetAdapterImageField : public TBaseDataSetAdapterField;
```

File

DBAdapt

Description

To allow TDataSetAdapter (see page 7872) to have an image field but not require that all applications built with TDataSetAdapter (see page 7872) use jpeg (see page 15685) units, the TDataSetAdapter (see page 7872) has an indirect reference to the image field class. DBAdapt.pas contains a variable named DataSetAdapterImageFieldClass (see page 8116) that references the class of the image field, which must be derived from TBaseDataSetAdapterImageField.variable. When the DBAdaptImage.pas unit is initialized, DataSetAdapterImageFieldClass (see page 8116) is assigned a class

reference to TDataSetAdapterImageField.

41.1.3 TBaseDataSetAdapterInputField Class

TBaseDataSetAdapterInputField is the common base class for adapter field objects that represent a dataset field of a type users can edit directly.

Class Hierarchy

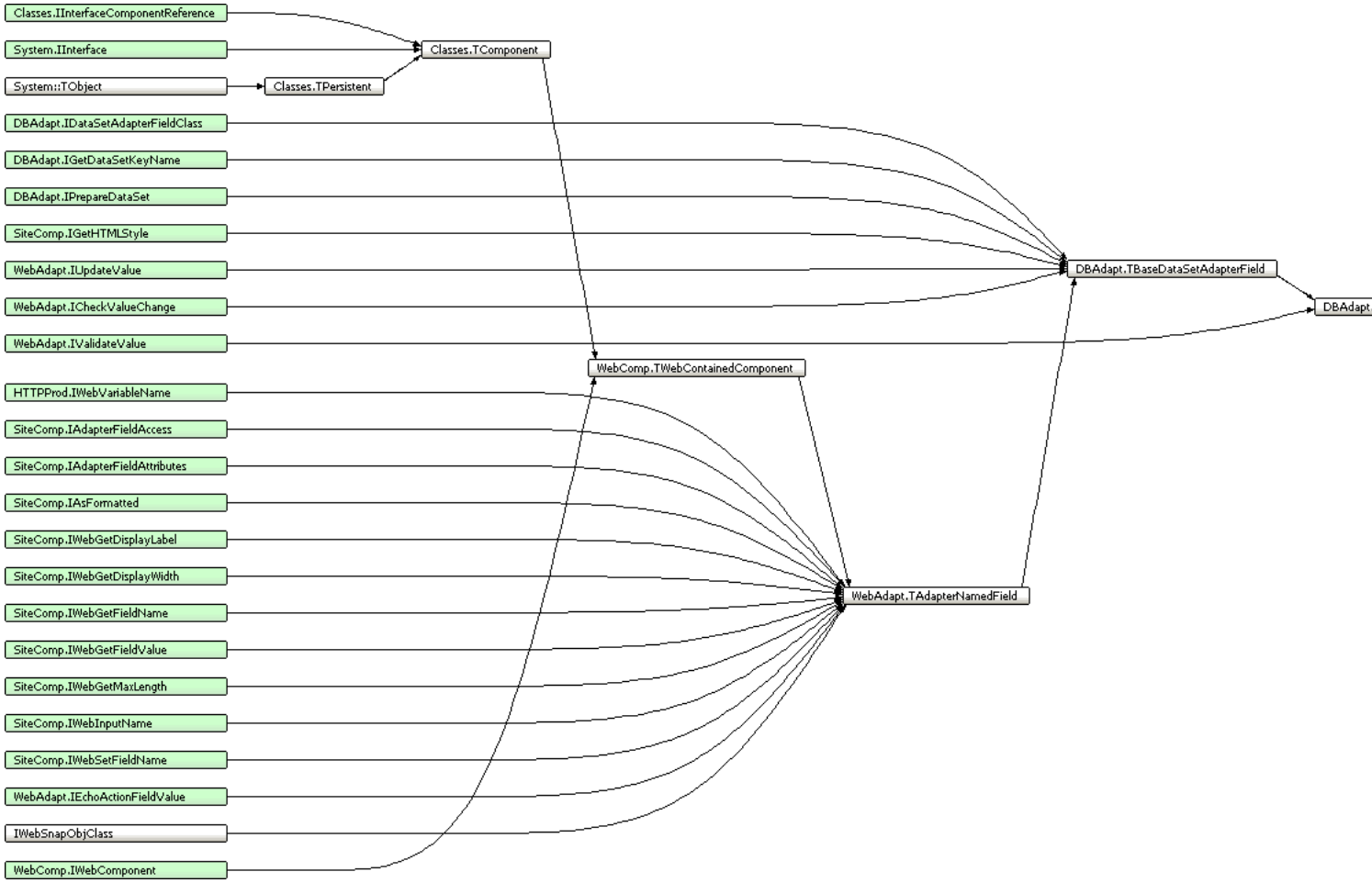

Pascal

```
TBaseDataSetAdapterInputField = class(TBaseDataSetAdapterField, IValidateValue);
```

C++

```
class TBaseDataSetAdapterInputField : public TBaseDataSetAdapterField, public IValidateValue;
```

File

DBAdapt

Description

TBaseDataSetAdapterInputField is the base class for adapter field objects that represent database fields of simple types that

users can edit directly. This is in contrast, for example, to TDataSetAdapterImageField, which represents a field type that users can only edit indirectly, by providing a file reference. TBaseDataSetAdapterInputField descendants are included as the fields of a dataset adapter (TDataSetAdapter (see page 7872)). Each TBaseDataSetAdapterInputField descendant represents a single field from a database table, and enables server-side script to refer to or update that field's value.

Do not use TBaseDataSetAdapterInputField directly. It is intended only as a base class, and the support for some actions (such as updating field values) consists of simple stubs that do nothing. Instead, use a TBaseDataSetAdapterInputField descendant such as TDataSetAdapterField (see page 7960) or TDataSetAdapterMemoField (see page 7996).

See Also

TDataSetAdapterImageField (see page 8138)

TDataSetAdapterField (see page 7960)

TDataSetAdapterMemoField (see page 7996)

TDataSetAdapter (see page 7872)

TField (see page 7115)

41.1.3.1 TBaseDataSetAdapterInputField Members

The following tables list the members exposed by TBaseDataSetAdapterInputField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckOrigValue (see page 7736)	Ensures that the current value of this field is the same as the original value.
	CheckValueChange (see page 7737)	Indicates whether the current value of this field is the same as the original value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 7738)	Indicates whether references to this field in HTML forms must be qualified with the name of the parent adapter.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 7738)	Returns the value of the Adapter (see page 7752) property.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7738)	Returns the dataset associated with this field's parent adapter.

	GetDataSetFieldValue (see page 7739)	Returns the value of the associated dataset field.
	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 7739)	Returns a string value that suggests an HTML display style for this adapter field.
	GetDisplayStyleType (see page 7740)	Returns the HTML display style type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 7741)	Returns a string value that suggests an HTML editing style for this adapter field.
	GetInputStyleType (see page 7742)	Returns the style of the HTML control to use for editing this adapter field.
	GetKeyName (see page 7742)	Returns the name of the associated dataset field if it appears in the index (key) that the parent dataset adapter uses.
	GetKeyParams (see page 7743)	Returns a list of fields that the parent dataset adapter uses to locate records, along with their current values in that adapter's dataset.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetReadOnly (see page 7743)	Indicates whether the adapter field represents a read-only database field.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 7744)	Returns a string value that indicates what function a control that represents this field should perform.
	GetViewModeType (see page 7744)	Returns the view mode for the function a control that represents this field should perform.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 7745)	Provides the underlying implementation for the AsFormatted method.
	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplCheckOrigValue (see page 7745)	Provides the underlying implementation for the CheckOrigValue (see page 7736) method.
	ImplCheckValueChange (see page 7746)	Provides the underlying implementation for the CheckValueChange (see page 7737) method.
	ImplGetDisplayLabel (see page 7747)	Provides the underlying implementation for the GetDisplayLabel method.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayWidth (see page 7747)	Provides the underlying implementation for the GetDisplayWidth method.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetEchoActionFieldValue (see page 7763)	Provides the underlying implementation for the GetEchoActionFieldValue method.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 7747)	Provides the underlying implementation for the GetRequired method.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 7748)	Provides the underlying implementation for the GetValue method.
	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 7748)	Provides the underlying implementation for the GetVisible method.

	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 7763)	Provides the underlying implementation for the SetEchoActionFieldValue method.
	ImplSetFieldName (see page 7749)	Provides the underlying implementation for the SetFieldName method.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	ImplUpdateValue (see page 7749)	Provides the underlying implementation for the UpdateValue (see page 7751) method.
	ImplValidateValue (see page 7764)	Provides the underlying implementation for the ValidateValue (see page 7765) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7750)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	PrepareDataSet (see page 7750)	Responds when the parent dataset adapter prepares its dataset.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSetField (see page 7751)	Sets the value of the DataSetField (see page 7752) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	TestValueChange (see page 7764)	Checks whether a new value matches the current value of this field.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 7751)	Updates the adapter field when the user assigns a new value.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateValue (see page 7765)	Validates a new value for this field that appears in the current request.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7752)	Indicates the dataset adapter that uses this object to represent one of its fields.
R	Adapter (see page 23943)	Returns the parent adapter component.
	DataSetField (see page 7752)	Specifies the name of the dataset field that this adapter field represents.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 7754)	Occurs when the adapter field needs to supply its formatted value as a string that can appear in HTML forms.
	OnGetValue (see page 7754)	Occurs when the adapter field needs to supply its value as a Variant.
	OnValidateValue (see page 7766)	Occurs when the parent adapter wants to validate the new value for this field.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 7735)	Creates and initializes an instance of TBaseDataSetAdapterField (see page 7729).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 7737)	Returns the dataset field whose value this adapter field represents.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetKeyName (see page 8105)	This is GetKeyName, a member of class IGetDataSetKeyName.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PrepareDataSet (see page 8109)	This is PrepareDataSet, a member of class IPrepareDataSet.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.

	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	FieldFlags (see page 7752)	Indicates various attributes of the adapter field.
	FieldModes (see page 7753)	Indicates what dataset adapter modes this field supports.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
R	KeyName (see page 8105)	This is KeyName, a member of class IGetDataSetKeyName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
	OnUpdateValue (see page 7765)	Occurs when the WebSnap application updates this adapter field's value based on in incoming request.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.3.2 TBaseDataSetAdapterInputField Methods

The methods of the TBaseDataSetAdapterInputField class are listed here.

Protected Methods

	Name	Description
	ImplGetEchoActionFieldValue (see page 7763)	Provides the underlying implementation for the GetEchoActionFieldValue method.
	ImplSetEchoActionFieldValue (see page 7763)	Provides the underlying implementation for the SetEchoActionFieldValue method.
	ImplValidateValue (see page 7764)	Provides the underlying implementation for the ValidateValue (see page 7765) method.
	TestValueChange (see page 7764)	Checks whether a new value matches the current value of this field.
	ValidateValue (see page 7765)	Validates a new value for this field that appears in the current request.

41.1.3.2.1 TBaseDataSetAdapterInputField.ImplGetEchoActionFieldValue Method

Provides the underlying implementation for the GetEchoActionFieldValue method.

Pascal

```
function ImplGetEchoActionFieldValue: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplGetEchoActionFieldValue();
```

Description

The GetEchoActionFieldValue method calls ImplGetEchoActionFieldValue to determine whether the current field value represents the adapter field's value reflects the current value of the associated dataset field, or whether it reflects the value stored in the current HTTP request.

ImplGetEchoActionFieldValue returns true if reading this field's Value property yields the value for this field that appears in the current HTTP request. It returns false if reading the Value property yields the value of the associated dataset field.

See Also

GetEchoActionFieldValue (see page 23933)

EchoActionFieldValue (see page 23944)

Value (see page 23945)

41.1.3.2.2 TBaseDataSetAdapterInputField.ImplSetEchoActionFieldValue Method

Provides the underlying implementation for the SetEchoActionFieldValue method.

Pascal

```
procedure ImplSetEchoActionFieldValue(AValue: Boolean); override;
```

C++

```
virtual __fastcall ImplSetEchoActionFieldValue(Boolean AValue);
```

Description

The SetEchoActionFieldValue method calls ImplSetEchoActionFieldValue to specify how this adapter field should implement its Value property.

AValue is the new value of the field's EchoActionFieldValue property. If AValue is true, the Value property should reflect the value for this field that appears in the current HTTP request. If AValue is false, the Value property should reflect the value of the associated dataset field.

See Also

SetEchoActionFieldValue (see page 23942)

EchoActionFieldValue (see page 23944)

Value (see page 23945)

41.1.3.2.3 TBaseDataSetAdapterInputField.ImplValidateValue Method

Provides the underlying implementation for the ValidateValue (see page 7765) method.

Pascal

```
procedure ImplValidateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); virtual;
```

C++

```
virtual __fastcall ImplValidateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The ValidateValue (see page 7765) method calls ImplValidateValue to validate a proposed new field value from the current request. This method raises an exception if the proposed new value is not valid.

AActionRequest is the interface of the action request object. ImplValidateValue uses this interface to obtain the new value for this field.

AFieldIndex is the index of this field in the list of fields that the action request object maintains.

ImplValidateValue looks up the appropriate value from the request. It then generates an OnValidateValue (see page 7766) event, supplying the value from the current request and a reference to the dataset field that is associated with this adapter field.

See Also

ValidateValue (see page 7765)

OnValidateValue (see page 7766)

41.1.3.2.4 TBaseDataSetAdapterInputField.TestValueChange Method

Checks whether a new value matches the current value of this field.

Pascal

```
function TestValueChange(AField: TField; const ANewValue: string): Boolean;
```

C++

```
__fastcall Boolean TestValueChange(TField AField, const AnsiString ANewValue);
```

Description

TestValueChange compares the current value of a field with a new value, which is typically taken from a request message. Dataset adapter fields use this method internally when checking for changes or when updating the field value.

AField is the field object whose value this adapter field represents.

ANewValue is the field value from the request message, formatted as a string.

TestValueChange returns true if ANewValue differs from the value of AField, false if it is the same as the value of AField.

See Also

Value (see page 7180)

41.1.3.2.5 TBaseDataSetAdapterInputField.ValidateValue Method

Validates a new value for this field that appears in the current request.

Pascal

```
procedure ValidateValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

```
__fastcall ValidateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

When the parent adapter receives an update request, it calls ValidateValue so that this field can validate the new value from the request message. If ValidateValue raises an exception, the adapter adds the exception message to an Errors object, which can then be used to report the error back to the user.

AActionRequest is the interface of the action request object. ValidateValue uses this interface to obtain the new value for this field.

AFieldIndex is the index of this field in the set of fields that AActionRequest maintains.

ValidateValue calls the ImplValidateValue (see page 7764) method, which generates an OnValidateValue (see page 7766) event to perform the validation.

See Also

ImplValidateValue (see page 7764)

OnValidateValue (see page 7766)

UpdateFieldsValidateValues (see page 7805)

41.1.3.3 TBaseDataSetAdapterInputField Properties

The properties of the TBaseDataSetAdapterInputField class are listed here.

Public Properties

	Name	Description
	OnUpdateValue (see page 7765)	Occurs when the WebSnap application updates this adapter field's value based on in incoming request.

41.1.3.3.1 TBaseDataSetAdapterInputField.OnUpdateValue Property

Occurs when the WebSnap application updates this adapter field's value based on in incoming request.

Pascal

```
property OnUpdateValue: TDatasetAdapterFieldValueEvent;
```

C++

```
__property TDatasetAdapterFieldValueEvent OnUpdateValue;
```

Description

Write an OnUpdateValue event handler to provide your own mechanism for updating the corresponding database field when the WebSnap application updates this adapter field's value. If you supply an OnUpdateValue event handler, the dataset adapter field assumes you have handled the update and does not write the new value to the associated dataset field. If there is no OnUpdateValue event handler, the dataset adapter field writes the field value to the associated dataset field.

Note: TBaseDataSetAdapterInputField (see page 7756) defines but does not generate this event. Instead, descendant classes generate this event when updating the adapter field value. If you create your own TBaseDataSetAdapterInputField (see page 7756) descendant, you must generate this event from your implementation of the UpdateValue (see page 7751) method.

See Also

Value (see page 7180)

Value (see page 23945)

OnGetValue (see page 7754)

ImplUpdateValue (see page 7749)

41.1.3.4 TBaseDataSetAdapterInputField Events

The events of the TBaseDataSetAdapterInputField class are listed here.

Public Events

	Name	Description
	OnValidateValue (see page 7766)	Occurs when the parent adapter wants to validate the new value for this field.

41.1.3.4.1 TBaseDataSetAdapterInputField.OnValidateValue Event

Occurs when the parent adapter wants to validate the new value for this field.

Pascal

```
property OnValidateValue: TDataAdapterFieldValidateValueEvent;
```

C++

```
__property TDataAdapterFieldValidateValueEvent OnValidateValue;
```

Description

Write an OnValidateValue event handler to provide your own validation of new field values before the WebSnap application attempts to update this field's value. If the new Value is not valid, the OnValidateValue event handler should raise an exception.

The parent dataset adapter generates this event when it receives new field values from an update request message.

See Also

Adapter (see page 7752)

Value (see page 23945)

OnGetValue (see page 7754)

OnValidate (see page 7183)

41.1.4 TCustomDataSetAdapter Class

TCustomDataSetAdapter is the base class for adapters that represent the information from a dataset.

Class Hierarchy

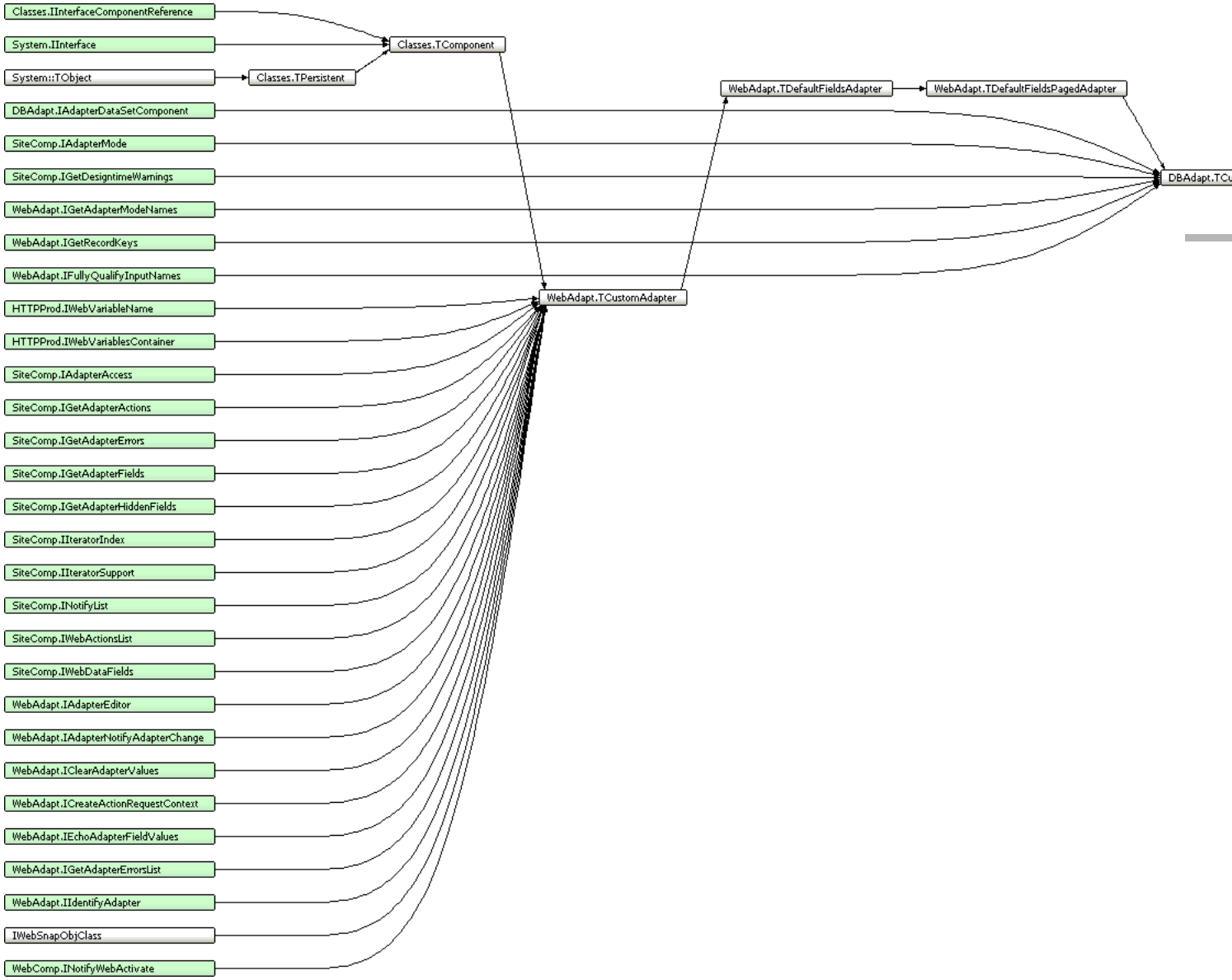

Pascal

```
TCustomDataSetAdapter = class(TDefaultFieldsPagedAdapter, IAdapterMode,
IAdapterDataSetComponent, IGetDesigntimeWarnings, IGetAdapterModeNames, IGetRecordKeys,
IFullyQualifyInputNames);
```

C++

```
class TCustomDataSetAdapter : public TDefaultFieldsPagedAdapter, public IAdapterMode, public
IAdapterDataSetComponent, public IGetDesigntimeWarnings, public IGetAdapterModeNames, public
IGetRecordKeys, public IFullyQualifyInputNames;
```

File

DBAdapt

Description

TCustomDataSetAdapter is the base class for adapters that define a scriptable interace to TDataSet objects. It allows datasets to operate properly in a stateless environment, and contains field and action components for working with the fields and records of the dataset. Each field of the dataset adapter represents a field in the dataset, and the actions operate on the underlying TDataSet object.

Two important properties that TCustomDataSetAdapter introduces are DataSet (see page 7807), which specifies the associated dataset for this adapter, and Mode (see page 7810), which indicates how forms built from this adapter are to operate on the information it provides. TCustomDataSetAdapter also introduces support for master/detail relationships between dataset. Use the MasterAdapter (see page 7809) property to indicate master/detail relationships between TDataSetAdapter (see page 7872) components.

Do not instantiate TCustomDataSetAdapter directly. Use TDataSetAdapter (see page 7872) or create your own concrete descendant.

See Also

TDataSetAdapter (see page 7872)

41.1.4.1 TCustomDataSetAdapter Members

The following tables list the members exposed by TCustomDataSetAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 7778)	Adds the default action components to this adapter.
V	AddDefaultFields (see page 7779)	Adds the default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
	EndOfPage (see page 24412)	Indicates whether the record index has gone past the last record in the current page.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	FullyQualifyInputNames (see page 7782)	Indicates whether the HTML input names must be qualified with the name of the adapter.
V	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetAdapterMode (see page 7783)	Returns a string that represents the current value of the Mode (see page 7810) property.
	GetAdapterModeNames (see page 7783)	Retrieves the strings for all the possible modes of this adapter.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetDataSet (see page 7784)	Returns the value of the DataSet (see page 7807) property.
V	GetDefaultMode (see page 7784)	Returns the default mode for this adapter.
	GetDesigntimeWarnings (see page 7784)	Populates a design-time warnings object with any warnings that the dataset adapter needs to emit.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
V	GetEOF (see page 7785)	Indicates whether the adapter is positioned at the end of its dataset.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
V	GetFirstRecord (see page 7785)	Positions the adapter on the first record of its dataset.
V	GetHiddenFieldOptions (see page 7786)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetKeyFields (see page 7786)	Returns a string listing the fields that this adapter uses to locate records.
	GetKeyParams (see page 7787)	Retrieves a list of fields that the dataset adapter uses to locate records, along with their current values.
	GetKeyParamStrings (see page 7787)	Retrieves a list of fields that the dataset adapter uses to locate records, along with their current values.
	GetMasterDetailAdapters (see page 7788)	Returns a list of all dataset adapters, including this one, linked to this adapter in a master/detail relationship.
	GetNeedAdapterMode (see page 7788)	Indicates whether the adapter requires a Mode (see page 7810) value.
V	GetNextRecord (see page 7789)	Positions the associated dataset on the next record.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.

	GetPageCount (see page 24413)	Calculates the number of pages needed to display all adapter records.
	GetRecordCount (see page 7789)	Returns the value of the RecordCount property.
	GetRecordIndex (see page 7790)	Returns the value of the RecordIndex property.
	GetRecordKeys (see page 7790)	Retrieves a list of field values that the dataset adapter uses to locate records, along with information for formatting hidden parameters from this values.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 7790)	Provides the underlying implementation for the CanAddActionClass method.
	ImplCanAddFieldClass (see page 7791)	Provides the underlying implementation for the CanAddFieldClass method.
	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
	ImplEndIterator (see page 7792)	Implements IteratorSupport's EndIterator method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 7792)	Provides the underlying implementation for the GetActionsList method.
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 7793)	Provides the underlying implementation for the GetFieldsList method.
	ImplGetIteratorIndex (see page 7793)	Provides the underlying implementation for the GetIteratorIndex method.
	ImplGetRecordKeys (see page 7794)	Retrieves a list of field values that the dataset adapter uses to locate records, along with information for formatting hidden parameters from this values.
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 7794)	Provides the underlying implementation for the InIterator method.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNextIteration (see page 24416)	Implements IteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 7795)	Provides the underlying implementation for the NotifyDeactivate method.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 7795)	Provides the underlying implementation for the SetEchoActionFieldValues method.
	ImplStartIterator (see page 7795)	Implements IteratorSupport.StartIterator to support iterating over the records of the adapter.

	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	LocateKeyParams (see page 7797)	Positions the dataset based on a specified set of field values.
	LocateRequestRecord (see page 7798)	Positions the dataset based on the parameters in the current request.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
V	Notification (see page 7798)	Responds when components are added to or deleted from the WebSnap application.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
V	OnDataSetStateChange (see page 7798)	Responds to changes in the associated dataset.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	PrepareDataSet (see page 7799)	Sets up the associated dataset for use by this adapter.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RestoreDefaultAdapterMode (see page 7801)	Returns the adapter to its default mode.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAdapterMode (see page 7801)	Sets the Mode (see page 7810) property based on a string that represents one of the available values.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDataSet (see page 7802)	Sets the value of the DataSet (see page 7807) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetMode (see page 7802)	Sets the value of the Mode (see page 7810) property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetPageSize (see page 24417)	Updates the pagesize variable.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SkipToPage (see page 24417)	Skips records until the current record is within the current page.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateFieldsExecuteValues (see page 7803)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 7804)	Indicates whether the values in the HTTP request change the value of any adapter fields.
V	UpdateFieldsUpdateValues (see page 7805)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 7805)	Validates the values passed in the current HTTP request for the adapter's fields.
V	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	AdapterRowNotFound (see page 7807)	Indicates the dataset adapter for which this adapter could not locate the current record.
	DataSetPrepared (see page 7808)	Indicates whether the associated dataset has been properly set up for use by this adapter.
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 24184)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24185)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24186)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24186)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 24187)	Supplies name/value pairs needed to execute an adapter action.
	OnGetEOF (see page 24420)	Implement this event to indicate when the current record index has exceeded the total record count.
	OnGetFirstRecord (see page 24420)	Implement this method to initialize the iteration index to zero.
	OnGetNextRecord (see page 24421)	Implement this method to increment the iteration index.
	OnGetRecordCount (see page 24421)	Implement this event to identify the total number of records that can be iterated.
	OnGetRecordIndex (see page 24421)	Implement this event to identify the zero-based current record.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.
	OnPrepareDataSet (see page 7810)	Occurs before the dataset adapter uses any information from its associated dataset.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	ApplyAdapterMode (see page 7780)	Sets the Mode (see page 7810) property to reflect the value that the current request specifies.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 7775)	Creates an initializes an instance of TCustomDataSetAdapter (see page 7766).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 7776)	Frees an instance of TCustomDataSetAdapter (see page 7766).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EncodeActionParams (see page 7780)	Writes name/value pairs that represent this adapter's state.
	EncodeActionParamsFlags (see page 7781)	Writes name/value pairs that represent a specified type of information.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExtractRequestParams (see page 7781)	Reads name/value pairs from the current HTTP request.
	ExtractRequestParamsFlags (see page 7782)	Reads name/value pairs that represent a specified type of information from the current HTTP request.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.

	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	Locate (see page 7796)	Positions the associated dataset on the record identified by the LocateParamsList (see page 7809) property.
	LocateAndApply (see page 7796)	Positions the dataset and updates the resulting record based on the parameters of the current request.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RaiseFieldChangedByAnotherUser (see page 7799)	Raises an EAdapterFieldException with a message indicating that another user changed a field value.
	RaiseFieldNotFound (see page 7800)	Raises an EAdapterException with a message indicating that a specified field could not be found.
	RaiseNilDataSet (see page 7800)	Raises an EAdapterException with a message indicating that the dataset is unassigned.
	RaiseRowNotFound (see page 7800)	Raises an EAdapterException with a message indicating that a dataset record could not be located based on the current set of key fields.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SilentLocate (see page 7802)	Positions the associated dataset on a specified record.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	StringToMode (see page 7803)	Converts a mode string to the corresponding TDataSetAdapterMode (see page 8111) value.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 24181)	Actions is the container for adapter actions components.

R	Actions (see page 24402)	Container for adapter actions components.
	AdapterMode (see page 19679)	This is AdapterMode, a member of class IAdapterMode.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	CurrentPage (see page 24418)	Sets or gets the current page index.
R	Data (see page 24181)	Data is the container for adapter field components.
R	Data (see page 24402)	Container for adapter field components.
	DataSet (see page 7807)	Specifies the dataset that this adapter represents.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DetailAdapters (see page 7808)	Lists the dataset adapters for all datasets that are details of this one.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
	InDefaultMode (see page 7808)	Indicates whether the dataset adapter is in its default mode.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	LocateParamsList (see page 7809)	Lists the parameters that identify the current record of this dataset and all other datasets linked to it in a master/detail relationship.
	MasterAdapter (see page 7809)	Indicates the dataset adapter, if any, to which this adapter is linked in a master/detail relationship.
	Mode (see page 7810)	Indicates how forms built from this dataset adapter operate on the information it provides.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
R	NeedAdapterMode (see page 19679)	This is NeedAdapterMode, a member of class IAdapterMode.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	PageCount (see page 24418)	Calculates the number of pages needed to display all adapter records.
	PageSize (see page 24419)	Sets or gets the page size variable.
R	RecordCount (see page 24419)	Returns the total number of records.
R	RecordIndex (see page 24419)	Returns the index of the current record.
	RefreshAfterPost (see page 7810)	Indicates whether the data set should be refreshed or not after posting..
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.4.2 TCustomDataSetAdapter.Create Constructor

Creates an initializes an instance of TCustomDataSetAdapter (see page 7766).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomDataSetAdapter(TComponent * AOwner);
```

Description

The TDataSetAdapterImageField constructor allocates memory for the adapter field object and initializes its properties.

AOwner is the component that is responsible for starting the streaming process when a module containing the adapter field is loaded or saved. It becomes the value of the Owner property. Typically, AOwner is the Web module.

See Also

Owner (see page 2807)

41.1.4.3 TCustomDataSetAdapter.Destroy Destructor

Frees an instance of TCustomDataSetAdapter (see page 7766).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomDataSetAdapter();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks whether the dataset adapter reference is nil before calling Destroy.

See Also

Free (see page 22300)

41.1.4.4 TCustomDataSetAdapter Methods

The methods of the TCustomDataSetAdapter class are listed here.

Protected Methods

	Name	Description
V	AddDefaultActions (see page 7778)	Adds the default action components to this adapter.
V	AddDefaultFields (see page 7779)	Adds the default field components to this adapter.
	FullyQualifyInputNames (see page 7782)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapterMode (see page 7783)	Returns a string that represents the current value of the Mode (see page 7810) property.
	GetAdapterModeNames (see page 7783)	Retrieves the strings for all the possible modes of this adapter.
	GetDataSet (see page 7784)	Returns the value of the DataSet (see page 7807) property.
V	GetDefaultMode (see page 7784)	Returns the default mode for this adapter.
	GetDesigntimeWarnings (see page 7784)	Populates a design-time warnings object with any warnings that the dataset adapter needs to emit.
V	GetEOF (see page 7785)	Indicates whether the adapter is positioned at the end of its dataset.
V	GetFirstRecord (see page 7785)	Positions the adapter on the first record of its dataset.
V	GetHiddenFieldOptions (see page 7786)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.

	GetKeyFields (see page 7786)	Returns a string listing the fields that this adapter uses to locate records.
	GetKeyParams (see page 7787)	Retrieves a list of fields that the dataset adapter uses to locate records, along with their current values.
	GetKeyParamStrings (see page 7787)	Retrieves a list of fields that the dataset adapter uses to locate records, along with their current values.
	GetMasterDetailAdapters (see page 7788)	Returns a list of all dataset adapters, including this one, linked to this adapter in a master/detail relationship.
	GetNeedAdapterMode (see page 7788)	Indicates whether the adapter requires a Mode (see page 7810) value.
V	GetNextRecord (see page 7789)	Positions the associated dataset on the next record.
V	GetRecordCount (see page 7789)	Returns the value of the RecordCount property.
V	GetRecordIndex (see page 7790)	Returns the value of the RecordIndex property.
	GetRecordKeys (see page 7790)	Retrieves a list of field values that the dataset adapter uses to locate records, along with information for formatting hidden parameters from this values.
V	ImplCanAddActionClass (see page 7790)	Provides the underlying implementation for the CanAddActionClass method.
V	ImplCanAddFieldClass (see page 7791)	Provides the underlying implementation for the CanAddFieldClass method.
V	ImplEndIterator (see page 7792)	Implements IteratorSupport's EndIterator method to support iterating over the data records of the adapter.
V	ImplGetActionsList (see page 7792)	Provides the underlying implementation for the GetActionsList method.
V	ImplGetFieldsList (see page 7793)	Provides the underlying implementation for the GetFieldsList method.
V	ImplGetIteratorIndex (see page 7793)	Provides the underlying implementation for the GetIteratorIndex method.
V	ImplGetRecordKeys (see page 7794)	Retrieves a list of field values that the dataset adapter uses to locate records, along with information for formatting hidden parameters from this values.
V	ImplInIterator (see page 7794)	Provides the underlying implementation for the InIterator method.
V	ImplNotifyDeactivate (see page 7795)	Provides the underlying implementation for the NotifyDeactivate method.
V	ImplSetEchoActionFieldValues (see page 7795)	Provides the underlying implementation for the SetEchoActionFieldValues method.
V	ImplStartIterator (see page 7795)	Implements IteratorSupport.StartIterator to support iterating over the records of the adapter.
	LocateKeyParams (see page 7797)	Positions the dataset based on a specified set of field values.
	LocateRequestRecord (see page 7798)	Positions the dataset based on the parameters in the current request.
V	Notification (see page 7798)	Responds when components are added to or deleted from the WebSnap application.
V	OnDataSetStateChange (see page 7798)	Responds to changes in the associated dataset.
V	PrepareDataSet (see page 7799)	Sets up the associated dataset for use by this adapter.
	RestoreDefaultAdapterMode (see page 7801)	Returns the adapter to its default mode.
	SetAdapterMode (see page 7801)	Sets the Mode (see page 7810) property based on a string that represents one of the available values.
	SetDataSet (see page 7802)	Sets the value of the DataSet (see page 7807) property.
V	SetMode (see page 7802)	Sets the value of the Mode (see page 7810) property.
V	UpdateFieldsExecuteValues (see page 7803)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 7804)	Indicates whether the values in the HTTP request change the value of any adapter fields.
V	UpdateFieldsUpdateValues (see page 7805)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 7805)	Validates the values passed in the current HTTP request for the adapter's fields.

Public Methods

	Name	Description
	ApplyAdapterMode (see page 7780)	Sets the Mode (see page 7810) property to reflect the value that the current request specifies.
V	EncodeActionParams (see page 7780)	Writes name/value pairs that represent this adapter's state.
	EncodeActionParamsFlags (see page 7781)	Writes name/value pairs that represent a specified type of information.
V	ExtractRequestParams (see page 7781)	Reads name/value pairs from the current HTTP request.
	ExtractRequestParamsFlags (see page 7782)	Reads name/value pairs that represent a specified type of information from the current HTTP request.
	Locate (see page 7796)	Positions the associated dataset on the record identified by the LocateParamsList (see page 7809) property.
	LocateAndApply (see page 7796)	Positions the dataset and updates the resulting record based on the parameters of the current request.

	RaiseFieldChangedByAnotherUser (see page 7799)	Raises an EAdapterFieldException with a message indicating that another user changed a field value.
	RaiseFieldNotFound (see page 7800)	Raises an EAdapterException with a message indicating that a specified field could not be found.
	RaiseNilDataSet (see page 7800)	Raises an EAdapterException with a message indicating that the dataset is unassigned.
	RaiseRowNotFound (see page 7800)	Raises an EAdapterException with a message indicating that a dataset record could not be located based on the current set of key fields.
	SilentLocate (see page 7802)	Positions the associated dataset on a specified record.
	StringToMode (see page 7803)	Converts a mode string to the corresponding TDataSetAdapterMode (see page 8111) value.

41.1.4.4.1 TCustomDataSetAdapter.AddDefaultActions Method

Adds the default action components to this adapter.

Pascal

```
procedure AddDefaultActions(AComponents: TWebComponentList); override;
```

C++

```
virtual __fastcall AddDefaultActions(TWebComponentList AComponents);
```

Description

This method is called when the dataset adapter needs to generate the list of actions it supports by default (when you do not explicitly assign a set of actions to the adapter).

AComponents is an existing TWebComponentList object that this method fills with the default dataset actions.

This method adds the following action components to AComponents:

TDataSetAdapterDeleteRowAction (see page 7934)

TDataSetAdapterFirstRowAction (see page 7972)

TDataSetAdapterPrevRowAction (see page 8042)

TDataSetAdapterNextRowAction (see page 8030)

TDataSetAdapterLastRowAction (see page 7984)

TDataSetAdapterEditRowAction (see page 7947)

TDataSetAdapterBrowseRowAction (see page 7903)

TDataSetAdapterNewRowAction (see page 8016)

TDataSetAdapterCancelRowAction (see page 7916)

TDataSetAdapterApplyRowAction (see page 7890)

TDataSetAdapterRefreshRowAction (see page 8054)

TAdapterPrevPageAction

TAdapterGotoPageAction

TAdapterNextPageAction

Note: The last three actions are only added if the value of the PageSize property is positive.

See Also

PageSize (see page 24419)

TDataSetAdapterDeleteRowAction (see page 7934)

TDataSetAdapterFirstRowAction (see page 7972)

TDataSetAdapterPrevRowAction (see page 8042)

TDataSetAdapterNextRowAction (see page 8030)

TDataSetAdapterLastRowAction (see page 7984)

TDataSetAdapterEditRowAction (see page 7947)

TDataSetAdapterBrowseRowAction (see page 7903)

TDataSEtAdapterNewRowAction

TDataSetAdapterCancelRowAction (see page 7916)

TDataSetAdapterApplyRowAction (see page 7890)

TDataSetAdapterRefreshRowAction (see page 8054)

TAdapterPrevPageAction (see page 23979)

TAdapterGotoPageAction (see page 23849)

TAdapterNextPageAction (see page 23945)

41.1.4.4.2 TCustomDataSetAdapter.AddDefaultFields Method

Adds the default field components to this adapter.

Pascal

```
procedure AddDefaultFields(AComponents: TWebComponentList); override;
```

C++

```
virtual __fastcall AddDefaultFields(TWebComponentList AComponents);
```

Description

This method is called when the dataset adapter needs to generate the list of fields it supports by default (when you do not explicitly assign a set of fields to the adapter).

AComponents is an existing TWebComponentList object that this method fills with the default dataset fields.

AddDefaultFields iterates through the Fields and AggFields properties of the associated dataset, adding an adapter field component for each field. Each adapter field component is of type TDataSetAdapterField (see page 7960), TDataSetAdapterMemoField (see page 7996), or TDataSetAdapterImageField, depending on the data type of the corresponding dataset field.

See Also

DataSet (see page 7807)

Fields (see page 7023)

AggFields (see page 7006)

TDataSetAdapterField (see page 7960)

TDataSetAdapterImageField (see page 8138)

TDataSetAdapterMemoField (see page 7996)

41.1.4.4.3 TCustomDataSetAdapter.ApplyAdapterMode Method

Sets the Mode (see page 7810) property to reflect the value that the current request specifies.

Pascal

```
procedure ApplyAdapterMode(AActionRequest: IActionRequest);
```

C++

```
__fastcall ApplyAdapterMode(IActionRequest AActionRequest);
```

Description

ApplyAdapterMode ensures that the dataset adapter is in the correct mode for handling the current HTTP request. If the current request specifies a mode for this dataset adapter, ApplyAdapterMode sets the Mode (see page 7810) property to that value. If the current request specifies a mode for any adapter linked to this one in a master/detail relationship, ApplyAdapterMode sets that adapter's Mode (see page 7810) property as well. If the current request does not specify a mode for this adapter (or one linked to it in a master/detail relationship), ApplyAdapterMode restores the adapter to its default mode (amBrowse).

AActionRequest is an interface to the current request object. ApplyAdapterMode uses this to obtain the mode that the user requested, if any.

See Also

Mode (see page 7810)

MasterAdapter (see page 7809)

DetailAdapters (see page 7808)

InDefaultMode (see page 7808)

41.1.4.4.4 TCustomDataSetAdapter.EncodeActionParams Method

Writes name/value pairs that represent this adapter's state.

Pascal

```
procedure EncodeActionParams(AParams: IAdapterItemRequestParams); override;
```

C++

```
virtual __fastcall EncodeActionParams(IAdapterItemRequestParams AParams);
```

Description

This method assembles the parameters (name/value pairs) that the adapter needs to add to an HTML form so that request messages that originate from that form can correctly restore this adapter's state. This allows the dataset adapter to recapture dataset state information (such as the current record) as well as the current page when the adapter supports paging.

AParams is the interface for an object to which the adapter writes parameter values.

ExtractRequestParams (see page 7781) calls on the EncodeActionParamsFlags (see page 7781) method to fill the list with parameters for locating the current record and current page.

See Also

ExtractRequestParams (see page 7781)

EncodeActionParamsFlags (see page 7781)

41.1.4.4.5 TCustomDataSetAdapter.EncodeActionParamsFlags Method

Writes name/value pairs that represent a specified type of information.

Pascal

```
procedure EncodeActionParamsFlags(AParams: IAdapterItemRequestParams; AOptions:
TDataSetAdapterRequestParamOptions);
```

C++

```
__fastcall EncodeActionParamsFlags(IAdapterItemRequestParams AParams,
TDataSetAdapterRequestParamOptions AOptions);
```

Description

This method assembles the parameters (name/value pairs) that the adapter needs to add to an HTML form to perform the task indicated by the AOptions parameter.

AParams is the interface for an object to which the adapter writes parameter values.

AOptions indicates the type of parameters to add to AParams.

When AOptions includes poLocateParams, EncodeActionParamsFlags adds a set of name/value pairs that represent dataset fields and their current values. This set of parameters describe the values of all fields in the index that can be used to locate the current record.

When AOptions includes poCurrentPage, EncodeActionParamsFlags adds a parameter for the current page number.

See Also

ExtractRequestParams (see page 7781)

EncodeActionParams (see page 7780)

41.1.4.4.6 TCustomDataSetAdapter.ExtractRequestParams Method

Reads name/value pairs from the current HTTP request.

Pascal

```
procedure ExtractRequestParams(ARequest: IUnknown); override;
```

C++

```
virtual __fastcall ExtractRequestParams(IUnknown ARequest);
```

Description

ExtractRequestParams reads the name/value pairs from the current request that were added to the HTML form from which that request originated using the EncodeActionParams (see page 7780) method. ExtractRequestParams stores the parameters that represent information for locating to the current record to the LocateParamsList (see page 7809) property and uses the parameters that describe the current page to set the CurrentPage property.

ARequest is an interface to the current request object. ExtractRequestParams uses this interface to extract the parameter information.

ExtractRequestParams calls the ExtractRequestParamsFlags (see page 7782) method to read the parameter values and respond to them.

See Also

EncodeActionParams (see page 7780)

ExtractRequestParamsFlags (see page 7782)

LocateParamsList (see page 7809)

CurrentPage (see page 24418)

41.1.4.4.7 TCustomDataSetAdapter.ExtractRequestParamsFlags Method

Reads name/value pairs that represent a specified type of information from the current HTTP request.

Pascal

```
procedure ExtractRequestParamsFlags(ARequest: IUnknown; AOptions:
TDataSetAdapterRequestParamOptions);
```

C++

```
__fastcall ExtractRequestParamsFlags(IUnknown ARequest, TDataSetAdapterRequestParamOptions
AOptions);
```

Description

ExtractRequestParamsFlags reads the name/value pairs specified by AOptions from the current request. These parameters (name/value pairs) were initially generated by a call to the EncodeActionParams (see page 7780) method when this adapter was contributing to the HTML form from which the request originated.

ARequest is an interface to the current request object. ExtractRequestParamsFlags uses this interface to extract the parameter information.

AOptions indicates the type of parameters to read.

When AOptions includes poLocateParams, ExtractRequestParamsFlags reads the name/value pairs that represent dataset fields and their current values, and adds them to the LocateParamsList (see page 7809) property. This set of parameters describe the values of all fields in the index that can be used to locate the current record.

When AOptions includes poCurrentPage, ExtractRequestParamsFlags reads the parameter for the current page number and uses it to set the CurrentPage property.

See Also

EncodeActionParams (see page 7780)

ExtractRequestParams (see page 7781)

LocateParamsList (see page 7809)

CurrentPage (see page 24418)

41.1.4.4.8 TCustomDataSetAdapter.FullyQualifyInputNames Method

Indicates whether the HTML input names must be qualified with the name of the adapter.

Pascal

```
function FullyQualifyInputNames: Boolean;
```

C++

```
__fastcall Boolean FullyQualifyInputNames();
```

Description

FullyQualifyInputNames returns true if the names of input elements in generated HTML forms that represent the fields of this dataset adapter should include the name of this adapter. Field names must be qualified when the HTML form may contain multiple fields with the same name (that is, when the form contains input elements for multiple adapters).

This method checks whether this dataset adapter is part of a master/detail relationship (whether the MasterAdapter (see page

7809) or DetailAdapters (see page 7808) property indicates such a relationship). If so, it returns true. If this adapter is not part of a master/detail relationship, FullyQualifyInputNames returns false.

See Also

FullyQualifyInputName (see page 23932)

MasterAdapter (see page 7809)

DetailAdapters (see page 7808)

41.1.4.4.9 TCustomDataSetAdapter.GetAdapterMode Method

Returns a string that represents the current value of the Mode (see page 7810) property.

Pascal

```
function GetAdapterMode: string;
```

C++

```
__fastcall AnsiString GetAdapterMode();
```

Description

GetAdapterMode returns the string that corresponds to the current value of the Mode (see page 7810) property. If the mode for this adapter has not been explicitly set in response to a request message(that is, if InDefaultMode (see page 7808) is true), GetAdapterMode returns an empty string.

This method allows server-side script to read the Mode (see page 7810) property of this adapter.

See Also

Mode (see page 7810)

StringToMode (see page 7803)

GetAdapterModeNames (see page 7783)

InDefaultMode (see page 7808)

41.1.4.4.10 TCustomDataSetAdapter.GetAdapterModeNames Method

Retrieves the strings for all the possible modes of this adapter.

Pascal

```
procedure GetAdapterModeNames(AStrings: TStrings);
```

C++

```
__fastcall GetAdapterModeNames(TStrings AStrings);
```

Description

GetAdapterModeNames adds the names of all possible values of the Mode (see page 7810) property to the specified string list. At design time, this method is used to generate a drop-down list of adapter modes.

AStrings is an existing string list to fill with the names of the adapter modes.

GetAdapterModeNames adds the following names to AStrings:

Insert

Edit

Browse

Query

See Also

Mode (see page 7810)

StringToMode (see page 7803)

GetAdapterMode (see page 7783)

41.1.4.4.11 TCustomDataSetAdapter.GetDataSet Method

Returns the value of the DataSet (see page 7807) property.

Pascal

```
function GetDataSet: TDataSet;
```

C++

```
__fastcall TDataSet GetDataSet();
```

Description

GetDataSet is the protected read implementation for the DataSet (see page 7807) property. If necessary, GetDataSet calls the PrepareDataSet (see page 7799) method before returning the value of the DataSet (see page 7807) property. This ensures that any necessary initialization has occurred before the application tries to use DataSet (see page 7807).

See Also

DataSet (see page 7807)

PrepareDataSet (see page 7799)

41.1.4.4.12 TCustomDataSetAdapter.GetDefaultMode Method

Returns the default mode for this adapter.

Pascal

```
function GetDefaultMode: TDataSetAdapterMode; virtual;
```

C++

```
virtual __fastcall TDataSetAdapterMode GetDefaultMode();
```

Description

GetDefaultMode returns the value of the Mode (see page 7810) property when InDefaultMode (see page 7808) is true. This is the dataset adapter mode when the current request does not specify a mode for this adapter.

In TCustomDataSetAdapter (see page 7766), GetDefaultMode returns amBrowse.

See Also

InDefaultMode (see page 7808)

Mode (see page 7810)

41.1.4.4.13 TCustomDataSetAdapter.GetDesigntimeWarnings Method

Populates a design-time warnings object with any warnings that the dataset adapter needs to emit.

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

GetDesigntimeWarnings is called automatically at design time when the adapter page producer is assembling the current version of its HTML form. This gives the dataset adapter an opportunity to add messages to the version of the HTML form that appears at design time, describing problems that you can rectify.

AWarnings is the design-time warnings object that assembles design-time warning messages that appear in the HTML form.

If the DataSet (see page 7807) property has not been set, GetDesigntimeWarnings adds a message that indicates this omission. If DataSet (see page 7807) is set, but the dataset is not active, GetDesigntimeWarnings adds a message that indicates this problem. If DataSet (see page 7807) is set, the dataset is active, but you have not specified the key fields that should be used to locate records in that dataset, GetDesigntimeWarnings adds a message indicating this omission.

See Also

DataSet (see page 7807)

Active (see page 7004)

GetKeyFields (see page 7786)

41.1.4.4.14 TCustomDataSetAdapter.GetEOF Method

Indicates whether the adapter is positioned at the end of its dataset.

Pascal

```
function GetEOF: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetEOF();
```

Description

GetEOF indicates whether there are any more records after the current one. If the adapter's dataset is positioned on its last record (or if the DataSet (see page 7807) property is not set), GetEOF returns true. If the GetNextRecord (see page 7789) method can move to a new record, GetEOF returns false.

See Also

Eof (see page 7015)

RecordIndex

RecordCount

DataSet (see page 7807)

GetNextRecord (see page 7789)

41.1.4.4.15 TCustomDataSetAdapter.GetFirstRecord Method

Positions the adapter on the first record of its dataset.

Pascal

```
function GetFirstRecord: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetFirstRecord();
```

Description

GetFirstRecord calls the First method of the associated dataset and returns true if it successfully positions the dataset on its first record.

See Also

First (see page 6942)

DataSet (see page 7807)

41.1.4.4.16 TCustomDataSetAdapter.GetHiddenFieldOptions Method

Indicates the types of hidden fields that this adapter requires in a submitted HTML form.

Pascal

```
function GetHiddenFieldOptions: TAdapterHiddenFieldOptions; override;
```

C++

```
virtual __fastcall TAdapterHiddenFieldOptions GetHiddenFieldOptions();
```

Description

GetHiddenFieldOptions returns a set of flags that indicate what hidden information this adapter adds to HTML forms. The return value depends on the current value of the Mode (see page 7810) property. When Mode (see page 7810) is amEdit, GetHiddenFieldOptions returns the set [hoMode, hoOrigValue, hoRecordKeys]. For any other Mode (see page 7810), GetHiddenFieldOptions returns a set with only the hoMode flag.

See Also

listhiddenfields (see page 24173)

ListHiddenRecordFields (see page 24173)

Mode (see page 7810)

41.1.4.4.17 TCustomDataSetAdapter.GetKeyFields Method

Returns a string listing the fields that this adapter uses to locate records.

Pascal

```
function GetKeyFields: string;
```

C++

```
__fastcall AnsiString GetKeyFields();
```

Description

GetKeyFields generates the list of fields that are required to uniquely identify records in this adapter's dataset. These fields and their values are then used to create hidden parameters that are added to HTML forms. When the adapter receives requests from forms containing those hidden parameters, it assigns the parameters to the LocateParamsList (see page 7809) property.

GetKeyFields returns a semicolon-delimited list of fields that identify records in the adapter's dataset.

This method first queries all of its child field components, building a list of all fields that include the ffInKey flag in their FieldFlags property. If none of the child field components include the ffInKey flag, GetKeyFields obtains an IProviderSupport interface from the associated dataset and calls its PSGetKeyFields method.

See Also

FieldFlags (see page 7752)

PSGetKeyFields (see page 7581)

LocateParamsList (see page 7809)

GetKeyParams (see page 7787)

GetKeyParamStrings (see page 7787)

41.1.4.4.18 TCustomDataSetAdapter.GetKeyParams Method

Retrieves a list of fields that the dataset adapter uses to locate records, along with their current values.

Pascal

```
procedure GetKeyParams(AParams: TLocateParamsList; ARecurse: Boolean);
```

C++

```
__fastcall GetKeyParams(TLocateParamsList AParams, Boolean ARecurse);
```

Description

GetKeyParams fills out a TLocateParamsList (see page 8096) object with the current values of all fields that the adapter uses to uniquely identify records. These name/value pairs are then added as hidden fields to HTML forms. When the adapter receives requests from forms containing those hidden parameters, it assigns them to the LocateParamsList (see page 7809) property.

Params is a collection of TLocateParams (see page 8090) objects, where each TLocateParams (see page 8090) object lists the key fields and their values for a single dataset adapter.

ARecurse indicates whether GetKeyParams should return values for just this adapter, or to include values from all adapters linked to this one in a master/detail relationship as well.

If ARecurse is false, GetKeyParams calls the GetKeyParamStrings (see page 7787) method to obtain the key fields and their values for this dataset. It then adds these name/value pairs to Params. If ARecurse is true, GetKeyParams calls the GetKeyParams method (with ARecurse set to false) for this adapter and for every adapter linked to it in a master/detail relationship.

See Also

GetKeyParams (see page 7743)

LocateParamsList (see page 7809)

GetKeyParamStrings (see page 7787)

MasterAdapter (see page 7809)

DetailAdapters (see page 7808)

41.1.4.4.19 TCustomDataSetAdapter.GetKeyParamStrings Method

Retrieves a list of fields that the dataset adapter uses to locate records, along with their current values.

Pascal

```
procedure GetKeyParamStrings(AValues: TStrings);
```

C++

```
__fastcall GetKeyParamStrings(TStrings AValues);
```

Description

GetKeyParamStrings fills out a TStrings object with the current values of all fields in this adapter's dataset that are needed to uniquely identify records. These name/value pairs are then added as hidden fields to HTML forms. When the adapter receives requests from forms containing those hidden parameters, it assigns them to the LocateParamsList (see page 7809) property.

Unlike the GetKeyParams (see page 7787) method, GetKeyParamStrings retrieves only the key field names and values for this adapter's dataset, and not for any other datasets that may be linked to it in a master/detail relationship.

AValues is a string list to fill with the names and values of all key fields in this adapters dataset.

GetKeyParamStrings calls GetKeyFields (see page 7786) method to obtain a list of fields for which to create parameters. For each key field, it generates a string of the form 'name=value', where name is the field name and value is the value of the field's AsString property.

See Also

GetKeyName (see page 7742)

LocateParamsList (see page 7809)

GetKeyParams (see page 7787)

AsString (see page 7151)

41.1.4.4.20 TCustomDataSetAdapter.GetMasterDetailAdapters Method

Returns a list of all dataset adapters, including this one, linked to this adapter in a master/detail relationship.

Pascal

```
function GetMasterDetailAdapters: TComponentList;
```

C++

```
__fastcall TComponentList GetMasterDetailAdapters();
```

Description

GetMasterDetailAdapters allocates a new TComponentList object and fills it with every dataset adapter that is linked to this one in a master/detail relationship. The adapters are listed from the top-level adapter down, using a depth-first search.

DataSetAdapters use the returned list internally to guide processes that include all datasets linked to this one in a master/detail relationship.

Note: It is the callers responsibility to free the component list that this method returns.

See Also

Adapter (see page 7752)

DetailAdapters (see page 7808)

41.1.4.4.21 TCustomDataSetAdapter.GetNeedAdapterMode Method

Indicates whether the adapter requires a Mode (see page 7810) value.

Pascal

```
function GetNeedAdapterMode: Boolean;
```

C++

```
__fastcall Boolean GetNeedAdapterMode();
```

Description

GetNeedAdapterMode returns true, indicating that dataset adapters must always have a specific mode setting to operate properly. Note that this mode need not be explicitly assigned because Dataset adapters use the amBrowse mode setting if the application does not assign a value to the Mode (see page 7810) property.

See Also

Mode (see page 7810)

41.1.4.4.22 TCustomDataSetAdapter.GetNextRecord Method

Positions the associated dataset on the next record.

Pascal

```
function GetNextRecord: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetNextRecord();
```

Description

GetNextRecord calls the Next method of the associated dataset and returns true if it successfully moves to the next record in the dataset.

See Also

Next (see page 6973)

DataSet (see page 7807)

41.1.4.4.23 TCustomDataSetAdapter.GetRecordCount Method

Returns the value of the RecordCount property.

Pascal

```
function GetRecordCount: Integer; override;
```

C++

```
virtual __fastcall int GetRecordCount();
```

Description

GetRecordCount is the protected read implementation of the RecordCount property.

GetRecordCount first generates an OnGetRecordCount event, and returns the number of records that the event handler provides. If there is no OnGetRecordCount event handler, GetRecordCount returns the value of the associated dataset's RecordCount property.

See Also

OnGetRecordCount (see page 24421)

RecordCount

RecordCount (see page 7032)

41.1.4.4.24 TCustomDataSetAdapter.GetRecordIndex Method

Returns the value of the RecordIndex property.

Pascal

```
function GetRecordIndex: Integer; override;
```

C++

```
virtual __fastcall int GetRecordIndex();
```

Description

GetRecordIndex is the protected read implementation of the RecordIndex property.

This method returns an internally maintained index that indicates where the dataset is positioned.

See Also

RecordIndex

41.1.4.4.25 TCustomDataSetAdapter.GetRecordKeys Method

Retrieves a list of field values that the dataset adapter uses to locate records, along with information for formatting hidden parameters from this values.

Pascal

```
procedure GetRecordKeys(AValues: TStrings; var AFullyQualify: Boolean);
```

C++

```
__fastcall GetRecordKeys(TStrings AValues, Boolean AFullyQualify);
```

Description

GetRecordKeys fills out a TStrings object with the current values of all fields in this adapter's dataset that are needed to uniquely identify records. These name/value pairs are then added as hidden fields to HTML forms. GetRecordKeys also returns an indication of whether field names should be qualified by the name of the adapter when formatting the hidden fields for inclusion in the HTML form.

AValues is a string list to fill with the names and values of all key fields in this adapters dataset.

AFullyQualify returns true if the field names must be qualified by the name of this adapter.

GetRecordKeys delegates the task of providing values for AValues and AFullyQualify to the ImplGetRecordKeys (see page 7794) method.

See Also

ImplGetRecordKeys (see page 7794)

41.1.4.4.26 TCustomDataSetAdapter.ImplCanAddActionClass Method

Provides the underlying implementation for the CanAddActionClass method.

Pascal

```
function ImplCanAddActionClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddActionClass method calls ImplCanAddActionClass to determine whether a specified class of adapter action can be added as a child to an adapter of this type.

AParent is a dataset adapter to which a child action may be added. It must be a descendant of TCustomDataSetAdapter (see page 7766).

AClass is the class of an adapter action component that may be added to AParent.

ImplCanAddActionClass returns true if an action of type AClass can be added to AParent, false if AClass is not a valid action class for AParent.

In TCustomDataSetAdapter (see page 7766), ImplCanAddActionClass returns true if AClass descends from TCustomPagedAdapterAction or TCustomDataSetAdapterAction (see page 7811). These are the classes that the AddDefaultActions (see page 7778) method returns. For a complete list of action classes, see the topic for AddDefaultActions (see page 7778).

See Also

CanAddActionClass (see page 24150)

TCustomDataSetAdapterAction (see page 7811)

AddDefaultActions (see page 7778)

TCustomPagedAdapterAction (see page 24371)

41.1.4.4.27 TCustomDataSetAdapter.ImplCanAddFieldClass Method

Provides the underlying implementation for the CanAddFieldClass method.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddFieldClass method calls ImplCanAddFieldClass to determine whether a specified class of adapter field can be added as a child to an adapter of this type.

AParent is a dataset adapter to which a child field may be added. It must be a descendant of TCustomDataSetAdapter (see page 7766).

AClass is the class of an adapter field component that may be added to AParent.

ImplCanAddFieldClass returns true if a field of type AClass can be added to AParent, false if AClass is not a valid field class for AParent.

In TCustomDataSetAdapter (see page 7766), ImplCanAddFieldClass returns true if AClass supports the IDataSetAdapterFieldClass (see page 8104) interface. This includes the descendants of TBaseDataSetAdapterField (see page 7729) (TDataSetAdapterField (see page 7960), TDataSetAdapterMemoField (see page 7996), and TDataSetAdapterImageField).

See Also

CanAddFieldClass (see page 24150)

TBaseDataSetAdapterField (see page 7729)

TDataSetAdapterField (see page 7960)

TDataSetAdapterMemoField (see page 7996)

TDataSetAdapterImageField (see page 8138)

41.1.4.4.28 TCustomDataSetAdapter.ImplEndIterator Method

Implements IteratorSupport's EndIterator method to support iterating over the data records of the adapter.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer); override;
```

C++

```
virtual __fastcall ImplEndIterator(void * OwnerData);
```

Description

EndIterator is called when record iteration is complete.

See Also

enditerator (see page 19723)

41.1.4.4.29 TCustomDataSetAdapter.ImplGetActionsList Method

Provides the underlying implementation for the GetActionsList method.

Pascal

```
procedure ImplGetActionsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetActionsList(TStrings AList);
```

Description

The GetActionsList method calls ImplGetActionsList to obtain the names of all actions that can be added to this adapter. The action names are the names by which server-side script refers to this adapter's actions.

AList is an existing string list to fill with the names of valid actions.

GetActionsList adds the following strings to AList (with the corresponding action class references as associated objects):

- NextRow
- PrevRow
- LastRow
- FirstRow
- NewRow
- Cancel
- Apply
- RefreshRow
- EditRow
- BrowseRow
- PrevPage
- GotoPage

- NextPage

Note: The last three actions are only added if this adapter supports paging (if the value of the PageSize property is positive).

See Also

GetActionsList (see page 24156)

AddDefaultActions (see page 7778)

PageSize (see page 24419)

41.1.4.4.30 TCustomDataSetAdapter.ImplGetFieldsList Method

Provides the underlying implementation for the GetFieldsList method.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

The GetFieldsList method calls ImplGetFieldsList to obtain the names of all fields that can be added to this adapter. The field names are the names by which server-side script refers to this adapter's fields.

AList is an existing string list to fill with the names of valid fields.

GetFieldsList adds the names of all fields listed by the Fields and AggFields properties of the associated dataset. It adds class references for the appropriate adapter field components that represent each field as associated objects.

See Also

GetFieldsList (see page 24159)

AddDefaultFields (see page 7779)

Fields (see page 7023)

AggFields (see page 7006)

41.1.4.4.31 TCustomDataSetAdapter.ImplGetIteratorIndex Method

Provides the underlying implementation for the GetIteratorIndex method.

Pascal

```
function ImplGetIteratorIndex: Integer; override;
```

C++

```
virtual __fastcall int ImplGetIteratorIndex();
```

Description

The GetIteratorIndex method calls ImplGetIteratorIndex to obtain the current index that arises when server-side script uses an enumerator over the records of this adapter.

GetIteratorIndex returns the value of the RecordIndex property.

See Also

GetIteratorIndex (see page 24160)

RecordIndex

41.1.4.4.32 TCustomDataSetAdapter.ImplGetRecordKeys Method

Retrieves a list of field values that the dataset adapter uses to locate records, along with information for formatting hidden parameters from this values.

Pascal

```
procedure ImplGetRecordKeys(AValues: TStrings; var AFullyQualify: Boolean); virtual;
```

C++

```
virtual __fastcall ImplGetRecordKeys(TStrings AValues, Boolean AFullyQualify);
```

Description

The GetRecordKeys (see page 7790) method calls ImplGetRecordKeys to obtain the current values of all fields in this adapter's dataset that are needed to uniquely identify records and an indication of whether field names should be qualified by the name of the adapter when formatting hidden fields that describe this information.

AValues is a string list to fill with the names and values of all key fields in this adapters dataset.

AFullyQualify returns true if the field names must be qualified by the name of this adapter.

ImplGetRecordKeys calls the GetKeyParamStrings (see page 7787) method to fill AValues with the names and values of key fields. It then calls the FullyQualifyInputNames (see page 7782) method to set the value of AFullyQualify.

See Also

GetRecordKeys (see page 7790)

GetKeyParamStrings (see page 7787)

FullyQualifyInputNames (see page 7782)

41.1.4.4.33 TCustomDataSetAdapter.ImplInIterator Method

Provides the underlying implementation for the InIterator method.

Pascal

```
function ImplInIterator: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplInIterator();
```

Description

The InIterator method calls ImplInIterator to determine whether server-side script is using an enumerator to iterate over the records of this adapter.

InIterator returns true if the GetFirstRecord (see page 7785) method has been called. GetFirstRecord (see page 7785) is the first method that is called when an enumerator is applied to this adapter's records.

See Also

InIterator (see page 24172)

GetFirstRecord (see page 7785)

41.1.4.4.34 TCustomDataSetAdapter.ImplNotifyDeactivate Method

Provides the underlying implementation for the NotifyDeactivate method.

Pascal

```
procedure ImplNotifyDeactivate; override;
```

C++

```
virtual __fastcall ImplNotifyDeactivate();
```

Description

The NotifyDeactivate method calls ImplNotifyDeactivate to respond when the Web module that contains this adapter is deactivated.

ImplNotifyDeactivate clears any properties whose values should not affect the processing of the next request message. Typically, these properties have values that are set in response to parameters in the request message. By clearing them when the Web module is deactivated, the adapter ensures that the current settings do not contaminate the handling of the next request message if it does not include a parameter value for a specific property.

See Also

NotifyDeactivate (see page 24175)

41.1.4.4.35 TCustomDataSetAdapter.ImplSetEchoActionFieldValues Method

Provides the underlying implementation for the SetEchoActionFieldValues method.

Pascal

```
procedure ImplSetEchoActionFieldValues(AValue: Boolean); override;
```

C++

```
virtual __fastcall ImplSetEchoActionFieldValues(Boolean AValue);
```

Description

The SetEchoActionFieldValues method calls ImplSetEchoActionFieldValues to specify whether this adapter's fields should obtain their values from the current request message or from the associated dataset fields.

AValue indicates where this adapter's fields obtain their values. When AValue is true, the field values come from the current request message. When AValue is false, the field values come from the associated dataset fields.

ImplSetEchoActionFieldValues sets the EchoActionFieldValue property to AValue on every field that is a child of this adapter, plus every field that is a child of an adapter linked to this one in a master/detail relationship. The one exception is that the EchoActionFieldValue property is never set to true on fields whose parent adapter is in browse mode.

See Also

SetEchoActionFieldValues (see page 24176)

EchoActionFieldValue (see page 23944)

Mode (see page 7810)

41.1.4.4.36 TCustomDataSetAdapter.ImplStartIterator Method

Implements IteratorSupport.StartIterator to support iterating over the records of the adapter.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

ImplStartIterator locates the first record in the current page using the GetFirstRecord (see page 7785) and SkipToPage methods.

See Also

startiterator (see page 19724)

ImplNextIteration (see page 24416)

GetFirstRecord (see page 7785)

SkipToPage (see page 24417)

CurrentPage (see page 24418)

41.1.4.4.37 TCustomDataSetAdapter.Locate Method

Positions the associated dataset on the record identified by the LocateParamsList (see page 7809) property.

Pascal

```
function Locate: Boolean;
```

C++

```
__fastcall Boolean Locate();
```

Description

Locate tries to position this adapter's dataset, and the dataset of any adapter linked to this one in a master/detail relationship, to the record identified by the key fields that the LocateParamsList (see page 7809) property lists.

Locate returns true if it successfully positions all datasets on the specified record. Locate returns false if it was unable to position one or more datasets on the specified record. If Locate returns false, it adds a message to the Errors property describing the problem.

See Also

LocateParamsList (see page 7809)

Errors (see page 24182)

SilentLocate (see page 7802)

41.1.4.4.38 TCustomDataSetAdapter.LocateAndApply Method

Positions the dataset and updates the resulting record based on the parameters of the current request.

Pascal

```
function LocateAndApply(AActionRequest: IActionRequest): Boolean;
```

C++

```
__fastcall Boolean LocateAndApply(IActionRequest AActionRequest);
```

Description

LocateAndApply performs the following tasks:

1.Extracts the current request parameters from the current request, using them to set the Mode (see page 7810) and LocateParamsList (see page 7809) properties.

2.Positions the dataset (and any dataset linked to it in a master/detail relationship) on the record that the request parameters indicate.

3.If the request put the adapter in edit or insert mode, update fields to reflect the values in the request and post the current record.

4.If the adapter is in insert mode, change the mode to edit now that the new record has been inserted into the dataset.

AActionRequest is an interface to the current request object.

LocateAndApply returns true if it successfully positions the dataset and applies any specified edits. LocateAndApply returns false if it was unable to perform these tasks. If LocateAndApply returns false, it adds a message to the Errors property describing the problem.

See Also

Locate (see page 7796)

Errors (see page 24182)

LocateParamsList (see page 7809)

Mode (see page 7810)

41.1.4.4.39 TCustomDataSetAdapter.LocateKeyParams Method

Positions the dataset based on a specified set of field values.

Pascal

```
function LocateKeyParams(AParams: IKeyParams): Boolean;
```

C++

```
__fastcall Boolean LocateKeyParams(IKeyParams AParams);
```

Description

All other methods that position the associated dataset on a specified record call LocateKeyParams (directly or indirectly) to move to the specified record.

AParams is an interface from which to obtain the names and values of dataset fields to use when locating the desired record.

LocateKeyParams returns true if it successfully positions the dataset on the specified record, false if it was unable to perform this task.

This method generates a call to the associated dataset's Locate (see page 7796) method, based on the field values specified using AParams.

See Also

Locate (see page 7796)

SilentLocate (see page 7802)

LocateAndApply (see page 7796)

Locate (see page 6970)

41.1.4.4.40 TCustomDataSetAdapter.LocateRequestRecord Method

Positions the dataset based on the parameters in the current request.

Pascal

```
function LocateRequestRecord(AActionRequest: IActionRequest): Boolean;
```

C++

```
__fastcall Boolean LocateRequestRecord(IActionRequest AActionRequest);
```

Description

LocateRequestRecord tries to position this adapter's dataset on the record identified by parameters it extracts from the current request. Unlike the Locate (see page 7796) method, LocateRequestRecord does not use or set the LocateParamsList (see page 7809) property, nor does it concern itself with any datasets linked in a master/detail relationship.

AActionRequest is an interface to the request object from which LocateRequestRecord obtains key field parameters.

LocateRequestRecord returns true if it successfully positions the dataset on the specified record. LocateRequestRecord returns false if it was unable to position the dataset on the specified record. If LocateRequestRecord returns false, it adds a message to the Errors property describing the problem.

See Also

LocateParamsList (see page 7809)

Errors (see page 24182)

Locate (see page 7796)

41.1.4.4.41 TCustomDataSetAdapter.Notification Method

Responds when components are added to or deleted from the WebSnap application.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are created or destroyed. After calling the inherited method, Notification checks whether the object that that is the value of the DataSet (see page 7807) or MasterAdapter (see page 7809) property is about to be destroyed. If so, Notification sets the appropriate property to nil (Delphi) or NULL (C++) and performs any other adjustments to respond to the change.

See Also

MasterAdapter (see page 7809)

DataSet (see page 7807)

41.1.4.4.42 TCustomDataSetAdapter.OnDataSetStateChange Method

Responds to changes in the associated dataset.

Pascal

```
procedure OnDataSetStateChange(Sender: TObject); virtual;
```

C++

```
virtual __fastcall OnDataSetStateChange(TObject * Sender);
```

Description

OnDataSetStateChange is called automatically in response to changes in the associated dataset. It allows the adapter to clear any information (such as the set of default fields) that depend on the current state of the dataset.

Sender is a datalink object that handles the communication between the adapter and its dataset.

See Also

DataSet (see page 7807)

TDataLink (see page 6876)

41.1.4.4.43 TCustomDataSetAdapter.PrepareDataSet Method

Sets up the associated dataset for use by this adapter.

Pascal

```
procedure PrepareDataSet; virtual;
```

C++

```
virtual __fastcall PrepareDataSet();
```

Description

TCustomDataSetAdapter (see page 7766) calls PrepareDataSet before it reads the DataSet (see page 7807) property the first time after the dataset enters a new mode. This method performs any necessary tasks for initializing the dataset for use by the adapter in the current mode.

PrepareDataSet generates an OnPrepareDataSet (see page 7810) event to allow you to perform your own initializations. It then cancels any pending edits and informs any fields that are used in the current mode so that they can perform their own initializations.

Note: Do not confuse the PrepareDataSet property with the Prepare method of many dataset components. PrepareDataSet allows the adapter to performed any tasks it requires to set up the dataset for its own use in the current mode. The Prepare method of a dataset component instructs the underlying database engine to initialize itself to represent a database query.

See Also

OnPrepareDataSet (see page 7810)

DataSetPrepared (see page 7808)

DataSet (see page 7807)

Mode (see page 7810)

PrepareDataSet (see page 7750)

41.1.4.4.44 TCustomDataSetAdapter.RaiseFieldChangedByAnotherUser Method

Raises an EAdapterFieldException with a message indicating that another user changed a field value.

Pascal

```
procedure RaiseFieldChangedByAnotherUser(const AFieldName: string);
```

C++

```
__fastcall RaiseFieldChangedByAnotherUser(const AnsiString AFieldName);
```

Description

RaiseFieldChangedByAnotherUser raises an EAdapterFieldException for the situation where the values of this adapter's fields have changed and the WebSnap application expects them to be the same.

AFieldName is the name of the field that has changed. It is used to generate an error message for the exception.

See Also

EAdapterFieldException (see page 23698)

41.1.4.4.45 TCustomDataSetAdapter.RaiseFieldNotFound Method

Raises an EAdapterException with a message indicating that a specified field could not be found.

Pascal

```
procedure RaiseFieldNotFound(const AFieldName: string);
```

C++

```
__fastcall RaiseFieldNotFound(const AnsiString AFieldName);
```

Description

RaiseFieldNotFound raises an EAdapterException for the situation where the request needs to perform some action on a specified field in the dataset and can't locate the specified field. For example, this method is called when the current request tries to update a field and the dataset does not include a field with the specified name.

AFieldName is the name of the field that can't be located. It is used to generate an error message for the exception.

See Also

EAdapterException (see page 23696)

41.1.4.4.46 TCustomDataSetAdapter.RaiseNilDataSet Method

Raises an EAdapterException with a message indicating that the dataset is unassigned.

Pascal

```
procedure RaiseNilDataSet;
```

C++

```
__fastcall RaiseNilDataSet();
```

Description

RaiseNilDataSet raises an EAdapterException for the situation where the DataSet (see page 7807) property is found to be nil (Delphi) or NULL (C++).

See Also

EAdapterException (see page 23696)

DataSet (see page 7807)

41.1.4.4.47 TCustomDataSetAdapter.RaiseRowNotFound Method

Raises an EAdapterException with a message indicating that a dataset record could not be located based on the current set of

key fields.

Pascal

```
procedure RaiseRowNotFound;
```

C++

```
__fastcall RaiseRowNotFound();
```

Description

RaiseRowNotFound raises an EAdapterException for the situation where the adapter was unable to locate a record. This can arise, for example, when another user has deleted a record or changed a key field value. The problem record may belong to this adapter's dataset or to another dataset that is linked to it in a master/detail relationship. RaiseRowNotFound uses the AdapterRowNotFound (see page 7807) property to indicate the name of the adapter whose dataset could not be positioned on a specified row.

See Also

EAdapterException (see page 23696)

AdapterRowNotFound (see page 7807)

41.1.4.4.48 TCustomDataSetAdapter.RestoreDefaultAdapterMode Method

Returns the adapter to its default mode.

Pascal

```
procedure RestoreDefaultAdapterMode;
```

C++

```
__fastcall RestoreDefaultAdapterMode();
```

Description

RestoreDefaultAdapterMode sets the InDefaultMode (see page 7808) property to true, thereby restoring the Mode (see page 7810) property to its default value (amBrowse).

See Also

Mode (see page 7810)

InDefaultMode (see page 7808)

41.1.4.4.49 TCustomDataSetAdapter.SetAdapterMode Method

Sets the Mode (see page 7810) property based on a string that represents one of the available values.

Pascal

```
procedure SetAdapterMode(const AValue: string);
```

C++

```
__fastcall SetAdapterMode(const AnsiString AValue);
```

Description

SetAdapterMode interprets the specified string and uses it to set the Mode (see page 7810) property.

AValue is a string that indicates the desired mode.

If AValue is an empty string, SetAdapterMode sets the Mode (see page 7810) property to the default value (by setting InDefaultMode (see page 7808) to true). Otherwise, SetAdapterMode calls the StringToMode (see page 7803) method to

convert AValue into a Mode (see page 7810) value and assigns the result to the Mode (see page 7810) property.

This method allows server-side script to set the Mode (see page 7810) property of this adapter.

See Also

Mode (see page 7810)

StringToMode (see page 7803)

GetAdapterModeNames (see page 7783)

InDefaultMode (see page 7808)

41.1.4.4.50 TCustomDataSetAdapter.SetDataSet Method

Sets the value of the DataSet (see page 7807) property.

Pascal

```
procedure SetDataSet(ADataSet: TDataSet);
```

C++

```
__fastcall SetDataSet(TDataSet ADataSet);
```

Description

SetDataSet is the protected write implementation of the DataSet (see page 7807) property.

See Also

DataSet (see page 7807)

41.1.4.4.51 TCustomDataSetAdapter.SetMode Method

Sets the value of the Mode (see page 7810) property.

Pascal

```
procedure SetMode(const Value: TDataSetAdapterMode); virtual;
```

C++

```
virtual __fastcall SetMode(const TDataSetAdapterMode Value);
```

Description

SetMode is the protected write implementation of the Mode (see page 7810) property. As a side-effect, SetMode sets the InDefaultMode (see page 7808) and DataSetPrepared (see page 7808) properties to false.

See Also

Mode (see page 7810)

InDefaultMode (see page 7808)

DataSetPrepared (see page 7808)

41.1.4.4.52 TCustomDataSetAdapter.SilentLocate Method

Positions the associated dataset on a specified record.

Pascal

```
function SilentLocate(AParams: TLocateParamsList; ARecurse: Boolean): Boolean;
```

C++

```
__fastcall Boolean SilentLocate(TLocateParamsList AParams, Boolean ARecurse);
```

Description

SilentLocate tries to position this adapter's dataset on the record identified by a specified set of key field values.

Params is a collection of TLocateParams (see page 8090) objects, where each TLocateParams (see page 8090) object lists the key fields and their values for a single dataset adapter.

ARecurse indicates whether SilentLocate should position the dataset of just this adapter, or to position the datasets of all adapters linked to this one in a master/detail relationship as well.

SilentLocate returns true if it successfully positions all requested datasets on the specified records. SilentLocate returns false if it was unable to position one or more datasets on the specified record.

See Also

Locate (see page 7796)

TLocateParams (see page 8090)

41.1.4.4.53 TCustomDataSetAdapter.StringToMode Method

Converts a mode string to the corresponding TDataSetAdapterMode (see page 8111) value.

Pascal

```
function StringToMode(const AValue: string): TDataSetAdapterMode;
```

C++

```
__fastcall TDataSetAdapterMode StringToMode(const AnsiString AValue);
```

Description

StringToMode interprets the specified string to a value that can be applied to the Mode (see page 7810) property.

AValue is a string to convert.

If AValue is an empty string, StringToMode returns the default mode (amBrowse). Otherwise, StringToMode converts from one of the values in the list GetAdapterModeNames (see page 7783) supplies to the corresponding TDataSetAdapterMode (see page 8111) value. If AValue is not an empty string or one of the strings in the list that GetAdapterModeNames (see page 7783) supplies, StringToMode raises an EAdapterException.

See Also

Mode (see page 7810)

GetAdapterModeNames (see page 7783)

EAdapterException (see page 23696)

41.1.4.4.54 TCustomDataSetAdapter.UpdateFieldsExecuteValues Method

Executes the adapter's fields with values passed in the HTTP request.

Pascal

```
procedure UpdateFieldsExecuteValues(AActionRequest: IActionRequest; AAdapterFields:
TObjectList); override;
```

C++

```
virtual __fastcall UpdateFieldsExecuteValues(IActionRequest AActionRequest, TObjectList
AAdapterFields);
```

Description

UpdateFieldsExecuteValues first calls LocateRequestRecord (see page 7798) to position the associated dataset on the record identified by the parameters of the current request. It then loops through the specified adapter fields, calling on them to execute their value if they are executable. When a field is executable, the field value passed in the HTTP request is used to execute a command rather than update a value.

If any of the adapter fields encounters a problem while trying to execute the newly specified value, UpdateFieldsExecuteValues adds a message to the Errors property indicating what problem occurred before trying to execute the next field in the list.

AActionRequest is an interface to the current request object.

AAdapterFields is a list of fields to execute using values obtained from AActionRequest.

See Also

LocateRequestRecord (see page 7798)

Errors (see page 24182)

41.1.4.4.55 TCustomDataSetAdapter.UpdateFieldsGetAnyChanges Method

Indicates whether the values in the HTTP request change the value of any adapter fields.

Pascal

```
function UpdateFieldsGetAnyChanges(AActionRequest: IActionRequest; AAdapterFields:
TObjectList): Boolean; override;
```

C++

```
virtual __fastcall Boolean UpdateFieldsGetAnyChanges(IActionRequest AActionRequest,
TObjectList AAdapterFields);
```

Description

UpdateFieldsGetAnyChanges checks the specified adapter fields to see if the current request message causes any of them to change their values.

AActionRequest is an interface to the current request object.

AAdapterFields is a list of fields to check to see whether the request changes their values.

UpdateFieldsGetAnyChanges returns true if one or more field values changes as a result of the current request.

UpdateFieldsGetAnyChanges always returns true if the mode of the adapter is amQuery or amInsert, because queries and insertions always start with no field values and so any field values in the request represent a change.

If Mode (see page 7810) is not amQuery or amInsert, UpdateFieldsGetAnyChanges first calls LocateRequestRecord (see page 7798) to position the associated dataset on the record identified by the parameters of the current request. It then loops through the specified adapter fields, calling their CheckValueChange method to determine if they have changed. If the CheckValueChange method of any field returns true, UpdateFieldsGetAnyChanges returns true.

If any of the adapter fields encounters a problem in its CheckValueChange method, UpdateFieldsExecuteValues (see page 7803) adds a message to the Errors property indicating what problem occurred before trying to check the next field in the list.

See Also

Errors (see page 24182)

LocateRequestRecord (see page 7798)

CheckValueChange (see page 7737)

Mode (see page 7810)

41.1.4.4.56 TCustomDataSetAdapter.UpdateFieldsUpdateValues Method

Updates the adapter's fields with values passed in the HTTP request.

Pascal

```
procedure UpdateFieldsUpdateValues(AActionRequest: IActionRequest; AAdapterFields:
TObjectList); override;
```

C++

```
virtual __fastcall UpdateFieldsUpdateValues(IActionRequest AActionRequest, TObjectList
AAdapterFields);
```

Description

UpdateFieldsGetAnyChanges (see page 7804) applies the updates in the current request message to the specified adapter fields.

AActionRequest is an interface to the current request object.

AAdapterFields is a list of fields to update.

UpdateFieldsUpdateValues performs the following tasks:

1.It generates an OnBeforeUpdateFields event.

2.It calls LocateRequestRecord (see page 7798) to position the associated dataset on the record identified by the parameters of the current request.

3.It loops through the specified adapter fields, calling their UpdateValue method to update the value of the associated dataset field. If any of the adapter fields encounters a problem in its UpdateValue method, UpdateFieldsExecuteValues (see page 7803) adds a message to the Errors property indicating what problem occurred before trying to check the next field in the list.

4.If there were any errors updating field values, it cancels all edits to the current record, backing out the changes. Otherwise, it posts the current record to the dataset.

5. It generates an OnAfterUpdateFields event.

See Also

OnBeforeUpdateFields (see page 24187)

OnAfterUpdateFields (see page 24185)

Errors (see page 24182)

UpdateValue (see page 7751)

LocateRequestRecord (see page 7798)

41.1.4.4.57 TCustomDataSetAdapter.UpdateFieldsValidateValues Method

Validates the values passed in the current HTTP request for the adapter's fields.

Pascal

```
procedure UpdateFieldsValidateValues(AActionRequest: IActionRequest; AAdapterFields:
TObjectList); override;
```

C++

```
virtual __fastcall UpdateFieldsValidateValues(IActionRequest AActionRequest, TObjectList
AAdapterFields);
```

Description

UpdateFieldsValidateChanges screens the field values in the current request message to determine whether they appear valid.

AActionRequest is an interface to the current request object.

AAdapterFields is a list of fields to check.

UpdateFieldsUpdateValues (see page 7805) performs the following tasks:

1.If the adapter is in edit mode or browse mode, it calls LocateRequestRecord (see page 7798) to position the associated dataset on the record identified by the parameters of the current request and then loops through the specified adapter fields, calling their CheckOrigValue method to ensure that the field values on the specified record have not been changed by another user.

2.If any of the original field values have changed, UpdateFieldsUpdateValues (see page 7805) adds a message to the Errors property and returns.

3.It generates an OnBeforeValidateFields event.

4.It loops through the specified adapter fields, calling their ValidateValue method to validate the new value from the request message. If any of the adapter fields raises an exception its UpdateValue method (indicating an invalid value), UpdateFieldsValidateValues adds a message to the Errors property before trying to validate the next field value.

5. It generates an OnAfterValidateFields event.

See Also

OnBeforeValidateFields (see page 24187)

OnAfterValidateFields (see page 24186)

Errors (see page 24182)

ValidateValue (see page 7765)

LocateRequestRecord (see page 7798)

CheckOrigValue (see page 7736)

Mode (see page 7810)

41.1.4.5 TCustomDataSetAdapter Properties

The properties of the TCustomDataSetAdapter class are listed here.

Protected Properties

	Name	Description
	AdapterRowNotFound (see page 7807)	Indicates the dataset adapter for which this adapter could not locate the current record.
	DataSetPrepared (see page 7808)	Indicates whether the associated dataset has been properly set up for use by this adapter.

Public Properties

	Name	Description
	DataSet (see page 7807)	Specifies the dataset that this adapter represents.
R	DetailAdapters (see page 7808)	Lists the dataset adapters for all datasets that are details of this one.
	InDefaultMode (see page 7808)	Indicates whether the dataset adapter is in its default mode.
R	LocateParamsList (see page 7809)	Lists the parameters that identify the current record of this dataset and all other datasets linked to it in a master/detail relationship.
	MasterAdapter (see page 7809)	Indicates the dataset adapter, if any, to which this adapter is linked in a master/detail relationship.
	Mode (see page 7810)	Indicates how forms built from this dataset adapter operate on the information it provides.
	RefreshAfterPost (see page 7810)	Indicates whether the data set should be refreshed or not after posting..

41.1.4.5.1 TCustomDataSetAdapter.AdapterRowNotFound Property

Indicates the dataset adapter for which this adapter could not locate the current record.

Pascal

```
property AdapterRowNotFound: TCustomDataSetAdapter;
```

C++

```
__property TCustomDataSetAdapter AdapterRowNotFound;
```

Description

In the course of its normal operation, the dataset adapter often has to position this dataset on a specified record (such as the record to update in response to an update request). When the adapter positions itself in this way, it also positions all other dataset adapters that are linked to it in a master/detail relationship. If the adapter fails in its attempt to position all dataset adapters as specified, it sets AdapterRowNotFound to the first dataset adapter (starting at the master of all detail datasets) that can't be correctly positioned.

The RaiseRowNotFound (see page 7800) method uses this property to generate an error message when an attempt to position the dataset adapters fails.

See Also

RaiseRowNotFound (see page 7800)

MasterAdapter (see page 7809)

DetailAdapters (see page 7808)

41.1.4.5.2 TCustomDataSetAdapter.DataSet Property

Specifies the dataset that this adapter represents.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Set DataSet to associate this adapter with an existing dataset component. Read this property to access the dataset component that this adapter represents.

Note: Do not set DataSet to a unidirectional dataset. Unidirectional datasets do not work with the way dataset adapters position themselves on records.

41.1.4.5.3 TCustomDataSetAdapter.DataSetPrepared Property

Indicates whether the associated dataset has been properly set up for use by this adapter.

Pascal

```
property DataSetPrepared: Boolean;
```

C++

```
__property Boolean DataSetPrepared;
```

Description

TCustomDataSetAdapter (see page 7766) uses DataSetPrepared internally to keep track of whether it needs to prepare the dataset for use by this adapter. This property is initially false until the WebSnap application calls the PrepareDataSet (see page 7799) method. PrepareDataSet (see page 7799) changes this property to true, and it remains true until the Web module is deactivated or the Mode (see page 7810) of this adapter changes.

Note: Do not confuse the DataSetPrepared property with the Prepared property of many dataset components. DataSetPrepared indicates whether the adapter has performed any tasks it requires to accurately represent its corresponding dataset. (These tasks can include, for example, specifying and running a query on the underlying dataset, or checking which fields should be visible in the current mode.) The Prepared property of a dataset component indicates whether the underlying database engine has initialized itself to represent a database query.

See Also

PrepareDataSet (see page 7799)

OnPrepareDataSet (see page 7810)

DataSet (see page 7807)

Mode (see page 7810)

41.1.4.5.4 TCustomDataSetAdapter.DetailAdapters Property

Lists the dataset adapters for all datasets that are details of this one.

Pascal

```
property DetailAdapters: TComponentList;
```

C++

```
__property TComponentList DetailAdapters;
```

Description

DetailAdapters is a list of dataset adapters that are linked to this one in a master/detail relationship where this adapter represents the master dataset.

DetailAdapters is a read-only property. To set up a master/detail relationship between dataset adapters, set the MasterAdapter (see page 7809) property of the dataset adapter that acts as a detail dataset.

See Also

MasterAdapter (see page 7809)

41.1.4.5.5 TCustomDataSetAdapter.InDefaultMode Property

Indicates whether the dataset adapter is in its default mode.

Pascal

```
property InDefaultMode: Boolean;
```

C++

```
__property Boolean InDefaultMode;
```

Description

A newly-created dataset adapter has InDefaultMode true, indicating that the Mode (see page 7810) property has its default value (amBrowse). A dataset adapter in browse mode provides read-only access to the data in the corresponding dataset.

Changing the Mode (see page 7810) property sets InDefaultMode to false. Setting InDefaultMode to true restores the Mode (see page 7810) property to amBrowse.

See Also

Mode (see page 7810)

41.1.4.5.6 TCustomDataSetAdapter.LocateParamsList Property

Lists the parameters that identify the current record of this dataset and all other datasets linked to it in a master/detail relationship.

Pascal

```
property LocateParamsList: TLocateParamsList;
```

C++

```
__property TLocateParamsList LocateParamsList;
```

Description

TCustomDataSetAdapter (see page 7766) uses LocateParamsList to store the information it needs to position itself correctly on the current record. This information is taken from parameter values in the current request message. Those parameter values, in turn, originate from values added to an HTML form that were gathered using the GetKeyParams (see page 7787) method.

When you call the Locate (see page 7796) or LocateAndApply (see page 7796) method, TCustomDataSet uses the values in LocateParamsList to identify the target position in its associated dataset (and any datasets linked in a master/detail relationship).

See Also

ExtractRequestParams (see page 7781)

Locate (see page 7796)

LocateAndApply (see page 7796)

41.1.4.5.7 TCustomDataSetAdapter.MasterAdapter Property

Indicates the dataset adapter, if any, to which this adapter is linked in a master/detail relationship.

Pascal

```
property MasterAdapter: TCustomDataSetAdapter;
```

C++

```
__property TCustomDataSetAdapter MasterAdapter;
```

Description

Set MasterAdapter to link this adapter as a detail to another dataset adapter in a master/detail relationship. If MasterAdapter is

nil (Delphi) or NULL (C++), this adapter does not represent the detail set of another adapters dataset. If MasterAdapter is not nil (Delphi) or NULL (C++), this dataset adapter is listed by the master adapter's DetailAdapters (see page 7808) property.

See Also

DetailAdapters (see page 7808)

41.1.4.5.8 TCustomDataSetAdapter.Mode Property

Indicates how forms built from this dataset adapter operate on the information it provides.

Pascal

```
property Mode: TDataSetAdapterMode;
```

C++

```
__property TDataSetAdapterMode Mode;
```

Description

Mode controls what tasks forms built using this dataset adapter perform. By default, Mode is amBrowse, which indicates that the adapter supplies data for read-only forms. The default Web items you add to an adapter page producer, or code in server-side script, check Mode to determine how to build an HTML form that is sent in response to the current HTTP request mesaage. Mode controls:

The types of controls that are added to the HTML form.

The values of the fields in the HTML form.

The information that the HTML form stores as hidden fields.

41.1.4.5.9 TCustomDataSetAdapter.RefreshAfterPost Property

Indicates whether the data set should be refreshed or not after posting..

Pascal

```
property RefreshAfterPost: Boolean;
```

C++

```
__property Boolean RefreshAfterPost;
```

Description

RefreshAfterPost is used to determine whether or not the data set should be refreshed after it has been posted.

41.1.4.6 TCustomDataSetAdapter Events

The events of the TCustomDataSetAdapter class are listed here.

Public Events

	Name	Description
	OnPrepareDataSet (see page 7810)	Occurs before the dataset adapter uses any information from its associated dataset.

41.1.4.6.1 TCustomDataSetAdapter.OnPrepareDataSet Event

Occurs before the dataset adapter uses any information from its associated dataset.

Pascal

```
property OnPrepareDataSet: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPrepareDataSet;
```

Description

Write an OnPrepareDataSet event handler to set up the associated dataset before the WebSnap application uses this adapter. For example, you can use an OnPrepareDataSet event handler to dynamically generate and execute a query.

This event occurs when the Web module containing the dataset adapter is first activated, and whenever the WebSnap application changes the value of the Mode (see page 7810) property.

Note: Do not confuse the this event with the Prepared property of many dataset components. This event allows the adapter to perform any tasks it requires to accurately represent its corresponding dataset. The Prepared property of a dataset component indicates whether the underlying database engine has initialized itself to represent a database query.

See Also

PrepareDataSet (see page 7799)

DataSet (see page 7807)

Mode (see page 7810)

41.1.5 TCustomDataSetAdapterAction Class

TCustomDataSetAdapterAction is the base class for adapter actions that can be added to a dataset adapter.

Class Hierarchy

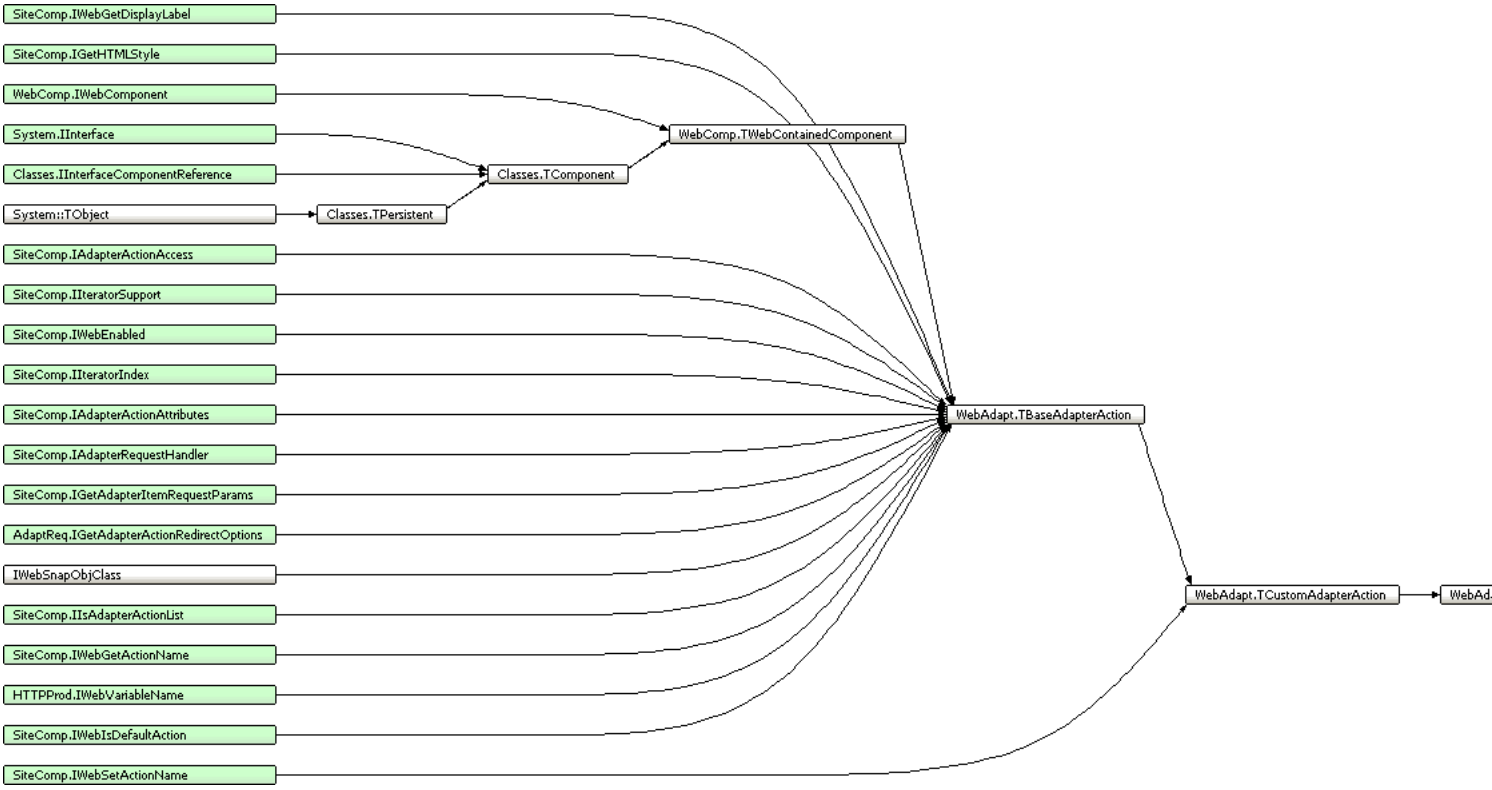

Pascal

```
TCustomDataSetAdapterAction = class(TImplementedAdapterAction);
```

C++

```
class TCustomDataSetAdapterAction : public TImplementedAdapterAction;
```

File

DBAdapt

Description

TCustomDataSetAdapterAction is the base class for all adapter action objects that perform functions for a dataset adapter. TCustomDataSetAdapterAction descendants are included as the actions of a dataset adapter (TDataSetAdapter (see page 7872)). Each TCustomDataSetAdapterAction descendant can be executed to perform a specific function on the dataset of the parent dataset adapter.

Do not use TCustomDataSetAdapterAction directly. It is intended only as a base class. Instead, use a TCustomDataSetAdapterAction descendant that represents the particular action you want.

See Also

TDataSetAdapterApplyRowAction (see page 7890)

TDataSetAdapterBrowseRowAction (see page 7903)

TDataSetAdapterCancelRowAction (see page 7916)

TDataSetAdapterDeleteRowAction (see page 7934)

TDataSetAdapterEditRowAction (see page 7947)

TDataSetAdapterFirstRowAction (see page 7972)

TDataSetAdapterLastRowAction (see page 7984)

TDataSetAdapterNextRowAction (see page 8030)

TDataSetAdapterPrevRowAction (see page 8042)

TDataSetAdapterRefreshRowAction (see page 8054)

TDataSetAdapterNewRowAction (see page 8016)

TDataSetAdapter (see page 7872)

TDataSetAdapterRowAction (see page 8066)

41.1.5.1 TCustomDataSetAdapterAction Members

The following tables list the members exposed by TCustomDataSetAdapterAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 7819)	Returns the value of the Adapter (see page 7822) property.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7819)	Returns the value of the DataSet (see page 7822) property.
V	GetDefaultActionName (see page 24055)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.

	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
V	ImplExecuteActionRequest (see page 24063)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplExecuteActionRequest (see page 24195)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
V	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
V	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetVisible (see page 7819)	Provides the underlying implementation for the GetVisible method.
V	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
V	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
V	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
V	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
V	ImplWebEnabled (see page 7820)	Provides the underlying implementation for the WebEnabled method.

	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7822)	Indicates the dataset adapter that uses this object to represent one of its actions.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 24075)	Called after the adapter action has executed.
	OnAfterExecute (see page 24198)	Called after the adapter action has executed.

	OnAfterGetResponse (see page 24075)	Called after the adapter action response is generated.
	OnAfterGetResponse (see page 24199)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24076)	Called before the adapter action is executed.
	OnBeforeExecute (see page 24199)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24076)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 24200)	Called before the adapter action response is generated.
	OnExecute (see page 7822)	Occurs when server-side script executes this action.
	OnGetEnabled (see page 7823)	Occurs when the WebSnap application to determine whether a control or hyperlink for this action should be enabled.
	OnGetParams (see page 24077)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 7818)	Creates and initializes an instance of TCustomDataSetAdapterAction (see page 7811).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.

	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionModes (see page 7821)	Indicates what dataset adapter modes this field supports.
	ActionName (see page 7821)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 7822)	Indicates the dataset on which this action operates.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 24073)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 24073)	Defines the rights required to execute this action.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RedirectOptions (see page 24074)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.5.2 TCustomDataSetAdapterAction.Create Constructor

Creates and initializes an instance of TCustomDataSetAdapterAction (see page 7811).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomDataSetAdapterAction(TComponent * AOwner);
```

Description

The TCustomDataSetAdapterAction (see page 7811) constructor allocates memory for the adapter action object and initializes its properties.

AOwner is the component that is responsible for starting the streaming process when a module containing the adapter action is loaded or saved. It becomes the value of the Owner property. Typically, AOwner is the Web module.

See Also

Owner (see page 2807)

41.1.5.3 TCustomDataSetAdapterAction Methods

The methods of the TCustomDataSetAdapterAction class are listed here.

Protected Methods

	Name	Description
	GetAdapter (see page 7819)	Returns the value of the Adapter (see page 7822) property.
	GetDataSet (see page 7819)	Returns the value of the DataSet (see page 7822) property.
V	ImplGetVisible (see page 7819)	Provides the underlying implementation for the GetVisible method.
V	ImplWebEnabled (see page 7820)	Provides the underlying implementation for the WebEnabled method.

41.1.5.3.1 TCustomDataSetAdapterAction.GetAdapter Method

Returns the value of the Adapter (see page 7822) property.

Pascal

```
function GetAdapter: TCustomDataSetAdapter;
```

C++

```
__fastcall TCustomDataSetAdapter GetAdapter();
```

Description

GetAdapter is the protected read implementation of the Adapter (see page 7822) property.

See Also

Adapter (see page 7822)

41.1.5.3.2 TCustomDataSetAdapterAction.GetDataSet Method

Returns the value of the DataSet (see page 7822) property.

Pascal

```
function GetDataSet: TDataSet;
```

C++

```
__fastcall TDataSet GetDataSet();
```

Description

GetDataSet is the protected read implementation of the DataSet (see page 7822) property.

See Also

DataSet (see page 7822)

41.1.5.3.3 TCustomDataSetAdapterAction.ImplGetVisible Method

Provides the underlying implementation for the GetVisible method.

Pascal

```
function ImplGetVisible: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplGetVisible();
```

Description

The GetVisible method calls ImplGetVisible to determine whether the HTML form should include a control or hyperlink for this action. (In server-side script, this returns the value of the Visible property).

ImplGetVisible returns true if the ActionModes (see page 7821) property includes the current mode of the parent dataset adapter. If not, it returns false.

See Also

GetVisible (see page 24061)

ActionModes (see page 7821)

Mode (see page 7810)

41.1.5.3.4 TCustomDataSetAdapterAction.ImplWebEnabled Method

Provides the underlying implementation for the WebEnabled method.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

The WebEnabled method calls ImplWebEnabled to determine whether the control or hyperlink for this action should appear enabled in the HTML form. WebEnabled returns true if controls should be enabled, false otherwise.

ImplWebEnabled generates an OnGetEnabled (see page 7823) event so that you can specify whether controls for this action are enabled. If there is no OnGetEnabled (see page 7823) event handler, ImplWebEnabled returns true unless the DataSet (see page 7822) property is not set.

See Also

WebEnabled (see page 24071)

OnGetEnabled (see page 7823)

DataSet (see page 7822)

41.1.5.4 TCustomDataSetAdapterAction Properties

The properties of the TCustomDataSetAdapterAction class are listed here.

Protected Properties

	Name	Description
	Adapter (see page 7822)	Indicates the dataset adapter that uses this object to represent one of its actions.

Public Properties

	Name	Description
	ActionModes (see page 7821)	Indicates what dataset adapter modes this field supports.
	ActionName (see page 7821)	Identifies the name of this action when using server-side script.
R	DataSet (see page 7822)	Indicates the dataset on which this action operates.

41.1.5.4.1 TCustomDataSetAdapterAction.ActionModes Property

Indicates what dataset adapter modes this field supports.

Pascal

```
property ActionModes: TDataSetAdapterModes;
```

C++

```
__property TDataSetAdapterModes ActionModes;
```

Description

ActionModes is a set of all the modes in which this adapter action can be used. If the parent dataset adapter is in a mode that is not included in this set, any forms generated using that adapter do not include a control for executing this action. Thus, for example, there is no sense in adding a control that applies edits to a form that presents a read-only view of the data. An apply action, then, should not include amBrowse in its action modes.

See Also

CanModify (see page 7154)

Adapter (see page 7752)

Mode (see page 7810)

GetDisplayStyle (see page 7739)

GetInputStyle (see page 7741)

GetViewMode (see page 7744)

41.1.5.4.2 TCustomDataSetAdapterAction.ActionName Property

Identifies the name of this action when using server-side script.

Pascal

```
property ActionName: string;
```

C++

```
__property AnsiString ActionName;__property AnsiString ActionName;__property AnsiString
ActionName;
```

Description

The ActionName property defines the name used to access the adapter action with server-side script.

See Also

ImplGetActionName (see page 24195)

ImplSetActionName (see page 24197)

41.1.5.4.3 TCustomDataSetAdapterAction.Adapter Property

Indicates the dataset adapter that uses this object to represent one of its actions.

Pascal

```
property Adapter: TCustomDataSetAdapter;
```

C++

```
__property TCustomDataSetAdapter Adapter;
```

Description

Use Adapter to access the parent adapter of this adapter action. This adapter is associated with the dataset on which this action operates. This adapter action is one of the objects listed in its Actions property.

See Also

Actions (see page 24402)

DataSet (see page 7822)

41.1.5.4.4 TCustomDataSetAdapterAction.DataSet Property

Indicates the dataset on which this action operates.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Read DataSet to access the dataset on which this action operates. DataSet is the value of the parent dataset adapter's DataSet property.

If this action is not the child of a dataset adapter, or if the parent dataset adapter has not been bound to a dataset component, DataSet is nil (Delphi) or NULL (C++).

See Also

DataSet (see page 7807)

41.1.5.5 TCustomDataSetAdapterAction Events

The events of the TCustomDataSetAdapterAction class are listed here.

Public Events

	Name	Description
	OnExecute (see page 7822)	Occurs when server-side script executes this action.
	OnGetEnabled (see page 7823)	Occurs when the WebSnap application to determine whether a control or hyperlink for this action should be enabled.

41.1.5.5.1 TCustomDataSetAdapterAction.OnExecute Event

Occurs when server-side script executes this action.

Pascal

```
property OnExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnExecute;
```

Description

Write an OnExecute event handler to provide your own implementation for the dataset adapter action. If you supply an event handler for this event, then the event handler replaces the default behavior of the action object. That is, when server-side script calls on this action to execute, the WebSnap application executes the OnExecute event handler instead of the built-in response of the action component.

Before the OnExecute event executes, the action fetches all parameter values from the current request and puts the parent adapter into the mode that the current request indicates.

See Also

OnGetParams (see page 24077)

OnBeforeExecute (see page 24076)

OnAfterExecute (see page 24075)

Mode (see page 7810)

41.1.5.5.2 TCustomDataSetAdapterAction.OnGetEnabled Event

Occurs when the WebSnap application to determine whether a control or hyperlink for this action should be enabled.

Pascal

```
property OnGetEnabled: TActionGetEnabledEvent;
```

C++

```
__property TActionGetEnabledEvent OnGetEnabled;
```

Description

Write an OnGetEnabled event handler to indicate whether a control or hyperlink that represents this event in an HTML form should be enabled. If the event handler sets the Enabled parameter to false, controls or hyperlinks for this action always appear disabled. If the event handler sets the Enabled parameter to true, the TCustomDataSetAdapterAction (see page 7811) descendant determines whether controls and hyperlinks are enabled, based on its ability to perform its built-in task.

See Also

DataSet (see page 7822)

WebEnabled (see page 24071)

41.1.6 TCustomDataSetAdapterField Class

TCustomDataSetAdapterField is the common base class for adapter field objects that represent a dataset field of a type whose value can be displayed as a simple text element.

Class Hierarchy

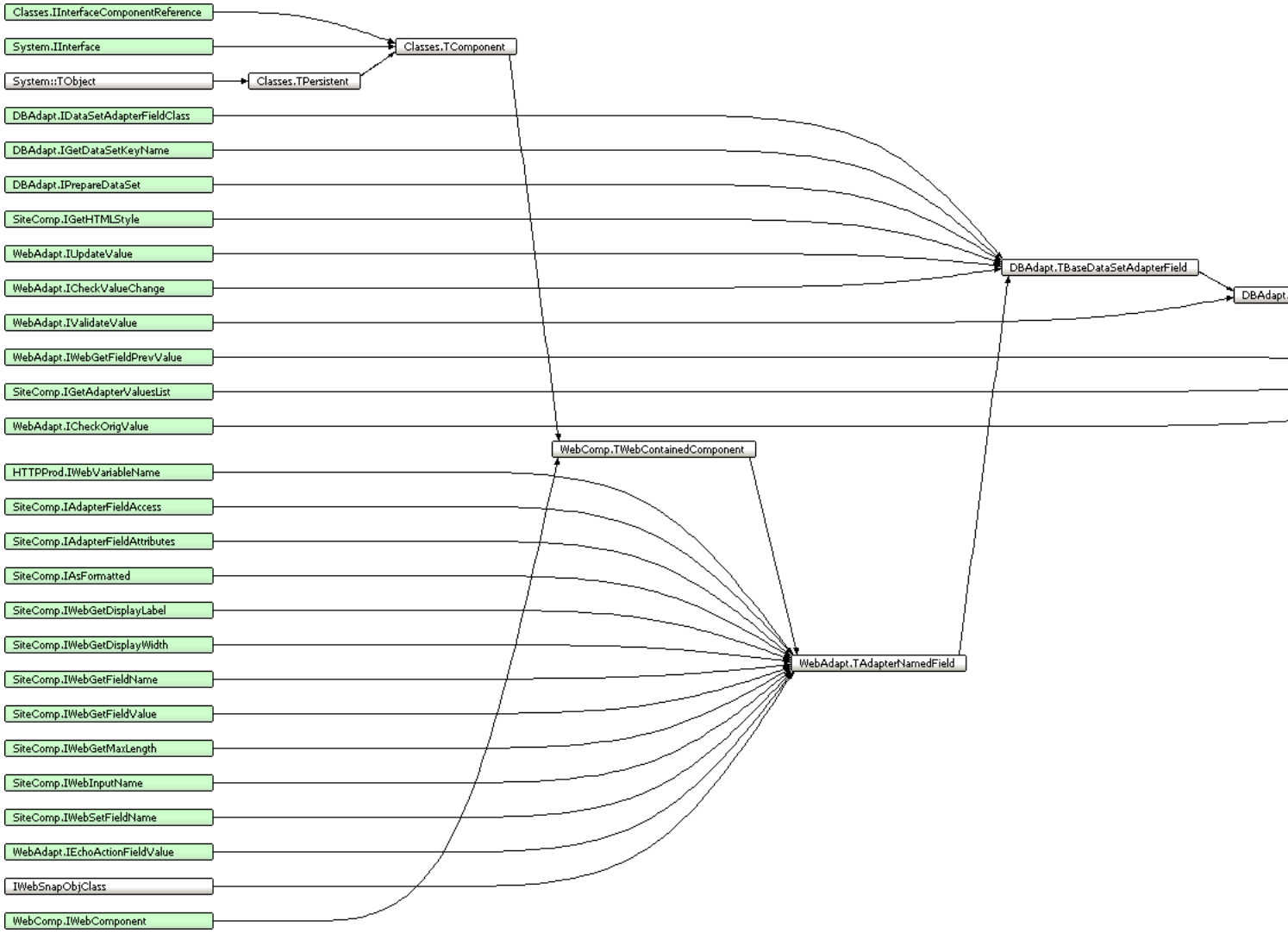

Pascal

```
TCustomDataSetAdapterField = class(TBaseDataSetAdapterInputField, IWebGetFieldPrevValue,
IGetAdapterValuesList, ICheckOrigValue);
```

C++

```
class TCustomDataSetAdapterField : public TBaseDataSetAdapterInputField, public
IWebGetFieldPrevValue, public IGetAdapterValuesList, public ICheckOrigValue;
```

File

DBAdapt

Description

TCustomDataSetAdapterField is the base class for adapter field objects that represent database fields of simple types that can appear in single-line edit controls. This is in contrast, for example, to TDataSetAdapterMemoField (see page 7996), which typically is displayed in a multiline edit control. Because TCustomDataSetAdapterField descendants have values that can be displayed simply, this class adds support for maintaining a list of possible values from which users can choose. If you provide

such a list, HTML forms that include this field typically use a selection control such as a list box.

TCustomDataSetAdapterField descendants are included as the fields of a dataset adapter (TDataSetAdapter (see page 7872)). Each TCustomDataSetAdapterField descendant represents a single field from a database table, and enables server-side script to refer to or update that field's value.

Do not use TCustomDataSetAdapterField directly. It is intended only as a base class. Instead, use a TCustomDataSetAdapterField descendant such as TDataSetAdapterField (see page 7960).

See Also

TDataSetAdapterField (see page 7960)

TDataSetAdapter (see page 7872)

TField (see page 7115)

TDataSetAdapterMemoField (see page 7996)

41.1.6.1 TCustomDataSetAdapterField Members

The following tables list the members exposed by TCustomDataSetAdapterField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckOrigValue (see page 7736)	Ensures that the current value of this field is the same as the original value.
	CheckValueChange (see page 7737)	Indicates whether the current value of this field is the same as the original value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 7738)	Indicates whether references to this field in HTML forms must be qualified with the name of the parent adapter.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 7738)	Returns the value of the Adapter (see page 7752) property.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 7831)	Returns the value of the ValuesList (see page 7837) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7738)	Returns the dataset associated with this field's parent adapter.
V	GetDataSetFieldValue (see page 7739)	Returns the value of the associated dataset field.

	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 7739)	Returns a string value that suggests an HTML display style for this adapter field.
	GetDisplayStyleType (see page 7740)	Returns the HTML display style type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 7741)	Returns a string value that suggests an HTML editing style for this adapter field.
	GetInputStyleType (see page 7831)	Returns the default style of the HTML control to use for editing this adapter field.
	GetKeyName (see page 7742)	Returns the name of the associated dataset field if it appears in the index (key) that the parent dataset adapter uses.
	GetKeyParams (see page 7743)	Returns a list of fields that the parent dataset adapter uses to locate records, along with their current values in that adapter's dataset.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetNeedPrevValue (see page 7832)	Indicates whether an HTML form should include a hidden field that stores this field's value so that generated requests include the previous field value.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPrevValue (see page 7832)	Returns the value of this field, as it should appear in a hidden field in an HTML form.
	GetReadOnly (see page 7743)	Indicates whether the adapter field represents a read-only database field.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 7744)	Returns a string value that indicates what function a control that represents this field should perform.
	GetViewModeType (see page 7744)	Returns the view mode for the function a control that represents this field should perform.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 7745)	Provides the underlying implementation for the AsFormatted method.
	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplCheckOrigValue (see page 7833)	Provides the underlying implementation for the CheckOrigValue (see page 7736) method.
	ImplCheckValueChange (see page 7833)	Provides the underlying implementation for the CheckValueChange (see page 7737) method.
	ImplGetDisplayLabel (see page 7747)	Provides the underlying implementation for the GetDisplayLabel method.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayWidth (see page 7747)	Provides the underlying implementation for the GetDisplayWidth method.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetEchoActionFieldValue (see page 7763)	Provides the underlying implementation for the GetEchoActionFieldValue method.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetNeedPrevValue (see page 7834)	Provides the underlying implementation for the GetNeedPrevValue (see page 7832) method.
	ImplGetPrevValue (see page 7835)	Provides the underlying implementation of the GetPrevValue (see page 7832) method.

	ImplGetRequired (see page 7747)	Provides the underlying implementation for the GetRequired method.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 7835)	Provides the underlying implementation for the GetValue method.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 7748)	Provides the underlying implementation for the GetVisible method.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 7763)	Provides the underlying implementation for the SetEchoActionFieldValue method.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 7749)	Provides the underlying implementation for the SetFieldName method.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	ImplUpdateValue (see page 7836)	Provides the underlying implementation for the UpdateValue (see page 7751) method.
	ImplValidateValue (see page 7764)	Provides the underlying implementation for the ValidateValue (see page 7765) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 7836)	Responds when components are added to or deleted from the WebSnap application.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	PrepareDataSet (see page 7750)	Responds when the parent dataset adapter prepares its dataset.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSetField (see page 7751)	Sets the value of the DataSetField (see page 7752) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	TestValueChange (see page 7764)	Checks whether a new value matches the current value of this field.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 7751)	Updates the adapter field when the user assigns a new value.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.

	ValidateValue (see page 7765)	Validates a new value for this field that appears in the current request.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7752)	Indicates the dataset adapter that uses this object to represent one of its fields.
R	Adapter (see page 23943)	Returns the parent adapter component.
	DataSetField (see page 7752)	Specifies the name of the dataset field that this adapter field represents.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 7754)	Occurs when the adapter field needs to supply its formatted value as a string that can appear in HTML forms.
	OnGetValue (see page 7754)	Occurs when the adapter field needs to supply its value as a Variant.
	OnValidateValue (see page 7766)	Occurs when the parent adapter wants to validate the new value for this field.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckOrigValue (see page 24569)	This is CheckOrigValue, a member of class ICheckOrigValue.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 7735)	Creates and initializes an instance of TBaseDataSetAdapterField (see page 7729).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 7737)	Returns the dataset field whose value this adapter field represents.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetKeyName (see page 8105)	This is GetKeyName, a member of class IGetDataSetKeyName.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PrepareDataSet (see page 8109)	This is PrepareDataSet, a member of class IPrepareDataSet.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)

V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	FieldFlags (see page 7752)	Indicates various attributes of the adapter field.
	FieldModes (see page 7753)	Indicates what dataset adapter modes this field supports.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
R	KeyName (see page 8105)	This is KeyName, a member of class IGetDataSetKeyName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	NeedPrevValue (see page 24587)	This is NeedPrevValue, a member of class IWebGetFieldPrevValue.
	OnUpdateValue (see page 7765)	Occurs when the WebSnap application updates this adapter field's value based on in incoming request.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	PrevValue (see page 24587)	This is PrevValue, a member of class IWebGetFieldPrevValue.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
	ValuesList (see page 7837)	Specifies the interface of an object that lists the possible values for this field.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.6.2 TCustomDataSetAdapterField Methods

The methods of the TCustomDataSetAdapterField class are listed here.

Protected Methods

	Name	Description
	GetAdapterValuesList (see page 7831)	Returns the value of the ValuesList (see page 7837) property.
V	GetInputStyleType (see page 7831)	Returns the default style of the HTML control to use for editing this adapter field.
	GetNeedPrevValue (see page 7832)	Indicates whether an HTML form should include a hidden field that stores this field's value so that generated requests include the previous field value.
	GetPrevValue (see page 7832)	Returns the value of this field, as it should appear in a hidden field in an HTML form.
V	ImplCheckOrigValue (see page 7833)	Provides the underlying implementation for the CheckOrigValue (see page 7736) method.
V	ImplCheckValueChange (see page 7833)	Provides the underlying implementation for the CheckValueChange (see page 7737) method.
V	ImplGetNeedPrevValue (see page 7834)	Provides the underlying implementation for the GetNeedPrevValue (see page 7832) method.
V	ImplGetPrevValue (see page 7835)	Provides the underlying implementation of the GetPrevValue (see page 7832) method.
V	ImplGetValue (see page 7835)	Provides the underlying implementation for the GetValue method.
V	ImplUpdateValue (see page 7836)	Provides the underlying implementation for the UpdateValue (see page 7751) method.
V	Notification (see page 7836)	Responds when components are added to or deleted from the WebSnap application.

41.1.6.2.1 TCustomDataSetAdapterField.GetAdapterValuesList Method

Returns the value of the ValuesList (see page 7837) property.

Pascal

```
function GetAdapterValuesList: IValuesListAdapter;
```

C++

```
__fastcall IValuesListAdapter GetAdapterValuesList();
```

Description

GetAdapterValuesList returns the interface of the adapter that lists the possible values for this field. If there is no such list, GetAdapterValuesList returns nil (Delphi) or NULL (C++).

See Also

ValuesList (see page 7837)

41.1.6.2.2 TCustomDataSetAdapterField.GetInputStyleType Method

Returns the default style of the HTML control to use for editing this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const AnsiString
AAdapterMode);
```

Description

GetInputStyleType returns the recommended type of control to use for editing this field's value in HTML forms.

AAdapterMode indicates the current mode of the parent dataset adapter. It is one of the following values:

Value	Mode	Meaning
Insert	amInsert	The control is to be used for entering a field value in a newly inserted record.
Edit	amEdit	The control is to be used for editing the current field value.
Browse	amBrowse	The control is to be used for read-only display.
Query	amQuery	The control is to be used for entering a criterion to use in a generated query.

GetInputStyleType returns htmliSelect (a single-selection list box) if the ValuesList (see page 7837) property provides a list of possible values for the field. Otherwise, GetInputStyleType returns htmliTextInput (a single-line edit control).

See Also

GetInputStyle (see page 7741)

TDataSetAdapterModes (see page 8115)

ValuesList (see page 7837)

41.1.6.2.3 TCustomDataSetAdapterField.GetNeedPrevValue Method

Indicates whether an HTML form should include a hidden field that stores this field's value so that generated requests include the previous field value.

Pascal

```
function GetNeedPrevValue: Boolean;
```

C++

```
__fastcall Boolean GetNeedPrevValue();
```

Description

GetNeedPrevValue returns true if the current value of this field should be included as a hidden field in HTML forms. This hidden field value then appears in requests that originate from that form so that if the form changes the field value, the previous value is still available. When GetNeedPrevValue returns true, the GetPrevValue (see page 7832) method supplies the current field value.

GetNeedPrevValue delegates the task of determining the need for a hidden field to the ImplGetNeedPrevValue (see page 7834) method.

See Also

ImplGetNeedPrevValue (see page 7834)

GetPrevValue (see page 7832)

41.1.6.2.4 TCustomDataSetAdapterField.GetPrevValue Method

Returns the value of this field, as it should appear in a hidden field in an HTML form.

Pascal

```
function GetPrevValue: Variant;
```

C++

```
__fastcall Variant GetPrevValue();
```

Description

When the GetNeedPrevValue (see page 7832) method returns true, GetPrevValue returns the value for this field that should appear as a hidden field in an HTML form. This hidden value is included in request messages that originate from that form. Thus, GetPrevValue returns a value that this field sends to itself, which is transparent to the end-user.

GetPrevValue delegates the task of providing the hidden value to the ImplGetPrevValue (see page 7835) method.

See Also

GetNeedPrevValue (see page 7832)

ImplGetPrevValue (see page 7835)

41.1.6.2.5 TCustomDataSetAdapterField.ImplCheckOrigValue Method

Provides the underlying implementation for the CheckOrigValue (see page 7736) method.

Pascal

```
procedure ImplCheckOrigValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplCheckOrigValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The CheckOrigValue (see page 7736) method calls ImplCheckOrigValue to raise an exception if the current value of this field is not the same as the original value.

AActionRequest is interface of the action request object. ImplCheckOrigValue can use this interface to obtain information about the current request (such as the value of the hidden field that contains the original value).

AFieldIndex is the index of this field in the set of original field values that AActionRequest maintains.

ImplCheckOrigValue fetches the original value for this field from AActionRequest and compares it to the value of the associated field object. If the two values differ, or if this object is unable to locate the associated dataset field, ImplCheckOrigValue raises an exception. If the two values match, or if AActionRequest does not include an original value for this field, ImplCheckOrigValue does not raise an exception.

See Also

CheckOrigValue (see page 7736)

FieldFlags (see page 7752)

ActionOrigValue (see page 23943)

Value (see page 23945)

41.1.6.2.6 TCustomDataSetAdapterField.ImplCheckValueChange Method

Provides the underlying implementation for the CheckValueChange (see page 7737) method.

Pascal

```
function ImplCheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
override;
```

C++

```
virtual __fastcall Boolean ImplCheckValueChange(IActionRequest AActionRequest, int
AFieldIndex);
```

Description

The CheckValueChange (see page 7737) method calls ImplCheckValueChange to determine whether the value of this adapter field has changed from the value it held when building the form from which the current request originates.

AActionRequest is the interface of the action request object. CheckOrigValue (see page 7736) can use this interface to obtain information about the current request (such as the value of the hidden field that contains the original value).

AFieldIndex is the index of this field in the set of field values that AActionRequest maintains.

CheckValueChange (see page 7737) returns false if there has been no change, true if the value may have changed.

If the FieldFlags (see page 7752) property includes ffInOrigValues, ImplCheckValueChange fetches the original and current values from AActionRequest and compares them. If the current request does not include information about the original value, ImplCheckValueChange returns true, indicating that the value may have changed.

See Also

CheckValueChange (see page 7737)

Value (see page 23945)

FieldFlags (see page 7752)

41.1.6.2.7 TCustomDataSetAdapterField.ImplGetNeedPrevValue Method

Provides the underlying implementation for the GetNeedPrevValue (see page 7832) method.

Pascal

```
function ImplGetNeedPrevValue: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetNeedPrevValue();
```

Description

GetNeedPrevValue (see page 7832) calls ImplGetNeedPrevValue to determine whether an HTML form should include a hidden field that stores this field's value. This method returns true if the parent dataset adapter is in edit mode and this field's FieldFlags (see page 7752) property includes ffInOrigValues. Otherwise, ImplGetNeedPrevValue returns false.

By including the current field value as a hidden field when the dataset adapter is in edit mode, this field ensures that update requests from the HTML form include sufficient information for locating the record that should be updated. When the dataset adapter is in any other mode, there is no need to locate a particular record in the dataset.

See Also

GetNeedPrevValue (see page 7832)

GetPrevValue (see page 7832)

Mode (see page 7810)

FieldFlags (see page 7752)

41.1.6.2.8 TCustomDataSetAdapterField.ImplGetPrevValue Method

Provides the underlying implementation of the GetPrevValue (see page 7832) method.

Pascal

```
function ImplGetPrevValue: Variant; virtual;
```

C++

```
virtual __fastcall Variant ImplGetPrevValue();
```

Description

The GetPrevValue (see page 7832) calls ImplGetPrevValue to obtain the value of this field as it should appear in a hidden field on an HTML form.

ImplGetPrevValue checks the EchoActionFieldValue property, and if it is true, returns the value for this field that appears in the current request. If EchoActionFieldValue is false, ImplGetPrevValue returns the current value of the associated dataset field.

If ImplGetPrevValue is unable to obtain a value (if the current request does not include a value for this field or if this field can't locate the associated dataset field), it returns an unassigned Variant.

See Also

GetNeedPrevValue (see page 7832)

GetPrevValue (see page 7832)

EchoActionFieldValue (see page 23944)

Value (see page 7180)

41.1.6.2.9 TCustomDataSetAdapterField.ImplGetValue Method

Provides the underlying implementation for the GetValue method.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

GetValue calls ImplGetValue to obtain the value of the adapter field as a Variant.

If the value of the EchoActionFieldValue property is true, ImplGetValue uses the ActionValue property to obtain the value for this field that the user entered.

If EchoActionFieldValue is false (or if the user did not enter a value for this field), ImplGetValue first generates an OnGetValue (see page 7754) event, to allow an event handler to supply the value of the field. If there is no OnGetValue (see page 7754) event handler, ImplGetValue returns the value of the associated dataset field.

If ImplGetValue can't locate a value for this field, it returns an unassigned Variant.

See Also

Value (see page 7180)

GetValue (see page 23935)

OnGetValue (see page 7754)

Value (see page 23945)

EchoActionFieldValue (see page 23944)

ActionValue (see page 23943)

41.1.6.2.10 TCustomDataSetAdapterField.ImplUpdateValue Method

Provides the underlying implementation for the UpdateValue (see page 7751) method.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The UpdateValue (see page 7751) method calls ImplUpdateValue to respond when the user assigns a new value to this field.

AActionRequest is interface of the action request object. ImplUpdateValue can use this interface to obtain information about the current request (such as the value the user assigned to this field).

AFieldIndex is the index of this field in the set of field values that AActionRequest maintains.

ImplUpdateValue generates an OnUpdateValue (see page 7765) event so that you can apply the update using an event handler. If there is no OnUpdateValue (see page 7765) event handler, ImplUpdateValue writes the current field value from AActionRequest to the associated dataset field.

If ImplUpdateValue can't locate the dataset field to update, or if the request contained multiple values or a file reference rather than a value, ImplUpdateValue raises an exception.

Note: ImplUpdateValue does nothing if the parent dataset adapter is in query mode.

See Also

UpdateValue (see page 7751)

Value (see page 23945)

OnUpdateValue (see page 7765)

Mode (see page 7810)

41.1.6.2.11 TCustomDataSetAdapterField.Notification Method

Responds when components are added to or deleted from the WebSnap application.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are created or destroyed. After calling the inherited method, Notification checks whether the object that implements the ValuesList (see page 7837) property is about to be destroyed. If the adapter that lists values for this field is going away, Notification sets the ValuesList (see page 7837) property to nil (Delphi) or NULL (C++).

See Also

ValuesList (see page 7837)

41.1.6.3 TCustomDataSetAdapterField Properties

The properties of the TCustomDataSetAdapterField class are listed here.

Public Properties

	Name	Description
	ValuesList (see page 7837)	Specifies the interface of an object that lists the possible values for this field.

41.1.6.3.1 TCustomDataSetAdapterField.ValuesList Property

Specifies the interface of an object that lists the possible values for this field.

Pascal

```
property ValuesList: IValuesListAdapter;
```

C++

```
__property IValuesListAdapter ValuesList;
```

Description

ValuesList is the interface of an adapter that lists a set of values. It can be the interface of any object that supports the IValuesListAdapter interface, such as TStringsValuesList or TDataSetValuesList (see page 8079). Each value in the list that this adapter maintains represents a possible value for this field. This is useful when you want to restrict the possible values a user can assign to the associated database field.

When you assign a value to the ValuesList property, the default control for editing this fields value changes to a single selection list box.

See Also

TStringsValuesList (see page 24540)

TDataSetValuesList (see page 8079)

GetInputStyleType (see page 7831)

41.1.7 TCustomDataSetAdapterMemoField Class

The TCustomDataSetAdapterMemoField is the ancestor of TDataSetAdapterMemoField (see page 7996).

Class Hierarchy

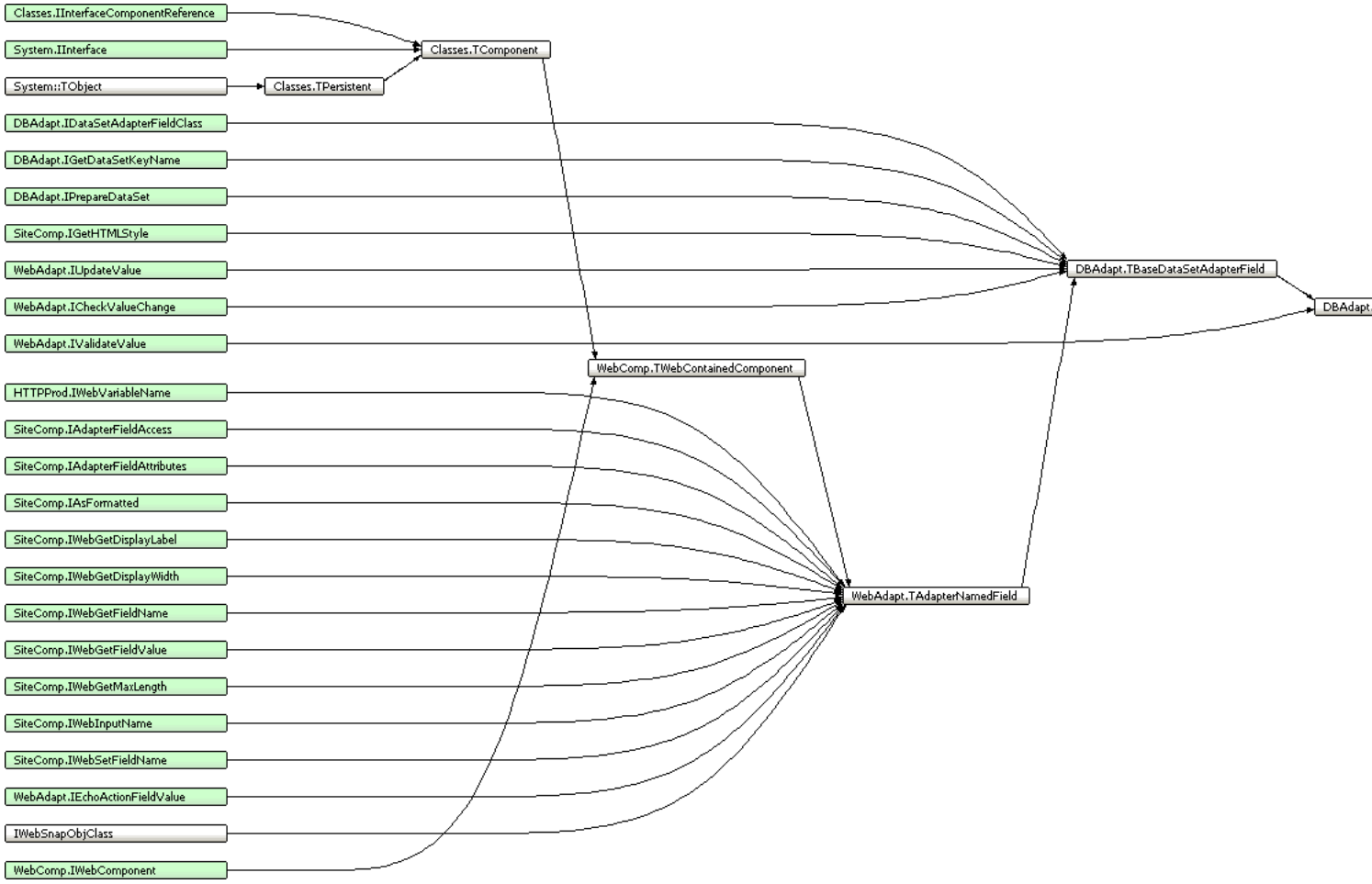

Pascal

```
TCustomDataSetAdapterMemoField = class(TBaseDataSetAdapterInputField);
```

C++

```
class TCustomDataSetAdapterMemoField : public TBaseDataSetAdapterInputField;
```

File

DBAdapt

Description

Derive from TCustomDataSetAdapterMemoField to define a field class that can be added to a TDataSetAdapter (see page 7872).

See Also

TDataSetAdapter (see page 7872)

TAdapterDisplayField (see page 25109)

TAdapterDisplayColumn (see page 25099)

TAdapterEditColumn (see page 25132)

41.1.7.1 TCustomDataSetAdapterMemoField Members

The following tables list the members exposed by TCustomDataSetAdapterMemoField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckOrigValue (see page 7736)	Ensures that the current value of this field is the same as the original value.
	CheckOrUpdateValue (see page 7845)	Called by ImplUpdateValue (see page 7846) and by ImplCheckValueChange (see page 7845).
	CheckValueChange (see page 7737)	Indicates whether the current value of this field is the same as the original value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 7738)	Indicates whether references to this field in HTML forms must be qualified with the name of the parent adapter.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 7738)	Returns the value of the Adapter (see page 7752) property.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7738)	Returns the dataset associated with this field's parent adapter.
V	GetDataSetFieldValue (see page 7739)	Returns the value of the associated dataset field.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 7739)	Returns a string value that suggests an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 7740)	Returns the HTML display style type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 7741)	Returns a string value that suggests an HTML editing style for this adapter field.
V	GetInputStyleType (see page 7845)	Returns htmliTextArea to indicate that this field should be represented by an HTML TextArea.
	GetKeyName (see page 7742)	Returns the name of the associated dataset field if it appears in the index (key) that the parent dataset adapter uses.

	GetKeyParams (see page 7743)	Returns a list of fields that the parent dataset adapter uses to locate records, along with their current values in that adapter's dataset.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetReadOnly (see page 7743)	Indicates whether the adapter field represents a read-only database field.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 7744)	Returns a string value that indicates what function a control that represents this field should perform.
	GetViewModeType (see page 7744)	Returns the view mode for the function a control that represents this field should perform.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 7845)	Returns the text that is used to display a read only representation of the dataset field.
	ImplCheckOrigValue (see page 7745)	Provides the underlying implementation for the CheckOrigValue (see page 7736) method.
	ImplCheckValueChange (see page 7845)	Indicates whether the data was changed by the user.
	ImplGetDisplayLabel (see page 7747)	Provides the underlying implementation for the GetDisplayLabel method.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayWidth (see page 7747)	Provides the underlying implementation for the GetDisplayWidth method.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetEchoActionFieldValue (see page 7763)	Provides the underlying implementation for the GetEchoActionFieldValue method.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 7747)	Provides the underlying implementation for the GetRequired method.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 7846)	Returns either the value of the associated dataset field as a Variant, or the last value entered by the end user.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 7748)	Provides the underlying implementation for the GetVisible method.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 7763)	Provides the underlying implementation for the SetEchoActionFieldValue method.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 7749)	Provides the underlying implementation for the SetFieldName method.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	ImplUpdateValue (see page 7846)	Verifies that the field has modify access rights, and if so updates the value for this TCustomDataSetAdapterMemoField (see page 7837).
	ImplValidateValue (see page 7764)	Provides the underlying implementation for the ValidateValue (see page 7765) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.

V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 7750)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	PrepareDataSet (see page 7750)	Responds when the parent dataset adapter prepares its dataset.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSetField (see page 7751)	Sets the value of the DataSetField (see page 7752) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	TestValueChange (see page 7764)	Checks whether a new value matches the current value of this field.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 7751)	Updates the adapter field when the user assigns a new value.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateValue (see page 7765)	Validates a new value for this field that appears in the current request.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7752)	Indicates the dataset adapter that uses this object to represent one of its fields.
R	Adapter (see page 23943)	Returns the parent adapter component.
	DataSetField (see page 7752)	Specifies the name of the dataset field that this adapter field represents.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 7754)	Occurs when the adapter field needs to supply its formatted value as a string that can appear in HTML forms.
	OnGetValue (see page 7754)	Occurs when the adapter field needs to supply its value as a Variant.
	OnValidateValue (see page 7766)	Occurs when the parent adapter wants to validate the new value for this field.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 7735)	Creates and initializes an instance of TBaseDataSetAdapterField (see page 7729).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 7737)	Returns the dataset field whose value this adapter field represents.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetKeyName (see page 8105)	This is GetKeyName, a member of class IGetDataSetKeyName.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PrepareDataSet (see page 8109)	This is PrepareDataSet, a member of class IPrepareDataSet.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.

W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	FieldFlags (see page 7752)	Indicates various attributes of the adapter field.
	FieldModes (see page 7753)	Indicates what dataset adapter modes this field supports.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
R	KeyName (see page 8105)	This is KeyName, a member of class IGetDataSetKeyName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
	OnUpdateValue (see page 7765)	Occurs when the WebSnap application updates this adapter field's value based on in incoming request.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.7.2 TCustomDataSetAdapterMemoField Methods

The methods of the TCustomDataSetAdapterMemoField class are listed here.

Protected Methods

	Name	Description
	CheckOrUpdateValue (see page 7845)	Called by ImplUpdateValue (see page 7846) and by ImplCheckValueChange (see page 7845).
V	GetInputStyleType (see page 7845)	Returns htmliTextArea to indicate that this field should be represented by an HTML TextArea.
V	ImplAsFormatted (see page 7845)	Returns the text that is used to display a read only representation of the dataset field.
V	ImplCheckValueChange (see page 7845)	Indicates whether the data was changed by the user.
V	ImplGetValue (see page 7846)	Returns either the value of the associated dataset field as a Variant, or the last value entered by the end user.
V	ImplUpdateValue (see page 7846)	Verifies that the field has modify access rights, and if so updates the value for this TCustomDataSetAdapterMemoField (see page 7837).

41.1.7.2.1 TCustomDataSetAdapterMemoField.CheckOrUpdateValue Method

Called by ImplUpdateValue (see page 7846) and by ImplCheckValueChange (see page 7845).

Pascal

```
function CheckOrUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer; AUpdate:
Boolean): Boolean;
```

C++

```
__fastcall Boolean CheckOrUpdateValue(IActionRequest AActionRequest, int AFieldIndex, Boolean
AUpdate);
```

Description

The CheckOrUpdateValue function is called by ImplUpdateValue (see page 7846) and by ImplCheckValueChange (see page 7845).

41.1.7.2.2 TCustomDataSetAdapterMemoField.GetInputStyleType Method

Returns htmliTextArea to indicate that this field should be represented by an HTML TextArea.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const AnsiString
AAdapterMode);
```

Description

The GetInputStyleType returns htmliTextArea to indicate that this field should be represented by an HTML TextArea. TAdapterDisplayField and TAdapterInputColumn request the TAdapterInputHTMLElementType when generating the default HTML representation of an adapter field.

See Also

TAdapterInputHTMLElementType

41.1.7.2.3 TCustomDataSetAdapterMemoField.ImplAsFormatted Method

Returns the text that is used to display a read only representation of the dataset field.

Pascal

```
function ImplAsFormatted: string; override;
```

C++

```
virtual __fastcall AnsiString ImplAsFormatted();
```

Description

The ImplAsFormatted function returns the text that is used to display a read only representation of the dataset field.

41.1.7.2.4 TCustomDataSetAdapterMemoField.ImplCheckValueChange Method

Indicates whether the data was changed by the user.

Pascal

```
function ImplCheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
override;
```

C++

```
virtual __fastcall Boolean ImplCheckValueChange(IActionRequest AActionRequest, int
AFieldIndex);
```

Description

The ImplCheckValueChange indicates whether the data was changed by the user.

41.1.7.2.5 TCustomDataSetAdapterMemoField.ImplGetValue Method

Returns either the value of the associated dataset field as a Variant, or the last value entered by the end user.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

The ImplGetValue function returns either the value of the associated dataset field as a Variant, or the last value entered by the end user. The last value entered by the end user is returned when generating a HTTP response to adapter action in which an error occurred.

41.1.7.2.6 TCustomDataSetAdapterMemoField.ImplUpdateValue Method

Verifies that the field has modify access rights, and if so updates the value for this TCustomDataSetAdapterMemoField (see page 7837).

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The ImplUpdateValue procedure first verifies that the field has modify access rights, and if so updates the value for this TCustomDataSetAdapterMemoField (see page 7837).

41.1.8 TCustomDataSetAdapterRowAction Class

TCustomDataSetAdapterRowAction is the base class for adapter actions that operate on the records of a dataset.

Class Hierarchy

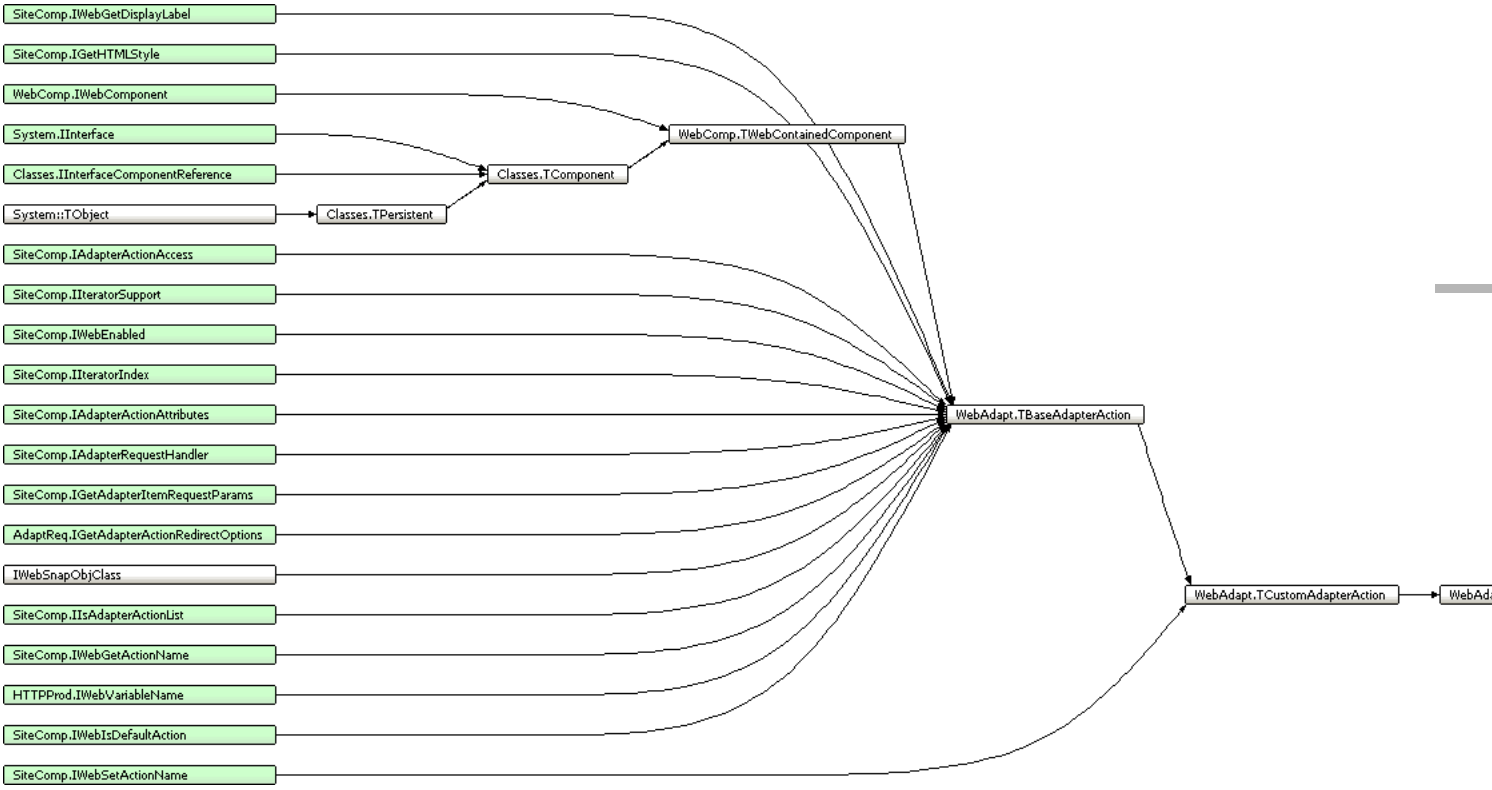

Pascal

```
TCustomDataSetAdapterRowAction = class(TCustomDataSetAdapterAction);
```

C++

```
class TCustomDataSetAdapterRowAction : public TCustomDataSetAdapterAction;
```

File

DBAdapt

Description

TCustomDataSetAdapterRowAction is the base class for all adapter action objects that operate on the records of the dataset associated with a dataset adapter. TCustomDataSetAdapterRowAction descendants are included as the actions of a dataset adapter (TDataSetAdapter (see page 7872)). Each TCustomDataSetAdapterRowAction descendant can be executed to perform a specific function that affects the records of the parent adapter's dataset.

Do not use TCustomDataSetAdapterRowAction directly. It is intended only as a base class. Instead, use a TCustomDataSetAdapterRowAction descendant that represents the particular action you want.

See Also

TDataSetAdapterApplyRowAction (see page 7890)

TDataSetAdapterBrowseRowAction (see page 7903)

TDataSetAdapterCancelRowAction (see page 7916)

TDataSetAdapterDeleteRowAction (see page 7934)

TDataSetAdapterEditRowAction (see page 7947)

TDataSetAdapterFirstRowAction (see page 7972)

TDataSetAdapterLastRowAction (see page 7984)

TDataSetAdapterNextRowAction (see page 8030)

TDataSetAdapterPrevRowAction (see page 8042)

TDataSetAdapterRefreshRowAction (see page 8054)

TDataSetAdapterNewRowAction (see page 8016)

TDataSetAdapter (see page 7872)

TDataSetAdapterRowAction (see page 8066)

41.1.8.1 TCustomDataSetAdapterRowAction Members

The following tables list the members exposed by TCustomDataSetAdapterRowAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 7819)	Returns the value of the Adapter (see page 7822) property.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7819)	Returns the value of the DataSet (see page 7822) property.
V	GetDefaultActionName (see page 24055)	Gets the default name used to reference this adapter action with server-side script.

	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 24063)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 24195)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 7854)	Provides the underlying implementation for the GetAdapterItemRequestParams method.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 7819)	Provides the underlying implementation for the GetVisible method.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.

	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
V	ImplWebEnabled (see page 7855)	Provides the underlying implementation for the WebEnabled method.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7822)	Indicates the dataset adapter that uses this object to represent one of its actions.
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 24075)	Called after the adapter action has executed.
	OnAfterExecute (see page 24198)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 24075)	Called after the adapter action response is generated.
	OnAfterGetResponse (see page 24199)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24076)	Called before the adapter action is executed.
	OnBeforeExecute (see page 24199)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24076)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 24200)	Called before the adapter action response is generated.
	OnExecute (see page 7822)	Occurs when server-side script executes this action.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 7823)	Occurs when the WebSnap application to determine whether a control or hyperlink for this action should be enabled.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 24077)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 7853)	Creates and initializes an instance of TCustomDataSetAdapterRowAction (see page 7846).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.

	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionModes (see page 7821)	Indicates what dataset adapter modes this field supports.
	ActionName (see page 7821)	Identifies the name of this action when using server-side script.
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 7822)	Indicates the dataset on which this action operates.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 24073)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 24073)	Defines the rights required to execute this action.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RedirectOptions (see page 24074)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.8.2 TCustomDataSetAdapterRowAction.Create Constructor

Creates and initializes an instance of TCustomDataSetAdapterRowAction (see page 7846).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomDataSetAdapterRowAction(TComponent * AOwner);
```

Description

The TCustomDataSetAdapterRowAction (see page 7846) constructor allocates memory for the adapter action object and initializes its properties.

AOwner is the component that is responsible for starting the streaming process when a module containing the adapter action is loaded or saved. It becomes the value of the Owner property. Typically, AOwner is the Web module.

See Also

Owner (see page 2807)

41.1.8.3 TCustomDataSetAdapterRowAction Methods

The methods of the TCustomDataSetAdapterRowAction class are listed here.

Protected Methods

	Name	Description
	ImplGetAdapterItemRequestParams (see page 7854)	Provides the underlying implementation for the GetAdapterItemRequestParams method.
	ImplWebEnabled (see page 7855)	Provides the underlying implementation for the WebEnabled method.

41.1.8.3.1 TCustomDataSetAdapterRowAction.ImplGetAdapterItemRequestParams Method

Provides the underlying implementation for the GetAdapterItemRequestParams method.

Pascal

```
procedure ImplGetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams); override;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(AnsiString AIdentifier,
IAdapterItemRequestParams Params);
```

Description

The GetAdapterItemRequestParams method calls ImplGetAdapterItemRequestParams to obtain the parameters this action needs when it executes. The name/value pairs this method provides are added to the HTML form and passed back in any HTTP request that form generates.

AIdentifier returns a unique identifier for this action object, which the WebSnap application uses to dispatch requests from the generated form to this object.

Params is an interface that ImplGetAdapterItemRequestParams uses to write the name/value pairs it needs.

ImplGetAdapterItemRequestParams first generates an OnGetActionParams event on the parent adapter, and then an OnGetParams event on this action object.

See Also

GetAdapterItemRequestParams (see page 24055)

OnGetParams (see page 24077)

OnGetActionParams (see page 24187)

41.1.8.3.2 TCustomDataSetAdapterRowAction.ImplWebEnabled Method

Provides the underlying implementation for the WebEnabled method.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

The WebEnabled method calls ImplWebEnabled to determine whether the control or hyperlink for this action should appear enabled in the HTML form. WebEnabled returns true if controls should be enabled, false otherwise.

ImplWebEnabled generates an OnGetEnabled (see page 7823) event so that you can specify whether controls for this action are enabled. If there is no OnGetEnabled (see page 7823) event handler, ImplWebEnabled returns true if the parent adapter's GetKeyFields method returns a set of fields that can locate records in the associated dataset.

See Also

WebEnabled (see page 24071)

OnGetEnabled (see page 7823)

GetKeyFields (see page 7786)

41.1.9 TCustomDataSetValuesList Class

TCustomDataSetValuesList is the base class for components that represent a set of name/value pairs taken from a dataset.

Class Hierarchy

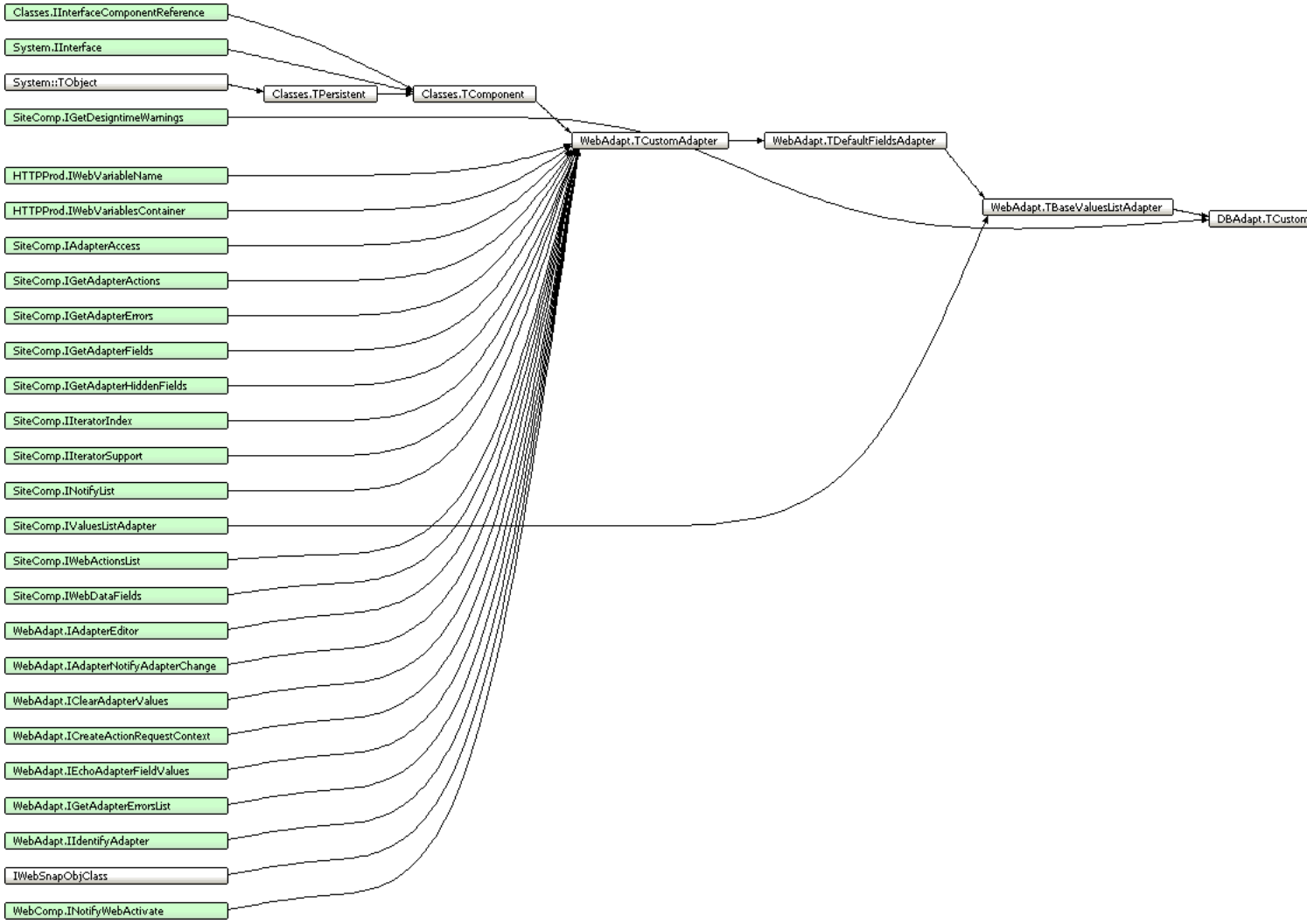

Pascal

```
TCustomDataSetValuesList = class(TBaseValuesListAdapter, IGetDesigntimeWarnings);
```

C++

```
class TCustomDataSetValuesList : public TBaseValuesListAdapter, public IGetDesigntimeWarnings;
```

File

DBAdapt

Description

Use TCustomDataSetValuesList as a base class for components that implement the ValuesList property of an adapter field. This class represents a set of name/value pairs, where the names and the values are taken from two columns of a dataset. Thus, this adapter acts like a lookup dataset, although it can be used with any adapter field that has a ValuesList property (it is not limited to dataset adapter fields).

Do not use TCustomDataSetValuesList directly in a WebSnap application. Instead, assign a TCustomDataSetValuesList descendant such as TDataSetValuesList (see page 8079) as the value of an adapter field's ValuesList property.

See Also

TDataSet (see page 6894)

TDataSetValuesList (see page 8079)

TCustomDataSetAdapterField (see page 7823)

TAdapterUpdateField (see page 24007)

41.1.9.1 TCustomDataSetValuesList Members

The following tables list the members exposed by TCustomDataSetValuesList.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.

	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetDesigntimeWarnings (see page 7864)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetListImage (see page 24128)	Returns an image associated with the current name/value pair.
	GetListImageOfValue (see page 24128)	Looks up an image for a value.
	GetListName (see page 24128)	Responsible returning the name of the name/value pair.
	GetListNameOfValue (see page 24128)	Looks up a name for a value.
	GetListValue (see page 24129)	Responsible for returning the value of the name/value pair.
V	GetNameFieldClass (see page 7865)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	GetValueFieldClass (see page 7865)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
V	GetWebSnapObjClass (see page 24130)	Returns the class type for this adapter field.
V	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 24130)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 24163)	Clears adapter state information.

	ImplEndIterator (see page 7866)	Provides the underlying implementation for the EndIterator method.
	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 24130)	Adds default field components to this adapter.
	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetListImage (see page 24131)	Responsible returning an image associated with the current name/value pair.
	ImplGetListImageOfValue (see page 24131)	Looks up an image for a value.
	ImplGetListName (see page 7866)	Provides the underlying implementation for the GetListName method.
	ImplGetListNameOfValue (see page 7867)	Provides the underlying implementation for the GetListNameOfValue method.
	ImplGetListValue (see page 7867)	Provides the underlying implementation for the GetListValue method.
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 7867)	Provides the underlying implementation for the NextIteration method.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 7868)	Provides the underlying implementation for the NotifyDeactivate method.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 7868)	Provides the underlying implementation for the StartIterator method.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 7869)	Responds when components are created or destroyed.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.

	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DataSetPrepared (see page 7870)	Indicates whether the OnPrepareDataSet (see page 7871) event has occurred.
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 24184)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24185)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24186)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24186)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 24187)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.
	OnPrepareDataSet (see page 7871)	Occurs before the dataset values list uses its associated dataset.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 7863)	Creates and initializes an instance of TCustomDataSetValuesList (see page 7855).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 7864)	Destroys an instance of TCustomDataSetValuesList (see page 7855).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.

	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
	GetListImage (see page 19741)	Returns the image object associated with the current name/value pair.
	GetListImageOfValue (see page 19742)	Returns the image object associated with a specified value.
	GetListName (see page 19742)	Returns the name part of the current name/value pair.
	GetListNameOfValue (see page 19742)	Returns the name associated with a specified value.
	GetListValue (see page 19743)	Returns the value of the current name/value pair.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 24181)	Actions is the container for adapter actions components.
R	Actions (see page 24402)	Container for adapter actions components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Data (see page 24181)	Data is the container for adapter field components.
R	Data (see page 24402)	Container for adapter field components.
	DataSet (see page 7869)	Specifies the dataset that stores the names and values for this list.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
	NameField (see page 7870)	Indicates the field in the associated dataset that supplies the name part of this list's name/value pairs.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ValueField (see page 7871)	Indicates the field in the associated dataset that supplies the value part of this list's name/value pairs.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.9.2 TCustomDataSetValuesList.Create Constructor

Creates and initializes an instance of TCustomDataSetValuesList (see page 7855).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomDataSetValuesList(TComponent * AOwner);
```

Description

The TCustomDataSetValuesList (see page 7855) constructor allocates memory for the TCustomDataSetValuesList (see page 7855) object and initializes its properties.

AOwner is the component that is responsible for starting the streaming process when a module containing the data set values list is loaded or saved. It becomes the value of the Owner property. Typically, AOwner is the Web module.

See Also

Owner (see page 2807)

41.1.9.3 TCustomDataSetValuesList.Destroy Destructor

Destroys an instance of TCustomDataSetValuesList (see page 7855).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomDataSetValuesList();
```

Description

Typically, applications do not need to explicitly free data set values list objects. These objects persist as long as the adapter field that uses them, and are freed when the Web module is destroyed.

Do not call Destroy directly. When you must delete a dataset values list at runtime, use the Free method, which first checks that the dataset values list reference is not nil.

See Also

Free (see page 22300)

41.1.9.4 TCustomDataSetValuesList Methods

The methods of the TCustomDataSetValuesList class are listed here.

Protected Methods

	Name	Description
	GetDesigntimeWarnings (see page 7864)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
V	GetNameFieldClass (see page 7865)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
V	GetValueFieldClass (see page 7865)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
V	ImplEndIterator (see page 7866)	Provides the underlying implementation for the EndIterator method.
V	ImplGetListName (see page 7866)	Provides the underlying implementation for the GetListName method.
V	ImplGetListNameOfValue (see page 7867)	Provides the underlying implementation for the GetListNameOfValue method.
V	ImplGetListValue (see page 7867)	Provides the underlying implementation for the GetListValue method.
V	ImplNextIteration (see page 7867)	Provides the underlying implementation for the NextIteration method.
V	ImplNotifyDeactivate (see page 7868)	Provides the underlying implementation for the NotifyDeactivate method.
V	ImplStartIterator (see page 7868)	Provides the underlying implementation for the StartIterator method.
V	Notification (see page 7869)	Responds when components are created or destroyed.

41.1.9.4.1 TCustomDataSetValuesList.GetDesigntimeWarnings Method

Populates a design-time warnings object with any warnings that the dataset values list needs to emit.

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

GetDesigntimeWarnings is called automatically at design time when the adapter page producer is assembling the current version of its HTML form. This gives the dataset values list an opportunity to add messages to the version of the HTML form that appears at design time, describing problems that you can rectify.

AWarnings is the design-time warnings object that assembles design-time warning messages that appear in the HTML form.

If the DataSet (see page 7869) property has not been set, GetDesigntimeWarnings adds a message that indicates this omission. If DataSet (see page 7869) is set, but the dataset is not active, GetDesigntimeWarnings adds a message that indicates this problem. Finally, if the ValueField (see page 7871) property is not set, GetDesigntimeWarnings adds a message indicating this omission.

See Also

DataSet (see page 7869)

Active (see page 7004)

ValueField (see page 7871)

41.1.9.4.2 TCustomDataSetValuesList.GetNameFieldClass Method

Returns the class of adapter field that is responsible for providing the value of the name/value pair.

Pascal

```
function GetNameFieldClass: TComponentClass; override;
```

C++

```
virtual __fastcall TComponentClass GetNameFieldClass();
```

Description

GetNameFieldClass returns TValuesListNameField. This is the class used to implement the adapter field called NameField (see page 7870). The value of the NameField (see page 7870) adapter field is the value of the dataset field specified by the NameField (see page 7870) property.

See Also

GetValueFieldClass (see page 7865)

NameField (see page 7870)

41.1.9.4.3 TCustomDataSetValuesList.GetValueFieldClass Method

Returns the class of adapter field that is responsible for providing the name of the name/value pair.

Pascal

```
function GetValueFieldClass: TComponentClass; override;
```

C++

```
virtual __fastcall TComponentClass GetValueFieldClass();
```

Description

GetValueFieldClass returns TValuesListNameField. This is the class used to implement the adapter field called ValueField (see page 7871). The value of the ValueField (see page 7871) adapter field is the value of the dataset field specified by the

ValueField (see page 7871) property.

See Also

GetNameFieldClass (see page 7865)

ValueField (see page 7871)

41.1.9.4.4 TCustomDataSetValuesList.ImplEndIterator Method

Provides the underlying implementation for the EndIterator method.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer); override;
```

C++

```
virtual __fastcall ImplEndIterator(void * OwnerData);
```

Description

The EndIterator method calls ImplEndIterator to respond when server-side script stops an enumeration of the name/value pairs in this list. ImplEndIterator clears the internal data that TCustomDataSetValuesList (see page 7855) maintains to keep track of the current record when iterating over the values in the list.

OwnerData is a pointer to adapter-specific information returned by the last call to the StartIterator or NextIteration method. TCustomDataSetValues does not use the OwnerData parameter.

See Also

EndIterator (see page 24152)

StartIterator (see page 24176)

NextIteration (see page 24173)

41.1.9.4.5 TCustomDataSetValuesList.ImplGetListName Method

Provides the underlying implementation for the GetListName method.

Pascal

```
function ImplGetListName: string; override;
```

C++

```
virtual __fastcall AnsiString ImplGetListName();
```

Description

The GetListName method calls ImplGetListName to obtain the name part of the current name/value pair. This is the implementation of the ListName property that can be called from server-side script.

ImplGetListName returns the value of the field that the NameField (see page 7870) property specifies for the current record. The current record is determined by the current iteration on this object.

See Also

GetListName (see page 24128)

NameField (see page 7870)

41.1.9.4.6 TCustomDataSetValuesList.ImplGetListNameOfValue Method

Provides the underlying implementation for the GetListNameOfValue method.

Pascal

```
function ImplGetListNameOfValue(Value: Variant): string; override;
```

C++

```
virtual __fastcall AnsiString ImplGetListNameOfValue(Variant Value);
```

Description

The GetListNameOfValue method calls ImplGetListNameOfValue to obtain the name that corresponds to a specified value.

Value is the value part of the name/value pair for which this method returns the name part.

ImplGetListNameOfValue searches the name/value pairs in the list. It returns the name part of the first pair it finds where the value part is equal to Value.

See Also

GetListNameOfValue (see page 24128)

41.1.9.4.7 TCustomDataSetValuesList.ImplGetListValue Method

Provides the underlying implementation for the GetListValue method.

Pascal

```
function ImplGetListValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetListValue();
```

Description

The GetListValue method calls ImplGetListValue to obtain the value part of the current name/value pair. This is the implementation of the ListValue property that can be called from server-side script.

ImplGetListValue returns the value of the field that the ValueField (see page 7871) property specifies for the current record. The current record is determined by the current iteration on this object.

See Also

GetListValue (see page 24129)

ValueField (see page 7871)

41.1.9.4.8 TCustomDataSetValuesList.ImplNextIteration Method

Provides the underlying implementation for the NextIteration method.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

The NextIteration method calls ImplNextIteration to move to the next name/value pair when server-side script enumerates the

name/value pairs in this list.

OwnerData is a pointer to adapter-specific information that the caller should supply to the next call to the NextIteration or EndIterator method. TCustomDataSetValues does not use the OwnerData parameter.

ImplNextIteration returns true if it successfully moves to the next name/value pair, false if there were no more name/value pairs in the list.

See Also

NextIteration (see page 24173)

StartIterator (see page 24176)

EndIterator (see page 24152)

41.1.9.4.9 TCustomDataSetValuesList.ImplNotifyDeactivate Method

Provides the underlying implementation for the NotifyDeactivate method.

Pascal

```
procedure ImplNotifyDeactivate; override;
```

C++

```
virtual __fastcall ImplNotifyDeactivate();
```

Description

The NotifyDeactivate method calls ImplNotifyDeactivate to respond when the Web module that contains this adapter is deactivated.

ImplNotifyDeactivate closes the dataset that is the value of the DataSet (see page 7869) property, and sets the protected DataSetPrepared (see page 7870) property to false. This ensures that the next time the WebSnap application activates the Web module (in response to a new HTTP request), the dataset values list must re-initialize and reopen its dataset, refreshing the name/value pairs in the list.

See Also

NotifyDeactivate (see page 24175)

DataSet (see page 7869)

DataSetPrepared (see page 7870)

41.1.9.4.10 TCustomDataSetValuesList.ImplStartIterator Method

Provides the underlying implementation for the StartIterator method.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

The StartIterator method calls ImplStartIterator to start the process of enumerating the name/value pairs in this list in response to server-side script.

OwnerData returns a pointer to adapter-specific information that should be passed to the next call to the NextIteration or EndIterator method. TCustomDataSetValues does not use the OwnerData parameter.

ImplStartIterator returns true if it successfully moves to the first name/value pair, false if there were no name/value pairs in the list.

See Also

StartIterator (see page 24176)

NextIteration (see page 24173)

EndIterator (see page 24152)

41.1.9.4.11 TCustomDataSetValuesList.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TCustomDataSetValuesList (see page 7855) overrides the Notification method to check whether the component that is the value of the DataSet (see page 7869) property is about to be freed. If so, it sets the DataSet (see page 7869) property to nil (Delphi) or NULL (C++).

See Also

DataSet (see page 7869)

41.1.9.5 TCustomDataSetValuesList Properties

The properties of the TCustomDataSetValuesList class are listed here.

Protected Properties

	Name	Description
	DataSetPrepared (see page 7870)	Indicates whether the OnPrepareDataSet (see page 7871) event has occurred.

Public Properties

	Name	Description
	DataSet (see page 7869)	Specifies the dataset that stores the names and values for this list.
	NameField (see page 7870)	Indicates the field in the associated dataset that supplies the name part of this list's name/value pairs.
	ValueField (see page 7871)	Indicates the field in the associated dataset that supplies the value part of this list's name/value pairs.

41.1.9.5.1 TCustomDataSetValuesList.DataSet Property

Specifies the dataset that stores the names and values for this list.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Use DataSet to specify the dataset from which this adapter gets its name/value pairs. The names of the name/value pairs in this list are the values of the field that the NameField (see page 7870) property specifies. The values are the corresponding values of the field that the ValueField (see page 7871) property specifies.

Note: When the WebSnap application reads the value of DataSet for the first time, TCustomDataSetValuesList (see page 7855) generates an OnPrepareDataSet (see page 7871) event. This lets you perform any initializations that ensure the dataset contains the desired data.

See Also

NameField (see page 7870)

ValueField (see page 7871)

OnPrepareDataSet (see page 7871)

41.1.9.5.2 TCustomDataSetValuesList.DataSetPrepared Property

Indicates whether the OnPrepareDataSet (see page 7871) event has occurred.

Pascal

```
property DataSetPrepared: Boolean;
```

C++

```
__property Boolean DataSetPrepared;
```

Description

TCustomDataSetValuesList (see page 7855) uses DataSetPrepared internally to keep track of whether it has initialized the dataset for its own use. When the WebSnap application reads the DataSet (see page 7869) property, TCustomDataSetValuesList (see page 7855) checks DataSetPrepared, and if the dataset has not been prepared, it sets up the dataset for use and changes DataSetPrepared to true. When the Web module that contains this values list is deactivated, DataSetPrepared changes back to false.

DataSetPrepared is true if the dataset has been prepared (that is, if TDataSetValuesList (see page 8079) has generated an OnPrepareDataSet (see page 7871) event). DataSetPrepared is false if the dataset needs to be prepared before the values list can use it.

See Also

DataSet (see page 7869)

OnPrepareDataSet (see page 7871)

41.1.9.5.3 TCustomDataSetValuesList.NameField Property

Indicates the field in the associated dataset that supplies the name part of this list's name/value pairs.

Pascal

```
property NameField: string;
```

C++

```
__property AnsiString NameField;
```

Description

Set NameField to specify the dataset field that holds the names of this adapter's name/value pairs. Each entry in TCustomDataSetValuesList (see page 7855) is a name/value pair that corresponds to the values of the NameField and ValueField (see page 7871) fields of the associated dataset for a single record in the dataset. NameField is the same as the FieldName property of the dataset field that supplies the names.

See Also

DataSet (see page 7869)

FieldName (see page 7164)

ValueField (see page 7871)

41.1.9.5.4 TCustomDataSetValuesList.ValueField Property

Indicates the field in the associated dataset that supplies the value part of this list's name/value pairs.

Pascal

```
property ValueField: string;
```

C++

```
__property AnsiString ValueField;
```

Description

Set ValueField to specify the dataset field that holds the values of this adapter's name/value pairs. Each entry in TCustomDataSetValuesList (see page 7855) is a name/value pair that corresponds to the values of the NameField (see page 7870) and ValueField fields of the associated dataset for a single record in the dataset. ValueField is the same as the FieldName property of the dataset field that supplies the values.

See Also

DataSet (see page 7869)

FieldName (see page 7164)

NameField (see page 7870)

41.1.9.6 TCustomDataSetValuesList Events

The events of the TCustomDataSetValuesList class are listed here.

Public Events

	Name	Description
	OnPrepareDataSet (see page 7871)	Occurs before the dataset values list uses its associated dataset.

41.1.9.6.1 TCustomDataSetValuesList.OnPrepareDataSet Event

Occurs before the dataset values list uses its associated dataset.

Pascal

```
property OnPrepareDataSet: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPrepareDataSet;
```

Description

Write an OnPrepareDataSet event handler to prepare the associated dataset before this component fetches names and values from it. This lets you customize the data for each incoming request. For example, you can use this event to set query parameters based on parameters in the current HTTP request and then execute the query. (Use the global Web context to obtain information about the current HTTP request.)

TCustomDataSetValuesList (see page 7855) generates an OnPrepareDataSet event the first time the application reads the DataSet (see page 7869) property for a new HTTP request.

See Also

DataSet (see page 7869)

WebContext (see page 24687)

41.1.10 TDataSetAdapter Class

TDataSetAdapter defines a scriptable interface to a TDataSet component and allows it to operate properly in a stateless environment.

Class Hierarchy

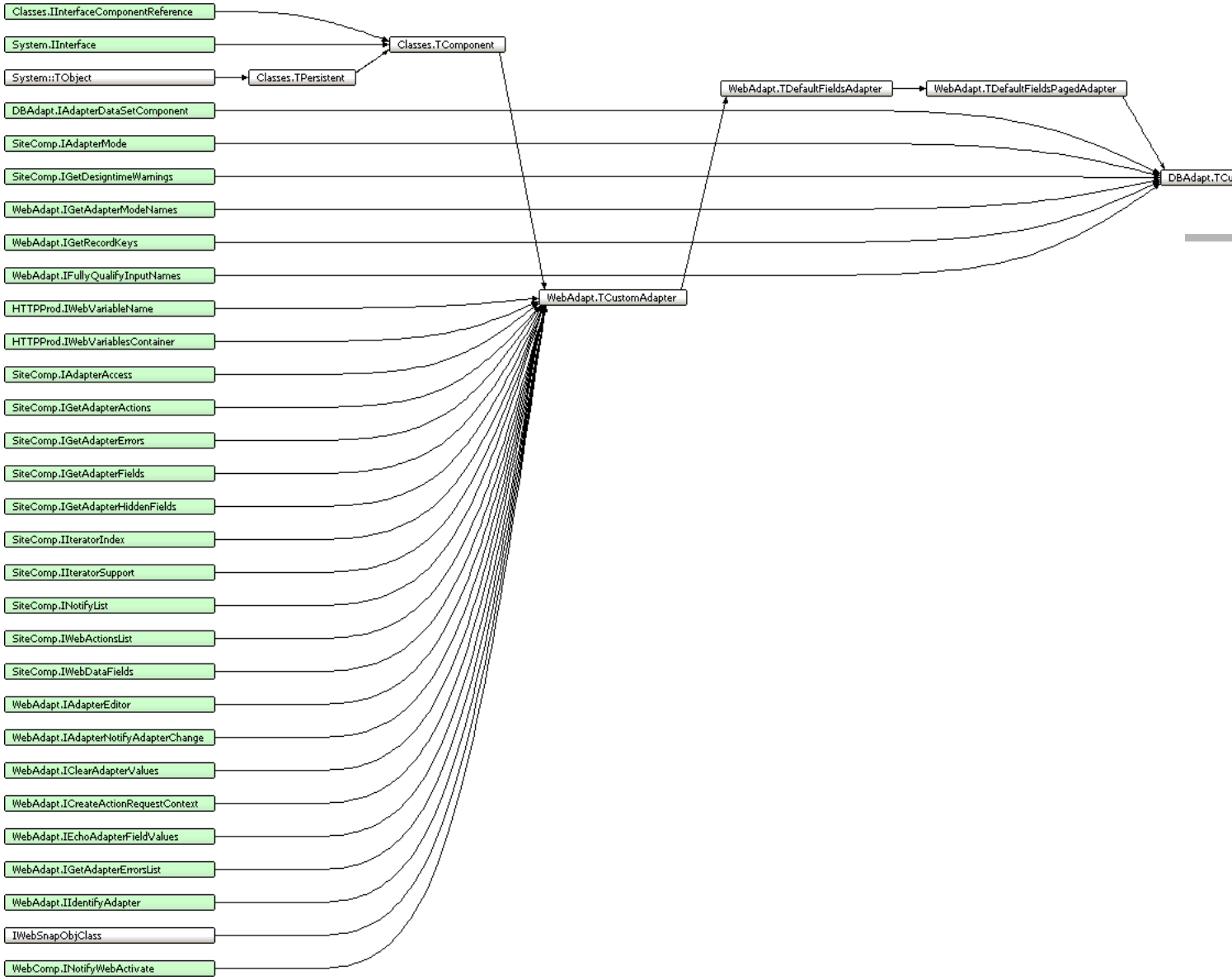

Pascal

```
TDataSetAdapter = class(TCustomDataSetAdapter);
```

C++

```
class TDataSetAdapter : public TCustomDataSetAdapter;
```

File

DBAdapt

Description

Use TDataSetAdapter like TAdapter to define a custom interface to application logic. (See TAdapter for a general description of Adapters.) Add adapter field components to the TDataSetAdapter to define the dataset fields that a script can access. Set

properties of the adapter field components to augment the meta data provided by the dataset fields. For example, the ValuesList property defines name/value pairs that can be used to populate HTML elements for editing the field such as select lists and radio buttons. If you do not add any adapter fields, all adapter fields for all supported dataset field types are added by default. Default adapter fields are not shown in the object tree view but can be accessed with script and will be used when generating a form or grid with the TAdapterPageProducer.

Add adapter action components to the TDataSetAdapter to define the commands that can be executed on the dataset. TDataSetAdapter has built-in support for commands such as Delete, Next Row, and Edit. If you do not add any adapter actions, all actions are added by default. Default adapter actions are not shown in the object tree view but can be accessed with script and will be used when generating a form or grid with the TAdapterPageProducer.

The TDataSetAdapter has a mode. The mode values include Edit, Insert, Browse, and Query. The mode of the TDataSetAdapter will affect the adapter field values retrieved by script statements. The mode may also affect the HTML generated by the TAdapterPageProducer component. For example, a form field will be displayed using an input element when in Edit, Insert, or Query mode but displayed as read-only text when in Browse mode. The TDataSetAdapter mode is modified during the execution of adapter action components such as TDataSetAdapterEditRowAction (see page 7947). The mode can also be modified using server side script (for example; <% Adapter.Mode (see page 7810) = 'Edit' %>). To cause the TAdapterPageProducer to generate script to modify the mode, change the AdapterMode property of a TAdapterFieldGroup component.

The TDataSetAdapter supports master/detail relationships between datasets. You must associate a TDataSetAdapter with each master and detail dataset and use the MasterAdapter (see page 7884) property to indicate the master/detail relationship between the TDataSetAdapter components.

In a WebSnap application there is no guarantee that the same web module will be used to service consecutive requests from the same user. The TDataSetAdapter manages the state of the dataset component to ensure data integrity. TDataSetAdapter responsibilities include opening and closing the dataset, locating records, validating edits, posting changes, canceling changes, and handling exceptions.

41.1.10.1 TDataSetAdapter Members

The following tables list the members exposed by TDataSetAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 7778)	Adds the default action components to this adapter.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultActions (see page 24411)	Adds default action components to this adapter.
V	AddDefaultFields (see page 7779)	Adds the default field components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.

	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
V	EncodeActionParams (see page 24411)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
	EndOfPage (see page 24412)	Indicates whether the record index has gone past the last record in the current page.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
V	ExtractRequestParams (see page 24412)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	FullyQualifyInputNames (see page 7782)	Indicates whether the HTML input names must be qualified with the name of the adapter.
V	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetAdapterMode (see page 7783)	Returns a string that represents the current value of the Mode (see page 7810) property.
	GetAdapterModeNames (see page 7783)	Retrieves the strings for all the possible modes of this adapter.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetDataSet (see page 7784)	Returns the value of the DataSet (see page 7807) property.
V	GetDefaultMode (see page 7784)	Returns the default mode for this adapter.
	GetDesigntimeWarnings (see page 7784)	Populates a design-time warnings object with any warnings that the dataset adapter needs to emit.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
V	GetEOF (see page 7785)	Indicates whether the adapter is positioned at the end of its dataset.
V	GetEOF (see page 24412)	Indicates whether the record position has passed the last record.

	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
	GetFirstRecord (see page 7785)	Positions the adapter on the first record of its dataset.
	GetFirstRecord (see page 24413)	Set the record position to the first record.
	GetHiddenFieldOptions (see page 7786)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetKeyFields (see page 7786)	Returns a string listing the fields that this adapter uses to locate records.
	GetKeyParams (see page 7787)	Retrieves a list of fields that the dataset adapter uses to locate records, along with their current values.
	GetKeyParamStrings (see page 7787)	Retrieves a list of fields that the dataset adapter uses to locate records, along with their current values.
	GetMasterDetailAdapters (see page 7788)	Returns a list of all dataset adapters, including this one, linked to this adapter in a master/detail relationship.
	GetNeedAdapterMode (see page 7788)	Indicates whether the adapter requires a Mode (see page 7810) value.
	GetNextRecord (see page 7789)	Positions the associated dataset on the next record.
	GetNextRecord (see page 24413)	Increments the record position.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPageCount (see page 24413)	Calculates the number of pages needed to display all adapter records.
	GetRecordCount (see page 7789)	Returns the value of the RecordCount property.
	GetRecordCount (see page 24414)	Returns the total number of records.
	GetRecordIndex (see page 7790)	Returns the value of the RecordIndex property.
	GetRecordIndex (see page 24414)	Returns the index of the current record.
	GetRecordKeys (see page 7790)	Retrieves a list of field values that the dataset adapter uses to locate records, along with information for formatting hidden parameters from this values.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 7790)	Provides the underlying implementation for the CanAddActionClass method.
	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddActionClass (see page 24415)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 7791)	Provides the underlying implementation for the CanAddFieldClass method.
	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
	ImplEndIterator (see page 7792)	Implements IteratorSupport's EndIterator method to support iterating over the data records of the adapter.
	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
	ImplEndIterator (see page 24415)	Implements IteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 7792)	Provides the underlying implementation for the GetActionsList method.

	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
	ImplGetActionsList (see page 24415)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 7793)	Provides the underlying implementation for the GetFieldsList method.
	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 7793)	Provides the underlying implementation for the GetIteratorIndex method.
	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetRecordKeys (see page 7794)	Retrieves a list of field values that the dataset adapter uses to locate records, along with information for formatting hidden parameters from this values.
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 7794)	Provides the underlying implementation for the InIterator method.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNextIteration (see page 24416)	Implements IteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 7795)	Provides the underlying implementation for the NotifyDeactivate method.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplNotifyDeactivate (see page 24416)	Clears the current page variable.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 7795)	Provides the underlying implementation for the SetEchoActionFieldValues method.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 7795)	Implements IteratorSupport.StartIterator to support iterating over the records of the adapter.
	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	ImplStartIterator (see page 24416)	Implements IteratorSupport.StartIterator (see page 24176) to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	LocateKeyParams (see page 7797)	Positions the dataset based on a specified set of field values.
	LocateRequestRecord (see page 7798)	Positions the dataset based on the parameters in the current request.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 2783)	Forwards notification messages to all owned components.

	Notification (see page 7798)	Responds when components are added to or deleted from the WebSnap application.
	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	OnDataSetStateChange (see page 7798)	Responds to changes in the associated dataset.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PrepareDataSet (see page 7799)	Sets up the associated dataset for use by this adapter.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RestoreDefaultAdapterMode (see page 7801)	Returns the adapter to its default mode.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAdapterMode (see page 7801)	Sets the Mode (see page 7810) property based on a string that represents one of the available values.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDataSet (see page 7802)	Sets the value of the DataSet (see page 7807) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetMode (see page 7802)	Sets the value of the Mode (see page 7810) property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetPageSize (see page 24417)	Updates the pagesize variable.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SkipToPage (see page 24417)	Skips records until the current record is within the current page.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateFieldsExecuteValues (see page 7803)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 7804)	Indicates whether the values in the HTTP request change the value of any adapter fields.
	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 7805)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 7805)	Validates the values passed in the current HTTP request for the adapter's fields.
	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	AdapterRowNotFound (see page 7807)	Indicates the dataset adapter for which this adapter could not locate the current record.
	DataSetPrepared (see page 7808)	Indicates whether the associated dataset has been properly set up for use by this adapter.
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnGetEOF (see page 24420)	Implement this event to indicate when the current record index has exceeded the total record count.
	OnGetFirstRecord (see page 24420)	Implement this method to initialize the iteration index to zero.
	OnGetNextRecord (see page 24421)	Implement this method to increment the iteration index.
	OnGetRecordIndex (see page 24421)	Implement this event to identify the zero-based current record.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	ApplyAdapterMode (see page 7780)	Sets the Mode (see page 7810) property to reflect the value that the current request specifies.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 7775)	Creates an initializes an instance of TCustomDataSetAdapter (see page 7766).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 7776)	Frees an instance of TCustomDataSetAdapter (see page 7766).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EncodeActionParams (see page 7780)	Writes name/value pairs that represent this adapter's state.
	EncodeActionParamsFlags (see page 7781)	Writes name/value pairs that represent a specified type of information.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExtractRequestParams (see page 7781)	Reads name/value pairs from the current HTTP request.
	ExtractRequestParamsFlags (see page 7782)	Reads name/value pairs that represent a specified type of information from the current HTTP request.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	FullyQualifyInputNames (see page 24577)	This is FullyQualifyInputNames, a member of class IFullyQualifyInputNames.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetAdapterMode (see page 19678)	This is GetAdapterMode, a member of class IAdapterMode.
	GetAdapterModeNames (see page 24579)	This is GetAdapterModeNames, a member of class IGetAdapterModeNames.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetDataSet (see page 8103)	This is GetDataSet, a member of class IAdapterDataSetComponent.
	GetDesigntimeWarnings (see page 19697)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNeedAdapterMode (see page 19678)	This is GetNeedAdapterMode, a member of class IAdapterMode.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetRecordKeys (see page 24580)	This is GetRecordKeys, a member of class IGetRecordKeys.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.

	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	Locate (see page 7796)	Positions the associated dataset on the record identified by the LocateParamsList (see page 7809) property.
	LocateAndApply (see page 7796)	Positions the dataset and updates the resulting record based on the parameters of the current request.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RaiseFieldChangedByAnotherUser (see page 7799)	Raises an EAdapterFieldException with a message indicating that another user changed a field value.
	RaiseFieldNotFound (see page 7800)	Raises an EAdapterException with a message indicating that a specified field could not be found.
	RaiseNilDataSet (see page 7800)	Raises an EAdapterException with a message indicating that the dataset is unassigned.
	RaiseRowNotFound (see page 7800)	Raises an EAdapterException with a message indicating that a dataset record could not be located based on the current set of key fields.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
	RestoreDefaultAdapterMode (see page 19679)	This is RestoreDefaultAdapterMode, a member of class IAdapterMode.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAdapterMode (see page 19679)	This is SetAdapterMode, a member of class IAdapterMode.
	SetDataSet (see page 8103)	This is SetDataSet, a member of class IAdapterDataSetComponent.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SilentLocate (see page 7802)	Positions the associated dataset on a specified record.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	StringToMode (see page 7803)	Converts a mode string to the corresponding TDataSetAdapterMode (see page 8111) value.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
	AdapterMode (see page 19679)	This is AdapterMode, a member of class IAdapterMode.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	CurrentPage (see page 24418)	Sets or gets the current page index.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DetailAdapters (see page 7808)	Lists the dataset adapters for all datasets that are details of this one.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
	InDefaultMode (see page 7808)	Indicates whether the dataset adapter is in its default mode.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	LocateParamsList (see page 7809)	Lists the parameters that identify the current record of this dataset and all other datasets linked to it in a master/detail relationship.
	Mode (see page 7810)	Indicates how forms built from this dataset adapter operate on the information it provides.
R	NeedAdapterMode (see page 19679)	This is NeedAdapterMode, a member of class IAdapterMode.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	PageCount (see page 24418)	Calculates the number of pages needed to display all adapter records.
R	RecordCount (see page 24419)	Returns the total number of records.
R	RecordIndex (see page 24419)	Returns the index of the current record.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 7886)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 7886)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 7887)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 7887)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 7887)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 7888)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 7888)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 7888)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 7889)	Supplies name/value pairs needed to execute an adapter action.
	OnGetRecordCount (see page 7889)	Implement this event to identify the total number of records that can be iterated.
	OnPrepareDataSet (see page 7889)	Occurs before the dataset adapter uses any information from its associated dataset.

Published Properties

	Name	Description
	Actions (see page 7883)	Container for adapter actions components.
	Data (see page 7883)	Container for adapter field components.
	DataSet (see page 7884)	Specifies the dataset that this adapter represents.
	MasterAdapter (see page 7884)	Indicates the dataset adapter, if any, to which this adapter is linked in a master/detail relationship.
	ModifyAccess (see page 7884)	This is ModifyAccess, a member of class TDataSetAdapter.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	PageSize (see page 7884)	Sets or gets the page size variable.

	RefreshAfterPost (see page 7885)	Indicates whether the data set should be refreshed or not after posting..
	Tag (see page 2808)	Stores an integer value as part of a component.
	ViewAccess (see page 7885)	Identifies the rights required to view this adapter's fields.

41.1.10.2 TDataSetAdapter Properties

The properties of the TDataSetAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 7883)	Container for adapter actions components.
	Data (see page 7883)	Container for adapter field components.
	DataSet (see page 7884)	Specifies the dataset that this adapter represents.
	MasterAdapter (see page 7884)	Indicates the dataset adapter, if any, to which this adapter is linked in a master/detail relationship.
	ModifyAccess (see page 7884)	This is ModifyAccess, a member of class TDataSetAdapter.
	PageSize (see page 7884)	Sets or gets the page size variable.
	RefreshAfterPost (see page 7885)	Indicates whether the data set should be refreshed or not after posting..
	ViewAccess (see page 7885)	Identifies the rights required to view this adapter's fields.

41.1.10.2.1 TDataSetAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 23770)

41.1.10.2.2 TDataSetAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 23778)

41.1.10.2.3 TDataSetAdapter.DataSet Property

Specifies the dataset that this adapter represents.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;__property TDataSet DataSet;
```

Description

Set DataSet to associate this adapter with an existing dataset component. Read this property to access the dataset component that this adapter represents.

Note: Do not set DataSet to a unidirectional dataset. Unidirectional datasets do not work with the way dataset adapters position themselves on records.

41.1.10.2.4 TDataSetAdapter.MasterAdapter Property

Indicates the dataset adapter, if any, to which this adapter is linked in a master/detail relationship.

Pascal

```
property MasterAdapter: TCustomDataSetAdapter;
```

C++

```
__property TCustomDataSetAdapter MasterAdapter;
```

Description

Set MasterAdapter to link this adapter as a detail to another dataset adapter in a master/detail relationship. If MasterAdapter is nil (Delphi) or NULL (C++), this adapter does not represent the detail set of another adapters dataset. If MasterAdapter is not nil (Delphi) or NULL (C++), this dataset adapter is listed by the master adapter's DetailAdapters property.

See Also

DetailAdapters (see page 7808)

41.1.10.2.5 TDataSetAdapter.ModifyAccess Property

Pascal

```
property ModifyAccess: string;
```

C++

```
__property AnsiString ModifyAccess;
```

Description

This is ModifyAccess, a member of class TDataSetAdapter.

41.1.10.2.6 TDataSetAdapter.PageSize Property

Sets or gets the page size variable.

Pascal

```
property PageSize: Integer;
```

C++

```
__property int PageSize;
```

Description

If the page size variable is set to a value greater than zero and no actions have been explicitly added to the adapter then the NextPage, PrevPage, and GotoPage are automatically added. If the page size variable is changed back to zero then the these actions are removed.

See Also

TAdapterNextPageAction (see page 23945)

TAdapterPrevPageAction (see page 23979)

TAdapterGotoPageAction (see page 23849)

41.1.10.2.7 TDataSetAdapter.RefreshAfterPost Property

Indicates whether the data set should be refreshed or not after posting..

Pascal

```
property RefreshAfterPost: Boolean;
```

C++

```
__property Boolean RefreshAfterPost;
```

Description

RefreshAfterPost is used to determine whether or not the data set should be refreshed after it has been posted.

41.1.10.2.8 TDataSetAdapter.ViewAccess Property

Identifies the rights required to view this adapter's fields.

Pascal

```
property ViewAccess: string;
```

C++

```
__property AnsiString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view fields of this adapter. When server-side script accesses the CanView property, the ViewAccess property value is checked against the end user's rights.

See Also

TWebUserList (see page 25625)

HasViewAccess (see page 24162)

implhasviewaccess (see page 24169)

41.1.10.3 TDataSetAdapter Events

The events of the TDataSetAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 7886)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 7886)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 7887)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 7887)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 7887)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 7888)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 7888)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 7888)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 7889)	Supplies name/value pairs needed to execute an adapter action.
	OnGetRecordCount (see page 7889)	Implement this event to identify the total number of records that can be iterated.
	OnPrepareDataSet (see page 7889)	Occurs before the dataset adapter uses any information from its associated dataset.

41.1.10.3.1 TDataSetAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

41.1.10.3.2 TDataSetAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.10.3.3 TDataSetAdapter.OnAfterUpdateFields Event

Occurs after values in the HTTP request are used to update adapter fields.

Pascal

```
property OnAfterUpdateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterUpdateFields;
```

Description

Implement this event handler to perform custom updates. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record update errors.

See Also

UpdateFieldsUpdateValues (see page 24178)

OnBeforeUpdateFields (see page 24187)

Errors (see page 24182)

41.1.10.3.4 TDataSetAdapter.OnAfterValidateFields Event

Occurs after values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnAfterValidateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record validation errors.

See Also

UpdateFieldsValidateValues (see page 24178)

OnBeforeValidateFields (see page 24187)

Errors (see page 24182)

41.1.10.3.5 TDataSetAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from

being executed.

See Also

OnBeforeExecute (see page 24076)

OnAfterExecuteAction (see page 24184)

41.1.10.3.6 TDataSetAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 24076)

OnAfterGetActionResponse (see page 24185)

41.1.10.3.7 TDataSetAdapter.OnBeforeUpdateFields Event

Occurs before values in the HTTP request are used to update adapter fields.

Pascal

```
property OnBeforeUpdateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeUpdateFields;
```

Description

Implement this event handler to perform custom updates of adapter fields. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further updates. Add update errors to the adapter's errors object.

See Also

UpdateFieldsUpdateValues (see page 24178)

OnAfterUpdateFields (see page 24185)

Errors (see page 24182)

41.1.10.3.8 TDataSetAdapter.OnBeforeValidateFields Event

Occurs before values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnBeforeValidateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further validation. Add validation errors to the adapter's errors object.

See Also

UpdateFieldsValidateValues (see page 24178)

OnAfterValidateFields (see page 24186)

Errors (see page 24182)

41.1.10.3.9 TDataSetAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 24077)

41.1.10.3.10 TDataSetAdapter.OnGetRecordCount Event

Implement this event to identify the total number of records that can be iterated.

Pascal

```
property OnGetRecordCount: TGetRecordCountEvent;
```

C++

```
__property TGetRecordCountEvent OnGetRecordCount;
```

Description

OnGetRecordCount is called to calculate the number of pages.

41.1.10.3.11 TDataSetAdapter.OnPrepareDataSet Event

Occurs before the dataset adapter uses any information from its associated dataset.

Pascal

```
property OnPrepareDataSet: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPrepareDataSet;
```

Description

Write an OnPrepareDataSet event handler to set up the associated dataset before the WebSnap application uses this adapter. For example, you can use an OnPrepareDataSet event handler to dynamically generate and execute a query.

This event occurs when the Web module containing the dataset adapter is first activated, and whenever the WebSnap application changes the value of the Mode property.

Note: Do not confuse the this event with the Prepared property of many dataset components. This event allows the adapter to perform any tasks it requires to accurately represent its corresponding dataset. The Prepared property of a dataset component indicates whether the underlying database engine has initialized itself to represent a database query.

See Also

PrepareDataSet (see page 7799)

DataSet (see page 7807)

Mode (see page 7810)

41.1.11 TDataSetAdapterApplyRowAction Class

TDataSetAdapterApplyRowAction applies the updates from an HTTP request to a single record of an adapter's dataset.

Class Hierarchy

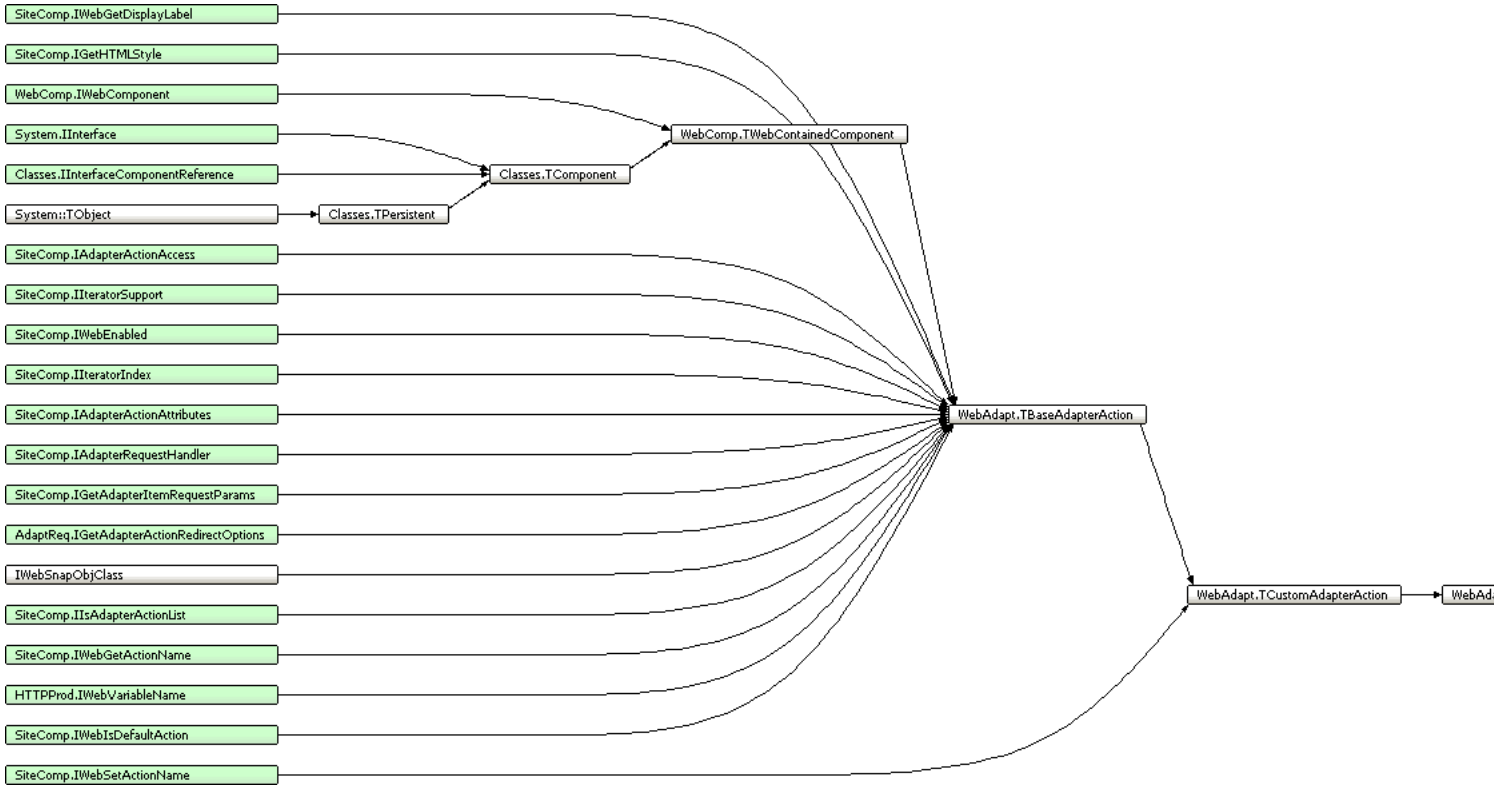

Pascal

```
TDataSetAdapterApplyRowAction = class(TCustomDataSetAdapterRowAction);
```

C++

```
class TDataSetAdapterApplyRowAction : public TCustomDataSetAdapterRowAction;
```

File

DBAdapt

Description

Add TDataSetAdapterApplyRowAction to a dataset adapter (TDataSetAdapter (see page 7872)) to support HTML forms that update a set of field values in the record of a dataset. When this action is executed, it positions the parent dataset on the record specified by the current request object and updates any fields for which new values are included in the current request.

Server-side script addresses this action as the 'Apply' action of the parent adapter. By default, this action is visible (included in HTML forms) only when the parent adapter is in insert or edit mode. You can, however, cause this action to be included when the dataset is in other modes by changing the value of its ActionModes (see page 7899) property. If the action is visible when the dataset is in browse mode, the control for this action appears disabled.

This action is included by default when the adapter is used to build forms that represent a single database record that is in edit or insert mode.

See Also

TDataSetAdapterBrowseRowAction (see page 7903)

TDataSetAdapterCancelRowAction (see page 7916)

TDataSetAdapterDeleteRowAction (see page 7934)

TDataSetAdapterEditRowAction (see page 7947)

TDataSetAdapterFirstRowAction (see page 7972)

TDataSetAdapterLastRowAction (see page 7984)

TDataSetAdapterNextRowAction (see page 8030)

TDataSetAdapterPrevRowAction (see page 8042)

TDataSetAdapterRefreshRowAction (see page 8054)

TDataSetAdapterNewRowAction (see page 8016)

TDataSetAdapter (see page 7872)

TCustomDataSetAdapter (see page 7766)

41.1.11.1 TDataSetAdapterApplyRowAction Members

The following tables list the members exposed by TDataSetAdapterApplyRowAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.

	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 7819)	Returns the value of the Adapter (see page 7822) property.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7819)	Returns the value of the DataSet (see page 7822) property.
V	GetDefaultActionName (see page 7898)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.

	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
V	ImplExecuteActionRequest (see page 7898)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 7854)	Provides the underlying implementation for the GetAdapterItemRequestParams method.
V	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
V	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
V	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetVisible (see page 7819)	Provides the underlying implementation for the GetVisible method.
V	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
V	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
V	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
V	ImplIsDefaultAction (see page 7898)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
V	ImplWebEnabled (see page 7899)	Provides the underlying implementation for the WebEnabled method.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.

V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7822)	Indicates the dataset adapter that uses this object to represent one of its actions.
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 7822)	Occurs when server-side script executes this action.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 7823)	Occurs when the WebSnap application to determine whether a control or hyperlink for this action should be enabled.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.

	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionName (see page 7821)	Identifies the name of this action when using server-side script.
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 7822)	Indicates the dataset on which this action operates.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 7901)	Called after the adapter action has executed.

	OnAfterGetResponse (see page 7902)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 7902)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 7902)	Called before the adapter action response is generated.
	OnGetParams (see page 7903)	Specifies name/value pairs needed to execute this adapter action.

Published Methods

	Name	Description
	Create (see page 7897)	Creates and initializes an instance of TDataSetAdapterApplyRowAction (see page 7890).

Published Properties

	Name	Description
	ActionModes (see page 7899)	Indicates what dataset adapter modes this field supports.
	DisplayLabel (see page 7900)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 7900)	Defines the rights required to execute this action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RedirectOptions (see page 7901)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.11.2 TDataSetAdapterApplyRowAction.Create Constructor

Creates and initializes an instance of TDataSetAdapterApplyRowAction (see page 7890).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDataSetAdapterApplyRowAction(TComponent * AOwner);
```

Description

The TDataSetAdapterApplyRowAction (see page 7890) constructor allocates memory for the adapter action object and initializes its properties.

AOwner is the component that is responsible for starting the streaming process when a module containing the adapter action is loaded or saved. It becomes the value of the Owner property. Typically, AOwner is the Web module.

See Also

Owner (see page 2807)

41.1.11.3 TDataSetAdapterApplyRowAction Methods

The methods of the TDataSetAdapterApplyRowAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 7898)	Gets the default name used to reference this adapter action with server-side script.
	ImplExecuteActionRequest (see page 7898)	Executes the adapter action using parameters passed in the HTTP request.
	ImplIsDefaultAction (see page 7898)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplWebEnabled (see page 7899)	Provides the underlying implementation for the WebEnabled method.

41.1.11.3.1 TDataSetAdapterApplyRowAction.GetDefaultActionName Method

Gets the default name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultActionName();
```

Description

GetDefaultActionName returns the name of the adapter action component. Descendants may override this function.

See Also

ImplGetActionName (see page 24063)

41.1.11.3.2 TDataSetAdapterApplyRowAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest create a TStrings containing the name/value pairs in AActionRequest and calls the OnExecute event handler.

See Also

ImplExecuteActionRequest (see page 24063)

OnExecute (see page 24200)

41.1.11.3.3 TDataSetAdapterApplyRowAction.ImplIsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function ImplIsDefaultAction(ADisplay: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplIsDefaultAction(TObject * ADisplay);
```

Description

ImplIsDefaultAction is called by IsDefaultAction. ImplIsDefaultAction returns true. ImplIsDefaultAction may be overridden by a descendant class.

See Also

IsDefaultAction (see page 24070)

ImplIsDefaultAction (see page 24243)

41.1.11.3.4 TDataSetAdapterApplyRowAction.ImplWebEnabled Method

Provides the underlying implementation for the WebEnabled method.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

The WebEnabled method calls ImplWebEnabled to determine whether the control or hyperlink for this action should appear enabled in the HTML form. WebEnabled returns true if controls should be enabled, false otherwise.

ImplWebEnabled generates an OnGetEnabled event so that you can specify whether controls for this action are enabled. If there is no OnGetEnabled event handler, ImplWebEnabled returns true if the parent adapter's GetKeyFields method returns a set of fields that can locate records in the associated dataset.

See Also

WebEnabled (see page 24071)

OnGetEnabled (see page 7823)

GetKeyFields (see page 7786)

41.1.11.4 TDataSetAdapterApplyRowAction Properties

The properties of the TDataSetAdapterApplyRowAction class are listed here.

Published Properties

	Name	Description
	ActionModes (see page 7899)	Indicates what dataset adapter modes this field supports.
	DisplayLabel (see page 7900)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 7900)	Defines the rights required to execute this action.
	RedirectOptions (see page 7901)	Indicates whether to perform an HTTP redirect after successfully executing this action.

41.1.11.4.1 TDataSetAdapterApplyRowAction.ActionModes Property

Indicates what dataset adapter modes this field supports.

Pascal

```
property ActionModes: TDataSetAdapterModes;
```

C++

```
__property TDataSetAdapterModes ActionModes;
```

Description

ActionModes is a set of all the modes in which this adapter action can be used. If the parent dataset adapter is in a mode that is not included in this set, any forms generated using that adapter do not include a control for executing this action. Thus, for example, there is no sense in adding a control that applies edits to a form that presents a read-only view of the data. An apply action, then, should not include amBrowse in its action modes.

See Also

CanModify (see page 7154)

Adapter (see page 7752)

Mode (see page 7810)

GetDisplayStyle (see page 7739)

GetInputStyle (see page 7741)

GetViewMode (see page 7744)

41.1.11.4.2 TDataSetAdapterApplyRowAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

41.1.11.4.3 TDataSetAdapterApplyRowAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property AnsiString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 25625)

hasexecuteaccess (see page 24162)

41.1.11.4.4 TDataSetAdapterApplyRowAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect (see page 15201)

41.1.11.5 TDataSetAdapterApplyRowAction Events

The events of the TDataSetAdapterApplyRowAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 7901)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 7902)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 7902)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 7902)	Called before the adapter action response is generated.
	OnGetParams (see page 7903)	Specifies name/value pairs needed to execute this adapter action.

41.1.11.5.1 TDataSetAdapterApplyRowAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

41.1.11.5.2 TDataSetAdapterApplyRowAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.11.5.3 TDataSetAdapterApplyRowAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

41.1.11.5.4 TDataSetAdapterApplyRowAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.11.5.5 TDataSetAdapterApplyRowAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

41.1.12 TDataSetAdapterBrowseRowAction Class

TDataSetAdapterBrowseRowAction applies the updates from an HTTP request to a single record of an adapter's dataset and changes the dataset adapter to browse mode.

Class Hierarchy

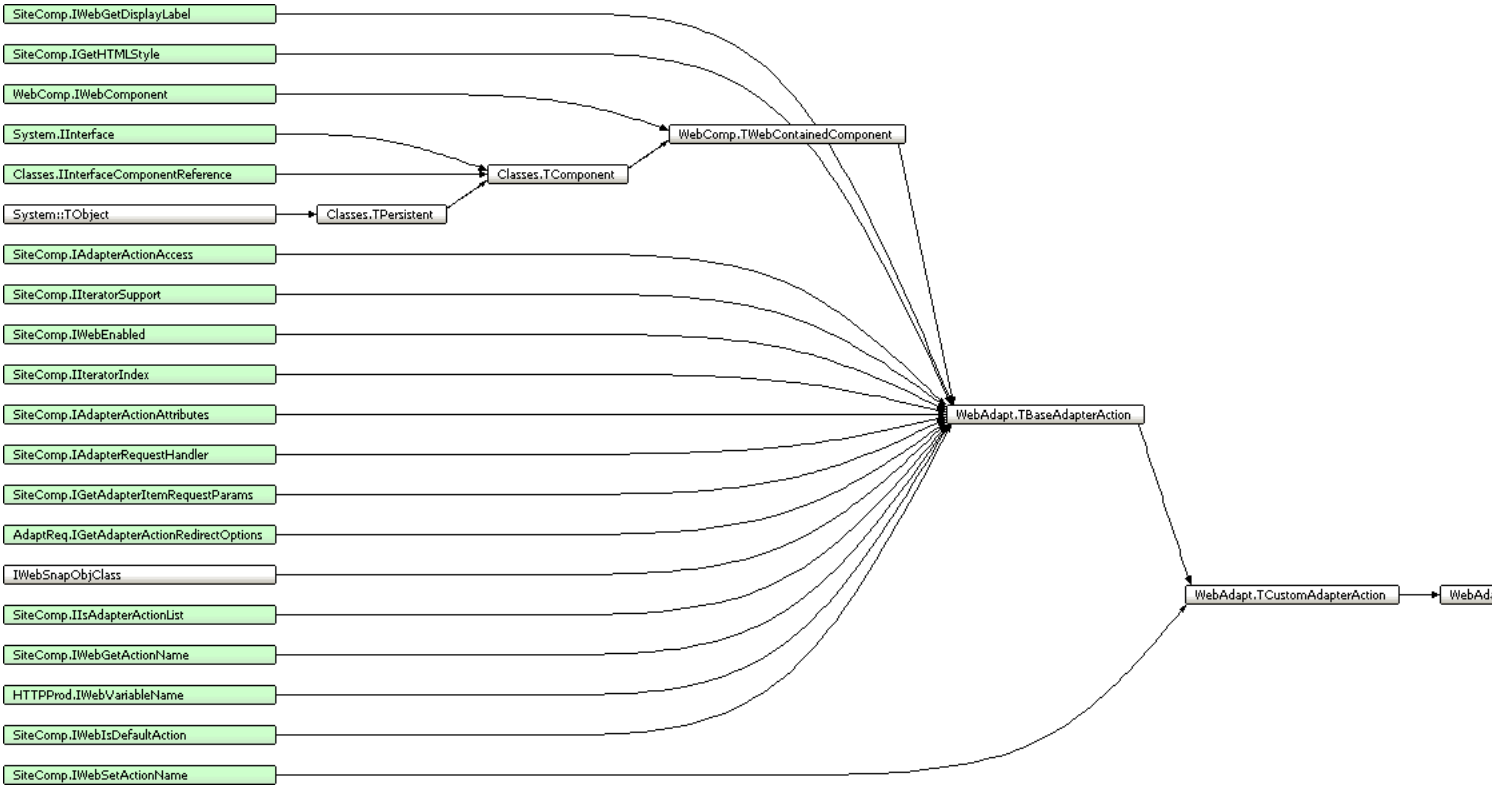

Pascal

```
TDataSetAdapterBrowseRowAction = class(TCustomDataSetAdapterRowAction);
```

C++

```
class TDataSetAdapterBrowseRowAction : public TCustomDataSetAdapterRowAction;
```

File

DBAdapt

Description

Add TDataSetAdapterBrowseRowAction to a dataset adapter (TDataSetAdapter (see page 7872)) to support HTML forms that can change from editing or inserting database information to read-only viewing of the data. When this action is executed, it positions the parent dataset on the record specified by the current request object and updates any fields for which new values are included in the current request. It then changes the parent dataset's mode to browse mode.

Server-side script addresses this action as the 'BrowseRow' action of the parent adapter. This action is included by default when the adapter is used to build forms that represent a multiple database records.

See Also

TDataSetAdapterApplyRowAction (see page 7890)

TDataSetAdapterCancelRowAction (see page 7916)

TDataSetAdapterDeleteRowAction (see page 7934)

TDataSetAdapterEditRowAction (see page 7947)

TDataSetAdapterFirstRowAction (see page 7972)

TDataSetAdapterLastRowAction (see page 7984)

TDataSetAdapterNextRowAction (see page 8030)

TDataSetAdapterPrevRowAction (see page 8042)

TDataSetAdapterRefreshRowAction (see page 8054)

TDataSetAdapterNewRowAction (see page 8016)

TDataSetAdapter (see page 7872)

TCustomDataSetAdapter (see page 7766)

TCustomDataSetAdapter (see page 7766)

41.1.12.1 TDataSetAdapterBrowseRowAction Members

The following tables list the members exposed by TDataSetAdapterBrowseRowAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 7819)	Returns the value of the Adapter (see page 7822) property.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7819)	Returns the value of the DataSet (see page 7822) property.
V	GetDefaultActionName (see page 7911)	Gets the default name used to reference this adapter action with server-side script.

	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 7911)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 7854)	Provides the underlying implementation for the GetAdapterItemRequestParams method.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 7819)	Provides the underlying implementation for the GetVisible method.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 7912)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.

	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
V	ImplWebEnabled (see page 7912)	Provides the underlying implementation for the WebEnabled method.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7822)	Indicates the dataset adapter that uses this object to represent one of its actions.
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 7822)	Occurs when server-side script executes this action.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 7823)	Occurs when the WebSnap application to determine whether a control or hyperlink for this action should be enabled.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.

	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionName (see page 7821)	Identifies the name of this action when using server-side script.
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.

	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 7822)	Indicates the dataset on which this action operates.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RedirectOptions (see page 24074)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 7914)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 7915)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 7915)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 7915)	Called before the adapter action response is generated.
	OnGetParams (see page 7916)	Specifies name/value pairs needed to execute this adapter action.

Published Methods

	Name	Description
V	Create (see page 7910)	Creates and initializes an instance of TDataSetAdapterBrowseRowAction (see page 7903).

Published Properties

	Name	Description
	ActionModes (see page 7913)	Indicates what dataset adapter modes this field supports.
	DisplayLabel (see page 7913)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 7914)	Defines the rights required to execute this action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.12.2 TDataSetAdapterBrowseRowAction.Create Constructor

Creates and initializes an instance of TDataSetAdapterBrowseRowAction (see page 7903).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDataSetAdapterBrowseRowAction(TComponent * AOwner);
```

Description

The TDataSetAdapterBrowseRowAction (see page 7903) constructor allocates memory for the adapter action object and initializes its properties.

AOwner is the component that is responsible for starting the streaming process when a module containing the adapter action is loaded or saved. It becomes the value of the Owner property. Typically, AOwner is the Web module.

See Also

Owner (see page 2807)

41.1.12.3 TDataSetAdapterBrowseRowAction Methods

The methods of the TDataSetAdapterBrowseRowAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 7911)	Gets the default name used to reference this adapter action with server-side script.
	ImplExecuteActionRequest (see page 7911)	Executes the adapter action using parameters passed in the HTTP request.
	ImplIsDefaultAction (see page 7912)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplWebEnabled (see page 7912)	Provides the underlying implementation for the WebEnabled method.

41.1.12.3.1 TDataSetAdapterBrowseRowAction.GetDefaultActionName Method

Gets the default name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultActionName();
```

Description

GetDefaultActionName returns the name of the adapter action component. Descendants may override this function.

See Also

ImplGetActionName (see page 24063)

41.1.12.3.2 TDataSetAdapterBrowseRowAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest create a TStrings containing the name/value pairs in AActionRequest and calls the OnExecute event handler.

See Also

ImplExecuteActionRequest (see page 24063)

OnExecute (see page 24200)

41.1.12.3.3 TDataSetAdapterBrowseRowAction.ImplIsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function ImplIsDefaultAction(ADisplay: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplIsDefaultAction(TObject * ADisplay);
```

Description

ImplIsDefaultAction is called by IsDefaultAction. ImplIsDefaultAction returns true. ImplIsDefaultAction may be overridden by a descendant class.

See Also

IsDefaultAction (see page 24070)

ImplIsDefaultAction (see page 24243)

41.1.12.3.4 TDataSetAdapterBrowseRowAction.ImplWebEnabled Method

Provides the underlying implementation for the WebEnabled method.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

The WebEnabled method calls ImplWebEnabled to determine whether the control or hyperlink for this action should appear enabled in the HTML form. WebEnabled returns true if controls should be enabled, false otherwise.

ImplWebEnabled generates an OnGetEnabled event so that you can specify whether controls for this action are enabled. If there is no OnGetEnabled event handler, ImplWebEnabled returns true if the parent adapter's GetKeyFields method returns a set of fields that can locate records in the associated dataset.

See Also

WebEnabled (see page 24071)

OnGetEnabled (see page 7823)

GetKeyFields (see page 7786)

41.1.12.4 TDataSetAdapterBrowseRowAction Properties

The properties of the TDataSetAdapterBrowseRowAction class are listed here.

Published Properties

	Name	Description
	ActionModes (see page 7913)	Indicates what dataset adapter modes this field supports.
	DisplayLabel (see page 7913)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 7914)	Defines the rights required to execute this action.

41.1.12.4.1 TDataSetAdapterBrowseRowAction.ActionModes Property

Indicates what dataset adapter modes this field supports.

Pascal

```
property ActionModes: TDataSetAdapterModes;
```

C++

```
__property TDataSetAdapterModes ActionModes;
```

Description

ActionModes is a set of all the modes in which this adapter action can be used. If the parent dataset adapter is in a mode that is not included in this set, any forms generated using that adapter do not include a control for executing this action. Thus, for example, there is no sense in adding a control that applies edits to a form that presents a read-only view of the data. An apply action, then, should not include amBrowse in its action modes.

See Also

CanModify (see page 7154)

Adapter (see page 7752)

Mode (see page 7810)

GetDisplayStyle (see page 7739)

GetInputStyle (see page 7741)

GetViewMode (see page 7744)

41.1.12.4.2 TDataSetAdapterBrowseRowAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

41.1.12.4.3 TDataSetAdapterBrowseRowAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property AnsiString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 25625)

hasexecuteaccess (see page 24162)

41.1.12.5 TDataSetAdapterBrowseRowAction Events

The events of the TDataSetAdapterBrowseRowAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 7914)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 7915)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 7915)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 7915)	Called before the adapter action response is generated.
	OnGetParams (see page 7916)	Specifies name/value pairs needed to execute this adapter action.

41.1.12.5.1 TDataSetAdapterBrowseRowAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

41.1.12.5.2 TDataSetAdapterBrowseRowAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.12.5.3 TDataSetAdapterBrowseRowAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

41.1.12.5.4 TDataSetAdapterBrowseRowAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.12.5.5 TDataSetAdapterBrowseRowAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

41.1.13 TDataSetAdapterCancelRowAction Class

TDataSetAdapterBrowseRowAction (see page 7903) cancels edits made in an HTML form and changes the dataset adapter to browse mode.

Class Hierarchy

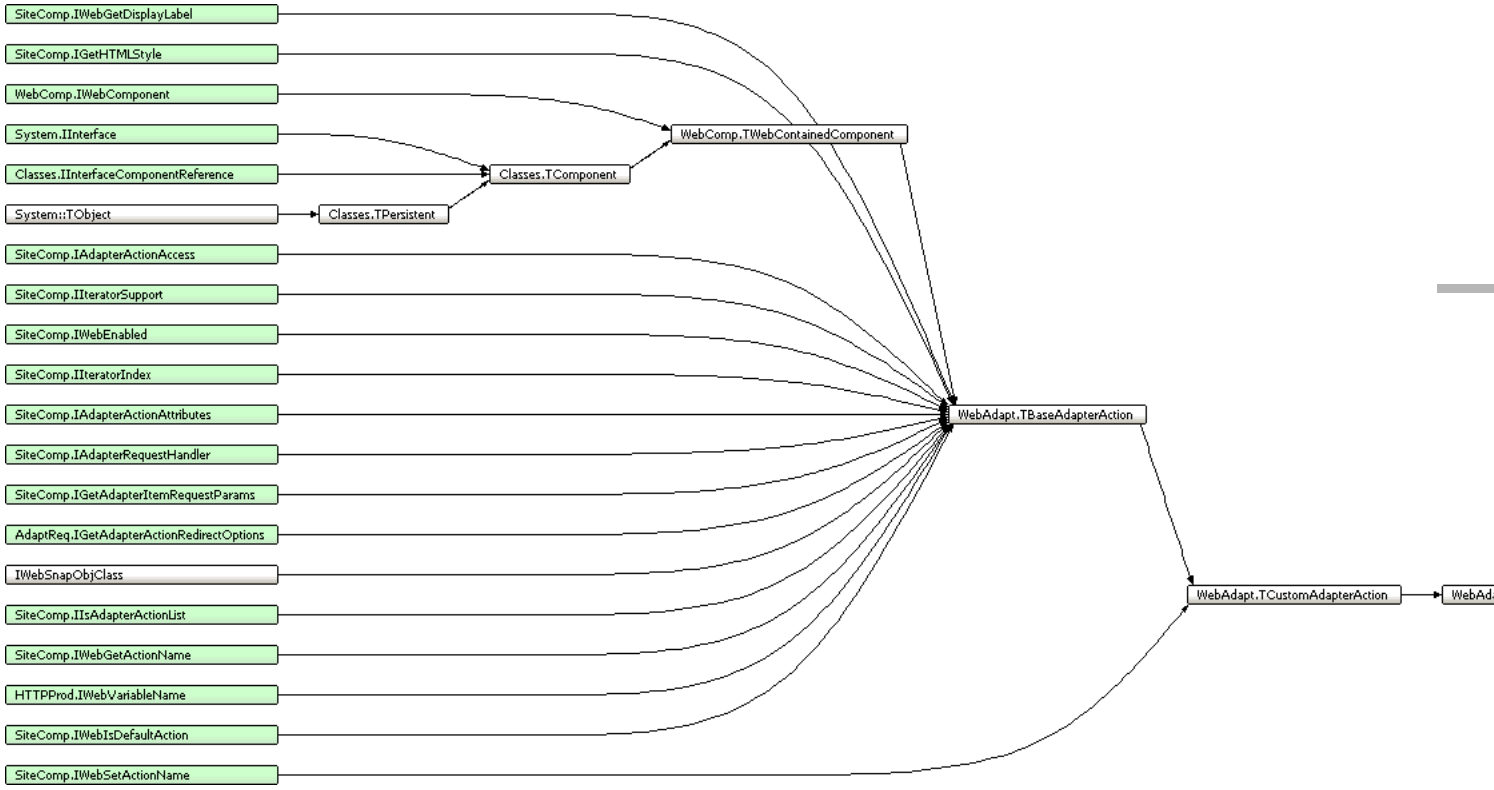

Pascal

```
TDataSetAdapterCancelRowAction = class(TCustomDataSetAdapterRowAction);
```

C++

```
class TDataSetAdapterCancelRowAction : public TCustomDataSetAdapterRowAction;
```

File

DBAdapt

Description

Add TDataSetAdapterBrowseRowAction (see page 7903) to a dataset adapter (TDataSetAdapter (see page 7872)) to support HTML forms that let users cancel the edits that they make and change to read-only viewing of database data. When this action is executed, it ignores any new field values and returns the parent dataset adapter to its default mode. It then positions the dataset on the record specified by the current request object.

Server-side script addresses this action as the 'Cancel' action of the parent adapter. By default, this action is visible (included in HTML forms) only when the parent adapter is in insert or edit mode. You can, however, cause this action to be included when the dataset is in other modes by changing the value of its ActionModes (see page 7926) property. If the action is visible when the dataset is in browse mode, the control for this action appears disabled.

This action is included by default when the adapter is used to build forms that represent a single database record.

See Also

TDataSetAdapterApplyRowAction (see page 7890)

TDataSetAdapterBrowseRowAction (see page 7903)

TDataSetAdapterDeleteRowAction (see page 7934)

TDataSetAdapterEditRowAction (see page 7947)

TDataSetAdapterFirstRowAction (see page 7972)

TDataSetAdapterLastRowAction (see page 7984)

TDataSetAdapterNextRowAction (see page 8030)

TDataSetAdapterPrevRowAction (see page 8042)

TDataSetAdapterRefreshRowAction (see page 8054)

TDataSetAdapterNewRowAction (see page 8016)

TDataSetAdapter (see page 7872)

TCustomDataSetAdapter (see page 7766)

TCustomDataSetAdapter (see page 7766)

41.1.13.1 TDataSetAdapterCancelRowAction Members

The following tables list the members exposed by TDataSetAdapterCancelRowAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 7819)	Returns the value of the Adapter (see page 7822) property.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.

	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7819)	Returns the value of the DataSet (see page 7822) property.
V	GetDefaultActionName (see page 7924)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
V	ImplExecuteActionRequest (see page 7925)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 7854)	Provides the underlying implementation for the GetAdapterItemRequestParams method.
V	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
V	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
V	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetVisible (see page 7819)	Provides the underlying implementation for the GetVisible method.
V	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.

V	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
V	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
V	ImplIsDefaultAction (see page 7925)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
V	ImplWebEnabled (see page 7925)	Provides the underlying implementation for the WebEnabled method.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7822)	Indicates the dataset adapter that uses this object to represent one of its actions.
R	Adapter (see page 24072)	Provides access to the parent adapter component.

R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 7822)	Occurs when server-side script executes this action.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 7823)	Occurs when the WebSnap application to determine whether a control or hyperlink for this action should be enabled.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.

	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionName (see page 7821)	Identifies the name of this action when using server-side script.
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 7822)	Indicates the dataset on which this action operates.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 7928)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 7928)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 7929)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 7929)	Called before the adapter action response is generated.
	OnGetParams (see page 7929)	Specifies name/value pairs needed to execute this adapter action.

Published Methods

	Name	Description
V	Create (see page 7924)	Creates and initializes an instance of TDataSetAdapterCancelRowAction (see page 7916).

Published Properties

	Name	Description
	ActionModes (see page 7926)	Indicates what dataset adapter modes this field supports.
	DisplayLabel (see page 7926)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 7927)	Defines the rights required to execute this action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RedirectOptions (see page 7927)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.13.2 TDataSetAdapterCancelRowAction.Create Constructor

Creates and initializes an instance of TDataSetAdapterCancelRowAction (see page 7916).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDataSetAdapterCancelRowAction(TComponent * AOwner);
```

Description

The TDataSetAdapterCancelRowAction (see page 7916) constructor allocates memory for the adapter action object and initializes its properties.

AOwner is the component that is responsible for starting the streaming process when a module containing the adapter action is loaded or saved. It becomes the value of the Owner property. Typically, AOwner is the Web module.

See Also

Owner (see page 2807)

41.1.13.3 TDataSetAdapterCancelRowAction Methods

The methods of the TDataSetAdapterCancelRowAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 7924)	Gets the default name used to reference this adapter action with server-side script.
	ImplExecuteActionRequest (see page 7925)	Executes the adapter action using parameters passed in the HTTP request.
	ImplIsDefaultAction (see page 7925)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplWebEnabled (see page 7925)	Provides the underlying implementation for the WebEnabled method.

41.1.13.3.1 TDataSetAdapterCancelRowAction.GetDefaultActionName Method

Gets the default name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultActionName();
```

Description

GetDefaultActionName returns the name of the adapter action component. Descendants may override this function.

See Also

ImplGetActionName (see page 24063)

41.1.13.3.2 TDataSetAdapterCancelRowAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest create a TStrings containing the name/value pairs in AActionRequest and calls the OnExecute event handler.

See Also

ImplExecuteActionRequest (see page 24063)

OnExecute (see page 24200)

41.1.13.3.3 TDataSetAdapterCancelRowAction.ImplIsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function ImplIsDefaultAction(ADisplay: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplIsDefaultAction(TObject * ADisplay);
```

Description

ImplIsDefaultAction is called by IsDefaultAction. ImplIsDefaultAction returns true. ImplIsDefaultAction may be overridden by a descendant class.

See Also

IsDefaultAction (see page 24070)

ImplIsDefaultAction (see page 24243)

41.1.13.3.4 TDataSetAdapterCancelRowAction.ImplWebEnabled Method

Provides the underlying implementation for the WebEnabled method.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

The WebEnabled method calls ImplWebEnabled to determine whether the control or hyperlink for this action should appear enabled in the HTML form. WebEnabled returns true if controls should be enabled, false otherwise.

ImplWebEnabled generates an OnGetEnabled event so that you can specify whether controls for this action are enabled. If there is no OnGetEnabled event handler, ImplWebEnabled returns true if the parent adapter's GetKeyFields method returns a set of fields that can locate records in the associated dataset.

See Also

WebEnabled (see page 24071)

OnGetEnabled (see page 7823)

GetKeyFields (see page 7786)

41.1.13.4 TDataSetAdapterCancelRowAction Properties

The properties of the TDataSetAdapterCancelRowAction class are listed here.

Published Properties

	Name	Description
	ActionModes (see page 7926)	Indicates what dataset adapter modes this field supports.
	DisplayLabel (see page 7926)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 7927)	Defines the rights required to execute this action.
	RedirectOptions (see page 7927)	Indicates whether to perform an HTTP redirect after successfully executing this action.

41.1.13.4.1 TDataSetAdapterCancelRowAction.ActionModes Property

Indicates what dataset adapter modes this field supports.

Pascal

```
property ActionModes: TDataSetAdapterModes;
```

C++

```
__property TDataSetAdapterModes ActionModes;
```

Description

ActionModes is a set of all the modes in which this adapter action can be used. If the parent dataset adapter is in a mode that is not included in this set, any forms generated using that adapter do not include a control for executing this action. Thus, for example, there is no sense in adding a control that applies edits to a form that presents a read-only view of the data. An apply action, then, should not include amBrowse in its action modes.

See Also

CanModify (see page 7154)

Adapter (see page 7752)

Mode (see page 7810)

GetDisplayStyle (see page 7739)

GetInputStyle (see page 7741)

GetViewMode (see page 7744)

41.1.13.4.2 TDataSetAdapterCancelRowAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

41.1.13.4.3 TDataSetAdapterCancelRowAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property AnsiString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 25625)

hasexecuteaccess (see page 24162)

41.1.13.4.4 TDataSetAdapterCancelRowAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect (see page 15201)

41.1.13.5 TDataSetAdapterCancelRowAction Events

The events of the TDataSetAdapterCancelRowAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 7928)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 7928)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 7929)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 7929)	Called before the adapter action response is generated.
	OnGetParams (see page 7929)	Specifies name/value pairs needed to execute this adapter action.

41.1.13.5.1 TDataSetAdapterCancelRowAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

41.1.13.5.2 TDataSetAdapterCancelRowAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.13.5.3 TDataSetAdapterCancelRowAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

41.1.13.5.4 TDataSetAdapterCancelRowAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.13.5.5 TDataSetAdapterCancelRowAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

41.1.14 TDataSetAdapterDataLink Class

<insert text here>

Class Hierarchy

Pascal

```
TDataSetAdapterDataLink = class(TDataLink);
```

C++

```
class TDataSetAdapterDataLink : public TDataLink;
```

File

DBAdapt

Description

<insert text here>

41.1.14.1 TDataSetAdapterDataLink Members

The following tables list the members exposed by TDataSetAdapterDataLink.

Protected Methods

	Name	Description
	ActiveChanged (see page 6880)	Responds to changes in the Active (see page 6888) property.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CheckBrowseMode (see page 6880)	Allows the data-aware object to respond just before an event that changes the dataset state.
	DataEvent (see page 7932)	Responds to various events that occur while working with data.
	DataSetChanged (see page 6881)	Responds to changes in the dataset.
	DataSetScrolled (see page 6882)	Allows a response to scrolling the representation of the dataset within the data-aware object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EditingChanged (see page 6883)	Responds to changes in the editing state of the data source.
	FocusControl (see page 6884)	Provides the interface for a method to force focus to the data-aware component associated with the TDataLink (see page 6876).
	GetActiveRecord (see page 6884)	This is GetActiveRecord, a member of class TDataLink.
	GetBOF (see page 6884)	This is GetBOF, a member of class TDataLink.
	GetBufferCount (see page 6884)	This is GetBufferCount, a member of class TDataLink.

	GetEOF (see page 6885)	This is GetEOF, a member of class TDataLink.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRecordCount (see page 6885)	This is GetRecordCount, a member of class TDataLink.
	LayoutChanged (see page 6885)	Responds to changes in the representation of the data by the data-aware object.
	MoveBy (see page 6885)	Repositions the dataset to another record relative to its active record.
	RecordChanged (see page 7933)	Responds to changes in the contents of the current record or field of the dataset.
	SetActiveRecord (see page 6886)	This is SetActiveRecord, a member of class TDataLink.
	SetBufferCount (see page 6887)	This is SetBufferCount, a member of class TDataLink.
	UpdateData (see page 6887)	Provides an interface for writing edits to a record in the dataset.

Protected Properties

	Name	Description
	VisualControl (see page 6894)	Indicates whether a visual control uses the datalink.

Public Events

	Name	Description
	OnDataSetStateChange (see page 7934)	This is OnDataSetStateChange, a member of class TDataSetAdapterDataLink.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 7932)	<insert text here>
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 6879)	Destroys an instance of TDataLink (see page 6876).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6882)	Attempts to drive the data source into an editing state if it is not already in one.
	ExecuteAction (see page 6883)	Invokes an action with the datasource as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	UpdateAction (see page 6887)	Updates an action to reflect the current state of the TDataLink (see page 6876).
	UpdateRecord (see page 6888)	Responds to notifications that edits should be posted to the dataset.

Public Properties

	Name	Description
R	Active (see page 6888)	Specifies whether the dataset that TDataLink (see page 6876) manages is active.
	ActiveRecord (see page 6889)	Specifies the index of the current record within the internal record buffer maintained by the dataset for the Owner of the TDataLink (see page 6876) object.
R	BOF (see page 6889)	Indicates whether the first record in a dataset is active.
	BufferCount (see page 6890)	Specifies the number of records cached by the dataset for the Owner of the TDataLink (see page 6876) object.
R	DataSet (see page 6890)	Specifies the TDataSet (see page 6894) object that represents the data that this TDataLink (see page 6876) is tracking.
	DataSource (see page 6891)	Specifies the TDataSource (see page 7070) object that the Owner of the TDataLink (see page 6876) uses to connect to the dataset.
	DataSourceFixed (see page 6891)	Indicates whether the DataSource (see page 6891) property can be set.
R	Editing (see page 6892)	Indicates whether the DataSource (see page 6891) is in edit mode.
R	Eof (see page 6892)	Indicates whether the last record in a dataset is active.
	ReadOnly (see page 6893)	Indicates whether the dataset for this TDataLink (see page 6876) can be modified.
R	RecordCount (see page 6893)	Indicates the number of records in the internal record buffer maintained by the dataset.

41.1.14.2 TDataSetAdapterDataLink.Create Constructor

<insert text here>

Pascal

```
constructor Create(AAdapter: TComponent);
```

C++

```
__fastcall TDataSetAdapterDataLink(TComponent * AAdapter);
```

Description

<insert text here>

41.1.14.3 TDataSetAdapterDataLink Methods

The methods of the TDataSetAdapterDataLink class are listed here.

Protected Methods

	Name	Description
V	DataEvent (see page 7932)	Responds to various events that occur while working with data.
V	RecordChanged (see page 7933)	Responds to changes in the contents of the current record or field of the dataset.

41.1.14.3.1 TDataSetAdapterDataLink.DataEvent Method

Responds to various events that occur while working with data.

Pascal

```
procedure DataEvent(Event: TDataEvent; Info: Longint); override;
```

C++

```
virtual __fastcall DataEvent(TDataEvent Event, Longint Info);
```

Description

DataEvent is called automatically when various data events occur. Event indicates the event that just occurred. Info provides additional context information for some events.

DataEvent dispatches these events by calling the appropriate methods, as indicated in the following table.

Data event	Info	Method called
deFieldChange | deRecordChange	The field that changed	RecordChanged
deDataSetChange	Current index into the record buffer	DataSetChanged
deDataSetScroll	Current index into the record buffer	DataSetScrolled
deLayoutChange	Current index into the record buffer	LayoutChanged
deUpdateRecord	UpdateRecord	
deUpdateState	EditingChanged or ActiveChanged	
deCheckBrowseMode	CheckBrowseMode	
deFocusControl	FocusControl	

See Also

DataSetScrolled (see page 6882)

LayoutChanged (see page 6885)

RecordChanged (see page 6886)

DataSetChanged (see page 6881)

UpdateRecord (see page 6888)

CheckBrowseMode (see page 6880)

FocusControl (see page 6884)

ActiveChanged (see page 6880)

EditingChanged (see page 6883)

41.1.14.3.2 TDataSetAdapterDataLink.RecordChanged Method

Responds to changes in the contents of the current record or field of the dataset.

Pascal

```
procedure RecordChanged(Field: TField); override;
```

C++

```
virtual __fastcall RecordChanged(TField Field);
```

Description

The RecordChanged method defined by TDataLink merely provides an interface for a method that can respond to changes to the contents of the current record. RecordChanged is called after changes have been posted to the current record in the dataset, and from DataSetChanged.

The Field parameter indicates which field of the current record has changed in value. If Field is nil (Delphi) or NULL (C++), any

number of fields within the current record may have changed.

Derived objects that do not need to respond to such changes can allow the inherited method to ignore them.

See Also

ActiveRecord (see page 6889)

DataSetChanged (see page 6881)

OnDataChange (see page 7080)

OnChange (see page 7181)

41.1.14.4 TDataSetAdapterDataLink Events

The events of the TDataSetAdapterDataLink class are listed here.

Public Events

	Name	Description
	OnDataSetStateChange (see page 7934)	This is OnDataSetStateChange, a member of class TDataSetAdapterDataLink.

41.1.14.4.1 TDataSetAdapterDataLink.OnDataSetStateChange Event

Pascal

```
property OnDataSetStateChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDataSetStateChange;
```

Description

This is OnDataSetStateChange, a member of class TDataSetAdapterDataLink.

41.1.15 TDataSetAdapterDeleteRowAction Class

TDataSetAdapterDeleteRowAction deletes the current row from a dataset, as specified in an HTTP request.

Class Hierarchy

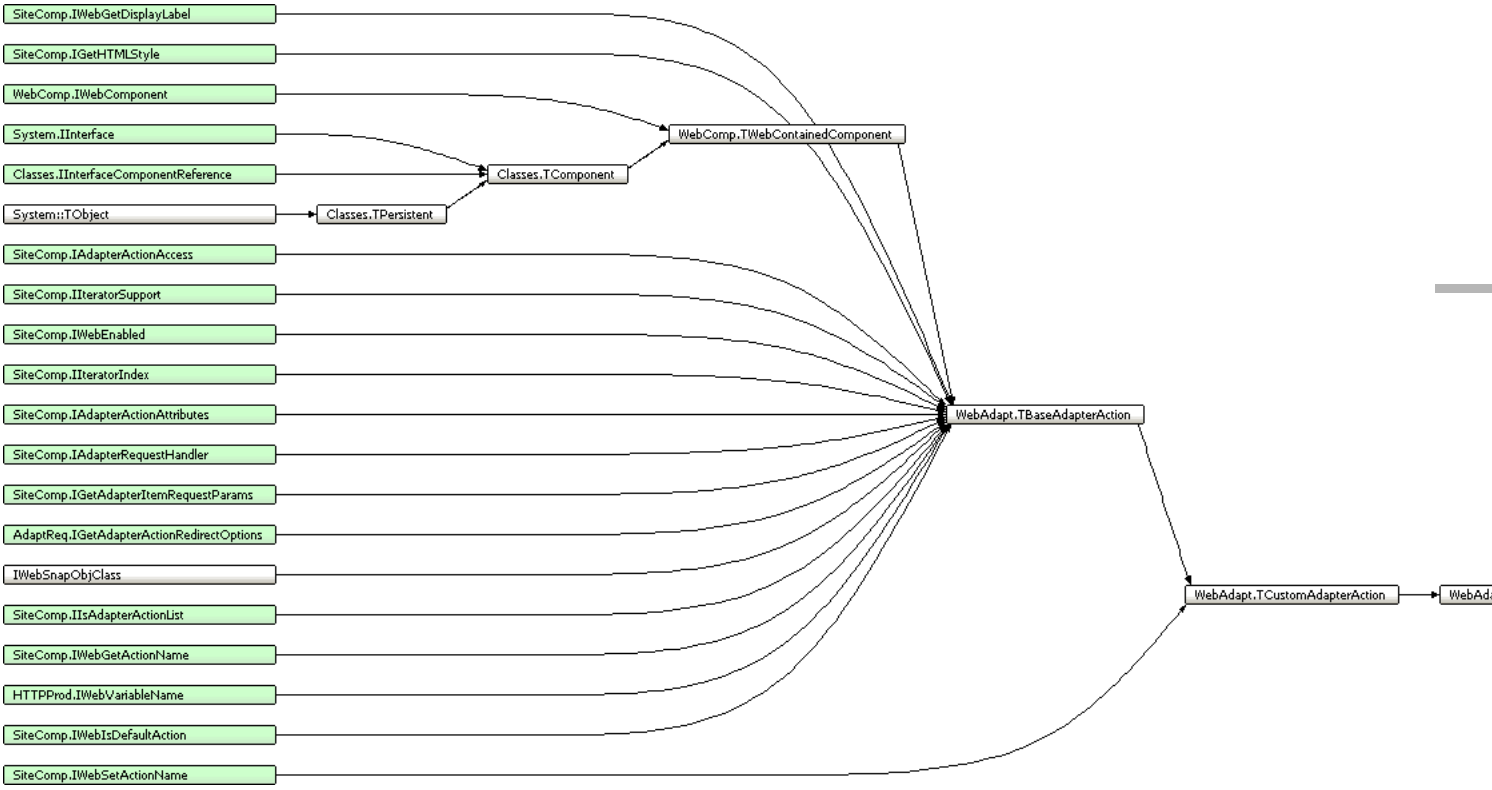

Pascal

```
TDataSetAdapterDeleteRowAction = class(TCustomDataSetAdapterRowAction);
```

C++

```
class TDataSetAdapterDeleteRowAction : public TCustomDataSetAdapterRowAction;
```

File

DBAdapt

Description

Add TDataSetAdapterDeleteRowAction to a dataset adapter (TDataSetAdapter (see page 7872)) to support HTML forms that let users delete database records. When this action is executed, it positions the dataset on the record specified in the current request and deletes it (or fails to post edits if the dataset adapter is in insert mode).

Server-side script addresses this action as the 'DeleteRow' action of the parent adapter. By default, this action is visible (included in HTML forms) only when the parent adapter is in insert or edit mode. You can, however, cause this action to be included when the dataset is in other modes by changing the value of its ActionModes (see page 7821) property.

See Also

TDataSetAdapterApplyRowAction (see page 7890)

TDataSetAdapterBrowseRowAction (see page 7903)

TDataSetAdapterCancelRowAction (see page 7916)

TDataSetAdapterEditRowAction (see page 7947)

TDataSetAdapterFirstRowAction (see page 7972)

TDataSetAdapterLastRowAction (see page 7984)

TDataSetAdapterNextRowAction (see page 8030)

TDataSetAdapterPrevRowAction (see page 8042)

TDataSetAdapterRefreshRowAction (see page 8054)

TDataSetAdapterNewRowAction (see page 8016)

TDataSetAdapter (see page 7872)

TCustomDataSetAdapter (see page 7766)

TDataSet (see page 6894)

TDataSEtAdapterRowAction

41.1.15.1 TDataSetAdapterDeleteRowAction Members

The following tables list the members exposed by TDataSetAdapterDeleteRowAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 7819)	Returns the value of the Adapter (see page 7822) property.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7819)	Returns the value of the DataSet (see page 7822) property.

	GetDefaultActionName (see page 7942)	Gets the default name used to reference this adapter action with server-side script.
	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 7942)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 7854)	Provides the underlying implementation for the GetAdapterItemRequestParams method.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 7819)	Provides the underlying implementation for the GetVisible method.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.

	ImplIsDefaultAction (see page 7943)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 7820)	Provides the underlying implementation for the WebEnabled method.
	ImplWebEnabled (see page 7855)	Provides the underlying implementation for the WebEnabled method.
	ImplWebEnabled (see page 24069)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 24197)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7822)	Indicates the dataset adapter that uses this object to represent one of its actions.
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 7822)	Occurs when server-side script executes this action.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 7823)	Occurs when the WebSnap application to determine whether a control or hyperlink for this action should be enabled.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 7818)	Creates and initializes an instance of TCustomDataSetAdapterAction (see page 7811).
V	Create (see page 7853)	Creates and initializes an instance of TCustomDataSetAdapterRowAction (see page 7846).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionModes (see page 7821)	Indicates what dataset adapter modes this field supports.
	ActionName (see page 7821)	Identifies the name of this action when using server-side script.
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 7822)	Indicates the dataset on which this action operates.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 7945)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 7945)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 7945)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 7946)	Called before the adapter action response is generated.
	OnGetParams (see page 7946)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 7943)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 7944)	Defines the rights required to execute this action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RedirectOptions (see page 7944)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.15.2 TDataSetAdapterDeleteRowAction Methods

The methods of the TDataSetAdapterDeleteRowAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 7942)	Gets the default name used to reference this adapter action with server-side script.
	ImplExecuteActionRequest (see page 7942)	Executes the adapter action using parameters passed in the HTTP request.
	ImplIsDefaultAction (see page 7943)	Indicates whether this adapter action should be added, by default, to the parent adapter.

41.1.15.2.1 TDataSetAdapterDeleteRowAction.GetDefaultActionName Method

Gets the default name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultActionName();
```

Description

GetDefaultActionName returns the name of the adapter action component. Descendants may override this function.

See Also

ImplGetActionName (see page 24063)

41.1.15.2.2 TDataSetAdapterDeleteRowAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest create a TStrings containing the name/value pairs in AActionRequest and calls the OnExecute event handler.

See Also

ImplExecuteActionRequest (see page 24063)

OnExecute (see page 24200)

41.1.15.2.3 TDataSetAdapterDeleteRowAction.ImplIsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function ImplIsDefaultAction(ADisplay: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplIsDefaultAction(TObject * ADisplay);
```

Description

ImplIsDefaultAction is called by IsDefaultAction. ImplIsDefaultAction returns true. ImplIsDefaultAction may be overridden by a descendant class.

See Also

IsDefaultAction (see page 24070)

ImplIsDefaultAction (see page 24243)

41.1.15.3 TDataSetAdapterDeleteRowAction Properties

The properties of the TDataSetAdapterDeleteRowAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 7943)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 7944)	Defines the rights required to execute this action.
	RedirectOptions (see page 7944)	Indicates whether to perform an HTTP redirect after successfully executing this action.

41.1.15.3.1 TDataSetAdapterDeleteRowAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

41.1.15.3.2 TDataSetAdapterDeleteRowAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property AnsiString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 25625)

hasexecuteaccess (see page 24162)

41.1.15.3.3 TDataSetAdapterDeleteRowAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect (see page 15201)

41.1.15.4 TDataSetAdapterDeleteRowAction Events

The events of the TDataSetAdapterDeleteRowAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 7945)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 7945)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 7945)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 7946)	Called before the adapter action response is generated.
	OnGetParams (see page 7946)	Specifies name/value pairs needed to execute this adapter action.

41.1.15.4.1 TDataSetAdapterDeleteRowAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

41.1.15.4.2 TDataSetAdapterDeleteRowAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.15.4.3 TDataSetAdapterDeleteRowAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

41.1.15.4.4 TDataSetAdapterDeleteRowAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.15.4.5 TDataSetAdapterDeleteRowAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

41.1.16 TDataSetAdapterEditRowAction Class

TDataSetAdapterDeleteRowAction (see page 7934) puts the parent dataset adapter into edit mode.

Class Hierarchy

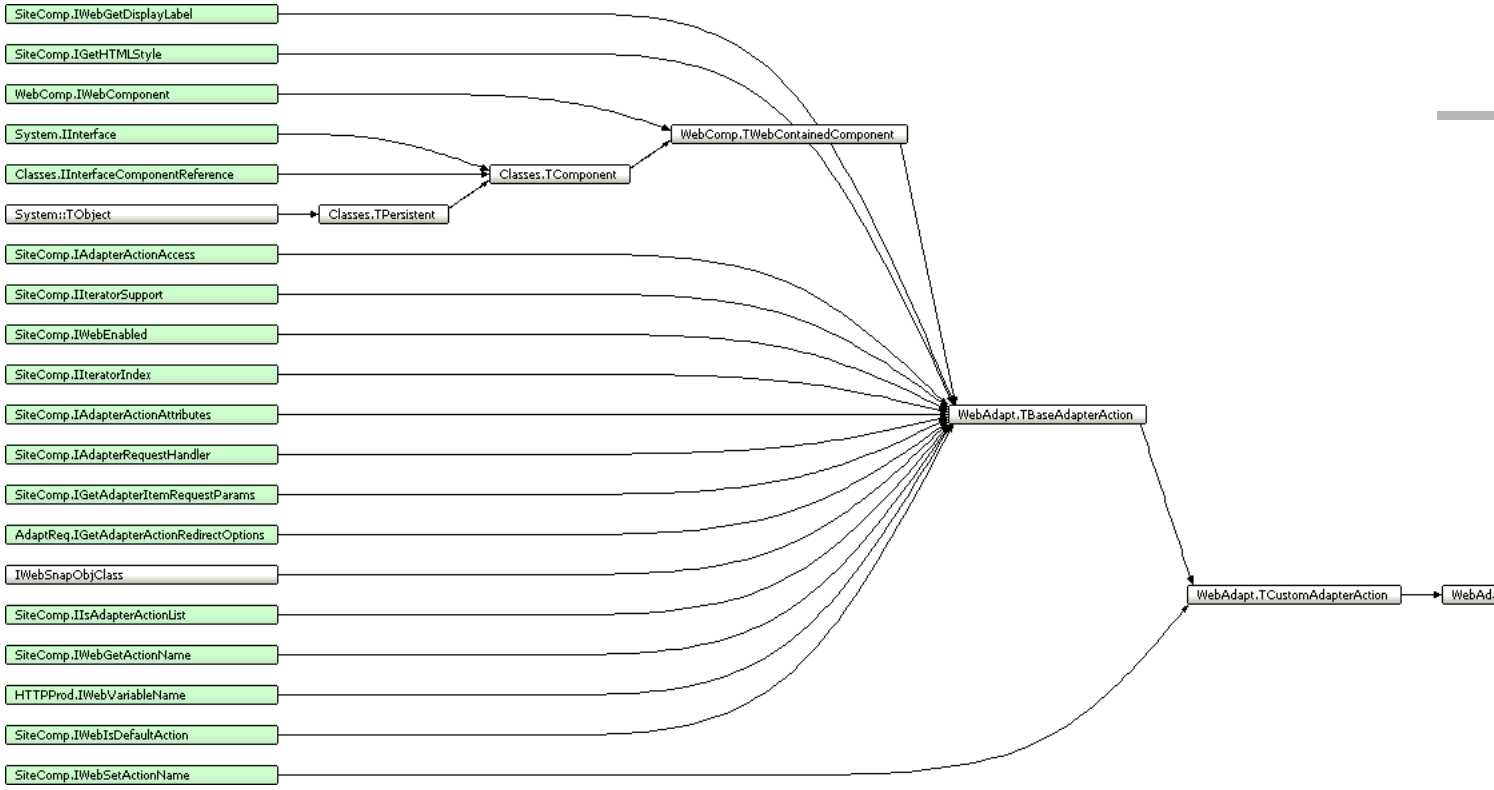

Pascal

```
TDataSetAdapterEditRowAction = class(TCustomDataSetAdapterRowAction);
```

C++

```
class TDataSetAdapterEditRowAction : public TCustomDataSetAdapterRowAction;
```

File

DBAdapt

Description

Add TDataSetAdapterEditRowAction to a dataset adapter (TDataSetAdapter (see page 7872)) to support HTML forms that let users start a new edit to the current database record. When this action is executed, it positions the dataset on the record specified in the current request, applies any edits contained in the current request, and puts the parent dataset adapter in edit mode.

Server-side script addresses this action as the 'EditRow' action of the parent adapter. By default, this action is visible (included in HTML forms) regardless of the mode of the parent adapter. This action is included by default when the adapter is used to build forms that represent multiple database records.

See Also

TDataSetAdapterApplyRowAction (see page 7890)

TDataSetAdapterBrowseRowAction (see page 7903)

TDataSetAdapterCancelRowAction (see page 7916)

TDataSetAdapterDeleteRowAction (see page 7934)

TDataSetAdapterFirstRowAction (see page 7972)

TDataSetAdapterLastRowAction (see page 7984)

TDataSetAdapterNextRowAction (see page 8030)

TDataSetAdapterPrevRowAction (see page 8042)

TDataSetAdapterRefreshRowAction (see page 8054)

TDataSetAdapterNewRowAction (see page 8016)

TDataSetAdapter (see page 7872)

TCustomDataSetAdapter (see page 7766)

TCustomDataSetAdapter (see page 7766)

TDataSetAdapterRowAction (see page 8066)

41.1.16.1 TDataSetAdapterEditRowAction Members

The following tables list the members exposed by TDataSetAdapterEditRowAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 7819)	Returns the value of the Adapter (see page 7822) property.

	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7819)	Returns the value of the DataSet (see page 7822) property.
	GetDefaultActionName (see page 7954)	Gets the default name used to reference this adapter action with server-side script.
	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 7955)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 7854)	Provides the underlying implementation for the GetAdapterItemRequestParams method.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.

	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 7819)	Provides the underlying implementation for the GetVisible method.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 7955)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 7955)	Provides the underlying implementation for the WebEnabled method.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Adapter (see page 7822)	Indicates the dataset adapter that uses this object to represent one of its actions.
	Adapter (see page 24072)	Provides access to the parent adapter component.
	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 7822)	Occurs when server-side script executes this action.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 7823)	Occurs when the WebSnap application to determine whether a control or hyperlink for this action should be enabled.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.

	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionName (see page 7821)	Identifies the name of this action when using server-side script.
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 7822)	Indicates the dataset on which this action operates.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RedirectOptions (see page 24074)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 7958)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 7958)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 7958)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 7959)	Called before the adapter action response is generated.
	OnGetParams (see page 7959)	Specifies name/value pairs needed to execute this adapter action.

Published Methods

	Name	Description
V	Create (see page 7954)	Creates and initializes an instance of TDataSetAdapterEditRowAction (see page 7947).

Published Properties

	Name	Description
	ActionModes (see page 7956)	Indicates what dataset adapter modes this field supports.
	DisplayLabel (see page 7957)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 7957)	Defines the rights required to execute this action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.16.2 TDataSetAdapterEditRowAction.Create Constructor

Creates and initializes an instance of TDataSetAdapterEditRowAction (see page 7947).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDataSetAdapterEditRowAction(TComponent * AOwner);
```

Description

The TDataSetAdapterEditRowAction (see page 7947) constructor allocates memory for the adapter action object and initializes its properties.

AOwner is the component that is responsible for starting the streaming process when a module containing the adapter action is loaded or saved. It becomes the value of the Owner property. Typically, AOwner is the Web module.

See Also

Owner (see page 2807)

41.1.16.3 TDataSetAdapterEditRowAction Methods

The methods of the TDataSetAdapterEditRowAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 7954)	Gets the default name used to reference this adapter action with server-side script.
	ImplExecuteActionRequest (see page 7955)	Executes the adapter action using parameters passed in the HTTP request.
	ImplIsDefaultAction (see page 7955)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplWebEnabled (see page 7955)	Provides the underlying implementation for the WebEnabled method.

41.1.16.3.1 TDataSetAdapterEditRowAction.GetDefaultActionName Method

Gets the default name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultActionName();
```

Description

GetDefaultActionName returns the name of the adapter action component. Descendants may override this function.

See Also

ImplGetActionName (see page 24063)

41.1.16.3.2 TDataSetAdapterEditRowAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest create a TStrings containing the name/value pairs in AActionRequest and calls the OnExecute event handler.

See Also

ImplExecuteActionRequest (see page 24063)

OnExecute (see page 24200)

41.1.16.3.3 TDataSetAdapterEditRowAction.ImplIsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function ImplIsDefaultAction(ADisplay: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplIsDefaultAction(TObject * ADisplay);
```

Description

ImplIsDefaultAction is called by IsDefaultAction. ImplIsDefaultAction returns true. ImplIsDefaultAction may be overridden by a descendant class.

See Also

IsDefaultAction (see page 24070)

ImplIsDefaultAction (see page 24243)

41.1.16.3.4 TDataSetAdapterEditRowAction.ImplWebEnabled Method

Provides the underlying implementation for the WebEnabled method.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

The WebEnabled method calls ImplWebEnabled to determine whether the control or hyperlink for this action should appear enabled in the HTML form. WebEnabled returns true if controls should be enabled, false otherwise.

ImplWebEnabled generates an OnGetEnabled event so that you can specify whether controls for this action are enabled. If there is no OnGetEnabled event handler, ImplWebEnabled returns true if the parent adapter's GetKeyFields method returns a set of fields that can locate records in the associated dataset.

See Also

WebEnabled (see page 24071)

OnGetEnabled (see page 7823)

GetKeyFields (see page 7786)

41.1.16.4 TDataSetAdapterEditRowAction Properties

The properties of the TDataSetAdapterEditRowAction class are listed here.

Published Properties

	Name	Description
	ActionModes (see page 7956)	Indicates what dataset adapter modes this field supports.
	DisplayLabel (see page 7957)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 7957)	Defines the rights required to execute this action.

41.1.16.4.1 TDataSetAdapterEditRowAction.ActionModes Property

Indicates what dataset adapter modes this field supports.

Pascal

```
property ActionModes: TDataSetAdapterModes;
```

C++

```
__property TDataSetAdapterModes ActionModes;
```

Description

ActionModes is a set of all the modes in which this adapter action can be used. If the parent dataset adapter is in a mode that is not included in this set, any forms generated using that adapter do not include a control for executing this action. Thus, for example, there is no sense in adding a control that applies edits to a form that presents a read-only view of the data. An apply action, then, should not include amBrowse in its action modes.

See Also

CanModify (see page 7154)

Adapter (see page 7752)

Mode (see page 7810)

GetDisplayStyle (see page 7739)

GetInputStyle (see page 7741)

GetViewMode (see page 7744)

41.1.16.4.2 TDataSetAdapterEditRowAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

41.1.16.4.3 TDataSetAdapterEditRowAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property AnsiString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 25625)

hasexecuteaccess (see page 24162)

41.1.16.5 TDataSetAdapterEditRowAction Events

The events of the TDataSetAdapterEditRowAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 7958)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 7958)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 7958)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 7959)	Called before the adapter action response is generated.
	OnGetParams (see page 7959)	Specifies name/value pairs needed to execute this adapter action.

41.1.16.5.1 TDataSetAdapterEditRowAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

41.1.16.5.2 TDataSetAdapterEditRowAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.16.5.3 TDataSetAdapterEditRowAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

41.1.16.5.4 TDataSetAdapterEditRowAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.16.5.5 TDataSetAdapterEditRowAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

41.1.17 TDataSetAdapterField Class

TDataSetAdapterField is the an adapter field object that represents a database field whose value can be displayed as a simple text element.

Class Hierarchy

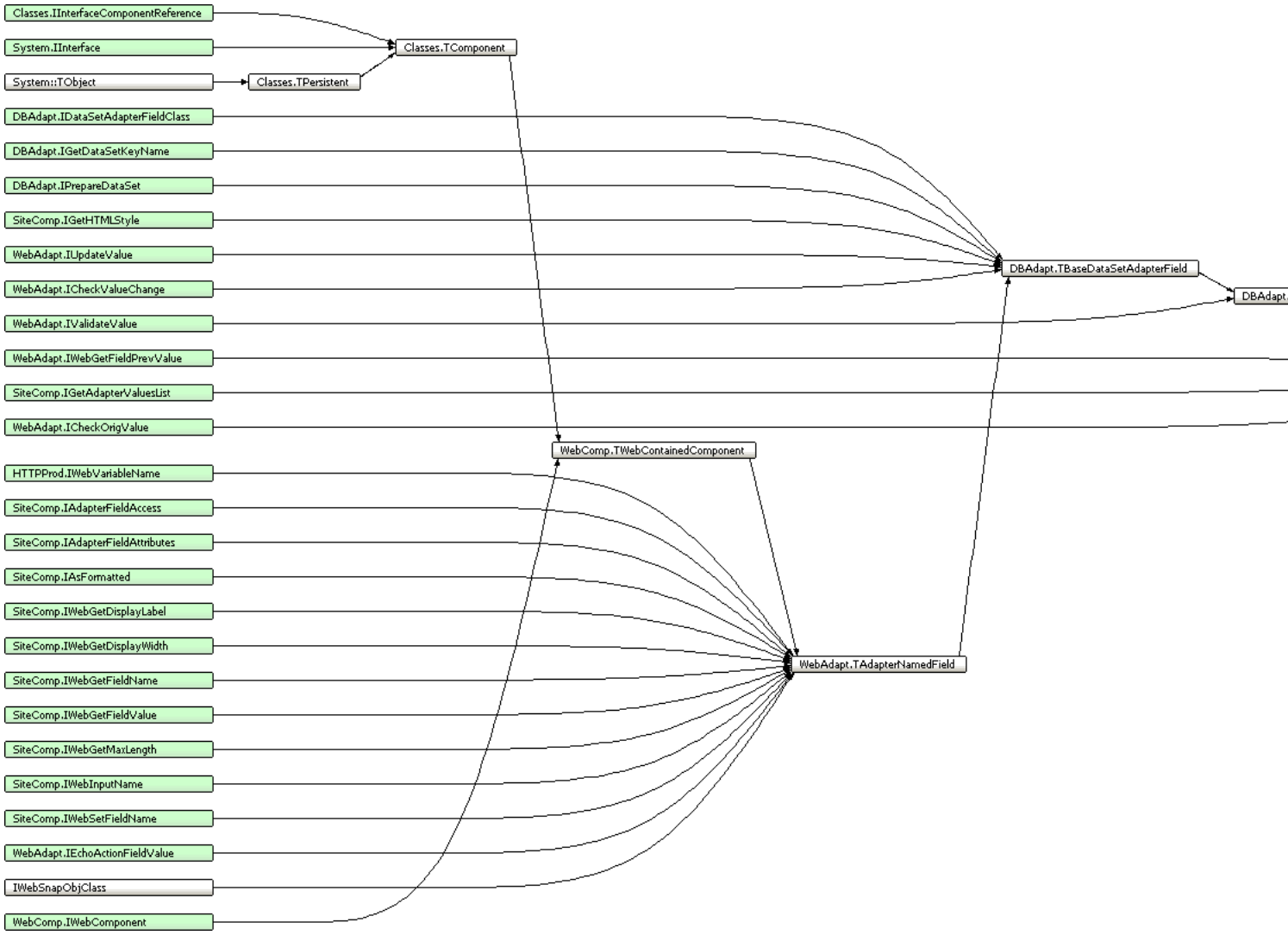

Pascal

```
TDataSetAdapterField = class(TCustomDataSetAdapterField);
```

C++

```
class TDataSetAdapterField : public TCustomDataSetAdapterField;
```

File

DBAdapt

Description

Add TDataSetAdapterField as a child of a TDataSetAdapter (see page 7872) component to represent a simple field in the dataset associated with the dataset adapter. TDataSetAdapterField represents simple field types that can appear in a single-line edit control. TDataSetAdapterField represents a single field from a database table, and enables server-side script to refer to or update that field's value.

TDataSetAdapterField supports several properties that can be used in server-side script. These include:

ValuesList (see page 7970), which is an object that lists the possible values for this field.

Value, which is the current value of the field.

DisplayText, which represents the current value of the field as a text string.

DisplayStyle and InputStyle, which indicate the type of control that should represent the field value in HTML forms.

DisplayLabel, which provides a text label that can be used to label HTML controls that represent the field.

Required, which indicates whether the field requires a value.

Visible, which indicates whether HTML forms should include a control for this field.

See Also

TDataSetAdapterMemoField (see page 7996)

TDataSetAdapterImageField (see page 8138)

TDataSetAdapter (see page 7872)

TField (see page 7115)

TAdapterDisplayField (see page 25109)

TAdapterDisplayColumn (see page 25099)

TAdapterEditColumn (see page 25132)

41.1.17.1 TDataSetAdapterField Members

The following tables list the members exposed by TDataSetAdapterField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckOrigValue (see page 7736)	Ensures that the current value of this field is the same as the original value.
	CheckValueChange (see page 7737)	Indicates whether the current value of this field is the same as the original value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FullyQualifyInputName (see page 7738)	Indicates whether references to this field in HTML forms must be qualified with the name of the parent adapter.
	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 7738)	Returns the value of the Adapter (see page 7752) property.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 7831)	Returns the value of the ValuesList (see page 7837) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7738)	Returns the dataset associated with this field's parent adapter.
	GetDataSetFieldValue (see page 7739)	Returns the value of the associated dataset field.
	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 7739)	Returns a string value that suggests an HTML display style for this adapter field.
	GetDisplayStyleType (see page 7740)	Returns the HTML display style type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 7741)	Returns a string value that suggests an HTML editing style for this adapter field.
	GetInputStyleType (see page 7742)	Returns the style of the HTML control to use for editing this adapter field.
	GetInputStyleType (see page 7831)	Returns the default style of the HTML control to use for editing this adapter field.
	GetKeyName (see page 7742)	Returns the name of the associated dataset field if it appears in the index (key) that the parent dataset adapter uses.
	GetKeyParams (see page 7743)	Returns a list of fields that the parent dataset adapter uses to locate records, along with their current values in that adapter's dataset.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetNeedPrevValue (see page 7832)	Indicates whether an HTML form should include a hidden field that stores this field's value so that generated requests include the previous field value.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPrevValue (see page 7832)	Returns the value of this field, as it should appear in a hidden field in an HTML form.
	GetReadOnly (see page 7743)	Indicates whether the adapter field represents a read-only database field.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 7744)	Returns a string value that indicates what function a control that represents this field should perform.
	GetViewModeType (see page 7744)	Returns the view mode for the function a control that represents this field should perform.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 7745)	Provides the underlying implementation for the AsFormatted method.

	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplCheckOrigValue (see page 7745)	Provides the underlying implementation for the CheckOrigValue (see page 7736) method.
	ImplCheckOrigValue (see page 7833)	Provides the underlying implementation for the CheckOrigValue (see page 7736) method.
	ImplCheckValueChange (see page 7746)	Provides the underlying implementation for the CheckValueChange (see page 7737) method.
	ImplCheckValueChange (see page 7833)	Provides the underlying implementation for the CheckValueChange (see page 7737) method.
	ImplGetDisplayLabel (see page 7747)	Provides the underlying implementation for the GetDisplayLabel method.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayWidth (see page 7747)	Provides the underlying implementation for the GetDisplayWidth method.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetEchoActionFieldValue (see page 7763)	Provides the underlying implementation for the GetEchoActionFieldValue method.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetNeedPrevValue (see page 7834)	Provides the underlying implementation for the GetNeedPrevValue (see page 7832) method.
	ImplGetPrevValue (see page 7835)	Provides the underlying implementation of the GetPrevValue (see page 7832) method.
	ImplGetRequired (see page 7747)	Provides the underlying implementation for the GetRequired method.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 7748)	Provides the underlying implementation for the GetValue method.
	ImplGetValue (see page 7835)	Provides the underlying implementation for the GetValue method.
	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 7748)	Provides the underlying implementation for the GetVisible method.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 7763)	Provides the underlying implementation for the SetEchoActionFieldValue method.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 7749)	Provides the underlying implementation for the SetFieldName method.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	ImplUpdateValue (see page 7749)	Provides the underlying implementation for the UpdateValue (see page 7751) method.
	ImplUpdateValue (see page 7836)	Provides the underlying implementation for the UpdateValue (see page 7751) method.
	ImplValidateValue (see page 7764)	Provides the underlying implementation for the ValidateValue (see page 7765) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7750)	Forwards notification messages to all owned components.
	Notification (see page 7836)	Responds when components are added to or deleted from the WebSnap application.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	PrepareDataSet (see page 7750)	Responds when the parent dataset adapter prepares its dataset.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.

V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSetField (see page 7751)	Sets the value of the DataSetField (see page 7752) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	TestValueChange (see page 7764)	Checks whether a new value matches the current value of this field.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 7751)	Updates the adapter field when the user assigns a new value.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateValue (see page 7765)	Validates a new value for this field that appears in the current request.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7752)	Indicates the dataset adapter that uses this object to represent one of its fields.
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckOrigValue (see page 24569)	This is CheckOrigValue, a member of class ICheckOrigValue.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 7735)	Creates and initializes an instance of TBaseDataSetAdapterField (see page 7729).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 7737)	Returns the dataset field whose value this adapter field represents.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterValuesList (see page 19695)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetKeyName (see page 8105)	This is GetKeyName, a member of class IGetDataSetKeyName.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNeedPrevValue (see page 24587)	This is GetNeedPrevValue, a member of class IWebGetFieldPrevValue.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetPrevValue (see page 24587)	This is GetPrevValue, a member of class IWebGetFieldPrevValue.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.

	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PrepareDataSet (see page 8109)	This is PrepareDataSet, a member of class IPrepareDataSet.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.

R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
R	KeyName (see page 8105)	This is KeyName, a member of class IGetDataSetKeyName.
R	NeedPrevValue (see page 24587)	This is NeedPrevValue, a member of class IWebGetFieldPrevValue.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	PrevValue (see page 24587)	This is PrevValue, a member of class IWebGetFieldPrevValue.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetDisplayText (see page 7971)	Occurs when the adapter field needs to supply its formatted value as a string that can appear in HTML forms.
	OnGetValue (see page 7971)	Occurs when the adapter field needs to supply its value as a Variant.
	OnValidateValue (see page 7972)	Occurs when the parent adapter wants to validate the new value for this field.

Published Properties

	Name	Description
	DataSetField (see page 7967)	Specifies the name of the dataset field that this adapter field represents.
	FieldFlags (see page 7968)	Indicates various attributes of the adapter field.
	FieldModes (see page 7968)	Indicates what dataset adapter modes this field supports.
	ModifyAccess (see page 7969)	Identifies the rights required to update this field.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OnUpdateValue (see page 7969)	Occurs when the WebSnap application updates this adapter field's value based on in incoming request.
	Tag (see page 2808)	Stores an integer value as part of a component.
	ValuesList (see page 7970)	Specifies the interface of an object that lists the possible values for this field.
	ViewAccess (see page 7970)	Identifies the rights required to view this field.

41.1.17.2 TDataSetAdapterField Properties

The properties of the TDataSetAdapterField class are listed here.

Published Properties

	Name	Description
	DataSetField (see page 7967)	Specifies the name of the dataset field that this adapter field represents.
	FieldFlags (see page 7968)	Indicates various attributes of the adapter field.
	FieldModes (see page 7968)	Indicates what dataset adapter modes this field supports.
	ModifyAccess (see page 7969)	Identifies the rights required to update this field.
	OnUpdateValue (see page 7969)	Occurs when the WebSnap application updates this adapter field's value based on in incoming request.
	ValuesList (see page 7970)	Specifies the interface of an object that lists the possible values for this field.
	ViewAccess (see page 7970)	Identifies the rights required to view this field.

41.1.17.2.1 TDataSetAdapterField.DataSetField Property

Specifies the name of the dataset field that this adapter field represents.

Pascal

```
property DataSetField: string;
```

C++

```
__property AnsiString DataSetField;
```

Description

DataSetField is the FieldName of the TField object whose value this adapter field represents. DataSetField can be the name of a simple field in a database table, the name of a subfield of an object field that has been qualified by the parent field's name, or the name of an aggregated field.

DataSetField is the same as the name of this adapter field in server-side script, except that the name of the adapter field omits any illegal characters (the only legal characters are the letters between a and z (or A and Z), digits, and the underline character).

If DataSetField is an empty string, this adapter field does not have a value, and the FindField method returns nil (Delphi) or NULL (C++).

See Also

FieldName (see page 7164)

FindField (see page 7737)

41.1.17.2.2 TDataSetAdapterField.FieldFlags Property

Indicates various attributes of the adapter field.

Pascal

```
property FieldFlags: TDataSetAdapterFieldFlags;
```

C++

```
__property TDataSetAdapterFieldFlags FieldFlags;
```

Description

FieldFlags indicates special uses of the value of this adapter field. This is a set that can indicate whether the value corresponds to a key field in the database (used to locate database records) and whether HTML forms store the original value of the field as a hidden field so that the application can detect changes to the field value.

See Also

IsIndexField (see page 7168)

DataSetField (see page 7752)

GetKeyName (see page 7742)

GetKeyFields (see page 7786)

41.1.17.2.3 TDataSetAdapterField.FieldModes Property

Indicates what dataset adapter modes this field supports.

Pascal

```
property FieldModes: TDataSetAdapterModes;
```

C++

```
__property TDataSetAdapterModes FieldModes;
```

Description

FieldModes is a set of all the modes in which this adapter field can be used. If the parent dataset adapter is in a mode that is not

included in this set, any forms generated using that adapter do not include a control to represent this field. Thus, for example, to make a field read-only, you can exclude amInsert and amEdit from FieldModes. Then, the field appears when the user is viewing or querying the dataset, but not when the user is editing or inserting a record.

See Also

CanModify (see page 7154)

Adapter (see page 7752)

Mode (see page 7810)

GetDisplayStyle (see page 7739)

GetInputStyle (see page 7741)

GetViewMode (see page 7744)

41.1.17.2.4 TDataSetAdapterField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property AnsiString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

hasmodifyaccess (see page 24162)

41.1.17.2.5 TDataSetAdapterField.OnUpdateValue Property

Occurs when the WebSnap application updates this adapter field's value based on in incoming request.

Pascal

```
property OnUpdateValue: TDatasetAdapterFieldValueEvent;
```

C++

```
__property TDatasetAdapterFieldValueEvent OnUpdateValue;
```

Description

Write an OnUpdateValue event handler to provide your own mechanism for updating the corresponding database field when the WebSnap application updates this adapter field's value. If you supply an OnUpdateValue event handler, the dataset adapter field assumes you have handled the update and does not write the new value to the associated dataset field. If there is no OnUpdateValue event handler, the dataset adapter field writes the field value to the associated dataset field.

Note: TBaseDataSetAdapterInputField defines but does not generate this event. Instead, descendant classes generate this event when updating the adapter field value. If you create your own TBaseDataSetAdapterInputField descendant, you must generate this event from your implementation of the UpdateValue method.

See Also

Value (see page 7180)

Value (see page 23945)

OnGetValue (see page 7754)

ImplUpdateValue (see page 7749)

41.1.17.2.6 TDataSetAdapterField.ValuesList Property

Specifies the interface of an object that lists the possible values for this field.

Pascal

```
property ValuesList: IValuesListAdapter;
```

C++

```
__property IValuesListAdapter ValuesList;
```

Description

ValuesList is the interface of an adapter that lists a set of values. It can be the interface of any object that supports the IValuesListAdapter interface, such as TStringsValuesList or TDataSetValuesList. Each value in the list that this adapter maintains represents a possible value for this field. This is useful when you want to restrict the possible values a user can assign to the associated database field.

When you assign a value to the ValuesList property, the default control for editing this fields value changes to a single selection list box.

See Also

TStringsValuesList (see page 24540)

TDataSetValuesList (see page 8079)

GetInputStyleType (see page 7831)

41.1.17.2.7 TDataSetAdapterField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property AnsiString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

HasViewAccess (see page 24162)

41.1.17.3 TDataSetAdapterField Events

The events of the TDataSetAdapterField class are listed here.

Published Events

	Name	Description
	OnGetDisplayText (see page 7971)	Occurs when the adapter field needs to supply its formatted value as a string that can appear in HTML forms.
	OnGetValue (see page 7971)	Occurs when the adapter field needs to supply its value as a Variant.
	OnValidateValue (see page 7972)	Occurs when the parent adapter wants to validate the new value for this field.

41.1.17.3.1 TDataSetAdapterField.OnGetDisplayText Event

Occurs when the adapter field needs to supply its formatted value as a string that can appear in HTML forms.

Pascal

```
property OnGetDisplayText: TDataSetAdapterFieldGetStringEvent;
```

C++

```
__property TDataSetAdapterFieldGetStringEvent OnGetDisplayText;
```

Description

TBaseDataSetAdapterField generates this event when supplying the read-only text representation of the field's value. This allows the WebSnap application to dynamically supply the formatted value of the field. Server-side script can access the resulting string using the DisplayText property.

Note: If you do not supply an OnGetDisplayText event handler, TBaseDataSetAdapterField uses the field component's DisplayText property.

See Also

DisplayText (see page 7162)

Value (see page 23945)

OnGetValue (see page 7754)

41.1.17.3.2 TDataSetAdapterField.OnGetValue Event

Occurs when the adapter field needs to supply its value as a Variant.

Pascal

```
property OnGetValue: TDataSetAdapterFieldGetValueEvent;
```

C++

```
__property TDataSetAdapterFieldGetValueEvent OnGetValue;
```

Description

TBaseDataSetAdapterField generates an OnGetValue event when the WebSnap application reads its Value property. This allows the adapter field to dynamically supply the value for the field. For example, you can use this event to decrypt values that are stored in the database in an encrypted form.

Note: If you do not supply an OnGetValue event handler, TBaseDataSetAdapterField uses the field component's Value property.

See Also

Value (see page 7180)

Value (see page 23945)

OnGetDisplayText (see page 7754)

41.1.17.3.3 TDataSetAdapterField.OnValidateValue Event

Occurs when the parent adapter wants to validate the new value for this field.

Pascal

```
property OnValidateValue: TDataAdapterFieldValidateValueEvent;
```

C++

```
__property TDataAdapterFieldValidateValueEvent OnValidateValue;
```

Description

Write an OnValidateValue event handler to provide your own validation of new field values before the WebSnap application attempts to update this field's value. If the new Value is not valid, the OnValidateValue event handler should raise an exception.

The parent dataset adapter generates this event when it receives new field values from an update request message.

See Also

Adapter (see page 7752)

Value (see page 23945)

OnGetValue (see page 7754)

OnValidate (see page 7183)

41.1.18 TDataSetAdapterFirstRowAction Class

TDataSetAdapterNextRowAction (see page 8030) posts any edits made in an HTML form and moves the dataset adapter to the first record.

Class Hierarchy

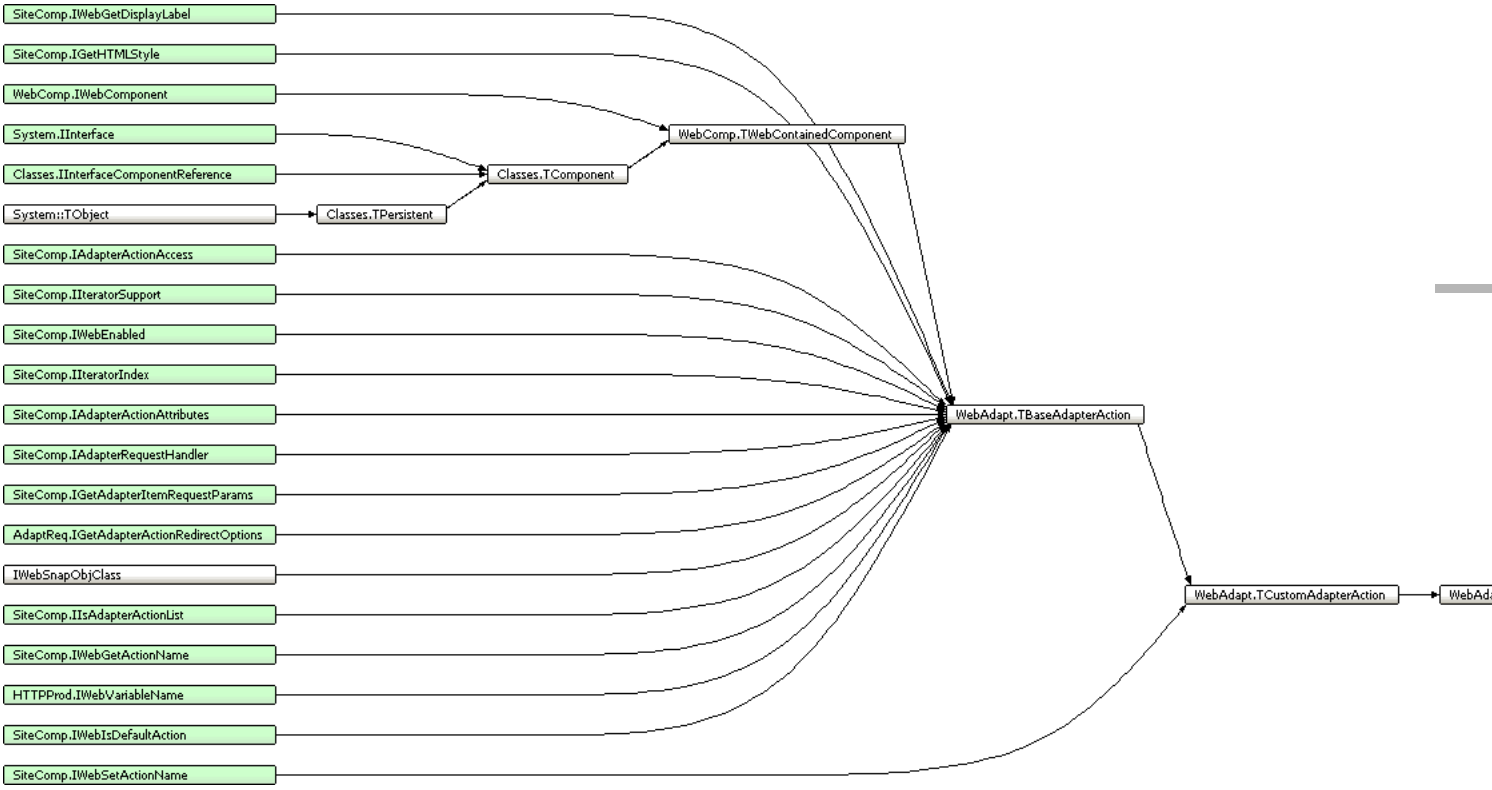

Pascal

```
TDataSetAdapterFirstRowAction = class(TDataSetAdapterMoveRowAction);
```

C++

```
class TDataSetAdapterFirstRowAction : public TDataSetAdapterMoveRowAction;
```

File

DBAdapt

Description

Add TDataSetAdapterBrowseRowAction (see page 7903) to a dataset adapter (TDataSetAdapter (see page 7872)) to support HTML forms that let users navigate to the first record in a dataset. When this action is executed, it locates the parent adapter's dataset on the record specified by the current request object, posts any edits included in the current request, and then moves the dataset to the first record.

Server-side script addresses this action as the 'FirstRow' action of the parent adapter. If the current request identifies the first record of the dataset as current, the control for this action appears disabled.

See Also

TDataSetAdapterApplyRowAction (see page 7890)

TDataSetAdapterBrowseRowAction (see page 7903)

TDataSetAdapterDeleteRowAction (see page 7934)

TDataSetAdapterEditRowAction (see page 7947)

TDataSetAdapterNextRowAction (see page 8030)

TDataSetAdapterLastRowAction (see page 7984)

TDataSetAdapterCancelRowAction (see page 7916)

TDataSetAdapterPrevRowAction (see page 8042)

TDataSetAdapterRefreshRowAction (see page 8054)

TDataSetAdapterNewRowAction (see page 8016)

TDataSetAdapter (see page 7872)

TCustomDataSetAdapter (see page 7766)

TDataSet (see page 6894)

41.1.18.1 TDataSetAdapterFirstRowAction Members

The following tables list the members exposed by TDataSetAdapterFirstRowAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 7819)	Returns the value of the Adapter (see page 7822) property.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7819)	Returns the value of the DataSet (see page 7822) property.
V	GetDefaultActionName (see page 7980)	Gets the default name used to reference this adapter action with server-side script.

	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 7980)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 7854)	Provides the underlying implementation for the GetAdapterItemRequestParams method.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 7819)	Provides the underlying implementation for the GetVisible method.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 8016)	Provides the underlying implementation for the IsDefaultAction method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.

	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
V	ImplWebEnabled (see page 7980)	Provides the underlying implementation for the WebEnabled method.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7822)	Indicates the dataset adapter that uses this object to represent one of its actions.
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 7822)	Occurs when server-side script executes this action.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 7823)	Occurs when the WebSnap application to determine whether a control or hyperlink for this action should be enabled.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 7818)	Creates and initializes an instance of TCustomDataSetAdapterAction (see page 7811).
V	Create (see page 7853)	Creates and initializes an instance of TCustomDataSetAdapterRowAction (see page 7846).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionModes (see page 7821)	Indicates what dataset adapter modes this field supports.
	ActionName (see page 7821)	Identifies the name of this action when using server-side script.
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 7822)	Indicates the dataset on which this action operates.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 7982)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 7983)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 7983)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 7984)	Called before the adapter action response is generated.
	OnGetParams (see page 7984)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 7981)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 7981)	Defines the rights required to execute this action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RedirectOptions (see page 7982)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.18.2 TDataSetAdapterFirstRowAction Methods

The methods of the TDataSetAdapterFirstRowAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 7980)	Gets the default name used to reference this adapter action with server-side script.
	ImplExecuteActionRequest (see page 7980)	Executes the adapter action using parameters passed in the HTTP request.
	ImplWebEnabled (see page 7980)	Provides the underlying implementation for the WebEnabled method.

41.1.18.2.1 TDataSetAdapterFirstRowAction.GetDefaultActionName Method

Gets the default name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultActionName();
```

Description

GetDefaultActionName returns the name of the adapter action component. Descendants may override this function.

See Also

ImplGetActionName (see page 24063)

41.1.18.2.2 TDataSetAdapterFirstRowAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest create a TStrings containing the name/value pairs in AActionRequest and calls the OnExecute event handler.

See Also

ImplExecuteActionRequest (see page 24063)

OnExecute (see page 24200)

41.1.18.2.3 TDataSetAdapterFirstRowAction.ImplWebEnabled Method

Provides the underlying implementation for the WebEnabled method.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

The WebEnabled method calls ImplWebEnabled to determine whether the control or hyperlink for this action should appear enabled in the HTML form. WebEnabled returns true if controls should be enabled, false otherwise.

ImplWebEnabled generates an OnGetEnabled event so that you can specify whether controls for this action are enabled. If there is no OnGetEnabled event handler, ImplWebEnabled returns true if the parent adapter's GetKeyFields method returns a set of fields that can locate records in the associated dataset.

See Also

WebEnabled (see page 24071)

OnGetEnabled (see page 7823)

GetKeyFields (see page 7786)

41.1.18.3 TDataSetAdapterFirstRowAction Properties

The properties of the TDataSetAdapterFirstRowAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 7981)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 7981)	Defines the rights required to execute this action.
	RedirectOptions (see page 7982)	Indicates whether to perform an HTTP redirect after successfully executing this action.

41.1.18.3.1 TDataSetAdapterFirstRowAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

41.1.18.3.2 TDataSetAdapterFirstRowAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property AnsiString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 25625)

hasexecuteaccess (see page 24162)

41.1.18.3.3 TDataSetAdapterFirstRowAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect (see page 15201)

41.1.18.4 TDataSetAdapterFirstRowAction Events

The events of the TDataSetAdapterFirstRowAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 7982)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 7983)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 7983)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 7984)	Called before the adapter action response is generated.
	OnGetParams (see page 7984)	Specifies name/value pairs needed to execute this adapter action.

41.1.18.4.1 TDataSetAdapterFirstRowAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

41.1.18.4.2 TDataSetAdapterFirstRowAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.18.4.3 TDataSetAdapterFirstRowAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

41.1.18.4.4 TDataSetAdapterFirstRowAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.18.4.5 TDataSetAdapterFirstRowAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

41.1.19 TDataSetAdapterLastRowAction Class

TDataSetAdapterLastRowAction posts any edits made in an HTML form and moves the dataset adapter to the last record.

Class Hierarchy

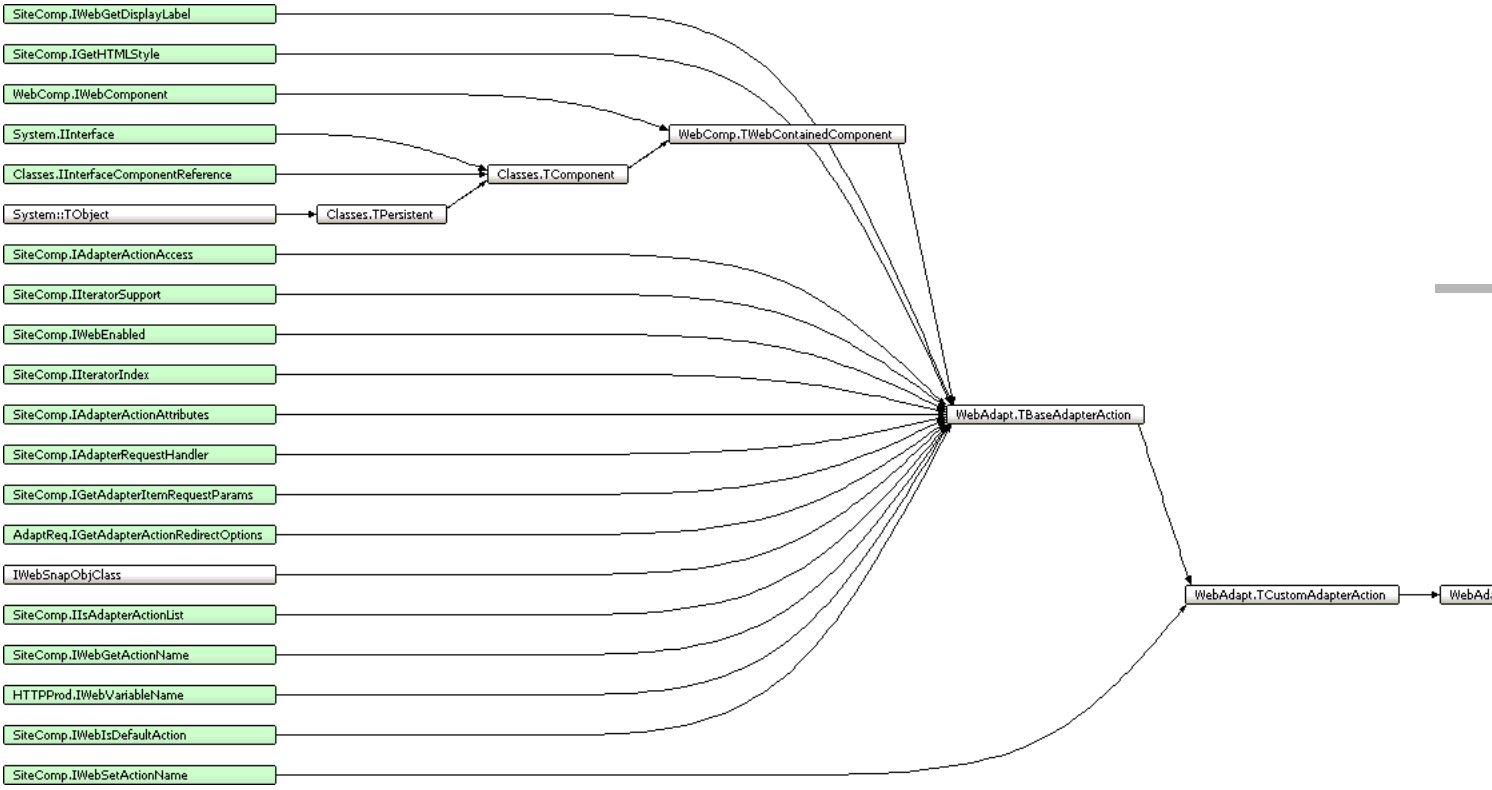

Pascal

```
TDataSetAdapterLastRowAction = class(TDataSetAdapterMoveRowAction);
```

C++

```
class TDataSetAdapterLastRowAction : public TDataSetAdapterMoveRowAction;
```

File

DBAdapt

Description

Add TDataSetAdapterLastRowAction to a dataset adapter (TDataSetAdapter (see page 7872)) to support HTML forms that let users jump to the last record in a dataset. When this action is executed, it locates the parent adapter's dataset on the record specified by the current request object, posts any edits included in the current request, and then moves the dataset to its last record.

Server-side script addresses this action as the 'LastRow' action of the parent adapter. If the current request identifies the last record of the dataset as current, the control for this action appears disabled.

See Also

TDataSetAdapterApplyRowAction (see page 7890)

TDataSetAdapterBrowseRowAction (see page 7903)

TDataSetAdapterDeleteRowAction (see page 7934)

TDataSetAdapterEditRowAction (see page 7947)

TDataSetAdapterFirstRowAction (see page 7972)

TDataSetAdapterNextRowAction (see page 8030)

TDataSetAdapterCancelRowAction (see page 7916)

TDataSetAdapterPrevRowAction (see page 8042)

TDataSetAdapterRefreshRowAction (see page 8054)

TDataSetAdapterNewRowAction (see page 8016)

TDataSetAdapter (see page 7872)

TCustomDataSetAdapter (see page 7766)

TDataSet (see page 6894)

41.1.19.1 TDataSetAdapterLastRowAction Members

The following tables list the members exposed by TDataSetAdapterLastRowAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 7819)	Returns the value of the Adapter (see page 7822) property.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7819)	Returns the value of the DataSet (see page 7822) property.
V	GetDefaultActionName (see page 7992)	Gets the default name used to reference this adapter action with server-side script.

	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 7992)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 7854)	Provides the underlying implementation for the GetAdapterItemRequestParams method.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 7819)	Provides the underlying implementation for the GetVisible method.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 8016)	Provides the underlying implementation for the IsDefaultAction method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.

	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
V	ImplWebEnabled (see page 7992)	Provides the underlying implementation for the WebEnabled method.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7822)	Indicates the dataset adapter that uses this object to represent one of its actions.
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 7822)	Occurs when server-side script executes this action.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 7823)	Occurs when the WebSnap application to determine whether a control or hyperlink for this action should be enabled.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 7818)	Creates and initializes an instance of TCustomDataSetAdapterAction (see page 7811).
V	Create (see page 7853)	Creates and initializes an instance of TCustomDataSetAdapterRowAction (see page 7846).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionModes (see page 7821)	Indicates what dataset adapter modes this field supports.
	ActionName (see page 7821)	Identifies the name of this action when using server-side script.
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 7822)	Indicates the dataset on which this action operates.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 7994)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 7995)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 7995)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 7996)	Called before the adapter action response is generated.
	OnGetParams (see page 7996)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 7993)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 7993)	Defines the rights required to execute this action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RedirectOptions (see page 7994)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.19.2 TDataSetAdapterLastRowAction Methods

The methods of the TDataSetAdapterLastRowAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 7992)	Gets the default name used to reference this adapter action with server-side script.
	ImplExecuteActionRequest (see page 7992)	Executes the adapter action using parameters passed in the HTTP request.
	ImplWebEnabled (see page 7992)	Provides the underlying implementation for the WebEnabled method.

41.1.19.2.1 TDataSetAdapterLastRowAction.GetDefaultActionName Method

Gets the default name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultActionName();
```

Description

GetDefaultActionName returns the name of the adapter action component. Descendants may override this function.

See Also

ImplGetActionName (see page 24063)

41.1.19.2.2 TDataSetAdapterLastRowAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest create a TStrings containing the name/value pairs in AActionRequest and calls the OnExecute event handler.

See Also

ImplExecuteActionRequest (see page 24063)

OnExecute (see page 24200)

41.1.19.2.3 TDataSetAdapterLastRowAction.ImplWebEnabled Method

Provides the underlying implementation for the WebEnabled method.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

The WebEnabled method calls ImplWebEnabled to determine whether the control or hyperlink for this action should appear enabled in the HTML form. WebEnabled returns true if controls should be enabled, false otherwise.

ImplWebEnabled generates an OnGetEnabled event so that you can specify whether controls for this action are enabled. If there is no OnGetEnabled event handler, ImplWebEnabled returns true if the parent adapter's GetKeyFields method returns a set of fields that can locate records in the associated dataset.

See Also

WebEnabled (see page 24071)

OnGetEnabled (see page 7823)

GetKeyFields (see page 7786)

41.1.19.3 TDataSetAdapterLastRowAction Properties

The properties of the TDataSetAdapterLastRowAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 7993)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 7993)	Defines the rights required to execute this action.
	RedirectOptions (see page 7994)	Indicates whether to perform an HTTP redirect after successfully executing this action.

41.1.19.3.1 TDataSetAdapterLastRowAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

41.1.19.3.2 TDataSetAdapterLastRowAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property AnsiString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 25625)

hasexecuteaccess (see page 24162)

41.1.19.3.3 TDataSetAdapterLastRowAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect (see page 15201)

41.1.19.4 TDataSetAdapterLastRowAction Events

The events of the TDataSetAdapterLastRowAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 7994)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 7995)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 7995)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 7996)	Called before the adapter action response is generated.
	OnGetParams (see page 7996)	Specifies name/value pairs needed to execute this adapter action.

41.1.19.4.1 TDataSetAdapterLastRowAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

41.1.19.4.2 TDataSetAdapterLastRowAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.19.4.3 TDataSetAdapterLastRowAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

41.1.19.4.4 TDataSetAdapterLastRowAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.19.4.5 TDataSetAdapterLastRowAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

41.1.20 TDataSetAdapterMemoField Class

The TDataSetAdapterMemoField represents a memo field in a dataset.

Class Hierarchy

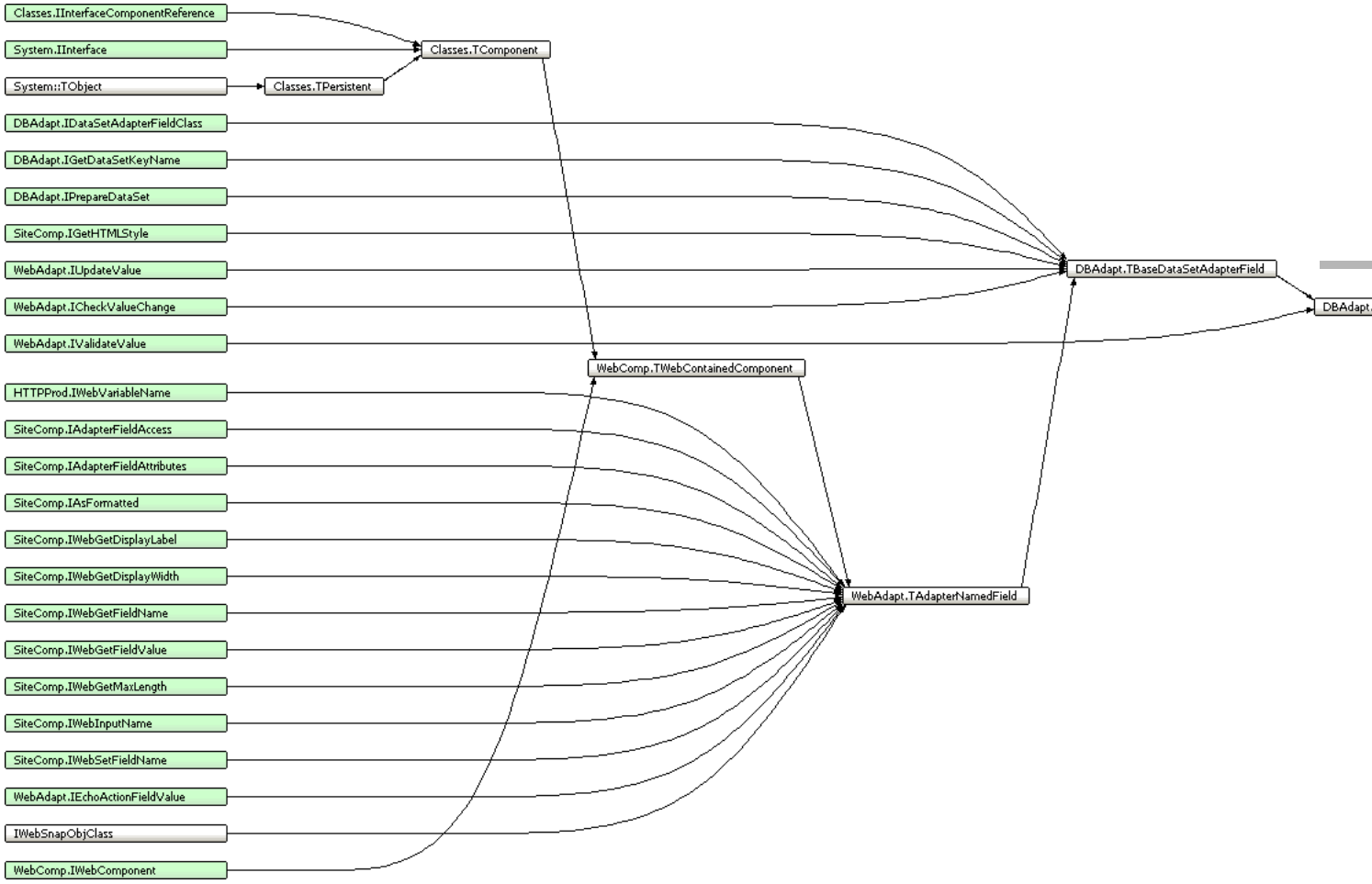

Pascal

```
TDataSetAdapterMemoField = class(TCustomDataSetAdapterMemoField);
```

C++

```
class TDataSetAdapterMemoField : public TCustomDataSetAdapterMemoField;
```

File

DBAdapt

Description

Add TDataSetAdapterMemoField as a child of a TDataSetAdapter (see page 7872) component to represent a memo field in the dataset associated with the dataset adapter. TDataSetAdapterMemoField represents a single field from a database table, and enables server-side script to refer to or update that field's value. When generating an HTML form or grid displaying dataset fields, server-side script calls the TDataSetAdapterMemoField to retrieve the value of the dataset field and write the value to the HTML content. When executing a TDataSetAdapter (see page 7872) action, the TDataSetAdapterMemoField is responsible for validating and updating the dataset field.

TDataSetAdapterField (see page 7960) supports several properties that can be used in server-side script. These include:

Value, which is the current value of the field.

DisplayText, which represents the current value of the field as a text string.

DisplayStyle and InputStyle, which indicate the type of control that should represent the field value in HTML forms.

DisplayLabel, which provides a text label that can be used to label HTML controls that represent the field.

Required, which indicates whether the field requires a value.

Visible, which indicates whether HTML forms should include a control for this field.

See Also

TMemoField (see page 7348)

TDataSetAdapter (see page 7872)

TAdapterDisplayField (see page 25109)

TAdapterDisplayColumn (see page 25099)

TDataSetAdapterField (see page 7960)

TDataSetAdapterImageField (see page 8138)

41.1.20.1 TDataSetAdapterMemoField Members

The following tables list the members exposed by TDataSetAdapterMemoField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckOrigValue (see page 7736)	Ensures that the current value of this field is the same as the original value.
	CheckOrUpdateValue (see page 7845)	Called by ImplUpdateValue (see page 7846) and by ImplCheckValueChange (see page 7845).
	CheckValueChange (see page 7737)	Indicates whether the current value of this field is the same as the original value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 7738)	Indicates whether references to this field in HTML forms must be qualified with the name of the parent adapter.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 7738)	Returns the value of the Adapter (see page 7752) property.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.

	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7738)	Returns the dataset associated with this field's parent adapter.
V	GetDataSetFieldValue (see page 7739)	Returns the value of the associated dataset field.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 7739)	Returns a string value that suggests an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 7740)	Returns the HTML display style type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 7741)	Returns a string value that suggests an HTML editing style for this adapter field.
V	GetInputStyleType (see page 7742)	Returns the style of the HTML control to use for editing this adapter field.
V	GetInputStyleType (see page 7845)	Returns htmliTextArea to indicate that this field should be represented by an HTML TextArea.
	GetKeyName (see page 7742)	Returns the name of the associated dataset field if it appears in the index (key) that the parent dataset adapter uses.
	GetKeyParams (see page 7743)	Returns a list of fields that the parent dataset adapter uses to locate records, along with their current values in that adapter's dataset.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetReadOnly (see page 7743)	Indicates whether the adapter field represents a read-only database field.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 7744)	Returns a string value that indicates what function a control that represents this field should perform.
V	GetViewModeType (see page 7744)	Returns the view mode for the function a control that represents this field should perform.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 7745)	Provides the underlying implementation for the AsFormatted method.
V	ImplAsFormatted (see page 7845)	Returns the text that is used to display a read only representation of the dataset field.
V	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
V	ImplCheckOrigValue (see page 7745)	Provides the underlying implementation for the CheckOrigValue (see page 7736) method.
V	ImplCheckValueChange (see page 7746)	Provides the underlying implementation for the CheckValueChange (see page 7737) method.
V	ImplCheckValueChange (see page 7845)	Indicates whether the data was changed by the user.
V	ImplGetDisplayLabel (see page 7747)	Provides the underlying implementation for the GetDisplayLabel method.
V	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 7747)	Provides the underlying implementation for the GetDisplayWidth method.
V	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 7763)	Provides the underlying implementation for the GetEchoActionFieldValue method.
V	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
V	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.

	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 7747)	Provides the underlying implementation for the GetRequired method.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 7748)	Provides the underlying implementation for the GetValue method.
	ImplGetValue (see page 7846)	Returns either the value of the associated dataset field as a Variant, or the last value entered by the end user.
	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 7748)	Provides the underlying implementation for the GetVisible method.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 7763)	Provides the underlying implementation for the SetEchoActionFieldValue method.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 7749)	Provides the underlying implementation for the SetFieldName method.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	ImplUpdateValue (see page 7749)	Provides the underlying implementation for the UpdateValue (see page 7751) method.
	ImplUpdateValue (see page 7846)	Verifies that the field has modify access rights, and if so updates the value for this TCustomDataSetAdapterMemoField (see page 7837).
	ImplValidateValue (see page 7764)	Provides the underlying implementation for the ValidateValue (see page 7765) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7750)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	PrepareDataSet (see page 7750)	Responds when the parent dataset adapter prepares its dataset.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSetField (see page 7751)	Sets the value of the DataSetField (see page 7752) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	TestValueChange (see page 7764)	Checks whether a new value matches the current value of this field.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	UpdateValue (see page 7751)	Updates the adapter field when the user assigns a new value.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateValue (see page 7765)	Validates a new value for this field that appears in the current request.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7752)	Indicates the dataset adapter that uses this object to represent one of its fields.
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 7735)	Creates and initializes an instance of TBaseDataSetAdapterField (see page 7729).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 7737)	Returns the dataset field whose value this adapter field represents.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetKeyName (see page 8105)	This is GetKeyName, a member of class IGetDataSetKeyName.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PrepareDataSet (see page 8109)	This is PrepareDataSet, a member of class IPrepareDataSet.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
R	KeyName (see page 8105)	This is KeyName, a member of class IGetDataSetKeyName.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetDisplayText (see page 8007)	Occurs when the adapter field needs to supply its formatted value as a string that can appear in HTML forms.
	OnGetValue (see page 8007)	Occurs when the adapter field needs to supply its value as a Variant.
	OnValidateValue (see page 8008)	Occurs when the parent adapter wants to validate the new value for this field.

Published Properties

	Name	Description
	DataSetField (see page 8004)	Specifies the name of the dataset field that this adapter field represents.
	FieldFlags (see page 8004)	Indicates various attributes of the adapter field.
	FieldModes (see page 8005)	Indicates what dataset adapter modes this field supports.
	ModifyAccess (see page 8005)	Identifies the rights required to update this field.

	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OnUpdateValue (see page 8006)	Occurs when the WebSnap application updates this adapter field's value based on in incoming request.
	Tag (see page 2808)	Stores an integer value as part of a component.
	ViewAccess (see page 8006)	Identifies the rights required to view this field.

41.1.20.2 TDataSetAdapterMemoField Properties

The properties of the TDataSetAdapterMemoField class are listed here.

Published Properties

	Name	Description
	DataSetField (see page 8004)	Specifies the name of the dataset field that this adapter field represents.
	FieldFlags (see page 8004)	Indicates various attributes of the adapter field.
	FieldModes (see page 8005)	Indicates what dataset adapter modes this field supports.
	ModifyAccess (see page 8005)	Identifies the rights required to update this field.
	OnUpdateValue (see page 8006)	Occurs when the WebSnap application updates this adapter field's value based on in incoming request.
	ViewAccess (see page 8006)	Identifies the rights required to view this field.

41.1.20.2.1 TDataSetAdapterMemoField.DataSetField Property

Specifies the name of the dataset field that this adapter field represents.

Pascal

```
property DataSetField: string;
```

C++

```
__property AnsiString DataSetField;
```

Description

DataSetField is the FieldName of the TField object whose value this adapter field represents. DataSetField can be the name of a simple field in a database table, the name of a subfield of an object field that has been qualified by the parent field's name, or the name of an aggregated field.

DataSetField is the same as the name of this adapter field in server-side script, except that the name of the adapter field omits any illegal characters (the only legal characters are the letters between a and z (or A and Z), digits, and the underline character).

If DataSetField is an empty string, this adapter field does not have a value, and the FindField method returns nil (Delphi) or NULL (C++).

See Also

FieldName (see page 7164)

FindField (see page 7737)

41.1.20.2.2 TDataSetAdapterMemoField.FieldFlags Property

Indicates various attributes of the adapter field.

Pascal

```
property FieldFlags: TDataSetAdapterFieldFlags;
```

C++

```
__property TDataSetAdapterFieldFlags FieldFlags;
```

Description

FieldFlags indicates special uses of the value of this adapter field. This is a set that can indicate whether the value corresponds to a key field in the database (used to locate database records) and whether HTML forms store the original value of the field as a hidden field so that the application can detect changes to the field value.

See Also

IsIndexField (see page 7168)

DataSetField (see page 7752)

GetKeyName (see page 7742)

GetKeyFields (see page 7786)

41.1.20.2.3 TDataSetAdapterMemoField.FieldModes Property

Indicates what dataset adapter modes this field supports.

Pascal

```
property FieldModes: TDataSetAdapterModes;
```

C++

```
__property TDataSetAdapterModes FieldModes;
```

Description

FieldModes is a set of all the modes in which this adapter field can be used. If the parent dataset adapter is in a mode that is not included in this set, any forms generated using that adapter do not include a control to represent this field. Thus, for example, to make a field read-only, you can exclude amInsert and amEdit from FieldModes. Then, the field appears when the user is viewing or querying the dataset, but not when the user is editing or inserting a record.

See Also

CanModify (see page 7154)

Adapter (see page 7752)

Mode (see page 7810)

GetDisplayStyle (see page 7739)

GetInputStyle (see page 7741)

GetViewMode (see page 7744)

41.1.20.2.4 TDataSetAdapterMemoField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property AnsiString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user

attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

hasmodifyaccess (see page 24162)

41.1.20.2.5 TDataSetAdapterMemoField.OnUpdateValue Property

Occurs when the WebSnap application updates this adapter field's value based on in incoming request.

Pascal

```
property OnUpdateValue: TDatasetAdapterFieldValueEvent;
```

C++

```
__property TDatasetAdapterFieldValueEvent OnUpdateValue;
```

Description

Write an OnUpdateValue event handler to provide your own mechanism for updating the corresponding database field when the WebSnap application updates this adapter field's value. If you supply an OnUpdateValue event handler, the dataset adapter field assumes you have handled the update and does not write the new value to the associated dataset field. If there is no OnUpdateValue event handler, the dataset adapter field writes the field value to the associated dataset field.

Note: TBaseDataSetAdapterInputField defines but does not generate this event. Instead, descendant classes generate this event when updating the adapter field value. If you create your own TBaseDataSetAdapterInputField descendant, you must generate this event from your implementation of the UpdateValue method.

See Also

Value (see page 7180)

Value (see page 23945)

OnGetValue (see page 7754)

ImplUpdateValue (see page 7749)

41.1.20.2.6 TDataSetAdapterMemoField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property AnsiString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

HasViewAccess (see page 24162)

41.1.20.3 TDataSetAdapterMemoField Events

The events of the TDataSetAdapterMemoField class are listed here.

Published Events

	Name	Description
	OnGetDisplayText (see page 8007)	Occurs when the adapter field needs to supply its formatted value as a string that can appear in HTML forms.
	OnGetValue (see page 8007)	Occurs when the adapter field needs to supply its value as a Variant.
	OnValidateValue (see page 8008)	Occurs when the parent adapter wants to validate the new value for this field.

41.1.20.3.1 TDataSetAdapterMemoField.OnGetDisplayText Event

Occurs when the adapter field needs to supply its formatted value as a string that can appear in HTML forms.

Pascal

```
property OnGetDisplayText: TDataSetAdapterFieldGetStringEvent;
```

C++

```
__property TDataSetAdapterFieldGetStringEvent OnGetDisplayText;
```

Description

TBaseDataSetAdapterField generates this event when supplying the read-only text representation of the field's value. This allows the WebSnap application to dynamically supply the formatted value of the field. Server-side script can access the resulting string using the DisplayText property.

Note: If you do not supply an OnGetDisplayText event handler, TBaseDataSetAdapterField uses the field component's DisplayText property.

See Also

DisplayText (see page 7162)

Value (see page 23945)

OnGetValue (see page 7754)

41.1.20.3.2 TDataSetAdapterMemoField.OnGetValue Event

Occurs when the adapter field needs to supply its value as a Variant.

Pascal

```
property OnGetValue: TDataSetAdapterFieldGetValueEvent;
```

C++

```
__property TDataSetAdapterFieldGetValueEvent OnGetValue;
```

Description

TBaseDataSetAdapterField generates an OnGetValue event when the WebSnap application reads its Value property. This allows the adapter field to dynamically supply the value for the field. For example, you can use this event to decrypt values that are stored in the database in an encrypted form.

Note: If you do not supply an OnGetValue event handler, TBaseDataSetAdapterField uses the field component's Value property.

See Also

Value (see page 7180)

Value (see page 23945)

OnGetDisplayText (see page 7754)

41.1.20.3.3 TDataSetAdapterMemoField.OnValidateValue Event

Occurs when the parent adapter wants to validate the new value for this field.

Pascal

```
property OnValidateValue: TDataAdapterFieldValidateValueEvent;
```

C++

```
__property TDataAdapterFieldValidateValueEvent OnValidateValue;
```

Description

Write an OnValidateValue event handler to provide your own validation of new field values before the WebSnap application attempts to update this field's value. If the new Value is not valid, the OnValidateValue event handler should raise an exception.

The parent dataset adapter generates this event when it receives new field values from an update request message.

See Also

Adapter (see page 7752)

Value (see page 23945)

OnGetValue (see page 7754)

OnValidate (see page 7183)

41.1.21 TDataSetAdapterMoveRowAction Class

TDataSetAdapterMoveRowAction is the base class for adapter actions that change the current record of a dataset.

Class Hierarchy

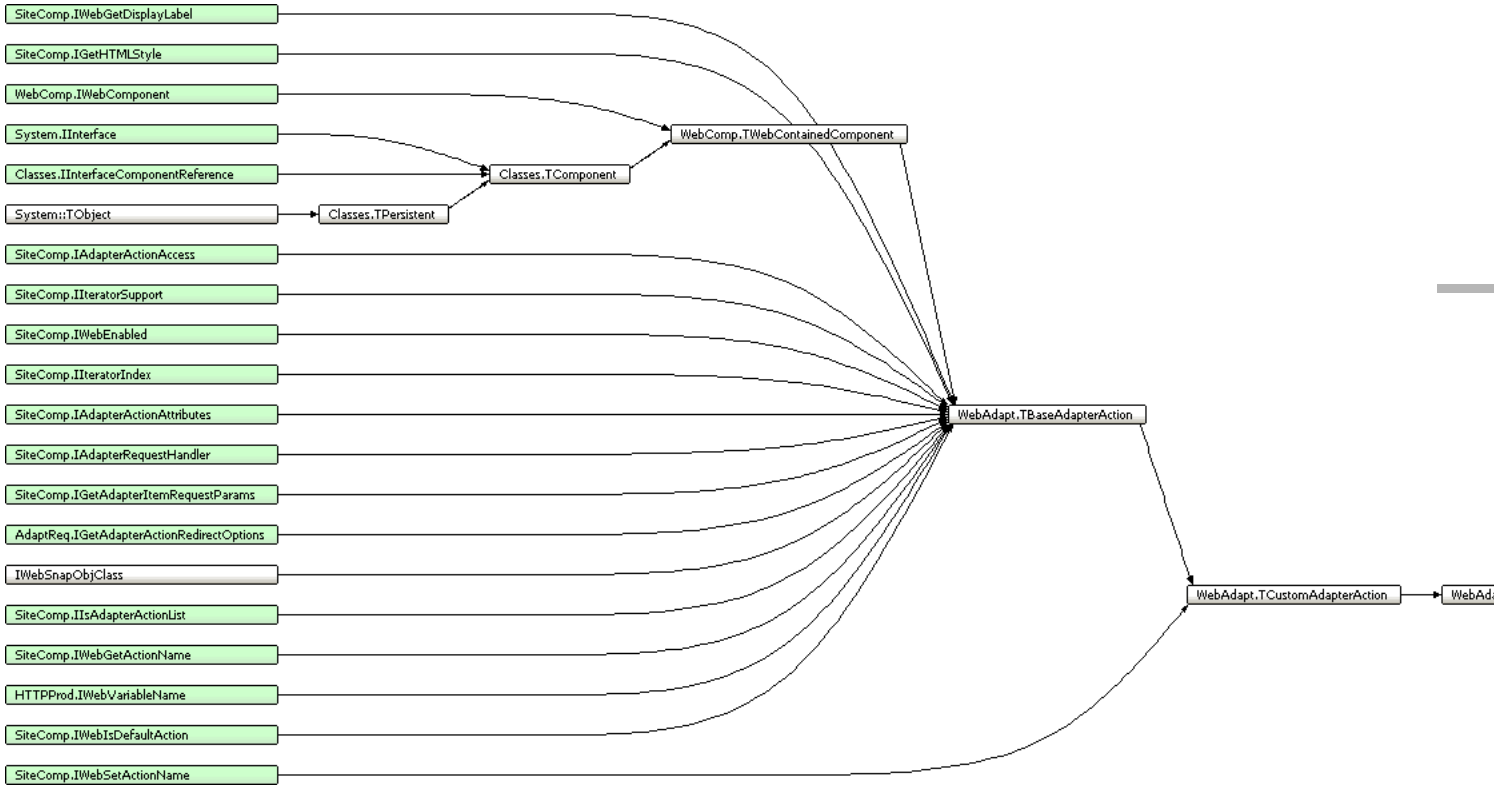

Pascal

```
TDataSetAdapterMoveRowAction = class(TCustomDataSetAdapterRowAction);
```

C++

```
class TDataSetAdapterMoveRowAction : public TCustomDataSetAdapterRowAction;
```

File

DBAdapt

Description

TDataSetAdapterMoveRowAction is the base class for all adapter action objects that change the current record of the dataset associated with a dataset adapter. TDataSetAdapterMoveRowAction descendants are included as the actions of a dataset adapter (TDataSetAdapter (see page 7872)). Each TDataSetAdapterMoveRowAction descendant can be executed to change the current record of the parent adapter's dataset.

Do not use TDataSetAdapterMoveRowAction directly. It is intended only as a base class. Instead, use a TDataSetAdapterMoveRowAction descendant such as TDataSetAdapterFirstRowAction (see page 7972), TDataSetAdapterLastRowAction (see page 7984), TDataSetAdapterNextRowAction (see page 8030), or TDataSetAdapterPrevRowAction (see page 8042).

See Also

TDataSetAdapterFirstRowAction (see page 7972)

TDataSetAdapterLastRowAction (see page 7984)

TDataSetAdapterNextRowAction (see page 8030)

TDataSetAdapterPrevRowAction (see page 8042)

TDataSetAdapter (see page 7872)

41.1.21.1 TDataSetAdapterMoveRowAction Members

The following tables list the members exposed by TDataSetAdapterMoveRowAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 7819)	Returns the value of the Adapter (see page 7822) property.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7819)	Returns the value of the DataSet (see page 7822) property.
V	GetDefaultActionName (see page 24055)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.

	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 24063)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 24195)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 7854)	Provides the underlying implementation for the GetAdapterItemRequestParams method.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 7819)	Provides the underlying implementation for the GetVisible method.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 8016)	Provides the underlying implementation for the IsDefaultAction method.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 7820)	Provides the underlying implementation for the WebEnabled method.
	ImplWebEnabled (see page 7855)	Provides the underlying implementation for the WebEnabled method.
	ImplWebEnabled (see page 24069)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 24197)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.

	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7822)	Indicates the dataset adapter that uses this object to represent one of its actions.
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 24075)	Called after the adapter action has executed.
	OnAfterExecute (see page 24198)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 24075)	Called after the adapter action response is generated.

	OnAfterGetResponse (see page 24199)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24076)	Called before the adapter action is executed.
	OnBeforeExecute (see page 24199)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24076)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 24200)	Called before the adapter action response is generated.
	OnExecute (see page 7822)	Occurs when server-side script executes this action.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 7823)	Occurs when the WebSnap application to determine whether a control or hyperlink for this action should be enabled.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 24077)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 7818)	Creates and initializes an instance of TCustomDataSetAdapterAction (see page 7811).
V	Create (see page 7853)	Creates and initializes an instance of TCustomDataSetAdapterRowAction (see page 7846).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.

	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionModes (see page 7821)	Indicates what dataset adapter modes this field supports.
	ActionName (see page 7821)	Identifies the name of this action when using server-side script.
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 7822)	Indicates the dataset on which this action operates.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 24073)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 24073)	Defines the rights required to execute this action.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RedirectOptions (see page 24074)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.21.2 TDataSetAdapterMoveRowAction Methods

The methods of the TDataSetAdapterMoveRowAction class are listed here.

Protected Methods

	Name	Description
	ImplIsDefaultAction (see page 8016)	Provides the underlying implementation for the IsDefaultAction method.

41.1.21.2.1 TDataSetAdapterMoveRowAction.ImplIsDefaultAction Method

Provides the underlying implementation for the IsDefaultAction method.

Pascal

```
function ImplIsDefaultAction(ADisplay: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplIsDefaultAction(TObject * ADisplay);
```

Description

The IsDefaultAction calls ImplIsDefaultAction to determine whether this action should be added, by default, to the parent adapter. Default actions are added to the adapter unless you explicitly specify the actions of the parent adapter.

ADisplay is the Web item associated with the parent adapter that generates HTML content to represent the adapter's fields and actions. This item is a member of the hierarchy of Web items rooted by a TAdapterForm object.

ImplIsDefaultAction returns true if this action should be included among the adapter's default actions, false if it should only be included when explicitly added to the parent adapter.

ImplIsDefaultAction always returns false when the parent adapter is in query mode. When the parent adapter is in any other mode, ImplIsDefaultAction checks whether ADisplay has a GetAdapterDisplayCharacteristics method. If ADisplay can provide display characteristics, ImplIsDefaultAction returns true only if those characteristics indicate an ability to change the current record. If ADisplay can't provide displaycharacteristics, ImplIsDefaultAction returns true.

See Also

IsDefaultAction (see page 24070)

Mode (see page 7810)

GetAdapterDisplayCharacteristics (see page 25129)

TAdapterForm (see page 25159)

41.1.22 TDataSetAdapterNewRowAction Class

TDataSetAdapterNewRowAction places the parent dataset adapter in insert mode.

Class Hierarchy

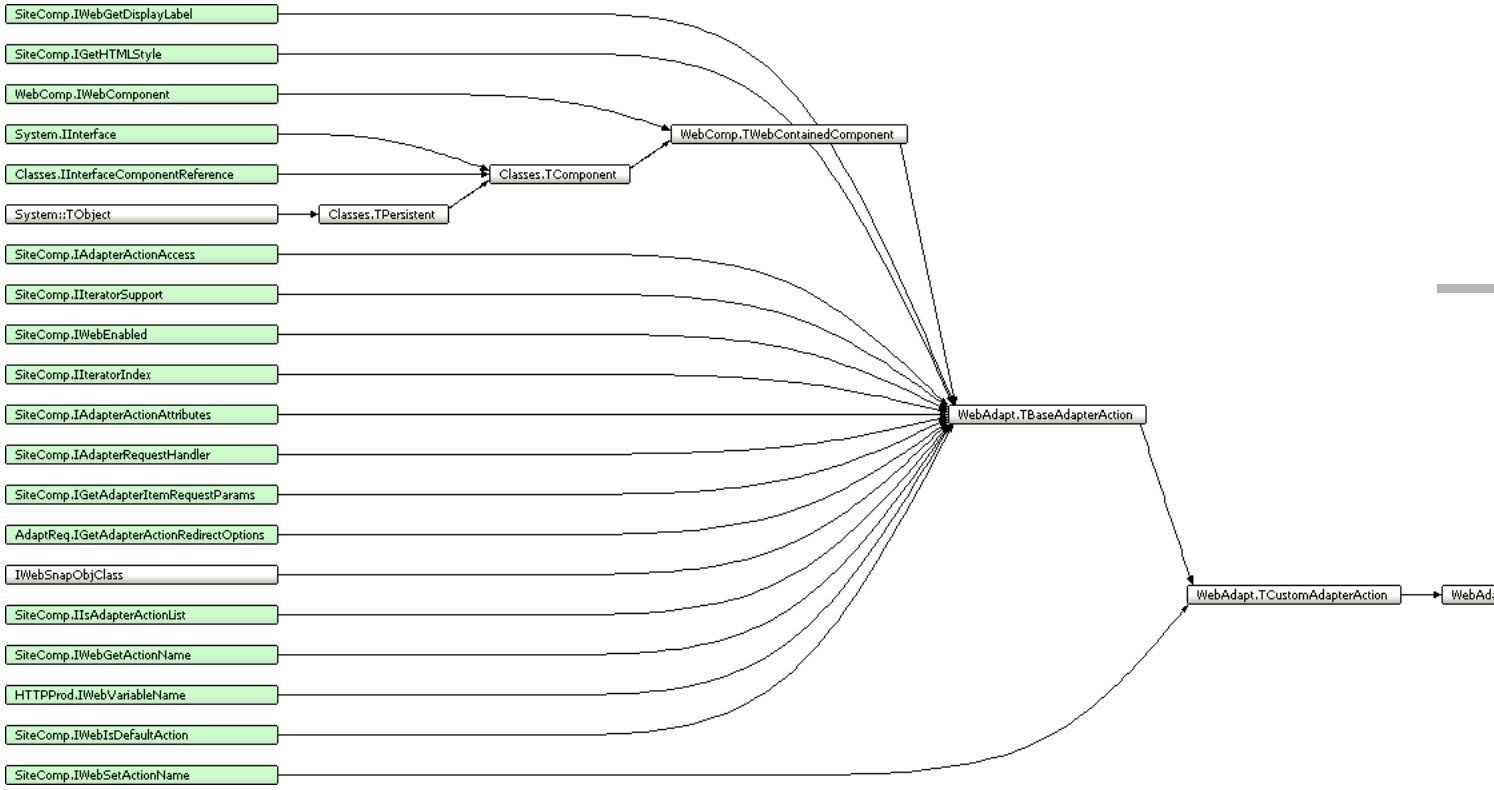

Pascal

```
TDataSetAdapterNewRowAction = class(TDataSetAdapterRowAction);
```

C++

```
class TDataSetAdapterNewRowAction : public TDataSetAdapterRowAction;
```

File

DBAdapt

Description

Add TDataSetAdapterDeleteRowAction (see page 7934) to a dataset adapter (TDataSetAdapter (see page 7872)) to support HTML forms that let users insert new database records. When this action is executed, it positions the dataset on the record specified in the current request, applies any updates in that request, and then puts the parent dataset adapter into insert mode.

Server-side script addresses this action as the 'NewRow' action of the parent adapter.

See Also

TDataSetAdapterApplyRowAction (see page 7890)

TDataSetAdapterBrowseRowAction (see page 7903)

TDataSetAdapterCancelRowAction (see page 7916)

TDataSetAdapterEditRowAction (see page 7947)

TDataSetAdapterFirstRowAction (see page 7972)

TDataSetAdapterLastRowAction (see page 7984)

TDataSetAdapterNextRowAction (see page 8030)

TDataSetAdapterPrevRowAction (see page 8042)

TDataSetAdapterRefreshRowAction (see page 8054)

TDataSetAdapterDeleteRowAction (see page 7934)

TDataSetAdapter (see page 7872)

TCustomDataSetAdapter (see page 7766)

TDataSetAdapterRowAction (see page 8066)

TCustomDataSetAdapter (see page 7766)

41.1.22.1 TDataSetAdapterNewRowAction Members

The following tables list the members exposed by TDataSetAdapterNewRowAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 7819)	Returns the value of the Adapter (see page 7822) property.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7819)	Returns the value of the DataSet (see page 7822) property.
V	GetDefaultActionName (see page 8024)	Gets the default name used to reference this adapter action with server-side script.

	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 8025)	Provides the underlying implementation for the ExecuteActionRequest method.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 7854)	Provides the underlying implementation for the GetAdapterItemRequestParams method.
	ImplGetAdapterItemRequestParams (see page 8074)	Provides the underlying implementation for the GetAdapterItemRequestParams method.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 7819)	Provides the underlying implementation for the GetVisible method.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.

V	ImplIsDefaultAction (see page 8025)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
V	ImplWebEnabled (see page 8026)	Provides the underlying implementation for the WebEnabled method.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7822)	Indicates the dataset adapter that uses this object to represent one of its actions.
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.

R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 7822)	Occurs when server-side script executes this action.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 7823)	Occurs when the WebSnap application to determine whether a control or hyperlink for this action should be enabled.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.

	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionName (see page 7821)	Identifies the name of this action when using server-side script.

R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 7822)	Indicates the dataset on which this action operates.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RedirectOptions (see page 24074)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 8028)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 8028)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 8029)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 8029)	Called before the adapter action response is generated.
	OnGetParams (see page 8029)	Specifies name/value pairs needed to execute this adapter action.

Published Methods

	Name	Description
V	Create (see page 8024)	Creates and initializes an instance of TDataSetAdapterNewRowAction (see page 8016).

Published Properties

	Name	Description
	ActionModes (see page 8026)	Indicates what dataset adapter modes this field supports.
	DisplayLabel (see page 8027)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 8027)	Defines the rights required to execute this action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RedirectOptions (see page 8076)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.22.2 TDataSetAdapterNewRowAction.Create Constructor

Creates and initializes an instance of TDataSetAdapterNewRowAction (see page 8016).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDataSetAdapterNewRowAction(TComponent * AOwner);
```

Description

The TDataSetAdapterNewRowAction (see page 8016) constructor allocates memory for the adapter action object and initializes its properties.

AOwner is the component that is responsible for starting the streaming process when a module containing the adapter action is loaded or saved. It becomes the value of the Owner property. Typically, AOwner is the Web module.

See Also

Owner (see page 2807)

41.1.22.3 TDataSetAdapterNewRowAction Methods

The methods of the TDataSetAdapterNewRowAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 8024)	Gets the default name used to reference this adapter action with server-side script.
	ImplExecuteActionRequest (see page 8025)	Provides the underlying implementation for the ExecuteActionRequest method.
	ImplIsDefaultAction (see page 8025)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplWebEnabled (see page 8026)	Provides the underlying implementation for the WebEnabled method.

41.1.22.3.1 TDataSetAdapterNewRowAction.GetDefaultActionName Method

Gets the default name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultActionName();
```

Description

GetDefaultActionName returns the name of the adapter action component. Descendants may override this function.

See Also

ImplGetActionName (see page 24063)

41.1.22.3.2 TDataSetAdapterNewRowAction.ImplExecuteActionRequest Method

Provides the underlying implementation for the ExecuteActionRequest method.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

The ExecuteActionRequest method calls ImplExecuteActionRequest to execute this action in response to server-side script.

AActionRequest is an interface to the current request object. ImplExecuteActionRequest uses this to obtain information about the current request (such as the values of request parameters).

AActionResponse is an interface to the current response object. ImplExecuteActionRequest uses this to indicate the status of the emerging HTTP response.

ImplExecuteActionRequest generates an OnExecute event, allowing you to provide the implementation for this action in an OnExecute event handler. If you do not supply an OnExecute event handler, ImplExecuteActionRequest calls the parent adapter's LocateAndApply method, to position the adapter's dataset on the record indicated by the current request and apply any updates that the request contains.

See Also

ExecuteActionRequest (see page 24053)

OnExecute (see page 7822)

LocateAndApply (see page 7796)

41.1.22.3.3 TDataSetAdapterNewRowAction.ImplIsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function ImplIsDefaultAction(ADisplay: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplIsDefaultAction(TObject * ADisplay);
```

Description

ImplIsDefaultAction is called by IsDefaultAction. ImplIsDefaultAction returns true. ImplIsDefaultAction may be overridden by a descendant class.

See Also

IsDefaultAction (see page 24070)

ImplIsDefaultAction (see page 24243)

41.1.22.3.4 TDataSetAdapterNewRowAction.ImplWebEnabled Method

Provides the underlying implementation for the WebEnabled method.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

The WebEnabled method calls ImplWebEnabled to determine whether the control or hyperlink for this action should appear enabled in the HTML form. WebEnabled returns true if controls should be enabled, false otherwise.

ImplWebEnabled generates an OnGetEnabled event so that you can specify whether controls for this action are enabled. If there is no OnGetEnabled event handler, ImplWebEnabled returns true if the parent adapter's GetKeyFields method returns a set of fields that can locate records in the associated dataset.

See Also

WebEnabled (see page 24071)

OnGetEnabled (see page 7823)

GetKeyFields (see page 7786)

41.1.22.4 TDataSetAdapterNewRowAction Properties

The properties of the TDataSetAdapterNewRowAction class are listed here.

Published Properties

	Name	Description
	ActionModes (see page 8026)	Indicates what dataset adapter modes this field supports.
	DisplayLabel (see page 8027)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 8027)	Defines the rights required to execute this action.

41.1.22.4.1 TDataSetAdapterNewRowAction.ActionModes Property

Indicates what dataset adapter modes this field supports.

Pascal

```
property ActionModes: TDataSetAdapterModes;
```

C++

```
__property TDataSetAdapterModes ActionModes;
```

Description

ActionModes is a set of all the modes in which this adapter action can be used. If the parent dataset adapter is in a mode that is not included in this set, any forms generated using that adapter do not include a control for executing this action. Thus, for example, there is no sense in adding a control that applies edits to a form that presents a read-only view of the data. An apply action, then, should not include amBrowse in its action modes.

See Also

CanModify (see page 7154)

Adapter (see page 7752)

Mode (see page 7810)

GetDisplayStyle (see page 7739)

GetInputStyle (see page 7741)

GetViewMode (see page 7744)

41.1.22.4.2 TDataSetAdapterNewRowAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

41.1.22.4.3 TDataSetAdapterNewRowAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property AnsiString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 25625)

hasexecuteaccess (see page 24162)

41.1.22.5 TDataSetAdapterNewRowAction Events

The events of the TDataSetAdapterNewRowAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 8028)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 8028)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 8029)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 8029)	Called before the adapter action response is generated.
	OnGetParams (see page 8029)	Specifies name/value pairs needed to execute this adapter action.

41.1.22.5.1 TDataSetAdapterNewRowAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

41.1.22.5.2 TDataSetAdapterNewRowAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.22.5.3 TDataSetAdapterNewRowAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

41.1.22.5.4 TDataSetAdapterNewRowAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.22.5.5 TDataSetAdapterNewRowAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

41.1.23 TDataSetAdapterNextRowAction Class

TDataSetAdapterNextRowAction posts any edits made in an HTML form and moves the dataset adapter to the next record.

Class Hierarchy

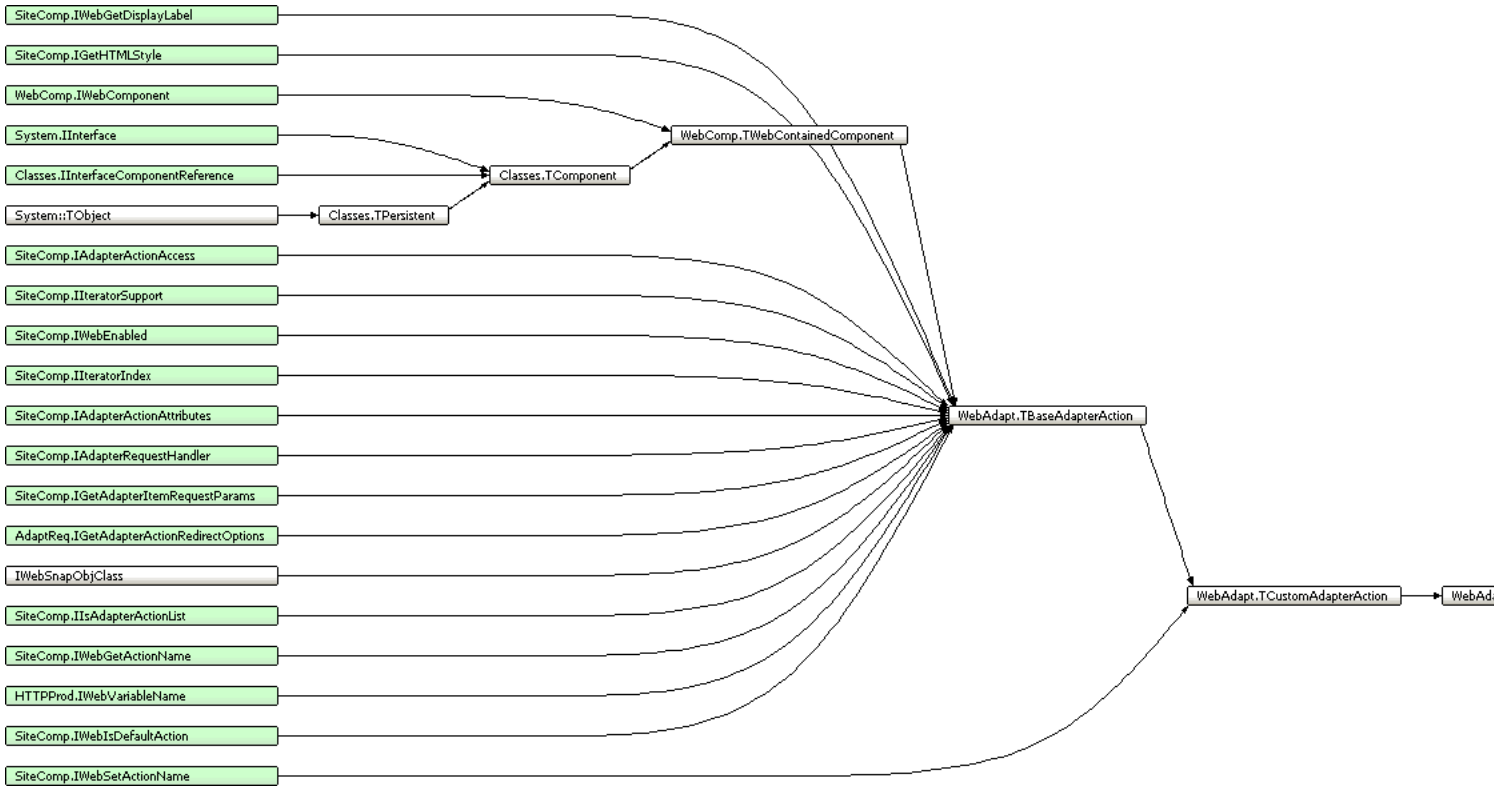

Pascal

```
TDataSetAdapterNextRowAction = class(TDataSetAdapterMoveRowAction);
```

C++

```
class TDataSetAdapterNextRowAction : public TDataSetAdapterMoveRowAction;
```

File

DBAdapt

Description

Add TDataSetAdapterNextRowAction to a dataset adapter (TDataSetAdapter (see page 7872)) to support HTML forms that let users progress to the next dataset record. When this action is executed, it locates the parent adapter's dataset on the record

specified by the current request object, posts any edits included in the current request, and then advances the dataset to the next record.

Server-side script addresses this action as the 'NextRow' action of the parent adapter. If the current request identifies the last record of the dataset as current, the control for this action appears disabled.

See Also

TDataSetAdapterApplyRowAction (see page 7890)

TDataSetAdapterBrowseRowAction (see page 7903)

TDataSetAdapterDeleteRowAction (see page 7934)

TDataSetAdapterEditRowAction (see page 7947)

TDataSetAdapterFirstRowAction (see page 7972)

TDataSetAdapterLastRowAction (see page 7984)

TDataSetAdapterCancelRowAction (see page 7916)

TDataSetAdapterPrevRowAction (see page 8042)

TDataSetAdapterRefreshRowAction (see page 8054)

TDataSetAdapterNewRowAction (see page 8016)

TDataSetAdapter (see page 7872)

TCustomDataSetAdapter (see page 7766)

TDataSet (see page 6894)

41.1.23.1 TDataSetAdapterNextRowAction Members

The following tables list the members exposed by TDataSetAdapterNextRowAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.

	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 7819)	Returns the value of the Adapter (see page 7822) property.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7819)	Returns the value of the DataSet (see page 7822) property.
	GetDefaultActionName (see page 8037)	Gets the default name used to reference this adapter action with server-side script.
	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 8037)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 7854)	Provides the underlying implementation for the GetAdapterItemRequestParams method.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.

	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 7819)	Provides the underlying implementation for the GetVisible method.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 8016)	Provides the underlying implementation for the IsDefaultAction method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 8038)	Provides the underlying implementation for the WebEnabled method.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.

	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7822)	Indicates the dataset adapter that uses this object to represent one of its actions.
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 7822)	Occurs when server-side script executes this action.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 7823)	Occurs when the WebSnap application to determine whether a control or hyperlink for this action should be enabled.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 7818)	Creates and initializes an instance of TCustomDataSetAdapterAction (see page 7811).
V	Create (see page 7853)	Creates and initializes an instance of TCustomDataSetAdapterRowAction (see page 7846).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionModes (see page 7821)	Indicates what dataset adapter modes this field supports.
	ActionName (see page 7821)	Identifies the name of this action when using server-side script.
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 7822)	Indicates the dataset on which this action operates.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 8040)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 8040)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 8040)	Called before the adapter action is executed.

	OnBeforeGetResponse (see page 8041)	Called before the adapter action response is generated.
	OnGetParams (see page 8041)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 8038)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 8039)	Defines the rights required to execute this action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RedirectOptions (see page 8039)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.23.2 TDataSetAdapterNextRowAction Methods

The methods of the TDataSetAdapterNextRowAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 8037)	Gets the default name used to reference this adapter action with server-side script.
	ImplExecuteActionRequest (see page 8037)	Executes the adapter action using parameters passed in the HTTP request.
	ImplWebEnabled (see page 8038)	Provides the underlying implementation for the WebEnabled method.

41.1.23.2.1 TDataSetAdapterNextRowAction.GetDefaultActionName Method

Gets the default name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultActionName();
```

Description

GetDefaultActionName returns the name of the adapter action component. Descendants may override this function.

See Also

ImplGetActionName (see page 24063)

41.1.23.2.2 TDataSetAdapterNextRowAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest create a TStrings containing the name/value pairs in AActionRequest and calls the OnExecute event

handler.

See Also

ImplExecuteActionRequest (see page 24063)

OnExecute (see page 24200)

41.1.23.2.3 TDataSetAdapterNextRowAction.ImplWebEnabled Method

Provides the underlying implementation for the WebEnabled method.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

The WebEnabled method calls ImplWebEnabled to determine whether the control or hyperlink for this action should appear enabled in the HTML form. WebEnabled returns true if controls should be enabled, false otherwise.

ImplWebEnabled generates an OnGetEnabled event so that you can specify whether controls for this action are enabled. If there is no OnGetEnabled event handler, ImplWebEnabled returns true if the parent adapter's GetKeyFields method returns a set of fields that can locate records in the associated dataset.

See Also

WebEnabled (see page 24071)

OnGetEnabled (see page 7823)

GetKeyFields (see page 7786)

41.1.23.3 TDataSetAdapterNextRowAction Properties

The properties of the TDataSetAdapterNextRowAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 8038)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 8039)	Defines the rights required to execute this action.
	RedirectOptions (see page 8039)	Indicates whether to perform an HTTP redirect after successfully executing this action.

41.1.23.3.1 TDataSetAdapterNextRowAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this

adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

41.1.23.3.2 TDataSetAdapterNextRowAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property AnsiString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 25625)

hasexecuteaccess (see page 24162)

41.1.23.3.3 TDataSetAdapterNextRowAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect (see page 15201)

41.1.23.4 TDataSetAdapterNextRowAction Events

The events of the TDataSetAdapterNextRowAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 8040)	Called after the adapter action has executed.

	OnAfterGetResponse (see page 8040)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 8040)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 8041)	Called before the adapter action response is generated.
	OnGetParams (see page 8041)	Specifies name/value pairs needed to execute this adapter action.

41.1.23.4.1 TDataSetAdapterNextRowAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

41.1.23.4.2 TDataSetAdapterNextRowAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.23.4.3 TDataSetAdapterNextRowAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

41.1.23.4.4 TDataSetAdapterNextRowAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.23.4.5 TDataSetAdapterNextRowAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

41.1.24 TDataSetAdapterPrevRowAction Class

TDataSetAdapterPrevRowAction posts any edits made in an HTML form and moves the dataset adapter to the prior record.

Class Hierarchy

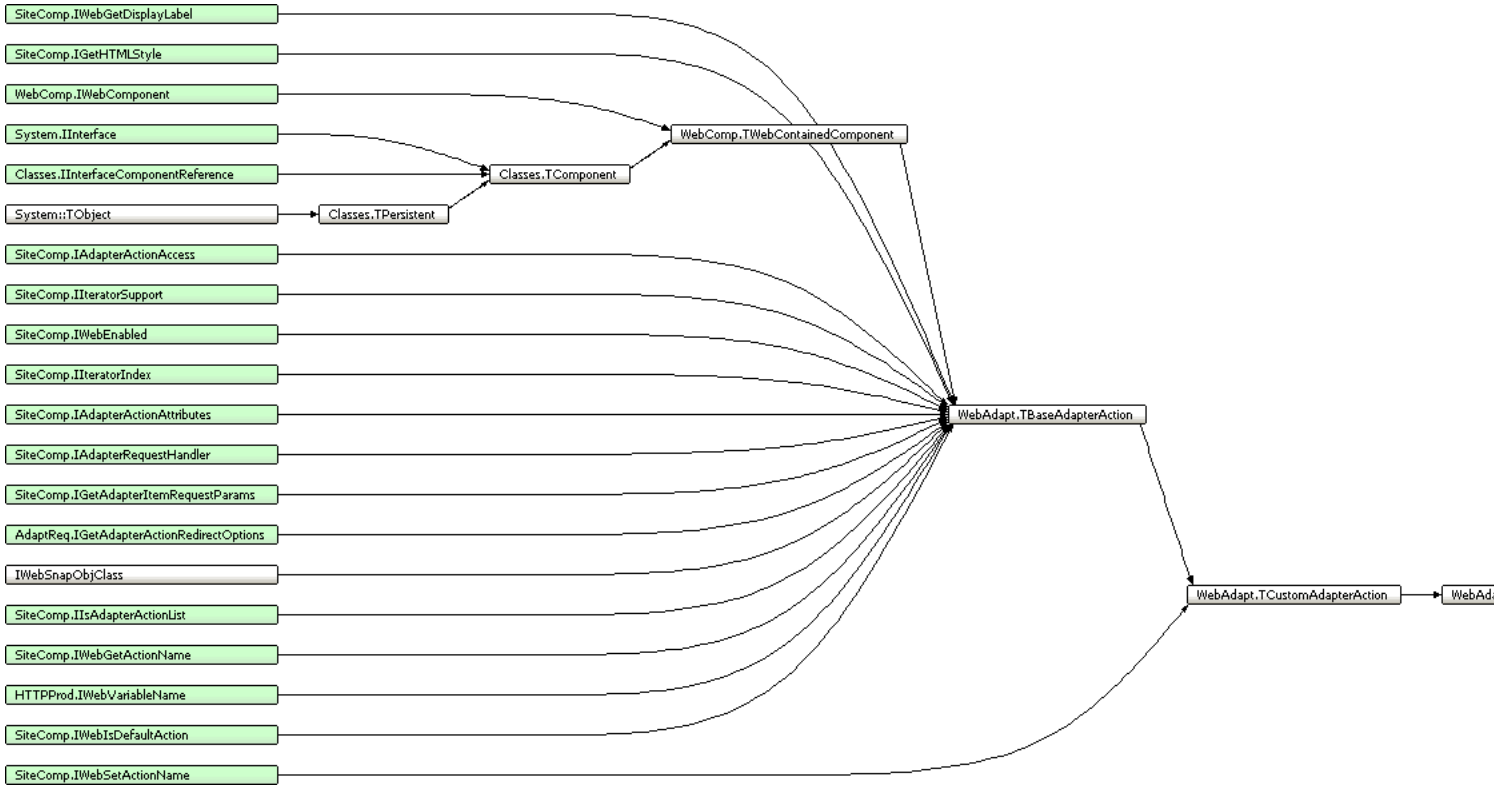

Pascal

```
TDataSetAdapterPrevRowAction = class(TDataSetAdapterMoveRowAction);
```

C++

```
class TDataSetAdapterPrevRowAction : public TDataSetAdapterMoveRowAction;
```

File

DBAdapt

Description

Add TDataSetAdapterPrevRowAction to a dataset adapter (TDataSetAdapter (see page 7872)) to support HTML forms that let users move to the previous dataset record. When this action is executed, it locates the parent adapter's dataset on the record specified by the current request object, posts any edits included in the current request, and then moves the dataset to the prior record.

Server-side script addresses this action as the 'PrevRow' action of the parent adapter. If the current request identifies the first record of the dataset as current, the control for this action appears disabled.

See Also

TDataSetAdapterApplyRowAction (see page 7890)

TDataSetAdapterBrowseRowAction (see page 7903)

TDataSetAdapterDeleteRowAction (see page 7934)

TDataSetAdapterEditRowAction (see page 7947)

TDataSetAdapterFirstRowAction (see page 7972)

TDataSetAdapterLastRowAction (see page 7984)

TDataSetAdapterCancelRowAction (see page 7916)

TDataSetAdapterNextRowAction (see page 8030)

TDataSetAdapterRefreshRowAction (see page 8054)

TDataSetAdapterNewRowAction (see page 8016)

TDataSetAdapter (see page 7872)

TCustomDataSetAdapter (see page 7766)

TDataSet (see page 6894)

41.1.24.1 TDataSetAdapterPrevRowAction Members

The following tables list the members exposed by TDataSetAdapterPrevRowAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 7819)	Returns the value of the Adapter (see page 7822) property.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.

	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7819)	Returns the value of the DataSet (see page 7822) property.
V	GetDefaultActionName (see page 8049)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
V	ImplExecuteActionRequest (see page 8049)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 7854)	Provides the underlying implementation for the GetAdapterItemRequestParams method.
V	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
V	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
V	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetVisible (see page 7819)	Provides the underlying implementation for the GetVisible method.
V	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.

	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 8016)	Provides the underlying implementation for the IsDefaultAction method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 8050)	Provides the underlying implementation for the WebEnabled method.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7822)	Indicates the dataset adapter that uses this object to represent one of its actions.
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 7822)	Occurs when server-side script executes this action.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 7823)	Occurs when the WebSnap application to determine whether a control or hyperlink for this action should be enabled.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 7818)	Creates and initializes an instance of TCustomDataSetAdapterAction (see page 7811).
V	Create (see page 7853)	Creates and initializes an instance of TCustomDataSetAdapterRowAction (see page 7846).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionModes (see page 7821)	Indicates what dataset adapter modes this field supports.
	ActionName (see page 7821)	Identifies the name of this action when using server-side script.
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 7822)	Indicates the dataset on which this action operates.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 8052)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 8052)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 8052)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 8053)	Called before the adapter action response is generated.
	OnGetParams (see page 8053)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 8050)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 8051)	Defines the rights required to execute this action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RedirectOptions (see page 8051)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.24.2 TDataSetAdapterPrevRowAction Methods

The methods of the TDataSetAdapterPrevRowAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 8049)	Gets the default name used to reference this adapter action with server-side script.
	ImplExecuteActionRequest (see page 8049)	Executes the adapter action using parameters passed in the HTTP request.
	ImplWebEnabled (see page 8050)	Provides the underlying implementation for the WebEnabled method.

41.1.24.2.1 TDataSetAdapterPrevRowAction.GetDefaultActionName Method

Gets the default name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultActionName();
```

Description

GetDefaultActionName returns the name of the adapter action component. Descendants may override this function.

See Also

ImplGetActionName (see page 24063)

41.1.24.2.2 TDataSetAdapterPrevRowAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest create a TStrings containing the name/value pairs in AActionRequest and calls the OnExecute event handler.

See Also

ImplExecuteActionRequest (see page 24063)

OnExecute (see page 24200)

41.1.24.2.3 TDataSetAdapterPrevRowAction.ImplWebEnabled Method

Provides the underlying implementation for the WebEnabled method.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

The WebEnabled method calls ImplWebEnabled to determine whether the control or hyperlink for this action should appear enabled in the HTML form. WebEnabled returns true if controls should be enabled, false otherwise.

ImplWebEnabled generates an OnGetEnabled event so that you can specify whether controls for this action are enabled. If there is no OnGetEnabled event handler, ImplWebEnabled returns true if the parent adapter's GetKeyFields method returns a set of fields that can locate records in the associated dataset.

See Also

WebEnabled (see page 24071)

OnGetEnabled (see page 7823)

GetKeyFields (see page 7786)

41.1.24.3 TDataSetAdapterPrevRowAction Properties

The properties of the TDataSetAdapterPrevRowAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 8050)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 8051)	Defines the rights required to execute this action.
	RedirectOptions (see page 8051)	Indicates whether to perform an HTTP redirect after successfully executing this action.

41.1.24.3.1 TDataSetAdapterPrevRowAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side

script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

41.1.24.3.2 TDataSetAdapterPrevRowAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property AnsiString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 25625)

hasexecuteaccess (see page 24162)

41.1.24.3.3 TDataSetAdapterPrevRowAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect (see page 15201)

41.1.24.4 TDataSetAdapterPrevRowAction Events

The events of the TDataSetAdapterPrevRowAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 8052)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 8052)	Called after the adapter action response is generated.

	OnBeforeExecute (see page 8052)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 8053)	Called before the adapter action response is generated.
	OnGetParams (see page 8053)	Specifies name/value pairs needed to execute this adapter action.

41.1.24.4.1 TDataSetAdapterPrevRowAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

41.1.24.4.2 TDataSetAdapterPrevRowAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.24.4.3 TDataSetAdapterPrevRowAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

41.1.24.4.4 TDataSetAdapterPrevRowAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.24.4.5 TDataSetAdapterPrevRowAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

41.1.25 TDataSetAdapterRefreshRowAction Class

TDataSetAdapterRefreshRowAction refetches the current field values for the dataset record specified in the current HTTP request.

Class Hierarchy

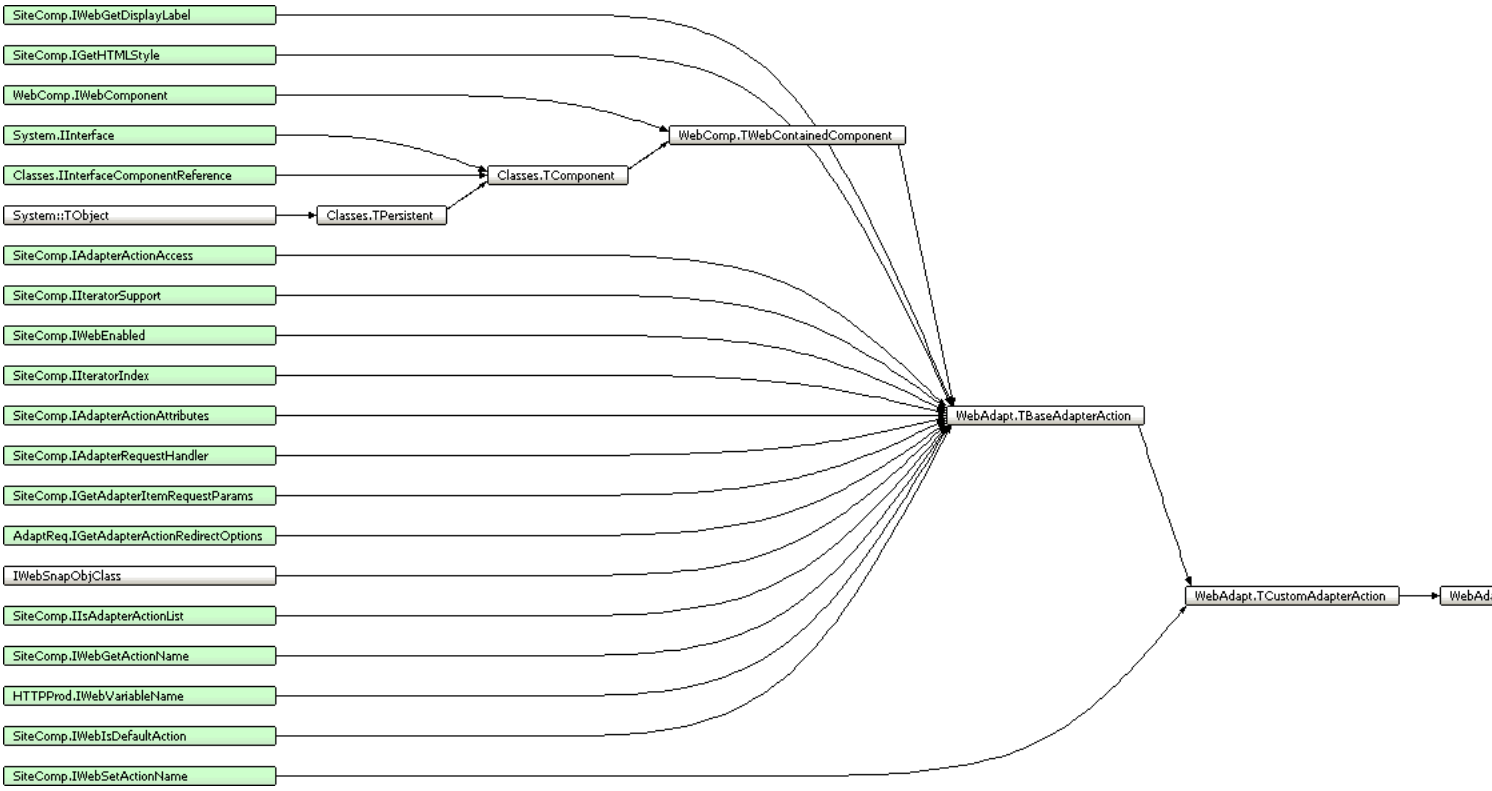

Pascal

```
TDataSetAdapterRefreshRowAction = class(TCustomDataSetAdapterRowAction);
```

C++

```
class TDataSetAdapterRefreshRowAction : public TCustomDataSetAdapterRowAction;
```

File

DBAdapt

Description

Add TDataSetAdapterRefreshRowAction to a dataset adapter (TDataSetAdapter (see page 7872)) to support HTML forms that let users re-fetch database information to reflect the current value in the associated dataset. When this action is executed, it sets the mode of the parent adapter from the value specified in the current request and positions the dataset on the record specified in the current request. This allows forms that include the adapter to represent the current values of the fields in the dataset. If the request specifies insert mode, this action first posts the new record to the dataset.

Server-side script addresses this action as the 'RefreshRow' action of the parent adapter. By default, this action is visible (included in HTML forms) regardless of the mode of the parent adapter. This action is included by default when the adapter is used to build forms that represent a single database record.

See Also

TDataSetAdapterApplyRowAction (see page 7890)

TDataSetAdapterBrowseRowAction (see page 7903)

TDataSetAdapterCancelRowAction (see page 7916)

TDataSetAdapterDeleteRowAction (see page 7934)

TDataSetAdapterFirstRowAction (see page 7972)

TDataSetAdapterLastRowAction (see page 7984)

TDataSetAdapterNextRowAction (see page 8030)

TDataSetAdapterPrevRowAction (see page 8042)

TDataSetAdapterEditRowAction (see page 7947)

TDataSetAdapterNewRowAction (see page 8016)

TDataSetAdapter (see page 7872)

TCustomDataSetAdapter (see page 7766)

TCustomDataSetAdapter (see page 7766)

41.1.25.1 TDataSetAdapterRefreshRowAction Members

The following tables list the members exposed by TDataSetAdapterRefreshRowAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 7819)	Returns the value of the Adapter (see page 7822) property.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.

	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7819)	Returns the value of the DataSet (see page 7822) property.
V	GetDefaultActionName (see page 8061)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
V	ImplExecuteActionRequest (see page 8061)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 7854)	Provides the underlying implementation for the GetAdapterItemRequestParams method.
V	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
V	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
V	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.

	ImplGetVisible (see page 7819)	Provides the underlying implementation for the GetVisible method.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 8062)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 8062)	Provides the underlying implementation for the WebEnabled method.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7822)	Indicates the dataset adapter that uses this object to represent one of its actions.
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 7822)	Occurs when server-side script executes this action.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 7823)	Occurs when the WebSnap application to determine whether a control or hyperlink for this action should be enabled.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 7818)	Creates and initializes an instance of TCustomDataSetAdapterAction (see page 7811).
V	Create (see page 7853)	Creates and initializes an instance of TCustomDataSetAdapterRowAction (see page 7846).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionModes (see page 7821)	Indicates what dataset adapter modes this field supports.
	ActionName (see page 7821)	Identifies the name of this action when using server-side script.
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 7822)	Indicates the dataset on which this action operates.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 8064)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 8064)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 8065)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 8065)	Called before the adapter action response is generated.
	OnGetParams (see page 8066)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 8063)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 8063)	Defines the rights required to execute this action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RedirectOptions (see page 8063)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.25.2 TDataSetAdapterRefreshRowAction Methods

The methods of the TDataSetAdapterRefreshRowAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 8061)	Gets the default name used to reference this adapter action with server-side script.
	ImplExecuteActionRequest (see page 8061)	Executes the adapter action using parameters passed in the HTTP request.
	ImplIsDefaultAction (see page 8062)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplWebEnabled (see page 8062)	Provides the underlying implementation for the WebEnabled method.

41.1.25.2.1 TDataSetAdapterRefreshRowAction.GetDefaultActionName Method

Gets the default name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultActionName();
```

Description

GetDefaultActionName returns the name of the adapter action component. Descendants may override this function.

See Also

ImplGetActionName (see page 24063)

41.1.25.2.2 TDataSetAdapterRefreshRowAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest create a TStrings containing the name/value pairs in AActionRequest and calls the OnExecute event

handler.

See Also

ImplExecuteActionRequest (see page 24063)

OnExecute (see page 24200)

41.1.25.2.3 TDataSetAdapterRefreshRowAction.ImplIsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function ImplIsDefaultAction(ADisplay: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplIsDefaultAction(TObject * ADisplay);
```

Description

ImplIsDefaultAction is called by IsDefaultAction. ImplIsDefaultAction returns true. ImplIsDefaultAction may be overridden by a descendant class.

See Also

IsDefaultAction (see page 24070)

ImplIsDefaultAction (see page 24243)

41.1.25.2.4 TDataSetAdapterRefreshRowAction.ImplWebEnabled Method

Provides the underlying implementation for the WebEnabled method.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

The WebEnabled method calls ImplWebEnabled to determine whether the control or hyperlink for this action should appear enabled in the HTML form. WebEnabled returns true if controls should be enabled, false otherwise.

ImplWebEnabled generates an OnGetEnabled event so that you can specify whether controls for this action are enabled. If there is no OnGetEnabled event handler, ImplWebEnabled returns true if the parent adapter's GetKeyFields method returns a set of fields that can locate records in the associated dataset.

See Also

WebEnabled (see page 24071)

OnGetEnabled (see page 7823)

GetKeyFields (see page 7786)

41.1.25.3 TDataSetAdapterRefreshRowAction Properties

The properties of the TDataSetAdapterRefreshRowAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 8063)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 8063)	Defines the rights required to execute this action.
	RedirectOptions (see page 8063)	Indicates whether to perform an HTTP redirect after successfully executing this action.

41.1.25.3.1 TDataSetAdapterRefreshRowAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

41.1.25.3.2 TDataSetAdapterRefreshRowAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property AnsiString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 25625)

hasexecuteaccess (see page 24162)

41.1.25.3.3 TDataSetAdapterRefreshRowAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect (see page 15201)

41.1.25.4 TDataSetAdapterRefreshRowAction Events

The events of the TDataSetAdapterRefreshRowAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 8064)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 8064)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 8065)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 8065)	Called before the adapter action response is generated.
	OnGetParams (see page 8066)	Specifies name/value pairs needed to execute this adapter action.

41.1.25.4.1 TDataSetAdapterRefreshRowAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

41.1.25.4.2 TDataSetAdapterRefreshRowAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.25.4.3 TDataSetAdapterRefreshRowAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

41.1.25.4.4 TDataSetAdapterRefreshRowAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.25.4.5 TDataSetAdapterRefreshRowAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

41.1.26 TDataSetAdapterRowAction Class

TDataSetAdapterRowAction is a general-purpose action for dataset adapters.

Class Hierarchy

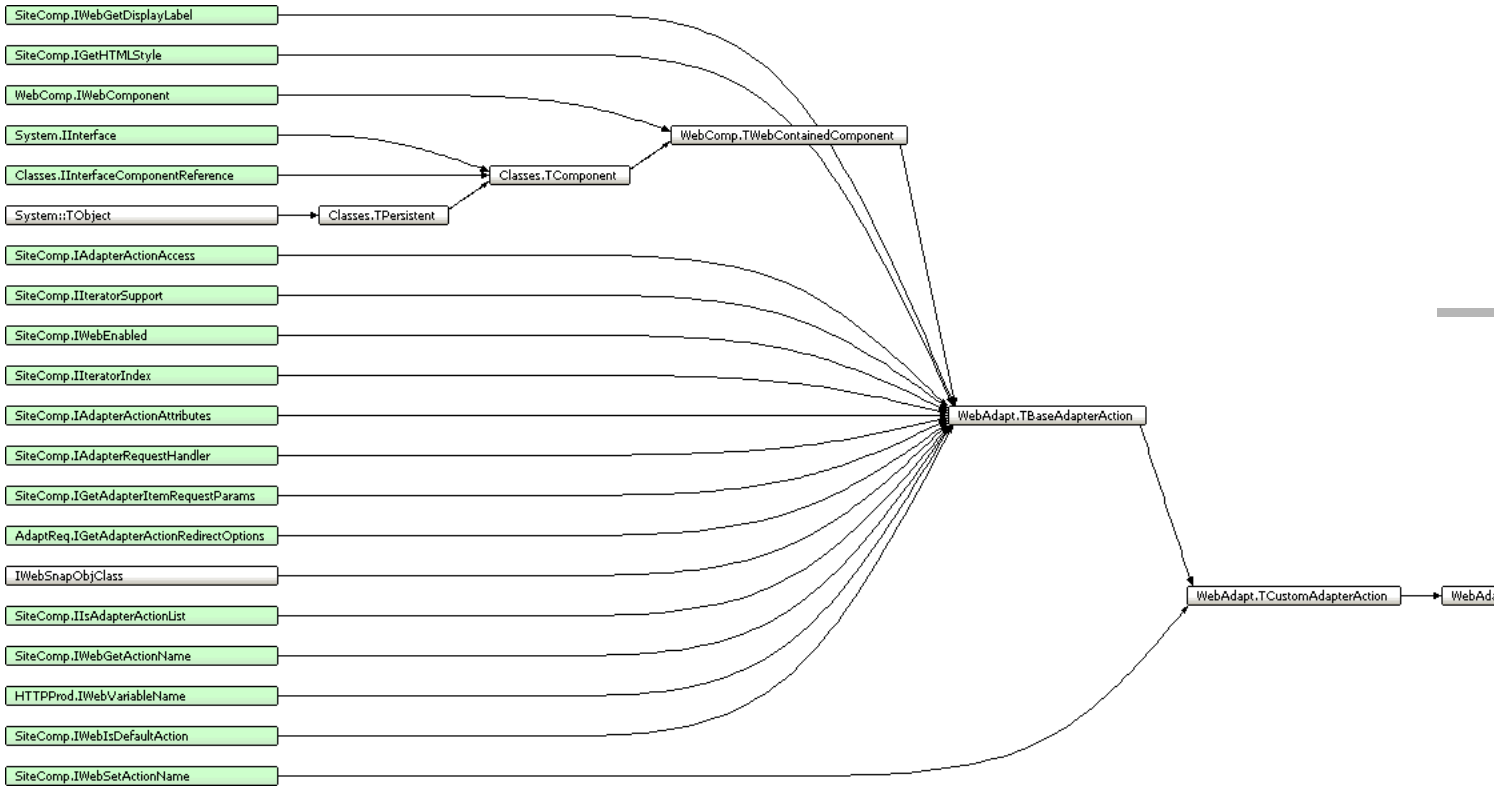

Pascal

```
TDataSetAdapterRowAction = class(TCustomDataSetAdapterRowAction);
```

C++

```
class TDataSetAdapterRowAction : public TCustomDataSetAdapterRowAction;
```

File

DBAdapt

Description

Add TDataSetAdapterRowAction to a dataset adapter (TDataSetAdapter (see page 7872)) to support your own custom actions that operate on the adapter's dataset. When this action is executed, it generates an OnExecute (see page 7822) event. You can supply an event handler that provides the implementation for this action.

By default, this action is visible (included in HTML forms) unless the parent adapter is in query mode. You can change when the action is visible by setting the ActionModes (see page 8075) property.

See Also

TDataSetAdapterApplyRowAction (see page 7890)

TDataSetAdapterBrowseRowAction (see page 7903)

TDataSetAdapterCancelRowAction (see page 7916)

TDataSetAdapterDeleteRowAction (see page 7934)

TDataSetAdapterFirstRowAction (see page 7972)

TDataSetAdapterLastRowAction (see page 7984)

TDataSetAdapterNextRowAction (see page 8030)

TDataSetAdapterPrevRowAction (see page 8042)

TDataSetAdapterEditRowAction (see page 7947)

TDataSetAdapterNewRowAction (see page 8016)

TDataSetAdapter (see page 7872)

TDataSetAdapterRefreshRowAction (see page 8054)

41.1.26.1 TDataSetAdapterRowAction Members

The following tables list the members exposed by TDataSetAdapterRowAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 7819)	Returns the value of the Adapter (see page 7822) property.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7819)	Returns the value of the DataSet (see page 7822) property.
V	GetDefaultActionName (see page 24055)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.

	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 8074)	Provides the underlying implementation for the ExecuteActionRequest method.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 8074)	Provides the underlying implementation for the GetAdapterItemRequestParams method.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 7819)	Provides the underlying implementation for the GetVisible method.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 7820)	Provides the underlying implementation for the WebEnabled method.
	ImplWebEnabled (see page 7855)	Provides the underlying implementation for the WebEnabled method.
	ImplWebEnabled (see page 24069)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

V	ImplWebEnabled (see page 24197)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7822)	Indicates the dataset adapter that uses this object to represent one of its actions.
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 7822)	Occurs when server-side script executes this action.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 7823)	Occurs when the WebSnap application to determine whether a control or hyperlink for this action should be enabled.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 7818)	Creates and initializes an instance of TCustomDataSetAdapterAction (see page 7811).
V	Create (see page 7853)	Creates and initializes an instance of TCustomDataSetAdapterRowAction (see page 7846).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionName (see page 7821)	Identifies the name of this action when using server-side script.
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 7822)	Indicates the dataset on which this action operates.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 8077)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 8077)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 8078)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 8078)	Called before the adapter action response is generated.
	OnGetParams (see page 8078)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	ActionModes (see page 8075)	Indicates what dataset adapter modes this field supports.
	DisplayLabel (see page 8076)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 8076)	Defines the rights required to execute this action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RedirectOptions (see page 8076)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Tag (see page 2808)	Stores an integer value as part of a component.

41.1.26.2 TDataSetAdapterRowAction Methods

The methods of the TDataSetAdapterRowAction class are listed here.

Protected Methods

	Name	Description
	ImplExecuteActionRequest (see page 8074)	Provides the underlying implementation for the ExecuteActionRequest method.
	ImplGetAdapterItemRequestParams (see page 8074)	Provides the underlying implementation for the GetAdapterItemRequestParams method.

41.1.26.2.1 TDataSetAdapterRowAction.ImplExecuteActionRequest Method

Provides the underlying implementation for the ExecuteActionRequest method.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

The ExecuteActionRequest method calls ImplExecuteActionRequest to execute this action in response to server-side script.

AActionRequest is an interface to the current request object. ImplExecuteActionRequest uses this to obtain information about the current request (such as the values of request parameters).

AActionResponse is an interface to the current response object. ImplExecuteActionRequest uses this to indicate the status of the emerging HTTP response.

ImplExecuteActionRequest generates an OnExecute (see page 7822) event, allowing you to provide the implementation for this action in an OnExecute (see page 7822) event handler. If you do not supply an OnExecute (see page 7822) event handler, ImplExecuteActionRequest calls the parent adapter's LocateAndApply method, to position the adapter's dataset on the record indicated by the current request and apply any updates that the request contains.

See Also

ExecuteActionRequest (see page 24053)

OnExecute (see page 7822)

LocateAndApply (see page 7796)

41.1.26.2.2 TDataSetAdapterRowAction.ImplGetAdapterItemRequestParams Method

Provides the underlying implementation for the GetAdapterItemRequestParams method.

Pascal

```
procedure ImplGetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams); override;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(AnsiString AIdentifier,
IAdapterItemRequestParams Params);
```

Description

The GetAdapterItemRequestParams method calls ImplGetAdapterItemRequestParams to obtain the parameters this action needs when it executes. The name/value pairs this method provides are added to the HTML form and passed back in any HTTP request that form generates.

AIdentifier returns a unique identifier for this action object, which the WebSnap application uses to dispatch requests from the generated form to this object.

Params is an interface that ImplGetAdapterItemRequestParams uses to write the name/value pairs it needs.

ImplGetAdapterItemRequestParams first generates an OnGetActionParams event on the parent adapter, and then an OnGetParams (see page 8078) event on this action object.

See Also

GetAdapterItemRequestParams (see page 24055)

OnGetParams (see page 24077)

OnGetActionParams (see page 24187)

41.1.26.3 TDataSetAdapterRowAction Properties

The properties of the TDataSetAdapterRowAction class are listed here.

Published Properties

	Name	Description
	ActionModes (see page 8075)	Indicates what dataset adapter modes this field supports.
	DisplayLabel (see page 8076)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 8076)	Defines the rights required to execute this action.
	RedirectOptions (see page 8076)	Indicates whether to perform an HTTP redirect after successfully executing this action.

41.1.26.3.1 TDataSetAdapterRowAction.ActionModes Property

Indicates what dataset adapter modes this field supports.

Pascal

```
property ActionModes: TDataSetAdapterModes;
```

C++

```
__property TDataSetAdapterModes ActionModes;
```

Description

ActionModes is a set of all the modes in which this adapter action can be used. If the parent dataset adapter is in a mode that is not included in this set, any forms generated using that adapter do not include a control for executing this action. Thus, for example, there is no sense in adding a control that applies edits to a form that presents a read-only view of the data. An apply action, then, should not include amBrowse in its action modes.

See Also

CanModify (see page 7154)

Adapter (see page 7752)

Mode (see page 7810)

GetDisplayStyle (see page 7739)

GetInputStyle (see page 7741)

GetViewMode (see page 7744)

41.1.26.3.2 TDataSetAdapterRowAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

41.1.26.3.3 TDataSetAdapterRowAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property AnsiString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 25625)

hasexecuteaccess (see page 24162)

41.1.26.3.4 TDataSetAdapterRowAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect (see page 15201)

41.1.26.4 TDataSetAdapterRowAction Events

The events of the TDataSetAdapterRowAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 8077)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 8077)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 8078)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 8078)	Called before the adapter action response is generated.
	OnGetParams (see page 8078)	Specifies name/value pairs needed to execute this adapter action.

41.1.26.4.1 TDataSetAdapterRowAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

41.1.26.4.2 TDataSetAdapterRowAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.26.4.3 TDataSetAdapterRowAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

41.1.26.4.4 TDataSetAdapterRowAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

41.1.26.4.5 TDataSetAdapterRowAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

41.1.27 TDataSetValuesList Class

TDataSetValuesList is an adapter component that provides a list of name/value pairs by retrieving values from a TDataSet.

Class Hierarchy

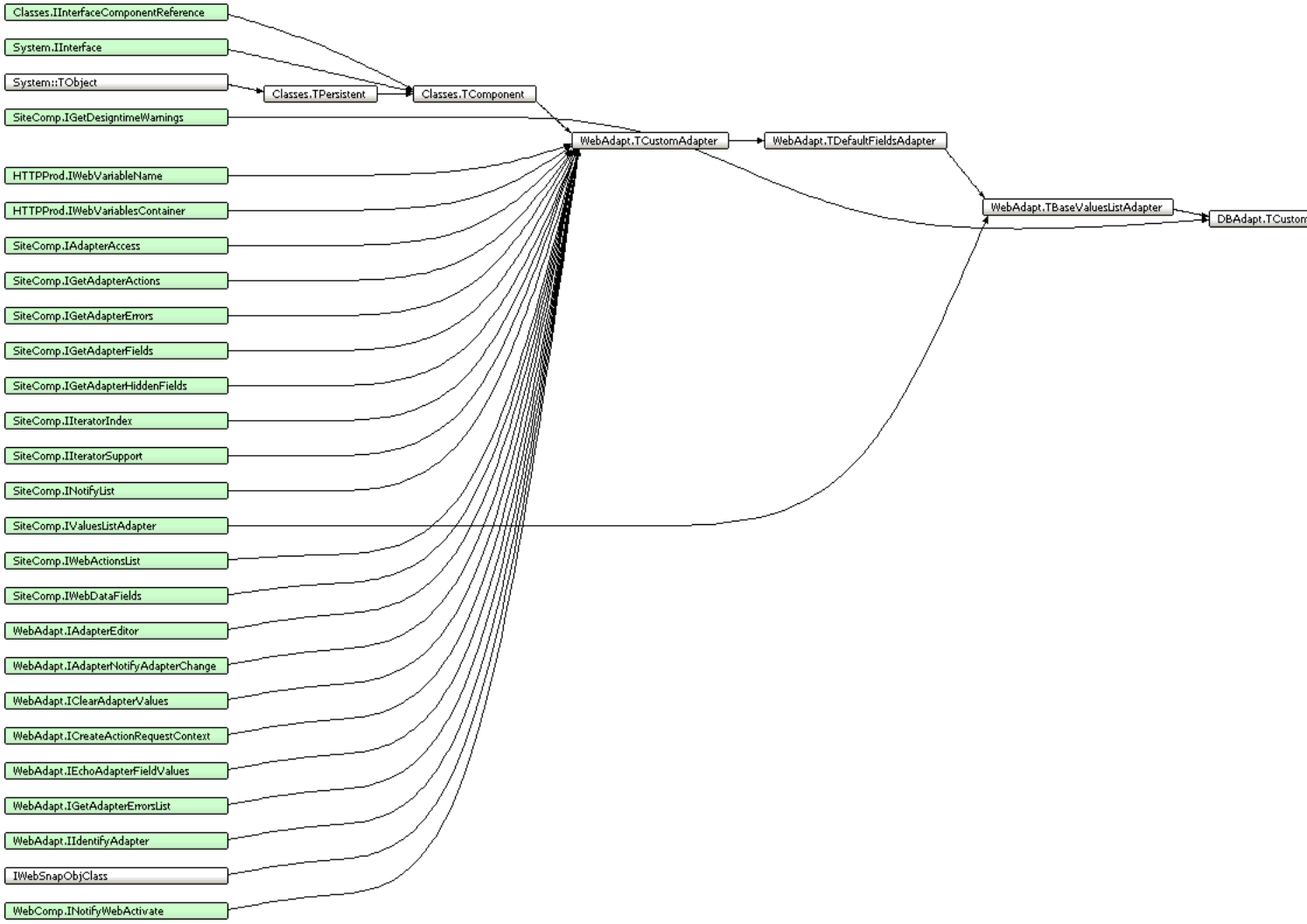

Pascal

```
TDataSetValuesList = class(TCustomDataSetValuesList);
```

C++

```
class TDataSetValuesList : public TCustomDataSetValuesList;
```

File

DBAdapt

Description

Use TDataSetValuesList to define a list of possible values for an adapter field by assigning the TDataSetValuesList component to the ValuesList property of the adapter field. Typically, the values are used to populate HTML elements such as a select list or radio buttons when generating HTML to edit an adapter field. The TAdapterPageProducer automatically generates appropriate HTML/JavaScript for an editable field when the ValuesList is set.

41.1.27.1 TDataSetValuesList Members

The following tables list the members exposed by TDataSetValuesList.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.

	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetDesigntimeWarnings (see page 7864)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetListImage (see page 24128)	Returns an image associated with the current name/value pair.
	GetListImageOfValue (see page 24128)	Looks up an image for a value.
	GetListName (see page 24128)	Responsible returning the name of the name/value pair.
	GetListNameOfValue (see page 24128)	Looks up a name for a value.
	GetListValue (see page 24129)	Responsible for returning the value of the name/value pair.
V	GetNameFieldClass (see page 7865)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
V	GetNameFieldClass (see page 24129)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	GetValueFieldClass (see page 7865)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
V	GetValueFieldClass (see page 24129)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
V	GetWebSnapObjClass (see page 24130)	Returns the class type for this adapter field.
V	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 24130)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
V	ImplEndIterator (see page 7866)	Provides the underlying implementation for the EndIterator method.
V	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.

	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 24130)	Adds default field components to this adapter.
	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetListImage (see page 24131)	Responsible returning an image associated with the current name/value pair.
	ImplGetListImageOfValue (see page 24131)	Looks up an image for a value.
	ImplGetListName (see page 7866)	Provides the underlying implementation for the GetListName method.
	ImplGetListName (see page 24131)	Responsible returning the name of the name/value pair.
	ImplGetListNameOfValue (see page 7867)	Provides the underlying implementation for the GetListNameOfValue method.
	ImplGetListNameOfValue (see page 24132)	Looks up a name for a value.
	ImplGetListValue (see page 7867)	Provides the underlying implementation for the GetListValue method.
	ImplGetListValue (see page 24132)	Responsible for returning the value of the name/value pair.
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 7867)	Provides the underlying implementation for the NextIteration method.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 7868)	Provides the underlying implementation for the NotifyDeactivate method.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 7868)	Provides the underlying implementation for the StartIterator method.
	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7869)	Responds when components are created or destroyed.
	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.

	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DataSetPrepared (see page 7870)	Indicates whether the OnPrepareDataSet (see page 7871) event has occurred.
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 24184)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24185)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24186)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24186)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.

	OnGetActionParams (see page 24187)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 7863)	Creates and initializes an instance of TCustomDataSetValuesList (see page 7855).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 7864)	Destroys an instance of TCustomDataSetValuesList (see page 7855).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.

	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetDesigntimeWarnings (see page 19697)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
	GetListImage (see page 19741)	Returns the image object associated with the current name/value pair.
	GetListImageOfValue (see page 19742)	Returns the image object associated with a specified value.
	GetListName (see page 19742)	Returns the name part of the current name/value pair.
	GetListNameOfValue (see page 19742)	Returns the name associated with a specified value.
	GetListValue (see page 19743)	Returns the value of the current name/value pair.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnPrepareDataSet (see page 8090)	Occurs before the dataset values list uses its associated dataset.

Published Properties

	Name	Description
	Actions (see page 8088)	Container for adapter actions components.
	Data (see page 8088)	Container for adapter field components.
	DataSet (see page 8088)	Specifies the dataset that stores the names and values for this list.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NameField (see page 8089)	Indicates the field in the associated dataset that supplies the name part of this list's name/value pairs.
	Tag (see page 2808)	Stores an integer value as part of a component.
	ValueField (see page 8089)	Indicates the field in the associated dataset that supplies the value part of this list's name/value pairs.

41.1.27.2 TDataSetValuesList Properties

The properties of the TDataSetValuesList class are listed here.

Published Properties

	Name	Description
	Actions (see page 8088)	Container for adapter actions components.
	Data (see page 8088)	Container for adapter field components.
	DataSet (see page 8088)	Specifies the dataset that stores the names and values for this list.
	NameField (see page 8089)	Indicates the field in the associated dataset that supplies the name part of this list's name/value pairs.
	ValueField (see page 8089)	Indicates the field in the associated dataset that supplies the value part of this list's name/value pairs.

41.1.27.2.1 TDataSetValuesList.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 23770)

41.1.27.2.2 TDataSetValuesList.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 23778)

41.1.27.2.3 TDataSetValuesList.DataSet Property

Specifies the dataset that stores the names and values for this list.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Use DataSet to specify the dataset from which this adapter gets its name/value pairs. The names of the name/value pairs in this list are the values of the field that the NameField property specifies. The values are the corresponding values of the field that the ValueField property specifies.

Note: When the WebSnap application reads the value of DataSet for the first time, TCustomDataSetValuesList generates an OnPrepareDataSet event. This lets you perform any initializations that ensure the dataset contains the desired data.

See Also

NameField (see page 7870)

ValueField (see page 7871)

OnPrepareDataSet (see page 7871)

41.1.27.2.4 TDataSetValuesList.NameField Property

Indicates the field in the associated dataset that supplies the name part of this list's name/value pairs.

Pascal

```
property NameField: string;
```

C++

```
__property AnsiString NameField;
```

Description

Set NameField to specify the dataset field that holds the names of this adapter's name/value pairs. Each entry in TCustomDataSetValuesList is a name/value pair that corresponds to the values of the NameField and ValueField fields of the associated dataset for a single record in the dataset. NameField is the same as the FieldName property of the dataset field that supplies the names.

See Also

DataSet (see page 7869)

FieldName (see page 7164)

ValueField (see page 7871)

41.1.27.2.5 TDataSetValuesList.ValueField Property

Indicates the field in the associated dataset that supplies the value part of this list's name/value pairs.

Pascal

```
property ValueField: string;
```

C++

```
__property AnsiString ValueField;
```

Description

Set ValueField to specify the dataset field that holds the values of this adapter's name/value pairs. Each entry in TCustomDataSetValuesList is a name/value pair that corresponds to the values of the NameField and ValueField fields of the associated dataset for a single record in the dataset. ValueField is the same as the FieldName property of the dataset field that supplies the values.

See Also

DataSet (see page 7869)

FieldName (see page 7164)

NameField (see page 7870)

41.1.27.3 TDataSetValuesList Events

The events of the TDataSetValuesList class are listed here.

Published Events

	Name	Description
	OnPrepareDataSet (see page 8090)	Occurs before the dataset values list uses its associated dataset.

41.1.27.3.1 TDataSetValuesList.OnPrepareDataSet Event

Occurs before the dataset values list uses its associated dataset.

Pascal

```
property OnPrepareDataSet: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPrepareDataSet;
```

Description

Write an OnPrepareDataSet event handler to prepare the associated dataset before this component fetches names and values from it. This lets you customize the data for each incoming request. For example, you can use this event to set query parameters based on parameters in the current HTTP request and then execute the query. (Use the global Web context to obtain information about the current HTTP request.)

TCustomDataSetValuesList generates an OnPrepareDataSet event the first time the application reads the DataSet property for a new HTTP request.

See Also

DataSet (see page 7869)

WebContext (see page 24687)

41.1.28 TLocateParams Class

TLocateParams is a collection of the key fields and their corresponding values that can be used to locate a record of a dataset adapter.

Class Hierarchy

System::TObject → DBAdapt.TLocateParams

Pascal

```
TLocateParams = class(TObject);
```

C++

```
class TLocateParams : public TObject;
```

File

DBAdapt

Description

TLocateParams is a collection of the key fields belonging to a single dataset adapter, along with their values. Each fieldname/value pair is stored as a string of the form Name=Value. TLocateParams provides access to these name/value pairs using the Items (see page 8095) property, or to the constituent names and values using the ParamNames (see page 8095) and ParamValues (see page 8096) properties.

Use the AdapterName (see page 8094) property to identify the dataset adapter whose key fields and values are in the collection. You can add new key field/value pairs by calling the AddParam (see page 8093) method.

See Also

TLocateParamsList (see page 8096)

TDataSetAdapter (see page 7872)

41.1.28.1 TLocateParams Members

The following tables list the members exposed by TLocateParams.

Public Methods

	Name	Description
	AddParam (see page 8093)	This is the overview for the AddParam method overload.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 8092)	Creates an initializes an instance of TLocateParams (see page 8090).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 8092)	Frees an instance of TLocateParams (see page 8090).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	AdapterName (see page 8094)	Specifies the name of the dataset adapter to which these key fields and values belong.
R	Count (see page 8094)	Indicates the number of fieldname/value pairs in the collection
R	Items (see page 8095)	Lists the key field names and values in this collection.
R	ParamNames (see page 8095)	Lists the key field names in this collection.
R	ParamValues (see page 8096)	Lists the key field values in this collection.

41.1.28.2 TLocateParams.Create Constructor

Creates an initializes an instance of TLocateParams (see page 8090).

Pascal

```
constructor Create(AList: TLocateParamsList);
```

C++

```
__fastcall TLocateParams(TLocateParamsList AList);
```

Description

WebSnap applications should not need to create instances of TLocateParams (see page 8090). These objects are created using the Add method of a TLocateParamsList (see page 8096) object.

AList is the TLocateParamsList (see page 8096) object that creates and stores the TLocateParams (see page 8090) instance.

See Also

Add (see page 8099)

41.1.28.3 TLocateParams.Destroy Destructor

Frees an instance of TLocateParams (see page 8090).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TLocateParams();
```

Description

WebSnap applications should not need to free instances of TLocateParams (see page 8090), as this is handled by the

TLocateParamsList (see page 8096) objects that create them. If you must free a TLocateParams (see page 8090) instance, use the Free method.

See Also

Free (see page 22300)

41.1.28.4 TLocateParams Methods

The methods of the TLocateParams class are listed here.

Public Methods

	Name	Description
	AddParam (see page 8093)	This is the overview for the AddParam method overload.

41.1.28.4.1 AddParam Method

This is the overview for the AddParam method overload.

Overload List

	Name	Description
	TLocateParams.AddParam (string) (see page 8093)	Adds a new key field name and value to the list.
	TLocateParams.AddParam (string, string) (see page 8093)	Adds a new key field name and value to the list.

41.1.28.4.1.1 TLocateParams.AddParam Method (string)

Adds a new key field name and value to the list.

Pascal

```
procedure AddParam(const ANameValue: string); overload;
procedure AddParam(const AName: string; const AValue: string); overload;
```

C++

```
__fastcall AddParam(const AnsiString ANameValue);
__fastcall AddParam(const AnsiString AName, const AnsiString AValue);
```

Description

Call AddParam to add a new key field name and value to the list. There are two forms of the AddParam method. The first syntax lets you supply the key field name and value as separate parameters. The second syntax lets you supply them as a string of the form 'Name=Value'.

AName is the name of the key field. This becomes the value of the ParamNames (see page 8095) property for the new entry.

AValue is the name of the key field's value. This becomes the value of the ParamValues (see page 8096) property for the new entry.

ANameValue is a string of the form 'Name=Value', where Name is the key field name and Value is its value. This becomes the value of the Items (see page 8095) property for the new entry.

See Also

Items (see page 8095)

ParamNames (see page 8095)

ParamValues (see page 8096)

41.1.28.5 TLocateParams Properties

The properties of the TLocateParams class are listed here.

Public Properties

	Name	Description
	AdapterName (see page 8094)	Specifies the name of the dataset adapter to which these key fields and values belong.
R	Count (see page 8094)	Indicates the number of fieldname/value pairs in the collection
R	Items (see page 8095)	Lists the key field names and values in this collection.
R	ParamNames (see page 8095)	Lists the key field names in this collection.
R	ParamValues (see page 8096)	Lists the key field values in this collection.

41.1.28.5.1 TLocateParams.AdapterName Property

Specifies the name of the dataset adapter to which these key fields and values belong.

Pascal

```
property AdapterName: string;
```

C++

```
__property AnsiString AdapterName;
```

Description

AdapterName identifies the dataset adapter that uses these key fields and value to locate a dataset record. This value is the same as the value of the dataset adapter's Name property.

See Also

TDataSetAdapter (see page 7872)

Name (see page 2805)

41.1.28.5.2 TLocateParams.Count Property

Indicates the number of fieldname/value pairs in the collection

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Count indicates the number of entries in the Items (see page 8095), ParamNames (see page 8095), or ParamValues (see page 8096) property array. Use this value as an upper bound for the index when iterating over all the fieldname/value pairs in this collection.

See Also

Items (see page 8095)

ParamNames (see page 8095)

ParamValues (see page 8096)

41.1.28.5.3 TLocateParams.Items Property

Lists the key field names and values in this collection.

Pascal

```
property Items [I: Integer]: string;
```

C++

```
__property AnsiString Items[int I];
```

Description

Use Items to obtain a key field name and value entry from the list. Each item is a string of the form 'Name=Value', where 'Name' is the name of a key field and 'Value' is its value on the identified record.

I is the index of the key field and value, where 0 specifies the first item, 1 specifies the second item, and so on. The Count (see page 8094) property gives an upper bound for this index.

Note: In Delphi, items is the default property of TLocateParams (see page 8090). That means the property name may be omitted in code. For example, the lines

Note: List := DataSetAdapter1.LocateParams[0];

Note: Param := List.Items[0];

Note: can also be written

Note: List := DataSetAdapter1.LocateParams[0];

Note: Param := List[0];

Note: Items provides indexed access to the entire 'Name=Value' string. To access just the key field name or just the value, use the ParamNames (see page 8095) or ParamValues (see page 8096) property instead.

See Also

Count (see page 8094)

ParamNames (see page 8095)

ParamValues (see page 8096)

operator_sb

41.1.28.5.4 TLocateParams.ParamNames Property

Lists the key field names in this collection.

Pascal

```
property ParamNames [I: Integer]: string;
```

C++

```
__property AnsiString ParamNames[int I];
```

Description

Use ParamNames to obtain a key field name from the list. This is the name of a key field of the dataset adapter.

I is the index of the key field name, where 0 specifies the first name, 1 specifies the second name, and so on. The Count (see page 8094) property gives an upper bound for this index.

Note: ParamNames lists only the names of the key fields. To obtain the corresponding values, use the ParamValues (see page 8096) property instead. To access the entire 'Name=Value' string, use the Items (see page 8095) property instead.

See Also

Count (see page 8094)

Items (see page 8095)

ParamValues (see page 8096)

41.1.28.5.5 TLocateParams.ParamValues Property

Lists the key field values in this collection.

Pascal

```
property ParamValues [I: Integer]: string;
```

C++

```
__property AnsiString ParamValues[int I];
```

Description

Use ParamValues to obtain a key field value from the list. This is the value of a key field of the dataset adapter.

I is the index of the key field value, where 0 specifies the first value, 1 specifies the second value, and so on. The Count (see page 8094) property gives an upper bound for this index.

Note: ParamValues lists only the values of the key fields. To obtain the corresponding key field names, use the ParamNames (see page 8095) property instead. To access the entire 'Name=Value' string, use the Items (see page 8095) property instead.

See Also

Count (see page 8094)

Items (see page 8095)

ParamNames (see page 8095)

41.1.29 TLocateParamsList Class

TLocateParamsList is a collection of TLocateParams (see page 8090) objects.

Class Hierarchy

System::TObject → DBAdapt.TLocateParamsList

Pascal

```
TLocateParamsList = class(TObject);
```

C++

```
class TLocateParamsList : public TObject;
```

File

DBAdapt

Description

TLocateParams (see page 8090) is a collection of TLocateParams (see page 8090) objects, where each TLocateParams (see page 8090) object lists the key fields and their values for a single dataset adapter. The entire collection of TLocateParams (see page 8090) gives the key fields and their values for a set of dataset adapters that are linked together in master/detail relationships.

TLocateParams (see page 8090) is useful for locating records of a dataset adapter. Because the dataset adapters are linked in master/detail relationships, it is impossible to locate a record in a detail dataset without first positioning the master datasets on their corresponding records.

See Also

TLocateParams (see page 8090)

TDataSetAdapter (see page 7872)

41.1.29.1 TLocateParamsList Members

The following tables list the members exposed by TLocateParamsList.

Public Methods

	Name	Description
	Add (see page 8099)	Adds a TLocateParams (see page 8090) object to the collection.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 8099)	Empties the TLocateParamsList (see page 8096).
	Create (see page 8098)	Creates an initializes an instance of TLocateParamsList (see page 8096).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 8098)	Frees an instance of TLocateParamsList (see page 8096).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	ItemByAdapterName (see page 8100)	Returns the TLocateParams (see page 8090) object associated with a specified dataset adapter.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Count (see page 8100)	Indicates the number of TLocateParams (see page 8090) objects in the collection.
R	Items (see page 8101)	Lists the key field/value lists for all the adapters that use this object to store their locate parameters.

41.1.29.2 TLocateParamsList.Create Constructor

Creates an initializes an instance of TLocateParamsList (see page 8096).

Pascal

```
constructor Create;
```

C++

```
__fastcall TLocateParamsList();
```

Description

WebSnap applications should not need to create instances of TLocateParamsList (see page 8096). DataSet adapters create these objects to collect information for locating their current record.

See Also

TDataSetAdapter (see page 7872)

41.1.29.3 TLocateParamsList.Destroy Destructor

Frees an instance of TLocateParamsList (see page 8096).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TLocateParamsList();
```

Description

WebSnap applications should not need to free instances of TLocateParamsList (see page 8096), as this is handled by the dataset adapters that create them. If you must free a TLocateParamsList (see page 8096) instance, use the Free method.

See Also

Free (see page 22300)

41.1.29.4 TLocateParamsList Methods

The methods of the TLocateParamsList class are listed here.

Public Methods

	Name	Description
	Add (see page 8099)	Adds a TLocateParams (see page 8090) object to the collection.
	Clear (see page 8099)	Empties the TLocateParamsList (see page 8096).
	ItemByAdapterName (see page 8100)	Returns the TLocateParams (see page 8090) object associated with a specified dataset adapter.

41.1.29.4.1 TLocateParamsList.Add Method

Adds a TLocateParams (see page 8090) object to the collection.

Pascal

```
function Add: TLocateParams;
```

C++

```
__fastcall TLocateParams Add();
```

Description

Call Add to insert a new TLocateParams (see page 8090) object into this collection.

Add returns the newly-inserted TLocateParams (see page 8090) object. The returned object is an empty list that is not yet associated with any dataset adapter. Set the returned object's AdapterName property to associate it with a dataset adapter, and then fill it by calling its AddParam method for every key field of that adapter.

See Also

Items (see page 8101)

Clear (see page 8099)

AdapterName (see page 8094)

AddParam (see page 8093)

41.1.29.4.2 TLocateParamsList.Clear Method

Empties the TLocateParamsList (see page 8096).

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the Items (see page 8101) property array.

See Also

Items (see page 8101)

Add (see page 8099)

41.1.29.4.3 TLocateParamsList.ItemByAdapterName Method

Returns the TLocateParams (see page 8090) object associated with a specified dataset adapter.

Pascal

```
function ItemByAdapterName(const AName: string): TLocateParams;
```

C++

```
__fastcall TLocateParams ItemByAdapterName(const AnsiString AName);
```

Description

Call ItemByAdapterName to access the list of key fields and values for a specified dataset adapter. ItemByAdapterName searches the Items (see page 8101) property array for an item whose AdapterName property matches the specified name.

AName is the name of the dataset adapter whose key fields you want to access. This is the same as the Name property of the TDataSetAdapter (see page 7872) object and the AdapterName property of the TLocateParams (see page 8090) object that holds its key fields and values.

ItemByAdapterName returns the TLocateParams (see page 8090) object for the specified adapter. If this TLocateParamsList (see page 8096) object does not include an item for the specified adapter, ItemByAdapterName returns nil (Delphi) or NULL (C++).

See Also

Items (see page 8101)

Name (see page 2805)

TDataSetAdapter (see page 7872)

AdapterName (see page 8094)

41.1.29.5 TLocateParamsList Properties

The properties of the TLocateParamsList class are listed here.

Public Properties

	Name	Description
R	Count (see page 8100)	Indicates the number of TLocateParams (see page 8090) objects in the collection.
R	Items (see page 8101)	Lists the key field/value lists for all the adapters that use this object to store their locate parameters.

41.1.29.5.1 TLocateParamsList.Count Property

Indicates the number of TLocateParams (see page 8090) objects in the collection.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Count indicates the number of entries in the Items (see page 8101) property array. Use this value as an upper bound for the index when iterating over all the key field/value lists in this collection.

See Also

Items (see page 8101)

41.1.29.5.2 TLocateParamsList.Items Property

Lists the key field/value lists for all the adapters that use this object to store their locate parameters.

Pascal

```
property Items [I: Integer]: TLocateParams;
```

C++

```
__property TLocateParams Items[int I];
```

Description

Use Items to obtain a TLocateParams (see page 8090) object given its position in the list. Each TLocateParams (see page 8090) object holds the list of key fields and values for a single dataset adapter.

I is the index of the TLocateParams (see page 8090) object, where 0 specifies the first item, 1 specifies the second item, and so on. The Count (see page 8100) property gives an upper bound for this index.

Note: In Delphi, Items is the default property of TLocateParamsList (see page 8096). That means the property name may be omitted in code. For example, the line

Note: List := DataSetAdapter1.LocateParams.Items[0];

Note: can also be written

Note: List := DataSetAdapter1.LocateParams[0];

Note: Items provides indexed access to the key field/value lists. To access the list for a specified dataset adapter, use the ItemByAdapterName (see page 8100) method instead.

See Also

Count (see page 8100)

ItemByAdapterName (see page 8100)

TDataSetAdapter (see page 7872)

operator_sb

41.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAdapterDataSetComponent (see page 8102)	This is class DBAdapt.IAdapterDataSetComponent.
	IDataSetAdapterFieldClass (see page 8104)	This is class DBAdapt.IDataSetAdapterFieldClass.
	IGetDataSetKeyName (see page 8104)	This is class DBAdapt.IGetDataSetKeyName.
	IKeyParams (see page 8105)	IKeyParams lists the names and values of the fields a dataset adapter uses to uniquely identify database records.
	IPrepareDataSet (see page 8109)	This is class DBAdapt.IPrepareDataSet.

41.2.1 IAdapterDataSetComponent Interface

Class Hierarchy

DBAdapt.IAdapterDataSetComponent

Pascal

```
IAdapterDataSetComponent = interface;
```

C++

```
__interface IAdapterDataSetComponent;
```

File

DBAdapt

Description

This is class DBAdapt.IAdapterDataSetComponent.

41.2.1.1 IAdapterDataSetComponent Members

The following tables list the members exposed by IAdapterDataSetComponent.

Public Methods

	Name	Description
	GetDataSet (see page 8103)	This is GetDataSet, a member of class IAdapterDataSetComponent.
	SetDataSet (see page 8103)	This is SetDataSet, a member of class IAdapterDataSetComponent.

Public Properties

	Name	Description
	DataSet (see page 8103)	This is DataSet, a member of class IAdapterDataSetComponent.

41.2.1.2 IAdapterDataSetComponent Methods

The methods of the IAdapterDataSetComponent class are listed here.

Public Methods

	Name	Description
	GetDataSet (see page 8103)	This is GetDataSet, a member of class IAdapterDataSetComponent.
	SetDataSet (see page 8103)	This is SetDataSet, a member of class IAdapterDataSetComponent.

41.2.1.2.1 IAdapterDataSetComponent.GetDataSet Method

Pascal

```
function GetDataSet: TDataSet;
```

C++

```
__fastcall TDataSet GetDataSet();
```

Description

This is GetDataSet, a member of class IAdapterDataSetComponent.

41.2.1.2.2 IAdapterDataSetComponent.SetDataSet Method

Pascal

```
procedure SetDataSet(Value: TDataSet);
```

C++

```
__fastcall SetDataSet(TDataSet Value);
```

Description

This is SetDataSet, a member of class IAdapterDataSetComponent.

41.2.1.3 IAdapterDataSetComponent Properties

The properties of the IAdapterDataSetComponent class are listed here.

Public Properties

	Name	Description
	DataSet (see page 8103)	This is DataSet, a member of class IAdapterDataSetComponent.

41.2.1.3.1 IAdapterDataSetComponent.DataSet Property

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

This is DataSet, a member of class IAdapterDataSetComponent.

41.2.2 IDataSetAdapterFieldClass Interface

Class Hierarchy

DBAdapt.IDataSetAdapterFieldClass

Pascal

```
IDataSetAdapterFieldClass = interface;
```

C++

```
__interface IDataSetAdapterFieldClass;
```

File

DBAdapt

Description

This is class DBAdapt.IDataSetAdapterFieldClass.

41.2.3 IGetDataSetKeyName Interface

Class Hierarchy

DBAdapt.IGetDataSetKeyName

Pascal

```
IGetDataSetKeyName = interface;
```

C++

```
__interface IGetDataSetKeyName;
```

File

DBAdapt

Description

This is class DBAdapt.IGetDataSetKeyName.

41.2.3.1 IGetDataSetKeyName Members

The following tables list the members exposed by IGetDataSetKeyName.

Public Methods

	Name	Description
	GetKeyName (see page 8105)	This is GetKeyName, a member of class IGetDataSetKeyName.

Public Properties

	Name	Description
R	KeyName (see page 8105)	This is KeyName, a member of class IGetDataSetKeyName.

41.2.3.2 IGetDataSetKeyName Methods

The methods of the IGetDataSetKeyName class are listed here.

Public Methods

	Name	Description
	GetKeyName (see page 8105)	This is GetKeyName, a member of class IGetDataSetKeyName.

41.2.3.2.1 IGetDataSetKeyName.GetKeyName Method

Pascal

```
function GetKeyName: string;
```

C++

```
__fastcall AnsiString GetKeyName();
```

Description

This is GetKeyName, a member of class IGetDataSetKeyName.

41.2.3.3 IGetDataSetKeyName Properties

The properties of the IGetDataSetKeyName class are listed here.

Public Properties

	Name	Description
R	KeyName (see page 8105)	This is KeyName, a member of class IGetDataSetKeyName.

41.2.3.3.1 IGetDataSetKeyName.KeyName Property

Pascal

```
property KeyName: string;
```

C++

```
__property AnsiString KeyName;
```

Description

This is KeyName, a member of class IGetDataSetKeyName.

41.2.4 IKeyParams Interface

IKeyParams lists the names and values of the fields a dataset adapter uses to uniquely identify database records.

Class Hierarchy

DBAdapt.IKeyParams

Pascal

```
IKeyParams = interface;
```

C++

```
__interface IKeyParams;
```

File

DBAdapt

Description

Dataset adapters (TDataSetAdapter (see page 7872)) use this interface internally to read the field values that uniquely identify database records. IKeyParams introduces two properties, Names (see page 8108) and Values (see page 8108), that represent the names of dataset fields and the corresponding values on a desired record. The set of names and values obtained from this interface identify a single database record.

Note: Method declarations that use IKeyParams use the _di_IKeyParams type instead. This type is a DelphiInterface wrapper around the IKeyParams interface:

typedef System (see page 22079):: DelphiInterface < IKeyParams > _di_IKeyParams;

See Also

TCustomDataSetAdapter (see page 7766)

TDataSetAdapter (see page 7872)

41.2.4.1 IKeyParams Members

The following tables list the members exposed by IKeyParams.

Public Methods

	Name	Description
	Count (see page 8106)	Returns the number of name/value pairs that can be read from this interface.
	GetName (see page 8107)	Returns the value of the Names (see page 8108) property.
	GetValue (see page 8107)	Returns the value of the Values (see page 8108) property.

Public Properties

	Name	Description
R	Names (see page 8108)	Lists the names of the key fields that uniquely identify dataset records.
R	Values (see page 8108)	Lists the values of the key fields that uniquely identify a specific dataset record.

41.2.4.2 IKeyParams Methods

The methods of the IKeyParams class are listed here.

Public Methods

	Name	Description
	Count (see page 8106)	Returns the number of name/value pairs that can be read from this interface.
	GetName (see page 8107)	Returns the value of the Names (see page 8108) property.
	GetValue (see page 8107)	Returns the value of the Values (see page 8108) property.

41.2.4.2.1 IKeyParams.Count Method

Returns the number of name/value pairs that can be read from this interface.

Pascal

```
function Count: Integer;
```

C++

```
__fastcall int Count();
```

Description

Count returns the number of key fields that this interface lists. The return value provides an upper bound to the index you can use when reading the Names (see page 8108) or Values (see page 8108) property.

See Also

Names (see page 8108)

Values (see page 8108)

41.2.4.2.2 IKeyParams.GetName Method

Returns the value of the Names (see page 8108) property.

Pascal

```
function GetName(I: Integer): string;
```

C++

```
__fastcall AnsiString GetName(int I);
```

Description

GetName is the read implementation of the Names (see page 8108) property.

I is the index of the fieldname to return.

See Also

Names (see page 8108)

41.2.4.2.3 IKeyParams.GetValue Method

Returns the value of the Values (see page 8108) property.

Pascal

```
function GetValue(I: Integer): string;
```

C++

```
__fastcall AnsiString GetValue(int I);
```

Description

GetValue is the read implementation of the Values (see page 8108) property.

I is the index of the field value to return.

See Also

Values (see page 8108)

41.2.4.3 IKeyParams Properties

The properties of the IKeyParams class are listed here.

Public Properties

	Name	Description
R	Names (see page 8108)	Lists the names of the key fields that uniquely identify dataset records.
R	Values (see page 8108)	Lists the values of the key fields that uniquely identify a specific dataset record.

41.2.4.3.1 IKeyParams.Names Property

Lists the names of the key fields that uniquely identify dataset records.

Pascal

```
property Names [I: Integer]: string;
```

C++

```
__property AnsiString Names[int I];
```

Description

Read Names to obtain the name of a key field. Each value in Names is equal to the FieldName property of a field in the target dataset.

I is the index of the field name, where 0 specifies the first key field, 1 specifies the second key field, and so on. The Count (see page 8106) method provides an upper bound for this index.

See Also

Values (see page 8108)

Count (see page 8106)

FieldName (see page 7164)

41.2.4.3.2 IKeyParams.Values Property

Lists the values of the key fields that uniquely identify a specific dataset record.

Pascal

```
property Values [I: Integer]: string;
```

C++

```
__property AnsiString Values[int I];
```

Description

Read Values to obtain the value of a key field. Each value in Values is equal to the AsString property of the corresponding field when the target dataset is positioned on the desired record.

I is the index of the field value, where 0 specifies the first key field, 1 specifies the second key field, and so on. The Count (see page 8106) method provides an upper bound for this index.

See Also

Names (see page 8108)

Count (see page 8106)

AsString (see page 7151)

41.2.5 IPrepareDataSet Interface

Class Hierarchy

DBAdapt.IPrepareDataSet

Pascal

```
IPrepareDataSet = interface;
```

C++

```
__interface IPrepareDataSet;
```

File

DBAdapt

Description

This is class DBAdapt.IPrepareDataSet.

41.2.5.1 IPrepareDataSet Members

The following tables list the members exposed by IPrepareDataSet.

Public Methods

	Name	Description
	PrepareDataSet (see page 8109)	This is PrepareDataSet, a member of class IPrepareDataSet.

41.2.5.2 IPrepareDataSet Methods

The methods of the IPrepareDataSet class are listed here.

Public Methods

	Name	Description
	PrepareDataSet (see page 8109)	This is PrepareDataSet, a member of class IPrepareDataSet.

41.2.5.2.1 IPrepareDataSet.PrepareDataSet Method

Pascal

```
procedure PrepareDataSet;
```

C++

```
__fastcall PrepareDataSet();
```

Description

This is PrepareDataSet, a member of class IPrepareDataSet.

41.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AddLocateParamsString (see page 8110)	This is function DBAdapt.AddLocateParamsString.

41.3.1 DBAdapt.AddLocateParamsString Function

Pascal

```
procedure AddLocateParamsString(AAdapterName: string; AKeyName: string; AKeyValue: string;
AParams: TLocateParamsList);
```

C++

```
AddLocateParamsString(AnsiString AAdapterName, AnsiString AKeyName, AnsiString AKeyValue,
TLocateParamsList AParams);
```

File

DBAdapt

Description

This is function DBAdapt.AddLocateParamsString.

41.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TDataSetAdapterFieldFlag (see page 8111)	This is record DBAdapt.TDataSetAdapterFieldFlag.
	TDataSetAdapterMode (see page 8111)	This is record DBAdapt.TDataSetAdapterMode.
	TDataSetAdapterRequestParamOption (see page 8112)	This is record DBAdapt.TDataSetAdapterRequestParamOption.

41.4.1 DBAdapt.TDataSetAdapterFieldFlag Enumeration

Pascal

```
TDataSetAdapterFieldFlag = (
  ffInKey,
  ffInOrigValues
);
```

C++

```
enum TDataSetAdapterFieldFlag {
  ffInKey,
  ffInOrigValues
};
```

File

DBAdapt

Description

This is record DBAdapt.TDataSetAdapterFieldFlag.

41.4.2 DBAdapt.TDataSetAdapterMode Enumeration

Pascal

```
TDataSetAdapterMode = (
  amInsert,
  amEdit,
  amBrowse,
  amQuery
);
```

C++

```
enum TDataSetAdapterMode {
  amInsert,
  amEdit,
  amBrowse,
  amQuery
};
```

File

DBAdapt

Description

This is record DBAdapt.TDataSetAdapterMode.

41.4.3 DBAdapt.TDataSetAdapterRequestParamOption Enumeration

Pascal

```
TDataSetAdapterRequestParamOption = (
  poLocateParams,
  poCurrentPage
);
```

C++

```
enum TDataSetAdapterRequestParamOption {
  poLocateParams,
  poCurrentPage
};
```

File

DBAdapt

Description

This is record DBAdapt.TDataSetAdapterRequestParamOption.

41.5 Types

The following table lists types in this documentation.

Types

Name	Description
TDataAdapterFieldValidateValueEvent (see page 8113)	This is type DBAdapt.TDataAdapterFieldValidateValueEvent.
TDataSetAdapterFieldFlags (see page 8113)	This is type DBAdapt.TDataSetAdapterFieldFlags.
TDataSetAdapterFieldGetStringEvent (see page 8114)	This is type DBAdapt.TDataSetAdapterFieldGetStringEvent.
TDataSetAdapterFieldGetValueEvent (see page 8114)	This is type DBAdapt.TDataSetAdapterFieldGetValueEvent.
TDataSetAdapterFieldValueEvent (see page 8114)	This is type DBAdapt.TDataSetAdapterFieldValueEvent.
TDataSetAdapterImageFieldClass (see page 8115)	This is type DBAdapt.TDataSetAdapterImageFieldClass.
TDataSetAdapterModes (see page 8115)	This is type DBAdapt.TDataSetAdapterModes.
TDataSetAdapterRequestParamOptions (see page 8115)	This is type DBAdapt.TDataSetAdapterRequestParamOptions.

41.5.1 DBAdapt.TDataAdapterFieldValidateValueEvent Type

Pascal

```
TDataAdapterFieldValidateValueEvent = procedure (Sender: TObject; Field: TField; Value:
IActionFieldValue; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; Field: TField; Value: IActionFieldValue; var Handled: Boolean) (
TDataAdapterFieldValidateValueEvent)();
```

File

DBAdapt

Description

This is type DBAdapt.TDataAdapterFieldValidateValueEvent.

41.5.2 DBAdapt.TDataSetAdapterFieldFlags Type

Pascal

```
TDataSetAdapterFieldFlags = set of TDataSetAdapterFieldFlag;
```

C++

```
TDataSetAdapterFieldFlag TDataSetAdapterFieldFlags;
```

File

DBAdapt

Description

This is type DBAdapt.TDataSetAdapterFieldFlags.

41.5.3 DBAdapt.TDataSetAdapterFieldGetStringEvent Type

Pascal

```
TDataSetAdapterFieldGetStringEvent = procedure (Sender: TObject; Field: TField; var Value:
string) of object;
```

C++

```
(Sender: TObject; Field: TField; var Value: string) ( TDataSetAdapterFieldGetStringEvent)();
```

File

DBAdapt

Description

This is type DBAdapt.TDataSetAdapterFieldGetStringEvent.

41.5.4 DBAdapt.TDataSetAdapterFieldGetValueEvent Type

Pascal

```
TDataSetAdapterFieldGetValueEvent = procedure (Sender: TObject; Field: TField; var Value:
Variant) of object;
```

C++

```
(Sender: TObject; Field: TField; var Value: Variant) ( TDataSetAdapterFieldGetValueEvent)();
```

File

DBAdapt

Description

This is type DBAdapt.TDataSetAdapterFieldGetValueEvent.

41.5.5 DBAdapt.TDataSetAdapterFieldValueEvent Type

Pascal

```
TDataSetAdapterFieldValueEvent = procedure (Sender: TObject; Field: TField; Value: Variant) of
object;
```

C++

```
(Sender: TObject; Field: TField; Value: Variant) ( TDataSetAdapterFieldValueEvent)();
```

File

DBAdapt

Description

This is type DBAdapt.TDataSetAdapterFieldValueEvent.

41.5.6 DBAdapt.TDataSetAdapterImageFieldClass Type

Pascal

```
TDataSetAdapterImageFieldClass = class of TBaseDataSetAdapterImageField;
```

C++

```
TBaseDataSetAdapterImageField TDataSetAdapterImageFieldClass;
```

File

DBAdapt

Description

This is type DBAdapt.TDataSetAdapterImageFieldClass.

41.5.7 DBAdapt.TDataSetAdapterModes Type

Pascal

```
TDataSetAdapterModes = set of TDataSetAdapterMode;
```

C++

```
TDataSetAdapterMode TDataSetAdapterModes;
```

File

DBAdapt

Description

This is type DBAdapt.TDataSetAdapterModes.

41.5.8 DBAdapt.TDataSetAdapterRequestParamOptions Type

Pascal

```
TDataSetAdapterRequestParamOptions = set of TDataSetAdapterRequestParamOption;
```

C++

```
TDataSetAdapterRequestParamOption TDataSetAdapterRequestParamOptions;
```

File

DBAdapt

Description

This is type DBAdapt.TDataSetAdapterRequestParamOptions.

41.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
DataSetAdapterImageFieldClass (see page 8116)	This is variable DBAdapt.DataSetAdapterImageFieldClass.

41.6.1 DBAdapt.DataSetAdapterImageFieldClass Variable

Pascal

```
DataSetAdapterImageFieldClass: TDataSetAdapterImageFieldClass;
```

C++

```
TDataSetAdapterImageFieldClass DataSetAdapterImageFieldClass;
```

File

DBAdapt

Description

This is variable DBAdapt.DataSetAdapterImageFieldClass.

41.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
AllActionParamOptions (see page 8117)	This is constant DBAdapt.AllActionParamOptions.
AllImageParamOptions (see page 8117)	This is constant DBAdapt.AllImageParamOptions.
DataSetAdapterModeNames (see page 8117)	This is constant DBAdapt.DataSetAdapterModeNames.

41.7.1 DBAdapt.AllActionParamOptions Constant

Pascal

```
AllActionParamOptions = [poLocateParams, poCurrentPage];
```

C++

```
AllActionParamOptions = [poLocateParams, poCurrentPage];
```

File

DBAdapt

Description

This is constant DBAdapt.AllActionParamOptions.

41.7.2 DBAdapt.AllImageParamOptions Constant

Pascal

```
AllImageParamOptions = [poLocateParams];
```

C++

```
AllImageParamOptions = [poLocateParams];
```

File

DBAdapt

Description

This is constant DBAdapt.AllImageParamOptions.

41.7.3 DBAdapt.DataSetAdapterModeNames Constant

Pascal

```
DataSetAdapterModeNames: array[TDataSetAdapterMode] of string = ('Insert', 'Edit', 'Browse',
```

```
'Query');
```

C++

```
array[TDataSetAdapterMode] of string DataSetAdapterModeNames = ('Insert', 'Edit', 'Browse',
'Query');
```

File

DBAdapt

Description

This is constant DBAdapt.DataSetAdapterModeNames.

42 DBAdaptImg Namespace

This is namespace DBAdaptImg.

Topics

Name	Description
Classes (see page 8120)	The following table lists classes in this documentation.
Types (see page 8149)	The following table lists types in this documentation.

Classes

	Name	Description
	TCustomDataSetAdapterImageField (see page 8120)	TCustomDataSetAdapterImageField is the base class for adapter field objects that represent a graphic field from a database.
	TDataSetAdapterImageField (see page 8138)	tdatasetadapterimagefield is the adapter field object that represents a graphic field from a database.

Types

Name	Description
TImageDataSetFieldGetImageEvent (see page 8149)	This is type DBAdaptImg.TImageDataSetFieldGetImageEvent.

42.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCustomDataSetAdapterImageField (see page 8120)	TCustomDataSetAdapterImageField is the base class for adapter field objects that represent a graphic field from a database.
	TDataSetAdapterImageField (see page 8138)	tdatasetadapterimagefield is the adapter field object that represents a graphic field from a database.

42.1.1 TCustomDataSetAdapterImageField Class

TCustomDataSetAdapterImageField is the base class for adapter field objects that represent a graphic field from a database.

Class Hierarchy

Pascal

```
TCustomDataSetAdapterImageField = class(TBaseDataSetAdapterImageField, IGetAdapterImage,
IGetAdapterItemRequestParams, IAdapterRequestHandler, IWebImageHREF);
```

C++

```
class TCustomDataSetAdapterImageField : public TBaseDataSetAdapterImageField, public
IGetAdapterImage, public IGetAdapterItemRequestParams, public IAdapterRequestHandler, public
IWebImageHREF;
```

File

DBAdaptImg

Description

TCustomDataSetAdapterImageField is the base class for all adapter field objects that represent graphics fields from a dataset (TGraphicField objects). TCustomDataSetAdapterImageField descendants are included as the fields of a dataset adapter (TDataSetAdapter) when that dataset adapter represents a database that includes graphics fields.

TCustomDataSetAdapterImageField represents a single field from a database table, and enables server-side script to refer to or update that field's value.

Do not use TCustomDataSetAdapterImageField directly: it is intended only as a base class. Instead, use a TCustomDataSetAdapterImageField descendant such as TDataSetAdapterImageField (see page 8138), which publishes several of the properties and events that you can use at design time to bind the adapter field to the corresponding dataset field.

See Also

TDataSetAdapterImageField (see page 8138)

TDataSetAdapter (see page 7872)

TGraphicField (see page 7257)

42.1.1.1 TCustomDataSetAdapterImageField Members

The following tables list the members exposed by TCustomDataSetAdapterImageField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckOrigValue (see page 7736)	Ensures that the current value of this field is the same as the original value.
	CheckOrUpdateValue (see page 8128)	Checks whether a specified request contains a valid value and, if requested, uses that value to update the database.
	CheckValueChange (see page 7737)	Indicates whether the current value of this field is the same as the original value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
	CreateRequestContext (see page 8129)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 7738)	Indicates whether references to this field in HTML forms must be qualified with the name of the parent adapter.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 7738)	Returns the value of the Adapter (see page 7752) property.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterImage (see page 8130)	Returns an interface that can be used to access the Image property of this adapter field.
	GetAdapterItemRequestParams (see page 8130)	Provides a unique identifier and a set of name/value pairs that can be used to identify this field's image file.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.

	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7738)	Returns the dataset associated with this field's parent adapter.
V	GetDataSetFieldValue (see page 8131)	Returns the value of the associated dataset field.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 7739)	Returns a string value that suggests an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 8131)	Returns the HTML display style type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 7741)	Returns a string value that suggests an HTML editing style for this adapter field.
V	GetInputStyleType (see page 8132)	Returns the style of the HTML control to use for editing this adapter field.
	GetKeyName (see page 7742)	Returns the name of the associated dataset field if it appears in the index (key) that the parent dataset adapter uses.
	GetKeyParams (see page 7743)	Returns a list of fields that the parent dataset adapter uses to locate records, along with their current values in that adapter's dataset.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetReadOnly (see page 7743)	Indicates whether the adapter field represents a read-only database field.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 7744)	Returns a string value that indicates what function a control that represents this field should perform.
V	GetViewModeType (see page 7744)	Returns the view mode for the function a control that represents this field should perform.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HandleRequest (see page 8132)	Handles incoming requests for the image value of this field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 7745)	Provides the underlying implementation for the AsFormatted method.
V	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
V	ImplCheckOrigValue (see page 7745)	Provides the underlying implementation for the CheckOrigValue (see page 7736) method.
V	ImplCheckValueChange (see page 8133)	Provides the underlying implementation for the CheckValueChange method.
	ImplCreateRequestContext (see page 8134)	Provides the underlying implementation for the CreateRequestContext (see page 8129) method.
V	ImplGetDisplayLabel (see page 7747)	Provides the underlying implementation for the GetDisplayLabel method.
V	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 7747)	Provides the underlying implementation for the GetDisplayWidth method.
V	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
V	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 7747)	Provides the underlying implementation for the GetRequired method.
V	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.

	ImplGetValue (see page 7748)	Provides the underlying implementation for the GetValue method.
	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 7748)	Provides the underlying implementation for the GetVisible method.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHandleRequest (see page 8134)	Provides the underlying implementation for the HandleRequest (see page 8132) method.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 7749)	Provides the underlying implementation for the SetFieldName method.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	ImplUpdateValue (see page 8135)	Provides the underlying implementation for the UpdateValue method.
	ImplWebImageHREF (see page 8135)	Provides the underlying implementation for the WebImageHREF (see page 8136) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7750)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	PrepareDataSet (see page 7750)	Responds when the parent dataset adapter prepares its dataset.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RenderAdapterImage (see page 8136)	Writes the image described by an image request to the current response message.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSetField (see page 7751)	Sets the value of the DataSetField (see page 7752) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 7751)	Updates the adapter field when the user assigns a new value.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.

	WebImageHREF (see page 8136)	Retrieves a URL for an external image file that contains the value of this field.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7752)	Indicates the dataset adapter that uses this object to represent one of its fields.
R	Adapter (see page 23943)	Returns the parent adapter component.
	DataSetField (see page 7752)	Specifies the name of the dataset field that this adapter field represents.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 7754)	Occurs when the adapter field needs to supply its formatted value as a string that can appear in HTML forms.
	OnGetHREF (see page 8137)	Occurs when the WebSnap application needs the URL that clients use to request this field's image.
	OnGetImage (see page 8137)	Occurs when the WebSnap application receives a request for this field's image.
	OnGetValue (see page 7754)	Occurs when the adapter field needs to supply its value as a Variant.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 8127)	Creates and initializes an instance of TCustomDataSetAdapterImageField (see page 8120).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 7737)	Returns the dataset field whose value this adapter field represents.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetKeyName (see page 8105)	This is GetKeyName, a member of class IGetDataSetKeyName.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PrepareDataSet (see page 8109)	This is PrepareDataSet, a member of class IPrepareDataSet.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.

	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	FieldFlags (see page 7752)	Indicates various attributes of the adapter field.
	FieldModes (see page 7753)	Indicates what dataset adapter modes this field supports.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
R	KeyName (see page 8105)	This is KeyName, a member of class IGetDataSetKeyName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

42.1.1.2 TCustomDataSetAdapterImageField.Create Constructor

Creates and initializes an instance of TCustomDataSetAdapterImageField (see page 8120).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomDataSetAdapterImageField(TComponent * AOwner);
```

Description

The TCustomDataSetAdapterImageField (see page 8120) constructor allocates memory for the adapter field object and initializes its properties.

AOwner is the component that is responsible for starting the streaming process when a module containing the adapter field is loaded or saved. It becomes the value of the Owner property. Typically, AOwner is the Web module.

See Also

Owner (see page 2807)

42.1.1.3 TCustomDataSetAdapterImageField Methods

The methods of the TCustomDataSetAdapterImageField class are listed here.

Protected Methods

	Name	Description
	CheckOrUpdateValue (see page 8128)	Checks whether a specified request contains a valid value and, if requested, uses that value to update the database.
	CreateRequestContext (see page 8129)	Creates an object that describes the adapter action parameters and adapter field values.
	GetAdapterImage (see page 8130)	Returns an interface that can be used to access the Image property of this adapter field.
	GetAdapterItemRequestParams (see page 8130)	Provides a unique identifier and a set of name/value pairs that can be used to identify this field's image file.
V	GetDataSetFieldValue (see page 8131)	Returns the value of the associated dataset field.
V	GetDisplayStyleType (see page 8131)	Returns the HTML display style type for this adapter field.
V	GetInputStyleType (see page 8132)	Returns the style of the HTML control to use for editing this adapter field.
	HandleRequest (see page 8132)	Handles incoming requests for the image value of this field.
V	ImplCheckValueChange (see page 8133)	Provides the underlying implementation for the CheckValueChange method.
	ImplCreateRequestContext (see page 8134)	Provides the underlying implementation for the CreateRequestContext (see page 8129) method.
V	ImplHandleRequest (see page 8134)	Provides the underlying implementation for the HandleRequest (see page 8132) method.
V	ImplUpdateValue (see page 8135)	Provides the underlying implementation for the UpdateValue method.
V	ImplWebImageHREF (see page 8135)	Provides the underlying implementation for the WebImageHREF (see page 8136) method.
	RenderAdapterImage (see page 8136)	Writes the image described by an image request to the current response message.
	WebImageHREF (see page 8136)	Retrieves a URL for an external image file that contains the value of this field.

42.1.1.3.1 TCustomDataSetAdapterImageField.CheckOrUpdateValue Method

Checks whether a specified request contains a valid value and, if requested, uses that value to update the database.

Pascal

```
function CheckOrUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer; AUpdate:
Boolean): Boolean;
```

C++

```
__fastcall Boolean CheckOrUpdateValue(IActionRequest AActionRequest, int AFieldIndex, Boolean
```

```
AUpdate);
```

Description

CheckOrUpdateValue is called internally by the implementation of both the UpdateValue and the CheckValueChange methods to check for a properly-formed image value from an incoming request message. If requested, it also updates the corresponding database field. If the request object does not contain an appropriate value (for example, multiple values or a value that is not in .bmp format), CheckOrUpdateValue raises an EAdapterFieldException exception.

AActionRequest is the interface of the action request object. CheckOrUpdateValue uses this interface to obtain the value from the incoming request to check or write to the database field.

AFieldIndex is the index of this field in the set of field values that AActionRequest maintains.

AUpdate indicates whether CheckOrUpdateValue should update the corresponding database field. If AUpdate is true, CheckOrUpdateValue assigns the properly validated value that it finds in AActionRequest to the associated database field. If AUpdate is false, CheckOrUpdateValue merely checks for the existence of an appropriate value.

CheckOrUpdateValue returns true if this is not an update request and the request contains a reference to an appropriate image file.

See Also

UpdateValue (see page 7751)

EAdapterFieldException (see page 23698)

CheckValueChange (see page 7737)

42.1.1.3.2 TCustomDataSetAdapterImageField.CreateRequestContext Method

Creates an object that describes the adapter action parameters and adapter field values.

Pascal

```
procedure CreateRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall CreateRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

CreateRequestContext creates a new request context based on the specified parameters and stores it with the global Web context. The adapter dispatcher calls this function before calling HandleRequest (see page 8132). The WebSnap can access the resulting request context by reading the AdapterRequest property of the Web context, which is available through the global WebContext function.

DispatchParams is an interface that contains the parameters that were included in the incoming request. These parameters can then be used to identify an image for this field's value.

See Also

AdapterRequest (see page 24655)

TAdapterDispatcher (see page 24840)

WebContext (see page 24687)

HandleRequest (see page 8132)

42.1.1.3.3 TCustomDataSetAdapterImageField.GetAdapterImage Method

Returns an interface that can be used to access the Image property of this adapter field.

Pascal

```
function GetAdapterImage: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterImage();
```

Description

GetAdapterImage provides the implementation of the Image property on this adapter field. This method is called automatically when server-side script references the Image property. The resulting interface can then be used to obtain the value of the image.

GetAdapterImage checks the Mode property of the parent adapter for this field. If the parent dataset adapter is in insert mode or query mode, GetAdapterImage returns nil (Delphi) or NULL (C++), because there is no current value to the image field (it does not currently represent the value of a graphic field in the adapter's dataset). Otherwise, GetAdapterImage returns self (Delphi) or this (C++), so that the WebSnap application can obtain the current image for this field.

See Also

Mode (see page 7810)

Adapter (see page 7752)

42.1.1.3.4 TCustomDataSetAdapterImageField.GetAdapterItemRequestParams Method

Provides a unique identifier and a set of name/value pairs that can be used to identify this field's image file.

Pascal

```
procedure GetAdapterItemRequestParams(var AIdentifier: string; AParams:
IAdapterItemRequestParams);
```

C++

```
__fastcall GetAdapterItemRequestParams(AnsiString AIdentifier, IAdapterItemRequestParams
AParams);
```

Description

GetAdapterItemRequestParams is used (along with the OnGetHREF (see page 8137) event) to identify the URL that addresses the image file associated with this adapter field. When server-side script uses the Image.AsHREF property, this field generates an OnGetHREF (see page 8137) event. If an OnGetHREF (see page 8137) event handler does not provide a URL for an external file that contains the field's image, then the WebSnap application calls GetAdapterItemRequestParams to obtain the information it needs to locate or construct the field's image.

AIdentifer returns a unique identifier for this adapter field object.

AParams is the interface to which GetAdapterItemRequestParams writes a set of name/value pairs that identify the image for this field.

See Also

OnGetHREF (see page 8137)

42.1.1.3.5 TCustomDataSetAdapterImageField.GetDataSetFieldValue Method

Returns the value of the associated dataset field.

Pascal

```
function GetDataSetFieldValue(Field: TField): Variant; override;
```

C++

```
virtual __fastcall Variant GetDataSetFieldValue(TField Field);
```

Description

GetDataSetFieldValue returns the value of the associated dataset field, given a field object. This method is called by the implementation of the Value property when there is no OnGetValue event handler.

Field is the dataset field whose value this adapter field represents. It is the component that the FindField method returns.

In TCustomDataSetAdapterImageField (see page 8120), GetDataSetFieldValue returns the DisplayText property of the Field parameter. This is the text-readable value of the image field. From server-side script, the image itself is available through the Image property of this adapter field.

See Also

OnGetValue (see page 7754)

Value (see page 23945)

FindField (see page 7737)

DisplayText (see page 7162)

OnGetImage (see page 8137)

42.1.1.3.6 TCustomDataSetAdapterImageField.GetDisplayStyleType Method

Returns the HTML display style type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const AnsiString
AAdapterMode);
```

Description

GetDisplayStyleType returns the best way to display this field's value in HTML forms.

AAdapterMode indicates the current mode of the parent dataset adapter. It is one of the following values:

Value	Mode	Meaning
Insert	amInsert	The field value is to be used for entering a value in a newly inserted record.
Edit	amEdit	The field value is to be used for editing the current field value.
Browse	amBrowse	The field value is to be displayed in a read-only manner.
Query	amQuery	The field value represents a criterion to use in a generated query.

In TBaseDataSetAdapterField, GetDisplayStyleType returns htmldImage, indicating that the value should appear as an image.

See Also

GetDisplayStyle (see page 7739)

TDataSetAdapterModes (see page 8115)

GetInputStyleType (see page 8132)

42.1.1.3.7 TCustomDataSetAdapterImageField.GetInputStyleType Method

Returns the style of the HTML control to use for editing this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const AnsiString
AAdapterMode);
```

Description

GetInputStyleType returns the recommended type of control to use for editing this field's value in HTML forms.

AAdapterMode indicates the current mode of the parent dataset adapter. It is one of the following values:

Value	Mode	Meaning
Insert	amInsert	The control is to be used for entering a field value in a newly inserted record.
Edit	amEdit	The control is to be used for editing the current field value.
Browse	amBrowse	The control is to be used for read-only display.
Query	amQuery	The control is to be used for entering a criterion to use in a generated query.

In TBaseDataSetAdapterField, GetInputStyleType always returns htmliFile, indicating that the default control is a file upload element.

See Also

GetInputStyle (see page 7741)

GetDisplayStyleType (see page 8131)

TDataSetAdapterModes (see page 8115)

42.1.1.3.8 TCustomDataSetAdapterImageField.HandleRequest Method

Handles incoming requests for the image value of this field.

Pascal

```
function HandleRequest(DispatchParams: IAdapterDispatchParams): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(IAdapterDispatchParams DispatchParams);
```

Description

The adapter dispatcher calls HandleRequest to process incoming request messages directed at this adapter field. HandleRequest interprets the incoming parameters that accompany the request, uses them to locate the image that is the value of this field, and writes that image to the current response.

DispatchParams is an interface for reading the parameters that accompanied the incoming request message. These parameters, which originated from the GetAdapterItemRequestParams (see page 8130) method, identify the image that is the value of this field.

HandleRequest delegates the task of processing the image request to the ImplHandleRequest (see page 8134) method. ImplHandleRequest (see page 8134) then obtains the image by generating an OnGetImage (see page 8137) event or by retrieving the value of the associated graphic field object.

See Also

ImplHandleRequest (see page 8134)

OnGetImage (see page 8137)

TAdapterDispatcher (see page 24840)

GetAdapterItemRequestParams (see page 8130)

TGraphicField (see page 7257)

42.1.1.3.9 TCustomDataSetAdapterImageField.ImplCheckValueChange Method

Provides the underlying implementation for the CheckValueChange method.

Pascal

```
function ImplCheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
override;
```

C++

```
virtual __fastcall Boolean ImplCheckValueChange(IActionRequest AActionRequest, int
AFieldIndex);
```

Description

The CheckValueChange method calls ImplCheckValueChange to determine whether the value of this adapter field has changed from the value originally retrieved from the database table.

AActionRequest is the interface of the action request object. CheckOrigValue can use this interface to obtain information about the current request (such as the value of the hidden field that contains the original value).

AFieldIndex is the index of this field in the set of field values that AActionRequest maintains.

ImplCheckValueChange calls CheckOrUpdateValue (see page 8128) to verify that the request contains a value that can be assigned to the associated database field. If so, it returns true. Although this does not truly reflect whether the value has changed, it is a conservative approach that ensures no valid changes to the image are lost. If the request contains no value, this method returns false, indicating no update. If the request contains an invalid value, this method raises an EAdapterFieldException exception.

See Also

CheckValueChange (see page 7737)

Value (see page 23945)

CheckOrUpdateValue (see page 8128)

EAdapterFieldException (see page 23698)

42.1.1.3.10 TCustomDataSetAdapterImageField.ImplCreateRequestContext Method

Provides the underlying implementation for the CreateRequestContext (see page 8129) method.

Pascal

```
procedure ImplCreateRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall ImplCreateRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

CreateRequestContext (see page 8129) calls this method to create a request context based on the specified parameters and store it with the global Web context.

DispatchParams is an interface that contains the parameters that were included in the incoming request. These parameters can then be used to identify an image for this field's value.

See Also

AdapterRequest (see page 24655)

CreateRequestContext (see page 8129)

WebContext (see page 24687)

42.1.1.3.11 TCustomDataSetAdapterImageField.ImplHandleRequest Method

Provides the underlying implementation for the HandleRequest (see page 8132) method.

Pascal

```
function ImplHandleRequest(DispatchParams: IAdapterDispatchParams): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest(IAdapterDispatchParams DispatchParams);
```

Description

The HandleRequest (see page 8132) method calls ImplHandleRequest to process incoming request messages directed at this adapter field.

DispatchParams is an interface for reading the parameters that accompanied the incoming request message. These parameters identify the image that is the value of this field.

ImplHandleRequest uses the AParams interface to access interfaces for the current request and response objects. It then generates an OnGetImage (see page 8137) event to obtain the current image. If the OnGetImage (see page 8137) event handler does not provide an image, ImplHandleRequest retrieves the value of the associated graphic field object. Finally, ImplHandleRequest writes the image to the current response object.

See Also

HandleRequest (see page 8132)

OnGetImage (see page 8137)

TGraphicField (see page 7257)

42.1.1.3.12 TCustomDataSetAdapterImageField.ImplUpdateValue Method

Provides the underlying implementation for the UpdateValue method.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The UpdateValue method calls ImplUpdateValue to respond when the user assigns a new value to this field.

AActionRequest is interface of the action request object. ImplUpdateValue can use this interface to obtain information about the current request (such as the value the user assigned to this field).

AFieldIndex is the index of this field in the set of field values that AActionRequest maintains.

ImplUpdateValue calls the CheckOrUpdateValue (see page 8128) method to write the image that AActionRequest contains to the associated graphic field.

See Also

UpdateValue (see page 7751)

Value (see page 23945)

CheckOrUpdateValue (see page 8128)

42.1.1.3.13 TCustomDataSetAdapterImageField.ImplWebImageHREF Method

Provides the underlying implementation for the WebImageHREF (see page 8136) method.

Pascal

```
function ImplWebImageHREF(var AHREF: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplWebImageHREF(AnsiString AHREF);
```

Description

The WebImageHREF (see page 8136) method calls ImplWebImageHREF to obtain the URL for an external image file that contains the value of this field.

AHREF returns the URL that identifies the image of this field.

ImplWebImageHREF returns true if it supplies a URL as the value of the AHREF parameter, false if the HTML form must request images using a set of request parameters instead.

In TCustomDataSetAdapterImageField (see page 8120), ImplWebImageHREF returns a default image at design time. At runtime, this method generates an OnGetHREF (see page 8137) event, so that you can specify a URL in an event handler.

See Also

WebImageHREF (see page 8136)

OnGetHREF (see page 8137)

42.1.1.3.14 TCustomDataSetAdapterImageField.RenderAdapterImage Method

Writes the image described by an image request to the current response message.

Pascal

```
procedure RenderAdapterImage(ARequest: IImageRequest; AResponse: IImageResponse);
```

C++

```
__fastcall RenderAdapterImage(IImageRequest ARequest, IImageResponse AResponse);
```

Description

You should not need to use the RenderAdapterImage method. This method is used internally by the ImplHandleRequest (see page 8134) method to obtain the image that is the value of this field and write it to the current response message.

ARequest is an interface from which to read the name of the requested image.

AResponse is an interface for the current response object.

RenderAdapterImage generates an OnGetImage (see page 8137) event to allow an event handler to supply an image for writing to the response message. If there is no image supplied by an OnGetImage (see page 8137) event handler, RenderAdapterImage fetches the value of the associated graphic field. Finally, this method writes the content of the image to the current response message.

See Also

ImplHandleRequest (see page 8134)

OnGetImage (see page 8137)

TGraphicField (see page 7257)

42.1.1.3.15 TCustomDataSetAdapterImageField.WebImageHREF Method

Retrieves a URL for an external image file that contains the value of this field.

Pascal

```
function WebImageHREF(var AHREF: string): Boolean;
```

C++

```
__fastcall Boolean WebImageHREF(AnsiString AHREF);
```

Description

WebImageHREF is called when server-side script uses the Image.AsHREF property. WebImageHREF returns the value of the AsHREF property (a URL that identifies the image of this field) if it is available as a simple file reference. If, instead, the field generates images dynamically in response to requests, then WebImageHREF returns false and the information needed to generate the image is supplied instead by the GetAdapterItemRequestParams (see page 8130) method.

AHREF returns the URL that identifies the image of this field.

WebImageHREF returns true if it supplies a URL as the value of the AHREF parameter, false if the HTML form must request images using a set of request parameters instead.

WebImageHREF delegates the task of providing an URL to the ImplWebImageHREF (see page 8135) method. ImplWebImageHREF (see page 8135) then obtains the URL by generating an OnGetHREF (see page 8137) event.

See Also

ImplWebImageHREF (see page 8135)

OnGetHREF (see page 8137)

GetAdapterItemRequestParams (see page 8130)

IWebImageHREF (see page 19765)

42.1.1.4 TCustomDataSetAdapterImageField Events

The events of the TCustomDataSetAdapterImageField class are listed here.

Public Events

	Name	Description
	OnGetHREF (see page 8137)	Occurs when the WebSnap application needs the URL that clients use to request this field's image.
	OnGetImage (see page 8137)	Occurs when the WebSnap application receives a request for this field's image.

42.1.1.4.1 TCustomDataSetAdapterImageField.OnGetHREF Event

Occurs when the WebSnap application needs the URL that clients use to request this field's image.

Pascal

```
property OnGetHREF: TImageFieldGetHREFEvent;
```

C++

```
__property TImageFieldGetHREFEvent OnGetHREF;
```

Description

Write an OnGetHREF event handler to specify the URL that addresses this field's image. This event supplies the URL that is the value of the field's Image.AsHREF property in server-side script.

If you do not supply an URL using an OnGetHREF event handler, the dataset adapter image field constructs one automatically based on the request parameters for the image file.

See Also

OnGetImage (see page 8137)

42.1.1.4.2 TCustomDataSetAdapterImageField.OnGetImage Event

Occurs when the WebSnap application receives a request for this field's image.

Pascal

```
property OnGetImage: TImageDataSetFieldGetImageEvent;
```

C++

```
__property TImageDataSetFieldGetImageEvent OnGetImage;
```

Description

Write an OnGetImage event handler to supply the image that this adapter field returns in response to incoming request messages. This event occurs when a request arrives that is addressed to the URL that the OnGetHREF (see page 8137) event handler supplies. If you do not provide an OnGetImage event handler, this adapter field retrieves the image from its associated graphic field object.

See Also

OnGetHREF (see page 8137)

FindField (see page 7737)

ContentType (see page 15209)

42.1.2 TDataSetAdapterImageField Class

tdatasetadapterimagefield is the adapter field object that represents a graphic field from a database.

Class Hierarchy

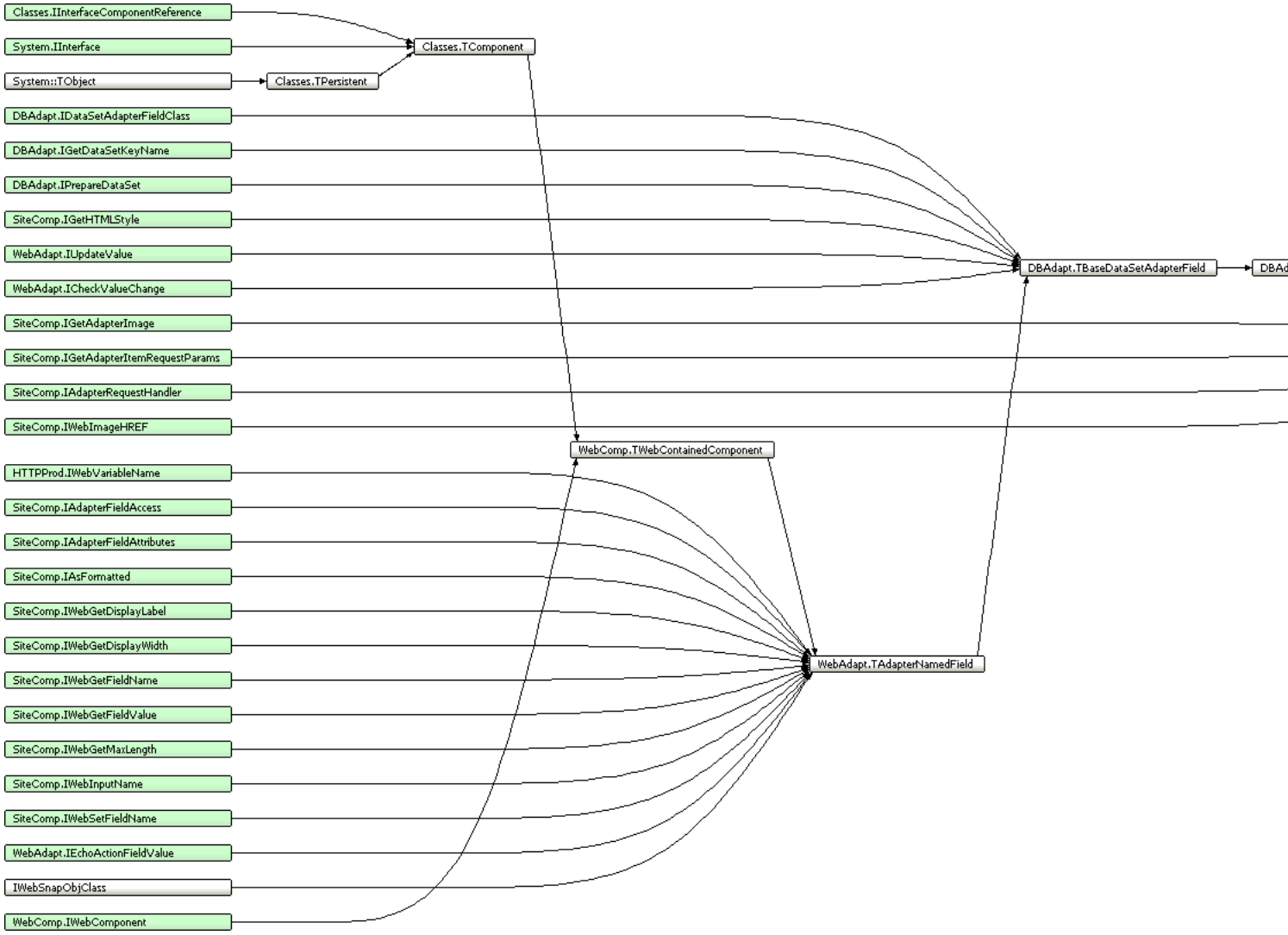

Pascal

```
TDataSetAdapterImageField = class(TCustomDataSetAdapterImageField);
```

C++

```
class TDataSetAdapterImageField : public TCustomDataSetAdapterImageField;
```

File

DBAdaptImg

Description

Add tdatasetadapterimagefield as a child of a TDataSetAdapter component to represent a graphic field in the dataset associated with the dataset adapter. tdatasetadapterimagefield represents a single field from a database table, and enables server-side script to refer to or update that field's value.

tdatasetadapterimagefield supports several properties that can be used in server-side script. These include:

Image, which represents the value of the field as an image. You can specify a URL for this image using the Image.AsHREF property.

Value and DisplayText, both of which represent the value of the field as a text string.

DisplayLabel, which provides a text label that can be used to label HTML controls that represent the field.

Required, which indicates whether the graphic field requires a value.

Visible, which indicates whether HTML forms should include a control for this field.

See Also

TDataSetAdapterMemoField (see page 7996)

TDataSetAdapterField (see page 7960)

TDataSetAdapter (see page 7872)

TGraphicField (see page 7257)

TAdapterDisplayField (see page 25109)

TAdapterDisplayColumn (see page 25099)

TAdapterEditColumn (see page 25132)

42.1.2.1 TDataSetAdapterImageField Members

The following tables list the members exposed by TDataSetAdapterImageField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckOrigValue (see page 7736)	Ensures that the current value of this field is the same as the original value.
	CheckOrUpdateValue (see page 8128)	Checks whether a specified request contains a valid value and, if requested, uses that value to update the database.
	CheckValueChange (see page 7737)	Indicates whether the current value of this field is the same as the original value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.

	CreateRequestContext (see page 8129)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 7738)	Indicates whether references to this field in HTML forms must be qualified with the name of the parent adapter.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 7738)	Returns the value of the Adapter (see page 7752) property.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterImage (see page 8130)	Returns an interface that can be used to access the Image property of this adapter field.
	GetAdapterItemRequestParams (see page 8130)	Provides a unique identifier and a set of name/value pairs that can be used to identify this field's image file.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 7738)	Returns the dataset associated with this field's parent adapter.
V	GetDataSetFieldValue (see page 7739)	Returns the value of the associated dataset field.
V	GetDataSetFieldValue (see page 8131)	Returns the value of the associated dataset field.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 7739)	Returns a string value that suggests an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 7740)	Returns the HTML display style type for this adapter field.
V	GetDisplayStyleType (see page 8131)	Returns the HTML display style type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 7741)	Returns a string value that suggests an HTML editing style for this adapter field.
V	GetInputStyleType (see page 7742)	Returns the style of the HTML control to use for editing this adapter field.
V	GetInputStyleType (see page 8132)	Returns the style of the HTML control to use for editing this adapter field.
	GetKeyName (see page 7742)	Returns the name of the associated dataset field if it appears in the index (key) that the parent dataset adapter uses.
	GetKeyParams (see page 7743)	Returns a list of fields that the parent dataset adapter uses to locate records, along with their current values in that adapter's dataset.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetReadOnly (see page 7743)	Indicates whether the adapter field represents a read-only database field.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 7744)	Returns a string value that indicates what function a control that represents this field should perform.
V	GetViewModeType (see page 7744)	Returns the view mode for the function a control that represents this field should perform.

	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HandleRequest (see page 8132)	Handles incoming requests for the image value of this field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 7745)	Provides the underlying implementation for the AsFormatted method.
V	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
V	ImplCheckOrigValue (see page 7745)	Provides the underlying implementation for the CheckOrigValue (see page 7736) method.
V	ImplCheckValueChange (see page 7746)	Provides the underlying implementation for the CheckValueChange (see page 7737) method.
V	ImplCheckValueChange (see page 8133)	Provides the underlying implementation for the CheckValueChange method.
	ImplCreateRequestContext (see page 8134)	Provides the underlying implementation for the CreateRequestContext (see page 8129) method.
V	ImplGetDisplayLabel (see page 7747)	Provides the underlying implementation for the GetDisplayLabel method.
V	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 7747)	Provides the underlying implementation for the GetDisplayWidth method.
V	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
V	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 7747)	Provides the underlying implementation for the GetRequired method.
V	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 7748)	Provides the underlying implementation for the GetValue method.
V	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
V	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 7748)	Provides the underlying implementation for the GetVisible method.
V	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHandleRequest (see page 8134)	Provides the underlying implementation for the HandleRequest (see page 8132) method.
V	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 7749)	Provides the underlying implementation for the SetFieldName method.
V	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
V	ImplUpdateValue (see page 7749)	Provides the underlying implementation for the UpdateValue (see page 7751) method.
V	ImplUpdateValue (see page 8135)	Provides the underlying implementation for the UpdateValue method.
V	ImplWebImageHREF (see page 8135)	Provides the underlying implementation for the WebImageHREF (see page 8136) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 7750)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	PrepareDataSet (see page 7750)	Responds when the parent dataset adapter prepares its dataset.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.

	RenderAdapterImage (see page 8136)	Writes the image described by an image request to the current response message.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSetField (see page 7751)	Sets the value of the DataSetField (see page 7752) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 7751)	Updates the adapter field when the user assigns a new value.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebImageHREF (see page 8136)	Retrieves a URL for an external image file that contains the value of this field.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 7752)	Indicates the dataset adapter that uses this object to represent one of its fields.
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 7754)	Occurs when the adapter field needs to supply its formatted value as a string that can appear in HTML forms.
	OnGetValue (see page 7754)	Occurs when the adapter field needs to supply its value as a Variant.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 7735)	Creates and initializes an instance of TBaseDataSetAdapterField (see page 7729).
V	Create (see page 8127)	Creates and initializes an instance of TCustomDataSetAdapterImageField (see page 8120).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 7737)	Returns the dataset field whose value this adapter field represents.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterImage (see page 19691)	This is GetAdapterImage, a member of class IGetAdapterImage.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetKeyName (see page 8105)	This is GetKeyName, a member of class IGetDataSetKeyName.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.

	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PrepareDataSet (see page 8109)	This is PrepareDataSet, a member of class IPrepareDataSet.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.
	WebImageHREF (see page 19766)	Retrieves a URL for an external image file.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.

	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	FieldFlags (see page 7752)	Indicates various attributes of the adapter field.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
R	KeyName (see page 8105)	This is KeyName, a member of class IGetDataSetKeyName.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetHREF (see page 8147)	Occurs when the WebSnap application needs the URL that clients use to request this field's image.
	OnGetImage (see page 8148)	Occurs when the WebSnap application receives a request for this field's image.

Published Properties

	Name	Description
	DataSetField (see page 8145)	Specifies the name of the dataset field that this adapter field represents.
	FieldModes (see page 8146)	Indicates what dataset adapter modes this field supports.
	ModifyAccess (see page 8146)	Identifies the rights required to update this field.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	ViewAccess (see page 8147)	Identifies the rights required to view this field.

42.1.2.2 TDataSetAdapterImageField Properties

The properties of the TDataSetAdapterImageField class are listed here.

Published Properties

	Name	Description
	DataSetField (see page 8145)	Specifies the name of the dataset field that this adapter field represents.
	FieldModes (see page 8146)	Indicates what dataset adapter modes this field supports.
	ModifyAccess (see page 8146)	Identifies the rights required to update this field.
	ViewAccess (see page 8147)	Identifies the rights required to view this field.

42.1.2.2.1 TDataSetAdapterImageField.DataSetField Property

Specifies the name of the dataset field that this adapter field represents.

Pascal

```
property DataSetField: string;
```

C++

```
__property AnsiString DataSetField;
```

Description

DataSetField is the FieldName of the TField object whose value this adapter field represents. DataSetField can be the name of a simple field in a database table, the name of a subfield of an object field that has been qualified by the parent field's name, or the name of an aggregated field.

DataSetField is the same as the name of this adapter field in server-side script, except that the name of the adapter field omits any illegal characters (the only legal characters are the letters between a and z (or A and Z), digits, and the underline character).

If DataSetField is an empty string, this adapter field does not have a value, and the FindField method returns nil (Delphi) or NULL (C++).

See Also

FieldName (see page 7164)

FindField (see page 7737)

42.1.2.2.2 TDataSetAdapterImageField.FieldModes Property

Indicates what dataset adapter modes this field supports.

Pascal

```
property FieldModes: TDataSetAdapterModes;
```

C++

```
__property TDataSetAdapterModes FieldModes;
```

Description

FieldModes is a set of all the modes in which this adapter field can be used. If the parent dataset adapter is in a mode that is not included in this set, any forms generated using that adapter do not include a control to represent this field. Thus, for example, to make a field read-only, you can exclude amInsert and amEdit from FieldModes. Then, the field appears when the user is viewing or querying the dataset, but not when the user is editing or inserting a record.

See Also

CanModify (see page 7154)

Adapter (see page 7752)

Mode (see page 7810)

GetDisplayStyle (see page 7739)

GetInputStyle (see page 7741)

GetViewMode (see page 7744)

42.1.2.2.3 TDataSetAdapterImageField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property AnsiString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

hasmodifyaccess (see page 24162)

42.1.2.2.4 TDataSetAdapterImageField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property AnsiString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

HasViewAccess (see page 24162)

42.1.2.3 TDataSetAdapterImageField Events

The events of the TDataSetAdapterImageField class are listed here.

Published Events

	Name	Description
	OnGetHREF (see page 8147)	Occurs when the WebSnap application needs the URL that clients use to request this field's image.
	OnGetImage (see page 8148)	Occurs when the WebSnap application receives a request for this field's image.

42.1.2.3.1 TDataSetAdapterImageField.OnGetHREF Event

Occurs when the WebSnap application needs the URL that clients use to request this field's image.

Pascal

```
property OnGetHREF: TImageFieldGetHREFEvent;
```

C++

```
__property TImageFieldGetHREFEvent OnGetHREF;
```

Description

Write an OnGetHREF event handler to specify the URL that addresses this field's image. This event supplies the URL that is the value of the field's Image.AsHREF property in server-side script.

If you do not supply an URL using an OnGetHREF event handler, the dataset adapter image field constructs one automatically based on the request parameters for the image file.

See Also

OnGetImage (see page 8137)

42.1.2.3.2 TDataSetAdapterImageField.OnGetImage Event

Occurs when the WebSnap application receives a request for this field's image.

Pascal

```
property OnGetImage: TImageDataSetFieldGetImageEvent;
```

C++

```
__property TImageDataSetFieldGetImageEvent OnGetImage;
```

Description

Write an OnGetImage event handler to supply the image that this adapter field returns in response to incoming request messages. This event occurs when a request arrives that is addressed to the URL that the OnGetHREF event handler supplies. If you do not provide an OnGetImage event handler, this adapter field retrieves the image from its associated graphic field object.

See Also

OnGetHREF (see page 8137)

FindField (see page 7737)

ContentType (see page 15209)

42.2 Types

The following table lists types in this documentation.

Types

Name	Description
TImageDataSetFieldGetImageEvent (see page 8149)	This is type DBAdaptImg.TImageDataSetFieldGetImageEvent.

42.2.1 DBAdaptImg.TImageDataSetFieldGetImageEvent Type

Pascal

```
TImageDataSetFieldGetImageEvent = procedure (Sender: TObject; var MimeType: string; var Image:
TStream; var Owned: Boolean) of object;
```

C++

```
(Sender: TObject; var MimeType: string; var Image: TStream; var Owned: Boolean) (
TImageDataSetFieldGetImageEvent)();
```

File

DBAdaptImg

Description

This is type DBAdaptImg.TImageDataSetFieldGetImageEvent.

43 DBBdeWeb Namespace

This is namespace DBBdeWeb.

Topics

Name	Description
Classes (see page 8152)	The following table lists classes in this documentation.

Classes

	Name	Description
	TQueryTableProducer (see page 8152)	TQueryTableProducer assembles a sequence of HTML commands to display the result set of a query in a tabular format.

43.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TQueryTableProducer (see page 8152)	TQueryTableProducer assembles a sequence of HTML commands to display the result set of a query in a tabular format.

43.1.1 TQueryTableProducer Class

TQueryTableProducer assembles a sequence of HTML commands to display the result set of a query in a tabular format.

Class Hierarchy

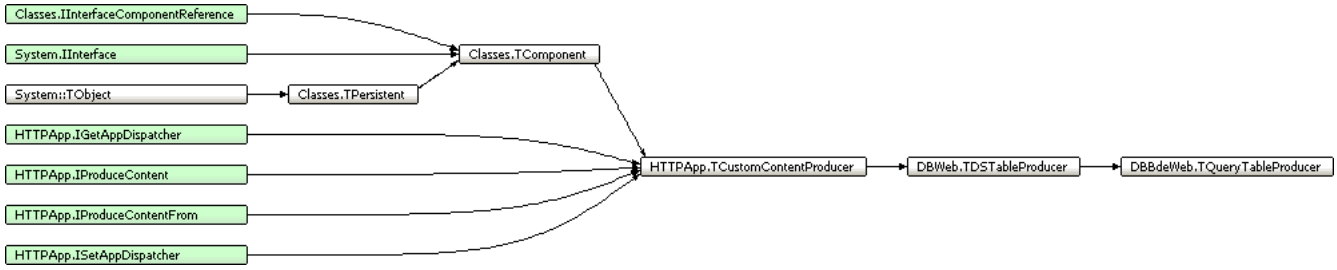

Pascal

```
TQueryTableProducer = class(TDSTableProducer);
```

C++

```
class TQueryTableProducer : public TDSTableProducer;
```

File

DBBdeWeb

Description

Use TQueryTableProducer to produce an HTML table for the records of a parameterized query. The TQuery object obtains its parameters from an HTTP request message. If the request is a GET message, the parameters for the query are the QueryFields property of the request object. If the request is a POST message, the parameters for the query are the ContentFields property of the request object.

To produce an HTML table for a query that has no parameters, use the TDataSetTableProducer object instead.

Note: See the WebServ demo for an example of how to use TQueryTableProducer.

See Also

TDataSetTableProducer (see page 10277)

TQuery (see page 10059)

TWebRequest (see page 15161)

TWebRequest (see page 15161)

TWebRequest (see page 15161)

The Web Module

Using TQueryTableProducer

43.1.1.1 TQueryTableProducer Members

The following tables list the members exposed by TQueryTableProducer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AcquireLayoutLock (see page 10296)	Calls BeginLayout (see page 10297) when the HTML table is not currently being laid out.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BeginLayout (see page 10297)	Increments a counter that indicates the number of times the computation of the column order was begun but not completed.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ColumnHeader (see page 10298)	Returns the string of HTML <TH> tags that define the column headers for the table.
	DefineFieldMap (see page 10298)	Builds the Fields array in DataLink (see page 10309).
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoCreateContent (see page 10299)	This is DoCreateContent, a member of class TDSTableProducer.
	DoFormatCell (see page 10299)	Generates an OnFormatCell (see page 10314) event.
	DoGetCaption (see page 10300)	Generates an OnGetCaption event.
	EndLayout (see page 10300)	Decrements a counter that indicates the number of times the computation of the column order has been begun but not completed.
	FormatCell (see page 10301)	Returns an HTML tag of a specified type that reflects a set of specified options.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDataSet (see page 8158)	Provides the abstract prototype of a method to access the DataSet property.
	GetDataSource (see page 10302)	Returns the value of the DataSource (see page 10309) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	LayoutChanged (see page 10303)	Causes the Columns (see page 10308) property to be recomputed when the dataset changes or changes are made to individual column objects.

	LinkActive (see page 10303)	Recomputes the Columns (see page 10308) property when the dataset becomes active.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 8158)	Adjusts the properties of TDSTableProducer when its helper objects are destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RowHeader (see page 10304)	Returns the HTML <TR> tag that defines each row of the HTML table.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetColumns (see page 10304)	Sets the Columns (see page 10308) property.
V	SetDataSet (see page 8158)	Provides the abstract prototype of a method to set the DataSet property.
	SetDataSource (see page 10305)	Sets the DataSource (see page 10309) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StoreColumns (see page 10305)	Indicates whether the Columns (see page 10308) property is stored when the TDSTableProducer (see page 10290) is saved.
	TableCaption (see page 10306)	Returns the HTML <Caption (see page 10307)> tag that defines the caption for the HTML table.
	TableHeader (see page 10306)	Returns the HTML <Table> tag for the HTML table.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	DataLink (see page 10309)	Manages the link to the dataset.
	DataSource (see page 10309)	Represents the link to the DataSet (see page 10309) of the TDSTableProducer (see page 10290).
R	InternalDataSource (see page 10311)	Represents the link to the dataset.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	BeginUpdate (see page 10297)	Increments a counter that indicates the number of changes to the contents of the HTML table that have been initiated but not completed.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Content (see page 8157)	Returns the HTML representation of the query results.
V	ContentFromStream (see page 15107)	Returns the content produced in response to input read from a stream.
V	ContentFromString (see page 15107)	Returns the content produced in response to an input string.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 10295)	Creates an instance of TDSTableProducer (see page 10290).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 10295)	Destroys an instance of TDSTableProducer (see page 10290).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 10301)	Decrements a counter that indicates the number of changes to the contents of the HTML table.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DataSet (see page 10309)	Specifies the dataset from which the data in the HTML table is taken.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.
	Editor (see page 10310)	Writes changes made in the property editor at design time to the TDSTableProducer (see page 10290) object.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCreateContent (see page 8164)	Occurs in descendants of TDSTableProducer when the Content method needs to generate the HTML table image.
	OnFormatCell (see page 8164)	Occurs when the HTML for a specified cell specified is about to be generated.
	OnGetTableCaption (see page 8165)	Occurs when the HTML table producer is about to generate the HTML for the table caption.

Published Properties

	Name	Description
	Caption (see page 8159)	Specifies the caption string that appears with the HTML table.
	CaptionAlignment (see page 8159)	Specifies where the Caption string should appear relative to the HTML table.
	Columns (see page 8160)	Describes the field bindings and display attributes of the columns in the HTML table.
	Footer (see page 8160)	Specifies a set of HTML commands that appear in the HTML document following the HTML table.
	Header (see page 8161)	Specifies a set of HTML commands that appear in the HTML document preceding the HTML table.
	MaxRows (see page 8161)	Specifies the maximum number of rows that appear in the HTML table.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Query (see page 8162)	Specifies the TQuery object that TQueryTableProducer (see page 8152) runs to obtain its data.

	RowAttributes (see page 8162)	Specifies the display attributes of all the rows in the HTML table.
	TableAttributes (see page 8163)	Specifies the display attributes that pertain to the entire HTML table produced by TDSTableProducer.
	Tag (see page 2808)	Stores an integer value as part of a component.

43.1.1.2 TQueryTableProducer Methods

The methods of the TQueryTableProducer class are listed here.

Protected Methods

	Name	Description
	GetDataSet (see page 8158)	Provides the abstract prototype of a method to access the DataSet property.
	Notification (see page 8158)	Adjusts the properties of TDSTableProducer when its helper objects are destroyed.
	SetDataSet (see page 8158)	Provides the abstract prototype of a method to set the DataSet property.

Public Methods

	Name	Description
	Content (see page 8157)	Returns the HTML representation of the query results.

43.1.1.2.1 TQueryTableProducer.Content Method

Returns the HTML representation of the query results.

Pascal

```
function Content: string; override;
```

C++

```
virtual __fastcall AnsiString Content();
```

Description

Call Content to run the query with the parameters taken from the HTTP request message and retrieve the resulting HTML representation. If the MethodType of the request is mtGet, the parameters for the query are the QueryFields property of the request. If the MethodType of the request is mtPost, the parameters for the query are the ContentFields property of the request.

After running the query, Content calls the OnCreateContent (see page 8164) event handler. If the OnCreateContent (see page 8164) event handler indicates that the Content method should continue, Content returns the Header (see page 8161) property, followed by an HTML table image of the records in the query, followed by the Footer (see page 8160) property. If the OnCreateContent (see page 8164) event handler indicates that the Content method should not continue, Content returns an empty string.

The string returned by Content can be used as the Content property of the response object associated with the request message.

See Also

Footer (see page 10310)

Header (see page 10311)

OnCreateContent (see page 10313)

ContentFields (see page 15177)

MethodType

QueryFields (see page 15187)

Content (see page 15207)

Using TQueryTableProducer

43.1.1.2.2 TQueryTableProducer.GetDataSet Method

Provides the abstract prototype of a method to access the DataSet property.

Pascal

```
function GetDataSet: TDataSet; override;
```

C++

```
virtual __fastcall TDataSet GetDataSet();
```

Description

Descendants of TDSTableProducer implement the GetDataSet method to access their specific implementation of the DataSet property. Descendants of TDSTableProducer restrict the DataSet property to different classes of TDataSet objects.

See Also

DataSet (see page 10309)

SetDataSet (see page 10305)

43.1.1.2.3 TQueryTableProducer.Notification Method

Adjusts the properties of TDSTableProducer when its helper objects are destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification overrides the inherited method to respond to messages that the DataSource or one of its field objects is about to be destroyed. Notification sets the DataSource property to nil (Delphi) or NULL (C++), or unbinds the column that references the field component.

See Also

DataSource (see page 10309)

Columns (see page 10308)

43.1.1.2.4 TQueryTableProducer.SetDataSet Method

Provides the abstract prototype of a method to set the DataSet property.

Pascal

```
procedure SetDataSet(ADataSet: TDataSet); override;
```

C++

```
virtual __fastcall SetDataSet(TDataSet ADataSet);
```

Description

Descendants of TDSTableProducer override SetDataSet to ensure that the DataSet property can only be set to an appropriate descendant of TDataSet.

See Also

DataSet (see page 10309)

GetDataSet (see page 10302)

43.1.1.3 TQueryTableProducer Properties

The properties of the TQueryTableProducer class are listed here.

Published Properties

	Name	Description
	Caption (see page 8159)	Specifies the caption string that appears with the HTML table.
	CaptionAlignment (see page 8159)	Specifies where the Caption string should appear relative to the HTML table.
	Columns (see page 8160)	Describes the field bindings and display attributes of the columns in the HTML table.
	Footer (see page 8160)	Specifies a set of HTML commands that appear in the HTML document following the HTML table.
	Header (see page 8161)	Specifies a set of HTML commands that appear in the HTML document preceding the HTML table.
	MaxRows (see page 8161)	Specifies the maximum number of rows that appear in the HTML table.
	Query (see page 8162)	Specifies the TQuery object that TQueryTableProducer (see page 8152) runs to obtain its data.
	RowAttributes (see page 8162)	Specifies the display attributes of all the rows in the HTML table.
	TableAttributes (see page 8163)	Specifies the display attributes that pertain to the entire HTML table produced by TDSTableProducer.

43.1.1.3.1 TQueryTableProducer.Caption Property

Specifies the caption string that appears with the HTML table.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Set Caption to add a caption to the HTML table that represents the dataset. Caption indicates the text that should appear inside the HTML CAPTION tag.

See Also

CaptionAlignment (see page 10308)

OnGetTableCaption (see page 10314)

Embedding Tables in HTML Documents

43.1.1.3.2 TQueryTableProducer.CaptionAlignment Property

Specifies where the Caption string should appear relative to the HTML table.

Pascal

```
property CaptionAlignment: THTMLCaptionAlignment;
```

C++

```
__property THTMLCaptionAlignment CaptionAlignment;
```

Description

Set CaptionAlignment to specify whether the Caption should appear above or below the HTML table.

CaptionAlignment is the value of the ALIGN option on the HTML CAPTION tag.

See Also

Caption (see page 10307)

Embedding Tables in HTML Documents

43.1.1.3.3 TQueryTableProducer.Columns Property

Describes the field bindings and display attributes of the columns in the HTML table.

Pascal

```
property Columns: THTMLTableColumns;
```

C++

```
__property THTMLTableColumns Columns;
```

Description

Use Columns to read or set the field bindings and display attributes of the columns in the HTML table. Each THTMLTableColumn object specifies the display attributes of the corresponding column. Each column object also specifies the name of the field the column represents. The field binding of a column designates a field within the dataset specified by the DataSet property.

Columns can be set at design time through a property editor, or programmatically at runtime.

See Also

DataSet (see page 10309)

DefineFieldMap (see page 10298)

Editor (see page 10310)

RowAttributes (see page 10312)

SetColumns (see page 10304)

StoreColumns (see page 10305)

TableAttributes (see page 10313)

THTMLTableColumn (see page 10318)

THTMLTableColumns (see page 10327)

Specifying the Columns

43.1.1.3.4 TQueryTableProducer.Footer Property

Specifies a set of HTML commands that appear in the HTML document following the HTML table.

Pascal

```
property Footer: TStrings;
```

C++

```
__property TStrings Footer;
```

Description

Set Footer to embed the representation of DataSet in a larger document. The Content method of TDSTableProducer descendants returns any Header commands, followed by the HTML table that represents the DataSet, followed by any Footer commands.

See Also

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

DataSet (see page 10309)

Header (see page 10311)

Embedding Tables in HTML Documents

43.1.1.3.5 TQueryTableProducer.Header Property

Specifies a set of HTML commands that appear in the HTML document preceding the HTML table.

Pascal

```
property Header: TStrings;
```

C++

```
__property TStrings Header;
```

Description

Set Header to embed the representation of DataSet in a larger document. The Content method of TDSTableProducer descendants returns any Header commands, followed by the HTML table that represents the DataSet, followed by any Footer commands.

See Also

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

DataSet (see page 10309)

Footer (see page 10310)

Embedding Tables in HTML Documents

43.1.1.3.6 TQueryTableProducer.MaxRows Property

Specifies the maximum number of rows that appear in the HTML table.

Pascal

```
property MaxRows: Integer;
```

C++

```
__property int MaxRows;
```

Description

Set MaxRows to change the upper limit on the size of the HTML table produced to represent the DataSet. MaxRows is initialized to 20.

If DataSet contains more than MaxRows records, only the first MaxRows records appear in the HTML commands produced by Content. If DataSet contains fewer than MaxRows records, the HTML table produced by TDSTableProducer will contain only enough rows to display all the records in DataSet.

See Also

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

DataSet (see page 10309)

Specifying the Row Attributes

43.1.1.3.7 TQueryTableProducer.Query Property

Specifies the TQuery object that TQueryTableProducer (see page 8152) runs to obtain its data.

Pascal

```
property Query: TQuery;
```

C++

```
__property TQuery Query;
```

Description

Set Query to specify the query to run to obtain the data for the HTML table. Use Query to set the query for the TQueryTableProducer (see page 8152) rather than the DataSource property: TQueryTableProducer (see page 8152) generates its data source internally.

The individual field bindings specified by the THTMLTableColumn objects in the Columns (see page 8160) property refer to fields in DataSet.

Note: Query is identical to the DataSet property, except Query does not use the TDataSet base class of the query object.

See Also

Columns (see page 10308)

DataSet (see page 10309)

DataSource (see page 10309)

FieldName (see page 10326)

Using TQueryTableProducer

43.1.1.3.8 TQueryTableProducer.RowAttributes Property

Specifies the display attributes of all the rows in the HTML table.

Pascal

```
property RowAttributes: THTMLTableRowAttributes;
```

C++

```
__property THTMLTableRowAttributes RowAttributes;
```

Description

Set RowAttributes to a THTMLTableRowAttributes object that describes the display attributes of the rows in the HTML table. Setting RowAttributes calls the Assign method of the THTMLTableRowAttributes object stored internally by TDSTableProducer. Thus, the memory for any row attributes object assigned to the RowAttributes property is never owned or freed by TDSTableProducer.

Note: Not all the properties of the row attributes object are copied when one row attributes object is assigned to another. To ensure that all the attributes represented by RowAttributes are set appropriately, read RowAttributes and set the individual properties of the THTMLTableRowAttributes object.

See Also

TableAttributes (see page 10313)

Title (see page 10327)

AssignTo (see page 15297)

Specifying the Row Attributes

43.1.1.3.9 TQueryTableProducer.TableAttributes Property

Specifies the display attributes that pertain to the entire HTML table produced by TDSTableProducer.

Pascal

```
property TableAttributes: THTMLTableAttributes;
```

C++

```
__property THTMLTableAttributes TableAttributes;
```

Description

Set TableAttributes to a THTMLTableAttributes object that describes the display attributes for the HTML TABLE tag. Setting TableAttributes calls the Assign method of the THTMLTableAttributes object stored internally by TDSTableProducer. Thus, the memory for any table attributes object assigned to the TableAttributes property is never owned or freed by TDSTableProducer.

Note: Not all the properties of the table attributes object are copied when one table attributes object is assigned to another. To ensure that all the attributes represented by TableAttributes are set appropriately, read TableAttributes and set the individual properties of the THTMLTableAttributes object.

See Also

RowAttributes (see page 10312)

AssignTo (see page 15291)

Title (see page 10327)

Specifying the Table Attributes

43.1.1.4 TQueryTableProducer Events

The events of the TQueryTableProducer class are listed here.

Published Events

	Name	Description
	OnCreateContent (see page 8164)	Occurs in descendants of TDSTableProducer when the Content method needs to generate the HTML table image.
	OnFormatCell (see page 8164)	Occurs when the HTML for a specified cell specified is about to be generated.
	OnGetTableCaption (see page 8165)	Occurs when the HTML table producer is about to generate the HTML for the table caption.

43.1.1.4.1 TQueryTableProducer.OnCreateContent Event

Occurs in descendants of TDSTableProducer when the Content method needs to generate the HTML table image.

Pascal

```
property OnCreateContent: TCreateContentEvent;
```

C++

```
__property TCreateContentEvent OnCreateContent;
```

Description

Write an OnCreateContent event handler to conditionally prevent the Content method from producing any HTML commands. Set the Continue parameter to false to make Content return an empty string.

See Also

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

docreatecontent (see page 10299)

43.1.1.4.2 TQueryTableProducer.OnFormatCell Event

Occurs when the HTML for a specified cell specified is about to be generated.

Pascal

```
property OnFormatCell: THTMLFormatCellEvent;
```

C++

```
__property THTMLFormatCellEvent OnFormatCell;
```

Description

Write an OnFormatCell event handler to change the appearance or contents of individual cells in the HTML table from the default values produced by TDSTableProducer.

See Also

Columns (see page 10308)

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

DoFormatCell (see page 10299)

DisplayText (see page 7162)

Align (see page 10324)

BgColor (see page 10325)

Custom (see page 10325)

VAlign (see page 10327)

43.1.1.4.3 TQueryTableProducer.OnGetTableCaption Event

Occurs when the HTML table producer is about to generate the HTML for the table caption.

Pascal

```
property OnGetTableCaption: THTMLGetTableCaptionEvent;
```

C++

```
__property THTMLGetTableCaptionEvent OnGetTableCaption;
```

Description

Write an OnGetTableCaption event handler to change the value or placement of the HTML table caption. The Caption and Alignment parameters are originally set to the values of the Caption and CaptionAlignment properties, respectively. The OnGetTableCaption event handler can change these values to alter the caption for the HTML table returned by the Content method.

See Also

Caption (see page 10307)

CaptionAlignment (see page 10308)

Content (see page 15106)

DoGetCaption (see page 10300)

44 DBCGrids Namespace

This is namespace DBCGrids.

Topics

Name	Description
Classes (see page 8168)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 8257)	The following table lists structs, records, enums in this documentation.
Types (see page 8259)	The following table lists types in this documentation.

Classes

	Name	Description
	TDBCtrlGrid (see page 8168)	TDBCtrlGrid displays records from a data source in a free-form layout.
	TDBCtrlGridLink (see page 8240)	TDBCtrlGridLink helps manage data events for a TDBCtrlGrid (see page 8168) object.
	TDBCtrlPanel (see page 8244)	TDBCtrlPanel represents a panel in a TDBCtrlGrid (see page 8168) object.

Structs, Records, Enums

	Name	Description
	TDBCtrlGridBorder (see page 8257)	This is record DBCGrids.TDBCtrlGridBorder.
	TDBCtrlGridKey (see page 8257)	This is record DBCGrids.TDBCtrlGridKey.
	TDBCtrlGridOrientation (see page 8258)	This is record DBCGrids.TDBCtrlGridOrientation.

Types

Name	Description
TPaintPanelEvent (see page 8259)	This is type DBCGrids.TPaintPanelEvent.

44.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TDBCtrlGrid (see page 8168)	TDBCtrlGrid displays records from a data source in a free-form layout.
	TDBCtrlGridLink (see page 8240)	TDBCtrlGridLink helps manage data events for a TDBCtrlGrid (see page 8168) object.
	TDBCtrlPanel (see page 8244)	TDBCtrlPanel represents a panel in a TDBCtrlGrid (see page 8168) object.

44.1.1 TDBCtrlGrid Class

TDBCtrlGrid displays records from a data source in a free-form layout.

Class Hierarchy

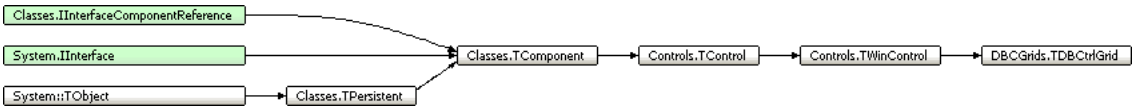

Pascal

```
TDBCtrlGrid = class(TWinControl);
```

C++

```
class TDBCtrlGrid : public TWinControl;
```

File

dbcgrids

Description

Put a TDBCtrlGrid object on a form to display and edit the records from a database table or query. Unlike the TDBGrid object, which displays each record in a single row, TDBCtrlGrid allows a free-form layout for each record in the dataset. Each record is displayed in its own panel; Design one panel at design time and TDBCtrlGrid replicates that panel for each record displayed.

At runtime, users can use the database navigator (TDBNavigator) to move through data in the TDBCtrlGrid object, and to insert, delete, and edit the data. Edits made using the data grid are not posted to the underlying dataset until the user moves to a different record or closes the application.

See Also

TDBCtrlPanel (see page 8244)

TDBGrid (see page 9668)

TDBNavigator (see page 9230)

Creating a Grid That Contains Other Data-aware Controls

Using Common Data Control Features

Grids

44.1.1.1 TDBCtrlGrid Members

The following tables list the members exposed by TDBCtrlGrid.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.

	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 8180)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 8181)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.

	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
V	GetChildParent (see page 8183)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
V	GetChildren (see page 8183)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.

	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintPanel (see page 8185)	This is PaintPanel, a member of class TDBCtrlGrid.
V	PaintWindow (see page 8185)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 8185)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.

	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.

	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	Panel (see page 8199)	This is Panel, a member of class TDBCtrlGrid.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 8179)	Creates and initializes an instance of TDBCtrlGrid (see page 8168).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 8179)	Destroys an instance of TDBCtrlGrid (see page 8168).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DoKey (see page 8181)	Implements the responses to a number of standard keystrokes.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 8182)	Executes an action with the DB (see page 6725) control grid as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 8183)	Overrides the default to remove all child controls from the tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.

	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
V	KeyDown (see page 8184)	Performs special processing when a key is pressed.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 8186)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
V	UpdateAction (see page 8186)	Updates an action to reflect the current state of the grid.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.

R	Canvas (see page 8193)	Indicates the drawing surface used to draw the TDBCtrlGridObject.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EditMode (see page 8197)	Determines whether the TDBCtrlGrid (see page 8168) can be used to insert, delete, or edit records.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
R	PanelCount (see page 8200)	Specifies the number of records visible in the grid.
	PanelIndex (see page 8201)	Specifies the index of the panel in the grid that represents the current record.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 8213)	Occurs when the user clicks the control.
	OnDblClick (see page 8215)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 8215)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 8218)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 8220)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 8221)	Occurs when a control receives the input focus.
	OnExit (see page 8224)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 8225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 8228)	Occurs when a key is pressed.
	OnKeyUp (see page 8229)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 8231)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 8232)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 8234)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 8234)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 8234)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 8237)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaintPanel (see page 8239)	Occurs when TDBCtrlGrid (see page 8168) needs to draw a panel in the grid.
	OnStartDrag (see page 8240)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 8188)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AllowDelete (see page 8191)	Determines whether the user can delete the current record from the dataset by pressing Ctrl+Delete.
	AllowInsert (see page 8192)	Determines whether the user can insert new records with the Insert key or append new records with the Ctrl+Insert key combination.
	Anchors (see page 8192)	Specifies how the control is anchored to its parent.
	ColCount (see page 8193)	Specifies the number of columns of panels shown in the grid.
	Color (see page 8193)	Specifies the background color of the control.
	Constraints (see page 8194)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DataSource (see page 8195)	Identifies the link to the dataset where the TDBCtrlGrid (see page 8168) object finds its data.
	DragCursor (see page 8195)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 8195)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 8197)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 8198)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.

	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Orientation (see page 8199)	Determines the order in which records from the dataset are displayed in the TDBCtrlGrid (see page 8168).
	PanelBorder (see page 8200)	Determines whether each panel is drawn with a raised border around it.
	PanelHeight (see page 8200)	Specifies the height (in pixels) of each panel.
	PanelWidth (see page 8201)	Specifies the width (in pixels) of each panel.
	ParentColor (see page 8201)	Determines where a control looks for its color information.
	ParentFont (see page 8202)	Determines where a control looks for its font information.
	ParentShowHint (see page 8202)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 8203)	Identifies the pop-up menu associated with the control.
	RowCount (see page 8207)	Specifies the number of rows of panels displayed in the grid.
	SelectedColor (see page 8208)	Specifies the background color of the panel that represents the current record in the dataset.
	ShowFocus (see page 8208)	Indicates whether to draw a focus rectangle around the current record in the TDBCtrlGrid (see page 8168) when it has input focus.
	ShowHint (see page 8209)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 8210)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 8211)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 8211)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

44.1.1.2 TDBCtrlGrid.Create Constructor

Creates and initializes an instance of TDBCtrlGrid (see page 8168).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDBCtrlGrid(TComponent * AOwner);
```

Description

Call Create to create and initialize a TDBCtrlGrid (see page 8168) object at runtime.TDBCtrlGrid (see page 8168) objects placed in a form at design time are created automatically.

AOwner is a component that becomes the value of the Owner property. This component, typically the form, is responsible for freeing the TDBCtrlGrid (see page 8168) instance.

See Also

Owner (see page 2807)

44.1.1.3 TDBCtrlGrid.Destroy Destructor

Destroys an instance of TDBCtrlGrid (see page 8168).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBCtrlGrid();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which checks that the TDBCtrlGrid (see page 8168) reference is not nil, and only then calls Destroy.

Destroy frees the helper objects used by TDBCtrlGrid (see page 8168) before calling the inherited destructor.

See Also

Free (see page 22300)

Create (see page 8179)

44.1.1.4 TDBCtrlGrid Methods

The methods of the TDBCtrlGrid class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 8180)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 8181)	Creates the underlying window.
	GetChildParent (see page 8183)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 8183)	Calls a specified method for each child of the control.
	PaintPanel (see page 8185)	This is PaintPanel, a member of class TDBCtrlGrid.
	PaintWindow (see page 8185)	Renders the image of a windowed control.
	ReadState (see page 8185)	Prepares the control for having its properties assigned values from a stream.

Public Methods

	Name	Description
	DoKey (see page 8181)	Implements the responses to a number of standard keystrokes.
	ExecuteAction (see page 8182)	Executes an action with the DB (see page 6725) control grid as its target.
	GetTabOrderList (see page 8183)	Overrides the default to remove all child controls from the tab order.
	KeyDown (see page 8184)	Performs special processing when a key is pressed.
	SetBounds (see page 8186)	Sets the windowed control's boundary properties all at once.
	UpdateAction (see page 8186)	Updates an action to reflect the current state of the grid.

44.1.1.4.1 TDBCtrlGrid.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information

needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

44.1.1.4.2 TDBCtrlGrid.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams (see page 6135)

CreateWindowHandle (see page 6137)

Perform (see page 5776)

DestroyWnd (see page 6141)

44.1.1.4.3 TDBCtrlGrid.DoKey Method

Implements the responses to a number of standard keystrokes.

Pascal

```
procedure DoKey(Key: TDBCtrlGridKey);
```

C++

```
__fastcall DoKey(TDBCtrlGridKey Key);
```

Description

Call DoKey to perform the operation specified by the Key parameter. For example, use DoKey to remap keystrokes from an OnKeyDown (see page 8225) event handler.

The actions performed for the different values of Key are given in the following table:

Value	Action
gkNull	Do nothing.
gkEditMode	Toggle the EditMode (see page 8197) property.

gkPriorTab	Move to the previous panel.
gkNextTab	Move to the next panel.
gkLeft	Move one panel to the left.
gkRight	Move one panel to the right.
gkUp	Move one panel up.
gkDown	Move one panel down.
gkScrollUp	Move one panel up.
gkScrollDown	Move one panel down.
gkPageUp	Move ColCount (see page 8193) * RowCount (see page 8207) records up in the dataset.
gkPageDown	Move ColCount (see page 8193) * RowCount (see page 8207) records down in the dataset.
gkHome	Move to the first record in the dataset.
gkEnd	Move to the last record in the dataset.
gkInsert	Insert a new record prior to the current record in the dataset and set EditMode (see page 8197) to true.
gkAppend	Insert a new record at the end of the dataset and set EditMode (see page 8197) to true.
gkDelete	Delete the current record from the dataset and set EditMode (see page 8197) to false.
gkCancel	Cancel any edits not written to the dataset and set EditMode (see page 8197) to false.

See Also

AllowDelete (see page 8191)

AllowInsert (see page 8192)

EditMode (see page 8197)

KeyDown (see page 8184)

OnKeyDown (see page 6225)

44.1.1.4.4 TDBCtrlGrid.ExecuteAction Method

Executes an action with the DB (see page 6725) control grid as its target.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

ExecuteAction is called automatically when the user invokes an action with the DB (see page 6725) control grid as its target. The Action parameter specifies the action that was invoked. ExecuteAction returns a boolean value indicating whether the action was successfully executed.

ExecuteAction first calls the inherited method, which checks whether the action knows how to execute with the grid as a target. If this does not successfully dispatch the action, ExecuteAction attempts to invoke the action with the grid's associated dataset as a target.

See Also

Execute (see page 2712)

DataSource (see page 8195)

UpdateAction (see page 8186)

44.1.1.4.5 TDBCtrlGrid.GetChildParent Method

Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.

Pascal

```
function GetChildParent: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetChildParent();
```

Description

GetChildParent is used internally in the component streaming system. It is not necessary to call it directly.

As implemented in TComponent, GetChildParent returns Self (Delphi) or this (C++). If GetChildParent returns nil (Delphi) or NULL (C++), the parent is assumed to be the root component currently being read (usually a form).

See Also

GetParentComponent (see page 2778)

44.1.1.4.6 TDBCtrlGrid.GetChildren Method

Calls a specified method for each child of the control.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the streaming system that loads and saves components. Applications seldom need to call this routine.

GetChildren executes the callback specified by the Proc parameter for every child control listed by the Controls property that lists the Root parameter as its Owner.

Override GetChildren to limit or augment which child controls are saved with the control. When overriding GetChildren, call the procedure passed as the Proc parameter for every child control that should be saved. The Root parameter indicates the component (usually a form) that owns the control.

See Also

Controls (see page 6197)

Owner (see page 2807)

44.1.1.4.7 TDBCtrlGrid.GetTabOrderList Method

Overrides the default to remove all child controls from the tab order.

Pascal

```
procedure GetTabOrderList(List: TList); override;
```

C++

```
virtual __fastcall GetTabOrderList(TList List);
```

Description

GetTabOrderList returns an empty list. TDBCtrlGrid (see page 8168) implements its own handling of the tab key to move through the panels in database order.

See Also

TabOrder (see page 6211)

44.1.1.4.8 TDBCtrlGrid.KeyDown Method

Performs special processing when a key is pressed.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

After calling the OnKeyDown (see page 8225) event handler, KeyDown translates each combination of the Key and Shift parameters into a logical key code and calls the DoKey (see page 8181) method to perform the appropriate action.

KeyDown performs the following translations:

Key and Shift values	Key code
LeftArrow	gkLeft
RightArrow	gkRight
UpArrow	gkUp
DownArrow	gkDown
PageUp	gkPageUp
PageDown	gkPageDown
Home	gkHome
End	gkEnd
Return	gkEditMode
F2	gkEditMode
Insert	gkAppend
Ctrl+Insert	gkInsert
Ctrl+Delete	gkDelete
Escape	gkCancel
All other keys	gkNull

See Also

DoKey (see page 8181)

OnKeyDown (see page 6225)

44.1.1.4.9 TDBCtrlGrid.PaintPanel Method

Pascal

```
procedure PaintPanel(Index: Integer); virtual;
```

C++

```
virtual __fastcall PaintPanel(int Index);
```

Description

This is PaintPanel, a member of class TDBCtrlGrid.

44.1.1.4.10 TDBCtrlGrid.PaintWindow Method

Renders the image of a windowed control.

Pascal

```
procedure PaintWindow(DC: HDC); override;
```

C++

```
virtual __fastcall PaintWindow(HDC DC);
```

Description

Call PaintWindow to repaint the control. PaintWindow sends a WM_PAINT message to the windowed control's DefaultHandler method, setting the message record's WParam field to the value passed in DC and the other parameter and result fields to zeros.

See Also

DefaultHandler (see page 6139)

PaintTo (see page 6167)

DoubleBuffered (see page 6205)

44.1.1.4.11 TDBCtrlGrid.ReadState Method

Prepares the control for having its properties assigned values from a stream.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

Override ReadState to change the preparations the control makes as it readies itself to have its property values assigned from a stream. For example, a control might destroy temporary internal data structures or objects before new instances are loaded from the stream. Be sure to include a call to the inherited method when overriding ReadState.

As implemented in TWinControl, ReadState disables control alignment until all the child controls have been read from the stream, calls the inherited ReadState method, and re-enables the alignment of all the controls. The method then builds the tab order list of controls and synchronizes its visibility with that of its parent control.

See Also

Parent (see page 5832)

ReadState (see page 5778)

44.1.1.4.12 TDBCtrlGrid.SetBounds Method

Sets the windowed control's boundary properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to change all of the control's boundary properties at once. The same effect can be achieved by setting the Left, Top, Width, and Height properties separately. By setting all four properties at once, SetBounds ensures that the control will not repaint between changes

Specify the values for the Left, Top, Width, and Height properties as the value of the ALeft, ATop, AWidth, and AHeight parameters, respectively.

See Also

Height (see page 5825)

Left (see page 5830)

Top (see page 5844)

Width (see page 5849)

44.1.1.4.13 TDBCtrlGrid.UpdateAction Method

Updates an action to reflect the current state of the grid.

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

UpdateAction is called automatically when the application is idle allow the grid to set action properties to reflect the current state of the grid. The Action parameter is the action to be updated. UpdateAction returns a boolean that indicates whether the action was updated by the grid.

As implemented in TDBCtrlGrid (see page 8168), UpdateAction checks whether the action relies on the state of the associated dataset, and if so, updates the action accordingly.

See Also

UpdateAction (see page 6887)

DataSource (see page 8195)

Update (see page 2714)

44.1.1.5 TDBCtrlGrid Properties

The properties of the TDBCtrlGrid class are listed here.

Protected Properties

	Name	Description
R	Panel (see page 8199)	This is Panel, a member of class TDBCtrlGrid.

Public Properties

	Name	Description
R	Canvas (see page 8193)	Indicates the drawing surface used to draw the TDBCtrlGridObject.
	EditMode (see page 8197)	Determines whether the TDBCtrlGrid (see page 8168) can be used to insert, delete, or edit records.
R	PanelCount (see page 8200)	Specifies the number of records visible in the grid.
	PanelIndex (see page 8201)	Specifies the index of the panel in the grid that represents the current record.

Published Properties

	Name	Description
	Align (see page 8188)	Determines how the control aligns within its container (parent control).
	AllowDelete (see page 8191)	Determines whether the user can delete the current record from the dataset by pressing Ctrl+Delete.
	AllowInsert (see page 8192)	Determines whether the user can insert new records with the Insert key or append new records with the Ctrl+Insert key combination.
	Anchors (see page 8192)	Specifies how the control is anchored to its parent.
	ColCount (see page 8193)	Specifies the number of columns of panels shown in the grid.
	Color (see page 8193)	Specifies the background color of the control.
	Constraints (see page 8194)	Specifies the size constraints for the control.
	DataSource (see page 8195)	Identifies the link to the dataset where the TDBCtrlGrid (see page 8168) object finds its data.
	DragCursor (see page 8195)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 8195)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 8197)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 8198)	Controls the attributes of text written on or in the control.
	Orientation (see page 8199)	Determines the order in which records from the dataset are displayed in the TDBCtrlGrid (see page 8168).
	PanelBorder (see page 8200)	Determines whether each panel is drawn with a raised border around it.
	PanelHeight (see page 8200)	Specifies the height (in pixels) of each panel.
	PanelWidth (see page 8201)	Specifies the width (in pixels) of each panel.
	ParentColor (see page 8201)	Determines where a control looks for its color information.
	ParentFont (see page 8202)	Determines where a control looks for its font information.
	ParentShowHint (see page 8202)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 8203)	Identifies the pop-up menu associated with the control.
	RowCount (see page 8207)	Specifies the number of rows of panels displayed in the grid.
	SelectedColor (see page 8208)	Specifies the background color of the panel that represents the current record in the dataset.
	ShowFocus (see page 8208)	Indicates whether to draw a focus rectangle around the current record in the TDBCtrlGrid (see page 8168) when it has input focus.
	ShowHint (see page 8209)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 8210)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 8211)	Determines if the user can tab to a control.
	Visible (see page 8211)	Determines whether the component appears onscreen.

44.1.1.5.1 TDBCtrlGrid.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
```

```
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
```

```
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
```

```
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

44.1.1.5.2 TDBCtrlGrid.AllowDelete Property

Determines whether the user can delete the current record from the dataset by pressing Ctrl+Delete.

Pascal

```
property AllowDelete: Boolean;
```

C++

```
__property Boolean AllowDelete;
```

Description

Set AllowDelete to false to prevent the TDBCtrlGrid (see page 8168) object from deleting records.

When AllowDelete is false, TDBCtrlGrid (see page 8168) will not delete records from the dataset when the user presses Ctrl+Delete or the DoKey (see page 8181) method is called with the Key parameter set to gkDelete. The TDBCtrlGrid (see page 8168) object can still be used to insert records.

When AllowDelete is true (the default), TDBCtrlGrid (see page 8168) will delete the current record when the user presses Ctrl+Delete or when the DoKey (see page 8181) method is called with the Key parameter set to gkDelete. Even with AllowDelete set to true, records cannot be deleted unless the dataset is in edit mode.

See Also

DoKey (see page 8181)

44.1.1.5.3 TDBCtrlGrid.AllowInsert Property

Determines whether the user can insert new records with the Insert key or append new records with the Ctrl+Insert key combination.

Pascal

```
property AllowInsert: Boolean;
```

C++

```
__property Boolean AllowInsert;
```

Description

Set AllowInsert to false to prevent the TDBCtrlGrid (see page 8168) object from inserting records.

If AllowInsert is false, TDBCtrlGrid (see page 8168) does not insert records into the dataset when the user presses Insert or Ctrl+Insert or when the DoKey (see page 8181) method is called with the Key parameter set to gkInsert or gkAppend. The TDBCtrlGrid (see page 8168) object can still be used to delete records.

If AllowInsert is true (the default), TDBCtrlGrid (see page 8168) inserts a new record into the dataset before the current record when the user presses Insert or appends a new record to the end of the dataset when the user presses Ctrl+Insert. Even with AllowInsert set to true, records cannot be inserted unless the dataset is in edit mode.

See Also

DoKey (see page 8181)

44.1.1.5.4 TDBCtrlGrid.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

44.1.1.5.5 TDBCtrlGrid.Canvas Property

Indicates the drawing surface used to draw the TDBCtrlGridObject.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use the properties and methods of Canvas to draw an image of a panel in the TDBCtrlGrid (see page 8168) object from an OnPaintPanel (see page 8239) event handler.

See Also

OnPaintPanel (see page 8239)

44.1.1.5.6 TDBCtrlGrid.ColCount Property

Specifies the number of columns of panels shown in the grid.

Pascal

```
property ColCount: Integer;
```

C++

```
__property int ColCount;
```

Description

Set ColCount to the number of columns of panels that should appear in the TDBCtrlGrid (see page 8168) object. Each panel represents a separate record in the dataset.

Setting ColCount causes the TDBCtrlGrid (see page 8168) object to resize to accommodate the new number of columns. PanelWidth (see page 8201) doesn't change.

See Also

PanelWidth (see page 8201)

RowCount (see page 8207)

44.1.1.5.7 TDBCtrlGrid.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically

set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

44.1.1.5.8 TDBCtrlGrid.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

44.1.1.5.9 TDBCtrlGrid.DataSource Property

Identifies the link to the dataset where the TDBCtrlGrid (see page 8168) object finds its data.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Set DataSource to the TDataSource object that links to the dataset where the grid should fetch its data. DataSource allows the data-aware grid to read from, write to, and navigate around the dataset.

```
DBCtrlGrid1.DataSource := DataSource1;
DBCtrlGrid1->DataSource = DataSource1;
```

See Also

Creating a Grid That Contains Other Data-aware Controls

44.1.1.5.10 TDBCtrlGrid.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

44.1.1.5.11 TDBCtrlGrid.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
```

```
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

44.1.1.5.12 TDBCtrlGrid.EditMode Property

Determines whether the TDBCtrlGrid (see page 8168) can be used to insert, delete, or edit records.

Pascal

```
property EditMode: Boolean;
```

C++

```
__property Boolean EditMode;
```

Description

Set EditMode to true to allow the user to edit the dataset using the grid. The user can toggle EditMode by pressing F2 or Enter. Inserting or deleting records using the grid will set EditMode to true.

See Also

AllowDelete (see page 8191)

AllowInsert (see page 8192)

DataSource (see page 8195)

DoKey (see page 8181)

44.1.1.5.13 TDBCtrlGrid.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

44.1.1.5.14 TDBCtrlGrid.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
```

```
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

44.1.1.5.15 TDBCtrlGrid.Orientation Property

Determines the order in which records from the dataset are displayed in the TDBCtrlGrid (see page 8168).

Pascal

```
property Orientation: TDBCtrlGridOrientation;
```

C++

```
__property TDBCtrlGridOrientation Orientation;
```

Description

Set Orientation to goVertical (the default) to have the panels ordered in rows with a vertical scroll bar. Set Orientation to goHorizontal to arrange panels in columns with a horizontal scroll bar.

See Also

ColCount (see page 8193)

RowCount (see page 8207)

44.1.1.5.16 TDBCtrlGrid.Panel Property

Pascal

```
property Panel: TDBCtrlPanel;
```

C++

```
__property TDBCtrlPanel Panel;
```

Description

This is Panel, a member of class TDBCtrlGrid.

44.1.1.5.17 TDBCtrlGrid.PanelBorder Property

Determines whether each panel is drawn with a raised border around it.

Pascal

```
property PanelBorder: TDBCtrlGridBorder;
```

C++

```
__property TDBCtrlGridBorder PanelBorder;
```

Description

The default value of gbRaised draws a raised border. Set PanelBorder to gbNone to eliminate the border.

See Also

OnPaintPanel (see page 8239)

44.1.1.5.18 TDBCtrlGrid.PanelCount Property

Specifies the number of records visible in the grid.

Pascal

```
property PanelCount: Integer;
```

C++

```
__property int PanelCount;
```

Description

TDBCtrlGrid (see page 8168) can represent a number of panels equal to the number in ColCount (see page 8193) times the number in RowCount (see page 8207). Usually, PanelCount will be the same as this value. PanelCount will be smaller than this if there are not enough records in the dataset to fill the entire grid.

See Also

ColCount (see page 8193)

RowCount (see page 8207)

44.1.1.5.19 TDBCtrlGrid.PanelHeight Property

Specifies the height (in pixels) of each panel.

Pascal

```
property PanelHeight: Integer;
```

C++

```
__property int PanelHeight;
```

Description

Set PanelHeight to the desired size of each panel in the grid. When the PanelHeight property is set, the grid resizes itself to fit the new panel height.

See Also

PanelWidth (see page 8201)

44.1.1.5.20 TDBCtrlGrid.PanelIndex Property

Specifies the index of the panel in the grid that represents the current record.

Pascal

```
property PanelIndex: Integer;
```

C++

```
__property int PanelIndex;
```

Description

Read PanelIndex to determine which panel represents the current record in the dataset. Set PanelIndex to move in the dataset to the record corresponding to a panel in the grid.

PanelIndex is a number in the range 0 to PanelCount (see page 8200) - 1, where 0 is the first panel, 1 is the second panel, and so on. The location of the panel with the index PanelIndex depends on the value of the Orientation (see page 8199) property. When Orientation (see page 8199) is goVertical, the panels are counted by rows. When Orientation (see page 8199) is goHorizontal, the panels are counted by columns.

See Also

Orientation (see page 8199)

PanelCount (see page 8200)

44.1.1.5.21 TDBCtrlGrid.PanelWidth Property

Specifies the width (in pixels) of each panel.

Pascal

```
property PanelWidth: Integer;
```

C++

```
__property int PanelWidth;
```

Description

Set PanelWidth to the desired size of each panel in the grid. When the PanelWidth property is set, the grid resizes itself to fit the new panel width.

See Also

PanelHeight (see page 8200)

44.1.1.5.22 TDBCtrlGrid.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

44.1.1.5.23 TDBCtrlGrid.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

44.1.1.5.24 TDBCtrlGrid.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

44.1.1.5.25 TDBCtrlGrid.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
```

```
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Cut1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      const int num_items = 4;
      for (int index = 0; index < num_items; ++index)
      {
          TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
          PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
          NewItem->Caption = "Menu Item " + IntToStr(index);
          NewItem->Tag = index;
          NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
          TNotifyEvent();
      }
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
          {
```

```
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
```

```
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
```

```
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

44.1.1.5.26 TDBCtrlGrid.RowCount Property

Specifies the number of rows of panels displayed in the grid.

Pascal

```
property RowCount: Integer;
```

C++

```
__property int RowCount;
```

Description

Set RowCount to the number of rows of panels that should appear in the TDBCtrlGrid (see page 8168) object. Each panel represents a separate record in the dataset.

Setting RowCount causes the TDBCtrlGrid (see page 8168) object to resize to accommodate the new number of rows. PanelHeight (see page 8200) does not change.

See Also

ColCount (see page 8193)

PanelHeight (see page 8200)

44.1.1.5.27 TDBCtrlGrid.SelectedColor Property

Specifies the background color of the panel that represents the current record in the dataset.

Pascal

```
property SelectedColor: TColor;
```

C++

```
__property TColor SelectedColor;
```

Description

Set SelectedColor to a value other than value of the Color (see page 8193) property to make the panel that represents the current record stand out. TDBCtrlGrid (see page 8168) draws the selected panel using SelectedColor even when it does not have input focus. All other panels are drawn with the background set to the value of the Color (see page 8193) property.

See Also

Color (see page 5812)

44.1.1.5.28 TDBCtrlGrid.ShowFocus Property

Indicates whether to draw a focus rectangle around the current record in the TDBCtrlGrid (see page 8168) when it has input focus.

Pascal

```
property ShowFocus: Boolean;
```

C++

```
__property Boolean ShowFocus;
```

Description

Set ShowFocus to true to have TDBCtrlGrid (see page 8168) draw a focus rectangle around the current record when it has input focus. Set ShowFocus to false when providing visual feedback of focus in some other manner in an OnPaintPanel (see page 8239) event handler.

See Also

OnPaintPanel (see page 8239)

PanelIndex (see page 8201)

SelectedColor (see page 8208)

44.1.1.5.29 TDBCtrlGrid.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

44.1.1.5.30 TDBCtrlGrid.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

44.1.1.5.31 TDBCtrlGrid.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

44.1.1.5.32 TDBCtrlGrid.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
```

```
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

44.1.1.6 TDBCtrlGrid Events

The events of the TDBCtrlGrid class are listed here.

Published Events

	Name	Description
	OnClick (see page 8213)	Occurs when the user clicks the control.
	OnDblClick (see page 8215)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 8215)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 8218)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 8220)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 8221)	Occurs when a control receives the input focus.
	OnExit (see page 8224)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 8225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 8228)	Occurs when a key is pressed.
	OnKeyUp (see page 8229)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 8231)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 8232)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 8234)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 8234)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 8234)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 8237)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaintPanel (see page 8239)	Occurs when TDBCtrlGrid (see page 8168) needs to draw a panel in the grid.
	OnStartDrag (see page 8240)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

44.1.1.6.1 TDBCtrlGrid.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).

- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
```

```
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

44.1.1.6.2 TDBCtrlGrid.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

44.1.1.6.3 TDBCtrlGrid.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

44.1.1.6.4 TDBCtrlGrid.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.
```

```
procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

44.1.1.6.5 TDBCtrlGrid.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
```

```
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

44.1.1.6.6 TDBCtrlGrid.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
```

```
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

44.1.1.6.7 TDBCtrlGrid.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

44.1.1.6.8 TDBCtrlGrid.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the

OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
```

```
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

44.1.1.6.9 TDBCtrlGrid.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
```

```
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

44.1.1.6.10 TDBCtrlGrid.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
```

```
      else
        ComboBox1->Text = "KeyDown";
      Form1->Refresh();
      Memo1->Lines->Add(ComboBox1->Text);
    }
  }

  void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
  {
    if (Key == VK_F1)
    {
      Form1->Color = FormColor;
      if (ComboBox1->Text == "KeyUp")
        ComboBox1->Text = "KeyUp2";
      else
        ComboBox1->Text = "KeyUp";
      Form1->Refresh();
      Memo1->Lines->Add(ComboBox1->Text);
    }
  }
```

Delphi Examples:

```
  {
  This example changes a form's color to aqua when the user
  presses the F1 key, even when a control on the form has the
  focus. When the user releases the key, the form returns to
  its original color.  Set the KeyPreview on the form to True.
  Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
  event to FormKeyUp.  Set the KeyPreview on the form to True.
  }
  var
    FormColor: TColor;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    KeyPreview := True;
    FormColor := Form1.Color;
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

44.1.1.6.11 TDBCtrlGrid.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

44.1.1.6.12 TDBCtrlGrid.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
```

```
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
```

```
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

44.1.1.6.13 TDBCtrlGrid.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

44.1.1.6.14 TDBCtrlGrid.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

44.1.1.6.15 TDBCtrlGrid.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
```

```
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

44.1.1.6.16 TDBCtrlGrid.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
```

```
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
```

```
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

44.1.1.6.17 TDBCtrlGrid.OnPaintPanel Event

Occurs when TDBCtrlGrid (see page 8168) needs to draw a panel in the grid.

Pascal

```
property OnPaintPanel: TPaintPanelEvent;
```

C++

```
__property TPaintPanelEvent OnPaintPanel;
```

Description

Write an OnPaintPanel event handler to customize the drawing of a panel in the TDBCtrlGrid (see page 8168) object. TDBCtrlGrid (see page 8168) paints the background using the Color (see page 8193) or SelectedColor (see page 8208) property, including the raised border if it is specified by the PanelBorder (see page 8200) property. Then the OnPaintPanel event occurs to allow an event handler to add to the image of the panel.

The DBCtrlGrid parameter is the grid that contains the panel. The Index parameter is the index of the panel being painted, where 0 is the first panel, 1 is the second panel, and so on. Use the PanelIndex (see page 8201) property to determine whether the panel being drawn is selected.

Use the Canvas (see page 8193) property to paint the panel. The point (0,0) on the canvas is the upper left corner of the panel, and the point (PanelWidth (see page 8201), PanelHeight (see page 8200)) is the lower right corner.

Controls (see page 5711) placed on the panel are drawn separately and do not need to be drawn in an OnPaintPanel event handler.

See Also

Canvas (see page 8193)

Color (see page 5812)

PanelBorder (see page 8200)

PanelHeight (see page 8200)

PanelIndex (see page 8201)

PanelWidth (see page 8201)

SelectedColor (see page 8208)

44.1.1.6.18 TDBCtrlGrid.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

44.1.2 TDBCtrlGridLink Class

TDBCtrlGridLink helps manage data events for a TDBCtrlGrid (see page 8168) object.

Class Hierarchy

Pascal

```
TDBCtrlGridLink = class(TDataLink);
```

C++

```
class TDBCtrlGridLink : public TDataLink;
```

File

dbcgrids

Description

TDBCtrlGridLink is tailored to work with a TDBCtrlGrid (see page 8168). It should only be used by the TDBCtrlGrid (see page 8168) class.

See Also

TDataSource (see page 7070)

TDBCtrlGrid (see page 8168)

44.1.2.1 TDBCtrlGridLink Members

The following tables list the members exposed by TDBCtrlGridLink.

Protected Methods

	Name	Description
	ActiveChanged (see page 8243)	Responds to changes in the Active property.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CheckBrowseMode (see page 6880)	Allows the data-aware object to respond just before an event that changes the dataset state.
	DataEvent (see page 6881)	Responds to various events that occur while working with data.
	DataSetChanged (see page 8243)	Responds to changes in the dataset.
	DataSetScrolled (see page 6882)	Allows a response to scrolling the representation of the dataset within the data-aware object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EditingChanged (see page 6883)	Responds to changes in the editing state of the data source.
	FocusControl (see page 6884)	Provides the interface for a method to force focus to the data-aware component associated with the TDataLink (see page 6876).
	GetActiveRecord (see page 6884)	This is GetActiveRecord, a member of class TDataLink.
	GetBOF (see page 6884)	This is GetBOF, a member of class TDataLink.
	GetBufferCount (see page 6884)	This is GetBufferCount, a member of class TDataLink.
	GetEOF (see page 6885)	This is GetEOF, a member of class TDataLink.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRecordCount (see page 6885)	This is GetRecordCount, a member of class TDataLink.
	LayoutChanged (see page 6885)	Responds to changes in the representation of the data by the data-aware object.
	MoveBy (see page 6885)	Repositions the dataset to another record relative to its active record.
	RecordChanged (see page 6886)	Responds to changes in the contents of the current record or field of the dataset.
	SetActiveRecord (see page 6886)	This is SetActiveRecord, a member of class TDataLink.
	SetBufferCount (see page 6887)	This is SetBufferCount, a member of class TDataLink.
	UpdateData (see page 6887)	Provides an interface for writing edits to a record in the dataset.

Protected Properties

	Name	Description
	VisualControl (see page 6894)	Indicates whether a visual control uses the datalink.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 8243)	Creates an instance of TDBCtrlGridLink (see page 8240).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 6879)	Destroys an instance of TDataLink (see page 6876).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6882)	Attempts to drive the data source into an editing state if it is not already in one.
	ExecuteAction (see page 6883)	Invokes an action with the datasource as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	UpdateAction (see page 6887)	Updates an action to reflect the current state of the TDataLink (see page 6876).
	UpdateRecord (see page 6888)	Responds to notifications that edits should be posted to the dataset.

Public Properties

	Name	Description
R	Active (see page 6888)	Specifies whether the dataset that TDataLink (see page 6876) manages is active.
	ActiveRecord (see page 6889)	Specifies the index of the current record within the internal record buffer maintained by the dataset for the Owner of the TDataLink (see page 6876) object.
R	BOF (see page 6889)	Indicates whether the first record in a dataset is active.
	BufferCount (see page 6890)	Specifies the number of records cached by the dataset for the Owner of the TDataLink (see page 6876) object.
R	DataSet (see page 6890)	Specifies the TDataSet (see page 6894) object that represents the data that this TDataLink (see page 6876) is tracking.
	DataSource (see page 6891)	Specifies the TDataSource (see page 7070) object that the Owner of the TDataLink (see page 6876) uses to connect to the dataset.
	DataSourceFixed (see page 6891)	Indicates whether the DataSource (see page 6891) property can be set.
R	Editing (see page 6892)	Indicates whether the DataSource (see page 6891) is in edit mode.
R	Eof (see page 6892)	Indicates whether the last record in a dataset is active.
	ReadOnly (see page 6893)	Indicates whether the dataset for this TDataLink (see page 6876) can be modified.
R	RecordCount (see page 6893)	Indicates the number of records in the internal record buffer maintained by the dataset.

44.1.2.2 TDBCtrlGridLink.Create Constructor

Creates an instance of TDBCtrlGridLink (see page 8240).

Pascal

```
constructor Create(DBCtrlGrid: TDBCtrlGrid);
```

C++

```
__fastcall TDBCtrlGridLink(TDBCtrlGrid DBCtrlGrid);
```

Description

Applications should not call the Create method for a TDBCtrlGridLink (see page 8240) directly. The TDBCtrlGrid (see page 8168) that owns a TDBCtrlGridLink (see page 8240) calls Create from its own constructor.

See Also

Create (see page 6878)

44.1.2.3 TDBCtrlGridLink Methods

The methods of the TDBCtrlGridLink class are listed here.

Protected Methods

	Name	Description
	ActiveChanged (see page 8243)	Responds to changes in the Active property.
	DataSetChanged (see page 8243)	Responds to changes in the dataset.

44.1.2.3.1 TDBCtrlGridLink.ActiveChanged Method

Responds to changes in the Active property.

Pascal

```
procedure ActiveChanged; override;
```

C++

```
virtual __fastcall ActiveChanged();
```

Description

The ActiveChanged method defined by TDataLink merely provides an interface for a method that can respond to changes in the Active property. Derived objects that do not need to respond to such changes can allow the inherited method to ignore them.

See Also

Active (see page 6888)

44.1.2.3.2 TDBCtrlGridLink.DataSetChanged Method

Responds to changes in the dataset.

Pascal

```
procedure DataSetChanged; override;
```

C++

```
virtual __fastcall DataSetChanged();
```

Description

DataSetChanged responds to changes to the contents of the dataset. Anything that changes the contents of the dataset, whether it is editing the data, inserting or deleting records, or changing the key triggers this method. Changes specific to the representation of the data within the data-aware object, such as scrolling the dataset or changing the layout of data elements within the object, also trigger this method. DataSetChanged simply calls RecordChanged. Derived classes can override this procedure to make additional adjustments to changes in the dataset.

See Also

DataSetScrolled (see page 6882)

LayoutChanged (see page 6885)

RecordChanged (see page 6886)

44.1.3 TDBCtrlPanel Class

TDBCtrlPanel represents a panel in a TDBCtrlGrid (see page 8168) object.

Class Hierarchy

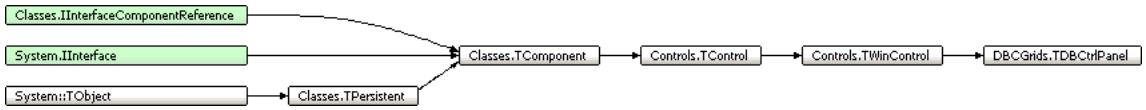

Pascal

```
TDBCtrlPanel = class(TWinControl);
```

C++

```
class TDBCtrlPanel : public TWinControl;
```

File

dbcgrids

Description

Use TDBCtrlPanel to hold the fields from a single record in a dataset. TDBCtrlGrid (see page 8168) represents the records in a dataset by arranging copies of the same TDBCtrlPanel object in rows or columns, where each panel represents a single record in the dataset.

Lay out a single panel at design time. Add data-aware edit controls or data display objects. At runtime, TDBCtrlGrid (see page 8168) replicates the panel for the records in the dataset.

The appearance of the panel is determined by the properties of the TDBCtrlGrid (see page 8168) object that contains the panel. The OnPaintPanel event of TDBCtrlGrid (see page 8168) allows customized painting of the panels.

See Also

TDBCtrlGrid (see page 8168)

TDBEdit (see page 8735)

TDBImage (see page 8813)

TDBMemo (see page 9149)

TDBRichEdit (see page 9343)

TDBText (see page 9433)

44.1.3.1 TDBCtrlPanel Members

The following tables list the members exposed by TDBCtrlPanel.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 8255)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.

	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.

	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 8256)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.

	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateLinked (see page 8255)	Creates and initializes an instance of TDBCtrlPanel (see page 8244).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.

	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.

	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.

R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.

	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

44.1.3.2 TDBCtrlPanel.CreateLinked Constructor

Creates and initializes an instance of TDBCtrlPanel (see page 8244).

Pascal

```
constructor CreateLinked(DBCtrlGrid: TDBCtrlGrid);
```

C++

```
__fastcall TDBCtrlPanel(TDBCtrlGrid DBCtrlGrid);
```

Description

Do not call the constructor for TDBCtrlPanel (see page 8244) directly. TDBCtrlGrid (see page 8168) calls the CreateLinked method for TDBCtrlPanel (see page 8244) from its own constructor.

CreateLinked initializes the Parent property to the DBCtrlGrid parameter and initializes the ControlStyle property to include [csAcceptsControls, csCaptureMouse, csClickEvents, csDoubleClicks, csOpaque, csReplicatable].

44.1.3.3 TDBCtrlPanel Methods

The methods of the TDBCtrlPanel class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 8255)	Initializes a window-creation parameter data structure.
	PaintWindow (see page 8256)	Renders the image of a windowed control.

44.1.3.3.1 TDBCtrlPanel.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

44.1.3.3.2 TDBCtrlPanel.PaintWindow Method

Renders the image of a windowed control.

Pascal

```
procedure PaintWindow(DC: HDC); override;
```

C++

```
virtual __fastcall PaintWindow(HDC DC);
```

Description

Call PaintWindow to repaint the control. PaintWindow sends a WM_PAINT message to the windowed control's DefaultHandler method, setting the message record's WParam field to the value passed in DC and the other parameter and result fields to zeros.

See Also

DefaultHandler (see page 6139)

PaintTo (see page 6167)

DoubleBuffered (see page 6205)

44.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TDBCtrlGridBorder (see page 8257)	This is record DBCGrids.TDBCtrlGridBorder.
	TDBCtrlGridKey (see page 8257)	This is record DBCGrids.TDBCtrlGridKey.
	TDBCtrlGridOrientation (see page 8258)	This is record DBCGrids.TDBCtrlGridOrientation.

44.2.1 DBCGrids.TDBCtrlGridBorder Enumeration

Pascal

```
TDBCtrlGridBorder = (
  gbNone,
  gbRaised
);
```

C++

```
enum TDBCtrlGridBorder {
  gbNone,
  gbRaised
};
```

File

dbcgrids

Description

This is record DBCGrids.TDBCtrlGridBorder.

44.2.2 DBCGrids.TDBCtrlGridKey Enumeration

Pascal

```
TDBCtrlGridKey = (
  gkNull,
  gkEditMode,
  gkPriorTab,
  gkNextTab,
  gkLeft,
  gkRight,
  gkUp,
  gkDown,
  gkScrollUp,
  gkScrollDown,
  gkPageUp,
  gkPageDown,
  gkHome,
  gkEnd,
```

```
    gkInsert,
    gkAppend,
    gkDelete,
    gkCancel
  );
```

C++

```
  enum TDBCtrlGridKey {
    gkNull,
    gkEditMode,
    gkPriorTab,
    gkNextTab,
    gkLeft,
    gkRight,
    gkUp,
    gkDown,
    gkScrollUp,
    gkScrollDown,
    gkPageUp,
    gkPageDown,
    gkHome,
    gkEnd,
    gkInsert,
    gkAppend,
    gkDelete,
    gkCancel
  };
```

File

dbcgrids

Description

This is record DBCGrids.TDBCtrlGridKey.

44.2.3 DBCGrids.TDBCtrlGridOrientation Enumeration

Pascal

```
  TDBCtrlGridOrientation = (
    goVertical,
    goHorizontal
  );
```

C++

```
  enum TDBCtrlGridOrientation {
    goVertical,
    goHorizontal
  };
```

File

dbcgrids

Description

This is record DBCGrids.TDBCtrlGridOrientation.

44.3 Types

The following table lists types in this documentation.

Types

Name	Description
TPaintPanelEvent (see page 8259)	This is type DBCGrids.TPaintPanelEvent.

44.3.1 DBCGrids.TPaintPanelEvent Type

Pascal

```
TPaintPanelEvent = procedure (DBCtrlGrid: TDBCtrlGrid; Index: Integer) of object;
```

C++

```
(DBCtrlGrid: TDBCtrlGrid; Index: Integer) ( TPaintPanelEvent)();
```

File

dbcgrids

Description

This is type DBCGrids.TPaintPanelEvent.

45 DBClient Namespace

This is namespace DBClient.

Topics

Name	Description
Classes (see page 8263)	The following table lists classes in this documentation.
Functions (see page 8534)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 8536)	The following table lists structs, records, enums in this documentation.
Types (see page 8541)	The following table lists types in this documentation.
Constants (see page 8545)	The following table lists constants in this documentation.

Classes

	Name	Description
	EDBClient (see page 8263)	EDBClient is the exception class for client dataset errors.
	EReconcileError (see page 8265)	EReconcileError is the exception class for client dataset update errors.
	TAggregate (see page 8268)	TAggregate represents a maintained aggregate in a client dataset.
	TAggregates (see page 8280)	TAggregates is a collection of TAggregate (see page 8268) objects.
	TClientBlobStream (see page 8285)	TClientBlobStream lets applications read from or write to BLOB fields in a client dataset.
	TClientDataSet (see page 8289)	TClientDataSet implements a database-independent dataset.
	TConnectionBroker (see page 8360)	TConnectionBroker centralizes the connection to an application server for a set of client datasets that all use the same connection.
	TCustomClientDataSet (see page 8373)	TCustomClientDataSet is the base class for datasets that cache their data in memory.
	TCustomRemoteServer (see page 8525)	TCustomRemoteServer is the base class for all components that connect to an application server in a multi-tiered database application.

Constants

Name	Description
AllParamTypes (see page 8545)	This is constant DBClient.AllParamTypes.
AllRecords (see page 8545)	This is constant DBClient.AllRecords.

Functions

	Name	Description
	PackageParams (see page 8534)	Converts a TParams object into an OleVariant that can be used in multi-tiered applications.
	UnpackParams (see page 8534)	Converts an OleVariant that represents a set of parameters into a TParams object.

Structs, Records, Enums

	Name	Description
	TDataPacketFormat (see page 8536)	TDataPacketFormat indicates how information is encoded in a data packet or delta packet.
	TDataSetOption (see page 8537)	This is record DBClient.TDataSetOption.
	TFetchOption (see page 8537)	TFetchOption and TFetchOptions (see page 8543) indicate the type of information about a record that a client dataset can request.
	TKeyBuffer (see page 8538)	This is record DBClient.TKeyBuffer.
	TRecInfo (see page 8538)	This is record DBClient.TRecInfo.
	TReconcileAction (see page 8539)	TReconcileAction indicates how a client dataset should handle a record whose update generated an error.
	TReconcileInfo (see page 8539)	This is record DBClient.TReconcileInfo.

Types

Name	Description
PKeyBuffer (see page 8541)	This is type DBClient.PKeyBuffer.
PRecInfo (see page 8541)	This is type DBClient.PRecInfo.
TAggUpdateEvent (see page 8542)	This is type DBClient.TAggUpdateEvent.
TCDSKeyBuffer (see page 8542)	This is type DBClient.TCDSKeyBuffer.
TCDSRecInfo (see page 8542)	This is type DBClient.TCDSRecInfo.
TDataSetOptions (see page 8543)	This is type DBClient.TDataSetOptions.
TFetchOptions (see page 8543)	This is type DBClient.TFetchOptions.
TFieldDescList (see page 8543)	This is type DBClient.TFieldDescList.
TGetUsernameEvent (see page 8544)	This is type DBClient.TGetUsernameEvent.
TReconcileErrorEvent (see page 8544)	This is type DBClient.TReconcileErrorEvent.
TRemoteEvent (see page 8544)	This is type DBClient.TRemoteEvent.

45.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EDBClient (see page 8263)	EDBClient is the exception class for client dataset errors.
	EReconcileError (see page 8265)	EReconcileError is the exception class for client dataset update errors.
	TAggregate (see page 8268)	TAggregate represents a maintained aggregate in a client dataset.
	TAggregates (see page 8280)	TAggregates is a collection of TAggregate (see page 8268) objects.
	TClientBlobStream (see page 8285)	TClientBlobStream lets applications read from or write to BLOB fields in a client dataset.
	TClientDataSet (see page 8289)	TClientDataSet implements a database-independent dataset.
	TConnectionBroker (see page 8360)	TConnectionBroker centralizes the connection to an application server for a set of client datasets that all use the same connection.
	TCustomClientDataSet (see page 8373)	TCustomClientDataSet is the base class for datasets that cache their data in memory.
	TCustomRemoteServer (see page 8525)	TCustomRemoteServer is the base class for all components that connect to an application server in a multi-tiered database application.

45.1.1 EDBClient Class

EDBClient is the exception class for client dataset errors.

Class Hierarchy

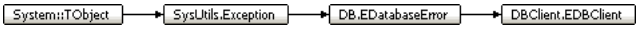

Pascal

```
EDBClient = class(EDatabaseError);
```

C++

```
class EDBClient : public EDatabaseError;
```

File

DBClient

Description

EDBClient is raised if a client dataset encounters a problem in its normal functioning.

See Also

TClientDataSet (see page 8289)

45.1.1.1 EDBClient Members

The following tables list the members exposed by EDBClient.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 8264)	Creates and initializes an instance of EDBClient (see page 8263).
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ErrorCode (see page 8265)	The error code returned by the BDE.
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.

45.1.1.2 EDBClient.Create Constructor

Creates and initializes an instance of EDBClient (see page 8263).

Pascal

```
constructor Create(Message: string; ErrorCode: DBResult);
```

C++

```
__fastcall EDBClient(AnsiString Message, DBResult ErrorCode);
```

Description

Create generates an EDBClient (see page 8263) object. Message is the error message for the exception. ErrorCode (see page 8265) is the error code returned from the data access mechanism that led to this exception.

45.1.1.3 EDBClient Properties

The properties of the EDBClient class are listed here.

Public Properties

	Name	Description
R	ErrorCode (see page 8265)	The error code returned by the BDE.

45.1.1.3.1 EDBClient.ErrorCode Property

The error code returned by the BDE.

Pascal

```
property ErrorCode: DBResult;
```

C++

```
__property DBResult ErrorCode;
```

Description

ErrorCode is the error code returned by the BDE, ADO, dbExpress, or other underlying database access mechanism.

45.1.2 EReconcileError Class

EReconcileError is the exception class for client dataset update errors.

Class Hierarchy

System::TObject → SysUtils.Exception → DB.EDatabaseError → DBClient.EDBClient → DBClient.EReconcileError

Pascal

```
EReconcileError = class(EDBClient);
```

C++

```
class EReconcileError : public EDBClient;
```

File

DBClient

Description

EReconcileError is raised if an error occurs when updates are being applied from a TClientDataSet (see page 8289).

See Also

TCustomClientDataSet (see page 8373)

45.1.2.1 EReconcileError Members

The following tables list the members exposed by EReconcileError.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 8267)	Creates and initializes an instance of EReconcileError (see page 8265).
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Context (see page 8267)	Contains context information from the error stack.
R	ErrorCode (see page 8265)	The error code returned by the BDE.

	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.
R	PreviousError (see page 8268)	Indicates the previous error in the reconcile process.

45.1.2.2 EReconcileError.Create Constructor

Creates and initializes an instance of EReconcileError (see page 8265).

Pascal

```
constructor Create(NativeError: string; Context: string; ErrorCode: DBResult; PreviousError:
DBResult);
```

C++

```
__fastcall EReconcileError(AnsiString NativeError, AnsiString Context, DBResult ErrorCode,
DBResult PreviousError);
```

Description

Create generates an EReconcileError (see page 8265) object.

NativeError is the runtime error message to display in the exception dialog box.

Context (see page 8267) provides context information about the error that occurred. If no context information is available, Context (see page 8267) is nil (Delphi) or NULL (C++). Context (see page 8267) initializes the Context (see page 8267) property.

ErrorCode (see page 8265) is the error code returned by the underlying database access mechanism (ADO, BDE, or dbExpress). Not all exceptions are associated with an error code. If there is no error code, ErrorCode (see page 8265) is 0.

PreviousError (see page 8268) is the error code of the last error in this reconciliation process. If the last exception did not have an error code, PreviousError (see page 8268) is 0.

45.1.2.3 EReconcileError Properties

The properties of the EReconcileError class are listed here.

Public Properties

	Name	Description
R	Context (see page 8267)	Contains context information from the error stack.
R	PreviousError (see page 8268)	Indicates the previous error in the reconcile process.

45.1.2.3.1 EReconcileError.Context Property

Contains context information from the error stack.

Pascal

```
property Context: string;
```

C++

```
__property AnsiString Context;
```

Description

Context contains any additional information returned with the error message. If no additional context information is available, Context is nil (Delphi) or NULL (C++).

See Also

ErrorCode (see page 8265)

45.1.2.3.2 EReconcileError.PreviousError Property

Indicates the previous error in the reconcile process.

Pascal

```
property PreviousError: DBResult;
```

C++

```
__property DBResult PreviousError;
```

Description

PreviousError is the error code of the last update error in this update operation. If the last update error did not have an error code, PreviousError is 0.

See Also

ErrorCode (see page 8265)

45.1.3 TAggregate Class

TAggregate represents a maintained aggregate in a client dataset.

Class Hierarchy

Pascal

```
TAggregate = class(TCollectionItem);
```

C++

```
class TAggregate : public TCollectionItem;
```

File

DBClient

Description

Use TAggregate to specify the formula for an aggregate that can be used by a client dataset. Each aggregate object describes a calculation that summarizes the data in a set of records. The group of records that is summarized is part of the aggregate object. Thus, if a client dataset is performing the same summary calculation over subgroups of records and over the entire dataset, it must use two separate TAggregate objects.

When aggregates summarize data over a group of records (rather than the entire client dataset), they must be associated with an index. The fields in the index define the possible groups of records. When a client dataset changes its index, any aggregate not associated with the new index becomes unavailable, and any active aggregate associated with the new index becomes available.

Aggregate objects can be created and used directly in code at runtime. At design time, you can use the Fields editor to add aggregate fields to a client dataset. When you define aggregate fields at design time, the Fields editor automatically creates the TAggregate objects for them.

See Also

TAggregates (see page 8280)

TAggregateField (see page 6743)

TClientDataSet (see page 8289)

TCustomClientDataSet (see page 8373)

TDataSet (see page 6894)

Specifying Aggregates

45.1.3.1 TAggregate Members

The following tables list the members exposed by TAggregate.

Protected Methods

	Name	Description
	Activate (see page 8271)	This is Activate, a member of class TAggregate.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
V	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
V	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
V	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Protected Properties

	Name	Description
R	DependentFields (see page 8276)	This is DependentFields, a member of class TAggregate.
	RecBufOfs (see page 8278)	This is RecBufOfs, a member of class TAggregate.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 8272)	Copies the properties of another TAggregate (see page 8268) object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 8270)	Creates and initializes an instance of TAggregate (see page 8268).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 8271)	Destroys an instance of TAggregate (see page 8268).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
V	GetDisplayName (see page 8272)	Specifies the name of the aggregate object as it appears in the Object Inspector.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Value (see page 8273)	Returns the calculated value of the aggregate.

Public Properties

	Name	Description
	AggHandle (see page 8274)	Provides access to the aggregate handle.
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
R	DataSet (see page 8275)	Indicates the client dataset whose records are summarized by this aggregate.
R	DataSize (see page 8275)	Indicates the number of bytes needed to store the aggregate's value.
R	DataType (see page 8276)	Indicates the data type of the calculated value.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
	InUse (see page 8278)	Indicates whether the aggregate value is currently available.

Published Events

	Name	Description
	OnUpdate (see page 8279)	Occurs when the value of the aggregate is recalculated.

Published Properties

	Name	Description
	Active (see page 8274)	Indicates whether the aggregate should be calculated.
	AggregateName (see page 8275)	Specifies the name user interface components use when describing the aggregate.
	Expression (see page 8276)	Specifies the formula used to calculate the aggregate value.
	GroupingLevel (see page 8277)	Specifies the set of records summarized by this aggregate.
	IndexName (see page 8278)	Specifies the index that is used to define subgroups of records.
	Visible (see page 8279)	Indicates whether controls should display the aggregated value.

45.1.3.2 TAggregate.Create Constructor

Creates and initializes an instance of TAggregate (see page 8268).

Pascal

```
constructor Create(Aggregates: TAggregates; ADataSet: TCustomClientDataSet); overload;
reintroduce;
```

C++

```
__fastcall TAggregate(TAggregates Aggregates, TCustomClientDataSet ADataSet);
```

Description

Most applications do not call Create to create an aggregate object. Instead, they call the Add method of the TAggregates (see page 8280) collection to which the aggregate should belong.

The Aggregates or Collection parameter specifies the TAggregates (see page 8280) collection to which the TAggregate (see page 8268) instance should belong.

The ADataSet parameter specifies the client dataset whose data the aggregate summarizes.

See Also

Add (see page 8282)

45.1.3.3 TAggregate.Destroy Destructor

Destroys an instance of TAggregate (see page 8268).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAggregate();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which checks that the TAggregate (see page 8268) reference is not nil before calling Destroy.

Destroy frees the internal buffer that is used for calculating values.

See Also

Free (see page 22300)

45.1.3.4 TAggregate Methods

The methods of the TAggregate class are listed here.

Protected Methods

	Name	Description
	Activate (see page 8271)	This is Activate, a member of class TAggregate.

Public Methods

	Name	Description
V	Assign (see page 8272)	Copies the properties of another TAggregate (see page 8268) object.
V	GetDisplayName (see page 8272)	Specifies the name of the aggregate object as it appears in the Object Inspector.
	Value (see page 8273)	Returns the calculated value of the aggregate.

45.1.3.4.1 TAggregate.Activate Method

Pascal

```
procedure Activate;
```

C++

```
__fastcall Activate();
```

Description

This is Activate, a member of class TAggregate.

45.1.3.4.2 TAggregate.Assign Method

Copies the properties of another TAggregate (see page 8268) object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties of another object to the aggregate. If the Source object is another TAggregate (see page 8268) object, Assign copies the values of its Expression (see page 8276), Active (see page 8274), IndexName (see page 8278), and GroupingLevel (see page 8277) properties. Otherwise, Assign calls the inherited (Delphi) or parent class's (C++) method, which copies the properties of any object that can copy to a TAggregate (see page 8268) from its AssignTo method.

See Also

AssignTo (see page 2931)

Expression (see page 8276)

Active (see page 8274)

GroupingLevel (see page 8277)

IndexName (see page 8278)

45.1.3.4.3 TAggregate.GetDisplayName Method

Specifies the name of the aggregate object as it appears in the Object Inspector.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName overrides the inherited method to provide more meaningful names for the aggregate when it appears in the object inspector. If the AggregateName (see page 8275) property is set, GetDisplayName returns that. Otherwise, if the Expression (see page 8276) property is set, it returns that. If all else fails, it calls the inherited method, which returns the class name with a number suffixed to ensure uniqueness.

See Also

AggregateName (see page 8275)

Expression (see page 8276)

45.1.3.4.4 TAggregate.Value Method

Returns the calculated value of the aggregate.

Pascal

```
function Value: Variant;
```

C++

```
__fastcall Variant Value();
```

Description

Call Value to obtain the evaluation of Expression (see page 8276) over the group of records specified by GroupingLevel (see page 8277). If GroupingLevel (see page 8277) is nonzero (that is, if the aggregate summarizes a subgroup of the records in the client dataset), Value summarizes the data for the group which contains the current record in the client dataset.

If the aggregate is not currently maintained, Value returns Null. An aggregate is not currently maintained when

The Active (see page 8274) property is false.

The DataSet (see page 8275) property is nil (Delphi) or NULL (C++).

The aggregate summarizes a subgroup and the IndexName (see page 8278) property does not match the current index of the client dataset.

See Also

Expression (see page 8276)

GroupingLevel (see page 8277)

Active (see page 8274)

DataSet (see page 8275)

IndexName (see page 8278)

Obtaining Aggregate Values

45.1.3.5 TAggregate Properties

The properties of the TAggregate class are listed here.

Protected Properties

	Name	Description
R	DependentFields (see page 8276)	This is DependentFields, a member of class TAggregate.
	RecBufOfs (see page 8278)	This is RecBufOfs, a member of class TAggregate.

Public Properties

	Name	Description
	AggHandle (see page 8274)	Provides access to the aggregate handle.
R	DataSet (see page 8275)	Indicates the client dataset whose records are summarized by this aggregate.
R	DataSize (see page 8275)	Indicates the number of bytes needed to store the aggregate's value.
R	DataType (see page 8276)	Indicates the data type of the calculated value.
	InUse (see page 8278)	Indicates whether the aggregate value is currently available.

Published Properties

	Name	Description
	Active (see page 8274)	Indicates whether the aggregate should be calculated.
	AggregateName (see page 8275)	Specifies the name user interface components use when describing the aggregate.
	Expression (see page 8276)	Specifies the formula used to calculate the aggregate value.
	GroupingLevel (see page 8277)	Specifies the set of records summarized by this aggregate.
	IndexName (see page 8278)	Specifies the index that is used to define subgroups of records.
	Visible (see page 8279)	Indicates whether controls should display the aggregated value.

45.1.3.5.1 TAggregate.Active Property

Indicates whether the aggregate should be calculated.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Use the Active property to indicate that the aggregate value should be maintained. If Active is false, the value of the aggregate is never calculated.

If Active is true, the aggregate value is calculated unless:

It summarizes records over a group not defined by the current index.

The client dataset's AggregatesActive property is false.

Note: To determine whether the aggregate is currently maintained, read the InUse (see page 8278) property.

See Also

InUse (see page 8278)

AggregatesActive (see page 8461)

Value (see page 8273)

IndexName (see page 8278)

45.1.3.5.2 TAggregate.AggHandle Property

Provides access to the aggregate handle.

Pascal

```
property AggHandle: hDSAggregate;
```

C++

```
__property hDSAggregate AggHandle;
```

Description

AggHandle is the handle for the aggregate instance that is used by the utilities that implement the aggregate object's behavior. Most applications do not need to use this value.

Component writers can use AggHandle, if necessary, with the routines declared in the DSIntf (see page 11667) unit.

See Also

Handle (see page 6758)

45.1.3.5.3 TAggregate.AggregateName Property

Specifies the name user interface components use when describing the aggregate.

Pascal

```
property AggregateName: String;
```

C++

```
__property AnsiString AggregateName;
```

Description

Set AggregateName to provide a name that can be used to label values displayed to the user. When displaying aggregates, read AggregateName to obtain a label for the value.

See Also

Find (see page 8283)

IndexOf (see page 8284)

DisplayName (see page 6756)

45.1.3.5.4 TAggregate.DataSet Property

Indicates the client dataset whose records are summarized by this aggregate.

Pascal

```
property DataSet: TCustomClientDataSet;
```

C++

```
__property TCustomClientDataSet DataSet;
```

Description

Read DataSet to access the client dataset that uses this aggregate.

See Also

IndexName (see page 8278)

Aggregates (see page 8461)

45.1.3.5.5 TAggregate.DataSize Property

Indicates the number of bytes needed to store the aggregate's value.

Pascal

```
property DataSize: Integer;
```

C++

```
__property int DataSize;
```

Description

Read DataSize to determine how much memory is required to store the aggregate's value. This is useful when defining objects

that store aggregate values. For example, the Fields editor uses DataSize when defining aggregate fields.

See Also

DataType (see page 8276)

DataSize (see page 7157)

TAggregateField (see page 6743)

45.1.3.5.6 TAggregate.DataType Property

Indicates the data type of the calculated value.

Pascal

```
property DataType: TFieldType;
```

C++

```
__property TFieldType DataType;
```

Description

Read DataType to determine the data type of the value calculated by the aggregate. This may differ from the data type of the fields that are summarized. For example, an aggregate that counts non-null fields will return an integer (ftInteger), even if the records themselves are some other type.

See Also

DataSize (see page 8275)

DataType (see page 7159)

45.1.3.5.7 TAggregate.DependentFields Property

Pascal

```
property DependentFields: TBits;
```

C++

```
__property TBits DependentFields;
```

Description

This is DependentFields, a member of class TAggregate.

45.1.3.5.8 TAggregate.Expression Property

Specifies the formula used to calculate the aggregate value.

Pascal

```
property Expression: string;
```

C++

```
__property AnsiString Expression;
```

Description

Set Expression to indicate how aggregate values should be calculated. Expression is an expression that includes one or more of the summary operators in the following table:

Operator	Use
Sum	Totals the values for a numeric field or expression
Avg	Computes the average value for a numeric or date-time field or expression
Count	Specifies the number of non-blank values for a field or expression
Min	Indicates the minimum value for a string, numeric, or date-time field or expression
Max	Indicates the maximum value for a string, numeric, or date-time field or expression
Sum(Qty * Price)	legal -- summary of an expression on fields
Max(Field1) - Max(Field2)	legal -- expression on summaries
Avg(DiscountRate) * 100	legal -- expression of summary and constant
Min(Sum(Field1))	illegal -- nested summaries
Count(Field1) - Field2	illegal -- expression of summary and field

The summary operators act on field values or on expressions built from field values using the same operators you use to create filters. (You can't nest summary operators, however.) You can create expressions by using operators on summarized values with other summarized values, or on summarized values and constants. However, you can't combine summarized values with field values, because such expressions are ambiguous (there is no indication of which record should supply the field value.) These rules are illustrated in the following expressions:

Note: To specify the set of records summarized by this expression, use the GroupingLevel (see page 8277) property.

See Also

GroupingLevel (see page 8277)

Value (see page 8273)

Filter (see page 8476)

Specifying Aggregates

45.1.3.5.9 TAggregate.GroupingLevel Property

Specifies the set of records summarized by this aggregate.

Pascal

```
property GroupingLevel: Integer;
```

C++

```
__property int GroupingLevel;
```

Description

Use GroupingLevel to indicate the groups of records over which this aggregate is calculated. When GroupingLevel is 0 (the default), the aggregate summarizes all the records in the client dataset. When GroupingLevel is greater than 0, the aggregate summarizes each group of records that has the same values on the first GroupingLevel fields of the index.

GroupingLevel can be any value from 0 to the number of fields in the index specified by IndexName (see page 8278).

See Also

IndexName (see page 8278)

Expression (see page 8276)

45.1.3.5.10 TAggregate.IndexName Property

Specifies the index that is used to define subgroups of records.

Pascal

```
property IndexName: string;
```

C++

```
__property AnsiString IndexName;
```

Description

Set IndexName to specify an index that sorts the records of DataSet (see page 8275) into subgroups. Each subgroup is defined as a set of records which all have the same values on a set of fields. This defining set of fields is the first GroupingLevel (see page 8277) fields of the index.

IndexName is required for any aggregate that does not summarize all the records in the client dataset (GroupingLevel (see page 8277) > 0). When the index associated with an aggregate is not the current index of the client dataset, its value is unavailable.

See Also

DataSet (see page 8275)

GroupingLevel (see page 8277)

IndexName (see page 8483)

InUse (see page 8278)

45.1.3.5.11 TAggregate.InUse Property

Indicates whether the aggregate value is currently available.

Pascal

```
property InUse: Boolean;
```

C++

```
__property Boolean InUse;
```

Description

Read InUse to determine whether the Value (see page 8273) method returns a summarized value. When InUse is false, Value (see page 8273) returns Null. When InUse is true, the aggregate is maintained so that it calculates the formula specified by Expression (see page 8276) for the current set of records, including any edits to records in the current group.

See Also

Value (see page 8273)

Active (see page 8274)

Expression (see page 8276)

45.1.3.5.12 TAggregate.RecBufOfs Property

Pascal

```
property RecBufOfs: Integer;
```

C++

```
__property int RecBufOfs;
```

Description

This is RecBufOfs, a member of class TAggregate.

45.1.3.5.13 TAggregate.Visible Property

Indicates whether controls should display the aggregated value.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Set Visible to indicate to controls that the aggregate value should be hidden. When Visible is false, its value is still calculated.

To prevent the aggregate from being calculated, use the Active (see page 8274) property instead.

See Also

InUse (see page 8278)

Active (see page 8274)

45.1.3.6 TAggregate Events

The events of the TAggregate class are listed here.

Published Events

	Name	Description
	OnUpdate (see page 8279)	Occurs when the value of the aggregate is recalculated.

45.1.3.6.1 TAggregate.OnUpdate Event

Occurs when the value of the aggregate is recalculated.

Pascal

```
property OnUpdate: TAggUpdateEvent;
```

C++

```
__property TAggUpdateEvent OnUpdate;
```

Description

Write an OnUpdate event handler to respond when the aggregate value is recomputed. For example, write code in an OnUpdate event handler to invalidate any controls that display the value of the aggregate so that they can repaint with the new value.

See Also

Value (see page 8273)

InUse (see page 8278)

Invalidate

45.1.4 TAggregates Class

TAggregates is a collection of TAggregate (see page 8268) objects.

Class Hierarchy

System::TObject → Classes.TPersistent → Classes.TCollection → DBClient.TAggregates

Pascal

```
TAggregates = class(TCollection);
```

C++

```
class TAggregates : public TCollection;
```

File

DBClient

Description

TAggregates is the type of the Aggregates property, which represents all the maintained aggregates for a client dataset. A single TAggregate (see page 8268) object represents each maintained aggregate. Maintained aggregates summarize data over the records in the client dataset.

TAggregates can include a mix of active and inactive aggregate objects. Active aggregates are updated dynamically as the data in the client dataset is edited. Inactive aggregates define a formula for summarizing data and a group of records to summarize, but these formulas are not evaluated. Aggregates can be inactive because they are not currently needed, or because the index that defines the group of records they summarize is inactive.

See Also

TAggregate (see page 8268)

TCustomClientDataSet (see page 8373)

45.1.4.1 TAggregates Members

The following tables list the members exposed by TAggregates.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 8283)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.

	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 8282)	Creates a new TAggregate (see page 8268) object and adds it to the end of the Items (see page 8285) property array.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 8283)	Removes all aggregates from the Items (see page 8285) property array.
	Create (see page 8282)	Creates and initializes a TAggregates (see page 8280) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
	Destroy (see page 2735)	Destroys the collection and each item in it.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Find (see page 8283)	Returns the TAggregate (see page 8268) object with a specified name.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	IndexOf (see page 8284)	Returns the index of the TAggregate (see page 8268) object with a specified name.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 8285)	Provides indexed access to the TAggregate (see page 8268) objects in the collection.

45.1.4.2 TAggregates.Create Constructor

Creates and initializes a TAggregates (see page 8280) object.

Pascal

```
constructor Create(Owner: TPersistent);
```

C++

```
__fastcall TAggregates(TPersistent * Owner);
```

Description

Applications should not need to call the Create method directly. The client dataset that uses TAggregates (see page 8280) to represent its maintained aggregates creates this object from its own constructor.

45.1.4.3 TAggregates Methods

The methods of the TAggregates class are listed here.

Protected Methods

	Name	Description
V	GetOwner (see page 8283)	Returns the owner of an object.

Public Methods

	Name	Description
	Add (see page 8282)	Creates a new TAggregate (see page 8268) object and adds it to the end of the Items (see page 8285) property array.
	Clear (see page 8283)	Removes all aggregates from the Items (see page 8285) property array.
	Find (see page 8283)	Returns the TAggregate (see page 8268) object with a specified name.
	IndexOf (see page 8284)	Returns the index of the TAggregate (see page 8268) object with a specified name.

45.1.4.3.1 TAggregates.Add Method

Creates a new TAggregate (see page 8268) object and adds it to the end of the Items (see page 8285) property array.

Pascal

```
function Add: TAggregate;
```

C++

```
__fastcall TAggregate Add();
```

Description

Call Add to add a new aggregate to the set of aggregates that can be used with a client dataset. Once the aggregate is created and added using Add, its properties can be set to indicate how it is used.

See Also

Items (see page 8285)

45.1.4.3.2 TAggregates.Clear Method

Removes all aggregates from the Items (see page 8285) property array.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the Items (see page 8285) property array. After Clear frees each aggregate in the Items (see page 8285) array, it informs the associated client dataset so that it can update it's ActiveAggs property.

See Also

Items (see page 8285)

ActiveAggs (see page 8460)

45.1.4.3.3 TAggregates.Find Method

Returns the TAggregate (see page 8268) object with a specified name.

Pascal

```
function Find(const DisplayName: string): TAggregate;
```

C++

```
__fastcall TAggregate Find(const AnsiString DisplayName);
```

Description

Call Find to obtain a specified TAggregate (see page 8268) object from the Items (see page 8285) property array. The DisplayName parameter specifies the value of the aggregate's AggregateName property. If there is no aggregate with the specified name in the Items (see page 8285) property array, Find returns nil (Delphi) or NULL (C++).

See Also

AggregateName (see page 8275)

IndexOf (see page 8284)

Items (see page 8285)

45.1.4.3.4 TAggregates.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

45.1.4.3.5 TAggregates.IndexOf Method

Returns the index of the TAggregate (see page 8268) object with a specified name.

Pascal

```
function IndexOf(const DisplayName: string): Integer;
```

C++

```
__fastcall int IndexOf(const AnsiString DisplayName);
```

Description

Call IndexOf to obtain the index of a specified TAggregate (see page 8268) object in the Items (see page 8285) property array. The DisplayName parameter specifies the value of the aggregate's AggregateName property. If there is no aggregate with the specified name in the Items (see page 8285) property array, IndexOf returns -1.

See Also

AggregateName (see page 8275)

Find (see page 8283)

Items (see page 8285)

45.1.4.4 TAggregates Properties

The properties of the TAggregates class are listed here.

Public Properties

	Name	Description
	Items (see page 8285)	Provides indexed access to the TAggregate (see page 8268) objects in the collection.

45.1.4.4.1 TAggregates.Items Property

Provides indexed access to the TAggregate (see page 8268) objects in the collection.

Pascal

```
property Items [Index: Integer]: TAggregate;
```

C++

```
__property TAggregate Items[int Index];
```

Description

Use Items to access individual TAggregate (see page 8268) objects maintained by TAggregates (see page 8280).

In Delphi, Items is the default property of TAggregates (see page 8280), so the name of the Items property can be left off when using this property. Thus the line

```
FirstAggregate := TClientDataSet1.Aggregates.Items[0];
```

can be written

```
FirstAggregate := TClientDataSet1.Aggregates[0];
```

See Also

Aggregates (see page 8461)

operator_sb

45.1.5 TClientBlobStream Class

TClientBlobStream lets applications read from or write to BLOB fields in a client dataset.

Class Hierarchy

Pascal

```
TClientBlobStream = class(TMemoryStream);
```

C++

```
class TClientBlobStream : public TMemoryStream;
```

File

DBClient

Description

Use TClientBlobStream to access or modify the value of a Binary Large Object (BLOB) field in a client dataset. BLOB fields are TBlobField objects and descendants of TBlobField such as TGraphicField and TMemoField. BLOB fields in client datasets use client BLOB streams to implement many of their data access properties and methods.

TClientBlobStream allows objects that have no specialized knowledge of how data is stored in a BLOB field to read or write such data by employing the uniform stream interface.

To use a client BLOB stream, create an instance of TClientBlobStream, use the methods of the stream to read or write the data, and then free the BLOB stream. Do not use the same instance of TClientBlobStream to access data from more than one record. Instead, create a new TClientBlobStream object every time you need to read or write BLOB data on a new record.

Note: Explicitly creating a TClientBlobStream instance does not ensure that the current record has fetched its BLOB data. To ensure that the BLOB field contains its data before reading from the BLOB stream, call the client dataset's FetchBlobs method before creating the BLOB stream or use the client dataset's CreateBlobStream method to create the BLOB stream indirectly.

See Also

TBlobField (see page 6801)

TGraphicField (see page 7257)

TMemoField (see page 7348)

TClientDataSet (see page 8289)

TCustomClientDataSet (see page 8373)

TCustomClientDataSet (see page 8373)

45.1.5.1 TClientBlobStream Members

The following tables list the members exposed by TClientBlobStream.

Protected Methods

	Name	Description
	GetSize (see page 2982)	This is GetSize, a member of class TStream.
	Realloc (see page 2910)	This is Realloc, a member of class TMemoryStream.
	SetPointer (see page 2823)	Replaces the memory buffer associated with the memory stream.
	SetSize (see page 2989)	This is the overview for the SetSize method overload.

Protected Properties

	Name	Description
	Capacity (see page 2912)	This is Capacity, a member of class TMemoryStream.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2909)	Frees the memory buffer, discarding all data associated with the memory stream.
	CopyFrom (see page 2979)	Copies a specified number of bytes from one stream to another.
	Create (see page 8287)	Creates and initializes an instance of TClientBlobStream (see page 8285).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 8288)	Frees an instance of TClientBlobStream (see page 8285).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FixupResourceHeader (see page 2981)	Patches the resource header for a resource that has been written to the stream.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LoadFromFile (see page 2909)	Loads the entire contents of a file into the memory buffer.
	LoadFromStream (see page 2909)	Loads the entire contents of a stream into the memory buffer.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Read (see page 2817)	Reads up to Count bytes from the memory stream into Buffer and advances the current position of the stream by the number of bytes read.
A	Read (see page 2982)	Introduces an abstract or pure virtual method responsible for reading from the stream.
	ReadBuffer (see page 2983)	Reads bytes from the stream into Buffer.
	ReadComponent (see page 2983)	Initiates streaming of components and their properties.
	ReadComponentRes (see page 2987)	Reads components and their properties from a stream in a resource file format.
	ReadResHeader (see page 2987)	Reads a resource-file header from the stream.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 2818)	Writes the entire contents of the memory stream to the file with a given file name.
	SaveToStream (see page 2818)	Writes the entire contents of the memory stream to the stream object specified by Stream.
V	Seek (see page 2819)	Moves the current position of the stream by Offset bytes, relative to the origin specified by Origin.
V	Seek (see page 2988)	This is the overview for the Seek method overload.
V	SetSize (see page 2910)	Sets the Size (see page 2997) property of the memory stream.
	Truncate (see page 8288)	Discards all data in the BLOB field from the current position on.
V	Write (see page 8289)	Writes Count bytes from Buffer to the current position in the field and updates the current position by Count bytes.
	WriteBuffer (see page 2990)	Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.
	WriteComponent (see page 2990)	Initiates the writing of components and their properties to a stream.
	WriteComponentRes (see page 2994)	Writes components and their properties to a stream from a resource file format.
	WriteDescendent (see page 2994)	Streams components and their properties in inherited forms.
	WriteDescendentRes (see page 2995)	Streams components and their properties in inherited forms.
	WriteResourceHeader (see page 2995)	Writes a resource-file header to the stream.

Public Properties

	Name	Description
R	Memory (see page 2823)	Provides direct access to the memory pool allocated for the memory stream.
	Position (see page 2996)	Indicates the current offset into the stream for reading and writing.
	Size (see page 2997)	Indicates the size in bytes of the stream.

45.1.5.2 TClientBlobStream.Create Constructor

Creates and initializes an instance of TClientBlobStream (see page 8285).

Pascal

```
constructor Create(Field: TBlobField; Mode: TBlobStreamMode);
```

C++

```
__fastcall TClientBlobStream(TBlobField Field, TBlobStreamMode Mode);
```

Description

Call Create to obtain an instance of TClientBlobStream (see page 8285) for reading from or writing to a specific TBlobField object in a client dataset.

Create links the TClientBlobStream (see page 8285) to the field object specified by the Field parameter. Mode specifies whether the BLOB stream will be used to read data (bmRead), write data (bmWrite) or modify data (bmReadWrite).

Tip: Because Client datasets sometimes fetch BLOB data on demand, it is a good idea to use the Client dataset's CreateBlobStream method rather than calling the Client BLOB stream's constructor. CreateBlobStream ensures that BLOB data is fetched for the current record unless the FetchOnDemand property is false.

See Also

CreateBlobStream (see page 8401)

45.1.5.3 TClientBlobStream.Destroy Destructor

Frees an instance of TClientBlobStream (see page 8285).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TClientBlobStream();
```

Description

Do not call Destroy directly in an application. Instead call Free. Free verifies that the TBlobStream object is not nil and only then calls Destroy.

Destroy the TClientBlobStream (see page 8285) object by calling Free when it is no longer needed for reading from or writing to the BLOB field. Client blob streams should be destroyed when the client dataset changes the current record

Destroy triggers an OnDataChange event if the BLOB stream was used to overwrite or modify the data in the field. Destroy then frees any buffers that were allocated to handle the data.

See Also

Free (see page 22300)

OnDataChange (see page 9497)

45.1.5.4 TClientBlobStream Methods

The methods of the TClientBlobStream class are listed here.

Public Methods

	Name	Description
	Truncate (see page 8288)	Discards all data in the BLOB field from the current position on.
V	Write (see page 8289)	Writes Count bytes from Buffer to the current position in the field and updates the current position by Count bytes.

45.1.5.4.1 TClientBlobStream.Truncate Method

Discards all data in the BLOB field from the current position on.

Pascal

```
procedure Truncate;
```

C++

```
__fastcall Truncate();
```

Description

Use Truncate to limit the size of the BLOB data. Calling Truncate does the same thing as setting the Size property to the value of the Position property. Calling Truncate when the current position is 0 clears the contents of the BLOB field.

Note: Do not call Truncate when the TBlobStream was created in bmRead mode.

See Also

Size (see page 2997)

Position (see page 2996)

Create (see page 8287)

45.1.5.4.2 TClientBlobStream.Write Method

Writes Count bytes from Buffer to the current position in the field and updates the current position by Count bytes.

Pascal

```
function Write(const Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  Buffer, Longint Count);
```

Description

Use Write to write Count bytes to the BLOB field, starting at the current position. The Write method for TClientBlobStream (see page 8285) always writes the entire Count bytes, as BLOB data does not necessarily include a termination character. Thus, Write is equivalent to the WriteBuffer method.

All the other data-writing methods of a BLOB stream (WriteBuffer, WriteComponent) call Write to do their actual writing.

Note: Do not call Write when the TClientBlobStream (see page 8285) was created in bmRead mode.

See Also

Position (see page 2996)

Create (see page 8287)

WriteBuffer (see page 2990)

45.1.6 TClientDataSet Class

TClientDataSet implements a database-independent dataset.

Class Hierarchy

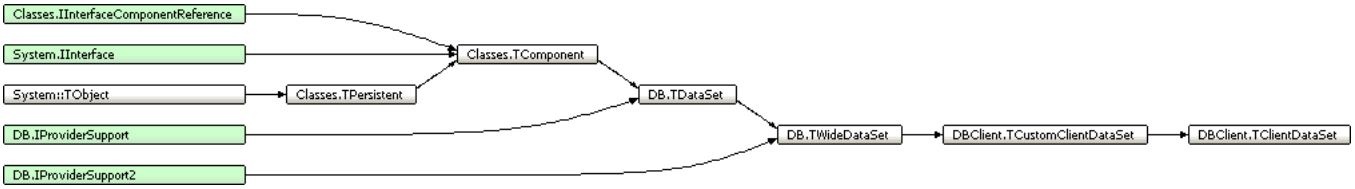

Pascal

```
TClientDataSet = class(TCustomClientDataSet);
```

C++

```
class TClientDataSet : public TCustomClientDataSet;
```

File

DBClient

Description

TClientDataSet represents an in-memory dataset. A client dataset can be used as

A fully-functional, stand-alone, file-based dataset for single-tiered database applications. When used in this manner, the client dataset represents the data stored in a dedicated file on the user's hard disk.

A local in-memory buffer of the records from another dataset. The other (source) dataset can reside in the same form or data module as the client dataset (for example, when the client dataset provides navigation and editing support for the data from a unidirectional dataset). The source dataset can also reside on a separate system when the client dataset supports the client portion of a multi-tiered database application.

Note: The two functions described above are not mutually exclusive. An application can be designed to support the option of working with data offline, using the "briefcase" model. On site, the application uses TClientDataSet to communicate with the database server associated with a source dataset. When a user works off site, the client dataset writes a snapshot of the data to the hard disk. The application works with this snapshot off site, with the client dataset acting as a file-based dataset in a single-tiered application.

When a client dataset represents the data from another dataset, it communicates with that dataset through a dataset provider. The client dataset communicates with this provider through the IAppServer interface. When the provider is in the same form or data module as the client dataset, the IAppServer interface is implemented by a hidden object that is created for you. When used in a client application as part of a multi-tiered application, the client dataset passes all calls to the provider through a remote data module's IAppServer interface.

See Also

TDataSetProvider (see page 18508)

TDispatchConnection (see page 16017)

IAppServer (see page 16450)

TSQLClientDataSet

TBDEClientDataSEt

TIBClientDataSet

Types of Datasets

Choosing the Type of Dataset for Caching Updates

45.1.6.1 TClientDataSet Members

The following tables list the members exposed by TClientDataSet.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	ActivateFilters (see page 8391)	This is ActivateFilters, a member of class TCustomClientDataSet.
V	AddDataPacket (see page 8391)	This is AddDataPacket, a member of class TCustomClientDataSet.
	AddFieldDesc (see page 8391)	This is AddFieldDesc, a member of class TCustomClientDataSet.
	AllocKeyBuffers (see page 8392)	This is AllocKeyBuffers, a member of class TCustomClientDataSet.
V	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
V	AllocRecordBuffer (see page 8393)	This is AllocRecordBuffer, a member of class TCustomClientDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Check (see page 8397)	This is Check, a member of class TCustomClientDataSet.
V	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
V	CheckDetailRecords (see page 8397)	This is CheckDetailRecords, a member of class TCustomClientDataSet.
V	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
V	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
V	CheckProviderEOF (see page 8397)	This is CheckProviderEOF, a member of class TCustomClientDataSet.
	CheckSetKeyMode (see page 8397)	This is CheckSetKeyMode, a member of class TCustomClientDataSet.
V	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
V	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
V	ClearCalcFields (see page 8398)	This is ClearCalcFields, a member of class TCustomClientDataSet.
V	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
V	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
V	CloseCursor (see page 8400)	This is CloseCursor, a member of class TCustomClientDataSet.
	ConstraintsStored (see page 8401)	This is ConstraintsStored, a member of class TCustomClientDataSet.
V	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
V	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
V	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
V	DataConvert (see page 8404)	This is DataConvert, a member of class TCustomClientDataSet.
V	DataEvent (see page 6924)	This is DataEvent, a member of class TDataSet.
V	DataEvent (see page 8404)	This is DataEvent, a member of class TCustomClientDataSet.
	DeactivateFilters (see page 8405)	This is DeactivateFilters, a member of class TCustomClientDataSet.
V	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
V	DefChanged (see page 8405)	This is DefChanged, a member of class TCustomClientDataSet.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	DefineProperties (see page 8405)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
	DestroyLookupCursor (see page 8406)	This is DestroyLookupCursor, a member of class TCustomClientDataSet.
	DoAfterApplyUpdates (see page 8407)	This is DoAfterApplyUpdates, a member of class TCustomClientDataSet.
	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
	DoAfterExecute (see page 8407)	This is DoAfterExecute, a member of class TCustomClientDataSet.
	DoAfterGetParams (see page 8407)	This is DoAfterGetParams, a member of class TCustomClientDataSet.
	DoAfterGetRecords (see page 8407)	This is DoAfterGetRecords, a member of class TCustomClientDataSet.
	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterRowRequest (see page 8408)	This is DoAfterRowRequest, a member of class TCustomClientDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoApplyUpdates (see page 8408)	This is DoApplyUpdates, a member of class TCustomClientDataSet.
	DoBeforeApplyUpdates (see page 8408)	This is DoBeforeApplyUpdates, a member of class TCustomClientDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeExecute (see page 8408)	This is DoBeforeExecute, a member of class TCustomClientDataSet.
	DoBeforeGetParams (see page 8408)	This is DoBeforeGetParams, a member of class TCustomClientDataSet.
	DoBeforeGetRecords (see page 8409)	This is DoBeforeGetRecords, a member of class TCustomClientDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeInsert (see page 8409)	This is DoBeforeInsert, a member of class TCustomClientDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeRowRequest (see page 8409)	This is DoBeforeRowRequest, a member of class TCustomClientDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoExecute (see page 8409)	This is DoExecute, a member of class TCustomClientDataSet.
	DoGetRecords (see page 8410)	This is DoGetRecords, a member of class TCustomClientDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	DoOnNewRecord (see page 8410)	This is DoOnNewRecord, a member of class TCustomClientDataSet.
	DoRowRequest (see page 8410)	This is DoRowRequest, a member of class TCustomClientDataSet.
	FetchMoreData (see page 8415)	This is FetchMoreData, a member of class TCustomClientDataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FindRecord (see page 8417)	This is FindRecord, a member of class TCustomClientDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeKeyBuffers (see page 8417)	This is FreeKeyBuffers, a member of class TCustomClientDataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
	FreeRecordBuffer (see page 8417)	This is FreeRecordBuffer, a member of class TCustomClientDataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
	GetAggregateValue (see page 8418)	This is GetAggregateValue, a member of class TCustomClientDataSet.
	GetAppServer (see page 8418)	This is GetAppServer, a member of class TCustomClientDataSet.
	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
	GetBookmarkData (see page 8418)	This is GetBookmarkData, a member of class TCustomClientDataSet.
	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.

V	GetBookmarkFlag (see page 8418)	This is GetBookmarkFlag, a member of class TCustomClientDataSet.
V	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
V	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
V	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
V	GetCanModify (see page 8418)	This is GetCanModify, a member of class TCustomClientDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
V	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
V	GetDataSource (see page 8419)	This is GetDataSource, a member of class TCustomClientDataSet.
V	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
V	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIndexField (see page 8421)	This is GetIndexField, a member of class TCustomClientDataSet.
	GetIndexFieldCount (see page 8421)	This is GetIndexFieldCount, a member of class TCustomClientDataSet.
V	GetIsClone (see page 8422)	This is GetIsClone, a member of class TCustomClientDataSet.
V	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
V	GetIsIndexField (see page 8422)	This is GetIsIndexField, a member of class TCustomClientDataSet.
	GetKeyBuffer (see page 8423)	This is GetKeyBuffer, a member of class TCustomClientDataSet.
	GetKeyExclusive (see page 8423)	This is GetKeyExclusive, a member of class TCustomClientDataSet.
	GetKeyFieldCount (see page 8423)	This is GetKeyFieldCount, a member of class TCustomClientDataSet.
V	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
V	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
V	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
V	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
V	GetRecNo (see page 8425)	This is GetRecNo, a member of class TCustomClientDataSet.
A	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
V	GetRecord (see page 8425)	This is GetRecord, a member of class TCustomClientDataSet.
V	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
V	GetRecordCount (see page 8426)	This is GetRecordCount, a member of class TCustomClientDataSet.
V	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
V	GetRecordSize (see page 8426)	This is GetRecordSize, a member of class TCustomClientDataSet.
V	GetRemoteServer (see page 8426)	This is GetRemoteServer, a member of class TCustomClientDataSet.
V	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
V	GetStateFieldValue (see page 8426)	This is GetStateFieldValue, a member of class TCustomClientDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
	InitKeyBuffer (see page 8428)	This is InitKeyBuffer, a member of class TCustomClientDataSet.
V	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
V	InitRecord (see page 8428)	This is InitRecord, a member of class TCustomClientDataSet.
V	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
V	InternalAddRecord (see page 8428)	This is InternalAddRecord, a member of class TCustomClientDataSet.
V	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
V	InternalCancel (see page 8429)	This is InternalCancel, a member of class TCustomClientDataSet.
A	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.

	InternalClose (see page 8429)	This is InternalClose, a member of class TCustomClientDataSet.
	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
	InternalDelete (see page 8429)	This is InternalDelete, a member of class TCustomClientDataSet.
	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
	InternalEdit (see page 8429)	This is InternalEdit, a member of class TCustomClientDataSet.
	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
	InternalFirst (see page 8430)	This is InternalFirst, a member of class TCustomClientDataSet.
	InternalGetOptionalParam (see page 8430)	This is InternalGetOptionalParam, a member of class TCustomClientDataSet.
	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
	InternalGotoBookmark (see page 8430)	This is InternalGotoBookmark, a member of class TCustomClientDataSet.
	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
	InternalHandleException (see page 8430)	This is InternalHandleException, a member of class TCustomClientDataSet.
	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
	InternalInitFieldDefs (see page 8430)	This is InternalInitFieldDefs, a member of class TCustomClientDataSet.
	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
	InternalInitRecord (see page 8431)	This is InternalInitRecord, a member of class TCustomClientDataSet.
	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
	InternalInsert (see page 8431)	This is InternalInsert, a member of class TCustomClientDataSet.
	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
	InternalLast (see page 8431)	This is InternalLast, a member of class TCustomClientDataSet.
	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
	InternalOpen (see page 8431)	This is InternalOpen, a member of class TCustomClientDataSet.
	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
	InternalPost (see page 8432)	This is InternalPost, a member of class TCustomClientDataSet.
	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
	InternalRefresh (see page 8432)	This is InternalRefresh, a member of class TCustomClientDataSet.
	InternalSetOptionalParam (see page 8432)	This is InternalSetOptionalParam, a member of class TCustomClientDataSet.
	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	InternalSetToRecord (see page 8432)	This is InternalSetToRecord, a member of class TCustomClientDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
	IsCursorOpen (see page 8432)	This is IsCursorOpen, a member of class TCustomClientDataSet.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
	Loaded (see page 8433)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 8436)	This is LocateRecord, a member of class TCustomClientDataSet.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 8437)	Forwards notification messages to all owned components.
	OpenCursor (see page 6976)	This is OpenCursor, a member of class TDataSet.
	OpenCursor (see page 8438)	This is OpenCursor, a member of class TCustomClientDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PostKeyBuffer (see page 8439)	This is PostKeyBuffer, a member of class TCustomClientDataSet.
	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSExecuteStatement (see page 7535)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 6982)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 7537)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 8439)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandTextW (see page 7537)	This is PSGetCommandTextW, a member of class TWideDataSet.

	PSGetCommandType (see page 6982)	Gets the command type to be executed by PSExecute.
	PSGetCommandType (see page 8440)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 8440)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7538)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetKeyFieldsW (see page 7538)	This is PSGetKeyFieldsW, a member of class TWideDataSet.
	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7538)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetQuoteCharW (see page 7539)	This is PSGetQuoteCharW, a member of class TWideDataSet.
	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7539)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetTableNameW (see page 7539)	This is PSGetTableNameW, a member of class TWideDataSet.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetCommandText (see page 7540)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadDataPacket (see page 8441)	This is ReadDataPacket, a member of class TCustomClientDataSet.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	RefreshInternalCalcFields (see page 8441)	This is RefreshInternalCalcFields, a member of class TCustomClientDataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	ResetAggField (see page 8442)	This is ResetAggField, a member of class TCustomClientDataSet.
	ResetCursorRange (see page 8443)	This is ResetCursorRange, a member of class TCustomClientDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAggregates (see page 8445)	This is SetAggregates, a member of class TCustomClientDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppServer (see page 8446)	This is SetAppServer, a member of class TCustomClientDataSet.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkData (see page 8446)	This is SetBookmarkData, a member of class TCustomClientDataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkFlag (see page 8446)	This is SetBookmarkFlag, a member of class TCustomClientDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.

	SetCommandText (see page 8447)	This is SetCommandText, a member of class TCustomClientDataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetCursorRange (see page 8447)	This is SetCursorRange, a member of class TCustomClientDataSet.
	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDataSetField (see page 8447)	This is SetDataSetField, a member of class TCustomClientDataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldData (see page 8447)	This is SetFieldData, a member of class TCustomClientDataSet.
	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
	SetFilterData (see page 8448)	This is SetFilterData, a member of class TCustomClientDataSet.
	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
	SetFiltered (see page 8448)	This is SetFiltered, a member of class TCustomClientDataSet.
	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
	SetFilterOptions (see page 8448)	This is SetFilterOptions, a member of class TCustomClientDataSet.
	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
	SetFilterText (see page 8448)	This is SetFilterText, a member of class TCustomClientDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetIndexField (see page 8448)	This is SetIndexField, a member of class TCustomClientDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetKeyBuffer (see page 8449)	This is SetKeyBuffer, a member of class TCustomClientDataSet.
	SetKeyExclusive (see page 8449)	This is SetKeyExclusive, a member of class TCustomClientDataSet.
	SetKeyFieldCount (see page 8450)	This is SetKeyFieldCount, a member of class TCustomClientDataSet.
	SetKeyFields (see page 8450)	This is SetKeyFields, a member of class TCustomClientDataSet.
	SetLinkRanges (see page 8450)	This is SetLinkRanges, a member of class TCustomClientDataSet.
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetOnFilterRecord (see page 8450)	This is SetOnFilterRecord, a member of class TCustomClientDataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetProviderEOF (see page 8452)	This is SetProviderEOF, a member of class TCustomClientDataSet.
	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
	SetRecNo (see page 8454)	This is SetRecNo, a member of class TCustomClientDataSet.
	SetRemoteServer (see page 8454)	This is SetRemoteServer, a member of class TCustomClientDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	SwitchToIndex (see page 8455)	This is SwitchToIndex, a member of class TCustomClientDataSet.
	SyncCursors (see page 8455)	This is SyncCursors, a member of class TCustomClientDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.
	UpdateIndexDefs (see page 8455)	This is UpdateIndexDefs, a member of class TCustomClientDataSet.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.

	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteDataPacket (see page 8456)	This is WriteDataPacket, a member of class TCustomClientDataSet.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
	DSBase (see page 8471)	Specifies the interface the client dataset uses to communicate with classes in Midas.dll or Midaslib.dcu (on Windows) or midas.so (on Linux).
R	DSCursor (see page 8471)	Provides access to the interface for the client dataset's data cursor.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
R	IsClone (see page 8484)	This is IsClone, a member of class TCustomClientDataSet.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
	ProviderEOF (see page 8491)	Indicates whether the client dataset has all of its data.
	Reserved (see page 7035)	For internal use only.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
	AddIndex (see page 8391)	Creates a new index for the client dataset.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendData (see page 8393)	Adds a newly fetched packet of records to those already stored in the client dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
	ApplyRange (see page 8393)	Applies a range to the dataset.
V	ApplyUpdates (see page 8394)	Sends all updated, inserted, and deleted records from the client dataset to the provider for writing to the database.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
V	BookmarkValid (see page 8395)	Tests the validity of a specified bookmark.
V	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
V	Cancel (see page 8395)	Cancels unposted changes to the current record.
	CancelRange (see page 8396)	Removes any ranges currently in effect for the client dataset.
	CancelUpdates (see page 8396)	Clears all pending edits and clears the change log.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
V	CloneCursor (see page 8398)	Shares the data belonging to another client dataset.
	Close (see page 6920)	Closes a dataset.
V	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
V	CompareBookmarks (see page 8400)	Compares two bookmarks.
	ConstraintsDisabled (see page 8400)	Indicates whether constraints are enabled for a client dataset.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 6904)	Creates an instance of a TDataSet (see page 6894) component.
V	Create (see page 8386)	Creates and initializes instance of TCustomClientDataSet (see page 8373).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
V	CreateBlobStream (see page 8401)	Returns a TClientBlobStream (see page 8285) object for reading or writing the data in a specified blob field.
	CreateDataSet (see page 8401)	Creates a new, empty client dataset.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
V	DataRequest (see page 8404)	Generates an OnDataRequest event on the associated provider
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
	DeleteIndex (see page 8406)	Deletes an index for the client dataset.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 6904)	Destroys the TDataSet (see page 6894) instance.
V	Destroy (see page 8387)	Destroys the instance of a client dataset component.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableConstraints (see page 8406)	Disables the enforcement of constraints.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EditKey (see page 8410)	Enables modification of the search key buffer.
	EditRangeEnd (see page 8411)	Enables changing the ending value for an existing range.
	EditRangeStart (see page 8411)	Enables changing the starting value for an existing range.
V	EmptyDataSet (see page 8412)	Removes all records from the dataset.
	EnableConstraints (see page 8412)	Reenables enforcement of constraints when records are posted.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
V	Execute (see page 8413)	Executes an SQL command by the provider's dataset.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FetchBlobs (see page 8414)	Retrieves BLOB field values from the provider.
	FetchDetails (see page 8414)	Retrieves nested detail datasets from the provider.
	FetchParams (see page 8415)	Retrieves parameter values from the provider.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
V	FindKey (see page 8416)	Searches for a record containing specified field values.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNearest (see page 8416)	Moves the cursor to the record that most closely matches a specified set of key values.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
V	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
V	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
V	GetCurrentRecord (see page 8419)	Retrieves the current record into a buffer.
V	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
V	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
V	GetFieldData (see page 8419)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
V	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetGroupState (see page 8420)	Indicates where the current record sits within a specified group of records.
	GetIndexInfo (see page 8421)	Retrieves information about the current index.
	GetIndexNames (see page 8422)	Retrieves a list of available indexes for a client dataset.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetNextPacket (see page 8423)	Retrieves the next block of records from the provider.
	GetOptionalParam (see page 8425)	Retrieves a custom parameter from the data packet.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	GotoCurrent (see page 8427)	Sets the current record in this dataset the same as the current record in another, cloned dataset.
	GotoKey (see page 8427)	Moves the cursor to a record specified by the current key.
	GotoNearest (see page 8427)	Moves the cursor to the record that most closely matches the current key.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.

	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
V	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	Last (see page 6968)	Moves to the last record in the dataset.
	LoadFromFile (see page 8433)	Loads a client dataset's data from a file.
	LoadFromStream (see page 8435)	Loads a client dataset's data from a stream.
V	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Locate (see page 8435)	Searches the dataset for a specified record and makes that record current.
V	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
V	Lookup (see page 8436)	Retrieves field values from a record that matches specified search values.
	MergeChangeLog (see page 8437)	Merges any data changes in the change log into the client dataset, overwriting existing values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	Open (see page 6976)	Opens the dataset.
V	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
V	Post (see page 8438)	Writes a modified record to the Data (see page 8467) property or the change log.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSEndTransaction (see page 7589)	Ends a transaction that was started by a call to PSStartTransaction (see page 7597).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecute (see page 7590)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSExecuteStatement (see page 7590)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 7591)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 7579)	Gets the SQL command to be executed by PSExecute (see page 7578).
	PSGetCommandText (see page 7591)	Gets the SQL command to be executed by PSExecute (see page 7590).
	PSGetCommandType (see page 7580)	Gets the command type to be executed by PSExecute (see page 7578).
	PSGetCommandType (see page 7591)	Gets the command type to be executed by PSExecute (see page 7590).
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 7592)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 7592)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7593)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetParams (see page 7593)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.

	PSGetQuoteChar (see page 7594)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7594)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSGetUpdateException (see page 7594)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 7595)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 7595)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 7596)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 7596)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7585)	Assigns a specified SQL command to be executed by PSExecute (see page 7578).
	PSSetCommandText (see page 7596)	Assigns a specified SQL command to be executed by PSExecute (see page 7590).
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 7597)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).
	PSStartTransaction (see page 7597)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7589).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 7598)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Reconcile (see page 8441)	Clears successfully updated records from the client dataset's cache of changes.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RefreshRecord (see page 8442)	Updates the base values of the current record to match the current values of the provider's dataset.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
	RevertRecord (see page 8443)	Undoes edits to the current record.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 8443)	Saves a client dataset's data to an external file.
	SaveToStream (see page 8445)	Saves a client dataset's data to a stream.
	SetAltRecBuffers (see page 8446)	Sets record buffers when accessing field values.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetKey (see page 8449)	Enables setting of keys and ranges for a dataset prior to a search.
V	SetOptionalParam (see page 8450)	Saves a custom parameter with the data packet.
	SetProvider (see page 8451)	Associates the client dataset with a provider in the same process space.
	SetRange (see page 8452)	Sets the starting and ending values of a range, and applies it.
	SetRangeEnd (see page 8453)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
	SetRangeStart (see page 8454)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
	UndoLastChange (see page 8455)	Undoes the last edit, insert, or delete operation to a record in the client dataset.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.
V	UpdateStatus (see page 8456)	Reports the update status for the current record in the dataset.

Public Properties

	Name	Description
R	ActiveAggs (see page 8460)	Lists the maintained aggregates supported by the current index.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AppServer (see page 8462)	Provides access to the interface the client dataset uses to communicate with a provider.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ChangeCount (see page 8463)	Indicates the number of changes in the change log.
R	CloneSource (see page 8464)	Indicates the client dataset (if any) with which this client dataset shares data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Data (see page 8467)	Represents the data of the client dataset in a transportable format.
R	DataSize (see page 8469)	Indicates the number of bytes required to represent the Data (see page 8467) property.
R	DataSource (see page 7014)	Represents the data source of another dataset that supplies values to the dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Delta (see page 8470)	Represents a packet of change log data.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
R	GroupingLevel (see page 8479)	Indicates the depth of grouping support provided by the current index.
R	HasAppServer (see page 8480)	Indicates whether the client dataset is connected to a provider.
R	IndexFieldCount (see page 8481)	Indicates the number of fields that make up the current index.
	IndexFields (see page 8482)	Specifies the fields associated with the current index.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
	KeyExclusive (see page 8484)	Specifies the upper and lower boundaries for a range.
	KeyFieldCount (see page 8484)	Specifies the number of fields to use when conducting a partial key search on a multi-field key.
R	KeySize (see page 8485)	Indicates the size of the current key used for the dataset.
	LogChanges (see page 8485)	Indicates whether to maintain a change log separate from the data.
R	Modified (see page 7029)	Indicates whether the active record is modified.

R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
	SavePoint (see page 8494)	Represents the current state of edits when logging changes.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	StatusFilter (see page 8495)	Filters the client dataset so that it exposes only those records with a specific update status.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	XMLData (see page 8496)	Represents the data of the client dataset in XML.

Published Events

	Name	Description
	AfterApplyUpdates (see page 8332)	Occurs after the client dataset applies updates to the provider but before it attempts to reconcile update errors.
	AfterCancel (see page 8333)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 8335)	Occurs after an application closes a dataset.
	AfterDelete (see page 8336)	Occurs after an application deletes a record.
	AfterEdit (see page 8339)	Occurs after an application starts editing a record.
	AfterExecute (see page 8340)	Occurs after the client dataset receives the results from an execute command.
	AfterGetParams (see page 8340)	Occurs after the client dataset receives parameter values from the provider.
	AfterGetRecords (see page 8341)	Occurs after the client dataset fetches a data packet from the provider.
	AfterInsert (see page 8342)	Occurs after an application inserts a new record.
	AfterOpen (see page 8342)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 8344)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 8344)	Occurs after an application refreshes the data in the dataset.
	AfterRowRequest (see page 8345)	Occurs after the client dataset fetches new information about the current record.
	AfterScroll (see page 8346)	Occurs after an application scrolls from one record to another.
	BeforeApplyUpdates (see page 8346)	Occurs before the client dataset applies updates to the remote server.
	BeforeCancel (see page 8347)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 8347)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 8348)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 8348)	Occurs before an application enters edit mode for the active record.
	BeforeExecute (see page 8349)	Occurs before the client dataset executes the provider's query or stored procedure.
	BeforeGetParams (see page 8350)	Occurs before the client dataset fetches parameter values from the provider.
	BeforeGetRecords (see page 8350)	Occurs before the client dataset fetches a data packet from the provider.
	BeforeInsert (see page 8352)	Occurs before an application enters insert mode.
	BeforeOpen (see page 8353)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 8354)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 8354)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeRowRequest (see page 8355)	Occurs before the client dataset fetches new information about the current record.
	BeforeScroll (see page 8356)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 8356)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 8357)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 8357)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 8358)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.

	OnNewRecord (see page 8359)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 8359)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnReconcileError (see page 8359)	Occurs when a client dataset needs to reconcile an update to a record that could not be applied.

Published Properties

	Name	Description
	Active (see page 8305)	Specifies whether the client dataset contains data.
	Aggregates (see page 8308)	Lists all aggregates that apply to the client dataset.
	AggregatesActive (see page 8308)	Specifies whether the client dataset calculates and maintains aggregate values.
	AutoCalcFields (see page 8309)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	CommandText (see page 8310)	Specifies an SQL command to be executed by the database server.
	ConnectionBroker (see page 8311)	Specifies a connection broker that handles the connection to an application server.
	Constraints (see page 8311)	Specifies record-level constraints that must be met when editing the data.
	DataSetField (see page 8312)	Indicates the TDataSetField component that represents this client dataset.
	DisableStringTrim (see page 8312)	Specifies whether trailing spaces are trimmed from field values when records are posted.
	FetchOnDemand (see page 8313)	Indicates whether to fetch data packets from the provider on an as-needed basis.
	FieldDefs (see page 8314)	Points to the list of field definitions for the dataset.
	FileName (see page 8317)	Specifies a file that stores the data of the client dataset.
	Filter (see page 8317)	Specifies the text of the current filter.
	Filtered (see page 8319)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 8320)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
	IndexDefs (see page 8320)	Contains information about the indexes for a client dataset.
	IndexFieldNames (see page 8322)	Lists the fields to use as an index.
	IndexName (see page 8322)	Identifies an index for the client dataset.
	MasterFields (see page 8323)	Names one or more fields in a master table to link with corresponding fields in this dataset in order to establish a master-detail relationship.
	MasterSource (see page 8324)	Specifies a data source component for the master dataset when establishing a detail-master relationship between this dataset and another one.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectView (see page 8324)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	PacketRecords (see page 8325)	Indicates the number or type of records in a single data packet.
	Params (see page 8326)	Contains parameter values that are sent to the provider.
	ProviderName (see page 8329)	Specifies the name of a provider component that supplies data to and applies updates from the client dataset..
	Ranged (see page 8329)	This is Ranged, a member of class TClientDataSet.
	ReadOnly (see page 8330)	Specifies whether the client dataset is read-only for this application.
	RemoteServer (see page 8330)	Specifies the connection component that this client dataset uses to connect to an application server.
	StoreDefs (see page 8331)	Indicates whether field and index definitions are saved with the client dataset.
	Tag (see page 2808)	Stores an integer value as part of a component.

45.1.6.2 TClientDataSet Properties

The properties of the TClientDataSet class are listed here.

Published Properties

	Name	Description
	Active (see page 8305)	Specifies whether the client dataset contains data.
	Aggregates (see page 8308)	Lists all aggregates that apply to the client dataset.
	AggregatesActive (see page 8308)	Specifies whether the client dataset calculates and maintains aggregate values.

	AutoCalcFields (see page 8309)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	CommandText (see page 8310)	Specifies an SQL command to be executed by the database server.
	ConnectionBroker (see page 8311)	Specifies a connection broker that handles the connection to an application server.
	Constraints (see page 8311)	Specifies record-level constraints that must be met when editing the data.
	DataSetField (see page 8312)	Indicates the TDataSetField component that represents this client dataset.
	DisableStringTrim (see page 8312)	Specifies whether trailing spaces are trimmed from field values when records are posted.
	FetchOnDemand (see page 8313)	Indicates whether to fetch data packets from the provider on an as-needed basis.
	FieldDefs (see page 8314)	Points to the list of field definitions for the dataset.
	FileName (see page 8317)	Specifies a file that stores the data of the client dataset.
	Filter (see page 8317)	Specifies the text of the current filter.
	Filtered (see page 8319)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 8320)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
	IndexDefs (see page 8320)	Contains information about the indexes for a client dataset.
	IndexFieldNames (see page 8322)	Lists the fields to use as an index.
	IndexName (see page 8322)	Identifies an index for the client dataset.
	MasterFields (see page 8323)	Names one or more fields in a master table to link with corresponding fields in this dataset in order to establish a master-detail relationship.
	MasterSource (see page 8324)	Specifies a data source component for the master dataset when establishing a detail-master relationship between this dataset and another one.
	ObjectView (see page 8324)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	PacketRecords (see page 8325)	Indicates the number or type of records in a single data packet.
	Params (see page 8326)	Contains parameter values that are sent to the provider.
	ProviderName (see page 8329)	Specifies the name of a provider component that supplies data to and applies updates from the client dataset..
	Ranged (see page 8329)	This is Ranged, a member of class TClientDataSet.
	ReadOnly (see page 8330)	Specifies whether the client dataset is read-only for this application.
	RemoteServer (see page 8330)	Specifies the connection component that this client dataset uses to connect to an application server.
	StoreDefs (see page 8331)	Indicates whether field and index definitions are saved with the client dataset.

45.1.6.2.1 TClientDataSet.Active Property

Specifies whether the client dataset contains data.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Use Active to determine whether the client dataset makes its data available. When Active is false, the client dataset is closed; It can't manipulate the data it represents. When Active is true, data can be read or edited using the client dataset.

Setting Active to true:

1Fills the client dataset with data. Depending on the other properties of the client dataset, this data comes from:

The file specified by the FileName property.

The provider specified by the ProviderName property or SetProvider method. (For some TCustomClientDataSet descendants, this is an internal provider component)

The data that was active when the client dataset was deactivated (only if the application has run continuously since the dataset

was deactivated).

2Triggers a BeforeOpen event.

3Sets the State property to dsBrowse.

4Opens a cursor into the dataset.

5Triggers an After Open event.

If the client dataset has never been active previously, setting Active to true defines the structure (metadata) of the client dataset. This structure is retrieved with the data packet from the provider or stored on disk, or, if there is no data packet, it is built using the current value of the FieldDefs property or from the persistent field components listed by the Fields property.

If an error occurs during the dataset open, the dataset state is set to dsInactive, and the cursor is closed.

Setting Active to false:

1Triggers a BeforeClose event.

2Sets the State property to dsInactive.

3Closes the cursor, saving the current data to disk if the FileName property is set, and saving the current data packet to a cache so that it can be restored when the client dataset is reopened.

4Triggers an AfterClose event.

Note: Calling the Open method sets Active to true; calling the Close method sets Active to false.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
value of the first field for the last record to the server so
that the server can return the correct records. The complete
project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
var
  LastValue: OleVariant;
  CDSClone: TClientDataSet;
begin
  if ClientDataSet1.Active then
  begin
    CDSClone := TClientDataSet.Create(Form1);
    try
      CDSClone.CloneCursor(ClientDataSet1, True);
      { turn off FetchOnDemand so that the clone only fetches
        the last LOCAL record }
      CDSClone.FetchOnDemand := False;
      CDSClone.Last;
      LastValue := CDSClone.Fields[0].AsString;
      CDSClone.Close;
    finally
      CDSClone.Free;
    end;
  end else
```

```
    LastValue := NULL;
  OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
  ClientDataSet1.GetNextPacket;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
  RemoteServer1.Connected := False;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].DisableControls;
  end;
end;

procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
```

See Also

FileName (see page 8475)

Open (see page 6976)

Close (see page 6920)

State (see page 7036)

AfterOpen (see page 7045)

BeforeOpen (see page 7051)

FieldDefs (see page 7019)

Fields (see page 7023)

ProviderName (see page 8492)

45.1.6.2.2 TClientDataSet.Aggregates Property

Lists all aggregates that apply to the client dataset.

Pascal

```
property Aggregates: TAggregates;
```

C++

```
__property TAggregates Aggregates;
```

Description

Use Aggregates to define aggregates that summarize the data in the client dataset. Aggregates is a collection of TAggregate objects, each of which defines a formula for calculating an aggregate value from a group of records in the client dataset. The individual aggregates can summarize all the records in the client dataset or subgroups of records that have the same value on a set of fields. Aggregates that summarize a subgroup of records are associated with indexes, and can only be used when the associated index is current.

Do not use Aggregates to obtain summarized values maintained by the client dataset. Aggregates can contain TAggregate objects that are not currently active. Instead, use the ActiveAggs property, which represents only those TAggregate objects that are currently active.

Note: You do not need to add a TAggregate object to the Aggregates collection when you add an aggregate field using the Fields editor. The Fields editor automatically adds TAggregate objects to Aggregates for every aggregate field you define.

See Also

ActiveAggs (see page 8460)

TAggregate (see page 8268)

IndexName (see page 8483)

TAggregateField (see page 6743)

Obtaining Aggregate Values

Specifying Aggregates

Defining an Aggregate Field

45.1.6.2.3 TClientDataSet.AggregatesActive Property

Specifies whether the client dataset calculates and maintains aggregate values.

Pascal

```
property AggregatesActive: Boolean;
```

C++

```
__property Boolean AggregatesActive;
```

Description

Use AggregatesActive to indicate whether the client dataset should maintain aggregate values. When AggregatesActive is false (the default), the client dataset does not support maintained aggregates. This allows the client dataset to avoid the overhead of calculating values that the application is not using. When AggregatesActive is true, the client dataset calculates and maintains all aggregate values specified by the Aggregates property that are compatible with the current index.

When aggregates are maintained, the Value method of every active aggregate object returns a value that reflects the current data in the client dataset. When users edit the data in the dataset, these values are recalculated to reflect the user's changes.

Note: To selectively enable and disable aggregates rather than turning them all on or off at once, use the Active property of individual TAggregate objects. These objects are available through the Aggregates property.

See Also

ActiveAggs (see page 8460)

Aggregates (see page 8461)

IndexName (see page 8483)

Value (see page 8273)

Active (see page 8274)

45.1.6.2.4 TClientDataSet.AutoCalcFields Property

Determines when the OnCalcFields event is triggered and when lookup field values are calculated.

Pascal

```
property AutoCalcFields: Boolean;
```

C++

```
__property Boolean AutoCalcFields;
```

Description

Set AutoCalcFields to control when the OnCalcFields event is triggered to update calculated fields and when lookup fields are calculated.

A calculated field is one that derives its value from the values of one or more fields in the active record, sometimes with additional processing. Lookup fields are fields whose values come from a secondary dataset or lookup cache.

Note: Unidirectional datasets support calculated fields but not unidirectional fields. For unidirectional datasets, OnCalcFields only controls when calculated fields are updated.

When AutoCalcFields is true (the default), Lookup fields are recalculated and OnCalcFields is triggered when:

The dataset is opened.

The dataset is put into dsEdit state.

Focus moves from one visual control to another, or from one column to another in a data-aware grid and modifications have been made to the record.

When AutoCalcFields is false, Lookup fields are recalculated and the OnCalcFields event occurs only when

The dataset is opened.

The dataset is put into dsEdit state.

A record is retrieved from a database.

If an application permits users to change data, OnCalcFields is frequently triggered. In these cases an application may set AutoCalcFields to false to reduce the frequency with which the OnCalcFields event occurs and with which lookup values are fetched.

See Also

OnCalcFields (see page 7053)

Calculating Fields

45.1.6.2.5 TClientDataSet.CommandText Property

Specifies an SQL command to be executed by the database server.

Pascal

```
property CommandText: WideString;
```

C++

```
__property BSTR CommandText;
```

Description

CommandText specifies what data the client dataset wants to receive from its (internal or external) provider. It is either

An SQL statement (query) for the database server to execute.

The name of a table or stored procedure.

If the client dataset uses an internal provider, CommandText is required. On TSQLClientDataSet, the CommandType property indicates whether it represents a query, table name, or stored procedure name.

If the client dataset uses an external provider component, CommandText represents a replacement for the property that the source dataset uses to identify its data (the SQL of a query, or the name of a table or stored procedure). If the provider's Options property does not include poAllowCommandText (which, by default, it does not), then CommandText has no effect.

When CommandText is a non-empty string, its value is sent to the provider when the client dataset opens or when you call the Execute method. The specified SQL command, table, or stored procedure overrides the current value of the provider's associated dataset.

If the SQL statement takes parameters, be sure that they are in the correct order, as the provider performs parameter binding on CommandText statements by index only.

Delphi Examples:

```
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  // Memo1 must contain a "select * from xxx" employee, customer, client
  // Look in Program Files/Common Files/CodeGear Shared/Data for db files
  // Notice, using the CommandText, don't need a BeforeGetRecords event
  // handler for the ClientDataSet or the DataSetProvider.

  ClientDataSet1.CommandText := Memo1.Lines.Text;
  ClientDataSet1.Open;
end;
```

See Also

AS_Execute (see page 16453)

AS_GetRecords (see page 16455)

Options (see page 18440)

Execute (see page 8413)

Open (see page 6976)

DataSet (see page 18529)

ProviderName (see page 8492)

CommandType

Overriding the Dataset On the Application Server

Overview of Using Cached Updates

Setting Up a Simple Dataset

45.1.6.2.6 TClientDataSet.ConnectionBroker Property

Specifies a connection broker that handles the connection to an application server.

Pascal

```
property ConnectionBroker: TConnectionBroker;
```

C++

```
__property TConnectionBroker ConnectionBroker;
```

Description

Use ConnectionBroker to let a connection broker handle the connection to the server that supplies the client dataset with data. The connection broker specified by ConnectionBroker simply sets the RemoteServer property to the value of its Connection property.

ConnectionBroker adds an extra layer of indirection to the specification of a connection component. This can make it easier to change the component you use to connect to an application server. If you have several client datasets in an application that all use the same connection broker, then you can change the component that connects them all to the application server by changing a single property (the Connection property of the connection broker) rather than having to change the RemoteServer property for every client dataset in the application.

See Also

RemoteServer (see page 8493)

Connection (see page 8368)

Specifying a Provider

Specifying the Provider using TLocalConnection or TConnectionBroker

45.1.6.2.7 TClientDataSet.Constraints Property

Specifies record-level constraints that must be met when editing the data.

Pascal

```
property Constraints: TCheckConstraints;
```

C++

```
__property TCheckConstraints Constraints;
```

Description

Use Constraints to read or add record-level constraints to the dataset. Record-level constraints usually impose relationships between the fields in a single record. Constraints that limit the value of a single field should be added to the individual field components.

See Also

TCheckConstraint (see page 6841)

CustomConstraint (see page 7156)

ImportedConstraint (see page 7167)

Constraining Data Values

45.1.6.2.8 TClientDataSet.DataSetField Property

Indicates the TDataSetField component that represents this client dataset.

Pascal

```
property DataSetField: TDataSetField;
```

C++

```
__property TDataSetField DataSetField;
```

Description

Use DataSetField when the client dataset is the nested detail set of another (master) client dataset. Read DataSetField to access the data set field component in the master dataset that comprises this client dataset. Set DataSetField to a field component in the master dataset to assign this client dataset as the value of that field component.

When setting DataSetField, the ProviderName, RemoteServer, and FileName properties of the client dataset are cleared. This is because the client dataset's data is now maintained by the master table in which it is nested.

Note: The master dataset that includes the TDataSetField component must be another client dataset. Otherwise, setting DataSetField raises an exception. Further, setting DataSetField to a TDataSetField that is contained in the same client dataset leads to an exception (A dataset can't be nested in itself).

See Also

NestedDataSet (see page 7070)

ProviderName (see page 8492)

RemoteServer (see page 8493)

FileName (see page 8475)

45.1.6.2.9 TClientDataSet.DisableStringTrim Property

Specifies whether trailing spaces are trimmed from field values when records are posted.

Pascal

```
property DisableStringTrim: Boolean;
```

C++

```
__property Boolean DisableStringTrim;
```

Description

When DisableStringTrim is false, trailing spaces are trimmed from field values when records are posted. When DisableStringTrim is true, field values are posted exactly as typed, including any trailing spaces.

45.1.6.2.10 TClientDataSet.FetchOnDemand Property

Indicates whether to fetch data packets from the provider on an as-needed basis.

Pascal

```
property FetchOnDemand: Boolean;
```

C++

```
__property Boolean FetchOnDemand;
```

Description

By default FetchOnDemand is true, meaning that the client dataset fetches additional packets of data as needed (for example, as a user scrolls through data, or conducts a search). If the provider does not automatically include BLOB fields or nested detail sets in data packets, setting FetchOnDemand to true causes the client dataset to fetch these values when needed as well.

Set FetchOnDemand to false only if your application contains code that explicitly fetches data packets as needed using the GetNextPacket method. Similarly, if FetchOnDemand is false and the provider does not include BLOB data by default (Options includes poFetchBlobsOnDemand), the application must fetch BLOB fields explicitly using the FetchBlobs method. If the provider does not include nested detail sets (Options includes poFetchDetailsOnDemand) and FetchOnDemand is false, the application must fetch nested details explicitly using the FetchDetails method.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
value of the first field for the last record to the server so
that the server can return the correct records. The complete
project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
var
  LastValue: OleVariant;
  CDSClone: TClientDataSet;
begin
  if ClientDataSet1.Active then
  begin
    CDSClone := TClientDataSet.Create(Form1);
    try
      CDSClone.CloneCursor(ClientDataSet1, True);
      { turn off FetchOnDemand so that the clone only fetches
        the last LOCAL record }
```

```
      CDSClone.FetchOnDemand := False;
      CDSClone.Last;
      LastValue := CDSClone.Fields[0].AsString;
      CDSClone.Close;
    finally
      CDSClone.Free;
    end;
  end else
    LastValue := NULL;
  OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
  ClientDataSet1.GetNextPacket;
end;
```

See Also

GetNextPacket (see page 8423)

Options (see page 18440)

FetchBlobs (see page 8414)

FetchDetails (see page 8414)

Requesting Data from the Source Dataset or Document

45.1.6.2.11 TClientDataSet.FieldDefs Property

Points to the list of field definitions for the dataset.

Pascal

```
property FieldDefs: TFieldDefs;
```

C++

```
__property TFieldDefs FieldDefs;
```

Description

FieldDefs lists the field definitions for a dataset. While an application can examine FieldDefs to explore the field definitions for a dataset, it should not change these definitions unless creating a new table with CreateTable or CreateDataSet.

To access fields and field values in a dataset, use the Fields, AggFields, and FieldValues properties, and the FieldsByName method.

Note: If the dataset includes object field descendants, FieldDefs represents a hierarchical view of the data, meaning that the definitions include object field definitions. To determine the definitions in a flattened view, use FieldDefList instead.

C++ Examples:

```
/*
The following code creates and activates a client dataset in
the form's OnCreate event handler.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TFieldDefs *pDefs = CDS2->FieldDefs;
  TFieldDef *pDef = pDefs->AddFieldDef();
  pDef->DataType = ftInteger;
  pDef->Name = "Field1";
```

```
  pDef = pDefs->AddFieldDef();
  pDef->DataType = ftString;
  pDef->Size = 10;
  pDef->Name = "Field2";

  TIndexDef *pIDef = CDS2->IndexDefs->AddIndexDef();
  pIDef->Fields = "Field1";
  pIDef->Name = "IntIndex";

  CDS2->CreateDataSet();
}
```

Delphi Examples:

```
{
The following code creates and activates a client dataset in
the form’s OnCreate event handler.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  with CDS2 do
  begin
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftInteger;
      Name := 'Field1';
    end;
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftString;
      Size := 10;
      Name := 'Field2';
    end;
    with IndexDefs.AddIndexDef do
    begin
      Fields := 'Field1';
      Name := 'IntIndex';
    end;
    CreateDataSet;
  end;
end;
{ The following example shows how to create a table. }
procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Close;
  Table1.DeleteTable;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
    else
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
```

```
        { it a name }
        { Next, describe the fields in the table }
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.Active:= True;
  end;
```

See Also

CreateTable (see page 10189)

Fields (see page 7023)

FieldValues (see page 7025)

ClearFields (see page 6919)

GetFieldList (see page 6953)

GetFieldNames (see page 6954)

FieldByName (see page 6936)

FindField (see page 6938)

SetFields (see page 6997)

ObjectView (see page 7031)

AggFields (see page 7006)

CreateDataSet (see page 8401)

Creating and Deleting Tables

45.1.6.2.12 TClientDataSet.FileName Property

Specifies a file that stores the data of the client dataset.

Pascal

```
property FileName: string;
```

C++

```
__property AnsiString FileName;
```

Description

Set FileName if the client dataset always reads its data from and writes its data to a single file. When FileName represents a valid file name, the client dataset automatically writes its data to that file when it is closed. When FileName specifies the name of an existing file, the client dataset automatically reads its data from that file when it is opened.

Changes are not merged into the data when it is saved to the file. This allows an application that employs the "briefcase" model to set FileName without losing updates that are destined for a database server. Single-tiered applications should explicitly merge changes before closing the dataset and saving the data to disk.

Note: When client datasets write their data to a file, they overwrite the contents of the file. Any previous contents of the file are lost.

To read from or write to a different file, use the LoadFromFile and SaveToFile methods instead. These methods can also be used to load and save data at times other than when the Active property changes.

See Also

Open (see page 6976)

Close (see page 6920)

Active (see page 8458)

LoadFromFile (see page 8433)

SaveToFile (see page 8443)

Loading Data from a File or Stream

Saving Data to a File or Stream

Using a Dedicated File on Disk

45.1.6.2.13 TClientDataSet.Filter Property

Specifies the text of the current filter.

Pascal

```
property Filter: string;
```

C++

```
__property AnsiString Filter;
```

Description

Use Filter to specify a dataset filter. When filtering is applied to a client dataset, only those records that meet a filter's conditions are available to an application. Filter contains the string that describes the filter condition. Client datasets support the following set of filter operations: See the Limiting what records appear link at the bottom of this topic for a complete list of available filter operations.

Operator or function ComparisonsExampleComment

Operator or function Comparisons	Example	Comment
=	State = 'CA'	
<>	State <> 'CA'	
>=	DateEntered >= '1/1/1998'	
<=	Total <= 100,000	
>	Percentile > 50	
<	Field1 < Field2	
BLANK	State <> 'CA' or State = BLANK	Blank records do not appear unless explicitly included in the filter.
IS NULL	Field1 IS NULL	
IS NOT NULL	Field1 IS NOT NULL	
Logical operators		
and	State = 'CA' and Country = 'US'	
or	State = 'CA' or State = 'MA'	
not	not (State = 'CA')	
Arithmetic operators		
+	Total + 5 > 100	Applies to numbers, strings, or date (time) + number
-	Field1 - 7 <> 10	Applies to numbers, dates, or date (time) - number
*	Discount * 100 > 20	Applies to numbers only
/	Discount > Total / 5	Applies to numbers only
String functions		
Upper	Upper(Field1) = 'ALWAYS'	
Lower	Lower(Field1 + Field2) = 'josp'	
Substring	Substring(DateString,8) = '1998' | Substring(DateString,1,3) = 'JAN'	Value goes from position of second argument to end or number of chars in third arg. First char has position 1.
Trim	Trim(Field1 + Field2) | Trim(Field1, '-')	removes third argument from front and back. If no third argument, trims spaces.
TrimLeft	TrimLeft(StringField) | TrimLeft(Field1, '$') <> ''	See Trim
TrimRight	TrimRight(StringField) | TrimRight(Field1, '.') <> ''	See Trim
DateTime functions		

Year	Year(DateField) = 2000	
Month	Month(DateField) <> 12	
Day	Day(DateField) = 1	
Hour	Hour(DateField) < 16	
Minute	Minute(DateField) = 0	
Second	Second(DateField) = 30	
GetDate	GetDate - DateField > 7	Represents current date and time
Date	DateField = Date(GetDate)	Returns the date portion of a datetime value
Time	TimeField > Time(GetDate)	Returns the time portion of a datetime value
Miscellaneous		
Like	Memo LIKE '%filters%'	Works like SQL-92 without the ESC clause. When applied to BLOB fields, FilterOptions determines whether case is considered.
In	Day(DateField) in (1,7)	Works like SQL-92. Second argument is a list of values all with the same type.
*	State = 'M*'	Wildcard for partial comparisons

See Also

Filtered (see page 7027)

FilterOptions (see page 7027)

Limiting Which Records Appear

FindFirst (see page 6939)

FindLast (see page 6940)

FindNext (see page 6940)

FindPrior (see page 6941)

OnFilterRecord (see page 7055)

Filter (see page 7026)

45.1.6.2.14 TClientDataSet.Filtered Property

Specifies whether or not filtering is active for a dataset.

Pascal

```
property Filtered: Boolean;
```

C++

```
__property Boolean Filtered;
```

Description

Check Filtered to determine whether or not dataset filtering is in effect. If Filtered is true, then filtering is active. To apply filter conditions specified in the Filter property or the OnFilterRecord event handler, set Filtered to true. Because unidirectional datasets do not support filters, when you set Filtered to true on a unidirectional dataset, it raises an exception.

Note: When filtering is enabled, user edits to a record may mean that the record no longer meets a filter's test condition. The

next time an attempt is made to the read the record from the dataset while the filter is in effect, the record is not visible. If that happens, the next record that passes the filter condition becomes the active record.

See Also

Filter (see page 7026)

FilterOptions (see page 7027)

OnFilterRecord (see page 7055)

Locate (see page 6970)

Lookup (see page 6970)

45.1.6.2.15 TClientDataSet.FilterOptions Property

Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.

Pascal

```
property FilterOptions: TFilterOptions;
```

C++

```
__property TFilterOptions FilterOptions;
```

Description

Set FilterOptions to specify whether or not filtering is case insensitive when filtering on string or character fields, and whether or not partial comparisons for matching filter conditions is allowed.

Note: Because unidirectional datasets do not support filters, when you set the FilterOptions property of a unidirectional dataset, it raises an exception.

By default, FilterOptions is set to an empty set. For filters based on string fields, set FilterOptions to foCaseInsensitive to catch all variations on a string regardless of capitalization.

When a string in a filter ends with an asterisk (*), it can be used to match partial strings. To disable matching of partial strings and to treat the asterisk as a literal character in string comparisons, set FilterOptions to include foNoPartialCompare.

See Also

Filter (see page 7026)

Filtered (see page 7027)

OnFilterRecord (see page 7055)

Locate (see page 6970)

Lookup (see page 6970)

45.1.6.2.16 TClientDataSet.IndexDefs Property

Contains information about the indexes for a client dataset.

Pascal

```
property IndexDefs: TIndexDefs;
```

C++

```
__property TIndexDefs IndexDefs;
```

Description

Examine IndexDefs for index information. IndexDefs maintains an array of TIndexDef items, each of which describes an available index for the dataset. Set the value of IndexDefs before calling CreateDataSet to create a set of indexes with the client dataset.

Note: The index items in IndexDefs may not always reflect the current indexes available for a dataset. Before examining IndexDefs, call the IndexDefs.Update (Delphi) or IndexDefs->Update (C++) method to refresh the item list.

C++ Examples:

```
/*
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
*/
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2->Active = False;
  // Get the current available indices
  CDS2->IndexDefs->Update();
  // Find a field named "Field2"
  for (int I = 0; I < CDS2->IndexDefs->Count; I++)
    if (CDS2->IndexDefs->Items[I]->Fields == "Field2")
    {
      // set that index as the current index for the dataset}
      CDS2->IndexName = CDS2->IndexDefs->Items[I]->Name;
    }
  CDS2->Active = True;
}
```

Delphi Examples:

```
{
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
}
procedure TForm1.Button3Click(Sender: TObject);
var
  I : Integer;
begin
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2.Active := False;
  { Get the current available indices }
  CDS2.IndexDefs.Update;
  { Find a field named 'Field2' }
  for I := 0 to CDS2.IndexDefs.Count - 1 do
    if CDS2.IndexDefs.Items[I].Fields = 'Field2' then
    begin
      { set that index as the current index for the dataset}
      CDS2.IndexName := CDS2.IndexDefs.Items[I].Name;
    end;
  CDS2.Active := True;
end;
```

See Also

IndexFieldNames (see page 8482)

TIndexDef (see page 7283)

CreateDataSet (see page 8401)

Adding a New Index

45.1.6.2.17 TClientDataSet.IndexFieldNames Property

Lists the fields to use as an index.

Pascal

```
property IndexFieldNames: Widestring;
```

C++

```
__property BSTR IndexFieldNames;
```

Description

Use IndexFieldNames as an alternative method of specifying the index to use for a client dataset. Specify the name of each field on which to index the dataset, separating names with semicolons. Ordering of field names is significant.

Note: Indexes added using IndexFieldNames do not support grouping or maintained aggregates.

Tip: Use IndexFieldNames to create sort orders on the fly at runtime.

Note: The IndexFieldNames and IndexName properties are mutually exclusive. Setting one clears the other.

See Also

IndexDefs (see page 8480)

IndexName (see page 8483)

Creating an Index with IndexFieldNames

Adding a New Index

45.1.6.2.18 TClientDataSet.IndexName Property

Identifies an index for the client dataset.

Pascal

```
property IndexName: Widestring;
```

C++

```
__property BSTR IndexName;
```

Description

Use IndexName to specify an alternative index for a client dataset.If IndexName is empty, the dataset's sort order is based on the IndexFieldNames property or on its default ordering in the data packet. Default ordering is determined by the predefined index, DEFAULT_ORDER.

If IndexName contains a valid index name, then that index is used to determine sort order of records.

Note: IndexFieldNames and IndexName are mutually exclusive. Setting one clears the other.

C++ Examples:

```
/*
This example uses the IndexName property to sort the records
```

```
in a client dataset on the Field2 field.
*/
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2->Active = False;
  // Get the current available indices
  CDS2->IndexDefs->Update();
  // Find a field named "Field2"
  for (int I = 0; I < CDS2->IndexDefs->Count; I++)
    if (CDS2->IndexDefs->Items[I]->Fields == "Field2")
    {
      // set that index as the current index for the dataset}
      CDS2->IndexName = CDS2->IndexDefs->Items[I]->Name;
    }
  CDS2->Active = True;
}
```

Delphi Examples:

```
{
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
}
procedure TForm1.Button3Click(Sender: TObject);
var
  I : Integer;
begin
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2.Active := False;
  { Get the current available indices }
  CDS2.IndexDefs.Update;
  { Find a field named 'Field2' }
  for I := 0 to CDS2.IndexDefs.Count - 1 do
    if CDS2.IndexDefs.Items[I].Fields = 'Field2' then
    begin
      { set that index as the current index for the dataset}
      CDS2.IndexName := CDS2.IndexDefs.Items[I].Name;
    end;
  CDS2.Active := True;
end;
```

See Also

IndexFieldNames (see page 8482)

Deleting and Switching Indexes

45.1.6.2.19 TClientDataSet.MasterFields Property

Names one or more fields in a master table to link with corresponding fields in this dataset in order to establish a master-detail relationship.

Pascal

```
property MasterFields: Widestring;
```

C++

```
__property BSTR MasterFields;
```

Description

Use MasterFields after setting the MasterSource property to specify the names of one or more fields to use in establishing a detail-master relationship between this dataset and the one specified in MasterSource.

MasterFields is a string containing one or more field names in the master table. Separate field names with semicolons.

Each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this dataset for display.

Note: At design time, use the Field Link designer to establish the master-detail relationship between two datasets.

See Also

MasterSource (see page 8486)

Making the Table a Detail of Another Dataset

45.1.6.2.20 TClientDataSet.MasterSource Property

Specifies a data source component for the master dataset when establishing a detail-master relationship between this dataset and another one.

Pascal

```
property MasterSource: TDataSource;
```

C++

```
__property TDataSource MasterSource;
```

Description

Use MasterSource to specify the name of the data source component whose DataSet property identifies a dataset to use as a master table in establishing a detail-master relationship between this dataset and another one.

At design time choose an available data source from the MasterSource property's drop-down list in the Object Inspector.

After setting the MasterSource property, specify which fields to use in the master table by setting the MasterFields property. At runtime each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this dataset for display.

Note: At design time, use the Field Link designer to establish the master-detail relationship between two datasets.

Tip: MasterSource establishes a master/detail relationship using the traditional linked cursor approach. Alternately, client datasets can participate in master/detail relationships using nested datasets, where the detail dataset is stored with the master table's data. To assign a client dataset's data as the value of a nested detail, use the DataSetField property.

See Also

MasterFields (see page 8486)

Making the Table a Detail of Another Dataset

45.1.6.2.21 TClientDataSet.ObjectView Property

Specifies whether fields are stored hierarchically or flattened out in the Fields property.

Pascal

```
property ObjectView: Boolean;
```

C++

```
__property Boolean ObjectView;
```

Description

ObjectView affects the way the Fields property stores object fields and the way data-aware grids display ADT and array fields.

When ObjectView is true, the fields are stored hierarchically in the Fields property, meaning any child fields of an object field are referenced by the object field and don't appear sequentially after the object field in the TFields:: .Fields array. When ObjectView is false, the fields are stored sequentially in the Fields property, meaning any child fields of an object field are stored after the object field as siblings in the Fields array.

When ObjectView is false, object field types, such as TADTField, are not created. This switch is provided for increased compatibility with older data-aware controls, which may not handle object field types properly. The default is false for TBDEDataSet, and true for TClientDataSet.

Note: If the ObjectView property of the dataset provided to a TClientDataSet is false, then the TClientDataSet's ObjectView property always acts as false, since the data received is already flattened.

See Also

FieldDefList (see page 7019)

Fields (see page 7023)

Displaying ADT and Array Fields

Using Nested Detail Tables

Using Object Fields

Working with ADT Fields

Working with Array Fields

45.1.6.2.22 TClientDataSet.PacketRecords Property

Indicates the number or type of records in a single data packet.

Pascal

```
property PacketRecords: Integer;
```

C++

```
__property int PacketRecords;
```

Description

Use PacketRecords to specify how many records should be contained in a single packet fetched from a provider, or to indicate that the packet returned should contain only metadata.

When a client dataset is instantiated, PacketRecords is automatically set to -1, meaning that a single packet should contain all records in the dataset.

If PacketRecords is greater than zero, then it specifies the number of records to return in a packet.

To retrieve the metadata for a dataset, set PacketRecords to 0. When PacketRecords is zero, the provider returns only information from its dataset that defines the database's structure, such as table, column, constraint, and domain definitions.

Warning: When PacketRecords is greater than 0, the client dataset must fetch its data incrementally. If the provider is on a stateless application server, the client dataset must inform the provider where to start fetching data. This communication can occur using a BeforeGetRecords event handler.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
value of the first field for the last record to the server so
that the server can return the correct records. The complete
project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
var
  LastValue: OleVariant;
  CDSClone: TClientDataSet;
begin
  if ClientDataSet1.Active then
  begin
    CDSClone := TClientDataSet.Create(Form1);
    try
      CDSClone.CloneCursor(ClientDataSet1, True);
      { turn off FetchOnDemand so that the clone only fetches
        the last LOCAL record }
      CDSClone.FetchOnDemand := False;
      CDSClone.Last;
      LastValue := CDSClone.Fields[0].AsString;
      CDSClone.Close;
    finally
      CDSClone.Free;
    end;
  end else
    LastValue := NULL;
  OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
  ClientDataSet1.GetNextPacket;
end;
```

See Also

FetchOnDemand (see page 8471)

GetNextPacket (see page 8423)

BeforeGetRecords (see page 8515)

Requesting Data from the Source Dataset or Document

Using the BDE to Cache Updates

Updating Data in dbExpress Applications

45.1.6.2.23 TClientDataSet.Params Property

Contains parameter values that are sent to the provider.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Use Params to specify parameter values that the provider should pass to a source dataset. If the client dataset uses an internal provider, Params represents the values for any parameters used by the SQL statement specified by the CommandText property. If the client dataset uses an external provider that is associated with a dataset that represents a query or stored procedure, these parameter values supply input parameter values for that query or stored procedure and return any output parameters of a stored procedure. If the provider is associated with a TTable or TSQLTable component, Params limits the records that are sent in data packets. After specifying the parameters (at design time or runtime), they are sent to the provider automatically when the client dataset fetches data or executes a query or stored procedure using its Execute method.

To send parameter values to a query or stored procedure, for every parameter on the query or stored procedure for which the client dataset should send a value, add a parameter object (TParam) to Params. Assign each parameter object the same Name, DataType, and ParamType as the corresponding parameter on the query or stored procedure. These values are automatically encoded and sent to the provider when the client dataset fetches data.

To send parameter values to a TTable or TSQLTable, add a parameter object for each field in the table that is used to limit the values sent in data packets. Assign each parameter object the same Name and DataType as the corresponding field component on the table. These parameter values are automatically sent when the client dataset fetches data, and the records returned in data packets from the provider only contain records that match the assigned value on the corresponding fields. You can achieve the same affect by applying a filter to the field, but using Params is more efficient because the data need not be included in data packets.

Note: To initialize Params so that it reflects the parameters of a provider's associated query or stored procedure, call the FetchParams method.

Warning: The parameters are only applied to the provider when a query or stored procedure is run the first time the client dataset fetches records. To change parameter values and force a query or stored procedure to be rerun with new values, use the Execute method.

Delphi Examples:

```
{
This example fills in the parameters of a client dataset
from the entries of a list box.
}
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
  ListItem: string;
begin
  { Call FetchParams to ensure parameters reflect server metadata }
  ClientDataSet1.FetchParams;
  with ClientDataSet1.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
```

```
          ftWord:
            Items[I].AsWord := StrToIntDef(ListItem, 0);
          ftBoolean:
            begin
              if ListItem = 'True' then
                Items[I].AsBoolean := True
              else
                Items[I].AsBoolean := False;
            end;
          ftFloat:
            Items[I].AsFloat := StrToFloat(ListItem);
          ftCurrency:
            Items[I].AsCurrency := StrToFloat(ListItem);
          ftBCD:
            Items[I].AsBCD := StrToCurr(ListItem);
          ftDate:
            Items[I].AsDate := StrToDate(ListItem);
          ftTime:
            Items[I].AsTime := StrToTime(ListItem);
          ftDateTime:
            Items[I].AsDateTime := StrToDateTime(ListItem);
        end;
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Here is a string');
    ListBox1.Items.Add('4');
    ListBox1.Items.Add('3456789');
    ListBox1.Items.Add('34');
    ListBox1.Items.Add('0');
    ListBox1.Items.Add('12.56');
    ListBox1.Items.Add('4.23');
    ListBox1.Items.Add('99.5');
    ListBox1.Items.Add('4/12/53');
    ListBox1.Items.Add('7:45:00');
    ListBox1.Items.Add('4/12/53 7:45:00');

  end;
```

See Also

Params (see page 10087)

Params (see page 10165)

Active (see page 8458)

FetchParams (see page 8415)

Params (see page 20197)

Parameters (see page 1220)

Getting Parameters from the Application Server

Limiting Records with Parameters

Overview of Using Cached Updates

Passing Parameters to the Source Dataset

Sending Query or Stored Procedure Parameters

Setting Up a Simple Dataset

45.1.6.2.24 TClientDataSet.ProviderName Property

Specifies the name of a provider component that supplies data to and applies updates from the client dataset..

Pascal

```
property ProviderName: string;
```

C++

```
__property AnsiString ProviderName;
```

Description

Use ProviderName to specify an external provider object that links the client dataset to another (source) dataset or to an XML document. This property is passed as a parameter to IAppServer interface calls when the client fetches data from or applies updates to the source dataset, or its database server, or the source XML document.

The provider specified by ProviderName can reside in the same application as the client dataset or in an application server running on another system.

If the provider is in the same application as the client dataset, it must have the same Owner as the client dataset. To use a local provider that has a different Owner, use the SetProvider method instead.

If the provider is on a remote application server, it must be exported by that application server. If the application server contains multiple provider objects, you can set ProviderName to any of them once the client dataset establishes an initial connection to the application server through its RemoteServer property. If the connection component is a Web connection or socket connection, or if the application server is registered on the client system (using TRegsvr.exe), ProviderName can be set from a drop-down in the Object Inspector once the RemoteServer property is set.

See Also

RemoteServer (see page 8493)

TDataSetProvider (see page 18508)

IAppServer (see page 16450)

SetProvider (see page 8451)

Specifying a Provider

Connecting a Client Dataset to Another Dataset in the Same Application

Using a Multi-Tiered Architecture

Communicating with the Client Dataset

45.1.6.2.25 TClientDataSet.Ranged Property

Pascal

```
property Ranged: Boolean;
```

C++

```
__property Boolean Ranged;
```

Description

This is Ranged, a member of class TClientDataSet.

45.1.6.2.26 TClientDataSet.ReadOnly Property

Specifies whether the client dataset is read-only for this application.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use the ReadOnly property to prevent users from updating, inserting, or deleting data in the dataset. By default, ReadOnly is false, meaning users can potentially alter the dataset's data.

To guarantee that users cannot modify or add data to a dataset, set ReadOnly to true.

When ReadOnly is true, the dataset's CanModify property is false.

Note: Even if ReadOnly is false, users may not be able to modify or add data to a client dataset if it gets its data from a provider. The provider's Options property can limit whether data can be edited, and if so, what types of edits are allowed.

See Also

Active (see page 7004)

CanModify

Options (see page 18440)

45.1.6.2.27 TClientDataSet.RemoteServer Property

Specifies the connection component that this client dataset uses to connect to an application server.

Pascal

```
property RemoteServer: TCustomRemoteServer;
```

C++

```
__property TCustomRemoteServer RemoteServer;
```

Description

If the client dataset is in the client portion of a multi-tiered application, use RemoteServer to specify the connection component that connects to the application server. The connection component locates the application server and obtains its IAppServer interface, which the client dataset can use for communicating with a remote provider.

Although it is not required, you can also set RemoteServer to a TLocalConnection component when working with a provider that resides in the same application.

If a client application uses more than connection component, the application can change which connection a client dataset uses by changing the value for this property.

Tip: If you have several client datasets that all use the same connection component and you think you may change the connection component later (for example, to change the protocol), you can use the ConnectionBroker property to centralize the connection.

See Also

ProviderName (see page 8492)

TSocketConnection (see page 18928)

TDCOMConnection (see page 16008)

TWebConnection (see page 18968)

AppServer (see page 8532)

IAppServer (see page 16450)

ConnectionBroker (see page 8466)

TLocalConnection (see page 23236)

TSoapConnection (see page 19864)

Specifying a Provider

Using a Multi-Tiered Architecture

Specifying the Provider using TLocalConnection or TConnectionBroker

45.1.6.2.28 TClientDataSet.StoreDefs Property

Indicates whether field and index definitions are saved with the client dataset.

Pascal

```
property StoreDefs: Boolean;
```

C++

```
__property Boolean StoreDefs;
```

Description

If StoreDefs is true, the client dataset's index and field definitions are stored with the client dataset when the form or data module that contains it is saved. Setting StoreDefs to true makes the CreateDataSet method into a one-step procedure that creates fields and indexes at runtime.

See Also

FieldDefs (see page 7019)

IndexDefs (see page 8480)

CreateDataSet (see page 8401)

45.1.6.3 TClientDataSet Events

The events of the TClientDataSet class are listed here.

Published Events

	Name	Description
	AfterApplyUpdates (see page 8332)	Occurs after the client dataset applies updates to the provider but before it attempts to reconcile update errors.
	AfterCancel (see page 8333)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 8335)	Occurs after an application closes a dataset.
	AfterDelete (see page 8336)	Occurs after an application deletes a record.
	AfterEdit (see page 8339)	Occurs after an application starts editing a record.
	AfterExecute (see page 8340)	Occurs after the client dataset receives the results from an execute command.
	AfterGetParams (see page 8340)	Occurs after the client dataset receives parameter values from the provider.
	AfterGetRecords (see page 8341)	Occurs after the client dataset fetches a data packet from the provider.

	AfterInsert (see page 8342)	Occurs after an application inserts a new record.
	AfterOpen (see page 8342)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 8344)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 8344)	Occurs after an application refreshes the data in the dataset.
	AfterRowRequest (see page 8345)	Occurs after the client dataset fetches new information about the current record.
	AfterScroll (see page 8346)	Occurs after an application scrolls from one record to another.
	BeforeApplyUpdates (see page 8346)	Occurs before the client dataset applies updates to the remote server.
	BeforeCancel (see page 8347)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 8347)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 8348)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 8348)	Occurs before an application enters edit mode for the active record.
	BeforeExecute (see page 8349)	Occurs before the client dataset executes the provider's query or stored procedure.
	BeforeGetParams (see page 8350)	Occurs before the client dataset fetches parameter values from the provider.
	BeforeGetRecords (see page 8350)	Occurs before the client dataset fetches a data packet from the provider.
	BeforeInsert (see page 8352)	Occurs before an application enters insert mode.
	BeforeOpen (see page 8353)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 8354)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 8354)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeRowRequest (see page 8355)	Occurs before the client dataset fetches new information about the current record.
	BeforeScroll (see page 8356)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 8356)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 8357)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 8357)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 8358)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 8359)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 8359)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnReconcileError (see page 8359)	Occurs when a client dataset needs to reconcile an update to a record that could not be applied.

45.1.6.3.1 TClientDataSet.AfterApplyUpdates Event

Occurs after the client dataset applies updates to the provider but before it attempts to reconcile update errors.

Pascal

```
property AfterApplyUpdates: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterApplyUpdates;
```

Description

Write an AfterApplyUpdates event handler to respond to custom information returned to the client dataset from a provider's AfterApplyUpdates event handler. AfterApplyUpdates is part of the mechanism by which a client dataset and a provider communicate information about updates. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's ApplyUpdates method, the following events occur:

1.The client dataset receives a BeforeApplyUpdates event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeApplyUpdates event, where it can respond to or change that information before it applies the updates.

3.The provider applies the updates, and receives an OnUpdateError event for any problem records.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider in its AfterApplyUpdates event handler.

6.The client dataset generates an OnReconcileError for every record that could not be applied by the provider.

See Also

ApplyUpdates (see page 8394)

AS_ApplyUpdates (see page 16451)

applyupdates (see page 18459)

AfterApplyUpdates (see page 18475)

BeforeApplyUpdates (see page 8511)

OnReconcileError (see page 8524)

Applying Updates

Communicating with the Client Dataset

45.1.6.3.2 TClientDataSet.AfterCancel Event

Occurs after an application completes a request to cancel modifications to the active record.

Pascal

```
property AfterCancel: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterCancel;
```

Description

Write an AfterCancel event handler to take specific action after an application cancels changes to the active record. AfterCancel is called by the Cancel method after it updates the current position, releases the lock on the active record if necessary, and sets the dataset state to dsBrowse. If an application requires additional processing before returning control to a user after a Cancel event, code it in the AfterCancel event.

Delphi Examples:

```
{
This example updates the form's status bar with a message
when an AfterCancel event occurs.  Set the StatusBar
SimplePanel property to True.
}
procedure TForm1.Table1AfterCancel(DataSet: TDataSet);
begin
  StatusBar1.SimpleText := 'Record changes cancelled for ' + DataSet.Name;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Table1.Cancel;
end;
```

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  Table1.Post
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Name := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
    begin
        Table1.Close;
        Table1.DeleteTable;
    end;
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
```

```
      Table1.Active:= True;
      for i := 1 to 20 do
        Table1.AppendRecord([i*111, i*222]);
    end;
  end;
  DS2.DataSet:= Table1;
  DBGrid2.DataSource.DataSet:= Table1;
  Table1.AfterCancel:= Table1AfterCancel;
  Table1.Active:= True;
end;
```

See Also

BeforeCancel (see page 7048)

Cancel (see page 6916)

State (see page 7036)

45.1.6.3.3 TClientDataSet.AfterClose Event

Occurs after an application closes a dataset.

Pascal

```
property AfterClose: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterClose;
```

Description

Write an AfterClose event handler to take specific action immediately after an application closes a dataset. For example, as a security measure, an application might clear a PASSWORD entry from the Params property of a TSQLConnection component when the dataset is closed.

AfterClose is called after a dataset is closed and the dataset state is set to dsInactive.

Delphi Examples:

```
{
This example adds an entry to a memo with a message when an
AfterOpen event or an AfterClose event occurs.  Populate the
ClientDataSet events with CheckButtonActive.
}
procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  if (Button1.Enabled) then
    Memo2.Lines.Add('Now that the dataset is open, record information is available.')
  else
    Memo2.Lines.Add('Now that the dataset is closed, record information is not available.');
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
```

```
      RemoteServer1.Connected := True;
    ClientDataSet1.Close;
    with RemoteServer1 do
    begin
      for I := 0 to DataSetCount - 1 do
        DataSets[I].EnableControls;
    end;
    ClientDataSet1.Open;
  end;

  procedure TForm1.Button3Click(Sender: TObject);
  var
    I: Integer;
  begin
    ClientDataSet1.Close;
    RemoteServer1.Connected := False;
    with RemoteServer1 do
    begin
      for I := 0 to DataSetCount - 1 do
        DataSets[I].DisableControls;
    end;
  end;

  procedure TForm1.CheckButtonActive(DataSet: TDataSet);
  begin
    Button1.Enabled := not ClientDataSet1.Active;
    Button2.Enabled := ClientDataSet1.Active;
    Button3.Enabled := ClientDataSet1.Active;
  end;
```

See Also

Active (see page 7004)

State (see page 7036)

BeforeClose (see page 7048)

Close (see page 6920)

Params (see page 20253)

Opening and Closing Datasets

45.1.6.3.4 TClientDataSet.AfterDelete Event

Occurs after an application deletes a record.

Pascal

```
property AfterDelete: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterDelete;
```

Description

Write an AfterDelete event handler to take specific action immediately after an application deletes the active record in a dataset. AfterDelete is called by Delete after it deletes the record, sets the dataset state to dsBrowse, and repositions the current record.

C++ Examples:

```
/*
This example displays a message on the form's status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
```

```
*/
void __fastcall TForm1::Table1AfterDelete(TDataSet *DataSet)
{
  StatusBar1->SimpleText = Format(
    "There are now %d records in the table",
    ARRAYOFCONST(((int)DataSet->RecordCount)));
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Table1->Delete();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Table1 = new TTable(Form1);
  Table1->Active = false; // The Table component must not be active
  Table1->DatabaseName = "DBDEMOS";
  Table1->TableType = ttParadox;
  Table1->TableName = "CustInfo";
  Table1->Active = False;
  if (!Table1->Exists) // Don't overwrite an existing table
  {
    Table1->Close();
    Table1->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Table1->FieldDefs->Clear();
  TFieldDef *pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Table1->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Table1->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Table1->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Table1->CreateTable();
  Table1->Active = True;
  for (int i = 1; i <= 20; i++)
    Table1->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  DS2->DataSet = Table1;
  DBGrid2->DataSource->DataSet = Table1;
  Table1->AfterDelete = Table1AfterDelete;
  Table1->Active = True;
}
```

Delphi Examples:

```
{
This example displays a message on the form’s status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
}
procedure TForm1.Table1AfterDelete(DataSet: TDataSet);
begin
  StatusBar1.SimpleText := Format('There are now %d records in the table',
[DataSet.RecordCount]);
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Delete;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Name := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
    begin
        Table1.Close;
        Table1.DeleteTable;
    end;
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
```

```
      Table1.Active:= True;
      for i := 1 to 20 do
        Table1.AppendRecord([i*111, i*222]);
    end;
  end;
  DS2.DataSet:= Table1;
  DBGrid2.DataSource.DataSet:= Table1;
  Table1.AfterDelete:= Table1AfterDelete;
  Table1.Active:= True;
end;
```

See Also

State (see page 7036)

BeforeDelete (see page 7049)

Delete (see page 6924)

45.1.6.3.5 TClientDataSet.AfterEdit Event

Occurs after an application starts editing a record.

Pascal

```
property AfterEdit: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterEdit;
```

Description

Write an AfterEdit event handler to take specific action immediately after dataset enters edit mode. AfterEdit is called by Edit after it enables editing of a record, recalculates calculated fields, and calls the data event handler to process a record change.

Delphi Examples:

```
{
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

BeforeEdit (see page 7049)

Edit (see page 6932)

45.1.6.3.6 TClientDataSet.AfterExecute Event

Occurs after the client dataset receives the results from an execute command.

Pascal

```
property AfterExecute: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterExecute;
```

Description

Write an AfterExecute event handler to respond to custom information returned to the client dataset from a provider's AfterExecute event handler. AfterExecute is part of the mechanism by which a client dataset and a provider communicate information when executing an SQL command from the client. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's Execute method, the following events occur:

1.The client dataset receives a BeforeExecute event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeExecute event, where it can respond to or change that information before it executes the query or stored procedure.

3.The provider executes the appropriate command (either the command supplied by the CommandText property, or the command associated with a dataset such as a query or stored procedure).

4.The provider receives an AfterExecute event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeExecute event handler.

5.The client dataset unpacks any output parameters from the provider into the Params property and then receives an AfterExecute event, where it can respond to the custom information returned by the provider's AfterExecute event handler.

See Also

Execute (see page 8413)

AS_Execute (see page 16453)

Execute (see page 18466)

AfterExecute (see page 18476)

BeforeExecute (see page 8514)

CommandText (see page 8465)

Communicating with the Client Dataset

45.1.6.3.7 TClientDataSet.AfterGetParams Event

Occurs after the client dataset receives parameter values from the provider.

Pascal

```
property AfterGetParams: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterGetParams;
```

Description

Write an AfterGetParams event handler to respond to custom information returned to the client dataset from a provider's AfterGetParams event handler. AfterGetParams is part of the mechanism by which a client dataset and a provider communicate information when the client application calls FetchParams. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's FetchParams method, the following events occur:

1.The client dataset receives a BeforeGetParams event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeGetParams event, where it can respond to or change that information before it retrieves parameter values.

3.The provider encodes all of its dataset's current parameter values into an OleVariant.

4.The provider receives an AfterGetParams event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetParams event handler.

5.The client dataset unpacks the parameters from the provider into the Params property and then receives an AfterGetParams event, where it can respond to the custom information returned by the provider's AfterGetParams event handler.

See Also

FetchParams (see page 8415)

AS_GetParams (see page 16454)

GetParams (see page 18467)

aftergetparams (see page 18477)

BeforeGetParams (see page 8514)

Communicating with the Client Dataset

45.1.6.3.8 TClientDataSet.AfterGetRecords Event

Occurs after the client dataset fetches a data packet from the provider.

Pascal

```
property AfterGetRecords: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterGetRecords;
```

Description

Write an AfterGetRecords event handler to respond to custom information returned to the client dataset from a provider's AfterGetRecords event handler. AfterGetRecords is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches data from the application server, the following events occur:

1. The client dataset receives a BeforeGetRecords event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.
2. The provider receives a BeforeGetRecords event, where it can respond to or change that information before it creates a data packet.
3. The provider generates an OnGetData event where it can edit the data before creating a data packet, and then generates the

data packet.

4. The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.
5. The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider's AfterGetRecords event handler.

See Also

AS_GetRecords (see page 16455)

GetRecords (see page 18468)

AfterGetRecords (see page 18477)

BeforeGetRecords (see page 8515)

Communicating with the Client Dataset

45.1.6.3.9 TClientDataSet.AfterInsert Event

Occurs after an application inserts a new record.

Pascal

```
property AfterInsert: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterInsert;
```

Description

Write an AfterInsert event handler to take specific action immediately after an application inserts a record. The Insert and Append methods generate an AfterInsert event after inserting or appending a new record.

Delphi Examples:

```
{
This example updates a memo with a message when an
AfterInsert event or a BeforeInsert event occurs.
}
procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After inserting a record');
end;

procedure TForm1.ClientDataSet1BeforeInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before inserting a record');
end;
```

See Also

BeforeInsert (see page 7050)

Append (see page 6910)

Insert (see page 6959)

45.1.6.3.10 TClientDataSet.AfterOpen Event

Occurs after an application completes opening a dataset and before any data access occurs.

Pascal

```
property AfterOpen: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterOpen;
```

Description

Write an AfterOpen event handler to take specific action immediately after an application opens the dataset. AfterOpen is called after the dataset establishes access to its data and the dataset is put into dsBrowse state. For example, an AfterOpen event handler might check an ini file to determine the last record touched in the dataset the previous time the application ran, and position the dataset at that record.

Delphi Examples:

```
{
This example adds an entry to a memo with a message when an
AfterOpen event or an AfterClose event occurs.  Populate the
ClientDataSet events with CheckButtonActive.
}
procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  if (Button1.Enabled) then
    Memo2.Lines.Add('Now that the dataset is open, record information is available.')
  else
    Memo2.Lines.Add('Now that the dataset is closed, record information is not available.');
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
  RemoteServer1.Connected := False;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].DisableControls;
  end;
end;

procedure TForm1.CheckButtonActive(DataSet: TDataSet);
```

```
begin
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
```

See Also

Active (see page 7004)

State (see page 7036)

BeforeOpen (see page 7051)

Open (see page 6976)

Opening and Closing Datasets

45.1.6.3.11 TClientDataSet.AfterPost Event

Occurs after an application writes the active record to the database or change log and returns to browse state.

Pascal

```
property AfterPost: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterPost;
```

Description

Write an AfterPost event handler to take specific action immediately after an application posts a change to the active record. AfterPost is called after a modification or insertion is made to a record.

Delphi Examples:

```
{
This example updates the form's status bar with a message
when an AfterPost event occurs.
}
procedure TForm1.ClientDataSet1AfterPost(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before posting a record');
end;
```

See Also

BeforePost (see page 7052)

Post (see page 6977)

45.1.6.3.12 TClientDataSet.AfterRefresh Event

Occurs after an application refreshes the data in the dataset.

Pascal

```
property AfterRefresh: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterRefresh;
```

Description

Write an AfterRefresh event handler to take specific action immediately after an application has updated the records in the

dataset. AfterRefresh is generated by calls to the Refresh method.

See Also

BeforeRefresh (see page 7052)

Refresh (see page 6991)

45.1.6.3.13 TClientDataSet.AfterRowRequest Event

Occurs after the client dataset fetches new information about the current record.

Pascal

```
property AfterRowRequest: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterRowRequest;
```

Description

Write an AfterRowRequest event handler to respond to custom information returned to the client dataset from a provider's AfterRowRequest event handler. AfterRowRequest is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches detail information about the current record or refreshes the current record, the following events occur:

1.The client dataset receives a BeforeRowRequest event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeRowRequest event, where it can respond to or change that information before it looks up the requested information.

3.The provider creates a delta packet that contains the requested information.

4.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeRowRequest event handler.

5.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider's AfterRowRequest event handler.

See Also

AS_RowRequest (see page 16456)

RowRequest (see page 18472)

AfterRowRequest (see page 18478)

BeforeRowRequest (see page 8520)

FetchBlobs (see page 8414)

FetchParams (see page 8415)

RefreshRecord (see page 8442)

Communicating with the Client Dataset

45.1.6.3.14 TClientDataSet.AfterScroll Event

Occurs after an application scrolls from one record to another.

Pascal

```
property AfterScroll: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterScroll;
```

Description

Write an AfterScroll event handler to take specific action immediately after an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. AfterScroll is called after all other events triggered by these methods and any other methods that switch from record to record in the dataset.

See Also

BeforeScroll (see page 7053)

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Prior (see page 6979)

Navigating Datasets

45.1.6.3.15 TClientDataSet.BeforeApplyUpdates Event

Occurs before the client dataset applies updates to the remote server.

Pascal

```
property BeforeApplyUpdates: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeApplyUpdates;
```

Description

Write a BeforeApplyUpdates event handler to send custom information to the client dataset's provider. BeforeApplyUpdates is part of the mechanism by which a client dataset and a provider communicate information about updates. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's ApplyUpdates method, the following events occur:

1.The client dataset receives a BeforeApplyUpdates event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeApplyUpdates event, where it can respond to or change that information before it applies the updates.

3.The provider applies the updates and receives an OnUpdateError event for any problem records.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider's AfterApplyUpdates event handler.

See Also

ApplyUpdates (see page 8394)

AS_ApplyUpdates (see page 16451)

applyupdates (see page 18459)

BeforeApplyUpdates (see page 18479)

AfterApplyUpdates (see page 8497)

Applying Updates

Communicating with the Client Dataset

45.1.6.3.16 TClientDataSet.BeforeCancel Event

Occurs before an application executes a request to cancel changes to the active record.

Pascal

```
property BeforeCancel: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeCancel;
```

Description

Write a BeforeCancel event to take specific action before an application carries out a request to cancel changes. BeforeCancel is called by the Cancel method before it cancels a dataset operation such as Edit, Insert, or Delete.

An application might use the BeforeCancel event to record a user's changes in an undo buffer.

See Also

AfterCancel (see page 7038)

Delete (see page 6924)

Edit (see page 6932)

Insert (see page 6959)

45.1.6.3.17 TClientDataSet.BeforeClose Event

Occurs immediately before the dataset closes.

Pascal

```
property BeforeClose: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeClose;
```

Description

Write a BeforeClose event to take specific action before an application closes a dataset. Calling Close or setting the Active property to false results in a call to the BeforeClose event handler.

See Also

Active (see page 7004)

AfterClose (see page 7040)

Close (see page 6920)

Opening and Closing Datasets

Posting Data

45.1.6.3.18 TClientDataSet.BeforeDelete Event

Occurs before an application attempts to delete the active record.

Pascal

```
property BeforeDelete: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeDelete;
```

Description

Write a BeforeDelete event handler to take specific action before an application deletes the active record. BeforeDelete is called by Delete before it actually deletes a record.

In Delphi, making use of this event an application might, for example, display a dialog box asking for confirmation before deleting the record. On denial of confirmation, the application could abort the deletion by calling the Abort procedure.

See Also

AfterDelete (see page 7041)

Delete (see page 6924)

Abort (see page 22786)

45.1.6.3.19 TClientDataSet.BeforeEdit Event

Occurs before an application enters edit mode for the active record.

Pascal

```
property BeforeEdit: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeEdit;
```

Description

Write a BeforeEdit event handler to take specific action before an application enables editing of the active record. For example, an application that keeps a log of database edits could use the BeforeEdit event to record the edit request, time, and user before entering edit state.

Delphi Examples:

```
{
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
```

```
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

State (see page 7036)

AfterEdit (see page 7044)

Edit (see page 6932)

45.1.6.3.20 TClientDataSet.BeforeExecute Event

Occurs before the client dataset executes the provider's query or stored procedure.

Pascal

```
property BeforeExecute: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to send custom information to the client dataset's provider. BeforeExecute is part of the mechanism by which a client dataset and a provider communicate information when explicitly executing an SQL command by the provider's dataset. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's Execute method, the following events occur:

1.The client dataset receives a BeforeExecute event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeExecute event, where it can respond to or change that information before it executes the query or stored procedure.

3.The provider executes the appropriate command (either the command supplied by the CommandText property, or the command associated with a dataset such as a query or stored procedure).

4.The provider receives an AfterExecute event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeExecute event handler.

5.The client dataset unpacks any output parameters from the provider into the Params property and then receives an AfterExecute event, where it can respond to the custom information returned by the provider's AfterExecute event handler.

See Also

Execute (see page 8413)

AS_Execute (see page 16453)

Execute (see page 18466)

beforeexecute (see page 18479)

AfterExecute (see page 8504)

CommandText (see page 8465)

Communicating with the Client Dataset

45.1.6.3.21 TClientDataSet.BeforeGetParams Event

Occurs before the client dataset fetches parameter values from the provider.

Pascal

```
property BeforeGetParams: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeGetParams;
```

Description

Write a BeforeGetParams event handler to send custom information to the client dataset's provider. BeforeGetparams is part of the mechanism by which a client dataset and a provider communicate information when passing parameter values from the provider's dataset. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's FetchParams method, the following events occur:

1.The client dataset receives a BeforeGetParams event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeGetParams event, where it can respond to or change that information before it retrieves parameter values.

3.The provider encodes all of its dataset's current parameter values into an OleVariant.

4.The provider receives an AfterGetParams event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetParams event handler.

5.The client dataset unpacks the parameters from the provider into the Params property and then receives an AfterGetParams event, where it can respond to the custom information returned by the provider's AfterGetParams event handler.

See Also

FetchParams (see page 8415)

AS_GetParams (see page 16454)

GetParams (see page 18467)

BeforeGetParams (see page 18480)

AfterGetParams (see page 8505)

Communicating with the Client Dataset

45.1.6.3.22 TClientDataSet.BeforeGetRecords Event

Occurs before the client dataset fetches a data packet from the provider.

Pascal

```
property BeforeGetRecords: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeGetRecords;
```

Description

Write a BeforeGetRecords event handler to send custom information to the client dataset's provider. BeforeGetRecords is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches data from the application server, the following events occur:

1.The client dataset receives a BeforeGetRecords event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeGetRecords event, where it can respond to or change that information before it creates a data packet.

3.The provider generates the data packet.

4.The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.

5.The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider's AfterGetRecords event handler.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
value of the first field for the last record to the server so
that the server can return the correct records. The complete
project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
var
  LastValue: OleVariant;
  CDSClone: TClientDataSet;
begin
  if ClientDataSet1.Active then
  begin
    CDSClone := TClientDataSet.Create(Form1);
    try
      CDSClone.CloneCursor(ClientDataSet1, True);
      { turn off FetchOnDemand so that the clone only fetches
        the last LOCAL record }
      CDSClone.FetchOnDemand := False;
      CDSClone.Last;
      LastValue := CDSClone.Fields[0].AsString;
      CDSClone.Close;
    finally
      CDSClone.Free;
    end;
```

```
    end else
      LastValue := NULL;
    OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
    ClientDataSet1.GetNextPacket;
  end;
```

See Also

AS_GetRecords (see page 16455)

GetRecords (see page 18468)

BeforeGetRecords (see page 18480)

AfterGetRecords (see page 8506)

Communicating with the Client Dataset

45.1.6.3.23 TClientDataSet.BeforeInsert Event

Occurs before an application enters insert mode.

Pascal

```
property BeforeInsert: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeInsert;
```

Description

Write a BeforeInsert event handler to take specific action before an application inserts or appends a new record. The Insert or Append method generates a BeforeInsert method before it sets the dataset into dsInsert state.

Delphi Examples:

```
{
This example updates a memo with a message when an
AfterInsert event or a BeforeInsert event occurs.
}
procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After inserting a record');
end;

procedure TForm1.ClientDataSet1BeforeInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before inserting a record');
end;
{
This example uses the BeforeInsert event to do data
validation. If the StrToInt function raises an exception,
the edit control's contents are set to a valid value so
that the assignment to the INTEGER field in the dataset
will succeed.  Continuing after the exception causes a
'0' to be placed in the PORTA field.
DataSet design:
This example requires a ClientDataSet, a DBGrid, a
DataSource and a DBNavigator.  Name the ClientDataSet CDS.
Set the DataSet property in the DataSource to CDS and name
that DataSource DS.  Set the DataSource in the DBNavigator
```

```
  to DS the DataSource in the DBGrid to DS.  Right click on CDS
  and select the Fields Editor, then right click on the Field
  Editor form and create two fields by selecting "new field".
  In the field configuration window, set the name to HOST, the
  Component value will automatically be set to CDSHOST.  Set
  the filed type to String.  In the field configuration window
  for the second field, set the name to PORTA, the Component
  value will automatically be set to CDSPORTA.  Set the field
  type to Integer.  The file used to load CDS must have an
  attribute name of HOST with an attribute type of string and
  an attribute name of PORTA with an attribute type of integer.
  Add DB, DBCtrls, Grids, DBGrids and DBClient to the uses clause.
  }
  {$IFNDEF UNICODE}
  uses SwSystem;
  {$ENDIF}

  procedure TForm1.Button3Click(Sender: TObject);
  begin
    CDS.Insert;
    CDS.FieldByName('PORTA').AsInteger := StrToInt(Edit1.Text);
    CDS.Post;
  end;

  procedure TForm1.CDSBeforeInsert(DataSet: TDataSet);
  begin
    try
    { Make sure edit field can be converted to integer --
     this will raise an exception if it can't }
      StrToInt(Edit1.Text);
    except
      Edit1.Text := '0';
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
  {$IFDEF UNICODE}
    CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
  {$ELSE}
    CDS.LoadFromFile(gsAppPath + 'CDS.XML');
  {$ENDIF}
  end;
```

See Also

State (see page 7036)

AfterInsert (see page 7044)

Append (see page 6910)

Insert (see page 6959)

45.1.6.3.24 TClientDataSet.BeforeOpen Event

Occurs before an application executes a request to open a dataset.

Pascal

```
property BeforeOpen: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeOpen;
```

Description

Write a BeforeOpen event handler to take specific action before an application opens a dataset for viewing or editing. BeforeOpen is triggered when an application sets the Active property to true for a dataset or an application calls Open.

See Also

Active (see page 7004)

AfterOpen (see page 7045)

Open (see page 6976)

Opening and Closing Datasets

45.1.6.3.25 TClientDataSet.BeforePost Event

Occurs before an application posts changes for the active record to the database or change log.

Pascal

```
property BeforePost: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforePost;
```

Description

Write a BeforePost event handler to take specific action before an application posts dataset changes. BeforePost is triggered when an application calls the Post method. Post checks to make sure all required fields are present, then calls BeforePost before posting the record.

An application might use BeforePost to perform validity checks on data changes before committing them. If it encountered a validity problem, it could call Abort to cancel the Post operation (Delphi) or throw an exception (C++).

Delphi Examples:

```
{
This example checks for a valid entry in a TDBEdit control
and calls the Abort procedure if the control is empty; Abort
cancels the post before it happens.
}
procedure TForm1.ClientDataSet1BeforePost(DataSet: TDataSet);
begin
  if DBEdit1.Text = '' then
    Abort;
  DataSet.Fields[2].AsString := DBEdit1.Text;
end;
```

See Also

AfterPost (see page 7046)

Post (see page 6977)

Abort (see page 22786)

Working with Field Component Methods at Runtime

45.1.6.3.26 TClientDataSet.BeforeRefresh Event

Occurs immediately before an application refreshes the data in the dataset.

Pascal

```
property BeforeRefresh: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeRefresh;
```

Description

Write a BeforeRefresh event handler to take specific action immediately before an application updates the records in the dataset. BeforeRefresh is generated by calls to the Refresh method.

See Also

AfterRefresh (see page 7047)

Refresh (see page 6991)

45.1.6.3.27 TClientDataSet.BeforeRowRequest Event

Occurs before the client dataset fetches new information about the current record.

Pascal

```
property BeforeRowRequest: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeRowRequest;
```

Description

Write a BeforeRowRequest event handler to send custom information to the client dataset's provider. BeforeRowRequest is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches detail information about the current record or refreshes the current record, the following events occur:

1.The client dataset receives a BeforeRowRequest event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider on the application server receives a BeforeRowRequest event, where it can respond to or change that information before it looks up the requested information.

3.The provider creates a delta packet that contains the requested information.

4.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeRowRequest event handler.

5.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider's AfterRowRequest event handler.

See Also

AS_RowRequest (see page 16456)

RowRequest (see page 18472)

BeforeRowRequest (see page 18482)

AfterRowRequest (see page 8509)

FetchBlobs (see page 8414)

FetchParams (see page 8415)

RefreshRecord (see page 8442)

Communicating with the Client Dataset

45.1.6.3.28 TClientDataSet.BeforeScroll Event

Occurs before an application scrolls from one record to another.

Pascal

```
property BeforeScroll: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeScroll;
```

Description

Write a BeforeScroll event handler to take specific action immediately before an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. BeforeScroll is called before all other events triggered by these methods and any other methods that switch from record to record in the dataset.

See Also

AfterScroll (see page 7047)

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Prior (see page 6979)

Navigating Datasets

45.1.6.3.29 TClientDataSet.OnCalcFields Event

Occurs when an application recalculates calculated fields.

Pascal

```
property OnCalcFields: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent OnCalcFields;
```

Description

Write an OnCalcFields event handler to take specific action when an application recalculates calculated fields. A calculated field is one that derives its value from the values in one or more fields in the dataset, sometimes with additional processing.

OnCalcFields is triggered when:

A dataset is opened.

A dataset is put into dsEdit state.

A record is retrieved from a database.

When the AutoCalcFields property is true, OnCalcFields is also triggered when:

Focus moves from one visual control to another, or from one column to another is a data-aware grid control and modifications were made to the record.

Note: When the AutoCalcFields property is true, an OnCalcFields event handler should not modify the dataset (or a linked dataset if it is part of a master-detail relationship), because such modifications retrigger the OnCalcField event, leading to infinite recursion.

If an application permits users to change data, OnCalcFields is frequently triggered. To reduce the frequency with which OnCalcFields occurs, set AutoCalcFields to false.

Warning: When the dataset is the master table of a master-detail relationship, OnCalcFields occurs before detail sets have been synchronized with the master table.

See Also

Active (see page 7004)

AutoCalcFields (see page 7006)

Open (see page 6976)

Post (see page 6977)

Calculating Fields

Using Internally Calculated Fields in Client Datasets

Defining a Calculated Field

45.1.6.3.30 TClientDataSet.OnDeleteError Event

Occurs when an application attempts to delete a record and an exception is raised.

Pascal

```
property OnDeleteError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnDeleteError;
```

Description

Write an OnDeleteError event handler to handle exceptions that occur when an attempt to delete a record fails.

See Also

OnEditError (see page 7054)

OnPostError (see page 7056)

Delete (see page 6924)

45.1.6.3.31 TClientDataSet.OnEditError Event

Occurs when an application attempts to modify or insert a record and an exception is raised.

Pascal

```
property OnEditError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnEditError;
```

Description

Write an OnEditError event handler to handle exceptions that occur when an attempt to edit a record fails.

See Also

OnDeleteError (see page 7054)

OnPostError (see page 7056)

Edit (see page 6932)

Insert (see page 6959)

45.1.6.3.32 TClientDataSet.OnFilterRecord Event

Occurs each time a different record in the dataset becomes the active record and filtering is enabled.

Pascal

```
property OnFilterRecord: TFilterRecordEvent;
```

C++

```
__property TFilterRecordEvent OnFilterRecord;
```

Description

Write an OnFilterRecord event handler to specify for each record in a dataset whether it should be visible to the application. To indicate that a record passes the filter condition, the OnFilterRecord event handler must set the Accept parameter to true. To exclude a record, set the Accept parameter to false. Most TDataSet descendants (such as ADO datasets or BDE-based datasets) initialize Accept to true before calling the OnFilterRecord event handler.

Warning: Do not supply an OnFilterRecord event handler to a unidirectional dataset. Unidirectional datasets do not support filters, and assigning an OnFilterRecord event handler causes the dataset to raise an exception.

Filtering is enabled if the Filtered property is true. When an application is processing a filter, the State property for the dataset is dsFilter.

Use an OnFilterRecord event handler to filter records using a criterion that can't be implemented using the Filter property. For example, using the Filter property, field comparisons are not supported against local tables (Paradox, dBASE, Access, FoxPro), but an OnFilterRecord event handler can implement any criterion at all.

Tip: Be sure that any interactions between the Filter property and the OnFilterRecord event handler do not result in an empty filter set when they are used simultaneously in an application.

See Also

Filter (see page 7026)

Filtered (see page 7027)

FilterOptions (see page 7027)

State (see page 7036)

Creating Filters

Navigating Records in a Filtered Dataset

Displaying and Editing a Subset of Data Using Filters

Writing an OnFilterRecord Event Handler

45.1.6.3.33 TClientDataSet.OnNewRecord Event

Occurs when an application inserts or appends a new dataset record.

Pascal

```
property OnNewRecord: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent OnNewRecord;
```

Description

Write an OnNewRecord event handler to take specific actions as an application inserts or appends a new record. OnNewRecord is called as part of the insert or append process. An application might use the OnNewRecord event to set initial values for a record or as a way of implementing cascading insertions in related datasets.

See Also

AfterInsert (see page 7044)

BeforeInsert (see page 7050)

45.1.6.3.34 TClientDataSet.OnPostError Event

Occurs when an application attempts to modify or insert a record and an exception is raised.

Pascal

```
property OnPostError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnPostError;
```

Description

Write an OnPostError event handler to handle exceptions that occur when an attempt to post a record fails.

See Also

OnDeleteError (see page 7054)

OnEditError (see page 7054)

Post (see page 6977)

45.1.6.3.35 TClientDataSet.OnReconcileError Event

Occurs when a client dataset needs to reconcile an update to a record that could not be applied.

Pascal

```
property OnReconcileError: TReconcileErrorEvent;
```

C++

```
__property TReconcileErrorEvent OnReconcileError;
```

Description

Write an OnReconcileError event handler to respond to error conditions that arise when the Reconcile method cycles through records that could not be applied to the database. This event handles only those update errors that were not handled in an OnUpdateError event handler.

The OnReconcileError event handler is called once for each record that could not be applied to the dataset or handled by the provider.

Warning: A provider can't detect when an update conflicts with another user's changes to a memo field. This situation does not generate an OnReconcileError event.

See Also

Reconcile (see page 8441)

OldValue (see page 7175)

NewValue (see page 7174)

CurValue (see page 7155)

ApplyUpdates (see page 8394)

OnUpdateError (see page 18444)

OnUpdateError

Applying Updates

Setting Up a Simple Dataset

Reconciling Update Errors

Using the BDE to Cache Updates

Updating Data in dbExpress Applications

45.1.7 TConnectionBroker Class

TConnectionBroker centralizes the connection to an application server for a set of client datasets that all use the same connection.

Class Hierarchy

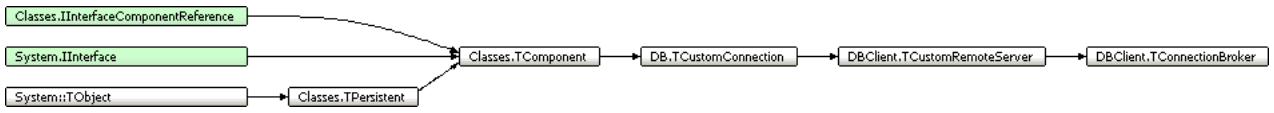

Pascal

```
TConnectionBroker = class(TCustomRemoteServer);
```

C++

```
class TConnectionBroker : public TCustomRemoteServer;
```

File

DBClient

Description

Use a connection broker when your application has several client datasets that all use the same connection component to establish a connection to an application server. If you set the ConnectionBroker property of all your client datasets to a single connection broker, then you can change the connection component that handles the connection to an application server for all of them by changing the connection broker's Connection (see page 8368) property. That is, instead of changing the RemoteServer property of every client dataset, you can change a single property on the connection broker.

Similarly, the connection broker centralizes the use of the application server's interface and events that occur when you open or close a connection to the application server. Thus, if you need to change the connection component (for example, to use a different protocol), you do need to rewrite your event handlers for the BeforeConnect (see page 8372), AfterConnect (see page 8371), OnGetUserName, OnLogin (see page 8370), BeforeDisconnect (see page 8372), or AfterDisconnect (see page 8372) events or any calls that use the connection broker's AppServer (see page 8532) property. The AppServer (see page 8532) property and the event handlers can reside on the protocol-neutral connection broker rather than on the connection component, and so will not disappear when you switch to a different connection component.

See Also

TCustomClientDataSet (see page 8373)

TSocketConnection (see page 18928)

TWebConnection (see page 18968)

TDCOMConnection (see page 16008)

TSoapConnection (see page 19864)

45.1.7.1 TConnectionBroker Members

The following tables list the members exposed by TConnectionBroker.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoConnect (see page 6864)	Provides the interface for a method that opens a connection.
V	DoDisconnect (see page 6864)	Provides the interface for a method that terminates the connection.
V	GetAppServer (see page 8365)	Returns the value of the AppServer property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetConnected (see page 8365)	Returns the value of the Connected property.
V	GetDataSet (see page 6865)	Returns a dataset, given its index in the DataSets (see page 6870) property array.
V	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 8366)	Initializes the connection component after all objects in the form or data module have been loaded.
V	Notification (see page 8366)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	RegisterClient (see page 6866)	Registers an object that uses the connection component.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAppServer (see page 8531)	Sets the AppServer (see page 8532) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetConnected (see page 8367)	Changes the value of the Connected property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	UnRegisterClient (see page 6868)	Removes a registered client so that it is no longer associated with the connection component.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 6863)	Closes the connection.

	Create (see page 8364)	Creates an instance of a remote server component.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 6863)	Destroys the instance of a connection component.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProviderNames (see page 8530)	Executes a callback for all provider objects on the application server.
	GetServer (see page 8366)	Returns the IAppServer interface of the application server.
	GetServerList (see page 8531)	Returns a list of application servers (remote data modules) to which the component can connect.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 6866)	Opens the connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AppServer (see page 8532)	Contains the interface for communicating with the application server.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 6870)	Lists the datasets currently using this connection component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterConnect (see page 8371)	Occurs after a connection is established.
	AfterDisconnect (see page 8372)	Occurs after the connection closes.
	BeforeConnect (see page 8372)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 8372)	Occurs immediately before the connection closes.
	OnGetUsername (see page 8373)	This is OnGetUsername, a member of class TConnectionBroker.

Published Properties

	Name	Description
	Connected (see page 8368)	Determines whether a connection has been established to the remote source of data.
	Connection (see page 8368)	Specifies the connection component that actually forms the connection to the application server.
	LoginPrompt (see page 8369)	Specifies whether a login dialog appears immediately before opening a new connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OnLogin (see page 8370)	Occurs when a communication channel opens to the server.
	Tag (see page 2808)	Stores an integer value as part of a component.

45.1.7.2 TConnectionBroker.Create Constructor

Creates an instance of a remote server component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TConnectionBroker(TComponent * AOwner);
```

Description

Create instantiates an instance of a remote server component and initializes its properties.

AOwner is the component that is responsible for freeing the TCustomRemoteServer instance. It becomes the value of the Owner property.

See Also

Create (see page 6862)

Owner (see page 2807)

45.1.7.3 TConnectionBroker Methods

The methods of the TConnectionBroker class are listed here.

Protected Methods

	Name	Description
	GetAppServer (see page 8365)	Returns the value of the AppServer property.
	GetConnected (see page 8365)	Returns the value of the Connected property.
	Loaded (see page 8366)	Initializes the connection component after all objects in the form or data module have been loaded.
	Notification (see page 8366)	Forwards notification messages to all owned components.
	SetConnected (see page 8367)	Changes the value of the Connected property.

Public Methods

	Name	Description
	GetServer (see page 8366)	Returns the IAppServer interface of the application server.

45.1.7.3.1 TConnectionBroker.GetAppServer Method

Returns the value of the AppServer property.

Pascal

```
function GetAppServer: Variant; override;
```

C++

```
virtual __fastcall Variant GetAppServer();
```

Description

GetAppServer is the protected implementation of the AppServer property. Override GetAppServer to change the way this property is stored or managed.

See Also

AppServer (see page 8532)

GetServer (see page 8530)

45.1.7.3.2 TConnectionBroker.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected read implementation of the Connected property. Descendant classes override this method to

return true if a connection is open and false otherwise. As defined in TCustomConnection, GetConnected always returns false.

See Also

Connected (see page 6868)

SetConnected (see page 6867)

45.1.7.3.3 TConnectionBroker.GetServer Method

Returns the IAppServer interface of the application server.

Pascal

```
function GetServer: IAppServer; override;
```

C++

```
virtual __fastcall IAppServer GetServer();
```

Description

Descendant classes override this function to return the interface of the application server after establishing a connection. As implemented in TCustomRemoteServer, GetServer always returns nil (Delphi) or NULL (C++).

See Also

Connected (see page 6868)

AppServer (see page 8532)

AppServer (see page 8462)

45.1.7.3.4 TConnectionBroker.Loaded Method

Initializes the connection component after all objects in the form or data module have been loaded.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. It is called automatically after the form or data module that contains the connection component has loaded all its owned objects. TCustomConnection overrides the inherited method to establish a connection if the Connected property is true.

See Also

StreamedConnected (see page 6874)

Loaded (see page 2782)

45.1.7.3.5 TConnectionBroker.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

45.1.7.3.6 TConnectionBroker.SetConnected Method

Changes the value of the Connected property.

Pascal

```
procedure SetConnected(Value: Boolean); override;
```

C++

```
virtual __fastcall SetConnected(Boolean Value);
```

Description

SetConnected is the protected write implementation of the Connected property. Most descendants of TCustomConnection do not override this method to change the implementation of Connected. Instead, they override the DoConnect and DoDisconnect methods to change the way the connection is opened and closed.

See Also

Connected (see page 6868)

DoConnect (see page 6864)

DoDisconnect (see page 6864)

45.1.7.4 TConnectionBroker Properties

The properties of the TConnectionBroker class are listed here.

Published Properties

	Name	Description
	Connected (see page 8368)	Determines whether a connection has been established to the remote source of data.
	Connection (see page 8368)	Specifies the connection component that actually forms the connection to the application server.

	LoginPrompt (see page 8369)	Specifies whether a login dialog appears immediately before opening a new connection.
	OnLogin (see page 8370)	Occurs when a communication channel opens to the server.

45.1.7.4.1 TConnectionBroker.Connected Property

Determines whether a connection has been established to the remote source of data.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Set Connected to true to open the connection. Set Connected to false to terminate the connection.

Setting Connected to true generates a BeforeConnect event, calls the protected DoConnect method to establish the connection, and generates an AfterConnect event. In addition, when setting Connected to true, TCustomConnection may display a login dialog, depending on the value of LoginPrompt.

Setting Connected to false generates a BeforeDisconnect event, calls the protected DoConnect method to drop the connection, and generates an AfterDisconnect event.

When deriving custom connection components from TCustomConnection, override GetConnected to return true when a connection is established, and override DoConnect and DoDisconnect to create and drop the connection.

See Also

DataSets (see page 6870)

Open (see page 6866)

Close (see page 6863)

DoConnect (see page 6864)

DoDisconnect (see page 6864)

BeforeConnect (see page 6875)

AfterConnect (see page 6874)

BeforeDisconnect (see page 6875)

AfterDisconnect (see page 6875)

GetConnected (see page 6865)

LoginPrompt (see page 6872)

Disconnecting from a Database Server

Connecting to a Database Server

45.1.7.4.2 TConnectionBroker.Connection Property

Specifies the connection component that actually forms the connection to the application server.

Pascal

```
property Connection: TCustomRemoteServer;
```

C++

```
__property TCustomRemoteServer Connection;
```

Description

Set Connection to the connection component that establishes a connection to the application server for all the client datasets listed in the DataSets property. For each client dataset listed in DataSets, the connection broker assigns Connection to its RemoteServer property. This connection locates the application server and obtains its IAppServer interface, which the client datasets then use for communicating with remote providers.

See Also

RemoteServer (see page 8493)

DataSets (see page 6870)

Connecting to a Data Store Using TADOConnection

Using TSimpleDataSet

45.1.7.4.3 TConnectionBroker.LoginPrompt Property

Specifies whether a login dialog appears immediately before opening a new connection.

Pascal

```
property LoginPrompt: Boolean;
```

C++

```
__property Boolean LoginPrompt;
```

Description

Set LoginPrompt to true to provide login support when establishing a connection. LoginPrompt controls two things: the occurrence of the OnLogin event, and the appearance of a default login dialog that prompts users for a name and password when you include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi). When the OnLogin event occurs and when the default login dialog appears depend on the type of connection component:

For TDatabase, TIBDatabase, and TSQLConnection, the dialog appears after the BeforeConnect event and before the AfterConnect event, unless you supply an OnLogin event handler. If there is an OnLogin event handler, that event occurs in place of the login dialog, and there is no need to include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi). If correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler, the connection fails. The OnLogin event does not fire unless LoginPrompt is set to true.

For TADOConnection components, the dialog appears after the OnWillConnect event and before the BeforeConnect event. If there is an OnLogin event handler, that event occurs after the login dialog. If you do not include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi), the OnLogin event occurs but there is no default login dialog. If correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler, the connection fails. The OnLogin event does not fire unless LoginPrompt is set to true.

For DataSnap SOAP connection components, the dialog appears after the OnGetUsername event and before the BeforeConnect, AfterConnect, and OnLogin events. If the user cancels from the login dialog, no attempt is made to open a connection.

When LoginPrompt is false, the application must supply user name and password values programmatically:

For TDatabase, the user name and password can be supplied as USER NAME and PASSWORD parameters in the Params property.

For TADOConnection, the user name and password can be supplied as the ConnectionString property.

For TSQLConnection, the user name, password, and database can be supplied as UserName, Password, and Database parameters in the Params property, or provided as connection parameters associated with the connection name.

For DataSnap SOAP connection components, there is no built-in use for the user name and password supplied by the login dialog. (The UserName and Password properties of TWebConnection are unrelated).

Warning: Storing hard-coded user name and password entries as property values or in code for an OnLogin event handler can compromise server security.

See Also

OnLogin (see page 6873)

BeforeConnect (see page 6875)

AfterConnect (see page 6874)

OnGetUserName (see page 8533)

OnLogin (see page 20259)

OnLogin (see page 9994)

Controlling Server Login

45.1.7.4.4 TConnectionBroker.OnLogin Property

Occurs when a communication channel opens to the server.

Pascal

```
property OnLogin: TLoginEvent;
```

C++

```
__property TLoginEvent OnLogin;
```

Description

OnLogin occurs when connections are opened if the LoginPrompt property is true.

For DataSnap SOAP connection components, the following sequence of events occurs when opening a connection to an application server:

1. An OnGetUsername event occurs, where the application can supply an initial user name for the login dialog.
2. The login dialog appears, where the user can supply a user name and password.
3. A BeforeConnect event occurs, the connection is opened, and an AfterConnect event occurs.
4. The OnLogin event occurs, where you can write code to log in to the application server using the username and password that were provided in the remote login dialog.

   Note: Do not use the OnLogin event if the connection component provides a different scheme for logging in. For example, TWebConnection uses the Username and Password properties instead.

   For ADO connection components, the following sequence of events occurs when opening a connection to an application server:

1. A BeforeConnect event occurs.

2. An OnWillConnect event occurs, where the application can supply an initial user name and password for the login dialog, as well as the connection options and status.

3. The login dialog appears, where the user can supply a user name and password.

4. The OnLogin event occurs, where you can write code to respond to the user-supplied values. (For example, you could convert an application-specific user name and password to database user name and password).

5. The connection is opened, and an AfterConnect event occurs.

Note: Some

C connection components that do not connect to an application server in a multitiered database application replace the OnLogin event defined in TCustomConnection with a different event that occurs before a connection is established.

See Also

Connected (see page 6868)

Open (see page 6866)

OnGetUserName (see page 8533)

BeforeConnect (see page 6875)

AfterConnect (see page 6874)

LoginPrompt (see page 6872)

OnWillConnect (see page 1039)

45.1.7.5 TConnectionBroker Events

The events of the TConnectionBroker class are listed here.

Published Events

	Name	Description
	AfterConnect (see page 8371)	Occurs after a connection is established.
	AfterDisconnect (see page 8372)	Occurs after the connection closes.
	BeforeConnect (see page 8372)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 8372)	Occurs immediately before the connection closes.
	OnGetUsername (see page 8373)	This is OnGetUsername, a member of class TConnectionBroker.

45.1.7.5.1 TConnectionBroker.AfterConnect Event

Occurs after a connection is established.

Pascal

```
property AfterConnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterConnect;
```

Description

Write an AfterConnect event handler to take application-specific actions immediately after the connection component opens a connection to the remote source of database information.

See Also

BeforeConnect (see page 6875)

Connected (see page 6868)

Connecting to a Database Server

45.1.7.5.2 TConnectionBroker.AfterDisconnect Event

Occurs after the connection closes.

Pascal

```
property AfterDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterDisconnect;
```

Description

Write an AfterDisconnect event handler to take application-specific actions after the connection component drops a connection.

See Also

BeforeDisconnect (see page 6875)

Connected (see page 6868)

Disconnecting from a Database Server

45.1.7.5.3 TConnectionBroker.BeforeConnect Event

Occurs immediately before establishing a connection.

Pascal

```
property BeforeConnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeConnect;
```

Description

Write a BeforeConnect event handler to take application-specific actions before the connection component opens a connection to the remote source of database information.

See Also

AfterConnect (see page 6874)

Connected (see page 6868)

Connecting to a Database Server

45.1.7.5.4 TConnectionBroker.BeforeDisconnect Event

Occurs immediately before the connection closes.

Pascal

```
property BeforeDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeDisconnect;
```

Description

Write a BeforeDisconnect event handler to take application-specific actions before dropping a connection.

Delphi Examples:

```
{
This BeforeDisconnect event handler makes sure that the
client dataset has applied all its pending updates before
terminating the connection to the server.
}
procedure TForm1.RemoteServer1BeforeDisconnect(Sender: TObject);
begin
  if (ClientDataSet1.ChangeCount > 0) then
    ClientDataSet1.ApplyUpdates(-1);
end;
```

See Also

AfterDisconnect (see page 6875)

Connected (see page 6868)

Disconnecting from a Database Server

45.1.7.5.5 TConnectionBroker.OnGetUsername Event

Pascal

```
property OnGetUsername: TGetUsernameEvent;
```

C++

```
__property TGetUsernameEvent OnGetUsername;
```

Description

This is OnGetUsername, a member of class TConnectionBroker.

45.1.8 TCustomClientDataSet Class

TCustomClientDataSet is the base class for datasets that cache their data in memory.

Class Hierarchy

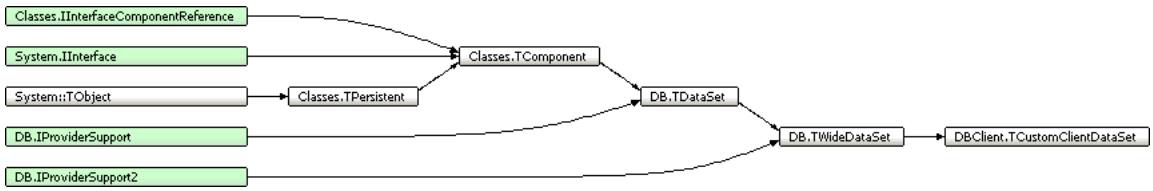

Pascal

```
TCustomClientDataSet = class(TWideDataSet);
```

C++

```
class TCustomClientDataSet : public TWideDataSet;
```

File

DBClient

Description

TCustomClientDataSet represents an in-memory dataset that stores its data in transportable data packets. On Windows, it relies on Midas.dll or Midaslib.dcu, and On Linux, it relies on midas.so to provide the underlying support for managing database operations such as navigating records, posting records to a change log, filtering records, and so on.

There are two basic ways a client dataset can obtain its data and apply its updates:

It can read from and write to a dedicated file on disk, acting as a file-based dataset.

It can obtain records from another dataset and post updates to that dataset or its database server. When working with another dataset, the client dataset communicates with that source dataset using a dataset provider. Depending on the TCustomClientDataSet descendant, the provider and the source dataset can be external to the client dataset, or they can be internal, owned objects.

Client datasets are used primarily for caching updates or to represent the dataset in the client portion of a multi-tiered database application.

Do not create instances of TCustomClientDataSet. TCustomClientDataSet is intended only as a base class for datasets that store data in memory as data packets. To add this functionality to a database application, use a TCustomClientDataSet descendant such as TClientDataSet (see page 8289), TBDEClientDataSet, or TSQLClientDataSet.

See Also

TDataSetProvider (see page 18508)

TCustomRemoteServer (see page 8525)

IAppServer (see page 16450)

TClientDataSet (see page 8289)

TSQLClientDataSet

TBDEClientDataSet

TIBClientDataSet

45.1.8.1 TCustomClientDataSet Members

The following tables list the members exposed by TCustomClientDataSet.

Protected Events

	Name	Description
	AfterApplyUpdates (see page 8497)	Occurs after the client dataset applies updates to the provider but before it attempts to reconcile update errors.
	AfterCancel (see page 8498)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 8500)	Occurs after an application closes a dataset.
	AfterDelete (see page 8501)	Occurs after an application deletes a record.
	AfterEdit (see page 8504)	Occurs after an application starts editing a record.
	AfterExecute (see page 8504)	Occurs after the client dataset receives the results from an execute command.
	AfterGetParams (see page 8505)	Occurs after the client dataset receives parameter values from the provider.
	AfterGetRecords (see page 8506)	Occurs after the client dataset fetches a data packet from the provider.
	AfterInsert (see page 8507)	Occurs after an application inserts a new record.
	AfterOpen (see page 8507)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 8509)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 8509)	Occurs after an application refreshes the data in the dataset.
	AfterRowRequest (see page 8509)	Occurs after the client dataset fetches new information about the current record.
	AfterScroll (see page 8510)	Occurs after an application scrolls from one record to another.
	BeforeApplyUpdates (see page 8511)	Occurs before the client dataset applies updates to the remote server.
	BeforeCancel (see page 8512)	Occurs before an application executes a request to cancel changes to the active record.

	BeforeClose (see page 8512)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 8512)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 8513)	Occurs before an application enters edit mode for the active record.
	BeforeExecute (see page 8514)	Occurs before the client dataset executes the provider's query or stored procedure.
	BeforeGetParams (see page 8514)	Occurs before the client dataset fetches parameter values from the provider.
	BeforeGetRecords (see page 8515)	Occurs before the client dataset fetches a data packet from the provider.
	BeforeInsert (see page 8517)	Occurs before an application enters insert mode.
	BeforeOpen (see page 8518)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 8518)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 8519)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeRowRequest (see page 8520)	Occurs before the client dataset fetches new information about the current record.
	BeforeScroll (see page 8520)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 8521)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 8522)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 8522)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 8523)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 8523)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 8524)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnReconcileError (see page 8524)	Occurs when a client dataset needs to reconcile an update to a record that could not be applied.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	ActivateFilters (see page 8391)	This is ActivateFilters, a member of class TCustomClientDataSet.
V	AddDataPacket (see page 8391)	This is AddDataPacket, a member of class TCustomClientDataSet.
	AddFieldDesc (see page 8391)	This is AddFieldDesc, a member of class TCustomClientDataSet.
	AllocKeyBuffers (see page 8392)	This is AllocKeyBuffers, a member of class TCustomClientDataSet.
V	AllocRecordBuffer (see page 8393)	This is AllocRecordBuffer, a member of class TCustomClientDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Check (see page 8397)	This is Check, a member of class TCustomClientDataSet.
V	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
V	CheckDetailRecords (see page 8397)	This is CheckDetailRecords, a member of class TCustomClientDataSet.
V	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
V	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
V	CheckProviderEOF (see page 8397)	This is CheckProviderEOF, a member of class TCustomClientDataSet.
	CheckSetKeyMode (see page 8397)	This is CheckSetKeyMode, a member of class TCustomClientDataSet.
V	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.

	ClearCalcFields (see page 8398)	This is ClearCalcFields, a member of class TCustomClientDataSet.
	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
	CloseCursor (see page 8400)	This is CloseCursor, a member of class TCustomClientDataSet.
	ConstraintsStored (see page 8401)	This is ConstraintsStored, a member of class TCustomClientDataSet.
	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
	DataConvert (see page 8404)	This is DataConvert, a member of class TCustomClientDataSet.
	DataEvent (see page 8404)	This is DataEvent, a member of class TCustomClientDataSet.
	DeactivateFilters (see page 8405)	This is DeactivateFilters, a member of class TCustomClientDataSet.
	DefChanged (see page 8405)	This is DefChanged, a member of class TCustomClientDataSet.
	DefineProperties (see page 8405)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
	DestroyLookupCursor (see page 8406)	This is DestroyLookupCursor, a member of class TCustomClientDataSet.
	DoAfterApplyUpdates (see page 8407)	This is DoAfterApplyUpdates, a member of class TCustomClientDataSet.
	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
	DoAfterExecute (see page 8407)	This is DoAfterExecute, a member of class TCustomClientDataSet.
	DoAfterGetParams (see page 8407)	This is DoAfterGetParams, a member of class TCustomClientDataSet.
	DoAfterGetRecords (see page 8407)	This is DoAfterGetRecords, a member of class TCustomClientDataSet.
	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterRowRequest (see page 8408)	This is DoAfterRowRequest, a member of class TCustomClientDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoApplyUpdates (see page 8408)	This is DoApplyUpdates, a member of class TCustomClientDataSet.
	DoBeforeApplyUpdates (see page 8408)	This is DoBeforeApplyUpdates, a member of class TCustomClientDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeExecute (see page 8408)	This is DoBeforeExecute, a member of class TCustomClientDataSet.
	DoBeforeGetParams (see page 8408)	This is DoBeforeGetParams, a member of class TCustomClientDataSet.
	DoBeforeGetRecords (see page 8409)	This is DoBeforeGetRecords, a member of class TCustomClientDataSet.
	DoBeforeInsert (see page 8409)	This is DoBeforeInsert, a member of class TCustomClientDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeRowRequest (see page 8409)	This is DoBeforeRowRequest, a member of class TCustomClientDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoExecute (see page 8409)	This is DoExecute, a member of class TCustomClientDataSet.
	DoGetRecords (see page 8410)	This is DoGetRecords, a member of class TCustomClientDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 8410)	This is DoOnNewRecord, a member of class TCustomClientDataSet.
	DoRowRequest (see page 8410)	This is DoRowRequest, a member of class TCustomClientDataSet.
	FetchMoreData (see page 8415)	This is FetchMoreData, a member of class TCustomClientDataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FindRecord (see page 8417)	This is FindRecord, a member of class TCustomClientDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeKeyBuffers (see page 8417)	This is FreeKeyBuffers, a member of class TCustomClientDataSet.
	FreeRecordBuffer (see page 8417)	This is FreeRecordBuffer, a member of class TCustomClientDataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.

	GetAggregateValue (see page 8418)	This is GetAggregateValue, a member of class TCustomClientDataSet.
	GetAppServer (see page 8418)	This is GetAppServer, a member of class TCustomClientDataSet.
	GetBookmarkData (see page 8418)	This is GetBookmarkData, a member of class TCustomClientDataSet.
	GetBookmarkFlag (see page 8418)	This is GetBookmarkFlag, a member of class TCustomClientDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 8418)	This is GetCanModify, a member of class TCustomClientDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
	GetDataSource (see page 8419)	This is GetDataSource, a member of class TCustomClientDataSet.
	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIndexField (see page 8421)	This is GetIndexField, a member of class TCustomClientDataSet.
	GetIndexFieldCount (see page 8421)	This is GetIndexFieldCount, a member of class TCustomClientDataSet.
	GetIsClone (see page 8422)	This is GetIsClone, a member of class TCustomClientDataSet.
	GetIsIndexField (see page 8422)	This is GetIsIndexField, a member of class TCustomClientDataSet.
	GetKeyBuffer (see page 8423)	This is GetKeyBuffer, a member of class TCustomClientDataSet.
	GetKeyExclusive (see page 8423)	This is GetKeyExclusive, a member of class TCustomClientDataSet.
	GetKeyFieldCount (see page 8423)	This is GetKeyFieldCount, a member of class TCustomClientDataSet.
	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
	GetRecNo (see page 8425)	This is GetRecNo, a member of class TCustomClientDataSet.
	GetRecord (see page 8425)	This is GetRecord, a member of class TCustomClientDataSet.
	GetRecordCount (see page 8426)	This is GetRecordCount, a member of class TCustomClientDataSet.
	GetRecordSize (see page 8426)	This is GetRecordSize, a member of class TCustomClientDataSet.
	GetRemoteServer (see page 8426)	This is GetRemoteServer, a member of class TCustomClientDataSet.
	GetStateFieldValue (see page 8426)	This is GetStateFieldValue, a member of class TCustomClientDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
	InitKeyBuffer (see page 8428)	This is InitKeyBuffer, a member of class TCustomClientDataSet.
	InitRecord (see page 8428)	This is InitRecord, a member of class TCustomClientDataSet.
	InternalAddRecord (see page 8428)	This is InternalAddRecord, a member of class TCustomClientDataSet.
	InternalCancel (see page 8429)	This is InternalCancel, a member of class TCustomClientDataSet.
	InternalClose (see page 8429)	This is InternalClose, a member of class TCustomClientDataSet.
	InternalDelete (see page 8429)	This is InternalDelete, a member of class TCustomClientDataSet.
	InternalEdit (see page 8429)	This is InternalEdit, a member of class TCustomClientDataSet.
	InternalFirst (see page 8430)	This is InternalFirst, a member of class TCustomClientDataSet.
	InternalGetOptionalParam (see page 8430)	This is InternalGetOptionalParam, a member of class TCustomClientDataSet.
	InternalGotoBookmark (see page 8430)	This is InternalGotoBookmark, a member of class TCustomClientDataSet.
	InternalHandleException (see page 8430)	This is InternalHandleException, a member of class TCustomClientDataSet.
	InternalInitFieldDefs (see page 8430)	This is InternalInitFieldDefs, a member of class TCustomClientDataSet.
	InternalInitRecord (see page 8431)	This is InternalInitRecord, a member of class TCustomClientDataSet.

	InternalInsert (see page 8431)	This is InternalInsert, a member of class TCustomClientDataSet.
	InternalLast (see page 8431)	This is InternalLast, a member of class TCustomClientDataSet.
	InternalOpen (see page 8431)	This is InternalOpen, a member of class TCustomClientDataSet.
	InternalPost (see page 8432)	This is InternalPost, a member of class TCustomClientDataSet.
	InternalRefresh (see page 8432)	This is InternalRefresh, a member of class TCustomClientDataSet.
	InternalSetOptionalParam (see page 8432)	This is InternalSetOptionalParam, a member of class TCustomClientDataSet.
	InternalSetToRecord (see page 8432)	This is InternalSetToRecord, a member of class TCustomClientDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsCursorOpen (see page 8432)	This is IsCursorOpen, a member of class TCustomClientDataSet.
	Loaded (see page 8433)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 8436)	This is LocateRecord, a member of class TCustomClientDataSet.
	Notification (see page 8437)	Forwards notification messages to all owned components.
	OpenCursor (see page 8438)	This is OpenCursor, a member of class TCustomClientDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PostKeyBuffer (see page 8439)	This is PostKeyBuffer, a member of class TCustomClientDataSet.
	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSExecuteStatement (see page 7535)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 8439)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandTextW (see page 7537)	This is PSGetCommandTextW, a member of class TWideDataSet.
	PSGetCommandType (see page 8440)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 8440)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7538)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetKeyFieldsW (see page 7538)	This is PSGetKeyFieldsW, a member of class TWideDataSet.
	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7538)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetQuoteCharW (see page 7539)	This is PSGetQuoteCharW, a member of class TWideDataSet.
	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7539)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetTableNameW (see page 7539)	This is PSGetTableNameW, a member of class TWideDataSet.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetCommandText (see page 7540)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.

	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadDataPacket (see page 8441)	This is ReadDataPacket, a member of class TCustomClientDataSet.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RefreshInternalCalcFields (see page 8441)	This is RefreshInternalCalcFields, a member of class TCustomClientDataSet.
	ResetAggField (see page 8442)	This is ResetAggField, a member of class TCustomClientDataSet.
	ResetCursorRange (see page 8443)	This is ResetCursorRange, a member of class TCustomClientDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAggregates (see page 8445)	This is SetAggregates, a member of class TCustomClientDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppServer (see page 8446)	This is SetAppServer, a member of class TCustomClientDataSet.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBookmarkData (see page 8446)	This is SetBookmarkData, a member of class TCustomClientDataSet.
	SetBookmarkFlag (see page 8446)	This is SetBookmarkFlag, a member of class TCustomClientDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCommandText (see page 8447)	This is SetCommandText, a member of class TCustomClientDataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetCursorRange (see page 8447)	This is SetCursorRange, a member of class TCustomClientDataSet.
	SetDataSetField (see page 8447)	This is SetDataSetField, a member of class TCustomClientDataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldData (see page 8447)	This is SetFieldData, a member of class TCustomClientDataSet.
	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
	SetFilterData (see page 8448)	This is SetFilterData, a member of class TCustomClientDataSet.
	SetFiltered (see page 8448)	This is SetFiltered, a member of class TCustomClientDataSet.
	SetFilterOptions (see page 8448)	This is SetFilterOptions, a member of class TCustomClientDataSet.
	SetFilterText (see page 8448)	This is SetFilterText, a member of class TCustomClientDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetIndexField (see page 8448)	This is SetIndexField, a member of class TCustomClientDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetKeyBuffer (see page 8449)	This is SetKeyBuffer, a member of class TCustomClientDataSet.
	SetKeyExclusive (see page 8449)	This is SetKeyExclusive, a member of class TCustomClientDataSet.
	SetKeyFieldCount (see page 8450)	This is SetKeyFieldCount, a member of class TCustomClientDataSet.
	SetKeyFields (see page 8450)	This is SetKeyFields, a member of class TCustomClientDataSet.
	SetLinkRanges (see page 8450)	This is SetLinkRanges, a member of class TCustomClientDataSet.
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
	SetOnFilterRecord (see page 8450)	This is SetOnFilterRecord, a member of class TCustomClientDataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetProviderEOF (see page 8452)	This is SetProviderEOF, a member of class TCustomClientDataSet.
	SetRecNo (see page 8454)	This is SetRecNo, a member of class TCustomClientDataSet.
	SetRemoteServer (see page 8454)	This is SetRemoteServer, a member of class TCustomClientDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.

	SwitchToIndex (see page 8455)	This is SwitchToIndex, a member of class TCustomClientDataSet.
	SyncCursors (see page 8455)	This is SyncCursors, a member of class TCustomClientDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateIndexDefs (see page 8455)	This is UpdateIndexDefs, a member of class TCustomClientDataSet.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteDataPacket (see page 8456)	This is WriteDataPacket, a member of class TCustomClientDataSet.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
	Aggregates (see page 8461)	Lists all aggregates that apply to the client dataset.
	AggregatesActive (see page 8461)	Specifies whether the client dataset calculates and maintains aggregate values.
	AutoCalcFields (see page 8463)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
	CommandText (see page 8465)	Specifies an SQL command to be executed by the database server.
	Constraints (see page 8466)	Specifies record-level constraints that must be met when editing the data.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
	DisableStringTrim (see page 8470)	Specifies whether trailing spaces are trimmed from field values when records are posted.
	DSBase (see page 8471)	Specifies the interface the client dataset uses to communicate with classes in Midas.dll or Midaslib.dcu (on Windows) or midas.so (on Linux).
R	DSCursor (see page 8471)	Provides access to the interface for the client dataset's data cursor.
	FetchOnDemand (see page 8471)	Indicates whether to fetch data packets from the provider on an as-needed basis.
	FieldDefs (see page 8472)	Points to the list of field definitions for the dataset.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
	Filter (see page 8476)	Specifies the text of the current filter.
	Filtered (see page 8478)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 8478)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
	IndexDefs (see page 8480)	Contains information about the indexes for a client dataset.
	IndexName (see page 8483)	Identifies an index for the client dataset.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
R	IsClone (see page 8484)	This is IsClone, a member of class TCustomClientDataSet.
	MasterSource (see page 8486)	Specifies a data source component for the master dataset when establishing a detail-master relationship between this dataset and another one.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.

	ObjectView (see page 8487)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	Params (see page 8489)	Contains parameter values that are sent to the provider.
	ProviderEOF (see page 8491)	Indicates whether the client dataset has all of its data.
R	Ranged (see page 8492)	This is Ranged, a member of class TCustomClientDataSet.
	ReadOnly (see page 8493)	Specifies whether the client dataset is read-only for this application.
	Reserved (see page 7035)	For internal use only.
	StoreDefs (see page 8495)	Indicates whether field and index definitions are saved with the client dataset.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
	AddIndex (see page 8391)	Creates a new index for the client dataset.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendData (see page 8393)	Adds a newly fetched packet of records to those already stored in the client dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
	ApplyRange (see page 8393)	Applies a range to the dataset.
V	ApplyUpdates (see page 8394)	Sends all updated, inserted, and deleted records from the client dataset to the provider for writing to the database.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 8395)	Tests the validity of a specified bookmark.
V	Cancel (see page 8395)	Cancels unposted changes to the current record.
	CancelRange (see page 8396)	Removes any ranges currently in effect for the client dataset.
	CancelUpdates (see page 8396)	Clears all pending edits and clears the change log.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
V	CloneCursor (see page 8398)	Shares the data belonging to another client dataset.
	Close (see page 6920)	Closes a dataset.
V	CompareBookmarks (see page 8400)	Compares two bookmarks.
	ConstraintsDisabled (see page 8400)	Indicates whether constraints are enabled for a client dataset.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
V	Create (see page 8386)	Creates and initializes instance of TCustomClientDataSet (see page 8373).
V	CreateBlobStream (see page 8401)	Returns a TClientBlobStream (see page 8285) object for reading or writing the data in a specified blob field.
	CreateDataSet (see page 8401)	Creates a new, empty client dataset.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
V	DataRequest (see page 8404)	Generates an OnDataRequest event on the associated provider
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
	DeleteIndex (see page 8406)	Deletes an index for the client dataset.
V	Destroy (see page 8387)	Destroys the instance of a client dataset component.

	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableConstraints (see page 8406)	Disables the enforcement of constraints.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EditKey (see page 8410)	Enables modification of the search key buffer.
	EditRangeEnd (see page 8411)	Enables changing the ending value for an existing range.
	EditRangeStart (see page 8411)	Enables changing the starting value for an existing range.
V	EmptyDataSet (see page 8412)	Removes all records from the dataset.
	EnableConstraints (see page 8412)	Reenables enforcement of constraints when records are posted.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
V	Execute (see page 8413)	Executes an SQL command by the provider's dataset.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FetchBlobs (see page 8414)	Retrieves BLOB field values from the provider.
	FetchDetails (see page 8414)	Retrieves nested detail datasets from the provider.
	FetchParams (see page 8415)	Retrieves parameter values from the provider.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
V	FindKey (see page 8416)	Searches for a record containing specified field values.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNearest (see page 8416)	Moves the cursor to the record that most closely matches a specified set of key values.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
V	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
V	GetCurrentRecord (see page 8419)	Retrieves the current record into a buffer.
V	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
V	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
V	GetFieldData (see page 8419)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
V	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetGroupState (see page 8420)	Indicates where the current record sits within a specified group of records.
	GetIndexInfo (see page 8421)	Retrieves information about the current index.
	GetIndexNames (see page 8422)	Retrieves a list of available indexes for a client dataset.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetNextPacket (see page 8423)	Retrieves the next block of records from the provider.
	GetOptionalParam (see page 8425)	Retrieves a custom parameter from the data packet.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	GotoCurrent (see page 8427)	Sets the current record in this dataset the same as the current record in another, cloned dataset.
	GotoKey (see page 8427)	Moves the cursor to a record specified by the current key.
	GotoNearest (see page 8427)	Moves the cursor to the record that most closely matches the current key.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
V	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	Last (see page 6968)	Moves to the last record in the dataset.
	LoadFromFile (see page 8433)	Loads a client dataset's data from a file.
	LoadFromStream (see page 8435)	Loads a client dataset's data from a stream.
V	Locate (see page 8435)	Searches the dataset for a specified record and makes that record current.
V	Lookup (see page 8436)	Retrieves field values from a record that matches specified search values.
	MergeChangeLog (see page 8437)	Merges any data changes in the change log into the client dataset, overwriting existing values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	Open (see page 6976)	Opens the dataset.
V	Post (see page 8438)	Writes a modified record to the Data (see page 8467) property or the change log.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSEndTransaction (see page 7589)	Ends a transaction that was started by a call to PSStartTransaction (see page 7597).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecute (see page 7590)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSExecuteStatement (see page 7590)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 7591)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.

	PSGetDefaultOrder (see page 7592)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7593)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetParams (see page 7593)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7594)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7594)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSGetUpdateException (see page 7594)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 7595)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 7595)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 7596)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 7596)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7585)	Assigns a specified SQL command to be executed by PSExecute (see page 7578).
	PSSetCommandText (see page 7596)	Assigns a specified SQL command to be executed by PSExecute (see page 7590).
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 7597)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).
	PSStartTransaction (see page 7597)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7589).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 7598)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Reconcile (see page 8441)	Clears successfully updated records from the client dataset's cache of changes.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RefreshRecord (see page 8442)	Updates the base values of the current record to match the current values of the provider's dataset.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
	RevertRecord (see page 8443)	Undoes edits to the current record.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 8443)	Saves a client dataset's data to an external file.
	SaveToStream (see page 8445)	Saves a client dataset's data to a stream.
	SetAltRecBuffers (see page 8446)	Sets record buffers when accessing field values.
	SetFields (see page 6997)	Sets the values for all fields in a record

	SetKey (see page 8449)	Enables setting of keys and ranges for a dataset prior to a search.
V	SetOptionalParam (see page 8450)	Saves a custom parameter with the data packet.
	SetProvider (see page 8451)	Associates the client dataset with a provider in the same process space.
	SetRange (see page 8452)	Sets the starting and ending values of a range, and applies it.
	SetRangeEnd (see page 8453)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
	SetRangeStart (see page 8454)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
	UndoLastChange (see page 8455)	Undoes the last edit, insert, or delete operation to a record in the client dataset.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 8456)	Reports the update status for the current record in the dataset.

Public Properties

	Name	Description
	Active (see page 8458)	Specifies whether the client dataset contains data.
R	ActiveAggs (see page 8460)	Lists the maintained aggregates supported by the current index.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AppServer (see page 8462)	Provides access to the interface the client dataset uses to communicate with a provider.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ChangeCount (see page 8463)	Indicates the number of changes in the change log.
R	CloneSource (see page 8464)	Indicates the client dataset (if any) with which this client dataset shares data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	ConnectionBroker (see page 8466)	Specifies a connection broker that handles the connection to an application server.
	Data (see page 8467)	Represents the data of the client dataset in a transportable format.
	DataSetField (see page 8469)	Indicates the TDataSetField component that represents this client dataset.
R	DataSize (see page 8469)	Indicates the number of bytes required to represent the Data (see page 8467) property.
R	DataSource (see page 7014)	Represents the data source of another dataset that supplies values to the dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Delta (see page 8470)	Represents a packet of change log data.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset

	FileName (see page 8475)	Specifies a file that stores the data of the client dataset.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
R	GroupingLevel (see page 8479)	Indicates the depth of grouping support provided by the current index.
R	HasAppServer (see page 8480)	Indicates whether the client dataset is connected to a provider.
R	IndexFieldCount (see page 8481)	Indicates the number of fields that make up the current index.
	IndexFieldNames (see page 8482)	Lists the fields to use as an index.
	IndexFields (see page 8482)	Specifies the fields associated with the current index.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
	KeyExclusive (see page 8484)	Specifies the upper and lower boundaries for a range.
	KeyFieldCount (see page 8484)	Specifies the number of fields to use when conducting a partial key search on a multi-field key.
R	KeySize (see page 8485)	Indicates the size of the current key used for the dataset.
	LogChanges (see page 8485)	Indicates whether to maintain a change log separate from the data.
	MasterFields (see page 8486)	Names one or more fields in a master table to link with corresponding fields in this dataset in order to establish a master-detail relationship.
R	Modified (see page 7029)	Indicates whether the active record is modified.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	PacketRecords (see page 8488)	Indicates the number or type of records in a single data packet.
	ProviderName (see page 8492)	Specifies the name of a provider component that supplies data to and applies updates from the client dataset..
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
	RemoteServer (see page 8493)	Specifies the connection component that this client dataset uses to connect to an application server.
	SavePoint (see page 8494)	Represents the current state of edits when logging changes.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	StatusFilter (see page 8495)	Filters the client dataset so that it exposes only those records with a specific update status.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	XMLData (see page 8496)	Represents the data of the client dataset in XML.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

45.1.8.2 TCustomClientDataSet.Create Constructor

Creates and initializes instance of TCustomClientDataSet (see page 8373).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomClientDataSet(TComponent * AOwner);
```

Description

Call Create to instantiate a client dataset component at runtime. Client datasets added to forms or data modules at design time are created automatically.

AOwner is a component, typically a form or data module, that is responsible for freeing the client dataset.

See Also

Destroy (see page 8387)

Owner (see page 2807)

45.1.8.3 TCustomClientDataSet.Destroy Destructor

Destroys the instance of a client dataset component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomClientDataSet();
```

Description

Do not call Destroy directly in an application. Usually destruction of datasets is handled automatically. If an application creates its own instances of a dataset without assigning an Owner, however, it should call Free, which verifies that the dataset component is not nil before calling Destroy.

See Also

Owner (see page 2807)

Create (see page 8386)

Free (see page 22300)

45.1.8.4 TCustomClientDataSet Methods

The methods of the TCustomClientDataSet class are listed here.

Protected Methods

	Name	Description
	ActivateFilters (see page 8391)	This is ActivateFilters, a member of class TCustomClientDataSet.
V	AddDataPacket (see page 8391)	This is AddDataPacket, a member of class TCustomClientDataSet.
	AddFieldDesc (see page 8391)	This is AddFieldDesc, a member of class TCustomClientDataSet.
	AllocKeyBuffers (see page 8392)	This is AllocKeyBuffers, a member of class TCustomClientDataSet.
V	AllocRecordBuffer (see page 8393)	This is AllocRecordBuffer, a member of class TCustomClientDataSet.
	Check (see page 8397)	This is Check, a member of class TCustomClientDataSet.
V	CheckDetailRecords (see page 8397)	This is CheckDetailRecords, a member of class TCustomClientDataSet.
V	CheckProviderEOF (see page 8397)	This is CheckProviderEOF, a member of class TCustomClientDataSet.
	CheckSetKeyMode (see page 8397)	This is CheckSetKeyMode, a member of class TCustomClientDataSet.
V	ClearCalcFields (see page 8398)	This is ClearCalcFields, a member of class TCustomClientDataSet.
V	CloseCursor (see page 8400)	This is CloseCursor, a member of class TCustomClientDataSet.
	ConstraintsStored (see page 8401)	This is ConstraintsStored, a member of class TCustomClientDataSet.
V	DataConvert (see page 8404)	This is DataConvert, a member of class TCustomClientDataSet.
V	DataEvent (see page 8404)	This is DataEvent, a member of class TCustomClientDataSet.
	DeactivateFilters (see page 8405)	This is DeactivateFilters, a member of class TCustomClientDataSet.
V	DefChanged (see page 8405)	This is DefChanged, a member of class TCustomClientDataSet.
V	DefineProperties (see page 8405)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DestroyLookupCursor (see page 8406)	This is DestroyLookupCursor, a member of class TCustomClientDataSet.
V	DoAfterApplyUpdates (see page 8407)	This is DoAfterApplyUpdates, a member of class TCustomClientDataSet.

	DoAfterExecute (see page 8407)	This is DoAfterExecute, a member of class TCustomClientDataSet.
	DoAfterGetParams (see page 8407)	This is DoAfterGetParams, a member of class TCustomClientDataSet.
	DoAfterGetRecords (see page 8407)	This is DoAfterGetRecords, a member of class TCustomClientDataSet.
	DoAfterRowRequest (see page 8408)	This is DoAfterRowRequest, a member of class TCustomClientDataSet.
	DoApplyUpdates (see page 8408)	This is DoApplyUpdates, a member of class TCustomClientDataSet.
	DoBeforeApplyUpdates (see page 8408)	This is DoBeforeApplyUpdates, a member of class TCustomClientDataSet.
	DoBeforeExecute (see page 8408)	This is DoBeforeExecute, a member of class TCustomClientDataSet.
	DoBeforeGetParams (see page 8408)	This is DoBeforeGetParams, a member of class TCustomClientDataSet.
	DoBeforeGetRecords (see page 8409)	This is DoBeforeGetRecords, a member of class TCustomClientDataSet.
	DoBeforeInsert (see page 8409)	This is DoBeforeInsert, a member of class TCustomClientDataSet.
	DoBeforeRowRequest (see page 8409)	This is DoBeforeRowRequest, a member of class TCustomClientDataSet.
	DoExecute (see page 8409)	This is DoExecute, a member of class TCustomClientDataSet.
	DoGetRecords (see page 8410)	This is DoGetRecords, a member of class TCustomClientDataSet.
	DoOnNewRecord (see page 8410)	This is DoOnNewRecord, a member of class TCustomClientDataSet.
	DoRowRequest (see page 8410)	This is DoRowRequest, a member of class TCustomClientDataSet.
	FetchMoreData (see page 8415)	This is FetchMoreData, a member of class TCustomClientDataSet.
	FindRecord (see page 8417)	This is FindRecord, a member of class TCustomClientDataSet.
	FreeKeyBuffers (see page 8417)	This is FreeKeyBuffers, a member of class TCustomClientDataSet.
	FreeRecordBuffer (see page 8417)	This is FreeRecordBuffer, a member of class TCustomClientDataSet.
	GetAggregateValue (see page 8418)	This is GetAggregateValue, a member of class TCustomClientDataSet.
	GetAppServer (see page 8418)	This is GetAppServer, a member of class TCustomClientDataSet.
	GetBookmarkData (see page 8418)	This is GetBookmarkData, a member of class TCustomClientDataSet.
	GetBookmarkFlag (see page 8418)	This is GetBookmarkFlag, a member of class TCustomClientDataSet.
	GetCanModify (see page 8418)	This is GetCanModify, a member of class TCustomClientDataSet.
	GetDataSource (see page 8419)	This is GetDataSource, a member of class TCustomClientDataSet.
	GetIndexField (see page 8421)	This is GetIndexField, a member of class TCustomClientDataSet.
	GetIndexFieldCount (see page 8421)	This is GetIndexFieldCount, a member of class TCustomClientDataSet.
	GetIsClone (see page 8422)	This is GetIsClone, a member of class TCustomClientDataSet.
	GetIsIndexField (see page 8422)	This is GetIsIndexField, a member of class TCustomClientDataSet.
	GetKeyBuffer (see page 8423)	This is GetKeyBuffer, a member of class TCustomClientDataSet.
	GetKeyExclusive (see page 8423)	This is GetKeyExclusive, a member of class TCustomClientDataSet.
	GetKeyFieldCount (see page 8423)	This is GetKeyFieldCount, a member of class TCustomClientDataSet.
	GetRecNo (see page 8425)	This is GetRecNo, a member of class TCustomClientDataSet.
	GetRecord (see page 8425)	This is GetRecord, a member of class TCustomClientDataSet.
	GetRecordCount (see page 8426)	This is GetRecordCount, a member of class TCustomClientDataSet.
	GetRecordSize (see page 8426)	This is GetRecordSize, a member of class TCustomClientDataSet.
	GetRemoteServer (see page 8426)	This is GetRemoteServer, a member of class TCustomClientDataSet.
	GetStateFieldValue (see page 8426)	This is GetStateFieldValue, a member of class TCustomClientDataSet.
	InitKeyBuffer (see page 8428)	This is InitKeyBuffer, a member of class TCustomClientDataSet.
	InitRecord (see page 8428)	This is InitRecord, a member of class TCustomClientDataSet.
	InternalAddRecord (see page 8428)	This is InternalAddRecord, a member of class TCustomClientDataSet.
	InternalCancel (see page 8429)	This is InternalCancel, a member of class TCustomClientDataSet.
	InternalClose (see page 8429)	This is InternalClose, a member of class TCustomClientDataSet.
	InternalDelete (see page 8429)	This is InternalDelete, a member of class TCustomClientDataSet.
	InternalEdit (see page 8429)	This is InternalEdit, a member of class TCustomClientDataSet.
	InternalFirst (see page 8430)	This is InternalFirst, a member of class TCustomClientDataSet.
	InternalGetOptionalParam (see page 8430)	This is InternalGetOptionalParam, a member of class TCustomClientDataSet.
	InternalGotoBookmark (see page 8430)	This is InternalGotoBookmark, a member of class TCustomClientDataSet.
	InternalHandleException (see page 8430)	This is InternalHandleException, a member of class TCustomClientDataSet.
	InternalInitFieldDefs (see page 8430)	This is InternalInitFieldDefs, a member of class TCustomClientDataSet.
	InternalInitRecord (see page 8431)	This is InternalInitRecord, a member of class TCustomClientDataSet.
	InternalInsert (see page 8431)	This is InternalInsert, a member of class TCustomClientDataSet.
	InternalLast (see page 8431)	This is InternalLast, a member of class TCustomClientDataSet.
	InternalOpen (see page 8431)	This is InternalOpen, a member of class TCustomClientDataSet.

	InternalPost (see page 8432)	This is InternalPost, a member of class TCustomClientDataSet.
	InternalRefresh (see page 8432)	This is InternalRefresh, a member of class TCustomClientDataSet.
	InternalSetOptionalParam (see page 8432)	This is InternalSetOptionalParam, a member of class TCustomClientDataSet.
	InternalSetToRecord (see page 8432)	This is InternalSetToRecord, a member of class TCustomClientDataSet.
	IsCursorOpen (see page 8432)	This is IsCursorOpen, a member of class TCustomClientDataSet.
	Loaded (see page 8433)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 8436)	This is LocateRecord, a member of class TCustomClientDataSet.
	Notification (see page 8437)	Forwards notification messages to all owned components.
	OpenCursor (see page 8438)	This is OpenCursor, a member of class TCustomClientDataSet.
	PostKeyBuffer (see page 8439)	This is PostKeyBuffer, a member of class TCustomClientDataSet.
	PSGetCommandText (see page 8439)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandType (see page 8440)	Gets the command type to be executed by PSExecute.
	PSGetIndexDefs (see page 8440)	Returns the definitions of all specified indexes defined for the dataset.
	ReadDataPacket (see page 8441)	This is ReadDataPacket, a member of class TCustomClientDataSet.
	RefreshInternalCalcFields (see page 8441)	This is RefreshInternalCalcFields, a member of class TCustomClientDataSet.
	ResetAggField (see page 8442)	This is ResetAggField, a member of class TCustomClientDataSet.
	ResetCursorRange (see page 8443)	This is ResetCursorRange, a member of class TCustomClientDataSet.
	SetAggregates (see page 8445)	This is SetAggregates, a member of class TCustomClientDataSet.
	SetAppServer (see page 8446)	This is SetAppServer, a member of class TCustomClientDataSet.
	SetBookmarkData (see page 8446)	This is SetBookmarkData, a member of class TCustomClientDataSet.
	SetBookmarkFlag (see page 8446)	This is SetBookmarkFlag, a member of class TCustomClientDataSet.
	SetCommandText (see page 8447)	This is SetCommandText, a member of class TCustomClientDataSet.
	SetCursorRange (see page 8447)	This is SetCursorRange, a member of class TCustomClientDataSet.
	SetDataSetField (see page 8447)	This is SetDataSetField, a member of class TCustomClientDataSet.
	SetFieldData (see page 8447)	This is SetFieldData, a member of class TCustomClientDataSet.
	SetFilterData (see page 8448)	This is SetFilterData, a member of class TCustomClientDataSet.
	SetFiltered (see page 8448)	This is SetFiltered, a member of class TCustomClientDataSet.
	SetFilterOptions (see page 8448)	This is SetFilterOptions, a member of class TCustomClientDataSet.
	SetFilterText (see page 8448)	This is SetFilterText, a member of class TCustomClientDataSet.
	SetIndexField (see page 8448)	This is SetIndexField, a member of class TCustomClientDataSet.
	SetKeyBuffer (see page 8449)	This is SetKeyBuffer, a member of class TCustomClientDataSet.
	SetKeyExclusive (see page 8449)	This is SetKeyExclusive, a member of class TCustomClientDataSet.
	SetKeyFieldCount (see page 8450)	This is SetKeyFieldCount, a member of class TCustomClientDataSet.
	SetKeyFields (see page 8450)	This is SetKeyFields, a member of class TCustomClientDataSet.
	SetLinkRanges (see page 8450)	This is SetLinkRanges, a member of class TCustomClientDataSet.
	SetOnFilterRecord (see page 8450)	This is SetOnFilterRecord, a member of class TCustomClientDataSet.
	SetProviderEOF (see page 8452)	This is SetProviderEOF, a member of class TCustomClientDataSet.
	SetRecNo (see page 8454)	This is SetRecNo, a member of class TCustomClientDataSet.
	SetRemoteServer (see page 8454)	This is SetRemoteServer, a member of class TCustomClientDataSet.
	SwitchToIndex (see page 8455)	This is SwitchToIndex, a member of class TCustomClientDataSet.
	SyncCursors (see page 8455)	This is SyncCursors, a member of class TCustomClientDataSet.
	UpdateIndexDefs (see page 8455)	This is UpdateIndexDefs, a member of class TCustomClientDataSet.
	WriteDataPacket (see page 8456)	This is WriteDataPacket, a member of class TCustomClientDataSet.

Public Methods

	Name	Description
	AddIndex (see page 8391)	Creates a new index for the client dataset.
	AppendData (see page 8393)	Adds a newly fetched packet of records to those already stored in the client dataset.
	ApplyRange (see page 8393)	Applies a range to the dataset.
	ApplyUpdates (see page 8394)	Sends all updated, inserted, and deleted records from the client dataset to the provider for writing to the database.
	BookmarkValid (see page 8395)	Tests the validity of a specified bookmark.
	Cancel (see page 8395)	Cancels unposted changes to the current record.
	CancelRange (see page 8396)	Removes any ranges currently in effect for the client dataset.

	CancelUpdates (see page 8396)	Clears all pending edits and clears the change log.
V	CloneCursor (see page 8398)	Shares the data belonging to another client dataset.
V	CompareBookmarks (see page 8400)	Compares two bookmarks.
	ConstraintsDisabled (see page 8400)	Indicates whether constraints are enabled for a client dataset.
V	CreateBlobStream (see page 8401)	Returns a TClientBlobStream (see page 8285) object for reading or writing the data in a specified blob field.
	CreateDataSet (see page 8401)	Creates a new, empty client dataset.
V	DataRequest (see page 8404)	Generates an OnDataRequest event on the associated provider
	DeleteIndex (see page 8406)	Deletes an index for the client dataset.
	DisableConstraints (see page 8406)	Disables the enforcement of constraints.
	EditKey (see page 8410)	Enables modification of the search key buffer.
	EditRangeEnd (see page 8411)	Enables changing the ending value for an existing range.
	EditRangeStart (see page 8411)	Enables changing the starting value for an existing range.
V	EmptyDataSet (see page 8412)	Removes all records from the dataset.
	EnableConstraints (see page 8412)	Reenables enforcement of constraints when records are posted.
V	Execute (see page 8413)	Executes an SQL command by the provider's dataset.
	FetchBlobs (see page 8414)	Retrieves BLOB field values from the provider.
	FetchDetails (see page 8414)	Retrieves nested detail datasets from the provider.
	FetchParams (see page 8415)	Retrieves parameter values from the provider.
V	FindKey (see page 8416)	Searches for a record containing specified field values.
	FindNearest (see page 8416)	Moves the cursor to the record that most closely matches a specified set of key values.
V	GetCurrentRecord (see page 8419)	Retrieves the current record into a buffer.
V	GetFieldData (see page 8419)	This is the overview for the GetFieldData method overload.
	GetGroupState (see page 8420)	Indicates where the current record sits within a specified group of records.
	GetIndexInfo (see page 8421)	Retrieves information about the current index.
	GetIndexNames (see page 8422)	Retrieves a list of available indexes for a client dataset.
V	GetNextPacket (see page 8423)	Retrieves the next block of records from the provider.
	GetOptionalParam (see page 8425)	Retrieves a custom parameter from the data packet.
	GotoCurrent (see page 8427)	Sets the current record in this dataset the same as the current record in another, cloned dataset.
	GotoKey (see page 8427)	Moves the cursor to a record specified by the current key.
	GotoNearest (see page 8427)	Moves the cursor to the record that most closely matches the current key.
	LoadFromFile (see page 8433)	Loads a client dataset's data from a file.
	LoadFromStream (see page 8435)	Loads a client dataset's data from a stream.
V	Locate (see page 8435)	Searches the dataset for a specified record and makes that record current.
V	Lookup (see page 8436)	Retrieves field values from a record that matches specified search values.
	MergeChangeLog (see page 8437)	Merges any data changes in the change log into the client dataset, overwriting existing values.
V	Post (see page 8438)	Writes a modified record to the Data (see page 8467) property or the change log.
	Reconcile (see page 8441)	Clears successfully updated records from the client dataset's cache of changes.
	RefreshRecord (see page 8442)	Updates the base values of the current record to match the current values of the provider's dataset.
	RevertRecord (see page 8443)	Undoes edits to the current record.
	SaveToFile (see page 8443)	Saves a client dataset's data to an external file.
	SaveToStream (see page 8445)	Saves a client dataset's data to a stream.
	SetAltRecBuffers (see page 8446)	Sets record buffers when accessing field values.
	SetKey (see page 8449)	Enables setting of keys and ranges for a dataset prior to a search.
V	SetOptionalParam (see page 8450)	Saves a custom parameter with the data packet.
	SetProvider (see page 8451)	Associates the client dataset with a provider in the same process space.
	SetRange (see page 8452)	Sets the starting and ending values of a range, and applies it.
	SetRangeEnd (see page 8453)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
	SetRangeStart (see page 8454)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
	UndoLastChange (see page 8455)	Undoes the last edit, insert, or delete operation to a record in the client dataset.

	UpdateStatus (see page 8456)	Reports the update status for the current record in the dataset.

45.1.8.4.1 TCustomClientDataSet.ActivateFilters Method

Pascal

```
procedure ActivateFilters;
```

C++

```
__fastcall ActivateFilters();
```

Description

This is ActivateFilters, a member of class TCustomClientDataSet.

45.1.8.4.2 TCustomClientDataSet.AddDataPacket Method

Pascal

```
procedure AddDataPacket(const Data: OleVariant; HitEOF: Boolean); virtual;
```

C++

```
virtual __fastcall AddDataPacket(const OleVariant Data, Boolean HitEOF);
```

Description

This is AddDataPacket, a member of class TCustomClientDataSet.

45.1.8.4.3 TCustomClientDataSet.AddFieldDesc Method

Pascal

```
procedure AddFieldDesc(FieldDescs: TFieldDescList; var DescNo: Integer; var FieldID: Integer;
FieldDefs: TFieldDefs);
```

C++

```
__fastcall AddFieldDesc(TFieldDescList FieldDescs, int DescNo, int FieldID, TFieldDefs
FieldDefs);
```

Description

This is AddFieldDesc, a member of class TCustomClientDataSet.

45.1.8.4.4 TCustomClientDataSet.AddIndex Method

Creates a new index for the client dataset.

Pascal

```
procedure AddIndex(const Name: string; const Fields: string; Options: TIndexOptions; const
DescFields: string = ''; const CaseInsFields: string = ''; const GroupingLevel: Integer = 0);
```

C++

```
__fastcall AddIndex(const AnsiString Name, const AnsiString Fields, TIndexOptions Options,
const AnsiString DescFields = '', const AnsiString CaseInsFields = '', const int GroupingLevel
= 0);
```

Description

Call AddIndex to create a new index for the client dataset.

Name is the name of the new index.

Fields is a semicolon-delimited list of the fields to include in the index.

Options is a (potentially empty) set that can include up to two TIndexOptions values. The valid members for that set include ixDescending (specifying that the index sorts in descending alphanumeric order) and ixCaseInsensitive (specifying that the index should ignore case when sorting).

Warning: AddIndex raises an exception if you attempt to create an index that includes any other TIndexOptions values .

DescFields is a semicolon-delimited list of field names. Use DescFields instead of an Options value that includes ixDescending to create an index that is in ascending order on some fields and descending order on others. All fields named in DescFields are sorted in descending order.

CaseInsFields is a semicolon-delimited list of field names. Use CaseInsFields instead of an Options value that includes ixCaseInsensitive to create an index that is case-insensitive on some fields and case-sensitive on others. All fields named in CaseInsFields are sorted without regard to case.

GroupingLevel (see page 8479) is the default grouping level supported by the index. When the client dataset uses this index, grouping information is automatically available for subgroups defined by up to GroupingLevel (see page 8479) fields in the index. (If the client dataset uses maintained aggregates, information about more deeply nested groups can become available when the aggregates are defined on a grouping level greater than the default value of the index). GroupingLevel (see page 8479) can be any value from 0 to the number of fields in the index.

Note: Indexes added by AddIndex are not saved with data when using SaveToFile (see page 8443). They do not persist when the client dataset is closed. (That is, they are lost when you reopen the client dataset). You can't call AddIndex when the dataset is closed.

Tip: You can support maintained aggregates using an index added with AddIndex.

See Also

DeleteIndex (see page 8406)

IndexDefs (see page 8480)

IndexName (see page 8483)

DescFields (see page 7288)

CaseInsFields (see page 7287)

GroupingLevel (see page 7291)

GroupingLevel (see page 8479)

Adding a New Index

45.1.8.4.5 TCustomClientDataSet.AllocKeyBuffers Method

Pascal

```
procedure AllocKeyBuffers;
```

C++

```
__fastcall AllocKeyBuffers();
```

Description

This is AllocKeyBuffers, a member of class TCustomClientDataSet.

45.1.8.4.6 TCustomClientDataSet.AllocRecordBuffer Method

Pascal

```
function AllocRecordBuffer: PChar; override;
```

C++

```
virtual __fastcall const char * AllocRecordBuffer();
```

Description

This is AllocRecordBuffer, a member of class TCustomClientDataSet.

45.1.8.4.7 TCustomClientDataSet.AppendData Method

Adds a newly fetched packet of records to those already stored in the client dataset.

Pascal

```
procedure AppendData(const Data: OleVariant; HitEOF: Boolean);
```

C++

```
__fastcall AppendData(const OleVariant Data, Boolean HitEOF);
```

Description

Use AppendData to add a newly fetched packet of records from a provider to the client dataset. Data (see page 8467) indicates the packet of records to add. HitEOF indicates whether the provider encountered the end of the dataset when it fetched the records from the database server.

Most applications call GetNextPacket (see page 8423) instead to fetch a packet of records and automatically add them to those already available. Use AppendData when fetching data packets directly from the provider rather than letting the client dataset fetch data.

See Also

GetNextPacket (see page 8423)

TDataSetProvider (see page 18508)

45.1.8.4.8 TCustomClientDataSet.ApplyRange Method

Applies a range to the dataset.

Pascal

```
procedure ApplyRange;
```

C++

```
__fastcall ApplyRange();
```

Description

Call ApplyRange to cause a range established with SetRangeStart (see page 8454) and SetRangeEnd (see page 8453), or EditRangeStart (see page 8411) and EditRangeEnd (see page 8411), to take effect. When a range is in effect, only those records that fall within the range are available to the application for viewing and editing.

After a call to ApplyRange, the cursor is left on the first record in the range.

See Also

CancelRange (see page 8396)

EditRangeEnd (see page 8411)

EditRangeStart (see page 8411)

SetRange (see page 8452)

SetRangeEnd (see page 8453)

SetRangeStart (see page 8454)

Applying or Canceling a Range

Modifying a Range

45.1.8.4.9 TCustomClientDataSet.ApplyUpdates Method

Sends all updated, inserted, and deleted records from the client dataset to the provider for writing to the database.

Pascal

```
function ApplyUpdates(MaxErrors: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int ApplyUpdates(int MaxErrors);
```

Description

Call ApplyUpdates to write all updated, inserted, and deleted records from the client dataset to the database or source dataset. MaxErrors indicates the maximum number of errors that the provider should allow before prematurely stopping the update operation. Set MaxErrors to –1 to indicate that there is no limit to the number of errors.

ApplyUpdates

1.Generates a BeforeApplyUpdates (see page 8511) event. (This event may not be public on some TCustomClientDataSet (see page 8373) descendants.)

2.Calls the provider to apply the updates in the Delta (see page 8470) property and receives any records returned by the provider because they generated errors when it attempted to apply them to the database.

3.Generates an AfterApplyUpdates (see page 8497) event. (This event may not be public on some TCustomClientDataSet (see page 8373) descendants.)

4.Calls the client dataset's Reconcile (see page 8441) method to reconcile any records that are returned in step 2.

Warning: The provider can't detect when an update conflicts with another user's changes to a memo field. The server will not detect and return these conflicts as errors.

ApplyUpdates returns the number of errors it encountered. Based on this return value and the setting of MaxErrors, successfully applied records are removed from the client dataset's change log. If the update process is aborted before all updates are applied, any unapplied updates remain in the change log.

Delphi Examples:

```
{
This BeforeDisconnect event handler makes sure that the
client dataset has applied all its pending updates before
terminating the connection to the server.
}
procedure TForm1.RemoteServer1BeforeDisconnect(Sender: TObject);
begin
  if (ClientDataSet1.ChangeCount > 0) then
    ClientDataSet1.ApplyUpdates(-1);
end;
```

See Also

Reconcile (see page 8441)

AS_ApplyUpdates (see page 16451)

BeforeApplyUpdates (see page 8511)

AfterApplyUpdates (see page 8497)

Delta (see page 8470)

Applying Updates

Setting Up a Simple Dataset

Using the BDE to Cache Updates

Connecting a Client Dataset to Another Dataset in the Same Application

Communicating with the Client Dataset

Updating Data in dbExpress Applications

45.1.8.4.10 TCustomClientDataSet.BookmarkValid Method

Tests the validity of a specified bookmark.

Pascal

```
function BookmarkValid(Bookmark: TBookmark): Boolean; override;
```

C++

```
virtual __fastcall Boolean BookmarkValid(TBookmark Bookmark);
```

Description

Call BookmarkValid to determine if a bookmark is valid. Bookmark is the bookmark pointer to test. If the bookmark is valid, BookmarkValid returns true. Otherwise it returns false.

See Also

CompareBookmarks (see page 8400)

45.1.8.4.11 TCustomClientDataSet.Cancel Method

Cancels unposted changes to the current record.

Pascal

```
procedure Cancel; override;
```

C++

```
virtual __fastcall Cancel();
```

Description

Call Cancel to undo modifications made to one or more fields belonging to the current record. As long as those changes are not already posted to the change log, Cancel returns the record to its previous state, and sets the dataset state to dsBrowse.

Note: Do not confuse Cancel with the RevertRecord (see page 8443) method, which also backs out changes to the current record. Cancel backs out unposted changes while RevertRecord (see page 8443) removes changes that have already been posted to the change log.

See Also

Post (see page 8438)

State (see page 7036)

CancelUpdates (see page 8396)

RevertRecord (see page 8443)

UndoLastChange (see page 8455)

LogChanges (see page 8485)

45.1.8.4.12 TCustomClientDataSet.CancelRange Method

Removes any ranges currently in effect for the client dataset.

Pascal

```
procedure CancelRange;
```

C++

```
__fastcall CancelRange();
```

Description

Call CancelRange to remove a range currently applied to a client dataset. Canceling a range reenables access to all records in the dataset.

See Also

ApplyRange (see page 8393)

EditRangeEnd (see page 8411)

EditRangeStart (see page 8411)

SetRange (see page 8452)

SetRangeEnd (see page 8453)

SetRangeStart (see page 8454)

Applying or Canceling a Range

45.1.8.4.13 TCustomClientDataSet.CancelUpdates Method

Clears all pending edits and clears the change log.

Pascal

```
procedure CancelUpdates;
```

C++

```
__fastcall CancelUpdates();
```

Description

Call CancelUpdates to back out all pending updates from the Delta (see page 8470) property. Delta (see page 8470) is a Variant maintained by the client dataset to record updates, inserts, and deletes.

See Also

ApplyUpdates (see page 8394)

Delta (see page 8470)

RevertRecord (see page 8443)

UndoLastChange (see page 8455)

SavePoint (see page 8494)

Overview of Using Cached Updates

Undoing Changes

Using the BDE to Cache Updates

Updating Data in dbExpress Applications

45.1.8.4.14 TCustomClientDataSet.Check Method

Pascal

```
procedure Check(Status: DBResult);
```

C++

```
__fastcall Check(DBResult Status);
```

Description

This is Check, a member of class TCustomClientDataSet.

45.1.8.4.15 TCustomClientDataSet.CheckDetailRecords Method

Pascal

```
procedure CheckDetailRecords; virtual;
```

C++

```
virtual __fastcall CheckDetailRecords();
```

Description

This is CheckDetailRecords, a member of class TCustomClientDataSet.

45.1.8.4.16 TCustomClientDataSet.CheckProviderEOF Method

Pascal

```
procedure CheckProviderEOF; virtual;
```

C++

```
virtual __fastcall CheckProviderEOF();
```

Description

This is CheckProviderEOF, a member of class TCustomClientDataSet.

45.1.8.4.17 TCustomClientDataSet.CheckSetKeyMode Method

Pascal

```
procedure CheckSetKeyMode;
```

C++

```
__fastcall CheckSetKeyMode();
```

Description

This is CheckSetKeyMode, a member of class TCustomClientDataSet.

45.1.8.4.18 TCustomClientDataSet.ClearCalcFields Method

Pascal

```
procedure ClearCalcFields(Buffer: PChar); override;
```

C++

```
virtual __fastcall ClearCalcFields(const char * Buffer);
```

Description

This is ClearCalcFields, a member of class TCustomClientDataSet.

45.1.8.4.19 TCustomClientDataSet.CloneCursor Method

Shares the data belonging to another client dataset.

Pascal

```
procedure CloneCursor(Source: TCustomClientDataSet; Reset: Boolean; KeepSettings: Boolean =
False); virtual;
```

C++

```
virtual __fastcall CloneCursor(TCustomClientDataSet Source, Boolean Reset, Boolean
KeepSettings = False);
```

Description

Call CloneCursor to share the data belonging to another client dataset. Source indicates another client dataset whose data is to be shared.

Reset and KeepSettings determine how to set the values of the following properties and events:

Filter (see page 8476), Filtered (see page 8478), FilterOptions (see page 8478), and OnFilterRecord (see page 8523)

IndexName (see page 8483)

MasterSource (see page 8486) and MasterFields (see page 8486)

ReadOnly (see page 8493)

RemoteServer (see page 8493) and ProviderName (see page 8492)

If Reset and KeepSettings are both false, the values of the properties listed above are all set to match the source dataset.

If Reset is true, the properties listed above are all cleared.

If Reset is false and KeepSettings is true, the properties listed above are not changed. In this case, the application must ensure that existing indexes, providers, and so on are compatible with the cloned data.

After calling CloneCursor, the data for the client dataset is the same as the data for Source. Edits performed by either client dataset are reflected in the data of both datasets.

Note: Because cloned cursors are shared, changing properties that affect the cursor affect both datasets. For example, setting the ReadOnly (see page 8493) property of either dataset will make both datasets read-only.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
value of the first field for the last record to the server so
that the server can return the correct records. The complete
project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
var
  LastValue: OleVariant;
  CDSClone: TClientDataSet;
begin
  if ClientDataSet1.Active then
  begin
    CDSClone := TClientDataSet.Create(Form1);
    try
      CDSClone.CloneCursor(ClientDataSet1, True);
      { turn off FetchOnDemand so that the clone only fetches
        the last LOCAL record }
      CDSClone.FetchOnDemand := False;
      CDSClone.Last;
      LastValue := CDSClone.Fields[0].AsString;
      CDSClone.Close;
    finally
      CDSClone.Free;
    end;
  end else
    LastValue := NULL;
  OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
  ClientDataSet1.GetNextPacket;
end;
```

See Also

Data (see page 8467)

Filter (see page 8476)

Filtered (see page 7027)

FilterOptions (see page 7027)

OnFilterRecord (see page 7055)

IndexName (see page 8483)

MasterSource (see page 8486)

MasterFields (see page 8486)

ReadOnly (see page 8493)

ProviderName (see page 8492)

RemoteServer (see page 8493)

CloneSource (see page 8464)

45.1.8.4.20 TCustomClientDataSet.CloseCursor Method

Pascal

```
procedure CloseCursor; override;
```

C++

```
virtual __fastcall CloseCursor();
```

Description

This is CloseCursor, a member of class TCustomClientDataSet.

45.1.8.4.21 TCustomClientDataSet.CompareBookmarks Method

Compares two bookmarks.

Pascal

```
function CompareBookmarks(Bookmark1: TBookmark; Bookmark2: TBookmark): Integer; override;
```

C++

```
virtual __fastcall int CompareBookmarks(TBookmark Bookmark1, TBookmark Bookmark2);
```

Description

Call CompareBookmarks to determine if two bookmarks are identical. Bookmark1 and Bookmark2 are the bookmarks to compare. CompareBookmarks returns -1 if Bookmark1 is less than Bookmark2, 1 if Bookmark1 is greater than Bookmark2, and 0 if the bookmarks are identical or nil (Delphi) or NULL (C++).

See Also

BookmarkValid (see page 8395)

45.1.8.4.22 TCustomClientDataSet.ConstraintsDisabled Method

Indicates whether constraints are enabled for a client dataset.

Pascal

```
function ConstraintsDisabled: Boolean;
```

C++

```
__fastcall Boolean ConstraintsDisabled();
```

Description

Call ConstraintsDisabled to determine if constraints are currently disabled for a client dataset. If ConstraintsDisabled returns true, constraints are not enforced when edits are posted. If false, constraints are enforced when records are posted.

See Also

DisableConstraints (see page 8406)

EnableConstraints (see page 8412)

CustomConstraint (see page 7156)

ImportedConstraint (see page 7167)

Constraints (see page 7013)

45.1.8.4.23 TCustomClientDataSet.ConstraintsStored Method

Pascal

```
function ConstraintsStored: Boolean;
```

C++

```
__fastcall Boolean ConstraintsStored();
```

Description

This is ConstraintsStored, a member of class TCustomClientDataSet.

45.1.8.4.24 TCustomClientDataSet.CreateBlobStream Method

Returns a TClientBlobStream (see page 8285) object for reading or writing the data in a specified blob field.

Pascal

```
function CreateBlobStream(Field: TField; Mode: TBlobStreamMode): TStream; override;
```

C++

```
virtual __fastcall TStream CreateBlobStream(TField Field, TBlobStreamMode Mode);
```

Description

Call CreateBlobStream to obtain a stream for reading data from or writing data to a binary large object (BLOB) field. The Field parameter must specify a TBlobField component from the Fields property array. The Mode parameter specifies whether the stream will be used for reading, writing, or updating the contents of the field.

Note: Calling CreateBlobStream ensures that the current record has fetched its BLOB data if the FetchOnDemand (see page 8471) property is true. If FetchOnDemand (see page 8471) is false and the provider does not include BLOB data in data packets by default, call FetchBlobs (see page 8414) before calling CreateBlobStream.

See Also

Fields (see page 7023)

TClientBlobStream (see page 8285)

FetchBlobs (see page 8414)

FetchOnDemand (see page 8471)

45.1.8.4.25 TCustomClientDataSet.CreateDataSet Method

Creates a new, empty client dataset.

Pascal

```
procedure CreateDataSet;
```

C++

```
__fastcall CreateDataSet();
```

Description

Call CreateDataSet at runtime to create a new, empty client dataset that can then be edited and saved.

If the FieldDefs (see page 8472) property contains values, these values are used to create field definitions. Otherwise the Fields property is used. One or both of these properties must contain values in order to create a dataset. If neither property is set, CreateDataSet raises an exception.

If the IndexDefs (see page 8480) property contains values, these values are used to create indexes for the dataset. Indexes created from the IndexDefs (see page 8480) property are saved with the client dataset when it is saved to a file.

Note: You can create a dataset at design time as well: Create (see page 8386) persistent field components or field definitions. Then right-click the client dataset and select Create (see page 8386) Data (see page 8467) Set.

C++ Examples:

```
/*
The following code creates and activates a client dataset in
the form's OnCreate event handler.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TFieldDefs *pDefs = CDS2->FieldDefs;
  TFieldDef *pDef = pDefs->AddFieldDef();
  pDef->DataType = ftInteger;
  pDef->Name = "Field1";

  pDef = pDefs->AddFieldDef();
  pDef->DataType = ftString;
  pDef->Size = 10;
  pDef->Name = "Field2";

  TIndexDef *pIDef = CDS2->IndexDefs->AddIndexDef();
  pIDef->Fields = "Field1";
  pIDef->Name = "IntIndex";

  CDS2->CreateDataSet();
}
```

Delphi Examples:

```
{
This example requires two ClientDataSets, two DBGrids, two
DataSources and two DBNavigators.  Name the two
ClientDataSets CDS and CDS2. Set the DataSet property in the
first DataSource to CDS and name that DataSource DS.  Set the
DataSet property in the second DataSource to CDS2 and name
that DataSource DS2.  Set the DataSource in the first
DBNavigator to DS and the DataSource in the second DBNavigator
to DS2.  Set the DataSource in the first DBGrid to DS and the
DataSource in the second DBGrid to DS2.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
type to Integer.  The file used to load CDS must have an
attribute name of HOST with an attribute type of string and
an attribute name of PORTA with an attribute type of integer.
Add DB, DBCtrls, Grids, DBGrids, DBClient and SwSystem to the
uses clause.  "LoadFormFile" to start with the file, or
"CreateDataSet" to start from scratch.  You must "CloseDataSet"
after a "LoadFromFile" before you can do another "CreateDataSet".
}
{$IFNDEF UNICODE}
uses SwSystem;
```

```
{$ENDIF}

procedure TForm1.Button1Click(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.SaveToFile(GetCurrentDir + '\CDS.XML', dfXML);
{$ELSE}
  CDS.SaveToFile(gsAppPath + 'CDS.XML', dfXML);
{$ENDIF}
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  CDS.CreateDataSet;
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  CDS2.Data := CDS.Data;
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  CDS.Close;
end;
{
The following code creates and activates a client dataset in
the form’s OnCreate event handler.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  with CDS2 do
  begin
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftInteger;
      Name := 'Field1';
    end;
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftString;
      Size := 10;
      Name := 'Field2';
    end;
    with IndexDefs.AddIndexDef do
    begin
      Fields := 'Field1';
      Name := 'IntIndex';
    end;
    CreateDataSet;
  end;
end;
```

See Also

AddIndex (see page 8391)

DeleteIndex (see page 8406)

FieldDefs (see page 7019)

Fields (see page 7023)

IndexDefs (see page 8480)

Adding a New Index

45.1.8.4.26 TCustomClientDataSet.DataConvert Method

Pascal

```
procedure DataConvert(Field: TField; Source: Pointer; Dest: Pointer; ToNative: Boolean);
override;
```

C++

```
virtual __fastcall DataConvert(TField Field, void * Source, void * Dest, Boolean ToNative);
```

Description

This is DataConvert, a member of class TCustomClientDataSet.

45.1.8.4.27 TCustomClientDataSet.DataEvent Method

Pascal

```
procedure DataEvent(Event: TDataEvent; Info: Longint); override;
```

C++

```
virtual __fastcall DataEvent(TDataEvent Event, Longint Info);
```

Description

This is DataEvent, a member of class TCustomClientDataSet.

45.1.8.4.28 TCustomClientDataSet.DataRequest Method

Generates an OnDataRequest event on the associated provider

Pascal

```
function DataRequest(Data: OleVariant): OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant DataRequest(OleVariant Data);
```

Description

Call DataRequest to invoke an OnDataRequest event handler. This method is provided to allow application developers to customize the communication between a client application and an external provider component.

There is no predefined meaning for the Input parameter or the returned value. Instead, the Data (see page 8467) parameter provides the input parameter to an OnDataRequest event handler and DataRequest returns the value that that event handler returns.

See Also

OnDataRequest (see page 18482)

ProviderName (see page 8492)

Communicating with Providers Using Custom Events

Communicating with the Client Dataset

Responding to Client-generated Events

45.1.8.4.29 TCustomClientDataSet.DeactivateFilters Method

Pascal

```
procedure DeactivateFilters;
```

C++

```
__fastcall DeactivateFilters();
```

Description

This is DeactivateFilters, a member of class TCustomClientDataSet.

45.1.8.4.30 TCustomClientDataSet.DefChanged Method

Pascal

```
procedure DefChanged(Sender: TObject); override;
```

C++

```
virtual __fastcall DefChanged(TObject * Sender);
```

Description

This is DefChanged, a member of class TCustomClientDataSet.

45.1.8.4.31 TCustomClientDataSet.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor (see page 2839)

DefineBinaryProperty (see page 2837)

DefineProperties (see page 2931)

DefineProperty (see page 2838)

45.1.8.4.32 TCustomClientDataSet.DeleteIndex Method

Deletes an index for the client dataset.

Pascal

```
procedure DeleteIndex(const Name: string);
```

C++

```
__fastcall DeleteIndex(const AnsiString Name);
```

Description

Call DeleteIndex to remove an index for a client dataset. Name is the name of the index to delete.

Note: You cannot delete the reserved indexes, DEFAULT_ORDER and CHANGEINDEX.

See Also

AddIndex (see page 8391)

IndexDefs (see page 8480)

Deleting and Switching Indexes

45.1.8.4.33 TCustomClientDataSet.DestroyLookupCursor Method

Pascal

```
procedure DestroyLookupCursor; virtual;
```

C++

```
virtual __fastcall DestroyLookupCursor();
```

Description

This is DestroyLookupCursor, a member of class TCustomClientDataSet.

45.1.8.4.34 TCustomClientDataSet.DisableConstraints Method

Disables the enforcement of constraints.

Pascal

```
procedure DisableConstraints;
```

C++

```
__fastcall DisableConstraints();
```

Description

Call DisableConstraints to prevent, on a temporary basis, enforcement of constraints that have been assigned to the dataset or its fields. For example, when an application iterates through a client dataset, it may improve performance to disable constraints temporarily.

Note: Calling DisableConstraints increments a reference count. As long as this reference count is greater than zero, constraints are disabled for the dataset. To prevent accidental disabling of constraints, always group a call to DisableConstraints with a call to EnableConstraints (see page 8412).

See Also

EnableConstraints (see page 8412)

ConstraintsDisabled (see page 8400)

CustomConstraint (see page 7156)

ImportedConstraint (see page 7167)

Constraints (see page 7013)

Handling Constraints from the Server

45.1.8.4.35 TCustomClientDataSet.DoAfterApplyUpdates Method

Pascal

```
procedure DoAfterApplyUpdates(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoAfterApplyUpdates(OleVariant OwnerData);
```

Description

This is DoAfterApplyUpdates, a member of class TCustomClientDataSet.

45.1.8.4.36 TCustomClientDataSet.DoAfterExecute Method

Pascal

```
procedure DoAfterExecute(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoAfterExecute(OleVariant OwnerData);
```

Description

This is DoAfterExecute, a member of class TCustomClientDataSet.

45.1.8.4.37 TCustomClientDataSet.DoAfterGetParams Method

Pascal

```
procedure DoAfterGetParams(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoAfterGetParams(OleVariant OwnerData);
```

Description

This is DoAfterGetParams, a member of class TCustomClientDataSet.

45.1.8.4.38 TCustomClientDataSet.DoAfterGetRecords Method

Pascal

```
procedure DoAfterGetRecords(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoAfterGetRecords(OleVariant OwnerData);
```

Description

This is DoAfterGetRecords, a member of class TCustomClientDataSet.

45.1.8.4.39 TCustomClientDataSet.DoAfterRowRequest Method

Pascal

```
procedure DoAfterRowRequest(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoAfterRowRequest(OleVariant OwnerData);
```

Description

This is DoAfterRowRequest, a member of class TCustomClientDataSet.

45.1.8.4.40 TCustomClientDataSet.DoApplyUpdates Method

Pascal

```
function DoApplyUpdates(Delta: OleVariant; MaxErrors: Integer; out ErrorCount: Integer):
OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant DoApplyUpdates(OleVariant Delta, int MaxErrors, int ErrorCount);
```

Description

This is DoApplyUpdates, a member of class TCustomClientDataSet.

45.1.8.4.41 TCustomClientDataSet.DoBeforeApplyUpdates Method

Pascal

```
procedure DoBeforeApplyUpdates(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoBeforeApplyUpdates(OleVariant OwnerData);
```

Description

This is DoBeforeApplyUpdates, a member of class TCustomClientDataSet.

45.1.8.4.42 TCustomClientDataSet.DoBeforeExecute Method

Pascal

```
procedure DoBeforeExecute(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoBeforeExecute(OleVariant OwnerData);
```

Description

This is DoBeforeExecute, a member of class TCustomClientDataSet.

45.1.8.4.43 TCustomClientDataSet.DoBeforeGetParams Method

Pascal

```
procedure DoBeforeGetParams(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoBeforeGetParams(OleVariant OwnerData);
```

Description

This is DoBeforeGetParams, a member of class TCustomClientDataSet.

45.1.8.4.44 TCustomClientDataSet.DoBeforeGetRecords Method

Pascal

```
procedure DoBeforeGetRecords(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoBeforeGetRecords(OleVariant OwnerData);
```

Description

This is DoBeforeGetRecords, a member of class TCustomClientDataSet.

45.1.8.4.45 TCustomClientDataSet.DoBeforeInsert Method

Pascal

```
procedure DoBeforeInsert; override;
```

C++

```
virtual __fastcall DoBeforeInsert();
```

Description

This is DoBeforeInsert, a member of class TCustomClientDataSet.

45.1.8.4.46 TCustomClientDataSet.DoBeforeRowRequest Method

Pascal

```
procedure DoBeforeRowRequest(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoBeforeRowRequest(OleVariant OwnerData);
```

Description

This is DoBeforeRowRequest, a member of class TCustomClientDataSet.

45.1.8.4.47 TCustomClientDataSet.DoExecute Method

Pascal

```
procedure DoExecute(Params: OleVariant); virtual;
```

C++

```
virtual __fastcall DoExecute(OleVariant Params);
```

Description

This is DoExecute, a member of class TCustomClientDataSet.

45.1.8.4.48 TCustomClientDataSet.DoGetRecords Method

Pascal

```
function DoGetRecords(Count: Integer; out RecsOut: Integer; Options: Integer; const
CommandText: WideString; Params: OleVariant): OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant DoGetRecords(int Count, int RecsOut, int Options, const BSTR
CommandText, OleVariant Params);
```

Description

This is DoGetRecords, a member of class TCustomClientDataSet.

45.1.8.4.49 TCustomClientDataSet.DoOnNewRecord Method

Pascal

```
procedure DoOnNewRecord; override;
```

C++

```
virtual __fastcall DoOnNewRecord();
```

Description

This is DoOnNewRecord, a member of class TCustomClientDataSet.

45.1.8.4.50 TCustomClientDataSet.DoRowRequest Method

Pascal

```
function DoRowRequest(Row: OleVariant; RequestType: Integer): OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant DoRowRequest(OleVariant Row, int RequestType);
```

Description

This is DoRowRequest, a member of class TCustomClientDataSet.

45.1.8.4.51 TCustomClientDataSet.EditKey Method

Enables modification of the search key buffer.

Pascal

```
procedure EditKey;
```

C++

```
__fastcall EditKey();
```

Description

Call EditKey to put the client dataset in dsSetKey state while preserving the current contents of the current search key buffer. To determine current search keys, you can use the IndexFields (see page 8482) property to iterate over the fields used by the current index.

EditKey is especially useful when performing multiple searches where only one or two field values among many change between each search.

See Also

Fields (see page 7023)

FindKey (see page 8416)

GotoKey (see page 8427)

GotoNearest (see page 8427)

SetKey (see page 8449)

State (see page 7036)

45.1.8.4.52 TCustomClientDataSet.EditRangeEnd Method

Enables changing the ending value for an existing range.

Pascal

```
procedure EditRangeEnd;
```

C++

```
__fastcall EditRangeEnd();
```

Description

Call EditRangeEnd to change the ending value for an existing range. To specify an end range value, call FieldByName after calling EditRangeEnd. After assigning a new ending value, call ApplyRange (see page 8393) to activate the modified range.

See Also

ApplyRange (see page 8393)

CancelRange (see page 8396)

EditRangeStart (see page 8411)

FieldByName (see page 6936)

IndexFieldNames (see page 8482)

KeyExclusive (see page 8484)

SetRange (see page 8452)

SetRangeEnd (see page 8453)

SetRangeStart (see page 8454)

Modifying a Range

45.1.8.4.53 TCustomClientDataSet.EditRangeStart Method

Enables changing the starting value for an existing range.

Pascal

```
procedure EditRangeStart;
```

C++

```
__fastcall EditRangeStart();
```

Description

Call EditRangeStart to change the starting value for an existing range. To specify a start range value, call FieldByName after calling EditRangeStart. After assigning a new ending value, call ApplyRange (see page 8393) to activate the modified range.

See Also

ApplyRange (see page 8393)

CancelRange (see page 8396)

EditRangeEnd (see page 8411)

FieldByName (see page 6936)

IndexFieldNames (see page 8482)

SetRange (see page 8452)

SetRangeEnd (see page 8453)

SetRangeStart (see page 8454)

Modifying a Range

45.1.8.4.54 TCustomClientDataSet.EmptyDataSet Method

Removes all records from the dataset.

Pascal

```
procedure EmptyDataSet; virtual;
```

C++

```
virtual __fastcall EmptyDataSet();
```

Description

Call EmptyDataSet to empty the client dataset.

See Also

Data (see page 8467)

45.1.8.4.55 TCustomClientDataSet.EnableConstraints Method

Reenables enforcement of constraints when records are posted.

Pascal

```
procedure EnableConstraints;
```

C++

```
__fastcall EnableConstraints();
```

Description

Call EnableConstraints to turn on constraints previously disabled by a call to DisableConstraints (see page 8406).

Note: Calling EnableConstraints decrements a reference count. When this reference count is zero constraints are enabled for the dataset. To prevent accidental disabling of constraints, always follow a call to DisableConstraints (see page 8406) with a matching call to EnableConstraints.

See Also

DisableConstraints (see page 8406)

ConstraintsDisabled (see page 8400)

Constraints (see page 7013)

ImportedConstraint (see page 7167)

CustomConstraint (see page 7156)

Handling Constraints from the Server

45.1.8.4.56 TCustomClientDataSet.Execute Method

Executes an SQL command by the provider's dataset.

Pascal

```
procedure Execute; virtual;
```

C++

```
virtual __fastcall Execute();
```

Description

Execute allows the client dataset to interact with a query or stored procedure that does not generate a result set. Call Execute to execute the command specified by CommandText (see page 8465) or the provider's query or stored procedure. If CommandText (see page 8465) is not an empty string and the provider Options allow clients to supply a CommandText (see page 8465) value, then Execute passes CommandText (see page 8465) to the provider along with the Params (see page 8489) property (to supply parameters for the statement). Otherwise, Execute tells the provider to apply any Params (see page 8489) and execute its associated query or stored procedure. After executing the appropriate statement, the provider returns any output parameters and Execute updates the Params (see page 8489) property to reflect them.

When the application calls Execute, the client dataset receives a BeforeExecute (see page 8514) event before the request is passed to the provider component. After the provider executes the query or stored procedure, it returns any output parameters, which are used to update the Params (see page 8489) property. Finally, the client dataset receives an AfterExecute (see page 8504) event.

Note: Do not use Execute to execute a query or stored procedure that returns a cursor. Most datasets use separate commands to execute queries or stored procedures when they do or don't return data. When a provider passes the Execute command on to its dataset, the dataset uses the command for executing queries or stored procedures that don't return a cursor.

See Also

ExecSQL (see page 10075)

ExecProc (see page 10156)

BeforeExecute (see page 8514)

AfterExecute (see page 8504)

Params (see page 8489)

CommandText (see page 8465)

Getting Parameters from the Application Server

Overriding the Dataset On the Application Server

Sending Query or Stored Procedure Parameters

Communicating with the Client Dataset

45.1.8.4.57 TCustomClientDataSet.FetchBlobs Method

Retrieves BLOB field values from the provider.

Pascal

```
procedure FetchBlobs;
```

C++

```
__fastcall FetchBlobs();
```

Description

Call FetchBlobs to retrieve BLOB field values from a provider when this information is not provided automatically. BLOB field values are automatically included in data packets unless the provider includes poFetchBlobsOnDemand in its Options property. (If the client dataset uses an internal provider, that provider's Options property is available as the client dataset's Options property)

There is no need to call FetchBlobs if the provider automatically includes BLOB data or if the client dataset's FetchOnDemand (see page 8471) property is true.

See Also

RowRequest (see page 18472)

FetchOnDemand (see page 8471)

Options (see page 18440)

BeforeRowRequest (see page 8520)

AfterRowRequest (see page 8509)

Options

Requesting Data from the Source Dataset or Document

Communicating with the Client Dataset

45.1.8.4.58 TCustomClientDataSet.FetchDetails Method

Retrieves nested detail datasets from the provider.

Pascal

```
procedure FetchDetails;
```

C++

```
__fastcall FetchDetails();
```

Description

Call FetchDetails to retrieve nested detail sets from a provider when this information is not provided automatically. Nested details are automatically included in data packets unless the provider includes poFetchDetailsOnDemand in its Options property. (If the client dataset uses an internal provider, that provider's Options property is available as the client dataset's Options property)

FetchDetails updates the current record to include the missing detail data sets.

There is no need to call FetchDetails if the provider automatically includes nested detail sets or if the client dataset's FetchOnDemand (see page 8471) property is true.

See Also

RowRequest (see page 18472)

FetchOnDemand (see page 8471)

Options (see page 18440)

BeforeRowRequest (see page 8520)

AfterRowRequest (see page 8509)

Options

Requesting Data from the Source Dataset or Document

Communicating with the Client Dataset

45.1.8.4.59 TCustomClientDataSet.FetchMoreData Method

Pascal

```
procedure FetchMoreData(All: Boolean); virtual;
```

C++

```
virtual __fastcall FetchMoreData(Boolean All);
```

Description

This is FetchMoreData, a member of class TCustomClientDataSet.

45.1.8.4.60 TCustomClientDataSet.FetchParams Method

Retrieves parameter values from the provider.

Pascal

```
procedure FetchParams;
```

C++

```
__fastcall FetchParams();
```

Description

Call FetchParams to initialize the Params (see page 8489) property with the current parameter values on the provider's dataset. FetchParams does nothing unless the provider's dataset can supply parameter values.

Note: At design time, you can fetch parameter values by right-clicking the client dataset and choosing Fetch Params (see page 8489).

Warning: If the provider is on a stateless application server, the values returned by FetchParams can be changed at any time by other clients. To obtain output parameters from a stateless application server, use the Execute (see page 8413) method instead.

See Also

AS_GetParams (see page 16454)

Params (see page 8489)

Execute (see page 8413)

Getting Parameters from the Application Server

Passing Parameters to the Source Dataset

Communicating with the Client Dataset

45.1.8.4.61 TCustomClientDataSet.FindKey Method

Searches for a record containing specified field values.

Pascal

```
function FindKey(const KeyValues: array of const): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean FindKey(const array of const KeyValues);
```

Description

Call FindKey to search for a specific record in a dataset. KeyValues contains a comma-delimited array of field values, called a key. Each value in the key can be a literal, a variable, nil (Delphi), or NULL (C++). If the number of values passed in KeyValues is less than the number of columns in the index used for the search, the missing values are assumed to be NULL. For C++, KeyValues_Size indicates the index of the last value in the KeyValues array. This is one less than the number of key values.

If a search is successful, FindKey positions the cursor on the matching record and returns true. Otherwise the cursor is not moved, and FindKey returns false.

See Also

FindNearest (see page 8416)

GotoKey (see page 8427)

GotoNearest (see page 8427)

IndexName (see page 8483)

IndexFieldNames (see page 8482)

Locate (see page 8435)

Executing a Search with Find Methods

45.1.8.4.62 TCustomClientDataSet.FindNearest Method

Moves the cursor to the record that most closely matches a specified set of key values.

Pascal

```
procedure FindNearest(const KeyValues: array of const);
```

C++

```
__fastcall FindNearest(const array of const KeyValues);
```

Description

Call FindNearest to move the cursor to a specific record in a dataset or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter. If there are no records that match or exceed the specified criteria, FindNearest positions the cursor on the last record in the table. The KeyExclusive (see page 8484) property controls whether matching values are considered.

KeyValues contains a comma-delimited array of field values, called a key. If the number of values passed in KeyValues is less than the number of columns in the index used for the search, the missing values are assumed to be NULL.. For C++,

KeyValues_Size specifies the index of the last value in the KeyValues array. This is one less than the number of key values.

Note: FindNearest works only with string data types.

See Also

FindKey (see page 8416)

GotoKey (see page 8427)

GotoNearest (see page 8427)

IndexName (see page 8483)

IndexFieldNames (see page 8482)

KeyExclusive (see page 8484)

Locate (see page 8435)

Executing a Search with Find Methods

45.1.8.4.63 TCustomClientDataSet.FindRecord Method

Pascal

```
function FindRecord(Restart: Boolean; GoForward: Boolean): Boolean; override;
```

C++

```
virtual __fastcall Boolean FindRecord(Boolean Restart, Boolean GoForward);
```

Description

This is FindRecord, a member of class TCustomClientDataSet.

45.1.8.4.64 TCustomClientDataSet.FreeKeyBuffers Method

Pascal

```
procedure FreeKeyBuffers;
```

C++

```
__fastcall FreeKeyBuffers();
```

Description

This is FreeKeyBuffers, a member of class TCustomClientDataSet.

45.1.8.4.65 TCustomClientDataSet.FreeRecordBuffer Method

Pascal

```
procedure FreeRecordBuffer(var Buffer: PChar); override;
```

C++

```
virtual __fastcall FreeRecordBuffer(const char * Buffer);
```

Description

This is FreeRecordBuffer, a member of class TCustomClientDataSet.

45.1.8.4.66 TCustomClientDataSet.GetAggregateValue Method

Pascal

```
function GetAggregateValue(Field: TField): Variant; override;
```

C++

```
virtual __fastcall Variant GetAggregateValue(TField Field);
```

Description

This is GetAggregateValue, a member of class TCustomClientDataSet.

45.1.8.4.67 TCustomClientDataSet.GetAppServer Method

Pascal

```
function GetAppServer: IAppServer; virtual;
```

C++

```
virtual __fastcall IAppServer GetAppServer();
```

Description

This is GetAppServer, a member of class TCustomClientDataSet.

45.1.8.4.68 TCustomClientDataSet.GetBookmarkData Method

Pascal

```
procedure GetBookmarkData(Buffer: PChar; Data: Pointer); override;
```

C++

```
virtual __fastcall GetBookmarkData(const char * Buffer, void * Data);
```

Description

This is GetBookmarkData, a member of class TCustomClientDataSet.

45.1.8.4.69 TCustomClientDataSet.GetBookmarkFlag Method

Pascal

```
function GetBookmarkFlag(Buffer: PChar): TBookmarkFlag; override;
```

C++

```
virtual __fastcall TBookmarkFlag GetBookmarkFlag(const char * Buffer);
```

Description

This is GetBookmarkFlag, a member of class TCustomClientDataSet.

45.1.8.4.70 TCustomClientDataSet.GetCanModify Method

Pascal

```
function GetCanModify: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetCanModify();
```

Description

This is GetCanModify, a member of class TCustomClientDataSet.

45.1.8.4.71 TCustomClientDataSet.GetCurrentRecord Method

Retrieves the current record into a buffer.

Pascal

```
function GetCurrentRecord(Buffer: PChar): Boolean; override;
```

C++

```
virtual __fastcall Boolean GetCurrentRecord(const char * Buffer);
```

Description

Most applications should not need to call GetCurrentRecord. TCustomClientDataSet (see page 8373) automatically allocates a buffer for the active record.

Call GetCurrentRecord to copy the current record into a buffer allocated by the application. Buffer must be at least as big as the number of bytes indicated by the RecordSize property.

See Also

RecordSize

ActiveBuffer (see page 6909)

CreateBlobStream (see page 8401)

45.1.8.4.72 TCustomClientDataSet.GetDataSource Method

Pascal

```
function GetDataSource: TDataSource; override;
```

C++

```
virtual __fastcall TDataSource GetDataSource();
```

Description

This is GetDataSource, a member of class TCustomClientDataSet.

45.1.8.4.73 GetFieldData Method

This is the overview for the GetFieldData method overload.

Overload List

	Name	Description
	TCustomClientDataSet.GetFieldData (Integer, Pointer) (see page 8419)	Retrieves the current value of a field.
	TCustomClientDataSet.GetFieldData (TField, Pointer) (see page 8419)	Retrieves the current value of a field.

45.1.8.4.73.1 TCustomClientDataSet.GetFieldData Method (Integer, Pointer)

Retrieves the current value of a field.

Pascal

```
function GetFieldData(Field: TField; Buffer: Pointer): Boolean; override;
function GetFieldData(FieldNo: Integer; Buffer: Pointer): Boolean; override; overload;
```

C++

```
virtual __fastcall Boolean GetFieldData(TField Field, void * Buffer);
virtual __fastcall Boolean GetFieldData(int FieldNo, void * Buffer);
```

Description

Call GetFieldData to obtain the value of a field component as it is stored in the record buffer. TField objects call GetFieldData to implement their GetData method.

The Field or FieldNo parameter indicates the field whose data should be fetched. Field specifies the component itself, while FieldNo indicates its field number. The Buffer parameter is a memory buffer with sufficient space to accept the value of the field as it exists in the database (unformatted and untranslated). NativeFormat indicates whether the dataset fetches the field in the IDE's native format for the field type. When NativeFormat is false, the dataset must convert the field value to the native type using its Translate method. This allows the field to handle data from different types of datasets (ADO-based, BDE-based, and so on) in a uniform manner.

GetFieldData returns a value that indicates whether the data was successfully fetched.

See Also

GetData (see page 7131)

Translate (see page 7001)

45.1.8.4.74 TCustomClientDataSet.GetGroupState Method

Indicates where the current record sits within a specified group of records.

Pascal

```
function GetGroupState(Level: Integer): TGroupPosInds;
```

C++

```
__fastcall TGroupPosInds GetGroupState(int Level);
```

Description

Call GetGroupState to determine where the current record falls in the group of records specified by the Level parameter.

Level identifies a group by its grouping level on the current index. Grouping level 1 is the set of records with the same value on the first field in the index. Grouping level 2 is the set of records with the same values on the first two fields in the index, and so on. If Level specifies a grouping level greater than the GroupingLevel (see page 8479) property, GetGroupState raises an exception.

GetGroupState returns a set that indicates where the current record falls in the current grouping level. It has one of the following values:

Value	Meaning
[gbMiddle]	The current record is neither the first nor the last in the group.
[gbFirst]	The current record is the first in the group, which contains at least two records.
[gbLast]	The current record is the last in the group, which contains at least two records.
[gbFirst,gbLast]	The current record is the only record in the group.

Note: If AggregatesActive (see page 8461) is false or the current index does not support grouping, GetGroupState returns an

empty set.

See Also

GroupingLevel (see page 8479)

IndexName (see page 8483)

GroupingLevel (see page 7291)

AggregatesActive (see page 8461)

Obtaining Aggregate Values

Using Indexes to Group Data

45.1.8.4.75 TCustomClientDataSet.GetIndexField Method

Pascal

```
function GetIndexField(Index: Integer): TField;
```

C++

```
__fastcall TField GetIndexField(int Index);
```

Description

This is GetIndexField, a member of class TCustomClientDataSet.

45.1.8.4.76 TCustomClientDataSet.GetIndexFieldCount Method

Pascal

```
function GetIndexFieldCount: Integer;
```

C++

```
__fastcall int GetIndexFieldCount();
```

Description

This is GetIndexFieldCount, a member of class TCustomClientDataSet.

45.1.8.4.77 TCustomClientDataSet.GetIndexInfo Method

Retrieves information about the current index.

Pascal

```
procedure GetIndexInfo(IndexName: string);
```

C++

```
__fastcall GetIndexInfo(AnsiString IndexName);
```

Description

Call GetIndexInfo to populate a dataset's internal index fields with information about the current index, including its field count, map, and key size.

Pass the name of the current index as the IndexName (see page 8483) to retrieve the default grouping level of the index as well. If an empty string is supplied for IndexName (see page 8483), no grouping level information is retrieved.

Warning: Do not pass GetIndexInfo the name of any index other than the current index.

See Also

FieldCount (see page 7017)

IndexDefs (see page 8480)

KeySize (see page 8485)

GetIndexNames (see page 8422)

GroupingLevel (see page 7291)

45.1.8.4.78 TCustomClientDataSet.GetIndexNames Method

Retrieves a list of available indexes for a client dataset.

Pascal

```
procedure GetIndexNames(List: TStrings);
```

C++

```
__fastcall GetIndexNames(TStrings List);
```

Description

Call GetIndexNames to retrieve a list of all available indexes for a client dataset. List is a string list object, created and maintained by the application, into which to retrieve the index names.

Note: Two indexes are automatically generated for all client datasets:

DEFAULT_ORDER, which describes the order of data when it is put into the data packet.

CHANGEINDEX, which orders the data as it would if it were a delta assigned to another dataset.

See Also

IndexName (see page 8483)

Obtaining Information About Indexes

45.1.8.4.79 TCustomClientDataSet.GetIsClone Method

Pascal

```
function GetIsClone: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetIsClone();
```

Description

This is GetIsClone, a member of class TCustomClientDataSet.

45.1.8.4.80 TCustomClientDataSet.GetIsIndexField Method

Pascal

```
function GetIsIndexField(Field: TField): Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsIndexField(TField Field);
```

Description

This is GetIsIndexField, a member of class TCustomClientDataSet.

45.1.8.4.81 TCustomClientDataSet.GetKeyBuffer Method

Pascal

```
function GetKeyBuffer(KeyIndex: TKeyIndex): PKeyBuffer;
```

C++

```
__fastcall PKeyBuffer GetKeyBuffer(TKeyIndex KeyIndex);
```

Description

This is GetKeyBuffer, a member of class TCustomClientDataSet.

45.1.8.4.82 TCustomClientDataSet.GetKeyExclusive Method

Pascal

```
function GetKeyExclusive: Boolean;
```

C++

```
__fastcall Boolean GetKeyExclusive();
```

Description

This is GetKeyExclusive, a member of class TCustomClientDataSet.

45.1.8.4.83 TCustomClientDataSet.GetKeyFieldCount Method

Pascal

```
function GetKeyFieldCount: Integer;
```

C++

```
__fastcall int GetKeyFieldCount();
```

Description

This is GetKeyFieldCount, a member of class TCustomClientDataSet.

45.1.8.4.84 TCustomClientDataSet.GetNextPacket Method

Retrieves the next block of records from the provider.

Pascal

```
function GetNextPacket: Integer; virtual;
```

C++

```
virtual __fastcall int GetNextPacket();
```

Description

Call GetNextPacket to retrieve the next sequential block of records from a provider. To specify the maximum number of records to return in a packet, set the PacketRecords (see page 8488) property before calling GetNextPacket. A packet is appended to those records already stored in the Data (see page 8467) property.

GetNextPacket returns the number of records added to Data (see page 8467). If the return value is 0, the client dataset has

already received all available records from the provider.

Warning: If the provider is in a stateless application server, you must use a BeforeGetRecords (see page 8515) event handler to ensure that the data packet returns the correct set of records.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
value of the first field for the last record to the server so
that the server can return the correct records. The complete
project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
var
  LastValue: OleVariant;
  CDSClone: TClientDataSet;
begin
  if ClientDataSet1.Active then
  begin
    CDSClone := TClientDataSet.Create(Form1);
    try
      CDSClone.CloneCursor(ClientDataSet1, True);
      { turn off FetchOnDemand so that the clone only fetches
        the last LOCAL record }
      CDSClone.FetchOnDemand := False;
      CDSClone.Last;
      LastValue := CDSClone.Fields[0].AsString;
      CDSClone.Close;
    finally
      CDSClone.Free;
    end;
  end else
    LastValue := NULL;
  OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
  ClientDataSet1.GetNextPacket;
end;
```

See Also

AppendData (see page 8393)

Data (see page 8467)

FetchOnDemand (see page 8471)

PacketRecords (see page 8488)

BeforeGetRecords (see page 8515)

AfterGetRecords (see page 8506)

Requesting Data from the Source Dataset or Document

Using the BDE to Cache Updates

Communicating with the Client Dataset

Updating Data in dbExpress Applications

45.1.8.4.85 TCustomClientDataSet.GetOptionalParam Method

Retrieves a custom parameter from the data packet.

Pascal

```
function GetOptionalParam(const ParamName: string): OleVariant;
```

C++

```
__fastcall OleVariant GetOptionalParam(const AnsiString ParamName);
```

Description

Application developers can add custom information to the data packet using the client dataset's SetOptionalParam (see page 8450) method or a provider's OnGetDataSetProperties event. This information is saved with the data packet when it is saved to a file or stream, and copied with the value of the Data (see page 8467) property. Call GetOptionalParam to retrieve this custom information from the data packet.

The ParamName parameter specifies the name under which the custom information was stored.

Note: Do not confuse the optional data retrieved by this method with the Params (see page 8489) property.

See Also

SetOptionalParam (see page 8450)

Data (see page 8467)

OnGetDataSetProperties (see page 18542)

Adding Application-specific Information to the Data

Adding Custom Information to Data Packets

45.1.8.4.86 TCustomClientDataSet.GetRecNo Method

Pascal

```
function GetRecNo: Integer; override;
```

C++

```
virtual __fastcall int GetRecNo();
```

Description

This is GetRecNo, a member of class TCustomClientDataSet.

45.1.8.4.87 TCustomClientDataSet.GetRecord Method

Pascal

```
function GetRecord(Buffer: PChar; GetMode: TGetMode; DoCheck: Boolean): TGetResult; override;
```

C++

```
virtual __fastcall TGetResult GetRecord(const char * Buffer, TGetMode GetMode, Boolean
DoCheck);
```

Description

This is GetRecord, a member of class TCustomClientDataSet.

45.1.8.4.88 TCustomClientDataSet.GetRecordCount Method

Pascal

```
function GetRecordCount: Integer; override;
```

C++

```
virtual __fastcall int GetRecordCount();
```

Description

This is GetRecordCount, a member of class TCustomClientDataSet.

45.1.8.4.89 TCustomClientDataSet.GetRecordSize Method

Pascal

```
function GetRecordSize: Word; override;
```

C++

```
virtual __fastcall Word GetRecordSize();
```

Description

This is GetRecordSize, a member of class TCustomClientDataSet.

45.1.8.4.90 TCustomClientDataSet.GetRemoteServer Method

Pascal

```
function GetRemoteServer: TCustomRemoteServer; virtual;
```

C++

```
virtual __fastcall TCustomRemoteServer GetRemoteServer();
```

Description

This is GetRemoteServer, a member of class TCustomClientDataSet.

45.1.8.4.91 TCustomClientDataSet.GetStateFieldValue Method

Pascal

```
function GetStateFieldValue(State: TDataSetState; Field: TField): Variant; override;
```

C++

```
virtual __fastcall Variant GetStateFieldValue(TDataSetState State, TField Field);
```

Description

This is GetStateFieldValue, a member of class TCustomClientDataSet.

45.1.8.4.92 TCustomClientDataSet.GotoCurrent Method

Sets the current record in this dataset the same as the current record in another, cloned dataset.

Pascal

```
procedure GotoCurrent(DataSet: TCustomClientDataSet);
```

C++

```
__fastcall GotoCurrent(TCustomClientDataSet DataSet);
```

Description

Call GotoCurrent to position the cursor on the same record that is current in the specified dataset. DataSet is the cloned dataset whose cursor position is to be matched in this dataset.

GotoCurrent is only useful for synchronizing two client datasets that use separate data sources to view and edit the same data.

See Also

CloneCursor (see page 8398)

45.1.8.4.93 TCustomClientDataSet.GotoKey Method

Moves the cursor to a record specified by the current key.

Pascal

```
function GotoKey: Boolean;
```

C++

```
__fastcall Boolean GotoKey();
```

Description

Use GotoKey to move to a record specified by key values assigned with previous calls to SetKey (see page 8449) or EditKey (see page 8410) and actual search values indicated in the search key buffer.

If GotoKey finds a matching record, it positions the cursor on the record and returns true. Otherwise the current cursor position remains unchanged, and GotoKey returns false.

See Also

FindKey (see page 8416)

FindNearest (see page 8416)

GotoNearest (see page 8427)

SetKey (see page 8449)

State (see page 7036)

Executing a Search with Goto Methods

45.1.8.4.94 TCustomClientDataSet.GotoNearest Method

Moves the cursor to the record that most closely matches the current key.

Pascal

```
procedure GotoNearest;
```

C++

```
__fastcall GotoNearest();
```

Description

Call GotoNearest to position the cursor on the record that is either the exact record specified by the current key values in the key buffer, or on the first record whose values exceed those specified. If there is no record that matches or exceeds the specified criteria, GotoNearest positions the cursor on the last record in the dataset.

Note: KeyExclusive (see page 8484) determines which records are considered part of a search range.

Before calling GotoNearest, an application must specify key values by calling SetKey (see page 8449) or EditKey (see page 8410) to put the dataset is dsSetKey state, and then use FieldByName to populate the buffer with search values.

See Also

FindNearest (see page 8416)

GotoKey (see page 8427)

KeyExclusive (see page 8484)

SetKey (see page 8449)

State (see page 7036)

Executing a Search with Goto Methods

45.1.8.4.95 TCustomClientDataSet.InitKeyBuffer Method

Pascal

```
function InitKeyBuffer(Buffer: PKeyBuffer): PKeyBuffer;
```

C++

```
__fastcall PKeyBuffer InitKeyBuffer(PKeyBuffer Buffer);
```

Description

This is InitKeyBuffer, a member of class TCustomClientDataSet.

45.1.8.4.96 TCustomClientDataSet.InitRecord Method

Pascal

```
procedure InitRecord(Buffer: PChar); override;
```

C++

```
virtual __fastcall InitRecord(const char * Buffer);
```

Description

This is InitRecord, a member of class TCustomClientDataSet.

45.1.8.4.97 TCustomClientDataSet.InternalAddRecord Method

Pascal

```
procedure InternalAddRecord(Buffer: Pointer; Append: Boolean); override;
```

C++

```
virtual __fastcall InternalAddRecord(void * Buffer, Boolean Append);
```

Description

This is InternalAddRecord, a member of class TCustomClientDataSet.

45.1.8.4.98 TCustomClientDataSet.InternalCancel Method

Pascal

```
procedure InternalCancel; override;
```

C++

```
virtual __fastcall InternalCancel();
```

Description

This is InternalCancel, a member of class TCustomClientDataSet.

45.1.8.4.99 TCustomClientDataSet.InternalClose Method

Pascal

```
procedure InternalClose; override;
```

C++

```
virtual __fastcall InternalClose();
```

Description

This is InternalClose, a member of class TCustomClientDataSet.

45.1.8.4.100 TCustomClientDataSet.InternalDelete Method

Pascal

```
procedure InternalDelete; override;
```

C++

```
virtual __fastcall InternalDelete();
```

Description

This is InternalDelete, a member of class TCustomClientDataSet.

45.1.8.4.101 TCustomClientDataSet.InternalEdit Method

Pascal

```
procedure InternalEdit; override;
```

C++

```
virtual __fastcall InternalEdit();
```

Description

This is InternalEdit, a member of class TCustomClientDataSet.

45.1.8.4.102 TCustomClientDataSet.InternalFirst Method

Pascal

```
procedure InternalFirst; override;
```

C++

```
virtual __fastcall InternalFirst();
```

Description

This is InternalFirst, a member of class TCustomClientDataSet.

45.1.8.4.103 TCustomClientDataSet.InternalGetOptionalParam Method

Pascal

```
function InternalGetOptionalParam(const ParamName: string; FieldNo: Integer = 0): OleVariant;
```

C++

```
__fastcall OleVariant InternalGetOptionalParam(const AnsiString ParamName, int FieldNo = 0);
```

Description

This is InternalGetOptionalParam, a member of class TCustomClientDataSet.

45.1.8.4.104 TCustomClientDataSet.InternalGotoBookmark Method

Pascal

```
procedure InternalGotoBookmark(Bookmark: TBookmark); override;
```

C++

```
virtual __fastcall InternalGotoBookmark(TBookmark Bookmark);
```

Description

This is InternalGotoBookmark, a member of class TCustomClientDataSet.

45.1.8.4.105 TCustomClientDataSet.InternalHandleException Method

Pascal

```
procedure InternalHandleException; override;
```

C++

```
virtual __fastcall InternalHandleException();
```

Description

This is InternalHandleException, a member of class TCustomClientDataSet.

45.1.8.4.106 TCustomClientDataSet.InternalInitFieldDefs Method

Pascal

```
procedure InternalInitFieldDefs; override;
```

C++

```
virtual __fastcall InternalInitFieldDefs();
```

Description

This is InternalInitFieldDefs, a member of class TCustomClientDataSet.

45.1.8.4.107 TCustomClientDataSet.InternalInitRecord Method

Pascal

```
procedure InternalInitRecord(Buffer: PChar); override;
```

C++

```
virtual __fastcall InternalInitRecord(const char * Buffer);
```

Description

This is InternalInitRecord, a member of class TCustomClientDataSet.

45.1.8.4.108 TCustomClientDataSet.InternalInsert Method

Pascal

```
procedure InternalInsert; override;
```

C++

```
virtual __fastcall InternalInsert();
```

Description

This is InternalInsert, a member of class TCustomClientDataSet.

45.1.8.4.109 TCustomClientDataSet.InternalLast Method

Pascal

```
procedure InternalLast; override;
```

C++

```
virtual __fastcall InternalLast();
```

Description

This is InternalLast, a member of class TCustomClientDataSet.

45.1.8.4.110 TCustomClientDataSet.InternalOpen Method

Pascal

```
procedure InternalOpen; override;
```

C++

```
virtual __fastcall InternalOpen();
```

Description

This is InternalOpen, a member of class TCustomClientDataSet.

45.1.8.4.111 TCustomClientDataSet.InternalPost Method

Pascal

```
procedure InternalPost; override;
```

C++

```
virtual __fastcall InternalPost();
```

Description

This is InternalPost, a member of class TCustomClientDataSet.

45.1.8.4.112 TCustomClientDataSet.InternalRefresh Method

Pascal

```
procedure InternalRefresh; override;
```

C++

```
virtual __fastcall InternalRefresh();
```

Description

This is InternalRefresh, a member of class TCustomClientDataSet.

45.1.8.4.113 TCustomClientDataSet.InternalSetOptionalParam Method

Pascal

```
procedure InternalSetOptionalParam(const ParamName: string; const Value: OleVariant;
IncludeInDelta: Boolean = False; FieldNo: Integer = 0);
```

C++

```
__fastcall InternalSetOptionalParam(const AnsiString ParamName, const OleVariant Value,
Boolean IncludeInDelta = False, int FieldNo = 0);
```

Description

This is InternalSetOptionalParam, a member of class TCustomClientDataSet.

45.1.8.4.114 TCustomClientDataSet.InternalSetToRecord Method

Pascal

```
procedure InternalSetToRecord(Buffer: PChar); override;
```

C++

```
virtual __fastcall InternalSetToRecord(const char * Buffer);
```

Description

This is InternalSetToRecord, a member of class TCustomClientDataSet.

45.1.8.4.115 TCustomClientDataSet.IsCursorOpen Method

Pascal

```
function IsCursorOpen: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCursorOpen();
```

Description

This is IsCursorOpen, a member of class TCustomClientDataSet.

45.1.8.4.116 TCustomClientDataSet.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState (see page 2804)

Create (see page 2766)

TReader (see page 2933)

45.1.8.4.117 TCustomClientDataSet.LoadFromFile Method

Loads a client dataset's data from a file.

Pascal

```
procedure LoadFromFile(const FileName: string = '');
```

C++

```
__fastcall LoadFromFile(const AnsiString FileName = '');
```

Description

Call LoadFromFile to populate the Data (see page 8467) property of the client dataset with data stored in an external file. FileName (see page 8475) is the name of the file containing the data to load. If the FileName (see page 8475) parameter is

an empty string, or if this parameter is omitted (Delphi), LoadFromFile reads from the file specified by the FileName (see page 8475) property.

To load data from a file at design time, right-click on the client dataset and choose Load from File.

Note: FileName (see page 8475) must contain data saved to a file by a previous call to this or another client dataset's SaveToFile (see page 8443) method, or by a TXMLTransform component

.

Delphi Examples:

```
{
This example requires two ClientDataSets, two DBGrids, two
DataSources and two DBNavigators.  Name the two
ClientDataSets CDS and CDS2. Set the DataSet property in the
first DataSource to CDS and name that DataSource DS.  Set the
DataSet property in the second DataSource to CDS2 and name
that DataSource DS2.  Set the DataSource in the first
DBNavigator to DS and the DataSource in the second DBNavigator
to DS2.  Set the DataSource in the first DBGrid to DS and the
DataSource in the second DBGrid to DS2.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
type to Integer.  The file used to load CDS must have an
attribute name of HOST with an attribute type of string and
an attribute name of PORTA with an attribute type of integer.
Add DB, DBCtrls, Grids, DBGrids, DBClient and SwSystem to the
uses clause.  "LoadFormFile" to start with the file, or
"CreateDataSet" to start from scratch.  You must "CloseDataSet"
after a "LoadFromFile" before you can do another "CreateDataSet".
}
{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button1Click(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.SaveToFile(GetCurrentDir + '\CDS.XML', dfXML);
{$ELSE}
  CDS.SaveToFile(gsAppPath + 'CDS.XML', dfXML);
{$ENDIF}
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  CDS.CreateDataSet;
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
```

```
  CDS2.Data := CDS.Data;
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  CDS.Close;
end;
```

See Also

LoadFromStream (see page 8435)

SaveToFile (see page 8443)

FileName (see page 8475)

TXMLTransform (see page 26467)

Loading Data from a File or Stream

Using a Dedicated File on Disk

45.1.8.4.118 TCustomClientDataSet.LoadFromStream Method

Loads a client dataset's data from a stream.

Pascal

```
procedure LoadFromStream(Stream: TStream);
```

C++

```
__fastcall LoadFromStream(TStream Stream);
```

Description

Call LoadFromStream to populate the Data (see page 8467) property of the client dataset with data from a stream. Stream is the name of the stream from which to read data.

Note: Stream must contain data in a client dataset's internal format (for example, data written to a stream by another client dataset using the SaveToStream (see page 8445) method.

See Also

LoadFromFile (see page 8433)

SaveToStream (see page 8445)

45.1.8.4.119 TCustomClientDataSet.Locate Method

Searches the dataset for a specified record and makes that record current.

Pascal

```
function Locate(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions):
Boolean; override;
```

C++

```
virtual __fastcall Boolean Locate(const AnsiString KeyFields, const Variant KeyValues,
TLocateOptions Options);
```

Description

Call Locate to search a dataset for a specific record and position the cursor on it. KeyFields is a string containing a semicolon-delimited list of field names on which to search.

KeyValues is a Variant containing the values to match in the key fields. If KeyFields lists a single field, KeyValues represents the search value for that field. If KeyFields lists multiple fields, pass a Variant array as the KeyValues parameter. In Delphi, you can construct a variant array on the fly using the VarArrayOf routine. For example:

```
with CustTable do
  Locate('Company;Contact;Phone', VarArrayOf(['Sight Diver', 'P', '831-431-1000']),
[loPartialKey]);
TLocateOptions Opts;
Opts.Clear();
Opts << loPartialKey;
Variant locvalues[3];
locvalues[0] = Variant("Sight Diver");
locvalues[1] = Variant("P");
locvalues[2] = Variant("831-431-1000");
CustTable->Locate("Company;Contact;Phone", VarArrayOf(locvalues, 2), Opts);
```

Options is a set that optionally specifies additional search latitude when searching on string fields. If Options contains the loCaseInsensitive setting, then Locate ignores case when matching fields. If Options contains the loPartialKey setting, then the KeyValues parameter can specify incomplete strings for field values. (For example, in the previous statement, the Contact field would match any record where Contact begins with 'P'). If Options is an empty set, or if the KeyFields are not string fields, Options is ignored.

Locate returns true if it finds a matching record, and makes that record the current one. Otherwise Locate returns false.

See Also

FindField (see page 6938)

FindFirst (see page 6939)

FindKey (see page 8416)

Lookup (see page 8436)

45.1.8.4.120 TCustomClientDataSet.LocateRecord Method

Pascal

```
function LocateRecord(const KeyFields: string; const KeyValues: Variant; Options:
TLocateOptions; SyncCursor: Boolean): Boolean;
```

C++

```
__fastcall Boolean LocateRecord(const AnsiString KeyFields, const Variant KeyValues,
TLocateOptions Options, Boolean SyncCursor);
```

Description

This is LocateRecord, a member of class TCustomClientDataSet.

45.1.8.4.121 TCustomClientDataSet.Lookup Method

Retrieves field values from a record that matches specified search values.

Pascal

```
function Lookup(const KeyFields: string; const KeyValues: Variant; const ResultFields:
string): Variant; override;
```

C++

```
virtual __fastcall Variant Lookup(const AnsiString KeyFields, const Variant KeyValues, const
AnsiString ResultFields);
```

Description

Call Lookup to retrieve values for specified fields from a record that matches search criteria. KeyFields is a string containing a semicolon-delimited list of field names on which to search.

KeyValues is a Variant containing the values to match in the key fields. To specify multiple search values, pass KeyValues as a variant array as an argument, or, in Delphi, construct a variant array on the fly using the VarArrayOf routine. For example (C++):

```
Variant locvalues[2];
locvalues[0] = Variant("Sight Diver");
locvalues[1] = Variant("Kato Paphos");
Variant LookupResults = CustTable->Lookup("Company;City", VarArrayOf(locvalues, 1),
"Company;Addr1;Addr2;State;Zip");
```

ResultFields is a string containing a semicolon-delimited list of field names whose values should be returned from the matching record.

Lookup returns a variant array containing the values from the fields specified in ResultFields.

See Also

Locate (see page 8435)

45.1.8.4.122 TCustomClientDataSet.MergeChangeLog Method

Merges any data changes in the change log into the client dataset, overwriting existing values.

Pascal

```
procedure MergeChangeLog;
```

C++

```
__fastcall MergeChangeLog();
```

Description

Call MergeChangeLog to merge changes in the change log into the actual data for the dataset, effectively creating a new data baseline. Existing values for which changes exist are overwritten.

Note: This method is seldom called explicitly when using a provider because is called automatically by ApplyUpdates (see page 8394) as part of the updating process. In file-based applications, MergeChangeLog must be called explicitly if LogChanges (see page 8485) is true.

See Also

Delta (see page 8470)

LogChanges (see page 8485)

Reconcile (see page 8441)

Merging Changes into Data

45.1.8.4.123 TCustomClientDataSet.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

45.1.8.4.124 TCustomClientDataSet.OpenCursor Method

Pascal

```
procedure OpenCursor(InfoQuery: Boolean); override;
```

C++

```
virtual __fastcall OpenCursor(Boolean InfoQuery);
```

Description

This is OpenCursor, a member of class TCustomClientDataSet.

45.1.8.4.125 TCustomClientDataSet.Post Method

Writes a modified record to the Data (see page 8467) property or the change log.

Pascal

```
procedure Post; override;
```

C++

```
virtual __fastcall Post();
```

Description

Call Post to commit changes to the current record. Methods that change the dataset state, such as Edit, Insert, or Append, or that move from one record to another, such as First, Last, Next, and Prior automatically call Post.

Note: If the LogChanges (see page 8485) property is true, Post updates the change log specified by the Delta (see page 8470) property. If LogChanges (see page 8485) is false, Post writes the changes directly to the Data (see page 8467) property.

Delphi Examples:

```
{
This example appends a new record to a table or client
dataset when the user clicks a button. The two fields
Field1 and Field2 are filled from the contents of two edit
```

```
controls.  This example requires a populated TTable or
TDataSet with the appropriate field values.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Customers.Append;
  Customers.FieldValues['Field1'] := StrToInt(Edit1.text);
  Customers.FieldValues['Field2'] := Edit2.text;
  Customers.Post;
end;
```

See Also

Delta (see page 8470)

LogChanges (see page 8485)

Data (see page 8467)

45.1.8.4.126 TCustomClientDataSet.PostKeyBuffer Method

Pascal

```
procedure PostKeyBuffer(Commit: Boolean);
```

C++

```
__fastcall PostKeyBuffer(Boolean Commit);
```

Description

This is PostKeyBuffer, a member of class TCustomClientDataSet.

45.1.8.4.127 TCustomClientDataSet.PSGetCommandText Method

Gets the SQL command to be executed by PSExecute.

Pascal

```
function PSGetCommandText: string; override;
```

C++

```
virtual __fastcall AnsiString PSGetCommandText();
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

PSSetCommandText (see page 7585)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

45.1.8.4.128 TCustomClientDataSet.PSGetCommandType Method

Gets the command type to be executed by PSExecute.

Pascal

```
function PSGetCommandType: TPSCommandType; override;
```

C++

```
virtual __fastcall TPSCommandType PSGetCommandType();
```

Description

PSGetCommandType gets the command type executed by PSExecute. The type may be a type of SQL command or other command types, such as stored procedures.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

PSGetCommandText (see page 7579)

Execute (see page 18466)

Options (see page 18440)

45.1.8.4.129 TCustomClientDataSet.PSGetIndexDefs Method

Returns the definitions of all specified indexes defined for the dataset.

Pascal

```
function PSGetIndexDefs(IndexTypes: TIndexOptions): TIndexDefs; override;
```

C++

```
virtual __fastcall TIndexDefs PSGetIndexDefs(TIndexOptions IndexTypes);
```

Description

The provider component calls PSGetIndexDefs to locate an index that contains information it needs. For example, it uses PSGetIndexDefs to locate an index that contains the key fields returned by PSGetKeyFields. This information allows the provider to locate the records that need to change when it applies updates. The provider also calls this method to locate indexes that it adds to the metadata of data packets.

IndexTypes indicates the options required of the returned indexes. PSGetIndexDefs returns all indexes whose Options property includes the specified flags.

PSGetIndexDefs returns a TIndexDefs object that contains all the indexes of the dataset that match the IndexTypes parameter.

See Also

Options (see page 7291)

PSGetKeyFields (see page 7581)

PSGetDefaultOrder (see page 7580)

45.1.8.4.130 TCustomClientDataSet.ReadDataPacket Method

Pascal

```
procedure ReadDataPacket(Stream: TStream; ReadSize: Boolean);
```

C++

```
__fastcall ReadDataPacket(TStream Stream, Boolean ReadSize);
```

Description

This is ReadDataPacket, a member of class TCustomClientDataSet.

45.1.8.4.131 TCustomClientDataSet.Reconcile Method

Clears successfully updated records from the client dataset's cache of changes.

Pascal

```
function Reconcile(const Results: OleVariant): Boolean;
```

C++

```
__fastcall Boolean Reconcile(const OleVariant Results);
```

Description

Reconcile is called by the ApplyUpdates (see page 8394) method to update the client dataset so that it reflects the result of an update. Reconcile is called after the provider has attempted to apply all updates in the client dataset's Delta (see page 8470) property. (The provider may be internal to the client dataset, an external component in the same application as the client dataset, or on a separate application server.)

Results is a Variant, returned by the ApplyUpdates (see page 8394) method of a provider, that contains error information and record information for each record that could not be applied. Reconcile generates an OnReconcileError (see page 8524) event for every record in Results (every record that was not successfully applied by the provider). Finally, Reconcile adjusts the Delta (see page 8470) property so that it includes only those records that were not successfully applied. Any changes made in the OnReconcileError (see page 8524) event are reflected in the new value of Delta (see page 8470).

Reconcile returns a value that indicates whether reconciling is successful. If true, Reconcile succeeded without error. If false, one or more errors could not be reconciled.

See Also

ApplyUpdates (see page 8394)

Delta (see page 8470)

OnReconcileError (see page 8524)

Applying Updates

Using the BDE to Cache Updates

Updating Data in dbExpress Applications

45.1.8.4.132 TCustomClientDataSet.RefreshInternalCalcFields Method

Pascal

```
procedure RefreshInternalCalcFields(Buffer: PChar); override;
```

C++

```
virtual __fastcall RefreshInternalCalcFields(const char * Buffer);
```

Description

This is RefreshInternalCalcFields, a member of class TCustomClientDataSet.

45.1.8.4.133 TCustomClientDataSet.RefreshRecord Method

Updates the base values of the current record to match the current values of the provider's dataset.

Pascal

```
procedure RefreshRecord;
```

C++

```
__fastcall RefreshRecord();
```

Description

Call Refresh to change the base values of the current record to match the current value on the server. Refresh replaces the original values obtained from a provider, but leaves any changes in the change log intact.

Warning: RefreshRecord should be used with care. If the user has modified the current record, refreshing the original values may mask update conflicts. When the application calls ApplyUpdates (see page 8394), no reconcile error will occur, even where this is appropriate.

Note: A similar method, Refresh, replaces the entire contents of the client dataset with the current values supplied by the provider. Unlike RefreshRecord, which leaves unapplied changes in the change log, Refresh raises an exception when the change log is not empty.

See Also

Refresh (see page 6991)

LogChanges (see page 8485)

BeforeRowRequest (see page 8520)

AfterRowRequest (see page 8509)

Refreshing Records

Communicating with the Client Dataset

45.1.8.4.134 TCustomClientDataSet.ResetAggField Method

Pascal

```
procedure ResetAggField(Field: TField); override;
```

C++

```
virtual __fastcall ResetAggField(TField Field);
```

Description

This is ResetAggField, a member of class TCustomClientDataSet.

45.1.8.4.135 TCustomClientDataSet.ResetCursorRange Method

Pascal

```
function ResetCursorRange: Boolean;
```

C++

```
__fastcall Boolean ResetCursorRange();
```

Description

This is ResetCursorRange, a member of class TCustomClientDataSet.

45.1.8.4.136 TCustomClientDataSet.RevertRecord Method

Undoes edits to the current record.

Pascal

```
procedure RevertRecord;
```

C++

```
__fastcall RevertRecord();
```

Description

Call RevertRecord to remove any modifications made to the current record. RevertRecord can only remove modifications to records that are still in the change log. Changes that have been removed because they were applied by a provider or merged into the Data (see page 8467) property can't be undone using RevertRecord.

Note: To undo the last edit made to a record, call UndoLastChange (see page 8455). To cancel all pending modifications, call CancelUpdates (see page 8396). To undo all edits since an application-defined state, use the SavePoint (see page 8494) property.

See Also

ApplyUpdates (see page 8394)

CancelUpdates (see page 8396)

Reconcile (see page 8441)

UndoLastChange (see page 8455)

SavePoint (see page 8494)

MergeChangeLog (see page 8437)

StatusFilter (see page 8495)

Undoing Changes

Using the BDE to Cache Updates

Updating Data in dbExpress Applications

45.1.8.4.137 TCustomClientDataSet.SaveToFile Method

Saves a client dataset's data to an external file.

Pascal

```
procedure SaveToFile(const FileName: string = ''; Format: TDataPacketFormat = dfBinary);
```

C++

```
__fastcall SaveToFile(const AnsiString FileName = '', TDataPacketFormat Format = dfBinary);
```

Description

Call SaveToFile to write a client dataset's data to an external file for later use by this or other client datasets.

FileName (see page 8475) is the name of the external file to use. If the file already exists, its current contents are overwritten. If an empty string is passed for the FileName (see page 8475) parameter (or, in Delphi, if this parameter is omitted), the data is saved to the file specified by the FileName (see page 8475) property.

Format indicates what format to use when saving the data, binary (dfBinary) or XML (dfXML), or UTF8-based XML (dfXMLUTF8).

Delphi Examples:

```
{
This example requires two ClientDataSets, two DBGrids, two
DataSources and two DBNavigators.  Name the two
ClientDataSets CDS and CDS2. Set the DataSet property in the
first DataSource to CDS and name that DataSource DS.  Set the
DataSet property in the second DataSource to CDS2 and name
that DataSource DS2.  Set the DataSource in the first
DBNavigator to DS and the DataSource in the second DBNavigator
to DS2.  Set the DataSource in the first DBGrid to DS and the
DataSource in the second DBGrid to DS2.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
type to Integer.  The file used to load CDS must have an
attribute name of HOST with an attribute type of string and
an attribute name of PORTA with an attribute type of integer.
Add DB, DBCtrls, Grids, DBGrids, DBClient and SwSystem to the
uses clause.  "LoadFormFile" to start with the file, or
"CreateDataSet" to start from scratch.  You must "CloseDataSet"
after a "LoadFromFile" before you can do another "CreateDataSet".
}
{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button1Click(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.SaveToFile(GetCurrentDir + '\CDS.XML', dfXML);
{$ELSE}
  CDS.SaveToFile(gsAppPath + 'CDS.XML', dfXML);
{$ENDIF}
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
```

```
    CDS.CreateDataSet;
  end;

  procedure TForm1.Button4Click(Sender: TObject);
  begin
    CDS2.Data := CDS.Data;
  end;

  procedure TForm1.Button5Click(Sender: TObject);
  begin
    CDS.Close;
  end;
```

See Also

LoadFromFile (see page 8433)

SaveToStream (see page 8445)

FileName (see page 8475)

Saving Data to a File or Stream

Using a Dedicated File on Disk

45.1.8.4.138 TCustomClientDataSet.SaveToStream Method

Saves a client dataset's data to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream; Format: TDataPacketFormat = dfBinary);
```

C++

```
__fastcall SaveToStream(TStream Stream, TDataPacketFormat Format = dfBinary);
```

Description

Call SaveToStream to write a client dataset's data to a stream.

Stream is the name of an existing stream to write to.

Format indicates what format to use when saving the data, binary (dfBinary) or XML (dfXML), or UTF8-based XML (dfXMLUTF8).

If the stream requires a buffer, use the DataSize (see page 8469) property to determine the required size of the buffer.

See Also

LoadFromStream (see page 8435)

SaveToFile (see page 8443)

DataSize (see page 8469)

Saving Data to a File or Stream

45.1.8.4.139 TCustomClientDataSet.SetAggregates Method

Pascal

```
procedure SetAggregates(Value: TAggregates); virtual;
```

C++

```
virtual __fastcall SetAggregates(TAggregates Value);
```

Description

This is SetAggregates, a member of class TCustomClientDataSet.

45.1.8.4.140 TCustomClientDataSet.SetAltRecBuffers Method

Sets record buffers when accessing field values.

Pascal

```
procedure SetAltRecBuffers(Old: PChar; New: PChar; Cur: PChar);
```

C++

```
__fastcall SetAltRecBuffers(const char * Old, const char * New, const char * Cur);
```

Description

Applications do not normally need to call this method. SetAltRecBuffers sets pointers to records buffers that are used when accessing the OldValue, NewValue, and CurValue properties of a field component belonging to the client dataset. These value are accessed only while processing updates.

See Also

TField (see page 7115)

45.1.8.4.141 TCustomClientDataSet.SetAppServer Method

Pascal

```
procedure SetAppServer(Value: IAppServer); virtual;
```

C++

```
virtual __fastcall SetAppServer(IAppServer Value);
```

Description

This is SetAppServer, a member of class TCustomClientDataSet.

45.1.8.4.142 TCustomClientDataSet.SetBookmarkData Method

Pascal

```
procedure SetBookmarkData(Buffer: PChar; Data: Pointer); override;
```

C++

```
virtual __fastcall SetBookmarkData(const char * Buffer, void * Data);
```

Description

This is SetBookmarkData, a member of class TCustomClientDataSet.

45.1.8.4.143 TCustomClientDataSet.SetBookmarkFlag Method

Pascal

```
procedure SetBookmarkFlag(Buffer: PChar; Value: TBookmarkFlag); override;
```

C++

```
virtual __fastcall SetBookmarkFlag(const char * Buffer, TBookmarkFlag Value);
```

Description

This is SetBookmarkFlag, a member of class TCustomClientDataSet.

45.1.8.4.144 TCustomClientDataSet.SetCommandText Method

Pascal

```
procedure SetCommandText(Value: WideString); virtual;
```

C++

```
virtual __fastcall SetCommandText(BSTR Value);
```

Description

This is SetCommandText, a member of class TCustomClientDataSet.

45.1.8.4.145 TCustomClientDataSet.SetCursorRange Method

Pascal

```
function SetCursorRange: Boolean;
```

C++

```
__fastcall Boolean SetCursorRange();
```

Description

This is SetCursorRange, a member of class TCustomClientDataSet.

45.1.8.4.146 TCustomClientDataSet.SetDataSetField Method

Pascal

```
procedure SetDataSetField(const Value: TDataSetField); override;
```

C++

```
virtual __fastcall SetDataSetField(const TDataSetField Value);
```

Description

This is SetDataSetField, a member of class TCustomClientDataSet.

45.1.8.4.147 TCustomClientDataSet.SetFieldData Method

Pascal

```
procedure SetFieldData(Field: TField; Buffer: Pointer); override;
```

C++

```
virtual __fastcall SetFieldData(TField Field, void * Buffer);
```

Description

This is SetFieldData, a member of class TCustomClientDataSet.

45.1.8.4.148 TCustomClientDataSet.SetFilterData Method

Pascal

```
procedure SetFilterData(const Text: string; Options: TFilterOptions);
```

C++

```
__fastcall SetFilterData(const AnsiString Text, TFilterOptions Options);
```

Description

This is SetFilterData, a member of class TCustomClientDataSet.

45.1.8.4.149 TCustomClientDataSet.SetFiltered Method

Pascal

```
procedure SetFiltered(Value: Boolean); override;
```

C++

```
virtual __fastcall SetFiltered(Boolean Value);
```

Description

This is SetFiltered, a member of class TCustomClientDataSet.

45.1.8.4.150 TCustomClientDataSet.SetFilterOptions Method

Pascal

```
procedure SetFilterOptions(Value: TFilterOptions); override;
```

C++

```
virtual __fastcall SetFilterOptions(TFilterOptions Value);
```

Description

This is SetFilterOptions, a member of class TCustomClientDataSet.

45.1.8.4.151 TCustomClientDataSet.SetFilterText Method

Pascal

```
procedure SetFilterText(const Value: string); override;
```

C++

```
virtual __fastcall SetFilterText(const AnsiString Value);
```

Description

This is SetFilterText, a member of class TCustomClientDataSet.

45.1.8.4.152 TCustomClientDataSet.SetIndexField Method

Pascal

```
procedure SetIndexField(Index: Integer; Value: TField);
```

C++

```
__fastcall SetIndexField(int Index, TField Value);
```

Description

This is SetIndexField, a member of class TCustomClientDataSet.

45.1.8.4.153 TCustomClientDataSet.SetKey Method

Enables setting of keys and ranges for a dataset prior to a search.

Pascal

```
procedure SetKey;
```

C++

```
__fastcall SetKey();
```

Description

Call SetKey to put the dataset into dsSetKey state and clear the current contents of the search key buffer. Use FieldByName to supply the buffer then with a new set of values prior to conducting a search.

Note: To modify an existing key or range, call EditKey (see page 8410).

See Also

EditKey (see page 8410)

FieldByName (see page 6936)

GotoKey (see page 8427)

SetRange (see page 8452)

SetRangeEnd (see page 8453)

SetRangeStart (see page 8454)

State (see page 7036)

Executing a Search with Goto Methods

45.1.8.4.154 TCustomClientDataSet.SetKeyBuffer Method

Pascal

```
procedure SetKeyBuffer(KeyIndex: TKeyIndex; Clear: Boolean);
```

C++

```
__fastcall SetKeyBuffer(TKeyIndex KeyIndex, Boolean Clear);
```

Description

This is SetKeyBuffer, a member of class TCustomClientDataSet.

45.1.8.4.155 TCustomClientDataSet.SetKeyExclusive Method

Pascal

```
procedure SetKeyExclusive(Value: Boolean);
```

C++

```
__fastcall SetKeyExclusive(Boolean Value);
```

Description

This is SetKeyExclusive, a member of class TCustomClientDataSet.

45.1.8.4.156 TCustomClientDataSet.SetKeyFieldCount Method

Pascal

```
procedure SetKeyFieldCount(Value: Integer);
```

C++

```
__fastcall SetKeyFieldCount(int Value);
```

Description

This is SetKeyFieldCount, a member of class TCustomClientDataSet.

45.1.8.4.157 TCustomClientDataSet.SetKeyFields Method

Pascal

```
procedure SetKeyFields(KeyIndex: TKeyIndex; const Values: array of const);
```

C++

```
__fastcall SetKeyFields(TKeyIndex KeyIndex, const array of const Values);
```

Description

This is SetKeyFields, a member of class TCustomClientDataSet.

45.1.8.4.158 TCustomClientDataSet.SetLinkRanges Method

Pascal

```
procedure SetLinkRanges(MasterFields: TList);
```

C++

```
__fastcall SetLinkRanges(TList MasterFields);
```

Description

This is SetLinkRanges, a member of class TCustomClientDataSet.

45.1.8.4.159 TCustomClientDataSet.SetOnFilterRecord Method

Pascal

```
procedure SetOnFilterRecord(const Value: TFilterRecordEvent); override;
```

C++

```
virtual __fastcall SetOnFilterRecord(const TFilterRecordEvent Value);
```

Description

This is SetOnFilterRecord, a member of class TCustomClientDataSet.

45.1.8.4.160 TCustomClientDataSet.SetOptionalParam Method

Saves a custom parameter with the data packet.

Pascal

```
procedure SetOptionalParam(const ParamName: string; const Value: OleVariant; IncludeInDelta:
Boolean = False); virtual;
```

C++

```
virtual __fastcall SetOptionalParam(const AnsiString ParamName, const OleVariant Value,
Boolean IncludeInDelta = False);
```

Description

You can add custom information to a data packet using the SetOptionalParam method. This information is saved with the data packet when it is saved to a file or stream, and copied with the value of the Data (see page 8467) property. For example, by setting a parameter in the OnGetData event of provider, application servers can include custom information with the data that is sent to client applications. Call GetOptionalParam (see page 8425) to retrieve the custom information from the data packet.

The ParamName parameter specifies the name under which the custom information is stored. It can be any value except one of the following:

UNIQUE_KEY	DEFAULT_ORDER	CHANGE_LOG	SERVER_COL
CONSTRAINTS	DATASET_CONTEXT	DATASET_DELTA	LCID
BDERECORD_X	TABLE_NAME	MD_FIELDLINKS	UPDATEMODE

The Value parameter is an OleVariant that represents the custom information.

The IncludeInDelta parameter specifies whether the parameter is also stored with the Delta (see page 8470) property. By including the parameter in the Delta (see page 8470) property, it is sent along with updates to the provider. The provider can then retrieve this information using the GetOptionalParam (see page 8425) method of the client dataset passed as a parameter to an OnUpdateData event handler.

Note: Do not confuse the optional data saved by this method with the Params (see page 8489) property.

See Also

GetOptionalParam (see page 8425)

Data (see page 8467)

Delta (see page 8470)

OnUpdateData (see page 18443)

Adding Application-specific Information to the Data

45.1.8.4.161 TCustomClientDataSet.SetProvider Method

Associates the client dataset with a provider in the same process space.

Pascal

```
procedure SetProvider(Provider: TComponent);
```

C++

```
__fastcall SetProvider(TComponent * Provider);
```

Description

Use SetProvider to specify a local provider component that provides data to and applies updates from the client dataset. The

client dataset communicates with this provider using the IAppServer interface specified by the AppServer (see page 8462) property.

Do not call SetProvider if the client dataset already uses an internal provider component. Also, do not use SetProvider for remote providers on an application server. If the provider is not in the same process space as the client dataset, use the ProviderName (see page 8492) property. If the provider has the same Owner as the client dataset, you can also use the ProviderName (see page 8492) property.

The Provider (see page 18419) parameter must be a descendant of TCustomProvider. Otherwise, SetProvider raises an exception.

Note: If you close the client dataset after assigning a provider using the SetProvider method, when you reopen the client dataset you must call SetProvider again to reassign that provider.

See Also

AppServer (see page 8462)

ProviderName (see page 8492)

TCustomProvider (see page 18449)

Specifying a Provider

45.1.8.4.162 TCustomClientDataSet.SetProviderEOF Method

Pascal

```
procedure SetProviderEOF(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetProviderEOF(Boolean Value);
```

Description

This is SetProviderEOF, a member of class TCustomClientDataSet.

45.1.8.4.163 TCustomClientDataSet.SetRange Method

Sets the starting and ending values of a range, and applies it.

Pascal

```
procedure SetRange(const StartValues: array of const; const EndValues: array of const);
```

C++

```
__fastcall SetRange(const array of const StartValues, const array of const EndValues);
```

Description

Call SetRange to specify a range and apply it to the dataset. The new range replaces the currently specified range, if any.

StartValues indicates the field values that designate the first record in the range. In C++, StartValues_Size is the index of the last value in the StartValues array.

EndValues indicates the field values that designate the last record in the range. In C++, EndValues_Size is the index of the last value in the EndValues array.

SetRange combines the functionality of SetRangeStart (see page 8454), SetRangeEnd (see page 8453), and ApplyRange (see page 8393) in a single procedure call. SetRange performs the following functions:

1Puts the dataset into dsSetKey state.

2Erases any previously specified starting range values and ending range values.

3Sets the start and end range values.

4Applies the range to the dataset.

After a call to SetRange, the cursor is left on the first record in the range.

If either StartValues or EndValues has fewer elements than the number of fields in the current index, then the remaining entries are set to NULL.

See Also

ApplyRange (see page 8393)

CancelRange (see page 8396)

EditRangeEnd (see page 8411)

EditRangeStart (see page 8411)

FieldValues (see page 7025)

IndexFieldNames (see page 8482)

KeyExclusive (see page 8484)

SetRangeEnd (see page 8453)

SetRangeStart (see page 8454)

45.1.8.4.164 TCustomClientDataSet.SetRangeEnd Method

Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.

Pascal

```
procedure SetRangeEnd;
```

C++

```
__fastcall SetRangeEnd();
```

Description

Call SetRangeEnd to put the dataset into dsSetKey state, erase any previous end range values, and set them to NULL. Subsequent field assignments made with FieldByName specify the actual set of ending values for a range.

After assigning end-range values, call ApplyRange (see page 8393) to activate the modified range.

See Also

ApplyRange (see page 8393)

CancelRange (see page 8396)

EditRangeEnd (see page 8411)

EditRangeStart (see page 8411)

FieldByName (see page 6936)

IndexFieldNames (see page 8482)

KeyExclusive (see page 8484)

SetRange (see page 8452)

SetRangeStart (see page 8454)

45.1.8.4.165 TCustomClientDataSet.SetRangeStart Method

Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.

Pascal

```
procedure SetRangeStart;
```

C++

```
__fastcall SetRangeStart();
```

Description

Call SetRangeStart to put the dataset into dsSetKey state, erase any previous start range values, and set them to NULL. Subsequent field assignments to FieldByName specify the actual set of starting values for a range.

After assigning start-range values, call ApplyRange (see page 8393) to activate the modified range.

See Also

ApplyRange (see page 8393)

CancelRange (see page 8396)

EditRangeEnd (see page 8411)

EditRangeStart (see page 8411)

FieldByName (see page 6936)

IndexFieldNames (see page 8482)

KeyExclusive (see page 8484)

SetRange (see page 8452)

SetRangeEnd (see page 8453)

45.1.8.4.166 TCustomClientDataSet.SetRecNo Method

Pascal

```
procedure SetRecNo(Value: Integer); override;
```

C++

```
virtual __fastcall SetRecNo(int Value);
```

Description

This is SetRecNo, a member of class TCustomClientDataSet.

45.1.8.4.167 TCustomClientDataSet.SetRemoteServer Method

Pascal

```
procedure SetRemoteServer(Value: TCustomRemoteServer); virtual;
```

C++

```
virtual __fastcall SetRemoteServer(TCustomRemoteServer Value);
```

Description

This is SetRemoteServer, a member of class TCustomClientDataSet.

45.1.8.4.168 TCustomClientDataSet.SwitchToIndex Method

Pascal

```
procedure SwitchToIndex(const IndexName: string);
```

C++

```
__fastcall SwitchToIndex(const AnsiString IndexName);
```

Description

This is SwitchToIndex, a member of class TCustomClientDataSet.

45.1.8.4.169 TCustomClientDataSet.SyncCursors Method

Pascal

```
procedure SyncCursors(Cursor1: IDSCursor; Cursor2: IDSCursor);
```

C++

```
__fastcall SyncCursors(IDSCursor Cursor1, IDSCursor Cursor2);
```

Description

This is SyncCursors, a member of class TCustomClientDataSet.

45.1.8.4.170 TCustomClientDataSet.UndoLastChange Method

Undoes the last edit, insert, or delete operation to a record in the client dataset.

Pascal

```
function UndoLastChange(FollowChange: Boolean): Boolean;
```

C++

```
__fastcall Boolean UndoLastChange(Boolean FollowChange);
```

Description

Call UndoLastChange to undo the last edit, insert, or delete operation to a record in the client dataset. FollowChange indicates whether to reposition the cursor on the restored record. If FollowChange is true, the cursor is positioned on the restored record. If FollowChange is false, the cursor remains on current record.

See Also

LogChanges (see page 8485)

CreateDataSet (see page 8401)

Undoing Changes

45.1.8.4.171 TCustomClientDataSet.UpdateIndexDefs Method

Pascal

```
procedure UpdateIndexDefs; override;
```

C++

```
virtual __fastcall UpdateIndexDefs();
```

Description

This is UpdateIndexDefs, a member of class TCustomClientDataSet.

45.1.8.4.172 TCustomClientDataSet.UpdateStatus Method

Reports the update status for the current record in the dataset.

Pascal

```
function UpdateStatus: TUpdateStatus; override;
```

C++

```
virtual __fastcall TUpdateStatus UpdateStatus();
```

Description

Call UpdateStatus to determine the update status for the current record in the client dataset. Update status can change frequently as records are edited, inserted, or deleted. UpdateStatus offers a convenient method for applications to assess the current status before undertaking or completing operations that depend on the update status of the current record.

Note: To determine whether there are updates pending for an entire dataset, check the ChangeCount (see page 8463) property.

See Also

ApplyUpdates (see page 8394)

CancelUpdates (see page 8396)

RevertRecord (see page 8443)

Indicating What Records Are Modified

Using the BDE to Cache Updates

Updating Data in dbExpress Applications

45.1.8.4.173 TCustomClientDataSet.WriteDataPacket Method

Pascal

```
procedure WriteDataPacket(Stream: TStream; WriteSize: Boolean; Format: TDataPacketFormat =
dfBinary);
```

C++

```
__fastcall WriteDataPacket(TStream Stream, Boolean WriteSize, TDataPacketFormat Format =
dfBinary);
```

Description

This is WriteDataPacket, a member of class TCustomClientDataSet.

45.1.8.5 TCustomClientDataSet Properties

The properties of the TCustomClientDataSet class are listed here.

Protected Properties

	Name	Description
	Aggregates (see page 8461)	Lists all aggregates that apply to the client dataset.
	AggregatesActive (see page 8461)	Specifies whether the client dataset calculates and maintains aggregate values.
	AutoCalcFields (see page 8463)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	CommandText (see page 8465)	Specifies an SQL command to be executed by the database server.
	Constraints (see page 8466)	Specifies record-level constraints that must be met when editing the data.
	DisableStringTrim (see page 8470)	Specifies whether trailing spaces are trimmed from field values when records are posted.
	DSBase (see page 8471)	Specifies the interface the client dataset uses to communicate with classes in Midas.dll or Midaslib.dcu (on Windows) or midas.so (on Linux).
R	DSCursor (see page 8471)	Provides access to the interface for the client dataset's data cursor.
	FetchOnDemand (see page 8471)	Indicates whether to fetch data packets from the provider on an as-needed basis.
	FieldDefs (see page 8472)	Points to the list of field definitions for the dataset.
	Filter (see page 8476)	Specifies the text of the current filter.
	Filtered (see page 8478)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 8478)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
	IndexDefs (see page 8480)	Contains information about the indexes for a client dataset.
	IndexName (see page 8483)	Identifies an index for the client dataset.
R	IsClone (see page 8484)	This is IsClone, a member of class TCustomClientDataSet.
	MasterSource (see page 8486)	Specifies a data source component for the master dataset when establishing a detail-master relationship between this dataset and another one.
	ObjectView (see page 8487)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	Params (see page 8489)	Contains parameter values that are sent to the provider.
	ProviderEOF (see page 8491)	Indicates whether the client dataset has all of its data.
R	Ranged (see page 8492)	This is Ranged, a member of class TCustomClientDataSet.
	ReadOnly (see page 8493)	Specifies whether the client dataset is read-only for this application.
	StoreDefs (see page 8495)	Indicates whether field and index definitions are saved with the client dataset.

Public Properties

	Name	Description
	Active (see page 8458)	Specifies whether the client dataset contains data.
R	ActiveAggs (see page 8460)	Lists the maintained aggregates supported by the current index.
	AppServer (see page 8462)	Provides access to the interface the client dataset uses to communicate with a provider.
R	ChangeCount (see page 8463)	Indicates the number of changes in the change log.
R	CloneSource (see page 8464)	Indicates the client dataset (if any) with which this client dataset shares data.
	ConnectionBroker (see page 8466)	Specifies a connection broker that handles the connection to an application server.
	Data (see page 8467)	Represents the data of the client dataset in a transportable format.
	DataSetField (see page 8469)	Indicates the TDataSetField component that represents this client dataset.
R	DataSize (see page 8469)	Indicates the number of bytes required to represent the Data (see page 8467) property.
R	Delta (see page 8470)	Represents a packet of change log data.
	FileName (see page 8475)	Specifies a file that stores the data of the client dataset.
R	GroupingLevel (see page 8479)	Indicates the depth of grouping support provided by the current index.
R	HasAppServer (see page 8480)	Indicates whether the client dataset is connected to a provider.
R	IndexFieldCount (see page 8481)	Indicates the number of fields that make up the current index.
	IndexFieldNames (see page 8482)	Lists the fields to use as an index.
	IndexFields (see page 8482)	Specifies the fields associated with the current index.
	KeyExclusive (see page 8484)	Specifies the upper and lower boundaries for a range.
	KeyFieldCount (see page 8484)	Specifies the number of fields to use when conducting a partial key search on a multi-field key.
R	KeySize (see page 8485)	Indicates the size of the current key used for the dataset.

	LogChanges (see page 8485)	Indicates whether to maintain a change log separate from the data.
	MasterFields (see page 8486)	Names one or more fields in a master table to link with corresponding fields in this dataset in order to establish a master-detail relationship.
	PacketRecords (see page 8488)	Indicates the number or type of records in a single data packet.
	ProviderName (see page 8492)	Specifies the name of a provider component that supplies data to and applies updates from the client dataset..
	RemoteServer (see page 8493)	Specifies the connection component that this client dataset uses to connect to an application server.
	SavePoint (see page 8494)	Represents the current state of edits when logging changes.
	StatusFilter (see page 8495)	Filters the client dataset so that it exposes only those records with a specific update status.
	XMLData (see page 8496)	Represents the data of the client dataset in XML.

45.1.8.5.1 TCustomClientDataSet.Active Property

Specifies whether the client dataset contains data.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Use Active to determine whether the client dataset makes its data available. When Active is false, the client dataset is closed; It can't manipulate the data it represents. When Active is true, data can be read or edited using the client dataset.

Setting Active to true:

1Fills the client dataset with data. Depending on the other properties of the client dataset, this data comes from:

The file specified by the FileName (see page 8475) property.

The provider specified by the ProviderName (see page 8492) property or SetProvider (see page 8451) method. (For some TCustomClientDataSet (see page 8373) descendants, this is an internal provider component)

The data that was active when the client dataset was deactivated (only if the application has run continuously since the dataset was deactivated).

2Triggers a BeforeOpen (see page 8518) event.

3Sets the State property to dsBrowse.

4Opens a cursor into the dataset.

5Triggers an After Open event.

If the client dataset has never been active previously, setting Active to true defines the structure (metadata) of the client dataset. This structure is retrieved with the data packet from the provider or stored on disk, or, if there is no data packet, it is built using the current value of the FieldDefs (see page 8472) property or from the persistent field components listed by the Fields property.

If an error occurs during the dataset open, the dataset state is set to dsInactive, and the cursor is closed.

Setting Active to false:

1Triggers a BeforeClose (see page 8512) event.

2Sets the State property to dsInactive.

3Closes the cursor, saving the current data to disk if the FileName (see page 8475) property is set, and saving the current

data packet to a cache so that it can be restored when the client dataset is reopened.

4Triggers an AfterClose (see page 8500) event.

Note: Calling the Open method sets Active to true; calling the Close method sets Active to false.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
value of the first field for the last record to the server so
that the server can return the correct records. The complete
project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
var
  LastValue: OleVariant;
  CDSClone: TClientDataSet;
begin
  if ClientDataSet1.Active then
  begin
    CDSClone := TClientDataSet.Create(Form1);
    try
      CDSClone.CloneCursor(ClientDataSet1, True);
      { turn off FetchOnDemand so that the clone only fetches
        the last LOCAL record }
      CDSClone.FetchOnDemand := False;
      CDSClone.Last;
      LastValue := CDSClone.Fields[0].AsString;
      CDSClone.Close;
    finally
      CDSClone.Free;
    end;
  end else
    LastValue := NULL;
  OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
  ClientDataSet1.GetNextPacket;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
```

```
    begin
      for I := 0 to DataSetCount - 1 do
        DataSets[I].EnableControls;
    end;
    ClientDataSet1.Open;
  end;

  procedure TForm1.Button3Click(Sender: TObject);
  var
    I: Integer;
  begin
    ClientDataSet1.Close;
    RemoteServer1.Connected := False;
    with RemoteServer1 do
    begin
      for I := 0 to DataSetCount - 1 do
        DataSets[I].DisableControls;
    end;
  end;

  procedure TForm1.CheckButtonActive(DataSet: TDataSet);
  begin
    Button1.Enabled := not ClientDataSet1.Active;
    Button2.Enabled := ClientDataSet1.Active;
    Button3.Enabled := ClientDataSet1.Active;
  end;
```

See Also

FileName (see page 8475)

Open (see page 6976)

Close (see page 6920)

State (see page 7036)

AfterOpen (see page 7045)

BeforeOpen (see page 7051)

FieldDefs (see page 7019)

Fields (see page 7023)

ProviderName (see page 8492)

45.1.8.5.2 TCustomClientDataSet.ActiveAggs Property

Lists the maintained aggregates supported by the current index.

Pascal

```
property ActiveAggs [Index: Integer]: TList;
```

C++

```
__property TList ActiveAggs[int Index];
```

Description

Read ActiveAggs to obtain a list of currently maintained aggregates for a specified grouping level. The Index parameter specifies whether the aggregates summarize data over the entire client dataset (Index = 0), over the current value of the first grouping level (Index = 1), over the current value of the second grouping level (Index = 2), and so on.

ActiveAggs is a List object (TList) whose items are a subset of the possible aggregates maintained by the Aggregates (see page 8461) property. If there are no aggregates maintained for the specified grouping level, the value of ActiveAggs is either an

empty list ornil (Delphi) or NULL (C++).

Note: Each grouping level is determined by the current index. Grouping level 1 is the set of records with the same value on the first field in the index. Grouping level 2 is the set of records with the same values on the first two fields in the index, and so on.

See Also

Aggregates (see page 8461)

AggregatesActive (see page 8461)

GroupingLevel (see page 8479)

IndexName (see page 8483)

45.1.8.5.3 TCustomClientDataSet.Aggregates Property

Lists all aggregates that apply to the client dataset.

Pascal

```
property Aggregates: TAggregates;
```

C++

```
__property TAggregates Aggregates;
```

Description

Use Aggregates to define aggregates that summarize the data in the client dataset. Aggregates is a collection of TAggregate (see page 8268) objects, each of which defines a formula for calculating an aggregate value from a group of records in the client dataset. The individual aggregates can summarize all the records in the client dataset or subgroups of records that have the same value on a set of fields. Aggregates that summarize a subgroup of records are associated with indexes, and can only be used when the associated index is current.

Do not use Aggregates to obtain summarized values maintained by the client dataset. Aggregates can contain TAggregate (see page 8268) objects that are not currently active. Instead, use the ActiveAggs (see page 8460) property, which represents only those TAggregate (see page 8268) objects that are currently active.

Note: You do not need to add a TAggregate (see page 8268) object to the Aggregates collection when you add an aggregate field using the Fields editor. The Fields editor automatically adds TAggregate (see page 8268) objects to Aggregates for every aggregate field you define.

See Also

ActiveAggs (see page 8460)

TAggregate (see page 8268)

IndexName (see page 8483)

TAggregateField (see page 6743)

Obtaining Aggregate Values

Specifying Aggregates

Defining an Aggregate Field

45.1.8.5.4 TCustomClientDataSet.AggregatesActive Property

Specifies whether the client dataset calculates and maintains aggregate values.

Pascal

```
property AggregatesActive: Boolean;
```

C++

```
__property Boolean AggregatesActive;
```

Description

Use AggregatesActive to indicate whether the client dataset should maintain aggregate values. When AggregatesActive is false (the default), the client dataset does not support maintained aggregates. This allows the client dataset to avoid the overhead of calculating values that the application is not using. When AggregatesActive is true, the client dataset calculates and maintains all aggregate values specified by the Aggregates (see page 8461) property that are compatible with the current index.

When aggregates are maintained, the Value method of every active aggregate object returns a value that reflects the current data in the client dataset. When users edit the data in the dataset, these values are recalculated to reflect the user's changes.

Note: To selectively enable and disable aggregates rather than turning them all on or off at once, use the Active (see page 8458) property of individual TAggregate (see page 8268) objects. These objects are available through the Aggregates (see page 8461) property.

See Also

ActiveAggs (see page 8460)

Aggregates (see page 8461)

IndexName (see page 8483)

Value (see page 8273)

Active (see page 8274)

45.1.8.5.5 TCustomClientDataSet.AppServer Property

Provides access to the interface the client dataset uses to communicate with a provider.

Pascal

```
property AppServer: IAppServer;
```

C++

```
__property IAppServer AppServer;
```

Description

Use AppServer to communicate with the provider that supplies data to and resolves updates for the client dataset. If the client dataset uses a provider in the same process space, AppServer is the interface to an internally created object that dispatches IAppServer calls to the local or internal provider. If the provider is on a remote application server, AppServer has the same value as the AppServer property of the connection component.

Applications don't need to work directly with the IAppServer interface. The properties and methods of the client dataset are automatically translated into IAppServer calls as needed.

See Also

HasAppServer (see page 8480)

AppServer (see page 8532)

SetProvider (see page 8451)

ProviderName (see page 8492)

TDataSetProvider (see page 18508)

Overriding the Dataset On the Application Server

Communicating with the Client Dataset

45.1.8.5.6 TCustomClientDataSet.AutoCalcFields Property

Determines when the OnCalcFields event is triggered and when lookup field values are calculated.

Pascal

```
property AutoCalcFields: Boolean;
```

C++

```
__property Boolean AutoCalcFields;
```

Description

Set AutoCalcFields to control when the OnCalcFields event is triggered to update calculated fields and when lookup fields are calculated.

A calculated field is one that derives its value from the values of one or more fields in the active record, sometimes with additional processing. Lookup fields are fields whose values come from a secondary dataset or lookup cache.

Note: Unidirectional datasets support calculated fields but not unidirectional fields. For unidirectional datasets, OnCalcFields only controls when calculated fields are updated.

When AutoCalcFields is true (the default), Lookup fields are recalculated and OnCalcFields is triggered when:

The dataset is opened.

The dataset is put into dsEdit state.

Focus moves from one visual control to another, or from one column to another in a data-aware grid and modifications have been made to the record.

When AutoCalcFields is false, Lookup fields are recalculated and the OnCalcFields event occurs only when

The dataset is opened.

The dataset is put into dsEdit state.

A record is retrieved from a database.

If an application permits users to change data, OnCalcFields is frequently triggered. In these cases an application may set AutoCalcFields to false to reduce the frequency with which the OnCalcFields event occurs and with which lookup values are fetched.

See Also

OnCalcFields (see page 7053)

Calculating Fields

45.1.8.5.7 TCustomClientDataSet.ChangeCount Property

Indicates the number of changes in the change log.

Pascal

```
property ChangeCount: Integer;
```

C++

```
__property int ChangeCount;
```

Description

Check (see page 8397) ChangeCount to determine how many modifications are pending for the dataset. ChangeCount increases when the data is edited. It is reset when updates are applied or merged and any differences are reconciled by the client dataset.

Delphi Examples:

```
{
This BeforeDisconnect event handler makes sure that the
client dataset has applied all its pending updates before
terminating the connection to the server.
}
procedure TForm1.RemoteServer1BeforeDisconnect(Sender: TObject);
begin
  if (ClientDataSet1.ChangeCount > 0) then
    ClientDataSet1.ApplyUpdates(-1);
end;
```

See Also

ApplyUpdates (see page 8394)

CancelUpdates (see page 8396)

Delta (see page 8470)

LogChanges (see page 8485)

MergeChangeLog (see page 8437)

UndoLastChange (see page 8455)

SavePoint (see page 8494)

Merging Changes into Data

Undoing Changes

Using the BDE to Cache Updates

Updating Data in dbExpress Applications

45.1.8.5.8 TCustomClientDataSet.CloneSource Property

Indicates the client dataset (if any) with which this client dataset shares data.

Pascal

```
property CloneSource: TCustomClientDataSet;
```

C++

```
__property TCustomClientDataSet CloneSource;
```

Description

Read CloneSource to access another client dataset that shares its data cursor with this one. CloneSource is set by the CloneCursor (see page 8398) method when the datasets are linked.

If the client dataset has not called CloneCursor (see page 8398) to share the data cursor of another client dataset, CloneSource is nil (Delphi) or NULL (C++).

See Also

CloneCursor (see page 8398)

45.1.8.5.9 TCustomClientDataSet.CommandText Property

Specifies an SQL command to be executed by the database server.

Pascal

```
property CommandText: WideString;
```

C++

```
__property BSTR CommandText;
```

Description

CommandText specifies what data the client dataset wants to receive from its (internal or external) provider. It is either

An SQL statement (query) for the database server to execute.

The name of a table or stored procedure.

If the client dataset uses an internal provider, CommandText is required. On TSQLClientDataSet, the CommandType property indicates whether it represents a query, table name, or stored procedure name.

If the client dataset uses an external provider component, CommandText represents a replacement for the property that the source dataset uses to identify its data (the SQL of a query, or the name of a table or stored procedure). If the provider's Options property does not include poAllowCommandText (which, by default, it does not), then CommandText has no effect.

When CommandText is a non-empty string, its value is sent to the provider when the client dataset opens or when you call the Execute (see page 8413) method. The specified SQL command, table, or stored procedure overrides the current value of the provider's associated dataset.

If the SQL statement takes parameters, be sure that they are in the correct order, as the provider performs parameter binding on CommandText statements by index only.

Delphi Examples:

```
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  // Memo1 must contain a "select * from xxx" employee, customer, client
  // Look in Program Files/Common Files/CodeGear Shared/Data for db files
  // Notice, using the CommandText, don't need a BeforeGetRecords event
  // handler for the ClientDataSet or the DataSetProvider.

  ClientDataSet1.CommandText := Memo1.Lines.Text;
  ClientDataSet1.Open;
```

```
end;
```

See Also

AS_Execute (see page 16453)

AS_GetRecords (see page 16455)

Options (see page 18440)

Execute (see page 8413)

Open (see page 6976)

DataSet (see page 18529)

ProviderName (see page 8492)

CommandType

Overriding the Dataset On the Application Server

Overview of Using Cached Updates

Setting Up a Simple Dataset

45.1.8.5.10 TCustomClientDataSet.ConnectionBroker Property

Specifies a connection broker that handles the connection to an application server.

Pascal

```
property ConnectionBroker: TConnectionBroker;
```

C++

```
__property TConnectionBroker ConnectionBroker;
```

Description

Use ConnectionBroker to let a connection broker handle the connection to the server that supplies the client dataset with data. The connection broker specified by ConnectionBroker simply sets the RemoteServer (see page 8493) property to the value of its Connection property.

ConnectionBroker adds an extra layer of indirection to the specification of a connection component. This can make it easier to change the component you use to connect to an application server. If you have several client datasets in an application that all use the same connection broker, then you can change the component that connects them all to the application server by changing a single property (the Connection property of the connection broker) rather than having to change the RemoteServer (see page 8493) property for every client dataset in the application.

See Also

RemoteServer (see page 8493)

Connection (see page 8368)

Specifying a Provider

Specifying the Provider using TLocalConnection or TConnectionBroker

45.1.8.5.11 TCustomClientDataSet.Constraints Property

Specifies record-level constraints that must be met when editing the data.

Pascal

```
property Constraints: TCheckConstraints;
```

C++

```
__property TCheckConstraints Constraints;
```

Description

Use Constraints to read or add record-level constraints to the dataset. Record-level constraints usually impose relationships between the fields in a single record. Constraints that limit the value of a single field should be added to the individual field components.

See Also

TCheckConstraint (see page 6841)

CustomConstraint (see page 7156)

ImportedConstraint (see page 7167)

Constraining Data Values

45.1.8.5.12 TCustomClientDataSet.Data Property

Represents the data of the client dataset in a transportable format.

Pascal

```
property Data: OleVariant;
```

C++

```
__property OleVariant Data;
```

Description

Data represents the client dataset's local, in-memory copy of its data, encoded as a data packet. This is the data in the binary format used internally by the client dataset.

This data packet can originate from a provider, a file on disk, or the Data property of another client data set.

Note: To obtain a copy of the client dataset's data as an XML data packet instead of in a binary format, use the XMLData (see page 8496) property.

Delphi Examples:

```
{
This example requires two ClientDataSets, two DBGrids, two
DataSources and two DBNavigators.  Name the two
ClientDataSets CDS and CDS2. Set the DataSet property in the
first DataSource to CDS and name that DataSource DS.  Set the
DataSet property in the second DataSource to CDS2 and name
that DataSource DS2.  Set the DataSource in the first
DBNavigator to DS and the DataSource in the second DBNavigator
to DS2.  Set the DataSource in the first DBGrid to DS and the
DataSource in the second DBGrid to DS2.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
```

```
type to Integer.  The file used to load CDS must have an
attribute name of HOST with an attribute type of string and
an attribute name of PORTA with an attribute type of integer.
Add DB, DBCtrls, Grids, DBGrids, DBClient and SwSystem to the
uses clause.  "LoadFormFile" to start with the file, or
"CreateDataSet" to start from scratch.  You must "CloseDataSet"
after a "LoadFromFile" before you can do another "CreateDataSet".
}
{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button1Click(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.SaveToFile(GetCurrentDir + '\CDS.XML', dfXML);
{$ELSE}
  CDS.SaveToFile(gsAppPath + 'CDS.XML', dfXML);
{$ENDIF}
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  CDS.CreateDataSet;
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  CDS2.Data := CDS.Data;
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  CDS.Close;
end;
```

See Also

Delta (see page 8470)

LoadFromFile (see page 8433)

Data (see page 18474)

ProviderName (see page 8492)

DataSize (see page 8469)

AS_GetRecords (see page 16455)

XMLData (see page 8496)

Applying Updates

Editing Data

Merging Changes into Data

Communicating with the Client Dataset

45.1.8.5.13 TCustomClientDataSet.DataSetField Property

Indicates the TDataSetField component that represents this client dataset.

Pascal

```
property DataSetField: TDataSetField;
```

C++

```
__property TDataSetField DataSetField;
```

Description

Use DataSetField when the client dataset is the nested detail set of another (master) client dataset. Read DataSetField to access the data set field component in the master dataset that comprises this client dataset. Set DataSetField to a field component in the master dataset to assign this client dataset as the value of that field component.

When setting DataSetField, the ProviderName (see page 8492), RemoteServer (see page 8493), and FileName (see page 8475) properties of the client dataset are cleared. This is because the client dataset's data is now maintained by the master table in which it is nested.

Note: The master dataset that includes the TDataSetField component must be another client dataset. Otherwise, setting DataSetField raises an exception. Further, setting DataSetField to a TDataSetField that is contained in the same client dataset leads to an exception (A dataset can't be nested in itself).

See Also

NestedDataSet (see page 7070)

ProviderName (see page 8492)

RemoteServer (see page 8493)

FileName (see page 8475)

45.1.8.5.14 TCustomClientDataSet.DataSize Property

Indicates the number of bytes required to represent the Data (see page 8467) property.

Pascal

```
property DataSize: Integer;
```

C++

```
__property int DataSize;
```

Description

Read DataSize to determine the number of bytes required for storing the client dataset's data. For example, before streaming the Data (see page 8467) property to a fixed size buffer using the SaveToStream (see page 8445) method, read DataSize to determine the necessary size of that buffer.

See Also

Data (see page 8467)

SaveToStream (see page 8445)

RecordSize

45.1.8.5.15 TCustomClientDataSet.Delta Property

Represents a packet of change log data.

Pascal

```
property Delta: OleVariant;
```

C++

```
__property OleVariant Delta;
```

Description

Use Delta to pass a packet of change log data from the client dataset to a provider. Delta contains only information about those records inserted, modified, or deleted through the client.

When the client dataset is linked to a provider, Delta is passed as an argument to the ApplyUpdates (see page 8394) and Reconcile (see page 8441) methods, which use the information in the change log to update the database. On return from successful application of updates, Delta is cleared. If update errors occur, the value of Delta after applying updates depends on the error tolerance allowed by the client dataset. This value is

- All change log data when the maximum number of tolerated errors is encountered and all changes are rolled back.
- Only those changes that could not be applied when fewer errors occurred than the maximum specified as a tolerance level.

In file-based applications, Delta is cleared when the changes are merged into the Data (see page 8467) property using the MergeChangeLog (see page 8437) method.

See Also

ApplyUpdates (see page 8394)

Data (see page 8467)

MergeChangeLog (see page 8437)

Reconcile (see page 8441)

AS_ApplyUpdates (see page 16451)

Applying Updates

Editing Data

45.1.8.5.16 TCustomClientDataSet.DisableStringTrim Property

Specifies whether trailing spaces are trimmed from field values when records are posted.

Pascal

```
property DisableStringTrim: Boolean;
```

C++

```
__property Boolean DisableStringTrim;
```

Description

When DisableStringTrim is false, trailing spaces are trimmed from field values when records are posted. When DisableStringTrim is true, field values are posted exactly as typed, including any trailing spaces.

45.1.8.5.17 TCustomClientDataSet.DSBase Property

Specifies the interface the client dataset uses to communicate with classes in Midas.dll or Midaslib.dcu (on Windows) or midas.so (on Linux).

Pascal

```
property DSBase: IDSBase;
```

C++

```
__property IDSBase DSBase;
```

Description

Applications do not use the protected DSBase property. Client datasets use this interface internally to implement database operations.

See Also

DSCursor (see page 8471)

45.1.8.5.18 TCustomClientDataSet.DSCursor Property

Provides access to the interface for the client dataset's data cursor.

Pascal

```
property DSCursor: IDSCursor;
```

C++

```
__property IDSCursor DSCursor;
```

Description

Applications do not use the protected DSCursor property. Client datasets obtain this interface when the Active (see page 8458) property is set to true. They use the interface for navigating, posting edits, filtering records, and so on.

See Also

DSBase (see page 8471)

Active (see page 7004)

45.1.8.5.19 TCustomClientDataSet.FetchOnDemand Property

Indicates whether to fetch data packets from the provider on an as-needed basis.

Pascal

```
property FetchOnDemand: Boolean;
```

C++

```
__property Boolean FetchOnDemand;
```

Description

By default FetchOnDemand is true, meaning that the client dataset fetches additional packets of data as needed (for example, as a user scrolls through data, or conducts a search). If the provider does not automatically include BLOB fields or nested detail sets in data packets, setting FetchOnDemand to true causes the client dataset to fetch these values when needed as well.

Set FetchOnDemand to false only if your application contains code that explicitly fetches data packets as needed using the GetNextPacket (see page 8423) method. Similarly, if FetchOnDemand is false and the provider does not include BLOB data

by default (Options includes poFetchBlobsOnDemand), the application must fetch BLOB fields explicitly using the FetchBlobs (see page 8414) method. If the provider does not include nested detail sets (Options includes poFetchDetailsOnDemand) and FetchOnDemand is false, the application must fetch nested details explicitly using the FetchDetails (see page 8414) method.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
value of the first field for the last record to the server so
that the server can return the correct records. The complete
project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
var
  LastValue: OleVariant;
  CDSClone: TClientDataSet;
begin
  if ClientDataSet1.Active then
  begin
    CDSClone := TClientDataSet.Create(Form1);
    try
      CDSClone.CloneCursor(ClientDataSet1, True);
      { turn off FetchOnDemand so that the clone only fetches
        the last LOCAL record }
      CDSClone.FetchOnDemand := False;
      CDSClone.Last;
      LastValue := CDSClone.Fields[0].AsString;
      CDSClone.Close;
    finally
      CDSClone.Free;
    end;
  end else
    LastValue := NULL;
  OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
  ClientDataSet1.GetNextPacket;
end;
```

See Also

GetNextPacket (see page 8423)

Options (see page 18440)

FetchBlobs (see page 8414)

FetchDetails (see page 8414)

Requesting Data from the Source Dataset or Document

45.1.8.5.20 TCustomClientDataSet.FieldDefs Property

Points to the list of field definitions for the dataset.

Pascal

```
property FieldDefs: TFieldDefs;
```

C++

```
__property TFieldDefs FieldDefs;
```

Description

FieldDefs lists the field definitions for a dataset. While an application can examine FieldDefs to explore the field definitions for a dataset, it should not change these definitions unless creating a new table with CreateTable or CreateDataSet.

To access fields and field values in a dataset, use the Fields, AggFields, and FieldValues properties, and the FieldsByName method.

Note: If the dataset includes object field descendants, FieldDefs represents a hierarchical view of the data, meaning that the definitions include object field definitions. To determine the definitions in a flattened view, use FieldDefList instead.

C++ Examples:

```
/*
The following code creates and activates a client dataset in
the form's OnCreate event handler.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TFieldDefs *pDefs = CDS2->FieldDefs;
  TFieldDef *pDef = pDefs->AddFieldDef();
  pDef->DataType = ftInteger;
  pDef->Name = "Field1";

  pDef = pDefs->AddFieldDef();
  pDef->DataType = ftString;
  pDef->Size = 10;
  pDef->Name = "Field2";

  TIndexDef *pIDef = CDS2->IndexDefs->AddIndexDef();
  pIDef->Fields = "Field1";
  pIDef->Name = "IntIndex";

  CDS2->CreateDataSet();
}
```

Delphi Examples:

```
{
The following code creates and activates a client dataset in
the form's OnCreate event handler.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  with CDS2 do
  begin
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftInteger;
      Name := 'Field1';
    end;
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftString;
      Size := 10;
      Name := 'Field2';
    end;
```

```
      with IndexDefs.AddIndexDef do
      begin
        Fields := 'Field1';
        Name := 'IntIndex';
      end;
      CreateDataSet;
    end;
  end;
  { The following example shows how to create a table. }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Table1.Close;
    Table1.DeleteTable;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    Table1:= TTable.Create(Form1);
    with Table1 do
    begin
      DatabaseName := 'DBDEMOS';
      TableType := ttParadox;
      TableName := 'CustInfo';
      Table1.Active := False;
      { Don't overwrite an existing table }
      if Table1.Exists then
        MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
      else
      begin
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
        { Next, describe the fields in the table }
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
```

```
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.Active:= True;
  end;
```

See Also

CreateTable (see page 10189)

Fields (see page 7023)

FieldValues (see page 7025)

ClearFields (see page 6919)

GetFieldList (see page 6953)

GetFieldNames (see page 6954)

FieldByName (see page 6936)

FindField (see page 6938)

SetFields (see page 6997)

ObjectView (see page 7031)

AggFields (see page 7006)

CreateDataSet (see page 8401)

Creating and Deleting Tables

45.1.8.5.21 TCustomClientDataSet.FileName Property

Specifies a file that stores the data of the client dataset.

Pascal

```
property FileName: string;
```

C++

```
__property AnsiString FileName;
```

Description

Set FileName if the client dataset always reads its data from and writes its data to a single file. When FileName represents a valid file name, the client dataset automatically writes its data to that file when it is closed. When FileName specifies the name of an existing file, the client dataset automatically reads its data from that file when it is opened.

Changes are not merged into the data when it is saved to the file. This allows an application that employs the "briefcase" model to set FileName without losing updates that are destined for a database server. Single-tiered applications should explicitly merge changes before closing the dataset and saving the data to disk.

Note: When client datasets write their data to a file, they overwrite the contents of the file. Any previous contents of the file are

lost.

To read from or write to a different file, use the LoadFromFile (see page 8433) and SaveToFile (see page 8443) methods instead. These methods can also be used to load and save data at times other than when the Active (see page 8458) property changes.

See Also

Open (see page 6976)

Close (see page 6920)

Active (see page 8458)

LoadFromFile (see page 8433)

SaveToFile (see page 8443)

Loading Data from a File or Stream

Saving Data to a File or Stream

Using a Dedicated File on Disk

45.1.8.5.22 TCustomClientDataSet.Filter Property

Specifies the text of the current filter.

Pascal

```
property Filter: string;
```

C++

```
__property AnsiString Filter;
```

Description

Use Filter to specify a dataset filter. When filtering is applied to a client dataset, only those records that meet a filter's conditions are available to an application. Filter contains the string that describes the filter condition. Client datasets support the following set of filter operations: See the Limiting what records appear link at the bottom of this topic for a complete list of available filter operations.

Operator or function ComparisonsExampleComment

Operator or function Comparisons	Example	Comment
=	State = 'CA'	
<>	State <> 'CA'	
>=	DateEntered >= '1/1/1998'	
<=	Total <= 100,000	
>	Percentile > 50	
<	Field1 < Field2	
BLANK	State <> 'CA' or State = BLANK	Blank records do not appear unless explicitly included in the filter.
IS NULL	Field1 IS NULL	
IS NOT NULL	Field1 IS NOT NULL	

Logical operators		
and	State = 'CA' and Country = 'US'	
or	State = 'CA' or State = 'MA'	
not	not (State = 'CA')	
Arithmetic operators		
+	Total + 5 > 100	Applies to numbers, strings, or date (time) + number
-	Field1 - 7 <> 10	Applies to numbers, dates, or date (time) - number
*	Discount * 100 > 20	Applies to numbers only
/	Discount > Total / 5	Applies to numbers only
String functions		
Upper	Upper(Field1) = 'ALWAYS'	
Lower	Lower(Field1 + Field2) = 'josp'	
Substring	Substring(DateString,8) = '1998' | Substring(DateString,1,3) = 'JAN'	Value goes from position of second argument to end or number of chars in third arg. First char has position 1.
Trim	Trim(Field1 + Field2) | Trim(Field1, '-')	removes third argument from front and back. If no third argument, trims spaces.
TrimLeft	TrimLeft(StringField) | TrimLeft(Field1, '$') <> ''	See Trim
TrimRight	TrimRight(StringField) | TrimRight(Field1, '.') <> ''	See Trim
DateTime functions		
Year	Year(DateField) = 2000	
Month	Month(DateField) <> 12	
Day	Day(DateField) = 1	
Hour	Hour(DateField) < 16	
Minute	Minute(DateField) = 0	
Second	Second(DateField) = 30	
GetDate	GetDate - DateField > 7	Represents current date and time
Date	DateField = Date(GetDate)	Returns the date portion of a datetime value
Time	TimeField > Time(GetDate)	Returns the time portion of a datetime value
Miscellaneous		
Like	Memo LIKE '%filters%'	Works like SQL-92 without the ESC clause. When applied to BLOB fields, FilterOptions (see page 8478) determines whether case is considered.
In	Day(DateField) in (1,7)	Works like SQL-92. Second argument is a list of values all with the same type.
*	State = 'M*'	Wildcard for partial comparisons

See Also

Filtered (see page 7027)

FilterOptions (see page 7027)

Limiting Which Records Appear

FindFirst (see page 6939)

FindLast (see page 6940)

FindNext (see page 6940)

FindPrior (see page 6941)

OnFilterRecord (see page 7055)

Filter (see page 7026)

45.1.8.5.23 TCustomClientDataSet.Filtered Property

Specifies whether or not filtering is active for a dataset.

Pascal

```
property Filtered: Boolean;
```

C++

```
__property Boolean Filtered;
```

Description

Check (see page 8397) Filtered to determine whether or not dataset filtering is in effect. If Filtered is true, then filtering is active. To apply filter conditions specified in the Filter (see page 8476) property or the OnFilterRecord (see page 8523) event handler, set Filtered to true. Because unidirectional datasets do not support filters, when you set Filtered to true on a unidirectional dataset, it raises an exception.

Note: When filtering is enabled, user edits to a record may mean that the record no longer meets a filter's test condition. The next time an attempt is made to the read the record from the dataset while the filter is in effect, the record is not visible. If that happens, the next record that passes the filter condition becomes the active record.

See Also

Filter (see page 7026)

FilterOptions (see page 7027)

OnFilterRecord (see page 7055)

Locate (see page 6970)

Lookup (see page 6970)

45.1.8.5.24 TCustomClientDataSet.FilterOptions Property

Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.

Pascal

```
property FilterOptions: TFilterOptions;
```

C++

```
__property TFilterOptions FilterOptions;
```

Description

Set FilterOptions to specify whether or not filtering is case insensitive when filtering on string or character fields, and whether or

not partial comparisons for matching filter conditions is allowed.

Note: Because unidirectional datasets do not support filters, when you set the FilterOptions property of a unidirectional dataset, it raises an exception.

By default, FilterOptions is set to an empty set. For filters based on string fields, set FilterOptions to foCaseInsensitive to catch all variations on a string regardless of capitalization.

When a string in a filter ends with an asterisk (*), it can be used to match partial strings. To disable matching of partial strings and to treat the asterisk as a literal character in string comparisons, set FilterOptions to include foNoPartialCompare.

See Also

Filter (see page 7026)

Filtered (see page 7027)

OnFilterRecord (see page 7055)

Locate (see page 6970)

Lookup (see page 6970)

45.1.8.5.25 TCustomClientDataSet.GroupingLevel Property

Indicates the depth of grouping support provided by the current index.

Pascal

```
property GroupingLevel: Integer;
```

C++

```
__property int GroupingLevel;
```

Description

Read GroupingLevel to determine the maximum level group for which grouping support is available. Client datasets provide grouping support in two ways:

The GetGroupState (see page 8420) method indicates where the current record falls within a group.

The ActiveAggs (see page 8460) property provides summary information over all the records in a group.

A group is defined as the set of records that have the same value on each of a set of fields. A level 0 group is the set of all records in the client dataset. A level 1 group is the set of records with the same value on the first field of the index. A level 2 group is the set of records with the same values on each of the first two fields of the index. A level n group is the set of records with the same values on each of the first n fields of the index.

By default, GroupingLevel is the default grouping level of the current index or the maximum grouping level of a maintained aggregate on the current index, whichever is greater. GroupingLevel can never be greater than IndexFieldCount (see page 8481).

See Also

IndexName (see page 8483)

GetGroupState (see page 8420)

ActiveAggs (see page 8460)

GroupingLevel (see page 7291)

IndexFieldCount (see page 8481)

45.1.8.5.26 TCustomClientDataSet.HasAppServer Property

Indicates whether the client dataset is connected to a provider.

Pascal

```
property HasAppServer: Boolean;
```

C++

```
__property Boolean HasAppServer;
```

Description

Read HasAppServer to determine whether the client dataset gets its data from an IAppServer interface. When HasAppServer is true, the client can interact with an internal provider, a local provider in the same process space, or a provider on a remote application server using the IAppServer interface.

Use HasAppServer to check whether the client dataset is connected rather than reading the AppServer (see page 8462) property. Reading AppServer (see page 8462) causes the client dataset to establish a connection and obtain an IAppServer interface if it does not already have one.

See Also

IAppServer (see page 16450)

AppServer (see page 8462)

ProviderName (see page 8492)

45.1.8.5.27 TCustomClientDataSet.IndexDefs Property

Contains information about the indexes for a client dataset.

Pascal

```
property IndexDefs: TIndexDefs;
```

C++

```
__property TIndexDefs IndexDefs;
```

Description

Examine IndexDefs for index information. IndexDefs maintains an array of TIndexDef items, each of which describes an available index for the dataset. Set the value of IndexDefs before calling CreateDataSet (see page 8401) to create a set of indexes with the client dataset.

Note: The index items in IndexDefs may not always reflect the current indexes available for a dataset. Before examining IndexDefs, call the IndexDefs.Update (Delphi) or IndexDefs->Update (C++) method to refresh the item list.

C++ Examples:

```
/*
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
*/
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2->Active = False;
  // Get the current available indices
```

```
  CDS2->IndexDefs->Update();
  // Find a field named "Field2"
  for (int I = 0; I < CDS2->IndexDefs->Count; I++)
    if (CDS2->IndexDefs->Items[I]->Fields == "Field2")
    {
      // set that index as the current index for the dataset}
      CDS2->IndexName = CDS2->IndexDefs->Items[I]->Name;
    }
  CDS2->Active = True;
}
```

Delphi Examples:

```
{
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
}
procedure TForm1.Button3Click(Sender: TObject);
var
  I : Integer;
begin
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2.Active := False;
  { Get the current available indices }
  CDS2.IndexDefs.Update;
  { Find a field named 'Field2' }
  for I := 0 to CDS2.IndexDefs.Count - 1 do
    if CDS2.IndexDefs.Items[I].Fields = 'Field2' then
    begin
      { set that index as the current index for the dataset}
      CDS2.IndexName := CDS2.IndexDefs.Items[I].Name;
    end;
  CDS2.Active := True;
end;
```

See Also

IndexFieldNames (see page 8482)

TIndexDef (see page 7283)

CreateDataSet (see page 8401)

Adding a New Index

45.1.8.5.28 TCustomClientDataSet.IndexFieldCount Property

Indicates the number of fields that make up the current index.

Pascal

```
property IndexFieldCount: Integer;
```

C++

```
__property int IndexFieldCount;
```

Description

Check (see page 8397) IndexFieldCount to determine how many fields are used by the current index for the dataset.

See Also

IndexFieldNames (see page 8482)

IndexDefs (see page 8480)

45.1.8.5.29 TCustomClientDataSet.IndexFieldNames Property

Lists the fields to use as an index.

Pascal

```
property IndexFieldNames: Widestring;
```

C++

```
__property BSTR IndexFieldNames;
```

Description

Use IndexFieldNames as an alternative method of specifying the index to use for a client dataset. Specify the name of each field on which to index the dataset, separating names with semicolons. Ordering of field names is significant.

Note: Indexes added using IndexFieldNames do not support grouping or maintained aggregates.

Tip: Use IndexFieldNames to create sort orders on the fly at runtime.

Note: The IndexFieldNames and IndexName (see page 8483) properties are mutually exclusive. Setting one clears the other.

See Also

IndexDefs (see page 8480)

IndexName (see page 8483)

Creating an Index with IndexFieldNames

Adding a New Index

45.1.8.5.30 TCustomClientDataSet.IndexFields Property

Specifies the fields associated with the current index.

Pascal

```
property IndexFields [Index: Integer]: TField;
```

C++

```
__property TField IndexFields[int Index];
```

Description

IndexFields is a zero-based array of field objects, each of which corresponds to a field in the current index. Index is an ordinal value indicating the position of a field in the index. The first field in the index is IndexFields[0], the second is IndexFields[1], and so on.

Note: Do not set IndexField directly. Instead use the IndexFieldNames (see page 8482) property to order datasets on the fly at runtime.

See Also

AddIndex (see page 8391)

IndexDefs (see page 8480)

IndexFieldNames (see page 8482)

IndexName (see page 8483)

45.1.8.5.31 TCustomClientDataSet.IndexName Property

Identifies an index for the client dataset.

Pascal

```
property IndexName: Widestring;
```

C++

```
__property BSTR IndexName;
```

Description

Use IndexName to specify an alternative index for a client dataset.If IndexName is empty, the dataset's sort order is based on the IndexFieldNames (see page 8482) property or on its default ordering in the data packet. Default ordering is determined by the predefined index, DEFAULT_ORDER.

If IndexName contains a valid index name, then that index is used to determine sort order of records.

Note: IndexFieldNames (see page 8482) and IndexName are mutually exclusive. Setting one clears the other.

C++ Examples:

```
/*
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
*/
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2->Active = False;
  // Get the current available indices
  CDS2->IndexDefs->Update();
  // Find a field named "Field2"
  for (int I = 0; I < CDS2->IndexDefs->Count; I++)
    if (CDS2->IndexDefs->Items[I]->Fields == "Field2")
    {
      // set that index as the current index for the dataset}
      CDS2->IndexName = CDS2->IndexDefs->Items[I]->Name;
    }
  CDS2->Active = True;
}
```

Delphi Examples:

```
{
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
}
procedure TForm1.Button3Click(Sender: TObject);
var
  I : Integer;
begin
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2.Active := False;
  { Get the current available indices }
  CDS2.IndexDefs.Update;
  { Find a field named 'Field2' }
  for I := 0 to CDS2.IndexDefs.Count - 1 do
```

```
      if CDS2.IndexDefs.Items[I].Fields = 'Field2' then
      begin
        { set that index as the current index for the dataset}
        CDS2.IndexName := CDS2.IndexDefs.Items[I].Name;
      end;
    CDS2.Active := True;
  end;
```

See Also

IndexFieldNames (see page 8482)

Deleting and Switching Indexes

45.1.8.5.32 TCustomClientDataSet.IsClone Property

Pascal

```
property IsClone: Boolean;
```

C++

```
__property Boolean IsClone;
```

Description

This is IsClone, a member of class TCustomClientDataSet.

45.1.8.5.33 TCustomClientDataSet.KeyExclusive Property

Specifies the upper and lower boundaries for a range.

Pascal

```
property KeyExclusive: Boolean;
```

C++

```
__property Boolean KeyExclusive;
```

Description

Use KeyExclusive to specify whether a range includes or excludes the records that match its specified starting and ending values. By default, KeyExclusive is false, meaning that matching values are included.

To restrict a range to those records that are greater than the specified starting value and less than the specified ending value, set KeyExclusive to true.

See Also

FindNearest (see page 8416)

GotoNearest (see page 8427)

SetRange (see page 8452)

SetRangeEnd (see page 8453)

SetRangeStart (see page 8454)

45.1.8.5.34 TCustomClientDataSet.KeyFieldCount Property

Specifies the number of fields to use when conducting a partial key search on a multi-field key.

Pascal

```
property KeyFieldCount: Integer;
```

C++

```
__property int KeyFieldCount;
```

Description

Use KeyFieldCount to limit a search based on a multi-field key to a consecutive subset of the fields in that key. For example, if the primary key for a dataset consists of three fields, a partial-key search can be conducted using only the first field in the key by setting KeyFieldCount to 1. Setting KeyFieldCount to 0 allows the client dataset to search on all key fields.

Note: Searches are only conducted based on consecutive key fields beginning with the first field in the key. For example if a key consists of three fields, an application can set KeyFieldCount to 1 to search on the first field, 2 to search on the first and second fields, or 3 to search on all fields. By default KeyFieldCount is initially set to include all fields in a search.

See Also

FindKey (see page 8416)

FindNearest (see page 8416)

GotoKey (see page 8427)

GotoNearest (see page 8427)

Locate (see page 8435)

45.1.8.5.35 TCustomClientDataSet.KeySize Property

Indicates the size of the current key used for the dataset.

Pascal

```
property KeySize: Word;
```

C++

```
__property Word KeySize;
```

Description

Examine KeySize to determine the size of the current key used for the client dataset. KeySize corresponds to the size of the key fields in the primary index for the dataset.

See Also

RecordSize

45.1.8.5.36 TCustomClientDataSet.LogChanges Property

Indicates whether to maintain a change log separate from the data.

Pascal

```
property LogChanges: Boolean;
```

C++

```
__property Boolean LogChanges;
```

Description

Use LogChanges to control whether changes made to the data in a client dataset are recorded. When LogChanges is true (the

default), all changes are recorded in the Delta (see page 8470) property. They can later be applied to a database or source dataset, backed out, or merged into the Data (see page 8467) property. When LogChanges is false, data changes are made directly to the Data (see page 8467) property and cannot be applied to a database or source dataset via a provider.

Warning: Never set LogChanges to false if the client dataset uses an internal provider unless the data is read-only. Setting LogChanges to false results in a client dataset that can't apply updates.

Tip: When using a client dataset in a file-based application where data changes made by the application are never applied to another dataset, set LogChanges to false to conserve system resources.

See Also

CreateDataSet (see page 8401)

MergeChangeLog (see page 8437)

UndoLastChange (see page 8455)

SavePoint (see page 8494)

Editing Data

Merging Changes into Data

45.1.8.5.37 TCustomClientDataSet.MasterFields Property

Names one or more fields in a master table to link with corresponding fields in this dataset in order to establish a master-detail relationship.

Pascal

```
property MasterFields: Widestring;
```

C++

```
__property BSTR MasterFields;
```

Description

Use MasterFields after setting the MasterSource (see page 8486) property to specify the names of one or more fields to use in establishing a detail-master relationship between this dataset and the one specified in MasterSource (see page 8486).

MasterFields is a string containing one or more field names in the master table. Separate field names with semicolons.

Each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this dataset for display.

Note: At design time, use the Field Link designer to establish the master-detail relationship between two datasets.

See Also

MasterSource (see page 8486)

Making the Table a Detail of Another Dataset

45.1.8.5.38 TCustomClientDataSet.MasterSource Property

Specifies a data source component for the master dataset when establishing a detail-master relationship between this dataset and another one.

Pascal

```
property MasterSource: TDataSource;
```

C++

```
__property TDataSource MasterSource;
```

Description

Use MasterSource to specify the name of the data source component whose DataSet property identifies a dataset to use as a master table in establishing a detail-master relationship between this dataset and another one.

At design time choose an available data source from the MasterSource property's drop-down list in the Object Inspector.

After setting the MasterSource property, specify which fields to use in the master table by setting the MasterFields (see page 8486) property. At runtime each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this dataset for display.

Note: At design time, use the Field Link designer to establish the master-detail relationship between two datasets.

Tip: MasterSource establishes a master/detail relationship using the traditional linked cursor approach. Alternately, client datasets can participate in master/detail relationships using nested datasets, where the detail dataset is stored with the master table's data. To assign a client dataset's data as the value of a nested detail, use the DataSetField (see page 8469) property.

See Also

MasterFields (see page 8486)

Making the Table a Detail of Another Dataset

45.1.8.5.39 TCustomClientDataSet.ObjectView Property

Specifies whether fields are stored hierarchically or flattened out in the Fields property.

Pascal

```
property ObjectView: Boolean;
```

C++

```
__property Boolean ObjectView;
```

Description

ObjectView affects the way the Fields property stores object fields and the way data-aware grids display ADT and array fields.

When ObjectView is true, the fields are stored hierarchically in the Fields property, meaning any child fields of an object field are referenced by the object field and don't appear sequentially after the object field in the TFields:: .Fields array. When ObjectView is false, the fields are stored sequentially in the Fields property, meaning any child fields of an object field are stored after the object field as siblings in the Fields array.

When ObjectView is false, object field types, such as TADTField, are not created. This switch is provided for increased compatibility with older data-aware controls, which may not handle object field types properly. The default is false for TBDEDataSet, and true for TClientDataSet.

Note: If the ObjectView property of the dataset provided to a TClientDataSet is false, then the TClientDataSet's ObjectView property always acts as false, since the data received is already flattened.

See Also

FieldDefList (see page 7019)

Fields (see page 7023)

Displaying ADT and Array Fields

Using Nested Detail Tables

Using Object Fields

Working with ADT Fields

Working with Array Fields

45.1.8.5.40 TCustomClientDataSet.PacketRecords Property

Indicates the number or type of records in a single data packet.

Pascal

```
property PacketRecords: Integer;
```

C++

```
__property int PacketRecords;
```

Description

Use PacketRecords to specify how many records should be contained in a single packet fetched from a provider, or to indicate that the packet returned should contain only metadata.

When a client dataset is instantiated, PacketRecords is automatically set to -1, meaning that a single packet should contain all records in the dataset.

If PacketRecords is greater than zero, then it specifies the number of records to return in a packet.

To retrieve the metadata for a dataset, set PacketRecords to 0. When PacketRecords is zero, the provider returns only information from its dataset that defines the database's structure, such as table, column, constraint, and domain definitions.

Warning: When PacketRecords is greater than 0, the client dataset must fetch its data incrementally. If the provider is on a stateless application server, the client dataset must inform the provider where to start fetching data. This communication can occur using a BeforeGetRecords (see page 8515) event handler.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
value of the first field for the last record to the server so
that the server can return the correct records. The complete
project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
var
  LastValue: OleVariant;
  CDSClone: TClientDataSet;
begin
  if ClientDataSet1.Active then
  begin
```

```
      CDSClone := TClientDataSet.Create(Form1);
      try
        CDSClone.CloneCursor(ClientDataSet1, True);
        { turn off FetchOnDemand so that the clone only fetches
          the last LOCAL record }
        CDSClone.FetchOnDemand := False;
        CDSClone.Last;
        LastValue := CDSClone.Fields[0].AsString;
        CDSClone.Close;
      finally
        CDSClone.Free;
      end;
    end else
      LastValue := NULL;
    OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
    ClientDataSet1.GetNextPacket;
  end;
```

See Also

FetchOnDemand (see page 8471)

GetNextPacket (see page 8423)

BeforeGetRecords (see page 8515)

Requesting Data from the Source Dataset or Document

Using the BDE to Cache Updates

Updating Data in dbExpress Applications

45.1.8.5.41 TCustomClientDataSet.Params Property

Contains parameter values that are sent to the provider.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Use Params to specify parameter values that the provider should pass to a source dataset. If the client dataset uses an internal provider, Params represents the values for any parameters used by the SQL statement specified by the CommandText (see page 8465) property. If the client dataset uses an external provider that is associated with a dataset that represents a query or stored procedure, these parameter values supply input parameter values for that query or stored procedure and return any output parameters of a stored procedure. If the provider is associated with a TTable or TSQLTable component, Params limits the records that are sent in data packets. After specifying the parameters (at design time or runtime), they are sent to the provider automatically when the client dataset fetches data or executes a query or stored procedure using its Execute (see page 8413) method.

To send parameter values to a query or stored procedure, for every parameter on the query or stored procedure for which the client dataset should send a value, add a parameter object (TParam) to Params. Assign each parameter object the same Name, DataType, and ParamType as the corresponding parameter on the query or stored procedure. These values are automatically encoded and sent to the provider when the client dataset fetches data.

To send parameter values to a TTable or TSQLTable, add a parameter object for each field in the table that is used to limit the values sent in data packets. Assign each parameter object the same Name and DataType as the corresponding field component on the table. These parameter values are automatically sent when the client dataset fetches data, and the records returned in data packets from the provider only contain records that match the assigned value on the corresponding fields. You can achieve the same affect by applying a filter to the field, but using Params is more efficient because the data need not be included in data packets.

Note: To initialize Params so that it reflects the parameters of a provider's associated query or stored procedure, call the FetchParams (see page 8415) method.

Warning: The parameters are only applied to the provider when a query or stored procedure is run the first time the client dataset fetches records. To change parameter values and force a query or stored procedure to be rerun with new values, use the Execute (see page 8413) method.

Delphi Examples:

```
{
This example fills in the parameters of a client dataset
from the entries of a list box.
}
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
  ListItem: string;
begin
  { Call FetchParams to ensure parameters reflect server metadata }
  ClientDataSet1.FetchParams;
  with ClientDataSet1.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
              Items[I].AsBoolean := True
            else
              Items[I].AsBoolean := False;
          end;
        ftFloat:
          Items[I].AsFloat := StrToFloat(ListItem);
        ftCurrency:
          Items[I].AsCurrency := StrToFloat(ListItem);
        ftBCD:
          Items[I].AsBCD := StrToCurr(ListItem);
        ftDate:
          Items[I].AsDate := StrToDate(ListItem);
        ftTime:
          Items[I].AsTime := StrToTime(ListItem);
        ftDateTime:
          Items[I].AsDateTime := StrToDateTime(ListItem);
```

```
      end;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Here is a string');
  ListBox1.Items.Add('4');
  ListBox1.Items.Add('3456789');
  ListBox1.Items.Add('34');
  ListBox1.Items.Add('0');
  ListBox1.Items.Add('12.56');
  ListBox1.Items.Add('4.23');
  ListBox1.Items.Add('99.5');
  ListBox1.Items.Add('4/12/53');
  ListBox1.Items.Add('7:45:00');
  ListBox1.Items.Add('4/12/53 7:45:00');

end;
```

See Also

Params (see page 10087)

Params (see page 10165)

Active (see page 8458)

FetchParams (see page 8415)

Params (see page 20197)

Parameters (see page 1220)

Getting Parameters from the Application Server

Limiting Records with Parameters

Overview of Using Cached Updates

Passing Parameters to the Source Dataset

Sending Query or Stored Procedure Parameters

Setting Up a Simple Dataset

45.1.8.5.42 TCustomClientDataSet.ProviderEOF Property

Indicates whether the client dataset has all of its data.

Pascal

```
property ProviderEOF: Boolean;
```

C++

```
__property Boolean ProviderEOF;
```

Description

ProviderEOF is used internally to check whether an associated provider has supplied the client dataset with its last record. If the client dataset fetches its data from a file or represents a nested detail dataset, ProviderEOF is always true. If the client dataset fetches its data from a provider, it sets ProviderEOF to true when it is certain that it has fetched the last data packet from the provider.

Note: When ProviderEOF is false, it is possible that the provider has supplied the last record but the client dataset has not yet

tried to obtain the next record. That is, when ProviderEOF is true, there is definitely no need to fetch more records, but when ProviderEOF is false, it is uncertain.

See Also

TDataSetProvider (see page 18508)

ProviderName (see page 8492)

AppendData (see page 8393)

45.1.8.5.43 TCustomClientDataSet.ProviderName Property

Specifies the name of a provider component that supplies data to and applies updates from the client dataset..

Pascal

```
property ProviderName: string;
```

C++

```
__property AnsiString ProviderName;
```

Description

Use ProviderName to specify an external provider object that links the client dataset to another (source) dataset or to an XML document. This property is passed as a parameter to IAppServer interface calls when the client fetches data from or applies updates to the source dataset, or its database server, or the source XML document.

The provider specified by ProviderName can reside in the same application as the client dataset or in an application server running on another system.

If the provider is in the same application as the client dataset, it must have the same Owner as the client dataset. To use a local provider that has a different Owner, use the SetProvider (see page 8451) method instead.

If the provider is on a remote application server, it must be exported by that application server. If the application server contains multiple provider objects, you can set ProviderName to any of them once the client dataset establishes an initial connection to the application server through its RemoteServer (see page 8493) property. If the connection component is a Web connection or socket connection, or if the application server is registered on the client system (using TRegsvr.exe), ProviderName can be set from a drop-down in the Object Inspector once the RemoteServer (see page 8493) property is set.

See Also

RemoteServer (see page 8493)

TDataSetProvider (see page 18508)

IAppServer (see page 16450)

SetProvider (see page 8451)

Specifying a Provider

Connecting a Client Dataset to Another Dataset in the Same Application

Using a Multi-Tiered Architecture

Communicating with the Client Dataset

45.1.8.5.44 TCustomClientDataSet.Ranged Property

Pascal

```
property Ranged: Boolean;
```

C++

```
__property Boolean Ranged;
```

Description

This is Ranged, a member of class TCustomClientDataSet.

45.1.8.5.45 TCustomClientDataSet.ReadOnly Property

Specifies whether the client dataset is read-only for this application.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use the ReadOnly property to prevent users from updating, inserting, or deleting data in the dataset. By default, ReadOnly is false, meaning users can potentially alter the dataset's data.

To guarantee that users cannot modify or add data to a dataset, set ReadOnly to true.

When ReadOnly is true, the dataset's CanModify property is false.

Note: Even if ReadOnly is false, users may not be able to modify or add data to a client dataset if it gets its data from a provider. The provider's Options property can limit whether data can be edited, and if so, what types of edits are allowed.

See Also

Active (see page 7004)

CanModify

Options (see page 18440)

45.1.8.5.46 TCustomClientDataSet.RemoteServer Property

Specifies the connection component that this client dataset uses to connect to an application server.

Pascal

```
property RemoteServer: TCustomRemoteServer;
```

C++

```
__property TCustomRemoteServer RemoteServer;
```

Description

If the client dataset is in the client portion of a multi-tiered application, use RemoteServer to specify the connection component that connects to the application server. The connection component locates the application server and obtains its IAppServer interface, which the client dataset can use for communicating with a remote provider.

Although it is not required, you can also set RemoteServer to a TLocalConnection component when working with a provider that resides in the same application.

If a client application uses more than connection component, the application can change which connection a client dataset uses by changing the value for this property.

Tip: If you have several client datasets that all use the same connection component and you think you may change the

connection component later (for example, to change the protocol), you can use the ConnectionBroker (see page 8466) property to centralize the connection.

See Also

ProviderName (see page 8492)

TSocketConnection (see page 18928)

TDCOMConnection (see page 16008)

TWebConnection (see page 18968)

AppServer (see page 8532)

IAppServer (see page 16450)

ConnectionBroker (see page 8466)

TLocalConnection (see page 23236)

TSoapConnection (see page 19864)

Specifying a Provider

Using a Multi-Tiered Architecture

Specifying the Provider using TLocalConnection or TConnectionBroker

45.1.8.5.47 TCustomClientDataSet.SavePoint Property

Represents the current state of edits when logging changes.

Pascal

```
property SavePoint: Integer;
```

C++

```
__property int SavePoint;
```

Description

Use SavePoint to save and later revert to an editing state. SavePoint is an integer that represents the current state of the change log. To use SavePoint to roll back the changes in the change log to an earlier state, set SavePoint to the value it had before the changes that should be rolled back.

Thus, when the client dataset is in a state you may want to return to, assign SavePoint to a variable:

```
BeforeChanges := ClientDataSet1.SavePoint;
BeforeChanges = ClientDataSet->SavePoint;
```

Later, to roll back all edits that were made since that point, assign the variable to SavePoint:

```
ClientDataSet1.SavePoint := BeforeChanges;
ClientDataSet->SavePoint = BeforeChanges;
```

Warning: Client datasets can't return to a SavePoint once the change log has been backed out to a prior state. This includes undoing changes by setting SavePoint to a previously saved value, backing out too many changes using the RevertRecord (see page 8443) method, or clearing the change log using CancelUpdates (see page 8396). Trying to set SavePoint to a state that is no longer available in the change log raises an exception.

See Also

Delta (see page 8470)

ChangeCount (see page 8463)

LogChanges (see page 8485)

UndoLastChange (see page 8455)

RevertRecord (see page 8443)

CancelUpdates (see page 8396)

Undoing Changes

45.1.8.5.48 TCustomClientDataSet.StatusFilter Property

Filters the client dataset so that it exposes only those records with a specific update status.

Pascal

```
property StatusFilter: TUpdateStatusSet;
```

C++

```
__property TUpdateStatusSet StatusFilter;
```

Description

Use StatusFilter to limit the client dataset to records that represent a specific update status. This can be useful when processing updates in a provider's event handlers (or, if the client dataset has an internal provider, in its own BeforeUpdateRecord event handler).

StatusFilter is a set of TUpdateStatus values. It can indicate more than one type of update. If StatusFilter is an empty set (the default), the client dataset appears as if all updates in the change log have been incorporated.

Including usUnmodified in StatusFilter has no effect.

See Also

UpdateStatus (see page 8456)

BeforeUpdateRecord

Indicating What Records Are Modified

Undoing Changes

Using the BDE to Cache Updates

Responding to Client Update Requests

Updating Data in dbExpress Applications

45.1.8.5.49 TCustomClientDataSet.StoreDefs Property

Indicates whether field and index definitions are saved with the client dataset.

Pascal

```
property StoreDefs: Boolean;
```

C++

```
__property Boolean StoreDefs;
```

Description

If StoreDefs is true, the client dataset's index and field definitions are stored with the client dataset when the form or data module that contains it is saved. Setting StoreDefs to true makes the CreateDataSet (see page 8401) method into a one-step procedure that creates fields and indexes at runtime.

See Also

FieldDefs (see page 7019)

IndexDefs (see page 8480)

CreateDataSet (see page 8401)

45.1.8.5.50 TCustomClientDataSet.XMLData Property

Represents the data of the client dataset in XML.

Pascal

```
property XMLData: string;
```

C++

```
__property AnsiString XMLData;
```

Description

XMLData represents the client dataset's local, in-memory copy of its data, encoded as an XML data packet. This is the same as the Data (see page 8467) property, except that the data is represented in XML rather than the binary format used internally by the client dataset.

See Also

Data (see page 8467)

45.1.8.6 TCustomClientDataSet Events

The events of the TCustomClientDataSet class are listed here.

Protected Events

	Name	Description
	AfterApplyUpdates (see page 8497)	Occurs after the client dataset applies updates to the provider but before it attempts to reconcile update errors.
	AfterCancel (see page 8498)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 8500)	Occurs after an application closes a dataset.
	AfterDelete (see page 8501)	Occurs after an application deletes a record.
	AfterEdit (see page 8504)	Occurs after an application starts editing a record.
	AfterExecute (see page 8504)	Occurs after the client dataset receives the results from an execute command.
	AfterGetParams (see page 8505)	Occurs after the client dataset receives parameter values from the provider.
	AfterGetRecords (see page 8506)	Occurs after the client dataset fetches a data packet from the provider.
	AfterInsert (see page 8507)	Occurs after an application inserts a new record.
	AfterOpen (see page 8507)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 8509)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 8509)	Occurs after an application refreshes the data in the dataset.
	AfterRowRequest (see page 8509)	Occurs after the client dataset fetches new information about the current record.
	AfterScroll (see page 8510)	Occurs after an application scrolls from one record to another.
	BeforeApplyUpdates (see page 8511)	Occurs before the client dataset applies updates to the remote server.
	BeforeCancel (see page 8512)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 8512)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 8512)	Occurs before an application attempts to delete the active record.

	BeforeEdit (see page 8513)	Occurs before an application enters edit mode for the active record.
	BeforeExecute (see page 8514)	Occurs before the client dataset executes the provider's query or stored procedure.
	BeforeGetParams (see page 8514)	Occurs before the client dataset fetches parameter values from the provider.
	BeforeGetRecords (see page 8515)	Occurs before the client dataset fetches a data packet from the provider.
	BeforeInsert (see page 8517)	Occurs before an application enters insert mode.
	BeforeOpen (see page 8518)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 8518)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 8519)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeRowRequest (see page 8520)	Occurs before the client dataset fetches new information about the current record.
	BeforeScroll (see page 8520)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 8521)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 8522)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 8522)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 8523)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 8523)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 8524)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnReconcileError (see page 8524)	Occurs when a client dataset needs to reconcile an update to a record that could not be applied.

45.1.8.6.1 TCustomClientDataSet.AfterApplyUpdates Event

Occurs after the client dataset applies updates to the provider but before it attempts to reconcile update errors.

Pascal

```
property AfterApplyUpdates: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterApplyUpdates;
```

Description

Write an AfterApplyUpdates event handler to respond to custom information returned to the client dataset from a provider's AfterApplyUpdates event handler. AfterApplyUpdates is part of the mechanism by which a client dataset and a provider communicate information about updates. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's ApplyUpdates (see page 8394) method, the following events occur:

1.The client dataset receives a BeforeApplyUpdates (see page 8511) event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeApplyUpdates (see page 8511) event, where it can respond to or change that information before it applies the updates.

3.The provider applies the updates, and receives an OnUpdateError event for any problem records.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates (see page 8511) event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider in its AfterApplyUpdates event handler.

6.The client dataset generates an OnReconcileError (see page 8524) for every record that could not be applied by the provider.

See Also

ApplyUpdates (see page 8394)

AS_ApplyUpdates (see page 16451)

applyupdates (see page 18459)

AfterApplyUpdates (see page 18475)

BeforeApplyUpdates (see page 8511)

OnReconcileError (see page 8524)

Applying Updates

Communicating with the Client Dataset

45.1.8.6.2 TCustomClientDataSet.AfterCancel Event

Occurs after an application completes a request to cancel modifications to the active record.

Pascal

```
property AfterCancel: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterCancel;
```

Description

Write an AfterCancel event handler to take specific action after an application cancels changes to the active record. AfterCancel is called by the Cancel method after it updates the current position, releases the lock on the active record if necessary, and sets the dataset state to dsBrowse. If an application requires additional processing before returning control to a user after a Cancel event, code it in the AfterCancel event.

Delphi Examples:

```
{
This example updates the form's status bar with a message
when an AfterCancel event occurs.  Set the StatusBar
SimplePanel property to True.
}
procedure TForm1.Table1AfterCancel(DataSet: TDataSet);
begin
  StatusBar1.SimpleText := 'Record changes cancelled for ' + DataSet.Name;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Table1.Cancel;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  Table1.Post
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
```

```
      DatabaseName := 'DBDEMOS';
      TableType := ttParadox;
      TableName := 'CustInfo';
      Name := 'CustInfo';
      Table1.Active := False;
      { Don't overwrite an existing table }
      if Table1.Exists then
      begin
          Table1.Close;
          Table1.DeleteTable;
      end;
      begin
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
        { Next, describe the fields in the table }
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.AfterCancel:= Table1AfterCancel;
    Table1.Active:= True;
  end;
```

See Also

BeforeCancel (see page 7048)

Cancel (see page 6916)

State (see page 7036)

45.1.8.6.3 TCustomClientDataSet.AfterClose Event

Occurs after an application closes a dataset.

Pascal

```
property AfterClose: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterClose;
```

Description

Write an AfterClose event handler to take specific action immediately after an application closes a dataset. For example, as a security measure, an application might clear a PASSWORD entry from the Params property of a TSQLConnection component when the dataset is closed.

AfterClose is called after a dataset is closed and the dataset state is set to dsInactive.

Delphi Examples:

```
{
This example adds an entry to a memo with a message when an
AfterOpen event or an AfterClose event occurs.  Populate the
ClientDataSet events with CheckButtonActive.
}
procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  if (Button1.Enabled) then
    Memo2.Lines.Add('Now that the dataset is open, record information is available.')
  else
    Memo2.Lines.Add('Now that the dataset is closed, record information is not available.');
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
```

```
    ClientDataSet1.Close;
    RemoteServer1.Connected := False;
    with RemoteServer1 do
    begin
      for I := 0 to DataSetCount - 1 do
        DataSets[I].DisableControls;
    end;
  end;

  procedure TForm1.CheckButtonActive(DataSet: TDataSet);
  begin
    Button1.Enabled := not ClientDataSet1.Active;
    Button2.Enabled := ClientDataSet1.Active;
    Button3.Enabled := ClientDataSet1.Active;
  end;
```

See Also

Active (see page 7004)

State (see page 7036)

BeforeClose (see page 7048)

Close (see page 6920)

Params (see page 20253)

Opening and Closing Datasets

45.1.8.6.4 TCustomClientDataSet.AfterDelete Event

Occurs after an application deletes a record.

Pascal

```
property AfterDelete: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterDelete;
```

Description

Write an AfterDelete event handler to take specific action immediately after an application deletes the active record in a dataset. AfterDelete is called by Delete after it deletes the record, sets the dataset state to dsBrowse, and repositions the current record.

C++ Examples:

```
/*
This example displays a message on the form's status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
*/
void __fastcall TForm1::Table1AfterDelete(TDataSet *DataSet)
{
  StatusBar1->SimpleText = Format(
    "There are now %d records in the table",
    ARRAYOFCONST(((int)DataSet->RecordCount)));
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Table1->Delete();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  Table1 = new TTable(Form1);
  Table1->Active = false; // The Table component must not be active
  Table1->DatabaseName = "DBDEMOS";
  Table1->TableType = ttParadox;
  Table1->TableName = "CustInfo";
  Table1->Active = False;
  if (!Table1->Exists) // Don't overwrite an existing table
  {
    Table1->Close();
    Table1->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Table1->FieldDefs->Clear();
  TFieldDef *pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Table1->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Table1->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Table1->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Table1->CreateTable();
  Table1->Active = True;
  for (int i = 1; i <= 20; i++)
    Table1->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  DS2->DataSet = Table1;
  DBGrid2->DataSource->DataSet = Table1;
  Table1->AfterDelete = Table1AfterDelete;
  Table1->Active = True;
}
```

Delphi Examples:

```
{
This example displays a message on the form’s status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
}
procedure TForm1.Table1AfterDelete(DataSet: TDataSet);
begin
  StatusBar1.SimpleText := Format('There are now %d records in the table',
[DataSet.RecordCount]);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Delete;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
```

```
      TableType := ttParadox;
      TableName := 'CustInfo';
      Name := 'CustInfo';
      Table1.Active := False;
      { Don't overwrite an existing table }
      if Table1.Exists then
      begin
          Table1.Close;
          Table1.DeleteTable;
      end;
      begin
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
        { Next, describe the fields in the table }
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.AfterDelete:= Table1AfterDelete;
    Table1.Active:= True;
  end;
```

See Also

State (see page 7036)

BeforeDelete (see page 7049)

Delete (see page 6924)

45.1.8.6.5 TCustomClientDataSet.AfterEdit Event

Occurs after an application starts editing a record.

Pascal

```
property AfterEdit: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterEdit;
```

Description

Write an AfterEdit event handler to take specific action immediately after dataset enters edit mode. AfterEdit is called by Edit after it enables editing of a record, recalculates calculated fields, and calls the data event handler to process a record change.

Delphi Examples:

```
{
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

BeforeEdit (see page 7049)

Edit (see page 6932)

45.1.8.6.6 TCustomClientDataSet.AfterExecute Event

Occurs after the client dataset receives the results from an execute command.

Pascal

```
property AfterExecute: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterExecute;
```

Description

Write an AfterExecute event handler to respond to custom information returned to the client dataset from a provider's

AfterExecute event handler. AfterExecute is part of the mechanism by which a client dataset and a provider communicate information when executing an SQL command from the client. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's Execute (see page 8413) method, the following events occur:

1.The client dataset receives a BeforeExecute (see page 8514) event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeExecute (see page 8514) event, where it can respond to or change that information before it executes the query or stored procedure.

3.The provider executes the appropriate command (either the command supplied by the CommandText (see page 8465) property, or the command associated with a dataset such as a query or stored procedure).

4.The provider receives an AfterExecute event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeExecute (see page 8514) event handler.

5.The client dataset unpacks any output parameters from the provider into the Params (see page 8489) property and then receives an AfterExecute event, where it can respond to the custom information returned by the provider's AfterExecute event handler.

See Also

Execute (see page 8413)

AS_Execute (see page 16453)

Execute (see page 18466)

AfterExecute (see page 18476)

BeforeExecute (see page 8514)

CommandText (see page 8465)

Communicating with the Client Dataset

45.1.8.6.7 TCustomClientDataSet.AfterGetParams Event

Occurs after the client dataset receives parameter values from the provider.

Pascal

```
property AfterGetParams: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterGetParams;
```

Description

Write an AfterGetParams event handler to respond to custom information returned to the client dataset from a provider's AfterGetParams event handler. AfterGetParams is part of the mechanism by which a client dataset and a provider communicate information when the client application calls FetchParams (see page 8415). When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's FetchParams (see page 8415) method, the following events occur:

1.The client dataset receives a BeforeGetParams (see page 8514) event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeGetParams (see page 8514) event, where it can respond to or change that information before it retrieves parameter values.

3.The provider encodes all of its dataset's current parameter values into an OleVariant.

4.The provider receives an AfterGetParams event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetParams (see page 8514) event handler.

5.The client dataset unpacks the parameters from the provider into the Params (see page 8489) property and then receives an AfterGetParams event, where it can respond to the custom information returned by the provider's AfterGetParams event handler.

See Also

FetchParams (see page 8415)

AS_GetParams (see page 16454)

GetParams (see page 18467)

aftergetparams (see page 18477)

BeforeGetParams (see page 8514)

Communicating with the Client Dataset

45.1.8.6.8 TCustomClientDataSet.AfterGetRecords Event

Occurs after the client dataset fetches a data packet from the provider.

Pascal

```
property AfterGetRecords: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterGetRecords;
```

Description

Write an AfterGetRecords event handler to respond to custom information returned to the client dataset from a provider's AfterGetRecords event handler. AfterGetRecords is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches data from the application server, the following events occur:

1. The client dataset receives a BeforeGetRecords (see page 8515) event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.
2. The provider receives a BeforeGetRecords (see page 8515) event, where it can respond to or change that information before it creates a data packet.
3. The provider generates an OnGetData event where it can edit the data before creating a data packet, and then generates the data packet.
4. The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords (see page 8515) event handler.
5. The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider's AfterGetRecords event handler.

See Also

AS_GetRecords (see page 16455)

GetRecords (see page 18468)

AfterGetRecords (see page 18477)

BeforeGetRecords (see page 8515)

Communicating with the Client Dataset

45.1.8.6.9 TCustomClientDataSet.AfterInsert Event

Occurs after an application inserts a new record.

Pascal

```
property AfterInsert: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterInsert;
```

Description

Write an AfterInsert event handler to take specific action immediately after an application inserts a record. The Insert and Append methods generate an AfterInsert event after inserting or appending a new record.

Delphi Examples:

```
{
This example updates a memo with a message when an
AfterInsert event or a BeforeInsert event occurs.
}
procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After inserting a record');
end;

procedure TForm1.ClientDataSet1BeforeInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before inserting a record');
end;
```

See Also

BeforeInsert (see page 7050)

Append (see page 6910)

Insert (see page 6959)

45.1.8.6.10 TCustomClientDataSet.AfterOpen Event

Occurs after an application completes opening a dataset and before any data access occurs.

Pascal

```
property AfterOpen: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterOpen;
```

Description

Write an AfterOpen event handler to take specific action immediately after an application opens the dataset. AfterOpen is called after the dataset establishes access to its data and the dataset is put into dsBrowse state. For example, an AfterOpen event handler might check an ini file to determine the last record touched in the dataset the previous time the application ran, and position the dataset at that record.

Delphi Examples:

```
{
This example adds an entry to a memo with a message when an
```

```
AfterOpen event or an AfterClose event occurs.  Populate the
ClientDataSet events with CheckButtonActive.
}
procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  if (Button1.Enabled) then
    Memo2.Lines.Add('Now that the dataset is open, record information is available.')
  else
    Memo2.Lines.Add('Now that the dataset is closed, record information is not available.');
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
  RemoteServer1.Connected := False;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].DisableControls;
  end;
end;

procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
```

See Also

Active (see page 7004)

State (see page 7036)

BeforeOpen (see page 7051)

Open (see page 6976)

Opening and Closing Datasets

45.1.8.6.11 TCustomClientDataSet.AfterPost Event

Occurs after an application writes the active record to the database or change log and returns to browse state.

Pascal

```
property AfterPost: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterPost;
```

Description

Write an AfterPost event handler to take specific action immediately after an application posts a change to the active record. AfterPost is called after a modification or insertion is made to a record.

Delphi Examples:

```
{
This example updates the form's status bar with a message
when an AfterPost event occurs.
}
procedure TForm1.ClientDataSet1AfterPost(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before posting a record');
end;
```

See Also

BeforePost (see page 7052)

Post (see page 6977)

45.1.8.6.12 TCustomClientDataSet.AfterRefresh Event

Occurs after an application refreshes the data in the dataset.

Pascal

```
property AfterRefresh: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterRefresh;
```

Description

Write an AfterRefresh event handler to take specific action immediately after an application has updated the records in the dataset. AfterRefresh is generated by calls to the Refresh method.

See Also

BeforeRefresh (see page 7052)

Refresh (see page 6991)

45.1.8.6.13 TCustomClientDataSet.AfterRowRequest Event

Occurs after the client dataset fetches new information about the current record.

Pascal

```
property AfterRowRequest: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterRowRequest;
```

Description

Write an AfterRowRequest event handler to respond to custom information returned to the client dataset from a provider's AfterRowRequest event handler. AfterRowRequest is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches detail information about the current record or refreshes the current record, the following events occur:

1.The client dataset receives a BeforeRowRequest (see page 8520) event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeRowRequest (see page 8520) event, where it can respond to or change that information before it looks up the requested information.

3.The provider creates a delta packet that contains the requested information.

4.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeRowRequest (see page 8520) event handler.

5.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider's AfterRowRequest event handler.

See Also

AS_RowRequest (see page 16456)

RowRequest (see page 18472)

AfterRowRequest (see page 18478)

BeforeRowRequest (see page 8520)

FetchBlobs (see page 8414)

FetchParams (see page 8415)

RefreshRecord (see page 8442)

Communicating with the Client Dataset

45.1.8.6.14 TCustomClientDataSet.AfterScroll Event

Occurs after an application scrolls from one record to another.

Pascal

```
property AfterScroll: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterScroll;
```

Description

Write an AfterScroll event handler to take specific action immediately after an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. AfterScroll is called after all other events triggered by these methods and any other methods that switch from record to record in the dataset.

See Also

BeforeScroll (see page 7053)

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Prior (see page 6979)

Navigating Datasets

45.1.8.6.15 TCustomClientDataSet.BeforeApplyUpdates Event

Occurs before the client dataset applies updates to the remote server.

Pascal

```
property BeforeApplyUpdates: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeApplyUpdates;
```

Description

Write a BeforeApplyUpdates event handler to send custom information to the client dataset's provider. BeforeApplyUpdates is part of the mechanism by which a client dataset and a provider communicate information about updates. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's ApplyUpdates (see page 8394) method, the following events occur:

1.The client dataset receives a BeforeApplyUpdates event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeApplyUpdates event, where it can respond to or change that information before it applies the updates.

3.The provider applies the updates and receives an OnUpdateError event for any problem records.

4.The provider receives an AfterApplyUpdates (see page 8497) event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates (see page 8497) event, where it can respond to the custom information returned by the provider's AfterApplyUpdates (see page 8497) event handler.

See Also

ApplyUpdates (see page 8394)

AS_ApplyUpdates (see page 16451)

applyupdates (see page 18459)

BeforeApplyUpdates (see page 18479)

AfterApplyUpdates (see page 8497)

Applying Updates

Communicating with the Client Dataset

45.1.8.6.16 TCustomClientDataSet.BeforeCancel Event

Occurs before an application executes a request to cancel changes to the active record.

Pascal

```
property BeforeCancel: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeCancel;
```

Description

Write a BeforeCancel event to take specific action before an application carries out a request to cancel changes. BeforeCancel is called by the Cancel method before it cancels a dataset operation such as Edit, Insert, or Delete.

An application might use the BeforeCancel event to record a user's changes in an undo buffer.

See Also

AfterCancel (see page 7038)

Delete (see page 6924)

Edit (see page 6932)

Insert (see page 6959)

45.1.8.6.17 TCustomClientDataSet.BeforeClose Event

Occurs immediately before the dataset closes.

Pascal

```
property BeforeClose: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeClose;
```

Description

Write a BeforeClose event to take specific action before an application closes a dataset. Calling Close or setting the Active property to false results in a call to the BeforeClose event handler.

See Also

Active (see page 7004)

AfterClose (see page 7040)

Close (see page 6920)

Opening and Closing Datasets

Posting Data

45.1.8.6.18 TCustomClientDataSet.BeforeDelete Event

Occurs before an application attempts to delete the active record.

Pascal

```
property BeforeDelete: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeDelete;
```

Description

Write a BeforeDelete event handler to take specific action before an application deletes the active record. BeforeDelete is called by Delete before it actually deletes a record.

In Delphi, making use of this event an application might, for example, display a dialog box asking for confirmation before deleting the record. On denial of confirmation, the application could abort the deletion by calling the Abort procedure.

See Also

AfterDelete (see page 7041)

Delete (see page 6924)

Abort (see page 22786)

45.1.8.6.19 TCustomClientDataSet.BeforeEdit Event

Occurs before an application enters edit mode for the active record.

Pascal

```
property BeforeEdit: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeEdit;
```

Description

Write a BeforeEdit event handler to take specific action before an application enables editing of the active record. For example, an application that keeps a log of database edits could use the BeforeEdit event to record the edit request, time, and user before entering edit state.

Delphi Examples:

```
{
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

State (see page 7036)

AfterEdit (see page 7044)

Edit (see page 6932)

45.1.8.6.20 TCustomClientDataSet.BeforeExecute Event

Occurs before the client dataset executes the provider's query or stored procedure.

Pascal

```
property BeforeExecute: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to send custom information to the client dataset's provider. BeforeExecute is part of the mechanism by which a client dataset and a provider communicate information when explicitly executing an SQL command by the provider's dataset. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's Execute (see page 8413) method, the following events occur:

1.The client dataset receives a BeforeExecute event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeExecute event, where it can respond to or change that information before it executes the query or stored procedure.

3.The provider executes the appropriate command (either the command supplied by the CommandText (see page 8465) property, or the command associated with a dataset such as a query or stored procedure).

4.The provider receives an AfterExecute (see page 8504) event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeExecute event handler.

5.The client dataset unpacks any output parameters from the provider into the Params (see page 8489) property and then receives an AfterExecute (see page 8504) event, where it can respond to the custom information returned by the provider's AfterExecute (see page 8504) event handler.

See Also

Execute (see page 8413)

AS_Execute (see page 16453)

Execute (see page 18466)

beforeexecute (see page 18479)

AfterExecute (see page 8504)

CommandText (see page 8465)

Communicating with the Client Dataset

45.1.8.6.21 TCustomClientDataSet.BeforeGetParams Event

Occurs before the client dataset fetches parameter values from the provider.

Pascal

```
property BeforeGetParams: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeGetParams;
```

Description

Write a BeforeGetParams event handler to send custom information to the client dataset's provider. BeforeGetparams is part of the mechanism by which a client dataset and a provider communicate information when passing parameter values from the provider's dataset. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's FetchParams (see page 8415) method, the following events occur:

1.The client dataset receives a BeforeGetParams event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeGetParams event, where it can respond to or change that information before it retrieves parameter values.

3.The provider encodes all of its dataset's current parameter values into an OleVariant.

4.The provider receives an AfterGetParams (see page 8505) event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetParams event handler.

5.The client dataset unpacks the parameters from the provider into the Params (see page 8489) property and then receives an AfterGetParams (see page 8505) event, where it can respond to the custom information returned by the provider's AfterGetParams (see page 8505) event handler.

See Also

FetchParams (see page 8415)

AS_GetParams (see page 16454)

GetParams (see page 18467)

BeforeGetParams (see page 18480)

AfterGetParams (see page 8505)

Communicating with the Client Dataset

45.1.8.6.22 TCustomClientDataSet.BeforeGetRecords Event

Occurs before the client dataset fetches a data packet from the provider.

Pascal

```
property BeforeGetRecords: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeGetRecords;
```

Description

Write a BeforeGetRecords event handler to send custom information to the client dataset's provider. BeforeGetRecords is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches data from the application server, the following events occur:

1.The client dataset receives a BeforeGetRecords event, where it can encode custom information into an OleVariant that is

passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeGetRecords event, where it can respond to or change that information before it creates a data packet.

3.The provider generates the data packet.

4.The provider receives an AfterGetRecords (see page 8506) event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.

5.The client dataset receives an AfterGetRecords (see page 8506) event, where it can respond to the custom information returned by the provider's AfterGetRecords (see page 8506) event handler.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
value of the first field for the last record to the server so
that the server can return the correct records. The complete
project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
var
  LastValue: OleVariant;
  CDSClone: TClientDataSet;
begin
  if ClientDataSet1.Active then
  begin
    CDSClone := TClientDataSet.Create(Form1);
    try
      CDSClone.CloneCursor(ClientDataSet1, True);
      { turn off FetchOnDemand so that the clone only fetches
        the last LOCAL record }
      CDSClone.FetchOnDemand := False;
      CDSClone.Last;
      LastValue := CDSClone.Fields[0].AsString;
      CDSClone.Close;
    finally
      CDSClone.Free;
    end;
  end else
    LastValue := NULL;
  OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
  ClientDataSet1.GetNextPacket;
end;
```

See Also

AS_GetRecords (see page 16455)

GetRecords (see page 18468)

BeforeGetRecords (see page 18480)

AfterGetRecords (see page 8506)

Communicating with the Client Dataset

45.1.8.6.23 TCustomClientDataSet.BeforeInsert Event

Occurs before an application enters insert mode.

Pascal

```
property BeforeInsert: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeInsert;
```

Description

Write a BeforeInsert event handler to take specific action before an application inserts or appends a new record. The Insert or Append method generates a BeforeInsert method before it sets the dataset into dsInsert state.

Delphi Examples:

```
{
This example updates a memo with a message when an
AfterInsert event or a BeforeInsert event occurs.
}
procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After inserting a record');
end;

procedure TForm1.ClientDataSet1BeforeInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before inserting a record');
end;
{
This example uses the BeforeInsert event to do data
validation. If the StrToInt function raises an exception,
the edit control’s contents are set to a valid value so
that the assignment to the INTEGER field in the dataset
will succeed.  Continuing after the exception causes a
'0' to be placed in the PORTA field.
DataSet design:
This example requires a ClientDataSet, a DBGrid, a
DataSource and a DBNavigator.  Name the ClientDataSet CDS.
Set the DataSet property in the DataSource to CDS and name
that DataSource DS.  Set the DataSource in the DBNavigator
to DS the DataSource in the DBGrid to DS.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
type to Integer.  The file used to load CDS must have an
attribute name of HOST with an attribute type of string and
an attribute name of PORTA with an attribute type of integer.
Add DB, DBCtrls, Grids, DBGrids and DBClient to the uses clause.
}
{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button3Click(Sender: TObject);
```

```
begin
  CDS.Insert;
  CDS.FieldByName('PORTA').AsInteger := StrToInt(Edit1.Text);
  CDS.Post;
end;

procedure TForm1.CDSBeforeInsert(DataSet: TDataSet);
begin
  try
  { Make sure edit field can be converted to integer --
   this will raise an exception if it can't }
    StrToInt(Edit1.Text);
  except
    Edit1.Text := '0';
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;
```

See Also

State (see page 7036)

AfterInsert (see page 7044)

Append (see page 6910)

Insert (see page 6959)

45.1.8.6.24 TCustomClientDataSet.BeforeOpen Event

Occurs before an application executes a request to open a dataset.

Pascal

```
property BeforeOpen: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeOpen;
```

Description

Write a BeforeOpen event handler to take specific action before an application opens a dataset for viewing or editing. BeforeOpen is triggered when an application sets the Active property to true for a dataset or an application calls Open.

See Also

Active (see page 7004)

AfterOpen (see page 7045)

Open (see page 6976)

Opening and Closing Datasets

45.1.8.6.25 TCustomClientDataSet.BeforePost Event

Occurs before an application posts changes for the active record to the database or change log.

Pascal

```
property BeforePost: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforePost;
```

Description

Write a BeforePost event handler to take specific action before an application posts dataset changes. BeforePost is triggered when an application calls the Post method. Post checks to make sure all required fields are present, then calls BeforePost before posting the record.

An application might use BeforePost to perform validity checks on data changes before committing them. If it encountered a validity problem, it could call Abort to cancel the Post operation (Delphi) or throw an exception (C++).

Delphi Examples:

```
{
This example checks for a valid entry in a TDBEdit control
and calls the Abort procedure if the control is empty; Abort
cancels the post before it happens.
}
procedure TForm1.ClientDataSet1BeforePost(DataSet: TDataSet);
begin
  if DBEdit1.Text = '' then
    Abort;
  DataSet.Fields[2].AsString := DBEdit1.Text;
end;
```

See Also

AfterPost (see page 7046)

Post (see page 6977)

Abort (see page 22786)

Working with Field Component Methods at Runtime

45.1.8.6.26 TCustomClientDataSet.BeforeRefresh Event

Occurs immediately before an application refreshes the data in the dataset.

Pascal

```
property BeforeRefresh: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeRefresh;
```

Description

Write a BeforeRefresh event handler to take specific action immediately before an application updates the records in the dataset. BeforeRefresh is generated by calls to the Refresh method.

See Also

AfterRefresh (see page 7047)

Refresh (see page 6991)

45.1.8.6.27 TCustomClientDataSet.BeforeRowRequest Event

Occurs before the client dataset fetches new information about the current record.

Pascal

```
property BeforeRowRequest: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeRowRequest;
```

Description

Write a BeforeRowRequest event handler to send custom information to the client dataset's provider. BeforeRowRequest is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches detail information about the current record or refreshes the current record, the following events occur:

1.The client dataset receives a BeforeRowRequest event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider on the application server receives a BeforeRowRequest event, where it can respond to or change that information before it looks up the requested information.

3.The provider creates a delta packet that contains the requested information.

4.The provider receives an AfterRowRequest (see page 8509) event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeRowRequest event handler.

5.The client dataset receives an AfterRowRequest (see page 8509) event, where it can respond to the custom information returned by the provider's AfterRowRequest (see page 8509) event handler.

See Also

AS_RowRequest (see page 16456)

RowRequest (see page 18472)

BeforeRowRequest (see page 18482)

AfterRowRequest (see page 8509)

FetchBlobs (see page 8414)

FetchParams (see page 8415)

RefreshRecord (see page 8442)

Communicating with the Client Dataset

45.1.8.6.28 TCustomClientDataSet.BeforeScroll Event

Occurs before an application scrolls from one record to another.

Pascal

```
property BeforeScroll: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeScroll;
```

Description

Write a BeforeScroll event handler to take specific action immediately before an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. BeforeScroll is called before all other events triggered by these methods and any other methods that switch from record to record in the dataset.

See Also

AfterScroll (see page 7047)

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Prior (see page 6979)

Navigating Datasets

45.1.8.6.29 TCustomClientDataSet.OnCalcFields Event

Occurs when an application recalculates calculated fields.

Pascal

```
property OnCalcFields: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent OnCalcFields;
```

Description

Write an OnCalcFields event handler to take specific action when an application recalculates calculated fields. A calculated field is one that derives its value from the values in one or more fields in the dataset, sometimes with additional processing.

OnCalcFields is triggered when:

A dataset is opened.

A dataset is put into dsEdit state.

A record is retrieved from a database.

When the AutoCalcFields property is true, OnCalcFields is also triggered when:

Focus moves from one visual control to another, or from one column to another is a data-aware grid control and modifications were made to the record.

Note: When the AutoCalcFields property is true, an OnCalcFields event handler should not modify the dataset (or a linked dataset if it is part of a master-detail relationship), because such modifications retrigger the OnCalcField event, leading to infinite recursion.

If an application permits users to change data, OnCalcFields is frequently triggered. To reduce the frequency with which OnCalcFields occurs, set AutoCalcFields to false.

Warning: When the dataset is the master table of a master-detail relationship, OnCalcFields occurs before detail sets have

been synchronized with the master table.

See Also

Active (see page 7004)

AutoCalcFields (see page 7006)

Open (see page 6976)

Post (see page 6977)

Calculating Fields

Using Internally Calculated Fields in Client Datasets

Defining a Calculated Field

45.1.8.6.30 TCustomClientDataSet.OnDeleteError Event

Occurs when an application attempts to delete a record and an exception is raised.

Pascal

```
property OnDeleteError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnDeleteError;
```

Description

Write an OnDeleteError event handler to handle exceptions that occur when an attempt to delete a record fails.

See Also

OnEditError (see page 7054)

OnPostError (see page 7056)

Delete (see page 6924)

45.1.8.6.31 TCustomClientDataSet.OnEditError Event

Occurs when an application attempts to modify or insert a record and an exception is raised.

Pascal

```
property OnEditError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnEditError;
```

Description

Write an OnEditError event handler to handle exceptions that occur when an attempt to edit a record fails.

See Also

OnDeleteError (see page 7054)

OnPostError (see page 7056)

Edit (see page 6932)

Insert (see page 6959)

45.1.8.6.32 TCustomClientDataSet.OnFilterRecord Event

Occurs each time a different record in the dataset becomes the active record and filtering is enabled.

Pascal

```
property OnFilterRecord: TFilterRecordEvent;
```

C++

```
__property TFilterRecordEvent OnFilterRecord;
```

Description

Write an OnFilterRecord event handler to specify for each record in a dataset whether it should be visible to the application. To indicate that a record passes the filter condition, the OnFilterRecord event handler must set the Accept parameter to true. To exclude a record, set the Accept parameter to false. Most TDataSet descendants (such as ADO datasets or BDE-based datasets) initialize Accept to true before calling the OnFilterRecord event handler.

Warning: Do not supply an OnFilterRecord event handler to a unidirectional dataset. Unidirectional datasets do not support filters, and assigning an OnFilterRecord event handler causes the dataset to raise an exception.

Filtering is enabled if the Filtered property is true. When an application is processing a filter, the State property for the dataset is dsFilter.

Use an OnFilterRecord event handler to filter records using a criterion that can't be implemented using the Filter property. For example, using the Filter property, field comparisons are not supported against local tables (Paradox, dBASE, Access, FoxPro), but an OnFilterRecord event handler can implement any criterion at all.

Tip: Be sure that any interactions between the Filter property and the OnFilterRecord event handler do not result in an empty filter set when they are used simultaneously in an application.

See Also

Filter (see page 7026)

Filtered (see page 7027)

FilterOptions (see page 7027)

State (see page 7036)

Creating Filters

Navigating Records in a Filtered Dataset

Displaying and Editing a Subset of Data Using Filters

Writing an OnFilterRecord Event Handler

45.1.8.6.33 TCustomClientDataSet.OnNewRecord Event

Occurs when an application inserts or appends a new dataset record.

Pascal

```
property OnNewRecord: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent OnNewRecord;
```

Description

Write an OnNewRecord event handler to take specific actions as an application inserts or appends a new record. OnNewRecord

is called as part of the insert or append process. An application might use the OnNewRecord event to set initial values for a record or as a way of implementing cascading insertions in related datasets.

See Also

AfterInsert (see page 7044)

BeforeInsert (see page 7050)

45.1.8.6.34 TCustomClientDataSet.OnPostError Event

Occurs when an application attempts to modify or insert a record and an exception is raised.

Pascal

```
property OnPostError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnPostError;
```

Description

Write an OnPostError event handler to handle exceptions that occur when an attempt to post a record fails.

See Also

OnDeleteError (see page 7054)

OnEditError (see page 7054)

Post (see page 6977)

45.1.8.6.35 TCustomClientDataSet.OnReconcileError Event

Occurs when a client dataset needs to reconcile an update to a record that could not be applied.

Pascal

```
property OnReconcileError: TReconcileErrorEvent;
```

C++

```
__property TReconcileErrorEvent OnReconcileError;
```

Description

Write an OnReconcileError event handler to respond to error conditions that arise when the Reconcile (see page 8441) method cycles through records that could not be applied to the database. This event handles only those update errors that were not handled in an OnUpdateError event handler.

The OnReconcileError event handler is called once for each record that could not be applied to the dataset or handled by the provider.

Warning: A provider can't detect when an update conflicts with another user's changes to a memo field. This situation does not generate an OnReconcileError event.

See Also

Reconcile (see page 8441)

OldValue (see page 7175)

NewValue (see page 7174)

CurValue (see page 7155)

ApplyUpdates (see page 8394)

OnUpdateError (see page 18444)

OnUpdateError

Applying Updates

Setting Up a Simple Dataset

Reconciling Update Errors

Using the BDE to Cache Updates

Updating Data in dbExpress Applications

45.1.9 TCustomRemoteServer Class

TCustomRemoteServer is the base class for all components that connect to an application server in a multi-tiered database application.

Class Hierarchy

Pascal

```
TCustomRemoteServer = class(TCustomConnection);
```

C++

```
class TCustomRemoteServer : public TCustomConnection;
```

File

DBClient

Description

Do not create instances of TCustomRemoteServer. TCustomRemoteServer introduces the properties and methods needed to connect to a remote application server and make its IAppServer interface available to one or more client datasets. Descendant classes, which are often called DataSnap connection components, override many of TCustomRemoteServer's methods to create or close connections to the application server and to manage the interface communication with the application server.

To enable a client application to connect to an application server, use a descendant of TCustomRemoteServer such as TSOAPConnection, TSocketConnection, TWebConnection, or TDCOMConnection.

See Also

TClientDataSet (see page 8289)

TBaseProvider (see page 18423)

TSocketConnection (see page 18928)

TDCOMConnection (see page 16008)

IAppServer (see page 16450)

TWebconnection

TSOAPConnection

Using a Multi-Tiered Architecture

45.1.9.1 TCustomRemoteServer Members

The following tables list the members exposed by TCustomRemoteServer.

Protected Events

	Name	Description
	OnGetUsername (see page 8533)	This is OnGetUsername, a member of class TCustomRemoteServer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoConnect (see page 6864)	Provides the interface for a method that opens a connection.
V	DoDisconnect (see page 6864)	Provides the interface for a method that terminates the connection.
V	GetAppServer (see page 8530)	Returns the value of the AppServer (see page 8532) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetConnected (see page 6865)	Returns the value of the Connected (see page 6868) property.
V	GetDataSet (see page 6865)	Returns a dataset, given its index in the DataSets (see page 6870) property array.
V	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 6866)	Initializes the connection component after all objects in the form or data module have been loaded.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	RegisterClient (see page 6866)	Registers an object that uses the connection component.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.

	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppServer (see page 8531)	Sets the AppServer (see page 8532) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnected (see page 6867)	Changes the value of the Connected (see page 6868) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	UnRegisterClient (see page 6868)	Removes a registered client so that it is no longer associated with the connection component.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.

Public Events

	Name	Description
	AfterConnect (see page 6874)	Occurs after a connection is established.
	AfterDisconnect (see page 6875)	Occurs after the connection closes.
	BeforeConnect (see page 6875)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 6875)	Occurs immediately before the connection closes.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 6863)	Closes the connection.
	Create (see page 8529)	Creates an instance of a remote server component.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.

V	Destroy (see page 6863)	Destroys the instance of a connection component.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetProviderNames (see page 8530)	Executes a callback for all provider objects on the application server.
V	GetServer (see page 8530)	Returns the IAppServer interface of the application server.
V	GetServerList (see page 8531)	Returns a list of application servers (remote data modules) to which the component can connect.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 6866)	Opens the connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AppServer (see page 8532)	Contains the interface for communicating with the application server.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Connected (see page 6868)	Determines whether a connection has been established to the remote source of data.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 6870)	Lists the datasets currently using this connection component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	LoginPrompt (see page 6872)	Specifies whether a login dialog appears immediately before opening a new connection.
	OnLogin (see page 6873)	Occurs when a communication channel opens to the server.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

45.1.9.2 TCustomRemoteServer.Create Constructor

Creates an instance of a remote server component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomRemoteServer(TComponent * AOwner);
```

Description

Create instantiates an instance of a remote server component and initializes its properties.

AOwner is the component that is responsible for freeing the TCustomRemoteServer (see page 8525) instance. It becomes the value of the Owner property.

See Also

Create (see page 6862)

Owner (see page 2807)

45.1.9.3 TCustomRemoteServer Methods

The methods of the TCustomRemoteServer class are listed here.

Protected Methods

	Name	Description
V	GetAppServer (see page 8530)	Returns the value of the AppServer (see page 8532) property.
V	SetAppServer (see page 8531)	Sets the AppServer (see page 8532) property.

Public Methods

	Name	Description
	GetProviderNames (see page 8530)	Executes a callback for all provider objects on the application server.
	GetServer (see page 8530)	Returns the IAppServer interface of the application server.
	GetServerList (see page 8531)	Returns a list of application servers (remote data modules) to which the component can connect.

45.1.9.3.1 TCustomRemoteServer.GetAppServer Method

Returns the value of the AppServer (see page 8532) property.

Pascal

```
function GetAppServer: Variant; virtual;
```

C++

```
virtual __fastcall Variant GetAppServer();
```

Description

GetAppServer is the protected implementation of the AppServer (see page 8532) property. Override GetAppServer to change the way this property is stored or managed.

See Also

AppServer (see page 8532)

GetServer (see page 8530)

45.1.9.3.2 TCustomRemoteServer.GetProviderNames Method

Executes a callback for all provider objects on the application server.

Pascal

```
procedure GetProviderNames(Proc: TGetStrProc); virtual;
```

C++

```
virtual __fastcall GetProviderNames(TGetStrProc Proc);
```

Description

Call GetProviderNames to execute the procedure specified by the Proc parameter for every provider exposed by the application server. For every provider listed by the remote data module's AS_GetProviderNames method, Proc is called with the name of the provider.

See Also

AS_GetProviderNames (see page 16454)

AppServer (see page 8532)

ProviderName (see page 8492)

45.1.9.3.3 TCustomRemoteServer.GetServer Method

Returns the IAppServer interface of the application server.

Pascal

```
function GetServer: IAppServer; virtual;
```

C++

```
virtual __fastcall IAppServer GetServer();
```

Description

Descendant classes override this function to return the interface of the application server after establishing a connection. As implemented in TCustomRemoteServer (see page 8525), GetServer always returns nil (Delphi) or NULL (C++).

See Also

Connected (see page 6868)

AppServer (see page 8532)

AppServer (see page 8462)

45.1.9.3.4 TCustomRemoteServer.GetServerList Method

Returns a list of application servers (remote data modules) to which the component can connect.

Pascal

```
function GetServerList: OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant GetServerList();
```

Description

Connection components use GetServerList internally to fill the drop down list in the Object Inspector for the ServerName property.

As implemented in TCustomRemoteServer (see page 8525), GetServerList always returns NULL. Descendant classes override this method to return a list of application servers that are on the local machine and registered in the system registry (TDispatchConnection), or fetched from the server machine (TStreamedConnection).

See Also

ServerName (see page 16030)

45.1.9.3.5 TCustomRemoteServer.SetAppServer Method

Sets the AppServer (see page 8532) property.

Pascal

```
procedure SetAppServer(Value: Variant); virtual;
```

C++

```
virtual __fastcall SetAppServer(Variant Value);
```

Description

SetAppServer allows descendants of TCustomRemoteServer (see page 8525) to set the read-only AppServer (see page 8532) property when they establish a connection to the application server.

See Also

AppServer (see page 8532)

Connected (see page 6868)

45.1.9.4 TCustomRemoteServer Properties

The properties of the TCustomRemoteServer class are listed here.

Public Properties

	Name	Description
R	AppServer (see page 8532)	Contains the interface for communicating with the application server.

45.1.9.4.1 TCustomRemoteServer.AppServer Property

Contains the interface for communicating with the application server.

Pascal

```
property AppServer: Variant;
```

C++

```
__property Variant AppServer;
```

Description

AppServer provides read-only access to the interface returned by the application server's remote data module (a descendant of IAppServer). If the remote data module is not a SOAP data module, client applications can communicate directly with the application server through this interface.

In Delphi, when calling methods directly using AppServer, the client application uses late binding. For details on how to improve on this, see Calling server interfaces .

Note: In C++, to call this interface, you must obtain a dispatch interface from this Variant. The dispatch interface has the same name as the application server's interface, but with the string "Disp" appended. Thus, if the remote data module in the application server is called MyAppServer, you can use AppServer to call its interface as follows:

Note: IDispatch* disp = (IDispatch*)(MyConnection->AppServer);

Note: IMyAppServerDisp TempInterface((IMyAppServer*)disp);

Note: TempInterface.SpecialMethod(x,y);

Note: To use the IAppServer methods of the application server's interface, obtain an interface using the GetServer (see page 8530) method instead.

See Also

TClientDataSet (see page 8289)

GetServer (see page 8530)

TRemoteDataModule (see page 6585)

IAppServer (see page 16450)

45.1.9.5 TCustomRemoteServer Events

The events of the TCustomRemoteServer class are listed here.

Protected Events

	Name	Description
	OnGetUsername (see page 8533)	This is OnGetUsername, a member of class TCustomRemoteServer.

45.1.9.5.1 TCustomRemoteServer.OnGetUsername Event

Pascal

```
property OnGetUsername: TGetUsernameEvent;
```

C++

```
__property TGetUsernameEvent OnGetUsername;
```

Description

This is OnGetUsername, a member of class TCustomRemoteServer.

45.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	PackageParams (see page 8534)	Converts a TParams object into an OleVariant that can be used in multi-tiered applications.
	UnpackParams (see page 8534)	Converts an OleVariant that represents a set of parameters into a TParams object.

45.2.1 DBClient.PackageParams Function

Converts a TParams object into an OleVariant that can be used in multi-tiered applications.

Pascal

```
function PackageParams(Params: TParams; Types: TParamTypes = AllParamTypes): OleVariant;
```

C++

```
OleVariant PackageParams(TParams Params, TParamTypes Types = AllParamTypes);
```

File

DBClient

Description

IAppServer methods that pass parameter values to or from the application server use an OleVariant to represent those parameters. Client datasets use PackageParams to convert their Params property into the OleVariant format. Use PackageParams to encode parameter values when using the IAppServer interface directly rather than using the methods of TClientDataSet (see page 8289).

The Params parameter is a TParams object (usually the Params property of the source component) whose parameter values are to be encoded.

The Types (see page 23293) parameter indicates which parameters from the Params parameter are to be included. This allows applications to include, for example, only input parameters if no others are needed.

PackageParams returns an OleVariant that encodes the parameter values as a Variant array.

See Also

IAppServer

TCustomClientDataSet (see page 8373)

UnpackParams (see page 8534)

45.2.2 DBClient.UnpackParams Function

Converts an OleVariant that represents a set of parameters into a TParams object.

Pascal

```
procedure UnpackParams(const Source: OleVariant; Dest: TParams);
```

C++

```
UnpackParams(const OleVariant Source, TParams Dest);
```

File

DBClient

Description

Client datasets use UnpackParams update their Params property with parameter values received over an IAppServer interface. Use UnpackParams to decode parameter values when using the IAppServer interface directly rather than using the methods of TClientDataSet (see page 8289).

The Source parameter is an OleVariant that represents a set of parameter values. Parameter values can be encoded into this format using the PackageParams (see page 8534) function.

The Dest parameter is a TParams object (usually the Params property of a destination component) that receives the parameter values.

See Also

IAppServer

TCustomClientDataSet (see page 8373)

PackageParams (see page 8534)

45.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TDataPacketFormat (see page 8536)	TDataPacketFormat indicates how information is encoded in a data packet or delta packet.
	TDataSetOption (see page 8537)	This is record DBClient.TDataSetOption.
	TFetchOption (see page 8537)	TFetchOption and TFetchOptions (see page 8543) indicate the type of information about a record that a client dataset can request.
	TReconcileAction (see page 8539)	TReconcileAction indicates how a client dataset should handle a record whose update generated an error.

Records

	Name	Description
	TKeyBuffer (see page 8538)	This is record DBClient.TKeyBuffer.
	TRecInfo (see page 8538)	This is record DBClient.TRecInfo.
	TReconcileInfo (see page 8539)	This is record DBClient.TReconcileInfo.

45.3.1 DBClient.TDataPacketFormat Enumeration

TDataPacketFormat indicates how information is encoded in a data packet or delta packet.

Pascal

```
TDataPacketFormat = (
  dfBinary,
  dfXML,
  dfXMLUTF8
);
```

C++

```
enum TDataPacketFormat {
  dfBinary,
  dfXML,
  dfXMLUTF8
};
```

File

DBClient

Description

TDataPacketFormat indicates how a provider encodes database information into data packets. The following table lists the possible values:

Value	Meaning
dfBinary	Information is encoded in binary format.
dfXML	Information is encoded in XML, with extended characters encoded using an escape sequence.
dfXMLUTF8	Information is encoded in XML, with extended characters represented using UTF8.

45.3.2 DBClient.TDataSetOption Enumeration

Pascal

```
TDataSetOption = (
  doDisableInserts,
  doDisableDeletes,
  doDisableEdits,
  doNoResetCall
);
```

C++

```
enum TDataSetOption {
  doDisableInserts,
  doDisableDeletes,
  doDisableEdits,
  doNoResetCall
};
```

File

DBClient

Description

This is record DBClient.TDataSetOption.

45.3.3 DBClient.TFetchOption Enumeration

TFetchOption and TFetchOptions (see page 8543) indicate the type of information about a record that a client dataset can request.

Pascal

```
TFetchOption = (
  foRecord,
  foBlobs,
  foDetails
);
```

C++

```
enum TFetchOption {
  foRecord,
  foBlobs,
  foDetails
};
```

File

DBClient

Description

Providers encode the flags received from a client dataset that indicate what type of row information they require into a value of type TFetchOptions (see page 8543). This value is used internally when assembling the requested information. TFetchOptions (see page 8543) is a set that includes zero or more of the following values:

Value	Meaning
foRecord	The values of fields on the current record. (Used to refresh records).
foBlobs	The values of BLOB fields on the current record. (Used to implement delayed fetching of BLOB data).
foDetails	Any nested dataset fields for the current record. (Used to implement delayed fetching of detail datasets.

45.3.4 DBClient.TKeyBuffer Record

Pascal

```
TKeyBuffer = record
  Modified: Boolean;
  Exclusive: Boolean;
  FieldCount: Integer;
  Data: record;
end;
```

C++

```
struct TKeyBuffer {
  Boolean Modified;
  Boolean Exclusive;
  int FieldCount;
  record Data;
};
```

File

DBClient

Description

This is record DBClient.TKeyBuffer.

45.3.5 DBClient.TRecInfo Record

Pascal

```
TRecInfo = packed record
  RecordNumber: Longint;
  BookmarkFlag: TBookmarkFlag;
  Attribute: DSAttr;
end;
```

C++

```
struct TRecInfo {
  Longint RecordNumber;
  TBookmarkFlag BookmarkFlag;
  DSAttr Attribute;
};
```

File

DBClient

Description

This is record DBClient.TRecInfo.

45.3.6 DBClient.TReconcileAction Enumeration

TReconcileAction indicates how a client dataset should handle a record whose update generated an error.

Pascal

```
TReconcileAction = (
  raSkip,
  raAbort,
  raMerge,
  raCorrect,
  raCancel,
  raRefresh
);
```

C++

```
enum TReconcileAction {
  raSkip,
  raAbort,
  raMerge,
  raCorrect,
  raCancel,
  raRefresh
};
```

File

DBClient

Description

TReconcileAction is the type of the return parameter in a client dataset's OnReconcileError event handler. It indicates the action to take when the OnReconcileError handler exits. The following table lists the possible values and what they indicate:

Value	Meaning
raSkip	Skip updating the record that raised the error condition, and leave the unapplied changes in the change log.
raAbort	Abort the entire reconcile operation.
raMerge	Merge the updated record with the record on the server.
raCorrect	Replace the current updated record with the value of the record in the event handler.
raCancel	Back out all changes for this record, reverting to the original field values.
raRefresh	Back out all changes for this record, replacing it with the current values from the server.

See Also

TCustomClientDataSet (see page 8373)

TResolverResponse (see page 18620)

45.3.7 DBClient.TReconcileInfo Record

Pascal

```
TReconcileInfo = record
  DataSet: TDataSet;
```

```
    UpdateKind: TUpdateKind;
    ReconcileError: EReconcileError;
    ActionRef: ^TReconcileAction;
  end;
```

C++

```
  struct TReconcileInfo {
    TDataSet DataSet;
    TUpdateKind UpdateKind;
    EReconcileError ReconcileError;
    ^TReconcileAction ActionRef;
  };
```

File

DBClient

Description

This is record DBClient.TReconcileInfo.

45.4 Types

The following table lists types in this documentation.

Types

Name	Description
PKeyBuffer (see page 8541)	This is type DBClient.PKeyBuffer.
PRecInfo (see page 8541)	This is type DBClient.PRecInfo.
TAggUpdateEvent (see page 8542)	This is type DBClient.TAggUpdateEvent.
TCDSKeyBuffer (see page 8542)	This is type DBClient.TCDSKeyBuffer.
TCDSRecInfo (see page 8542)	This is type DBClient.TCDSRecInfo.
TDataSetOptions (see page 8543)	This is type DBClient.TDataSetOptions.
TFetchOptions (see page 8543)	This is type DBClient.TFetchOptions.
TFieldDescList (see page 8543)	This is type DBClient.TFieldDescList.
TGetUsernameEvent (see page 8544)	This is type DBClient.TGetUsernameEvent.
TReconcileErrorEvent (see page 8544)	This is type DBClient.TReconcileErrorEvent.
TRemoteEvent (see page 8544)	This is type DBClient.TRemoteEvent.

45.4.1 DBClient.PKeyBuffer Type

Pascal

```
PKeyBuffer = ^TKeyBuffer;
```

C++

```
^TKeyBuffer PKeyBuffer;
```

File

DBClient

Description

This is type DBClient.PKeyBuffer.

45.4.2 DBClient.PRecInfo Type

Pascal

```
PRecInfo = ^TRecInfo;
```

C++

```
^TRecInfo PRecInfo;
```

File

DBClient

Description

This is type DBClient.PRecInfo.

45.4.3 DBClient.TAggUpdateEvent Type

Pascal

```
TAggUpdateEvent = procedure (Agg: TAggregate) of object;
```

C++

```
(Agg: TAggregate) ( TAggUpdateEvent)();
```

File

DBClient

Description

This is type DBClient.TAggUpdateEvent.

45.4.4 DBClient.TCDSKeyBuffer Type

Pascal

```
TCDSKeyBuffer = TKeyBuffer;
```

C++

```
TKeyBuffer TCDSKeyBuffer;
```

File

DBClient

Description

This is type DBClient.TCDSKeyBuffer.

45.4.5 DBClient.TCDSRecInfo Type

Pascal

```
TCDSRecInfo = TRecInfo;
```

C++

```
TRecInfo TCDSRecInfo;
```

File

DBClient

Description

This is type DBClient.TCDSRecInfo.

45.4.6 DBClient.TDataSetOptions Type

Pascal

```
TDataSetOptions = set of TDataSetOption;
```

C++

```
TDataSetOption TDataSetOptions;
```

File

DBClient

Description

This is type DBClient.TDataSetOptions.

45.4.7 DBClient.TFetchOptions Type

Pascal

```
TFetchOptions = set of TFetchOption;
```

C++

```
TFetchOption TFetchOptions;
```

File

DBClient

Description

This is type DBClient.TFetchOptions.

45.4.8 DBClient.TFieldDescList Type

Pascal

```
TFieldDescList = array of DSFLDDesc;
```

C++

```
array of DSFLDDesc TFieldDescList;
```

File

DBClient

Description

This is type DBClient.TFieldDescList.

45.4.9 DBClient.TGetUsernameEvent Type

Pascal

```
TGetUsernameEvent = procedure (Sender: TObject; var Username: string) of object;
```

C++

```
(Sender: TObject; var Username: string) ( TGetUsernameEvent)();
```

File

DBClient

Description

This is type DBClient.TGetUsernameEvent.

45.4.10 DBClient.TReconcileErrorEvent Type

Pascal

```
TReconcileErrorEvent = procedure (DataSet: TCustomClientDataSet; E: EReconcileError;
UpdateKind: TUpdateKind; var Action: TReconcileAction) of object;
```

C++

```
(DataSet: TCustomClientDataSet; E: EReconcileError; UpdateKind: TUpdateKind; var Action:
TReconcileAction) ( TReconcileErrorEvent)();
```

File

DBClient

Description

This is type DBClient.TReconcileErrorEvent.

45.4.11 DBClient.TRemoteEvent Type

Pascal

```
TRemoteEvent = procedure (Sender: TObject; var OwnerData: OleVariant) of object;
```

C++

```
(Sender: TObject; var OwnerData: OleVariant) ( TRemoteEvent)();
```

File

DBClient

Description

This is type DBClient.TRemoteEvent.

45.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
AllParamTypes (see page 8545)	This is constant DBClient.AllParamTypes.
AllRecords (see page 8545)	This is constant DBClient.AllRecords.

45.5.1 DBClient.AllParamTypes Constant

Pascal

```
AllParamTypes = [ptUnknown, ptInput, ptOutput, ptInputOutput, ptResult];
```

C++

```
AllParamTypes = [ptUnknown, ptInput, ptOutput, ptInputOutput, ptResult];
```

File

DBClient

Description

This is constant DBClient.AllParamTypes.

45.5.2 DBClient.AllRecords Constant

Pascal

```
AllRecords = -1;
```

C++

```
AllRecords = -1;
```

File

DBClient

Description

This is constant DBClient.AllRecords.

46 DBClientActns Namespace

This is namespace DBClientActns.

Topics

Name	Description
Classes (see page 8548)	The following table lists classes in this documentation.

Classes

	Name	Description
	TClientDataSetApply (see page 8548)	TClientDataSetApply causes a target client dataset to apply the updates in its change log.
	TClientDataSetRevert (see page 8556)	TClientDataSetRevert causes a target client dataset to back out all changes to the current record.
	TClientDataSetUndo (see page 8563)	TClientDataSetUndo causes a target client dataset to back out the last edit to the current record.
	TReconcileActionLink (see page 8570)	TReconcileActionLink helps a TClientDataSetApply (see page 8548) action display a dialog when the client dataset encounters reconcile errors.

46.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TClientDataSetApply (see page 8548)	TClientDataSetApply causes a target client dataset to apply the updates in its change log.
	TClientDataSetRevert (see page 8556)	TClientDataSetRevert causes a target client dataset to back out all changes to the current record.
	TClientDataSetUndo (see page 8563)	TClientDataSetUndo causes a target client dataset to back out the last edit to the current record.
	TReconcileActionLink (see page 8570)	TReconcileActionLink helps a TClientDataSetApply (see page 8548) action display a dialog when the client dataset encounters reconcile errors.

46.1.1 TClientDataSetApply Class

TClientDataSetApply causes a target client dataset to apply the updates in its change log.

Class Hierarchy

Pascal

```
TClientDataSetApply = class(TDataSetAction);
```

C++

```
class TClientDataSetApply : public TDataSetAction;
```

File

DBClientActns

Description

Add TClientDataSetApply to an action list or action manager to allow users to apply the updates that a client dataset caches. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the target dataset to apply the updates in its change log. If you set the DisplayErrorDlg (see page 8556) property to true, then if the applying of updates results in errors, TClientDataSetApply displays an error dialog so that the user can indicate how to respond to those errors.

TClientDataSetApply is designed to work with a target that is a TDataSource component whose dataset is a TClientDataSet instance. You can explicitly provide this target by setting the DataSource (see page 8555) property, or you can allow the action to infer the data source from an active data-aware control. TClientDataSetApply automatically disables itself if it does not have a target data source attached to an active client dataset.

TClientDatasetApply action automatically generates a reconcile error dialog, so you do not have to write any code to invoke the dialog. Typically, you would assign the OnReconcileError (TClientDataset) event and write the code to invoke the dialog.

See Also

TClientDataSetRevert (see page 8556)

TClientDataSetUndo (see page 8563)

TCustomClientDataSet (see page 8373)

TDataSource (see page 7070)

TReconcileActionLink (see page 8570)

TClientDataSet (see page 8289)

46.1.1.1 TClientDataSetApply Members

The following tables list the members exposed by TClientDataSetApply.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoReconcileError (see page 8554)	This is DoReconcileError, a member of class TClientDataSetApply.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDataSet (see page 7655)	This is GetDataSet, a member of class TDataSetAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.

	Notification (see page 7656)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 8553)	Creates an instance of TClientDataSetApply (see page 8548).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 8553)	Frees the memory associated with the TClientDataSetApply (see page 8548) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 8554)	Applies any updates the target client dataset has cached in its change log.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 7656)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)

V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 8555)	Enables or disables the action, depending on the state of the target dataset.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.

	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	DataSource (see page 8555)	Provides access to the target of the action.
	DisplayErrorDlg (see page 8556)	Indicates whether the action should display a reconcile dialog when applying updates results in errors.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	MaxErrors (see page 8556)	Indicates the maximum number of errors that can be allowed when applying updates.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

46.1.1.2 TClientDataSetApply.Create Constructor

Creates an instance of TClientDataSetApply (see page 8548).

Pascal

```
constructor Create(Owner: TComponent); override;
```

C++

```
virtual __fastcall TClientDataSetApply(TComponent * Owner);
```

Description

Most applications do not need to call the TClientDataSetApply (see page 8548) constructor because actions are added at design time to an action manager or action list.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TClientDataSetApply (see page 8548) instance.

See Also

Owner (see page 2807)

46.1.1.3 TClientDataSetApply.Destroy Destructor

Frees the memory associated with the TClientDataSetApply (see page 8548) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TClientDataSetApply();
```

Description

Do not explicitly destroy actions. Action objects are created and destroyed by the action manager or action list that contains them.

See Also

delete

46.1.1.4 TClientDataSetApply Methods

The methods of the TClientDataSetApply class are listed here.

Protected Methods

	Name	Description
	DoReconcileError (see page 8554)	This is DoReconcileError, a member of class TClientDataSetApply.

Public Methods

	Name	Description
	ExecuteTarget (see page 8554)	Applies any updates the target client dataset has cached in its change log.
	UpdateTarget (see page 8555)	Enables or disables the action, depending on the state of the target dataset.

46.1.1.4.1 TClientDataSetApply.DoReconcileError Method

Pascal

```
procedure DoReconcileError(ReconcileInfo: TReconcileInfo);
```

C++

```
__fastcall DoReconcileError(TReconcileInfo ReconcileInfo);
```

Description

This is DoReconcileError, a member of class TClientDataSetApply.

46.1.1.4.2 TClientDataSetApply.ExecuteTarget Method

Applies any updates the target client dataset has cached in its change log.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

You can call ExecuteTarget directly to execute this action, or it can be called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

ExecuteTarget checks that Target is a data source component whose DataSet property references a TClientDataSet instance. If the DataSource (see page 8555) property is set, Target must be the same data source.

If Target is a valid data source, ExecuteTarget calls the associated client dataset's ApplyUpdates method, passing in the value of the MaxErrors (see page 8556) property. If the client dataset encounters errors applying its updates, and if the DisplayErrorDlg (see page 8556) property is true, then TClientDataSetApply (see page 8548) responds to those errors by displaying a dialog where the user can indicate how to handle them.

See Also

ApplyUpdates (see page 8394)

DataSource (see page 7657)

DataSet (see page 7078)

MaxErrors (see page 8556)

DisplayErrorDlg (see page 8556)

46.1.1.4.3 TClientDataSetApply.UpdateTarget Method

Enables or disables the action, depending on the state of the target dataset.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TClientDataSetApply (see page 8548), UpdateTarget checks whether the current target object (specified by the Target parameter) is a data source whose dataset is an active TClientDataSet instance that has changes in its change log. If so, it sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

46.1.1.5 TClientDataSetApply Properties

The properties of the TClientDataSetApply class are listed here.

Published Properties

	Name	Description
	DataSource (see page 8555)	Provides access to the target of the action.
	DisplayErrorDlg (see page 8556)	Indicates whether the action should display a reconcile dialog when applying updates results in errors.
	MaxErrors (see page 8556)	Indicates the maximum number of errors that can be allowed when applying updates.

46.1.1.5.1 TClientDataSetApply.DataSource Property

Provides access to the target of the action.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource is the data source that represents the dataset on which the action performs its function. You can set this property to indicate the dataset that is the target of the action.

If you do not explicitly set the DataSource property, it is obtained from an active data-aware control when the action fires.

See Also

DataSet (see page 7078)

46.1.1.5.2 TClientDataSetApply.DisplayErrorDlg Property

Indicates whether the action should display a reconcile dialog when applying updates results in errors.

Pascal

```
property DisplayErrorDlg: Boolean;
```

C++

```
__property Boolean DisplayErrorDlg;
```

Description

Set DisplayErrorDlg to indicate whether you want the action to respond when the target client dataset encounters an OnReconcileError event.

When DisplayErrorDlg is true, TClientDataSetApply (see page 8548) is informed when the target client dataset encounters update errors and responds by calling the HandleReconcileError function. The resulting choice of reconcile action is used to initialize the Action parameter of the client dataset's OnReconcileError event handler if it exists. If the client dataset does not have an OnReconcileError event handler, the chosen reconcile action determines the client dataset's response to the update error.

See Also

HandleReconcileError

OnReconcileError (see page 8524)

46.1.1.5.3 TClientDataSetApply.MaxErrors Property

Indicates the maximum number of errors that can be allowed when applying updates.

Pascal

```
property MaxErrors: Integer;
```

C++

```
__property int MaxErrors;
```

Description

MaxErrors indicates the maximum number of errors that are allowed before prematurely stopping an update operation. Set MaxErrors to –1 to indicate that there is no limit to the number of errors.

TClientDataSetApply (see page 8548) passes the value of MaxErrors as the value of the MaxErrors parameter when it calls the ApplyUpdates method of the target client dataset.

See Also

ApplyUpdates (see page 8394)

46.1.2 TClientDataSetRevert Class

TClientDataSetRevert causes a target client dataset to back out all changes to the current record.

Class Hierarchy

Pascal

```
TClientDataSetRevert = class(TDataSetAction);
```

C++

```
class TClientDataSetRevert : public TDataSetAction;
```

File

DBClientActns

Description

Add TClientDataSetRevert to an action list or action manager to allow users to undo changes to the current record of a client dataset. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the target dataset to back out all edits to the current record.

TClientDataSetRevert is designed to work with a target that is a TDataSource component whose dataset is a TClientDataSet instance. You can explicitly provide this target by setting the DataSource (see page 8563) property, or you can allow the action to infer the data source from an active data-aware control. TClientDataSetRevert automatically disables itself if it does not have a target data source attached to an active client dataset.

See Also

TClientDataSetApply (see page 8548)

TClientDataSetUndo (see page 8563)

TCustomClientDataSet (see page 8373)

TDataSource (see page 7070)

TClientDataSet (see page 8289)

46.1.2.1 TClientDataSetRevert Members

The following tables list the members exposed by TClientDataSetRevert.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.

	AssignTo (see page 303)	Copies the properties of this action to a destination object.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDataSet (see page 7655)	This is GetDataSet, a member of class TDataSetAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7656)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 8562)	Backs out all changes to the current record of the target client dataset.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 7656)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 8562)	Enables or disables the action, depending on the state of the target dataset.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	DataSource (see page 8563)	Provides access to the target of the action.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

46.1.2.2 TClientDataSetRevert Methods

The methods of the TClientDataSetRevert class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 8562)	Backs out all changes to the current record of the target client dataset.
	UpdateTarget (see page 8562)	Enables or disables the action, depending on the state of the target dataset.

46.1.2.2.1 TClientDataSetRevert.ExecuteTarget Method

Backs out all changes to the current record of the target client dataset.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

You can call ExecuteTarget directly to execute this action, or it can be called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

ExecuteTarget checks that Target is a data source component whose DataSet property references a TClientDataSet instance. If the DataSource (see page 8563) property is set, Target must be the same data source.

If Target is a valid data source, ExecuteTarget calls the associated client dataset's RevertRecord method. This returns the record to the values it held when first fetched from the source dataset. Note that unlike the TClientDataSetUndo (see page 8563) action, this backs out all edits to the record, not just those made during the last edit operation.

See Also

RevertRecord (see page 8443)

DataSource (see page 7657)

DataSet (see page 7078)

46.1.2.2.2 TClientDataSetRevert.UpdateTarget Method

Enables or disables the action, depending on the state of the target dataset.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TClientDataSetRevert (see page 8556), UpdateTarget checks whether the current target object (specified by the Target parameter) is a data source whose dataset is an active TClientDataSet instance whose current record has been edited. If so, it sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

46.1.2.3 TClientDataSetRevert Properties

The properties of the TClientDataSetRevert class are listed here.

Published Properties

	Name	Description
	DataSource (see page 8563)	Provides access to the target of the action.

46.1.2.3.1 TClientDataSetRevert.DataSource Property

Provides access to the target of the action.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource is the data source that represents the dataset on which the action performs its function. You can set this property to indicate the dataset that is the target of the action.

If you do not explicitly set the DataSource property, it is obtained from an active data-aware control when the action fires.

See Also

DataSet (see page 7078)

46.1.3 TClientDataSetUndo Class

TClientDataSetUndo causes a target client dataset to back out the last edit to the current record.

Class Hierarchy

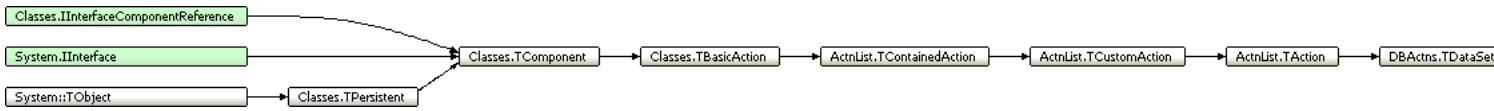

Pascal

```
TClientDataSetUndo = class(TDataSetAction);
```

C++

```
class TClientDataSetUndo : public TDataSetAction;
```

File

DBClientActns

Description

Add TClientDataSetUndo to an action list or action manager to allow users to back out the last edit to the current record of a client dataset. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the target dataset to undo the last edit.

TClientDataSetUndo is designed to work with a target that is a TDataSource component whose dataset is a TClientDataSet instance. You can explicitly provide this target by setting the DataSource (see page 8569) property, or you can allow the action to infer the data source from an active data-aware control. TClientDataSetUndo automatically disables itself if it does not have a target data source attached to an active client dataset.

See Also

TClientDataSetRevert (see page 8556)

TClientDataSetApply (see page 8548)

TCustomClientDataSet (see page 8373)

TDataSource (see page 7070)

TClientDataSet (see page 8289)

46.1.3.1 TClientDataSetUndo Members

The following tables list the members exposed by TClientDataSetUndo.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDataSet (see page 7655)	This is GetDataSet, a member of class TDataSetAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).

	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 7656)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 8568)	Backs out the last change to the current record of the target client dataset.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 7656)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 8569)	Enables or disables the action, depending on the state of the target dataset.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	DataSource (see page 8569)	Provides access to the target of the action.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	FollowChange (see page 8570)	Indicates whether the client dataset should keep its cursor on the same record, even after backing out the last edit.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

46.1.3.2 TClientDataSetUndo Methods

The methods of the TClientDataSetUndo class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 8568)	Backs out the last change to the current record of the target client dataset.
	UpdateTarget (see page 8569)	Enables or disables the action, depending on the state of the target dataset.

46.1.3.2.1 TClientDataSetUndo.ExecuteTarget Method

Backs out the last change to the current record of the target client dataset.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

You can call ExecuteTarget directly to execute this action, or it can be called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

ExecuteTarget checks that Target is a data source component whose DataSet property references a TClientDataSet instance. If the DataSource (see page 8569) property is set, Target must be the same data source.

If Target is a valid data source, ExecuteTarget calls the associated client dataset's UndoLastChange method, passing in the value of the FollowChange (see page 8570) property.

See Also

UndoLastChange (see page 8455)

DataSource (see page 7657)

DataSet (see page 7078)

FollowChange (see page 8570)

46.1.3.2.2 TClientDataSetUndo.UpdateTarget Method

Enables or disables the action, depending on the state of the target dataset.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TClientDataSetUndo (see page 8563), UpdateTarget checks whether the current target object (specified by the Target parameter) is a data source whose dataset is an active TClientDataSet instance that has changes in its change log. If so, it sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

46.1.3.3 TClientDataSetUndo Properties

The properties of the TClientDataSetUndo class are listed here.

Published Properties

	Name	Description
	DataSource (see page 8569)	Provides access to the target of the action.
	FollowChange (see page 8570)	Indicates whether the client dataset should keep its cursor on the same record, even after backing out the last edit.

46.1.3.3.1 TClientDataSetUndo.DataSource Property

Provides access to the target of the action.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource is the data source that represents the dataset on which the action performs its function. You can set this property to indicate the dataset that is the target of the action.

If you do not explicitly set the DataSource property, it is obtained from an active data-aware control when the action fires.

See Also

DataSet (see page 7078)

46.1.3.3.2 TClientDataSetUndo.FollowChange Property

Indicates whether the client dataset should keep its cursor on the same record, even after backing out the last edit.

Pascal

```
property FollowChange: Boolean;
```

C++

```
__property Boolean FollowChange;
```

Description

FollowChange indicates how the target client dataset should determine the current record after backing out the last edit. When FollowChange is true, the record whose edit was just backed out remains current. When FollowChange is false, the record in the current position remains current, even if undoing the last edit causes the record to move to a different position in the dataset's sort order.

TClientDataSetUndo (see page 8563) passes the value of FollowChange as the value of the FollowChange parameter when it calls the UndoLastChange method of the target client dataset.

See Also

UndoLastChange (see page 8455)

46.1.4 TReconcileActionLink Class

TReconcileActionLink helps a TClientDataSetApply (see page 8548) action display a dialog when the client dataset encounters reconcile errors.

Class Hierarchy

Pascal

```
TReconcileActionLink = class(TDataLink);
```

C++

```
class TReconcileActionLink : public TDataLink;
```

File

DBClientActns

Description

Applications should not need to use TReconcileActionLink directly. The TClientDataSetApply (see page 8548) class creates an instance of this data link internally when its DisplayErrorDlg property it true. This allows the TClientDataSetApply (see page

8548) instance to respond when its target (a data source) encounters reconcile errors. That is, when the client data set associated with the target data source has an OnReconcileError event, this data link notifies the TClientDataSetApply (see page 8548) instance, which displays a dialog that lets the use indicate how to respond to the error.

See Also

TClientDataSetApply (see page 8548)

TCustomClientDataSet (see page 8373)

TDataSource (see page 7070)

46.1.4.1 TReconcileActionLink Members

The following tables list the members exposed by TReconcileActionLink.

Protected Methods

	Name	Description
	ActiveChanged (see page 6880)	Responds to changes in the Active (see page 6888) property.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CheckBrowseMode (see page 6880)	Allows the data-aware object to respond just before an event that changes the dataset state.
	DataEvent (see page 8573)	Responds to various events that occur while working with data.
	DataSetChanged (see page 6881)	Responds to changes in the dataset.
	DataSetScrolled (see page 6882)	Allows a response to scrolling the representation of the dataset within the data-aware object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EditingChanged (see page 6883)	Responds to changes in the editing state of the data source.
	FocusControl (see page 6884)	Provides the interface for a method to force focus to the data-aware component associated with the TDataLink (see page 6876).
	GetActiveRecord (see page 6884)	This is GetActiveRecord, a member of class TDataLink.
	GetBOF (see page 6884)	This is GetBOF, a member of class TDataLink.
	GetBufferCount (see page 6884)	This is GetBufferCount, a member of class TDataLink.
	GetEOF (see page 6885)	This is GetEOF, a member of class TDataLink.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRecordCount (see page 6885)	This is GetRecordCount, a member of class TDataLink.
	LayoutChanged (see page 6885)	Responds to changes in the representation of the data by the data-aware object.
	MoveBy (see page 6885)	Repositions the dataset to another record relative to its active record.
	RecordChanged (see page 6886)	Responds to changes in the contents of the current record or field of the dataset.
	SetActiveRecord (see page 6886)	This is SetActiveRecord, a member of class TDataLink.
	SetBufferCount (see page 6887)	This is SetBufferCount, a member of class TDataLink.
	UpdateData (see page 6887)	Provides an interface for writing edits to a record in the dataset.

Protected Properties

	Name	Description
	VisualControl (see page 6894)	Indicates whether a visual control uses the datalink.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6878)	Creates an instance of TDataLink (see page 6876).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 6879)	Destroys an instance of TDataLink (see page 6876).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6882)	Attempts to drive the data source into an editing state if it is not already in one.
	ExecuteAction (see page 6883)	Invokes an action with the datasource as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	UpdateAction (see page 6887)	Updates an action to reflect the current state of the TDataLink (see page 6876).
	UpdateRecord (see page 6888)	Responds to notifications that edits should be posted to the dataset.

Public Properties

	Name	Description
R	Active (see page 6888)	Specifies whether the dataset that TDataLink (see page 6876) manages is active.
	ActiveRecord (see page 6889)	Specifies the index of the current record within the internal record buffer maintained by the dataset for the Owner of the TDataLink (see page 6876) object.
	ApplyAction (see page 8573)	Specifies the client dataset apply action that uses this action link to respond to reconcile errors.
R	BOF (see page 6889)	Indicates whether the first record in a dataset is active.
	BufferCount (see page 6890)	Specifies the number of records cached by the dataset for the Owner of the TDataLink (see page 6876) object.
R	DataSet (see page 6890)	Specifies the TDataSet (see page 6894) object that represents the data that this TDataLink (see page 6876) is tracking.
	DataSource (see page 6891)	Specifies the TDataSource (see page 7070) object that the Owner of the TDataLink (see page 6876) uses to connect to the dataset.
	DataSourceFixed (see page 6891)	Indicates whether the DataSource (see page 6891) property can be set.
R	Editing (see page 6892)	Indicates whether the DataSource (see page 6891) is in edit mode.
R	Eof (see page 6892)	Indicates whether the last record in a dataset is active.
	ReadOnly (see page 6893)	Indicates whether the dataset for this TDataLink (see page 6876) can be modified.

R	RecordCount (see page 6893)	Indicates the number of records in the internal record buffer maintained by the dataset.

46.1.4.2 TReconcileActionLink Methods

The methods of the TReconcileActionLink class are listed here.

Protected Methods

	Name	Description
V	DataEvent (see page 8573)	Responds to various events that occur while working with data.

46.1.4.2.1 TReconcileActionLink.DataEvent Method

Responds to various events that occur while working with data.

Pascal

```
procedure DataEvent(Event: TDataEvent; Info: Integer); override;
```

C++

```
virtual __fastcall DataEvent(TDataEvent Event, int Info);
```

Description

DataEvent is called automatically when various data events occur. Event indicates the event that just occurred. Info provides additional context information for some events.

In TReconcileActionLink (see page 8570), DataEvent first calls the method inherited from TDataLink to dispatch most events by calling appropriate methods. Then, if the data event is deReconcileError, it notifies the action specified by the ApplyAction (see page 8573) property so that it can display an error dialog.

See Also

DataEvent (see page 6881)

ApplyAction (see page 8573)

46.1.4.3 TReconcileActionLink Properties

The properties of the TReconcileActionLink class are listed here.

Public Properties

	Name	Description
	ApplyAction (see page 8573)	Specifies the client dataset apply action that uses this action link to respond to reconcile errors.

46.1.4.3.1 TReconcileActionLink.ApplyAction Property

Specifies the client dataset apply action that uses this action link to respond to reconcile errors.

Pascal

```
property ApplyAction: TClientDataSetApply;
```

C++

```
__property TClientDataSetApply ApplyAction;
```

Description

This property is set by the TClientDataSetApply (see page 8548) instance that owns this TReconcileActionLink (see page 8570) instance. It provides access to that TClientDataSetApply (see page 8548) instance.

When the client dataset that is the value of the DataSet property encounters an OnReconcileError event, the TReconcileActionLink (see page 8570) instance forwards this notification to the action that is the value of ApplyAction. If that action's DisplayErrorDlg property is true, it then responds by calling the HandleReconcileError function to display a dialog in which the user can specify how to respond to the update errors.

See Also

DisplayErrorDlg (see page 8556)

HandleReconcileError

DataSet (see page 6890)

OnReconcileError (see page 8524)

47 DBCtrls Namespace

This is namespace DBCtrls.

Topics

Name	Description
Classes (see page 8577)	The following table lists classes in this documentation.
Functions (see page 9536)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 9538)	The following table lists structs, records, enums in this documentation.
Types (see page 9541)	The following table lists types in this documentation.
Constants (see page 9543)	The following table lists constants in this documentation.

Classes

	Name	Description
	TDataSourceLink (see page 8577)	TDataSourceLink helps manage the link to the data source for a TDBLookupControl (see page 9044) object.
	TDBCheckBox (see page 8582)	TDBCheckBox is a data-aware control that allows the user to select or deselect a single value.
	TDBComboBox (see page 8657)	TDBComboBox represents a data-aware combo box control.
	TDBEdit (see page 8735)	TDBEdit represents a single-line edit control that can display and edit a field in a dataset.
	TDBImage (see page 8813)	TDBImage represents a graphic image from a BLOB (binary large object) field of the current record of a dataset.
	TDBListBox (see page 8884)	TDBListBox represents a data-aware list box that allows users to change field values by selecting an item from a list.
	TDBLookupComboBox (see page 8963)	TDBLookupComboBox represents a combo box that identifies a set of field values in one dataset with a corresponding set of values from another dataset.
	TDBLookupControl (see page 9044)	TDBLookupControl is the base class for controls that provide the user with a list of lookup items for filling in fields that require data from another dataset.
	TDBLookupListBox (see page 9070)	TDBLookupListBox provides a list of lookup items for filling in fields that require data from another dataset.
	TDBMemo (see page 9149)	TDBMemo represents a multiline edit control that can display and edit a field in a dataset.
	TDBNavigator (see page 9230)	TDBNavigator (the database navigator) is used to move through the data in a dataset and perform operations on the data, such as inserting a blank record or posting a record.
	TDBRadioGroup (see page 9283)	TDBRadioGroup represents a group of radio buttons connected to a database.
	TDBRichEdit (see page 9343)	TDBRichEdit represents a multiline edit control that can display and edit a rich text memo field in a dataset.
	TDBText (see page 9433)	TDBText represents a data-aware control that displays the value of a field on a form.

	TFieldDataLink (see page 9487)	TFieldDataLink helps a data-aware windowed control manage its link to a TField object and respond to data events.
	TListSourceLink (see page 9498)	TListSourceLink helps manage the link to the ListSource of a TDBLookupControl (see page 9044) object.
	TNavButton (see page 9502)	This is class DBCtrls.TNavButton.
	TNavDataLink (see page 9513)	TNavDataLink helps manage the link to the dataset for a TDBNavigator (see page 9230) object.
	TPaintControl (see page 9518)	TPaintControl draws data aware controls that are replicated in a data control grid.
	TPopupDataList (see page 9521)	This is class DBCtrls.TPopupDataList.

Constants

Name	Description
InitRepeatPause (see page 9543)	This is constant DBCtrls.InitRepeatPause.
RepeatPause (see page 9543)	This is constant DBCtrls.RepeatPause.
SpaceSize (see page 9543)	This is constant DBCtrls.SpaceSize.

Functions

	Name	Description
	DBUseRightToLeftAlignment (see page 9536)	Indicates whether a data-aware control's alignment is reversed to a right-to-left mode.
	OkToChangeFieldAlignment (see page 9536)	This is function DBCtrls.OkToChangeFieldAlignment.

Structs, Records, Enums

	Name	Description
	TDropDownAlign (see page 9538)	TDropDownAlign indicates the relative alignment of a drop-down list to the edit portion of a DB (see page 6725) lookup combo box.
	TNavGlyph (see page 9539)	This is record DBCtrls.TNavGlyph.
	TNavigateBtn (see page 9539)	TNavigateBtn defines values identifying different possible buttons on a TDBNavigator (see page 9230) object.

Types

Name	Description
ENavClick (see page 9541)	This is type DBCtrls.ENavClick.
TNavButtonSet (see page 9541)	This is type DBCtrls.TNavButtonSet.
TNavButtonStyle (see page 9541)	This is type DBCtrls.TNavButtonStyle.

47.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TDataSourceLink (see page 8577)	TDataSourceLink helps manage the link to the data source for a TDBLookupControl (see page 9044) object.
	TDBCheckBox (see page 8582)	TDBCheckBox is a data-aware control that allows the user to select or deselect a single value.
	TDBComboBox (see page 8657)	TDBComboBox represents a data-aware combo box control.
	TDBEdit (see page 8735)	TDBEdit represents a single-line edit control that can display and edit a field in a dataset.
	TDBImage (see page 8813)	TDBImage represents a graphic image from a BLOB (binary large object) field of the current record of a dataset.
	TDBListBox (see page 8884)	TDBListBox represents a data-aware list box that allows users to change field values by selecting an item from a list.
	TDBLookupComboBox (see page 8963)	TDBLookupComboBox represents a combo box that identifies a set of field values in one dataset with a corresponding set of values from another dataset.
	TDBLookupControl (see page 9044)	TDBLookupControl is the base class for controls that provide the user with a list of lookup items for filling in fields that require data from another dataset.
	TDBLookupListBox (see page 9070)	TDBLookupListBox provides a list of lookup items for filling in fields that require data from another dataset.
	TDBMemo (see page 9149)	TDBMemo represents a multiline edit control that can display and edit a field in a dataset.
	TDBNavigator (see page 9230)	TDBNavigator (the database navigator) is used to move through the data in a dataset and perform operations on the data, such as inserting a blank record or posting a record.
	TDBRadioGroup (see page 9283)	TDBRadioGroup represents a group of radio buttons connected to a database.
	TDBRichEdit (see page 9343)	TDBRichEdit represents a multiline edit control that can display and edit a rich text memo field in a dataset.
	TDBText (see page 9433)	TDBText represents a data-aware control that displays the value of a field on a form.
	TFieldDataLink (see page 9487)	TFieldDataLink helps a data-aware windowed control manage its link to a TField object and respond to data events.
	TListSourceLink (see page 9498)	TListSourceLink helps manage the link to the ListSource of a TDBLookupControl (see page 9044) object.
	TNavButton (see page 9502)	This is class DBCtrls.TNavButton.
	TNavDataLink (see page 9513)	TNavDataLink helps manage the link to the dataset for a TDBNavigator (see page 9230) object.
	TPaintControl (see page 9518)	TPaintControl draws data aware controls that are replicated in a data control grid.
	TPopupDataList (see page 9521)	This is class DBCtrls.TPopupDataList.

47.1.1 TDataSourceLink Class

TDataSourceLink helps manage the link to the data source for a TDBLookupControl (see page 9044) object.

Class Hierarchy

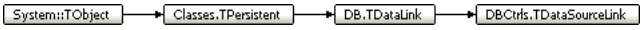

Pascal

```
TDataSourceLink = class(TDataLink);
```

C++

```
class TDataSourceLink : public TDataLink;
```

File

DBCtrls

Description

TDataSourceLink is tailored to work with TDBLookupControl (see page 9044). It should only be used by the TDBLookupControl (see page 9044) class.

See Also

TDataSource (see page 7070)

TDBLookupControl (see page 9044)

47.1.1.1 TDataSourceLink Members

The following tables list the members exposed by TDataSourceLink.

Protected Methods

	Name	Description
	ActiveChanged (see page 8580)	Informs the TDBLookupControl (see page 9044) of changes in the Active property of its DataSource.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CheckBrowseMode (see page 6880)	Allows the data-aware object to respond just before an event that changes the dataset state.
	DataEvent (see page 6881)	Responds to various events that occur while working with data.
	DataSetChanged (see page 6881)	Responds to changes in the dataset.
	DataSetScrolled (see page 6882)	Allows a response to scrolling the representation of the dataset within the data-aware object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EditingChanged (see page 6883)	Responds to changes in the editing state of the data source.
	FocusControl (see page 8581)	Moves input focus to the data field of the TDBLookupControl (see page 9044) object.
	GetActiveRecord (see page 6884)	This is GetActiveRecord, a member of class TDataLink.
	GetBOF (see page 6884)	This is GetBOF, a member of class TDataLink.
	GetBufferCount (see page 6884)	This is GetBufferCount, a member of class TDataLink.
	GetEOF (see page 6885)	This is GetEOF, a member of class TDataLink.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRecordCount (see page 6885)	This is GetRecordCount, a member of class TDataLink.
	LayoutChanged (see page 8581)	Responds to changes in the representation of the data.
	MoveBy (see page 6885)	Repositions the dataset to another record relative to its active record.
	RecordChanged (see page 8581)	Informs the TDBLookupControl (see page 9044) of changes in the current record or field of the DataSource.
	SetActiveRecord (see page 6886)	This is SetActiveRecord, a member of class TDataLink.
	SetBufferCount (see page 6887)	This is SetBufferCount, a member of class TDataLink.
	UpdateData (see page 6887)	Provides an interface for writing edits to a record in the dataset.

Protected Properties

	Name	Description
	VisualControl (see page 6894)	Indicates whether a visual control uses the datalink.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 8580)	Creates an instance of TDataLink.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 6879)	Destroys an instance of TDataLink (see page 6876).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6882)	Attempts to drive the data source into an editing state if it is not already in one.
	ExecuteAction (see page 6883)	Invokes an action with the datasource as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	UpdateAction (see page 6887)	Updates an action to reflect the current state of the TDataLink (see page 6876).
	UpdateRecord (see page 6888)	Responds to notifications that edits should be posted to the dataset.

Public Properties

	Name	Description
R	Active (see page 6888)	Specifies whether the dataset that TDataLink (see page 6876) manages is active.
	ActiveRecord (see page 6889)	Specifies the index of the current record within the internal record buffer maintained by the dataset for the Owner of the TDataLink (see page 6876) object.
R	BOF (see page 6889)	Indicates whether the first record in a dataset is active.
	BufferCount (see page 6890)	Specifies the number of records cached by the dataset for the Owner of the TDataLink (see page 6876) object.
R	DataSet (see page 6890)	Specifies the TDataSet (see page 6894) object that represents the data that this TDataLink (see page 6876) is tracking.
	DataSource (see page 6891)	Specifies the TDataSource (see page 7070) object that the Owner of the TDataLink (see page 6876) uses to connect to the dataset.

	DataSourceFixed (see page 6891)	Indicates whether the DataSource (see page 6891) property can be set.
R	Editing (see page 6892)	Indicates whether the DataSource (see page 6891) is in edit mode.
R	Eof (see page 6892)	Indicates whether the last record in a dataset is active.
	ReadOnly (see page 6893)	Indicates whether the dataset for this TDataLink (see page 6876) can be modified.
R	RecordCount (see page 6893)	Indicates the number of records in the internal record buffer maintained by the dataset.

47.1.1.2 TDataSourceLink.Create Constructor

Creates an instance of TDataLink.

Pascal

```
constructor Create;
```

C++

```
__fastcall TDataSourceLink();
```

Description

Create is called from the constructor of any data-aware object that uses a TDataLink to implement its DataSource property.

After calling the inherited constructor, Create initializes the BufferCount property to 1. Data-aware objects that use a TDataLink object to manage their link to a DataSource should change the BufferCount property to the number or records they represent, after calling the inherited constructor.

See Also

BufferCount (see page 6890)

47.1.1.3 TDataSourceLink Methods

The methods of the TDataSourceLink class are listed here.

Protected Methods

	Name	Description
V	ActiveChanged (see page 8580)	Informs the TDBLookupControl (see page 9044) of changes in the Active property of its DataSource.
V	FocusControl (see page 8581)	Moves input focus to the data field of the TDBLookupControl (see page 9044) object.
V	LayoutChanged (see page 8581)	Responds to changes in the representation of the data.
V	RecordChanged (see page 8581)	Informs the TDBLookupControl (see page 9044) of changes in the current record or field of the DataSource.

47.1.1.3.1 TDataSourceLink.ActiveChanged Method

Informs the TDBLookupControl (see page 9044) of changes in the Active property of its DataSource.

Pascal

```
procedure ActiveChanged; override;
```

C++

```
virtual __fastcall ActiveChanged();
```

Description

Applications cannot call the ActiveChanged procedure directly. Changes in the Active property trigger this procedure. When the

Active property becomes true, ActiveChanged causes the TDBLookupControl (see page 9044) object to re-acquire its DataField property. This ensures that the lookup control is working with the correct field component.

See Also

Active (see page 6888)

Active (see page 7004)

State (see page 7080)

DataField (see page 9062)

47.1.1.3.2 TDataSourceLink.FocusControl Method

Moves input focus to the data field of the TDBLookupControl (see page 9044) object.

Pascal

```
procedure FocusControl(Field: TFieldRef); override;
```

C++

```
virtual __fastcall FocusControl(TFieldRef Field);
```

Description

Applications cannot call the FocusControl procedure. It is called automatically when a field component in the DataSet tries to set focus to a control on the form that represents the field. FocusControl moves focus to the DataField of the TDBLookupControl (see page 9044) if it matches the Field parameter.

See Also

DataField (see page 9062)

FocusControl (see page 7128)

47.1.1.3.3 TDataSourceLink.LayoutChanged Method

Responds to changes in the representation of the data.

Pascal

```
procedure LayoutChanged; override;
```

C++

```
virtual __fastcall LayoutChanged();
```

Description

LayoutChanged updates the associated lookup control so that its field bindings remain accurate when the field order in the dataset changes.

See Also

DataSetChanged (see page 6881)

47.1.1.3.4 TDataSourceLink.RecordChanged Method

Informs the TDBLookupControl (see page 9044) of changes in the current record or field of the DataSource.

Pascal

```
procedure RecordChanged(Field: TField); override;
```

C++

```
virtual __fastcall RecordChanged(TField Field);
```

Description

Applications cannot call RecordChanged directly. It is triggered automatically after changes in the contents of the current record. If the Field parameter is the key field of the lookup control, or if the Field parameter is nil (Delphi) or NULL (C++), indicating that the entire record changed, RecordChanged causes the lookup control to refetch the value of its key field.

See Also

KeyField (see page 9064)

UpdateData (see page 6887)

47.1.2 TDBCheckBox Class

TDBCheckBox is a data-aware control that allows the user to select or deselect a single value.

Class Hierarchy

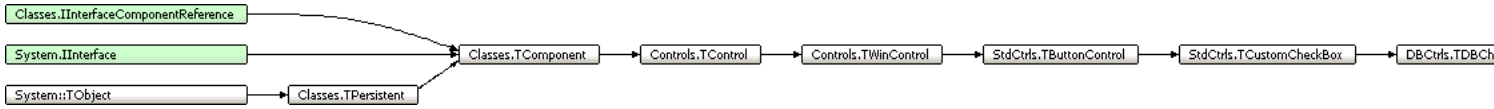

Pascal

```
TDBCheckBox = class(TCustomCheckBox);
```

C++

```
class TDBCheckBox : public TCustomCheckBox;
```

File

DBCtrls

Description

Use TDBCheckBox to place a data-aware check box on a form. A check box presents an option to the user; the user can check it to select the option or uncheck it to deselect the option. A database check box (TDBCheckBox) is much like an ordinary check box (TCheckBox), except that it is aware of the data in a particular field of a dataset. Because check boxes can represent only two values (checked and unchecked), database check boxes are most appropriate for boolean fields. They can, however, be used to group the values of any field into two sets.

For applications that don't require the data-aware capabilities of TDBCheckBox, use TCheckBox instead to conserve system resources.

See Also

TBooleanField (see page 6822)

TCheckBox (see page 20996)

Handling Boolean Field Values with Check Boxes

Using Common Data Control Features

Check Boxes

47.1.2.1 TDBCheckBox Members

The following tables list the members exposed by TDBCheckBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
V	ActionChange (see page 20992)	Updates the button control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.

	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	Click (see page 21149)	Click generates an OnClick event.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 20993)	Initializes a window-creation parameter data structure.
	CreateParams (see page 21149)	Initializes a window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 21150)	Creates a Windows control corresponding to the check box.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus

	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetActionLinkClass (see page 20993)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 20993)	Returns whether the button control is checked.
	GetChecked (see page 21150)	Returns the value of the Checked (see page 20995) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
V	KeyPress (see page 8595)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 8595)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
V	SetChecked (see page 20994)	Sets the Checked (see page 20995) property.
V	SetChecked (see page 21151)	Sets the value of the Checked (see page 20995) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	Toggle (see page 8596)	Changes the state of the check box.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 8597)	Handles Windows messages.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.

	BorderWidth (see page 6195)	Specifies the width of the control's border.
	ClicksDisabled (see page 20996)	This is ClicksDisabled, a member of class TButtonControl.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 8593)	Creates an instance of TDBCheckBox (see page 8582).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 8594)	Destroys an instance of TDBCheckBox (see page 8582).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 8594)	Invokes an action with the TDBCheckBox (see page 8582) as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetControlsAlignment (see page 21151)	Returns the alignment of the check box caption.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.

	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
V	UpdateAction (see page 8596)	Updates an action to reflect the current state of the TDBCheckBox (see page 8582).
	UpdateControlState (see page 6186)	Respond to state change.
V	UseRightToLeftAlignment (see page 8596)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.

	Checked (see page 8607)	Specifies whether the button control is checked.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Field (see page 8613)	Indicates the TField object whose current value the check box represents.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	State (see page 8623)	Indicates whether the check box is selected, deselected, or grayed.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 8629)	Occurs when the user clicks the control.
	OnContextPopup (see page 8631)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 8631)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 8634)	Occurs when the user drags an object over a control.
	OnEndDock (see page 8636)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 8637)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 8638)	Occurs when a control receives the input focus.
	OnExit (see page 8640)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 8642)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 8644)	Occurs when a key is pressed.
	OnKeyUp (see page 8645)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 8647)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 8648)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 8650)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 8651)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 8651)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 8653)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 8655)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 8656)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Action (see page 8598)	Designates the action associated with the control.
	Align (see page 8599)	Determines how the control aligns within its container (parent control).
	Alignment (see page 8602)	This is Alignment, a member of class TDBCheckBox.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AllowGrayed (see page 8602)	Determines whether check box can be in a "grayed" state.
	Anchors (see page 8603)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 8604)	Specifies the bi-directional mode for the control.
	Caption (see page 8604)	Specifies a text string that identifies the control to the user.
	Color (see page 8608)	Specifies the background color of the control.
	Constraints (see page 8609)	Specifies the size constraints for the control.
	Ctl3D (see page 8609)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DataField (see page 8610)	Identifies the field from which the data-aware control displays data.
	DataSource (see page 8610)	Links the check box to a dataset.
	DragCursor (see page 8611)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 8611)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 8611)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 8613)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 8614)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic

	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 8615)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 8615)	Determines where a control looks for its color information.
	ParentCtl3D (see page 8616)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 8616)	Determines where a control looks for its font information.
	ParentShowHint (see page 8616)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 8617)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 8622)	Determines whether the user can change the value of the field.
	ShowHint (see page 8622)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 8624)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 8625)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	ValueChecked (see page 8625)	Specifies the field value that corresponds to the checked state of the check box.
	ValueUnchecked (see page 8626)	Specifies the field value that corresponds to the unchecked state of the check box.
	Visible (see page 8626)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	WordWrap (see page 8628)	Specifies whether the button text wraps to fit the width of the control.

47.1.2.2 TDBCheckBox.Create Constructor

Creates an instance of TDBCheckBox (see page 8582).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDBCheckBox(TComponent * AOwner);
```

Description

Call Create to instantiate a TDBCheckBox (see page 8582) object at runtime. For TDBCheckBox (see page 8582) objects placed on forms or data modules at design time, Create is called automatically.

Pass a single Component as a parameter to provide the TDBCheckBox (see page 8582) object with an Owner that is responsible for freeing it.After calling the inherited constructor to allocate and initialize a generic check box, Create

Adds csReplicatable to ControlStyle.

Creates and initializes the helper objects needed to link to a dataset.

See Also

Destroy (see page 8594)

Create (see page 21148)

TFieldDataLink (see page 9487)

47.1.2.3 TDBCheckBox.Destroy Destructor

Destroys an instance of TDBCheckBox (see page 8582).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBCheckBox();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TDBCheckBox (see page 8582) reference is not nil and only then calls Destroy.

Destroy frees the helper objects of the TDBCheckBox (see page 8582) before calling the inherited destructor.

See Also

Create (see page 8593)

Free (see page 22300)

47.1.2.4 TDBCheckBox Methods

The methods of the TDBCheckBox class are listed here.

Protected Methods

	Name	Description
	KeyPress (see page 8595)	Respond to keyboard input.
	Notification (see page 8595)	Responds to notifications that components are being created or destroyed.
	Toggle (see page 8596)	Changes the state of the check box.
	WndProc (see page 8597)	Handles Windows messages.

Public Methods

	Name	Description
	ExecuteAction (see page 8594)	Invokes an action with the TDBCheckBox (see page 8582) as its target.
	UpdateAction (see page 8596)	Updates an action to reflect the current state of the TDBCheckBox (see page 8582).
	UseRightToLeftAlignment (see page 8596)	Specifies whether the control's alignment should be reversed right-to-left.

47.1.2.4.1 TDBCheckBox.ExecuteAction Method

Invokes an action with the TDBCheckBox (see page 8582) as its target.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

ExecuteAction is called automatically when the user invokes an action with the TDBCheckBox (see page 8582) as a target. The Action (see page 8598) parameter specifies the action that was invoked. ExecuteAction returns trueif the action was successfully dispatched, and false if the TDBCheckBox (see page 8582) could not handle the action.

See Also

UpdateAction (see page 8596)

Execute (see page 2712)

HandlesTarget (see page 2713)

47.1.2.4.2 TDBCheckBox.KeyPress Method

Respond to keyboard input.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

When a windowed control receives a key-press message (WM_CHAR) from Windows, its message handler calls the DoKeyPress method. If DoKeyPress determines that the control should, in fact, process the character, it calls KeyPress, passing the key code in the Key parameter.

KeyPress calls any event handler attached to the OnKeyPress event. Override KeyPress to provide other responses in addition to the event-handler call.

Either KeyPress or the OnKeyPress event handler it calls can suppress further processing of a character by setting the Key parameter to zero.

Note: The Key parameter is the character represented by the key that is pressed, not a Windows virtual key code.

See Also

KeyDown (see page 6163)

KeyUp (see page 6164)

OnKeyPress (see page 6227)

47.1.2.4.3 TDBCheckBox.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

47.1.2.4.4 TDBCheckBox.Toggle Method

Changes the state of the check box.

Pascal

```
procedure Toggle; override;
```

C++

```
virtual __fastcall Toggle();
```

Description

Use the Toggle method to select or deselect the check box programmatically. Toggle switches among the valid check-box states—from checked to unchecked to grayed, or from checked to unchecked.

See Also

AllowGrayed (see page 21152)

State (see page 21153)

47.1.2.4.5 TDBCheckBox.UpdateAction Method

Updates an action to reflect the current state of the TDBCheckBox (see page 8582).

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

UpdateAction is called automatically when the application is idle to give the TDBCheckBox (see page 8582) an opportunity to update associated actions. This allows actions to be grayed, checked, or to otherwise reflect the status of the TDBCheckBox (see page 8582) they target.

The Action (see page 8598) parameter specifies the action component that should be updated. UpdateAction returns trueif the action component now reflects the state of the component, and false if it did not know how to update the action.

See Also

ExecuteAction (see page 8594)

UpdateTarget (see page 2715)

HandlesTarget (see page 2713)

47.1.2.4.6 TDBCheckBox.UseRightToLeftAlignment Method

Specifies whether the control's alignment should be reversed right-to-left.

Pascal

```
function UseRightToLeftAlignment: Boolean; override;
```

C++

```
virtual __fastcall Boolean UseRightToLeftAlignment();
```

Description

Controls call UseRightToLeftAlignment when implementing the BiDiMode property. UseRightToLeftAlignment indicates whether the control's alignment should be reversed right-to-left. This information can be used when painting the control or when responding to mouse events.

UseRightToLeftAlignment returns true if the application is running on a system with a Middle Eastern locale and the BiDiMode property is bdRightToLeft. Otherwise, it returns false. Descendants override UseRightToLeftAlignment if alignment in Middle Eastern locales depends on other factors.

See Also

BiDiMode (see page 5803)

GetControlsAlignment (see page 5759)

Text (see page 5842)

Caption (see page 5806)

47.1.2.4.7 TDBCheckBox.WndProc Method

Handles Windows messages.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc enables button objects to receive input focus when WM_LBUTTONDOWN and WM_LBUTTONDBLCLK messages occur.

See Also

WndProc (see page 5791)

47.1.2.5 TDBCheckBox Properties

The properties of the TDBCheckBox class are listed here.

Public Properties

	Name	Description
	Checked (see page 8607)	Specifies whether the button control is checked.
R	Field (see page 8613)	Indicates the TField object whose current value the check box represents.
	State (see page 8623)	Indicates whether the check box is selected, deselected, or grayed.

Published Properties

	Name	Description
	Action (see page 8598)	Designates the action associated with the control.

	Align (see page 8599)	Determines how the control aligns within its container (parent control).
	Alignment (see page 8602)	This is Alignment, a member of class TDBCheckBox.
	AllowGrayed (see page 8602)	Determines whether check box can be in a "grayed" state.
	Anchors (see page 8603)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 8604)	Specifies the bi-directional mode for the control.
	Caption (see page 8604)	Specifies a text string that identifies the control to the user.
	Color (see page 8608)	Specifies the background color of the control.
	Constraints (see page 8609)	Specifies the size constraints for the control.
	Ctl3D (see page 8609)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DataField (see page 8610)	Identifies the field from which the data-aware control displays data.
	DataSource (see page 8610)	Links the check box to a dataset.
	DragCursor (see page 8611)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 8611)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 8611)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 8613)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 8614)	Controls the attributes of text written on or in the control.
	ParentBiDiMode (see page 8615)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 8615)	Determines where a control looks for its color information.
	ParentCtl3D (see page 8616)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 8616)	Determines where a control looks for its font information.
	ParentShowHint (see page 8616)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 8617)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 8622)	Determines whether the user can change the value of the field.
	ShowHint (see page 8622)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 8624)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 8625)	Determines if the user can tab to a control.
	ValueChecked (see page 8625)	Specifies the field value that corresponds to the checked state of the check box.
	ValueUnchecked (see page 8626)	Specifies the field value that corresponds to the unchecked state of the check box.
	Visible (see page 8626)	Determines whether the component appears onscreen.
	WordWrap (see page 8628)	Specifies whether the button text wraps to fit the width of the control.

47.1.2.5.1 TDBCheckBox.Action Property

Designates the action associated with the control.

Pascal

```
property Action: TBasicAction;
```

C++

```
__property TBasicAction Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

47.1.2.5.2 TDBCheckBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
```

```
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
```

```
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
```

```
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

47.1.2.5.3 TDBCheckBox.Alignment Property

Pascal

```
property Alignment: TLeftRight;
```

C++

```
__property TLeftRight Alignment;
```

Description

This is Alignment, a member of class TDBCheckBox.

47.1.2.5.4 TDBCheckBox.AllowGrayed Property

Determines whether check box can be in a "grayed" state.

Pascal

```
property AllowGrayed: Boolean;
```

C++

```
__property Boolean AllowGrayed;
```

Description

If AllowGrayed is set to true, the check box has three possible states: checked, unchecked, and grayed. If AllowGrayed is set to

false, the check box has only two possible states: checked and unchecked.

C++ Examples:

```
/*
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->AllowGrayed = true;
  CheckBox1->State = cbChecked;
}
```

Delphi Examples:

```
{
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  CheckBox1.AllowGrayed := True;
  CheckBox1.State := cbChecked;
end;
```

See Also

Checked

State (see page 21153)

47.1.2.5.5 TDBCheckBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

47.1.2.5.6 TDBCheckBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

47.1.2.5.7 TDBCheckBox.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

```
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
```

```
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

47.1.2.5.8 TDBCheckBox.Checked Property

Specifies whether the button control is checked.

Pascal

```
property Checked: Boolean;
```

C++

```
__property Boolean Checked;
```

Description

Use Checked to determine whether a button control is checked.

C++ Examples:

```
/*
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
*/

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->MultiLine = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->Caption = "Stack Tabs?";
  for (int i = 0; i < 20; i++)
  {
    TTabSheet *pPage = new TTabSheet(PageControl1);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
  }
}
```

Delphi Examples:

```
{
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
}
```

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.MultiLine := CheckBox1.Checked;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Checkbox1.Caption := 'Stack Tabs?';
  for i := 0 to 20 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'Tab #' + IntToStr(i);
    end;
end;
```

See Also

WndProc (see page 5791)

47.1.2.5.9 TDBCheckBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

47.1.2.5.10 TDBCheckBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

47.1.2.5.11 TDBCheckBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

47.1.2.5.12 TDBCheckBox.DataField Property

Identifies the field from which the data-aware control displays data.

Pascal

```
property DataField: WideString;
```

C++

```
__property BSTR DataField;
```

Description

Set DataField to the field name of the field component that the check box represents. Access by the TDBCheckBox (see page 8582) to the dataset in which the field is located is provided by a TDataSource component, specified in the DataSource (see page 8610) property.

```
DBCheckBox1.DataField := 'HasPaid';
DBCheckBox1->DataField = "HasPaid";
```

See Also

DataSource (see page 8610)

Field (see page 8613)

47.1.2.5.13 TDBCheckBox.DataSource Property

Links the check box to a dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Specify the data source component through which the data from a dataset component is provided to the TDBCheckBox (see page 8582). To allow the check box to represent the data for a field, both the DataSource and the DataField (see page 8610) properties must be set.

```
DBCheckBox1.DataSource := DataSource1;
DBCheckBox1->DataSource = DataSource1;
```

See Also

DataField (see page 8610)

Field (see page 8613)

47.1.2.5.14 TDBCheckBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

47.1.2.5.15 TDBCheckBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

47.1.2.5.16 TDBCheckBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

47.1.2.5.17 TDBCheckBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

47.1.2.5.18 TDBCheckBox.Field Property

Indicates the TField object whose current value the check box represents.

Pascal

```
property Field: TField;
```

C++

```
__property TField Field;
```

Description

Use the TField reference provided by the Field property when you want to change the value of the data in the field programmatically.

```
DBCheckBox1.Field.AsBoolean := True;
```

```
DBCheckBox1->Field->AsBoolean = true;
```

See Also

DataField (see page 8610)

DataSource (see page 8610)

47.1.2.5.19 TDBCheckBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
```

```
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

47.1.2.5.20 TDBCheckBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

47.1.2.5.21 TDBCheckBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

47.1.2.5.22 TDBCheckBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

47.1.2.5.23 TDBCheckBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

47.1.2.5.24 TDBCheckBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

47.1.2.5.25 TDBCheckBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
```

```
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
```

```
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;
```

```
{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);
```

```
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

47.1.2.5.26 TDBCheckBox.ReadOnly Property

Determines whether the user can change the value of the field.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to specify whether the check box allows the user to change the field value by checking or unchecking the box. When ReadOnly is true, the check box can only be used to display the value of the field on the current record. When ReadOnly is false, the user can use the check box to change the field value.

See Also

Field (see page 8613)

47.1.2.5.27 TDBCheckBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
```

```
    {
      strcpy(string, "Item");
      itoa(i, index, 10);
      strcat(string, index);
      ListBox1->Items->Add(string);
    }
    ListBox1->Hint = "Select an item";
    ListBox1->ShowHint = true;
    Application->ShowHint = true;
  }
  /*
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  */
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

47.1.2.5.28 TDBCheckBox.State Property

Indicates whether the check box is selected, deselected, or grayed.

Pascal

```
property State: TCheckBoxState;
```

C++

```
__property TCheckBoxState State;
```

Description

Use State to determine whether the check box is checked (cbChecked), unchecked (cbUnchecked), or grayed (cbGrayed).

C++ Examples:

```
/*
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->AllowGrayed = true;
  CheckBox1->State = cbChecked;
}
```

Delphi Examples:

```
{
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  CheckBox1.AllowGrayed := True;
  CheckBox1.State := cbChecked;
end;
```

See Also

AllowGrayed (see page 21152)

Checked

47.1.2.5.29 TDBCheckBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the

control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

47.1.2.5.30 TDBCheckBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

47.1.2.5.31 TDBCheckBox.ValueChecked Property

Specifies the field value that corresponds to the checked state of the check box.

Pascal

```
property ValueChecked: string;
```

C++

```
__property AnsiString ValueChecked;
```

Description

Use ValueChecked to specify the field value represented by the check box when it is checked. If the value of the ValueChecked property is equal to the data in the field of the current record of the dataset, the database check box appears checked. When the user checks the database check box, the field value is set to ValueChecked.

ValueChecked can represent more than one value in a semicolon-delimited list of items. If any of the items matches the contents of the field of the current record in the dataset, the check box appears checked. For example, set the value of ValueChecked string like this:

```
DBCheckBox1.ValueChecked := 'True;Yes;On';
DBCheckBox1->ValueChecked = "True;Yes;On";
```

If the contents of the associated field is the string true, Yes, or On, the check box is checked. The value of the field is compared to ValueChecked in a case-insensitive comparison. If the user selects a check box where ValueChecked represents more than one value, the first item in the list is assigned to the field.

If the contents of the field of the current record matches a string specified as the value of the ValueUnchecked (see page 8626) property, the check box appears unchecked. If the contents of the field matches no string in either ValueChecked or ValueUnchecked (see page 8626), the check box appears gray.

Note: If the DataField (see page 8610) of the database check box is a logical field, the check box is always checked if the contents of the field is true, and it is always unchecked if the contents of the field is false. The values of the ValueChecked and ValueUnchecked (see page 8626) properties have no affect on logical fields.

See Also

TBooleanField (see page 6822)

ValueUnchecked (see page 8626)

Handling Boolean Field Values with Check Boxes

47.1.2.5.32 TDBCheckBox.ValueUnchecked Property

Specifies the field value that corresponds to the unchecked state of the check box.

Pascal

```
property ValueUnchecked: string;
```

C++

```
__property AnsiString ValueUnchecked;
```

Description

Use ValueUnchecked to specify the field value represented by the check box when it is unchecked. If the value of the ValueUnchecked property is equal to the data in the field of the current record of the dataset, the database check box appears unchecked. When the user unchecks the database check box, the field value is set to ValueUnchecked.

ValueUnchecked can represent more than one value in a semicolon-delimited list of items. If any of the items matches the contents of the field of the current record in the dataset, the check box appears unchecked. For example, set the value of ValueUnchecked string like this:

```
DBCheckBox1.ValueUnchecked := 'False;No;Off';
DBCheckBox1->ValueUnchecked = "False;No;Off";
```

If the contents of the associated field is the string false, No, or Off, the check box appears unchecked. The value of the field is compared to ValueUnchecked in a case-insensitive comparison. If the user unchecks a check box where ValueUnchecked represents more than one value, the first item in the list is assigned to the field.

If the contents of the field of the current record matches a string specified as the value of the ValueChecked (see page 8625) property, the check box appears checked. If the contents of the field matches no string in either ValueChecked (see page 8625) or ValueUnchecked, the check box appears gray.

Note: If the DataField (see page 8610) of the database check box is a logical field, the check box is always unchecked if the contents of the field is false, and it is always checked if the contents of the field is true. The values of the ValueChecked (see page 8625) and ValueUnchecked properties have no affect on logical fields.

See Also

ValueChecked (see page 8625)

Handling Boolean Field Values with Check Boxes

47.1.2.5.33 TDBCheckBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

47.1.2.5.34 TDBCheckBox.WordWrap Property

Specifies whether the button text wraps to fit the width of the control.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the display of multiple lines of text. When WordWrap is true, text that is too wide for the control wraps at the right margin.

Set WordWrap to false to limit the text to a single line. When WordWrap is false, text that is too wide for the control appears truncated.

See Also

ClientWidth (see page 5811)

47.1.2.6 TDBCheckBox Events

The events of the TDBCheckBox class are listed here.

Published Events

	Name	Description
	OnClick (see page 8629)	Occurs when the user clicks the control.
	OnContextPopup (see page 8631)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 8631)	Occurs when the user drops an object being dragged.

	OnDragOver (see page 8634)	Occurs when the user drags an object over a control.
	OnEndDock (see page 8636)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 8637)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 8638)	Occurs when a control receives the input focus.
	OnExit (see page 8640)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 8642)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 8644)	Occurs when a key is pressed.
	OnKeyUp (see page 8645)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 8647)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 8648)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 8650)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 8651)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 8651)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 8653)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 8655)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 8656)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

47.1.2.6.1 TDBCheckBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
```

```
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

47.1.2.6.2 TDBCheckBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

47.1.2.6.3 TDBCheckBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

47.1.2.6.4 TDBCheckBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.2.6.5 TDBCheckBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.2.6.6 TDBCheckBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

47.1.2.6.7 TDBCheckBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
```

```
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

47.1.2.6.8 TDBCheckBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

47.1.2.6.9 TDBCheckBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
```

```
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

47.1.2.6.10 TDBCheckBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

47.1.2.6.11 TDBCheckBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
```

```
    FormColor: TColor;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    KeyPreview := True;
    FormColor := Form1.Color;
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

47.1.2.6.12 TDBCheckBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as

maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.2.6.13 TDBCheckBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
```

```
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
```

```
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.2.6.14 TDBCheckBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

47.1.2.6.15 TDBCheckBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

47.1.2.6.16 TDBCheckBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}
```

```
// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
```

```
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.2.6.17 TDBCheckBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
```

```
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

47.1.2.6.18 TDBCheckBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

47.1.2.6.19 TDBCheckBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

47.1.3 TDBComboBox Class

TDBComboBox represents a data-aware combo box control.

Class Hierarchy

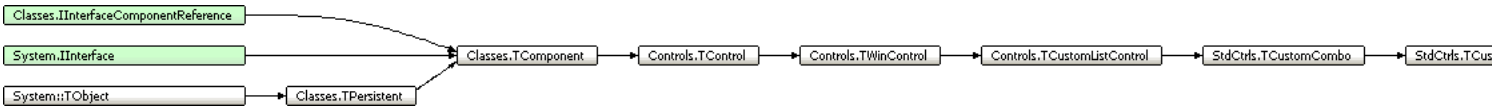

Pascal

```
TDBComboBox = class(TCustomComboBox);
```

C++

```
class TDBComboBox : public TCustomComboBox;
```

File

DBCtrls

Description

Use TDBComboBox to allow users to change the value of a field on the current record in a dataset either by selecting an item from a list or by typing in the edit box part of the control. The selected item or entered text becomes the new value of the field if the database combo box's ReadOnly (see page 8703) property is false. The combo box can be customized to enable or disable typing in the edit region of the control, to display the list as a drop down or as a permanently displayed list, to sort the items in the list, and so on.

See Also

TDBListBox (see page 8884)

TDBLookupComboBox (see page 8963)

Displaying and Editing Data in List and Combo Boxes

Using Common Data Control Features

47.1.3.1 TDBComboBox Members

The following tables list the members exposed by TDBComboBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.

	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnCloseUp (see page 21199)	Occurs when the drop-down list closes up due to some user action.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnSelect (see page 21201)	Occurs when the user selects a string in the drop-down list.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefEditProc (see page 21167)	This is FDefEditProc, a member of class TCustomCombo.
	FDefListProc (see page 21167)	This is FDefListProc, a member of class TCustomCombo.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FDropHandle (see page 21168)	This is FDropHandle, a member of class TCustomCombo.
	FDroppingDown (see page 21168)	This is FDroppingDown, a member of class TCustomCombo.
	FEditHandle (see page 21168)	This is FEditHandle, a member of class TCustomCombo.
	FEditInstance (see page 21168)	This is FEditInstance, a member of class TCustomCombo.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFocusChanged (see page 21168)	This is FFocusChanged, a member of class TCustomCombo.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FIsFocused (see page 21169)	This is FIsFocused, a member of class TCustomCombo.
	FListHandle (see page 21169)	This is FListHandle, a member of class TCustomCombo.
	FListInstance (see page 21169)	This is FListInstance, a member of class TCustomCombo.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveIndex (see page 21169)	This is FSaveIndex, a member of class TCustomCombo.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.

	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustDropDown (see page 21171)	Adjusts the size and position of the drop down list.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 8671)	Generates an OnChange event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 8671)	Respond to user click.
	CloseUp (see page 21172)	Generates an OnCloseUp (see page 21199) event.
	ComboWndProc (see page 8671)	Responds to Windows messages sent to the edit control or drop-down list.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 21215)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 8672)	Creates the Windows control that represents the combo box.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWindowHandle (see page 21174)	Destroys all of the windows controls that implement the combo box.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21215)	Destroys the Windows control that represents the combo box.

	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawItem (see page 21216)	Generates an OnDrawItem (see page 21224) event.
V	DropDown (see page 8672)	Generates an OnDropDown event.
V	EditWndProc (see page 21175)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.

	GetDroppedDown (see page 21176)	Returns the value of the DroppedDown (see page 21185) property.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
A	GetItemCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
V	GetItemCount (see page 21216)	Returns the number of items in the Items (see page 21187) property list.
A	GetItemHt (see page 21177)	Returns the value of the ItemHeight (see page 21186) property.
V	GetItemHt (see page 21216)	Returns the value of the ItemHeight (see page 21186) property.
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
V	GetItemIndex (see page 21177)	Returns the value of the ItemIndex property.
A	GetItemsClass (see page 21177)	Indicates the class that implements the Items (see page 21187) property.
V	GetItemsClass (see page 21217)	Returns the class used to implement the Items (see page 21187) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelLength (see page 21178)	Returns the value of the SelLength (see page 21190) property.
	GetSelStart (see page 21178)	Returns the value of the SelStart (see page 21193) property.
	GetSelText (see page 21217)	Returns the value of the SelText (see page 21222) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
V	KeyDown (see page 8673)	Respond to key press events.
V	KeyPress (see page 8674)	Responds to WM_CHAR messages sent to the combo box.
	KeyUp (see page 6164)	Respond to released key.
	ListWndProc (see page 21178)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
V	Loaded (see page 8674)	Initializes the combo box after its form is loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
V	MeasureItem (see page 21218)	Generates an OnMeasureItem (see page 21225) event.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 8675)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.

	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Select (see page 21179)	Generates an OnSelect (see page 21201) event.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectItem (see page 21218)	Selects a specified item from the drop-down list.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetDropDownCount (see page 21180)	Sets the value of the DropDownCount (see page 21185) property.
	SetDroppedDown (see page 21180)	Sets the value of the DroppedDown (see page 21185) property.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetItemHeight (see page 21181)	Sets the value of the ItemHeight (see page 21186) property.
A	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
V	SetItemIndex (see page 21181)	Sets the value of the ItemIndex property.
V	SetItems (see page 8675)	Sets the value of the Items property.
	SetMaxLength (see page 21182)	Sets the value of the MaxLength (see page 21189) property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelLength (see page 21182)	Sets the value of the SelLength (see page 21190) property.
	SetSelStart (see page 21182)	Sets the value of the SelStart (see page 21193) property.

V	SetStyle (see page 8676)	This is SetStyle, a member of class TDBComboBox.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 8677)	Processes messages received by the combo box.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
R	EditHandle (see page 21185)	Returns the window handle for the edit portion of the combo box.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	ItemCount (see page 21186)	Specifies the number of items in the drop-down list.
R	ListHandle (see page 21189)	Provides access to the window handle of the list of items in the combo box.
	MaxLength (see page 21189)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
A	AddItem (see page 5940)	Adds an item to the list control.
V	AddItem (see page 21170)	Adds an item to the drop-down list of items.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 5940)	Deletes all of the items from the list control.
V	Clear (see page 21172)	Deletes all text from the edit box and all items from the Items (see page 21187) list.
A	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
V	ClearSelection (see page 21172)	Deselects any selected items in the drop-down list.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
A	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
V	CopySelection (see page 21173)	Copies the selected item in the drop-down list to another list control.
V	Create (see page 8669)	Creates an instance of TDBComboBox (see page 8657).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	DeleteSelected (see page 5941)	Deletes all the selected items.
V	DeleteSelected (see page 21174)	Deletes the selected item from the drop-down list.
V	Destroy (see page 8670)	Destroys an instance of TDBComboBox (see page 8657).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.

	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
V	ExecuteAction (see page 8673)	Invokes an action with the TDBComboBox (see page 8657) as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
V	Focused (see page 21175)	Indicates whether the combo box has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.

V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
A	SelectAll (see page 5943)	Selects all items or all text in the selected item.
V	SelectAll (see page 21180)	Selects the entire block of text in the edit box.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
V	UpdateAction (see page 8676)	Updates an action to reflect the current state of the TDBComboBox (see page 8657).
	UpdateControlState (see page 6186)	Respond to state change.
V	UseRightToLeftAlignment (see page 8676)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AutoCloseUp (see page 21220)	Specifies whether the drop-down closes up automatically when the user selects an item.
	AutoCompleteDelay (see page 21220)	Specifies the delay between a key press and an attempt to autocomplete the field.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.

R	Canvas (see page 21184)	Provides access to the drawing surface of the combo box.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DroppedDown (see page 21185)	Indicates whether the drop-down list is currently displayed.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Field (see page 8692)	Identifies the TField object whose current value the combo box represents.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelLength (see page 21190)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 21193)	Specifies the position of the first selected character in the edit portion of the combo box.
	SelText (see page 21222)	Represents the selected text in the edit region.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	Text (see page 8708)	Specifies the value in the edit portion of the combo box.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.

	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 8710)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 8712)	Occurs when the user clicks the control.
	OnContextPopup (see page 8714)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 8715)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 8715)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 8717)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 8720)	Occurs when an item in an owner-draw combo box needs to be displayed.
	OnDropDown (see page 8720)	Occurs when the user opens the drop-down list.
	OnEndDock (see page 8722)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 8722)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 8723)	Occurs when a control receives the input focus.
	OnExit (see page 8726)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 8727)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 8730)	Occurs when a key is pressed.
	OnKeyUp (see page 8731)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 8733)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.
	OnMouseEnter (see page 8733)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 8733)	Occurs when the user moves the mouse outside of a control.
	OnStartDock (see page 8734)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 8734)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 8678)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 8682)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 8682)	Positions to matching list items as you type.
	AutoDropDown (see page 8683)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 8683)	Specifies which edges of the control are beveled.
	BevelInner (see page 8683)	Specifies the cut of the inner bevel.
	BevelKind (see page 8684)	Specifies the control's bevel style.
	BevelOuter (see page 8684)	Specifies the cut of the outer bevel.
	BevelWidth (see page 8685)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 8685)	Specifies the bi-directional mode for the control.
	CharCase (see page 8686)	Determines the case of the text in the combo box.
	Color (see page 8686)	Specifies the background color of the control.
	Constraints (see page 8687)	Specifies the size constraints for the control.
	Ctl3D (see page 8688)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DataField (see page 8688)	Identifies the field from which the combo box displays data.
	DataSource (see page 8689)	Links the combo box to a dataset.

	DragCursor (see page 8689)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 8689)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 8690)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownCount (see page 8691)	Specifies the maximum number of items displayed in the drop-down list.
	Enabled (see page 8692)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 8693)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 8694)	Determines the behavior of the input method editor (IME).
	ImeName (see page 8694)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	ItemHeight (see page 8695)	Specifies the height, in pixels, of the items in the drop-down list.
W	Items (see page 8695)	Contains the strings that appear in the list of the database combo box.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 8696)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 8696)	Determines where a control looks for its color information.
	ParentCtl3D (see page 8697)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 8697)	Determines where a control looks for its font information.
	ParentShowHint (see page 8698)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 8698)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 8703)	Determines if the user can change the value of the field.
	ShowHint (see page 8703)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Sorted (see page 8705)	Determines whether the list portion of the combo box is alphabetized.
	Style (see page 8706)	Determines how the combo box displays its items.
	TabOrder (see page 8707)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 8707)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 8708)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

47.1.3.2 TDBComboBox.Create Constructor

Creates an instance of TDBComboBox (see page 8657).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDBComboBox(TComponent * AOwner);
```

Description

Call Create to instantiate a TDBComboBox (see page 8657) object at runtime. For TDBComboBox (see page 8657) objects placed on forms at design time, Create is called automatically.

AOwner is the component, typically the form, that is responsible for freeing the combo box. It becomes the value of the Owner property.

See Also

Destroy (see page 8670)

Owner (see page 2807)

47.1.3.3 TDBComboBox.Destroy Destructor

Destroys an instance of TDBComboBox (see page 8657).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBComboBox();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TDBComboBox (see page 8657) reference is not nil and only then calls Destroy.

See Also

Create (see page 8669)

Free (see page 22300)

TFieldDataLink (see page 9487)

47.1.3.4 TDBComboBox Methods

The methods of the TDBComboBox class are listed here.

Protected Methods

	Name	Description
	Change (see page 8671)	Generates an OnChange event.
	Click (see page 8671)	Respond to user click.
	ComboWndProc (see page 8671)	Responds to Windows messages sent to the edit control or drop-down list.
	CreateWnd (see page 8672)	Creates the Windows control that represents the combo box.
	DropDown (see page 8672)	Generates an OnDropDown event.
	KeyDown (see page 8673)	Respond to key press events.
	KeyPress (see page 8674)	Responds to WM_CHAR messages sent to the combo box.
	Loaded (see page 8674)	Initializes the combo box after its form is loaded into memory.
	Notification (see page 8675)	Responds to notifications that components are being created or destroyed.
	SetItems (see page 8675)	Sets the value of the Items property.
	SetStyle (see page 8676)	This is SetStyle, a member of class TDBComboBox.
	WndProc (see page 8677)	Processes messages received by the combo box.

Public Methods

	Name	Description
	ExecuteAction (see page 8673)	Invokes an action with the TDBComboBox (see page 8657) as its target.
	UpdateAction (see page 8676)	Updates an action to reflect the current state of the TDBComboBox (see page 8657).
	UseRightToLeftAlignment (see page 8676)	Specifies whether the control's alignment should be reversed right-to-left.

47.1.3.4.1 TDBComboBox.Change Method

Generates an OnChange event.

Pascal

```
procedure Change; override;
```

C++

```
virtual __fastcall Change();
```

Description

Change is called automatically in response to notifications that the text has changed. It responds by generating an OnChange event. Override this method to add additional processing when the text changes, or to block the OnChange event.

See Also

OnChange (see page 21197)

Changed (see page 5742)

47.1.3.4.2 TDBComboBox.Click Method

Respond to user click.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Click is called automatically when the user left-clicks the control. Component or application code can call Click to simulate a user mouse click. This is often done in menu actions and hotkey handlers.

As implemented in TControl, Click queries whether the OnClick event handler exists and is different from the OnExecute handler for the control's Action. If this is true, the OnClick event handler is called. If this is false, and the Action OnExecute event handler exists, the Action's Execute method is called. Override Click to provide additional behavior.

See Also

Action (see page 5797)

OnExecute (see page 2716)

Execute (see page 2712)

DblClick (see page 5747)

OnClick (see page 5851)

47.1.3.4.3 TDBComboBox.ComboWndProc Method

Responds to Windows messages sent to the edit control or drop-down list.

Pascal

```
procedure ComboWndProc(var Message: TMessage; ComboWnd: HWnd; ComboProc: Pointer); override;
```

C++

```
virtual __fastcall ComboWndProc(TMessage Message, HWnd ComboWnd, void * ComboProc);
```

Description

ComboWndProc handles the common processing of Windows messages sent to the edit region or list portion of the combo box. It is called by the EditWndProc and ListWndProc methods to take care of all processing that is not specific to the edit region or drop-down list.

The Message parameter describes the Windows message. The ComboWnd parameter is the windows handle for the edit region or list. The ComboProc parameter is a pointer to the default window procedure for the window that received the message. ComboWndProc provides default responses to focus events, mouse events, and key strokes.

Override ComboWndProc to change the way the combo box responds to Windows messages. When overriding ComboWndProc, be sure to call the parent class's method to process all unhandled messages.

Note: Do not confuse ComboWndProc with the WndProc method. ComboWndProc responds to messages that are sent to the edit region or drop-down list. WndProc responds to messages sent to the combo box itself.

See Also

WndProc (see page 21183)

EditWndProc (see page 21175)

ListWndProc (see page 21178)

47.1.3.4.4 TDBComboBox.CreateWnd Method

Creates the Windows control that represents the combo box.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

The CreateHandle method calls CreateWnd to handle the particulars of creating the combo box window. TCustomCombo overrides the inherited method to ensure that the newly-created window and the TCustomCombo properties correctly reflect each other.

The Windows control for the combo box can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. Thus, CreateWnd may be called more than once during the lifetime of the TCustomCombo instance.

See Also

CreateHandle (see page 6135)

47.1.3.4.5 TDBComboBox.DropDown Method

Generates an OnDropDown event.

Pascal

```
procedure DropDown; override;
```

C++

```
virtual __fastcall DropDown();
```

Description

DropDown is called automatically when the DroppedDown property changes to true. It responds by generating an OnDropDown event. To perform additional processing or block the OnDropDown event, override DropDown.

See Also

OnDropDown (see page 21199)

DroppedDown (see page 21185)

47.1.3.4.6 TDBComboBox.ExecuteAction Method

Invokes an action with the TDBComboBox (see page 8657) as its target.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

ExecuteAction is called automatically when the user invokes an action with the TDBComboBox (see page 8657) as a target. The Action parameter specifies the action that was invoked. ExecuteAction returns trueif the action was successfully dispatched, and false if the TDBComboBox (see page 8657) could not handle the action.

See Also

UpdateAction (see page 8676)

Execute (see page 2712)

HandlesTarget (see page 2713)

47.1.3.4.7 TDBComboBox.KeyDown Method

Respond to key press events.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-down message (WM_KEYDOWN) from Windows, its message handler calls the DoKeyDown method. If DoKeyDown determines that the control should, in fact, process the character, it calls KeyDown, passing the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

See Also

KeyUp (see page 6164)

OnKeyDown (see page 6225)

DoKeyDown (see page 6145)

DoKeyPress (see page 6146)

DoKeyUp (see page 6146)

47.1.3.4.8 TDBComboBox.KeyPress Method

Responds to WM_CHAR messages sent to the combo box.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress processes standard keystroke messages for the combo box control. If the KeyPress method does not handle the keystroke, it calls the inherited method, which generates an OnKeyPress event.

Key is the character represented by the key that was pressed. If the KeyPress method changes Key to 0, no further processing occurs for the keystroke.

See Also

KeyDown (see page 6163)

KeyUp (see page 6164)

OnKeyPress (see page 6227)

47.1.3.4.9 TDBComboBox.Loaded Method

Initializes the combo box after its form is loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The component streaming system calls this method after it loads the combo box's form from a stream.

TCustomCombo overrides the inherited method to select the item specified by the ItemIndex property once the form is fully loaded.

See Also

Loaded (see page 2782)

ItemIndex

47.1.3.4.10 TDBComboBox.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

47.1.3.4.11 TDBComboBox.SetItems Method

Sets the value of the Items property.

Pascal

```
procedure SetItems(const Value: TStrings); override;
```

C++

```
virtual __fastcall SetItems(const TStrings Value);
```

Description

SetItems is the protected write implementation for the Items property. It copies the strings (and possibly their associated objects) from Value to the Items property.

See Also

Items (see page 21187)

Assign (see page 3043)

47.1.3.4.12 TDBComboBox.SetStyle Method

Pascal

```
procedure SetStyle(Value: TComboboxStyle); override;
```

C++

```
virtual __fastcall SetStyle(TComboboxStyle Value);
```

Description

This is SetStyle, a member of class TDBComboBox.

47.1.3.4.13 TDBComboBox.UpdateAction Method

Updates an action to reflect the current state of the TDBComboBox (see page 8657).

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

UpdateAction is called automatically when the application is idle to give the TDBComboBox (see page 8657) an opportunity to update associated actions. This allows actions to be grayed, checked, or to otherwise reflect the status of the TDBComboBox (see page 8657) they target.

The Action parameter specifies the action component that should be updated. UpdateAction returns trueif the action component now reflects the state of the component, and false if it did not know how to update the action.

See Also

ExecuteAction (see page 8673)

UpdateTarget (see page 2715)

HandlesTarget (see page 2713)

47.1.3.4.14 TDBComboBox.UseRightToLeftAlignment Method

Specifies whether the control's alignment should be reversed right-to-left.

Pascal

```
function UseRightToLeftAlignment: Boolean; override;
```

C++

```
virtual __fastcall Boolean UseRightToLeftAlignment();
```

Description

Controls call UseRightToLeftAlignment when implementing the BiDiMode property. UseRightToLeftAlignment indicates whether the control's alignment should be reversed right-to-left. This information can be used when painting the control or when responding to mouse events.

UseRightToLeftAlignment returns true if the application is running on a system with a Middle Eastern locale and the BiDiMode property is bdRightToLeft. Otherwise, it returns false. Descendants override UseRightToLeftAlignment if alignment in Middle Eastern locales depends on other factors.

See Also

BiDiMode (see page 5803)

GetControlsAlignment (see page 5759)

Text (see page 5842)

Caption (see page 5806)

47.1.3.4.15 TDBComboBox.WndProc Method

Processes messages received by the combo box.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc is the first method to received Windows messages that are sent to the combo box. Messages sent to the child windows of the combo box (for the edit region and the list portion) go to the EditWndProc or ListWndProc method instead. WndProc handles messages about changes to the system colors defined in the Windows control panel, and closes up the drop-down list when the Enter key or the Escape key is pressed. All other messages are passed to Dispatch for default processing.

See Also

ComboWndProc (see page 21173)

Dispatch

WndProc (see page 6187)

EditWndProc (see page 21175)

ListWndProc (see page 21178)

47.1.3.5 TDBComboBox Properties

The properties of the TDBComboBox class are listed here.

Public Properties

	Name	Description
R	Field (see page 8692)	Identifies the TField object whose current value the combo box represents.
	Text (see page 8708)	Specifies the value in the edit portion of the combo box.

Published Properties

	Name	Description
	Align (see page 8678)	Determines how the control aligns within its container (parent control).
	Anchors (see page 8682)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 8682)	Positions to matching list items as you type.
	AutoDropDown (see page 8683)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 8683)	Specifies which edges of the control are beveled.
	BevelInner (see page 8683)	Specifies the cut of the inner bevel.
	BevelKind (see page 8684)	Specifies the control's bevel style.

	BevelOuter (see page 8684)	Specifies the cut of the outer bevel.
	BevelWidth (see page 8685)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 8685)	Specifies the bi-directional mode for the control.
	CharCase (see page 8686)	Determines the case of the text in the combo box.
	Color (see page 8686)	Specifies the background color of the control.
	Constraints (see page 8687)	Specifies the size constraints for the control.
	Ctl3D (see page 8688)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DataField (see page 8688)	Identifies the field from which the combo box displays data.
	DataSource (see page 8689)	Links the combo box to a dataset.
	DragCursor (see page 8689)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 8689)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 8690)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownCount (see page 8691)	Specifies the maximum number of items displayed in the drop-down list.
	Enabled (see page 8692)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 8693)	Controls the attributes of text written on or in the control.
	ImeMode (see page 8694)	Determines the behavior of the input method editor (IME).
	ImeName (see page 8694)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	ItemHeight (see page 8695)	Specifies the height, in pixels, of the items in the drop-down list.
W	Items (see page 8695)	Contains the strings that appear in the list of the database combo box.
	ParentBiDiMode (see page 8696)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 8696)	Determines where a control looks for its color information.
	ParentCtl3D (see page 8697)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 8697)	Determines where a control looks for its font information.
	ParentShowHint (see page 8698)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 8698)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 8703)	Determines if the user can change the value of the field.
	ShowHint (see page 8703)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Sorted (see page 8705)	Determines whether the list portion of the combo box is alphabetized.
	Style (see page 8706)	Determines how the combo box displays its items.
	TabOrder (see page 8707)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 8707)	Determines if the user can tab to a control.
	Visible (see page 8708)	Determines whether the component appears onscreen.

47.1.3.5.1 TDBComboBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
```

```
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
```

```
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
```

```
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

47.1.3.5.2 TDBComboBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

47.1.3.5.3 TDBComboBox.AutoComplete Property

Positions to matching list items as you type.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

AutoComplete will position to matching list values as you type, saving you keystrokes.

As you type, the ItemIndex property is updated with the list entry that matches the characters you have entered. As you type,

different list entries may match, and ItemIndex changes accordingly.

There is a delay of AutoCompleteDelay milliseconds between the characters typed and the autocomplete operation.

See Also

AutoCompleteDelay (see page 21220)

47.1.3.5.4 TDBComboBox.AutoDropDown Property

Specifies whether the drop-down list drops down automatically in response to user keystrokes.

Pascal

```
property AutoDropDown: Boolean;
```

C++

```
__property Boolean AutoDropDown;
```

Description

When AutoDropDown is true, the combo box automatically drops down its list when the user starts typing a string while the combo box has focus.

When AutoDropDown is false, the user must explicitly use the drop-down button to drop down the combo box list.

See Also

autocomplete (see page 21220)

OnDropDown (see page 21199)

DroppedDown (see page 21185)

47.1.3.5.5 TDBComboBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

47.1.3.5.6 TDBComboBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

47.1.3.5.7 TDBComboBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

47.1.3.5.8 TDBComboBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

47.1.3.5.9 TDBComboBox.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

47.1.3.5.10 TDBComboBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

47.1.3.5.11 TDBComboBox.CharCase Property

Determines the case of the text in the combo box.

Pascal

```
property CharCase: TEditCharCase;
```

C++

```
__property TEditCharCase CharCase;
```

Description

Use CharCase to force the contents of the combo box into upper or lower case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types into the combo box. Changing the CharCase property changes the actual text, not just its appearance; case information is lost and cannot be recaptured by changing CharCase back to ecNormal.

See Also

Text (see page 5842)

47.1.3.5.12 TDBComboBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

47.1.3.5.13 TDBComboBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

47.1.3.5.14 TDBComboBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

47.1.3.5.15 TDBComboBox.DataField Property

Identifies the field from which the combo box displays data.

Pascal

```
property DataField: WideString;
```

C++

```
__property BSTR DataField;
```

Description

Set DataField to specify which field is represented by the combo box. (Note that this is does not refer to the source of the list, but to the field into which the selected value is inserted. The list values must be supplied manually using the Items (see page 8695) property.) The combo box displays the current value of this field, and allows the user to set the value of this field on the current record. The dataset the field is located in is specified by the DataSource (see page 8689) property.

```
DBComboBox1.DataField := 'CountryName';
DBComboBox1->DataField = "CountryName";
```

Tip: For a data-aware combo box component, where the list values are automatically supplied, use a TDBLookupComboBox (see page 8963) instead.

See Also

DataSource (see page 8689)

Field (see page 8692)

Items (see page 8695)

TDBLookupComboBox (see page 8963)

Using TDBListBox and TDBComboBox

47.1.3.5.16 TDBComboBox.DataSource Property

Links the combo box to a dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Specify the data source component that identifies the dataset in which the target field is found. To allow the combo box to represent the data for a field, both the DataSource and the DataField (see page 8688) properties must be set.

```
DBComboBox1.DataSource := DataSource1;
DBComboBox1->DataSource = DataSource1;
```

See Also

DataField (see page 8688)

Items (see page 8695)

47.1.3.5.17 TDBComboBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

47.1.3.5.18 TDBComboBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

47.1.3.5.19 TDBComboBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
```

```
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

47.1.3.5.20 TDBComboBox.DropDownCount Property

Specifies the maximum number of items displayed in the drop-down list.

Pascal

```
property DropDownCount: Integer;
```

C++

```
__property int DropDownCount;
```

Description

By default, the drop-down list is long enough to contain eight items without requiring the user to scroll to see them all. To make the drop-down list smaller or larger, specify a number larger or smaller than eight as the DropDownCount value.

If the DropDownCount value is larger than the number of items in the Items property, the drop-down list will be just large enough to hold all the possible choices and no more. If the DropDownCount value is smaller then the number of item in the Items property, the drop down list will display a scroll bar.

See Also

Items (see page 21187)

ItemCount (see page 21186)

47.1.3.5.21 TDBComboBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

47.1.3.5.22 TDBComboBox.Field Property

Identifies the TField object whose current value the combo box represents.

Pascal

```
property Field: TField;
```

C++

```
__property TField Field;
```

Description

Use the TField reference provided by the Field property when you want to change the value of the data in the field programmatically.

```
DBComboBox1.Field.AsString := 'Brazil';
DBComboBox1->Field->AsString = "Brazil";
```

See Also

DataField (see page 8688)

47.1.3.5.23 TDBComboBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

47.1.3.5.24 TDBComboBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

47.1.3.5.25 TDBComboBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

47.1.3.5.26 TDBComboBox.ItemHeight Property

Specifies the height, in pixels, of the items in the drop-down list.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Read ItemHeight to determine the height of the items in the drop-down list.

The effect of setting ItemHeight varies, depending on the type of combo box and the style. For example, TComboBox lets you set the item height only when Style is set to csOwnerDrawFixed or csOwnerDrawVariable. If Style is csOwnerDrawVariable, ItemHeight is the default height for drawing list items, but this value can be overridden by an OnMeasureItem event handler.

TComboBoxEx does not let you change the value of ItemHeight.

Note: Combo boxes that do not support changes to ItemHeight do not raise an exception when you attempt to change ItemHeight; they simply ignore the attempt.

See Also

Items (see page 21187)

OnMeasureItem (see page 21225)

Style (see page 21224)

47.1.3.5.27 TDBComboBox.Items Property

Contains the strings that appear in the list of the database combo box.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to supply the values in the list from which the user can choose. Because Items is an object of type TStrings, you can add, delete, insert, and move items using the Add, Delete, Insert, Exchange, and Move methods of the TStrings object. For example, to add each string to the combo box list, write code such as the following:

while not CountryTbl.Eof do begin

```
DBComboBox1.Items.Add(CountryTbl.FieldByName('Country').AsString);
CountryTbl.Next;
```

end;

while (!CountryTbl->Eof)

```
{
  DBComboBox1->Items->Add(CountryTbl->FieldByName("Country")->AsString);
  CountryTbl->Next();
}
```

The ItemIndex property determines which item is selected, if any.

See Also

ItemIndex

47.1.3.5.28 TDBComboBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

47.1.3.5.29 TDBComboBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the

form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

47.1.3.5.30 TDBComboBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

47.1.3.5.31 TDBComboBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

47.1.3.5.32 TDBComboBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

47.1.3.5.33 TDBComboBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
```

```
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
```

```
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
```

```
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
```

```
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
```

```
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

47.1.3.5.34 TDBComboBox.ReadOnly Property

Determines if the user can change the value of the field.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set ReadOnly to specify whether the combo box should be used for display purposes only. If ReadOnly is true, the combo box can only be used to display the value of the field on the current record. If ReadOnly is false, the user can use the combo box to edit the field's value.

See Also

Field (see page 8692)

Edit (see page 6932)

47.1.3.5.35 TDBComboBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
```

```
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

47.1.3.5.36 TDBComboBox.Sorted Property

Determines whether the list portion of the combo box is alphabetized.

Pascal

```
property Sorted: Boolean;
```

C++

```
__property Boolean Sorted;
```

Description

Set Sorted to true to sort the items in the Items list alphabetically. New items added to the list while Sorted is true are inserted in the correct alphabetical position.

When Sorted is changed from false to true, the original order of the items is lost. Setting Sorted back to false does not restore the original order.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if (((TForm1 *)Screen->CustomForms[i])->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
```

```
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
    end;
    Items.EndUpdate; {reenable painting }
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.Show;
end;
```

See Also

Items

Using TDBListBox and TDBComboBox

47.1.3.5.37 TDBComboBox.Style Property

Determines how the combo box displays its items.

Pascal

```
property Style: TComboBoxStyle;
```

C++

```
__property TComboBoxStyle Style;
```

Description

Use Style to specify whether the user can type values into the edit region as well as select values from the list, whether the list drops down when the user clicks a button or if it is always displayed, whether the list is owner-drawn, and so on. By default, Style is csDropDown, meaning that the combo box displays each item as a string in a drop-down list, and that users can edit the value in the edit region. Style can be any of the values: csDropDown, csSimple, csDropDownList, csOwnerDrawFixed, and csOwnerDrawVariable.

Owner-draw combo boxes can display items other than strings. For example, a combo box could display graphical images along with or instead of its strings. Owner-draw combo boxes require more programming, however, as the application must render the image for each item in the list.

When Style is csOwnerDrawFixed or csOwnerDrawVariable, the OnDrawItem (see page 8720) event occurs when the list items need to be drawn. If Style is csOwnerDrawVariable, an OnMeasureItem (see page 8733) event occurs as well.

See Also

Height (see page 5825)

ItemHeight

Items (see page 8695)

OnDrawItem (see page 21224)

OnMeasureItem (see page 21225)

Using TDBListBox and TDBComboBox

47.1.3.5.38 TDBComboBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

47.1.3.5.39 TDBComboBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

47.1.3.5.40 TDBComboBox.Text Property

Specifies the value in the edit portion of the combo box.

Pascal

```
property Text: TCaption;
```

C++

```
__property TCaption Text;
```

Description

Use Text to supply read a field value from the combo box or assign a value to the field that the combo box represents. If the combo box has the style csDropDownList, then the value of Text is the same as the item in the drop-down list identified by the ItemIndex property. For other styles of combo boxes, the value of Text can be a value that is not included in the drop-down list.

See Also

ItemIndex

Style (see page 8706)

47.1.3.5.41 TDBComboBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
```

```
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

47.1.3.6 TDBComboBox Events

The events of the TDBComboBox class are listed here.

Published Events

	Name	Description
	OnChange (see page 8710)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 8712)	Occurs when the user clicks the control.
	OnContextPopup (see page 8714)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 8715)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 8715)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 8717)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 8720)	Occurs when an item in an owner-draw combo box needs to be displayed.
	OnDropDown (see page 8720)	Occurs when the user opens the drop-down list.
	OnEndDock (see page 8722)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 8722)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 8723)	Occurs when a control receives the input focus.
	OnExit (see page 8726)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 8727)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 8730)	Occurs when a key is pressed.
	OnKeyUp (see page 8731)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 8733)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.
	OnMouseEnter (see page 8733)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 8733)	Occurs when the user moves the mouse outside of a control.
	OnStartDock (see page 8734)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 8734)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

47.1.3.6.1 TDBComboBox.OnChange Event

Occurs when the user changes the text displayed in the edit region.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action immediately after the user edits the text in the edit region or selects an item from the list. The Text property gives the new value in the edit region.

Note: OnChange only occurs in response to user actions. Changing the Text property programmatically does not trigger an OnChange event.

C++ Examples:

```
/*
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
*/
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  if (pCB)
    PageControl1->ActivePage = (TTabSheet *)(pCB->Items->Objects[pCB->ItemIndex]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
    ComboBox1->Items->AddObject(PageControl1->Pages[i]->Name, PageControl1->Pages[i]);
    ComboBox1->ItemIndex = 0;
  }
}
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
```

```
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to PageControl1.PageCount - 1 do
    ComboBox1.Items.AddObject(PageControl1.Pages[i].Name,
      PageControl1.Pages[i]);
  ComboBox1.ItemIndex := 0;
end;

procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  if (Sender is TComboBox) then
    with (Sender as TComboBox) do
      PageControl1.ActivePage := TTabSheet(Items.Objects[ItemIndex]);
end;
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

See Also

Text (see page 5842)

OnCloseUp (see page 21199)

47.1.3.6.2 TDBComboBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that

action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
```

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

47.1.3.6.3 TDBComboBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property

of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

47.1.3.6.4 TDBComboBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

47.1.3.6.5 TDBComboBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
```

```
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
```

```
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

47.1.3.6.6 TDBComboBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
```

```
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.3.6.7 TDBComboBox.OnDrawItem Event

Occurs when an item in an owner-draw combo box needs to be displayed.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

Write an OnDrawItem event handler to draw the items in the drop-down list of an owner-draw combo box. An OnDrawItem event handler can add graphic elements to the list items, or replace the list item text by graphics.

Draw the items on the Canvas using the coordinates provided by the Rect parameter. OnDrawItem occurs only if Style is set to csOwnerDrawFixed or csOwnerDrawVariable.

If an OnDrawItem event handler is not provided, the combo box fills the Rect parameter with the current brush and writes the text value of the item specified by the Index parameter.

See Also

Canvas

Items

OnMeasureItem (see page 21225)

Style (see page 21224)

47.1.3.6.8 TDBComboBox.OnDropDown Event

Occurs when the user opens the drop-down list.

Pascal

```
property OnDropDown: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDropDown;
```

Description

Write an OnDropDown event handler to implement special processing that needs to occur only when the drop-down list is activated.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if (((TForm1 *)Screen->CustomForms[i])->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
    end;
    Items.EndUpdate; {reenable painting }
  end;
```

```
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.Show;
end;
```

See Also

DroppedDown (see page 21185)

OnCloseUp (see page 21199)

47.1.3.6.9 TDBComboBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.3.6.10 TDBComboBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
```

```
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

47.1.3.6.11 TDBComboBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
```

```
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
```

```
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

47.1.3.6.12 TDBComboBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
```

```
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

47.1.3.6.13 TDBComboBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
```

```
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
```

```
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

47.1.3.6.14 TDBComboBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not

generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

47.1.3.6.15 TDBComboBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
```

```
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
```

```
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

47.1.3.6.16 TDBComboBox.OnMeasureItem Event

Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.

Pascal

```
property OnMeasureItem: TMeasureItemEvent;
```

C++

```
__property TMeasureItemEvent OnMeasureItem;
```

Description

When Style is set to csOwnerDrawVariable, the OnMeasureItem event precedes OnDrawItem. Write an OnMeasureItem event handler to specify the height, in pixels, needed to draw an item in the drop-down list.

See Also

ItemHeight

Items

OnDrawItem (see page 21224)

Style (see page 21224)

47.1.3.6.17 TDBComboBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

47.1.3.6.18 TDBComboBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

47.1.3.6.19 TDBComboBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

47.1.3.6.20 TDBComboBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

47.1.4 TDBEdit Class

TDBEdit represents a single-line edit control that can display and edit a field in a dataset.

Class Hierarchy

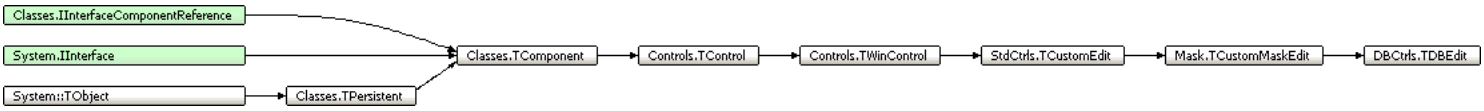

Pascal

```
TDBEdit = class(TCustomMaskEdit);
```

C++

```
class TDBEdit : public TCustomMaskEdit;
```

File

DBCtrls

Description

Use TDBEdit to enable users to edit a database field. TDBEdit uses the Text property to represent the contents of the field.

TDBEdit permits only a single line of text. If the field may contain lengthy data that would require multiple lines, consider using a TDBMemo (see page 9149) object.

If the application does not require the data-aware capabilities of TDBEdit, use an edit control (TEdit) or a masked edit control (TMaskEdit) instead, to conserve system resources.

To provide a mask that restricts input and controls the display format of the data, use mask-related properties of TField and descendants. Such properties include: EditMask (TField), DisplayFormat (TDateTimeField), and DisplayFormat (TNumericField). Which property you should use depends on the field's type and the TField descendant that corresponds to that type.

See Also

TDataSet (see page 6894)

TDataSource (see page 7070)

TDBMemo (see page 9149)

TEdit (see page 21402)

TField (see page 7115)

TMaskEdit (see page 15824)

Displaying and Editing Fields in an Edit Box

Displaying a Single Record

Using Common Data Control Features

Navigating Datasets

Displaying

Using Object Fields

Working with Field Components: Overview

47.1.4.1 TDBEdit Members

The following tables list the members exposed by TDBEdit.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.

	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
V	Change (see page 8749)	Responds when the text is modified.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CheckCursor (see page 15811)	Moves the cursor to the closest editable character and selects it if nothing is currently selected.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 21248)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWindowHandle (see page 21248)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CreateWnd (see page 21248)	Creates a Windows control corresponding to the edit control.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.

	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21251)	Destroys the Windows control that corresponds to the edit control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoSetMaxLength (see page 21251)	Sets the MaxLength (see page 21263) property.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EditCanModify (see page 8749)	Allows or disallows editing of the text depending on whether the dataset is in edit mode.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.

	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFirstEditChar (see page 15813)	Returns the index in the EditMask (see page 15821) of the first nonliteral character.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetLastEditChar (see page 15813)	Returns the index in the EditMask (see page 15821) of the last nonliteral character.
	GetMaxChars (see page 15814)	Returns the length of the EditText (see page 15821).
	GetNextEditChar (see page 15814)	Returns the first index of a nonliteral character equal to or greater than an offset.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetPriorEditChar (see page 15815)	Returns the first index of a nonliteral character equal to or less than an offset.
	GetSel (see page 15815)	Returns the endpoints of the current selection in the EditText (see page 15821).
	GetSelLength (see page 21252)	Returns the SelLength (see page 21265) property.
	GetSelStart (see page 21252)	Returns the SelStart (see page 21269) property.
	GetSelText (see page 21252)	Returns the SelText (see page 21272) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 8750)	Supplements the inherited method to drive the dataset into edit mode when the user presses Del or Shift+Insert.
	KeyPress (see page 8751)	Supplements the inherited method to handle any keystrokes that alter the state of the edit control.
	KeyUp (see page 6164)	Respond to released key.
	KeyUp (see page 15817)	Key up event dispatcher.
	Loaded (see page 8751)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..

	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 8752)	Overrides the inherited method to respond to opRemove notifications on the DataSource (see page 8765).
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReformatText (see page 15817)	Applies the mask NewMask to the underlying text, and updates the internal representation of the mask.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
V	Reset (see page 8752)	Cancels all unposted edits.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetAutoSize (see page 21257)	Sets the AutoSize (see page 21260) property.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetCursor (see page 15818)	Moves the cursor to the first editable character at or before Pos, and selects that character.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.

	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSel (see page 15819)	Selects character range.
	SetSelLength (see page 21257)	Sets the SelLength (see page 21265) property.
	SetSelStart (see page 21257)	Sets the SelStart (see page 21269) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	Validate (see page 15819)	Validates a specified string against the current mask.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateError (see page 15820)	Beeps and raises an EDBEditError (see page 15798) exception.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	EditMask (see page 15821)	Specifies the mask that represents what text is valid for the masked edit control.
	HideSelection (see page 21262)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MaskState (see page 15822)	Indicates state information about how the mask is being applied.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	OEMConvert (see page 21264)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.

	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 15812)	Sets the Text (see page 15823) property to an empty string.
V	Clear (see page 21245)	Deletes all text from the edit control.
	ClearSelection (see page 21245)	Removes the selected text from the edit control.
	ClearUndo (see page 21246)	Clears the undo buffer so that no changes to the text can be backed out.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopyToClipboard (see page 21246)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
V	Create (see page 8747)	Creates an instance of TDBEdit (see page 8735).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CutToClipboard (see page 21249)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 21250)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 8748)	Destroys an instance of TDBEdit (see page 8735).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.

	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
V	ExecuteAction (see page 8750)	Invokes an action with the TDBEdit (see page 8735) as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
V	GetSelTextBuf (see page 21253)	Copies the selected text into a buffer and returns the number of characters copied.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	GetTextLen (see page 15816)	Returns the length of the underlying text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.

	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	PasteFromClipboard (see page 21254)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectAll (see page 21255)	Selects all text in the edit control.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetDesignVisible (see page 6178)	Makes a control visible at design time.
	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSelTextBuf (see page 21258)	Replaces the selected text with a null-terminated string.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Undo (see page 21259)	Backs out all changes in the undo buffer.
	Update (see page 5789)	Processes any pending paint messages immediately.
	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 8752)	Updates an action to reflect the current state of the TDBEdit (see page 8735).
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 8753)	Specifies whether the control's alignment is in a right-to-left mode.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.
	ValidateEdit (see page 15819)	Validates the EditText (see page 15821) against the current mask.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.

R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	CanUndo (see page 21262)	Indicates whether the edit control contains changes that can be backed out.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EditText (see page 15821)	Specifies the text that appears in the window of the masked edit control.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Field (see page 8769)	Specifies the TField object for the database field the edit box represents.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
R	IsMasked (see page 15822)	Indicates whether a mask exists for the edit control.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Modified (see page 21263)	Indicates whether the user edited the text of the edit control.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelLength (see page 21265)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 21269)	Specifies the position of the first selected character in the text.
	SelText (see page 21272)	Specifies the selected portion of the edit control's text.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.

	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	Text (see page 15823)	Denotes the underlying text for the masked edit control before the mask has been applied.
	Text (see page 21273)	Contains a text string associated with the control.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 8784)	Occurs when the text for the edit control may have changed.
	OnClick (see page 8784)	Occurs when the user clicks the control.
	OnContextPopup (see page 8786)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 8787)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 8787)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 8790)	Occurs when the user drags an object over a control.
	OnEndDock (see page 8792)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 8792)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 8793)	Occurs when a control receives the input focus.
	OnExit (see page 8796)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 8797)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 8800)	Occurs when a key is pressed.
	OnKeyUp (see page 8801)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 8803)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 8804)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 8806)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 8807)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 8807)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 8809)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 8811)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 8812)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 8754)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 8758)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 8758)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 8759)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 8760)	Specifies which edges of the control are beveled.
	BevelInner (see page 8760)	Specifies the cut of the inner bevel.
	BevelKind (see page 8761)	Specifies the control's bevel style.
	BevelOuter (see page 8761)	Specifies the cut of the outer bevel.
	BevelWidth (see page 8761)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 8762)	Specifies the bi-directional mode for the control.

	BorderStyle (see page 8762)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 8763)	Determines the case of the text within the edit control.
	Color (see page 8763)	Specifies the background color of the control.
	Constraints (see page 8764)	Specifies the size constraints for the control.
	Ctl3D (see page 8764)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DataField (see page 8765)	Specifies the field from which the edit control displays data.
	DataSource (see page 8765)	Links the edit control to the dataset that contains the field it represents.
	DragCursor (see page 8766)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 8766)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 8767)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 8768)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 8769)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 8770)	Determines the behavior of the input method editor (IME).
	ImeName (see page 8771)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	MaxLength (see page 8771)	Specifies the maximum number of characters that can appear in the edit control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 8772)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 8772)	Determines where a control looks for its color information.
	ParentCtl3D (see page 8773)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 8773)	Determines where a control looks for its font information.
	ParentShowHint (see page 8774)	Determines where a control looks to find out if its Help Hint should be shown.
	PasswordChar (see page 8774)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 8774)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 8779)	Determines whether the user can use the database edit box to change the value of the field in the current record.
	ShowHint (see page 8779)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 8781)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 8782)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 8782)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

47.1.4.2 TDBEdit.Create Constructor

Creates an instance of TDBEdit (see page 8735).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDBEdit(TComponent * AOwner);
```

Description

Call Create to create a TDBEdit (see page 8735) object at runtime. TDBEdit (see page 8735) objects placed on a form at design time are created automatically.

Create overrides the inherited constructor to initialize the database handling capabilities of the edit control.

AOwner is the component, typically the form, that is responsible for freeing the TDBEdit (see page 8735) instance. It becomes the value of the Owner property.

See Also

Create (see page 15810)

Owner (see page 2807)

47.1.4.3 TDBEdit.Destroy Destructor

Destroys an instance of TDBEdit (see page 8735).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBEdit();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TDBEdit (see page 8735) reference is not nil and only then calls Destroy.

Destroy overrides the inherited destructor to free the helper objects that were allocated to manage display issues and database interactions.

See Also

Create (see page 8747)

Free (see page 22300)

47.1.4.4 TDBEdit Methods

The methods of the TDBEdit class are listed here.

Protected Methods

	Name	Description
	Change (see page 8749)	Responds when the text is modified.
	EditCanModify (see page 8749)	Allows or disallows editing of the text depending on whether the dataset is in edit mode.
	KeyDown (see page 8750)	Supplements the inherited method to drive the dataset into edit mode when the user presses Del or Shift+Insert.

	KeyPress (see page 8751)	Supplements the inherited method to handle any keystrokes that alter the state of the edit control.
	Loaded (see page 8751)	Initializes the control after it is loaded from a stream.
	Notification (see page 8752)	Overrides the inherited method to respond to opRemove notifications on the DataSource (see page 8765).
	Reset (see page 8752)	Cancels all unposted edits.

Public Methods

	Name	Description
	ExecuteAction (see page 8750)	Invokes an action with the TDBEdit (see page 8735) as its target.
	UpdateAction (see page 8752)	Updates an action to reflect the current state of the TDBEdit (see page 8735).
	UseRightToLeftAlignment (see page 8753)	Specifies whether the control's alignment is in a right-to-left mode.

47.1.4.4.1 TDBEdit.Change Method

Responds when the text is modified.

Pascal

```
procedure Change; override;
```

C++

```
virtual __fastcall Change();
```

Description

Applications cannot call this protected method directly: It is triggered automatically by Windows' events.

Change responds to changes in the text by internally keeping track of the fact that the text has been modified. This information can then be used to know whether there are any changes to post to the dataset or to remove if the user requests a reset.

Additionally, Change calls the inherited method, which generates an OnChange (see page 8784) event.

See Also

OnChange (see page 21275)

Reset (see page 8752)

Text (see page 5842)

47.1.4.4.2 TDBEdit.EditCanModify Method

Allows or disallows editing of the text depending on whether the dataset is in edit mode.

Pascal

```
function EditCanModify: Boolean; override;
```

C++

```
virtual __fastcall Boolean EditCanModify();
```

Description

EditCanModify overrides the inherited method so that the control will only allow edits when the dataset is in edit mode. If the edit control should never allow edits, the ReadOnly (see page 8779) property should be set.

Note: EditCanModify is a protected method. Applications cannot call it directly.

See Also

DataSource (see page 8765)

ReadOnly (see page 8779)

Text (see page 5842)

47.1.4.4.3 TDBEdit.ExecuteAction Method

Invokes an action with the TDBEdit (see page 8735) as its target.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

ExecuteAction is called automatically when the user invokes an action with the TDBEdit (see page 8735) as a target. The Action parameter specifies the action that was invoked. ExecuteAction returns true if the action was successfully dispatched, and false if the TDBEdit (see page 8735) could not handle the action.

See Also

UpdateAction (see page 8752)

Execute (see page 2712)

HandlesTarget (see page 2713)

47.1.4.4.4 TDBEdit.KeyDown Method

Supplements the inherited method to drive the dataset into edit mode when the user presses Del or Shift+Insert.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

KeyDown first calls the inherited method, which calls the OnKeyDown (see page 8797) event handler. KeyDown next attempts to put the dataset into an editing state when the user types Del or Shift+Insert. Derived classes can override KeyDown to handle additional key combinations, but the inherited method should always be called.

Note: KeyDown is a protected method. Applications cannot call it directly. It is called automatically in response to user actions.

See Also

DataSource (see page 8765)

EditCanModify (see page 8749)

KeyPress (see page 8751)

OnKeyDown (see page 6225)

KeyDown (see page 15816)

47.1.4.4.5 TDBEdit.KeyPress Method

Supplements the inherited method to handle any keystrokes that alter the state of the edit control.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress first calls the inherited method, which calls the OnKeyPress (see page 8800) event handler. Next, KeyPress discards the keystroke if it is not a valid character for the field type, or if there is no link to a database field. Valid characters cause KeyPress to attempt to put the dataset into edit mode.

Pressing the Escape key causes KeyPress to call Reset (see page 8752), throwing away all unposted edits.

Note: KeyPress is a protected method. Applications cannot call it directly. It is called automatically in response to user actions.

See Also

DataSource (see page 8765)

EditCanModify (see page 8749)

KeyDown (see page 8750)

OnKeyPress (see page 6227)

Reset (see page 8752)

KeyPress (see page 15816)

47.1.4.4.6 TDBEdit.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded (see page 2782)

Action (see page 5797)

ActionChange (see page 5737)

47.1.4.4.7 TDBEdit.Notification Method

Overrides the inherited method to respond to opRemove notifications on the DataSource (see page 8765).

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

After calling the inherited method, Notification checks whether the notification indicates the removal of the DataSource (see page 8765). If so, Notification sets the DataSource (see page 8765) property to nil (Delphi) or NULL (C++).

Derived classes can override Notification method to handle additional notifications, but the inherited method should be always be called.

Note: Notification is a protected method. Applications can't call it directly.

See Also

DataSource (see page 8765)

47.1.4.4.8 TDBEdit.Reset Method

Cancels all unposted edits.

Pascal

```
procedure Reset; override;
```

C++

```
virtual __fastcall Reset();
```

Description

Reset overrides the default method to cancel all unposted edits to the dataset when the user presses the Escape key.

Note: Reset is a protected method. Applications cannot call it directly. It is called automatically when the user presses the Escape key.

See Also

Change (see page 8749)

KeyPress (see page 8751)

47.1.4.4.9 TDBEdit.UpdateAction Method

Updates an action to reflect the current state of the TDBEdit (see page 8735).

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

UpdateAction is called automatically when the application is idle to give the TDBEdit (see page 8735) an opportunity to update associated actions. This allows actions to be grayed, checked, or to otherwise reflect the status of the TDBEdit (see page 8735) they target.

The Action parameter specifies the action component that should be updated. UpdateAction returns trueif the action component now reflects the state of the component, and false if it did not know how to update the action.

See Also

ExecuteAction (see page 8750)

UpdateTarget (see page 2715)

HandlesTarget (see page 2713)

47.1.4.4.10 TDBEdit.UseRightToLeftAlignment Method

Specifies whether the control's alignment is in a right-to-left mode.

Pascal

```
function UseRightToLeftAlignment: Boolean; override;
```

C++

```
virtual __fastcall Boolean UseRightToLeftAlignment();
```

Description

Call UseRightToLeftAlignment to determine whether the TDBEdit (see page 8735)'s alignment is in a right-to-left mode. This information can be used when painting the TDBEdit (see page 8735) data or when responding to mouse events.

UseRightToLeftAlignment returns true for middle east locales if the BiDiMode (see page 8762) property is bdRightToLeft. Otherwise, it returns false.

See Also

UseRightToLeftReading (see page 5790)

BiDiMode (see page 5803)

47.1.4.5 TDBEdit Properties

The properties of the TDBEdit class are listed here.

Public Properties

	Name	Description
R	Field (see page 8769)	Specifies the TField object for the database field the edit box represents.

Published Properties

	Name	Description
	Align (see page 8754)	Determines how the control aligns within its container (parent control).
	Anchors (see page 8758)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 8758)	Determines whether all the text in the edit control is automatically selected when the control gets focus.

	AutoSize (see page 8759)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 8760)	Specifies which edges of the control are beveled.
	BevelInner (see page 8760)	Specifies the cut of the inner bevel.
	BevelKind (see page 8761)	Specifies the control's bevel style.
	BevelOuter (see page 8761)	Specifies the cut of the outer bevel.
	BevelWidth (see page 8761)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 8762)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 8762)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 8763)	Determines the case of the text within the edit control.
	Color (see page 8763)	Specifies the background color of the control.
	Constraints (see page 8764)	Specifies the size constraints for the control.
	Ctl3D (see page 8764)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DataField (see page 8765)	Specifies the field from which the edit control displays data.
	DataSource (see page 8765)	Links the edit control to the dataset that contains the field it represents.
	DragCursor (see page 8766)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 8766)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 8767)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 8768)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 8769)	Controls the attributes of text written on or in the control.
	ImeMode (see page 8770)	Determines the behavior of the input method editor (IME).
	ImeName (see page 8771)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	MaxLength (see page 8771)	Specifies the maximum number of characters that can appear in the edit control.
	ParentBiDiMode (see page 8772)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 8772)	Determines where a control looks for its color information.
	ParentCtl3D (see page 8773)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 8773)	Determines where a control looks for its font information.
	ParentShowHint (see page 8774)	Determines where a control looks to find out if its Help Hint should be shown.
	PasswordChar (see page 8774)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 8774)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 8779)	Determines whether the user can use the database edit box to change the value of the field in the current record.
	ShowHint (see page 8779)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 8781)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 8782)	Determines if the user can tab to a control.
	Visible (see page 8782)	Determines whether the component appears onscreen.

47.1.4.5.1 TDBEdit.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it

continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
```

```
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
```

```
    Dirs.Align := alLeft;
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
```

```
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

47.1.4.5.2 TDBEdit.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

47.1.4.5.3 TDBEdit.AutoSelect Property

Determines whether all the text in the edit control is automatically selected when the control gets focus.

Pascal

```
property AutoSelect: Boolean;
```

C++

```
__property Boolean AutoSelect;
```

Description

Set AutoSelect to select all the text when the edit control gets focus. AutoSelect only applies to single-line edit controls.

Use AutoSelect when the user is more likely to replace the text in the edit control than to append to it.

See Also

SelectAll (see page 21255)

SelLength (see page 21265)

SelStart (see page 21269)

SelText (see page 21272)

SetSelTextBuf (see page 21258)

47.1.4.5.4 TDBEdit.AutoSize Property

Determines whether the height of the edit control automatically resizes to accommodate the text.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;__property Boolean AutoSize;
```

Description

Use AutoSize to make the edit control adjust its size automatically so the client area accommodates the height of the text. When AutoSize is false, the edit control has a fixed height. When AutoSize is true, the size of the control is readjusted whenever a change occurs that could affect the height of the control, such as a change to the font or border style.

Note: AutoSize only adjusts the height of the edit control. Even with AutoSize set to true, text in the edit control may appear truncated if it extends beyond the width of the control.

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit1->AutoSize = true;
  Edit1->Font->Size = 20;
  Edit1->Text = "The edit box is taller now";
}
```

Delphi Examples:

```
{
{
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  Edit1.AutoSize := True;
  Edit1.Font.Size := 20;
  Edit1.Text := 'The edit box is taller now';
end;
```

See Also

BorderStyle (see page 21261)

ClientHeight (see page 5808)

ControlStyle (see page 5814)

Height (see page 5825)

47.1.4.5.5 TDBEdit.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

47.1.4.5.6 TDBEdit.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

47.1.4.5.7 TDBEdit.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

47.1.4.5.8 TDBEdit.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

47.1.4.5.9 TDBEdit.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

47.1.4.5.10 TDBEdit.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

47.1.4.5.11 TDBEdit.BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the edit control stands out. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the edit control has a single-line border around the client area. If BorderStyle is bsNone, there will be no border.

47.1.4.5.12 TDBEdit.CharCase Property

Determines the case of the text within the edit control.

Pascal

```
property CharCase: TEditCharCase;
```

C++

```
__property TEditCharCase CharCase;
```

Description

Use CharCase to force the contents of the edit control to assume a particular case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types them into the edit control. Changing the CharCase property to ecLowerCase or ecUpperCase changes the actual contents of the text, not just the appearance. Any case information is lost and can't be recaptured by changing CharCase to ecNormal.

See Also

Text (see page 5842)

47.1.4.5.13 TDBEdit.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

47.1.4.5.14 TDBEdit.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

47.1.4.5.15 TDBEdit.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

47.1.4.5.16 TDBEdit.DataField Property

Specifies the field from which the edit control displays data.

Pascal

```
property DataField: WideString;
```

C++

```
__property BSTR DataField;
```

Description

Set DataField to the field name of the field component that the check box represents:

```
DBEdit1.DataField := 'Company';
DBEdit1->DataField = "Company";
```

Access by the TDBEdit (see page 8735) to the dataset in which the field is located is provided by a TDataSource component, specified in the DataSource (see page 8765) property.

If the DataField refers to a database field that contains integer or floating-point data, only characters that are valid in such a field can be entered in the edit box. Characters that are not legal are not accepted.

See Also

DataSource (see page 8765)

Field (see page 8769)

47.1.4.5.17 TDBEdit.DataSource Property

Links the edit control to the dataset that contains the field it represents.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Use DataSource to specify the data source component through which the data from a dataset component is provided to the TDBEdit (see page 8735):

```
DBEdit1.DataSource := DataSource1;
DBEdit1->DataSource = DataSource1;
```

To allow the check box to represent the data for a field, both the DataSource and the DataField (see page 8765) properties must be set.

See Also

DataField (see page 8765)

Field (see page 8769)

47.1.4.5.18 TDBEdit.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

47.1.4.5.19 TDBEdit.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

47.1.4.5.20 TDBEdit.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
```

```
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

47.1.4.5.21 TDBEdit.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

47.1.4.5.22 TDBEdit.Field Property

Specifies the TField object for the database field the edit box represents.

Pascal

```
property Field: TField;
```

C++

```
__property TField Field;
```

Description

Read Field to get direct access to the contents and properties of the database field without going through the edit window. Use Field to change the contents of the database field programmatically:

```
DBEdit1.Field.AsString := 'San Francisco Baking';
DBEdit1->Field->AsString = "San Francisco Baking";
```

Use the Text property instead if you want to change the contents of the edit window without immediately posting the changes to the dataset. Changing the Text property allows the user to cancel the changes.

See Also

DataField (see page 8765)

DataSource (see page 8765)

Undo

Text (see page 5842)

47.1.4.5.23 TDBEdit.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
```

```
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

47.1.4.5.24 TDBEdit.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

47.1.4.5.25 TDBEdit.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

47.1.4.5.26 TDBEdit.MaxLength Property

Specifies the maximum number of characters that can appear in the edit control.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;__property int MaxLength;
```

Description

MaxLength is the length of the EditText. Set MaxLength to limit the number of characters that can appear in the edit control when there is no EditMask. If there is an EditMask, MaxLength is implied by the mask itself, and cannot be changed. The EditText contains blank characters for each character that has not been entered, padding the end or beginning of every optional section, so it remains constant in length.

See Also

EditMask (see page 15821)

EditText (see page 15821)

IsMasked (see page 15822)

Text (see page 15823)

47.1.4.5.27 TDBEdit.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

47.1.4.5.28 TDBEdit.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

47.1.4.5.29 TDBEdit.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

47.1.4.5.30 TDBEdit.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

47.1.4.5.31 TDBEdit.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

47.1.4.5.32 TDBEdit.PasswordChar Property

Indicates the character, if any, to display in place of the actual characters typed in the control.

Pascal

```
property PasswordChar: Char;
```

C++

```
__property Char PasswordChar;
```

Description

Use the PasswordChar property to create an edit control that displays a special character in place of any entered text. If PasswordChar is set to the null character (ANSI character zero), the edit control displays its text normally. If PasswordChar is any other character, the edit control displays PasswordChar in place of each character typed. PasswordChar affects the appearance of the edit control only. The value of the Text property reflects the actual characters that are typed.

See Also

Text (see page 5842)

47.1.4.5.33 TDBEdit.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
```

```
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
```

```
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
```

```
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
```

```
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

47.1.4.5.34 TDBEdit.ReadOnly Property

Determines whether the user can use the database edit box to change the value of the field in the current record.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set ReadOnly to true to prevent the user from changing the contents of the database field. When ReadOnly is true, the edit control is used only to display the contents of the field. If ReadOnly is false, the user can change the field's value as long as the dataset is in edit mode.

See Also

Field (see page 8769)

47.1.4.5.35 TDBEdit.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
```

```
    begin
      for I := 1 to 10 do
        Items.Add('Item ' + IntToStr(I));
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

47.1.4.5.36 TDBEdit.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

47.1.4.5.37 TDBEdit.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

47.1.4.5.38 TDBEdit.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
```

```
    {
      TTabSheet *pPage = new TTabSheet(this);
      pPage->PageControl = PageControl1;
      pPage->Caption = AnsiString("Page") + IntToStr(i);
      TEdit *pEdit = new TEdit(this);
      pEdit->Parent = pPage;
      pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
      pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
    }
    PageControl1Change(Sender);
  }
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

47.1.4.6 TDBEdit Events

The events of the TDBEdit class are listed here.

Published Events

	Name	Description
	OnChange (see page 8784)	Occurs when the text for the edit control may have changed.
	OnClick (see page 8784)	Occurs when the user clicks the control.
	OnContextPopup (see page 8786)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 8787)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 8787)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 8790)	Occurs when the user drags an object over a control.
	OnEndDock (see page 8792)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 8792)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 8793)	Occurs when a control receives the input focus.
	OnExit (see page 8796)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 8797)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 8800)	Occurs when a key is pressed.
	OnKeyUp (see page 8801)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 8803)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 8804)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 8806)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 8807)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 8807)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 8809)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 8811)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 8812)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

47.1.4.6.1 TDBEdit.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change (see page 21245)

Text (see page 5842)

47.1.4.6.2 TDBEdit.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
```

```
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

47.1.4.6.3 TDBEdit.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

47.1.4.6.4 TDBEdit.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

47.1.4.6.5 TDBEdit.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

47.1.4.6.6 TDBEdit.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.
```

```
procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.4.6.7 TDBEdit.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.4.6.8 TDBEdit.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

47.1.4.6.9 TDBEdit.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
```

```
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;
```

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

47.1.4.6.10 TDBEdit.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
```

```
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

47.1.4.6.11 TDBEdit.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
```

```
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
```

```
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

47.1.4.6.12 TDBEdit.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp

event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

47.1.4.6.13 TDBEdit.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For

non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
```

```
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

47.1.4.6.14 TDBEdit.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.4.6.15 TDBEdit.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
```

```
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
```

```
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.4.6.16 TDBEdit.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

47.1.4.6.17 TDBEdit.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

47.1.4.6.18 TDBEdit.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;
```

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.4.6.19 TDBEdit.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
```

```
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
```

```
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

47.1.4.6.20 TDBEdit.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

47.1.4.6.21 TDBEdit.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

47.1.5 TDBImage Class

TDBImage represents a graphic image from a BLOB (binary large object) field of the current record of a dataset.

Class Hierarchy

Pascal

```
TDBImage = class(TCustomControl);
```

C++

```
class TDBImage : public TCustomControl;
```

File

DBCtrls

Description

Use TDBImage to represent the value of graphic fields. TDBImage allows a form to display graphical data from a dataset. TDBImage can allow the user to edit graphical data using the clipboard. While the application is running and the database image control has focus, users can cut, copy, and paste using the standard Windows accelerator keys (Ctrl+X, Ctrl+C, and Ctrl+V). Users can return to the original state of the database image control by pressing Esc before moving to another record.

If the application doesn't require the data-aware capabilities of TDBImage, use an image control (TImage) instead to conserve system resources.

See Also

TDBEdit (see page 8735)

TDBMemo (see page 9149)

TDBRichEdit (see page 9343)

TImage (see page 12777)

Displaying and Editing Graphics Fields in an Image Control

Using Common Data Control Features

47.1.5.1 TDBImage Members

The following tables list the members exposed by TDBImage.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.

	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.

	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 8825)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.

	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 8827)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 8827)	Respond to key press events.
	KeyPress (see page 8828)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 8829)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 8829)	Renders the image of a custom control.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.

	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.

	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopyToClipboard (see page 8825)	Copies the picture in the image control to the Clipboard.
V	Create (see page 8824)	Creates and initializes an instance of TDBImage (see page 8813).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CutToClipboard (see page 8826)	Deletes the image in the control and copies it to the Clipboard.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 8824)	Destroys an instance of TDBImage (see page 8813).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
V	ExecuteAction (see page 8826)	Responds to actions that execute with the TDBImage (see page 8813) as target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.

	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	LoadPicture (see page 8828)	Loads the image stored in the field into the database image control.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	PasteFromClipboard (see page 8830)	Copies the contents of the Clipboard to the control.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.

	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
V	UpdateAction (see page 8830)	Updates an action to reflect the current state of the TDBImage (see page 8813).
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.

R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Field (see page 8842)	Specifies the TField component the database image is linked to.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	Picture (see page 8845)	Represents the image that appears on the database image control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 8856)	Occurs when the user clicks the control.
	OnContextPopup (see page 8858)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 8858)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 8859)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 8861)	Occurs when the user drags an object over a control.
	OnEndDock (see page 8863)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 8864)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 8865)	Occurs when a control receives the input focus.
	OnExit (see page 8867)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 8869)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 8872)	Occurs when a key is pressed.

	OnKeyUp (see page 8873)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 8874)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 8875)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 8878)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 8878)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 8878)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 8880)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 8883)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 8883)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 8831)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 8835)	Specifies how the control is anchored to its parent.
	AutoDisplay (see page 8835)	Determines whether to automatically display the contents of a graphic BLOB in the database image control.
	BorderStyle (see page 8836)	Determines whether the database image control has a border.
	Center (see page 8836)	Determines whether the image is centered in the image control.
	Color (see page 8836)	Specifies the background color of the control.
	Constraints (see page 8837)	Specifies the size constraints for the control.
	Ctl3D (see page 8838)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DataField (see page 8838)	Specifies the field from which the database image displays data.
	DataSource (see page 8839)	Links the image control to a dataset.
	DragCursor (see page 8839)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 8839)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 8840)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 8841)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 8842)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentColor (see page 8843)	Determines where a control looks for its color information.
	ParentCtl3D (see page 8844)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 8844)	Determines where a control looks for its font information.
	ParentShowHint (see page 8845)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 8846)	Identifies the pop-up menu associated with the control.
	QuickDraw (see page 8850)	Specifies whether the image is displayed using a palette.
	ReadOnly (see page 8850)	Determines whether the user can change the contents of the field using the image control.

	ShowHint (see page 8851)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Stretch (see page 8852)	Determines whether bitmaps and metafiles assume the size and shape of the database image.
	TabOrder (see page 8853)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 8853)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 8854)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

47.1.5.2 TDBImage.Create Constructor

Creates and initializes an instance of TDBImage (see page 8813).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDBImage(TComponent * AOwner);
```

Description

Call Create to instantiate a TDBImage (see page 8813) object at runtime. For TDBImage (see page 8813) objects placed on forms or data modules at design time, the constructor is called automatically.

Pass a single Component as a parameter to provide the TDBLookupComboBox (see page 8963) object with an Owner that is responsible for freeing it.

See Also

Destroy (see page 8824)

Owner (see page 2807)

47.1.5.3 TDBImage.Destroy Destructor

Destroys an instance of TDBImage (see page 8813).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBImage();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TDBImage (see page 8813) reference is not nil and only then calls Destroy.

See Also

Create (see page 8824)

Free (see page 22300)

47.1.5.4 TDBImage Methods

The methods of the TDBImage class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 8825)	Initializes a window-creation parameter data structure.
	GetPalette (see page 8827)	Returns the handle to a palette for use by the control.
	KeyDown (see page 8827)	Respond to key press events.
	KeyPress (see page 8828)	Respond to keyboard input.
	Notification (see page 8829)	Responds to notifications that components are being created or destroyed.
	Paint (see page 8829)	Renders the image of a custom control.

Public Methods

	Name	Description
	CopyToClipboard (see page 8825)	Copies the picture in the image control to the Clipboard.
	CutToClipboard (see page 8826)	Deletes the image in the control and copies it to the Clipboard.
	ExecuteAction (see page 8826)	Responds to actions that execute with the TDBImage (see page 8813) as target.
	LoadPicture (see page 8828)	Loads the image stored in the field into the database image control.
	PasteFromClipboard (see page 8830)	Copies the contents of the Clipboard to the control.
	UpdateAction (see page 8830)	Updates an action to reflect the current state of the TDBImage (see page 8813).

47.1.5.4.1 TDBImage.CopyToClipboard Method

Copies the picture in the image control to the Clipboard.

Pascal

```
procedure CopyToClipboard;
```

C++

```
__fastcall CopyToClipboard();
```

Description

Use CopyToClipboard to put a copy of the graphic specified by the Picture (see page 8845) property in the Clipboard.

See Also

CutToClipboard (see page 8826)

PasteFromClipboard (see page 8830)

Picture (see page 8845)

Displaying and Editing Graphics Fields in an Image Control

47.1.5.4.2 TDBImage.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

47.1.5.4.3 TDBImage.CutToClipboard Method

Deletes the image in the control and copies it to the Clipboard.

Pascal

```
procedure CutToClipboard;
```

C++

```
__fastcall CutToClipboard();
```

Description

Use the CutToClipboard method to delete the image in the control when copying it to the Clipboard. CutToClipboard gives the associated field a null value.

See Also

CopyToClipboard (see page 8825)

PasteFromClipboard (see page 8830)

Displaying and Editing Graphics Fields in an Image Control

47.1.5.4.4 TDBImage.ExecuteAction Method

Responds to actions that execute with the TDBImage (see page 8813) as target.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

ExecuteAction is called automatically when the user invokes an action with the TDBImage (see page 8813) as a target. The

Action parameter specifies the action that was invoked. ExecuteAction returns trueif the action was successfully dispatched, and false if the TDBImage (see page 8813) could not handle the action.

See Also

UpdateAction (see page 8830)

Execute (see page 2712)

HandlesTarget (see page 2713)

47.1.5.4.5 TDBImage.GetPalette Method

Returns the handle to a palette for use by the control.

Pascal

```
function GetPalette: HPALETTE; override;
```

C++

```
virtual __fastcall HPALETTE GetPalette();
```

Description

As implemented in TControl, GetPalette returns a null handle, indicating that the control does not have a palette. Override GetPalette to implement derived objects that use a palette. Create a palette handle for the control by a call to the Windows API function CreatePalette. GetPalette should return the palette handle for the control.

Specifying the palette for a control tells the application that the control's palette needs to be realized and designates the palette to use for realization.

GetPalette is called only if the run-time video mode requires color palette support, for example, in 256-color mode.

See Also

PaletteChanged (see page 5775)

47.1.5.4.6 TDBImage.KeyDown Method

Respond to key press events.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-down message (WM_KEYDOWN) from Windows, its message handler calls the DoKeyDown method. If DoKeyDown determines that the control should, in fact, process the character, it calls KeyDown, passing the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown event handler it calls can suppress further processing of a key by setting the Key parameter

to zero.

See Also

KeyUp (see page 6164)

OnKeyDown (see page 6225)

DoKeyDown (see page 6145)

DoKeyPress (see page 6146)

DoKeyUp (see page 6146)

47.1.5.4.7 TDBImage.KeyPress Method

Respond to keyboard input.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

When a windowed control receives a key-press message (WM_CHAR) from Windows, its message handler calls the DoKeyPress method. If DoKeyPress determines that the control should, in fact, process the character, it calls KeyPress, passing the key code in the Key parameter.

KeyPress calls any event handler attached to the OnKeyPress event. Override KeyPress to provide other responses in addition to the event-handler call.

Either KeyPress or the OnKeyPress event handler it calls can suppress further processing of a character by setting the Key parameter to zero.

Note: The Key parameter is the character represented by the key that is pressed, not a Windows virtual key code.

See Also

KeyDown (see page 6163)

KeyUp (see page 6164)

OnKeyPress (see page 6227)

47.1.5.4.8 TDBImage.LoadPicture Method

Loads the image stored in the field into the database image control.

Pascal

```
procedure LoadPicture;
```

C++

```
__fastcall LoadPicture();
```

Description

If the value of the AutoDisplay (see page 8835) property is false, the image of a database image control is not automatically loaded. If AutoDisplay (see page 8835) is false, control when the image is loaded at runtime by calling LoadPicture when the image should appear in the control.

See Also

Picture (see page 8845)

PasteFromClipboard (see page 8830)

47.1.5.4.9 TDBImage.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

47.1.5.4.10 TDBImage.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas (see page 5921)

PaintWindow (see page 5920)

GetDeviceContext (see page 6155)

ClipRect (see page 14486)

47.1.5.4.11 TDBImage.PasteFromClipboard Method

Copies the contents of the Clipboard to the control.

Pascal

```
procedure PasteFromClipboard;
```

C++

```
__fastcall PasteFromClipboard();
```

Description

Use PasteFromClipboard to set the field value to an image stored in the Clipboard.

See Also

CutToClipboard (see page 8826)

CopyToClipboard (see page 8825)

Displaying and Editing Graphics Fields in an Image Control

47.1.5.4.12 TDBImage.UpdateAction Method

Updates an action to reflect the current state of the TDBImage (see page 8813).

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

UpdateAction is called automatically when the application is idle to give the TDBImage (see page 8813) an opportunity to update associated actions. This allows actions to be grayed, checked, or to otherwise reflect the status of the TDBImage (see page 8813) they target.

The Action parameter specifies the action component that should be updated. UpdateAction returns true if the action component now reflects the state of the component, and false if it did not know how to update the action.

See Also

ExecuteAction (see page 8826)

UpdateTarget (see page 2715)

HandlesTarget (see page 2713)

47.1.5.5 TDBImage Properties

The properties of the TDBImage class are listed here.

Public Properties

	Name	Description
R	Field (see page 8842)	Specifies the TField component the database image is linked to.
	Picture (see page 8845)	Represents the image that appears on the database image control.

Published Properties

	Name	Description
	Align (see page 8831)	Determines how the control aligns within its container (parent control).
	Anchors (see page 8835)	Specifies how the control is anchored to its parent.
	AutoDisplay (see page 8835)	Determines whether to automatically display the contents of a graphic BLOB in the database image control.
	BorderStyle (see page 8836)	Determines whether the database image control has a border.
	Center (see page 8836)	Determines whether the image is centered in the image control.
	Color (see page 8836)	Specifies the background color of the control.
	Constraints (see page 8837)	Specifies the size constraints for the control.
	Ctl3D (see page 8838)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DataField (see page 8838)	Specifies the field from which the database image displays data.
	DataSource (see page 8839)	Links the image control to a dataset.
	DragCursor (see page 8839)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 8839)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 8840)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 8841)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 8842)	Controls the attributes of text written on or in the control.
	ParentColor (see page 8843)	Determines where a control looks for its color information.
	ParentCtl3D (see page 8844)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 8844)	Determines where a control looks for its font information.
	ParentShowHint (see page 8845)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 8846)	Identifies the pop-up menu associated with the control.
	QuickDraw (see page 8850)	Specifies whether the image is displayed using a palette.
	ReadOnly (see page 8850)	Determines whether the user can change the contents of the field using the image control.
	ShowHint (see page 8851)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Stretch (see page 8852)	Determines whether bitmaps and metafiles assume the size and shape of the database image.
	TabOrder (see page 8853)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 8853)	Determines if the user can tab to a control.
	Visible (see page 8854)	Determines whether the component appears onscreen.

47.1.5.5.1 TDBImage.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
```

```
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
```

```
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
```

```
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

47.1.5.5.2 TDBImage.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

47.1.5.5.3 TDBImage.AutoDisplay Property

Determines whether to automatically display the contents of a graphic BLOB in the database image control.

Pascal

```
property AutoDisplay: Boolean;
```

C++

```
__property Boolean AutoDisplay;
```

Description

If AutoDisplay is true (the default value), the image automatically displays new data when the underlying BLOB field changes (such as when moving to a new record).

If AutoDisplay is false, the image clears whenever the underlying BLOB field changes. To display the data, the user can double-click on the control or select it and press Enter. In addition, calling the LoadPicture (see page 8828) method ensures that the control is showing data.

Change the value of AutoDisplay to false if the automatic loading of BLOB fields seems to take too long.

See Also

LoadPicture (see page 8828)

Displaying and Editing Graphics Fields in an Image Control

47.1.5.5.4 TDBImage.BorderStyle Property

Determines whether the database image control has a border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Set BorderStyle to specify whether a single line border should be drawn around the database image control.

See Also

Ctl3D (see page 6198)

47.1.5.5.5 TDBImage.Center Property

Determines whether the image is centered in the image control.

Pascal

```
property Center: Boolean;
```

C++

```
__property Boolean Center;
```

Description

When the image does not fit perfectly within the image control, use Center to position the image. When Center is true, the image is centered in the control. When Center is false, the upper left corner of the image is positioned at the upper left corner of the control.

Note: Center has no effect if the Stretch (see page 8852) property is true.

See Also

Stretch (see page 8852)

47.1.5.5.6 TDBImage.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

47.1.5.5.7 TDBImage.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the

ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

47.1.5.5.8 TDBImage.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

47.1.5.5.9 TDBImage.DataField Property

Specifies the field from which the database image displays data.

Pascal

```
property DataField: WideString;
```

C++

```
__property BSTR DataField;
```

Description

Use DataField to bind the image control to a field in the dataset. To fully specify a database field, both the dataset and the field

within that dataset must be defined. The DataSource (see page 8839) property of the image control specifies the dataset which contains the DataField. DataField should specify a graphic field.

```
DBImage1.DataField := 'Portrait';
DBImage1->DataField = "Portrait";
```

See Also

DataSource (see page 8839)

Field (see page 8842)

47.1.5.5.10 TDBImage.DataSource Property

Links the image control to a dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Use DataSource to link the image control to a dataset in which the data can be found. To fully specify a database field for the image control, both the dataset and a field within that dataset must be defined. Use the DataField (see page 8838) property to specify the particular field within the dataset.

```
DBImage1.DataSource := DataSource1;
DBImage1->DataSource = DataSource1;
```

See Also

DataField (see page 8838)

47.1.5.5.11 TDBImage.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

47.1.5.5.12 TDBImage.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

47.1.5.5.13 TDBImage.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
```

```
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

47.1.5.5.14 TDBImage.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

47.1.5.5.15 TDBImage.Field Property

Specifies the TField component the database image is linked to.

Pascal

```
property Field: TField;
```

C++

```
__property TField Field;
```

Description

Read Field to get direct access to the contents and properties of the database field without going through the image control's window. Use Field to change the contents of the database field programmatically. Use the Picture (see page 8845) property to set the value of the field and display it in the image control's window.

```
(DBEdit1.Field as TBlobField).Assign(Image1.Picture.Bitmap);
```

See Also

DataField (see page 8838)

Picture (see page 8845)

47.1.5.5.16 TDBImage.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
```

```
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

47.1.5.5.17 TDBImage.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

47.1.5.5.18 TDBImage.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

47.1.5.5.19 TDBImage.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

47.1.5.5.20 TDBImage.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

47.1.5.5.21 TDBImage.Picture Property

Represents the image that appears on the database image control.

Pascal

```
property Picture: TPicture;
```

C++

```
__property TPicture Picture;
```

Description

Read Picture to obtain the TPicture object that represents the value of the field. Set Picture to programmatically change the value of the graphic field. The property value is a TPicture object which can contain an icon, metafile, bitmap, or other graphic object.

```
Image1.Picture.Assign(DBImage1.Picture);
```

See Also

CutToClipboard (see page 8826)

CopyToClipboard (see page 8825)

LoadPicture (see page 8828)

PasteFromClipboard (see page 8830)

47.1.5.5.22 TDBImage.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
```

```
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
```

```
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
```

```
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

47.1.5.5.23 TDBImage.QuickDraw Property

Specifies whether the image is displayed using a palette.

Pascal

```
property QuickDraw: Boolean;
```

C++

```
__property Boolean QuickDraw;
```

Description

Set QuickDraw to specify whether a customized palette should be used when displaying field values. If false, a palette is used, to provide the best possible image quality at the expense of additional processing time. If true, no special palette is used, which is faster, but results in poorer picture quality, especially with 256-color images on a 256-color video driver.

See Also

LoadPicture (see page 8828)

47.1.5.5.24 TDBImage.ReadOnly Property

Determines whether the user can change the contents of the field using the image control.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set ReadOnly to true to prevent the user from changing the contents of the database field. When ReadOnly is true, the image control is used only to display the contents of the field. If ReadOnly is false, the user can change the field's value as long as the dataset is in edit mode.

See Also

DataField (see page 8838)

DataSource (see page 8839)

47.1.5.5.25 TDBImage.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
```

```
      strcat(string, index);
      ListBox1->Items->Add(string);
    }
    ListBox1->Hint = "Select an item";
    ListBox1->ShowHint = true;
    Application->ShowHint = true;
  }
  /*
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  */
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

47.1.5.5.26 TDBImage.Stretch Property

Determines whether bitmaps and metafiles assume the size and shape of the database image.

Pascal

```
property Stretch: Boolean;
```

C++

```
__property Boolean Stretch;
```

Description

Set Stretch to true to cause the picture to assume the size and shape of the image control. When the image control resizes, the

picture resizes also. Stretch resizes the height and width of the image independently. Thus, unlike a simple change in magnification, stretch can distort the image if the image control is not the same shape as the image.

See Also

Picture (see page 8845)

Displaying and Editing Graphics Fields in an Image Control

47.1.5.5.27 TDBImage.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

47.1.5.5.28 TDBImage.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

47.1.5.5.29 TDBImage.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
```

```
  edit control placed somewhere on the client area of the
  sheet.  Unchecking the checkbox will cause the client area
  (not the tab area) of the Tab Control to become invisible.
  }
  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    PageControl1.ActivePage.Visible := CheckBox1.Checked;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    CheckBox1.Checked := PageControl1.ActivePage.Visible;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

47.1.5.6 TDBImage Events

The events of the TDBImage class are listed here.

Published Events

	Name	Description
	OnClick (see page 8856)	Occurs when the user clicks the control.
	OnContextPopup (see page 8858)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 8858)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 8859)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 8861)	Occurs when the user drags an object over a control.
	OnEndDock (see page 8863)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 8864)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 8865)	Occurs when a control receives the input focus.
	OnExit (see page 8867)	Occurs when the input focus shifts away from one control to another.

	OnKeyDown (see page 8869)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 8872)	Occurs when a key is pressed.
	OnKeyUp (see page 8873)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 8874)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 8875)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 8878)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 8878)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 8878)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 8880)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 8883)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 8883)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

47.1.5.6.1 TDBImage.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
```

```
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

47.1.5.6.2 TDBImage.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

47.1.5.6.3 TDBImage.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

47.1.5.6.4 TDBImage.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
```

```
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
```

```
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

47.1.5.6.5 TDBImage.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/
```

```
// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
```

```
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.5.6.6 TDBImage.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.5.6.7 TDBImage.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
```

```
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

47.1.5.6.8 TDBImage.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}
```

```
void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
```

```
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

47.1.5.6.9 TDBImage.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit

event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
```

```
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

47.1.5.6.10 TDBImage.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
```

```
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
```

```
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
  {
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

47.1.5.6.11 TDBImage.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

47.1.5.6.12 TDBImage.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

47.1.5.6.13 TDBImage.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.5.6.14 TDBImage.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
```

```
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.5.6.15 TDBImage.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

47.1.5.6.16 TDBImage.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

47.1.5.6.17 TDBImage.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
```

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.5.6.18 TDBImage.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

47.1.5.6.19 TDBImage.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

47.1.5.6.20 TDBImage.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

47.1.6 TDBListBox Class

TDBListBox represents a data-aware list box that allows users to change field values by selecting an item from a list.

Class Hierarchy

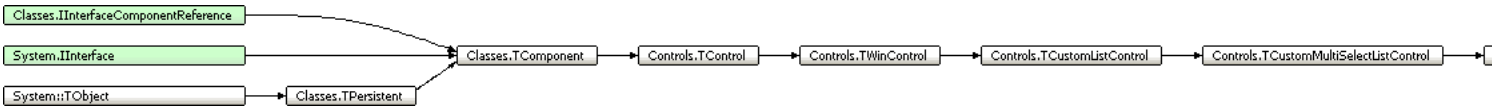

Pascal

```
TDBListBox = class(TCustomListBox);
```

C++

```
class TDBListBox : public TCustomListBox;
```

File

DBCtrls

Description

Use TDBListBox to add a list box to a form that permits users to change the value of a field on the current record to one of a fixed set of choices.

If the application doesn't require the data-aware capabilities of TDBListBox, use a list box (TListBox) instead to conserve system resources.

Component writers wanting to create a customized list box objects should use TCustomListBox as a base class.

See Also

TDBComboBox (see page 8657)

TDBLookupListBox (see page 9070)

TListBox (see page 21598)

Displaying and Editing Data in List and Combo Boxes

Using Common Data Control Features

47.1.6.1 TDBListBox Members

The following tables list the members exposed by TDBListBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnData (see page 21357)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 21357)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 21358)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FMoving (see page 21321)	This is FMoving, a member of class TCustomListBox.
	FMultiSelect (see page 5956)	This is FMultiSelect, a member of class TCustomMultiSelectListControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.

	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
V	Click (see page 8897)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 21324)	Initializes the window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CreateWnd (see page 21325)	Creates a Windows control corresponding to the list box component.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DeleteString (see page 21326)	Deletes a string from the list box.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
V	DestroyWnd (see page 21326)	Saves the list box items in a string list before freeing the underlying Windows control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.

	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFindData (see page 21326)	Returns the index of a string in the list box.
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetData (see page 21327)	Returns a string in the list box given its index.
	DoGetDataObject (see page 21327)	Returns the object associated with an item in a virtual list box.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragCanceled (see page 21328)	Sends a mouse message canceling the drag-and-drop operation.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawItem (see page 21328)	Generates an OnDrawItem (see page 21358) event.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetCount (see page 21329)	Returns the value of the Count (see page 21346) property.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetItemData (see page 21329)	Returns a 32-bit value associated with the specified item in a list box.
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.

	GetItemIndex (see page 21329)	Returns the value of the ItemIndex property.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelCount (see page 5956)	Returns the value of the SelCount (see page 5957) property.
	GetSelCount (see page 21330)	Returns the number of selected items.
	GetSelected (see page 21330)	Indicates whether a particular item is selected.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InternalGetItemData (see page 21330)	Retrieves the data associated with an item in the list box.
	InternalSetItemData (see page 21330)	Associates an integer with a specified item in the list box.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 8898)	Respond to key press events.
	KeyPress (see page 8899)	Responds when the user presses a key.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	LoadRecreateItems (see page 21334)	This is LoadRecreateItems, a member of class TCustomListBox.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MeasureItem (see page 21335)	Generates an OnMeasureItem (see page 21360) event.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 8899)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.

	ResetContent (see page 21335)	Removes all of the items from a list box.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
V	SaveRecreateItems (see page 21336)	This is SaveRecreateItems, a member of class TCustomListBox.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItemData (see page 21336)	Sets the 32-bit value associated with the specified item in a list box.
A	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
V	SetItemIndex (see page 21336)	Gives focus to the item with a specified index.
A	SetMultiSelect (see page 5956)	Sets the value of the MultiSelect (see page 5957) property.
V	SetMultiSelect (see page 21337)	Sets the MultiSelect property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WndProc (see page 21337)	Handles Windows messages.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Columns (see page 21346)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ExtendedSelect (see page 21346)	This is ExtendedSelect, a member of class TCustomListBox.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Sorted (see page 21353)	Specifies whether the items in a list box are arranged alphabetically.
	TabWidth (see page 21355)	Specifies the size of the tabs in the list box.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddItem (see page 5940)	Adds an item to the list control.
	AddItem (see page 21322)	Adds an item to the list box
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.

	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 5940)	Deletes all of the items from the list control.
V	Clear (see page 21322)	Deletes all items from the list box.
A	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
V	ClearSelection (see page 21323)	Deselects all selected items in the list box, if possible.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
A	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
V	CopySelection (see page 21323)	Copies the selected items to another list-type control
V	Create (see page 8896)	Creates and initializes an instance of TDBListBox (see page 8884).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	DeleteSelected (see page 5941)	Deletes all the selected items.
V	DeleteSelected (see page 21325)	Deletes the selected items from the list box
V	Destroy (see page 8897)	Destroys an instance of TDBListBox (see page 8884).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
V	ExecuteAction (see page 8898)	Invokes an action with the TDBListBox (see page 8884) as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ItemAtPos (see page 21331)	Returns the index of the list box item indicated by the coordinates of a point on the control.
	ItemRect (see page 21333)	Returns the rectangle that surrounds the item specified in the Item parameter.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
A	SelectAll (see page 5943)	Selects all items or all text in the selected item.
V	SelectAll (see page 21336)	Selects the entire block of text in the list box.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
V	UpdateAction (see page 8900)	Updates an action to reflect the current state of the TDBListBox (see page 8884).
	UpdateControlState (see page 6186)	Respond to state change.
V	UseRightToLeftAlignment (see page 8900)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AutoCompleteDelay (see page 21342)	Specifies the delay between a key press and an attempt to autocomplete the field.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 21343)	Provides a drawing surface when implementing a handler for the OnDrawItem (see page 21358) event of an owner-draw list box.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.

R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Count (see page 21346)	Indicates the number of items in the list box.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Field (see page 8915)	Identifies the TField object the listbox is linked to.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	MultiSelect (see page 5957)	Specifies whether the user can select more than one item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ScrollWidth (see page 21351)	Specifies the width, in pixels, by which the list box can scroll horizontally.
R	SelCount (see page 5957)	Indicates the number of selected items.
	Selected (see page 21351)	Indicates whether a particular item is selected.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TopIndex (see page 21356)	Specifies the index number of the item that appears at the top of the list box.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 8931)	Occurs when the user clicks the control.
	OnContextPopup (see page 8933)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 8934)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 8934)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 8936)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 8939)	Occurs when an item in an owner-draw list box needs to be redisplayed.

	OnEndDock (see page 8941)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 8941)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 8942)	Occurs when a control receives the input focus.
	OnExit (see page 8945)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 8946)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 8949)	Occurs when a key is pressed.
	OnKeyUp (see page 8950)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 8952)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.
	OnMouseActivate (see page 8953)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 8954)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 8957)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 8957)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 8957)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 8959)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 8962)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 8962)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 8902)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 8905)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 8906)	Determines whether the user can give focus to items by typing in the list.
	BevelEdges (see page 8906)	Specifies which edges of the control are beveled.
	BevelInner (see page 8906)	Specifies the cut of the inner bevel.
	BevelKind (see page 8907)	Specifies the control's bevel style.
	BevelOuter (see page 8907)	Specifies the cut of the outer bevel.
	BevelWidth (see page 8908)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 8908)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 8909)	Determines whether the list box has a border.
	Color (see page 8909)	Specifies the background color of the control.
	Constraints (see page 8910)	Specifies the size constraints for the control.
	Ctl3D (see page 8910)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DataField (see page 8911)	Identifies the field whose value is displayed by the listbox.
	DataSource (see page 8911)	Links the listbox to a dataset.
	DragCursor (see page 8912)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 8912)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 8913)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 8914)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 8915)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.

	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 8916)	Determines the behavior of the input method editor (IME).
	ImeName (see page 8917)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 8917)	Determines whether the list box displays the partial items.
	ItemHeight (see page 8918)	Specifies the height in pixels of an item in an owner-draw list box.
W	Items (see page 8918)	Lists the strings that appear in the list box.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 8919)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 8919)	Determines where a control looks for its color information.
	ParentCtl3D (see page 8920)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 8920)	Determines where a control looks for its font information.
	ParentShowHint (see page 8920)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 8921)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 8926)	Determines if the user can change the field value using the listbox.
	ShowHint (see page 8926)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 8927)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabOrder (see page 8928)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 8929)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 8929)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

47.1.6.2 TDBListBox.Create Constructor

Creates and initializes an instance of TDBListBox (see page 8884).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDBListBox(TComponent * AOwner);
```

Description

Call Create to instantiate a TDBListBox (see page 8884) object at runtime. For TDBListBox (see page 8884) objects placed on forms or data modules at design time, Create is called automatically.

AOwner is the component, typically the form, that is responsible for freeing the TDBListBox (see page 8884) instance. It becomes the value of the Owner property.

See Also

Destroy (see page 8897)

Owner (see page 2807)

47.1.6.3 TDBListBox.Destroy Destructor

Destroys an instance of TDBListBox (see page 8884).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBListBox();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TDBListBox (see page 8884) reference is not nil and only then calls Destroy.

See Also

Create (see page 8896)

Free (see page 22300)

47.1.6.4 TDBListBox Methods

The methods of the TDBListBox class are listed here.

Protected Methods

	Name	Description
	Click (see page 8897)	Respond to user click.
	KeyDown (see page 8898)	Respond to key press events.
	KeyPress (see page 8899)	Responds when the user presses a key.
	Notification (see page 8899)	Responds to notifications that components are being created or destroyed.

Public Methods

	Name	Description
	ExecuteAction (see page 8898)	Invokes an action with the TDBListBox (see page 8884) as its target.
	UpdateAction (see page 8900)	Updates an action to reflect the current state of the TDBListBox (see page 8884).
	UseRightToLeftAlignment (see page 8900)	Specifies whether the control's alignment should be reversed right-to-left.

47.1.6.4.1 TDBListBox.Click Method

Respond to user click.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Click is called automatically when the user left-clicks the control. Component or application code can call Click to simulate a user mouse click. This is often done in menu actions and hotkey handlers.

As implemented in TControl, Click queries whether the OnClick event handler exists and is different from the OnExecute handler for the control's Action. If this is true, the OnClick event handler is called. If this is false, and the Action OnExecute event handler

exists, the Action's Execute method is called. Override Click to provide additional behavior.

See Also

Action (see page 5797)

OnExecute (see page 2716)

Execute (see page 2712)

DblClick (see page 5747)

OnClick (see page 5851)

47.1.6.4.2 TDBListBox.ExecuteAction Method

Invokes an action with the TDBListBox (see page 8884) as its target.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

ExecuteAction is called automatically when the user invokes an action with the TDBListBox (see page 8884) as a target. The Action parameter specifies the action that was invoked. ExecuteAction returns true if the action was successfully dispatched, and false if the TDBListBox (see page 8884) could not handle the action.

See Also

UpdateAction (see page 8900)

Execute (see page 2712)

HandlesTarget (see page 2713)

47.1.6.4.3 TDBListBox.KeyDown Method

Respond to key press events.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-down message (WM_KEYDOWN) from Windows, its message handler calls the DoKeyDown method. If DoKeyDown determines that the control should, in fact, process the character, it calls KeyDown, passing the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

See Also

KeyUp (see page 6164)

OnKeyDown (see page 6225)

DoKeyDown (see page 6145)

DoKeyPress (see page 6146)

DoKeyUp (see page 6146)

47.1.6.4.4 TDBListBox.KeyPress Method

Responds when the user presses a key.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress is called automatically when the list box receives a key-press message (WM_CHAR). It generates an OnKeyPress event, and then, if the AutoComplete property is true, updates ItemIndex to reflect the next character that the user typed.

The Key parameter is the character represented by the key that is pressed, not a Windows virtual key code.

See Also

OnKeyPress (see page 6227)

47.1.6.4.5 TDBListBox.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

47.1.6.4.6 TDBListBox.UpdateAction Method

Updates an action to reflect the current state of the TDBListBox (see page 8884).

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

UpdateAction is called automatically when the application is idle to give the TDBListBox (see page 8884) an opportunity to update associated actions. This allows actions to be grayed, checked, or to otherwise reflect the status of the TDBListBox (see page 8884) they target.

The Action parameter specifies the action component that should be updated. UpdateAction returns trueif the action component now reflects the state of the component, and false if it did not know how to update the action.

See Also

ExecuteAction (see page 8898)

UpdateTarget (see page 2715)

HandlesTarget (see page 2713)

47.1.6.4.7 TDBListBox.UseRightToLeftAlignment Method

Specifies whether the control's alignment should be reversed right-to-left.

Pascal

```
function UseRightToLeftAlignment: Boolean; override;
```

C++

```
virtual __fastcall Boolean UseRightToLeftAlignment();
```

Description

Controls call UseRightToLeftAlignment when implementing the BiDiMode property. UseRightToLeftAlignment indicates whether the control's alignment should be reversed right-to-left. This information can be used when painting the control or when responding to mouse events.

UseRightToLeftAlignment returns true if the application is running on a system with a Middle Eastern locale and the BiDiMode property is bdRightToLeft. Otherwise, it returns false. Descendants override UseRightToLeftAlignment if alignment in Middle Eastern locales depends on other factors.

See Also

BiDiMode (see page 5803)

GetControlsAlignment (see page 5759)

Text (see page 5842)

Caption (see page 5806)

47.1.6.5 TDBListBox Properties

The properties of the TDBListBox class are listed here.

Public Properties

	Name	Description
R	Field (see page 8915)	Identifies the TField object the listbox is linked to.

Published Properties

	Name	Description
	Align (see page 8902)	Determines how the control aligns within its container (parent control).
	Anchors (see page 8905)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 8906)	Determines whether the user can give focus to items by typing in the list.
	BevelEdges (see page 8906)	Specifies which edges of the control are beveled.
	BevelInner (see page 8906)	Specifies the cut of the inner bevel.
	BevelKind (see page 8907)	Specifies the control's bevel style.
	BevelOuter (see page 8907)	Specifies the cut of the outer bevel.
	BevelWidth (see page 8908)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 8908)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 8909)	Determines whether the list box has a border.
	Color (see page 8909)	Specifies the background color of the control.
	Constraints (see page 8910)	Specifies the size constraints for the control.
	Ctl3D (see page 8910)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DataField (see page 8911)	Identifies the field whose value is displayed by the listbox.
	DataSource (see page 8911)	Links the listbox to a dataset.
	DragCursor (see page 8912)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 8912)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 8913)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 8914)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 8915)	Controls the attributes of text written on or in the control.
	ImeMode (see page 8916)	Determines the behavior of the input method editor (IME).
	ImeName (see page 8917)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 8917)	Determines whether the list box displays the partial items.
	ItemHeight (see page 8918)	Specifies the height in pixels of an item in an owner-draw list box.
W	Items (see page 8918)	Lists the strings that appear in the list box.
	ParentBiDiMode (see page 8919)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 8919)	Determines where a control looks for its color information.
	ParentCtl3D (see page 8920)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 8920)	Determines where a control looks for its font information.
	ParentShowHint (see page 8920)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 8921)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 8926)	Determines if the user can change the field value using the listbox.
	ShowHint (see page 8926)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 8927)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabOrder (see page 8928)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 8929)	Determines if the user can tab to a control.
	Visible (see page 8929)	Determines whether the component appears onscreen.

47.1.6.5.1 TDBListBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
```

```
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
```

```
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
```

```
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

47.1.6.5.2 TDBListBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

47.1.6.5.3 TDBListBox.AutoComplete Property

Determines whether the user can give focus to items by typing in the list.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

Use AutoComplete to specify whether the list box automatically updates the ItemIndex property in response to user keystrokes. When AutoComplete is true, the item with focus changes to the first item that begins with the sequence of characters that the user typed. As the user types more characters, ItemIndex may change to a later item because the string to match becomes longer.

See Also

ItemIndex

47.1.6.5.4 TDBListBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

47.1.6.5.5 TDBListBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

47.1.6.5.6 TDBListBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

47.1.6.5.7 TDBListBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

47.1.6.5.8 TDBListBox.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

47.1.6.5.9 TDBListBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

47.1.6.5.10 TDBListBox.BorderStyle Property

Determines whether the list box has a border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to change the appearance of the outer boundary of the list box control. A list box can have a single border (bsSingle) or no visible border (bsNone).

See Also

Ctl3D (see page 6198)

47.1.6.5.11 TDBListBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
```

```
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

47.1.6.5.12 TDBListBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

47.1.6.5.13 TDBListBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

47.1.6.5.14 TDBListBox.DataField Property

Identifies the field whose value is displayed by the listbox.

Pascal

```
property DataField: WideString;
```

C++

```
__property BSTR DataField;
```

Description

Set DataField to specify which field is represented by the listbox. The listbox displays the current value of this field, and lets the user set its value on the current record. The dataset in which the field is located is specified by the DataSource (see page 8911) property.

Set DataField to specify which field is represented by the listbox. (Note that this is does not refer to the source of the list, but to the field into which the selected value is inserted. The list values must be supplied manually using the Items (see page 8918) property.) The listbox displays the current value of this field as the selected (highlighted) item, and allows the user to set the value of this field on the current record by selecting a value from the list. The dataset the field is located in is specified by the DataSource (see page 8911) property.

```
DBListBox1.DataField := 'CountryName';
DBListBox1->DataField = "CountryName";
```

Tip: For a data-aware listbox component for which the list values are automatically supplied, use a TDBLookupListBox (see page 9070) instead.

See Also

DataSource (see page 8911)

Field (see page 8915)

47.1.6.5.15 TDBListBox.DataSource Property

Links the listbox to a dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Specify the data source component that identifies the dataset the data is found in. To allow the listbox to represent the data for a field, both the DataSource and the DataField (see page 8911) properties must be set.

Specify the data source component that identifies the dataset in which the target field is found. To allow the listbox to represent the data for a field, both the DataSource and the DataField (see page 8911) properties must be set.

```
DBListBox1.DataSource := DataSource1;
DBListBox1->DataSource = DataSource1;
```

See Also

DataField (see page 8911)

Field (see page 8915)

47.1.6.5.16 TDBListBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

47.1.6.5.17 TDBListBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

47.1.6.5.18 TDBListBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
```

```
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

47.1.6.5.19 TDBListBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

47.1.6.5.20 TDBListBox.Field Property

Identifies the TField object the listbox is linked to.

Pascal

```
property Field: TField;
```

C++

```
__property TField Field;
```

Description

Use the Field object when you want to change the value of the data in the field programmatically.

```
DBListBox1.Field.AsString := 'Lithuania';
DBListBox1->Field->AsString = "Lithuania";
```

See Also

DataField (see page 8911)

DataSource (see page 8911)

47.1.6.5.21 TDBListBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

47.1.6.5.22 TDBListBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

47.1.6.5.23 TDBListBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

47.1.6.5.24 TDBListBox.IntegralHeight Property

Determines whether the list box displays the partial items.

Pascal

```
property IntegralHeight: Boolean;
```

C++

```
__property Boolean IntegralHeight;
```

Description

IntegralHeight controls the height of the list box itself.

When IntegralHeight is true, and the Style property is lsOwnerDrawFixed, the list box height will always be a multiple of the ItemHeight property value. In this case, items will never be partially displayed.

If IntegralHeight is false, the list box height can be set to any value, but the last item may not be fully displayed.

If the Style property value of the list box is lbOwnerDrawVariable, setting the IntegralHeight property to true has no effect.

See Also

ItemHeight (see page 21347)

Items (see page 21348)

Style (see page 21354)

Using TDBListBox and TDBComboBox

47.1.6.5.25 TDBListBox.ItemHeight Property

Specifies the height in pixels of an item in an owner-draw list box.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Use ItemHeight to control the height of an item in a list box when the Style property is lbOwnerDrawFixed. The ItemHeight property can also affect the height of the list box itself. When the IntegralHeight property is true, the height of the list box is always an increment of the ItemHeight value.

If the Style property is lbStandard or lbOwnerDrawVariable, the value of ItemHeight is ignored.

See Also

DrawItem (see page 21328)

IntegralHeight (see page 21347)

Items (see page 21348)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

Style (see page 21354)

Using TDBListBox and TDBComboBox

47.1.6.5.26 TDBListBox.Items Property

Lists the strings that appear in the list box.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to specify the values the user can choose from. Because Items is an object of type TStrings, you can add, delete, insert, and move items using the Add, Delete, Insert, Exchange, and Move methods of the TStrings object. For example, to add each string to the listbox, write code such as the following:

```
while not CountryTbl.Eof do begin
  DBListBox1.Items.Add(CountryTbl.FieldByName('Country').AsString);
  CountryTbl.Next;
end;
DBListBox1->Items->Add("AFieldValue");
```

Note: The ItemIndex property determines which item is selected, if any.

See Also

ItemIndex

Using TDBListBox and TDBComboBox

47.1.6.5.27 TDBListBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

47.1.6.5.28 TDBListBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

47.1.6.5.29 TDBListBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

47.1.6.5.30 TDBListBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

47.1.6.5.31 TDBListBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

47.1.6.5.32 TDBListBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
```

```
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
```

```
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
```

```
    Edit1.PopupMenu := PopupMenu1;
    Edit2.PopupMenu := PopupMenu1;
  end;

  {
  These are the cut, copy, and paste OnClick events for the commands
  on the pop-up menu. Double click on the pop-up menu and add three
  TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
  The names will be automatically set to "Cut1", "Copy1", and
  "Paste1".  Then double click on the OnClick events and they will
  be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
  Note: only the selected portion of the string in the edit box will
  be copied, cut or replaced.
  }
  procedure TForm1.Copy1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
     Edit1.CopyToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
```

```
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

47.1.6.5.33 TDBListBox.ReadOnly Property

Determines if the user can change the field value using the listbox.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set ReadOnly to specify whether the listbox should be used for display purposes only. If ReadOnly is true, the listbox can only be used to display the value of the field on the current record. If ReadOnly is false, the user can use the listbox to change the field's value to one of the values in the Items (see page 8918) list.

See Also

Items (see page 8918)

Edit (see page 6932)

47.1.6.5.34 TDBListBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

47.1.6.5.35 TDBListBox.Style Property

Determines whether the list box is standard or owner-draw and whether it is virtual.

Pascal

```
property Style: TListBoxStyle;
```

C++

```
__property TListBoxStyle Style;
```

Description

Use Style to specify whether the list box is a standard list box that displays text strings, or an owner-draw list box that can display graphical images. Owner-draw list boxes let you display items in some nonstandard way. In this case, you must write the code to paint items in the list box.

In addition, Style controls whether the list box is virtual, which means that items are supplied dynamically using event handlers. By default, Style is lbStandard, meaning that the list box is not virtual and it displays each item as a string.

See Also

DrawItem (see page 21328)

ItemHeight (see page 21347)

Items (see page 21348)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

OnData (see page 21357)

OnDataFind (see page 21357)

OnDataObject (see page 21358)

47.1.6.5.36 TDBListBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

47.1.6.5.37 TDBListBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

47.1.6.5.38 TDBListBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
```

```
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

47.1.6.6 TDBListBox Events

The events of the TDBListBox class are listed here.

Published Events

	Name	Description
	OnClick (see page 8931)	Occurs when the user clicks the control.
	OnContextPopup (see page 8933)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 8934)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 8934)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 8936)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 8939)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnEndDock (see page 8941)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 8941)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 8942)	Occurs when a control receives the input focus.
	OnExit (see page 8945)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 8946)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 8949)	Occurs when a key is pressed.
	OnKeyUp (see page 8950)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 8952)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.
	OnMouseActivate (see page 8953)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 8954)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 8957)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 8957)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 8957)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 8959)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 8962)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 8962)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

47.1.6.6.1 TDBListBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This

event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;
```

```
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

47.1.6.6.2 TDBListBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by

a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

47.1.6.6.3 TDBListBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

47.1.6.6.4 TDBListBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/
```

```
// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
```

```
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

47.1.6.6.5 TDBListBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.6.6.6 TDBListBox.OnDrawItem Event

Occurs when an item in an owner-draw list box needs to be redisplayed.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

Use OnDrawItem to write a handler for drawing of the items in list boxes with the Style values lbOwnerDrawFixed, lbOwnerDrawVariable, or lbVirtualOwnerDraw. OnDrawItem occurs when the list box needs to display an item. OnDrawItem occurs only for owner-draw list boxes.

The size of the rectangle that contains the item is determined either by the ItemHeight property for fixed owner-draw list boxes or by the response to the OnMeasureItem event for variable owner-draw list boxes.

C++ Examples:

```
/*
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyList = new TStringList;
  bitmap0 = new Graphics::TBitmap;
  bitmap1 = new Graphics::TBitmap;
  bitmap2 = new Graphics::TBitmap;
//  MyList->Add("Animal");
//  MyList->Add("Flowers");
//  MyList->Add("Butterfly");
//  ListBox1.Items->AddStrings(MyList);

  ImageList1->GetBitmap(0, bitmap0);
  ListBox1->Items->AddObject("Flowers", bitmap0);
```

```
  ImageList1->GetBitmap(1, bitmap1);
  ListBox1->Items->AddObject("Animal", bitmap1);
  ImageList1->GetBitmap(2, bitmap2);
  ListBox1->Items->AddObject("Butterfly", bitmap2);
}

void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
      TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBitmap; // temporary variable for item's bitmap
  int     Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *pCanvas = ((TListBox *)Control)->Canvas;
  pCanvas->FillRect(Rect); // clear the rectangle
  pBitmap = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
  if (pBitmap)
  {
    pCanvas->BrushCopy(Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
pBitmap, Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
    Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  pCanvas->TextOut(Rect.Left + Offset, Rect.Top, ((TListBox *)Control)->Items->Strings[Index]);
}
```

Delphi Examples:

```
{
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  MyList := TStringList.Create;
  bitmap0 := TBitmap.Create;
  bitmap1 := TBitmap.Create;
  bitmap2 := TBitmap.Create;
//  MyList.Add('Animal');
//  MyList.Add('Flowers');
//  MyList.Add('Butterfly');
//  ListBox1.Items.AddStrings(MyList);

  ImageList1.GetBitmap(0, bitmap0);
  ListBox1.Items.AddObject('Flowers', bitmap0);
  ImageList1.GetBitmap(1, bitmap1);
  ListBox1.Items.AddObject('Animal', bitmap1);
  ImageList1.GetBitmap(2, bitmap2);
  ListBox1.Items.AddObject('Butterfly', bitmap2);
end;

procedure TForm1.ListBox1DrawItem(Control: TWinControl; Index: Integer; Rect:TRect; State:
TOwnerDrawState);
var
  Bitmap: TBitmap;      { temporary variable for the item's bitmap }
  Offset: Integer;      { text offset width }
begin
  Bitmap := TBitmap.Create;
  with (Control as TListBox).Canvas do  { draw on control canvas, not on the form }
  begin
    FillRect(Rect);       { clear the rectangle }
    Offset := 2;          { provide default offset }
```

```
      Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
      if Bitmap <> nil then
      begin
        BrushCopy(
          Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
          Bitmap,
          Bounds(0, 0, Bitmap.Width, Bitmap.Height),
          clRed);  {render bitmap}
        Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
      end;
      TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
  text }
    end;
  end;
```

See Also

DrawItem (see page 21328)

ItemHeight (see page 21347)

MeasureItem (see page 21335)

OnMeasureItem (see page 21360)

Style (see page 21354)

47.1.6.6.7 TDBListBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.6.6.8 TDBListBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

47.1.6.6.9 TDBListBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
```

```
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
```

```
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

47.1.6.6.10 TDBListBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

47.1.6.6.11 TDBListBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can

respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
```

```
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
```

```
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

47.1.6.6.12 TDBListBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the

OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

47.1.6.6.13 TDBListBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
```

```
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
```

```
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

47.1.6.6.14 TDBListBox.OnMeasureItem Event

Occurs when the application needs to redisplay an item in a variable height owner-draw list box.

Pascal

```
property OnMeasureItem: TMeasureItemEvent;
```

C++

```
__property TMeasureItemEvent OnMeasureItem;
```

Description

Use OnMeasureItem to write a handler to measure items to be drawn in a list box with a Style property value of lbOwnerDrawVariable.

OnMeasureItem is of type TMeasureItemEvent which contains three parameters describing the item to measure:

Control, specifying the control containing the item.

Index, specifying the index of the item in the control.

Height, specifying the height of the item.

The Index parameter identifies the position of the item in the list box.

The Height parameter should specify the height in pixels that the given item will occupy in the control. The Height parameter is passed by reference (a var parameter), which initially contains the default height of the item or the height of the item text in the control's font. The handler can set Height to a value appropriate to the contents of the item, such as the height of a graphical image to be displayed within the item.

After the OnMeasureItem event occurs, the OnDrawItem event occurs, rendering the item with the measured size.

C++ Examples:

```
/*
Here is a typical handler for an OnMeasureItem event. The
example assumes that a variable owner-draw list box already
has bitmaps associated with each of its strings. It sets the
height of the item to the height of the associated bitmap if
that height is greater than the default height
(ListBox1.ItemHeight).  Note that TWinControl can also be
used in place of TWidgetControl on Windows.
```

```
*/
void __fastcall TForm1::ListBox1MeasureItem(TWinControl *Control, int Index,
      int &Height)
{
  Graphics::TBitmap *bitmap = (Graphics::TBitmap *)((TListBox
*)Control)->Items->Objects[Index];
  if (bitmap && bitmap->Height > Height)
    Height = bitmap->Height;
}

void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
      TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBitmap; // temporary variable for item's bitmap
  int     Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *pCanvas = ((TListBox *)Control)->Canvas;
  pCanvas->FillRect(Rect); // clear the rectangle
  pBitmap = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
  if (pBitmap)
  {
    pCanvas->BrushCopy(Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
pBitmap, Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
    Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  pCanvas->TextOut(Rect.Left + Offset, Rect.Top, ((TListBox *)Control)->Items->Strings[Index]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Graphics::TBitmap *bitmap0 = new Graphics::TBitmap;
  Graphics::TBitmap *bitmap1 = new Graphics::TBitmap;
  Graphics::TBitmap *bitmap2 = new Graphics::TBitmap;
  bitmap0->LoadFromFile("../littleB_64.bmp");
//  ImageList1->GetBitmap(0, bitmap0);
  ListBox1->Items->AddObject("Butterfly", bitmap0);
  ImageList1->GetBitmap(3, bitmap1);
  ListBox1->Items->AddObject("Animal", bitmap1); // no bitmap, see if it leaves this height at
64.
  ImageList1->GetBitmap(4, bitmap2);
  ListBox1->Items->AddObject("Sunflower", bitmap2); // no bitmap, see if it leaves this height
at 64.

}
```

See Also

DrawItem (see page 21328)

ItemHeight (see page 21347)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

Style (see page 21354)

47.1.6.6.15 TDBListBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.6.6.16 TDBListBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
```

```
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.6.6.17 TDBListBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

47.1.6.6.18 TDBListBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

47.1.6.6.19 TDBListBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
```

```
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.6.6.20 TDBListBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
```

```
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
```

```
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

47.1.6.6.21 TDBListBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

47.1.6.6.22 TDBListBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

47.1.7 TDBLookupComboBox Class

TDBLookupComboBox represents a combo box that identifies a set of field values in one dataset with a corresponding set of values from another dataset.

Class Hierarchy

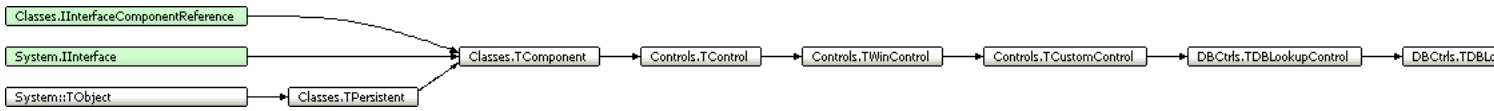

Pascal

```
TDBLookupComboBox = class(TDBLookupControl);
```

C++

```
class TDBLookupComboBox : public TDBLookupControl;
```

File

DBCtrls

Description

Use TDBLookupComboBox to provide the user with a convenient drop-down list of lookup items for filling in fields that require data from another dataset.

If TDBLookupComboBox is linked to a lookup field component, it automatically reads the relationship between the field value and the lookup values in the lookup dataset from the field component. The relationship between field values and the corresponding

values in the lookup dataset can also be explicitly set using the properties of the lookup combo box when the combo box is not linked to a lookup field component.

See Also

TComboBox (see page 21064)

TDBLookupListBox (see page 9070)

Displaying and Editing Data in List and Combo Boxes

Displaying and Editing Data in Lookup List and Combo Boxes

Using Common Data Control Features

47.1.7.1 TDBLookupComboBox Members

The following tables list the members exposed by TDBLookupComboBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.

	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanModify (see page 9057)	Indicates whether the lookup control has rights to modify data.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 8976)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.

	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
V	GetBorderSize (see page 9058)	This is GetBorderSize, a member of class TDBLookupControl.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.

	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
V	GetTextHeight (see page 9058)	Indicates how many pixels are required for one line of text in the list.
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
V	KeyDown (see page 8977)	Respond to key press events.
V	KeyPress (see page 8978)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	KeyValueChanged (see page 8978)	Updates the control when the value of the protected KeyValue property changes.
V	ListLinkDataChanged (see page 8979)	Updates the control when its data binding changes.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	LocateKey (see page 9059)	Locates the value specified by KeyValue (see page 9065) in the lookup dataset.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 8979)	OnMouseDown event dispatcher.
V	MouseMove (see page 8980)	Respond to mouse moving over control area..
V	MouseUp (see page 8980)	OnMouseUp event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 9059)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 8981)	Renders the image of a custom control.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	ProcessSearchKey (see page 9060)	Adds a character to the protected SearchText (see page 9069) property and updates the selection to the first matching list value.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.

	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
V	SelectKeyValue (see page 9060)	Updates the value of the control so that it reflects a new key field value.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateDataFields (see page 9061)	Updates the control when the DataField (see page 9062) property changes.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateListFields (see page 8982)	Updates the control when the list of lookup values changes.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.

	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
R	DataLink (see page 9063)	Helps the lookup control manage its link to the data source and respond to data events.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
R	HasFocus (see page 9064)	Indicates whether the lookup control has input focus.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	ListActive (see page 9065)	Indicates whether the ListFields (see page 9067) property is currently being updated.
R	ListFields (see page 9067)	Lists the field components whose values are displayed in the lookup control.
R	ListLink (see page 9067)	Identifies the TListSourceLink (see page 9498) object that coordinates data communication with the lookup dataset.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	SearchText (see page 9069)	This is SearchText, a member of class TDBLookupControl.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	CloseUp (see page 8976)	Closes an opened or "dropped-down" list.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 8975)	Creates and initializes an instance of TDBLookupComboBox (see page 8963).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 9056)	Destroys an instance of TDBLookupControl (see page 9044).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
V	DropDown (see page 8977)	Opens or "drops down" the lookup list so that the user can choose a lookup value.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
V	ExecuteAction (see page 8977)	Invokes an action with the TDBLookupComboBox (see page 8963) as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.

	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 8981)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
V	UpdateAction (see page 8982)	Updates an action to reflect the current state of the TDBLookupComboBox (see page 8963).
	UpdateControlState (see page 6186)	Respond to state change.
V	UseRightToLeftAlignment (see page 8982)	Specifies whether the control's alignment is in a right-to-left mode.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.

R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Field (see page 9064)	Identifies the TField object the lookup control represents.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	KeyValue (see page 9000)	Represents the common value of the KeyField field and the DataField field.
R	ListVisible (see page 9002)	Specifies whether the lookup list is open or "dropped-down".
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
R	Text (see page 9013)	Indicates the lookup value that corresponds to the value of the field on the current record.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 9015)	Occurs when the user clicks the control.
	OnCloseUp (see page 9017)	Occurs immediately after an opened or "dropped-down" list is closed.
	OnContextPopup (see page 9017)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 9018)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 9020)	Occurs when the user drags an object over a control.
	OnDropDown (see page 9023)	Occurs immediately before the lookup list is opened.
	OnEndDock (see page 9023)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 9024)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 9025)	Occurs when a control receives the input focus.
	OnExit (see page 9027)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 9029)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 9031)	Occurs when a key is pressed.
	OnKeyUp (see page 9033)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 9034)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 9035)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 9038)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 9038)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 9038)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 9040)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 9043)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 9043)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 8984)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 8987)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 8988)	Specifies which edges of the control are beveled.
	BevelInner (see page 8988)	Specifies the cut of the inner bevel.
	BevelKind (see page 8989)	Specifies the control's bevel style.
	BevelOuter (see page 8989)	Specifies the cut of the outer bevel.
	BevelWidth (see page 8990)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 8990)	Specifies the bi-directional mode for the control.
	Color (see page 8991)	Specifies the background color of the control.
	Constraints (see page 8991)	Specifies the size constraints for the control.
	Ctl3D (see page 8992)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DataField (see page 8992)	Identifies the field whose value is represented by the lookup control.
	DataSource (see page 8993)	Links the lookup control to the dataset that contains the DataField.
	DragCursor (see page 8993)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 8994)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 8994)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownAlign (see page 8996)	Specifies how the drop-down list is aligned relative to its edit box.
	DropDownRows (see page 8996)	Specifies the number of items displayed in the drop-down list of the TDBLookupComboBox (see page 8963).
	DropDownWidth (see page 8996)	Specifies the width, in pixels, of the drop-down list.
	Enabled (see page 8997)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 8997)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 8998)	Determines the behavior of the input method editor (IME).
	ImeName (see page 8999)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	KeyField (see page 9000)	Identifies the field in the ListSource dataset that must match the value of the DataField field.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	ListField (see page 9001)	Identifies the field or fields whose values are displayed in the lookup control.
	ListFieldIndex (see page 9001)	Specifies which field from the ListField property is used for incremental searching.
	ListSource (see page 9002)	Identifies a data source for the data displayed in the lookup control.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NullValueKey (see page 9003)	Specifies the key combination that assigns a blank value to the associated field.
	ParentBiDiMode (see page 9003)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 9003)	Determines where a control looks for its color information.
	ParentCtl3D (see page 9004)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 9004)	Determines where a control looks for its font information.
	ParentShowHint (see page 9005)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 9005)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 9010)	This is ReadOnly, a member of class TDBLookupComboBox.

	ShowHint (see page 9010)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 9012)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 9012)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 9013)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

47.1.7.2 TDBLookupComboBox.Create Constructor

Creates and initializes an instance of TDBLookupComboBox (see page 8963).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDBLookupComboBox(TComponent * AOwner);
```

Description

Call Create to instantiate a lookup combo box at runtime. TDBLookupComboBox (see page 8963) objects placed on forms at design time are created automatically.

AOwner is the component that is responsible for freeing the lookup combo box instance. It becomes the value of the Owner property.

See Also

Destroy (see page 9056)

TFieldDataLink (see page 9487)

Owner (see page 2807)

47.1.7.3 TDBLookupComboBox Methods

The methods of the TDBLookupComboBox class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 8976)	Initializes a window-creation parameter data structure.
	KeyDown (see page 8977)	Respond to key press events.
	KeyPress (see page 8978)	Respond to keyboard input.
	KeyValueChanged (see page 8978)	Updates the control when the value of the protected KeyValue property changes.
	ListLinkDataChanged (see page 8979)	Updates the control when its data binding changes.
	MouseDown (see page 8979)	OnMouseDown event dispatcher.
	MouseMove (see page 8980)	Respond to mouse moving over control area..
	MouseUp (see page 8980)	OnMouseUp event dispatcher.
	Paint (see page 8981)	Renders the image of a custom control.
	UpdateListFields (see page 8982)	Updates the control when the list of lookup values changes.

Public Methods

	Name	Description
	CloseUp (see page 8976)	Closes an opened or "dropped-down" list.

	DropDown (see page 8977)	Opens or "drops down" the lookup list so that the user can choose a lookup value.
	ExecuteAction (see page 8977)	Invokes an action with the TDBLookupComboBox (see page 8963) as its target.
	SetBounds (see page 8981)	Sets the windowed control's boundary properties all at once.
	UpdateAction (see page 8982)	Updates an action to reflect the current state of the TDBLookupComboBox (see page 8963).
	UseRightToLeftAlignment (see page 8982)	Specifies whether the control's alignment is in a right-to-left mode.

47.1.7.3.1 TDBLookupComboBox.CloseUp Method

Closes an opened or "dropped-down" list.

Pascal

```
procedure CloseUp(Accept: Boolean); virtual;
```

C++

```
virtual __fastcall CloseUp(Boolean Accept);
```

Description

Call CloseUp to programmatically close the list of the lookup combo box. The Accept parameter determines whether to modify the field value with the value that corresponds to the selected value in the lookup list.

```
DBLookupComboBox1.CloseUp(True);
DBLookupComboBox1->CloseUp(true);
```

See Also

DataField (see page 9062)

OnCloseUp (see page 9017)

47.1.7.3.2 TDBLookupComboBox.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

47.1.7.3.3 TDBLookupComboBox.DropDown Method

Opens or "drops down" the lookup list so that the user can choose a lookup value.

Pascal

```
procedure DropDown; virtual;
```

C++

```
virtual __fastcall DropDown();
```

Description

Call DropDown to programmatically open the lookup list. Before the list is displayed, an OnDropDown (see page 9023) event is generated.

See Also

OnDropDown (see page 9023)

47.1.7.3.4 TDBLookupComboBox.ExecuteAction Method

Invokes an action with the TDBLookupComboBox (see page 8963) as its target.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

ExecuteAction is called automatically when the user invokes an action with the TDBLookupComboBox (see page 8963) as a target. The Action parameter specifies the action that was invoked. ExecuteAction returns trueif the action was successfully dispatched, and false if the TDBLookupComboBox (see page 8963) could not handle the action.

See Also

UpdateAction (see page 8982)

Execute (see page 2712)

HandlesTarget (see page 2713)

47.1.7.3.5 TDBLookupComboBox.KeyDown Method

Respond to key press events.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-down message (WM_KEYDOWN) from Windows, its message handler calls the DoKeyDown method. If DoKeyDown determines that the control should, in fact, process the character, it calls KeyDown, passing

the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

See Also

KeyUp (see page 6164)

OnKeyDown (see page 6225)

DoKeyDown (see page 6145)

DoKeyPress (see page 6146)

DoKeyUp (see page 6146)

47.1.7.3.6 TDBLookupComboBox.KeyPress Method

Respond to keyboard input.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

When a windowed control receives a key-press message (WM_CHAR) from Windows, its message handler calls the DoKeyPress method. If DoKeyPress determines that the control should, in fact, process the character, it calls KeyPress, passing the key code in the Key parameter.

KeyPress calls any event handler attached to the OnKeyPress event. Override KeyPress to provide other responses in addition to the event-handler call.

Either KeyPress or the OnKeyPress event handler it calls can suppress further processing of a character by setting the Key parameter to zero.

Note: The Key parameter is the character represented by the key that is pressed, not a Windows virtual key code.

See Also

KeyDown (see page 6163)

KeyUp (see page 6164)

OnKeyPress (see page 6227)

47.1.7.3.7 TDBLookupComboBox.KeyValueChanged Method

Updates the control when the value of the protected KeyValue property changes.

Pascal

```
procedure KeyValueChanged; override;
```

C++

```
virtual __fastcall KeyValueChanged();
```

Description

KeyValueChanged is called automatically when the value of the KeyValue property changes. It allows the lookup control to make any necessary adjustments to display the new value in the list.

As implemented in TDBLookupControl, KeyValueChanged does nothing. Descendant classes override this method to update the display properties of the control to reflect the new value.

See Also

KeyValue (see page 9065)

47.1.7.3.8 TDBLookupComboBox.ListLinkDataChanged Method

Updates the control when its data binding changes.

Pascal

```
procedure ListLinkDataChanged; override;
```

C++

```
virtual __fastcall ListLinkDataChanged();
```

Description

ListLinkDataChanged is called automatically when the data displayed in the lookup list changes for any reason.

As implemented in TDBLookupControl, ListLinkDataChanged does nothing. Descendant classes override this method to update any internal properties that depend on the current list.

See Also

ListField (see page 9066)

47.1.7.3.9 TDBLookupComboBox.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN,

WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove (see page 5773)

MouseUp (see page 5774)

OnMouseDown (see page 5862)

47.1.7.3.10 TDBLookupComboBox.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown (see page 5773)

MouseUp (see page 5774)

OnMouseMove (see page 5865)

47.1.7.3.11 TDBLookupComboBox.MouseUp Method

OnMouseUp event dispatcher.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user releases a previously pressed mouse button while the cursor's hotspot is over the control.

A control calls MouseUp in response to any of the Windows mouse-up messages (WM_LBUTTONUP, WM_MBUTTONUP, WM_RBUTTONUP), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

See Also

MouseDown (see page 5773)

MouseMove (see page 5773)

OnMouseUp (see page 5867)

47.1.7.3.12 TDBLookupComboBox.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas (see page 5921)

PaintWindow (see page 5920)

GetDeviceContext (see page 6155)

ClipRect (see page 14486)

47.1.7.3.13 TDBLookupComboBox.SetBounds Method

Sets the windowed control's boundary properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to change all of the control's boundary properties at once. The same effect can be achieved by setting the Left, Top, Width, and Height properties separately. By setting all four properties at once, SetBounds ensures that the control will not repaint between changes

Specify the values for the Left, Top, Width, and Height properties as the value of the ALeft, ATop, AWidth, and AHeight parameters, respectively.

See Also

Height (see page 5825)

Left (see page 5830)

Top (see page 5844)

Width (see page 5849)

47.1.7.3.14 TDBLookupComboBox.UpdateAction Method

Updates an action to reflect the current state of the TDBLookupComboBox (see page 8963).

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

UpdateAction is called automatically when the application is idle to give the TDBLookupComboBox (see page 8963) an opportunity to update associated actions. This allows actions to be grayed, checked, or to otherwise reflect the status of the TDBLookupComboBox (see page 8963) they target.

The Action parameter specifies the action component that should be updated. UpdateAction returns trueif the action component now reflects the state of the component, and false if it did not know how to update the action.

See Also

ExecuteAction (see page 8977)

UpdateTarget (see page 2715)

HandlesTarget (see page 2713)

47.1.7.3.15 TDBLookupComboBox.UpdateListFields Method

Updates the control when the list of lookup values changes.

Pascal

```
procedure UpdateListFields; override;
```

C++

```
virtual __fastcall UpdateListFields();
```

Description

UpdateListFields is called automatically when the list of lookup values changes. It synchronizes all the properties of the control so that they reflect the new list of values.

See Also

ListField (see page 9066)

KeyField (see page 9064)

47.1.7.3.16 TDBLookupComboBox.UseRightToLeftAlignment Method

Specifies whether the control's alignment is in a right-to-left mode.

Pascal

```
function UseRightToLeftAlignment: Boolean; override;
```

C++

```
virtual __fastcall Boolean UseRightToLeftAlignment();
```

Description

Call UseRightToLeftAlignment to determine whether the TDBLookupComboBox (see page 8963)'s alignment is in a right-to-left mode. This information can be used when painting the TDBLookupComboBox (see page 8963) data or when responding to mouse events.

UseRightToLeftAlignment returns true for middle east locales if the BiDiMode (see page 8990) property is bdRightToLeft. Otherwise, it returns false.

See Also

UseRightToLeftReading (see page 5790)

BiDiMode (see page 5803)

47.1.7.4 TDBLookupComboBox Properties

The properties of the TDBLookupComboBox class are listed here.

Public Properties

	Name	Description
	KeyValue (see page 9000)	Represents the common value of the KeyField field and the DataField field.
R	ListVisible (see page 9002)	Specifies whether the lookup list is open or "dropped-down".
R	Text (see page 9013)	Indicates the lookup value that corresponds to the value of the field on the current record.

Published Properties

	Name	Description
	Align (see page 8984)	Determines how the control aligns within its container (parent control).
	Anchors (see page 8987)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 8988)	Specifies which edges of the control are beveled.
	BevelInner (see page 8988)	Specifies the cut of the inner bevel.
	BevelKind (see page 8989)	Specifies the control's bevel style.
	BevelOuter (see page 8989)	Specifies the cut of the outer bevel.
	BevelWidth (see page 8990)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 8990)	Specifies the bi-directional mode for the control.
	Color (see page 8991)	Specifies the background color of the control.
	Constraints (see page 8991)	Specifies the size constraints for the control.
	Ctl3D (see page 8992)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DataField (see page 8992)	Identifies the field whose value is represented by the lookup control.
	DataSource (see page 8993)	Links the lookup control to the dataset that contains the DataField.
	DragCursor (see page 8993)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 8994)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 8994)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownAlign (see page 8996)	Specifies how the drop-down list is aligned relative to its edit box.
	DropDownRows (see page 8996)	Specifies the number of items displayed in the drop-down list of the TDBLookupComboBox (see page 8963).
	DropDownWidth (see page 8996)	Specifies the width, in pixels, of the drop-down list.
	Enabled (see page 8997)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 8997)	Controls the attributes of text written on or in the control.
	ImeMode (see page 8998)	Determines the behavior of the input method editor (IME).

	ImeName (see page 8999)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	KeyField (see page 9000)	Identifies the field in the ListSource dataset that must match the value of the DataField field.
	ListField (see page 9001)	Identifies the field or fields whose values are displayed in the lookup control.
	ListFieldIndex (see page 9001)	Specifies which field from the ListField property is used for incremental searching.
	ListSource (see page 9002)	Identifies a data source for the data displayed in the lookup control.
	NullValueKey (see page 9003)	Specifies the key combination that assigns a blank value to the associated field.
	ParentBiDiMode (see page 9003)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 9003)	Determines where a control looks for its color information.
	ParentCtl3D (see page 9004)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 9004)	Determines where a control looks for its font information.
	ParentShowHint (see page 9005)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 9005)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 9010)	This is ReadOnly, a member of class TDBLookupComboBox.
	ShowHint (see page 9010)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 9012)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 9012)	Determines if the user can tab to a control.
	Visible (see page 9013)	Determines whether the component appears onscreen.

47.1.7.4.1 TDBLookupComboBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
```

```
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

47.1.7.4.2 TDBLookupComboBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

47.1.7.4.3 TDBLookupComboBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

47.1.7.4.4 TDBLookupComboBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

47.1.7.4.5 TDBLookupComboBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

47.1.7.4.6 TDBLookupComboBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

47.1.7.4.7 TDBLookupComboBox.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

47.1.7.4.8 TDBLookupComboBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

47.1.7.4.9 TDBLookupComboBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

47.1.7.4.10 TDBLookupComboBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

47.1.7.4.11 TDBLookupComboBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

47.1.7.4.12 TDBLookupComboBox.DataField Property

Identifies the field whose value is represented by the lookup control.

Pascal

```
property DataField: WideString;
```

C++

```
__property BSTR DataField;
```

Description

Use DataField to bind the lookup control to a field in the dataset specified by the DataSource property. The DataField is the field whose value can be set by the lookup control, not the field which supplies the lookup values that are displayed by the lookup control.

If DataField specifies a lookup field component, there is no need to set the KeyField or ListField properties, as the field component contains all the information about the relationship between the data field value and the lookup value.

See Also

DataSource (see page 9063)

KeyField (see page 9064)

ListField (see page 9066)

Displaying and Editing Data in Lookup List and Combo Boxes

47.1.7.4.13 TDBLookupComboBox.DataSource Property

Links the lookup control to the dataset that contains the DataField.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Use DataSource to specify the data source component that identifies the dataset the lookup control represents. The lookup control represents a field in one table by displaying the value of a corresponding field in another table. DataSource links to the dataset for the field the control represents, as opposed to the ListSource, which links the lookup control to the lookup table that holds the field that is actually displayed. If the lookup control is used to edit data, the field that actually gets changed is the one in the DataSource.

See Also

DataField (see page 9062)

ListSource (see page 9068)

Displaying and Editing Data in Lookup List and Combo Boxes

47.1.7.4.14 TDBLookupComboBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

47.1.7.4.15 TDBLookupComboBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

47.1.7.4.16 TDBLookupComboBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
```

```
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

47.1.7.4.17 TDBLookupComboBox.DropDownAlign Property

Specifies how the drop-down list is aligned relative to its edit box.

Pascal

```
property DropDownAlign: TDropDownAlign;
```

C++

```
__property TDropDownAlign DropDownAlign;
```

Description

Use DropDownAlign to specify the position of the drop-down list relative to the edit region of the lookup combo box. The drop down list can be aligned to the left or right, or centered with the edit box.

See Also

DropDownWidth (see page 8996)

DropDownRows (see page 8996)

OnDropDown (see page 9023)

47.1.7.4.18 TDBLookupComboBox.DropDownRows Property

Specifies the number of items displayed in the drop-down list of the TDBLookupComboBox (see page 8963).

Pascal

```
property DropDownRows: Integer;
```

C++

```
__property int DropDownRows;
```

Description

Use DropDownRows to specify how many rows appear in the drop down list of the combo box. If there are more lookup items than DropDownRows, the lookup list box displays a scroll bar.

The default value for DropDownRows is 7.

See Also

DropDownWidth (see page 8996)

DropDownAlign (see page 8996)

OnDropDown (see page 9023)

Displaying and Editing Data in Lookup List and Combo Boxes

47.1.7.4.19 TDBLookupComboBox.DropDownWidth Property

Specifies the width, in pixels, of the drop-down list.

Pascal

```
property DropDownWidth: Integer;
```

C++

```
__property int DropDownWidth;
```

Description

Use DropDownWidth to customize the width of the drop-down list. If DropDownWidth is 0 (the default), the drop-down list is the same width as the combo box.

DropDownWidth is useful when displaying multiple lookup fields, and therefore, multiple columns in the database lookup combo box.

See Also

DropDownRows (see page 8996)

DropDownAlign (see page 8996)

OnDropDown (see page 9023)

47.1.7.4.20 TDBLookupComboBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

47.1.7.4.21 TDBLookupComboBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name,

Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

47.1.7.4.22 TDBLookupComboBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

47.1.7.4.23 TDBLookupComboBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

47.1.7.4.24 TDBLookupComboBox.KeyField Property

Identifies the field in the ListSource dataset that must match the value of the DataField field.

Pascal

```
property KeyField: string;
```

C++

```
__property AnsiString KeyField;
```

Description

Use KeyField to link the ListSource of the lookup control to the DataSource. Although the name of the field specified as the KeyField does not have to be the same as the name of the field specified as the DataField, the two fields must have the same values.

After specifying the ListField, choose which field the lookup control will actually display with the ListField property.

If the DataField field is a lookup field, don't specify anything for KeyField or ListField; lookup controls automatically use the data field's LookupKeyFields property for KeyField.

See Also

DataField (see page 9062)

DataSource (see page 9063)

KeyValue (see page 9065)

ListField (see page 9066)

ListSource (see page 9068)

LookupKeyFields (see page 7172)

Displaying and Editing Data in Lookup List and Combo Boxes

47.1.7.4.25 TDBLookupComboBox.KeyValue Property

Represents the common value of the KeyField field and the DataField field.

Pascal

```
property KeyValue: Variant;
```

C++

```
__property Variant KeyValue;
```

Description

Use KeyField to determine the value represented by the lookup control (not the value displayed by the lookup control). When KeyValue is set, the lookup control attempts to find a record from the ListSource's dataset where the value of KeyField matches KeyValue. If such a match is found, the lookup control displays the value of ListField on that record.

See Also

DataField (see page 9062)

KeyField (see page 9064)

ListField (see page 9066)

ListSource (see page 9068)

LookupKeyFields (see page 7172)

47.1.7.4.26 TDBLookupComboBox.ListField Property

Identifies the field or fields whose values are displayed in the lookup control.

Pascal

```
property ListField: string;
```

C++

```
__property AnsiString ListField;
```

Description

The lookup control represents a field in one table by displaying the value of one or more corresponding fields in another table. ListField is the name of the field or fields in the lookup table that are actually displayed, as opposed to the DataField, which is the field in the DataSource that the lookup control actually represents, or the KeyField, which is the field in the lookup table with the same value as the DataField.

To fully specify the list fields, both a dataset for the lookup table and the fields within that dataset must be defined. The ListSource property of the lookup control specifies the dataset for the lookup table.

ListField can represent more than one field. Separate multiple field names with semicolons.

Before specifying ListField, specify the link between the two datasets using the KeyField property. If ListField is not set, lookup controls display KeyField field values by default. If the DataField field is a lookup field, don't specify anything for ListField; the data controls automatically use the lookup field's LookupResultField property as ListField.

See Also

DataField (see page 9062)

DataSource (see page 9063)

KeyField (see page 9064)

KeyValue (see page 9065)

ListFieldIndex (see page 9067)

ListSource (see page 9068)

Displaying and Editing Data in Lookup List and Combo Boxes

47.1.7.4.27 TDBLookupComboBox.ListFieldIndex Property

Specifies which field from the ListField property is used for incremental searching.

Pascal

```
property ListFieldIndex: Integer;
```

C++

```
__property int ListFieldIndex;
```

Description

When the ListField property specifies more than one field, use ListFieldIndex to specify which of those fields is the one to use for incremental searches. For the TDBLookupComboBox object, ListFieldIndex also determines which field appears in the edit region of the combo box.

ListFieldIndex allows the most important value to appear in a position other than the first when all the values of the ListField

fields are displayed. The value of ListFieldIndex must be less than the number of fields specified by the ListField property.

See Also

ListField (see page 9066)

ListFields (see page 9067)

47.1.7.4.28 TDBLookupComboBox.ListSource Property

Identifies a data source for the data displayed in the lookup control.

Pascal

```
property ListSource: TDataSource;
```

C++

```
__property TDataSource ListSource;
```

Description

Set LookupSource to the data source that contains the KeyField and ListField fields. If the DataField field is a lookup field, don't specify anything for ListSource; data controls automatically use the lookup field's LookupDataSet property to create a data source.

See Also

DataField (see page 9062)

DataSource (see page 9063)

KeyField (see page 9064)

ListField (see page 9066)

ListFieldIndex (see page 9067)

LookupDataSet (see page 7171)

Displaying and Editing Data in Lookup List and Combo Boxes

47.1.7.4.29 TDBLookupComboBox.ListVisible Property

Specifies whether the lookup list is open or "dropped-down".

Pascal

```
property ListVisible: Boolean;
```

C++

```
__property Boolean ListVisible;
```

Description

Read ListVisible to determine whether the list of lookup values is currently in the open (dropped-down) position. If ListVisible is true, the list is open; if ListVisible is false, the list is closed.

See Also

OnDropDown (see page 9023)

47.1.7.4.30 TDBLookupComboBox.NullValueKey Property

Specifies the key combination that assigns a blank value to the associated field.

Pascal

```
property NullValueKey: TShortCut;
```

C++

```
__property TShortCut NullValueKey;
```

Description

Set NullValueKey to let users enter a "blank" value into the associated field. When the user types the key combination that this shortcut represents, the associated field is cleared.

Tip: Use the ShortCut function or the TextToShortCut function to create a value to assign to NullValueKey.

See Also

DataField (see page 9062)

Clear (see page 7126)

TextToShortCut (see page 16196)

ShortCut (see page 16193)

47.1.7.4.31 TDBLookupComboBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

47.1.7.4.32 TDBLookupComboBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

47.1.7.4.33 TDBLookupComboBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

47.1.7.4.34 TDBLookupComboBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform

appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

47.1.7.4.35 TDBLookupComboBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

47.1.7.4.36 TDBLookupComboBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse

button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
```

```
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
```

```
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
```

```
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
```

```
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

47.1.7.4.37 TDBLookupComboBox.ReadOnly Property

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

This is ReadOnly, a member of class TDBLookupComboBox.

47.1.7.4.38 TDBLookupComboBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

47.1.7.4.39 TDBLookupComboBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

47.1.7.4.40 TDBLookupComboBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

47.1.7.4.41 TDBLookupComboBox.Text Property

Indicates the lookup value that corresponds to the value of the field on the current record.

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

Inspect Text to determine the currently selected lookup value. Text is the value that appears in the edit region of the lookup combo box. The user can change the value of Text by entering a new value if ReadOnly (see page 9010) is false and the dataset is in edit mode.

See Also

Edit (see page 6932)

readonly (see page 9069)

47.1.7.4.42 TDBLookupComboBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
```

```
      TTabSheet *pPage = new TTabSheet(this);
      pPage->PageControl = PageControl1;
      pPage->Caption = AnsiString("Page") + IntToStr(i);
      TEdit *pEdit = new TEdit(this);
      pEdit->Parent = pPage;
      pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
      pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
    }
    PageControl1Change(Sender);
  }
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

47.1.7.5 TDBLookupComboBox Events

The events of the TDBLookupComboBox class are listed here.

Published Events

	Name	Description
	OnClick (see page 9015)	Occurs when the user clicks the control.
	OnCloseUp (see page 9017)	Occurs immediately after an opened or "dropped-down" list is closed.
	OnContextPopup (see page 9017)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 9018)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 9020)	Occurs when the user drags an object over a control.
	OnDropDown (see page 9023)	Occurs immediately before the lookup list is opened.
	OnEndDock (see page 9023)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 9024)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 9025)	Occurs when a control receives the input focus.
	OnExit (see page 9027)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 9029)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 9031)	Occurs when a key is pressed.
	OnKeyUp (see page 9033)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 9034)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 9035)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 9038)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 9038)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 9038)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 9040)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 9043)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 9043)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

47.1.7.5.1 TDBLookupComboBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check

box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.

- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
```

```
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

47.1.7.5.2 TDBLookupComboBox.OnCloseUp Event

Occurs immediately after an opened or "dropped-down" list is closed.

Pascal

```
property OnCloseUp: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCloseUp;
```

Description

Write an OnCloseUp event handler to respond when the combo box list is closed. When the list is closed, the value that corresponds to the selected lookup value is assigned to the field. The list can be closed by the user or by calling the CloseUp (see page 8976) method.

```
procedure TForm1.DBLookupComboBox1CloseUp(Sender: TObject);
begin
  StatusBar1.SimpleText := DBLookupComboBox1.Text;
end;
```

See Also

CloseUp (see page 8976)

OnDropDown (see page 9023)

47.1.7.5.3 TDBLookupComboBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

47.1.7.5.4 TDBLookupComboBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/
```

```
// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

47.1.7.5.5 TDBLookupComboBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.7.5.6 TDBLookupComboBox.OnDropDown Event

Occurs immediately before the lookup list is opened.

Pascal

```
property OnDropDown: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDropDown;
```

Description

Write an OnDropDown event handler to take specific action before the lookup list is displayed to the user. The list can be opened by the user or by calling the DropDown (see page 8977) method.

```
procedure TForm1.DBLookupComboBox1DropDown(Sender: TObject);
begin
  StatusBar1.SimpleText := '';
end;
```

See Also

DropDown (see page 8977)

OnCloseUp (see page 9017)

47.1.7.5.7 TDBLookupComboBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.7.5.8 TDBLookupComboBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

47.1.7.5.9 TDBLookupComboBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
```

```
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
```

```
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

47.1.7.5.10 TDBLookupComboBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK

button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

47.1.7.5.11 TDBLookupComboBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
```

```
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
```

```
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

47.1.7.5.12 TDBLookupComboBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

47.1.7.5.13 TDBLookupComboBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
```

```
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

47.1.7.5.14 TDBLookupComboBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.7.5.15 TDBLookupComboBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
```

```
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.7.5.16 TDBLookupComboBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

47.1.7.5.17 TDBLookupComboBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

47.1.7.5.18 TDBLookupComboBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
```

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.7.5.19 TDBLookupComboBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

47.1.7.5.20 TDBLookupComboBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

47.1.7.5.21 TDBLookupComboBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

47.1.8 TDBLookupControl Class

TDBLookupControl is the base class for controls that provide the user with a list of lookup items for filling in fields that require data from another dataset.

Class Hierarchy

Pascal

```
TDBLookupControl = class(TCustomControl);
```

C++

```
class TDBLookupControl : public TCustomControl;
```

File

DBCtrls

Description

Use TDBLookupControl as a base class when defining controls that permit the user to set a field value by selecting a corresponding value from another dataset. The TDBLookupComboBox (see page 8963) and TDBLookupListBox (see page 9070) controls are both descended from the TDBLookupControl and provide the user with automatic lookup capabilities that are implemented in the TDBLookupControl object.

Do not create instances of TDBLookupControl. Instead, instantiate a descendant of TDBLookupControl to allow the user to set field values from a set of lookup values.

See Also

TDBLookupComboBox (see page 8963)

TDBLookupListBox (see page 9070)

47.1.8.1 TDBLookupControl Members

The following tables list the members exposed by TDBLookupControl.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanModify (see page 9057)	Indicates whether the lookup control has rights to modify data.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorderSize (see page 9058)	This is GetBorderSize, a member of class TDBLookupControl.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.

	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTextHeight (see page 9058)	Indicates how many pixels are required for one line of text in the list.
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	KeyValueChanged (see page 9058)	Updates the control when the value of the protected KeyValue (see page 9065) property changes.
	ListLinkDataChanged (see page 9058)	Updates the control when its data binding changes.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	LocateKey (see page 9059)	Locates the value specified by KeyValue (see page 9065) in the lookup dataset.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 9059)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 5920)	Renders the image of a custom control.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.

	ProcessSearchKey (see page 9060)	Adds a character to the protected SearchText (see page 9069) property and updates the selection to the first matching list value.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectKeyValue (see page 9060)	Updates the value of the control so that it reflects a new key field value.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateDataFields (see page 9061)	Updates the control when the DataField (see page 9062) property changes.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateListFields (see page 9061)	Updates the control when the list of lookup values changes.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DataField (see page 9062)	Identifies the field whose value is represented by the lookup control.
R	DataLink (see page 9063)	Helps the lookup control manage its link to the data source and respond to data events.
	DataSource (see page 9063)	Links the lookup control to the dataset that contains the DataField (see page 9062).
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
R	HasFocus (see page 9064)	Indicates whether the lookup control has input focus.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	KeyField (see page 9064)	Identifies the field in the ListSource (see page 9068) dataset that must match the value of the DataField (see page 9062) field.
	KeyValue (see page 9065)	Represents the common value of the KeyField (see page 9064) field and the DataField (see page 9062) field.

R	ListActive (see page 9065)	Indicates whether the ListFields (see page 9067) property is currently being updated.
	ListField (see page 9066)	Identifies the field or fields whose values are displayed in the lookup control.
	ListFieldIndex (see page 9067)	Specifies which field from the ListField (see page 9066) property is used for incremental searching.
R	ListFields (see page 9067)	Lists the field components whose values are displayed in the lookup control.
R	ListLink (see page 9067)	Identifies the TListSourceLink (see page 9498) object that coordinates data communication with the lookup dataset.
	ListSource (see page 9068)	Identifies a data source for the data displayed in the lookup control.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	NullValueKey (see page 9068)	Specifies the key combination that assigns a blank value to the associated field.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 9069)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 9069)	This is ReadOnly, a member of class TDBLookupControl.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	SearchText (see page 9069)	This is SearchText, a member of class TDBLookupControl.
	TabStop (see page 9070)	Determines if the user can tab to a control.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.

	Create (see page 9055)	Creates an instance of TDBLookupControl (see page 9044).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 9056)	Destroys an instance of TDBLookupControl (see page 9044).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 9057)	Invokes an action with the TDBLookupControl (see page 9044) as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
V	UpdateAction (see page 9060)	Updates an action to reflect the current state of the TDBLookupControl (see page 9044).

	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Field (see page 9064)	Identifies the TField object the lookup control represents.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.

R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

47.1.8.2 TDBLookupControl.Create Constructor

Creates an instance of TDBLookupControl (see page 9044).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDBLookupControl(TComponent * AOwner);
```

Description

Call Create to instantiate a lookup control at runtime. For lookup controls placed on forms or data modules at design time, Create is called automatically.

AOwner is the component that is responsible for freeing the lookup control. It becomes the value of the Owner property.

See Also

Destroy (see page 9056)

Create (see page 21320)

TDataSourceLink (see page 8577)

TListSourceLink (see page 9498)

47.1.8.3 TDBLookupControl.Destroy Destructor

Destroys an instance of TDBLookupControl (see page 9044).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBLookupControl();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TDBLookupControl (see page 9044) reference is not nil and only then calls Destroy.

See Also

Create (see page 9055)

Free (see page 22300)

47.1.8.4 TDBLookupControl Methods

The methods of the TDBLookupControl class are listed here.

Protected Methods

	Name	Description
	CanModify (see page 9057)	Indicates whether the lookup control has rights to modify data.
	GetBorderSize (see page 9058)	This is GetBorderSize, a member of class TDBLookupControl.
	GetTextHeight (see page 9058)	Indicates how many pixels are required for one line of text in the list.
	KeyValueChanged (see page 9058)	Updates the control when the value of the protected KeyValue (see page 9065) property changes.
	ListLinkDataChanged (see page 9058)	Updates the control when its data binding changes.
	LocateKey (see page 9059)	Locates the value specified by KeyValue (see page 9065) in the lookup dataset.
	Notification (see page 9059)	Responds to notifications that components are being created or destroyed.
	ProcessSearchKey (see page 9060)	Adds a character to the protected SearchText (see page 9069) property and updates the selection to the first matching list value.
	SelectKeyValue (see page 9060)	Updates the value of the control so that it reflects a new key field value.
	UpdateDataFields (see page 9061)	Updates the control when the DataField (see page 9062) property changes.
	UpdateListFields (see page 9061)	Updates the control when the list of lookup values changes.

Public Methods

	Name	Description
	ExecuteAction (see page 9057)	Invokes an action with the TDBLookupControl (see page 9044) as its target.
	UpdateAction (see page 9060)	Updates an action to reflect the current state of the TDBLookupControl (see page 9044).

47.1.8.4.1 TDBLookupControl.CanModify Method

Indicates whether the lookup control has rights to modify data.

Pascal

```
function CanModify: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanModify();
```

Description

CanModify is used internally to determine whether the lookup control is able to modify the value of the field specified by the DataField (see page 9062) property.

CanModify returns trueif

The ReadOnly (see page 9069) property is false.

The ListActive (see page 9065) property is true.

The DataField (see page 9062) property refers to a field that can be modified.

Otherwise, CanModify returns false.

See Also

readonly (see page 9069)

DataField (see page 9062)

ListActive (see page 9065)

47.1.8.4.2 TDBLookupControl.ExecuteAction Method

Invokes an action with the TDBLookupControl (see page 9044) as its target.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

ExecuteAction is called automatically when the user invokes an action with the TDBLookupControl (see page 9044) as a target. The Action parameter specifies the action that was invoked. ExecuteAction returns trueif the action was successfully dispatched, and false if the TDBLookupControl (see page 9044) could not handle the action.

See Also

UpdateAction (see page 9060)

Execute (see page 2712)

HandlesTarget (see page 2713)

47.1.8.4.3 TDBLookupControl.GetBorderSize Method

Pascal

```
function GetBorderSize: Integer; virtual;
```

C++

```
virtual __fastcall int GetBorderSize();
```

Description

This is GetBorderSize, a member of class TDBLookupControl.

47.1.8.4.4 TDBLookupControl.GetTextHeight Method

Indicates how many pixels are required for one line of text in the list.

Pascal

```
function GetTextHeight: Integer; virtual;
```

C++

```
virtual __fastcall int GetTextHeight();
```

Description

GetTextHeight is used internally to manage the height of the control so that the list does not include partial (clipped) rows. It indicates the number of pixels required for a single row of text in the list.

47.1.8.4.5 TDBLookupControl.KeyValueChanged Method

Updates the control when the value of the protected KeyValue (see page 9065) property changes.

Pascal

```
procedure KeyValueChanged; virtual;
```

C++

```
virtual __fastcall KeyValueChanged();
```

Description

KeyValueChanged is called automatically when the value of the KeyValue (see page 9065) property changes. It allows the lookup control to make any necessary adjustments to display the new value in the list.

As implemented in TDBLookupControl (see page 9044), KeyValueChanged does nothing. Descendant classes override this method to update the display properties of the control to reflect the new value.

See Also

KeyValue (see page 9065)

47.1.8.4.6 TDBLookupControl.ListLinkDataChanged Method

Updates the control when its data binding changes.

Pascal

```
procedure ListLinkDataChanged; virtual;
```

C++

```
virtual __fastcall ListLinkDataChanged();
```

Description

ListLinkDataChanged is called automatically when the data displayed in the lookup list changes for any reason.

As implemented in TDBLookupControl (see page 9044), ListLinkDataChanged does nothing. Descendant classes override this method to update any internal properties that depend on the current list.

See Also

ListField (see page 9066)

47.1.8.4.7 TDBLookupControl.LocateKey Method

Locates the value specified by KeyValue (see page 9065) in the lookup dataset.

Pascal

```
function LocateKey: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean LocateKey();
```

Description

LocateKey is used internally to look up key values in the lookup dataset so that the control can find the appropriate display values.

LocateKey calls the lookup dataset's Locate method, passing in the current values of KeyField (see page 9064) and KeyValue (see page 9065). If the key value is found, LocateKey positions the lookup dataset on the record with the matching value and returns true. Otherwise, it returns false.

See Also

KeyValue (see page 9065)

Locate (see page 6970)

KeyField (see page 9064)

47.1.8.4.8 TDBLookupControl.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

After calling the inherited class' method, Notification checks whether the notification indicates the removal of DataSource (see page 9063) or ListSource (see page 9068). If so, Notification sets the property to nil (Delphi) or NULL (C++).

Derived classes can override Notification method to handle additional notifications, but the inherited class' method should be always be called.

See Also

DataSource (see page 9063)

ListSource (see page 9068)

47.1.8.4.9 TDBLookupControl.ProcessSearchKey Method

Adds a character to the protected SearchText (see page 9069) property and updates the selection to the first matching list value.

Pascal

```
procedure ProcessSearchKey(Key: Char); virtual;
```

C++

```
virtual __fastcall ProcessSearchKey(Char Key);
```

Description

ProcessSearchKey is called automatically when the user types a key into the lookup control. It forms the basis of the control's incremental searching, updating the list to the string that begins with the characters typed in thus far.

See Also

searchtext (see page 9069)

ListField (see page 9066)

47.1.8.4.10 TDBLookupControl.SelectKeyValue Method

Updates the value of the control so that it reflects a new key field value.

Pascal

```
procedure SelectKeyValue(const Value: Variant); virtual;
```

C++

```
virtual __fastcall SelectKeyValue(const Variant Value);
```

Description

SelectKeyValue is used internally to set the value of the control to reflect the specified value on its KeyField (see page 9064).

Value is the new value for the KeyValue (see page 9065) property.

See Also

KeyValue (see page 9065)

KeyField (see page 9064)

47.1.8.4.11 TDBLookupControl.UpdateAction Method

Updates an action to reflect the current state of the TDBLookupControl (see page 9044).

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

UpdateAction is called automatically when the application is idle to give the TDBLookupControl (see page 9044) an opportunity to update associated actions. This allows actions to be grayed, checked, or to otherwise reflect the status of the TDBLookupControl (see page 9044) they target.

The Action parameter specifies the action component that should be updated. UpdateAction returns trueif the action component now reflects the state of the component, and false if it did not know how to update the action.

See Also

ExecuteAction (see page 9057)

UpdateTarget (see page 2715)

HandlesTarget (see page 2713)

47.1.8.4.12 TDBLookupControl.UpdateDataFields Method

Updates the control when the DataField (see page 9062) property changes.

Pascal

```
procedure UpdateDataFields; virtual;
```

C++

```
virtual __fastcall UpdateDataFields();
```

Description

UpdateDataFields is called automatically when the DataField (see page 9062) property changes. It updates the internal properties that keep track of the field that the lookup control represents.

See Also

DataField (see page 9062)

47.1.8.4.13 TDBLookupControl.UpdateListFields Method

Updates the control when the list of lookup values changes.

Pascal

```
procedure UpdateListFields; virtual;
```

C++

```
virtual __fastcall UpdateListFields();
```

Description

UpdateListFields is called automatically when the list of lookup values changes. It synchronizes all the properties of the control so that they reflect the new list of values.

See Also

ListField (see page 9066)

KeyField (see page 9064)

47.1.8.5 TDBLookupControl Properties

The properties of the TDBLookupControl class are listed here.

Protected Properties

	Name	Description
	DataField (see page 9062)	Identifies the field whose value is represented by the lookup control.
R	DataLink (see page 9063)	Helps the lookup control manage its link to the data source and respond to data events.
	DataSource (see page 9063)	Links the lookup control to the dataset that contains the DataField (see page 9062).
R	HasFocus (see page 9064)	Indicates whether the lookup control has input focus.
	KeyField (see page 9064)	Identifies the field in the ListSource (see page 9068) dataset that must match the value of the DataField (see page 9062) field.
	KeyValue (see page 9065)	Represents the common value of the KeyField (see page 9064) field and the DataField (see page 9062) field.
R	ListActive (see page 9065)	Indicates whether the ListFields (see page 9067) property is currently being updated.
	ListField (see page 9066)	Identifies the field or fields whose values are displayed in the lookup control.
	ListFieldIndex (see page 9067)	Specifies which field from the ListField (see page 9066) property is used for incremental searching.
R	ListFields (see page 9067)	Lists the field components whose values are displayed in the lookup control.
R	ListLink (see page 9067)	Identifies the TListSourceLink (see page 9498) object that coordinates data communication with the lookup dataset.
	ListSource (see page 9068)	Identifies a data source for the data displayed in the lookup control.
	NullValueKey (see page 9068)	Specifies the key combination that assigns a blank value to the associated field.
	ParentColor (see page 9069)	Determines where a control looks for its color information.
	ReadOnly (see page 9069)	This is ReadOnly, a member of class TDBLookupControl.
	SearchText (see page 9069)	This is SearchText, a member of class TDBLookupControl.
	TabStop (see page 9070)	Determines if the user can tab to a control.

Public Properties

	Name	Description
R	Field (see page 9064)	Identifies the TField object the lookup control represents.

47.1.8.5.1 TDBLookupControl.DataField Property

Identifies the field whose value is represented by the lookup control.

Pascal

```
property DataField: WideString;
```

C++

```
__property BSTR DataField;
```

Description

Use DataField to bind the lookup control to a field in the dataset specified by the DataSource (see page 9063) property. The DataField is the field whose value can be set by the lookup control, not the field which supplies the lookup values that are displayed by the lookup control.

If DataField specifies a lookup field component, there is no need to set the KeyField (see page 9064) or ListField (see page 9066) properties, as the field component contains all the information about the relationship between the data field value and the lookup value.

See Also

DataSource (see page 9063)

KeyField (see page 9064)

ListField (see page 9066)

Displaying and Editing Data in Lookup List and Combo Boxes

47.1.8.5.2 TDBLookupControl.DataLink Property

Helps the lookup control manage its link to the data source and respond to data events.

Pascal

```
property DataLink: TDataSourceLink;
```

C++

```
__property TDataSourceLink DataLink;
```

Description

DataLink is used internally to access the TDataSourceLink (see page 8577) that coordinates communication between the dataset in which you find the field that this control represents, the data source for that dataset (identified by the DataSource (see page 9063) property), and the lookup control.

See Also

DataSource (see page 9063)

DataField (see page 9062)

ListLink (see page 9067)

47.1.8.5.3 TDBLookupControl.DataSource Property

Links the lookup control to the dataset that contains the DataField (see page 9062).

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Use DataSource to specify the data source component that identifies the dataset the lookup control represents. The lookup control represents a field in one table by displaying the value of a corresponding field in another table. DataSource links to the dataset for the field the control represents, as opposed to the ListSource (see page 9068), which links the lookup control to the lookup table that holds the field that is actually displayed. If the lookup control is used to edit data, the field that actually gets changed is the one in the DataSource.

See Also

DataField (see page 9062)

ListSource (see page 9068)

Displaying and Editing Data in Lookup List and Combo Boxes

47.1.8.5.4 TDBLookupControl.Field Property

Identifies the TField object the lookup control represents.

Pascal

```
property Field: TField;
```

C++

```
__property TField Field;
```

Description

Use the Field object to directly access the field component for the field the lookup control represents. If Field is a lookup field, the properties of the field component describe the relationship between the field and the lookup dataset. To allow a lookup control to represent a set of lookup values that do not come from another dataset, use the DataField (see page 9062) property to bind to a lookup field and use the LookupList property of the TField object to specify a list of lookup values.

See Also

DataField (see page 9062)

DataSource (see page 9063)

LookupList (see page 7172)

47.1.8.5.5 TDBLookupControl.HasFocus Property

Indicates whether the lookup control has input focus.

Pascal

```
property HasFocus: Boolean;
```

C++

```
__property Boolean HasFocus;
```

Description

HasFocus is used internally to determine whether the lookup control has input focus. Applications should use the public Focused method instead.

See Also

FocusControl (see page 13875)

FocusControl (see page 7128)

Focused (see page 6150)

47.1.8.5.6 TDBLookupControl.KeyField Property

Identifies the field in the ListSource (see page 9068) dataset that must match the value of the DataField (see page 9062) field.

Pascal

```
property KeyField: string;
```

C++

```
__property AnsiString KeyField;
```

Description

Use KeyField to link the ListSource (see page 9068) of the lookup control to the DataSource (see page 9063). Although the name of the field specified as the KeyField does not have to be the same as the name of the field specified as the DataField (see page 9062), the two fields must have the same values.

After specifying the ListField (see page 9066), choose which field the lookup control will actually display with the ListField (see page 9066) property.

If the DataField (see page 9062) field is a lookup field, don't specify anything for KeyField or ListField (see page 9066); lookup controls automatically use the data field's LookupKeyFields property for KeyField.

See Also

DataField (see page 9062)

DataSource (see page 9063)

KeyValue (see page 9065)

ListField (see page 9066)

ListSource (see page 9068)

LookupKeyFields (see page 7172)

Displaying and Editing Data in Lookup List and Combo Boxes

47.1.8.5.7 TDBLookupControl.KeyValue Property

Represents the common value of the KeyField (see page 9064) field and the DataField (see page 9062) field.

Pascal

```
property KeyValue: Variant;
```

C++

```
__property Variant KeyValue;
```

Description

Use KeyField (see page 9064) to determine the value represented by the lookup control (not the value displayed by the lookup control). When KeyValue is set, the lookup control attempts to find a record from the ListSource (see page 9068)'s dataset where the value of KeyField (see page 9064) matches KeyValue. If such a match is found, the lookup control displays the value of ListField (see page 9066) on that record.

See Also

DataField (see page 9062)

KeyField (see page 9064)

ListField (see page 9066)

ListSource (see page 9068)

LookupKeyFields (see page 7172)

47.1.8.5.8 TDBLookupControl.ListActive Property

Indicates whether the ListFields (see page 9067) property is currently being updated.

Pascal

```
property ListActive: Boolean;
```

C++

```
__property Boolean ListActive;
```

Description

ListActive is used internally to prevent the lookup control from attempting to use its data-aware properties while it is in the middle of an update. While refreshing the ListFields (see page 9067) cache to reflect the current fields in the dataset, ListActive is false. At all other times, ListActive is true.

See Also

ListFields (see page 9067)

47.1.8.5.9 TDBLookupControl.ListField Property

Identifies the field or fields whose values are displayed in the lookup control.

Pascal

```
property ListField: string;
```

C++

```
__property AnsiString ListField;
```

Description

The lookup control represents a field in one table by displaying the value of one or more corresponding fields in another table. ListField is the name of the field or fields in the lookup table that are actually displayed, as opposed to the DataField (see page 9062), which is the field in the DataSource (see page 9063) that the lookup control actually represents, or the KeyField (see page 9064), which is the field in the lookup table with the same value as the DataField (see page 9062).

To fully specify the list fields, both a dataset for the lookup table and the fields within that dataset must be defined. The ListSource (see page 9068) property of the lookup control specifies the dataset for the lookup table.

ListField can represent more than one field. Separate multiple field names with semicolons.

Before specifying ListField, specify the link between the two datasets using the KeyField (see page 9064) property. If ListField is not set, lookup controls display KeyField (see page 9064) field values by default. If the DataField (see page 9062) field is a lookup field, don't specify anything for ListField; the data controls automatically use the lookup field's LookupResultField property as ListField.

See Also

DataField (see page 9062)

DataSource (see page 9063)

KeyField (see page 9064)

KeyValue (see page 9065)

ListFieldIndex (see page 9067)

ListSource (see page 9068)

Displaying and Editing Data in Lookup List and Combo Boxes

47.1.8.5.10 TDBLookupControl.ListFieldIndex Property

Specifies which field from the ListField (see page 9066) property is used for incremental searching.

Pascal

```
property ListFieldIndex: Integer;
```

C++

```
__property int ListFieldIndex;
```

Description

When the ListField (see page 9066) property specifies more than one field, use ListFieldIndex to specify which of those fields is the one to use for incremental searches. For the TDBLookupComboBox (see page 8963) object, ListFieldIndex also determines which field appears in the edit region of the combo box.

ListFieldIndex allows the most important value to appear in a position other than the first when all the values of the ListField (see page 9066) fields are displayed. The value of ListFieldIndex must be less than the number of fields specified by the ListField (see page 9066) property.

See Also

ListField (see page 9066)

ListFields (see page 9067)

47.1.8.5.11 TDBLookupControl.ListFields Property

Lists the field components whose values are displayed in the lookup control.

Pascal

```
property ListFields: TList;
```

C++

```
__property TList ListFields;
```

Description

ListFields caches the field components that are named by the ListField (see page 9066) property.

See Also

ListField (see page 9066)

47.1.8.5.12 TDBLookupControl.ListLink Property

Identifies the TListSourceLink (see page 9498) object that coordinates data communication with the lookup dataset.

Pascal

```
property ListLink: TListSourceLink;
```

C++

```
__property TListSourceLink ListLink;
```

Description

ListLink is used internally to access the TListSourceLink (see page 9498) that coordinates communication between the lookup dataset, the lookup data source (identified by the ListSource (see page 9068) property), and the lookup control.

See Also

ListSource (see page 9068)

DataLink (see page 9063)

47.1.8.5.13 TDBLookupControl.ListSource Property

Identifies a data source for the data displayed in the lookup control.

Pascal

```
property ListSource: TDataSource;
```

C++

```
__property TDataSource ListSource;
```

Description

Set LookupSource to the data source that contains the KeyField (see page 9064) and ListField (see page 9066) fields. If the DataField (see page 9062) field is a lookup field, don't specify anything for ListSource; data controls automatically use the lookup field's LookupDataSet property to create a data source.

See Also

DataField (see page 9062)

DataSource (see page 9063)

KeyField (see page 9064)

ListField (see page 9066)

ListFieldIndex (see page 9067)

LookupDataSet (see page 7171)

Displaying and Editing Data in Lookup List and Combo Boxes

47.1.8.5.14 TDBLookupControl.NullValueKey Property

Specifies the key combination that assigns a blank value to the associated field.

Pascal

```
property NullValueKey: TShortCut;
```

C++

```
__property TShortCut NullValueKey;
```

Description

Set NullValueKey to let users enter a "blank" value into the associated field. When the user types the key combination that this shortcut represents, the associated field is cleared.

Tip: Use the ShortCut function or the TextToShortCut function to create a value to assign to NullValueKey.

See Also

DataField (see page 9062)

Clear (see page 7126)

TextToShortCut (see page 16196)

ShortCut (see page 16193)

47.1.8.5.15 TDBLookupControl.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

47.1.8.5.16 TDBLookupControl.ReadOnly Property

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

This is ReadOnly, a member of class TDBLookupControl.

47.1.8.5.17 TDBLookupControl.SearchText Property

Pascal

```
property SearchText: string;
```

C++

```
__property AnsiString SearchText;
```

Description

This is SearchText, a member of class TDBLookupControl.

47.1.8.5.18 TDBLookupControl.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

47.1.9 TDBLookupListBox Class

TDBLookupListBox provides a list of lookup items for filling in fields that require data from another dataset.

Class Hierarchy

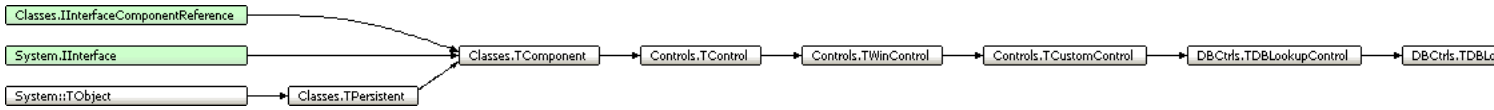

Pascal

```
TDBLookupListBox = class(TDBLookupControl);
```

C++

```
class TDBLookupListBox : public TDBLookupControl;
```

File

DBCtrls

Description

Use TDBLookupListBox to provide users with a convenient list of lookup items to set a field value using the values of a field in another dataset. Lookup list boxes usually display values that are a represent a more meaningful description of the actual field value.

If TDBLookupListBox is linked to a lookup field component, it automatically reads the relationship between the field value and the lookup values in the lookup dataset from the field component. The relationship between field values and the corresponding values in the lookup dataset can also be explicitly set using the properties of the lookup list box when the list box is not linked to a lookup field component.

See Also

TDBLookupComboBox (see page 8963)

TDBListBox (see page 8884)

Displaying and Editing Data in List and Combo Boxes

Displaying and Editing Data in Lookup List and Combo Boxes

Using Common Data Control Features

47.1.9.1 TDBLookupListBox Members

The following tables list the members exposed by TDBLookupListBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.

	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanModify (see page 9057)	Indicates whether the lookup control has rights to modify data.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 9082)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 9083)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.

	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorderSize (see page 9058)	This is GetBorderSize, a member of class TDBLookupControl.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTextHeight (see page 9058)	Indicates how many pixels are required for one line of text in the list.
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
V	KeyDown (see page 9084)	Respond to key press events.
V	KeyPress (see page 9085)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	KeyValueChanged (see page 9085)	Updates the control when the value of the protected KeyValue property changes.
V	ListLinkDataChanged (see page 9085)	Updates the control when its data binding changes.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	LocateKey (see page 9059)	Locates the value specified by KeyValue (see page 9065) in the lookup dataset.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 9086)	OnMouseDown event dispatcher.
V	MouseMove (see page 9086)	Respond to mouse moving over control area..
V	MouseUp (see page 9087)	OnMouseUp event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 9059)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 9087)	Renders the image of a custom control.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	ProcessSearchKey (see page 9060)	Adds a character to the protected SearchText (see page 9069) property and updates the selection to the first matching list value.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.

	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectKeyValue (see page 9060)	Updates the value of the control so that it reflects a new key field value.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateDataFields (see page 9061)	Updates the control when the DataField (see page 9062) property changes.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateListFields (see page 9089)	Updates the control when the list of lookup values changes.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.

V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
R	DataLink (see page 9063)	Helps the lookup control manage its link to the data source and respond to data events.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
R	HasFocus (see page 9064)	Indicates whether the lookup control has input focus.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	ListActive (see page 9065)	Indicates whether the ListFields (see page 9067) property is currently being updated.
R	ListFields (see page 9067)	Lists the field components whose values are displayed in the lookup control.
R	ListLink (see page 9067)	Identifies the TListSourceLink (see page 9498) object that coordinates data communication with the lookup dataset.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	SearchText (see page 9069)	This is SearchText, a member of class TDBLookupControl.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.

	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 9082)	Creates an instance of TDBLookupListBox (see page 9070).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 9056)	Destroys an instance of TDBLookupControl (see page 9044).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
V	ExecuteAction (see page 9083)	Performs an action directed at the DB (see page 6725) lookup list box.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.

	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 9088)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
V	UpdateAction (see page 9088)	Updates an action component to reflect the current state of the DB (see page 6725) lookup list box.
	UpdateControlState (see page 6186)	Respond to state change.
V	UseRightToLeftAlignment (see page 9089)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Field (see page 9064)	Identifies the TField object the lookup control represents.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	KeyValue (see page 9106)	Represents the common value of the KeyField field and the DataField field.

	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	SelectedItem (see page 9116)	Contains the value displayed in the list box that is currently selected.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 9121)	Occurs when the user clicks the control.
	OnContextPopup (see page 9123)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 9124)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 9124)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 9126)	Occurs when the user drags an object over a control.
	OnEndDock (see page 9129)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 9129)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 9130)	Occurs when a control receives the input focus.
	OnExit (see page 9132)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 9134)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 9137)	Occurs when a key is pressed.
	OnKeyUp (see page 9138)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 9140)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 9141)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 9143)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 9143)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 9143)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 9146)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 9148)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 9148)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 9091)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 9094)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 9095)	Specifies which edges of the control are beveled.
	BevelInner (see page 9095)	Specifies the cut of the inner bevel.
	BevelKind (see page 9095)	Specifies the control's bevel style.
	BevelOuter (see page 9096)	Specifies the cut of the outer bevel.

	BevelWidth (see page 9096)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 9097)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 9097)	Determines whether the lookup list box has a border.
	Color (see page 9098)	Specifies the background color of the control.
	Constraints (see page 9098)	Specifies the size constraints for the control.
	Ctl3D (see page 9099)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DataField (see page 9099)	Identifies the field whose value is represented by the lookup control.
	DataSource (see page 9100)	Links the lookup control to the dataset that contains the DataField.
	DragCursor (see page 9100)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 9101)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 9101)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 9103)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 9103)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 9104)	Determines the behavior of the input method editor (IME).
	ImeName (see page 9105)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	KeyField (see page 9105)	Identifies the field in the ListSource dataset that must match the value of the DataField field.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	ListField (see page 9106)	Identifies the field or fields whose values are displayed in the lookup control.
	ListFieldIndex (see page 9107)	Specifies which field from the ListField property is used for incremental searching.
	ListSource (see page 9108)	Identifies a data source for the data displayed in the lookup control.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NullValueKey (see page 9108)	Specifies the key combination that assigns a blank value to the associated field.
	ParentBiDiMode (see page 9109)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 9109)	Determines where a control looks for its color information.
	ParentCtl3D (see page 9109)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 9110)	Determines where a control looks for its font information.
	ParentShowHint (see page 9110)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 9111)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 9115)	This is ReadOnly, a member of class TDBLookupListBox.
	RowCount (see page 9116)	Specifies how many rows are visible in the list box.
	ShowHint (see page 9116)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 9118)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 9118)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 9119)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

47.1.9.2 TDBLookupListBox.Create Constructor

Creates an instance of TDBLookupListBox (see page 9070).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDBLookupListBox(TComponent * AOwner);
```

Description

Call Create to instantiate a TDBLookupListBox (see page 9070) object at runtime. For TDBLookupListBox (see page 9070) objects placed on forms or data modules at design time, Create is called automatically.

Pass a single Component as a parameter to provide the TDBLookupListBox (see page 9070) object with an Owner that is responsible for freeing it.After calling the inherited constructor to allocate and initialize a generic lookup control, Create creates and initializes the helper objects needed to link to a dataset.

See Also

TFieldDataLink (see page 9487)

47.1.9.3 TDBLookupListBox Methods

The methods of the TDBLookupListBox class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 9082)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 9083)	Creates the underlying window.
	KeyDown (see page 9084)	Respond to key press events.
	KeyPress (see page 9085)	Respond to keyboard input.
	KeyValueChanged (see page 9085)	Updates the control when the value of the protected KeyValue property changes.
	ListLinkDataChanged (see page 9085)	Updates the control when its data binding changes.
	MouseDown (see page 9086)	OnMouseDown event dispatcher.
	MouseMove (see page 9086)	Respond to mouse moving over control area..
	MouseUp (see page 9087)	OnMouseUp event dispatcher.
	Paint (see page 9087)	Renders the image of a custom control.
	UpdateListFields (see page 9089)	Updates the control when the list of lookup values changes.

Public Methods

	Name	Description
	ExecuteAction (see page 9083)	Performs an action directed at the DB (see page 6725) lookup list box.
	SetBounds (see page 9088)	Sets the windowed control's boundary properties all at once.
	UpdateAction (see page 9088)	Updates an action component to reflect the current state of the DB (see page 6725) lookup list box.
	UseRightToLeftAlignment (see page 9089)	Specifies whether the control's alignment should be reversed right-to-left.

47.1.9.3.1 TDBLookupListBox.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

47.1.9.3.2 TDBLookupListBox.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams (see page 6135)

CreateWindowHandle (see page 6137)

Perform (see page 5776)

DestroyWnd (see page 6141)

47.1.9.3.3 TDBLookupListBox.ExecuteAction Method

Performs an action directed at the DB (see page 6725) lookup list box.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

ExecuteAction is called automatically when the user invokes an action with the DB (see page 6725) lookup list box as a target. The Action parameter specifies the action component that was invoked.

ExecuteAction first calls the inherited method, which checks whether the action component knows how to perform its function with the DB (see page 6725) lookup list box as a target. If the action does not handle the invocation with the list box as a target, ExecuteAction checks whether this is an action that takes a dataset as a target. If so, ExecuteAction invokes the action, using the dataset associated with the DataSource (see page 9100) property as a target.

See Also

UpdateAction (see page 9088)

DataSource (see page 9063)

ExecuteTarget (see page 2712)

47.1.9.3.4 TDBLookupListBox.KeyDown Method

Respond to key press events.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-down message (WM_KEYDOWN) from Windows, its message handler calls the DoKeyDown method. If DoKeyDown determines that the control should, in fact, process the character, it calls KeyDown, passing the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

See Also

KeyUp (see page 6164)

OnKeyDown (see page 6225)

DoKeyDown (see page 6145)

DoKeyPress (see page 6146)

DoKeyUp (see page 6146)

47.1.9.3.5 TDBLookupListBox.KeyPress Method

Respond to keyboard input.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

When a windowed control receives a key-press message (WM_CHAR) from Windows, its message handler calls the DoKeyPress method. If DoKeyPress determines that the control should, in fact, process the character, it calls KeyPress, passing the key code in the Key parameter.

KeyPress calls any event handler attached to the OnKeyPress event. Override KeyPress to provide other responses in addition to the event-handler call.

Either KeyPress or the OnKeyPress event handler it calls can suppress further processing of a character by setting the Key parameter to zero.

Note: The Key parameter is the character represented by the key that is pressed, not a Windows virtual key code.

See Also

KeyDown (see page 6163)

KeyUp (see page 6164)

OnKeyPress (see page 6227)

47.1.9.3.6 TDBLookupListBox.KeyValueChanged Method

Updates the control when the value of the protected KeyValue property changes.

Pascal

```
procedure KeyValueChanged; override;
```

C++

```
virtual __fastcall KeyValueChanged();
```

Description

KeyValueChanged is called automatically when the value of the KeyValue property changes. It allows the lookup control to make any necessary adjustments to display the new value in the list.

As implemented in TDBLookupControl, KeyValueChanged does nothing. Descendant classes override this method to update the display properties of the control to reflect the new value.

See Also

KeyValue (see page 9065)

47.1.9.3.7 TDBLookupListBox.ListLinkDataChanged Method

Updates the control when its data binding changes.

Pascal

```
procedure ListLinkDataChanged; override;
```

C++

```
virtual __fastcall ListLinkDataChanged();
```

Description

ListLinkDataChanged is called automatically when the data displayed in the lookup list changes for any reason.

As implemented in TDBLookupControl, ListLinkDataChanged does nothing. Descendant classes override this method to update any internal properties that depend on the current list.

See Also

ListField (see page 9066)

47.1.9.3.8 TDBLookupListBox.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove (see page 5773)

MouseUp (see page 5774)

OnMouseDown (see page 5862)

47.1.9.3.9 TDBLookupListBox.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown (see page 5773)

MouseUp (see page 5774)

OnMouseMove (see page 5865)

47.1.9.3.10 TDBLookupListBox.MouseUp Method

OnMouseUp event dispatcher.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user releases a previously pressed mouse button while the cursor's hotspot is over the control.

A control calls MouseUp in response to any of the Windows mouse-up messages (WM_LBUTTONUP, WM_MBUTTONUP, WM_RBUTTONUP), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

See Also

MouseDown (see page 5773)

MouseMove (see page 5773)

OnMouseUp (see page 5867)

47.1.9.3.11 TDBLookupListBox.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device

context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas (see page 5921)

PaintWindow (see page 5920)

GetDeviceContext (see page 6155)

ClipRect (see page 14486)

47.1.9.3.12 TDBLookupListBox.SetBounds Method

Sets the windowed control's boundary properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to change all of the control's boundary properties at once. The same effect can be achieved by setting the Left, Top, Width, and Height properties separately. By setting all four properties at once, SetBounds ensures that the control will not repaint between changes

Specify the values for the Left, Top, Width, and Height properties as the value of the ALeft, ATop, AWidth, and AHeight parameters, respectively.

See Also

Height (see page 5825)

Left (see page 5830)

Top (see page 5844)

Width (see page 5849)

47.1.9.3.13 TDBLookupListBox.UpdateAction Method

Updates an action component to reflect the current state of the DB (see page 6725) lookup list box.

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

UpdateAction is called automatically when the CPU is idle to allow the DB (see page 6725) lookup list box the opportunity of updating an action. For example, UpdateAction disables actions that require the dataset to be active when the dataset associated with the DataSource (see page 9100) property is not active.

The Action parameter specifies the action component that should be updated.

See Also

ExecuteAction (see page 9083)

DataSource (see page 9063)

UpdateTarget (see page 2715)

47.1.9.3.14 TDBLookupListBox.UpdateListFields Method

Updates the control when the list of lookup values changes.

Pascal

```
procedure UpdateListFields; override;
```

C++

```
virtual __fastcall UpdateListFields();
```

Description

UpdateListFields is called automatically when the list of lookup values changes. It synchronizes all the properties of the control so that they reflect the new list of values.

See Also

ListField (see page 9066)

KeyField (see page 9064)

47.1.9.3.15 TDBLookupListBox.UseRightToLeftAlignment Method

Specifies whether the control's alignment should be reversed right-to-left.

Pascal

```
function UseRightToLeftAlignment: Boolean; override;
```

C++

```
virtual __fastcall Boolean UseRightToLeftAlignment();
```

Description

Controls call UseRightToLeftAlignment when implementing the BiDiMode property. UseRightToLeftAlignment indicates whether the control's alignment should be reversed right-to-left. This information can be used when painting the control or when responding to mouse events.

UseRightToLeftAlignment returns true if the application is running on a system with a Middle Eastern locale and the BiDiMode property is bdRightToLeft. Otherwise, it returns false. Descendants override UseRightToLeftAlignment if alignment in Middle Eastern locales depends on other factors.

See Also

BiDiMode (see page 5803)

GetControlsAlignment (see page 5759)

Text (see page 5842)

Caption (see page 5806)

47.1.9.4 TDBLookupListBox Properties

The properties of the TDBLookupListBox class are listed here.

Public Properties

	Name	Description
	KeyValue (see page 9106)	Represents the common value of the KeyField field and the DataField field.
R	SelectedItem (see page 9116)	Contains the value displayed in the list box that is currently selected.

Published Properties

	Name	Description
	Align (see page 9091)	Determines how the control aligns within its container (parent control).
	Anchors (see page 9094)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 9095)	Specifies which edges of the control are beveled.
	BevelInner (see page 9095)	Specifies the cut of the inner bevel.
	BevelKind (see page 9095)	Specifies the control's bevel style.
	BevelOuter (see page 9096)	Specifies the cut of the outer bevel.
	BevelWidth (see page 9096)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 9097)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 9097)	Determines whether the lookup list box has a border.
	Color (see page 9098)	Specifies the background color of the control.
	Constraints (see page 9098)	Specifies the size constraints for the control.
	Ctl3D (see page 9099)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DataField (see page 9099)	Identifies the field whose value is represented by the lookup control.
	DataSource (see page 9100)	Links the lookup control to the dataset that contains the DataField.
	DragCursor (see page 9100)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 9101)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 9101)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 9103)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 9103)	Controls the attributes of text written on or in the control.
	ImeMode (see page 9104)	Determines the behavior of the input method editor (IME).
	ImeName (see page 9105)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	KeyField (see page 9105)	Identifies the field in the ListSource dataset that must match the value of the DataField field.
	ListField (see page 9106)	Identifies the field or fields whose values are displayed in the lookup control.
	ListFieldIndex (see page 9107)	Specifies which field from the ListField property is used for incremental searching.
	ListSource (see page 9108)	Identifies a data source for the data displayed in the lookup control.
	NullValueKey (see page 9108)	Specifies the key combination that assigns a blank value to the associated field.
	ParentBiDiMode (see page 9109)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 9109)	Determines where a control looks for its color information.
	ParentCtl3D (see page 9109)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 9110)	Determines where a control looks for its font information.
	ParentShowHint (see page 9110)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 9111)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 9115)	This is ReadOnly, a member of class TDBLookupListBox.
	RowCount (see page 9116)	Specifies how many rows are visible in the list box.
	ShowHint (see page 9116)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 9118)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 9118)	Determines if the user can tab to a control.
	Visible (see page 9119)	Determines whether the component appears onscreen.

47.1.9.4.1 TDBLookupListBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
```

```
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
```

```
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
```

```
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

47.1.9.4.2 TDBLookupListBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

47.1.9.4.3 TDBLookupListBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

47.1.9.4.4 TDBLookupListBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

47.1.9.4.5 TDBLookupListBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

47.1.9.4.6 TDBLookupListBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

47.1.9.4.7 TDBLookupListBox.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

47.1.9.4.8 TDBLookupListBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

47.1.9.4.9 TDBLookupListBox.BorderStyle Property

Determines whether the lookup list box has a border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Set BorderStyle to specify whether a border should be drawn around the lookup list box. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

47.1.9.4.10 TDBLookupListBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

47.1.9.4.11 TDBLookupListBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

47.1.9.4.12 TDBLookupListBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

47.1.9.4.13 TDBLookupListBox.DataField Property

Identifies the field whose value is represented by the lookup control.

Pascal

```
property DataField: WideString;
```

C++

```
__property BSTR DataField;
```

Description

Use DataField to bind the lookup control to a field in the dataset specified by the DataSource property. The DataField is the field whose value can be set by the lookup control, not the field which supplies the lookup values that are displayed by the lookup control.

If DataField specifies a lookup field component, there is no need to set the KeyField or ListField properties, as the field component contains all the information about the relationship between the data field value and the lookup value.

See Also

DataSource (see page 9063)

KeyField (see page 9064)

ListField (see page 9066)

Displaying and Editing Data in Lookup List and Combo Boxes

47.1.9.4.14 TDBLookupListBox.DataSource Property

Links the lookup control to the dataset that contains the DataField.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Use DataSource to specify the data source component that identifies the dataset the lookup control represents. The lookup control represents a field in one table by displaying the value of a corresponding field in another table. DataSource links to the dataset for the field the control represents, as opposed to the ListSource, which links the lookup control to the lookup table that holds the field that is actually displayed. If the lookup control is used to edit data, the field that actually gets changed is the one in the DataSource.

See Also

DataField (see page 9062)

ListSource (see page 9068)

Displaying and Editing Data in Lookup List and Combo Boxes

47.1.9.4.15 TDBLookupListBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

47.1.9.4.16 TDBLookupListBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

47.1.9.4.17 TDBLookupListBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
```

```
      return;
    THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
    TNodeAttachMode AttachMode;
    TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
    if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
      AttachMode = naAddChild;
    else if (HT.Contains(htNowhere))
      AttachMode = naAdd;
    else if (HT.Contains(htOnIndent))
      AttachMode = naInsert;
    else
      return;
    TreeView1->Selected->MoveTo(Item, AttachMode);
  }

  void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
        int X, int Y, TDragState State, bool &Accept)
  {
    Accept = (Source == (TObject *)TreeView1);
  }
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

47.1.9.4.18 TDBLookupListBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

47.1.9.4.19 TDBLookupListBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
```

```
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

47.1.9.4.20 TDBLookupListBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

47.1.9.4.21 TDBLookupListBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

47.1.9.4.22 TDBLookupListBox.KeyField Property

Identifies the field in the ListSource dataset that must match the value of the DataField field.

Pascal

```
property KeyField: string;
```

C++

```
__property AnsiString KeyField;
```

Description

Use KeyField to link the ListSource of the lookup control to the DataSource. Although the name of the field specified as the KeyField does not have to be the same as the name of the field specified as the DataField, the two fields must have the same values.

After specifying the ListField, choose which field the lookup control will actually display with the ListField property.

If the DataField field is a lookup field, don't specify anything for KeyField or ListField; lookup controls automatically use the data field's LookupKeyFields property for KeyField.

See Also

DataField (see page 9062)

DataSource (see page 9063)

KeyValue (see page 9065)

ListField (see page 9066)

ListSource (see page 9068)

LookupKeyFields (see page 7172)

Displaying and Editing Data in Lookup List and Combo Boxes

47.1.9.4.23 TDBLookupListBox.KeyValue Property

Represents the common value of the KeyField field and the DataField field.

Pascal

```
property KeyValue: Variant;
```

C++

```
__property Variant KeyValue;
```

Description

Use KeyField to determine the value represented by the lookup control (not the value displayed by the lookup control). When KeyValue is set, the lookup control attempts to find a record from the ListSource's dataset where the value of KeyField matches KeyValue. If such a match is found, the lookup control displays the value of ListField on that record.

See Also

DataField (see page 9062)

KeyField (see page 9064)

ListField (see page 9066)

ListSource (see page 9068)

LookupKeyFields (see page 7172)

47.1.9.4.24 TDBLookupListBox.ListField Property

Identifies the field or fields whose values are displayed in the lookup control.

Pascal

```
property ListField: string;
```

C++

```
__property AnsiString ListField;
```

Description

The lookup control represents a field in one table by displaying the value of one or more corresponding fields in another table. ListField is the name of the field or fields in the lookup table that are actually displayed, as opposed to the DataField, which is the field in the DataSource that the lookup control actually represents, or the KeyField, which is the field in the lookup table with the same value as the DataField.

To fully specify the list fields, both a dataset for the lookup table and the fields within that dataset must be defined. The ListSource property of the lookup control specifies the dataset for the lookup table.

ListField can represent more than one field. Separate multiple field names with semicolons.

Before specifying ListField, specify the link between the two datasets using the KeyField property. If ListField is not set, lookup controls display KeyField field values by default. If the DataField field is a lookup field, don't specify anything for ListField; the data controls automatically use the lookup field's LookupResultField property as ListField.

See Also

DataField (see page 9062)

DataSource (see page 9063)

KeyField (see page 9064)

KeyValue (see page 9065)

ListFieldIndex (see page 9067)

ListSource (see page 9068)

Displaying and Editing Data in Lookup List and Combo Boxes

47.1.9.4.25 TDBLookupListBox.ListFieldIndex Property

Specifies which field from the ListField property is used for incremental searching.

Pascal

```
property ListFieldIndex: Integer;
```

C++

```
__property int ListFieldIndex;
```

Description

When the ListField property specifies more than one field, use ListFieldIndex to specify which of those fields is the one to use for incremental searches. For the TDBLookupComboBox object, ListFieldIndex also determines which field appears in the edit region of the combo box.

ListFieldIndex allows the most important value to appear in a position other than the first when all the values of the ListField fields are displayed. The value of ListFieldIndex must be less than the number of fields specified by the ListField property.

See Also

ListField (see page 9066)

ListFields (see page 9067)

47.1.9.4.26 TDBLookupListBox.ListSource Property

Identifies a data source for the data displayed in the lookup control.

Pascal

```
property ListSource: TDataSource;
```

C++

```
__property TDataSource ListSource;
```

Description

Set LookupSource to the data source that contains the KeyField and ListField fields. If the DataField field is a lookup field, don't specify anything for ListSource; data controls automatically use the lookup field's LookupDataSet property to create a data source.

See Also

DataField (see page 9062)

DataSource (see page 9063)

KeyField (see page 9064)

ListField (see page 9066)

ListFieldIndex (see page 9067)

LookupDataSet (see page 7171)

Displaying and Editing Data in Lookup List and Combo Boxes

47.1.9.4.27 TDBLookupListBox.NullValueKey Property

Specifies the key combination that assigns a blank value to the associated field.

Pascal

```
property NullValueKey: TShortCut;
```

C++

```
__property TShortCut NullValueKey;
```

Description

Set NullValueKey to let users enter a "blank" value into the associated field. When the user types the key combination that this shortcut represents, the associated field is cleared.

Tip: Use the ShortCut function or the TextToShortCut function to create a value to assign to NullValueKey.

See Also

DataField (see page 9062)

Clear (see page 7126)

TextToShortCut (see page 16196)

ShortCut (see page 16193)

47.1.9.4.28 TDBLookupListBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

47.1.9.4.29 TDBLookupListBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

47.1.9.4.30 TDBLookupListBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

47.1.9.4.31 TDBLookupListBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

47.1.9.4.32 TDBLookupListBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

47.1.9.4.33 TDBLookupListBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
```

```
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Cut1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      const int num_items = 4;
      for (int index = 0; index < num_items; ++index)
      {
          TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
          PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
          NewItem->Caption = "Menu Item " + IntToStr(index);
          NewItem->Tag = index;
          NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
          TNotifyEvent();
      }
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
          {
```

```
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
```

```
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
```

```
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

47.1.9.4.34 TDBLookupListBox.ReadOnly Property

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

This is ReadOnly, a member of class TDBLookupListBox.

47.1.9.4.35 TDBLookupListBox.RowCount Property

Specifies how many rows are visible in the list box.

Pascal

```
property RowCount: Integer;
```

C++

```
__property int RowCount;
```

Description

Set RowCount to the number of rows displayed in the lookup list box. The list box is resized to match the specified number of rows. Resizing the list box (such as by changing its Height property) automatically changes the value of RowCount to match the new height.

See Also

Height (see page 5825)

Displaying and Editing Data in Lookup List and Combo Boxes

47.1.9.4.36 TDBLookupListBox.SelectedItem Property

Contains the value displayed in the list box that is currently selected.

Pascal

```
property SelectedItem: string;
```

C++

```
__property AnsiString SelectedItem;
```

Description

Read SelectedItem to determine the value the user selected using the lookup list box. If no item is selected, SelectedItem is an empty string.

```
StrVar := DBLookupListBox1.SelectedItem;
StrVar = DBLookupListBox1->SelectedItem;
```

See Also

KeyField (see page 9064)

47.1.9.4.37 TDBLookupListBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
```

```
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

47.1.9.4.38 TDBLookupListBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

47.1.9.4.39 TDBLookupListBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

47.1.9.4.40 TDBLookupListBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
```

```
      pEdit->Parent = pPage;
      pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
      pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
    }
    PageControl1Change(Sender);
  }
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

47.1.9.5 TDBLookupListBox Events

The events of the TDBLookupListBox class are listed here.

Published Events

	Name	Description
	OnClick (see page 9121)	Occurs when the user clicks the control.
	OnContextPopup (see page 9123)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

	OnDblClick (see page 9124)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 9124)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 9126)	Occurs when the user drags an object over a control.
	OnEndDock (see page 9129)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 9129)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 9130)	Occurs when a control receives the input focus.
	OnExit (see page 9132)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 9134)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 9137)	Occurs when a key is pressed.
	OnKeyUp (see page 9138)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 9140)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 9141)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 9143)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 9143)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 9143)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 9146)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 9148)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 9148)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

47.1.9.5.1 TDBLookupListBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
```

```
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

47.1.9.5.2 TDBLookupListBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

47.1.9.5.3 TDBLookupListBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

47.1.9.5.4 TDBLookupListBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

47.1.9.5.5 TDBLookupListBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
```

```
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code comes from an application that contains a list box
  and three labels, each with a different font and color.  The
  DragMode property for each of the labels is dmAutomatic.
  The user can select a label and drag it to a list box and
  drop it. When the label is dropped, the items in the list
  box assume the color and font of the dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.9.5.6 TDBLookupListBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.9.5.7 TDBLookupListBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
```

```
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

47.1.9.5.8 TDBLookupListBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
```

```
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

47.1.9.5.9 TDBLookupListBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

47.1.9.5.10 TDBLookupListBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not

generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
```

```
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;
```

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

47.1.9.5.11 TDBLookupListBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

47.1.9.5.12 TDBLookupListBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

```
void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

47.1.9.5.13 TDBLookupListBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.9.5.14 TDBLookupListBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
```

```
      }
      else
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
```

```
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.9.5.15 TDBLookupListBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

47.1.9.5.16 TDBLookupListBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

47.1.9.5.17 TDBLookupListBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
```

```
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.9.5.18 TDBLookupListBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
```

```
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
```

```
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

47.1.9.5.19 TDBLookupListBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

47.1.9.5.20 TDBLookupListBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

47.1.10 TDBMemo Class

TDBMemo represents a multiline edit control that can display and edit a field in a dataset.

Class Hierarchy

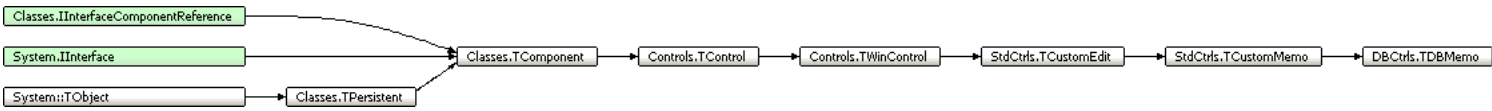

Pascal

```
TDBMemo = class(TCustomMemo);
```

C++

```
class TDBMemo : public TCustomMemo;
```

File

DBCtrls

Description

Use TDBMemo to let users edit a field that may contain lengthy textual data or to simply display the contents of such a field.

TDBMemo uses the Text property to represent the contents of the field.

TDBMemo permits multiple lines of text. Thus, TDBMemo is appropriate for long alphanumeric fields or text binary large objects (BLOBs). For short alphanumeric fields, consider using a TDBEdit (see page 8735) component instead.

If the application doesn't require the data-aware capabilities of TDBMemo, use a memo control (TMemo) instead, to conserve system resources.

See Also

TDataSet (see page 6894)

TDataSource (see page 7070)

TDBEdit (see page 8735)

TField (see page 7115)

TMemo (see page 21678)

Displaying and Editing Text in a Memo Control

Using Common Data Control Features

Navigating Datasets

47.1.10.1 TDBMemo Members

The following tables list the members exposed by TDBMemo.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.

	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
V	Change (see page 9162)	Informs the data source of changes to the value of the text, before invoking the OnChange (see page 9202) event handler.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 21248)	Initializes a window-creation parameter record passed in the Params parameter.
V	CreateParams (see page 21375)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWindowHandle (see page 21248)	Creates a Windows control to represent the control.
V	CreateWindowHandle (see page 21376)	Creates a window based on the window-creation parameter record.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CreateWnd (see page 21248)	Creates a Windows control corresponding to the edit control.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21251)	Destroys the Windows control that corresponds to the edit control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoSetMaxLength (see page 21251)	Sets the MaxLength (see page 21263) property.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetCaretPos (see page 21376)	Returns the value of the CaretPos (see page 21380) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.

	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelLength (see page 21252)	Returns the SelLength (see page 21265) property.
	GetSelStart (see page 21252)	Returns the SelStart (see page 21269) property.
	GetSelText (see page 21252)	Returns the SelText (see page 21272) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 9163)	Overrides the inherited method to put the dataset in an editing state if the user types Del or Shift+Insert.
	KeyPress (see page 9164)	KeyPress overrides the inherited method to handle any keystrokes that alter the state of the edit control.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 9164)	Initializes the memo control when it is first loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 9165)	Overrides the inherited method to respond to opRemove notifications on the DataSource (see page 9178).
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.

	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAlignment (see page 21377)	Sets the Alignment (see page 21379) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetAutoSize (see page 21257)	Sets the AutoSize (see page 21260) property.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetCaretPos (see page 21378)	Sets the CaretPos (see page 21380) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetLines (see page 21378)	Sets the Lines (see page 21380) property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetScrollBars (see page 21378)	Sets the ScrollBars (see page 21381) property.
	SetSelLength (see page 21257)	Sets the SelLength (see page 21265) property.
	SetSelStart (see page 21257)	Sets the SelStart (see page 21269) property.
	SetWordWrap (see page 21379)	Sets the WordWrap (see page 21384) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.

	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 9166)	Provides specific message responses for the memo control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSelect (see page 21260)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AutoSize (see page 21260)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	CharCase (see page 21262)	Determines the case of the text within the edit control.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	OEMConvert (see page 21264)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	PasswordChar (see page 21265)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.

	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 21245)	Deletes all text from the edit control.
	ClearSelection (see page 21245)	Removes the selected text from the edit control.
	ClearUndo (see page 21246)	Clears the undo buffer so that no changes to the text can be backed out.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopyToClipboard (see page 21246)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
	Create (see page 9161)	Creates an instance of TDBMemo (see page 9149).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CutToClipboard (see page 21249)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 21250)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 9162)	Destroys an instance of TDBMemo (see page 9149).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 9163)	Invokes an action with the TDBMemo (see page 9149) as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.

	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
V	GetControlsAlignment (see page 21376)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
V	GetSelTextBuf (see page 21253)	Copies the selected text into a buffer and returns the number of characters copied.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
V	LoadMemo (see page 9164)	LoadMemo loads the text BLOB from the field in the database into the memo control.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	PasteFromClipboard (see page 21254)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.

	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectAll (see page 21255)	Selects all text in the edit control.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSelTextBuf (see page 21258)	Replaces the selected text with a null-terminated string.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Undo (see page 21259)	Backs out all changes in the undo buffer.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
V	UpdateAction (see page 9165)	Updates an action to reflect the current state of the TDBMemo (see page 9149).
	UpdateControlState (see page 6186)	Respond to state change.
V	UseRightToLeftAlignment (see page 9166)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	CanUndo (see page 21262)	Indicates whether the edit control contains changes that can be backed out.
	CaretPos (see page 21380)	Indicates the position of the caret in the client area of the memo.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.

R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Field (see page 9181)	Identifies the TField component the database memo is linked to.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	Lines (see page 21380)	Contains the individual lines of text in the memo control.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Modified (see page 21263)	Indicates whether the user edited the text of the edit control.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelLength (see page 21265)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 21269)	Specifies the position of the first selected character in the text.
	SelText (see page 21272)	Specifies the selected portion of the edit control's text.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	Text (see page 21273)	Contains a text string associated with the control.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 9202)	Occurs when the text for the edit control may have changed.

	OnClick (see page 9202)	Occurs when the user clicks the control.
	OnContextPopup (see page 9204)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 9205)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 9205)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 9207)	Occurs when the user drags an object over a control.
	OnEndDock (see page 9210)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 9210)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 9211)	Occurs when a control receives the input focus.
	OnExit (see page 9214)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 9215)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 9218)	Occurs when a key is pressed.
	OnKeyUp (see page 9219)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 9221)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 9222)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 9224)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 9224)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 9225)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 9227)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 9229)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 9230)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 9168)	Determines how the control aligns within its container (parent control).
	Alignment (see page 9171)	Determines how the text is aligned within the memo control.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 9172)	Specifies how the control is anchored to its parent.
	AutoDisplay (see page 9172)	Determines whether to automatically display the contents of a memo or graphic BLOB in a database memo.
	BevelEdges (see page 9173)	Specifies which edges of the control are beveled.
	BevelInner (see page 9173)	Specifies the cut of the inner bevel.
	BevelKind (see page 9174)	Specifies the control's bevel style.
	BevelOuter (see page 9174)	Specifies the cut of the outer bevel.
	BevelWidth (see page 9175)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 9175)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 9175)	Determines whether the edit control has a single line border around the client area.
	Color (see page 9176)	Specifies the background color of the control.
	Constraints (see page 9177)	Specifies the size constraints for the control.
	Ctl3D (see page 9177)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DataField (see page 9178)	Specifies the name of the field for which the database memo displays data.
	DataSource (see page 9178)	Links the memo control to the dataset that contains the field for which the memo control displays data.
	DragCursor (see page 9178)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 9179)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 9179)	Determines how the control initiates drag-and-drop or drag-and-dock operations.

	Enabled (see page 9181)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 9182)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	HideSelection (see page 9183)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 9183)	Determines the behavior of the input method editor (IME).
	ImeName (see page 9184)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	MaxLength (see page 9184)	Specifies the maximum number of characters the user can enter into the edit control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 9185)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 9185)	Determines where a control looks for its color information.
	ParentCtl3D (see page 9186)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 9186)	Determines where a control looks for its font information.
	ParentShowHint (see page 9187)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 9187)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 9192)	Determines whether the user can change the contents of the database memo.
	ScrollBars (see page 9192)	Determines whether the memo control has scroll bars.
	ShowHint (see page 9194)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 9196)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 9196)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 9197)	Determines whether the component appears onscreen.
	WantReturns (see page 9198)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 9199)	Determines whether the user can insert tab characters into the text.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	WordWrap (see page 9199)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

47.1.10.2 TDBMemo.Create Constructor

Creates an instance of TDBMemo (see page 9149).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDBMemo(TComponent * AOwner);
```

Description

Call Create to create and initialize an instance of TDBMemo (see page 9149). Create overrides the inherited constructor to initialize the data handling aspects of a TDBMemo (see page 9149).

AOwner is the component, typically the form, that is responsible for freeing the DB (see page 6725) memo instance. It becomes the value of the Owner property.

47.1.10.3 TDBMemo.Destroy Destructor

Destroys an instance of TDBMemo (see page 9149).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBMemo();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TDBMemo (see page 9149) reference is not nil and only then calls Destroy.

Destroy frees up the memory associated with data handling that was allocated in the constructor, before calling the inherited destructor.

See Also

Free (see page 22300)

47.1.10.4 TDBMemo Methods

The methods of the TDBMemo class are listed here.

Protected Methods

	Name	Description
	Change (see page 9162)	Informs the data source of changes to the value of the text, before invoking the OnChange (see page 9202) event handler.
	KeyDown (see page 9163)	Overrides the inherited method to put the dataset in an editing state if the user types Del or Shift+Insert.
	KeyPress (see page 9164)	KeyPress overrides the inherited method to handle any keystrokes that alter the state of the edit control.
	Loaded (see page 9164)	Initializes the memo control when it is first loaded into memory.
	Notification (see page 9165)	Overrides the inherited method to respond to opRemove notifications on the DataSource (see page 9178).
	WndProc (see page 9166)	Provides specific message responses for the memo control.

Public Methods

	Name	Description
	ExecuteAction (see page 9163)	Invokes an action with the TDBMemo (see page 9149) as its target.
	LoadMemo (see page 9164)	LoadMemo loads the text BLOB from the field in the database into the memo control.
	UpdateAction (see page 9165)	Updates an action to reflect the current state of the TDBMemo (see page 9149).
	UseRightToLeftAlignment (see page 9166)	Specifies whether the control's alignment should be reversed right-to-left.

47.1.10.4.1 TDBMemo.Change Method

Informs the data source of changes to the value of the text, before invoking the OnChange (see page 9202) event handler.

Pascal

```
procedure Change; override;
```

C++

```
virtual __fastcall Change();
```

Description

Applications do not call the Change method directly: it is triggered automatically by Windows events. Derived classes can override the Change method to make additional responses to changes in the Text property. Any class that overrides the Change method should call the inherited method as well.

See Also

OnChange (see page 21275)

47.1.10.4.2 TDBMemo.ExecuteAction Method

Invokes an action with the TDBMemo (see page 9149) as its target.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

ExecuteAction is called automatically when the user invokes an action with the TDBMemo (see page 9149) as a target. The Action parameter specifies the action that was invoked. ExecuteAction returns trueif the action was successfully dispatched, and false if the TDBMemo (see page 9149) could not handle the action.

See Also

UpdateAction (see page 9165)

Execute (see page 2712)

HandlesTarget (see page 2713)

47.1.10.4.3 TDBMemo.KeyDown Method

Overrides the inherited method to put the dataset in an editing state if the user types Del or Shift+Insert.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

KeyDown first calls the inherited method, which calls the OnKeyDown (see page 9215) event handler. KeyDown next attempts to put the dataset into an editing state when the user types Del or Shift+Insert. If the memo control is not linked to a dataset, KeyDown discards the keystroke. Derived classes can override KeyDown to handle additional key combinations, but the inherited method should always be called.

See Also

DataSource (see page 9178)

KeyPress (see page 9164)

OnKeyDown (see page 6225)

47.1.10.4.4 TDBMemo.KeyPress Method

KeyPress overrides the inherited method to handle any keystrokes that alter the state of the edit control.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress first calls the inherited method, which calls the OnKeyPress (see page 9218) event handler. Next, KeyPress checks the value of the key, and throws it away if it is not a valid character for the field type, or if there is no link to a database field. Valid characters cause KeyPress to attempt to put the dataset into edit mode.

If the field for the memo control is a text BLOB, pressing the Pause key causes KeyPress to call the LoadMemo (see page 9164) method. Pressing the Escape key causes KeyPress to throw away unposted edits.

See Also

AutoDisplay (see page 9172)

LoadMemo (see page 9164)

OnKeyPress (see page 6227)

47.1.10.4.5 TDBMemo.Loaded Method

Initializes the memo control when it is first loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Loaded overrides the inherited method so that the Modified property is set to false when the form containing the memo is first loaded into memory. Derived classes can override the Loaded method to perform additional initializations after the memo's form is loaded.

See Also

Modified (see page 21263)

47.1.10.4.6 TDBMemo.LoadMemo Method

LoadMemo loads the text BLOB from the field in the database into the memo control.

Pascal

```
procedure LoadMemo; virtual;
```

C++

```
virtual __fastcall LoadMemo();
```

Description

LoadMemo allows an application to control when the memo control displays the actual contents of a text BLOB field. Use LoadMemo to update the text in the memo control to the value of the field in the current record. LoadMemo only works with text BLOB fields, when the AutoDisplay (see page 9172) property is false, so that the value of the field is not loaded automatically.

See Also

AutoDisplay (see page 9172)

Field (see page 9181)

OnKeyPress

Text (see page 5842)

47.1.10.4.7 TDBMemo.Notification Method

Overrides the inherited method to respond to opRemove notifications on the DataSource (see page 9178).

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

After calling the inherited method, Notification checks whether the notification indicates the removal of the DataSource (see page 9178). If so, Notification sets the DataSource (see page 9178) property to nil (Delphi) or NULL (C++).

Derived classes can override Notification method to handle additional notifications, but the inherited method should be always be called.

See Also

DataSource (see page 9178)

47.1.10.4.8 TDBMemo.UpdateAction Method

Updates an action to reflect the current state of the TDBMemo (see page 9149).

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

UpdateAction is called automatically when the application is idle to give the TDBMemo (see page 9149) an opportunity to update associated actions. This allows actions to be grayed, checked, or to otherwise reflect the status of the TDBMemo (see page 9149) they target.

The Action parameter specifies the action component that should be updated. UpdateAction returns trueif the action component now reflects the state of the component, and false if it did not know how to update the action.

See Also

ExecuteAction (see page 9163)

UpdateTarget (see page 2715)

HandlesTarget (see page 2713)

47.1.10.4.9 TDBMemo.UseRightToLeftAlignment Method

Specifies whether the control's alignment should be reversed right-to-left.

Pascal

```
function UseRightToLeftAlignment: Boolean; override;
```

C++

```
virtual __fastcall Boolean UseRightToLeftAlignment();
```

Description

Controls call UseRightToLeftAlignment when implementing the BiDiMode property. UseRightToLeftAlignment indicates whether the control's alignment should be reversed right-to-left. This information can be used when painting the control or when responding to mouse events.

UseRightToLeftAlignment returns true if the application is running on a system with a Middle Eastern locale and the BiDiMode property is bdRightToLeft. Otherwise, it returns false. Descendants override UseRightToLeftAlignment if alignment in Middle Eastern locales depends on other factors.

See Also

BiDiMode (see page 5803)

GetControlsAlignment (see page 5759)

Text (see page 5842)

Caption (see page 5806)

47.1.10.4.10 TDBMemo.WndProc Method

Provides specific message responses for the memo control.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc overrides the WndProc for TWinControl to handle window messages that are specific to painting a TDBMemo (see page 9149) control, and sends all others to the inherited WndProc.

Derived classes that override WndProc to provide specialized responses to messages should call the inherited WndProc at the end to dispatch any other messages.

See Also

WndProc (see page 6187)

47.1.10.5 TDBMemo Properties

The properties of the TDBMemo class are listed here.

Public Properties

	Name	Description
R	Field (see page 9181)	Identifies the TField component the database memo is linked to.

Published Properties

	Name	Description
	Align (see page 9168)	Determines how the control aligns within its container (parent control).
	Alignment (see page 9171)	Determines how the text is aligned within the memo control.
	Anchors (see page 9172)	Specifies how the control is anchored to its parent.
	AutoDisplay (see page 9172)	Determines whether to automatically display the contents of a memo or graphic BLOB in a database memo.
	BevelEdges (see page 9173)	Specifies which edges of the control are beveled.
	BevelInner (see page 9173)	Specifies the cut of the inner bevel.
	BevelKind (see page 9174)	Specifies the control's bevel style.
	BevelOuter (see page 9174)	Specifies the cut of the outer bevel.
	BevelWidth (see page 9175)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 9175)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 9175)	Determines whether the edit control has a single line border around the client area.
	Color (see page 9176)	Specifies the background color of the control.
	Constraints (see page 9177)	Specifies the size constraints for the control.
	Ctl3D (see page 9177)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DataField (see page 9178)	Specifies the name of the field for which the database memo displays data.
	DataSource (see page 9178)	Links the memo control to the dataset that contains the field for which the memo control displays data.
	DragCursor (see page 9178)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 9179)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 9179)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 9181)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 9182)	Controls the attributes of text written on or in the control.
	HideSelection (see page 9183)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 9183)	Determines the behavior of the input method editor (IME).
	ImeName (see page 9184)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	MaxLength (see page 9184)	Specifies the maximum number of characters the user can enter into the edit control.
	ParentBiDiMode (see page 9185)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 9185)	Determines where a control looks for its color information.
	ParentCtl3D (see page 9186)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 9186)	Determines where a control looks for its font information.
	ParentShowHint (see page 9187)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 9187)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 9192)	Determines whether the user can change the contents of the database memo.
	ScrollBars (see page 9192)	Determines whether the memo control has scroll bars.
	ShowHint (see page 9194)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 9196)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 9196)	Determines if the user can tab to a control.

	Visible (see page 9197)	Determines whether the component appears onscreen.
	WantReturns (see page 9198)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 9199)	Determines whether the user can insert tab characters into the text.
	WordWrap (see page 9199)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

47.1.10.5.1 TDBMemo.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
```

```
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
```

```
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
```

```
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

47.1.10.5.2 TDBMemo.Alignment Property

Determines how the text is aligned within the memo control.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to change the way the text is formatted by the memo control. Alignment can take one of the following values:

Value	Meaning
taLeftJustify	Align text to the left side of the control
taCenter	Center text horizontally in the control.
taRightJustify	Align text to the right side of the control.

See Also

SetAlignment (see page 21377)

Text (see page 5842)

Displaying and Editing Text in a Memo Control

47.1.10.5.3 TDBMemo.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

47.1.10.5.4 TDBMemo.AutoDisplay Property

Determines whether to automatically display the contents of a memo or graphic BLOB in a database memo.

Pascal

```
property AutoDisplay: Boolean;
```

C++

```
__property Boolean AutoDisplay;
```

Description

If AutoDisplay is true, the control automatically displays new data when the underlying BLOB field changes (such as when moving to a new record).

If AutoDisplay is false, the control shows only the field name whenever the underlying BLOB field changes. To display the data, the user can double-click on the control or select it and press Enter.

The effect of AutoDisplay is not purely cosmetic. The data shown by the TDBMemo (see page 9149) is the value of the Text property. When AutoDisplay is false, if the data changes, the value of the Text property changes to the name of the field. Thus, if

AutoDisplay is false, applications should be cautious about using the value of the Text property to ascertain the value of the underlying field.

Calling the LoadMemo (see page 9164) method causes Text to update to the current value of the BLOB field. This change will also be reflected in the appearance of the control on screen.

Change (see page 9162) the value of AutoDisplay to false if the automatic loading of BLOB fields takes too long.

Note: AutoDisplay only applies to BLOB fields. If the control is bound to another type of text field, AutoDisplay has no effect.

See Also

DataField (see page 9178)

Field (see page 9181)

LoadMemo (see page 9164)

Text

Displaying and Editing Text in a Memo Control

47.1.10.5.5 TDBMemo.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

47.1.10.5.6 TDBMemo.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

47.1.10.5.7 TDBMemo.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

47.1.10.5.8 TDBMemo.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

47.1.10.5.9 TDBMemo.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

47.1.10.5.10 TDBMemo.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

47.1.10.5.11 TDBMemo.BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the edit control stands out. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the edit control has a single-line border around the client area. If BorderStyle is bsNone, there will be no border.

47.1.10.5.12 TDBMemo.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

47.1.10.5.13 TDBMemo.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

47.1.10.5.14 TDBMemo.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

47.1.10.5.15 TDBMemo.DataField Property

Specifies the name of the field for which the database memo displays data.

Pascal

```
property DataField: WideString;
```

C++

```
__property BSTR DataField;
```

Description

Use DataField to link the memo control to a field in the dataset. Just setting DataField by itself will not link the memo control to a database field. Additionally, the dataset which contains the field must be specified by setting the DataSource (see page 9178) property.

```
DBMemo1.DataField := 'Notes';
DBMemo1->DataField = "Notes";
```

See Also

DataSource (see page 9178)

Field (see page 9181)

Text (see page 5842)

47.1.10.5.16 TDBMemo.DataSource Property

Links the memo control to the dataset that contains the field for which the memo control displays data.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Link the memo control with a dataset by setting DataSource to a data source component that identifies the desired dataset. Specify which field in the dataset contains the memo data using the DataField (see page 9178) property.

```
DBMemo1.DataSource := DataSource1;
DBMemo1->DataSource = DataSource1;
```

See Also

DataField (see page 9178)

47.1.10.5.17 TDBMemo.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

47.1.10.5.18 TDBMemo.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

47.1.10.5.19 TDBMemo.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
```

```
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

47.1.10.5.20 TDBMemo.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

47.1.10.5.21 TDBMemo.Field Property

Identifies the TField component the database memo is linked to.

Pascal

```
property Field: TField;
```

C++

```
__property TField Field;
```

Description

Read Field to get direct access to the contents and properties of the database field without going through the edit window. Use Field to change the contents of the database field programmatically.

Use the Text property instead to change the contents of the memo control without immediately writing the changes to the record in the dataset. Changing the Text property allows the user to cancel the changes.

```
(DBMemo1.Field as TMemoField).Assign(Memo1.Lines);
```

See Also

DataField (see page 9178)

Text (see page 5842)

47.1.10.5.22 TDBMemo.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
```

```
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

47.1.10.5.23 TDBMemo.HideSelection Property

Determines whether the visual indication of the selected text remains when focus shifts to another control.

Pascal

```
property HideSelection: Boolean;
```

C++

```
__property Boolean HideSelection;
```

Description

Set HideSelection to false to provide visual feedback of the selected portion of the text even when the edit control does not have focus. Set HideSelection to true to show the selection only when the edit control has focus. HideSelection does not affect the actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms with many edit controls look too busy.

See Also

SelLength (see page 21265)

SelStart (see page 21269)

SelText (see page 21272)

SetSelTextBuf (see page 21258)

47.1.10.5.24 TDBMemo.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

47.1.10.5.25 TDBMemo.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

47.1.10.5.26 TDBMemo.MaxLength Property

Specifies the maximum number of characters the user can enter into the edit control.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the edit control. A value of 0 indicates that there is no application-defined limit on the length.

Use MaxLength to limit the length of the text in an edit control if that text will be copied into a fixed-length buffer.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

Note: The unit of MaxLength is characters (not bytes) if the application is using themes.

See Also

Text (see page 5842)

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

47.1.10.5.27 TDBMemo.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

47.1.10.5.28 TDBMemo.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its

own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

47.1.10.5.29 TDBMemo.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

47.1.10.5.30 TDBMemo.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

47.1.10.5.31 TDBMemo.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

47.1.10.5.32 TDBMemo.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
```

```
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
```

```
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
```

```
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
```

```
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

47.1.10.5.33 TDBMemo.ReadOnly Property

Determines whether the user can change the contents of the database memo.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

If ReadOnly is true, the user can't change the contents. If ReadOnly is false, the user can modify the contents.

The ReadOnly property determines whether the user can use the database memo to change the value of the field of the current record, or if the control only displays the value. If ReadOnly is false, the user can change the field's value as long as the dataset is in edit mode.

See Also

Text (see page 5842)

Displaying and Editing Text in a Memo Control

47.1.10.5.34 TDBMemo.ScrollBars Property

Determines whether the memo control has scroll bars.

Pascal

```
property ScrollBars: TScrollStyle;
```

C++

```
__property TScrollStyle ScrollBars;
```

Description

Use ScrollBars to give a multiline edit control horizontal or vertical scroll bars. ScrollBars can take one of the following values:

Value	Meaning
ssNone	The control has no scroll bars.
ssHorizontal	The control has a single scroll bar on the bottom edge.
ssVertical	The control has a single scroll bar on the right edge.
ssBoth	The control has a scroll bar on both the bottom and right edges.

C++ Examples:

```
/*
```

```
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
```

```
    RichEdit1.SelStart := 0;
  end;

  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    RichEdit1.HideScrollbars := CheckBox1.Checked;
  end;

  procedure TForm1.RadioGroup1Click(Sender: TObject);
  const
    ScrollBarA: array[0..3] of TScrollStyle = (
      ssBoth,
      ssHorizontal,
      ssNone,
      ssVertical);
  begin
    RichEdit1.ScrollBars :=
      ScrollBarA[RadioGroup1.ItemIndex];
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with RadioGroup1, Items do
    begin
      Caption := 'Scroll Bars';
      Clear;
      Add('ssBoth');
      Add('ssHorizontal');
      Add('ssNone');
      Add('ssVertical');
    end;
    with CheckBox1 do
    begin
      Caption := 'HideScrollBars';
      Checked := True;
    end;
    Button1.Caption := 'Loaded';
    RichEdit1.Text := 'Just a little text.';
    RichEdit1.WordWrap := False;
  end;
```

See Also

SetScrollBars (see page 21378)

WordWrap (see page 21384)

Displaying and Editing Text in a Memo Control

47.1.10.5.35 TDBMemo.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
```

```
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

47.1.10.5.36 TDBMemo.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

47.1.10.5.37 TDBMemo.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

47.1.10.5.38 TDBMemo.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

47.1.10.5.39 TDBMemo.WantReturns Property

Determines whether the user can insert return characters into the text.

Pascal

```
property WantReturns: Boolean;
```

C++

```
__property Boolean WantReturns;
```

Description

Set WantReturns to true to allow users to enter return characters into the text. Set WantReturns to false to allow the form to handle return characters instead.

For example, in a form with a default button (such as an OK button) and a memo control, if WantReturns is false, pressing Enter chooses the default button. If WantReturns is true, pressing Enter inserts a return character in the text.

Note: If WantReturns is false, users can still enter return characters into the text by pressing Ctrl+Enter.

See Also

Text (see page 5842)

WantTabs (see page 21384)

WordWrap (see page 21384)

47.1.10.5.40 TDBMemo.WantTabs Property

Determines whether the user can insert tab characters into the text.

Pascal

```
property WantTabs: Boolean;
```

C++

```
__property Boolean WantTabs;
```

Description

Set WantTabs to true to allow users to enter tab characters into the text. Set WantTabs to false if you want the tab character to select the next control on the form instead.

Note: If WantTabs is true, users can tab into the edit control, but they can't tab out.

See Also

Text (see page 5842)

WantReturns (see page 21384)

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

47.1.10.5.41 TDBMemo.WordWrap Property

Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to make the edit control wrap text at the right margin so it fits in the client area. The wrapping is cosmetic only. The text does not include any return characters that were not explicitly entered. Set WordWrap to false to have the edit control show a separate line only where return characters were explicitly entered into the text.

Note: There should be no use for a horizontal scroll bar if WordWrap is true.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
```

```
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;
```

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
    ssHorizontal,
    ssNone,
    ssVertical);
begin
  RichEdit1.ScrollBars :=
    ScrollBarA[RadioGroup1.ItemIndex];
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with RadioGroup1, Items do
  begin
    Caption := 'Scroll Bars';
    Clear;
    Add('ssBoth');
    Add('ssHorizontal');
    Add('ssNone');
    Add('ssVertical');
  end;
  with CheckBox1 do
  begin
    Caption := 'HideScrollBars';
    Checked := True;
  end;
  Button1.Caption := 'Loaded';
  RichEdit1.Text := 'Just a little text.';
  RichEdit1.WordWrap := False;
end;
```

See Also

Lines (see page 21380)

ScrollBars (see page 21381)

Text (see page 5842)

WantReturns (see page 21384)

Displaying and Editing Text in a Memo Control

47.1.10.6 TDBMemo Events

The events of the TDBMemo class are listed here.

Published Events

	Name	Description
	OnChange (see page 9202)	Occurs when the text for the edit control may have changed.
	OnClick (see page 9202)	Occurs when the user clicks the control.
	OnContextPopup (see page 9204)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 9205)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 9205)	Occurs when the user drops an object being dragged.

	OnDragOver (see page 9207)	Occurs when the user drags an object over a control.
	OnEndDock (see page 9210)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 9210)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 9211)	Occurs when a control receives the input focus.
	OnExit (see page 9214)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 9215)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 9218)	Occurs when a key is pressed.
	OnKeyUp (see page 9219)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 9221)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 9222)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 9224)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 9224)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 9225)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 9227)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 9229)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 9230)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

47.1.10.6.1 TDBMemo.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change (see page 21245)

Text (see page 5842)

47.1.10.6.2 TDBMemo.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an

OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
```

```
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

47.1.10.6.3 TDBMemo.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the

default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

47.1.10.6.4 TDBMemo.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

47.1.10.6.5 TDBMemo.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
```

```
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

47.1.10.6.6 TDBMemo.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.10.6.7 TDBMemo.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.10.6.8 TDBMemo.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
```

```
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

47.1.10.6.9 TDBMemo.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter

event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
```

```
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
```

```
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

47.1.10.6.10 TDBMemo.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

47.1.10.6.11 TDBMemo.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
```

```
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
```

```
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

47.1.10.6.12 TDBMemo.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not

generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

47.1.10.6.13 TDBMemo.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
```

```
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
```

```
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

47.1.10.6.14 TDBMemo.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate

event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.10.6.15 TDBMemo.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
```

```
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
```

```
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.10.6.16 TDBMemo.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

47.1.10.6.17 TDBMemo.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

47.1.10.6.18 TDBMemo.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.10.6.19 TDBMemo.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
```

```
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
```

```
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

47.1.10.6.20 TDBMemo.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

47.1.10.6.21 TDBMemo.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

47.1.11 TDBNavigator Class

TDBNavigator (the database navigator) is used to move through the data in a dataset and perform operations on the data, such as inserting a blank record or posting a record.

Class Hierarchy

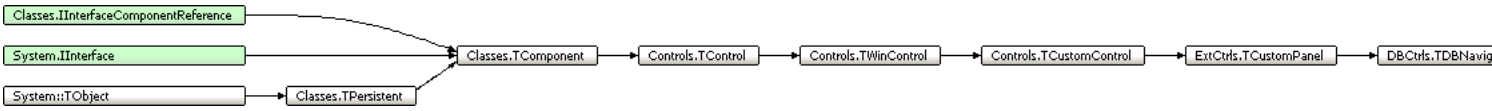

Pascal

```
TDBNavigator = class(TCustomPanel);
```

C++

```
class TDBNavigator : public TCustomPanel;
```

File

DBCtrls

Description

Use the database navigator on forms that contain data-aware controls, such as TDBGrid or TDBEdit (see page 8735). TDBNavigator lets the user control the dataset when editing or viewing the data.

When the user chooses one of the navigator buttons, the appropriate action occurs on the dataset to which the navigator is linked. For example, if the user clicks the Insert button, a blank record is inserted in the dataset.

TDBNavigator can show any or all of the following buttons:

Button	Purpose
First	Sets the current record to the first record in the dataset, disables the First and Prior buttons, and enables the Next and last buttons if they are disabled.
Prior	Sets the current record to the previous record and enables the Last and Next buttons if they are disabled.
Next	Sets the current record to the next record and enables the First and Prior buttons if they are disabled.
Last	Sets the current record to the last record in the dataset, disables the Last and Next buttons, and enables the First and Prior buttons if they are disabled.
Insert	Inserts a new record before the current record, and sets the dataset into Insert and Edit states.
Delete	Deletes the current record and makes the next record the current record.
Edit	Puts the dataset into Edit state so that the current record can be modified.
Post	Writes changes in the current record to the database.
Cancel	Cancels edits to the current record, restores the record display to its condition prior to editing, and turns off Insert and Edit states if they are active.
Refresh	Refreshes the buffered data in the associated dataset.

Delphi Examples:

```
{
This example requires two ClientDataSets, two DBGrids, two
DataSources and two DBNavigators.  Name the two
ClientDataSets CDS and CDS2. Set the DataSet property in the
first DataSource to CDS and name that DataSource DS.  Set the
DataSet property in the second DataSource to CDS2 and name
that DataSource DS2.  Set the DataSource in the first
DBNavigator to DS and the DataSource in the second DBNavigator
to DS2.  Set the DataSource in the first DBGrid to DS and the
DataSource in the second DBGrid to DS2.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
```

```
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
type to Integer.  The file used to load CDS must have an
attribute name of HOST with an attribute type of string and
an attribute name of PORTA with an attribute type of integer.
Add DB, DBCtrls, Grids, DBGrids, DBClient and SwSystem to the
uses clause.  "LoadFormFile" to start with the file, or
"CreateDataSet" to start from scratch.  You must "CloseDataSet"
after a "LoadFromFile" before you can do another "CreateDataSet".
}
{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button1Click(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.SaveToFile(GetCurrentDir + '\CDS.XML', dfXML);
{$ELSE}
  CDS.SaveToFile(gsAppPath + 'CDS.XML', dfXML);
{$ENDIF}
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  CDS.CreateDataSet;
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  CDS2.Data := CDS.Data;
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  CDS.Close;
end;
```

See Also

TDataSource (see page 7070)

TDataSet (see page 6894)

TDBGrid (see page 9668)

TDBEdit (see page 8735)

Choosing How to Organize the Data

Choosing Navigator Buttons to Display

Enabling Editing in Controls On User Entry

Using Common Data Control Features

Navigating and Manipulating Records

Using the First and Last Methods

Navigating Datasets

47.1.11.1 TDBNavigator Members

The following tables list the members exposed by TDBNavigator.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	Buttons (see page 9244)	This is Buttons, a member of class TDBNavigator.
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.

	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	ActiveChanged (see page 9245)	This is ActiveChanged, a member of class TDBNavigator.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustClientRect (see page 12537)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcMinSize (see page 9245)	This is CalcMinSize, a member of class TDBNavigator.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanAutoSize (see page 12537)	Adjusts the way the panel responds to resize attempts.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 12537)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DataChanged (see page 9246)	This is DataChanged, a member of class TDBNavigator.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.

	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EditingChanged (see page 9246)	This is EditingChanged, a member of class TDBNavigator.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 9246)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.

	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 9247)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 9247)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 9248)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 5920)	Renders the image of a custom control.
	Paint (see page 12538)	Renders the image of the panel.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.

	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBackground (see page 12539)	This is SetParentBackground, a member of class TCustomPanel.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.

V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	Alignment (see page 12539)	Determines how the caption is aligned within the panel.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelInner (see page 12540)	Determines the style of the inner bevel of a panel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelOuter (see page 12541)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BevelWidth (see page 12542)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BorderStyle (see page 12543)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	BorderWidth (see page 12543)	Specifies the distance, in pixels, between the outer and inner bevels.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Color (see page 12544)	Specifies the background color of the control.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	FullRepaint (see page 12545)	Determines how the panel repaints itself when it is resized.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	Locked (see page 12545)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 12546)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	VerticalAlignment (see page 12547)	Sets the vertical position of the caption.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.

	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	BtnClick (see page 9245)	Simulates a button click on the database navigator, invoking the action of the button.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 9243)	Creates an instance of TDBNavigator (see page 9230).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 9244)	Destroys an instance of TDBNavigator (see page 9230).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
V	GetControlsAlignment (see page 12538)	Returns the alignment of text in the panel.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.

V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 9248)	Sets the database navigator's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.

	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentBackground (see page 12546)	Determines whether control uses parent's theme background.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	BeforeAction (see page 9269)	Occurs when a button on the database navigator is clicked, before the action is executed.
	OnClick (see page 9269)	Occurs when a button on the database navigator is clicked, after the action is executed.
	OnContextPopup (see page 9270)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 9271)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 9271)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 9273)	Occurs when the user drags an object over a control.
	OnEndDock (see page 9276)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 9276)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 9277)	Occurs when a control receives the input focus.
	OnExit (see page 9279)	Occurs when the input focus shifts away from one control to another.
	OnResize (see page 9281)	Occurs immediately after the control is resized.
	OnStartDock (see page 9282)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 9282)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 9249)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 9253)	Specifies how the control is anchored to its parent.

	ConfirmDelete (see page 9253)	Determines whether a message box appears asking the user to confirm record deletions initiated using the database navigator.
	Constraints (see page 9254)	Specifies the size constraints for the control.
	Ctl3D (see page 9254)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DataSource (see page 9255)	Links the navigator to the dataset that it controls.
	DragCursor (see page 9255)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 9256)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 9256)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 9257)	Controls whether the control responds to mouse, keyboard, and timer events.
	Flat (see page 9258)	Determines whether the buttons of the navigator have a three-dimensional (3-D) look with borders or two-dimensional look without borders.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Hints (see page 9258)	Provides a way to customize the Help Hints for the buttons on the database navigator.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentCtl3D (see page 9259)	Determines where a component looks to determine if it should appear three dimensional.
	ParentShowHint (see page 9259)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 9260)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 9264)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 9266)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 9266)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 9267)	Determines whether the component appears onscreen.
	VisibleButtons (see page 9268)	Determines which buttons appear on the database navigator.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

47.1.11.2 TDBNavigator.Create Constructor

Creates an instance of TDBNavigator (see page 9230).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDBNavigator(TComponent * AOwner);
```

Description

Call Create to instantiate TDBNavigator (see page 9230) at runtime. For TDBNavigator (see page 9230) objects placed on a form at design time, the constructor is called automatically.

Pass a single component as a parameter to provide the TDBNavigator (see page 9230) object with an owner that is

responsible for freeing it.

See Also

Destroy (see page 9244)

47.1.11.3 TDBNavigator.Destroy Destructor

Destroys an instance of TDBNavigator (see page 9230).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBNavigator();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TDBNavigator (see page 9230) reference is not nil and only then calls destroy.

See Also

Create (see page 9243)

Free (see page 22300)

47.1.11.4 TDBNavigator Fields

The fields of the TDBNavigator class are listed here.

Protected Fields

	Name	Description
	Buttons (see page 9244)	This is Buttons, a member of class TDBNavigator.

47.1.11.4.1 TDBNavigator.Buttons Field

Pascal

```
Buttons: array[TNavigateBtn] of TNavButton;
```

C++

```
array[TNavigateBtn] of TNavButton Buttons;
```

Description

This is Buttons, a member of class TDBNavigator.

47.1.11.5 TDBNavigator Methods

The methods of the TDBNavigator class are listed here.

Protected Methods

	Name	Description
	ActiveChanged (see page 9245)	This is ActiveChanged, a member of class TDBNavigator.
	CalcMinSize (see page 9245)	This is CalcMinSize, a member of class TDBNavigator.

	DataChanged (see page 9246)	This is DataChanged, a member of class TDBNavigator.
	EditingChanged (see page 9246)	This is EditingChanged, a member of class TDBNavigator.
V	GetChildren (see page 9246)	Calls a specified method for each child of the control.
V	KeyDown (see page 9247)	Respond to key press events.
V	Loaded (see page 9247)	Initializes the control after it is loaded from a stream.
V	Notification (see page 9248)	Responds to notifications that components are being created or destroyed.

Public Methods

	Name	Description
V	BtnClick (see page 9245)	Simulates a button click on the database navigator, invoking the action of the button.
V	SetBounds (see page 9248)	Sets the database navigator's boundary properties all at once.

47.1.11.5.1 TDBNavigator.ActiveChanged Method

Pascal

```
procedure ActiveChanged;
```

C++

```
__fastcall ActiveChanged();
```

Description

This is ActiveChanged, a member of class TDBNavigator.

47.1.11.5.2 TDBNavigator.BtnClick Method

Simulates a button click on the database navigator, invoking the action of the button.

Pascal

```
procedure BtnClick(Index: TNavigateBtn); virtual;
```

C++

```
virtual __fastcall BtnClick(TNavigateBtn Index);
```

Description

Use BtnClick when you want the application, not the user, to control the database navigator. Call BtnClick, specifying a TNavigateBtn (see page 9539) type as the value of the Index parameter, to invoke the command associated with the selected button.

Note: Calling BtnClick triggers the navigator's OnClick (see page 9269) event after the default action of the button.

See Also

OnClick (see page 9269)

VisibleButtons (see page 9268)

47.1.11.5.3 TDBNavigator.CalcMinSize Method

Pascal

```
procedure CalcMinSize(var W: Integer; var H: Integer);
```

C++

```
__fastcall CalcMinSize(int W, int H);
```

Description

This is CalcMinSize, a member of class TDBNavigator.

47.1.11.5.4 TDBNavigator.DataChanged Method

Pascal

```
procedure DataChanged;
```

C++

```
__fastcall DataChanged();
```

Description

This is DataChanged, a member of class TDBNavigator.

47.1.11.5.5 TDBNavigator.EditingChanged Method

Pascal

```
procedure EditingChanged;
```

C++

```
__fastcall EditingChanged();
```

Description

This is EditingChanged, a member of class TDBNavigator.

47.1.11.5.6 TDBNavigator.GetChildren Method

Calls a specified method for each child of the control.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the streaming system that loads and saves components. Applications seldom need to call this routine.

GetChildren executes the callback specified by the Proc parameter for every child control listed by the Controls property that lists the Root parameter as its Owner.

Override GetChildren to limit or augment which child controls are saved with the control. When overriding GetChildren, call the procedure passed as the Proc parameter for every child control that should be saved. The Root parameter indicates the component (usually a form) that owns the control.

See Also

Controls (see page 6197)

Owner (see page 2807)

47.1.11.5.7 TDBNavigator.KeyDown Method

Respond to key press events.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-down message (WM_KEYDOWN) from Windows, its message handler calls the DoKeyDown method. If DoKeyDown determines that the control should, in fact, process the character, it calls KeyDown, passing the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

See Also

KeyUp (see page 6164)

OnKeyDown (see page 6225)

DoKeyDown (see page 6145)

DoKeyPress (see page 6146)

DoKeyUp (see page 6146)

47.1.11.5.8 TDBNavigator.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded (see page 2782)

Action (see page 5797)

ActionChange (see page 5737)

47.1.11.5.9 TDBNavigator.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

47.1.11.5.10 TDBNavigator.SetBounds Method

Sets the database navigator's boundary properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Call SetBounds when to set more than one of the navigator's boundary properties at a time. The Left, Top, Width, and Height properties are set to the values passed in the ALeft, ATop, AWidth, and AHeight parameters, respectively. SetBounds arranges the buttons on the navigator to fit within the boundaries.

Although you can always set the individual boundaries, using SetBounds makes several changes at once without rearranging the buttons for each change.

See Also

Height (see page 5825)

Left (see page 5830)

Top (see page 5844)

Width (see page 5849)

47.1.11.6 TDBNavigator Properties

The properties of the TDBNavigator class are listed here.

Published Properties

	Name	Description
	Align (see page 9249)	Determines how the control aligns within its container (parent control).
	Anchors (see page 9253)	Specifies how the control is anchored to its parent.
	ConfirmDelete (see page 9253)	Determines whether a message box appears asking the user to confirm record deletions initiated using the database navigator.
	Constraints (see page 9254)	Specifies the size constraints for the control.
	Ctl3D (see page 9254)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DataSource (see page 9255)	Links the navigator to the dataset that it controls.
	DragCursor (see page 9255)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 9256)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 9256)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 9257)	Controls whether the control responds to mouse, keyboard, and timer events.
	Flat (see page 9258)	Determines whether the buttons of the navigator have a three-dimensional (3-D) look with borders or two-dimensional look without borders.
	Hints (see page 9258)	Provides a way to customize the Help Hints for the buttons on the database navigator.
	ParentCtl3D (see page 9259)	Determines where a component looks to determine if it should appear three dimensional.
	ParentShowHint (see page 9259)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 9260)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 9264)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 9266)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 9266)	Determines if the user can tab to a control.
	Visible (see page 9267)	Determines whether the component appears onscreen.
	VisibleButtons (see page 9268)	Determines which buttons appear on the database navigator.

47.1.11.6.1 TDBNavigator.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
```

```
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
```

```
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
```

```
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

47.1.11.6.2 TDBNavigator.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

47.1.11.6.3 TDBNavigator.ConfirmDelete Property

Determines whether a message box appears asking the user to confirm record deletions initiated using the database navigator.

Pascal

```
property ConfirmDelete: Boolean;
```

C++

```
__property Boolean ConfirmDelete;
```

Description

Use the ConfirmDelete property to help prevent the user from accidentally deleting a record from the dataset.

If ConfirmDelete is true, a message box appears when the user presses the delete button and the record isn't deleted unless the user chooses the OK button. If ConfirmDelete is false, no message box appears and the record is simply deleted.

See Also

DataSource (see page 9255)

VisibleButtons (see page 9268)

47.1.11.6.4 TDBNavigator.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

47.1.11.6.5 TDBNavigator.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

47.1.11.6.6 TDBNavigator.DataSource Property

Links the navigator to the dataset that it controls.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Use the DataSource property to link the database navigator with a dataset. Specify the data source component that identifies the dataset that the navigator manipulates.

See Also

TDataSource (see page 7070)

TNavDataLink (see page 9513)

Using a Single Navigator for Multiple Datasets

47.1.11.6.7 TDBNavigator.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

47.1.11.6.8 TDBNavigator.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

47.1.11.6.9 TDBNavigator.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
```

```
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

47.1.11.6.10 TDBNavigator.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

47.1.11.6.11 TDBNavigator.Flat Property

Determines whether the buttons of the navigator have a three-dimensional (3-D) look with borders or two-dimensional look without borders.

Pascal

```
property Flat: Boolean;
```

C++

```
__property Boolean Flat;
```

Description

When Flat is true, the navigator buttons appear flat and do not have borders separating them. When Flat is false, the buttons are clearly defined.

47.1.11.6.12 TDBNavigator.Hints Property

Provides a way to customize the Help Hints for the buttons on the database navigator.

Pascal

```
property Hints: TStrings;
```

C++

```
__property TStrings Hints;
```

Description

Use the Hints property to supply Help Hints of your choosing for the individual navigator buttons. Each button has a default Help Hint. Hints allow the values of any or all of these default Help Hints to be replaced by customized hints.

Hints is a string list. Each hint is a string. The first string in the string list becomes the Help Hint for the first button on the navigator (the First button). The seventh hint becomes the Help Hint for the seventh button (the Edit button).

When specifying Hints at runtime, enter an empty string for any Help Hint that should keep the default value. Simply leave the line blank when using the string list property editor of the Object Inspector for the Hints property.

To have the Help Hints appear at runtime, set the ShowHint (see page 9264) property to true.

See Also

ShowHint (see page 5840)

Displaying Fly-over Help

47.1.11.6.13 TDBNavigator.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

47.1.11.6.14 TDBNavigator.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

47.1.11.6.15 TDBNavigator.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
```

```
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
```

```
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
```

```
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

47.1.11.6.16 TDBNavigator.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

47.1.11.6.17 TDBNavigator.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

47.1.11.6.18 TDBNavigator.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

47.1.11.6.19 TDBNavigator.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
```

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

47.1.11.6.20 TDBNavigator.VisibleButtons Property

Determines which buttons appear on the database navigator.

Pascal

```
property VisibleButtons: TNavButtonSet;
```

C++

```
__property TNavButtonSet VisibleButtons;
```

Description

Use VisibleButtons to select the buttons that will appear on the navigator. Leave any of the navigator buttons out of the VisibleButtons set to hide those buttons and thereby prevent the user from performing certain operations. For example, to only allow the user to view the records in the dataset, VisibleButtons should include only the nbFirst, nbPrior, nbNext, and nbLast

See Also

Hints (see page 9258)

Visible (see page 5847)

Choosing Navigator Buttons to Display

47.1.11.7 TDBNavigator Events

The events of the TDBNavigator class are listed here.

Published Events

	Name	Description
	BeforeAction (see page 9269)	Occurs when a button on the database navigator is clicked, before the action is executed.
	OnClick (see page 9269)	Occurs when a button on the database navigator is clicked, after the action is executed.
	OnContextPopup (see page 9270)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 9271)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 9271)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 9273)	Occurs when the user drags an object over a control.
	OnEndDock (see page 9276)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 9276)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 9277)	Occurs when a control receives the input focus.
	OnExit (see page 9279)	Occurs when the input focus shifts away from one control to another.
	OnResize (see page 9281)	Occurs immediately after the control is resized.
	OnStartDock (see page 9282)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 9282)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

47.1.11.7.1 TDBNavigator.BeforeAction Event

Occurs when a button on the database navigator is clicked, before the action is executed.

Pascal

```
property BeforeAction: ENavClick;
```

C++

```
__property ENavClick BeforeAction;
```

Description

Write a BeforeAction event handler to take specific action before the default response when a user clicks a button in the database navigator.

See Also

OnClick (see page 9269)

VisibleButtons (see page 9268)

47.1.11.7.2 TDBNavigator.OnClick Event

Occurs when a button on the database navigator is clicked, after the action is executed.

Pascal

```
property OnClick: ENavClick;
```

C++

```
__property ENavClick OnClick;
```

Description

Use the OnClick event handler to write code that responds to the click event. Because the buttons on the navigator already have default actions that occur when they are clicked, it isn't necessary to write an OnClick event handler.

The OnClick event occurs when

The user clicks one the buttons on the navigator with the mouse

The user presses Spacebar while the navigator has focus.

The BtnClick (see page 9245) method is called.

Note: The OnClick event occurs after the default action for the clicked button.

See Also

BeforeAction (see page 9269)

VisibleButtons (see page 9268)

Using the First and Last Methods

47.1.11.7.3 TDBNavigator.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

47.1.11.7.4 TDBNavigator.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

47.1.11.7.5 TDBNavigator.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

47.1.11.7.6 TDBNavigator.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
```

```
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code comes from an application that contains a list box
  and three labels, each with a different font and color.  The
  DragMode property for each of the labels is dmAutomatic.
  The user can select a label and drag it to a list box and
  drop it. When the label is dropped, the items in the list
  box assume the color and font of the dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.11.7.7 TDBNavigator.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.11.7.8 TDBNavigator.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
```

```
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

47.1.11.7.9 TDBNavigator.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
```

```
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

47.1.11.7.10 TDBNavigator.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

47.1.11.7.11 TDBNavigator.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

47.1.11.7.12 TDBNavigator.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

47.1.11.7.13 TDBNavigator.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method

for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

47.1.12 TDBRadioGroup Class

TDBRadioGroup represents a group of radio buttons connected to a database.

Class Hierarchy

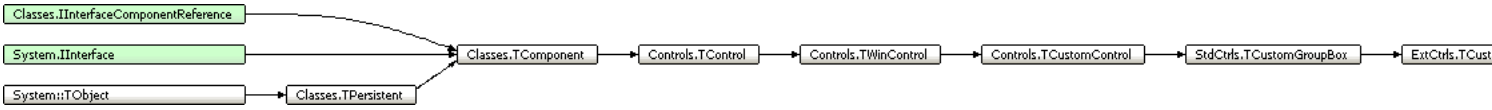

Pascal

```
TDBRadioGroup = class(TCustomRadioGroup);
```

C++

```
class TDBRadioGroup : public TCustomRadioGroup;
```

File

DBCtrls

Description

Use TDBRadioGroup to group a set of data-aware radio buttons. Radio buttons that are contained directly in the same control component are said to be "grouped"; when a radio button is checked, all other radio buttons in its group become unchecked. Database radio groups provide an easy way to ensure that the user enters one (or at most one) of the options presented for a field. They are also useful for displaying data from fields that have only a few possible values.

To add radio buttons to a TDBRadioGroup, edit the Items (see page 9315) property in the Object Inspector. Each string in Items (see page 9315) makes a radio button appear in the group box with the string as its caption.

When the user selects a radio button, the "value" of the selected button becomes the contents of the linked field for the current record in the dataset. Specify values for the buttons by editing the Values (see page 9326) property in the Object Inspector; the first string in Values (see page 9326) corresponds to the first string in Items (see page 9315), and so forth. If no Values (see page 9326) strings are set, each button's caption serves as its value.

See Also

TDataSource (see page 7070)

TRadioButton (see page 21753)

Restricting Field Values with Radio Controls

Using Common Data Control Features

47.1.12.1 TDBRadioGroup Members

The following tables list the members exposed by TDBRadioGroup.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 21287)	Adjusts the ClientRect property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanModify (see page 9295)	Determines whether the user can select a different radio button in the group.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 9296)	This is Change, a member of class TDBRadioGroup.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
V	Click (see page 9296)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 21287)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.

	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetChildren (see page 12560)	Prevents the streaming system from loading and saving child controls in the usual fashion.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.

	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 9297)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	Loaded (see page 12560)	Initializes the radio group after it is loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 9297)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 5920)	Renders the image of a custom control.
	Paint (see page 21287)	Renders the onscreen image of the group box.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	ReadState (see page 12560)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.

	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.

	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
R	DataLink (see page 9309)	This is DataLink, a member of class TDBRadioGroup.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 9294)	Creates and initializes a TDBRadioGroup (see page 9283) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 9295)	Destroys the TDBRadioGroup (see page 9283) instance and the radio buttons it contains.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
V	ExecuteAction (see page 9296)	Invokes an action with the TDBRadioGroup (see page 9283) as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
V	FlipChildren (see page 12559)	Prevents the default behavior of flipping the child radio buttons.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.

	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.

V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
V	UpdateAction (see page 9298)	Updates an action to reflect the current state of the TDBRadioGroup (see page 9283).
	UpdateControlState (see page 6186)	Respond to state change.
V	UseRightToLeftAlignment (see page 9298)	Specifies whether the control's alignment is in a right-to-left mode.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Buttons (see page 12561)	Access individual TRadioButton objects.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Field (see page 9313)	Identifies the TField component to which the radio group is linked.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.

R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemIndex (see page 9314)	Indicates which radio button in the group is currently selected.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	Value (see page 9325)	Holds the content of the linked field for the current record in the database.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 9328)	Occurs when the user changes the data in the linked field.
	OnClick (see page 9328)	Occurs when the user clicks the control.
	OnContextPopup (see page 9330)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 9331)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 9333)	Occurs when the user drags an object over a control.
	OnEndDock (see page 9335)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 9336)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 9337)	Occurs when a control receives the input focus.
	OnExit (see page 9339)	Occurs when the input focus shifts away from one control to another.
	OnMouseEnter (see page 9341)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 9341)	Occurs when the user moves the mouse outside of a control.
	OnStartDock (see page 9341)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 9342)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 9300)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 9303)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 9304)	Specifies the bi-directional mode for the control.
	Caption (see page 9304)	Specifies a text string that identifies the control to the user.
	Color (see page 9307)	Specifies the background color of the control.
	Columns (see page 9308)	Specifies number of columns in the radio group.
	Constraints (see page 9308)	Specifies the size constraints for the control.
	Ctl3D (see page 9308)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DataField (see page 9309)	Specifies the field from which the radio group displays data.
	DataSource (see page 9309)	Determines where the database radio group obtains the data to display.
	DragCursor (see page 9310)	Indicates the image used to represent the mouse pointer when the control is being dragged.

	DragKind (see page 9310)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 9311)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 9312)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 9313)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
W	Items (see page 9315)	Lists the radio buttons in the radio group.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBackground (see page 9315)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 9316)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 9316)	Determines where a control looks for its color information.
	ParentCtl3D (see page 9317)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 9317)	Determines where a control looks for its font information.
	ParentShowHint (see page 9317)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 9318)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 9323)	Determines whether the user can select a different radio button in the group, and hence modify the data in the field.
	ShowHint (see page 9323)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 9324)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 9325)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Values (see page 9326)	Determines the values of the radio buttons.
	Visible (see page 9326)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

47.1.12.2 TDBRadioGroup.Create Constructor

Creates and initializes a TDBRadioGroup (see page 9283) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDBRadioGroup(TComponent * AOwner);
```

Description

Call Create to instantiate a TDBRadioGroup (see page 9283) object at runtime. For radio groups placed on forms at design time, Create is called automatically.

Pass a single Component as a parameter to provide the TDBRadioGroup (see page 9283) object with an Owner that is responsible for freeing it.

Create calls the inherited constructor method, then initializes internal objects and variables used to control data access.

See Also

Destroy (see page 9295)

47.1.12.3 TDBRadioGroup.Destroy Destructor

Destroys the TDBRadioGroup (see page 9283) instance and the radio buttons it contains.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBRadioGroup();
```

Description

Destroy frees internally created objects that control data access, then calls the inherited destructor method.

See Also

Create (see page 9294)

47.1.12.4 TDBRadioGroup Methods

The methods of the TDBRadioGroup class are listed here.

Protected Methods

	Name	Description
	CanModify (see page 9295)	Determines whether the user can select a different radio button in the group.
	Change (see page 9296)	This is Change, a member of class TDBRadioGroup.
	Click (see page 9296)	Respond to user click.
	KeyPress (see page 9297)	Respond to keyboard input.
	Notification (see page 9297)	Responds to notifications that components are being created or destroyed.

Public Methods

	Name	Description
	ExecuteAction (see page 9296)	Invokes an action with the TDBRadioGroup (see page 9283) as its target.
	UpdateAction (see page 9298)	Updates an action to reflect the current state of the TDBRadioGroup (see page 9283).
	UseRightToLeftAlignment (see page 9298)	Specifies whether the control's alignment is in a right-to-left mode.

47.1.12.4.1 TDBRadioGroup.CanModify Method

Determines whether the user can select a different radio button in the group.

Pascal

```
function CanModify: Boolean; override;
```

C++

```
virtual __fastcall Boolean CanModify();
```

Description

Event handlers call CanModify to verify that the user is allowed to select a different radio button in the group. As defined in TCustomRadioGroup, this method always returns true; override it to return false in situations where the user should not be able to make changes.

See Also

CanModify

47.1.12.4.2 TDBRadioGroup.Change Method

Pascal

```
procedure Change; dynamic;
```

C++

```
__fastcall Change();
```

Description

This is Change, a member of class TDBRadioGroup.

47.1.12.4.3 TDBRadioGroup.Click Method

Respond to user click.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Click is called automatically when the user left-clicks the control. Component or application code can call Click to simulate a user mouse click. This is often done in menu actions and hotkey handlers.

As implemented in TControl, Click queries whether the OnClick event handler exists and is different from the OnExecute handler for the control's Action. If this is true, the OnClick event handler is called. If this is false, and the Action OnExecute event handler exists, the Action's Execute method is called. Override Click to provide additional behavior.

See Also

Action (see page 5797)

OnExecute (see page 2716)

Execute (see page 2712)

DblClick (see page 5747)

OnClick (see page 5851)

47.1.12.4.4 TDBRadioGroup.ExecuteAction Method

Invokes an action with the TDBRadioGroup (see page 9283) as its target.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

ExecuteAction is called automatically when the user invokes an action with the TDBRadioGroup (see page 9283) as a target. The Action parameter specifies the action that was invoked. ExecuteAction returns trueif the action was successfully dispatched, and false if the TDBRadioGroup (see page 9283) could not handle the action.

See Also

UpdateAction (see page 9298)

Execute (see page 2712)

HandlesTarget (see page 2713)

47.1.12.4.5 TDBRadioGroup.KeyPress Method

Respond to keyboard input.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

When a windowed control receives a key-press message (WM_CHAR) from Windows, its message handler calls the DoKeyPress method. If DoKeyPress determines that the control should, in fact, process the character, it calls KeyPress, passing the key code in the Key parameter.

KeyPress calls any event handler attached to the OnKeyPress event. Override KeyPress to provide other responses in addition to the event-handler call.

Either KeyPress or the OnKeyPress event handler it calls can suppress further processing of a character by setting the Key parameter to zero.

Note: The Key parameter is the character represented by the key that is pressed, not a Windows virtual key code.

See Also

KeyDown (see page 6163)

KeyUp (see page 6164)

OnKeyPress (see page 6227)

47.1.12.4.6 TDBRadioGroup.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been

created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

47.1.12.4.7 TDBRadioGroup.UpdateAction Method

Updates an action to reflect the current state of the TDBRadioGroup (see page 9283).

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

UpdateAction is called automatically when the application is idle to give the TDBRadioGroup (see page 9283) an opportunity to update associated actions. This allows actions to be grayed, checked, or to otherwise reflect the status of the TDBRadioGroup (see page 9283) they target.

The Action parameter specifies the action component that should be updated. UpdateAction returns true if the action component now reflects the state of the component, and false if it did not know how to update the action.

See Also

ExecuteAction (see page 9296)

UpdateTarget (see page 2715)

HandlesTarget (see page 2713)

47.1.12.4.8 TDBRadioGroup.UseRightToLeftAlignment Method

Specifies whether the control's alignment is in a right-to-left mode.

Pascal

```
function UseRightToLeftAlignment: Boolean; override;
```

C++

```
virtual __fastcall Boolean UseRightToLeftAlignment();
```

Description

Call UseRightToLeftAlignment to determine whether the TDBRadioGroup (see page 9283)'s alignment is in a right-to-left mode. This information can be used when painting the TDBRadioGroup (see page 9283) data or when responding to mouse events.

UseRightToLeftAlignment returns true for Middle East locales if the BiDiMode (see page 9304) property is bdRightToLeft. Otherwise, it returns false.

See Also

UseRightToLeftReading (see page 5790)

BiDiMode (see page 5803)

47.1.12.5 TDBRadioGroup Properties

The properties of the TDBRadioGroup class are listed here.

Protected Properties

	Name	Description
R	DataLink (see page 9309)	This is DataLink, a member of class TDBRadioGroup.

Public Properties

	Name	Description
R	Field (see page 9313)	Identifies the TField component to which the radio group is linked.
	ItemIndex (see page 9314)	Indicates which radio button in the group is currently selected.
	Value (see page 9325)	Holds the content of the linked field for the current record in the database.

Published Properties

	Name	Description
	Align (see page 9300)	Determines how the control aligns within its container (parent control).
	Anchors (see page 9303)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 9304)	Specifies the bi-directional mode for the control.
	Caption (see page 9304)	Specifies a text string that identifies the control to the user.
	Color (see page 9307)	Specifies the background color of the control.
	Columns (see page 9308)	Specifies number of columns in the radio group.
	Constraints (see page 9308)	Specifies the size constraints for the control.
	Ctl3D (see page 9308)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DataField (see page 9309)	Specifies the field from which the radio group displays data.
	DataSource (see page 9309)	Determines where the database radio group obtains the data to display.
	DragCursor (see page 9310)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 9310)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 9311)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 9312)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 9313)	Controls the attributes of text written on or in the control.
W	Items (see page 9315)	Lists the radio buttons in the radio group.
	ParentBackground (see page 9315)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 9316)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 9316)	Determines where a control looks for its color information.
	ParentCtl3D (see page 9317)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 9317)	Determines where a control looks for its font information.
	ParentShowHint (see page 9317)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 9318)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 9323)	Determines whether the user can select a different radio button in the group, and hence modify the data in the field.
	ShowHint (see page 9323)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 9324)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 9325)	Determines if the user can tab to a control.
	Values (see page 9326)	Determines the values of the radio buttons.

	Visible (see page 9326)	Determines whether the component appears onscreen.

47.1.12.5.1 TDBRadioGroup.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;
```

```
  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
```

```
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
```

```
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

47.1.12.5.2 TDBRadioGroup.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other

aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

47.1.12.5.3 TDBRadioGroup.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

47.1.12.5.4 TDBRadioGroup.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

```
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
```

```
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

47.1.12.5.5 TDBRadioGroup.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

47.1.12.5.6 TDBRadioGroup.Columns Property

Specifies number of columns in the radio group.

Pascal

```
property Columns: Integer;
```

C++

```
__property int Columns;
```

Description

Columns determines the number of columns in the radio group if Orientation is orVertical (the default), and the number of rows if Orientation is orHorizontal. Its value can range from 1 to 16. The default value is 1, which means that the radio buttons are arranged in a single line.

47.1.12.5.7 TDBRadioGroup.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

47.1.12.5.8 TDBRadioGroup.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

47.1.12.5.9 TDBRadioGroup.DataField Property

Specifies the field from which the radio group displays data.

Pascal

```
property DataField: WideString;
```

C++

```
__property BSTR DataField;
```

Description

Use DataField to specify the name of the database field that the radio group represents.

```
DBRadioGroup1.DataField := 'PaymentMethod';
DBRadioGroup1->DataField = "PaymentMethod";
```

See Also

DataSource (see page 9309)

47.1.12.5.10 TDBRadioGroup.DataLink Property

Pascal

```
property DataLink: TFieldDataLink;
```

C++

```
__property TFieldDataLink DataLink;
```

Description

This is DataLink, a member of class TDBRadioGroup.

47.1.12.5.11 TDBRadioGroup.DataSource Property

Determines where the database radio group obtains the data to display.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource holds a TDataSource object that points to a dataset component. The dataset component should be linked to the database you want to access.

```
DBRadioGroup1.DataSource := DataSource1;
DBRadioGroup1->DataSource = DataSource1;
```

See Also

DataField (see page 9309)

TDataSet (see page 6894)

47.1.12.5.12 TDBRadioGroup.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

47.1.12.5.13 TDBRadioGroup.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

47.1.12.5.14 TDBRadioGroup.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
```

```
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

47.1.12.5.15 TDBRadioGroup.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

47.1.12.5.16 TDBRadioGroup.Field Property

Identifies the TField component to which the radio group is linked.

Pascal

```
property Field: TField;
```

C++

```
__property TField Field;
```

Description

Field holds the TField object that the linked dataset component creates to access the database.

```
DBRadioGroup1.Field.AsString := 'Yes';
DBRadioGroup1->Field->AsString = "Yes";
```

See Also

TDataSet (see page 6894)

DataField (see page 9309)

47.1.12.5.17 TDBRadioGroup.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

47.1.12.5.18 TDBRadioGroup.ItemIndex Property

Indicates which radio button in the group is currently selected.

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

Description

ItemIndex holds the index of the selected radio button in the Items list. (The first button is 0.) The value of ItemIndex changes at runtime as the user selects radio buttons. If you want one of the buttons to appear selected when the application starts, assign that button to ItemIndex at design time; otherwise, leave ItemIndex set to the default value of -1, which means that no button is selected.

C++ Examples:

```
/*
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Items->Add("Vertical");
  RadioGroup1->Items->Add("Horizontal");
  RadioGroup1->ItemIndex = 2;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Vertical")
```

```
    ScrollBar1->Kind = sbVertical;
  else if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Horizontal")
    ScrollBar1->Kind = sbHorizontal;
}
```

Delphi Examples:

```
{
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  RadioGroup1.Items.Add('Vertical');
  RadioGroup1.Items.Add('Horizontal');
  RadioGroup1.ItemIndex := 2;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Vertical' then
    ScrollBar1.Kind := sbVertical;
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Horizontal' then
    ScrollBar1.Kind := sbHorizontal;
end;
```

See Also

Items (see page 12563)

47.1.12.5.19 TDBRadioGroup.Items Property

Lists the radio buttons in the radio group.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Items holds a TStrings object that lists the captions of the radio buttons in the group. (To add or remove radio buttons, edit the Items list from the Object Inspector. These captions become the "values" of the radio buttons, unless you override them in the Values (see page 9326) property; when a button is selected, its value becomes the content of the linked field for the current record in the database.

See Also

Items (see page 12563)

ItemIndex (see page 12562)

Values (see page 9326)

Restricting Field Values with Radio Controls

47.1.12.5.20 TDBRadioGroup.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground: Boolean;
```

C++

```
__property Boolean ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

47.1.12.5.21 TDBRadioGroup.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

47.1.12.5.22 TDBRadioGroup.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

47.1.12.5.23 TDBRadioGroup.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

47.1.12.5.24 TDBRadioGroup.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

47.1.12.5.25 TDBRadioGroup.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

47.1.12.5.26 TDBRadioGroup.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
```

```
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
```

```
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
```

```
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
```

```
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

47.1.12.5.27 TDBRadioGroup.ReadOnly Property

Determines whether the user can select a different radio button in the group, and hence modify the data in the field.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

If ReadOnly is true, the radio group displays data from the current record, but cannot be used to change the contents of the field. The default is false.

47.1.12.5.28 TDBRadioGroup.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
```

```
      strcpy(string, "Item");
      itoa(i, index, 10);
      strcat(string, index);
      ListBox1->Items->Add(string);
    }
    ListBox1->Hint = "Select an item";
    ListBox1->ShowHint = true;
    Application->ShowHint = true;
  }
  /*
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  */
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

47.1.12.5.29 TDBRadioGroup.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

47.1.12.5.30 TDBRadioGroup.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

47.1.12.5.31 TDBRadioGroup.Value Property

Holds the content of the linked field for the current record in the database.

Pascal

```
property Value: string;
```

C++

```
__property AnsiString Value;
```

Description

Value holds the content of the linked field for the current record. If the user selects a different radio button, the Value property changes.

47.1.12.5.32 TDBRadioGroup.Values Property

Determines the values of the radio buttons.

Pascal

```
property Values: TStrings;
```

C++

```
__property TStrings Values;
```

Description

When the user selects a radio button, the "value" of that button is written to the linked field in the database. By default, the value of a button is simply the caption that appears next to it on the screen, as determined by the Items (see page 9315) property.

In some cases, you may want the values of the radio buttons to differ from their captions. For example, if you use radio buttons to represent a database field whose content can be "Y" or "N", you may want the buttons' captions to be "Yes" and "No". In this case, you would enter "Yes" and "No" in the Items (see page 9315) list, and enter "Y" and "N" in the Values list.

You can edit the Values list in the Object Inspector, just as you would the Items (see page 9315) list. To keep the buttons' captions as their values, leave the Values property empty.

See Also

Items (see page 9315)

Restricting Field Values with Radio Controls

47.1.12.5.33 TDBRadioGroup.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
```

```
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

47.1.12.6 TDBRadioGroup Events

The events of the TDBRadioGroup class are listed here.

Published Events

	Name	Description
	OnChange (see page 9328)	Occurs when the user changes the data in the linked field.
	OnClick (see page 9328)	Occurs when the user clicks the control.
	OnContextPopup (see page 9330)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 9331)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 9333)	Occurs when the user drags an object over a control.
	OnEndDock (see page 9335)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 9336)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 9337)	Occurs when a control receives the input focus.
	OnExit (see page 9339)	Occurs when the input focus shifts away from one control to another.
	OnMouseEnter (see page 9341)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 9341)	Occurs when the user moves the mouse outside of a control.
	OnStartDock (see page 9341)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 9342)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

47.1.12.6.1 TDBRadioGroup.OnChange Event

Occurs when the user changes the data in the linked field.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

The OnChange event occurs when a change is made to the contents of the database field accessed by the radio group. Use an OnChange event handler to implement any special processing that should be executed when the radio group is updated.

47.1.12.6.2 TDBRadioGroup.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
```

```
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

47.1.12.6.3 TDBRadioGroup.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

47.1.12.6.4 TDBRadioGroup.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
```

```
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.
```

```
procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

47.1.12.6.5 TDBRadioGroup.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
```

```
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
```

```
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.12.6.6 TDBRadioGroup.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.12.6.7 TDBRadioGroup.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
```

```
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

47.1.12.6.8 TDBRadioGroup.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
```

```
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;
```

```
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

47.1.12.6.9 TDBRadioGroup.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
```

```
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

47.1.12.6.10 TDBRadioGroup.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

47.1.12.6.11 TDBRadioGroup.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

47.1.12.6.12 TDBRadioGroup.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

47.1.12.6.13 TDBRadioGroup.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

47.1.13 TDBRichEdit Class

TDBRichEdit represents a multiline edit control that can display and edit a rich text memo field in a dataset.

Class Hierarchy

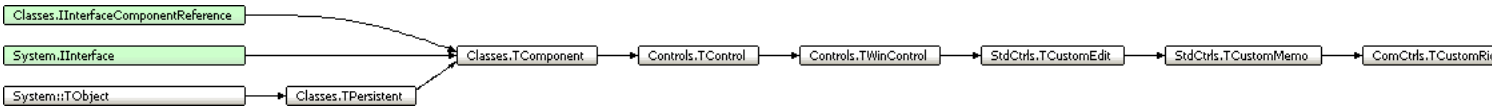

Pascal

```
TDBRichEdit = class(TCustomRichEdit);
```

C++

```
class TDBRichEdit : public TCustomRichEdit;
```

File

DBCtrls

Description

Use a TDBRichEdit object to enable users to edit a database field in a rich text edit control. TDBRichEdit provides the properties and methods to enter and work with rich text. TDBRichEdit does not provide any user interface components to make these formatting options available to the user. Applications must implement the user interface components to surface the rich text capabilities of a TDBRichEdit object. make this easier, a number of pre-defined standard actions operate on rich edit controls to perform common formatting tasks.

TDBRichEdit uses the Text property to represent the contents of the field. Like other data-aware controls, TDBRichEdit controls automatically put the data source into edit mode when the value of the Text property is changed. However, if only the formatting of the text is changed, but not the text itself, the application must explicitly put the data source into edit mode so that the changes will be saved. This should be implemented as part of the user interface for the rich text capabilities.

If the database field does not include text formatting information, consider using a TDBMemo (see page 9149) object or a TDBEdit (see page 8735) object instead.

If the application doesn't require the data-aware capabilities of TDBRichEdit, use a rich text edit control (TRichEdit) instead, to conserve system resources.

See Also

TDataSet (see page 6894)

TDataSource (see page 7070)

TDBEdit (see page 8735)

TDBMemo (see page 9149)

TField (see page 7115)

TRichEdit (see page 4685)

TRichEditAction (see page 11992)

Displaying and Editing Text in a Rich Edit Memo Control

Using Common Data Control Features

47.1.13.1 TDBRichEdit Members

The following tables list the members exposed by TDBRichEdit.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.

	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 9357)	Keeps track of the fact that the text has been modified, before letting the inherited method call the OnChange (see page 9401) event handler if it is assigned.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 3868)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 21248)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateParams (see page 21375)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWindowHandle (see page 21248)	Creates a Windows control to represent the control.
	CreateWindowHandle (see page 21376)	Creates a window based on the window-creation parameter record.
	CreateWnd (see page 3868)	Creates a Windows control corresponding to the edit control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 21248)	Creates a Windows control corresponding to the edit control.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.

	DestroyWnd (see page 3868)	Destroys the rich edit control's window.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21251)	Destroys the Windows control that corresponds to the edit control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoSetMaxLength (see page 3869)	Sets the MaxLength property.
	DoSetMaxLength (see page 21251)	Sets the MaxLength (see page 21263) property.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetCaretPos (see page 3871)	Returns the cursor position.
	GetCaretPos (see page 21376)	Returns the value of the CaretPos (see page 21380) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.

	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelLength (see page 3871)	Returns the SelLength property.
	GetSelLength (see page 21252)	Returns the SelLength (see page 21265) property.
	GetSelStart (see page 3872)	Returns the SelStart property.
	GetSelStart (see page 21252)	Returns the SelStart (see page 21269) property.
	GetSelText (see page 3872)	Returns the SelText property.
	GetSelText (see page 21252)	Returns the SelText (see page 21272) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 9358)	Supplements the inherited method to drive the dataset into edit mode when the user presses Del or Shift+Insert.
	KeyPress (see page 9359)	Supplements the inherited method to handle any keystrokes that alter the state of the edit control.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 9359)	Initializes the memo control when it is first loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 9360)	Overrides the inherited method to respond to opRemove notifications on the DataSource (see page 9373).
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.

	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
V	RequestSize (see page 3874)	Resize request event dispatcher.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectionChange (see page 3875)	Selection change event dispatcher.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAlignment (see page 21377)	Sets the Alignment (see page 21379) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetAutoSize (see page 21257)	Sets the AutoSize (see page 21260) property.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
V	SetCaretPos (see page 3875)	Sets the cursor position in the text.
V	SetCaretPos (see page 21378)	Sets the CaretPos (see page 21380) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetLines (see page 21378)	Sets the Lines (see page 21380) property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetScrollBars (see page 21378)	Sets the ScrollBars (see page 21381) property.
V	SetSelLength (see page 3876)	Sets the SelLength property.
V	SetSelLength (see page 21257)	Sets the SelLength (see page 21265) property.
V	SetSelStart (see page 3876)	Returns the SelStart property.
V	SetSelStart (see page 21257)	Sets the SelStart (see page 21269) property.
	SetWordWrap (see page 21379)	Sets the WordWrap (see page 21384) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.

	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSelect (see page 21260)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AutoSize (see page 21260)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	CharCase (see page 21262)	Determines the case of the text within the edit control.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	Lines (see page 3882)	Contains the individual lines of text in the rich text edit control.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	OEMConvert (see page 21264)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	PasswordChar (see page 21265)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 3867)	Deletes all text from the rich edit control.
V	Clear (see page 21245)	Deletes all text from the edit control.
	ClearSelection (see page 21245)	Removes the selected text from the edit control.
	ClearUndo (see page 21246)	Clears the undo buffer so that no changes to the text can be backed out.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopyToClipboard (see page 21246)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
V	Create (see page 9356)	Creates an instance of TDBRichEdit (see page 9343).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CutToClipboard (see page 21249)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 21250)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 9356)	Destroys an instance of TDBRichEdit (see page 9343).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.

	EndDrag (see page 5757)	Stops a control from being dragged any further.
V	ExecuteAction (see page 9358)	Invokes an action with the TDBRichEdit (see page 9343) as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindText (see page 3869)	Searches a given range in the text for a target string.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
V	GetControlsAlignment (see page 21376)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
V	GetSelTextBuf (see page 3872)	Copies the selected text into a buffer and returns the number of characters copied.
V	GetSelTextBuf (see page 21253)	Copies the selected text into a buffer and returns the number of characters copied.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
V	LoadMemo (see page 9359)	LoadMemo loads the memo BLOB from the field in the database into the rich edit control.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	PasteFromClipboard (see page 21254)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	Print (see page 3873)	Formats and prints the contents of the rich edit control.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RegisterConversionFormat (see page 3874)	Registers an association between a file extension and a TConversion (see page 3582) object that can convert between the native file format and rich text format (RTF).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectAll (see page 21255)	Selects all text in the edit control.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetDesignVisible (see page 6178)	Makes a control visible at design time.
	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSelTextBuf (see page 21258)	Replaces the selected text with a null-terminated string.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Undo (see page 21259)	Backs out all changes in the undo buffer.
	Update (see page 5789)	Processes any pending paint messages immediately.
	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 9360)	Updates an action to reflect the current state of the TDBRichEdit (see page 9343).
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 9361)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.

R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	CanUndo (see page 21262)	Indicates whether the edit control contains changes that can be backed out.
	CaretPos (see page 21380)	Indicates the position of the caret in the client area of the memo.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DefAttributes (see page 3877)	Describes the rich text characteristics of the default font for the rich edit control
	DefaultConverter (see page 3878)	Specifies the class of the TConversion (see page 3582) object that is used to convert between native file format and the internal format of a rich text control when working with files that do not have a registered extension.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Field (see page 9376)	Indicates the TField object for the database field the rich edit control represents.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	Lines (see page 21380)	Contains the individual lines of text in the memo control.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Modified (see page 21263)	Indicates whether the user edited the text of the edit control.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	PageRect (see page 3884)	Specifies the dimensions, in pixels, of the logical page size used when printing the contents of a rich text edit control.
R	Paragraph (see page 3884)	Specifies the formatting information for the current paragraphs.

	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelAttributes (see page 3887)	Describes the rich text characteristics of the selected text in the rich edit control.
	SelLength (see page 21265)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 21269)	Specifies the position of the first selected character in the text.
	SelText (see page 21272)	Specifies the selected portion of the edit control's text.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	Text (see page 21273)	Contains a text string associated with the control.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 9401)	Occurs when the text for the edit control may have changed.
	OnClick (see page 9401)	Occurs when the user clicks the control.
	OnContextPopup (see page 9403)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 9404)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 9404)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 9406)	Occurs when the user drags an object over a control.
	OnEndDock (see page 9409)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 9409)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 9410)	Occurs when a control receives the input focus.
	OnExit (see page 9413)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 9414)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 9417)	Occurs when a key is pressed.
	OnKeyUp (see page 9418)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 9420)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 9421)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 9423)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 9423)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 9424)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 9426)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnProtectChange (see page 9428)	Occurs when a user attempts to modify text that is marked as protected.
	OnResizeRequest (see page 9429)	Occurs when the text has grown either smaller or larger than the control's window size.
	OnSaveClipboard (see page 9430)	Occurs when the window of the edit control is about to be destroyed and the edit control has delayed rendering of text copied to the clipboard..
	OnSelectionChange (see page 9430)	Occurs when the current selection changes.
	OnStartDock (see page 9431)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 9432)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 9362)	Determines how the control aligns within its container (parent control).
	Alignment (see page 9366)	Determines how the text is aligned within the memo control.

	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 9366)	Specifies how the control is anchored to its parent.
	AutoDisplay (see page 9367)	Determines whether to automatically display the contents of the memo field in the rich edit control.
	BevelEdges (see page 9368)	Specifies which edges of the control are beveled.
	BevelInner (see page 9368)	Specifies the cut of the inner bevel.
	BevelKind (see page 9368)	Specifies the control's bevel style.
	BevelOuter (see page 9369)	Specifies the cut of the outer bevel.
	BevelWidth (see page 9369)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 9370)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 9370)	Determines whether the edit control has a single line border around the client area.
	Color (see page 9370)	Specifies the background color of the control.
	Constraints (see page 9371)	Specifies the size constraints for the control.
	Ctl3D (see page 9372)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DataField (see page 9372)	Specifies the field from which the rich edit control displays data.
	DataSource (see page 9373)	Links the rich edit control to the dataset that contains the field it represents.
	DragCursor (see page 9373)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 9374)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 9374)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 9375)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 9376)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	HideScrollBars (see page 9377)	Determines whether the scroll bars disappear when not needed.
	HideSelection (see page 9379)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 9381)	Determines the behavior of the input method editor (IME).
	ImeName (see page 9381)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	MaxLength (see page 9382)	Specifies the maximum number of characters the user can enter into the edit control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 9382)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 9383)	Determines where a control looks for its color information.
	ParentCtl3D (see page 9383)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 9384)	Determines where a control looks for its font information.
	ParentShowHint (see page 9384)	Determines where a control looks to find out if its Help Hint should be shown.
	PlainText (see page 9385)	Controls whether the rich edit control treats the text as plain text or rich text when streaming to or from a file.
	PopupMenu (see page 9386)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 9390)	Determines whether the user can use the rich database edit control to change the value of the field in the current record.
	ScrollBars (see page 9391)	Determines whether the memo control has scroll bars.
	ShowHint (see page 9393)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

	TabOrder (see page 9395)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 9395)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 9396)	Determines whether the component appears onscreen.
	WantReturns (see page 9397)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 9398)	Determines whether the user can insert tab characters into the text.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	WordWrap (see page 9398)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

47.1.13.2 TDBRichEdit.Create Constructor

Creates an instance of TDBRichEdit (see page 9343).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDBRichEdit(TComponent * AOwner);
```

Description

Call Create to instantiate a TDBRichEdit (see page 9343) object at runtime. For TDBRichEdit (see page 9343) objects placed on forms at design-time, the constructor is called automatically.

Pass a single Component as a parameter to provide the TDBRichEdit (see page 9343) object with an Owner that is responsible for freeing it.

After calling the inherited constructor to allocate and initialize a generic rich edit object, Create

Sets ReadOnly (see page 9390) to true.

Sets AutoDisplay (see page 9367) to true.

Creates and initializes the helper objects needed to link to a dataset.

See Also

Destroy (see page 9356)

AutoDisplay (see page 9367)

ReadOnly (see page 9390)

Create (see page 3866)

TFieldDataLink (see page 9487)

47.1.13.3 TDBRichEdit.Destroy Destructor

Destroys an instance of TDBRichEdit (see page 9343).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBRichEdit();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TDBRichEdit (see page 9343) reference is not nil and only then calls Destroy.

Destroy frees the helper objects of the TDBRichEdit (see page 9343) before calling the inherited destructor.

See Also

Create (see page 9356)

Free (see page 22300)

47.1.13.4 TDBRichEdit Methods

The methods of the TDBRichEdit class are listed here.

Protected Methods

	Name	Description
	Change (see page 9357)	Keeps track of the fact that the text has been modified, before letting the inherited method call the OnChange (see page 9401) event handler if it is assigned.
	KeyDown (see page 9358)	Supplements the inherited method to drive the dataset into edit mode when the user presses Del or Shift+Insert.
	KeyPress (see page 9359)	Supplements the inherited method to handle any keystrokes that alter the state of the edit control.
	Loaded (see page 9359)	Initializes the memo control when it is first loaded into memory.
	Notification (see page 9360)	Overrides the inherited method to respond to opRemove notifications on the DataSource (see page 9373).

Public Methods

	Name	Description
	ExecuteAction (see page 9358)	Invokes an action with the TDBRichEdit (see page 9343) as its target.
	LoadMemo (see page 9359)	LoadMemo loads the memo BLOB from the field in the database into the rich edit control.
	UpdateAction (see page 9360)	Updates an action to reflect the current state of the TDBRichEdit (see page 9343).
	UseRightToLeftAlignment (see page 9361)	Specifies whether the control's alignment should be reversed right-to-left.

47.1.13.4.1 TDBRichEdit.Change Method

Keeps track of the fact that the text has been modified, before letting the inherited method call the OnChange (see page 9401) event handler if it is assigned.

Pascal

```
procedure Change; override;
```

C++

```
virtual __fastcall Change();
```

Description

Applications cannot call this protected method directly. It is triggered automatically by Windows' events. Change responds to these events by internally keeping track of the fact that the text has been modified. This information can then be used to know whether there are any changes to post to the dataset or to remove if the user requests a reset.

Change also calls the inherited method, which calls the OnChange (see page 9401) event handler if it is assigned.

See Also

OnChange (see page 21275)

Text (see page 5842)

47.1.13.4.2 TDBRichEdit.ExecuteAction Method

Invokes an action with the TDBRichEdit (see page 9343) as its target.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

ExecuteAction is called automatically when the user invokes an action with the TDBRichEdit (see page 9343) as a target. The Action parameter specifies the action that was invoked. ExecuteAction returns trueif the action was successfully dispatched, and false if the TDBRichEdit (see page 9343) could not handle the action.

See Also

UpdateAction (see page 9360)

Execute (see page 2712)

HandlesTarget (see page 2713)

47.1.13.4.3 TDBRichEdit.KeyDown Method

Supplements the inherited method to drive the dataset into edit mode when the user presses Del or Shift+Insert.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

KeyDown first calls the inherited method, which calls the OnKeyDown (see page 9414) event handler. KeyDown next attempts to put the dataset into an editing state when the user types Del or Shift+Insert. Derived classes can override KeyDown to handle additional key combinations, but the inherited method should always be called.

Note: Applications cannot call this protected method directly. It is called automatically in response to Windows events.

See Also

DataSource (see page 9373)

KeyPress (see page 9359)

OnKeyDown (see page 6225)

Edit (see page 9491)

47.1.13.4.4 TDBRichEdit.KeyPress Method

Supplements the inherited method to handle any keystrokes that alter the state of the edit control.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress first calls the inherited method, which calls the OnKeyPress (see page 9417) event handler. Next, KeyPress checks the value of the key, and throws it away if it is not a valid character for the field type, or if there is no link to a database field. Valid characters cause KeyPress to attempt to put the dataset into edit mode.

Pressing the Escape key causes KeyPress to throw away all unposted edits. Pressing the Pause key causes KeyPress to call LoadMemo (see page 9359) to load the value of the memo field associated with the rich edit control.

Note: Applications cannot call this protected method directly. It is called automatically in response to Windows events.

See Also

DataSource (see page 9373)

KeyDown (see page 9358)

LoadMemo (see page 9359)

OnKeyPress (see page 6227)

Reset (see page 9493)

47.1.13.4.5 TDBRichEdit.Loaded Method

Initializes the memo control when it is first loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Loaded overrides the inherited method so that the Modified property is set to false when the form containing the memo is first loaded into memory. Derived classes can override the Loaded method to perform additional initializations after the memo's form is loaded.

See Also

Modified (see page 21263)

47.1.13.4.6 TDBRichEdit.LoadMemo Method

LoadMemo loads the memo BLOB from the field in the database into the rich edit control.

Pascal

```
procedure LoadMemo; virtual;
```

C++

```
virtual __fastcall LoadMemo();
```

Description

LoadMemo allows an application to control when the rich edit control displays the contents of the memo field. Use LoadMemo to update the text in the rich edit control to the value of the field in the current record. LoadMemo only works with BLOB fields when the AutoDisplay (see page 9367) property is false, so that the value of the field is not loaded automatically.

See Also

AutoDisplay (see page 9367)

Field (see page 9376)

Text (see page 5842)

47.1.13.4.7 TDBRichEdit.Notification Method

Overrides the inherited method to respond to opRemove notifications on the DataSource (see page 9373).

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

After calling the inherited method, Notification checks whether the notification indicates the removal of the DataSource (see page 9373). If so, Notification sets the DataSource (see page 9373) property to nil (Delphi) or NULL (C++).

Derived classes can override Notification method to handle additional notifications, but the inherited method should be always be called.

Note: Notification is a protected method. Applications cannot call it directly.

See Also

DataSource (see page 9373)

47.1.13.4.8 TDBRichEdit.UpdateAction Method

Updates an action to reflect the current state of the TDBRichEdit (see page 9343).

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

UpdateAction is called automatically when the application is idle to give the TDBRichEdit (see page 9343) an opportunity to update associated actions. This allows actions to be grayed, checked, or to otherwise reflect the status of the TDBRichEdit (see page 9343) they target.

The Action parameter specifies the action component that should be updated. UpdateAction returns trueif the action component now reflects the state of the component, and false if it did not know how to update the action.

See Also

ExecuteAction (see page 9358)

UpdateTarget (see page 2715)

HandlesTarget (see page 2713)

47.1.13.4.9 TDBRichEdit.UseRightToLeftAlignment Method

Specifies whether the control's alignment should be reversed right-to-left.

Pascal

```
function UseRightToLeftAlignment: Boolean; override;
```

C++

```
virtual __fastcall Boolean UseRightToLeftAlignment();
```

Description

Controls call UseRightToLeftAlignment when implementing the BiDiMode property. UseRightToLeftAlignment indicates whether the control's alignment should be reversed right-to-left. This information can be used when painting the control or when responding to mouse events.

UseRightToLeftAlignment returns true if the application is running on a system with a Middle Eastern locale and the BiDiMode property is bdRightToLeft. Otherwise, it returns false. Descendants override UseRightToLeftAlignment if alignment in Middle Eastern locales depends on other factors.

See Also

BiDiMode (see page 5803)

GetControlsAlignment (see page 5759)

Text (see page 5842)

Caption (see page 5806)

47.1.13.5 TDBRichEdit Properties

The properties of the TDBRichEdit class are listed here.

Public Properties

	Name	Description
R	Field (see page 9376)	Indicates the TField object for the database field the rich edit control represents.

Published Properties

	Name	Description
	Align (see page 9362)	Determines how the control aligns within its container (parent control).
	Alignment (see page 9366)	Determines how the text is aligned within the memo control.
	Anchors (see page 9366)	Specifies how the control is anchored to its parent.
	AutoDisplay (see page 9367)	Determines whether to automatically display the contents of the memo field in the rich edit control.
	BevelEdges (see page 9368)	Specifies which edges of the control are beveled.
	BevelInner (see page 9368)	Specifies the cut of the inner bevel.
	BevelKind (see page 9368)	Specifies the control's bevel style.
	BevelOuter (see page 9369)	Specifies the cut of the outer bevel.
	BevelWidth (see page 9369)	Specifies the width of the inner and outer bevels.

	BiDiMode (see page 9370)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 9370)	Determines whether the edit control has a single line border around the client area.
	Color (see page 9370)	Specifies the background color of the control.
	Constraints (see page 9371)	Specifies the size constraints for the control.
	Ctl3D (see page 9372)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DataField (see page 9372)	Specifies the field from which the rich edit control displays data.
	DataSource (see page 9373)	Links the rich edit control to the dataset that contains the field it represents.
	DragCursor (see page 9373)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 9374)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 9374)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 9375)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 9376)	Controls the attributes of text written on or in the control.
	HideScrollBars (see page 9377)	Determines whether the scroll bars disappear when not needed.
	HideSelection (see page 9379)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 9381)	Determines the behavior of the input method editor (IME).
	ImeName (see page 9381)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	MaxLength (see page 9382)	Specifies the maximum number of characters the user can enter into the edit control.
	ParentBiDiMode (see page 9382)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 9383)	Determines where a control looks for its color information.
	ParentCtl3D (see page 9383)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 9384)	Determines where a control looks for its font information.
	ParentShowHint (see page 9384)	Determines where a control looks to find out if its Help Hint should be shown.
	PlainText (see page 9385)	Controls whether the rich edit control treats the text as plain text or rich text when streaming to or from a file.
	PopupMenu (see page 9386)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 9390)	Determines whether the user can use the rich database edit control to change the value of the field in the current record.
	ScrollBars (see page 9391)	Determines whether the memo control has scroll bars.
	ShowHint (see page 9393)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 9395)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 9395)	Determines if the user can tab to a control.
	Visible (see page 9396)	Determines whether the component appears onscreen.
	WantReturns (see page 9397)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 9398)	Determines whether the user can insert tab characters into the text.
	WordWrap (see page 9398)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

47.1.13.5.1 TDBRichEdit.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it

continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
```

```
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
```

```
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
```

```
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

47.1.13.5.2 TDBRichEdit.Alignment Property

Determines how the text is aligned within the memo control.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to change the way the text is formatted by the memo control. Alignment can take one of the following values:

Value	Meaning
taLeftJustify	Align text to the left side of the control
taCenter	Center text horizontally in the control.
taRightJustify	Align text to the right side of the control.

See Also

SetAlignment (see page 21377)

Text (see page 5842)

Displaying and Editing Text in a Memo Control

47.1.13.5.3 TDBRichEdit.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized.

When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

47.1.13.5.4 TDBRichEdit.AutoDisplay Property

Determines whether to automatically display the contents of the memo field in the rich edit control.

Pascal

```
property AutoDisplay: Boolean;
```

C++

```
__property Boolean AutoDisplay;
```

Description

If AutoDisplay is true, the rich edit control automatically displays new data when the underlying BLOB field changes (such as when moving to a new record).

If AutoDisplay is false, the control shows only the field name whenever the underlying BLOB field changes. To display the data, the user can double-click on the control or select it and press Enter.

The effect of AutoDisplay is not purely cosmetic. The data shown by the TDBRichEdit (see page 9343) object is the value of the Text property. When AutoDisplay is false, if the data changes, the value of the Text property changes to the name of the field. Thus, if AutoDisplay is false, applications should be cautious about using the value of the Text property to ascertain the value of the underlying field.

Calling the LoadMemo (see page 9359) method causes Text to update to the current value of the BLOB field. This change will also be reflected in the appearance of the control on screen.

Change (see page 9357) the value of AutoDisplay to false if the automatic loading of BLOB fields seems to take too long.

Note: AutoDisplay only applies to BLOB fields. If the control is bound to another type of text field, AutoDisplay has no effect.

See Also

DataField (see page 9372)

Field (see page 9376)

KeyPress (see page 9359)

LoadMemo (see page 9359)

Text (see page 5842)

Displaying and Editing Text in a Rich Edit Memo Control

47.1.13.5.5 TDBRichEdit.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

47.1.13.5.6 TDBRichEdit.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

47.1.13.5.7 TDBRichEdit.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

47.1.13.5.8 TDBRichEdit.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

47.1.13.5.9 TDBRichEdit.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

47.1.13.5.10 TDBRichEdit.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

47.1.13.5.11 TDBRichEdit.BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the edit control stands out. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the edit control has a single-line border around the client area. If BorderStyle is bsNone, there will be no border.

47.1.13.5.12 TDBRichEdit.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

47.1.13.5.13 TDBRichEdit.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the

ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

47.1.13.5.14 TDBRichEdit.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

47.1.13.5.15 TDBRichEdit.DataField Property

Specifies the field from which the rich edit control displays data.

Pascal

```
property DataField: WideString;
```

C++

```
__property BSTR DataField;
```

Description

Use DataField to bind the rich edit control to a field in the dataset. To fully specify a database field, both the dataset and the field

within that dataset must be defined. The DataSource (see page 9373) property of the rich edit control specifies the dataset which contains the DataField.

```
DBRichEdit1.DataField := 'Document';
DBRichEdit1->DataField = "Document";
```

See Also

DataSource (see page 9373)

Field (see page 9376)

47.1.13.5.16 TDBRichEdit.DataSource Property

Links the rich edit control to the dataset that contains the field it represents.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Use DataSource to link the rich edit control to a dataset in which the data can be found. To fully specify a database field for the edit control, both the dataset and a field within that dataset must be defined. Use the DataField (see page 9372) property to specify the particular field within the dataset.

```
DBRichEdit1.DataSource := DataSource1;
DBRichEdit1->DataSource = DataSource1;
```

See Also

DataField (see page 9372)

Field (see page 9376)

47.1.13.5.17 TDBRichEdit.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

47.1.13.5.18 TDBRichEdit.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

47.1.13.5.19 TDBRichEdit.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
```

```
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

47.1.13.5.20 TDBRichEdit.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

47.1.13.5.21 TDBRichEdit.Field Property

Indicates the TField object for the database field the rich edit control represents.

Pascal

```
property Field: TField;
```

C++

```
__property TField Field;
```

Description

Read Field to get direct access to the contents and properties of the database field without going through the edit window. Use Field to change the contents of the database field programmatically. Use the Text property instead if you want to change the contents of the edit window without immediately posting the changes to the dataset. Changing the Text property allows the user to cancel the changes.

```
(DBRichEdit1.Field as TMemoField).Assign(RichEdit1.Lines);
```

See Also

DataField (see page 9372)

DataSource (see page 9373)

Text (see page 5842)

47.1.13.5.22 TDBRichEdit.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

47.1.13.5.23 TDBRichEdit.HideScrollBars Property

Determines whether the scroll bars disappear when not needed.

Pascal

```
property HideScrollBars: Boolean;
```

C++

```
__property Boolean HideScrollBars;
```

Description

Use HideScrollBars to determine whether the scroll bars disappear from the edit control when the text appears completely within the edit window. Set HideScrollBars to false to prevent the scroll bars from flashing on and off when the contents of the rich edit control change.

HideScrollBars does nothing when the ScrollBars property is set to ssNone (The default for TRichEdit).

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
```

```
  Rich Edit with a large amount of contiguous text.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    I: Integer;
  begin
    RichEdit1.Lines.Clear;
    for I := 0 to 500 do
    begin
      RichEdit1.Text :=
        RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
      if (i mod 4 = 0) then
        RichEdit1.Text := RichEdit1.Text + Chr(13);
    end;
    // move text cursor to top of editend;
    RichEdit1.SelStart := 0;
  end;

  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    RichEdit1.HideScrollbars := CheckBox1.Checked;
  end;

  procedure TForm1.RadioGroup1Click(Sender: TObject);
  const
    ScrollBarA: array[0..3] of TScrollStyle = (
      ssBoth,
      ssHorizontal,
      ssNone,
      ssVertical);
  begin
    RichEdit1.ScrollBars :=
      ScrollBarA[RadioGroup1.ItemIndex];
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with RadioGroup1, Items do
    begin
      Caption := 'Scroll Bars';
      Clear;
      Add('ssBoth');
      Add('ssHorizontal');
      Add('ssNone');
      Add('ssVertical');
    end;
    with CheckBox1 do
    begin
      Caption := 'HideScrollBars';
      Checked := True;
    end;
    Button1.Caption := 'Loaded';
    RichEdit1.Text := 'Just a little text.';
    RichEdit1.WordWrap := False;
  end;
```

See Also

ScrollBars (see page 21381)

47.1.13.5.24 TDBRichEdit.HideSelection Property

Determines whether the visual indication of the selected text remains when focus shifts to another control.

Pascal

```
property HideSelection: Boolean;
```

C++

```
__property Boolean HideSelection;__property Boolean HideSelection;
```

Description

Set HideSelection to false when it is important to provide visual feedback of the selected portion of the text even when the edit control does not have focus. Set HideSelection to true to make the visual indication of selection appear only when the edit control has focus. HideSelection does not affect the actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms containing many edit controls look busy.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
HideSelection, Height example
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
        HideSelection := False;
```

```
    end;
end;
```

See Also

SelLength

SelStart

SelText

SetSelTextBuf (see page 21258)

47.1.13.5.25 TDBRichEdit.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

47.1.13.5.26 TDBRichEdit.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted

characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

47.1.13.5.27 TDBRichEdit.MaxLength Property

Specifies the maximum number of characters the user can enter into the edit control.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the edit control. A value of 0 indicates that there is no application-defined limit on the length.

Use MaxLength to limit the length of the text in an edit control if that text will be copied into a fixed-length buffer.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

Note: The unit of MaxLength is characters (not bytes) if the application is using themes.

See Also

Text (see page 5842)

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

47.1.13.5.28 TDBRichEdit.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

47.1.13.5.29 TDBRichEdit.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

47.1.13.5.30 TDBRichEdit.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

47.1.13.5.31 TDBRichEdit.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

47.1.13.5.32 TDBRichEdit.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

47.1.13.5.33 TDBRichEdit.PlainText Property

Controls whether the rich edit control treats the text as plain text or rich text when streaming to or from a file.

Pascal

```
property PlainText: Boolean;
```

C++

```
__property Boolean PlainText;
```

Description

To write the rich text in the control to a plain text file, set PlainText to true before streaming the text to a file. To ignore the rich text information encoded in a file, set PlainText to true before streaming the text to the control. To stream in the rich text attributes encoded in a file, or save the encoding of the rich text attributes to a file, set PlainText to false.

If the rich text attributes of a file are encoded in some format other than rich text format (RTF), it is necessary to use a converter on the text, even when PlainText is true.

Note: Rich edit controls do not directly support streaming. Use the Lines property to stream to or from a file.

C++ Examples:

```
/*
This example requires two TRichEdit controls placed on the
form.  When the form becomes visible, the first Rich Edit
control will display the rich text in its raw form, including
formatting characters. The second will show the rich text in
its intended format.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
// you may need to change this path to suit your environment
  char const *Path = "..\\OVERVIEW.RTF";
  RichEdit1->PlainText = true;
  RichEdit1->Lines->LoadFromFile(Path);
  RichEdit2->PlainText = false;
  RichEdit2->Lines->LoadFromFile(Path);
}
```

Delphi Examples:

```
{
This example requires two TRichEdit controls placed on the
form.  When the form becomes visible, the first Rich Edit
control will display the rich text in its raw form, including
formatting characters. The second will show the rich text in
its intended format.
}
```

```
procedure TForm1.FormCreate(Sender: TObject);
const
  // you may need to change this path to suit your environment
  Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := True;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
  RichEdit2.PlainText := False;
  RichEdit2.Lines.LoadFromFile(Path);
  RichEdit2.ScrollBars := ssVertical;
end;
```

See Also

DefaultConverter (see page 3878)

Lines (see page 3882)

RegisterConversionFormat (see page 3874)

47.1.13.5.34 TDBRichEdit.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
```

```
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
```

```
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
```

```
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

47.1.13.5.35 TDBRichEdit.ReadOnly Property

Determines whether the user can use the rich database edit control to change the value of the field in the current record.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set ReadOnly to true to prevent the user from changing the contents of the database field. When ReadOnly is true, the edit control is used only to display the contents of the field. If ReadOnly is false, the user can change the field's value as long as the dataset is in edit mode.

See Also

Change (see page 9357)

Field (see page 9376)

Displaying and Editing Text in a Rich Edit Memo Control

47.1.13.5.36 TDBRichEdit.ScrollBars Property

Determines whether the memo control has scroll bars.

Pascal

```
property ScrollBars: TScrollStyle;
```

C++

```
__property TScrollStyle ScrollBars;
```

Description

Use ScrollBars to give a multiline edit control horizontal or vertical scroll bars. ScrollBars can take one of the following values:

Value	Meaning
ssNone	The control has no scroll bars.
ssHorizontal	The control has a single scroll bar on the bottom edge.
ssVertical	The control has a single scroll bar on the right edge.
ssBoth	The control has a scroll bar on both the bottom and right edges.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
```

```
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
    ssHorizontal,
    ssNone,
    ssVertical);
begin
  RichEdit1.ScrollBars :=
```

```
    ScrollBarA[RadioGroup1.ItemIndex];
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with RadioGroup1, Items do
  begin
    Caption := 'Scroll Bars';
    Clear;
    Add('ssBoth');
    Add('ssHorizontal');
    Add('ssNone');
    Add('ssVertical');
  end;
  with CheckBox1 do
  begin
    Caption := 'HideScrollBars';
    Checked := True;
  end;
  Button1.Caption := 'Loaded';
  RichEdit1.Text := 'Just a little text.';
  RichEdit1.WordWrap := False;
end;
```

See Also

SetScrollBars (see page 21378)

WordWrap (see page 21384)

Displaying and Editing Text in a Memo Control

47.1.13.5.37 TDBRichEdit.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
```

```
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

47.1.13.5.38 TDBRichEdit.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

47.1.13.5.39 TDBRichEdit.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

47.1.13.5.40 TDBRichEdit.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;
```

```
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

47.1.13.5.41 TDBRichEdit.WantReturns Property

Determines whether the user can insert return characters into the text.

Pascal

```
property WantReturns: Boolean;
```

C++

```
__property Boolean WantReturns;
```

Description

Set WantReturns to true to allow users to enter return characters into the text. Set WantReturns to false to allow the form to handle return characters instead.

For example, in a form with a default button (such as an OK button) and a memo control, if WantReturns is false, pressing Enter chooses the default button. If WantReturns is true, pressing Enter inserts a return character in the text.

Note: If WantReturns is false, users can still enter return characters into the text by pressing Ctrl+Enter.

See Also

Text (see page 5842)

WantTabs (see page 21384)

WordWrap (see page 21384)

47.1.13.5.42 TDBRichEdit.WantTabs Property

Determines whether the user can insert tab characters into the text.

Pascal

```
property WantTabs: Boolean;
```

C++

```
__property Boolean WantTabs;
```

Description

Set WantTabs to true to allow users to enter tab characters into the text. Set WantTabs to false if you want the tab character to select the next control on the form instead.

Note: If WantTabs is true, users can tab into the edit control, but they can't tab out.

See Also

Text (see page 5842)

WantReturns (see page 21384)

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

47.1.13.5.43 TDBRichEdit.WordWrap Property

Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to make the edit control wrap text at the right margin so it fits in the client area. The wrapping is cosmetic only. The text does not include any return characters that were not explicitly entered. Set WordWrap to false to have the edit control show a separate line only where return characters were explicitly entered into the text.

Note: There should be no use for a horizontal scroll bar if WordWrap is true.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
```

```
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
```

```
      ssHorizontal,
      ssNone,
      ssVertical);
  begin
    RichEdit1.ScrollBars :=
      ScrollBarA[RadioGroup1.ItemIndex];
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with RadioGroup1, Items do
    begin
      Caption := 'Scroll Bars';
      Clear;
      Add('ssBoth');
      Add('ssHorizontal');
      Add('ssNone');
      Add('ssVertical');
    end;
    with CheckBox1 do
    begin
      Caption := 'HideScrollBars';
      Checked := True;
    end;
    Button1.Caption := 'Loaded';
    RichEdit1.Text := 'Just a little text.';
    RichEdit1.WordWrap := False;
  end;
```

See Also

Lines (see page 21380)

ScrollBars (see page 21381)

Text (see page 5842)

WantReturns (see page 21384)

Displaying and Editing Text in a Memo Control

47.1.13.6 TDBRichEdit Events

The events of the TDBRichEdit class are listed here.

Published Events

	Name	Description
	OnChange (see page 9401)	Occurs when the text for the edit control may have changed.
	OnClick (see page 9401)	Occurs when the user clicks the control.
	OnContextPopup (see page 9403)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 9404)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 9404)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 9406)	Occurs when the user drags an object over a control.
	OnEndDock (see page 9409)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 9409)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 9410)	Occurs when a control receives the input focus.
	OnExit (see page 9413)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 9414)	Occurs when a user presses any key while the control has focus.

	OnKeyPress (see page 9417)	Occurs when a key is pressed.
	OnKeyUp (see page 9418)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 9420)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 9421)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 9423)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 9423)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 9424)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 9426)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnProtectChange (see page 9428)	Occurs when a user attempts to modify text that is marked as protected.
	OnResizeRequest (see page 9429)	Occurs when the text has grown either smaller or larger than the control's window size.
	OnSaveClipboard (see page 9430)	Occurs when the window of the edit control is about to be destroyed and the edit control has delayed rendering of text copied to the clipboard..
	OnSelectionChange (see page 9430)	Occurs when the current selection changes.
	OnStartDock (see page 9431)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 9432)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

47.1.13.6.1 TDBRichEdit.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change (see page 21245)

Text (see page 5842)

47.1.13.6.2 TDBRichEdit.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
```

```
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

47.1.13.6.3 TDBRichEdit.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

47.1.13.6.4 TDBRichEdit.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

47.1.13.6.5 TDBRichEdit.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
```

```
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
```

```
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

47.1.13.6.6 TDBRichEdit.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.13.6.7 TDBRichEdit.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.13.6.8 TDBRichEdit.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
```

```
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

47.1.13.6.9 TDBRichEdit.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter

event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
```

```
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
```

```
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

47.1.13.6.10 TDBRichEdit.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

47.1.13.6.11 TDBRichEdit.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
```

```
  event to FormKeyUp.  Set the KeyPreview on the form to True.
  }
  var
    FormColor: TColor;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    KeyPreview := True;
    FormColor := Form1.Color;
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
  {
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
```

```
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

47.1.13.6.12 TDBRichEdit.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not

generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

47.1.13.6.13 TDBRichEdit.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
```

```
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
```

```
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

47.1.13.6.14 TDBRichEdit.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate

event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.13.6.15 TDBRichEdit.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
```

```
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
```

```
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.13.6.16 TDBRichEdit.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

47.1.13.6.17 TDBRichEdit.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

47.1.13.6.18 TDBRichEdit.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.13.6.19 TDBRichEdit.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
```

```
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
```

```
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

47.1.13.6.20 TDBRichEdit.OnProtectChange Event

Occurs when a user attempts to modify text that is marked as protected.

Pascal

```
property OnProtectChange: TRichEditProtectChange;
```

C++

```
__property TRichEditProtectChange OnProtectChange;
```

Description

Write an OnProtectChange event handler to allow modification of text with the protected attribute. The StartPos and EndPos parameters indicate the beginning and end of the range of text that would be affected by the modification, where 0 is the first character in the text. To allow modification of protected text, set AllowChange to true.

To set or unset the protected attribute for a section of text, select the text and use the Protected property of the rich edit control's SelAttributes property. To cause text to have the protected attribute by default, use the Protected property of the rich edit object's DefAttributes property.

See Also

DefAttributes (see page 3877)

SelAttributes (see page 3887)

SelLength

SelStart

47.1.13.6.21 TDBRichEdit.OnResizeRequest Event

Occurs when the text has grown either smaller or larger than the control's window size.

Pascal

```
property OnResizeRequest: TRichEditResizeEvent;
```

C++

```
__property TRichEditResizeEvent OnResizeRequest;
```

Description

Write an OnResizeRequest event handler to respond to notifications that the edit control is trying to resize due to changes in the text. The Rect parameter gives the desired bounding rectangle that will display all the text, in screen coordinates.

C++ Examples:

```
/*
The following code resizes the rich edit control to
accommodate growing or shrinking lines of text.  This
example requires a button, a TEdit and a TRichEdit.
Double click on the TRichEdit OnResizeRequest event
to create the event handler.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Add(Edit1->Text);
}

void __fastcall TForm1::RichEdit1ResizeRequest(TObject *Sender, TRect &Rect)
{
  ((TRichEdit *)Sender)->BoundsRect = Rect;
}
```

Delphi Examples:

```
{
The following code resizes the rich edit control to
accommodate growing or shrinking lines of text.  This
example requires a button, a TEdit and a TRichEdit.
Double click on the TRichEdit OnResizeRequest event
to create the event handler.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  RichEdit1.Lines.Add(Edit1.Text);
end;

procedure TForm1.RichEdit1ResizeRequest(Sender: TObject; Rect: TRect);
begin
  (Sender as TRichEdit).BoundsRect := Rect;
end;
```

See Also

BoundsRect (see page 5803)

ClientRect

Text (see page 5842)

47.1.13.6.22 TDBRichEdit.OnSaveClipboard Event

Occurs when the window of the edit control is about to be destroyed and the edit control has delayed rendering of text copied to the clipboard..

Pascal

```
property OnSaveClipboard: TRichEditSaveClipboard;
```

C++

```
__property TRichEditSaveClipboard OnSaveClipboard;
```

Description

Write an OnSaveClipboard event handler to respond to notifications that the window of the edit control is about to be destroyed and the Clipboard needs information from the edit control. The parameter NumObjects is the number of objects on the Clipboard, and NumChars is the number of characters on the Clipboard. To clear all information about the edit control from the Clipboard, set SaveClipboard to false.

See Also

CopyToClipboard (see page 21246)

CutToClipboard (see page 21249)

47.1.13.6.23 TDBRichEdit.OnSelectionChange Event

Occurs when the current selection changes.

Pascal

```
property OnSelectionChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSelectionChange;
```

Description

Write an OnSelectionChange event handler to respond to changes in the selected text. Changes in selection can occur when the user changes the selection using the mouse or keyboard, or when the selection is changed through the SelText property. Portions of the selection can be changed through the Lines property.

C++ Examples:

```
/*
This example requires a TStatusBar and a TRichEdit.  The
code within the form's OnCreate will take care of sizing the
Rich Edit and setting the OnSelectionChange event to be
handled by RichEdit1SelectionChange (or you can paste the
code directly into the OnSelectionChange event handler.)  As
you type in the Rich Edit control, the current character
location the selection begin and end will display in the
status bar control.
*/
void __fastcall TForm1::RichEdit1SelectionChange(TObject *Sender)
{
  StatusBar1->SimpleText = "  Selection Start: " + IntToStr(RichEdit1->SelStart) +
      "  Selection End: " + IntToStr(RichEdit1->SelStart + RichEdit1->SelLength);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  char const *Path = "..\\OVERVIEW.RTF";
```

```
  RichEdit1->Lines->LoadFromFile(Path);
  RichEdit1->Align = alClient;
  RichEdit1->OnSelectionChange = RichEdit1SelectionChange;
  StatusBar1->SimplePanel = True;
}
```

Delphi Examples:

```
{
This example requires a TStatusBar and a TRichEdit.  The
code within the form's OnCreate will take care of sizing the
Rich Edit and setting the OnSelectionChange event to be
handled by RichEdit1SelectionChange (or you can paste the
code directly into the OnSelectionChange event handler.)  As
you type in the Rich Edit control, the current character
location the selection begin and end will display in the
status bar control.
}
procedure TForm1.RichEdit1SelectionChange(Sender: TObject);
begin
  with RichEdit1 do
    begin
    StatusBar1.SimpleText := '  Selection Start: ' + IntToStr(SelStart) +
      '  Selection End: ' + IntToStr(SelStart + SelLength);
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
const
  Path = 'OVERVIEW.RTF';
begin
  with RichEdit1 do
  begin
    Lines.LoadFromFile(Path);
    Align := alClient;
    OnSelectionChange := RichEdit1SelectionChange;
  end;
  StatusBar1.SimplePanel := True;
end;
```

See Also

AutoSelect (see page 21260)

ClearSelection (see page 21245)

Lines (see page 3882)

SelAttributes (see page 3887)

SelectAll (see page 21255)

SelLength

SelStart

SelText

SetSelTextBuf (see page 21258)

47.1.13.6.24 TDBRichEdit.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

47.1.13.6.25 TDBRichEdit.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

47.1.14 TDBText Class

TDBText represents a data-aware control that displays the value of a field on a form.

Class Hierarchy

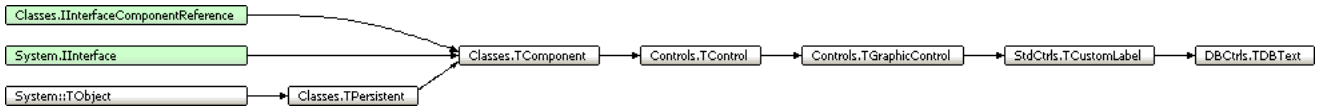

Pascal

```
TDBText = class(TCustomLabel);
```

C++

```
class TDBText : public TCustomLabel;
```

File

DBCtrls

Description

Use TDBText to display the contents of a field in the current record of a dataset on a form. Field (see page 9456) values displayed by database text controls cannot be modified by the user using the text control. To allow the user to edit the field value, use TDBEdit (see page 8735) or TDBMemo (see page 9149) instead.

If the application does not require the data-aware capabilities of TDBText, use the label component (TLabel) or the static text component (TStaticText) instead to conserve system resources.

See Also

TDBEdit (see page 8735)

TLabel (see page 21544)

TDBMemo (see page 9149)

TStaticText (see page 21887)

Displaying Data as Labels

Using Common Data Control Features

47.1.14.1 TDBText Members

The following tables list the members exposed by TDBText.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustBounds (see page 21296)	Resizes the label when the text changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDrawText (see page 21296)	Renders the text in the client area.

	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLabelText (see page 9442)	Returns the value of the Caption property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 9443)	Adjusts the size of the label when it is first loaded. .
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 9443)	Responds to notifications when objects are about to be created or destroyed.
V	Paint (see page 6052)	Render control surface.
V	Paint (see page 21298)	Draws the image of the label.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Resize (see page 5780)	Respond to control resize.

	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 9443)	Sets the AutoSize property.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	EllipsisPosition (see page 21305)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	FocusControl (see page 21306)	Designates a windowed control associated with the label.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	Layout (see page 21306)	Specifies the vertical placement of the text within the label.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ShowAccelChar (see page 21306)	Determines how an ampersand in the label text is displayed.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 9441)	Creates an instance of TDBText (see page 9433).
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 9441)	Destroys an instance of TDBText (see page 9433).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
V	ExecuteAction (see page 9442)	Invokes an action with the TDBText (see page 9433) as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	UpdateAction (see page 9444)	Updates an action to reflect the current state of the TDBText (see page 9433).
V	UseRightToLeftAlignment (see page 9444)	Specifies whether the control's alignment is in a right-to-left mode.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Canvas (see page 21300)	Provides the drawing surface used by the graphic control.
	Caption (see page 21303)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Field (see page 9456)	Indicates the TField object for the database field the database text control represents.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnClick (see page 9468)	Occurs when the user clicks the control.
	OnContextPopup (see page 9470)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 9471)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 9471)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 9473)	Occurs when the user drags an object over a control.
	OnEndDock (see page 9476)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

	OnEndDrag (see page 9476)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 9477)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 9478)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 9480)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 9481)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 9481)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 9483)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 9485)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 9486)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 9445)	Determines how the control aligns within its container (parent control).
	Alignment (see page 9449)	Controls the horizontal placement of the text within the label.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 9449)	Specifies how the control is anchored to its parent.
	AutoSize (see page 9450)	Determines whether the database text control automatically resizes to the width and length of the current string.
	BiDiMode (see page 9450)	Specifies the bi-directional mode for the control.
	Color (see page 9451)	Specifies the background color of the control.
	Constraints (see page 9452)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DataField (see page 9452)	Specifies the field whose value is displayed by the database text control.
	DataSource (see page 9453)	Links the text control to the dataset that contains the field it represents.
	DragCursor (see page 9453)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 9453)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 9454)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 9455)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 9456)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 9457)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 9458)	Determines where a control looks for its color information.
	ParentFont (see page 9458)	Determines where a control looks for its font information.
	ParentShowHint (see page 9459)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 9459)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 9464)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.

	Transparent (see page 9465)	Specifies whether controls that sit below the label on a form can be seen through the label.
	Visible (see page 9466)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	WordWrap (see page 9467)	Specifies whether the label text wraps when it is too long for the width of the label.

47.1.14.2 TDBText.Create Constructor

Creates an instance of TDBText (see page 9433).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDBText(TComponent * AOwner);
```

Description

Call Create to instantiate a TDBText (see page 9433) object at runtime. For TDBText (see page 9433) objects placed on forms or data modules at design time, Create is called automatically.

The AOwner parameter specifies the Owner for the TDBText (see page 9433) control. The Owner is responsible for freeing the control. Typically, this is a form.

See Also

AutoSize (see page 9450)

ControlStyle (see page 5814)

Create (see page 21295)

TFieldDataLink (see page 9487)

47.1.14.3 TDBText.Destroy Destructor

Destroys an instance of TDBText (see page 9433).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBText();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TDBText (see page 9433) reference is not nil and only then calls Destroy.

Destroy frees the helper objects of the TDBText (see page 9433) before calling the inherited destructor.

See Also

Free (see page 22300)

Create (see page 9441)

47.1.14.4 TDBText Methods

The methods of the TDBText class are listed here.

Protected Methods

	Name	Description
	GetLabelText (see page 9442)	Returns the value of the Caption property.
	Loaded (see page 9443)	Adjusts the size of the label when it is first loaded. .
	Notification (see page 9443)	Responds to notifications when objects are about to be created or destroyed.
	SetAutoSize (see page 9443)	Sets the AutoSize property.

Public Methods

	Name	Description
	ExecuteAction (see page 9442)	Invokes an action with the TDBText (see page 9433) as its target.
	UpdateAction (see page 9444)	Updates an action to reflect the current state of the TDBText (see page 9433).
	UseRightToLeftAlignment (see page 9444)	Specifies whether the control's alignment is in a right-to-left mode.

47.1.14.4.1 TDBText.ExecuteAction Method

Invokes an action with the TDBText (see page 9433) as its target.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

ExecuteAction is called automatically when the user invokes an action with the TDBText (see page 9433) as a target. The Action parameter specifies the action that was invoked. ExecuteAction returns trueif the action was successfully dispatched, and false if the TDBText (see page 9433) could not handle the action.

See Also

UpdateAction (see page 9444)

Execute (see page 2712)

HandlesTarget (see page 2713)

47.1.14.4.2 TDBText.GetLabelText Method

Returns the value of the Caption property.

Pascal

```
function GetLabelText: string; override;
```

C++

```
virtual __fastcall AnsiString GetLabelText();
```

Description

Call GetLabelText to obtain the string that appears as the text of the label.

See Also

Caption (see page 5806)

47.1.14.4.3 TDBText.Loaded Method

Adjusts the size of the label when it is first loaded. .

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

The protected Loaded method is called immediately after a saved version of the form that contains the label is loaded into memory. TCustomLabel uses this method to adjust the size of the label to the text it contains if the AutoSize property is true.

See Also

AutoSize (see page 21299)

47.1.14.4.4 TDBText.Notification Method

Responds to notifications when objects are about to be created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are created or destroyed. After calling the inherited method, Notification checks whether the windowed control specified by the FocusControl property is about to be destroyed. If the windowed control is going away, Notification sets the FocusControl property to nil (Delphi) or NULL (C++).

See Also

FocusControl (see page 21306)

47.1.14.4.5 TDBText.SetAutoSize Method

Sets the AutoSize property.

Pascal

```
procedure SetAutoSize(Value: Boolean); override;
```

C++

```
virtual __fastcall SetAutoSize(Boolean Value);
```

Description

SetAutoSize is the protected access method for the AutoSize property. It is exposed as a protected method so descendants can override it to change the implementation of the AutoSize property.

See Also

AutoSize (see page 21299)

47.1.14.4.6 TDBText.UpdateAction Method

Updates an action to reflect the current state of the TDBText (see page 9433).

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

UpdateAction is called automatically when the application is idle to give the TDBText (see page 9433) an opportunity to update associated actions. This allows actions to be grayed, checked, or to otherwise reflect the status of the TDBText (see page 9433) they target.

The Action parameter specifies the action component that should be updated. UpdateAction returns trueif the action component now reflects the state of the component, and false if it did not know how to update the action.

See Also

ExecuteAction (see page 9442)

UpdateTarget (see page 2715)

HandlesTarget (see page 2713)

47.1.14.4.7 TDBText.UseRightToLeftAlignment Method

Specifies whether the control's alignment is in a right-to-left mode.

Pascal

```
function UseRightToLeftAlignment: Boolean; override;
```

C++

```
virtual __fastcall Boolean UseRightToLeftAlignment();
```

Description

Call UseRightToLeftAlignment to determine whether the TDBText (see page 9433)'s alignment is in a right-to-left mode. This information can be used when painting the TDBText (see page 9433).

UseRightToLeftAlignment returns true for middle east locales if the BiDiMode (see page 9450) property is bdRightToLeft. Otherwise, it returns false.

See Also

UseRightToLeftReading (see page 5790)

BiDiMode (see page 5803)

47.1.14.5 TDBText Properties

The properties of the TDBText class are listed here.

Public Properties

	Name	Description
R	Field (see page 9456)	Indicates the TField object for the database field the database text control represents.

Published Properties

	Name	Description
	Align (see page 9445)	Determines how the control aligns within its container (parent control).
	Alignment (see page 9449)	Controls the horizontal placement of the text within the label.
	Anchors (see page 9449)	Specifies how the control is anchored to its parent.
	AutoSize (see page 9450)	Determines whether the database text control automatically resizes to the width and length of the current string.
	BiDiMode (see page 9450)	Specifies the bi-directional mode for the control.
	Color (see page 9451)	Specifies the background color of the control.
	Constraints (see page 9452)	Specifies the size constraints for the control.
	DataField (see page 9452)	Specifies the field whose value is displayed by the database text control.
	DataSource (see page 9453)	Links the text control to the dataset that contains the field it represents.
	DragCursor (see page 9453)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 9453)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 9454)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 9455)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 9456)	Controls the attributes of text written on or in the control.
	ParentBiDiMode (see page 9457)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 9458)	Determines where a control looks for its color information.
	ParentFont (see page 9458)	Determines where a control looks for its font information.
	ParentShowHint (see page 9459)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 9459)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 9464)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Transparent (see page 9465)	Specifies whether controls that sit below the label on a form can be seen through the label.
	Visible (see page 9466)	Determines whether the component appears onscreen.
	WordWrap (see page 9467)	Specifies whether the label text wraps when it is too long for the width of the label.

47.1.14.5.1 TDBText.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
```

```
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
```

```
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
```

```
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

47.1.14.5.2 TDBText.Alignment Property

Controls the horizontal placement of the text within the label.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Set Alignment to specify how the text of the label is justified within the ClientRect of the label control.

The effect of the Alignment property is more obvious if the WordWrap property is true and the label includes more than one line of text.

See Also

ClientRect (see page 5810)

GetLabelText (see page 21297)

Layout (see page 21306)

WordWrap (see page 21308)

47.1.14.5.3 TDBText.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the

length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

47.1.14.5.4 TDBText.AutoSize Property

Determines whether the database text control automatically resizes to the width and length of the current string.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;__property Boolean AutoSize;
```

Description

Use AutoSize to make the database text control adjust its size automatically so the client area accommodates the height and width of the field value. When AutoSize is false (the default), the control is fixed in size. When AutoSize is true, the size of the control is readjusted whenever its caption changes. The size of the control is also readjusted when the Font (see page 9456) property changes.

When WordWrap (see page 9467) is true, the width of the database text control is fixed. If AutoSize is also true, changes to the field value cause the control to change in height. When AutoSize is true and WordWrap (see page 9467) is false, the height of the control is determined by the font, and changes to the field value cause the control to change in width.

See Also

Font (see page 5823)

WordWrap (see page 21308)

Displaying Data as Labels

47.1.14.5.5 TDBText.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

47.1.14.5.6 TDBText.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

47.1.14.5.7 TDBText.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

47.1.14.5.8 TDBText.DataField Property

Specifies the field whose value is displayed by the database text control.

Pascal

```
property DataField: WideString;
```

C++

```
__property BSTR DataField;
```

Description

Use DataField to bind the database text control to a field in the dataset. To fully specify a database field, both the dataset and the field within that dataset must be defined. The DataSource (see page 9453) property of the control specifies the dataset that contains the DataField.

```
DBText1.DataField := 'Company';
DBText1->DataField = "Company";
```

See Also

DataSource (see page 9453)

Field (see page 9456)

47.1.14.5.9 TDBText.DataSource Property

Links the text control to the dataset that contains the field it represents.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Use DataSource to link the database text control to a dataset in which the data can be found. To fully specify a database field for the control, both the dataset and a field within that dataset must be defined. Use the DataField (see page 9452) property to specify the particular field within the dataset.

```
DBText1.DataSource := DataSource1;
DBText1->DataSource = DataSource1;
```

See Also

DataField (see page 9452)

47.1.14.5.10 TDBText.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

47.1.14.5.11 TDBText.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

47.1.14.5.12 TDBText.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
```

```
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

47.1.14.5.13 TDBText.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

47.1.14.5.14 TDBText.Field Property

Indicates the TField object for the database field the database text control represents.

Pascal

```
property Field: TField;
```

C++

```
__property TField Field;
```

Description

Use the Field object to change the value of the data in the field programmatically.

```
DBText1.Field.AsString := 'Santa Cruz Wharf';
DBText1->Field->AsString = "Santa Cruz Wharf";
```

See Also

DataField (see page 9452)

47.1.14.5.15 TDBText.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
```

```
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

47.1.14.5.16 TDBText.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

47.1.14.5.17 TDBText.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

47.1.14.5.18 TDBText.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

47.1.14.5.19 TDBText.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

47.1.14.5.20 TDBText.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
```

```
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
```

```
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
```

```
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
```

```
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
```

```
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

47.1.14.5.21 TDBText.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
```

```
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

47.1.14.5.22 TDBText.Transparent Property

Specifies whether controls that sit below the label on a form can be seen through the label.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Set Transparent to true to prevent the label from obscuring other controls on the form. For example, if the label is used to add text to a graphic, set Transparent to true so that the label does not stand out as a separate object.

Writing text so that the background is transparent is slower than writing text when Transparent is false. If the label is not obscuring a complicated image, performance can be improved by setting the background color of the label to match the object beneath it and setting Transparent to false.

See Also

Color (see page 5812)

ParentColor (see page 5834)

47.1.14.5.23 TDBText.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
```

```
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

47.1.14.5.24 TDBText.WordWrap Property

Specifies whether the label text wraps when it is too long for the width of the label.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the label to display multiple line of text. When WordWrap is true, text that is too wide for the label control wraps at the right margin and continues in additional lines.

Set WordWrap to false to limit the label to a single line. When WordWrap is false, text that is too wide for the label appears truncated.

See Also

ClientWidth (see page 5811)

GetLabelText (see page 21297)

47.1.14.6 TDBText Events

The events of the TDBText class are listed here.

Published Events

	Name	Description
	OnClick (see page 9468)	Occurs when the user clicks the control.
	OnContextPopup (see page 9470)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 9471)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 9471)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 9473)	Occurs when the user drags an object over a control.
	OnEndDock (see page 9476)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 9476)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 9477)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 9478)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 9480)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 9481)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 9481)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 9483)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 9485)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 9486)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

47.1.14.6.1 TDBText.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B.

However, focus does not move to the control in these instances.

- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
```

```
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

47.1.14.6.2 TDBText.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

47.1.14.6.3 TDBText.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

47.1.14.6.4 TDBText.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
```

```
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
```

```
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

47.1.14.6.5 TDBText.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
```

```
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.14.6.6 TDBText.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

47.1.14.6.7 TDBText.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

47.1.14.6.8 TDBText.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.14.6.9 TDBText.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
```

```
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
```

```
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.14.6.10 TDBText.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

47.1.14.6.11 TDBText.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

47.1.14.6.12 TDBText.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;
```

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

47.1.14.6.13 TDBText.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
```

```
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
```

```
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

47.1.14.6.14 TDBText.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

47.1.14.6.15 TDBText.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

47.1.15 TFieldDataLink Class

TFieldDataLink helps a data-aware windowed control manage its link to a TField object and respond to data events.

Class Hierarchy

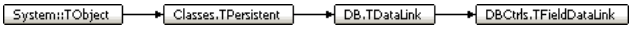

Pascal

```
TFieldDataLink = class(TDataLink);
```

C++

```
class TFieldDataLink : public TDataLink;
```

File

DBCtrls

Description

Use TFieldDataLink as a member in any data-aware object that needs to link to a TField object and respond to data events or track database information. The data-aware object must be a descendant of TWin WidgetControl (see page 9495).

The constructor of the data-aware object should call the constructor of the TFieldDataLink member, and initialize any relevant event-handlers. The data-aware object can then link to a TField object by using the FieldName (see page 9496) property of the TFieldDataLink and can link to a dataset by using the DataSource property of the TFieldDataLink.

See Also

TDataSource (see page 7070)

TField (see page 7115)

TWinControl (see page 6112)

TWidgetControl

47.1.15.1 TFieldDataLink Members

The following tables list the members exposed by TFieldDataLink.

Protected Methods

	Name	Description
	ActiveChanged (see page 9490)	Generates an OnActiveChanged event.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CheckBrowseMode (see page 6880)	Allows the data-aware object to respond just before an event that changes the dataset state.
	DataEvent (see page 9491)	This is DataEvent, a member of class TFieldDataLink.
	DataSetChanged (see page 6881)	Responds to changes in the dataset.
	DataSetScrolled (see page 6882)	Allows a response to scrolling the representation of the dataset within the data-aware object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EditingChanged (see page 9491)	Enforces the CanModify (see page 9494) property when the field binding changes.
	FocusControl (see page 9492)	Sets the input focus to Control (see page 9495).
	GetActiveRecord (see page 6884)	This is GetActiveRecord, a member of class TDataLink.

	GetBOF (see page 6884)	This is GetBOF, a member of class TDataLink.
	GetBufferCount (see page 6884)	This is GetBufferCount, a member of class TDataLink.
	GetEOF (see page 6885)	This is GetEOF, a member of class TDataLink.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRecordCount (see page 6885)	This is GetRecordCount, a member of class TDataLink.
	LayoutChanged (see page 9492)	Ensures that the Field (see page 9496) property is valid after changes in the layout of a container of the Control (see page 9495) for this TFieldDataLink (see page 9487).
	MoveBy (see page 6885)	Repositions the dataset to another record relative to its active record.
	RecordChanged (see page 9493)	Generates an OnDataChange (see page 9497) event.
	SetActiveRecord (see page 6886)	This is SetActiveRecord, a member of class TDataLink.
	SetBufferCount (see page 6887)	This is SetBufferCount, a member of class TDataLink.
	UpdateData (see page 9494)	Generates an OnUpdateData (see page 9498) event.

Protected Properties

	Name	Description
	VisualControl (see page 6894)	Indicates whether a visual control uses the datalink.

Public Events

	Name	Description
	OnActiveChange (see page 9497)	Occurs when the Active property of the TFieldDataLink (see page 9487) changes.
	OnDataChange (see page 9497)	Occurs when the contents of the dataset change.
	OnEditingChange (see page 9497)	Occurs after the DataSource transitions into or out of an editing state.
	OnUpdateData (see page 9498)	Occurs when it is time to post any pending changes to the dataset.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 9489)	Creates an instance of TFieldDataLink (see page 9487).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 6879)	Destroys an instance of TDataLink (see page 6876).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 9491)	Attempts to drive the DataSource into an editing state if it is not already in one.
	ExecuteAction (see page 6883)	Invokes an action with the datasource as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	Modified (see page 9492)	Keeps track of changes made to the data for this field that are not yet written to the record in the dataset.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Reset (see page 9493)	Ignores all pending changes and generates an OnDataChange (see page 9497) event.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	UpdateAction (see page 6887)	Updates an action to reflect the current state of the TDataLink (see page 6876).
	UpdateRecord (see page 6888)	Responds to notifications that edits should be posted to the dataset.

Public Properties

	Name	Description
R	Active (see page 6888)	Specifies whether the dataset that TDataLink (see page 6876) manages is active.
	ActiveRecord (see page 6889)	Specifies the index of the current record within the internal record buffer maintained by the dataset for the Owner of the TDataLink (see page 6876) object.
R	BOF (see page 6889)	Indicates whether the first record in a dataset is active.
	BufferCount (see page 6890)	Specifies the number of records cached by the dataset for the Owner of the TDataLink (see page 6876) object.
R	CanModify (see page 9494)	Indicates whether the Control (see page 9495) can modify a field in the database table managed by TFieldDataLink (see page 9487).
	Control (see page 9495)	Specifies the data-aware component that uses this TFieldDataLink (see page 9487) object to manage its link to a TField object.
R	DataSet (see page 6890)	Specifies the TDataSet (see page 6894) object that represents the data that this TDataLink (see page 6876) is tracking.
	DataSource (see page 6891)	Specifies the TDataSource (see page 7070) object that the Owner of the TDataLink (see page 6876) uses to connect to the dataset.
	DataSourceFixed (see page 6891)	Indicates whether the DataSource (see page 6891) property can be set.
R	Editing (see page 9495)	Indicates whether the dataset that contains the associated field is in edit mode.
R	Eof (see page 6892)	Indicates whether the last record in a dataset is active.
R	Field (see page 9496)	Specifies the field component managed by this TFieldDataLink (see page 9487) object.
	FieldName (see page 9496)	Specifies the name of the field to which the Control (see page 9495) of this TFieldDataLink (see page 9487) is bound.
	ReadOnly (see page 6893)	Indicates whether the dataset for this TDataLink (see page 6876) can be modified.
R	RecordCount (see page 6893)	Indicates the number of records in the internal record buffer maintained by the dataset.

47.1.15.2 TFieldDataLink.Create Constructor

Creates an instance of TFieldDataLink (see page 9487).

Pascal

```
constructor Create;
```

C++

```
__fastcall TFieldDataLink();
```

Description

Applications should never need to create a TFieldDataLink (see page 9487) component. Component writers will need to

instantiate a TFieldDataLink (see page 9487) object from the constructor of data-aware controls. To do so, use the new keyword, which returns an indirect reference to the new TFieldDataLink (see page 9487) object.

See Also

Destroy

47.1.15.3 TFieldDataLink Methods

The methods of the TFieldDataLink class are listed here.

Protected Methods

	Name	Description
	ActiveChanged (see page 9490)	Generates an OnActiveChanged event.
	DataEvent (see page 9491)	This is DataEvent, a member of class TFieldDataLink.
	EditingChanged (see page 9491)	Enforces the CanModify (see page 9494) property when the field binding changes.
	FocusControl (see page 9492)	Sets the input focus to Control (see page 9495).
	LayoutChanged (see page 9492)	Ensures that the Field (see page 9496) property is valid after changes in the layout of a container of the Control (see page 9495) for this TFieldDataLink (see page 9487).
	RecordChanged (see page 9493)	Generates an OnDataChange (see page 9497) event.
	UpdateData (see page 9494)	Generates an OnUpdateData (see page 9498) event.

Public Methods

	Name	Description
	Edit (see page 9491)	Attempts to drive the DataSource into an editing state if it is not already in one.
	Modified (see page 9492)	Keeps track of changes made to the data for this field that are not yet written to the record in the dataset.
	Reset (see page 9493)	Ignores all pending changes and generates an OnDataChange (see page 9497) event.

47.1.15.3.1 TFieldDataLink.ActiveChanged Method

Generates an OnActiveChanged event.

Pascal

```
procedure ActiveChanged; override;
```

C++

```
virtual __fastcall ActiveChanged();
```

Description

Changes to the Active property trigger the ActiveChanged method. If an OnActiveChange (see page 9497) event handler is assigned, ActiveChanged calls this event handler. If ActiveChanged is triggered by a transition into an active state, then before calling the event handler, ActiveChanged makes sure that the Field (see page 9496) for this TFieldDataLink (see page 9487) is still valid.

See Also

Active (see page 6888)

Field (see page 9496)

OnActiveChange (see page 9497)

47.1.15.3.2 TFieldDataLink.DataEvent Method

Pascal

```
procedure DataEvent(Event: TDataEvent; Info: Integer); override;
```

C++

```
virtual __fastcall DataEvent(TDataEvent Event, int Info);
```

Description

This is DataEvent, a member of class TFieldDataLink.

47.1.15.3.3 TFieldDataLink.Edit Method

Attempts to drive the DataSource into an editing state if it is not already in one.

Pascal

```
function Edit: Boolean;
```

C++

```
__fastcall Boolean Edit();
```

Description

Use Edit to try to ensure that the contents of the field can be modified. A return value of true indicates that the field was already in an editing state, or that the DataSource was successfully changed to allow editing. A return value of false indicates that the DataSource could not be changed to allow editing. For example, if the CanModify (see page 9494) property is false, Edit fails, and returns false.

See Also

CanModify (see page 9494)

DataSource (see page 6891)

Editing (see page 9495)

Field (see page 9496)

47.1.15.3.4 TFieldDataLink.EditingChanged Method

Enforces the CanModify (see page 9494) property when the field binding changes.

Pascal

```
procedure EditingChanged; override;
```

C++

```
virtual __fastcall EditingChanged();
```

Description

Changing the field binding can change the validity of the CanModify (see page 9494) property, since individual field components can disallow edits. If TFieldDataLink (see page 9487) is in an editing state when the Field (see page 9496) property is changed, EditingChanged checks the CanModify (see page 9494) property. If CanModify (see page 9494) is false, it changes back out of the editing state.

Note: This differs significantly from theinherited EditingChanged method of TDataLink. The functionality of the inherited method is replaced in TFieldDataLink (see page 9487) by the OnEditingChange (see page 9497) event handler.

See Also

CanModify (see page 9494)

Editing (see page 9495)

Field (see page 9496)

OnEditingChange (see page 9497)

47.1.15.3.5 TFieldDataLink.FocusControl Method

Sets the input focus to Control (see page 9495).

Pascal

```
procedure FocusControl(Field: TFieldRef); override;
```

C++

```
virtual __fastcall FocusControl(TFieldRef Field);
```

Description

Call FocusControl to give the Control (see page 9495) associated with this TFieldDataLink (see page 9487) object the input focus. FocusControl checks whether the Control (see page 9495) can receive input focus, and if so, calls its SetFocus method to move focus to the Control (see page 9495).

See Also

Control (see page 9495)

SetFocus

47.1.15.3.6 TFieldDataLink.LayoutChanged Method

Ensures that the Field (see page 9496) property is valid after changes in the layout of a container of the Control (see page 9495) for this TFieldDataLink (see page 9487).

Pascal

```
procedure LayoutChanged; override;
```

C++

```
virtual __fastcall LayoutChanged();
```

Description

LayoutChanged is called after changes in the layout of one of the containers of the Control (see page 9495) for this TFieldDataLink (see page 9487) that might change the validity of its field binding. For example, if the Control (see page 9495) is embedded within a TCustomDBGrid, and one of the columns is deleted, the Field (see page 9496) property for the Control (see page 9495) might become invalid.

See Also

Control (see page 9495)

Field (see page 9496)

47.1.15.3.7 TFieldDataLink.Modified Method

Keeps track of changes made to the data for this field that are not yet written to the record in the dataset.

Pascal

```
procedure Modified;
```

C++

```
__fastcall Modified();
```

Description

Call Modified when the Control (see page 9495) for this TFieldDataLink (see page 9487) begins processing edits.

See Also

Control (see page 9495)

Editing (see page 9495)

RecordChanged (see page 9493)

47.1.15.3.8 TFieldDataLink.RecordChanged Method

Generates an OnDataChange (see page 9497) event.

Pascal

```
procedure RecordChanged(Field: TField); override;
```

C++

```
virtual __fastcall RecordChanged(TField Field);
```

Description

Applications can not call this protected method. It is triggered automatically when the contents of the current record change. RecordChanged calls the OnDataChange (see page 9497) event handler if there is one.

See Also

OnDataChange (see page 9497)

UpdateData (see page 9494)

47.1.15.3.9 TFieldDataLink.Reset Method

Ignores all pending changes and generates an OnDataChange (see page 9497) event.

Pascal

```
procedure Reset;
```

C++

```
__fastcall Reset();
```

Description

The Control (see page 9495) that owns a TFieldDataLink (see page 9487) object calls its Reset method to process a UI action that cancels edits to the field. Reset calls the OnDataChange (see page 9497) event handler without writing any pending changes to the record in the dataset.

See Also

Control (see page 9495)

OnDataChange (see page 9497)

47.1.15.3.10 TFieldDataLink.UpdateData Method

Generates an OnUpdateData (see page 9498) event.

Pascal

```
procedure UpdateData; override;
```

C++

```
virtual __fastcall UpdateData();
```

Description

UpdateData overrides the default UpdateData method to call the OnUpdateData (see page 9498) event handler where the data-aware control can write any pending edits to the record in the dataset.

See Also

OnUpdateData (see page 9498)

RecordChanged (see page 9493)

47.1.15.4 TFieldDataLink Properties

The properties of the TFieldDataLink class are listed here.

Public Properties

	Name	Description
R	CanModify (see page 9494)	Indicates whether the Control (see page 9495) can modify a field in the database table managed by TFieldDataLink (see page 9487).
	Control (see page 9495)	Specifies the data-aware component that uses this TFieldDataLink (see page 9487) object to manage its link to a TField object.
R	Editing (see page 9495)	Indicates whether the dataset that contains the associated field is in edit mode.
R	Field (see page 9496)	Specifies the field component managed by this TFieldDataLink (see page 9487) object.
	FieldName (see page 9496)	Specifies the name of the field to which the Control (see page 9495) of this TFieldDataLink (see page 9487) is bound.

47.1.15.4.1 TFieldDataLink.CanModify Property

Indicates whether the Control (see page 9495) can modify a field in the database table managed by TFieldDataLink (see page 9487).

Pascal

```
property CanModify: Boolean;
```

C++

```
__property Boolean CanModify;
```

Description

Read CanModify to find out whether the Control (see page 9495) that owns this TFieldDataLink (see page 9487) is able to alter the data in the database that this field represents. CanModify may be false because the control is read-only, or because the underlying field does not permit updates.

See Also

Control (see page 9495)

Edit (see page 9491)

Field (see page 9496)

ReadOnly (see page 6893)

47.1.15.4.2 TFieldDataLink.Control Property

Specifies the data-aware component that uses this TFieldDataLink (see page 9487) object to manage its link to a TField object.

Pascal

```
property Control: TComponent;
```

C++

```
__property TComponent * Control;
```

Description

This property should only be set by components that use TFieldDataLink (see page 9487) to manage the link to a TField object. In their own constructors, these data-aware components can set the Control property of their TFieldDataLink (see page 9487) member to self (Delphi) or this (C++), after calling the TFieldDataLink (see page 9487) constructor.

The Control property can be read to access methods of the component when processing data events.

See Also

Create (see page 6878)

OnActiveChange (see page 9497)

OnDataChange (see page 9497)

OnEditingChange (see page 9497)

OnUpdateData (see page 9498)

47.1.15.4.3 TFieldDataLink.Editing Property

Indicates whether the dataset that contains the associated field is in edit mode.

Pascal

```
property Editing: Boolean;
```

C++

```
__property Boolean Editing;
```

Description

Use Editing to determine whether the value of the field associated with this TDataLink can currently be edited. If Editing is false, calling the Edit (see page 9491) method may drive the dataset into edit mode. The Edit (see page 9491) method may not be successful, however, if the field cannot be edited. For example, if the CanModify (see page 9494) property is false, Editing will always be false.

See Also

CanModify (see page 9494)

DataSource (see page 6891)

Edit (see page 9491)

OnEditingChange (see page 9497)

47.1.15.4.4 TFieldDataLink.Field Property

Specifies the field component managed by this TFieldDataLink (see page 9487) object.

Pascal

```
property Field: TField;
```

C++

```
__property TField Field;
```

Description

Read Field to gain access to the field object to which the Control (see page 9495) of this TFieldDataLink (see page 9487) is bound.

Note: Before using Field to alter the value of a field's contents, applications should check the CanModify (see page 9494) property. The Field object will not be aware of all constraints to edits for this TFieldDataLink (see page 9487).

See Also

CanModify (see page 9494)

FieldName (see page 9496)

47.1.15.4.5 TFieldDataLink.FieldName Property

Specifies the name of the field to which the Control (see page 9495) of this TFieldDataLink (see page 9487) is bound.

Pascal

```
property FieldName: WideString;
```

C++

```
__property BSTR FieldName;
```

Description

Use FieldName to implement the FieldName property of the data-aware control that owns the TFieldDataLink (see page 9487). Setting FieldName links the TFieldDataLink (see page 9487), and hence the control, to the field in the dataset with that name.

See Also

Control (see page 9495)

DataSet (see page 6890)

Field (see page 9496)

47.1.15.5 TFieldDataLink Events

The events of the TFieldDataLink class are listed here.

Public Events

	Name	Description
	OnActiveChange (see page 9497)	Occurs when the Active property of the TFieldDataLink (see page 9487) changes.
	OnDataChange (see page 9497)	Occurs when the contents of the dataset change.
	OnEditingChange (see page 9497)	Occurs after the DataSource transitions into or out of an editing state.

	OnUpdateData (see page 9498)	Occurs when it is time to post any pending changes to the dataset.

47.1.15.5.1 TFieldDataLink.OnActiveChange Event

Occurs when the Active property of the TFieldDataLink (see page 9487) changes.

Pascal

```
property OnActiveChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActiveChange;
```

Description

Write an OnActiveChange event handler to take specific action immediately after the dataset transitions into or out of an active state. OnActiveChange is particularly useful for updating objects to reflect changes that occurred while the dataset was not active, as the object can not respond to events while its TFieldDataLink (see page 9487) is not Active.

See Also

Active (see page 6888)

ActiveChanged (see page 9490)

DataSet (see page 6890)

47.1.15.5.2 TFieldDataLink.OnDataChange Event

Occurs when the contents of the dataset change.

Pascal

```
property OnDataChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDataChange;
```

Description

Write an OnDataChange event handler to take specific action when there is a change to the contents of the field. OnDataChange allows the control to respond when the dataset sends notifications that the data has changed.

See Also

DataSet (see page 6890)

OnUpdateData (see page 9498)

RecordChanged (see page 9493)

47.1.15.5.3 TFieldDataLink.OnEditingChange Event

Occurs after the DataSource transitions into or out of an editing state.

Pascal

```
property OnEditingChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEditingChange;
```

Description

Write an OnEditingChange event handler to take specific action when the DataSource transitions into or out of an editing state. This event can be used to create visual indications of the editing state in the control that uses this TFieldDataLink (see page 9487).

See Also

DataSource (see page 6891)

Edit (see page 9491)

Editing (see page 9495)

EditingChanged (see page 9491)

47.1.15.5.4 TFieldDataLink.OnUpdateData Event

Occurs when it is time to post any pending changes to the dataset.

Pascal

```
property OnUpdateData: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdateData;
```

Description

Write an OnUpdateData event handler to write the data in the representation of the field to the database record. Posting the data from within OnUpdateData does not trigger a call to OnDataChange (see page 9497).

See Also

OnDataChange (see page 9497)

UpdateData (see page 9494)

47.1.16 TListSourceLink Class

TListSourceLink helps manage the link to the ListSource of a TDBLookupControl (see page 9044) object.

Class Hierarchy

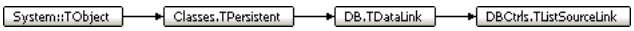

Pascal

```
TListSourceLink = class(TDataLink);
```

C++

```
class TListSourceLink : public TDataLink;
```

File

DBCtrls

Description

TListSourceLink is tailored to work with TDBLookupControl (see page 9044). It should only be used by the TDBLookupControl (see page 9044) class.

See Also

TDataSource (see page 7070)

TDBLookupControl (see page 9044)

47.1.16.1 TListSourceLink Members

The following tables list the members exposed by TListSourceLink.

Protected Methods

	Name	Description
	ActiveChanged (see page 9501)	Informs the TDBLookupControl (see page 9044) of changes to the Active property of the dataset that it maintains for the ListSource.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CheckBrowseMode (see page 6880)	Allows the data-aware object to respond just before an event that changes the dataset state.
	DataEvent (see page 6881)	Responds to various events that occur while working with data.
	DataSetChanged (see page 9502)	Informs the TDBLookupControl (see page 9044) of changes to the current record of the dataset that the TDBLookupControl (see page 9044) uses to populate its list.
	DataSetScrolled (see page 6882)	Allows a response to scrolling the representation of the dataset within the data-aware object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EditingChanged (see page 6883)	Responds to changes in the editing state of the data source.
	FocusControl (see page 6884)	Provides the interface for a method to force focus to the data-aware component associated with the TDataLink (see page 6876).
	GetActiveRecord (see page 6884)	This is GetActiveRecord, a member of class TDataLink.
	GetBOF (see page 6884)	This is GetBOF, a member of class TDataLink.
	GetBufferCount (see page 6884)	This is GetBufferCount, a member of class TDataLink.
	GetEOF (see page 6885)	This is GetEOF, a member of class TDataLink.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRecordCount (see page 6885)	This is GetRecordCount, a member of class TDataLink.
	LayoutChanged (see page 9502)	Responds to changes in the representation of the data by the data-aware object.
	MoveBy (see page 6885)	Repositions the dataset to another record relative to its active record.
	RecordChanged (see page 6886)	Responds to changes in the contents of the current record or field of the dataset.
	SetActiveRecord (see page 6886)	This is SetActiveRecord, a member of class TDataLink.
	SetBufferCount (see page 6887)	This is SetBufferCount, a member of class TDataLink.
	UpdateData (see page 6887)	Provides an interface for writing edits to a record in the dataset.

Protected Properties

	Name	Description
	VisualControl (see page 6894)	Indicates whether a visual control uses the datalink.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 9500)	Creates an instance of TDataSourceLink (see page 8577).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 6879)	Destroys an instance of TDataLink (see page 6876).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6882)	Attempts to drive the data source into an editing state if it is not already in one.
	ExecuteAction (see page 6883)	Invokes an action with the datasource as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	UpdateAction (see page 6887)	Updates an action to reflect the current state of the TDataLink (see page 6876).
	UpdateRecord (see page 6888)	Responds to notifications that edits should be posted to the dataset.

Public Properties

	Name	Description
R	Active (see page 6888)	Specifies whether the dataset that TDataLink (see page 6876) manages is active.
	ActiveRecord (see page 6889)	Specifies the index of the current record within the internal record buffer maintained by the dataset for the Owner of the TDataLink (see page 6876) object.
R	BOF (see page 6889)	Indicates whether the first record in a dataset is active.
	BufferCount (see page 6890)	Specifies the number of records cached by the dataset for the Owner of the TDataLink (see page 6876) object.
R	DataSet (see page 6890)	Specifies the TDataSet (see page 6894) object that represents the data that this TDataLink (see page 6876) is tracking.
	DataSource (see page 6891)	Specifies the TDataSource (see page 7070) object that the Owner of the TDataLink (see page 6876) uses to connect to the dataset.
	DataSourceFixed (see page 6891)	Indicates whether the DataSource (see page 6891) property can be set.
R	Editing (see page 6892)	Indicates whether the DataSource (see page 6891) is in edit mode.
R	Eof (see page 6892)	Indicates whether the last record in a dataset is active.
	ReadOnly (see page 6893)	Indicates whether the dataset for this TDataLink (see page 6876) can be modified.
R	RecordCount (see page 6893)	Indicates the number of records in the internal record buffer maintained by the dataset.

47.1.16.2 TListSourceLink.Create Constructor

Creates an instance of TDataSourceLink (see page 8577).

Pascal

```
constructor Create;
```

C++

```
__fastcall TListSourceLink();
```

Description

Create is called from the constructor of any TlookupControl descendant object that uses a TListSourceLink (see page 9498) to implement its ListSource property.

Create calls the inherited Create method and then sets VisualControl to True.

See Also

Destroy (see page 6879)

Free (see page 22300)

VisualControl (see page 6894)

47.1.16.3 TListSourceLink Methods

The methods of the TListSourceLink class are listed here.

Protected Methods

	Name	Description
	ActiveChanged (see page 9501)	Informs the TDBLookupControl (see page 9044) of changes to the Active property of the dataset that it maintains for the ListSource.
	DataSetChanged (see page 9502)	Informs the TDBLookupControl (see page 9044) of changes to the current record of the dataset that the TDBLookupControl (see page 9044) uses to populate its list.
	LayoutChanged (see page 9502)	Responds to changes in the representation of the data by the data-aware object.

47.1.16.3.1 TListSourceLink.ActiveChanged Method

Informs the TDBLookupControl (see page 9044) of changes to the Active property of the dataset that it maintains for the ListSource.

Pascal

```
procedure ActiveChanged; override;
```

C++

```
virtual __fastcall ActiveChanged();
```

Description

Applications should not call the ActiveChanged procedure directly. Changes in the Active property of the dataset trigger this procedure. ActiveChanged causes the TDBLookupControl (see page 9044) to re-acquire its lookup list.

See Also

Active (see page 6888)

DataSetChanged (see page 9502)

47.1.16.3.2 TListSourceLink.DataSetChanged Method

Informs the TDBLookupControl (see page 9044) of changes to the current record of the dataset that the TDBLookupControl (see page 9044) uses to populate its list.

Pascal

```
procedure DataSetChanged; override;
```

C++

```
virtual __fastcall DataSetChanged();
```

Description

Applications should not call the DataSetChanged procedure directly. These calls are triggered automatically by the dataset. DataSetChanged informs the TListSourceLink (see page 9498) of dataset changes so that it can move to the appropriate record in the dataset.

See Also

ActiveChanged (see page 9501)

DataSet (see page 6890)

47.1.16.3.3 TListSourceLink.LayoutChanged Method

Responds to changes in the representation of the data by the data-aware object.

Pascal

```
procedure LayoutChanged; override;
```

C++

```
virtual __fastcall LayoutChanged();
```

Description

The LayoutChanged method simply calls the DataSetChanged method. LayoutChanged is intended to provide an interface for a method that can respond after changes in the representation of the data by the data-aware object. An example of such a change would be changing the number or order of the columns in a TCustomDBGrid.

See Also

DataSetChanged (see page 6881)

47.1.17 TNavButton Class

Class Hierarchy

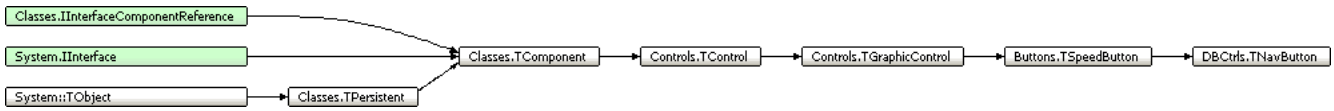

Pascal

```
TNavButton = class(TSpeedButton);
```

C++

```
class TNavButton : public TSpeedButton;
```

File

DBCtrls

Description

This is class DBCtrls.TNavButton.

47.1.17.1 TNavButton Members

The following tables list the members exposed by TNavButton.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.

	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FState (see page 2345)	This is FState, a member of class TSpeedButton.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActionChange (see page 2345)	Responds when the action associated with the control changes.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	CopyImage (see page 2346)	This is CopyImage, a member of class TSpeedButton.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
V	GetActionLinkClass (see page 2346)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.

	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 2347)	Returns the handle to a palette for use by the control.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2347)	Initializes the control after it is loaded from a stream.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 9511)	OnMouseDown event dispatcher.
	MouseMove (see page 2348)	Respond to mouse moving over control area..
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 9512)	OnMouseUp event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Paint (see page 9512)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.

V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	MouseInControl (see page 2361)	This is MouseInControl, a member of class TSpeedButton.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Click (see page 2346)	Generates an OnClick (see page 2372) event or triggers the associated action.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 2344)	Creates and initializes an instance of TSpeedButton (see page 2335).
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6051)	Creates an instance of TGraphicControl (see page 6044).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 9511)	Destroys an instance of TSpeedButton.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	Index (see page 9513)	This is Index, a member of class TNavButton.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	NavStyle (see page 9513)	This is NavStyle, a member of class TNavButton.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Events

	Name	Description
	OnClick (see page 2372)	Occurs when the user clicks the control.

	OnDblClick (see page 2374)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnMouseActivate (see page 2374)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2375)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2377)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2378)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2378)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2380)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Published Properties

	Name	Description
	Action (see page 2350)	Designates the action associated with the control.
	Align (see page 2350)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AllowAllUp (see page 2354)	Specifies whether all speed buttons in the group that contains this speed button can be unselected at the same time.
	Anchors (see page 2355)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2355)	Specifies the bi-directional mode for the control.
	Caption (see page 2356)	Specifies the caption, or label, that appears on the speed button.
	Constraints (see page 2356)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Down (see page 2357)	Specifies whether the button is selected (down) or unselected (up).
	Enabled (see page 2357)	Controls whether the control responds to mouse, keyboard, and timer events.
	Flat (see page 2357)	Determines whether the button has a 3D border that provides a raised or lowered look.
	Font (see page 2358)	Controls the attributes of text written on or in the control.
	Glyph (see page 2359)	Specifies the bitmap that appears on the speed button.
	GroupIndex (see page 2360)	Allows speed buttons to work together as a group.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Layout (see page 2360)	Determines where the image or text appears on the speed button.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margin (see page 2361)	Specifies the number of pixels between the edge of the button and the image or caption drawn on its surface.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NumGlyphs (see page 2361)	Specifies the number of images included in the Glyph (see page 2359) property.
	ParentBiDiMode (see page 2362)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 2362)	Determines where a control looks for its font information.
	ParentShowHint (see page 2363)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2363)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2368)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Spacing (see page 2369)	Determines where the image and text appear on a speed button.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.

	Transparent (see page 2370)	Specifies whether the background of the button is transparent.
	Visible (see page 2370)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

47.1.17.2 TNavButton.Destroy Destructor

Destroys an instance of TSpeedButton.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TNavButton();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the speed button is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

47.1.17.3 TNavButton Methods

The methods of the TNavButton class are listed here.

Protected Methods

	Name	Description
	MouseDown (see page 9511)	OnMouseDown event dispatcher.
	MouseUp (see page 9512)	OnMouseUp event dispatcher.
	Paint (see page 9512)	Render control surface.

47.1.17.3.1 TNavButton.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was

pressed down: left, right, or middle.

See Also

MouseMove (see page 5773)

MouseUp (see page 5774)

OnMouseDown (see page 5862)

47.1.17.3.2 TNavButton.MouseUp Method

OnMouseUp event dispatcher.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user releases a previously pressed mouse button while the cursor's hotspot is over the control.

A control calls MouseUp in response to any of the Windows mouse-up messages (WM_LBUTTONUP, WM_MBUTTONUP, WM_RBUTTONUP), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

See Also

MouseDown (see page 5773)

MouseMove (see page 5773)

OnMouseUp (see page 5867)

47.1.17.3.3 TNavButton.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas (see page 6053)

47.1.17.4 TNavButton Properties

The properties of the TNavButton class are listed here.

Public Properties

	Name	Description
	Index (see page 9513)	This is Index, a member of class TNavButton.
	NavStyle (see page 9513)	This is NavStyle, a member of class TNavButton.

47.1.17.4.1 TNavButton.Index Property

Pascal

```
property Index: TNavigateBtn;
```

C++

```
__property TNavigateBtn Index;
```

Description

This is Index, a member of class TNavButton.

47.1.17.4.2 TNavButton.NavStyle Property

Pascal

```
property NavStyle: TNavButtonStyle;
```

C++

```
__property TNavButtonStyle NavStyle;
```

Description

This is NavStyle, a member of class TNavButton.

47.1.18 TNavDataLink Class

TNavDataLink helps manage the link to the dataset for a TDBNavigator (see page 9230) object.

Class Hierarchy

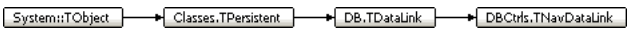

Pascal

```
TNavDataLink = class(TDataLink);
```

C++

```
class TNavDataLink : public TDataLink;
```

File

DBCtrls

Description

TNavDataLink is tailored to work with a TDBNavigator (see page 9230). It should only be used by the TDBNavigator (see

page 9230) class.

See Also

TDataSource (see page 7070)

TDBNavigator (see page 9230)

47.1.18.1 TNavDataLink Members

The following tables list the members exposed by TNavDataLink.

Protected Methods

	Name	Description
	ActiveChanged (see page 9516)	Informs TDBNavigator (see page 9230) of changes in the Active property.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CheckBrowseMode (see page 6880)	Allows the data-aware object to respond just before an event that changes the dataset state.
	DataEvent (see page 6881)	Responds to various events that occur while working with data.
	DataSetChanged (see page 9517)	Informs the TDBNavigator (see page 9230) of changes to the dataset.
	DataSetScrolled (see page 6882)	Allows a response to scrolling the representation of the dataset within the data-aware object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EditingChanged (see page 9517)	Informs TDBNavigator (see page 9230) of changes to the editing state of the DataSource.
	FocusControl (see page 6884)	Provides the interface for a method to force focus to the data-aware component associated with the TDataLink (see page 6876).
	GetActiveRecord (see page 6884)	This is GetActiveRecord, a member of class TDataLink.
	GetBOF (see page 6884)	This is GetBOF, a member of class TDataLink.
	GetBufferCount (see page 6884)	This is GetBufferCount, a member of class TDataLink.
	GetEOF (see page 6885)	This is GetEOF, a member of class TDataLink.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRecordCount (see page 6885)	This is GetRecordCount, a member of class TDataLink.
	LayoutChanged (see page 6885)	Responds to changes in the representation of the data by the data-aware object.
	MoveBy (see page 6885)	Repositions the dataset to another record relative to its active record.
	RecordChanged (see page 6886)	Responds to changes in the contents of the current record or field of the dataset.
	SetActiveRecord (see page 6886)	This is SetActiveRecord, a member of class TDataLink.
	SetBufferCount (see page 6887)	This is SetBufferCount, a member of class TDataLink.
	UpdateData (see page 6887)	Provides an interface for writing edits to a record in the dataset.

Protected Properties

	Name	Description
	VisualControl (see page 6894)	Indicates whether a visual control uses the datalink.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 9515)	Creates an instance of TNavDataLink (see page 9513).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 9516)	Destroys an instance of TNavDataLink (see page 9513).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6882)	Attempts to drive the data source into an editing state if it is not already in one.
	ExecuteAction (see page 6883)	Invokes an action with the datasource as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	UpdateAction (see page 6887)	Updates an action to reflect the current state of the TDataLink (see page 6876).
	UpdateRecord (see page 6888)	Responds to notifications that edits should be posted to the dataset.

Public Properties

	Name	Description
R	Active (see page 6888)	Specifies whether the dataset that TDataLink (see page 6876) manages is active.
	ActiveRecord (see page 6889)	Specifies the index of the current record within the internal record buffer maintained by the dataset for the Owner of the TDataLink (see page 6876) object.
R	BOF (see page 6889)	Indicates whether the first record in a dataset is active.
	BufferCount (see page 6890)	Specifies the number of records cached by the dataset for the Owner of the TDataLink (see page 6876) object.
R	DataSet (see page 6890)	Specifies the TDataSet (see page 6894) object that represents the data that this TDataLink (see page 6876) is tracking.
	DataSource (see page 6891)	Specifies the TDataSource (see page 7070) object that the Owner of the TDataLink (see page 6876) uses to connect to the dataset.
	DataSourceFixed (see page 6891)	Indicates whether the DataSource (see page 6891) property can be set.
R	Editing (see page 6892)	Indicates whether the DataSource (see page 6891) is in edit mode.
R	Eof (see page 6892)	Indicates whether the last record in a dataset is active.
	ReadOnly (see page 6893)	Indicates whether the dataset for this TDataLink (see page 6876) can be modified.
R	RecordCount (see page 6893)	Indicates the number of records in the internal record buffer maintained by the dataset.

47.1.18.2 TNavDataLink.Create Constructor

Creates an instance of TNavDataLink (see page 9513).

Pascal

```
constructor Create(ANav: TDBNavigator);
```

C++

```
__fastcall TNavDataLink(TDBNavigator ANav);
```

Description

Applications should not create instances of TNavDataLink (see page 9513). The TDBNavigator (see page 9230) that owns a TNavDataLink (see page 9513) calls Create from its own constructor.

Create calls its inherited Create constructor, and then sets VisualControl to true.

See Also

Destroy (see page 9516)

VisualControl (see page 6894)

47.1.18.3 TNavDataLink.Destroy Destructor

Destroys an instance of TNavDataLink (see page 9513).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TNavDataLink();
```

Description

Applications should not free TNavDataLink (see page 9513) objects. The TDBNavigator (see page 9230) that owns a TNavDataLink (see page 9513) calls Free to destroy the TNavDataLink (see page 9513) from its own destructor.

See Also

Free (see page 22300)

Create (see page 9515)

47.1.18.4 TNavDataLink Methods

The methods of the TNavDataLink class are listed here.

Protected Methods

	Name	Description
	ActiveChanged (see page 9516)	Informs TDBNavigator (see page 9230) of changes in the Active property.
	DataSetChanged (see page 9517)	Informs the TDBNavigator (see page 9230) of changes to the dataset.
	EditingChanged (see page 9517)	Informs TDBNavigator (see page 9230) of changes to the editing state of the DataSource.

47.1.18.4.1 TNavDataLink.ActiveChanged Method

Informs TDBNavigator (see page 9230) of changes in the Active property.

Pascal

```
procedure ActiveChanged; override;
```

C++

```
virtual __fastcall ActiveChanged();
```

Description

Applications should not call the ActiveChanged procedure directly. Changes in the Active property trigger this procedure, which in turn causes the navigator to enable or disable its buttons, as appropriate.

See Also

Active (see page 6888)

DataSet (see page 6890)

47.1.18.4.2 TNavDataLink.DataSetChanged Method

Informs the TDBNavigator (see page 9230) of changes to the dataset.

Pascal

```
procedure DataSetChanged; override;
```

C++

```
virtual __fastcall DataSetChanged();
```

Description

Applications do not need to call the DataSetChanged procedure. It is called automatically following changes to the contents of the dataset. TNavDataLink (see page 9513) informs the TDBNavigator (see page 9230) of these changes so that it can enable and disable buttons as appropriate.

See Also

ActiveChanged (see page 9516)

UpdateRecord (see page 6888)

47.1.18.4.3 TNavDataLink.EditingChanged Method

Informs TDBNavigator (see page 9230) of changes to the editing state of the DataSource.

Pascal

```
procedure EditingChanged; override;
```

C++

```
virtual __fastcall EditingChanged();
```

Description

Applications should not call the EditingChanged procedure directly. Changes to the editing state of the DataSource trigger this procedure, which in turn causes the navigator to enable or disable editing buttons as appropriate.

See Also

DataSource (see page 6891)

Edit (see page 6882)

Editing (see page 6892)

47.1.19 TPaintControl Class

TPaintControl draws data aware controls that are replicated in a data control grid.

Class Hierarchy

DBCtrls.TPaintControl

Pascal

```
TPaintControl = class;
```

C++

```
class TPaintControl;
```

File

DBCtrls

Description

Several data-aware controls use TPaintControl — including TDBCheckBox (see page 8582), TDBComboBox (see page 8657), and TDBMemo (see page 9149) - to replicate themselves within a data control grid (TDBCtrlGrid).

See Also

TDBCheckBox (see page 8582)

TDBComboBox (see page 8657)

TDBCtrlGrid (see page 8168)

TDBMemo (see page 9149)

47.1.19.1 TPaintControl Members

The following tables list the members exposed by TPaintControl.

Public Methods

	Name	Description
	Create (see page 9518)	Creates and initializes a TPaintControl (see page 9518) object.
V	Destroy (see page 9519)	Destroys the TPaintControl (see page 9518) and frees its memory.
	DestroyHandle (see page 9519)	Destroys the window used by the paint control.

Public Properties

	Name	Description
	Ctl3DButton (see page 9520)	Determines whether the control has a three-dimensional look.
R	Handle (see page 9520)	Provides access to paint control's window handle.

47.1.19.2 TPaintControl.Create Constructor

Creates and initializes a TPaintControl (see page 9518) object.

Pascal

```
constructor Create(AOwner: TWinControl; const ClassName: string);
```

C++

```
__fastcall TPaintControl(TWinControl * AOwner, const AnsiString ClassName);
```

Description

The Create method creates an instance of TPaintControl (see page 9518). The Owner parameter specifies the data-aware control that uses this TPaintControl (see page 9518) object to paint its image. The ClassName parameter specifies the Window class that should be created by the paint control.

See Also

Handle (see page 9520)

47.1.19.3 TPaintControl.Destroy Destructor

Destroys the TPaintControl (see page 9518) and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPaintControl();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TPaintControl (see page 9518) reference is not nil, and only then calls Destroy.

Destroy frees the window used by the paint control.

See Also

DestroyHandle (see page 9519)

Free (see page 22300)

47.1.19.4 TPaintControl Methods

The methods of the TPaintControl class are listed here.

Public Methods

	Name	Description
	DestroyHandle (see page 9519)	Destroys the window used by the paint control.

47.1.19.4.1 TPaintControl.DestroyHandle Method

Destroys the window used by the paint control.

Pascal

```
procedure DestroyHandle;
```

C++

```
__fastcall DestroyHandle();
```

Description

The destructor of TPaintControl (see page 9518) calls DestroyHandle to release the Windows resources allocated on behalf of

the paint control.

See Also

Destroy (see page 9519)

Free (see page 22300)

Handle (see page 9520)

47.1.19.5 TPaintControl Properties

The properties of the TPaintControl class are listed here.

Public Properties

	Name	Description
	Ctl3DButton (see page 9520)	Determines whether the control has a three-dimensional look.
R	Handle (see page 9520)	Provides access to paint control's window handle.

47.1.19.5.1 TPaintControl.Ctl3DButton Property

Determines whether the control has a three-dimensional look.

Pascal

```
property Ctl3DButton: Boolean;
```

C++

```
__property Boolean Ctl3DButton;
```

Description

Ctl3DButton is provided for backward compatibility. It has no effect on 32-bit Windows and NT 4.0 and later.

Ctl3DButton controlled whether the data-aware control had beveled borders, and required Ctl3d32.dll to be installed in the System (see page 22079) 32 directory.

47.1.19.5.2 TPaintControl.Handle Property

Provides access to paint control's window handle.

Pascal

```
property Handle: HWnd;
```

C++

```
__property HWnd Handle;
```

Description

Handle is the window handle of the TPaintControl (see page 9518) instance. It represents a window with the classname supplied to the constructor of the TPaintControl (see page 9518) object. The characteristics of the window are determined by the Owner supplied to the constructor of the paint control object.

See Also

Create (see page 9518)

47.1.20 TPopupDataList Class

Class Hierarchy

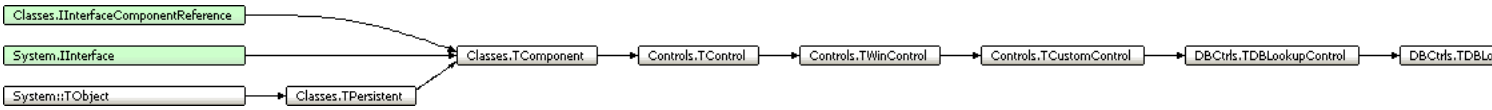

Pascal

```
TPopupDataList = class(TDBLookupListBox);
```

C++

```
class TPopupDataList : public TDBLookupListBox;
```

File

DBCtrls

Description

This is class DBCtrls.TPopupDataList.

47.1.20.1 TPopupDataList Members

The following tables list the members exposed by TPopupDataList.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.

	CanModify (see page 9057)	Indicates whether the lookup control has rights to modify data.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 9534)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 9083)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.

	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorderSize (see page 9058)	This is GetBorderSize, a member of class TDBLookupControl.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTextHeight (see page 9058)	Indicates how many pixels are required for one line of text in the list.
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyDown (see page 9084)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.

	KeyPress (see page 9085)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	KeyValueChanged (see page 9058)	Updates the control when the value of the protected KeyValue (see page 9065) property changes.
	KeyValueChanged (see page 9085)	Updates the control when the value of the protected KeyValue property changes.
	ListLinkDataChanged (see page 9058)	Updates the control when its data binding changes.
	ListLinkDataChanged (see page 9085)	Updates the control when its data binding changes.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	LocateKey (see page 9059)	Locates the value specified by KeyValue (see page 9065) in the lookup dataset.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseDown (see page 9086)	OnMouseDown event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseMove (see page 9086)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	MouseUp (see page 9087)	OnMouseUp event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	Notification (see page 9059)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 5920)	Renders the image of a custom control.
	Paint (see page 9087)	Renders the image of a custom control.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	ProcessSearchKey (see page 9060)	Adds a character to the protected SearchText (see page 9069) property and updates the selection to the first matching list value.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectKeyValue (see page 9060)	Updates the value of the control so that it reflects a new key field value.

	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateDataFields (see page 9061)	Updates the control when the DataField (see page 9062) property changes.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateListFields (see page 9061)	Updates the control when the list of lookup values changes.
V	UpdateListFields (see page 9089)	Updates the control when the list of lookup values changes.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DataField (see page 9062)	Identifies the field whose value is represented by the lookup control.
R	DataLink (see page 9063)	Helps the lookup control manage its link to the data source and respond to data events.
	DataSource (see page 9063)	Links the lookup control to the dataset that contains the DataField (see page 9062).
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
R	HasFocus (see page 9064)	Indicates whether the lookup control has input focus.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	KeyField (see page 9064)	Identifies the field in the ListSource (see page 9068) dataset that must match the value of the DataField (see page 9062) field.
	KeyValue (see page 9065)	Represents the common value of the KeyField (see page 9064) field and the DataField (see page 9062) field.
R	ListActive (see page 9065)	Indicates whether the ListFields (see page 9067) property is currently being updated.
	ListField (see page 9066)	Identifies the field or fields whose values are displayed in the lookup control.
	ListFieldIndex (see page 9067)	Specifies which field from the ListField (see page 9066) property is used for incremental searching.
R	ListFields (see page 9067)	Lists the field components whose values are displayed in the lookup control.
R	ListLink (see page 9067)	Identifies the TListSourceLink (see page 9498) object that coordinates data communication with the lookup dataset.
	ListSource (see page 9068)	Identifies a data source for the data displayed in the lookup control.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	NullValueKey (see page 9068)	Specifies the key combination that assigns a blank value to the associated field.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 9069)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.

	ReadOnly (see page 9069)	This is ReadOnly, a member of class TDBLookupControl.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	SearchText (see page 9069)	This is SearchText, a member of class TDBLookupControl.
	TabStop (see page 9070)	Determines if the user can tab to a control.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 9534)	Creates an instance of TDBLookupListBox.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 9056)	Destroys an instance of TDBLookupControl (see page 9044).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteAction (see page 9057)	Invokes an action with the TDBLookupControl (see page 9044) as its target.
	ExecuteAction (see page 9083)	Performs an action directed at the DB (see page 6725) lookup list box.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.

	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
V	SetBounds (see page 9088)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateAction (see page 9060)	Updates an action to reflect the current state of the TDBLookupControl (see page 9044).
V	UpdateAction (see page 9088)	Updates an action component to reflect the current state of the DB (see page 6725) lookup list box.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
V	UseRightToLeftAlignment (see page 9089)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Field (see page 9064)	Identifies the TField object the lookup control represents.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	KeyValue (see page 9106)	Represents the common value of the KeyField field and the DataField field.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.

	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	SelectedItem (see page 9116)	Contains the value displayed in the list box that is currently selected.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 9121)	Occurs when the user clicks the control.
	OnContextPopup (see page 9123)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 9124)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 9124)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 9126)	Occurs when the user drags an object over a control.
	OnEndDock (see page 9129)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 9129)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 9130)	Occurs when a control receives the input focus.
	OnExit (see page 9132)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 9134)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 9137)	Occurs when a key is pressed.
	OnKeyUp (see page 9138)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 9140)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 9141)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 9143)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 9143)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 9143)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 9146)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 9148)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 9148)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 9091)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 9094)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 9095)	Specifies which edges of the control are beveled.
	BevelInner (see page 9095)	Specifies the cut of the inner bevel.
	BevelKind (see page 9095)	Specifies the control's bevel style.
	BevelOuter (see page 9096)	Specifies the cut of the outer bevel.

	BevelWidth (see page 9096)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 9097)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 9097)	Determines whether the lookup list box has a border.
	Color (see page 9098)	Specifies the background color of the control.
	Constraints (see page 9098)	Specifies the size constraints for the control.
	Ctl3D (see page 9099)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DataField (see page 9099)	Identifies the field whose value is represented by the lookup control.
	DataSource (see page 9100)	Links the lookup control to the dataset that contains the DataField.
	DragCursor (see page 9100)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 9101)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 9101)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 9103)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 9103)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 9104)	Determines the behavior of the input method editor (IME).
	ImeName (see page 9105)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	KeyField (see page 9105)	Identifies the field in the ListSource dataset that must match the value of the DataField field.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	ListField (see page 9106)	Identifies the field or fields whose values are displayed in the lookup control.
	ListFieldIndex (see page 9107)	Specifies which field from the ListField property is used for incremental searching.
	ListSource (see page 9108)	Identifies a data source for the data displayed in the lookup control.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NullValueKey (see page 9108)	Specifies the key combination that assigns a blank value to the associated field.
	ParentBiDiMode (see page 9109)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 9109)	Determines where a control looks for its color information.
	ParentCtl3D (see page 9109)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 9110)	Determines where a control looks for its font information.
	ParentShowHint (see page 9110)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 9111)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 9115)	This is ReadOnly, a member of class TDBLookupListBox.
	RowCount (see page 9116)	Specifies how many rows are visible in the list box.
	ShowHint (see page 9116)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 9118)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 9118)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 9119)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

47.1.20.2 TPopupDataList.Create Constructor

Creates an instance of TDBLookupListBox.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TPopupDataList(TComponent * AOwner);
```

Description

Call Create to instantiate a TDBLookupListBox object at runtime. For TDBLookupListBox objects placed on forms or data modules at design time, Create is called automatically.

Pass a single Component as a parameter to provide the TDBLookupListBox object with an Owner that is responsible for freeing it.After calling the inherited constructor to allocate and initialize a generic lookup control, Create creates and initializes the helper objects needed to link to a dataset.

See Also

TFieldDataLink (see page 9487)

47.1.20.3 TPopupDataList Methods

The methods of the TPopupDataList class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 9534)	Initializes a window-creation parameter data structure.

47.1.20.3.1 TPopupDataList.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

47.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DBUseRightToLeftAlignment (see page 9536)	Indicates whether a data-aware control's alignment is reversed to a right-to-left mode.
	OkToChangeFieldAlignment (see page 9536)	This is function DBCtrls.OkToChangeFieldAlignment.

47.2.1 DBCtrls.DBUseRightToLeftAlignment Function

Indicates whether a data-aware control's alignment is reversed to a right-to-left mode.

Pascal

```
function DBUseRightToLeftAlignment(AControl: TControl; AField: TField): Boolean;
```

C++

```
Boolean DBUseRightToLeftAlignment(TControl * AControl, TField AField);
```

File

DBCtrls

Description

Call DBUseRightToLeftAlignment to determine whether a control's alignment has been reversed to a right-to-left mode. This information can be used when painting the control or when responding to mouse events.

DBUseRightToLeftAlignment returns true for middle east locales if the BiDiMode property is bdRightToLeft. Otherwise, it returns false.

AControl is the data-aware control to check.

AField is the field associated with the control specified in AControl. The AField parameter is necessary so that DBUseRightToLeftAlignment can be used with controls such as TDBLookupComboBox (see page 8963), which uses the list field and not the target field for determining alignment.

See Also

TControl

TControl

47.2.2 DBCtrls.OkToChangeFieldAlignment Function

Pascal

```
function OkToChangeFieldAlignment(AField: TField; Alignment: TAlignment): Boolean;
```

C++

```
Boolean OkToChangeFieldAlignment(TField AField, TAlignment Alignment);
```

File

DBCtrls

Description

This is function DBCtrls.OkToChangeFieldAlignment.

47.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TDropDownAlign (see page 9538)	TDropDownAlign indicates the relative alignment of a drop-down list to the edit portion of a DB (see page 6725) lookup combo box.
	TNavGlyph (see page 9539)	This is record DBCtrls.TNavGlyph.
	TNavigateBtn (see page 9539)	TNavigateBtn defines values identifying different possible buttons on a TDBNavigator (see page 9230) object.

47.3.1 DBCtrls.TDropDownAlign Enumeration

TDropDownAlign indicates the relative alignment of a drop-down list to the edit portion of a DB (see page 6725) lookup combo box.

Pascal

```
TDropDownAlign = (
  daLeft,
  daRight,
  daCenter
);
```

C++

```
enum TDropDownAlign {
  daLeft,
  daRight,
  daCenter
};
```

File

DBCtrls

Description

The following table summarizes the possible values:

Value	Meaning
daLeft	The drop-down list is aligned with the left edge of the edit region.
daRight	The drop-down list is aligned with the right edge of the edit region.
daCenter	The drop-down list is centered beneath the edit region.

47.3.2 DBCtrls.TNavGlyph Enumeration

Pascal

```
TNavGlyph = (
  ngEnabled,
  ngDisabled
);
```

C++

```
enum TNavGlyph {
  ngEnabled,
  ngDisabled
};
```

File

DBCtrls

Description

This is record DBCtrls.TNavGlyph.

47.3.3 DBCtrls.TNavigateBtn Enumeration

TNavigateBtn defines values identifying different possible buttons on a TDBNavigator (see page 9230) object.

Pascal

```
TNavigateBtn = (
  nbFirst,
  nbPrior,
  nbNext,
  nbLast,
  nbInsert,
  nbDelete,
  nbEdit,
  nbPost,
  nbCancel,
  nbRefresh
);
```

C++

```
enum TNavigateBtn {
  nbFirst,
  nbPrior,
  nbNext,
  nbLast,
  nbInsert,
  nbDelete,
  nbEdit,
  nbPost,
  nbCancel,
  nbRefresh
};
```

File

DBCtrls

Description

Each TNavigateBtn value identifies a different button that can appear on a TDBNavigator (see page 9230) object. The following table lists all the button types, their corresponding TNavigateBtn values, and the action associated with each button:

ButtonValueAction

Button	Value	Action
First	nbFirst	Go to the first record
Prior	nbPrior	Go to the previous record
Next	nbNext	Go to the next record
Last	nbLast	Go to the last record
Insert	nbInsert	Insert a blank record
Delete	nbDelete	Delete the current record
Edit	nbEdit	Permit users to edit the current record
Post	nbPost	Post the current record
Cancel	nbCancel	Cancel the current edit
Refresh	nbRefresh	Refresh the data in the dataset

47.4 Types

The following table lists types in this documentation.

Types

Name	Description
ENavClick (see page 9541)	This is type DBCtrls.ENavClick.
TNavButtonSet (see page 9541)	This is type DBCtrls.TNavButtonSet.
TNavButtonStyle (see page 9541)	This is type DBCtrls.TNavButtonStyle.

47.4.1 DBCtrls.ENavClick Type

Pascal

```
ENavClick = procedure (Sender: TObject; Button: TNavigateBtn) of object;
```

C++

```
(Sender: TObject; Button: TNavigateBtn) ( ENavClick)();
```

File

DBCtrls

Description

This is type DBCtrls.ENavClick.

47.4.2 DBCtrls.TNavButtonSet Type

Pascal

```
TNavButtonSet = set of TNavigateBtn;
```

C++

```
TNavigateBtn TNavButtonSet;
```

File

DBCtrls

Description

This is type DBCtrls.TNavButtonSet.

47.4.3 DBCtrls.TNavButtonStyle Type

Pascal

```
TNavButtonStyle = set of (nsAllowTimer, nsFocusRect);
```

C++

```
(nsAllowTimer, nsFocusRect) TNavButtonStyle;
```

File

DBCtrls

Description

This is type DBCtrls.TNavButtonStyle.

47.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
InitRepeatPause (see page 9543)	This is constant DBCtrls.InitRepeatPause.
RepeatPause (see page 9543)	This is constant DBCtrls.RepeatPause.
SpaceSize (see page 9543)	This is constant DBCtrls.SpaceSize.

47.5.1 DBCtrls.InitRepeatPause Constant

Pascal

```
InitRepeatPause = 400;
```

C++

```
InitRepeatPause = 400;
```

File

DBCtrls

Description

This is constant DBCtrls.InitRepeatPause.

47.5.2 DBCtrls.RepeatPause Constant

Pascal

```
RepeatPause = 100;
```

C++

```
RepeatPause = 100;
```

File

DBCtrls

Description

This is constant DBCtrls.RepeatPause.

47.5.3 DBCtrls.SpaceSize Constant

Pascal

```
SpaceSize = 5;
```

C++

```
SpaceSize = 5;
```

File

DBCtrls

Description

This is constant DBCtrls.SpaceSize.

48 DbExcept Namespace

This is namespace DbExcept.

Topics

Name	Description
Classes (see page 9546)	The following table lists classes in this documentation.
Variables (see page 9572)	The following table lists variables in this documentation.

Classes

	Name	Description
	TDbEngineErrorDlg (see page 9546)	TDbEngineErrorDlg replaces the normal message box for EDBEngineError exceptions raised at design time.

Variables

Name	Description
DbEngineErrorDlg (see page 9572)	This is variable DbExcept.DbEngineErrorDlg.

48.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TDbEngineErrorDlg (see page 9546)	TDbEngineErrorDlg replaces the normal message box for EDBEngineError exceptions raised at design time.

48.1.1 TDbEngineErrorDlg Class

TDbEngineErrorDlg replaces the normal message box for EDBEngineError exceptions raised at design time.

Class Hierarchy

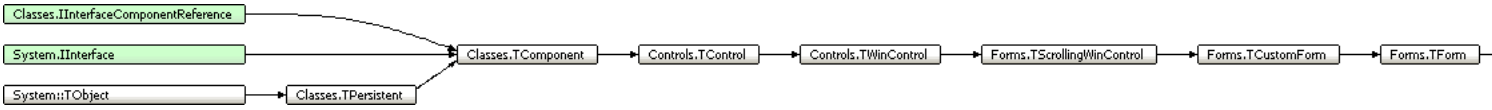

Pascal

```
TDbEngineErrorDlg = class(TForm);
```

C++

```
class TDbEngineErrorDlg : public TForm;
```

File

DbExcept

Description

The standard message box displayed by an unhandled exception shows only the exception's name and description. EDBEngineError exceptions, however, contain additional information that can be useful to application developers. For this reason, exceptions of the EDBEngineError class, when raised at design time, appear in a special dialog box called TDbEngineErrorDlg. The dialog shows the entire stack of BDE errors as represented in the EDBEngineError object.

At runtime, unhandled EDBEngineError exceptions generate the standard message box. But you can use the HookExceptions (see page 9569) or ShowException (see page 9570) method to display TDbEngineErrorDlg instead. This can be useful for debugging or in cases where you want the end user to have access to BDE error information.

See Also

EDBEngineError (see page 9854)

TDBError (see page 10029)

Exception (see page 22750)

48.1.1.1 TDbEngineErrorDlg Members

The following tables list the members exposed by TDbEngineErrorDlg.

Protected Events

	Name	Description
	OnActivate (see page 13943)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnCanResize (see page 13947)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnClick (see page 13947)	Occurs when the user clicks the control.
	OnClose (see page 13949)	Occurs when the form closes.
	OnCloseQuery (see page 13950)	Occurs when close is attempted.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 13951)	Occurs when the form is created.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDblClick (see page 13953)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 13953)	Occurs when the form loses focus.
	OnDestroy (see page 13953)	Occurs when the form is destroyed.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragDrop (see page 13954)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDragOver (see page 13957)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnHelp (see page 13959)	Occurs when the form receives a request for help.
	OnHide (see page 13960)	Occurs when the form is hidden (that is, when its Visible (see page 13940) property is set to false).
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyDown (see page 13962)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyPress (see page 13964)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnKeyUp (see page 13966)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseActivate (see page 13967)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseDown (see page 13968)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseMove (see page 13971)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseUp (see page 13973)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnPaint (see page 13975)	Occurs when the form is redrawn.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnResize (see page 13976)	Occurs immediately after the control is resized.
	OnShortCut (see page 13977)	Occurs when the user presses a key (before the OnKeyDown (see page 13962) event).
	OnShow (see page 13977)	Occurs when the form is shown (that is, when its Visible (see page 13940) property is set to true).
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFormState (see page 13863)	This is FFormState, a member of class TCustomForm.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	Activate (see page 13865)	Activation event dispatcher.
	ActiveChanged (see page 13865)	Active (see page 13896) state event dispatcher.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 14324)	Adjusts the ClientRect property to exclude the scroll bars.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 13866)	Aligns all child controls of the form that fall within a specified rectangle.
V	AlignControls (see page 14325)	Aligns the controls in the scrolling windowed control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.

	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	AutoScrollEnabled (see page 14325)	Indicates whether automatic scrolling is enabled.
	AutoScrollInView (see page 14325)	Scrolls a control into the visible area of the scrolling windowed control.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	BeginAutoDrag (see page 13867)	Responds when the user starts dragging the form if DragMode is dmAutomatic.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	ChangeScale (see page 13867)	Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.
	ChangeScale (see page 14326)	Resizes the scrolling windowed control by a specific ratio.
	Click (see page 5743)	Respond to user click.
	CloseModal (see page 13868)	This is CloseModal, a member of class TCustomForm.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 13868)	Initializes the window-creation parameter record when the form window is created.
	CreateParams (see page 14326)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWindowHandle (see page 13869)	Creates a form window given a set of window-creation parameters.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 13869)	Creates the form window.
	CreateWnd (see page 14327)	Creates a Windows control corresponding to the TScrollingWinControl (see page 14312) object.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	Deactivate (see page 13870)	Generates an OnDeactivate (see page 13953) event.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DefineProperties (see page 13871)	Designates methods for storing a form's unpublished data on a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.

	DestroyHandle (see page 13871)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWindowHandle (see page 13872)	Destroys the window created in the CreateWindowHandle (see page 13869) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoClose (see page 13873)	Form closing event dispatcher.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoCreate (see page 13873)	Form creation event dispatcher.
	DoDestroy (see page 13873)	Form destruction event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDock (see page 13874)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoFlipChildren (see page 14327)	Flips the control's children.
	DoHide (see page 13874)	Form hide event dispatcher.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoShow (see page 13875)	Form show event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorderIconStyles (see page 13875)	This is GetBorderIconStyles, a member of class TCustomForm.
	GetBorderStyles (see page 13876)	This is GetBorderStyles, a member of class TCustomForm.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetChildren (see page 13876)	Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.

	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetClientRect (see page 13876)	Returns the value of the ClientRect property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloating (see page 13877)	Returns the value of the Floating property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetOwnerWindow (see page 13878)	This is GetOwnerWindow, a member of class TCustomForm.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 13878)	Handles exceptions that are raised by the OnCreate (see page 13951) event handler.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	Loaded (see page 13880)	Performs fix-ups when the form is first loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	Notification (see page 13882)	Responds to notifications that objects are about to be inserted or removed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 13882)	Generates an OnPaint (see page 13975) event.

	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
V	PaintWindow (see page 13882)	Renders the image of the form.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
V	PaletteChanged (see page 13883)	Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	QueryInterface (see page 13884)	Returns a reference to a specified interface if that interface is supported by the form.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
V	ReadState (see page 13884)	Reads the form's published properties, stored data, and owned components from a reader object.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
V	RequestAlign (see page 13885)	Instructs the form's parent (if any) to reposition the form, enforcing its Align property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
V	Resizing (see page 13885)	Makes any required adjustments when the form changes size.
V	Resizing (see page 14328)	This is Resizing, a member of class TScrollingWinControl.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
V	SetChildOrder (see page 13886)	Changes the order in which Child appears when GetChildren (see page 13876) is called.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.

V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParent (see page 13887)	Sets the Parent property.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
V	SetParentBiDiMode (see page 13887)	Sets the ParentBiDiMode property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
V	UpdateActions (see page 13891)	Updates all actions associated with the form.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	UpdateWindowState (see page 13892)	Updates the WindowState (see page 13942) property to reflect the current state of the form.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	ValidateRename (see page 13892)	Determines if a form can rename one of its owned components.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	VisibleChanging (see page 13893)	Responds just before the form changes the value of its Visible (see page 13940) property.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WndProc (see page 13893)	Provides specific message responses for the form.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
R	ActiveMDIChild (see page 13897)	Specifies the MDI child that has focus.
	AlphaBlend (see page 13898)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 13898)	Specifies the degree of translucency on a translucent form.
	AutoScroll (see page 13899)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoScroll (see page 14329)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.

	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderIcons (see page 13899)	Specifies which icons appear on the title bar of the form.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
R	ClientHandle (see page 13904)	Provides access to the handle of the form's internal MDI client window.
W	ClientHeight (see page 13905)	Specifies the height (in pixels) of the form's client area.
W	ClientWidth (see page 13905)	Specifies the width (in pixels) of the form's client area.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Ctl3D (see page 13909)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultMonitor (see page 13909)	Specifies the monitor on which the form appears.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	FormStyle (see page 13912)	Determines the form's style.
	Height (see page 13915)	Specifies the vertical size of the control in pixels.
	HorzScrollBar (see page 13916)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 13917)	Specifies the icon that appears when the form is minimized.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	MDIChildCount (see page 13921)	Specifies the number of open MDI child forms.
R	MDIChildren (see page 13922)	Provides indexed access to all MDI child forms.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ObjectMenuItem (see page 13926)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 13926)	Specifies when OnCreate (see page 13951) and OnDestroy (see page 13953) events occur.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentFont (see page 13927)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PixelsPerInch (see page 13928)	Represents the proportion of the font on the system on which the form was designed.
R	PopupChildren (see page 13928)	This is PopupChildren, a member of class TCustomForm.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	PopupMenu (see page 13928)	Identifies the pop-up menu associated with the control.
	Position (see page 13933)	Represents the size and placement of the form.
	PrintScale (see page 13934)	Represents the proportions of a printed form.
	Scaled (see page 13935)	Specifies whether the form is sized according to the value of the PixelsPerInch (see page 13928) property.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.

	TileMode (see page 13936)	Represents how MDI child forms are arranged when the Tile method is called.
	TransparentColor (see page 13938)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 13939)	Indicates the color on the form that appears transparent when TransparentColor (see page 13938) is true.
	VertScrollBar (see page 13939)	Represents the vertical scroll bar for the scrolling windowed control.
W	Visible (see page 13940)	Indicates whether the form is visible.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	Width (see page 13941)	Specifies the horizontal size of the control or form in pixels.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowMenu (see page 13941)	Specifies the Window menu for an MDI parent form.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Fields

	Name	Description
	BackBtn (see page 9564)	This is BackBtn, a member of class TDbEngineErrorDlg.
	BasicPanel (see page 9564)	This is BasicPanel, a member of class TDbEngineErrorDlg.
	BDELabel (see page 9564)	This is BDELabel, a member of class TDbEngineErrorDlg.
	ButtonPanel (see page 9564)	This is ButtonPanel, a member of class TDbEngineErrorDlg.
	DbCatSub (see page 9564)	This is DbCatSub, a member of class TDbEngineErrorDlg.
	DbMessageText (see page 9565)	This is DbMessageText, a member of class TDbEngineErrorDlg.
	DbResult (see page 9565)	This is DbResult, a member of class TDbEngineErrorDlg.
	DetailsBtn (see page 9565)	This is DetailsBtn, a member of class TDbEngineErrorDlg.
	DetailsPanel (see page 9565)	This is DetailsPanel, a member of class TDbEngineErrorDlg.
	ErrorText (see page 9566)	This is ErrorText, a member of class TDbEngineErrorDlg.
	IconImage (see page 9566)	This is IconImage, a member of class TDbEngineErrorDlg.
	IconPanel (see page 9566)	This is IconPanel, a member of class TDbEngineErrorDlg.
	NativeLabel (see page 9566)	This is NativeLabel, a member of class TDbEngineErrorDlg.
	NativeResult (see page 9566)	This is NativeResult, a member of class TDbEngineErrorDlg.
	NextBtn (see page 9567)	This is NextBtn, a member of class TDbEngineErrorDlg.
	OKBtn (see page 9567)	This is OKBtn, a member of class TDbEngineErrorDlg.
	RightPanel (see page 9567)	This is RightPanel, a member of class TDbEngineErrorDlg.
	TopPanel (see page 9567)	This is TopPanel, a member of class TDbEngineErrorDlg.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 13866)	Construction event dispatcher.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	ArrangeIcons (see page 14006)	Arranges the icons of minimized MDI child forms.
V	Assign (see page 2930)	Copies the contents of another, similar object.
	BackClick (see page 9568)	Displays the previous exception.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 13866)	Destruction event dispatcher.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	Cascade (see page 14007)	Arranges MDI child forms so they overlap.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Close (see page 13867)	Closes the form.
V	CloseQuery (see page 13868)	Close (see page 13867) attempt event dispatcher.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 13861)	Creates and initializes a new TForm (see page 13992) object.
V	Create (see page 14323)	Creates and initializes a new TScrollingWinControl (see page 14312) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	CreateNew (see page 13862)	Creates and initializes a new form.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 13870)	Provides message handling for all messages that the form does not fully process.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DefocusControl (see page 13871)	Removes focus from a control on the form.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 13862)	Removes the form from memory.
V	Destroy (see page 14323)	Removes the TScrollingWinControl (see page 14312) object from memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DetailsBtnClick (see page 9568)	Displays or hides the detail context information for an exception.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	DisableAutoRange (see page 14327)	Disables automatic scrolling.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
V	Dock (see page 13872)	Handles automatic docking of the form into other windows.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EnableAutoRange (see page 14328)	Reenables automatic scrolling.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	FocusControl (see page 13875)	Sets focus to a control on the form.
	Focused (see page 6150)	Determines whether the control has input focus.
	FormCreate (see page 9568)	Initializes the dialog when it is first created.
	FormDestroy (see page 9569)	This is FormDestroy, a member of class TDbEngineErrorDlg.
	FormShow (see page 9569)	Displays the text for the current exception and its details.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFormImage (see page 13877)	Returns a bitmap of the form.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	Hide (see page 13879)	Hides the form.
	HookExceptions (see page 9569)	Enables the TDbEngineErrorDlg (see page 9546) instance, replacing the standard exception message at runtime.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	IsShortCut (see page 13880)	Processes shortcut keys when the form has focus.
	MakeFullyVisible (see page 13881)	Ensures that the form is fully visible on a specified monitor.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MouseWheelHandler (see page 13881)	Directs mouse wheel messages to the appropriate control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 14008)	Activates the next child form in the form sequence.
	NextClick (see page 9570)	Displays the next exception.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	Previous (see page 14009)	Activates the previous MDI child form in the form sequence.
	Print (see page 13883)	Prints the form.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	RecreateAsPopup (see page 13884)	This is RecreateAsPopup, a member of class TCustomForm.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	Release (see page 13885)	Destroys the form and frees its associated memory.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollInView (see page 14328)	Scrolls a control into the visible area of the scrolling windowed control.
	SendCancelMode (see page 13886)	Cancels modes on the form.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
V	SetFocus (see page 13886)	Sets focus to the form.
V	SetFocusedControl (see page 13887)	Sets focus to a control on the form.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Show (see page 13888)	Shows the form.
	ShowException (see page 9570)	Displays an EDBEngineError exception in the TDbEngineErrorDlg (see page 9546).
V	ShowModal (see page 13890)	Shows a form as a modal dialog.
	Tile (see page 14009)	Arranges MDI child forms so that they are all the same size.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

	UpdateControlState (see page 6186)	Respond to state change.
	UpdateGlassFrame (see page 13892)	This is UpdateGlassFrame, a member of class TCustomForm.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.
V	WantChildKey (see page 13893)	Indicates whether the form processes keyboard input for a control it owns.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Action (see page 13895)	Designates the action associated with the control.
R	Active (see page 13896)	Specifies whether the form has focus.
	ActiveControl (see page 13896)	Specifies the control that has focus on the form.
	ActiveMDIChild (see page 14013)	Specifies the MDI child that has focus.
	ActiveOleControl (see page 13897)	Specifies the OLE control on the form that reacts to changes in focus.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 13900)	Specifies the appearance and behavior of the form border.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 13900)	Provides access to the drawing area of the form.
	Caption (see page 13902)	Specifies a text string that identifies the control to the user.
	ClientHandle (see page 14024)	Provides access to the handle of the form's internal MDI client window.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 13908)	Specifies the background color of the control.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DbException (see page 9571)	Holds an instance of EDBEngineError.
	Designer (see page 13910)	Specifies the designer interface for the form.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockManager (see page 14029)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.

	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropTarget (see page 13910)	Specifies whether the form is the target of a drag-and-drop operation.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Font (see page 13910)	Controls the attributes of text written on or in the control.
R	FormState (see page 13911)	Indicates transitional state information about the form.
	GlassFrame (see page 13914)	This is GlassFrame, a member of class TCustomForm.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HelpFile (see page 13916)	Specifies the name of the file the form uses to display Help.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	KeyPreview (see page 13918)	Specifies whether the form should receive keyboard events before the active control.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	MDIChildCount (see page 14044)	Specifies the number of open MDI child forms.
	MDIChildren (see page 14044)	Provides indexed access to all MDI child forms.
	Menu (see page 13923)	Specifies the form's main menu.
	ModalResult (see page 13924)	Represents the return value of a form that is used as a modal dialog.
R	Monitor (see page 13926)	Provides access to the monitor on which the form appears.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	OleFormObject (see page 13927)	Specifies the IOleForm (see page 14340) interface for an in-place OLE object contained in the form.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PopupMode (see page 13933)	This is PopupMode, a member of class TCustomForm.
	PopupParent (see page 13933)	Sets an order for stacked forms that users cannot change.
	ScreenSnap (see page 13935)	Specifies whether form snaps to edge of screen.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	SnapBuffer (see page 13936)	Specifies distance for screen snap.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TileMode (see page 14059)	Represents how MDI child forms are arranged when the Tile method is called.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.
	WindowState (see page 13942)	Represents how the form appears on the screen.

Published Events

	Name	Description
	OnActivate (see page 14065)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 14069)	Occurs when an object with custom alignment is aligned.
	OnAlignInsertBefore (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14069)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 14070)	Occurs when an attempt is made to resize the control.
	OnClick (see page 14070)	Occurs when the user clicks the control.
	OnClose (see page 14072)	Occurs when the form closes.
	OnCloseQuery (see page 14073)	Occurs when close is attempted.
	OnConstrainedResize (see page 14074)	Adjust resize constraints.
	OnContextPopup (see page 14075)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 14076)	Occurs when the form is created.
	OnDblClick (see page 14077)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 14078)	Occurs when the form loses focus.
	OnDestroy (see page 14078)	Occurs when the form is destroyed.
	OnDockDrop (see page 14079)	Occurs when another control is docked to the control.
	OnDockOver (see page 14081)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 14083)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 14085)	Occurs when the user drags an object over a control.
	OnEndDock (see page 14088)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnGetSiteInfo (see page 14088)	Returns the control's docking information.
	OnHelp (see page 14090)	Occurs when the form receives a request for help.
	OnHide (see page 14091)	Occurs when the form is hidden (that is, when its Visible property is set to false).
	OnKeyDown (see page 14093)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 14095)	Occurs when a key is pressed.
	OnKeyUp (see page 14096)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 14098)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 14099)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 14101)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 14102)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 14102)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 14104)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 14107)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 14107)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 14107)	Occurs when the mouse wheel is rotated upward.
	OnPaint (see page 14108)	Occurs when the form is redrawn.
	OnResize (see page 14109)	Occurs immediately after the control is resized.
	OnShortCut (see page 14109)	Occurs when the user presses a key (before the OnKeyDown event).
	OnShow (see page 14110)	Occurs when the form is shown (that is, when its Visible property is set to true).
	OnStartDock (see page 14111)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnUnDock (see page 14111)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Action (see page 14011)	Designates the action associated with the control.
	ActiveControl (see page 14012)	Specifies the control that has focus on the form.

	Align (see page 14013)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AlphaBlend (see page 14017)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 14017)	Specifies the degree of translucency on a translucent form.
	Anchors (see page 14017)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 14018)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 14018)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BiDiMode (see page 14019)	Specifies the bi-directional mode for the control.
	BorderIcons (see page 14019)	Specifies which icons appear on the title bar of the form.
	BorderStyle (see page 14020)	Specifies the appearance and behavior of the form border.
	BorderWidth (see page 14021)	Specifies the width of the control's border.
	Caption (see page 14021)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 14024)	Specifies the height (in pixels) of the form's client area.
	ClientWidth (see page 14025)	Specifies the width (in pixels) of the form's client area.
	Color (see page 14027)	Specifies the background color of the control.
	Constraints (see page 14028)	Specifies the size constraints for the control.
	Ctl3D (see page 14028)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DefaultMonitor (see page 14029)	Specifies the monitor on which the form appears.
	DockSite (see page 14031)	Specifies whether the control can be the target of drag-and-dock operations.
	DragKind (see page 14032)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 14033)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 14034)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 14035)	Controls the attributes of text written on or in the control.
	FormStyle (see page 14036)	Determines the form's style.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	Height (see page 14038)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpFile (see page 14039)	Specifies the name of the file the form uses to display Help.
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HorzScrollBar (see page 14040)	Represents the horizontal scroll bar for the scrolling windowed control.
	HorzScrollBar (see page 14329)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 14041)	Specifies the icon that appears when the form is minimized.
	KeyPreview (see page 14041)	Specifies whether the form should receive keyboard events before the active control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Left (see page 13920)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Menu (see page 14045)	Specifies the form's main menu.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectMenuItem (see page 14047)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 14047)	Specifies when OnCreate and OnDestroy events occur.
	Padding (see page 14047)	Specifies the padding of a control.
	ParentBiDiMode (see page 14048)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 14048)	Determines where a control looks for its font information.
	PixelsPerInch (see page 14049)	Represents the proportion of the font on the system on which the form was designed.

	PopupMenu (see page 14049)	Identifies the pop-up menu associated with the control.
	PopupMode (see page 14054)	Controls (see page 5711) how the top-level form behaves with respect to Window's WS_POPUP style.
	PopupParent (see page 14054)	Sets an order for stacked forms that users cannot change.
	Position (see page 14055)	Represents the size and placement of the form.
	PrintScale (see page 14056)	Represents the proportions of a printed form.
	Scaled (see page 14056)	Specifies whether the form is sized according to the value of the PixelsPerInch property.
	ScreenSnap (see page 14056)	Specifies whether form snaps to edge of screen.
	ShowHint (see page 14057)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SnapBuffer (see page 14058)	Specifies distance for screen snap.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Top (see page 13937)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	TransparentColor (see page 14060)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 14060)	Indicates the color on the form that appears transparent when TransparentColor is true.
	UseDockManager (see page 14060)	Specifies whether the docking manager is used in drag-and-dock operations.
	VertScrollBar (see page 14061)	Represents the vertical scroll bar for the scrolling windowed control.
	VertScrollBar (see page 14330)	Represents the vertical scroll bar for the scrolling windowed control.
	Visible (see page 14062)	Indicates whether the form is visible.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	Width (see page 14063)	Specifies the horizontal size of the control or form in pixels.
	WindowMenu (see page 14063)	Specifies the Window menu for an MDI parent form.
	WindowState (see page 14063)	Represents how the form appears on the screen.

48.1.1.2 TDbEngineErrorDlg Fields

The fields of the TDbEngineErrorDlg class are listed here.

Public Fields

	Name	Description
	BackBtn (see page 9564)	This is BackBtn, a member of class TDbEngineErrorDlg.
	BasicPanel (see page 9564)	This is BasicPanel, a member of class TDbEngineErrorDlg.
	BDELabel (see page 9564)	This is BDELabel, a member of class TDbEngineErrorDlg.
	ButtonPanel (see page 9564)	This is ButtonPanel, a member of class TDbEngineErrorDlg.
	DbCatSub (see page 9564)	This is DbCatSub, a member of class TDbEngineErrorDlg.
	DbMessageText (see page 9565)	This is DbMessageText, a member of class TDbEngineErrorDlg.
	DbResult (see page 9565)	This is DbResult, a member of class TDbEngineErrorDlg.
	DetailsBtn (see page 9565)	This is DetailsBtn, a member of class TDbEngineErrorDlg.
	DetailsPanel (see page 9565)	This is DetailsPanel, a member of class TDbEngineErrorDlg.
	ErrorText (see page 9566)	This is ErrorText, a member of class TDbEngineErrorDlg.
	IconImage (see page 9566)	This is IconImage, a member of class TDbEngineErrorDlg.
	IconPanel (see page 9566)	This is IconPanel, a member of class TDbEngineErrorDlg.
	NativeLabel (see page 9566)	This is NativeLabel, a member of class TDbEngineErrorDlg.
	NativeResult (see page 9566)	This is NativeResult, a member of class TDbEngineErrorDlg.
	NextBtn (see page 9567)	This is NextBtn, a member of class TDbEngineErrorDlg.
	OKBtn (see page 9567)	This is OKBtn, a member of class TDbEngineErrorDlg.
	RightPanel (see page 9567)	This is RightPanel, a member of class TDbEngineErrorDlg.
	TopPanel (see page 9567)	This is TopPanel, a member of class TDbEngineErrorDlg.

48.1.1.2.1 TDbEngineErrorDlg.BackBtn Field

Pascal

```
BackBtn: TButton;
```

C++

```
TButton BackBtn;
```

Description

This is BackBtn, a member of class TDbEngineErrorDlg.

48.1.1.2.2 TDbEngineErrorDlg.BasicPanel Field

Pascal

```
BasicPanel: TPanel;
```

C++

```
TPanel BasicPanel;
```

Description

This is BasicPanel, a member of class TDbEngineErrorDlg.

48.1.1.2.3 TDbEngineErrorDlg.BDELabel Field

Pascal

```
BDELabel: TLabel;
```

C++

```
TLabel BDELabel;
```

Description

This is BDELabel, a member of class TDbEngineErrorDlg.

48.1.1.2.4 TDbEngineErrorDlg.ButtonPanel Field

Pascal

```
ButtonPanel: TPanel;
```

C++

```
TPanel ButtonPanel;
```

Description

This is ButtonPanel, a member of class TDbEngineErrorDlg.

48.1.1.2.5 TDbEngineErrorDlg.DbCatSub Field

Pascal

```
DbCatSub: TEdit;
```

C++

```
TEdit DbCatSub;
```

Description

This is DbCatSub, a member of class TDbEngineErrorDlg.

48.1.1.2.6 TDbEngineErrorDlg.DbMessageText Field

Pascal

```
DbMessageText: TMemo;
```

C++

```
TMemo DbMessageText;
```

Description

This is DbMessageText, a member of class TDbEngineErrorDlg.

48.1.1.2.7 TDbEngineErrorDlg.DbResult Field

Pascal

```
DbResult: TEdit;
```

C++

```
TEdit DbResult;
```

Description

This is DbResult, a member of class TDbEngineErrorDlg.

48.1.1.2.8 TDbEngineErrorDlg.DetailsBtn Field

Pascal

```
DetailsBtn: TButton;
```

C++

```
TButton DetailsBtn;
```

Description

This is DetailsBtn, a member of class TDbEngineErrorDlg.

48.1.1.2.9 TDbEngineErrorDlg.DetailsPanel Field

Pascal

```
DetailsPanel: TPanel;
```

C++

```
TPanel DetailsPanel;
```

Description

This is DetailsPanel, a member of class TDbEngineErrorDlg.

48.1.1.2.10 TDbEngineErrorDlg.ErrorText Field

Pascal

```
ErrorText: TLabel;
```

C++

```
TLabel ErrorText;
```

Description

This is ErrorText, a member of class TDbEngineErrorDlg.

48.1.1.2.11 TDbEngineErrorDlg.IconImage Field

Pascal

```
IconImage: TImage;
```

C++

```
TImage IconImage;
```

Description

This is IconImage, a member of class TDbEngineErrorDlg.

48.1.1.2.12 TDbEngineErrorDlg.IconPanel Field

Pascal

```
IconPanel: TPanel;
```

C++

```
TPanel IconPanel;
```

Description

This is IconPanel, a member of class TDbEngineErrorDlg.

48.1.1.2.13 TDbEngineErrorDlg.NativeLabel Field

Pascal

```
NativeLabel: TLabel;
```

C++

```
TLabel NativeLabel;
```

Description

This is NativeLabel, a member of class TDbEngineErrorDlg.

48.1.1.2.14 TDbEngineErrorDlg.NativeResult Field

Pascal

```
NativeResult: TEdit;
```

C++

```
TEdit NativeResult;
```

Description

This is NativeResult, a member of class TDbEngineErrorDlg.

48.1.1.2.15 TDbEngineErrorDlg.NextBtn Field

Pascal

```
NextBtn: TButton;
```

C++

```
TButton NextBtn;
```

Description

This is NextBtn, a member of class TDbEngineErrorDlg.

48.1.1.2.16 TDbEngineErrorDlg.OKBtn Field

Pascal

```
OKBtn: TButton;
```

C++

```
TButton OKBtn;
```

Description

This is OKBtn, a member of class TDbEngineErrorDlg.

48.1.1.2.17 TDbEngineErrorDlg.RightPanel Field

Pascal

```
RightPanel: TPanel;
```

C++

```
TPanel RightPanel;
```

Description

This is RightPanel, a member of class TDbEngineErrorDlg.

48.1.1.2.18 TDbEngineErrorDlg.TopPanel Field

Pascal

```
TopPanel: TPanel;
```

C++

```
TPanel TopPanel;
```

Description

This is TopPanel, a member of class TDbEngineErrorDlg.

48.1.1.3 TDbEngineErrorDlg Methods

The methods of the TDbEngineErrorDlg class are listed here.

Public Methods

	Name	Description
	BackClick (see page 9568)	Displays the previous exception.
	DetailsBtnClick (see page 9568)	Displays or hides the detail context information for an exception.
	FormCreate (see page 9568)	Initializes the dialog when it is first created.
	FormDestroy (see page 9569)	This is FormDestroy, a member of class TDbEngineErrorDlg.
	FormShow (see page 9569)	Displays the text for the current exception and its details.
	HookExceptions (see page 9569)	Enables the TDbEngineErrorDlg (see page 9546) instance, replacing the standard exception message at runtime.
	NextClick (see page 9570)	Displays the next exception.
	ShowException (see page 9570)	Displays an EDBEngineError exception in the TDbEngineErrorDlg (see page 9546).

48.1.1.3.1 TDbEngineErrorDlg.BackClick Method

Displays the previous exception.

Pascal

```
procedure BackClick(Sender: TObject);
```

C++

```
__fastcall BackClick(TObject * Sender);
```

Description

Call BackClick to simulate clicking the Back button in the dialog. BackClick displays information about the previous exception in the stack of exceptions shown in the dialog.

See Also

NextClick (see page 9570)

DetailsBtnClick (see page 9568)

48.1.1.3.2 TDbEngineErrorDlg.DetailsBtnClick Method

Displays or hides the detail context information for an exception.

Pascal

```
procedure DetailsBtnClick(Sender: TObject);
```

C++

```
__fastcall DetailsBtnClick(TObject * Sender);
```

Description

Call DetailsBtnClick to simulate clicking the Details button in the dialog. If the detail context information is showing, DetailsBtnClick hides it. If the detail information is not showing, DetailsBtnClick displays it..

See Also

NextClick (see page 9570)

BackClick (see page 9568)

48.1.1.3.3 TDbEngineErrorDlg.FormCreate Method

Initializes the dialog when it is first created.

Pascal

```
procedure FormCreate(Sender: TObject);
```

C++

```
__fastcall FormCreate(TObject * Sender);
```

Description

Applications should not call FormCreate. It initializes the detail window so that the Details button works correctly.

See Also

formdestroy (see page 9569)

FormShow (see page 9569)

DetailsBtnClick (see page 9568)

48.1.1.3.4 TDbEngineErrorDlg.FormDestroy Method

Pascal

```
procedure FormDestroy(Sender: TObject);
```

C++

```
__fastcall FormDestroy(TObject * Sender);
```

Description

This is FormDestroy, a member of class TDbEngineErrorDlg.

48.1.1.3.5 TDbEngineErrorDlg.FormShow Method

Displays the text for the current exception and its details.

Pascal

```
procedure FormShow(Sender: TObject);
```

C++

```
__fastcall FormShow(TObject * Sender);
```

Description

Applications should not call FormShow. It is called automatically when the dialog becomes visible. FormShow sets the text for the current exception and, if detail information is showing, displays its details.

See Also

FormCreate (see page 9568)

OnException (see page 13728)

DetailsBtnClick (see page 9568)

48.1.1.3.6 TDbEngineErrorDlg.HookExceptions Method

Enables the TDbEngineErrorDlg (see page 9546) instance, replacing the standard exception message at runtime.

Pascal

```
procedure HookExceptions;
```

C++

```
__fastcall HookExceptions();
```

Description

To display all unhandled EDBEngineError exceptions in a TDbEngineErrorDlg (see page 9546) (rather than the standard message box), create a TDbEngineErrorDlg (see page 9546) and call HookExceptions to enable it:

```
TDbEngineErrorDlg.Create(Application).HookExceptions;
TDbEngineErrorDlg *pDBEDlg = new TDbEngineErrorDlg (Application);
pDBEDlg->HookExceptions();
```

If you include this code in an application's project file, in an OnCreate event handler, or the constructor for the project's main form, the TDbEngineErrorDlg (see page 9546) will trap and display unhandled EDBEngineError exceptions at runtime.

See Also

EDbEngineError

ShowException (see page 9570)

48.1.1.3.7 TDbEngineErrorDlg.NextClick Method

Displays the next exception.

Pascal

```
procedure NextClick(Sender: TObject);
```

C++

```
__fastcall NextClick(TObject * Sender);
```

Description

Call NextClick to simulate clicking the Next button in the dialog. NextClick displays information about the next exception in the stack of exceptions shown in the dialog.

See Also

BackClick (see page 9568)

DetailsBtnClick (see page 9568)

48.1.1.3.8 TDbEngineErrorDlg.ShowException Method

Displays an EDBEngineError exception in the TDbEngineErrorDlg (see page 9546).

Pascal

```
function ShowException(Error: EDbEngineError): TModalResult;
```

C++

```
__fastcall TModalResult ShowException(EDbEngineError Error);
```

Description

Call ShowException in an EDBEngineError exception handler to display the exception in a TDbEngineErrorDlg (see page 9546). You can create the TDbEngineErrorDlg (see page 9546) in the exception handler itself, or in the initialization section of your application.

See Also

EDbEngineError

ModalResult (see page 13924)

HookExceptions (see page 9569)

48.1.1.4 TDbEngineErrorDlg Properties

The properties of the TDbEngineErrorDlg class are listed here.

Public Properties

	Name	Description
	DbException (see page 9571)	Holds an instance of EDBEngineError.

48.1.1.4.1 TDbEngineErrorDlg.DbException Property

Holds an instance of EDBEngineError.

Pascal

```
property DbException: EDbEngineError;
```

C++

```
__property EDbEngineError DbException;
```

Description

DBException is the EDBEngineError instance that causes the error dialog to appear.

48.2 Variables

The following table lists variables in this documentation.

Variables

Name	Description
DbEngineErrorDlg (see page 9572)	This is variable DbExcept.DbEngineErrorDlg.

48.2.1 DbExcept.DbEngineErrorDlg Variable

Pascal

```
DbEngineErrorDlg: TDbEngineErrorDlg;
```

C++

```
TDbEngineErrorDlg DbEngineErrorDlg;
```

File

DbExcept

Description

This is variable DbExcept.DbEngineErrorDlg.

49 DBGrids Namespace

This is namespace DBGrids.

Topics

Name	Description
Classes (see page 9575)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 9765)	The following table lists structs, records, enums in this documentation.
Types (see page 9769)	The following table lists types in this documentation.
Constants (see page 9772)	The following table lists constants in this documentation.

Classes

	Name	Description
	TBookmarkList (see page 9575)	TBookmarkList is a collection of bookmarks that identify a set of records in a dataset.
	TColumn (see page 9583)	TColumn represents a column in a data grid (TDBGrid (see page 9668)).
	TColumnTitle (see page 9601)	TColumnTitle represents the title of a data-grid column (TColumn (see page 9583)).
	TCustomDBGrid (see page 9608)	TCustomDBGrid is the base class for grid controls that display the records from a dataset in a tabular format.
	TDBGrid (see page 9668)	TDBGrid displays and manipulates records from a dataset in a tabular grid.
	TDBGridColumns (see page 9746)	TDBGridColumns represents a container for TColumn (see page 9583) objects.
	TGridDataLink (see page 9754)	TGridDataLink helps manage the link to a data source and maintain the field mapping for a TCustomDBGrid (see page 9608) object.

Constants

Name	Description
cm_DeferLayout (see page 9772)	This is constant DBGrids.cm_DeferLayout.
ColumnTitleValues (see page 9772)	This is constant DBGrids.ColumnTitleValues.
IndicatorWidth (see page 9772)	This is constant DBGrids.IndicatorWidth.

Structs, Records, Enums

	Name	Description
	TColumnButtonStyle (see page 9765)	TColumnButtonStyle indicates whether and how the user can select values for a column from a list.
	TColumnValue (see page 9766)	TColumnValue and TColumnValues (see page 9769) indicate which properties on the column of a DB (see page 6725) grid are persistent.

	TDBGridColumnsState (see page 9767)	TDBGridColumnsState indicates whether the columns in a DB (see page 6725) grid dynamically reflect dataset fields.
	TDBGridOption (see page 9767)	TDBGridOption and TDBGridOptions (see page 9770) indicate display and behavioral property of a DB (see page 6725) grid.

Types

Name	Description
TColumnClass (see page 9769)	TColumnClass defines the metaclass for TColumn (see page 9583).
TColumnValues (see page 9769)	This is type DBGrids.TColumnValues.
TDBGridClickEvent (see page 9770)	This is type DBGrids.TDBGridClickEvent.
TDBGridOptions (see page 9770)	This is type DBGrids.TDBGridOptions.
TDrawColumnCellEvent (see page 9770)	This is type DBGrids.TDrawColumnCellEvent.
TDrawDataCellEvent (see page 9771)	This is type DBGrids.TDrawDataCellEvent.

49.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TBookmarkList (see page 9575)	TBookmarkList is a collection of bookmarks that identify a set of records in a dataset.
	TColumn (see page 9583)	TColumn represents a column in a data grid (TDBGrid (see page 9668)).
	TColumnTitle (see page 9601)	TColumnTitle represents the title of a data-grid column (TColumn (see page 9583)).
	TCustomDBGrid (see page 9608)	TCustomDBGrid is the base class for grid controls that display the records from a dataset in a tabular format.
	TDBGrid (see page 9668)	TDBGrid displays and manipulates records from a dataset in a tabular grid.
	TDBGridColumns (see page 9746)	TDBGridColumns represents a container for TColumn (see page 9583) objects.
	TGridDataLink (see page 9754)	TGridDataLink helps manage the link to a data source and maintain the field mapping for a TCustomDBGrid (see page 9608) object.

49.1.1 TBookmarkList Class

TBookmarkList is a collection of bookmarks that identify a set of records in a dataset.

Class Hierarchy

DBGrids.TBookmarkList

Pascal

```
TBookmarkList = class;
```

C++

```
class TBookmarkList;
```

File

DBGrids

Description

Use TBookmarkList to manage a set of bookmarks when working with the records in a dataset. Each bookmark identifies a particular record in the dataset, and can be identified by a string.

See Also

TCustomDBGrid (see page 9608)

49.1.1.1 TBookmarkList Members

The following tables list the members exposed by TBookmarkList.

Protected Methods

	Name	Description
	Compare (see page 9578)	This is Compare, a member of class TBookmarkList.
	CurrentRow (see page 9578)	This is CurrentRow, a member of class TBookmarkList.
	LinkActive (see page 9579)	This is LinkActive, a member of class TBookmarkList.

Public Methods

	Name	Description
	Clear (see page 9577)	Deletes all bookmarks from the list.
	Create (see page 9576)	Creates an instance of TBookmarkList (see page 9575) and associates it with a DBGrid.
	Delete (see page 9578)	Deletes all the records specified by the bookmarks in the list from the dataset of the associated DBGrid.
V	Destroy (see page 9576)	Destroys an instance of TBookmarkList (see page 9575).
	Find (see page 9578)	Indicates whether a specified bookmark string is contained in the bookmark list.
	IndexOf (see page 9579)	Returns the index of a bookmark string in the Items (see page 9581) property array.
	Refresh (see page 9580)	Removes all invalid bookmarks from the Items (see page 9581) property array.

Public Properties

	Name	Description
R	Count (see page 9580)	Indicates the number of bookmarks in the bookmark list.
	CurrentRowSelected (see page 9581)	Indicates whether the Bookmark property of the associated DBGrid's dataset is in the Items (see page 9581) array.
R	Items (see page 9581)	Provides indexed access to the bookmarks for marked records in the dataset.

49.1.1.2 TBookmarkList.Create Constructor

Creates an instance of TBookmarkList (see page 9575) and associates it with a DBGrid.

Pascal

```
constructor Create(AGrid: TCustomDBGrid);
```

C++

```
__fastcall TBookmarkList(TCustomDBGrid AGrid);
```

Description

Applications do not need to call create to instantiate the bookmark list used by DBGrids (see page 9573) to implement their SelectedRows property. DBGrids (see page 9573) call Create from their constructor to create this bookmark list. Component writers that implement additional properties for descendants of TCustomDBGrid (see page 9608) can create additional bookmark list objects to represent other sets of records in the dataset.

See Also

TCustomDBGrid (see page 9608)

49.1.1.3 TBookmarkList.Destroy Destructor

Destroys an instance of TBookmarkList (see page 9575).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBookmarkList();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TBookmarkList (see page 9575) object is not nil, and only then calls Destroy.

Destroy frees the memory used to store the Items (see page 9581) property array.

See Also

Free (see page 22300)

Items (see page 9581)

delete

49.1.1.4 TBookmarkList Methods

The methods of the TBookmarkList class are listed here.

Protected Methods

	Name	Description
	Compare (see page 9578)	This is Compare, a member of class TBookmarkList.
	CurrentRow (see page 9578)	This is CurrentRow, a member of class TBookmarkList.
	LinkActive (see page 9579)	This is LinkActive, a member of class TBookmarkList.

Public Methods

	Name	Description
	Clear (see page 9577)	Deletes all bookmarks from the list.
	Delete (see page 9578)	Deletes all the records specified by the bookmarks in the list from the dataset of the associated DBGrid.
	Find (see page 9578)	Indicates whether a specified bookmark string is contained in the bookmark list.
	IndexOf (see page 9579)	Returns the index of a bookmark string in the Items (see page 9581) property array.
	Refresh (see page 9580)	Removes all invalid bookmarks from the Items (see page 9581) property array.

49.1.1.4.1 TBookmarkList.Clear Method

Deletes all bookmarks from the list.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the Items (see page 9581) array and set the Count (see page 9580) property to 0 when all rows of the associated DBGrid are deselected. Calling Clear invalidates the associated DBGrid so that the selected rows that were referenced by the bookmarks in the bookmark list can be repainted.

Note: Clear removes the bookmarks from the bookmark list to reflect the fact that the corresponding rows are not selected. To delete the rows from the dataset rather than just deselecting them, call the Delete (see page 9578) method instead.

See Also

Items (see page 9581)

Delete (see page 9578)

Refresh (see page 9580)

Count (see page 9580)

49.1.1.4.2 TBookmarkList.Compare Method

Pascal

```
function Compare(const Item1: TBookmarkStr; const Item2: TBookmarkStr): Integer;
```

C++

```
__fastcall int Compare(const TBookmarkStr Item1, const TBookmarkStr Item2);
```

Description

This is Compare, a member of class TBookmarkList.

49.1.1.4.3 TBookmarkList.CurrentRow Method

Pascal

```
function CurrentRow: TBookmarkStr;
```

C++

```
__fastcall TBookmarkStr CurrentRow();
```

Description

This is CurrentRow, a member of class TBookmarkList.

49.1.1.4.4 TBookmarkList.Delete Method

Deletes all the records specified by the bookmarks in the list from the dataset of the associated DBGrid.

Pascal

```
procedure Delete;
```

C++

```
__fastcall Delete();
```

Description

Call Delete to delete the set of records represented by the bookmark list from the dataset of the associated DBGrid. To remove all the bookmarks from the list without changing the dataset, use the Clear (see page 9577) method instead.

See Also

Clear (see page 9577)

Items (see page 9581)

49.1.1.4.5 TBookmarkList.Find Method

Indicates whether a specified bookmark string is contained in the bookmark list.

Pascal

```
function Find(const Item: TBookmarkStr; var Index: Integer): Boolean;
```

C++

```
__fastcall Boolean Find(const TBookmarkStr Item, int Index);
```

Description

Call Find to determine whether the string specified by the Item parameter is one of the bookmark strings in the Items (see page 9581) property array. Find returns true if the bookmark list contains the bookmark string, false otherwise. If Find returns true, the index of the bookmark string is returned as the value of the Index parameter.

Note: The index of a specific bookmark string can also be obtained by using the IndexOf (see page 9579) method. Unlike the IndexOf (see page 9579) method, which returns a value of -1 if the bookmark string is not in the Items (see page 9581) array, Find does not change the value of the Index parameter if the bookmark string can't be found. Use Find when a default value should be changed only if the bookmark is not in the list.

See Also

Items (see page 9581)

IndexOf (see page 9579)

49.1.1.4.6 TBookmarkList.IndexOf Method

Returns the index of a bookmark string in the Items (see page 9581) property array.

Pascal

```
function IndexOf(const Item: TBookmarkStr): Integer;
```

C++

```
__fastcall int IndexOf(const TBookmarkStr Item);
```

Description

Call IndexOf to get the index for a bookmark string in the Items (see page 9581) array. The first item in the array has index 0, the second item has index 1, and so on. If an item is not in the list, IndexOf returns -1.

See Also

Find (see page 9578)

Items (see page 9581)

49.1.1.4.7 TBookmarkList.LinkActive Method

Pascal

```
procedure LinkActive(Value: Boolean);
```

C++

```
__fastcall LinkActive(Boolean Value);
```

Description

This is LinkActive, a member of class TBookmarkList.

49.1.1.4.8 TBookmarkList.Refresh Method

Removes all invalid bookmarks from the Items (see page 9581) property array.

Pascal

```
function Refresh: Boolean;
```

C++

```
__fastcall Boolean Refresh();
```

Description

Bookmarks in the bookmark list are not automatically deleted when records in the dataset are deleted. As a result, some bookmarks in the Items (see page 9581) array may become invalid. Call Refresh to ensure that every bookmark in the bookmark list is valid. Refresh attempts to locate every record in the dataset that corresponds to a bookmark in the Items (see page 9581) property array. All bookmarks that do not correspond to records in the dataset are removed.

If any of the bookmarks in the list are invalid, Refresh returns true, and invalidates the DBGrid so that it will repaint, reflecting the removal of all invalid records. If all bookmarks in the list are still valid, Refresh returns false.

See Also

Items (see page 9581)

49.1.1.5 TBookmarkList Properties

The properties of the TBookmarkList class are listed here.

Public Properties

	Name	Description
R	Count (see page 9580)	Indicates the number of bookmarks in the bookmark list.
	CurrentRowSelected (see page 9581)	Indicates whether the Bookmark property of the associated DBGrid's dataset is in the Items (see page 9581) array.
R	Items (see page 9581)	Provides indexed access to the bookmarks for marked records in the dataset.

49.1.1.5.1 TBookmarkList.Count Property

Indicates the number of bookmarks in the bookmark list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read Count to determine the number of entries in the Items (see page 9581) array. Use Count as an upper limit when iterating over the strings in the Items (see page 9581) array.

Delphi Examples:

```
{
The following example copies the selected rows in a db grid
to a list box.  Set the db grid Options dgRowSelect,
dgAlwaysShowSelection and dgMultiSelect to True.  Make a
multiple selecton using the CNTL key.  This example requires
a TDataSet associated with a TDataSource and a TDBGrid.}
```

```
procedure TForm1.Button2Click(Sender: TObject);
var
  i, j: Integer;
  s: string;
begin
  if DBGrid2.SelectedRows.Count>0 then
    with DBGrid2.DataSource.DataSet do
      for i:=0 to DBGrid2.SelectedRows.Count-1 do
      begin
        GotoBookmark(pointer(DBGrid2.SelectedRows.Items[i]));
        for j := 0 to FieldCount-1 do
        begin
          if (j>0) then s:=s+', ';
          s:=s+Fields[j].AsString;
        end;
        Listbox1.Items.Add(s);
        s:= '';
      end;
end;
```

See Also

Items (see page 9581)

49.1.1.5.2 TBookmarkList.CurrentRowSelected Property

Indicates whether the Bookmark property of the associated DBGrid's dataset is in the Items (see page 9581) array.

Pascal

```
property CurrentRowSelected: Boolean;
```

C++

```
__property Boolean CurrentRowSelected;
```

Description

Read CurrentRowSelected to determine whether the Bookmark property of the associated DBGrid's dataset specifies a bookmark string in the Items (see page 9581) property array. CurrentRowSelected is true when the bookmark string is included, false when it is not. The Bookmark property of the dataset indicates the current record in the dataset.

Set CurrentRowSelected to specify whether the Bookmark string of the associated DBGrid's dataset should be included in the bookmark list. Setting CurrentRowSelected to true adds the Bookmark property of the dataset to the Items (see page 9581) array if it is not already there. Setting CurrentRowSelected to false removes the Bookmark of the dataset from the Items (see page 9581) array if it is there. If the Bookmark property of the dataset is an empty string (no bookmark), setting CurrentRowSelected does nothing.

Note: Trying to read or write CurrentRowSelected when the dataset is not active will raise an EInvalidGridOperation exception.

See Also

Bookmark (see page 7009)

Items (see page 9581)

49.1.1.5.3 TBookmarkList.Items Property

Provides indexed access to the bookmarks for marked records in the dataset.

Pascal

```
property Items [Index: Integer]: TBookmarkStr;
```

C++

```
__property TBookmarkStr Items[int Index];
```

Description

Use Items to obtain a bookmark string from the set maintained by the TBookmarkList (see page 9575) object. Each bookmark string represents a single record in the dataset of the associated DBGrid. Bookmark strings can be compared with other bookmark strings to determine relative position within the database, and used to position the dataset on an arbitrary record. The Index parameter indicates the index of the bookmark, where 0 is the index of the first bookmark, 1 is the index of the second bookmark, and so on.

Use Items with the Count (see page 9580) property to iterate through all of the bookmarks in the list.

Note: Bookmarks in the bookmark list are not automatically deleted when records in the dataset are deleted. As a result, some bookmarks in the Items array may become invalid. Call the Refresh (see page 9580) method to delete invalid bookmarks from the Items array.

Delphi Examples:

```
{
The following example copies the selected rows in a db grid
to a list box.  Set the db grid Options dgRowSelect,
dgAlwaysShowSelection and dgMultiSelect to True.  Make a
multiple selecton using the CNTL key.  This example requires
a TDataSet associated with a TDataSource and a TDBGrid.}
procedure TForm1.Button2Click(Sender: TObject);
var
  i, j: Integer;
  s: string;
begin
  if DBGrid2.SelectedRows.Count>0 then
    with DBGrid2.DataSource.DataSet do
      for i:=0 to DBGrid2.SelectedRows.Count-1 do
      begin
        GotoBookmark(pointer(DBGrid2.SelectedRows.Items[i]));
        for j := 0 to FieldCount-1 do
        begin
          if (j>0) then s:=s+', ';
          s:=s+Fields[j].AsString;
        end;
        Listbox1.Items.Add(s);
        s:= '';
      end;
end;
```

See Also

Count (see page 9580)

Clear (see page 9577)

Find (see page 9578)

Refresh (see page 9580)

operator_sb

49.1.2 TColumn Class

TColumn represents a column in a data grid (TDBGrid (see page 9668)).

Class Hierarchy

Pascal

```
TColumn = class(TCollectionItem);
```

C++

```
class TColumn : public TCollectionItem;
```

File

DBGrids

Description

Each TDBGrid (see page 9668) uses a TDBGridColumns (see page 9746) to maintain a collection of TColumn objects. Each TColumn object represents the visual characteristics and data binding of a column in the grid.

See Also

TCollection (see page 2731)

TCollectionItem (see page 2755)

TDBGridColumns (see page 9746)

TDBGrid (see page 9668)

49.1.2.1 TColumn Members

The following tables list the members exposed by TColumn.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	CreateTitle (see page 9587)	This is CreateTitle, a member of class TColumn.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 9590)	Returns the name of the collection item as it appears in the collection editor.
	GetGrid (see page 9590)	This is GetGrid, a member of class TColumn.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	RefreshDefaultFont (see page 9591)	This is RefreshDefaultFont, a member of class TColumn.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 9591)	Moves the collection item to a specified position in the parent collection.

Protected Properties

	Name	Description
	IsStored (see page 9598)	This is IsStored, a member of class TColumn.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 9586)	Copies the contents of the source column to a new column.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 9585)	Creates and initializes a data-grid column.
	DefaultAlignment (see page 9587)	Returns the default alignment for the column.
	DefaultColor (see page 9587)	Returns the default background color for the column.
	DefaultFont (see page 9588)	Returns the default font for data displayed in the column.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DefaultImeMode (see page 9588)	Describes the default behavior of the Input Method Editor (IME).
	DefaultImeName (see page 9588)	Returns the name of the default Input Method Editor (IME).
	DefaultReadOnly (see page 9589)	Returns the default ReadOnly (see page 9599) setting for the column.
	DefaultWidth (see page 9589)	Returns the default width for the column.
	Depth (see page 9590)	Returns the position of the column in relation to any parent columns. A column with no parent column returns 0.
V	Destroy (see page 9585)	Destroys the column and frees its memory.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	RestoreDefaults (see page 9591)	Restores the column's default settings.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	AssignedValues (see page 9593)	Indicates which TColumn (see page 9583) and TColumnTitle (see page 9601) properties have been set.
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	Expandable (see page 9594)	Specifies whether the column can be expanded to show any child field columns.
	Field (see page 9595)	Indicates the TField instance represented by the column.

R	Grid (see page 9596)	Indicates the grid that contains the column.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
R	ParentColumn (see page 9598)	Refers to the TColumn (see page 9583) object that owns the column.
R	Showing (see page 9599)	Indicates whether the column is showing.

Published Properties

	Name	Description
	Alignment (see page 9592)	Specifies how text is aligned within the column.
	ButtonStyle (see page 9593)	Determines whether and how the user can select values for the column from a list.
	Color (see page 9594)	Specifies the background color for the column.
	DropDownRows (see page 9594)	Specifies the number of lines displayed in the column's drop-down list.
	Expanded (see page 9595)	Specifies whether the column is expanded.
	FieldName (see page 9595)	Indicates the name of the field represented by the column.
	Font (see page 9596)	Controls (see page 5711) the font in which the column displays its data.
	ImeMode (see page 9597)	Determines the behavior of the input method editor (IME) when the column is edited.
	ImeName (see page 9597)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	PickList (see page 9598)	Lists values that the user can select for the column.
	PopupMenu (see page 9599)	Identifies the pop-up menu associated with the column.
	ReadOnly (see page 9599)	Determines whether data displayed in the column is editable.
	Title (see page 9600)	Indicates the TColumn (see page 9583) Title that represents the column's title.
	Visible (see page 9600)	Specifies whether the column is visible in the grid.
	Width (see page 9600)	The width of the column.

49.1.2.2 TColumn.Create Constructor

Creates and initializes a data-grid column.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TColumn(TCollection Collection);
```

Description

Call Create to instantiate a TColumn (see page 9583) object. Create should take a TDBGridColumns (see page 9746) instance as its argument. It creates a data-grid column along with its title (TColumnTitle (see page 9601)) and font (TFont).

See Also

Title (see page 9600)

Font (see page 9596)

Create (see page 9603)

Create (see page 14499)

TDBGridColumns (see page 9746)

49.1.2.3 TColumn.Destroy Destructor

Destroys the column and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TColumn();
```

Description

Destroy eliminates the TColumn (see page 9583) instance along with its title (TColumnTitle (see page 9601)), font (TFont), and pick list (TStrings).

See Also

Title (see page 9600)

Font (see page 9596)

PickList (see page 9598)

Free (see page 22300)

49.1.2.4 TColumn Methods

The methods of the TColumn class are listed here.

Protected Methods

	Name	Description
	CreateTitle (see page 9587)	This is CreateTitle, a member of class TColumn.
	GetDisplayName (see page 9590)	Returns the name of the collection item as it appears in the collection editor.
	GetGrid (see page 9590)	This is GetGrid, a member of class TColumn.
	RefreshDefaultFont (see page 9591)	This is RefreshDefaultFont, a member of class TColumn.
	SetIndex (see page 9591)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	Assign (see page 9586)	Copies the contents of the source column to a new column.
	DefaultAlignment (see page 9587)	Returns the default alignment for the column.
	DefaultColor (see page 9587)	Returns the default background color for the column.
	DefaultFont (see page 9588)	Returns the default font for data displayed in the column.
	DefaultImeMode (see page 9588)	Describes the default behavior of the Input Method Editor (IME).
	DefaultImeName (see page 9588)	Returns the name of the default Input Method Editor (IME).
	DefaultReadOnly (see page 9589)	Returns the default ReadOnly (see page 9599) setting for the column.
	DefaultWidth (see page 9589)	Returns the default width for the column.
	Depth (see page 9590)	Returns the position of the column in relation to any parent columns. A column with no parent column returns 0.
	RestoreDefaults (see page 9591)	Restores the column's default settings.

49.1.2.4.1 TColumn.Assign Method

Copies the contents of the source column to a new column.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Assign creates a new column with the same FieldName (see page 9595), Title (see page 9600), DropDownRows (see page 9594), ButtonStyle (see page 9593), and PickList (see page 9598) values as the source object. Color (see page 9594), Width (see page 9600), Font (see page 9596), Alignment (see page 9592), and ReadOnly (see page 9599) are also copied if AssignedValues (see page 9593) indicates that they have been changed.

See Also

Assign (see page 2930)

AssignedValues (see page 9593)

49.1.2.4.2 TColumn.CreateTitle Method

Pascal

```
function CreateTitle: TColumnTitle; virtual;
```

C++

```
virtual __fastcall TColumnTitle CreateTitle();
```

Description

This is CreateTitle, a member of class TColumn.

49.1.2.4.3 TColumn.DefaultAlignment Method

Returns the default alignment for the column.

Pascal

```
function DefaultAlignment: TAlignment;
```

C++

```
__fastcall TAlignment DefaultAlignment();
```

Description

Use DefaultAlignment to determine the alignment of the data in the column when the Alignment (see page 9592) property has not been explicitly set. DefaultAlignment returns the Alignment (see page 9592) property of the TField associated with the column. If the column has no dataset field associated with it, DefaultAlignment returns taLeftJustify.

See Also

Alignment (see page 9592)

Field (see page 9595)

Alignment (see page 7146)

49.1.2.4.4 TColumn.DefaultColor Method

Returns the default background color for the column.

Pascal

```
function DefaultColor: TColor;
```

C++

```
__fastcall TColor DefaultColor();
```

Description

Use Default color to determine the color of the column when the Color (see page 9594) property is not explicitly set. DefaultColor returns the Color (see page 9594) of the data grid to which the column belongs. If the column has no associated data grid, DefaultColor returns clWindow.

See Also

Color (see page 9594)

Color (see page 5812)

49.1.2.4.5 TColumn.DefaultFont Method

Returns the default font for data displayed in the column.

Pascal

```
function DefaultFont: TFont;
```

C++

```
__fastcall TFont DefaultFont();
```

Description

DefaultFont returns the Font (see page 9596) of the data grid to which the column belongs. If the column has no associated data grid, DefaultFont returns the current value of the column's Font (see page 9596) property.

See Also

Font (see page 9596)

Font (see page 5823)

49.1.2.4.6 TColumn.DefaultImeMode Method

Describes the default behavior of the Input Method Editor (IME).

Pascal

```
function DefaultImeMode: TImeMode;
```

C++

```
__fastcall TImeMode DefaultImeMode();
```

Description

DefaultImeMode returns the ImeMode (see page 9597) property of the data grid to which the column belongs. If the column has no associated data grid, DefaultImeMode returns the current value of the column's ImeMode (see page 9597) property.

See Also

ImeMode (see page 9597)

ImeMode (see page 6207)

49.1.2.4.7 TColumn.DefaultImeName Method

Returns the name of the default Input Method Editor (IME).

Pascal

```
function DefaultImeName: TImeName;
```

C++

```
__fastcall TImeName DefaultImeName();
```

Description

DefaultImeName returns the ImeName (see page 9597) property of the data grid to which the column belongs. If the column has no associated data grid, DefaultImeName returns the current value of the column's ImeName (see page 9597) property.

See Also

ImeName (see page 9597)

ImeName (see page 6208)

49.1.2.4.8 TColumn.DefaultReadOnly Method

Returns the default ReadOnly (see page 9599) setting for the column.

Pascal

```
function DefaultReadOnly: Boolean;
```

C++

```
__fastcall Boolean DefaultReadOnly();
```

Description

Reflects the ReadOnly (see page 9599) state of the grid and the grid's dataset and field.

See Also

ReadOnly (see page 9599)

49.1.2.4.9 TColumn.DefaultWidth Method

Returns the default width for the column.

Pascal

```
function DefaultWidth: Integer;
```

C++

```
__fastcall int DefaultWidth();
```

Description

The value of DefaultWidth is determined as follows:

If the column has no data grid associated with it, DefaultWidth returns 64.

If the column has an associated data grid but the Field (see page 9595) property is empty, DefaultWidth returns data grid's protected DefaultColWidth property. Usually, this value is 64.

If the column has an associated data field, DefaultWidth is calculated from the TField object's DisplayWidth property using the column's font.

If the dgTitles flag is set in the data grid's Options property, DefaultWidth is increased (as necessary) to accommodate the column's title caption.

See Also

Width (see page 9600)

Field (see page 9595)

TField (see page 7115)

DisplayWidth (see page 7162)

TDBGrid (see page 9668)

Options (see page 9659)

Title (see page 9600)

Caption (see page 9607)

49.1.2.4.10 TColumn.Depth Method

Returns the position of the column in relation to any parent columns. A column with no parent column returns 0.

Pascal

```
function Depth: Integer;
```

C++

```
__fastcall int Depth();
```

Description

If a column does not have any parent columns, Depth returns 0. Columns with parent columns return a number equal to the number of parents it has. For example, a parentless ADT field called Date has a Depth of 0. Its three child fields, Month, Day, and Year, each have a Depth of 1.

See Also

ParentColumn (see page 9598)

49.1.2.4.11 TColumn.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName (see page 2761)

GetNamePath (see page 2759)

49.1.2.4.12 TColumn.GetGrid Method

Pascal

```
function GetGrid: TCustomDBGrid;
```

C++

```
__fastcall TCustomDBGrid GetGrid();
```

Description

This is GetGrid, a member of class TColumn.

49.1.2.4.13 TColumn.RefreshDefaultFont Method

Pascal

```
procedure RefreshDefaultFont;
```

C++

```
__fastcall RefreshDefaultFont();
```

Description

This is RefreshDefaultFont, a member of class TColumn.

49.1.2.4.14 TColumn.RestoreDefaults Method

Restores the column's default settings.

Pascal

```
procedure RestoreDefaults; virtual;
```

C++

```
virtual __fastcall RestoreDefaults();
```

Description

RestoreDefaults reinitializes the AssignedValues (see page 9593), ButtonStyle (see page 9593), Font (see page 9596), and PickList (see page 9598) properties, freeing the pick list (TStrings) object if there is one. It also calls the RestoreDefaults method in the column's title (TColumnTitle (see page 9601)) to restore the default properties of the column title.

See Also

AssignedValues (see page 9593)

ButtonStyle (see page 9593)

Font (see page 9596)

PickList (see page 9598)

RestoreDefaults (see page 9606)

Restoring Default Values to a Column

49.1.2.4.15 TColumn.SetIndex Method

Moves the collection item to a specified position in the parent collection.

Pascal

```
procedure SetIndex(Value: Integer); override;
```

C++

```
virtual __fastcall SetIndex(int Value);
```

Description

SetIndex is the protected write implementation of the Index property. The Value parameter specifies the new (0-based) position

for the item in its parent collection.

See Also

Index (see page 2762)

49.1.2.5 TColumn Properties

The properties of the TColumn class are listed here.

Protected Properties

	Name	Description
	IsStored (see page 9598)	This is IsStored, a member of class TColumn.

Public Properties

	Name	Description
R	AssignedValues (see page 9593)	Indicates which TColumn (see page 9583) and TColumnTitle (see page 9601) properties have been set.
R	Expandable (see page 9594)	Specifies whether the column can be expanded to show any child field columns.
	Field (see page 9595)	Indicates the TField instance represented by the column.
R	Grid (see page 9596)	Indicates the grid that contains the column.
R	ParentColumn (see page 9598)	Refers to the TColumn (see page 9583) object that owns the column.
R	Showing (see page 9599)	Indicates whether the column is showing.

Published Properties

	Name	Description
	Alignment (see page 9592)	Specifies how text is aligned within the column.
	ButtonStyle (see page 9593)	Determines whether and how the user can select values for the column from a list.
	Color (see page 9594)	Specifies the background color for the column.
	DropDownRows (see page 9594)	Specifies the number of lines displayed in the column's drop-down list.
	Expanded (see page 9595)	Specifies whether the column is expanded.
	FieldName (see page 9595)	Indicates the name of the field represented by the column.
	Font (see page 9596)	Controls (see page 5711) the font in which the column displays its data.
	ImeMode (see page 9597)	Determines the behavior of the input method editor (IME) when the column is edited.
	ImeName (see page 9597)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	PickList (see page 9598)	Lists values that the user can select for the column.
	PopupMenu (see page 9599)	Identifies the pop-up menu associated with the column.
	ReadOnly (see page 9599)	Determines whether data displayed in the column is editable.
	Title (see page 9600)	Indicates the TColumn (see page 9583) Title that represents the column's title.
	Visible (see page 9600)	Specifies whether the column is visible in the grid.
	Width (see page 9600)	The width of the column.

49.1.2.5.1 TColumn.Alignment Property

Specifies how text is aligned within the column.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to specify whether values should be left-justified, right-justified, or centered in the column.

See Also

State (see page 9753)

Alignment (see page 9606)

49.1.2.5.2 TColumn.AssignedValues Property

Indicates which TColumn (see page 9583) and TColumnTitle (see page 9601) properties have been set.

Pascal

```
property AssignedValues: TColumnValues;
```

C++

```
__property TColumnValues AssignedValues;
```

Description

For example, if the AssignedValues property contains cvColor, and cvTitleCaption, then the column's font and title caption have been changed and no longer track with changes made to their default source.

Many column properties obtain their values from external sources, such as properties of the associated TField objects. The column will reflect changes made to the property's default source until a value is assigned to the column property. After assignment, the column property is independent of its default source.

See Also

Title (see page 9600)

TColumnTitle (see page 9601)

RestoreDefaults (see page 9591)

Restoring Default Values to a Column

49.1.2.5.3 TColumn.ButtonStyle Property

Determines whether and how the user can select values for the column from a list.

Pascal

```
property ButtonStyle: TColumnButtonStyle;
```

C++

```
__property TColumnButtonStyle ButtonStyle;
```

Description

The ButtonStyle property determines how users can edit the data of a data-grid column.

See Also

PickList (see page 9598)

DropDownRows (see page 9594)

OnEditButtonClick (see page 9668)

Defining a Lookup List Column

49.1.2.5.4 TColumn.Color Property

Specifies the background color for the column.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

The Color property determines the background color of the data-grid column. You can set Color to one of the constants defined in the Graphics (see page 14387) unit (such as clBlue), or to an explicit RGB integer value.

49.1.2.5.5 TColumn.DropDownRows Property

Specifies the number of lines displayed in the column's drop-down list.

Pascal

```
property DropDownRows: Cardinal;
```

C++

```
__property unsigned DropDownRows;
```

Description

DropDownRows determines the number of lines of text displayed in the drop-down list associated with the column. This property is used only if ButtonStyle (see page 9593) is set to cbsAuto and the column has a lookup field or pick list associated with it.

See Also

ButtonStyle (see page 9593)

PickList (see page 9598)

TDBLookupComboBox (see page 8963)

Defining a Lookup List Column

49.1.2.5.6 TColumn.Expandable Property

Specifies whether the column can be expanded to show any child field columns.

Pascal

```
property Expandable: Boolean;
```

C++

```
__property Boolean Expandable;
```

Description

Expandable will be false if the represented field is not an ADT, array, or reference field.

See Also

Depth (see page 9590)

Expanded (see page 9595)

ObjectView (see page 7031)

49.1.2.5.7 TColumn.Expanded Property

Specifies whether the column is expanded.

Pascal

```
property Expanded: Boolean;
```

C++

```
__property Boolean Expanded;
```

Description

Expanded only applies to columns that represent fields that are TObjectField descendants. When a column is expanded, a column appears for each child field of the object field. These child field columns each have a column title and appear under the column title of the parent object field. When Expanded is false, child field values are summarized in a comma-delimited string in the object field's column, and cannot be edited.

Object fields can also be expanded and contracted at runtime through the UI by clicking on the triangle that appears next to its name in the column title.

See Also

Expandable (see page 9594)

ObjectView (see page 7031)

49.1.2.5.8 TColumn.Field Property

Indicates the TField instance represented by the column.

Pascal

```
property Field: TField;
```

C++

```
__property TField Field;
```

Description

Field points to the TField object that corresponds to the dataset field displayed in the column. The value of Field will be nil (Delphi) or NULL (C++), if the column does not correspond to a field in the dataset (for example, a calculated field).

See Also

FieldName (see page 9595)

Fields (see page 9657)

Creating a Customized Grid

49.1.2.5.9 TColumn.FieldName Property

Indicates the name of the field represented by the column.

Pascal

```
property FieldName: WideString;
```

C++

```
__property BSTR FieldName;
```

Description

Setting FieldName changes the Field (see page 9595) property so that it points to the dataset field with the same name. If the dataset does not have a field with the same name, Field (see page 9595) is set to nil (Delphi), or NULL (C++).

See Also

Field (see page 9595)

Title (see page 9600)

Caption (see page 9607)

Creating a Customized Grid

Creating Persistent Columns

Defining a Lookup List Column

49.1.2.5.10 TColumn.Font Property

Controls (see page 5711) the font in which the column displays its data.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

The Font property points to a TFont object that determines typographic attributes of text displayed in the column.

See Also

Font (see page 5823)

Font (see page 9608)

Using a Grid Control in Its Default State

49.1.2.5.11 TColumn.Grid Property

Indicates the grid that contains the column.

Pascal

```
property Grid: TCustomDBGrid;
```

C++

```
__property TCustomDBGrid Grid;
```

Description

Use the Grid property to access the grid that contains the column.

See Also

TDBGrid (see page 9668)

49.1.2.5.12 TColumn.ImeMode Property

Determines the behavior of the input method editor (IME) when the column is edited.

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes when the user is editing the column. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows the type of conversion performed by the IME to be tailored to the input expected for the column. For example, a column that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

See Also

DefaultImeMode (see page 9588)

ImeName (see page 9597)

49.1.2.5.13 TColumn.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes when the user is editing the column. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

See Also

DefaultImeName (see page 9588)

ImeMode (see page 9597)

DefaultIme (see page 14216)

Imes (see page 14227)

49.1.2.5.14 TColumn.IsStored Property

Pascal

```
property IsStored: Boolean;
```

C++

```
__property Boolean IsStored;
```

Description

This is IsStored, a member of class TColumn.

49.1.2.5.15 TColumn.ParentColumn Property

Refers to the TColumn (see page 9583) object that owns the column.

Pascal

```
property ParentColumn: TColumn;
```

C++

```
__property TColumn ParentColumn;
```

Description

If the column represents a child field, ParentColumn refers to the column of the object field that parents the child field.

See Also

Expanded (see page 9595)

49.1.2.5.16 TColumn.PickList Property

Lists values that the user can select for the column.

Pascal

```
property PickList: TStrings;
```

C++

```
__property TStrings PickList;
```

Description

The PickList property points to a TStrings object. If ButtonStyle (see page 9593) is cbsAuto, these strings appear in the drop-down list associated with the column.

See Also

TStrings (see page 3029)

ButtonStyle (see page 9593)

DropDownRows (see page 9594)

PopupMenu (see page 9599)

Defining a Lookup List Column

49.1.2.5.17 TColumn.PopupMenu Property

Identifies the pop-up menu associated with the column.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign (see page 9586) a value to PopupMenu to make a pop-up menu appear when the user clicks the right mouse button on the column. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method.

See Also

PickList (see page 9598)

AutoPopup (see page 16178)

49.1.2.5.18 TColumn.ReadOnly Property

Determines whether data displayed in the column is editable.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Data displayed in the column cannot be changed by the user at runtime if ReadOnly is set to true. If ReadOnly is set to false (and the data grid's Options property includes the dgEditing flag), users can edit data directly from the grid.

See Also

Options (see page 9659)

49.1.2.5.19 TColumn.Showing Property

Indicates whether the column is showing.

Pascal

```
property Showing: Boolean;
```

C++

```
__property Boolean Showing;
```

Description

Showing will return true if Visible (see page 9600) is true and Expanded (see page 9595) is false, and if the field represented is the child of an object field, if the parent column is expanded.

See Also

Visible (see page 9600)

Expanded (see page 9595)

49.1.2.5.20 TColumn.Title Property

Indicates the TColumn (see page 9583) Title that represents the column's title.

Pascal

```
property Title: TColumnTitle;
```

C++

```
__property TColumnTitle Title;
```

Description

The Title property points to a TColumn (see page 9583) Title object that determines attributes of the column's title. If FieldName (see page 9595) is set, the value of FieldName (see page 9595) becomes the default column title (the Caption property of the TColumn (see page 9583) Title object).

The column title appears at runtime only if the dgTitles flag is set in the data grid's Options property.

See Also

TColumnTitle (see page 9601)

Caption (see page 9607)

FieldName (see page 9595)

Options (see page 9659)

Creating Persistent Columns

49.1.2.5.21 TColumn.Visible Property

Specifies whether the column is visible in the grid.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

To hide a column in the grid, set Visible to false. If Visible is true, and the field's Visible property is true, the column will appear in the grid. This property only determines whether the column is allowed to be drawn/shown. The Showing (see page 9599) property reflects the actual display ability of the column based on additional factors.

See Also

Showing (see page 9599)

Expanded (see page 9595)

49.1.2.5.22 TColumn.Width Property

The width of the column.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

The Width property determines the width of the column, in pixels. If the dgColumnResize flag is set in the data grid's Options property, users can resize the column at runtime.

See Also

Options (see page 9659)

Using a Grid Control in Its Default State

49.1.3 TColumnTitle Class

TColumnTitle represents the title of a data-grid column (TColumn (see page 9583)).

Class Hierarchy

Pascal

```
TColumnTitle = class(TPersistent);
```

C++

```
class TColumnTitle : public TPersistent;
```

File

DBGrids

Description

TDBGrid (see page 9668) uses a TDBGridColumns (see page 9746) to maintain a collection of TColumn (see page 9583) objects. Each TColumn (see page 9583) has an associated TColumnTitle that holds information about its title. The TColumnTitle instance is stored in the column's Title property.

See Also

TDBGrid (see page 9668)

TDBGridColumns (see page 9746)

TColumn (see page 9583)

TColumn (see page 9583)

TColumn (see page 9583)

49.1.3.1 TColumnTitle Members

The following tables list the members exposed by TColumnTitle.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.
	RefreshDefaultFont (see page 9606)	This is RefreshDefaultFont, a member of class TColumnTitle.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 9604)	Copies the contents of the source column title to a new column title.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 9603)	Creates and initializes a column title.
	DefaultAlignment (see page 9604)	Returns the default alignment of the column title.
	DefaultCaption (see page 9604)	Returns the default caption of the column title.
	DefaultColor (see page 9605)	Returns the default background color for the column title.
	DefaultFont (see page 9605)	Returns the default font for the column title.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 9603)	Destroys the column and frees its memory.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RestoreDefaults (see page 9606)	Restores the column title's default settings.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Column (see page 9607)	Specifies the TColumn (see page 9583) object that is associated with the column title.

Published Properties

	Name	Description
	Alignment (see page 9606)	Specifies how text is aligned within the column title.
	Caption (see page 9607)	Specifies the text that appears at the top of the column.
	Color (see page 9607)	Specifies the background color for the column title.

	Font (see page 9608)	Controls (see page 5711) the font in which the column title displays its caption.

49.1.3.2 TColumnTitle.Create Constructor

Creates and initializes a column title.

Pascal

```
constructor Create(Column: TColumn);
```

C++

```
__fastcall TColumnTitle(TColumn Column);
```

Description

Call Create to instantiate an instance of TColumnTitle (see page 9601). Create takes a TColumn (see page 9583) instance as its argument.

Most applications need not create column title instances, as these are instantiated by the associated column.

See Also

TColumn (see page 9583)

49.1.3.3 TColumnTitle.Destroy Destructor

Destroys the column and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TColumnTitle();
```

Description

Destroy eliminates the TColumnTitle (see page 9601) instance along with its associated font (TFont).

See Also

Font (see page 9608)

TFont (see page 14497)

Free (see page 22300)

49.1.3.4 TColumnTitle Methods

The methods of the TColumnTitle class are listed here.

Protected Methods

	Name	Description
	RefreshDefaultFont (see page 9606)	This is RefreshDefaultFont, a member of class TColumnTitle.

Public Methods

	Name	Description
	Assign (see page 9604)	Copies the contents of the source column title to a new column title.
	DefaultAlignment (see page 9604)	Returns the default alignment of the column title.

	DefaultCaption (see page 9604)	Returns the default caption of the column title.
	DefaultColor (see page 9605)	Returns the default background color for the column title.
	DefaultFont (see page 9605)	Returns the default font for the column title.
V	RestoreDefaults (see page 9606)	Restores the column title's default settings.

49.1.3.4.1 TColumnTitle.Assign Method

Copies the contents of the source column title to a new column title.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Assign copies the Alignment (see page 9606), Color (see page 9607), Caption (see page 9607), and Font (see page 9608) values of the source object if it is another column title. Otherwise, it calls the inherited method, which allows properties to be copied from any object that implements its AssignTo method with a target of TColumn (see page 9583).

See Also

Assign (see page 2930)

49.1.3.4.2 TColumnTitle.DefaultAlignment Method

Returns the default alignment of the column title.

Pascal

```
function DefaultAlignment: TAlignment;
```

C++

```
__fastcall TAlignment DefaultAlignment();
```

Description

DefaultAlignment returns the alignment for the column that should be used if the Alignment (see page 9606) property is not explicitly set. The DefaultAlignment property for TColumnTitle (see page 9601) always returns taLeftJustify. Descendants of TColumnTitle (see page 9601) can override this to compute the default alignment in an appropriate way.

See Also

Alignment (see page 9606)

49.1.3.4.3 TColumnTitle.DefaultCaption Method

Returns the default caption of the column title.

Pascal

```
function DefaultCaption: string;
```

C++

```
__fastcall AnsiString DefaultCaption();
```

Description

DefaultCaption returns the DisplayLabel of the TField object associated with the column. If there is no dataset field associated with the column (that is, if the column's Field property is empty), it returns the column's FieldName.

See Also

Caption (see page 9607)

DisplayName (see page 7160)

Field (see page 9595)

FieldName (see page 9595)

49.1.3.4.4 TColumnTitle.DefaultColor Method

Returns the default background color for the column title.

Pascal

```
function DefaultColor: TColor;
```

C++

```
__fastcall TColor DefaultColor();
```

Description

Use DefaultColor to determine the background color of the column title if the Color (see page 9607) property is not explicitly set. DefaultColor returns the FixedColor of the data grid to which the column belongs. If the column has no associated data grid, DefaultColor returns clBtnFace.

See Also

Color (see page 9607)

FixedColor (see page 14848)

49.1.3.4.5 TColumnTitle.DefaultFont Method

Returns the default font for the column title.

Pascal

```
function DefaultFont: TFont;
```

C++

```
__fastcall TFont DefaultFont();
```

Description

Use DefaultColor (see page 9605) to determine the font of the column title if the Font (see page 9608) property is not explicitly set. DefaultFont returns the TitleFont of the data grid to which the column belongs. If the column has no associated data grid, DefaultFont returns the current value of the associated column's Font (see page 9608) property.

See Also

Font (see page 9608)

TitleFont (see page 9663)

Font (see page 9596)

49.1.3.4.6 TColumnTitle.RefreshDefaultFont Method

Pascal

```
procedure RefreshDefaultFont;
```

C++

```
__fastcall RefreshDefaultFont();
```

Description

This is RefreshDefaultFont, a member of class TColumnTitle.

49.1.3.4.7 TColumnTitle.RestoreDefaults Method

Restores the column title's default settings.

Pascal

```
procedure RestoreDefaults; virtual;
```

C++

```
virtual __fastcall RestoreDefaults();
```

Description

RestoreDefaults reinitializes the Caption (see page 9607) and Font (see page 9608) properties.

See Also

AssignedValues (see page 9593)

RestoreDefaults (see page 9591)

49.1.3.5 TColumnTitle Properties

The properties of the TColumnTitle class are listed here.

Public Properties

	Name	Description
R	Column (see page 9607)	Specifies the TColumn (see page 9583) object that is associated with the column title.

Published Properties

	Name	Description
	Alignment (see page 9606)	Specifies how text is aligned within the column title.
	Caption (see page 9607)	Specifies the text that appears at the top of the column.
	Color (see page 9607)	Specifies the background color for the column title.
	Font (see page 9608)	Controls (see page 5711) the font in which the column title displays its caption.

49.1.3.5.1 TColumnTitle.Alignment Property

Specifies how text is aligned within the column title.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to specify whether the column title is left-justified, right-justified, or centered.

See Also

Caption (see page 9607)

49.1.3.5.2 TColumnTitle.Caption Property

Specifies the text that appears at the top of the column.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

The Caption property contains a text string that identifies the column. If the dgTitles flag is set in the data grid's Options property, the title appears on the form at runtime.

If the FieldName property is set in TColumn (see page 9583), the Caption default value comes from the DisplayLabel property of the Field object, which itself defaults to FieldName.

See Also

Options (see page 9659)

Title (see page 9600)

FieldName (see page 9595)

49.1.3.5.3 TColumnTitle.Color Property

Specifies the background color for the column title.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

The Color property determines the background color of the column title. You can set Color to one of the constants defined in the Graphics (see page 14387) unit (such as clBlue), or to an explicit RGB integer value. The default value is the grid's FixedColor property.

49.1.3.5.4 TColumnTitle.Column Property

Specifies the TColumn (see page 9583) object that is associated with the column title.

Pascal

```
property Column: TColumn;
```

C++

```
__property TColumn Column;
```

Description

Use Column to determine the column associated with the column title.

See Also

Title (see page 9600)

49.1.3.5.5 TColumnTitle.Font Property

Controls (see page 5711) the font in which the column title displays its caption.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

The Font property points to a TFont object that determines typographic attributes of text displayed in the column title. The default value is the grid's TitleFont property.

See Also

Font (see page 9596)

49.1.4 TCustomDBGrid Class

TCustomDBGrid is the base class for grid controls that display the records from a dataset in a tabular format.

Class Hierarchy

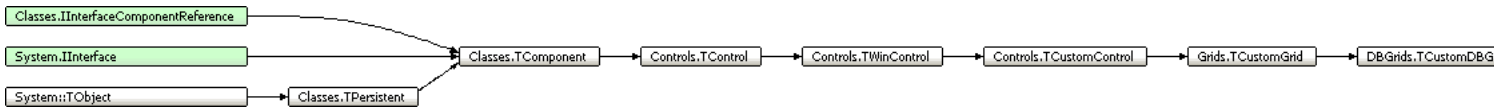

Pascal

```
TCustomDBGrid = class(TCustomGrid);
```

C++

```
class TCustomDBGrid : public TCustomGrid;
```

File

DBGrids

Description

TCustomDBGrid introduces new properties, events, and methods to expand the capabilities of TCustomGrid for displaying the fields from a dataset. At runtime, users can manipulate TCustomDBGrid objects using the TDBNavigator object.

Do not create instances of TCustomDBGrid. Use TCustomDBGrid as a base class when declaring grid objects that display information from datasets. Properties and methods of TCustomDBGrid provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TDBGrid (see page 9668)

TDataSet (see page 6894)

TDBNavigator (see page 9230)

49.1.4.1 TCustomDBGrid Members

The following tables list the members exposed by TCustomDBGrid.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnCellClick (see page 9664)	Occurs when the user releases the mouse in one of the cells of the grid.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnColEnter (see page 9665)	Occurs when focus moves to a new cell in the grid.
	OnColExit (see page 9665)	Occurs just before a cell in the grid loses input focus.
	OnColumnMoved (see page 9666)	Occurs when the user moves a column using the mouse.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDrawColumnCell (see page 9666)	Occurs when the grid needs to paint a cell.
	OnDrawDataCell (see page 9667)	Occurs when the grid needs to paint a cell if the State property of Columns (see page 9653) is csDefault.
	OnEditButtonClick (see page 9668)	Occurs when the user presses the ellipsis button in a grid column.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.

	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnTitleClick (see page 9668)	Occurs when the user releases the mouse in one of the column headers.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 14814)	This is DesignOptionsBoost, a member of class TCustomGrid.
	FAcquireFocus (see page 9623)	This is FAcquireFocus, a member of class TCustomDBGrid.
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FGridState (see page 14814)	This is FGridState, a member of class TCustomGrid.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveCellExtents (see page 14814)	This is FSaveCellExtents, a member of class TCustomGrid.
	FUpdateFields (see page 9623)	This is FUpdateFields, a member of class TCustomDBGrid.
	VirtualView (see page 14814)	This is VirtualView, a member of class TCustomGrid.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AcquireLayoutLock (see page 9625)	Calls BeginLayout (see page 9626) if the table is not currently being reformatted.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AdjustSize (see page 14816)	Adds or deletes rows or columns from the grid.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
V	BeginColumnDrag (see page 9625)	Starts the dragging of a column in the grid.
	BeginLayout (see page 9626)	Increments the LayoutLock (see page 9659) property when the column bindings or fields in the dataset change.

	BeginRowDrag (see page 14817)	Starts the dragging of a row in the grid.
	BeginUpdate (see page 9627)	Increments the UpdateLock (see page 9664) property when the data in the grid changes.
	BoxRect (see page 14818)	Returns the screen coordinates of a specified set of cells.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcDrawInfo (see page 14818)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 14818)	Calculates the pixel locations of the fixed cells.
V	CalcSizingState (see page 9627)	Determines the appearance of the mouse cursor during resizing.
	CalcTitleRect (see page 9627)	This is CalcTitleRect, a member of class TCustomDBGrid.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CancelLayout (see page 9628)	Decrements the LayoutLock (see page 9659) property.
V	CanEditAcceptKey (see page 9628)	Indicates whether the Key parameter can be entered into the current cell.
V	CanEditModify (see page 9629)	Indicates whether the current column of the grid can be edited.
V	CanEditShow (see page 9629)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 14820)	Indicates whether the grid should respond to a given key combination.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	CellClick (see page 9629)	Generates an OnCellClick (see page 9664) event.
	CellRect (see page 14821)	Returns the screen coordinates of a cell in the grid.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeGridOrientation (see page 14821)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
V	CheckColumnDrag (see page 9630)	Validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 14822)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
	Click (see page 5743)	Respond to user click.
	ColEnter (see page 9630)	Generates an OnColEnter (see page 9665) event.
	ColExit (see page 9630)	Generates an OnColExit (see page 9665) event.
	ColumnAtDepth (see page 9631)	Returns the column object associated with nth parent field of the given column.
V	ColumnMoved (see page 9631)	Generates an OnColumnMoved (see page 9666) event.
V	ColWidthsChanged (see page 9631)	Updates the column widths in the Columns (see page 9653) property when the user resizes the columns.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateColumns (see page 9632)	Creates the TDBGridColumns (see page 9746) object that stores the column descriptors for the grid.
	CreateDataLink (see page 9632)	Creates the data link that the grid uses to manage the connection to its data source.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateEditor (see page 9632)	Creates the inplace editor that allows the cells of the grid to be edited.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 14823)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 9633)	Creates the window used to display the data-aware grid.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.

	DataToRawColumn (see page 9633)	Converts a column index in the Columns (see page 9653) property to the index of the corresponding column in the visual display.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DeferLayout (see page 9635)	Decrements the LayoutLock (see page 9659) property and posts a message to the grid to trigger a new layout sequence at a more appropriate time.
V	DefineFieldMap (see page 9636)	Builds the Fields (see page 9657) array from the Columns (see page 9653) property and the data source.
V	DefineProperties (see page 9636)	Defines a savable version of the Columns (see page 9653) property when necessary.
V	DeleteColumn (see page 14824)	Deletes a specified column from the grid.
V	DeleteRow (see page 14825)	Deletes a specified from the grid.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
V	DoExit (see page 14825)	Hides the inplace editor when selection moves to another control in a form.
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
V	DoMouseWheelDown (see page 9636)	Performs tasks when the mouse wheel is rotated downward.
V	DoMouseWheelUp (see page 9637)	Performs tasks when the mouse wheel is rotated upward.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawCell (see page 9637)	Draws the cell specified by the ACol and ARow parameters.
	DrawColumnCell (see page 9638)	Generates an OnDrawColumnCell (see page 9666) event.
	DrawDataCell (see page 9638)	Generates an OnDrawDataCell (see page 9667) event.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EditButtonClick (see page 9638)	Generates an OnEditButtonClick (see page 9668) event.
V	EndColumnDrag (see page 9639)	Ends the dragging of a column in the grid.
	EndLayout (see page 9639)	Decrements the LayoutLock (see page 9659) property and recomputes the Columns (see page 9653) property if LayoutLock (see page 9659) reaches 0.
	EndRowDrag (see page 14827)	Ends the dragging of a row in the grid.
	EndUpdate (see page 9640)	Decrements the UpdateLock (see page 9664) property.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.

	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	FocusCell (see page 14827)	Changes the current cell to a specified cell.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetColField (see page 9640)	Returns the field component that represents the binding of a specified column.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEditLimit (see page 9641)	Limits the number of characters that can be typed into cells representing string fields.
V	GetEditMask (see page 9641)	Returns the input mask associated with the field for the selected cell.
V	GetEditStyle (see page 9642)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
V	GetEditText (see page 9642)	Returns the value of the indicated cell formatted for editing.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFieldValue (see page 9643)	Returns the value of the indicated cell formatted for display.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetGridHeight (see page 14829)	Returns the value of the GridHeight (see page 14850) property.
	GetGridWidth (see page 14830)	Returns the value of the GridWidth (see page 14851) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HideEdit (see page 14830)	Hides the inplace editor.
	HideEditor (see page 14830)	Hides the inplace editor when it is not in use.
V	HighlightCell (see page 9643)	Indicates whether a specified cell should be highlighted when it is drawn.
	InvalidateCell (see page 14831)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCol (see page 14831)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	InvalidateEditor (see page 14832)	Causes the inplace editor to repaint itself, moving to a new location if necessary.

	InvalidateGrid (see page 14832)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 14832)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
	InvalidateTitles (see page 9644)	Invalidates the column titles so that they will be repainted.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
V	KeyDown (see page 9644)	Provides special processing when the user presses down on a key.
V	KeyPress (see page 9645)	Responds when the user presses a key.
	KeyUp (see page 6164)	Respond to released key.
V	LayoutChanged (see page 9645)	Recomputes the Columns (see page 9653) property.
V	LinkActive (see page 9646)	Recomputes the Columns (see page 9653) property and the vertical scroll bar of the grid.
V	Loaded (see page 9646)	Sets up the Columns (see page 9653) property once the data grid and all its child components are loaded.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 9647)	Provides special processing when the user clicks on the mouse.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
V	MouseMove (see page 14834)	Provides special processing when the user moves the mouse.
V	MouseUp (see page 9647)	Provides special processing when the user releases the mouse button.
	MoveColRow (see page 14835)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 14836)	Moves a column to a new position.
	MoveRow (see page 14836)	Moves a row to a new position.
V	Notification (see page 9647)	Responds to notifications that components are about to be destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 5920)	Renders the image of a custom control.
V	Paint (see page 14836)	Draws the image of the grid control on the screen.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	RawToDataColumn (see page 9648)	Converts the index of a column in the visual display to the corresponding index in the Columns (see page 9653) property.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.

	RowHeightsChanged (see page 14837)	Responds when the row heights change.
	RowMoved (see page 14837)	Provides the interface for a method that responds when the position of a row changes.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
V	Scroll (see page 9648)	Updates the image of the grid to reflect a new record position.
	ScrollData (see page 14838)	Scrolls the visual image of the cells in the grid.
V	SelectCell (see page 14838)	Determines whether a particular cell in the grid can be selected.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
V	SetColumnAttributes (see page 9649)	This is SetColumnAttributes, a member of class TCustomDBGrid.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEditText (see page 9649)	Updates the edited version of the current cell's data.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	ShowEditor (see page 14839)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 14839)	Displays the inplace editor, with its text set to the specified character.
	SizeChanged (see page 14840)	Responds to changes in the size of the grid.
	Sizing (see page 14840)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	StoreColumns (see page 9650)	Indicates whether the columns of the grid are stored when the grid is saved, or whether they are generated dynamically.
V	TimedScroll (see page 9650)	Moves the current record in response to the user dragging the mouse outside of the data cells.
	TitleClick (see page 9651)	Generates an OnTitleClick (see page 9668) event.
V	TopLeftChanged (see page 9651)	Updates the position of the inplace editor when the grid scrolls.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.

V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateDesigner (see page 14841)	Notifies the designer when rows and columns of the grid are moved or resized.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
V	UpdateScrollBar (see page 9652)	This is UpdateScrollBar, a member of class TCustomDBGrid.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UseRightToLeftAlignmentForField (see page 9652)	Indicates whether the alignment of a specified field should be changed from left-aligned to right-aligned.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 14843)	Determines whether a single line border is drawn around the grid.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Col (see page 14843)	Specifies the index of the column that contains the selected cell.
	ColCount (see page 14844)	Specifies the number of columns in the grid.
	Color (see page 5812)	Specifies the background color of the control.
	Color (see page 14844)	Specifies the background color of the control.
	Columns (see page 9653)	Describes the display attributes and field bindings of the columns.
	ColWidths (see page 14845)	Indicates the width (in pixels) of all the columns in the grid.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
R	DataLink (see page 9654)	Helps the data-aware grid manage its link to the data source and respond to data events.
	DefaultColWidth (see page 14846)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 9655)	Indicates whether the cells in the data-aware grid are drawn automatically.
	DefaultRowHeight (see page 14847)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.

	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	FixedColor (see page 14848)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 14848)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 14849)	Specifies the number of rows on the top of the grid that cannot be scrolled.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
R	GridHeight (see page 14850)	Specifies the height of the grid in pixels.
	GridLineWidth (see page 14850)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
R	GridWidth (see page 14851)	Specifies the width of the grid in pixels.
R	HitTest (see page 14851)	Indicates the current position of the mouse cursor.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
R	IndicatorOffset (see page 9658)	Indicates the index of the first column in the grid that contains data.
R	InplaceEditor (see page 14852)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	LayoutLock (see page 9659)	Counts the number of times BeginLayout (see page 9626) has been called without an EndLayout (see page 9639) or CancelLayout (see page 9628) call.
	LeftCol (see page 14852)	Specifies the index of the first visible scrollable column in the grid.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	Options (see page 9659)	Specifies various display and behavioral properties of the data-aware grid.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 9660)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 9660)	Indicates whether the grid is used for display only, or whether the user can edit data using the grid.
	Row (see page 14854)	Specifies the index of the row that contains the selected cell.
	RowCount (see page 14854)	Specifies the number of rows in the grid.
	RowHeights (see page 14855)	Gives the height (in pixels) of all rows in the grid.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ScrollBars (see page 14856)	Specifies whether the grid includes horizontal and vertical scroll bars.
R	SelectedRows (see page 9662)	Specifies a set of bookmarks for all the records in the dataset that correspond to rows selected in the grid.
	Selection (see page 14856)	Indicates the boundaries of the current selection.
	TabStops (see page 14858)	Indicates whether the user can tab to specified columns in the grid.
	Text (see page 5842)	Contains a text string associated with the control.
	TitleFont (see page 9663)	Describes the font used to draw the column titles in the grid.
	TopRow (see page 14858)	Specifies the index of the first visible scrollable row in the grid.
R	UpdateLock (see page 9664)	Counts the number of times BeginUpdate (see page 9627) is called without a corresponding call to EndUpdate (see page 9640).
R	VisibleColCount (see page 14859)	Indicates the number of scrollable columns visible in the grid.
R	VisibleRowCount (see page 14859)	Indicates the number of scrollable rows visible in the grid.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 9622)	Creates and initializes an instance of TCustomDBGrid (see page 9608).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultDrawColumnCell (see page 9634)	Draws the text in a column cell.
	DefaultDrawDataCell (see page 9634)	Draws the text in a column cell.
V	DefaultHandler (see page 9635)	Displays the popup menu when a column is right-clicked.
V	Destroy (see page 9622)	Destroys an instance of TCustomDBGrid (see page 9608).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
V	ExecuteAction (see page 9640)	Executes an action with the DB (see page 6725) grid as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseCoord (see page 14834)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	ShowPopupEditor (see page 9649)	Displays the nested dataset associated with Column in a new window.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
V	UpdateAction (see page 9651)	Updates an action to reflect the current state of the grid.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.
	ValidFieldIndex (see page 9652)	Indicates whether an index into the columns array refers to a valid field in the dataset.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.

	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DataSource (see page 9655)	Identifies the link to the dataset where the data-aware grid finds its data.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EditorMode (see page 9656)	Determines whether the current cell can be edited.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	FieldCount (see page 9656)	Specifies the number of fields (columns) displayed in the grid.
R	Fields (see page 9657)	Provides indexed access to the field components that correspond to the columns.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelectedField (see page 9661)	Specifies the field component for the currently selected cell in the grid.
	SelectedIndex (see page 9661)	Specifies the index of the currently selected column in the Columns (see page 9653) array.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.

	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	TabStop (see page 14857)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

49.1.4.2 TCustomDBGrid.Create Constructor

Creates and initializes an instance of TCustomDBGrid (see page 9608).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomDBGrid(TComponent * AOwner);
```

Description

Call Create to instantiate a data-aware grid at runtime. Grids (see page 14765) placed on a form at design time are created automatically.

Pass a single Component as a parameter to provide the grid with an Owner that is responsible for freeing the grid.

See Also

DefaultDrawing (see page 9655)

Options (see page 9659)

ParentColor (see page 5834)

Color (see page 5812)

Create (see page 14813)

49.1.4.3 TCustomDBGrid.Destroy Destructor

Destroys an instance of TCustomDBGrid (see page 9608).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomDBGrid();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the data-aware grid is not nil, and only then calls Destroy. Destroy frees all the helper objects used by the grid.

See Also

Free (see page 22300)

Create (see page 9622)

49.1.4.4 TCustomDBGrid Fields

The fields of the TCustomDBGrid class are listed here.

Protected Fields

	Name	Description
	FAcquireFocus (see page 9623)	This is FAcquireFocus, a member of class TCustomDBGrid.
	FUpdateFields (see page 9623)	This is FUpdateFields, a member of class TCustomDBGrid.

49.1.4.4.1 TCustomDBGrid.FAcquireFocus Field

Pascal

```
FAcquireFocus: Boolean;
```

C++

```
Boolean FAcquireFocus;
```

Description

This is FAcquireFocus, a member of class TCustomDBGrid.

49.1.4.4.2 TCustomDBGrid.FUpdateFields Field

Pascal

```
FUpdateFields: Boolean;
```

C++

```
Boolean FUpdateFields;
```

Description

This is FUpdateFields, a member of class TCustomDBGrid.

49.1.4.5 TCustomDBGrid Methods

The methods of the TCustomDBGrid class are listed here.

Protected Methods

	Name	Description
	AcquireLayoutLock (see page 9625)	Calls BeginLayout (see page 9626) if the table is not currently being reformatted.
	BeginColumnDrag (see page 9625)	Starts the dragging of a column in the grid.
	BeginLayout (see page 9626)	Increments the LayoutLock (see page 9659) property when the column bindings or fields in the dataset change.

	BeginUpdate (see page 9627)	Increments the UpdateLock (see page 9664) property when the data in the grid changes.
V	CalcSizingState (see page 9627)	Determines the appearance of the mouse cursor during resizing.
	CalcTitleRect (see page 9627)	This is CalcTitleRect, a member of class TCustomDBGrid.
	CancelLayout (see page 9628)	Decrements the LayoutLock (see page 9659) property.
V	CanEditAcceptKey (see page 9628)	Indicates whether the Key parameter can be entered into the current cell.
V	CanEditModify (see page 9629)	Indicates whether the current column of the grid can be edited.
V	CanEditShow (see page 9629)	Indicates whether the inplace edit control can be created to allow editing.
	CellClick (see page 9629)	Generates an OnCellClick (see page 9664) event.
V	CheckColumnDrag (see page 9630)	Validates whether the column currently selected for dragging can be dropped at the current location.
	ColEnter (see page 9630)	Generates an OnColEnter (see page 9665) event.
	ColExit (see page 9630)	Generates an OnColExit (see page 9665) event.
	ColumnAtDepth (see page 9631)	Returns the column object associated with nth parent field of the given column.
V	ColumnMoved (see page 9631)	Generates an OnColumnMoved (see page 9666) event.
V	ColWidthsChanged (see page 9631)	Updates the column widths in the Columns (see page 9653) property when the user resizes the columns.
	CreateColumns (see page 9632)	Creates the TDBGridColumns (see page 9746) object that stores the column descriptors for the grid.
	CreateDataLink (see page 9632)	Creates the data link that the grid uses to manage the connection to its data source.
V	CreateEditor (see page 9632)	Creates the inplace editor that allows the cells of the grid to be edited.
V	CreateWnd (see page 9633)	Creates the window used to display the data-aware grid.
	DataToRawColumn (see page 9633)	Converts a column index in the Columns (see page 9653) property to the index of the corresponding column in the visual display.
	DeferLayout (see page 9635)	Decrements the LayoutLock (see page 9659) property and posts a message to the grid to trigger a new layout sequence at a more appropriate time.
V	DefineFieldMap (see page 9636)	Builds the Fields (see page 9657) array from the Columns (see page 9653) property and the data source.
V	DefineProperties (see page 9636)	Defines a savable version of the Columns (see page 9653) property when necessary.
V	DoMouseWheelDown (see page 9636)	Performs tasks when the mouse wheel is rotated downward.
V	DoMouseWheelUp (see page 9637)	Performs tasks when the mouse wheel is rotated upward.
V	DrawCell (see page 9637)	Draws the cell specified by the ACol and ARow parameters.
	DrawColumnCell (see page 9638)	Generates an OnDrawColumnCell (see page 9666) event.
	DrawDataCell (see page 9638)	Generates an OnDrawDataCell (see page 9667) event.
	EditButtonClick (see page 9638)	Generates an OnEditButtonClick (see page 9668) event.
V	EndColumnDrag (see page 9639)	Ends the dragging of a column in the grid.
	EndLayout (see page 9639)	Decrements the LayoutLock (see page 9659) property and recomputes the Columns (see page 9653) property if LayoutLock (see page 9659) reaches 0.
	EndUpdate (see page 9640)	Decrements the UpdateLock (see page 9664) property.
	GetColField (see page 9640)	Returns the field component that represents the binding of a specified column.
V	GetEditLimit (see page 9641)	Limits the number of characters that can be typed into cells representing string fields.
V	GetEditMask (see page 9641)	Returns the input mask associated with the field for the selected cell.
V	GetEditStyle (see page 9642)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
V	GetEditText (see page 9642)	Returns the value of the indicated cell formatted for editing.
	GetFieldValue (see page 9643)	Returns the value of the indicated cell formatted for display.
V	HighlightCell (see page 9643)	Indicates whether a specified cell should be highlighted when it is drawn.
	InvalidateTitles (see page 9644)	Invalidates the column titles so that they will be repainted.
V	KeyDown (see page 9644)	Provides special processing when the user presses down on a key.
V	KeyPress (see page 9645)	Responds when the user presses a key.
V	LayoutChanged (see page 9645)	Recomputes the Columns (see page 9653) property.
V	LinkActive (see page 9646)	Recomputes the Columns (see page 9653) property and the vertical scroll bar of the grid.
V	Loaded (see page 9646)	Sets up the Columns (see page 9653) property once the data grid and all its child components are loaded.
V	MouseDown (see page 9647)	Provides special processing when the user clicks on the mouse.

	MouseUp (see page 9647)	Provides special processing when the user releases the mouse button.
	Notification (see page 9647)	Responds to notifications that components are about to be destroyed.
	RawToDataColumn (see page 9648)	Converts the index of a column in the visual display to the corresponding index in the Columns (see page 9653) property.
	Scroll (see page 9648)	Updates the image of the grid to reflect a new record position.
	SetColumnAttributes (see page 9649)	This is SetColumnAttributes, a member of class TCustomDBGrid.
	SetEditText (see page 9649)	Updates the edited version of the current cell's data.
	StoreColumns (see page 9650)	Indicates whether the columns of the grid are stored when the grid is saved, or whether they are generated dynamically.
	TimedScroll (see page 9650)	Moves the current record in response to the user dragging the mouse outside of the data cells.
	TitleClick (see page 9651)	Generates an OnTitleClick (see page 9668) event.
	TopLeftChanged (see page 9651)	Updates the position of the inplace editor when the grid scrolls.
	UpdateScrollBar (see page 9652)	This is UpdateScrollBar, a member of class TCustomDBGrid.
	UseRightToLeftAlignmentForField (see page 9652)	Indicates whether the alignment of a specified field should be changed from left-aligned to right-aligned.

Public Methods

	Name	Description
	DefaultDrawColumnCell (see page 9634)	Draws the text in a column cell.
	DefaultDrawDataCell (see page 9634)	Draws the text in a column cell.
	DefaultHandler (see page 9635)	Displays the popup menu when a column is right-clicked.
	ExecuteAction (see page 9640)	Executes an action with the DB (see page 6725) grid as its target.
	ShowPopupEditor (see page 9649)	Displays the nested dataset associated with Column in a new window.
	UpdateAction (see page 9651)	Updates an action to reflect the current state of the grid.
	ValidFieldIndex (see page 9652)	Indicates whether an index into the columns array refers to a valid field in the dataset.

49.1.4.5.1 TCustomDBGrid.AcquireLayoutLock Method

Calls BeginLayout (see page 9626) if the table is not currently being reformatted.

Pascal

```
function AcquireLayoutLock: Boolean;
```

C++

```
__fastcall Boolean AcquireLayoutLock();
```

Description

Applications cannot call this protected method. AcquireLayoutLock is called before recomputing Columns (see page 9653) when the fields in the dataset change or when the Columns (see page 9653) property changes. AcquireLayoutLock returns true if the table is not already being laid out, to indicate that the computation of Columns (see page 9653) should proceed. AcquireLayoutLock prevents the computation of Columns (see page 9653) from looping endlessly because of the changes that must occur during the computation.

See Also

BeginLayout (see page 9626)

Columns (see page 9653)

LayoutChanged (see page 9645)

49.1.4.5.2 TCustomDBGrid.BeginColumnDrag Method

Starts the dragging of a column in the grid.

Pascal

```
function BeginColumnDrag(var Origin: Integer; var Destination: Integer; const MousePt:
TPoint): Boolean; override;
```

C++

```
virtual __fastcall Boolean BeginColumnDrag(int Origin, int Destination, const TPoint MousePt);
```

Description

BeginColumnDrag is called automatically when the user clicks a column to initiate a drag operation.

Origin is the column number of the column to move. When BeginColumnDrag is called, this is the column under the mouse. BeginColumnDrag may adjust this value to take into account non-scrolling columns.

Destination is the current destination of the column. On both entry and exit, this is the same as Origin, because the column has not yet moved.

MousePt is the coordinates of the mouse at the point when the button was clicked.

BeginColumnDrag returns true if the grid permits the column to be moved, false if the column move should be aborted.

See Also

BeginRowDrag (see page 14817)

EndColumnDrag (see page 9639)

49.1.4.5.3 TCustomDBGrid.BeginLayout Method

Increments the LayoutLock (see page 9659) property when the column bindings or fields in the dataset change.

Pascal

```
procedure BeginLayout;
```

C++

```
__fastcall BeginLayout();
```

Description

Applications cannot call this protected method. Data-aware grids call BeginLayout internally before making changes to the Columns (see page 9653) property or any of the individual TColumn (see page 9583) objects in Columns (see page 9653). Once the change is complete, the data-aware grid calls EndLayout (see page 9639). If the response to the change must be canceled or postponed, the grid calls CancelLayout (see page 9628).

Each call to BeginLayout increments the LayoutLock (see page 9659) property. Each call to EndLayout (see page 9639) or CancelLayout (see page 9628) decrements the LayoutLock (see page 9659) property. If LayoutLock (see page 9659) returns to 0 in a call to EndLayout (see page 9639), the Fields (see page 9657) property is updated to match both the bindings of the Columns (see page 9653) property and the fields available in the dataset. The Columns (see page 9653) property is then updated in turn to match the Fields (see page 9657) property. This ensures that the Columns (see page 9653) property matches both the specified column bindings and the fields available in the dataset.

See Also

AcquireLayoutLock (see page 9625)

CancelLayout (see page 9628)

Columns (see page 9653)

DeferLayout (see page 9635)

DefineFieldMap (see page 9636)

EndLayout (see page 9639)

Fields (see page 9657)

LayoutLock (see page 9659)

49.1.4.5.4 TCustomDBGrid.BeginUpdate Method

Increments the UpdateLock (see page 9664) property when the data in the grid changes.

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

Applications cannot call this protected method. Data-aware grids call BeginUpdate internally before making changes that affect the data that is drawn in the grid. Once the change is complete, the data-aware grid calls EndUpdate (see page 9640), which decrements the UpdateLock (see page 9664) property.

While UpdateLock (see page 9664) is greater than 0, the grid does not repaint its cells. BeginUpdate and EndUpdate (see page 9640) prevent the grid from flickering when the grid is resized or the user scrolls through the data.

See Also

EndUpdate (see page 9640)

UpdateLock (see page 9664)

49.1.4.5.5 TCustomDBGrid.CalcSizingState Method

Determines the appearance of the mouse cursor during resizing.

Pascal

```
procedure CalcSizingState(X: Integer; Y: Integer; var State: TGridState; var Index: Longint;
var SizingPos: Integer; var SizingOfs: Integer; var FixedInfo: TGridDrawInfo); override;
```

C++

```
virtual __fastcall CalcSizingState(int X, int Y, TGridState State, Longint Index, int
SizingPos, int SizingOfs, TGridDrawInfo FixedInfo);
```

Description

Use CalcSizingState to determine if the mouse cursor should be changed to a resize or drag cursor based on where the mouse is in the grid. For example, cursor over a grid line, in the fixed cell region, and column resize is enabled = resize cursor.

See Also

CalcFixedInfo (see page 14818)

49.1.4.5.6 TCustomDBGrid.CalcTitleRect Method

Pascal

```
function CalcTitleRect(Col: TColumn; ARow: Integer; var MasterCol: TColumn): TRect;
```

C++

```
__fastcall TRect CalcTitleRect(TColumn Col, int ARow, TColumn MasterCol);
```

Description

This is CalcTitleRect, a member of class TCustomDBGrid.

49.1.4.5.7 TCustomDBGrid.CancelLayout Method

Decrements the LayoutLock (see page 9659) property.

Pascal

```
procedure CancelLayout;
```

C++

```
__fastcall CancelLayout();
```

Description

Applications cannot call this protected method. Data-aware grids call CancelLayout internally when postponing the recomputation of Columns (see page 9653).

Each call to BeginLayout (see page 9626) increments the LayoutLock (see page 9659) property. Each call to EndLayout (see page 9639) or CancelLayout decrements the LayoutLock (see page 9659) property. If LayoutLock (see page 9659) returns to 0 in a call to CancelLayout, the Columns (see page 9653) property is not updated. CancelLayout is called from the DeferLayout (see page 9635) method to decrement the LayoutLock (see page 9659) and prevent the recomputation of Columns (see page 9653) until a more appropriate time.

See Also

BeginLayout (see page 9626)

Columns (see page 9653)

DeferLayout (see page 9635)

EndLayout (see page 9639)

LayoutLock (see page 9659)

49.1.4.5.8 TCustomDBGrid.CanEditAcceptKey Method

Indicates whether the Key parameter can be entered into the current cell.

Pascal

```
function CanEditAcceptKey(Key: Char): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanEditAcceptKey(Char Key);
```

Description

Data-aware grids use CanEditAcceptKey internally to determine whether keys typed by the user are valid for the field represented by the current cell. CanEditAcceptKey calls the IsValidChar method of the field component associated with the column that has input focus.

CanEditAcceptKey provides a crude validation of keyboard input. More refined validation is provided when the value in the cell is written to the record in the dataset, or by an edit mask.

See Also

GetColField (see page 9640)

GetEditMask (see page 9641)

IsValidChar (see page 7136)

49.1.4.5.9 TCustomDBGrid.CanEditModify Method

Indicates whether the current column of the grid can be edited.

Pascal

```
function CanEditModify: Boolean; override;
```

C++

```
virtual __fastcall Boolean CanEditModify();
```

Description

Data-aware grids use CanEditModify internally to determine whether to allow the user to edit the current cell. CanEditModify returns true if the field in the current column can be edited. CanEditModify returns false if the field cannot be edited. If the dataset is not in edit mode, CanEditModify will attempt to put it in edit mode.

Note: CanEditModify returns false when the current field is a binary large object (BLOB) field. BLOB fields can only be edited through the clipboard or by streaming.

See Also

ReadOnly (see page 9660)

ReadOnly (see page 9599)

CanModify (see page 7154)

49.1.4.5.10 TCustomDBGrid.CanEditShow Method

Indicates whether the inplace edit control can be created to allow editing.

Pascal

```
function CanEditShow: Boolean; override;
```

C++

```
virtual __fastcall Boolean CanEditShow();
```

Description

TCustomDBGrid (see page 9608) overrides the inherited method to prevent the inplace edit control from coming up while the layout of the grid is being recomputed.

Grid controls call CanEditShow internally before creating the editor for a cell. CanEditShow is called after the grid has determined that the field can be edited by calling CanEditModify (see page 9629).

See Also

CanEditModify (see page 9629)

CanEditShow (see page 14820)

49.1.4.5.11 TCustomDBGrid.CellClick Method

Generates an OnCellClick (see page 9664) event.

Pascal

```
procedure CellClick(Column: TColumn); dynamic;
```

C++

```
__fastcall CellClick(TColumn Column);
```

Description

Applications cannot directly call CellClick. It is exposed as a protected method so that descendants can call the OnCellClick (see page 9664) event handler in response to other events or override the response when the user releases the mouse in a cell of the grid.

See Also

OnCellClick (see page 9664)

49.1.4.5.12 TCustomDBGrid.CheckColumnDrag Method

Validates whether the column currently selected for dragging can be dropped at the current location.

Pascal

```
function CheckColumnDrag(var Origin: Integer; var Destination: Integer; const MousePt:
TPoint): Boolean; override;
```

C++

```
virtual __fastcall Boolean CheckColumnDrag(int Origin, int Destination, const TPoint MousePt);
```

Description

Use CheckColumnDrag to validate that the column currently selected for dragging can be dropped at the current location. CheckColumnDrag can change the Origin (the column being dragged) and/or the Destination (the drop target), as needed, to keep the grid's columns in order.

See Also

BeginColumnDrag (see page 14817)

EndColumnDrag (see page 14827)

49.1.4.5.13 TCustomDBGrid.ColEnter Method

Generates an OnColEnter (see page 9665) event.

Pascal

```
procedure ColEnter; dynamic;
```

C++

```
__fastcall ColEnter();
```

Description

Applications cannot directly call ColEnter. It is exposed as a protected method so that descendants can call the OnCellEnter event handler in response to other events or override the response when a cell in the grid is selected.

See Also

OnColEnter (see page 9665)

49.1.4.5.14 TCustomDBGrid.ColExit Method

Generates an OnColExit (see page 9665) event.

Pascal

```
procedure ColExit; dynamic;
```

C++

```
__fastcall ColExit();
```

Description

Applications cannot directly call ColExit. It is exposed as a protected method so that descendants can call the OnColExit (see page 9665) event handler in response to other events or override the response when a cell in the grid loses focus.

See Also

OnColExit (see page 9665)

49.1.4.5.15 TCustomDBGrid.ColumnAtDepth Method

Returns the column object associated with nth parent field of the given column.

Pascal

```
function ColumnAtDepth(Col: TColumn; ADepth: Integer): TColumn;
```

C++

```
__fastcall TColumn ColumnAtDepth(TColumn Col, int ADepth);
```

Description

ColumnAtDepth is used to calculate MasterCol used in the CalcTitleRec method.

See Also

CalcTitleRec

49.1.4.5.16 TCustomDBGrid.ColumnMoved Method

Generates an OnColumnMoved (see page 9666) event.

Pascal

```
procedure ColumnMoved(FromIndex: Longint; ToIndex: Longint); override;
```

C++

```
virtual __fastcall ColumnMoved(Longint FromIndex, Longint ToIndex);
```

Description

Applications cannot directly call ColumnMoved. It is exposed as a protected method so that descendants can call the OnColumnMoved (see page 9666) event handler in response to other events or override the response after a column in the grid is moved.

See Also

OnColumnMoved (see page 9666)

49.1.4.5.17 TCustomDBGrid.ColWidthsChanged Method

Updates the column widths in the Columns (see page 9653) property when the user resizes the columns.

Pascal

```
procedure ColWidthsChanged; override;
```

C++

```
virtual __fastcall ColWidthsChanged();
```

Description

Applications cannot directly call ColWidthsChanged. It is called automatically when the user resizes the columns in the grid. ColWidthsChanged overrides the inherited method to update the column widths in the Columns (see page 9653) property.

See Also

Columns (see page 9653)

Width (see page 9600)

49.1.4.5.18 TCustomDBGrid.CreateColumns Method

Creates the TDBGridColumns (see page 9746) object that stores the column descriptors for the grid.

Pascal

```
function CreateColumns: TDBGridColumns; dynamic;
```

C++

```
__fastcall TDBGridColumns CreateColumns();
```

Description

Applications cannot call CreateColumns. It is exposed as a protected method so that component writers can substitute a descendant of TDBGridColumns (see page 9746) to manage the column descriptors of a data-aware grid.

See Also

Columns (see page 9653)

49.1.4.5.19 TCustomDBGrid.CreateDataLink Method

Creates the data link that the grid uses to manage the connection to its data source.

Pascal

```
function CreateDataLink: TGridDataLink; dynamic;
```

C++

```
__fastcall TGridDataLink CreateDataLink();
```

Description

TCustomDBGrid (see page 9608) calls CreateDataLink from its constructor to create the data link that manages the connection to a data source. Descendant classes can override this method to create a different TGridDataLink (see page 9754) descendant.

See Also

DataSource (see page 9655)

49.1.4.5.20 TCustomDBGrid.CreateEditor Method

Creates the inplace editor that allows the cells of the grid to be edited.

Pascal

```
function CreateEditor: TInplaceEdit; override;
```

C++

```
virtual __fastcall TInplaceEdit CreateEditor();
```

Description

CreateEditor overrides the inherited method to create a data-aware descendant of TInplaceEdit.

See Also

TInplaceEdit (see page 14938)

49.1.4.5.21 TCustomDBGrid.CreateWnd Method

Creates the window used to display the data-aware grid.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

The Windows control for the grid can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. TCustomDBGrid (see page 9608) calls CreateWnd to create this Windows control when necessary.

CreateWnd overrides the inherited method to set up the relationship between the visual display and the data source after the window is created.

See Also

CreateWnd (see page 6137)

49.1.4.5.22 TCustomDBGrid.DataToRawColumn Method

Converts a column index in the Columns (see page 9653) property to the index of the corresponding column in the visual display.

Pascal

```
function DataToRawColumn(ACol: Integer): Integer;
```

C++

```
__fastcall int DataToRawColumn(int ACol);
```

Description

Applications cannot call this protected method. It is used internally to allow the data-aware grid to adjust to the presence or absence of the column that contains the row indicator.

See Also

Columns (see page 9653)

IndicatorOffset (see page 9658)

RawToDataColumn (see page 9648)

49.1.4.5.23 TCustomDBGrid.DefaultDrawColumnCell Method

Draws the text in a column cell.

Pascal

```
procedure DefaultDrawColumnCell(const Rect: TRect; DataCol: Integer; Column: TColumn; State:
TGridDrawState);
```

C++

```
__fastcall DefaultDrawColumnCell(const TRect Rect, int DataCol, TColumn Column, TGridDrawState
State);
```

Description

Call DefaultDrawColumnCell from an OnDrawColumnCell (see page 9666) event handler to lookup up the text representation of a field and write it to the cell. DefaultDrawColumnCell does exactly what the grid does when the DefaultDrawing (see page 9655) property is true, except it never draws a focus rectangle on selected cells.

The Rect parameter is the position of the cell on the canvas. The DataCol parameter is the index of the column in the Columns (see page 9653) array. The Column parameter is the TColumn (see page 9583) object that describes the display attributes and field binding for the cell. The State parameter indicates whether the cell has input focus, whether the cell is selected, and whether the cell is a fixed (non-data) cell such as a column header.

See Also

Columns (see page 9653)

DefaultDrawing (see page 9655)

OnDrawColumnCell (see page 9666)

49.1.4.5.24 TCustomDBGrid.DefaultDrawDataCell Method

Draws the text in a column cell.

Pascal

```
procedure DefaultDrawDataCell(const Rect: TRect; Field: TField; State: TGridDrawState);
```

C++

```
__fastcall DefaultDrawDataCell(const TRect Rect, TField Field, TGridDrawState State);
```

Description

Call DefaultDrawDataCell from an OnDrawDataCell (see page 9667) event handler to lookup up the text representation of a field and write it to the cell. Unlike what the grid draws when the DefaultDrawing (see page 9655) property is true, DefaultDrawDataCell gets the alignment for the text from the field component rather than the column descriptor, and it never draws a focus rectangle on selected cells.

The Rect parameter gives the position of the cell on the canvas. The Field parameter gives the field component that provides the data and alignment for the cell. The State parameter indicates whether the cell has input focus, whether the cell is selected, and whether the cell is a fixed (non-data) cell such as a column header.

See Also

DefaultDrawing (see page 9655)

OnDrawDataCell (see page 9667)

49.1.4.5.25 TCustomDBGrid.DefaultHandler Method

Displays the popup menu when a column is right-clicked.

Pascal

```
procedure DefaultHandler(var Msg); override;
```

C++

```
virtual __fastcall DefaultHandler( Msg);
```

Description

DefaultHandler is the last method to receive Windows messages that are sent to the grid. After calling the inherited method and allowing it to process the message, DefaultHandler checks if the Msg parameter indicates the right mouse button was released over a column with a popup menu. If the column has a popup menu with its AutoPopup property set to true, DefaultHandler displays the popup menu.

When overriding DefaultHandler, be sure to call the inherited method for all unhandled messages.

See Also

PopupMenu (see page 9599)

AutoPopup (see page 16178)

DefaultHandler (see page 6139)

49.1.4.5.26 TCustomDBGrid.DeferLayout Method

Decrements the LayoutLock (see page 9659) property and posts a message to the grid to trigger a new layout sequence at a more appropriate time.

Pascal

```
procedure DeferLayout;
```

C++

```
__fastcall DeferLayout();
```

Description

Applications cannot call this protected method. Data-aware grids call DeferLayout internally to postpone the recomputation of the Columns (see page 9653) property after the dataset changes when it is inappropriate to recompute columns immediately.

DeferLayout calls CancelLayout (see page 9628) to decrement the LayoutLock (see page 9659) property, which was incremented in response to the change in the dataset. It then posts a message to the grid, which triggers another layout attempt at a later point in time. When the message is received, the grid calls BeginLayout (see page 9626) and then either EndLayout (see page 9639) or DeferLayout, depending on whether it is an appropriate time to proceed with the computation of Columns (see page 9653).

See Also

BeginLayout (see page 9626)

CancelLayout (see page 9628)

Columns (see page 9653)

EndLayout (see page 9639)

LayoutLock (see page 9659)

49.1.4.5.27 TCustomDBGrid.DefineFieldMap Method

Builds the Fields (see page 9657) array from the Columns (see page 9653) property and the data source.

Pascal

```
procedure DefineFieldMap; virtual;
```

C++

```
virtual __fastcall DefineFieldMap();
```

Description

Applications cannot call this protected method. DefineFieldMap rebuilds the Fields (see page 9657) array to reflect the fields from the data source that correspond to the columns in the Columns (see page 9653) property. After calling DefineFieldMap, the Columns (see page 9653) property should be updated to use the field components in the Fields (see page 9657) array. This ensures that the Columns (see page 9653) property reflects any changes imposed by the dataset.

See Also

Columns (see page 9653)

EndLayout (see page 9639)

Fields (see page 9657)

49.1.4.5.28 TCustomDBGrid.DefineProperties Method

Defines a savable version of the Columns (see page 9653) property when necessary.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

DefineProperties overrides the inherited method to load and save the Columns (see page 9653) property if the columns have been customized.

See Also

Columns (see page 9653)

DefineProperties (see page 2931)

49.1.4.5.29 TCustomDBGrid.DoMouseWheelDown Method

Performs tasks when the mouse wheel is rotated downward.

Pascal

```
function DoMouseWheelDown(Shift: TShiftState; MousePos: TPoint): Boolean; override;
```

C++

```
virtual __fastcall Boolean DoMouseWheelDown(TShiftState Shift, TPoint MousePos);
```

Description

Override DoMouseWheelDown to perform tasks when the mouse wheel is rotated downward.

DoMouseWheelDown returns false if the control's parent should handle the message.

In TCustomGrid, DoMouseWheelDown increments the current row and returns true.

See Also

DoMouseWheelUp (see page 14826)

Row (see page 14854)

49.1.4.5.30 TCustomDBGrid.DoMouseWheelUp Method

Performs tasks when the mouse wheel is rotated upward.

Pascal

```
function DoMouseWheelUp(Shift: TShiftState; MousePos: TPoint): Boolean; override;
```

C++

```
virtual __fastcall Boolean DoMouseWheelUp(TShiftState Shift, TPoint MousePos);
```

Description

Override DoMouseWheelUp to perform tasks when the mouse wheel is rotated downward.

DoMouseWheelUp should return false if the control's parent should handle the message.

In TCustomGrid, DoMouseWheelUp decrements the current row and returns true.

See Also

DoMouseWheelDown (see page 14825)

Row (see page 14854)

49.1.4.5.31 TCustomDBGrid.DrawCell Method

Draws the cell specified by the ACol and ARow parameters.

Pascal

```
procedure DrawCell(ACol: Longint; ARow: Longint; ARect: TRect; AState: TGridDrawState);
override;
```

C++

```
virtual __fastcall DrawCell(Longint ACol, Longint ARow, TRect ARect, TGridDrawState AState);
```

Description

DrawCell overrides the inherited method to use the Columns (see page 9653) property when drawing cells in the grid.

DrawCell sets up the font and brush and paints the background of the cell. If the DefaultDrawing (see page 9655) property is true, DrawCell fetches the data for the cell and writes it to the position described by the ARect parameter. Whether DefaultDrawing (see page 9655) is true or false, DrawCell generates the OnDrawDataCell (see page 9667) and OnDrawColumnCell (see page 9666) events. Finally, if DefaultDrawing (see page 9655) is true and either the cell has focus or the cell is selected and Options (see page 9659) includes dgAlwaysShowSelection, DrawCell draws a focus rectangle around the cell.

See Also

Columns (see page 9653)

DefaultDrawing (see page 9655)

OnDrawColumnCell (see page 9666)

OnDrawDataCell (see page 9667)

Options (see page 9659)

49.1.4.5.32 TCustomDBGrid.DrawColumnCell Method

Generates an OnDrawColumnCell (see page 9666) event.

Pascal

```
procedure DrawColumnCell(const Rect: TRect; DataCol: Integer; Column: TColumn; State:
TGridDrawState); dynamic;
```

C++

```
__fastcall DrawColumnCell(const TRect Rect, int DataCol, TColumn Column, TGridDrawState State);
```

Description

Applications cannot call DrawColumnCell directly. It is exposed as a protected method so that descendants can call the OnDrawColumnCell (see page 9666) event handler or override the method to block the event.

See Also

OnDrawColumnCell (see page 9666)

49.1.4.5.33 TCustomDBGrid.DrawDataCell Method

Generates an OnDrawDataCell (see page 9667) event.

Pascal

```
procedure DrawDataCell(const Rect: TRect; Field: TField; State: TGridDrawState); dynamic;
```

C++

```
__fastcall DrawDataCell(const TRect Rect, TField Field, TGridDrawState State);
```

Description

Applications cannot call DrawDataCell directly. It is exposed as a protected method so that descendants can call the OnDrawDataCell (see page 9667) event handler or override the method to block the event.

See Also

OnDrawDataCell (see page 9667)

49.1.4.5.34 TCustomDBGrid.EditButtonClick Method

Generates an OnEditButtonClick (see page 9668) event.

Pascal

```
procedure EditButtonClick; dynamic;
```

C++

```
__fastcall EditButtonClick();
```

Description

Applications cannot call EditButtonClick directly. It is exposed as a protected method so that descendants can call the OnEditButtonClick (see page 9668) event handler or override the method to block the event.

See Also

OnEditButtonClick (see page 9668)

49.1.4.5.35 TCustomDBGrid.EndColumnDrag Method

Ends the dragging of a column in the grid.

Pascal

```
function EndColumnDrag(var Origin: Integer; var Destination: Integer; const MousePt: TPoint):
Boolean; override;
```

C++

```
virtual __fastcall Boolean EndColumnDrag(int Origin, int Destination, const TPoint MousePt);
```

Description

Call EndColumnDrag to end a drag operation. EndColumnDrag enables the grid to control where the column can be dropped. It is exposed as a protected method so that descendants of TCustomDBGrid (see page 9608) can customize column dragging in a grid.

See Also

BeginColumnDrag (see page 9625)

49.1.4.5.36 TCustomDBGrid.EndLayout Method

Decrements the LayoutLock (see page 9659) property and recomputes the Columns (see page 9653) property if LayoutLock (see page 9659) reaches 0.

Pascal

```
procedure EndLayout;
```

C++

```
__fastcall EndLayout();
```

Description

Applications cannot call this protected method. Data-aware grids call EndLayout internally after changes to the Columns (see page 9653) property or the dataset. EndLayout decrements the LayoutLock (see page 9659) property which was previously incremented by BeginLayout (see page 9626) before the Columns (see page 9653) property was changed or when notification of the change to the dataset was received.

If LayoutLock (see page 9659) returns to 0, EndLayout updates the Fields (see page 9657) property to match both the bindings of the Columns (see page 9653) property and the fields available in the dataset. The Columns (see page 9653) property is then updated in turn to match the Fields (see page 9657) property. This ensures that the Columns (see page 9653) property matches both the specified column bindings and the fields available in the dataset.

See Also

BeginLayout (see page 9626)

Columns (see page 9653)

DefineFieldMap (see page 9636)

Fields (see page 9657)

LayoutLock (see page 9659)

49.1.4.5.37 TCustomDBGrid.EndUpdate Method

Decrements the UpdateLock (see page 9664) property.

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

Applications cannot call this protected method. Data-aware grids call EndUpdate internally after making changes that affect the data that is drawn in the grid. EndUpdate returns the UpdateLock (see page 9664) to the state it was in before the BeginUpdate (see page 9627) method was called at the start of the changes.

While UpdateLock (see page 9664) is greater than 0, the grid does not repaint its cells. When UpdateLock (see page 9664) returns to 0, painting is reenabled. BeginUpdate (see page 9627) and EndUpdate prevent the grid from flickering when the grid is resized or the user scrolls through the data.

See Also

BeginUpdate (see page 9627)

UpdateLock (see page 9664)

49.1.4.5.38 TCustomDBGrid.ExecuteAction Method

Executes an action with the DB (see page 6725) grid as its target.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

ExecuteAction is called automatically when the user invokes an action with the DB (see page 6725) grid as its target. The Action parameter specifies the action that was invoked. ExecuteAction returns a boolean value indicating whether the action was successfully executed.

ExecuteAction first calls the inherited method, which checks whether the action knows how to execute with the grid as a target. If this does not successfully dispatch the action, ExecuteAction attempts to invoke the action with the grid's associated dataset as a target.

See Also

Execute (see page 2712)

DataSource (see page 9655)

49.1.4.5.39 TCustomDBGrid.GetColField Method

Returns the field component that represents the binding of a specified column.

Pascal

```
function GetColField(DataCol: Integer): TField;
```

C++

```
__fastcall TField GetColField(int DataCol);
```

Description

Call GetColField to obtain the field component for a column in the data-aware grid. The DataCol parameter is the index of the column in the Columns (see page 9653) property, where 0 is the first column, 1 is the second column, and so on.

Unlike the Fields (see page 9657) property, GetColField will return field components for columns that do not represent fields in the dataset, such as calculated fields. GetColField will return nil (Delphi) or NULL (C++) if no field is associated with the column or if the dataset is not active.

See Also

Columns (see page 9653)

Fields (see page 9657)

49.1.4.5.40 TCustomDBGrid.GetEditLimit Method

Limits the number of characters that can be typed into cells representing string fields.

Pascal

```
function GetEditLimit: Integer; override;
```

C++

```
virtual __fastcall int GetEditLimit();
```

Description

GetEditLimit is called by the inplace editor to determine what limit, if any, should be placed on the number of characters a user can type into a cell. GetEditLimit sets no limits except when the selected cell represents a string field.

See Also

CreateEditor (see page 9632)

49.1.4.5.41 TCustomDBGrid.GetEditMask Method

Returns the input mask associated with the field for the selected cell.

Pascal

```
function GetEditMask(ACol: Longint; ARow: Longint): string; override;
```

C++

```
virtual __fastcall AnsiString GetEditMask(Longint ACol, Longint ARow);
```

Description

The inplace editor calls GetEditMask to obtain any input mask associated with the field binding of the indicated cell. GetEditMask returns the EditMask property of the field component from the Columns (see page 9653) array.

The ACol and ARow parameters indicate the index of the row and column of the cell in the grid. The ACol parameter is not the same as the index of the column in the Columns (see page 9653) array if the Options (see page 9659) property includes dgIndicator.

See Also

Columns (see page 9653)

Options (see page 9659)

RawToDataColumn (see page 9648)

EditMask (see page 7163)

49.1.4.5.42 TCustomDBGrid.GetEditStyle Method

Indicates whether a specified cell has an ellipsis button or drop-down arrow.

Pascal

```
function GetEditStyle(ACol: Longint; ARow: Longint): TEditStyle; override;
```

C++

```
virtual __fastcall TEditStyle GetEditStyle(Longint ACol, Longint ARow);
```

Description

GetEditStyle indicates whether the specified cell can only be edited directly (esSimple), whether it has an ellipsis button that users can click to launch an editor for modifying field values (esEllipsis), or whether it has a drop-down arrow that users can click to display a pick list of possible values (esPickList).

ACol is the column that contains the cell. ARow is the row that contains the cell.

In TCustomDBGrid (see page 9608), GetEditStyle checks the properties of the column object that corresponds to ACol.

If the column has a ButtonStyle of cbsEllipsis or if it has a ButtonStyle of cbsAuto and it represents a dataset or reference field, then GetEditStyle returns esEllipsis.

If the column has a ButtonStyle of cbsAuto, is not read-only, and either has a pick list or represents a lookup field, then GetEditStyle returns esPickList.

If the column has a ButtonStyle of cbsNone or neither of the two previous conditions are met, then GetEditStyle returns esSimple.

See Also

ButtonStyle (see page 9593)

FieldName (see page 9595)

ReadOnly (see page 9599)

Columns (see page 9653)

PickList (see page 9598)

49.1.4.5.43 TCustomDBGrid.GetEditText Method

Returns the value of the indicated cell formatted for editing.

Pascal

```
function GetEditText(ACol: Longint; ARow: Longint): string; override;
```

C++

```
virtual __fastcall AnsiString GetEditText(Longint ACol, Longint ARow);
```

Description

The inplace editor calls GetEditText to obtain the text for the indicated cell. GetEditText locates the field component for the cell from the Columns (see page 9653) array and returns its value formatted for editing.

The ACol and ARow parameters indicate the index of the row and column of the cell in the grid. The ACol parameter is the "raw" column index, which is not the same as the index of the column in the Columns (see page 9653) array if the Options (see

page 9659) property includes dgIndicator.

Note: The string returned by GetEditText may differ from the string returned by the GetFieldValue (see page 9643) method if the field formats its value differently for editing and display.

See Also

Columns (see page 9653)

Options (see page 9659)

RawToDataColumn (see page 9648)

EditMask (see page 7163)

49.1.4.5.44 TCustomDBGrid.GetFieldValue Method

Returns the value of the indicated cell formatted for display.

Pascal

```
function GetFieldValue(ACol: Integer): string;
```

C++

```
__fastcall AnsiString GetFieldValue(int ACol);
```

Description

The DefaultDrawColumnCell (see page 9634) method calls GetFieldValue to obtain the text that should be displayed in a cell. Applications do not call this protected method.

The ACol parameter is the index of the column in the columns array. If the Options (see page 9659) property includes dgIndicator, this is not the same as the index of the column in the physical display.

See Also

Columns (see page 9653)

Options

DisplayText (see page 7162)

DefaultDrawColumnCell (see page 9634)

49.1.4.5.45 TCustomDBGrid.HighlightCell Method

Indicates whether a specified cell should be highlighted when it is drawn.

Pascal

```
function HighlightCell(DataCol: Integer; DataRow: Integer; const Value: string; AState:
TGridDrawState): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean HighlightCell(int DataCol, int DataRow, const AnsiString Value,
TGridDrawState AState);
```

Description

Applications cannot call this protected method. It is called internally by the DrawCell (see page 9637) method to determine whether the cell specified by the DataCol and DataRow parameters should be highlighted. HighlightCell returns true if the cell has focus, the cell is selected and Options (see page 9659) includes dgAlwaysShowSelection, or the cell is in a selected row and Options (see page 9659) includes dgRowSelect.

See Also

DrawCell (see page 9637)

Options (see page 9659)

49.1.4.5.46 TCustomDBGrid.InvalidateTitles Method

Invalidates the column titles so that they will be repainted.

Pascal

```
procedure InvalidateTitles;
```

C++

```
__fastcall InvalidateTitles();
```

Description

Call InvalidateTitles to cause the column titles to be repainted.

49.1.4.5.47 TCustomDBGrid.KeyDown Method

Provides special processing when the user presses down on a key.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

Applications cannot call KeyDown. It is called automatically when the user presses a key, before the KeyPress (see page 9645) method is called.

KeyDown generates an OnKeyDown event and then responds to the following keystrokes:

Keystroke	Response
UpArrow	Move to the previous row. If Options (see page 9659) includes dgRowSelect, select it.
Ctrl+UpArrow	Move to the first row in the grid. Deselect any selected rows.
DownArrow	Move to the next row. If Options (see page 9659) includes dgRowSelect, select it.
Ctrl+DownArrow	Move to the last row in the grid. Deselect any selected rows.
PageUp	Move up by the number of records shown in the grid. Deselect any selected rows.
Ctrl+PageUp	Move to the first row in the grid. Deselect any selected rows.
PageDown	Move down by the number of records shown in the grid. Deselect any selected rows.
LeftArrow	Move left one column. If Options (see page 9659) includes dgRowSelect, move to the previous row without selecting it.
Ctrl+LeftArrow	Move to the first column in the current row.
RightArrow	Move right one column. If Options (see page 9659) includes dgRowSelect, move to the next row without selecting it.
Ctrl+RightArrow	Move to the last column in the current row.

Home	Move to the first column of the current row. If Options (see page 9659) includes dgRowSelect or the grid has only one column, move to the first record in the dataset. Deselect any selected rows.
Ctrl+Home	Move to the first record in the dataset. Deselect any selected rows.
End	Move to the last column of the current row. If Options (see page 9659) includes dgRowSelect or the grid has only one column, move to the last record in the dataset. Deselect any selected rows.
Ctrl+End	Move to the last record in the dataset. Deselect any selected rows.
Tab	Move to the next cell in the grid.
Shift+Tab	Move to the previous cell in the grid.
Ctrl+Delete	Delete the current selection.
Insert	Insert a new row above the current one, and move to it. Deselect any selected rows.
Escape	Cancel any unposted edits. Deselect any selected rows.
F2	Put the grid in edit mode.

See Also

KeyPress (see page 9645)

Options (see page 9659)

OnKeyDown

49.1.4.5.48 TCustomDBGrid.KeyPress Method

Responds when the user presses a key.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

When the user presses a key, first the KeyDown (see page 9644) method is called, then the KeyPress method, and finally the KeyUp method when the user releases the key. KeyPress overrides the inherited method to write edits to the dataset when the user presses Return.

See Also

KeyDown (see page 9644)

KeyPress (see page 14833)

KeyUp (see page 6164)

49.1.4.5.49 TCustomDBGrid.LayoutChanged Method

Recomputes the Columns (see page 9653) property.

Pascal

```
procedure LayoutChanged; virtual;
```

C++

```
virtual __fastcall LayoutChanged();
```

Description

Applications cannot call this protected method. It is called automatically in response to changes the affect the layout of the columns. These include

Changes to the fields in the dataset.

Changes in the Options (see page 9659) property that affect the layout.

Changes to the TitleFont (see page 9663) property.

LayoutChanged calls AcquireLayoutLock (see page 9625) to initiate the changes to Columns (see page 9653). If AcquireLayoutLock (see page 9625) does not indicate that the call to LayoutChanged is recursive, LayoutChanged calls EndLayout (see page 9639) to perform the actual computation.

See Also

AcquireLayoutLock (see page 9625)

Columns (see page 9653)

EndLayout (see page 9639)

Options (see page 9659)

TitleFont (see page 9663)

49.1.4.5.50 TCustomDBGrid.LinkActive Method

Recomputes the Columns (see page 9653) property and the vertical scroll bar of the grid.

Pascal

```
procedure LinkActive(Value: Boolean); virtual;
```

C++

```
virtual __fastcall LinkActive(Boolean Value);
```

Description

Applications cannot call this protected method. It is called automatically when the dataset is opened. LinkActive calls LayoutChanged (see page 9645) to recompute the Columns (see page 9653) property, and adjusts the vertical scroll bar for the data in the dataset.

See Also

Columns (see page 9653)

LayoutChanged (see page 9645)

49.1.4.5.51 TCustomDBGrid.Loaded Method

Sets up the Columns (see page 9653) property once the data grid and all its child components are loaded.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Applications cannot call this protected method. Loaded overrides the inherited method to lay out the grid once all of its parts

have been loaded.

See Also

Columns (see page 9653)

49.1.4.5.52 TCustomDBGrid.MouseDown Method

Provides special processing when the user clicks on the mouse.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

MouseDown is called automatically when the user presses the mouse button. If the mouse is pressed in the row or column resize region of the grid, MouseDown posts any pending edits and calls the inherited method to handle the resize. When Options (see page 9659) includes dgRowSelect, MouseDown selects the row that received the click, or toggles its selection state if the Ctrl key is pressed. If the mouse is clicked in a data cell, MouseDown tries to put the grid into edit mode.

See Also

MouseUp (see page 9647)

Options (see page 9659)

MouseDown (see page 14834)

49.1.4.5.53 TCustomDBGrid.MouseUp Method

Provides special processing when the user releases the mouse button.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

MouseUp is called automatically when the user releases the mouse button. After calling the inherited method, MouseUp locates the cell that was under the mouse when it was released. MouseUp generates an OnTitleClick (see page 9668) or OnCellClick (see page 9664) event, depending on the location of the mouse.

See Also

OnCellClick (see page 9664)

OnTitleClick (see page 9668)

MouseUp (see page 14835)

49.1.4.5.54 TCustomDBGrid.Notification Method

Responds to notifications that components are about to be destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

After calling the inherited method, Notification

Checks if the pop-up menu for any of its columns is going away, and if so, resets the PopupMenu property of the column object.

Checks if the data source is going away, and if so, sets the DataSource (see page 9655) property to nil (Delphi) or NULL (C++).

Checks if any of the field components associated with its columns is going away, and if so, removes the column from the layout.

See Also

Columns (see page 9653)

DataSource (see page 9655)

PopupMenu (see page 9599)

49.1.4.5.55 TCustomDBGrid.RawToDataColumn Method

Converts the index of a column in the visual display to the corresponding index in the Columns (see page 9653) property.

Pascal

```
function RawToDataColumn(ACol: Integer): Integer;
```

C++

```
__fastcall int RawToDataColumn(int ACol);
```

Description

Applications cannot call this protected method. It is used internally to allow the data-aware grid to adjust to the presence or absence of the column that contains the row indicator.

See Also

Columns (see page 9653)

DataToRawColumn (see page 9633)

IndicatorOffset (see page 9658)

49.1.4.5.56 TCustomDBGrid.Scroll Method

Updates the image of the grid to reflect a new record position.

Pascal

```
procedure Scroll(Distance: Integer); virtual;
```

C++

```
virtual __fastcall Scroll(int Distance);
```

Description

Scroll is called in response to changes in the current record in the dataset. Scroll updates the image in the grid, scrolling the data if necessary.

See Also

TimedScroll (see page 9650)

49.1.4.5.57 TCustomDBGrid.SetColumnAttributes Method

Pascal

```
procedure SetColumnAttributes; virtual;
```

C++

```
virtual __fastcall SetColumnAttributes();
```

Description

This is SetColumnAttributes, a member of class TCustomDBGrid.

49.1.4.5.58 TCustomDBGrid.SetEditText Method

Updates the edited version of the current cell's data.

Pascal

```
procedure SetEditText(ACol: Longint; ARow: Longint; const Value: string); override;
```

C++

```
virtual __fastcall SetEditText(Longint ACol, Longint ARow, const AnsiString Value);
```

Description

The inplace editor calls SetEditText so that the grid can store the new value for the current cell. The grid stores this value until the user moves to a new cell or presses Escape. If the user moves to a new cell, this value is written to the record in the database. If the user presses Escape, the value is discarded.

See Also

GetEditText (see page 9642)

49.1.4.5.59 TCustomDBGrid.ShowPopupEditor Method

Displays the nested dataset associated with Column in a new window.

Pascal

```
procedure ShowPopupEditor(Column: TColumn; X: Integer = Low(Integer); Y: Integer =
Low(Integer)); dynamic;
```

C++

```
__fastcall ShowPopupEditor(TColumn Column, int X = Low(Integer), int Y = Low(Integer));
```

Description

Call ShowPopupEditor to display a nested dataset or ADT in a new grid in a new window. Column is the column that is representing the dataset or reference field of the dataset.

For example, if the seventh column in the grid represents a dataset field, the following code displays the dataset associated with that field for the current record.

```
DBGrid1.ShowPopupEditor(DBGrid1.Columns[7]);
DBGrid1->ShowPopupEditor(DBGrid1->Columns->Items[7]);
```

ShowPopupEditor expects the TColumn (see page 9583) object field type passed to be ftDataSet.

When the ellipsis button in the grid cell of a nested dataset or reference field is clicked, EditButtonClick (see page 9638) is called. If the grid's OnEditButtonClick (see page 9668) event has an event handler, that event is fired and ShowPopupEditor is not called. If the event handler is not assigned, ShowPopupEditor is called.

The X and Y parameters are screen coordinates to position the popup editor form. The default behavior is to position the form below the cell in the grid.

See Also

TDataSetField (see page 7060)

TReferenceField (see page 7452)

OnEditButtonClick (see page 9668)

EditButtonClick (see page 9638)

Working with DataSet Fields

Working with Reference Fields

49.1.4.5.60 TCustomDBGrid.StoreColumns Method

Indicates whether the columns of the grid are stored when the grid is saved, or whether they are generated dynamically.

Pascal

```
function StoreColumns: Boolean;
```

C++

```
__fastcall Boolean StoreColumns();
```

Description

StoreColumns causes the Columns (see page 9653) property to be stored only when the Columns (see page 9653) property has a State property of csCustomized. The State property for Columns (see page 9653) indicates whether the columns are generated automatically from the fields in the dataset, or if they are specified programmatically or by the Columns (see page 9653) editor.

See Also

Columns (see page 9653)

State (see page 9753)

49.1.4.5.61 TCustomDBGrid.TimedScroll Method

Moves the current record in response to the user dragging the mouse outside of the data cells.

Pascal

```
procedure TimedScroll(Direction: TGridScrollDirection); override;
```

C++

```
virtual __fastcall TimedScroll(TGridScrollDirection Direction);
```

Description

If the Direction parameter includes sdUp, TimedScroll moves the dataset to the first visible record in the grid. If the Direction parameter includes sdDown, TimedScroll moves the dataset to the last visible record in the grid. After adjusting the current record, TimedScroll calls the inherited method to update the image.

See Also

TimedScroll (see page 14840)

49.1.4.5.62 TCustomDBGrid.TitleClick Method

Generates an OnTitleClick (see page 9668) event.

Pascal

```
procedure TitleClick(Column: TColumn); dynamic;
```

C++

```
__fastcall TitleClick(TColumn Column);
```

Description

Applications cannot call TitleClick directly. It is exposed as a protected method so that descendants can call the OnTitleClick (see page 9668) event handler or override the method to block the event.

See Also

OnTitleClick (see page 9668)

49.1.4.5.63 TCustomDBGrid.TopLeftChanged Method

Updates the position of the inplace editor when the grid scrolls.

Pascal

```
procedure TopLeftChanged; override;
```

C++

```
virtual __fastcall TopLeftChanged();
```

Description

TopLeftChanged is called whenever the LeftCol or TopRow property changes. InvalidateTitles (see page 9644) and the inherited method TopLeftChanged method are called.

See Also

TopLeftChanged (see page 14841)

InvalidateTitles (see page 9644)

49.1.4.5.64 TCustomDBGrid.UpdateAction Method

Updates an action to reflect the current state of the grid.

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; override;
```

C++

```
virtual __fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

UpdateAction is called automatically when the application is idle allow the grid to set action properties to reflect the current state of the grid. The Action parameter is the action to be updated. UpdateAction returns a boolean that indicates whether the action was updated by the grid.

As implemented in TCustomDBGrid (see page 9608), UpdateAction checks whether the action relies on the state of the associated dataset, and if so, updates the action accordingly.

See Also

UpdateAction (see page 6887)

DataSource (see page 9655)

Update (see page 2714)

49.1.4.5.65 TCustomDBGrid.UpdateScrollBar Method

Pascal

```
procedure UpdateScrollBar; virtual;
```

C++

```
virtual __fastcall UpdateScrollBar();
```

Description

This is UpdateScrollBar, a member of class TCustomDBGrid.

49.1.4.5.66 TCustomDBGrid.UseRightToLeftAlignmentForField Method

Indicates whether the alignment of a specified field should be changed from left-aligned to right-aligned.

Pascal

```
function UseRightToLeftAlignmentForField(const AField: TField; Alignment: TAlignment): Boolean;
```

C++

```
__fastcall Boolean UseRightToLeftAlignmentForField(const TField AField, TAlignment Alignment);
```

Description

UseRightToLeftAlignmentForField is called internally when drawing field values to determine whether the Alignment of AField should be changed from left-aligned to right-aligned. This method lets the data-aware grid honor the current setting of the BiDiMode property. UseRightToLeftAlignment returns true if the application is running in a mideast locale (where alignment should be reversed), the field contains text (as opposed to numeric or date/time values), and the specified alignment is not taCenter.

See Also

IsRightToLeft (see page 5770)

FieldKind (see page 7164)

BiDiMode (see page 5803)

49.1.4.5.67 TCustomDBGrid.ValidFieldIndex Method

Indicates whether an index into the columns array refers to a valid field in the dataset.

Pascal

```
function ValidFieldIndex(FieldIndex: Integer): Boolean;
```

C++

```
__fastcall Boolean ValidFieldIndex(int FieldIndex);
```

Description

Call ValidFieldIndex to determine whether a column displays data from a field in the dataset. ValidFieldIndex returns true if the column is associated with a database field. ValidFieldIndex returns false if the column is a calculated field or if the field for the column is undefined.

See Also

Columns (see page 9653)

Fields (see page 9657)

GetColField (see page 9640)

49.1.4.6 TCustomDBGrid Properties

The properties of the TCustomDBGrid class are listed here.

Protected Properties

	Name	Description
	Columns (see page 9653)	Describes the display attributes and field bindings of the columns.
R	DataLink (see page 9654)	Helps the data-aware grid manage its link to the data source and respond to data events.
	DefaultDrawing (see page 9655)	Indicates whether the cells in the data-aware grid are drawn automatically.
R	IndicatorOffset (see page 9658)	Indicates the index of the first column in the grid that contains data.
R	LayoutLock (see page 9659)	Counts the number of times BeginLayout (see page 9626) has been called without an EndLayout (see page 9639) or CancelLayout (see page 9628) call.
	Options (see page 9659)	Specifies various display and behavioral properties of the data-aware grid.
	ParentColor (see page 9660)	Determines where a control looks for its color information.
	ReadOnly (see page 9660)	Indicates whether the grid is used for display only, or whether the user can edit data using the grid.
R	SelectedRows (see page 9662)	Specifies a set of bookmarks for all the records in the dataset that correspond to rows selected in the grid.
	TitleFont (see page 9663)	Describes the font used to draw the column titles in the grid.
R	UpdateLock (see page 9664)	Counts the number of times BeginUpdate (see page 9627) is called without a corresponding call to EndUpdate (see page 9640).

Public Properties

	Name	Description
	DataSource (see page 9655)	Identifies the link to the dataset where the data-aware grid finds its data.
	EditorMode (see page 9656)	Determines whether the current cell can be edited.
R	FieldCount (see page 9656)	Specifies the number of fields (columns) displayed in the grid.
R	Fields (see page 9657)	Provides indexed access to the field components that correspond to the columns.
	SelectedField (see page 9661)	Specifies the field component for the currently selected cell in the grid.
	SelectedIndex (see page 9661)	Specifies the index of the currently selected column in the Columns (see page 9653) array.

49.1.4.6.1 TCustomDBGrid.Columns Property

Describes the display attributes and field bindings of the columns.

Pascal

```
property Columns: TDBGridColumns;
```

C++

```
__property TDBGridColumns Columns;
```

Description

Use Columns to read or set the field bindings and display attributes of the columns in the data-aware grid. The Columns property is an indexed collection of TColumn (see page 9583) objects. Use the properties of the TColumn (see page 9583) objects to specify the display attributes or field bindings of individual columns in the grid. The field binding of a column designates a field within the dataset specified by the DataSource (see page 9655) property.

Columns can be set at design time through the Columns editor, or programmatically at runtime.

See Also

CreateColumns (see page 9632)

Fields (see page 9657)

IndicatorOffset (see page 9658)

setcolumnattributes (see page 9649)

StoreColumns (see page 9650)

TDBGridColumns (see page 9746)

TColumn (see page 9583)

Creating Persistent Columns

Viewing and Editing Data with TDBGrid

49.1.4.6.2 TCustomDBGrid.DataLink Property

Helps the data-aware grid manage its link to the data source and respond to data events.

Pascal

```
property DataLink: TGridDataLink;
```

C++

```
__property TGridDataLink DataLink;
```

Description

Applications should not need to use the DataLink property. It is exposed as a protected property so that component writers can use descendants of TGridDataLink (see page 9754) to respond to additional data events.

DataLink helps TCustomDBGrid (see page 9608) to

Resolve the field bindings specified by Columns (see page 9653) with the fields in the dataset.

Respond to changes in the values of fields.

Respond to changes in record currency.

Respond to changes in the editing state of the dataset.

Keep track of unposted edits and discard or post them as appropriate.

See Also

Columns (see page 9653)

DataSource (see page 9655)

Fields (see page 9657)

TGridDataLink (see page 9754)

49.1.4.6.3 TCustomDBGrid.DataSource Property

Identifies the link to the dataset where the data-aware grid finds its data.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Set DataSource to the TDataSource object that links to the dataset where the data-aware grid should fetch its data. DataSource allows the data-aware grid to read from, write to, and navigate around the dataset.

Delphi Examples:

```
{
The following example copies the selected rows in a db grid
to a list box.  Set the db grid Options dgRowSelect,
dgAlwaysShowSelection and dgMultiSelect to True.  Make a
multiple selecton using the CNTL key.  This example requires
a TDataSet associated with a TDataSource and a TDBGrid.}
procedure TForm1.Button2Click(Sender: TObject);
var
  i, j: Integer;
  s: string;
begin
  if DBGrid2.SelectedRows.Count>0 then
    with DBGrid2.DataSource.DataSet do
      for i:=0 to DBGrid2.SelectedRows.Count-1 do
      begin
        GotoBookmark(pointer(DBGrid2.SelectedRows.Items[i]));
        for j := 0 to FieldCount-1 do
        begin
          if (j>0) then s:=s+', ';
          s:=s+Fields[j].AsString;
        end;
        Listbox1.Items.Add(s);
        s:= '';
      end;
end;
```

See Also

DataLink (see page 9654)

Fields (see page 9657)

TDataSource (see page 7070)

Making the Table a Detail of Another Dataset

49.1.4.6.4 TCustomDBGrid.DefaultDrawing Property

Indicates whether the cells in the data-aware grid are drawn automatically.

Pascal

```
property DefaultDrawing: Boolean;
```

C++

```
__property Boolean DefaultDrawing;
```

Description

Set DefaultDrawing to true to allow the data-aware grid to draw the data in the cells of the grid automatically. Set DefaultDrawing to false to turn off the default drawing when providing customized painting in an OnDrawColumnCell (see page 9666) event handler or an OnDrawDataCell (see page 9667) event handler.

When DefaultDrawing is true, the data-aware grid fetches the data for each cell from the appropriate entry in the Fields (see page 9657) array, and writes it to the cell. It then draws the focus rectangle if the cell has focus, or if the cell is selected and the Options (see page 9659) property includes dgAlwaysShowSelection.

When DefaultDrawing is false, the data-aware grid draws the appropriate background color on the cell and sets up the brush and font described by the appropriate TColumn (see page 9583) object. However, the value of the field is not written to the cell, and no focus rectangle is drawn.

See Also

Columns (see page 9653)

Fields (see page 9657)

OnDrawColumnCell (see page 9666)

OnDrawDataCell (see page 9667)

TColumn (see page 9583)

Controlling Grid Drawing

49.1.4.6.5 TCustomDBGrid.EditorMode Property

Determines whether the current cell can be edited.

Pascal

```
property EditorMode: Boolean;
```

C++

```
__property Boolean EditorMode;
```

Description

Set EditorMode to true, at runtime, to put the grid in edit mode. When EditorMode is true, the user can edit cells in the grid. When the user presses F2, EditorMode is set to true. When the user presses Enter, the value of EditorMode is toggled.

Note: EditorMode has no effect if the Options property does not include goEditing or if the Options property includes goAlwaysShowEditor.

See Also

HideEditor (see page 14830)

Options (see page 14853)

ShowEditor (see page 14839)

49.1.4.6.6 TCustomDBGrid.FieldCount Property

Specifies the number of fields (columns) displayed in the grid.

Pascal

```
property FieldCount: Integer;
```

C++

```
__property int FieldCount;
```

Description

Use FieldCount to iterate through all the field components indexed by the Fields (see page 9657) property. FieldCount may differ from the number of fields in the underlying dataset because the grid may contain calculated fields, and not all fields in the dataset are necessarily shown in the grid. Individual fields are removed and calculated fields are added using the Columns (see page 9653) editor at design time.

FieldCount is the same as the number of columns at runtime. At design time, the grid may contain empty columns.

Delphi Examples:

```
{
The following example copies the selected rows in a db grid
to a list box.  Set the db grid Options dgRowSelect,
dgAlwaysShowSelection and dgMultiSelect to True.  Make a
multiple selecton using the CNTL key.  This example requires
a TDataSet associated with a TDataSource and a TDBGrid.}
procedure TForm1.Button2Click(Sender: TObject);
var
  i, j: Integer;
  s: string;
begin
  if DBGrid2.SelectedRows.Count>0 then
    with DBGrid2.DataSource.DataSet do
      for i:=0 to DBGrid2.SelectedRows.Count-1 do
      begin
        GotoBookmark(pointer(DBGrid2.SelectedRows.Items[i]));
        for j := 0 to FieldCount-1 do
        begin
          if (j>0) then s:=s+', ';
          s:=s+Fields[j].AsString;
        end;
        Listbox1.Items.Add(s);
        s:= '';
      end;
end;
```

See Also

Columns (see page 9653)

Fields (see page 9657)

Creating a Customized Grid

49.1.4.6.7 TCustomDBGrid.Fields Property

Provides indexed access to the field components that correspond to the columns.

Pascal

```
property Fields [FieldIndex: Integer]: TField;
```

C++

```
__property TField Fields[int FieldIndex];
```

Description

Use Fields to gain direct access to the field component in the dataset for a particular column in the grid. The field for the first data column is obtained using an Index of 0, the field for the second data column using an Index of 1, and so on. The range for Index is 0... FieldCount (see page 9656) - 1.

The first data column in the grid may not be the same as the first column in the grid. If Options (see page 9659) includes dgIndicator, the first column in the grid is a non-scrolling column that contains the current row indicator.

The individual field components in Fields can be used to obtain information such as the value of the field on the current record, its preferred display width, or the string to draw when displaying or editing the field. Use the field component to write values directly to the dataset without using the editor in the cell.

Note: If the column in the grid represents a calculated field, there is no field component for that column in the dataset. Fields returns nil (Delphi) or NULL (C++) when the Index specifies the column that contains a calculated field. To obtain the field component for a calculated field, use the GetColField (see page 9640) method.

Delphi Examples:

```
{
The following example copies the selected rows in a db grid
to a list box.  Set the db grid Options dgRowSelect,
dgAlwaysShowSelection and dgMultiSelect to True.  Make a
multiple selecton using the CNTL key.  This example requires
a TDataSet associated with a TDataSource and a TDBGrid.}
procedure TForm1.Button2Click(Sender: TObject);
var
  i, j: Integer;
  s: string;
begin
  if DBGrid2.SelectedRows.Count>0 then
    with DBGrid2.DataSource.DataSet do
      for i:=0 to DBGrid2.SelectedRows.Count-1 do
      begin
        GotoBookmark(pointer(DBGrid2.SelectedRows.Items[i]));
        for j := 0 to FieldCount-1 do
        begin
          if (j>0) then s:=s+', ';
          s:=s+Fields[j].AsString;
        end;
        Listbox1.Items.Add(s);
        s:= '';
      end;
end;
```

See Also

Columns (see page 9653)

GetColField (see page 9640)

Options (see page 9659)

SelectedField (see page 9661)

TField (see page 7115)

49.1.4.6.8 TCustomDBGrid.IndicatorOffset Property

Indicates the index of the first column in the grid that contains data.

Pascal

```
property IndicatorOffset: Byte;
```

C++

```
__property Byte IndicatorOffset;
```

Description

Use IndicatorOffset to convert between the positions of the columns in the Columns (see page 9653) property and the columns drawn on the data-aware grid. IndicatorOffset is 1 if the Options (see page 9659) property includes dgIndicator, because the first column of the grid is used for the current row indicator. IndicatorOffset is 0 if all columns contain data.

See Also

Columns (see page 9653)

DataToRawColumn (see page 9633)

Options (see page 9659)

RawToDataColumn (see page 9648)

49.1.4.6.9 TCustomDBGrid.LayoutLock Property

Counts the number of times BeginLayout (see page 9626) has been called without an EndLayout (see page 9639) or CancelLayout (see page 9628) call.

Pascal

```
property LayoutLock: Byte;
```

C++

```
__property Byte LayoutLock;
```

Description

Use LayoutLock to determine when the column bindings change. Whenever a column width or binding changes or the fields in the data source change, the BeginLayout (see page 9626) method is called. BeginLayout (see page 9626) increments LayoutLock to keep track of the fact that the column bindings are changing. When changes are complete, the EndLayout (see page 9639) method is called. EndLayout (see page 9639) decrements LayoutLock. When LayoutLock returns to 0, the data-aware grid knows that all changes are complete, and the Fields (see page 9657) and Columns (see page 9653) properties are updated.

See Also

AcquireLayoutLock (see page 9625)

BeginLayout (see page 9626)

CancelLayout (see page 9628)

DeferLayout (see page 9635)

DefineFieldMap (see page 9636)

EndLayout (see page 9639)

49.1.4.6.10 TCustomDBGrid.Options Property

Specifies various display and behavioral properties of the data-aware grid.

Pascal

```
property Options: TDBGridOptions;
```

C++

```
__property TDBGridOptions Options;
```

Description

Set Options to include the desired properties for the data-aware grid.

See Also

ReadOnly (see page 9660)

SelectedIndex (see page 9661)

SelectedRows (see page 9662)

Setting Grid Options

49.1.4.6.11 TCustomDBGrid.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

49.1.4.6.12 TCustomDBGrid.ReadOnly Property

Indicates whether the grid is used for display only, or whether the user can edit data using the grid.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set ReadOnly to true to prevent users from changing the data in the dataset. Set ReadOnly to false to allow users to edit data using the grid.

When ReadOnly is true, users cannot use the Insert key to insert a new row in the grid or use the Down Arrow key to add a row to the end of the grid.

See Also

CanEditModify (see page 9629)

Options (see page 9659)

Editing in the Grid

49.1.4.6.13 TCustomDBGrid.SelectedField Property

Specifies the field component for the currently selected cell in the grid.

Pascal

```
property SelectedField: TField;
```

C++

```
__property TField SelectedField;
```

Description

Set SelectedField to move focus to a particular field in the grid. Read SelectedField to obtain access to the field component for the currently selected cell. If there is not currently selected cell, SelectedField is nil (Delphi) or NULL (C++). For example, when an entire row is selected, SelectedField is nil or NULL.

See Also

Fields (see page 9657)

SelectedIndex (see page 9661)

Creating a Customized Grid

Responding to User Actions at Runtime

49.1.4.6.14 TCustomDBGrid.SelectedIndex Property

Specifies the index of the currently selected column in the Columns (see page 9653) array.

Pascal

```
property SelectedIndex: Integer;
```

C++

```
__property int SelectedIndex;
```

Description

Set SelectedIndex to move focus to a column in the grid that is identified by position. Read SelectedIndex to determine which column in the grid has focus. A value of 0 indicates the first data column, 1 is the second data column, and so on. SelectedIndex is -1 if there is no currently selected column.

If the Options (see page 9659) property includes dgIndicator, the index of the data column given by SelectedIndex will differ from the index of the physical column in the grid.

To access the field component for the selected column, use the SelectedField (see page 9661) property.

C++ Examples:

```
/*
The following example enables the user to move the current
selected cell in a db grid.  The Up and Down buttons have
their OnClick events assigned to the UpDownClick procedure.
```

```
The Left and Right buttons have their OnClick events
assigned to the LeftRightClick procedure.  The Up and Left
buttons have their Tag property set to -1, while the Down
and Right buttons have their Tag property set to 1.  MoveBy
does not move the cell in the db grid, it only moves the
selection.  The data set is not edited.  Notice that the
Up button and the Down button are separate TButtons, not
one TUpDown.  The tags must be set different.
*/
void __fastcall TForm1::UpDownClick(TObject *Sender)
{
  CDS2->MoveBy(((TComponent *)Sender)->Tag);
  DBGrid2->SetFocus();
}

void __fastcall TForm1::LeftRightClick(TObject *Sender)
{
  DBGrid2->SelectedIndex = DBGrid2->SelectedIndex + ((TComponent *)Sender)->Tag;
  DBGrid2->SetFocus();
}
```

Delphi Examples:

```
{
The following example enables the user to move the current
selected cell in a db grid.  The Up and Down buttons have
their OnClick events assigned to the UpDownClick procedure.
The Left and Right buttons have their OnClick events
assigned to the LeftRightClick procedure.  The Up and Left
buttons have their Tag property set to -1, while the Down
and Right buttons have their Tag property set to 1.  MoveBy
does not move the cell in the db grid, it only moves the
selection.  The data set is not edited.  Notice that the
Up button and the Down button are separate TButtons, not
one TUpDown.  The tags must be set different.
}
procedure TForm1.UpDownClick(Sender: TObject);
begin
  CDS.MoveBy(TComponent(Sender).Tag);
  DBGrid1.SetFocus;
end;

procedure TForm1.LeftRightClick(Sender: TObject);
begin
  DBGrid1.SelectedIndex := DBGrid1.SelectedIndex + TComponent(Sender).Tag;
  DBGrid1.SetFocus;
end;
```

See Also

Columns (see page 9653)

Options (see page 9659)

SelectedField (see page 9661)

Creating a Customized Grid

49.1.4.6.15 TCustomDBGrid.SelectedRows Property

Specifies a set of bookmarks for all the records in the dataset that correspond to rows selected in the grid.

Pascal

```
property SelectedRows: TBookmarkList;
```

C++

```
__property TBookmarkList SelectedRows;
```

Description

Use the properties and methods of the TBookmarkList (see page 9575) object returned by SelectedRows to

Determine the number of rows in the grid that are selected.

Determine whether the current record in the dataset is selected.

Determine whether a particular record in the dataset is selected.

Delete all selected rows from the dataset.

SelectedRows is only meaningful when the Options (see page 9659) property includes dgRowSelect and dgMultiSelect.

Delphi Examples:

```
{
The following example copies the selected rows in a db grid
to a list box.  Set the db grid Options dgRowSelect,
dgAlwaysShowSelection and dgMultiSelect to True.  Make a
multiple selecton using the CNTL key.  This example requires
a TDataSet associated with a TDataSource and a TDBGrid.}
procedure TForm1.Button2Click(Sender: TObject);
var
  i, j: Integer;
  s: string;
begin
  if DBGrid2.SelectedRows.Count>0 then
    with DBGrid2.DataSource.DataSet do
      for i:=0 to DBGrid2.SelectedRows.Count-1 do
      begin
        GotoBookmark(pointer(DBGrid2.SelectedRows.Items[i]));
        for j := 0 to FieldCount-1 do
        begin
          if (j>0) then s:=s+', ';
          s:=s+Fields[j].AsString;
        end;
        Listbox1.Items.Add(s);
        s:= '';
      end;
end;
```

See Also

Options (see page 9659)

TBookmarkList (see page 9575)

49.1.4.6.16 TCustomDBGrid.TitleFont Property

Describes the font used to draw the column titles in the grid.

Pascal

```
property TitleFont: TFont;
```

C++

```
__property TFont TitleFont;
```

Description

Set TitleFont to change the font used to draw the column headings. TitleFont is only meaningful if the Options (see page 9659) property includes dgTitles.

See Also

Options (see page 9659)

TFont (see page 14497)

49.1.4.6.17 TCustomDBGrid.UpdateLock Property

Counts the number of times BeginUpdate (see page 9627) is called without a corresponding call to EndUpdate (see page 9640).

Pascal

```
property UpdateLock: Byte;
```

C++

```
__property Byte UpdateLock;
```

Description

UpdateLock prevents the grid from repainting while the data in the grid changes. Whenever a change is made that affects the data that is drawn in the grid, the BeginUpdate (see page 9627) method is called. BeginUpdate (see page 9627) increments UpdateLock to keep track of the fact that the contents of the grid are changing. When changes are complete, the EndUpdate (see page 9640) method is called. EndUpdate (see page 9640) decrements UpdateLock.

When UpdateLock returns to 0, the data-aware grid can repaint its cells. When UpdateLock is greater than 0, the grid defers repaints. UpdateLock prevents the grid from flickering when the grid is resized or the user scrolls through the data.

See Also

BeginUpdate (see page 9627)

EndUpdate (see page 9640)

LayoutLock (see page 9659)

49.1.4.7 TCustomDBGrid Events

The events of the TCustomDBGrid class are listed here.

Protected Events

	Name	Description
	OnCellClick (see page 9664)	Occurs when the user releases the mouse in one of the cells of the grid.
	OnColEnter (see page 9665)	Occurs when focus moves to a new cell in the grid.
	OnColExit (see page 9665)	Occurs just before a cell in the grid loses input focus.
	OnColumnMoved (see page 9666)	Occurs when the user moves a column using the mouse.
	OnDrawColumnCell (see page 9666)	Occurs when the grid needs to paint a cell.
	OnDrawDataCell (see page 9667)	Occurs when the grid needs to paint a cell if the State property of Columns (see page 9653) is csDefault.
	OnEditButtonClick (see page 9668)	Occurs when the user presses the ellipsis button in a grid column.
	OnTitleClick (see page 9668)	Occurs when the user releases the mouse in one of the column headers.

49.1.4.7.1 TCustomDBGrid.OnCellClick Event

Occurs when the user releases the mouse in one of the cells of the grid.

Pascal

```
property OnCellClick: TDBGridClickEvent;
```

C++

```
__property TDBGridClickEvent OnCellClick;
```

Description

Write an OnCellClick event handler to take specific action when the user clicks in one of the cells of the data-aware grid.

OnCellClick occurs when the user releases the mouse. To take action when the user clicks down on the mouse, use the OnColEnter (see page 9665) event.

OnCellClick will not occur when the mouse is released at the end of resizing a row or column, or when the cell is being edited.

See Also

Columns (see page 9653)

OnColEnter (see page 9665)

OnTitleClick (see page 9668)

TColumn (see page 9583)

49.1.4.7.2 TCustomDBGrid.OnColEnter Event

Occurs when focus moves to a new cell in the grid.

Pascal

```
property OnColEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnColEnter;
```

Description

Write an OnColEnter event handler to take specific action when a new cell has just been selected. Focus moves to a cell when

The user navigates to the cell using the keyboard. For example, when the user uses the Tab key, or the Home key.

The user clicks the mouse button down in the cell.

The SelectedField (see page 9661) or SelectedIndex (see page 9661) property is set.

Read the SelectedField (see page 9661) or SelectedIndex (see page 9661) property to determine which cell was just entered.

See Also

OnCellClick (see page 9664)

OnColExit (see page 9665)

SelectedField (see page 9661)

SelectedIndex (see page 9661)

49.1.4.7.3 TCustomDBGrid.OnColExit Event

Occurs just before a cell in the grid loses input focus.

Pascal

```
property OnColExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnColExit;
```

Description

Write an OnColExit event handler to take specific action when a cell is about to lose focus. For example, use OnColExit to post edits to the underlying dataset when the user leaves a cell, rather than waiting for the user to leave the record.

A cell loses focus when

The user navigates out of the cell using the keyboard. For example, when the user presses the Tab key or the Home key.

The user clicks the mouse button down in another cell.

The SelectedField (see page 9661) or SelectedIndex (see page 9661) property is set.

When the selected cell in the data-aware grid changes, first the OnColExit event occurs. Then the values of SelectedField (see page 9661) and SelectedIndex (see page 9661) are updated. Finally the OnColEnter (see page 9665) event occurs. Thus, in an OnColExit event handler, the SelectedField (see page 9661) or SelectedIndex (see page 9661) property can be used to determine which column is about to lose focus.

See Also

OnColEnter (see page 9665)

SelectedField (see page 9661)

SelectedIndex (see page 9661)

49.1.4.7.4 TCustomDBGrid.OnColumnMoved Event

Occurs when the user moves a column using the mouse.

Pascal

```
property OnColumnMoved: TMovedEvent;
```

C++

```
__property TMovedEvent OnColumnMoved;
```

Description

Write an OnColumnMoved event handler to take specific action just after a column in the grid has been moved.

OnColumnMoved can only occur when the Options (see page 9659) property includes dgColumnResize.

See Also

Columns (see page 9653)

Options (see page 9659)

49.1.4.7.5 TCustomDBGrid.OnDrawColumnCell Event

Occurs when the grid needs to paint a cell.

Pascal

```
property OnDrawColumnCell: TDrawColumnCellEvent;
```

C++

```
__property TDrawColumnCellEvent OnDrawColumnCell;
```

Description

Write an OnDrawColumnCell event handler to provide customized drawing for the data in the cells of the grid. Draw on the cell using the methods of the Canvas property.

An OnDrawColumnCell event handler can call the DefaultDrawColumnCell (see page 9634) method to instruct the data-aware grid to write the data value in the cell. The OnDrawColumnCell event handler can augment the DefaultDrawColumnCell (see page 9634) method to provide visual indications of selection and focus.

If DefaultDrawing (see page 9655) is true, the data will already be drawn in the cell before the OnDrawColumnCell event, and the grid draws a focus rectangle around selected cells after the OnDrawColumnCell event.

Note: If the Columns (see page 9653) property has a State property of csDefault, the OnDrawDataCell (see page 9667) event occurs before the OnDrawColumnCell. The OnDrawDataCell (see page 9667) event is obsolete, and only included for backward compatibility. Do not set both an OnDrawDataCell (see page 9667) event handler and an OnDrawColumnCell event handler.

See Also

DefaultDrawColumnCell (see page 9634)

DefaultDrawing (see page 9655)

DrawCell (see page 9637)

OnDrawDataCell (see page 9667)

Canvas (see page 5921)

State (see page 9753)

Controlling Grid Drawing

49.1.4.7.6 TCustomDBGrid.OnDrawDataCell Event

Occurs when the grid needs to paint a cell if the State property of Columns (see page 9653) is csDefault.

Pascal

```
property OnDrawDataCell: TDrawDataCellEvent;
```

C++

```
__property TDrawDataCellEvent OnDrawDataCell;
```

Description

Do not write an OnDrawDataCell event handler. OnDrawDataCell is obsolete and included for backward compatibility. Instead, write an OnDrawColumnCell (see page 9666) event handler.

See Also

DefaultDrawDataCell (see page 9634)

DefaultDrawing (see page 9655)

OnDrawColumnCell (see page 9666)

State (see page 9753)

49.1.4.7.7 TCustomDBGrid.OnEditButtonClick Event

Occurs when the user presses the ellipsis button in a grid column.

Pascal

```
property OnEditButtonClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEditButtonClick;
```

Description

Write an OnEditButtonClick event handler to bring up an appropriate list or dialog when the user presses the ellipsis button in a grid cell. The OnEditButtonClick event handler can set the value of the field based on the user response to the list or dialog. Use the SelectedField (see page 9661) property to access the TField component whose value should be set.

See Also

OnCellClick (see page 9664)

SelectedField (see page 9661)

TField (see page 7115)

Putting a Button in a Column

49.1.4.7.8 TCustomDBGrid.OnTitleClick Event

Occurs when the user releases the mouse in one of the column headers.

Pascal

```
property OnTitleClick: TDBGridClickEvent;
```

C++

```
__property TDBGridClickEvent OnTitleClick;
```

Description

Write an OnTitleClick event handler to take specific action when the user clicks in one of the column headers. The Column parameter is the TColumn (see page 9583) object that corresponds to the column where the mouse was when the user released the left mouse button.

See Also

Columns (see page 9653)

OnCellClick (see page 9664)

TColumn (see page 9583)

49.1.5 TDBGrid Class

TDBGrid displays and manipulates records from a dataset in a tabular grid.

Class Hierarchy

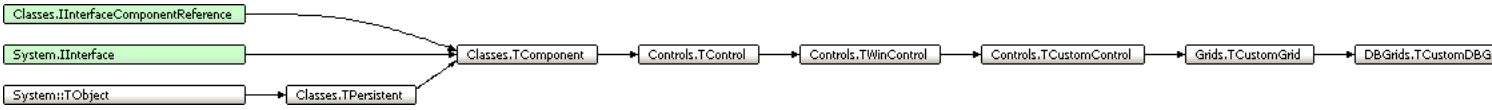

Pascal

```
TDBGrid = class(TCustomDBGrid);
```

C++

```
class TDBGrid : public TCustomDBGrid;
```

File

DBGrids

Description

Put a TDBGrid object on a form to display and edit the records from a database table or query. Applications can use the data grid to insert, delete, or edit data in the database, or simply to display it.

At runtime, users can use the database navigator (TDBNavigator) to move through data in the grid, and to insert, delete, and edit the data. Edits that are made in the data grid are not posted to the underlying dataset until the user moves to a different record or closes the application.

TDBGrid implements the generic behavior introduced in TCustomDBGrid (see page 9608). TDBGrid publishes many of the properties inherited from TCustomDBGrid (see page 9608), but does not introduce any new behavior.

Delphi Examples:

```
{
This example requires two ClientDataSets, two DBGrids, two
DataSources and two DBNavigators.  Name the two
ClientDataSets CDS and CDS2. Set the DataSet property in the
first DataSource to CDS and name that DataSource DS.  Set the
DataSet property in the second DataSource to CDS2 and name
that DataSource DS2.  Set the DataSource in the first
DBNavigator to DS and the DataSource in the second DBNavigator
to DS2.  Set the DataSource in the first DBGrid to DS and the
DataSource in the second DBGrid to DS2.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
type to Integer.  The file used to load CDS must have an
attribute name of HOST with an attribute type of string and
an attribute name of PORTA with an attribute type of integer.
Add DB, DBCtrls, Grids, DBGrids, DBClient and SwSystem to the
uses clause.  "LoadFormFile" to start with the file, or
"CreateDataSet" to start from scratch.  You must "CloseDataSet"
after a "LoadFromFile" before you can do another "CreateDataSet".
}
{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button1Click(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
```

```
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.SaveToFile(GetCurrentDir + '\CDS.XML', dfXML);
{$ELSE}
  CDS.SaveToFile(gsAppPath + 'CDS.XML', dfXML);
{$ENDIF}
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  CDS.CreateDataSet;
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  CDS2.Data := CDS.Data;
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  CDS.Close;
end;
```

See Also

TDBCtrlGrid (see page 8168)

TDBNavigator (see page 9230)

Editing Data in a Control

Using Common Data Control Features

Viewing and Editing Data with TDBGrid

Displaying

Using Object Fields

Working with DataSet Fields

Working with Reference Fields

Setting Persistent Field Properties and Events

Working with Field Components: Overview

Grids

49.1.5.1 TDBGrid Members

The following tables list the members exposed by TDBGrid.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.

	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 14814)	This is DesignOptionsBoost, a member of class TCustomGrid.
	FAcquireFocus (see page 9623)	This is FAcquireFocus, a member of class TCustomDBGrid.
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FGridState (see page 14814)	This is FGridState, a member of class TCustomGrid.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveCellExtents (see page 14814)	This is FSaveCellExtents, a member of class TCustomGrid.
	FUpdateFields (see page 9623)	This is FUpdateFields, a member of class TCustomDBGrid.
	VirtualView (see page 14814)	This is VirtualView, a member of class TCustomGrid.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AcquireLayoutLock (see page 9625)	Calls BeginLayout (see page 9626) if the table is not currently being reformatted.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AdjustSize (see page 14816)	Adds or deletes rows or columns from the grid.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	BeginColumnDrag (see page 9625)	Starts the dragging of a column in the grid.
	BeginColumnDrag (see page 14817)	Starts the dragging of a column in the grid.

	BeginLayout (see page 9626)	Increments the LayoutLock (see page 9659) property when the column bindings or fields in the dataset change.
	BeginRowDrag (see page 14817)	Starts the dragging of a row in the grid.
	BeginUpdate (see page 9627)	Increments the UpdateLock (see page 9664) property when the data in the grid changes.
	BoxRect (see page 14818)	Returns the screen coordinates of a specified set of cells.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcDrawInfo (see page 14818)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 14818)	Calculates the pixel locations of the fixed cells.
	CalcSizingState (see page 9627)	Determines the appearance of the mouse cursor during resizing.
	CalcSizingState (see page 14819)	Determines the appearance of the mouse cursor during resizing.
	CalcTitleRect (see page 9627)	This is CalcTitleRect, a member of class TCustomDBGrid.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CancelLayout (see page 9628)	Decrements the LayoutLock (see page 9659) property.
	CanEditAcceptKey (see page 9628)	Indicates whether the Key parameter can be entered into the current cell.
	CanEditAcceptKey (see page 14819)	Indicates whether the Key parameter can be entered into the current cell.
	CanEditModify (see page 9629)	Indicates whether the current column of the grid can be edited.
	CanEditModify (see page 14819)	Indicates whether the current cell in the grid can be edited.
	CanEditShow (see page 9629)	Indicates whether the inplace edit control can be created to allow editing.
	CanEditShow (see page 14820)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 14820)	Indicates whether the grid should respond to a given key combination.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	CellClick (see page 9629)	Generates an OnCellClick (see page 9664) event.
	CellRect (see page 14821)	Returns the screen coordinates of a cell in the grid.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeGridOrientation (see page 14821)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CheckColumnDrag (see page 9630)	Validates whether the column currently selected for dragging can be dropped at the current location.
	CheckColumnDrag (see page 14821)	CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 14822)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
	Click (see page 5743)	Respond to user click.
	ColEnter (see page 9630)	Generates an OnColEnter (see page 9665) event.
	ColExit (see page 9630)	Generates an OnColExit (see page 9665) event.
	ColumnAtDepth (see page 9631)	Returns the column object associated with nth parent field of the given column.
	ColumnMoved (see page 9631)	Generates an OnColumnMoved (see page 9666) event.
	ColumnMoved (see page 14822)	Provides the interface for a method that responds when the position of a column changes.
	ColWidthsChanged (see page 9631)	Updates the column widths in the Columns (see page 9653) property when the user resizes the columns.
	ColWidthsChanged (see page 14822)	Responds when the column widths change.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateColumns (see page 9632)	Creates the TDBGridColumns (see page 9746) object that stores the column descriptors for the grid.
	CreateDataLink (see page 9632)	Creates the data link that the grid uses to manage the connection to its data source.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.

	CreateEditor (see page 9632)	Creates the inplace editor that allows the cells of the grid to be edited.
	CreateEditor (see page 14823)	Creates the inplace editor control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 14823)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 9633)	Creates the window used to display the data-aware grid.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DataToRawColumn (see page 9633)	Converts a column index in the Columns (see page 9653) property to the index of the corresponding column in the visual display.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DeferLayout (see page 9635)	Decrements the LayoutLock (see page 9659) property and posts a message to the grid to trigger a new layout sequence at a more appropriate time.
	DefineFieldMap (see page 9636)	Builds the Fields (see page 9657) array from the Columns (see page 9653) property and the data source.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DefineProperties (see page 9636)	Defines a savable version of the Columns (see page 9653) property when necessary.
	DefineProperties (see page 14824)	Adds the ColWidths (see page 14845) and RowHeights (see page 14855) properties to the values saved with the grid.
	DeleteColumn (see page 14824)	Deletes a specified column from the grid.
	DeleteRow (see page 14825)	Deletes a specified from the grid.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoExit (see page 14825)	Hides the inplace editor when selection moves to another control in a form.
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.

	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
V	DoMouseWheelDown (see page 9636)	Performs tasks when the mouse wheel is rotated downward.
V	DoMouseWheelDown (see page 14825)	Performs tasks when the mouse wheel is rotated downward.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
V	DoMouseWheelUp (see page 9637)	Performs tasks when the mouse wheel is rotated upward.
V	DoMouseWheelUp (see page 14826)	Performs tasks when the mouse wheel is rotated upward.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawCell (see page 9637)	Draws the cell specified by the ACol and ARow parameters.
A	DrawCell (see page 14826)	Provides the prototype for a method that draws the contents of a cell in the grid.
	DrawColumnCell (see page 9638)	Generates an OnDrawColumnCell (see page 9666) event.
	DrawDataCell (see page 9638)	Generates an OnDrawDataCell (see page 9667) event.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EditButtonClick (see page 9638)	Generates an OnEditButtonClick (see page 9668) event.
V	EndColumnDrag (see page 9639)	Ends the dragging of a column in the grid.
	EndColumnDrag (see page 14827)	Ends the dragging of a column in the grid.
	EndLayout (see page 9639)	Decrements the LayoutLock (see page 9659) property and recomputes the Columns (see page 9653) property if LayoutLock (see page 9659) reaches 0.
	EndRowDrag (see page 14827)	Ends the dragging of a row in the grid.
	EndUpdate (see page 9640)	Decrements the UpdateLock (see page 9664) property.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	FocusCell (see page 14827)	Changes the current cell to a specified cell.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetColField (see page 9640)	Returns the field component that represents the binding of a specified column.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEditLimit (see page 9641)	Limits the number of characters that can be typed into cells representing string fields.
	GetEditLimit (see page 14828)	Provides the interface for a method that limits the number of characters the user can type into cells.
V	GetEditMask (see page 9641)	Returns the input mask associated with the field for the selected cell.
	GetEditMask (see page 14828)	Provides the interface for a method that returns an input mask for the specified cell.
V	GetEditStyle (see page 9642)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.

	GetEditStyle (see page 14828)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
V	GetEditText (see page 9642)	Returns the value of the indicated cell formatted for editing.
	GetEditText (see page 14829)	Returns the value of the indicated cell formatted for editing.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFieldValue (see page 9643)	Returns the value of the indicated cell formatted for display.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetGridHeight (see page 14829)	Returns the value of the GridHeight (see page 14850) property.
	GetGridWidth (see page 14830)	Returns the value of the GridWidth (see page 14851) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HideEdit (see page 14830)	Hides the inplace editor.
	HideEditor (see page 14830)	Hides the inplace editor when it is not in use.
V	HighlightCell (see page 9643)	Indicates whether a specified cell should be highlighted when it is drawn.
	InvalidateCell (see page 14831)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCol (see page 14831)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	InvalidateEditor (see page 14832)	Causes the inplace editor to repaint itself, moving to a new location if necessary.
	InvalidateGrid (see page 14832)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 14832)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
	InvalidateTitles (see page 9644)	Invalidates the column titles so that they will be repainted.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
V	KeyDown (see page 9644)	Provides special processing when the user presses down on a key.
V	KeyDown (see page 14833)	Provides special processing when the user presses a key.
	KeyPress (see page 6163)	Respond to keyboard input.
V	KeyPress (see page 9645)	Responds when the user presses a key.
V	KeyPress (see page 14833)	Provides special processing when the user presses a key.
	KeyUp (see page 6164)	Respond to released key.
V	LayoutChanged (see page 9645)	Recomputes the Columns (see page 9653) property.
V	LinkActive (see page 9646)	Recomputes the Columns (see page 9653) property and the vertical scroll bar of the grid.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 9646)	Sets up the Columns (see page 9653) property once the data grid and all its child components are loaded.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.

	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
V	MouseDown (see page 9647)	Provides special processing when the user clicks on the mouse.
V	MouseDown (see page 14834)	Provides special processing when the user presses a mouse button.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
V	MouseMove (see page 14834)	Provides special processing when the user moves the mouse.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	MouseUp (see page 9647)	Provides special processing when the user releases the mouse button.
V	MouseUp (see page 14835)	Provides special processing when the user releases the mouse button.
	MoveColRow (see page 14835)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 14836)	Moves a column to a new position.
	MoveRow (see page 14836)	Moves a row to a new position.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 9647)	Responds to notifications that components are about to be destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 5920)	Renders the image of a custom control.
V	Paint (see page 14836)	Draws the image of the grid control on the screen.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	RawToDataColumn (see page 9648)	Converts the index of a column in the visual display to the corresponding index in the Columns (see page 9653) property.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	RowHeightsChanged (see page 14837)	Responds when the row heights change.
	RowMoved (see page 14837)	Provides the interface for a method that responds when the position of a row changes.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
V	Scroll (see page 9648)	Updates the image of the grid to reflect a new record position.
	ScrollData (see page 14838)	Scrolls the visual image of the cells in the grid.
V	SelectCell (see page 14838)	Determines whether a particular cell in the grid can be selected.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.

	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
V	SetColumnAttributes (see page 9649)	This is SetColumnAttributes, a member of class TCustomDBGrid.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEditText (see page 9649)	Updates the edited version of the current cell's data.
	SetEditText (see page 14839)	Provides the interface for a method that updates the text associated with a cell.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	ShowEditor (see page 14839)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 14839)	Displays the inplace editor, with its text set to the specified character.
	SizeChanged (see page 14840)	Responds to changes in the size of the grid.
	Sizing (see page 14840)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	StoreColumns (see page 9650)	Indicates whether the columns of the grid are stored when the grid is saved, or whether they are generated dynamically.
V	TimedScroll (see page 9650)	Moves the current record in response to the user dragging the mouse outside of the data cells.
	TimedScroll (see page 14840)	Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.
	TitleClick (see page 9651)	Generates an OnTitleClick (see page 9668) event.
V	TopLeftChanged (see page 9651)	Updates the position of the inplace editor when the grid scrolls.
	TopLeftChanged (see page 14841)	Updates the position of the inplace editor when the grid scrolls.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateDesigner (see page 14841)	Notifies the designer when rows and columns of the grid are moved or resized.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
V	UpdateScrollBar (see page 9652)	This is UpdateScrollBar, a member of class TCustomDBGrid.

	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UseRightToLeftAlignmentForField (see page 9652)	Indicates whether the alignment of a specified field should be changed from left-aligned to right-aligned.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Col (see page 14843)	Specifies the index of the column that contains the selected cell.
	ColCount (see page 14844)	Specifies the number of columns in the grid.
	ColWidths (see page 14845)	Indicates the width (in pixels) of all the columns in the grid.
R	DataLink (see page 9654)	Helps the data-aware grid manage its link to the data source and respond to data events.
	DefaultColWidth (see page 14846)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultRowHeight (see page 14847)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	EditorMode (see page 14848)	Determines whether the current cell can be edited.
	FixedCols (see page 14848)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 14849)	Specifies the number of rows on the top of the grid that cannot be scrolled.
R	GridHeight (see page 14850)	Specifies the height of the grid in pixels.
	GridLineWidth (see page 14850)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
R	GridWidth (see page 14851)	Specifies the width of the grid in pixels.
R	HitTest (see page 14851)	Indicates the current position of the mouse cursor.
R	IndicatorOffset (see page 9658)	Indicates the index of the first column in the grid that contains data.
R	InplaceEditor (see page 14852)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	LayoutLock (see page 9659)	Counts the number of times BeginLayout (see page 9626) has been called without an EndLayout (see page 9639) or CancelLayout (see page 9628) call.
	LeftCol (see page 14852)	Specifies the index of the first visible scrollable column in the grid.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	Row (see page 14854)	Specifies the index of the row that contains the selected cell.
	RowCount (see page 14854)	Specifies the number of rows in the grid.
	RowHeights (see page 14855)	Gives the height (in pixels) of all rows in the grid.

	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ScrollBars (see page 14856)	Specifies whether the grid includes horizontal and vertical scroll bars.
	Selection (see page 14856)	Indicates the boundaries of the current selection.
	TabStops (see page 14858)	Indicates whether the user can tab to specified columns in the grid.
	Text (see page 5842)	Contains a text string associated with the control.
	TopRow (see page 14858)	Specifies the index of the first visible scrollable row in the grid.
R	UpdateLock (see page 9664)	Counts the number of times BeginUpdate (see page 9627) is called without a corresponding call to EndUpdate (see page 9640).
R	VisibleColCount (see page 14859)	Indicates the number of scrollable columns visible in the grid.
R	VisibleRowCount (see page 14859)	Indicates the number of scrollable rows visible in the grid.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 5918)	Creates an instance of TCustomControl (see page 5908).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 9622)	Creates and initializes an instance of TCustomDBGrid (see page 9608).
V	Create (see page 14813)	Creates an instance of TCustomGrid (see page 14800).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultDrawColumnCell (see page 9634)	Draws the text in a column cell.
	DefaultDrawDataCell (see page 9634)	Draws the text in a column cell.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.

	DefaultHandler (see page 9635)	Displays the popup menu when a column is right-clicked.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 9622)	Destroys an instance of TCustomDBGrid (see page 9608).
	Destroy (see page 14813)	Destroys an instance of TCustomGrid (see page 14800).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteAction (see page 9640)	Executes an action with the DB (see page 6725) grid as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.

	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseCoord (see page 14834)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.

	Show (see page 5788)	Makes a control visible.
	ShowPopupEditor (see page 9649)	Displays the nested dataset associated with Column in a new window.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateAction (see page 9651)	Updates an action to reflect the current state of the grid.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.
	ValidFieldIndex (see page 9652)	Indicates whether an index into the columns array refers to a valid field in the dataset.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	Canvas (see page 9690)	Specifies the TCanvas object that presents a drawing surface for the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EditorMode (see page 9656)	Determines whether the current cell can be edited.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	FieldCount (see page 9656)	Specifies the number of fields (columns) displayed in the grid.
R	Fields (see page 9657)	Provides indexed access to the field components that correspond to the columns.
R	Floating (see page 5823)	Indicates whether the control is floating.

	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelectedField (see page 9661)	Specifies the field component for the currently selected cell in the grid.
	SelectedIndex (see page 9661)	Specifies the index of the currently selected column in the Columns (see page 9653) array.
	SelectedRows (see page 9709)	Specifies a set of bookmarks for all the records in the dataset that correspond to rows selected in the grid.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnCellClick (see page 9715)	Occurs when the user releases the mouse in one of the cells of the grid.
	OnColEnter (see page 9716)	Occurs when focus moves to a new cell in the grid.
	OnColExit (see page 9716)	Occurs just before a cell in the grid loses input focus.
	OnColumnMoved (see page 9717)	Occurs when the user moves a column using the mouse.
	OnDblClick (see page 9717)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 9717)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 9720)	Occurs when the user drags an object over a control.
	OnDrawColumnCell (see page 9722)	Occurs when the grid needs to paint a cell.
	OnDrawDataCell (see page 9723)	Occurs when the grid needs to paint a cell if the State property of Columns is csDefault.
	OnEditButtonClick (see page 9723)	Occurs when the user presses the ellipsis button in a grid column.
	OnEndDock (see page 9724)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 9724)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 9725)	Occurs when a control receives the input focus.
	OnExit (see page 9728)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 9729)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 9732)	Occurs when a key is pressed.
	OnKeyUp (see page 9733)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 9735)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 9736)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 9738)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 9738)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 9738)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 9741)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 9743)	Occurs when the mouse wheel is rotated.

	OnMouseWheelDown (see page 9743)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 9744)	Occurs when the mouse wheel is rotated upward.
	OnStartDock (see page 9744)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 9745)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnTitleClick (see page 9745)	Occurs when the user releases the mouse in one of the column headers.

Published Properties

	Name	Description
	Align (see page 9686)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 9689)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 9690)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 9690)	Determines whether a single line border is drawn around the grid.
	Color (see page 9692)	Specifies the background color of the control.
	Columns (see page 9693)	Describes the display attributes and field bindings of the columns.
	Constraints (see page 9694)	Specifies the size constraints for the control.
	Ctl3D (see page 9694)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DataSource (see page 9695)	Identifies the link to the dataset where the data-aware grid finds its data.
	DefaultDrawing (see page 9696)	Indicates whether the cells in the data-aware grid are drawn automatically.
	DragCursor (see page 9696)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 9697)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 9697)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 9699)	Controls whether the control responds to mouse, keyboard, and timer events.
	FixedColor (see page 9699)	Specifies the background color of the fixed rows and columns in the grid.
	Font (see page 9699)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 9700)	Determines the behavior of the input method editor (IME).
	ImeName (see page 9701)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 9702)	Specifies various display and behavioral properties of the data-aware grid.
	ParentBiDiMode (see page 9702)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 9702)	Determines where a control looks for its color information.
	ParentCtl3D (see page 9703)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 9703)	Determines where a control looks for its font information.
	ParentShowHint (see page 9704)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 9704)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 9709)	Indicates whether the grid is used for display only, or whether the user can edit data using the grid.
	ShowHint (see page 9710)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 9712)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 9712)	Determines if the user can tab to a control.

	Tag (see page 2808)	Stores an integer value as part of a component.
	TitleFont (see page 9713)	Describes the font used to draw the column titles in the grid.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 9713)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

49.1.5.2 TDBGrid Properties

The properties of the TDBGrid class are listed here.

Public Properties

	Name	Description
	Canvas (see page 9690)	Specifies the TCanvas object that presents a drawing surface for the control.
	SelectedRows (see page 9709)	Specifies a set of bookmarks for all the records in the dataset that correspond to rows selected in the grid.

Published Properties

	Name	Description
	Align (see page 9686)	Determines how the control aligns within its container (parent control).
	Anchors (see page 9689)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 9690)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 9690)	Determines whether a single line border is drawn around the grid.
	Color (see page 9692)	Specifies the background color of the control.
	Columns (see page 9693)	Describes the display attributes and field bindings of the columns.
	Constraints (see page 9694)	Specifies the size constraints for the control.
	Ctl3D (see page 9694)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DataSource (see page 9695)	Identifies the link to the dataset where the data-aware grid finds its data.
	DefaultDrawing (see page 9696)	Indicates whether the cells in the data-aware grid are drawn automatically.
	DragCursor (see page 9696)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 9697)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 9697)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 9699)	Controls whether the control responds to mouse, keyboard, and timer events.
	FixedColor (see page 9699)	Specifies the background color of the fixed rows and columns in the grid.
	Font (see page 9699)	Controls the attributes of text written on or in the control.
	ImeMode (see page 9700)	Determines the behavior of the input method editor (IME).
	ImeName (see page 9701)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Options (see page 9702)	Specifies various display and behavioral properties of the data-aware grid.
	ParentBiDiMode (see page 9702)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 9702)	Determines where a control looks for its color information.
	ParentCtl3D (see page 9703)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 9703)	Determines where a control looks for its font information.
	ParentShowHint (see page 9704)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 9704)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 9709)	Indicates whether the grid is used for display only, or whether the user can edit data using the grid.
	ShowHint (see page 9710)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 9712)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 9712)	Determines if the user can tab to a control.
	TitleFont (see page 9713)	Describes the font used to draw the column titles in the grid.
	Visible (see page 9713)	Determines whether the component appears onscreen.

49.1.5.2.1 TDBGrid.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
```

```
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
```

```
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
```

```
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

49.1.5.2.2 TDBGrid.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

49.1.5.2.3 TDBGrid.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

49.1.5.2.4 TDBGrid.BorderStyle Property

Determines whether a single line border is drawn around the grid.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Set BorderStyle to bsSingle to add a single line border around the grid's image. Set BorderStyle to bsNone to omit the border.

49.1.5.2.5 TDBGrid.Canvas Property

Specifies the TCanvas object that presents a drawing surface for the control.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control. Canvas encapsulates a Windows device context, providing all the tools and methods needed for drawing and painting.

Canvas is a protected property that is usually redeclared as public in descendants of TCustomControl.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas,Rect.Left,Rect.Top,index);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(index) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TStringGrid *myStringGrid;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TStringGrid")
  {
    myStringGrid = (TStringGrid *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myStringGrid->Canvas->MoveTo(myStringGrid->ClientRect.Left, myStringGrid->ClientRect.Top);
    myStringGrid->Canvas->LineTo(myStringGrid->ClientRect.Right,
myStringGrid->ClientRect.Bottom);
    myStringGrid = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
```

```
  component to draw the contents of each cell in a draw grid.
  It draws a focus rectangle around the cell that has focus.
  goDrawFocusSelect in the DrawGrid Options parameter must be
  True to set focus on a cell.  The ImageList Draw method must
  be called after DrawFocusRect. The OnSelectCell event handler
  must implemented to return true.
  }
  procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
    Rect: TRect; State: TGridDrawState);
  var
    index: Integer;
  begin
    index := ARow * DrawGrid1.ColCount + ACol;
    DrawGrid1.Canvas.Brush.Color := clWhite;
    DrawGrid1.Canvas.FillRect(Rect);
    if (gdFocused in State) then
    begin
      DrawGrid1.Canvas.DrawFocusRect(Rect);
      Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
    end;
    ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  end;

  procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  begin
    CanSelect:= True;
  end;
  {
  The following code uses ClientRect to find and draw a line
  from the top left to the bottom right of the current
  control.  Select a TControl in the ListBox list.  This
  example assumes that the current object is a descendent of
  TControl and has a public Canvas property.
  }
    if (ListBox1.Items.Objects[Index] is TStringGrid) then
    begin
      myStringGrid:= (ListBox1.Items.Objects[Index] as TStringGrid);
      with myStringGrid.ClientRect do
      begin // make sure we are using the TControl's canvas and not the form's!
        myStringGrid.Canvas.MoveTo(Left, Top);
        myStringGrid.Canvas.LineTo(Right, Bottom);
      end;
    myStringGrid:= nil;
    end;
```

See Also

Paint (see page 5920)

49.1.5.2.6 TDBGrid.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the

Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

49.1.5.2.7 TDBGrid.Columns Property

Describes the display attributes and field bindings of the columns.

Pascal

```
property Columns: TDBGridColumns;
```

C++

```
__property TDBGridColumns Columns;
```

Description

Use Columns to read or set the field bindings and display attributes of the columns in the data-aware grid. The Columns property is an indexed collection of TColumn objects. Use the properties of the TColumn objects to specify the display attributes or field bindings of individual columns in the grid. The field binding of a column designates a field within the dataset specified by the DataSource property.

Columns can be set at design time through the Columns editor, or programmatically at runtime.

See Also

CreateColumns (see page 9632)

Fields (see page 9657)

IndicatorOffset (see page 9658)

setcolumnattributes (see page 9649)

StoreColumns (see page 9650)

TDBGridColumns (see page 9746)

TColumn (see page 9583)

Creating Persistent Columns

Viewing and Editing Data with TDBGrid

49.1.5.2.8 TDBGrid.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

49.1.5.2.9 TDBGrid.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception

of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

49.1.5.2.10 TDBGrid.DataSource Property

Identifies the link to the dataset where the data-aware grid finds its data.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Set DataSource to the TDataSource object that links to the dataset where the data-aware grid should fetch its data. DataSource allows the data-aware grid to read from, write to, and navigate around the dataset.

Delphi Examples:

```
{
The following example copies the selected rows in a db grid
to a list box.  Set the db grid Options dgRowSelect,
dgAlwaysShowSelection and dgMultiSelect to True.  Make a
multiple selecton using the CNTL key.  This example requires
a TDataSet associated with a TDataSource and a TDBGrid.}
procedure TForm1.Button2Click(Sender: TObject);
var
  i, j: Integer;
  s: string;
begin
  if DBGrid2.SelectedRows.Count>0 then
    with DBGrid2.DataSource.DataSet do
      for i:=0 to DBGrid2.SelectedRows.Count-1 do
      begin
        GotoBookmark(pointer(DBGrid2.SelectedRows.Items[i]));
        for j := 0 to FieldCount-1 do
        begin
          if (j>0) then s:=s+', ';
          s:=s+Fields[j].AsString;
        end;
        Listbox1.Items.Add(s);
        s:= '';
      end;
end;
```

See Also

DataLink (see page 9654)

Fields (see page 9657)

TDataSource (see page 7070)

Making the Table a Detail of Another Dataset

49.1.5.2.11 TDBGrid.DefaultDrawing Property

Indicates whether the cells in the data-aware grid are drawn automatically.

Pascal

```
property DefaultDrawing: Boolean;
```

C++

```
__property Boolean DefaultDrawing;__property Boolean DefaultDrawing;
```

Description

Set DefaultDrawing to true to allow the data-aware grid to draw the data in the cells of the grid automatically. Set DefaultDrawing to false to turn off the default drawing when providing customized painting in an OnDrawColumnCell event handler or an OnDrawDataCell event handler.

When DefaultDrawing is true, the data-aware grid fetches the data for each cell from the appropriate entry in the Fields array, and writes it to the cell. It then draws the focus rectangle if the cell has focus, or if the cell is selected and the Options property includes dgAlwaysShowSelection.

When DefaultDrawing is false, the data-aware grid draws the appropriate background color on the cell and sets up the brush and font described by the appropriate TColumn object. However, the value of the field is not written to the cell, and no focus rectangle is drawn.

See Also

Columns (see page 9653)

Fields (see page 9657)

OnDrawColumnCell (see page 9666)

OnDrawDataCell (see page 9667)

TColumn (see page 9583)

Controlling Grid Drawing

49.1.5.2.12 TDBGrid.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

49.1.5.2.13 TDBGrid.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

49.1.5.2.14 TDBGrid.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
```

```
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

49.1.5.2.15 TDBGrid.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

49.1.5.2.16 TDBGrid.FixedColor Property

Specifies the background color of the fixed rows and columns in the grid.

Pascal

```
property FixedColor: TColor;
```

C++

```
__property TColor FixedColor;
```

Description

Set FixedColor to specify the color for the nonscrolling cells in the grid. The scrolling region of the grid is drawn using the Color property.

See Also

Color (see page 5812)

FixedCols (see page 14848)

FixedRows (see page 14849)

49.1.5.2.17 TDBGrid.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

49.1.5.2.18 TDBGrid.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

49.1.5.2.19 TDBGrid.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

49.1.5.2.20 TDBGrid.Options Property

Specifies various display and behavioral properties of the data-aware grid.

Pascal

```
property Options: TGridOptions;
```

C++

```
__property TDBGridOptions Options;__property TGridOptions Options;
```

Description

Set Options to include the desired properties for the data-aware grid.

See Also

ReadOnly (see page 9660)

SelectedIndex (see page 9661)

SelectedRows (see page 9662)

Setting Grid Options

49.1.5.2.21 TDBGrid.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

49.1.5.2.22 TDBGrid.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

49.1.5.2.23 TDBGrid.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

49.1.5.2.24 TDBGrid.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

49.1.5.2.25 TDBGrid.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

49.1.5.2.26 TDBGrid.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
```

```
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}
```

```
void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
```

```
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
```

```
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

49.1.5.2.27 TDBGrid.ReadOnly Property

Indicates whether the grid is used for display only, or whether the user can edit data using the grid.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set ReadOnly to true to prevent users from changing the data in the dataset. Set ReadOnly to false to allow users to edit data using the grid.

When ReadOnly is true, users cannot use the Insert key to insert a new row in the grid or use the Down Arrow key to add a row to the end of the grid.

See Also

CanEditModify (see page 9629)

Options (see page 9659)

Editing in the Grid

49.1.5.2.28 TDBGrid.SelectedRows Property

Specifies a set of bookmarks for all the records in the dataset that correspond to rows selected in the grid.

Pascal

```
property SelectedRows: TBookmarkList;
```

C++

```
__property TBookmarkList SelectedRows;
```

Description

Use the properties and methods of the TBookmarkList object returned by SelectedRows to

Determine the number of rows in the grid that are selected.

Determine whether the current record in the dataset is selected.

Determine whether a particular record in the dataset is selected.

Delete all selected rows from the dataset.

SelectedRows is only meaningful when the Options property includes dgRowSelect and dgMultiSelect.

Delphi Examples:

```
{
The following example copies the selected rows in a db grid
to a list box.  Set the db grid Options dgRowSelect,
dgAlwaysShowSelection and dgMultiSelect to True.  Make a
multiple selecton using the CNTL key.  This example requires
a TDataSet associated with a TDataSource and a TDBGrid.}
procedure TForm1.Button2Click(Sender: TObject);
var
  i, j: Integer;
  s: string;
begin
  if DBGrid2.SelectedRows.Count>0 then
    with DBGrid2.DataSource.DataSet do
      for i:=0 to DBGrid2.SelectedRows.Count-1 do
      begin
        GotoBookmark(pointer(DBGrid2.SelectedRows.Items[i]));
        for j := 0 to FieldCount-1 do
        begin
          if (j>0) then s:=s+', ';
          s:=s+Fields[j].AsString;
        end;
        Listbox1.Items.Add(s);
        s:= '';
      end;
end;
```

See Also

Options (see page 9659)

TBookmarkList (see page 9575)

49.1.5.2.29 TDBGrid.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

49.1.5.2.30 TDBGrid.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

49.1.5.2.31 TDBGrid.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

49.1.5.2.32 TDBGrid.TitleFont Property

Describes the font used to draw the column titles in the grid.

Pascal

```
property TitleFont: TFont;
```

C++

```
__property TFont TitleFont;
```

Description

Set TitleFont to change the font used to draw the column headings. TitleFont is only meaningful if the Options property includes dgTitles.

See Also

Options (see page 9659)

TFont (see page 14497)

49.1.5.2.33 TDBGrid.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

49.1.5.3 TDBGrid Events

The events of the TDBGrid class are listed here.

Published Events

	Name	Description
	OnCellClick (see page 9715)	Occurs when the user releases the mouse in one of the cells of the grid.
	OnColEnter (see page 9716)	Occurs when focus moves to a new cell in the grid.
	OnColExit (see page 9716)	Occurs just before a cell in the grid loses input focus.
	OnColumnMoved (see page 9717)	Occurs when the user moves a column using the mouse.
	OnDblClick (see page 9717)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 9717)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 9720)	Occurs when the user drags an object over a control.
	OnDrawColumnCell (see page 9722)	Occurs when the grid needs to paint a cell.
	OnDrawDataCell (see page 9723)	Occurs when the grid needs to paint a cell if the State property of Columns is csDefault.
	OnEditButtonClick (see page 9723)	Occurs when the user presses the ellipsis button in a grid column.
	OnEndDock (see page 9724)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 9724)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 9725)	Occurs when a control receives the input focus.
	OnExit (see page 9728)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 9729)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 9732)	Occurs when a key is pressed.
	OnKeyUp (see page 9733)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 9735)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 9736)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 9738)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 9738)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 9738)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 9741)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 9743)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 9743)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 9744)	Occurs when the mouse wheel is rotated upward.
	OnStartDock (see page 9744)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 9745)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnTitleClick (see page 9745)	Occurs when the user releases the mouse in one of the column headers.

49.1.5.3.1 TDBGrid.OnCellClick Event

Occurs when the user releases the mouse in one of the cells of the grid.

Pascal

```
property OnCellClick: TDBGridClickEvent;
```

C++

```
__property TDBGridClickEvent OnCellClick;
```

Description

Write an OnCellClick event handler to take specific action when the user clicks in one of the cells of the data-aware grid.

OnCellClick occurs when the user releases the mouse. To take action when the user clicks down on the mouse, use the OnColEnter event.

OnCellClick will not occur when the mouse is released at the end of resizing a row or column, or when the cell is being edited.

See Also

Columns (see page 9653)

OnColEnter (see page 9665)

OnTitleClick (see page 9668)

TColumn (see page 9583)

49.1.5.3.2 TDBGrid.OnColEnter Event

Occurs when focus moves to a new cell in the grid.

Pascal

```
property OnColEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnColEnter;
```

Description

Write an OnColEnter event handler to take specific action when a new cell has just been selected. Focus moves to a cell when

The user navigates to the cell using the keyboard. For example, when the user uses the Tab key, or the Home key.

The user clicks the mouse button down in the cell.

The SelectedField or SelectedIndex property is set.

Read the SelectedField or SelectedIndex property to determine which cell was just entered.

See Also

OnCellClick (see page 9664)

OnColExit (see page 9665)

SelectedField (see page 9661)

SelectedIndex (see page 9661)

49.1.5.3.3 TDBGrid.OnColExit Event

Occurs just before a cell in the grid loses input focus.

Pascal

```
property OnColExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnColExit;
```

Description

Write an OnColExit event handler to take specific action when a cell is about to lose focus. For example, use OnColExit to post edits to the underlying dataset when the user leaves a cell, rather than waiting for the user to leave the record.

A cell loses focus when

The user navigates out of the cell using the keyboard. For example, when the user presses the Tab key or the Home key.

The user clicks the mouse button down in another cell.

The SelectedField or SelectedIndex property is set.

When the selected cell in the data-aware grid changes, first the OnColExit event occurs. Then the values of SelectedField and SelectedIndex are updated. Finally the OnColEnter event occurs. Thus, in an OnColExit event handler, the SelectedField or SelectedIndex property can be used to determine which column is about to lose focus.

See Also

OnColEnter (see page 9665)

SelectedField (see page 9661)

SelectedIndex (see page 9661)

49.1.5.3.4 TDBGrid.OnColumnMoved Event

Occurs when the user moves a column using the mouse.

Pascal

```
property OnColumnMoved: TMovedEvent;
```

C++

```
__property TMovedEvent OnColumnMoved;
```

Description

Write an OnColumnMoved event handler to take specific action just after a column in the grid has been moved.

OnColumnMoved can only occur when the Options property includes dgColumnResize.

See Also

Columns (see page 9653)

Options (see page 9659)

49.1.5.3.5 TDBGrid.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

49.1.5.3.6 TDBGrid.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

49.1.5.3.7 TDBGrid.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

49.1.5.3.8 TDBGrid.OnDrawColumnCell Event

Occurs when the grid needs to paint a cell.

Pascal

```
property OnDrawColumnCell: TDrawColumnCellEvent;
```

C++

```
__property TDrawColumnCellEvent OnDrawColumnCell;
```

Description

Write an OnDrawColumnCell event handler to provide customized drawing for the data in the cells of the grid. Draw on the cell using the methods of the Canvas property.

An OnDrawColumnCell event handler can call the DefaultDrawColumnCell method to instruct the data-aware grid to write the data value in the cell. The OnDrawColumnCell event handler can augment the DefaultDrawColumnCell method to provide visual indications of selection and focus.

If DefaultDrawing is true, the data will already be drawn in the cell before the OnDrawColumnCell event, and the grid draws a focus rectangle around selected cells after the OnDrawColumnCell event.

Note: If the Columns property has a State property of csDefault, the OnDrawDataCell event occurs before the OnDrawColumnCell. The OnDrawDataCell event is obsolete, and only included for backward compatibility. Do not set both an

OnDrawDataCell event handler and an OnDrawColumnCell event handler.

See Also

DefaultDrawColumnCell (see page 9634)

DefaultDrawing (see page 9655)

DrawCell (see page 9637)

OnDrawDataCell (see page 9667)

Canvas (see page 5921)

State (see page 9753)

Controlling Grid Drawing

49.1.5.3.9 TDBGrid.OnDrawDataCell Event

Occurs when the grid needs to paint a cell if the State property of Columns is csDefault.

Pascal

```
property OnDrawDataCell: TDrawDataCellEvent;
```

C++

```
__property TDrawDataCellEvent OnDrawDataCell;
```

Description

Do not write an OnDrawDataCell event handler. OnDrawDataCell is obsolete and included for backward compatibility. Instead, write an OnDrawColumnCell event handler.

See Also

DefaultDrawDataCell (see page 9634)

DefaultDrawing (see page 9655)

OnDrawColumnCell (see page 9666)

State (see page 9753)

49.1.5.3.10 TDBGrid.OnEditButtonClick Event

Occurs when the user presses the ellipsis button in a grid column.

Pascal

```
property OnEditButtonClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEditButtonClick;
```

Description

Write an OnEditButtonClick event handler to bring up an appropriate list or dialog when the user presses the ellipsis button in a grid cell. The OnEditButtonClick event handler can set the value of the field based on the user response to the list or dialog. Use the SelectedField property to access the TField component whose value should be set.

See Also

OnCellClick (see page 9664)

SelectedField (see page 9661)

TField (see page 7115)

Putting a Button in a Column

49.1.5.3.11 TDBGrid.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

49.1.5.3.12 TDBGrid.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
```

```
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

49.1.5.3.13 TDBGrid.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
```

```
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

49.1.5.3.14 TDBGrid.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

49.1.5.3.15 TDBGrid.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the

OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
```

```
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

49.1.5.3.16 TDBGrid.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
```

```
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

49.1.5.3.17 TDBGrid.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
```

```
      else
        ComboBox1->Text = "KeyDown";
      Form1->Refresh();
      Memo1->Lines->Add(ComboBox1->Text);
    }
  }

  void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
  {
    if (Key == VK_F1)
    {
      Form1->Color = FormColor;
      if (ComboBox1->Text == "KeyUp")
        ComboBox1->Text = "KeyUp2";
      else
        ComboBox1->Text = "KeyUp";
      Form1->Refresh();
      Memo1->Lines->Add(ComboBox1->Text);
    }
  }
```

Delphi Examples:

```
  {
  This example changes a form's color to aqua when the user
  presses the F1 key, even when a control on the form has the
  focus. When the user releases the key, the form returns to
  its original color.  Set the KeyPreview on the form to True.
  Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
  event to FormKeyUp.  Set the KeyPreview on the form to True.
  }
  var
    FormColor: TColor;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    KeyPreview := True;
    FormColor := Form1.Color;
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

49.1.5.3.18 TDBGrid.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

49.1.5.3.19 TDBGrid.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
```

```
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
```

```
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

49.1.5.3.20 TDBGrid.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

49.1.5.3.21 TDBGrid.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

49.1.5.3.22 TDBGrid.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
```

```
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

49.1.5.3.23 TDBGrid.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
```

```
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
```

```
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

49.1.5.3.24 TDBGrid.OnMouseWheel Event

Occurs when the mouse wheel is rotated.

Pascal

```
property OnMouseWheel: TMouseWheelEvent;
```

C++

```
__property TMouseWheelEvent OnMouseWheel;
```

Description

Write code in the OnMouseWheel event handler to respond to mouse wheel messages. If there is no OnMouseWheel event handler, or if the mouse wheel message is not handled in the OnMouseWheel event handler, then an OnMouseWheelUp or OnMouseWheelDown event occurs, depending on the direction the mouse wheel was rotated.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheelUp (see page 5870)

49.1.5.3.25 TDBGrid.OnMouseWheelDown Event

Occurs when the mouse wheel is rotated downward.

Pascal

```
property OnMouseWheelDown: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelDown;
```

Description

Write code in the OnMouseWheelDown event handler to perform actions when the mouse wheel is rotated downward. OnMouseWheelDown occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheel (see page 5869)

OnMouseWheelUp (see page 5870)

49.1.5.3.26 TDBGrid.OnMouseWheelUp Event

Occurs when the mouse wheel is rotated upward.

Pascal

```
property OnMouseWheelUp: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelUp;
```

Description

Write code in the OnMouseWheelUp event handler to perform actions when the mouse wheel is rotated upward. OnMouseWheelUp occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheel (see page 5869)

49.1.5.3.27 TDBGrid.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

49.1.5.3.28 TDBGrid.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

49.1.5.3.29 TDBGrid.OnTitleClick Event

Occurs when the user releases the mouse in one of the column headers.

Pascal

```
property OnTitleClick: TDBGridClickEvent;
```

C++

```
__property TDBGridClickEvent OnTitleClick;
```

Description

Write an OnTitleClick event handler to take specific action when the user clicks in one of the column headers. The Column parameter is the TColumn object that corresponds to the column where the mouse was when the user released the left mouse button.

See Also

Columns (see page 9653)

OnCellClick (see page 9664)

TColumn (see page 9583)

49.1.6 TDBGridColumns Class

TDBGridColumns represents a container for TColumn (see page 9583) objects.

Class Hierarchy

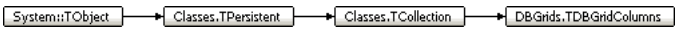

Pascal

```
TDBGridColumns = class(TCollection);
```

C++

```
class TDBGridColumns : public TCollection;
```

File

DBGrids

Description

Each TDBGridColumns holds a collection of TColumn (see page 9583) objects in a data grid (TDBGrid (see page 9668)). TDBGridColumns maintains an index of the columns in its Items (see page 9753) array. The Count property contains the number of columns in the collection. At design time, use the data grid's Columns editor to add, remove, or modify columns.

See Also

TCollection (see page 2731)

TCollectionItem (see page 2755)

TColumn (see page 9583)

TDBGrid (see page 9668)

TCustomDBGrid (see page 9608)

Viewing and Editing Data with TDBGrid

49.1.6.1 TDBGridColumns Members

The following tables list the members exposed by TDBGridColumns.

Protected Methods

	Name	Description
	Added (see page 2736)	Responds when items are added to the collection.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	Deleting (see page 2741)	Responds when items are deleted from the collection.

	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 9749)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 9752)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 9749)	Creates a new TColumn (see page 9583) instance and adds it to the Items (see page 9753) array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 9748)	Creates and initializes a TDBGridColumns (see page 9746) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.

	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LoadFromFile (see page 9750)	Reads the file specified in FileName and loads the data into the TDBGridColumns (see page 9746) object.
	LoadFromStream (see page 9750)	Reads grid columns data from a stream and stores the contents in the TDBGridColumns (see page 9746) object.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
	RebuildColumns (see page 9750)	Deletes existing columns and generates new columns from the dataset associated with the grid.
	RestoreDefaults (see page 9751)	Restores all columns to their default settings.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 9751)	Saves the TDBGridColumns (see page 9746) object to the file specified in FileName.
	SaveToStream (see page 9751)	Writes a stream with the name passed in Stream with the contents of the TDBGridColumns (see page 9746) object.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	Grid (see page 9752)	Indicates the data-grid control to which the TDBGridColumns (see page 9746) instance belongs.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 9753)	Lists the columns in the collection.
	State (see page 9753)	Determines whether the columns in the data grid are dependent on dataset fields.

49.1.6.2 TDBGridColumns.Create Constructor

Creates and initializes a TDBGridColumns (see page 9746) object.

Pascal

```
constructor Create(Grid: TCustomDBGrid; ColumnClass: TColumnClass);
```

C++

```
__fastcall TDBGridColumns(TCustomDBGrid Grid, TColumnClass ColumnClass);
```

Description

The Create method takes two parameters: a data-grid instance object and TColumn (see page 9583) (or the name of a class derived from TColumn (see page 9583)).

An application should seldom need to call Create directly. Instead, objects that use a TDBGridColumns (see page 9746) object instantiate the object as part of their initialization. An example of such an object is the TDBGrid (see page 9668) and its Columns property.

See Also

TDBGrid (see page 9668)

Columns (see page 9653)

49.1.6.3 TDBGridColumns Methods

The methods of the TDBGridColumns class are listed here.

Protected Methods

	Name	Description
	GetOwner (see page 9749)	Returns the owner of an object.
	Update (see page 9752)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 9749)	Creates a new TColumn (see page 9583) instance and adds it to the Items (see page 9753) array.
	LoadFromFile (see page 9750)	Reads the file specified in FileName and loads the data into the TDBGridColumns (see page 9746) object.
	LoadFromStream (see page 9750)	Reads grid columns data from a stream and stores the contents in the TDBGridColumns (see page 9746) object.
	RebuildColumns (see page 9750)	Deletes existing columns and generates new columns from the dataset associated with the grid.
	RestoreDefaults (see page 9751)	Restores all columns to their default settings.
	SaveToFile (see page 9751)	Saves the TDBGridColumns (see page 9746) object to the file specified in FileName.
	SaveToStream (see page 9751)	Writes a stream with the name passed in Stream with the contents of the TDBGridColumns (see page 9746) object.

49.1.6.3.1 TDBGridColumns.Add Method

Creates a new TColumn (see page 9583) instance and adds it to the Items (see page 9753) array.

Pascal

```
function Add: TColumn;
```

C++

```
__fastcall TColumn Add();
```

Description

Add returns the new column. At design time, use the data grid's Columns editor to add columns to the grid.

See Also

Items (see page 9753)

Columns (see page 9653)

49.1.6.3.2 TDBGridColumns.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are

introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

49.1.6.3.3 TDBGridColumns.LoadFromFile Method

Reads the file specified in FileName and loads the data into the TDBGridColumns (see page 9746) object.

Pascal

```
procedure LoadFromFile(const Filename: string);
```

C++

```
__fastcall LoadFromFile(const AnsiString Filename);
```

Description

Use the LoadFromFile method to retrieve grid columns data from a file and load into a TDBGridColumns (see page 9746) object. The file must be created by calling the SaveToFile (see page 9751) method of a TDBGridColumns (see page 9746) object.

See Also

SaveToFile (see page 9751)

49.1.6.3.4 TDBGridColumns.LoadFromStream Method

Reads grid columns data from a stream and stores the contents in the TDBGridColumns (see page 9746) object.

Pascal

```
procedure LoadFromStream(S: TStream);
```

C++

```
__fastcall LoadFromStream(TStream S);
```

Description

Use the LoadFromStream method to read Stream and store the contents in the TDBGridColumns (see page 9746) object. The stream should contain data previously saved using the SaveToStream (see page 9751) method.

See Also

SaveToStream (see page 9751)

49.1.6.3.5 TDBGridColumns.RebuildColumns Method

Deletes existing columns and generates new columns from the dataset associated with the grid.

Pascal

```
procedure RebuildColumns;
```

C++

```
__fastcall RebuildColumns();
```

Description

RebuildColumns initializes the TDBGridColumns (see page 9746) object by deleting all existing columns, then creating new ones from the grid's associated dataset. If the grid is not connected to a dataset, RebuildColumns simply deletes any existing columns.

49.1.6.3.6 TDBGridColumns.RestoreDefaults Method

Restores all columns to their default settings.

Pascal

```
procedure RestoreDefaults;
```

C++

```
__fastcall RestoreDefaults();
```

Description

RestoreDefaults iterates through the Items (see page 9753) array, calling the RestoreDefaults method for each TColumn (see page 9583) object. This method removes all changes made to properties of the TColumn (see page 9583) objects and restores the default settings.

See Also

RestoreDefaults (see page 9591)

49.1.6.3.7 TDBGridColumns.SaveToFile Method

Saves the TDBGridColumns (see page 9746) object to the file specified in FileName.

Pascal

```
procedure SaveToFile(const Filename: string);
```

C++

```
__fastcall SaveToFile(const AnsiString Filename);
```

Description

Use the SaveToFile method to store TDBGridColumns (see page 9746) data to a file.

See Also

LoadFromFile (see page 9750)

49.1.6.3.8 TDBGridColumns.SaveToStream Method

Writes a stream with the name passed in Stream with the contents of the TDBGridColumns (see page 9746) object.

Pascal

```
procedure SaveToStream(S: TStream);
```

C++

```
__fastcall SaveToStream(TStream S);
```

Description

Use the SaveToStream method to stream out TDBGridColumns (see page 9746) data.

See Also

LoadFromStream (see page 9750)

49.1.6.3.9 TDBGridColumns.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed (see page 2740)

Changed (see page 2758)

49.1.6.4 TDBGridColumns Properties

The properties of the TDBGridColumns class are listed here.

Public Properties

	Name	Description
R	Grid (see page 9752)	Indicates the data-grid control to which the TDBGridColumns (see page 9746) instance belongs.
	Items (see page 9753)	Lists the columns in the collection.
	State (see page 9753)	Determines whether the columns in the data grid are dependent on dataset fields.

49.1.6.4.1 TDBGridColumns.Grid Property

Indicates the data-grid control to which the TDBGridColumns (see page 9746) instance belongs.

Pascal

```
property Grid: TCustomDBGrid;
```

C++

```
__property TCustomDBGrid Grid;
```

Description

The read-only Grid property indicates which data-grid control (TDBGrid (see page 9668)) owns the TDBGridColumns (see page 9746) object.

See Also

Columns (see page 9653)

49.1.6.4.2 TDBGridColumns.Items Property

Lists the columns in the collection.

Pascal

```
property Items [Index: Integer]: TColumn;
```

C++

```
__property TColumn Items[int Index];
```

Description

Use Items to access individual columns. The value of the Index parameter corresponds to the Index property of TColumn (see page 9583). It represents the position of the column in the data grid.

Items is the default property of TDBGridColumns (see page 9746). This means that the property name, Items, can be omitted when referring to columns of the TDBGridColumns (see page 9746) object. Thus, the line

```
FirstCol := DBGrid1.Columns.Items[0];
```

can also be written

```
FirstCol := DBGrid1.Columns[0];
```

See Also

Index (see page 2762)

operator_sb

49.1.6.4.3 TDBGridColumns.State Property

Determines whether the columns in the data grid are dependent on dataset fields.

Pascal

```
property State: TDBGridColumnsState;
```

C++

```
__property TDBGridColumnsState State;
```

Description

If State is set to csDefault, columns in the data grid are generated dynamically from information in the grid's associated dataset. The dataset fields determine the appearance, order, and other properties of the columns. Changes made to the columns are reflected in the dataset fields, and vice versa.

If State is set to csCustomized, many properties of the data-grid columns are independent of the associated dataset. For example, a column's title caption is, by default, the same as the associated field's DisplayLabel property; if State is csCustomized, however, the column title's Caption property and the field's DisplayLabel property can be changed independently. In fact, a customized (or "persistent") column need not be associated with a dataset field at all.

Note: If the Columns editor is used to configure data-grid columns, State is automatically set to csCustomized. Changing the

State property at runtime deletes all existing columns.

See Also

Items (see page 9753)

Columns (see page 9653)

DataSource (see page 9655)

AssignedValues (see page 9593)

Caption (see page 9607)

DisplayLabel (see page 7160)

Creating Persistent Columns

Using a Grid Control in Its Default State

49.1.7 TGridDataLink Class

TGridDataLink helps manage the link to a data source and maintain the field mapping for a TCustomDBGrid (see page 9608) object.

Class Hierarchy

Pascal

```
TGridDataLink = class(TDataLink);
```

C++

```
class TGridDataLink : public TDataLink;
```

File

DBGrids

Description

TGridDataLink is tailored to work with a TCustomDBGrid (see page 9608). It should only be used by the TCustomDBGrid (see page 9608) class.

See Also

TCustomDBGrid (see page 9608)

TDataSet (see page 6894)

TDataSource (see page 7070)

49.1.7.1 TGridDataLink Members

The following tables list the members exposed by TGridDataLink.

Protected Methods

	Name	Description
V	ActiveChanged (see page 9757)	Informs the TCustomDBGrid (see page 9608) of changes in the Active property.
	BuildAggMap (see page 9757)	Reserved for future use.
V	DataSetChanged (see page 9758)	Informs TCustomDBGrid (see page 9608) of changes to the dataset.
V	DataSetScrolled (see page 9758)	Informs the TCustomDBGrid (see page 9608) when the dataset has finished scrolling.
V	EditingChanged (see page 9759)	Informs the TCustomDBGrid (see page 9608) of changes in the editing state of the dataset.
V	FocusControl (see page 9759)	Displays the editor and gives focus to a field control in the TCustomDBGrid (see page 9608) object.
	GetMappedIndex (see page 9760)	GetMappedIndex provides the index of a field component in the dataset that corresponds to an index within the field map.
V	IsAggRow (see page 9760)	Reserved for future use.
V	LayoutChanged (see page 9760)	Informs the grid after the layout of the TCustomDBGrid (see page 9608) has changed in a way that affects the representation of the dataset.
V	RecordChanged (see page 9761)	Informs the grid of changes in the current record or field of the DataSource.
V	UpdateData (see page 9762)	Allows the grid to post changes to the dataset.

Public Methods

	Name	Description
	AddMapping (see page 9757)	Adds a field component to the end of the field map.
	ClearMapping (see page 9758)	Empties the field map and frees any associated memory.
	Create (see page 9755)	Creates an instance of TGridDataLink (see page 9754).
V	Destroy (see page 9756)	Destroys an instance of TGridDataLink (see page 9754).
	Modified (see page 9761)	Tracks changes to the data which have not been written to the current record in the database.
	Reset (see page 9761)	Cancels any pending edits to the dataset that have not been written to the current record in the database.

Public Properties

	Name	Description
R	DefaultFields (see page 9762)	Indicates whether the dataset's underlying field components are generated dynamically when the dataset is opened.
R	FieldCount (see page 9763)	Indicates the number of entries in the field map.
R	Fields (see page 9763)	Lists all the field components contained in the field map.
R	Grid (see page 9764)	Identifies the Grid component that uses this data link to manage its data.
	SparseMap (see page 9764)	Indicates whether the associated grid has customized the field map (which means it may contain undefined fields).

49.1.7.2 TGridDataLink.Create Constructor

Creates an instance of TGridDataLink (see page 9754).

Pascal

```
constructor Create(AGrid: TCustomDBGrid);
```

C++

```
__fastcall TGridDataLink(TCustomDBGrid AGrid);
```

Description

Applications should not call the Create method for a TGridDataLink (see page 9754) directly. The TCustomDBGrid (see page 9608) object that owns a TGridDataLink (see page 9754) calls Create from its own constructor.

See Also

Create (see page 6878)

Destroy (see page 9756)

49.1.7.3 TGridDataLink.Destroy Destructor

Destroys an instance of TGridDataLink (see page 9754).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TGridDataLink();
```

Description

Do not call Destroy directly in an application. The TCustomDBGrid (see page 9608) that owns a TGridDataLink (see page 9754) calls Free from its own destructor to destroy the TGridDataLink (see page 9754) object.

Destroy frees up the memory used by the field map before calling the inherited destructor.

See Also

Free (see page 22300)

Create (see page 9755)

49.1.7.4 TGridDataLink Methods

The methods of the TGridDataLink class are listed here.

Protected Methods

	Name	Description
	ActiveChanged (see page 9757)	Informs the TCustomDBGrid (see page 9608) of changes in the Active property.
	BuildAggMap (see page 9757)	Reserved for future use.
	DataSetChanged (see page 9758)	Informs TCustomDBGrid (see page 9608) of changes to the dataset.
	DataSetScrolled (see page 9758)	Informs the TCustomDBGrid (see page 9608) when the dataset has finished scrolling.
	EditingChanged (see page 9759)	Informs the TCustomDBGrid (see page 9608) of changes in the editing state of the dataset.
	FocusControl (see page 9759)	Displays the editor and gives focus to a field control in the TCustomDBGrid (see page 9608) object.
	GetMappedIndex (see page 9760)	GetMappedIndex provides the index of a field component in the dataset that corresponds to an index within the field map.
	IsAggRow (see page 9760)	Reserved for future use.
	LayoutChanged (see page 9760)	Informs the grid after the layout of the TCustomDBGrid (see page 9608) has changed in a way that affects the representation of the dataset.
	RecordChanged (see page 9761)	Informs the grid of changes in the current record or field of the DataSource.
	UpdateData (see page 9762)	Allows the grid to post changes to the dataset.

Public Methods

	Name	Description
	AddMapping (see page 9757)	Adds a field component to the end of the field map.
	ClearMapping (see page 9758)	Empties the field map and frees any associated memory.

	Modified (see page 9761)	Tracks changes to the data which have not been written to the current record in the database.
	Reset (see page 9761)	Cancels any pending edits to the dataset that have not been written to the current record in the database.

49.1.7.4.1 TGridDataLink.ActiveChanged Method

Informs the TCustomDBGrid (see page 9608) of changes in the Active property.

Pascal

```
procedure ActiveChanged; override;
```

C++

```
virtual __fastcall ActiveChanged();
```

Description

Applications cannot call the ActiveChanged procedure directly. Changes in the Active property trigger this procedure. When the dataset becomes inactive, ActiveChanged makes sure that the grid is not in edit mode. When the dataset becomes active, ActiveChanged makes sure that the grid does not contain references to fields that are no longer in the dataset, and that the grid does is not scrolled inappropriately.

See Also

Active (see page 6888)

TCustomDBGrid (see page 9608)

49.1.7.4.2 TGridDataLink.AddMapping Method

Adds a field component to the end of the field map.

Pascal

```
function AddMapping(const FieldName: string): Boolean;
```

C++

```
__fastcall Boolean AddMapping(const AnsiString FieldName);
```

Description

Applications should not call AddMapping directly, because this would cause the field map to become out of sync with the columns in TCustomDBGrid (see page 9608). TCustomDBGrid (see page 9608) calls AddMapping iteratively to build a field map that corresponds to its columns.

See Also

ClearMapping (see page 9758)

Fields (see page 9763)

49.1.7.4.3 TGridDataLink.BuildAggMap Method

Reserved for future use.

Pascal

```
procedure BuildAggMap;
```

C++

```
__fastcall BuildAggMap();
```

Description

BuildAggMap is not currently used. It is reserved for future support of aggregates in data-aware grids.

See Also

Aggregates (see page 8461)

49.1.7.4.4 TGridDataLink.ClearMapping Method

Empties the field map and frees any associated memory.

Pascal

```
procedure ClearMapping;
```

C++

```
__fastcall ClearMapping();
```

Description

Applications should not call ClearMapping directly, because this would cause the field map to become out of sync with the columns of the TCustomDBGrid (see page 9608). The TCustomDBGrid (see page 9608) calls ClearMapping first, every time it regenerates the field map.

See Also

AddMapping (see page 9757)

49.1.7.4.5 TGridDataLink.DataSetChanged Method

Informs TCustomDBGrid (see page 9608) of changes to the dataset.

Pascal

```
procedure DataSetChanged; override;
```

C++

```
virtual __fastcall DataSetChanged();
```

Description

Applications can not call DataSetChanged directly. Changes to the contents of the dataset trigger this method automatically. DataSetChanged allows the TCustomDBGrid (see page 9608) to keep track of when the contents of the dataset change.

See Also

ActiveChanged (see page 9757)

DataSetScrolled (see page 9758)

TDataSet (see page 6894)

49.1.7.4.6 TGridDataLink.DataSetScrolled Method

Informs the TCustomDBGrid (see page 9608) when the dataset has finished scrolling.

Pascal

```
procedure DataSetScrolled(Distance: Integer); override;
```

C++

```
virtual __fastcall DataSetScrolled(int Distance);
```

Description

Applications cannot call the DataSetScrolled procedure directly. This procedure is called automatically after the dataset for the TCustomDBGrid (see page 9608) scrolls. TGridDataLink (see page 9754) informs the TCustomDBGrid (see page 9608) that a scroll of the dataset has just occurred by calling its Scroll method.

See Also

DataSetChanged (see page 9758)

Scroll (see page 9648)

49.1.7.4.7 TGridDataLink.EditingChanged Method

Informs the TCustomDBGrid (see page 9608) of changes in the editing state of the dataset.

Pascal

```
procedure EditingChanged; override;
```

C++

```
virtual __fastcall EditingChanged();
```

Description

Applications cannot call the EditingChanged procedure directly. Changes in the Editing property trigger this procedure. EditingChanged allows the TCustomDBGrid (see page 9608) to provide visual feedback to the user about the editing state of the dataset.

See Also

Editing (see page 6892)

49.1.7.4.8 TGridDataLink.FocusControl Method

Displays the editor and gives focus to a field control in the TCustomDBGrid (see page 9608) object.

Pascal

```
procedure FocusControl(Field: TFieldRef); override;
```

C++

```
virtual __fastcall FocusControl(TFieldRef Field);
```

Description

Call the FocusControl procedure to bring up the editor for the representation of Field in the TCustomDBGrid (see page 9608). FocusControl selects Field in the TCustomDBGrid (see page 9608), and then tells the TCustomDBGrid (see page 9608) object to bring up the editor.

See Also

TabStop

ShowEditor (see page 14839)

49.1.7.4.9 TGridDataLink.GetMappedIndex Method

GetMappedIndex provides the index of a field component in the dataset that corresponds to an index within the field map.

Pascal

```
function GetMappedIndex(ColIndex: Integer): Integer;
```

C++

```
__fastcall int GetMappedIndex(int ColIndex);
```

Description

Use GetMappedIndex on an index into the field map to get an index into the dataset for the same field component. Both the field map and the dataset are zero-based arrays. If an entry in the field map is an undefined field, GetMappedIndex returns -1.

See Also

Fields (see page 9763)

49.1.7.4.10 TGridDataLink.IsAggRow Method

Reserved for future use.

Pascal

```
function IsAggRow(Value: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsAggRow(int Value);
```

Description

IsAggRow indicates whether the row specified by Value contains aggregate values rather than data. Value is the index of a row in the cache of rows currently displayed.

IsAggRow always returns false because data-aware grids do not currently display aggregate values. This method is reserved for future support of aggregates in data-aware grids.

See Also

Aggregates (see page 8461)

49.1.7.4.11 TGridDataLink.LayoutChanged Method

Informs the grid after the layout of the TCustomDBGrid (see page 9608) has changed in a way that affects the representation of the dataset.

Pascal

```
procedure LayoutChanged; override;
```

C++

```
virtual __fastcall LayoutChanged();
```

Description

Applications cannot call the LayoutChanged method directly. The TCustomDBGrid (see page 9608) that owns the TGridDataLink (see page 9754) calls this method automatically when layout changes affect the representation of the dataset. LayoutChanged allows the TCustomDBGrid (see page 9608) to regenerate its layout by calling its LayoutChanged method.

See Also

DataSetChanged (see page 9758)

LayoutChanged (see page 9645)

49.1.7.4.12 TGridDataLink.Modified Method

Tracks changes to the data which have not been written to the current record in the database.

Pascal

```
procedure Modified;
```

C++

```
__fastcall Modified();
```

Description

Applications should not call Modified directly. Modified is called automatically whenever the TCustomDBGrid (see page 9608) starts editing one of its fields. Modified allows the TGridDataLink (see page 9754) to keep track of whether there are any edits that have not been posted.

See Also

Edit (see page 6882)

Editing (see page 6892)

Reset (see page 9761)

49.1.7.4.13 TGridDataLink.RecordChanged Method

Informs the grid of changes in the current record or field of the DataSource.

Pascal

```
procedure RecordChanged(Field: TField); override;
```

C++

```
virtual __fastcall RecordChanged(TField Field);
```

Description

Applications cannot call RecordChanged directly. It is called automatically when the contents of the current record change. The Field parameter indicates which field in the current record has changed. If Field is nil (Delphi) or NULL (C++), the entire record has changed.

RecordChanged informs TCustomDBGrid (see page 9608) of these changes so that it can update the display of the affected column or columns.

See Also

DataSetChanged (see page 9758)

49.1.7.4.14 TGridDataLink.Reset Method

Cancels any pending edits to the dataset that have not been written to the current record in the database.

Pascal

```
procedure Reset;
```

C++

```
__fastcall Reset();
```

Description

Reset is triggered by user actions that cancel any pending edits to the dataset.

See Also

Modified (see page 9761)

UpdateData (see page 9762)

49.1.7.4.15 TGridDataLink.UpdateData Method

Allows the grid to post changes to the dataset.

Pascal

```
procedure UpdateData; override;
```

C++

```
virtual __fastcall UpdateData();
```

Description

Applications cannot call the UpdateData procedure directly. It is called automatically when pending changes need to be posted to the dataset. UpdateData tells the TCustomDBGrid (see page 9608) to write any unposted changes to the current field to the underlying Field component.

See Also

DataSetChanged (see page 9758)

Modified (see page 9761)

Reset (see page 9761)

TField (see page 7115)

49.1.7.5 TGridDataLink Properties

The properties of the TGridDataLink class are listed here.

Public Properties

	Name	Description
R	DefaultFields (see page 9762)	Indicates whether the dataset's underlying field components are generated dynamically when the dataset is opened.
R	FieldCount (see page 9763)	Indicates the number of entries in the field map.
R	Fields (see page 9763)	Lists all the field components contained in the field map.
R	Grid (see page 9764)	Identifies the Grid component that uses this data link to manage its data.
	SparseMap (see page 9764)	Indicates whether the associated grid has customized the field map (which means it may contain undefined fields).

49.1.7.5.1 TGridDataLink.DefaultFields Property

Indicates whether the dataset's underlying field components are generated dynamically when the dataset is opened.

Pascal

```
property DefaultFields: Boolean;
```

C++

```
__property Boolean DefaultFields;
```

Description

Read DefaultFields to determine whether or not a dataset uses dynamically generated field components or persistent field components. If DefaultFields is true, the dataset uses dynamically allocated field components. If DefaultFields is false, the dataset uses persistent field components or the custom field mapping contains unmapped fields.

If DefaultFields is false, the TCustomDBGrid (see page 9608) may not show all the fields that are present in the underlying database table.

See Also

SparseMap (see page 9764)

49.1.7.5.2 TGridDataLink.FieldCount Property

Indicates the number of entries in the field map.

Pascal

```
property FieldCount: Integer;
```

C++

```
__property int FieldCount;
```

Description

Read FieldCount to determine the number of entries in the field map. Used with the Fields (see page 9763) property, this is useful for applications that need to iterate over all of the mapped fields.

FieldCount is the same as the number of columns in the TCustomDBGrid (see page 9608). The field map can contain undefined fields, which do not appear in the dataset. Undefined fields in the field map stand in for fields in the columns of the grid that are not in the dataset, such as calculated fields.

See Also

AddMapping (see page 9757)

DataSet (see page 6890)

Fields (see page 9763)

SparseMap (see page 9764)

49.1.7.5.3 TGridDataLink.Fields Property

Lists all the field components contained in the field map.

Pascal

```
property Fields [I: Integer]: TField;
```

C++

```
__property TField Fields[int I];
```

Description

Use Fields to access field components by index number. Fields is a zero-based array. If the SparseMap (see page 9764) property is true, the field map may contain undefined fields. For these entries, Fields returns nil (Delphi) or NULL (C++).

See Also

AddMapping (see page 9757)

FieldCount (see page 9763)

SparseMap (see page 9764)

49.1.7.5.4 TGridDataLink.Grid Property

Identifies the Grid component that uses this data link to manage its data.

Pascal

```
property Grid: TCustomDBGrid;
```

C++

```
__property TCustomDBGrid Grid;
```

Description

Use Grid to access the data-aware grid that owns this data link. The value of Grid is assigned by the TGridDataLink (see page 9754) constructor.

See Also

Create (see page 9755)

49.1.7.5.5 TGridDataLink.SparseMap Property

Indicates whether the associated grid has customized the field map (which means it may contain undefined fields).

Pascal

```
property SparseMap: Boolean;
```

C++

```
__property Boolean SparseMap;
```

Description

Read SparseMap to determine whether the field map is not a simple reflection of the underlying dataset. The TCustomDBGrid (see page 9608) sets SparseMap when it builds the field map. SparseMap is false if the TCustomDBGrid (see page 9608) builds a default field map directly from the dataset. Any changes to the field components associated with each column cause TCustomDBGrid (see page 9608) to change SparseMap to true. When SparseMap is true, the field map may contain undefined fields if the column attributes indicate an invalid field name or an unassigned field component.

See Also

AddMapping (see page 9757)

Fields (see page 9763)

49.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TColumnButtonStyle (see page 9765)	TColumnButtonStyle indicates whether and how the user can select values for a column from a list.
	TColumnValue (see page 9766)	TColumnValue and TColumnValues (see page 9769) indicate which properties on the column of a DB (see page 6725) grid are persistent.
	TDBGridColumnsState (see page 9767)	TDBGridColumnsState indicates whether the columns in a DB (see page 6725) grid dynamically reflect dataset fields.
	TDBGridOption (see page 9767)	TDBGridOption and TDBGridOptions (see page 9770) indicate display and behavioral property of a DB (see page 6725) grid.

49.2.1 DBGrids.TColumnButtonStyle Enumeration

TColumnButtonStyle indicates whether and how the user can select values for a column from a list.

Pascal

```
TColumnButtonStyle = (
  cbsAuto,
  cbsEllipsis,
  cbsNone
);
```

C++

```
enum TColumnButtonStyle {
  cbsAuto,
  cbsEllipsis,
  cbsNone
};
```

File

DBGrids

Description

These are the possible values of TColumnButtonStyle:

Value	Meaning
cbsAuto	If the column is associated with a lookup field or has a value assigned to its PickList property, the grid displays a combo box in the column. The user can choose a value from the drop-down list.
cbsEllipsis	The column displays an ellipsis button that the user can click to choose a value. Clicking the ellipsis button triggers an OnEditButtonClick event.
cbsNone	No combo box or ellipsis button is provided. The user cannot select the column's content from a list.

49.2.2 DBGrids.TColumnValue Enumeration

TColumnValue and TColumnValues (see page 9769) indicate which properties on the column of a DB (see page 6725) grid are persistent.

Pascal

```
TColumnValue = (
  cvColor,
  cvWidth,
  cvFont,
  cvAlignment,
  cvReadOnly,
  cvTitleColor,
  cvTitleCaption,
  cvTitleAlignment,
  cvTitleFont,
  cvImeMode,
  cvImeName
);
```

C++

```
enum TColumnValue {
  cvColor,
  cvWidth,
  cvFont,
  cvAlignment,
  cvReadOnly,
  cvTitleColor,
  cvTitleCaption,
  cvTitleAlignment,
  cvTitleFont,
  cvImeMode,
  cvImeName
};
```

File

DBGrids

Description

TColumnValues (see page 9769) is a set that indicates which properties of a column have been set as persistent properties (overriding the defaults). It can contain zero or more of the following TColumnValue values:

Value	Indicates changes to
cvColor	The column's Color property.
cvWidth	The column's Width property.
cvFont	The column's Font property.
cvAlignment	The column's Alignment property.
cvReadOnly	The column's ReadOnly property.
cvTitleColor	The column title's Color property.
cvTitleCaption	The column title's Caption property.
cvTitleAlignment	The column title's Alignment property.
cvTitleFont	The column title's Font property.

cvImeMode	The column's ImeMode property.
cvImeName	The column's ImeName property.

49.2.3 DBGrids.TDBGridColumnsState Enumeration

TDBGridColumnsState indicates whether the columns in a DB (see page 6725) grid dynamically reflect dataset fields.

Pascal

```
TDBGridColumnsState = (
  csDefault,
  csCustomized
);
```

C++

```
enum TDBGridColumnsState {
  csDefault,
  csCustomized
};
```

File

DBGrids

Description

The following table lists the possible values.

Value	Meaning
csDefault	Columns in the data grid are generated dynamically from information in the grid's associated dataset. The dataset fields determine the appearance, order, and other properties of the columns. Changes made to the columns are reflected in the dataset fields, and vice versa.
csCustomized	Properties of the data-grid columns are independent of the associated dataset.

49.2.4 DBGrids.TDBGridOption Enumeration

TDBGridOption and TDBGridOptions (see page 9770) indicate display and behavioral property of a DB (see page 6725) grid.

Pascal

```
TDBGridOption = (
  dgEditing,
  dgAlwaysShowEditor,
  dgTitles,
  dgIndicator,
  dgColumnResize,
  dgColLines,
  dgRowLines,
  dgTabs,
  dgRowSelect,
  dgAlwaysShowSelection,
  dgConfirmDelete,
  dgCancelOnExit,
  dgMultiSelect
```

```
);
```

C++

```
enum TDBGridOption {
  dgEditing,
  dgAlwaysShowEditor,
  dgTitles,
  dgIndicator,
  dgColumnResize,
  dgColLines,
  dgRowLines,
  dgTabs,
  dgRowSelect,
  dgAlwaysShowSelection,
  dgConfirmDelete,
  dgCancelOnExit,
  dgMultiSelect
};
```

File

DBGrids

Description

TDBGridOptions (see page 9770) is a set drawn from the following TDBGridOption values:

Value	Meaning
dgEditing	The user can edit data using the grid. dgEditing is ignored if Options includes dgRowSelect.
dgAlwaysShowEditor	The grid is always in edit mode. That is, the user does not have to press Enter or F2 before editing the contents of a cell. dgAlwaysShowEditor does nothing unless dgEditing is also included in Options. dgAlwaysShowEditor is ignored if Options includes dgRowSelect.
dgTitles	Titles appear at the top of the columns in the grid.
dgIndicator	A small pointer appears in the first column to indicate which row is current.
dgColumnResize	Columns that are bound to fields can be resized or moved.
dgColLines	Lines appear between columns in the grid.
dgRowLines	Lines appear between the rows of the grid.
dgTabs	The user can navigate through the grid using the Tab and Shift+Tab keys.
dgRowSelect	The user can select an entire row, as well as selecting individual cells. If Options includes dgRowSelect, dgEditing and dgAlwaysShowEditor are ignored.
dgAlwaysShowSelection	The selected cell displays the focus rectangle even when the grid does not have focus.
dgConfirmDelete	A message box appears, asking for confirmation, when the user types Ctrl+Delete to delete a row in the grid.
dgCancelOnExit	When the user exits the grid from an inserted record to which the user made no modifications, the inserted record is not posted to the dataset. This prevents the inadvertent posting of empty records.
dgMultiSelect	More than one row in the grid can be selected at a time.

49.3 Types

The following table lists types in this documentation.

Types

Name	Description
TColumnClass (see page 9769)	TColumnClass defines the metaclass for TColumn (see page 9583).
TColumnValues (see page 9769)	This is type DBGrids.TColumnValues.
TDBGridClickEvent (see page 9770)	This is type DBGrids.TDBGridClickEvent.
TDBGridOptions (see page 9770)	This is type DBGrids.TDBGridOptions.
TDrawColumnCellEvent (see page 9770)	This is type DBGrids.TDrawColumnCellEvent.
TDrawDataCellEvent (see page 9771)	This is type DBGrids.TDrawDataCellEvent.

49.3.1 DBGrids.TColumnClass Type

TColumnClass defines the metaclass for TColumn (see page 9583).

Pascal

```
TColumnClass = class of TColumn;
```

C++

```
TColumn TColumnClass;
```

File

DBGrids

Description

TColumnClass is the metaclass for TColumn (see page 9583). Its value is the class reference for TColumn (see page 9583) or for one of its descendants.

49.3.2 DBGrids.TColumnValues Type

Pascal

```
TColumnValues = set of TColumnValue;
```

C++

```
TColumnValue TColumnValues;
```

File

DBGrids

Description

This is type DBGrids.TColumnValues.

49.3.3 DBGrids.TDBGridClickEvent Type

Pascal

```
TDBGridClickEvent = procedure (Column: TColumn) of object;
```

C++

```
(Column: TColumn) ( TDBGridClickEvent)();
```

File

DBGrids

Description

This is type DBGrids.TDBGridClickEvent.

49.3.4 DBGrids.TDBGridOptions Type

Pascal

```
TDBGridOptions = set of TDBGridOption;
```

C++

```
TDBGridOption TDBGridOptions;
```

File

DBGrids

Description

This is type DBGrids.TDBGridOptions.

49.3.5 DBGrids.TDrawColumnCellEvent Type

Pascal

```
TDrawColumnCellEvent = procedure (Sender: TObject; const Rect: TRect; DataCol: Integer;
Column: TColumn; State: TGridDrawState) of object;
```

C++

```
(Sender: TObject; const Rect: TRect; DataCol: Integer; Column: TColumn; State: TGridDrawState)
( TDrawColumnCellEvent)();
```

File

DBGrids

Description

This is type DBGrids.TDrawColumnCellEvent.

49.3.6 DBGrids.TDrawDataCellEvent Type

Pascal

```
TDrawDataCellEvent = procedure (Sender: TObject; const Rect: TRect; Field: TField; State:
TGridDrawState) of object;
```

C++

```
(Sender: TObject; const Rect: TRect; Field: TField; State: TGridDrawState) (
TDrawDataCellEvent)();
```

File

DBGrids

Description

This is type DBGrids.TDrawDataCellEvent.

49.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
cm_DeferLayout (see page 9772)	This is constant DBGrids.cm_DeferLayout.
ColumnTitleValues (see page 9772)	This is constant DBGrids.ColumnTitleValues.
IndicatorWidth (see page 9772)	This is constant DBGrids.IndicatorWidth.

49.4.1 DBGrids.cm_DeferLayout Constant

Pascal

```
cm_DeferLayout = WM_USER + 100;
```

C++

```
cm_DeferLayout = WM_USER + 100;
```

File

DBGrids

Description

This is constant DBGrids.cm_DeferLayout.

49.4.2 DBGrids.ColumnTitleValues Constant

Pascal

```
ColumnTitleValues = [cvTitleColor..cvTitleFont];
```

C++

```
ColumnTitleValues = [cvTitleColor..cvTitleFont];
```

File

DBGrids

Description

This is constant DBGrids.ColumnTitleValues.

49.4.3 DBGrids.IndicatorWidth Constant

Pascal

```
IndicatorWidth = 11;
```

C++

```
IndicatorWidth = 11;
```

File

DBGrids

Description

This is constant DBGrids.IndicatorWidth.

50 DBLogDlg Namespace

This is namespace DBLogDlg.

Topics

Name	Description
Classes (see page 9776)	The following table lists classes in this documentation.
Functions (see page 9797)	The following table lists functions in this documentation.

Classes

	Name	Description
	TLoginDialog (see page 9776)	TLoginDialog is a login dialog box that can be used to provide login security in an application.

Functions

	Name	Description
	LoginDialog (see page 9797)	Brings up the database Login dialog to allow the user to connect to a database server.
	LoginDialogEx (see page 9798)	Brings up the database Login dialog to allow the user to connect to a database server.
	RemoteLoginDialog (see page 9798)	Brings up the database Login dialog to allow the user to connect to a database server.
	SetCursorType (see page 9799)	This is function DBLogDlg.SetCursorType.

50.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TLoginDialog (see page 9776)	TLoginDialog is a login dialog box that can be used to provide login security in an application.

50.1.1 TLoginDialog Class

TLoginDialog is a login dialog box that can be used to provide login security in an application.

Class Hierarchy

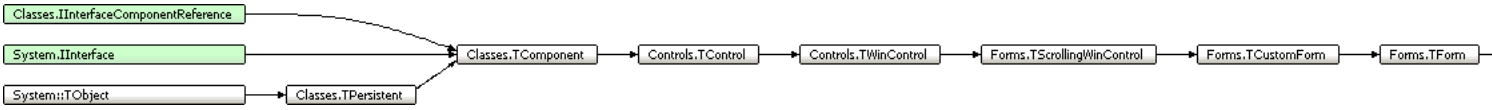

Pascal

```
TLoginDialog = class(TForm);
```

C++

```
class TLoginDialog : public TForm;
```

File

DBLogDlg

Description

Use TLoginDialog to provide login security within an application. Usually a TLoginDialog object is created with the LoginDialogEx (see page 9798) or RemoteLoginDialog (see page 9798) function and is used to provide secure access to local database tables. However, TLoginDialog can also be used as a base object for other types of login dialog boxes.

See Also

LoginDialogEx (see page 9798)

TPasswordDialog (see page 9828)

RemoteLoginDialog (see page 9798)

50.1.1.1 TLoginDialog Members

The following tables list the members exposed by TLoginDialog.

Protected Events

	Name	Description
	OnActivate (see page 13943)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.

	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnCanResize (see page 13947)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnClick (see page 13947)	Occurs when the user clicks the control.
	OnClose (see page 13949)	Occurs when the form closes.
	OnCloseQuery (see page 13950)	Occurs when close is attempted.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 13951)	Occurs when the form is created.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDblClick (see page 13953)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 13953)	Occurs when the form loses focus.
	OnDestroy (see page 13953)	Occurs when the form is destroyed.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragDrop (see page 13954)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDragOver (see page 13957)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnHelp (see page 13959)	Occurs when the form receives a request for help.
	OnHide (see page 13960)	Occurs when the form is hidden (that is, when its Visible (see page 13940) property is set to false).
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyDown (see page 13962)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyPress (see page 13964)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnKeyUp (see page 13966)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseActivate (see page 13967)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseDown (see page 13968)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseMove (see page 13971)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseUp (see page 13973)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.

	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnPaint (see page 13975)	Occurs when the form is redrawn.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnResize (see page 13976)	Occurs immediately after the control is resized.
	OnShortCut (see page 13977)	Occurs when the user presses a key (before the OnKeyDown (see page 13962) event).
	OnShow (see page 13977)	Occurs when the form is shown (that is, when its Visible (see page 13940) property is set to true).
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFormState (see page 13863)	This is FFormState, a member of class TCustomForm.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	Activate (see page 13865)	Activation event dispatcher.
	ActiveChanged (see page 13865)	Active (see page 13896) state event dispatcher.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 14324)	Adjusts the ClientRect property to exclude the scroll bars.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 13866)	Aligns all child controls of the form that fall within a specified rectangle.
V	AlignControls (see page 14325)	Aligns the controls in the scrolling windowed control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
V	AutoScrollEnabled (see page 14325)	Indicates whether automatic scrolling is enabled.
V	AutoScrollInView (see page 14325)	Scrolls a control into the visible area of the scrolling windowed control.

	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	BeginAutoDrag (see page 13867)	Responds when the user starts dragging the form if DragMode is dmAutomatic.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	ChangeScale (see page 13867)	Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.
	ChangeScale (see page 14326)	Resizes the scrolling windowed control by a specific ratio.
	Click (see page 5743)	Respond to user click.
	CloseModal (see page 13868)	This is CloseModal, a member of class TCustomForm.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 13868)	Initializes the window-creation parameter record when the form window is created.
	CreateParams (see page 14326)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWindowHandle (see page 13869)	Creates a form window given a set of window-creation parameters.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 13869)	Creates the form window.
	CreateWnd (see page 14327)	Creates a Windows control corresponding to the TScrollingWinControl (see page 14312) object.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	Deactivate (see page 13870)	Generates an OnDeactivate (see page 13953) event.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DefineProperties (see page 13871)	Designates methods for storing a form's unpublished data on a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyHandle (see page 13871)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWindowHandle (see page 13872)	Destroys the window created in the CreateWindowHandle (see page 13869) method.
	DestroyWnd (see page 6141)	Destroys the control's window.

	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoClose (see page 13873)	Form closing event dispatcher.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
V	DoCreate (see page 13873)	Form creation event dispatcher.
V	DoDestroy (see page 13873)	Form destruction event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
V	DoDock (see page 13874)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
V	DoFlipChildren (see page 14327)	Flips the control's children.
	DoHide (see page 13874)	Form hide event dispatcher.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoShow (see page 13875)	Form show event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorderIconStyles (see page 13875)	This is GetBorderIconStyles, a member of class TCustomForm.
	GetBorderStyles (see page 13876)	This is GetBorderStyles, a member of class TCustomForm.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetChildren (see page 13876)	Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetClientRect (see page 13876)	Returns the value of the ClientRect property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.

	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloating (see page 13877)	Returns the value of the Floating property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetOwnerWindow (see page 13878)	This is GetOwnerWindow, a member of class TCustomForm.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 13878)	Handles exceptions that are raised by the OnCreate (see page 13951) event handler.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	Loaded (see page 13880)	Performs fix-ups when the form is first loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	Notification (see page 13882)	Responds to notifications that objects are about to be inserted or removed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 13882)	Generates an OnPaint (see page 13975) event.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaintWindow (see page 13882)	Renders the image of the form.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.

	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteChanged (see page 13883)	Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	QueryInterface (see page 13884)	Returns a reference to a specified interface if that interface is supported by the form.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	ReadState (see page 13884)	Reads the form's published properties, stored data, and owned components from a reader object.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	RequestAlign (see page 13885)	Instructs the form's parent (if any) to reposition the form, enforcing its Align property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	Resizing (see page 13885)	Makes any required adjustments when the form changes size.
	Resizing (see page 14328)	This is Resizing, a member of class TScrollingWinControl.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetChildOrder (see page 13886)	Changes the order in which Child appears when GetChildren (see page 13876) is called.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParent (see page 13887)	Sets the Parent property.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.

V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
V	SetParentBiDiMode (see page 13887)	Sets the ParentBiDiMode property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
V	UpdateActions (see page 13891)	Updates all actions associated with the form.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	UpdateWindowState (see page 13892)	Updates the WindowState (see page 13942) property to reflect the current state of the form.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	ValidateRename (see page 13892)	Determines if a form can rename one of its owned components.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	VisibleChanging (see page 13893)	Responds just before the form changes the value of its Visible (see page 13940) property.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WndProc (see page 13893)	Provides specific message responses for the form.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
R	ActiveMDIChild (see page 13897)	Specifies the MDI child that has focus.
	AlphaBlend (see page 13898)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 13898)	Specifies the degree of translucency on a translucent form.
	AutoScroll (see page 13899)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoScroll (see page 14329)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderIcons (see page 13899)	Specifies which icons appear on the title bar of the form.

	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
R	ClientHandle (see page 13904)	Provides access to the handle of the form's internal MDI client window.
W	ClientHeight (see page 13905)	Specifies the height (in pixels) of the form's client area.
W	ClientWidth (see page 13905)	Specifies the width (in pixels) of the form's client area.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Ctl3D (see page 13909)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultMonitor (see page 13909)	Specifies the monitor on which the form appears.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	FormStyle (see page 13912)	Determines the form's style.
	Height (see page 13915)	Specifies the vertical size of the control in pixels.
	HorzScrollBar (see page 13916)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 13917)	Specifies the icon that appears when the form is minimized.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	MDIChildCount (see page 13921)	Specifies the number of open MDI child forms.
R	MDIChildren (see page 13922)	Provides indexed access to all MDI child forms.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ObjectMenuItem (see page 13926)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 13926)	Specifies when OnCreate (see page 13951) and OnDestroy (see page 13953) events occur.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentFont (see page 13927)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PixelsPerInch (see page 13928)	Represents the proportion of the font on the system on which the form was designed.
R	PopupChildren (see page 13928)	This is PopupChildren, a member of class TCustomForm.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	PopupMenu (see page 13928)	Identifies the pop-up menu associated with the control.
	Position (see page 13933)	Represents the size and placement of the form.
	PrintScale (see page 13934)	Represents the proportions of a printed form.
	Scaled (see page 13935)	Specifies whether the form is sized according to the value of the PixelsPerInch (see page 13928) property.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	TileMode (see page 13936)	Represents how MDI child forms are arranged when the Tile method is called.
	TransparentColor (see page 13938)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 13939)	Indicates the color on the form that appears transparent when TransparentColor (see page 13938) is true.
	VertScrollBar (see page 13939)	Represents the vertical scroll bar for the scrolling windowed control.

	Visible (see page 13940)	Indicates whether the form is visible.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	Width (see page 13941)	Specifies the horizontal size of the control or form in pixels.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowMenu (see page 13941)	Specifies the Window menu for an MDI parent form.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Fields

	Name	Description
	Bevel (see page 9793)	This is Bevel, a member of class TLoginDialog.
	CancelButton (see page 9793)	This is CancelButton, a member of class TLoginDialog.
	DatabaseName (see page 9793)	This is DatabaseName, a member of class TLoginDialog.
	Label1 (see page 9794)	This is Label1, a member of class TLoginDialog.
	Label2 (see page 9794)	This is Label2, a member of class TLoginDialog.
	Label3 (see page 9794)	This is Label3, a member of class TLoginDialog.
	OKButton (see page 9794)	This is OKButton, a member of class TLoginDialog.
	Panel (see page 9794)	This is Panel, a member of class TLoginDialog.
	Panel1 (see page 9795)	This is Panel1, a member of class TLoginDialog.
	Password (see page 9795)	This is Password, a member of class TLoginDialog.
	UserName (see page 9795)	This is UserName, a member of class TLoginDialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 13866)	Construction event dispatcher.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	ArrangeIcons (see page 14006)	Arranges the icons of minimized MDI child forms.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 13866)	Destruction event dispatcher.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	Cascade (see page 14007)	Arranges MDI child forms so they overlap.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Close (see page 13867)	Closes the form.
	CloseQuery (see page 13868)	Close (see page 13867) attempt event dispatcher.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.

	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 5731)	Creates an instance of TControl (see page 5724).
	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 13861)	Creates and initializes a new TForm (see page 13992) object.
	Create (see page 14323)	Creates and initializes a new TScrollingWinControl (see page 14312) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateNew (see page 13862)	Creates and initializes a new form.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 13870)	Provides message handling for all messages that the form does not fully process.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DefocusControl (see page 13871)	Removes focus from a control on the form.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 13862)	Removes the form from memory.
	Destroy (see page 14323)	Removes the TScrollingWinControl (see page 14312) object from memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	DisableAutoRange (see page 14327)	Disables automatic scrolling.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	Dock (see page 13872)	Handles automatic docking of the form into other windows.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EnableAutoRange (see page 14328)	Reenables automatic scrolling.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	FocusControl (see page 13875)	Sets focus to a control on the form.
	Focused (see page 6150)	Determines whether the control has input focus.
	FormShow (see page 9795)	Adjusts the contents of the login dialog when it becomes visible.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.

	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFormImage (see page 13877)	Returns a bitmap of the form.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	Hide (see page 13879)	Hides the form.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	IsShortCut (see page 13880)	Processes shortcut keys when the form has focus.
	MakeFullyVisible (see page 13881)	Ensures that the form is fully visible on a specified monitor.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MouseWheelHandler (see page 13881)	Directs mouse wheel messages to the appropriate control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 14008)	Activates the next child form in the form sequence.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	Previous (see page 14009)	Activates the previous MDI child form in the form sequence.
	Print (see page 13883)	Prints the form.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	RecreateAsPopup (see page 13884)	This is RecreateAsPopup, a member of class TCustomForm.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.

	Release (see page 13885)	Destroys the form and frees its associated memory.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollInView (see page 14328)	Scrolls a control into the visible area of the scrolling windowed control.
	SendCancelMode (see page 13886)	Cancels modes on the form.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
V	SetFocus (see page 13886)	Sets focus to the form.
V	SetFocusedControl (see page 13887)	Sets focus to a control on the form.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Show (see page 13888)	Shows the form.
V	ShowModal (see page 13890)	Shows a form as a modal dialog.
	Tile (see page 14009)	Arranges MDI child forms so that they are all the same size.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UpdateGlassFrame (see page 13892)	This is UpdateGlassFrame, a member of class TCustomForm.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.
V	WantChildKey (see page 13893)	Indicates whether the form processes keyboard input for a control it owns.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Action (see page 13895)	Designates the action associated with the control.
R	Active (see page 13896)	Specifies whether the form has focus.
	ActiveControl (see page 13896)	Specifies the control that has focus on the form.
	ActiveMDIChild (see page 14013)	Specifies the MDI child that has focus.
	ActiveOleControl (see page 13897)	Specifies the OLE control on the form that reacts to changes in focus.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 13900)	Specifies the appearance and behavior of the form border.

	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 13900)	Provides access to the drawing area of the form.
	Caption (see page 13902)	Specifies a text string that identifies the control to the user.
	ClientHandle (see page 14024)	Provides access to the handle of the form's internal MDI client window.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 13908)	Specifies the background color of the control.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designer (see page 13910)	Specifies the designer interface for the form.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockManager (see page 14029)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropTarget (see page 13910)	Specifies whether the form is the target of a drag-and-drop operation.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Font (see page 13910)	Controls the attributes of text written on or in the control.
R	FormState (see page 13911)	Indicates transitional state information about the form.
	GlassFrame (see page 13914)	This is GlassFrame, a member of class TCustomForm.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HelpFile (see page 13916)	Specifies the name of the file the form uses to display Help.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	KeyPreview (see page 13918)	Specifies whether the form should receive keyboard events before the active control.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	MDIChildCount (see page 14044)	Specifies the number of open MDI child forms.

	MDIChildren (see page 14044)	Provides indexed access to all MDI child forms.
	Menu (see page 13923)	Specifies the form's main menu.
	ModalResult (see page 13924)	Represents the return value of a form that is used as a modal dialog.
R	Monitor (see page 13926)	Provides access to the monitor on which the form appears.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	OleFormObject (see page 13927)	Specifies the IOleForm (see page 14340) interface for an in-place OLE object contained in the form.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PopupMode (see page 13933)	This is PopupMode, a member of class TCustomForm.
	PopupParent (see page 13933)	Sets an order for stacked forms that users cannot change.
	ScreenSnap (see page 13935)	Specifies whether form snaps to edge of screen.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	SnapBuffer (see page 13936)	Specifies distance for screen snap.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TileMode (see page 14059)	Represents how MDI child forms are arranged when the Tile method is called.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.
	WindowState (see page 13942)	Represents how the form appears on the screen.

Published Events

	Name	Description
	OnActivate (see page 14065)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 14069)	Occurs when an object with custom alignment is aligned.
	OnAlignInsertBefore (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14069)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 14070)	Occurs when an attempt is made to resize the control.
	OnClick (see page 14070)	Occurs when the user clicks the control.
	OnClose (see page 14072)	Occurs when the form closes.
	OnCloseQuery (see page 14073)	Occurs when close is attempted.
	OnConstrainedResize (see page 14074)	Adjust resize constraints.
	OnContextPopup (see page 14075)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 14076)	Occurs when the form is created.
	OnDblClick (see page 14077)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 14078)	Occurs when the form loses focus.
	OnDestroy (see page 14078)	Occurs when the form is destroyed.
	OnDockDrop (see page 14079)	Occurs when another control is docked to the control.
	OnDockOver (see page 14081)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 14083)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 14085)	Occurs when the user drags an object over a control.
	OnEndDock (see page 14088)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

	OnGetSiteInfo (see page 14088)	Returns the control's docking information.
	OnHelp (see page 14090)	Occurs when the form receives a request for help.
	OnHide (see page 14091)	Occurs when the form is hidden (that is, when its Visible property is set to false).
	OnKeyDown (see page 14093)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 14095)	Occurs when a key is pressed.
	OnKeyUp (see page 14096)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 14098)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 14099)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 14101)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 14102)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 14102)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 14104)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 14107)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 14107)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 14107)	Occurs when the mouse wheel is rotated upward.
	OnPaint (see page 14108)	Occurs when the form is redrawn.
	OnResize (see page 14109)	Occurs immediately after the control is resized.
	OnShortCut (see page 14109)	Occurs when the user presses a key (before the OnKeyDown event).
	OnShow (see page 14110)	Occurs when the form is shown (that is, when its Visible property is set to true).
	OnStartDock (see page 14111)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnUnDock (see page 14111)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Action (see page 14011)	Designates the action associated with the control.
	ActiveControl (see page 14012)	Specifies the control that has focus on the form.
	Align (see page 14013)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AlphaBlend (see page 14017)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 14017)	Specifies the degree of translucency on a translucent form.
	Anchors (see page 14017)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 14018)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 14018)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BiDiMode (see page 14019)	Specifies the bi-directional mode for the control.
	BorderIcons (see page 14019)	Specifies which icons appear on the title bar of the form.
	BorderStyle (see page 14020)	Specifies the appearance and behavior of the form border.
	BorderWidth (see page 14021)	Specifies the width of the control's border.
	Caption (see page 14021)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 14024)	Specifies the height (in pixels) of the form's client area.
	ClientWidth (see page 14025)	Specifies the width (in pixels) of the form's client area.
	Color (see page 14027)	Specifies the background color of the control.
	Constraints (see page 14028)	Specifies the size constraints for the control.
	Ctl3D (see page 14028)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DefaultMonitor (see page 14029)	Specifies the monitor on which the form appears.
	DockSite (see page 14031)	Specifies whether the control can be the target of drag-and-dock operations.
	DragKind (see page 14032)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 14033)	Determines how the control initiates drag-and-drop or drag-and-dock operations.

	Enabled (see page 14034)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 14035)	Controls the attributes of text written on or in the control.
	FormStyle (see page 14036)	Determines the form's style.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	Height (see page 14038)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpFile (see page 14039)	Specifies the name of the file the form uses to display Help.
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HorzScrollBar (see page 14040)	Represents the horizontal scroll bar for the scrolling windowed control.
	HorzScrollBar (see page 14329)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 14041)	Specifies the icon that appears when the form is minimized.
	KeyPreview (see page 14041)	Specifies whether the form should receive keyboard events before the active control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Left (see page 13920)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Menu (see page 14045)	Specifies the form's main menu.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectMenuItem (see page 14047)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 14047)	Specifies when OnCreate and OnDestroy events occur.
	Padding (see page 14047)	Specifies the padding of a control.
	ParentBiDiMode (see page 14048)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 14048)	Determines where a control looks for its font information.
	PixelsPerInch (see page 14049)	Represents the proportion of the font on the system on which the form was designed.
	PopupMenu (see page 14049)	Identifies the pop-up menu associated with the control.
	PopupMode (see page 14054)	Controls (see page 5711) how the top-level form behaves with respect to Window's WS_POPUP style.
	PopupParent (see page 14054)	Sets an order for stacked forms that users cannot change.
	Position (see page 14055)	Represents the size and placement of the form.
	PrintScale (see page 14056)	Represents the proportions of a printed form.
	Scaled (see page 14056)	Specifies whether the form is sized according to the value of the PixelsPerInch property.
	ScreenSnap (see page 14056)	Specifies whether form snaps to edge of screen.
	ShowHint (see page 14057)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SnapBuffer (see page 14058)	Specifies distance for screen snap.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Top (see page 13937)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	TransparentColor (see page 14060)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 14060)	Indicates the color on the form that appears transparent when TransparentColor is true.
	UseDockManager (see page 14060)	Specifies whether the docking manager is used in drag-and-dock operations.
	VertScrollBar (see page 14061)	Represents the vertical scroll bar for the scrolling windowed control.
	VertScrollBar (see page 14330)	Represents the vertical scroll bar for the scrolling windowed control.
	Visible (see page 14062)	Indicates whether the form is visible.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	Width (see page 14063)	Specifies the horizontal size of the control or form in pixels.
	WindowMenu (see page 14063)	Specifies the Window menu for an MDI parent form.

	WindowState (see page 14063)	Represents how the form appears on the screen.

50.1.1.2 TLoginDialog Fields

The fields of the TLoginDialog class are listed here.

Public Fields

	Name	Description
	Bevel (see page 9793)	This is Bevel, a member of class TLoginDialog.
	CancelButton (see page 9793)	This is CancelButton, a member of class TLoginDialog.
	DatabaseName (see page 9793)	This is DatabaseName, a member of class TLoginDialog.
	Label1 (see page 9794)	This is Label1, a member of class TLoginDialog.
	Label2 (see page 9794)	This is Label2, a member of class TLoginDialog.
	Label3 (see page 9794)	This is Label3, a member of class TLoginDialog.
	OKButton (see page 9794)	This is OKButton, a member of class TLoginDialog.
	Panel (see page 9794)	This is Panel, a member of class TLoginDialog.
	Panel1 (see page 9795)	This is Panel1, a member of class TLoginDialog.
	Password (see page 9795)	This is Password, a member of class TLoginDialog.
	UserName (see page 9795)	This is UserName, a member of class TLoginDialog.

50.1.1.2.1 TLoginDialog.Bevel Field

Pascal

```
Bevel: TBevel;
```

C++

```
TBevel Bevel;
```

Description

This is Bevel, a member of class TLoginDialog.

50.1.1.2.2 TLoginDialog.CancelButton Field

Pascal

```
CancelButton: TButton;
```

C++

```
TButton CancelButton;
```

Description

This is CancelButton, a member of class TLoginDialog.

50.1.1.2.3 TLoginDialog.DatabaseName Field

Pascal

```
DatabaseName: TLabel;
```

C++

```
TLabel DatabaseName;
```

Description

This is DatabaseName, a member of class TLoginDialog.

50.1.1.2.4 TLoginDialog.Label1 Field

Pascal

```
Label1: TLabel;
```

C++

```
TLabel Label1;
```

Description

This is Label1, a member of class TLoginDialog.

50.1.1.2.5 TLoginDialog.Label2 Field

Pascal

```
Label2: TLabel;
```

C++

```
TLabel Label2;
```

Description

This is Label2, a member of class TLoginDialog.

50.1.1.2.6 TLoginDialog.Label3 Field

Pascal

```
Label3: TLabel;
```

C++

```
TLabel Label3;
```

Description

This is Label3, a member of class TLoginDialog.

50.1.1.2.7 TLoginDialog.OKButton Field

Pascal

```
OKButton: TButton;
```

C++

```
TButton OKButton;
```

Description

This is OKButton, a member of class TLoginDialog.

50.1.1.2.8 TLoginDialog.Panel Field

Pascal

```
Panel: TPanel;
```

C++

```
TPanel Panel;
```

Description

This is Panel, a member of class TLoginDialog.

50.1.1.2.9 TLoginDialog.Panel1 Field

Pascal

```
Panel1: TPanel;
```

C++

```
TPanel Panel1;
```

Description

This is Panel1, a member of class TLoginDialog.

50.1.1.2.10 TLoginDialog.Password Field

Pascal

```
Password: TEdit;
```

C++

```
TEdit Password;
```

Description

This is Password, a member of class TLoginDialog.

50.1.1.2.11 TLoginDialog.UserName Field

Pascal

```
UserName: TEdit;
```

C++

```
TEdit UserName;
```

Description

This is UserName, a member of class TLoginDialog.

50.1.1.3 TLoginDialog Methods

The methods of the TLoginDialog class are listed here.

Public Methods

	Name	Description
	FormShow (see page 9795)	Adjusts the contents of the login dialog when it becomes visible.

50.1.1.3.1 TLoginDialog.FormShow Method

Adjusts the contents of the login dialog when it becomes visible.

Pascal

```
procedure FormShow(Sender: TObject);
```

C++

```
__fastcall FormShow(TObject * Sender);
```

Description

FormShow is called automatically just before the login dialog becomes visible. It fixes up the controls in the login dialog so that the text appears properly formatted.

50.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	LoginDialog (see page 9797)	Brings up the database Login dialog to allow the user to connect to a database server.
	LoginDialogEx (see page 9798)	Brings up the database Login dialog to allow the user to connect to a database server.
	RemoteLoginDialog (see page 9798)	Brings up the database Login dialog to allow the user to connect to a database server.
	SetCursorType (see page 9799)	This is function DBLogDlg.SetCursorType.

50.2.1 DBLogDlg.LoginDialog Function

Brings up the database Login dialog to allow the user to connect to a database server.

Pascal

```
function LoginDialog(const ADatabaseName: string; var AUserName: string; var APassword:
string): Boolean;
```

C++

```
Boolean LoginDialog(const AnsiString ADatabaseName, AnsiString AUserName, AnsiString
APassword);
```

File

DBLogDlg

Description

Call LoginDialog to bring up the standard Login dialog box to prompt the user for a valid user name and password. The ADatabaseName parameter specifies the database for which access is being requested. The dialog returns the user name and password as the AUserName and APassword parameters.

Note: When building cross-platform applications in C++, add QDBLogDlg.pas unit (instead of DBLogDlg (see page 9775)) to your project if you want the default Database Login dialog to display at runtime. QDBLogDlg does not need to be included if your C++ application includes a DBGrid component.

LoginDialog returns false if the user cancels out of the dialog. If LoginDialog returns true, the application can attempt to connect to the database server using the supplied user name and password.

Note: If the user should not be permitted to change the user name from within the login dialog, use the LoginDialogEx (see page 9798) function.

See Also

LoginDialogEx

RemoteLoginDialog

50.2.2 DBLogDlg.LoginDialogEx Function

Brings up the database Login dialog to allow the user to connect to a database server.

Pascal

```
function LoginDialogEx(const ADatabaseName: string; var AUserName: string; var APassword:
string; NameReadOnly: Boolean): Boolean;
```

C++

```
Boolean LoginDialogEx(const AnsiString ADatabaseName, AnsiString AUserName, AnsiString
APassword, Boolean NameReadOnly);
```

File

DBLogDlg

Description

Call LoginDialogEx to bring up the standard Login dialog box to prompt the user for a valid user name and password. The ADatabaseName parameter specifies the database for which access is being requested. The NameReadOnly parameter specifies whether the user can change the user name from within the dialog.

The dialog returns the user name and password as the AUserName and APassword parameters.

LoginDialogEx returns falseif the user cancels out of the dialog. If LoginDialogEx returns true, the application can attempt to connect to the database server using the supplied user name and password.

Note: When building cross-platform applications in C++, add QDBLogDlg.pas unit (instead of DBLogDlg (see page 9775)) to your project if you want the default Database Login dialog to display at runtime. QDBLogDlg does not need to be included if your C++ application includes a DBGrid component.

See Also

LoginDialog

PasswordDialog

RemoteLoginDialog

50.2.3 DBLogDlg.RemoteLoginDialog Function

Brings up the database Login dialog to allow the user to connect to a database server.

Pascal

```
function RemoteLoginDialog(var AUserName: string; var APassword: string): Boolean;
```

C++

```
Boolean RemoteLoginDialog(AnsiString AUserName, AnsiString APassword);
```

File

DBLogDlg

Description

Call RemoteLoginDialog to bring up the standard Login dialog box to prompt the user for a valid user name and password.

The dialog returns the user name and password as the AUserName and APassword parameters.

RemotLoginDialog returns false if the user cancels out of the dialog. If RemoteLoginDialog returns true, the application can attempt to connect to the database server using the supplied user name and password.

Note: When building cross-platform applications in C++, add QDBLogDlg.pas unit (instead of DBLogDlg (see page 9775)) to your project if you want the default Database Login dialog to display at runtime. QDBLogDlg does not need to be included if your C++ application includes a DBGrid component.

See Also

LoginDialog

PasswordDialog

LoginDialogEx

50.2.4 DBLogDlg.SetCursorType Function

Pascal

```
procedure SetCursorType(const CurIndex: Integer);
```

C++

```
SetCursorType(const int CurIndex);
```

File

DBLogDlg

Description

This is function DBLogDlg.SetCursorType.

51 DBOleCtl Namespace

This is namespace DBOleCtl.

Topics

Name	Description
Classes (see page 9802)	The following table lists classes in this documentation.

Classes

	Name	Description
	TDataBindings (see page 9802)	TDataBindings holds a group of TDataBindItem (see page 9806) objects.
	TDataBindItem (see page 9806)	TDataBindItem represents a data binding in a data-aware ActiveX control.
	TDBOleControl (see page 9810)	Supports simple data binding for ActiveX controls.

51.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TDataBindings (see page 9802)	TDataBindings holds a group of TDataBindItem (see page 9806) objects.
	TDataBindItem (see page 9806)	TDataBindItem represents a data binding in a data-aware ActiveX control.
	TDBOleControl (see page 9810)	Supports simple data binding for ActiveX controls.

51.1.1 TDataBindings Class

TDataBindings holds a group of TDataBindItem (see page 9806) objects.

Class Hierarchy

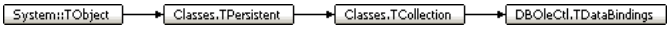

Pascal

```
TDataBindings = class(TCollection);
```

C++

```
class TDataBindings : public TCollection;
```

File

DBOleCtl

Description

Use TDataBindings to connect the properties of a data-aware ActiveX control to the fields in a dataset. TDataBindings maintains an index of data bindings (TDataBindItem (see page 9806) objects) in its Items (see page 9806) array. The Count property of its ancestor TCollection, contains the number of items in the collection. Use the Add (see page 9804) method to add items to the collection.

Typically, this object is for internal use. Instead, you access an object's data bindings through the DataBindings property of the DBOleControl object.

51.1.1.1 TDataBindings Members

The following tables list the members exposed by TDataBindings.

Protected Methods

	Name	Description
	Added (see page 2736)	Responds when items are added to the collection.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	Deleting (see page 2741)	Responds when items are deleted from the collection.

	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 9804)	Creates a new TDataBindItem (see page 9806) object and adds it to the end of the Items (see page 9806) property array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 9804)	Creates an instance of TDataBindings (see page 9802).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItemByDispID (see page 9805)	Returns the associated TDataBindItem (see page 9806) property for the specified DispID.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.

	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetOwner (see page 9805)	Returns the owner of an object.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Update (see page 9805)	For internal use.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 9806)	Provides indexed access to the TDataBindItem (see page 9806) objects in the collection.

51.1.1.2 TDataBindings.Create Constructor

Creates an instance of TDataBindings (see page 9802).

Pascal

```
constructor Create(DBOleControl: TDBOleControl);
```

C++

```
__fastcall TDataBindings(TDBOleControl DBOleControl);
```

Description

Applications should not need to call Create directly. The data-aware control object that uses the TDataBindings (see page 9802) collection to represent its data bindings creates the TDataBindings (see page 9802) object from its own constructor.

51.1.1.3 TDataBindings Methods

The methods of the TDataBindings class are listed here.

Public Methods

	Name	Description
	Add (see page 9804)	Creates a new TDataBindItem (see page 9806) object and adds it to the end of the Items (see page 9806) property array.
	GetItemByDispID (see page 9805)	Returns the associated TDataBindItem (see page 9806) property for the specified DispID.
V	GetOwner (see page 9805)	Returns the owner of an object.
V	Update (see page 9805)	For internal use.

51.1.1.3.1 TDataBindings.Add Method

Creates a new TDataBindItem (see page 9806) object and adds it to the end of the Items (see page 9806) property array.

Pascal

```
function Add: TDataBindItem;
```

C++

```
__fastcall TDataBindItem Add();
```

Description

Call Add to add a new TDataBindItem (see page 9806) to the set of data bindings associated with the specified data-aware control. With this method, you must also set the fields within the TDataBindItem (see page 9806) at the same time. These data bindings can be modified programmatically at a later time.

See Also

Add (see page 2736)

51.1.1.3.2 TDataBindings.GetItemByDispID Method

Returns the associated TDataBindItem (see page 9806) property for the specified DispID.

Pascal

```
function GetItemByDispID(ADispID: TDispID): TDataBindItem;
```

C++

```
__fastcall TDataBindItem GetItemByDispID(TDispID ADispID);
```

Description

GetItemByDispID returns the TDataBindItem (see page 9806) property for the specified DispID. It returns nil (Delphi) or NULL (C++) if no TDataBindItem (see page 9806) property exists for the specified DispID.

51.1.1.3.3 TDataBindings.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used to find the owner of a data binding, that is, the data-aware control.

See Also

GetOwner (see page 2932)

51.1.1.3.4 TDataBindings.Update Method

For internal use.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Update refreshes the field definitions in the Items (see page 9806) property array to reflect the current state of the physical fields in the underlying dataset. This is typically for internal use.

51.1.1.4 TDataBindings Properties

The properties of the TDataBindings class are listed here.

Public Properties

	Name	Description
	Items (see page 9806)	Provides indexed access to the TDataBindItem (see page 9806) objects in the collection.

51.1.1.4.1 TDataBindings.Items Property

Provides indexed access to the TDataBindItem (see page 9806) objects in the collection.

Pascal

```
property Items [Index: Integer]: TDataBindItem;
```

C++

```
__property TDataBindItem Items[int Index];
```

Description

Use Items to access the properties of TDataBindItem (see page 9806) objects maintained by TDataBindings (see page 9802).

Note: In Delphi code, Items is the default property of TDataBindings (see page 9802), so the name of the property can be left off when using this property. Thus, the line:

```
FirstDataBinding := DBOleCtl1.DataBindings.Items[0];
```

Note: can be written as:

```
FirstDataBinding := DBOleCtl1.DataBindings[0];
```

See Also

TDBOleControl (see page 9810)

TDataBindItem (see page 9806)

operator_[]

51.1.2 TDataBindItem Class

TDataBindItem represents a data binding in a data-aware ActiveX control.

Class Hierarchy

Pascal

```
TDataBindItem = class(TCollectionItem);
```

C++

```
class TDataBindItem : public TCollectionItem;
```

File

DBOleCtl

Description

The TDataBindItem object stores the association between a property in a data-aware control and a field in the dataset. TDataBindings (see page 9802) holds a collection of data bindings for a particular ActiveX control, which is represented by a TDBOleControl (see page 9810) object.

51.1.2.1 TDataBindItem Members

The following tables list the members exposed by TDataBindItem.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 9808)	Creates an instance of TDataBindItem (see page 9806).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 9808)	Destroys the TDataBindItem (see page 9806) instance and frees its memory.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	DataField (see page 9809)	Represents the name of the field to be bound.
	DispID (see page 9809)	The identifier (DispID) of the data-aware control property to be bound.

51.1.2.2 TDataBindItem.Create Constructor

Creates an instance of TDataBindItem (see page 9806).

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TDataBindItem(TCollection Collection);
```

Description

Do not create an instance of TDataBindItem (see page 9806) directly. Instead, an instance is created when the Add method of the TDataBindings (see page 9802) object is called.

See Also

Add (see page 9804)

51.1.2.3 TDataBindItem.Destroy Destructor

Destroys the TDataBindItem (see page 9806) instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataBindItem();
```

Description

Destroy is called indirectly by TCollection's Clear method.

See Also

Free (see page 22300)

51.1.2.4 TDataBindItem Properties

The properties of the TDataBindItem class are listed here.

Published Properties

	Name	Description
	DataField (see page 9809)	Represents the name of the field to be bound.
	DispID (see page 9809)	The identifier (DispID) of the data-aware control property to be bound.

51.1.2.4.1 TDataBindItem.DataField Property

Represents the name of the field to be bound.

Pascal

```
property DataField: string;
```

C++

```
__property AnsiString DataField;
```

Description

DataField relates a property in a data-aware ActiveX control to a specific field in the dataset. The dataset is specified by the DataSource property of TDBOleControl (see page 9810) object.

See Also

DataSource (see page 9825)

TDataBindings (see page 9802)

51.1.2.4.2 TDataBindItem.DispID Property

The identifier (DispID) of the data-aware control property to be bound.

Pascal

```
property DispID: TDispID;
```

C++

```
__property TDispID DispID;
```

Description

Every data-aware property in an ActiveX control has a unique DispID. The collection of databindings is represented by the DataBindings property of TDBOleControl (see page 9810) object.

See Also

DataBindings (see page 9825)

TDataBindings (see page 9802)

51.1.3 TDBOleControl Class

Supports simple data binding for ActiveX controls.

Class Hierarchy

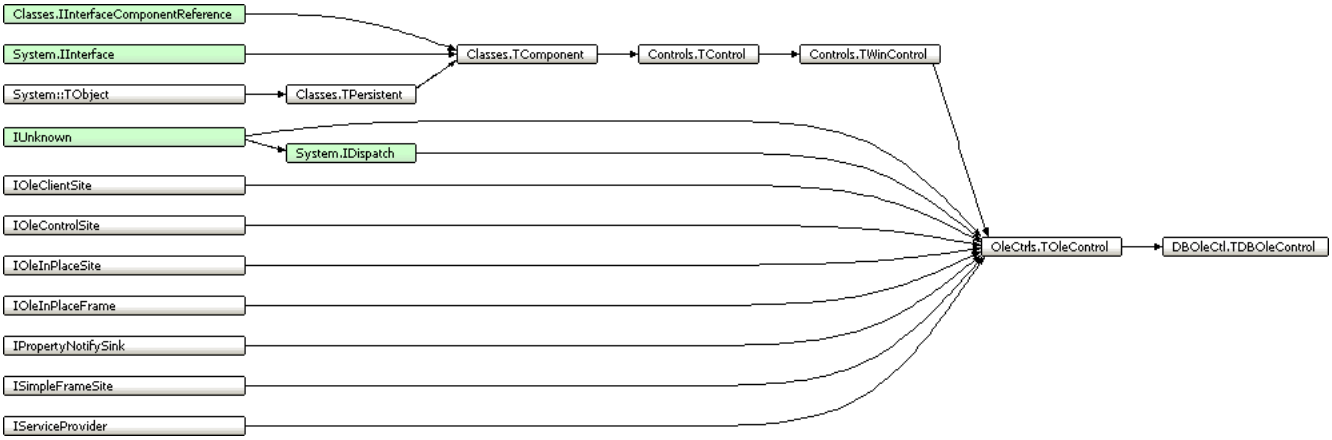

Pascal

```
TDBOleControl = class(TOleControl);
```

C++

```
class TDBOleControl : public TOleControl;
```

File

DBOleCtl

Description

Use TBDOleControl to import ActiveX controls that contain simple bindable properties. With simple data binding, you can hook up properties of ActiveX controls to a data source.

Applications typically do not use TDBOleControl; it is available through the ActiveX control wizard and the ActiveForm wizard. Once you create an ActiveX control using either wizard, a type library is created. Edit the type library binding properties to add simple data binding to your applications.

51.1.3.1 TDBOleControl Members

The following tables list the members exposed by TDBOleControl.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_AddRef (see page 18046)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.

	_Release (see page 18047)	Called when an application releases a component interface.
	_SetColorProp (see page 18047)	This is _SetColorProp, a member of class TOleControl.
	_SetTColorProp (see page 18048)	This is _SetTColorProp, a member of class TOleControl.
	_SetTFontProp (see page 18048)	This is _SetTFontProp, a member of class TOleControl.
	_SetTOleEnumProp (see page 18048)	This is _SetTOleEnumProp, a member of class TOleControl.
	_SetTPictureProp (see page 18048)	This is _SetTPictureProp, a member of class TOleControl.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanInPlaceActivate (see page 18049)	This is CanInPlaceActivate, a member of class TOleControl.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ContextSensitiveHelp (see page 18049)	This is ContextSensitiveHelp, a member of class TOleControl.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CreateWnd (see page 18049)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DeactivateAndUndo (see page 18050)	This is DeactivateAndUndo, a member of class TOleControl.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.

	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DefineProperties (see page 18050)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWindowHandle (see page 18051)	Destroys the window created in the CreateWindowHandle method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DiscardUndoState (see page 18051)	This is DiscardUndoState, a member of class TOleControl.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EnableModeless (see page 18052)	This is EnableModeless, a member of class TOleControl.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorder (see page 18052)	This is GetBorder, a member of class TOleControl.
	GetByteProp (see page 18052)	This is GetByteProp, a member of class TOleControl.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.

	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetColorProp (see page 18053)	This is GetColorProp, a member of class TOleControl.
	GetCompProp (see page 18053)	This is GetCompProp, a member of class TOleControl.
	GetContainer (see page 18053)	This is GetContainer, a member of class TOleControl.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetCurrencyProp (see page 18053)	This is GetCurrencyProp, a member of class TOleControl.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDoubleProp (see page 18053)	This is GetDoubleProp, a member of class TOleControl.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetExtendedControl (see page 18054)	This is GetExtendedControl, a member of class TOleControl.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDispatchProp (see page 18055)	This is GetIDispatchProp, a member of class TOleControl.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIDsOfNames (see page 18055)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIntegerProp (see page 18055)	This is GetIntegerProp, a member of class TOleControl.
	GetIUnknownProp (see page 18055)	This is GetIUnknownProp, a member of class TOleControl.
	GetMoniker (see page 18056)	This is GetMoniker, a member of class TOleControl.
	GetOleBoolProp (see page 18056)	This is GetOleBoolProp, a member of class TOleControl.
	GetOleDateProp (see page 18056)	This is GetOleDateProp, a member of class TOleControl.
	GetOleEnumProp (see page 18057)	This is GetOleEnumProp, a member of class TOleControl.
	GetOleVariantProp (see page 18057)	This is GetOleVariantProp, a member of class TOleControl.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetProperty (see page 18058)	This is GetProperty, a member of class TOleControl.
	GetShortIntProp (see page 18058)	This is GetShortIntProp, a member of class TOleControl.
	GetSingleProp (see page 18058)	This is GetSingleProp, a member of class TOleControl.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetSmallintProp (see page 18058)	This is GetSmallintProp, a member of class TOleControl.
	GetStringProp (see page 18059)	This is GetStringProp, a member of class TOleControl.
	GetTColorProp (see page 18059)	This is GetTColorProp, a member of class TOleControl.
	GetTDateTimeProp (see page 18059)	This is GetTDateTimeProp, a member of class TOleControl.
	GetTFontProp (see page 18059)	This is GetTFontProp, a member of class TOleControl.
	GetTOleEnumProp (see page 18059)	This is GetTOleEnumProp, a member of class TOleControl.
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTPictureProp (see page 18060)	This is GetTPictureProp, a member of class TOleControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfo (see page 18060)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetTypeInfoCount (see page 18060)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariantProp (see page 18061)	This is GetVariantProp, a member of class TOleControl.
	GetWideStringProp (see page 18061)	This is GetWideStringProp, a member of class TOleControl.
	GetWindowContext (see page 18061)	This is GetWindowContext, a member of class TOleControl.

	GetWordBoolProp (see page 18061)	This is GetWordBoolProp, a member of class TOleControl.
	GetWordProp (see page 18062)	This is GetWordProp, a member of class TOleControl.
A	InitControlData (see page 18062)	This is InitControlData, a member of class TOleControl.
V	InitControlInterface (see page 18062)	This is InitControlInterface, a member of class TOleControl.
	InsertMenus (see page 18062)	This is InsertMenus, a member of class TOleControl.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Invoke (see page 18062)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	InvokeEvent (see page 18063)	This is InvokeEvent, a member of class TOleControl.
	InvokeMethod (see page 18063)	This is InvokeMethod, a member of class TOleControl.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	LockInPlaceActive (see page 18064)	This is LockInPlaceActive, a member of class TOleControl.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 9824)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	OleControlSite_TranslateAccelerator (see page 18064)	This is OleControlSite_TranslateAccelerator, a member of class TOleControl.
	OleInPlaceFrame_GetWindow (see page 18064)	This is OleInPlaceFrame_GetWindow, a member of class TOleControl.
	OleInPlaceFrame_TranslateAccelerator (see page 18065)	This is OleInPlaceFrame_TranslateAccelerator, a member of class TOleControl.
	OleInPlaceSite_GetWindow (see page 18065)	This is OleInPlaceSite_GetWindow, a member of class TOleControl.
V	OnChanged (see page 9824)	This is OnChanged, a member of class TDBOleControl.
	OnControlInfoChanged (see page 18065)	This is OnControlInfoChanged, a member of class TOleControl.
	OnFocus (see page 18066)	This is OnFocus, a member of class TOleControl.
	OnInPlaceActivate (see page 18066)	This is OnInPlaceActivate, a member of class TOleControl.
	OnInPlaceDeactivate (see page 18066)	This is OnInPlaceDeactivate, a member of class TOleControl.
	OnPosRectChange (see page 18066)	This is OnPosRectChange, a member of class TOleControl.
V	OnRequestEdit (see page 9825)	This is OnRequestEdit, a member of class TDBOleControl.
	OnShowWindow (see page 18067)	This is OnShowWindow, a member of class TOleControl.
	OnUIActivate (see page 18067)	This is OnUIActivate, a member of class TOleControl.
	OnUIDeactivate (see page 18067)	This is OnUIDeactivate, a member of class TOleControl.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
V	PaletteChanged (see page 18067)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PictureChanged (see page 18068)	This is PictureChanged, a member of class TOleControl.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	PostMessageFilter (see page 18068)	This is PostMessageFilter, a member of class TOleControl.

	PreMessageFilter (see page 18069)	This is PreMessageFilter, a member of class TOleControl.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	QueryInterface (see page 18069)	Returns a reference to a specified interface if the object supports that interface.
	QueryService (see page 18069)	This is QueryService, a member of class TOleControl.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveMenus (see page 18070)	This is RemoveMenus, a member of class TOleControl.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	RequestBorderSpace (see page 18070)	This is RequestBorderSpace, a member of class TOleControl.
	RequestNewObjectLayout (see page 18070)	This is RequestNewObjectLayout, a member of class TOleControl.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	SaveObject (see page 18070)	This is SaveObject, a member of class TOleControl.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Scroll (see page 18070)	This is Scroll, a member of class TOleControl.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActiveObject (see page 18071)	This is SetActiveObject, a member of class TOleControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetBorderSpace (see page 18071)	This is SetBorderSpace, a member of class TOleControl.
	SetByteProp (see page 18071)	This is SetByteProp, a member of class TOleControl.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetColorProp (see page 18072)	This is SetColorProp, a member of class TOleControl.
	SetCompProp (see page 18072)	This is SetCompProp, a member of class TOleControl.
	SetCurrencyProp (see page 18072)	This is SetCurrencyProp, a member of class TOleControl.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDoubleProp (see page 18072)	This is SetDoubleProp, a member of class TOleControl.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIDispatchProp (see page 18073)	This is SetIDispatchProp, a member of class TOleControl.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetIntegerProp (see page 18073)	This is SetIntegerProp, a member of class TOleControl.
	SetIUnknownProp (see page 18073)	This is SetIUnknownProp, a member of class TOleControl.
	SetMenu (see page 18073)	This is SetMenu, a member of class TOleControl.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.

	SetName (see page 5785)	Sets the name of the control.
	SetName (see page 18073)	Sets the name of the control.
	SetOleBoolProp (see page 18074)	This is SetOleBoolProp, a member of class TOleControl.
	SetOleDateProp (see page 18074)	This is SetOleDateProp, a member of class TOleControl.
	SetOleEnumProp (see page 18074)	This is SetOleEnumProp, a member of class TOleControl.
	SetOleVariantProp (see page 18075)	This is SetOleVariantProp, a member of class TOleControl.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParent (see page 18075)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetProperty (see page 18076)	This is SetProperty, a member of class TOleControl.
	SetShortIntProp (see page 18076)	This is SetShortIntProp, a member of class TOleControl.
	SetSingleProp (see page 18076)	This is SetSingleProp, a member of class TOleControl.
	SetSmallintProp (see page 18076)	This is SetSmallintProp, a member of class TOleControl.
	SetStatusText (see page 18077)	This is SetStatusText, a member of class TOleControl.
	SetStringProp (see page 18077)	This is SetStringProp, a member of class TOleControl.
	SetTColorProp (see page 18077)	This is SetTColorProp, a member of class TOleControl.
	SetTDateTimeProp (see page 18077)	This is SetTDateTimeProp, a member of class TOleControl.
	SetTFontProp (see page 18077)	This is SetTFontProp, a member of class TOleControl.
	SetTOleEnumProp (see page 18078)	This is SetTOleEnumProp, a member of class TOleControl.
	SetTPictureProp (see page 18078)	This is SetTPictureProp, a member of class TOleControl.
	SetVariantProp (see page 18078)	This is SetVariantProp, a member of class TOleControl.
	SetWideStringProp (see page 18078)	This is SetWideStringProp, a member of class TOleControl.
	SetWordBoolProp (see page 18079)	This is SetWordBoolProp, a member of class TOleControl.
	SetWordProp (see page 18079)	This is SetWordProp, a member of class TOleControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	ShowObject (see page 18079)	This is ShowObject, a member of class TOleControl.
	ShowPropertyFrame (see page 18079)	This is ShowPropertyFrame, a member of class TOleControl.
	StandardEvent (see page 18080)	This is StandardEvent, a member of class TOleControl.
	SuppressException (see page 18080)	This is SuppressException, a member of class TOleControl.
	TransformCoords (see page 18080)	This is TransformCoords, a member of class TOleControl.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.

	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WndProc (see page 18080)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	ControlData (see page 18086)	This is ControlData, a member of class TOleControl.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	ServiceQuery (see page 18087)	This is ServiceQuery, a member of class TOleControl.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	BrowseProperties (see page 18048)	Notifies the ActiveX control to display its Properties dialog box.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 9823)	Creates an instance of TDBOleControl (see page 9810).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 18050)	Handles all otherwise-unhandled messages.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 9823)	Destroys an instance of TDBOleControl (see page 9810).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DoObjectVerb (see page 18051)	Notifies the ActiveX control to perform one of its actions.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.

	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetEnumPropDesc (see page 18054)	Returns descriptions of the ActiveX control's properties.
	GetHelpContext (see page 18054)	Retrieves the Help context of a member of the ActiveX control.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetObjectVerbs (see page 18056)	Retrieves the names of the verbs implemented by the ActiveX control.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetPropDisplayString (see page 18057)	Returns the display string for an interface for the ActiveX control.
	GetPropDisplayStrings (see page 18057)	Returns the display strings for an interface for the ActiveX control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	IsCustomProperty (see page 18063)	Indicates whether an interface for the ActiveX control is a custom property.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsPropPageProperty (see page 18064)	Indicates whether an interface for the OCX control is a property page property.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.

	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetBounds (see page 18071)	Sets the boundaries of the ActiveX control.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetDesignVisible (see page 6178)	Makes a control visible at design time.
	SetFocus (see page 6179)	Gives the input focus to the control.
	SetPropDisplayString (see page 18075)	Sets the display string for an interface for the ActiveX control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	ShowAboutBox (see page 18079)	Notifies the ActiveX control to display its About box.
	Update (see page 5789)	Processes any pending paint messages immediately.
	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	Anchors (see page 18085)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.

R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
R	DefaultDispatch (see page 18086)	The default dispatch interface.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	OleObject (see page 18086)	The ActiveX control associated with the TOleControl (see page 18030) object.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	PerPropBrowsing (see page 18086)	Provides access to information about the property page properties of the OleControl.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TabStop (see page 18087)	Determines if the user can tab to a control.

	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DataBindings (see page 9825)	Represents the data bindings between ActiveX control properties and fields in a dataset.
	DataSource (see page 9825)	Specifies the link to the dataset where the data-aware ActiveX control finds its data.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

51.1.3.2 TDBOleControl.Create Constructor

Creates an instance of TDBOleControl (see page 9810).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDBOleControl(TComponent * AOwner);
```

Description

Use Create to create a new instance of TDBOleControl (see page 9810) object. Create calls the inherited constructor, then creates and initializes TDataBinding objects.

See Also

Create (see page 18043)

DataBindings (see page 9825)

51.1.3.3 TDBOleControl.Destroy Destructor

Destroys an instance of TDBOleControl (see page 9810).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBOleControl();
```

Description

Do not call Destroy. Instead call Free, which calls Destroy if this TDBOleControl (see page 9810) reference is not nil.

See Also

Free (see page 22300)

51.1.3.4 TDBOleControl Methods

The methods of the TDBOleControl class are listed here.

Protected Methods

	Name	Description
	Notification (see page 9824)	Responds to notifications that components are being created or destroyed.
	OnChanged (see page 9824)	This is OnChanged, a member of class TDBOleControl.
	OnRequestEdit (see page 9825)	This is OnRequestEdit, a member of class TDBOleControl.

51.1.3.4.1 TDBOleControl.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

51.1.3.4.2 TDBOleControl.OnChanged Method

Pascal

```
function OnChanged(dispid: TDispID): HResult; override;
```

C++

```
virtual __fastcall HRESULT OnChanged(TDispID dispid);
```

Description

This is OnChanged, a member of class TDBOleControl.

51.1.3.4.3 TDBOleControl.OnRequestEdit Method

Pascal

```
function OnRequestEdit(dispid: TDispID): HResult; override;
```

C++

```
virtual __fastcall HRESULT OnRequestEdit(TDispID dispid);
```

Description

This is OnRequestEdit, a member of class TDBOleControl.

51.1.3.5 TDBOleControl Properties

The properties of the TDBOleControl class are listed here.

Published Properties

	Name	Description
	DataBindings (see page 9825)	Represents the data bindings between ActiveX control properties and fields in a dataset.
	DataSource (see page 9825)	Specifies the link to the dataset where the data-aware ActiveX control finds its data.

51.1.3.5.1 TDBOleControl.DataBindings Property

Represents the data bindings between ActiveX control properties and fields in a dataset.

Pascal

```
property DataBindings: TDataBindings;
```

C++

```
__property TDataBindings DataBindings;
```

Description

Use DataBindings to read or set the data bindings in a data-aware ActiveX control. DataBindings is a collection of TDataBindItem (see page 9806) objects. Use the properties of the TDataBindings (see page 9802) object to specify the data bindings of individual properties in the data-aware control.

Data bindings are typically set by editing the ActiveX control's type library.

See Also

DataSource (see page 9825)

51.1.3.5.2 TDBOleControl.DataSource Property

Specifies the link to the dataset where the data-aware ActiveX control finds its data.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Set DataSource to the TDataSource object that links to the dataset where the data-aware object should fetch its data. DataSource allows the data-aware control to read from, write to, and navigate around the dataset.

See Also

DataBindings (see page 9825)

52 DBPWDlg Namespace

This is namespace DBPWDlg.

Topics

Name	Description
Classes (see page 9828)	The following table lists classes in this documentation.
Functions (see page 9849)	The following table lists functions in this documentation.

Classes

	Name	Description
	TPasswordDialog (see page 9828)	TPasswordDialog is a dialog box that allows the user to specify passwords for the current BDE session.

Functions

	Name	Description
	PasswordDialog (see page 9849)	Displays a dialog box that prompts the user for the password of a local password-protected table.

52.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TPasswordDialog (see page 9828)	TPasswordDialog is a dialog box that allows the user to specify passwords for the current BDE session.

52.1.1 TPasswordDialog Class

TPasswordDialog is a dialog box that allows the user to specify passwords for the current BDE session.

Class Hierarchy

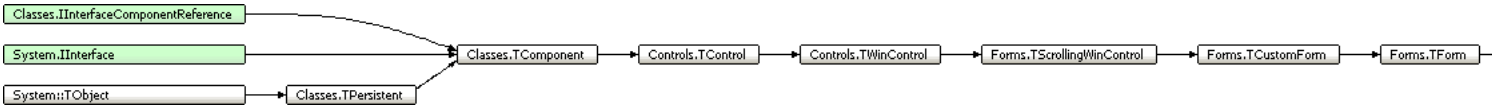

Pascal

```
TPasswordDialog = class(TForm);
```

C++

```
class TPasswordDialog : public TForm;
```

File

DBPWDlg

Description

Use TPasswordDialog to allow the user to specify passwords for the current session in a BDE-based application. (The current session is represented by a TSession object.) The passwords are used when the application opens a Paradox table that requires a password. If a table requires a password that has not been added to the current session, the user will be prompted for the password.

A TPasswordDialog object is usually created with the PasswordDialog (see page 9849) function. However, TPasswordDialog can also be used as a base object for other types of password dialog boxes.

See Also

PasswordDialog (see page 9849)

TLoginDialog (see page 9776)

TSession (see page 10093)

52.1.1.1 TPasswordDialog Members

The following tables list the members exposed by TPasswordDialog.

Protected Events

	Name	Description
	OnActivate (see page 13943)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnCanResize (see page 13947)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnClick (see page 13947)	Occurs when the user clicks the control.
	OnClose (see page 13949)	Occurs when the form closes.
	OnCloseQuery (see page 13950)	Occurs when close is attempted.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 13951)	Occurs when the form is created.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDblClick (see page 13953)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 13953)	Occurs when the form loses focus.
	OnDestroy (see page 13953)	Occurs when the form is destroyed.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragDrop (see page 13954)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDragOver (see page 13957)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnHelp (see page 13959)	Occurs when the form receives a request for help.
	OnHide (see page 13960)	Occurs when the form is hidden (that is, when its Visible (see page 13940) property is set to false).
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyDown (see page 13962)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyPress (see page 13964)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnKeyUp (see page 13966)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseActivate (see page 13967)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseDown (see page 13968)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseMove (see page 13971)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseUp (see page 13973)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnPaint (see page 13975)	Occurs when the form is redrawn.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnResize (see page 13976)	Occurs immediately after the control is resized.
	OnShortCut (see page 13977)	Occurs when the user presses a key (before the OnKeyDown (see page 13962) event).
	OnShow (see page 13977)	Occurs when the form is shown (that is, when its Visible (see page 13940) property is set to true).
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFormState (see page 13863)	This is FFormState, a member of class TCustomForm.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	Activate (see page 13865)	Activation event dispatcher.
	ActiveChanged (see page 13865)	Active (see page 13896) state event dispatcher.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 14324)	Adjusts the ClientRect property to exclude the scroll bars.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 13866)	Aligns all child controls of the form that fall within a specified rectangle.
V	AlignControls (see page 14325)	Aligns the controls in the scrolling windowed control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.

	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	AutoScrollEnabled (see page 14325)	Indicates whether automatic scrolling is enabled.
	AutoScrollInView (see page 14325)	Scrolls a control into the visible area of the scrolling windowed control.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	BeginAutoDrag (see page 13867)	Responds when the user starts dragging the form if DragMode is dmAutomatic.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	ChangeScale (see page 13867)	Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.
	ChangeScale (see page 14326)	Resizes the scrolling windowed control by a specific ratio.
	Click (see page 5743)	Respond to user click.
	CloseModal (see page 13868)	This is CloseModal, a member of class TCustomForm.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 13868)	Initializes the window-creation parameter record when the form window is created.
	CreateParams (see page 14326)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWindowHandle (see page 13869)	Creates a form window given a set of window-creation parameters.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 13869)	Creates the form window.
	CreateWnd (see page 14327)	Creates a Windows control corresponding to the TScrollingWinControl (see page 14312) object.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	Deactivate (see page 13870)	Generates an OnDeactivate (see page 13953) event.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DefineProperties (see page 13871)	Designates methods for storing a form's unpublished data on a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.

	DestroyHandle (see page 13871)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWindowHandle (see page 13872)	Destroys the window created in the CreateWindowHandle (see page 13869) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoClose (see page 13873)	Form closing event dispatcher.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoCreate (see page 13873)	Form creation event dispatcher.
	DoDestroy (see page 13873)	Form destruction event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDock (see page 13874)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoFlipChildren (see page 14327)	Flips the control's children.
	DoHide (see page 13874)	Form hide event dispatcher.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoShow (see page 13875)	Form show event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorderIconStyles (see page 13875)	This is GetBorderIconStyles, a member of class TCustomForm.
	GetBorderStyles (see page 13876)	This is GetBorderStyles, a member of class TCustomForm.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetChildren (see page 13876)	Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.

	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetClientRect (see page 13876)	Returns the value of the ClientRect property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloating (see page 13877)	Returns the value of the Floating property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetOwnerWindow (see page 13878)	This is GetOwnerWindow, a member of class TCustomForm.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 13878)	Handles exceptions that are raised by the OnCreate (see page 13951) event handler.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	Loaded (see page 13880)	Performs fix-ups when the form is first loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	Notification (see page 13882)	Responds to notifications that objects are about to be inserted or removed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 13882)	Generates an OnPaint (see page 13975) event.

	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
V	PaintWindow (see page 13882)	Renders the image of the form.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
V	PaletteChanged (see page 13883)	Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	QueryInterface (see page 13884)	Returns a reference to a specified interface if that interface is supported by the form.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
V	ReadState (see page 13884)	Reads the form's published properties, stored data, and owned components from a reader object.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
V	RequestAlign (see page 13885)	Instructs the form's parent (if any) to reposition the form, enforcing its Align property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
V	Resizing (see page 13885)	Makes any required adjustments when the form changes size.
V	Resizing (see page 14328)	This is Resizing, a member of class TScrollingWinControl.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
V	SetChildOrder (see page 13886)	Changes the order in which Child appears when GetChildren (see page 13876) is called.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.

V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParent (see page 13887)	Sets the Parent property.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
V	SetParentBiDiMode (see page 13887)	Sets the ParentBiDiMode property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
V	UpdateActions (see page 13891)	Updates all actions associated with the form.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	UpdateWindowState (see page 13892)	Updates the WindowState (see page 13942) property to reflect the current state of the form.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	ValidateRename (see page 13892)	Determines if a form can rename one of its owned components.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	VisibleChanging (see page 13893)	Responds just before the form changes the value of its Visible (see page 13940) property.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WndProc (see page 13893)	Provides specific message responses for the form.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
R	ActiveMDIChild (see page 13897)	Specifies the MDI child that has focus.
	AlphaBlend (see page 13898)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 13898)	Specifies the degree of translucency on a translucent form.
	AutoScroll (see page 13899)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoScroll (see page 14329)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.

	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderIcons (see page 13899)	Specifies which icons appear on the title bar of the form.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
R	ClientHandle (see page 13904)	Provides access to the handle of the form's internal MDI client window.
W	ClientHeight (see page 13905)	Specifies the height (in pixels) of the form's client area.
W	ClientWidth (see page 13905)	Specifies the width (in pixels) of the form's client area.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Ctl3D (see page 13909)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultMonitor (see page 13909)	Specifies the monitor on which the form appears.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	FormStyle (see page 13912)	Determines the form's style.
	Height (see page 13915)	Specifies the vertical size of the control in pixels.
	HorzScrollBar (see page 13916)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 13917)	Specifies the icon that appears when the form is minimized.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	MDIChildCount (see page 13921)	Specifies the number of open MDI child forms.
R	MDIChildren (see page 13922)	Provides indexed access to all MDI child forms.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ObjectMenuItem (see page 13926)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 13926)	Specifies when OnCreate (see page 13951) and OnDestroy (see page 13953) events occur.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentFont (see page 13927)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PixelsPerInch (see page 13928)	Represents the proportion of the font on the system on which the form was designed.
R	PopupChildren (see page 13928)	This is PopupChildren, a member of class TCustomForm.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	PopupMenu (see page 13928)	Identifies the pop-up menu associated with the control.
	Position (see page 13933)	Represents the size and placement of the form.
	PrintScale (see page 13934)	Represents the proportions of a printed form.
	Scaled (see page 13935)	Specifies whether the form is sized according to the value of the PixelsPerInch (see page 13928) property.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.

	TileMode (see page 13936)	Represents how MDI child forms are arranged when the Tile method is called.
	TransparentColor (see page 13938)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 13939)	Indicates the color on the form that appears transparent when TransparentColor (see page 13938) is true.
	VertScrollBar (see page 13939)	Represents the vertical scroll bar for the scrolling windowed control.
W	Visible (see page 13940)	Indicates whether the form is visible.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	Width (see page 13941)	Specifies the horizontal size of the control or form in pixels.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowMenu (see page 13941)	Specifies the Window menu for an MDI parent form.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Fields

	Name	Description
	AddButton (see page 9845)	This is AddButton, a member of class TPasswordDialog.
	CancelButton (see page 9845)	This is CancelButton, a member of class TPasswordDialog.
	Edit (see page 9845)	This is Edit, a member of class TPasswordDialog.
	GroupBox1 (see page 9846)	This is GroupBox1, a member of class TPasswordDialog.
	OKButton (see page 9846)	This is OKButton, a member of class TPasswordDialog.
	RemoveAllButton (see page 9846)	This is RemoveAllButton, a member of class TPasswordDialog.
	RemoveButton (see page 9846)	This is RemoveButton, a member of class TPasswordDialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddButtonClick (see page 9847)	Controls (see page 5711) what happens when the Add button is clicked.
V	AfterConstruction (see page 13866)	Construction event dispatcher.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	ArrangeIcons (see page 14006)	Arranges the icons of minimized MDI child forms.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 13866)	Destruction event dispatcher.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	Cascade (see page 14007)	Arranges MDI child forms so they overlap.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Close (see page 13867)	Closes the form.
V	CloseQuery (see page 13868)	Close (see page 13867) attempt event dispatcher.

	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 13861)	Creates and initializes a new TForm (see page 13992) object.
V	Create (see page 14323)	Creates and initializes a new TScrollingWinControl (see page 14312) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	CreateNew (see page 13862)	Creates and initializes a new form.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 13870)	Provides message handling for all messages that the form does not fully process.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DefocusControl (see page 13871)	Removes focus from a control on the form.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 13862)	Removes the form from memory.
V	Destroy (see page 14323)	Removes the TScrollingWinControl (see page 14312) object from memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	DisableAutoRange (see page 14327)	Disables automatic scrolling.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
V	Dock (see page 13872)	Handles automatic docking of the form into other windows.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EditChange (see page 9847)	Controls (see page 5711) what happens when the contents of the edit box change.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EnableAutoRange (see page 14328)	Reenables automatic scrolling.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	FocusControl (see page 13875)	Sets focus to a control on the form.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFormImage (see page 13877)	Returns a bitmap of the form.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	Hide (see page 13879)	Hides the form.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	IsShortCut (see page 13880)	Processes shortcut keys when the form has focus.
	MakeFullyVisible (see page 13881)	Ensures that the form is fully visible on a specified monitor.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MouseWheelHandler (see page 13881)	Directs mouse wheel messages to the appropriate control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 14008)	Activates the next child form in the form sequence.
	OKButtonClick (see page 9848)	Controls (see page 5711) what happens when the OK button is clicked.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	Previous (see page 14009)	Activates the previous MDI child form in the form sequence.
	Print (see page 13883)	Prints the form.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.

	RecreateAsPopup (see page 13884)	This is RecreateAsPopup, a member of class TCustomForm.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	Release (see page 13885)	Destroys the form and frees its associated memory.
	RemoveAllButtonClick (see page 9848)	Removes all passwords from the current session.
	RemoveButtonClick (see page 9848)	Removes the password in the edit box from the current session.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollInView (see page 14328)	Scrolls a control into the visible area of the scrolling windowed control.
	SendCancelMode (see page 13886)	Cancels modes on the form.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
V	SetFocus (see page 13886)	Sets focus to the form.
V	SetFocusedControl (see page 13887)	Sets focus to a control on the form.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Show (see page 13888)	Shows the form.
V	ShowModal (see page 13890)	Shows a form as a modal dialog.
	Tile (see page 14009)	Arranges MDI child forms so that they are all the same size.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UpdateGlassFrame (see page 13892)	This is UpdateGlassFrame, a member of class TCustomForm.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.
V	WantChildKey (see page 13893)	Indicates whether the form processes keyboard input for a control it owns.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Action (see page 13895)	Designates the action associated with the control.
R	Active (see page 13896)	Specifies whether the form has focus.
	ActiveControl (see page 13896)	Specifies the control that has focus on the form.
	ActiveMDIChild (see page 14013)	Specifies the MDI child that has focus.

	ActiveOleControl (see page 13897)	Specifies the OLE control on the form that reacts to changes in focus.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 13900)	Specifies the appearance and behavior of the form border.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 13900)	Provides access to the drawing area of the form.
	Caption (see page 13902)	Specifies a text string that identifies the control to the user.
	ClientHandle (see page 14024)	Provides access to the handle of the form's internal MDI client window.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 13908)	Specifies the background color of the control.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designer (see page 13910)	Specifies the designer interface for the form.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockManager (see page 14029)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropTarget (see page 13910)	Specifies whether the form is the target of a drag-and-drop operation.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Font (see page 13910)	Controls the attributes of text written on or in the control.
R	FormState (see page 13911)	Indicates transitional state information about the form.
	GlassFrame (see page 13914)	This is GlassFrame, a member of class TCustomForm.

R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HelpFile (see page 13916)	Specifies the name of the file the form uses to display Help.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	KeyPreview (see page 13918)	Specifies whether the form should receive keyboard events before the active control.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	MDIChildCount (see page 14044)	Specifies the number of open MDI child forms.
	MDIChildren (see page 14044)	Provides indexed access to all MDI child forms.
	Menu (see page 13923)	Specifies the form's main menu.
	ModalResult (see page 13924)	Represents the return value of a form that is used as a modal dialog.
R	Monitor (see page 13926)	Provides access to the monitor on which the form appears.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	OleFormObject (see page 13927)	Specifies the IOleForm (see page 14340) interface for an in-place OLE object contained in the form.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PopupMode (see page 13933)	This is PopupMode, a member of class TCustomForm.
	PopupParent (see page 13933)	Sets an order for stacked forms that users cannot change.
	ScreenSnap (see page 13935)	Specifies whether form snaps to edge of screen.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	SnapBuffer (see page 13936)	Specifies distance for screen snap.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TileMode (see page 14059)	Represents how MDI child forms are arranged when the Tile method is called.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.
	WindowState (see page 13942)	Represents how the form appears on the screen.

Published Events

	Name	Description
	OnActivate (see page 14065)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 14069)	Occurs when an object with custom alignment is aligned.
	OnAlignInsertBefore (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14069)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 14070)	Occurs when an attempt is made to resize the control.
	OnClick (see page 14070)	Occurs when the user clicks the control.
	OnClose (see page 14072)	Occurs when the form closes.
	OnCloseQuery (see page 14073)	Occurs when close is attempted.
	OnConstrainedResize (see page 14074)	Adjust resize constraints.
	OnContextPopup (see page 14075)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 14076)	Occurs when the form is created.
	OnDblClick (see page 14077)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 14078)	Occurs when the form loses focus.

	OnDestroy (see page 14078)	Occurs when the form is destroyed.
	OnDockDrop (see page 14079)	Occurs when another control is docked to the control.
	OnDockOver (see page 14081)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 14083)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 14085)	Occurs when the user drags an object over a control.
	OnEndDock (see page 14088)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnGetSiteInfo (see page 14088)	Returns the control's docking information.
	OnHelp (see page 14090)	Occurs when the form receives a request for help.
	OnHide (see page 14091)	Occurs when the form is hidden (that is, when its Visible property is set to false).
	OnKeyDown (see page 14093)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 14095)	Occurs when a key is pressed.
	OnKeyUp (see page 14096)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 14098)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 14099)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 14101)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 14102)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 14102)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 14104)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 14107)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 14107)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 14107)	Occurs when the mouse wheel is rotated upward.
	OnPaint (see page 14108)	Occurs when the form is redrawn.
	OnResize (see page 14109)	Occurs immediately after the control is resized.
	OnShortCut (see page 14109)	Occurs when the user presses a key (before the OnKeyDown event).
	OnShow (see page 14110)	Occurs when the form is shown (that is, when its Visible property is set to true).
	OnStartDock (see page 14111)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnUnDock (see page 14111)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Action (see page 14011)	Designates the action associated with the control.
	ActiveControl (see page 14012)	Specifies the control that has focus on the form.
	Align (see page 14013)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AlphaBlend (see page 14017)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 14017)	Specifies the degree of translucency on a translucent form.
	Anchors (see page 14017)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 14018)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 14018)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BiDiMode (see page 14019)	Specifies the bi-directional mode for the control.
	BorderIcons (see page 14019)	Specifies which icons appear on the title bar of the form.
	BorderStyle (see page 14020)	Specifies the appearance and behavior of the form border.
	BorderWidth (see page 14021)	Specifies the width of the control's border.
	Caption (see page 14021)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 14024)	Specifies the height (in pixels) of the form's client area.
	ClientWidth (see page 14025)	Specifies the width (in pixels) of the form's client area.
	Color (see page 14027)	Specifies the background color of the control.
	Constraints (see page 14028)	Specifies the size constraints for the control.

	Ctl3D (see page 14028)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DefaultMonitor (see page 14029)	Specifies the monitor on which the form appears.
	DockSite (see page 14031)	Specifies whether the control can be the target of drag-and-dock operations.
	DragKind (see page 14032)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 14033)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 14034)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 14035)	Controls the attributes of text written on or in the control.
	FormStyle (see page 14036)	Determines the form's style.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	Height (see page 14038)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpFile (see page 14039)	Specifies the name of the file the form uses to display Help.
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HorzScrollBar (see page 14040)	Represents the horizontal scroll bar for the scrolling windowed control.
	HorzScrollBar (see page 14329)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 14041)	Specifies the icon that appears when the form is minimized.
	KeyPreview (see page 14041)	Specifies whether the form should receive keyboard events before the active control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Left (see page 13920)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Menu (see page 14045)	Specifies the form's main menu.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectMenuItem (see page 14047)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 14047)	Specifies when OnCreate and OnDestroy events occur.
	Padding (see page 14047)	Specifies the padding of a control.
	ParentBiDiMode (see page 14048)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 14048)	Determines where a control looks for its font information.
	PixelsPerInch (see page 14049)	Represents the proportion of the font on the system on which the form was designed.
	PopupMenu (see page 14049)	Identifies the pop-up menu associated with the control.
	PopupMode (see page 14054)	Controls (see page 5711) how the top-level form behaves with respect to Window's WS_POPUP style.
	PopupParent (see page 14054)	Sets an order for stacked forms that users cannot change.
	Position (see page 14055)	Represents the size and placement of the form.
	PrintScale (see page 14056)	Represents the proportions of a printed form.
	Scaled (see page 14056)	Specifies whether the form is sized according to the value of the PixelsPerInch property.
	ScreenSnap (see page 14056)	Specifies whether form snaps to edge of screen.
	ShowHint (see page 14057)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SnapBuffer (see page 14058)	Specifies distance for screen snap.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Top (see page 13937)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	TransparentColor (see page 14060)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 14060)	Indicates the color on the form that appears transparent when TransparentColor is true.

	UseDockManager (see page 14060)	Specifies whether the docking manager is used in drag-and-dock operations.
	VertScrollBar (see page 14061)	Represents the vertical scroll bar for the scrolling windowed control.
	VertScrollBar (see page 14330)	Represents the vertical scroll bar for the scrolling windowed control.
	Visible (see page 14062)	Indicates whether the form is visible.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	Width (see page 14063)	Specifies the horizontal size of the control or form in pixels.
	WindowMenu (see page 14063)	Specifies the Window menu for an MDI parent form.
	WindowState (see page 14063)	Represents how the form appears on the screen.

52.1.1.2 TPasswordDialog Fields

The fields of the TPasswordDialog class are listed here.

Public Fields

	Name	Description
	AddButton (see page 9845)	This is AddButton, a member of class TPasswordDialog.
	CancelButton (see page 9845)	This is CancelButton, a member of class TPasswordDialog.
	Edit (see page 9845)	This is Edit, a member of class TPasswordDialog.
	GroupBox1 (see page 9846)	This is GroupBox1, a member of class TPasswordDialog.
	OKButton (see page 9846)	This is OKButton, a member of class TPasswordDialog.
	RemoveAllButton (see page 9846)	This is RemoveAllButton, a member of class TPasswordDialog.
	RemoveButton (see page 9846)	This is RemoveButton, a member of class TPasswordDialog.

52.1.1.2.1 TPasswordDialog.AddButton Field

Pascal

```
AddButton: TButton;
```

C++

```
TButton AddButton;
```

Description

This is AddButton, a member of class TPasswordDialog.

52.1.1.2.2 TPasswordDialog.CancelButton Field

Pascal

```
CancelButton: TButton;
```

C++

```
TButton CancelButton;
```

Description

This is CancelButton, a member of class TPasswordDialog.

52.1.1.2.3 TPasswordDialog.Edit Field

Pascal

```
Edit: TEdit;
```

C++

```
TEdit Edit;
```

Description

This is Edit, a member of class TPasswordDialog.

52.1.1.2.4 TPasswordDialog.GroupBox1 Field

Pascal

```
GroupBox1: TGroupBox;
```

C++

```
TGroupBox GroupBox1;
```

Description

This is GroupBox1, a member of class TPasswordDialog.

52.1.1.2.5 TPasswordDialog.OKButton Field

Pascal

```
OKButton: TButton;
```

C++

```
TButton OKButton;
```

Description

This is OKButton, a member of class TPasswordDialog.

52.1.1.2.6 TPasswordDialog.RemoveAllButton Field

Pascal

```
RemoveAllButton: TButton;
```

C++

```
TButton RemoveAllButton;
```

Description

This is RemoveAllButton, a member of class TPasswordDialog.

52.1.1.2.7 TPasswordDialog.RemoveButton Field

Pascal

```
RemoveButton: TButton;
```

C++

```
TButton RemoveButton;
```

Description

This is RemoveButton, a member of class TPasswordDialog.

52.1.1.3 TPasswordDialog Methods

The methods of the TPasswordDialog class are listed here.

Public Methods

	Name	Description
	AddButtonClick (see page 9847)	Controls (see page 5711) what happens when the Add button is clicked.
	EditChange (see page 9847)	Controls (see page 5711) what happens when the contents of the edit box change.
	OKButtonClick (see page 9848)	Controls (see page 5711) what happens when the OK button is clicked.
	RemoveAllButtonClick (see page 9848)	Removes all passwords from the current session.
	RemoveButtonClick (see page 9848)	Removes the password in the edit box from the current session.

52.1.1.3.1 TPasswordDialog.AddButtonClick Method

Controls (see page 5711) what happens when the Add button is clicked.

Pascal

```
procedure AddButtonClick(Sender: TObject);
```

C++

```
__fastcall AddButtonClick(TObject * Sender);
```

Description

AddButtonClick adds the password in the edit box to the current session. Then AddButtonClick clears and sets focus to the edit box.

After the AddButtonClick method is called, the OK Button is enabled, regardless of whether the edit box currently contains a password.

See Also

OKButtonClick (see page 9848)

AddPassword (see page 10102)

52.1.1.3.2 TPasswordDialog.EditChange Method

Controls (see page 5711) what happens when the contents of the edit box change.

Pascal

```
procedure EditChange(Sender: TObject);
```

C++

```
__fastcall EditChange(TObject * Sender);
```

Description

EditChange enables or disables the Add, OK, and Remove buttons according to whether the edit box contains a password. If the edit box contains a password, the buttons are enabled. Otherwise the buttons are disabled.

Note: If the AddButtonClick (see page 9847) method has been called, then the OK button is enabled, regardless of whether the edit box contains a password.

See Also

AddButtonClick (see page 9847)

52.1.1.3.3 TPasswordDialog.OKButtonClick Method

Controls (see page 5711) what happens when the OK button is clicked.

Pascal

```
procedure OKButtonClick(Sender: TObject);
```

C++

```
__fastcall OKButtonClick(TObject * Sender);
```

Description

OKButtonClick adds the password in the edit box to the current session.

See Also

AddPassword (see page 10102)

52.1.1.3.4 TPasswordDialog.RemoveAllButtonClick Method

Removes all passwords from the current session.

Pascal

```
procedure RemoveAllButtonClick(Sender: TObject);
```

C++

```
__fastcall RemoveAllButtonClick(TObject * Sender);
```

Description

RemoveAllButtonClick removes all passwords from the current session. Then RemoveAllButtonClick sets focus to the edit box.

See Also

RemoveButtonClick (see page 9848)

RemoveAllPasswords (see page 10118)

52.1.1.3.5 TPasswordDialog.RemoveButtonClick Method

Removes the password in the edit box from the current session.

Pascal

```
procedure RemoveButtonClick(Sender: TObject);
```

C++

```
__fastcall RemoveButtonClick(TObject * Sender);
```

Description

RemoveButtonClick removes the password in the edit box from the current session. Then RemoveButtonClick clears and sets focus to the edit box.

See Also

RemoveAllButtonClick (see page 9848)

RemovePassword (see page 10118)

52.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	PasswordDialog (see page 9849)	Displays a dialog box that prompts the user for the password of a local password-protected table.

52.2.1 DBPWDlg.PasswordDialog Function

Displays a dialog box that prompts the user for the password of a local password-protected table.

Pascal

```
function PasswordDialog(const ASession: IDBSession): Boolean;
```

C++

```
Boolean PasswordDialog(const IDBSession ASession);
```

File

DBPWDlg

Description

Use PasswordDialog to allow the user to enter a password for a password-protected Paradox or dBase table.

ASession is the interface of a component that represents the session onto which the user logs in. Typically, this is the interface of a TSession component.

If the user clicks OK in the dialog box, PasswordDialog returns true. Otherwise it returns false.

Note: When an application opens a Paradox or dBASE table that requires a password, the user is prompted to enter a password unless the Session already has a valid password for that table.

See Also

TSession

Working with Password-protected Paradox and dBASE Tables

53 DBTables Namespace

This is namespace DBTables.

Topics

Name	Description
Classes (see page 9854)	The following table lists classes in this documentation.
Functions (see page 10244)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 10251)	The following table lists structs, records, enums in this documentation.
Types (see page 10258)	The following table lists types in this documentation.
Variables (see page 10265)	The following table lists variables in this documentation.
Constants (see page 10267)	The following table lists constants in this documentation.

Classes

	Name	Description
	EDBEngineError (see page 9854)	EDBEngineError is the exception class for Borland (see page 1761) Database Engine errors.
	ENoResultSet (see page 9858)	ENoResultSet is the exception class for unsuccessful attempts to open a dataset query.
	TBatchMove (see page 9858)	TBatchMove performs database operations on groups of records or entire tables.
	TBDECallback (see page 9875)	TBDECallback is a wrapper for a Borland (see page 1761) Database Engine (BDE) callback function.
	TBDEDataSet (see page 9878)	Encapsulates Borland (see page 1761) Database Engine (BDE) functionality for descendant dataset objects.
	TBlobStream (see page 9958)	TBlobStream is a stream object that provides services which allow applications to read from or write to field objects that represent Binary large object (BLOB) fields.
	TDatabase (see page 9964)	TDatabase provides discrete control over a connection to a single database in a BDE-based database application.
	TDataSetUpdateObject (see page 9994)	TDataSetUpdateObject is a base class for update objects used to update otherwise unupdatable queries or stored procedures in applications that cache updates.
	TDBDataSet (see page 10000)	TDBDataSet encapsulates database connectivity for descendant dataset objects.
	TDBError (see page 10029)	TDBError represents Borland (see page 1761) Database Engine errors for the EDBEngineError (see page 9854) exception class.
	TIndexFiles (see page 10033)	This is class DBTables.TIndexFiles.
	TNestedTable (see page 10040)	TNestedTable encapsulates a dataset that is nested as a field within another table.
	TParamList (see page 10056)	TParamList manages a list of session parameters.
	TQuery (see page 10059)	TQuery represents a dataset with a result set that is based on an SQL (see page 10089) statement.

	TSession (see page 10093)	TSession provides global management of a group of database connections in an application.
	TSessionList (see page 10129)	TSessionList manages one or more Borland (see page 1761) Database Engine (BDE) sessions in a database application.
	TSQLUpdateObject (see page 10135)	TSQLUpdateObject is the base class for classes that use an SQL command to apply cached updates on behalf of queries or stored procedures that can't post updates directly.
	TStoredProc (see page 10139)	TStoredProc encapsulates a stored procedure in a BDE-based application.
	TTable (see page 10168)	TTable encapsulates a database table.
	TUpdateSQL (see page 10229)	TUpdateSQL applies cached updates on behalf of queries or stored procedures that can't post updates directly.

Constants

Name	Description
BlobTypeMap (see page 10267)	This is constant DBTables.BlobTypeMap.
cmAll (see page 10268)	This is constant DBTables.cmAll.
cmPersistent (see page 10268)	This is constant DBTables.cmPersistent.
cmSession (see page 10268)	This is constant DBTables.cmSession.
cmVirtual (see page 10269)	This is constant DBTables.cmVirtual.
DataTypeMap (see page 10269)	This is constant DBTables.DataTypeMap.
dbfDatabase (see page 10269)	This is constant DBTables.dbfDatabase.
dbfExecProc (see page 10270)	This is constant DBTables.dbfExecProc.
dbfExecSQL (see page 10270)	This is constant DBTables.dbfExecSQL.
dbfFieldList (see page 10270)	This is constant DBTables.dbfFieldList.
dbfIndexList (see page 10271)	This is constant DBTables.dbfIndexList.
dbfOpened (see page 10271)	This is constant DBTables.dbfOpened.
dbfPrepared (see page 10271)	This is constant DBTables.dbfPrepared.
dbfProcDesc (see page 10272)	This is constant DBTables.dbfProcDesc.
dbfProvider (see page 10272)	This is constant DBTables.dbfProvider.
dbfStoredProc (see page 10272)	This is constant DBTables.dbfStoredProc.
dbfTable (see page 10273)	This is constant DBTables.dbfTable.
FldSubTypeMap (see page 10273)	This is constant DBTables.FldSubTypeMap.
FldTypeMap (see page 10273)	This is constant DBTables.FldTypeMap.
smTraceBufSize (see page 10274)	This is constant DBTables.smTraceBufSize.

Functions

	Name	Description
	AnsiToNative (see page 10244)	Converts a string from the ANSI character set to the character set associated with a given locale.
	AnsiToNativeBuf (see page 10245)	Converts a string from the ANSI character set to the character set associated with a given locale.
	Check (see page 10245)	Determines whether a value returned from the Borland (see page 1761) Database Engine (BDE) represents an error condition.
	DbiError (see page 10246)	Creates and raises an EDBEngineError (see page 9854) exception for an error code returned by the Borland (see page 1761) Database Engine (BDE).
	GetFieldSource (see page 10246)	This is function DBTables.GetFieldSource.
	NativeCompareStr (see page 10246)	Compares strings based on a database locale case sensitively.
	NativeCompareStrBuf (see page 10247)	Compares null-terminated strings based on a database locale case sensitively.
	NativeCompareText (see page 10248)	Compares strings based on a database locale without case sensitivity.
	NativeCompareTextBuf (see page 10248)	Compares null-terminated strings based on a database locale and is not case sensitive.
	NativeToAnsi (see page 10249)	Converts a string from the character set of the given locale to the ANSI character set.
	NativeToAnsiBuf (see page 10249)	Converts a string from the character set of a specified locale into the ANSI character set.
	RegisterBDEInitProc (see page 10250)	This is function DBTables.RegisterBDEInitProc.

Structs, Records, Enums

	Name	Description
	TBatchMode (see page 10251)	TBatchMode indicates the type of batch operation to be performed on a database table.
	TBDEKeyBuffer (see page 10252)	This is record DBTables.TBDEKeyBuffer.
	TBDERecInfo (see page 10252)	This is record DBTables.TBDERecInfo.
	TConfigModes (see page 10253)	This is record DBTables.TConfigModes.
	TDatabaseEvent (see page 10253)	This is record DBTables.TDatabaseEvent.
	TLockType (see page 10254)	Indicates how a Paradox or dBASE table can be locked.
	TParamBindMode (see page 10254)	This is record DBTables.TParamBindMode.
	TRecNoStatus (see page 10255)	This is record DBTables.TRecNoStatus.
	TServerDesc (see page 10255)	This is record DBTables.TServerDesc.
	TTableType (see page 10256)	This is record DBTables.TTableType.
	TTraceFlag (see page 10256)	This is record DBTables.TTraceFlag.
	TTransIsolation (see page 10257)	This is record DBTables.TTransIsolation.

Types

Name	Description
PKeyBuffer (see page 10258)	This is type DBTables.PKeyBuffer.
PRecInfo (see page 10258)	This is type DBTables.PRecInfo.
TBDECallbackEvent (see page 10259)	This is type DBTables.TBDECallbackEvent.
TBDEInitProc (see page 10259)	This is type DBTables.TBDEInitProc.
TBlobDataArray (see page 10259)	This is type DBTables.TBlobDataArray.
TConfigMode (see page 10260)	This is type DBTables.TConfigMode.
TDatabaseLoginEvent (see page 10260)	This is type DBTables.TDatabaseLoginEvent.
TDatabaseNotifyEvent (see page 10260)	This is type DBTables.TDatabaseNotifyEvent.
TDBFlags (see page 10261)	This is type DBTables.TDBFlags.
TFieldDescList (see page 10261)	This is type DBTables.TFieldDescList.
TIndexDescList (see page 10261)	This is type DBTables.TIndexDescList.
TIndexName (see page 10262)	This is type DBTables.TIndexName.
TLocale (see page 10262)	TLocale is the type for the Locale or DBLocale property.
TPasswordEvent (see page 10262)	This is type DBTables.TPasswordEvent.
TServerDescList (see page 10263)	This is type DBTables.TServerDescList.
TSPParamDescList (see page 10263)	This is type DBTables.TSPParamDescList.
TTraceFlags (see page 10263)	TTraceFlags and TTraceFlags indicate the types of messages that pass between an application and the InterBase server.
TValCheckList (see page 10264)	This is type DBTables.TValCheckList.

Variables

Name	Description
GetObjectContextProc (see page 10265)	This is variable DBTables.GetObjectContextProc.
Session (see page 10265)	Maintains database components used by an application.
Sessions (see page 10266)	Provides access to TSessionList (see page 10129) methods and properties.

53.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EDBEngineError (see page 9854)	EDBEngineError is the exception class for Borland (see page 1761) Database Engine errors.
	ENoResultSet (see page 9858)	ENoResultSet is the exception class for unsuccessful attempts to open a dataset query.
	TBatchMove (see page 9858)	TBatchMove performs database operations on groups of records or entire tables.
	TBDECallback (see page 9875)	TBDECallback is a wrapper for a Borland (see page 1761) Database Engine (BDE) callback function.
	TBDEDataSet (see page 9878)	Encapsulates Borland (see page 1761) Database Engine (BDE) functionality for descendant dataset objects.
	TBlobStream (see page 9958)	TBlobStream is a stream object that provides services which allow applications to read from or write to field objects that represent Binary large object (BLOB) fields.
	TDatabase (see page 9964)	TDatabase provides discrete control over a connection to a single database in a BDE-based database application.
	TDataSetUpdateObject (see page 9994)	TDataSetUpdateObject is a base class for update objects used to update otherwise unupdatable queries or stored procedures in applications that cache updates.
	TDBDataSet (see page 10000)	TDBDataSet encapsulates database connectivity for descendant dataset objects.
	TDBError (see page 10029)	TDBError represents Borland (see page 1761) Database Engine errors for the EDBEngineError (see page 9854) exception class.
	TIndexFiles (see page 10033)	This is class DBTables.TIndexFiles.
	TNestedTable (see page 10040)	TNestedTable encapsulates a dataset that is nested as a field within another table.
	TParamList (see page 10056)	TParamList manages a list of session parameters.
	TQuery (see page 10059)	TQuery represents a dataset with a result set that is based on an SQL (see page 10089) statement.
	TSession (see page 10093)	TSession provides global management of a group of database connections in an application.
	TSessionList (see page 10129)	TSessionList manages one or more Borland (see page 1761) Database Engine (BDE) sessions in a database application.
	TSQLUpdateObject (see page 10135)	TSQLUpdateObject is the base class for classes that use an SQL command to apply cached updates on behalf of queries or stored procedures that can't post updates directly.
	TStoredProc (see page 10139)	TStoredProc encapsulates a stored procedure in a BDE-based application.
	TTable (see page 10168)	TTable encapsulates a database table.
	TUpdateSQL (see page 10229)	TUpdateSQL applies cached updates on behalf of queries or stored procedures that can't post updates directly.

53.1.1 EDBEngineError Class

EDBEngineError is the exception class for Borland (see page 1761) Database Engine errors.

Class Hierarchy

System::TObject → SysUtils.Exception → DB.EDatabaseError → DBTables.EDBEngineError

Pascal

```
EDBEngineError = class(EDatabaseError);
```

C++

```
class EDBEngineError : public EDatabaseError;
```

File

DBTables

Description

EDBEngineError is raised whenever a BDE error occurs. Its Errors (see page 9857) property contains TDBError (see page 10029) objects.

When EDBEngineError exceptions are raised at design time, they appear in a special dialog (in Delphi, TDBEngineErrorDlg) that displays the complete stack of BDE errors. At runtime, unhandled EDBEngineError exceptions generate the same message box as other exceptions.

See Also

TDBError (see page 10029)

TDbEngineErrorDlg (see page 9546)

DBIError

53.1.1.1 EDBEngineError Members

The following tables list the members exposed by EDBEngineError.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 9856)	Creates an instance of EDBEngineError (see page 9854) using a specified BDE error code.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 9856)	Disposes of the EDBEngineError (see page 9854) instance and each TDBError (see page 10029) instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ErrorCount (see page 9857)	Indicates the total number of errors contained in the Errors (see page 9857) property.
R	Errors (see page 9857)	Lists the entire Borland (see page 1761) Database Engine error stack.
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.

53.1.1.2 EDBEngineError.Create Constructor

Creates an instance of EDBEngineError (see page 9854) using a specified BDE error code.

Pascal

```
constructor Create(ErrorCode: DBIResult);
```

C++

```
__fastcall EDBEngineError(DBIResult ErrorCode);
```

Description

Create calls the Create method inherited from Exception (using an empty string) to construct an initialized instance of EDBEngineError (see page 9854). It then creates a TDBError (see page 10029) instance for each error in the BDE error stack and builds a single string containing all the error messages. If the BDE is uninitialized when Create is called, the method generates only one generic error message and one TDBError (see page 10029) instance.

See Also

Exception (see page 22750)

Create (see page 22752)

TDBError (see page 10029)

53.1.1.3 EDBEngineError.Destroy Destructor

Disposes of the EDBEngineError (see page 9854) instance and each TDBError (see page 10029) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~EDBEngineError();
```

Description

Destroy first disposes of each TDBError (see page 10029) instance referenced in the Errors (see page 9857) property, then calls the inherited destructor to dispose of the EDBEngineError (see page 9854) instance itself.

See Also

TDBError (see page 10029)

Errors (see page 9857)

Free (see page 22300)

53.1.1.4 EDBEngineError Properties

The properties of the EDBEngineError class are listed here.

Public Properties

	Name	Description
R	ErrorCount (see page 9857)	Indicates the total number of errors contained in the Errors (see page 9857) property.
R	Errors (see page 9857)	Lists the entire Borland (see page 1761) Database Engine error stack.

53.1.1.4.1 EDBEngineError.ErrorCount Property

Indicates the total number of errors contained in the Errors (see page 9857) property.

Pascal

```
property ErrorCount: Integer;
```

C++

```
__property int ErrorCount;
```

Description

Use ErrorCount as an upper bound when iterating through the errors in the error stack.

See Also

Errors (see page 9857)

53.1.1.4.2 EDBEngineError.Errors Property

Lists the entire Borland (see page 1761) Database Engine error stack.

Pascal

```
property Errors [Index: Integer]: TDBError;
```

C++

```
__property TDBError Errors[int Index];
```

Description

Errors is an index of TDBError (see page 10029) objects. The first error has an index value of 0.

See Also

TDBError (see page 10029)

53.1.2 ENoResultSet Class

ENoResultSet is the exception class for unsuccessful attempts to open a dataset query.

Class Hierarchy

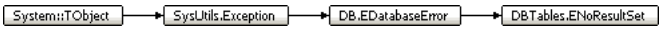

Pascal

```
ENoResultSet = class(EDatabaseError);
```

C++

```
class ENoResultSet : public EDatabaseError;
```

File

DBTables

Description

ENoResultSet is raised by TQuery (see page 10059) when an attempt is made to open the query without a SELECT statement.

See Also

TQuery (see page 10059)

TDataSet (see page 6894)

53.1.3 TBatchMove Class

TBatchMove performs database operations on groups of records or entire tables.

Class Hierarchy

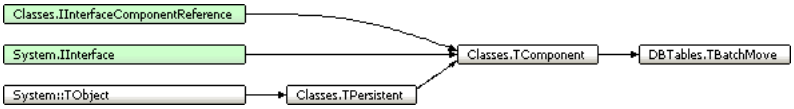

Pascal

```
TBatchMove = class(TComponent);
```

C++

```
class TBatchMove : public TComponent;
```

File

DBTables

Description

Use a TBatchMove object to

Add the records in a dataset to a database table.

Delete the records in a dataset from a database table.

Copy a dataset to create a new database table or overwrite an existing table.

Set the Mode (see page 9871) property to specify the desired operation. The Source (see page 9874) and Destination (see page 9868) properties indicate the datasets whose records are added, deleted, or copied. Other properties further specify how to perform the operation and how to handle problems that TBatchMove encounters when performing its operation. Once you have set properties to indicate what you want to do, call the Execute (see page 9863) method to perform the actual operation.

Note: TTable (see page 10168) objects can also perform the same batch operations, but only using the options specified by the default TBatchMove property values.

Tip: Applications can use a TBDECallback (see page 9875) object to further refine the response to problems that arise.

See Also

TBDECallback (see page 9875)

TTable (see page 10168)

Creating a Batch Move Component

Using TBatchMove

Handling Batch Move Errors

53.1.3.1 TBatchMove Members

The following tables list the members exposed by TBatchMove.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.

	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 9864)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 9862)	Creates an instance of TBatchMove (see page 9858).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 9863)	Destroys an instance of TBatchMove (see page 9858).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 9863)	Performs the batch operation specified by Mode (see page 9871).
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ChangedCount (see page 9866)	Indicates the number of records from the Destination (see page 9868) table that are altered as a result of the batch operation.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	KeyViolCount (see page 9869)	Reports the number of records which could not be replaced, added, or deleted from Destination (see page 9868) because of integrity or key violations.
R	MovedCount (see page 9872)	Reports the number of records form the Source (see page 9874) which were applied to the destination.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	ProblemCount (see page 9872)	Indicates the number of records which could not be added to Destination (see page 9868) without loss of data due to a field type mismatch.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	AbortOnKeyViol (see page 9865)	Specifies whether the batch operation is terminated immediately when integrity or key violations occur.
	AbortOnProblem (see page 9866)	Specifies whether the batch operation is terminated immediately when it is necessary to truncate data to make it fit in the specified Destination (see page 9868).
	ChangedTableName (see page 9867)	Specifies the name of a local Paradox table that is created to hold copies of all records from the destination table changed by the batch operation.
	CommitCount (see page 9867)	Specifies how many records are batch moved before a commit occurs.
	Destination (see page 9868)	Specifies the TTable (see page 10168) object for the database table that is the destination of the batch operation.
	KeyViolTableName (see page 9869)	Specifies the name of a Paradox table that will be created to contain all records from the Source (see page 9874) that cannot participate in the batch operation because of integrity or key violations.
	Mappings (see page 9870)	Specifies the column mappings for a batch operation.
	Mode (see page 9871)	Specifies what the TBatchMove (see page 9858) object does when the Execute (see page 9863) method is called.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ProblemTableName (see page 9873)	Specifies the name of a Paradox table that will be created to contain all records from the Source (see page 9874) that contain fields which were trimmed to match the corresponding field type in the Destination (see page 9868) table.
	RecordCount (see page 9873)	Specifies the maximum number of records that are applied to the Destination (see page 9868) when Execute (see page 9863) is called.
	Source (see page 9874)	Specifies the dataset that is the source of the batch operation.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Transliterate (see page 9875)	Specifies whether the data in the Source (see page 9874) records should be converted form the locale of the Source (see page 9874) to the locale of the Destination (see page 9868) when the Execute (see page 9863) method is called.

53.1.3.2 TBatchMove.Create Constructor

Creates an instance of TBatchMove (see page 9858).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBatchMove(TComponent * AOwner);
```

Description

Call Create to create an instance of TBatchMove (see page 9858) for performing a batch operation. Create allocates the memory for the TBatchMove (see page 9858) object and

Sets AbortOnKeyViol (see page 9865) to true.

Sets AbortOnProblem (see page 9866) to true.

Sets Transliterate (see page 9875) to true.

Creates a TStringList object to hold the Mappings (see page 9870).

The property settings made by Create are the values used when a TTable (see page 10168) object performs batch operations using its BatchMove method.

See Also

AbortOnKeyViol (see page 9865)

AbortOnProblem (see page 9866)

Mappings (see page 9870)

Transliterate (see page 9875)

BatchMove (see page 10186)

53.1.3.3 TBatchMove.Destroy Destructor

Destroys an instance of TBatchMove (see page 9858).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBatchMove();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which checks that the TBatchMove (see page 9858) object is not nil before calling Destroy.Before calling the inherited destructor, Destroy frees the TStringList object that implements the Mappings (see page 9870) property.

See Also

Free (see page 22300)

Mappings (see page 9870)

53.1.3.4 TBatchMove Methods

The methods of the TBatchMove class are listed here.

Protected Methods

	Name	Description
	Notification (see page 9864)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
	Execute (see page 9863)	Performs the batch operation specified by Mode (see page 9871).

53.1.3.4.1 TBatchMove.Execute Method

Performs the batch operation specified by Mode (see page 9871).

Pascal

```
procedure Execute;
```

C++

```
__fastcall Execute();
```

Description

After setting properties to indicate what batch operation should be performed and how, call Execute to perform the operation. As a minimum, the Mode (see page 9871), Source (see page 9874), and Destination (see page 9868) properties must be set.

After calling Execute, the values of the ChangedCount (see page 9866), KeyViolCount (see page 9869), MovedCount (see page 9872), ProblemCount (see page 9872), and RecordCount (see page 9873) properties will become available.

These properties will give an indication of what happened as a result of the call to Execute. Additional information about the batch operation will be available through the creation of a changed table, a keyviol table, and/or a problems table if it was requested.

See Also

ChangedCount (see page 9866)

KeyViolCount (see page 9869)

MovedCount (see page 9872)

ProblemCount (see page 9872)

RecordCount (see page 9873)

Source (see page 9874)

Destination (see page 9868)

Executing a Batch Move

53.1.3.4.2 TBatchMove.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

53.1.3.5 TBatchMove Properties

The properties of the TBatchMove class are listed here.

Public Properties

	Name	Description
R	ChangedCount (see page 9866)	Indicates the number of records from the Destination (see page 9868) table that are altered as a result of the batch operation.
R	KeyViolCount (see page 9869)	Reports the number of records which could not be replaced, added, or deleted from Destination (see page 9868) because of integrity or key violations.
R	MovedCount (see page 9872)	Reports the number of records form the Source (see page 9874) which were applied to the destination.
R	ProblemCount (see page 9872)	Indicates the number of records which could not be added to Destination (see page 9868) without loss of data due to a field type mismatch.

Published Properties

	Name	Description
	AbortOnKeyViol (see page 9865)	Specifies whether the batch operation is terminated immediately when integrity or key violations occur.
	AbortOnProblem (see page 9866)	Specifies whether the batch operation is terminated immediately when it is necessary to truncate data to make it fit in the specified Destination (see page 9868).
	ChangedTableName (see page 9867)	Specifies the name of a local Paradox table that is created to hold copies of all records from the destination table changed by the batch operation.
	CommitCount (see page 9867)	Specifies how many records are batch moved before a commit occurs.
	Destination (see page 9868)	Specifies the TTable (see page 10168) object for the database table that is the destination of the batch operation.
	KeyViolTableName (see page 9869)	Specifies the name of a Paradox table that will be created to contain all records from the Source (see page 9874) that cannot participate in the batch operation because of integrity or key violations.
	Mappings (see page 9870)	Specifies the column mappings for a batch operation.
	Mode (see page 9871)	Specifies what the TBatchMove (see page 9858) object does when the Execute (see page 9863) method is called.
	ProblemTableName (see page 9873)	Specifies the name of a Paradox table that will be created to contain all records from the Source (see page 9874) that contain fields which were trimmed to match the corresponding field type in the Destination (see page 9868) table.
	RecordCount (see page 9873)	Specifies the maximum number of records that are applied to the Destination (see page 9868) when Execute (see page 9863) is called.
	Source (see page 9874)	Specifies the dataset that is the source of the batch operation.
	Transliterate (see page 9875)	Specifies whether the data in the Source (see page 9874) records should be converted form the locale of the Source (see page 9874) to the locale of the Destination (see page 9868) when the Execute (see page 9863) method is called.

53.1.3.5.1 TBatchMove.AbortOnKeyViol Property

Specifies whether the batch operation is terminated immediately when integrity or key violations occur.

Pascal

```
property AbortOnKeyViol: Boolean;
```

C++

```
__property Boolean AbortOnKeyViol;
```

Description

Set AbortOnKeyViol to true to cancel the batch operation when it causes integrity or key violations in the destination table. Set AbortOnKeyViol to false to cause records that would cause integrity or key violations to be posted to a key violations table instead. Provide a KeyViolTableName (see page 9869) when setting AbortOnKeyViol to false. If there is no KeyViolTableName (see page 9869), AbortOnKeyViol may still be set to false, in which case problem records are counted but not saved in a table.

Key violations occur when appending records to a table that has a primary key. On a table with a primary key, no two records may have the same values on all of the key fields. If a record that is being appended has the same values on all the key fields as

a record in the destination table, it cannot be appended.

Integrity violations occur when deleting, appending or updating records to a table that has a referential integrity specification. A referential integrity specification requires that certain fields in one table must have values that exist in a particular field of another table. If a record that is being added does not match the restrictions imposed by referential integrity, it cannot be added. If deleting record would cause tighter referential integrity restrictions that another table cannot meet, it cannot be deleted.

See Also

KeyViolCount (see page 9869)

KeyViolTableName (see page 9869)

Handling Batch Move Errors

53.1.3.5.2 TBatchMove.AbortOnProblem Property

Specifies whether the batch operation is terminated immediately when it is necessary to truncate data to make it fit in the specified Destination (see page 9868).

Pascal

```
property AbortOnProblem: Boolean;
```

C++

```
__property Boolean AbortOnProblem;
```

Description

Set AbortOnProblem to true to cancel the batch operation when types of the fields in the Source (see page 9874) table records do not match the types of the fields in the Destination (see page 9868) table records. Set AbortOnProblem to false to cause the mismatched fields to be "trimmed". When AbortOnProblem is false, set the ProblemTableName (see page 9873) to have all the untrimmed versions of records that were trimmed saved to a separate table.

Trimming a field means converting its value to a value compatible with the destination field type. For example, if a field in the destination table holds a string of size 10 characters, and the corresponding field in the source table holds a string of size 15 characters, any values from the source table longer than 10 characters must be truncated. Values that cannot be converted will cause TBatchMove (see page 9858) to raise an exception.

Trimming does not occur when the Mode (see page 9871) property is set to batCopy. Trimming fields on a delete operation means that records in the destination table may be deleted that do not exactly match the values from the source table.

Note: To allow trimming on a field by field basis, use a TBDECallback (see page 9875) object.

See Also

Mode (see page 9871)

ProblemCount (see page 9872)

ProblemTableName (see page 9873)

TBDECallback (see page 9875)

Handling Batch Move Errors

53.1.3.5.3 TBatchMove.ChangedCount Property

Indicates the number of records from the Destination (see page 9868) table that are altered as a result of the batch operation.

Pascal

```
property ChangedCount: Longint;
```

C++

```
__property Longint ChangedCount;
```

Description

Use ChangedCount to learn the number of records in the Destination (see page 9868) table that are updated if the Mode (see page 9871) is batUpdate or batAppendUpdate, or to learn the number that are deleted if the Mode (see page 9871) is batDelete.

Set the ChangedTableName (see page 9867) property to save the unchanged versions of the altered records to a separate table. ChangedCount is then the number of records in that separate table.

See Also

ChangedTableName (see page 9867)

Destination (see page 9868)

Mode (see page 9871)

53.1.3.5.4 TBatchMove.ChangedTableName Property

Specifies the name of a local Paradox table that is created to hold copies of all records from the destination table changed by the batch operation.

Pascal

```
property ChangedTableName: TFileName;
```

C++

```
__property TFileName ChangedTableName;
```

Description

Set ChangedTableName to create a table that saves all unchanged versions of records that are changed by the batch operation. ChangedTableName must specify the name of a Paradox table. Setting ChangedTableName ensures that no information in the Destination (see page 9868) table gets lost when the records are altered or deleted.

When the Mode (see page 9871) property is batUpdate or batAppendUpdate, the new table will hold all records that were updated. When the Mode (see page 9871) property is batDelete, the new table will hold all records that were deleted.

The ChangedCount (see page 9866) property specifies the number of records in the new table.

See Also

ChangedCount (see page 9866)

Destination (see page 9868)

Mode (see page 9871)

Creating a Batch Move Component

Handling Batch Move Errors

53.1.3.5.5 TBatchMove.CommitCount Property

Specifies how many records are batch moved before a commit occurs.

Pascal

```
property CommitCount: Integer;
```

C++

```
__property int CommitCount;
```

Description

Use CommitCount to chunk batch operations into smaller operations. CommitCount allows an application to accommodate server transaction logs that are not big enough to handle the entire batch. Setting CommitCount to 0 will cause chunking into the number of records that will fit in 32Kb.

Note: If the Execute (see page 9863) method is called within an explicit transaction on the database containing the tables, CommitCount is ignored.

See Also

Execute (see page 9863)

Commit (see page 9971)

StartTransaction (see page 9977)

53.1.3.5.6 TBatchMove.Destination Property

Specifies the TTable (see page 10168) object for the database table that is the destination of the batch operation.

Pascal

```
property Destination: TTable;
```

C++

```
__property TTable Destination;
```

Description

Use Destination to indicate the database table that will be created or altered by the batch operation. The batch operation affects the Destination differently depending on the Mode (see page 9871):

Mode (see page 9871)	Effect on Destination
batAppend	Records are added to the destination table. No pre-existing records are altered. Destination table must already exist.
batUpdate	Records in the destination table with key fields matching the corresponding fields in the source are changed to match the source records. No new records are added. Destination table must already exist and must have an index defined.
batAppendUpdate	Records in the destination table with key fields matching the corresponding fields in the source are changed to match the source. Source (see page 9874) records that do not match key fields are added to the destination table. Destination table must already exist and must have an index defined.
batDelete	Records in the destination table that match the records in the source are deleted. Destination table must already exist and must have an index defined.
batCopy	Destination table is created to match the records in the source. If Destination already exists, it is replaced by the copy of the source.

See Also

Mode (see page 9871)

Source (see page 9874)

Creating a Batch Move Component

53.1.3.5.7 TBatchMove.KeyViolCount Property

Reports the number of records which could not be replaced, added, or deleted from Destination (see page 9868) because of integrity or key violations.

Pascal

```
property KeyViolCount: Longint;
```

C++

```
__property Longint KeyViolCount;
```

Description

Use KeyViolCount to learn the number of records form the Source (see page 9874) table that could not be applied to the Destination (see page 9868) table because of integrity or key violations. To obtain a table of the offending records from the Source (see page 9874) table, specify a KeyViolTableName (see page 9869).

Key violations occur when appending records to a table that has a primary key. On a table with a primary key, no two records may have the same values on all of the key fields. If a record that is being appended has the same values on all the key fields as a record in the destination table, it cannot be appended.

Integrity violations occur when deleting, appending or updating records to a table that has a referential integrity specification. A referential integrity specification requires that certain fields in one table must have values that exist in a particular field of another table. If a record that is being added does not match the restrictions imposed by referential integrity, it cannot be added. If deleting record would cause tighter referential integrity restrictions that another table cannot meet, it cannot be deleted.

Note: If AbortOnKeyViol (see page 9865) is true, KeyViolCount will never be greater than one, because the first violation will cause the batch operation to terminate.

See Also

AbortOnKeyViol (see page 9865)

Destination (see page 9868)

KeyViolTableName (see page 9869)

Mode (see page 9871)

Source (see page 9874)

53.1.3.5.8 TBatchMove.KeyViolTableName Property

Specifies the name of a Paradox table that will be created to contain all records from the Source (see page 9874) that cannot participate in the batch operation because of integrity or key violations.

Pascal

```
property KeyViolTableName: TFileName;
```

C++

```
__property TFileName KeyViolTableName;
```

Description

Use KeyViolTableName to obtain a table of records that could not be applied to the Destination (see page 9868) table because of integrity or key violations. KeyViolCount (see page 9869) has the number of records in the table that is created.

Key violations occur when appending records to a table that has a primary key. On a table with a primary key, no two records may have the same values on all of the key fields. If a record that is being appended has the same values on all the key fields as a record in the destination table, it cannot be appended.

Integrity violations occur when deleting, appending or updating records to a table that has a referential integrity specification. A referential integrity specification requires that certain fields in one table must have values that exist in a particular field of another table. If a record that is being added does not match the restrictions imposed by referential integrity, it cannot be added. If deleting record would cause tighter referential integrity restrictions that another table cannot meet, it cannot be deleted.

Note: If AbortOnKeyViol (see page 9865) is true, then there will be at most one record in this table because the operation aborts with that first record.

See Also

AbortOnKeyViol (see page 9865)

Destination (see page 9868)

KeyViolCount (see page 9869)

Source (see page 9874)

Creating a Batch Move Component

Handling Batch Move Errors

53.1.3.5.9 TBatchMove.Mappings Property

Specifies the column mappings for a batch operation.

Pascal

```
property Mappings: TStrings;
```

C++

```
__property TStrings Mappings;
```

Description

Set Mappings to specify the correspondence between fields in the Source (see page 9874) and fields in the Destination (see page 9868). By default TBatchMove (see page 9858) matches fields based on their position in the source and destination tables. That is, the first column in the source is matched with the first column in the destination, and so on. Mappings enables an application to override this default.

Mappings is a list of column mappings (one per line) in one of two forms. To map the column ColName in the source table to the column of the same name in the destination table, use:

```
ColName
```

To map the column named SourceColName in the source table to the column named DestColName in the destination table, use:

```
DestColName=SourceColName
```

When adding or appending records, fields in Destination (see page 9868) which have no entry in Mappings will be set to NULL. When copying a dataset, fields in Destination (see page 9868) which have no entry in Mappings will not appear as columns in the copy of the table.

If source and destination column data types are not the same, TBatchMove (see page 9858) can either cancel the batch

operation or perform a "best fit". If AbortOnProblem (see page 9866) is true, type mismatches will terminate the batch move. Otherwise, the field values from the Source (see page 9874) will be converted, if possible, to be compatible with the Destination (see page 9868).

See Also

AbortOnProblem (see page 9866)

Destination (see page 9868)

Mode (see page 9871)

Source (see page 9874)

Creating a Batch Move Component

Mapping Data Types

53.1.3.5.10 TBatchMove.Mode Property

Specifies what the TBatchMove (see page 9858) object does when the Execute (see page 9863) method is called.

Pascal

```
property Mode: TBatchMode;
```

C++

```
__property TBatchMode Mode;
```

Description

Use Mode to indicate whether the TBatchMove (see page 9858) object should add records, replace records, delete records, or copy the Source (see page 9874). These are the possible values for Mode:

Value	Meaning
batAppend	Append the records in the source to the destination table. The destination table must already exist, and the two tables must not have records with duplicate keys. This is the default mode.
batUpdate	Replace records in the destination table with matching records from the source table. The destination table must exist and must have an index defined to match records.
batAppendUpdate	If a matching record exists in the destination table, replace it. Otherwise, append records to the destination table. The destination table must exist and must have an index defined to match records.
batCopy	Create (see page 9862) the destination table based on the structure of the source table. If the destination already exists, the operation will delete it, and replace it with the new copy of the source.
batDelete	Delete records in the destination table that match records in the source table. The destination table must already exist and must have an index defined.

See Also

Destination (see page 9868)

Execute (see page 9863)

Source (see page 9874)

Creating a Batch Move Component

Specifying a Batch Move Mode

53.1.3.5.11 TBatchMove.MovedCount Property

Reports the number of records form the Source (see page 9874) which were applied to the destination.

Pascal

```
property MovedCount: Longint;
```

C++

```
__property Longint MovedCount;
```

Description

Read MovedCount to learn the number of records from the Source (see page 9874) that were read during the Execute (see page 9863) method. The MovedCount value includes any records which caused key or integrity violations, or which were trimmed.

See Also

Execute (see page 9863)

Source (see page 9874)

Executing a Batch Move

53.1.3.5.12 TBatchMove.ProblemCount Property

Indicates the number of records which could not be added to Destination (see page 9868) without loss of data due to a field type mismatch.

Pascal

```
property ProblemCount: Longint;
```

C++

```
__property Longint ProblemCount;
```

Description

Read ProblemCount to learn the number of records from the Source (see page 9874) that had field values which could not be mapped to destination fields without being "trimmed". If a ProblemTableName (see page 9873) is specified, ProblemCount is the number of records in that table.

Trimming a field means converting its value to a value compatible with the destination field type. For example, if a field in the destination table holds a string of size 10 characters, and the corresponding field in the source table holds a string of size 15 characters, any values from the source table longer than 10 characters must be truncated. Values that cannot be converted will cause TBatchMove (see page 9858) to raise an exception.

Trimming does not occur when the Mode (see page 9871) property is set to batCopy. Trimming fields on a delete operation means that records in the destination table may be deleted that do not exactly match the values from the source table.

Note: If AbortOnProblem (see page 9866) is true, then ProblemCount will be at most one, because the operation aborts when the problem occurs.

See Also

AbortOnProblem (see page 9866)

Destination (see page 9868)

Mode (see page 9871)

ProblemTableName (see page 9873)

Source (see page 9874)

Handling Batch Move Errors

53.1.3.5.13 TBatchMove.ProblemTableName Property

Specifies the name of a Paradox table that will be created to contain all records from the Source (see page 9874) that contain fields which were trimmed to match the corresponding field type in the Destination (see page 9868) table.

Pascal

```
property ProblemTableName: TFileName;
```

C++

```
__property TFileName ProblemTableName;
```

Description

Set ProblemTableName to create a table that holds records from the Source (see page 9874) that contain fields that had to be trimmed to be applied to the Destination (see page 9868). ProblemCount (see page 9872) will have the number of records placed in the new table. If ProblemTableName is not specified, the data in the record is still trimmed when applied to the destination table, but there will be no easy way to tell which fields were trimmed.

Trimming a field means converting its value to a value compatible with the destination field type. For example, if a field in the destination table holds a string of size 10 characters, and the corresponding field in the source table holds a string of size 15 characters, any values from the source table longer than 10 characters must be truncated. Values that cannot be converted will cause TBatchMove (see page 9858) to raise an exception.

Trimming does not occur when the Mode (see page 9871) property is set to batCopy. Trimming fields on a delete operation means that records in the destination table may be deleted that do not exactly match the values from the source table.

Note: If AbortOnProblem (see page 9866) is true, then there will be at most one record in this table because the operation aborts with the first record that has a problem.

See Also

AbortOnProblem (see page 9866)

Destination (see page 9868)

Mode (see page 9871)

ProblemCount (see page 9872)

Source (see page 9874)

Creating a Batch Move Component

Handling Batch Move Errors

53.1.3.5.14 TBatchMove.RecordCount Property

Specifies the maximum number of records that are applied to the Destination (see page 9868) when Execute (see page 9863) is called.

Pascal

```
property RecordCount: Longint;
```

C++

```
__property Longint RecordCount;
```

Description

Use RecordCount to limit the number of Source (see page 9874) records that are applied to the Destination (see page 9868). If zero, all records are added, subtracted, or copied, beginning with the first record in Source (see page 9874). If RecordCount is greater than zero, up to RecordCount records are applied to the Destination (see page 9868), starting with the current record in the dataset. If RecordCount exceeds the number of records remaining in Source (see page 9874), no wraparound occurs; the operation is terminated.

See Also

CommitCount (see page 9867)

Destination (see page 9868)

MovedCount (see page 9872)

Source (see page 9874)

Executing a Batch Move

53.1.3.5.15 TBatchMove.Source Property

Specifies the dataset that is the source of the batch operation.

Pascal

```
property Source: TBDEDataSet;
```

C++

```
__property TBDEDataSet Source;
```

Description

Use Source to indicate the dataset that contains the records that will be added to, subtracted from, or copied to the Destination (see page 9868). If Source is a TQuery (see page 10059) component, the query is executed before the move is executed. If a filter is currently in effect on Source, the filter conditions are respected when the batch move executes. The Mode (see page 9871) property indicates how the Source records are applied to the Destination (see page 9868):

Mode (see page 9871)	Effect on Destination (see page 9868)
batAppend	Records are added to the destination table.
batUpdate	Records in the destination table with key fields matching the corresponding fields in the source are changed to match the source records.
batAppendUpdate	Records in the destination table with key fields matching the corresponding fields in the source are changed to match the source. Source records that do not match key fields are added to the destination table.
batDelete	Records in the destination table that match the records in the source are deleted.
batCopy	Destination (see page 9868) table is created to match the records in the source.

See Also

Destination (see page 9868)

Mode (see page 9871)

TQuery (see page 10059)

TDataSet (see page 6894)

TTable (see page 10168)

Creating a Batch Move Component

53.1.3.5.16 TBatchMove.Transliterate Property

Specifies whether the data in the Source (see page 9874) records should be converted form the locale of the Source (see page 9874) to the locale of the Destination (see page 9868) when the Execute (see page 9863) method is called.

Pascal

```
property Transliterate: Boolean;
```

C++

```
__property Boolean Transliterate;
```

Description

Set Transliterate to true when the Source (see page 9874) dataset and the Destination (see page 9868) table use different language drivers and the data may contain extended ascii characters. Set Transliterate to false to avoid the overhead of the character set conversion when both the Source (see page 9874) and the Destination (see page 9868) use the same language driver or the data does not contain extended ascii characters.

See Also

Destination (see page 9868)

Execute (see page 9863)

Source (see page 9874)

Locale (see page 9934)

Creating a Batch Move Component

53.1.4 TBDECallback Class

TBDECallback is a wrapper for a Borland (see page 1761) Database Engine (BDE) callback function.

Class Hierarchy

DBTables.TBDECallback

Pascal

```
TBDECallback = class;
```

C++

```
class TBDECallback;
```

File

DBTables

Description

Use a TBDECallback object to register a callback function with the BDE. Callbacks allow an application to instruct the BDE to

take specific actions in response to events that occur during a BDE function call.

For example, the TBatchMove (see page 9858) object may encounter problems when executing the corresponding BDE batch operation. A callback function can respond to those problems by instructing the BDE to generate a problems table or to trim the data so that the operation succeeds.

To use a TBDECallback object, take the following steps:

1. Create (see page 9876) an instance of TBDECallback to register the callback with the BDE.
2. Call the BDE function in which the pertinent events take place. For example, create an instance of TBatchMove (see page 9858), and call its Execute method.
3. When the BDE function returns, destroy the TBDECallback object to unregister the callback function.

See Also

TBatchMove (see page 9858)

53.1.4.1 TBDECallback Members

The following tables list the members exposed by TBDECallback.

Protected Methods

	Name	Description
	Invoke (see page 9878)	This is Invoke, a member of class TBDECallback.

Public Methods

	Name	Description
	Create (see page 9876)	Creates an instance of TBDECallback (see page 9875) and registers the specified callback with the Borland (see page 1761) Database Engine (BDE).
	Destroy (see page 9877)	Unregisters the callback and destroys an instance of TBDECallback (see page 9875).

53.1.4.2 TBDECallback.Create Constructor

Creates an instance of TBDECallback (see page 9875) and registers the specified callback with the Borland (see page 1761) Database Engine (BDE).

Pascal

```
constructor Create(AOwner: TObject; Handle: hDBICur; CBType: CBType; CBBuf: Pointer;
CBBufSize: Integer; CallbackEvent: TBDECallbackEvent; Chain: Boolean);
```

C++

```
__fastcall TBDECallback(TObject * AOwner, hDBICur Handle, CBType CBType, void * CBBuf, int
CBBufSize, TBDECallbackEvent CallbackEvent, Boolean Chain);
```

Description

Call Create to register a callback with the BDE. Create a TBDECallback (see page 9875) object before calling any BDE function that should use the callback to respond to events.

When the BDE encounters the type of event indicated by the CBType parameter, the function specified by the CallbackEvent parameter is called. The CallbackEvent function receives a pointer to a callback descriptor. The type of the callback descriptor varies depending on the CBType. The Delphi application must allocate the memory for the callback descriptor. A pointer to this memory is passed in as the CBBuf parameter, and its size is given by the CBBufSize parameter.

The Chain parameter indicates whether this callback function should replace any other callbacks of this CBType. When Chain is false, the CallbackEvent function replaces any callbacks registered for the CBType. When Chain is true, the CallbackEvent

function is called in addition to other registered callbacks.

The Handle parameter is an optional parameter indicating the database cursor associated with the BDE function that may generate the callback. Provide a Handle to make the callback specific to function calls for that cursor.

The types of callback descriptors associated with each CBType are given in the following table. For more information, see the BDE documentation.

CBTypeUsagecallback descriptor

CBType	Usage	callback descriptor
cbGENPROGRESS	Respond to information about progress during large batch operations. For example, implement a progress bar or cancel button. Return cbrABORT to cancel, cbrCONTINUE to continue.	CBPROGRESSDesc
cbRESTRUCTURE	Instruct BDE on how to proceed with situations encountered during a table restructure.	Return cbrYES, to proceed with the default handling, cbrNO to proceed without the default behavior, cbrABORT to cancel the restructure.
cbBATCHRESULT	Instruct BDE on how to proceed with situations encountered during a batch operation. Return cbrYES, to proceed with the default handling, cbrNO to proceed without the default behavior, cbrABORT to cancel the batch operation.	RESTCbDesc
cbTABLECHANGED	Respond to notifications that a table has changed. The return value is ignored.	not used
cbCANCELQRY	Return cbrABORT to cancel a long running query, or cbrCONTINUE to keep going. The callback is called periodically during the execution of the query.	not used
cbINPUTREQ	Instruct BDE on how to respond when a file (such as a lookup table) is missing. Indicate the chosen behavior in the CBInputDesc and return cbrCONTINUE.	CBInputDesc
cbDBASELOGIN	Enter a password for an encrypted dBASE table and return cbrCONTINUE.	CBLoginDesc
cbFIELDRECALC	Respond to notifications that fields need to be recalculated. The return value is ignored.	not used
cbTRACE	Respond to trace information about the actions taken by the BDE. The return value is ignored.	TRACEDesc
cbDBLOGIN	Log in to a database server. Return cbrCONTINUE.	not used.
cbDELAYEDUPD	Respond to the notification when cached updates fail to write a modified record to the database. Return cbrABORT to cancel all cached updates, cbrSKIP or cbrCONTINUE to discard the failed update and continue posting records, cbrRETRY to try the failed commit again.	DELAYUPDCbDesc

Note: The VCL uses many BDE callback functions. Calling Create with Chain set to false may disable the expected behavior of some objects.

See Also

Destroy (see page 9877)

53.1.4.3 TBDECallback.Destroy Destructor

Unregisters the callback and destroys an instance of TBDECallback (see page 9875).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBDECallback();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which checks that the TBDECallback (see page 9875) object is not nil and only then calls Destroy.

Destroy unregisters the callback before freeing the memory associated with the TBDECallback (see page 9875) object. After unregistering the callback, any previous callback of the same type will be reinstated.

See Also

Create (see page 9876)

Free (see page 22300)

53.1.4.4 TBDECallback Methods

The methods of the TBDECallback class are listed here.

Protected Methods

	Name	Description
	Invoke (see page 9878)	This is Invoke, a member of class TBDECallback.

53.1.4.4.1 TBDECallback.Invoke Method

Pascal

```
function Invoke(CallType: CBType; CBInfo: Pointer): CBRType;
```

C++

```
__fastcall CBRType Invoke(CBType CallType, void * CBInfo);
```

Description

This is Invoke, a member of class TBDECallback.

53.1.5 TBDEDataSet Class

Encapsulates Borland (see page 1761) Database Engine (BDE) functionality for descendant dataset objects.

Class Hierarchy

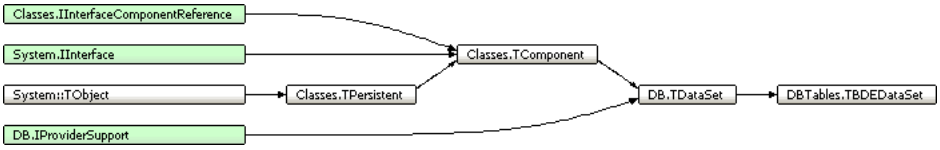

Pascal

```
TBDEDataSet = class(TDataSet);
```

C++

```
class TBDEDataSet : public TDataSet;
```

File

DBTables

Description

TBDEDataSet is a dataset object that defines BDE functionality for a dataset. Applications never use TBDEDataSet objects directly. Instead they use the descendants of TBDEDataSet and TDBDataSet (see page 10000), such as TQuery (see page 10059), TStoredProc (see page 10139), and TTable (see page 10168), which inherit its database-related properties and methods.

Developers who create custom dataset components that work through the BDE may want to derive them directly from TBDEDataSet to inherit all the functionality of TDataSet and the BDE-related properties and methods of TBDEDataSet. To inherit control over database objects, however, most developers should derive custom dataset components that use the BDE from the immediate descendant of TBDEDataSet, TDBDataSet (see page 10000).

See Also

TDataSet (see page 6894)

TDBDataSet (see page 10000)

TQuery (see page 10059)

TStoredProc (see page 10139)

TTable (see page 10168)

53.1.5.1 TBDEDataSet Members

The following tables list the members exposed by TBDEDataSet.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	ActivateFilters (see page 9893)	This is ActivateFilters, a member of class TBDEDataSet.
	AddFieldDesc (see page 9893)	This is AddFieldDesc, a member of class TBDEDataSet.
	AllocCachedUpdateBuffers (see page 9893)	This is AllocCachedUpdateBuffers, a member of class TBDEDataSet.
	AllocKeyBuffers (see page 9893)	This is AllocKeyBuffers, a member of class TBDEDataSet.
V	AllocRecordBuffer (see page 9894)	This is AllocRecordBuffer, a member of class TBDEDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 9894)	This is BlockReadNext, a member of class TBDEDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
	CachedUpdateCallBack (see page 9895)	This is CachedUpdateCallBack, a member of class TBDEDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.

	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
	CheckCachedUpdateMode (see page 9896)	This is CheckCachedUpdateMode, a member of class TBDEDataSet.
	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
	CheckSetKeyMode (see page 9896)	This is CheckSetKeyMode, a member of class TBDEDataSet.
	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
	ClearCalcFields (see page 9897)	This is ClearCalcFields, a member of class TBDEDataSet.
	CloseBlob (see page 9897)	This is CloseBlob, a member of class TBDEDataSet.
	CloseCursor (see page 9897)	This is CloseCursor, a member of class TBDEDataSet.
	CreateExprFilter (see page 9900)	This is CreateExprFilter, a member of class TBDEDataSet.
	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
	CreateFuncFilter (see page 9900)	This is CreateFuncFilter, a member of class TBDEDataSet.
	CreateHandle (see page 9900)	This is CreateHandle, a member of class TBDEDataSet.
	CreateLookupFilter (see page 9901)	This is CreateLookupFilter, a member of class TBDEDataSet.
	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
	DataEvent (see page 9901)	This is DataEvent, a member of class TBDEDataSet.
	DeactivateFilters (see page 9901)	This is DeactivateFilters, a member of class TBDEDataSet.
	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
	DestroyHandle (see page 9901)	This is DestroyHandle, a member of class TBDEDataSet.
	DestroyLookupCursor (see page 9902)	This is DestroyLookupCursor, a member of class TBDEDataSet.
	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FindRecord (see page 9903)	This is FindRecord, a member of class TBDEDataSet.
	ForceUpdateCallback (see page 9904)	This is ForceUpdateCallback, a member of class TBDEDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeKeyBuffers (see page 9904)	This is FreeKeyBuffers, a member of class TBDEDataSet.
	FreeRecordBuffer (see page 9904)	This is FreeRecordBuffer, a member of class TBDEDataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.

	GetBookmarkData (see page 9905)	This is GetBookmarkData, a member of class TBDEDataSet.
	GetBookmarkFlag (see page 9905)	This is GetBookmarkFlag, a member of class TBDEDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 9905)	This is GetCanModify, a member of class TBDEDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
	GetFieldFullName (see page 9907)	This is GetFieldFullName, a member of class TBDEDataSet.
	GetHandle (see page 9907)	This is GetHandle, a member of class TBDEDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIndexField (see page 9907)	This is GetIndexField, a member of class TBDEDataSet.
	GetIndexFieldCount (see page 9907)	This is GetIndexFieldCount, a member of class TBDEDataSet.
	GetIsIndexField (see page 9908)	This is GetIsIndexField, a member of class TBDEDataSet.
	GetKeyBuffer (see page 9908)	This is GetKeyBuffer, a member of class TBDEDataSet.
	GetKeyExclusive (see page 9908)	This is GetKeyExclusive, a member of class TBDEDataSet.
	GetKeyFieldCount (see page 9909)	This is GetKeyFieldCount, a member of class TBDEDataSet.
	GetLookupCursor (see page 9909)	This is GetLookupCursor, a member of class TBDEDataSet.
	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetObjectTypeNames (see page 9909)	This is GetObjectTypeNames, a member of class TBDEDataSet.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
	GetRecNo (see page 9909)	This is GetRecNo, a member of class TBDEDataSet.
	GetRecord (see page 9910)	This is GetRecord, a member of class TBDEDataSet.
	GetRecordCount (see page 9910)	This is GetRecordCount, a member of class TBDEDataSet.
	GetRecordSize (see page 9910)	This is GetRecordSize, a member of class TBDEDataSet.
	GetStateFieldValue (see page 9910)	This is GetStateFieldValue, a member of class TBDEDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetUpdateRecordSet (see page 9910)	This is GetUpdateRecordSet, a member of class TBDEDataSet.
	GetUpdatesPending (see page 9911)	This is GetUpdatesPending, a member of class TBDEDataSet.
	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
	InitKeyBuffer (see page 9911)	This is InitKeyBuffer, a member of class TBDEDataSet.
	InitRecord (see page 9911)	This is InitRecord, a member of class TBDEDataSet.
	InternalAddRecord (see page 9911)	This is InternalAddRecord, a member of class TBDEDataSet.
	InternalCancel (see page 9912)	This is InternalCancel, a member of class TBDEDataSet.
	InternalClose (see page 9912)	This is InternalClose, a member of class TBDEDataSet.
	InternalDelete (see page 9912)	This is InternalDelete, a member of class TBDEDataSet.
	InternalEdit (see page 9912)	This is InternalEdit, a member of class TBDEDataSet.
	InternalFirst (see page 9912)	This is InternalFirst, a member of class TBDEDataSet.
	InternalGotoBookmark (see page 9913)	This is InternalGotoBookmark, a member of class TBDEDataSet.
	InternalHandleException (see page 9913)	This is InternalHandleException, a member of class TBDEDataSet.
	InternalInitFieldDefs (see page 9913)	This is InternalInitFieldDefs, a member of class TBDEDataSet.
	InternalInitRecord (see page 9913)	This is InternalInitRecord, a member of class TBDEDataSet.

	InternalInsert (see page 9914)	This is InternalInsert, a member of class TBDEDataSet.
	InternalLast (see page 9914)	This is InternalLast, a member of class TBDEDataSet.
	InternalOpen (see page 9914)	This is InternalOpen, a member of class TBDEDataSet.
	InternalPost (see page 9914)	This is InternalPost, a member of class TBDEDataSet.
	InternalRefresh (see page 9914)	This is InternalRefresh, a member of class TBDEDataSet.
	InternalSetToRecord (see page 9915)	This is InternalSetToRecord, a member of class TBDEDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsCursorOpen (see page 9915)	This is IsCursorOpen, a member of class TBDEDataSet.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 9916)	This is LocateRecord, a member of class TBDEDataSet.
	MapsToIndex (see page 9918)	This is MapsToIndex, a member of class TBDEDataSet.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	OpenCursor (see page 9918)	This is OpenCursor, a member of class TBDEDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PostKeyBuffer (see page 9919)	This is PostKeyBuffer, a member of class TBDEDataSet.
	PrepareCursor (see page 9919)	This is PrepareCursor, a member of class TBDEDataSet.
	ProcessUpdates (see page 9919)	This is ProcessUpdates, a member of class TBDEDataSet.
	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 6982)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandType (see page 6982)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetUpdateException (see page 9920)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 9920)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 9921)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 9921)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	ResetCursorRange (see page 9922)	This is ResetCursorRange, a member of class TBDEDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.

	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBlockReadSize (see page 9922)	This is SetBlockReadSize, a member of class TBDEDataSet.
	SetBookmarkData (see page 9922)	This is SetBookmarkData, a member of class TBDEDataSet.
	SetBookmarkFlag (see page 9923)	This is SetBookmarkFlag, a member of class TBDEDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetCachedUpdates (see page 9923)	This is SetCachedUpdates, a member of class TBDEDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetCursorRange (see page 9923)	This is SetCursorRange, a member of class TBDEDataSet.
	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldData (see page 9923)	This is SetFieldData, a member of class TBDEDataSet.
	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
	SetFilterData (see page 9923)	This is SetFilterData, a member of class TBDEDataSet.
	SetFiltered (see page 9924)	This is SetFiltered, a member of class TBDEDataSet.
	SetFilterHandle (see page 9924)	This is SetFilterHandle, a member of class TBDEDataSet.
	SetFilterOptions (see page 9924)	This is SetFilterOptions, a member of class TBDEDataSet.
	SetFilterText (see page 9924)	This is SetFilterText, a member of class TBDEDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetIndexField (see page 9925)	This is SetIndexField, a member of class TBDEDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetKeyBuffer (see page 9925)	This is SetKeyBuffer, a member of class TBDEDataSet.
	SetKeyExclusive (see page 9925)	This is SetKeyExclusive, a member of class TBDEDataSet.
	SetKeyFieldCount (see page 9925)	This is SetKeyFieldCount, a member of class TBDEDataSet.
	SetKeyFields (see page 9925)	This is SetKeyFields, a member of class TBDEDataSet.
	SetLinkRanges (see page 9926)	This is SetLinkRanges, a member of class TBDEDataSet.
	SetLocale (see page 9926)	This is SetLocale, a member of class TBDEDataSet.
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
	SetOnFilterRecord (see page 9926)	This is SetOnFilterRecord, a member of class TBDEDataSet.
	SetOnUpdateError (see page 9926)	This is SetOnUpdateError, a member of class TBDEDataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetRecNo (see page 9927)	This is SetRecNo, a member of class TBDEDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
	SetStateFieldValue (see page 9927)	This is SetStateFieldValue, a member of class TBDEDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	SetupCallBack (see page 9927)	This is SetupCallBack, a member of class TBDEDataSet.
	SetUpdateObject (see page 9927)	This is SetUpdateObject, a member of class TBDEDataSet.
	SetUpdateRecordSet (see page 9927)	This is SetUpdateRecordSet, a member of class TBDEDataSet.
	SwitchToIndex (see page 9928)	This is SwitchToIndex, a member of class TBDEDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	UpdateCallbackRequired (see page 9928)	This is UpdateCallbackRequired, a member of class TBDEDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
	NestedDataSets (see page 7030)	Lists all nested datasets.
	Reserved (see page 7035)	For internal use only.
	StmtHandle (see page 9935)	Specifies the Borland (see page 1761) Database Engine (BDE) statement handle for the dataset.

Public Events

	Name	Description
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
	ApplyUpdates (see page 9894)	Writes a dataset's pending cached updates to the database.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BookmarkValid (see page 9895)	Tests the validity of a specified bookmark.
	Cancel (see page 9895)	Cancels modifications to the current record if those changes are not yet posted.
	CancelUpdates (see page 9896)	Clears all pending cached updates from the cache and restores the dataset its prior state.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Close (see page 6920)	Closes a dataset.
	CommitUpdates (see page 9897)	Clears the cached updates buffer.
V	CompareBookmarks (see page 9898)	Indicates the relationship between two bookmarks.
	ConstraintCallBack (see page 9898)	Specifies the callback function that allows the dataset to communicate with the BDE constraint layer.
	ConstraintsDisabled (see page 9899)	Indicates whether constraints are enabled for a dataset.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
V	Create (see page 9889)	Creates an instance of a TBDEDataSet (see page 9878) component.
V	CreateBlobStream (see page 9899)	Returns a TStream object for reading or writing the data in a specified blob field.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
V	Destroy (see page 9890)	Destroys the instance of a dataset component.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableConstraints (see page 9902)	Disables application of server constraints on the dataset.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableConstraints (see page 9902)	Enables application of server constraints against dataset entries.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FetchAll (see page 9903)	Retrieves all records from the current cursor position to the end of the file and stores them locally.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	FlushBuffers (see page 9903)	Posts all changes that have been written to the record buffer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetBlobFieldData (see page 9904)	Reads BLOB data into a buffer.

	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentRecord (see page 9905)	Retrieves the current record into a buffer.
	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
	GetFieldData (see page 9906)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetIndexInfo (see page 9907)	Retrieves information about the current index into the index data fields of the dataset.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
	IsSequenced (see page 9915)	IsSequenced indicates whether sequence numbers are available for database records.
	Last (see page 6968)	Moves to the last record in the dataset.
	Locate (see page 9915)	Searches the dataset for a specified record and makes that record the current record.
	Lookup (see page 9916)	Retrieves field values from a record that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	Open (see page 6976)	Opens the dataset.
	Post (see page 9918)	Writes a modified record to the database.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 7579)	Gets the SQL command to be executed by PSExecute (see page 7578).
	PSGetCommandType (see page 7580)	Gets the command type to be executed by PSExecute (see page 7578).
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.

	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSSetCommandText (see page 7585)	Assigns a specified SQL command to be executed by PSExecute (see page 7578).
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
	RevertRecord (see page 9922)	Restores the current record in the dataset to an unmodified state when cached updates are enabled.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Translate (see page 9928)	Converts a data string between the Microsoft ANSI character set, and the local code page (OEM character set).
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 9928)	Reports the update status for the current record.

Public Properties

	Name	Description
R	AggFields (see page 7006)	Contains any aggregate fields.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
	CacheBlobs (see page 9932)	Determines whether BLOB fields are cached in memory.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
R	DataSource (see page 7014)	Represents the data source of another dataset that supplies values to the dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.

R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	ExpIndex (see page 9933)	Indicates whether a dataset is using a dBASE expression index.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
	FieldDefs (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
R	Handle (see page 9933)	Specifies the Borland (see page 1761) Database Engine (BDE) cursor handle for the dataset.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	KeySize (see page 9934)	Specifies the size of the key for the current index of the dataset.
R	Locale (see page 9934)	Identifies the Borland (see page 1761) Database Engine (BDE) language driver for the dataset.
R	Modified (see page 7029)	Indicates whether the active record is modified.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	UpdateObject (see page 9936)	Specifies the update object component used to update a read-only result set when cached updates are enabled.
	UpdateRecordTypes (see page 9936)	Specifies the type of records visible in a dataset when cached updates are enabled.
R	UpdatesPending (see page 9937)	Indicates whether the cached updates buffer contains records that are not yet applied.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterCancel (see page 9938)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 9940)	Occurs after an application closes a dataset.
	AfterDelete (see page 9942)	Occurs after an application deletes a record.
	AfterEdit (see page 9944)	Occurs after an application starts editing a record.
	AfterInsert (see page 9945)	Occurs after an application inserts a new record.
	AfterOpen (see page 9946)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 9947)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 9947)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 9948)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 9948)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 9949)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 9949)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 9950)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 9950)	Occurs before an application enters insert mode.

	BeforeOpen (see page 9952)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 9952)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 9953)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 9953)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 9954)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 9955)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 9955)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnNewRecord (see page 9955)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 9956)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnUpdateError (see page 9956)	Occurs if an exception is generated when cached updates are applied to a database.
	OnUpdateRecord (see page 9957)	Occurs when cached updates are applied to a record.

Published Properties

	Name	Description
	Active (see page 9929)	Specifies whether or not a dataset is open.
	AutoCalcFields (see page 9931)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	CachedUpdates (see page 9932)	Specifies whether cached updates are enabled for a dataset.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectView (see page 9935)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	Tag (see page 2808)	Stores an integer value as part of a component.

53.1.5.2 TBDEDataSet.Create Constructor

Creates an instance of a TBDEDataSet (see page 9878) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBDEDataSet(TComponent * AOwner);
```

Description

Call Create to instantiate a dataset component at runtime. Ordinarily applications instantiate TBDEDataSet (see page 9878) descendants, such as TTable (see page 10168), TQuery (see page 10059), or TStoredProc (see page 10139).

Create

Calls its inherited Create method.

Set the Locale (see page 9934) property of the dataset to the locale of the default session.

Sets the CacheBlobs (see page 9932) property to true.

See Also

CacheBlobs (see page 9932)

Destroy (see page 9890)

Locale (see page 9934)

TQuery (see page 10059)

TStoredProc (see page 10139)

TTable (see page 10168)

53.1.5.3 TBDEDataSet.Destroy Destructor

Destroys the instance of a dataset component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBDEDataSet();
```

Description

Do not call Destroy directly in an application. Usually destruction of datasets is handled automatically If an application creates its own instances of a dataset, however, the application should call Free, which checks that the dataset component is not nil before calling Destroy.

Destroy calls its inherited destructor if necessary, frees the resources allocated for the BDE asynchronous callback, and sets the UpdateObject (see page 9936) property for the dataset to nil.

See Also

Create (see page 9889)

UpdateObject (see page 9936)

Free (see page 22300)

53.1.5.4 TBDEDataSet Methods

The methods of the TBDEDataSet class are listed here.

Protected Methods

	Name	Description
	ActivateFilters (see page 9893)	This is ActivateFilters, a member of class TBDEDataSet.
	AddFieldDesc (see page 9893)	This is AddFieldDesc, a member of class TBDEDataSet.
	AllocCachedUpdateBuffers (see page 9893)	This is AllocCachedUpdateBuffers, a member of class TBDEDataSet.
	AllocKeyBuffers (see page 9893)	This is AllocKeyBuffers, a member of class TBDEDataSet.
V	AllocRecordBuffer (see page 9894)	This is AllocRecordBuffer, a member of class TBDEDataSet.
V	BlockReadNext (see page 9894)	This is BlockReadNext, a member of class TBDEDataSet.
	CachedUpdateCallBack (see page 9895)	This is CachedUpdateCallBack, a member of class TBDEDataSet.
	CheckCachedUpdateMode (see page 9896)	This is CheckCachedUpdateMode, a member of class TBDEDataSet.
	CheckSetKeyMode (see page 9896)	This is CheckSetKeyMode, a member of class TBDEDataSet.
V	ClearCalcFields (see page 9897)	This is ClearCalcFields, a member of class TBDEDataSet.
V	CloseBlob (see page 9897)	This is CloseBlob, a member of class TBDEDataSet.
V	CloseCursor (see page 9897)	This is CloseCursor, a member of class TBDEDataSet.
	CreateExprFilter (see page 9900)	This is CreateExprFilter, a member of class TBDEDataSet.
	CreateFuncFilter (see page 9900)	This is CreateFuncFilter, a member of class TBDEDataSet.
V	CreateHandle (see page 9900)	This is CreateHandle, a member of class TBDEDataSet.
	CreateLookupFilter (see page 9901)	This is CreateLookupFilter, a member of class TBDEDataSet.
V	DataEvent (see page 9901)	This is DataEvent, a member of class TBDEDataSet.
	DeactivateFilters (see page 9901)	This is DeactivateFilters, a member of class TBDEDataSet.
V	DestroyHandle (see page 9901)	This is DestroyHandle, a member of class TBDEDataSet.

	DestroyLookupCursor (see page 9902)	This is DestroyLookupCursor, a member of class TBDEDataSet.
	FindRecord (see page 9903)	This is FindRecord, a member of class TBDEDataSet.
	ForceUpdateCallback (see page 9904)	This is ForceUpdateCallback, a member of class TBDEDataSet.
	FreeKeyBuffers (see page 9904)	This is FreeKeyBuffers, a member of class TBDEDataSet.
	FreeRecordBuffer (see page 9904)	This is FreeRecordBuffer, a member of class TBDEDataSet.
	GetBookmarkData (see page 9905)	This is GetBookmarkData, a member of class TBDEDataSet.
	GetBookmarkFlag (see page 9905)	This is GetBookmarkFlag, a member of class TBDEDataSet.
	GetCanModify (see page 9905)	This is GetCanModify, a member of class TBDEDataSet.
	GetFieldFullName (see page 9907)	This is GetFieldFullName, a member of class TBDEDataSet.
	GetHandle (see page 9907)	This is GetHandle, a member of class TBDEDataSet.
	GetIndexField (see page 9907)	This is GetIndexField, a member of class TBDEDataSet.
	GetIndexFieldCount (see page 9907)	This is GetIndexFieldCount, a member of class TBDEDataSet.
	GetIsIndexField (see page 9908)	This is GetIsIndexField, a member of class TBDEDataSet.
	GetKeyBuffer (see page 9908)	This is GetKeyBuffer, a member of class TBDEDataSet.
	GetKeyExclusive (see page 9908)	This is GetKeyExclusive, a member of class TBDEDataSet.
	GetKeyFieldCount (see page 9909)	This is GetKeyFieldCount, a member of class TBDEDataSet.
	GetLookupCursor (see page 9909)	This is GetLookupCursor, a member of class TBDEDataSet.
	GetObjectTypeNames (see page 9909)	This is GetObjectTypeNames, a member of class TBDEDataSet.
	GetRecNo (see page 9909)	This is GetRecNo, a member of class TBDEDataSet.
	GetRecord (see page 9910)	This is GetRecord, a member of class TBDEDataSet.
	GetRecordCount (see page 9910)	This is GetRecordCount, a member of class TBDEDataSet.
	GetRecordSize (see page 9910)	This is GetRecordSize, a member of class TBDEDataSet.
	GetStateFieldValue (see page 9910)	This is GetStateFieldValue, a member of class TBDEDataSet.
	GetUpdateRecordSet (see page 9910)	This is GetUpdateRecordSet, a member of class TBDEDataSet.
	GetUpdatesPending (see page 9911)	This is GetUpdatesPending, a member of class TBDEDataSet.
	InitKeyBuffer (see page 9911)	This is InitKeyBuffer, a member of class TBDEDataSet.
	InitRecord (see page 9911)	This is InitRecord, a member of class TBDEDataSet.
	InternalAddRecord (see page 9911)	This is InternalAddRecord, a member of class TBDEDataSet.
	InternalCancel (see page 9912)	This is InternalCancel, a member of class TBDEDataSet.
	InternalClose (see page 9912)	This is InternalClose, a member of class TBDEDataSet.
	InternalDelete (see page 9912)	This is InternalDelete, a member of class TBDEDataSet.
	InternalEdit (see page 9912)	This is InternalEdit, a member of class TBDEDataSet.
	InternalFirst (see page 9912)	This is InternalFirst, a member of class TBDEDataSet.
	InternalGotoBookmark (see page 9913)	This is InternalGotoBookmark, a member of class TBDEDataSet.
	InternalHandleException (see page 9913)	This is InternalHandleException, a member of class TBDEDataSet.
	InternalInitFieldDefs (see page 9913)	This is InternalInitFieldDefs, a member of class TBDEDataSet.
	InternalInitRecord (see page 9913)	This is InternalInitRecord, a member of class TBDEDataSet.
	InternalInsert (see page 9914)	This is InternalInsert, a member of class TBDEDataSet.
	InternalLast (see page 9914)	This is InternalLast, a member of class TBDEDataSet.
	InternalOpen (see page 9914)	This is InternalOpen, a member of class TBDEDataSet.
	InternalPost (see page 9914)	This is InternalPost, a member of class TBDEDataSet.
	InternalRefresh (see page 9914)	This is InternalRefresh, a member of class TBDEDataSet.
	InternalSetToRecord (see page 9915)	This is InternalSetToRecord, a member of class TBDEDataSet.
	IsCursorOpen (see page 9915)	This is IsCursorOpen, a member of class TBDEDataSet.
	LocateRecord (see page 9916)	This is LocateRecord, a member of class TBDEDataSet.
	MapsToIndex (see page 9918)	This is MapsToIndex, a member of class TBDEDataSet.
	OpenCursor (see page 9918)	This is OpenCursor, a member of class TBDEDataSet.
	PostKeyBuffer (see page 9919)	This is PostKeyBuffer, a member of class TBDEDataSet.
	PrepareCursor (see page 9919)	This is PrepareCursor, a member of class TBDEDataSet.
	ProcessUpdates (see page 9919)	This is ProcessUpdates, a member of class TBDEDataSet.
	PSGetUpdateException (see page 9920)	Generates an EUpdateError object based on another exception object.
	PSIsSQLSupported (see page 9920)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 9921)	Resets the dataset so that it is positioned on the first record.
	PSUpdateRecord (see page 9921)	Applies a single update to the underlying database table or tables.

	ResetCursorRange (see page 9922)	This is ResetCursorRange, a member of class TBDEDataSet.
	SetBlockReadSize (see page 9922)	This is SetBlockReadSize, a member of class TBDEDataSet.
	SetBookmarkData (see page 9922)	This is SetBookmarkData, a member of class TBDEDataSet.
	SetBookmarkFlag (see page 9923)	This is SetBookmarkFlag, a member of class TBDEDataSet.
	SetCachedUpdates (see page 9923)	This is SetCachedUpdates, a member of class TBDEDataSet.
	SetCursorRange (see page 9923)	This is SetCursorRange, a member of class TBDEDataSet.
	SetFieldData (see page 9923)	This is SetFieldData, a member of class TBDEDataSet.
	SetFilterData (see page 9923)	This is SetFilterData, a member of class TBDEDataSet.
	SetFiltered (see page 9924)	This is SetFiltered, a member of class TBDEDataSet.
	SetFilterHandle (see page 9924)	This is SetFilterHandle, a member of class TBDEDataSet.
	SetFilterOptions (see page 9924)	This is SetFilterOptions, a member of class TBDEDataSet.
	SetFilterText (see page 9924)	This is SetFilterText, a member of class TBDEDataSet.
	SetIndexField (see page 9925)	This is SetIndexField, a member of class TBDEDataSet.
	SetKeyBuffer (see page 9925)	This is SetKeyBuffer, a member of class TBDEDataSet.
	SetKeyExclusive (see page 9925)	This is SetKeyExclusive, a member of class TBDEDataSet.
	SetKeyFieldCount (see page 9925)	This is SetKeyFieldCount, a member of class TBDEDataSet.
	SetKeyFields (see page 9925)	This is SetKeyFields, a member of class TBDEDataSet.
	SetLinkRanges (see page 9926)	This is SetLinkRanges, a member of class TBDEDataSet.
	SetLocale (see page 9926)	This is SetLocale, a member of class TBDEDataSet.
	SetOnFilterRecord (see page 9926)	This is SetOnFilterRecord, a member of class TBDEDataSet.
	SetOnUpdateError (see page 9926)	This is SetOnUpdateError, a member of class TBDEDataSet.
	SetRecNo (see page 9927)	This is SetRecNo, a member of class TBDEDataSet.
	SetStateFieldValue (see page 9927)	This is SetStateFieldValue, a member of class TBDEDataSet.
	SetupCallBack (see page 9927)	This is SetupCallBack, a member of class TBDEDataSet.
	SetUpdateObject (see page 9927)	This is SetUpdateObject, a member of class TBDEDataSet.
	SetUpdateRecordSet (see page 9927)	This is SetUpdateRecordSet, a member of class TBDEDataSet.
	SwitchToIndex (see page 9928)	This is SwitchToIndex, a member of class TBDEDataSet.
	UpdateCallbackRequired (see page 9928)	This is UpdateCallbackRequired, a member of class TBDEDataSet.

Public Methods

	Name	Description
	ApplyUpdates (see page 9894)	Writes a dataset's pending cached updates to the database.
	BookmarkValid (see page 9895)	Tests the validity of a specified bookmark.
	Cancel (see page 9895)	Cancels modifications to the current record if those changes are not yet posted.
	CancelUpdates (see page 9896)	Clears all pending cached updates from the cache and restores the dataset its prior state.
	CommitUpdates (see page 9897)	Clears the cached updates buffer.
	CompareBookmarks (see page 9898)	Indicates the relationship between two bookmarks.
	ConstraintCallBack (see page 9898)	Specifies the callback function that allows the dataset to communicate with the BDE constraint layer.
	ConstraintsDisabled (see page 9899)	Indicates whether constraints are enabled for a dataset.
	CreateBlobStream (see page 9899)	Returns a TStream object for reading or writing the data in a specified blob field.
	DisableConstraints (see page 9902)	Disables application of server constraints on the dataset.
	EnableConstraints (see page 9902)	Enables application of server constraints against dataset entries.
	FetchAll (see page 9903)	Retrieves all records from the current cursor position to the end of the file and stores them locally.
	FlushBuffers (see page 9903)	Posts all changes that have been written to the record buffer.
	GetBlobFieldData (see page 9904)	Reads BLOB data into a buffer.
	GetCurrentRecord (see page 9905)	Retrieves the current record into a buffer.
	GetFieldData (see page 9906)	This is the overview for the GetFieldData method overload.
	GetIndexInfo (see page 9907)	Retrieves information about the current index into the index data fields of the dataset.
	IsSequenced (see page 9915)	IsSequenced indicates whether sequence numbers are available for database records.
	Locate (see page 9915)	Searches the dataset for a specified record and makes that record the current record.

	Lookup (see page 9916)	Retrieves field values from a record that matches specified search values.
	Post (see page 9918)	Writes a modified record to the database.
	RevertRecord (see page 9922)	Restores the current record in the dataset to an unmodified state when cached updates are enabled.
	Translate (see page 9928)	Converts a data string between the Microsoft ANSI character set, and the local code page (OEM character set).
	UpdateStatus (see page 9928)	Reports the update status for the current record.

53.1.5.4.1 TBDEDataSet.ActivateFilters Method

Pascal

```
procedure ActivateFilters;
```

C++

```
__fastcall ActivateFilters();
```

Description

This is ActivateFilters, a member of class TBDEDataSet.

53.1.5.4.2 TBDEDataSet.AddFieldDesc Method

Pascal

```
procedure AddFieldDesc(FieldDescs: TFieldDescList; var DescNo: Integer; var FieldID: Integer;
RequiredFields: TBits; FieldDefs: TFieldDefs);
```

C++

```
__fastcall AddFieldDesc(TFieldDescList FieldDescs, int DescNo, int FieldID, TBits
RequiredFields, TFieldDefs FieldDefs);
```

Description

This is AddFieldDesc, a member of class TBDEDataSet.

53.1.5.4.3 TBDEDataSet.AllocCachedUpdateBuffers Method

Pascal

```
procedure AllocCachedUpdateBuffers(Allocate: Boolean);
```

C++

```
__fastcall AllocCachedUpdateBuffers(Boolean Allocate);
```

Description

This is AllocCachedUpdateBuffers, a member of class TBDEDataSet.

53.1.5.4.4 TBDEDataSet.AllocKeyBuffers Method

Pascal

```
procedure AllocKeyBuffers;
```

C++

```
__fastcall AllocKeyBuffers();
```

Description

This is AllocKeyBuffers, a member of class TBDEDataSet.

53.1.5.4.5 TBDEDataSet.AllocRecordBuffer Method

Pascal

```
function AllocRecordBuffer: PChar; override;
```

C++

```
virtual __fastcall const char * AllocRecordBuffer();
```

Description

This is AllocRecordBuffer, a member of class TBDEDataSet.

53.1.5.4.6 TBDEDataSet.ApplyUpdates Method

Writes a dataset's pending cached updates to the database.

Pascal

```
procedure ApplyUpdates;
```

C++

```
__fastcall ApplyUpdates();
```

Description

Call ApplyUpdates to write a dataset's pending cached updates to a database. This method passes cached data to the database for storage, but the changes are not committed to the database. An application must explicitly call the database component's Commit method to commit the changes to the database if the write is successful, or call the database's Rollback method to undo the changes if there is an error.

Following a successful write to the database, and following a successful call to the database's Commit method, an application should call the CommitUpdates (see page 9897) method to clear the cached update buffer.

Note: The preferred method for updating datasets is to call a database component's ApplyUpdates method rather than to call each individual dataset's ApplyUpdates method. The database component's ApplyUpdates method takes care of committing and rolling back transactions and clearing the cache when the operation is successful.

See Also

CachedUpdates (see page 9932)

UpdatesPending (see page 9937)

ApplyUpdates (see page 9970)

Using TTable

Applying BDE-based Cached Updates

Applying Cached Updates with Dataset Component Methods

53.1.5.4.7 TBDEDataSet.BlockReadNext Method

Pascal

```
procedure BlockReadNext; override;
```

C++

```
virtual __fastcall BlockReadNext();
```

Description

This is BlockReadNext, a member of class TBDEDataSet.

53.1.5.4.8 TBDEDataSet.BookmarkValid Method

Tests the validity of a specified bookmark.

Pascal

```
function BookmarkValid(Bookmark: TBookmark): Boolean; override;
```

C++

```
virtual __fastcall Boolean BookmarkValid(TBookmark Bookmark);
```

Description

Call BookmarkValid to determine if a specified bookmark is currently assigned a value. Bookmark specifies the bookmark to test.

BookmarkValid returns true if a bookmark is valid. Otherwise, it returns false.

See Also

CompareBookmarks (see page 9898)

TTable (see page 10168)

53.1.5.4.9 TBDEDataSet.CachedUpdateCallBack Method

Pascal

```
function CachedUpdateCallBack(CBInfo: Pointer): CBRType;
```

C++

```
__fastcall CBRType CachedUpdateCallBack(void * CBInfo);
```

Description

This is CachedUpdateCallBack, a member of class TBDEDataSet.

53.1.5.4.10 TBDEDataSet.Cancel Method

Cancels modifications to the current record if those changes are not yet posted.

Pascal

```
procedure Cancel; override;
```

C++

```
virtual __fastcall Cancel();
```

Description

Call Cancel to undo modifications made to one or more fields belonging to the current record. As long as those changes are not already posted to the database, Cancel returns the record to its previous state, and sets the dataset state to dsBrowse.

Typically Cancel is used to back out of changes in response to user request, or in field validation routines that back out illegal field values. The TDBNavigator object contains a Cancel button that triggers a call to Cancel.

See Also

State (see page 7036)

Post (see page 9918)

TDBNavigator (see page 9230)

53.1.5.4.11 TBDEDataSet.CancelUpdates Method

Clears all pending cached updates from the cache and restores the dataset its prior state.

Pascal

```
procedure CancelUpdates;
```

C++

```
__fastcall CancelUpdates();
```

Description

Call CancelUpdates to clear all pending cached updates from the cache and restore the dataset to the state it was in when the table was opened, cached updates were last enabled, or updates were last successfully applied to the database.

When a dataset is closed, or the CachedUpdates (see page 9932) property is set to false, CancelUpdates is called automatically.

Note: To undo changes to a single record, call RevertRecord (see page 9922).

See Also

CachedUpdates (see page 9932)

State (see page 7036)

ApplyUpdates (see page 9894)

RevertRecord (see page 9922)

UpdatesPending (see page 9937)

Enabling BDE-based Cached Updates

Using the BDE to Cache Updates

53.1.5.4.12 TBDEDataSet.CheckCachedUpdateMode Method

Pascal

```
procedure CheckCachedUpdateMode;
```

C++

```
__fastcall CheckCachedUpdateMode();
```

Description

This is CheckCachedUpdateMode, a member of class TBDEDataSet.

53.1.5.4.13 TBDEDataSet.CheckSetKeyMode Method

Pascal

```
procedure CheckSetKeyMode;
```

C++

```
__fastcall CheckSetKeyMode();
```

Description

This is CheckSetKeyMode, a member of class TBDEDataSet.

53.1.5.4.14 TBDEDataSet.ClearCalcFields Method

Pascal

```
procedure ClearCalcFields(Buffer: PChar); override;
```

C++

```
virtual __fastcall ClearCalcFields(const char * Buffer);
```

Description

This is ClearCalcFields, a member of class TBDEDataSet.

53.1.5.4.15 TBDEDataSet.CloseBlob Method

Pascal

```
procedure CloseBlob(Field: TField); override;
```

C++

```
virtual __fastcall CloseBlob(TField Field);
```

Description

This is CloseBlob, a member of class TBDEDataSet.

53.1.5.4.16 TBDEDataSet.CloseCursor Method

Pascal

```
procedure CloseCursor; override;
```

C++

```
virtual __fastcall CloseCursor();
```

Description

This is CloseCursor, a member of class TBDEDataSet.

53.1.5.4.17 TBDEDataSet.CommitUpdates Method

Clears the cached updates buffer.

Pascal

```
procedure CommitUpdates;
```

C++

```
__fastcall CommitUpdates();
```

Description

Call CommitUpdates to clear the cached updates buffer after both a successful call to ApplyUpdates (see page 9894) and a database component's Commit method. Clearing the cache after applying updates ensures that the cache is empty except for records that could not be processed and were skipped by the OnUpdateRecord (see page 9957) or OnUpdateError (see page 9956) event handlers. An application can attempt to modify the records still in the cache.

Record modifications made after a call to CommitUpdates repopulate the cached update buffer and require a subsequent call to ApplyUpdates (see page 9894) to move them to the database.

Note: Applications that use a database component's ApplyUpdates (see page 9894) method to apply and commit pending updates for all datasets associated with the database component do not need to call CommitUpdates.

See Also

CachedUpdates (see page 9932)

UpdatesPending (see page 9937)

OnUpdateError (see page 9956)

OnUpdateRecord (see page 9957)

ApplyUpdates (see page 9894)

ApplyUpdates (see page 9970)

Using the BDE to Cache Updates

Applying BDE-based Cached Updates

Applying Cached Updates with Dataset Component Methods

53.1.5.4.18 TBDEDataSet.CompareBookmarks Method

Indicates the relationship between two bookmarks.

Pascal

```
function CompareBookmarks(Bookmark1: TBookmark; Bookmark2: TBookmark): Integer; override;
```

C++

```
virtual __fastcall int CompareBookmarks(TBookmark Bookmark1, TBookmark Bookmark2);
```

Description

Call CompareBookmarks to determine if two bookmarks are identical. Bookmark1 and Bookmark2 are the bookmarks to compare.

If the bookmarks differ, CompareBookmarks returns -1 if Bookmark1 is less than Bookmark2, 1 if Bookmark1 is greater than Bookmark2, and 0 if the bookmarks are identical, nil (Delphi), or NULL (C++)

See Also

BookmarkValid (see page 9895)

TTable (see page 10168)

53.1.5.4.19 TBDEDataSet.ConstraintCallBack Method

Specifies the callback function that allows the dataset to communicate with the BDE constraint layer.

Pascal

```
function ConstraintCallBack(Req: DsInfoReq; var ADataSources: DataSources): DBIResult; stdcall;
```

C++

```
__fastcall __stdcall DBIResult ConstraintCallBack(DsInfoReq Req, DataSources ADataSources);
```

Description

Use the address of ConstraintCallBack as a parameter when making direct API calls to Borland (see page 1761) Database Engine (BDE) functions that require a callback to provide information about the constraints on the dataset or its fields.

See Also

DisableConstraints (see page 9902)

EnableConstraints (see page 9902)

53.1.5.4.20 TBDEDataSet.ConstraintsDisabled Method

Indicates whether constraints are enabled for a dataset.

Pascal

```
function ConstraintsDisabled: Boolean;
```

C++

```
__fastcall Boolean ConstraintsDisabled();
```

Description

Call ConstraintsDisabled to determine if constraints are currently disabled for a dataset. If ConstraintsDisabled returns true, constraints are currently disabled. If false, constraints are currently enabled.

See Also

DisableConstraints (see page 9902)

EnableConstraints (see page 9902)

53.1.5.4.21 TBDEDataSet.CreateBlobStream Method

Returns a TStream object for reading or writing the data in a specified blob field.

Pascal

```
function CreateBlobStream(Field: TField; Mode: TBlobStreamMode): TStream; override;
```

C++

```
virtual __fastcall TStream CreateBlobStream(TField Field, TBlobStreamMode Mode);
```

Description

Call CreateBlobStream to obtain a stream for reading data from or writing data to a binary large object (BLOB) field. The Field parameter must specify a TBlobField component from the Fields property array. The Mode parameter specifies whether the stream will be used for reading, writing, or updating the contents of the field.

Blob streams are created in a specific mode for a specific field on a specific record. Applications should create a new blob stream every time the record in the dataset changes rather than reusing an existing blob stream.

Delphi Examples:

```
{
The following example copies the data in the Notes field of
Table1 or SQLDataSet1 to the Remarks field of ClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Stream1: TBlobStream;
  Stream2: TStream;
```

```
    blobType : TBlobType;
  begin
    blobType := Table1Notes.BlobType;
    Stream1 := TBlobStream.Create(Table1Notes, bmRead);
    try
      CDS2.Edit;
      { here's a different way to create a blob stream }
      Stream2 := CDS2.CreateBlobStream(CDS2.FieldByName('Remarks'), bmReadWrite);
      try
        Stream2.CopyFrom(Stream1, Stream1.Size);
  //    CDS2.Post;
  //    CDS2.Active := True;
      finally
        Stream2.Free;
      end;
    finally
      Stream1.Free;
    end;
  end;
```

See Also

Fields (see page 7023)

TBlobStream (see page 9958)

53.1.5.4.22 TBDEDataSet.CreateExprFilter Method

Pascal

```
function CreateExprFilter(const Expr: string; Options: TFilterOptions; Priority: Integer):
HDBIFilter;
```

C++

```
__fastcall HDBIFilter CreateExprFilter(const AnsiString Expr, TFilterOptions Options, int
Priority);
```

Description

This is CreateExprFilter, a member of class TBDEDataSet.

53.1.5.4.23 TBDEDataSet.CreateFuncFilter Method

Pascal

```
function CreateFuncFilter(FilterFunc: Pointer; Priority: Integer): HDBIFilter;
```

C++

```
__fastcall HDBIFilter CreateFuncFilter(void * FilterFunc, int Priority);
```

Description

This is CreateFuncFilter, a member of class TBDEDataSet.

53.1.5.4.24 TBDEDataSet.CreateHandle Method

Pascal

```
function CreateHandle: HDBICur; virtual;
```

C++

```
virtual __fastcall HDBICur CreateHandle();
```

Description

This is CreateHandle, a member of class TBDEDataSet.

53.1.5.4.25 TBDEDataSet.CreateLookupFilter Method

Pascal

```
function CreateLookupFilter(Fields: TList; const Values: Variant; Options: TLocateOptions;
Priority: Integer): HDBIFilter;
```

C++

```
__fastcall HDBIFilter CreateLookupFilter(TList Fields, const Variant Values, TLocateOptions
Options, int Priority);
```

Description

This is CreateLookupFilter, a member of class TBDEDataSet.

53.1.5.4.26 TBDEDataSet.DataEvent Method

Pascal

```
procedure DataEvent(Event: TDataEvent; Info: Longint); override;
```

C++

```
virtual __fastcall DataEvent(TDataEvent Event, Longint Info);
```

Description

This is DataEvent, a member of class TBDEDataSet.

53.1.5.4.27 TBDEDataSet.DeactivateFilters Method

Pascal

```
procedure DeactivateFilters;
```

C++

```
__fastcall DeactivateFilters();
```

Description

This is DeactivateFilters, a member of class TBDEDataSet.

53.1.5.4.28 TBDEDataSet.DestroyHandle Method

Pascal

```
procedure DestroyHandle; virtual;
```

C++

```
virtual __fastcall DestroyHandle();
```

Description

This is DestroyHandle, a member of class TBDEDataSet.

53.1.5.4.29 TBDEDataSet.DestroyLookupCursor Method

Pascal

```
procedure DestroyLookupCursor; virtual;
```

C++

```
virtual __fastcall DestroyLookupCursor();
```

Description

This is DestroyLookupCursor, a member of class TBDEDataSet.

53.1.5.4.30 TBDEDataSet.DisableConstraints Method

Disables application of server constraints on the dataset.

Pascal

```
procedure DisableConstraints;
```

C++

```
__fastcall DisableConstraints();
```

Description

Call DisableConstraints to prevent, on a temporary basis, application of server constraints replicated to the dataset. For example, when an application iterates through a dataset, it may improve performance to disable constraints temporarily.

Note: Calling DisableConstraints increments a reference count. As long as this reference count is greater than zero, constraints are disabled for the dataset. To prevent accidental disabling of constraints, always group a call to DisableConstraints with a call to EnableConstraints (see page 9902).

See Also

EnableConstraints (see page 9902)

53.1.5.4.31 TBDEDataSet.EnableConstraints Method

Enables application of server constraints against dataset entries.

Pascal

```
procedure EnableConstraints;
```

C++

```
__fastcall EnableConstraints();
```

Description

Call EnableConstraints to turn on application server constraints previously disabled by a call to DisableConstraints (see page 9902).

Note: Calling EnableConstraints decrements a reference count. When this reference count is zero constraints are enabled for the dataset. To prevent accidental disabling of constraints, always follow a call to DisableConstraints (see page 9902) with a matching call to EnableConstraints.

See Also

DisableConstraints (see page 9902)

53.1.5.4.32 TBDEDataSet.FetchAll Method

Retrieves all records from the current cursor position to the end of the file and stores them locally.

Pascal

```
procedure FetchAll;
```

C++

```
__fastcall FetchAll();
```

Description

Call FetchAll to reduce network traffic when using cached updates. FetchAll calls CheckBrowseMode to post any pending changes, and then retrieves all records from the current cursor position to the end of the file, and store them locally.

Ordinarily when cached updates are enabled, a transaction retrieves only as much data as it needs for display purposes. Each new fetch starts a new, read-only transaction. To consolidate transactions and reduce network traffic, an application can call FetchAll to retrieve all data in a single transaction.

Note: Using FetchAll is not always appropriate. For example, when an application accesses a database used by many simultaneous clients and there is a high degree of contention for updating the same records, fetching all records at once may not be advantageous because some fetched records may be changed by other applications. Always weigh the advantages of reduced network traffic against the need for reduced record contention.

See Also

CachedUpdates (see page 9932)

Filter

Filtered

OnFilterRecord (see page 7055)

OnUpdateRecord (see page 9957)

Using the BDE to Cache Updates

53.1.5.4.33 TBDEDataSet.FindRecord Method

Pascal

```
function FindRecord(Restart: Boolean; GoForward: Boolean): Boolean; override;
```

C++

```
virtual __fastcall Boolean FindRecord(Boolean Restart, Boolean GoForward);
```

Description

This is FindRecord, a member of class TBDEDataSet.

53.1.5.4.34 TBDEDataSet.FlushBuffers Method

Posts all changes that have been written to the record buffer.

Pascal

```
procedure FlushBuffers;
```

C++

```
__fastcall FlushBuffers();
```

Description

Call FlushBuffers to cause the dataset to post all pending changes to the database, including any cached updates. Use FlushBuffers instead of CheckBrowseMode if it is important that cached record buffers are posted.

See Also

CheckBrowseMode (see page 6918)

53.1.5.4.35 TBDEDataSet.ForceUpdateCallback Method

Pascal

```
function ForceUpdateCallback: Boolean;
```

C++

```
__fastcall Boolean ForceUpdateCallback();
```

Description

This is ForceUpdateCallback, a member of class TBDEDataSet.

53.1.5.4.36 TBDEDataSet.FreeKeyBuffers Method

Pascal

```
procedure FreeKeyBuffers;
```

C++

```
__fastcall FreeKeyBuffers();
```

Description

This is FreeKeyBuffers, a member of class TBDEDataSet.

53.1.5.4.37 TBDEDataSet.FreeRecordBuffer Method

Pascal

```
procedure FreeRecordBuffer(var Buffer: PChar); override;
```

C++

```
virtual __fastcall FreeRecordBuffer(const char * Buffer);
```

Description

This is FreeRecordBuffer, a member of class TBDEDataSet.

53.1.5.4.38 TBDEDataSet.GetBlobFieldData Method

Reads BLOB data into a buffer.

Pascal

```
function GetBlobFieldData(FieldNo: Integer; var Buffer: TBlobByteData): Integer; override;
```

C++

```
virtual __fastcall int GetBlobFieldData(int FieldNo, TBlobByteData Buffer);
```

Description

GetBlobFieldData reads blob data from the field specified by FieldNo into a Buffer. The buffer is a dynamic array of bytes, so that it can grow to accommodate the size of the BLOB data. GetBlobFieldData returns the size of the buffer.

See Also

BlobFieldCount (see page 7007)

GetFieldData (see page 6952)

FieldNo (see page 7166)

53.1.5.4.39 TBDEDataSet.GetBookmarkData Method

Pascal

```
procedure GetBookmarkData(Buffer: PChar; Data: Pointer); override;
```

C++

```
virtual __fastcall GetBookmarkData(const char * Buffer, void * Data);
```

Description

This is GetBookmarkData, a member of class TBDEDataSet.

53.1.5.4.40 TBDEDataSet.GetBookmarkFlag Method

Pascal

```
function GetBookmarkFlag(Buffer: PChar): TBookmarkFlag; override;
```

C++

```
virtual __fastcall TBookmarkFlag GetBookmarkFlag(const char * Buffer);
```

Description

This is GetBookmarkFlag, a member of class TBDEDataSet.

53.1.5.4.41 TBDEDataSet.GetCanModify Method

Pascal

```
function GetCanModify: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetCanModify();
```

Description

This is GetCanModify, a member of class TBDEDataSet.

53.1.5.4.42 TBDEDataSet.GetCurrentRecord Method

Retrieves the current record into a buffer.

Pascal

```
function GetCurrentRecord(Buffer: PChar): Boolean; override;
```

C++

```
virtual __fastcall Boolean GetCurrentRecord(const char * Buffer);
```

Description

Most applications should not need to call GetCurrentRecord. TDataSet automatically allocates a buffer for the active record.

Call GetCurrentRecord to copy the current record into a buffer allocated by the application. Buffer must be at least as big as the record size indicated by the RecordSize property.

See Also

RecordSize

ActiveBuffer (see page 6909)

53.1.5.4.43 GetFieldData Method

This is the overview for the GetFieldData method overload.

Overload List

	Name	Description
	TBDEDataSet.GetFieldData (Integer, Pointer) (see page 9906)	Retrieves the current value of a field into a buffer.
	TBDEDataSet.GetFieldData (TField, Pointer) (see page 9906)	Retrieves the current value of a field into a buffer.

53.1.5.4.43.1 TBDEDataSet.GetFieldData Method (Integer, Pointer)

Retrieves the current value of a field into a buffer.

Pascal

```
function GetFieldData(Field: TField; Buffer: Pointer): Boolean; override; overload;
function GetFieldData(FieldNo: Integer; Buffer: Pointer): Boolean; override; overload;
```

C++

```
virtual __fastcall Boolean GetFieldData(TField Field, void * Buffer);
virtual __fastcall Boolean GetFieldData(int FieldNo, void * Buffer);
```

Description

Most applications do not need to call GetFieldData. TField objects call this method to implement their GetData method.

The Field or FieldNo parameter indicates the field whose data should be fetched. Field specifies the component itself, while FieldNo indicates its field number. The Buffer parameter is a memory buffer with sufficient space to accept the value of the field as it exists in the database (unformatted and untranslated). NativeFormat indicates whether the dataset fetches the field in the library's native format for the field type. When NativeFormat is false, the dataset must convert the field value to the native type. This allows the field to handle data from different types of datasets (ADO-based, BDE-based, and so on) in a uniform manner.

GetFieldData returns a value that indicates whether the data was successfully fetched.

GetFieldData returns true if the buffer is successfully filled with the field's data, and false if the data could not be fetched.

See Also

GetBlobFieldData (see page 9904)

FieldNo (see page 7166)

53.1.5.4.44 TBDEDataSet.GetFieldFullName Method

Pascal

```
function GetFieldFullName(Field: TField): string; override;
```

C++

```
virtual __fastcall AnsiString GetFieldFullName(TField Field);
```

Description

This is GetFieldFullName, a member of class TBDEDataSet.

53.1.5.4.45 TBDEDataSet.GetHandle Method

Pascal

```
function GetHandle: HDBICur;
```

C++

```
__fastcall HDBICur GetHandle();
```

Description

This is GetHandle, a member of class TBDEDataSet.

53.1.5.4.46 TBDEDataSet.GetIndexField Method

Pascal

```
function GetIndexField(Index: Integer): TField;
```

C++

```
__fastcall TField GetIndexField(int Index);
```

Description

This is GetIndexField, a member of class TBDEDataSet.

53.1.5.4.47 TBDEDataSet.GetIndexFieldCount Method

Pascal

```
function GetIndexFieldCount: Integer;
```

C++

```
__fastcall int GetIndexFieldCount();
```

Description

This is GetIndexFieldCount, a member of class TBDEDataSet.

53.1.5.4.48 TBDEDataSet.GetIndexInfo Method

Retrieves information about the current index into the index data fields of the dataset.

Pascal

```
procedure GetIndexInfo;
```

C++

```
__fastcall GetIndexInfo();
```

Description

Call GetIndexInfo to update information about the current index. Ordinarily an application does not need to call GetIndexInfo because this routine is used internally to retrieve index information as needed. Some applications, however, may want to ensure that the index information used by the dataset is up-to-date.

GetIndexInfo queries the BDE for index information, including:

Whether the current index is a dBASE expression index.

Whether the index is case insensitive.

The number of fields that make up the key for the index.

The field map for the index.

The size of the index key.

Number of fields and field map are not retrieved if the index is a dBASE expression index.

See Also

ExpIndex (see page 9933)

53.1.5.4.49 TBDEDataSet.GetIsIndexField Method

Pascal

```
function GetIsIndexField(Field: TField): Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsIndexField(TField Field);
```

Description

This is GetIsIndexField, a member of class TBDEDataSet.

53.1.5.4.50 TBDEDataSet.GetKeyBuffer Method

Pascal

```
function GetKeyBuffer(KeyIndex: TKeyIndex): PKeyBuffer;
```

C++

```
__fastcall PKeyBuffer GetKeyBuffer(TKeyIndex KeyIndex);
```

Description

This is GetKeyBuffer, a member of class TBDEDataSet.

53.1.5.4.51 TBDEDataSet.GetKeyExclusive Method

Pascal

```
function GetKeyExclusive: Boolean;
```

C++

```
__fastcall Boolean GetKeyExclusive();
```

Description

This is GetKeyExclusive, a member of class TBDEDataSet.

53.1.5.4.52 TBDEDataSet.GetKeyFieldCount Method

Pascal

```
function GetKeyFieldCount: Integer;
```

C++

```
__fastcall int GetKeyFieldCount();
```

Description

This is GetKeyFieldCount, a member of class TBDEDataSet.

53.1.5.4.53 TBDEDataSet.GetLookupCursor Method

Pascal

```
function GetLookupCursor(const KeyFields: string; CaseInsensitive: Boolean): HDBICur; virtual;
```

C++

```
virtual __fastcall HDBICur GetLookupCursor(const AnsiString KeyFields, Boolean
CaseInsensitive);
```

Description

This is GetLookupCursor, a member of class TBDEDataSet.

53.1.5.4.54 TBDEDataSet.GetObjectTypeNames Method

Pascal

```
procedure GetObjectTypeNames(Fields: TFields);
```

C++

```
__fastcall GetObjectTypeNames(TFields Fields);
```

Description

This is GetObjectTypeNames, a member of class TBDEDataSet.

53.1.5.4.55 TBDEDataSet.GetRecNo Method

Pascal

```
function GetRecNo: Integer; override;
```

C++

```
virtual __fastcall int GetRecNo();
```

Description

This is GetRecNo, a member of class TBDEDataSet.

53.1.5.4.56 TBDEDataSet.GetRecord Method

Pascal

```
function GetRecord(Buffer: PChar; GetMode: TGetMode; DoCheck: Boolean): TGetResult; override;
```

C++

```
virtual __fastcall TGetResult GetRecord(const char * Buffer, TGetMode GetMode, Boolean
DoCheck);
```

Description

This is GetRecord, a member of class TBDEDataSet.

53.1.5.4.57 TBDEDataSet.GetRecordCount Method

Pascal

```
function GetRecordCount: Integer; override;
```

C++

```
virtual __fastcall int GetRecordCount();
```

Description

This is GetRecordCount, a member of class TBDEDataSet.

53.1.5.4.58 TBDEDataSet.GetRecordSize Method

Pascal

```
function GetRecordSize: Word; override;
```

C++

```
virtual __fastcall Word GetRecordSize();
```

Description

This is GetRecordSize, a member of class TBDEDataSet.

53.1.5.4.59 TBDEDataSet.GetStateFieldValue Method

Pascal

```
function GetStateFieldValue(State: TDataSetState; Field: TField): Variant; override;
```

C++

```
virtual __fastcall Variant GetStateFieldValue(TDataSetState State, TField Field);
```

Description

This is GetStateFieldValue, a member of class TBDEDataSet.

53.1.5.4.60 TBDEDataSet.GetUpdateRecordSet Method

Pascal

```
function GetUpdateRecordSet: TUpdateRecordTypes;
```

C++

```
__fastcall TUpdateRecordTypes GetUpdateRecordSet();
```

Description

This is GetUpdateRecordSet, a member of class TBDEDataSet.

53.1.5.4.61 TBDEDataSet.GetUpdatesPending Method

Pascal

```
function GetUpdatesPending: Boolean;
```

C++

```
__fastcall Boolean GetUpdatesPending();
```

Description

This is GetUpdatesPending, a member of class TBDEDataSet.

53.1.5.4.62 TBDEDataSet.InitKeyBuffer Method

Pascal

```
function InitKeyBuffer(Buffer: PKeyBuffer): PKeyBuffer;
```

C++

```
__fastcall PKeyBuffer InitKeyBuffer(PKeyBuffer Buffer);
```

Description

This is InitKeyBuffer, a member of class TBDEDataSet.

53.1.5.4.63 TBDEDataSet.InitRecord Method

Pascal

```
procedure InitRecord(Buffer: PChar); override;
```

C++

```
virtual __fastcall InitRecord(const char * Buffer);
```

Description

This is InitRecord, a member of class TBDEDataSet.

53.1.5.4.64 TBDEDataSet.InternalAddRecord Method

Pascal

```
procedure InternalAddRecord(Buffer: Pointer; Append: Boolean); override;
```

C++

```
virtual __fastcall InternalAddRecord(void * Buffer, Boolean Append);
```

Description

This is InternalAddRecord, a member of class TBDEDataSet.

53.1.5.4.65 TBDEDataSet.InternalCancel Method

Pascal

```
procedure InternalCancel; override;
```

C++

```
virtual __fastcall InternalCancel();
```

Description

This is InternalCancel, a member of class TBDEDataSet.

53.1.5.4.66 TBDEDataSet.InternalClose Method

Pascal

```
procedure InternalClose; override;
```

C++

```
virtual __fastcall InternalClose();
```

Description

This is InternalClose, a member of class TBDEDataSet.

53.1.5.4.67 TBDEDataSet.InternalDelete Method

Pascal

```
procedure InternalDelete; override;
```

C++

```
virtual __fastcall InternalDelete();
```

Description

This is InternalDelete, a member of class TBDEDataSet.

53.1.5.4.68 TBDEDataSet.InternalEdit Method

Pascal

```
procedure InternalEdit; override;
```

C++

```
virtual __fastcall InternalEdit();
```

Description

This is InternalEdit, a member of class TBDEDataSet.

53.1.5.4.69 TBDEDataSet.InternalFirst Method

Pascal

```
procedure InternalFirst; override;
```

C++

```
virtual __fastcall InternalFirst();
```

Description

This is InternalFirst, a member of class TBDEDataSet.

53.1.5.4.70 TBDEDataSet.InternalGotoBookmark Method

Pascal

```
procedure InternalGotoBookmark(Bookmark: TBookmark); override;
```

C++

```
virtual __fastcall InternalGotoBookmark(TBookmark Bookmark);
```

Description

This is InternalGotoBookmark, a member of class TBDEDataSet.

53.1.5.4.71 TBDEDataSet.InternalHandleException Method

Pascal

```
procedure InternalHandleException; override;
```

C++

```
virtual __fastcall InternalHandleException();
```

Description

This is InternalHandleException, a member of class TBDEDataSet.

53.1.5.4.72 TBDEDataSet.InternalInitFieldDefs Method

Pascal

```
procedure InternalInitFieldDefs; override;
```

C++

```
virtual __fastcall InternalInitFieldDefs();
```

Description

This is InternalInitFieldDefs, a member of class TBDEDataSet.

53.1.5.4.73 TBDEDataSet.InternalInitRecord Method

Pascal

```
procedure InternalInitRecord(Buffer: PChar); override;
```

C++

```
virtual __fastcall InternalInitRecord(const char * Buffer);
```

Description

This is InternalInitRecord, a member of class TBDEDataSet.

53.1.5.4.74 TBDEDataSet.InternalInsert Method

Pascal

```
procedure InternalInsert; override;
```

C++

```
virtual __fastcall InternalInsert();
```

Description

This is InternalInsert, a member of class TBDEDataSet.

53.1.5.4.75 TBDEDataSet.InternalLast Method

Pascal

```
procedure InternalLast; override;
```

C++

```
virtual __fastcall InternalLast();
```

Description

This is InternalLast, a member of class TBDEDataSet.

53.1.5.4.76 TBDEDataSet.InternalOpen Method

Pascal

```
procedure InternalOpen; override;
```

C++

```
virtual __fastcall InternalOpen();
```

Description

This is InternalOpen, a member of class TBDEDataSet.

53.1.5.4.77 TBDEDataSet.InternalPost Method

Pascal

```
procedure InternalPost; override;
```

C++

```
virtual __fastcall InternalPost();
```

Description

This is InternalPost, a member of class TBDEDataSet.

53.1.5.4.78 TBDEDataSet.InternalRefresh Method

Pascal

```
procedure InternalRefresh; override;
```

C++

```
virtual __fastcall InternalRefresh();
```

Description

This is InternalRefresh, a member of class TBDEDataSet.

53.1.5.4.79 TBDEDataSet.InternalSetToRecord Method

Pascal

```
procedure InternalSetToRecord(Buffer: PChar); override;
```

C++

```
virtual __fastcall InternalSetToRecord(const char * Buffer);
```

Description

This is InternalSetToRecord, a member of class TBDEDataSet.

53.1.5.4.80 TBDEDataSet.IsCursorOpen Method

Pascal

```
function IsCursorOpen: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCursorOpen();
```

Description

This is IsCursorOpen, a member of class TBDEDataSet.

53.1.5.4.81 TBDEDataSet.IsSequenced Method

IsSequenced indicates whether sequence numbers are available for database records.

Pascal

```
function IsSequenced: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsSequenced();
```

Description

Call IsSequenced to determine whether database records can be located by sequence numbers. When IsSequenced is true, the dataset can navigate directly to a specific record by setting the RecNo property. If IsSequenced is false, the only way to navigate to a specific record is to start at the beginning and count records.

See Also

RecNo

53.1.5.4.82 TBDEDataSet.Locate Method

Searches the dataset for a specified record and makes that record the current record.

Pascal

```
function Locate(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions):
Boolean; override;
```

C++

```
virtual __fastcall Boolean Locate(const AnsiString KeyFields, const Variant KeyValues,
TLocateOptions Options);
```

Description

Call Locate to search a dataset for a specific record and position the cursor on it.

KeyFields is a string containing a semicolon-delimited list of field names on which to search.

KeyValues is a variant array containing the values to match in the key fields. If KeyFields lists a single field, KeyValues specifies the value for that field on the desired record. To specify multiple search values, pass a variant array as KeyValues, or construct a variant array on the fly using the VarArrayOf routine. For example:

```
with CustTable do
  Locate('Company;Contact;Phone', VarArrayOf(['Sight Diver', 'P', '408-431-1000']),
[loPartialKey]);
TLocateOptions Opts;
Opts.Clear();
Opts << loPartialKey;
Variant locvalues[2];
locvalues[0] = Variant("Sight Diver");
locvalues[1] = Variant("P");
CustTable->Locate("Company;Contact", VarArrayOf(locvalues, 1), Opts);
```

Options is a set that optionally specifies additional search latitude when searching on string fields. If Options contains the loCaseInsensitive setting, then Locate ignores case when matching fields. If Options contains the loPartialKey setting, then Locate allows partial-string matching on strings in KeyValues. If Options is an empty set, or if the KeyFields property does not include any string fields, Options is ignored.

Locate returns true if it finds a matching record, and makes that record the current one. Otherwise Locate returns false.

Locate uses the fastest possible method to locate matching records. If the search fields in KeyFields are indexed and the index is compatible with the specified search options, Locate uses the index. Otherwise Locate creates a filter for the search.

See Also

FindField (see page 6938)

Lookup (see page 9916)

53.1.5.4.83 TBDEDataSet.LocateRecord Method

Pascal

```
function LocateRecord(const KeyFields: string; const KeyValues: Variant; Options:
TLocateOptions; SyncCursor: Boolean): Boolean;
```

C++

```
__fastcall Boolean LocateRecord(const AnsiString KeyFields, const Variant KeyValues,
TLocateOptions Options, Boolean SyncCursor);
```

Description

This is LocateRecord, a member of class TBDEDataSet.

53.1.5.4.84 TBDEDataSet.Lookup Method

Retrieves field values from a record that matches specified search values.

Pascal

```
function Lookup(const KeyFields: string; const KeyValues: Variant; const ResultFields:
```

```
string): Variant; override;
```

C++

```
virtual __fastcall Variant Lookup(const AnsiString KeyFields, const Variant KeyValues, const
AnsiString ResultFields);
```

Description

Call Lookup to search for a record in the dataset in which specified fields contain specified values and return other field values from the found record. Unlike other dataset search methods, Lookup performs its search without moving the record pointer of the dataset. This is useful when doing data validation on a record's data while the record is still in edit or insert mode.

KeyFields is a string containing the names of the field or fields on which the search is predicated. If the search is based on more than one field, KeyFields is a list of field names delimited with semicolons.

KeyValues is a variant array containing the values to find in the fields specified in KeyFields. To specify multiple search values, pass KeyValues as a variant array as an argument, or construct a variant array on the fly using the VarArrayOf routine. These search values must be specified in the order of the fields in KeyFields to which each corresponds. For instance, the first value is used for the first field in KeyFields, the second value for the second field, and so on.

ResultFields is a string containing the names of the field or fields from which to return values. If values are to be returned from more than one field, ResultFields is a list of field names delimited with semicolons.

Lookup returns a variant array containing the values from the fields specified in ResultFields.

Lookup uses the fastest possible method to locate matching records. If the search fields in KeyFields are indexed, Lookup uses the index. Otherwise Lookup creates a filter for the search.

The example below uses the Lookup method to search the sample Paradox table Customer. The search is based on two fields: Company and State. The search is also based on the following two values in these two fields: "Blue Sports" and "OR". For the search to be successful, there must be a record where the Company field contains the value "Blue Sports" AND the field State contains "OR". If either criteria is not met, the search is unsuccessful (such a record does exist in this table, so the search will be successful). The values of the CustNo and Addr1 fields are returned by the execution of the Lookup method. If the search is unsuccessful, the VarType function applied to the returned variant array returns varNull.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  V: Variant;
  C: Integer;
  A: String;
begin
  V := Table1.Lookup('Company;State',
    VarArrayOf(['Blue Sports', 'OR']),
    'CustNo;Addr1');
if not (VarType(V) in [varNull]) then begin
    C := V[0];
    A := V[1];
    ShowMessage(IntToStr(C) + #10 + A);
end
else
    ShowMessage('Search unsuccessful!');
end;
int C;
AnsiString A;
Variant V;
V = Table1->Lookup("Company;State",
  VarArrayOf(OPENARRAY(Variant, ("Blue Sports","OR"))),
  "CustNo;Addr1");
if !(VarType(V) == varNull)
{
  C = V[0];
  A = V[1];
  ShowMessage(IntToStr(C) + #10 + A);
```

```
}
else
  ShowMessage("Search unsuccessful!");
```

See Also

Locate (see page 9915)

VarArrayOf (see page 23648)

VarArraycreate

53.1.5.4.85 TBDEDataSet.MapsToIndex Method

Pascal

```
function MapsToIndex(Fields: TList; CaseInsensitive: Boolean): Boolean;
```

C++

```
__fastcall Boolean MapsToIndex(TList Fields, Boolean CaseInsensitive);
```

Description

This is MapsToIndex, a member of class TBDEDataSet.

53.1.5.4.86 TBDEDataSet.OpenCursor Method

Pascal

```
procedure OpenCursor(InfoQuery: Boolean); override;
```

C++

```
virtual __fastcall OpenCursor(Boolean InfoQuery);
```

Description

This is OpenCursor, a member of class TBDEDataSet.

53.1.5.4.87 TBDEDataSet.Post Method

Writes a modified record to the database.

Pascal

```
procedure Post; override;
```

C++

```
virtual __fastcall Post();
```

Description

Call Post to write a modified record to the database. Dataset methods that change the dataset state, such as Edit, Insert, or Append, or that move from one record to another, such as First, Last, Next, and Prior automatically call Post.

Delphi Examples:

```
{
This example appends a new record to a table or client
dataset when the user clicks a button. The two fields
Field1 and Field2 are filled from the contents of two edit
controls.  This example requires a populated TTable or
TDataSet with the appropriate field values.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  Customers.Append;
  Customers.FieldValues['Field1'] := StrToInt(Edit1.text);
  Customers.FieldValues['Field2'] := Edit2.text;
  Customers.Post;
end;
```

See Also

State (see page 7036)

AfterPost (see page 7046)

BeforePost (see page 7052)

Append (see page 6910)

Edit (see page 6932)

First (see page 6942)

Insert (see page 6959)

Last (see page 6968)

Next (see page 6973)

Prior (see page 6979)

Resync (see page 6994)

UpdateRecord (see page 7002)

53.1.5.4.88 TBDEDataSet.PostKeyBuffer Method

Pascal

```
procedure PostKeyBuffer(Commit: Boolean);
```

C++

```
__fastcall PostKeyBuffer(Boolean Commit);
```

Description

This is PostKeyBuffer, a member of class TBDEDataSet.

53.1.5.4.89 TBDEDataSet.PrepareCursor Method

Pascal

```
procedure PrepareCursor; virtual;
```

C++

```
virtual __fastcall PrepareCursor();
```

Description

This is PrepareCursor, a member of class TBDEDataSet.

53.1.5.4.90 TBDEDataSet.ProcessUpdates Method

Pascal

```
function ProcessUpdates(UpdCmd: DBIDelayedUpdCmd): DBIResult;
```

C++

```
__fastcall DBIResult ProcessUpdates(DBIDelayedUpdCmd UpdCmd);
```

Description

This is ProcessUpdates, a member of class TBDEDataSet.

53.1.5.4.91 TBDEDataSet.PSGetUpdateException Method

Generates an EUpdateError object based on another exception object.

Pascal

```
function PSGetUpdateException(E: Exception; Prev: EUpdateError): EUpdateError; override;
```

C++

```
virtual __fastcall EUpdateError PSGetUpdateException(Exception E, EUpdateError Prev);
```

Description

The provider component calls PSGetUpdateException when the exception E occurs while applying updates. This method allows the provider to create a single EUpdateError object that encapsulates all the exceptions that occur in an update operation.

PSGetUpdateError determines the error message, context string, and error code from the exception specified by E and creates a new EUpdateError exception that includes that information as well as the information about a previous error as specified by Prev. The context string and error code for E may be specific to the Exception class associated with the dataset. By default, TDataSet generates an EUpdateError with an error code of 1 and an empty context string.

See Also

Create (see page 6733)

53.1.5.4.92 TBDEDataSet.PSIsSQLSupported Method

Indicates whether the dataset can execute SQL statements.

Pascal

```
function PSIsSQLSupported: Boolean; override;
```

C++

```
virtual __fastcall Boolean PSIsSQLSupported();
```

Description

PSIsSQLSupported indicates whether the provider can call PSExecuteStatement to execute an SQL statement. When PSIsSQLSupported returns false, calling PSExecuteStatement generates an exception. The provider can still work with a dataset that returns false from PSIsSQLSupported, but it must set the ResolveToDataSet property to true, or provide data on a read-only basis.

See Also

PSIsSQLBased (see page 7584)

PSExecuteStatement (see page 7579)

ResolveToDataSet (see page 18531)

53.1.5.4.93 TBDEDataSet.PSReset Method

Resets the dataset so that it is positioned on the first record.

Pascal

```
procedure PSReset; override;
```

C++

```
virtual __fastcall PSReset();
```

Description

The provider calls PSReset to reposition the dataset on the first record before fetching records.

See Also

GetRecords (see page 18468)

AS_GetRecords (see page 16455)

53.1.5.4.94 TBDEDataSet.PSUpdateRecord Method

Applies a single update to the underlying database table or tables.

Pascal

```
function PSUpdateRecord(UpdateKind: TUpdateKind; Delta: TDataSet): Boolean; override;
```

C++

```
virtual __fastcall Boolean PSUpdateRecord(TUpdateKind UpdateKind, TDataSet Delta);
```

Description

Before the provider generates SQL statements to apply an update, it calls PSUpdateRecord to let the dataset apply the update in some other fashion. For example, BDE-enabled datasets generate an OnUpdateRecord to allow the application to apply the update and, if the OnUpdateRecord event handler does not apply the update, uses any associated TUpdateSQL object to apply the update.

UpdateKind indicates whether the update is an insertion (ukInsert), deletion (ukDelete), or modification (ukModify).

Delta is the delta datapacket. The current record represents the record to be updated.

PSUpdateRecord returns true if it updates the record, false otherwise. If PSUpdateRecord returns false, the provider generates dynamic SQL to perform the update and calls PSExecuteStatement.

Note: PSUpdateRecord is only called if the provider's ResolveToDataSet property is false.

See Also

OnUpdateRecord (see page 9957)

TUpdateSQL (see page 10229)

applyupdates (see page 18459)

ResolveToDataSet (see page 18531)

PSExecuteStatement (see page 7579)

53.1.5.4.95 TBDEDataSet.ResetCursorRange Method

Pascal

```
function ResetCursorRange: Boolean;
```

C++

```
__fastcall Boolean ResetCursorRange();
```

Description

This is ResetCursorRange, a member of class TBDEDataSet.

53.1.5.4.96 TBDEDataSet.RevertRecord Method

Restores the current record in the dataset to an unmodified state when cached updates are enabled.

Pascal

```
procedure RevertRecord;
```

C++

```
__fastcall RevertRecord();
```

Description

Call RevertRecord to undo changes made to the current record when cached updates are enabled.

Note: To undo all changes to all pending updates in the cache, call CancelUpdates (see page 9896).

See Also

CancelUpdates (see page 9896)

Using the BDE to Cache Updates

53.1.5.4.97 TBDEDataSet.SetBlockReadSize Method

Pascal

```
procedure SetBlockReadSize(Value: Integer); override;
```

C++

```
virtual __fastcall SetBlockReadSize(int Value);
```

Description

This is SetBlockReadSize, a member of class TBDEDataSet.

53.1.5.4.98 TBDEDataSet.SetBookmarkData Method

Pascal

```
procedure SetBookmarkData(Buffer: PChar; Data: Pointer); override;
```

C++

```
virtual __fastcall SetBookmarkData(const char * Buffer, void * Data);
```

Description

This is SetBookmarkData, a member of class TBDEDataSet.

53.1.5.4.99 TBDEDataSet.SetBookmarkFlag Method

Pascal

```
procedure SetBookmarkFlag(Buffer: PChar; Value: TBookmarkFlag); override;
```

C++

```
virtual __fastcall SetBookmarkFlag(const char * Buffer, TBookmarkFlag Value);
```

Description

This is SetBookmarkFlag, a member of class TBDEDataSet.

53.1.5.4.100 TBDEDataSet.SetCachedUpdates Method

Pascal

```
procedure SetCachedUpdates(Value: Boolean);
```

C++

```
__fastcall SetCachedUpdates(Boolean Value);
```

Description

This is SetCachedUpdates, a member of class TBDEDataSet.

53.1.5.4.101 TBDEDataSet.SetCursorRange Method

Pascal

```
function SetCursorRange: Boolean;
```

C++

```
__fastcall Boolean SetCursorRange();
```

Description

This is SetCursorRange, a member of class TBDEDataSet.

53.1.5.4.102 TBDEDataSet.SetFieldData Method

Pascal

```
procedure SetFieldData(Field: TField; Buffer: Pointer); override;
```

C++

```
virtual __fastcall SetFieldData(TField Field, void * Buffer);
```

Description

This is SetFieldData, a member of class TBDEDataSet.

53.1.5.4.103 TBDEDataSet.SetFilterData Method

Pascal

```
procedure SetFilterData(const Text: string; Options: TFilterOptions);
```

C++

```
__fastcall SetFilterData(const AnsiString Text, TFilterOptions Options);
```

Description

This is SetFilterData, a member of class TBDEDataSet.

53.1.5.4.104 TBDEDataSet.SetFiltered Method

Pascal

```
procedure SetFiltered(Value: Boolean); override;
```

C++

```
virtual __fastcall SetFiltered(Boolean Value);
```

Description

This is SetFiltered, a member of class TBDEDataSet.

53.1.5.4.105 TBDEDataSet.SetFilterHandle Method

Pascal

```
procedure SetFilterHandle(var Filter: HDBIFilter; Value: HDBIFilter);
```

C++

```
__fastcall SetFilterHandle(HDBIFilter Filter, HDBIFilter Value);
```

Description

This is SetFilterHandle, a member of class TBDEDataSet.

53.1.5.4.106 TBDEDataSet.SetFilterOptions Method

Pascal

```
procedure SetFilterOptions(Value: TFilterOptions); override;
```

C++

```
virtual __fastcall SetFilterOptions(TFilterOptions Value);
```

Description

This is SetFilterOptions, a member of class TBDEDataSet.

53.1.5.4.107 TBDEDataSet.SetFilterText Method

Pascal

```
procedure SetFilterText(const Value: string); override;
```

C++

```
virtual __fastcall SetFilterText(const AnsiString Value);
```

Description

This is SetFilterText, a member of class TBDEDataSet.

53.1.5.4.108 TBDEDataSet.SetIndexField Method

Pascal

```
procedure SetIndexField(Index: Integer; Value: TField);
```

C++

```
__fastcall SetIndexField(int Index, TField Value);
```

Description

This is SetIndexField, a member of class TBDEDataSet.

53.1.5.4.109 TBDEDataSet.SetKeyBuffer Method

Pascal

```
procedure SetKeyBuffer(KeyIndex: TKeyIndex; Clear: Boolean);
```

C++

```
__fastcall SetKeyBuffer(TKeyIndex KeyIndex, Boolean Clear);
```

Description

This is SetKeyBuffer, a member of class TBDEDataSet.

53.1.5.4.110 TBDEDataSet.SetKeyExclusive Method

Pascal

```
procedure SetKeyExclusive(Value: Boolean);
```

C++

```
__fastcall SetKeyExclusive(Boolean Value);
```

Description

This is SetKeyExclusive, a member of class TBDEDataSet.

53.1.5.4.111 TBDEDataSet.SetKeyFieldCount Method

Pascal

```
procedure SetKeyFieldCount(Value: Integer);
```

C++

```
__fastcall SetKeyFieldCount(int Value);
```

Description

This is SetKeyFieldCount, a member of class TBDEDataSet.

53.1.5.4.112 TBDEDataSet.SetKeyFields Method

Pascal

```
procedure SetKeyFields(KeyIndex: TKeyIndex; const Values: array of const);
```

C++

```
__fastcall SetKeyFields(TKeyIndex KeyIndex, const array of const Values);
```

Description

This is SetKeyFields, a member of class TBDEDataSet.

53.1.5.4.113 TBDEDataSet.SetLinkRanges Method

Pascal

```
procedure SetLinkRanges(MasterFields: TList);
```

C++

```
__fastcall SetLinkRanges(TList MasterFields);
```

Description

This is SetLinkRanges, a member of class TBDEDataSet.

53.1.5.4.114 TBDEDataSet.SetLocale Method

Pascal

```
procedure SetLocale(Value: TLocale);
```

C++

```
__fastcall SetLocale(TLocale Value);
```

Description

This is SetLocale, a member of class TBDEDataSet.

53.1.5.4.115 TBDEDataSet.SetOnFilterRecord Method

Pascal

```
procedure SetOnFilterRecord(const Value: TFilterRecordEvent); override;
```

C++

```
virtual __fastcall SetOnFilterRecord(const TFilterRecordEvent Value);
```

Description

This is SetOnFilterRecord, a member of class TBDEDataSet.

53.1.5.4.116 TBDEDataSet.SetOnUpdateError Method

Pascal

```
procedure SetOnUpdateError(UpdateEvent: TUpdateErrorEvent);
```

C++

```
__fastcall SetOnUpdateError(TUpdateErrorEvent UpdateEvent);
```

Description

This is SetOnUpdateError, a member of class TBDEDataSet.

53.1.5.4.117 TBDEDataSet.SetRecNo Method

Pascal

```
procedure SetRecNo(Value: Integer); override;
```

C++

```
virtual __fastcall SetRecNo(int Value);
```

Description

This is SetRecNo, a member of class TBDEDataSet.

53.1.5.4.118 TBDEDataSet.SetStateFieldValue Method

Pascal

```
procedure SetStateFieldValue(State: TDataSetState; Field: TField; const Value: Variant);
override;
```

C++

```
virtual __fastcall SetStateFieldValue(TDataSetState State, TField Field, const Variant Value);
```

Description

This is SetStateFieldValue, a member of class TBDEDataSet.

53.1.5.4.119 TBDEDataSet.SetupCallBack Method

Pascal

```
procedure SetupCallBack(Value: Boolean);
```

C++

```
__fastcall SetupCallBack(Boolean Value);
```

Description

This is SetupCallBack, a member of class TBDEDataSet.

53.1.5.4.120 TBDEDataSet.SetUpdateObject Method

Pascal

```
procedure SetUpdateObject(Value: TDataSetUpdateObject);
```

C++

```
__fastcall SetUpdateObject(TDataSetUpdateObject Value);
```

Description

This is SetUpdateObject, a member of class TBDEDataSet.

53.1.5.4.121 TBDEDataSet.SetUpdateRecordSet Method

Pascal

```
procedure SetUpdateRecordSet(RecordTypes: TUpdateRecordTypes);
```

C++

```
__fastcall SetUpdateRecordSet(TUpdateRecordTypes RecordTypes);
```

Description

This is SetUpdateRecordSet, a member of class TBDEDataSet.

53.1.5.4.122 TBDEDataSet.SwitchToIndex Method

Pascal

```
procedure SwitchToIndex(const IndexName: string; const TagName: string);
```

C++

```
__fastcall SwitchToIndex(const AnsiString IndexName, const AnsiString TagName);
```

Description

This is SwitchToIndex, a member of class TBDEDataSet.

53.1.5.4.123 TBDEDataSet.Translate Method

Converts a data string between the Microsoft ANSI character set, and the local code page (OEM character set).

Pascal

```
function Translate(Src: PChar; Dest: PChar; ToOem: Boolean): Integer; override;
```

C++

```
virtual __fastcall int Translate(const char * Src, const char * Dest, Boolean ToOem);
```

Description

When the ToOem parameter is true, Translate converts the source string from the ANSI character set to the OEM character set. If ToOem is false, Translate converts the source string from the OEM character set to the Microsoft ANSI character set. By default, BDE datasets work with strings in the OEM character set.

53.1.5.4.124 TBDEDataSet.UpdateCallbackRequired Method

Pascal

```
function UpdateCallbackRequired: Boolean;
```

C++

```
__fastcall Boolean UpdateCallbackRequired();
```

Description

This is UpdateCallbackRequired, a member of class TBDEDataSet.

53.1.5.4.125 TBDEDataSet.UpdateStatus Method

Reports the update status for the current record.

Pascal

```
function UpdateStatus: TUpdateStatus; override;
```

C++

```
virtual __fastcall TUpdateStatus UpdateStatus();
```

Description

Call UpdateStatus to determine the update status for the current record in a dataset when cached updates are enabled. Update status can change frequently as records are edited, inserted, or deleted. UpdateStatus offers a convenient method for applications to assess the current status before undertaking or completing operations that depend on the update status of individual records in the dataset.

See Also

CachedUpdates (see page 9932)

ApplyUpdates (see page 9894)

CancelUpdates (see page 9896)

CommitUpdates (see page 9897)

RevertRecord (see page 9922)

Using the BDE to Cache Updates

53.1.5.5 TBDEDataSet Properties

The properties of the TBDEDataSet class are listed here.

Protected Properties

	Name	Description
	StmtHandle (see page 9935)	Specifies the Borland (see page 1761) Database Engine (BDE) statement handle for the dataset.

Public Properties

	Name	Description
	CacheBlobs (see page 9932)	Determines whether BLOB fields are cached in memory.
R	ExpIndex (see page 9933)	Indicates whether a dataset is using a dBASE expression index.
R	Handle (see page 9933)	Specifies the Borland (see page 1761) Database Engine (BDE) cursor handle for the dataset.
R	KeySize (see page 9934)	Specifies the size of the key for the current index of the dataset.
R	Locale (see page 9934)	Identifies the Borland (see page 1761) Database Engine (BDE) language driver for the dataset.
	UpdateObject (see page 9936)	Specifies the update object component used to update a read-only result set when cached updates are enabled.
	UpdateRecordTypes (see page 9936)	Specifies the type of records visible in a dataset when cached updates are enabled.
R	UpdatesPending (see page 9937)	Indicates whether the cached updates buffer contains records that are not yet applied.

Published Properties

	Name	Description
	Active (see page 9929)	Specifies whether or not a dataset is open.
	AutoCalcFields (see page 9931)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	CachedUpdates (see page 9932)	Specifies whether cached updates are enabled for a dataset.
	ObjectView (see page 9935)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.

53.1.5.5.1 TBDEDataSet.Active Property

Specifies whether or not a dataset is open.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Use Active to determine or set whether a dataset is populated with data. When Active is false, the dataset is closed; the dataset cannot read or write data and data-aware controls can not use it to fetch data or post edits. When Active is true, the dataset can be populated with data. It can read data from a database or other source (such as a provider). Depending on the CanModify property, active datasets can post changes.

Setting Active to true:

1. Generates a BeforeOpen event.
2. Sets the dataset state to dsBrowse.
3. Establishes a way to fetch data (typically by opening a cursor).
4. Generates an AfterOpen event.

If an error occurs while opening the dataset, dataset state is set to dsInactive, and any cursor is closed.

Setting Active to false:

1. Triggers a BeforeClose event.
2. Sets the State property to dsInactive.
3. Closes the cursor.
4. Triggers an AfterClose event.

An application must set Active to false before changing other properties that affect the status of a database or the controls that display data in an application.

Note: Calling the Open method sets Active to true; calling the Close method sets Active to false.

Delphi Examples:

```
{
This example requires a database with the following name or
in the form of a *.DB file in the following path.  The table
should be closed when changing the database.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Customers.Active := False;
  try
  { First try to use an alias }
    Customers.DatabaseName := 'DBDEMOS';
    Customers.Active := True;
  except
    on EDatabaseError do
    begin
      { If that fails, try to use the drive and directory }
      Customers.TableName := 'Flights';
      Customers.DatabaseName :=
        'c:\Documents and Settings\All Users\Documents\RAD
Studio\5.0\Demos\IntraWeb\Win32\FlightInformation\Data';
      Customers.Active := True;
    end;
  end;
```

```
end;
```

See Also

DataSource (see page 7014)

State (see page 7036)

AfterOpen (see page 7045)

BeforeOpen (see page 7051)

Close (see page 6920)

Open (see page 6976)

CanModify (see page 7012)

BeforeClose (see page 7048)

AfterClose (see page 7040)

Using Datasets with Decision Support Components

Making the Table a Detail of Another Dataset

Opening and Closing Datasets

Fetching the Data

53.1.5.5.2 TBDEDataSet.AutoCalcFields Property

Determines when the OnCalcFields event is triggered and when lookup field values are calculated.

Pascal

```
property AutoCalcFields: Boolean;
```

C++

```
__property Boolean AutoCalcFields;
```

Description

Set AutoCalcFields to control when the OnCalcFields event is triggered to update calculated fields and when lookup fields are calculated.

A calculated field is one that derives its value from the values of one or more fields in the active record, sometimes with additional processing. Lookup fields are fields whose values come from a secondary dataset or lookup cache.

Note: Unidirectional datasets support calculated fields but not unidirectional fields. For unidirectional datasets, OnCalcFields only controls when calculated fields are updated.

When AutoCalcFields is true (the default), Lookup fields are recalculated and OnCalcFields is triggered when:

The dataset is opened.

The dataset is put into dsEdit state.

Focus moves from one visual control to another, or from one column to another in a data-aware grid and modifications have been made to the record.

When AutoCalcFields is false, Lookup fields are recalculated and the OnCalcFields event occurs only when

The dataset is opened.

The dataset is put into dsEdit state.

A record is retrieved from a database.

If an application permits users to change data, OnCalcFields is frequently triggered. In these cases an application may set AutoCalcFields to false to reduce the frequency with which the OnCalcFields event occurs and with which lookup values are fetched.

See Also

OnCalcFields (see page 7053)

Calculating Fields

53.1.5.5.3 TBDEDataSet.CacheBlobs Property

Determines whether BLOB fields are cached in memory.

Pascal

```
property CacheBlobs: Boolean;
```

C++

```
__property Boolean CacheBlobs;
```

Description

Use CacheBlobs to specify whether to store BLOB images in memory to improve performance when scrolling through records that display BLOB images. CacheBlobs is true by default, meaning that BLOBs are cached in memory. If an application does not need to display the BLOBs associated with records, then set CacheBlobs to false to conserve system resources.

See Also

TTable (see page 10168)

Caching BLOBs

53.1.5.5.4 TBDEDataSet.CachedUpdates Property

Specifies whether cached updates are enabled for a dataset.

Pascal

```
property CachedUpdates: Boolean;
```

C++

```
__property Boolean CachedUpdates;
```

Description

CachedUpdates enables or disables the use of cached updates for a dataset. If CachedUpdates is true, cached updates are enabled. If CachedUpdates is false, cached updates are disabled.

When cached updates are enabled, updates to a dataset (such as posting changes, inserting new records, or deleting records), are stored in an internal cache on the client machine instead of being written directly to the dataset's underlying database tables. When changes are complete, an application writes all cached changes to the database in the context of a single transaction.

Cached updates are most useful to client applications in two-tiered applications. The main benefits of enabling cached updates are:

Fewer transactions and shorter transaction times.

Minimization of network traffic.

The potential drawbacks of enabling cached updates are:

Other applications can access and change the actual data on the server while users are editing local copies of the data, resulting in an update conflict when cached updates are applied to the database.

Other applications cannot access data changes made by an application until its cached updates are applied to the database.

Note: Instead of using cached updates, applications can obtain the same benefits with greater control by using a client dataset and a provider component.

Note: When cached updates are enabled, data changes are stored in local memory. The storage medium is a Paradox format in-memory table. This has implications for applications run in a network environment in that the NET DIR parameter for the Borland (see page 1761) Database Engine (BDE) driver for Paradox needs to be set -- even though the application may be designed to use other database types. Failure to properly set this parameter can result in an EDBEngineError (see page 9854) exception. There are two ways to set NET DIR. One is through the BDE configuration utility BDE Administrator (described in the BDE Administrator online help). The other way is through the NetFileDir property of TSession (see page 10093).

See Also

UpdateObject (see page 9936)

ApplyUpdates (see page 9894)

CancelUpdates (see page 9896)

CommitUpdates (see page 9897)

UpdateRecord (see page 7002)

ApplyUpdates (see page 9970)

CacheBlobs (see page 9932)

NetFileDir (see page 10125)

Enabling BDE-based Cached Updates

Using the BDE to Cache Updates

Using TUpdateSQL to Update a Dataset

53.1.5.5.5 TBDEDataSet.ExpIndex Property

Indicates whether a dataset is using a dBASE expression index.

Pascal

```
property ExpIndex: Boolean;
```

C++

```
__property Boolean ExpIndex;
```

Description

Read ExpIndex to determine if a dataset is using a dBASE expression index. If ExpIndex is true, an expression index is in use; otherwise ExpIndex is false.

53.1.5.5.6 TBDEDataSet.Handle Property

Specifies the Borland (see page 1761) Database Engine (BDE) cursor handle for the dataset.

Pascal

```
property Handle: HDBICur;
```

C++

```
__property HDBICur Handle;
```

Description

Use Handle only to bypass TDataSet methods and call directly into the BDE. Many BDE function calls require a cursor handle parameter. Handle is assigned an initial value when a dataset is opened. If used with a BDE call that changes the current record position, call Resync immediately after returning from the BDE call.

Note: Do not use this property unless an application requires BDE functionality not available through built-in components.

See Also

Locale (see page 9934)

Resync (see page 6994)

53.1.5.5.7 TBDEDataSet.KeySize Property

Specifies the size of the key for the current index of the dataset.

Pascal

```
property KeySize: Word;
```

C++

```
__property Word KeySize;
```

Description

Check (see page 10245) KeySize to determine the size, in bytes, of the key for the current index. KeySize varies depending on the number and type of fields that make up the current index.

See Also

RecordSize (see page 7035)

53.1.5.5.8 TBDEDataSet.Locale Property

Identifies the Borland (see page 1761) Database Engine (BDE) language driver for the dataset.

Pascal

```
property Locale: TLocale;
```

C++

```
__property TLocale Locale;
```

Description

Examine Locale to determine the BDE language driver for the dataset. Applications that make direct calls to the BDE API may need to pass Locale information as a function parameter.

Language driver information may also be available as one of the values in the Params property of the database component that owns the dataset.

Note: Do not use this property unless an application requires BDE functionality not available through built-in components.

See Also

Handle (see page 9933)

Locale (see page 9985)

53.1.5.5.9 TBDEDataSet.ObjectView Property

Specifies whether fields are stored hierarchically or flattened out in the Fields property.

Pascal

```
property ObjectView: Boolean;
```

C++

```
__property Boolean ObjectView;
```

Description

ObjectView affects the way the Fields property stores object fields and the way data-aware grids display ADT and array fields.

When ObjectView is true, the fields are stored hierarchically in the Fields property, meaning any child fields of an object field are referenced by the object field and don't appear sequentially after the object field in the TFields:: .Fields array. When ObjectView is false, the fields are stored sequentially in the Fields property, meaning any child fields of an object field are stored after the object field as siblings in the Fields array.

When ObjectView is false, object field types, such as TADTField, are not created. This switch is provided for increased compatibility with older data-aware controls, which may not handle object field types properly. The default is false for TBDEDataSet, and true for TClientDataSet.

Note: If the ObjectView property of the dataset provided to a TClientDataSet is false, then the TClientDataSet's ObjectView property always acts as false, since the data received is already flattened.

See Also

FieldDefList (see page 7019)

Fields (see page 7023)

Displaying ADT and Array Fields

Using Nested Detail Tables

Using Object Fields

Working with ADT Fields

Working with Array Fields

53.1.5.5.10 TBDEDataSet.StmtHandle Property

Specifies the Borland (see page 1761) Database Engine (BDE) statement handle for the dataset.

Pascal

```
property StmtHandle: HDBIStmt;
```

C++

```
__property HDBIStmt StmtHandle;
```

Description

Use StmtHandle for direct calls to the BDE that bypass the methods of this dataset. Some BDE API calls require a statement

handle as a parameter. For example, TBDEDataSet (see page 9878) descendants use StmtHandle when preparing a query or binding parameters.

Typically, applications do not need to access this property. In TBDEDataSet (see page 9878), StmtHandle is protected, and so unavailable to applications. Some descendants make this property public so that you can bypass the dataset methods and make direct calls to the BDE.

See Also

Handle (see page 9933)

Prepare (see page 10077)

Prepare (see page 10157)

Params (see page 10087)

Params (see page 10165)

53.1.5.5.11 TBDEDataSet.UpdateObject Property

Specifies the update object component used to update a read-only result set when cached updates are enabled.

Pascal

```
property UpdateObject: TDataSetUpdateObject;
```

C++

```
__property TDataSetUpdateObject UpdateObject;
```

Description

Use UpdateObject to specify the TUpdateObject component to use in an application that must be able to update a read-only result set returned by the Borland (see page 1761) Database Engine (BDE).

The BDE always attempts to provide an updatable, or "live" query result unless an application specifically requests a read-only view of data. In some cases, such as a query made against multiple tables, a live result set cannot be returned. In these cases, UpdateObject can be used to specify a TUpdateSQL (see page 10229) component that performs updates as a separate transaction that is transparent to the application.

See Also

CachedUpdates (see page 9932)

ApplyUpdates (see page 9894)

TUpdateSQL (see page 10229)

Using the BDE to Cache Updates

Using Update Objects to Update a Dataset

Updating a Read-only Result Set

Using TQuery

Using TUpdateSQL to Update a Dataset

53.1.5.5.12 TBDEDataSet.UpdateRecordTypes Property

Specifies the type of records visible in a dataset when cached updates are enabled.

Pascal

```
property UpdateRecordTypes: TUpdateRecordTypes;
```

C++

```
__property TUpdateRecordTypes UpdateRecordTypes;
```

Description

Use UpdateRecordTypes to specify the records that are visible in a dataset when cached updates are enabled. UpdateRecordTypes is a set that can have the following values:

Value	Meaning
rtModified	Modified records are visible.
rtInserted	Inserted records are visible.
rtDeleted	Deleted records are visible.
rtUnmodified	Unmodified records are visible.

By default, a dataset is created with an UpdateRecordTypes set of [rtModified, rtInserted, rtUnmodified], meaning that all existing, edited, or inserted records are visible to the user.

An application that must cycle through a dataset to undelete records may change UpdateRecordTypes as part of an undelete method, so that deleted records are "visible" long enough to restore them to their previously undeleted conditions.

An application might also use UpdateRecordTypes like a filter to temporarily limit visible records to those added or inserted by the user during the current session.

See Also

CachedUpdates (see page 9932)

UpdateObject (see page 9936)

OnUpdateRecord (see page 9957)

RevertRecord (see page 9922)

UpdateStatus (see page 9928)

Using the BDE to Cache Updates

53.1.5.5.13 TBDEDataSet.UpdatesPending Property

Indicates whether the cached updates buffer contains records that are not yet applied.

Pascal

```
property UpdatesPending: Boolean;
```

C++

```
__property Boolean UpdatesPending;
```

Description

Examine UpdatesPending to check the status of the cached updates buffer. If UpdatesPending is true, then there are edited, deleted, or inserted records to apply to the database. If UpdatesPending is false, there are no records in the cache.

See Also

CachedUpdates (see page 9932)

OnUpdateRecord (see page 9957)

UpdateStatus (see page 9928)

Enabling BDE-based Cached Updates

Using the BDE to Cache Updates

53.1.5.6 TBDEDataSet Events

The events of the TBDEDataSet class are listed here.

Published Events

	Name	Description
	AfterCancel (see page 9938)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 9940)	Occurs after an application closes a dataset.
	AfterDelete (see page 9942)	Occurs after an application deletes a record.
	AfterEdit (see page 9944)	Occurs after an application starts editing a record.
	AfterInsert (see page 9945)	Occurs after an application inserts a new record.
	AfterOpen (see page 9946)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 9947)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 9947)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 9948)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 9948)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 9949)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 9949)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 9950)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 9950)	Occurs before an application enters insert mode.
	BeforeOpen (see page 9952)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 9952)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 9953)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 9953)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 9954)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 9955)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 9955)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnNewRecord (see page 9955)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 9956)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnUpdateError (see page 9956)	Occurs if an exception is generated when cached updates are applied to a database.
	OnUpdateRecord (see page 9957)	Occurs when cached updates are applied to a record.

53.1.5.6.1 TBDEDataSet.AfterCancel Event

Occurs after an application completes a request to cancel modifications to the active record.

Pascal

```
property AfterCancel: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterCancel;
```

Description

Write an AfterCancel event handler to take specific action after an application cancels changes to the active record. AfterCancel is called by the Cancel method after it updates the current position, releases the lock on the active record if necessary, and sets the dataset state to dsBrowse. If an application requires additional processing before returning control to a user after a Cancel event, code it in the AfterCancel event.

Delphi Examples:

```
{
This example updates the form’s status bar with a message
when an AfterCancel event occurs.  Set the StatusBar
SimplePanel property to True.
}
procedure TForm1.Table1AfterCancel(DataSet: TDataSet);
begin
  StatusBar1.SimpleText := 'Record changes cancelled for ' + DataSet.Name;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Table1.Cancel;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  Table1.Post
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Name := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
    begin
        Table1.Close;
        Table1.DeleteTable;
    end;
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
```

```
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.AfterCancel:= Table1AfterCancel;
    Table1.Active:= True;
  end;
```

See Also

BeforeCancel (see page 7048)

Cancel (see page 6916)

State (see page 7036)

53.1.5.6.2 TBDEDataSet.AfterClose Event

Occurs after an application closes a dataset.

Pascal

```
property AfterClose: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterClose;
```

Description

Write an AfterClose event handler to take specific action immediately after an application closes a dataset. For example, as a security measure, an application might clear a PASSWORD entry from the Params property of a TSQLConnection component when the dataset is closed.

AfterClose is called after a dataset is closed and the dataset state is set to dsInactive.

Delphi Examples:

```
{
This example adds an entry to a memo with a message when an
AfterOpen event or an AfterClose event occurs.  Populate the
ClientDataSet events with CheckButtonActive.
}
procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  if (Button1.Enabled) then
    Memo2.Lines.Add('Now that the dataset is open, record information is available.')
  else
    Memo2.Lines.Add('Now that the dataset is closed, record information is not available.');
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
  RemoteServer1.Connected := False;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].DisableControls;
  end;
end;

procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
```

See Also

Active (see page 7004)

State (see page 7036)

BeforeClose (see page 7048)

Close (see page 6920)

Params (see page 20253)

Opening and Closing Datasets

53.1.5.6.3 TBDEDataSet.AfterDelete Event

Occurs after an application deletes a record.

Pascal

```
property AfterDelete: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterDelete;
```

Description

Write an AfterDelete event handler to take specific action immediately after an application deletes the active record in a dataset. AfterDelete is called by Delete after it deletes the record, sets the dataset state to dsBrowse, and repositions the current record.

C++ Examples:

```
/*
This example displays a message on the form's status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
*/
void __fastcall TForm1::Table1AfterDelete(TDataSet *DataSet)
{
  StatusBar1->SimpleText = Format(
    "There are now %d records in the table",
    ARRAYOFCONST(((int)DataSet->RecordCount)));
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Table1->Delete();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Table1 = new TTable(Form1);
  Table1->Active = false; // The Table component must not be active
  Table1->DatabaseName = "DBDEMOS";
  Table1->TableType = ttParadox;
  Table1->TableName = "CustInfo";
  Table1->Active = False;
  if (!Table1->Exists) // Don't overwrite an existing table
  {
    Table1->Close();
    Table1->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Table1->FieldDefs->Clear();
  TFieldDef *pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Table1->IndexDefs->Clear();
```

```
  /* the 1st index has no name because it is a Paradox primary key */
  Table1->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Table1->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Table1->CreateTable();
  Table1->Active = True;
  for (int i = 1; i <= 20; i++)
    Table1->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  DS2->DataSet = Table1;
  DBGrid2->DataSource->DataSet = Table1;
  Table1->AfterDelete = Table1AfterDelete;
  Table1->Active = True;
}
```

Delphi Examples:

```
{
This example displays a message on the form's status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
}
procedure TForm1.Table1AfterDelete(DataSet: TDataSet);
begin
  StatusBar1.SimpleText := Format('There are now %d records in the table',
[DataSet.RecordCount]);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Delete;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Name := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
    begin
        Table1.Close;
        Table1.DeleteTable;
    end;
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
```

```
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.AfterDelete:= Table1AfterDelete;
    Table1.Active:= True;
  end;
```

See Also

State (see page 7036)

BeforeDelete (see page 7049)

Delete (see page 6924)

53.1.5.6.4 TBDEDataSet.AfterEdit Event

Occurs after an application starts editing a record.

Pascal

```
property AfterEdit: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterEdit;
```

Description

Write an AfterEdit event handler to take specific action immediately after dataset enters edit mode. AfterEdit is called by Edit after it enables editing of a record, recalculates calculated fields, and calls the data event handler to process a record change.

Delphi Examples:

```
{
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
```

```
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

BeforeEdit (see page 7049)

Edit (see page 6932)

53.1.5.6.5 TBDEDataSet.AfterInsert Event

Occurs after an application inserts a new record.

Pascal

```
property AfterInsert: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterInsert;
```

Description

Write an AfterInsert event handler to take specific action immediately after an application inserts a record. The Insert and Append methods generate an AfterInsert event after inserting or appending a new record.

Delphi Examples:

```
{
This example updates a memo with a message when an
AfterInsert event or a BeforeInsert event occurs.
}
procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After inserting a record');
end;

procedure TForm1.ClientDataSet1BeforeInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before inserting a record');
end;
```

See Also

BeforeInsert (see page 7050)

Append (see page 6910)

Insert (see page 6959)

53.1.5.6.6 TBDEDataSet.AfterOpen Event

Occurs after an application completes opening a dataset and before any data access occurs.

Pascal

```
property AfterOpen: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterOpen;
```

Description

Write an AfterOpen event handler to take specific action immediately after an application opens the dataset. AfterOpen is called after the dataset establishes access to its data and the dataset is put into dsBrowse state. For example, an AfterOpen event handler might check an ini file to determine the last record touched in the dataset the previous time the application ran, and position the dataset at that record.

Delphi Examples:

```
{
This example adds an entry to a memo with a message when an
AfterOpen event or an AfterClose event occurs.  Populate the
ClientDataSet events with CheckButtonActive.
}
procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  if (Button1.Enabled) then
    Memo2.Lines.Add('Now that the dataset is open, record information is available.')
  else
    Memo2.Lines.Add('Now that the dataset is closed, record information is not available.');
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
  RemoteServer1.Connected := False;
  with RemoteServer1 do
  begin
```

```
    for I := 0 to DataSetCount - 1 do
      DataSets[I].DisableControls;
  end;
end;

procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
```

See Also

Active (see page 7004)

State (see page 7036)

BeforeOpen (see page 7051)

Open (see page 6976)

Opening and Closing Datasets

53.1.5.6.7 TBDEDataSet.AfterPost Event

Occurs after an application writes the active record to the database or change log and returns to browse state.

Pascal

```
property AfterPost: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterPost;
```

Description

Write an AfterPost event handler to take specific action immediately after an application posts a change to the active record. AfterPost is called after a modification or insertion is made to a record.

Delphi Examples:

```
{
This example updates the form's status bar with a message
when an AfterPost event occurs.
}
procedure TForm1.ClientDataSet1AfterPost(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before posting a record');
end;
```

See Also

BeforePost (see page 7052)

Post (see page 6977)

53.1.5.6.8 TBDEDataSet.AfterRefresh Event

Occurs after an application refreshes the data in the dataset.

Pascal

```
property AfterRefresh: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterRefresh;
```

Description

Write an AfterRefresh event handler to take specific action immediately after an application has updated the records in the dataset. AfterRefresh is generated by calls to the Refresh method.

See Also

BeforeRefresh (see page 7052)

Refresh (see page 6991)

53.1.5.6.9 TBDEDataSet.AfterScroll Event

Occurs after an application scrolls from one record to another.

Pascal

```
property AfterScroll: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterScroll;
```

Description

Write an AfterScroll event handler to take specific action immediately after an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. AfterScroll is called after all other events triggered by these methods and any other methods that switch from record to record in the dataset.

See Also

BeforeScroll (see page 7053)

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Prior (see page 6979)

Navigating Datasets

53.1.5.6.10 TBDEDataSet.BeforeCancel Event

Occurs before an application executes a request to cancel changes to the active record.

Pascal

```
property BeforeCancel: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeCancel;
```

Description

Write a BeforeCancel event to take specific action before an application carries out a request to cancel changes. BeforeCancel is called by the Cancel method before it cancels a dataset operation such as Edit, Insert, or Delete.

An application might use the BeforeCancel event to record a user's changes in an undo buffer.

See Also

AfterCancel (see page 7038)

Delete (see page 6924)

Edit (see page 6932)

Insert (see page 6959)

53.1.5.6.11 TBDEDataSet.BeforeClose Event

Occurs immediately before the dataset closes.

Pascal

```
property BeforeClose: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeClose;
```

Description

Write a BeforeClose event to take specific action before an application closes a dataset. Calling Close or setting the Active property to false results in a call to the BeforeClose event handler.

See Also

Active (see page 7004)

AfterClose (see page 7040)

Close (see page 6920)

Opening and Closing Datasets

Posting Data

53.1.5.6.12 TBDEDataSet.BeforeDelete Event

Occurs before an application attempts to delete the active record.

Pascal

```
property BeforeDelete: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeDelete;
```

Description

Write a BeforeDelete event handler to take specific action before an application deletes the active record. BeforeDelete is called by Delete before it actually deletes a record.

In Delphi, making use of this event an application might, for example, display a dialog box asking for confirmation before deleting the record. On denial of confirmation, the application could abort the deletion by calling the Abort procedure.

See Also

AfterDelete (see page 7041)

Delete (see page 6924)

Abort (see page 22786)

53.1.5.6.13 TBDEDataSet.BeforeEdit Event

Occurs before an application enters edit mode for the active record.

Pascal

```
property BeforeEdit: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeEdit;
```

Description

Write a BeforeEdit event handler to take specific action before an application enables editing of the active record. For example, an application that keeps a log of database edits could use the BeforeEdit event to record the edit request, time, and user before entering edit state.

Delphi Examples:

```
{
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

State (see page 7036)

AfterEdit (see page 7044)

Edit (see page 6932)

53.1.5.6.14 TBDEDataSet.BeforeInsert Event

Occurs before an application enters insert mode.

Pascal

```
property BeforeInsert: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeInsert;
```

Description

Write a BeforeInsert event handler to take specific action before an application inserts or appends a new record. The Insert or

Append method generates a BeforeInsert method before it sets the dataset into dsInsert state.

Delphi Examples:

```
{
This example updates a memo with a message when an
AfterInsert event or a BeforeInsert event occurs.
}
procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After inserting a record');
end;

procedure TForm1.ClientDataSet1BeforeInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before inserting a record');
end;
{
This example uses the BeforeInsert event to do data
validation. If the StrToInt function raises an exception,
the edit control’s contents are set to a valid value so
that the assignment to the INTEGER field in the dataset
will succeed.  Continuing after the exception causes a
'0' to be placed in the PORTA field.
DataSet design:
This example requires a ClientDataSet, a DBGrid, a
DataSource and a DBNavigator.  Name the ClientDataSet CDS.
Set the DataSet property in the DataSource to CDS and name
that DataSource DS.  Set the DataSource in the DBNavigator
to DS the DataSource in the DBGrid to DS.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
type to Integer.  The file used to load CDS must have an
attribute name of HOST with an attribute type of string and
an attribute name of PORTA with an attribute type of integer.
Add DB, DBCtrls, Grids, DBGrids and DBClient to the uses clause.
}
{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button3Click(Sender: TObject);
begin
  CDS.Insert;
  CDS.FieldByName('PORTA').AsInteger := StrToInt(Edit1.Text);
  CDS.Post;
end;

procedure TForm1.CDSBeforeInsert(DataSet: TDataSet);
begin
  try
  { Make sure edit field can be converted to integer --
   this will raise an exception if it can’t }
    StrToInt(Edit1.Text);
  except
    Edit1.Text := '0';
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;
```

See Also

State (see page 7036)

AfterInsert (see page 7044)

Append (see page 6910)

Insert (see page 6959)

53.1.5.6.15 TBDEDataSet.BeforeOpen Event

Occurs before an application executes a request to open a dataset.

Pascal

```
property BeforeOpen: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeOpen;
```

Description

Write a BeforeOpen event handler to take specific action before an application opens a dataset for viewing or editing. BeforeOpen is triggered when an application sets the Active property to true for a dataset or an application calls Open.

See Also

Active (see page 7004)

AfterOpen (see page 7045)

Open (see page 6976)

Opening and Closing Datasets

53.1.5.6.16 TBDEDataSet.BeforePost Event

Occurs before an application posts changes for the active record to the database or change log.

Pascal

```
property BeforePost: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforePost;
```

Description

Write a BeforePost event handler to take specific action before an application posts dataset changes. BeforePost is triggered when an application calls the Post method. Post checks to make sure all required fields are present, then calls BeforePost before posting the record.

An application might use BeforePost to perform validity checks on data changes before committing them. If it encountered a validity problem, it could call Abort to cancel the Post operation (Delphi) or throw an exception (C++).

Delphi Examples:

```
{
This example checks for a valid entry in a TDBEdit control
and calls the Abort procedure if the control is empty; Abort
cancels the post before it happens.
}
procedure TForm1.ClientDataSet1BeforePost(DataSet: TDataSet);
begin
  if DBEdit1.Text = '' then
    Abort;
  DataSet.Fields[2].AsString := DBEdit1.Text;
end;
```

See Also

AfterPost (see page 7046)

Post (see page 6977)

Abort (see page 22786)

Working with Field Component Methods at Runtime

53.1.5.6.17 TBDEDataSet.BeforeRefresh Event

Occurs immediately before an application refreshes the data in the dataset.

Pascal

```
property BeforeRefresh: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeRefresh;
```

Description

Write a BeforeRefresh event handler to take specific action immediately before an application updates the records in the dataset. BeforeRefresh is generated by calls to the Refresh method.

See Also

AfterRefresh (see page 7047)

Refresh (see page 6991)

53.1.5.6.18 TBDEDataSet.BeforeScroll Event

Occurs before an application scrolls from one record to another.

Pascal

```
property BeforeScroll: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeScroll;
```

Description

Write a BeforeScroll event handler to take specific action immediately before an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. BeforeScroll is called before all other events triggered by these methods and any other methods that switch from record to record in the dataset.

See Also

AfterScroll (see page 7047)

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Prior (see page 6979)

Navigating Datasets

53.1.5.6.19 TBDEDataSet.OnCalcFields Event

Occurs when an application recalculates calculated fields.

Pascal

```
property OnCalcFields: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent OnCalcFields;
```

Description

Write an OnCalcFields event handler to take specific action when an application recalculates calculated fields. A calculated field is one that derives its value from the values in one or more fields in the dataset, sometimes with additional processing.

OnCalcFields is triggered when:

A dataset is opened.

A dataset is put into dsEdit state.

A record is retrieved from a database.

When the AutoCalcFields property is true, OnCalcFields is also triggered when:

Focus moves from one visual control to another, or from one column to another is a data-aware grid control and modifications were made to the record.

Note: When the AutoCalcFields property is true, an OnCalcFields event handler should not modify the dataset (or a linked dataset if it is part of a master-detail relationship), because such modifications retrigger the OnCalcField event, leading to infinite recursion.

If an application permits users to change data, OnCalcFields is frequently triggered. To reduce the frequency with which OnCalcFields occurs, set AutoCalcFields to false.

Warning: When the dataset is the master table of a master-detail relationship, OnCalcFields occurs before detail sets have been synchronized with the master table.

See Also

Active (see page 7004)

AutoCalcFields (see page 7006)

Open (see page 6976)

Post (see page 6977)

Calculating Fields

Using Internally Calculated Fields in Client Datasets

Defining a Calculated Field

53.1.5.6.20 TBDEDataSet.OnDeleteError Event

Occurs when an application attempts to delete a record and an exception is raised.

Pascal

```
property OnDeleteError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnDeleteError;
```

Description

Write an OnDeleteError event handler to handle exceptions that occur when an attempt to delete a record fails.

See Also

OnEditError (see page 7054)

OnPostError (see page 7056)

Delete (see page 6924)

53.1.5.6.21 TBDEDataSet.OnEditError Event

Occurs when an application attempts to modify or insert a record and an exception is raised.

Pascal

```
property OnEditError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnEditError;
```

Description

Write an OnEditError event handler to handle exceptions that occur when an attempt to edit a record fails.

See Also

OnDeleteError (see page 7054)

OnPostError (see page 7056)

Edit (see page 6932)

Insert (see page 6959)

53.1.5.6.22 TBDEDataSet.OnNewRecord Event

Occurs when an application inserts or appends a new dataset record.

Pascal

```
property OnNewRecord: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent OnNewRecord;
```

Description

Write an OnNewRecord event handler to take specific actions as an application inserts or appends a new record. OnNewRecord

is called as part of the insert or append process. An application might use the OnNewRecord event to set initial values for a record or as a way of implementing cascading insertions in related datasets.

See Also

AfterInsert (see page 7044)

BeforeInsert (see page 7050)

53.1.5.6.23 TBDEDataSet.OnPostError Event

Occurs when an application attempts to modify or insert a record and an exception is raised.

Pascal

```
property OnPostError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnPostError;
```

Description

Write an OnPostError event handler to handle exceptions that occur when an attempt to post a record fails.

See Also

OnDeleteError (see page 7054)

OnEditError (see page 7054)

Post (see page 6977)

53.1.5.6.24 TBDEDataSet.OnUpdateError Event

Occurs if an exception is generated when cached updates are applied to a database.

Pascal

```
property OnUpdateError: TUpdateErrorEvent;
```

C++

```
__property TUpdateErrorEvent OnUpdateError;
```

Description

Write an OnUpdateError event handler to respond to exceptions generated while applying cached updates to a database.

Because there is a delay between the time a record is first cached and the time cached updates are applied, there is a possibility that another application may change one or more of the same records in the database before the cached changes can be applied. The Borland (see page 1761) Database Engine (BDE) checks for this condition and raises an exception. TBDEDataSet (see page 9878) responds by calling the OnUpdateError event handler if it exists.

DataSet is the name of the dataset to which updates are applied.

E is a pointer to a EDBEngineError (see page 9854) object from which an application can extract an error message and the actual cause of the error condition. The OnUpdateError handler can use this information to determine how to respond to the error condition.

UpdateKind indicates whether the error occurred while inserting, deleting, or modifying a record.

UpdateAction indicates the action to take when the OnUpdateError handler exits. On entry into the handler, UpdateAction is always set to uaFail. If OnUpdateError can handle or correct the error, set UpdateAction to uaRetry before exiting the error handler.

The error handler can use the OldValue and NewValue properties TField to evaluate error conditions and set NewValue to a new value to reapply. In this case, set UpdateAction to uaRetry before exiting.

Note: If a call to ApplyUpdates (see page 9894) raises an exception and ApplyUpdates (see page 9894) is not called within the context of a exception block, an error message is displayed. If an OnUpdateError handler cannot correct the error condition and leaves UpdateAction set to uaFail, the error message is displayed twice. To prevent redisplay, set UpdateAction to uaAbort in the error handler.

Warning: Important:The code in an OnUpdateError handler must not call any methods that make a different record the current one.

See Also

ApplyUpdates (see page 9894)

CachedUpdates (see page 9932)

EDBEngineError (see page 9854)

OnUpdateRecord (see page 9957)

TField (see page 7115)

UpdateObject (see page 9936)

UpdateRecordTypes (see page 9936)

UpdatesPending (see page 9937)

Using the BDE to Cache Updates

Handling Cached Update Errors

53.1.5.6.25 TBDEDataSet.OnUpdateRecord Event

Occurs when cached updates are applied to a record.

Pascal

```
property OnUpdateRecord: TUpdateRecordEvent;
```

C++

```
__property TUpdateRecordEvent OnUpdateRecord;
```

Description

Write an OnUpdateRecord event handler to process updates that cannot be handled by a single update component, such as implementation of cascading updates, insertions, or deletions. This handler is also useful for applications that require additional control over parameter substitution in update components.

DataSet is the name of the dataset to which updates are applied.

UpdateKind indicates whether the current update is the insertion of a record, the deletion of a record, or the modification of a record.

UpdateAction indicates the action taken by the OnUpdateRecord handler before it exits. On entry into the handler, UpdateAction is always set to uaFail. If OnUpdateRecord is successful, it should set UpdateAction to uaApplied before exiting.

The code in an OnUpdateRecord handler must not call any methods that make a different record the current one.

Note: As an alternate approach, application can use a client dataset and a provider component to handle cached updates. This alternative provides more support and greater control.

See Also

CachedUpdates (see page 9932)

UpdateObject (see page 9936)

UpdateRecordTypes (see page 9936)

UpdatesPending (see page 9937)

OnUpdateError (see page 9956)

ApplyUpdates (see page 9894)

CancelUpdates (see page 9896)

CommitUpdates (see page 9897)

RevertRecord (see page 9922)

UpdateRecord (see page 7002)

UpdateStatus (see page 9928)

Using Multiple Update Objects

Using the BDE to Cache Updates

Using Update Objects to Update a Dataset

53.1.6 TBlobStream Class

TBlobStream is a stream object that provides services which allow applications to read from or write to field objects that represent Binary large object (BLOB) fields.

Class Hierarchy

Pascal

```
TBlobStream = class(TStream);
```

C++

```
class TBlobStream : public TStream;
```

File

DBTables

Description

Use TBlobStream to access or modify the value of a BLOB field in a BDE-enabled dataset. TBlob stream works with persistent TBlobField objects (including descendants of TBlobField such as TGraphicField and TMemoField). BLOB fields use BLOB streams to read data from and write data to the dataset.

TBlobStream allows objects that have no specialized knowledge of how data is stored in a BLOB field to read or write such data by employing the uniform stream mechanism.

To use a BLOB stream, create an instance of TBlobStream, use the methods of the stream to read or write the data, and then free the BLOB stream. Do not use the same instance of TBlobStream to access data from more than one record. Instead, create a new TBlobStream object every time you need to read or write BLOB data on a new record.

See Also

TBlobField (see page 6801)

TGraphicField (see page 7257)

TMemoField (see page 7348)

TClientBlobStream (see page 8285)

TADOBlobStream (see page 979)

Specifying the HTML Template

53.1.6.1 TBlobStream Members

The following tables list the members exposed by TBlobStream.

Protected Methods

	Name	Description
	GetSize (see page 2982)	This is GetSize, a member of class TStream.
	SetSize (see page 2989)	This is the overview for the SetSize method overload.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CopyFrom (see page 2979)	Copies a specified number of bytes from one stream to another.
	Create (see page 9960)	Creates an instance of TBlobStream (see page 9958).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 9961)	Destroys an instance of TBlobStream (see page 9958).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FixupResourceHeader (see page 2981)	Patches the resource header for a resource that has been written to the stream.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Read (see page 9962)	Reads up to Count bytes from the current position in the field's data into Buffer.
	ReadBuffer (see page 2983)	Reads bytes from the stream into Buffer.
	ReadComponent (see page 2983)	Initiates streaming of components and their properties.
	ReadComponentRes (see page 2987)	Reads components and their properties from a stream in a resource file format.
	ReadResHeader (see page 2987)	Reads a resource-file header from the stream.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Seek (see page 9962)	Resets the current position of the TBlobStream (see page 9958) object.
	Seek (see page 2988)	This is the overview for the Seek method overload.
	Truncate (see page 9963)	Discards all data in the BLOB field from the current position on.
	Write (see page 9963)	Writes Count bytes from Buffer to the current position in the field and updates the current position by Count bytes.
	WriteBuffer (see page 2990)	Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.
	WriteComponent (see page 2990)	Initiates the writing of components and their properties to a stream.
	WriteComponentRes (see page 2994)	Writes components and their properties to a stream from a resource file format.
	WriteDescendent (see page 2994)	Streams components and their properties in inherited forms.
	WriteDescendentRes (see page 2995)	Streams components and their properties in inherited forms.
	WriteResourceHeader (see page 2995)	Writes a resource-file header to the stream.

Public Properties

	Name	Description
	Position (see page 2996)	Indicates the current offset into the stream for reading and writing.
	Size (see page 2997)	Indicates the size in bytes of the stream.

53.1.6.2 TBlobStream.Create Constructor

Creates an instance of TBlobStream (see page 9958).

Pascal

```
constructor Create(Field: TBlobField; Mode: TBlobStreamMode);
```

C++

```
__fastcall TBlobStream(TBlobField Field, TBlobStreamMode Mode);
```

Description

Call Create to obtain an instance of TBlobStream (see page 9958) for reading from or writing to a specific TBlobField object.

Tip: Different types of dataset use different classes of BLOB streams. It is usually a better idea to call the dataset's CreateBlobStream method than to call the BLOB stream constructor.

Create links the TBlobStream (see page 9958) to the field object specified by the Field parameter. Mode specifies whether the BLOB stream will be used to read data (bmRead), write data (bmWrite) or modify data (bmReadWrite). Mode must be one of the following values:

Value	Meaning
bmRead	The BLOB stream can read data from the field.
bmWrite	The BLOB stream can replace the data in the field.
bmReadWrite	The BLOB stream can modify the data in the field.

Note: : A TBlobStream (see page 9958) object should always be destroyed before navigating to a new record. Create a new instance of TBlobStream (see page 9958) for the new record.

Delphi Examples:

```
{
The following example copies the data in the Notes field of
Table1 or SQLDataSet1 to the Remarks field of ClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Stream1: TBlobStream;
  Stream2: TStream;
  blobType : TBlobType;
begin
  blobType := Table1Notes.BlobType;
  Stream1 := TBlobStream.Create(Table1Notes, bmRead);
  try
    CDS2.Edit;
    { here's a different way to create a blob stream }
    Stream2 := CDS2.CreateBlobStream(CDS2.FieldByName('Remarks'), bmReadWrite);
    try
      Stream2.CopyFrom(Stream1, Stream1.Size);
//    CDS2.Post;
//    CDS2.Active := True;
    finally
      Stream2.Free;
    end;
  finally
    Stream1.Free;
  end;
end;
```

See Also

TBlobField (see page 6801)

CreateBlobStream (see page 6922)

53.1.6.3 TBlobStream.Destroy Destructor

Destroys an instance of TBlobStream (see page 9958).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBlobStream();
```

Description

Do not call Destroy directly in an application. Instead call Free. Free verifies that the TBlobStream (see page 9958) object is not nil and only then calls Destroy. Destroy the TBlobStream (see page 9958) object when it is no longer needed for reading from or writing to the BLOB field.

Destroy triggers an OnDataChange event if the BLOB stream was used to overwrite or modify the data in the field. Destroy then frees any buffers that were allocated to handle the data.

Note: : A TBlobStream (see page 9958) object should always be destroyed before navigating to a new record. Create (see page 9960) a new instance of TBlobStream (see page 9958) for the new record.

See Also

Free (see page 22300)

OnDataChange (see page 7080)

OnDataChange (see page 9497)

delete

53.1.6.4 TBlobStream Methods

The methods of the TBlobStream class are listed here.

Public Methods

	Name	Description
	Read (see page 9962)	Reads up to Count bytes from the current position in the field's data into Buffer.
	Seek (see page 9962)	Resets the current position of the TBlobStream (see page 9958) object.
	Truncate (see page 9963)	Discards all data in the BLOB field from the current position on.
	Write (see page 9963)	Writes Count bytes from Buffer to the current position in the field and updates the current position by Count bytes.

53.1.6.4.1 TBlobStream.Read Method

Reads up to Count bytes from the current position in the field's data into Buffer.

Pascal

```
function Read(var Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Read( Buffer, Longint Count);
```

Description

Call Read to read data from the BLOB field when the number of bytes in the field's data is not known.

Buffer is the target container for Read and must be of type PChar (Delphi), char * (C++), or equivalent. Buffer must have at least Count bytes allocated to hold the data that was read from the field. Read transfers up to Count bytes from the BLOB data into Buffer, starting in the current position, and then advances the current position by the number of bytes actually transferred. Read returns the number of bytes actually transferred (which may be less than the number requested in Count).

Read checks the Transliterate property of the field, and converts the data into ANSI from the character set specified by the dataset if Transliterate is true.

All the other data-reading methods of a BLOB stream (ReadBuffer, ReadComponent) call Read to do their actual reading.

Note: Do not call Read when the TBlobStream (see page 9958) was created in bmWrite mode.

See Also

Create (see page 9960)

Position (see page 2996)

ReadBuffer (see page 2983)

ReadComponent (see page 2983)

Transliterate (see page 6821)

53.1.6.4.2 TBlobStream.Seek Method

Resets the current position of the TBlobStream (see page 9958) object.

Pascal

```
function Seek(Offset: Longint; Origin: Word): Longint; override;
```

C++

```
virtual __fastcall Longint Seek(Longint Offset, Word Origin);
```

Description

Use Seek to move the current position within the BLOB data by the indicated offset. Seek allows an application to read from or write to a particular location within the BLOB data.

The Origin parameter indicates how to interpret the Offset parameter. Origin should be one of the following values:

Value	Meaning
soFromBeginning	Offset is from the beginning of the BLOB data. Seek moves to the position Offset. Offset must be >= 0.
soFromCurrent	Offset is from the current position in the BLOB data. Seek moves to Position + Offset.
soFromEnd	Offset is from the end of the BLOB data. Offset must be <= 0 to indicate a number of bytes before the end of the BLOB.

Seek returns the new value of the Position property, the new current position in the BLOB data.

See Also

Position (see page 2996)

Read (see page 9962)

Write (see page 9963)

53.1.6.4.3 TBlobStream.Truncate Method

Discards all data in the BLOB field from the current position on.

Pascal

```
procedure Truncate;
```

C++

```
__fastcall Truncate();
```

Description

Use Truncate to limit the size of the BLOB data. Calling Truncate when the current position is 0 will clear the contents of the BLOB field.

Note: Do not call Truncate when the TBlobStream (see page 9958) was created in bmRead mode.

See Also

Create (see page 9960)

Clear (see page 6810)

53.1.6.4.4 TBlobStream.Write Method

Writes Count bytes from Buffer to the current position in the field and updates the current position by Count bytes.

Pascal

```
function Write(const Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  Buffer, Longint Count);
```

Description

Use Write to write Count bytes to the BLOB field, starting at the current position. The Write method for TBlobStream (see page 9958) always writes the entire Count bytes, as BLOB data does not necessarily include a termination character. Thus, Write is equivalent to the WriteBuffer method.

Buffer is the source container for Write and must be of type PChar (Delphi), char * (C++), or equivalent. Write transfers up to Count bytes from Buffer into the BLOB field, starting in the current position, and then advances the current position by the number of bytes actually transferred. Write returns the number of bytes actually transferred (which may be less than the number requested in Count).

Write checks the Transliterate property of the field, and converts the data from ANSI into the character set of the dataset if Transliterate is true.

All the other data-writing methods of a BLOB stream (WriteBuffer, WriteComponent) call Write to do their actual writing.

Note: Do not call Write when the TBlobStream (see page 9958) was created in bmRead mode.

See Also

Create (see page 9960)

Position (see page 2996)

Transliterate (see page 6821)

WriteBuffer (see page 2990)

WriteComponent (see page 2990)

53.1.7 TDatabase Class

TDatabase provides discrete control over a connection to a single database in a BDE-based database application.

Class Hierarchy

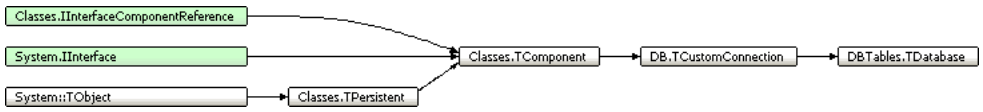

Pascal

```
TDatabase = class(TCustomConnection);
```

C++

```
class TDatabase : public TCustomConnection;
```

File

DBTables

Description

Use TDatabase when a BDE-based database application requires any of the following control over a database connection:

Persistent database connections

Customized database server logins

Transaction control

Application-specific BDE aliases

TDatabase is especially important for the control it permits over database transaction processing with the BDE when connected to a remote SQL database server.

Note: Explicit declaration of a TDatabase component for each database connection in an application is optional if the application does not need to explicitly control that connection. If a TDatabase component is not explicitly declared and instantiated for a database connection, a temporary database component with a default set of properties is created for it at runtime.

See Also

TSession (see page 10093)

TSessionList (see page 10129)

Connecting to Databases: Overview

Associating a Dataset with Database and Session Connections

Connecting to Databases with TDatabase

Distributing Database Applications

Connecting to Databases with TDatabase

Using TQuery

53.1.7.1 TDatabase Members

The following tables list the members exposed by TDatabase.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoConnect (see page 9972)	Provides the interface for a method that opens a connection.
V	DoDisconnect (see page 9972)	Provides the interface for a method that terminates the connection.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetConnected (see page 9974)	Returns the value of the Connected property.
	GetDataSet (see page 9974)	Returns a dataset, given its index in the DataSets property array.
V	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 9975)	Initializes the connection component after all objects in the form or data module have been loaded.
	Notification (see page 9976)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RegisterClient (see page 6866)	Registers an object that uses the connection component.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnected (see page 6867)	Changes the value of the Connected (see page 6868) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	UnRegisterClient (see page 6868)	Removes a registered client so that it is no longer associated with the connection component.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	ApplyUpdates (see page 9970)	Posts pending cached updates for specified datasets to the database server.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 6863)	Closes the connection.
	CloseDataSets (see page 9971)	Closes all datasets associated with the database component without disconnecting from the database server.
	Commit (see page 9971)	Permanently stores updates, insertions, and deletions of data associated with the current transaction, and ends the current transactions.
V	Create (see page 9969)	Creates an instance of a TDatabase (see page 9964) component.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 9969)	Destroys the instance of a database component.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 9973)	Executes an SQL statement.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlushSchemaCache (see page 9973)	Flushes the cached schema information for a table.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldNames (see page 9975)	Populates a string list with the names of fields in a table.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetTableNames (see page 9975)	Populates a string list with the names of tables in the database.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 6866)	Opens the connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Rollback (see page 9976)	Cancels all updates, insertions, and deletions for the current transaction and ends the transaction.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartTransaction (see page 9977)	Begins a new transaction against the database server.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateName (see page 9977)	Raises an exception if a specified database is already open in the active session.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 9981)	Provides an indexed array of all active datasets for a database component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Directory (see page 9981)	Specifies the working directory for a Paradox or dBASE database.
	Handle (see page 9983)	Specifies the Borland (see page 1761) Database Engine (BDE) database handle.
R	InTransaction (see page 9983)	Indicates whether a database transaction is in progress or not.
R	IsSQLBased (see page 9984)	Indicates if a database component is using either the Borland (see page 1761) Database Engine (BDE) SQL Links driver or the BDE ODBC socket.
R	Locale (see page 9985)	Identifies the Borland (see page 1761) Database Engine (BDE) language driver for the database component.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Session (see page 9988)	Points to the session component with which this database component is associated.
R	SessionAlias (see page 9988)	Specifies whether a database component is using a session alias.
	Temporary (see page 9989)	Indicates whether a database component is a temporary one, created by the system as needed, or a persistent one explicitly created, managed, and freed within the application.
	TraceFlags (see page 9990)	Specifies the database operations to track with the SQL Monitor at runtime.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterConnect (see page 9992)	Occurs after a connection is established.
	AfterDisconnect (see page 9993)	Occurs after the connection closes.
	BeforeConnect (see page 9993)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 9993)	Occurs immediately before the connection closes.
	OnLogin (see page 9994)	Occurs when an application connects to a database.

Published Properties

	Name	Description
	AliasName (see page 9979)	Specifies the Borland (see page 1761) Database Engine (BDE) alias used by this database connection.
	Connected (see page 9979)	Indicates whether a database connection is active.
	DatabaseName (see page 9980)	Specifies the name of the database to associate with this database component.
	DriverName (see page 9981)	Specifies the name of the BDE driver for the database.

	Exclusive (see page 9982)	Enables an application to gain sole access to a database.
	HandleShared (see page 9983)	Specifies whether to share a database handle.
	KeepConnection (see page 9984)	Specifies whether an application remains connected to a database even if no datasets are open.
	LoginPrompt (see page 9985)	Specifies whether a login dialog appears immediately before opening a new connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Params (see page 9986)	Contains database connection parameters for the Borland (see page 1761) Database Engine (BDE) alias associated with the database component.
	ReadOnly (see page 9987)	Specifies that the database connection provides read-only access.
	SessionName (see page 9989)	Identifies the name of the session used by this database component.
	Tag (see page 2808)	Stores an integer value as part of a component.
	TransIsolation (see page 9991)	Specifies the transaction isolation level for transactions managed by the Borland (see page 1761) Database Engine (BDE).

53.1.7.2 TDatabase.Create Constructor

Creates an instance of a TDatabase (see page 9964) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDatabase(TComponent * AOwner);
```

Description

Call Create to instantiate a database component at runtime. An application can create a database component in order to control the component's existence and set its properties and events, or an application can let Delphi create temporary database components as needed at runtime.

AOwner is the component that is responsible for freeing the database instance. It becomes the value of the Owner property.

See Also

Owner (see page 2807)

53.1.7.3 TDatabase.Destroy Destructor

Destroys the instance of a database component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDatabase();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which verifies that the database reference is not nil before calling Destroy

Destroy closes all active datasets and disconnects from the database server, if necessary. It then frees the string resources allocated for the Params (see page 9986) and DataSets (see page 9981) properties before calling its inherited destructor.

See Also

DataSets (see page 9981)

Params (see page 9986)

Create (see page 9969)

Free (see page 22300)

53.1.7.4 TDatabase Methods

The methods of the TDatabase class are listed here.

Protected Methods

	Name	Description
	DoConnect (see page 9972)	Provides the interface for a method that opens a connection.
	DoDisconnect (see page 9972)	Provides the interface for a method that terminates the connection.
	GetConnected (see page 9974)	Returns the value of the Connected property.
	GetDataSet (see page 9974)	Returns a dataset, given its index in the DataSets property array.
	Loaded (see page 9975)	Initializes the connection component after all objects in the form or data module have been loaded.
	Notification (see page 9976)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
	ApplyUpdates (see page 9970)	Posts pending cached updates for specified datasets to the database server.
	CloseDataSets (see page 9971)	Closes all datasets associated with the database component without disconnecting from the database server.
	Commit (see page 9971)	Permanently stores updates, insertions, and deletions of data associated with the current transaction, and ends the current transactions.
	Execute (see page 9973)	Executes an SQL statement.
	FlushSchemaCache (see page 9973)	Flushes the cached schema information for a table.
	GetFieldNames (see page 9975)	Populates a string list with the names of fields in a table.
	GetTableNames (see page 9975)	Populates a string list with the names of tables in the database.
	Rollback (see page 9976)	Cancels all updates, insertions, and deletions for the current transaction and ends the transaction.
	StartTransaction (see page 9977)	Begins a new transaction against the database server.
	ValidateName (see page 9977)	Raises an exception if a specified database is already open in the active session.

53.1.7.4.1 TDatabase.ApplyUpdates Method

Posts pending cached updates for specified datasets to the database server.

Pascal

```
procedure ApplyUpdates(const DataSets: array of TDBDataSet);
```

C++

```
__fastcall ApplyUpdates(const array of TDBDataSet DataSets);
```

Description

Call ApplyUpdates to post pending cached updates for a specific set of open datasets to the database server. ApplyUpdates is only meaningful if the CachedUpdates property of a specified dataset is true.

DataSets (see page 9981) is a list of dataset names specifying the datasets for which to post pending updates. DataSets (see page 9981) need not list every currently open dataset. For each listed dataset ApplyUpdates calls the dataset's ApplyUpdates and CommitUpdates methods to post that dataset's pending cached updates.

Applying updates is a two-phase process that takes place within the context of the database component's transaction control. When an application calls ApplyUpdates, the following events take place:

1A database transaction starts.

2Cached updates are written to the database (Phase 1).

If the database write is successful:

1 Database changes are committed, ending the transaction.

2 Cachedupdates are committed, clearing the internal cache buffer (Phase 2).

If the database write fails database changes are rolled back, ending the transaction.

The two-phased approach allows for effective error recovery, especially when updating multiple and interrelated datasets (for example, the datasets associated with a master/detail form).

See Also

Commit (see page 9971)

Open (see page 6866)

Rollback (see page 9976)

StartTransaction (see page 9977)

CachedUpdates (see page 9932)

ApplyUpdates (see page 9894)

CommitUpdates (see page 9897)

Applying Cached Updates Using a Database

Using the BDE to Cache Updates

53.1.7.4.2 TDatabase.CloseDataSets Method

Closes all datasets associated with the database component without disconnecting from the database server.

Pascal

```
procedure CloseDataSets;
```

C++

```
__fastcall CloseDataSets();
```

Description

Call CloseDataSets to close all active datasets without disconnecting from the database server. Ordinarily, when an application calls Close, all datasets are closed, and the connection to the database server is dropped. Calling CloseDataSets instead of Close ensures that an application can close all active datasets without having to reconnect to the database server at a later time.

See Also

KeepConnection (see page 9984)

Close (see page 6863)

53.1.7.4.3 TDatabase.Commit Method

Permanently stores updates, insertions, and deletions of data associated with the current transaction, and ends the current transactions.

Pascal

```
procedure Commit;
```

C++

```
__fastcall Commit();
```

Description

Call Commit to permanently store to the database server all updates, insertions, and deletions of data associated with the current transaction and then end the transaction. The current transaction is the last transaction started by calling StartTransaction (see page 9977).

Note: Before calling Commit, an application may check the status of the InTransaction (see page 9983) property. If an application calls Commit and there is no current transaction, an exception is raised.

See Also

InTransaction (see page 9983)

Rollback (see page 9976)

StartTransaction (see page 9977)

TransIsolation (see page 9991)

53.1.7.4.4 TDatabase.DoConnect Method

Provides the interface for a method that opens a connection.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

The Connected property uses DoConnect to establish a connection. Descendant classes override the DoConnect method to establish their connection as appropriate. As implemented in TCustomConnection, DoConnect does nothing.

See Also

Connected (see page 6868)

DoDisconnect (see page 6864)

53.1.7.4.5 TDatabase.DoDisconnect Method

Provides the interface for a method that terminates the connection.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

The Connected property uses DoDisconnect to close a connection. Descendant classes override the DoDisconnect method to drop a connection. As implemented in TCustomConnection, DoDisconnect does nothing.

See Also

Connected (see page 6868)

DoConnect (see page 6864)

53.1.7.4.6 TDatabase.Execute Method

Executes an SQL statement.

Pascal

```
function Execute(const SQL: string; Params: TParams = nil; Cache: Boolean = False; Cursor:
phDBICur = nil): Integer;
```

C++

```
__fastcall int Execute(const AnsiString SQL, TParams Params = nil, Boolean Cache = False,
phDBICur Cursor = nil);
```

Description

Use Execute to execute an SQL statement against the database without the overhead of using a TQuery (see page 10059) object. Execute supports Data Definition Language (DDL) SQL statements and those Data Manipulation Language (DDL) SQL statements that do not return result sets. Examples of DDL statements include: CREATE INDEX, ALTER TABLE, and DROP DOMAIN. The DML statements that perform an action on data but do not return a result set are: INSERT, DELETE, and UPDATE.

SQL is a string value containing the statement to be executed.

Params (see page 9986) is a value of type TParams and lists any parameters used by the SQL statement. Parameter binding is by index only (not by name). The TParam objects in Params (see page 9986) must be in the same order as the the parameters in the SQL statement to which each corresponds. Use the CreateParam method (TParams) to create one TParams object for each parameter in the SQL statement. Use properties and methods of TParam to configure each TParam and to give each a value prior to calling Execute. If the SQL statement does not include any parameters, pass a nil (Delphi) or NULL (C++) value for Params (see page 9986).

Cache specifies whether the prepared SQL statement is cached for reuse within the current transaction. Caching statements can speed their processing if they are used more than once in a transaction.

Cursor allows you to execute a SELECT statement and from it create a Borland (see page 1761) Database Engine (BDE) dataset. If SQL is a SELECT statement, Cursor returns the BDE cursor to the result set. Cursor is a pointer to an already existing BDE cursor. There is currently no provision in the VCL to assign the cursor created by this parameter to a dataset component (like TTable (see page 10168)). This limits its use to cursors subsequently operated on using BDE API code. The result set returned in Cursor is always read-only. If the SQL statement does not return a result set, pass a nil (Delphi) or NULL (C++) value for Cursor.

Execute returns the number of records affected by executing the SQL statement.

See Also

TQuery (see page 10059)

Handle (see page 9933)

53.1.7.4.7 TDatabase.FlushSchemaCache Method

Flushes the cached schema information for a table.

Pascal

```
procedure FlushSchemaCache(const TableName: string);
```

C++

```
__fastcall FlushSchemaCache(const AnsiString TableName);
```

Description

Call FlushSchemaCache if the schema information for an SQL table is changed (for example, because a field, index, or table is added to or dropped from a database).

When an application opens an SQL table, schema information that describes the field names, field types, and index information is stored in memory. If an application changes the schema for an SQL table, it should call FlushSchemaCache to update the schema information in memory.

See Also

Open (see page 6866)

53.1.7.4.8 TDatabase.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected read implementation of the Connected property. Descendant classes override this method to return true if a connection is open and false otherwise. As defined in TCustomConnection, GetConnected always returns false.

See Also

Connected (see page 6868)

SetConnected (see page 6867)

53.1.7.4.9 TDatabase.GetDataSet Method

Returns a dataset, given its index in the DataSets property array.

Pascal

```
function GetDataSet(Index: Integer): TDBDataSet; reintroduce;
```

C++

```
__fastcall TDBDataSet GetDataSet(int Index);
```

Description

GetDataSet is the protected read implementation of the DataSets property.

See Also

DataSets (see page 6870)

53.1.7.4.10 TDatabase.GetFieldNames Method

Populates a string list with the names of fields in a table.

Pascal

```
procedure GetFieldNames(const TableName: string; List: TStrings);
```

C++

```
__fastcall GetFieldNames(const AnsiString TableName, TStrings List);
```

Description

Call GetFieldNames to retrieve a list of fields in a table.

TableName names the table whose field names you want added to the list.

List is a TStrings descendant that receives the field names. Any existing strings are deleted from the list before GetFieldNames adds the names of all the fields in TableName.

For example, the following line fills a list box with the names of all fields in the Employee table:

```
Database1.GetFieldNames('Employee', ListBox1.Items);
Database1->GetFieldNames("Employee", ListBox1->Items);
```

See Also

GetTableNames (see page 9975)

53.1.7.4.11 TDatabase.GetTableNames Method

Populates a string list with the names of tables in the database.

Pascal

```
procedure GetTableNames(List: TStrings; SystemTables: Boolean = False);
```

C++

```
__fastcall GetTableNames(TStrings List, Boolean SystemTables = False);
```

Description

Call GetTableNames to retrieve a list of tables in the associated database.

List is a TStrings descendant that receives the table names. Any existing strings are deleted from the list before GetTableNames adds the names of all tables in the database.

SystemTables specifies whether the list of table names should include only the database's system tables. If SystemTables is true, only the system tables are added to List. If SystemTables is false, the list is filled with ordinary tables.

For example, the following line fills a list box with the names of all tables in the database:

```
Database1.GetTableNames(ListBox1.Items, False);
Database1->GetTableNames(ListBox1->Items, false);
```

See Also

GetFieldNames (see page 9975)

53.1.7.4.12 TDatabase.Loaded Method

Initializes the connection component after all objects in the form or data module have been loaded.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. It is called automatically after the form or data module that contains the connection component has loaded all its owned objects. TCustomConnection overrides the inherited method to establish a connection if the Connected property is true.

See Also

StreamedConnected (see page 6874)

Loaded (see page 2782)

53.1.7.4.13 TDatabase.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

53.1.7.4.14 TDatabase.Rollback Method

Cancels all updates, insertions, and deletions for the current transaction and ends the transaction.

Pascal

```
procedure Rollback;
```

C++

```
__fastcall Rollback();
```

Description

Call Rollback to cancel all updates, insertions, and deletions for the current transaction and to end the transaction. The current transaction is the last transaction started by calling StartTransaction (see page 9977).

Note: Before calling Rollback, an application may check the status of the InTransaction (see page 9983) property. If an application calls Rollback and there is no current transaction, an exception is raised.

See Also

InTransaction (see page 9983)

Commit (see page 9971)

StartTransaction (see page 9977)

TransIsolation (see page 9991)

Connecting to Databases with TDatabase

53.1.7.4.15 TDatabase.StartTransaction Method

Begins a new transaction against the database server.

Pascal

```
procedure StartTransaction;
```

C++

```
__fastcall StartTransaction();
```

Description

Call StartTransaction to begin a new transaction against the database server. Before calling StartTransaction, an application should check the status of the InTransaction (see page 9983) property and adjust the setting of the TransIsolation (see page 9991) property as desired. If InTransaction (see page 9983) is true, indicating that a transaction is already in progress, a subsequent call to StartTransaction without first calling Commit (see page 9971) or Rollback (see page 9976) to end the current transaction raises an exception.

Updates, insertions, and deletions that take place after a call to StartTransaction are held by the server until an application calls Commit (see page 9971) to save the changes or Rollback (see page 9976) is to cancel them.

See Also

InTransaction (see page 9983)

Commit (see page 9971)

Rollback (see page 9976)

TransIsolation (see page 9991)

53.1.7.4.16 TDatabase.ValidateName Method

Raises an exception if a specified database is already open in the active session.

Pascal

```
procedure ValidateName(const Name: string);
```

C++

```
__fastcall ValidateName(const AnsiString Name);
```

Description

Call ValidateName to prevent duplicate access to a database from within a single session. Name is a string containing the name of the database to test. If the database is already open, ValidateName raises the EDatabaseError exception. If the database is not open, the procedure returns, and the application continues processing.

Note: Most applications should not need to call this method directly. It is called automatically each time a database is opened.

See Also

DatabaseName (see page 9980)

Open (see page 6866)

EDatabaseError (see page 6731)

53.1.7.5 TDatabase Properties

The properties of the TDatabase class are listed here.

Public Properties

	Name	Description
R	DataSets (see page 9981)	Provides an indexed array of all active datasets for a database component.
	Directory (see page 9981)	Specifies the working directory for a Paradox or dBASE database.
	Handle (see page 9983)	Specifies the Borland (see page 1761) Database Engine (BDE) database handle.
R	InTransaction (see page 9983)	Indicates whether a database transaction is in progress or not.
R	IsSQLBased (see page 9984)	Indicates if a database component is using either the Borland (see page 1761) Database Engine (BDE) SQL Links driver or the BDE ODBC socket.
R	Locale (see page 9985)	Identifies the Borland (see page 1761) Database Engine (BDE) language driver for the database component.
R	Session (see page 9988)	Points to the session component with which this database component is associated.
R	SessionAlias (see page 9988)	Specifies whether a database component is using a session alias.
	Temporary (see page 9989)	Indicates whether a database component is a temporary one, created by the system as needed, or a persistent one explicitly created, managed, and freed within the application.
	TraceFlags (see page 9990)	Specifies the database operations to track with the SQL Monitor at runtime.

Published Properties

	Name	Description
	AliasName (see page 9979)	Specifies the Borland (see page 1761) Database Engine (BDE) alias used by this database connection.
	Connected (see page 9979)	Indicates whether a database connection is active.
	DatabaseName (see page 9980)	Specifies the name of the database to associate with this database component.
	DriverName (see page 9981)	Specifies the name of the BDE driver for the database.
	Exclusive (see page 9982)	Enables an application to gain sole access to a database.
	HandleShared (see page 9983)	Specifies whether to share a database handle.
	KeepConnection (see page 9984)	Specifies whether an application remains connected to a database even if no datasets are open.
	LoginPrompt (see page 9985)	Specifies whether a login dialog appears immediately before opening a new connection.
	Params (see page 9986)	Contains database connection parameters for the Borland (see page 1761) Database Engine (BDE) alias associated with the database component.
	ReadOnly (see page 9987)	Specifies that the database connection provides read-only access.
	SessionName (see page 9989)	Identifies the name of the session used by this database component.
	TransIsolation (see page 9991)	Specifies the transaction isolation level for transactions managed by the Borland (see page 1761) Database Engine (BDE).

53.1.7.5.1 TDatabase.AliasName Property

Specifies the Borland (see page 1761) Database Engine (BDE) alias used by this database connection.

Pascal

```
property AliasName: string;
```

C++

```
__property AnsiString AliasName;
```

Description

Use AliasName to specify the name of an existing BDE alias for the database component. A BDE alias contains database configuration information for connection to a specific kind of database, such as InterBase, dBASE, or Paradox. The configuration information associated with a BDE alias differs by database type.

If an application sets AliasName, any value previously assigned to the DriverName (see page 9981) property is cleared because the alias automatically specifies a driver name as one of its parameters. Other alias connection information is stored in the Params (see page 9986) property when AliasName is set.

Note: Attempting to set AliasName when the Connected (see page 9979) property is true raises an exception.

Note: An application can specify the DatabaseName (see page 9980), DriverName (see page 9981), and Params (see page 9986) properties rather than setting AliasName if the desired BDE alias configuration does not already exist. An application can also provide an application-specific "alias" name by setting DatabaseName (see page 9980) when AliasName is already set.

Tip: At design time double-click a TDatabase (see page 9964) component to invoke the Database editor and set the AliasName.

See Also

Connected (see page 9979)

DatabaseName (see page 9980)

DriverName (see page 9981)

Params (see page 9986)

Identifying the Database

53.1.7.5.2 TDatabase.Connected Property

Indicates whether a database connection is active.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Set Connected to true to establish a database connection without opening a dataset. Set Connected to false to close a database connection. An application can check Connected to determine the current status of a database connection. If Connected is true,

the database connection is active; if false, and the KeepConnection (see page 9984) property is also false, then the connection is inactive.

Note: Set KeepConnection (see page 9984) to true to avoid having to log in to the server each time a database connection is reopened.

See Also

DataSets (see page 9981)

KeepConnection (see page 9984)

Connecting to Databases with TDatabase

53.1.7.5.3 TDatabase.DatabaseName Property

Specifies the name of the database to associate with this database component.

Pascal

```
property DatabaseName: string;
```

C++

```
__property AnsiString DatabaseName;
```

Description

Use DatabaseName to specify the name of the database to use with a database component. If DatabaseName is the same as an existing Borland (see page 1761) Database Engine (BDE) alias, then the AliasName (see page 9979) and DriverName (see page 9981) properties need not be set. If DatabaseName does not match an existing BDE alias, then either the application must also supply a valid alias in the AliasName (see page 9979) property in addition to the DatabaseName, or it must supply the DriverName (see page 9981) and Params (see page 9986) properties.

DatabaseName can be a fully qualified path name when connecting to a Paradox or dBASE database.

Note: Attempting to set DatabaseName when the Connected (see page 9979) property is true raises an exception.

Tip: At design time double-click a TDatabase (see page 9964) component to invoke the Database editor and set the DatabaseName.

See Also

AliasName (see page 9979)

Connected (see page 9979)

DriverName (see page 9981)

Params (see page 9986)

Identifying the Database

Connecting to Databases with TDatabase

Using TQuery

Using TStoredProc

Using TTable

53.1.7.5.4 TDatabase.DataSets Property

Provides an indexed array of all active datasets for a database component.

Pascal

```
property DataSets [Index: Integer]: TDBDataSet;
```

C++

```
__property TDBDataSet DataSets[int Index];
```

Description

Use DataSets to access active datasets associated with a database component. An active dataset is one that is currently open.

Note: If a TBDEClientDataSet uses the connection component to connect to a database server, DataSets lists the internal dataset that the client dataset creates to access data, not the client dataset itself.

See Also

DataSetCount (see page 6869)

TBDEClientDataSet

53.1.7.5.5 TDatabase.Directory Property

Specifies the working directory for a Paradox or dBASE database.

Pascal

```
property Directory: string;
```

C++

```
__property AnsiString Directory;
```

Description

Use Directory at runtime to change or retrieve the working directory for a Paradox or dBASE database. This is the location these databases use to create temporary tables at runtime.

Ordinarily a directory location is specified in the PATH parameter of the database alias (specified by the AliasName (see page 9979) property). At runtime, however, some applications may need to change the working directory after a database is opened.

To change the value of Directory at runtime, a database must already be open, or an exception is raised. Clearing the current value of Directory sets its value to the original working directory specified when the database was first opened.

Note: Do not use Directory when connected to remote databases. An exception is raised in these circumstances. An exception is also raised if an application accesses Directory when the database is closed.

See Also

AliasName (see page 9979)

DatabaseName (see page 9980)

Open (see page 6866)

53.1.7.5.6 TDatabase.DriverName Property

Specifies the name of the BDE driver for the database.

Pascal

```
property DriverName: string;
```

C++

```
__property AnsiString DriverName;
```

Description

Use DriverName to specify the name of the BDE driver to use for databases connections that do not specify an AliasName (see page 9979), but that do specify a DatabaseName (see page 9980). DriverName must be a valid BDE driver (for instance, STANDARD, ORACLE, SYBASE, or INTERBASE).

Note: If an application sets DriverName, it must also specify connection parameters in the Params (see page 9986) property. Ordinarily the alias specified in the AliasName (see page 9979) property supplies connection parameters, but when DriverName is set, the AliasName (see page 9979) property is automatically cleared to avoid BDE driver name contention.

Attempting to set DriverName when the Connected (see page 9979) property is true raises an exception.

Tip: At design time double-click a TDatabase (see page 9964) component to invoke the Database editor and set the DriverName.

See Also

AliasName (see page 9979)

Connected (see page 9979)

DatabaseName (see page 9980)

Params (see page 9986)

Identifying the Database

53.1.7.5.7 TDatabase.Exclusive Property

Enables an application to gain sole access to a database.

Pascal

```
property Exclusive: Boolean;
```

C++

```
__property Boolean Exclusive;
```

Description

Use Exclusive to prevent other applications from accessing a Paradox or dBASE database while this application is using it. Before opening the database, set Exclusive to true. The default value for Exclusive is false, allowing other applications to use the opened database.

When Exclusive is true and the application successfully opens the database, no other application can access it. If the database for which the application has requested exclusive access is already in use by another application, an exception is raised. To handle such exceptions, write an exception handler. All tables in the database are effectively opened exclusively, as other applications would be prevented from opening the database and the tables would be inaccessible.

A table database must be closed before changing the setting of the Exclusive property. Do not set Exclusive to true at design time if you also intend to set the Connected (see page 9979) property to true at design time. In this case an exception is raised because the database is already in use by the IDE.

Note: An application can attempt to set Exclusive to true for SQL databases, but some SQL servers do not support exclusive database-locking. Others may grant an exclusive read/write lock, but still permit read-only access to other applications. For more

information about exclusive locking of databases for a specific server, see the server's documentation.

See Also

Connected (see page 9979)

53.1.7.5.8 TDatabase.Handle Property

Specifies the Borland (see page 1761) Database Engine (BDE) database handle.

Pascal

```
property Handle: HDBIDB;
```

C++

```
__property HDBIDB Handle;
```

Description

Use Handle only to bypass TDatabase (see page 9964) methods and make direct calls to the directly to the BDE API. Many BDE function calls require a handle parameter. Handle is assigned an initial value when a database is opened.

See Also

IsSQLBased (see page 9984)

Locale (see page 9985)

53.1.7.5.9 TDatabase.HandleShared Property

Specifies whether to share a database handle.

Pascal

```
property HandleShared: Boolean;
```

C++

```
__property Boolean HandleShared;
```

Description

Use HandleShared to indicate that a database component can share its BDE handle in a session component. Set HandleShared to true to avoid namespace conflicts for database components that appear in a remote data module, or that appear in data modules you inherit from the Object Repository.

See Also

Handle (see page 9983)

Using Database Components in Data Modules

53.1.7.5.10 TDatabase.InTransaction Property

Indicates whether a database transaction is in progress or not.

Pascal

```
property InTransaction: Boolean;
```

C++

```
__property Boolean InTransaction;
```

Description

Examine InTransaction at run-time to determine if a database transaction is currently in progress. InTransaction is true if a transaction is in progress, false otherwise.

The value of InTransaction cannot be changed directly. Calling StartTransaction (see page 9977) sets InTransaction to true. Calling Commit (see page 9971) or Rollback (see page 9976) sets InTransaction to false.

See Also

Commit (see page 9971)

Rollback (see page 9976)

StartTransaction (see page 9977)

53.1.7.5.11 TDatabase.IsSQLBased Property

Indicates if a database component is using either the Borland (see page 1761) Database Engine (BDE) SQL Links driver or the BDE ODBC socket.

Pascal

```
property IsSQLBased: Boolean;
```

C++

```
__property Boolean IsSQLBased;
```

Description

Examine IsSQLBased at runtime to determine if a database connection is using a BDE SQL Links driver, the ODBC socket, or the Paradox, dBASE, and ASCII text file STANDARD driver. If IsSQLBased is true, the connection is using an SQL Links driver or is using the ODBC socket to communicate with a third-party ODBC driver. If the connection uses the STANDARD driver, IsSQLBased is false.

See Also

Handle (see page 9983)

Locale (see page 9985)

53.1.7.5.12 TDatabase.KeepConnection Property

Specifies whether an application remains connected to a database even if no datasets are open.

Pascal

```
property KeepConnection: Boolean;
```

C++

```
__property Boolean KeepConnection;
```

Description

Use KeepConnection to specify whether an application remains connected to a database even if no datasets are currently open. When KeepConnection is true (the default) the connection is maintained. For connections to remote database servers, or for applications that frequently open and close datasets, set KeepConnection to true to reduce network traffic, speed up applications, and avoid logging in to the server each time the connection is reestablished.

When KeepConnection is false a connection is dropped when there are no open datasets. Dropping a connection releases system resources allocated to the connection, but if a dataset is later opened that uses the database, the connection must be

reestablished and initialized.

Note: The KeepConnection setting for temporary database components created automatically as needed is determined by the KeepConnections property of TSession (see page 10093).

See Also

Close (see page 6863)

Open (see page 6866)

KeepConnections (see page 10124)

Identifying the Database

53.1.7.5.13 TDatabase.Locale Property

Identifies the Borland (see page 1761) Database Engine (BDE) language driver for the database component.

Pascal

```
property Locale: TLocale;
```

C++

```
__property TLocale Locale;
```

Description

Examine Locale to determine the BDE language driver used by the database component. Applications that make direct calls to the BDE may also need to pass Locale information as an API function parameter.

Language driver information may also be available as one of the values in the Params (see page 9986) property when a database component is using a BDE alias.

See Also

AliasName (see page 9979)

Handle (see page 9983)

IsSQLBased (see page 9984)

Params (see page 9986)

53.1.7.5.14 TDatabase.LoginPrompt Property

Specifies whether a login dialog appears immediately before opening a new connection.

Pascal

```
property LoginPrompt: Boolean;
```

C++

```
__property Boolean LoginPrompt;
```

Description

Set LoginPrompt to true to provide login support when establishing a connection. LoginPrompt controls two things: the occurrence of the OnLogin event, and the appearance of a default login dialog that prompts users for a name and password when you include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi). When the OnLogin event occurs and when the default login dialog appears depend on the type of connection component:

For TDatabase, TIBDatabase, and TSQLConnection, the dialog appears after the BeforeConnect event and before the

AfterConnect event, unless you supply an OnLogin event handler. If there is an OnLogin event handler, that event occurs in place of the login dialog, and there is no need to include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi). If correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler, the connection fails. The OnLogin event does not fire unless LoginPrompt is set to true.

For TADOConnection components, the dialog appears after the OnWillConnect event and before the BeforeConnect event. If there is an OnLogin event handler, that event occurs after the login dialog. If you do not include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi), the OnLogin event occurs but there is no default login dialog. If correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler, the connection fails. The OnLogin event does not fire unless LoginPrompt is set to true.

For DataSnap SOAP connection components, the dialog appears after the OnGetUsername event and before the BeforeConnect, AfterConnect, and OnLogin events. If the user cancels from the login dialog, no attempt is made to open a connection.

When LoginPrompt is false, the application must supply user name and password values programmatically:

For TDatabase, the user name and password can be supplied as USER NAME and PASSWORD parameters in the Params property.

For TADOConnection, the user name and password can be supplied as the ConnectionString property.

For TSQLConnection, the user name, password, and database can be supplied as UserName, Password, and Database parameters in the Params property, or provided as connection parameters associated with the connection name.

For DataSnap SOAP connection components, there is no built-in use for the user name and password supplied by the login dialog. (The UserName and Password properties of TWebConnection are unrelated).

Warning: Storing hard-coded user name and password entries as property values or in code for an OnLogin event handler can compromise server security.

See Also

OnLogin (see page 6873)

BeforeConnect (see page 6875)

AfterConnect (see page 6874)

OnGetUserName (see page 8533)

OnLogin (see page 20259)

OnLogin (see page 9994)

Controlling Server Login

53.1.7.5.15 TDatabase.Params Property

Contains database connection parameters for the Borland (see page 1761) Database Engine (BDE) alias associated with the database component.

Pascal

```
property Params: TStrings;
```

C++

```
__property TStrings Params;
```

Description

Use Params to examine or specify database connection parameters, such as path name, server name, schema-caching size,

language driver, user name, and password. Actual parameters in Params differ depending on the current BDE alias and driver for the database component.

Params is a list of string items, each representing a different database connection parameter. If the AliasName (see page 9979) property specifies a valid BDE alias, then Params automatically contains the parameters defined for that alias. If, instead of providing an alias, an application uses the DriverName (see page 9981) property to provide a local alias, the application must provide Params values directly.

Note: Applications can use Params to override the default settings supplied by a BDE alias.

Tip: At design time double-click a TDatabase (see page 9964) component to invoke the Database editor and set Params.

See Also

AliasName (see page 9979)

DatabaseName (see page 9980)

LoginPrompt (see page 6872)

OnLogin (see page 9994)

Identifying the Database

Setting BDE Alias Parameters

53.1.7.5.16 TDatabase.ReadOnly Property

Specifies that the database connection provides read-only access.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to specify whether the database connection should allow the application to update the tables and other metadata in the database. Set ReadOnly before opening the database.

When ReadOnly is false (the default), the application can modify tables and database metadata (like indexes). When ReadOnly is true, applications can browse tables but cannot update them. The application is also prevented from creating or deleting metadata objects like tables and indexes.

Note: A ReadOnly property value of true should be used when accessing a dBASE or Paradox database on a read-only storage medium, such as a CD.

Note: For SQL databases, if ReadOnly is true, you must also explicitly set the ReadOnly property to true for dataset components like TTable (see page 10168) and TQuery (see page 10059).

See Also

CanModify (see page 7012)

53.1.7.5.17 TDatabase.Session Property

Points to the session component with which this database component is associated.

Pascal

```
property Session: TSession;
```

C++

```
__property TSession Session;
```

Description

Use Session to determine the session component that controls the database component. By default, a database component is associated with the default session component, Session, that is automatically created for all database applications. To assign a database component to a different session in a multi-threaded application, specify the name of a different session component in the SessionName (see page 9989) property.

See Also

SessionAlias (see page 9988)

SessionName (see page 9989)

Associating a Database Component with a Session

Connecting to Databases with TDatabase

53.1.7.5.18 TDatabase.SessionAlias Property

Specifies whether a database component is using a session alias.

Pascal

```
property SessionAlias: Boolean;
```

C++

```
__property Boolean SessionAlias;
```

Description

Examine SessionAlias to see if a database component is using a session alias. If SessionAlias is true, a session alias is in use. If false, a session alias is not in use.

A session alias is automatically created for a database component if it

Uses the DriverName (see page 9981) and Params (see page 9986) properties instead of the AliasName (see page 9979) property, and the Params (see page 9986) property contains any parameters.

Overrides Params (see page 9986) values supplied by a BDE alias in AliasName (see page 9979).

A session alias may also be created if an application permits the user to log into a server using a different user name than the one specified in the Params (see page 9986) property.

A session alias exists only while the application is running. It is not stored in the BDE configuration file, and cannot be accessed from outside the application.

See Also

AliasName (see page 9979)

DriverName (see page 9981)

IsSQLBased (see page 9984)

Params (see page 9986)

Session (see page 9988)

SessionName (see page 9989)

53.1.7.5.19 TDatabase.SessionName Property

Identifies the name of the session used by this database component.

Pascal

```
property SessionName: string;
```

C++

```
__property AnsiString SessionName;
```

Description

Use SessionName to specify the session with which a database component is associated. If SessionName is blank, a database component is automatically associated with the default session, Session (see page 9988).

To associate a database component with a different session in a database application SessionName must matches the SessionName property of an existing session component.

See Also

Session (see page 9988)

SessionAlias (see page 9988)

TSession (see page 10093)

TSessionList (see page 10129)

SessionName (see page 10126)

Associating a Database Component with a Session

53.1.7.5.20 TDatabase.Temporary Property

Indicates whether a database component is a temporary one, created by the system as needed, or a persistent one explicitly created, managed, and freed within the application.

Pascal

```
property Temporary: Boolean;
```

C++

```
__property Boolean Temporary;
```

Description

Examine Temporary to determine if a database component is a temporary one, created by the system as needed, or a persistent one explicitly created, managed, and freed within the application. Temporary is true if the system created the component, false otherwise.

A temporary database component is created when a dataset is opened and the dataset is not already associated with an existing database component. If Temporary remains true, then a temporary database component is freed when the dataset is closed. An application can prevent the destruction of a temporary database component by setting Temporary to false while the dataset is active, but the application is then responsible for closing the database when it is no longer needed.

See Also

Create (see page 9969)

Free (see page 22300)

53.1.7.5.21 TDatabase.TraceFlags Property

Specifies the database operations to track with the SQL Monitor at runtime.

Pascal

```
property TraceFlags: TTraceFlags;
```

C++

```
__property TTraceFlags TraceFlags;
```

Description

Use TraceFlags to specify which database operations the SQL Monitor should track in an application at runtime. TraceFlags is only meaningful for the SQL Monitor, which is provided to enable performance tuning and SQL debugging when working with remote SQL database servers.

Note: Normally trace options are set from the SQL Monitor rather than setting TraceFlags in application code.

The value of a session component's TraceFlags property determines the initial settings of the TraceFlags property for database components associated with the session.

The TTraceFlags (see page 10263) type defines the individual values that can be included in the TraceFlags property. The following table summarizes those values:

Values	Meaning
tfQPrepare	Monitor Prepare statements.
tfQExecute	Monitor ExecSQL statements.
tfError	Monitor server error messages. Such messages may include an error code.
tfStmt	Monitor all SQL statements.
tfConnect	Monitor database connect and disconnect operations, including allocation of connection handles, and freeing connection handles.
tfTransact	Monitor transaction statements, such as StartTransaction (see page 9977), Commit (see page 9971), and Rollback (see page 9976).
tfBlob	Monitor operations on blob data types.
tfMisc	Monitor any statements not covered by other flag options.
tfVendor	Monitor direct API function calls to the database server.
tfDataIn	Monitor data received from a server.
tfDataOut	Monitor data sent to a server.

Because TraceFlags is a set property, an application can specify different combinations of flags to monitor different combinations of statements. For example, the following statement limits monitoring to database connections and SQL statement preparation:

```
TraceFlags := [tfConnect, tfQPrepare];
TraceFlags  << tfConnect << tfQPrepare;
```

See Also

Session (see page 9988)

SessionName (see page 9989)

TraceFlags (see page 10127)

53.1.7.5.22 TDatabase.TransIsolation Property

Specifies the transaction isolation level for transactions managed by the Borland (see page 1761) Database Engine (BDE).

Pascal

```
property TransIsolation: TTransIsolation;
```

C++

```
__property TTransIsolation TransIsolation;
```

Description

Use TransIsolation to specify the transaction isolation level for database transactions managed by the BDE. Transaction isolation level determines how a transaction interacts with other simultaneous transactions when they work with the same tables, and how much a transaction sees of the work performed by other transactions.

Note: Applications that use passthrough SQL for handling transactions must pass a transaction isolation level directly to the database server using the appropriate SQL statement.

TransIsolation can be any one of the three values summarized in the following table:

Isolation level	Meaning
tiDirtyRead	Permits reading of uncommitted changes made to the database by other simultaneous transactions. Uncommitted changes are not permanent, and might be rolled back (undone) at any time. At this level a transaction is least isolated from the effects of other transactions.
tiReadCommitted	Permits reading of committed (permanent) changes made to the database by other simultaneous transactions. This is the default TransIsolation property value.
tiRepeatableRead	Permits a single, one-time reading of the database. The transaction cannot see any subsequent changes made by other simultaneous transactions. This isolation level guarantees that once a transaction reads a record, its view of that record does not change unless it makes a modification to the record itself. At this level, a transaction is most isolated from other transactions.

Different database servers support different levels of transaction isolation. If an application sets TransIsolation to an unsupported level for a remote SQL server, the BDE uses the next highest level supported by that server. The following table summarizes the isolation levels supported by the servers recognized by the BDE:

ServerSpecified LevelActual Level

Server	Specified Level	Actual Level
Oracle	tiDirtyRead | tiReadCommitted | tiRepeatableRead	tiReadCommitted | tiReadCommitted | tiRepeatableRead (READONLY)
Sybase, MS-SQL	tiDirtyRead | tiReadCommitted | tiRepeatableRead	tiReadCommitted | tiReadCommitted | Not supported
DB2	tiDirtyRead | tiReadCommitted | tiRepeatableRead	tiDirtyRead | tiReadCommitted | tiRepeatableRead
Informix	tiDirtyRead | tiReadCommitted | tiRepeatableRead	tiDirtyRead | tiReadCommitted | tiRepeatableRead
InterBase	tiDirtyRead | tiReadCommitted | tiRepeatableRead	tiReadCommitted | tiReadCommitted | tiRepeatableRead

Paradox, dBASE, Access, FoxPro	tiDirtyRead | tiReadCommitted | tiRepeatableRead	tiDirtyRead | Not supported | Not supported

Note: For local transactions against Paradox and dBASE, TransIsolation must be set to tiDirtyRead. Otherwise, an exception is raised.

If an application uses ODBC to interface with a server, the ODBC driver must also support the isolation level. For more information about supported isolation levels, see the documentation for the ODBC driver in question.

See Also

AliasName (see page 9979)

Commit (see page 9971)

DriverName (see page 9981)

Params (see page 9986)

Rollback (see page 9976)

StartTransaction (see page 9977)

53.1.7.6 TDatabase Events

The events of the TDatabase class are listed here.

Published Events

	Name	Description
	AfterConnect (see page 9992)	Occurs after a connection is established.
	AfterDisconnect (see page 9993)	Occurs after the connection closes.
	BeforeConnect (see page 9993)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 9993)	Occurs immediately before the connection closes.
	OnLogin (see page 9994)	Occurs when an application connects to a database.

53.1.7.6.1 TDatabase.AfterConnect Event

Occurs after a connection is established.

Pascal

```
property AfterConnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterConnect;
```

Description

Write an AfterConnect event handler to take application-specific actions immediately after the connection component opens a connection to the remote source of database information.

See Also

BeforeConnect (see page 6875)

Connected (see page 6868)

Connecting to a Database Server

53.1.7.6.2 TDatabase.AfterDisconnect Event

Occurs after the connection closes.

Pascal

```
property AfterDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterDisconnect;
```

Description

Write an AfterDisconnect event handler to take application-specific actions after the connection component drops a connection.

See Also

BeforeDisconnect (see page 6875)

Connected (see page 6868)

Disconnecting from a Database Server

53.1.7.6.3 TDatabase.BeforeConnect Event

Occurs immediately before establishing a connection.

Pascal

```
property BeforeConnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeConnect;
```

Description

Write a BeforeConnect event handler to take application-specific actions before the connection component opens a connection to the remote source of database information.

See Also

AfterConnect (see page 6874)

Connected (see page 6868)

Connecting to a Database Server

53.1.7.6.4 TDatabase.BeforeDisconnect Event

Occurs immediately before the connection closes.

Pascal

```
property BeforeDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeDisconnect;
```

Description

Write a BeforeDisconnect event handler to take application-specific actions before dropping a connection.

Delphi Examples:

```
{
This BeforeDisconnect event handler makes sure that the
client dataset has applied all its pending updates before
terminating the connection to the server.
}
procedure TForm1.RemoteServer1BeforeDisconnect(Sender: TObject);
begin
  if (ClientDataSet1.ChangeCount > 0) then
    ClientDataSet1.ApplyUpdates(-1);
end;
```

See Also

AfterDisconnect (see page 6875)

Connected (see page 6868)

Disconnecting from a Database Server

53.1.7.6.5 TDatabase.OnLogin Event

Occurs when an application connects to a database.

Pascal

```
property OnLogin: TDatabaseLoginEvent;
```

C++

```
__property TDatabaseLoginEvent OnLogin;
```

Description

Write an OnLogin event handler to take specific actions when an application attempts to connect to a database. When the LoginPrompt (see page 9985) property is true, a database login is required.

If there is no OnLogin event handler, the current UserName is read from the Params (see page 9986) property, and a standard Login dialog box opens. The dialog prompts for a user name and password combination, and then uses the values entered by the user to set the UserName and PASSWORD values in the Params (see page 9986) property. These values are then passed to the remote server.

Applications that provide alternative OnLogin event handlers must set the UserName and PASSWORD values in LoginParams. LoginParams is a temporary string list and is freed automatically when no longer needed.

Note: The login dialog and OnLogin event only occur when the LoginPrompt (see page 9985) property is true.

See Also

Connected (see page 9979)

KeepConnection (see page 9984)

Params (see page 9986)

Open (see page 6866)

53.1.8 TDataSetUpdateObject Class

TDataSetUpdateObject is a base class for update objects used to update otherwise unupdatable queries or stored procedures in applications that cache updates.

Class Hierarchy

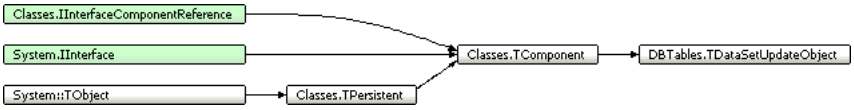

Pascal

```
TDataSetUpdateObject = class(TComponent);
```

C++

```
class TDataSetUpdateObject : public TComponent;
```

File

DBTables

Description

Use TDataSetUpdateObject as a base class when creating customized update objects that update BDE datasets.

See Also

TUpdateSQL (see page 10229)

53.1.8.1 TDataSetUpdateObject Members

The following tables list the members exposed by TDataSetUpdateObject.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
A	GetDataSet (see page 9998)	GetDataSet returns the value of the DataSet (see page 9999) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.

	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 9999)	SetDataSet sets the value of the DataSet (see page 9999) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DataSet (see page 9999)	Specifies the dataset that stores the cached updates to be applied by the update object.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Apply (see page 9998)	Apply applies changes to the dataset specified by the DataSet (see page 9999) property.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.

	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

53.1.8.2 TDataSetUpdateObject Methods

The methods of the TDataSetUpdateObject class are listed here.

Protected Methods

	Name	Description
	GetDataSet (see page 9998)	GetDataSet returns the value of the DataSet (see page 9999) property.
	SetDataSet (see page 9999)	SetDataSet sets the value of the DataSet (see page 9999) property.

Public Methods

	Name	Description
	Apply (see page 9998)	Apply applies changes to the dataset specified by the DataSet (see page 9999) property.

53.1.8.2.1 TDataSetUpdateObject.Apply Method

Apply applies changes to the dataset specified by the DataSet (see page 9999) property.

Pascal

```
procedure Apply(UpdateKind: TUpdateKind); virtual; abstract;
```

C++

```
virtual __fastcall Apply(TUpdateKind UpdateKind) = 0;
```

Description

Descendants of TDataSetUpdateObject (see page 9994) implement the Apply method. This method is intended to perform the updates specified by the update object. Descendants introduce properties to describe the details of the updates that should be performed. The UpdateKind parameter indicates whether the update object should modify existing records, insert new records, or delete existing records.

See Also

DataSet (see page 9999)

53.1.8.2.2 TDataSetUpdateObject.GetDataSet Method

GetDataSet returns the value of the DataSet (see page 9999) property.

Pascal

```
function GetDataSet: TDataSet; virtual; abstract;
```

C++

```
virtual __fastcall TDataSet GetDataSet() = 0;
```

Description

Override GetDataSet, along with the SetDataSet (see page 9999) method, to provide an implementation of the DataSet (see page 9999) property. The DataSet (see page 9999) property should represent the dataset that caches the updates to be

applied by the update object.

See Also

DataSet (see page 9999)

SetDataSet (see page 9999)

53.1.8.2.3 TDataSetUpdateObject.SetDataSet Method

SetDataSet sets the value of the DataSet (see page 9999) property.

Pascal

```
procedure SetDataSet(ADataSet: TDataSet); virtual; abstract;
```

C++

```
virtual __fastcall SetDataSet(TDataSet ADataSet) = 0;
```

Description

Override SetDataSet, along with the GetDataSet (see page 9998) method, to provide an implementation of the DataSet (see page 9999) property. The DataSet (see page 9999) property should represent the dataset that caches the updates to be applied by the update object.

See Also

DataSet (see page 9999)

GetDataSet (see page 9998)

53.1.8.3 TDataSetUpdateObject Properties

The properties of the TDataSetUpdateObject class are listed here.

Protected Properties

	Name	Description
	DataSet (see page 9999)	Specifies the dataset that stores the cached updates to be applied by the update object.

53.1.8.3.1 TDataSetUpdateObject.DataSet Property

Specifies the dataset that stores the cached updates to be applied by the update object.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

DataSet is the dataset that stores the updates that this update object applies to a database server. The update object uses the dataset to obtain information about the original and updated field values.

See Also

GetDataSet (see page 9998)

SetDataSet (see page 9999)

Using TSimpleDataSet

Using TSQLQuery

Using TSQLStoredProc

Using TSQLTable

53.1.9 TDBDataSet Class

TDBDataSet encapsulates database connectivity for descendant dataset objects.

Class Hierarchy

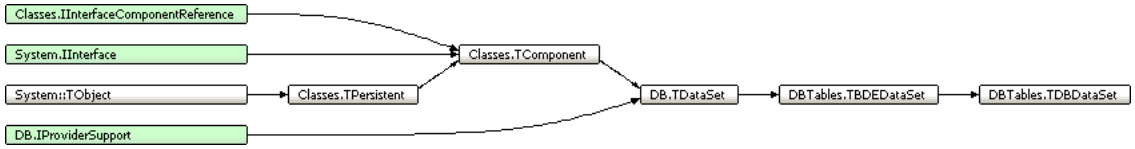

Pascal

```
TDBDataSet = class(TBDEDataSet);
```

C++

```
class TDBDataSet : public TBDEDataSet;
```

File

DBTables

Description

TDBDataSet defines database-related connectivity properties and methods for a dataset. Applications never use TDBDataSet objects directly. Instead they use the descendants of TDBDataSet, such as TQuery (see page 10059), TStoredProc (see page 10139), and TTable (see page 10168), which inherit its database-related properties and methods.

Developers who create custom dataset components may want to derive them from TDBDataSet to inherit to the database-specific properties of TDBDataSet in addition to all the functionality of TDataSet and the BDE-related properties and methods of TBDEDataSet (see page 9878). In this case, developers should examine the source code to study the protected methods of TDBDataSet that are not documented for this object.

See Also

TDataSet (see page 6894)

TBDEDataSet (see page 9878)

TQuery (see page 10059)

TStoredProc (see page 10139)

TTable (see page 10168)

53.1.9.1 TDBDataSet Members

The following tables list the members exposed by TDBDataSet.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	ActivateFilters (see page 9893)	This is ActivateFilters, a member of class TBDEDataSet.
	AddFieldDesc (see page 9893)	This is AddFieldDesc, a member of class TBDEDataSet.
	AllocCachedUpdateBuffers (see page 9893)	This is AllocCachedUpdateBuffers, a member of class TBDEDataSet.
	AllocKeyBuffers (see page 9893)	This is AllocKeyBuffers, a member of class TBDEDataSet.
V	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
V	AllocRecordBuffer (see page 9894)	This is AllocRecordBuffer, a member of class TBDEDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
V	BlockReadNext (see page 9894)	This is BlockReadNext, a member of class TBDEDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
	CachedUpdateCallBack (see page 9895)	This is CachedUpdateCallBack, a member of class TBDEDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
	CheckCachedUpdateMode (see page 9896)	This is CheckCachedUpdateMode, a member of class TBDEDataSet.
V	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
V	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
	CheckSetKeyMode (see page 9896)	This is CheckSetKeyMode, a member of class TBDEDataSet.
V	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
V	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
V	ClearCalcFields (see page 9897)	This is ClearCalcFields, a member of class TBDEDataSet.
V	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
V	CloseBlob (see page 9897)	This is CloseBlob, a member of class TBDEDataSet.
V	CloseCursor (see page 10014)	This is CloseCursor, a member of class TDBDataSet.
	ConstraintsStored (see page 10015)	This is ConstraintsStored, a member of class TDBDataSet.
	CreateExprFilter (see page 9900)	This is CreateExprFilter, a member of class TBDEDataSet.
V	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
	CreateFuncFilter (see page 9900)	This is CreateFuncFilter, a member of class TBDEDataSet.
V	CreateHandle (see page 9900)	This is CreateHandle, a member of class TBDEDataSet.
	CreateLookupFilter (see page 9901)	This is CreateLookupFilter, a member of class TBDEDataSet.
V	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
V	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
V	DataEvent (see page 6924)	This is DataEvent, a member of class TDataSet.
V	DataEvent (see page 9901)	This is DataEvent, a member of class TBDEDataSet.
	DeactivateFilters (see page 9901)	This is DeactivateFilters, a member of class TBDEDataSet.
V	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
V	DestroyHandle (see page 9901)	This is DestroyHandle, a member of class TBDEDataSet.
V	DestroyLookupCursor (see page 9902)	This is DestroyLookupCursor, a member of class TBDEDataSet.

	Disconnect (see page 10015)	This is Disconnect, a member of class TDBDataSet.
	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FindRecord (see page 9903)	This is FindRecord, a member of class TBDEDataSet.
	ForceUpdateCallback (see page 9904)	This is ForceUpdateCallback, a member of class TBDEDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeKeyBuffers (see page 9904)	This is FreeKeyBuffers, a member of class TBDEDataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
	FreeRecordBuffer (see page 9904)	This is FreeRecordBuffer, a member of class TBDEDataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
	GetBookmarkData (see page 9905)	This is GetBookmarkData, a member of class TBDEDataSet.
	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
	GetBookmarkFlag (see page 9905)	This is GetBookmarkFlag, a member of class TBDEDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
	GetCanModify (see page 9905)	This is GetCanModify, a member of class TBDEDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetFieldFullName (see page 9907)	This is GetFieldFullName, a member of class TBDEDataSet.
	GetHandle (see page 9907)	This is GetHandle, a member of class TBDEDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIndexField (see page 9907)	This is GetIndexField, a member of class TBDEDataSet.
	GetIndexFieldCount (see page 9907)	This is GetIndexFieldCount, a member of class TBDEDataSet.
	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.

V	GetIsIndexField (see page 9908)	This is GetIsIndexField, a member of class TBDEDataSet.
	GetKeyBuffer (see page 9908)	This is GetKeyBuffer, a member of class TBDEDataSet.
	GetKeyExclusive (see page 9908)	This is GetKeyExclusive, a member of class TBDEDataSet.
	GetKeyFieldCount (see page 9909)	This is GetKeyFieldCount, a member of class TBDEDataSet.
V	GetLookupCursor (see page 9909)	This is GetLookupCursor, a member of class TBDEDataSet.
V	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
V	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetObjectTypeNames (see page 9909)	This is GetObjectTypeNames, a member of class TBDEDataSet.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
V	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
V	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
V	GetRecNo (see page 9909)	This is GetRecNo, a member of class TBDEDataSet.
A	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
V	GetRecord (see page 9910)	This is GetRecord, a member of class TBDEDataSet.
V	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
V	GetRecordCount (see page 9910)	This is GetRecordCount, a member of class TBDEDataSet.
V	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
V	GetRecordSize (see page 9910)	This is GetRecordSize, a member of class TBDEDataSet.
V	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
V	GetStateFieldValue (see page 9910)	This is GetStateFieldValue, a member of class TBDEDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetUpdateRecordSet (see page 9910)	This is GetUpdateRecordSet, a member of class TBDEDataSet.
	GetUpdatesPending (see page 9911)	This is GetUpdatesPending, a member of class TBDEDataSet.
V	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
	InitKeyBuffer (see page 9911)	This is InitKeyBuffer, a member of class TBDEDataSet.
V	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
V	InitRecord (see page 9911)	This is InitRecord, a member of class TBDEDataSet.
V	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
V	InternalAddRecord (see page 9911)	This is InternalAddRecord, a member of class TBDEDataSet.
V	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
V	InternalCancel (see page 9912)	This is InternalCancel, a member of class TBDEDataSet.
A	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.
V	InternalClose (see page 9912)	This is InternalClose, a member of class TBDEDataSet.
V	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
V	InternalDelete (see page 9912)	This is InternalDelete, a member of class TBDEDataSet.
V	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
V	InternalEdit (see page 9912)	This is InternalEdit, a member of class TBDEDataSet.
V	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
V	InternalFirst (see page 9912)	This is InternalFirst, a member of class TBDEDataSet.
V	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
V	InternalGotoBookmark (see page 9913)	This is InternalGotoBookmark, a member of class TBDEDataSet.
A	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
V	InternalHandleException (see page 9913)	This is InternalHandleException, a member of class TBDEDataSet.
A	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
V	InternalInitFieldDefs (see page 9913)	This is InternalInitFieldDefs, a member of class TBDEDataSet.
V	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
V	InternalInitRecord (see page 9913)	This is InternalInitRecord, a member of class TBDEDataSet.
V	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
V	InternalInsert (see page 9914)	This is InternalInsert, a member of class TBDEDataSet.
V	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.

	InternalLast (see page 9914)	This is InternalLast, a member of class TBDEDataSet.
	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
	InternalOpen (see page 9914)	This is InternalOpen, a member of class TBDEDataSet.
	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
	InternalPost (see page 9914)	This is InternalPost, a member of class TBDEDataSet.
	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
	InternalRefresh (see page 9914)	This is InternalRefresh, a member of class TBDEDataSet.
	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	InternalSetToRecord (see page 9915)	This is InternalSetToRecord, a member of class TBDEDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
	IsCursorOpen (see page 9915)	This is IsCursorOpen, a member of class TBDEDataSet.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 9916)	This is LocateRecord, a member of class TBDEDataSet.
	MapsToIndex (see page 9918)	This is MapsToIndex, a member of class TBDEDataSet.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	OpenCursor (see page 10015)	This is OpenCursor, a member of class TDBDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PostKeyBuffer (see page 9919)	This is PostKeyBuffer, a member of class TBDEDataSet.
	PrepareCursor (see page 9919)	This is PrepareCursor, a member of class TBDEDataSet.
	ProcessUpdates (see page 9919)	This is ProcessUpdates, a member of class TBDEDataSet.
	PSEndTransaction (see page 10016)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 10017)	Executes a specified SQL command.
	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 10017)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 6982)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandType (see page 6982)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 10018)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSGetUpdateException (see page 9920)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 10018)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 10018)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 9920)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 9921)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.

	PSStartTransaction (see page 10019)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 10019)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	ResetCursorRange (see page 9922)	This is ResetCursorRange, a member of class TBDEDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBlockReadSize (see page 9922)	This is SetBlockReadSize, a member of class TBDEDataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkData (see page 9922)	This is SetBookmarkData, a member of class TBDEDataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkFlag (see page 9923)	This is SetBookmarkFlag, a member of class TBDEDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetCachedUpdates (see page 9923)	This is SetCachedUpdates, a member of class TBDEDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetCursorRange (see page 9923)	This is SetCursorRange, a member of class TBDEDataSet.
	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDBFlag (see page 10020)	This is SetDBFlag, a member of class TDBDataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldData (see page 9923)	This is SetFieldData, a member of class TBDEDataSet.
	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
	SetFilterData (see page 9923)	This is SetFilterData, a member of class TBDEDataSet.
	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
	SetFiltered (see page 9924)	This is SetFiltered, a member of class TBDEDataSet.
	SetFilterHandle (see page 9924)	This is SetFilterHandle, a member of class TBDEDataSet.
	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
	SetFilterOptions (see page 9924)	This is SetFilterOptions, a member of class TBDEDataSet.
	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
	SetFilterText (see page 9924)	This is SetFilterText, a member of class TBDEDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetHandle (see page 10020)	This is SetHandle, a member of class TDBDataSet.
	SetIndexField (see page 9925)	This is SetIndexField, a member of class TBDEDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetKeyBuffer (see page 9925)	This is SetKeyBuffer, a member of class TBDEDataSet.
	SetKeyExclusive (see page 9925)	This is SetKeyExclusive, a member of class TBDEDataSet.
	SetKeyFieldCount (see page 9925)	This is SetKeyFieldCount, a member of class TBDEDataSet.
	SetKeyFields (see page 9925)	This is SetKeyFields, a member of class TBDEDataSet.
	SetLinkRanges (see page 9926)	This is SetLinkRanges, a member of class TBDEDataSet.
	SetLocale (see page 9926)	This is SetLocale, a member of class TBDEDataSet.
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetOnFilterRecord (see page 9926)	This is SetOnFilterRecord, a member of class TBDEDataSet.

	SetOnUpdateError (see page 9926)	This is SetOnUpdateError, a member of class TBDEDataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
V	SetRecNo (see page 9927)	This is SetRecNo, a member of class TBDEDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
V	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
V	SetStateFieldValue (see page 9927)	This is SetStateFieldValue, a member of class TBDEDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	SetupCallBack (see page 9927)	This is SetupCallBack, a member of class TBDEDataSet.
	SetUpdateObject (see page 9927)	This is SetUpdateObject, a member of class TBDEDataSet.
	SetUpdateRecordSet (see page 9927)	This is SetUpdateRecordSet, a member of class TBDEDataSet.
	SwitchToIndex (see page 9928)	This is SwitchToIndex, a member of class TBDEDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	UpdateCallbackRequired (see page 9928)	This is UpdateCallbackRequired, a member of class TBDEDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
R	DBFlags (see page 10023)	Keeps track of the commands this dataset has active with the database.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
	Reserved (see page 7035)	For internal use only.
	StmtHandle (see page 9935)	Specifies the Borland (see page 1761) Database Engine (BDE) statement handle for the dataset.
	UpdateMode (see page 10028)	Determines how the Borland (see page 1761) Database (see page 10021) Engine (BDE) finds records when updating to an SQL database.

Public Events

	Name	Description
	AfterCancel (see page 7038)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 7040)	Occurs after an application closes a dataset.
	AfterDelete (see page 7041)	Occurs after an application deletes a record.
	AfterEdit (see page 7044)	Occurs after an application starts editing a record.
	AfterInsert (see page 7044)	Occurs after an application inserts a new record.
	AfterOpen (see page 7045)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 7046)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 7047)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 7047)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 7048)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 7048)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 7049)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 7049)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 7050)	Occurs before an application enters insert mode.
	BeforeOpen (see page 7051)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 7052)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 7052)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 7053)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 7053)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 7054)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 7054)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnNewRecord (see page 7056)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 7056)	Occurs when an application attempts to modify or insert a record and an exception is raised.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
	ApplyUpdates (see page 9894)	Writes a dataset's pending cached updates to the database.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
V	BookmarkValid (see page 9895)	Tests the validity of a specified bookmark.
V	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
V	Cancel (see page 9895)	Cancels modifications to the current record if those changes are not yet posted.
	CancelUpdates (see page 9896)	Clears all pending cached updates from the cache and restores the dataset its prior state.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	CheckOpen (see page 10014)	Checks the result of a call to the Borland (see page 1761) Database (see page 10021) Engine (BDE).
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Close (see page 6920)	Closes a dataset.
	CloseDatabase (see page 10014)	Closes a database connection associated with the database session.
	CommitUpdates (see page 9897)	Clears the cached updates buffer.
V	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
V	CompareBookmarks (see page 9898)	Indicates the relationship between two bookmarks.
	ConstraintCallBack (see page 9898)	Specifies the callback function that allows the dataset to communicate with the BDE constraint layer.
	ConstraintsDisabled (see page 9899)	Indicates whether constraints are enabled for a dataset.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
V	Create (see page 10013)	Creates a new TDBDataSet (see page 10000) object.
V	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
V	CreateBlobStream (see page 9899)	Returns a TStream object for reading or writing the data in a specified blob field.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 6904)	Destroys the TDataSet (see page 6894) instance.
V	Destroy (see page 9890)	Destroys the instance of a dataset component.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableConstraints (see page 9902)	Disables application of server constraints on the dataset.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableConstraints (see page 9902)	Enables application of server constraints against dataset entries.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FetchAll (see page 9903)	Retrieves all records from the current cursor position to the end of the file and stores them locally.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	FlushBuffers (see page 9903)	Posts all changes that have been written to the record buffer.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
V	GetBlobFieldData (see page 9904)	Reads BLOB data into a buffer.
V	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
V	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
V	GetCurrentRecord (see page 9905)	Retrieves the current record into a buffer.
V	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
V	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
V	GetFieldData (see page 9906)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
V	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetIndexInfo (see page 9907)	Retrieves information about the current index into the index data fields of the dataset.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
V	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
V	IsSequenced (see page 9915)	IsSequenced indicates whether sequence numbers are available for database records.
	Last (see page 6968)	Moves to the last record in the dataset.
V	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Locate (see page 9915)	Searches the dataset for a specified record and makes that record the current record.
V	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
V	Lookup (see page 9916)	Retrieves field values from a record that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	Open (see page 6976)	Opens the dataset.
	OpenDatabase (see page 10016)	Opens the database that contains the dataset.
V	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
V	Post (see page 9918)	Writes a modified record to the database.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSGetCommandText (see page 7579)	Gets the SQL command to be executed by PSExecute (see page 7578).
	PSGetCommandType (see page 7580)	Gets the command type to be executed by PSExecute (see page 7578).
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7585)	Assigns a specified SQL command to be executed by PSExecute (see page 7578).
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
	RevertRecord (see page 9922)	Restores the current record in the dataset to an unmodified state when cached updates are enabled.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
V	Translate (see page 9928)	Converts a data string between the Microsoft ANSI character set, and the local code page (OEM character set).
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.
V	UpdateStatus (see page 9928)	Reports the update status for the current record.

Public Properties

	Name	Description
	Active (see page 7004)	Specifies whether or not a dataset is open.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.

	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
	CacheBlobs (see page 9932)	Determines whether BLOB fields are cached in memory.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Database (see page 10021)	Identifies the database component for which this dataset represents one or more tables.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
R	DataSource (see page 7014)	Represents the data source of another dataset that supplies values to the dataset.
R	DBHandle (see page 10023)	Specifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) database handle for the dataset.
R	DBLocale (see page 10024)	Identifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) language driver for the dataset component.
R	DBSession (see page 10024)	Identifies the database session component associated with this dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	ExpIndex (see page 9933)	Indicates whether a dataset is using a dBASE expression index.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
	FieldDefs (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
	Handle (see page 10027)	Specifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) cursor handle for the dataset.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	KeySize (see page 9934)	Specifies the size of the key for the current index of the dataset.
R	Locale (see page 9934)	Identifies the Borland (see page 1761) Database Engine (BDE) language driver for the dataset.
R	Modified (see page 7029)	Indicates whether the active record is modified.
	ObjectView (see page 7031)	Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	UpdateObject (see page 9936)	Specifies the update object component used to update a read-only result set when cached updates are enabled.
	UpdateRecordTypes (see page 9936)	Specifies the type of records visible in a dataset when cached updates are enabled.

R	UpdatesPending (see page 9937)	Indicates whether the cached updates buffer contains records that are not yet applied.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterCancel (see page 9938)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 9940)	Occurs after an application closes a dataset.
	AfterDelete (see page 9942)	Occurs after an application deletes a record.
	AfterEdit (see page 9944)	Occurs after an application starts editing a record.
	AfterInsert (see page 9945)	Occurs after an application inserts a new record.
	AfterOpen (see page 9946)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 9947)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 9947)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 9948)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 9948)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 9949)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 9949)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 9950)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 9950)	Occurs before an application enters insert mode.
	BeforeOpen (see page 9952)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 9952)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 9953)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 9953)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 9954)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 9955)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 9955)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 10028)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 9955)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 9956)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnUpdateError (see page 9956)	Occurs if an exception is generated when cached updates are applied to a database.
	OnUpdateRecord (see page 9957)	Occurs when cached updates are applied to a record.

Published Properties

	Name	Description
	Active (see page 9929)	Specifies whether or not a dataset is open.
	AutoCalcFields (see page 9931)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	AutoRefresh (see page 10021)	Specifies whether server-generated field values are refetched automatically.
	CachedUpdates (see page 9932)	Specifies whether cached updates are enabled for a dataset.
	DatabaseName (see page 10022)	Specifies the name of the database associated with this dataset.
	Filter (see page 10025)	Specifies the text of the current filter for a dataset.
	Filtered (see page 10026)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 10026)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectView (see page 9935)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	SessionName (see page 10027)	Identifies the name of the session with which this dataset is associated.

	Tag (see page 2808)	Stores an integer value as part of a component.

53.1.9.2 TDBDataSet.Create Constructor

Creates a new TDBDataSet (see page 10000) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDBDataSet(TComponent * AOwner);
```

Description

Call Create to instantiate a DB (see page 6725) dataset at runtime.

Create:

- Set the Locale (see page 9934) property of the dataset to the locale of the default session.
- Sets the CacheBlobs (see page 9932) property to true.
- Checks if the specified Owner is a database component, and if so, uses it to set the SessionName (see page 10027) and DatabaseName (see page 10022) properties.

See Also

TTable (see page 10168)

TQuery (see page 10059)

TStoredProc (see page 10139)

53.1.9.3 TDBDataSet Methods

The methods of the TDBDataSet class are listed here.

Protected Methods

	Name	Description
	CloseCursor (see page 10014)	This is CloseCursor, a member of class TDBDataSet.
	ConstraintsStored (see page 10015)	This is ConstraintsStored, a member of class TDBDataSet.
	Disconnect (see page 10015)	This is Disconnect, a member of class TDBDataSet.
	OpenCursor (see page 10015)	This is OpenCursor, a member of class TDBDataSet.
	PSEndTransaction (see page 10016)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecuteStatement (see page 10017)	Executes a specified SQL command.
	PSGetAttributes (see page 10017)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetQuoteChar (see page 10018)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSInTransaction (see page 10018)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 10018)	Indicates whether the dataset includes native SQL support.
	PSStartTransaction (see page 10019)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 10019)	Applies a single update to the underlying database table or tables.
	SetDBFlag (see page 10020)	This is SetDBFlag, a member of class TDBDataSet.
	SetHandle (see page 10020)	This is SetHandle, a member of class TDBDataSet.

Public Methods

	Name	Description
	CheckOpen (see page 10014)	Checks the result of a call to the Borland (see page 1761) Database (see page 10021) Engine (BDE).
	CloseDatabase (see page 10014)	Closes a database connection associated with the database session.
	OpenDatabase (see page 10016)	Opens the database that contains the dataset.

53.1.9.3.1 TDBDataSet.CheckOpen Method

Checks the result of a call to the Borland (see page 1761) Database (see page 10021) Engine (BDE).

Pascal

```
function CheckOpen(Status: DBIResult): Boolean;
```

C++

```
__fastcall Boolean CheckOpen(DBIResult Status);
```

Description

Call CheckOpen to determine if a call to the BDE returns an error when an attempt is made to access a dataset. Status is the return result of a previous call to the BDE.

CheckOpen returns true if access is successful. If Status indicates insufficient table rights when accessing a Paradox or dBASE table, CheckOpen calls the database session's GetPassword method to prompt the user for a password. If the dialog is successful, CheckOpen returns true.

Otherwise CheckOpen returns false, indicating that dataset access failed.

See Also

TDatabase (see page 9964)

TSession (see page 10093)

GetPassword (see page 10111)

53.1.9.3.2 TDBDataSet.CloseCursor Method

Pascal

```
procedure CloseCursor; override;
```

C++

```
virtual __fastcall CloseCursor();
```

Description

This is CloseCursor, a member of class TDBDataSet.

53.1.9.3.3 TDBDataSet.CloseDatabase Method

Closes a database connection associated with the database session.

Pascal

```
procedure CloseDatabase(Database: TDatabase);
```

C++

```
__fastcall CloseDatabase(TDatabase Database);
```

Description

Call CloseDatabase to close a persistent database connection, or to close and free a temporary database connection. Database (see page 10021) specifies the name of the database component for which to close the connection.

CloseDatabase decrements the specified database component's reference count, and then, if the reference count is zero and the database component's KeepConnection property is false, CloseDatabase either frees a temporary database component or closes the connection for a persistent database component.

Note: Calling CloseDatabase for a persistent database component does not close the connection. To close a connection for a persistent database component, call the database component's Close method directly.

Temporary database components are closed automatically when the last dataset associated with the database component is closed, but an application can call CloseDatabase prior to that time to force closure. Closing a connection established by a temporary database component does not free the component if the session's KeepConnections property is true (the default). To free temporary database components after closing their connections call DropConnections method of the session.

See Also

TSession (see page 10093)

DBSession (see page 10024)

Database (see page 10021)

DropConnections (see page 10105)

Close (see page 6863)

53.1.9.3.4 TDBDataSet.ConstraintsStored Method

Pascal

```
function ConstraintsStored: Boolean;
```

C++

```
__fastcall Boolean ConstraintsStored();
```

Description

This is ConstraintsStored, a member of class TDBDataSet.

53.1.9.3.5 TDBDataSet.Disconnect Method

Pascal

```
procedure Disconnect; virtual;
```

C++

```
virtual __fastcall Disconnect();
```

Description

This is Disconnect, a member of class TDBDataSet.

53.1.9.3.6 TDBDataSet.OpenCursor Method

Pascal

```
procedure OpenCursor(InfoQuery: Boolean); override;
```

C++

```
virtual __fastcall OpenCursor(Boolean InfoQuery);
```

Description

This is OpenCursor, a member of class TDBDataSet.

53.1.9.3.7 TDBDataSet.OpenDatabase Method

Opens the database that contains the dataset.

Pascal

```
function OpenDatabase: TDatabase;
```

C++

```
__fastcall TDatabase OpenDatabase();
```

Description

Call OpenDatabase to connect to the database that contains the dataset, or to create a temporary database component and connect to a database. The DatabaseName (see page 10022) property specifies the database to open.

OpenDatabase makes the current session active, then determines whether the DatabaseName (see page 10022) property corresponds to the DatabaseName (see page 10022) property of an existing database component. If it does not, OpenDatabase creates a temporary database component, assigning the DatabaseName (see page 10022) parameter to the DatabaseName (see page 10022) property. Finally, OpenDatabase calls the Open method of the database to connect to a database server, then increments the session's database reference count by one.

See Also

TDatabase (see page 9964)

DatabaseName (see page 10022)

Close (see page 6863)

53.1.9.3.8 TDBDataSet.PSEndTransaction Method

Ends a transaction that was started by a call to PSStartTransaction.

Pascal

```
procedure PSEndTransaction(Commit: Boolean); override;
```

C++

```
virtual __fastcall PSEndTransaction(Boolean Commit);
```

Description

The provider component applies updates within a transaction, if possible. To do so, it uses PSEndTransaction in conjunction with the PSStartTransaction method.

The Commit parameter indicates whether the dataset should commit the current transaction (Commit is true), or roll it back (Commit is false).

Note: TDataSetProvider only starts a transaction (Using PSStartTransaction) if the PSInTransaction method indicates that there is no transaction already established. That is, it does not assume nested transaction support is available. Thus, there is no need to ensure that PSEndTransaction does not terminate a transaction that was not started by the provider.

See Also

PSStartTransaction (see page 7586)

PSInTransaction (see page 7583)

applyupdates (see page 18459)

53.1.9.3.9 TDBDataSet.PSExecuteStatement Method

Executes a specified SQL command.

Pascal

```
function PSExecuteStatement(const ASQL: string; AParams: TParams; ResultSet: Pointer = nil):
Integer; override;
```

C++

```
virtual __fastcall int PSExecuteStatement(const AnsiString ASQL, TParams AParams, void *
ResultSet = nil);
```

Description

If the value its ResolveToDataSet property is false, a provider component calls PSExecuteStatement (indirectly) to execute SQL statements that it generates for applying updates. Usually, the dataset passes these statements on to its database server for execution.

The ASQL parameter specifies the SQL command to execute.

AParams contains any parameter values that should be applied to ASQL before execution.

ResultSet allows PSExecuteStatement to return a dataset component that contains the results of a SELECT statement. If ResultSet is not nil (Delphi) or NULL (C++), PSExecuteStatement creates a new dataset component as the target of this pointer and fills it with the result set. The caller is responsible for freeing the dataset.

PSExecuteStatement returns the number of rows affected by executing ASQL.

See Also

PSUpdateRecord (see page 7586)

applyupdates (see page 18459)

ResolveToDataSet (see page 18531)

53.1.9.3.10 TDBDataSet.PSGetAttributes Method

Returns information that is stored in data packets as Name/Value pairs.

Pascal

```
procedure PSGetAttributes(List: TList); override;
```

C++

```
virtual __fastcall PSGetAttributes(TList List);
```

Description

The provider component calls PSGetAttributes to add information about the dataset to the metadata stored with generated data packets. This information is stored as Name/Value pairs that are added to the list specified by the List parameter.

See Also

DoGetProviderAttributes (see page 18521)

53.1.9.3.11 TDBDataSet.PSGetQuoteChar Method

Returns the character or characters to use in generated SQL statements to enclose quoted strings.

Pascal

```
function PSGetQuoteChar: string; override;
```

C++

```
virtual __fastcall AnsiString PSGetQuoteChar();
```

Description

The provider component calls PSGetQuoteChar (indirectly) when generating the dynamic SQL statements that apply updates. These statements are subsequently executed by a call to PSExecuteStatement.

PSGetQuoteChar returns the character or characters that delimit quoted strings in the generated SQL. This string appears both before and after quoted strings.

See Also

applyupdates (see page 18459)

PSExecuteStatement (see page 7579)

ResolveToDataSet (see page 18531)

53.1.9.3.12 TDBDataSet.PSInTransaction Method

Indicates whether actions against the dataset occur in the context of a transaction.

Pascal

```
function PSInTransaction: Boolean; override;
```

C++

```
virtual __fastcall Boolean PSInTransaction();
```

Description

The provider component applies updates within a transaction, if possible. To do so, it first checks whether a transaction is already underway, and if not, generates one using PSStartTransaction. To determine whether a transaction is already underway, the provider calls PSInTransaction.

PSInTransaction returns true if there is a transaction underway, and false otherwise.

See Also

PSStartTransaction (see page 7586)

PSEndTransaction (see page 7578)

53.1.9.3.13 TDBDataSet.PSIsSQLBased Method

Indicates whether the dataset includes native SQL support.

Pascal

```
function PSIsSQLBased: Boolean; override;
```

C++

```
virtual __fastcall Boolean PSIsSQLBased();
```

Description

The provider calls PSIsSQLBased to determine whether the dataset supports some form of native SQL. This information is used when generating SQL commands that are executed by PSExecuteStatement.

Even if the dataset is not SQL-based, it may still allow the provider to execute SQL statements for applying updates. For example, BDE-enabled local databases such as Paradox and DBASE provide SQL support via the BDE's local SQL engine, but are not SQL-based. However, table and field names are not specified in the same way for the local SQL engine as they are for native SQL. The provider uses PSIsSQLBased to determine how these names should appear in generated SQL.

Note: The PSIsSQLSupported method indicates whether the dataset can execute SQL commands at all, regardless of whether the dataset is SQL-based.

See Also

PSIsSQLSupported (see page 7584)

PSExecuteStatement (see page 7579)

53.1.9.3.14 TDBDataSet.PSStartTransaction Method

Starts a transaction that encapsulates all activity until a call to PSEndTransaction.

Pascal

```
procedure PSStartTransaction; override;
```

C++

```
virtual __fastcall PSStartTransaction();
```

Description

The provider component applies updates within a transaction, if possible. To do so, it calls PSStartTransaction to start a transaction before applying updates, and calls PSEndTransaction when all updates are applied or too many errors are encountered.

Note: TDataSetProvider only starts a transaction (Using PSStartTransaction) if the PSInTransaction method indicates that there is no transaction already established. That is, it does not assume nested transaction support is available.

See Also

PSEndTransaction (see page 7578)

PSInTransaction (see page 7583)

applyupdates (see page 18459)

53.1.9.3.15 TDBDataSet.PSUpdateRecord Method

Applies a single update to the underlying database table or tables.

Pascal

```
function PSUpdateRecord(UpdateKind: TUpdateKind; Delta: TDataSet): Boolean; override;
```

C++

```
virtual __fastcall Boolean PSUpdateRecord(TUpdateKind UpdateKind, TDataSet Delta);
```

Description

Before the provider generates SQL statements to apply an update, it calls PSUpdateRecord to let the dataset apply the update in some other fashion. For example, BDE-enabled datasets generate an OnUpdateRecord to allow the application to apply the

update and, if the OnUpdateRecord event handler does not apply the update, uses any associated TUpdateSQL object to apply the update.

UpdateKind indicates whether the update is an insertion (ukInsert), deletion (ukDelete), or modification (ukModify).

Delta is the delta datapacket. The current record represents the record to be updated.

PSUpdateRecord returns true if it updates the record, false otherwise. If PSUpdateRecord returns false, the provider generates dynamic SQL to perform the update and calls PSExecuteStatement.

Note: PSUpdateRecord is only called if the provider's ResolveToDataSet property is false.

See Also

OnUpdateRecord (see page 9957)

TUpdateSQL (see page 10229)

applyupdates (see page 18459)

ResolveToDataSet (see page 18531)

PSExecuteStatement (see page 7579)

53.1.9.3.16 TDBDataSet.SetDBFlag Method

Pascal

```
function SetDBFlag(Flag: Integer; Value: Boolean): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean SetDBFlag(int Flag, Boolean Value);
```

Description

This is SetDBFlag, a member of class TDBDataSet.

53.1.9.3.17 TDBDataSet.SetHandle Method

Pascal

```
procedure SetHandle(Value: HDBICur);
```

C++

```
__fastcall SetHandle(HDBICur Value);
```

Description

This is SetHandle, a member of class TDBDataSet.

53.1.9.4 TDBDataSet Properties

The properties of the TDBDataSet class are listed here.

Protected Properties

	Name	Description
	DBFlags (see page 10023)	Keeps track of the commands this dataset has active with the database.
	UpdateMode (see page 10028)	Determines how the Borland (see page 1761) Database (see page 10021) Engine (BDE) finds records when updating to an SQL database.

Public Properties

	Name	Description
R	Database (see page 10021)	Identifies the database component for which this dataset represents one or more tables.
R	DBHandle (see page 10023)	Specifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) database handle for the dataset.
R	DBLocale (see page 10024)	Identifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) language driver for the dataset component.
R	DBSession (see page 10024)	Identifies the database session component associated with this dataset.
	Handle (see page 10027)	Specifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) cursor handle for the dataset.

Published Properties

	Name	Description
	AutoRefresh (see page 10021)	Specifies whether server-generated field values are refetched automatically.
	DatabaseName (see page 10022)	Specifies the name of the database associated with this dataset.
	Filter (see page 10025)	Specifies the text of the current filter for a dataset.
	Filtered (see page 10026)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 10026)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
	SessionName (see page 10027)	Identifies the name of the session with which this dataset is associated.

53.1.9.4.1 TDBDataSet.AutoRefresh Property

Specifies whether server-generated field values are refetched automatically.

Pascal

```
property AutoRefresh: Boolean;
```

C++

```
__property Boolean AutoRefresh;
```

Description

When AutoRefresh is false (the default), values that the server creates for autoincrement fields and fields with default values when a record is posted are not automatically refetched by the dataset. Instead, the application must call the dataset's Refresh method to update these field values. When AutoRefresh is true, these field values are automatically refreshed without an explicit call to the Refresh method.

Note: Some database drivers can't determine which fields are autoincrement fields or have default values. For these cases, you must provide this information by setting the AutoGenerateValue property of all fields that should be refreshed. Similarly, you must set the field components' AutoGenerateValue properties when the dataset represents a query or stored procedure.

See Also

AutoGenerateValue (see page 7153)

Refresh (see page 6991)

53.1.9.4.2 TDBDataSet.Database Property

Identifies the database component for which this dataset represents one or more tables.

Pascal

```
property Database: TDatabase;
```

C++

```
__property TDatabase Database;
```

Description

Use Database to access the properties, events, and methods of the database component associated with this dataset. Database is a read-only property that is automatically set when the database specified by the DatabaseName (see page 10022) property is opened.

See Also

DatabaseName (see page 10022)

Connecting Directly to a Database Server

53.1.9.4.3 TDBDataSet.DatabaseName Property

Specifies the name of the database associated with this dataset.

Pascal

```
property DatabaseName: string;
```

C++

```
__property AnsiString DatabaseName;
```

Description

Use DatabaseName to specify the name of the database to associate with this dataset component. DatabaseName should match the name of a database component used in the application.

Note: Attempting to set DatabaseName when a database already associated with this component is open raises an exception.

Tip: At design time double-click a TDatabase (see page 9964) component to invoke the Database (see page 10021) editor and set the DatabaseName.

Delphi Examples:

```
{
This example requires a database with the following name or
in the form of a *.DB file in the following path.  The table
should be closed when changing the database.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Customers.Active := False;
  try
  { First try to use an alias }
    Customers.DatabaseName := 'DBDEMOS';
    Customers.Active := True;
  except
    on EDatabaseError do
    begin
      { If that fails, try to use the drive and directory }
      Customers.TableName := 'Flights';
      Customers.DatabaseName :=
        'c:\Documents and Settings\All Users\Documents\RAD
Studio\5.0\Demos\IntraWeb\Win32\FlightInformation\Data';
      Customers.Active := True;
    end;
```

```
  end;
end;
```

See Also

Database (see page 10021)

DBHandle (see page 10023)

Using Implicit Connections

Creating and Deleting Tables

Using Query-type Datasets

Using Stored Procedure-type Datasets

Using Table Type Datasets

Associating a Dataset with Database and Session Connections

Using Multiple Update Objects

Connecting Directly to a Database Server

53.1.9.4.4 TDBDataSet.DBFlags Property

Keeps track of the commands this dataset has active with the database.

Pascal

```
property DBFlags: TDBFlags;
```

C++

```
__property TDBFlags DBFlags;
```

Description

Applications can't use this protected property. TDBDataSet (see page 10000) uses this property internally to keep track of the statements it has active with the current database connection. When DBFlags is empty, the dataset is not using the database connection.

53.1.9.4.5 TDBDataSet.DBHandle Property

Specifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) database handle for the dataset.

Pascal

```
property DBHandle: HDBIDB;
```

C++

```
__property HDBIDB DBHandle;
```

Description

Check (see page 10245) DBHandle to determine the database handle for the dataset. DBHandle is only useful in applications that bypass database and dataset methods to write directly to the BDE API. Many BDE function calls require a handle parameter. DBHandle is assigned an initial value when a session is activated.

Note: Do not use this property unless an application requires BDE functionality not available through standard data access components.

See Also

DBLocale (see page 10024)

TDatabase (see page 9964)

Working with BDE Handle Properties

53.1.9.4.6 TDBDataSet.DBLocale Property

Identifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) language driver for the dataset component.

Pascal

```
property DBLocale: TLocale;
```

C++

```
__property TLocale DBLocale;
```

Description

Examine DBLocale to determine the BDE language driver used by the database component associated with which this dataset is associated. Applications that make direct calls to the BDE may need to pass DBLocale information as an API function parameter.

Note: Do not use this property unless an application requires BDE functionality not available through standard data access components.

See Also

DBHandle (see page 10023)

TDatabase (see page 9964)

Working with BDE Handle Properties

53.1.9.4.7 TDBDataSet.DBSession Property

Identifies the database session component associated with this dataset.

Pascal

```
property DBSession: TSession;
```

C++

```
__property TSession DBSession;
```

Description

Use DBSession to determine the session component that controls the database component with which this dataset component is associated. By default, a database component is associated with the default session component, Session (see page 10265), that is automatically created for all database applications.

See Also

SessionName (see page 10027)

Session (see page 9988)

Session (see page 10265)

Associating a Dataset with Database and Session Connections

Using TQuery

53.1.9.4.8 TDBDataSet.Filter Property

Specifies the text of the current filter for a dataset.

Pascal

```
property Filter: string;
```

C++

```
__property AnsiString Filter;
```

Description

Use Filter to specify a dataset filter. When filtering is applied to a dataset, only those records that meet a filter's conditions are available. Filter describes the filter condition. For example, the following filter condition displays only those records where the State field is 'CA' or 'MA':

```
State = 'CA' or State = 'MA'
```

When a filter is set, Blank records do not appear unless explicitly included in the filter. For example:

```
State <> 'CA' or State = NULL
```

When a field name contains spaces, you must enclose the field name in brackets. For example:

```
[Home State] = 'CA' or [Home State] = 'MA'
```

Filter expressions on remote SQL tables and on client datasets support field comparisons. For example:

```
Field1 > Field2
```

Field comparisons are not supported against local tables (Paradox, dBASE, Access, FoxPro).

For a complete list of operators you can use in filter strings, see SettingTheFilterProperty.

Note: Filters are not supported on unidirectional datasets. If you set the Filter property of a unidirectional dataset, it raises an exception.

The FilterOptions property controls case sensitivity and filtering on partial comparisons.

Tip: Applications can set Filter at runtime to change the dataset's filtering condition (for example, in response to user input).

See Also

Filtered (see page 7027)

FilterOptions (see page 7027)

OnFilterRecord (see page 7055)

Locate (see page 6970)

Lookup (see page 6970)

Creating Filters

Navigating Records in a Filtered Dataset

Setting the Filter Property

Displaying and Editing a Subset of Data Using Filters

53.1.9.4.9 TDBDataSet.Filtered Property

Specifies whether or not filtering is active for a dataset.

Pascal

```
property Filtered: Boolean;
```

C++

```
__property Boolean Filtered;
```

Description

Check Filtered to determine whether or not dataset filtering is in effect. If Filtered is true, then filtering is active. To apply filter conditions specified in the Filter property or the OnFilterRecord event handler, set Filtered to true. Because unidirectional datasets do not support filters, when you set Filtered to true on a unidirectional dataset, it raises an exception.

Note: When filtering is enabled, user edits to a record may mean that the record no longer meets a filter's test condition. The next time an attempt is made to the read the record from the dataset while the filter is in effect, the record is not visible. If that happens, the next record that passes the filter condition becomes the active record.

See Also

Filter (see page 7026)

FilterOptions (see page 7027)

OnFilterRecord (see page 7055)

Locate (see page 6970)

Lookup (see page 6970)

Enabling and Disabling Filtering

Setting the Filter Property

53.1.9.4.10 TDBDataSet.FilterOptions Property

Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.

Pascal

```
property FilterOptions: TFilterOptions;
```

C++

```
__property TFilterOptions FilterOptions;
```

Description

Set FilterOptions to specify whether or not filtering is case insensitive when filtering on string or character fields, and whether or not partial comparisons for matching filter conditions is allowed.

Note: Because unidirectional datasets do not support filters, when you set the FilterOptions property of a unidirectional dataset, it raises an exception.

By default, FilterOptions is set to an empty set. For filters based on string fields, set FilterOptions to foCaseInsensitive to catch all variations on a string regardless of capitalization.

When a string in a filter ends with an asterisk (*), it can be used to match partial strings. To disable matching of partial strings and to treat the asterisk as a literal character in string comparisons, set FilterOptions to include foNoPartialCompare.

See Also

Filter (see page 7026)

Filtered (see page 7027)

OnFilterRecord (see page 7055)

Locate (see page 6970)

Lookup (see page 6970)

Setting Filter Options

53.1.9.4.11 TDBDataSet.Handle Property

Specifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) cursor handle for the dataset.

Pascal

```
property Handle: HDBICur;
```

C++

```
__property HDBICur Handle;
```

Description

Use Handle only to bypass TDataSet methods and call directly into the BDE. Many BDE function calls require a cursor handle parameter. Handle is assigned an initial value when a dataset is opened. If used with a BDE call that changes the current record position, call Resync immediately after returning from the BDE call.

TBDEDataSet (see page 9878) allows the Handle property to be set to an existing BDE cursor handle. Setting the Handle property is intended primarily for internal use. Descendant classes redeclare the Handle property to re-establish it as a read-only property

Note: Do not use this property unless an application requires BDE functionality not available through built-in components.

See Also

Locale (see page 9934)

Resync (see page 6994)

Working with BDE Handle Properties

53.1.9.4.12 TDBDataSet.SessionName Property

Identifies the name of the session with which this dataset is associated.

Pascal

```
property SessionName: string;
```

C++

```
__property AnsiString SessionName;
```

Description

Use SessionName to determine the session with which a dataset component is associated. SessionName is automatically set to the name of the SessionName property of the database component with which a dataset component is associated. If SessionName is blank, a dataset component is automatically associated with the default session, Session (see page 10265).

To associate a dataset component with a different session, SessionName must match the SessionName property of an existing

session component that is also used by the database component with which this dataset is associated.

See Also

DBSession (see page 10024)

Database (see page 10021)

Session (see page 10265)

Associating a Database Component with a Session

Associating a Dataset with Database and Session Connections

Using Multiple Update Objects

Adding a Session to the Web Module

53.1.9.4.13 TDBDataSet.UpdateMode Property

Determines how the Borland (see page 1761) Database (see page 10021) Engine (BDE) finds records when updating to an SQL database.

Pascal

```
property UpdateMode: TUpdateMode;
```

C++

```
__property TUpdateMode UpdateMode;
```

Description

Use UpdateMode to specify the criteria to use when locating a record in the dataset. UpdateMode specifies whether modified records are located based on all columns (fields), on only the key fields, or on the key fields plus the original values of fields that have been modified.

See Also

UpdateRecordTypes (see page 9936)

53.1.9.5 TDBDataSet Events

The events of the TDBDataSet class are listed here.

Published Events

	Name	Description
	OnFilterRecord (see page 10028)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.

53.1.9.5.1 TDBDataSet.OnFilterRecord Event

Occurs each time a different record in the dataset becomes the active record and filtering is enabled.

Pascal

```
property OnFilterRecord: TFilterRecordEvent;
```

C++

```
__property TFilterRecordEvent OnFilterRecord;
```

Description

Write an OnFilterRecord event handler to specify for each record in a dataset whether it should be visible to the application. To indicate that a record passes the filter condition, the OnFilterRecord event handler must set the Accept parameter to true. To exclude a record, set the Accept parameter to false. Most TDataSet descendants (such as ADO datasets or BDE-based datasets) initialize Accept to true before calling the OnFilterRecord event handler.

Warning: Do not supply an OnFilterRecord event handler to a unidirectional dataset. Unidirectional datasets do not support filters, and assigning an OnFilterRecord event handler causes the dataset to raise an exception.

Filtering is enabled if the Filtered property is true. When an application is processing a filter, the State property for the dataset is dsFilter.

Use an OnFilterRecord event handler to filter records using a criterion that can't be implemented using the Filter property. For example, using the Filter property, field comparisons are not supported against local tables (Paradox, dBASE, Access, FoxPro), but an OnFilterRecord event handler can implement any criterion at all.

Tip: Be sure that any interactions between the Filter property and the OnFilterRecord event handler do not result in an empty filter set when they are used simultaneously in an application.

See Also

Filter (see page 7026)

Filtered (see page 7027)

FilterOptions (see page 7027)

State (see page 7036)

Creating Filters

Navigating Records in a Filtered Dataset

Displaying and Editing a Subset of Data Using Filters

Writing an OnFilterRecord Event Handler

53.1.10 TDBError Class

TDBError represents Borland (see page 1761) Database Engine errors for the EDBEngineError (see page 9854) exception class.

Class Hierarchy

DBTables.TDBError

Pascal

```
TDBError = class;
```

C++

```
class TDBError;
```

File

DBTables

Description

The TDBError object is a container for the information pertaining to one database error. One or more TDBError objects are

contained in the Errors property (EDBEngineError (see page 9854)).

An EDBEngineError (see page 9854) exception is raised when Borland (see page 1761) Database Engine (BDE) or a database back-end accessed via the BDE produces an error condition on execution of a database operation. Information about each error involved in an error condition (such as an error code and error message) is passed by the BDE to the EDBEngineError (see page 9854) object. If the error occurs at the level of a database system accessed by the BDE (such as InterBase, Oracle, or Sybase), the error information is passed through the BDE to the application's exception object. EDBEngineError (see page 9854) reads the stack of database engine errors and creates a separate TDBError object to represent each one. These TDBError objects are contained in the Errors property of EDBEngineError (see page 9854).

When an error condition originates with a database system accessed through the BDE, there will typically be two TDBError objects in Errors. The first will be the BDE version of the error and the second the database system's version of the same error. In both cases, the Category (see page 10031), ErrorCode (see page 10032), and SubCode (see page 10033) properties will contain the same values: the BDE error code for the particular type of error condition. In the TDBError object for the BDE's version of the error the NativeCode property is empty and the text for the Message (see page 10032) property contains the BDE error message the type type of error that occurred. In the TDBError object for the database system's version of the error, the NativeCode property contains the database systems specific error code and the Message (see page 10032) property text provided by the database system (and not the BDE).

For example, when a violation of an InterBase CHECK constraint occurs (an error condition that originates with InterBase and not the BDE), the error message provided by the BDE is the generic "General SQL error," while the error message provided by InterBase is a more explanatory error like "Operation violates CHECK constraint INTEG_30 on view or table EMPLOYEE."

It is also possible to have only BDE type errors in the TDBError objects. This is usually only applicable to use of local table types (dBASE, Paradox, and FoxPro), where the BDE is the database back-end and there is no external database system used.

The categories and subcodes for BDE errors are defined in bde.int (Delphi) or bde.hpp (C++). For error information provided by a database system other than the BDE, consult the vendor-supplied documentation for the specific database system for definition or clarification.

See Also

EDBEngineError (see page 9854)

53.1.10.1 TDBError Members

The following tables list the members exposed by TDBError.

Public Methods

	Name	Description
	Create (see page 10030)	Initializes an instance of TDBError (see page 10029) using error information returned by the BDE.

Public Properties

	Name	Description
R	Category (see page 10031)	Indicates the category of error referenced by ErrorCode (see page 10032).
R	ErrorCode (see page 10032)	Indicates the error code returned by the BDE.
R	Message (see page 10032)	Specifies the text of the error message.
R	NativeError (see page 10032)	Indicates the remote error code returned from the server.
R	SubCode (see page 10033)	Indicates the subcode of the error referenced by ErrorCode (see page 10032).

53.1.10.2 TDBError.Create Constructor

Initializes an instance of TDBError (see page 10029) using error information returned by the BDE.

Pascal

```
constructor Create(Owner: EDBEngineError; ErrorCode: DBIResult; NativeError: Longint; Message:
PChar);
```

C++

```
__fastcall TDBError(EDBEngineError Owner, DBIResult ErrorCode, Longint NativeError, const char
* Message);
```

Description

TDBError (see page 10029)'s Create method is called by the EDBEngineError (see page 9854) constructor, just as TDBError (see page 10029) objects are freed by the EDBEngineError (see page 9854) destructor. Application's should never have cause for calling Create directly, except in the case of custom exception-handling objects descending from or similar to the EDBEngineError (see page 9854) exception object.

See Also

EDBEngineError (see page 9854)

Create (see page 9856)

53.1.10.3 TDBError Properties

The properties of the TDBError class are listed here.

Public Properties

	Name	Description
R	Category (see page 10031)	Indicates the category of error referenced by ErrorCode (see page 10032).
R	ErrorCode (see page 10032)	Indicates the error code returned by the BDE.
R	Message (see page 10032)	Specifies the text of the error message.
R	NativeError (see page 10032)	Indicates the remote error code returned from the server.
R	SubCode (see page 10033)	Indicates the subcode of the error referenced by ErrorCode (see page 10032).

53.1.10.3.1 TDBError.Category Property

Indicates the category of error referenced by ErrorCode (see page 10032).

Pascal

```
property Category: Byte;
```

C++

```
__property Byte Category;
```

Description

Category contains the high byte of the BDE error code. Combined with the low byte of the BDE error code (in the SubCode (see page 10033) property) this produces the whole BDE error code (also contained in the ErrorCode (see page 10032) property).

See Also

ErrorCode (see page 10032)

SubCode (see page 10033)

53.1.10.3.2 TDBError.ErrorCode Property

Indicates the error code returned by the BDE.

Pascal

```
property ErrorCode: DBIResult;
```

C++

```
__property DBIResult ErrorCode;
```

Description

Use ErrorCode to determine the error code returned by the BDE. The value of ErrorCode is the combination of the high byte of the BDE error code (also in the Category (see page 10031) property) and the low byte (in the SubCode (see page 10033) property).

See Also

SubCode (see page 10033)

53.1.10.3.3 TDBError.Message Property

Specifies the text of the error message.

Pascal

```
property Message: string;
```

C++

```
__property AnsiString Message;
```

Description

Message contains the remote server message for native (server) errors or the BDE message associated with ErrorCode (see page 10032) for non-server errors. To determine which type of message is contained in the Message property, inspect the NativeError (see page 10032) property. A zero value there indicates a message provided by the BDE. A nonzero value indicates a message provided by the database system (server).

See Also

NativeError (see page 10032)

53.1.10.3.4 TDBError.NativeError Property

Indicates the remote error code returned from the server.

Pascal

```
property NativeError: Longint;
```

C++

```
__property Longint NativeError;
```

Description

Use NativeError to determine whether the error represents a remote error received from the database server. If NativeError is 0, the error is not a server error.

Values in the NativeError property originate with the database system and not the BDE. Refer to vendor documentation for the specific database system used.

See Also

ErrorCode (see page 10032)

53.1.10.3.5 TDBError.SubCode Property

Indicates the subcode of the error referenced by ErrorCode (see page 10032).

Pascal

```
property SubCode: Byte;
```

C++

```
__property Byte SubCode;
```

Description

SubCode contains the low byte of the BDE error code. Combined with the high byte of the BDE error code (in the Category (see page 10031) property) this produces the whole BDE error code (also contained in the ErrorCode (see page 10032) property).

See Also

ErrorCode (see page 10032)

Category (see page 10031)

53.1.11 TIndexFiles Class

Class Hierarchy

Pascal

```
TIndexFiles = class(TStringList);
```

C++

```
class TIndexFiles : public TStringList;
```

File

DBTables

Description

This is class DBTables.TIndexFiles.

53.1.11.1 TIndexFiles Members

The following tables list the members exposed by TIndexFiles.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 3013)	This is Changed, a member of class TStringList.
	Changing (see page 3013)	This is Changing, a member of class TStringList.

V	CompareStrings (see page 3014)	Compares two strings.
V	CompareStrings (see page 3048)	Compares two strings.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 3049)	Reads and writes the Strings (see page 3080) property as if it were published.
	Error (see page 3052)	This is the overview for the Error method overload.
	ExtractName (see page 3053)	Returns the name portion of a string that is a name value pair.
V	Get (see page 3017)	Returns a string given its index.
A	Get (see page 3053)	Returns a string given its index.
V	GetCapacity (see page 3017)	Returns the currently allocated size of the list of strings.
V	GetCapacity (see page 3054)	Returns the currently allocated size of the list of strings.
V	GetCount (see page 3018)	Returns the number of strings in the list
A	GetCount (see page 3054)	Returns the number of strings in the list
V	GetObject (see page 3018)	Returns the object associated with the string at a specified index.
V	GetObject (see page 3055)	Returns the object associated with the string at a specified index.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetTextStr (see page 3056)	Returns the value of the Text (see page 3084) property.
V	InsertItem (see page 3020)	This is InsertItem, a member of class TStringList.
V	Put (see page 3022)	Changes the value of the string with a specified index.
V	Put (see page 3064)	Changes the value of the string with a specified index.
V	PutObject (see page 3022)	Changes the object associated with the string at a specified index.
V	PutObject (see page 3065)	Changes the object associated with the string at a specified index.
V	SetCapacity (see page 3022)	Changes the amount of memory allocated to hold strings in the list.
V	SetCapacity (see page 3068)	Changes the amount of memory allocated to hold strings in the list.
V	SetTextStr (see page 3069)	Sets the Text (see page 3084) property.
V	SetUpdateState (see page 3023)	Performs internal adjustments before or after a series of updates.
V	SetUpdateState (see page 3069)	Performs internal adjustments before or after a series of updates.

Protected Properties

	Name	Description
R	UpdateCount (see page 3085)	Indicates the number of calls to BeginUpdate (see page 3045) that have not been matched by a call to EndUpdate (see page 3050).

Public Events

	Name	Description
	OnChange (see page 3028)	Occurs immediately after the list of strings changes.
	OnChanging (see page 3028)	Occurs immediately before the list of strings changes.

Public Methods

	Name	Description
V	Add (see page 10037)	Adds a new string to the list.
V	AddObject (see page 3012)	Adds a string to the list, and associates an object with the string.
V	AddObject (see page 3036)	Adds a string to the list, and associates an object with the string.
V	AddStrings (see page 3039)	Adds a group of strings to the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 3041)	Adds a string to the list.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	Assign (see page 3043)	Sets the strings in the list, and possibly associated objects, from a source object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 3045)	Enables the TStrings (see page 3029) object to track when the list of strings is changing.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 10039)	Deletes all the strings from the list.
	Create (see page 10036)	This is Create, a member of class TIndexFiles.
V	CustomSort (see page 3014)	Sorts the strings in the list in a customized order.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Delete (see page 10039)	Removes the string specified by the Index parameter.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 3009)	Destroys an instance of TStringList (see page 3006).
V	Destroy (see page 3032)	Destroys the TPersistent instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 3050)	Enables the TStrings (see page 3029) object to keep track of when the list of strings has finished changing.
	Equals (see page 3051)	Compares the list of strings to the list from another TStrings (see page 3029) object and returns true if the two lists match.
V	Exchange (see page 3015)	Swaps the position of two strings in the list.
V	Exchange (see page 3053)	Swaps the position of two strings in the list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
V	Find (see page 3016)	Locates the index for a string in a sorted list and indicates whether a string with that value already exists in the list.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 3054)	Returns a TStrings (see page 3029) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetText (see page 3055)	Allocates a text buffer and fills it with the value of the Text (see page 3084) property.
V	IndexOf (see page 3019)	Returns the position of a string in the list.
V	IndexOf (see page 3056)	Returns the position of a string in the list.
V	IndexOfName (see page 3057)	Returns the position of the first name-value pair with the specified name.
V	IndexOfObject (see page 3059)	Returns the index of the first string in the list associated with a given object.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Insert (see page 10040)	Inserts a string to the list at the position specified by Index.
V	InsertObject (see page 3020)	Inserts a string into the list at the specified position, and associates it with an object.
V	InsertObject (see page 3060)	Inserts a string into the list at the specified position, and associates it with an object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	LoadFromFile (see page 3061)	Fills the list with the lines of text in a specified file.
V	LoadFromStream (see page 3062)	Fills the list with lines of text read from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	Move (see page 3063)	Changes the position of a string in the list.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SaveToFile (see page 3065)	Saves the strings in the list to the specified file.
V	SaveToStream (see page 3067)	Writes the value of the Text (see page 3084) property to a stream object.
V	SetText (see page 3068)	Sets the Text (see page 3084) property.

V	Sort (see page 3023)	Sorts the strings in the list in ascending order.

Public Properties

	Name	Description
	Capacity (see page 3070)	Indicates the number of strings the TStrings (see page 3029) object can hold.
	CaseSensitive (see page 3025)	Controls (see page 5711) whether strings are located, sorted, and identified as duplicates in a case-sensitive or case-insensitive manner.
	CommaText (see page 3070)	Lists the strings in the TStrings (see page 3029) object in system data format (SDF).
R	Count (see page 3071)	Introduces an abstract property to represent the number of strings in the list.
	DelimitedText (see page 3074)	Represents all the strings in the TStrings (see page 3029) object as a single delimited string.
	Delimiter (see page 3074)	Specifies the delimiter used by the DelimitedText (see page 3074) property.
	Duplicates (see page 3025)	Specifies whether duplicate strings can be added to sorted lists.
	LineBreak (see page 3075)	Defines the line break character(s).
R	Names (see page 3075)	Indicates the name part of strings that are name-value pairs.
	NameValueSeparator (see page 3077)	Indicates the character used to separate names from values.
	Objects (see page 3077)	Represents a set of objects that are associated one with each of the strings in the Strings (see page 3080) property.
	QuoteChar (see page 3080)	Specifies the quote character used by the DelimitedText (see page 3074) property.
	Sorted (see page 3026)	Specifies whether the strings in the list should be automatically sorted.
	StrictDelimiter (see page 3080)	Determines how the Delimiter (see page 3074) property is used.
	Strings (see page 3080)	References the strings in the list by their positions.
	StringsAdapter (see page 3084)	Implements an IStringsAdapter (see page 3208) interface for the TStrings (see page 3029) object.
	Text (see page 3084)	Lists the strings in the TStrings (see page 3029) object as a single string with the individual strings delimited by carriage returns and line feeds.
	ValueFromIndex (see page 3085)	Represents the value part of a string with a given index, on strings that are name-value pairs.
	Values (see page 3086)	Represents the value part of a string associated with a given name, on strings that are name-value pairs.

53.1.11.2 TIndexFiles.Create Constructor

Pascal

```
constructor Create(AOwner: TTable);
```

C++

```
__fastcall TIndexFiles(TTable AOwner);
```

Description

This is Create, a member of class TIndexFiles.

53.1.11.3 TIndexFiles Methods

The methods of the TIndexFiles class are listed here.

Public Methods

	Name	Description
V	Add (see page 10037)	Adds a new string to the list.
V	Clear (see page 10039)	Deletes all the strings from the list.
V	Delete (see page 10039)	Removes the string specified by the Index parameter.
V	Insert (see page 10040)	Inserts a string to the list at the position specified by Index.

53.1.11.3.1 TIndexFiles.Add Method

Adds a new string to the list.

Pascal

```
function Add(const S: string): Integer; override;
```

C++

```
virtual __fastcall int Add(const AnsiString S);
```

Description

Call Add to add the string S to the list. If the list is sorted, S is added to the appropriate position in the sort order. If the list is not sorted, S is added to the end of the list. Add returns the position of the item in the list, where the first item in the list has a value of 0.

Note: For sorted lists, Add will raise an EListError exception if the string S already appears in the list and Duplicates is set to dupError. If Duplicates is set to dupIgnore, trying to add a duplicate string causes Add to return the index of the existing entry.

C++ Examples:

```
/*
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  int Index;
  TStringList* MyList = new TStringList();
  try
  {
    MyList->Add("Animals");
    MyList->Add("Flowers");
    MyList->Add("Ocean");
    MyList->Sort(); // Find will only work on sorted lists
    if (MyList->Find("Flowers", Index))
    {
      ListBox1->Items->AddStrings(MyList);
      Label1->Caption = "Flowers has an index value of " + AnsiString(Index);
    }
  }
  __finally
  {
    delete MyList;
  }
}
/*
This example creates a stringlist, adds elements to it and
assigns the strings to items of a combo box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TStrings *StringList = new TStringList();
  try
  {
    StringList->Add("This example uses A string List. ");
```

```
    StringList->Add("It is the easiest way to add strings");
    StringList->Add("to a combobox's list of strings. ");
    StringList->Add("Always remember TStrings is abstract,");
    StringList->Add("So create a TStringList instead.");
    ComboBox1->Width = 210;
    ComboBox1->Items->Assign(StringList);
    ComboBox1->ItemIndex = 0;
  }
  __finally
  {
    delete StringList;
  }
}
```

Delphi Examples:

```
{
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  MyList: TStringList;
  Index: Integer;
begin
  MyList := TStringList.Create;
  try
    MyList.Add('Animals');
    MyList.Add('Flowers');
    MyList.Add('Cars');
    MyList.Sort;   { Find will only work on sorted lists! }
    if MyList.Find('Flowers', Index) then
    begin
      ListBox1.Items.AddStrings(MyList);
      Label1.Caption := 'Flowers has an index value of ' + IntToStr(Index);
    end;
  finally
    MyList.Free;
  end;
end;
{
This example creates a stringlist, adds elements to it and
assigns the strings to items of a combo box.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  StringList: TStringList;
begin
  StringList := TStringList.Create;
  try
    with StringList do begin
      Add('This example uses A string List.');
      Add('It is the easiest way to add strings');
      Add('to a combobox''s list of strings.');
      Add('Always remember TStrings.Create method');
      Add('is abstract; So use TStringList.Create');
      Add('method instead.');
    end;
    with ComboBox1 do begin
      Width := 210;
      Items.Assign(StringList);
      ItemIndex := 0;
```

```
    end;
  finally
    StringList.free;
  end;
end;
```

See Also

AddStrings (see page 3039)

Append (see page 3041)

Delete (see page 3015)

Duplicates (see page 3025)

Insert (see page 3019)

Sorted (see page 3026)

53.1.11.3.2 TIndexFiles.Clear Method

Deletes all the strings from the list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call clear to empty the list of strings. All references to associated objects are also removed. However, the objects themselves are not freed.

See Also

Delete (see page 3015)

Objects

Strings

53.1.11.3.3 TIndexFiles.Delete Method

Removes the string specified by the Index parameter.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Call Delete to remove a single string from the list. If an object is associated with the string, the reference to the object is removed as well. Index gives the position of the string, where 0 is the first string, 1 is the second string, and so on.

See Also

Add (see page 3010)

AddObject (see page 3012)

Clear (see page 3013)

IndexOf (see page 3019)

InsertObject (see page 3060)

Objects

Strings

53.1.11.3.4 TIndexFiles.Insert Method

Inserts a string to the list at the position specified by Index.

Pascal

```
procedure Insert(Index: Integer; const S: string); override;
```

C++

```
virtual __fastcall Insert(int Index, const AnsiString S);
```

Description

Call Insert to add the string S to the list at the position specified by Index. If Index is 0, the string is inserted at the beginning of the list. If Index is 1, the string is put in the second position of the list, and so on.

If the string has an associated object, use the InsertObject method instead.

Note: If the list is sorted, calling Insert or InsertObject will raise an EListError exception. Use Add or AddObject with sorted lists.

See Also

Add (see page 3010)

AddObject (see page 3012)

EListError (see page 2701)

InsertObject (see page 3020)

Sorted (see page 3026)

Strings

53.1.12 TNestedTable Class

TNestedTable encapsulates a dataset that is nested as a field within another table.

Class Hierarchy

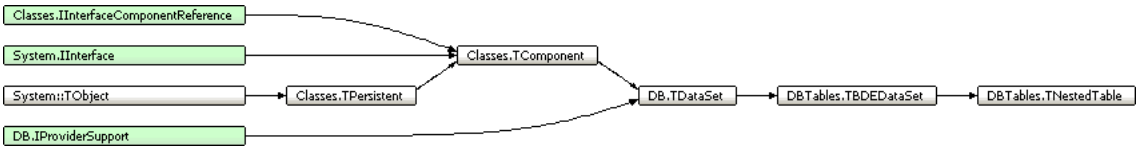

Pascal

```
TNestedTable = class(TBDEDataSet);
```

C++

```
class TNestedTable : public TBDEDataSet;
```

File

DBTables

Description

Use TNestedTable to access data contained in a nested dataset. A nested table inherits BDE functionality from TBDEDataSet (see page 9878) and so uses the Borland (see page 1761) Database Engine (BDE) to access the nested table data. A nested table provides much of the functionality of a table component, except that the data it accesses is stored in a nested table.

Note: TNestedTable is used only for dataset fields in tables that have a BDE driver (such as Oracle8 tables). For example, detail sets nested in a client dataset use another client dataset instead.

53.1.12.1 TNestedTable Members

The following tables list the members exposed by TNestedTable.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	ActivateFilters (see page 9893)	This is ActivateFilters, a member of class TBDEDataSet.
	AddFieldDesc (see page 9893)	This is AddFieldDesc, a member of class TBDEDataSet.
	AllocCachedUpdateBuffers (see page 9893)	This is AllocCachedUpdateBuffers, a member of class TBDEDataSet.
	AllocKeyBuffers (see page 9893)	This is AllocKeyBuffers, a member of class TBDEDataSet.
V	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
V	AllocRecordBuffer (see page 9894)	This is AllocRecordBuffer, a member of class TBDEDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
V	BlockReadNext (see page 9894)	This is BlockReadNext, a member of class TBDEDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
	CachedUpdateCallBack (see page 9895)	This is CachedUpdateCallBack, a member of class TBDEDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
	CheckCachedUpdateMode (see page 9896)	This is CheckCachedUpdateMode, a member of class TBDEDataSet.
V	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
V	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
	CheckSetKeyMode (see page 9896)	This is CheckSetKeyMode, a member of class TBDEDataSet.
V	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
V	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
V	ClearCalcFields (see page 9897)	This is ClearCalcFields, a member of class TBDEDataSet.
V	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
V	CloseBlob (see page 9897)	This is CloseBlob, a member of class TBDEDataSet.

	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
	CloseCursor (see page 9897)	This is CloseCursor, a member of class TBDEDataSet.
	CreateExprFilter (see page 9900)	This is CreateExprFilter, a member of class TBDEDataSet.
	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
	CreateFuncFilter (see page 9900)	This is CreateFuncFilter, a member of class TBDEDataSet.
	CreateHandle (see page 10053)	This is CreateHandle, a member of class TNestedTable.
	CreateLookupFilter (see page 9901)	This is CreateLookupFilter, a member of class TBDEDataSet.
	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
	DataEvent (see page 6924)	This is DataEvent, a member of class TDataSet.
	DataEvent (see page 9901)	This is DataEvent, a member of class TBDEDataSet.
	DeactivateFilters (see page 9901)	This is DeactivateFilters, a member of class TBDEDataSet.
	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
	DestroyHandle (see page 9901)	This is DestroyHandle, a member of class TBDEDataSet.
	DestroyLookupCursor (see page 9902)	This is DestroyLookupCursor, a member of class TBDEDataSet.
	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
	DoAfterPost (see page 10054)	This is DoAfterPost, a member of class TNestedTable.
	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeInsert (see page 10054)	This is DoBeforeInsert, a member of class TNestedTable.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FindRecord (see page 9903)	This is FindRecord, a member of class TBDEDataSet.
	ForceUpdateCallback (see page 9904)	This is ForceUpdateCallback, a member of class TBDEDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeKeyBuffers (see page 9904)	This is FreeKeyBuffers, a member of class TBDEDataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
	FreeRecordBuffer (see page 9904)	This is FreeRecordBuffer, a member of class TBDEDataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
	GetBookmarkData (see page 9905)	This is GetBookmarkData, a member of class TBDEDataSet.
	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
	GetBookmarkFlag (see page 9905)	This is GetBookmarkFlag, a member of class TBDEDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.

	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
	GetCanModify (see page 9905)	This is GetCanModify, a member of class TBDEDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetFieldFullName (see page 9907)	This is GetFieldFullName, a member of class TBDEDataSet.
	GetHandle (see page 9907)	This is GetHandle, a member of class TBDEDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIndexField (see page 9907)	This is GetIndexField, a member of class TBDEDataSet.
	GetIndexFieldCount (see page 9907)	This is GetIndexFieldCount, a member of class TBDEDataSet.
	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
	GetIsIndexField (see page 9908)	This is GetIsIndexField, a member of class TBDEDataSet.
	GetKeyBuffer (see page 9908)	This is GetKeyBuffer, a member of class TBDEDataSet.
	GetKeyExclusive (see page 9908)	This is GetKeyExclusive, a member of class TBDEDataSet.
	GetKeyFieldCount (see page 9909)	This is GetKeyFieldCount, a member of class TBDEDataSet.
	GetLookupCursor (see page 9909)	This is GetLookupCursor, a member of class TBDEDataSet.
	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetObjectTypeNames (see page 9909)	This is GetObjectTypeNames, a member of class TBDEDataSet.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
	GetRecNo (see page 9909)	This is GetRecNo, a member of class TBDEDataSet.
	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
	GetRecord (see page 9910)	This is GetRecord, a member of class TBDEDataSet.
	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
	GetRecordCount (see page 9910)	This is GetRecordCount, a member of class TBDEDataSet.
	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
	GetRecordSize (see page 9910)	This is GetRecordSize, a member of class TBDEDataSet.
	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
	GetStateFieldValue (see page 9910)	This is GetStateFieldValue, a member of class TBDEDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetUpdateRecordSet (see page 9910)	This is GetUpdateRecordSet, a member of class TBDEDataSet.
	GetUpdatesPending (see page 9911)	This is GetUpdatesPending, a member of class TBDEDataSet.
	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
	InitKeyBuffer (see page 9911)	This is InitKeyBuffer, a member of class TBDEDataSet.
	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
	InitRecord (see page 9911)	This is InitRecord, a member of class TBDEDataSet.
	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
	InternalAddRecord (see page 9911)	This is InternalAddRecord, a member of class TBDEDataSet.
	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
	InternalCancel (see page 9912)	This is InternalCancel, a member of class TBDEDataSet.

	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.
	InternalClose (see page 9912)	This is InternalClose, a member of class TBDEDataSet.
	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
	InternalDelete (see page 9912)	This is InternalDelete, a member of class TBDEDataSet.
	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
	InternalEdit (see page 9912)	This is InternalEdit, a member of class TBDEDataSet.
	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
	InternalFirst (see page 9912)	This is InternalFirst, a member of class TBDEDataSet.
	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
	InternalGotoBookmark (see page 9913)	This is InternalGotoBookmark, a member of class TBDEDataSet.
	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
	InternalHandleException (see page 9913)	This is InternalHandleException, a member of class TBDEDataSet.
	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
	InternalInitFieldDefs (see page 9913)	This is InternalInitFieldDefs, a member of class TBDEDataSet.
	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
	InternalInitRecord (see page 9913)	This is InternalInitRecord, a member of class TBDEDataSet.
	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
	InternalInsert (see page 9914)	This is InternalInsert, a member of class TBDEDataSet.
	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
	InternalLast (see page 9914)	This is InternalLast, a member of class TBDEDataSet.
	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
	InternalOpen (see page 9914)	This is InternalOpen, a member of class TBDEDataSet.
	InternalPost (see page 10054)	This is InternalPost, a member of class TNestedTable.
	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
	InternalRefresh (see page 9914)	This is InternalRefresh, a member of class TBDEDataSet.
	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	InternalSetToRecord (see page 9915)	This is InternalSetToRecord, a member of class TBDEDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
	IsCursorOpen (see page 9915)	This is IsCursorOpen, a member of class TBDEDataSet.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 9916)	This is LocateRecord, a member of class TBDEDataSet.
	MapsToIndex (see page 9918)	This is MapsToIndex, a member of class TBDEDataSet.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	OpenCursor (see page 6976)	This is OpenCursor, a member of class TDataSet.
	OpenCursor (see page 9918)	This is OpenCursor, a member of class TBDEDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PostKeyBuffer (see page 9919)	This is PostKeyBuffer, a member of class TBDEDataSet.
	PrepareCursor (see page 9919)	This is PrepareCursor, a member of class TBDEDataSet.
	ProcessUpdates (see page 9919)	This is ProcessUpdates, a member of class TBDEDataSet.
	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 6982)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandType (see page 6982)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.

	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSGetUpdateException (see page 9920)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 9920)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 9921)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 9921)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	ResetCursorRange (see page 9922)	This is ResetCursorRange, a member of class TBDEDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBlockReadSize (see page 9922)	This is SetBlockReadSize, a member of class TBDEDataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkData (see page 9922)	This is SetBookmarkData, a member of class TBDEDataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkFlag (see page 9923)	This is SetBookmarkFlag, a member of class TBDEDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetCachedUpdates (see page 9923)	This is SetCachedUpdates, a member of class TBDEDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetCursorRange (see page 9923)	This is SetCursorRange, a member of class TBDEDataSet.
	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldData (see page 9923)	This is SetFieldData, a member of class TBDEDataSet.
	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
	SetFilterData (see page 9923)	This is SetFilterData, a member of class TBDEDataSet.
	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
	SetFiltered (see page 9924)	This is SetFiltered, a member of class TBDEDataSet.
	SetFilterHandle (see page 9924)	This is SetFilterHandle, a member of class TBDEDataSet.
	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.

V	SetFilterOptions (see page 9924)	This is SetFilterOptions, a member of class TBDEDataSet.
V	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
V	SetFilterText (see page 9924)	This is SetFilterText, a member of class TBDEDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetIndexField (see page 9925)	This is SetIndexField, a member of class TBDEDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetKeyBuffer (see page 9925)	This is SetKeyBuffer, a member of class TBDEDataSet.
	SetKeyExclusive (see page 9925)	This is SetKeyExclusive, a member of class TBDEDataSet.
	SetKeyFieldCount (see page 9925)	This is SetKeyFieldCount, a member of class TBDEDataSet.
	SetKeyFields (see page 9925)	This is SetKeyFields, a member of class TBDEDataSet.
	SetLinkRanges (see page 9926)	This is SetLinkRanges, a member of class TBDEDataSet.
	SetLocale (see page 9926)	This is SetLocale, a member of class TBDEDataSet.
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
V	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
V	SetOnFilterRecord (see page 9926)	This is SetOnFilterRecord, a member of class TBDEDataSet.
	SetOnUpdateError (see page 9926)	This is SetOnUpdateError, a member of class TBDEDataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
V	SetRecNo (see page 9927)	This is SetRecNo, a member of class TBDEDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
V	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
V	SetStateFieldValue (see page 9927)	This is SetStateFieldValue, a member of class TBDEDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	SetupCallBack (see page 9927)	This is SetupCallBack, a member of class TBDEDataSet.
	SetUpdateObject (see page 9927)	This is SetUpdateObject, a member of class TBDEDataSet.
	SetUpdateRecordSet (see page 9927)	This is SetUpdateRecordSet, a member of class TBDEDataSet.
	SwitchToIndex (see page 9928)	This is SwitchToIndex, a member of class TBDEDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	UpdateCallbackRequired (see page 9928)	This is UpdateCallbackRequired, a member of class TBDEDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.

R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
	Reserved (see page 7035)	For internal use only.
	StmtHandle (see page 9935)	Specifies the Borland (see page 1761) Database Engine (BDE) statement handle for the dataset.

Public Events

	Name	Description
	AfterCancel (see page 7038)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 7040)	Occurs after an application closes a dataset.
	AfterDelete (see page 7041)	Occurs after an application deletes a record.
	AfterEdit (see page 7044)	Occurs after an application starts editing a record.
	AfterInsert (see page 7044)	Occurs after an application inserts a new record.
	AfterOpen (see page 7045)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 7046)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 7047)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 7047)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 7048)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 7048)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 7049)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 7049)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 7050)	Occurs before an application enters insert mode.
	BeforeOpen (see page 7051)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 7052)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 7052)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 7053)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 7053)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 7054)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 7054)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 7056)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 7056)	Occurs when an application attempts to modify or insert a record and an exception is raised.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.

	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
	ApplyUpdates (see page 9894)	Writes a dataset's pending cached updates to the database.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
V	BookmarkValid (see page 9895)	Tests the validity of a specified bookmark.
V	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
V	Cancel (see page 9895)	Cancels modifications to the current record if those changes are not yet posted.
	CancelUpdates (see page 9896)	Clears all pending cached updates from the cache and restores the dataset its prior state.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Close (see page 6920)	Closes a dataset.
	CommitUpdates (see page 9897)	Clears the cached updates buffer.
V	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
V	CompareBookmarks (see page 9898)	Indicates the relationship between two bookmarks.
	ConstraintCallBack (see page 9898)	Specifies the callback function that allows the dataset to communicate with the BDE constraint layer.
	ConstraintsDisabled (see page 9899)	Indicates whether constraints are enabled for a dataset.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
V	Create (see page 10053)	Creates an instance of a TNestedTable (see page 10040) component.
V	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
V	CreateBlobStream (see page 9899)	Returns a TStream object for reading or writing the data in a specified blob field.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 6904)	Destroys the TDataSet (see page 6894) instance.
V	Destroy (see page 9890)	Destroys the instance of a dataset component.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableConstraints (see page 9902)	Disables application of server constraints on the dataset.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableConstraints (see page 9902)	Enables application of server constraints against dataset entries.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FetchAll (see page 9903)	Retrieves all records from the current cursor position to the end of the file and stores them locally.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FieldByName (see page 6936)	Finds a field based on its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	FlushBuffers (see page 9903)	Posts all changes that have been written to the record buffer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
V	GetBlobFieldData (see page 9904)	Reads BLOB data into a buffer.
V	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
V	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
V	GetCurrentRecord (see page 9905)	Retrieves the current record into a buffer.
V	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
V	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
V	GetFieldData (see page 9906)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
V	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetIndexInfo (see page 9907)	Retrieves information about the current index into the index data fields of the dataset.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.

	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	IsSequenced (see page 9915)	IsSequenced indicates whether sequence numbers are available for database records.
	Last (see page 6968)	Moves to the last record in the dataset.
	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Locate (see page 9915)	Searches the dataset for a specified record and makes that record the current record.
	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	Lookup (see page 9916)	Retrieves field values from a record that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	Open (see page 6976)	Opens the dataset.
	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
	Post (see page 9918)	Writes a modified record to the database.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 7579)	Gets the SQL command to be executed by PSExecute (see page 7578).
	PSGetCommandType (see page 7580)	Gets the command type to be executed by PSExecute (see page 7578).
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7585)	Assigns a specified SQL command to be executed by PSExecute (see page 7578).
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)

V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
	RevertRecord (see page 9922)	Restores the current record in the dataset to an unmodified state when cached updates are enabled.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
V	Translate (see page 9928)	Converts a data string between the Microsoft ANSI character set, and the local code page (OEM character set).
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.
V	UpdateStatus (see page 9928)	Reports the update status for the current record.

Public Properties

	Name	Description
	Active (see page 7004)	Specifies whether or not a dataset is open.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
	CacheBlobs (see page 9932)	Determines whether BLOB fields are cached in memory.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	DataSource (see page 7014)	Represents the data source of another dataset that supplies values to the dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	ExpIndex (see page 9933)	Indicates whether a dataset is using a dBASE expression index.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
	FieldDefs (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
R	Handle (see page 9933)	Specifies the Borland (see page 1761) Database Engine (BDE) cursor handle for the dataset.

R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	KeySize (see page 9934)	Specifies the size of the key for the current index of the dataset.
R	Locale (see page 9934)	Identifies the Borland (see page 1761) Database Engine (BDE) language driver for the dataset.
R	Modified (see page 7029)	Indicates whether the active record is modified.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	UpdateObject (see page 9936)	Specifies the update object component used to update a read-only result set when cached updates are enabled.
	UpdateRecordTypes (see page 9936)	Specifies the type of records visible in a dataset when cached updates are enabled.
R	UpdatesPending (see page 9937)	Indicates whether the cached updates buffer contains records that are not yet applied.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterCancel (see page 9938)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 9940)	Occurs after an application closes a dataset.
	AfterDelete (see page 9942)	Occurs after an application deletes a record.
	AfterEdit (see page 9944)	Occurs after an application starts editing a record.
	AfterInsert (see page 9945)	Occurs after an application inserts a new record.
	AfterOpen (see page 9946)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 9947)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 9947)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 9948)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 9948)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 9949)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 9949)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 9950)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 9950)	Occurs before an application enters insert mode.
	BeforeOpen (see page 9952)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 9952)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 9953)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 9953)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 9954)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 9955)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 9955)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnNewRecord (see page 9955)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 9956)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnUpdateError (see page 9956)	Occurs if an exception is generated when cached updates are applied to a database.
	OnUpdateRecord (see page 9957)	Occurs when cached updates are applied to a record.

Published Properties

	Name	Description
	Active (see page 9929)	Specifies whether or not a dataset is open.
	AutoCalcFields (see page 9931)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	CachedUpdates (see page 9932)	Specifies whether cached updates are enabled for a dataset.
	DataSetField (see page 10054)	Indicates the persistent TDataSetField object that owns a nested dataset.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectView (see page 10055)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	Tag (see page 2808)	Stores an integer value as part of a component.

53.1.12.2 TNestedTable.Create Constructor

Creates an instance of a TNestedTable (see page 10040) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TNestedTable(TComponent * AOwner);
```

Description

Call Create to instantiate a nested table declared in an application if it was not placed on a form at design time. Create calls its inherited Create constructor and sets ObjectView (see page 10055) to True.

See Also

ObjectView (see page 7031)

53.1.12.3 TNestedTable Methods

The methods of the TNestedTable class are listed here.

Protected Methods

	Name	Description
	CreateHandle (see page 10053)	This is CreateHandle, a member of class TNestedTable.
	DoAfterPost (see page 10054)	This is DoAfterPost, a member of class TNestedTable.
	DoBeforeInsert (see page 10054)	This is DoBeforeInsert, a member of class TNestedTable.
	InternalPost (see page 10054)	This is InternalPost, a member of class TNestedTable.

53.1.12.3.1 TNestedTable.CreateHandle Method

Pascal

```
function CreateHandle: HDBICur; override;
```

C++

```
virtual __fastcall HDBICur CreateHandle();
```

Description

This is CreateHandle, a member of class TNestedTable.

53.1.12.3.2 TNestedTable.DoAfterPost Method

Pascal

```
procedure DoAfterPost; override;
```

C++

```
virtual __fastcall DoAfterPost();
```

Description

This is DoAfterPost, a member of class TNestedTable.

53.1.12.3.3 TNestedTable.DoBeforeInsert Method

Pascal

```
procedure DoBeforeInsert; override;
```

C++

```
virtual __fastcall DoBeforeInsert();
```

Description

This is DoBeforeInsert, a member of class TNestedTable.

53.1.12.3.4 TNestedTable.InternalPost Method

Pascal

```
procedure InternalPost; override;
```

C++

```
virtual __fastcall InternalPost();
```

Description

This is InternalPost, a member of class TNestedTable.

53.1.12.4 TNestedTable Properties

The properties of the TNestedTable class are listed here.

Published Properties

	Name	Description
	DataSetField (see page 10054)	Indicates the persistent TDataSetField object that owns a nested dataset.
	ObjectView (see page 10055)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.

53.1.12.4.1 TNestedTable.DataSetField Property

Indicates the persistent TDataSetField object that owns a nested dataset.

Pascal

```
property DataSetField: TDataSetField;
```

C++

```
__property TDataSetField DataSetField;
```

Description

Use DataSetField when the dataset is the nested detail set of another dataset, such as an Oracle8 nested dataset. Read DataSetField to access the data set field component in the master dataset that includes this dataset as a detail. Set DataSetField to a field component in the master dataset to assign this dataset as the value of that field.

Note: When setting the DataSetField property, the dataset must be of (or a descendant of) the class specified by the master dataset's NestedDataSetClass property. Otherwise, setting DataSetField raises an exception. Further, setting DataSetField to a TDataSetField that is contained in the same dataset raises an exception (A dataset can't be nested in itself).

See Also

TNestedTable (see page 10040)

NestedDataSet (see page 7070)

NestedDataSetClass (see page 7030)

Using Nested Detail Tables

Working with DataSet Fields

Working with Reference Fields

53.1.12.4.2 TNestedTable.ObjectView Property

Specifies whether fields are stored hierarchically or flattened out in the Fields property.

Pascal

```
property ObjectView: Boolean;
```

C++

```
__property Boolean ObjectView;
```

Description

ObjectView affects the way the Fields property stores object fields and the way data-aware grids display ADT and array fields.

When ObjectView is true, the fields are stored hierarchically in the Fields property, meaning any child fields of an object field are referenced by the object field and don't appear sequentially after the object field in the TFields:: .Fields array. When ObjectView is false, the fields are stored sequentially in the Fields property, meaning any child fields of an object field are stored after the object field as siblings in the Fields array.

When ObjectView is false, object field types, such as TADTField, are not created. This switch is provided for increased compatibility with older data-aware controls, which may not handle object field types properly. The default is false for TBDEDataSet, and true for TClientDataSet.

Note: If the ObjectView property of the dataset provided to a TClientDataSet is false, then the TClientDataSet's ObjectView property always acts as false, since the data received is already flattened.

See Also

FieldDefList (see page 7019)

Fields (see page 7023)

Displaying ADT and Array Fields

Using Nested Detail Tables

Using Object Fields

Working with ADT Fields

Working with Array Fields

53.1.13 TParamList Class

TParamList manages a list of session parameters.

Class Hierarchy

System::TObject → DBTables.TParamList

Pascal

```
TParamList = class(TObject);
```

C++

```
class TParamList : public TObject;
```

File

DBTables

Description

Do not use TParamList directly. TParamList is an internal object used to manage a list of session parameters for a TSession (see page 10093) component.

See Also

TSession (see page 10093)

53.1.13.1 TParamList Members

The following tables list the members exposed by TParamList.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 10057)	Creates an instance of TParamList (see page 10056).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 10057)	Destroys an instance of TParamList (see page 10056).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Buffer (see page 10058)	Contains all parameter strings for the parameter list.
	FieldCount (see page 10058)	Indicates the number of session parameters in the parameter list.
	FieldDescs (see page 10059)	Contains the field descriptors for each parameter maintained by the parameter list object.

53.1.13.2 TParamList.Create Constructor

Creates an instance of TParamList (see page 10056).

Pascal

```
constructor Create(Params: TStrings);
```

C++

```
__fastcall TParamList(TStrings Params);
```

Description

Call Create to instantiate a TParamList (see page 10056) object. Most applications do not need to create a TParamList (see page 10056) object because it is automatically created by TSession (see page 10093).

Params is a string list containing the names of the parameters to create for the session. These parameters correspond to Borland (see page 1761) Database Engine (BDE) configuration parameters.

See Also

TSession (see page 10093)

Destroy (see page 10057)

53.1.13.3 TParamList.Destroy Destructor

Destroys an instance of TParamList (see page 10056).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TParamList();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which checks for a nil reference before calling Destroy.

Destroy frees the memory previously allocated for field descriptors and the buffer used by TParamList (see page 10056).

Note: An application should never need to destroy a TParamList (see page 10056) object. It is automatically destroyed when its parent session object is destroyed.

See Also

Free (see page 22300)

TSession (see page 10093)

Create (see page 10057)

53.1.13.4 TParamList Properties

The properties of the TParamList class are listed here.

Public Properties

	Name	Description
R	Buffer (see page 10058)	Contains all parameter strings for the parameter list.
R	FieldCount (see page 10058)	Indicates the number of session parameters in the parameter list.
R	FieldDescs (see page 10059)	Contains the field descriptors for each parameter maintained by the parameter list object.

53.1.13.4.1 TParamList.Buffer Property

Contains all parameter strings for the parameter list.

Pascal

```
property Buffer: PChar;
```

C++

```
__property const char * Buffer;
```

Description

Buffer is used internally to store parameters in the the parameter list object. While applications can parse this list, there is usually no need to do so.

See Also

FieldDescs (see page 10059)

53.1.13.4.2 TParamList.FieldCount Property

Indicates the number of session parameters in the parameter list.

Pascal

```
property FieldCount: Integer;
```

C++

```
__property int FieldCount;
```

Description

Check (see page 10245) FieldCount to determine how many parameters there are for a session.

See Also

FieldDescs (see page 10059)

53.1.13.4.3 TParamList.FieldDescs Property

Contains the field descriptors for each parameter maintained by the parameter list object.

Pascal

```
property FieldDescs: TFieldDescList;
```

C++

```
__property TFieldDescList FieldDescs;
```

Description

Examine FieldDescs to determine the characteristics of each parameter maintained by the parameter list object. Applications usually have no need to access this property directly.

See Also

FieldCount (see page 10058)

53.1.14 TQuery Class

TQuery represents a dataset with a result set that is based on an SQL (see page 10089) statement.

Class Hierarchy

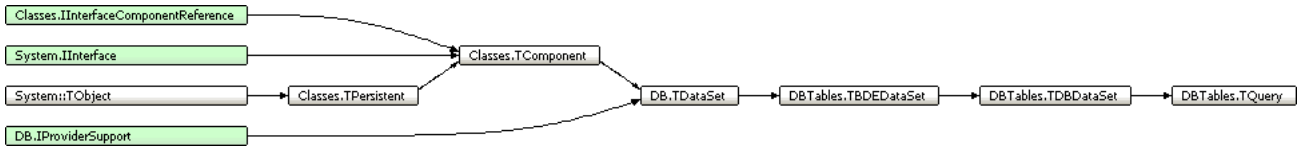

Pascal

```
TQuery = class(TDBDataSet);
```

C++

```
class TQuery : public TDBDataSet;
```

File

DBTables

Description

Use TQuery to access one or more tables in a database using SQL (see page 10089) statements. Query components can be used with remote database servers (such as Sybase, SQL (see page 10089) Server, Oracle, Informix, DB2, and InterBase), with local tables (Paradox, InterBase, dBASE, Access, and FoxPro), and with ODBC-compliant databases.

Query components are useful because they can

Access more than one table at a time (called a "join" in SQL (see page 10089)).

Automatically access a subset of rows and columns in its underlying table(s), rather than always returning all rows and columns.

Note: TQuery is of particular importance to the development of scalable database applications. If there is any chance that an application built to run against local databases will be scaled to a remote SQL (see page 10089) database server in the future, use TQuery components from the start to ensure easier scaling later.

See Also

TStoredProc (see page 10139)

TTable (see page 10168)

Creating Decision Datasets with TQuery or TTable

Types of Datasets

Using TQueryTableProducer

53.1.14.1 TQuery Members

The following tables list the members exposed by TQuery.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	ActivateFilters (see page 9893)	This is ActivateFilters, a member of class TBDEDataSet.
	AddFieldDesc (see page 9893)	This is AddFieldDesc, a member of class TBDEDataSet.
	AllocCachedUpdateBuffers (see page 9893)	This is AllocCachedUpdateBuffers, a member of class TBDEDataSet.
	AllocKeyBuffers (see page 9893)	This is AllocKeyBuffers, a member of class TBDEDataSet.
V	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
V	AllocRecordBuffer (see page 9894)	This is AllocRecordBuffer, a member of class TBDEDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
V	BlockReadNext (see page 9894)	This is BlockReadNext, a member of class TBDEDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
	CachedUpdateCallBack (see page 9895)	This is CachedUpdateCallBack, a member of class TBDEDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
	CheckCachedUpdateMode (see page 9896)	This is CheckCachedUpdateMode, a member of class TBDEDataSet.
V	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
V	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
	CheckSetKeyMode (see page 9896)	This is CheckSetKeyMode, a member of class TBDEDataSet.
V	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
V	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
V	ClearCalcFields (see page 9897)	This is ClearCalcFields, a member of class TBDEDataSet.
V	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.

	CloseBlob (see page 9897)	This is CloseBlob, a member of class TBDEDataSet.
	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
	CloseCursor (see page 9897)	This is CloseCursor, a member of class TBDEDataSet.
	CloseCursor (see page 10014)	This is CloseCursor, a member of class TDBDataSet.
	ConstraintsStored (see page 10015)	This is ConstraintsStored, a member of class TDBDataSet.
	CreateExprFilter (see page 9900)	This is CreateExprFilter, a member of class TBDEDataSet.
	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
	CreateFuncFilter (see page 9900)	This is CreateFuncFilter, a member of class TBDEDataSet.
	CreateHandle (see page 10074)	This is CreateHandle, a member of class TQuery.
	CreateLookupFilter (see page 9901)	This is CreateLookupFilter, a member of class TBDEDataSet.
	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
	DataEvent (see page 6924)	This is DataEvent, a member of class TDataSet.
	DataEvent (see page 9901)	This is DataEvent, a member of class TBDEDataSet.
	DeactivateFilters (see page 9901)	This is DeactivateFilters, a member of class TBDEDataSet.
	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
	DefineProperties (see page 10074)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
	DestroyHandle (see page 9901)	This is DestroyHandle, a member of class TBDEDataSet.
	DestroyLookupCursor (see page 9902)	This is DestroyLookupCursor, a member of class TBDEDataSet.
	Disconnect (see page 10075)	This is Disconnect, a member of class TQuery.
	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FindRecord (see page 9903)	This is FindRecord, a member of class TBDEDataSet.
	ForceUpdateCallback (see page 9904)	This is ForceUpdateCallback, a member of class TBDEDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeKeyBuffers (see page 9904)	This is FreeKeyBuffers, a member of class TBDEDataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
	FreeRecordBuffer (see page 9904)	This is FreeRecordBuffer, a member of class TBDEDataSet.
	FreeStatement (see page 10076)	This is FreeStatement, a member of class TQuery.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
	GetBookmarkData (see page 9905)	This is GetBookmarkData, a member of class TBDEDataSet.

	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
	GetBookmarkFlag (see page 9905)	This is GetBookmarkFlag, a member of class TBDEDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
	GetCanModify (see page 9905)	This is GetCanModify, a member of class TBDEDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
	GetDataSource (see page 10076)	This is GetDataSource, a member of class TQuery.
	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetFieldFullName (see page 9907)	This is GetFieldFullName, a member of class TBDEDataSet.
	GetHandle (see page 9907)	This is GetHandle, a member of class TBDEDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIndexField (see page 9907)	This is GetIndexField, a member of class TBDEDataSet.
	GetIndexFieldCount (see page 9907)	This is GetIndexFieldCount, a member of class TBDEDataSet.
	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
	GetIsIndexField (see page 9908)	This is GetIsIndexField, a member of class TBDEDataSet.
	GetKeyBuffer (see page 9908)	This is GetKeyBuffer, a member of class TBDEDataSet.
	GetKeyExclusive (see page 9908)	This is GetKeyExclusive, a member of class TBDEDataSet.
	GetKeyFieldCount (see page 9909)	This is GetKeyFieldCount, a member of class TBDEDataSet.
	GetLookupCursor (see page 9909)	This is GetLookupCursor, a member of class TBDEDataSet.
	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetObjectTypeNames (see page 9909)	This is GetObjectTypeNames, a member of class TBDEDataSet.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetParamsCount (see page 10076)	This is GetParamsCount, a member of class TQuery.
	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
	GetRecNo (see page 9909)	This is GetRecNo, a member of class TBDEDataSet.
	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
	GetRecord (see page 9910)	This is GetRecord, a member of class TBDEDataSet.
	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
	GetRecordCount (see page 9910)	This is GetRecordCount, a member of class TBDEDataSet.
	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
	GetRecordSize (see page 9910)	This is GetRecordSize, a member of class TBDEDataSet.
	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
	GetStateFieldValue (see page 9910)	This is GetStateFieldValue, a member of class TBDEDataSet.
	GetStatementHandle (see page 10077)	This is GetStatementHandle, a member of class TQuery.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetUpdateRecordSet (see page 9910)	This is GetUpdateRecordSet, a member of class TBDEDataSet.
	GetUpdatesPending (see page 9911)	This is GetUpdatesPending, a member of class TBDEDataSet.
	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
	InitKeyBuffer (see page 9911)	This is InitKeyBuffer, a member of class TBDEDataSet.
	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.

	InitRecord (see page 9911)	This is InitRecord, a member of class TBDEDataSet.
	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
	InternalAddRecord (see page 9911)	This is InternalAddRecord, a member of class TBDEDataSet.
	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
	InternalCancel (see page 9912)	This is InternalCancel, a member of class TBDEDataSet.
	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.
	InternalClose (see page 9912)	This is InternalClose, a member of class TBDEDataSet.
	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
	InternalDelete (see page 9912)	This is InternalDelete, a member of class TBDEDataSet.
	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
	InternalEdit (see page 9912)	This is InternalEdit, a member of class TBDEDataSet.
	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
	InternalFirst (see page 9912)	This is InternalFirst, a member of class TBDEDataSet.
	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
	InternalGotoBookmark (see page 9913)	This is InternalGotoBookmark, a member of class TBDEDataSet.
	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
	InternalHandleException (see page 9913)	This is InternalHandleException, a member of class TBDEDataSet.
	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
	InternalInitFieldDefs (see page 9913)	This is InternalInitFieldDefs, a member of class TBDEDataSet.
	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
	InternalInitRecord (see page 9913)	This is InternalInitRecord, a member of class TBDEDataSet.
	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
	InternalInsert (see page 9914)	This is InternalInsert, a member of class TBDEDataSet.
	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
	InternalLast (see page 9914)	This is InternalLast, a member of class TBDEDataSet.
	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
	InternalOpen (see page 9914)	This is InternalOpen, a member of class TBDEDataSet.
	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
	InternalPost (see page 9914)	This is InternalPost, a member of class TBDEDataSet.
	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
	InternalRefresh (see page 9914)	This is InternalRefresh, a member of class TBDEDataSet.
	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	InternalSetToRecord (see page 9915)	This is InternalSetToRecord, a member of class TBDEDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
	IsCursorOpen (see page 9915)	This is IsCursorOpen, a member of class TBDEDataSet.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 9916)	This is LocateRecord, a member of class TBDEDataSet.
	MapsToIndex (see page 9918)	This is MapsToIndex, a member of class TBDEDataSet.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	OpenCursor (see page 6976)	This is OpenCursor, a member of class TDataSet.
	OpenCursor (see page 9918)	This is OpenCursor, a member of class TBDEDataSet.
	OpenCursor (see page 10015)	This is OpenCursor, a member of class TDBDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PostKeyBuffer (see page 9919)	This is PostKeyBuffer, a member of class TBDEDataSet.
	PrepareCursor (see page 9919)	This is PrepareCursor, a member of class TBDEDataSet.
	ProcessUpdates (see page 9919)	This is ProcessUpdates, a member of class TBDEDataSet.
	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
	PSEndTransaction (see page 10016)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 10078)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 10017)	Executes a specified SQL command.

	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 10017)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 10078)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandType (see page 10079)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 10079)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetParams (see page 10079)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 10018)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetTableName (see page 10080)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSGetUpdateException (see page 9920)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 10018)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 10018)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 9920)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 9921)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 10080)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 10081)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSStartTransaction (see page 10019)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 9921)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 10019)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	ResetCursorRange (see page 9922)	This is ResetCursorRange, a member of class TBDEDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBlockReadSize (see page 9922)	This is SetBlockReadSize, a member of class TBDEDataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkData (see page 9922)	This is SetBookmarkData, a member of class TBDEDataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkFlag (see page 9923)	This is SetBookmarkFlag, a member of class TBDEDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.

	SetCachedUpdates (see page 9923)	This is SetCachedUpdates, a member of class TBDEDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
V	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetCursorRange (see page 9923)	This is SetCursorRange, a member of class TBDEDataSet.
V	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
V	SetDBFlag (see page 10081)	This is SetDBFlag, a member of class TQuery.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetFieldData (see page 9923)	This is SetFieldData, a member of class TBDEDataSet.
V	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
	SetFilterData (see page 9923)	This is SetFilterData, a member of class TBDEDataSet.
V	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
V	SetFiltered (see page 9924)	This is SetFiltered, a member of class TBDEDataSet.
	SetFilterHandle (see page 9924)	This is SetFilterHandle, a member of class TBDEDataSet.
V	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
V	SetFilterOptions (see page 9924)	This is SetFilterOptions, a member of class TBDEDataSet.
V	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
V	SetFilterText (see page 9924)	This is SetFilterText, a member of class TBDEDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetHandle (see page 10020)	This is SetHandle, a member of class TDBDataSet.
	SetIndexField (see page 9925)	This is SetIndexField, a member of class TBDEDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetKeyBuffer (see page 9925)	This is SetKeyBuffer, a member of class TBDEDataSet.
	SetKeyExclusive (see page 9925)	This is SetKeyExclusive, a member of class TBDEDataSet.
	SetKeyFieldCount (see page 9925)	This is SetKeyFieldCount, a member of class TBDEDataSet.
	SetKeyFields (see page 9925)	This is SetKeyFields, a member of class TBDEDataSet.
	SetLinkRanges (see page 9926)	This is SetLinkRanges, a member of class TBDEDataSet.
	SetLocale (see page 9926)	This is SetLocale, a member of class TBDEDataSet.
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
V	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
V	SetOnFilterRecord (see page 9926)	This is SetOnFilterRecord, a member of class TBDEDataSet.
	SetOnUpdateError (see page 9926)	This is SetOnUpdateError, a member of class TBDEDataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
V	SetRecNo (see page 9927)	This is SetRecNo, a member of class TBDEDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
V	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
V	SetStateFieldValue (see page 9927)	This is SetStateFieldValue, a member of class TBDEDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	SetupCallBack (see page 9927)	This is SetupCallBack, a member of class TBDEDataSet.
	SetUpdateObject (see page 9927)	This is SetUpdateObject, a member of class TBDEDataSet.
	SetUpdateRecordSet (see page 9927)	This is SetUpdateRecordSet, a member of class TBDEDataSet.
	SwitchToIndex (see page 9928)	This is SwitchToIndex, a member of class TBDEDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	UpdateCallbackRequired (see page 9928)	This is UpdateCallbackRequired, a member of class TBDEDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

V	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
R	DataLink (see page 10084)	This is DataLink, a member of class TQuery.
R	DBFlags (see page 10023)	Keeps track of the commands this dataset has active with the database.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
	Reserved (see page 7035)	For internal use only.

Public Events

	Name	Description
	AfterCancel (see page 7038)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 7040)	Occurs after an application closes a dataset.
	AfterDelete (see page 7041)	Occurs after an application deletes a record.
	AfterEdit (see page 7044)	Occurs after an application starts editing a record.
	AfterInsert (see page 7044)	Occurs after an application inserts a new record.
	AfterOpen (see page 7045)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 7046)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 7047)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 7047)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 7048)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 7048)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 7049)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 7049)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 7050)	Occurs before an application enters insert mode.
	BeforeOpen (see page 7051)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 7052)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 7052)	Occurs immediately before an application refreshes the data in the dataset.

	BeforeScroll (see page 7053)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 7053)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 7054)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 7054)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 7056)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 7056)	Occurs when an application attempts to modify or insert a record and an exception is raised.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
	ApplyUpdates (see page 9894)	Writes a dataset's pending cached updates to the database.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
V	BookmarkValid (see page 9895)	Tests the validity of a specified bookmark.
V	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
V	Cancel (see page 9895)	Cancels modifications to the current record if those changes are not yet posted.
	CancelUpdates (see page 9896)	Clears all pending cached updates from the cache and restores the dataset its prior state.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	CheckOpen (see page 10014)	Checks the result of a call to the Borland (see page 1761) Database (see page 10021) Engine (BDE).
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Close (see page 6920)	Closes a dataset.
	CloseDatabase (see page 10014)	Closes a database connection associated with the database session.
	CommitUpdates (see page 9897)	Clears the cached updates buffer.
V	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
V	CompareBookmarks (see page 9898)	Indicates the relationship between two bookmarks.
	ConstraintCallBack (see page 9898)	Specifies the callback function that allows the dataset to communicate with the BDE constraint layer.
	ConstraintsDisabled (see page 9899)	Indicates whether constraints are enabled for a dataset.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
V	Create (see page 10073)	Creates an instance of a query component.
V	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
V	CreateBlobStream (see page 9899)	Returns a TStream object for reading or writing the data in a specified blob field.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.

	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
	Destroy (see page 10073)	Destroys the instance of a query.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableConstraints (see page 9902)	Disables application of server constraints on the dataset.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableConstraints (see page 9902)	Enables application of server constraints against dataset entries.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	ExecSQL (see page 10075)	Executes the SQL (see page 10089) statement for the query.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FetchAll (see page 9903)	Retrieves all records from the current cursor position to the end of the file and stores them locally.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	FlushBuffers (see page 9903)	Posts all changes that have been written to the record buffer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
	GetBlobFieldData (see page 9904)	Reads BLOB data into a buffer.
	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
	GetCurrentRecord (see page 9905)	Retrieves the current record into a buffer.
	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
	GetDetailLinkFields (see page 10076)	Fills lists with the master and detail fields of the link.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
	GetFieldData (see page 9906)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetIndexInfo (see page 9907)	Retrieves information about the current index into the index data fields of the dataset.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	IsSequenced (see page 9915)	IsSequenced indicates whether sequence numbers are available for database records.
	Last (see page 6968)	Moves to the last record in the dataset.
	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Locate (see page 9915)	Searches the dataset for a specified record and makes that record the current record.
	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	Lookup (see page 9916)	Retrieves field values from a record that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	Open (see page 6976)	Opens the dataset.
	OpenDatabase (see page 10016)	Opens the database that contains the dataset.
	ParamByName (see page 10077)	Accesses parameter information based on a specified parameter name.
	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
	Post (see page 9918)	Writes a modified record to the database.
	Prepare (see page 10077)	Sends a query to the Borland (see page 1761) Database (see page 10021) Engine (BDE) and the server for optimization prior to execution.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.

	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
	RevertRecord (see page 9922)	Restores the current record in the dataset to an unmodified state when cached updates are enabled.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
V	Translate (see page 9928)	Converts a data string between the Microsoft ANSI character set, and the local code page (OEM character set).
	UnPrepare (see page 10081)	Frees the resources allocated for a previously prepared query.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.
V	UpdateStatus (see page 9928)	Reports the update status for the current record.

Public Properties

	Name	Description
	Active (see page 7004)	Specifies whether or not a dataset is open.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
	CacheBlobs (see page 9932)	Determines whether BLOB fields are cached in memory.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Database (see page 10021)	Identifies the database component for which this dataset represents one or more tables.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
R	DBHandle (see page 10023)	Specifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) database handle for the dataset.
R	DBLocale (see page 10024)	Identifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) language driver for the dataset component.
R	DBSession (see page 10024)	Identifies the database session component associated with this dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.

R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	ExpIndex (see page 9933)	Indicates whether a dataset is using a dBASE expression index.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
	FieldDefs (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
R	Handle (see page 9933)	Specifies the Borland (see page 1761) Database Engine (BDE) cursor handle for the dataset.
	Handle (see page 10027)	Specifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) cursor handle for the dataset.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	KeySize (see page 9934)	Specifies the size of the key for the current index of the dataset.
R	Local (see page 10085)	Indicates whether a query references one or more Paradox or dBASE tables rather than SQL (see page 10089) tables on a remote server.
R	Locale (see page 9934)	Identifies the Borland (see page 1761) Database Engine (BDE) language driver for the dataset.
R	Modified (see page 7029)	Indicates whether the active record is modified.
	ObjectView (see page 7031)	Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	ParamCount (see page 10086)	Indicates the current number of parameters for the query.
	Prepared (see page 10087)	Determines whether or not a query is prepared for execution.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
R	RowsAffected (see page 10089)	Returns the number of rows operated upon by the latest query execution.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
	SQLBinary (see page 10090)	Points to the binary data stream that represents an SQL (see page 10089) query statement or result set.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	StmtHandle (see page 10091)	Specifies the Borland Database Engine (BDE) statement handle for the dataset.
R	Text (see page 10091)	Points to the actual text of the SQL (see page 10089) query passed to the Borland (see page 1761) Database (see page 10021) Engine (BDE).
	UpdateRecordTypes (see page 9936)	Specifies the type of records visible in a dataset when cached updates are enabled.
R	UpdatesPending (see page 9937)	Indicates whether the cached updates buffer contains records that are not yet applied.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterCancel (see page 9938)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 9940)	Occurs after an application closes a dataset.
	AfterDelete (see page 9942)	Occurs after an application deletes a record.
	AfterEdit (see page 9944)	Occurs after an application starts editing a record.
	AfterInsert (see page 9945)	Occurs after an application inserts a new record.

	AfterOpen (see page 9946)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 9947)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 9947)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 9948)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 9948)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 9949)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 9949)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 9950)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 9950)	Occurs before an application enters insert mode.
	BeforeOpen (see page 9952)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 9952)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 9953)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 9953)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 9954)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 9955)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 9955)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 10028)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 9955)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 9956)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnUpdateError (see page 9956)	Occurs if an exception is generated when cached updates are applied to a database.
	OnUpdateRecord (see page 9957)	Occurs when cached updates are applied to a record.

Published Properties

	Name	Description
	Active (see page 9929)	Specifies whether or not a dataset is open.
	AutoCalcFields (see page 9931)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	AutoRefresh (see page 10021)	Specifies whether server-generated field values are refetched automatically.
	CachedUpdates (see page 9932)	Specifies whether cached updates are enabled for a dataset.
	Constrained (see page 10083)	Indicates whether updates and inserts that don't conform to the result set generated by a SELECT statement are permitted for Paradox and dBASE tables
	Constraints (see page 10083)	Specifies record-level constraints that must be met when editing the data.
	DatabaseName (see page 10022)	Specifies the name of the database associated with this dataset.
	DataSource (see page 10084)	Specifies the data source component from which to extract current field values to use with same-name parameters in the query's SQL (see page 10089) statement.
	Filter (see page 10025)	Specifies the text of the current filter for a dataset.
	Filtered (see page 10026)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 10026)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectView (see page 9935)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	ParamCheck (see page 10086)	Specifies whether the parameter list for a query is regenerated if the SQL (see page 10089) property changes at runtime.
	Params (see page 10087)	Contains the parameters for a query's SQL (see page 10089) statement.
	RequestLive (see page 10088)	Requests an updatable query result set from the database.
	SessionName (see page 10027)	Identifies the name of the session with which this dataset is associated.
	SQL (see page 10089)	Contains the text of the SQL statement to execute for the query.
	Tag (see page 2808)	Stores an integer value as part of a component.

	UniDirectional (see page 10092)	Determines whether or not Borland (see page 1761) Database (see page 10021) Engine (BDE) bidirectional cursors are enabled for a query's result set.
	UpdateMode (see page 10092)	Determines how the Borland Database Engine (BDE) finds records when updating to an SQL database.
	UpdateObject (see page 10092)	Specifies the update object component used to update a read-only result set when cached updates are enabled.

53.1.14.2 TQuery.Create Constructor

Creates an instance of a query component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TQuery(TComponent * AOwner);
```

Description

Call Create to instantiate a query at runtime. Query components placed in forms or data modules at design time are created automatically.

AOwner is the component that is responsible for freeing the query. It becomes the value of the Owner property.

See Also

Destroy (see page 10073)

Owner (see page 2807)

53.1.14.3 TQuery.Destroy Destructor

Destroys the instance of a query.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TQuery();
```

Description

Do not call Destroy directly. Instead, call Free, which checks that the TQuery (see page 10059) is not nil before calling Destroy.

Destroy disconnects from the server, frees the SQL (see page 10089) statement list, the parameter list, and the data link and SQL (see page 10089) binary storage area, and then calls its inherited destructor.

See Also

Create (see page 10073)

Free (see page 22300)

53.1.14.4 TQuery Methods

The methods of the TQuery class are listed here.

Protected Methods

	Name	Description
	CreateHandle (see page 10074)	This is CreateHandle, a member of class TQuery.
	DefineProperties (see page 10074)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	Disconnect (see page 10075)	This is Disconnect, a member of class TQuery.
	FreeStatement (see page 10076)	This is FreeStatement, a member of class TQuery.
	GetDataSource (see page 10076)	This is GetDataSource, a member of class TQuery.
	GetParamsCount (see page 10076)	This is GetParamsCount, a member of class TQuery.
	GetStatementHandle (see page 10077)	This is GetStatementHandle, a member of class TQuery.
	PSExecute (see page 10078)	Executes the SQL command associated with the dataset.
	PSGetCommandText (see page 10078)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandType (see page 10079)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 10079)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetParams (see page 10079)	Returns the current parameter values of the dataset.
	PSGetTableName (see page 10080)	Returns the name of the dataset table that appears in generated SQL statements.
	PSSetCommandText (see page 10080)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetParams (see page 10081)	Assigns the specified parameter values to the dataset.
	SetDBFlag (see page 10081)	This is SetDBFlag, a member of class TQuery.

Public Methods

	Name	Description
	ExecSQL (see page 10075)	Executes the SQL (see page 10089) statement for the query.
	GetDetailLinkFields (see page 10076)	Fills lists with the master and detail fields of the link.
	ParamByName (see page 10077)	Accesses parameter information based on a specified parameter name.
	Prepare (see page 10077)	Sends a query to the Borland (see page 1761) Database (see page 10021) Engine (BDE) and the server for optimization prior to execution.
	UnPrepare (see page 10081)	Frees the resources allocated for a previously prepared query.

53.1.14.4.1 TQuery.CreateHandle Method

Pascal

```
function CreateHandle: HDBICur; override;
```

C++

```
virtual __fastcall HDBICur CreateHandle();
```

Description

This is CreateHandle, a member of class TQuery.

53.1.14.4.2 TQuery.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are

hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor (see page 2839)

DefineBinaryProperty (see page 2837)

DefineProperties (see page 2931)

DefineProperty (see page 2838)

53.1.14.4.3 TQuery.Disconnect Method

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

This is Disconnect, a member of class TQuery.

53.1.14.4.4 TQuery.ExecSQL Method

Executes the SQL (see page 10089) statement for the query.

Pascal

```
procedure ExecSQL;
```

C++

```
__fastcall ExecSQL();
```

Description

Call ExecSQL to execute the SQL (see page 10089) statement currently assigned to the SQL (see page 10089) property. Use ExecSQL to execute queries that do not return a cursor to data (such as INSERT, UPDATE, DELETE, and CREATE TABLE).

Note: For SELECT statements, call Open instead of ExecSQL.

ExecSQL prepares the statement in SQL (see page 10089) property for execution if it has not already been prepared. To speed performance, an application should ordinarily call Prepare (see page 10077) before calling ExecSQL for the first time.

See Also

Open (see page 6976)

Prepare (see page 10077)

SQL (see page 10089)

Executing the Commands using TSQLDataSet

Using TQuery

53.1.14.4.5 TQuery.FreeStatement Method

Pascal

```
procedure FreeStatement; virtual;
```

C++

```
virtual __fastcall FreeStatement();
```

Description

This is FreeStatement, a member of class TQuery.

53.1.14.4.6 TQuery.GetDataSource Method

Pascal

```
function GetDataSource: TDataSource; override;
```

C++

```
virtual __fastcall TDataSource GetDataSource();
```

Description

This is GetDataSource, a member of class TQuery.

53.1.14.4.7 TQuery.GetDetailLinkFields Method

Fills lists with the master and detail fields of the link.

Pascal

```
procedure GetDetailLinkFields(MasterFields: TList; DetailFields: TList); override;
```

C++

```
virtual __fastcall GetDetailLinkFields(TList MasterFields, TList DetailFields);
```

Description

Creates two lists of TFields from the master-detail relationship between two tables; one containing the master fields, and the other containing the detail fields.

See Also

Params (see page 10087)

53.1.14.4.8 TQuery.GetParamsCount Method

Pascal

```
function GetParamsCount: Word;
```

C++

```
__fastcall Word GetParamsCount();
```

Description

This is GetParamsCount, a member of class TQuery.

53.1.14.4.9 TQuery.GetStatementHandle Method

Pascal

```
procedure GetStatementHandle(SQLText: PChar); virtual;
```

C++

```
virtual __fastcall GetStatementHandle(const char * SQLText);
```

Description

This is GetStatementHandle, a member of class TQuery.

53.1.14.4.10 TQuery.ParamByName Method

Accesses parameter information based on a specified parameter name.

Pascal

```
function ParamByName(const Value: string): TParam;
```

C++

```
__fastcall TParam ParamByName(const AnsiString Value);
```

Description

Call ParamByName to set or use parameter information for a specific parameter based on its name.

Value is the name of the parameter for which to retrieve information.

ParamByName is primarily used to set an parameter's value at runtime. For example, the following statement retrieves the current value of a parameter called "Contact" into an edit box:

```
Edit1.Text := Query1.ParamByName('Contact').AsString;
Edit1->Text = Query1->ParamByName("Contact")->AsString;
```

Parameters used in SELECT statements cannot be NULL, but they can be NULL for UPDATE and INSERT statements.

See Also

Params (see page 10087)

53.1.14.4.11 TQuery.Prepare Method

Sends a query to the Borland (see page 1761) Database (see page 10021) Engine (BDE) and the server for optimization prior to execution.

Pascal

```
procedure Prepare;
```

C++

```
__fastcall Prepare();
```

Description

Call Prepare to have the BDE and a remote database server allocate resources for the query and to perform additional optimizations. Preparing a query consumes some database resources, so it is good practice to call UnPrepare (see page 10081) once the query is no longer needed.

If the query will only be executed once, the application does not need to explicitly call Prepare or UnPrepare (see page 10081). Executing an unprepared query generates these calls automatically. However, if the same query is to be executed

repeatedly, it is more efficient to prevent these automatic calls by calling Prepare and UnPrepare (see page 10081) explicitly.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also

ExecSQL (see page 10075)

Open (see page 6976)

UnPrepare (see page 10081)

Using TQuery

53.1.14.4.12 TQuery.PSExecute Method

Executes the SQL command associated with the dataset.

Pascal

```
procedure PSExecute; override;
```

C++

```
virtual __fastcall PSExecute();
```

Description

The provider component calls PSExecute to implement its Execute method. If the dataset represents a query or stored procedure that does not return a cursor, this method executes the query or stored procedure. If the provider Options includes poAllowCommandText and the dataset supports the PSSetCommandText method, the statement specified by PSSetCommandText may be executed instead.

If there is not statement to execute, PSExecute raises an EDatabaseError exception.

See Also

PSSetCommandText (see page 7585)

Execute (see page 18466)

Options (see page 18440)

53.1.14.4.13 TQuery.PSGetCommandText Method

Gets the SQL command to be executed by PSExecute.

Pascal

```
function PSGetCommandText: string; override;
```

C++

```
virtual __fastcall AnsiString PSGetCommandText();
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

PSSetCommandText (see page 7585)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

53.1.14.4.14 TQuery.PSGetCommandType Method

Gets the command type to be executed by PSExecute.

Pascal

```
function PSGetCommandType: TPSCommandType; override;
```

C++

```
virtual __fastcall TPSCommandType PSGetCommandType();
```

Description

PSGetCommandType gets the command type executed by PSExecute. The type may be a type of SQL command or other command types, such as stored procedures.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

PSGetCommandText (see page 7579)

Execute (see page 18466)

Options (see page 18440)

53.1.14.4.15 TQuery.PSGetDefaultOrder Method

Returns the definition of an index that imposes the default order on the data included in data packets.

Pascal

```
function PSGetDefaultOrder: TIndexDef; override;
```

C++

```
virtual __fastcall TIndexDef PSGetDefaultOrder();
```

Description

The provider component calls PSGetDefaultOrder to generate a default index, called DEFAULT_ORDER, for the records that appear in data packets. This index defines the default sort order for records when they appear in the client dataset on the client.

PSGetDefaultOrder creates a TIndexDef object to describe the default index. The provider that calls PSGetDefaultOrder subsequently frees this TIndexDef object.

See Also

CreateDataPacket (see page 18518)

53.1.14.4.16 TQuery.PSGetParams Method

Returns the current parameter values of the dataset.

Pascal

```
function PSGetParams: TParams; override;
```

C++

```
virtual __fastcall TParams PSGetParams();
```

Description

The provider component calls PSGetParams to implement its GetParams method.

PSGetParams returns a TParams object that describes the current values of all parameters of the dataset. If the dataset does not have any parameters, PSGetParams returns nil (Delphi) or NULL (C++).

Note: The PSGetParams method should not create a new instance of TParams for the return value. The provider that calls PSGetParams does not free the returned value.

See Also

GetParams (see page 18467)

53.1.14.4.17 TQuery.PSGetTableName Method

Returns the name of the dataset table that appears in generated SQL statements.

Pascal

```
function PSGetTableName: string; override;
```

C++

```
virtual __fastcall AnsiString PSGetTableName();
```

Description

PSGetTableName returns the name of the underlying database table against which updates are applied when the provider's ResolveToDataSet property is false. This value is used for generating the SQL statements that apply updates. The generated SQL statements are subsequently executed by a call to PSExecuteStatement.

If PSGetTableName does not return a table name, the provider must have an OnGetTableName event handler to supply this value.

See Also

applyupdates (see page 18459)

PSExecuteStatement (see page 7579)

ResolveToDataSet (see page 18531)

ongettablename (see page 18542)

53.1.14.4.18 TQuery.PSSetCommandText Method

Assigns a specified SQL command to be executed by PSExecute.

Pascal

```
procedure PSSetCommandText(const CommandText: string); override;
```

C++

```
virtual __fastcall PSSetCommandText(const AnsiString CommandText);
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

The value of the CommandText parameter replaces the dataset's SQL statement (if it has one), or replaces the name of an underlying database table or stored procedure.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

53.1.14.4.19 TQuery.PSSetParams Method

Assigns the specified parameter values to the dataset.

Pascal

```
procedure PSSetParams(AParams: TParams); override;
```

C++

```
virtual __fastcall PSSetParams(TParams AParams);
```

Description

PSSetParams assigns the parameter values specified by AParams to the parameters of the dataset.

See Also

Params (see page 8489)

PSGetParams (see page 7582)

53.1.14.4.20 TQuery.SetDBFlag Method

Pascal

```
function SetDBFlag(Flag: Integer; Value: Boolean): Boolean; override;
```

C++

```
virtual __fastcall Boolean SetDBFlag(int Flag, Boolean Value);
```

Description

This is SetDBFlag, a member of class TQuery.

53.1.14.4.21 TQuery.UnPrepare Method

Frees the resources allocated for a previously prepared query.

Pascal

```
procedure UnPrepare;
```

C++

```
__fastcall UnPrepare();
```

Description

Call UnPrepare to free the resources allocated for a previously prepared query on the server and client sides.

Preparing a query consumes some database resources, so it is good practice for an application to unprepare a query once it is done using it. The UnPrepare method unprepares a query.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also

ExecSQL (see page 10075)

Open (see page 6976)

Prepare (see page 10077)

53.1.14.5 TQuery Properties

The properties of the TQuery class are listed here.

Protected Properties

	Name	Description
R	DataLink (see page 10084)	This is DataLink, a member of class TQuery.

Public Properties

	Name	Description
R	Local (see page 10085)	Indicates whether a query references one or more Paradox or dBASE tables rather than SQL (see page 10089) tables on a remote server.
R	ParamCount (see page 10086)	Indicates the current number of parameters for the query.
	Prepared (see page 10087)	Determines whether or not a query is prepared for execution.
R	RowsAffected (see page 10089)	Returns the number of rows operated upon by the latest query execution.
	SQLBinary (see page 10090)	Points to the binary data stream that represents an SQL (see page 10089) query statement or result set.
	StmtHandle (see page 10091)	Specifies the Borland Database Engine (BDE) statement handle for the dataset.
R	Text (see page 10091)	Points to the actual text of the SQL (see page 10089) query passed to the Borland (see page 1761) Database (see page 10021) Engine (BDE).

Published Properties

	Name	Description
	Constrained (see page 10083)	Indicates whether updates and inserts that don't conform to the result set generated by a SELECT statement are permitted for Paradox and dBASE tables
	Constraints (see page 10083)	Specifies record-level constraints that must be met when editing the data.
	DataSource (see page 10084)	Specifies the data source component from which to extract current field values to use with same-name parameters in the query's SQL (see page 10089) statement.
	ParamCheck (see page 10086)	Specifies whether the parameter list for a query is regenerated if the SQL (see page 10089) property changes at runtime.
	Params (see page 10087)	Contains the parameters for a query's SQL (see page 10089) statement.
	RequestLive (see page 10088)	Requests an updatable query result set from the database.
	SQL (see page 10089)	Contains the text of the SQL statement to execute for the query.
	UniDirectional (see page 10092)	Determines whether or not Borland (see page 1761) Database (see page 10021) Engine (BDE) bidirectional cursors are enabled for a query's result set.
	UpdateMode (see page 10092)	Determines how the Borland Database Engine (BDE) finds records when updating to an SQL database.

	UpdateObject (see page 10092)	Specifies the update object component used to update a read-only result set when cached updates are enabled.

53.1.14.5.1 TQuery.Constrained Property

Indicates whether updates and inserts that don't conform to the result set generated by a SELECT statement are permitted for Paradox and dBASE tables

Pascal

```
property Constrained: Boolean;
```

C++

```
__property Boolean Constrained;
```

Description

Use Constrained to determine whether or not edits and insertions are permitted to a Paradox or dBASE table underlying a result set when the UPDATE or INSERT statements contain one or more values that do not comply with the WHERE clause conditions of the SELECT statement that generated the result set. Constrained only applies to live result sets. Constraints (see page 10083) are enforced during the editing process.

If Constrained is false (the default), updates and inserts are permitted regardless of the SELECT statement criteria. If true, updates and inserts are not permitted if they violate the criteria of the SELECT statement that generated the result set.

For example, if Constrained is true and a query component's SQL (see page 10089) property is

```
SELECT *
FROM CUSTOMERS
WHERE (CUSTNO > 1400) AND (CUSTNO < 1500)
```

then the user would not be allowed to enter a value for CustNo that fell outside this range of values.

See Also

SQL (see page 10089)

53.1.14.5.2 TQuery.Constraints Property

Specifies record-level constraints that must be met when editing the data.

Pascal

```
property Constraints: TCheckConstraints;
```

C++

```
__property TCheckConstraints Constraints;
```

Description

Use Constraints to read or add record-level constraints to the dataset. Record-level constraints usually impose relationships between the fields in a single record. Constraints that limit the value of a single field should be added to the individual field components.

See Also

TCheckConstraint (see page 6841)

CustomConstraint (see page 7156)

ImportedConstraint (see page 7167)

Constraining Data Values

53.1.14.5.3 TQuery.DataLink Property

Pascal

```
property DataLink: TDataLink;
```

C++

```
__property TDataLink DataLink;
```

Description

This is DataLink, a member of class TQuery.

53.1.14.5.4 TQuery.DataSource Property

Specifies the data source component from which to extract current field values to use with same-name parameters in the query's SQL (see page 10089) statement.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Set DataSource to automatically fill parameters in a query with fields values from another dataset. Parameters that have the same name as fields in the other dataset are filled with the field values. Parameters with names that are not the same as fields in the other dataset do not automatically get values, and must be programmatically set. For example, if the SQL (see page 10089) property of the TQuery (see page 10059) contains the SQL (see page 10089) statement below and the dataset referenced through DataSource has a CustNo field, the value from the current record in that other dataset is used in the CustNo parameter.

```
SELECT *
FROM Orders O
WHERE (O.CustNo = :CustNo)
```

DataSource must point to a TDataSource component linked to another dataset component; it cannot point to this query's data source component.

The dataset specified in DataSource must be created, populated, and opened before attempting to bind parameters. Parameters are bound by calling the query's Prepare (see page 10077) method prior to executing the query.

Tip: DataSource is especially of use when creating a master-detail relationship between tables using a linked query. It is also of use to guarantee binding for parameters that are not already set in the Params (see page 10087) property or through a call to the ParamByName (see page 10077) method.

If the SQL (see page 10089) statement used by a query does not contain parameters, or all parameters are bound by the application using the Params (see page 10087) property or the ParamByName (see page 10077) method, DataSource need not be assigned. The example below shows setting the DataSource property of Query2 to the data source for Query1, preparing Query2, and activating Query2.

```
with Query2 do begin
  DataSource := DataSource1;
  Prepare;
  Open;
end;
Query2->DataSource = DataSource1;
Query2->Prepare();
Query2->Open();
```

If the SQL (see page 10089) statement in the TQuery (see page 10059) is a SELECT query, the query is executed using the new field values each time the record pointer in the other dataset is changed. It is not necessary to call the Open method of the TQuery (see page 10059) each time. This makes using the DataSource property to dynamically filter a query result set useful for establishing Master-Detail relationships. Set the DataSource property in the Detail query to the TDataSource component for the Master dataset.

If the SQL (see page 10089) statement uses other than a SELECT query (such as INSERT or UPDATE), the parameters with the same name as fields in the other dataset still get values, but the query must be explicitly executed each time the other dataset's record pointer moves. For example, the SQL (see page 10089) statement below uses the INSERT statement and has the parameters CustNo and CompanyName.

```
INSERT INTO Customer
(CustNo, Company)
VALUES (:CustNo, :CompanyName)
```

Another dataset, Query1 and DataSource1, has a CustNo field but no CompanyName field. If this dataset is used through the DataSource property, the CompanyName parameter must be programmatically assigned a value. Because Query1 has a CustNo field and Query1 is referenced through the DataSource property, the CustNo parameter automatically receives a value.

```
with Query2 do begin
  DataSource := DataSource1;
  ParamByName('CompanyName').AsString := Edit1.Text;
  Prepare;
  ExecSQL;
end;
Query2->DataSource = DataSource1;
Query2->ParamByName("CompanyName")->AsString = Edit1->Text;
Query2->Prepare();
Query2->ExecSQL();
```

If the SQL (see page 10089) statement contains parameters with the same name as fields in the other dataset, do not manually set values for these parameters. Any values programmatically set, such as by using the Params (see page 10087) property or the ParamByName (see page 10077) method, will be overridden with automatic values. Parameters of other names must be programmatically given values. These parameters are unaffected by setting DataSource.

DataSource can be set at runtime or at design-time using the Object Inspector. At design-time, select the desired TDataSource from the drop-down list or type in the name.

See Also

ParamByName (see page 10077)

Params (see page 10087)

SQL (see page 10089)

53.1.14.5.5 TQuery.Local Property

Indicates whether a query references one or more Paradox or dBASE tables rather than SQL (see page 10089) tables on a remote server.

Pascal

```
property Local: Boolean;
```

C++

```
__property Boolean Local;
```

Description

Check (see page 10245) Local to determine if a query accesses one or more Paradox or dBASE tables rather than SQL (see page 10089) tables. If Local is true, then the query only accesses dBASE or Paradox tables. Otherwise Local is false. Local

is set automatically when an application calls the query's Prepare (see page 10077) method prior to executing the query.

Some query component settings, such as Constrained (see page 10083), are only meaningful for queries against Paradox and dBASE tables. Others are only meaningful for queries against remote SQL (see page 10089) servers. Local can provide useful information for branching logic in code based on the type of database accessed. It can also be used to determine whether or not the BDE parsed a query using its local query engine rather than passing SQL (see page 10089) statements directly to a remote server.

See Also

Constrained (see page 10083)

ExecSQL (see page 10075)

Open (see page 6976)

Prepare (see page 10077)

53.1.14.5.6 TQuery.ParamCheck Property

Specifies whether the parameter list for a query is regenerated if the SQL (see page 10089) property changes at runtime.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

Set ParamCheck to specify whether or not the Params (see page 10087) property is cleared and regenerated if an application modifies the query's SQL (see page 10089) property at runtime. By default ParamCheck is true, meaning that the Params (see page 10087) property is automatically regenerated at runtime. When ParamCheck is true, the proper number of parameters is guaranteed to be generated for the current SQL (see page 10089) statement.

This property is useful for data definition language (DDL) statements that contain parameters as part of the DDL statement and that are not parameters for the TQuery (see page 10059). For example, the DDL statement to create a stored procedure may contain parameter statements that are part of the stored procedure. Set ParamCheck to false to prevent these parameters from being mistaken for parameters of the TQuery (see page 10059) executing the DDL statement.

An application that does not use parameterized queries may choose to set ParamCheck to false, but otherwise ParamCheck should be true.

See Also

Params (see page 10087)

SQL (see page 10089)

TStoredProc (see page 10139)

53.1.14.5.7 TQuery.ParamCount Property

Indicates the current number of parameters for the query.

Pascal

```
property ParamCount: Word;
```

C++

```
__property Word ParamCount;
```

Description

Inspect ParamCount to determine how many parameters are in the Params (see page 10087) property. If the ParamCheck (see page 10086) property is true, ParamCount always corresponds to the number of actual parameters in the SQL (see page 10089) statement for the query.

See Also

Params (see page 10087)

SQL (see page 10089)

53.1.14.5.8 TQuery.Params Property

Contains the parameters for a query's SQL (see page 10089) statement.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Access Params at runtime to view and set parameter names, values, and data types dynamically (at design time use the collection editor for the Params property to set parameter information). Params is a zero-based array of TParams parameter records. Index specifies the array element to access.

Note: An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName (see page 10077). ParamByName (see page 10077) cannot, however, be used to change a parameter's data type or name.

Parameters used in SELECT statements cannot be NULL, but they can be NULL for UPDATE and INSERT statements.

See Also

ParamByName (see page 10077)

SQL (see page 10089)

53.1.14.5.9 TQuery.Prepared Property

Determines whether or not a query is prepared for execution.

Pascal

```
property Prepared: Boolean;
```

C++

```
__property Boolean Prepared;
```

Description

Examine Prepared to determine if a query is already prepared for execution. If Prepared is true, the query is prepared, and if Prepared is false, the query is not prepared. While a query need not be prepared before execution, execution performance is enhanced if the query is prepared beforehand, particularly if it is a parameterized query that is executed more than once using the same parameter values.

Note: An application can change the current setting of Prepared to prepare or unprepare a query. If Prepared is true, setting it to

false calls the UnPrepare (see page 10081) method to unprepare the query. If Prepared is false, setting it to true calls the Prepare (see page 10077) method to prepare the query. Generally, however, it is better programming practice to call Prepare (see page 10077) and UnPrepare (see page 10081) directly. These methods automatically update the Prepared property.

See Also

ExecSQL (see page 10075)

Open (see page 6976)

Prepare (see page 10077)

UnPrepare (see page 10081)

53.1.14.5.10 TQuery.RequestLive Property

Requests an updatable query result set from the database.

Pascal

```
property RequestLive: Boolean;
```

C++

```
__property Boolean RequestLive;
```

Description

RequestLive is a request that a SELECT query returns an updatable (or live) or read-only result set from the database back-end. A false value for RequestLive (the default) causes the result set to always be read-only. No request is made to the database back-end to return an updatable result set. A true value for RequestLive is a request to database back-end to return an updatable result set. An updatable result set can be made available to the application's user for direct data editing through visual data controls.

A true value for RequestLive is a request for an updatable result set. It does not guarantee that the database back-end will return an updatable result set. See the documentation for the specific database system used for the criteria needed for a live query result set. If the database cannot return an updatable result set, a read-only result set is returned instead. This is done automatically and usually without error. Some database systems may raise an exception on requests for live result sets that cannot be fulfilled. Check (see page 10245) the success of the request by inspecting the dataset component's CanModify property.

If the SQL (see page 10089) statement used in a TQuery (see page 10059) is a general, multi-row SELECT statement, RequestLive can be used with either a true or a false value. For all other SQL (see page 10089) statements issued from the TQuery (see page 10059), RequestLive should only be set to false. These other statements include noncursor single-row SELECT statements (more commonly found in stored procedure programming), noncursor Data Manipulation Language (DML) statements like UPDATE or DELETE, and Data Definition Language (DDL) statements like CREATE TABLE and DROP INDEX.

Note: All multi-table queries return read-only result sets. Other conditions may cause a query to return a read-only result set. For local tables (such as dBASE and Paradox), see the local SQL (see page 10089) help file for rules governing live result sets (updatable queries). For other table types (such as InterBase, Oracle, and Sybase), consult the documentation for the specific database system.

Note: Some SQL (see page 10089) database systems require strict case-sensitivity for names of metadata objects in SQL (see page 10089) statements. These SQL (see page 10089) databases typically have problems with the way metadata names are passed by the BDE in live queries and an exception is raised. A workaround that helps in most cases is to enclose the metadata object references (like table and column names) in quotation marks.

See Also

CanModify

ExecSQL (see page 10075)

Open (see page 6976)

SQL (see page 10089)

Obtaining an Editable Result Set

Using TQuery

53.1.14.5.11 TQuery.RowsAffected Property

Returns the number of rows operated upon by the latest query execution.

Pascal

```
property RowsAffected: Integer;
```

C++

```
__property int RowsAffected;
```

Description

Inspect RowsAffected to determine how many rows were updated or deleted by the last query operation. If no rows were updated or deleted, RowsAffected has a value of zero. RowsAffected will have a value of –1 if the execution of the SQL (see page 10089) statement could not be executed due to an error condition. This latter situation would typically follow the raising of an exception.

See Also

ExecSQL (see page 10075)

53.1.14.5.12 TQuery.SQL Property

Contains the text of the SQL statement to execute for the query.

Pascal

```
property SQL: TStrings;
```

C++

```
__property TStrings SQL;
```

Description

Use SQL to provide the SQL statement that a query component executes when its ExecSQL (see page 10075) or Open method is called. At design time the SQL property can be edited by invoking the String List editor in the Object Inspector.

The SQL property may contain only one complete SQL statement at a time. In general, multiple "batch" statements are not allowed unless a particular server supports them.

The SQL property can be used to access

Paradox or dBASE tables, using local SQL. The allowable syntax is a subset of SQL-92. See the local SQL help file for information on supported syntax.

Databases on the Local (see page 10085) InterBase Server, using standard SQL-92 and specific InterBase extensions to SQL syntax. For information on syntax and limitations, see the InterBase Language Reference and online help.

Databases on remote database servers using the appropriate SQL Links driver or ODBC (not available in all versions). Any standard statement in the server's SQL dialect is allowed. For information on SQL syntax and limitations, see the server documentation.

The SQL statement in the SQL property may contain replaceable parameters, following standard SQL-92 syntax conventions. Parameters are created and stored in the Params (see page 10087) property.

Note: Developers with advanced versions can also invoke the Visual Query Builder to create an SQL statement for execution. Right-click the TQuery (see page 10059) icon on the form in the IDE and select the command from the context menu.

See Also

Add (see page 3033)

Clear (see page 3047)

Close (see page 6920)

ExecSQL (see page 10075)

LoadFromFile (see page 3061)

Open (see page 6976)

ParamByName (see page 10077)

Prepare (see page 10077)

Using Datasets with Decision Support Components

Using TQuery

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TQuery

Using TSQLQuery

53.1.14.5.13 TQuery.SQLBinary Property

Points to the binary data stream that represents an SQL (see page 10089) query statement or result set.

Pascal

```
property SQLBinary: PChar;
```

C++

```
__property const char * SQLBinary;
```

Description

Do not access SQLBinary. It is an internal binary data stream used by the query component to communicate directly with the Borland (see page 1761) Database (see page 10021) Engine. To access or set the SQL (see page 10089) statement that this query component executes, use the SQL (see page 10089) property. To access or set the parameters used in a parameterized SQL (see page 10089) statement, use the Params (see page 10087) property.

See Also

Params (see page 10087)

SQL (see page 10089)

53.1.14.5.14 TQuery.StmtHandle Property

Specifies the Borland Database Engine (BDE) statement handle for the dataset.

Pascal

```
property StmtHandle: HDBIStmt;
```

C++

```
__property HDBIStmt StmtHandle;
```

Description

Use StmtHandle for direct calls to the BDE that bypass the methods of this dataset. Some BDE API calls require a statement handle as a parameter. For example, TBDEDataSet descendants use StmtHandle when preparing a query or binding parameters.

Typically, applications do not need to access this property. In TBDEDataSet, StmtHandle is protected, and so unavailable to applications. Some descendants make this property public so that you can bypass the dataset methods and make direct calls to the BDE.

See Also

Handle (see page 9933)

Prepare (see page 10077)

Prepare (see page 10157)

Params (see page 10087)

Params (see page 10165)

53.1.14.5.15 TQuery.Text Property

Points to the actual text of the SQL (see page 10089) query passed to the Borland (see page 1761) Database (see page 10021) Engine (BDE).

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

Text is a read-only property that can be examined to determine the actual contents of SQL (see page 10089) statement passed to the BDE. For parameterized queries, Text contains the SQL (see page 10089) statement with parameters replaced by the parameter substitution symbol (?) in place of actual parameter values.

In general there should be no need to examine the Text property. To access or change the SQL (see page 10089) statement for the query, use the SQL (see page 10089) property. To examine or modify parameters, use the Params (see page 10087) property.

See Also

Params (see page 10087)

SQL (see page 10089)

53.1.14.5.16 TQuery.UniDirectional Property

Determines whether or not Borland (see page 1761) Database (see page 10021) Engine (BDE) bidirectional cursors are enabled for a query's result set.

Pascal

```
property UniDirectional: Boolean;
```

C++

```
__property Boolean UniDirectional;
```

Description

Set UniDirectional to control whether or not a BDE cursor can move forward and backward through a result set. By default UniDirectional is false, enabling forward and backward navigation.

Traditionally SQL (see page 10089) cursors are unidirectional. They can travel only forward through a dataset. The BDE, however, permits bidirectional travel by caching records. If an application does not need bidirectional access to records in a result set, set UniDirectional to true. When UniDirectional is true, an application requires less memory and performance is improved.

Note: Do not confuse the UniDirectional property, which indicates whether the BDE cursor is unidirectional, with the IsUniDirectional property inherited from TDataSet. IsUniDirectional, which implies additional restrictions such as a lack of buffering, is always false for TQuery (see page 10059).

See Also

IsUniDirectional (see page 7029)

53.1.14.5.17 TQuery.UpdateMode Property

Determines how the Borland Database Engine (BDE) finds records when updating to an SQL database.

Pascal

```
property UpdateMode: TUpdateMode;
```

C++

```
__property TUpdateMode UpdateMode;
```

Description

Use UpdateMode to specify the criteria to use when locating a record in the dataset. UpdateMode specifies whether modified records are located based on all columns (fields), on only the key fields, or on the key fields plus the original values of fields that have been modified.

See Also

UpdateRecordTypes (see page 9936)

53.1.14.5.18 TQuery.UpdateObject Property

Specifies the update object component used to update a read-only result set when cached updates are enabled.

Pascal

```
property UpdateObject: TDataSetUpdateObject;
```

C++

```
__property TDataSetUpdateObject UpdateObject;
```

Description

Use UpdateObject to specify the TUpdateObject component to use in an application that must be able to update a read-only result set returned by the Borland Database Engine (BDE).

The BDE always attempts to provide an updatable, or "live" query result unless an application specifically requests a read-only view of data. In some cases, such as a query made against multiple tables, a live result set cannot be returned. In these cases, UpdateObject can be used to specify a TUpdateSQL component that performs updates as a separate transaction that is transparent to the application.

See Also

CachedUpdates (see page 9932)

ApplyUpdates (see page 9894)

TUpdateSQL (see page 10229)

Using the BDE to Cache Updates

Using Update Objects to Update a Dataset

Updating a Read-only Result Set

Using TQuery

Using TUpdateSQL to Update a Dataset

53.1.15 TSession Class

TSession provides global management of a group of database connections in an application.

Class Hierarchy

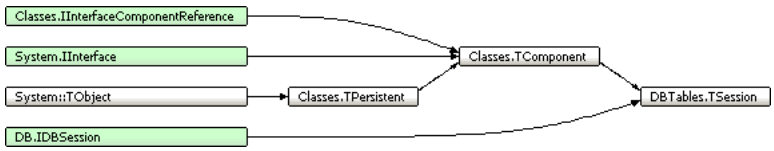

Pascal

```
TSession = class(TComponent, IDBSession);
```

C++

```
class TSession : public TComponent, public IDBSession;
```

File

DBTables

Description

Use TSession to manage a group of database connections within an application. There are three uses for TSession: standard, multiple net files for Paradox, and multi-threaded database applications.

The library automatically creates a globally accessible default TSession component named Session (see page 10265) for all database applications. The default session component handles standard database connections. An application can control the

default session by accessing its properties, events, and methods at runtime.

Database applications that must simultaneously access Paradox tables located in different network locations can establish multiple sessions, one for each network location.

Finally, database applications that must establish multiple, concurrent connections to the same database, such as performing two queries against the same data at once, are multi-threaded applications. Multi-threaded applications must create and maintain one additional session component for each simultaneous connection to a single database server.

Note: Applications that maintain multiple sessions can manage them through the TSessionList (see page 10129) component. A default session list component, called Sessions (see page 10266)—note the plural which distinguishes it from the default session component, Session (see page 10265)—is automatically created for all database applications.

Note: TSession implements the IDBSession interface, which allows the global PasswordDialog function to supply passwords to the component.

See Also

TDatabase (see page 9964)

TSessionList (see page 10129)

Session (see page 10265)

Sessions (see page 10266)

IDBSession (see page 7573)

PasswordDialog

Associating a Database Component with a Session

Understanding Database and Session Component Interactions

53.1.15.1 TSession Members

The following tables list the members exposed by TSession.

Protected Events

	Name	Description
	OnDBNotify (see page 10128)	This is OnDBNotify, a member of class TSession.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.

	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 10114)	Initializes the component after the form file has been read into memory.
	ModifyConfigParams (see page 10115)	This is ModifyConfigParams, a member of class TSession.
V	Notification (see page 10116)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 10119)	Sets the value of the Name property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	BDEOwnsLoginCbDb (see page 10122)	This is BDEOwnsLoginCbDb, a member of class TSession.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddAlias (see page 10100)	Adds a specified Borland (see page 1761) Database Engine (BDE) alias to the session for an SQL database server.
	AddDriver (see page 10101)	Adds a specified Borland (see page 1761) Database Engine (BDE) driver to the session for an SQL database server.
	AddPassword (see page 10102)	Adds a password to the current session for accessing encrypted Paradox or dBase tables.
	AddStandardAlias (see page 10102)	Adds a standard Borland (see page 1761) Database Engine (BDE) alias to the session for Paradox, dBASE, or ASCII tables.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 10103)	Disconnects from all databases, and closes the session.
	CloseDatabase (see page 10103)	Closes a database connection associated with the current session.
V	Create (see page 10098)	Creates an instance of a TSession (see page 10093) component.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DeleteAlias (see page 10104)	Removes a specified Borland (see page 1761) Database Engine (BDE) alias from the session.
	DeleteDriver (see page 10105)	Removes a specified Borland (see page 1761) Database Engine (BDE) driver from the session.
V	Destroy (see page 10099)	Destroys the instance of a session component.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DropConnections (see page 10105)	Frees all temporary database components associated with a session if they are inactive.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindDatabase (see page 10105)	Searches a session's list of database components for a specified database.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAliasDriverName (see page 10106)	Provides the name of the database driver used by a specified Borland (see page 1761) Database Engine (BDE) alias associated with the session.
	GetAliasNames (see page 10107)	Populates a string list with the names of persistent Borland (see page 1761) Database Engine (BDE) aliases.
	GetAliasParams (see page 10108)	Retrieves the parameters associated with a specified Borland (see page 1761) Database Engine (BDE) alias.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetConfigParams (see page 10108)	Retrieves Borland (see page 1761) Database Engine (BDE) configuration information.
	GetDatabaseNames (see page 10109)	Populates a string list with the names of persistent Borland (see page 1761) Database Engine (BDE) aliases and the names of all databases components known to the session.
	GetDriverNames (see page 10110)	Populates a string list with the names of all Borland (see page 1761) Database Engine (BDE) drivers available to the session.
	GetDriverParams (see page 10110)	Populates a string list with the parameters for a specified Borland (see page 1761) Database Engine (BDE) driver.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldNames (see page 10110)	Populates a string list with the names of all fields in a specified table.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.

	GetPassword (see page 10111)	Invokes the OnPassword (see page 10128) event handler for the session or displays the default password dialog box.
	GetStoredProcNames (see page 10112)	Populates a string list with the names of all stored procedures associated with a specified database component that is connected to an SQL database server.
	GetTableNames (see page 10113)	Populates a string list with the names of tables associated with a specified database component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsAlias (see page 10113)	Determines whether or not a string value corresponds to an existing Borland (see page 1761) Database Engine (BDE) database alias known to the session.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	ModifyAlias (see page 10114)	Adds or changes parameters for a Borland (see page 1761) Database Engine (BDE) alias.
	ModifyDriver (see page 10115)	Adds or changes parameters for a Borland (see page 1761) Database Engine (BDE) driver.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 10116)	Starts a session and makes it the current session.
	OpenDatabase (see page 10117)	Opens an existing database, or creates a temporary database component and opens it.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveAllPasswords (see page 10118)	Deletes all passwords previously added to the current session for accessing encrypted Paradox tables.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemovePassword (see page 10118)	Deletes a single password previously added to the current session for accessing an encrypted Paradox table.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveConfigFile (see page 10119)	Writes the current information in the Borland (see page 1761) Database Engine (BDE) persistent store from memory to the BDE configuration file on disk.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	ConfigMode (see page 10122)	Specifies how the Borland (see page 1761) Database Engine (BDE) should handle aliases for the session.
R	DatabaseCount (see page 10123)	Indicates the number of active database components currently associated with the session.
R	Databases (see page 10123)	Provides an indexed array of all active database component names for a session.
	DesignInfo (see page 2805)	Contains information used by the Form designer.

R	Handle (see page 10123)	Specifies the Borland (see page 1761) Database Engine (BDE) handle for the session.
R	Locale (see page 10124)	Identifies the Borland (see page 1761) Database Engine (BDE) language driver for the session component.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	TraceFlags (see page 10127)	Specifies the database operations to track with the SQL Monitor at runtime.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnPassword (see page 10128)	Occurs when an application attempts to open a Paradox table for the first time and the Borland (see page 1761) Database Engine (BDE) reports insufficient access rights.
	OnStartup (see page 10129)	Occurs when an application activates a session.

Published Properties

	Name	Description
	Active (see page 10120)	Specifies whether or not a session is active.
	AutoSessionName (see page 10121)	Indicates whether or not a unique session name is automatically generated for a session.
	KeepConnections (see page 10124)	Specifies whether or not a temporary database component created in the context of a session maintains a database server connection even if there are no active datasets associated with the database component.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NetFileDir (see page 10125)	Specifies the directory that contains the Borland (see page 1761) Database Engine (BDE) network control file, PDOXUSRS.NET.
	PrivateDir (see page 10125)	Specifies the directory in which to store temporary table processing files generated by the Borland (see page 1761) Database Engine (BDE) for database components associated with a session.
	SessionName (see page 10126)	Specifies a unique session name that can be used by database and dataset components to link to this session.
	SQLHourGlass (see page 10126)	Indicates whether or not the mouse cursor changes to an hourglass during BDE operations.
	Tag (see page 2808)	Stores an integer value as part of a component.

53.1.15.2 TSession.Create Constructor

Creates an instance of a TSession (see page 10093) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSession(TComponent * AOwner);
```

Description

Do not call Create directly. The default session component, Session (see page 10265), is created automatically for all database applications. To create additional sessions for a multi-threaded database application at runtime call Sessions.OpenSession instead of Create. OpenSession calls Create only if the session does not already exist. (At design time, create additional sessions by dropping them on a data module.)

Create calls its inherited constructor and ensures that the session component cannot be inherited should the data module or form on which it is placed be stored in the Object Repository.

See Also

KeepConnections (see page 10124)

Handle (see page 10123)

Destroy (see page 10099)

OpenSession (see page 10132)

53.1.15.3 TSession.Destroy Destructor

Destroys the instance of a session component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSession();
```

Description

Do not call Destroy directly. For session components other than the default session, an application should call Free, which verifies that the session component is not nil before calling Destroy.

Warning: Never destroy the default session. Freeing of the default session is handled automatically when an application terminates.

Destroy performs the following actions before calling its inherited destructor:

Makes the session inactive, which closes all database connections.

Removes the session from the session list.

Resets database session references, if necessary.

Frees the list of database components associated with the session.

See Also

Active (see page 10120)

Create (see page 10098)

OpenSession (see page 10132)

Free (see page 22300)

TSessionList (see page 10129)

53.1.15.4 TSession Methods

The methods of the TSession class are listed here.

Protected Methods

	Name	Description
V	Loaded (see page 10114)	Initializes the component after the form file has been read into memory.
	ModifyConfigParams (see page 10115)	This is ModifyConfigParams, a member of class TSession.
V	Notification (see page 10116)	Forwards notification messages to all owned components.
V	SetName (see page 10119)	Sets the value of the Name property.

Public Methods

	Name	Description
	AddAlias (see page 10100)	Adds a specified Borland (see page 1761) Database Engine (BDE) alias to the session for an SQL database server.
	AddDriver (see page 10101)	Adds a specified Borland (see page 1761) Database Engine (BDE) driver to the session for an SQL database server.
	AddPassword (see page 10102)	Adds a password to the current session for accessing encrypted Paradox or dBase tables.
	AddStandardAlias (see page 10102)	Adds a standard Borland (see page 1761) Database Engine (BDE) alias to the session for Paradox, dBASE, or ASCII tables.
	Close (see page 10103)	Disconnects from all databases, and closes the session.
	CloseDatabase (see page 10103)	Closes a database connection associated with the current session.
	DeleteAlias (see page 10104)	Removes a specified Borland (see page 1761) Database Engine (BDE) alias from the session.
	DeleteDriver (see page 10105)	Removes a specified Borland (see page 1761) Database Engine (BDE) driver from the session.
	DropConnections (see page 10105)	Frees all temporary database components associated with a session if they are inactive.
	FindDatabase (see page 10105)	Searches a session's list of database components for a specified database.
	GetAliasDriverName (see page 10106)	Provides the name of the database driver used by a specified Borland (see page 1761) Database Engine (BDE) alias associated with the session.
	GetAliasNames (see page 10107)	Populates a string list with the names of persistent Borland (see page 1761) Database Engine (BDE) aliases.
	GetAliasParams (see page 10108)	Retrieves the parameters associated with a specified Borland (see page 1761) Database Engine (BDE) alias.
	GetConfigParams (see page 10108)	Retrieves Borland (see page 1761) Database Engine (BDE) configuration information.
	GetDatabaseNames (see page 10109)	Populates a string list with the names of persistent Borland (see page 1761) Database Engine (BDE) aliases and the names of all databases components known to the session.
	GetDriverNames (see page 10110)	Populates a string list with the names of all Borland (see page 1761) Database Engine (BDE) drivers available to the session.
	GetDriverParams (see page 10110)	Populates a string list with the parameters for a specified Borland (see page 1761) Database Engine (BDE) driver.
	GetFieldNames (see page 10110)	Populates a string list with the names of all fields in a specified table.
	GetPassword (see page 10111)	Invokes the OnPassword (see page 10128) event handler for the session or displays the default password dialog box.
	GetStoredProcNames (see page 10112)	Populates a string list with the names of all stored procedures associated with a specified database component that is connected to an SQL database server.
	GetTableNames (see page 10113)	Populates a string list with the names of tables associated with a specified database component.
	IsAlias (see page 10113)	Determines whether or not a string value corresponds to an existing Borland (see page 1761) Database Engine (BDE) database alias known to the session.
	ModifyAlias (see page 10114)	Adds or changes parameters for a Borland (see page 1761) Database Engine (BDE) alias.
	ModifyDriver (see page 10115)	Adds or changes parameters for a Borland (see page 1761) Database Engine (BDE) driver.
	Open (see page 10116)	Starts a session and makes it the current session.
	OpenDatabase (see page 10117)	Opens an existing database, or creates a temporary database component and opens it.
	RemoveAllPasswords (see page 10118)	Deletes all passwords previously added to the current session for accessing encrypted Paradox tables.
	RemovePassword (see page 10118)	Deletes a single password previously added to the current session for accessing an encrypted Paradox table.
	SaveConfigFile (see page 10119)	Writes the current information in the Borland (see page 1761) Database Engine (BDE) persistent store from memory to the BDE configuration file on disk.

53.1.15.4.1 TSession.AddAlias Method

Adds a specified Borland (see page 1761) Database Engine (BDE) alias to the session for an SQL database server.

Pascal

```
procedure AddAlias(const Name: string; const Driver: string; List: TStrings);
```

C++

```
__fastcall AddAlias(const AnsiString Name, const AnsiString Driver, TStrings List);
```

Description

Call AddAlias at runtime to create a session-specific BDE alias for an SQL database server. Name is the unique name to assign to the alias. Driver is the name of the SQL Links database driver to use for the alias. List is a string list containing parameters for the alias.

A session's ConfigMode (see page 10122) property determines whether an alias is local to the session, or is available to all applications in the BDE persistent store as long as any BDE clients are active. To save an alias permanently, call the SaveConfigFile (see page 10119) method before ending the session.

Note: Although AddAlias can be used to create a Paradox or dBASE alias, AddStandardAlias (see page 10102) provides a simpler method for Paradox and dBASE alias creation.

See Also

ConfigMode (see page 10122)

AddStandardAlias (see page 10102)

DeleteAlias (see page 10104)

IsAlias (see page 10113)

ModifyAlias (see page 10114)

SaveConfigFile (see page 10119)

Working with BDE Aliases

53.1.15.4.2 TSession.AddDriver Method

Adds a specified Borland (see page 1761) Database Engine (BDE) driver to the session for an SQL database server.

Pascal

```
procedure AddDriver(const Name: string; List: TStrings);
```

C++

```
__fastcall AddDriver(const AnsiString Name, TStrings List);
```

Description

Call AddDriver at runtime to create a BDE driver for an SQL database server. AddDriver can also be used to create ODBC driver entries. Name is the unique name to assign to the driver. List is a string list containing parameters for the alias.

A session's ConfigMode (see page 10122) property determines whether a driver is local to the session, or is available to all applications in the BDE persistent store as long as any BDE clients are active. To save a driver permanently, call the SaveConfigFile (see page 10119) method before ending the session.

See Also

ConfigMode (see page 10122)

DeleteDriver (see page 10105)

IsAlias (see page 10113)

SaveConfigFile (see page 10119)

53.1.15.4.3 TSession.AddPassword Method

Adds a password to the current session for accessing encrypted Paradox or dBase tables.

Pascal

```
procedure AddPassword(const Password: string);
```

C++

```
__fastcall AddPassword(const AnsiString Password);
```

Description

Call AddPassword to provide a password for a session prior to opening an encrypted Paradox or dBase table that requires a password. If an application opens a table that requires a password for access, the session does not have a password assigned to it, and the application does not provide its own OnPassword (see page 10128) event handler, the library displays a dialog box prompting the user for a valid password before allowing table access.

For example, to provide the String literal "secret" as a password for the session:

```
Session.AddPassword('secret');
Session->AddPassword("secret");
```

Note: If an application defines its own OnPassword (see page 10128) event handler, the handler should call this method to add passwords for the session. All passwords assigned by AddPassword should be deleted with RemoveAllPasswords (see page 10118) or RemovePassword (see page 10118) before an application closes.

See Also

OnPassword (see page 10128)

RemoveAllPasswords (see page 10118)

RemovePassword (see page 10118)

GetPassword (see page 10111)

PasswordDialog (see page 9849)

Working with Password-protected Paradox and dBASE Tables

53.1.15.4.4 TSession.AddStandardAlias Method

Adds a standard Borland (see page 1761) Database Engine (BDE) alias to the session for Paradox, dBASE, or ASCII tables.

Pascal

```
procedure AddStandardAlias(const Name: string; const Path: string; const DefaultDriver:
string);
```

C++

```
__fastcall AddStandardAlias(const AnsiString Name, const AnsiString Path, const AnsiString
DefaultDriver);
```

Description

Call AddStandardAlias at runtime to create a session-specific BDE alias for Paradox, dBASE or ASCII tables. Name is the unique name to assign to the alias. Path specifies the directory where the database is located. DefaultDriver is the name of the standard BDE driver to use for the alias. If set, DefaultDriver must be one of "Paradox," "DBASE," or "ASCIIDRV". If left blank DefaultDriver defaults to "Paradox". DefaultDriver is used when attempting to open a table that both does not have an extension and is defined as a table of type ttDefault.

A session's ConfigMode (see page 10122) property determines whether an alias is local to the session, or is available to all applications in the BDE persistent store as long as any BDE clients are active. the lifetime of the session. To save an alias permanently, call the SaveConfigFile (see page 10119) method before ending the session.

Note: To create an alias for an SQL database, call AddAlias (see page 10100) instead of AddStandardAlias.

See Also

ConfigMode (see page 10122)

AddAlias (see page 10100)

DeleteAlias (see page 10104)

IsAlias (see page 10113)

ModifyAlias (see page 10114)

SaveConfigFile (see page 10119)

Working with BDE Aliases

53.1.15.4.5 TSession.Close Method

Disconnects from all databases, and closes the session.

Pascal

```
procedure Close;
```

C++

```
__fastcall Close();
```

Description

Call Close to end a session. Close disconnects from all active databases by freeing temporary database components, and calling the Close methods persistent database components, and then sets its own session handle to nil (Delphi) or NULL (C++).

Note: Setting the Active (see page 10120) property to false also closes a session.

See Also

Active (see page 10120)

Open (see page 10116)

Close (see page 6863)

Dropping Inactive Database Connections

53.1.15.4.6 TSession.CloseDatabase Method

Closes a database connection associated with the current session.

Pascal

```
procedure CloseDatabase(Database: TDatabase);
```

C++

```
__fastcall CloseDatabase(TDatabase Database);
```

Description

Call CloseDatabase to close a persistent database connection, or to close and free a temporary database connection. Database

specifies the name of the database component for which to close the connection.

CloseDatabase decrements the specified database component's reference count, and then, if the reference count is zero and the database component's KeepConnection property is false, CloseDatabase either frees a temporary database component or closes the connection for a persistent database component.

Note: Calling CloseDatabase for a persistent database component does not close the connection. To close a connection for a persistent database component, call the database component's Close (see page 10103) method directly.

Temporary database components are closed automatically when the last dataset associated with the database component is closed, but an application can call CloseDatabase prior to that time to force closure. Closing a connection established by a temporary database component does not free the component if the session's KeepConnections (see page 10124) property is true (the default). To free temporary database components after closing their connections call DropConnections (see page 10105).

See Also

KeepConnections (see page 10124)

DropConnections (see page 10105)

OpenDatabase (see page 10117)

KeepConnection (see page 9984)

Close (see page 6863)

Closing Database Connections

53.1.15.4.7 TSession.DeleteAlias Method

Removes a specified Borland (see page 1761) Database Engine (BDE) alias from the session.

Pascal

```
procedure DeleteAlias(const Name: string);
```

C++

```
__fastcall DeleteAlias(const AnsiString Name);
```

Description

Call DeleteAlias to remove a BDE alias from a session. Removing an alias makes the alias unavailable to the session, but does not remove it from the BDE configuration file if the alias was stored there when it was created. To remove an alias from the BDE configuration file follow the call to DeleteAlias with a call to SaveConfigFile (see page 10119).

See Also

ConfigMode (see page 10122)

AddAlias (see page 10100)

AddStandardAlias (see page 10102)

IsAlias (see page 10113)

SaveConfigFile (see page 10119)

Working with BDE Aliases

53.1.15.4.8 TSession.DeleteDriver Method

Removes a specified Borland (see page 1761) Database Engine (BDE) driver from the session.

Pascal

```
procedure DeleteDriver(const Name: string);
```

C++

```
__fastcall DeleteDriver(const AnsiString Name);
```

Description

Call DeleteDriver to remove a BDE driver from a session. Removing a driver makes the driver unavailable to the session, but does not remove it from the BDE configuration file if the driver was stored there when it was created. To remove a driver from the BDE configuration file follow the call to DeleteDriver with a call to SaveConfigFile (see page 10119).

See Also

ConfigMode (see page 10122)

AddDriver (see page 10101)

SaveConfigFile (see page 10119)

53.1.15.4.9 TSession.DropConnections Method

Frees all temporary database components associated with a session if they are inactive.

Pascal

```
procedure DropConnections;
```

C++

```
__fastcall DropConnections();
```

Description

Use DropConnections to free all temporary database components that are currently inactive. If the KeepConnections (see page 10124) property of the session is true (the default), then temporary database components created as needed for the session by the library at runtime are not automatically freed when their database connections are closed. DropConnections enables an application to free these components when they are no longer needed.

See Also

KeepConnections (see page 10124)

CloseDatabase (see page 10103)

Understanding Database and Session Component Interactions

Specifying Default Database Connection Behavior

Dropping Inactive Database Connections

53.1.15.4.10 TSession.FindDatabase Method

Searches a session's list of database components for a specified database.

Pascal

```
function FindDatabase(const DatabaseName: string): TDatabase;
```

C++

```
__fastcall TDatabase FindDatabase(const AnsiString DatabaseName);
```

Description

Call FindDatabase to determine whether or not a specified database is associated with a session. DatabaseName specifies the name of the database component for which to search.

FindDatabase compares DatabaseName to the DatabaseName property for each database component in a session's list. If a match is found, FindDatabase returns a pointer to the database component. Otherwise FindDatabase returns nil (Delphi) or NULL (C++).

An application can call FindDatabase to see if a database component already exists before creating a new database component for a session.

```
var
  DB: TDatabase;
begin
  DB := Session.FindDatabase('DBDEMOS');
if (DB = nil) then { database doesn't exist for session so,}
    DB := Session.OpenDatabase('DBDEMOS'); { create and open it}
if Assigned(DB) and DB.Active then begin
    DB.StartTransaction;
...
end;
end;
```

See Also

Databases (see page 10123)

OpenDatabase (see page 10117)

DatabaseName (see page 9980)

Searching for a Database Connection

53.1.15.4.11 TSession.GetAliasDriverName Method

Provides the name of the database driver used by a specified Borland (see page 1761) Database Engine (BDE) alias associated with the session.

Pascal

```
function GetAliasDriverName(const AliasName: string): string;
```

C++

```
__fastcall AnsiString GetAliasDriverName(const AnsiString AliasName);
```

Description

Call GetAliasDriverName to determine the BDE database driver used by a specific alias. AliasName is the name of the alias for which to return driver information.

For aliases corresponding to Paradox and dBASE tables, GetAliasDriverName returns "STANDARD". For ASCII tables the return value is "ASCIIDRV". For SQL databases, the driver name depends on the database server.

See Also

GetAliasNames (see page 10107)

GetAliasParams (see page 10108)

GetDriverNames (see page 10110)

GetDriverParams (see page 10110)

Retrieving Information About a Session

Working with BDE Aliases

53.1.15.4.12 TSession.GetAliasNames Method

Populates a string list with the names of persistent Borland (see page 1761) Database Engine (BDE) aliases.

Pascal

```
procedure GetAliasNames(List: TStrings);
```

C++

```
__fastcall GetAliasNames(TStrings List);
```

Description

Call GetAliasNames to retrieve a list of persistent BDE aliases—those aliases that exist in persistent storage or in the BDE configuration file. List is a string list object, created and maintained by the application, into which to store the alias names.

A session's ConfigMode (see page 10122) property determines whether an alias created in the context of this session is reported by GetAliasNames. If ConfigMode (see page 10122) is cmAll (see page 10268), aliases created within the session are reported.

C++ Examples:

```
/*
This example calls the TSession GetAliasNames method to fill
the list box with Borland Database Engine (BDE) alias names.
*/
void __fastcall TAdhocForm::Button1Click(TObject *Sender)
{
  TStringList *MyStringList = new TStringList();
  try
  {
    Session->GetAliasNames(MyStringList);
    // fill a list box with alias names for the user to select from
    for (int I = 0; I < MyStringList->Count; I++)
      ListBox1->Items->Add(MyStringList->Strings[I]);
  }
  __finally
  {
    delete MyStringList;
  }
}
```

Delphi Examples:

```
{
This example calls the TSession GetAliasNames method to fill
the list box with Borland Database Engine (BDE) alias names.
}
procedure TAdhocForm.Button1Click(Sender: TObject);
var
  MyStringList: TStringList;
  I: Integer;
begin
  MyStringList := TStringList.Create;
  try
    Session.GetAliasNames(MyStringList);
    { fill a list box with alias names for the user to select from }
    for I := 0 to MyStringList.Count - 1 do
      ListBox1.Items.Add(MyStringList[I]);
```

```
    finally
      MyStringList.Free;
    end;
  end;
```

See Also

ConfigMode (see page 10122)

AddAlias (see page 10100)

AddStandardAlias (see page 10102)

GetAliasDriverName (see page 10106)

GetAliasParams (see page 10108)

SaveConfigFile (see page 10119)

Retrieving Information About a Session

Working with BDE Aliases

53.1.15.4.13 TSession.GetAliasParams Method

Retrieves the parameters associated with a specified Borland (see page 1761) Database Engine (BDE) alias.

Pascal

```
procedure GetAliasParams(const AliasName: string; List: TStrings);
```

C++

```
__fastcall GetAliasParams(const AnsiString AliasName, TStrings List);
```

Description

Call GetAliasParams to retrieve the BDE configuration parameters for a specified alias. AliasName is the name of the alias for which to retrieve parameters. List is a string list object, created and maintained by the application, into which to store the parameters.

See Also

GetAliasDriverName (see page 10106)

GetConfigParams (see page 10108)

Retrieving Information About a Session

Working with BDE Aliases

53.1.15.4.14 TSession.GetConfigParams Method

Retrieves Borland (see page 1761) Database Engine (BDE) configuration information.

Pascal

```
procedure GetConfigParams(const Path: string; const Section: string; List: TStrings);
```

C++

```
__fastcall GetConfigParams(const AnsiString Path, const AnsiString Section, TStrings List);
```

Description

Call GetConfigParams to retrieve specific types of configuration information from the BDE configuration file used to initialize the application.

Path represents the internal node structure of the BDE configuration file, and must always start at the root, denoted by a backslash. Each node must be separated from others by a backslash, and no node may be empty. For example, a Path of \SYSTEM\FORMATS\TIME, returns the time format used by the BDE configuration file.

Path can contain a replaceable string formatting symbol (%s) in one of its node designations. If it does, Section must contain the missing node information. In this case, GetConfigParams inserts the contents of Section into Path before calling the BDE for the requested information.

List is a string list object, created and maintained by the application, into which to store the configuration information. For example, if Path is \DATABASES\%s\DB (see page 6725) INFO, and Section is an alias name, then GetConfigParams returns the parameters for the alias in List.

For more information about the BDE Configuration File, see the online BDE help.

See Also

GetAliasDriverName (see page 10106)

GetAliasNames (see page 10107)

GetAliasParams (see page 10108)

GetDatabaseNames (see page 10109)

GetDriverNames (see page 10110)

GetDriverParams (see page 10110)

GetStoredProcNames (see page 10112)

GetTableNames (see page 10113)

Retrieving Information About a Session

53.1.15.4.15 TSession.GetDatabaseNames Method

Populates a string list with the names of persistent Borland (see page 1761) Database Engine (BDE) aliases and the names of all databases components known to the session.

Pascal

```
procedure GetDatabaseNames(List: TStrings);
```

C++

```
__fastcall GetDatabaseNames(TStrings List);
```

Description

Call GetDatabaseNames to retrieve a sorted, combined list of persistent BDE aliases—those aliases that exist in persistent storage or in the BDE configuration file—and the names of all database components known to the session. List is a string list object, created and maintained by the application, into which to store the database and alias names.

See Also

GetAliasNames (see page 10107)

GetDriverNames (see page 10110)

SaveConfigFile (see page 10119)

Retrieving Information About a Session

53.1.15.4.16 TSession.GetDriverNames Method

Populates a string list with the names of all Borland (see page 1761) Database Engine (BDE) drivers available to the session.

Pascal

```
procedure GetDriverNames(List: TStrings);
```

C++

```
__fastcall GetDriverNames(TStrings List);
```

Description

Call GetDriverNames to retrieve the names of all BDE drivers that can be used by a session. List is a string list object, created and maintained by the application, into which to store the driver names.

See Also

GetDriverParams (see page 10110)

Retrieving Information About a Session

Working with BDE Aliases

53.1.15.4.17 TSession.GetDriverParams Method

Populates a string list with the parameters for a specified Borland (see page 1761) Database Engine (BDE) driver.

Pascal

```
procedure GetDriverParams(const DriverName: string; List: TStrings);
```

C++

```
__fastcall GetDriverParams(const AnsiString DriverName, TStrings List);
```

Description

Call GetDriverParams to retrieve a list of parameters associated with a specified BDE driver. DriverName is the BDE driver for which to return a parameter list. List is a string list object, created and maintained by the application, into which to store the driver parameters.

The actual number of parameters returned by GetDriverParams depends on the specified driver. For example, for the STANDARD driver used by Paradox and dBASE, returns only a few parameters. For SQL Links drivers multiple parameters are returned.

See Also

GetDriverNames (see page 10110)

Retrieving Information About a Session

Working with BDE Aliases

53.1.15.4.18 TSession.GetFieldNames Method

Populates a string list with the names of all fields in a specified table.

Pascal

```
procedure GetFieldNames(const DatabaseName: string; const TableName: string; List: TStrings);
```

C++

```
__fastcall GetFieldNames(const AnsiString DatabaseName, const AnsiString TableName, TStrings
List);
```

Description

Call GetFieldNames to retrieve a list of all the columns (fields) in a specified table.

DatabaseName is the name of the database that contains the table. It can be the name of an existing Borland (see page 1761) Database Engine (BDE) alias, the name of a TDatabase (see page 9964) component's DatabaseName property, or a fully qualified path name for the location of a Paradox or dBASE table.

TableName is the name of the table for which you want to list the fields.

List is a string list object, created and maintained by the application, into which GetFieldNames adds the names of all fields in the table specified by TableName.

See Also

DatabaseName (see page 9980)

GetFieldNames (see page 9975)

GetTableNames (see page 10113)

GetStoredProcNames (see page 10112)

GetDatabaseNames (see page 10109)

Retrieving Information About a Session

53.1.15.4.19 TSession.GetPassword Method

Invokes the OnPassword (see page 10128) event handler for the session or displays the default password dialog box.

Pascal

```
function GetPassword: Boolean;
```

C++

```
__fastcall Boolean GetPassword();
```

Description

Call GetPassword to invoke the OnPassword (see page 10128) event handler for the session and handle response logic when an application is working with password-protected Paradox files. If an OnPassword (see page 10128) event handler is not defined, GetPassword displays the default password dialog box.

An application can use the return value of GetPassword to control program logic. GetPassword returns true if a user chooses OK, or false if the user chooses Cancel.

```
var
  rslt: Boolean;
begin
  rslt := Session.GetPassword;
try
if rslt then
      Table1.Open
else
      ShowMessage('No password given');
except
    ShowMessage('Could not open table!');
end;
ifnot Table1.Active then
```

```
    Application.Terminate;
end;
void __fastcall TForm1::OpenTableButtonClick(TObject *Sender)
{
bool rslt = Session->GetPassword();
try
  {
if (rslt)
      Table1->Open();
else
      ShowMessage("No password given");
  }
catch (...)
  {
    ShowMessage("Could not open table!");
  }
if (!Table1->Active)
    Application.Terminate();
}
```

See Also

OnPassword (see page 10128)

AddPassword (see page 10102)

RemovePassword (see page 10118)

RemoveAllPasswords (see page 10118)

PasswordDialog (see page 9849)

Working with Password-protected Paradox and dBASE Tables

53.1.15.4.20 TSession.GetStoredProcNames Method

Populates a string list with the names of all stored procedures associated with a specified database component that is connected to an SQL database server.

Pascal

```
procedure GetStoredProcNames(const DatabaseName: string; List: TStrings);
```

C++

```
__fastcall GetStoredProcNames(const AnsiString DatabaseName, TStrings List);
```

Description

Call GetStoredProcNames to retrieve a list of all stored procedures for a database component that is attached to an SQL database server. DatabaseName specifies the name of the database for which to retrieve stored procedure names. List is a string list object, created and maintained by the application, into which to return the stored procedure names.

Note: GetStoredProcNames raises an exception if called on Paradox or dBASE databases.

See Also

GetAliasNames (see page 10107)

GetAliasDriverName (see page 10106)

GetDriverNames (see page 10110)

GetTableNames (see page 10113)

Retrieving Information About a Session

53.1.15.4.21 TSession.GetTableNames Method

Populates a string list with the names of tables associated with a specified database component.

Pascal

```
procedure GetTableNames(const DatabaseName: string; const Pattern: string; Extensions:
Boolean; SystemTables: Boolean; List: TStrings);
```

C++

```
__fastcall GetTableNames(const AnsiString DatabaseName, const AnsiString Pattern, Boolean
Extensions, Boolean SystemTables, TStrings List);
```

Description

Call GetTableNames to retrieve a list of the tables associated with a given database. DatabaseName specifies the name of the database from which to retrieve table names.

Pattern specifies a delimiter string that restricts the tables returned to those that match the string. Pattern can include wildcard symbols. Pass an empty Pattern string to match all files not restricted by other criteria.

For searches on Paradox and dBASE, set Extensions to true to include filename extensions as part of the table name. For SQL-based databases, set Extension to false.

Set SystemTables to true for SQL-based databases to return both data tables and the system tables that define the database structure. Set SystemTables to false for Paradox and dBASE tables.

List is a string list object, created and maintained by the application, into which to return the table names.

See Also

GetAliasNames (see page 10107)

GetAliasDriverName (see page 10106)

GetDatabaseNames (see page 10109)

GetDriverNames (see page 10110)

GetStoredProcNames (see page 10112)

GetFieldNames (see page 10110)

Retrieving Information About a Session

53.1.15.4.22 TSession.IsAlias Method

Determines whether or not a string value corresponds to an existing Borland (see page 1761) Database Engine (BDE) database alias known to the session.

Pascal

```
function IsAlias(const Name: string): Boolean;
```

C++

```
__fastcall Boolean IsAlias(const AnsiString Name);
```

Description

Call IsAlias to determine if a string corresponds to the name of an existing BDE alias that can be used by the session. Name is the string to test.

If IsAlias returns true, Name is a valid alias.

Note: The aliases known to IsAlias depend on the session's ConfigMode (see page 10122) property. If ConfigMode (see page 10122) is cmAll (see page 10268), then all aliases are available to the session.

See Also

ConfigMode (see page 10122)

AddAlias (see page 10100)

AddStandardAlias (see page 10102)

DeleteAlias (see page 10104)

GetAliasNames (see page 10107)

GetAliasParams (see page 10108)

ModifyAlias (see page 10114)

53.1.15.4.23 TSession.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState (see page 2804)

Create (see page 2766)

TReader (see page 2933)

53.1.15.4.24 TSession.ModifyAlias Method

Adds or changes parameters for a Borland (see page 1761) Database Engine (BDE) alias.

Pascal

```
procedure ModifyAlias(Name: string; List: TStrings);
```

C++

```
__fastcall ModifyAlias(AnsiString Name, TStrings List);
```

Description

Call ModifyAlias to change the parameters for a specified BDE alias. Name is the name of the alias to modify. List is a string list object previously populated with parameter values to add or change. An application must create and maintain List.

Populate List with parameter strings before calling ModifyAlias. List need only contain the parameters to change. If ModifyAlias passes a parameter for which a value already exists, the new parameter replaces the existing one. New parameters that are not yet defined for the alias are added to the existing parameters. If List is empty, ModifyAlias does not change any existing parameters.

ModifyAlias changes the alias parameters for the session, but does not change them in the BDE configuration file. To modify a driver in the BDE configuration file, follow the call to ModifyAlias with a call to SaveConfigFile (see page 10119).

See Also

ConfigMode (see page 10122)

GetAliasParams (see page 10108)

Params (see page 9986)

SaveConfigFile (see page 10119)

Working with BDE Aliases

53.1.15.4.25 TSession.ModifyConfigParams Method

Pascal

```
procedure ModifyConfigParams(const Path: string; const Node: string; List: TStrings);
```

C++

```
__fastcall ModifyConfigParams(const AnsiString Path, const AnsiString Node, TStrings List);
```

Description

This is ModifyConfigParams, a member of class TSession.

53.1.15.4.26 TSession.ModifyDriver Method

Adds or changes parameters for a Borland (see page 1761) Database Engine (BDE) driver.

Pascal

```
procedure ModifyDriver(Name: string; List: TStrings);
```

C++

```
__fastcall ModifyDriver(AnsiString Name, TStrings List);
```

Description

Call ModifyDriver to change the parameters for a specified BDE database driver. Name is the name of the driver to modify. List is a string list object previously populated with parameter values to add or change. An application must create and maintain List.

Populate List with parameter strings before calling ModifyDriver. List need only contain the parameters to change. If ModifyDriver passes a parameter for which a value already exists, the new parameter replaces the existing one. New parameters that are not

yet defined for the driver are added to the existing parameters. If List is empty, ModifyDriver does not change any existing parameters.

A session's ConfigMode (see page 10122) property determines whether a driver is local to the session, or is available to all applications in the BDE persistent store as long as any BDE clients are active. To save a driver permanently, call the SaveConfigFile (see page 10119) method before ending the session.

See Also

ConfigMode (see page 10122)

GetDriverParams (see page 10110)

Params (see page 9986)

53.1.15.4.27 TSession.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

53.1.15.4.28 TSession.Open Method

Starts a session and makes it the current session.

Pascal

```
procedure Open;
```

C++

```
__fastcall Open();
```

Description

Call Open to start a session other than the default session in a database application. Open sets Active (see page 10120) to true, triggers an OnStartup (see page 10129) event handler for the session and initializes the NetFileDir (see page 10125),

PrivateDir (see page 10125), and ConfigMode (see page 10122) properties.

See Also

Active (see page 10120)

NetFileDir (see page 10125)

PrivateDir (see page 10125)

ConfigMode (see page 10122)

OnStartup (see page 10129)

Close (see page 10103)

OpenDatabase (see page 10117)

Fetching the Data using TSQLDataSet

Connecting to Databases with TDatabase

Using TQuery

Using TSimpleDataSet

Using TSQLQuery

Using TSQLStoredProc

53.1.15.4.29 TSession.OpenDatabase Method

Opens an existing database, or creates a temporary database component and opens it.

Pascal

```
function OpenDatabase(const DatabaseName: string): TDatabase;
```

C++

```
__fastcall TDatabase OpenDatabase(const AnsiString DatabaseName);
```

Description

Call OpenDatabase to connect to a database for which a persistent database component already exists, or to create a temporary database component and connect to a database. DatabaseName specifies the database to open.

OpenDatabase makes the current session active, then calls FindDatabase (see page 10105) to determine if the DatabaseName parameter corresponds to the DatabaseName property of an existing database component. If it does not, OpenDatabase creates a temporary database component, assigning the DatabaseName parameter to the DatabaseName property. Finally, OpenDatabase calls the Open (see page 10116) method of the database to connect to a database server, then increments the session's database reference count by one.

See Also

CloseDatabase (see page 10103)

FindDatabase (see page 10105)

DatabaseName (see page 9980)

Open (see page 6866)

Activating a Session

Opening Database Connections

53.1.15.4.30 TSession.RemoveAllPasswords Method

Deletes all passwords previously added to the current session for accessing encrypted Paradox tables.

Pascal

```
procedure RemoveAllPasswords;
```

C++

```
__fastcall RemoveAllPasswords();
```

Description

Call RemoveAllPasswords to delete all current Paradox table passwords associated with the current session. Subsequent attempts to access encrypted Paradox tables fail unless an application first calls AddPassword (see page 10102) to reestablish a password.

Note: Ordinarily an application should call RemoveAllPasswords to remove all session-specific passwords it creates from the Borland (see page 1761) Database Engine (BDE) persistent store before terminating.

See Also

AddPassword (see page 10102)

RemovePassword (see page 10118)

PasswordDialog (see page 9849)

Working with Password-protected Paradox and dBASE Tables

53.1.15.4.31 TSession.RemovePassword Method

Deletes a single password previously added to the current session for accessing an encrypted Paradox table.

Pascal

```
procedure RemovePassword(const Password: string);
```

C++

```
__fastcall RemovePassword(const AnsiString Password);
```

Description

Call RemovePassword to delete a single Paradox table password associated with the current session from the Borland (see page 1761) Database Engine (BDE) persistent store. Subsequent attempts to access the encrypted Paradox table fails unless an application first calls AddPassword (see page 10102) to reestablish a password.

```
Session.AddPassword('secret');
Session.RemovePassword('secret');
Session->AddPassword("secret");
Session->RemovePassword("secret");
```

See Also

AddPassword (see page 10102)

RemoveAllPasswords (see page 10118)

PasswordDialog (see page 9849)

Working with Password-protected Paradox and dBASE Tables

53.1.15.4.32 TSession.SaveConfigFile Method

Writes the current information in the Borland (see page 1761) Database Engine (BDE) persistent store from memory to the BDE configuration file on disk.

Pascal

```
procedure SaveConfigFile;
```

C++

```
__fastcall SaveConfigFile();
```

Description

Call SaveConfigFile to write any BDE configuration changes currently in memory to the BDE configuration file on disk. When the BDE is first initialized, the current BDE configuration file is read into persistent storage in memory. Applications can read or modify the configuration file in memory, but modification exist only in memory, not on disk.

For example, if an application defines a new BDE alias with AddAlias (see page 10100), the alias exists in memory only for as long as the BDE resides in memory. To store the new alias for later use by all applications that use the BDE, call SaveConfigFile. SaveConfigFile overwrites the disk image of the configuration file with the in-memory image of the persistent configuration file.

Note: If a session's ConfigMode (see page 10122) property is cmSession (see page 10268), added aliases cannot be saved to the BDE configuration file.

See Also

ConfigMode (see page 10122)

AddAlias (see page 10100)

Working with BDE Aliases

53.1.15.4.33 TSession.SetName Method

Sets the value of the Name property.

Pascal

```
procedure SetName(const NewName: TComponentName); override;
```

C++

```
virtual __fastcall SetName(const TComponentName NewName);
```

Description

SetName is the virtual property setter for the Name property. It calls the ChangeName method, which performs the actual name change.

ChangeName is not virtual; override SetName instead when modifying the behavior of the Name property.

Warning: Using SetName to change the name of a component at runtime causes any references to the old name to become undefined. Any subsequent code that uses the old name will cause an exception.

See Also

ChangeName (see page 2770)

Name (see page 2805)

ValidateRename (see page 2792)

53.1.15.5 TSession Properties

The properties of the TSession class are listed here.

Protected Properties

	Name	Description
	BDEOwnsLoginCbDb (see page 10122)	This is BDEOwnsLoginCbDb, a member of class TSession.

Public Properties

	Name	Description
	ConfigMode (see page 10122)	Specifies how the Borland (see page 1761) Database Engine (BDE) should handle aliases for the session.
R	DatabaseCount (see page 10123)	Indicates the number of active database components currently associated with the session.
R	Databases (see page 10123)	Provides an indexed array of all active database component names for a session.
R	Handle (see page 10123)	Specifies the Borland (see page 1761) Database Engine (BDE) handle for the session.
R	Locale (see page 10124)	Identifies the Borland (see page 1761) Database Engine (BDE) language driver for the session component.
	TraceFlags (see page 10127)	Specifies the database operations to track with the SQL Monitor at runtime.

Published Properties

	Name	Description
	Active (see page 10120)	Specifies whether or not a session is active.
	AutoSessionName (see page 10121)	Indicates whether or not a unique session name is automatically generated for a session.
	KeepConnections (see page 10124)	Specifies whether or not a temporary database component created in the context of a session maintains a database server connection even if there are no active datasets associated with the database component.
	NetFileDir (see page 10125)	Specifies the directory that contains the Borland (see page 1761) Database Engine (BDE) network control file, PDOXUSRS.NET.
	PrivateDir (see page 10125)	Specifies the directory in which to store temporary table processing files generated by the Borland (see page 1761) Database Engine (BDE) for database components associated with a session.
	SessionName (see page 10126)	Specifies a unique session name that can be used by database and dataset components to link to this session.
	SQLHourGlass (see page 10126)	Indicates whether or not the mouse cursor changes to an hourglass during BDE operations.

53.1.15.5.1 TSession.Active Property

Specifies whether or not a session is active.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Examine Active at runtime for the default session, Session (see page 10265), to determine if it is active or not. Active is false by default, meaning there are no open database connections or datasets associated with the session. If Active is true, the session is active even if there are currently no active database connections.

Set the Active property at design time or runtime for additional session components in a database application. Setting Active to true starts the session and makes it the current session. Setting Active to true triggers an OnStartup (see page 10129) event handler for the session and initializes the NetFileDir (see page 10125), PrivateDir (see page 10125), and ConfigMode (

see page 10122) properties.

Setting Active to false (the default), closes any open datasets, and disconnects from attached database servers.

Note: An application can close the default session by setting Active to false, but this is not recommended.

See Also

ConfigMode (see page 10122)

NetFileDir (see page 10125)

PrivateDir (see page 10125)

Close (see page 10103)

Open (see page 10116)

Activating a Session

Debugging dbExpress Applications using TSQLMonitor

53.1.15.5.2 TSession.AutoSessionName Property

Indicates whether or not a unique session name is automatically generated for a session.

Pascal

```
property AutoSessionName: Boolean;
```

C++

```
__property Boolean AutoSessionName;
```

Description

Set AutoSessionName to control whether or not a unique session name is automatically generated. AutoSessionName is intended to guarantee developers of multi-threaded applications that sessions spawned for each thread are unique at runtime.

When AutoSessionName is false (the default), the application must set the SessionName (see page 10126) property for a session component. When AutoSessionName is true, TSession (see page 10093) assigns the SessionName (see page 10126) property automatically and replicates the session name across the SessionName (see page 10126) properties of all dataset components in the data module or form where the session component appears. This allows applications to use sessions in data modules that are replicated over multiple threads. TSession (see page 10093) constructs a session name by taking the current value of the Name property and appending an underscore (_) followed by a numeric value. For example, if Name is CustomerSession, then the AutoSessionName might be CustomerSession_2.

Note: The following restrictions apply to AutoSessionName:

- You cannot set AutoSessionName for a session in a data module or form that contains more than one session component.
- You cannot add a session component to a data module or form that already contains a session component with AutoSessionName set to true.
- You cannot directly set the SessionName (see page 10126) property of a session when AutoSessionName is true.

See Also

Name (see page 2805)

SessionName (see page 10126)

Managing Multiple Sessions

Adding a Session to the Web Module

53.1.15.5.3 TSession.BDEOwnsLoginCbDb Property

Pascal

```
property BDEOwnsLoginCbDb: Boolean;
```

C++

```
__property Boolean BDEOwnsLoginCbDb;
```

Description

This is BDEOwnsLoginCbDb, a member of class TSession.

53.1.15.5.4 TSession.ConfigMode Property

Specifies how the Borland (see page 1761) Database Engine (BDE) should handle aliases for the session.

Pascal

```
property ConfigMode: TConfigMode;
```

C++

```
__property TConfigMode ConfigMode;
```

Description

Set ConfigMode to specify how the BDE should handle aliases created using the AddAlias (see page 10100) or AddStandardAlias (see page 10102) methods within the context of a session. ConfigMode also determines which BDE aliases are reported to a session when an application calls the GetAliasNames (see page 10107) method.

The default setting for ConfigMode is cmAll (see page 10268), which is a constant that includes all possible set items defined in TConfigModes (see page 10253). The following table summarizes the possible BDE configuration modes that can be set for aliases:

Mode	Meaning
cfmVirtual	All aliases in the BDE configuration file, the BDE persistent store, and aliases local to the session are available to the session.
cfmPersistent	Only aliases in the BDE configuration file, or that are added to the persistent store are available to the session.
cfmSession	Only aliases created within this session are available to the session.

The main purpose of ConfigMode is to enable an application to specify and restrict aliases at the session level.

For backward compatibility with the older TConfigMode (see page 10260), use the constants cmVirtual (see page 10269), cmPersistent (see page 10268), cmSession (see page 10268), and cmAll (see page 10268). The first three correspond to the three TConfigModes (see page 10253) values of the same names. The constant cmAll (see page 10268) represents a ConfigMode setting of all three. The two command lines below, the first using TConfigModes (see page 10253) values and the second using the corresponding constant, are functionally the same.

```
Session.ConfigMode := [cfmSession];
Session.ConfigMode := cmSession;
```

Note: Aliases created within a session are not automatically saved to the BDE configuration file. To save an alias to the configuration file, call the SaveConfigFile (see page 10119) method.

See Also

AddAlias (see page 10100)

AddStandardAlias (see page 10102)

GetAliasNames (see page 10107)

SaveConfigFile (see page 10119)

53.1.15.5.5 TSession.DatabaseCount Property

Indicates the number of active database components currently associated with the session.

Pascal

```
property DatabaseCount: Integer;
```

C++

```
__property int DatabaseCount;
```

Description

Examine DatabaseCount to determine the number of active database components associated with a session. As database connections are opened or closed during a session's life-span, this number can change. If DatabaseCount is zero, there are currently no active database components for the session.

DatabaseCount is typically used with the Databases (see page 10123) property to iterate through the current set of active databases in a session.

See Also

Databases (see page 10123)

TDatabase (see page 9964)

53.1.15.5.6 TSession.Databases Property

Provides an indexed array of all active database component names for a session.

Pascal

```
property Databases [Index: Integer]: TDatabase;
```

C++

```
__property TDatabase Databases[int Index];
```

Description

Use Databases to access active database components associated with a session. An active database is one that is currently connected to a database server.

Databases is typically used with the DatabaseCount (see page 10123) property to iterate through the current set of active databases in a session.

See Also

DatabaseCount (see page 10123)

53.1.15.5.7 TSession.Handle Property

Specifies the Borland (see page 1761) Database Engine (BDE) handle for the session.

Pascal

```
property Handle: HDBISES;
```

C++

```
__property HDBISES Handle;
```

Description

Use Handle only to bypass TSession (see page 10093) methods and write directly to the BDE API. Many BDE function calls require a handle parameter. Handle is assigned an initial value when a session is activated.

Note: Do not use this property unless an application requires BDE functionality not available through library components.

See Also

Locale (see page 10124)

53.1.15.5.8 TSession.KeepConnections Property

Specifies whether or not a temporary database component created in the context of a session maintains a database server connection even if there are no active datasets associated with the database component.

Pascal

```
property KeepConnections: Boolean;
```

C++

```
__property Boolean KeepConnections;
```

Description

Use KeepConnections to specify whether or not temporary database components created at runtime maintain server connections even when they have no active datasets. If KeepConnections is true (the default), the application maintains database connections until the application exits or calls the DropConnections (see page 10105) method. For connections to remote database servers, or for applications that frequently open and close datasets, KeepConnections should remain true to reduce network traffic and speed up processing.

When KeepConnections is false, an application disconnects from a database when all datasets associated with a database component are closed. Dropping a connection releases system resources allocated to the connection, but if a dataset is later reopened that uses the same database, the connection must be reestablished and initialized.

Note: Duration of server connection for a persistent database component is determined by the database component's KeepConnection property instead of the session's KeepConnections property.

See Also

DropConnections (see page 10105)

TDatabase (see page 9964)

KeepConnection (see page 9984)

Understanding Database and Session Component Interactions

Specifying Default Database Connection Behavior

53.1.15.5.9 TSession.Locale Property

Identifies the Borland (see page 1761) Database Engine (BDE) language driver for the session component.

Pascal

```
property Locale: TLocale;
```

C++

```
__property TLocale Locale;
```

Description

Examine Locale to determine the BDE language driver used by the session component. Applications that make direct calls to the BDE may need to pass Locale information as an API function parameter.

Do not use this property unless an application requires BDE functionality not available through standard data access components.

See Also

Handle (see page 10123)

53.1.15.5.10 TSession.NetFileDir Property

Specifies the directory that contains the Borland (see page 1761) Database Engine (BDE) network control file, PDOXUSRS.NET.

Pascal

```
property NetFileDir: string;
```

C++

```
__property AnsiString NetFileDir;
```

Description

Use NetFileDir to specify the directory that contains the BDE network control file, PDOXUSRS.NET. This file governs the sharing of Paradox tables on network drives. All applications that share Paradox tables on a network directory must specify the same value for this property, and must have read, write, and create rights for the directory.

The library derives an initial value for NetFileDir from the BDE configuration file for the Paradox driver. Any value assigned to NetFileDir at design time or runtime overrides the BDE configuration setting.

Note: Any value you assign to NetFileDir is not saved to the BDE configuration file. (Even if you call the SaveConfigFile (see page 10119) method.)

See Also

PrivateDir (see page 10125)

53.1.15.5.11 TSession.PrivateDir Property

Specifies the directory in which to store temporary table processing files generated by the Borland (see page 1761) Database Engine (BDE) for database components associated with a session.

Pascal

```
property PrivateDir: string;
```

C++

```
__property AnsiString PrivateDir;
```

Description

Use PrivateDir to set the directory in which to store temporary table processing files for all database connections, such as those generated by the BDE to handle local SQL statements. Ordinarily this value is only set at runtime, so that a user's local hard disk is used to store temporary files. Local storage of these files improves performance. If no value is specified for PrivateDir, the BDE automatically stores temporary files in the directory current at the time the BDE was initialized.

Note: For applications that run directly from a networked file server, the application should set PrivateDir to a user's local drive to improve performance and to prevent temporary files from being created on the server where they might conflict with temporary files created by other instances of the application.

See Also

NetFileDir (see page 10125)

53.1.15.5.12 TSession.SessionName Property

Specifies a unique session name that can be used by database and dataset components to link to this session.

Pascal

```
property SessionName: string;
```

C++

```
__property AnsiString SessionName;
```

Description

Set SessionName to a unique and meaningful name that can be used to link database and dataset components to the session. Database and dataset components each have corresponding SessionName properties that must either match the SessionName property of an active session or that be blank, indicating that they should be associated with the default session, Session (see page 10265).

Note: If the AutoSessionName (see page 10121) property is true, an application cannot set SessionName directly.

See Also

TDatabase (see page 9964)

TDataSet (see page 6894)

SessionName (see page 9989)

SessionName (see page 10027)

AutoSessionName (see page 10121)

Associating a Database Component with a Session

Connecting to Databases with TDatabase

Using TQuery

Using TStoredProc

Using TTable

53.1.15.5.13 TSession.SQLHourGlass Property

Indicates whether or not the mouse cursor changes to an hourglass during BDE operations.

Pascal

```
property SQLHourGlass: Boolean;
```

C++

```
__property Boolean SQLHourGlass;
```

Description

Set SQLHourGlass to false if you do not want the mouse cursor to change to an hourglass while the BDE is waiting for an SQL

server to perform an operation. This property must be set before the session is opened.

53.1.15.5.14 TSession.TraceFlags Property

Specifies the database operations to track with the SQL Monitor at runtime.

Pascal

```
property TraceFlags: TTraceFlags;
```

C++

```
__property TTraceFlags TraceFlags;
```

Description

Use TraceFlags to specify which database operations the SQL Monitor should track in an application at runtime. TraceFlags is only meaningful for the SQL Monitor, which is provided to enable performance tuning and SQL debugging when working with remote SQL database servers.

Note: Normally trace options are set from the SQL Monitor rather than setting TraceFlags in application code.

The value of a session component's TraceFlags property determines the initial settings of the TraceFlags property for database components associated with the session.

The TTraceFlags (see page 10263) type defines the individual values that can be included in the TraceFlags property. The following table summarizes those values:

Values	Meaning
tfQPrepare	Monitor Prepare statements.
tfQExecute	Monitor ExecSQL statements.
tfError	Monitor server error messages. Such messages may include an error code.
tfStmt	Monitor all SQL statements.
tfConnect	Monitor database connect and disconnect operations, including allocation of connection handles, and freeing connection handles.
tfTransact	Monitor transaction statements, such as StartTransaction, Commit, and Rollback.
tfBlob	Monitor operations on blob data types.
tfMisc	Monitor any statements not covered by other flag options.
tfVendor	Monitor direct API function calls to the database server.
tfDataIn	Monitor data received from a server.
tfDataOut	Monitor data sent to a server.

Because TraceFlags is a set property, an application can specify different combinations of flags to monitor different combinations of statements. For example, the following statement limits monitoring to database connections and SQL statement preparation:

```
TraceFlags := [tfConnect, tfQPrepare];
TraceFlags << tfConnect << tfQPrepare;
```

See Also

SessionName (see page 10126)

TraceFlags (see page 9990)

53.1.15.6 TSession Events

The events of the TSession class are listed here.

Protected Events

	Name	Description
	OnDBNotify (see page 10128)	This is OnDBNotify, a member of class TSession.

Published Events

	Name	Description
	OnPassword (see page 10128)	Occurs when an application attempts to open a Paradox table for the first time and the Borland (see page 1761) Database Engine (BDE) reports insufficient access rights.
	OnStartup (see page 10129)	Occurs when an application activates a session.

53.1.15.6.1 TSession.OnDBNotify Event

Pascal

```
property OnDBNotify: TDatabaseNotifyEvent;
```

C++

```
__property TDatabaseNotifyEvent OnDBNotify;
```

Description

This is OnDBNotify, a member of class TSession.

53.1.15.6.2 TSession.OnPassword Event

Occurs when an application attempts to open a Paradox table for the first time and the Borland (see page 1761) Database Engine (BDE) reports insufficient access rights.

Pascal

```
property OnPassword: TPasswordEvent;
```

C++

```
__property TPasswordEvent OnPassword;
```

Description

Write an OnPassword event handler to take specific action when an application attempts to open a password-protected Paradox table for the first time. To gain access to the Paradox table, the event handler must pass a valid password to the BDE. The event handler should call the AddPassword (see page 10102) method to make the password available to the session.

Use Continue to conditionally finalize adding the password. If Continue is set to True, the password addition is attempted. If set to False, the attempt to add the password is abandoned.

Note: If an OnPassword event does not exist, but the BDE reports insufficient access rights, the library displays a default dialog box that prompts for a password.

See Also

AddPassword (see page 10102)

GetPassword (see page 10111)

RemovePassword (see page 10118)

PasswordDialog (see page 9849)

Understanding Database and Session Component Interactions

Working with Password-protected Paradox and dBASE Tables

53.1.15.6.3 TSession.OnStartup Event

Occurs when an application activates a session.

Pascal

```
property OnStartup: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnStartup;
```

Description

Write an OnStartup event handler to take specific actions when an application activates a session. A session is activated by setting its Active (see page 10120) property to true, or by opening a database or dataset associated with the session (both of which set Active (see page 10120) to true).

See Also

Active (see page 10120)

Open (see page 10116)

OpenDatabase (see page 10117)

Activating a Session

53.1.16 TSessionList Class

TSessionList manages one or more Borland (see page 1761) Database Engine (BDE) sessions in a database application.

Class Hierarchy

Pascal

```
TSessionList = class(TObject);
```

C++

```
class TSessionList : public TObject;
```

File

DBTables

Description

Use TSessionList to manage session components in applications that provide multiple sessions. The library automatically creates a globally accessible default session list component named Sessions (see page 10134) for all database applications. In a multi-session application there can be more than one active session, but only one session can be current at a time. TSessionList enables an application to switch current sessions.

Note: Applications should not instantiate additional session list components.

See Also

TSession (see page 10093)

Sessions (see page 10266)

53.1.16.1 TSessionList Members

The following tables list the members exposed by TSessionList.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 10131)	Creates an instance of a TSessionList (see page 10129) component.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 10131)	Destroys the instance of a session list component.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindSession (see page 10132)	Searches the list of sessions for a specified session.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetSessionNames (see page 10132)	Populates a string list with the names of all available sessions.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	OpenSession (see page 10132)	Makes an existing session active, or creates a new session component and makes it active.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Count (see page 10133)	Indicates the number of available sessions for the application.
	CurrentSession (see page 10134)	Returns a pointer to the current session component
R	List (see page 10134)	Returns a specified session component.
R	Sessions (see page 10134)	Lists all active session components in the session list.

53.1.16.2 TSessionList.Create Constructor

Creates an instance of a TSessionList (see page 10129) component.

Pascal

```
constructor Create;
```

C++

```
__fastcall TSessionList();
```

Description

Do not create instances of TSessionList (see page 10129). A default session list component, Sessions (see page 10134), is automatically created for all database applications. Secondary session lists are not necessary.

Create calls its inherited constructor, creates an empty list object to keep track of session components, and both creates and initializes a critical section for opening databases.

See Also

Destroy (see page 10131)

53.1.16.3 TSessionList.Destroy Destructor

Destroys the instance of a session list component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSessionList();
```

Description

Do not call Destroy. Delphi automatically handles the destruction of the default session list component, Sessions (see page 10134), when an application closes.

Destroy closes all session components, deletes the BDE critical section, and calls its inherited destructor.

See Also

Create (see page 10131)

Free (see page 22300)

53.1.16.4 TSessionList Methods

The methods of the TSessionList class are listed here.

Public Methods

	Name	Description
	FindSession (see page 10132)	Searches the list of sessions for a specified session.
	GetSessionNames (see page 10132)	Populates a string list with the names of all available sessions.
	OpenSession (see page 10132)	Makes an existing session active, or creates a new session component and makes it active.

53.1.16.4.1 TSessionList.FindSession Method

Searches the list of sessions for a specified session.

Pascal

```
function FindSession(const SessionName: string): TSession;
```

C++

```
__fastcall TSession FindSession(const AnsiString SessionName);
```

Description

Call FindSession to determine whether or not a specified session exists. SessionName specifies the session to search for.

FindSession compares the SessionName parameter to the SessionName property for each session component in the session list. If a match is found, FindSession returns a pointer to the session component. If an application passes an empty string in SessionName, FindSession returns the default session, Session (see page 10265). If a match is not found, FindSession returns nil (Delphi) or NULL (C++).

See Also

List (see page 10134)

GetSessionNames (see page 10132)

SessionName (see page 10126)

TSession (see page 10093)

53.1.16.4.2 TSessionList.GetSessionNames Method

Populates a string list with the names of all available sessions.

Pascal

```
procedure GetSessionNames(List: TStrings);
```

C++

```
__fastcall GetSessionNames(TStrings List);
```

Description

Call GetSessionNames to retrieve a list of all available sessions. List (see page 10134) is a string list object, created and maintained by the application, into which to store session names. Names returned by GetSessionNames correspond to the SessionName properties of all available sessions.

See Also

FindSession (see page 10132)

SessionName (see page 10126)

53.1.16.4.3 TSessionList.OpenSession Method

Makes an existing session active, or creates a new session component and makes it active.

Pascal

```
function OpenSession(const SessionName: string): TSession;
```

C++

```
__fastcall TSession OpenSession(const AnsiString SessionName);
```

Description

Call OpenSession to make an existing session active, or to create a new session and make it active. SessionName specifies the name of the session to open.

OpenSession calls FindSession (see page 10132) to see if the session specified in the SessionName parameter already exists. If it finds a match, it starts that session if necessary, and makes the session active. If OpenSession does not find an existing session, it creates a new session using the name specified in the SessionName parameter, starts the session, and makes it active.

In either case, OpenSession returns a session object.

See Also

List (see page 10134)

Sessions (see page 10134)

FindSession (see page 10132)

53.1.16.5 TSessionList Properties

The properties of the TSessionList class are listed here.

Public Properties

	Name	Description
R	Count (see page 10133)	Indicates the number of available sessions for the application.
	CurrentSession (see page 10134)	Returns a pointer to the current session component
R	List (see page 10134)	Returns a specified session component.
R	Sessions (see page 10134)	Lists all active session components in the session list.

53.1.16.5.1 TSessionList.Count Property

Indicates the number of available sessions for the application.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read Count to determine the current number of available sessions. Count is typically used to iterate through the list of available sessions to perform some universal action on all sessions. Used with the Sessions (see page 10134) property, which enables an application to access a session by index number, Count specifies the maximum number of index entries for Sessions (see page 10134) (Count - 1).

See Also

CurrentSession (see page 10134)

List (see page 10134)

Sessions (see page 10134)

53.1.16.5.2 TSessionList.CurrentSession Property

Returns a pointer to the current session component

Pascal

```
property CurrentSession: TSession;
```

C++

```
__property TSession CurrentSession;
```

Description

Use CurrentSession to determine or specify which session in a database application is the current session. By default the current session is a pointer to the default session component, Session (see page 10265), that is created automatically by the library for all database applications. Setting CurrentSession makes the specified session component the current one. While any number of sessions may be active, only one session can be current at a time.

See Also

List (see page 10134)

FindSession (see page 10132)

OpenSession (see page 10132)

53.1.16.5.3 TSessionList.List Property

Returns a specified session component.

Pascal

```
property List [const SessionName: string]: TSession;
```

C++

```
__property TSession List[AnsiString const SessionName];
```

Description

Use List to return a pointer to a specific session component managed by the session list. List does not change which session is current. SessionName specifies the session component to return, and must match the SessionName property of an existing session component. If no such session exists, List raises an EDatabaseError exception with a message specifying the invalid session name.

Note: To prevent raising an exception, call the FindSession (see page 10132) method instead of using the List property.

See Also

CurrentSession (see page 10134)

Sessions (see page 10134)

FindSession (see page 10132)

SessionName (see page 10126)

EDatabaseError (see page 6731)

53.1.16.5.4 TSessionList.Sessions Property

Lists all active session components in the session list.

Pascal

```
property Sessions [Index: Integer]: TSession;
```

C++

```
__property TSession Sessions[int Index];
```

Description

Use Sessions to access available session components. Index specifies the session to access. Index must be in the range of zero to the current value of the Count (see page 10133) property minus one, or an EListError exception is raised describing the out of bounds condition.

For example, an application might iterate through all available sessions in the list to perform a global action on them.

See Also

Count (see page 10133)

EListError (see page 2701)

operator_[]

53.1.17 TSQLUpdateObject Class

TSQLUpdateObject is the base class for classes that use an SQL command to apply cached updates on behalf of queries or stored procedures that can't post updates directly.

Class Hierarchy

Pascal

```
TSQLUpdateObject = class(TDataSetUpdateObject);
```

C++

```
class TSQLUpdateObject : public TDataSetUpdateObject;
```

File

DBTables

Description

Use TSQLUpdateObject as a base class when creating customized update objects that can update BDE datasets. TSQLUpdateObject adds support for accessing a SQL statement that applies an update to the properties and methods introduced by TDataSetUpdateObject (see page 9994).

TSQLUpdateObject is an abstract base class and cannot be instantiated.

See Also

TUpdateSQL (see page 10229)

EAbstractError (see page 22722)

53.1.17.1 TSQLUpdateObject Members

The following tables list the members exposed by TSQLUpdateObject.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
A	GetDataSet (see page 9998)	GetDataSet returns the value of the DataSet (see page 9999) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
A	GetSQL (see page 10138)	Returns the SQL statement that performs a specified update operation.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
A	SetDataSet (see page 9999)	SetDataSet sets the value of the DataSet (see page 9999) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.

	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DataSet (see page 9999)	Specifies the dataset that stores the cached updates to be applied by the update object.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
A	Apply (see page 9998)	Apply applies changes to the dataset specified by the DataSet (see page 9999) property.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.

	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

53.1.17.2 TSQLUpdateObject Methods

The methods of the TSQLUpdateObject class are listed here.

Protected Methods

	Name	Description
A	GetSQL (see page 10138)	Returns the SQL statement that performs a specified update operation.

53.1.17.2.1 TSQLUpdateObject.GetSQL Method

Returns the SQL statement that performs a specified update operation.

Pascal

```
function GetSQL(UpdateKind: TUpdateKind): TStrings; virtual; abstract;
```

C++

```
virtual __fastcall TStrings GetSQL(TUpdateKind UpdateKind) = 0;
```

Description

Override GetSQL to allow the update object to access an SQL statement that applies an update of the type specified by UpdateKind.

GetSQL provides the underlying implementation for the SQL property in the descendant class TUpdateSQL (see page 10229).

See Also

SQL (see page 10242)

53.1.18 TStoredProc Class

TStoredProc encapsulates a stored procedure in a BDE-based application.

Class Hierarchy

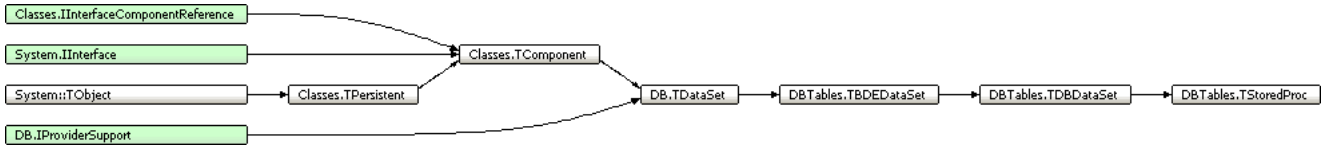

Pascal

```
TStoredProc = class(TDBDataSet);
```

C++

```
class TStoredProc : public TDBDataSet;
```

File

DBTables

Description

Use a TStoredProc object in BDE-based applications to use a stored procedure on a database server. A stored procedure is a grouped set of statements, stored as part of a database server's metadata (just like tables, indexes, and domains), that performs a frequently repeated, database-related task on the server and passes results to the client.

Note: Not all database servers support stored procedures. See a specific server's documentation to determine if it supports stored procedures.

Many stored procedures require a series of input arguments, or parameters, that are used during processing. TStoredProc provides a Params (see page 10165) property that enables an application to set these parameters before executing the stored procedure.

TStoredProc reuses the Params (see page 10165) property to hold the results returned by a stored procedure. Params (see page 10165) is an array of values. Depending on server implementation, a stored procedure can return either a single set of values, or a result set similar to the result set returned by a query.

See Also

TQuery (see page 10059)

TTable (see page 10168)

Types of Datasets

53.1.18.1 TStoredProc Members

The following tables list the members exposed by TStoredProc.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	ActivateFilters (see page 9893)	This is ActivateFilters, a member of class TBDEDataSet.
	AddFieldDesc (see page 9893)	This is AddFieldDesc, a member of class TBDEDataSet.
	AllocCachedUpdateBuffers (see page 9893)	This is AllocCachedUpdateBuffers, a member of class TBDEDataSet.
	AllocKeyBuffers (see page 9893)	This is AllocKeyBuffers, a member of class TBDEDataSet.
V	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
V	AllocRecordBuffer (see page 9894)	This is AllocRecordBuffer, a member of class TBDEDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
V	BlockReadNext (see page 9894)	This is BlockReadNext, a member of class TBDEDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
	CachedUpdateCallBack (see page 9895)	This is CachedUpdateCallBack, a member of class TBDEDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
	CheckCachedUpdateMode (see page 9896)	This is CheckCachedUpdateMode, a member of class TBDEDataSet.
V	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
V	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
	CheckSetKeyMode (see page 9896)	This is CheckSetKeyMode, a member of class TBDEDataSet.
V	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
V	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
V	ClearCalcFields (see page 9897)	This is ClearCalcFields, a member of class TBDEDataSet.
V	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
V	CloseBlob (see page 9897)	This is CloseBlob, a member of class TBDEDataSet.
V	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
V	CloseCursor (see page 9897)	This is CloseCursor, a member of class TBDEDataSet.
V	CloseCursor (see page 10014)	This is CloseCursor, a member of class TDBDataSet.
	ConstraintsStored (see page 10015)	This is ConstraintsStored, a member of class TDBDataSet.
	CreateExprFilter (see page 9900)	This is CreateExprFilter, a member of class TBDEDataSet.
V	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
	CreateFuncFilter (see page 9900)	This is CreateFuncFilter, a member of class TBDEDataSet.
V	CreateHandle (see page 10154)	This is CreateHandle, a member of class TStoredProc.
	CreateLookupFilter (see page 9901)	This is CreateLookupFilter, a member of class TBDEDataSet.
V	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
V	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.

	DataEvent (see page 6924)	This is DataEvent, a member of class TDataSet.
	DataEvent (see page 9901)	This is DataEvent, a member of class TBDEDataSet.
	DeactivateFilters (see page 9901)	This is DeactivateFilters, a member of class TBDEDataSet.
	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
	DefineProperties (see page 10155)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
	DestroyHandle (see page 9901)	This is DestroyHandle, a member of class TBDEDataSet.
	DestroyLookupCursor (see page 9902)	This is DestroyLookupCursor, a member of class TBDEDataSet.
	Disconnect (see page 10156)	This is Disconnect, a member of class TStoredProc.
	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FindRecord (see page 9903)	This is FindRecord, a member of class TBDEDataSet.
	ForceUpdateCallback (see page 9904)	This is ForceUpdateCallback, a member of class TBDEDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeKeyBuffers (see page 9904)	This is FreeKeyBuffers, a member of class TBDEDataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
	FreeRecordBuffer (see page 9904)	This is FreeRecordBuffer, a member of class TBDEDataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
	GetBookmarkData (see page 9905)	This is GetBookmarkData, a member of class TBDEDataSet.
	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
	GetBookmarkFlag (see page 9905)	This is GetBookmarkFlag, a member of class TBDEDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
	GetCanModify (see page 9905)	This is GetCanModify, a member of class TBDEDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.

	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetFieldFullName (see page 9907)	This is GetFieldFullName, a member of class TBDEDataSet.
	GetHandle (see page 9907)	This is GetHandle, a member of class TBDEDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIndexField (see page 9907)	This is GetIndexField, a member of class TBDEDataSet.
	GetIndexFieldCount (see page 9907)	This is GetIndexFieldCount, a member of class TBDEDataSet.
	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
	GetIsIndexField (see page 9908)	This is GetIsIndexField, a member of class TBDEDataSet.
	GetKeyBuffer (see page 9908)	This is GetKeyBuffer, a member of class TBDEDataSet.
	GetKeyExclusive (see page 9908)	This is GetKeyExclusive, a member of class TBDEDataSet.
	GetKeyFieldCount (see page 9909)	This is GetKeyFieldCount, a member of class TBDEDataSet.
	GetLookupCursor (see page 9909)	This is GetLookupCursor, a member of class TBDEDataSet.
	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetObjectTypeNames (see page 9909)	This is GetObjectTypeNames, a member of class TBDEDataSet.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetParamsCount (see page 10156)	This is GetParamsCount, a member of class TStoredProc.
	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
	GetRecNo (see page 9909)	This is GetRecNo, a member of class TBDEDataSet.
	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
	GetRecord (see page 9910)	This is GetRecord, a member of class TBDEDataSet.
	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
	GetRecordCount (see page 9910)	This is GetRecordCount, a member of class TBDEDataSet.
	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
	GetRecordSize (see page 9910)	This is GetRecordSize, a member of class TBDEDataSet.
	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
	GetStateFieldValue (see page 9910)	This is GetStateFieldValue, a member of class TBDEDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetUpdateRecordSet (see page 9910)	This is GetUpdateRecordSet, a member of class TBDEDataSet.
	GetUpdatesPending (see page 9911)	This is GetUpdatesPending, a member of class TBDEDataSet.
	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
	InitKeyBuffer (see page 9911)	This is InitKeyBuffer, a member of class TBDEDataSet.
	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
	InitRecord (see page 9911)	This is InitRecord, a member of class TBDEDataSet.
	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
	InternalAddRecord (see page 9911)	This is InternalAddRecord, a member of class TBDEDataSet.
	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
	InternalCancel (see page 9912)	This is InternalCancel, a member of class TBDEDataSet.
	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.
	InternalClose (see page 9912)	This is InternalClose, a member of class TBDEDataSet.
	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
	InternalDelete (see page 9912)	This is InternalDelete, a member of class TBDEDataSet.
	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
	InternalEdit (see page 9912)	This is InternalEdit, a member of class TBDEDataSet.
	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
	InternalFirst (see page 9912)	This is InternalFirst, a member of class TBDEDataSet.
	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.

	InternalGotoBookmark (see page 9913)	This is InternalGotoBookmark, a member of class TBDEDataSet.
	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
	InternalHandleException (see page 9913)	This is InternalHandleException, a member of class TBDEDataSet.
	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
	InternalInitFieldDefs (see page 9913)	This is InternalInitFieldDefs, a member of class TBDEDataSet.
	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
	InternalInitRecord (see page 9913)	This is InternalInitRecord, a member of class TBDEDataSet.
	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
	InternalInsert (see page 9914)	This is InternalInsert, a member of class TBDEDataSet.
	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
	InternalLast (see page 9914)	This is InternalLast, a member of class TBDEDataSet.
	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
	InternalOpen (see page 9914)	This is InternalOpen, a member of class TBDEDataSet.
	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
	InternalPost (see page 9914)	This is InternalPost, a member of class TBDEDataSet.
	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
	InternalRefresh (see page 9914)	This is InternalRefresh, a member of class TBDEDataSet.
	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	InternalSetToRecord (see page 9915)	This is InternalSetToRecord, a member of class TBDEDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
	IsCursorOpen (see page 9915)	This is IsCursorOpen, a member of class TBDEDataSet.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 9916)	This is LocateRecord, a member of class TBDEDataSet.
	MapsToIndex (see page 9918)	This is MapsToIndex, a member of class TBDEDataSet.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	OpenCursor (see page 6976)	This is OpenCursor, a member of class TDataSet.
	OpenCursor (see page 9918)	This is OpenCursor, a member of class TBDEDataSet.
	OpenCursor (see page 10015)	This is OpenCursor, a member of class TDBDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PostKeyBuffer (see page 9919)	This is PostKeyBuffer, a member of class TBDEDataSet.
	PrepareCursor (see page 9919)	This is PrepareCursor, a member of class TBDEDataSet.
	ProcessUpdates (see page 9919)	This is ProcessUpdates, a member of class TBDEDataSet.
	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
	PSEndTransaction (see page 10016)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 10158)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 10017)	Executes a specified SQL command.
	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 10017)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 10158)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandType (see page 10159)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetParams (see page 10159)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.

	PSGetQuoteChar (see page 10018)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetTableName (see page 10160)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSGetUpdateException (see page 9920)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 10018)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 10018)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 9920)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 9921)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 10160)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 10161)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSStartTransaction (see page 10019)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 9921)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 10019)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	ResetCursorRange (see page 9922)	This is ResetCursorRange, a member of class TBDEDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBlockReadSize (see page 9922)	This is SetBlockReadSize, a member of class TBDEDataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkData (see page 9922)	This is SetBookmarkData, a member of class TBDEDataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkFlag (see page 9923)	This is SetBookmarkFlag, a member of class TBDEDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetCachedUpdates (see page 9923)	This is SetCachedUpdates, a member of class TBDEDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetCursorRange (see page 9923)	This is SetCursorRange, a member of class TBDEDataSet.
	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDBFlag (see page 10161)	This is SetDBFlag, a member of class TStoredProc.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldData (see page 9923)	This is SetFieldData, a member of class TBDEDataSet.
	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
	SetFilterData (see page 9923)	This is SetFilterData, a member of class TBDEDataSet.
	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.

	SetFiltered (see page 9924)	This is SetFiltered, a member of class TBDEDataSet.
	SetFilterHandle (see page 9924)	This is SetFilterHandle, a member of class TBDEDataSet.
	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
	SetFilterOptions (see page 9924)	This is SetFilterOptions, a member of class TBDEDataSet.
	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
	SetFilterText (see page 9924)	This is SetFilterText, a member of class TBDEDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetHandle (see page 10020)	This is SetHandle, a member of class TDBDataSet.
	SetIndexField (see page 9925)	This is SetIndexField, a member of class TBDEDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetKeyBuffer (see page 9925)	This is SetKeyBuffer, a member of class TBDEDataSet.
	SetKeyExclusive (see page 9925)	This is SetKeyExclusive, a member of class TBDEDataSet.
	SetKeyFieldCount (see page 9925)	This is SetKeyFieldCount, a member of class TBDEDataSet.
	SetKeyFields (see page 9925)	This is SetKeyFields, a member of class TBDEDataSet.
	SetLinkRanges (see page 9926)	This is SetLinkRanges, a member of class TBDEDataSet.
	SetLocale (see page 9926)	This is SetLocale, a member of class TBDEDataSet.
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetOnFilterRecord (see page 9926)	This is SetOnFilterRecord, a member of class TBDEDataSet.
	SetOnUpdateError (see page 9926)	This is SetOnUpdateError, a member of class TBDEDataSet.
	SetOverLoad (see page 10161)	This is SetOverLoad, a member of class TStoredProc.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetPrepare (see page 10162)	This is SetPrepare, a member of class TStoredProc.
	SetPrepared (see page 10162)	This is SetPrepared, a member of class TStoredProc.
	SetProcName (see page 10162)	This is SetProcName, a member of class TStoredProc.
	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
	SetRecNo (see page 9927)	This is SetRecNo, a member of class TBDEDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
	SetStateFieldValue (see page 9927)	This is SetStateFieldValue, a member of class TBDEDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	SetupCallBack (see page 9927)	This is SetupCallBack, a member of class TBDEDataSet.
	SetUpdateObject (see page 9927)	This is SetUpdateObject, a member of class TBDEDataSet.
	SetUpdateRecordSet (see page 9927)	This is SetUpdateRecordSet, a member of class TBDEDataSet.
	SwitchToIndex (see page 9928)	This is SwitchToIndex, a member of class TBDEDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	UpdateCallbackRequired (see page 9928)	This is UpdateCallbackRequired, a member of class TBDEDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
R	DBFlags (see page 10023)	Keeps track of the commands this dataset has active with the database.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
	Reserved (see page 7035)	For internal use only.
	UpdateMode (see page 10028)	Determines how the Borland (see page 1761) Database (see page 10021) Engine (BDE) finds records when updating to an SQL database.

Public Events

	Name	Description
	AfterCancel (see page 7038)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 7040)	Occurs after an application closes a dataset.
	AfterDelete (see page 7041)	Occurs after an application deletes a record.
	AfterEdit (see page 7044)	Occurs after an application starts editing a record.
	AfterInsert (see page 7044)	Occurs after an application inserts a new record.
	AfterOpen (see page 7045)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 7046)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 7047)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 7047)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 7048)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 7048)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 7049)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 7049)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 7050)	Occurs before an application enters insert mode.
	BeforeOpen (see page 7051)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 7052)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 7052)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 7053)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 7053)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 7054)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 7054)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 7056)	Occurs when an application inserts or appends a new dataset record.

	OnPostError (see page 7056)	Occurs when an application attempts to modify or insert a record and an exception is raised.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
	ApplyUpdates (see page 9894)	Writes a dataset's pending cached updates to the database.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
V	BookmarkValid (see page 9895)	Tests the validity of a specified bookmark.
V	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
V	Cancel (see page 9895)	Cancels modifications to the current record if those changes are not yet posted.
	CancelUpdates (see page 9896)	Clears all pending cached updates from the cache and restores the dataset its prior state.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	CheckOpen (see page 10014)	Checks the result of a call to the Borland (see page 1761) Database (see page 10021) Engine (BDE).
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Close (see page 6920)	Closes a dataset.
	CloseDatabase (see page 10014)	Closes a database connection associated with the database session.
	CommitUpdates (see page 9897)	Clears the cached updates buffer.
V	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
V	CompareBookmarks (see page 9898)	Indicates the relationship between two bookmarks.
	ConstraintCallBack (see page 9898)	Specifies the callback function that allows the dataset to communicate with the BDE constraint layer.
	ConstraintsDisabled (see page 9899)	Indicates whether constraints are enabled for a dataset.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
	CopyParams (see page 10154)	Copies a stored procedure's parameters into another parameter list.
V	Create (see page 10152)	Creates an instance of a stored procedure component.
V	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
V	CreateBlobStream (see page 9899)	Returns a TStream object for reading or writing the data in a specified blob field.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
	DescriptionsAvailable (see page 10155)	Determines if parameter information is available from the stored procedure on the server.
V	Destroy (see page 10153)	Destroys the instance of a stored procedure.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableConstraints (see page 9902)	Disables application of server constraints on the dataset.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableConstraints (see page 9902)	Enables application of server constraints against dataset entries.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	ExecProc (see page 10156)	Executes the stored procedure on the server.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FetchAll (see page 9903)	Retrieves all records from the current cursor position to the end of the file and stores them locally.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	FlushBuffers (see page 9903)	Posts all changes that have been written to the record buffer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
V	GetBlobFieldData (see page 9904)	Reads BLOB data into a buffer.
V	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
V	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
V	GetCurrentRecord (see page 9905)	Retrieves the current record into a buffer.
V	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
V	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
V	GetFieldData (see page 9906)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
V	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetIndexInfo (see page 9907)	Retrieves information about the current index into the index data fields of the dataset.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.

	GetResults (see page 10157)	Returns the output parameter values from a Sybase or MSSQL stored procedure.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
V	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
V	IsSequenced (see page 9915)	IsSequenced indicates whether sequence numbers are available for database records.
	Last (see page 6968)	Moves to the last record in the dataset.
V	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Locate (see page 9915)	Searches the dataset for a specified record and makes that record the current record.
V	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
V	Lookup (see page 9916)	Retrieves field values from a record that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	Open (see page 6976)	Opens the dataset.
	OpenDatabase (see page 10016)	Opens the database that contains the dataset.
	ParamByName (see page 10157)	Accesses parameter information based on a specified parameter name.
V	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
V	Post (see page 9918)	Writes a modified record to the database.
	Prepare (see page 10157)	Prepares a stored procedure for execution.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.

	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
	RevertRecord (see page 9922)	Restores the current record in the dataset to an unmodified state when cached updates are enabled.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
V	Translate (see page 9928)	Converts a data string between the Microsoft ANSI character set, and the local code page (OEM character set).
	UnPrepare (see page 10162)	Frees the resources allocated for a previously prepared stored procedure.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.
V	UpdateStatus (see page 9928)	Reports the update status for the current record.

Public Properties

	Name	Description
	Active (see page 7004)	Specifies whether or not a dataset is open.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
	CacheBlobs (see page 9932)	Determines whether BLOB fields are cached in memory.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Database (see page 10021)	Identifies the database component for which this dataset represents one or more tables.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
R	DataSource (see page 7014)	Represents the data source of another dataset that supplies values to the dataset.
R	DBHandle (see page 10023)	Specifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) database handle for the dataset.
R	DBLocale (see page 10024)	Identifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) language driver for the dataset component.
R	DBSession (see page 10024)	Identifies the database session component associated with this dataset.

R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	ExpIndex (see page 9933)	Indicates whether a dataset is using a dBASE expression index.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
	FieldDefs (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
R	Handle (see page 10163)	Specifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) cursor handle for the stored procedure.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	KeySize (see page 9934)	Specifies the size of the key for the current index of the dataset.
R	Locale (see page 9934)	Identifies the Borland (see page 1761) Database Engine (BDE) language driver for the dataset.
R	Modified (see page 7029)	Indicates whether the active record is modified.
	ObjectView (see page 7031)	Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	ParamCount (see page 10165)	Indicates the number of parameters for the stored procedure component.
	Prepared (see page 10165)	Determines whether or not a stored procedure is prepared for execution.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	StmtHandle (see page 10166)	Specifies the Borland Database Engine (BDE) statement handle for the dataset.
	UpdateRecordTypes (see page 9936)	Specifies the type of records visible in a dataset when cached updates are enabled.
R	UpdatesPending (see page 9937)	Indicates whether the cached updates buffer contains records that are not yet applied.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterCancel (see page 9938)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 9940)	Occurs after an application closes a dataset.
	AfterDelete (see page 9942)	Occurs after an application deletes a record.
	AfterEdit (see page 9944)	Occurs after an application starts editing a record.
	AfterInsert (see page 9945)	Occurs after an application inserts a new record.
	AfterOpen (see page 9946)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 9947)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 9947)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 9948)	Occurs after an application scrolls from one record to another.

	BeforeCancel (see page 9948)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 9949)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 9949)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 9950)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 9950)	Occurs before an application enters insert mode.
	BeforeOpen (see page 9952)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 9952)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 9953)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 9953)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 9954)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 9955)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 9955)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 10028)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 9955)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 9956)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnUpdateError (see page 9956)	Occurs if an exception is generated when cached updates are applied to a database.
	OnUpdateRecord (see page 9957)	Occurs when cached updates are applied to a record.

Published Properties

	Name	Description
	Active (see page 9929)	Specifies whether or not a dataset is open.
	AutoCalcFields (see page 9931)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	AutoRefresh (see page 10021)	Specifies whether server-generated field values are refetched automatically.
	CachedUpdates (see page 9932)	Specifies whether cached updates are enabled for a dataset.
	DatabaseName (see page 10022)	Specifies the name of the database associated with this dataset.
	Filter (see page 10025)	Specifies the text of the current filter for a dataset.
	Filtered (see page 10026)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 10026)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectView (see page 9935)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	Overload (see page 10163)	Specifies which Oracle overloaded stored procedure to execute.
	ParamBindMode (see page 10164)	Determines the order in which a component's parameters are assigned to the parameter list for the stored procedure on the server.
	Params (see page 10165)	Stores the input and output parameters for a stored procedure.
	SessionName (see page 10027)	Identifies the name of the session with which this dataset is associated.
	StoredProcName (see page 10167)	Identifies the name of the stored procedure on the server for which this object is an encapsulation.
	Tag (see page 2808)	Stores an integer value as part of a component.
	UpdateObject (see page 10167)	Specifies the update object component used to update a read-only result set when cached updates are enabled.

53.1.18.2 TStoredProc.Create Constructor

Creates an instance of a stored procedure component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TStoredProc(TComponent * AOwner);
```

Description

Call Create to instantiate a stored procedure declared in an application. Create calls its inherited Create constructor, creates an empty parameter list for the newly instantiated stored procedure, and initializes its parameter, server, and record buffers to nil (Delphi) or NULL (C++).

See Also

Destroy (see page 10153)

53.1.18.3 TStoredProc.Destroy Destructor

Destroys the instance of a stored procedure.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TStoredProc();
```

Description

Do not call Destroy directly. Instead, call Free, which verifies that the stored procedure is not nil before calling Destroy.

Destroy disconnects from the server, frees the parameter list, and calls its inherited Destroy destructor.

See Also

Create (see page 10152)

Params (see page 10165)

53.1.18.4 TStoredProc Methods

The methods of the TStoredProc class are listed here.

Protected Methods

	Name	Description
	CreateHandle (see page 10154)	This is CreateHandle, a member of class TStoredProc.
	DefineProperties (see page 10155)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	Disconnect (see page 10156)	This is Disconnect, a member of class TStoredProc.
	GetParamsCount (see page 10156)	This is GetParamsCount, a member of class TStoredProc.
	PSExecute (see page 10158)	Executes the SQL command associated with the dataset.
	PSGetCommandText (see page 10158)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandType (see page 10159)	Gets the command type to be executed by PSExecute.
	PSGetParams (see page 10159)	Returns the current parameter values of the dataset.
	PSGetTableName (see page 10160)	Returns the name of the dataset table that appears in generated SQL statements.
	PSSetCommandText (see page 10160)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetParams (see page 10161)	Assigns the specified parameter values to the dataset.
	SetDBFlag (see page 10161)	This is SetDBFlag, a member of class TStoredProc.
	SetOverLoad (see page 10161)	This is SetOverLoad, a member of class TStoredProc.
	SetPrepare (see page 10162)	This is SetPrepare, a member of class TStoredProc.
	SetPrepared (see page 10162)	This is SetPrepared, a member of class TStoredProc.

	SetProcName (see page 10162)	This is SetProcName, a member of class TStoredProc.

Public Methods

	Name	Description
	CopyParams (see page 10154)	Copies a stored procedure's parameters into another parameter list.
	DescriptionsAvailable (see page 10155)	Determines if parameter information is available from the stored procedure on the server.
	ExecProc (see page 10156)	Executes the stored procedure on the server.
	GetResults (see page 10157)	Returns the output parameter values from a Sybase or MSSQL stored procedure.
	ParamByName (see page 10157)	Accesses parameter information based on a specified parameter name.
	Prepare (see page 10157)	Prepares a stored procedure for execution.
	UnPrepare (see page 10162)	Frees the resources allocated for a previously prepared stored procedure.

53.1.18.4.1 TStoredProc.CopyParams Method

Copies a stored procedure's parameters into another parameter list.

Pascal

```
procedure CopyParams(Value: TParams);
```

C++

```
__fastcall CopyParams(TParams Value);
```

Description

Call CopyParams to copy this stored procedure's parameters into a separate parameter list object. Value is the parameter list into which to assign this stored procedure's parameters. Value can be the parameter list of another stored procedure. For example:

```
TStoredProc1.CopyParams(TStoredProc2.Params);
StoredProc1->CopyParams(StoredProc2->Params);
```

If the stored procedure is not prepared when an application calls CopyParams, CopyParams calls Prepare (see page 10157) before assigning the parameters to the target parameters list, and then calls UnPrepare (see page 10162) to return the stored procedure to its previous state.

See Also

Params (see page 10165)

Prepare (see page 10157)

UnPrepare (see page 10162)

53.1.18.4.2 TStoredProc.CreateHandle Method

Pascal

```
function CreateHandle: HDBICur; override;
```

C++

```
virtual __fastcall HDBICur CreateHandle();
```

Description

This is CreateHandle, a member of class TStoredProc.

53.1.18.4.3 TStoredProc.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor (see page 2839)

DefineBinaryProperty (see page 2837)

DefineProperties (see page 2931)

DefineProperty (see page 2838)

53.1.18.4.4 TStoredProc.DescriptionsAvailable Method

Determines if parameter information is available from the stored procedure on the server.

Pascal

```
function DescriptionsAvailable: Boolean;
```

C++

```
__fastcall Boolean DescriptionsAvailable();
```

Description

Call DescriptionsAvailable to determine if a stored procedure on a server can return parameter information to the application. If DescriptionsAvailable returns true, parameter information is available from the server. If DescriptionsAvailable returns false, parameter information is not available.

Note: If DescriptionsAvailable returns false, the application must still provide correct parameter information to the stored procedure on the server. It is up to the application developer to know the necessary parameters at design time.

See Also

ExecProc (see page 10156)

GetResults (see page 10157)

Params (see page 10165)

53.1.18.4.5 TStoredProc.Disconnect Method

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

This is Disconnect, a member of class TStoredProc.

53.1.18.4.6 TStoredProc.ExecProc Method

Executes the stored procedure on the server.

Pascal

```
procedure ExecProc;
```

C++

```
__fastcall ExecProc();
```

Description

Call ExecProc to execute a stored procedure on the server. Before calling ExecProc:

1. Provide any input parameters in the Params (see page 10165) property. At design time, a developer can provide parameters using the Parameters editor. At runtime an application must access Params (see page 10165) directly.
2. Call Prepare (see page 10157) to bind the parameters.
3. If a stored procedure returns output parameters, they are stored in the Params (see page 10165) property when ExecProc returns control to the application. An application can access the output parameters by indexing into the Params (see page 10165) list, or by using the ParamByName (see page 10157) method.

Tip: If an application is only interested in the result set returned by a stored procedure, call its Open method or set its Active (see page 9929) property to true.

See Also

Params (see page 10165)

ParamByName (see page 10157)

Prepare (see page 10157)

Executing the Commands using TSQLDataSet

53.1.18.4.7 TStoredProc.GetParamsCount Method

Pascal

```
function GetParamsCount: Word;
```

C++

```
__fastcall Word GetParamsCount();
```

Description

This is GetParamsCount, a member of class TStoredProc.

53.1.18.4.8 TStoredProc.GetResults Method

Returns the output parameter values from a Sybase or MSSQL stored procedure.

Pascal

```
procedure GetResults;
```

C++

```
__fastcall GetResults();
```

Description

Call GetResults to force a Sybase or MSSQL stored procedure to return its result set, if any, to the client. For stored procedures on all other servers, result sets are automatically flushed to the client, but Sybase and MSSQL stored procedures do not return a result set until the cursor is positioned at the end of the set. GetResults forces the cursor to the end of the set.

If a stored procedure returns a result set and output parameters in response to GetResults, the output parameters are stored in the Params (see page 10165) property. An application can access the output parameters by indexing into the Params (see page 10165) list, or by using the ParamByName (see page 10157) method.

Tip: If an application is only interested in the result set returned by a stored procedure, call Open or set the Active (see page 9929) property to true.

See Also

ExecProc (see page 10156)

Params (see page 10165)

ParamByName (see page 10157)

53.1.18.4.9 TStoredProc.ParamByName Method

Accesses parameter information based on a specified parameter name.

Pascal

```
function ParamByName(const Value: string): TParam;
```

C++

```
__fastcall TParam ParamByName(const AnsiString Value);
```

Description

Call ParamByName to return parameter information for a specific parameter based on its name. Value is the name of the parameter for which to retrieve information. Typically ParamByName is used to set an input parameter's value at runtime, or to retrieve the value of an output parameter. The following command line assigns the value "Jane Smith" as the value for the parameter named Contact:

```
StoredProc1.ParamByName('Contact').AsString := 'Jane Smith';
StoredProc1->ParamByName("Contact")->AsString = "Jane Smith";
```

See Also

Params (see page 10165)

53.1.18.4.10 TStoredProc.Prepare Method

Prepares a stored procedure for execution.

Pascal

```
procedure Prepare;
```

C++

```
__fastcall Prepare();
```

Description

Call Prepare to bind a stored procedure's parameters before calling ExecProc (see page 10156) to execute the procedure. Prepare readies a stored procedure's parameters, initializes the Borland (see page 1761) Database (see page 10021) Engine (BDE), and informs the server of the stored procedure's readiness. These steps allocate system resources and optimize the query for server performance.

Note: If an application attempts to execute a stored procedure that has not been prepared, the library automatically prepares the procedure before executing it, and then unprepares it when execution is complete. If a stored procedure will be executed more than once, it is more efficient for an application to call Prepare explicitly once to avoid repeated and unnecessary preparing and unpreparing of the stored procedure, and then call UnPrepare (see page 10162) when the stored procedure is no longer needed.

See Also

ExecProc (see page 10156)

Prepared (see page 10165)

UnPrepare (see page 10162)

53.1.18.4.11 TStoredProc.PSExecute Method

Executes the SQL command associated with the dataset.

Pascal

```
procedure PSExecute; override;
```

C++

```
virtual __fastcall PSExecute();
```

Description

The provider component calls PSExecute to implement its Execute method. If the dataset represents a query or stored procedure that does not return a cursor, this method executes the query or stored procedure. If the provider Options includes poAllowCommandText and the dataset supports the PSSetCommandText method, the statement specified by PSSetCommandText may be executed instead.

If there is not statement to execute, PSExecute raises an EDatabaseError exception.

See Also

PSSetCommandText (see page 7585)

Execute (see page 18466)

Options (see page 18440)

53.1.18.4.12 TStoredProc.PSGetCommandText Method

Gets the SQL command to be executed by PSExecute.

Pascal

```
function PSGetCommandText: string; override;
```

C++

```
virtual __fastcall AnsiString PSGetCommandText();
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

PSSetCommandText (see page 7585)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

53.1.18.4.13 TStoredProc.PSGetCommandType Method

Gets the command type to be executed by PSExecute.

Pascal

```
function PSGetCommandType: TPSCommandType; override;
```

C++

```
virtual __fastcall TPSCommandType PSGetCommandType();
```

Description

PSGetCommandType gets the command type executed by PSExecute. The type may be a type of SQL command or other command types, such as stored procedures.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

PSGetCommandText (see page 7579)

Execute (see page 18466)

Options (see page 18440)

53.1.18.4.14 TStoredProc.PSGetParams Method

Returns the current parameter values of the dataset.

Pascal

```
function PSGetParams: TParams; override;
```

C++

```
virtual __fastcall TParams PSGetParams();
```

Description

The provider component calls PSGetParams to implement its GetParams method.

PSGetParams returns a TParams object that describes the current values of all parameters of the dataset. If the dataset does not have any parameters, PSGetParams returns nil (Delphi) or NULL (C++).

Note: The PSGetParams method should not create a new instance of TParams for the return value. The provider that calls PSGetParams does not free the returned value.

See Also

GetParams (see page 18467)

53.1.18.4.15 TStoredProc.PSGetTableName Method

Returns the name of the dataset table that appears in generated SQL statements.

Pascal

```
function PSGetTableName: string; override;
```

C++

```
virtual __fastcall AnsiString PSGetTableName();
```

Description

PSGetTableName returns the name of the underlying database table against which updates are applied when the provider's ResolveToDataSet property is false. This value is used for generating the SQL statements that apply updates. The generated SQL statements are subsequently executed by a call to PSExecuteStatement.

If PSGetTableName does not return a table name, the provider must have an OnGetTableName event handler to supply this value.

See Also

applyupdates (see page 18459)

PSExecuteStatement (see page 7579)

ResolveToDataSet (see page 18531)

ongettablename (see page 18542)

53.1.18.4.16 TStoredProc.PSSetCommandText Method

Assigns a specified SQL command to be executed by PSExecute.

Pascal

```
procedure PSSetCommandText(const CommandText: string); override;
```

C++

```
virtual __fastcall PSSetCommandText(const AnsiString CommandText);
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed

by a call to PSExecute.

The value of the CommandText parameter replaces the dataset's SQL statement (if it has one), or replaces the name of an underlying database table or stored procedure.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

53.1.18.4.17 TStoredProc.PSSetParams Method

Assigns the specified parameter values to the dataset.

Pascal

```
procedure PSSetParams(AParams: TParams); override;
```

C++

```
virtual __fastcall PSSetParams(TParams AParams);
```

Description

PSSetParams assigns the parameter values specified by AParams to the parameters of the dataset.

See Also

Params (see page 8489)

PSGetParams (see page 7582)

53.1.18.4.18 TStoredProc.SetDBFlag Method

Pascal

```
function SetDBFlag(Flag: Integer; Value: Boolean): Boolean; override;
```

C++

```
virtual __fastcall Boolean SetDBFlag(int Flag, Boolean Value);
```

Description

This is SetDBFlag, a member of class TStoredProc.

53.1.18.4.19 TStoredProc.SetOverLoad Method

Pascal

```
procedure SetOverLoad(Value: Word);
```

C++

```
__fastcall SetOverLoad(Word Value);
```

Description

This is SetOverLoad, a member of class TStoredProc.

53.1.18.4.20 TStoredProc.SetPrepare Method

Pascal

```
procedure SetPrepare(Value: Boolean);
```

C++

```
__fastcall SetPrepare(Boolean Value);
```

Description

This is SetPrepare, a member of class TStoredProc.

53.1.18.4.21 TStoredProc.SetPrepared Method

Pascal

```
procedure SetPrepared(Value: Boolean);
```

C++

```
__fastcall SetPrepared(Boolean Value);
```

Description

This is SetPrepared, a member of class TStoredProc.

53.1.18.4.22 TStoredProc.SetProcName Method

Pascal

```
procedure SetProcName(const Value: string);
```

C++

```
__fastcall SetProcName(const AnsiString Value);
```

Description

This is SetProcName, a member of class TStoredProc.

53.1.18.4.23 TStoredProc.UnPrepare Method

Frees the resources allocated for a previously prepared stored procedure.

Pascal

```
procedure UnPrepare;
```

C++

```
__fastcall UnPrepare();
```

Description

Call UnPrepare to free the resources allocated for a previously prepared stored procedure on the server and client sides.

See Also

ExecProc (see page 10156)

Prepare (see page 10157)

53.1.18.5 TStoredProc Properties

The properties of the TStoredProc class are listed here.

Public Properties

	Name	Description
R	Handle (see page 10163)	Specifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) cursor handle for the stored procedure.
R	ParamCount (see page 10165)	Indicates the number of parameters for the stored procedure component.
	Prepared (see page 10165)	Determines whether or not a stored procedure is prepared for execution.
	StmtHandle (see page 10166)	Specifies the Borland Database Engine (BDE) statement handle for the dataset.

Published Properties

	Name	Description
	Overload (see page 10163)	Specifies which Oracle overloaded stored procedure to execute.
	ParamBindMode (see page 10164)	Determines the order in which a component's parameters are assigned to the parameter list for the stored procedure on the server.
	Params (see page 10165)	Stores the input and output parameters for a stored procedure.
	StoredProcName (see page 10167)	Identifies the name of the stored procedure on the server for which this object is an encapsulation.
	UpdateObject (see page 10167)	Specifies the update object component used to update a read-only result set when cached updates are enabled.

53.1.18.5.1 TStoredProc.Handle Property

Specifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) cursor handle for the stored procedure.

Pascal

```
property Handle: HDBICur;
```

C++

```
__property HDBICur Handle;
```

Description

Use Handle only to bypass TStoredProc (see page 10139) methods and call directly into the BDE. Many BDE function calls require a cursor handle parameter. Handle is assigned an initial value when a stored procedure is executed. If used with a BDE call that changes the current record position, call Resync immediately after returning from the BDE call.

Note: Do not use this property unless an application requires BDE functionality not available through built-in components.

See Also

Locale (see page 9934)

Resync (see page 6994)

53.1.18.5.2 TStoredProc.Overload Property

Specifies which Oracle overloaded stored procedure to execute.

Pascal

```
property Overload: Word;
```

C++

```
__property Word Overload;
```

Description

Use Overload to specify which overloaded stored procedure to execute on an Oracle server. An Oracle overloaded stored procedure is one that shares a name with one or more other stored procedures. Oracle distinguishes among overloaded stored procedures by assigning each procedure a unique numeric identifier. An application can specify this identifier using the Overload property.

By default, Overload is zero, which assumes there is no procedure overloading. For all servers except Oracle, do not change this value. If an application attempts to access an Oracle overloaded procedure without setting Overload, the Borland (see page 1761) Database (see page 10021) Engine (BDE) accesses the lowest numbered, or first, overloaded procedure on the server.

If Overload is 1, the BDE executes the first overloaded procedure on the server. If Overload is 2, it executes the second, and so on.

Note: While Oracle overloaded procedures share names, their parameter lists are unique. An application must ensure that it passes the correct parameter list to an overloaded procedure.

See Also

Params (see page 10165)

Working with Oracle Overloaded Stored Procedures

53.1.18.5.3 TStoredProc.ParamBindMode Property

Determines the order in which a component's parameters are assigned to the parameter list for the stored procedure on the server.

Pascal

```
property ParamBindMode: TParamBindMode;
```

C++

```
__property TParamBindMode ParamBindMode;
```

Description

Examine or set ParamBindMode to determine the order in which parameters in the Params (see page 10165) property are matched to the parameters used by the stored procedure on the server. ParamBindMode can be one of the following:

Value	Ordering
pbByName	Parameters specified in the Params (see page 10165) property are matched to identically named parameters on the server. This is the default.
pbByNumber	Parameters in Params (see page 10165) are assigned one-by-one to the next available parameter on the server (for example, the first parameter in Params (see page 10165) is assigned to the first parameter used by the stored procedure, and so on).

Whenever possible, ParamBindMode should be pbByName. This guarantees that parameters are matched to the correct parameters used by the stored procedure regardless of physical ordering in Params (see page 10165). At design time, the names of known parameters appear in the Parameters editor.

In some cases stored procedures on the server (for example, Informix) do not provide the Borland (see page 1761) Database (see page 10021) Engine (BDE) with parameter names. If parameter names as they exist on the server side are not known at design or runtime, it may be necessary to set ParamBindMode to pbByNumber. In this case, however, it is necessary to know the correct data type for each parameter in sequence. Otherwise an exception may be raised on execution.

See Also

ExecProc (see page 10156)

ParamCount (see page 10165)

Params (see page 10165)

Binding Parameters

Using TStoredProc

53.1.18.5.4 TStoredProc.ParamCount Property

Indicates the number of parameters for the stored procedure component.

Pascal

```
property ParamCount: Word;
```

C++

```
__property Word ParamCount;
```

Description

Examine ParamCount to determine the number of parameters currently stored in the Params (see page 10165) property.

See Also

ExecProc (see page 10156)

Params (see page 10165)

Prepare (see page 10157)

53.1.18.5.5 TStoredProc.Params Property

Stores the input and output parameters for a stored procedure.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Access Params at runtime to set input parameter names, values, and data types dynamically (at design time use the Parameters editor to set parameter information). Params is an array of parameter values.

An application can also access Params after executing a stored procedure to retrieve the output parameters returned to the procedure by the server.

See Also

ExecProc (see page 10156)

ParamByName (see page 10157)

Prepare (see page 10157)

53.1.18.5.6 TStoredProc.Prepared Property

Determines whether or not a stored procedure is prepared for execution.

Pascal

```
property Prepared: Boolean;
```

C++

```
__property Boolean Prepared;
```

Description

Examine Prepared to determine if a stored procedure is already prepared for execution. If Prepared is true, the stored procedure is prepared, and if Prepared is false, the procedure is not prepared. A stored procedure must be prepared before it can be executed.

Note: The library automatically prepares a stored procedure if it is unprepared when the application calls ExecProc (see page 10156) and unprepares it afterwards. If a procedure will be executed a number of times, it is more efficient for the application to prepare the stored procedure once, and unprepare it when it is no longer needed.

An application can change the current setting of Prepared to prepare or unprepare a stored procedure. If Prepared is true, setting it to false calls the Unprepare method to unprepare the stored procedure. If Prepared is false, setting it to true calls the Prepare (see page 10157) method to prepare the procedure. Generally, however, it is better programming practice to call Prepare (see page 10157) and Unprepare directly. These methods automatically update the Prepared property.

See Also

ExecProc (see page 10156)

Prepare (see page 10157)

UnPrepare (see page 10162)

Executing the Commands using TSQLDataSet

Fetching the Data using TSQLDataSet

53.1.18.5.7 TStoredProc.StmtHandle Property

Specifies the Borland Database Engine (BDE) statement handle for the dataset.

Pascal

```
property StmtHandle: HDBIStmt;
```

C++

```
__property HDBIStmt StmtHandle;
```

Description

Use StmtHandle for direct calls to the BDE that bypass the methods of this dataset. Some BDE API calls require a statement handle as a parameter. For example, TBDEDataSet descendants use StmtHandle when preparing a query or binding parameters.

Typically, applications do not need to access this property. In TBDEDataSet, StmtHandle is protected, and so unavailable to applications. Some descendants make this property public so that you can bypass the dataset methods and make direct calls to the BDE.

See Also

Handle (see page 9933)

Prepare (see page 10077)

Prepare (see page 10157)

Params (see page 10087)

Params (see page 10165)

53.1.18.5.8 TStoredProc.StoredProcName Property

Identifies the name of the stored procedure on the server for which this object is an encapsulation.

Pascal

```
property StoredProcName: string;
```

C++

```
__property AnsiString StoredProcName;
```

Description

Set StoredProcName to specify the name of the stored procedure to call on the server. If StoredProcName does not match the name of an existing stored procedure on the server, then when the application attempts to prepare the procedure prior to execution, an exception is raised.

See Also

Prepare (see page 10157)

ExecProc (see page 10156)

Using TStoredProc

53.1.18.5.9 TStoredProc.UpdateObject Property

Specifies the update object component used to update a read-only result set when cached updates are enabled.

Pascal

```
property UpdateObject: TDataSetUpdateObject;
```

C++

```
__property TDataSetUpdateObject UpdateObject;
```

Description

Use UpdateObject to specify the TUpdateObject component to use in an application that must be able to update a read-only result set returned by the Borland Database Engine (BDE).

The BDE always attempts to provide an updatable, or "live" query result unless an application specifically requests a read-only view of data. In some cases, such as a query made against multiple tables, a live result set cannot be returned. In these cases, UpdateObject can be used to specify a TUpdateSQL component that performs updates as a separate transaction that is transparent to the application.

See Also

CachedUpdates (see page 9932)

ApplyUpdates (see page 9894)

TUpdateSQL (see page 10229)

Using the BDE to Cache Updates

Using Update Objects to Update a Dataset

Updating a Read-only Result Set

Using TQuery

Using TUpdateSQL to Update a Dataset

53.1.19 TTable Class

TTable encapsulates a database table.

Class Hierarchy

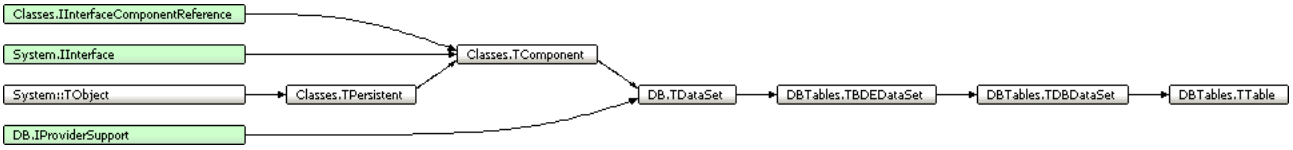

Pascal

```
TTable = class(TDBDataSet);
```

C++

```
class TTable : public TDBDataSet;
```

File

DBTables

Description

Use TTable to access data in a single database table using the Borland (see page 1761) Database (see page 10021) Engine (BDE). TTable provides direct access to every record and field in an underlying database table, whether it is from Paradox, dBASE, Access, FoxPro, an ODBC-compliant database, or an SQL database on a remote server, such as InterBase, Oracle, Sybase, MS-SQL Server, Informix, or DB2. A table component can also work with a subset of records within a database table using ranges and filters.

At design time, create, delete, update, or rename the database table connected to a TTable by right-clicking on the TTable and using the pop-up menu.

See Also

TQuery (see page 10059)

TStoredProc (see page 10139)

TClientDataSet (see page 8289)

TADODataSet (see page 1041)

TADOTable (see page 1125)

Creating Decision Datasets with TQuery or TTable

Modifying Entire Records

Opening and Closing Datasets

Types of Datasets

53.1.19.1 TTable Members

The following tables list the members exposed by TTable.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	ActivateFilters (see page 9893)	This is ActivateFilters, a member of class TBDEDataSet.
	AddFieldDesc (see page 9893)	This is AddFieldDesc, a member of class TBDEDataSet.
	AllocCachedUpdateBuffers (see page 9893)	This is AllocCachedUpdateBuffers, a member of class TBDEDataSet.
	AllocKeyBuffers (see page 9893)	This is AllocKeyBuffers, a member of class TBDEDataSet.
V	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
V	AllocRecordBuffer (see page 9894)	This is AllocRecordBuffer, a member of class TBDEDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
V	BlockReadNext (see page 9894)	This is BlockReadNext, a member of class TBDEDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
	CachedUpdateCallBack (see page 9895)	This is CachedUpdateCallBack, a member of class TBDEDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
	CheckCachedUpdateMode (see page 9896)	This is CheckCachedUpdateMode, a member of class TBDEDataSet.
V	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
V	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
	CheckSetKeyMode (see page 9896)	This is CheckSetKeyMode, a member of class TBDEDataSet.
V	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
V	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
V	ClearCalcFields (see page 9897)	This is ClearCalcFields, a member of class TBDEDataSet.
V	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
V	CloseBlob (see page 9897)	This is CloseBlob, a member of class TBDEDataSet.
V	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
V	CloseCursor (see page 9897)	This is CloseCursor, a member of class TBDEDataSet.
V	CloseCursor (see page 10014)	This is CloseCursor, a member of class TDBDataSet.
	ConstraintsStored (see page 10015)	This is ConstraintsStored, a member of class TDBDataSet.
	CreateExprFilter (see page 9900)	This is CreateExprFilter, a member of class TBDEDataSet.
V	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
	CreateFuncFilter (see page 9900)	This is CreateFuncFilter, a member of class TBDEDataSet.
V	CreateHandle (see page 10189)	This is CreateHandle, a member of class TTable.
	CreateLookupFilter (see page 9901)	This is CreateLookupFilter, a member of class TBDEDataSet.
V	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
V	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
V	DataEvent (see page 10191)	This is DataEvent, a member of class TTable.
	DeactivateFilters (see page 9901)	This is DeactivateFilters, a member of class TBDEDataSet.
V	DefChanged (see page 10191)	This is DefChanged, a member of class TTable.

	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
	DestroyHandle (see page 10193)	This is DestroyHandle, a member of class TTable.
	DestroyLookupCursor (see page 10193)	This is DestroyLookupCursor, a member of class TTable.
	Disconnect (see page 10015)	This is Disconnect, a member of class TDBDataSet.
	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 10194)	This is DoOnNewRecord, a member of class TTable.
	EncodeFieldDesc (see page 10196)	This is EncodeFieldDesc, a member of class TTable.
	EncodeIndexDesc (see page 10196)	This is EncodeIndexDesc, a member of class TTable.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FindRecord (see page 9903)	This is FindRecord, a member of class TBDEDataSet.
	ForceUpdateCallback (see page 9904)	This is ForceUpdateCallback, a member of class TBDEDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeKeyBuffers (see page 9904)	This is FreeKeyBuffers, a member of class TBDEDataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
	FreeRecordBuffer (see page 9904)	This is FreeRecordBuffer, a member of class TBDEDataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
	GetBookmarkData (see page 9905)	This is GetBookmarkData, a member of class TBDEDataSet.
	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
	GetBookmarkFlag (see page 9905)	This is GetBookmarkFlag, a member of class TBDEDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 10198)	This is GetCanModify, a member of class TTable.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
	GetDataSource (see page 10199)	This is GetDataSource, a member of class TTable.
	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.

V	GetFieldFullName (see page 9907)	This is GetFieldFullName, a member of class TBDEDataSet.
	GetFileName (see page 10199)	This is GetFileName, a member of class TTable.
	GetHandle (see page 10199)	This is GetHandle, a member of class TTable.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIndexField (see page 9907)	This is GetIndexField, a member of class TBDEDataSet.
	GetIndexFieldCount (see page 9907)	This is GetIndexFieldCount, a member of class TBDEDataSet.
V	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
V	GetIsIndexField (see page 9908)	This is GetIsIndexField, a member of class TBDEDataSet.
	GetKeyBuffer (see page 9908)	This is GetKeyBuffer, a member of class TBDEDataSet.
	GetKeyExclusive (see page 9908)	This is GetKeyExclusive, a member of class TBDEDataSet.
	GetKeyFieldCount (see page 9909)	This is GetKeyFieldCount, a member of class TBDEDataSet.
	GetLanguageDriverName (see page 10200)	This is GetLanguageDriverName, a member of class TTable.
V	GetLookupCursor (see page 10200)	This is GetLookupCursor, a member of class TTable.
V	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
V	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetObjectTypeNames (see page 9909)	This is GetObjectTypeNames, a member of class TBDEDataSet.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
V	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
V	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
V	GetRecNo (see page 9909)	This is GetRecNo, a member of class TBDEDataSet.
A	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
V	GetRecord (see page 9910)	This is GetRecord, a member of class TBDEDataSet.
V	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
V	GetRecordCount (see page 9910)	This is GetRecordCount, a member of class TBDEDataSet.
V	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
V	GetRecordSize (see page 9910)	This is GetRecordSize, a member of class TBDEDataSet.
V	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
V	GetStateFieldValue (see page 9910)	This is GetStateFieldValue, a member of class TBDEDataSet.
	GetTableType (see page 10200)	This is GetTableType, a member of class TTable.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetUpdateRecordSet (see page 9910)	This is GetUpdateRecordSet, a member of class TBDEDataSet.
	GetUpdatesPending (see page 9911)	This is GetUpdatesPending, a member of class TBDEDataSet.
V	InitFieldDefs (see page 10203)	This is InitFieldDefs, a member of class TTable.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
	InitKeyBuffer (see page 9911)	This is InitKeyBuffer, a member of class TBDEDataSet.
V	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
V	InitRecord (see page 9911)	This is InitRecord, a member of class TBDEDataSet.
V	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
V	InternalAddRecord (see page 9911)	This is InternalAddRecord, a member of class TBDEDataSet.
V	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
V	InternalCancel (see page 9912)	This is InternalCancel, a member of class TBDEDataSet.
A	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.
V	InternalClose (see page 9912)	This is InternalClose, a member of class TBDEDataSet.
V	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
V	InternalDelete (see page 9912)	This is InternalDelete, a member of class TBDEDataSet.
V	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
V	InternalEdit (see page 9912)	This is InternalEdit, a member of class TBDEDataSet.
V	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
V	InternalFirst (see page 9912)	This is InternalFirst, a member of class TBDEDataSet.

	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
	InternalGotoBookmark (see page 9913)	This is InternalGotoBookmark, a member of class TBDEDataSet.
	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
	InternalHandleException (see page 9913)	This is InternalHandleException, a member of class TBDEDataSet.
	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
	InternalInitFieldDefs (see page 9913)	This is InternalInitFieldDefs, a member of class TBDEDataSet.
	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
	InternalInitRecord (see page 9913)	This is InternalInitRecord, a member of class TBDEDataSet.
	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
	InternalInsert (see page 9914)	This is InternalInsert, a member of class TBDEDataSet.
	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
	InternalLast (see page 9914)	This is InternalLast, a member of class TBDEDataSet.
	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
	InternalOpen (see page 9914)	This is InternalOpen, a member of class TBDEDataSet.
	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
	InternalPost (see page 9914)	This is InternalPost, a member of class TBDEDataSet.
	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
	InternalRefresh (see page 9914)	This is InternalRefresh, a member of class TBDEDataSet.
	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	InternalSetToRecord (see page 9915)	This is InternalSetToRecord, a member of class TBDEDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
	IsCursorOpen (see page 9915)	This is IsCursorOpen, a member of class TBDEDataSet.
	IsProductionIndex (see page 10203)	This is IsProductionIndex, a member of class TTable.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 9916)	This is LocateRecord, a member of class TBDEDataSet.
	MapsToIndex (see page 9918)	This is MapsToIndex, a member of class TBDEDataSet.
	NativeTableName (see page 10203)	This is NativeTableName, a member of class TTable.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	OpenCursor (see page 6976)	This is OpenCursor, a member of class TDataSet.
	OpenCursor (see page 9918)	This is OpenCursor, a member of class TBDEDataSet.
	OpenCursor (see page 10015)	This is OpenCursor, a member of class TDBDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PostKeyBuffer (see page 9919)	This is PostKeyBuffer, a member of class TBDEDataSet.
	PrepareCursor (see page 10204)	This is PrepareCursor, a member of class TTable.
	ProcessUpdates (see page 9919)	This is ProcessUpdates, a member of class TBDEDataSet.
	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
	PSEndTransaction (see page 10016)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 10017)	Executes a specified SQL command.
	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 10017)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 10204)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandType (see page 10205)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 10205)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 10205)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 10206)	Returns the names of all fields required to uniquely identify records in the data packet.

	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 10018)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetTableName (see page 10206)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSGetUpdateException (see page 9920)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 10018)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 10018)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 9920)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 9921)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 10207)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 10207)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSStartTransaction (see page 10019)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 9921)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 10019)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	ResetCursorRange (see page 9922)	This is ResetCursorRange, a member of class TBDEDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBlockReadSize (see page 9922)	This is SetBlockReadSize, a member of class TBDEDataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkData (see page 9922)	This is SetBookmarkData, a member of class TBDEDataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkFlag (see page 9923)	This is SetBookmarkFlag, a member of class TBDEDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetCachedUpdates (see page 9923)	This is SetCachedUpdates, a member of class TBDEDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetCursorRange (see page 9923)	This is SetCursorRange, a member of class TBDEDataSet.
	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDBFlag (see page 10020)	This is SetDBFlag, a member of class TDBDataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldData (see page 9923)	This is SetFieldData, a member of class TBDEDataSet.

	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
	SetFilterData (see page 9923)	This is SetFilterData, a member of class TBDEDataSet.
	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
	SetFiltered (see page 9924)	This is SetFiltered, a member of class TBDEDataSet.
	SetFilterHandle (see page 9924)	This is SetFilterHandle, a member of class TBDEDataSet.
	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
	SetFilterOptions (see page 9924)	This is SetFilterOptions, a member of class TBDEDataSet.
	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
	SetFilterText (see page 9924)	This is SetFilterText, a member of class TBDEDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetHandle (see page 10020)	This is SetHandle, a member of class TDBDataSet.
	SetIndexField (see page 9925)	This is SetIndexField, a member of class TBDEDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetKeyBuffer (see page 9925)	This is SetKeyBuffer, a member of class TBDEDataSet.
	SetKeyExclusive (see page 9925)	This is SetKeyExclusive, a member of class TBDEDataSet.
	SetKeyFieldCount (see page 9925)	This is SetKeyFieldCount, a member of class TBDEDataSet.
	SetKeyFields (see page 9925)	This is SetKeyFields, a member of class TBDEDataSet.
	SetLinkRanges (see page 9926)	This is SetLinkRanges, a member of class TBDEDataSet.
	SetLocale (see page 9926)	This is SetLocale, a member of class TBDEDataSet.
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetOnFilterRecord (see page 9926)	This is SetOnFilterRecord, a member of class TBDEDataSet.
	SetOnUpdateError (see page 9926)	This is SetOnUpdateError, a member of class TBDEDataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
	SetRecNo (see page 9927)	This is SetRecNo, a member of class TBDEDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
	SetStateFieldValue (see page 9927)	This is SetStateFieldValue, a member of class TBDEDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	SetupCallBack (see page 9927)	This is SetupCallBack, a member of class TBDEDataSet.
	SetUpdateObject (see page 9927)	This is SetUpdateObject, a member of class TBDEDataSet.
	SetUpdateRecordSet (see page 9927)	This is SetUpdateRecordSet, a member of class TBDEDataSet.
	SwitchToIndex (see page 9928)	This is SwitchToIndex, a member of class TBDEDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	UpdateCallbackRequired (see page 9928)	This is UpdateCallbackRequired, a member of class TBDEDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateIndexDefs (see page 10213)	This is UpdateIndexDefs, a member of class TTable.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
R	DBFlags (see page 10023)	Keeps track of the commands this dataset has active with the database.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
R	MasterLink (see page 10224)	This is MasterLink, a member of class TTable.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
	Reserved (see page 7035)	For internal use only.
	StmtHandle (see page 9935)	Specifies the Borland (see page 1761) Database Engine (BDE) statement handle for the dataset.

Public Events

	Name	Description
	AfterCancel (see page 7038)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 7040)	Occurs after an application closes a dataset.
	AfterDelete (see page 7041)	Occurs after an application deletes a record.
	AfterEdit (see page 7044)	Occurs after an application starts editing a record.
	AfterInsert (see page 7044)	Occurs after an application inserts a new record.
	AfterOpen (see page 7045)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 7046)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 7047)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 7047)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 7048)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 7048)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 7049)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 7049)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 7050)	Occurs before an application enters insert mode.
	BeforeOpen (see page 7051)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 7052)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 7052)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 7053)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 7053)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 7054)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 7054)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 7056)	Occurs when an application inserts or appends a new dataset record.

	OnPostError (see page 7056)	Occurs when an application attempts to modify or insert a record and an exception is raised.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
	AddIndex (see page 10184)	Creates a new index for the table.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
	ApplyRange (see page 10185)	Applies a range to the dataset.
	ApplyUpdates (see page 9894)	Writes a dataset's pending cached updates to the database.
V	Assign (see page 2930)	Copies the contents of another, similar object.
	BatchMove (see page 10186)	Moves records from a dataset into this table.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
V	BookmarkValid (see page 9895)	Tests the validity of a specified bookmark.
V	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
V	Cancel (see page 9895)	Cancels modifications to the current record if those changes are not yet posted.
	CancelRange (see page 10186)	Removes any ranges currently in effect for the table.
	CancelUpdates (see page 9896)	Clears all pending cached updates from the cache and restores the dataset its prior state.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	CheckOpen (see page 10014)	Checks the result of a call to the Borland (see page 1761) Database (see page 10021) Engine (BDE).
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Close (see page 6920)	Closes a dataset.
	CloseDatabase (see page 10014)	Closes a database connection associated with the database session.
	CloseIndexFile (see page 10188)	Closes a specified dBASE non-production index file.
	CommitUpdates (see page 9897)	Clears the cached updates buffer.
V	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
V	CompareBookmarks (see page 9898)	Indicates the relationship between two bookmarks.
	ConstraintCallBack (see page 9898)	Specifies the callback function that allows the dataset to communicate with the BDE constraint layer.
	ConstraintsDisabled (see page 9899)	Indicates whether constraints are enabled for a dataset.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
V	Create (see page 10182)	Creates an instance of a table component.
V	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
V	CreateBlobStream (see page 9899)	Returns a TStream object for reading or writing the data in a specified blob field.
	CreateTable (see page 10189)	Builds a new table using new structure information.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.

	DeleteIndex (see page 10191)	Deletes a secondary index for the table.
	DeleteTable (see page 10192)	Deletes an existing database table.
V	Destroy (see page 10182)	Destroys the instance of a table.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableConstraints (see page 9902)	Disables application of server constraints on the dataset.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EditKey (see page 10194)	Enables modification of the search key buffer.
	EditRangeEnd (see page 10194)	Enables changing the ending value for an existing range.
	EditRangeStart (see page 10195)	Enables changing the starting value for an existing range.
	EmptyTable (see page 10196)	Deletes all records from the table.
	EnableConstraints (see page 9902)	Enables application of server constraints against dataset entries.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FetchAll (see page 9903)	Retrieves all records from the current cursor position to the end of the file and stores them locally.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindKey (see page 10196)	Searches for a record containing specified field values.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNearest (see page 10197)	Moves the cursor to the record that most closely matches a specified set of key values.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	FlushBuffers (see page 9903)	Posts all changes that have been written to the record buffer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
V	GetBlobFieldData (see page 9904)	Reads BLOB data into a buffer.
V	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
V	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
V	GetCurrentRecord (see page 9905)	Retrieves the current record into a buffer.
V	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
V	GetDetailLinkFields (see page 10199)	Lists the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
V	GetFieldData (see page 9906)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
V	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.

	GetIndexInfo (see page 9907)	Retrieves information about the current index into the index data fields of the dataset.
	GetIndexNames (see page 10200)	Retrieves a list of available indexes for a table.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	GotoCurrent (see page 10201)	Synchronizes the current record for this table with the current record of a specified table component.
	GotoKey (see page 10201)	Moves the cursor to a record specified by the current key.
	GotoNearest (see page 10202)	Moves the cursor to the record that most closely matches the current key.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
V	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
V	IsSequenced (see page 9915)	IsSequenced indicates whether sequence numbers are available for database records.
	Last (see page 6968)	Moves to the last record in the dataset.
V	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Locate (see page 9915)	Searches the dataset for a specified record and makes that record the current record.
	LockTable (see page 10203)	Locks a Paradox or dBASE table.
V	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
V	Lookup (see page 9916)	Retrieves field values from a record that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	Open (see page 6976)	Opens the dataset.
	OpenDatabase (see page 10016)	Opens the database that contains the dataset.
	OpenIndexFile (see page 10204)	Opens a specified dBASE non-production index file.
V	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
V	Post (see page 9918)	Writes a modified record to the database.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.

	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RenameTable (see page 10208)	Renames the Paradox or dBASE table associated with this table component.
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
	RevertRecord (see page 9922)	Restores the current record in the dataset to an unmodified state when cached updates are enabled.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetKey (see page 10208)	Enables setting of keys and ranges for a dataset prior to a search.
	SetRange (see page 10209)	Sets the starting and ending values of a range, and applies it.
	SetRangeEnd (see page 10211)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
	SetRangeStart (see page 10212)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
V	Translate (see page 9928)	Converts a data string between the Microsoft ANSI character set, and the local code page (OEM character set).
	UnlockTable (see page 10212)	Removes a previously applied lock on a Paradox or dBASE table.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.
V	UpdateStatus (see page 9928)	Reports the update status for the current record.

Public Properties

	Name	Description
	Active (see page 7004)	Specifies whether or not a dataset is open.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
	CacheBlobs (see page 9932)	Determines whether BLOB fields are cached in memory.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.

R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Database (see page 10021)	Identifies the database component for which this dataset represents one or more tables.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
R	DataSource (see page 7014)	Represents the data source of another dataset that supplies values to the dataset.
R	DBHandle (see page 10023)	Specifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) database handle for the dataset.
R	DBLocale (see page 10024)	Identifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) language driver for the dataset component.
R	DBSession (see page 10024)	Identifies the database session component associated with this dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	Exists (see page 10215)	Indicates whether the underlying database table exists.
R	ExpIndex (see page 9933)	Indicates whether a dataset is using a dBASE expression index.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
R	Handle (see page 9933)	Specifies the Borland (see page 1761) Database Engine (BDE) cursor handle for the dataset.
	Handle (see page 10027)	Specifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) cursor handle for the dataset.
R	IndexFieldCount (see page 10220)	Indicates the number of fields that comprise the current key.
	IndexFields (see page 10221)	Lists the fields of the current index.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
	KeyExclusive (see page 10223)	Specifies how the upper and lower boundaries for a range should be interpreted.
	KeyFieldCount (see page 10223)	Specifies the number of fields to use when conducting a partial key search on a multi-field key.
R	KeySize (see page 9934)	Specifies the size of the key for the current index of the dataset.
R	Locale (see page 9934)	Identifies the Borland (see page 1761) Database Engine (BDE) language driver for the dataset.
R	Modified (see page 7029)	Indicates whether the active record is modified.
	ObjectView (see page 7031)	Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.

	TableLevel (see page 10226)	Specifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) driver-dependent table level.
	UpdateRecordTypes (see page 9936)	Specifies the type of records visible in a dataset when cached updates are enabled.
R	UpdatesPending (see page 9937)	Indicates whether the cached updates buffer contains records that are not yet applied.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterCancel (see page 9938)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 9940)	Occurs after an application closes a dataset.
	AfterDelete (see page 9942)	Occurs after an application deletes a record.
	AfterEdit (see page 9944)	Occurs after an application starts editing a record.
	AfterInsert (see page 9945)	Occurs after an application inserts a new record.
	AfterOpen (see page 9946)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 9947)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 9947)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 9948)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 9948)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 9949)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 9949)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 9950)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 9950)	Occurs before an application enters insert mode.
	BeforeOpen (see page 9952)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 9952)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 9953)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 9953)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 9954)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 9955)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 9955)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 10028)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 9955)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 9956)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnUpdateError (see page 9956)	Occurs if an exception is generated when cached updates are applied to a database.
	OnUpdateRecord (see page 9957)	Occurs when cached updates are applied to a record.

Published Properties

	Name	Description
	Active (see page 9929)	Specifies whether or not a dataset is open.
	AutoCalcFields (see page 9931)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	AutoRefresh (see page 10021)	Specifies whether server-generated field values are refetched automatically.
	CachedUpdates (see page 9932)	Specifies whether cached updates are enabled for a dataset.
	Constraints (see page 10214)	Specifies record-level constraints that must be met when editing the data.
	DatabaseName (see page 10022)	Specifies the name of the database associated with this dataset.
	DefaultIndex (see page 10214)	Specifies if the data in the table should be ordered on a default index when opened.
	Exclusive (see page 10215)	Enables an application to gain sole access to a Paradox or dBASE table.

	FieldDefs (see page 10217)	Points to the list of field definitions for the dataset.
	Filter (see page 10025)	Specifies the text of the current filter for a dataset.
	Filtered (see page 10026)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 10026)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
	IndexDefs (see page 10219)	Contains information about the indexes for a table.
	IndexFieldNames (see page 10220)	Lists the columns to use as an index for a table.
	IndexFiles (see page 10221)	Specifies one or more dBASE index file to use for a dBASE table that uses non-production indexes.
	IndexName (see page 10222)	Identifies a secondary index for the table.
	MasterFields (see page 10224)	Specifies one or more fields in a master table to link with corresponding fields in this table in order to establish a master-detail relationship between the tables.
	MasterSource (see page 10224)	Specifies the name of the data source for a dataset to use as a master table in establishing a detail-master relationship between this table and another one.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectView (see page 9935)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
R	Ranged (see page 10225)	This is Ranged, a member of class TTable.
	ReadOnly (see page 10225)	Specifies whether a table is read-only for this application.
	SessionName (see page 10027)	Identifies the name of the session with which this dataset is associated.
	StoreDefs (see page 10226)	Indicates whether the table's field and index definitions persist with the data module or form.
	TableName (see page 10226)	Indicates the name of the database table that this component encapsulates.
	TableType (see page 10227)	Indicates the database table structure for the table that this component represents.
	Tag (see page 2808)	Stores an integer value as part of a component.
	UpdateMode (see page 10228)	Determines how the Borland Database Engine (BDE) finds records when updating to an SQL database.
	UpdateObject (see page 10228)	Specifies the update object component used to update a read-only result set when cached updates are enabled.

53.1.19.2 TTable.Create Constructor

Creates an instance of a table component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TTable(TComponent * AOwner);
```

Description

Call Create to instantiate a table declared in an application if it was not placed on a form or data module at design time. Create calls its inherited constructor, creates an empty index definitions list, creates an empty data link, and creates an empty list of index files.

See Also

TDataLink (see page 6876)

TIndexDefs (see page 7292)

53.1.19.3 TTable.Destroy Destructor

Destroys the instance of a table.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTable();
```

Description

Do not call Destroy directly. Instead, call Free, which verifies that the table is not nil before calling Destroy. Destroy frees the index files list for the table, frees its data link, frees its index definitions, and then calls its inherited destructor.

See Also

Create (see page 10182)

Free (see page 22300)

53.1.19.4 TTable Methods

The methods of the TTable class are listed here.

Protected Methods

	Name	Description
	CreateHandle (see page 10189)	This is CreateHandle, a member of class TTable.
	DataEvent (see page 10191)	This is DataEvent, a member of class TTable.
	DefChanged (see page 10191)	This is DefChanged, a member of class TTable.
	DestroyHandle (see page 10193)	This is DestroyHandle, a member of class TTable.
	DestroyLookupCursor (see page 10193)	This is DestroyLookupCursor, a member of class TTable.
	DoOnNewRecord (see page 10194)	This is DoOnNewRecord, a member of class TTable.
	EncodeFieldDesc (see page 10196)	This is EncodeFieldDesc, a member of class TTable.
	EncodeIndexDesc (see page 10196)	This is EncodeIndexDesc, a member of class TTable.
	GetCanModify (see page 10198)	This is GetCanModify, a member of class TTable.
	GetDataSource (see page 10199)	This is GetDataSource, a member of class TTable.
	GetFileName (see page 10199)	This is GetFileName, a member of class TTable.
	GetHandle (see page 10199)	This is GetHandle, a member of class TTable.
	GetLanguageDriverName (see page 10200)	This is GetLanguageDriverName, a member of class TTable.
	GetLookupCursor (see page 10200)	This is GetLookupCursor, a member of class TTable.
	GetTableType (see page 10200)	This is GetTableType, a member of class TTable.
	InitFieldDefs (see page 10203)	This is InitFieldDefs, a member of class TTable.
	IsProductionIndex (see page 10203)	This is IsProductionIndex, a member of class TTable.
	NativeTableName (see page 10203)	This is NativeTableName, a member of class TTable.
	PrepareCursor (see page 10204)	This is PrepareCursor, a member of class TTable.
	PSGetCommandText (see page 10204)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandType (see page 10205)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 10205)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 10205)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 10206)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetTableName (see page 10206)	Returns the name of the dataset table that appears in generated SQL statements.
	PSSetCommandText (see page 10207)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetParams (see page 10207)	Assigns the specified parameter values to the dataset.
	UpdateIndexDefs (see page 10213)	This is UpdateIndexDefs, a member of class TTable.

Public Methods

	Name	Description
	AddIndex (see page 10184)	Creates a new index for the table.
	ApplyRange (see page 10185)	Applies a range to the dataset.
	BatchMove (see page 10186)	Moves records from a dataset into this table.
	CancelRange (see page 10186)	Removes any ranges currently in effect for the table.
	CloseIndexFile (see page 10188)	Closes a specified dBASE non-production index file.
	CreateTable (see page 10189)	Builds a new table using new structure information.
	DeleteIndex (see page 10191)	Deletes a secondary index for the table.
	DeleteTable (see page 10192)	Deletes an existing database table.
	EditKey (see page 10194)	Enables modification of the search key buffer.
	EditRangeEnd (see page 10194)	Enables changing the ending value for an existing range.
	EditRangeStart (see page 10195)	Enables changing the starting value for an existing range.
	EmptyTable (see page 10196)	Deletes all records from the table.
	FindKey (see page 10196)	Searches for a record containing specified field values.
	FindNearest (see page 10197)	Moves the cursor to the record that most closely matches a specified set of key values.
V	GetDetailLinkFields (see page 10199)	Lists the field components that link this dataset as a detail of a master dataset.
	GetIndexNames (see page 10200)	Retrieves a list of available indexes for a table.
	GotoCurrent (see page 10201)	Synchronizes the current record for this table with the current record of a specified table component.
	GotoKey (see page 10201)	Moves the cursor to a record specified by the current key.
	GotoNearest (see page 10202)	Moves the cursor to the record that most closely matches the current key.
	LockTable (see page 10203)	Locks a Paradox or dBASE table.
	OpenIndexFile (see page 10204)	Opens a specified dBASE non-production index file.
	RenameTable (see page 10208)	Renames the Paradox or dBASE table associated with this table component.
	SetKey (see page 10208)	Enables setting of keys and ranges for a dataset prior to a search.
	SetRange (see page 10209)	Sets the starting and ending values of a range, and applies it.
	SetRangeEnd (see page 10211)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
	SetRangeStart (see page 10212)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
	UnlockTable (see page 10212)	Removes a previously applied lock on a Paradox or dBASE table.

53.1.19.4.1 TTable.AddIndex Method

Creates a new index for the table.

Pascal

```
procedure AddIndex(const Name: string; const Fields: string; Options: TIndexOptions; const
DescFields: string = '');
```

C++

```
__fastcall AddIndex(const AnsiString Name, const AnsiString Fields, TIndexOptions Options,
const AnsiString DescFields = '');
```

Description

Call AddIndex to create a new index for the already-existing table associated with a dataset component. The index created with this procedure is added to the database table underlying the dataset component.

Name is the name of the new index. Name must contain an index name that is valid for the particular database type used (naming rules vary from one database type to another).

Fields contains the names of the fields on which the new index will be based. If more than one field is used, separate the field names in the list with semicolons.

Options is a set of attributes for the index. The Options parameter may contain any one, multiple, or none of the TIndexOptions constants: ixPrimary, ixUnique, ixDescending, ixCaseInsensitive, and ixExpression. Not all database systems will necessarily support all of these options. Even within a particular table type, any given option may only be supported in certain versions of a table type. Consult the documentation for the particular database system used to determine which options are and are not supported. See the topic for the TIndexOptions type for definitions of individual constants and the implications of their use.

DescFields is a string containing a list of field names, separated by semicolons. The fields specified in DescFields are the fields in the new index for which the ordering will be descending. Fields in the index definition but not in the DescFields list use the default ascending ordering. It is possible that a single index can have fields using both ascending and descending ordering.

```
Table1.AddIndex('MostPaid', 'CustNo;SaleDate;AmountPaid', [ixCaseInsensitive],
'SaleDate;AmountPaid');
TIndexOptions Options;
Options << ixCaseInsensitive;
Table1->AddIndex("MostPaid", "CustNo;SaleDate;AmountPaid", Options, "SaleDate;AmountPaid");
```

Warning: Attempting to create an index using options that are not applicable to the table type causes AddIndex to raise an exception.

dBASE tables only support primary indexes and true unique indexes when the table level is 7 or higher (concurrent with Visual dBASE 7). dBASE tables do not support case-insensitive indexes at all. You must use the ixExpression constant when the index is based on multiple fields or uses dBASE Data Manipulation Language (DML) functions (that is, for expression indexes). Multi-field dBASE indexes cannot be created simply with a list of field names (separated by semicolons) in the Fields parameter of AddIndex. The ixExpression constant is only applicable to dBASE tables.

Paradox tables support the ixDescending option for secondary indexes if the table level is 7 or higher and ixUnique if the table level is 5 or higher. The ixDescending and ixCaseInsensitive constants are not applicable to primary indexes.

For other table types, see the vendor-supplied documentation for the particular database system for details on what index options are applicable.

See Also

DeleteIndex (see page 10191)

IndexDefs (see page 10219)

AddIndexDef (see page 7296)

IndexFiles (see page 10221)

IndexName (see page 10222)

53.1.19.4.2 TTable.ApplyRange Method

Applies a range to the dataset.

Pascal

```
procedure ApplyRange;
```

C++

```
__fastcall ApplyRange();
```

Description

Call ApplyRange to cause a range established with SetRangeStart (see page 10212) and SetRangeEnd (see page 10211), or EditRangeStart (see page 10195) and EditRangeEnd (see page 10194), to take effect. When a range is in effect, only those records that fall within the range are available to the application for viewing and editing.

After a call to ApplyRange, the cursor is left on the first record in the range.

See Also

CancelRange (see page 10186)

EditRangeEnd (see page 10194)

EditRangeStart (see page 10195)

SetRange (see page 10209)

SetRangeEnd (see page 10211)

SetRangeStart (see page 10212)

Applying or Canceling a Range

Modifying a Range

53.1.19.4.3 TTable.BatchMove Method

Moves records from a dataset into this table.

Pascal

```
function BatchMove(ASource: TBDEDataSet; AMode: TBatchMode): Longint;
```

C++

```
__fastcall Longint BatchMove(TBDEDataSet ASource, TBatchMode AMode);
```

Description

Call BatchMove to

Copy records from another table into this table.

Update records in this table that occur in another table.

Append records from another table to the end of this table.

Delete records in this table that occur in another table.

ASource is a dataset component containing the records to import or (if deleting) match. The AMode parameter indicates what operation to perform (copy, update, append, or delete). This table is the destination of the batch operation.

BatchMove returns the number of records operated on.

See Also

TBatchMove (see page 9858)

Importing Data from Another Table

Using TTable

53.1.19.4.4 TTable.CancelRange Method

Removes any ranges currently in effect for the table.

Pascal

```
procedure CancelRange;
```

C++

```
__fastcall CancelRange();
```

Description

Call CancelRange to remove a range currently applied to a table. Canceling a range reenables access to all records in the dataset.

Delphi Examples:

```
{
The following example sets a range for a dataset. The form
requires two edit boxes, a data source, a client dataset or
table, a db grid, and a button.  Make sure that the Table1
info in FormActivate agrees with the table used.  Also, the
caption for the button must be initialized to '&ApplyRange'.
}

procedure TForm1.FormActivate(Sender: TObject);
begin
  Table1.DatabaseName := 'DBDEMOS';
  Table1.TableName := 'CustInfo';
  Table1.Active := True;
  Table1.IndexName := '';
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  if Button2.Caption = '&Apply Range' then
    begin
      Table1.SetRange([Edit1.Text],[Edit2.Text]);
      Button2.Caption := '&Drop Range';
    end
  else
    begin
      Table1.CancelRange;
      Table1.Refresh;
      Button2.Caption := '&Apply Range';
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
    else
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
```

```
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.Active:= False;
  end;
```

See Also

ApplyRange (see page 10185)

EditRangeEnd (see page 10194)

EditRangeStart (see page 10195)

SetRange (see page 10209)

SetRangeEnd (see page 10211)

SetRangeStart (see page 10212)

Applying or Canceling a Range

53.1.19.4.5 TTable.CloseIndexFile Method

Closes a specified dBASE non-production index file.

Pascal

```
procedure CloseIndexFile(const IndexFileName: string);
```

C++

```
__fastcall CloseIndexFile(const AnsiString IndexFileName);
```

Description

Call CloseIndexFile to shut down a specified dBASE non-production index file. IndexFileName is the name of the file to close, and is either the name of an index file previously opened with a call to OpenIndexFile (see page 10204) or previously opened because the file's name is listed in the IndexFiles (see page 10221) property.

Once closed, non-production index files are no longer maintained. Subsequent changes made to the dataset are not reflected in the index.

See Also

IndexFiles (see page 10221)

OpenIndexFile (see page 10204)

53.1.19.4.6 TTable.CreateHandle Method

Pascal

```
function CreateHandle: HDBICur; override;
```

C++

```
virtual __fastcall HDBICur CreateHandle();
```

Description

This is CreateHandle, a member of class TTable.

53.1.19.4.7 TTable.CreateTable Method

Builds a new table using new structure information.

Pascal

```
procedure CreateTable;
```

C++

```
__fastcall CreateTable();
```

Description

Call CreateTable at runtime to create a table using this dataset's current definitions. If the table already exists, CreateTable overwrites the table's structure and data. To avoid overwriting an existing table, check the Exists (see page 10215) property before calling CreateTable.

If the FieldDefs (see page 10217) property contains values, these values are used to create field definitions. Otherwise the Fields property is used. One or both of these properties must contain values in order to create a database table.

If the IndexDefs (see page 10219) property contain values, these values are used to create indexes on the table.

Delphi Examples:

```
{ The following example shows how to create a table. }
procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Close;
  Table1.DeleteTable;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
```

```
    Table1:= TTable.Create(Form1);
    with Table1 do
    begin
      DatabaseName := 'DBDEMOS';
      TableType := ttParadox;
      TableName := 'CustInfo';
      Table1.Active := False;
      { Don't overwrite an existing table }
      if Table1.Exists then
        MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
      else
      begin
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
        { Next, describe the fields in the table }
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.Active:= True;
  end;
```

See Also

AddIndex (see page 10184)

DeleteIndex (see page 10191)

DeleteTable (see page 10192)

EmptyTable (see page 10196)

Exists (see page 10215)

FieldDefs (see page 7019)

Fields (see page 7023)

IndexDefs (see page 10219)

RenameTable (see page 10208)

53.1.19.4.8 TTable.DataEvent Method

Pascal

```
procedure DataEvent(Event: TDataEvent; Info: Longint); override;
```

C++

```
virtual __fastcall DataEvent(TDataEvent Event, Longint Info);
```

Description

This is DataEvent, a member of class TTable.

53.1.19.4.9 TTable.DefChanged Method

Pascal

```
procedure DefChanged(Sender: TObject); override;
```

C++

```
virtual __fastcall DefChanged(TObject * Sender);
```

Description

This is DefChanged, a member of class TTable.

53.1.19.4.10 TTable.DeleteIndex Method

Deletes a secondary index for the table.

Pascal

```
procedure DeleteIndex(const Name: string);
```

C++

```
__fastcall DeleteIndex(const AnsiString Name);
```

Description

Call DeleteIndex to remove a secondary index for a table. Name is the name of the index to delete. DeleteIndex cannot remove a primary index.

Note: To delete an index, an application must first open the table for exclusive access.

See Also

AddIndex (see page 10184)

Exclusive (see page 10215)

IndexDefs (see page 10219)

53.1.19.4.11 TTable.DeleteTable Method

Deletes an existing database table.

Pascal

```
procedure DeleteTable;
```

C++

```
__fastcall DeleteTable();
```

Description

Call DeleteTable to delete an existing database table associated with the table component through its DatabaseName (see page 10022) and TableName (see page 10226) properties. A table must be closed before it can be deleted.

Warning: Deleting a table erases any data the table contains and destroys the table's structure information.

Delphi Examples:

```
{ The following example shows how to create a table. }
procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Close;
  Table1.DeleteTable;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
    else
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
```

```
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.Active:= True;
  end;
```

See Also

Active (see page 7004)

Close (see page 6920)

CreateTable (see page 10189)

EmptyTable (see page 10196)

53.1.19.4.12 TTable.DestroyHandle Method

Pascal

```
procedure DestroyHandle; override;
```

C++

```
virtual __fastcall DestroyHandle();
```

Description

This is DestroyHandle, a member of class TTable.

53.1.19.4.13 TTable.DestroyLookupCursor Method

Pascal

```
procedure DestroyLookupCursor; override;
```

C++

```
virtual __fastcall DestroyLookupCursor();
```

Description

This is DestroyLookupCursor, a member of class TTable.

53.1.19.4.14 TTable.DoOnNewRecord Method

Pascal

```
procedure DoOnNewRecord; override;
```

C++

```
virtual __fastcall DoOnNewRecord();
```

Description

This is DoOnNewRecord, a member of class TTable.

53.1.19.4.15 TTable.EditKey Method

Enables modification of the search key buffer.

Pascal

```
procedure EditKey;
```

C++

```
__fastcall EditKey();
```

Description

Call EditKey to put the dataset in dsSetKey state while preserving the current contents of the current search key buffer. To determine current search keys, you can use the IndexFields (see page 10221) property to iterate over the fields used by the current index.

EditKey is especially useful when performing multiple searches where only one or two field values among many change between each search.

See Also

Fields (see page 7023)

FindKey (see page 10196)

GotoKey (see page 10201)

GotoNearest (see page 10202)

SetKey (see page 10208)

State (see page 7036)

53.1.19.4.16 TTable.EditRangeEnd Method

Enables changing the ending value for an existing range.

Pascal

```
procedure EditRangeEnd;
```

C++

```
__fastcall EditRangeEnd();
```

Description

Call EditRangeEnd to change the ending value for an existing range. To specify an end range value, call FieldByName after calling EditRangeEnd. After assigning a new ending value, call ApplyRange (see page 10185) to activate the modified range.

Note: With Paradox or dBASE tables, EditRangeEnd works only on indexed fields. With SQL databases, EditRangeEnd also works with any columns specified in the IndexFieldNames (see page 10220) property.

See Also

ApplyRange (see page 10185)

CancelRange (see page 10186)

EditRangeStart (see page 10195)

FieldByName (see page 6936)

IndexFieldNames (see page 10220)

KeyExclusive (see page 10223)

SetRange (see page 10209)

SetRangeEnd (see page 10211)

SetRangeStart (see page 10212)

Modifying a Range

53.1.19.4.17 TTable.EditRangeStart Method

Enables changing the starting value for an existing range.

Pascal

```
procedure EditRangeStart;
```

C++

```
__fastcall EditRangeStart();
```

Description

Call EditRangeStart to change the starting value for an existing range. To specify a start range value, call FieldByName after calling EditRangeStart. After assigning a new ending value, call ApplyRange (see page 10185) to activate the modified range.

Note: With Paradox or dBASE tables, EditRangeStart works only on indexed fields. With SQL databases, EditRangeStart also works with any columns specified in the IndexFieldNames (see page 10220) property.

See Also

ApplyRange (see page 10185)

CancelRange (see page 10186)

EditRangeEnd (see page 10194)

FieldByName (see page 6936)

IndexFieldNames (see page 10220)

SetRange (see page 10209)

SetRangeEnd (see page 10211)

SetRangeStart (see page 10212)

Modifying a Range

53.1.19.4.18 TTable.EmptyTable Method

Deletes all records from the table.

Pascal

```
procedure EmptyTable;
```

C++

```
__fastcall EmptyTable();
```

Description

The EmptyTable method deletes all records from the database table specified by the DatabaseName (see page 10022) and TableName (see page 10226) properties.

Note: Deletion of records can fail if the user lacks sufficient privileges to perform the delete operation.

See Also

DeleteTable (see page 10192)

53.1.19.4.19 TTable.EncodeFieldDesc Method

Pascal

```
procedure EncodeFieldDesc(var FieldDesc: BDEFLDDesc; const Name: string; DataType: TFieldType;
Size: Integer; Precision: Integer);
```

C++

```
__fastcall EncodeFieldDesc(BDEFLDDesc FieldDesc, const AnsiString Name, TFieldType DataType,
int Size, int Precision);
```

Description

This is EncodeFieldDesc, a member of class TTable.

53.1.19.4.20 TTable.EncodeIndexDesc Method

Pascal

```
procedure EncodeIndexDesc(var IndexDesc: IDXDesc; const Name: string; const FieldExpression:
string; Options: TIndexOptions; const DescFields: string = '');
```

C++

```
__fastcall EncodeIndexDesc(IDXDesc IndexDesc, const AnsiString Name, const AnsiString
FieldExpression, TIndexOptions Options, const AnsiString DescFields = '');
```

Description

This is EncodeIndexDesc, a member of class TTable.

53.1.19.4.21 TTable.FindKey Method

Searches for a record containing specified field values.

Pascal

```
function FindKey(const KeyValues: array of const): Boolean;
```

C++

```
__fastcall Boolean FindKey(const array of const KeyValues);
```

Description

Call FindKey to search for a specific record in a dataset. KeyValues contains a comma-delimited array of field values, called a key. Each value in the key can be a literal, a variable, nil (Delphi) or NULL (C++). If the number of values passed in KeyValues is less than the number of columns in the index used for the search, the missing values are assumed to be null. KeyValues_Size specifies the index of the last value in KeyValues (one less than the total number of values supplied). In the statement below, the FindKey method is used to find the table row with the value "'Princess Island SCUBA" in the index field.

```
Table1.FindKey(['Princess Island SCUBA']);
TVarRec vr = ("Princess Island SCUBA");
Table1->FindKey(&vr, 0);
```

Note: In C++ code, the search values can be passed to FindKey using either a TVarRec (as above) or an ARRAYOFCONST construct. The example below performs the same search as the preceding example, but using ARRAYOFCONST.

```
Table1->FindKey(ARRAYOFCONST(("Princess Island SCUBA")));
```

For Paradox and dBASE tables, the key must always be an index, which can be specified in the IndexName (see page 10222) property. If IndexName (see page 10222) is empty, FindKey uses the table's primary index. Note that FindKey does not work with dBASE expression indexes. Use the GotoKey (see page 10201) or GotoNearest (see page 10202) method instead.

For SQL tables, the key may correspond to a specified index in IndexName (see page 10222), or to a list of field names in the IndexFieldNames (see page 10220) property.

If the search is successful, FindKey positions the cursor on the matching record and returns true. Otherwise the cursor is not moved, and FindKey returns false.

See Also

FindNearest (see page 10197)

GotoKey (see page 10201)

GotoNearest (see page 10202)

IndexName (see page 10222)

IndexFieldNames (see page 10220)

KeyExclusive (see page 10223)

Locate (see page 9915)

Executing a Search with Find Methods

53.1.19.4.22 TTable.FindNearest Method

Moves the cursor to the record that most closely matches a specified set of key values.

Pascal

```
procedure FindNearest(const KeyValues: array of const);
```

C++

```
__fastcall FindNearest(const array of const KeyValues);
```

Description

Call FindNearest to move the cursor to a specific record in a dataset or to the first record in the dataset that is greater than the values specified in the KeyValues parameter. KeyValues contains a comma-delimited array of field values, called a key. Each value in the key can be a literal, a variable, or nil (Delphi) or NULL (C++). If the number of values passed in KeyValues is less than the number of columns in the index used for the search, the missing values are assumed to be null. KeyValues_Size specifies the index of the last value in KeyValues (one less than the total number of values supplied). In the statement below, the FindKey (see page 10196) method is used to find the table row with the value "'Princess Island SCUBA" in the index field.

```
Table1.FindNearest(['Princess Island SCUBA']);
TVarRec vr = ("Princess Island SCUBA");
Table1->FindNearest(&vr, 0);
```

Note: In C++ code, the search values can be passed to FindNearest using either a TVarRec (as above) or an ARRAYOFCONST construct. The example below performs the same search as the preceding example, but using ARRAYOFCONST.

```
Table1->FindNearest(ARRAYOFCONST(("Princess Island SCUBA")));
```

For Paradox and dBASE tables, the key must always be an index, which can be specified in the IndexName (see page 10222) property. If IndexName (see page 10222) is empty, FindNearest uses the table's primary index. Note that FindKey (see page 10196) does not work with dBASE expression indexes. Use the GotoKey (see page 10201) or GotoNearest (see page 10202) method instead.

For SQL tables, the key may correspond to a specified index in IndexName (see page 10222), or to a list of field names in the IndexFieldNames (see page 10220) property.

FindNearest positions the cursor either on a record that exactly matches the search criteria, or on the first record whose values are greater than those specified in the search criteria. If there are no records that match or exceed the specified criteria, FindNearest positions the cursor on the last record in the table. KeyExclusive (see page 10223) affects the boundary conditions of ranges and will affect the record selected by FindNearest.

See Also

FindKey (see page 10196)

GotoKey (see page 10201)

GotoNearest (see page 10202)

IndexName (see page 10222)

IndexFieldNames (see page 10220)

KeyExclusive (see page 10223)

Locate (see page 9915)

Executing a Search with Find Methods

53.1.19.4.23 TTable.GetCanModify Method

Pascal

```
function GetCanModify: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetCanModify();
```

Description

This is GetCanModify, a member of class TTable.

53.1.19.4.24 TTable.GetDataSource Method

Pascal

```
function GetDataSource: TDataSource; override;
```

C++

```
virtual __fastcall TDataSource GetDataSource();
```

Description

This is GetDataSource, a member of class TTable.

53.1.19.4.25 TTable.GetDetailLinkFields Method

Lists the field components that link this dataset as a detail of a master dataset.

Pascal

```
procedure GetDetailLinkFields(MasterFields: TList; DetailFields: TList); override;
```

C++

```
virtual __fastcall GetDetailLinkFields(TList MasterFields, TList DetailFields);
```

Description

GetDetailLinkFields fills two lists of TFields that define a master-detail relationship between this table and another (master) dataset. The MasterFields (see page 10224) list is filled with fields from the master table whose values must equal the values of the fields in the DetailFields list. The DetailFields list is filled with fields from the calling dataset.

See Also

MasterFields (see page 10224)

MasterSource (see page 10224)

53.1.19.4.26 TTable.GetFileName Method

Pascal

```
function GetFileName: string;
```

C++

```
__fastcall AnsiString GetFileName();
```

Description

This is GetFileName, a member of class TTable.

53.1.19.4.27 TTable.GetHandle Method

Pascal

```
function GetHandle(const IndexName: string; const IndexTag: string): HDBICur;
```

C++

```
__fastcall HDBICur GetHandle(const AnsiString IndexName, const AnsiString IndexTag);
```

Description

This is GetHandle, a member of class TTable.

53.1.19.4.28 TTable.GetIndexNames Method

Retrieves a list of available indexes for a table.

Pascal

```
procedure GetIndexNames(List: TStrings);
```

C++

```
__fastcall GetIndexNames(TStrings List);
```

Description

Call GetIndexNames to retrieve a list of all available indexes for a table. List is a string list object, created and maintained by the application, into which to retrieve the index names.

```
Table1.GetIndexNames(ListBox1.Items);
Table1->GetIndexNames(ListBox1->Items);
```

See Also

IndexName (see page 10222)

Obtaining Information About Indexes

53.1.19.4.29 TTable.GetLanguageDriverName Method

Pascal

```
function GetLanguageDriverName: string;
```

C++

```
__fastcall AnsiString GetLanguageDriverName();
```

Description

This is GetLanguageDriverName, a member of class TTable.

53.1.19.4.30 TTable.GetLookupCursor Method

Pascal

```
function GetLookupCursor(const KeyFields: string; CaseInsensitive: Boolean): HDBICur; override;
```

C++

```
virtual __fastcall HDBICur GetLookupCursor(const AnsiString KeyFields, Boolean
CaseInsensitive);
```

Description

This is GetLookupCursor, a member of class TTable.

53.1.19.4.31 TTable.GetTableType Method

Pascal

```
function GetTableType: TTableType;
```

C++

```
__fastcall TTableType GetTableType();
```

Description

This is GetTableType, a member of class TTable.

53.1.19.4.32 TTable.GotoCurrent Method

Synchronizes the current record for this table with the current record of a specified table component.

Pascal

```
procedure GotoCurrent(Table: TTable);
```

C++

```
__fastcall GotoCurrent(TTable Table);
```

Description

Call GotoCurrent to synchronize the cursor position for this table with the cursor position in another dataset that uses a different data source component, but which is connected to the same underlying database table. Table is the name of the table component whose cursor position to use for synchronizing.

Note: This procedure works only for table components that have the same DatabaseName (see page 10022) and TableName (see page 10226) properties. Otherwise, GotoCurrent raises an exception.

GotoCurrent is mainly for use in applications that have two table components that are linked to the same underlying database table through different data source components. It enables an application to ensure that separate views of the data appear to be linked.

See Also

DatabaseName (see page 10022)

TableName (see page 10226)

TDataSource (see page 7070)

53.1.19.4.33 TTable.GotoKey Method

Moves the cursor to a record specified by the current key.

Pascal

```
function GotoKey: Boolean;
```

C++

```
__fastcall Boolean GotoKey();
```

Description

Use GotoKey to move to a record specified by key values assigned with previous calls to SetKey (see page 10208) or EditKey (see page 10194) and actual search values indicated in the Fields property.

If GotoKey finds a matching record, it positions the cursor on the record and returns true. Otherwise the current cursor position remains unchanged, and GotoKey returns false.

For Paradox and dBASE tables, the key must always be an index, which can be specified in the IndexName (see page 10222) property. If IndexName (see page 10222) is empty, GotoKey uses the table's primary index. GotoKey works with dBASE expression indexes.

For SQL tables, the key may correspond to a specified index in IndexName (see page 10222), or to a list of field names in the IndexFieldNames (see page 10220) property.

See Also

Fields (see page 7023)

FindKey (see page 10196)

FindNearest (see page 10197)

GotoNearest (see page 10202)

SetKey (see page 10208)

State (see page 7036)

Executing a Search with Goto Methods

53.1.19.4.34 TTable.GotoNearest Method

Moves the cursor to the record that most closely matches the current key.

Pascal

```
procedure GotoNearest;
```

C++

```
__fastcall GotoNearest();
```

Description

Call GotoNearest to position the cursor on the record that is either the exact record specified by the current key values in the key buffer, or on the first record whose values exceed those specified. If there are no records that match or exceed the specified criteria, GotoNearest positions the cursor on the last record in the table.

Before calling GotoNearest, an application must specify key values by calling SetKey (see page 10208) or EditKey (see page 10194) to put the dataset is dsSetKey state, and then use FieldByName to populate the key buffer property with search values.

For Paradox and dBASE tables, the key must always be an index, which can be specified in the IndexName (see page 10222) property. If IndexName (see page 10222) is empty, GotoNearest uses the table's primary index. GotoNearest works with dBASE expression indexes.

For SQL tables, the key may correspond to a specified index in IndexName (see page 10222), or to a list of field names in the IndexFieldNames (see page 10220) property.

Note: KeyExclusive (see page 10223) determines which records are considered part of a search range.

See Also

FieldByName (see page 6936)

FindKey (see page 10196)

FindNearest (see page 10197)

GotoKey (see page 10201)

SetKey (see page 10208)

State (see page 7036)

Executing a Search with Goto Methods

53.1.19.4.35 TTable.InitFieldDefs Method

Pascal

```
procedure InitFieldDefs; override;
```

C++

```
virtual __fastcall InitFieldDefs();
```

Description

This is InitFieldDefs, a member of class TTable.

53.1.19.4.36 TTable.IsProductionIndex Method

Pascal

```
function IsProductionIndex(const IndexName: string): Boolean;
```

C++

```
__fastcall Boolean IsProductionIndex(const AnsiString IndexName);
```

Description

This is IsProductionIndex, a member of class TTable.

53.1.19.4.37 TTable.LockTable Method

Locks a Paradox or dBASE table.

Pascal

```
procedure LockTable(LockType: TLockType);
```

C++

```
__fastcall LockTable(TLockType LockType);
```

Description

Call LockTable to lock a Paradox or dBASE table to prevent other applications from placing a particular type of lock on the table. LockType specifies the lock requested by this application.

Requesting a read lock prevents other applications from reading a table. Requesting a write lock prevents other application from writing to a table.

An application can request both types of lock by calling LockTable twice, once for each lock type.

See Also

UnlockTable (see page 10212)

Exclusive (see page 10215)

53.1.19.4.38 TTable.NativeTableName Method

Pascal

```
function NativeTableName: PChar;
```

C++

```
__fastcall const char * NativeTableName();
```

Description

This is NativeTableName, a member of class TTable.

53.1.19.4.39 TTable.OpenIndexFile Method

Opens a specified dBASE non-production index file.

Pascal

```
procedure OpenIndexFile(const IndexName: string);
```

C++

```
__fastcall OpenIndexFile(const AnsiString IndexName);
```

Description

Call OpenIndexFile to open a specified dBASE non-production index file. IndexFileName is the name of the file to open. OpenIndexFile opens the file and adds its name to the list of index files in the IndexFiles (see page 10221) property.

While a non-production index file is open, it is maintained.

See Also

IndexFiles (see page 10221)

CloseIndexFile (see page 10188)

53.1.19.4.40 TTable.PrepareCursor Method

Pascal

```
procedure PrepareCursor; override;
```

C++

```
virtual __fastcall PrepareCursor();
```

Description

This is PrepareCursor, a member of class TTable.

53.1.19.4.41 TTable.PSGetCommandText Method

Gets the SQL command to be executed by PSExecute.

Pascal

```
function PSGetCommandText: string; override;
```

C++

```
virtual __fastcall AnsiString PSGetCommandText();
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

PSSetCommandText (see page 7585)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

53.1.19.4.42 TTable.PSGetCommandType Method

Gets the command type to be executed by PSExecute.

Pascal

```
function PSGetCommandType: TPSCommandType; override;
```

C++

```
virtual __fastcall TPSCommandType PSGetCommandType();
```

Description

PSGetCommandType gets the command type executed by PSExecute. The type may be a type of SQL command or other command types, such as stored procedures.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

PSGetCommandText (see page 7579)

Execute (see page 18466)

Options (see page 18440)

53.1.19.4.43 TTable.PSGetDefaultOrder Method

Returns the definition of an index that imposes the default order on the data included in data packets.

Pascal

```
function PSGetDefaultOrder: TIndexDef; override;
```

C++

```
virtual __fastcall TIndexDef PSGetDefaultOrder();
```

Description

The provider component calls PSGetDefaultOrder to generate a default index, called DEFAULT_ORDER, for the records that appear in data packets. This index defines the default sort order for records when they appear in the client dataset on the client.

PSGetDefaultOrder creates a TIndexDef object to describe the default index. The provider that calls PSGetDefaultOrder subsequently frees this TIndexDef object.

See Also

CreateDataPacket (see page 18518)

53.1.19.4.44 TTable.PSGetIndexDefs Method

Returns the definitions of all specified indexes defined for the dataset.

Pascal

```
function PSGetIndexDefs(IndexTypes: TIndexOptions): TIndexDefs; override;
```

C++

```
virtual __fastcall TIndexDefs PSGetIndexDefs(TIndexOptions IndexTypes);
```

Description

The provider component calls PSGetIndexDefs to locate an index that contains information it needs. For example, it uses PSGetIndexDefs to locate an index that contains the key fields returned by PSGetKeyFields. This information allows the provider to locate the records that need to change when it applies updates. The provider also calls this method to locate indexes that it adds to the metadata of data packets.

IndexTypes indicates the options required of the returned indexes. PSGetIndexDefs returns all indexes whose Options property includes the specified flags.

PSGetIndexDefs returns a TIndexDefs object that contains all the indexes of the dataset that match the IndexTypes parameter.

See Also

Options (see page 7291)

PSGetKeyFields (see page 7581)

PSGetDefaultOrder (see page 7580)

53.1.19.4.45 TTable.PSGetKeyFields Method

Returns the names of all fields required to uniquely identify records in the data packet.

Pascal

```
function PSGetKeyFields: string; override;
```

C++

```
virtual __fastcall AnsiString PSGetKeyFields();
```

Description

The provider component calls PSGetKeyFields to determine what information is required to uniquely identify records in the data packet. This information allows it to locate the records that need to change when it applies updates.

PSGetKeyFields returns a semicolon-delimited string that lists the names of the key fields of the dataset. By default (as implemented by TDataSet), these are the fields with pfInKey included in their ProviderFlags property.

See Also

PSGetDefaultOrder (see page 7580)

ProviderFlags (see page 7176)

PSGetIndexDefs (see page 7581)

53.1.19.4.46 TTable.PSGetTableName Method

Returns the name of the dataset table that appears in generated SQL statements.

Pascal

```
function PSGetTableName: string; override;
```

C++

```
virtual __fastcall AnsiString PSGetTableName();
```

Description

PSGetTableName returns the name of the underlying database table against which updates are applied when the provider's ResolveToDataSet property is false. This value is used for generating the SQL statements that apply updates. The generated SQL statements are subsequently executed by a call to PSExecuteStatement.

If PSGetTableName does not return a table name, the provider must have an OnGetTableName event handler to supply this value.

See Also

applyupdates (see page 18459)

PSExecuteStatement (see page 7579)

ResolveToDataSet (see page 18531)

ongettablename (see page 18542)

53.1.19.4.47 TTable.PSSetCommandText Method

Assigns a specified SQL command to be executed by PSExecute.

Pascal

```
procedure PSSetCommandText(const CommandText: string); override;
```

C++

```
virtual __fastcall PSSetCommandText(const AnsiString CommandText);
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

The value of the CommandText parameter replaces the dataset's SQL statement (if it has one), or replaces the name of an underlying database table or stored procedure.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

53.1.19.4.48 TTable.PSSetParams Method

Assigns the specified parameter values to the dataset.

Pascal

```
procedure PSSetParams(AParams: TParams); override;
```

C++

```
virtual __fastcall PSSetParams(TParams AParams);
```

Description

PSSetParams assigns the parameter values specified by AParams to the parameters of the dataset.

See Also

Params (see page 8489)

PSGetParams (see page 7582)

53.1.19.4.49 TTable.RenameTable Method

Renames the Paradox or dBASE table associated with this table component.

Pascal

```
procedure RenameTable(const NewTableName: string);
```

C++

```
__fastcall RenameTable(const AnsiString NewTableName);
```

Description

Call RenameTable to give a new name to the Paradox or dBASE table underlying this table component. RenameTable renames the table and any support files (for example, Paradox .DB (see page 6725) and .MB files).

See Also

CreateTable (see page 10189)

DeleteTable (see page 10192)

EmptyTable (see page 10196)

LockTable (see page 10203)

UnlockTable (see page 10212)

BatchMove (see page 10186)

Renaming a Table

Using TTable

53.1.19.4.50 TTable.SetKey Method

Enables setting of keys and ranges for a dataset prior to a search.

Pascal

```
procedure SetKey;
```

C++

```
__fastcall SetKey();
```

Description

Call SetKey to put the dataset into dsSetKey state and clear the current contents of the key buffer. The FieldByName method can then be used to supply a new set of search values prior to conducting a search.

Note: To modify an existing key or range, call EditKey (see page 10194).

See Also

EditKey (see page 10194)

SetRange (see page 10209)

SetRangeEnd (see page 10211)

SetRangeStart (see page 10212)

State (see page 7036)

Executing a Search with Goto Methods

53.1.19.4.51 TTable.SetRange Method

Sets the starting and ending values of a range, and applies it.

Pascal

```
procedure SetRange(const StartValues: array of const; const EndValues: array of const);
```

C++

```
__fastcall SetRange(const array of const StartValues, const array of const EndValues);
```

Description

Call SetRange to specify a range and apply it to the dataset. The new range replaces the currently applied range, if any.

StartValues indicates the field values that designate the first record in the range. StartValues_Size specifies the index of the last value in StartValues (one less than the total number of values).

EndValues indicates the field values that designate the last record in the range. EndValues_Size specifies the index of the last value in EndValues (one less than the total number of values).

SetRange combines the functionality of SetRangeStart (see page 10212), SetRangeEnd (see page 10211), and ApplyRange (see page 10185) in a single procedure call. SetRange performs the following functions:

1Puts the dataset into dsSetKey state.

2Erases any previously specified starting range values and ending range values.

3Sets the start and end range values.

4Applies the range to the dataset.

If either StartValues or EndValues has fewer elements than the number of fields in the current index, then the remaining entries are set to NULL.

After a call to SetRange, the cursor is left on the first record in the range.

Note: With Paradox or dBASE tables, SetRange works only on indexed fields. With SQL databases, SetRange also works with any columns specified in the IndexFieldNames (see page 10220) property.

Delphi Examples:

```
{
The following example sets a range for a dataset. The form
requires two edit boxes, a data source, a client dataset or
table, a db grid, and a button.  Make sure that the Table1
info in FormActivate agrees with the table used.  Also, the
caption for the button must be initialized to '&ApplyRange'.
}
```

```
procedure TForm1.FormActivate(Sender: TObject);
begin
  Table1.DatabaseName := 'DBDEMOS';
  Table1.TableName := 'CustInfo';
  Table1.Active := True;
  Table1.IndexName := '';
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  if Button2.Caption = '&Apply Range' then
    begin
      Table1.SetRange([Edit1.Text],[Edit2.Text]);
      Button2.Caption := '&Drop Range';
    end
  else
    begin
      Table1.CancelRange;
      Table1.Refresh;
      Button2.Caption := '&Apply Range';
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
    else
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
```

```
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.Active:= False;
  end;
```

See Also

ApplyRange (see page 10185)

CancelRange (see page 10186)

EditRangeEnd (see page 10194)

EditRangeStart (see page 10195)

FieldValues (see page 7025)

IndexFieldNames (see page 10220)

KeyExclusive (see page 10223)

SetRangeEnd (see page 10211)

SetRangeStart (see page 10212)

53.1.19.4.52 TTable.SetRangeEnd Method

Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.

Pascal

```
procedure SetRangeEnd;
```

C++

```
__fastcall SetRangeEnd();
```

Description

Call SetRangeEnd to put the dataset into dsSetKey state, erase any previous end range values, and set them to NULL. Subsequent key buffer field assignments can be made using the FieldByName method to set the ending values for a range.

After assigning end-range values to FieldValues, call ApplyRange (see page 10185) to activate the modified range.

Note: With Paradox or dBASE tables, SetRangeEnd works only on indexed fields. With SQL databases, SetRangeEnd also works with any columns specified in the IndexFieldNames (see page 10220) property.

See Also

ApplyRange (see page 10185)

CancelRange (see page 10186)

EditRangeEnd (see page 10194)

EditRangeStart (see page 10195)

FieldByName (see page 6936)

IndexFieldNames (see page 10220)

KeyExclusive (see page 10223)

SetRange (see page 10209)

SetRangeStart (see page 10212)

53.1.19.4.53 TTable.SetRangeStart Method

Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.

Pascal

```
procedure SetRangeStart;
```

C++

```
__fastcall SetRangeStart();
```

Description

Call SetRangeStart to put the dataset into dsSetKey state, erase any previous start range values, and set them to NULL. Subsequent field assignments to the FieldValues property indicate the actual set of starting values for a range.

After assigning start-range values to FieldValues, call ApplyRange (see page 10185) to activate the modified range.

Note: With Paradox or dBASE tables, SetRangeStart works only on indexed fields. With SQL databases, SetRangeStart also works with any columns specified in the IndexFieldNames (see page 10220) property.

See Also

ApplyRange (see page 10185)

CancelRange (see page 10186)

EditRangeEnd (see page 10194)

EditRangeStart (see page 10195)

FieldValues (see page 7025)

IndexFieldNames (see page 10220)

KeyExclusive (see page 10223)

SetRange (see page 10209)

SetRangeEnd (see page 10211)

53.1.19.4.54 TTable.UnlockTable Method

Removes a previously applied lock on a Paradox or dBASE table.

Pascal

```
procedure UnlockTable(LockType: TLockType);
```

C++

```
__fastcall UnlockTable(TLockType LockType);
```

Description

Call UnlockTable to remove a lock previously applied to a Paradox or dBASE table. LockType specifies the lock to remove.

Removing a read lock enables other applications to read a table. Removing a write lock enables other application to write to a table.

An application can remove both types of lock by calling UnlockTable twice, once for each lock type.

See Also

LockTable (see page 10203)

53.1.19.4.55 TTable.UpdateIndexDefs Method

Pascal

```
procedure UpdateIndexDefs; override;
```

C++

```
virtual __fastcall UpdateIndexDefs();
```

Description

This is UpdateIndexDefs, a member of class TTable.

53.1.19.5 TTable Properties

The properties of the TTable class are listed here.

Protected Properties

	Name	Description
R	MasterLink (see page 10224)	This is MasterLink, a member of class TTable.

Public Properties

	Name	Description
R	Exists (see page 10215)	Indicates whether the underlying database table exists.
R	IndexFieldCount (see page 10220)	Indicates the number of fields that comprise the current key.
	IndexFields (see page 10221)	Lists the fields of the current index.
	KeyExclusive (see page 10223)	Specifies how the upper and lower boundaries for a range should be interpreted.
	KeyFieldCount (see page 10223)	Specifies the number of fields to use when conducting a partial key search on a multi-field key.
	TableLevel (see page 10226)	Specifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) driver-dependent table level.

Published Properties

	Name	Description
	Constraints (see page 10214)	Specifies record-level constraints that must be met when editing the data.
	DefaultIndex (see page 10214)	Specifies if the data in the table should be ordered on a default index when opened.
	Exclusive (see page 10215)	Enables an application to gain sole access to a Paradox or dBASE table.
	FieldDefs (see page 10217)	Points to the list of field definitions for the dataset.

	IndexDefs (see page 10219)	Contains information about the indexes for a table.
	IndexFieldNames (see page 10220)	Lists the columns to use as an index for a table.
	IndexFiles (see page 10221)	Specifies one or more dBASE index file to use for a dBASE table that uses non-production indexes.
	IndexName (see page 10222)	Identifies a secondary index for the table.
	MasterFields (see page 10224)	Specifies one or more fields in a master table to link with corresponding fields in this table in order to establish a master-detail relationship between the tables.
	MasterSource (see page 10224)	Specifies the name of the data source for a dataset to use as a master table in establishing a detail-master relationship between this table and another one.
R	Ranged (see page 10225)	This is Ranged, a member of class TTable.
	ReadOnly (see page 10225)	Specifies whether a table is read-only for this application.
	StoreDefs (see page 10226)	Indicates whether the table's field and index definitions persist with the data module or form.
	TableName (see page 10226)	Indicates the name of the database table that this component encapsulates.
	TableType (see page 10227)	Indicates the database table structure for the table that this component represents.
	UpdateMode (see page 10228)	Determines how the Borland Database Engine (BDE) finds records when updating to an SQL database.
	UpdateObject (see page 10228)	Specifies the update object component used to update a read-only result set when cached updates are enabled.

53.1.19.5.1 TTable.Constraints Property

Specifies record-level constraints that must be met when editing the data.

Pascal

```
property Constraints: TCheckConstraints;
```

C++

```
__property TCheckConstraints Constraints;
```

Description

Use Constraints to read or add record-level constraints to the dataset. Record-level constraints usually impose relationships between the fields in a single record. Constraints that limit the value of a single field should be added to the individual field components.

See Also

TCheckConstraint (see page 6841)

CustomConstraint (see page 7156)

ImportedConstraint (see page 7167)

Constraining Data Values

53.1.19.5.2 TTable.DefaultIndex Property

Specifies if the data in the table should be ordered on a default index when opened.

Pascal

```
property DefaultIndex: Boolean;
```

C++

```
__property Boolean DefaultIndex;
```

Description

When this property is set to false, the BDE does not use an ORDER BY clause when opening a table on a SQL server. When

DefaultIndex is true, the BDE attempts to order the data based on the primary key or a unique index when opening the table. DefaultIndex defaults to true.

See Also

Active (see page 7004)

53.1.19.5.3 TTable.Exclusive Property

Enables an application to gain sole access to a Paradox or dBASE table.

Pascal

```
property Exclusive: Boolean;
```

C++

```
__property Boolean Exclusive;
```

Description

Use Exclusive to prevent other applications from accessing a Paradox or dBASE table while it is open in this application. Before opening the table, set Exclusive to true. A table must be closed before changing the Exclusive property.

When Exclusive is true, then when the application successfully opens the table, no other application can access it. If the table for which the application has requested exclusive access is already in use by another application, an exception is raised. To handle such exceptions, wrap the code that opens the table in a try..catchexcept block.

Do not set Exclusive to true at design time if you also set the Active (see page 9929) property to true at design time. In this case an exception is raised because the table is already in use by the IDE.

Note: An application can attempt to set Exclusive to true for SQL tables, but some SQL servers do not support exclusive table locking. Others may grant an exclusive read/write lock, but still permit read-only access to other applications. For more information about exclusive locking of database tables for a specific server, see the server's documentation.

See Also

Active (see page 7004)

Close (see page 6920)

Open (see page 6976)

Controlling Read/Write Access to Local Tables

Using TTable

53.1.19.5.4 TTable.Exists Property

Indicates whether the underlying database table exists.

Pascal

```
property Exists: Boolean;
```

C++

```
__property Boolean Exists;
```

Description

Read Exists at runtime to determine whether a database table exists. If the table does not exist, create a table from the field definitions and index definitions using the CreateTable (see page 10189) method.

This property is read-only.

Delphi Examples:

```
{ The following example shows how to create a table. }
procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Close;
  Table1.DeleteTable;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
    else
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
```

```
      Table1.Active:= True;
      for i := 1 to 20 do
        Table1.AppendRecord([i*111, i*222]);
    end;
  end;
  DS2.DataSet:= Table1;
  DBGrid2.DataSource.DataSet:= Table1;
  Table1.Active:= True;
end;
```

See Also

CreateTable (see page 10189)

RenameTable (see page 10208)

53.1.19.5.5 TTable.FieldDefs Property

Points to the list of field definitions for the dataset.

Pascal

```
property FieldDefs: TFieldDefs;
```

C++

```
__property TFieldDefs FieldDefs;
```

Description

FieldDefs lists the field definitions for a dataset. While an application can examine FieldDefs to explore the field definitions for a dataset, it should not change these definitions unless creating a new table with CreateTable or CreateDataSet.

To access fields and field values in a dataset, use the Fields, AggFields, and FieldValues properties, and the FieldsByName method.

Note: If the dataset includes object field descendants, FieldDefs represents a hierarchical view of the data, meaning that the definitions include object field definitions. To determine the definitions in a flattened view, use FieldDefList instead.

C++ Examples:

```
/*
The following code creates and activates a client dataset in
the form's OnCreate event handler.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TFieldDefs *pDefs = CDS2->FieldDefs;
  TFieldDef *pDef = pDefs->AddFieldDef();
  pDef->DataType = ftInteger;
  pDef->Name = "Field1";

  pDef = pDefs->AddFieldDef();
  pDef->DataType = ftString;
  pDef->Size = 10;
  pDef->Name = "Field2";

  TIndexDef *pIDef = CDS2->IndexDefs->AddIndexDef();
  pIDef->Fields = "Field1";
  pIDef->Name = "IntIndex";

  CDS2->CreateDataSet();
}
```

Delphi Examples:

```
{
The following code creates and activates a client dataset in
the form’s OnCreate event handler.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  with CDS2 do
  begin
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftInteger;
      Name := 'Field1';
    end;
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftString;
      Size := 10;
      Name := 'Field2';
    end;
    with IndexDefs.AddIndexDef do
    begin
      Fields := 'Field1';
      Name := 'IntIndex';
    end;
    CreateDataSet;
  end;
end;
{ The following example shows how to create a table. }
procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Close;
  Table1.DeleteTable;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
    else
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
```

```
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.Active:= True;
  end;
```

See Also

CreateTable (see page 10189)

Fields (see page 7023)

FieldValues (see page 7025)

ClearFields (see page 6919)

GetFieldList (see page 6953)

GetFieldNames (see page 6954)

FieldByName (see page 6936)

FindField (see page 6938)

SetFields (see page 6997)

ObjectView (see page 7031)

AggFields (see page 7006)

CreateDataSet (see page 8401)

Creating and Deleting Tables

53.1.19.5.6 TTable.IndexDefs Property

Contains information about the indexes for a table.

Pascal

```
property IndexDefs: TIndexDefs;
```

C++

```
__property TIndexDefs IndexDefs;
```

Description

IndexDefs is a collection of index definitions, each of which describes an available index for the table. Define the index definitions of a table before calling CreateTable (see page 10189) or creating a table at design time.

Ordinarily, an application accesses or specifies indexes at runtime through the IndexFieldNames (see page 10220) or IndexFields (see page 10221) properties.

If IndexDefs is updated or manually edited, the StoreDefs (see page 10226) property becomes true.

Note: The index definitions in IndexDefs may not always reflect the current indexes available for a table. Before examining IndexDefs, call its Update method to refresh the list.

See Also

IndexFieldNames (see page 10220)

StoreDefs (see page 10226)

TIndexDef (see page 7283)

CreateTable (see page 10189)

53.1.19.5.7 TTable.IndexFieldCount Property

Indicates the number of fields that comprise the current key.

Pascal

```
property IndexFieldCount: Integer;
```

C++

```
__property int IndexFieldCount;
```

Description

Examine IndexFieldCount to determine the number of fields that comprise the current index. For indexes based on a single column, IndexFieldCount returns 1. For multi-column indexes, IndexFieldCount indicates the number of fields upon which the index is based.

See Also

IndexDefs (see page 10219)

IndexFields (see page 10221)

53.1.19.5.8 TTable.IndexFieldNames Property

Lists the columns to use as an index for a table.

Pascal

```
property IndexFieldNames: string;
```

C++

```
__property AnsiString IndexFieldNames;
```

Description

Use IndexFieldNames as an alternative method of specifying the index to use for a table. In IndexFieldNames, specify the name of each column to use as an index for a table. Ordering of column names is significant. Separate names with semicolon.

For Paradox and dBASE any column names specified in IndexFieldNames must already be indexed. This must be a single index using the same fields in the same order as specified in IndexFieldNames.

For SQL-based tables, the specified columns need not be indexed.

Note: The IndexFieldNames and IndexName (see page 10222) properties are mutually exclusive. Setting one clears the other.

See Also

IndexDefs (see page 10219)

IndexName (see page 10222)

Creating an Index with IndexFieldNames

Specifying the Data to Display using TSQLDataSet

53.1.19.5.9 TTable.IndexFields Property

Lists the fields of the current index.

Pascal

```
property IndexFields [Index: Integer]: TField;
```

C++

```
__property TField IndexFields[int Index];
```

Description

IndexFields is a zero-based array of field objects, each of which corresponds to a field in the current index. Index is an ordinal value indicating the position of a field in the index. The first field in the index is IndexFields[0], the second is IndexFields[1], and so on.

Note: Do not set IndexFields directly. Instead use the IndexFieldNames (see page 10220) property to order datasets on the fly at runtime.

See Also

AddIndex (see page 10184)

IndexDefs (see page 10219)

IndexFieldNames (see page 10220)

IndexName (see page 10222)

53.1.19.5.10 TTable.IndexFiles Property

Specifies one or more dBASE index file to use for a dBASE table that uses non-production indexes.

Pascal

```
property IndexFiles: TStrings;
```

C++

```
__property TStrings IndexFiles;
```

Description

Use IndexFiles to specify dBASE non-production index files and dBASE III PLUS-style .NDX indexes to use with the table. Set IndexFiles to the name of the non-production .MDX file. All index tags in the file become available for activation. This is as opposed to the index tags for a production index that become available automatically on opening the table. When using dBASE III PLUS-style .NDX indexes, add individual files to the list in the IndexFiles property. Both types of indexes become accessible through such means as the IndexName (see page 10222) property and the GetIndexNames (see page 10200) method.

When index files are added to the list of available indexes, the table component opens them. The indexes are updated as data is added to, changed in, or deleted from the table. When files are removed from the list, the table component closes them, and they are no longer maintained.

Note: At design time, use the Object Inspector to add or remove index file names for the IndexFiles property. At runtime, set IndexFiles as a string list.

See Also

IndexFieldNames (see page 10220)

IndexName (see page 10222)

GetIndexNames (see page 10200)

Specifying a dBASE Index File

Using TTable

53.1.19.5.11 TTable.IndexName Property

Identifies a secondary index for the table.

Pascal

```
property IndexName: string;
```

C++

```
__property AnsiString IndexName;
```

Description

Use IndexName to specify an alternative index for a table. If IndexName is empty, a table's sort order is based on its default index, or, for dBASE tables, on its physical record order.

If IndexName contains a valid index name, then that index determines the sort order of records. For dBASE tables, an index name supplied to the IndexName property must either reside in the table's master index file, or in another index file already specified in the IndexFiles (see page 10221) property.

Note: IndexFieldNames (see page 10220) and IndexName are mutually exclusive. Setting one clears the other.

See Also

IndexFieldNames (see page 10220)

IndexFiles (see page 10221)

DefaultIndex (see page 10214)

Specifying a dBASE Index File

Specifying the Data to Display using TSQLDataSet

Using TTable

53.1.19.5.12 TTable.KeyExclusive Property

Specifies how the upper and lower boundaries for a range should be interpreted.

Pascal

```
property KeyExclusive: Boolean;
```

C++

```
__property Boolean KeyExclusive;
```

Description

Use KeyExclusive to specify whether a range includes or excludes the records that match the starting and ending values of the range. By default, KeyExclusive is false, meaning that matching values are included.

To restrict a range to those records that are greater than the specified starting value and less than the specified ending value, set KeyExclusive to true.

See Also

FindNearest (see page 10197)

GotoNearest (see page 10202)

SetRange (see page 10209)

SetRangeEnd (see page 10211)

SetRangeStart (see page 10212)

53.1.19.5.13 TTable.KeyFieldCount Property

Specifies the number of fields to use when conducting a partial key search on a multi-field key.

Pascal

```
property KeyFieldCount: Integer;
```

C++

```
__property int KeyFieldCount;
```

Description

Use KeyFieldCount to limit a search based on a multi-field key to a consecutive subset of those fields. For example, if the primary key for a dataset consists of three fields, a partial-key search can be conducted using only the first field in the key by setting KeyFieldCount to 1. If KeyFieldCount is 0, the dataset searches on all fields in the key.

Note: Searches are only conducted based on consecutive key fields beginning with the first field in the key. For example if a key consists of three fields, an application can set KeyFieldCount to 1 to search on the first field, 2 to search on the first and second fields, or 3 to search on all fields. By default KeyFieldCount is initially set to include all fields in a search.

See Also

FindKey (see page 10196)

FindNearest (see page 10197)

GotoKey (see page 10201)

GotoNearest (see page 10202)

Locate (see page 9915)

53.1.19.5.14 TTable.MasterFields Property

Specifies one or more fields in a master table to link with corresponding fields in this table in order to establish a master-detail relationship between the tables.

Pascal

```
property MasterFields: string;
```

C++

```
__property AnsiString MasterFields;
```

Description

Use MasterFields after setting the MasterSource (see page 10224) property to specify the names of one or more fields to use in establishing a detail-master relationship between this table and the one specified in MasterSource (see page 10224).

MasterFields is a string containing one or more field names in the master table. Separate field names with semicolons.

Each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this table for display.

Note: At design time, use the Field Link designer to establish the master-detail relationship between two tables.

See Also

MasterSource (see page 10224)

Making the Table a Detail of Another Dataset

53.1.19.5.15 TTable.MasterLink Property

Pascal

```
property MasterLink: TMasterDataLink;
```

C++

```
__property TMasterDataLink MasterLink;
```

Description

This is MasterLink, a member of class TTable.

53.1.19.5.16 TTable.MasterSource Property

Specifies the name of the data source for a dataset to use as a master table in establishing a detail-master relationship between this table and another one.

Pascal

```
property MasterSource: TDataSource;
```

C++

```
__property TDataSource MasterSource;
```

Description

Use MasterSource to specify the name of the data source component whose DataSet property identifies a dataset to use as a master table in establishing a detail-master relationship between this table and another one.

Note: At design time choose an available data source from the MasterSource property's drop-down list in the Object Inspector.

After setting the MasterSource property, specify which fields to use in the master table by setting the MasterFields (see page 10224) property. At runtime each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this table for display.

Note: At design time, use the Field Link designer to establish the master-detail relationship between two tables.

See Also

MasterFields (see page 10224)

DataSetField (see page 7013)

Making the Table a Detail of Another Dataset

53.1.19.5.17 TTable.Ranged Property

Pascal

```
property Ranged: Boolean;
```

C++

```
__property Boolean Ranged;
```

Description

This is Ranged, a member of class TTable.

53.1.19.5.18 TTable.ReadOnly Property

Specifies whether a table is read-only for this application.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use the ReadOnly property to prevent users from updating, inserting, or deleting data in the table. By default, ReadOnly is false, meaning users can potentially alter a table's data.

Note: Even if ReadOnly is false, users may not be able to modify or add data to a table. Other factors, such as insufficient SQL privileges for the application or its current user may prevent successful alterations.

To guarantee that users cannot modify or add data to a table,

1Set the Active (see page 9929) property to false.

2Set ReadOnly to true.

When ReadOnly is true, the table's CanModify property is false.

See Also

Active (see page 7004)

CanModify

Controlling Read/Write Access to Local Tables

53.1.19.5.19 TTable.StoreDefs Property

Indicates whether the table's field and index definitions persist with the data module or form.

Pascal

```
property StoreDefs: Boolean;
```

C++

```
__property Boolean StoreDefs;
```

Description

If StoreDefs is true, the table's index and field definitions are stored with the data module or form. Setting StoreDefs to true makes the CreateTable (see page 10189) method into a one-step procedure that creates fields, indexes, and validity checks at runtime.

StoreDefs is false by default. It becomes true whenever FieldDefs (see page 10217) or IndexDefs (see page 10219) is updated or edited manually; to prevent edited (or imported) definitions from being stored, reset StoreDefs to false.

See Also

FieldDefs (see page 7019)

IndexDefs (see page 10219)

CreateTable (see page 10189)

53.1.19.5.20 TTable.TableLevel Property

Specifies the Borland (see page 1761) Database (see page 10021) Engine (BDE) driver-dependent table level.

Pascal

```
property TableLevel: Integer;
```

C++

```
__property int TableLevel;
```

Description

Use TableLevel when creating a table to specify that the table associated with this dataset requires a specific level of BDE driver. If TableLevel is 0, the default, this property is ignored.

For more information about BDE driver table level specification, see the online BDE help file, BDE32.HLP.

See Also

CreateTable (see page 10189)

Database (see page 10021)

TableType (see page 10227)

53.1.19.5.21 TTable.TableName Property

Indicates the name of the database table that this component encapsulates.

Pascal

```
property TableName: TFileName;
```

C++

```
__property TFileName TableName;
```

Description

Use TableName to specify the name of the database table this component encapsulates. To set TableName to a meaningful value, the DatabaseName (see page 10022) property should already be set. If DatabaseName (see page 10022) is set at design time, then select a valid table name from the TableName drop-down list in the Object Inspector.

Note: To set TableName, the Active (see page 9929) property must be false.

See Also

Active (see page 7004)

DatabaseName (see page 10022)

TableType (see page 10227)

Creating and Deleting Tables

Opening and Closing Datasets

Using TTable

Specifying the Data to Display using TSQLDataSet

Using TSQLTable

53.1.19.5.22 TTable.TableType Property

Indicates the database table structure for the table that this component represents.

Pascal

```
property TableType: TTableType;
```

C++

```
__property TTableType TableType;
```

Description

Use TableType to specify the database table structure for a dBASE, Paradox, FoxPro or ASCII table. TableType does not apply to tables for remote SQL servers. TableType can be set to any of the following values:

Value	Meaning
ttDefault	(Default) Determine table type based on file extension for the table.
ttParadox	Table is a Paradox table.
ttDBase	Table is a dBASE table.
ttFoxPro	Table is a FoxPro table.
ttASCII	Table is a text file with comma-delimited, quoted strings for each field
DB (see page 6725) or none	Paradox table
DBF	dBASE table
TXT	ASCII table

If TableType is set to ttDefault, the table's file-name extension determines the table type:

Extension	Meaning
DB (see page 6725) or none	Paradox table
DBF	dBASE table
TXT	ASCII table

If TableType is ttParadox, ttDBase, ttFoxPro or ttASCII, the table type is assumed to be of the type specified, regardless of the table's file name extension.

See Also

IndexName (see page 10222)

TableName (see page 10226)

Creating and Deleting Tables

Specifying the Table Type for Local Tables

Using TTable

Using TTable

53.1.19.5.23 TTable.UpdateMode Property

Determines how the Borland Database Engine (BDE) finds records when updating to an SQL database.

Pascal

```
property UpdateMode: TUpdateMode;
```

C++

```
__property TUpdateMode UpdateMode;
```

Description

Use UpdateMode to specify the criteria to use when locating a record in the dataset. UpdateMode specifies whether modified records are located based on all columns (fields), on only the key fields, or on the key fields plus the original values of fields that have been modified.

See Also

UpdateRecordTypes (see page 9936)

53.1.19.5.24 TTable.UpdateObject Property

Specifies the update object component used to update a read-only result set when cached updates are enabled.

Pascal

```
property UpdateObject: TDataSetUpdateObject;
```

C++

```
__property TDataSetUpdateObject UpdateObject;
```

Description

Use UpdateObject to specify the TUpdateObject component to use in an application that must be able to update a read-only result set returned by the Borland Database Engine (BDE).

The BDE always attempts to provide an updatable, or "live" query result unless an application specifically requests a read-only

view of data. In some cases, such as a query made against multiple tables, a live result set cannot be returned. In these cases, UpdateObject can be used to specify a TUpdateSQL component that performs updates as a separate transaction that is transparent to the application.

See Also

CachedUpdates (see page 9932)

ApplyUpdates (see page 9894)

TUpdateSQL (see page 10229)

Using the BDE to Cache Updates

Using Update Objects to Update a Dataset

Updating a Read-only Result Set

Using TQuery

Using TUpdateSQL to Update a Dataset

53.1.20 TUpdateSQL Class

TUpdateSQL applies cached updates on behalf of queries or stored procedures that can't post updates directly.

Class Hierarchy

Pascal

```
TUpdateSQL = class(TSQLUpdateObject);
```

C++

```
class TUpdateSQL : public TSQLUpdateObject;
```

File

DBTables

Description

Use a TUpdateSQL object to provide SQL (see page 10242) statements used to update read-only datasets represented by TQuery (see page 10059) or TStoredProc (see page 10139) components when cached updates are enabled. A dataset is read-only either by design or circumstance. If a dataset is read-only by design, the application itself does not provide a user interface for updating data, but may institute a programmatic scheme behind the scenes. If a dataset is read-only by circumstance, it indicates that the Borland (see page 1761) Database Engine (BDE) returned a read-only result set. This usually happens for queries made against multiple tables. Such queries are, by SQL (see page 10242)-92 definitions, read-only.

TUpdateSQL provides a mechanism for circumventing what some developers consider an SQL (see page 10242)-92 limitation. It enables a developer to provide INSERT, UPDATE, and DELETE statements for performing separate update queries on otherwise read-only result sets in such a manner that the separate update queries are transparent to the end user.

In practical application, a TUpdateSQL object is placed on a data module or form, and linked to a TQuery (see page 10059) or TStoredProc (see page 10139) component through that component's UpdateObject property. If the UpdateObject property

points to a valid TUpdateSQL object, the SQL (see page 10242) statements belonging to the update object are automatically applied when cached updates are applied.

See Also

TQuery (see page 10059)

TStoredProc (see page 10139)

TBDEDataSet (see page 9878)

Using Update Objects to Update a Dataset

Updating a Read-only Result Set

53.1.20.1 TUpdateSQL Members

The following tables list the members exposed by TUpdateSQL.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDataSet (see page 10235)	GetDataSet returns the value of the DataSet property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetSQL (see page 10236)	Returns the SQL statement that performs a specified update operation.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetDataSet (see page 10236)	SetDataSet sets the value of the DataSet property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SQLChanged (see page 10237)	This is SQLChanged, a member of class TUpdateSQL.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Apply (see page 10234)	This is the overview for the Apply method overload.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 10233)	Creates an instance of an update object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 10233)	Frees an instance of an update object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	ExecSQL (see page 10235)	Executes a specified type of SQL (see page 10242) statement to perform an update for an otherwise read-only results set when cached updates is enabled.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetParams (see page 10236)	This is the overview for the SetParams method overload.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DatabaseName (see page 10238)	Identifies the database to which updates are applied.
	DataSet (see page 10239)	Identifies the dataset that holds the updated data that the update object applies.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Query (see page 10241)	Returns the query object used to perform a specified kind of update.
	SessionName (see page 10242)	Identifies the session under which updates are applied.
	SQL (see page 10242)	Returns a specified SQL statement used when applying cached updates.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	DeleteSQL (see page 10239)	Specifies the SQL (see page 10242) DELETE statement to use when applying a cached deletion of a record.
	InsertSQL (see page 10240)	Specifies the SQL (see page 10242) INSERT statement to use when applying a cached insertion of a record.
	ModifySQL (see page 10240)	Specifies the SQL (see page 10242) UPDATE statement to use when applying an update to a record and cached updates is enabled.

	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

53.1.20.2 TUpdateSQL.Create Constructor

Creates an instance of an update object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TUpdateSQL(TComponent * AOwner);
```

Description

Call Create to instantiate an update object at run time. You do not need to call Create for update objects placed in a data module or form at design time. Delphi automatically handles these objects.

See Also

Destroy (see page 10233)

53.1.20.3 TUpdateSQL.Destroy Destructor

Frees an instance of an update object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TUpdateSQL();
```

Description

Do not call Destroy directly in an application. Usually destruction of update objects is handled automatically by Delphi. If an application creates its own instance of an update object, however, the application should call Free, which verifies that the update object is not nil before calling Destroy

See Also

Create (see page 10233)

Free (see page 22300)

53.1.20.4 TUpdateSQL Methods

The methods of the TUpdateSQL class are listed here.

Protected Methods

	Name	Description
	GetDataSet (see page 10235)	GetDataSet returns the value of the DataSet property.
	GetSQL (see page 10236)	Returns the SQL statement that performs a specified update operation.
	SetDataSet (see page 10236)	SetDataSet sets the value of the DataSet property.
	SQLChanged (see page 10237)	This is SQLChanged, a member of class TUpdateSQL.

Public Methods

	Name	Description
	Apply (see page 10234)	This is the overview for the Apply method overload.
V	ExecSQL (see page 10235)	Executes a specified type of SQL (see page 10242) statement to perform an update for an otherwise read-only results set when cached updates is enabled.
V	SetParams (see page 10236)	This is the overview for the SetParams method overload.

53.1.20.4.1 Apply Method

This is the overview for the Apply method overload.

Overload List

	Name	Description
	TUpdateSQL.Apply (TDataset, TUpdateKind) (see page 10234)	Sets the parameters for a specified SQL (see page 10242) statement type, and executes the resulting statement.
V	TUpdateSQL.Apply (TUpdateKind) (see page 10234)	Sets the parameters for a specified SQL (see page 10242) statement type, and executes the resulting statement.

53.1.20.4.1.1 TUpdateSQL.Apply Method (TDataset, TUpdateKind)

Sets the parameters for a specified SQL (see page 10242) statement type, and executes the resulting statement.

Pascal

```
procedure Apply(UpdateKind: TUpdateKind); override; overload;
procedure Apply(ADataset: TDataset; UpdateKind: TUpdateKind); overload; reintroduce;
```

C++

```
virtual __fastcall Apply(TUpdateKind UpdateKind);
__fastcall Apply(TDataset ADataset, TUpdateKind UpdateKind);
```

Description

Call Apply to set parameters for an SQL (see page 10242) statement and execute it to update a record. UpdateKind indicates which SQL (see page 10242) statement to bind and execute, and can be one of the following values:

Value	Meaning
ukDelete	Bind and execute the SQL (see page 10242) statement in the DeleteSQL (see page 10239) property
ukInsert	Bind and execute the SQL (see page 10242) statement in the InsertSQL (see page 10240) property
ukModify	Bind and execute the SQL (see page 10242) statement in the ModifySQL (see page 10240) property

Apply is primarily intended for manually executing update statements from an OnUpdateRecord event handler.

Note: If an SQL (see page 10242) statement does not contain parameters, it is more efficient to call ExecSQL (see page 10235) instead of Apply.

See Also

DeleteSQL (see page 10239)

ExecSQL (see page 10235)

InsertSQL (see page 10240)

ModifySQL (see page 10240)

SetParams (see page 10237)

Using Multiple Update Objects

53.1.20.4.2 TUpdateSQL.ExecSQL Method

Executes a specified type of SQL (see page 10242) statement to perform an update for an otherwise read-only results set when cached updates is enabled.

Pascal

```
procedure ExecSQL(UpdateKind: TUpdateKind); virtual;
```

C++

```
virtual __fastcall ExecSQL(TUpdateKind UpdateKind);
```

Description

Call ExecSQL to execute the SQL (see page 10242) statement necessary for updating the records belonging to a read-only result set when cached updates is enabled. UpdateKind specifies the statement to execute, and can be one of the following values:

Value	Meaning
ukDelete	Execute the SQL (see page 10242) statement used to delete records in the dataset (DeleteSQL (see page 10239)).
ukInsert	Execute the SQL (see page 10242) statement used to insert new records into the dataset (InsertSQL (see page 10240)).
ukModify	Execute the SQL (see page 10242) statement used to update records in the dataset (ModifySQL (see page 10240)).

If the statement to execute contains any parameters, an application must call SetParams (see page 10237) to bind the parameters before calling ExecSQL. To determine if a statement contains parameters, examine the appropriate ModifySQL (see page 10240), InsertSQL (see page 10240), or DeleteSQL (see page 10239) property, depending on the statement type intended for execution.

Note: To both bind parameters and execute a statement, call Apply (see page 10234).

See Also

Apply (see page 10234)

DeleteSQL (see page 10239)

InsertSQL (see page 10240)

ModifySQL (see page 10240)

SetParams (see page 10237)

Using Multiple Update Objects

53.1.20.4.3 TUpdateSQL.GetDataSet Method

GetDataSet returns the value of the DataSet property.

Pascal

```
function GetDataSet: TDataSet; override;
```

C++

```
virtual __fastcall TDataSet GetDataSet();
```

Description

Override GetDataSet, along with the SetDataSet method, to provide an implementation of the DataSet property. The DataSet property should represent the dataset that caches the updates to be applied by the update object.

See Also

DataSet (see page 9999)

SetDataSet (see page 9999)

53.1.20.4.4 TUpdateSQL.GetSQL Method

Returns the SQL statement that performs a specified update operation.

Pascal

```
function GetSQL(UpdateKind: TUpdateKind): TStrings; override;
```

C++

```
virtual __fastcall TStrings GetSQL(TUpdateKind UpdateKind);
```

Description

Override GetSQL to allow the update object to access an SQL statement that applies an update of the type specified by UpdateKind.

GetSQL provides the underlying implementation for the SQL property in the descendant class TUpdateSQL.

See Also

SQL (see page 10242)

53.1.20.4.5 TUpdateSQL.SetDataSet Method

SetDataSet sets the value of the DataSet property.

Pascal

```
procedure SetDataSet(ADataSet: TDataSet); override;
```

C++

```
virtual __fastcall SetDataSet(TDataSet ADataSet);
```

Description

Override SetDataSet, along with the GetDataSet method, to provide an implementation of the DataSet property. The DataSet property should represent the dataset that caches the updates to be applied by the update object.

See Also

DataSet (see page 9999)

GetDataSet (see page 9998)

53.1.20.4.6 SetParams Method

This is the overview for the SetParams method overload.

Overload List

	Name	Description
	TUpdateSQL.SetParams (TDataset, TUpdateKind) (see page 10237)	Binds parameters in an SQL (see page 10242) statement prior to statement execution.
	TUpdateSQL.SetParams (TUpdateKind) (see page 10237)	Binds parameters in an SQL (see page 10242) statement prior to statement execution.

53.1.20.4.6.1 TUpdateSQL.SetParams Method (TDataset, TUpdateKind)

Binds parameters in an SQL (see page 10242) statement prior to statement execution.

Pascal

```
procedure SetParams(ADataset: TDataset; UpdateKind: TUpdateKind); virtual; overload;
procedure SetParams(UpdateKind: TUpdateKind); virtual; overload;
```

C++

```
virtual __fastcall SetParams(TDataset ADataset, TUpdateKind UpdateKind);
virtual __fastcall SetParams(TUpdateKind UpdateKind);
```

Description

Call SetParams to bind parameters in an SQL (see page 10242) statement associated with the update object prior to executing the statement. UpdateKind indicates the type of statement for which to bind parameters, and can be one of the following values:

Value	Meaning
ukDelete	Bind parameters for the SQL (see page 10242) statement used to delete records (DeleteSQL (see page 10239)).
ukInsert	Bind parameters for the SQL (see page 10242) statement used to insert new records (InsertSQL (see page 10240)).
ukModify	Bind parameters for SQL (see page 10242) statement used to update records (ModifySQL (see page 10240)).

Parameters are indicated in an SQL (see page 10242) statement by a colon. Except for the leading colon in the parameter name, the parameter name must exactly match the name of an existing field name for the dataset.

Note: Parameter names can be prefaced by the 'OLD_' indicator. If so, the old value of the field is used to perform the update instead of any updates in the cache.

See Also

DeleteSQL (see page 10239)

ExecSQL (see page 10235)

InsertSQL (see page 10240)

ModifySQL (see page 10240)

53.1.20.4.7 TUpdateSQL.SQLChanged Method

Pascal

```
procedure SQLChanged(Sender: TObject);
```

C++

```
__fastcall SQLChanged(TObject * Sender);
```

Description

This is SQLChanged, a member of class TUpdateSQL.

53.1.20.5 TUpdateSQL Properties

The properties of the TUpdateSQL class are listed here.

Public Properties

	Name	Description
	DatabaseName (see page 10238)	Identifies the database to which updates are applied.
	DataSet (see page 10239)	Identifies the dataset that holds the updated data that the update object applies.
R	Query (see page 10241)	Returns the query object used to perform a specified kind of update.
	SessionName (see page 10242)	Identifies the session under which updates are applied.
	SQL (see page 10242)	Returns a specified SQL statement used when applying cached updates.

Published Properties

	Name	Description
	DeleteSQL (see page 10239)	Specifies the SQL (see page 10242) DELETE statement to use when applying a cached deletion of a record.
	InsertSQL (see page 10240)	Specifies the SQL (see page 10242) INSERT statement to use when applying a cached insertion of a record.
	ModifySQL (see page 10240)	Specifies the SQL (see page 10242) UPDATE statement to use when applying an update to a record and cached updates is enabled.

53.1.20.5.1 TUpdateSQL.DatabaseName Property

Identifies the database to which updates are applied.

Pascal

```
property DatabaseName: string;
```

C++

```
__property AnsiString DatabaseName;
```

Description

When the DataSet (see page 10239) property is not a BDE-enabled dataset, you must set DatabaseName to identify the database to which updates are applied. DatabaseName should be the same as the DatabaseName property of the source dataset (the dataset that fetches data from the database server). This property is used to set the DatabaseName property of the internal queries that apply the updates.

See Also

DatabaseName (see page 10022)

DataSet (see page 10239)

Query (see page 10241)

Using Multiple Update Objects

53.1.20.5.2 TUpdateSQL.DataSet Property

Identifies the dataset that holds the updated data that the update object applies.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

TUpdateSQL (see page 10229) uses the DataSet property for two purposes:

It fetches the original and updated field values from this dataset when performing parameter substitution.

If this dataset is a TDBDataSet (see page 10000) descendant, it uses its DatabaseName (see page 10238) and SessionName (see page 10242) properties to identify the database and session that it uses when performing its update queries.

If you are using a single update object, then the DataSet property is set automatically when you set the source dataset's UpdateObject property.

If you are using multiple update objects, you must set the DataSet property at runtime in a BeforeUpdateRecord or OnUpdateRecord event handler. If you are using the source dataset to cache its own updates, set DataSet to the source dataset in an OnUpdateRecord event handler. If you are using a client dataset to cache updates, set DataSet to the DeltaDS parameter of the BeforeUpdateRecord event handler.

See Also

UpdateObject (see page 9936)

TDBDataSet (see page 10000)

OnUpdateRecord (see page 9957)

beforeupdaterecord (see page 18442)

Using Multiple Update Objects

53.1.20.5.3 TUpdateSQL.DeleteSQL Property

Specifies the SQL (see page 10242) DELETE statement to use when applying a cached deletion of a record.

Pascal

```
property DeleteSQL: TStrings;
```

C++

```
__property TStrings DeleteSQL;
```

Description

Set DeleteSQL to the SQL (see page 10242) DELETE statement to use when applying a deletion to a record. Statements can be parameterized queries. To create a DELETE statement at design time, use the UpdateSQL editor to create statements, such as:

```
DELETE FROM "Country.db"
WHERE Name = :OLD_Name
```

At run time, an application can write a statement directly to this property to set or change the DELETE statement.

Note: As the example illustrates, DeleteSQL supports an extension to normal parameter binding. To retrieve the value of a field as it exists prior to application of cached updates, the field name with 'OLD_'. This is especially useful when doing field comparisons in the WHERE clause of the statement.

See Also

InsertSQL (see page 10240)

ModifySQL (see page 10240)

Using Update Objects to Update a Dataset

Updating a Read-only Result Set

Using TQuery

Using TUpdateSQL to Update a Dataset

53.1.20.5.4 TUpdateSQL.InsertSQL Property

Specifies the SQL (see page 10242) INSERT statement to use when applying a cached insertion of a record.

Pascal

```
property InsertSQL: TStrings;
```

C++

```
__property TStrings InsertSQL;
```

Description

Set InsertSQL to the SQL (see page 10242) INSERT statement to use when applying an insertion to a dataset. Statements can be parameterized queries. To create a INSERT statement at design time, use the UpdateSQL editor to create statements, such as:

```
INSERT INTO "Country.db"
(Name, Capital, Continent)
VALUES (:Name, :Capital, :Continent)
WHERE :OLD_Name = "Rangoon"
```

At run time, an application can write a statement directly to this property to set or change the INSERT statement.

Note: As the example illustrates, InsertSQL supports an extension to normal parameter binding. To retrieve the value of a field as it exists prior to application of cached updates, the field name with 'OLD_'. This is especially useful when doing field comparisons in the WHERE clause of the statement.

See Also

DeleteSQL (see page 10239)

ModifySQL (see page 10240)

Using Update Objects to Update a Dataset

Updating a Read-only Result Set

Using TQuery

Using TUpdateSQL to Update a Dataset

53.1.20.5.5 TUpdateSQL.ModifySQL Property

Specifies the SQL (see page 10242) UPDATE statement to use when applying an update to a record and cached updates is

enabled.

Pascal

```
property ModifySQL: TStrings;
```

C++

```
__property TStrings ModifySQL;
```

Description

Set ModifySQL to the SQL (see page 10242) UPDATE statement to use when applying an updated record to a dataset. Statements can be parameterized queries. To create a UPDATE statement at design time, use the UpdateSQL editor to create statements, such as:

```
UPDATE "Country.db"
SET Name = :Name, Capital = :Capital, Continent = :Continent
WHERE Name = :OLD_Name
```

At run time, an application can write a statement directly to this property to set or change the UPDATE statement.

Note: As the example illustrates, ModifySQL supports an extension to normal parameter binding. To retrieve the value of a field as it exists prior to application of cached updates, the field name with 'OLD_'. This is especially useful when doing field comparisons in the WHERE clause of the statement.

See Also

DeleteSQL (see page 10239)

InsertSQL (see page 10240)

Using Update Objects to Update a Dataset

Updating a Read-only Result Set

Using TUpdateSQL to Update a Dataset

53.1.20.5.6 TUpdateSQL.Query Property

Returns the query object used to perform a specified kind of update.

Pascal

```
property Query [UpdateKind: TUpdateKind]: TQuery;
```

C++

```
__property TQuery Query[TUpdateKind UpdateKind];
```

Description

Query is a read-only property that provides a reference to the internal TQuery (see page 10059) that executes the SQL (see page 10242) that applies the cached data updates. Use properties and methods of TQuery (see page 10059) to work with this reference.

Using one of the TUpdateKind constants as an index for the Query property, the internal TQuery (see page 10059) object will have the SQL (see page 10242) specified in the corresponding update SQL (see page 10242) property: DeleteSQL (see page 10239), InsertSQL (see page 10240), or ModifySQL (see page 10240). UpdateKind can be one of the following:

Value	Meaning
ukDelete	Return the query object used to execute DELETE statements (DeleteSQL (see page 10239)).
ukInsert	Return the query object used to execute INSERT statements (InsertSQL (see page 10240)).
ukModify	Return the query object used to execute UPDATE statements (ModifySQL (see page 10240)).

Each query object executes a particular kind of SQL (see page 10242) statement. The contents of the SQL (see page 10242) statements executed by these objects can be accessed directly using the ModifySQL (see page 10240), InsertSQL (see page 10240), and DeleteSQL (see page 10239) properties.

The main purpose of Query is to provide a a way for an application to set the properties for an update query object or to call the query object's methods.

Note: If a particular kind of update statement is not provided, then its corresponding query object is nil (Delphi) or NULL (C++). For example, if an application does not provide an SQL (see page 10242) statement for the DeleteSQL (see page 10239) property, then setting Query[ukDelete] returns nil (Delphi) or NULL (C++).

See Also

DeleteSQL (see page 10239)

InsertSQL (see page 10240)

ModifySQL (see page 10240)

53.1.20.5.7 TUpdateSQL.SessionName Property

Identifies the session under which updates are applied.

Pascal

```
property SessionName: string;
```

C++

```
__property AnsiString SessionName;
```

Description

When the DataSet (see page 10239) property is not a BDE-enabled dataset, you must set SessionName to identify the session under which updates are applied. SessionName should be the same as the SessionName property of the source dataset (the dataset that fetches data from the database server). This property is used to set the SessionName property of the internal queries that apply the updates.

See Also

SessionName (see page 10027)

DataSet (see page 10239)

Query (see page 10241)

Using Multiple Update Objects

53.1.20.5.8 TUpdateSQL.SQL Property

Returns a specified SQL statement used when applying cached updates.

Pascal

```
property SQL [UpdateKind: TUpdateKind]: TStrings;
```

C++

```
__property TStrings SQL[TUpdateKind UpdateKind];
```

Description

Returns the SQL statement in the ModifySQL (see page 10240), InsertSQL (see page 10240), or DeleteSQL (see page 10239) property, depending on the setting of the UpdateKind index. UpdateKind can be any of the following:

Value	Meaning
ukDelete	Return the query object used to execute DELETE statements (DeleteSQL (see page 10239)).
ukInsert	Return the query object used to execute INSERT statements (InsertSQL (see page 10240)).
ukModify	Return the query object used to execute UPDATE statements (ModifySQL (see page 10240)).

See Also

DeleteSQL (see page 10239)

InsertSQL (see page 10240)

ModifySQL (see page 10240)

53.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AnsiToNative (see page 10244)	Converts a string from the ANSI character set to the character set associated with a given locale.
	AnsiToNativeBuf (see page 10245)	Converts a string from the ANSI character set to the character set associated with a given locale.
	Check (see page 10245)	Determines whether a value returned from the Borland (see page 1761) Database Engine (BDE) represents an error condition.
	DbiError (see page 10246)	Creates and raises an EDBEngineError (see page 9854) exception for an error code returned by the Borland (see page 1761) Database Engine (BDE).
	GetFieldSource (see page 10246)	This is function DBTables.GetFieldSource.
	NativeCompareStr (see page 10246)	Compares strings based on a database locale case sensitively.
	NativeCompareStrBuf (see page 10247)	Compares null-terminated strings based on a database locale case sensitively.
	NativeCompareText (see page 10248)	Compares strings based on a database locale without case sensitivity.
	NativeCompareTextBuf (see page 10248)	Compares null-terminated strings based on a database locale and is not case sensitive.
	NativeToAnsi (see page 10249)	Converts a string from the character set of the given locale to the ANSI character set.
	NativeToAnsiBuf (see page 10249)	Converts a string from the character set of a specified locale into the ANSI character set.
	RegisterBDEInitProc (see page 10250)	This is function DBTables.RegisterBDEInitProc.

53.2.1 DBTables.AnsiToNative Function

Converts a string from the ANSI character set to the character set associated with a given locale.

Pascal

```
function AnsiToNative(Locale: TLocale; const AnsiStr: string; NativeStr: PChar; MaxLen:
Integer): PChar;
```

C++

```
const char * AnsiToNative(TLocale Locale, const AnsiString AnsiStr, const char * NativeStr,
int MaxLen);
```

File

DBTables

Description

Call AnsiToNative to convert a string represented in the ANSI character set (used internally) to the character set associated with the database locale specified by the Locale parameter. The resulting string is copied into the buffer pointed to by the NativeStr parameter. Use the MaxLen parameter to specify the size of this buffer. If the buffer is too small to contain all characters in the converted version of AnsiStr, the returned value is truncated. AnsiToNative returns a pointer to NativeStr.

Use AnsiToNative to convert strings typed by the user into the character set used by a database table.

See Also

AnsiToNativeBuf

NativeToAnsi

53.2.2 DBTables.AnsiToNativeBuf Function

Converts a string from the ANSI character set to the character set associated with a given locale.

Pascal

```
procedure AnsiToNativeBuf(Locale: TLocale; Source: PChar; Dest: PChar; Len: Integer);
```

C++

```
AnsiToNativeBuf(TLocale Locale, const char * Source, const char * Dest, int Len);
```

File

DBTables

Description

Call AnsiToNativeBuf to convert a null-terminated string in the ANSI character set (used internally by Windows) to the character set associated with the database locale specified by the Locale parameter. The resulting string is copied into the buffer pointed to by the Dest parameter. Use the Len parameter to specify the size of this buffer.

Use AnsiToNativeBuf to convert strings typed by the user into the character set used by a database table.

See Also

AnsiToNative

NativeToAnsiBuf

TLocale

53.2.3 DBTables.Check Function

Determines whether a value returned from the Borland (see page 1761) Database Engine (BDE) represents an error condition.

Pascal

```
procedure Check(Status: DBIResult);
```

C++

```
Check(DBIResult Status);
```

File

DBTables

Description

Pass the return code from a direct BDE API call as the Status parameter when calling Check. Check determines whether Status indicates an error condition, and if so, calls DbiError (see page 10246) to raise an exception.

See Also

DbiError

53.2.4 DBTables.DbiError Function

Creates and raises an EDBEngineError (see page 9854) exception for an error code returned by the Borland (see page 1761) Database Engine (BDE).

Pascal

```
procedure DbiError(ErrorCode: DBIResult);
```

C++

```
DbiError(DBIResult ErrorCode);
```

File

DBTables

Description

Call DbiError to raise an EDBEngineError (see page 9854) for the error code passed as the ErrorCode parameter. DbiError obtains the error string from the BDE for the error and raises an EDBEngineError (see page 9854) exception.

See Also

Check

DatabaseError

EDBEngineError

53.2.5 DBTables.GetFieldSource Function

Pascal

```
function GetFieldSource(ADataSet: TDataSet; var ADataSources: DataSources): Boolean;
```

C++

```
Boolean GetFieldSource(TDataSet ADataSet, DataSources ADataSources);
```

File

DBTables

Description

This is function DBTables.GetFieldSource.

53.2.6 DBTables.NativeCompareStr Function

Compares strings based on a database locale case sensitively.

Pascal

```
function NativeCompareStr(Locale: TLocale; const S1: string; const S2: string; Len: Integer):
Integer;
```

C++

```
int NativeCompareStr(TLocale Locale, const AnsiString S1, const AnsiString S2, int Len);
```

File

DBTables

Description

NativeCompareStr compares S1 to S2, up to a length of Len characters. If the Len parameter is 0, NativeCompareStr compares the strings up to their entire length. The comparison is performed with case sensitivity. The compare operation is controlled by the database locale specified as the Locale parameter. The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

See Also

NativeCompareText

NativeCompareStrBuf

53.2.7 DBTables.NativeCompareStrBuf Function

Compares null-terminated strings based on a database locale case sensitively.

Pascal

```
function NativeCompareStrBuf(Locale: TLocale; const S1: PChar; const S2: PChar; Len: Integer):
Integer;
```

C++

```
int NativeCompareStrBuf(TLocale Locale, const const char * S1, const const char * S2, int Len);
```

File

DBTables

Description

NativeCompareStrBuf compares up to the first Len characters of S1 to those of S2, with case sensitivity. If the Len parameter is 0, NativeCompareStrBuf compares the strings up to their entire length. The compare operation is controlled by the database locale specified as the Locale parameter. The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

See Also

NativeCompareText

NativeCompareStr

53.2.8 DBTables.NativeCompareText Function

Compares strings based on a database locale without case sensitivity.

Pascal

```
function NativeCompareText(Locale: TLocale; const S1: string; const S2: string; Len: Integer):
Integer;
```

C++

```
int NativeCompareText(TLocale Locale, const AnsiString S1, const AnsiString S2, int Len);
```

File

DBTables

Description

NativeCompareText compares up to the first Len characters of S1 to those of S2, without case sensitivity. If the Len parameter is 0, NativeCompareText compares the strings up to their entire length. The compare operation is controlled by the database locale specified as the Locale parameter. NativeCompareText returns a value less than 0 if S1 < S2, a value greater than 0 if S1 > S2, and returns 0 if S1 = S2.

See Also

NativeCompareStr

NativeCompareTextBuf

53.2.9 DBTables.NativeCompareTextBuf Function

Compares null-terminated strings based on a database locale and is not case sensitive.

Pascal

```
function NativeCompareTextBuf(Locale: TLocale; const S1: PChar; const S2: PChar; Len:
Integer): Integer;
```

C++

```
int NativeCompareTextBuf(TLocale Locale, const const char * S1, const const char * S2, int
Len);
```

File

DBTables

Description

NativeCompareTextBuf compares up to the first Len characters of S1 to those of S2, without case sensitivity. If the Len parameter is 0, NativeCompareTextBuf compares the strings up to their entire length. The compare operation is controlled by the database locale specified as the Locale parameter. NativeCompareTextBuf returns a value less than 0 if S1 < S2, a value greater than 0 if S1 > S2, and returns 0 if S1 = S2.

See Also

NativeCompareStr

NativeCompareText

53.2.10 DBTables.NativeToAnsi Function

Converts a string from the character set of the given locale to the ANSI character set.

Pascal

```
procedure NativeToAnsi(Locale: TLocale; NativeStr: PChar; var AnsiStr: string);
```

C++

```
NativeToAnsi(TLocale Locale, const char * NativeStr, AnsiString AnsiStr);
```

File

DBTables

Description

Call NativeToAnsi to convert a string represented in the character set of the Locale parameter to the ANSI character set (used internally). The resulting string is returned as the AnsiStr parameter.

Use NativeToAnsi to convert strings retrieved from a database table into strings that can be displayed in the user interface.

See Also

AnsiToNative

NativeToAnsiBuf

53.2.11 DBTables.NativeToAnsiBuf Function

Converts a string from the character set of a specified locale into the ANSI character set.

Pascal

```
procedure NativeToAnsiBuf(Locale: TLocale; Source: PChar; Dest: PChar; Len: Integer);
```

C++

```
NativeToAnsiBuf(TLocale Locale, const char * Source, const char * Dest, int Len);
```

File

DBTables

Description

Call NativeToAnsiBuf to convert a null-terminated string in the character set of the Locale parameter to the ANSI character set (used internally). The resulting string is copied into the buffer pointed to by the Dest parameter. Use the Len parameter to specify the size of this buffer.

Use NativeToAnsiBuf to convert strings from a database table into the ANSI character set so that they can be displayed in the user interface.

See Also

AnsiToNativeBuf

NativeToAnsi

53.2.12 DBTables.RegisterBDEInitProc Function

Pascal

```
procedure RegisterBDEInitProc(const InitProc: TBDEInitProc);
```

C++

```
RegisterBDEInitProc(const TBDEInitProc InitProc);
```

File

DBTables

Description

This is function DBTables.RegisterBDEInitProc.

53.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TBatchMode (see page 10251)	TBatchMode indicates the type of batch operation to be performed on a database table.
	TConfigModes (see page 10253)	This is record DBTables.TConfigModes.
	TDatabaseEvent (see page 10253)	This is record DBTables.TDatabaseEvent.
	TLockType (see page 10254)	Indicates how a Paradox or dBASE table can be locked.
	TParamBindMode (see page 10254)	This is record DBTables.TParamBindMode.
	TRecNoStatus (see page 10255)	This is record DBTables.TRecNoStatus.
	TTableType (see page 10256)	This is record DBTables.TTableType.
	TTraceFlag (see page 10256)	This is record DBTables.TTraceFlag.
	TTransIsolation (see page 10257)	This is record DBTables.TTransIsolation.

Records

	Name	Description
	TBDEKeyBuffer (see page 10252)	This is record DBTables.TBDEKeyBuffer.
	TBDERecInfo (see page 10252)	This is record DBTables.TBDERecInfo.
	TServerDesc (see page 10255)	This is record DBTables.TServerDesc.

53.3.1 DBTables.TBatchMode Enumeration

TBatchMode indicates the type of batch operation to be performed on a database table.

Pascal

```
TBatchMode = (
  batAppend,
  batUpdate,
  batAppendUpdate,
  batDelete,
  batCopy
);
```

C++

```
enum TBatchMode {
  batAppend,
  batUpdate,
  batAppendUpdate,
  batDelete,
  batCopy
};
```

File

DBTables

Description

TBatchMode can take the following values:

Value	Meaning
batAppend	Append all records from the source table to the end of destination table. The destination and source tables must not have records with duplicate keys.
batAppendUpdate	If a matching record exists in the destination table, replace it with the corresponding record from the source table. Otherwise, append records to the destination table. The destination table must have an index defined for matching records.
batCopy	Copy the structure and records from the source table into the destination table. If the destination table already exists, delete it and replace it with the new copy of the source.
batDelete	Delete all records in the destination table that also appear in the source table.
batUpdate	If a matching record exists in the destination table, replace it with the corresponding record from the source table. The destination table must have an index defined for matching records.

53.3.2 DBTables.TBDEKeyBuffer Record

Pascal

```
TBDEKeyBuffer = packed record
  Modified: Boolean;
  Exclusive: Boolean;
  FieldCount: Integer;
end;
```

C++

```
struct TBDEKeyBuffer {
  Boolean Modified;
  Boolean Exclusive;
  int FieldCount;
};
```

File

DBTables

Description

This is record DBTables.TBDEKeyBuffer.

53.3.3 DBTables.TBDERecInfo Record

Pascal

```
TBDERecInfo = packed record
  RecordNumber: Longint;
  UpdateStatus: TUpdateStatus;
  BookmarkFlag: TBookmarkFlag;
end;
```

C++

```
struct TBDERecInfo {
  Longint RecordNumber;
  TUpdateStatus UpdateStatus;
  TBookmarkFlag BookmarkFlag;
};
```

File

DBTables

Description

This is record DBTables.TBDERecInfo.

53.3.4 DBTables.TConfigModes Enumeration

Pascal

```
TConfigModes = (
  cfmVirtual,
  cfmPersistent,
  cfmSession
);
```

C++

```
enum TConfigModes {
  cfmVirtual,
  cfmPersistent,
  cfmSession
};
```

File

DBTables

Description

This is record DBTables.TConfigModes.

53.3.5 DBTables.TDatabaseEvent Enumeration

Pascal

```
TDatabaseEvent = (
  dbOpen,
  dbClose,
  dbAdd,
  dbRemove,
  dbAddAlias,
  dbDeleteAlias,
  dbAddDriver,
  dbDeleteDriver
);
```

C++

```
enum TDatabaseEvent {
  dbOpen,
  dbClose,
  dbAdd,
  dbRemove,
  dbAddAlias,
  dbDeleteAlias,
  dbAddDriver,
  dbDeleteDriver
};
```

File

DBTables

Description

This is record DBTables.TDatabaseEvent.

53.3.6 DBTables.TLockType Enumeration

Indicates how a Paradox or dBASE table can be locked.

Pascal

```
TLockType = (
  ltReadLock,
  ltWriteLock
);
```

C++

```
enum TLockType {
  ltReadLock,
  ltWriteLock
};
```

File

DBTables

Description

TLockType identifies the type of a BDE lock on a Paradox or dBASE table. The following table lists the possible values:

Value	Description
ltReadLock	Other applications (or sessions) can not read from the table.
ltWriteLock	Other applications (or sessions) can not write to the table.'

53.3.7 DBTables.TParamBindMode Enumeration

Pascal

```
TParamBindMode = (
  pbByName,
  pbByNumber
);
```

C++

```
enum TParamBindMode {
  pbByName,
  pbByNumber
};
```

File

DBTables

Description

This is record DBTables.TParamBindMode.

53.3.8 DBTables.TRecNoStatus Enumeration

Pascal

```
TRecNoStatus = (
  rnDbase,
  rnParadox,
  rnNotSupported
);
```

C++

```
enum TRecNoStatus {
  rnDbase,
  rnParadox,
  rnNotSupported
};
```

File

DBTables

Description

This is record DBTables.TRecNoStatus.

53.3.9 DBTables.TServerDesc Record

Pascal

```
TServerDesc = record
  ParamName: string[DBIMAXSPNAMELEN];
  BindType: TFieldType;
end;
```

C++

```
struct TServerDesc {
  string[DBIMAXSPNAMELEN] ParamName;
  TFieldType BindType;
};
```

File

DBTables

Description

This is record DBTables.TServerDesc.

53.3.10 DBTables.TTableType Enumeration

Pascal

```
TTableType = (
  ttDefault,
  ttParadox,
  ttDBase,
  ttFoxPro,
  ttASCII
);
```

C++

```
enum TTableType {
  ttDefault,
  ttParadox,
  ttDBase,
  ttFoxPro,
  ttASCII
};
```

File

DBTables

Description

This is record DBTables.TTableType.

53.3.11 DBTables.TTraceFlag Enumeration

Pascal

```
TTraceFlag = (
  tfQPrepare,
  tfQExecute,
  tfError,
  tfStmt,
  tfConnect,
  tfTransact,
  tfBlob,
  tfMisc,
  tfVendor,
  tfDataIn,
  tfDataOut
);
```

C++

```
enum TTraceFlag {
  tfQPrepare,
  tfQExecute,
  tfError,
  tfStmt,
  tfConnect,
  tfTransact,
  tfBlob,
  tfMisc,
  tfVendor,
```

```
    tfDataIn,
    tfDataOut
  };
```

File

DBTables

Description

This is record DBTables.TTraceFlag.

53.3.12 DBTables.TTransIsolation Enumeration

Pascal

```
  TTransIsolation = (
    tiDirtyRead,
    tiReadCommitted,
    tiRepeatableRead
  );
```

C++

```
  enum TTransIsolation {
    tiDirtyRead,
    tiReadCommitted,
    tiRepeatableRead
  };
```

File

DBTables

Description

This is record DBTables.TTransIsolation.

53.4 Types

The following table lists types in this documentation.

Types

Name	Description
PKeyBuffer (see page 10258)	This is type DBTables.PKeyBuffer.
PRecInfo (see page 10258)	This is type DBTables.PRecInfo.
TBDECallbackEvent (see page 10259)	This is type DBTables.TBDECallbackEvent.
TBDEInitProc (see page 10259)	This is type DBTables.TBDEInitProc.
TBlobDataArray (see page 10259)	This is type DBTables.TBlobDataArray.
TConfigMode (see page 10260)	This is type DBTables.TConfigMode.
TDatabaseLoginEvent (see page 10260)	This is type DBTables.TDatabaseLoginEvent.
TDatabaseNotifyEvent (see page 10260)	This is type DBTables.TDatabaseNotifyEvent.
TDBFlags (see page 10261)	This is type DBTables.TDBFlags.
TFieldDescList (see page 10261)	This is type DBTables.TFieldDescList.
TIndexDescList (see page 10261)	This is type DBTables.TIndexDescList.
TIndexName (see page 10262)	This is type DBTables.TIndexName.
TLocale (see page 10262)	TLocale is the type for the Locale or DBLocale property.
TPasswordEvent (see page 10262)	This is type DBTables.TPasswordEvent.
TServerDescList (see page 10263)	This is type DBTables.TServerDescList.
TSPParamDescList (see page 10263)	This is type DBTables.TSPParamDescList.
TTraceFlags (see page 10263)	TTraceFlags and TTraceFlags indicate the types of messages that pass between an application and the InterBase server.
TValCheckList (see page 10264)	This is type DBTables.TValCheckList.

53.4.1 DBTables.PKeyBuffer Type

Pascal

```
PKeyBuffer = ^TBDEKeyBuffer;
```

C++

```
^TBDEKeyBuffer PKeyBuffer;
```

File

DBTables

Description

This is type DBTables.PKeyBuffer.

53.4.2 DBTables.PRecInfo Type

Pascal

```
PRecInfo = ^TBDERecInfo;
```

C++

```
^TBDERecInfo PRecInfo;
```

File

DBTables

Description

This is type DBTables.PRecInfo.

53.4.3 DBTables.TBDECallbackEvent Type

Pascal

```
TBDECallbackEvent = function (CBInfo: Pointer): CBRType of object;
```

C++

```
(CBInfo: Pointer): CBRType ( TBDECallbackEvent)();
```

File

DBTables

Description

This is type DBTables.TBDECallbackEvent.

53.4.4 DBTables.TBDEInitProc Type

Pascal

```
TBDEInitProc = procedure (Session: TSession);
```

C++

```
(Session: TSession) ( TBDEInitProc)();
```

File

DBTables

Description

This is type DBTables.TBDEInitProc.

53.4.5 DBTables.TBlobDataArray Type

Pascal

```
TBlobDataArray = array of TBlobData;
```

C++

```
array of TBlobData TBlobDataArray;
```

File

DBTables

Description

This is type DBTables.TBlobDataArray.

53.4.6 DBTables.TConfigMode Type

Pascal

```
TConfigMode = set of TConfigModes;
```

C++

```
TConfigModes TConfigMode;
```

File

DBTables

Description

This is type DBTables.TConfigMode.

53.4.7 DBTables.TDatabaseLoginEvent Type

Pascal

```
TDatabaseLoginEvent = procedure (Database: TDatabase; LoginParams: TStrings) of object;
```

C++

```
(Database: TDatabase; LoginParams: TStrings) ( TDatabaseLoginEvent)();
```

File

DBTables

Description

This is type DBTables.TDatabaseLoginEvent.

53.4.8 DBTables.TDatabaseNotifyEvent Type

Pascal

```
TDatabaseNotifyEvent = procedure (DBEvent: TDatabaseEvent; const Param) of object;
```

C++

```
(DBEvent: TDatabaseEvent; const Param) ( TDatabaseNotifyEvent)();
```

File

DBTables

Description

This is type DBTables.TDatabaseNotifyEvent.

53.4.9 DBTables.TDBFlags Type

Pascal

```
TDBFlags = set of 0..15;
```

C++

```
0..15 TDBFlags;
```

File

DBTables

Description

This is type DBTables.TDBFlags.

53.4.10 DBTables.TFieldDescList Type

Pascal

```
TFieldDescList = array of BDEFLDDesc;
```

C++

```
array of BDEFLDDesc TFieldDescList;
```

File

DBTables

Description

This is type DBTables.TFieldDescList.

53.4.11 DBTables.TIndexDescList Type

Pascal

```
TIndexDescList = array of IDXDesc;
```

C++

```
array of IDXDesc TIndexDescList;
```

File

DBTables

Description

This is type DBTables.TIndexDescList.

53.4.12 DBTables.TIndexName Type

Pascal

```
TIndexName = type string;
```

C++

```
type string TIndexName;
```

File

DBTables

Description

This is type DBTables.TIndexName.

53.4.13 DBTables.TLocale Type

TLocale is the type for the Locale or DBLocale property.

Pascal

```
TLocale = Pointer;
```

C++

```
void * TLocale;
```

File

DBTables

Description

The TLocale type is the type of a Locale or DBLocale property. It is used for direct calls to BDE. These properties are only used or needed when making direct calls to the Borland (see page 1761) Database Engine.

Note: Do not confuse TLocale with the TSysLocale type that represents a Windows locale. TLocale is specific to the Borland (see page 1761) Database Engine.

See Also

TSysLocale

53.4.14 DBTables.TPasswordEvent Type

Pascal

```
TPasswordEvent = procedure (Sender: TObject; var Continue: Boolean) of object;
```

C++

```
(Sender: TObject; var Continue: Boolean) ( TPasswordEvent)();
```

File

DBTables

Description

This is type DBTables.TPasswordEvent.

53.4.15 DBTables.TServerDescList Type

Pascal

```
TServerDescList = array of TServerDesc;
```

C++

```
array of TServerDesc TServerDescList;
```

File

DBTables

Description

This is type DBTables.TServerDescList.

53.4.16 DBTables.TSPParamDescList Type

Pascal

```
TSPParamDescList = array of BDESPParamDesc;
```

C++

```
array of BDESPParamDesc TSPParamDescList;
```

File

DBTables

Description

This is type DBTables.TSPParamDescList.

53.4.17 DBTables.TTraceFlags Type

TTraceFlags and TTraceFlags indicate the types of messages that pass between an application and the InterBase server.

Pascal

```
TTraceFlags = set of TTraceFlag;
```

C++

```
TTraceFlag TTraceFlags;
```

File

DBTables

Description

TTraceFlags is a set of TTraceFlag (see page 10256) values. The following table lists the possible values:

Value	Meaning
tfQPrepare	Monitor Prepare statements.
tfQExecute	Monitor ExecSQL statements.
tfError	Monitor server error messages. Such messages may include an error code.
tfStmt	Monitor all SQL statements.
tfConnect	Monitor database connect and disconnect operations, including allocation of connection handles, and freeing connection handles.
tfTransact	Monitor transaction statements, such as StartTransaction, Commit, and Rollback.
tfBlob	Monitor operations on blob data types.
tfService	Monitor messages to one of the IB services.
tfMisc	Monitor any statements not covered by other flag options.

53.4.18 DBTables.TValCheckList Type

Pascal

```
TValCheckList = array of VCHKDesc;
```

C++

```
array of VCHKDesc TValCheckList;
```

File

DBTables

Description

This is type DBTables.TValCheckList.

53.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
GetObjectContextProc (see page 10265)	This is variable DBTables.GetObjectContextProc.
Session (see page 10265)	Maintains database components used by an application.
Sessions (see page 10266)	Provides access to TSessionList (see page 10129) methods and properties.

53.5.1 DBTables.GetObjectContextProc Variable

Pascal

```
GetObjectContextProc: function: IUnknown;
```

C++

```
function: IUnknown GetObjectContextProc;
```

File

DBTables

Description

This is variable DBTables.GetObjectContextProc.

53.5.2 DBTables.Session Variable

Maintains database components used by an application.

Pascal

```
Session: TSession;
```

C++

```
TSession Session;
```

File

DBTables

Description

Session is responsible for maintaining all of the database components used by an application. It is created automatically as part of an application's initialization and destroyed as part of the application's termination. The Session variable must remain active at all times; it can not be destroyed and recreated.

See Also

Sessions

53.5.3 DBTables.Sessions Variable

Provides access to TSessionList (see page 10129) methods and properties.

Pascal

```
Sessions: TSessionList;
```

C++

```
TSessionList Sessions;
```

File

DBTables

Description

Sessions is a way to access TSessionList (see page 10129) methods and properties for manipulating sessions. The DB (see page 6725) unit automatically creates and destroys Sessions for you; you don't need to do anything but use it.

The Session (see page 10265) variable, on the other hand, is responsible for maintaining all of the database components used by your application. It is created automatically as part of your application's initialization and destroyed as part of your application's termination. The Session (see page 10265) variable must remain active at all times; it can not be destroyed and recreated.

See Also

Session

TSession

53.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
BlobTypeMap (see page 10267)	This is constant DBTables.BlobTypeMap.
cmAll (see page 10268)	This is constant DBTables.cmAll.
cmPersistent (see page 10268)	This is constant DBTables.cmPersistent.
cmSession (see page 10268)	This is constant DBTables.cmSession.
cmVirtual (see page 10269)	This is constant DBTables.cmVirtual.
DataTypeMap (see page 10269)	This is constant DBTables.DataTypeMap.
dbfDatabase (see page 10269)	This is constant DBTables.dbfDatabase.
dbfExecProc (see page 10270)	This is constant DBTables.dbfExecProc.
dbfExecSQL (see page 10270)	This is constant DBTables.dbfExecSQL.
dbfFieldList (see page 10270)	This is constant DBTables.dbfFieldList.
dbfIndexList (see page 10271)	This is constant DBTables.dbfIndexList.
dbfOpened (see page 10271)	This is constant DBTables.dbfOpened.
dbfPrepared (see page 10271)	This is constant DBTables.dbfPrepared.
dbfProcDesc (see page 10272)	This is constant DBTables.dbfProcDesc.
dbfProvider (see page 10272)	This is constant DBTables.dbfProvider.
dbfStoredProc (see page 10272)	This is constant DBTables.dbfStoredProc.
dbfTable (see page 10273)	This is constant DBTables.dbfTable.
FldSubTypeMap (see page 10273)	This is constant DBTables.FldSubTypeMap.
FldTypeMap (see page 10273)	This is constant DBTables.FldTypeMap.
smTraceBufSize (see page 10274)	This is constant DBTables.smTraceBufSize.

53.6.1 DBTables.BlobTypeMap Constant

Pascal

```
BlobTypeMap: array[fldstMEMO..fldstBFILE] of TFieldType = ( ftMemo, ftBlob, ftFmtMemo,
ftParadoxOle, ftGraphic, ftDBaseOle, ftTypedBinary, ftBlob, ftBlob, ftBlob, ftBlob, ftOraClob,
ftOraBlob, ftBlob, ftBlob);
```

C++

```
array[fldstMEMO..fldstBFILE] of TFieldType BlobTypeMap = ( ftMemo, ftBlob, ftFmtMemo,
ftParadoxOle, ftGraphic, ftDBaseOle, ftTypedBinary, ftBlob, ftBlob, ftBlob, ftBlob, ftOraClob,
ftOraBlob, ftBlob, ftBlob);
```

File

DBTables

Description

This is constant DBTables.BlobTypeMap.

53.6.2 DBTables.cmAll Constant

Pascal

```
cmAll = [cfmVirtual, cfmPersistent, cfmSession];
```

C++

```
cmAll = [cfmVirtual, cfmPersistent, cfmSession];
```

File

DBTables

Description

This is constant DBTables.cmAll.

53.6.3 DBTables.cmPersistent Constant

Pascal

```
cmPersistent = [cfmPersistent];
```

C++

```
cmPersistent = [cfmPersistent];
```

File

DBTables

Description

This is constant DBTables.cmPersistent.

53.6.4 DBTables.cmSession Constant

Pascal

```
cmSession = [cfmSession];
```

C++

```
cmSession = [cfmSession];
```

File

DBTables

Description

This is constant DBTables.cmSession.

53.6.5 DBTables.cmVirtual Constant

Pascal

```
cmVirtual = [cfmVirtual];
```

C++

```
cmVirtual = [cfmVirtual];
```

File

DBTables

Description

This is constant DBTables.cmVirtual.

53.6.6 DBTables.DataTypeMap Constant

Pascal

```
DataTypeMap: array[0..MAXLOGFLDTYPES - 1] of TFieldType = ( ftUnknown, ftString, ftDate,
ftBlob, ftBoolean, ftSmallint, ftInteger, ftFloat, ftBCD, ftBytes, ftTime, ftDateTime, ftWord,
ftInteger, ftUnknown, ftVarBytes, ftUnknown, ftUnknown, ftLargeInt, ftLargeInt, ftADT,
ftArray, ftReference, ftDataSet, ftTimeStamp);
```

C++

```
array[0..MAXLOGFLDTYPES - 1] of TFieldType DataTypeMap = ( ftUnknown, ftString, ftDate,
ftBlob, ftBoolean, ftSmallint, ftInteger, ftFloat, ftBCD, ftBytes, ftTime, ftDateTime, ftWord,
ftInteger, ftUnknown, ftVarBytes, ftUnknown, ftUnknown, ftLargeInt, ftLargeInt, ftADT,
ftArray, ftReference, ftDataSet, ftTimeStamp);
```

File

DBTables

Description

This is constant DBTables.DataTypeMap.

53.6.7 DBTables.dbfDatabase Constant

Pascal

```
dbfDatabase = 9;
```

C++

```
dbfDatabase = 9;
```

File

DBTables

Description

This is constant DBTables.dbfDatabase.

53.6.8 DBTables.dbfExecProc Constant

Pascal

```
dbfExecProc = 7;
```

C++

```
dbfExecProc = 7;
```

File

DBTables

Description

This is constant DBTables.dbfExecProc.

53.6.9 DBTables.dbfExecSQL Constant

Pascal

```
dbfExecSQL = 2;
```

C++

```
dbfExecSQL = 2;
```

File

DBTables

Description

This is constant DBTables.dbfExecSQL.

53.6.10 DBTables.dbfFieldList Constant

Pascal

```
dbfFieldList = 4;
```

C++

```
dbfFieldList = 4;
```

File

DBTables

Description

This is constant DBTables.dbfFieldList.

53.6.11 DBTables.dbfIndexList Constant

Pascal

```
dbfIndexList = 5;
```

C++

```
dbfIndexList = 5;
```

File

DBTables

Description

This is constant DBTables.dbfIndexList.

53.6.12 DBTables.dbfOpened Constant

Pascal

```
dbfOpened = 0;
```

C++

```
dbfOpened = 0;
```

File

DBTables

Description

This is constant DBTables.dbfOpened.

53.6.13 DBTables.dbfPrepared Constant

Pascal

```
dbfPrepared = 1;
```

C++

```
dbfPrepared = 1;
```

File

DBTables

Description

This is constant DBTables.dbfPrepared.

53.6.14 DBTables.dbfProcDesc Constant

Pascal

```
dbfProcDesc = 8;
```

C++

```
dbfProcDesc = 8;
```

File

DBTables

Description

This is constant DBTables.dbfProcDesc.

53.6.15 DBTables.dbfProvider Constant

Pascal

```
dbfProvider = 10;
```

C++

```
dbfProvider = 10;
```

File

DBTables

Description

This is constant DBTables.dbfProvider.

53.6.16 DBTables.dbfStoredProc Constant

Pascal

```
dbfStoredProc = 6;
```

C++

```
dbfStoredProc = 6;
```

File

DBTables

Description

This is constant DBTables.dbfStoredProc.

53.6.17 DBTables.dbfTable Constant

Pascal

```
dbfTable = 3;
```

C++

```
dbfTable = 3;
```

File

DBTables

Description

This is constant DBTables.dbfTable.

53.6.18 DBTables.FldSubTypeMap Constant

Pascal

```
FldSubTypeMap: array[TFieldType] of Word = ( 0, 0, 0, 0, 0, 0, 0, fldstMONEY, 0, 0, 0, 0, 0,
0, fldstAUTOINC, fldstBINARY, fldstMEMO, fldstGRAPHIC, fldstFMTMEMO, fldstOLEOBJ,
fldstDBSOLEOBJ, fldstTYPEDBINARY, 0, fldstFIXED, fldstUNICODE, 0, 0, 0, 0, 0, fldstHBINARY,
fldstHMEMO, 0, 0, 0, 0, 0, 0, fldstFIXED, fldstMEMO, 0, 0 );
```

C++

```
array[TFieldType] of Word FldSubTypeMap = ( 0, 0, 0, 0, 0, 0, 0, fldstMONEY, 0, 0, 0, 0, 0, 0,
fldstAUTOINC, fldstBINARY, fldstMEMO, fldstGRAPHIC, fldstFMTMEMO, fldstOLEOBJ, fldstDBSOLEOBJ,
fldstTYPEDBINARY, 0, fldstFIXED, fldstUNICODE, 0, 0, 0, 0, 0, fldstHBINARY, fldstHMEMO, 0, 0,
0, 0, 0, 0, fldstFIXED, fldstMEMO, 0, 0 );
```

File

DBTables

Description

This is constant DBTables.FldSubTypeMap.

53.6.19 DBTables.FldTypeMap Constant

Pascal

```
FldTypeMap: TFieldMap = ( fldUNKNOWN, fldZSTRING, fldINT16, fldINT32, fldUINT16, fldBOOL,
fldFLOAT, fldFLOAT, fldBCD, fldDATE, fldTIME, fldTIMESTAMP, fldBYTES, fldVARBYTES, fldINT32,
fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldCURSOR, fldZSTRING,
fldZSTRING, fldINT64, fldADT, fldArray, fldREF, fldTABLE, fldBLOB, fldBLOB, fldUNKNOWN,
fldUNKNOWN, fldUNKNOWN, fldZSTRING, fldDATETIME, fldBCD, fldZSTRING, fldBLOB, fldDATETIME,
fldINT32);
```

C++

```
TFieldMap FldTypeMap = ( fldUNKNOWN, fldZSTRING, fldINT16, fldINT32, fldUINT16, fldBOOL,
fldFLOAT, fldFLOAT, fldBCD, fldDATE, fldTIME, fldTIMESTAMP, fldBYTES, fldVARBYTES, fldINT32,
fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldCURSOR, fldZSTRING,
```

```
fldZSTRING, fldINT64, fldADT, fldArray, fldREF, fldTABLE, fldBLOB, fldBLOB, fldUNKNOWN,
fldUNKNOWN, fldUNKNOWN, fldZSTRING, fldDATETIME, fldBCD, fldZSTRING, fldBLOB, fldDATETIME,
fldINT32);
```

File

DBTables

Description

This is constant DBTables.FldTypeMap.

53.6.20 DBTables.smTraceBufSize Constant

Pascal

```
smTraceBufSize = 32767 + SizeOf(TraceDesc);
```

C++

```
smTraceBufSize = 32767 + SizeOf(TraceDesc);
```

File

DBTables

Description

This is constant DBTables.smTraceBufSize.

54 DBWeb Namespace

This is namespace DBWeb.

Topics

Name	Description
Classes (see page 10277)	The following table lists classes in this documentation.
Functions (see page 10346)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 10347)	The following table lists structs, records, enums in this documentation.
Types (see page 10349)	The following table lists types in this documentation.

Classes

	Name	Description
	TDataSetTableProducer (see page 10277)	TDataSetTableProducer assembles a sequence of HTML commands to display the records from a dataset in a tabular format.
	TDSTableProducer (see page 10290)	TDSTableProducer is the base class for objects that produce a set of HTML commands to display the records from a dataset in a tabular format.
	TDSTableProducerEditor (see page 10315)	TDSTableProducer (see page 10290) is the base class for the helper object used by TDSTableProducer (see page 10290) objects to communicate with a property editor at design time.
	THTMLTableColumn (see page 10318)	THTMLTableColumn represents the display attributes and field binding of a column in the HTML representation of a dataset.
	THTMLTableColumns (see page 10327)	THTMLTableColumns is a collection of THTMLTableColumn (see page 10318) objects that describe the display attributes and field bindings of the columns in an HTML representation of tabular data.
	THTTPDataLink (see page 10335)	THTTPDataLink helps TDSTableProducer (see page 10290) manage its link to the data source.

Functions

	Name	Description
	HtmlTable (see page 10346)	Generates the HTML image of a dataset, using the properties and events of a table producer object.

Structs, Records, Enums

	Name	Description
	THTMLCaptionAlignment (see page 10347)	This is record DBWeb.THTMLCaptionAlignment.
	THTMLColumnState (see page 10347)	This is record DBWeb.THTMLColumnState.

Types

Name	Description
TCreateContentEvent (see page 10349)	This is type DBWeb.TCreateContentEvent.
THTMLDataSetEmpty (see page 10349)	This is type DBWeb.THTMLDataSetEmpty.
THTMLFormatCellEvent (see page 10350)	This is type DBWeb.THTMLFormatCellEvent.
THTMLGetTableCaptionEvent (see page 10350)	This is type DBWeb.THTMLGetTableCaptionEvent.
THTMLTableColumnClass (see page 10350)	This is type DBWeb.THTMLTableColumnClass.

54.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TDataSetTableProducer (see page 10277)	TDataSetTableProducer assembles a sequence of HTML commands to display the records from a dataset in a tabular format.
	TDSTableProducer (see page 10290)	TDSTableProducer is the base class for objects that produce a set of HTML commands to display the records from a dataset in a tabular format.
	TDSTableProducerEditor (see page 10315)	TDSTableProducer (see page 10290) is the base class for the helper object used by TDSTableProducer (see page 10290) objects to communicate with a property editor at design time.
	THTMLTableColumn (see page 10318)	THTMLTableColumn represents the display attributes and field binding of a column in the HTML representation of a dataset.
	THTMLTableColumns (see page 10327)	THTMLTableColumns is a collection of THTMLTableColumn (see page 10318) objects that describe the display attributes and field bindings of the columns in an HTML representation of tabular data.
	THTTPDataLink (see page 10335)	THTTPDataLink helps TDSTableProducer (see page 10290) manage its link to the data source.

54.1.1 TDataSetTableProducer Class

TDataSetTableProducer assembles a sequence of HTML commands to display the records from a dataset in a tabular format.

Class Hierarchy

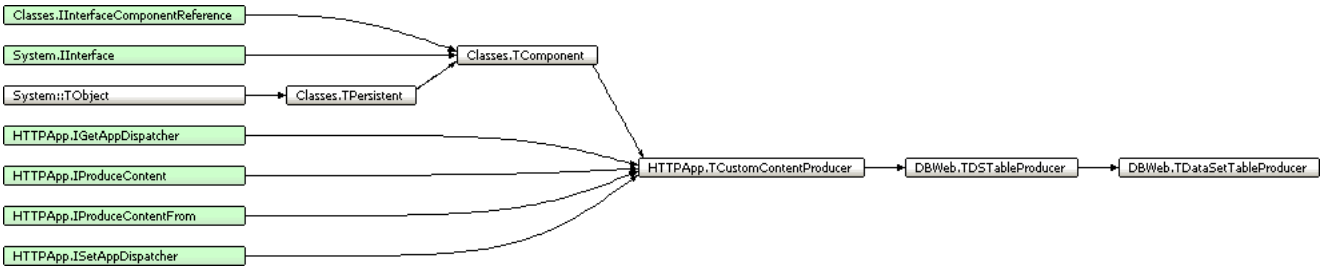

Pascal

```
TDataSetTableProducer = class(TDSTableProducer);
```

C++

```
class TDataSetTableProducer : public TDSTableProducer;
```

File

DBWeb

Description

Use TDataSetTableProducer to produce an HTML table for the records of a TDataSet object. If the dataset is a query that requires input parameters from an HTTP request message, use TQueryTableProducer instead.

TDataSetTableProducer allows an application to create images of a dataset for an HTTP response message.

See Also

TDataSet (see page 6894)

TQueryTableProducer (see page 8152)

TSQLQueryTableProducer (see page 10842)

TWebResponse (see page 15191)

TDataSetPageProducer (see page 11807)

The Web Module

Using TDataSetTableProducer

54.1.1.1 TDataSetTableProducer Members

The following tables list the members exposed by TDataSetTableProducer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AcquireLayoutLock (see page 10296)	Calls BeginLayout (see page 10297) when the HTML table is not currently being laid out.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BeginLayout (see page 10297)	Increments a counter that indicates the number of times the computation of the column order was begun but not completed.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ColumnHeader (see page 10298)	Returns the string of HTML <TH> tags that define the column headers for the table.
	DefineFieldMap (see page 10298)	Builds the Fields array in DataLink (see page 10309).
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoCreateContent (see page 10299)	This is DoCreateContent, a member of class TDSTableProducer.
	DoFormatCell (see page 10299)	Generates an OnFormatCell (see page 10314) event.
	DoGetCaption (see page 10300)	Generates an OnGetCaption event.
	EndLayout (see page 10300)	Decrements a counter that indicates the number of times the computation of the column order has been begun but not completed.
	FormatCell (see page 10301)	Returns an HTML tag of a specified type that reflects a set of specified options.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDataSet (see page 10283)	Provides the abstract prototype of a method to access the DataSet property.
	GetDataSource (see page 10302)	Returns the value of the DataSource (see page 10309) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.

	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	LayoutChanged (see page 10303)	Causes the Columns (see page 10308) property to be recomputed when the dataset changes or changes are made to individual column objects.
	LinkActive (see page 10303)	Recomputes the Columns (see page 10308) property when the dataset becomes active.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 10283)	Adjusts the properties of TDSTableProducer when its helper objects are destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RowHeader (see page 10304)	Returns the HTML <TR> tag that defines each row of the HTML table.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetColumns (see page 10304)	Sets the Columns (see page 10308) property.
V	SetDataSet (see page 10283)	Provides the abstract prototype of a method to set the DataSet property.
	SetDataSource (see page 10305)	Sets the DataSource (see page 10309) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StoreColumns (see page 10305)	Indicates whether the Columns (see page 10308) property is stored when the TDSTableProducer (see page 10290) is saved.
	TableCaption (see page 10306)	Returns the HTML <Caption (see page 10307)> tag that defines the caption for the HTML table.
	TableHeader (see page 10306)	Returns the HTML <Table> tag for the HTML table.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	DataLink (see page 10309)	Manages the link to the dataset.
	DataSource (see page 10309)	Represents the link to the DataSet (see page 10309) of the TDSTableProducer (see page 10290).
R	InternalDataSource (see page 10311)	Represents the link to the dataset.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 10297)	Increments a counter that indicates the number of changes to the contents of the HTML table that have been initiated but not completed.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Content (see page 10282)	Returns the HTML representation of the records in the dataset.
V	ContentFromStream (see page 15107)	Returns the content produced in response to input read from a stream.
V	ContentFromString (see page 15107)	Returns the content produced in response to an input string.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 10295)	Creates an instance of TDSTableProducer (see page 10290).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 10295)	Destroys an instance of TDSTableProducer (see page 10290).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 10301)	Decrements a counter that indicates the number of changes to the contents of the HTML table.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.
	Editor (see page 10310)	Writes changes made in the property editor at design time to the TDSTableProducer (see page 10290) object.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCreateContent (see page 10289)	Occurs in descendants of TDSTableProducer when the Content method needs to generate the HTML table image.
	OnFormatCell (see page 10289)	Occurs when the HTML for a specified cell specified is about to be generated.
	OnGetTableCaption (see page 10290)	Occurs when the HTML table producer is about to generate the HTML for the table caption.

Published Properties

	Name	Description
	Caption (see page 10284)	Specifies the caption string that appears with the HTML table.
	CaptionAlignment (see page 10284)	Specifies where the Caption string should appear relative to the HTML table.

	Columns (see page 10285)	Describes the field bindings and display attributes of the columns in the HTML table.
	DataSet (see page 10285)	Specifies the dataset from which the data in the HTML table is taken.
	Footer (see page 10286)	Specifies a set of HTML commands that appear in the HTML document following the HTML table.
	Header (see page 10286)	Specifies a set of HTML commands that appear in the HTML document preceding the HTML table.
	MaxRows (see page 10287)	Specifies the maximum number of rows that appear in the HTML table.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RowAttributes (see page 10287)	Specifies the display attributes of all the rows in the HTML table.
	TableAttributes (see page 10288)	Specifies the display attributes that pertain to the entire HTML table produced by TDSTableProducer.
	Tag (see page 2808)	Stores an integer value as part of a component.

54.1.1.2 TDataSetTableProducer Methods

The methods of the TDataSetTableProducer class are listed here.

Protected Methods

	Name	Description
	GetDataSet (see page 10283)	Provides the abstract prototype of a method to access the DataSet property.
	Notification (see page 10283)	Adjusts the properties of TDSTableProducer when its helper objects are destroyed.
	SetDataSet (see page 10283)	Provides the abstract prototype of a method to set the DataSet property.

Public Methods

	Name	Description
	Content (see page 10282)	Returns the HTML representation of the records in the dataset.

54.1.1.2.1 TDataSetTableProducer.Content Method

Returns the HTML representation of the records in the dataset.

Pascal

```
function Content: string; override;
```

C++

```
virtual __fastcall AnsiString Content();
```

Description

Call Content to retrieve the HTML representation specified by the properties of the TDataSetTableProducer (see page 10277) object. Content calls the OnCreateContent (see page 10289) event handler. If the OnCreateContent (see page 10289) event handler indicates that the Content method should continue, Content returns the Header (see page 10286) property, followed by an HTML table image of the records in the dataset, followed by the Footer (see page 10286) property. If the OnCreateContent (see page 10289) event handler indicates that the Content method should not continue, Content returns an empty string.

The string returned by Content can be used as the Content property of the response to an HTTP request message.

See Also

Footer (see page 10310)

Header (see page 10311)

OnCreateContent (see page 10313)

TWebRequest (see page 15161)

Content (see page 15207)

54.1.1.2.2 TDataSetTableProducer.GetDataSet Method

Provides the abstract prototype of a method to access the DataSet property.

Pascal

```
function GetDataSet: TDataSet; override;
```

C++

```
virtual __fastcall TDataSet GetDataSet();
```

Description

Descendants of TDSTableProducer implement the GetDataSet method to access their specific implementation of the DataSet property. Descendants of TDSTableProducer restrict the DataSet property to different classes of TDataSet objects.

See Also

DataSet (see page 10309)

SetDataSet (see page 10305)

54.1.1.2.3 TDataSetTableProducer.Notification Method

Adjusts the properties of TDSTableProducer when its helper objects are destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification overrides the inherited method to respond to messages that the DataSource or one of its field objects is about to be destroyed. Notification sets the DataSource property to nil (Delphi) or NULL (C++), or unbinds the column that references the field component.

See Also

DataSource (see page 10309)

Columns (see page 10308)

54.1.1.2.4 TDataSetTableProducer.SetDataSet Method

Provides the abstract prototype of a method to set the DataSet property.

Pascal

```
procedure SetDataSet(ADataSet: TDataSet); override;
```

C++

```
virtual __fastcall SetDataSet(TDataSet ADataSet);
```

Description

Descendants of TDSTableProducer override SetDataSet to ensure that the DataSet property can only be set to an appropriate

descendant of TDataSet.

See Also

DataSet (see page 10309)

GetDataSet (see page 10302)

54.1.1.3 TDataSetTableProducer Properties

The properties of the TDataSetTableProducer class are listed here.

Published Properties

	Name	Description
	Caption (see page 10284)	Specifies the caption string that appears with the HTML table.
	CaptionAlignment (see page 10284)	Specifies where the Caption string should appear relative to the HTML table.
	Columns (see page 10285)	Describes the field bindings and display attributes of the columns in the HTML table.
	DataSet (see page 10285)	Specifies the dataset from which the data in the HTML table is taken.
	Footer (see page 10286)	Specifies a set of HTML commands that appear in the HTML document following the HTML table.
	Header (see page 10286)	Specifies a set of HTML commands that appear in the HTML document preceding the HTML table.
	MaxRows (see page 10287)	Specifies the maximum number of rows that appear in the HTML table.
	RowAttributes (see page 10287)	Specifies the display attributes of all the rows in the HTML table.
	TableAttributes (see page 10288)	Specifies the display attributes that pertain to the entire HTML table produced by TDSTableProducer.

54.1.1.3.1 TDataSetTableProducer.Caption Property

Specifies the caption string that appears with the HTML table.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Set Caption to add a caption to the HTML table that represents the dataset. Caption indicates the text that should appear inside the HTML CAPTION tag.

See Also

CaptionAlignment (see page 10308)

OnGetTableCaption (see page 10314)

Embedding Tables in HTML Documents

54.1.1.3.2 TDataSetTableProducer.CaptionAlignment Property

Specifies where the Caption string should appear relative to the HTML table.

Pascal

```
property CaptionAlignment: THTMLCaptionAlignment;
```

C++

```
__property THTMLCaptionAlignment CaptionAlignment;
```

Description

Set CaptionAlignment to specify whether the Caption should appear above or below the HTML table.

CaptionAlignment is the value of the ALIGN option on the HTML CAPTION tag.

See Also

Caption (see page 10307)

Embedding Tables in HTML Documents

54.1.1.3.3 TDataSetTableProducer.Columns Property

Describes the field bindings and display attributes of the columns in the HTML table.

Pascal

```
property Columns: THTMLTableColumns;
```

C++

```
__property THTMLTableColumns Columns;
```

Description

Use Columns to read or set the field bindings and display attributes of the columns in the HTML table. Each THTMLTableColumn object specifies the display attributes of the corresponding column. Each column object also specifies the name of the field the column represents. The field binding of a column designates a field within the dataset specified by the DataSet property.

Columns can be set at design time through a property editor, or programmatically at runtime.

See Also

DataSet (see page 10309)

DefineFieldMap (see page 10298)

Editor (see page 10310)

RowAttributes (see page 10312)

SetColumns (see page 10304)

StoreColumns (see page 10305)

TableAttributes (see page 10313)

THTMLTableColumn (see page 10318)

THTMLTableColumns (see page 10327)

Specifying the Columns

54.1.1.3.4 TDataSetTableProducer.DataSet Property

Specifies the dataset from which the data in the HTML table is taken.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Set DataSet to specify where the TDSTableProducer should look for the data it displays in the HTML table. The individual field bindings specified by the THTMLTableColumn objects in the Columns property refer to fields in DataSet.

Specify the data binding of a TDSTableProducer using the DataSet property rather than through a DataSource property. TDSTableProducer objects do not use an external TDataSource object.

See Also

DataSource (see page 10309)

InternalDataSource (see page 10311)

Columns (see page 10308)

GetDataSet (see page 10302)

SetDataSet (see page 10305)

FieldName (see page 10326)

Using TDataSetTableProducer

54.1.1.3.5 TDataSetTableProducer.Footer Property

Specifies a set of HTML commands that appear in the HTML document following the HTML table.

Pascal

```
property Footer: TStrings;
```

C++

```
__property TStrings Footer;
```

Description

Set Footer to embed the representation of DataSet in a larger document. The Content method of TDSTableProducer descendants returns any Header commands, followed by the HTML table that represents the DataSet, followed by any Footer commands.

See Also

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

DataSet (see page 10309)

Header (see page 10311)

Embedding Tables in HTML Documents

54.1.1.3.6 TDataSetTableProducer.Header Property

Specifies a set of HTML commands that appear in the HTML document preceding the HTML table.

Pascal

```
property Header: TStrings;
```

C++

```
__property TStrings Header;
```

Description

Set Header to embed the representation of DataSet in a larger document. The Content method of TDSTableProducer descendants returns any Header commands, followed by the HTML table that represents the DataSet, followed by any Footer commands.

See Also

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

DataSet (see page 10309)

Footer (see page 10310)

Embedding Tables in HTML Documents

54.1.1.3.7 TDataSetTableProducer.MaxRows Property

Specifies the maximum number of rows that appear in the HTML table.

Pascal

```
property MaxRows: Integer;
```

C++

```
__property int MaxRows;
```

Description

Set MaxRows to change the upper limit on the size of the HTML table produced to represent the DataSet. MaxRows is initialized to 20.

If DataSet contains more than MaxRows records, only the first MaxRows records appear in the HTML commands produced by Content. If DataSet contains fewer than MaxRows records, the HTML table produced by TDSTableProducer will contain only enough rows to display all the records in DataSet.

See Also

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

DataSet (see page 10309)

Specifying the Row Attributes

54.1.1.3.8 TDataSetTableProducer.RowAttributes Property

Specifies the display attributes of all the rows in the HTML table.

Pascal

```
property RowAttributes: THTMLTableRowAttributes;
```

C++

```
__property THTMLTableRowAttributes RowAttributes;
```

Description

Set RowAttributes to a THTMLTableRowAttributes object that describes the display attributes of the rows in the HTML table. Setting RowAttributes calls the Assign method of the THTMLTableRowAttributes object stored internally by TDSTableProducer. Thus, the memory for any row attributes object assigned to the RowAttributes property is never owned or freed by TDSTableProducer.

Note: Not all the properties of the row attributes object are copied when one row attributes object is assigned to another. To ensure that all the attributes represented by RowAttributes are set appropriately, read RowAttributes and set the individual properties of the THTMLTableRowAttributes object.

See Also

TableAttributes (see page 10313)

Title (see page 10327)

AssignTo (see page 15297)

Specifying the Row Attributes

54.1.1.3.9 TDataSetTableProducer.TableAttributes Property

Specifies the display attributes that pertain to the entire HTML table produced by TDSTableProducer.

Pascal

```
property TableAttributes: THTMLTableAttributes;
```

C++

```
__property THTMLTableAttributes TableAttributes;
```

Description

Set TableAttributes to a THTMLTableAttributes object that describes the display attributes for the HTML TABLE tag. Setting TableAttributes calls the Assign method of the THTMLTableAttributes object stored internally by TDSTableProducer. Thus, the memory for any table attributes object assigned to the TableAttributes property is never owned or freed by TDSTableProducer.

Note: Not all the properties of the table attributes object are copied when one table attributes object is assigned to another. To ensure that all the attributes represented by TableAttributes are set appropriately, read TableAttributes and set the individual properties of the THTMLTableAttributes object.

See Also

RowAttributes (see page 10312)

AssignTo (see page 15291)

Title (see page 10327)

Specifying the Table Attributes

54.1.1.4 TDataSetTableProducer Events

The events of the TDataSetTableProducer class are listed here.

Published Events

	Name	Description
	OnCreateContent (see page 10289)	Occurs in descendants of TDSTableProducer when the Content method needs to generate the HTML table image.
	OnFormatCell (see page 10289)	Occurs when the HTML for a specified cell specified is about to be generated.
	OnGetTableCaption (see page 10290)	Occurs when the HTML table producer is about to generate the HTML for the table caption.

54.1.1.4.1 TDataSetTableProducer.OnCreateContent Event

Occurs in descendants of TDSTableProducer when the Content method needs to generate the HTML table image.

Pascal

```
property OnCreateContent: TCreateContentEvent;
```

C++

```
__property TCreateContentEvent OnCreateContent;
```

Description

Write an OnCreateContent event handler to conditionally prevent the Content method from producing any HTML commands. Set the Continue parameter to false to make Content return an empty string.

See Also

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

docreatecontent (see page 10299)

54.1.1.4.2 TDataSetTableProducer.OnFormatCell Event

Occurs when the HTML for a specified cell specified is about to be generated.

Pascal

```
property OnFormatCell: THTMLFormatCellEvent;
```

C++

```
__property THTMLFormatCellEvent OnFormatCell;
```

Description

Write an OnFormatCell event handler to change the appearance or contents of individual cells in the HTML table from the default values produced by TDSTableProducer.

See Also

Columns (see page 10308)

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

DoFormatCell (see page 10299)

DisplayText (see page 7162)

Align (see page 10324)

BgColor (see page 10325)

Custom (see page 10325)

VAlign (see page 10327)

54.1.1.4.3 TDataSetTableProducer.OnGetTableCaption Event

Occurs when the HTML table producer is about to generate the HTML for the table caption.

Pascal

```
property OnGetTableCaption: THTMLGetTableCaptionEvent;
```

C++

```
__property THTMLGetTableCaptionEvent OnGetTableCaption;
```

Description

Write an OnGetTableCaption event handler to change the value or placement of the HTML table caption. The Caption and Alignment parameters are originally set to the values of the Caption and CaptionAlignment properties, respectively. The OnGetTableCaption event handler can change these values to alter the caption for the HTML table returned by the Content method.

See Also

Caption (see page 10307)

CaptionAlignment (see page 10308)

Content (see page 15106)

DoGetCaption (see page 10300)

54.1.2 TDSTableProducer Class

TDSTableProducer is the base class for objects that produce a set of HTML commands to display the records from a dataset in a tabular format.

Class Hierarchy

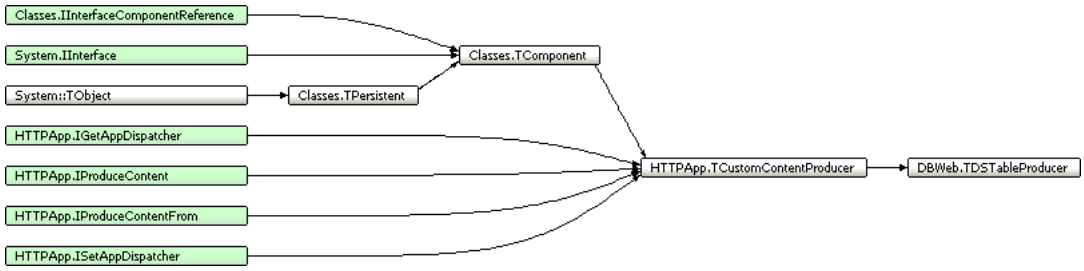

Pascal

```
TDSTableProducer = class(TCustomContentProducer);
```

C++

```
class TDSTableProducer : public TCustomContentProducer;
```

File

DBWeb

Description

Use TDSTableProducer as a base class when defining a class that produces an HTML table that represents the records of a dataset. TDSTableProducer introduces many properties to describe the appearance and data binding of the HTML table. The events and methods of TDSTableProducer allow an application to retrieve the database information and produce the HTML commands that describe the table.

See Also

TDataSetTableProducer (see page 10277)

TQueryTableProducer (see page 8152)

TSQLQueryTableProducer (see page 10842)

54.1.2.1 TDSTableProducer Members

The following tables list the members exposed by TDSTableProducer.

Protected Events

	Name	Description
	OnCreateContent (see page 10313)	Occurs in descendants of TDSTableProducer (see page 10290) when the Content method needs to generate the HTML table image.
	OnFormatCell (see page 10314)	Occurs when the HTML for a specified cell specified is about to be generated.
	OnGetTableCaption (see page 10314)	Occurs when the HTML table producer is about to generate the HTML for the table caption.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AcquireLayoutLock (see page 10296)	Calls BeginLayout (see page 10297) when the HTML table is not currently being laid out.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BeginLayout (see page 10297)	Increments a counter that indicates the number of times the computation of the column order was begun but not completed.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ColumnHeader (see page 10298)	Returns the string of HTML <TH> tags that define the column headers for the table.
	DefineFieldMap (see page 10298)	Builds the Fields array in DataLink (see page 10309).
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	DoCreateContent (see page 10299)	This is DoCreateContent, a member of class TDSTableProducer.
	DoFormatCell (see page 10299)	Generates an OnFormatCell (see page 10314) event.
	DoGetCaption (see page 10300)	Generates an OnGetCaption event.
	EndLayout (see page 10300)	Decrements a counter that indicates the number of times the computation of the column order has been begun but not completed.
	FormatCell (see page 10301)	Returns an HTML tag of a specified type that reflects a set of specified options.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
A	GetDataSet (see page 10302)	Provides the abstract prototype of a method to access the DataSet (see page 10309) property.
	GetDataSource (see page 10302)	Returns the value of the DataSource (see page 10309) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	LayoutChanged (see page 10303)	Causes the Columns (see page 10308) property to be recomputed when the dataset changes or changes are made to individual column objects.
	LinkActive (see page 10303)	Recomputes the Columns (see page 10308) property when the dataset becomes active.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 10304)	Adjusts the properties of TDSTableProducer (see page 10290) when its helper objects are destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RowHeader (see page 10304)	Returns the HTML <TR> tag that defines each row of the HTML table.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetColumns (see page 10304)	Sets the Columns (see page 10308) property.
A	SetDataSet (see page 10305)	Provides the abstract prototype of a method to set the DataSet (see page 10309) property.
	SetDataSource (see page 10305)	Sets the DataSource (see page 10309) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StoreColumns (see page 10305)	Indicates whether the Columns (see page 10308) property is stored when the TDSTableProducer (see page 10290) is saved.
	TableCaption (see page 10306)	Returns the HTML <Caption (see page 10307)> tag that defines the caption for the HTML table.
	TableHeader (see page 10306)	Returns the HTML <Table> tag for the HTML table.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DataLink (see page 10309)	Manages the link to the dataset.
	DataSource (see page 10309)	Represents the link to the DataSet (see page 10309) of the TDSTableProducer (see page 10290).
	InternalDataSource (see page 10311)	Represents the link to the dataset.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 10297)	Increments a counter that indicates the number of changes to the contents of the HTML table that have been initiated but not completed.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 15106)	Returns the content produced by the content producer.
	ContentFromStream (see page 15107)	Returns the content produced in response to input read from a stream.
	ContentFromString (see page 15107)	Returns the content produced in response to an input string.
	Create (see page 10295)	Creates an instance of TDSTableProducer (see page 10290).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 10295)	Destroys an instance of TDSTableProducer (see page 10290).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 10301)	Decrements a counter that indicates the number of changes to the contents of the HTML table.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 10307)	Specifies the caption string that appears with the HTML table.
	CaptionAlignment (see page 10308)	Specifies where the Caption (see page 10307) string should appear relative to the HTML table.
	Columns (see page 10308)	Describes the field bindings and display attributes of the columns in the HTML table.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DataSet (see page 10309)	Specifies the dataset from which the data in the HTML table is taken.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.

	Editor (see page 10310)	Writes changes made in the property editor at design time to the TDSTableProducer (see page 10290) object.
	Footer (see page 10310)	Specifies a set of HTML commands that appear in the HTML document following the HTML table.
	Header (see page 10311)	Specifies a set of HTML commands that appear in the HTML document preceding the HTML table.
	MaxRows (see page 10312)	Specifies the maximum number of rows that appear in the HTML table.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RowAttributes (see page 10312)	Specifies the display attributes of all the rows in the HTML table.
	TableAttributes (see page 10313)	Specifies the display attributes that pertain to the entire HTML table produced by TDSTableProducer (see page 10290).
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

54.1.2.2 TDSTableProducer.Create Constructor

Creates an instance of TDSTableProducer (see page 10290).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDSTableProducer(TComponent * AOwner);
```

Description

Call Create to create a table producer from within the OnAction event handler of a TWebActionItem that needs to produce an HTML image of a dataset in response to an HTTP request message.

After calling the inherited constructor, Create allocates all the helper objects used by TDSTableProducer (see page 10290) and initializes the MaxRows (see page 10312) property to 20.

See Also

MaxRows (see page 10312)

Create (see page 2766)

54.1.2.3 TDSTableProducer.Destroy Destructor

Destroys an instance of TDSTableProducer (see page 10290).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDSTableProducer();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TDSTableProducer (see page 10290) reference is not nil, and only then calls Destroy.

Destroy frees all the helper objects used by TDSTableProducer (see page 10290).

See Also

Free (see page 22300)

54.1.2.4 TDSTableProducer Methods

The methods of the TDSTableProducer class are listed here.

Protected Methods

	Name	Description
	AcquireLayoutLock (see page 10296)	Calls BeginLayout (see page 10297) when the HTML table is not currently being laid out.
	BeginLayout (see page 10297)	Increments a counter that indicates the number of times the computation of the column order was begun but not completed.
	ColumnHeader (see page 10298)	Returns the string of HTML <TH> tags that define the column headers for the table.
	DefineFieldMap (see page 10298)	Builds the Fields array in DataLink (see page 10309).
	DoCreateContent (see page 10299)	This is DoCreateContent, a member of class TDSTableProducer.
	DoFormatCell (see page 10299)	Generates an OnFormatCell (see page 10314) event.
	DoGetCaption (see page 10300)	Generates an OnGetCaption event.
	EndLayout (see page 10300)	Decrements a counter that indicates the number of times the computation of the column order has been begun but not completed.
	FormatCell (see page 10301)	Returns an HTML tag of a specified type that reflects a set of specified options.
A	GetDataSet (see page 10302)	Provides the abstract prototype of a method to access the DataSet (see page 10309) property.
	GetDataSource (see page 10302)	Returns the value of the DataSource (see page 10309) property.
	LayoutChanged (see page 10303)	Causes the Columns (see page 10308) property to be recomputed when the dataset changes or changes are made to individual column objects.
	LinkActive (see page 10303)	Recomputes the Columns (see page 10308) property when the dataset becomes active.
V	Notification (see page 10304)	Adjusts the properties of TDSTableProducer (see page 10290) when its helper objects are destroyed.
	RowHeader (see page 10304)	Returns the HTML <TR> tag that defines each row of the HTML table.
	SetColumns (see page 10304)	Sets the Columns (see page 10308) property.
A	SetDataSet (see page 10305)	Provides the abstract prototype of a method to set the DataSet (see page 10309) property.
	SetDataSource (see page 10305)	Sets the DataSource (see page 10309) property.
	StoreColumns (see page 10305)	Indicates whether the Columns (see page 10308) property is stored when the TDSTableProducer (see page 10290) is saved.
	TableCaption (see page 10306)	Returns the HTML <Caption (see page 10307)> tag that defines the caption for the HTML table.
	TableHeader (see page 10306)	Returns the HTML <Table> tag for the HTML table.

Public Methods

	Name	Description
	BeginUpdate (see page 10297)	Increments a counter that indicates the number of changes to the contents of the HTML table that have been initiated but not completed.
	EndUpdate (see page 10301)	Decrements a counter that indicates the number of changes to the contents of the HTML table.

54.1.2.4.1 TDSTableProducer.AcquireLayoutLock Method

Calls BeginLayout (see page 10297) when the HTML table is not currently being laid out.

Pascal

```
function AcquireLayoutLock: Boolean;
```

C++

```
__fastcall Boolean AcquireLayoutLock();
```

Description

Applications cannot call this protected method. AcquireLayoutLock is called before recomputing Columns (see page 10308) when the fields in the dataset change or when the Columns (see page 10308) property changes. AcquireLayoutLock returns true if the HTML table is not already being laid out, to indicate that the computation of Columns (see page 10308) should proceed. AcquireLayoutLock prevents the computation of Columns (see page 10308) from looping endlessly in response to the changes that occur during the computation.

See Also

BeginLayout (see page 10297)

Columns (see page 10308)

LayoutChanged (see page 10303)

54.1.2.4.2 TDSTableProducer.BeginLayout Method

Increments a counter that indicates the number of times the computation of the column order was begun but not completed.

Pascal

```
procedure BeginLayout;
```

C++

```
__fastcall BeginLayout();
```

Description

Applications cannot call this protected method. TDSTableProducer (see page 10290) calls BeginLayout internally before making changes to the Columns (see page 10308) property or any of the individual THTMLTableColumn (see page 10318) objects in Columns (see page 10308). Once the change is complete, TDSTableProducer (see page 10290) calls EndLayout (see page 10300).

When all BeginLayout calls have been matched by a corresponding call to EndLayout (see page 10300), the field mapping maintained by DataLink (see page 10309) for the fields in the dataset is updated to match both the field bindings of the Columns (see page 10308) property and the fields available in the dataset. The Columns (see page 10308) property is updated in turn to match the field bindings in DataLink (see page 10309). This ensures that the Columns (see page 10308) property matches both the specified column bindings and the fields available in the dataset.

See Also

AcquireLayoutLock (see page 10296)

BeginUpdate (see page 10297)

Columns (see page 10308)

DataLink (see page 10309)

DefineFieldMap (see page 10298)

EndLayout (see page 10300)

54.1.2.4.3 TDSTableProducer.BeginUpdate Method

Increments a counter that indicates the number of changes to the contents of the HTML table that have been initiated but not

completed.

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

Applications should not call BeginUpdate. BeginUpdate is called internally before changes that affect the contents of the HTML table. Once a change is complete, EndUpdate (see page 10301) is called. BeginUpdate and EndUpdate (see page 10301) allow descendants of TDSTableProducer (see page 10290) to wait until changes to the dataset are complete before fetching records for the HTML image of the table.

See Also

BeginLayout (see page 10297)

DataSet (see page 10309)

EndUpdate (see page 10301)

54.1.2.4.4 TDSTableProducer.ColumnHeader Method

Returns the string of HTML <TH> tags that define the column headers for the table.

Pascal

```
function ColumnHeader: string; dynamic;
```

C++

```
__fastcall AnsiString ColumnHeader();
```

Description

The table producer uses this protected method when it assembles its HTML table. For each THTMLTableColumn (see page 10318) object listed in the Columns (see page 10308) property, ColumnHeader looks up the string that labels it (using its Title property or the display name for the field if there is no title caption). ColumnHeader then creates a table header (<TH>) tag that describes the column label, including all the properties specified by the THTMLTableColumn (see page 10318) object.

See Also

Columns (see page 10308)

RowHeader (see page 10304)

TableHeader (see page 10306)

Title (see page 10327)

54.1.2.4.5 TDSTableProducer.DefineFieldMap Method

Builds the Fields array in DataLink (see page 10309).

Pascal

```
procedure DefineFieldMap;
```

C++

```
__fastcall DefineFieldMap();
```

Description

Applications cannot call this protected method. DefineFieldMap rebuilds the field mapping for the dataset that is maintained by DataLink (see page 10309). This mapping reflects both the column bindings of the Columns (see page 10308) property and the fields available in the dataset. After calling DefineFieldMap, the Columns (see page 10308) property should be updated to match the field bindings in DataLink (see page 10309). This ensures that the Columns (see page 10308) property reflects any changes imposed by the dataset.

See Also

Columns (see page 10308)

DataSet (see page 10309)

EndLayout (see page 10300)

Fields (see page 10345)

54.1.2.4.6 TDSTableProducer.DoCreateContent Method

Pascal

```
function DoCreateContent: Boolean;
```

C++

```
__fastcall Boolean DoCreateContent();
```

Description

This is DoCreateContent, a member of class TDSTableProducer.

54.1.2.4.7 TDSTableProducer.DoFormatCell Method

Generates an OnFormatCell (see page 10314) event.

Pascal

```
procedure DoFormatCell(CellRow: Integer; CellColumn: Integer; var BgColor: THTMLBgColor; var
Align: THTMLAlign; var VAlign: THTMLVAlign; var CustomAttrs: string; var CellData: string);
dynamic;
```

C++

```
__fastcall DoFormatCell(int CellRow, int CellColumn, THTMLBgColor BgColor, THTMLAlign Align,
THTMLVAlign VAlign, AnsiString CustomAttrs, AnsiString CellData);
```

Description

Applications can't call this protected method. It is called internally by the Content method to allow the OnFormatCell (see page 10314) event handler to customize the contents or display attributes of any cell in the HTML table. CellRow and CellColumn indicate the cell about to be rendered as HTML, where CellRow for the table header is 0, and CellColumn for the first column is 0. BgColor, Align, VAlign, and CustomAttrs are the display attributes that can be changed. CellData is the text that goes in the cell.

Descendants of TDSTableProducer (see page 10290) can override DoFormatCell to make additional changes to the cells as they are rendered or to block the OnFormatCell (see page 10314) event.

See Also

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

OnFormatCell (see page 10314)

54.1.2.4.8 TDSTableProducer.DoGetCaption Method

Generates an OnGetCaption event.

Pascal

```
procedure DoGetCaption(var TableCaption: string; var CaptionAlign: THTMLCaptionAlignment);
dynamic;
```

C++

```
__fastcall DoGetCaption(AnsiString TableCaption, THTMLCaptionAlignment CaptionAlign);
```

Description

Applications can't call this protected method. It is called internally by the Content method to let the OnGetTableCaption (see page 10314) event handler change the Caption (see page 10307) and CaptionAlignment (see page 10308) properties when the table caption is rendered as HTML.

Descendants of TDSTableProducer (see page 10290) can override DoCreateCaption to make additional changes when the caption is rendered or to block the OnCreateCaption event.

See Also

Caption (see page 10307)

CaptionAlignment (see page 10308)

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

OnGetTableCaption (see page 10314)

54.1.2.4.9 TDSTableProducer.EndLayout Method

Decrements a counter that indicates the number of times the computation of the column order has been begun but not completed.

Pascal

```
procedure EndLayout;
```

C++

```
__fastcall EndLayout();
```

Description

Applications can't call this protected method. TDSTableProducer (see page 10290) objects call EndLayout internally after making changes to the Columns (see page 10308) property or when changes are made to the fields in DataSet (see page 10309).

Each call to EndLayout should match a corresponding call to BeginLayout (see page 10297). If all BeginLayout (see page 10297) calls have been matched by an EndLayout call, EndLayout calls DefineFieldMap (see page 10298) to regenerate the field mapping for the dataset maintained by DataLink (see page 10309). The Columns (see page 10308) property is then updated to match the field map in DataLink (see page 10309). This ensures that the Columns (see page 10308) property matches both the specified column bindings and the fields available in the dataset.

See Also

BeginLayout (see page 10297)

Columns (see page 10308)

DataLink (see page 10309)

DefineFieldMap (see page 10298)

EndUpdate (see page 10301)

54.1.2.4.10 TDSTableProducer.EndUpdate Method

Decrements a counter that indicates the number of changes to the contents of the HTML table.

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

Applications should not call EndUpdate. EndUpdate is called internally after changes to the column bindings or DataSet (see page 10309) of the TDSTableProducer (see page 10290). BeginUpdate (see page 10297) and EndUpdate allow descendants of TDSTableProducer (see page 10290) to wait until changes to the dataset are complete before fetching records for the HTML image of the table.

See Also

BeginUpdate (see page 10297)

Columns (see page 10308)

DataSet (see page 10309)

EndLayout (see page 10300)

54.1.2.4.11 TDSTableProducer.FormatCell Method

Returns an HTML tag of a specified type that reflects a set of specified options.

Pascal

```
function FormatCell(CellRow: Integer; CellColumn: Integer; CellData: string; const Tag:
string; const BgColor: THTMLBgColor; Align: THTMLAlign; VAlign: THTMLVAlign; const Custom:
string): string; dynamic;
```

C++

```
__fastcall AnsiString FormatCell(int CellRow, int CellColumn, AnsiString CellData, const
AnsiString Tag, const THTMLBgColor BgColor, THTMLAlign Align, THTMLVAlign VAlign, const
AnsiString Custom);
```

Description

The table producer uses this protected utility to assemble a set of properties into a single HTML tag. FormatCell first generates an OnFormatCell (see page 10314) event, so that the application can override any of the properties that control the tag options. Then it assembles the resulting values into the specified tag.

CellRow and CellColumn identify the table cell to be formatted. These parameters are passed to an OnFormatCell (see page 10314) event handler so that it can identify the cell.

CellData contains the text displayed in the cell (a data value or column header).

Tag indicates the type of HTML tag to generate (<TD> for data cells, <TH> for column headers).

BgColor specifies the background color of the cell. An OnFormatCell (see page 10314) event handler can override this value.

Align specifies the horizontal alignment of CellData within the cell. An OnFormatCell (see page 10314) event handler can override this value.

VAlign specifies the vertical alignment of CellData within the cell. An OnFormatCell (see page 10314) event handler can override this value.

Custom specifies any other options for the generated HTML tag. An OnFormatCell (see page 10314) event handler can override this value.

FormatCell returns a string with the generated HTML tag.

See Also

OnFormatCell (see page 10314)

54.1.2.4.12 TDSTableProducer.GetDataSet Method

Provides the abstract prototype of a method to access the DataSet (see page 10309) property.

Pascal

```
function GetDataSet: TDataSet; virtual; abstract;
```

C++

```
virtual __fastcall TDataSet GetDataSet() = 0;
```

Description

Descendants of TDSTableProducer (see page 10290) implement the GetDataSet method to access their specific implementation of the DataSet (see page 10309) property. Descendants of TDSTableProducer (see page 10290) restrict the DataSet (see page 10309) property to different classes of TDataSet objects.

See Also

DataSet (see page 10309)

SetDataSet (see page 10305)

54.1.2.4.13 TDSTableProducer.GetDataSource Method

Returns the value of the DataSource (see page 10309) property.

Pascal

```
function GetDataSource: TDataSource;
```

C++

```
__fastcall TDataSource GetDataSource();
```

Description

GetDataSource is exposed as a protected method so that descendants of TDSTableProducer (see page 10290) can override the implementation of the DataSource (see page 10309) property.

The DataSource (see page 10309) property for TDSTableProducer (see page 10290) is an internally generated TDataSource object, not an external component that can be shared with other objects. It is surfaced only for internal APIs that require a TDataSource object rather than working directly with TDataSet.

See Also

DataSource (see page 10309)

SetDataSource (see page 10305)

54.1.2.4.14 TDSTableProducer.LayoutChanged Method

Causes the Columns (see page 10308) property to be recomputed when the dataset changes or changes are made to individual column objects.

Pascal

```
procedure LayoutChanged;
```

C++

```
__fastcall LayoutChanged();
```

Description

Applications cannot call this protected method. It is called when the Columns (see page 10308) property changes or the fields in the dataset change. LayoutChanged recomputes the Columns (see page 10308) property after guarding against recursion from the changes to the Columns (see page 10308) property that arise in the computation.

LayoutChanged calls AcquireLayoutLock (see page 10296) to initiate the changes to Columns (see page 10308). If AcquireLayoutLock (see page 10296) does not indicate that the call to LayoutChanged is recursive, LayoutChanged calls EndLayout (see page 10300) to perform the actual computation.

See Also

AcquireLayoutLock (see page 10296)

Columns (see page 10308)

EndLayout (see page 10300)

54.1.2.4.15 TDSTableProducer.LinkActive Method

Recomputes the Columns (see page 10308) property when the dataset becomes active.

Pascal

```
procedure LinkActive(Value: Boolean);
```

C++

```
__fastcall LinkActive(Boolean Value);
```

Description

Applications cannot call this protected method. LinkActive is called when the DataSource (see page 10309) property changes or when the dataset becomes active. It recomputes Columns (see page 10308) to reflect any changes that may have occurred in the fields of the dataset when it was not active.

See Also

Columns (see page 10308)

DataSource (see page 10309)

LayoutChanged (see page 10303)

54.1.2.4.16 TDSTableProducer.Notification Method

Adjusts the properties of TDSTableProducer (see page 10290) when its helper objects are destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification overrides the inherited method to respond to messages that the DataSource (see page 10309) or one of its field objects is about to be destroyed. Notification sets the DataSource (see page 10309) property to nil (Delphi) or NULL (C++), or unbinds the column that references the field component.

See Also

DataSource (see page 10309)

Columns (see page 10308)

54.1.2.4.17 TDSTableProducer.RowHeader Method

Returns the HTML <TR> tag that defines each row of the HTML table.

Pascal

```
function RowHeader: string; dynamic;
```

C++

```
__fastcall AnsiString RowHeader();
```

Description

The table producer calls this protected method for each row when it assembles its HTML table. RowHeader generates an HTML table row tag (<TR>), using the RowAttributes (see page 10312) property to determine what options are added to the tag.

See Also

RowAttributes (see page 10312)

54.1.2.4.18 TDSTableProducer.SetColumns Method

Sets the Columns (see page 10308) property.

Pascal

```
procedure SetColumns(Value: THTMLTableColumns);
```

C++

```
__fastcall SetColumns(THTMLTableColumns Value);
```

Description

In Delphi, SetColumns is the protected method used to set the Columns (see page 10308) property. In C++, SetColumns is the protected implementation of the Columns (see page 10308) property. SetColumns calls the Assign method of the THTMLTableColumns (see page 10327) object stored internally by TDSTableProducer (see page 10290). Thus, the memory for any columns object assigned to the Columns (see page 10308) property is never owned or freed by TDSTableProducer (see page 10290).

See Also

Columns (see page 10308)

Assign (see page 2736)

AssignTo (see page 10322)

54.1.2.4.19 TDSTableProducer.SetDataSet Method

Provides the abstract prototype of a method to set the DataSet (see page 10309) property.

Pascal

```
procedure SetDataSet(ADataSet: TDataSet); virtual; abstract;
```

C++

```
virtual __fastcall SetDataSet(TDataSet ADataSet) = 0;
```

Description

Descendants of TDSTableProducer (see page 10290) override SetDataSet to ensure that the DataSet (see page 10309) property can only be set to an appropriate descendant of TDataSet.

See Also

DataSet (see page 10309)

GetDataSet (see page 10302)

54.1.2.4.20 TDSTableProducer.SetDataSource Method

Sets the DataSource (see page 10309) property.

Pascal

```
procedure SetDataSource(Value: TDataSource);
```

C++

```
__fastcall SetDataSource(TDataSource Value);
```

Description

SetDataSource is the protected implementation of the DataSource (see page 10309) property.

The DataSource (see page 10309) property for TDSTableProducer (see page 10290) is an internally generated TDataSource object, not an external component that can be shared with other objects. It is surfaced only for internal API calls that require a TDataSource object rather than working directly with TDataSet.

See Also

DataSource (see page 10309)

GetDataSource (see page 10302)

54.1.2.4.21 TDSTableProducer.StoreColumns Method

Indicates whether the Columns (see page 10308) property is stored when the TDSTableProducer (see page 10290) is saved.

Pascal

```
function StoreColumns: Boolean;
```

C++

```
__fastcall Boolean StoreColumns();
```

Description

StoreColumns causes the Columns (see page 10308) property to be stored only when the Columns (see page 10308) property has a State property of csCustom. The State property for Columns (see page 10308) indicates whether the columns represent default display attributes for all the fields in the dataset, or if the display attributes or field bindings of the columns have been customized programmatically or by the Columns (see page 10308) editor.

See Also

Columns (see page 10308)

State (see page 10335)

54.1.2.4.22 TDSTableProducer.TableCaption Method

Returns the HTML <Caption (see page 10307)> tag that defines the caption for the HTML table.

Pascal

```
function TableCaption: string; dynamic;
```

C++

```
__fastcall AnsiString TableCaption();
```

Description

The table producer calls this protected method when it assembles its HTML table. TableCaption generates an OnGetTableCaption (see page 10314) event to allow an event handler to override the Caption (see page 10307) and CaptionAlignment (see page 10308) properties. It then returns the HTML table caption tag (<caption>) with the resulting caption and alignment.

See Also

Caption (see page 10307)

OnGetTableCaption (see page 10314)

CaptionAlignment (see page 10308)

54.1.2.4.23 TDSTableProducer.TableHeader Method

Returns the HTML <Table> tag for the HTML table.

Pascal

```
function TableHeader: string; dynamic;
```

C++

```
__fastcall AnsiString TableHeader();
```

Description

The table producer calls this protected method when it assembles its HTML table. TableHeader generates an HTML table tag (<Table>), using the TableAttributes (see page 10313) property to determine what options are added to the tag.

See Also

TableAttributes (see page 10313)

54.1.2.5 TDSTableProducer Properties

The properties of the TDSTableProducer class are listed here.

Protected Properties

	Name	Description
R	DataLink (see page 10309)	Manages the link to the dataset.
	DataSource (see page 10309)	Represents the link to the DataSet (see page 10309) of the TDSTableProducer (see page 10290).
R	InternalDataSource (see page 10311)	Represents the link to the dataset.

Public Properties

	Name	Description
	Caption (see page 10307)	Specifies the caption string that appears with the HTML table.
	CaptionAlignment (see page 10308)	Specifies where the Caption (see page 10307) string should appear relative to the HTML table.
	Columns (see page 10308)	Describes the field bindings and display attributes of the columns in the HTML table.
	DataSet (see page 10309)	Specifies the dataset from which the data in the HTML table is taken.
	Editor (see page 10310)	Writes changes made in the property editor at design time to the TDSTableProducer (see page 10290) object.
	Footer (see page 10310)	Specifies a set of HTML commands that appear in the HTML document following the HTML table.
	Header (see page 10311)	Specifies a set of HTML commands that appear in the HTML document preceding the HTML table.
	MaxRows (see page 10312)	Specifies the maximum number of rows that appear in the HTML table.
	RowAttributes (see page 10312)	Specifies the display attributes of all the rows in the HTML table.
	TableAttributes (see page 10313)	Specifies the display attributes that pertain to the entire HTML table produced by TDSTableProducer (see page 10290).

54.1.2.5.1 TDSTableProducer.Caption Property

Specifies the caption string that appears with the HTML table.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Set Caption to add a caption to the HTML table that represents the dataset. Caption indicates the text that should appear inside the HTML CAPTION tag.

See Also

CaptionAlignment (see page 10308)

OnGetTableCaption (see page 10314)

Embedding Tables in HTML Documents

54.1.2.5.2 TDSTableProducer.CaptionAlignment Property

Specifies where the Caption (see page 10307) string should appear relative to the HTML table.

Pascal

```
property CaptionAlignment: THTMLCaptionAlignment;
```

C++

```
__property THTMLCaptionAlignment CaptionAlignment;
```

Description

Set CaptionAlignment to specify whether the Caption (see page 10307) should appear above or below the HTML table.

CaptionAlignment is the value of the ALIGN option on the HTML CAPTION tag.

See Also

Caption (see page 10307)

Embedding Tables in HTML Documents

54.1.2.5.3 TDSTableProducer.Columns Property

Describes the field bindings and display attributes of the columns in the HTML table.

Pascal

```
property Columns: THTMLTableColumns;
```

C++

```
__property THTMLTableColumns Columns;
```

Description

Use Columns to read or set the field bindings and display attributes of the columns in the HTML table. Each THTMLTableColumn (see page 10318) object specifies the display attributes of the corresponding column. Each column object also specifies the name of the field the column represents. The field binding of a column designates a field within the dataset specified by the DataSet (see page 10309) property.

Columns can be set at design time through a property editor, or programmatically at runtime.

See Also

DataSet (see page 10309)

DefineFieldMap (see page 10298)

Editor (see page 10310)

RowAttributes (see page 10312)

SetColumns (see page 10304)

StoreColumns (see page 10305)

TableAttributes (see page 10313)

THTMLTableColumn (see page 10318)

THTMLTableColumns (see page 10327)

Specifying the Columns

54.1.2.5.4 TDSTableProducer.DataLink Property

Manages the link to the dataset.

Pascal

```
property DataLink: THTTPDataLink;
```

C++

```
__property THTTPDataLink DataLink;
```

Description

Applications cannot use the DataLink property. It is exposed as a protected property so that descendant HTML producers can customize their response to data events.

See Also

DataSet (see page 10309)

InternalDataSource (see page 10311)

54.1.2.5.5 TDSTableProducer.DataSet Property

Specifies the dataset from which the data in the HTML table is taken.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Set DataSet to specify where the TDSTableProducer (see page 10290) should look for the data it displays in the HTML table. The individual field bindings specified by the THTMLTableColumn (see page 10318) objects in the Columns (see page 10308) property refer to fields in DataSet.

Specify the data binding of a TDSTableProducer (see page 10290) using the DataSet property rather than through a DataSource (see page 10309) property. TDSTableProducer (see page 10290) objects do not use an external TDataSource object.

See Also

DataSource (see page 10309)

InternalDataSource (see page 10311)

Columns (see page 10308)

GetDataSet (see page 10302)

SetDataSet (see page 10305)

FieldName (see page 10326)

Using TDataSetTableProducer

54.1.2.5.6 TDSTableProducer.DataSource Property

Represents the link to the DataSet (see page 10309) of the TDSTableProducer (see page 10290).

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Do not use the DataSource property. The data source of a TDSTableProducer (see page 10290) object is an internally generated TDataSource object that is used for internal interfaces that do not work directly with a TDataSet object.

The InternalDataSource (see page 10311) property provides read-only access to the data source represented by DataSource.

See Also

DataSet (see page 10309)

GetDataSource (see page 10302)

InternalDataSource (see page 10311)

SetDataSource (see page 10305)

54.1.2.5.7 TDSTableProducer.Editor Property

Writes changes made in the property editor at design time to the TDSTableProducer (see page 10290) object.

Pascal

```
property Editor: TDSTableProducerEditor;
```

C++

```
__property TDSTableProducerEditor Editor;
```

Description

Descendants of TDSTableProducer (see page 10290) set the Editor property to the TDSTableProducer (see page 10290) Editor object that connects their implementation with a common property editor. Editor allows a single property editor to represent the properties of different table producers.

54.1.2.5.8 TDSTableProducer.Footer Property

Specifies a set of HTML commands that appear in the HTML document following the HTML table.

Pascal

```
property Footer: TStrings;
```

C++

```
__property TStrings Footer;
```

Description

Set Footer to embed the representation of DataSet (see page 10309) in a larger document. The Content method of TDSTableProducer (see page 10290) descendants returns any Header (see page 10311) commands, followed by the HTML table that represents the DataSet (see page 10309), followed by any Footer commands.

See Also

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

DataSet (see page 10309)

Header (see page 10311)

Embedding Tables in HTML Documents

54.1.2.5.9 TDSTableProducer.Header Property

Specifies a set of HTML commands that appear in the HTML document preceding the HTML table.

Pascal

```
property Header: TStrings;
```

C++

```
__property TStrings Header;
```

Description

Set Header to embed the representation of DataSet (see page 10309) in a larger document. The Content method of TDSTableProducer (see page 10290) descendants returns any Header commands, followed by the HTML table that represents the DataSet (see page 10309), followed by any Footer (see page 10310) commands.

See Also

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

DataSet (see page 10309)

Footer (see page 10310)

Embedding Tables in HTML Documents

54.1.2.5.10 TDSTableProducer.InternalDataSource Property

Represents the link to the dataset.

Pascal

```
property InternalDataSource: TDataSource;
```

C++

```
__property TDataSource InternalDataSource;
```

Description

Descendants of TDSTableProducer (see page 10290) read InternalDataSource to access the properties and methods of the data source. Unlike other data-aware components, TDSTableProducer (see page 10290) does not use an external TDataSource object.

See Also

DataSet (see page 10309)

54.1.2.5.11 TDSTableProducer.MaxRows Property

Specifies the maximum number of rows that appear in the HTML table.

Pascal

```
property MaxRows: Integer;
```

C++

```
__property int MaxRows;
```

Description

Set MaxRows to change the upper limit on the size of the HTML table produced to represent the DataSet (see page 10309). MaxRows is initialized to 20.

If DataSet (see page 10309) contains more than MaxRows records, only the first MaxRows records appear in the HTML commands produced by Content. If DataSet (see page 10309) contains fewer than MaxRows records, the HTML table produced by TDSTableProducer (see page 10290) will contain only enough rows to display all the records in DataSet (see page 10309).

See Also

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

DataSet (see page 10309)

Specifying the Row Attributes

54.1.2.5.12 TDSTableProducer.RowAttributes Property

Specifies the display attributes of all the rows in the HTML table.

Pascal

```
property RowAttributes: THTMLTableRowAttributes;
```

C++

```
__property THTMLTableRowAttributes RowAttributes;
```

Description

Set RowAttributes to a THTMLTableRowAttributes object that describes the display attributes of the rows in the HTML table. Setting RowAttributes calls the Assign method of the THTMLTableRowAttributes object stored internally by TDSTableProducer (see page 10290). Thus, the memory for any row attributes object assigned to the RowAttributes property is never owned or freed by TDSTableProducer (see page 10290).

Note: Not all the properties of the row attributes object are copied when one row attributes object is assigned to another. To ensure that all the attributes represented by RowAttributes are set appropriately, read RowAttributes and set the individual properties of the THTMLTableRowAttributes object.

See Also

TableAttributes (see page 10313)

Title (see page 10327)

AssignTo (see page 15297)

Specifying the Row Attributes

54.1.2.5.13 TDSTableProducer.TableAttributes Property

Specifies the display attributes that pertain to the entire HTML table produced by TDSTableProducer (see page 10290).

Pascal

```
property TableAttributes: THTMLTableAttributes;
```

C++

```
__property THTMLTableAttributes TableAttributes;
```

Description

Set TableAttributes to a THTMLTableAttributes object that describes the display attributes for the HTML TABLE tag. Setting TableAttributes calls the Assign method of the THTMLTableAttributes object stored internally by TDSTableProducer (see page 10290). Thus, the memory for any table attributes object assigned to the TableAttributes property is never owned or freed by TDSTableProducer (see page 10290).

Note: Not all the properties of the table attributes object are copied when one table attributes object is assigned to another. To ensure that all the attributes represented by TableAttributes are set appropriately, read TableAttributes and set the individual properties of the THTMLTableAttributes object.

See Also

RowAttributes (see page 10312)

AssignTo (see page 15291)

Title (see page 10327)

Specifying the Table Attributes

54.1.2.6 TDSTableProducer Events

The events of the TDSTableProducer class are listed here.

Protected Events

	Name	Description
	OnCreateContent (see page 10313)	Occurs in descendants of TDSTableProducer (see page 10290) when the Content method needs to generate the HTML table image.
	OnFormatCell (see page 10314)	Occurs when the HTML for a specified cell specified is about to be generated.
	OnGetTableCaption (see page 10314)	Occurs when the HTML table producer is about to generate the HTML for the table caption.

54.1.2.6.1 TDSTableProducer.OnCreateContent Event

Occurs in descendants of TDSTableProducer (see page 10290) when the Content method needs to generate the HTML table image.

Pascal

```
property OnCreateContent: TCreateContentEvent;
```

C++

```
__property TCreateContentEvent OnCreateContent;
```

Description

Write an OnCreateContent event handler to conditionally prevent the Content method from producing any HTML commands. Set the Continue parameter to false to make Content return an empty string.

See Also

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

docreatecontent (see page 10299)

54.1.2.6.2 TDSTableProducer.OnFormatCell Event

Occurs when the HTML for a specified cell specified is about to be generated.

Pascal

```
property OnFormatCell: THTMLFormatCellEvent;
```

C++

```
__property THTMLFormatCellEvent OnFormatCell;
```

Description

Write an OnFormatCell event handler to change the appearance or contents of individual cells in the HTML table from the default values produced by TDSTableProducer (see page 10290).

See Also

Columns (see page 10308)

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

DoFormatCell (see page 10299)

DisplayText (see page 7162)

Align (see page 10324)

BgColor (see page 10325)

Custom (see page 10325)

VAlign (see page 10327)

54.1.2.6.3 TDSTableProducer.OnGetTableCaption Event

Occurs when the HTML table producer is about to generate the HTML for the table caption.

Pascal

```
property OnGetTableCaption: THTMLGetTableCaptionEvent;
```

C++

```
__property THTMLGetTableCaptionEvent OnGetTableCaption;
```

Description

Write an OnGetTableCaption event handler to change the value or placement of the HTML table caption. The Caption (see page 10307) and Alignment parameters are originally set to the values of the Caption (see page 10307) and CaptionAlignment (see page 10308) properties, respectively. The OnGetTableCaption event handler can change these values to alter the caption for the HTML table returned by the Content method.

See Also

Caption (see page 10307)

CaptionAlignment (see page 10308)

Content (see page 15106)

DoGetCaption (see page 10300)

54.1.3 TDSTableProducerEditor Class

TDSTableProducer (see page 10290) is the base class for the helper object used by TDSTableProducer (see page 10290) objects to communicate with a property editor at design time.

Class Hierarchy

Pascal

```
TDSTableProducerEditor = class;
```

C++

```
class TDSTableProducerEditor;
```

File

DBWeb

Description

Component writers can derive objects from TDSTableProducer (see page 10290) to enable new descendants of TDSTableProducer (see page 10290) to take advantage of the property editors in the IDE. TDSTableProducerEditor introduces methods to communicate changes between a TDSTableProducer (see page 10290) object and the property editor.

See Also

TDSTableProducer (see page 10290)

54.1.3.1 TDSTableProducerEditor Members

The following tables list the members exposed by TDSTableProducerEditor.

Public Methods

	Name	Description
	Changed (see page 10317)	Provides the interface for a method that responds to change notifications from the TDSTableProducer (see page 10290) object.
	Create (see page 10316)	Creates an instance of TDSTableProducerEditor (see page 10315).
	Destroy (see page 10316)	Destroys an instance of TDSTableProducerEditor (see page 10315).

	PostChange (see page 10317)	Provides an interface for a method that responds to update notifications from the columns of a TDSTableProducer (see page 10290) object.

Public Properties

	Name	Description
	DataSource (see page 10318)	Indicates the DataSource property of the associated TDSTableProducer (see page 10290).
	DSTableProducer (see page 10318)	Indicates the TDSTableProducer (see page 10290) object that uses TDSTableProducerEditor (see page 10315) to communicate with its property editor at design time.

54.1.3.2 TDSTableProducerEditor.Create Constructor

Creates an instance of TDSTableProducerEditor (see page 10315).

Pascal

```
constructor Create(DSTableProducer: TDSTableProducer);
```

C++

```
__fastcall TDSTableProducerEditor(TDSTableProducer DSTableProducer);
```

Description

Do not call Create directly in an application. A TDSTableProducerEditor (see page 10315) is created at design time when needed.

Create initializes the DSTableProducer (see page 10318) property, and sets the Editor property in the TDSTableProducer (see page 10290) object passed as the DSTableProducer (see page 10318) parameter.

See Also

DSTableProducer (see page 10318)

Editor (see page 10310)

54.1.3.3 TDSTableProducerEditor.Destroy Destructor

Destroys an instance of TDSTableProducerEditor (see page 10315).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDSTableProducerEditor();
```

Description

Do not call Destroy from an application. A TDSTableProducerEditor (see page 10315) object is freed automatically at design time. Destroy sets the Editor property of the associated TDSTableProducer (see page 10290) object to nil.

See Also

Free (see page 22300)

Editor (see page 10310)

54.1.3.4 TDSTableProducerEditor Methods

The methods of the TDSTableProducerEditor class are listed here.

Public Methods

	Name	Description
	Changed (see page 10317)	Provides the interface for a method that responds to change notifications from the TDSTableProducer (see page 10290) object.
	PostChange (see page 10317)	Provides an interface for a method that responds to update notifications from the columns of a TDSTableProducer (see page 10290) object.

54.1.3.4.1 TDSTableProducerEditor.Changed Method

Provides the interface for a method that responds to change notifications from the TDSTableProducer (see page 10290) object.

Pascal

```
procedure Changed; virtual;
```

C++

```
virtual __fastcall Changed();
```

Description

The Changed method for TDSTableProducerEditor (see page 10315) does nothing. It exists so that descendants of TDSTableProducerEditor (see page 10315) can perform any necessary updates when the TDSTableProducer (see page 10290) object changes. TDSTableProducer (see page 10290) calls Changed when any of its properties change.

See Also

PostChange (see page 10317)

TDSTableProducer (see page 10290)

54.1.3.4.2 TDSTableProducerEditor.PostChange Method

Provides an interface for a method that responds to update notifications from the columns of a TDSTableProducer (see page 10290) object.

Pascal

```
procedure PostChange; virtual;
```

C++

```
virtual __fastcall PostChange();
```

Description

The PostChange method for TDSTableProducerEditor (see page 10315) does nothing. It exists so that descendants of TDSTableProducerEditor (see page 10315) can perform any necessary updates when the columns of the TDSTableProducer (see page 10290) object change. THTMLTableColumns (see page 10327) calls PostChange when its columns are changed.

See Also

Changed (see page 10317)

THTMLTableColumns (see page 10327)

54.1.3.5 TDSTableProducerEditor Properties

The properties of the TDSTableProducerEditor class are listed here.

Public Properties

	Name	Description
	DataSource (see page 10318)	Indicates the DataSource property of the associated TDSTableProducer (see page 10290).
R	DSTableProducer (see page 10318)	Indicates the TDSTableProducer (see page 10290) object that uses TDSTableProducerEditor (see page 10315) to communicate with its property editor at design time.

54.1.3.5.1 TDSTableProducerEditor.DataSource Property

Indicates the DataSource property of the associated TDSTableProducer (see page 10290).

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

Applications should not directly change the DataSource property. DataSource allows the property editor to read or change the internal data source of the TDSTableProducer (see page 10290) object.

See Also

DataSource (see page 10309)

54.1.3.5.2 TDSTableProducerEditor.DSTableProducer Property

Indicates the TDSTableProducer (see page 10290) object that uses TDSTableProducerEditor (see page 10315) to communicate with its property editor at design time.

Pascal

```
property DSTableProducer: TDSTableProducer;
```

C++

```
__property TDSTableProducer DSTableProducer;
```

Description

Read TDSTableProducer (see page 10290) to obtain direct access to the TDSTableProducer (see page 10290) object that uses TDSTableProducerEditor (see page 10315).

See Also

TDSTableProducer (see page 10290)

54.1.4 THTMLTableColumn Class

THTMLTableColumn represents the display attributes and field binding of a column in the HTML representation of a dataset.

Class Hierarchy

Pascal

```
THTMLTableColumn = class(TCollectionItem);
```

C++

```
class THTMLTableColumn : public TCollectionItem;
```

File

DBWeb

Description

Use THTMLTableColumn to specify the appearance or value of a column in a table produced by TDSTableProducer (see page 10290). Applications can set the properties of the columns programmatically at runtime, or use the Columns editor at design time.

See Also

THTMLTableColumns (see page 10327)

TDSTableProducer (see page 10290)

Specifying the Columns

54.1.4.1 THTMLTableColumn Members

The following tables list the members exposed by THTMLTableColumn.

Protected Methods

	Name	Description
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetDisplayName (see page 10322)	Returns the name of the collection item as it appears in the collection editor.
	GeTDSTableProducer (see page 10323)	This is GeTDSTableProducer, a member of class THTMLTableColumn.
V	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	IsStored (see page 10323)	This is IsStored, a member of class THTMLTableColumn.
V	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
V	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
V	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	AssignTo (see page 10322)	Copies the properties of a THTMLTableColumn (see page 10318) object to another THTMLTableColumn (see page 10318) object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 10320)	Creates an instance of THTMLTableColumn (see page 10318).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 10321)	Destroys an instance of THTMLTableColumn (see page 10318).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RestoreDefaults (see page 10323)	Changes all properties of the attributes object back to their initial settings.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Update (see page 10324)	Ensures that the Field (see page 10326) property reflects the underlying dataset.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	DSTableProducer (see page 10325)	Indicates the table producer component that uses this column.
	Field (see page 10326)	Indicates the field component associated with the field binding of the column.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	Align (see page 10324)	Specifies the horizontal alignment of the contents of the cells in the column.
	BgColor (see page 10325)	Specifies the background color of the cells in the column.
	Custom (see page 10325)	Specifies a tag option and its value in the HTML TD tags for the cells in the column.
	FieldName (see page 10326)	Specifies the name of the field to which the column is bound.
	Title (see page 10327)	Specifies the display attributes to be added to the header cell that appears at the top of the column.
	VAlign (see page 10327)	Specifies the vertical alignment of the contents of the cells in the column.

54.1.4.2 THTMLTableColumn.Create Constructor

Creates an instance of THTMLTableColumn (see page 10318).

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall THTMLTableColumn(TCollection Collection);
```

Description

Do not call Create to create and initialize a THTMLTableColumn (see page 10318) object. Columns created using the Columns editor at design time are created automatically. At runtime, create THTMLTableColumn (see page 10318) objects using the Add or Insert methods of the THTMLTableColumns (see page 10327) object that manages them.

After allocating the THTMLTableColumn (see page 10318) object, Create tells the associated TDSTableProducer (see page 10290) object to update its layout.

See Also

Create (see page 2757)

BeginLayout (see page 10297)

EndLayout (see page 10300)

54.1.4.3 THTMLTableColumn.Destroy Destructor

Destroys an instance of THTMLTableColumn (see page 10318).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THTMLTableColumn();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the THTMLTableColumn (see page 10318) reference is not nil, and only then calls Destroy. Do not explicitly free a THTMLTableColumn (see page 10318) that is included in a THTMLTableColumns (see page 10327) object. THTMLTableColumns (see page 10327) frees the individual column objects it manages.

See Also

Free (see page 22300)

THTMLTableColumns (see page 10327)

54.1.4.4 THTMLTableColumn Methods

The methods of the THTMLTableColumn class are listed here.

Protected Methods

	Name	Description
	GetDisplayName (see page 10322)	Returns the name of the collection item as it appears in the collection editor.
	GeTDSTableProducer (see page 10323)	This is GeTDSTableProducer, a member of class THTMLTableColumn.
	IsStored (see page 10323)	This is IsStored, a member of class THTMLTableColumn.

Public Methods

	Name	Description
	AssignTo (see page 10322)	Copies the properties of a THTMLTableColumn (see page 10318) object to another THTMLTableColumn (see page 10318) object.
	RestoreDefaults (see page 10323)	Changes all properties of the attributes object back to their initial settings.
	Update (see page 10324)	Ensures that the Field (see page 10326) property reflects the underlying dataset.

54.1.4.4.1 THTMLTableColumn.AssignTo Method

Copies the properties of a THTMLTableColumn (see page 10318) object to another THTMLTableColumn (see page 10318) object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Applications cannot call the AssignTo method. To assign the properties of a THTMLTableColumn (see page 10318) object to another object, use the Assign method of the destination object.

When the Assign method of another object specifies a THTMLTableColumn (see page 10318) object as the source of the assignment, the AssignTo method of the THTMLTableColumn (see page 10318) object is called. AssignTo checks whether the Dest parameter is another THTMLTableColumn (see page 10318) object, and if so, copies the values of all properties except Custom (see page 10325) and DisplayName. If Dest is any other class of object, the inherited method is called.

See Also

Custom (see page 10325)

Assign (see page 2930)

AssignTo (see page 2931)

54.1.4.4.2 THTMLTableColumn.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName (see page 2761)

GetNamePath (see page 2759)

54.1.4.4.3 THTMLTableColumn.GeTDSTableProducer Method

Pascal

```
function GeTDSTableProducer: TDSTableProducer;
```

C++

```
__fastcall TDSTableProducer GeTDSTableProducer();
```

Description

This is GeTDSTableProducer, a member of class THTMLTableColumn.

54.1.4.4.4 THTMLTableColumn.IsStored Method

Pascal

```
function IsStored: Boolean;
```

C++

```
__fastcall Boolean IsStored();
```

Description

This is IsStored, a member of class THTMLTableColumn.

54.1.4.4.5 THTMLTableColumn.RestoreDefaults Method

Changes all properties of the attributes object back to their initial settings.

Pascal

```
procedure RestoreDefaults;
```

C++

```
__fastcall RestoreDefaults();
```

Description

Call RestoreDefaults to undo any changes that were made to the table element attributes either programmatically, or using the Columns editor at design time. RestoreAttributes sets

Align (see page 10324) to haDefault, and VAlign (see page 10327) to haVDefault.

BgColor (see page 10325) to an empty string.

Custom (see page 10325) to an empty string.

It also restores the default settings for the THTMLTableHeaderAttributes object that represents the title of the column.

See Also

Align (see page 10324)

BgColor (see page 10325)

Custom (see page 10325)

RestoreDefaults (see page 15303)

VAlign (see page 10327)

54.1.4.4.6 THTMLTableColumn.Update Method

Ensures that the Field (see page 10326) property reflects the underlying dataset.

Pascal

```
procedure Update;
```

C++

```
__fastcall Update();
```

Description

Call Update to ensure that the Field (see page 10326) property reflects any changes to the fields in the underlying dataset.

See Also

Field (see page 10326)

54.1.4.5 THTMLTableColumn Properties

The properties of the THTMLTableColumn class are listed here.

Public Properties

	Name	Description
R	DSTableProducer (see page 10325)	Indicates the table producer component that uses this column.
	Field (see page 10326)	Indicates the field component associated with the field binding of the column.

Published Properties

	Name	Description
	Align (see page 10324)	Specifies the horizontal alignment of the contents of the cells in the column.
	BgColor (see page 10325)	Specifies the background color of the cells in the column.
	Custom (see page 10325)	Specifies a tag option and its value in the HTML TD tags for the cells in the column.
	FieldName (see page 10326)	Specifies the name of the field to which the column is bound.
	Title (see page 10327)	Specifies the display attributes to be added to the header cell that appears at the top of the column.
	VAlign (see page 10327)	Specifies the vertical alignment of the contents of the cells in the column.

54.1.4.5.1 THTMLTableColumn.Align Property

Specifies the horizontal alignment of the contents of the cells in the column.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Set Align to indicate the desired horizontal alignment of text in the cells of the column. Align indicates the desired value of the Align option of the HTML TD tag.

The default value of Align is haDefault, which means the web browser that reads the HTML document horizontally positions the contents of cells wherever it wishes.

See Also

VAlign (see page 10327)

Align (see page 15292)

54.1.4.5.2 THTMLTableColumn.BgColor Property

Specifies the background color of the cells in the column.

Pascal

```
property BgColor: THTMLBgColor;
```

C++

```
__property THTMLBgColor BgColor;
```

Description

Use BgColor to indicate a value for the BgColor option of the HTML TD tag. The value of BgColor is the name of color. This can be one of the following strings:

Use the BgColor property of the THTMLTableAttributes object to set a background color for the entire table. Use the BgColor property of THTMLTableColumn (see page 10318) to make a column stand out from the rest of the table.

See Also

BgColor (see page 15293)

54.1.4.5.3 THTMLTableColumn.Custom Property

Specifies a tag option and its value in the HTML TD tags for the cells in the column.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to represent one or more options to be added to an HTML tag. Each option and its value should have the form

```
OPTIONNAME=Value
```

If Custom specifies more than one option, each object should be separated from the others by spaces.

See Also

Custom (see page 15307)

54.1.4.5.4 THTMLTableColumn.DSTableProducer Property

Indicates the table producer component that uses this column.

Pascal

```
property DSTableProducer: TDSTableProducer;
```

C++

```
__property TDSTableProducer DSTableProducer;
```

Description

Read DSTableProducer to obtain information about the component that uses this object to represent the attributes of a column in the HTML table it produces. For example, to access the dataset which contains the field for this column, use the Dataset property of DSTableProducer.

See Also

DataSet (see page 10309)

54.1.4.5.5 THTMLTableColumn.Field Property

Indicates the field component associated with the field binding of the column.

Pascal

```
property Field: TField;
```

C++

```
__property TField Field;
```

Description

Read Field to get direct access to the field component for the column. Usually, Field is a field component maintained by the dataset. In some cases, however, Field is maintained directly by THTMLTableColumn (see page 10318), such as when the column represents a calculated field. Do not set the Field property unless the TField object is obtained from the dataset of the TDSTableProducer (see page 10290) that uses this THTMLTableColumn (see page 10318) object, or is a calculated field based on that dataset.

Setting Field automatically updates the FieldName (see page 10326) property.

See Also

FieldName (see page 10326)

DataSet (see page 10309)

54.1.4.5.6 THTMLTableColumn.FieldName Property

Specifies the name of the field to which the column is bound.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;
```

Description

Set FieldName to specify the field binding for the column in the HTML table. FieldName identifies a particular field component in the dataset of the TDSTableProducer (see page 10290) object that uses this THTMLTableColumn (see page 10318) object.

Setting FieldName automatically updates the Field (see page 10326) property.

See Also

Field (see page 10326)

FieldName (see page 7164)

DataSet (see page 10309)

54.1.4.5.7 THTMLTableColumn.Title Property

Specifies the display attributes to be added to the header cell that appears at the top of the column.

Pascal

```
property Title: THTMLTableHeaderAttributes;
```

C++

```
__property THTMLTableHeaderAttributes Title;
```

Description

Set Title to a THTMLTableHeaderAttributes object that describes the display attributes for the HTML TH tag for the header cell at the top of the column. Setting Title calls the Assign method of the THTMLTableHeaderAttributes object stored internally by THTMLTableColumn (see page 10318). Thus, the memory for any table header attributes object assigned to the Title property is never owned or freed by the HTML table column.

Note: Not all the properties of the table header attributes object are copied when one table attributes object is assigned to another. To ensure that all the attributes represented by Title are set appropriately, read Title and set the individual properties of the THTMLTableHeaderAttributes object.

See Also

RowAttributes (see page 10312)

AssignTo (see page 15297)

54.1.4.5.8 THTMLTableColumn.VAlign Property

Specifies the vertical alignment of the contents of the cells in the column.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

Set VAlign to indicate the desired vertical alignment of text in the cells of the column. VAlign indicates the desired value of the VAlign option of the HTML <TD> tag.

The default value of VAlign is haVDefault, which means the web browser that reads the HTML document vertically positions the contents of cells wherever it wishes.

See Also

Align (see page 10324)

54.1.5 THTMLTableColumns Class

THTMLTableColumns is a collection of THTMLTableColumn (see page 10318) objects that describe the display attributes and field bindings of the columns in an HTML representation of tabular data.

Class Hierarchy

Pascal

```
THTMLTableColumns = class(TCollection);
```

C++

```
class THTMLTableColumns : public TCollection;
```

File

DBWeb

Description

THTMLTableColumns is tailored for use by TDSTableProducer (see page 10290). It stores and manages changes to the THTMLTableColumn (see page 10318) objects that describe the columns of the HTML table produced by TDSTableProducer (see page 10290). THTMLTableColumns informs the table producer of changes to the column bindings so the table producer can recompute its layout.

See Also

TDSTableProducer (see page 10290)

THTMLTableColumn (see page 10318)

54.1.5.1 THTMLTableColumns Members

The following tables list the members exposed by THTMLTableColumns.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
V	GetAttr (see page 10331)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
V	GetAttrCount (see page 10331)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
V	GetItemAttr (see page 10331)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 10332)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 10333)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 10330)	Adds a new column descriptor to the end of the Items (see page 10334) array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 10330)	This is Create, a member of class THTMLTableColumns.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
	Destroy (see page 2735)	Destroys the collection and each item in it.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
	RebuildColumns (see page 10332)	Replaces all the column descriptors in the Items (see page 10334) array with new column descriptors that exactly match the fields in the dataset of DSTableProducer (see page 10334).
	RestoreDefaults (see page 10333)	Restores the default properties of every column in the Items (see page 10334) property array.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.

R	DSTableProducer (see page 10334)	Indicates the table producer that uses THTMLTableColumns (see page 10327) to store and manage column descriptions.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 10334)	Lists the THTMLTableColumn (see page 10318) objects that THTMLTableColumns (see page 10327) stores and manages for its TDSTableProducer (see page 10290) object.
	State (see page 10335)	Indicates whether the individual column descriptors are generated dynamically or statically stored.

54.1.5.2 THTMLTableColumns.Create Constructor

Pascal

```
constructor Create(DSTableProducer: TDSTableProducer; ColumnClass: THTMLTableColumnClass);
```

C++

```
__fastcall THTMLTableColumns(TDSTableProducer DSTableProducer, THTMLTableColumnClass
ColumnClass);
```

Description

This is Create, a member of class THTMLTableColumns.

54.1.5.3 THTMLTableColumns Methods

The methods of the THTMLTableColumns class are listed here.

Protected Methods

	Name	Description
V	GetAttr (see page 10331)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
V	GetAttrCount (see page 10331)	Returns the number of custom attributes associated with items in the collection.
V	GetItemAttr (see page 10331)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 10332)	Returns the owner of an object.
V	Update (see page 10333)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 10330)	Adds a new column descriptor to the end of the Items (see page 10334) array.
	RebuildColumns (see page 10332)	Replaces all the column descriptors in the Items (see page 10334) array with new column descriptors that exactly match the fields in the dataset of DSTableProducer (see page 10334).
	RestoreDefaults (see page 10333)	Restores the default properties of every column in the Items (see page 10334) property array.

54.1.5.3.1 THTMLTableColumns.Add Method

Adds a new column descriptor to the end of the Items (see page 10334) array.

Pascal

```
function Add: THTMLTableColumn;
```

C++

```
__fastcall THTMLTableColumn Add();
```

Description

Call Add to add a column descriptor to the collection of columns shown in the HTML table generated by TDSTableProducer (

see page 10290). Applications should only call Add when the value of the State (see page 10335) property is csCustom.

See Also

State (see page 10335)

TDSTableProducer (see page 10290)

54.1.5.3.2 THTMLTableColumns.GetAttr Method

Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.

Pascal

```
function GetAttr(Index: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetAttr(int Index);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. The GetAttr method returns the name of an attribute.

Index identifies the attribute whose name is requested. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

As implemented by TCollection, GetAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttrCount (see page 2744)

GetItemAttr (see page 2745)

54.1.5.3.3 THTMLTableColumns.GetAttrCount Method

Returns the number of custom attributes associated with items in the collection.

Pascal

```
function GetAttrCount: Integer; override;
```

C++

```
virtual __fastcall int GetAttrCount();
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetAttrCount returns the number of distinct attributes assigned to each item in the collection.

As implemented in TCollection, GetAttrCount always returns 0, because TCollection defines no custom attributes.

See Also

GetAttr (see page 2743)

GetItemAttr (see page 2745)

54.1.5.3.4 THTMLTableColumns.GetItemAttr Method

Returns the value of a custom attribute assigned to one of the collection's items.

Pascal

```
function GetItemAttr(Index: Integer; ItemIndex: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetItemAttr(int Index, int ItemIndex);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetItemAttr returns the value of one of these attributes for a specified item in the collection.

Index identifies which attribute's value is desired. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

ItemIndex identifies the item whose attribute value is desired. This is an index into the Items property array.

As implemented in TCollection, GetItemAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttr (see page 2743)

GetAttrCount (see page 2744)

54.1.5.3.5 THTMLTableColumns.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

54.1.5.3.6 THTMLTableColumns.RebuildColumns Method

Replaces all the column descriptors in the Items (see page 10334) array with new column descriptors that exactly match the fields in the dataset of DSTableProducer (see page 10334).

Pascal

```
procedure RebuildColumns;
```

C++

```
__fastcall RebuildColumns();
```

Description

Call RebuildColumns to set the collection of column descriptors to the default state which matches the order and number of fields in the dataset, and which imposes no special display attributes. RebuildColumns returns the collection of column descriptors to the state that is dynamically maintained when the State (see page 10335) property is csDefault.

See Also

DSTableProducer (see page 10334)

State (see page 10335)

DataSet (see page 10309)

54.1.5.3.7 THTMLTableColumns.RestoreDefaults Method

Restores the default properties of every column in the Items (see page 10334) property array.

Pascal

```
procedure RestoreDefaults;
```

C++

```
__fastcall RestoreDefaults();
```

Description

At design time, RestoreDefaults is called when the Restore defaults button is pressed in the Response editor.

See Also

Items (see page 10334)

RestoreDefaults (see page 10323)

54.1.5.3.8 THTMLTableColumns.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed (see page 2740)

Changed (see page 2758)

54.1.5.4 THTMLTableColumns Properties

The properties of the THTMLTableColumns class are listed here.

Public Properties

	Name	Description
	DSTableProducer (see page 10334)	Indicates the table producer that uses THTMLTableColumns (see page 10327) to store and manage column descriptions.
	Items (see page 10334)	Lists the THTMLTableColumn (see page 10318) objects that THTMLTableColumns (see page 10327) stores and manages for its TDSTableProducer (see page 10290) object.
	State (see page 10335)	Indicates whether the individual column descriptors are generated dynamically or statically stored.

54.1.5.4.1 THTMLTableColumns.DSTableProducer Property

Indicates the table producer that uses THTMLTableColumns (see page 10327) to store and manage column descriptions.

Pascal

```
property DSTableProducer: TDSTableProducer;
```

C++

```
__property TDSTableProducer DSTableProducer;
```

Description

Read DSTableProducer to gain direct access to the table producer that uses the THTMLTableColumns (see page 10327) object. THTMLTableColumns (see page 10327) uses this property to inform the table producer of changes to the columns so that it can recompute its layout.

54.1.5.4.2 THTMLTableColumns.Items Property

Lists the THTMLTableColumn (see page 10318) objects that THTMLTableColumns (see page 10327) stores and manages for its TDSTableProducer (see page 10290) object.

Pascal

```
property Items [Index: Integer]: THTMLTableColumn;
```

C++

```
__property THTMLTableColumn Items[int Index];
```

Description

Use Items to access the individual column descriptors managed by THTMLTableColumns (see page 10327). Items is an indexed array, where the first column descriptor has index 0, the second column descriptor has index 1, and so on.

Note: In Delphi, Items is the default property of THTMLTableColumns (see page 10327). This means that applications do not need to specify the Items property name when referring to the elements of the Items array.

Note: In C++, you can use the [] operator to obtain an effect similar to using Items as the default property in Delphi.

See Also

Count (see page 2749)

Columns (see page 10308)

THTMLTableColumn (see page 10318)

operator_sb

54.1.5.4.3 THTMLTableColumns.State Property

Indicates whether the individual column descriptors are generated dynamically or statically stored.

Pascal

```
property State: THTMLColumnState;
```

C++

```
__property THTMLColumnState State;
```

Description

Set State to csDefault to indicate that the column descriptors should be generated dynamically. Dynamically generated column descriptors impose no special display attributes on the columns of the HTML table, and correspond exactly in order and number to the fields of the dataset. Setting State to csDefault replaces all the column descriptors in the collection with dynamically generated defaults.

Set State to csCustom before customizing the column descriptors by adding display attributes, reordering the fields, adding columns for calculated fields, or including only a subset of the fields in the dataset. When State is csCustom, the column descriptors are saved with the TDSTableProducer (see page 10290) object.

Note: Using the Columns editor to customize the column descriptors automatically sets State to csCustom.

See Also

RebuildColumns (see page 10332)

Fields (see page 7023)

StoreColumns (see page 10305)

54.1.6 THTTPDataLink Class

THTTPDataLink helps TDSTableProducer (see page 10290) manage its link to the data source.

Class Hierarchy

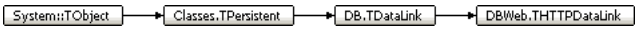

Pascal

```
THTTPDataLink = class(TDataLink);
```

C++

```
class THTTPDataLink : public TDataLink;
```

File

DBWeb

Description

THTTPDataLink is tailored to work with a TDSTableProducer (see page 10290) object. It should only be used by the TDSTableProducer (see page 10290) class.

See Also

TDataSource (see page 7070)

TDSTableProducer (see page 10290)

54.1.6.1 THTTPDataLink Members

The following tables list the members exposed by THTTPDataLink.

Protected Methods

	Name	Description
	ActiveChanged (see page 10339)	Informs the table producer when the Active property changes.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CheckBrowseMode (see page 6880)	Allows the data-aware object to respond just before an event that changes the dataset state.
	DataEvent (see page 6881)	Responds to various events that occur while working with data.
	DataSetChanged (see page 10340)	Informs the associated table producer of changes to the dataset.
	DataSetScrolled (see page 10340)	Ignores scrolling events on the dataset.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EditingChanged (see page 10341)	Ignores changes to the editing state of the dataset.
	FocusControl (see page 10341)	Discards requests to set focus to the indicated field.
	GetActiveRecord (see page 6884)	This is GetActiveRecord, a member of class TDataLink.
	GetBOF (see page 6884)	This is GetBOF, a member of class TDataLink.
	GetBufferCount (see page 6884)	This is GetBufferCount, a member of class TDataLink.
	GetEOF (see page 6885)	This is GetEOF, a member of class TDataLink.
	GetMappedIndex (see page 10342)	Returns the index of the field component in the dataset that corresponds to the index of the field component within the Fields (see page 10345) array.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRecordCount (see page 6885)	This is GetRecordCount, a member of class TDataLink.
	LayoutChanged (see page 10342)	Informs TDSTableProducer (see page 10290) of changes in the layout of columns at design time.
	MoveBy (see page 6885)	Repositions the dataset to another record relative to its active record.
	RecordChanged (see page 10343)	Ignores changes to the contents of the current record or field of the data source.
	SetActiveRecord (see page 6886)	This is SetActiveRecord, a member of class TDataLink.
	SetBufferCount (see page 6887)	This is SetBufferCount, a member of class TDataLink.
	UpdateData (see page 10343)	Discards requests to post changes to the dataset.

Protected Properties

	Name	Description
	VisualControl (see page 6894)	Indicates whether a visual control uses the datalink.

Public Methods

	Name	Description
	AddMapping (see page 10339)	Adds a field component to the end of the Fields (see page 10345) array.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearMapping (see page 10340)	Empties the Fields (see page 10345) array and frees any associated memory.
	Create (see page 10338)	Creates an instance of THTTPDataLink (see page 10335).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 10338)	Destroys an instance of THTTPDataLink (see page 10335).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6882)	Attempts to drive the data source into an editing state if it is not already in one.
	ExecuteAction (see page 6883)	Invokes an action with the datasource as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	Modified (see page 10342)	Tracks changes to the data which have not been written to the current record in the database.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Reset (see page 10343)	Cancels any pending edits to the dataset that have not been written to the current record in the database.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	UpdateAction (see page 6887)	Updates an action to reflect the current state of the TDataLink (see page 6876).
	UpdateRecord (see page 6888)	Responds to notifications that edits should be posted to the dataset.

Public Properties

	Name	Description
R	Active (see page 6888)	Specifies whether the dataset that TDataLink (see page 6876) manages is active.
	ActiveRecord (see page 6889)	Specifies the index of the current record within the internal record buffer maintained by the dataset for the Owner of the TDataLink (see page 6876) object.
R	BOF (see page 6889)	Indicates whether the first record in a dataset is active.
	BufferCount (see page 6890)	Specifies the number of records cached by the dataset for the Owner of the TDataLink (see page 6876) object.
R	DataSet (see page 6890)	Specifies the TDataSet (see page 6894) object that represents the data that this TDataLink (see page 6876) is tracking.
	DataSource (see page 6891)	Specifies the TDataSource (see page 7070) object that the Owner of the TDataLink (see page 6876) uses to connect to the dataset.
	DataSourceFixed (see page 6891)	Indicates whether the DataSource (see page 6891) property can be set.
R	DefaultFields (see page 10344)	Indicates whether the associated table producer uses dynamically generated field components.
R	Editing (see page 6892)	Indicates whether the DataSource (see page 6891) is in edit mode.
R	Eof (see page 6892)	Indicates whether the last record in a dataset is active.

R	FieldCount (see page 10344)	Specifies the number of entries in the Fields (see page 10345) array.
R	Fields (see page 10345)	Lists all the field components from the dataset that may be shown in the image of the datasource.
	ReadOnly (see page 6893)	Indicates whether the dataset for this TDataLink (see page 6876) can be modified.
R	RecordCount (see page 6893)	Indicates the number of records in the internal record buffer maintained by the dataset.
	SparseMap (see page 10345)	Indicates whether the associated table producer has customized the number or order of the fields that appear in the table.

54.1.6.2 THTTPDataLink.Create Constructor

Creates an instance of THTTPDataLink (see page 10335).

Pascal

```
constructor Create(DSTableProducer: TDSTableProducer);
```

C++

```
__fastcall THTTPDataLink(TDSTableProducer DSTableProducer);
```

Description

Applications should not create instances of THTTPDataLink (see page 10335). TDSTableProducer (see page 10290) instantiates a THTTPDataLink (see page 10335) object from its own constructor.

See Also

TDSTableProducer (see page 10290)

54.1.6.3 THTTPDataLink.Destroy Destructor

Destroys an instance of THTTPDataLink (see page 10335).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THTTPDataLink();
```

Description

Do not call Destroy directly in an application. The TDSTableProducer (see page 10290) object that owns the THTTPDataLink (see page 10335) frees the THTTPDataLink (see page 10335) from its own destructor.

Destroy frees up the memory used by the Fields (see page 10345) array before calling the inherited destructor.

See Also

Fields (see page 10345)

Free (see page 22300)

54.1.6.4 THTTPDataLink Methods

The methods of the THTTPDataLink class are listed here.

Protected Methods

	Name	Description
	ActiveChanged (see page 10339)	Informs the table producer when the Active property changes.
	DataSetChanged (see page 10340)	Informs the associated table producer of changes to the dataset.
	DataSetScrolled (see page 10340)	Ignores scrolling events on the dataset.
	EditingChanged (see page 10341)	Ignores changes to the editing state of the dataset.
	FocusControl (see page 10341)	Discards requests to set focus to the indicated field.
	GetMappedIndex (see page 10342)	Returns the index of the field component in the dataset that corresponds to the index of the field component within the Fields (see page 10345) array.
	LayoutChanged (see page 10342)	Informs TDSTableProducer (see page 10290) of changes in the layout of columns at design time.
	RecordChanged (see page 10343)	Ignores changes to the contents of the current record or field of the data source.
	UpdateData (see page 10343)	Discards requests to post changes to the dataset.

Public Methods

	Name	Description
	AddMapping (see page 10339)	Adds a field component to the end of the Fields (see page 10345) array.
	ClearMapping (see page 10340)	Empties the Fields (see page 10345) array and frees any associated memory.
	Modified (see page 10342)	Tracks changes to the data which have not been written to the current record in the database.
	Reset (see page 10343)	Cancels any pending edits to the dataset that have not been written to the current record in the database.

54.1.6.4.1 THTTPDataLink.ActiveChanged Method

Informs the table producer when the Active property changes.

Pascal

```
procedure ActiveChanged; override;
```

C++

```
virtual __fastcall ActiveChanged();
```

Description

Applications cannot call the ActiveChanged method directly. Changes in the Active property trigger this procedure. ActiveChanged calls the LinkActive method of the TDSTableProducer (see page 10290), which recomputes the layout for the table if the dataset has just become active.

See Also

Active (see page 6888)

LinkActive (see page 10303)

54.1.6.4.2 THTTPDataLink.AddMapping Method

Adds a field component to the end of the Fields (see page 10345) array.

Pascal

```
function AddMapping(const FieldName: string): Boolean;
```

C++

```
__fastcall Boolean AddMapping(const AnsiString FieldName);
```

Description

Applications should not call AddMapping directly, because this would cause the Fields (see page 10345) array to become out

of sync with the THTMLTableColumns (see page 10327) object used by TDSTableProducer (see page 10290). TDSTableProducer (see page 10290) calls AddMapping iteratively to set up the Fields (see page 10345) array so that it corresponds to the columns in the table image.

See Also

ClearMapping (see page 10340)

Fields (see page 10345)

TDSTableProducer (see page 10290)

THTMLTableColumns (see page 10327)

54.1.6.4.3 THTTPDataLink.ClearMapping Method

Empties the Fields (see page 10345) array and frees any associated memory.

Pascal

```
procedure ClearMapping;
```

C++

```
__fastcall ClearMapping();
```

Description

Applications should not call ClearMapping directly, because this would cause the Fields (see page 10345) array to become out of sync with the THTMLTableColumns (see page 10327) object used by TDSTableProducer (see page 10290). TDSTableProducer (see page 10290) calls ClearMapping before rebuilding the Fields (see page 10345) list, every time the column mapping changes.

See Also

AddMapping (see page 10339)

Fields (see page 10345)

THTMLTableColumns (see page 10327)

54.1.6.4.4 THTTPDataLink.DataSetChanged Method

Informs the associated table producer of changes to the dataset.

Pascal

```
procedure DataSetChanged; override;
```

C++

```
virtual __fastcall DataSetChanged();
```

Description

Applications can not call DataSetChanged directly. Changes to the contents of the dataset trigger this method automatically.

See Also

TDSTableProducer (see page 10290)

54.1.6.4.5 THTTPDataLink.DataSetScrolled Method

Ignores scrolling events on the dataset.

Pascal

```
procedure DataSetScrolled(Distance: Integer); override;
```

C++

```
virtual __fastcall DataSetScrolled(int Distance);
```

Description

Applications can not call DataSetScrolled directly. This method overrides the inherited method so that TDSTableProducer (see page 10290) can ignore scrolling events that would occur if the TDSTableProducer (see page 10290) were a visual control.

See Also

DataSetScrolled (see page 6882)

TDataSource (see page 7070)

TDSTableProducer (see page 10290)

54.1.6.4.6 THTTPDataLink.EditingChanged Method

Ignores changes to the editing state of the dataset.

Pascal

```
procedure EditingChanged; override;
```

C++

```
virtual __fastcall EditingChanged();
```

Description

Applications can not call EditingChanged directly. This method ignores changes to the editing state of the dataset, which are irrelevant to TDSTableProducer (see page 10290).

See Also

TDataSet (see page 6894)

TDSTableProducer (see page 10290)

54.1.6.4.7 THTTPDataLink.FocusControl Method

Discards requests to set focus to the indicated field.

Pascal

```
procedure FocusControl(Field: TFieldRef); override;
```

C++

```
virtual __fastcall FocusControl(TFieldRef Field);
```

Description

TDSTableProducer (see page 10290) does not support the concept of focus assigned to an individual field. FocusControl discards requests sent through the data source to assign focus to an individual field.

See Also

TDataSource (see page 7070)

TDSTableProducer (see page 10290)

54.1.6.4.8 THTTPDataLink.GetMappedIndex Method

Returns the index of the field component in the dataset that corresponds to the index of the field component within the Fields (see page 10345) array.

Pascal

```
function GetMappedIndex(ColIndex: Integer): Integer;
```

C++

```
__fastcall int GetMappedIndex(int ColIndex);
```

Description

Applications can not call GetMappedIndex directly. GetMappedIndex takes ColIndex, the index of a field component in the Fields (see page 10345) array, and returns the corresponding index of the field component in the dataset.

See Also

Fields (see page 10345)

Fields (see page 7023)

54.1.6.4.9 THTTPDataLink.LayoutChanged Method

Informs TDSTableProducer (see page 10290) of changes in the layout of columns at design time.

Pascal

```
procedure LayoutChanged; override;
```

C++

```
virtual __fastcall LayoutChanged();
```

Description

Applications can not call the LayoutChanged method directly. LayoutChanged is triggered automatically by changes to the layout of columns at design time. LayoutChanged instructs the TDSTableProducer (see page 10290) object to recompute the layout for the table.

See Also

LayoutChanged (see page 10303)

54.1.6.4.10 THTTPDataLink.Modified Method

Tracks changes to the data which have not been written to the current record in the database.

Pascal

```
procedure Modified;
```

C++

```
__fastcall Modified();
```

Description

Call Modified before changing the value of a field in the current record of the dataset. It is not necessary to call Modified if the changes are written directly to the field components indexed by the Fields (see page 10345) property.

See Also

Fields (see page 10345)

54.1.6.4.11 THTTPDataLink.RecordChanged Method

Ignores changes to the contents of the current record or field of the data source.

Pascal

```
procedure RecordChanged(Field: TField); override;
```

C++

```
virtual __fastcall RecordChanged(TField Field);
```

Description

Applications can not call RecordChanged directly. This method ignores changes to the contents of the current record in the data source, which are irrelevant to the TDSTableProducer (see page 10290).

See Also

DataSource (see page 6891)

TDSTableProducer (see page 10290)

54.1.6.4.12 THTTPDataLink.Reset Method

Cancels any pending edits to the dataset that have not been written to the current record in the database.

Pascal

```
procedure Reset;
```

C++

```
__fastcall Reset();
```

Description

Applications can not call Reset directly. Descendants of TDSTableProducer (see page 10290) can implement a method or user interface that triggers the cancellation of pending edits.

See Also

TDSTableProducer (see page 10290)

54.1.6.4.13 THTTPDataLink.UpdateData Method

Discards requests to post changes to the dataset.

Pascal

```
procedure UpdateData; override;
```

C++

```
virtual __fastcall UpdateData();
```

Description

TDSTableProducer (see page 10290) produces an HTML image of a dataset. It is not a control that can be used to edit the data in the dataset. UpdateData discards requests sent through the data source to post edits to the dataset.

See Also

DataSource (see page 6891)

TDSTableProducer (see page 10290)

54.1.6.5 THTTPDataLink Properties

The properties of the THTTPDataLink class are listed here.

Public Properties

	Name	Description
R	DefaultFields (see page 10344)	Indicates whether the associated table producer uses dynamically generated field components.
R	FieldCount (see page 10344)	Specifies the number of entries in the Fields (see page 10345) array.
R	Fields (see page 10345)	Lists all the field components from the dataset that may be shown in the image of the datasource.
	SparseMap (see page 10345)	Indicates whether the associated table producer has customized the number or order of the fields that appear in the table.

54.1.6.5.1 THTTPDataLink.DefaultFields Property

Indicates whether the associated table producer uses dynamically generated field components.

Pascal

```
property DefaultFields: Boolean;
```

C++

```
__property Boolean DefaultFields;
```

Description

Read DefaultFields to determine whether the image of the table will display dynamically generated field components or persistent field components. If DefaultFields is true, the fields in the image of the table will represent dynamically allocated field components. If DefaultFields is false, the image will be built from persistent field components.

If the image of the datasource is built from persistent field components, it may not show all of the fields that are present in the underlying database table.

See Also

DataSource (see page 6891)

SparseMap (see page 10345)

54.1.6.5.2 THTTPDataLink.FieldCount Property

Specifies the number of entries in the Fields (see page 10345) array.

Pascal

```
property FieldCount: Integer;
```

C++

```
__property int FieldCount;
```

Description

Use FieldCount when iterating over all the field components in the Fields (see page 10345) property. FieldCount is the number of columns shown in the image of the datasource. The Fields (see page 10345) array can contain undefined field components, which correspond to field components that are not in the dataset, such as calculated fields.

See Also

DefaultFields (see page 10344)

Fields (see page 10345)

SparseMap (see page 10345)

54.1.6.5.3 THTTPDataLink.Fields Property

Lists all the field components from the dataset that may be shown in the image of the datasource.

Pascal

```
property Fields [I: Integer]: TField;
```

C++

```
__property TField Fields[int I];
```

Description

Use Fields to access field components by index. Fields[0] is the first field component, Fields[1] is the second field component, and so on. If the SparseMap (see page 10345) property is true, Fields may contain undefined field components. For these entries, Fields returns nil. NULL.

TDSTableProducer (see page 10290) uses Fields to keep track of all field components that are part of its dataset. Whenever necessary, TDSTableProducer (see page 10290) reconciles the field bindings of its columns with the field components in Fields.

See Also

AddMapping (see page 10339)

FieldCount (see page 10344)

SparseMap (see page 10345)

TField (see page 7115)

54.1.6.5.4 THTTPDataLink.SparseMap Property

Indicates whether the associated table producer has customized the number or order of the fields that appear in the table.

Pascal

```
property SparseMap: Boolean;
```

C++

```
__property Boolean SparseMap;
```

Description

Read SparseMap to determine whether Fields (see page 10345) is a simple reflection of the underlying dataset. TDSTableProducer (see page 10290) sets SparseMap when it builds the field map. SparseMap is false if TDSTableProducer (see page 10290) builds a default field map directly from the dataset. SparseMap is true if the number or order of the fields has been changed from the default. When SparseMap is true, the Fields (see page 10345) array may contain undefined fields.

See Also

AddMapping (see page 10339)

Fields (see page 10345)

54.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	HtmlTable (see page 10346)	Generates the HTML image of a dataset, using the properties and events of a table producer object.

54.2.1 DBWeb.HtmlTable Function

Generates the HTML image of a dataset, using the properties and events of a table producer object.

Pascal

```
function HtmlTable(DataSet: TDataSet; DataSetHandler: TDSTableProducer; MaxRows: Integer):
string;
```

C++

```
AnsiString HtmlTable(TDataSet DataSet, TDSTableProducer DataSetHandler, int MaxRows);
```

File

DBWeb

Description

Call HtmlTable to obtain a sequence of HTML commands that describe a dataset in tabular format. The DataSet parameter indicates the dataset whose records are converted into an HTML image. The DataSetHandler parameter is the table producer whose properties describe the attributes of the resulting image and whose event handlers respond to events that occur during the generation of the image. The MaxRows parameter is the maximum number of records from the dataset that will appear in the image.

Calling HtmlTable is equivalent to calling the Content method of the DataSetHandler parameter after setting its DataSet and MaxRows properties to the values given for the parameters of the same names.

See Also

TDataSetTableproducer

TSQLQueryTableProducer

TQueryTableProducer

TDSTableProducer (see page 10290)

TDSTableProducer (see page 10290)

54.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	THTMLCaptionAlignment (see page 10347)	This is record DBWeb.THTMLCaptionAlignment.
	THTMLColumnState (see page 10347)	This is record DBWeb.THTMLColumnState.

54.3.1 DBWeb.THTMLCaptionAlignment Enumeration

Pascal

```
THTMLCaptionAlignment = (
  caDefault,
  caTop,
  caBottom
);
```

C++

```
enum THTMLCaptionAlignment {
  caDefault,
  caTop,
  caBottom
};
```

File

DBWeb

Description

This is record DBWeb.THTMLCaptionAlignment.

54.3.2 DBWeb.THTMLColumnState Enumeration

Pascal

```
THTMLColumnState = (
  csDefault,
  csCustom
);
```

C++

```
enum THTMLColumnState {
  csDefault,
  csCustom
};
```

File

DBWeb

Description

This is record DBWeb.THTMLColumnState.

54.4 Types

The following table lists types in this documentation.

Types

Name	Description
TCreateContentEvent (see page 10349)	This is type DBWeb.TCreateContentEvent.
THTMLDataSetEmpty (see page 10349)	This is type DBWeb.THTMLDataSetEmpty.
THTMLFormatCellEvent (see page 10350)	This is type DBWeb.THTMLFormatCellEvent.
THTMLGetTableCaptionEvent (see page 10350)	This is type DBWeb.THTMLGetTableCaptionEvent.
THTMLTableColumnClass (see page 10350)	This is type DBWeb.THTMLTableColumnClass.

54.4.1 DBWeb.TCreateContentEvent Type

Pascal

```
TCreateContentEvent = procedure (Sender: TObject; var Continue: Boolean) of object;
```

C++

```
(Sender: TObject; var Continue: Boolean) ( TCreateContentEvent)();
```

File

DBWeb

Description

This is type DBWeb.TCreateContentEvent.

54.4.2 DBWeb.THTMLDataSetEmpty Type

Pascal

```
THTMLDataSetEmpty = procedure (Sender: TObject; var Continue: Boolean) of object;
```

C++

```
(Sender: TObject; var Continue: Boolean) ( THTMLDataSetEmpty)();
```

File

DBWeb

Description

This is type DBWeb.THTMLDataSetEmpty.

54.4.3 DBWeb.THTMLFormatCellEvent Type

Pascal

```
THTMLFormatCellEvent = procedure (Sender: TObject; CellRow, CellColumn: Integer; var BgColor:
THTMLBgColor; var Align: THTMLAlign; var VAlign: THTMLVAlign; var CustomAttrs, CellData:
string) of object;
```

C++

```
(Sender: TObject; CellRow, CellColumn: Integer; var BgColor: THTMLBgColor; var Align:
THTMLAlign; var VAlign: THTMLVAlign; var CustomAttrs, CellData: string) (
THTMLFormatCellEvent)();
```

File

DBWeb

Description

This is type DBWeb.THTMLFormatCellEvent.

54.4.4 DBWeb.THTMLGetTableCaptionEvent Type

Pascal

```
THTMLGetTableCaptionEvent = procedure (Sender: TObject; var Caption: string; var Alignment:
THTMLCaptionAlignment) of object;
```

C++

```
(Sender: TObject; var Caption: string; var Alignment: THTMLCaptionAlignment) (
THTMLGetTableCaptionEvent)();
```

File

DBWeb

Description

This is type DBWeb.THTMLGetTableCaptionEvent.

54.4.5 DBWeb.THTMLTableColumnClass Type

Pascal

```
THTMLTableColumnClass = class of THTMLTableColumn;
```

C++

```
THTMLTableColumn THTMLTableColumnClass;
```

File

DBWeb

Description

This is type DBWeb.THTMLTableColumnClass.

55 DBXCommon Namespace

This is namespace DBXCommon.

Topics

Name	Description
Classes (see page 10355)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 10668)	The following table lists structs, records, enums in this documentation.
Types (see page 10669)	The following table lists types in this documentation.
Constants (see page 10675)	The following table lists constants in this documentation.

Classes

	Name	Description
	TDBXAnsiStringValue (see page 10356)	Implementation of writable ANSI string values.
	TDBXBcdValue (see page 10359)	Implementation of binary-compressed decimal values.
	TDBXBooleanValue (see page 10362)	Implementation of writable boolean values.
	TDBXByteArrayValue (see page 10364)	Implementation of writable byte array values.
	TDBXByteReader (see page 10368)	Allows for primitive retrieval of data using byte arrays.
	TDBXByteReaderEx (see page 10373)	This is class DBXCommon.TDBXByteReaderEx.
	TDBXByteStreamReader (see page 10374)	Used internally by driver implementations and TDBXStreamValue (see page 10608).
	TDBXCommand (see page 10378)	Class to execute SQL statements and stored procedures
	TDBXCommandEx (see page 10391)	Extension of TDBXCommand (see page 10378).
	TDBXCommandFactory (see page 10395)	This is class DBXCommon.TDBXCommandFactory.
	TDBXCommandTypes (see page 10396)	Type of command to be executed by a TDBXCommand (see page 10378)
	TDBXConnection (see page 10398)	Connection for the dbExpress driver framework
	TDBXConnectionBuilder (see page 10412)	Utility class used by driver writers when connections are created.
	TDBXConnectionEx (see page 10417)	Temporary extension of TDBXConnection (see page 10398) class.
	TDBXConnectionFactory (see page 10422)	Abstract base class used for creating new instances of TDBXConnection (see page 10398)
	TDBXContext (see page 10432)	Context information for a connection.
	TDBXCursorValue (see page 10438)	Implementation of TDBXWritableValue (see page 10661) for TDBXDataTypes.CursorType.
	TDBXDatabaseMetaData (see page 10440)	Common database specific metadata.
	TDBXDatabaseMetaDataEx (see page 10444)	Temporary extension of TDBXDatabaseMetaData (see page 10440).
	TDBXDataTypes (see page 10447)	Data types supported by the dbExpress framework (TDBX).
	TDBXDataTypesEx (see page 10459)	Data types supported by dbExpress.
	TDBXDateValue (see page 10461)	Implementation of writable date type values.

	TDBXDelegateDriver (see page 10464)	Delegate driver base class.
	TDBXDelegateItem (see page 10468)	Used by driver implementors.
	TDBXDoubleValue (see page 10469)	Implementation of double-precision floating point values.
	TDBXDriver (see page 10472)	Driver class for DBX framework.
	TDBXDriverEx (see page 10477)	This is class DBXCommon.TDBXDriverEx.
	TDBXDriverHelp (see page 10480)	Used by driver implementors to access private or protected API.
	TDBXDriverHelpEx (see page 10482)	This is class DBXCommon.TDBXDriverHelpEx.
	TDBXDriverLoader (see page 10483)	Collection of parameters needed when loading a driver.
	TDBXDriverRegistry (see page 10485)	Registry (see page 18633) for TDBXDriverLoader (see page 10483) keyed by driver name.
	TDBXError (see page 10486)	Exception type for all dbExpress framework (TDBX) errors.
	TDBXErrorCodes (see page 10489)	Error codes for dbExpress framework (TDBX) exceptions
	TDBXIniFileConnectionFactory (see page 10501)	Creates new instances of TDBXConnection (see page 10398) from the dbxdrivers.ini and dbxconnections.ini files
	TDBXIniFileConnectionFactoryEx (see page 10504)	Extension of TDBXIniFileConnectionFactory (see page 10501).
	TDBXInt16Value (see page 10506)	Implementation of 16-byte integer values.
	TDBXInt32Value (see page 10508)	Implementation of writable 32-byte integer values.
	TDBXInt64Value (see page 10511)	Implementation of writable 64-byte integer values.
	TDBXIsolations (see page 10513)	Isolation levels supported by TDBXTransaction (see page 10623).
	TDBXLookAheadStreamReader (see page 10516)	Used internally by driver implementations and TDBXStreamValue (see page 10608).
	TDBXMemoryConnectionFactory (see page 10518)	Implementation of TDBXConnectionFactory (see page 10422) that does not load driver and connection properties.
	TDBXMetaDatabaseColumnNames (see page 10521)	Column names for metadata.
	TDBXMetaDataCommands (see page 10525)	List of available metadata commands to get metadata.
	TDBXMetaDataCommandsEx (see page 10531)	Temporary extension of TDBXMetaDataCommands (see page 10525) class.
	TDBXMetaDataKeyword (see page 10537)	Keyword for metadata.
	TDBXMetaDataProcedureTypes (see page 10538)	Constants passed to TDBXMetaDataCommands.GetProcedures metadata command.
	TDBXMetaDataTableTypes (see page 10540)	Constants for the GetTables metadata command
	TDBXMetaDataTableTypesEx (see page 10542)	Temporary extension of TDBXMetaDataTableTypes (see page 10540) class.
	TDBXNullValue (see page 10543)	This is class DBXCommon.TDBXNullValue.
	TDBXParameter (see page 10545)	Individual parameters for queries and stored commands.
	TDBXParameterDirections (see page 10549)	Parameter types supported by TDBXParameter (see page 10545)
	TDBXParameterEx (see page 10551)	Extension of TDBXParameter (see page 10545).
	TDBXParameterList (see page 10553)	A collection of TDBXParameter (see page 10545) instances that can be used to get and set parameters for a TDBXCommand (see page 10378).
	TDBXProperties (see page 10559)	Manages connection and driver level name/value pair property settings.
	TDBXPropertiesItem (see page 10564)	Contains a set of TDBXProperties (see page 10559).
	TDBXPropertyNames (see page 10566)	Property names for connections and drivers.
	TDBXPropertyNamesEx (see page 10574)	Temporary extension of TDBXPropertyNames (see page 10566) class.
	TDBXReader (see page 10576)	TDBXReader provides a unidirectional reader for a collection of database rows
	TDBXReaderByteReader (see page 10585)	Implementation of TDBXByteReader (see page 10368).
	TDBXRow (see page 10590)	Represents a row in a database.
	TDBXRowEx (see page 10600)	This is class DBXCommon.TDBXRowEx.
	TDBXStateItem (see page 10604)	This is class DBXCommon.TDBXStateItem.
	TDBXStateItemList (see page 10605)	This is class DBXCommon.TDBXStateItemList.
	TDBXStreamReader (see page 10607)	Used internally by driver implementations and TDBXStreamValue (see page 10608).
	TDBXStreamValue (see page 10608)	Implementation of TDBXWritableValue (see page 10661) for TDBXDataTypes.ByteArrayType values.
	TDBXTimeStampValue (see page 10612)	Implementation of writable timestamp values.
	TDBXTimeValue (see page 10614)	Implementation of writable time values.
	TDBXTraceFlags (see page 10617)	Flags to indicate tracing type.
	TDBXTraceInfo (see page 10622)	Trace information for TDBXTraceEvent (see page 10672) handlers.
	TDBXTransaction (see page 10623)	Transaction management class.

	TDBXValue (see page 10626)	Gets and sets values for TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
	TDBXValueList (see page 10634)	Manage a list of TDBXValueType (see page 10636) instances.
	TDBXValueType (see page 10636)	Metadata for TDBXReader (see page 10576) columns and TDBXParameter (see page 10545) parameters.
	TDBXValueTypeEx (see page 10649)	Extension of TDBXValueType (see page 10636).
	TDBXValueTypeFlags (see page 10651)	Flags used by TDBXValueType (see page 10636).
	TDBXValueTypeFlagsEx (see page 10653)	Temporary extension of TDBXValueTypeFlags (see page 10651) class.
	TDBXWideCharsValue (see page 10655)	This is class DBXCommon.TDBXWideCharsValue.
	TDBXWideStringValue (see page 10658)	Implementation of writable wide string values.
	TDBXWritableValue (see page 10661)	Extension of TDBXValue (see page 10626) that adds the ability to set a value.

Constants

Name	Description
DBXDefaultRowSetSize (see page 10675)	This is constant DBXCommon.DBXDefaultRowSetSize.
DBXVersion25 (see page 10676)	This is constant DBXCommon.DBXVersion25.
DBXVersion30 (see page 10676)	This is constant DBXCommon.DBXVersion30.
DBXVersion40 (see page 10676)	This is constant DBXCommon.DBXVersion40.
SAlreadyPrepared (see page 10677)	This is constant DBXCommon.SAlreadyPrepared.
SConnectionFactoryInitFailed (see page 10677)	This is constant DBXCommon.SConnectionFactoryInitFailed.
SDefaultErrorMessage (see page 10677)	This is constant DBXCommon.SDefaultErrorMessage.
SDllLoadError (see page 10678)	This is constant DBXCommon.SDllLoadError.
SDllProcLoadError (see page 10678)	This is constant DBXCommon.SDllProcLoadError.
SDriverLoadError (see page 10678)	This is constant DBXCommon.SDriverLoadError.
SInvalidArgument (see page 10679)	This is constant DBXCommon.SInvalidArgument.
SInvalidDelegationDepth (see page 10679)	This is constant DBXCommon.SInvalidDelegationDepth.
SInvalidOrdinal (see page 10679)	This is constant DBXCommon.SInvalidOrdinal.
SInvalidTransaction (see page 10680)	This is constant DBXCommon.SInvalidTransaction.
SInvalidTypeAccess (see page 10680)	This is constant DBXCommon.SInvalidTypeAccess.
SNotImplemented (see page 10680)	This is constant DBXCommon.SNotImplemented.
SReaderClosed (see page 10681)	This is constant DBXCommon.SReaderClosed.
SReaderNew (see page 10681)	This is constant DBXCommon.SReaderNew.
SReadOnlyParameter (see page 10681)	This is constant DBXCommon.SReadOnlyParameter.
SReadOnlyType (see page 10682)	This is constant DBXCommon.SReadOnlyType.
SRequiredProperty (see page 10682)	This is constant DBXCommon.SRequiredProperty.
SSetFactoryOnce (see page 10682)	This is constant DBXCommon.SSetFactoryOnce.
SUnknownDriver (see page 10683)	This is constant DBXCommon.SUnknownDriver.
TDBXConnectionFile (see page 10683)	This is constant DBXCommon.TDBXConnectionFile.
TDBXDriverFile (see page 10683)	This is constant DBXCommon.TDBXDriverFile.
TDBXRegistryConnectionValue (see page 10684)	This is constant DBXCommon.TDBXRegistryConnectionValue.
TDBXRegistryDriverValue (see page 10684)	This is constant DBXCommon.TDBXRegistryDriverValue.
TDBXRegistryKey (see page 10684)	This is constant DBXCommon.TDBXRegistryKey.

Structs, Records, Enums

	Name	Description
	TDBXDriverDef (see page 10668)	Collection of parameters needed when loading a driver.

Types

Name	Description
TDBXChars (see page 10669)	This is type DBXCommon.TDBXChars.
TDBXCreateCommandEvent (see page 10669)	This is type DBXCommon.TDBXCreateCommandEvent.
TDBXDate (see page 10670)	This is type DBXCommon.TDBXDate.
TDBXDriverLoaderClass (see page 10670)	This is type DBXCommon.TDBXDriverLoaderClass.
TDBXErrorCode (see page 10670)	This is type DBXCommon.TDBXErrorCode.
TDBXErrorEvent (see page 10671)	Event handler for TDBXConnection.OnErrorEvent property.

TDBXIsolation (see page 10671)	This is type DBXCommon.TDBXIsolation.
TDBXParameterDirection (see page 10671)	This is type DBXCommon.TDBXParameterDirection.
TDBXTime (see page 10672)	This is type DBXCommon.TDBXTime.
TDBXTraceEvent (see page 10672)	Tracing information for drivers.
TDBXTraceFlag (see page 10673)	This is type DBXCommon.TDBXTraceFlag.
TDBXType (see page 10673)	Type for TDBXDataTypes (see page 10447).
TDBXValueArray (see page 10673)	This is type DBXCommon.TDBXValueArray.
TWideStringArray (see page 10674)	This is type DBXCommon.TWideStringArray.

55.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TDBXAnsiStringValue (see page 10356)	Implementation of writable ANSI string values.
	TDBXBcdValue (see page 10359)	Implementation of binary-compressed decimal values.
	TDBXBooleanValue (see page 10362)	Implementation of writable boolean values.
	TDBXByteArrayValue (see page 10364)	Implementation of writable byte array values.
	TDBXByteReader (see page 10368)	Allows for primitive retrieval of data using byte arrays.
	TDBXByteReaderEx (see page 10373)	This is class DBXCommon.TDBXByteReaderEx.
	TDBXByteStreamReader (see page 10374)	Used internally by driver implementations and TDBXStreamValue (see page 10608).
	TDBXCommand (see page 10378)	Class to execute SQL statements and stored procedures
	TDBXCommandEx (see page 10391)	Extension of TDBXCommand (see page 10378).
	TDBXCommandFactory (see page 10395)	This is class DBXCommon.TDBXCommandFactory.
	TDBXCommandTypes (see page 10396)	Type of command to be executed by a TDBXCommand (see page 10378)
	TDBXConnection (see page 10398)	Connection for the dbExpress driver framework
	TDBXConnectionBuilder (see page 10412)	Utility class used by driver writers when connections are created.
	TDBXConnectionEx (see page 10417)	Temporary extension of TDBXConnection (see page 10398) class.
	TDBXConnectionFactory (see page 10422)	Abstract base class used for creating new instances of TDBXConnection (see page 10398)
	TDBXContext (see page 10432)	Context information for a connection.
	TDBXCursorValue (see page 10438)	Implementation of TDBXWritableValue (see page 10661) for TDBXDataTypes.CursorType.
	TDBXDatabaseMetaData (see page 10440)	Common database specific metadata.
	TDBXDatabaseMetaDataEx (see page 10444)	Temporary extension of TDBXDatabaseMetaData (see page 10440).
	TDBXDataTypes (see page 10447)	Data types supported by the dbExpress framework (TDBX).
	TDBXDataTypesEx (see page 10459)	Data types supported by dbExpress.
	TDBXDateValue (see page 10461)	Implementation of writable date type values.
	TDBXDelegateDriver (see page 10464)	Delegate driver base class.
	TDBXDelegateItem (see page 10468)	Used by driver implementors.
	TDBXDoubleValue (see page 10469)	Implementation of double-precision floating point values.
	TDBXDriver (see page 10472)	Driver class for DBX framework.
	TDBXDriverEx (see page 10477)	This is class DBXCommon.TDBXDriverEx.
	TDBXDriverHelp (see page 10480)	Used by driver implementors to access private or protected API.
	TDBXDriverHelpEx (see page 10482)	This is class DBXCommon.TDBXDriverHelpEx.
	TDBXDriverLoader (see page 10483)	Collection of parameters needed when loading a driver.
	TDBXDriverRegistry (see page 10485)	Registry (see page 18633) for TDBXDriverLoader (see page 10483) keyed by driver name.
	TDBXError (see page 10486)	Exception type for all dbExpress framework (TDBX) errors.
	TDBXErrorCodes (see page 10489)	Error codes for dbExpress framework (TDBX) exceptions
	TDBXIniFileConnectionFactory (see page 10501)	Creates new instances of TDBXConnection (see page 10398) from the dbxdrivers.ini and dbxconnections.ini files
	TDBXIniFileConnectionFactoryEx (see page 10504)	Extension of TDBXIniFileConnectionFactory (see page 10501).
	TDBXInt16Value (see page 10506)	Implementation of 16-byte integer values.
	TDBXInt32Value (see page 10508)	Implementation of writable 32-byte integer values.
	TDBXInt64Value (see page 10511)	Implementation of writable 64-byte integer values.
	TDBXIsolations (see page 10513)	Isolation levels supported by TDBXTransaction (see page 10623).
	TDBXLookAheadStreamReader (see page 10516)	Used internally by driver implementations and TDBXStreamValue (see page 10608).

	TDBXMemoryConnectionFactory (see page 10518)	Implementation of TDBXConnectionFactory (see page 10422) that does not load driver and connection properties.
	TDBXMetaDatabaseColumnNames (see page 10521)	Column names for metadata.
	TDBXMetaDataCommands (see page 10525)	List of available metadata commands to get metadata.
	TDBXMetaDataCommandsEx (see page 10531)	Temporary extension of TDBXMetaDataCommands (see page 10525) class.
	TDBXMetaDataKeyword (see page 10537)	Keyword for metadata.
	TDBXMetaDataProcedureTypes (see page 10538)	Constants passed to TDBXMetaDataCommands.GetProcedures metadata command.
	TDBXMetaDataTableTypes (see page 10540)	Constants for the GetTables metadata command
	TDBXMetaDataTableTypesEx (see page 10542)	Temporary extension of TDBXMetaDataTableTypes (see page 10540) class.
	TDBXNullValue (see page 10543)	This is class DBXCommon.TDBXNullValue.
	TDBXParameter (see page 10545)	Individual parameters for queries and stored commands.
	TDBXParameterDirections (see page 10549)	Parameter types supported by TDBXParameter (see page 10545)
	TDBXParameterEx (see page 10551)	Extension of TDBXParameter (see page 10545).
	TDBXParameterList (see page 10553)	A collection of TDBXParameter (see page 10545) instances that can be used to get and set parameters for a TDBXCommand (see page 10378).
	TDBXProperties (see page 10559)	Manages connection and driver level name/value pair property settings.
	TDBXPropertiesItem (see page 10564)	Contains a set of TDBXProperties (see page 10559).
	TDBXPropertyNames (see page 10566)	Property names for connections and drivers.
	TDBXPropertyNamesEx (see page 10574)	Temporary extension of TDBXPropertyNames (see page 10566) class.
	TDBXReader (see page 10576)	TDBXReader provides a unidirectional reader for a collection of database rows
	TDBXReaderByteReader (see page 10585)	Implementation of TDBXByteReader (see page 10368).
	TDBXRow (see page 10590)	Represents a row in a database.
	TDBXRowEx (see page 10600)	This is class DBXCommon.TDBXRowEx.
	TDBXStateItem (see page 10604)	This is class DBXCommon.TDBXStateItem.
	TDBXStateItemList (see page 10605)	This is class DBXCommon.TDBXStateItemList.
	TDBXStreamReader (see page 10607)	Used internally by driver implementations and TDBXStreamValue (see page 10608).
	TDBXStreamValue (see page 10608)	Implementation of TDBXWritableValue (see page 10661) for TDBXDataTypes.ByteArrayType values.
	TDBXTimeStampValue (see page 10612)	Implementation of writable timestamp values.
	TDBXTimeValue (see page 10614)	Implementation of writable time values.
	TDBXTraceFlags (see page 10617)	Flags to indicate tracing type.
	TDBXTransaction (see page 10623)	Transaction management class.
	TDBXValue (see page 10626)	Gets and sets values for TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
	TDBXValueList (see page 10634)	Manage a list of TDBXValueType (see page 10636) instances.
	TDBXValueType (see page 10636)	Metadata for TDBXReader (see page 10576) columns and TDBXParameter (see page 10545) parameters.
	TDBXValueTypeEx (see page 10649)	Extension of TDBXValueType (see page 10636).
	TDBXValueTypeFlags (see page 10651)	Flags used by TDBXValueType (see page 10636).
	TDBXValueTypeFlagsEx (see page 10653)	Temporary extension of TDBXValueTypeFlags (see page 10651) class.
	TDBXWideCharsValue (see page 10655)	This is class DBXCommon.TDBXWideCharsValue.
	TDBXWideStringValue (see page 10658)	Implementation of writable wide string values.
	TDBXWritableValue (see page 10661)	Extension of TDBXValue (see page 10626) that adds the ability to set a value.

Records

	Name	Description
	TDBXTraceInfo (see page 10622)	Trace information for TDBXTraceEvent (see page 10672) handlers.

55.1.1 TDBXAnsiStringValue Class

Implementation of writable ANSI string values.

Class Hierarchy

Pascal

```
TDBXAnsiStringValue = class(TDBXWritableValue);
```

C++

```
class TDBXAnsiStringValue : public TDBXWritableValue;
```

File

DBXCommon

Description

Implementation of TDBXWritableValue (see page 10661) for TDBXDataTypes.AnsiStringType values.

55.1.1.1 TDBXAnsiStringValue Members

The following tables list the members exposed by TDBXAnsiStringValue.

Protected Methods

	Name	Description
	Create (see page 10358)	Creates an instance of TDBXAnsiStringValue (see page 10356)
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.
V	SetPendingValue (see page 10666)	This is SetPendingValue, a member of class TDBXWritableValue.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Methods

	Name	Description
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
V	Destroy (see page 10358)	Destroys a TDBXAnsiStringValue (see page 10356) instance.
V	GetAnsiString (see page 10359)	Gets the string value for ANSI strings.
V	GetBcd (see page 10629)	Gets the BCD value for binary-compressed decimal data types.
V	GetBoolean (see page 10629)	Gets the boolean value for boolean data types.
V	GetBytes (see page 10629)	Gets the byte array value for byte array and blob types.
V	GetDate (see page 10630)	Gets the date value for date data types.
V	GetDouble (see page 10630)	Gets the double floating point value for double data types.
V	GetInt16 (see page 10631)	Gets the smallInt value for 16-byte integer data types.
V	GetInt32 (see page 10631)	Gets the 32-byte integer value for 32-byte integer data types.
V	GetInt64 (see page 10631)	Gets the 64-byte integer value for 64-byte integer data types.
V	GetTime (see page 10631)	Gets the time value for time data types.
V	GetTimeStamp (see page 10632)	Gets the timestamp value for timestamp data types.
V	GetValueSize (see page 10632)	Gets the size in bytes for values.
V	GetWideString (see page 10632)	Gets the wide string value for wide string data types.
V	IsNull (see page 10359)	Returns true if the value is **null**.
V	SetAnsiString (see page 10359)	Sets the string value for ANSI strings.
V	SetBcd (see page 10664)	Sets the BCD value for binary-compressed decimal data types.
V	SetBoolean (see page 10664)	Sets the boolean value for boolean data types..
V	SetDate (see page 10664)	Sets the date value for date data types..
V	SetDouble (see page 10664)	Sets the double floating point value for double data types.

	SetDynamicBytes (see page 10665)	Sets the dynamic array value for byte array and blob types.
	SetInt16 (see page 10665)	Sets the the smallInt value for 16-byte integer data types.
	SetInt32 (see page 10665)	Sets the 32-byte integer value for 32-byte integer data types.
	SetInt64 (see page 10665)	Sets the 64-byte integer value for 64-byte integer data types.
	SetNull (see page 10666)	Sets a writable value of any type to null.
	SetStaticBytes (see page 10666)	Sets the open array value for byte array and blob types
	SetTime (see page 10666)	Sets the time value for time data types.
	SetTimeStamp (see page 10667)	Sets the timestamp value for timestamp data types.
	SetWideString (see page 10667)	Sets the wide string value for wide string data types.

Public Properties

	Name	Description
	ValueType (see page 10633)	Gets the value type for this value.

55.1.1.2 TDBXAnsiStringValue.Create Constructor

Creates an instance of TDBXAnsiStringValue (see page 10356)

Pascal

```
constructor Create(ValueType: TDBXValueType);
```

C++

```
__fastcall TDBXAnsiStringValue(TDBXValueType ValueType);
```

Description

This constructor creates and initializes an instance of TDBXAnsiStringValue (see page 10356).

See Also

Destroy (see page 10358)

55.1.1.3 TDBXAnsiStringValue.Destroy Destructor

Destroys a TDBXAnsiStringValue (see page 10356) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXAnsiStringValue();
```

Description

This destructor destroys an instance of TDBXAnsiStringValue (see page 10356).

See Also

Create (see page 10358)

55.1.1.4 TDBXAnsiStringValue Methods

The methods of the TDBXAnsiStringValue class are listed here.

Public Methods

	Name	Description
	GetAnsiString (see page 10359)	Gets the string value for ANSI strings.
	IsNull (see page 10359)	Returns true if the value is **null**.
	SetAnsiString (see page 10359)	Sets the string value for ANSI strings.

55.1.1.4.1 TDBXAnsiStringValue.GetAnsiString Method

Gets the string value for ANSI strings.

Pascal

```
function GetAnsiString: String; override;
```

C++

```
virtual __fastcall AnsiString GetAnsiString();
```

Description

Gets the string value for TDBXValue (see page 10626) instances with a data type of AnsiStringType. Throws a TDBXError (see page 10486) exception if IsNull (see page 10359) is true.

55.1.1.4.2 TDBXAnsiStringValue.IsNull Method

Returns true if the value is **null**.

Pascal

```
function IsNull: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsNull();
```

Description

Returns true if the value is **null**. This method should be called before calling any of the Get methods. If the value is null, the Get methods throw a TDBXError (see page 10486) exception if they are called.

55.1.1.4.3 TDBXAnsiStringValue.SetAnsiString Method

Sets the string value for ANSI strings.

Pascal

```
procedure SetAnsiString(const Value: String); override;
```

C++

```
virtual __fastcall SetAnsiString(const AnsiString Value);
```

Description

Sets the string value for TDBXValue (see page 10626) instances with a data type of AnsiStringType.

55.1.2 TDBXBcdValue Class

Implementation of binary-compressed decimal values.

Class Hierarchy

Pascal

```
TDBXBcdValue = class(TDBXWritableValue);
```

C++

```
class TDBXBcdValue : public TDBXWritableValue;
```

File

DBXCommon

Description

Implementation of TDBXWritableValue (see page 10661) for TDBXDataTypes.BcdType values.

55.1.2.1 TDBXBcdValue Members

The following tables list the members exposed by TDBXBcdValue.

Protected Methods

	Name	Description
	Create (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
	Create (see page 10662)	Creates an instance of TDBXWritableValue (see page 10661)
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.
V	SetPendingValue (see page 10666)	This is SetPendingValue, a member of class TDBXWritableValue.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Methods

	Name	Description
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
V	Destroy (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
V	Destroy (see page 10663)	Destroys a TDBXWritableValue (see page 10661) instance.
V	GetAnsiString (see page 10629)	Gets the string value for ANSI strings.
V	GetBcd (see page 10361)	Gets the BCD value for binary-compressed decimal data types.
V	GetBoolean (see page 10629)	Gets the boolean value for boolean data types.
V	GetBytes (see page 10629)	Gets the byte array value for byte array and blob types.
V	GetDate (see page 10630)	Gets the date value for date data types.
V	GetDouble (see page 10630)	Gets the double floating point value for double data types.
V	GetInt16 (see page 10631)	Gets the smallInt value for 16-byte integer data types.
V	GetInt32 (see page 10631)	Gets the 32-byte integer value for 32-byte integer data types.
V	GetInt64 (see page 10631)	Gets the 64-byte integer value for 64-byte integer data types.
V	GetTime (see page 10631)	Gets the time value for time data types.
V	GetTimeStamp (see page 10632)	Gets the timestamp value for timestamp data types.
V	GetValueSize (see page 10632)	Gets the size in bytes for values.
V	GetWideString (see page 10632)	Gets the wide string value for wide string data types.
V	IsNull (see page 10361)	Returns true if the value is **null**.
V	SetAnsiString (see page 10663)	Sets the string value for ANSI strings.

	SetBcd (see page 10362)	Sets the BCD value for binary-compressed decimal data types.
	SetBoolean (see page 10664)	Sets the boolean value for boolean data types..
	SetDate (see page 10664)	Sets the date value for date data types..
	SetDouble (see page 10664)	Sets the double floating point value for double data types.
	SetDynamicBytes (see page 10665)	Sets the dynamic array value for byte array and blob types.
	SetInt16 (see page 10665)	Sets the the smallInt value for 16-byte integer data types.
	SetInt32 (see page 10665)	Sets the 32-byte integer value for 32-byte integer data types.
	SetInt64 (see page 10665)	Sets the 64-byte integer value for 64-byte integer data types.
	SetNull (see page 10666)	Sets a writable value of any type to null.
	SetStaticBytes (see page 10666)	Sets the open array value for byte array and blob types
	SetTime (see page 10666)	Sets the time value for time data types.
	SetTimeStamp (see page 10667)	Sets the timestamp value for timestamp data types.
	SetWideString (see page 10667)	Sets the wide string value for wide string data types.

Public Properties

	Name	Description
	ValueType (see page 10633)	Gets the value type for this value.

55.1.2.2 TDBXBcdValue Methods

The methods of the TDBXBcdValue class are listed here.

Public Methods

	Name	Description
	GetBcd (see page 10361)	Gets the BCD value for binary-compressed decimal data types.
	IsNull (see page 10361)	Returns true if the value is **null**.
	SetBcd (see page 10362)	Sets the BCD value for binary-compressed decimal data types.

55.1.2.2.1 TDBXBcdValue.GetBcd Method

Gets the BCD value for binary-compressed decimal data types.

Pascal

```
function GetBcd: TBcd; override;
```

C++

```
virtual __fastcall TBcd GetBcd();
```

Description

Gets the TBcd value for TDBXValue (see page 10626) instances with a with a data type of BcdType. Throws a TDBXError (see page 10486) exception if IsNull (see page 10361) is true.

55.1.2.2.2 TDBXBcdValue.IsNull Method

Returns true if the value is **null**.

Pascal

```
function IsNull: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsNull();
```

Description

Returns true if the value is **null**. This method should be called before calling any of the Get methods. If the value is null, the Get methods will throw TDBXError (see page 10486) exception if they are called.

55.1.2.2.3 TDBXBcdValue.SetBcd Method

Sets the BCD value for binary-compressed decimal data types.

Pascal

```
procedure SetBcd(const Value: TBcd); override;
```

C++

```
virtual __fastcall SetBcd(const TBcd Value);
```

Description

Sets the TBcd value for TDBXValue (see page 10626) instances with a with a data type of BcdType.

55.1.3 TDBXBooleanValue Class

Implementation of writable boolean values.

Class Hierarchy

Pascal

```
TDBXBooleanValue = class(TDBXWritableValue);
```

C++

```
class TDBXBooleanValue : public TDBXWritableValue;
```

File

DBXCommon

Description

Implementation of TDBXWritableValue (see page 10661) for TDBXDataTypes.BooleanType values.

55.1.3.1 TDBXBooleanValue Members

The following tables list the members exposed by TDBXBooleanValue.

Protected Methods

	Name	Description
	Create (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
	Create (see page 10662)	Creates an instance of TDBXWritableValue (see page 10661)
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.
V	SetPendingValue (see page 10666)	This is SetPendingValue, a member of class TDBXWritableValue.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Methods

	Name	Description
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
V	Destroy (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
V	Destroy (see page 10663)	Destroys a TDBXWritableValue (see page 10661) instance.
V	GetAnsiString (see page 10629)	Gets the string value for ANSI strings.
V	GetBcd (see page 10629)	Gets the BCD value for binary-compressed decimal data types.
V	GetBoolean (see page 10364)	Gets the boolean value for boolean data types.
V	GetBytes (see page 10629)	Gets the byte array value for byte array and blob types.
V	GetDate (see page 10630)	Gets the date value for date data types.
V	GetDouble (see page 10630)	Gets the double floating point value for double data types.
V	GetInt16 (see page 10631)	Gets the smallInt value for 16-byte integer data types.
V	GetInt32 (see page 10631)	Gets the 32-byte integer value for 32-byte integer data types.
V	GetInt64 (see page 10631)	Gets the 64-byte integer value for 64-byte integer data types.
V	GetTime (see page 10631)	Gets the time value for time data types.
V	GetTimeStamp (see page 10632)	Gets the timestamp value for timestamp data types.
V	GetValueSize (see page 10632)	Gets the size in bytes for values.
V	GetWideString (see page 10632)	Gets the wide string value for wide string data types.
V	IsNull (see page 10364)	Returns true if the value is **null**.
V	SetAnsiString (see page 10663)	Sets the string value for ANSI strings.
V	SetBcd (see page 10664)	Sets the BCD value for binary-compressed decimal data types.
V	SetBoolean (see page 10364)	Sets the boolean value for boolean data types..
V	SetDate (see page 10664)	Sets the date value for date data types..
V	SetDouble (see page 10664)	Sets the double floating point value for double data types.
V	SetDynamicBytes (see page 10665)	Sets the dynamic array value for byte array and blob types.
V	SetInt16 (see page 10665)	Sets the the smallInt value for 16-byte integer data types.
V	SetInt32 (see page 10665)	Sets the 32-byte integer value for 32-byte integer data types.
V	SetInt64 (see page 10665)	Sets the 64-byte integer value for 64-byte integer data types.
V	SetNull (see page 10666)	Sets a writable value of any type to null.
V	SetStaticBytes (see page 10666)	Sets the open array value for byte array and blob types
V	SetTime (see page 10666)	Sets the time value for time data types.
V	SetTimeStamp (see page 10667)	Sets the timestamp value for timestamp data types.
V	SetWideString (see page 10667)	Sets the wide string value for wide string data types.

Public Properties

	Name	Description
R	ValueType (see page 10633)	Gets the value type for this value.

55.1.3.2 TDBXBooleanValue Methods

The methods of the TDBXBooleanValue class are listed here.

Public Methods

	Name	Description
V	GetBoolean (see page 10364)	Gets the boolean value for boolean data types.
V	IsNull (see page 10364)	Returns true if the value is **null**.
V	SetBoolean (see page 10364)	Sets the boolean value for boolean data types..

55.1.3.2.1 TDBXBooleanValue.GetBoolean Method

Gets the boolean value for boolean data types.

Pascal

```
function GetBoolean: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetBoolean();
```

Description

Gets the boolean value for TDBXValue (see page 10626) instances with a data types of BooleanType. Throws a TDBXError (see page 10486) exception if IsNull (see page 10364) is true.

55.1.3.2.2 TDBXBooleanValue.IsNull Method

Returns true if the value is **null**.

Pascal

```
function IsNull: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsNull();
```

Description

Returns true if the value is **null**. This method should be called before calling any of the Get methods. If the value is null, the Get methods will throw TDBXError (see page 10486) exception if they are called.

55.1.3.2.3 TDBXBooleanValue.SetBoolean Method

Sets the boolean value for boolean data types..

Pascal

```
procedure SetBoolean(Value: Boolean); override;
```

C++

```
virtual __fastcall SetBoolean(Boolean Value);
```

Description

Sets the boolean value for TDBXValue (see page 10626) instances with a data types of BooleanType..

55.1.4 TDBXByteArrayValue Class

Implementation of writable byte array values.

Class Hierarchy

Pascal

```
TDBXByteArrayValue = class(TDBXWritableValue);
```

C++

```
class TDBXByteArrayValue : public TDBXWritableValue;
```

File

DBXCommon

Description

Implementation of TDBXWritableValue (see page 10661) for TDBXDataTypes.ByteArrayType values.

55.1.4.1 TDBXByteArrayValue Members

The following tables list the members exposed by TDBXByteArrayValue.

Protected Methods

	Name	Description
	Create (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
	Create (see page 10662)	Creates an instance of TDBXWritableValue (see page 10661)
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.
V	SetPendingValue (see page 10666)	This is SetPendingValue, a member of class TDBXWritableValue.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Methods

	Name	Description
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
V	Destroy (see page 10366)	Destroy a TDBXByteArrayValue (see page 10364) instance
V	GetAnsiString (see page 10629)	Gets the string value for ANSI strings.
V	GetBcd (see page 10629)	Gets the BCD value for binary-compressed decimal data types.
V	GetBoolean (see page 10629)	Gets the boolean value for boolean data types.
V	GetBytes (see page 10366)	Gets the byte array value for byte array and blob types.
V	GetDate (see page 10630)	Gets the date value for date data types.
V	GetDouble (see page 10630)	Gets the double floating point value for double data types.
V	GetInt16 (see page 10631)	Gets the smallInt value for 16-byte integer data types.
V	GetInt32 (see page 10631)	Gets the 32-byte integer value for 32-byte integer data types.
V	GetInt64 (see page 10631)	Gets the 64-byte integer value for 64-byte integer data types.
V	GetTime (see page 10631)	Gets the time value for time data types.
V	GetTimeStamp (see page 10632)	Gets the timestamp value for timestamp data types.
V	GetValueSize (see page 10367)	Gets the size in bytes for values.
V	GetWideString (see page 10632)	Gets the wide string value for wide string data types.
V	IsNull (see page 10367)	Returns true if the value is **null**.
V	SetAnsiString (see page 10663)	Sets the string value for ANSI strings.
V	SetBcd (see page 10664)	Sets the BCD value for binary-compressed decimal data types.
V	SetBoolean (see page 10664)	Sets the boolean value for boolean data types..
V	SetDate (see page 10664)	Sets the date value for date data types..
V	SetDouble (see page 10664)	Sets the double floating point value for double data types.
V	SetDynamicBytes (see page 10367)	Sets the dynamic array value for byte array and blob types.
V	SetInt16 (see page 10665)	Sets the the smallInt value for 16-byte integer data types.
V	SetInt32 (see page 10665)	Sets the 32-byte integer value for 32-byte integer data types.
V	SetInt64 (see page 10665)	Sets the 64-byte integer value for 64-byte integer data types.

	SetNull (see page 10666)	Sets a writable value of any type to null.
	SetStaticBytes (see page 10367)	Sets the open array value for byte array and blob types
	SetTime (see page 10666)	Sets the time value for time data types.
	SetTimeStamp (see page 10667)	Sets the timestamp value for timestamp data types.
	SetWideString (see page 10667)	Sets the wide string value for wide string data types.

Public Properties

	Name	Description
	ValueType (see page 10633)	Gets the value type for this value.

55.1.4.2 TDBXByteArrayValue.Destroy Destructor

Destroy a TDBXByteArrayValue (see page 10364) instance

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXByteArrayValue();
```

Description

This method destroys a TDBXByteArrayValue (see page 10364) instance.

55.1.4.3 TDBXByteArrayValue Methods

The methods of the TDBXByteArrayValue class are listed here.

Public Methods

	Name	Description
	GetBytes (see page 10366)	Gets the byte array value for byte array and blob types.
	GetValueSize (see page 10367)	Gets the size in bytes for values.
	IsNull (see page 10367)	Returns true if the value is **null**.
	SetDynamicBytes (see page 10367)	Sets the dynamic array value for byte array and blob types.
	SetStaticBytes (see page 10367)	Sets the open array value for byte array and blob types

55.1.4.3.1 TDBXByteArrayValue.GetBytes Method

Gets the byte array value for byte array and blob types.

Pascal

```
function GetBytes(Offset: Int64; const Buffer: TBytes; BufferOffset: Int64; Length: Int64):
Int64; override;
```

C++

```
virtual __fastcall Int64 GetBytes(Int64 Offset, const TBytes Buffer, Int64 BufferOffset, Int64
Length);
```

Description

Gets the byte array value for TDBXValue (see page 10626) instances with a data type of BytesType and .BlobType. Throws a TDBXError (see page 10486) exception if IsNull (see page 10367) is true.

55.1.4.3.2 TDBXByteArrayValue.GetValueSize Method

Gets the size in bytes for values.

Pascal

```
function GetValueSize: Int64; override;
```

C++

```
virtual __fastcall Int64 GetValueSize();
```

Description

Gets the size in bytes for a TDBXValue (see page 10626). For string data types the number of characters is returned. Throws a TDBXError (see page 10486) exception if IsNull (see page 10367) is true.

55.1.4.3.3 TDBXByteArrayValue.IsNull Method

Returns true if the value is **null**.

Pascal

```
function IsNull: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsNull();
```

Description

Returns true if the value is **null**. This method should be called before calling any of the Get methods. If the value is null, the Get methods will throw TDBXError (see page 10486) exception if they are called.

55.1.4.3.4 TDBXByteArrayValue.SetDynamicBytes Method

Sets the dynamic array value for byte array and blob types.

Pascal

```
procedure SetDynamicBytes(Offset: Int64; const Buffer: TBytes; BufferOffset: Int64; Length:
Int64); override;
```

C++

```
virtual __fastcall SetDynamicBytes(Int64 Offset, const TBytes Buffer, Int64 BufferOffset,
Int64 Length);
```

Description

Sets the dynamic array value for TDBXValue (see page 10626) instances with a data type of BytesType and .BlobType.

55.1.4.3.5 TDBXByteArrayValue.SetStaticBytes Method

Sets the open array value for byte array and blob types

Pascal

```
procedure SetStaticBytes(Offset: Int64; const Buffer: array of Byte; BufferOffset: Int64;
Length: Int64); override;
```

C++

```
virtual __fastcall SetStaticBytes(Int64 Offset, const array of Byte Buffer, Int64
BufferOffset, Int64 Length);
```

Description

Sets the open array value for TDBXValue (see page 10626) instances with a data type of BytesType and .BlobType.

55.1.5 TDBXByteReader Class

Allows for primitive retrieval of data using byte arrays.

Class Hierarchy

DBXCommon.TDBXByteReader

Pascal

```
TDBXByteReader = class;
```

C++

```
class TDBXByteReader;
```

File

DBXCommon

Description

This class is used by dbExpress related TDataSet components. It allows for primitive retrieval of data using byte arrays. It is not for general usage.

See Also

TDataSet (see page 6894)

55.1.5.1 TDBXByteReader Members

The following tables list the members exposed by TDBXByteReader.

Protected Methods

	Name	Description
	Create (see page 10369)	Creates an instance of TDBXByteReader (see page 10368)

Public Methods

	Name	Description
	GetAnsiString (see page 10370)	This is GetAnsiString, a member of class TDBXByteReader.
	GetBcd (see page 10370)	This is GetBcd, a member of class TDBXByteReader.
	GetByteLength (see page 10370)	This is GetByteLength, a member of class TDBXByteReader.
	GetBytes (see page 10370)	This is GetBytes, a member of class TDBXByteReader.
	GetDate (see page 10371)	This is GetDate, a member of class TDBXByteReader.
	GetDouble (see page 10371)	This is GetDouble, a member of class TDBXByteReader.
	GetInt16 (see page 10371)	This is GetInt16, a member of class TDBXByteReader.
	GetInt32 (see page 10371)	This is GetInt32, a member of class TDBXByteReader.
	GetInt64 (see page 10372)	This is GetInt64, a member of class TDBXByteReader.
	GetTime (see page 10372)	This is GetTime, a member of class TDBXByteReader.
	GetTimeStamp (see page 10372)	This is GetTimeStamp, a member of class TDBXByteReader.
	GetWideString (see page 10372)	This is GetWideString, a member of class TDBXByteReader.

Strict protected Fields

	Name	Description
	FDBXContext (see page 10369)	This is FDBXContext, a member of class TDBXByteReader.

55.1.5.2 TDBXByteReader.Create Constructor

Creates an instance of TDBXByteReader (see page 10368)

Pascal

```
constructor Create(DBXContext: TDBXContext);
```

C++

```
__fastcall TDBXByteReader(TDBXContext DBXContext);
```

Description

This constructor creates and initializes an instance of TDBXByteReader (see page 10368).

55.1.5.3 TDBXByteReader Fields

The fields of the TDBXByteReader class are listed here.

Strict protected Fields

	Name	Description
	FDBXContext (see page 10369)	This is FDBXContext, a member of class TDBXByteReader.

55.1.5.3.1 TDBXByteReader.FDBXContext Field

Pascal

```
FDBXContext: TDBXContext;
```

C++

```
TDBXContext FDBXContext;
```

Description

This is FDBXContext, a member of class TDBXByteReader.

55.1.5.4 TDBXByteReader Methods

The methods of the TDBXByteReader class are listed here.

Public Methods

	Name	Description
	GetAnsiString (see page 10370)	This is GetAnsiString, a member of class TDBXByteReader.
	GetBcd (see page 10370)	This is GetBcd, a member of class TDBXByteReader.
	GetByteLength (see page 10370)	This is GetByteLength, a member of class TDBXByteReader.
	GetBytes (see page 10370)	This is GetBytes, a member of class TDBXByteReader.
	GetDate (see page 10371)	This is GetDate, a member of class TDBXByteReader.
	GetDouble (see page 10371)	This is GetDouble, a member of class TDBXByteReader.
	GetInt16 (see page 10371)	This is GetInt16, a member of class TDBXByteReader.
	GetInt32 (see page 10371)	This is GetInt32, a member of class TDBXByteReader.

	GetInt64 (see page 10372)	This is GetInt64, a member of class TDBXByteReader.
	GetTime (see page 10372)	This is GetTime, a member of class TDBXByteReader.
	GetTimeStamp (see page 10372)	This is GetTimeStamp, a member of class TDBXByteReader.
	GetWideString (see page 10372)	This is GetWideString, a member of class TDBXByteReader.

55.1.5.4.1 TDBXByteReader.GetAnsiString Method

Pascal

```
procedure GetAnsiString(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull:
LongBool); virtual; abstract;
```

C++

```
virtual __fastcall GetAnsiString(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool
IsNull) = 0;
```

Description

This is GetAnsiString, a member of class TDBXByteReader.

55.1.5.4.2 TDBXByteReader.GetBcd Method

Pascal

```
procedure GetBcd(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull: LongBool);
virtual; abstract;
```

C++

```
virtual __fastcall GetBcd(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool IsNull)
= 0;
```

Description

This is GetBcd, a member of class TDBXByteReader.

55.1.5.4.3 TDBXByteReader.GetByteLength Method

Pascal

```
procedure GetByteLength(Ordinal: TInt32; var Length: Int64; var IsNull: LongBool); virtual;
abstract;
```

C++

```
virtual __fastcall GetByteLength(TInt32 Ordinal, Int64 Length, LongBool IsNull) = 0;
```

Description

This is GetByteLength, a member of class TDBXByteReader.

55.1.5.4.4 TDBXByteReader.GetBytes Method

Pascal

```
function GetBytes(Ordinal: TInt32; Offset: Int64; const Value: TBytes; ValueOffset: Int64;
Length: Int64; var IsNull: LongBool): Int64; virtual; abstract;
```

C++

```
virtual __fastcall Int64 GetBytes(TInt32 Ordinal, Int64 Offset, const TBytes Value, Int64
ValueOffset, Int64 Length, LongBool IsNull) = 0;
```

Description

This is GetBytes, a member of class TDBXByteReader.

55.1.5.4.5 TDBXByteReader.GetDate Method

Pascal

```
procedure GetDate(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull: LongBool);
virtual; abstract;
```

C++

```
virtual __fastcall GetDate(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool IsNull)
= 0;
```

Description

This is GetDate, a member of class TDBXByteReader.

55.1.5.4.6 TDBXByteReader.GetDouble Method

Pascal

```
procedure GetDouble(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull:
LongBool); virtual; abstract;
```

C++

```
virtual __fastcall GetDouble(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool
IsNull) = 0;
```

Description

This is GetDouble, a member of class TDBXByteReader.

55.1.5.4.7 TDBXByteReader.GetInt16 Method

Pascal

```
procedure GetInt16(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull:
LongBool); virtual; abstract;
```

C++

```
virtual __fastcall GetInt16(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool
IsNull) = 0;
```

Description

This is GetInt16, a member of class TDBXByteReader.

55.1.5.4.8 TDBXByteReader.GetInt32 Method

Pascal

```
procedure GetInt32(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull:
LongBool); virtual; abstract;
```

C++

```
virtual __fastcall GetInt32(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool
IsNull) = 0;
```

Description

This is GetInt32, a member of class TDBXByteReader.

55.1.5.4.9 TDBXByteReader.GetInt64 Method

Pascal

```
procedure GetInt64(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull:
LongBool); virtual; abstract;
```

C++

```
virtual __fastcall GetInt64(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool
IsNull) = 0;
```

Description

This is GetInt64, a member of class TDBXByteReader.

55.1.5.4.10 TDBXByteReader.GetTime Method

Pascal

```
procedure GetTime(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull: LongBool);
virtual; abstract;
```

C++

```
virtual __fastcall GetTime(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool IsNull)
= 0;
```

Description

This is GetTime, a member of class TDBXByteReader.

55.1.5.4.11 TDBXByteReader.GetTimeStamp Method

Pascal

```
procedure GetTimeStamp(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull:
LongBool); virtual; abstract;
```

C++

```
virtual __fastcall GetTimeStamp(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool
IsNull) = 0;
```

Description

This is GetTimeStamp, a member of class TDBXByteReader.

55.1.5.4.12 TDBXByteReader.GetWideString Method

Pascal

```
procedure GetWideString(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull:
LongBool); virtual; abstract;
```

C++

```
virtual __fastcall GetWideString(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool
IsNull) = 0;
```

Description

This is GetWideString, a member of class TDBXByteReader.

55.1.6 TDBXByteReaderEx Class

Class Hierarchy

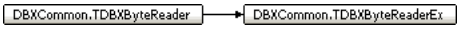

Pascal

```
TDBXByteReaderEx = class(TDBXByteReader);
```

C++

```
class TDBXByteReaderEx : public TDBXByteReader;
```

File

DBXCommon

Description

This is class DBXCommon.TDBXByteReaderEx.

55.1.6.1 TDBXByteReaderEx Members

The following tables list the members exposed by TDBXByteReaderEx.

Protected Methods

	Name	Description
	Create (see page 10369)	Creates an instance of TDBXByteReader (see page 10368)

Public Methods

	Name	Description
	GetAnsiString (see page 10370)	This is GetAnsiString, a member of class TDBXByteReader.
	GetBcd (see page 10370)	This is GetBcd, a member of class TDBXByteReader.
	GetBoolean (see page 10374)	This is GetBoolean, a member of class TDBXByteReaderEx.
	GetByteLength (see page 10370)	This is GetByteLength, a member of class TDBXByteReader.
	GetBytes (see page 10370)	This is GetBytes, a member of class TDBXByteReader.
	GetDate (see page 10371)	This is GetDate, a member of class TDBXByteReader.
	GetDouble (see page 10371)	This is GetDouble, a member of class TDBXByteReader.
	GetInt16 (see page 10371)	This is GetInt16, a member of class TDBXByteReader.
	GetInt32 (see page 10371)	This is GetInt32, a member of class TDBXByteReader.
	GetInt64 (see page 10372)	This is GetInt64, a member of class TDBXByteReader.
	GetTime (see page 10372)	This is GetTime, a member of class TDBXByteReader.
	GetTimeStamp (see page 10372)	This is GetTimeStamp, a member of class TDBXByteReader.
	GetWideString (see page 10372)	This is GetWideString, a member of class TDBXByteReader.

Strict protected Fields

	Name	Description
	FDBXContext (see page 10369)	This is FDBXContext, a member of class TDBXByteReader.

55.1.6.2 TDBXByteReaderEx Methods

The methods of the TDBXByteReaderEx class are listed here.

Public Methods

	Name	Description
	GetBoolean (see page 10374)	This is GetBoolean, a member of class TDBXByteReaderEx.

55.1.6.2.1 TDBXByteReaderEx.GetBoolean Method

Pascal

```
procedure GetBoolean(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull:
LongBool); virtual;
```

C++

```
virtual __fastcall GetBoolean(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool
IsNull);
```

Description

This is GetBoolean, a member of class TDBXByteReaderEx.

55.1.7 TDBXByteStreamReader Class

Used internally by driver implementations and TDBXStreamValue (see page 10608).

Class Hierarchy

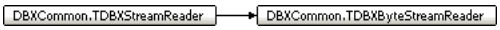

Pascal

```
TDBXByteStreamReader = class(TDBXStreamReader);
```

C++

```
class TDBXByteStreamReader : public TDBXStreamReader;
```

File

DBXCommon

Description

TDBXByteStreamReader is used internally by driver implementations and TDBXStreamValue (see page 10608).

See Also

TDBXStreamValue (see page 10608)

55.1.7.1 TDBXByteStreamReader Members

The following tables list the members exposed by TDBXByteStreamReader.

Public Fields

	Name	Description
	FBytes (see page 10376)	This is FBytes, a member of class TDBXByteStreamReader.
	FCount (see page 10376)	This is FCount, a member of class TDBXByteStreamReader.
	FEOS (see page 10376)	This is FEOS, a member of class TDBXByteStreamReader.
	FOffset (see page 10376)	This is FOffset, a member of class TDBXByteStreamReader.
	FPosition (see page 10376)	This is FPosition, a member of class TDBXByteStreamReader.

Public Methods

	Name	Description
	Create (see page 10375)	This is Create, a member of class TDBXByteStreamReader.
V	Destroy (see page 10375)	This is Destroy, a member of class TDBXByteStreamReader.
V	Eos (see page 10377)	This is Eos, a member of class TDBXByteStreamReader.
V	Read (see page 10377)	This is Read, a member of class TDBXByteStreamReader.
V	Size (see page 10377)	This is Size, a member of class TDBXByteStreamReader.

55.1.7.2 TDBXByteStreamReader.Create Constructor

Pascal

```
constructor Create(Bytes: TBytes; Offset: Integer; Count: Integer);
```

C++

```
__fastcall TDBXByteStreamReader(TBytes Bytes, int Offset, int Count);
```

Description

This is Create, a member of class TDBXByteStreamReader.

55.1.7.3 TDBXByteStreamReader.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXByteStreamReader();
```

Description

This is Destroy, a member of class TDBXByteStreamReader.

55.1.7.4 TDBXByteStreamReader Fields

The fields of the TDBXByteStreamReader class are listed here.

Public Fields

	Name	Description
	FBytes (see page 10376)	This is FBytes, a member of class TDBXByteStreamReader.
	FCount (see page 10376)	This is FCount, a member of class TDBXByteStreamReader.
	FEOS (see page 10376)	This is FEOS, a member of class TDBXByteStreamReader.
	FOffset (see page 10376)	This is FOffset, a member of class TDBXByteStreamReader.
	FPosition (see page 10376)	This is FPosition, a member of class TDBXByteStreamReader.

55.1.7.4.1 TDBXByteStreamReader.FBytes Field

Pascal

```
FBytes: TBytes;
```

C++

```
TBytes FBytes;
```

Description

This is FBytes, a member of class TDBXByteStreamReader.

55.1.7.4.2 TDBXByteStreamReader.FCount Field

Pascal

```
FCount: Integer;
```

C++

```
int FCount;
```

Description

This is FCount, a member of class TDBXByteStreamReader.

55.1.7.4.3 TDBXByteStreamReader.FEOS Field

Pascal

```
FEOS: Boolean;
```

C++

```
Boolean FEOS;
```

Description

This is FEOS, a member of class TDBXByteStreamReader.

55.1.7.4.4 TDBXByteStreamReader.FOffset Field

Pascal

```
FOffset: Integer;
```

C++

```
int FOffset;
```

Description

This is FOffset, a member of class TDBXByteStreamReader.

55.1.7.4.5 TDBXByteStreamReader.FPosition Field

Pascal

```
FPosition: Integer;
```

C++

```
int FPosition;
```

Description

This is FPosition, a member of class TDBXByteStreamReader.

55.1.7.5 TDBXByteStreamReader Methods

The methods of the TDBXByteStreamReader class are listed here.

Public Methods

	Name	Description
	Eos (see page 10377)	This is Eos, a member of class TDBXByteStreamReader.
	Read (see page 10377)	This is Read, a member of class TDBXByteStreamReader.
	Size (see page 10377)	This is Size, a member of class TDBXByteStreamReader.

55.1.7.5.1 TDBXByteStreamReader.Eos Method

Pascal

```
function Eos: Boolean; override;
```

C++

```
virtual __fastcall Boolean Eos();
```

Description

This is Eos, a member of class TDBXByteStreamReader.

55.1.7.5.2 TDBXByteStreamReader.Read Method

Pascal

```
function Read(const Buffer: TBytes; Offset: Integer; Count: Integer): Integer; override;
```

C++

```
virtual __fastcall int Read(const TBytes Buffer, int Offset, int Count);
```

Description

This is Read, a member of class TDBXByteStreamReader.

55.1.7.5.3 TDBXByteStreamReader.Size Method

Pascal

```
function Size: Int64; override;
```

C++

```
virtual __fastcall Int64 Size();
```

Description

This is Size, a member of class TDBXByteStreamReader.

55.1.8 TDBXCommand Class

Class to execute SQL statements and stored procedures

Class Hierarchy

DBXCommon.TDBXCommand

Pascal

```
TDBXCommand = class;
```

C++

```
class TDBXCommand;
```

File

DBXCommon

Description

The TDBXCommand class is typically used to execute SQL statements and stored procedures.

The TDBXCommand.Text (see page 10390) property is set to the desired command to execute.

Parameters (see page 10389) can be added and accessed through the TDBXCommand.Parameters (see page 10389) property. Parameters (see page 10389) can be created by calling the TDBXCommand.CreateParameter (see page 10382) method.

TDBXCommand can also execute other types of commands. The TDBXCommandTypes (see page 10396) class lists constants for common command types. Because the command type is represented as a `String`, driver implementations can use TDBXCommandTypes (see page 10396) as a namespace to create additional types of commands. For example, the TDBXPool delegate driver has a command type that allows a command to return a TDBXReader (see page 10576) with information on the active connection pools.

TDBXCommand instances can be created by calling one of the TDBXConnection.CreateCommandmethods. As soon as an application is finished using a command, the TDBXCommand.Free method must be called. This releases memory for the command and any associated resources.

See Also

Connecting to a Database using the dbExpress Driver Framework

TDBXCommandTypes (see page 10396)

TDBXConnection (see page 10398)

TDBXReader (see page 10576)

Parameters (see page 10389)

Text (see page 10390)

CreateParameter (see page 10382)

CreateCommand (see page 10404)

55.1.8.1 TDBXCommand Members

The following tables list the members exposed by TDBXCommand.

Protected Fields

	Name	Description
	FDBXContext (see page 10381)	This is FDBXContext, a member of class TDBXCommand.

Protected Methods

	Name	Description
V	Close (see page 10382)	This is Close, a member of class TDBXCommand.
	Create (see page 10380)	Creates an instance of TDBXCommand (see page 10378)
V	CreateParameterRow (see page 10382)	This is CreateParameterRow, a member of class TDBXCommand.
V	CreateParameters (see page 10382)	This is CreateParameters, a member of class TDBXCommand.
A	DerivedClose (see page 10383)	This is DerivedClose, a member of class TDBXCommand.
A	DerivedExecuteQuery (see page 10383)	This is DerivedExecuteQuery, a member of class TDBXCommand.
A	DerivedExecuteUpdate (see page 10383)	This is DerivedExecuteUpdate, a member of class TDBXCommand.
A	DerivedGetNextReader (see page 10383)	This is DerivedGetNextReader, a member of class TDBXCommand.
A	DerivedOpen (see page 10384)	This is DerivedOpen, a member of class TDBXCommand.
A	DerivedPrepare (see page 10384)	This is DerivedPrepare, a member of class TDBXCommand.
V	GetCommandType (see page 10385)	This is GetCommandType, a member of class TDBXCommand.
V	GetErrorEvent (see page 10385)	This is GetErrorEvent, a member of class TDBXCommand.
V	GetParameters (see page 10385)	This is GetParameters, a member of class TDBXCommand.
A	GetRowsAffected (see page 10386)	This is GetRowsAffected, a member of class TDBXCommand.
V	GetText (see page 10386)	This is GetText, a member of class TDBXCommand.
V	Open (see page 10386)	This is Open, a member of class TDBXCommand.
V	SetCommandType (see page 10387)	This is SetCommandType, a member of class TDBXCommand.
A	SetMaxBlobSize (see page 10387)	This is SetMaxBlobSize, a member of class TDBXCommand.
A	SetRowSetSize (see page 10387)	This is SetRowSetSize, a member of class TDBXCommand.
V	SetText (see page 10387)	This is SetText, a member of class TDBXCommand.

Public Events

	Name	Description
R	OnErrorEvent (see page 10391)	Error handling event

Public Methods

	Name	Description
V	CreateParameter (see page 10382)	Create (see page 10380) a parameter
V	Destroy (see page 10380)	Destroy a command instance
V	ExecuteQuery (see page 10384)	Executes a command that returns an instance of TDBXReader (see page 10576)
V	ExecuteUpdate (see page 10384)	Executes an update command
V	GetNextReader (see page 10385)	Gets next instance of TDBXReader (see page 10576)
V	Prepare (see page 10386)	Prepare the command for execution

Public Properties

	Name	Description
	CommandType (see page 10388)	Command type
R	IsPrepared (see page 10388)	Is command prepared.
W	MaxBlobSize (see page 10389)	Maximum blob size
R	Parameters (see page 10389)	List of parameters
R	RowsAffected (see page 10389)	Number of rows updated
W	RowSetSize (see page 10390)	Row buffering property used to optimize reading data from TDBXReader (see page 10576) instances.

	Text (see page 10390)	Command to execute

Strict protected Fields

	Name	Description
	FParameters (see page 10381)	This is FParameters, a member of class TDBXCommand.

Strict protected Methods

	Name	Description
	NotImplemented (see page 10386)	This is NotImplemented, a member of class TDBXCommand.

55.1.8.2 TDBXCommand.Create Constructor

Creates an instance of TDBXCommand (see page 10378)

Pascal

```
constructor Create(DBXContext: TDBXContext);
```

C++

```
__fastcall TDBXCommand(TDBXContext DBXContext);
```

Description

This constructor creates and initializes an instance of TDBXCommand (see page 10378).

See Also

Destroy (see page 10380)

55.1.8.3 TDBXCommand.Destroy Destructor

Destroy a command instance

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXCommand();
```

Description

This method destroys a TDBXCommand (see page 10378) instance.

55.1.8.4 TDBXCommand Fields

The fields of the TDBXCommand class are listed here.

Protected Fields

	Name	Description
	FDBXContext (see page 10381)	This is FDBXContext, a member of class TDBXCommand.

Strict protected Fields

	Name	Description
	FParameters (see page 10381)	This is FParameters, a member of class TDBXCommand.

55.1.8.4.1 TDBXCommand.FDBXContext Field

Pascal

```
FDBXContext: TDBXContext;
```

C++

```
TDBXContext FDBXContext;
```

Description

This is FDBXContext, a member of class TDBXCommand.

55.1.8.4.2 TDBXCommand.FParameters Field

Pascal

```
FParameters: TDBXParameterList;
```

C++

```
TDBXParameterList FParameters;
```

Description

This is FParameters, a member of class TDBXCommand.

55.1.8.5 TDBXCommand Methods

The methods of the TDBXCommand class are listed here.

Protected Methods

	Name	Description
V	Close (see page 10382)	This is Close, a member of class TDBXCommand.
V	CreateParameterRow (see page 10382)	This is CreateParameterRow, a member of class TDBXCommand.
V	CreateParameters (see page 10382)	This is CreateParameters, a member of class TDBXCommand.
A	DerivedClose (see page 10383)	This is DerivedClose, a member of class TDBXCommand.
A	DerivedExecuteQuery (see page 10383)	This is DerivedExecuteQuery, a member of class TDBXCommand.
A	DerivedExecuteUpdate (see page 10383)	This is DerivedExecuteUpdate, a member of class TDBXCommand.
A	DerivedGetNextReader (see page 10383)	This is DerivedGetNextReader, a member of class TDBXCommand.
A	DerivedOpen (see page 10384)	This is DerivedOpen, a member of class TDBXCommand.
A	DerivedPrepare (see page 10384)	This is DerivedPrepare, a member of class TDBXCommand.
V	GetCommandType (see page 10385)	This is GetCommandType, a member of class TDBXCommand.
V	GetErrorEvent (see page 10385)	This is GetErrorEvent, a member of class TDBXCommand.
V	GetParameters (see page 10385)	This is GetParameters, a member of class TDBXCommand.
A	GetRowsAffected (see page 10386)	This is GetRowsAffected, a member of class TDBXCommand.
V	GetText (see page 10386)	This is GetText, a member of class TDBXCommand.
V	Open (see page 10386)	This is Open, a member of class TDBXCommand.
V	SetCommandType (see page 10387)	This is SetCommandType, a member of class TDBXCommand.
A	SetMaxBlobSize (see page 10387)	This is SetMaxBlobSize, a member of class TDBXCommand.
A	SetRowSetSize (see page 10387)	This is SetRowSetSize, a member of class TDBXCommand.
V	SetText (see page 10387)	This is SetText, a member of class TDBXCommand.

Public Methods

	Name	Description
V	CreateParameter (see page 10382)	Create (see page 10380) a parameter

	ExecuteQuery (see page 10384)	Executes a command that returns an instance of TDBXReader (see page 10576)
	ExecuteUpdate (see page 10384)	Executes an update command
	GetNextReader (see page 10385)	Gets next instance of TDBXReader (see page 10576)
	Prepare (see page 10386)	Prepare the command for execution

Strict protected Methods

	Name	Description
	NotImplemented (see page 10386)	This is NotImplemented, a member of class TDBXCommand.

55.1.8.5.1 TDBXCommand.Close Method

Pascal

```
procedure Close; virtual;
```

C++

```
virtual __fastcall Close();
```

Description

This is Close, a member of class TDBXCommand.

55.1.8.5.2 TDBXCommand.CreateParameter Method

Create (see page 10380) a parameter

Pascal

```
function CreateParameter: TDBXParameter; virtual;
```

C++

```
virtual __fastcall TDBXParameter CreateParameter();
```

Description

This method allocates a new instance of TDBXParameter (see page 10545), which can be added to the Parameterslist.

See Also

TDBXParameter (see page 10545)

55.1.8.5.3 TDBXCommand.CreateParameterRow Method

Pascal

```
function CreateParameterRow: TDBXRow; virtual;
```

C++

```
virtual __fastcall TDBXRow CreateParameterRow();
```

Description

This is CreateParameterRow, a member of class TDBXCommand.

55.1.8.5.4 TDBXCommand.CreateParameters Method

Pascal

```
procedure CreateParameters(Command: TDBXCommand); virtual;
```

C++

```
virtual __fastcall CreateParameters(TDBXCommand Command);
```

Description

This is CreateParameters, a member of class TDBXCommand.

55.1.8.5.5 TDBXCommand.DerivedClose Method

Pascal

```
procedure DerivedClose; virtual; abstract;
```

C++

```
virtual __fastcall DerivedClose() = 0;
```

Description

This is DerivedClose, a member of class TDBXCommand.

55.1.8.5.6 TDBXCommand.DerivedExecuteQuery Method

Pascal

```
function DerivedExecuteQuery: TDBXReader; virtual; abstract;
```

C++

```
virtual __fastcall TDBXReader DerivedExecuteQuery() = 0;
```

Description

This is DerivedExecuteQuery, a member of class TDBXCommand.

55.1.8.5.7 TDBXCommand.DerivedExecuteUpdate Method

Pascal

```
procedure DerivedExecuteUpdate; virtual; abstract;
```

C++

```
virtual __fastcall DerivedExecuteUpdate() = 0;
```

Description

This is DerivedExecuteUpdate, a member of class TDBXCommand.

55.1.8.5.8 TDBXCommand.DerivedGetNextReader Method

Pascal

```
function DerivedGetNextReader: TDBXReader; virtual; abstract;
```

C++

```
virtual __fastcall TDBXReader DerivedGetNextReader() = 0;
```

Description

This is DerivedGetNextReader, a member of class TDBXCommand.

55.1.8.5.9 TDBXCommand.DerivedOpen Method

Pascal

```
procedure DerivedOpen; virtual; abstract;
```

C++

```
virtual __fastcall DerivedOpen() = 0;
```

Description

This is DerivedOpen, a member of class TDBXCommand.

55.1.8.5.10 TDBXCommand.DerivedPrepare Method

Pascal

```
procedure DerivedPrepare; virtual; abstract;
```

C++

```
virtual __fastcall DerivedPrepare() = 0;
```

Description

This is DerivedPrepare, a member of class TDBXCommand.

55.1.8.5.11 TDBXCommand.ExecuteQuery Method

Executes a command that returns an instance of TDBXReader (see page 10576)

Pascal

```
function ExecuteQuery: TDBXReader; virtual;
```

C++

```
virtual __fastcall TDBXReader ExecuteQuery();
```

Description

This method executes a command that returns an instance of TDBXReader (see page 10576). Use a TDBXReader (see page 10576) instance to read database records and data.

Note: As soon as an application is done with a TDBXReader (see page 10576) instance, it should call TDBXReader.Free to free resources.

See Also

Connecting to a Database using the dbExpress Driver Framework

TDBXReader (see page 10576)

Free

55.1.8.5.12 TDBXCommand.ExecuteUpdate Method

Executes an update command

Pascal

```
procedure ExecuteUpdate; virtual;
```

C++

```
virtual __fastcall ExecuteUpdate();
```

Description

This method executes a command that performs an update operation.

55.1.8.5.13 TDBXCommand.GetCommandType Method

Pascal

```
function GetCommandType: WideString; virtual;
```

C++

```
virtual __fastcall BSTR GetCommandType();
```

Description

This is GetCommandType, a member of class TDBXCommand.

55.1.8.5.14 TDBXCommand.GetErrorEvent Method

Pascal

```
function GetErrorEvent: TDBXErrorEvent; virtual;
```

C++

```
virtual __fastcall TDBXErrorEvent GetErrorEvent();
```

Description

This is GetErrorEvent, a member of class TDBXCommand.

55.1.8.5.15 TDBXCommand.GetNextReader Method

Gets next instance of TDBXReader (see page 10576)

Pascal

```
function GetNextReader: TDBXReader; virtual;
```

C++

```
virtual __fastcall TDBXReader GetNextReader();
```

Description

For commands that return more then one reader, this method gets the next instance of TDBXReader (see page 10576). If there are no more TDBXReader (see page 10576) instances, it returns `nil`.

See Also

TDBXReader (see page 10576)

55.1.8.5.16 TDBXCommand.GetParameters Method

Pascal

```
function GetParameters: TDBXParameterList; virtual;
```

C++

```
virtual __fastcall TDBXParameterList GetParameters();
```

Description

This is GetParameters, a member of class TDBXCommand.

55.1.8.5.17 TDBXCommand.GetRowsAffected Method

Pascal

```
function GetRowsAffected: Int64; virtual; abstract;
```

C++

```
virtual __fastcall Int64 GetRowsAffected() = 0;
```

Description

This is GetRowsAffected, a member of class TDBXCommand.

55.1.8.5.18 TDBXCommand.GetText Method

Pascal

```
function GetText: WideString; virtual;
```

C++

```
virtual __fastcall BSTR GetText();
```

Description

This is GetText, a member of class TDBXCommand.

55.1.8.5.19 TDBXCommand.NotImplemented Method

Pascal

```
procedure NotImplemented;
```

C++

```
__fastcall NotImplemented();
```

Description

This is NotImplemented, a member of class TDBXCommand.

55.1.8.5.20 TDBXCommand.Open Method

Pascal

```
procedure Open; virtual;
```

C++

```
virtual __fastcall Open();
```

Description

This is Open, a member of class TDBXCommand.

55.1.8.5.21 TDBXCommand.Prepare Method

Prepare the command for execution

Pascal

```
procedure Prepare; virtual;
```

C++

```
virtual __fastcall Prepare();
```

Description

This method prepares the command for execution.

Note: Prepare must be called before calling any of the TDBXCommand (see page 10378) Execute methods.

55.1.8.5.22 TDBXCommand.SetCommandType Method

Pascal

```
procedure SetCommandType(const CommandType: WideString); virtual;
```

C++

```
virtual __fastcall SetCommandType(const BSTR CommandType);
```

Description

This is SetCommandType, a member of class TDBXCommand.

55.1.8.5.23 TDBXCommand.SetMaxBlobSize Method

Pascal

```
procedure SetMaxBlobSize(const MaxBlobSize: Int64); virtual; abstract;
```

C++

```
virtual __fastcall SetMaxBlobSize(const Int64 MaxBlobSize) = 0;
```

Description

This is SetMaxBlobSize, a member of class TDBXCommand.

55.1.8.5.24 TDBXCommand.SetRowSetSize Method

Pascal

```
procedure SetRowSetSize(const RowSetSize: Int64); virtual; abstract;
```

C++

```
virtual __fastcall SetRowSetSize(const Int64 RowSetSize) = 0;
```

Description

This is SetRowSetSize, a member of class TDBXCommand.

55.1.8.5.25 TDBXCommand.SetText Method

Pascal

```
procedure SetText(const Value: WideString); virtual;
```

C++

```
virtual __fastcall SetText(const BSTR Value);
```

Description

This is SetText, a member of class TDBXCommand.

55.1.8.6 TDBXCommand Properties

The properties of the TDBXCommand class are listed here.

Public Properties

	Name	Description
	CommandType (see page 10388)	Command type
R	IsPrepared (see page 10388)	Is command prepared.
W	MaxBlobSize (see page 10389)	Maximum blob size
R	Parameters (see page 10389)	List of parameters
R	RowsAffected (see page 10389)	Number of rows updated
W	RowSetSize (see page 10390)	Row buffering property used to optimize reading data from TDBXReader (see page 10576) instances.
	Text (see page 10390)	Command to execute

55.1.8.6.1 TDBXCommand.CommandType Property

Command type

Pascal

```
property CommandType: WideString;
```

C++

```
__property BSTR CommandType;
```

Description

This property contains the command type, which is typically set to one of the constant strings in the TDBXCommandTypes (see page 10396) class. However, driver implementors can use this property as a namespace to create new types of commands.

See Also

TDBXCommandTypes (see page 10396)

55.1.8.6.2 TDBXCommand.IsPrepared Property

Is command prepared.

Pascal

```
property IsPrepared: Boolean;
```

C++

```
__property Boolean IsPrepared;
```

Description

This property is true if the command has been prepared and false otherwise.

See Also

Prepare (see page 10386)

55.1.8.6.3 TDBXCommand.MaxBlobSize Property

Maximum blob size

Pascal

```
property MaxBlobSize: Int64;
```

C++

```
__property Int64 MaxBlobSize;
```

Description

This property holds the maximum blob size allowed for TDBXReader (see page 10576) instances returned from ExecuteQuery (see page 10384).

See Also

TDBXReader (see page 10576)

ExecuteQuery (see page 10384)

55.1.8.6.4 TDBXCommand.Parameters Property

List of parameters

Pascal

```
property Parameters: TDBXParameterList;
```

C++

```
__property TDBXParameterList Parameters;
```

Description

This property contains a TDBXParametersList list of parameters used to get and set parameters for Text (see page 10390) property commands that contain '?' parameter markers.

See Also

TDBXParametersList

Text (see page 10390)

55.1.8.6.5 TDBXCommand.RowsAffected Property

Number of rows updated

Pascal

```
property RowsAffected: Int64;
```

C++

```
__property Int64 RowsAffected;
```

Description

This property holds the number of rows updated from the last call to ExecuteUpdate (see page 10384).

See Also

ExecuteUpdate (see page 10384)

55.1.8.6.6 TDBXCommand.RowSetSize Property

Row buffering property used to optimize reading data from TDBXReader (see page 10576) instances.

Pascal

```
property RowSetSize: Int64;
```

C++

```
__property Int64 RowSetSize;
```

Description

RowSetSize is a row buffering property used to optimize reading data from TDBXReader (see page 10576) instances returned from ExecuteQuery (see page 10384). It only takes affect for database vendors that return true for the TDBXDataBaseMetadata.SupportsRowSetSize property.

See Also

TDBXReader (see page 10576)

SupportsRowSetSize (see page 10443)

ExecuteQuery (see page 10384)

55.1.8.6.7 TDBXCommand.Text Property

Command to execute

Pascal

```
property Text: WideString;
```

C++

```
__property BSTR Text;
```

Description

This property contains the text of the command to execute. When CommandType (see page 10388) is set to TDBXCommandTypes.SQLStatement, Text can be set to SQL statements such as `select`, `insert`, `update`, and `delete`.

See Also

Connecting to a Database using the dbExpress Driver Framework

TDBXCommandTypes (see page 10396)

CommandType (see page 10388)

Text

55.1.8.7 TDBXCommand Events

The events of the TDBXCommand class are listed here.

Public Events

	Name	Description
R	OnErrorEvent (see page 10391)	Error handling event

55.1.8.7.1 TDBXCommand.OnErrorEvent Event

Error handling event

Pascal

```
property OnErrorEvent: TDBXErrorEvent;
```

C++

```
__property TDBXErrorEvent OnErrorEvent;
```

Description

This property contains the error handling event provided by the TDBXConnection (see page 10398) that created this command. This event is a TDBXErrorEvent (see page 10671) that is called just before any dbExpress TDBXError (see page 10486) exception is raised.

See Also

TDBXConnection (see page 10398)

TDBXError (see page 10486)

TDBXErrorEvent (see page 10671)

55.1.9 TDBXCommandEx Class

Extension of TDBXCommand (see page 10378).

Class Hierarchy

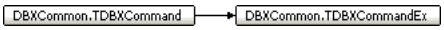

Pascal

```
TDBXCommandEx = class(TDBXCommand);
```

C++

```
class TDBXCommandEx : public TDBXCommand;
```

File

DBXCommon

Description

TDBXCommandEx is an extension of TDBXCommand (see page 10378).

Note: Classes (see page 2685) ending in “Ex” are temporary classes that exist to avoid breaking interfaces. Their contents will be migrated to their base class in the next release.

See Also

TDBXCommand (see page 10378)

55.1.9.1 TDBXCommandEx Members

The following tables list the members exposed by TDBXCommandEx.

Protected Fields

	Name	Description
	FDBXContext (see page 10381)	This is FDBXContext, a member of class TDBXCommand.

Protected Methods

	Name	Description
V	Close (see page 10382)	This is Close, a member of class TDBXCommand.
	Create (see page 10380)	Creates an instance of TDBXCommand (see page 10378)
V	CreateParameterRow (see page 10382)	This is CreateParameterRow, a member of class TDBXCommand.
V	CreateParameters (see page 10382)	This is CreateParameters, a member of class TDBXCommand.
A	DerivedClose (see page 10383)	This is DerivedClose, a member of class TDBXCommand.
A	DerivedExecuteQuery (see page 10383)	This is DerivedExecuteQuery, a member of class TDBXCommand.
A	DerivedExecuteUpdate (see page 10383)	This is DerivedExecuteUpdate, a member of class TDBXCommand.
A	DerivedGetNextReader (see page 10383)	This is DerivedGetNextReader, a member of class TDBXCommand.
A	DerivedOpen (see page 10384)	This is DerivedOpen, a member of class TDBXCommand.
A	DerivedPrepare (see page 10384)	This is DerivedPrepare, a member of class TDBXCommand.
V	GetCommandTimeout (see page 10393)	This is GetCommandTimeout, a member of class TDBXCommandEx.
V	GetCommandType (see page 10385)	This is GetCommandType, a member of class TDBXCommand.
V	GetErrorEvent (see page 10385)	This is GetErrorEvent, a member of class TDBXCommand.
V	GetParameters (see page 10385)	This is GetParameters, a member of class TDBXCommand.
A	GetRowsAffected (see page 10386)	This is GetRowsAffected, a member of class TDBXCommand.
V	GetText (see page 10386)	This is GetText, a member of class TDBXCommand.
V	Open (see page 10386)	This is Open, a member of class TDBXCommand.
V	SetCommandTimeout (see page 10393)	This is SetCommandTimeout, a member of class TDBXCommandEx.
V	SetCommandType (see page 10387)	This is SetCommandType, a member of class TDBXCommand.
A	SetMaxBlobSize (see page 10387)	This is SetMaxBlobSize, a member of class TDBXCommand.
A	SetRowSetSize (see page 10387)	This is SetRowSetSize, a member of class TDBXCommand.
V	SetText (see page 10387)	This is SetText, a member of class TDBXCommand.

Public Events

	Name	Description
R	OnErrorEvent (see page 10391)	Error handling event

Public Methods

	Name	Description
V	CreateParameter (see page 10382)	Create (see page 10380) a parameter
V	Destroy (see page 10380)	Destroy a command instance
V	ExecuteQuery (see page 10384)	Executes a command that returns an instance of TDBXReader (see page 10576)
V	ExecuteUpdate (see page 10384)	Executes an update command
V	GetNextReader (see page 10385)	Gets next instance of TDBXReader (see page 10576)
V	Prepare (see page 10386)	Prepare the command for execution

Public Properties

	Name	Description
	CommandTimeout (see page 10394)	Time-out value for the execution of a command.
	CommandType (see page 10388)	Command type
R	IsPrepared (see page 10388)	Is command prepared.
W	MaxBlobSize (see page 10389)	Maximum blob size
R	Parameters (see page 10389)	List of parameters
R	RowsAffected (see page 10389)	Number of rows updated
W	RowSetSize (see page 10390)	Row buffering property used to optimize reading data from TDBXReader (see page 10576) instances.
	Text (see page 10390)	Command to execute

Strict protected Fields

	Name	Description
	FParameters (see page 10381)	This is FParameters, a member of class TDBXCommand.

Strict protected Methods

	Name	Description
	NotImplemented (see page 10386)	This is NotImplemented, a member of class TDBXCommand.
	SetLastReader (see page 10394)	This is SetLastReader, a member of class TDBXCommandEx.

Strict protected Properties

	Name	Description
W	LastReader (see page 10394)	This is LastReader, a member of class TDBXCommandEx.

55.1.9.2 TDBXCommandEx Methods

The methods of the TDBXCommandEx class are listed here.

Protected Methods

	Name	Description
V	GetCommandTimeout (see page 10393)	This is GetCommandTimeout, a member of class TDBXCommandEx.
V	SetCommandTimeout (see page 10393)	This is SetCommandTimeout, a member of class TDBXCommandEx.

Strict protected Methods

	Name	Description
	SetLastReader (see page 10394)	This is SetLastReader, a member of class TDBXCommandEx.

55.1.9.2.1 TDBXCommandEx.GetCommandTimeout Method

Pascal

```
function GetCommandTimeout: Integer; virtual;
```

C++

```
virtual __fastcall int GetCommandTimeout();
```

Description

This is GetCommandTimeout, a member of class TDBXCommandEx.

55.1.9.2.2 TDBXCommandEx.SetCommandTimeout Method

Pascal

```
procedure SetCommandTimeout(Timeout: Integer); virtual;
```

C++

```
virtual __fastcall SetCommandTimeout(int Timeout);
```

Description

This is SetCommandTimeout, a member of class TDBXCommandEx.

55.1.9.2.3 TDBXCommandEx.SetLastReader Method

Pascal

```
procedure SetLastReader(Reader: TDBXReader);
```

C++

```
__fastcall SetLastReader(TDBXReader Reader);
```

Description

This is SetLastReader, a member of class TDBXCommandEx.

55.1.9.3 TDBXCommandEx Properties

The properties of the TDBXCommandEx class are listed here.

Public Properties

	Name	Description
	CommandTimeout (see page 10394)	Time-out value for the execution of a command.

Strict protected Properties

	Name	Description
	LastReader (see page 10394)	This is LastReader, a member of class TDBXCommandEx.

55.1.9.3.1 TDBXCommandEx.CommandTimeout Property

Time-out value for the execution of a command.

Pascal

```
property CommandTimeout: Integer;
```

C++

```
__property int CommandTimeout;
```

Description

Time-out value for the execution of a command. Not all drivers support this. Time in seconds to wait for a command to execute.

55.1.9.3.2 TDBXCommandEx.LastReader Property

Pascal

```
property LastReader: TDBXReader;
```

C++

```
__property TDBXReader LastReader;
```

Description

This is LastReader, a member of class TDBXCommandEx.

55.1.10 TDBXCommandFactory Class

Class Hierarchy

DBXCommon.TDBXCommandFactory

Pascal

```
TDBXCommandFactory = class;
```

C++

```
class TDBXCommandFactory;
```

File

DBXCommon

Description

This is class DBXCommon.TDBXCommandFactory.

55.1.10.1 TDBXCommandFactory Members

The following tables list the members exposed by TDBXCommandFactory.

Public Methods

	Name	Description
	CreateCommand (see page 10395)	This is CreateCommand, a member of class TDBXCommandFactory.

55.1.10.2 TDBXCommandFactory Methods

The methods of the TDBXCommandFactory class are listed here.

Public Methods

	Name	Description
	CreateCommand (see page 10395)	This is CreateCommand, a member of class TDBXCommandFactory.

55.1.10.2.1 TDBXCommandFactory.CreateCommand Method

Pascal

```
function CreateCommand(DbxContext: TDBXContext; Connection: TDBXConnection; MorphicCommand:
TDBXCommand): TDBXCommand; virtual; abstract;
```

C++

```
virtual __fastcall TDBXCommand CreateCommand(TDBXContext DbxContext, TDBXConnection
Connection, TDBXCommand MorphicCommand) = 0;
```

Description

This is CreateCommand, a member of class TDBXCommandFactory.

55.1.11 TDBXCommandTypes Class

Type of command to be executed by a TDBXCommand (see page 10378)

Class Hierarchy

DBXCommon.TDBXCommandTypes

Pascal

```
TDBXCommandTypes = class;
```

C++

```
class TDBXCommandTypes;
```

File

DBXCommon

Description

This class provides the types of commands that can be executed against a database using an instance of TDBXCommand (see page 10378).

This table lists the commands.

Constant	Description
DbxSQL (see page 10398)	Used for any SQL statement including selects, inserts, updates, deletes, and so on.
DbxStoredProcedure (see page 10398)	Used for any stored procedure.
DbxTable (see page 10398)	Used to retrieve all rows and columns of a table.
DbxMetaData (see page 10397)	Used to retrieve metadata objects such as tables and columns that are accessible from a TDBXConnection (see page 10398).
DbxCommand (see page 10397)	Deprecated. Do not use.
DbxPool (see page 10397)	Command type for TDBXPool delegate driver commands.

See Also

Connecting to a Database using the dbExpress Driver Framework

TDBXCommand (see page 10378)

TDBXConnection (see page 10398)

55.1.11.1 TDBXCommandTypes Members

The following tables list the members exposed by TDBXCommandTypes.

Public Constants

	Name	Description
	DbxCommand (see page 10397)	This is DbxCommand, a member of class TDBXCommandTypes.
	DbxMetaData (see page 10397)	This is DbxMetaData, a member of class TDBXCommandTypes.
	DbxPool (see page 10397)	This is DbxPool, a member of class TDBXCommandTypes.
	DbxSQL (see page 10398)	This is DbxSQL, a member of class TDBXCommandTypes.
	DbxStoredProcedure (see page 10398)	This is DbxStoredProcedure, a member of class TDBXCommandTypes.
	DbxTable (see page 10398)	This is DbxTable, a member of class TDBXCommandTypes.

55.1.11.2 TDBXCommandTypes Constants

The constants of the TDBXCommandTypes class are listed here.

Public Constants

	Name	Description
	DbxCommand (see page 10397)	This is DbxCommand, a member of class TDBXCommandTypes.
	DbxMetaData (see page 10397)	This is DbxMetaData, a member of class TDBXCommandTypes.
	DbxPool (see page 10397)	This is DbxPool, a member of class TDBXCommandTypes.
	DbxSQL (see page 10398)	This is DbxSQL, a member of class TDBXCommandTypes.
	DbxStoredProcedure (see page 10398)	This is DbxStoredProcedure, a member of class TDBXCommandTypes.
	DbxTable (see page 10398)	This is DbxTable, a member of class TDBXCommandTypes.

55.1.11.2.1 TDBXCommandTypes.DbxCommand Constant

Pascal

```
const DbxCommand = 'Dbx.Command';
```

C++

```
DbxCommand = 'Dbx.Command';
```

Description

This is DbxCommand, a member of class TDBXCommandTypes.

55.1.11.2.2 TDBXCommandTypes.DbxMetaData Constant

Pascal

```
const DbxMetaData = 'Dbx.MetaData';
```

C++

```
DbxMetaData = 'Dbx.MetaData';
```

Description

This is DbxMetaData, a member of class TDBXCommandTypes.

55.1.11.2.3 TDBXCommandTypes.DbxPool Constant

Pascal

```
const DbxPool = 'Dbx.Pool';
```

C++

```
DbxPool = 'Dbx.Pool';
```

Description

This is DbxPool, a member of class TDBXCommandTypes.

55.1.11.2.4 TDBXCommandTypes.DbxSQL Constant

Pascal

```
const DbxSQL = 'Dbx.SQL';
```

C++

```
DbxSQL = 'Dbx.SQL';
```

Description

This is DbxSQL, a member of class TDBXCommandTypes.

55.1.11.2.5 TDBXCommandTypes.DbxStoredProcedure Constant

Pascal

```
const DbxStoredProcedure = 'Dbx.StoredProcedure';
```

C++

```
DbxStoredProcedure = 'Dbx.StoredProcedure';
```

Description

This is DbxStoredProcedure, a member of class TDBXCommandTypes.

55.1.11.2.6 TDBXCommandTypes.DbxTable Constant

Pascal

```
const DbxTable = 'Dbx.Table';
```

C++

```
DbxTable = 'Dbx.Table';
```

Description

This is DbxTable, a member of class TDBXCommandTypes.

55.1.12 TDBXConnection Class

Connection for the dbExpress driver framework

Class Hierarchy

Pascal

```
TDBXConnection = class;
```

C++

```
class TDBXConnection;
```

File

DBXCommon

Description

This class represents a connection for the dbExpress driver framework. It provides support for creating commands, retrieving database records and related metadata, and transaction management.

Use the TDBXConnection.CreateCommand (see page 10404) to create a TDBXReader (see page 10576) object to read database records.

See Also

Connecting to a Database using the dbExpress Driver Framework

TDBXReader (see page 10576)

55.1.12.1 TDBXConnection Members

The following tables list the members exposed by TDBXConnection.

Protected Fields

	Name	Description
	FConnectionProperties (see page 10401)	This is FConnectionProperties, a member of class TDBXConnection.
	FDBXContext (see page 10401)	This is FDBXContext, a member of class TDBXConnection.

Protected Methods

	Name	Description
V	Close (see page 10403)	Closes the TDBXConnection (see page 10398)
A	Commit (see page 10403)	Commits work for a transaction
	Create (see page 10400)	Creates an instance of TDBXConnection (see page 10398)
A	CreateAndBeginTransaction (see page 10404)	This is CreateAndBeginTransaction, a member of class TDBXConnection.
A	DerivedCreateCommand (see page 10405)	This is DerivedCreateCommand, a member of class TDBXConnection.
A	DerivedGetCommands (see page 10405)	This is DerivedGetCommands, a member of class TDBXConnection.
A	DerivedGetCommandTypes (see page 10405)	This is DerivedGetCommandTypes, a member of class TDBXConnection.
A	DerivedOpen (see page 10405)	This is DerivedOpen, a member of class TDBXConnection.
V	GetConnectionProperties (see page 10406)	This is GetConnectionProperties, a member of class TDBXConnection.
V	GetDatabaseMetaData (see page 10406)	This is GetDatabaseMetaData, a member of class TDBXConnection.
V	GetErrorEvent (see page 10407)	This is GetErrorEvent, a member of class TDBXConnection.
V	GetIsOpen (see page 10407)	This is GetIsOpen, a member of class TDBXConnection.
V	GetTraceFlags (see page 10407)	This is GetTraceFlags, a member of class TDBXConnection.
V	GetTraceInfoEvent (see page 10407)	This is GetTraceInfoEvent, a member of class TDBXConnection.
V	Open (see page 10407)	Opens a connection
A	Rollback (see page 10408)	This is Rollback, a member of class TDBXConnection.
V	SetConnectionProperties (see page 10409)	This is SetConnectionProperties, a member of class TDBXConnection.
V	SetErrorEvent (see page 10409)	This is SetErrorEvent, a member of class TDBXConnection.
V	SetTraceFlags (see page 10409)	This is SetTraceFlags, a member of class TDBXConnection.
V	SetTraceInfoEvent (see page 10410)	This is SetTraceInfoEvent, a member of class TDBXConnection.

Public Events

	Name	Description
	OnErrorEvent (see page 10411)	Error handling event called before any exception is raised
	OnTrace (see page 10412)	Event handler for receiving trace events

Public Methods

	Name	Description
	BeginTransaction (see page 10402)	This is the overview for the BeginTransaction method overload.
	CommitFreeAndNil (see page 10404)	Commits work for a transaction
	CreateCommand (see page 10404)	Returns a TDBXCommand (see page 10378) instance to execute commands.
	Destroy (see page 10400)	Destroys a TDBXConnection (see page 10398) instance.
	GetCommands (see page 10406)	Gets list of known commands for command type.
	GetCommandTypes (see page 10406)	Gets list of supported command types for connection.
	RollbackFreeAndNil (see page 10408)	Rollbacks the work for a transaction
	RollbackIncompleteFreeAndNil (see page 10408)	Rollbacks a transaction without throwing an exception for invalid transaction.

Public Properties

	Name	Description
	ConnectionProperties (see page 10410)	Connection properties
	DatabaseMetaData (see page 10410)	Database metadata for connection
	IsOpen (see page 10411)	Indicates if connection open
	TraceFlags (see page 10411)	Trace flags to enable or disable tracing categories

Strict protected Fields

	Name	Description
	FIsolationLevel (see page 10401)	This is FIsolationLevel, a member of class TDBXConnection.

55.1.12.2 TDBXConnection.Create Constructor

Creates an instance of TDBXConnection (see page 10398)

Pascal

```
constructor Create(ConnectionBuilder: TDBXConnectionBuilder);
```

C++

```
__fastcall TDBXConnection(TDBXConnectionBuilder ConnectionBuilder);
```

Description

This constructor creates and initializes an instance of TDBXConnection (see page 10398).

See Also

Destroy (see page 10400)

55.1.12.3 TDBXConnection.Destroy Destructor

Destroys a TDBXConnection (see page 10398) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXConnection();
```

Description

This destructor destroys an instance of TDBXConnection (see page 10398).

See Also

Create (see page 10400)

55.1.12.4 TDBXConnection Fields

The fields of the TDBXConnection class are listed here.

Protected Fields

	Name	Description
	FConnectionProperties (see page 10401)	This is FConnectionProperties, a member of class TDBXConnection.
	FDBXContext (see page 10401)	This is FDBXContext, a member of class TDBXConnection.

Strict protected Fields

	Name	Description
	FIsolationLevel (see page 10401)	This is FIsolationLevel, a member of class TDBXConnection.

55.1.12.4.1 TDBXConnection.FConnectionProperties Field

Pascal

```
FConnectionProperties: TDBXProperties;
```

C++

```
TDBXProperties FConnectionProperties;
```

Description

This is FConnectionProperties, a member of class TDBXConnection.

55.1.12.4.2 TDBXConnection.FDBXContext Field

Pascal

```
FDBXContext: TDBXContext;
```

C++

```
TDBXContext FDBXContext;
```

Description

This is FDBXContext, a member of class TDBXConnection.

55.1.12.4.3 TDBXConnection.FIsolationLevel Field

Pascal

```
FIsolationLevel: TDBXIsolation;
```

C++

```
TDBXIsolation FIsolationLevel;
```

Description

This is FIsolationLevel, a member of class TDBXConnection.

55.1.12.5 TDBXConnection Methods

The methods of the TDBXConnection class are listed here.

Protected Methods

	Name	Description
V	Close (see page 10403)	Closes the TDBXConnection (see page 10398)
A	Commit (see page 10403)	Commits work for a transaction
A	CreateAndBeginTransaction (see page 10404)	This is CreateAndBeginTransaction, a member of class TDBXConnection.
A	DerivedCreateCommand (see page 10405)	This is DerivedCreateCommand, a member of class TDBXConnection.
A	DerivedGetCommands (see page 10405)	This is DerivedGetCommands, a member of class TDBXConnection.
A	DerivedGetCommandTypes (see page 10405)	This is DerivedGetCommandTypes, a member of class TDBXConnection.
A	DerivedOpen (see page 10405)	This is DerivedOpen, a member of class TDBXConnection.
V	GetConnectionProperties (see page 10406)	This is GetConnectionProperties, a member of class TDBXConnection.
V	GetDatabaseMetaData (see page 10406)	This is GetDatabaseMetaData, a member of class TDBXConnection.
V	GetErrorEvent (see page 10407)	This is GetErrorEvent, a member of class TDBXConnection.
V	GetIsOpen (see page 10407)	This is GetIsOpen, a member of class TDBXConnection.
V	GetTraceFlags (see page 10407)	This is GetTraceFlags, a member of class TDBXConnection.
V	GetTraceInfoEvent (see page 10407)	This is GetTraceInfoEvent, a member of class TDBXConnection.
V	Open (see page 10407)	Opens a connection
A	Rollback (see page 10408)	This is Rollback, a member of class TDBXConnection.
V	SetConnectionProperties (see page 10409)	This is SetConnectionProperties, a member of class TDBXConnection.
V	SetErrorEvent (see page 10409)	This is SetErrorEvent, a member of class TDBXConnection.
V	SetTraceFlags (see page 10409)	This is SetTraceFlags, a member of class TDBXConnection.
V	SetTraceInfoEvent (see page 10410)	This is SetTraceInfoEvent, a member of class TDBXConnection.

Public Methods

	Name	Description
V	BeginTransaction (see page 10402)	This is the overview for the BeginTransaction method overload.
V	CommitFreeAndNil (see page 10404)	Commits work for a transaction
V	CreateCommand (see page 10404)	Returns a TDBXCommand (see page 10378) instance to execute commands.
	GetCommands (see page 10406)	Gets list of known commands for command type.
	GetCommandTypes (see page 10406)	Gets list of supported command types for connection.
V	RollbackFreeAndNil (see page 10408)	Rollbacks the work for a transaction
V	RollbackIncompleteFreeAndNil (see page 10408)	Rollbacks a transaction without throwing an exception for invalid transaction.

55.1.12.5.1 BeginTransaction Method

This is the overview for the BeginTransaction method overload.

Overload List

	Name	Description
V	TDBXConnection.BeginTransaction () (see page 10402)	Returns a TDBXTransaction (see page 10623) instance
V	TDBXConnection.BeginTransaction (TDBXIsolation) (see page 10402)	Returns a TDBXTransaction (see page 10623) instance

55.1.12.5.1.1 TDBXConnection.BeginTransaction Method ()

Returns a TDBXTransaction (see page 10623) instance

Pascal

```
function BeginTransaction(Isolation: TDBXIsolation): TDBXTransaction; virtual; overload;
function BeginTransaction: TDBXTransaction; virtual; overload;
```

C++

```
virtual __fastcall TDBXTransaction BeginTransaction(TDBXIsolation Isolation);
virtual __fastcall TDBXTransaction BeginTransaction();
```

Description

This method gest a TDBXTransaction (see page 10623) instance for a new transaction.

Note: When the work for the transaction is complete, call either CommitFreeAndNil (see page 10404), RollbackFreeAndNil (see page 10408), or RollbackIncompleteFreeAndNil (see page 10408).

See Also

TDBXTransaction (see page 10623)

CommitFreeAndNil (see page 10404)

RollbackFreeAndNil (see page 10408)

RollbackIncompleteFreeAndNil (see page 10408)

55.1.12.5.2 TDBXConnection.Close Method

Closes the TDBXConnection (see page 10398)

Pascal

```
procedure Close; virtual;
```

C++

```
virtual __fastcall Close();
```

Description

This method closes the TDBXConnection (see page 10398), making it unavailable.

See Also

Open (see page 10407)

55.1.12.5.3 TDBXConnection.Commit Method

Commits work for a transaction

Pascal

```
procedure Commit(Transaction: TDBXTransaction); virtual; abstract;
```

C++

```
virtual __fastcall Commit(TDBXTransaction Transaction) = 0;
```

Description

This method commits the work for the specified TDBXTransaction (see page 10623).

Note: This method raises an exception if Transaction

is not an active transaction.

See Also

TDBXTransaction (see page 10623)

55.1.12.5.4 TDBXConnection.CommitFreeAndNil Method

Commits work for a transaction

Pascal

```
procedure CommitFreeAndNil(var Transaction: TDBXTransaction); virtual;
```

C++

```
virtual __fastcall CommitFreeAndNil(TDBXTransaction Transaction);
```

Description

This method commits the work for the specified TDBXTransaction (see page 10623). Any active transactions that were begun after this transaction are freed and effectively committed as well. The variable **Transaction** instance is set to `nil`.

Note: This method raises an exception if Transaction

is not an active transaction.

See Also

TDBXTransaction (see page 10623)

55.1.12.5.5 TDBXConnection.CreateAndBeginTransaction Method

Pascal

```
function CreateAndBeginTransaction(Isolation: TDBXIsolation): TDBXTransaction; virtual;
abstract;
```

C++

```
virtual __fastcall TDBXTransaction CreateAndBeginTransaction(TDBXIsolation Isolation) = 0;
```

Description

This is CreateAndBeginTransaction, a member of class TDBXConnection.

55.1.12.5.6 TDBXConnection.CreateCommand Method

Returns a TDBXCommand (see page 10378) instance to execute commands.

Pascal

```
function CreateCommand: TDBXCommand; virtual; overload;
```

C++

```
virtual __fastcall TDBXCommand CreateCommand();
```

Description

This method returns a TDBXCommand (see page 10378) instance to execute commands. The TDBXCommand.CommandType property specifies the type of command. Command types are specified in the TDBXCommandTypes (see page 10396) class. The default CommandType property setting is TDBXCommandTypes.SQLStatement., an SQL command.

Note: Call TDBXCommand.Free when the TDBXCommand (see page 10378) is no longer needed.

See Also

Connecting to a Database using the dbExpress Driver Framework

TDBXCommand (see page 10378)

TDBXCommandTypes (see page 10396)

CommandType (see page 10388)

Free

55.1.12.5.7 TDBXConnection.DerivedCreateCommand Method

Pascal

```
function DerivedCreateCommand: TDBXCommand; virtual; abstract;
```

C++

```
virtual __fastcall TDBXCommand DerivedCreateCommand() = 0;
```

Description

This is DerivedCreateCommand, a member of class TDBXConnection.

55.1.12.5.8 TDBXConnection.DerivedGetCommands Method

Pascal

```
procedure DerivedGetCommands(CommandType: WideString; List: TWideStrings); virtual; abstract;
```

C++

```
virtual __fastcall DerivedGetCommands(BSTR CommandType, TWideStrings List) = 0;
```

Description

This is DerivedGetCommands, a member of class TDBXConnection.

55.1.12.5.9 TDBXConnection.DerivedGetCommandTypes Method

Pascal

```
procedure DerivedGetCommandTypes(List: TWideStrings); virtual; abstract;
```

C++

```
virtual __fastcall DerivedGetCommandTypes(TWideStrings List) = 0;
```

Description

This is DerivedGetCommandTypes, a member of class TDBXConnection.

55.1.12.5.10 TDBXConnection.DerivedOpen Method

Pascal

```
procedure DerivedOpen; virtual; abstract;
```

C++

```
virtual __fastcall DerivedOpen() = 0;
```

Description

This is DerivedOpen, a member of class TDBXConnection.

55.1.12.5.11 TDBXConnection.GetCommands Method

Gets list of known commands for command type.

Pascal

```
procedure GetCommands(const CommandType: WideString; List: TWideStrings); overload;
```

C++

```
__fastcall GetCommands(const BSTR CommandType, TWideStrings List);
```

Description

This method populates a list with the known commands for **CommandType**.

55.1.12.5.12 TDBXConnection.GetCommandTypes Method

Gets list of supported command types for connection.

Pascal

```
procedure GetCommandTypes(List: TWideStrings); overload;
```

C++

```
__fastcall GetCommandTypes(TWideStrings List);
```

Description

This method populates a list with the supported command types for this connection. TDBXCommandTypes (see page 10396) provides constant names for the most common command types.

See Also

TDBXCommandTypes (see page 10396)

55.1.12.5.13 TDBXConnection.GetConnectionProperties Method

Pascal

```
function GetConnectionProperties: TDBXProperties; virtual;
```

C++

```
virtual __fastcall TDBXProperties GetConnectionProperties();
```

Description

This is GetConnectionProperties, a member of class TDBXConnection.

55.1.12.5.14 TDBXConnection.GetDatabaseMetaData Method

Pascal

```
function GetDatabaseMetaData: TDBXDatabaseMetaData; virtual;
```

C++

```
virtual __fastcall TDBXDatabaseMetaData GetDatabaseMetaData();
```

Description

This is GetDatabaseMetaData, a member of class TDBXConnection.

55.1.12.5.15 TDBXConnection.GetErrorEvent Method

Pascal

```
function GetErrorEvent: TDBXErrorEvent; virtual;
```

C++

```
virtual __fastcall TDBXErrorEvent GetErrorEvent();
```

Description

This is GetErrorEvent, a member of class TDBXConnection.

55.1.12.5.16 TDBXConnection.GetIsOpen Method

Pascal

```
function GetIsOpen: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetIsOpen();
```

Description

This is GetIsOpen, a member of class TDBXConnection.

55.1.12.5.17 TDBXConnection.GetTraceFlags Method

Pascal

```
function GetTraceFlags: TDBXTraceFlag; virtual;
```

C++

```
virtual __fastcall TDBXTraceFlag GetTraceFlags();
```

Description

This is GetTraceFlags, a member of class TDBXConnection.

55.1.12.5.18 TDBXConnection.GetTraceInfoEvent Method

Pascal

```
function GetTraceInfoEvent: TDBXTraceEvent; virtual;
```

C++

```
virtual __fastcall TDBXTraceEvent GetTraceInfoEvent();
```

Description

This is GetTraceInfoEvent, a member of class TDBXConnection.

55.1.12.5.19 TDBXConnection.Open Method

Opens a connection

Pascal

```
procedure Open; virtual;
```

C++

```
virtual __fastcall Open();
```

Description

This method opens a connection. A TDBXConnection (see page 10398) object provides support for creating commands and managing transactions.

Note: Attempting to use a TDBXConnection (see page 10398) instance before calling Open raises an exception

See Also

Connecting to a Database using the dbExpress Driver Framework

55.1.12.5.20 TDBXConnection.Rollback Method

Pascal

```
procedure Rollback(Transaction: TDBXTransaction); virtual; abstract;
```

C++

```
virtual __fastcall Rollback(TDBXTransaction Transaction) = 0;
```

Description

This is Rollback, a member of class TDBXConnection.

55.1.12.5.21 TDBXConnection.RollbackFreeAndNil Method

Rollbacks the work for a transaction

Pascal

```
procedure RollbackFreeAndNil(var Transaction: TDBXTransaction); virtual;
```

C++

```
virtual __fastcall RollbackFreeAndNil(TDBXTransaction Transaction);
```

Description

This method rollbacks the work for the TDBXTransaction (see page 10623). Any active transactions that were begun after this transaction are freed and effectively rolled back as well. The variable **Transaction** instance is set to `nil`.

Note: RollbackFreeAndNil raises an exception if Transaction

is not an active transaction.

See Also

TDBXTransaction (see page 10623)

55.1.12.5.22 TDBXConnection.RollbackIncompleteFreeAndNil Method

Rollbacks a transaction without throwing an exception for invalid transaction.

Pascal

```
procedure RollbackIncompleteFreeAndNil(var Transaction: TDBXTransaction); virtual;
```

C++

```
virtual __fastcall RollbackIncompleteFreeAndNil(TDBXTransaction Transaction);
```

Description

This method rolls back the work for the TDBXTransaction (see page 10623). Any active transactions that were begun after this transaction are freed and effectively rolled back as well. The variable **Transaction** instance is set to `nil`.

Unlike the RollbackFreeAndNil (see page 10408) procedure, this procedure does not throw an exception if **Transaction** is invalid or set to `nil`. This method is ideally used in a `finally` (or `except`) block. The `try` block uses either CommitFreeAndNil (see page 10404) or RollbackFreeAndNil (see page 10408), and the **Transaction** has already been completed, freed, and set to `nil`. If an exception is not raised in the `try` block, a call to RollbackIncompleteFreeAndNil in the `finally` block does nothing, because the **Transaction** instance is no longer valid. If an exception is raised, a call to RollbackIncompleteFreeAndNil in the `finally` block rolls back, frees, and sets the **Transaction** instance to `nil`.

See Also

TDBXTransaction (see page 10623)

CommitFreeAndNil

RollbackFreeAndNil

55.1.12.5.23 TDBXConnection.SetConnectionProperties Method

Pascal

```
procedure SetConnectionProperties(const Value: TDBXProperties); virtual;
```

C++

```
virtual __fastcall SetConnectionProperties(const TDBXProperties Value);
```

Description

This is SetConnectionProperties, a member of class TDBXConnection.

55.1.12.5.24 TDBXConnection.SetErrorEvent Method

Pascal

```
procedure SetErrorEvent(ErrorEvent: TDBXErrorEvent); virtual;
```

C++

```
virtual __fastcall SetErrorEvent(TDBXErrorEvent ErrorEvent);
```

Description

This is SetErrorEvent, a member of class TDBXConnection.

55.1.12.5.25 TDBXConnection.SetTraceFlags Method

Pascal

```
procedure SetTraceFlags(TraceFlags: TDBXTraceFlag); virtual;
```

C++

```
virtual __fastcall SetTraceFlags(TDBXTraceFlag TraceFlags);
```

Description

This is SetTraceFlags, a member of class TDBXConnection.

55.1.12.5.26 TDBXConnection.SetTraceInfoEvent Method

Pascal

```
procedure SetTraceInfoEvent(TraceInfoEvent: TDBXTraceEvent); virtual;
```

C++

```
virtual __fastcall SetTraceInfoEvent(TDBXTraceEvent TraceInfoEvent);
```

Description

This is SetTraceInfoEvent, a member of class TDBXConnection.

55.1.12.6 TDBXConnection Properties

The properties of the TDBXConnection class are listed here.

Public Properties

	Name	Description
	ConnectionProperties (see page 10410)	Connection properties
R	DatabaseMetaData (see page 10410)	Database metadata for connection
R	IsOpen (see page 10411)	Indicates if connection open
	TraceFlags (see page 10411)	Trace flags to enable or disable tracing categories

55.1.12.6.1 TDBXConnection.ConnectionProperties Property

Connection properties

Pascal

```
property ConnectionProperties: TDBXProperties;
```

C++

```
__property TDBXProperties ConnectionProperties;
```

Description

This property contains a TDBXProperties (see page 10559) instance specifying connection properties used to create this connection.

See Also

TDBXProperties (see page 10559)

55.1.12.6.2 TDBXConnection.DatabaseMetaData Property

Database metadata for connection

Pascal

```
property DatabaseMetaData: TDBXDatabaseMetaData;
```

C++

```
__property TDBXDatabaseMetaData DatabaseMetaData;
```

Description

This property contains a TDBXDatabaseMetaData (see page 10440) instance for this connection.

See Also

TDBXDatabaseMetaData (see page 10440)

55.1.12.6.3 TDBXConnection.IsOpen Property

Indicates if connection open

Pascal

```
property IsOpen: Boolean;
```

C++

```
__property Boolean IsOpen;
```

Description

This property is true if the connection is open and false otherwise.

55.1.12.6.4 TDBXConnection.TraceFlags Property

Trace flags to enable or disable tracing categories

Pascal

```
property TraceFlags: TDBXTraceFlag;
```

C++

```
__property TDBXTraceFlag TraceFlags;
```

Description

This property contains trace flags to enable or disable tracing categories. This value is a logical OR of the trace flags in the TDBXTraceFlags (see page 10617) class.

See Also

TDBXTraceFlags (see page 10617)

55.1.12.7 TDBXConnection Events

The events of the TDBXConnection class are listed here.

Public Events

	Name	Description
	OnErrorEvent (see page 10411)	Error handling event called before any exception is raised
	OnTrace (see page 10412)	Event handler for receiving trace events

55.1.12.7.1 TDBXConnection.OnErrorEvent Event

Error handling event called before any exception is raised

Pascal

```
property OnErrorEvent: TDBXErrorEvent;
```

C++

```
__property TDBXErrorEvent OnErrorEvent;
```

Description

This property contains an error handling event TDBXErrorEvent (see page 10671) that is called just before any dbExpress TDBXError (see page 10486) exception is raised.

See Also

TDBXError (see page 10486)

TDBXErrorEvent (see page 10671)

55.1.12.7.2 TDBXConnection.OnTrace Event

Event handler for receiving trace events

Pascal

```
property OnTrace: TDBXTraceEvent;
```

C++

```
__property TDBXTraceEvent OnTrace;
```

Description

This property contains an event handler TDBXTraceEvent (see page 10672) used for receiving trace events.

See Also

TDBXTraceEvent (see page 10672)

55.1.13 TDBXConnectionBuilder Class

Utility class used by driver writers when connections are created.

Class Hierarchy

Pascal

```
TDBXConnectionBuilder = class;
```

C++

```
class TDBXConnectionBuilder;
```

File

DBXCommon

Description

This utility class is used by driver writers when connections are created. It is particularly important for chaining delegate connections in front of a database connection.

55.1.13.1 TDBXConnectionBuilder Members

The following tables list the members exposed by TDBXConnectionBuilder.

Protected Methods

	Name	Description
	Create (see page 10413)	This is the overview for the Create constructor overload.

Public Methods

	Name	Description
	CreateDelegateeConnection (see page 10414)	Request creation of the connection the delegate connection delegates to.
V	Destroy (see page 10414)	Destroys an instance of TDBXConnectionBuilder (see page 10412).

Public Properties

	Name	Description
R	ConnectionProperties (see page 10415)	TDBXConnection (see page 10398) properties of current delegate or database connection being created.
R	DbxContext (see page 10415)	TDBXContext (see page 10432) used for the database connection.
R	DelegationSignature (see page 10415)	List of the delegate drivers and database driver used to create a connection.
R	InputConnectionName (see page 10416)	Connection name when connection first requested.
R	InputConnectionProperties (see page 10416)	TDBXProperties (see page 10559) when connection first requested.
R	InputPassword (see page 10416)	Database password when connection first requested.
R	InputUserName (see page 10417)	User name when connection first requested.

55.1.13.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TDBXConnectionBuilder.Create () (see page 10413)	Creates an instance of TDBXConnectionBuilder (see page 10412)
	TDBXConnectionBuilder.Create (TDBXConnectionBuilder) (see page 10413)	Creates an instance of TDBXConnectionBuilder (see page 10412)

55.1.13.2.1 TDBXConnectionBuilder.Create Constructor ()

Creates an instance of TDBXConnectionBuilder (see page 10412)

Pascal

```
constructor Create; overload;
constructor Create(Source: TDBXConnectionBuilder); overload;
```

C++

```
__fastcall TDBXConnectionBuilder();
__fastcall TDBXConnectionBuilder(TDBXConnectionBuilder Source);
```

Description

This constructor creates and initializes an instance of TDBXConnectionBuilder (see page 10412).

See Also

Destroy (see page 10414)

55.1.13.3 TDBXConnectionBuilder.Destroy Destructor

Destroys an instance of TDBXConnectionBuilder (see page 10412).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXConnectionBuilder();
```

Description

This destructor destroys an instance of TDBXConnectionBuilder (see page 10412).

55.1.13.4 TDBXConnectionBuilder Methods

The methods of the TDBXConnectionBuilder class are listed here.

Public Methods

	Name	Description
	CreateDelegateeConnection (see page 10414)	Request creation of the connection the delegate connection delegates to.

55.1.13.4.1 TDBXConnectionBuilder.CreateDelegateeConnection Method

Request creation of the connection the delegate connection delegates to.

Pascal

```
function CreateDelegateeConnection: TDBXConnection;
```

C++

```
__fastcall TDBXConnection CreateDelegateeConnection();
```

Description

Delegate driver implementations use this to request the creation of the connection the delegate connection delegates to.

55.1.13.5 TDBXConnectionBuilder Properties

The properties of the TDBXConnectionBuilder class are listed here.

Public Properties

	Name	Description
R	ConnectionProperties (see page 10415)	TDBXConnection (see page 10398) properties of current delegate or database connection being created.
R	DbxContext (see page 10415)	TDBXContext (see page 10432) used for the database connection.
R	DelegationSignature (see page 10415)	List of the delegate drivers and database driver used to create a connection.
R	InputConnectionName (see page 10416)	Connection name when connection first requested.
R	InputConnectionProperties (see page 10416)	TDBXProperties (see page 10559) when connection first requested.
R	InputPassword (see page 10416)	Database password when connection first requested.
R	InputUserName (see page 10417)	User name when connection first requested.

55.1.13.5.1 TDBXConnectionBuilder.ConnectionProperties Property

TDBXConnection (see page 10398) properties of current delegate or database connection being created.

Pascal

```
property ConnectionProperties: TDBXProperties;
```

C++

```
__property TDBXProperties ConnectionProperties;
```

Description

This property contains a TDBXProperties (see page 10559) instance describing the TDBXConnection (see page 10398) properties provided by the current delegate or database connection being created.

See Also

TDBXConnection (see page 10398)

TDBXProperties (see page 10559)

55.1.13.5.2 TDBXConnectionBuilder.DbxContext Property

TDBXContext (see page 10432) used for the database connection.

Pascal

```
property DbxContext: TDBXContext;
```

C++

```
__property TDBXContext DbxContext;
```

Description

This TDBXContext (see page 10432) instance is used for the database connection.

Note: Delegate connections should not use this instance of TDBXContext (see page 10432). They should create their own instead.

See Also

TDBXContext (see page 10432)

55.1.13.5.3 TDBXConnectionBuilder.DelegationSignature Property

List of the delegate drivers and database driver used to create a connection.

Pascal

```
property DelegationSignature: WideString;
```

C++

```
__property BSTR DelegationSignature;
```

Description

This property holds a slash ('/') separated list of the delegate drivers and the database driver used to create a connection by this TDBXConnectionBuilder (see page 10412) instance.

55.1.13.5.4 TDBXConnectionBuilder.InputConnectionName Property

Connection name when connection first requested.

Pascal

```
property InputConnectionName: WideString;
```

C++

```
__property BSTR InputConnectionName;
```

Description

This property contains the database connection name from when a connection was first requested from a TDBXConnectionFactory (see page 10422).

See Also

InputConnectionProperties

InputPassword

InputUserName

55.1.13.5.5 TDBXConnectionBuilder.InputConnectionProperties Property

TDBXProperties (see page 10559) when connection first requested.

Pascal

```
property InputConnectionProperties: TDBXProperties;
```

C++

```
__property TDBXProperties InputConnectionProperties;
```

Description

This property contains a database TDBXProperties (see page 10559) instance from when a connection was first requested from a TDBXConnectionFactory (see page 10422).

See Also

TDBXProperties (see page 10559)

InputConnectionName

InputPassword

InputUserName

55.1.13.5.6 TDBXConnectionBuilder.InputPassword Property

Database password when connection first requested.

Pascal

```
property InputPassword: WideString;
```

C++

```
__property BSTR InputPassword;
```

Description

This property contains the database password from when a connection was first requested from a TDBXConnectionFactory (

see page 10422).

See Also

InputConnectionName

InputConnectionProperties

InputUserName

55.1.13.5.7 TDBXConnectionBuilder.InputUserName Property

User name when connection first requested.

Pascal

```
property InputUserName: WideString;
```

C++

```
__property BSTR InputUserName;
```

Description

This property contains the database user name when a connection was first requested from a TDBXConnectionFactory (see page 10422).

See Also

InputConnectionName

InputConnectionProperties

InputPassword

55.1.14 TDBXConnectionEx Class

Temporary extension of TDBXConnection (see page 10398) class.

Class Hierarchy

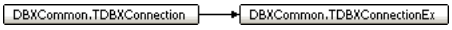

Pascal

```
TDBXConnectionEx = class(TDBXConnection);
```

C++

```
class TDBXConnectionEx : public TDBXConnection;
```

File

DBXCommon

Description

TDBXConnectionEx is an extension of TDBXConnection (see page 10398).

Note: Classes (see page 2685) ending in “Ex” are temporary classes that exist to avoid breaking interfaces. Their contents will be migrated to their base class in the next release.

See Also

TDBXConnection (see page 10398)

55.1.14.1 TDBXConnectionEx Members

The following tables list the members exposed by TDBXConnectionEx.

Protected Fields

	Name	Description
	FConnectionProperties (see page 10401)	This is FConnectionProperties, a member of class TDBXConnection.
	FDBXContext (see page 10401)	This is FDBXContext, a member of class TDBXConnection.

Protected Methods

	Name	Description
V	Close (see page 10403)	Closes the TDBXConnection (see page 10398)
A	Commit (see page 10403)	Commits work for a transaction
	Create (see page 10400)	Creates an instance of TDBXConnection (see page 10398)
A	CreateAndBeginTransaction (see page 10404)	This is CreateAndBeginTransaction, a member of class TDBXConnection.
V	CreateMorphCommand (see page 10419)	This is CreateMorphCommand, a member of class TDBXConnectionEx.
A	DerivedCreateCommand (see page 10405)	This is DerivedCreateCommand, a member of class TDBXConnection.
A	DerivedGetCommands (see page 10405)	This is DerivedGetCommands, a member of class TDBXConnection.
A	DerivedGetCommandTypes (see page 10405)	This is DerivedGetCommandTypes, a member of class TDBXConnection.
A	DerivedOpen (see page 10405)	This is DerivedOpen, a member of class TDBXConnection.
V	GetConnectionProperties (see page 10406)	This is GetConnectionProperties, a member of class TDBXConnection.
V	GetConnectionProperty (see page 10420)	This is GetConnectionProperty, a member of class TDBXConnectionEx.
V	GetDatabaseMetaData (see page 10406)	This is GetDatabaseMetaData, a member of class TDBXConnection.
V	GetErrorEvent (see page 10407)	This is GetErrorEvent, a member of class TDBXConnection.
V	GetIsOpen (see page 10407)	This is GetIsOpen, a member of class TDBXConnection.
V	GetProductName (see page 10420)	This is GetProductName, a member of class TDBXConnectionEx.
V	GetProductVersion (see page 10420)	This is GetProductVersion, a member of class TDBXConnectionEx.
V	GetTraceFlags (see page 10407)	This is GetTraceFlags, a member of class TDBXConnection.
V	GetTraceInfoEvent (see page 10407)	This is GetTraceInfoEvent, a member of class TDBXConnection.
V	Open (see page 10407)	Opens a connection
A	Rollback (see page 10408)	This is Rollback, a member of class TDBXConnection.
V	SetConnectionProperties (see page 10409)	This is SetConnectionProperties, a member of class TDBXConnection.
V	SetErrorEvent (see page 10409)	This is SetErrorEvent, a member of class TDBXConnection.
V	SetTraceFlags (see page 10409)	This is SetTraceFlags, a member of class TDBXConnection.
V	SetTraceInfoEvent (see page 10410)	This is SetTraceInfoEvent, a member of class TDBXConnection.

Public Events

	Name	Description
	OnErrorEvent (see page 10411)	Error handling event called before any exception is raised
	OnTrace (see page 10412)	Event handler for receiving trace events

Public Methods

	Name	Description
V	BeginTransaction (see page 10402)	This is the overview for the BeginTransaction method overload.
V	CommitFreeAndNil (see page 10404)	Commits work for a transaction
V	CreateCommand (see page 10404)	Returns a TDBXCommand (see page 10378) instance to execute commands.
V	Destroy (see page 10419)	Destroy a TDBXConnectionEx (see page 10417) instance
	GetCommands (see page 10406)	Gets list of known commands for command type.
	GetCommandTypes (see page 10406)	Gets list of supported command types for connection.
V	GetVendorProperty (see page 10420)	This is GetVendorProperty, a member of class TDBXConnectionEx.
V	RollbackFreeAndNil (see page 10408)	Rollbacks the work for a transaction
V	RollbackIncompleteFreeAndNil (see page 10408)	Rollbacks a transaction without throwing an exception for invalid transaction.

Public Properties

	Name	Description
	ConnectionProperties (see page 10410)	Connection properties
R	ConnectionProperty (see page 10421)	Connection properties.
R	DatabaseMetaData (see page 10410)	Database metadata for connection
R	IsOpen (see page 10411)	Indicates if connection open
	MetaDataReader (see page 10421)	This is MetaDataReader, a member of class TDBXConnectionEx.
R	ProductName (see page 10421)	Provide product name if supported.
R	ProductVersion (see page 10421)	Product version.
	TraceFlags (see page 10411)	Trace flags to enable or disable tracing categories

Strict protected Fields

	Name	Description
	FIsolationLevel (see page 10401)	This is FIsolationLevel, a member of class TDBXConnection.

55.1.14.2 TDBXConnectionEx.Destroy Destructor

Destroy a TDBXConnectionEx (see page 10417) instance

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXConnectionEx();
```

Description

This method destroys a TDBXConnectionEx (see page 10417) instance.

55.1.14.3 TDBXConnectionEx Methods

The methods of the TDBXConnectionEx class are listed here.

Protected Methods

	Name	Description
V	CreateMorphCommand (see page 10419)	This is CreateMorphCommand, a member of class TDBXConnectionEx.
V	GetConnectionProperty (see page 10420)	This is GetConnectionProperty, a member of class TDBXConnectionEx.
V	GetProductName (see page 10420)	This is GetProductName, a member of class TDBXConnectionEx.
V	GetProductVersion (see page 10420)	This is GetProductVersion, a member of class TDBXConnectionEx.

Public Methods

	Name	Description
V	GetVendorProperty (see page 10420)	This is GetVendorProperty, a member of class TDBXConnectionEx.

55.1.14.3.1 TDBXConnectionEx.CreateMorphCommand Method

Pascal

```
function CreateMorphCommand(MorphicCommand: TDBXCommand): TDBXCommand; virtual;
```

C++

```
virtual __fastcall TDBXCommand CreateMorphCommand(TDBXCommand MorphicCommand);
```

Description

This is CreateMorphCommand, a member of class TDBXConnectionEx.

55.1.14.3.2 TDBXConnectionEx.GetConnectionProperty Method

Pascal

```
function GetConnectionProperty(const Name: WideString): WideString; virtual;
```

C++

```
virtual __fastcall BSTR GetConnectionProperty(const BSTR Name);
```

Description

This is GetConnectionProperty, a member of class TDBXConnectionEx.

55.1.14.3.3 TDBXConnectionEx.GetProductName Method

Pascal

```
function GetProductName: WideString; virtual;
```

C++

```
virtual __fastcall BSTR GetProductName();
```

Description

This is GetProductName, a member of class TDBXConnectionEx.

55.1.14.3.4 TDBXConnectionEx.GetProductVersion Method

Pascal

```
function GetProductVersion: WideString; virtual;
```

C++

```
virtual __fastcall BSTR GetProductVersion();
```

Description

This is GetProductVersion, a member of class TDBXConnectionEx.

55.1.14.3.5 TDBXConnectionEx.GetVendorProperty Method

Pascal

```
function GetVendorProperty(const Name: WideString): WideString; virtual;
```

C++

```
virtual __fastcall BSTR GetVendorProperty(const BSTR Name);
```

Description

This is GetVendorProperty, a member of class TDBXConnectionEx.

55.1.14.4 TDBXConnectionEx Properties

The properties of the TDBXConnectionEx class are listed here.

Public Properties

	Name	Description
R	ConnectionProperty (see page 10421)	Connection properties.
	MetaDataReader (see page 10421)	This is MetaDataReader, a member of class TDBXConnectionEx.
R	ProductName (see page 10421)	Provide product name if supported.
R	ProductVersion (see page 10421)	Product version.

55.1.14.4.1 TDBXConnectionEx.ConnectionProperty Property

Connection properties.

Pascal

```
property ConnectionProperty [const Name: WideString]: WideString;
```

C++

```
__property BSTR ConnectionProperty[BSTR const Name];
```

Description

ConnectionProperty holds connection properties. It can be indexed by property name.

55.1.14.4.2 TDBXConnectionEx.MetaDataReader Property

Pascal

```
property MetaDataReader: TObject;
```

C++

```
__property TObject * MetaDataReader;
```

Description

This is MetaDataReader, a member of class TDBXConnectionEx.

55.1.14.4.3 TDBXConnectionEx.ProductName Property

Provide product name if supported.

Pascal

```
property ProductName: WideString;
```

C++

```
__property BSTR ProductName;
```

Description

Provide product name if supported. The default implementation tries to use the TDBXPropertyNamesEx.ProductName driver first. It calls **GetVendorProperty (see page 10420)** second. The TDBXPropertyNames.DriverName is used as a last resort.

See Also

TDBXPropertyNames (see page 10566)

TDBXPropertyNamesEx (see page 10574)

55.1.14.4.4 TDBXConnectionEx.ProductVersion Property

Product version.

Pascal

```
property ProductVersion: WideString;
```

C++

```
__property BSTR ProductVersion;
```

Description

Returns empty string if not implemented. Otherwise a string is returned with the format nn.nn.nnnn, where the first two digits are the major version, the second two digits are the minor version, and the last four digits are the release version. This value can be overridden by setting the TDBXPropertyNamesEx.ProductVersion connection property. Note that TDBXPropertyNamesEx.ProductVersion is the name of the property, not the property itself.

See Also

TDBXPropertyNamesEx (see page 10574)

55.1.15 TDBXConnectionFactory Class

Abstract base class used for creating new instances of TDBXConnection (see page 10398)

Class Hierarchy

Pascal

```
TDBXConnectionFactory = class;
```

C++

```
class TDBXConnectionFactory;
```

File

DBXCommon

Description

This abstract base class can be used for creating new instances of TDBXConnection (see page 10398).

TDBXConnection.GetConnectionFactory (see page 10427) is a class static method that returns a TDBXConnectionFactory singleton. This singleton instance defaults to the TDBXIniFileConnectionFactory (see page 10501) class. The dbxconnections.ini and dbxdrivers.ini files are loaded from the file names specified under the HKEY_CURRENT_USER\TDBXRegistryKey (see page 10684) windows registry setting. If this registry key is not set, the ini files are loaded from the location of the application's executable.

A TDBXConnectionFactory manages a collection of named driver and connection property sets. A separate TDBXProperties (see page 10559) object is used for each driver and connection property set. The driver properties contain enough information to load and instantiate a driver implementation. The connection properties include driver properties and contain enough information to instantiate a connection.

See Also

Connecting to a Database using the dbExpress Driver Framework

TDBXConnection (see page 10398)

TDBXIniFileConnectionFactory (see page 10501)

TDBXProperties (see page 10559)

GetConnectionFactory (see page 10427)

55.1.15.1 TDBXConnectionFactory Members

The following tables list the members exposed by TDBXConnectionFactory.

Protected Methods

	Name	Description
	DerivedClose (see page 10426)	This is DerivedClose, a member of class TDBXConnectionFactory.
	DerivedOpen (see page 10426)	This is DerivedOpen, a member of class TDBXConnectionFactory.
	GetErrorEvent (see page 10429)	This is GetErrorEvent, a member of class TDBXConnectionFactory.
	GetTraceFlags (see page 10429)	This is GetTraceFlags, a member of class TDBXConnectionFactory.
	GetTraceInfoEvent (see page 10429)	This is GetTraceInfoEvent, a member of class TDBXConnectionFactory.
	SetErrorEvent (see page 10430)	This is SetErrorEvent, a member of class TDBXConnectionFactory.
	SetTraceFlags (see page 10430)	This is SetTraceFlags, a member of class TDBXConnectionFactory.
	SetTraceInfoEvent (see page 10431)	This is SetTraceInfoEvent, a member of class TDBXConnectionFactory.

Public Events

	Name	Description
	OnError (see page 10432)	TDBXErrorEvent (see page 10671) handler.
	OnTrace (see page 10432)	TDBXTraceInfoEvent handler.

Public Methods

	Name	Description
	Close (see page 10425)	Closes the factory.
	Create (see page 10423)	Creates an instance of TDBXConnectionFactory (see page 10422)
	Destroy (see page 10424)	Destroys a TDBXConnectionFactory (see page 10422)
	GetConnection (see page 10426)	This is the overview for the GetConnection method overload.
	GetConnectionDriver (see page 10427)	Gets a TDBXDriver (see page 10472) instance for a connection name
	GetConnectionFactory (see page 10427)	Gets a TDBXConnectionFactory (see page 10422) singleton
	GetConnectionItems (see page 10428)	Adds names of all connections to Items list.
	GetConnectionProperties (see page 10428)	Gets a TDBXPropertiesinstance for a connection name
	GetDriver (see page 10428)	Returns TDBXDriver (see page 10472) instance for a driver name
	GetDriverProperties (see page 10429)	Returns TDBXProperties (see page 10559) instance of driver properties
	Open (see page 10430)	Opens the TDBXConnectionFactory (see page 10422)
	SetConnectionFactory (see page 10430)	Change the default TDBXConnectionFactorysingleton

Public Properties

	Name	Description
	TraceFlags (see page 10431)	Trace flags to enable or disable tracing categories

Strict protected Fields

	Name	Description
	FDBXContext (see page 10424)	This is FDBXContext, a member of class TDBXConnectionFactory.

Strict protected Methods

	Name	Description
	CreateDelegatePaths (see page 10426)	This is CreateDelegatePaths, a member of class TDBXConnectionFactory.

55.1.15.2 TDBXConnectionFactory.Create Constructor

Creates an instance of TDBXConnectionFactory (see page 10422)

Pascal

```
constructor Create;
```

C++

```
__fastcall TDBXConnectionFactory();
```

Description

This constructor creates and initializes an instance of TDBXConnectionFactory (see page 10422). You can create a new TDBXConnection (see page 10398) object from this object that allows you to connect to a database. The dbxconnections.ini and dbxdrivers.ini files can specify all the information about the database connection for TDBXConnectionFactory (see page 10422) to create the connection.

See Also

Connecting to a Database using the dbExpress Driver Framework

TDBXConnection (see page 10398)

55.1.15.3 TDBXConnectionFactory.Destroy Destructor

Destroys a TDBXConnectionFactory (see page 10422)

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXConnectionFactory();
```

Description

This destructor destroys an instance of TDBXConnectionFactory (see page 10422).

55.1.15.4 TDBXConnectionFactory Fields

The fields of the TDBXConnectionFactory class are listed here.

Strict protected Fields

	Name	Description
	FDBXContext (see page 10424)	This is FDBXContext, a member of class TDBXConnectionFactory.

55.1.15.4.1 TDBXConnectionFactory.FDBXContext Field

Pascal

```
FDBXContext: TDBXContext;
```

C++

```
TDBXContext FDBXContext;
```

Description

This is FDBXContext, a member of class TDBXConnectionFactory.

55.1.15.5 TDBXConnectionFactory Methods

The methods of the TDBXConnectionFactory class are listed here.

Protected Methods

	Name	Description
	DerivedClose (see page 10426)	This is DerivedClose, a member of class TDBXConnectionFactory.
	DerivedOpen (see page 10426)	This is DerivedOpen, a member of class TDBXConnectionFactory.
	GetErrorEvent (see page 10429)	This is GetErrorEvent, a member of class TDBXConnectionFactory.
	GetTraceFlags (see page 10429)	This is GetTraceFlags, a member of class TDBXConnectionFactory.
	GetTraceInfoEvent (see page 10429)	This is GetTraceInfoEvent, a member of class TDBXConnectionFactory.
	SetErrorEvent (see page 10430)	This is SetErrorEvent, a member of class TDBXConnectionFactory.
	SetTraceFlags (see page 10430)	This is SetTraceFlags, a member of class TDBXConnectionFactory.
	SetTraceInfoEvent (see page 10431)	This is SetTraceInfoEvent, a member of class TDBXConnectionFactory.

Public Methods

	Name	Description
	Close (see page 10425)	Closes the factory.
	GetConnection (see page 10426)	This is the overview for the GetConnection method overload.
	GetConnectionDriver (see page 10427)	Gets a TDBXDriver (see page 10472) instance for a connection name
	GetConnectionFactory (see page 10427)	Gets a TDBXConnectionFactory (see page 10422) singleton
	GetConnectionItems (see page 10428)	Adds names of all connections to Items list.
	GetConnectionProperties (see page 10428)	Gets a TDBXPropertiesinstance for a connection name
	GetDriver (see page 10428)	Returns TDBXDriver (see page 10472) instance for a driver name
	GetDriverProperties (see page 10429)	Returns TDBXProperties (see page 10559) instance of driver properties
	Open (see page 10430)	Opens the TDBXConnectionFactory (see page 10422)
	SetConnectionFactory (see page 10430)	Change the default TDBXConnectionFactorysingleton

Strict protected Methods

	Name	Description
	CreateDelegatePaths (see page 10426)	This is CreateDelegatePaths, a member of class TDBXConnectionFactory.

55.1.15.5.1 TDBXConnectionFactory.Close Method

Closes the factory.

Pascal

```
procedure Close; virtual;
```

C++

```
virtual __fastcall Close();
```

Description

This method closes the TDBXConnectionFactory (see page 10422). A call to Close then Open (see page 10430) reloads the contents of a TDBXConnectionFactory (see page 10422).

See Also

Open (see page 10430)

55.1.15.5.2 TDBXConnectionFactory.CreateDelegatePaths Method

Pascal

```
procedure CreateDelegatePaths;
```

C++

```
__fastcall CreateDelegatePaths();
```

Description

This is CreateDelegatePaths, a member of class TDBXConnectionFactory.

55.1.15.5.3 TDBXConnectionFactory.DerivedClose Method

Pascal

```
procedure DerivedClose; virtual; abstract;
```

C++

```
virtual __fastcall DerivedClose() = 0;
```

Description

This is DerivedClose, a member of class TDBXConnectionFactory.

55.1.15.5.4 TDBXConnectionFactory.DerivedOpen Method

Pascal

```
procedure DerivedOpen; virtual; abstract;
```

C++

```
virtual __fastcall DerivedOpen() = 0;
```

Description

This is DerivedOpen, a member of class TDBXConnectionFactory.

55.1.15.5.5 GetConnection Method

This is the overview for the GetConnection method overload.

Overload List

	Name	Description
	TDBXConnectionFactory.GetConnection (TDBXProperties) (see page 10426)	Creates a TDBXConnection (see page 10398) instance for connecting to a database
	TDBXConnectionFactory.GetConnection (WideString, WideString, WideString) (see page 10426)	Creates a TDBXConnection (see page 10398) instance for connecting to a database

55.1.15.5.5.1 TDBXConnectionFactory.GetConnection Method (TDBXProperties)

Creates a TDBXConnection (see page 10398) instance for connecting to a database

Pascal

```
function GetConnection(const ConnectionName: WideString; const UserName: WideString; const
Password: WideString): TDBXConnection; overload;
function GetConnection(ConnectionProperties: TDBXProperties): TDBXConnection; overload;
```

C++

```
__fastcall TDBXConnection GetConnection(const BSTR ConnectionName, const BSTR UserName, const
BSTR Password);
__fastcall TDBXConnection GetConnection(TDBXProperties ConnectionProperties);
```

Description

This method creates a TDBXConnection (see page 10398) instance for connecting to a database.

If the **UserName** parameter is an empty string, then **ConnectionName**'s associated TDBXProperties (see page 10559) setting for TDBXPropertyNamesUserName is used.

If the **Password**parameter is an empty string, then **ConnectionName**'s associated TDBXProperties (see page 10559) setting for TDBXPropertyNamesPassword is used.

Note: Call TDBXDriver.Freewhen the TDBXConnection (see page 10398) is no longer needed.

See Also

Connecting to a Database using the dbExpress Driver Framework

TDBXConnection (see page 10398)

TDBXProperties (see page 10559)

TDBXPropertyNames (see page 10566)

UserName (see page 10574)

Password (see page 10573)

55.1.15.5.6 TDBXConnectionFactory.GetConnectionDriver Method

Gets a TDBXDriver (see page 10472) instance for a connection name

Pascal

```
function GetConnectionDriver(const ConnectionName: WideString): TDBXDelegateDriver; overload;
```

C++

```
__fastcall TDBXDelegateDriver GetConnectionDriver(const BSTR ConnectionName);
```

Description

This method gets the associated TDBXDriver (see page 10472) instance for the supplied **ConnectionName**.

Note: Call TDBXDriver.Free when the TDBXDriver (see page 10472) is no longer needed.

See Also

TDBXDriver (see page 10472)

55.1.15.5.7 TDBXConnectionFactory.GetConnectionFactory Method

Gets a TDBXConnectionFactory (see page 10422) singleton

Pascal

```
class function GetConnectionFactory: TDBXConnectionFactory; static;
```

C++

```
static __fastcall TDBXConnectionFactory GetConnectionFactory();
```

Description

This method returns a class static TDBXConnectionFactory (see page 10422) singleton that can be used to create new TDBXConnection (see page 10398) instances.

See Also

TDBXConnection (see page 10398)

TDBXConnectionFactory (see page 10422)

55.1.15.5.8 TDBXConnectionFactory.GetConnectionItems Method

Adds names of all connections to Items list.

Pascal

```
procedure GetConnectionItems(Items: TStrings);
```

C++

```
__fastcall GetConnectionItems(TStrings Items);
```

Description

This method adds the names of all connections in the TDBXConnectionFactory (see page 10422) to the Items list.

55.1.15.5.9 TDBXConnectionFactory.GetConnectionProperties Method

Gets a TDBXPropertiesinstance for a connection name

Pascal

```
function GetConnectionProperties(const ConnectionName: WideString): TDBXProperties;
```

C++

```
__fastcall TDBXProperties GetConnectionProperties(const BSTR ConnectionName);
```

Description

This method returns a TDBXProperties (see page 10559) instance describing the connection properties for the given **ConnectionName**.

See Also

TDBXProperties (see page 10559)

55.1.15.5.10 TDBXConnectionFactory.GetDriver Method

Returns TDBXDriver (see page 10472) instance for a driver name

Pascal

```
function GetDriver(const DriverName: WideString): TDBXDriver; overload;
```

C++

```
__fastcall TDBXDriver GetDriver(const BSTR DriverName);
```

Description

This method returns a TDBXDriver (see page 10472) instance for the **DriverName** parameter.

Note: Call TDBXDriver.Free when the returned value is no longer needed.

See Also

TDBXDriver (see page 10472)

55.1.15.5.11 TDBXConnectionFactory.GetDriverProperties Method

Returns TDBXProperties (see page 10559) instance of driver properties

Pascal

```
function GetDriverProperties(const DriverName: WideString): TDBXProperties;
```

C++

```
__fastcall TDBXProperties GetDriverProperties(const BSTR DriverName);
```

Description

This method returns a TDBXProperties (see page 10559) instance of driver properties for the **DriverName** parameter.

See Also

TDBXProperties (see page 10559)

55.1.15.5.12 TDBXConnectionFactory.GetErrorEvent Method

Pascal

```
function GetErrorEvent: TDBXErrorEvent; virtual;
```

C++

```
virtual __fastcall TDBXErrorEvent GetErrorEvent();
```

Description

This is GetErrorEvent, a member of class TDBXConnectionFactory.

55.1.15.5.13 TDBXConnectionFactory.GetTraceFlags Method

Pascal

```
function GetTraceFlags: TDBXTraceFlag; virtual;
```

C++

```
virtual __fastcall TDBXTraceFlag GetTraceFlags();
```

Description

This is GetTraceFlags, a member of class TDBXConnectionFactory.

55.1.15.5.14 TDBXConnectionFactory.GetTraceInfoEvent Method

Pascal

```
function GetTraceInfoEvent: TDBXTraceEvent; virtual;
```

C++

```
virtual __fastcall TDBXTraceEvent GetTraceInfoEvent();
```

Description

This is GetTraceInfoEvent, a member of class TDBXConnectionFactory.

55.1.15.5.15 TDBXConnectionFactory.Open Method

Opens the TDBXConnectionFactory (see page 10422)

Pascal

```
procedure Open; virtual;
```

C++

```
virtual __fastcall Open();
```

Description

This method opens the TDBXConnectionFactory (see page 10422) instance, making it ready for use.

55.1.15.5.16 TDBXConnectionFactory.SetConnectionFactory Method

Change the default TDBXConnectionFactorysingleton

Pascal

```
class procedure SetConnectionFactory(ConnectionFactory: TDBXConnectionFactory); static;
```

C++

```
static __fastcall SetConnectionFactory(TDBXConnectionFactory ConnectionFactory);
```

Description

This method changes the default TDBXConnectionFactory (see page 10422) singleton that is returned by the GetConnectionFactoryclass static method.

Note: This method can be called only once, when an application is initialized.

See Also

GetConnectionFactory (see page 10427)

55.1.15.5.17 TDBXConnectionFactory.SetErrorEvent Method

Pascal

```
procedure SetErrorEvent(ErrorEvent: TDBXErrorEvent); virtual;
```

C++

```
virtual __fastcall SetErrorEvent(TDBXErrorEvent ErrorEvent);
```

Description

This is SetErrorEvent, a member of class TDBXConnectionFactory.

55.1.15.5.18 TDBXConnectionFactory.SetTraceFlags Method

Pascal

```
procedure SetTraceFlags(TraceFlags: TDBXTraceFlag); virtual;
```

C++

```
virtual __fastcall SetTraceFlags(TDBXTraceFlag TraceFlags);
```

Description

This is SetTraceFlags, a member of class TDBXConnectionFactory.

55.1.15.5.19 TDBXConnectionFactory.SetTraceInfoEvent Method

Pascal

```
procedure SetTraceInfoEvent(TraceInfoEvent: TDBXTraceEvent); virtual;
```

C++

```
virtual __fastcall SetTraceInfoEvent(TDBXTraceEvent TraceInfoEvent);
```

Description

This is SetTraceInfoEvent, a member of class TDBXConnectionFactory.

55.1.15.6 TDBXConnectionFactory Properties

The properties of the TDBXConnectionFactory class are listed here.

Public Properties

	Name	Description
	TraceFlags (see page 10431)	Trace flags to enable or disable tracing categories

55.1.15.6.1 TDBXConnectionFactory.TraceFlags Property

Trace flags to enable or disable tracing categories

Pascal

```
property TraceFlags: TDBXTraceFlag;
```

C++

```
__property TDBXTraceFlag TraceFlags;
```

Description

This property contains the TDBXTraceFlags (see page 10617) for which tracing should be enabled. This value is a logical OR of the trace flags in the TDBXTraceFlags (see page 10617) class.

See Also

TDBXTraceFlags (see page 10617)

55.1.15.7 TDBXConnectionFactory Events

The events of the TDBXConnectionFactory class are listed here.

Public Events

	Name	Description
	OnError (see page 10432)	TDBXErrorEvent (see page 10671) handler.
	OnTrace (see page 10432)	TDBXTraceInfoEvent handler.

55.1.15.7.1 TDBXConnectionFactory.OnError Event

TDBXErrorEvent (see page 10671) handler.

Pascal

```
property OnError: TDBXErrorEvent;
```

C++

```
__property TDBXErrorEvent OnError;
```

Description

The TDBXErrorEvent (see page 10671) handler is called before a TDBXError (see page 10486) exception is raised. This TDBXErrorEvent (see page 10671) is propagated to all TDBXConnection (see page 10398) and TDBXDriver (see page 10472) instances created by this TDBXConnectionFactory (see page 10422).

See Also

TDBXConnection (see page 10398)

TDBXDriver (see page 10472)

TDBXError (see page 10486)

TDBXErrorEvent (see page 10671)

55.1.15.7.2 TDBXConnectionFactory.OnTrace Event

TDBXTraceInfoEvent handler.

Pascal

```
property OnTrace: TDBXTraceEvent;
```

C++

```
__property TDBXTraceEvent OnTrace;
```

Description

The TDBXTraceInfoEvent handler is called for driver trace events. This TDBXTraceInfoEvent is propagated to all TDBXConnection (see page 10398) and TDBXDriver (see page 10472) instances created by this TDBXConnectionFactory (see page 10422).

See Also

TDBXConnection (see page 10398)

TDBXDriver (see page 10472)

TDBXTraceInfoEvent

55.1.16 TDBXContext Class

Context information for a connection.

Class Hierarchy

Pascal

```
TDBXContext = class;
```

C++

```
class TDBXContext;
```

File

DBXCommon

Description

This class holds context information mostly of use to dbExpress driver writers. Access to a TDBXContext instance should not be provided to applications using the dbExpress driver framework. A TDBXContext instance contains members that need to be shared across a single TDBXConnection (see page 10398) and all of its related objects, such as a TDBXCommand (see page 10378) or a TDBXReader (see page 10576). All objects created directly or indirectly from a TDBXConnection (see page 10398) use the same instance of TDBXContext as the TDBXConnection (see page 10398) that created them.

See Also

TDBXCommand (see page 10378)

TDBXConnection (see page 10398)

TDBXReader (see page 10576)

55.1.16.1 TDBXContext Members

The following tables list the members exposed by TDBXContext.

Protected Fields

	Name	Description
	FOnErrorDBXContext (see page 10435)	This is FOnErrorDBXContext, a member of class TDBXContext.
	FOnTraceDBXContext (see page 10435)	This is FOnTraceDBXContext, a member of class TDBXContext.

Protected Methods

	Name	Description
	Create (see page 10434)	This is the overview for the Create constructor overload.

Public Events

	Name	Description
	OnError (see page 10437)	TDBXErrorEvent (see page 10671) handler.
	OnTrace (see page 10438)	TDBXTraceEvent (see page 10672) handler.

Public Methods

	Name	Description
V	Destroy (see page 10434)	Destroys an instance of TDBXContext (see page 10432).
	Error (see page 10435)	Create (see page 10434) error event.
	IsTracing (see page 10436)	Is tracing enabled.
	Trace (see page 10436)	Create (see page 10434) trace event.

Public Properties

	Name	Description
R	ClosedByteReader (see page 10436)	Indicates closed byte readers that have no more rows.
	TraceFlags (see page 10437)	Collection of TDBXTraceFlags (see page 10617).

55.1.16.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TDBXContext.Create () (see page 10434)	Creates an instance of TDBXContext (see page 10432)
	TDBXContext.Create (TDBXContext) (see page 10434)	Creates an instance of TDBXContext (see page 10432)

55.1.16.2.1 TDBXContext.Create Constructor ()

Creates an instance of TDBXContext (see page 10432)

Pascal

```
constructor Create(DBXContext: TDBXContext); overload;
constructor Create; overload;
```

C++

```
__fastcall TDBXContext(TDBXContext DBXContext);
__fastcall TDBXContext();
```

Description

This constructor creates and initializes an instance of TDBXContext (see page 10432).

See Also

Destroy (see page 10434)

55.1.16.3 TDBXContext.Destroy Destructor

Destroys an instance of TDBXContext (see page 10432).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXContext();
```

Description

This destructor destroys an instance of TDBXContext (see page 10432).

55.1.16.4 TDBXContext Fields

The fields of the TDBXContext class are listed here.

Protected Fields

	Name	Description
	FOnErrorDBXContext (see page 10435)	This is FOnErrorDBXContext, a member of class TDBXContext.
	FOnTraceDBXContext (see page 10435)	This is FOnTraceDBXContext, a member of class TDBXContext.

55.1.16.4.1 TDBXContext.FOnErrorDBXContext Field

Pascal

```
FOnErrorDBXContext: TDBXContext;
```

C++

```
TDBXContext FOnErrorDBXContext;
```

Description

This is FOnErrorDBXContext, a member of class TDBXContext.

55.1.16.4.2 TDBXContext.FOnTraceDBXContext Field

Pascal

```
FOnTraceDBXContext: TDBXContext;
```

C++

```
TDBXContext FOnTraceDBXContext;
```

Description

This is FOnTraceDBXContext, a member of class TDBXContext.

55.1.16.5 TDBXContext Methods

The methods of the TDBXContext class are listed here.

Public Methods

	Name	Description
	Error (see page 10435)	Create (see page 10434) error event.
	IsTracing (see page 10436)	Is tracing enabled.
	Trace (see page 10436)	Create (see page 10434) trace event.

55.1.16.5.1 TDBXContext.Error Method

Create (see page 10434) error event.

Pascal

```
procedure Error(ErrorCode: TDBXErrorCode; ErrorMessage: WideString);
```

C++

```
__fastcall Error(TDBXErrorCode ErrorCode, BSTR ErrorMessage);
```

Description

This method creates a TDBXErrorEvent (see page 10671) error event. **ErrorCode** is one of the defined TDBXErrorCodes (see page 10489) values and **ErrorMessage** is the appropriate error text.

See Also

TDBXErrorCodes (see page 10489)

TDBXErrorEvent (see page 10671)

55.1.16.5.2 TDBXContext.IsTracing Method

Is tracing enabled.

Pascal

```
function IsTracing(TraceFlags: TDBXTraceFlag): Boolean;
```

C++

```
__fastcall Boolean IsTracing(TDBXTraceFlag TraceFlags);
```

Description

This method returns true if tracing is enabled for the given **TDBXTraceFlag (see page 10673)** and false otherwise.

See Also

TDBXTraceFlags (see page 10617)

55.1.16.5.3 TDBXContext.Trace Method

Create (see page 10434) trace event.

Pascal

```
function Trace(TraceFlag: TDBXTraceFlag; TraceMessage: WideString): CBRType;
```

C++

```
__fastcall CBRType Trace(TDBXTraceFlag TraceFlag, BSTR TraceMessage);
```

Description

This method creates a TDBXTraceEvent (see page 10672) trace event. .**TraceFlag** is one of the defined TDBXTraceFlags (see page 10617) values and **TraceMessage** is appropriate text.

See Also

TDBXTraceFlags (see page 10617)

TDBXTraceEvent (see page 10672)

55.1.16.6 TDBXContext Properties

The properties of the TDBXContext class are listed here.

Public Properties

	Name	Description
R	ClosedByteReader (see page 10436)	Indicates closed byte readers that have no more rows.
	TraceFlags (see page 10437)	Collection of TDBXTraceFlags (see page 10617).

55.1.16.6.1 TDBXContext.ClosedByteReader Property

Indicates closed byte readers that have no more rows.

Pascal

```
property ClosedByteReader: TDBXByteReader;
```

C++

```
__property TDBXByteReader ClosedByteReader;
```

Description

The framework marks TDBXByteReaders with no more rows as closed with this property.

See Also

TDBXByteReaders

ClosedReader

55.1.16.6.2 TDBXContext.TraceFlags Property

Collection of TDBXTraceFlags (see page 10617).

Pascal

```
property TraceFlags: TDBXTraceFlag;
```

C++

```
__property TDBXTraceFlag TraceFlags;
```

Description

This property holds a collection of TDBXTraceFlags (see page 10617) constants for which tracing is enabled. You can set this value to enable or disable various trace events.

See Also

TDBXTraceFlags (see page 10617)

Trace (see page 10436)

55.1.16.7 TDBXContext Events

The events of the TDBXContext class are listed here.

Public Events

	Name	Description
	OnError (see page 10437)	TDBXErrorEvent (see page 10671) handler.
	OnTrace (see page 10438)	TDBXTraceEvent (see page 10672) handler.

55.1.16.7.1 TDBXContext.OnError Event

TDBXErrorEvent (see page 10671) handler.

Pascal

```
property OnError: TDBXErrorEvent;
```

C++

```
__property TDBXErrorEvent OnError;
```

Description

This property contains the TDBXErrorEvent (see page 10671) handler.

See Also

TDBXErrorCodes (see page 10489)

Error (see page 10435)

TDBXErrorEvent (see page 10671)

55.1.16.7.2 TDBXContext.OnTrace Event

TDBXTraceEvent (see page 10672) handler.

Pascal

```
property OnTrace: TDBXTraceEvent;
```

C++

```
__property TDBXTraceEvent OnTrace;
```

Description

This property contains the TDBXTraceEvent (see page 10672) handler.

See Also

TDBXTraceFlags (see page 10617)

Trace (see page 10436)

TDBXTraceEvent (see page 10672)

55.1.17 TDBXCursorValue Class

Implementation of TDBXWritableValue (see page 10661) for TDBXDataTypes.CursorType.

Class Hierarchy

Pascal

```
TDBXCursorValue = class(TDBXWritableValue);
```

C++

```
class TDBXCursorValue : public TDBXWritableValue;
```

File

DBXCommon

Description

TDBXCursorValue implements TDBXWritableValue (see page 10661) for TDBXDataTypes.CursorType. This class is used for Oracle drivers to specify that a Reader (Oracle Cursor) gets returned from the execution of a stored procedure.

See Also

TDBXDataTypes (see page 10447)

TDBXWritableValue (see page 10661)

55.1.17.1 TDBXCursorValue Members

The following tables list the members exposed by TDBXCursorValue.

Protected Methods

	Name	Description
	Create (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
	Create (see page 10662)	Creates an instance of TDBXWritableValue (see page 10661)
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.
V	SetPendingValue (see page 10666)	This is SetPendingValue, a member of class TDBXWritableValue.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Methods

	Name	Description
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
V	Destroy (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
V	Destroy (see page 10663)	Destroys a TDBXWritableValue (see page 10661) instance.
V	GetAnsiString (see page 10629)	Gets the string value for ANSI strings.
V	GetBcd (see page 10629)	Gets the BCD value for binary-compressed decimal data types.
V	GetBoolean (see page 10629)	Gets the boolean value for boolean data types.
V	GetBytes (see page 10629)	Gets the byte array value for byte array and blob types.
V	GetDate (see page 10630)	Gets the date value for date data types.
V	GetDouble (see page 10630)	Gets the double floating point value for double data types.
V	GetInt16 (see page 10631)	Gets the smallInt value for 16-byte integer data types.
V	GetInt32 (see page 10631)	Gets the 32-byte integer value for 32-byte integer data types.
V	GetInt64 (see page 10631)	Gets the 64-byte integer value for 64-byte integer data types.
V	GetTime (see page 10631)	Gets the time value for time data types.
V	GetTimeStamp (see page 10632)	Gets the timestamp value for timestamp data types.
V	GetValueSize (see page 10632)	Gets the size in bytes for values.
V	GetWideString (see page 10632)	Gets the wide string value for wide string data types.
V	IsNull (see page 10440)	Returns true if the value is **null**.
V	SetAnsiString (see page 10663)	Sets the string value for ANSI strings.
V	SetBcd (see page 10664)	Sets the BCD value for binary-compressed decimal data types.
V	SetBoolean (see page 10664)	Sets the boolean value for boolean data types..
V	SetDate (see page 10664)	Sets the date value for date data types..
V	SetDouble (see page 10664)	Sets the double floating point value for double data types.
V	SetDynamicBytes (see page 10665)	Sets the dynamic array value for byte array and blob types.
V	SetInt16 (see page 10665)	Sets the the smallInt value for 16-byte integer data types.
V	SetInt32 (see page 10665)	Sets the 32-byte integer value for 32-byte integer data types.
V	SetInt64 (see page 10665)	Sets the 64-byte integer value for 64-byte integer data types.
V	SetNull (see page 10666)	Sets a writable value of any type to null.
V	SetStaticBytes (see page 10666)	Sets the open array value for byte array and blob types
V	SetTime (see page 10666)	Sets the time value for time data types.
V	SetTimeStamp (see page 10667)	Sets the timestamp value for timestamp data types.
V	SetWideString (see page 10667)	Sets the wide string value for wide string data types.

Public Properties

	Name	Description
R	ValueType (see page 10633)	Gets the value type for this value.

55.1.17.2 TDBXCursorValue Methods

The methods of the TDBXCursorValue class are listed here.

Public Methods

	Name	Description
V	IsNull (see page 10440)	Returns true if the value is **null**.

55.1.17.2.1 TDBXCursorValue.IsNull Method

Returns true if the value is **null**.

Pascal

```
function IsNull: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsNull();
```

Description

Returns true if the value is **null**. This method should be called before calling any of the Get methods. If the value is null, the Get methods will throw TDBXError exception if they are called.

55.1.18 TDBXDatabaseMetaData Class

Common database specific metadata.

Class Hierarchy

Pascal

```
TDBXDatabaseMetaData = class;
```

C++

```
class TDBXDatabaseMetaData;
```

File

DBXCommon

Description

TDBXDatabaseMetadata provides convenient access to common database specific metadata. This metadata can also be retrieved by creating a TDBXCommand (see page 10378) instance with its CommandType property set to TDBXCommandTypes.DbxMetadata, setting its TDBXCommand.Text property to 'GetDatabase', and calling TDBXCommand.ExecuteQuery. For most applications, it is more straightforward and efficient to retrieve this metadata using TDBXDatabaseMetadata because it executes the 'GetDatabase' command once and caches the retrieved metadata.

See Also

TDBXCommand (see page 10378)

TDBXCommandTypes (see page 10396)

CommandType (see page 10388)

Text (see page 10390)

ExecuteQuery (see page 10384)

55.1.18.1 TDBXDatabaseMetaData Members

The following tables list the members exposed by TDBXDatabaseMetaData.

Protected Fields

	Name	Description
	FDBXContext (see page 10442)	This is FDBXContext, a member of class TDBXDatabaseMetaData.

Protected Methods

	Name	Description
	Create (see page 10441)	Creates an instance of TDBXDatabaseMetaData (see page 10440)

Public Properties

	Name	Description
R	MaxCommands (see page 10442)	Maximum number of commands that can be created
R	ProcedureQuoteChar (see page 10442)	Quote character used for identifying stored procedures in commands.
R	QuoteChar (see page 10443)	Quote character used for quoting identifiers in commands.
R	SupportsNestedTransactions (see page 10443)	Indicates whether driver supports nested transactions
R	SupportsRowSetSize (see page 10443)	Is TDBXCommand.RowSetSize property supported?
R	SupportsTransactions (see page 10444)	Indicates whether the driver supports transactions

55.1.18.2 TDBXDatabaseMetaData.Create Constructor

Creates an instance of TDBXDatabaseMetaData (see page 10440)

Pascal

```
constructor Create(DBXContext: TDBXContext);
```

C++

```
__fastcall TDBXDatabaseMetaData(TDBXContext DBXContext);
```

Description

This constructor creates and initializes an instance of TDBXDatabaseMetaData (see page 10440).

55.1.18.3 TDBXDatabaseMetaData Fields

The fields of the TDBXDatabaseMetaData class are listed here.

Protected Fields

	Name	Description
	FDBXContext (see page 10442)	This is FDBXContext, a member of class TDBXDatabaseMetaData.

55.1.18.3.1 TDBXDatabaseMetaData.FDBXContext Field

Pascal

```
FDBXContext: TDBXContext;
```

C++

```
TDBXContext FDBXContext;
```

Description

This is FDBXContext, a member of class TDBXDatabaseMetaData.

55.1.18.4 TDBXDatabaseMetaData Properties

The properties of the TDBXDatabaseMetaData class are listed here.

Public Properties

	Name	Description
R	MaxCommands (see page 10442)	Maximum number of commands that can be created
R	ProcedureQuoteChar (see page 10442)	Quote character used for identifying stored procedures in commands.
R	QuoteChar (see page 10443)	Quote character used for quoting identifiers in commands.
R	SupportsNestedTransactions (see page 10443)	Indicates whether driver supports nested transactions
R	SupportsRowSetSize (see page 10443)	Is TDBXCommand.RowSetSize property supported?
R	SupportsTransactions (see page 10444)	Indicates whether the driver supports transactions

55.1.18.4.1 TDBXDatabaseMetaData.MaxCommands Property

Maximum number of commands that can be created

Pascal

```
property MaxCommands: TInt32;
```

C++

```
__property TInt32 MaxCommands;
```

Description

This property indicates the maximum number of commands that can be created.

55.1.18.4.2 TDBXDatabaseMetaData.ProcedureQuoteChar Property

Quote character used for identifying stored procedures in commands.

Pascal

```
property ProcedureQuoteChar: WideString;
```

C++

```
__property BSTR ProcedureQuoteChar;
```

Description

This property contains the quote character used for quoting identifiers for stored procedures in the TDBXCommand.Text property.

See Also

TDBXCommand (see page 10378)

TDBXCommand.Text

55.1.18.4.3 TDBXDatabaseMetaData.QuoteChar Property

Quote character used for quoting identifiers in commands.

Pascal

```
property QuoteChar: WideString;
```

C++

```
__property BSTR QuoteChar;
```

Description

This property contains the quote character used for quoting identifiers in the TDBXCommand.Text property.

Warning: deprecated. Use QuotePrefix and QuoteSuffix properties in TDBXDatabaseMetaDataEx (see page 10444).

See Also

TDBXCommand (see page 10378)

TDBXDatabaseMetaDataEx (see page 10444)

QuotePrefix (see page 10446)

QuoteSuffix (see page 10446)

Text (see page 10390)

55.1.18.4.4 TDBXDatabaseMetaData.SupportsNestedTransactions Property

Indicates whether driver supports nested transactions

Pascal

```
property SupportsNestedTransactions: Boolean;
```

C++

```
__property Boolean SupportsNestedTransactions;
```

Description

This property is true if the driver support nested transactions and false otherwise.

See Also

SupportsTransactions (see page 10444)

55.1.18.4.5 TDBXDatabaseMetaData.SupportsRowSetSize Property

Is TDBXCommand.RowSetSize property supported?

Pascal

```
property SupportsRowSetSize: Boolean;
```

C++

```
__property Boolean SupportsRowSetSize;
```

Description

This property indicates whether the TDBXCommand.RowSetSize property is supported to control row buffering when executing

queries. The constant DBXDefaultRowSetSize (see page 10675) is the default row buffering size to use for drivers that return true for this property.

See Also

TDBXCommand (see page 10378)

RowSetSize (see page 10390)

55.1.18.4.6 TDBXDatabaseMetaData.SupportsTransactions Property

Indicates whether the driver supports transactions

Pascal

```
property SupportsTransactions: Boolean;
```

C++

```
__property Boolean SupportsTransactions;
```

Description

This property is true if the driver support transactions and false otherwise.

See Also

SupportsNestedTransactions (see page 10443)

55.1.19 TDBXDatabaseMetaDataEx Class

Temporary extension of TDBXDatabaseMetaData (see page 10440).

Class Hierarchy

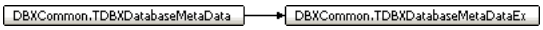

Pascal

```
TDBXDatabaseMetaDataEx = class(TDBXDatabaseMetaData);
```

C++

```
class TDBXDatabaseMetaDataEx : public TDBXDatabaseMetaData;
```

File

DBXCommon

Description

TDBXDatabaseMetaDataEx is an extension of the TDBXDatabaseMetaData (see page 10440) class, which provides convenient access to common database specific metadata.

Note: Classes (see page 2685) ending in “Ex” are temporary classes that exist to avoid breaking interfaces. Their contents will be migrated to their base class in the next release.

See Also

TDBXDatabaseMetaData (see page 10440)

55.1.19.1 TDBXDatabaseMetaDataEx Members

The following tables list the members exposed by TDBXDatabaseMetaDataEx.

Protected Fields

	Name	Description
	FDBXContext (see page 10442)	This is FDBXContext, a member of class TDBXDatabaseMetaData.

Protected Methods

	Name	Description
	Create (see page 10445)	Creates an instance of TDBXDatabaseMetaDataEx (see page 10444)

Public Properties

	Name	Description
R	MaxCommands (see page 10442)	Maximum number of commands that can be created
R	MetaDataVersion (see page 10446)	dbExpress version of metadata supported.
R	ProcedureQuoteChar (see page 10442)	Quote character used for identifying stored procedures in commands.
R	QuoteChar (see page 10443)	Quote character used for quoting identifiers in commands.
R	QuotePrefix (see page 10446)	Character to start a quoted identifier.
R	QuoteSuffix (see page 10446)	Character to end a quoted identifier.
R	SupportsLowerCaseIdentifiers (see page 10446)	Lowercase identifiers are stored in lowercase characters in the database.
R	SupportsNestedTransactions (see page 10443)	Indicates whether driver supports nested transactions
R	SupportsRowSetSize (see page 10443)	Is TDBXCommand.RowSetSize property supported?
R	SupportsTransactions (see page 10444)	Indicates whether the driver supports transactions
R	SupportsUpperCaseIdentifiers (see page 10447)	Uppercase identifiers are stored in uppercase characters in the database.

55.1.19.2 TDBXDatabaseMetaDataEx.Create Constructor

Creates an instance of TDBXDatabaseMetaDataEx (see page 10444)

Pascal

```
constructor Create(DBXContext: TDBXContext);
```

C++

```
__fastcall TDBXDatabaseMetaDataEx(TDBXContext DBXContext);
```

Description

This constructor creates and initializes an instance of TDBXDatabaseMetaDataEx (see page 10444).

55.1.19.3 TDBXDatabaseMetaDataEx Properties

The properties of the TDBXDatabaseMetaDataEx class are listed here.

Public Properties

	Name	Description
R	MetaDataVersion (see page 10446)	dbExpress version of metadata supported.
R	QuotePrefix (see page 10446)	Character to start a quoted identifier.
R	QuoteSuffix (see page 10446)	Character to end a quoted identifier.
R	SupportsLowerCaseIdentifiers (see page 10446)	Lowercase identifiers are stored in lowercase characters in the database.
R	SupportsUpperCaseIdentifiers (see page 10447)	Uppercase identifiers are stored in uppercase characters in the database.

55.1.19.3.1 TDBXDatabaseMetaDataEx.MetaDataVersion Property

dbExpress version of metadata supported.

Pascal

```
property MetaDataVersion: String;
```

C++

```
__property AnsiString MetaDataVersion;
```

Description

Returns dbExpress version of metadata supported. It is one of the following values:

Constant	Description
DBXVersion30 (see page 10676)	dbExpress 3.0.
DBXVersion40 (see page 10676)	dbExpress 4.0.

55.1.19.3.2 TDBXDatabaseMetaDataEx.QuotePrefix Property

Character to start a quoted identifier.

Pascal

```
property QuotePrefix: WideString;
```

C++

```
__property BSTR QuotePrefix;
```

Description

QuotePrefix contains the character to start a quoted identifier.

See Also

QuoteSuffix (see page 10446)

55.1.19.3.3 TDBXDatabaseMetaDataEx.QuoteSuffix Property

Character to end a quoted identifier.

Pascal

```
property QuoteSuffix: WideString;
```

C++

```
__property BSTR QuoteSuffix;
```

Description

QuoteSuffix contains the character to end a quoted identifier.

See Also

QuotePrefix (see page 10446)

55.1.19.3.4 TDBXDatabaseMetaDataEx.SupportsLowerCaseIdentifiers Property

Lowercase identifiers are stored in lowercase characters in the database.

Pascal

```
property SupportsLowerCaseIdentifiers: Boolean;
```

C++

```
__property Boolean SupportsLowerCaseIdentifiers;
```

Description

SupportsLowerCaseIdentifiers indicates whether lowercase identifiers are stored in lowercase characters in the database.

See Also

SupportsUpperCaseIdentifiers (see page 10447)

55.1.19.3.5 TDBXDatabaseMetaDataEx.SupportsUpperCaseIdentifiers Property

Uppercase identifiers are stored in uppercase characters in the database.

Pascal

```
property SupportsUpperCaseIdentifiers: Boolean;
```

C++

```
__property Boolean SupportsUpperCaseIdentifiers;
```

Description

SupportsUpperCaseIdentifiers indicates whether uppercase identifiers are stored in uppercase characters in the database.

See Also

SupportsLowerCaseIdentifiers (see page 10446)

55.1.20 TDBXDataTypes Class

Data types supported by the dbExpress framework (TDBX).

Class Hierarchy

Pascal

```
TDBXDataTypes = class;
```

C++

```
class TDBXDataTypes;
```

File

DBXCommon

Description

The dbExpress framework (TDBX) supports a variety of data types. This class provides a list of constants for all the types. TDBXType (see page 10673) is a type for TDBXDataTypes.

This table lists the data types.

Constant	Description
UnknownType (see page 10458)	UnknownType (see page 10458)
AnsiStringType (see page 10451)	8 bit ANSI String
DateType (see page 10453)	32 bit Date
BlobType (see page 10452)	Blob with a subtype
BooleanType (see page 10453)	16 big Boolean
Int16Type (see page 10454)	16 bit signed integer
Int32Type (see page 10455)	32 bit signed integer
DoubleType (see page 10453)	64 bit floating point
BcdType (see page 10452)	TBcd decimal from the FMTBcd (see page 13627) unit
BytesType (see page 10453)	Fixed length byte array
TimeType (see page 10457)	32 bit Time
UInt16Type (see page 10457)	TSql Unsigned 16 bit integer
Uint32Type (see page 10457)	Unsigned 32 bit integer
VarBytesType (see page 10458)	Variable length byte array with maximum length of 64 kilobytes
CursorType (see page 10453)	Oracle cursor type
Int64Type (see page 10455)	64 bit integer
Uint64Type (see page 10457)	unsigned 64 bit integer
AdtType (see page 10451)	Abstract data type
ArrayType (see page 10451)	Array data type
RefType (see page 10456)	Reference data type
TableType (see page 10456)	Nested table data type
TimeStampType (see page 10457)	TSQLTimeStamp in the SqlTimSt (see page 20383) unit
WideStringType (see page 10458)	UCS2 Unicode string
MaxBaseTypes (see page 10455)	Maximum number of base types excluding sub types
MoneySubType (see page 10455)	Monetary sub type
MemoSubType (see page 10455)	8 bit char blob
BinarySubType (see page 10452)	Binary blob
WideMemoSubType (see page 10458)	Unicode blob
HMemoSubType (see page 10454)	CLOB
HBinarySubType (see page 10454)	BLOB
BFileSubType (see page 10452)	BFILE
FixedSubType (see page 10454)	Fixed length char
AutoIncSubType (see page 10451)	32 bit integer autoincrement
AdtNestedTableSubType (see page 10451)	ADT for nested table (has no name)
AdtDateSubType (see page 10450)	DATE (OCIDate) within an ADT
OracleTimeStampSubType (see page 10456)	Oracle time stamp
OracleIntervalSubType (see page 10456)	Oracle Interval

See Also

TDBXType (see page 10673)

55.1.20.1 TDBXDataTypes Members

The following tables list the members exposed by TDBXDataTypes.

Public Constants

	Name	Description
	AdtDateSubType (see page 10450)	This is AdtDateSubType, a member of class TDBXDataTypes.
	AdtNestedTableSubType (see page 10451)	This is AdtNestedTableSubType, a member of class TDBXDataTypes.
	AdtType (see page 10451)	This is AdtType, a member of class TDBXDataTypes.
	AnsiStringType (see page 10451)	This is AnsiStringType, a member of class TDBXDataTypes.
	ArrayType (see page 10451)	This is ArrayType, a member of class TDBXDataTypes.
	AutoIncSubType (see page 10451)	This is AutoIncSubType, a member of class TDBXDataTypes.
	BcdType (see page 10452)	This is BcdType, a member of class TDBXDataTypes.
	BFileSubType (see page 10452)	This is BFileSubType, a member of class TDBXDataTypes.
	BinarySubType (see page 10452)	This is BinarySubType, a member of class TDBXDataTypes.
	BlobType (see page 10452)	This is BlobType, a member of class TDBXDataTypes.
	BooleanType (see page 10453)	This is BooleanType, a member of class TDBXDataTypes.
	BytesType (see page 10453)	This is BytesType, a member of class TDBXDataTypes.
	CursorType (see page 10453)	This is CursorType, a member of class TDBXDataTypes.
	DateType (see page 10453)	This is DateType, a member of class TDBXDataTypes.
	DoubleType (see page 10453)	This is DoubleType, a member of class TDBXDataTypes.
	FixedSubType (see page 10454)	This is FixedSubType, a member of class TDBXDataTypes.
	HBinarySubType (see page 10454)	This is HBinarySubType, a member of class TDBXDataTypes.
	HMemoSubType (see page 10454)	This is HMemoSubType, a member of class TDBXDataTypes.
	Int16Type (see page 10454)	This is Int16Type, a member of class TDBXDataTypes.
	Int32Type (see page 10455)	This is Int32Type, a member of class TDBXDataTypes.
	Int64Type (see page 10455)	This is Int64Type, a member of class TDBXDataTypes.
	MaxBaseTypes (see page 10455)	This is MaxBaseTypes, a member of class TDBXDataTypes.
	MemoSubType (see page 10455)	This is MemoSubType, a member of class TDBXDataTypes.
	MoneySubType (see page 10455)	This is MoneySubType, a member of class TDBXDataTypes.
	OracleIntervalSubType (see page 10456)	This is OracleIntervalSubType, a member of class TDBXDataTypes.
	OracleTimeStampSubType (see page 10456)	This is OracleTimeStampSubType, a member of class TDBXDataTypes.
	RefType (see page 10456)	This is RefType, a member of class TDBXDataTypes.
	TableType (see page 10456)	This is TableType, a member of class TDBXDataTypes.
	TimeStampType (see page 10457)	This is TimeStampType, a member of class TDBXDataTypes.
	TimeType (see page 10457)	This is TimeType, a member of class TDBXDataTypes.
	UInt16Type (see page 10457)	This is UInt16Type, a member of class TDBXDataTypes.
	Uint32Type (see page 10457)	This is Uint32Type, a member of class TDBXDataTypes.
	Uint64Type (see page 10457)	This is Uint64Type, a member of class TDBXDataTypes.
	UnknownType (see page 10458)	This is UnknownType, a member of class TDBXDataTypes.
	VarBytesType (see page 10458)	This is VarBytesType, a member of class TDBXDataTypes.
	WideMemoSubType (see page 10458)	This is WideMemoSubType, a member of class TDBXDataTypes.
	WideStringType (see page 10458)	This is WideStringType, a member of class TDBXDataTypes.

55.1.20.2 TDBXDataTypes Constants

The constants of the TDBXDataTypes class are listed here.

Public Constants

	Name	Description
	AdtDateSubType (see page 10450)	This is AdtDateSubType, a member of class TDBXDataTypes.
	AdtNestedTableSubType (see page 10451)	This is AdtNestedTableSubType, a member of class TDBXDataTypes.
	AdtType (see page 10451)	This is AdtType, a member of class TDBXDataTypes.
	AnsiStringType (see page 10451)	This is AnsiStringType, a member of class TDBXDataTypes.
	ArrayType (see page 10451)	This is ArrayType, a member of class TDBXDataTypes.
	AutoIncSubType (see page 10451)	This is AutoIncSubType, a member of class TDBXDataTypes.
	BcdType (see page 10452)	This is BcdType, a member of class TDBXDataTypes.
	BFileSubType (see page 10452)	This is BFileSubType, a member of class TDBXDataTypes.
	BinarySubType (see page 10452)	This is BinarySubType, a member of class TDBXDataTypes.
	BlobType (see page 10452)	This is BlobType, a member of class TDBXDataTypes.
	BooleanType (see page 10453)	This is BooleanType, a member of class TDBXDataTypes.
	BytesType (see page 10453)	This is BytesType, a member of class TDBXDataTypes.
	CursorType (see page 10453)	This is CursorType, a member of class TDBXDataTypes.
	DateType (see page 10453)	This is DateType, a member of class TDBXDataTypes.
	DoubleType (see page 10453)	This is DoubleType, a member of class TDBXDataTypes.
	FixedSubType (see page 10454)	This is FixedSubType, a member of class TDBXDataTypes.
	HBinarySubType (see page 10454)	This is HBinarySubType, a member of class TDBXDataTypes.
	HMemoSubType (see page 10454)	This is HMemoSubType, a member of class TDBXDataTypes.
	Int16Type (see page 10454)	This is Int16Type, a member of class TDBXDataTypes.
	Int32Type (see page 10455)	This is Int32Type, a member of class TDBXDataTypes.
	Int64Type (see page 10455)	This is Int64Type, a member of class TDBXDataTypes.
	MaxBaseTypes (see page 10455)	This is MaxBaseTypes, a member of class TDBXDataTypes.
	MemoSubType (see page 10455)	This is MemoSubType, a member of class TDBXDataTypes.
	MoneySubType (see page 10455)	This is MoneySubType, a member of class TDBXDataTypes.
	OracleIntervalSubType (see page 10456)	This is OracleIntervalSubType, a member of class TDBXDataTypes.
	OracleTimeStampSubType (see page 10456)	This is OracleTimeStampSubType, a member of class TDBXDataTypes.
	RefType (see page 10456)	This is RefType, a member of class TDBXDataTypes.
	TableType (see page 10456)	This is TableType, a member of class TDBXDataTypes.
	TimeStampType (see page 10457)	This is TimeStampType, a member of class TDBXDataTypes.
	TimeType (see page 10457)	This is TimeType, a member of class TDBXDataTypes.
	UInt16Type (see page 10457)	This is UInt16Type, a member of class TDBXDataTypes.
	Uint32Type (see page 10457)	This is Uint32Type, a member of class TDBXDataTypes.
	Uint64Type (see page 10457)	This is Uint64Type, a member of class TDBXDataTypes.
	UnknownType (see page 10458)	This is UnknownType, a member of class TDBXDataTypes.
	VarBytesType (see page 10458)	This is VarBytesType, a member of class TDBXDataTypes.
	WideMemoSubType (see page 10458)	This is WideMemoSubType, a member of class TDBXDataTypes.
	WideStringType (see page 10458)	This is WideStringType, a member of class TDBXDataTypes.

55.1.20.2.1 TDBXDataTypes.AdtDateSubType Constant

Pascal

```
const AdtDateSubType = 37;
```

C++

```
AdtDateSubType = 37;
```

Description

This is AdtDateSubType, a member of class TDBXDataTypes.

55.1.20.2.2 TDBXDataTypes.AdtNestedTableSubType Constant

Pascal

```
const AdtNestedTableSubType = 35;
```

C++

```
AdtNestedTableSubType = 35;
```

Description

This is AdtNestedTableSubType, a member of class TDBXDataTypes.

55.1.20.2.3 TDBXDataTypes.AdtType Constant

Pascal

```
const AdtType = 20;
```

C++

```
AdtType = 20;
```

Description

This is AdtType, a member of class TDBXDataTypes.

55.1.20.2.4 TDBXDataTypes.AnsiStringType Constant

Pascal

```
const AnsiStringType = 1;
```

C++

```
AnsiStringType = 1;
```

Description

This is AnsiStringType, a member of class TDBXDataTypes.

55.1.20.2.5 TDBXDataTypes.ArrayType Constant

Pascal

```
const ArrayType = 21;
```

C++

```
ArrayType = 21;
```

Description

This is ArrayType, a member of class TDBXDataTypes.

55.1.20.2.6 TDBXDataTypes.AutoIncSubType Constant

Pascal

```
const AutoIncSubType = 29;
```

C++

```
AutoIncSubType = 29;
```

Description

This is AutoIncSubType, a member of class TDBXDataTypes.

55.1.20.2.7 TDBXDataTypes.BcdType Constant

Pascal

```
const BcdType = 8;
```

C++

```
BcdType = 8;
```

Description

This is BcdType, a member of class TDBXDataTypes.

55.1.20.2.8 TDBXDataTypes.BFileSubType Constant

Pascal

```
const BFileSubType = 36;
```

C++

```
BFileSubType = 36;
```

Description

This is BFileSubType, a member of class TDBXDataTypes.

55.1.20.2.9 TDBXDataTypes.BinarySubType Constant

Pascal

```
const BinarySubType = 23;
```

C++

```
BinarySubType = 23;
```

Description

This is BinarySubType, a member of class TDBXDataTypes.

55.1.20.2.10 TDBXDataTypes.BlobType Constant

Pascal

```
const BlobType = 3;
```

C++

```
BlobType = 3;
```

Description

This is BlobType, a member of class TDBXDataTypes.

55.1.20.2.11 TDBXDataTypes.BooleanType Constant

Pascal

```
const BooleanType = 4;
```

C++

```
BooleanType = 4;
```

Description

This is BooleanType, a member of class TDBXDataTypes.

55.1.20.2.12 TDBXDataTypes.BytesType Constant

Pascal

```
const BytesType = 9;
```

C++

```
BytesType = 9;
```

Description

This is BytesType, a member of class TDBXDataTypes.

55.1.20.2.13 TDBXDataTypes.CursorType Constant

Pascal

```
const CursorType = 17;
```

C++

```
CursorType = 17;
```

Description

This is CursorType, a member of class TDBXDataTypes.

55.1.20.2.14 TDBXDataTypes.DateType Constant

Pascal

```
const DateType = 2;
```

C++

```
DateType = 2;
```

Description

This is DateType, a member of class TDBXDataTypes.

55.1.20.2.15 TDBXDataTypes.DoubleType Constant

Pascal

```
const DoubleType = 7;
```

C++

```
DoubleType = 7;
```

Description

This is DoubleType, a member of class TDBXDataTypes.

55.1.20.2.16 TDBXDataTypes.FixedSubType Constant

Pascal

```
const FixedSubType = 31;
```

C++

```
FixedSubType = 31;
```

Description

This is FixedSubType, a member of class TDBXDataTypes.

55.1.20.2.17 TDBXDataTypes.HBinarySubType Constant

Pascal

```
const HBinarySubType = 34;
```

C++

```
HBinarySubType = 34;
```

Description

This is HBinarySubType, a member of class TDBXDataTypes.

55.1.20.2.18 TDBXDataTypes.HMemoSubType Constant

Pascal

```
const HMemoSubType = 33;
```

C++

```
HMemoSubType = 33;
```

Description

This is HMemoSubType, a member of class TDBXDataTypes.

55.1.20.2.19 TDBXDataTypes.Int16Type Constant

Pascal

```
const Int16Type = 5;
```

C++

```
Int16Type = 5;
```

Description

This is Int16Type, a member of class TDBXDataTypes.

55.1.20.2.20 TDBXDataTypes.Int32Type Constant

Pascal

```
const Int32Type = 6;
```

C++

```
Int32Type = 6;
```

Description

This is Int32Type, a member of class TDBXDataTypes.

55.1.20.2.21 TDBXDataTypes.Int64Type Constant

Pascal

```
const Int64Type = 18;
```

C++

```
Int64Type = 18;
```

Description

This is Int64Type, a member of class TDBXDataTypes.

55.1.20.2.22 TDBXDataTypes.MaxBaseTypes Constant

Pascal

```
const MaxBaseTypes = 27;
```

C++

```
MaxBaseTypes = 27;
```

Description

This is MaxBaseTypes, a member of class TDBXDataTypes.

55.1.20.2.23 TDBXDataTypes.MemoSubType Constant

Pascal

```
const MemoSubType = 22;
```

C++

```
MemoSubType = 22;
```

Description

This is MemoSubType, a member of class TDBXDataTypes.

55.1.20.2.24 TDBXDataTypes.MoneySubType Constant

Pascal

```
const MoneySubType = 21;
```

C++

```
MoneySubType = 21;
```

Description

This is MoneySubType, a member of class TDBXDataTypes.

55.1.20.2.25 TDBXDataTypes.OracleIntervalSubType Constant

Pascal

```
const OracleIntervalSubType = 39;
```

C++

```
OracleIntervalSubType = 39;
```

Description

This is OracleIntervalSubType, a member of class TDBXDataTypes.

55.1.20.2.26 TDBXDataTypes.OracleTimeStampSubType Constant

Pascal

```
const OracleTimeStampSubType = 38;
```

C++

```
OracleTimeStampSubType = 38;
```

Description

This is OracleTimeStampSubType, a member of class TDBXDataTypes.

55.1.20.2.27 TDBXDataTypes.RefType Constant

Pascal

```
const RefType = 22;
```

C++

```
RefType = 22;
```

Description

This is RefType, a member of class TDBXDataTypes.

55.1.20.2.28 TDBXDataTypes.TableType Constant

Pascal

```
const TableType = 23;
```

C++

```
TableType = 23;
```

Description

This is TableType, a member of class TDBXDataTypes.

55.1.20.2.29 TDBXDataTypes.TimeStampType Constant

Pascal

```
const TimeStampType = 24;
```

C++

```
TimeStampType = 24;
```

Description

This is TimeStampType, a member of class TDBXDataTypes.

55.1.20.2.30 TDBXDataTypes.TimeType Constant

Pascal

```
const TimeType = 10;
```

C++

```
TimeType = 10;
```

Description

This is TimeType, a member of class TDBXDataTypes.

55.1.20.2.31 TDBXDataTypes.UInt16Type Constant

Pascal

```
const UInt16Type = 12;
```

C++

```
UInt16Type = 12;
```

Description

This is UInt16Type, a member of class TDBXDataTypes.

55.1.20.2.32 TDBXDataTypes.Uint32Type Constant

Pascal

```
const Uint32Type = 13;
```

C++

```
Uint32Type = 13;
```

Description

This is Uint32Type, a member of class TDBXDataTypes.

55.1.20.2.33 TDBXDataTypes.Uint64Type Constant

Pascal

```
const Uint64Type = 19;
```

C++

```
Uint64Type = 19;
```

Description

This is Uint64Type, a member of class TDBXDataTypes.

55.1.20.2.34 TDBXDataTypes.UnknownType Constant

Pascal

```
const UnknownType = 0;
```

C++

```
UnknownType = 0;
```

Description

This is UnknownType, a member of class TDBXDataTypes.

55.1.20.2.35 TDBXDataTypes.VarBytesType Constant

Pascal

```
const VarBytesType = 15;
```

C++

```
VarBytesType = 15;
```

Description

This is VarBytesType, a member of class TDBXDataTypes.

55.1.20.2.36 TDBXDataTypes.WideMemoSubType Constant

Pascal

```
const WideMemoSubType = 32;
```

C++

```
WideMemoSubType = 32;
```

Description

This is WideMemoSubType, a member of class TDBXDataTypes.

55.1.20.2.37 TDBXDataTypes.WideStringType Constant

Pascal

```
const WideStringType = 26;
```

C++

```
WideStringType = 26;
```

Description

This is WideStringType, a member of class TDBXDataTypes.

55.1.21 TDBXDataTypesEx Class

Data types supported by dbExpress.

Class Hierarchy

DBXCommon.TDBXDataTypesEx

Pascal

```
TDBXDataTypesEx = class;
```

C++

```
class TDBXDataTypesEx;
```

File

DBXCommon

Description

TDBXDataTypesEx lists additional data types supported by dbExpress.

Note: Classes (see page 2685) ending in “Ex” are temporary classes that exist to avoid breaking interfaces. Their contents will be migrated to their base class in the next release.

This table contains data types.

Constant	Description
SingleType (see page 10461)	32 bit floating point
Int8Type (see page 10460)	8 bit signed integer
Uint8Type (see page 10461)	8 bit unsigned integer
ObjectType (see page 10460)	Object serialization
CharArrayType (see page 10460)	Character array
IntervalType (see page 10460)	Time Interval

See Also

TDBXDataTypes (see page 10447)

55.1.21.1 TDBXDataTypesEx Members

The following tables list the members exposed by TDBXDataTypesEx.

Public Constants

	Name	Description
	CharArrayType (see page 10460)	This is CharArrayType, a member of class TDBXDataTypesEx.
	Int8Type (see page 10460)	This is Int8Type, a member of class TDBXDataTypesEx.
	IntervalType (see page 10460)	This is IntervalType, a member of class TDBXDataTypesEx.
	ObjectType (see page 10460)	This is ObjectType, a member of class TDBXDataTypesEx.
	SingleType (see page 10461)	This is SingleType, a member of class TDBXDataTypesEx.
	Uint8Type (see page 10461)	This is Uint8Type, a member of class TDBXDataTypesEx.

55.1.21.2 TDBXDataTypesEx Constants

The constants of the TDBXDataTypesEx class are listed here.

Public Constants

	Name	Description
	CharArrayType (see page 10460)	This is CharArrayType, a member of class TDBXDataTypesEx.
	Int8Type (see page 10460)	This is Int8Type, a member of class TDBXDataTypesEx.
	IntervalType (see page 10460)	This is IntervalType, a member of class TDBXDataTypesEx.
	ObjectType (see page 10460)	This is ObjectType, a member of class TDBXDataTypesEx.
	SingleType (see page 10461)	This is SingleType, a member of class TDBXDataTypesEx.
	Uint8Type (see page 10461)	This is Uint8Type, a member of class TDBXDataTypesEx.

55.1.21.2.1 TDBXDataTypesEx.CharArrayType Constant

Pascal

```
const CharArrayType = 31;
```

C++

```
CharArrayType = 31;
```

Description

This is CharArrayType, a member of class TDBXDataTypesEx.

55.1.21.2.2 TDBXDataTypesEx.Int8Type Constant

Pascal

```
const Int8Type = 28;
```

C++

```
Int8Type = 28;
```

Description

This is Int8Type, a member of class TDBXDataTypesEx.

55.1.21.2.3 TDBXDataTypesEx.IntervalType Constant

Pascal

```
const IntervalType = 32;
```

C++

```
IntervalType = 32;
```

Description

This is IntervalType, a member of class TDBXDataTypesEx.

55.1.21.2.4 TDBXDataTypesEx.ObjectType Constant

Pascal

```
const ObjectType = 30;
```

C++

```
ObjectType = 30;
```

Description

This is ObjectType, a member of class TDBXDataTypesEx.

55.1.21.2.5 TDBXDataTypesEx.SingleType Constant

Pascal

```
const SingleType = 27;
```

C++

```
SingleType = 27;
```

Description

This is SingleType, a member of class TDBXDataTypesEx.

55.1.21.2.6 TDBXDataTypesEx.Uint8Type Constant

Pascal

```
const Uint8Type = 29;
```

C++

```
Uint8Type = 29;
```

Description

This is Uint8Type, a member of class TDBXDataTypesEx.

55.1.22 TDBXDateValue Class

Implementation of writable date type values.

Class Hierarchy

Pascal

```
TDBXDateValue = class(TDBXWritableValue);
```

C++

```
class TDBXDateValue : public TDBXWritableValue;
```

File

DBXCommon

Description

Implementation of TDBXWritableValue (see page 10661) for TDBXDataTypes.DateType values.

55.1.22.1 TDBXDateValue Members

The following tables list the members exposed by TDBXDateValue.

Protected Methods

	Name	Description
	Create (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
	Create (see page 10662)	Creates an instance of TDBXWritableValue (see page 10661)
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.
V	SetPendingValue (see page 10666)	This is SetPendingValue, a member of class TDBXWritableValue.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Methods

	Name	Description
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
V	Destroy (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
V	Destroy (see page 10663)	Destroys a TDBXWritableValue (see page 10661) instance.
V	GetAnsiString (see page 10629)	Gets the string value for ANSI strings.
V	GetBcd (see page 10629)	Gets the BCD value for binary-compressed decimal data types.
V	GetBoolean (see page 10629)	Gets the boolean value for boolean data types.
V	GetBytes (see page 10629)	Gets the byte array value for byte array and blob types.
V	GetDate (see page 10463)	Gets the date value for date data types.
V	GetDouble (see page 10630)	Gets the double floating point value for double data types.
V	GetInt16 (see page 10631)	Gets the smallInt value for 16-byte integer data types.
V	GetInt32 (see page 10631)	Gets the 32-byte integer value for 32-byte integer data types.
V	GetInt64 (see page 10631)	Gets the 64-byte integer value for 64-byte integer data types.
V	GetTime (see page 10631)	Gets the time value for time data types.
V	GetTimeStamp (see page 10632)	Gets the timestamp value for timestamp data types.
V	GetValueSize (see page 10632)	Gets the size in bytes for values.
V	GetWideString (see page 10632)	Gets the wide string value for wide string data types.
V	IsNull (see page 10463)	Returns true if the value is **null**.
V	SetAnsiString (see page 10663)	Sets the string value for ANSI strings.
V	SetBcd (see page 10664)	Sets the BCD value for binary-compressed decimal data types.
V	SetBoolean (see page 10664)	Sets the boolean value for boolean data types..
V	SetDate (see page 10463)	Sets the date value for date data types..
V	SetDouble (see page 10664)	Sets the double floating point value for double data types.
V	SetDynamicBytes (see page 10665)	Sets the dynamic array value for byte array and blob types.
V	SetInt16 (see page 10665)	Sets the the smallInt value for 16-byte integer data types.
V	SetInt32 (see page 10665)	Sets the 32-byte integer value for 32-byte integer data types.
V	SetInt64 (see page 10665)	Sets the 64-byte integer value for 64-byte integer data types.
V	SetNull (see page 10666)	Sets a writable value of any type to null.
V	SetStaticBytes (see page 10666)	Sets the open array value for byte array and blob types
V	SetTime (see page 10666)	Sets the time value for time data types.
V	SetTimeStamp (see page 10667)	Sets the timestamp value for timestamp data types.
V	SetWideString (see page 10667)	Sets the wide string value for wide string data types.

Public Properties

	Name	Description
R	ValueType (see page 10633)	Gets the value type for this value.

55.1.22.2 TDBXDateValue Methods

The methods of the TDBXDateValue class are listed here.

Public Methods

	Name	Description
V	GetDate (see page 10463)	Gets the date value for date data types.
V	IsNull (see page 10463)	Returns true if the value is **null**.
V	SetDate (see page 10463)	Sets the date value for date data types..

55.1.22.2.1 TDBXDateValue.GetDate Method

Gets the date value for date data types.

Pascal

```
function GetDate: TDBXDate; override;
```

C++

```
virtual __fastcall TDBXDate GetDate();
```

Description

Gets the date value for TDBXValue (see page 10626) instances with a data type of DateType. Throws a TDBXError (see page 10486) exception if IsNull (see page 10463) is true.

55.1.22.2.2 TDBXDateValue.IsNull Method

Returns true if the value is **null**.

Pascal

```
function IsNull: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsNull();
```

Description

Returns true if the value is **null**. This method should be called before calling any of the Get methods. If the value is null, the Get methods will throw TDBXError (see page 10486) exception if they are called.

55.1.22.2.3 TDBXDateValue.SetDate Method

Sets the date value for date data types..

Pascal

```
procedure SetDate(Value: TDBXDate); override;
```

C++

```
virtual __fastcall SetDate(TDBXDate Value);
```

Description

SetDate

Sets the date value for TDBXValue (see page 10626) instances with a data type of DateType.

55.1.23 TDBXDelegateDriver Class

Delegate driver base class.

Class Hierarchy

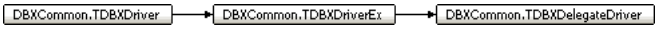

Pascal

```
TDBXDelegateDriver = class(TDBXDriverEx);
```

C++

```
class TDBXDelegateDriver : public TDBXDriverEx;
```

File

DBXCommon

Description

This is the delegate driver base class. The TDBXConnectionFactory.GetDriver method also returns an instance of TDBXDelegateDriver, so it can control the loading and unloading of drivers.

See Also

TDBXConnectionFactory (see page 10422)

GetDriver (see page 10428)

55.1.23.1 TDBXDelegateDriver Members

The following tables list the members exposed by TDBXDelegateDriver.

Protected Fields

	Name	Description
	FCommandFactories (see page 10479)	This is FCommandFactories, a member of class TDBXDriverEx.
	FDBXContext (see page 10473)	This is FDBXContext, a member of class TDBXDriver.
	FDriverLoader (see page 10474)	This is FDriverLoader, a member of class TDBXDriver.

Protected Methods

	Name	Description
	AddCommandFactory (see page 10479)	This is AddCommandFactory, a member of class TDBXDriverEx.
V	Close (see page 10466)	This is Close, a member of class TDBXDelegateDriver.
	Create (see page 10465)	This is Create, a member of class TDBXDelegateDriver.
V	CreateConnection (see page 10466)	Create instance of TDBXConnection.
V	CreateMorphCommand (see page 10466)	This is CreateMorphCommand, a member of class TDBXDelegateDriver.
V	GetDriverName (see page 10466)	This is GetDriverName, a member of class TDBXDelegateDriver.
V	GetDriverProperties (see page 10467)	This is GetDriverProperties, a member of class TDBXDelegateDriver.
	LoadMetaDataCommandFactory (see page 10479)	This is LoadMetaDataCommandFactory, a member of class TDBXDriverEx.

	SetDriverName (see page 10467)	This is SetDriverName, a member of class TDBXDelegateDriver.
	SetDriverProperties (see page 10467)	This is SetDriverProperties, a member of class TDBXDelegateDriver.

Public Methods

	Name	Description
	Destroy (see page 10465)	This is Destroy, a member of class TDBXDelegateDriver.
	GetDriverVersion (see page 10467)	Returns version information for this driver.

Public Properties

	Name	Description
	DriverName (see page 10476)	A unique name for this driver.
	DriverProperties (see page 10477)	Driver-specific properties for a particular driver.

Strict protected Methods

	Name	Description
	CacheUntilFinalization (see page 10474)	This is CacheUntilFinalization, a member of class TDBXDriver.

55.1.23.2 TDBXDelegateDriver.Create Constructor

Pascal

```
constructor Create(Driver: TDBXDriver);
```

C++

```
__fastcall TDBXDelegateDriver(TDBXDriver Driver);
```

Description

This is Create, a member of class TDBXDelegateDriver.

55.1.23.3 TDBXDelegateDriver.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXDelegateDriver();
```

Description

This is Destroy, a member of class TDBXDelegateDriver.

55.1.23.4 TDBXDelegateDriver Methods

The methods of the TDBXDelegateDriver class are listed here.

Protected Methods

	Name	Description
	Close (see page 10466)	This is Close, a member of class TDBXDelegateDriver.
	CreateConnection (see page 10466)	Create instance of TDBXConnection.
	CreateMorphCommand (see page 10466)	This is CreateMorphCommand, a member of class TDBXDelegateDriver.
	GetDriverName (see page 10466)	This is GetDriverName, a member of class TDBXDelegateDriver.
	GetDriverProperties (see page 10467)	This is GetDriverProperties, a member of class TDBXDelegateDriver.
	SetDriverName (see page 10467)	This is SetDriverName, a member of class TDBXDelegateDriver.

	SetDriverProperties (see page 10467)	This is SetDriverProperties, a member of class TDBXDelegateDriver.

Public Methods

	Name	Description
	GetDriverVersion (see page 10467)	Returns version information for this driver.

55.1.23.4.1 TDBXDelegateDriver.Close Method

Pascal

```
procedure Close; override;
```

C++

```
virtual __fastcall Close();
```

Description

This is Close, a member of class TDBXDelegateDriver.

55.1.23.4.2 TDBXDelegateDriver.CreateConnection Method

Create instance of TDBXConnection.

Pascal

```
function CreateConnection(ConnectionBuilder: TDBXConnectionBuilder): TDBXConnection; override;
```

C++

```
virtual __fastcall TDBXConnection CreateConnection(TDBXConnectionBuilder ConnectionBuilder);
```

Description

Used by TDBXConnectionFactory to create instances of TDBXConnection.

See Also

TDBXConnection (see page 10398)

TDBXConnectionFactory (see page 10422)

55.1.23.4.3 TDBXDelegateDriver.CreateMorphCommand Method

Pascal

```
function CreateMorphCommand(DbxContext: TDBXContext; Connection: TDBXConnection;
MorphicCommand: TDBXCommand): TDBXCommand; override;
```

C++

```
virtual __fastcall TDBXCommand CreateMorphCommand(TDBXContext DbxContext, TDBXConnection
Connection, TDBXCommand MorphicCommand);
```

Description

This is CreateMorphCommand, a member of class TDBXDelegateDriver.

55.1.23.4.4 TDBXDelegateDriver.GetDriverName Method

Pascal

```
function GetDriverName: WideString; override;
```

C++

```
virtual __fastcall BSTR GetDriverName();
```

Description

This is GetDriverName, a member of class TDBXDelegateDriver.

55.1.23.4.5 TDBXDelegateDriver.GetDriverProperties Method

Pascal

```
function GetDriverProperties: TDBXProperties; override;
```

C++

```
virtual __fastcall TDBXProperties GetDriverProperties();
```

Description

This is GetDriverProperties, a member of class TDBXDelegateDriver.

55.1.23.4.6 TDBXDelegateDriver.GetDriverVersion Method

Returns version information for this driver.

Pascal

```
function GetDriverVersion: WideString; override;
```

C++

```
virtual __fastcall BSTR GetDriverVersion();
```

Description

Use GetDriverVersion to return version information for an instance of TDBXDriver.

55.1.23.4.7 TDBXDelegateDriver.SetDriverName Method

Pascal

```
procedure SetDriverName(const DriverName: WideString); override;
```

C++

```
virtual __fastcall SetDriverName(const BSTR DriverName);
```

Description

This is SetDriverName, a member of class TDBXDelegateDriver.

55.1.23.4.8 TDBXDelegateDriver.SetDriverProperties Method

Pascal

```
procedure SetDriverProperties(DriverProperties: TDBXProperties); override;
```

C++

```
virtual __fastcall SetDriverProperties(TDBXProperties DriverProperties);
```

Description

This is SetDriverProperties, a member of class TDBXDelegateDriver.

55.1.24 TDBXDelegateItem Class

Used by driver implementors.

Class Hierarchy

DBXCommon.TDBXDelegateItem

Pascal

```
TDBXDelegateItem = class;
```

C++

```
class TDBXDelegateItem;
```

File

DBXCommon

Description

This class is used by driver implementors.

See Also

TDBXDelegateDriver (see page 10464)

55.1.24.1 TDBXDelegateItem Members

The following tables list the members exposed by TDBXDelegateItem.

Public Fields

	Name	Description
	FName (see page 10469)	This is FName, a member of class TDBXDelegateItem.
	FNext (see page 10469)	This is FNext, a member of class TDBXDelegateItem.
	FProperties (see page 10469)	This is FProperties, a member of class TDBXDelegateItem.

Public Methods

	Name	Description
	Destroy (see page 10468)	Destroys an instance of TDBXDelegateItem (see page 10468).

55.1.24.2 TDBXDelegateItem.Destroy Destructor

Destroys an instance of TDBXDelegateItem (see page 10468).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXDelegateItem();
```

Description

This destructor destroys an instance of TDBXDelegateItem (see page 10468).

55.1.24.3 TDBXDelegateItem Fields

The fields of the TDBXDelegateItem class are listed here.

Public Fields

	Name	Description
	FName (see page 10469)	This is FName, a member of class TDBXDelegateItem.
	FNext (see page 10469)	This is FNext, a member of class TDBXDelegateItem.
	FProperties (see page 10469)	This is FProperties, a member of class TDBXDelegateItem.

55.1.24.3.1 TDBXDelegateItem.FName Field

Pascal

```
FName: WideString;
```

C++

```
BSTR FName;
```

Description

This is FName, a member of class TDBXDelegateItem.

55.1.24.3.2 TDBXDelegateItem.FNext Field

Pascal

```
FNext: TDBXDelegateItem;
```

C++

```
TDBXDelegateItem FNext;
```

Description

This is FNext, a member of class TDBXDelegateItem.

55.1.24.3.3 TDBXDelegateItem.FProperties Field

Pascal

```
FProperties: TDBXProperties;
```

C++

```
TDBXProperties FProperties;
```

Description

This is FProperties, a member of class TDBXDelegateItem.

55.1.25 TDBXDoubleValue Class

Implementation of double-precision floating point values.

Class Hierarchy

Pascal

```
TDBXDoubleValue = class(TDBXWritableValue);
```

C++

```
class TDBXDoubleValue : public TDBXWritableValue;
```

File

DBXCommon

Description

Implementation of TDBXWritableValue (see page 10661) for TDBXDataTypes.DoubleType values.

55.1.25.1 TDBXDoubleValue Members

The following tables list the members exposed by TDBXDoubleValue.

Protected Methods

	Name	Description
	Create (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
	Create (see page 10662)	Creates an instance of TDBXWritableValue (see page 10661)
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.
V	SetPendingValue (see page 10666)	This is SetPendingValue, a member of class TDBXWritableValue.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Methods

	Name	Description
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
V	Destroy (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
V	Destroy (see page 10663)	Destroys a TDBXWritableValue (see page 10661) instance.
V	GetAnsiString (see page 10629)	Gets the string value for ANSI strings.
V	GetBcd (see page 10629)	Gets the BCD value for binary-compressed decimal data types.
V	GetBoolean (see page 10629)	Gets the boolean value for boolean data types.
V	GetBytes (see page 10629)	Gets the byte array value for byte array and blob types.
V	GetDate (see page 10630)	Gets the date value for date data types.
V	GetDouble (see page 10471)	Gets the double floating point value for double data types.
V	GetInt16 (see page 10631)	Gets the smallInt value for 16-byte integer data types.
V	GetInt32 (see page 10631)	Gets the 32-byte integer value for 32-byte integer data types.
V	GetInt64 (see page 10631)	Gets the 64-byte integer value for 64-byte integer data types.
V	GetTime (see page 10631)	Gets the time value for time data types.
V	GetTimeStamp (see page 10632)	Gets the timestamp value for timestamp data types.
V	GetValueSize (see page 10632)	Gets the size in bytes for values.
V	GetWideString (see page 10632)	Gets the wide string value for wide string data types.
V	IsNull (see page 10471)	Returns true if the value is **null**.
V	SetAnsiString (see page 10663)	Sets the string value for ANSI strings.
V	SetBcd (see page 10664)	Sets the BCD value for binary-compressed decimal data types.
V	SetBoolean (see page 10664)	Sets the boolean value for boolean data types..
V	SetDate (see page 10664)	Sets the date value for date data types..
V	SetDouble (see page 10472)	Sets the double floating point value for double data types.

	SetDynamicBytes (see page 10665)	Sets the dynamic array value for byte array and blob types.
	SetInt16 (see page 10665)	Sets the the smallInt value for 16-byte integer data types.
	SetInt32 (see page 10665)	Sets the 32-byte integer value for 32-byte integer data types.
	SetInt64 (see page 10665)	Sets the 64-byte integer value for 64-byte integer data types.
	SetNull (see page 10666)	Sets a writable value of any type to null.
	SetStaticBytes (see page 10666)	Sets the open array value for byte array and blob types
	SetTime (see page 10666)	Sets the time value for time data types.
	SetTimeStamp (see page 10667)	Sets the timestamp value for timestamp data types.
	SetWideString (see page 10667)	Sets the wide string value for wide string data types.

Public Properties

	Name	Description
	ValueType (see page 10633)	Gets the value type for this value.

55.1.25.2 TDBXDoubleValue Methods

The methods of the TDBXDoubleValue class are listed here.

Public Methods

	Name	Description
	GetDouble (see page 10471)	Gets the double floating point value for double data types.
	IsNull (see page 10471)	Returns true if the value is **null**.
	SetDouble (see page 10472)	Sets the double floating point value for double data types.

55.1.25.2.1 TDBXDoubleValue.GetDouble Method

Gets the double floating point value for double data types.

Pascal

```
function GetDouble: Double; override;
```

C++

```
virtual __fastcall Double GetDouble();
```

Description

Gets the double floating point value for TDBXValue (see page 10626) instances with a data type of DoubleType. Throws a TDBXError (see page 10486) exception if IsNull (see page 10471) is true.

55.1.25.2.2 TDBXDoubleValue.IsNull Method

Returns true if the value is **null**.

Pascal

```
function IsNull: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsNull();
```

Description

Returns true if the value is **null**. This method should be called before calling any of the Get methods. If the value is null, the Get methods will throw TDBXError (see page 10486) exception if they are called.

55.1.25.2.3 TDBXDoubleValue.SetDouble Method

Sets the double floating point value for double data types.

Pascal

```
procedure SetDouble(Value: Double); override;
```

C++

```
virtual __fastcall SetDouble(Double Value);
```

Description

Sets the double floating point value for TDBXValue (see page 10626) instances with a data type of DoubleType.

55.1.26 TDBXDriver Class

Driver class for DBX framework.

Class Hierarchy

DBXCommon.TDBXDriver

Pascal

```
TDBXDriver = class;
```

C++

```
class TDBXDriver;
```

File

DBXCommon

Description

Driver class for DBX framework. Loads client libraries if needed.

The DBX framework manages reference counts for all drivers that are loaded. When all TDBXConnection (see page 10398) and TDBXDriver instances are freed, the driver is unloaded. So if an application holds on to an instance of TDBXConnection (see page 10398) or TDBXDriver, that driver will not be unloaded.

55.1.26.1 TDBXDriver Members

The following tables list the members exposed by TDBXDriver.

Protected Fields

	Name	Description
	FDBXContext (see page 10473)	This is FDBXContext, a member of class TDBXDriver.
	FDriverLoader (see page 10474)	This is FDriverLoader, a member of class TDBXDriver.

Protected Methods

	Name	Description
A	Close (see page 10474)	This is Close, a member of class TDBXDriver.
A	CreateConnection (see page 10475)	Create instance of TDBXConnection (see page 10398).
V	GetDriverName (see page 10475)	This is GetDriverName, a member of class TDBXDriver.

V	GetDriverProperties (see page 10475)	This is GetDriverProperties, a member of class TDBXDriver.
V	SetDriverName (see page 10476)	This is SetDriverName, a member of class TDBXDriver.
V	SetDriverProperties (see page 10476)	This is SetDriverProperties, a member of class TDBXDriver.

Public Methods

	Name	Description
V	Destroy (see page 10473)	This is Destroy, a member of class TDBXDriver.
A	GetDriverVersion (see page 10475)	Returns version information for this driver.

Public Properties

	Name	Description
R	DriverName (see page 10476)	A unique name for this driver.
R	DriverProperties (see page 10477)	Driver-specific properties for a particular driver.

Strict protected Methods

	Name	Description
	CacheUntilFinalization (see page 10474)	This is CacheUntilFinalization, a member of class TDBXDriver.

55.1.26.2 TDBXDriver.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXDriver();
```

Description

This is Destroy, a member of class TDBXDriver.

55.1.26.3 TDBXDriver Fields

The fields of the TDBXDriver class are listed here.

Protected Fields

	Name	Description
	FDBXContext (see page 10473)	This is FDBXContext, a member of class TDBXDriver.
	FDriverLoader (see page 10474)	This is FDriverLoader, a member of class TDBXDriver.

55.1.26.3.1 TDBXDriver.FDBXContext Field

Pascal

```
FDBXContext: TDBXContext;
```

C++

```
TDBXContext FDBXContext;
```

Description

This is FDBXContext, a member of class TDBXDriver.

55.1.26.3.2 TDBXDriver.FDriverLoader Field

Pascal

```
FDriverLoader: TDBXDriverLoader;
```

C++

```
TDBXDriverLoader FDriverLoader;
```

Description

This is FDriverLoader, a member of class TDBXDriver.

55.1.26.4 TDBXDriver Methods

The methods of the TDBXDriver class are listed here.

Protected Methods

	Name	Description
	Close (see page 10474)	This is Close, a member of class TDBXDriver.
	CreateConnection (see page 10475)	Create instance of TDBXConnection (see page 10398).
	GetDriverName (see page 10475)	This is GetDriverName, a member of class TDBXDriver.
	GetDriverProperties (see page 10475)	This is GetDriverProperties, a member of class TDBXDriver.
	SetDriverName (see page 10476)	This is SetDriverName, a member of class TDBXDriver.
	SetDriverProperties (see page 10476)	This is SetDriverProperties, a member of class TDBXDriver.

Public Methods

	Name	Description
	GetDriverVersion (see page 10475)	Returns version information for this driver.

Strict protected Methods

	Name	Description
	CacheUntilFinalization (see page 10474)	This is CacheUntilFinalization, a member of class TDBXDriver.

55.1.26.4.1 TDBXDriver.CacheUntilFinalization Method

Pascal

```
procedure CacheUntilFinalization;
```

C++

```
__fastcall CacheUntilFinalization();
```

Description

This is CacheUntilFinalization, a member of class TDBXDriver.

55.1.26.4.2 TDBXDriver.Close Method

Pascal

```
procedure Close; virtual; abstract;
```

C++

```
virtual __fastcall Close() = 0;
```

Description

This is Close, a member of class TDBXDriver.

55.1.26.4.3 TDBXDriver.CreateConnection Method

Create instance of TDBXConnection (see page 10398).

Pascal

```
function CreateConnection(ConnectionBuilder: TDBXConnectionBuilder): TDBXConnection; virtual;
abstract;
```

C++

```
virtual __fastcall TDBXConnection CreateConnection(TDBXConnectionBuilder ConnectionBuilder) =
0;
```

Description

Used by TDBXConnectionFactory (see page 10422) to create instances of TDBXConnection (see page 10398).

See Also

TDBXConnection (see page 10398)

TDBXConnectionFactory (see page 10422)

55.1.26.4.4 TDBXDriver.GetDriverName Method

Pascal

```
function GetDriverName: WideString; virtual;
```

C++

```
virtual __fastcall BSTR GetDriverName();
```

Description

This is GetDriverName, a member of class TDBXDriver.

55.1.26.4.5 TDBXDriver.GetDriverProperties Method

Pascal

```
function GetDriverProperties: TDBXProperties; virtual;
```

C++

```
virtual __fastcall TDBXProperties GetDriverProperties();
```

Description

This is GetDriverProperties, a member of class TDBXDriver.

55.1.26.4.6 TDBXDriver.GetDriverVersion Method

Returns version information for this driver.

Pascal

```
function GetDriverVersion: WideString; virtual; abstract;
```

C++

```
virtual __fastcall BSTR GetDriverVersion() = 0;
```

Description

Use GetDriverVersion to return version information for an instance of TDBXDriver (see page 10472).

55.1.26.4.7 TDBXDriver.SetDriverName Method

Pascal

```
procedure SetDriverName(const DriverName: WideString); virtual;
```

C++

```
virtual __fastcall SetDriverName(const BSTR DriverName);
```

Description

This is SetDriverName, a member of class TDBXDriver.

55.1.26.4.8 TDBXDriver.SetDriverProperties Method

Pascal

```
procedure SetDriverProperties(DriverProperties: TDBXProperties); virtual;
```

C++

```
virtual __fastcall SetDriverProperties(TDBXProperties DriverProperties);
```

Description

This is SetDriverProperties, a member of class TDBXDriver.

55.1.26.5 TDBXDriver Properties

The properties of the TDBXDriver class are listed here.

Public Properties

	Name	Description
R	DriverName (see page 10476)	A unique name for this driver.
R	DriverProperties (see page 10477)	Driver-specific properties for a particular driver.

55.1.26.5.1 TDBXDriver.DriverName Property

A unique name for this driver.

Pascal

```
property DriverName: WideString;
```

C++

```
__property BSTR DriverName;
```

Description

A unique name for the driver referenced in an instance of TDBXDriver (see page 10472).

55.1.26.5.2 TDBXDriver.DriverProperties Property

Driver-specific properties for a particular driver.

Pascal

```
property DriverProperties: TDBXProperties;
```

C++

```
__property TDBXProperties DriverProperties;
```

Description

Driver-specific properties for an instance of TDBXDriver (see page 10472).

55.1.27 TDBXDriverEx Class

Class Hierarchy

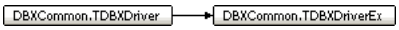

Pascal

```
TDBXDriverEx = class(TDBXDriver);
```

C++

```
class TDBXDriverEx : public TDBXDriver;
```

File

DBXCommon

Description

This is class DBXCommon.TDBXDriverEx.

55.1.27.1 TDBXDriverEx Members

The following tables list the members exposed by TDBXDriverEx.

Protected Fields

	Name	Description
	FCommandFactories (see page 10479)	This is FCommandFactories, a member of class TDBXDriverEx.
	FDBXContext (see page 10473)	This is FDBXContext, a member of class TDBXDriver.
	FDriverLoader (see page 10474)	This is FDriverLoader, a member of class TDBXDriver.

Protected Methods

	Name	Description
	AddCommandFactory (see page 10479)	This is AddCommandFactory, a member of class TDBXDriverEx.
A	Close (see page 10474)	This is Close, a member of class TDBXDriver.
A	CreateConnection (see page 10475)	Create instance of TDBXConnection (see page 10398).
V	CreateMorphCommand (see page 10479)	This is CreateMorphCommand, a member of class TDBXDriverEx.
V	GetDriverName (see page 10475)	This is GetDriverName, a member of class TDBXDriver.
V	GetDriverProperties (see page 10475)	This is GetDriverProperties, a member of class TDBXDriver.
	LoadMetaDataCommandFactory (see page 10479)	This is LoadMetaDataCommandFactory, a member of class TDBXDriverEx.

	SetDriverName (see page 10476)	This is SetDriverName, a member of class TDBXDriver.
	SetDriverProperties (see page 10476)	This is SetDriverProperties, a member of class TDBXDriver.

Public Methods

	Name	Description
	Create (see page 10478)	This is Create, a member of class TDBXDriverEx.
	Destroy (see page 10478)	This is Destroy, a member of class TDBXDriverEx.
	GetDriverVersion (see page 10475)	Returns version information for this driver.

Public Properties

	Name	Description
	DriverName (see page 10476)	A unique name for this driver.
	DriverProperties (see page 10477)	Driver-specific properties for a particular driver.

Strict protected Methods

	Name	Description
	CacheUntilFinalization (see page 10474)	This is CacheUntilFinalization, a member of class TDBXDriver.

55.1.27.2 TDBXDriverEx.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TDBXDriverEx();
```

Description

This is Create, a member of class TDBXDriverEx.

55.1.27.3 TDBXDriverEx.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXDriverEx();
```

Description

This is Destroy, a member of class TDBXDriverEx.

55.1.27.4 TDBXDriverEx Fields

The fields of the TDBXDriverEx class are listed here.

Protected Fields

	Name	Description
	FCommandFactories (see page 10479)	This is FCommandFactories, a member of class TDBXDriverEx.

55.1.27.4.1 TDBXDriverEx.FCommandFactories Field

Pascal

```
FCommandFactories: TWidestrings;
```

C++

```
TWidestrings FCommandFactories;
```

Description

This is FCommandFactories, a member of class TDBXDriverEx.

55.1.27.5 TDBXDriverEx Methods

The methods of the TDBXDriverEx class are listed here.

Protected Methods

	Name	Description
	AddCommandFactory (see page 10479)	This is AddCommandFactory, a member of class TDBXDriverEx.
V	CreateMorphCommand (see page 10479)	This is CreateMorphCommand, a member of class TDBXDriverEx.
	LoadMetaDataCommandFactory (see page 10479)	This is LoadMetaDataCommandFactory, a member of class TDBXDriverEx.

55.1.27.5.1 TDBXDriverEx.AddCommandFactory Method

Pascal

```
procedure AddCommandFactory(Name: WideString; OnCreateCommand: TDBXCreateCommandEvent);
```

C++

```
__fastcall AddCommandFactory(BSTR Name, TDBXCreateCommandEvent OnCreateCommand);
```

Description

This is AddCommandFactory, a member of class TDBXDriverEx.

55.1.27.5.2 TDBXDriverEx.CreateMorphCommand Method

Pascal

```
function CreateMorphCommand(DbxContext: TDBXContext; Connection: TDBXConnection;
MorphicCommand: TDBXCommand): TDBXCommand; virtual;
```

C++

```
virtual __fastcall TDBXCommand CreateMorphCommand(TDBXContext DbxContext, TDBXConnection
Connection, TDBXCommand MorphicCommand);
```

Description

This is CreateMorphCommand, a member of class TDBXDriverEx.

55.1.27.5.3 TDBXDriverEx.LoadMetaDataCommandFactory Method

Pascal

```
procedure LoadMetaDataCommandFactory(const MetaDataCommandFactoryClassName: WideString; const
MetaDataCommandFactoryPackageName: WideString);
```

C++

```
__fastcall LoadMetaDataCommandFactory(const BSTR MetaDataCommandFactoryClassName, const BSTR
MetaDataCommandFactoryPackageName);
```

Description

This is LoadMetaDataCommandFactory, a member of class TDBXDriverEx.

55.1.28 TDBXDriverHelp Class

Used by driver implementors to access private or protected API.

Class Hierarchy

DBXCommon.TDBXDriverHelp

Pascal

```
TDBXDriverHelp = class;
```

C++

```
class TDBXDriverHelp;
```

File

DBXCommon

Description

This class is used only by driver implementors to access private or protected constructors, procedures or functions.

55.1.28.1 TDBXDriverHelp Members

The following tables list the members exposed by TDBXDriverHelp.

Public Methods

	Name	Description
	CreateConnectionBuilder (see page 10481)	This is CreateConnectionBuilder, a member of class TDBXDriverHelp.
	CreateTDBXContext (see page 10481)	This is CreateTDBXContext, a member of class TDBXDriverHelp.
	CreateTDBXParameter (see page 10481)	This is CreateTDBXParameter, a member of class TDBXDriverHelp.
	CreateTDBXProperties (see page 10481)	This is CreateTDBXProperties, a member of class TDBXDriverHelp.
	CreateTDBXValueType (see page 10481)	This is CreateTDBXValueType, a member of class TDBXDriverHelp.

55.1.28.2 TDBXDriverHelp Methods

The methods of the TDBXDriverHelp class are listed here.

Public Methods

	Name	Description
	CreateConnectionBuilder (see page 10481)	This is CreateConnectionBuilder, a member of class TDBXDriverHelp.
	CreateTDBXContext (see page 10481)	This is CreateTDBXContext, a member of class TDBXDriverHelp.
	CreateTDBXParameter (see page 10481)	This is CreateTDBXParameter, a member of class TDBXDriverHelp.
	CreateTDBXProperties (see page 10481)	This is CreateTDBXProperties, a member of class TDBXDriverHelp.
	CreateTDBXValueType (see page 10481)	This is CreateTDBXValueType, a member of class TDBXDriverHelp.

55.1.28.2.1 TDBXDriverHelp.CreateConnectionBuilder Method

Pascal

```
class function CreateConnectionBuilder(ConnectionBuilder: TDBXConnectionBuilder):
TDBXConnectionBuilder; static;
```

C++

```
static __fastcall TDBXConnectionBuilder CreateConnectionBuilder(TDBXConnectionBuilder
ConnectionBuilder);
```

Description

This is CreateConnectionBuilder, a member of class TDBXDriverHelp.

55.1.28.2.2 TDBXDriverHelp.CreateTDBXContext Method

Pascal

```
class function CreateTDBXContext: TDBXContext; static;
```

C++

```
static __fastcall TDBXContext CreateTDBXContext();
```

Description

This is CreateTDBXContext, a member of class TDBXDriverHelp.

55.1.28.2.3 TDBXDriverHelp.CreateTDBXParameter Method

Pascal

```
class function CreateTDBXParameter(DBXContext: TDBXContext): TDBXParameter; static;
```

C++

```
static __fastcall TDBXParameter CreateTDBXParameter(TDBXContext DBXContext);
```

Description

This is CreateTDBXParameter, a member of class TDBXDriverHelp.

55.1.28.2.4 TDBXDriverHelp.CreateTDBXProperties Method

Pascal

```
class function CreateTDBXProperties(DBXContext: TDBXContext): TDBXProperties; static;
```

C++

```
static __fastcall TDBXProperties CreateTDBXProperties(TDBXContext DBXContext);
```

Description

This is CreateTDBXProperties, a member of class TDBXDriverHelp.

55.1.28.2.5 TDBXDriverHelp.CreateTDBXValueType Method

Pascal

```
class function CreateTDBXValueType(DBXContext: TDBXContext): TDBXValueType; static;
```

C++

```
static __fastcall TDBXValueType CreateTDBXValueType(TDBXContext DBXContext);
```

Description

This is CreateTDBXValueType, a member of class TDBXDriverHelp.

55.1.29 TDBXDriverHelpEx Class

Class Hierarchy

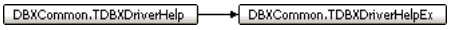

Pascal

```
TDBXDriverHelpEx = class(TDBXDriverHelp);
```

C++

```
class TDBXDriverHelpEx : public TDBXDriverHelp;
```

File

DBXCommon

Description

This is class DBXCommon.TDBXDriverHelpEx.

55.1.29.1 TDBXDriverHelpEx Members

The following tables list the members exposed by TDBXDriverHelpEx.

Public Methods

	Name	Description
	CreateConnectionBuilder (see page 10481)	This is CreateConnectionBuilder, a member of class TDBXDriverHelp.
	CreateTDBXContext (see page 10481)	This is CreateTDBXContext, a member of class TDBXDriverHelp.
	CreateTDBXParameter (see page 10481)	This is CreateTDBXParameter, a member of class TDBXDriverHelp.
	CreateTDBXProperties (see page 10481)	This is CreateTDBXProperties, a member of class TDBXDriverHelp.
	CreateTDBXValueType (see page 10483)	This is CreateTDBXValueType, a member of class TDBXDriverHelpEx.
	GetStreamReader (see page 10483)	This is GetStreamReader, a member of class TDBXDriverHelpEx.
	IsReadOnlyValueType (see page 10483)	This is IsReadOnlyValueType, a member of class TDBXDriverHelpEx.
	UpdateParameterType (see page 10483)	This is UpdateParameterType, a member of class TDBXDriverHelpEx.

55.1.29.2 TDBXDriverHelpEx Methods

The methods of the TDBXDriverHelpEx class are listed here.

Public Methods

	Name	Description
	CreateTDBXValueType (see page 10483)	This is CreateTDBXValueType, a member of class TDBXDriverHelpEx.
	GetStreamReader (see page 10483)	This is GetStreamReader, a member of class TDBXDriverHelpEx.
	IsReadOnlyValueType (see page 10483)	This is IsReadOnlyValueType, a member of class TDBXDriverHelpEx.
	UpdateParameterType (see page 10483)	This is UpdateParameterType, a member of class TDBXDriverHelpEx.

55.1.29.2.1 TDBXDriverHelpEx.CreateTDBXValueType Method

Pascal

```
class function CreateTDBXValueType(DBXContext: TDBXContext; DBXRow: TDBXRow): TDBXValueType;
overload; static;
```

C++

```
static __fastcall TDBXValueType CreateTDBXValueType(TDBXContext DBXContext, TDBXRow DBXRow);
```

Description

This is CreateTDBXValueType, a member of class TDBXDriverHelpEx.

55.1.29.2.2 TDBXDriverHelpEx.GetStreamReader Method

Pascal

```
class function GetStreamReader(Value: TDBXStreamValue): TDBXStreamReader; overload; static;
```

C++

```
static __fastcall TDBXStreamReader GetStreamReader(TDBXStreamValue Value);
```

Description

This is GetStreamReader, a member of class TDBXDriverHelpEx.

55.1.29.2.3 TDBXDriverHelpEx.IsReadOnlyValueType Method

Pascal

```
class function IsReadOnlyValueType(ValueType: TDBXValueType): Boolean; static;
```

C++

```
static __fastcall Boolean IsReadOnlyValueType(TDBXValueType ValueType);
```

Description

This is IsReadOnlyValueType, a member of class TDBXDriverHelpEx.

55.1.29.2.4 TDBXDriverHelpEx.UpdateParameterType Method

Pascal

```
class procedure UpdateParameterType(Parameter: TDBXParameter);
```

C++

```
__fastcall UpdateParameterType(TDBXParameter Parameter);
```

Description

This is UpdateParameterType, a member of class TDBXDriverHelpEx.

55.1.30 TDBXDriverLoader Class

Collection of parameters needed when loading a driver.

Class Hierarchy

Pascal

```
TDBXDriverLoader = class(TClassRegistryObject);
```

C++

```
class TDBXDriverLoader : public TClassRegistryObject;
```

File

DBXCommon

Description

This class is used by driver implementors. It provides a collection of parameters needed when loading a driver. By collecting them in a single record, parameters can be added without breaking older drivers. TDBXDriverLoader implementations can be registered with the TDBXDriverRegistry (see page 10485) to load a driver by a string based name.

See Also

TDBXDriverRegistry (see page 10485)

55.1.30.1 TDBXDriverLoader Members

The following tables list the members exposed by TDBXDriverLoader.

Protected Fields

	Name	Description
	FLoaderName (see page 10484)	This is FLoaderName, a member of class TDBXDriverLoader.

Public Methods

	Name	Description
	Load (see page 10485)	Load driver with parameters.

55.1.30.2 TDBXDriverLoader Fields

The fields of the TDBXDriverLoader class are listed here.

Protected Fields

	Name	Description
	FLoaderName (see page 10484)	This is FLoaderName, a member of class TDBXDriverLoader.

55.1.30.2.1 TDBXDriverLoader.FLoaderName Field

Pascal

```
FLoaderName: WideString;
```

C++

```
BSTR FLoaderName;
```

Description

This is FLoaderName, a member of class TDBXDriverLoader.

55.1.30.3 TDBXDriverLoader Methods

The methods of the TDBXDriverLoader class are listed here.

Public Methods

	Name	Description
	Load (see page 10485)	Load driver with parameters.

55.1.30.3.1 TDBXDriverLoader.Load Method

Load driver with parameters.

Pascal

```
function Load(DriverDef: TDBXDriverDef): TDBXDriver; virtual; abstract;
```

C++

```
virtual __fastcall TDBXDriver Load(TDBXDriverDef DriverDef) = 0;
```

Description

This method loads a driver using the provided **DriverDef** parameters.

55.1.31 TDBXDriverRegistry Class

Registry (see page 18633) for TDBXDriverLoader (see page 10483) keyed by driver name.

Class Hierarchy

DBXCommon.TDBXDriverRegistry

Pascal

```
TDBXDriverRegistry = class;
```

C++

```
class TDBXDriverRegistry;
```

File

DBXCommon

Description

This class is used by driver implementors. It provides a registry of TDBXDriverLoader (see page 10483) instances that is keyed by a driver name string.

See Also

TDBXDriverLoader (see page 10483)

55.1.31.1 TDBXDriverRegistry Members

The following tables list the members exposed by TDBXDriverRegistry.

Public Methods

	Name	Description
	Create (see page 10486)	Creates an instance of TDBXDriverRegistry (see page 10485)
V	Destroy (see page 10486)	Destroys a TDBXDriverRegistry (see page 10485) instance.

55.1.31.2 TDBXDriverRegistry.Create Constructor

Creates an instance of TDBXDriverRegistry (see page 10485)

Pascal

```
constructor Create;
```

C++

```
__fastcall TDBXDriverRegistry();
```

Description

This constructor creates and initializes an instance of TDBXDriverRegistry (see page 10485).

See Also

Destroy (see page 10486)

55.1.31.3 TDBXDriverRegistry.Destroy Destructor

Destroys a TDBXDriverRegistry (see page 10485) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXDriverRegistry();
```

Description

This destructor destroys an instance of TDBXDriverRegistry (see page 10485).

See Also

Create (see page 10486)

55.1.32 TDBXError Class

Exception type for all dbExpress framework (TDBX) errors.

Class Hierarchy

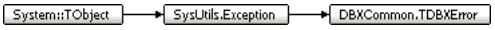

Pascal

```
TDBXError = class(Exception);
```

C++

```
class TDBXError : public Exception;
```

File

DBXCommon

Description

In general, methods in the dbExpress framework (TDBX) raise exceptions rather than return error codes. TDBXError is the exception class for all TDBX errors.

See Also

TDBXErrorCodes (see page 10489)

55.1.32.1 TDBXError Members

The following tables list the members exposed by TDBXError.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 10488)	Create a TDBXError (see page 10486) instance.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ErrorCodeToString (see page 10488)	Convert known error codes to String.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	ErrorCode (see page 10489)	TDBXErrorCode (see page 10670) for an exception
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 10489)	Explanation of exception

55.1.32.2 TDBXError.Create Constructor

Create a TDBXError (see page 10486) instance.

Pascal

```
constructor Create(ErrorCode: TDBXErrorCode; const ErrorMessage: WideString);
```

C++

```
__fastcall TDBXError(TDBXErrorCode ErrorCode, const BSTR ErrorMessage);
```

Description

Create creates and initializes an instance of the TDBXErrorclass, the exception class for the dbExpress framework. New exceptions you create should inherit from TDBXError (see page 10486).

See Also

TDBXError (see page 10486)

55.1.32.3 TDBXError Methods

The methods of the TDBXError class are listed here.

Public Methods

	Name	Description
	ErrorCodeToString (see page 10488)	Convert known error codes to String.

55.1.32.3.1 TDBXError.ErrorCodeToString Method

Convert known error codes to String.

Pascal

```
class function ErrorCodeToString(ErrorCode: TDBXErrorCode): WideString; static;
```

C++

```
static __fastcall BSTR ErrorCodeToString(TDBXErrorCode ErrorCode);
```

Description

ErrorCodeToString converts known error codes in TDBXErrorCode (see page 10670) to a `String`.

See Also

TDBXErrorCode (see page 10670)

55.1.32.4 TDBXError Properties

The properties of the TDBXError class are listed here.

Public Properties

	Name	Description
R	ErrorCode (see page 10489)	TDBXErrorCode (see page 10670) for an exception
R	Message (see page 10489)	Explanation of exception

55.1.32.4.1 TDBXError.ErrorCode Property

TDBXErrorCode (see page 10670) for an exception

Pascal

```
property ErrorCode: TDBXErrorCode;
```

C++

```
__property TDBXErrorCode ErrorCode;
```

Description

Use ErrorCode to get the TDBXErrorCode (see page 10670) value for an exception. See the `const` section of TDBXErrorCodes (see page 10489) class for a list of common error codes with explanations.

See Also

TDBXError (see page 10486)

TDBXErrorCode (see page 10670)

55.1.32.4.2 TDBXError.Message Property

Explanation of exception

Pascal

```
property Message: WideString;
```

C++

```
__property BSTR Message;
```

Description

This property contains text explaining the reason an exception occurred. Also see the ErrorCode (see page 10489) property for exception information.

See Also

TDBXError (see page 10486)

ErrorCode (see page 10489)

55.1.33 TDBXErrorCodes Class

Error codes for dbExpress framework (TDBX) exceptions

Class Hierarchy

DBXCommon.TDBXErrorCodes

Pascal

```
TDBXErrorCodes = class;
```

C++

```
class TDBXErrorCodes;
```

File

DBXCommon

Description

In general, methods in the dbExpress framework (TDBX) raise exceptions rather than return error codes. This class encapsulates the error codes for the standard exceptions.

This table contains the standard error codes for TDBX exceptions.

Constant	Description
None (see page 10498)	Successful completion.
Warning (see page 10500)	Non fatal warning. Currently unused.
NoMemory (see page 10498)	Insufficient memory to complete the operation.
UnsupportedFieldType (see page 10500)	Field type is not supported by this driver.
InvalidHandle (see page 10495)	Unexpected internal error. Occurs if attempting to use a TDBX Object such as a connection, command, or reader that is already closed.
NotSupported (see page 10499)	Unsupported feature for the driver in use. Typically caused by unsupported metadata related requests.
InvalidTime (see page 10496)	Invalid value for a TDBXDataTypes.TIME field.
InvalidType (see page 10497)	Invalid value read operation attempted for a TDBXReader (see page 10576) or a TDBXParameter (see page 10545) field. Use a get method of the correct type for this field.
InvalidOrdinal (see page 10495)	Ordinal used to access a TDBXReader (see page 10576) column or a TDBXParameter (see page 10545) is out of range.
InvalidParameter (see page 10496)	Invalid use of a parameter. Common causes of this error include: 1) Parameter already bound. 2) TDBXParameterDirections (see page 10549) setting used is not supported by this driver.
EOF (see page 10494)	Used internally by DBXDynalink drivers to indicate that a TDBXReader (see page 10576) has no more rows.
ParameterNotSet (see page 10499)	One or more parameters for a TDBXCommand (see page 10378) has not been set.
InvalidUserOrPassword (see page 10497)	User name or password provided for a TDBXConnection (see page 10398) is not valid.
InvalidPrecision (see page 10496)	Attempt to set a TDBXParameter (see page 10545) with an invalid precision or read a value with a TDBXDataTypes (see page 10447) value that is not large enough.
InvalidLength (see page 10495)	Unexpected error. Insufficient storage provided to retrieve a TDBXParameter (see page 10545) value.

InvalidIsolationLevel (see page 10495)	TDBXIsolations (see page 10513) level used is not valid for this driver.
InvalidTransactionId (see page 10497)	Unexpected internal error. Transaction id is either expired or invalid.
DuplicateTransactionId (see page 10494)	Unexpected internal error. Transaction id is already in use by an active transaction.
DriverRestricted (see page 10494)	This driver is not available for the Delphi SKU purchased.
TransactionActive (see page 10500)	Operation cannot be completed with a transaction active for the TDBXConnection (see page 10398).
MultipleTransactionNotEnabled (see page 10498)	Support for multiple transactions is not enabled for this driver.
ConnectionFailed (see page 10493)	TDBXConnection.Open connect operation failed.
DriverInitFailed (see page 10493)	Driver could not be properly initialized. Client library may be missing, installed improperly, have the wrong version.
OptimisticLockFailed (see page 10499)	Optimistic lock failed. Oracle driver return this error: "OCI-21700: object does not exist or is marked for delete".
InvalidReference (see page 10496)	Unexpected internal error accessing an Oracle REF object.
NoTable (see page 10499)	Unexpected internal error accessing an Oracle REF object.
MissingParameterMarker (see page 10497)	Parameterized query is missing a '?' parameter marker.
NotImplemented (see page 10499)	Feature not implemented.
DriverIncompatible (see page 10493)	Deprecated. Driver is not compatible with version of VCL data access components in use.
InvalidArgument (see page 10494)	Argument used for an operation is invalid.
InvalidOperation (see page 10495)	Attempted operation is invalid.
NoData (see page 10498)	Used internally by DBXDynalink drivers. Not an error. Indicates either there are no more rows for a TDBXReader (see page 10576) or there is no more data when reading blob fields.
MaxCommonErrors (see page 10497)	Max common errors.
VendorError (see page 10500)	Vendor specific error.

See Also

TDBXError (see page 10486)

55.1.33.1 TDBXErrorCodes Members

The following tables list the members exposed by TDBXErrorCodes.

Public Constants

	Name	Description
	ConnectionFailed (see page 10493)	This is ConnectionFailed, a member of class TDBXErrorCodes.
	DriverIncompatible (see page 10493)	This is DriverIncompatible, a member of class TDBXErrorCodes.
	DriverInitFailed (see page 10493)	This is DriverInitFailed, a member of class TDBXErrorCodes.
	DriverRestricted (see page 10494)	This is DriverRestricted, a member of class TDBXErrorCodes.
	DuplicateTransactionId (see page 10494)	This is DuplicateTransactionId, a member of class TDBXErrorCodes.
	EOF (see page 10494)	This is EOF, a member of class TDBXErrorCodes.
	InvalidArgument (see page 10494)	This is InvalidArgument, a member of class TDBXErrorCodes.
	InvalidHandle (see page 10495)	This is InvalidHandle, a member of class TDBXErrorCodes.
	InvalidIsolationLevel (see page 10495)	This is InvalidIsolationLevel, a member of class TDBXErrorCodes.
	InvalidLength (see page 10495)	This is InvalidLength, a member of class TDBXErrorCodes.
	InvalidOperation (see page 10495)	This is InvalidOperation, a member of class TDBXErrorCodes.
	InvalidOrdinal (see page 10495)	This is InvalidOrdinal, a member of class TDBXErrorCodes.
	InvalidParameter (see page 10496)	This is InvalidParameter, a member of class TDBXErrorCodes.
	InvalidPrecision (see page 10496)	This is InvalidPrecision, a member of class TDBXErrorCodes.
	InvalidReference (see page 10496)	This is InvalidReference, a member of class TDBXErrorCodes.
	InvalidTime (see page 10496)	This is InvalidTime, a member of class TDBXErrorCodes.
	InvalidTransactionId (see page 10497)	This is InvalidTransactionId, a member of class TDBXErrorCodes.
	InvalidType (see page 10497)	This is InvalidType, a member of class TDBXErrorCodes.
	InvalidUserOrPassword (see page 10497)	This is InvalidUserOrPassword, a member of class TDBXErrorCodes.
	MaxCommonErrors (see page 10497)	This is MaxCommonErrors, a member of class TDBXErrorCodes.
	MissingParameterMarker (see page 10497)	This is MissingParameterMarker, a member of class TDBXErrorCodes.
	MultipleTransactionNotEnabled (see page 10498)	This is MultipleTransactionNotEnabled, a member of class TDBXErrorCodes.
	NoData (see page 10498)	This is NoData, a member of class TDBXErrorCodes.
	NoMemory (see page 10498)	This is NoMemory, a member of class TDBXErrorCodes.
	None (see page 10498)	This is None, a member of class TDBXErrorCodes.
	NoTable (see page 10499)	This is NoTable, a member of class TDBXErrorCodes.
	NotImplemented (see page 10499)	This is NotImplemented, a member of class TDBXErrorCodes.
	NotSupported (see page 10499)	This is NotSupported, a member of class TDBXErrorCodes.
	OptimisticLockFailed (see page 10499)	This is OptimisticLockFailed, a member of class TDBXErrorCodes.
	ParameterNotSet (see page 10499)	This is ParameterNotSet, a member of class TDBXErrorCodes.
	TransactionActive (see page 10500)	This is TransactionActive, a member of class TDBXErrorCodes.
	UnsupportedFieldType (see page 10500)	This is UnsupportedFieldType, a member of class TDBXErrorCodes.
	VendorError (see page 10500)	This is VendorError, a member of class TDBXErrorCodes.
	Warning (see page 10500)	This is Warning, a member of class TDBXErrorCodes.

55.1.33.2 TDBXErrorCodes Constants

The constants of the TDBXErrorCodes class are listed here.

Public Constants

	Name	Description
	ConnectionFailed (see page 10493)	This is ConnectionFailed, a member of class TDBXErrorCodes.
	DriverIncompatible (see page 10493)	This is DriverIncompatible, a member of class TDBXErrorCodes.
	DriverInitFailed (see page 10493)	This is DriverInitFailed, a member of class TDBXErrorCodes.
	DriverRestricted (see page 10494)	This is DriverRestricted, a member of class TDBXErrorCodes.
	DuplicateTransactionId (see page 10494)	This is DuplicateTransactionId, a member of class TDBXErrorCodes.
	EOF (see page 10494)	This is EOF, a member of class TDBXErrorCodes.
	InvalidArgument (see page 10494)	This is InvalidArgument, a member of class TDBXErrorCodes.
	InvalidHandle (see page 10495)	This is InvalidHandle, a member of class TDBXErrorCodes.
	InvalidIsolationLevel (see page 10495)	This is InvalidIsolationLevel, a member of class TDBXErrorCodes.
	InvalidLength (see page 10495)	This is InvalidLength, a member of class TDBXErrorCodes.

	InvalidOperation (see page 10495)	This is InvalidOperation, a member of class TDBXErrorCodes.
	InvalidOrdinal (see page 10495)	This is InvalidOrdinal, a member of class TDBXErrorCodes.
	InvalidParameter (see page 10496)	This is InvalidParameter, a member of class TDBXErrorCodes.
	InvalidPrecision (see page 10496)	This is InvalidPrecision, a member of class TDBXErrorCodes.
	InvalidReference (see page 10496)	This is InvalidReference, a member of class TDBXErrorCodes.
	InvalidTime (see page 10496)	This is InvalidTime, a member of class TDBXErrorCodes.
	InvalidTransactionId (see page 10497)	This is InvalidTransactionId, a member of class TDBXErrorCodes.
	InvalidType (see page 10497)	This is InvalidType, a member of class TDBXErrorCodes.
	InvalidUserOrPassword (see page 10497)	This is InvalidUserOrPassword, a member of class TDBXErrorCodes.
	MaxCommonErrors (see page 10497)	This is MaxCommonErrors, a member of class TDBXErrorCodes.
	MissingParameterMarker (see page 10497)	This is MissingParameterMarker, a member of class TDBXErrorCodes.
	MultipleTransactionNotEnabled (see page 10498)	This is MultipleTransactionNotEnabled, a member of class TDBXErrorCodes.
	NoData (see page 10498)	This is NoData, a member of class TDBXErrorCodes.
	NoMemory (see page 10498)	This is NoMemory, a member of class TDBXErrorCodes.
	None (see page 10498)	This is None, a member of class TDBXErrorCodes.
	NoTable (see page 10499)	This is NoTable, a member of class TDBXErrorCodes.
	NotImplemented (see page 10499)	This is NotImplemented, a member of class TDBXErrorCodes.
	NotSupported (see page 10499)	This is NotSupported, a member of class TDBXErrorCodes.
	OptimisticLockFailed (see page 10499)	This is OptimisticLockFailed, a member of class TDBXErrorCodes.
	ParameterNotSet (see page 10499)	This is ParameterNotSet, a member of class TDBXErrorCodes.
	TransactionActive (see page 10500)	This is TransactionActive, a member of class TDBXErrorCodes.
	UnsupportedFieldType (see page 10500)	This is UnsupportedFieldType, a member of class TDBXErrorCodes.
	VendorError (see page 10500)	This is VendorError, a member of class TDBXErrorCodes.
	Warning (see page 10500)	This is Warning, a member of class TDBXErrorCodes.

55.1.33.2.1 TDBXErrorCodes.ConnectionFailed Constant

Pascal

```
const ConnectionFailed = $0015;
```

C++

```
ConnectionFailed = 0x0015;
```

Description

This is ConnectionFailed, a member of class TDBXErrorCodes.

55.1.33.2.2 TDBXErrorCodes.DriverIncompatible Constant

Pascal

```
const DriverIncompatible = $001C;
```

C++

```
DriverIncompatible = 0x001C;
```

Description

This is DriverIncompatible, a member of class TDBXErrorCodes.

55.1.33.2.3 TDBXErrorCodes.DriverInitFailed Constant

Pascal

```
const DriverInitFailed = $0016;
```

C++

```
DriverInitFailed = 0x0016;
```

Description

This is DriverInitFailed, a member of class TDBXErrorCodes.

55.1.33.2.4 TDBXErrorCodes.DriverRestricted Constant

Pascal

```
const DriverRestricted = $0012;
```

C++

```
DriverRestricted = 0x0012;
```

Description

This is DriverRestricted, a member of class TDBXErrorCodes.

55.1.33.2.5 TDBXErrorCodes.DuplicateTransactionId Constant

Pascal

```
const DuplicateTransactionId = $0011;
```

C++

```
DuplicateTransactionId = 0x0011;
```

Description

This is DuplicateTransactionId, a member of class TDBXErrorCodes.

55.1.33.2.6 TDBXErrorCodes.EOF Constant

Pascal

```
const EOF = $000A;
```

C++

```
EOF = 0x000A;
```

Description

This is EOF, a member of class TDBXErrorCodes.

55.1.33.2.7 TDBXErrorCodes.InvalidArgument Constant

Pascal

```
const InvalidArgument = $001D;
```

C++

```
InvalidArgument = 0x001D;
```

Description

This is InvalidArgument, a member of class TDBXErrorCodes.

55.1.33.2.8 TDBXErrorCodes.InvalidHandle Constant

Pascal

```
const InvalidHandle = $0004;
```

C++

```
InvalidHandle = 0x0004;
```

Description

This is InvalidHandle, a member of class TDBXErrorCodes.

55.1.33.2.9 TDBXErrorCodes.InvalidIsolationLevel Constant

Pascal

```
const InvalidIsolationLevel = $000F;
```

C++

```
InvalidIsolationLevel = 0x000F;
```

Description

This is InvalidIsolationLevel, a member of class TDBXErrorCodes.

55.1.33.2.10 TDBXErrorCodes.InvalidLength Constant

Pascal

```
const InvalidLength = $000E;
```

C++

```
InvalidLength = 0x000E;
```

Description

This is InvalidLength, a member of class TDBXErrorCodes.

55.1.33.2.11 TDBXErrorCodes.InvalidOperation Constant

Pascal

```
const InvalidOperation = $001E;
```

C++

```
InvalidOperation = 0x001E;
```

Description

This is InvalidOperation, a member of class TDBXErrorCodes.

55.1.33.2.12 TDBXErrorCodes.InvalidOrdinal Constant

Pascal

```
const InvalidOrdinal = $0008;
```

C++

```
InvalidOrdinal = 0x0008;
```

Description

This is InvalidOrdinal, a member of class TDBXErrorCodes.

55.1.33.2.13 TDBXErrorCodes.InvalidParameter Constant

Pascal

```
const InvalidParameter = $0009;
```

C++

```
InvalidParameter = 0x0009;
```

Description

This is InvalidParameter, a member of class TDBXErrorCodes.

55.1.33.2.14 TDBXErrorCodes.InvalidPrecision Constant

Pascal

```
const InvalidPrecision = $000D;
```

C++

```
InvalidPrecision = 0x000D;
```

Description

This is InvalidPrecision, a member of class TDBXErrorCodes.

55.1.33.2.15 TDBXErrorCodes.InvalidReference Constant

Pascal

```
const InvalidReference = $0018;
```

C++

```
InvalidReference = 0x0018;
```

Description

This is InvalidReference, a member of class TDBXErrorCodes.

55.1.33.2.16 TDBXErrorCodes.InvalidTime Constant

Pascal

```
const InvalidTime = $0006;
```

C++

```
InvalidTime = 0x0006;
```

Description

This is InvalidTime, a member of class TDBXErrorCodes.

55.1.33.2.17 TDBXErrorCodes.InvalidTransactionId Constant

Pascal

```
const InvalidTransactionId = $0010;
```

C++

```
InvalidTransactionId = 0x0010;
```

Description

This is InvalidTransactionId, a member of class TDBXErrorCodes.

55.1.33.2.18 TDBXErrorCodes.InvalidType Constant

Pascal

```
const InvalidType = $0007;
```

C++

```
InvalidType = 0x0007;
```

Description

This is InvalidType, a member of class TDBXErrorCodes.

55.1.33.2.19 TDBXErrorCodes.InvalidUserOrPassword Constant

Pascal

```
const InvalidUserOrPassword = $000C;
```

C++

```
InvalidUserOrPassword = 0x000C;
```

Description

This is InvalidUserOrPassword, a member of class TDBXErrorCodes.

55.1.33.2.20 TDBXErrorCodes.MaxCommonErrors Constant

Pascal

```
const MaxCommonErrors = $0064;
```

C++

```
MaxCommonErrors = 0x0064;
```

Description

This is MaxCommonErrors, a member of class TDBXErrorCodes.

55.1.33.2.21 TDBXErrorCodes.MissingParameterMarker Constant

Pascal

```
const MissingParameterMarker = $001A;
```

C++

```
MissingParameterMarker = 0x001A;
```

Description

This is MissingParameterMarker, a member of class TDBXErrorCodes.

55.1.33.2.22 TDBXErrorCodes.MultipleTransactionNotEnabled Constant

Pascal

```
const MultipleTransactionNotEnabled = $0014;
```

C++

```
MultipleTransactionNotEnabled = 0x0014;
```

Description

This is MultipleTransactionNotEnabled, a member of class TDBXErrorCodes.

55.1.33.2.23 TDBXErrorCodes.NoData Constant

Pascal

```
const NoData = $0064;
```

C++

```
NoData = 0x0064;
```

Description

This is NoData, a member of class TDBXErrorCodes.

55.1.33.2.24 TDBXErrorCodes.NoMemory Constant

Pascal

```
const NoMemory = $0002;
```

C++

```
NoMemory = 0x0002;
```

Description

This is NoMemory, a member of class TDBXErrorCodes.

55.1.33.2.25 TDBXErrorCodes.None Constant

Pascal

```
const None = $0000;
```

C++

```
None = 0x0000;
```

Description

This is None, a member of class TDBXErrorCodes.

55.1.33.2.26 TDBXErrorCodes.NoTable Constant

Pascal

```
const NoTable = $0019;
```

C++

```
NoTable = 0x0019;
```

Description

This is NoTable, a member of class TDBXErrorCodes.

55.1.33.2.27 TDBXErrorCodes.NotImplemented Constant

Pascal

```
const NotImplemented = $001B;
```

C++

```
NotImplemented = 0x001B;
```

Description

This is NotImplemented, a member of class TDBXErrorCodes.

55.1.33.2.28 TDBXErrorCodes.NotSupported Constant

Pascal

```
const NotSupported = $0005;
```

C++

```
NotSupported = 0x0005;
```

Description

This is NotSupported, a member of class TDBXErrorCodes.

55.1.33.2.29 TDBXErrorCodes.OptimisticLockFailed Constant

Pascal

```
const OptimisticLockFailed = $0017;
```

C++

```
OptimisticLockFailed = 0x0017;
```

Description

This is OptimisticLockFailed, a member of class TDBXErrorCodes.

55.1.33.2.30 TDBXErrorCodes.ParameterNotSet Constant

Pascal

```
const ParameterNotSet = $000B;
```

C++

```
ParameterNotSet = 0x000B;
```

Description

This is ParameterNotSet, a member of class TDBXErrorCodes.

55.1.33.2.31 TDBXErrorCodes.TransactionActive Constant

Pascal

```
const TransactionActive = $0013;
```

C++

```
TransactionActive = 0x0013;
```

Description

This is TransactionActive, a member of class TDBXErrorCodes.

55.1.33.2.32 TDBXErrorCodes.UnsupportedFieldType Constant

Pascal

```
const UnsupportedFieldType = $0003;
```

C++

```
UnsupportedFieldType = 0x0003;
```

Description

This is UnsupportedFieldType, a member of class TDBXErrorCodes.

55.1.33.2.33 TDBXErrorCodes.VendorError Constant

Pascal

```
const VendorError = $0065;
```

C++

```
VendorError = 0x0065;
```

Description

This is VendorError, a member of class TDBXErrorCodes.

55.1.33.2.34 TDBXErrorCodes.Warning Constant

Pascal

```
const Warning = $0001;
```

C++

```
Warning = 0x0001;
```

Description

This is Warning, a member of class TDBXErrorCodes.

55.1.34 TDBXIniFileConnectionFactory Class

Creates new instances of TDBXConnection (see page 10398) from the dbxdrivers.ini and dbxconnections.ini files

Class Hierarchy

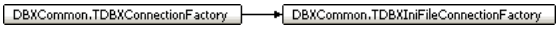

Pascal

```
TDBXIniFileConnectionFactory = class(TDBXConnectionFactory);
```

C++

```
class TDBXIniFileConnectionFactory : public TDBXConnectionFactory;
```

File

DBXCommon

Description

This class is an extension of TDBXConnectionFactory (see page 10422) that can be used to create new instances of TDBXConnection (see page 10398). It gets driver and connection properties from the dbxdrivers.ini and dbxconnections.ini files for a given connection name.

See Also

Connecting to a Database using the dbExpress Driver Framework

TDBXConnection (see page 10398)

TDBXConnectionFactory (see page 10422)

55.1.34.1 TDBXIniFileConnectionFactory Members

The following tables list the members exposed by TDBXIniFileConnectionFactory.

Protected Methods

	Name	Description
	DerivedClose (see page 10503)	This is DerivedClose, a member of class TDBXIniFileConnectionFactory.
	DerivedOpen (see page 10503)	This is DerivedOpen, a member of class TDBXIniFileConnectionFactory.
	GetErrorEvent (see page 10429)	This is GetErrorEvent, a member of class TDBXConnectionFactory.
	GetTraceFlags (see page 10429)	This is GetTraceFlags, a member of class TDBXConnectionFactory.
	GetTraceInfoEvent (see page 10429)	This is GetTraceInfoEvent, a member of class TDBXConnectionFactory.
	SetErrorEvent (see page 10430)	This is SetErrorEvent, a member of class TDBXConnectionFactory.
	SetTraceFlags (see page 10430)	This is SetTraceFlags, a member of class TDBXConnectionFactory.
	SetTraceInfoEvent (see page 10431)	This is SetTraceInfoEvent, a member of class TDBXConnectionFactory.

Public Events

	Name	Description
	OnError (see page 10432)	TDBXErrorEvent (see page 10671) handler.
	OnTrace (see page 10432)	TDBXTraceInfoEvent handler.

Public Methods

	Name	Description
	Close (see page 10425)	Closes the factory.

	Create (see page 10502)	Creates an instance of TDBXIniFileConnectionFactory (see page 10501)
	Destroy (see page 10502)	Destroys an instance of TDBXIniFileConnectionFactory (see page 10501)
	GetConnection (see page 10426)	This is the overview for the GetConnection method overload.
	GetConnectionDriver (see page 10427)	Gets a TDBXDriver (see page 10472) instance for a connection name
	GetConnectionFactory (see page 10427)	Gets a TDBXConnectionFactory (see page 10422) singleton
	GetConnectionItems (see page 10428)	Adds names of all connections to Items list.
	GetConnectionProperties (see page 10428)	Gets a TDBXPropertiesinstance for a connection name
	GetDriver (see page 10428)	Returns TDBXDriver (see page 10472) instance for a driver name
	GetDriverProperties (see page 10429)	Returns TDBXProperties (see page 10559) instance of driver properties
	Open (see page 10430)	Opens the TDBXConnectionFactory (see page 10422)
	SetConnectionFactory (see page 10430)	Change the default TDBXConnectionFactorysingleton

Public Properties

	Name	Description
	ConnectionsFile (see page 10503)	File name for the connections ini file
	DriversFile (see page 10504)	File name for the drivers ini file
	TraceFlags (see page 10431)	Trace flags to enable or disable tracing categories

Strict protected Fields

	Name	Description
	FDBXContext (see page 10424)	This is FDBXContext, a member of class TDBXConnectionFactory.

Strict protected Methods

	Name	Description
	CreateDelegatePaths (see page 10426)	This is CreateDelegatePaths, a member of class TDBXConnectionFactory.

55.1.34.2 TDBXIniFileConnectionFactory.Create Constructor

Creates an instance of TDBXIniFileConnectionFactory (see page 10501)

Pascal

```
constructor Create;
```

C++

```
__fastcall TDBXIniFileConnectionFactory();
```

Description

This constructor creates and initializes an instance of TDBXIniFileConnectionFactory (see page 10501). You can create a new TDBXConnection (see page 10398) object from this object that allows you to connect to a database. This class gets driver and connection properties from the dbxdrivers.ini and dbxconnections.ini files for a given connection name.

See Also

TDBXConnection (see page 10398)

TDBXIniFileConnectionFactory (see page 10501)

55.1.34.3 TDBXIniFileConnectionFactory.Destroy Destructor

Destroys an instance of TDBXIniFileConnectionFactory (see page 10501)

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXIniFileConnectionFactory();
```

Description

This destructor destroys an instance of TDBXIniFileConnectionFactory (see page 10501).

55.1.34.4 TDBXIniFileConnectionFactory Methods

The methods of the TDBXIniFileConnectionFactory class are listed here.

Protected Methods

	Name	Description
	DerivedClose (see page 10503)	This is DerivedClose, a member of class TDBXIniFileConnectionFactory.
	DerivedOpen (see page 10503)	This is DerivedOpen, a member of class TDBXIniFileConnectionFactory.

55.1.34.4.1 TDBXIniFileConnectionFactory.DerivedClose Method

Pascal

```
procedure DerivedClose; override;
```

C++

```
virtual __fastcall DerivedClose();
```

Description

This is DerivedClose, a member of class TDBXIniFileConnectionFactory.

55.1.34.4.2 TDBXIniFileConnectionFactory.DerivedOpen Method

Pascal

```
procedure DerivedOpen; override;
```

C++

```
virtual __fastcall DerivedOpen();
```

Description

This is DerivedOpen, a member of class TDBXIniFileConnectionFactory.

55.1.34.5 TDBXIniFileConnectionFactory Properties

The properties of the TDBXIniFileConnectionFactory class are listed here.

Public Properties

	Name	Description
	ConnectionsFile (see page 10503)	File name for the connections ini file
	DriversFile (see page 10504)	File name for the drivers ini file

55.1.34.5.1 TDBXIniFileConnectionFactory.ConnectionsFile Property

File name for the connections ini file

Pascal

```
property ConnectionsFile: WideString;
```

C++

```
__property BSTR ConnectionsFile;
```

Description

This property contains the file name for the connections ini file.

55.1.34.5.2 TDBXIniFileConnectionFactory.DriversFile Property

File name for the drivers ini file

Pascal

```
property DriversFile: WideString;
```

C++

```
__property BSTR DriversFile;
```

Description

This property contains the file name for the drivers ini file.

55.1.35 TDBXIniFileConnectionFactoryEx Class

Extension of TDBXIniFileConnectionFactory (see page 10501).

Class Hierarchy

Pascal

```
TDBXIniFileConnectionFactoryEx = class(TDBXIniFileConnectionFactory);
```

C++

```
class TDBXIniFileConnectionFactoryEx : public TDBXIniFileConnectionFactory;
```

File

DBXCommon

Description

TDBXIniFileConnectionFactoryEx is an extension of TDBXIniFileConnectionFactory (see page 10501).

Note: Classes (see page 2685) ending in “Ex” are temporary classes that exist to avoid breaking interfaces. Their contents will be migrated to their base class in the next release.

See Also

TDBXIniFileConnectionFactory (see page 10501)

55.1.35.1 TDBXIniFileConnectionFactoryEx Members

The following tables list the members exposed by TDBXIniFileConnectionFactoryEx.

Protected Methods

	Name	Description
	DerivedClose (see page 10426)	This is DerivedClose, a member of class TDBXConnectionFactory.
	DerivedClose (see page 10503)	This is DerivedClose, a member of class TDBXIniFileConnectionFactory.
	DerivedOpen (see page 10426)	This is DerivedOpen, a member of class TDBXConnectionFactory.
	DerivedOpen (see page 10503)	This is DerivedOpen, a member of class TDBXIniFileConnectionFactory.
	GetErrorEvent (see page 10429)	This is GetErrorEvent, a member of class TDBXConnectionFactory.
	GetTraceFlags (see page 10429)	This is GetTraceFlags, a member of class TDBXConnectionFactory.
	GetTraceInfoEvent (see page 10429)	This is GetTraceInfoEvent, a member of class TDBXConnectionFactory.
	SetErrorEvent (see page 10430)	This is SetErrorEvent, a member of class TDBXConnectionFactory.
	SetTraceFlags (see page 10430)	This is SetTraceFlags, a member of class TDBXConnectionFactory.
	SetTraceInfoEvent (see page 10431)	This is SetTraceInfoEvent, a member of class TDBXConnectionFactory.

Public Events

	Name	Description
	OnError (see page 10432)	TDBXErrorEvent (see page 10671) handler.
	OnTrace (see page 10432)	TDBXTraceInfoEvent handler.

Public Methods

	Name	Description
	Close (see page 10425)	Closes the factory.
	Create (see page 10423)	Creates an instance of TDBXConnectionFactory (see page 10422)
	Create (see page 10502)	Creates an instance of TDBXIniFileConnectionFactory (see page 10501)
	Destroy (see page 10424)	Destroys a TDBXConnectionFactory (see page 10422)
	Destroy (see page 10502)	Destroys an instance of TDBXIniFileConnectionFactory (see page 10501)
	GetConnection (see page 10426)	This is the overview for the GetConnection method overload.
	GetConnectionDriver (see page 10427)	Gets a TDBXDriver (see page 10472) instance for a connection name
	GetConnectionFactory (see page 10427)	Gets a TDBXConnectionFactory (see page 10422) singleton
	GetConnectionItems (see page 10428)	Adds names of all connections to Items list.
	GetConnectionProperties (see page 10428)	Gets a TDBXPropertiesinstance for a connection name
	GetDriver (see page 10428)	Returns TDBXDriver (see page 10472) instance for a driver name
	GetDriverItems (see page 10506)	Adds names of all Drivers in factory to Items list.
	GetDriverProperties (see page 10429)	Returns TDBXProperties (see page 10559) instance of driver properties
	Open (see page 10430)	Opens the TDBXConnectionFactory (see page 10422)
	SetConnectionFactory (see page 10430)	Change the default TDBXConnectionFactorysingleton

Public Properties

	Name	Description
	ConnectionsFile (see page 10503)	File name for the connections ini file
	DriversFile (see page 10504)	File name for the drivers ini file
	TraceFlags (see page 10431)	Trace flags to enable or disable tracing categories

Strict protected Fields

	Name	Description
	FDBXContext (see page 10424)	This is FDBXContext, a member of class TDBXConnectionFactory.

Strict protected Methods

	Name	Description
	CreateDelegatePaths (see page 10426)	This is CreateDelegatePaths, a member of class TDBXConnectionFactory.

55.1.35.2 TDBXIniFileConnectionFactoryEx Methods

The methods of the TDBXIniFileConnectionFactoryEx class are listed here.

Public Methods

	Name	Description
	GetDriverItems (see page 10506)	Adds names of all Drivers in factory to Items list.

55.1.35.2.1 TDBXIniFileConnectionFactoryEx.GetDriverItems Method

Adds names of all Drivers in factory to Items list.

Pascal

```
procedure GetDriverItems(Items: TStrings);
```

C++

```
__fastcall GetDriverItems(TStrings Items);
```

Description

GetDriverItems adds the names of all Drivers in the factory to the Items list.

55.1.36 TDBXInt16Value Class

Implementation of 16-byte integer values.

Class Hierarchy

Pascal

```
TDBXInt16Value = class(TDBXWritableValue);
```

C++

```
class TDBXInt16Value : public TDBXWritableValue;
```

File

DBXCommon

Description

Implementation of TDBXWritableValue (see page 10661) for TDBXDataTypes.Int16Type values.

55.1.36.1 TDBXInt16Value Members

The following tables list the members exposed by TDBXInt16Value.

Protected Methods

	Name	Description
	Create (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
	Create (see page 10662)	Creates an instance of TDBXWritableValue (see page 10661)
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.
V	SetPendingValue (see page 10666)	This is SetPendingValue, a member of class TDBXWritableValue.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Methods

	Name	Description
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
V	Destroy (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
V	Destroy (see page 10663)	Destroys a TDBXWritableValue (see page 10661) instance.
V	GetAnsiString (see page 10629)	Gets the string value for ANSI strings.
V	GetBcd (see page 10629)	Gets the BCD value for binary-compressed decimal data types.
V	GetBoolean (see page 10629)	Gets the boolean value for boolean data types.
V	GetBytes (see page 10629)	Gets the byte array value for byte array and blob types.
V	GetDate (see page 10630)	Gets the date value for date data types.
V	GetDouble (see page 10630)	Gets the double floating point value for double data types.
V	GetInt16 (see page 10508)	Gets the smallInt value for 16-byte integer data types.
V	GetInt32 (see page 10631)	Gets the 32-byte integer value for 32-byte integer data types.
V	GetInt64 (see page 10631)	Gets the 64-byte integer value for 64-byte integer data types.
V	GetTime (see page 10631)	Gets the time value for time data types.
V	GetTimeStamp (see page 10632)	Gets the timestamp value for timestamp data types.
V	GetValueSize (see page 10632)	Gets the size in bytes for values.
V	GetWideString (see page 10632)	Gets the wide string value for wide string data types.
V	IsNull (see page 10508)	Returns true if the value is **null**.
V	SetAnsiString (see page 10663)	Sets the string value for ANSI strings.
V	SetBcd (see page 10664)	Sets the BCD value for binary-compressed decimal data types.
V	SetBoolean (see page 10664)	Sets the boolean value for boolean data types..
V	SetDate (see page 10664)	Sets the date value for date data types..
V	SetDouble (see page 10664)	Sets the double floating point value for double data types.
V	SetDynamicBytes (see page 10665)	Sets the dynamic array value for byte array and blob types.
V	SetInt16 (see page 10508)	Sets the the smallInt value for 16-byte integer data types.
V	SetInt32 (see page 10665)	Sets the 32-byte integer value for 32-byte integer data types.
V	SetInt64 (see page 10665)	Sets the 64-byte integer value for 64-byte integer data types.
V	SetNull (see page 10666)	Sets a writable value of any type to null.
V	SetStaticBytes (see page 10666)	Sets the open array value for byte array and blob types
V	SetTime (see page 10666)	Sets the time value for time data types.
V	SetTimeStamp (see page 10667)	Sets the timestamp value for timestamp data types.
V	SetWideString (see page 10667)	Sets the wide string value for wide string data types.

Public Properties

	Name	Description
R	ValueType (see page 10633)	Gets the value type for this value.

55.1.36.2 TDBXInt16Value Methods

The methods of the TDBXInt16Value class are listed here.

Public Methods

	Name	Description
V	GetInt16 (see page 10508)	Gets the smallInt value for 16-byte integer data types.
V	IsNull (see page 10508)	Returns true if the value is **null**.
V	SetInt16 (see page 10508)	Sets the the smallInt value for 16-byte integer data types.

55.1.36.2.1 TDBXInt16Value.GetInt16 Method

Gets the smallInt value for 16-byte integer data types.

Pascal

```
function GetInt16: SmallInt; override;
```

C++

```
virtual __fastcall SmallInt GetInt16();
```

Description

Gets the smallInt value for TDBXValue (see page 10626) instances with a data type of Int16Type. Throws a TDBXError (see page 10486) exception if IsNull (see page 10508) is true.

55.1.36.2.2 TDBXInt16Value.IsNull Method

Returns true if the value is **null**.

Pascal

```
function IsNull: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsNull();
```

Description

Returns true if the value is **null**. This method should be called before calling any of the Get methods. If the value is null, the Get methods will throw TDBXError (see page 10486) exception if they are called.

55.1.36.2.3 TDBXInt16Value.SetInt16 Method

Sets the the smallInt value for 16-byte integer data types.

Pascal

```
procedure SetInt16(Value: SmallInt); override;
```

C++

```
virtual __fastcall SetInt16(SmallInt Value);
```

Description

Sets the the smallInt value for TDBXValue (see page 10626) instances with a data type of Int16Type.

55.1.37 TDBXInt32Value Class

Implementation of writable 32-byte integer values.

Class Hierarchy

Pascal

```
TDBXInt32Value = class(TDBXWritableValue);
```

C++

```
class TDBXInt32Value : public TDBXWritableValue;
```

File

DBXCommon

Description

Implementation of TDBXWritableValue (see page 10661) for TDBXDataTypes.Int32Type values.

55.1.37.1 TDBXInt32Value Members

The following tables list the members exposed by TDBXInt32Value.

Protected Methods

	Name	Description
	Create (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
	Create (see page 10662)	Creates an instance of TDBXWritableValue (see page 10661)
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.
V	SetPendingValue (see page 10666)	This is SetPendingValue, a member of class TDBXWritableValue.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Methods

	Name	Description
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
V	Destroy (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
V	Destroy (see page 10663)	Destroys a TDBXWritableValue (see page 10661) instance.
V	GetAnsiString (see page 10629)	Gets the string value for ANSI strings.
V	GetBcd (see page 10629)	Gets the BCD value for binary-compressed decimal data types.
V	GetBoolean (see page 10629)	Gets the boolean value for boolean data types.
V	GetBytes (see page 10629)	Gets the byte array value for byte array and blob types.
V	GetDate (see page 10630)	Gets the date value for date data types.
V	GetDouble (see page 10630)	Gets the double floating point value for double data types.
V	GetInt16 (see page 10631)	Gets the smallInt value for 16-byte integer data types.
V	GetInt32 (see page 10510)	Gets the 32-byte integer value for 32-byte integer data types.
V	GetInt64 (see page 10631)	Gets the 64-byte integer value for 64-byte integer data types.
V	GetTime (see page 10631)	Gets the time value for time data types.
V	GetTimeStamp (see page 10632)	Gets the timestamp value for timestamp data types.
V	GetValueSize (see page 10632)	Gets the size in bytes for values.
V	GetWideString (see page 10632)	Gets the wide string value for wide string data types.
V	IsNull (see page 10510)	Returns true if the value is **null**.
V	SetAnsiString (see page 10663)	Sets the string value for ANSI strings.
V	SetBcd (see page 10664)	Sets the BCD value for binary-compressed decimal data types.
V	SetBoolean (see page 10664)	Sets the boolean value for boolean data types..
V	SetDate (see page 10664)	Sets the date value for date data types..
V	SetDouble (see page 10664)	Sets the double floating point value for double data types.
V	SetDynamicBytes (see page 10665)	Sets the dynamic array value for byte array and blob types.
V	SetInt16 (see page 10665)	Sets the the smallInt value for 16-byte integer data types.

	SetInt32 (see page 10511)	Sets the 32-byte integer value for 32-byte integer data types.
	SetInt64 (see page 10665)	Sets the 64-byte integer value for 64-byte integer data types.
	SetNull (see page 10666)	Sets a writable value of any type to null.
	SetStaticBytes (see page 10666)	Sets the open array value for byte array and blob types
	SetTime (see page 10666)	Sets the time value for time data types.
	SetTimeStamp (see page 10667)	Sets the timestamp value for timestamp data types.
	SetWideString (see page 10667)	Sets the wide string value for wide string data types.

Public Properties

	Name	Description
	ValueType (see page 10633)	Gets the value type for this value.

55.1.37.2 TDBXInt32Value Methods

The methods of the TDBXInt32Value class are listed here.

Public Methods

	Name	Description
	GetInt32 (see page 10510)	Gets the 32-byte integer value for 32-byte integer data types.
	IsNull (see page 10510)	Returns true if the value is **null**.
	SetInt32 (see page 10511)	Sets the 32-byte integer value for 32-byte integer data types.

55.1.37.2.1 TDBXInt32Value.GetInt32 Method

Gets the 32-byte integer value for 32-byte integer data types.

Pascal

```
function GetInt32: TInt32; override;
```

C++

```
virtual __fastcall TInt32 GetInt32();
```

Description

Gets the 32-byte integer value for TDBXValue (see page 10626) instances with a data type of Int32Type. Throws a TDBXError (see page 10486) exception if IsNull (see page 10510) is true.

55.1.37.2.2 TDBXInt32Value.IsNull Method

Returns true if the value is **null**.

Pascal

```
function IsNull: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsNull();
```

Description

Returns true if the value is **null**. This method should be called before calling any of the Get methods. If the value is null, the Get methods will throw TDBXError (see page 10486) exception if they are called.

55.1.37.2.3 TDBXInt32Value.SetInt32 Method

Sets the 32-byte integer value for 32-byte integer data types.

Pascal

```
procedure SetInt32(Value: TInt32); override;
```

C++

```
virtual __fastcall SetInt32(TInt32 Value);
```

Description

Sets the 32-byte integer value for TDBXValue (see page 10626) instances with a data type of Int32Type.

55.1.38 TDBXInt64Value Class

Implementation of writable 64-byte integer values.

Class Hierarchy

Pascal

```
TDBXInt64Value = class(TDBXWritableValue);
```

C++

```
class TDBXInt64Value : public TDBXWritableValue;
```

File

DBXCommon

Description

Implementation of TDBXWritableValue (see page 10661) for TDBXDataTypes (see page 10447). Int64Type values.

55.1.38.1 TDBXInt64Value Members

The following tables list the members exposed by TDBXInt64Value.

Protected Methods

	Name	Description
	Create (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
	Create (see page 10662)	Creates an instance of TDBXWritableValue (see page 10661)
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.
V	SetPendingValue (see page 10666)	This is SetPendingValue, a member of class TDBXWritableValue.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Methods

	Name	Description
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
	Destroy (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
	Destroy (see page 10663)	Destroys a TDBXWritableValue (see page 10661) instance.
	GetAnsiString (see page 10629)	Gets the string value for ANSI strings.
	GetBcd (see page 10629)	Gets the BCD value for binary-compressed decimal data types.
	GetBoolean (see page 10629)	Gets the boolean value for boolean data types.
	GetBytes (see page 10629)	Gets the byte array value for byte array and blob types.
	GetDate (see page 10630)	Gets the date value for date data types.
	GetDouble (see page 10630)	Gets the double floating point value for double data types.
	GetInt16 (see page 10631)	Gets the smallInt value for 16-byte integer data types.
	GetInt32 (see page 10631)	Gets the 32-byte integer value for 32-byte integer data types.
	GetInt64 (see page 10512)	Gets the 64-byte integer value for 64-byte integer data types.
	GetTime (see page 10631)	Gets the time value for time data types.
	GetTimeStamp (see page 10632)	Gets the timestamp value for timestamp data types.
	GetValueSize (see page 10632)	Gets the size in bytes for values.
	GetWideString (see page 10632)	Gets the wide string value for wide string data types.
	IsNull (see page 10513)	Returns true if the value is **null**.
	SetAnsiString (see page 10663)	Sets the string value for ANSI strings.
	SetBcd (see page 10664)	Sets the BCD value for binary-compressed decimal data types.
	SetBoolean (see page 10664)	Sets the boolean value for boolean data types..
	SetDate (see page 10664)	Sets the date value for date data types..
	SetDouble (see page 10664)	Sets the double floating point value for double data types.
	SetDynamicBytes (see page 10665)	Sets the dynamic array value for byte array and blob types.
	SetInt16 (see page 10665)	Sets the the smallInt value for 16-byte integer data types.
	SetInt32 (see page 10665)	Sets the 32-byte integer value for 32-byte integer data types.
	SetInt64 (see page 10513)	Sets the 64-byte integer value for 64-byte integer data types.
	SetNull (see page 10666)	Sets a writable value of any type to null.
	SetStaticBytes (see page 10666)	Sets the open array value for byte array and blob types
	SetTime (see page 10666)	Sets the time value for time data types.
	SetTimeStamp (see page 10667)	Sets the timestamp value for timestamp data types.
	SetWideString (see page 10667)	Sets the wide string value for wide string data types.

Public Properties

	Name	Description
	ValueType (see page 10633)	Gets the value type for this value.

55.1.38.2 TDBXInt64Value Methods

The methods of the TDBXInt64Value class are listed here.

Public Methods

	Name	Description
	GetInt64 (see page 10512)	Gets the 64-byte integer value for 64-byte integer data types.
	IsNull (see page 10513)	Returns true if the value is **null**.
	SetInt64 (see page 10513)	Sets the 64-byte integer value for 64-byte integer data types.

55.1.38.2.1 TDBXInt64Value.GetInt64 Method

Gets the 64-byte integer value for 64-byte integer data types.

Pascal

```
function GetInt64: Int64; override;
```

C++

```
virtual __fastcall Int64 GetInt64();
```

Description

Gets the 64-byte integer value for TDBXValue (see page 10626) instances with a data type of Int64Type. Throws a TDBXError (see page 10486) exception if IsNull (see page 10513) is true.

55.1.38.2.2 TDBXInt64Value.IsNull Method

Returns true if the value is **null**.

Pascal

```
function IsNull: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsNull();
```

Description

Returns true if the value is **null**. This method should be called before calling any of the Get methods. If the value is null, the Get methods will throw TDBXError (see page 10486) exception if they are called.

55.1.38.2.3 TDBXInt64Value.SetInt64 Method

Sets the 64-byte integer value for 64-byte integer data types.

Pascal

```
procedure SetInt64(Value: Int64); override;
```

C++

```
virtual __fastcall SetInt64(Int64 Value);
```

Description

Sets the 64-byte integer value for TDBXValue (see page 10626) instances with a data type of Int64Type.

55.1.39 TDBXIsolations Class

Isolation levels supported by TDBXTransaction (see page 10623).

Class Hierarchy

Pascal

```
TDBXIsolations = class;
```

C++

```
class TDBXIsolations;
```

File

DBXCommon

Description

Isolation levels supported by the TDBXTransaction (see page 10623) class. The isolation level indicates how much transactions can interact with each other.

This table shows isolation levels constants.

Constant	Description
ReadCommitted (see page 10515)	Current transaction sees only changes that have been committed, but can receive an inconsistent view of the data if additional changes are committed before the transaction ends.
RepeatableRead (see page 10515)	Current transaction is guaranteed a consistent view of the data, which includes only changes committed by other transactions at the start of the transaction.
DirtyRead (see page 10515)	Current transaction sees all changes made by other transactions, even if they have not yet been committed. This option is not available for the Oracle driver.
Serializable (see page 10515)	Available since dbExpress 4. Current transaction sees no changes made by other transactions including phantoms.
SnapShot (see page 10515)	Available since dbExpress 4. Read only transaction that provides a transactionally consistent snapshot of the database.

See Also

TDBXTransaction (see page 10623)

55.1.39.1 TDBXIsolations Members

The following tables list the members exposed by TDBXIsolations.

Public Constants

	Name	Description
	DirtyRead (see page 10515)	This is DirtyRead, a member of class TDBXIsolations.
	ReadCommitted (see page 10515)	This is ReadCommitted, a member of class TDBXIsolations.
	RepeatableRead (see page 10515)	This is RepeatableRead, a member of class TDBXIsolations.
	Serializable (see page 10515)	This is Serializable, a member of class TDBXIsolations.
	SnapShot (see page 10515)	This is SnapShot, a member of class TDBXIsolations.

55.1.39.2 TDBXIsolations Constants

The constants of the TDBXIsolations class are listed here.

Public Constants

	Name	Description
	DirtyRead (see page 10515)	This is DirtyRead, a member of class TDBXIsolations.
	ReadCommitted (see page 10515)	This is ReadCommitted, a member of class TDBXIsolations.
	RepeatableRead (see page 10515)	This is RepeatableRead, a member of class TDBXIsolations.

	Serializable (see page 10515)	This is Serializable, a member of class TDBXIsolations.
	SnapShot (see page 10515)	This is SnapShot, a member of class TDBXIsolations.

55.1.39.2.1 TDBXIsolations.DirtyRead Constant

Pascal

```
const DirtyRead = 2;
```

C++

```
DirtyRead = 2;
```

Description

This is DirtyRead, a member of class TDBXIsolations.

55.1.39.2.2 TDBXIsolations.ReadCommitted Constant

Pascal

```
const ReadCommitted = 0;
```

C++

```
ReadCommitted = 0;
```

Description

This is ReadCommitted, a member of class TDBXIsolations.

55.1.39.2.3 TDBXIsolations.RepeatableRead Constant

Pascal

```
const RepeatableRead = 1;
```

C++

```
RepeatableRead = 1;
```

Description

This is RepeatableRead, a member of class TDBXIsolations.

55.1.39.2.4 TDBXIsolations.Serializable Constant

Pascal

```
const Serializable = 3;
```

C++

```
Serializable = 3;
```

Description

This is Serializable, a member of class TDBXIsolations.

55.1.39.2.5 TDBXIsolations.SnapShot Constant

Pascal

```
const SnapShot = 4;
```

C++

```
SnapShot = 4;
```

Description

This is SnapShot, a member of class TDBXIsolations.

55.1.40 TDBXLookAheadStreamReader Class

Used internally by driver implementations and TDBXStreamValue (see page 10608).

Class Hierarchy

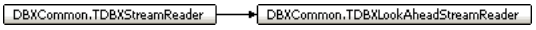

Pascal

```
TDBXLookAheadStreamReader = class(TDBXStreamReader);
```

C++

```
class TDBXLookAheadStreamReader : public TDBXStreamReader;
```

File

DBXCommon

Description

TDBXLookAheadStreamReader is used internally by driver implementations and TDBXStreamValue (see page 10608).

See Also

TDBXStreamValue (see page 10608)

55.1.40.1 TDBXLookAheadStreamReader Members

The following tables list the members exposed by TDBXLookAheadStreamReader.

Public Methods

	Name	Description
	ConvertToMemoryStream (see page 10517)	This is ConvertToMemoryStream, a member of class TDBXLookAheadStreamReader.
V	Destroy (see page 10516)	This is Destroy, a member of class TDBXLookAheadStreamReader.
V	Eos (see page 10517)	This is Eos, a member of class TDBXLookAheadStreamReader.
V	Read (see page 10517)	This is Read, a member of class TDBXLookAheadStreamReader.
	SetStream (see page 10518)	This is SetStream, a member of class TDBXLookAheadStreamReader.
V	Size (see page 10518)	This is Size, a member of class TDBXLookAheadStreamReader.

55.1.40.2 TDBXLookAheadStreamReader.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXLookAheadStreamReader();
```

Description

This is Destroy, a member of class TDBXLookAheadStreamReader.

55.1.40.3 TDBXLookAheadStreamReader Methods

The methods of the TDBXLookAheadStreamReader class are listed here.

Public Methods

	Name	Description
	ConvertToMemoryStream (see page 10517)	This is ConvertToMemoryStream, a member of class TDBXLookAheadStreamReader.
	Eos (see page 10517)	This is Eos, a member of class TDBXLookAheadStreamReader.
	Read (see page 10517)	This is Read, a member of class TDBXLookAheadStreamReader.
	SetStream (see page 10518)	This is SetStream, a member of class TDBXLookAheadStreamReader.
	Size (see page 10518)	This is Size, a member of class TDBXLookAheadStreamReader.

55.1.40.3.1 TDBXLookAheadStreamReader.ConvertToMemoryStream Method

Pascal

```
function ConvertToMemoryStream: TStream;
```

C++

```
__fastcall TStream ConvertToMemoryStream();
```

Description

This is ConvertToMemoryStream, a member of class TDBXLookAheadStreamReader.

55.1.40.3.2 TDBXLookAheadStreamReader.Eos Method

Pascal

```
function Eos: Boolean; override;
```

C++

```
virtual __fastcall Boolean Eos();
```

Description

This is Eos, a member of class TDBXLookAheadStreamReader.

55.1.40.3.3 TDBXLookAheadStreamReader.Read Method

Pascal

```
function Read(const Buffer: TBytes; Offset: Integer; Count: Integer): Integer; override;
```

C++

```
virtual __fastcall int Read(const TBytes Buffer, int Offset, int Count);
```

Description

This is Read, a member of class TDBXLookAheadStreamReader.

55.1.40.3.4 TDBXLookAheadStreamReader.SetStream Method

Pascal

```
procedure SetStream(Stream: TStream);
```

C++

```
__fastcall SetStream(TStream Stream);
```

Description

This is SetStream, a member of class TDBXLookAheadStreamReader.

55.1.40.3.5 TDBXLookAheadStreamReader.Size Method

Pascal

```
function Size: Int64; override;
```

C++

```
virtual __fastcall Int64 Size();
```

Description

This is Size, a member of class TDBXLookAheadStreamReader.

55.1.41 TDBXMemoryConnectionFactory Class

Implementation of TDBXConnectionFactory (see page 10422) that does not load driver and connection properties.

Class Hierarchy

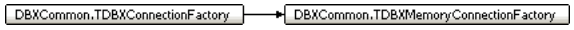

Pascal

```
TDBXMemoryConnectionFactory = class(TDBXConnectionFactory);
```

C++

```
class TDBXMemoryConnectionFactory : public TDBXConnectionFactory;
```

File

DBXCommon

Description

Implementation of TDBXConnectionFactory (see page 10422) that does not load driver and connection properties from initialization files. It is used by VCL when the dbxconnections.ini and dbxdrivers.ini files cannot be found.

See Also

TDBXConnectionFactory (see page 10422)

55.1.41.1 TDBXMemoryConnectionFactory Members

The following tables list the members exposed by TDBXMemoryConnectionFactory.

Protected Methods

	Name	Description
	DerivedClose (see page 10520)	This is DerivedClose, a member of class TDBXMemoryConnectionFactory.
	DerivedOpen (see page 10520)	This is DerivedOpen, a member of class TDBXMemoryConnectionFactory.
	GetErrorEvent (see page 10429)	This is GetErrorEvent, a member of class TDBXConnectionFactory.
	GetTraceFlags (see page 10429)	This is GetTraceFlags, a member of class TDBXConnectionFactory.
	GetTraceInfoEvent (see page 10429)	This is GetTraceInfoEvent, a member of class TDBXConnectionFactory.
	SetErrorEvent (see page 10430)	This is SetErrorEvent, a member of class TDBXConnectionFactory.
	SetTraceFlags (see page 10430)	This is SetTraceFlags, a member of class TDBXConnectionFactory.
	SetTraceInfoEvent (see page 10431)	This is SetTraceInfoEvent, a member of class TDBXConnectionFactory.

Public Events

	Name	Description
	OnError (see page 10432)	TDBXErrorEvent (see page 10671) handler.
	OnTrace (see page 10432)	TDBXTraceInfoEvent handler.

Public Methods

	Name	Description
	Close (see page 10425)	Closes the factory.
	Create (see page 10519)	Creates an instance of TDBXMemoryConnectionFactory (see page 10518)
	Destroy (see page 10520)	Destroys a TDBXMemoryConnectionFactory (see page 10518) instance.
	GetConnection (see page 10426)	This is the overview for the GetConnection method overload.
	GetConnectionDriver (see page 10427)	Gets a TDBXDriver (see page 10472) instance for a connection name
	GetConnectionFactory (see page 10427)	Gets a TDBXConnectionFactory (see page 10422) singleton
	GetConnectionItems (see page 10428)	Adds names of all connections to Items list.
	GetConnectionProperties (see page 10428)	Gets a TDBXPropertiesinstance for a connection name
	GetDriver (see page 10428)	Returns TDBXDriver (see page 10472) instance for a driver name
	GetDriverProperties (see page 10429)	Returns TDBXProperties (see page 10559) instance of driver properties
	Open (see page 10430)	Opens the TDBXConnectionFactory (see page 10422)
	SetConnectionFactory (see page 10430)	Change the default TDBXConnectionFactorysingleton

Public Properties

	Name	Description
	TraceFlags (see page 10431)	Trace flags to enable or disable tracing categories

Strict protected Fields

	Name	Description
	FDBXContext (see page 10424)	This is FDBXContext, a member of class TDBXConnectionFactory.

Strict protected Methods

	Name	Description
	CreateDelegatePaths (see page 10426)	This is CreateDelegatePaths, a member of class TDBXConnectionFactory.

55.1.41.2 TDBXMemoryConnectionFactory.Create Constructor

Creates an instance of TDBXMemoryConnectionFactory (see page 10518)

Pascal

```
constructor Create;
```

C++

```
__fastcall TDBXMemoryConnectionFactory();
```

Description

This constructor creates and initializes an instance of TDBXMemoryConnectionFactory (see page 10518).

See Also

Destroy (see page 10520)

55.1.41.3 TDBXMemoryConnectionFactory.Destroy Destructor

Destroys a TDBXMemoryConnectionFactory (see page 10518) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXMemoryConnectionFactory();
```

Description

This destructor destroys an instance of TDBXMemoryConnectionFactory (see page 10518).

See Also

Create (see page 10519)

55.1.41.4 TDBXMemoryConnectionFactory Methods

The methods of the TDBXMemoryConnectionFactory class are listed here.

Protected Methods

	Name	Description
	DerivedClose (see page 10520)	This is DerivedClose, a member of class TDBXMemoryConnectionFactory.
	DerivedOpen (see page 10520)	This is DerivedOpen, a member of class TDBXMemoryConnectionFactory.

55.1.41.4.1 TDBXMemoryConnectionFactory.DerivedClose Method

Pascal

```
procedure DerivedClose; override;
```

C++

```
virtual __fastcall DerivedClose();
```

Description

This is DerivedClose, a member of class TDBXMemoryConnectionFactory.

55.1.41.4.2 TDBXMemoryConnectionFactory.DerivedOpen Method

Pascal

```
procedure DerivedOpen; override;
```

C++

```
virtual __fastcall DerivedOpen();
```

Description

This is DerivedOpen, a member of class TDBXMemoryConnectionFactory.

55.1.42 TDBXMetaDatabaseColumnNames Class

Column names for metadata.

Class Hierarchy

DBXCommon.TDBXMetaDatabaseColumnNames

Pascal

```
TDBXMetaDatabaseColumnNames = class;
```

C++

```
class TDBXMetaDatabaseColumnNames;
```

File

DBXCommon

Description

TDBXMetaDatabaseColumnNames provides column names to access various kinds of database metadata.

This table contains column names in the metadata dataset for obtaining specific metadata.

Constant	Metadata column
QuoteChar (see page 10524)	Deprecated. Use QuotePrefix (see page 10524) and QuoteSuffix (see page 10524) properties in TDBXDatabaseMetaDataEx (see page 10444). Quote character used for quoting identifiers in the TDBXCommand.Text property
ProcedureQuoteChar (see page 10523)	Quote character used for quoting identifiers for stored procedures in the TDBXCommand.Text property.
MaxCommands (see page 10523)	Maximum number of commands that can be created.
SupportsTransactions (see page 10525)	Indicates whether the driver supports transactions.
SupportsNestedTransactions (see page 10524)	Indicates whether the driver supports nested transactions.
SupportsRowSetSize (see page 10525)	Indicates whether TDBXCommand.RowSetSize property is supported to control row buffering when executing queries.
ProductVersion (see page 10523)	ProductVersion (see page 10523) number in the format nn.nn.nnnn.
QuotePrefix (see page 10524)	Character to start a quoted identifier.
QuoteSuffix (see page 10524)	Character to end a quoted identifier.
SupportsLowerCaseIdentifiers (see page 10524)	Indicates whether lowercase identifiers are stored in lowercase characters on the database.
SupportsUpperCaseIdentifiers (see page 10525)	Indicates whether uppercase identifiers are stored in uppercase characters on the database.

See Also

TDBXCommand (see page 10378)

TDBXDatabaseMetaData (see page 10440)

TDBXDatabaseMetaDataEx (see page 10444)

RowSetSize (see page 10390)

Text (see page 10390)

55.1.42.1 TDBXMetaDatabaseColumnNames Members

The following tables list the members exposed by TDBXMetaDatabaseColumnNames.

Public Constants

	Name	Description
	MaxCommands (see page 10523)	This is MaxCommands, a member of class TDBXMetaDatabaseColumnNames.
	ProcedureQuoteChar (see page 10523)	This is ProcedureQuoteChar, a member of class TDBXMetaDatabaseColumnNames.
	ProductName (see page 10523)	This is ProductName, a member of class TDBXMetaDatabaseColumnNames.
	ProductVersion (see page 10523)	This is ProductVersion, a member of class TDBXMetaDatabaseColumnNames.
	QuoteChar (see page 10524)	This is QuoteChar, a member of class TDBXMetaDatabaseColumnNames.
	QuotePrefix (see page 10524)	This is QuotePrefix, a member of class TDBXMetaDatabaseColumnNames.
	QuoteSuffix (see page 10524)	This is QuoteSuffix, a member of class TDBXMetaDatabaseColumnNames.
	SupportsLowerCaseIdentifiers (see page 10524)	This is SupportsLowerCaseIdentifiers, a member of class TDBXMetaDatabaseColumnNames.
	SupportsNestedTransactions (see page 10524)	This is SupportsNestedTransactions, a member of class TDBXMetaDatabaseColumnNames.
	SupportsRowSetSize (see page 10525)	This is SupportsRowSetSize, a member of class TDBXMetaDatabaseColumnNames.
	SupportsTransactions (see page 10525)	This is SupportsTransactions, a member of class TDBXMetaDatabaseColumnNames.
	SupportsUpperCaseIdentifiers (see page 10525)	This is SupportsUpperCaseIdentifiers, a member of class TDBXMetaDatabaseColumnNames.

55.1.42.2 TDBXMetaDatabaseColumnNames Constants

The constants of the TDBXMetaDatabaseColumnNames class are listed here.

Public Constants

	Name	Description
	MaxCommands (see page 10523)	This is MaxCommands, a member of class TDBXMetaDatabaseColumnNames.
	ProcedureQuoteChar (see page 10523)	This is ProcedureQuoteChar, a member of class TDBXMetaDatabaseColumnNames.
	ProductName (see page 10523)	This is ProductName, a member of class TDBXMetaDatabaseColumnNames.
	ProductVersion (see page 10523)	This is ProductVersion, a member of class TDBXMetaDatabaseColumnNames.
	QuoteChar (see page 10524)	This is QuoteChar, a member of class TDBXMetaDatabaseColumnNames.
	QuotePrefix (see page 10524)	This is QuotePrefix, a member of class TDBXMetaDatabaseColumnNames.
	QuoteSuffix (see page 10524)	This is QuoteSuffix, a member of class TDBXMetaDatabaseColumnNames.
	SupportsLowerCaseIdentifiers (see page 10524)	This is SupportsLowerCaseIdentifiers, a member of class TDBXMetaDatabaseColumnNames.
	SupportsNestedTransactions (see page 10524)	This is SupportsNestedTransactions, a member of class TDBXMetaDatabaseColumnNames.
	SupportsRowSetSize (see page 10525)	This is SupportsRowSetSize, a member of class TDBXMetaDatabaseColumnNames.

	SupportsTransactions (see page 10525)	This is SupportsTransactions, a member of class TDBXMetaDatabaseColumnNames.
	SupportsUpperCaseIdentifiers (see page 10525)	This is SupportsUpperCaseIdentifiers, a member of class TDBXMetaDatabaseColumnNames.

55.1.42.2.1 TDBXMetaDatabaseColumnNames.MaxCommands Constant

Pascal

```
const MaxCommands = 'MaxCommands';
```

C++

```
MaxCommands = 'MaxCommands';
```

Description

This is MaxCommands, a member of class TDBXMetaDatabaseColumnNames.

55.1.42.2.2 TDBXMetaDatabaseColumnNames.ProcedureQuoteChar Constant

Pascal

```
const ProcedureQuoteChar = 'ProcedureQuoteChar';
```

C++

```
ProcedureQuoteChar = 'ProcedureQuoteChar';
```

Description

This is ProcedureQuoteChar, a member of class TDBXMetaDatabaseColumnNames.

55.1.42.2.3 TDBXMetaDatabaseColumnNames.ProductName Constant

Pascal

```
const ProductName = 'ProductName';
```

C++

```
ProductName = 'ProductName';
```

Description

This is ProductName, a member of class TDBXMetaDatabaseColumnNames.

55.1.42.2.4 TDBXMetaDatabaseColumnNames.ProductVersion Constant

Pascal

```
const ProductVersion = 'ProductVersion';
```

C++

```
ProductVersion = 'ProductVersion';
```

Description

This is ProductVersion, a member of class TDBXMetaDatabaseColumnNames.

55.1.42.2.5 TDBXMetaDatabaseColumnNames.QuoteChar Constant

Pascal

```
const QuoteChar = 'QuoteChar';
```

C++

```
QuoteChar = 'QuoteChar';
```

Description

This is QuoteChar, a member of class TDBXMetaDatabaseColumnNames.

55.1.42.2.6 TDBXMetaDatabaseColumnNames.QuotePrefix Constant

Pascal

```
const QuotePrefix = 'QuotePrefix';
```

C++

```
QuotePrefix = 'QuotePrefix';
```

Description

This is QuotePrefix, a member of class TDBXMetaDatabaseColumnNames.

55.1.42.2.7 TDBXMetaDatabaseColumnNames.QuoteSuffix Constant

Pascal

```
const QuoteSuffix = 'QuoteSuffix';
```

C++

```
QuoteSuffix = 'QuoteSuffix';
```

Description

This is QuoteSuffix, a member of class TDBXMetaDatabaseColumnNames.

55.1.42.2.8 TDBXMetaDatabaseColumnNames.SupportsLowerCaseIdentifiers Constant

Pascal

```
const SupportsLowerCaseIdentifiers = 'SupportsLowerCaseIdentifiers';
```

C++

```
SupportsLowerCaseIdentifiers = 'SupportsLowerCaseIdentifiers';
```

Description

This is SupportsLowerCaseIdentifiers, a member of class TDBXMetaDatabaseColumnNames.

55.1.42.2.9 TDBXMetaDatabaseColumnNames.SupportsNestedTransactions Constant

Pascal

```
const SupportsNestedTransactions = 'SupportsNestedTransactions';
```

C++

```
SupportsNestedTransactions = 'SupportsNestedTransactions';
```

Description

This is SupportsNestedTransactions, a member of class TDBXMetaDatabaseColumnNames.

55.1.42.2.10 TDBXMetaDatabaseColumnNames.SupportsRowSetSize Constant

Pascal

```
const SupportsRowSetSize = 'SupportsRowSetSize';
```

C++

```
SupportsRowSetSize = 'SupportsRowSetSize';
```

Description

This is SupportsRowSetSize, a member of class TDBXMetaDatabaseColumnNames.

55.1.42.2.11 TDBXMetaDatabaseColumnNames.SupportsTransactions Constant

Pascal

```
const SupportsTransactions = 'SupportsTransactions';
```

C++

```
SupportsTransactions = 'SupportsTransactions';
```

Description

This is SupportsTransactions, a member of class TDBXMetaDatabaseColumnNames.

55.1.42.2.12 TDBXMetaDatabaseColumnNames.SupportsUpperCaseIdentifiers Constant

Pascal

```
const SupportsUpperCaseIdentifiers = 'SupportsUpperCaseIdentifiers';
```

C++

```
SupportsUpperCaseIdentifiers = 'SupportsUpperCaseIdentifiers';
```

Description

This is SupportsUpperCaseIdentifiers, a member of class TDBXMetaDatabaseColumnNames.

55.1.43 TDBXMetaDataCommands Class

List of available metadata commands to get metadata.

Class Hierarchy

DBXCommon.TDBXMetaDataCommands

Pascal

```
TDBXMetaDataCommands = class;
```

C++

```
class TDBXMetaDataCommands;
```

File

DBXCommon

Description

You can use metadata commands to get database metadata, such as a list of tables in a database. Set the TDBXCommand.CommandType property to TDBXCommandTypes.DBXMetadata and set TDBXCommand.Text to one of the constants in the table to acquire the designated metadata. TDBXCommand.ExecuteQuery returns a TDBXReader (see page 10576) to access the metadata.

This table lists the types of metadata you can obtain.

Constant	Description
GetDatabase (see page 10529)	Set TDBXCommand.Text to 'GetDatabase (see page 10529)'. When the command is executed, a TDBXReader (see page 10576) instance is returned with a single row of database specific metadata.
GetDataTypes (see page 10529)	Set TDBXCommand.Text to 'GetDataTypes (see page 10529)'. When the command is executed, a TDBXReader (see page 10576) instance is returned with metadata for the data types supported by the driver. The TDBXDataTypesColumns and TDBXDataTypesIndex classes describe and provide access to this metadata's columns.
GetTables (see page 10531)	Set TDBXCommand.Text to 'GetTables (see page 10531) [catalog.[schema.[table]]] [table-types]'. 'table-types' is a space separated list of one or more of the constants in TDBXMetadataTableTypes. When the command is executed, a TDBXReader (see page 10576) instance is returned with metadata for the specified table(s). The TDBXTablesColumns and TDBXTablesIndex classes describe and provide access to this metadata's columns.
GetColumns (see page 10529)	Set TDBXCommand.Text to 'GetColumns (see page 10529) [catalog.[schema.[table]]]'. When the command is executed, a TDBXReader (see page 10576) instance is returned with metadata for the columns of the specified table(s). The TDBXColumnsColumns and TDBXColumnsIndex classes describe and provide access to this metadata's columns.
GetForeignKeys (see page 10530)	Set TDBXCommand.Text to 'GetForeignKeys (see page 10530) [catalog.[schema.[table]]]'. When the command is executed, a TDBXReader (see page 10576) instance is returned with metadata for the foreign keys of the specified table(s). The TDBXForeignKeysColumns and TDBXForeignKeysIndex classes describe and provide access to this metadata's columns.
GetForeignKeyColumns (see page 10529)	Set TDBXCommand.Text to 'GetForeignKeyColumns (see page 10529) [[[catalog.]schema.]table [foreign-key] [PrimaryKey|ForeignKey]]'[PrimaryKey|ForeignKey]. When the command is executed, a TDBXReader (see page 10576) instance is returned with metadata for each column pair in the specified foreign keys of the specified table(s). The TDBXForeignKeyColumnsColumns and TDBXForeignKeyColumnsIndex classes describe and provide access to this metadata's columns.

GetIndexes (see page 10530)	Set TDBXCommand.Text to 'GetIndexes (see page 10530) [[[catalog.]schema.]table]'. When the command is executed, a TDBXReader (see page 10576) instance is returned with metadata for the indexes of the specified table(s). The TDBXIndexesColumns and TDBXIndexesIndex classes describe and provide access to this metadata's columns.
GetIndexColumns (see page 10530)	Set TDBXCommand.Text to 'GetIndexColumns (see page 10530) [[[catalog.]schema.]table [index]]'. When the command is executed, a TDBXReader (see page 10576) instance is returned with metadata for the columns in the indexes of the specified table(s). The TDBXIndexColumnsColumns and TDBXIndexColumnsIndex classes describe and provide access to this metadata's columns.
GetPackages (see page 10530)	Set TDBXCommand.Text to 'GetPackages (see page 10530)'. When the command is executed, a TDBXReader (see page 10576) instance is returned with metadata for the specified package(s). The TDBXPackagesColumns and TDBXPackagesIndex classes describe and provide access to this metadata's columns.
GetProcedures (see page 10531)	Set TDBXCommand.Text to 'GetProcedures (see page 10531) [catalog.][schema.][procedure]'. When the command is executed, a TDBXReader (see page 10576) instance is returned with metadata for the specified procedure(s). The TDBXProceduresColumns and TDBXProceduresIndex classes describe and provide access to this metadata's columns.
GetProcedureParameters (see page 10531)	Set TDBXCommand.Text to 'GetProcedureParameters (see page 10531) [[[catalog.]schema.]procedure]'. When the command is executed, a TDBXReader (see page 10576) instance is returned with metadata about the parameters of the specified procedure(s). The TDBXProcedureParametersColumns and TDBXProcedureParametersIndex classes describe and provide access to this metadata's columns.
GetUsers (see page 10531)	Set TDBXCommand.Text to 'GetUsers (see page 10531)'. When the command is executed, a TDBXReader (see page 10576) instance is returned with metadata for the specified user(s). The TDBXUsersColumns and TDBXUsersIndex classes describe and provide access to this metadata's columns.

See Also

TDBXCommand (see page 10378)

TDBXCommandTypes (see page 10396)

TDBXColumnsColumns (see page 10695)

TDBXColumnsIndex (see page 10701)

TDBXDataTypesColumns (see page 10707)

TDBXDataTypesIndex (see page 10716)

TDBXForeignKeyColumnsColumns (see page 10725)

TDBXForeignKeyColumnsIndex (see page 10729)

TDBXForeignKeysColumns (see page 10734)

TDBXForeignKeysIndex (see page 10736)

TDBXIndexColumnsColumns (see page 10738)

TDBXIndexColumnsIndex (see page 10741)

TDBXIndexesColumns (see page 10744)

TDBXIndexesIndex (see page 10748)

TDBXMetaDataTableTypes (see page 10540)

TDBXPackagesColumns (see page 10783)

TDBXPackagesIndex (see page 10785)

TDBXProcedureParametersColumns (see page 10792)

TDBXProcedureParametersIndex (see page 10797)

TDBXProceduresColumns (see page 10803)

TDBXProceduresIndex (see page 10805)

TDBXTablesColumns (see page 10827)

TDBXTablesIndex (see page 10829)

TDBXUsersColumns (see page 10832)

TDBXUsersIndex (see page 10833)

ExecuteQuery (see page 10384)

CommandType (see page 10388)

Text (see page 10390)

55.1.43.1 TDBXMetaDataCommands Members

The following tables list the members exposed by TDBXMetaDataCommands.

Public Constants

	Name	Description
	GetColumns (see page 10529)	This is GetColumns, a member of class TDBXMetaDataCommands.
	GetDatabase (see page 10529)	This is GetDatabase, a member of class TDBXMetaDataCommands.
	GetDataTypes (see page 10529)	This is GetDataTypes, a member of class TDBXMetaDataCommands.
	GetForeignKeyColumns (see page 10529)	This is GetForeignKeyColumns, a member of class TDBXMetaDataCommands.
	GetForeignKeys (see page 10530)	This is GetForeignKeys, a member of class TDBXMetaDataCommands.
	GetIndexColumns (see page 10530)	This is GetIndexColumns, a member of class TDBXMetaDataCommands.
	GetIndexes (see page 10530)	This is GetIndexes, a member of class TDBXMetaDataCommands.
	GetPackages (see page 10530)	This is GetPackages, a member of class TDBXMetaDataCommands.
	GetProcedureParameters (see page 10531)	This is GetProcedureParameters, a member of class TDBXMetaDataCommands.
	GetProcedures (see page 10531)	This is GetProcedures, a member of class TDBXMetaDataCommands.
	GetTables (see page 10531)	This is GetTables, a member of class TDBXMetaDataCommands.
	GetUsers (see page 10531)	This is GetUsers, a member of class TDBXMetaDataCommands.

55.1.43.2 TDBXMetaDataCommands Constants

The constants of the TDBXMetaDataCommands class are listed here.

Public Constants

	Name	Description
	GetColumns (see page 10529)	This is GetColumns, a member of class TDBXMetaDataCommands.
	GetDatabase (see page 10529)	This is GetDatabase, a member of class TDBXMetaDataCommands.
	GetDataTypes (see page 10529)	This is GetDataTypes, a member of class TDBXMetaDataCommands.

	GetForeignKeyColumns (see page 10529)	This is GetForeignKeyColumns, a member of class TDBXMetaDataCommands.
	GetForeignKeys (see page 10530)	This is GetForeignKeys, a member of class TDBXMetaDataCommands.
	GetIndexColumns (see page 10530)	This is GetIndexColumns, a member of class TDBXMetaDataCommands.
	GetIndexes (see page 10530)	This is GetIndexes, a member of class TDBXMetaDataCommands.
	GetPackages (see page 10530)	This is GetPackages, a member of class TDBXMetaDataCommands.
	GetProcedureParameters (see page 10531)	This is GetProcedureParameters, a member of class TDBXMetaDataCommands.
	GetProcedures (see page 10531)	This is GetProcedures, a member of class TDBXMetaDataCommands.
	GetTables (see page 10531)	This is GetTables, a member of class TDBXMetaDataCommands.
	GetUsers (see page 10531)	This is GetUsers, a member of class TDBXMetaDataCommands.

55.1.43.2.1 TDBXMetaDataCommands.GetColumns Constant

Pascal

```
const GetColumns = 'GetColumns';
```

C++

```
GetColumns = 'GetColumns';
```

Description

This is GetColumns, a member of class TDBXMetaDataCommands.

55.1.43.2.2 TDBXMetaDataCommands.GetDatabase Constant

Pascal

```
const GetDatabase = 'GetDatabase';
```

C++

```
GetDatabase = 'GetDatabase';
```

Description

This is GetDatabase, a member of class TDBXMetaDataCommands.

55.1.43.2.3 TDBXMetaDataCommands.GetDataTypes Constant

Pascal

```
const GetDataTypes = 'GetDataTypes';
```

C++

```
GetDataTypes = 'GetDataTypes';
```

Description

This is GetDataTypes, a member of class TDBXMetaDataCommands.

55.1.43.2.4 TDBXMetaDataCommands.GetForeignKeyColumns Constant

Pascal

```
const GetForeignKeyColumns = 'GetForeignKeyColumns';
```

C++

```
GetForeignKeyColumns = 'GetForeignKeyColumns';
```

Description

This is GetForeignKeyColumns, a member of class TDBXMetaDataCommands.

55.1.43.2.5 TDBXMetaDataCommands.GetForeignKeys Constant

Pascal

```
const GetForeignKeys = 'GetForeignKeys';
```

C++

```
GetForeignKeys = 'GetForeignKeys';
```

Description

This is GetForeignKeys, a member of class TDBXMetaDataCommands.

55.1.43.2.6 TDBXMetaDataCommands.GetIndexColumns Constant

Pascal

```
const GetIndexColumns = 'GetIndexColumns';
```

C++

```
GetIndexColumns = 'GetIndexColumns';
```

Description

This is GetIndexColumns, a member of class TDBXMetaDataCommands.

55.1.43.2.7 TDBXMetaDataCommands.GetIndexes Constant

Pascal

```
const GetIndexes = 'GetIndexes';
```

C++

```
GetIndexes = 'GetIndexes';
```

Description

This is GetIndexes, a member of class TDBXMetaDataCommands.

55.1.43.2.8 TDBXMetaDataCommands.GetPackages Constant

Pascal

```
const GetPackages = 'GetPackages';
```

C++

```
GetPackages = 'GetPackages';
```

Description

This is GetPackages, a member of class TDBXMetaDataCommands.

55.1.43.2.9 TDBXMetaDataCommands.GetProcedureParameters Constant

Pascal

```
const GetProcedureParameters = 'GetProcedureParameters';
```

C++

```
GetProcedureParameters = 'GetProcedureParameters';
```

Description

This is GetProcedureParameters, a member of class TDBXMetaDataCommands.

55.1.43.2.10 TDBXMetaDataCommands.GetProcedures Constant

Pascal

```
const GetProcedures = 'GetProcedures';
```

C++

```
GetProcedures = 'GetProcedures';
```

Description

This is GetProcedures, a member of class TDBXMetaDataCommands.

55.1.43.2.11 TDBXMetaDataCommands.GetTables Constant

Pascal

```
const GetTables = 'GetTables';
```

C++

```
GetTables = 'GetTables';
```

Description

This is GetTables, a member of class TDBXMetaDataCommands.

55.1.43.2.12 TDBXMetaDataCommands.GetUsers Constant

Pascal

```
const GetUsers = 'GetUsers';
```

C++

```
GetUsers = 'GetUsers';
```

Description

This is GetUsers, a member of class TDBXMetaDataCommands.

55.1.44 TDBXMetaDataCommandsEx Class

Temporary extension of TDBXMetaDataCommands (see page 10525) class.

Class Hierarchy

DBXCommon.TDBXMetaDataCommandsEx

Pascal

```
TDBXMetaDataCommandsEx = class;
```

C++

```
class TDBXMetaDataCommandsEx;
```

File

DBXCommon

Description

You can use metadata commands to get database metadata, such as a list of tables in a database. Set the TDBXCommand.CommandType property to TDBXCommandTypes.DBXMetadata and set TDBXCommand.Text to one of the constants in the table to acquire the designated metadata. TDBXCommand.ExecuteQuery returns a TDBXReader (see page 10576) to access the metadata.

Note: Classes (see page 2685) ending in “Ex” are temporary classes that exist to avoid breaking interfaces. Their contents will be migrated to their base class in the next release.

This table lists the types of extended metadata you can obtain.

Constant	Description
GetCatalogs (see page 10535)	Set TDBXCommand.Text to 'GetCatalogs (see page 10535)'. When the command is executed, a TDBXReader (see page 10576) instance is returned with a list of existing catalogs in the database. The TDBXCatalogsColumns and TDBXCatalogsIndex classes describe and provide access to this metadata's columns.
GetSchemas (see page 10536)	Set TDBXCommand.Text to 'GetSchemas (see page 10536) [catalog]'. When the command is executed, a TDBXReader (see page 10576) instance is returned with a list of existing schemas in the specified catalog(s). The TDBXSchemasColumns and TDBXSchemasIndex classes describe and provide access to this metadata's columns.
GetProcedureSources (see page 10536)	Set TDBXCommand.Text to 'GetProcedureSources (see page 10536) [[[catalog.]schema.]procedure]'. When the command is executed, a TDBXReader (see page 10576) instance is returned with source of the specified procedure(s). The TDBXProcedureSourcesColumns and TDBXProcedureSourcesIndex classes describe and provide access to this metadata's columns.
GetPackageSources (see page 10535)	Set TDBXCommand.Text to 'GetPackageSources (see page 10535) [[[catalog.]schema.]package]'. When the command is executed, a TDBXReader (see page 10576) instance is returned with source of the specified package(s). The TDBXPackageSourcesColumns and TDBXPackageSourcesIndex classes describe and provide access to this metadata's columns.
GetRoles (see page 10536)	Set TDBXCommand.Text to 'GetRoles (see page 10536)'. When the command is executed, a TDBXReader (see page 10576) instance is returned with a list of existing role(s). The TDBXRolesColumns and TDBXRolesIndex classes describe and provide access to this metadata's columns.

GetReservedWords (see page 10536)	Set TDBXCommand.Text to 'GetReservedWords (see page 10536)'. When the command is executed, a TDBXReader (see page 10576) instance is returned with a list of reserved words for the database. The TDBXReservedWordsColumns and TDBXReservedWordsIndex classes describe and provide access to this metadata's columns.
GetViews (see page 10537)	Set TDBXCommand.Text to 'GetViews (see page 10537) [[[catalog.]schema.]view]'. When the command is executed, a TDBXReader (see page 10576) instance is returned with metadata for the view(s) specified. The TDBXViewsColumns and TDBXViewsIndex classes describe and provide access to this metadata's columns.
GetSynonyms (see page 10536)	Set TDBXCommand.Text to 'GetSynonyms (see page 10536) [[[catalog.]schema.]synonym]'. When the command is executed, a TDBXReader (see page 10576) instance is returned with metadata for the synonym(s) specified. The TDBXSynonymsColumns and TDBXSynonymsIndex classes describe and provide access to this metadata's columns.
GetPackageProcedures (see page 10535)	Set TDBXCommand.Text to 'GetPackageProcedures (see page 10535) [[[[[catalog.]schema.]package].procedure][procedure-type]]'.When the command is executed, a TDBXReader (see page 10576) instance is returned with metadata for the specified package procedure(s). The TDBXPackageProceduresColumns and TDBXPackageProceduresIndex classes describe and provide access to this metadata's columns.
GetPackageProcedureParameters (see page 10535)	Set TDBXCommand.Text to 'GetPackageProcedureParameters (see page 10535) [[[catalog.]schema.]package]'. When the command is executed, a TDBXReader (see page 10576) instance is returned with metadata for the parameters of the specified package procedure(s). The TDBXPackageProcedureParametersColumns and TDBXPackageProcedureParametersIndex classes describe and provide access to this metadata's columns.

See Also

TDBXCatalogsColumns (see page 10688)

TDBXCatalogsIndex (see page 10689)

TDBXCommand (see page 10378)

TDBXCommandTypes (see page 10396)

TDBXMetaDataCommands (see page 10525)

TDBXMetaDataTableTypes (see page 10540)

TDBXPackageProcedureParametersColumns (see page 10766)

TDBXPackageProcedureParametersIndex (see page 10772)

TDBXPackageProceduresColumns (see page 10778)

TDBXPackageProceduresIndex (see page 10781)

TDBXPackageSourcesColumns (see page 10787)

TDBXPackageSourcesIndex (see page 10789)

TDBXProcedureSourcesColumns (see page 10807)

TDBXProcedureSourcesIndex (see page 10810)

TDBXReservedWordsColumns (see page 10813)

TDBXReservedWordsIndex (see page 10814)

TDBXRolesColumns (see page 10816)

TDBXRolesIndex (see page 10817)

TDBXSchemasColumns (see page 10818)

TDBXSchemasIndex (see page 10820)

TDBXSynonymsColumns (see page 10822)

TDBXSynonymsIndex (see page 10824)

TDBXViewsColumns (see page 10834)

TDBXViewsIndex (see page 10836)

ExecuteQuery (see page 10384)

CommandType (see page 10388)

Text (see page 10390)

55.1.44.1 TDBXMetaDataCommandsEx Members

The following tables list the members exposed by TDBXMetaDataCommandsEx.

Public Constants

	Name	Description
	GetCatalogs (see page 10535)	This is GetCatalogs, a member of class TDBXMetaDataCommandsEx.
	GetPackageProcedureParameters (see page 10535)	This is GetPackageProcedureParameters, a member of class TDBXMetaDataCommandsEx.
	GetPackageProcedures (see page 10535)	This is GetPackageProcedures, a member of class TDBXMetaDataCommandsEx.
	GetPackageSources (see page 10535)	This is GetPackageSources, a member of class TDBXMetaDataCommandsEx.
	GetProcedureSources (see page 10536)	This is GetProcedureSources, a member of class TDBXMetaDataCommandsEx.
	GetReservedWords (see page 10536)	This is GetReservedWords, a member of class TDBXMetaDataCommandsEx.
	GetRoles (see page 10536)	This is GetRoles, a member of class TDBXMetaDataCommandsEx.
	GetSchemas (see page 10536)	This is GetSchemas, a member of class TDBXMetaDataCommandsEx.
	GetSynonyms (see page 10536)	This is GetSynonyms, a member of class TDBXMetaDataCommandsEx.
	GetViews (see page 10537)	This is GetViews, a member of class TDBXMetaDataCommandsEx.

55.1.44.2 TDBXMetaDataCommandsEx Constants

The constants of the TDBXMetaDataCommandsEx class are listed here.

Public Constants

	Name	Description
	GetCatalogs (see page 10535)	This is GetCatalogs, a member of class TDBXMetaDataCommandsEx.
	GetPackageProcedureParameters (see page 10535)	This is GetPackageProcedureParameters, a member of class TDBXMetaDataCommandsEx.
	GetPackageProcedures (see page 10535)	This is GetPackageProcedures, a member of class TDBXMetaDataCommandsEx.
	GetPackageSources (see page 10535)	This is GetPackageSources, a member of class TDBXMetaDataCommandsEx.

	GetProcedureSources (see page 10536)	This is GetProcedureSources, a member of class TDBXMetaDataCommandsEx.
	GetReservedWords (see page 10536)	This is GetReservedWords, a member of class TDBXMetaDataCommandsEx.
	GetRoles (see page 10536)	This is GetRoles, a member of class TDBXMetaDataCommandsEx.
	GetSchemas (see page 10536)	This is GetSchemas, a member of class TDBXMetaDataCommandsEx.
	GetSynonyms (see page 10536)	This is GetSynonyms, a member of class TDBXMetaDataCommandsEx.
	GetViews (see page 10537)	This is GetViews, a member of class TDBXMetaDataCommandsEx.

55.1.44.2.1 TDBXMetaDataCommandsEx.GetCatalogs Constant

Pascal

```
const GetCatalogs = 'GetCatalogs';
```

C++

```
GetCatalogs = 'GetCatalogs';
```

Description

This is GetCatalogs, a member of class TDBXMetaDataCommandsEx.

55.1.44.2.2 TDBXMetaDataCommandsEx.GetPackageProcedureParameters Constant

Pascal

```
const GetPackageProcedureParameters = 'GetPackageProcedureParameters';
```

C++

```
GetPackageProcedureParameters = 'GetPackageProcedureParameters';
```

Description

This is GetPackageProcedureParameters, a member of class TDBXMetaDataCommandsEx.

55.1.44.2.3 TDBXMetaDataCommandsEx.GetPackageProcedures Constant

Pascal

```
const GetPackageProcedures = 'GetPackageProcedures';
```

C++

```
GetPackageProcedures = 'GetPackageProcedures';
```

Description

This is GetPackageProcedures, a member of class TDBXMetaDataCommandsEx.

55.1.44.2.4 TDBXMetaDataCommandsEx.GetPackageSources Constant

Pascal

```
const GetPackageSources = 'GetPackageSources';
```

C++

```
GetPackageSources = 'GetPackageSources';
```

Description

This is GetPackageSources, a member of class TDBXMetaDataCommandsEx.

55.1.44.2.5 TDBXMetaDataCommandsEx.GetProcedureSources Constant

Pascal

```
const GetProcedureSources = 'GetProcedureSources';
```

C++

```
GetProcedureSources = 'GetProcedureSources';
```

Description

This is GetProcedureSources, a member of class TDBXMetaDataCommandsEx.

55.1.44.2.6 TDBXMetaDataCommandsEx.GetReservedWords Constant

Pascal

```
const GetReservedWords = 'GetReservedWords';
```

C++

```
GetReservedWords = 'GetReservedWords';
```

Description

This is GetReservedWords, a member of class TDBXMetaDataCommandsEx.

55.1.44.2.7 TDBXMetaDataCommandsEx.GetRoles Constant

Pascal

```
const GetRoles = 'GetRoles';
```

C++

```
GetRoles = 'GetRoles';
```

Description

This is GetRoles, a member of class TDBXMetaDataCommandsEx.

55.1.44.2.8 TDBXMetaDataCommandsEx.GetSchemas Constant

Pascal

```
const GetSchemas = 'GetSchemas';
```

C++

```
GetSchemas = 'GetSchemas';
```

Description

This is GetSchemas, a member of class TDBXMetaDataCommandsEx.

55.1.44.2.9 TDBXMetaDataCommandsEx.GetSynonyms Constant

Pascal

```
const GetSynonyms = 'GetSynonyms';
```

C++

```
GetSynonyms = 'GetSynonyms';
```

Description

This is GetSynonyms, a member of class TDBXMetaDataCommandsEx.

55.1.44.2.10 TDBXMetaDataCommandsEx.GetViews Constant

Pascal

```
const GetViews = 'GetViews';
```

C++

```
GetViews = 'GetViews';
```

Description

This is GetViews, a member of class TDBXMetaDataCommandsEx.

55.1.45 TDBXMetaDataKeyword Class

Keyword for metadata.

Class Hierarchy

DBXCommon.TDBXMetaDataKeyword

Pascal

```
TDBXMetaDataKeyword = class;
```

C++

```
class TDBXMetaDataKeyword;
```

File

DBXCommon

Description

TDBXMetaDataKeyword contains constants for metadata keywords.

This table contains metadata keywords.

Constant	Description
PrimaryKey (see page 10538)	PrimaryKey (see page 10538)
ForeignKey (see page 10538)	ForeignKey (see page 10538)

55.1.45.1 TDBXMetaDataKeyword Members

The following tables list the members exposed by TDBXMetaDataKeyword.

Public Constants

	Name	Description
	ForeignKey (see page 10538)	This is ForeignKey, a member of class TDBXMetaDataKeyword.
	PrimaryKey (see page 10538)	This is PrimaryKey, a member of class TDBXMetaDataKeyword.

55.1.45.2 TDBXMetaDataKeyword Constants

The constants of the TDBXMetaDataKeyword class are listed here.

Public Constants

	Name	Description
	ForeignKey (see page 10538)	This is ForeignKey, a member of class TDBXMetaDataKeyword.
	PrimaryKey (see page 10538)	This is PrimaryKey, a member of class TDBXMetaDataKeyword.

55.1.45.2.1 TDBXMetaDataKeyword.ForeignKey Constant

Pascal

```
const ForeignKey = 'ForeignKey';
```

C++

```
ForeignKey = 'ForeignKey';
```

Description

This is ForeignKey, a member of class TDBXMetaDataKeyword.

55.1.45.2.2 TDBXMetaDataKeyword.PrimaryKey Constant

Pascal

```
const PrimaryKey = 'PrimaryKey';
```

C++

```
PrimaryKey = 'PrimaryKey';
```

Description

This is PrimaryKey, a member of class TDBXMetaDataKeyword.

55.1.46 TDBXMetaDataProcedureTypes Class

Constants passed to TDBXMetaDataCommands.GetProcedures metadata command.

Class Hierarchy

Pascal

```
TDBXMetaDataProcedureTypes = class;
```

C++

```
class TDBXMetaDataProcedureTypes;
```

File

DBXCommon

Description

One or more of these constants in TDBXMetaDataProcedureTypes may be passed to the

TDBXMetaDataCommands.GetProcedures metadata command.

This table contains the procedure types.

Constant	Description
ProcedureType (see page 10539)	Get procedures.
FunctionType (see page 10539)	Get functions.

See Also

TDBXMetaDataCommands (see page 10525)

55.1.46.1 TDBXMetaDataProcedureTypes Members

The following tables list the members exposed by TDBXMetaDataProcedureTypes.

Public Constants

	Name	Description
	FunctionType (see page 10539)	This is FunctionType, a member of class TDBXMetaDataProcedureTypes.
	ProcedureType (see page 10539)	This is ProcedureType, a member of class TDBXMetaDataProcedureTypes.

55.1.46.2 TDBXMetaDataProcedureTypes Constants

The constants of the TDBXMetaDataProcedureTypes class are listed here.

Public Constants

	Name	Description
	FunctionType (see page 10539)	This is FunctionType, a member of class TDBXMetaDataProcedureTypes.
	ProcedureType (see page 10539)	This is ProcedureType, a member of class TDBXMetaDataProcedureTypes.

55.1.46.2.1 TDBXMetaDataProcedureTypes.FunctionType Constant

Pascal

```
const FunctionType = 'Function';
```

C++

```
FunctionType = 'Function';
```

Description

This is FunctionType, a member of class TDBXMetaDataProcedureTypes.

55.1.46.2.2 TDBXMetaDataProcedureTypes.ProcedureType Constant

Pascal

```
const ProcedureType = 'Procedure';
```

C++

```
ProcedureType = 'Procedure';
```

Description

This is ProcedureType, a member of class TDBXMetaDataProcedureTypes.

55.1.47 TDBXMetaDataTableTypes Class

Constants for the GetTables metadata command

Class Hierarchy

DBXCommon.TDBXMetaDataTableTypes

Pascal

```
TDBXMetaDataTableTypes = class;
```

C++

```
class TDBXMetaDataTableTypes;
```

File

DBXCommon

Description

The TDBXMetaDataCommands (see page 10525) class lists types of database metadata you can acquire. This class's constants may be passed to the GetTables command in TDBXMetaDataCommands (see page 10525).

This table lists the tables for which you can get metadata.

Constant	Description
Table (see page 10541)	Get tables
View (see page 10541)	Get Views
SystemTable (see page 10541)	Get System (see page 22079) tables
Synonym (see page 10541)	Get synonyms

See Also

TDBXMetaDataCommands (see page 10525)

55.1.47.1 TDBXMetaDataTableTypes Members

The following tables list the members exposed by TDBXMetaDataTableTypes.

Public Constants

	Name	Description
	Synonym (see page 10541)	This is Synonym, a member of class TDBXMetaDataTableTypes.
	SystemTable (see page 10541)	This is SystemTable, a member of class TDBXMetaDataTableTypes.
	Table (see page 10541)	This is Table, a member of class TDBXMetaDataTableTypes.
	View (see page 10541)	This is View, a member of class TDBXMetaDataTableTypes.

55.1.47.2 TDBXMetaDataTableTypes Constants

The constants of the TDBXMetaDataTableTypes class are listed here.

Public Constants

	Name	Description
	Synonym (see page 10541)	This is Synonym, a member of class TDBXMetaDataTableTypes.
	SystemTable (see page 10541)	This is SystemTable, a member of class TDBXMetaDataTableTypes.
	Table (see page 10541)	This is Table, a member of class TDBXMetaDataTableTypes.
	View (see page 10541)	This is View, a member of class TDBXMetaDataTableTypes.

55.1.47.2.1 TDBXMetaDataTableTypes.Synonym Constant

Pascal

```
const Synonym = 'Synonym';
```

C++

```
Synonym = 'Synonym';
```

Description

This is Synonym, a member of class TDBXMetaDataTableTypes.

55.1.47.2.2 TDBXMetaDataTableTypes.SystemTable Constant

Pascal

```
const SystemTable = 'SystemTable';
```

C++

```
SystemTable = 'SystemTable';
```

Description

This is SystemTable, a member of class TDBXMetaDataTableTypes.

55.1.47.2.3 TDBXMetaDataTableTypes.Table Constant

Pascal

```
const Table = 'Table';
```

C++

```
Table = 'Table';
```

Description

This is Table, a member of class TDBXMetaDataTableTypes.

55.1.47.2.4 TDBXMetaDataTableTypes.View Constant

Pascal

```
const View = 'View';
```

C++

```
View = 'View';
```

Description

This is View, a member of class TDBXMetaDataTableTypes.

55.1.48 TDBXMetaDataTableTypesEx Class

Temporary extension of TDBXMetaDataTableTypes (see page 10540) class.

Class Hierarchy

DBXCommon.TDBXMetaDataTableTypesEx

Pascal

```
TDBXMetaDataTableTypesEx = class;
```

C++

```
class TDBXMetaDataTableTypesEx;
```

File

DBXCommon

Description

The TDBXMetaDataCommandsEx (see page 10531) class lists additional types of database metadata you can acquire. This class's constants may be passed to the GetTables command in TDBXMetaDataCommands (see page 10525).

Note: Classes (see page 2685) ending in “Ex” are temporary classes that exist to avoid breaking interfaces. Their contents will be migrated to their base class in the next release.

This table contains tables for which you can get metadata.

Constant	Description
SystemView (see page 10543)	SystemView (see page 10543)

See Also

TDBXMetaDataTableTypes (see page 10540)

TDBXMetaDataCommands (see page 10525)

55.1.48.1 TDBXMetaDataTableTypesEx Members

The following tables list the members exposed by TDBXMetaDataTableTypesEx.

Public Constants

	Name	Description
	SystemView (see page 10543)	This is SystemView, a member of class TDBXMetaDataTableTypesEx.

55.1.48.2 TDBXMetaDataTableTypesEx Constants

The constants of the TDBXMetaDataTableTypesEx class are listed here.

Public Constants

	Name	Description
	SystemView (see page 10543)	This is SystemView, a member of class TDBXMetaDataTableTypesEx.

55.1.48.2.1 TDBXMetaDataTableTypesEx.SystemView Constant

Pascal

```
const SystemView = 'SystemView';
```

C++

```
SystemView = 'SystemView';
```

Description

This is SystemView, a member of class TDBXMetaDataTableTypesEx.

55.1.49 TDBXNullValue Class

Class Hierarchy

Pascal

```
TDBXNullValue = class(TDBXWritableValue);
```

C++

```
class TDBXNullValue : public TDBXWritableValue;
```

File

DBXCommon

Description

This is class DBXCommon.TDBXNullValue.

55.1.49.1 TDBXNullValue Members

The following tables list the members exposed by TDBXNullValue.

Protected Methods

	Name	Description
	Create (see page 10544)	Creates an instance of TDBXWritableValue
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.
V	SetPendingValue (see page 10666)	This is SetPendingValue, a member of class TDBXWritableValue.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Methods

	Name	Description
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
V	Destroy (see page 10544)	Destroys a TDBXWritableValue instance.
V	GetAnsiString (see page 10629)	Gets the string value for ANSI strings.
V	GetBcd (see page 10629)	Gets the BCD value for binary-compressed decimal data types.

	GetBoolean (see page 10629)	Gets the boolean value for boolean data types.
	GetBytes (see page 10629)	Gets the byte array value for byte array and blob types.
	GetDate (see page 10630)	Gets the date value for date data types.
	GetDouble (see page 10630)	Gets the double floating point value for double data types.
	GetInt16 (see page 10631)	Gets the smallInt value for 16-byte integer data types.
	GetInt32 (see page 10631)	Gets the 32-byte integer value for 32-byte integer data types.
	GetInt64 (see page 10631)	Gets the 64-byte integer value for 64-byte integer data types.
	GetTime (see page 10631)	Gets the time value for time data types.
	GetTimeStamp (see page 10632)	Gets the timestamp value for timestamp data types.
	GetValueSize (see page 10632)	Gets the size in bytes for values.
	GetWideString (see page 10632)	Gets the wide string value for wide string data types.
	IsNull (see page 10545)	Returns true if the value is **null**.
	SetAnsiString (see page 10663)	Sets the string value for ANSI strings.
	SetBcd (see page 10664)	Sets the BCD value for binary-compressed decimal data types.
	SetBoolean (see page 10664)	Sets the boolean value for boolean data types..
	SetDate (see page 10664)	Sets the date value for date data types..
	SetDouble (see page 10664)	Sets the double floating point value for double data types.
	SetDynamicBytes (see page 10665)	Sets the dynamic array value for byte array and blob types.
	SetInt16 (see page 10665)	Sets the the smallInt value for 16-byte integer data types.
	SetInt32 (see page 10665)	Sets the 32-byte integer value for 32-byte integer data types.
	SetInt64 (see page 10665)	Sets the 64-byte integer value for 64-byte integer data types.
	SetNull (see page 10666)	Sets a writable value of any type to null.
	SetStaticBytes (see page 10666)	Sets the open array value for byte array and blob types
	SetTime (see page 10666)	Sets the time value for time data types.
	SetTimeStamp (see page 10667)	Sets the timestamp value for timestamp data types.
	SetWideString (see page 10667)	Sets the wide string value for wide string data types.

Public Properties

	Name	Description
	ValueType (see page 10633)	Gets the value type for this value.

55.1.49.2 TDBXNullValue.Create Constructor

Creates an instance of TDBXWritableValue

Pascal

```
constructor Create(ValueType: TDBXValueType);
```

C++

```
__fastcall TDBXNullValue(TDBXValueType ValueType);
```

Description

This constructor creates and initializes an instance of TDBXWritableValue.

See Also

Destroy (see page 10663)

55.1.49.3 TDBXNullValue.Destroy Destructor

Destroys a TDBXWritableValue instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXNullValue();
```

Description

This destructor destroys an instance of TDBXWritableValue.

See Also

Create (see page 10662)

55.1.49.4 TDBXNullValue Methods

The methods of the TDBXNullValue class are listed here.

Public Methods

	Name	Description
	IsNull (see page 10545)	Returns true if the value is **null**.

55.1.49.4.1 TDBXNullValue.IsNull Method

Returns true if the value is **null**.

Pascal

```
function IsNull: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsNull();
```

Description

Returns true if the value is **null**. This method should be called before calling any of the Get methods. If the value is null, the Get methods will throw TDBXError exception if they are called.

55.1.50 TDBXParameter Class

Individual parameters for queries and stored commands.

Class Hierarchy

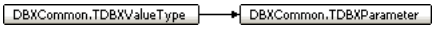

Pascal

```
TDBXParameter = class(TDBXValueType);
```

C++

```
class TDBXParameter : public TDBXValueType;
```

File

DBXCommon

Description

TDBXParameter should be created by calling CreateParameter for each "?" parameter marker specified in the Text property. Once a TDBXParameter is created, it must be added to a TDBXCommand (see page 10378) parameter list by calling

.AddParameter method.

55.1.50.1 TDBXParameter Members

The following tables list the members exposed by TDBXParameter.

Protected Methods

	Name	Description
	Create (see page 10547)	Creates an instance of TDBXParameter (see page 10545)
V	GetChildPosition (see page 10638)	Get ChildPosition (see page 10645) property.
V	GetDataType (see page 10639)	Get DataType (see page 10646) property.
V	GetFlags (see page 10639)	Get ValueTypeFlags (see page 10648) property.
V	GetName (see page 10639)	Get Name (see page 10646) property.
V	GetOrdinal (see page 10639)	Get Ordinal (see page 10646) property.
V	GetParameterDirection (see page 10640)	Get ParameterDirection (see page 10647) property.
V	GetPrecision (see page 10640)	Get Precision (see page 10647) property.
V	GetScale (see page 10640)	Get Scale (see page 10647) property.
V	GetSize (see page 10641)	Get Size (see page 10648) property.
V	GetSubType (see page 10641)	Get SubType (see page 10648) property.
V	GetValue (see page 10547)	This is GetValue, a member of class TDBXParameter.
V	SetChildPosition (see page 10641)	Set ChildPosition (see page 10645) property.
V	SetDataType (see page 10642)	Set DataType (see page 10646) property.
V	SetDbxRow (see page 10548)	This is SetDbxRow, a member of class TDBXParameter.
V	SetFlags (see page 10642)	Set ValueTypeFlags (see page 10648) property.
V	SetName (see page 10642)	Set Name (see page 10646) property.
V	SetNullable (see page 10642)	Set ChildPosition (see page 10645) property.
V	SetOrdinal (see page 10643)	Set Ordinal (see page 10646) property.
V	SetParameter (see page 10548)	This is SetParameter, a member of class TDBXParameter.
V	SetParameterDirection (see page 10643)	Set ParameterDirection (see page 10647) property.
V	SetPrecision (see page 10643)	Set Precision (see page 10647) property.
V	SetScale (see page 10644)	Set Scale (see page 10647) property.
V	SetSize (see page 10644)	Set Size (see page 10648) property.
V	SetSubType (see page 10644)	Set SubType (see page 10648) property.

Public Methods

	Name	Description
V	Destroy (see page 10547)	Destructor.

Public Properties

	Name	Description
R	AutoIncrement (see page 10645)	Gets the value that determines whether or not this describes an autoincrement value.
	ChildPosition (see page 10645)	Used by TDBXParameter (see page 10545) instances. Child position of Oracle ADT and Array types
	DataType (see page 10646)	Value data type.
	Name (see page 10646)	Name for this value.
	Nullable (see page 10646)	Determines whether or not a parameter can be set to null.
	Ordinal (see page 10646)	Ordinal position of this value.
	ParameterDirection (see page 10647)	Gets or sets the direction of a parameter.
	Precision (see page 10647)	The maximum number of digits allowed in the parameter.
R	ReadOnly (see page 10647)	True if this TDBXValueType (see page 10636) is read-only.
	Scale (see page 10647)	The maximum number of digits used to represent the property.
R	Searchable (see page 10648)	True if this TDBXValueType (see page 10636) is searchable.
	Size (see page 10648)	The size of the parameter value in bytes.

	SubType (see page 10648)	Data sub-type.
R	Value (see page 10548)	Used to read and write parameter values.
	ValueTypeFlags (see page 10648)	TDBXValue (see page 10626) metadata for TDBXReader (see page 10576).

Strict protected Fields

	Name	Description
	FDBXContext (see page 10637)	This is FDBXContext, a member of class TDBXValueType.

55.1.50.2 TDBXParameter.Create Constructor

Creates an instance of TDBXParameter (see page 10545)

Pascal

```
constructor Create(DbxContext: TDBXContext);
```

C++

```
__fastcall TDBXParameter(TDBXContext DbxContext);
```

Description

This constructor creates and initializes an instance of TDBXParameter (see page 10545).

See Also

Destroy (see page 10547)

55.1.50.3 TDBXParameter.Destroy Destructor

Destructor.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXParameter();
```

Description

Use Destroy to destroy an instance of TDBXParameter (see page 10545).

55.1.50.4 TDBXParameter Methods

The methods of the TDBXParameter class are listed here.

Protected Methods

	Name	Description
V	GetValue (see page 10547)	This is GetValue, a member of class TDBXParameter.
V	SetDbxRow (see page 10548)	This is SetDbxRow, a member of class TDBXParameter.
V	SetParameter (see page 10548)	This is SetParameter, a member of class TDBXParameter.

55.1.50.4.1 TDBXParameter.GetValue Method

Pascal

```
function GetValue: TDBXWritableValue; virtual;
```

C++

```
virtual __fastcall TDBXWritableValue GetValue();
```

Description

This is GetValue, a member of class TDBXParameter.

55.1.50.4.2 TDBXParameter.SetDbxRow Method

Pascal

```
procedure SetDbxRow(DbxRow: TDBXRow); virtual;
```

C++

```
virtual __fastcall SetDbxRow(TDBXRow DbxRow);
```

Description

This is SetDbxRow, a member of class TDBXParameter.

55.1.50.4.3 TDBXParameter.SetParameter Method

Pascal

```
procedure SetParameter; virtual;
```

C++

```
virtual __fastcall SetParameter();
```

Description

This is SetParameter, a member of class TDBXParameter.

55.1.50.5 TDBXParameter Properties

The properties of the TDBXParameter class are listed here.

Public Properties

	Name	Description
R	Value (see page 10548)	Used to read and write parameter values.

55.1.50.5.1 TDBXParameter.Value Property

Used to read and write parameter values.

Pascal

```
property Value: TDBXWritableValue;
```

C++

```
__property TDBXWritableValue Value;
```

Description

Returns a TDBXWritableValueinstance that can be used to read and write parameter values.

55.1.51 TDBXParameterDirections Class

Parameter types supported by TDBXParameter (see page 10545)

Class Hierarchy

DBXCommon.TDBXParameterDirections

Pascal

```
TDBXParameterDirections = class;
```

C++

```
class TDBXParameterDirections;
```

File

DBXCommon

Description

Parameter types supported by the TDBXParameter (see page 10545) class.

This table contains parameter direction constants.

Constant	Description
Unknown (see page 10551)	The parameter type is unknown (generates an error).
InParameter (see page 10550)	Input parameter.
OutParameter (see page 10550)	Output parameter.
InOutParameter (see page 10550)	Input/Output parameter.
ReturnParameter (see page 10550)	Return parameter.

See Also

TDBXParameter (see page 10545)

55.1.51.1 TDBXParameterDirections Members

The following tables list the members exposed by TDBXParameterDirections.

Public Constants

	Name	Description
	InOutParameter (see page 10550)	This is InOutParameter, a member of class TDBXParameterDirections.
	InParameter (see page 10550)	This is InParameter, a member of class TDBXParameterDirections.
	OutParameter (see page 10550)	This is OutParameter, a member of class TDBXParameterDirections.
	ReturnParameter (see page 10550)	This is ReturnParameter, a member of class TDBXParameterDirections.
	Unknown (see page 10551)	This is Unknown, a member of class TDBXParameterDirections.

55.1.51.2 TDBXParameterDirections Constants

The constants of the TDBXParameterDirections class are listed here.

Public Constants

	Name	Description
	InOutParameter (see page 10550)	This is InOutParameter, a member of class TDBXParameterDirections.
	InParameter (see page 10550)	This is InParameter, a member of class TDBXParameterDirections.
	OutParameter (see page 10550)	This is OutParameter, a member of class TDBXParameterDirections.
	ReturnParameter (see page 10550)	This is ReturnParameter, a member of class TDBXParameterDirections.
	Unknown (see page 10551)	This is Unknown, a member of class TDBXParameterDirections.

55.1.51.2.1 TDBXParameterDirections.InOutParameter Constant

Pascal

```
const InOutParameter = 3;
```

C++

```
InOutParameter = 3;
```

Description

This is InOutParameter, a member of class TDBXParameterDirections.

55.1.51.2.2 TDBXParameterDirections.InParameter Constant

Pascal

```
const InParameter = 1;
```

C++

```
InParameter = 1;
```

Description

This is InParameter, a member of class TDBXParameterDirections.

55.1.51.2.3 TDBXParameterDirections.OutParameter Constant

Pascal

```
const OutParameter = 2;
```

C++

```
OutParameter = 2;
```

Description

This is OutParameter, a member of class TDBXParameterDirections.

55.1.51.2.4 TDBXParameterDirections.ReturnParameter Constant

Pascal

```
const ReturnParameter = 4;
```

C++

```
ReturnParameter = 4;
```

Description

This is ReturnParameter, a member of class TDBXParameterDirections.

55.1.51.2.5 TDBXParameterDirections.Unknown Constant

Pascal

```
const Unknown = 0;
```

C++

```
Unknown = 0;
```

Description

This is Unknown, a member of class TDBXParameterDirections.

55.1.52 TDBXParameterEx Class

Extension of TDBXParameter (see page 10545).

Class Hierarchy

Pascal

```
TDBXParameterEx = class(TDBXParameter);
```

C++

```
class TDBXParameterEx : public TDBXParameter;
```

File

DBXCommon

Description

TDBXParameterEx is an extension of TDBXParameter (see page 10545).

Note: Classes (see page 2685) ending in “Ex” are temporary classes that exist to avoid breaking interfaces. Their contents will be migrated to their base class in the next release.

See Also

TDBXParameter (see page 10545)

55.1.52.1 TDBXParameterEx Members

The following tables list the members exposed by TDBXParameterEx.

Protected Methods

	Name	Description
	Create (see page 10547)	Creates an instance of TDBXParameter (see page 10545)
	Create (see page 10637)	Creates an instance of TDBXValueType (see page 10636)
V	GetChildPosition (see page 10638)	Get ChildPosition (see page 10645) property.
V	GetDataType (see page 10639)	Get DataType (see page 10646) property.
V	GetFlags (see page 10639)	Get ValueTypeFlags (see page 10648) property.
V	GetName (see page 10639)	Get Name (see page 10646) property.
V	GetOrdinal (see page 10639)	Get Ordinal (see page 10646) property.
V	GetParameterDirection (see page 10640)	Get ParameterDirection (see page 10647) property.

	GetPrecision (see page 10640)	Get Precision (see page 10647) property.
	GetScale (see page 10640)	Get Scale (see page 10647) property.
	GetSize (see page 10641)	Get Size (see page 10648) property.
	GetSubType (see page 10641)	Get SubType (see page 10648) property.
	GetValue (see page 10547)	This is GetValue, a member of class TDBXParameter.
	SetChildPosition (see page 10641)	Set ChildPosition (see page 10645) property.
	SetDataType (see page 10642)	Set DataType (see page 10646) property.
	SetDbxRow (see page 10548)	This is SetDbxRow, a member of class TDBXParameter.
	SetFlags (see page 10642)	Set ValueTypeFlags (see page 10648) property.
	SetName (see page 10642)	Set Name (see page 10646) property.
	SetNullable (see page 10642)	Set ChildPosition (see page 10645) property.
	SetOrdinal (see page 10643)	Set Ordinal (see page 10646) property.
	SetParameter (see page 10548)	This is SetParameter, a member of class TDBXParameter.
	SetParameterDirection (see page 10643)	Set ParameterDirection (see page 10647) property.
	SetPrecision (see page 10643)	Set Precision (see page 10647) property.
	SetScale (see page 10644)	Set Scale (see page 10647) property.
	SetSize (see page 10644)	Set Size (see page 10648) property.
	SetSubType (see page 10644)	Set SubType (see page 10648) property.

Public Methods

	Name	Description
	Assign (see page 10553)	Copies field values from one parameter to other.
	Clone (see page 10553)	Creates new TDBXParameter (see page 10545) instance and copies field values to new instance.
	Destroy (see page 10547)	Destructor.

Public Properties

	Name	Description
R	AutoIncrement (see page 10645)	Gets the value that determines whether or not this describes an autoincrement value.
	ChildPosition (see page 10645)	Used by TDBXParameter (see page 10545) instances. Child position of Oracle ADT and Array types
	DataType (see page 10646)	Value data type.
	Name (see page 10646)	Name for this value.
	Nullable (see page 10646)	Determines whether or not a parameter can be set to null.
	Ordinal (see page 10646)	Ordinal position of this value.
	ParameterDirection (see page 10647)	Gets or sets the direction of a parameter.
	Precision (see page 10647)	The maximum number of digits allowed in the parameter.
R	ReadOnly (see page 10647)	True if this TDBXValueType (see page 10636) is read-only.
	Scale (see page 10647)	The maximum number of digits used to represent the property.
R	Searchable (see page 10648)	True if this TDBXValueType (see page 10636) is searchable.
	Size (see page 10648)	The size of the parameter value in bytes.
	SubType (see page 10648)	Data sub-type.
R	Value (see page 10548)	Used to read and write parameter values.
	ValueTypeFlags (see page 10648)	TDBXValue (see page 10626) metadata for TDBXReader (see page 10576).

Strict protected Fields

	Name	Description
	FDBXContext (see page 10637)	This is FDBXContext, a member of class TDBXValueType.

55.1.52.2 TDBXParameterEx Methods

The methods of the TDBXParameterEx class are listed here.

Public Methods

	Name	Description
	Assign (see page 10553)	Copies field values from one parameter to other.
	Clone (see page 10553)	Creates new TDBXParameter (see page 10545) instance and copies field values to new instance.

55.1.52.2.1 TDBXParameterEx.Assign Method

Copies field values from one parameter to other.

Pascal

```
procedure Assign(Source: TDBXParameter); virtual;
```

C++

```
virtual __fastcall Assign(TDBXParameter Source);
```

Description

Assign copies field values from one parameter to the other.

See Also

TDBXParameter (see page 10545)

55.1.52.2.2 TDBXParameterEx.Clone Method

Creates new TDBXParameter (see page 10545) instance and copies field values to new instance.

Pascal

```
function Clone: TObject; virtual;
```

C++

```
virtual __fastcall TObject * Clone();
```

Description

Clone creates a new TDBXParameter (see page 10545) instance and calls Assign (see page 10553) to copy field values to new instance.

See Also

TDBXParameter (see page 10545)

Assign (see page 10553)

55.1.53 TDBXParameterList Class

A collection of TDBXParameter (see page 10545) instances that can be used to get and set parameters for a TDBXCommand (see page 10378).

Class Hierarchy

DBXCommon.TDBXParameterList

Pascal

```
TDBXParameterList = class;
```

C++

```
class TDBXParameterList;
```

File

DBXCommon

Description

The Parameters property returns an instance of TDBXParameterList. The TDBXParameterList manages a collection of TDBXParameter (see page 10545) instances that can be used to get and set parameters for a TDBXCommand (see page 10378). The lifetime of all TDBXParameter (see page 10545) instances added to this list will be managed by this list. So when Free is called, Free will also be called for all items in the list.

55.1.53.1 TDBXParameterList Members

The following tables list the members exposed by TDBXParameterList.

Protected Fields

	Name	Description
	FDBXContext (see page 10555)	This is FDBXContext, a member of class TDBXParameterList.

Protected Methods

	Name	Description
	Create (see page 10554)	Creates an instance of TDBXParameterList (see page 10553)
V	GetCount (see page 10556)	Number parameters in list.
V	GetParameterByOrdinal (see page 10557)	Get parameter by ordinal position.

Public Methods

	Name	Description
V	AddParameter (see page 10556)	Add a parameter to this list.
V	ClearParameters (see page 10556)	Remove all parameters.
V	Destroy (see page 10555)	Destroys a TDBXParameterList (see page 10553) instance.
V	GetOrdinal (see page 10556)	The ordinal index for the named parameter.
V	InsertParameter (see page 10557)	Inserts parameter into this list at the specified ordinal position.
V	RemoveParameter (see page 10557)	This is the overview for the RemoveParameter method overload.
V	SetCount (see page 10558)	Sets the length of the list to the specified number of parameters.
V	SetParameter (see page 10558)	Add a parameter to this list at the specified ordinal position.

Public Properties

	Name	Description
R	Count (see page 10559)	The number of parameters in this list.
R	Parameter (see page 10559)	Individual; TDBXParameter (see page 10545) instance.

55.1.53.2 TDBXParameterList.Create Constructor

Creates an instance of TDBXParameterList (see page 10553)

Pascal

```
constructor Create(DBXContext: TDBXContext; Command: TDBXCommand);
```

C++

```
__fastcall TDBXParameterList(TDBXContext DBXContext, TDBXCommand Command);
```

Description

This constructor creates and initializes an instance of TDBXParameterList (see page 10553).

See Also

Destroy (see page 10555)

55.1.53.3 TDBXParameterList.Destroy Destructor

Destroys a TDBXParameterList (see page 10553) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXParameterList();
```

Description

This destructor destroys an instance of TDBXParameterList (see page 10553).

See Also

Create (see page 10554)

55.1.53.4 TDBXParameterList Fields

The fields of the TDBXParameterList class are listed here.

Protected Fields

	Name	Description
	FDBXContext (see page 10555)	This is FDBXContext, a member of class TDBXParameterList.

55.1.53.4.1 TDBXParameterList.FDBXContext Field

Pascal

```
FDBXContext: TDBXContext;
```

C++

```
TDBXContext FDBXContext;
```

Description

This is FDBXContext, a member of class TDBXParameterList.

55.1.53.5 TDBXParameterList Methods

The methods of the TDBXParameterList class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 10556)	Number parameters in list.
	GetParameterByOrdinal (see page 10557)	Get parameter by ordinal position.

Public Methods

	Name	Description
	AddParameter (see page 10556)	Add a parameter to this list.
	ClearParameters (see page 10556)	Remove all parameters.
	GetOrdinal (see page 10556)	The ordinal index for the named parameter.
	InsertParameter (see page 10557)	Inserts parameter into this list at the specified ordinal position.
	RemoveParameter (see page 10557)	This is the overview for the RemoveParameter method overload.
	SetCount (see page 10558)	Sets the length of the list to the specified number of parameters.
	SetParameter (see page 10558)	Add a parameter to this list at the specified ordinal position.

55.1.53.5.1 TDBXParameterList.AddParameter Method

Add a parameter to this list.

Pascal

```
procedure AddParameter(Parameter: TDBXParameter); virtual;
```

C++

```
virtual __fastcall AddParameter(TDBXParameter Parameter);
```

Description

Add a TDBXParameter (see page 10545) instance to this list.

55.1.53.5.2 TDBXParameterList.ClearParameters Method

Remove all parameters.

Pascal

```
procedure ClearParameters; virtual; overload;
```

C++

```
virtual __fastcall ClearParameters();
```

Description

Remove and call Free for all parameters.

55.1.53.5.3 TDBXParameterList.GetCount Method

Number parameters in list.

Pascal

```
function GetCount: TInt32; virtual;
```

C++

```
virtual __fastcall TInt32 GetCount();
```

Description

GetCount holds the number of parameters in the parameter list.

55.1.53.5.4 TDBXParameterList.GetOrdinal Method

The ordinal index for the named parameter.

Pascal

```
function GetOrdinal(const Name: WideString): Integer; virtual;
```

C++

```
virtual __fastcall int GetOrdinal(const BSTR Name);
```

Description

UseGetOrdinal to get the ordinal position for the parameter specified by name.

55.1.53.5.5 TDBXParameterList.GetParameterByOrdinal Method

Get parameter by ordinal position.

Pascal

```
function GetParameterByOrdinal(const Ordinal: TInt32): TDBXParameter; virtual;
```

C++

```
virtual __fastcall TDBXParameter GetParameterByOrdinal(const TInt32 Ordinal);
```

Description

GetParameterByOrdinal returns the parameter in the specified ordinal position.

See Also

GetOrdinal (see page 10556)

55.1.53.5.6 TDBXParameterList.InsertParameter Method

Inserts parameter into this list at the specified ordinal position.

Pascal

```
procedure InsertParameter(Ordinal: Integer; Parameter: TDBXParameter); virtual;
```

C++

```
virtual __fastcall InsertParameter(int Ordinal, TDBXParameter Parameter);
```

Description

Use InsertParameter to insert a TDBXParameter (see page 10545) instance to this list at the specified ordinal position.

55.1.53.5.7 RemoveParameter Method

This is the overview for the RemoveParameter method overload.

Overload List

	Name	Description
	TDBXParameterList.RemoveParameter (Integer) (see page 10558)	Removes a parameter instance from this list.
	TDBXParameterList.RemoveParameter (TDBXParameter) (see page 10558)	Removes a parameter instance from this list.

55.1.53.5.7.1 TDBXParameterList.RemoveParameter Method (Integer)

Removes a parameter instance from this list.

Pascal

```
procedure RemoveParameter(Ordinal: Integer); virtual; overload;
procedure RemoveParameter(Parameter: TDBXParameter); virtual; overload;
```

C++

```
virtual __fastcall RemoveParameter(int Ordinal);
virtual __fastcall RemoveParameter(TDBXParameter Parameter);
```

Description

Use RemoveParameter to remove a TDBXParameter (see page 10545) instance from this list at either the specified ordinal position. or by specifying the instance of TDBXParameter (see page 10545) to be removed. The Free method will be called for the removed parameter.

55.1.53.5.8 TDBXParameterList.SetCount Method

Sets the length of the list to the specified number of parameters.

Pascal

```
procedure SetCount(Count: TInt32); virtual;
```

C++

```
virtual __fastcall SetCount(TInt32 Count);
```

Description

Set the length of the list to the specifed number of parameters. If this grows the list, new items will be set to **null**. If this shrinks the list, items beyond the new length will be released by calling the ;Free method.

55.1.53.5.9 TDBXParameterList.SetParameter Method

Add a parameter to this list at the specified ordinal position.

Pascal

```
procedure SetParameter(Ordinal: Integer; Parameter: TDBXParameter); virtual;
```

C++

```
virtual __fastcall SetParameter(int Ordinal, TDBXParameter Parameter);
```

Description

Add a TDBXParameter (see page 10545) instance to this list at the specified ordinal position.

55.1.53.6 TDBXParameterList Properties

The properties of the TDBXParameterList class are listed here.

Public Properties

	Name	Description
R	Count (see page 10559)	The number of parameters in this list.
R	Parameter (see page 10559)	Individual; TDBXParameter (see page 10545) instance.

55.1.53.6.1 TDBXParameterList.Count Property

The number of parameters in this list.

Pascal

```
property Count: TInt32;
```

C++

```
__property TInt32 Count;
```

Description

Count gets the number of parameters in this list.

55.1.53.6.2 TDBXParameterList.Parameter Property

Individual; TDBXParameter (see page 10545) instance.

Pascal

```
property Parameter [const Name: WideString]: TDBXParameter;
```

C++

```
__property TDBXParameter Parameter[BSTR const Name];
```

Description

Either gets the TDBXParameter (see page 10545) instance at the Ordinal position in the list or gets the TDBXParameter (see page 10545) instance that has the Name property provided.

55.1.54 TDBXProperties Class

Manages connection and driver level name/value pair property settings.

Class Hierarchy

Pascal

```
TDBXProperties = class;
```

C++

```
class TDBXProperties;
```

File

DBXCommon

Description

The TDBXProperties class manages connection and driver level name/value pair property settings.

55.1.54.1 TDBXProperties Members

The following tables list the members exposed by TDBXProperties.

Public Methods

	Name	Description
	Add (see page 10561)	Add a name/value pair.
	AddProperties (see page 10561)	Add (see page 10561) the name/value pairs from List into this instance of TDBXProperties (see page 10559).
	Clone (see page 10562)	Creates a new instance of TDBXProperties (see page 10559) with a copy of the same name/value pairs
	Create (see page 10560)	This is the overview for the Create constructor overload.
V	Destroy (see page 10561)	Destroys an instance of the TDBXProperties (see page 10559) class.
	GetBoolean (see page 10562)	Get property value as a Boolean.
	GetInteger (see page 10562)	Get property value as an Integer.
	GetLists (see page 10562)	Get the Names and Values (see page 10564) as two separate TWideStringArray (see page 10674) lists.
	GetRequiredValue (see page 10563)	Get property value as a string.
	SetProperties (see page 10563)	Sets the semicolon-separated name/value pairs from ConnectionString into this instance of TDBXProperties (see page 10559).

Public Properties

	Name	Description
R	Properties (see page 10563)	TWideString property name for TDBXProperties (see page 10559) name/value pairs.
	Values (see page 10564)	TWideString property value for TDBXProperties (see page 10559) name/value pairs.

55.1.54.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TDBXProperties.Create () (see page 10560)	Create an instance of the TDBXProperties (see page 10559) class.
	TDBXProperties.Create (TDBXContext) (see page 10560)	Create an instance of the TDBXProperties (see page 10559) class.

55.1.54.2.1 TDBXProperties.Create Constructor ()

Create an instance of the TDBXProperties (see page 10559) class.

Pascal

```
constructor Create(DBXContext: TDBXContext); overload;
constructor Create; overload;
```

C++

```
__fastcall TDBXProperties(TDBXContext DBXContext);
__fastcall TDBXProperties();
```

Description

Use Create to create an instance of the TDBXProperties (see page 10559) class.

55.1.54.3 TDBXProperties.Destroy Destructor

Destroys an instance of the TDBXProperties (see page 10559) class.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXProperties();
```

Description

Use Destroy to destroy an instance of the TDBXProperties (see page 10559) class.

55.1.54.4 TDBXProperties Methods

The methods of the TDBXProperties class are listed here.

Public Methods

	Name	Description
	Add (see page 10561)	Add a name/value pair.
	AddProperties (see page 10561)	Add (see page 10561) the name/value pairs from List into this instance of TDBXProperties (see page 10559).
	Clone (see page 10562)	Creates a new instance of TDBXProperties (see page 10559) with a copy of the same name/value pairs
	GetBoolean (see page 10562)	Get property value as a Boolean.
	GetInteger (see page 10562)	Get property value as an Integer.
	GetLists (see page 10562)	Get the Names and Values (see page 10564) as two separate TWideStringArray (see page 10674) lists.
	GetRequiredValue (see page 10563)	Get property value as a string.
	SetProperties (see page 10563)	Sets the semicolon-separated name/value pairs from ConnectionString into this instance of TDBXProperties (see page 10559).

55.1.54.4.1 TDBXProperties.Add Method

Add a name/value pair.

Pascal

```
procedure Add(const Name: WideString; const Value: WideString);
```

C++

```
__fastcall Add(const BSTR Name, const BSTR Value);
```

Description

Use Add to add a name/value pair for connection and driver properties.

55.1.54.4.2 TDBXProperties.AddProperties Method

Add (see page 10561) the name/value pairs from List into this instance of TDBXProperties (see page 10559).

Pascal

```
procedure AddProperties(List: TWideStrings); overload;
```

C++

```
__fastcall AddProperties(TWideStrings List);
```

Description

Use AddProperties to add the name/value pairs from List into this instance of TDBXProperties (see page 10559).

55.1.54.4.3 TDBXProperties.Clone Method

Creates a new instance of TDBXProperties (see page 10559) with a copy of the same name/value pairs

Pascal

```
function Clone: TDBXProperties;
```

C++

```
__fastcall TDBXProperties Clone();
```

Description

Use Clone to create a new instance of TDBXProperties (see page 10559) with a copy of the same name/value pairs

55.1.54.4.4 TDBXProperties.GetBoolean Method

Get property value as a Boolean.

Pascal

```
function GetBoolean(const Name: WideString): Boolean;
```

C++

```
__fastcall Boolean GetBoolean(const BSTR Name);
```

Description

Use GetBoolean to get a TDBXProperties (see page 10559) property value as a Boolean.

55.1.54.4.5 TDBXProperties.GetInteger Method

Get property value as an Integer.

Pascal

```
function GetInteger(const Name: WideString): Integer;
```

C++

```
__fastcall int GetInteger(const BSTR Name);
```

Description

Use GetInteger to get a TDBXProperties (see page 10559) property value as an Integer.

55.1.54.4.6 TDBXProperties.GetLists Method

Get the Names and Values (see page 10564) as two separate TWideStringArray (see page 10674) lists.

Pascal

```
procedure GetLists(var Names: TWideStringArray; var Values: TWideStringArray);
```

C++

```
__fastcall GetLists(TWideStringArray Names, TWideStringArray Values);
```

Description

Use GetLists to get the Names and Values (see page 10564) of an instance of TDBXProperties (see page 10559) as two separate TWideStringArray (see page 10674) lists.

55.1.54.4.7 TDBXProperties.GetRequiredValue Method

Get property value as a string.

Pascal

```
function GetRequiredValue(const Name: WideString): WideString;
```

C++

```
__fastcall BSTR GetRequiredValue(const BSTR Name);
```

Description

Use GetRequiredValue to get a TDBXPropertiesproperty value as a string. Throws an exception if this TDBXProperties (see page 10559) instance does not contain a property with the given name.

55.1.54.4.8 TDBXProperties.SetProperties Method

Sets the semicolon-separated name/value pairs from ConnectionString into this instance of TDBXProperties (see page 10559).

Pascal

```
procedure SetProperties(const ConnectionString: WideString); overload;
```

C++

```
__fastcall SetProperties(const BSTR ConnectionString);
```

Description

Use SetProperties to set the semicolon-separated name/value pairs from ConnectionString into this instance of TDBXProperties (see page 10559).

55.1.54.5 TDBXProperties Properties

The properties of the TDBXProperties class are listed here.

Public Properties

	Name	Description
R	Properties (see page 10563)	TWideString property name for TDBXProperties (see page 10559) name/value pairs.
	Values (see page 10564)	TWideString property value for TDBXProperties (see page 10559) name/value pairs.

55.1.54.5.1 TDBXProperties.Properties Property

TWideString property name for TDBXProperties (see page 10559) name/value pairs.

Pascal

```
property Properties: TWideStrings;
```

C++

```
__property TWideStrings Properties;
```

Description

TDBXProperties.Properties holds the name for connection and driver properties.

55.1.54.5.2 TDBXProperties.Values Property

TWideString property value for TDBXProperties (see page 10559) name/value pairs.

Pascal

```
property Values [const Name: WideString]: WideString;
```

C++

```
__property BSTR Values[BSTR const Name];
```

Description

TDBXProperties.Values holds the value for connection and driver properties.

55.1.55 TDBXPropertiesItem Class

Contains a set of TDBXProperties (see page 10559).

Class Hierarchy

DBXCommon.TDBXPropertiesItem

Pascal

```
TDBXPropertiesItem = class;
```

C++

```
class TDBXPropertiesItem;
```

File

DBXCommon

Description

This class is used by driver implementors. It contains a set of TDBXProperties (see page 10559).

See Also

TDBXProperties (see page 10559)

55.1.55.1 TDBXPropertiesItem Members

The following tables list the members exposed by TDBXPropertiesItem.

Public Fields

	Name	Description
	FDeletgatePath (see page 10565)	This is FDeletgatePath, a member of class TDBXPropertiesItem.
	FProperties (see page 10566)	This is FProperties, a member of class TDBXPropertiesItem.

Public Methods

	Name	Description
	Create (see page 10565)	Creates an instance of TDBXPropertiesItem (see page 10564)
V	Destroy (see page 10565)	Destroys a TDBXPropertiesItem (see page 10564) instance.

55.1.55.2 TDBXPropertiesItem.Create Constructor

Creates an instance of TDBXPropertiesItem (see page 10564)

Pascal

```
constructor Create(Properties: TDBXProperties);
```

C++

```
__fastcall TDBXPropertiesItem(TDBXProperties Properties);
```

Description

This constructor creates and initializes an instance of TDBXPropertiesItem (see page 10564).

See Also

Destroy (see page 10565)

55.1.55.3 TDBXPropertiesItem.Destroy Destructor

Destroys a TDBXPropertiesItem (see page 10564) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXPropertiesItem();
```

Description

This destructor destroys an instance of TDBXPropertiesItem (see page 10564).

See Also

Create (see page 10565)

55.1.55.4 TDBXPropertiesItem Fields

The fields of the TDBXPropertiesItem class are listed here.

Public Fields

	Name	Description
	FDeletgatePath (see page 10565)	This is FDeletgatePath, a member of class TDBXPropertiesItem.
	FProperties (see page 10566)	This is FProperties, a member of class TDBXPropertiesItem.

55.1.55.4.1 TDBXPropertiesItem.FDeletgatePath Field

Pascal

```
FDeletgatePath: TDBXDelegateItem;
```

C++

```
TDBXDelegateItem FDeletgatePath;
```

Description

This is FDeletgatePath, a member of class TDBXPropertiesItem.

55.1.55.4.2 TDBXPropertiesItem.FProperties Field

Pascal

```
FProperties: TDBXProperties;
```

C++

```
TDBXProperties FProperties;
```

Description

This is FProperties, a member of class TDBXPropertiesItem.

55.1.56 TDBXPropertyNames Class

Property names for connections and drivers.

Class Hierarchy

DBXCommon.TDBXPropertyNames

Pascal

```
TDBXPropertyNames = class;
```

C++

```
class TDBXPropertyNames;
```

File

DBXCommon

Description

Property names are used for both connections and drivers to provide information, such as user names and passwords.

This table contains property names.

Constant	Description
ConnectionName (see page 10569)	Name of connection. If connection properties are read from the dbxconnections.ini file, this is the ini file section name. AdoDbx Client can use this property setting for a simple one property connection string. Connection names can also be used to load a connection from a TDBXConnectionFactory (see page 10422) instance.
InstalledDrivers (see page 10572)	dbxdrivers.ini section that specifies which drivers are enabled.
GetDriverFunc (see page 10571)	Used by DBXDynalink drivers to specify exported entry point for loading a driver prior to dbExpress 4.

VendorLib (see page 10574)	Used by DBXDynalink drivers. Name of a vendor specific client library.
ErrorResourceFile (see page 10571)	Used only by Linux versions of dbExpress to specify a localized error message file.
LibraryName (see page 10572)	Used by DBXDynalink drivers to specify the name of the DBXDynalink DLL.
DriverName (see page 10570)	Unique name for a dbExpress driver. Used by the dbxconnections.ini file to refer to a connections driver in the dbxdrivers.ini file.
Host	Host name of a database server that a driver connects to.
Port (see page 10573)	Port (see page 10573) number of a database server that a driver connects to.
Database (see page 10569)	Name of a database to connect to.
UserName (see page 10574)	Login user name used to connect to a database.
Password (see page 10573)	Login password name used to connect to a database.
Role (see page 10573)	Login role used to connect to a database.
IsolationLevel (see page 10572)	Initial transaction isolation to use when connecting to a database. Transaction isolation can also be specified when a transaction is started.
MaxBlobSize (see page 10572)	Maximum BLOB size. Set to –1 to specify no limit.
DelegateConnection (see page 10569)	Delegate connection used by this connection. Delegate connections implement the dbExpress 4 driver framework. They can perform pre or post processing of all public and protected methods before delegating to the real connection or another delegate connection.
DriverUnit (see page 10571)	Used to provide informative error message when driver cannot be loaded.
DriverPackage (see page 10570)	Used to load a native driver dynamically. **Deprecated**. Preferable to just add the a comma plus the package name to the end of the **DriverPackageLoader (see page 10571)** property setting.
DelegateDriver (see page 10570)	Indicates that this driver is a delegate and can only be used in conjunction with a non-delegate driver.
DriverAssembly (see page 10570)	Used to load a managed driver dynamically. **Deprecated**. Preferable to just add the a comma plus the package name to the end of the **DriverPackageLoader (see page 10571)** property setting.
DriverPackageLoader (see page 10571)	Native Driver loader class used to load a native driver dynamically from a package. Comma separated class and package name.
DriverAssemblyLoader (see page 10570)	Managed Driver loader class used to load a managed driver dynamically from an assembly. Comma separated class and package name.
DelegateDriver (see page 10570)	Indicates that this driver is a delegate and can only be used in conjunction with a non-delegate driver.

SchemaOverride (see page 10573)	Most applications should not need to use this constant. It is provided for backwards compatibility of the dbExpress VCL. VCL components use this setting to scope metadata requests. This property should be set to **user-name.override-schema-name**. For example, tables created by the MSSQL sa user are placed in the dbo schema. In this case, SchemaOverride (see page 10573) can be set to 'sa.dbo'. This causes VCL to specify the 'dbo' schema for metadata requests when a connection is made using the 'sa' user. An SQL `'%'` like pattern can also be used for the user-name and the override-schema-name. For example, `'%.%'` causes the schema to not be specified at all for all users.

See Also

TDBXConnection (see page 10398)

TDBXDriver (see page 10472)

55.1.56.1 TDBXPropertyNames Members

The following tables list the members exposed by TDBXPropertyNames.

Public Constants

	Name	Description
	ConnectionName (see page 10569)	This is ConnectionName, a member of class TDBXPropertyNames.
	Database (see page 10569)	This is Database, a member of class TDBXPropertyNames.
	DelegateConnection (see page 10569)	This is DelegateConnection, a member of class TDBXPropertyNames.
	DelegateDriver (see page 10570)	This is DelegateDriver, a member of class TDBXPropertyNames.
	DriverAssembly (see page 10570)	This is DriverAssembly, a member of class TDBXPropertyNames.
	DriverAssemblyLoader (see page 10570)	This is DriverAssemblyLoader, a member of class TDBXPropertyNames.
	DriverName (see page 10570)	This is DriverName, a member of class TDBXPropertyNames.
	DriverPackage (see page 10570)	This is DriverPackage, a member of class TDBXPropertyNames.
	DriverPackageLoader (see page 10571)	This is DriverPackageLoader, a member of class TDBXPropertyNames.
	DriverUnit (see page 10571)	This is DriverUnit, a member of class TDBXPropertyNames.
	ErrorResourceFile (see page 10571)	This is ErrorResourceFile, a member of class TDBXPropertyNames.
	GetDriverFunc (see page 10571)	This is GetDriverFunc, a member of class TDBXPropertyNames.
	HostName (see page 10572)	This is HostName, a member of class TDBXPropertyNames.
	InstalledDrivers (see page 10572)	This is InstalledDrivers, a member of class TDBXPropertyNames.
	IsolationLevel (see page 10572)	This is IsolationLevel, a member of class TDBXPropertyNames.
	LibraryName (see page 10572)	This is LibraryName, a member of class TDBXPropertyNames.
	MaxBlobSize (see page 10572)	This is MaxBlobSize, a member of class TDBXPropertyNames.
	Password (see page 10573)	This is Password, a member of class TDBXPropertyNames.
	Port (see page 10573)	This is Port, a member of class TDBXPropertyNames.
	Role (see page 10573)	This is Role, a member of class TDBXPropertyNames.
	SchemaOverride (see page 10573)	This is SchemaOverride, a member of class TDBXPropertyNames.
	UserName (see page 10574)	This is UserName, a member of class TDBXPropertyNames.
	VendorLib (see page 10574)	This is VendorLib, a member of class TDBXPropertyNames.

55.1.56.2 TDBXPropertyNames Constants

The constants of the TDBXPropertyNames class are listed here.

Public Constants

	Name	Description
	ConnectionName (see page 10569)	This is ConnectionName, a member of class TDBXPropertyNames.
	Database (see page 10569)	This is Database, a member of class TDBXPropertyNames.
	DelegateConnection (see page 10569)	This is DelegateConnection, a member of class TDBXPropertyNames.

	DelegateDriver (see page 10570)	This is DelegateDriver, a member of class TDBXPropertyNames.
	DriverAssembly (see page 10570)	This is DriverAssembly, a member of class TDBXPropertyNames.
	DriverAssemblyLoader (see page 10570)	This is DriverAssemblyLoader, a member of class TDBXPropertyNames.
	DriverName (see page 10570)	This is DriverName, a member of class TDBXPropertyNames.
	DriverPackage (see page 10570)	This is DriverPackage, a member of class TDBXPropertyNames.
	DriverPackageLoader (see page 10571)	This is DriverPackageLoader, a member of class TDBXPropertyNames.
	DriverUnit (see page 10571)	This is DriverUnit, a member of class TDBXPropertyNames.
	ErrorResourceFile (see page 10571)	This is ErrorResourceFile, a member of class TDBXPropertyNames.
	GetDriverFunc (see page 10571)	This is GetDriverFunc, a member of class TDBXPropertyNames.
	HostName (see page 10572)	This is HostName, a member of class TDBXPropertyNames.
	InstalledDrivers (see page 10572)	This is InstalledDrivers, a member of class TDBXPropertyNames.
	IsolationLevel (see page 10572)	This is IsolationLevel, a member of class TDBXPropertyNames.
	LibraryName (see page 10572)	This is LibraryName, a member of class TDBXPropertyNames.
	MaxBlobSize (see page 10572)	This is MaxBlobSize, a member of class TDBXPropertyNames.
	Password (see page 10573)	This is Password, a member of class TDBXPropertyNames.
	Port (see page 10573)	This is Port, a member of class TDBXPropertyNames.
	Role (see page 10573)	This is Role, a member of class TDBXPropertyNames.
	SchemaOverride (see page 10573)	This is SchemaOverride, a member of class TDBXPropertyNames.
	UserName (see page 10574)	This is UserName, a member of class TDBXPropertyNames.
	VendorLib (see page 10574)	This is VendorLib, a member of class TDBXPropertyNames.

55.1.56.2.1 TDBXPropertyNames.ConnectionName Constant

Pascal

```
const ConnectionName = 'ConnectionName';
```

C++

```
ConnectionName = 'ConnectionName';
```

Description

This is ConnectionName, a member of class TDBXPropertyNames.

55.1.56.2.2 TDBXPropertyNames.Database Constant

Pascal

```
const Database = 'Database';
```

C++

```
Database = 'Database';
```

Description

This is Database, a member of class TDBXPropertyNames.

55.1.56.2.3 TDBXPropertyNames.DelegateConnection Constant

Pascal

```
const DelegateConnection = 'DelegateConnection';
```

C++

```
DelegateConnection = 'DelegateConnection';
```

Description

This is DelegateConnection, a member of class TDBXPropertyNames.

55.1.56.2.4 TDBXPropertyNames.DelegateDriver Constant

Pascal

```
const DelegateDriver = 'DelegateDriver';
```

C++

```
DelegateDriver = 'DelegateDriver';
```

Description

This is DelegateDriver, a member of class TDBXPropertyNames.

55.1.56.2.5 TDBXPropertyNames.DriverAssembly Constant

Pascal

```
const DriverAssembly = 'DriverAssembly';
```

C++

```
DriverAssembly = 'DriverAssembly';
```

Description

This is DriverAssembly, a member of class TDBXPropertyNames.

55.1.56.2.6 TDBXPropertyNames.DriverAssemblyLoader Constant

Pascal

```
const DriverAssemblyLoader = 'DriverAssemblyLoader';
```

C++

```
DriverAssemblyLoader = 'DriverAssemblyLoader';
```

Description

This is DriverAssemblyLoader, a member of class TDBXPropertyNames.

55.1.56.2.7 TDBXPropertyNames.DriverName Constant

Pascal

```
const DriverName = 'DriverName';
```

C++

```
DriverName = 'DriverName';
```

Description

This is DriverName, a member of class TDBXPropertyNames.

55.1.56.2.8 TDBXPropertyNames.DriverPackage Constant

Pascal

```
const DriverPackage = 'DriverPackage';
```

C++

```
DriverPackage = 'DriverPackage';
```

Description

This is DriverPackage, a member of class TDBXPropertyNames.

55.1.56.2.9 TDBXPropertyNames.DriverPackageLoader Constant

Pascal

```
const DriverPackageLoader = 'DriverPackageLoader';
```

C++

```
DriverPackageLoader = 'DriverPackageLoader';
```

Description

This is DriverPackageLoader, a member of class TDBXPropertyNames.

55.1.56.2.10 TDBXPropertyNames.DriverUnit Constant

Pascal

```
const DriverUnit = 'DriverUnit';
```

C++

```
DriverUnit = 'DriverUnit';
```

Description

This is DriverUnit, a member of class TDBXPropertyNames.

55.1.56.2.11 TDBXPropertyNames.ErrorResourceFile Constant

Pascal

```
const ErrorResourceFile = 'ErrorResourceFile';
```

C++

```
ErrorResourceFile = 'ErrorResourceFile';
```

Description

This is ErrorResourceFile, a member of class TDBXPropertyNames.

55.1.56.2.12 TDBXPropertyNames.GetDriverFunc Constant

Pascal

```
const GetDriverFunc = 'GetDriverFunc';
```

C++

```
GetDriverFunc = 'GetDriverFunc';
```

Description

This is GetDriverFunc, a member of class TDBXPropertyNames.

55.1.56.2.13 TDBXPropertyNames.HostName Constant

Pascal

```
const HostName = 'HostName';
```

C++

```
HostName = 'HostName';
```

Description

This is HostName, a member of class TDBXPropertyNames.

55.1.56.2.14 TDBXPropertyNames.InstalledDrivers Constant

Pascal

```
const InstalledDrivers = 'Installed Drivers';
```

C++

```
InstalledDrivers = 'Installed Drivers';
```

Description

This is InstalledDrivers, a member of class TDBXPropertyNames.

55.1.56.2.15 TDBXPropertyNames.IsolationLevel Constant

Pascal

```
const IsolationLevel = 'IsolationLevel';
```

C++

```
IsolationLevel = 'IsolationLevel';
```

Description

This is IsolationLevel, a member of class TDBXPropertyNames.

55.1.56.2.16 TDBXPropertyNames.LibraryName Constant

Pascal

```
const LibraryName = 'LibraryName';
```

C++

```
LibraryName = 'LibraryName';
```

Description

This is LibraryName, a member of class TDBXPropertyNames.

55.1.56.2.17 TDBXPropertyNames.MaxBlobSize Constant

Pascal

```
const MaxBlobSize = 'MaxBlobSize';
```

C++

```
MaxBlobSize = 'MaxBlobSize';
```

Description

This is MaxBlobSize, a member of class TDBXPropertyNames.

55.1.56.2.18 TDBXPropertyNames.Password Constant

Pascal

```
const Password = 'Password';
```

C++

```
Password = 'Password';
```

Description

This is Password, a member of class TDBXPropertyNames.

55.1.56.2.19 TDBXPropertyNames.Port Constant

Pascal

```
const Port = 'Port';
```

C++

```
Port = 'Port';
```

Description

This is Port, a member of class TDBXPropertyNames.

55.1.56.2.20 TDBXPropertyNames.Role Constant

Pascal

```
const Role = 'Role';
```

C++

```
Role = 'Role';
```

Description

This is Role, a member of class TDBXPropertyNames.

55.1.56.2.21 TDBXPropertyNames.SchemaOverride Constant

Pascal

```
const SchemaOverride = 'SchemaOverride';
```

C++

```
SchemaOverride = 'SchemaOverride';
```

Description

This is SchemaOverride, a member of class TDBXPropertyNames.

55.1.56.2.22 TDBXPropertyNames.UserName Constant

Pascal

```
const UserName = 'User_Name';
```

C++

```
UserName = 'User_Name';
```

Description

This is UserName, a member of class TDBXPropertyNames.

55.1.56.2.23 TDBXPropertyNames.VendorLib Constant

Pascal

```
const VendorLib = 'VendorLib';
```

C++

```
VendorLib = 'VendorLib';
```

Description

This is VendorLib, a member of class TDBXPropertyNames.

55.1.57 TDBXPropertyNamesEx Class

Temporary extension of TDBXPropertyNames (see page 10566) class.

Class Hierarchy

DBXCommon.TDBXPropertyNamesEx

Pascal

```
TDBXPropertyNamesEx = class;
```

C++

```
class TDBXPropertyNamesEx;
```

File

DBXCommon

Description

TDBXPropertyNamesEx contains common property names for both connections and drivers.

Note: Classes (see page 2685) ending in “Ex” are temporary classes that exist to avoid breaking interfaces. Their contents will be migrated to their base class in the next release.

This table contains property names.

Constant	Description
MetaDataPackageLoader (see page 10576)	Native MetaData loader class used to load a native driver dynamically from a package.

MetaDataAssemblyLoader (see page 10575)	Managed MetaData loader class used to load a managed driver dynamically from an assembly.
ProductVersion (see page 10576)	Can be specified as a driver property as an override of the TDBXConnectionEx.ProductVersion (see page 10576) property. This may be necessary for drivers that do not have the ability to report the product version. The ProductVersion (see page 10576) is a number in the format nn.nn.nnnn.
ProductName (see page 10576)	Provides the product name for the dbExpress metadata.

See Also

TDBXConnectionEx (see page 10417)

TDBXPropertyNames (see page 10566)

ProductVersion (see page 10421)

55.1.57.1 TDBXPropertyNamesEx Members

The following tables list the members exposed by TDBXPropertyNamesEx.

Public Constants

	Name	Description
	MetaDataAssemblyLoader (see page 10575)	This is MetaDataAssemblyLoader, a member of class TDBXPropertyNamesEx.
	MetaDataPackageLoader (see page 10576)	This is MetaDataPackageLoader, a member of class TDBXPropertyNamesEx.
	ProductName (see page 10576)	This is ProductName, a member of class TDBXPropertyNamesEx.
	ProductVersion (see page 10576)	This is ProductVersion, a member of class TDBXPropertyNamesEx.

55.1.57.2 TDBXPropertyNamesEx Constants

The constants of the TDBXPropertyNamesEx class are listed here.

Public Constants

	Name	Description
	MetaDataAssemblyLoader (see page 10575)	This is MetaDataAssemblyLoader, a member of class TDBXPropertyNamesEx.
	MetaDataPackageLoader (see page 10576)	This is MetaDataPackageLoader, a member of class TDBXPropertyNamesEx.
	ProductName (see page 10576)	This is ProductName, a member of class TDBXPropertyNamesEx.
	ProductVersion (see page 10576)	This is ProductVersion, a member of class TDBXPropertyNamesEx.

55.1.57.2.1 TDBXPropertyNamesEx.MetaDataAssemblyLoader Constant

Pascal

```
const MetaDataAssemblyLoader = 'MetaDataAssemblyLoader';
```

C++

```
MetaDataAssemblyLoader = 'MetaDataAssemblyLoader';
```

Description

This is MetaDataAssemblyLoader, a member of class TDBXPropertyNamesEx.

55.1.57.2.2 TDBXPropertyNamesEx.MetaDataPackageLoader Constant

Pascal

```
const MetaDataPackageLoader = 'MetaDataPackageLoader';
```

C++

```
MetaDataPackageLoader = 'MetaDataPackageLoader';
```

Description

This is MetaDataPackageLoader, a member of class TDBXPropertyNamesEx.

55.1.57.2.3 TDBXPropertyNamesEx.ProductName Constant

Pascal

```
const ProductName = 'ProductName';
```

C++

```
ProductName = 'ProductName';
```

Description

This is ProductName, a member of class TDBXPropertyNamesEx.

55.1.57.2.4 TDBXPropertyNamesEx.ProductVersion Constant

Pascal

```
const ProductVersion = 'ProductVersion';
```

C++

```
ProductVersion = 'ProductVersion';
```

Description

This is ProductVersion, a member of class TDBXPropertyNamesEx.

55.1.58 TDBXReader Class

TDBXReader provides a unidirectional reader for a collection of database rows

Class Hierarchy

Pascal

```
TDBXReader = class;
```

C++

```
class TDBXReader;
```

File

DBXCommon

Description

TDBXReader provides a forward only reader for a collection of database rows.

TDBXReader is returned by TDBXCommand.ExecuteQuery. Call TDBXReader.Next (see page 10582) to access the first and successive rows in the collection. Row values can be accessed using the Value (see page 10584) array property. Value (see page 10584) is overloaded so it can be indexed either by ordinal position or by column name.

Note: When an application is no longer needs a TDBXReader instance, it should call TDBXReader.Free. This ensures that all resources associated with the TDBXReader are released.

See Also

Connecting to a Database using the dbExpress Driver Framework

TDBXCommand (see page 10378)

Value (see page 10584)

ExecuteQuery (see page 10384)

Next (see page 10582)

55.1.58.1 TDBXReader Members

The following tables list the members exposed by TDBXReader.

Protected Methods

	Name	Description
	Create (see page 10578)	Creates an instance of TDBXReader (see page 10576)
A	DerivedClose (see page 10580)	This is DerivedClose, a member of class TDBXReader.
A	DerivedNext (see page 10580)	This is DerivedNext, a member of class TDBXReader.
V	GetByteReader (see page 10580)	This is GetByteReader, a member of class TDBXReader.
V	GetColumnCount (see page 10580)	Returns number of columns accessible from the Value (see page 10584) property.
V	GetValue (see page 10582)	This is GetValue, a member of class TDBXReader.
V	GetValueByName (see page 10582)	This is GetValueByName, a member of class TDBXReader.
V	GetValueType (see page 10582)	This is GetValueType, a member of class TDBXReader.
V	SetValues (see page 10583)	This is SetValues, a member of class TDBXReader.

Public Events

	Name	Description
R	OnErrorEvent (see page 10585)	Error handling event.

Public Methods

	Name	Description
V	Destroy (see page 10578)	Destroys an instance of TDBXReader (see page 10576).
V	GetObjectTypeName (see page 10581)	Object type name for ADT and Array Types (see page 23293).
V	GetOrdinal (see page 10581)	Returns ordinal position in Value (see page 10584) property for **Name**.
V	Next (see page 10582)	Navigate to first and successive rows.

Public Properties

	Name	Description
R	ByteReader (see page 10583)	Read values from a byte array.
R	Closed (see page 10583)	This is Closed, a member of class TDBXReader.
R	ColumnCount (see page 10584)	Number of columns that can be accessed from the Value (see page 10584) array.

R	Value (see page 10584)	Instance of TDBXValueat **Ordinal** position.
R	ValueType (see page 10584)	Instance of TDBXValueType (see page 10636) at **Ordinal** position.

Strict protected Fields

	Name	Description
	FDBXContext (see page 10579)	This is FDBXContext, a member of class TDBXReader.
	FDbxRow (see page 10579)	This is FDbxRow, a member of class TDBXReader.

Strict protected Methods

	Name	Description
	FailIfClosed (see page 10580)	This is FailIfClosed, a member of class TDBXReader.
	GetErrorEvent (see page 10581)	This is GetErrorEvent, a member of class TDBXReader.

55.1.58.2 TDBXReader.Create Constructor

Creates an instance of TDBXReader (see page 10576)

Pascal

```
constructor Create(DBXContext: TDBXContext; DbxRow: TDBXRow; ByteReader: TDBXByteReader);
```

C++

```
__fastcall TDBXReader(TDBXContext DBXContext, TDBXRow DbxRow, TDBXByteReader ByteReader);
```

Description

This constructor creates and initializes an instance of TDBXReader (see page 10576).

See Also

Destroy (see page 10578)

55.1.58.3 TDBXReader.Destroy Destructor

Destroys an instance of TDBXReader (see page 10576).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXReader();
```

Description

This destructor destroys an instance of TDBXReader (see page 10576).

55.1.58.4 TDBXReader Fields

The fields of the TDBXReader class are listed here.

Strict protected Fields

	Name	Description
	FDBXContext (see page 10579)	This is FDBXContext, a member of class TDBXReader.
	FDbxRow (see page 10579)	This is FDbxRow, a member of class TDBXReader.

55.1.58.4.1 TDBXReader.FDBXContext Field

Pascal

```
FDBXContext: TDBXContext;
```

C++

```
TDBXContext FDBXContext;
```

Description

This is FDBXContext, a member of class TDBXReader.

55.1.58.4.2 TDBXReader.FDbxRow Field

Pascal

```
FDbxRow: TDBXRow;
```

C++

```
TDBXRow FDbxRow;
```

Description

This is FDbxRow, a member of class TDBXReader.

55.1.58.5 TDBXReader Methods

The methods of the TDBXReader class are listed here.

Protected Methods

	Name	Description
	DerivedClose (see page 10580)	This is DerivedClose, a member of class TDBXReader.
	DerivedNext (see page 10580)	This is DerivedNext, a member of class TDBXReader.
	GetByteReader (see page 10580)	This is GetByteReader, a member of class TDBXReader.
	GetColumnCount (see page 10580)	Returns number of columns accessible from the Value (see page 10584) property.
	GetValue (see page 10582)	This is GetValue, a member of class TDBXReader.
	GetValueByName (see page 10582)	This is GetValueByName, a member of class TDBXReader.
	GetValueType (see page 10582)	This is GetValueType, a member of class TDBXReader.
	SetValues (see page 10583)	This is SetValues, a member of class TDBXReader.

Public Methods

	Name	Description
	GetObjectTypeName (see page 10581)	Object type name for ADT and Array Types (see page 23293).
	GetOrdinal (see page 10581)	Returns ordinal position in Value (see page 10584) property for **Name**.
	Next (see page 10582)	Navigate to first and successive rows.

Strict protected Methods

	Name	Description
	FailIfClosed (see page 10580)	This is FailIfClosed, a member of class TDBXReader.
	GetErrorEvent (see page 10581)	This is GetErrorEvent, a member of class TDBXReader.

55.1.58.5.1 TDBXReader.DerivedClose Method

Pascal

```
procedure DerivedClose; virtual; abstract;
```

C++

```
virtual __fastcall DerivedClose() = 0;
```

Description

This is DerivedClose, a member of class TDBXReader.

55.1.58.5.2 TDBXReader.DerivedNext Method

Pascal

```
function DerivedNext: Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean DerivedNext() = 0;
```

Description

This is DerivedNext, a member of class TDBXReader.

55.1.58.5.3 TDBXReader.FailIfClosed Method

Pascal

```
procedure FailIfClosed;
```

C++

```
__fastcall FailIfClosed();
```

Description

This is FailIfClosed, a member of class TDBXReader.

55.1.58.5.4 TDBXReader.GetByteReader Method

Pascal

```
function GetByteReader: TDBXByteReader; virtual;
```

C++

```
virtual __fastcall TDBXByteReader GetByteReader();
```

Description

This is GetByteReader, a member of class TDBXReader.

55.1.58.5.5 TDBXReader.GetColumnCount Method

Returns number of columns accessible from the Value (see page 10584) property.

Pascal

```
function GetColumnCount: TInt32; virtual;
```

C++

```
virtual __fastcall TInt32 GetColumnCount();
```

Description

This method returns the number of columns accessible from the Value (see page 10584) array property.

See Also

Connecting to a Database using the dbExpress Driver Framework

Value (see page 10584)

55.1.58.5.6 TDBXReader.GetErrorEvent Method

Pascal

```
function GetErrorEvent: TDBXErrorEvent;
```

C++

```
__fastcall TDBXErrorEvent GetErrorEvent();
```

Description

This is GetErrorEvent, a member of class TDBXReader.

55.1.58.5.7 TDBXReader.GetObjectTypeName Method

Object type name for ADT and Array Types (see page 23293).

Pascal

```
function GetObjectTypeName(Ordinal: TInt32): WideString; virtual;
```

C++

```
virtual __fastcall BSTR GetObjectTypeName(TInt32 Ordinal);
```

Description

GetObjectTypeName returns the object type name for ADT and Array Types (see page 23293). The Oracle driver supports these data types.

55.1.58.5.8 TDBXReader.GetOrdinal Method

Returns ordinal position in Value (see page 10584) property for **Name**.

Pascal

```
function GetOrdinal(const Name: WideString): TInt32; virtual; overload;
```

C++

```
virtual __fastcall TInt32 GetOrdinal(const BSTR Name);
```

Description

This method returns the ordinal position in the Value (see page 10584) array property of the column with the given **Name**.

See Also

Value (see page 10584)

55.1.58.5.9 TDBXReader.GetValue Method

Pascal

```
function GetValue(const Ordinal: TInt32): TDBXValue; virtual;
```

C++

```
virtual __fastcall TDBXValue GetValue(const TInt32 Ordinal);
```

Description

This is GetValue, a member of class TDBXReader.

55.1.58.5.10 TDBXReader.GetValueByName Method

Pascal

```
function GetValueByName(const Name: WideString): TDBXValue; virtual;
```

C++

```
virtual __fastcall TDBXValue GetValueByName(const BSTR Name);
```

Description

This is GetValueByName, a member of class TDBXReader.

55.1.58.5.11 TDBXReader.GetValueType Method

Pascal

```
function GetValueType(const Ordinal: TInt32): TDBXValueType; virtual;
```

C++

```
virtual __fastcall TDBXValueType GetValueType(const TInt32 Ordinal);
```

Description

This is GetValueType, a member of class TDBXReader.

55.1.58.5.12 TDBXReader.Next Method

Navigate to first and successive rows.

Pascal

```
function Next: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean Next();
```

Description

This method navigates to the first and successive rows of the database. It returns false when there are no more rows.

See Also

Connecting to a Database using the dbExpress Driver Framework

55.1.58.5.13 TDBXReader.SetValues Method

Pascal

```
procedure SetValues(Values: TDBXValueArray); virtual;
```

C++

```
virtual __fastcall SetValues(TDBXValueArray Values);
```

Description

This is SetValues, a member of class TDBXReader.

55.1.58.6 TDBXReader Properties

The properties of the TDBXReader class are listed here.

Public Properties

	Name	Description
R	ByteReader (see page 10583)	Read values from a byte array.
R	Closed (see page 10583)	This is Closed, a member of class TDBXReader.
R	ColumnCount (see page 10584)	Number of columns that can be accessed from the Value (see page 10584) array.
R	Value (see page 10584)	Instance of TDBXValueat **Ordinal** position.
R	ValueType (see page 10584)	Instance of TDBXValueType (see page 10636) at **Ordinal** position.

55.1.58.6.1 TDBXReader.ByteReader Property

Read values from a byte array.

Pascal

```
property ByteReader: TDBXByteReader;
```

C++

```
__property TDBXByteReader ByteReader;
```

Description

This property is used by TDataSet to read values from a byte array instead of distinct scalar values. It contains a TDBXByteReader (see page 10368) instance. This is a convenience property for TDataSet.

See Also

TDataSet (see page 6894)

TDBXByteReader (see page 10368)

55.1.58.6.2 TDBXReader.Closed Property

Pascal

```
property Closed: Boolean;
```

C++

```
__property Boolean Closed;
```

Description

This is Closed, a member of class TDBXReader.

55.1.58.6.3 TDBXReader.ColumnCount Property

Number of columns that can be accessed from the Value (see page 10584) array.

Pascal

```
property ColumnCount: TInt32;
```

C++

```
__property TInt32 ColumnCount;
```

Description

ColumnCount holds the number of columns that can be accessed from the Value (see page 10584) array property.

See Also

Value (see page 10584)

55.1.58.6.4 TDBXReader.Value Property

Instance of TDBXValueat **Ordinal** position.

Pascal

```
property Value [const Name: WideString]: TDBXValue;
```

C++

```
__property TDBXValue Value[BSTR const Name];
```

Description

This property holds an instance of TDBXValue (see page 10626) for the **Ordinal** position in the current row.

See Also

TDBXValue (see page 10626)

55.1.58.6.5 TDBXReader.ValueType Property

Instance of TDBXValueType (see page 10636) at **Ordinal** position.

Pascal

```
property ValueType [const Ordinal: TInt32]: TDBXValueType;
```

C++

```
__property TDBXValueType ValueType[TInt32 const Ordinal];
```

Description

This property contains an instance of TDBXValueType (see page 10636) for the **Ordinal** position in the current row.

See Also

TDBXValueType (see page 10636)

55.1.58.7 TDBXReader Events

The events of the TDBXReader class are listed here.

Public Events

	Name	Description
	OnErrorEvent (see page 10585)	Error handling event.

55.1.58.7.1 TDBXReader.OnErrorEvent Event

Error handling event.

Pascal

```
property OnErrorEvent: TDBXErrorEvent;
```

C++

```
__property TDBXErrorEvent OnErrorEvent;
```

Description

This property contains a TDBXErrorEvent (see page 10671) error handling event for this TDBXReader (see page 10576).

See Also

TDBXError (see page 10486)

TDBXErrorCodes (see page 10489)

55.1.59 TDBXReaderByteReader Class

Implementation of TDBXByteReader (see page 10368).

Class Hierarchy

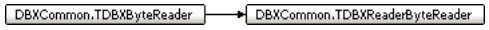

Pascal

```
TDBXReaderByteReader = class(TDBXByteReader);
```

C++

```
class TDBXReaderByteReader : public TDBXByteReader;
```

File

DBXCommon

Description

This class Is used by driver implementors who do not want to provide an optimized TDBXByteReader (see page 10368) implementation. It is an implementation of TDBXByteReader (see page 10368) on top of a TDBXReader (see page 10576) implementation.

See Also

TDBXByteReader (see page 10368)

TDBXReader (see page 10576)

55.1.59.1 TDBXReaderByteReader Members

The following tables list the members exposed by TDBXReaderByteReader.

Public Methods

	Name	Description
	Create (see page 10586)	Creates an instance of TDBXReaderByteReader (see page 10585)
	GetAnsiString (see page 10587)	This is GetAnsiString, a member of class TDBXReaderByteReader.
	GetBcd (see page 10587)	This is GetBcd, a member of class TDBXReaderByteReader.
	GetByteLength (see page 10587)	This is GetByteLength, a member of class TDBXReaderByteReader.
	GetBytes (see page 10587)	This is GetBytes, a member of class TDBXReaderByteReader.
	GetDate (see page 10588)	This is GetDate, a member of class TDBXReaderByteReader.
	GetDouble (see page 10588)	This is GetDouble, a member of class TDBXReaderByteReader.
	GetInt16 (see page 10588)	This is GetInt16, a member of class TDBXReaderByteReader.
	GetInt32 (see page 10588)	This is GetInt32, a member of class TDBXReaderByteReader.
	GetInt64 (see page 10589)	This is GetInt64, a member of class TDBXReaderByteReader.
	GetTime (see page 10589)	This is GetTime, a member of class TDBXReaderByteReader.
	GetTimeStamp (see page 10589)	This is GetTimeStamp, a member of class TDBXReaderByteReader.
	GetWideString (see page 10589)	This is GetWideString, a member of class TDBXReaderByteReader.

Strict protected Fields

	Name	Description
	FDBXContext (see page 10369)	This is FDBXContext, a member of class TDBXByteReader.

55.1.59.2 TDBXReaderByteReader.Create Constructor

Creates an instance of TDBXReaderByteReader (see page 10585)

Pascal

```
constructor Create(DBXContext: TDBXContext; DbxReader: TDBXReader);
```

C++

```
__fastcall TDBXReaderByteReader(TDBXContext DBXContext, TDBXReader DbxReader);
```

Description

This constructor creates and initializes an instance of TDBXReaderByteReader (see page 10585).

55.1.59.3 TDBXReaderByteReader Methods

The methods of the TDBXReaderByteReader class are listed here.

Public Methods

	Name	Description
	GetAnsiString (see page 10587)	This is GetAnsiString, a member of class TDBXReaderByteReader.
	GetBcd (see page 10587)	This is GetBcd, a member of class TDBXReaderByteReader.
	GetByteLength (see page 10587)	This is GetByteLength, a member of class TDBXReaderByteReader.
	GetBytes (see page 10587)	This is GetBytes, a member of class TDBXReaderByteReader.
	GetDate (see page 10588)	This is GetDate, a member of class TDBXReaderByteReader.
	GetDouble (see page 10588)	This is GetDouble, a member of class TDBXReaderByteReader.
	GetInt16 (see page 10588)	This is GetInt16, a member of class TDBXReaderByteReader.
	GetInt32 (see page 10588)	This is GetInt32, a member of class TDBXReaderByteReader.

	GetInt64 (see page 10589)	This is GetInt64, a member of class TDBXReaderByteReader.
	GetTime (see page 10589)	This is GetTime, a member of class TDBXReaderByteReader.
	GetTimeStamp (see page 10589)	This is GetTimeStamp, a member of class TDBXReaderByteReader.
	GetWideString (see page 10589)	This is GetWideString, a member of class TDBXReaderByteReader.

55.1.59.3.1 TDBXReaderByteReader.GetAnsiString Method

Pascal

```
procedure GetAnsiString(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull:
LongBool); override;
```

C++

```
virtual __fastcall GetAnsiString(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool
IsNull);
```

Description

This is GetAnsiString, a member of class TDBXReaderByteReader.

55.1.59.3.2 TDBXReaderByteReader.GetBcd Method

Pascal

```
procedure GetBcd(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull: LongBool);
override;
```

C++

```
virtual __fastcall GetBcd(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool IsNull);
```

Description

This is GetBcd, a member of class TDBXReaderByteReader.

55.1.59.3.3 TDBXReaderByteReader.GetByteLength Method

Pascal

```
procedure GetByteLength(Ordinal: TInt32; var Length: Int64; var IsNull: LongBool); override;
```

C++

```
virtual __fastcall GetByteLength(TInt32 Ordinal, Int64 Length, LongBool IsNull);
```

Description

This is GetByteLength, a member of class TDBXReaderByteReader.

55.1.59.3.4 TDBXReaderByteReader.GetBytes Method

Pascal

```
function GetBytes(Ordinal: TInt32; Offset: Int64; const Value: TBytes; ValueOffset: Int64;
Length: Int64; var IsNull: LongBool): Int64; override;
```

C++

```
virtual __fastcall Int64 GetBytes(TInt32 Ordinal, Int64 Offset, const TBytes Value, Int64
ValueOffset, Int64 Length, LongBool IsNull);
```

Description

This is GetBytes, a member of class TDBXReaderByteReader.

55.1.59.3.5 TDBXReaderByteReader.GetDate Method

Pascal

```
procedure GetDate(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull: LongBool);
override;
```

C++

```
virtual __fastcall GetDate(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool IsNull);
```

Description

This is GetDate, a member of class TDBXReaderByteReader.

55.1.59.3.6 TDBXReaderByteReader.GetDouble Method

Pascal

```
procedure GetDouble(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull:
LongBool); override;
```

C++

```
virtual __fastcall GetDouble(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool
IsNull);
```

Description

This is GetDouble, a member of class TDBXReaderByteReader.

55.1.59.3.7 TDBXReaderByteReader.GetInt16 Method

Pascal

```
procedure GetInt16(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull:
LongBool); override;
```

C++

```
virtual __fastcall GetInt16(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool
IsNull);
```

Description

This is GetInt16, a member of class TDBXReaderByteReader.

55.1.59.3.8 TDBXReaderByteReader.GetInt32 Method

Pascal

```
procedure GetInt32(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull:
LongBool); override;
```

C++

```
virtual __fastcall GetInt32(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool
IsNull);
```

Description

This is GetInt32, a member of class TDBXReaderByteReader.

55.1.59.3.9 TDBXReaderByteReader.GetInt64 Method

Pascal

```
procedure GetInt64(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull:
LongBool); override;
```

C++

```
virtual __fastcall GetInt64(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool
IsNull);
```

Description

This is GetInt64, a member of class TDBXReaderByteReader.

55.1.59.3.10 TDBXReaderByteReader.GetTime Method

Pascal

```
procedure GetTime(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull: LongBool);
override;
```

C++

```
virtual __fastcall GetTime(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool IsNull);
```

Description

This is GetTime, a member of class TDBXReaderByteReader.

55.1.59.3.11 TDBXReaderByteReader.GetTimeStamp Method

Pascal

```
procedure GetTimeStamp(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull:
LongBool); override;
```

C++

```
virtual __fastcall GetTimeStamp(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool
IsNull);
```

Description

This is GetTimeStamp, a member of class TDBXReaderByteReader.

55.1.59.3.12 TDBXReaderByteReader.GetWideString Method

Pascal

```
procedure GetWideString(Ordinal: TInt32; const Value: TBytes; Offset: TInt32; var IsNull:
LongBool); override;
```

C++

```
virtual __fastcall GetWideString(TInt32 Ordinal, const TBytes Value, TInt32 Offset, LongBool
IsNull);
```

Description

This is GetWideString, a member of class TDBXReaderByteReader.

55.1.60 TDBXRow Class

Represents a row in a database.

Class Hierarchy

DBXCommon.TDBXRow

Pascal

```
TDBXRow = class;
```

C++

```
class TDBXRow;
```

File

DBXCommon

Description

This class represents a row in a database. It provides a set of methods for manipulating database record data. It is extended by driver implementations. It is not used directly by applications.

55.1.60.1 TDBXRow Members

The following tables list the members exposed by TDBXRow.

Protected Fields

	Name	Description
	FGeneration (see page 10592)	This is FGeneration, a member of class TDBXRow.

Protected Methods

	Name	Description
	Create (see page 10591)	Creates an instance of TDBXRow (see page 10590)
V	GetAnsiString (see page 10593)	This is GetAnsiString, a member of class TDBXRow.
V	GetBcd (see page 10593)	This is GetBcd, a member of class TDBXRow.
V	GetBoolean (see page 10593)	This is GetBoolean, a member of class TDBXRow.
V	GetByteLength (see page 10593)	This is GetByteLength, a member of class TDBXRow.
V	GetBytes (see page 10594)	This is GetBytes, a member of class TDBXRow.
V	GetDate (see page 10594)	This is GetDate, a member of class TDBXRow.
V	GetDouble (see page 10594)	This is GetDouble, a member of class TDBXRow.
V	GetInt16 (see page 10594)	This is GetInt16, a member of class TDBXRow.
V	GetInt32 (see page 10595)	This is GetInt32, a member of class TDBXRow.
V	GetInt64 (see page 10595)	This is GetInt64, a member of class TDBXRow.
V	GetObjectTypeName (see page 10595)	This is GetObjectTypeName, a member of class TDBXRow.
V	GetTime (see page 10595)	This is GetTime, a member of class TDBXRow.
V	GetTimeStamp (see page 10595)	This is GetTimeStamp, a member of class TDBXRow.
V	GetWideString (see page 10596)	This is GetWideString, a member of class TDBXRow.
V	SetBCD (see page 10596)	This is SetBCD, a member of class TDBXRow.
V	SetBoolean (see page 10596)	This is SetBoolean, a member of class TDBXRow.
V	SetDate (see page 10597)	This is SetDate, a member of class TDBXRow.
V	SetDouble (see page 10597)	This is SetDouble, a member of class TDBXRow.
V	SetDynamicBytes (see page 10597)	This is SetDynamicBytes, a member of class TDBXRow.

	SetInt16 (see page 10597)	This is SetInt16, a member of class TDBXRow.
	SetInt32 (see page 10597)	This is SetInt32, a member of class TDBXRow.
	SetInt64 (see page 10598)	This is SetInt64, a member of class TDBXRow.
	SetNull (see page 10598)	This is SetNull, a member of class TDBXRow.
	SetString (see page 10598)	This is SetString, a member of class TDBXRow.
	SetTime (see page 10598)	This is SetTime, a member of class TDBXRow.
	SetTimestamp (see page 10599)	This is SetTimestamp, a member of class TDBXRow.
	SetValueType (see page 10599)	This is SetValueType, a member of class TDBXRow.
	SetWideString (see page 10599)	This is SetWideString, a member of class TDBXRow.

Protected Properties

	Name	Description
	Generation (see page 10599)	This is Generation, a member of class TDBXRow.

Strict protected Fields

	Name	Description
	FDBXContext (see page 10591)	This is FDBXContext, a member of class TDBXRow.

Strict protected Methods

	Name	Description
	NotImplemented (see page 10596)	This is NotImplemented, a member of class TDBXRow.

55.1.60.2 TDBXRow.Create Constructor

Creates an instance of TDBXRow (see page 10590)

Pascal

```
constructor Create(DBXContext: TDBXContext);
```

C++

```
__fastcall TDBXRow(TDBXContext DBXContext);
```

Description

This constructor creates and initializes an instance of TDBXRow (see page 10590).

55.1.60.3 TDBXRow Fields

The fields of the TDBXRow class are listed here.

Protected Fields

	Name	Description
	FGeneration (see page 10592)	This is FGeneration, a member of class TDBXRow.

Strict protected Fields

	Name	Description
	FDBXContext (see page 10591)	This is FDBXContext, a member of class TDBXRow.

55.1.60.3.1 TDBXRow.FDBXContext Field

Pascal

```
FDBXContext: TDBXContext;
```

C++

```
TDBXContext FDBXContext;
```

Description

This is FDBXContext, a member of class TDBXRow.

55.1.60.3.2 TDBXRow.FGeneration Field

Pascal

```
FGeneration: TInt32;
```

C++

```
TInt32 FGeneration;
```

Description

This is FGeneration, a member of class TDBXRow.

55.1.60.4 TDBXRow Methods

The methods of the TDBXRow class are listed here.

Protected Methods

	Name	Description
	GetAnsiString (see page 10593)	This is GetAnsiString, a member of class TDBXRow.
	GetBcd (see page 10593)	This is GetBcd, a member of class TDBXRow.
	GetBoolean (see page 10593)	This is GetBoolean, a member of class TDBXRow.
	GetByteLength (see page 10593)	This is GetByteLength, a member of class TDBXRow.
	GetBytes (see page 10594)	This is GetBytes, a member of class TDBXRow.
	GetDate (see page 10594)	This is GetDate, a member of class TDBXRow.
	GetDouble (see page 10594)	This is GetDouble, a member of class TDBXRow.
	GetInt16 (see page 10594)	This is GetInt16, a member of class TDBXRow.
	GetInt32 (see page 10595)	This is GetInt32, a member of class TDBXRow.
	GetInt64 (see page 10595)	This is GetInt64, a member of class TDBXRow.
	GetObjectTypeName (see page 10595)	This is GetObjectTypeName, a member of class TDBXRow.
	GetTime (see page 10595)	This is GetTime, a member of class TDBXRow.
	GetTimeStamp (see page 10595)	This is GetTimeStamp, a member of class TDBXRow.
	GetWideString (see page 10596)	This is GetWideString, a member of class TDBXRow.
	SetBCD (see page 10596)	This is SetBCD, a member of class TDBXRow.
	SetBoolean (see page 10596)	This is SetBoolean, a member of class TDBXRow.
	SetDate (see page 10597)	This is SetDate, a member of class TDBXRow.
	SetDouble (see page 10597)	This is SetDouble, a member of class TDBXRow.
	SetDynamicBytes (see page 10597)	This is SetDynamicBytes, a member of class TDBXRow.
	SetInt16 (see page 10597)	This is SetInt16, a member of class TDBXRow.
	SetInt32 (see page 10597)	This is SetInt32, a member of class TDBXRow.
	SetInt64 (see page 10598)	This is SetInt64, a member of class TDBXRow.
	SetNull (see page 10598)	This is SetNull, a member of class TDBXRow.
	SetString (see page 10598)	This is SetString, a member of class TDBXRow.
	SetTime (see page 10598)	This is SetTime, a member of class TDBXRow.
	SetTimestamp (see page 10599)	This is SetTimestamp, a member of class TDBXRow.
	SetValueType (see page 10599)	This is SetValueType, a member of class TDBXRow.
	SetWideString (see page 10599)	This is SetWideString, a member of class TDBXRow.

Strict protected Methods

	Name	Description
	NotImplemented (see page 10596)	This is NotImplemented, a member of class TDBXRow.

55.1.60.4.1 TDBXRow.GetAnsiString Method

Pascal

```
procedure GetAnsiString(DbxValue: TDBXAnsiStringValue; var AnsiStringBuilder:
TDBXAnsiStringBuilder; var IsNull: LongBool); virtual;
```

C++

```
virtual __fastcall GetAnsiString(TDBXAnsiStringValue DbxValue, TDBXAnsiStringBuilder
AnsiStringBuilder, LongBool IsNull);
```

Description

This is GetAnsiString, a member of class TDBXRow.

55.1.60.4.2 TDBXRow.GetBcd Method

Pascal

```
procedure GetBcd(DbxValue: TDBXBcdValue; var Value: TBcd; var IsNull: LongBool); virtual;
```

C++

```
virtual __fastcall GetBcd(TDBXBcdValue DbxValue, TBcd Value, LongBool IsNull);
```

Description

This is GetBcd, a member of class TDBXRow.

55.1.60.4.3 TDBXRow.GetBoolean Method

Pascal

```
procedure GetBoolean(DbxValue: TDBXBooleanValue; var Value: LongBool; var IsNull: LongBool);
virtual;
```

C++

```
virtual __fastcall GetBoolean(TDBXBooleanValue DbxValue, LongBool Value, LongBool IsNull);
```

Description

This is GetBoolean, a member of class TDBXRow.

55.1.60.4.4 TDBXRow.GetByteLength Method

Pascal

```
procedure GetByteLength(DbxValue: TDBXByteArrayValue; var ByteLength: Int64; var IsNull:
LongBool); virtual;
```

C++

```
virtual __fastcall GetByteLength(TDBXByteArrayValue DbxValue, Int64 ByteLength, LongBool
IsNull);
```

Description

This is GetByteLength, a member of class TDBXRow.

55.1.60.4.5 TDBXRow.GetBytes Method

Pascal

```
procedure GetBytes(DbxValue: TDBXByteArrayValue; Offset: Int64; const Buffer: TBytes;
BufferOffset: Int64; Length: Int64; var ReturnLength: Int64; var IsNull: LongBool); virtual;
```

C++

```
virtual __fastcall GetBytes(TDBXByteArrayValue DbxValue, Int64 Offset, const TBytes Buffer,
Int64 BufferOffset, Int64 Length, Int64 ReturnLength, LongBool IsNull);
```

Description

This is GetBytes, a member of class TDBXRow.

55.1.60.4.6 TDBXRow.GetDate Method

Pascal

```
procedure GetDate(DbxValue: TDBXDateValue; var Value: TDBXDate; var IsNull: LongBool); virtual;
```

C++

```
virtual __fastcall GetDate(TDBXDateValue DbxValue, TDBXDate Value, LongBool IsNull);
```

Description

This is GetDate, a member of class TDBXRow.

55.1.60.4.7 TDBXRow.GetDouble Method

Pascal

```
procedure GetDouble(DbxValue: TDBXDoubleValue; var Value: Double; var IsNull: LongBool);
virtual;
```

C++

```
virtual __fastcall GetDouble(TDBXDoubleValue DbxValue, Double Value, LongBool IsNull);
```

Description

This is GetDouble, a member of class TDBXRow.

55.1.60.4.8 TDBXRow.GetInt16 Method

Pascal

```
procedure GetInt16(DbxValue: TDBXInt16Value; var Value: SmallInt; var IsNull: LongBool);
virtual;
```

C++

```
virtual __fastcall GetInt16(TDBXInt16Value DbxValue, SmallInt Value, LongBool IsNull);
```

Description

This is GetInt16, a member of class TDBXRow.

55.1.60.4.9 TDBXRow.GetInt32 Method

Pascal

```
procedure GetInt32(DbxValue: TDBXInt32Value; var Value: TInt32; var IsNull: LongBool); virtual;
```

C++

```
virtual __fastcall GetInt32(TDBXInt32Value DbxValue, TInt32 Value, LongBool IsNull);
```

Description

This is GetInt32, a member of class TDBXRow.

55.1.60.4.10 TDBXRow.GetInt64 Method

Pascal

```
procedure GetInt64(DbxValue: TDBXInt64Value; var Value: Int64; var IsNull: LongBool); virtual;
```

C++

```
virtual __fastcall GetInt64(TDBXInt64Value DbxValue, Int64 Value, LongBool IsNull);
```

Description

This is GetInt64, a member of class TDBXRow.

55.1.60.4.11 TDBXRow.GetObjectTypeName Method

Pascal

```
function GetObjectTypeName(Ordinal: TInt32): WideString; virtual;
```

C++

```
virtual __fastcall BSTR GetObjectTypeName(TInt32 Ordinal);
```

Description

This is GetObjectTypeName, a member of class TDBXRow.

55.1.60.4.12 TDBXRow.GetTime Method

Pascal

```
procedure GetTime(DbxValue: TDBXTimeValue; var Value: TDBXTime; var IsNull: LongBool); virtual;
```

C++

```
virtual __fastcall GetTime(TDBXTimeValue DbxValue, TDBXTime Value, LongBool IsNull);
```

Description

This is GetTime, a member of class TDBXRow.

55.1.60.4.13 TDBXRow.GetTimeStamp Method

Pascal

```
procedure GetTimeStamp(DbxValue: TDBXTimeStampValue; var Value: TSQLTimeStamp; var IsNull:
LongBool); virtual;
```

C++

```
virtual __fastcall GetTimeStamp(TDBXTimeStampValue DbxValue, TSQLTimeStamp Value, LongBool
IsNull);
```

Description

This is GetTimeStamp, a member of class TDBXRow.

55.1.60.4.14 TDBXRow.GetWideString Method

Pascal

```
procedure GetWideString(DbxValue: TDBXWideStringValue; var WideStringBuilder:
TDBXWideStringBuilder; var IsNull: LongBool); virtual;
```

C++

```
virtual __fastcall GetWideString(TDBXWideStringValue DbxValue, TDBXWideStringBuilder
WideStringBuilder, LongBool IsNull);
```

Description

This is GetWideString, a member of class TDBXRow.

55.1.60.4.15 TDBXRow.NotImplemented Method

Pascal

```
procedure NotImplemented;
```

C++

```
__fastcall NotImplemented();
```

Description

This is NotImplemented, a member of class TDBXRow.

55.1.60.4.16 TDBXRow.SetBCD Method

Pascal

```
procedure SetBCD(DbxValue: TDBXBcdValue; var Value: TBcd); virtual;
```

C++

```
virtual __fastcall SetBCD(TDBXBcdValue DbxValue, TBcd Value);
```

Description

This is SetBCD, a member of class TDBXRow.

55.1.60.4.17 TDBXRow.SetBoolean Method

Pascal

```
procedure SetBoolean(DbxValue: TDBXBooleanValue; Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetBoolean(TDBXBooleanValue DbxValue, Boolean Value);
```

Description

This is SetBoolean, a member of class TDBXRow.

55.1.60.4.18 TDBXRow.SetDate Method

Pascal

```
procedure SetDate(DbxValue: TDBXDateValue; Value: TDBXDate); virtual;
```

C++

```
virtual __fastcall SetDate(TDBXDateValue DbxValue, TDBXDate Value);
```

Description

This is SetDate, a member of class TDBXRow.

55.1.60.4.19 TDBXRow.SetDouble Method

Pascal

```
procedure SetDouble(DbxValue: TDBXDoubleValue; Value: Double); virtual;
```

C++

```
virtual __fastcall SetDouble(TDBXDoubleValue DbxValue, Double Value);
```

Description

This is SetDouble, a member of class TDBXRow.

55.1.60.4.20 TDBXRow.SetDynamicBytes Method

Pascal

```
procedure SetDynamicBytes(DbxValue: TDBXValue; Offset: Int64; const Buffer: TBytes;
BufferOffset: Int64; Length: Int64); virtual;
```

C++

```
virtual __fastcall SetDynamicBytes(TDBXValue DbxValue, Int64 Offset, const TBytes Buffer,
Int64 BufferOffset, Int64 Length);
```

Description

This is SetDynamicBytes, a member of class TDBXRow.

55.1.60.4.21 TDBXRow.SetInt16 Method

Pascal

```
procedure SetInt16(DbxValue: TDBXInt16Value; Value: SmallInt); virtual;
```

C++

```
virtual __fastcall SetInt16(TDBXInt16Value DbxValue, SmallInt Value);
```

Description

This is SetInt16, a member of class TDBXRow.

55.1.60.4.22 TDBXRow.SetInt32 Method

Pascal

```
procedure SetInt32(DbxValue: TDBXInt32Value; Value: TInt32); virtual;
```

C++

```
virtual __fastcall SetInt32(TDBXInt32Value DbxValue, TInt32 Value);
```

Description

This is SetInt32, a member of class TDBXRow.

55.1.60.4.23 TDBXRow.SetInt64 Method

Pascal

```
procedure SetInt64(DbxValue: TDBXInt64Value; Value: Int64); virtual;
```

C++

```
virtual __fastcall SetInt64(TDBXInt64Value DbxValue, Int64 Value);
```

Description

This is SetInt64, a member of class TDBXRow.

55.1.60.4.24 TDBXRow.SetNull Method

Pascal

```
procedure SetNull(DbxValue: TDBXValue); virtual;
```

C++

```
virtual __fastcall SetNull(TDBXValue DbxValue);
```

Description

This is SetNull, a member of class TDBXRow.

55.1.60.4.25 TDBXRow.SetString Method

Pascal

```
procedure SetString(DbxValue: TDBXAnsiStringValue; const Value: String); virtual;
```

C++

```
virtual __fastcall SetString(TDBXAnsiStringValue DbxValue, const AnsiString Value);
```

Description

This is SetString, a member of class TDBXRow.

55.1.60.4.26 TDBXRow.SetTime Method

Pascal

```
procedure SetTime(DbxValue: TDBXTimeValue; Value: TDBXTime); virtual;
```

C++

```
virtual __fastcall SetTime(TDBXTimeValue DbxValue, TDBXTime Value);
```

Description

This is SetTime, a member of class TDBXRow.

55.1.60.4.27 TDBXRow.SetTimestamp Method

Pascal

```
procedure SetTimestamp(DbxValue: TDBXTimeStampValue; var Value: TSQLTimeStamp); virtual;
```

C++

```
virtual __fastcall SetTimestamp(TDBXTimeStampValue DbxValue, TSQLTimeStamp Value);
```

Description

This is SetTimestamp, a member of class TDBXRow.

55.1.60.4.28 TDBXRow.SetValueType Method

Pascal

```
procedure SetValueType(ValueType: TDBXValueType); virtual;
```

C++

```
virtual __fastcall SetValueType(TDBXValueType ValueType);
```

Description

This is SetValueType, a member of class TDBXRow.

55.1.60.4.29 TDBXRow.SetWideString Method

Pascal

```
procedure SetWideString(DbxValue: TDBXWideStringValue; const Value: WideString); virtual;
```

C++

```
virtual __fastcall SetWideString(TDBXWideStringValue DbxValue, const BSTR Value);
```

Description

This is SetWideString, a member of class TDBXRow.

55.1.60.5 TDBXRow Properties

The properties of the TDBXRow class are listed here.

Protected Properties

	Name	Description
	Generation (see page 10599)	This is Generation, a member of class TDBXRow.

55.1.60.5.1 TDBXRow.Generation Property

Pascal

```
property Generation: TInt32;
```

C++

```
__property TInt32 Generation;
```

Description

This is Generation, a member of class TDBXRow.

55.1.61 TDBXRowEx Class

Class Hierarchy

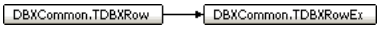

Pascal

```
TDBXRowEx = class(TDBXRow);
```

C++

```
class TDBXRowEx : public TDBXRow;
```

File

DBXCommon

Description

This is class DBXCommon.TDBXRowEx.

55.1.61.1 TDBXRowEx Members

The following tables list the members exposed by TDBXRowEx.

Protected Fields

	Name	Description
	FGeneration (see page 10592)	This is FGeneration, a member of class TDBXRow.

Protected Methods

	Name	Description
	Create (see page 10591)	Creates an instance of TDBXRow (see page 10590)
V	CreateCustomValue (see page 10601)	This is CreateCustomValue, a member of class TDBXRowEx.
V	GetAnsiString (see page 10593)	This is GetAnsiString, a member of class TDBXRow.
V	GetBcd (see page 10593)	This is GetBcd, a member of class TDBXRow.
V	GetBoolean (see page 10593)	This is GetBoolean, a member of class TDBXRow.
V	GetByteLength (see page 10593)	This is GetByteLength, a member of class TDBXRow.
V	GetBytes (see page 10594)	This is GetBytes, a member of class TDBXRow.
V	GetDate (see page 10594)	This is GetDate, a member of class TDBXRow.
V	GetDouble (see page 10594)	This is GetDouble, a member of class TDBXRow.
V	GetInt16 (see page 10594)	This is GetInt16, a member of class TDBXRow.
V	GetInt32 (see page 10595)	This is GetInt32, a member of class TDBXRow.
V	GetInt64 (see page 10595)	This is GetInt64, a member of class TDBXRow.
V	GetLength (see page 10602)	This is GetLength, a member of class TDBXRowEx.
V	GetObjectTypeName (see page 10595)	This is GetObjectTypeName, a member of class TDBXRow.
V	GetStream (see page 10602)	This is the overview for the GetStream method overload.
V	GetStreamBytes (see page 10603)	This is GetStreamBytes, a member of class TDBXRowEx.
V	GetStreamLength (see page 10603)	This is GetStreamLength, a member of class TDBXRowEx.
V	GetTime (see page 10595)	This is GetTime, a member of class TDBXRow.
V	GetTimeStamp (see page 10595)	This is GetTimeStamp, a member of class TDBXRow.

	GetWideChars (see page 10603)	This is GetWideChars, a member of class TDBXRowEx.
	GetWideString (see page 10596)	This is GetWideString, a member of class TDBXRow.
	SetBCD (see page 10596)	This is SetBCD, a member of class TDBXRow.
	SetBoolean (see page 10596)	This is SetBoolean, a member of class TDBXRow.
	SetDate (see page 10597)	This is SetDate, a member of class TDBXRow.
	SetDouble (see page 10597)	This is SetDouble, a member of class TDBXRow.
	SetDynamicBytes (see page 10597)	This is SetDynamicBytes, a member of class TDBXRow.
	SetInt16 (see page 10597)	This is SetInt16, a member of class TDBXRow.
	SetInt32 (see page 10597)	This is SetInt32, a member of class TDBXRow.
	SetInt64 (see page 10598)	This is SetInt64, a member of class TDBXRow.
	SetNull (see page 10598)	This is SetNull, a member of class TDBXRow.
	SetStream (see page 10603)	This is SetStream, a member of class TDBXRowEx.
	SetString (see page 10598)	This is SetString, a member of class TDBXRow.
	SetTime (see page 10598)	This is SetTime, a member of class TDBXRow.
	SetTimestamp (see page 10599)	This is SetTimestamp, a member of class TDBXRow.
	SetValueType (see page 10599)	This is SetValueType, a member of class TDBXRow.
	SetWideChars (see page 10604)	This is SetWideChars, a member of class TDBXRowEx.
	SetWideString (see page 10599)	This is SetWideString, a member of class TDBXRow.
	UseExtendedTypes (see page 10604)	This is UseExtendedTypes, a member of class TDBXRowEx.

Protected Properties

	Name	Description
	Generation (see page 10599)	This is Generation, a member of class TDBXRow.

Strict protected Fields

	Name	Description
	FDBXContext (see page 10591)	This is FDBXContext, a member of class TDBXRow.

Strict protected Methods

	Name	Description
	NotImplemented (see page 10596)	This is NotImplemented, a member of class TDBXRow.

55.1.61.2 TDBXRowEx Methods

The methods of the TDBXRowEx class are listed here.

Protected Methods

	Name	Description
	CreateCustomValue (see page 10601)	This is CreateCustomValue, a member of class TDBXRowEx.
	GetLength (see page 10602)	This is GetLength, a member of class TDBXRowEx.
	GetStream (see page 10602)	This is the overview for the GetStream method overload.
	GetStreamBytes (see page 10603)	This is GetStreamBytes, a member of class TDBXRowEx.
	GetStreamLength (see page 10603)	This is GetStreamLength, a member of class TDBXRowEx.
	GetWideChars (see page 10603)	This is GetWideChars, a member of class TDBXRowEx.
	SetStream (see page 10603)	This is SetStream, a member of class TDBXRowEx.
	SetWideChars (see page 10604)	This is SetWideChars, a member of class TDBXRowEx.
	UseExtendedTypes (see page 10604)	This is UseExtendedTypes, a member of class TDBXRowEx.

55.1.61.2.1 TDBXRowEx.CreateCustomValue Method

Pascal

```
function CreateCustomValue(ValueType: TDBXValueType): TDBXValue; virtual;
```

C++

```
virtual __fastcall TDBXValue CreateCustomValue(TDBXValueType ValueType);
```

Description

This is CreateCustomValue, a member of class TDBXRowEx.

55.1.61.2.2 TDBXRowEx.GetLength Method

Pascal

```
procedure GetLength(DbxValue: TDBXValue; var ByteLength: Int64; var IsNull: LongBool); virtual;
```

C++

```
virtual __fastcall GetLength(TDBXValue DbxValue, Int64 ByteLength, LongBool IsNull);
```

Description

This is GetLength, a member of class TDBXRowEx.

55.1.61.2.3 GetStream Method

This is the overview for the GetStream method overload.

Overload List

	Name	Description
	TDBXRowEx.GetStream (TDBXStreamValue, TStream, LongBool) (see page 10602)	This is GetStream, a member of class TDBXRowEx.
	TDBXRowEx.GetStream (TDBXWideStringValue, TStream, LongBool) (see page 10602)	This is GetStream, a member of class TDBXRowEx.

55.1.61.2.3.1 TDBXRowEx.GetStream Method (TDBXStreamValue, TStream, LongBool)

Pascal

```
procedure GetStream(DbxValue: TDBXStreamValue; var Stream: TStream; var IsNull: LongBool);
virtual; overload;
```

C++

```
virtual __fastcall GetStream(TDBXStreamValue DbxValue, TStream Stream, LongBool IsNull);
```

Description

This is GetStream, a member of class TDBXRowEx.

55.1.61.2.3.2 TDBXRowEx.GetStream Method (TDBXWideStringValue, TStream, LongBool)

Pascal

```
procedure GetStream(DbxValue: TDBXWideStringValue; var Stream: TStream; var IsNull: LongBool);
virtual; overload;
```

C++

```
virtual __fastcall GetStream(TDBXWideStringValue DbxValue, TStream Stream, LongBool IsNull);
```

Description

This is GetStream, a member of class TDBXRowEx.

55.1.61.2.4 TDBXRowEx.GetStreamBytes Method

Pascal

```
procedure GetStreamBytes(DbxValue: TDBXStreamValue; const Buffer: TBytes; BufferOffset: Int64;
Length: Int64; ReturnLength: Int64; var IsNull: LongBool); virtual;
```

C++

```
virtual __fastcall GetStreamBytes(TDBXStreamValue DbxValue, const TBytes Buffer, Int64
BufferOffset, Int64 Length, Int64 ReturnLength, LongBool IsNull);
```

Description

This is GetStreamBytes, a member of class TDBXRowEx.

55.1.61.2.5 TDBXRowEx.GetStreamLength Method

Pascal

```
procedure GetStreamLength(DbxValue: TDBXStreamValue; StreamLength: Int64; var IsNull:
LongBool); virtual;
```

C++

```
virtual __fastcall GetStreamLength(TDBXStreamValue DbxValue, Int64 StreamLength, LongBool
IsNull);
```

Description

This is GetStreamLength, a member of class TDBXRowEx.

55.1.61.2.6 TDBXRowEx.GetWideChars Method

Pascal

```
procedure GetWideChars(DbxValue: TDBXWideStringValue; var WideChars: TDBXWideChars; var Count:
Integer; var IsNull: LongBool); virtual;
```

C++

```
virtual __fastcall GetWideChars(TDBXWideStringValue DbxValue, TDBXWideChars WideChars, int
Count, LongBool IsNull);
```

Description

This is GetWideChars, a member of class TDBXRowEx.

55.1.61.2.7 TDBXRowEx.SetStream Method

Pascal

```
procedure SetStream(DbxValue: TDBXStreamValue; StreamReader: TDBXStreamReader); virtual;
overload;
```

C++

```
virtual __fastcall SetStream(TDBXStreamValue DbxValue, TDBXStreamReader StreamReader);
```

Description

This is SetStream, a member of class TDBXRowEx.

55.1.61.2.8 TDBXRowEx.SetWideChars Method

Pascal

```
procedure SetWideChars(DbxValue: TDBXWideStringValue; const Value: WideString); virtual;
```

C++

```
virtual __fastcall SetWideChars(TDBXWideStringValue DbxValue, const BSTR Value);
```

Description

This is SetWideChars, a member of class TDBXRowEx.

55.1.61.2.9 TDBXRowEx.UseExtendedTypes Method

Pascal

```
function UseExtendedTypes: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean UseExtendedTypes();
```

Description

This is UseExtendedTypes, a member of class TDBXRowEx.

55.1.62 TDBXStateItem Class

Class Hierarchy

Pascal

```
TDBXStateItem = class;
```

C++

```
class TDBXStateItem;
```

File

DBXCommon

Description

This is class DBXCommon.TDBXStateItem.

55.1.62.1 TDBXStateItem Members

The following tables list the members exposed by TDBXStateItem.

Public Methods

	Name	Description
	Close (see page 10605)	This is Close, a member of class TDBXStateItem.

55.1.62.2 TDBXStateItem Methods

The methods of the TDBXStateItem class are listed here.

Public Methods

	Name	Description
	Close (see page 10605)	This is Close, a member of class TDBXStateItem.

55.1.62.2.1 TDBXStateItem.Close Method

Pascal

```
procedure Close; virtual; abstract;
```

C++

```
virtual __fastcall Close() = 0;
```

Description

This is Close, a member of class TDBXStateItem.

55.1.63 TDBXStateItemList Class

Class Hierarchy

DBXCommon.TDBXStateItemList

Pascal

```
TDBXStateItemList = class;
```

C++

```
class TDBXStateItemList;
```

File

DBXCommon

Description

This is class DBXCommon.TDBXStateItemList.

55.1.63.1 TDBXStateItemList Members

The following tables list the members exposed by TDBXStateItemList.

Public Methods

	Name	Description
	AddStateItem (see page 10606)	This is AddStateItem, a member of class TDBXStateItemList.
	Destroy (see page 10606)	This is Destroy, a member of class TDBXStateItemList.
	GetStateItem (see page 10606)	This is GetStateItem, a member of class TDBXStateItemList.
	RemoveAndFreeStateItem (see page 10606)	This is RemoveAndFreeStateItem, a member of class TDBXStateItemList.

55.1.63.2 TDBXStateItemList.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXStateItemList();
```

Description

This is Destroy, a member of class TDBXStateItemList.

55.1.63.3 TDBXStateItemList Methods

The methods of the TDBXStateItemList class are listed here.

Public Methods

	Name	Description
	AddStateItem (see page 10606)	This is AddStateItem, a member of class TDBXStateItemList.
	GetStateItem (see page 10606)	This is GetStateItem, a member of class TDBXStateItemList.
	RemoveAndFreeStateItem (see page 10606)	This is RemoveAndFreeStateItem, a member of class TDBXStateItemList.

55.1.63.3.1 TDBXStateItemList.AddStateItem Method

Pascal

```
procedure AddStateItem(Name: WideString; Item: TDBXStateItem);
```

C++

```
__fastcall AddStateItem(BSTR Name, TDBXStateItem Item);
```

Description

This is AddStateItem, a member of class TDBXStateItemList.

55.1.63.3.2 TDBXStateItemList.GetStateItem Method

Pascal

```
function GetStateItem(Name: WideString): TDBXStateItem;
```

C++

```
__fastcall TDBXStateItem GetStateItem(BSTR Name);
```

Description

This is GetStateItem, a member of class TDBXStateItemList.

55.1.63.3.3 TDBXStateItemList.RemoveAndFreeStateItem Method

Pascal

```
procedure RemoveAndFreeStateItem(Name: WideString);
```

C++

```
__fastcall RemoveAndFreeStateItem(BSTR Name);
```

Description

This is RemoveAndFreeStateItem, a member of class TDBXStateItemList.

55.1.64 TDBXStreamReader Class

Used internally by driver implementations and TDBXStreamValue (see page 10608).

Class Hierarchy

DBXCommon.TDBXStreamReader

Pascal

```
TDBXStreamReader = class;
```

C++

```
class TDBXStreamReader;
```

File

DBXCommon

Description

TDBXStreamReader is used internally by driver implementations and TDBXStreamValue (see page 10608).

See Also

TDBXStreamValue (see page 10608)

55.1.64.1 TDBXStreamReader Members

The following tables list the members exposed by TDBXStreamReader.

Public Methods

	Name	Description
	Eos (see page 10608)	This is Eos, a member of class TDBXStreamReader.
	Read (see page 10608)	This is Read, a member of class TDBXStreamReader.
	Size (see page 10608)	This is Size, a member of class TDBXStreamReader.

55.1.64.2 TDBXStreamReader Methods

The methods of the TDBXStreamReader class are listed here.

Public Methods

	Name	Description
	Eos (see page 10608)	This is Eos, a member of class TDBXStreamReader.
	Read (see page 10608)	This is Read, a member of class TDBXStreamReader.
	Size (see page 10608)	This is Size, a member of class TDBXStreamReader.

55.1.64.2.1 TDBXStreamReader.Eos Method

Pascal

```
function Eos: Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean Eos() = 0;
```

Description

This is Eos, a member of class TDBXStreamReader.

55.1.64.2.2 TDBXStreamReader.Read Method

Pascal

```
function Read(const Buffer: TBytes; Offset: Integer; Size: Integer): Integer; virtual;
abstract;
```

C++

```
virtual __fastcall int Read(const TBytes Buffer, int Offset, int Size) = 0;
```

Description

This is Read, a member of class TDBXStreamReader.

55.1.64.2.3 TDBXStreamReader.Size Method

Pascal

```
function Size: Int64; virtual; abstract;
```

C++

```
virtual __fastcall Int64 Size() = 0;
```

Description

This is Size, a member of class TDBXStreamReader.

55.1.65 TDBXStreamValue Class

Implementation of TDBXWritableValue (see page 10661) for TDBXDataTypes.ByteArrayType values.

Class Hierarchy

Pascal

```
TDBXStreamValue = class(TDBXByteArrayValue);
```

C++

```
class TDBXStreamValue : public TDBXByteArrayValue;
```

File

DBXCommon

Description

TDBXStreamValue implements TDBXWritableValue (see page 10661) for TDBXDataTypes.ByteArrayType values.

See Also

TDBXByteStreamReader (see page 10374)

TDBXDataTypes (see page 10447)

TDBXStreamReader (see page 10607)

TDBXWritableValue (see page 10661)

55.1.65.1 TDBXStreamValue Members

The following tables list the members exposed by TDBXStreamValue.

Protected Methods

	Name	Description
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.
V	SetPendingValue (see page 10666)	This is SetPendingValue, a member of class TDBXWritableValue.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Methods

	Name	Description
	Create (see page 10610)	Creates an instance of TDBXStreamValue (see page 10608)
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
V	Destroy (see page 10610)	Destroy a TDBXByteArrayValue instance
V	GetAnsiString (see page 10629)	Gets the string value for ANSI strings.
V	GetBcd (see page 10629)	Gets the BCD value for binary-compressed decimal data types.
V	GetBoolean (see page 10629)	Gets the boolean value for boolean data types.
V	GetBytes (see page 10366)	Gets the byte array value for byte array and blob types.
V	GetBytes (see page 10629)	Gets the byte array value for byte array and blob types.
V	GetDate (see page 10630)	Gets the date value for date data types.
V	GetDouble (see page 10630)	Gets the double floating point value for double data types.
V	GetInt16 (see page 10631)	Gets the smallInt value for 16-byte integer data types.
V	GetInt32 (see page 10631)	Gets the 32-byte integer value for 32-byte integer data types.
V	GetInt64 (see page 10631)	Gets the 64-byte integer value for 64-byte integer data types.
V	GetStream (see page 10611)	This is GetStream, a member of class TDBXStreamValue.
V	GetTime (see page 10631)	Gets the time value for time data types.
V	GetTimeStamp (see page 10632)	Gets the timestamp value for timestamp data types.
V	GetValueSize (see page 10611)	Gets the size in bytes for values.
V	GetWideString (see page 10632)	Gets the wide string value for wide string data types.
V	IsNull (see page 10367)	Returns true if the value is **null**.
A	IsNull (see page 10632)	Returns true if the value is **null**.
V	SetAnsiString (see page 10663)	Sets the string value for ANSI strings.
V	SetBcd (see page 10664)	Sets the BCD value for binary-compressed decimal data types.
V	SetBoolean (see page 10664)	Sets the boolean value for boolean data types..
V	SetDate (see page 10664)	Sets the date value for date data types..
V	SetDouble (see page 10664)	Sets the double floating point value for double data types.
V	SetDynamicBytes (see page 10611)	Sets the dynamic array value for byte array and blob types.

	SetInt16 (see page 10665)	Sets the the smallInt value for 16-byte integer data types.
	SetInt32 (see page 10665)	Sets the 32-byte integer value for 32-byte integer data types.
	SetInt64 (see page 10665)	Sets the 64-byte integer value for 64-byte integer data types.
	SetNull (see page 10666)	Sets a writable value of any type to null.
	SetStaticBytes (see page 10367)	Sets the open array value for byte array and blob types
	SetStaticBytes (see page 10666)	Sets the open array value for byte array and blob types
	SetStream (see page 10611)	This is SetStream, a member of class TDBXStreamValue.
	SetTime (see page 10666)	Sets the time value for time data types.
	SetTimeStamp (see page 10667)	Sets the timestamp value for timestamp data types.
	SetWideString (see page 10667)	Sets the wide string value for wide string data types.
	UpdateType (see page 10612)	This is UpdateType, a member of class TDBXStreamValue.

Public Properties

	Name	Description
	ValueType (see page 10633)	Gets the value type for this value.

55.1.65.2 TDBXStreamValue.Create Constructor

Creates an instance of TDBXStreamValue (see page 10608)

Pascal

```
constructor Create(ValueType: TDBXValueType);
```

C++

```
__fastcall TDBXStreamValue(TDBXValueType ValueType);
```

Description

This constructor creates and initializes an instance of TDBXStreamValue (see page 10608).

See Also

Destroy (see page 10610)

55.1.65.3 TDBXStreamValue.Destroy Destructor

Destroy a TDBXByteArrayValue instance

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXStreamValue();
```

Description

This method destroys a TDBXByteArrayValue instance.

55.1.65.4 TDBXStreamValue Methods

The methods of the TDBXStreamValue class are listed here.

Public Methods

	Name	Description
	GetStream (see page 10611)	This is GetStream, a member of class TDBXStreamValue.
	GetValueSize (see page 10611)	Gets the size in bytes for values.
	SetDynamicBytes (see page 10611)	Sets the dynamic array value for byte array and blob types.
	SetStream (see page 10611)	This is SetStream, a member of class TDBXStreamValue.
	UpdateType (see page 10612)	This is UpdateType, a member of class TDBXStreamValue.

55.1.65.4.1 TDBXStreamValue.GetStream Method

Pascal

```
function GetStream: TStream; virtual;
```

C++

```
virtual __fastcall TStream GetStream();
```

Description

This is GetStream, a member of class TDBXStreamValue.

55.1.65.4.2 TDBXStreamValue.GetValueSize Method

Gets the size in bytes for values.

Pascal

```
function GetValueSize: Int64; override;
```

C++

```
virtual __fastcall Int64 GetValueSize();
```

Description

Gets the size in bytes for a TDBXValue. For string data types the number of characters is returned. Throws a TDBXError exception if IsNull is true.

55.1.65.4.3 TDBXStreamValue.SetDynamicBytes Method

Sets the dynamic array value for byte array and blob types.

Pascal

```
procedure SetDynamicBytes(Offset: Int64; const Buffer: TBytes; BufferOffset: Int64; Count:
Int64); override;
```

C++

```
virtual __fastcall SetDynamicBytes(Int64 Offset, const TBytes Buffer, Int64 BufferOffset,
Int64 Count);
```

Description

Sets the dynamic array value for TDBXValue instances with a data type of BytesType and .BlobType.

55.1.65.4.4 TDBXStreamValue.SetStream Method

Pascal

```
procedure SetStream(const Stream: TStream); virtual;
```

C++

```
virtual __fastcall SetStream(const TStream Stream);
```

Description

This is SetStream, a member of class TDBXStreamValue.

55.1.65.4.5 TDBXStreamValue.UpdateType Method

Pascal

```
procedure UpdateType; override;
```

C++

```
virtual __fastcall UpdateType();
```

Description

This is UpdateType, a member of class TDBXStreamValue.

55.1.66 TDBXTimeStampValue Class

Implementation of writable timestamp values.

Class Hierarchy

Pascal

```
TDBXTimeStampValue = class(TDBXWritableValue);
```

C++

```
class TDBXTimeStampValue : public TDBXWritableValue;
```

File

DBXCommon

Description

Implementation of TDBXWritableValue (see page 10661) for TDBXDataTypes.TimeStampType values.

55.1.66.1 TDBXTimeStampValue Members

The following tables list the members exposed by TDBXTimeStampValue.

Protected Methods

	Name	Description
	Create (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
	Create (see page 10662)	Creates an instance of TDBXWritableValue (see page 10661)
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.
V	SetPendingValue (see page 10666)	This is SetPendingValue, a member of class TDBXWritableValue.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Methods

	Name	Description
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
V	Destroy (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
V	Destroy (see page 10663)	Destroys a TDBXWritableValue (see page 10661) instance.
V	GetAnsiString (see page 10629)	Gets the string value for ANSI strings.
V	GetBcd (see page 10629)	Gets the BCD value for binary-compressed decimal data types.
V	GetBoolean (see page 10629)	Gets the boolean value for boolean data types.
V	GetBytes (see page 10629)	Gets the byte array value for byte array and blob types.
V	GetDate (see page 10630)	Gets the date value for date data types.
V	GetDouble (see page 10630)	Gets the double floating point value for double data types.
V	GetInt16 (see page 10631)	Gets the smallInt value for 16-byte integer data types.
V	GetInt32 (see page 10631)	Gets the 32-byte integer value for 32-byte integer data types.
V	GetInt64 (see page 10631)	Gets the 64-byte integer value for 64-byte integer data types.
V	GetTime (see page 10631)	Gets the time value for time data types.
V	GetTimeStamp (see page 10614)	Gets the timestamp value for timestamp data types.
V	GetValueSize (see page 10632)	Gets the size in bytes for values.
V	GetWideString (see page 10632)	Gets the wide string value for wide string data types.
V	IsNull (see page 10614)	Returns true if the value is **null**.
V	SetAnsiString (see page 10663)	Sets the string value for ANSI strings.
V	SetBcd (see page 10664)	Sets the BCD value for binary-compressed decimal data types.
V	SetBoolean (see page 10664)	Sets the boolean value for boolean data types..
V	SetDate (see page 10664)	Sets the date value for date data types..
V	SetDouble (see page 10664)	Sets the double floating point value for double data types.
V	SetDynamicBytes (see page 10665)	Sets the dynamic array value for byte array and blob types.
V	SetInt16 (see page 10665)	Sets the the smallInt value for 16-byte integer data types.
V	SetInt32 (see page 10665)	Sets the 32-byte integer value for 32-byte integer data types.
V	SetInt64 (see page 10665)	Sets the 64-byte integer value for 64-byte integer data types.
V	SetNull (see page 10666)	Sets a writable value of any type to null.
V	SetStaticBytes (see page 10666)	Sets the open array value for byte array and blob types
V	SetTime (see page 10666)	Sets the time value for time data types.
V	SetTimeStamp (see page 10614)	Sets the timestamp value for timestamp data types.
V	SetWideString (see page 10667)	Sets the wide string value for wide string data types.

Public Properties

	Name	Description
R	ValueType (see page 10633)	Gets the value type for this value.

55.1.66.2 TDBXTimeStampValue Methods

The methods of the TDBXTimeStampValue class are listed here.

Public Methods

	Name	Description
V	GetTimeStamp (see page 10614)	Gets the timestamp value for timestamp data types.
V	IsNull (see page 10614)	Returns true if the value is **null**.
V	SetTimeStamp (see page 10614)	Sets the timestamp value for timestamp data types.

55.1.66.2.1 TDBXTimeStampValue.GetTimeStamp Method

Gets the timestamp value for timestamp data types.

Pascal

```
function GetTimeStamp: TSQLTimeStamp; override;
```

C++

```
virtual __fastcall TSQLTimeStamp GetTimeStamp();
```

Description

Gets the timestamp value for TDBXValue (see page 10626) instances with a data types of TimeStampType. Throws a TDBXError (see page 10486) exception if IsNull (see page 10614) is true.

55.1.66.2.2 TDBXTimeStampValue.IsNull Method

Returns true if the value is **null**.

Pascal

```
function IsNull: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsNull();
```

Description

Returns true if the value is **null**. This method should be called before calling any of the Get methods. If the value is null, the Get methods will throw TDBXError (see page 10486) exception if they are called.

55.1.66.2.3 TDBXTimeStampValue.SetTimeStamp Method

Sets the timestamp value for timestamp data types.

Pascal

```
procedure SetTimeStamp(const Value: TSQLTimeStamp); override;
```

C++

```
virtual __fastcall SetTimeStamp(const TSQLTimeStamp Value);
```

Description

Sets the timestamp value for TDBXValue (see page 10626) instances with a data types of TimeStampType.

55.1.67 TDBXTimeValue Class

Implementation of writable time values.

Class Hierarchy

Pascal

```
TDBXTimeValue = class(TDBXWritableValue);
```

C++

```
class TDBXTimeValue : public TDBXWritableValue;
```

File

DBXCommon

Description

Implementation of TDBXWritableValue (see page 10661) for TDBXDataTypes.TimeType values.

55.1.67.1 TDBXTimeValue Members

The following tables list the members exposed by TDBXTimeValue.

Protected Methods

	Name	Description
	Create (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
	Create (see page 10662)	Creates an instance of TDBXWritableValue (see page 10661)
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.
V	SetPendingValue (see page 10666)	This is SetPendingValue, a member of class TDBXWritableValue.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Methods

	Name	Description
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
V	Destroy (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
V	Destroy (see page 10663)	Destroys a TDBXWritableValue (see page 10661) instance.
V	GetAnsiString (see page 10629)	Gets the string value for ANSI strings.
V	GetBcd (see page 10629)	Gets the BCD value for binary-compressed decimal data types.
V	GetBoolean (see page 10629)	Gets the boolean value for boolean data types.
V	GetBytes (see page 10629)	Gets the byte array value for byte array and blob types.
V	GetDate (see page 10630)	Gets the date value for date data types.
V	GetDouble (see page 10630)	Gets the double floating point value for double data types.
V	GetInt16 (see page 10631)	Gets the smallInt value for 16-byte integer data types.
V	GetInt32 (see page 10631)	Gets the 32-byte integer value for 32-byte integer data types.
V	GetInt64 (see page 10631)	Gets the 64-byte integer value for 64-byte integer data types.
V	GetTime (see page 10616)	Gets the time value for time data types.
V	GetTimeStamp (see page 10632)	Gets the timestamp value for timestamp data types.
V	GetValueSize (see page 10632)	Gets the size in bytes for values.
V	GetWideString (see page 10632)	Gets the wide string value for wide string data types.
V	IsNull (see page 10616)	Returns true if the value is **null**.
V	SetAnsiString (see page 10663)	Sets the string value for ANSI strings.
V	SetBcd (see page 10664)	Sets the BCD value for binary-compressed decimal data types.
V	SetBoolean (see page 10664)	Sets the boolean value for boolean data types..
V	SetDate (see page 10664)	Sets the date value for date data types..
V	SetDouble (see page 10664)	Sets the double floating point value for double data types.
V	SetDynamicBytes (see page 10665)	Sets the dynamic array value for byte array and blob types.
V	SetInt16 (see page 10665)	Sets the the smallInt value for 16-byte integer data types.

	SetInt32 (see page 10665)	Sets the 32-byte integer value for 32-byte integer data types.
	SetInt64 (see page 10665)	Sets the 64-byte integer value for 64-byte integer data types.
	SetNull (see page 10666)	Sets a writable value of any type to null.
	SetStaticBytes (see page 10666)	Sets the open array value for byte array and blob types
	SetTime (see page 10617)	Sets the time value for time data types.
	SetTimeStamp (see page 10667)	Sets the timestamp value for timestamp data types.
	SetWideString (see page 10667)	Sets the wide string value for wide string data types.

Public Properties

	Name	Description
	ValueType (see page 10633)	Gets the value type for this value.

55.1.67.2 TDBXTimeValue Methods

The methods of the TDBXTimeValue class are listed here.

Public Methods

	Name	Description
	GetTime (see page 10616)	Gets the time value for time data types.
	IsNull (see page 10616)	Returns true if the value is **null**.
	SetTime (see page 10617)	Sets the time value for time data types.

55.1.67.2.1 TDBXTimeValue.GetTime Method

Gets the time value for time data types.

Pascal

```
function GetTime: TDBXTime; override;
```

C++

```
virtual __fastcall TDBXTime GetTime();
```

Description

Gets the time value for TDBXValue (see page 10626) instances with a data type of TimeType. Throws a TDBXError (see page 10486) exception if IsNull (see page 10616) is true.

55.1.67.2.2 TDBXTimeValue.IsNull Method

Returns true if the value is **null**.

Pascal

```
function IsNull: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsNull();
```

Description

Returns true if the value is **null**. This method should be called before calling any of the Get methods. If the value is null, the Get methods will throw TDBXError (see page 10486) exception if they are called.

55.1.67.2.3 TDBXTimeValue.SetTime Method

Sets the time value for time data types.

Pascal

```
procedure SetTime(Value: TDBXTime); override;
```

C++

```
virtual __fastcall SetTime(TDBXTime Value);
```

Description

Sets the time value for TDBXValue (see page 10626) instances with a data type of TimeType.

55.1.68 TDBXTraceFlags Class

Flags to indicate tracing type.

Class Hierarchy

DBXCommon.TDBXTraceFlags

Pascal

```
TDBXTraceFlags = class;
```

C++

```
class TDBXTraceFlags;
```

File

DBXCommon

Description

These flags indicate what items should be traced. Logically OR the flags together for all items that you want to trace.

This table lists the traceable items.

Constant	Description
None (see page 10621)	Trace nothing
Prepare (see page 10621)	Prepared query statements
Execute (see page 10620)	Executed query statements
Error (see page 10620)	Errors
Command (see page 10619)	Command (see page 10619) related operations
Connect (see page 10619)	Connect (see page 10619) and disconnect
Transact (see page 10621)	Transaction commit, rollback
Blob (see page 10619)	Blob (see page 10619) access
Misc (see page 10620)	Miscellaneous
Vendor (see page 10622)	Vendor (see page 10622) specific operations
Parameter (see page 10621)	TDBXParameter (see page 10545) access
Reader (see page 10621)	TDBXReader (see page 10576) operations

DriverLoad (see page 10619)	Driver (see page 10619) loading operations
MetaData (see page 10620)	Metadata access operations
Driver (see page 10619)	Driver (see page 10619) operations

See Also

TDBXParameter (see page 10545)

TDBXReader (see page 10576)

55.1.68.1 TDBXTraceFlags Members

The following tables list the members exposed by TDBXTraceFlags.

Public Constants

	Name	Description
	Blob (see page 10619)	This is Blob, a member of class TDBXTraceFlags.
	Command (see page 10619)	This is Command, a member of class TDBXTraceFlags.
	Connect (see page 10619)	This is Connect, a member of class TDBXTraceFlags.
	Driver (see page 10619)	This is Driver, a member of class TDBXTraceFlags.
	DriverLoad (see page 10619)	This is DriverLoad, a member of class TDBXTraceFlags.
	Error (see page 10620)	This is Error, a member of class TDBXTraceFlags.
	Execute (see page 10620)	This is Execute, a member of class TDBXTraceFlags.
	MetaData (see page 10620)	This is MetaData, a member of class TDBXTraceFlags.
	Misc (see page 10620)	This is Misc, a member of class TDBXTraceFlags.
	None (see page 10621)	This is None, a member of class TDBXTraceFlags.
	Parameter (see page 10621)	This is Parameter, a member of class TDBXTraceFlags.
	Prepare (see page 10621)	This is Prepare, a member of class TDBXTraceFlags.
	Reader (see page 10621)	This is Reader, a member of class TDBXTraceFlags.
	Transact (see page 10621)	This is Transact, a member of class TDBXTraceFlags.
	Vendor (see page 10622)	This is Vendor, a member of class TDBXTraceFlags.

55.1.68.2 TDBXTraceFlags Constants

The constants of the TDBXTraceFlags class are listed here.

Public Constants

	Name	Description
	Blob (see page 10619)	This is Blob, a member of class TDBXTraceFlags.
	Command (see page 10619)	This is Command, a member of class TDBXTraceFlags.
	Connect (see page 10619)	This is Connect, a member of class TDBXTraceFlags.
	Driver (see page 10619)	This is Driver, a member of class TDBXTraceFlags.
	DriverLoad (see page 10619)	This is DriverLoad, a member of class TDBXTraceFlags.
	Error (see page 10620)	This is Error, a member of class TDBXTraceFlags.
	Execute (see page 10620)	This is Execute, a member of class TDBXTraceFlags.
	MetaData (see page 10620)	This is MetaData, a member of class TDBXTraceFlags.
	Misc (see page 10620)	This is Misc, a member of class TDBXTraceFlags.
	None (see page 10621)	This is None, a member of class TDBXTraceFlags.
	Parameter (see page 10621)	This is Parameter, a member of class TDBXTraceFlags.
	Prepare (see page 10621)	This is Prepare, a member of class TDBXTraceFlags.
	Reader (see page 10621)	This is Reader, a member of class TDBXTraceFlags.

	Transact (see page 10621)	This is Transact, a member of class TDBXTraceFlags.
	Vendor (see page 10622)	This is Vendor, a member of class TDBXTraceFlags.

55.1.68.2.1 TDBXTraceFlags.Blob Constant

Pascal

```
const Blob = $0040;
```

C++

```
Blob = 0x0040;
```

Description

This is Blob, a member of class TDBXTraceFlags.

55.1.68.2.2 TDBXTraceFlags.Command Constant

Pascal

```
const Command = $0008;
```

C++

```
Command = 0x0008;
```

Description

This is Command, a member of class TDBXTraceFlags.

55.1.68.2.3 TDBXTraceFlags.Connect Constant

Pascal

```
const Connect = $0010;
```

C++

```
Connect = 0x0010;
```

Description

This is Connect, a member of class TDBXTraceFlags.

55.1.68.2.4 TDBXTraceFlags.Driver Constant

Pascal

```
const Driver = $2000;
```

C++

```
Driver = 0x2000;
```

Description

This is Driver, a member of class TDBXTraceFlags.

55.1.68.2.5 TDBXTraceFlags.DriverLoad Constant

Pascal

```
const DriverLoad = $0800;
```

C++

```
DriverLoad = 0x0800;
```

Description

This is DriverLoad, a member of class TDBXTraceFlags.

55.1.68.2.6 TDBXTraceFlags.Error Constant

Pascal

```
const Error = $0004;
```

C++

```
Error = 0x0004;
```

Description

This is Error, a member of class TDBXTraceFlags.

55.1.68.2.7 TDBXTraceFlags.Execute Constant

Pascal

```
const Execute = $0002;
```

C++

```
Execute = 0x0002;
```

Description

This is Execute, a member of class TDBXTraceFlags.

55.1.68.2.8 TDBXTraceFlags.MetaData Constant

Pascal

```
const MetaData = $1000;
```

C++

```
MetaData = 0x1000;
```

Description

This is MetaData, a member of class TDBXTraceFlags.

55.1.68.2.9 TDBXTraceFlags.Misc Constant

Pascal

```
const Misc = $0080;
```

C++

```
Misc = 0x0080;
```

Description

This is Misc, a member of class TDBXTraceFlags.

55.1.68.2.10 TDBXTraceFlags.None Constant

Pascal

```
const None = $0000;
```

C++

```
None = 0x0000;
```

Description

This is None, a member of class TDBXTraceFlags.

55.1.68.2.11 TDBXTraceFlags.Parameter Constant

Pascal

```
const Parameter = $0200;
```

C++

```
Parameter = 0x0200;
```

Description

This is Parameter, a member of class TDBXTraceFlags.

55.1.68.2.12 TDBXTraceFlags.Prepare Constant

Pascal

```
const Prepare = $0001;
```

C++

```
Prepare = 0x0001;
```

Description

This is Prepare, a member of class TDBXTraceFlags.

55.1.68.2.13 TDBXTraceFlags.Reader Constant

Pascal

```
const Reader = $0400;
```

C++

```
Reader = 0x0400;
```

Description

This is Reader, a member of class TDBXTraceFlags.

55.1.68.2.14 TDBXTraceFlags.Transact Constant

Pascal

```
const Transact = $0020;
```

C++

```
Transact = 0x0020;
```

Description

This is Transact, a member of class TDBXTraceFlags.

55.1.68.2.15 TDBXTraceFlags.Vendor Constant

Pascal

```
const Vendor = $0100;
```

C++

```
Vendor = 0x0100;
```

Description

This is Vendor, a member of class TDBXTraceFlags.

55.1.69 TDBXTraceInfo Record

Trace information for TDBXTraceEvent (see page 10672) handlers.

Pascal

```
TDBXTraceInfo = packed record
end;
```

C++

```
struct TDBXTraceInfo {
};
```

File

DBXCommon

Description

TDBXTraceInfo is passed in to TDBXTraceEvent (see page 10672) handlers.

Message (see page 10623) is trace information for a TDBXTraceEvent (see page 10672).

TraceFlag (see page 10623) is a set of TDBXTraceFlags (see page 10617) tracing categories for a TDBXTraceEvent (see page 10672).

See Also

TDBXTraceFlags (see page 10617)

TDBXTraceEvent (see page 10672)

55.1.69.1 TDBXTraceInfo Members

The following tables list the members exposed by TDBXTraceInfo.

Public Properties

	Name	Description
	Message (see page 10623)	This is Message, a member of class TDBXTraceInfo.
	TraceFlag (see page 10623)	This is TraceFlag, a member of class TDBXTraceInfo.

55.1.69.2 TDBXTraceInfo Properties

The properties of the TDBXTraceInfo class are listed here.

Public Properties

	Name	Description
	Message (see page 10623)	This is Message, a member of class TDBXTraceInfo.
	TraceFlag (see page 10623)	This is TraceFlag, a member of class TDBXTraceInfo.

55.1.69.2.1 TDBXTraceInfo.Message Property

Pascal

```
property Message: WideString;
```

C++

```
__property BSTR Message;
```

Description

This is Message, a member of class TDBXTraceInfo.

55.1.69.2.2 TDBXTraceInfo.TraceFlag Property

Pascal

```
property TraceFlag: TDBXTraceFlag;
```

C++

```
__property TDBXTraceFlag TraceFlag;
```

Description

This is TraceFlag, a member of class TDBXTraceInfo.

55.1.70 TDBXTransaction Class

Transaction management class.

Class Hierarchy

DBXCommon.TDBXTransaction

Pascal

```
TDBXTransaction = class;
```

C++

```
class TDBXTransaction;
```

File

DBXCommon

Description

A TDBXTransaction is created when you call BeginTransaction. CommitFreeAndNil, RollbackFreeAndNil, or

RollbackIncompleteFreeAndNil must be called to complete a transaction. The TDBXIsolation (see page 10671) transaction isolation level can be set when the transaction is begun. SupportsTransactions must be true for an application to be able to use transactions. Nested transactions can be used if SupportsNestedTransactionsis true.

55.1.70.1 TDBXTransaction Members

The following tables list the members exposed by TDBXTransaction.

Protected Methods

	Name	Description
	Create (see page 10624)	Creates an instance of TDBXTransaction (see page 10623)

Public Methods

	Name	Description
V	Destroy (see page 10624)	Destroy an instance of TDBXTransaction (see page 10623).

Public Properties

	Name	Description
R	Connection (see page 10625)	TDBXConnection (see page 10398) instance that created the TDBXTransaction (see page 10623).

Strict protected Fields

	Name	Description
	FIsolationLevel (see page 10625)	This is FIsolationLevel, a member of class TDBXTransaction.

55.1.70.2 TDBXTransaction.Create Constructor

Creates an instance of TDBXTransaction (see page 10623)

Pascal

```
constructor Create(Connection: TDBXConnection);
```

C++

```
__fastcall TDBXTransaction(TDBXConnection Connection);
```

Description

This constructor creates and initializes an instance of TDBXTransaction (see page 10623).

See Also

Destroy (see page 10624)

55.1.70.3 TDBXTransaction.Destroy Destructor

Destroy an instance of TDBXTransaction (see page 10623).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXTransaction();
```

Description

Use Destroy to destroy an instance of TDBXTransaction (see page 10623).

55.1.70.4 TDBXTransaction Fields

The fields of the TDBXTransaction class are listed here.

Strict protected Fields

	Name	Description
	FIsolationLevel (see page 10625)	This is FIsolationLevel, a member of class TDBXTransaction.

55.1.70.4.1 TDBXTransaction.FIsolationLevel Field

Pascal

```
FIsolationLevel: TDBXIsolation;
```

C++

```
TDBXIsolation FIsolationLevel;
```

Description

This is FIsolationLevel, a member of class TDBXTransaction.

55.1.70.5 TDBXTransaction Properties

The properties of the TDBXTransaction class are listed here.

Public Properties

	Name	Description
R	Connection (see page 10625)	TDBXConnection (see page 10398) instance that created the TDBXTransaction (see page 10623).

55.1.70.5.1 TDBXTransaction.Connection Property

TDBXConnection (see page 10398) instance that created the TDBXTransaction (see page 10623).

Pascal

```
property Connection: TDBXConnection;
```

C++

```
__property TDBXConnection Connection;
```

Description

Connection contains the TDBXConnection (see page 10398) instance that created this TDBXTransaction (see page 10623) instance.

See Also

TDBXConnection (see page 10398)

55.1.71 TDBXValue Class

Gets and sets values for TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.

Class Hierarchy

DBXCommon.TDBXValue

Pascal

```
TDBXValue = class;
```

C++

```
class TDBXValue;
```

File

DBXCommon

Description

TDBXValue provides value support for the classes TDBXReader (see page 10576) and TDBXParameterList (see page 10553). Gets and sets values for Value and Value properties.

55.1.71.1 TDBXValue Members

The following tables list the members exposed by TDBXValue.

Protected Methods

	Name	Description
	Create (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Methods

	Name	Description
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
V	Destroy (see page 10627)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
V	GetAnsiString (see page 10629)	Gets the string value for ANSI strings.
V	GetBcd (see page 10629)	Gets the BCD value for binary-compressed decimal data types.
V	GetBoolean (see page 10629)	Gets the boolean value for boolean data types.
V	GetBytes (see page 10629)	Gets the byte array value for byte array and blob types.
V	GetDate (see page 10630)	Gets the date value for date data types.
V	GetDouble (see page 10630)	Gets the double floating point value for double data types.
V	GetInt16 (see page 10631)	Gets the smallInt value for 16-byte integer data types.
V	GetInt32 (see page 10631)	Gets the 32-byte integer value for 32-byte integer data types.
V	GetInt64 (see page 10631)	Gets the 64-byte integer value for 64-byte integer data types.
V	GetTime (see page 10631)	Gets the time value for time data types.
V	GetTimeStamp (see page 10632)	Gets the timestamp value for timestamp data types.

	GetValueSize (see page 10632)	Gets the size in bytes for values.
	GetWideString (see page 10632)	Gets the wide string value for wide string data types.
	IsNull (see page 10632)	Returns true if the value is **null**.

Public Properties

	Name	Description
	ValueType (see page 10633)	Gets the value type for this value.

55.1.71.2 TDBXValue.Create Constructor

Enter a short sentence, starting with a verb, that states the basic purpose of the property.

Pascal

```
constructor Create(ValueType: TDBXValueType);
```

C++

```
__fastcall TDBXValue(TDBXValueType ValueType);
```

Description

The long description elaborates on the short description, containing no more than a paragraph or two.

See Also

Chapter Title

Procedure Title

Example Title

ClassName

PropertyName

MethodName

EventName

RoutineName

TypeName

VariableName

ConstantName

55.1.71.3 TDBXValue.Destroy Destructor

Enter a short sentence, starting with a verb, that states the basic purpose of the property.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXValue();
```

Description

The long description elaborates on the short description, containing no more than a paragraph or two.

See Also

Chapter Title

Procedure Title

Example Title

ClassName

PropertyName

MethodName

EventName

RoutineName

TypeName

VariableName

ConstantName

55.1.71.4 TDBXValue Methods

The methods of the TDBXValue class are listed here.

Protected Methods

	Name	Description
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.

Public Methods

	Name	Description
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
	GetAnsiString (see page 10629)	Gets the string value for ANSI strings.
	GetBcd (see page 10629)	Gets the BCD value for binary-compressed decimal data types.
	GetBoolean (see page 10629)	Gets the boolean value for boolean data types.
	GetBytes (see page 10629)	Gets the byte array value for byte array and blob types.
	GetDate (see page 10630)	Gets the date value for date data types.
	GetDouble (see page 10630)	Gets the double floating point value for double data types.
	GetInt16 (see page 10631)	Gets the smallInt value for 16-byte integer data types.
	GetInt32 (see page 10631)	Gets the 32-byte integer value for 32-byte integer data types.
	GetInt64 (see page 10631)	Gets the 64-byte integer value for 64-byte integer data types.
	GetTime (see page 10631)	Gets the time value for time data types.
	GetTimeStamp (see page 10632)	Gets the timestamp value for timestamp data types.
	GetValueSize (see page 10632)	Gets the size in bytes for values.
	GetWideString (see page 10632)	Gets the wide string value for wide string data types.
	IsNull (see page 10632)	Returns true if the value is **null**.

55.1.71.4.1 TDBXValue.CreateValue Method

Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.

Pascal

```
class function CreateValue(DBXContext: TDBXContext; ValueType: TDBXValueType; DbxRow: TDBXRow;
ReadOnlyType: Boolean): TDBXValue;
```

C++

```
__fastcall TDBXValue CreateValue(TDBXContext DBXContext, TDBXValueType ValueType, TDBXRow
DbxRow, Boolean ReadOnlyType);
```

Description

Creates values for Value and Value properties.

55.1.71.4.2 TDBXValue.GetAnsiString Method

Gets the string value for ANSI strings.

Pascal

```
function GetAnsiString: String; virtual;
```

C++

```
virtual __fastcall AnsiString GetAnsiString();
```

Description

Gets the string value for TDBXValue (see page 10626) instances with a data type of AnsiStringType. Throws a TDBXError (see page 10486) exception if IsNull (see page 10632) is true.

55.1.71.4.3 TDBXValue.GetBcd Method

Gets the BCD value for binary-compressed decimal data types.

Pascal

```
function GetBcd: TBcd; virtual;
```

C++

```
virtual __fastcall TBcd GetBcd();
```

Description

Gets the TBcd value for TDBXValue (see page 10626) instances with a with a data type of BcdType. Throws a TDBXError (see page 10486) exception if IsNull (see page 10632) is true.

55.1.71.4.4 TDBXValue.GetBoolean Method

Gets the boolean value for boolean data types.

Pascal

```
function GetBoolean: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetBoolean();
```

Description

Gets the boolean value for TDBXValue (see page 10626) instances with a data types of BooleanType. Throws a TDBXError (see page 10486) exception if IsNull (see page 10632) is true.

55.1.71.4.5 TDBXValue.GetBytes Method

Gets the byte array value for byte array and blob types.

Pascal

```
function GetBytes(Offset: Int64; const Buffer: TBytes; BufferOffset: Int64; Length: Int64):
Int64; virtual; overload;
```

C++

```
virtual __fastcall Int64 GetBytes(Int64 Offset, const TBytes Buffer, Int64 BufferOffset, Int64
Length);
```

Description

Gets the byte array value for TDBXValue (see page 10626) instances with a data type of BytesType and .BlobType. Throws a TDBXError (see page 10486) exception if IsNull (see page 10632) is true.

55.1.71.4.6 TDBXValue.GetDate Method

Gets the date value for date data types.

Pascal

```
function GetDate: TDBXDate; virtual;
```

C++

```
virtual __fastcall TDBXDate GetDate();
```

Description

Gets the date value for TDBXValue (see page 10626) instances with a data type of DateType. Throws a TDBXError (see page 10486) exception if IsNull (see page 10632) is true.

55.1.71.4.7 TDBXValue.GetDbxContext Method

Get context information for a connection associated with value.

Pascal

```
function GetDbxContext: TDBXContext;
```

C++

```
__fastcall TDBXContext GetDbxContext();
```

Description

GetDbxContext gets the value of the DbxContext (see page 10633) property, which holds the TDBXContext (see page 10432) associated with the value. This is used by driver writers.

See Also

TDBXContext (see page 10432)

DbxContext (see page 10633)

55.1.71.4.8 TDBXValue.GetDouble Method

Gets the double floating point value for double data types.

Pascal

```
function GetDouble: Double; virtual;
```

C++

```
virtual __fastcall Double GetDouble();
```

Description

Gets the double floating point value for TDBXValue (see page 10626) instances with a data type of DoubleType. Throws a TDBXError (see page 10486) exception if IsNull (see page 10632) is true.

55.1.71.4.9 TDBXValue.GetInt16 Method

Gets the smallInt value for 16-byte integer data types.

Pascal

```
function GetInt16: SmallInt; virtual;
```

C++

```
virtual __fastcall SmallInt GetInt16();
```

Description

Gets the smallInt value for TDBXValue (see page 10626) instances with a data type of Int16Type. Throws a TDBXError (see page 10486) exception if IsNull (see page 10632) is true.

55.1.71.4.10 TDBXValue.GetInt32 Method

Gets the 32-byte integer value for 32-byte integer data types.

Pascal

```
function GetInt32: TInt32; virtual;
```

C++

```
virtual __fastcall TInt32 GetInt32();
```

Description

Gets the 32-byte integer value for TDBXValue (see page 10626) instances with a data type of Int32Type. Throws a TDBXError (see page 10486) exception if IsNull (see page 10632) is true.

55.1.71.4.11 TDBXValue.GetInt64 Method

Gets the 64-byte integer value for 64-byte integer data types.

Pascal

```
function GetInt64: Int64; virtual;
```

C++

```
virtual __fastcall Int64 GetInt64();
```

Description

Gets the 64-byte integer value for TDBXValue (see page 10626) instances with a data type of Int64Type. Throws a TDBXError (see page 10486) exception if IsNull (see page 10632) is true.

55.1.71.4.12 TDBXValue.GetTime Method

Gets the time value for time data types.

Pascal

```
function GetTime: TDBXTime; virtual;
```

C++

```
virtual __fastcall TDBXTime GetTime();
```

Description

Gets the time value for TDBXValue (see page 10626) instances with a data type of TimeType. Throws a TDBXError (see page 10486) exception if IsNull (see page 10632) is true.

55.1.71.4.13 TDBXValue.GetTimeStamp Method

Gets the timestamp value for timestamp data types.

Pascal

```
function GetTimeStamp: TSQLTimeStamp; virtual;
```

C++

```
virtual __fastcall TSQLTimeStamp GetTimeStamp();
```

Description

Gets the timestamp value for TDBXValue (see page 10626) instances with a data types of TimeStampType. Throws a TDBXError (see page 10486) exception if IsNull (see page 10632) is true.

55.1.71.4.14 TDBXValue.GetValueSize Method

Gets the size in bytes for values.

Pascal

```
function GetValueSize: Int64; virtual;
```

C++

```
virtual __fastcall Int64 GetValueSize();
```

Description

Gets the size in bytes for a TDBXValue (see page 10626). For string data types the number of characters is returned. Throws a TDBXError (see page 10486) exception if IsNull (see page 10632) is true.

55.1.71.4.15 TDBXValue.GetWideString Method

Gets the wide string value for wide string data types.

Pascal

```
function GetWideString: WideString; virtual;
```

C++

```
virtual __fastcall BSTR GetWideString();
```

Description

Gets the wide string value for TDBXValue (see page 10626) instances with a data type of WideString. Throws a TDBXError (see page 10486) exception if IsNull (see page 10632) is true.

55.1.71.4.16 TDBXValue.IsNull Method

Returns true if the value is **null**.

Pascal

```
function IsNull: Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean IsNull() = 0;
```

Description

Returns true if the value is **null**. This method should be called before calling any of the Get methods. If the value is null, the Get methods will throw TDBXError (see page 10486) exception if they are called.

55.1.71.5 TDBXValue Properties

The properties of the TDBXValue class are listed here.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Properties

	Name	Description
R	ValueType (see page 10633)	Gets the value type for this value.

55.1.71.5.1 TDBXValue.DbxContext Property

Context information for a connection associated with value.

Pascal

```
property DbxContext: TDBXContext;
```

C++

```
__property TDBXContext DbxContext;
```

Description

DbxContext holds the TDBXContext (see page 10432) associated with the value. This is used by driver writers.

See Also

TDBXContext (see page 10432)

GetDbxContext (see page 10630)

55.1.71.5.2 TDBXValue.ValueType Property

Gets the value type for this value.

Pascal

```
property ValueType: TDBXValueType;
```

C++

```
__property TDBXValueType ValueType;
```

Description

Gets the TDBXValueType (see page 10636) for this TDBXValue (see page 10626).

55.1.72 TDBXValueList Class

Manage a list of TDBXValueType (see page 10636) instances.

Class Hierarchy

DBXCommon.TDBXValueList

Pascal

```
TDBXValueList = class;
```

C++

```
class TDBXValueList;
```

File

DBXCommon

Description

This class is used internally by TDBXParameterList (see page 10553) to manage a list of TDBXValueType (see page 10636) instances.

See Also

TDBXParameterList (see page 10553)

TDBXValueType (see page 10636)

55.1.72.1 TDBXValueList Members

The following tables list the members exposed by TDBXValueList.

Public Methods

	Name	Description
	Destroy (see page 10634)	Destroys a TDBXValueList (see page 10634) instance.

Strict private Fields

	Name	Description
	FLastOrdinal (see page 10635)	This is FLastOrdinal, a member of class TDBXValueList.
	FValueTypeArray (see page 10635)	This is FValueTypeArray, a member of class TDBXValueList.

Strict private Methods

	Name	Description
	FindOrdinal (see page 10635)	This is FindOrdinal, a member of class TDBXValueList.

55.1.72.2 TDBXValueList.Destroy Destructor

Destroys a TDBXValueList (see page 10634) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXValueList();
```

Description

This destructor destroys an instance of TDBXValueList (see page 10634).

55.1.72.3 TDBXValueList Fields

The fields of the TDBXValueList class are listed here.

Strict private Fields

	Name	Description
	FLastOrdinal (see page 10635)	This is FLastOrdinal, a member of class TDBXValueList.
	FValueTypeArray (see page 10635)	This is FValueTypeArray, a member of class TDBXValueList.

55.1.72.3.1 TDBXValueList.FLastOrdinal Field

Pascal

```
FLastOrdinal: Integer;
```

C++

```
int FLastOrdinal;
```

Description

This is FLastOrdinal, a member of class TDBXValueList.

55.1.72.3.2 TDBXValueList.FValueTypeArray Field

Pascal

```
FValueTypeArray: array of TDBXValueType;
```

C++

```
array of TDBXValueType FValueTypeArray;
```

Description

This is FValueTypeArray, a member of class TDBXValueList.

55.1.72.4 TDBXValueList Methods

The methods of the TDBXValueList class are listed here.

Strict private Methods

	Name	Description
	FindOrdinal (see page 10635)	This is FindOrdinal, a member of class TDBXValueList.

55.1.72.4.1 TDBXValueList.FindOrdinal Method

Pascal

```
function FindOrdinal(const Name: WideString; StartOrdinal: TInt32; EndOrdinal: TInt32):
TInt32; virtual;
```

C++

```
virtual __fastcall TInt32 FindOrdinal(const BSTR Name, TInt32 StartOrdinal, TInt32 EndOrdinal);
```

Description

This is FindOrdinal, a member of class TDBXValueList.

55.1.73 TDBXValueType Class

Metadata for TDBXReader (see page 10576) columns and TDBXParameter (see page 10545) parameters.

Class Hierarchy

DBXCommon.TDBXValueType

Pascal

```
TDBXValueType = class;
```

C++

```
class TDBXValueType;
```

File

DBXCommon

Description

TDBXValueType properties describe the types expected or returned by queries and stored commands.

55.1.73.1 TDBXValueType Members

The following tables list the members exposed by TDBXValueType.

Protected Methods

	Name	Description
	Create (see page 10637)	Creates an instance of TDBXValueType (see page 10636)
	GetChildPosition (see page 10638)	Get ChildPosition (see page 10645) property.
	GetDataType (see page 10639)	Get DataType (see page 10646) property.
	GetFlags (see page 10639)	Get ValueTypeFlags (see page 10648) property.
	GetName (see page 10639)	Get Name (see page 10646) property.
	GetOrdinal (see page 10639)	Get Ordinal (see page 10646) property.
	GetParameterDirection (see page 10640)	Get ParameterDirection (see page 10647) property.
	GetPrecision (see page 10640)	Get Precision (see page 10647) property.
	GetScale (see page 10640)	Get Scale (see page 10647) property.
	GetSize (see page 10641)	Get Size (see page 10648) property.
	GetSubType (see page 10641)	Get SubType (see page 10648) property.
	SetChildPosition (see page 10641)	Set ChildPosition (see page 10645) property.
	SetDataType (see page 10642)	Set DataType (see page 10646) property.
	SetFlags (see page 10642)	Set ValueTypeFlags (see page 10648) property.
	SetName (see page 10642)	Set Name (see page 10646) property.
	SetNullable (see page 10642)	Set ChildPosition (see page 10645) property.
	SetOrdinal (see page 10643)	Set Ordinal (see page 10646) property.
	SetParameterDirection (see page 10643)	Set ParameterDirection (see page 10647) property.
	SetPrecision (see page 10643)	Set Precision (see page 10647) property.

V	SetScale (see page 10644)	Set Scale (see page 10647) property.
V	SetSize (see page 10644)	Set Size (see page 10648) property.
V	SetSubType (see page 10644)	Set SubType (see page 10648) property.

Public Properties

	Name	Description
R	AutoIncrement (see page 10645)	Gets the value that determines whether or not this describes an autoincrement value.
	ChildPosition (see page 10645)	Used by TDBXParameter (see page 10545) instances. Child position of Oracle ADT and Array types
	DataType (see page 10646)	Value data type.
	Name (see page 10646)	Name for this value.
	Nullable (see page 10646)	Determines whether or not a parameter can be set to null.
	Ordinal (see page 10646)	Ordinal position of this value.
	ParameterDirection (see page 10647)	Gets or sets the direction of a parameter.
	Precision (see page 10647)	The maximum number of digits allowed in the parameter.
R	ReadOnly (see page 10647)	True if this TDBXValueType (see page 10636) is read-only.
	Scale (see page 10647)	The maximum number of digits used to represent the property.
R	Searchable (see page 10648)	True if this TDBXValueType (see page 10636) is searchable.
	Size (see page 10648)	The size of the parameter value in bytes.
	SubType (see page 10648)	Data sub-type.
	ValueTypeFlags (see page 10648)	TDBXValue (see page 10626) metadata for TDBXReader (see page 10576).

Strict protected Fields

	Name	Description
	FDBXContext (see page 10637)	This is FDBXContext, a member of class TDBXValueType.

55.1.73.2 TDBXValueType.Create Constructor

Creates an instance of TDBXValueType (see page 10636)

Pascal

```
constructor Create(DBXContext: TDBXContext);
```

C++

```
__fastcall TDBXValueType(TDBXContext DBXContext);
```

Description

This constructor creates and initializes an instance of TDBXValueType (see page 10636).

55.1.73.3 TDBXValueType Fields

The fields of the TDBXValueType class are listed here.

Strict protected Fields

	Name	Description
	FDBXContext (see page 10637)	This is FDBXContext, a member of class TDBXValueType.

55.1.73.3.1 TDBXValueType.FDBXContext Field

Pascal

```
FDBXContext: TDBXContext;
```

C++

```
TDBXContext FDBXContext;
```

Description

This is FDBXContext, a member of class TDBXValueType.

55.1.73.4 TDBXValueType Methods

The methods of the TDBXValueType class are listed here.

Protected Methods

	Name	Description
	GetChildPosition (see page 10638)	Get ChildPosition (see page 10645) property.
	GetDataType (see page 10639)	Get DataType (see page 10646) property.
	GetFlags (see page 10639)	Get ValueTypeFlags (see page 10648) property.
	GetName (see page 10639)	Get Name (see page 10646) property.
	GetOrdinal (see page 10639)	Get Ordinal (see page 10646) property.
	GetParameterDirection (see page 10640)	Get ParameterDirection (see page 10647) property.
	GetPrecision (see page 10640)	Get Precision (see page 10647) property.
	GetScale (see page 10640)	Get Scale (see page 10647) property.
	GetSize (see page 10641)	Get Size (see page 10648) property.
	GetSubType (see page 10641)	Get SubType (see page 10648) property.
	SetChildPosition (see page 10641)	Set ChildPosition (see page 10645) property.
	SetDataType (see page 10642)	Set DataType (see page 10646) property.
	SetFlags (see page 10642)	Set ValueTypeFlags (see page 10648) property.
	SetName (see page 10642)	Set Name (see page 10646) property.
	SetNullable (see page 10642)	Set ChildPosition (see page 10645) property.
	SetOrdinal (see page 10643)	Set Ordinal (see page 10646) property.
	SetParameterDirection (see page 10643)	Set ParameterDirection (see page 10647) property.
	SetPrecision (see page 10643)	Set Precision (see page 10647) property.
	SetScale (see page 10644)	Set Scale (see page 10647) property.
	SetSize (see page 10644)	Set Size (see page 10648) property.
	SetSubType (see page 10644)	Set SubType (see page 10648) property.

55.1.73.4.1 TDBXValueType.GetChildPosition Method

Get ChildPosition (see page 10645) property.

Pascal

```
function GetChildPosition: TInt32; virtual;
```

C++

```
virtual __fastcall TInt32 GetChildPosition();
```

Description

GetChildPosition gets the ChildPosition (see page 10645) property.

See Also

ChildPosition (see page 10645)

55.1.73.4.2 TDBXValueType.GetDataType Method

Get DataType (see page 10646) property.

Pascal

```
function GetDataType: TDBXType; virtual;
```

C++

```
virtual __fastcall TDBXType GetDataType();
```

Description

GetDataType gets the DataType (see page 10646) property.

See Also

DataType (see page 10646)

55.1.73.4.3 TDBXValueType.GetFlags Method

Get ValueTypeFlags (see page 10648) property.

Pascal

```
function GetFlags: TInt32; virtual;
```

C++

```
virtual __fastcall TInt32 GetFlags();
```

Description

GetFlags gets the ValueTypeFlags (see page 10648) property.

See Also

ValueTypeFlags (see page 10648)

55.1.73.4.4 TDBXValueType.GetName Method

Get Name (see page 10646) property.

Pascal

```
function GetName: WideString; virtual;
```

C++

```
virtual __fastcall BSTR GetName();
```

Description

GetName gets the Name (see page 10646) property.

See Also

Name (see page 10646)

55.1.73.4.5 TDBXValueType.GetOrdinal Method

Get Ordinal (see page 10646) property.

Pascal

```
function GetOrdinal: TInt32; virtual;
```

C++

```
virtual __fastcall TInt32 GetOrdinal();
```

Description

GetOrdinal gets the Ordinal (see page 10646) property.

See Also

Ordinal (see page 10646)

55.1.73.4.6 TDBXValueType.GetParameterDirection Method

Get ParameterDirection (see page 10647) property.

Pascal

```
function GetParameterDirection: TDBXParameterDirection; virtual;
```

C++

```
virtual __fastcall TDBXParameterDirection GetParameterDirection();
```

Description

GetParameterDirection gets the ParameterDirection (see page 10647) property.

See Also

ParameterDirection (see page 10647)

55.1.73.4.7 TDBXValueType.GetPrecision Method

Get Precision (see page 10647) property.

Pascal

```
function GetPrecision: Int64; virtual;
```

C++

```
virtual __fastcall Int64 GetPrecision();
```

Description

GetPrecision gets the Precision (see page 10647) property.

See Also

Precision (see page 10647)

55.1.73.4.8 TDBXValueType.GetScale Method

Get Scale (see page 10647) property.

Pascal

```
function GetScale: TInt32; virtual;
```

C++

```
virtual __fastcall TInt32 GetScale();
```

Description

GetScale gets the Scale (see page 10647) property.

See Also

Scale (see page 10647)

55.1.73.4.9 TDBXValueType.GetSize Method

Get Size (see page 10648) property.

Pascal

```
function GetSize: Int64; virtual;
```

C++

```
virtual __fastcall Int64 GetSize();
```

Description

GetSize gets the Size (see page 10648) property.

See Also

Size (see page 10648)

55.1.73.4.10 TDBXValueType.GetSubType Method

Get SubType (see page 10648) property.

Pascal

```
function GetSubType: TDBXType; virtual;
```

C++

```
virtual __fastcall TDBXType GetSubType();
```

Description

GetSubType gets the SubType (see page 10648) property.

See Also

SubType (see page 10648)

55.1.73.4.11 TDBXValueType.SetChildPosition Method

Set ChildPosition (see page 10645) property.

Pascal

```
procedure SetChildPosition(ChildPosition: TInt32); virtual;
```

C++

```
virtual __fastcall SetChildPosition(TInt32 ChildPosition);
```

Description

SetChildPosition sets the ChildPosition (see page 10645) property.

See Also

ChildPosition (see page 10645)

55.1.73.4.12 TDBXValueType.SetDataType Method

Set DataType (see page 10646) property.

Pascal

```
procedure SetDataType(DataType: TDBXType); virtual;
```

C++

```
virtual __fastcall SetDataType(TDBXType DataType);
```

Description

SetDataType sets the DataType (see page 10646) property.

See Also

DataType (see page 10646)

55.1.73.4.13 TDBXValueType.SetFlags Method

Set ValueTypeFlags (see page 10648) property.

Pascal

```
procedure SetFlags(Flags: TInt32); virtual;
```

C++

```
virtual __fastcall SetFlags(TInt32 Flags);
```

Description

SetFlags sets theValueTypeFlags property.

See Also

ValueTypeFlags (see page 10648)

55.1.73.4.14 TDBXValueType.SetName Method

Set Name (see page 10646) property.

Pascal

```
procedure SetName(const Name: WideString); virtual;
```

C++

```
virtual __fastcall SetName(const BSTR Name);
```

Description

SetName sets the Name (see page 10646) property.

See Also

Name (see page 10646)

55.1.73.4.15 TDBXValueType.SetNullable Method

Set ChildPosition (see page 10645) property.

Pascal

```
procedure SetNullable(NullableValue: Boolean); virtual;
```

C++

```
virtual __fastcall SetNullable(Boolean NullableValue);
```

Description

SetChildPosition (see page 10641) sets the ChildPosition (see page 10645) property.

See Also

ChildPosition (see page 10645)

55.1.73.4.16 TDBXValueType.SetOrdinal Method

Set Ordinal (see page 10646) property.

Pascal

```
procedure SetOrdinal(Ordinal: TInt32); virtual;
```

C++

```
virtual __fastcall SetOrdinal(TInt32 Ordinal);
```

Description

SetOrdinal sets the Ordinal (see page 10646) property.

See Also

Ordinal (see page 10646)

55.1.73.4.17 TDBXValueType.SetParameterDirection Method

Set ParameterDirection (see page 10647) property.

Pascal

```
procedure SetParameterDirection(ParameterDirection: TDBXParameterDirection); virtual;
```

C++

```
virtual __fastcall SetParameterDirection(TDBXParameterDirection ParameterDirection);
```

Description

SetParameterDirection sets the ParameterDirection (see page 10647) property.

See Also

ParameterDirection (see page 10647)

55.1.73.4.18 TDBXValueType.SetPrecision Method

Set Precision (see page 10647) property.

Pascal

```
procedure SetPrecision(Precision: Int64); virtual;
```

C++

```
virtual __fastcall SetPrecision(Int64 Precision);
```

Description

SetPrecision sets the Precision (see page 10647) property.

See Also

Precision (see page 10647)

55.1.73.4.19 TDBXValueType.SetScale Method

Set Scale (see page 10647) property.

Pascal

```
procedure SetScale(Scale: TInt32); virtual;
```

C++

```
virtual __fastcall SetScale(TInt32 Scale);
```

Description

SetScale sets the Scale (see page 10647) property.

See Also

Scale (see page 10647)

55.1.73.4.20 TDBXValueType.SetSize Method

Set Size (see page 10648) property.

Pascal

```
procedure SetSize(Size: Int64); virtual;
```

C++

```
virtual __fastcall SetSize(Int64 Size);
```

Description

SetSize sets the Size (see page 10648) property.

See Also

Size (see page 10648)

55.1.73.4.21 TDBXValueType.SetSubType Method

Set SubType (see page 10648) property.

Pascal

```
procedure SetSubType(SubType: TDBXType); virtual;
```

C++

```
virtual __fastcall SetSubType(TDBXType SubType);
```

Description

SetSubType sets the SubType (see page 10648) property.

See Also

SubType (see page 10648)

55.1.73.5 TDBXValueType Properties

The properties of the TDBXValueType class are listed here.

Public Properties

	Name	Description
R	AutoIncrement (see page 10645)	Gets the value that determines whether or not this describes an autoincrement value.
	ChildPosition (see page 10645)	Used by TDBXParameter (see page 10545) instances. Child position of Oracle ADT and Array types
	DataType (see page 10646)	Value data type.
	Name (see page 10646)	Name for this value.
	Nullable (see page 10646)	Determines whether or not a parameter can be set to null.
	Ordinal (see page 10646)	Ordinal position of this value.
	ParameterDirection (see page 10647)	Gets or sets the direction of a parameter.
	Precision (see page 10647)	The maximum number of digits allowed in the parameter.
R	ReadOnly (see page 10647)	True if this TDBXValueType (see page 10636) is read-only.
	Scale (see page 10647)	The maximum number of digits used to represent the property.
R	Searchable (see page 10648)	True if this TDBXValueType (see page 10636) is searchable.
	Size (see page 10648)	The size of the parameter value in bytes.
	SubType (see page 10648)	Data sub-type.
	ValueTypeFlags (see page 10648)	TDBXValue (see page 10626) metadata for TDBXReader (see page 10576).

55.1.73.5.1 TDBXValueType.AutoIncrement Property

Gets the value that determines whether or not this describes an autoincrement value.

Pascal

```
property AutoIncrement: Boolean;
```

C++

```
__property Boolean AutoIncrement;
```

Description

Use AutoIncrement to get the value that determines whether or not this TDBXValueType (see page 10636) instance describes an autoincrement value.

55.1.73.5.2 TDBXValueType.ChildPosition Property

Used by TDBXParameter (see page 10545) instances. Child position of Oracle ADT and Array types

Pascal

```
property ChildPosition: TInt32;
```

C++

```
__property TInt32 ChildPosition;
```

Description

Use ChildPosition through TDBXParameter (see page 10545) instances to get or set the child position of Oracle ADT and Array types

55.1.73.5.3 TDBXValueType.DataType Property

Value data type.

Pascal

```
property DataType: TDBXType;
```

C++

```
__property TDBXType DataType;
```

Description

Returns one of the const values from TDBXDataTypes (see page 10447).

55.1.73.5.4 TDBXValueType.Name Property

Name for this value.

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

Name for this value. This value can be used as an index Value and Parameter array properties.

55.1.73.5.5 TDBXValueType.Nullable Property

Determines whether or not a parameter can be set to null.

Pascal

```
property Nullable: Boolean;
```

C++

```
__property Boolean Nullable;
```

Description

Gets or sets the value that determines whether or not a parameter can be set to null.

55.1.73.5.6 TDBXValueType.Ordinal Property

Ordinal position of this value.

Pascal

```
property Ordinal: TInt32;
```

C++

```
__property TInt32 Ordinal;
```

Description

Returns the ordinal position of this value for a TDBXReader (see page 10576) or TDBXParameterList (see page 10553).

55.1.73.5.7 TDBXValueType.ParameterDirection Property

Gets or sets the direction of a parameter.

Pascal

```
property ParameterDirection: TDBXParameterDirection;
```

C++

```
__property TDBXParameterDirection ParameterDirection;
```

Description

Use ParameterDirection to get or set the direction of a parameter. This parameter can be any of the TDBXParameterDirections (see page 10549) const values.

55.1.73.5.8 TDBXValueType.Precision Property

The maximum number of digits allowed in the parameter.

Pascal

```
property Precision: Int64;
```

C++

```
__property Int64 Precision;
```

Description

The maximum number of digits allowed in the parameter, including both significant and fractional.

55.1.73.5.9 TDBXValueType.ReadOnly Property

True if this TDBXValueType (see page 10636) is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

If ReadOnly is true, this TDBXValueType (see page 10636) parameter is read-only; if false, it can be read and set.

55.1.73.5.10 TDBXValueType.Scale Property

The maximum number of digits used to represent the property.

Pascal

```
property Scale: TInt32;
```

C++

```
__property TInt32 Scale;
```

Description

Gets or sets the maximum number of digits used to represent the property.

55.1.73.5.11 TDBXValueType.Searchable Property

True if this TDBXValueType (see page 10636) is searchable.

Pascal

```
property Searchable: Boolean;
```

C++

```
__property Boolean Searchable;
```

Description

Gets the value that indicates whether or not this TDBXValueType (see page 10636) parameter is searchable.

55.1.73.5.12 TDBXValueType.Size Property

The size of the parameter value in bytes.

Pascal

```
property Size: Int64;
```

C++

```
__property Int64 Size;
```

Description

Gets or sets the size of the parameter value in bytes. For string data, Size gets or sets the size of the parameter in characters.

55.1.73.5.13 TDBXValueType.SubType Property

Data sub-type.

Pascal

```
property SubType: TInt32;
```

C++

```
__property TInt32 SubType;
```

Description

Returns one of the "sub-type" const values from TDBXDataTypes (see page 10447).

55.1.73.5.14 TDBXValueType.ValueTypeFlags Property

TDBXValue (see page 10626) metadata for TDBXReader (see page 10576).

Pascal

```
property ValueTypeFlags: TInt32;
```

C++

```
__property TInt32 ValueTypeFlags;
```

Description

Used by drivers to set TDBXValue (see page 10626) metadata for TDBXReader (see page 10576).

55.1.74 TDBXValueTypeEx Class

Extension of TDBXValueType (see page 10636).

Class Hierarchy

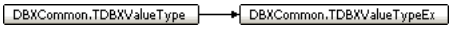

Pascal

```
TDBXValueTypeEx = class(TDBXValueType);
```

C++

```
class TDBXValueTypeEx : public TDBXValueType;
```

File

DBXCommon

Description

TDBXValueTypeEx is an extension of TDBXValueType (see page 10636).

Note: Classes (see page 2685) ending in “Ex” are temporary classes that exist to avoid breaking interfaces. Their contents will be migrated to their base class in the next release.

See Also

TDBXValueType (see page 10636)

55.1.74.1 TDBXValueTypeEx Members

The following tables list the members exposed by TDBXValueTypeEx.

Protected Methods

	Name	Description
	Create (see page 10637)	Creates an instance of TDBXValueType (see page 10636)
	GetChildPosition (see page 10638)	Get ChildPosition (see page 10645) property.
	GetDataType (see page 10639)	Get DataType (see page 10646) property.
	GetFlags (see page 10639)	Get ValueTypeFlags (see page 10648) property.
	GetName (see page 10639)	Get Name (see page 10646) property.
	GetOrdinal (see page 10639)	Get Ordinal (see page 10646) property.
	GetParameterDirection (see page 10640)	Get ParameterDirection (see page 10647) property.
	GetPrecision (see page 10640)	Get Precision (see page 10647) property.
	GetScale (see page 10640)	Get Scale (see page 10647) property.
	GetSize (see page 10641)	Get Size (see page 10648) property.
	GetSubType (see page 10641)	Get SubType (see page 10648) property.
	SetChildPosition (see page 10641)	Set ChildPosition (see page 10645) property.
	SetDataType (see page 10642)	Set DataType (see page 10646) property.
	SetFlags (see page 10642)	Set ValueTypeFlags (see page 10648) property.
	SetName (see page 10642)	Set Name (see page 10646) property.
	SetNullable (see page 10642)	Set ChildPosition (see page 10645) property.
	SetOrdinal (see page 10643)	Set Ordinal (see page 10646) property.
	SetParameterDirection (see page 10643)	Set ParameterDirection (see page 10647) property.
	SetPrecision (see page 10643)	Set Precision (see page 10647) property.

V	SetScale (see page 10644)	Set Scale (see page 10647) property.
V	SetSize (see page 10644)	Set Size (see page 10648) property.
V	SetSubType (see page 10644)	Set SubType (see page 10648) property.

Public Methods

	Name	Description
	DataTypeName (see page 10650)	Gets WideString for TDBXType (see page 10673).
	InvalidTypeAccess (see page 10651)	Raises **InvalidType** TDBXError (see page 10486) exception.

Public Properties

	Name	Description
R	AutoIncrement (see page 10645)	Gets the value that determines whether or not this describes an autoincrement value.
	ChildPosition (see page 10645)	Used by TDBXParameter (see page 10545) instances. Child position of Oracle ADT and Array types
	DataType (see page 10646)	Value data type.
	Name (see page 10646)	Name for this value.
	Nullable (see page 10646)	Determines whether or not a parameter can be set to null.
	Ordinal (see page 10646)	Ordinal position of this value.
	ParameterDirection (see page 10647)	Gets or sets the direction of a parameter.
	Precision (see page 10647)	The maximum number of digits allowed in the parameter.
R	ReadOnly (see page 10647)	True if this TDBXValueType (see page 10636) is read-only.
	Scale (see page 10647)	The maximum number of digits used to represent the property.
R	Searchable (see page 10648)	True if this TDBXValueType (see page 10636) is searchable.
	Size (see page 10648)	The size of the parameter value in bytes.
	SubType (see page 10648)	Data sub-type.
	ValueTypeFlags (see page 10648)	TDBXValue (see page 10626) metadata for TDBXReader (see page 10576).

Strict protected Fields

	Name	Description
	FDBXContext (see page 10637)	This is FDBXContext, a member of class TDBXValueType.

55.1.74.2 TDBXValueTypeEx Methods

The methods of the TDBXValueTypeEx class are listed here.

Public Methods

	Name	Description
	DataTypeName (see page 10650)	Gets WideString for TDBXType (see page 10673).
	InvalidTypeAccess (see page 10651)	Raises **InvalidType** TDBXError (see page 10486) exception.

55.1.74.2.1 TDBXValueTypeEx.DataTypeName Method

Gets WideString for TDBXType (see page 10673).

Pascal

```
class function DataTypeName(DataType: TDBXType): WideString; static;
```

C++

```
static __fastcall BSTR DataTypeName(TDBXType DataType);
```

Description

DataTypeName gets a WideString value for a TDBXType (see page 10673).

See Also

TDBXDataTypes (see page 10447)

TDBXType (see page 10673)

55.1.74.2.2 TDBXValueTypeEx.InvalidTypeAccess Method

Raises **InvalidType** TDBXError (see page 10486) exception.

Pascal

```
class procedure InvalidTypeAccess(ExpectedDataType: TDBXType; DataType: TDBXType); static;
```

C++

```
static __fastcall InvalidTypeAccess(TDBXType ExpectedDataType, TDBXType DataType);
```

Description

InvalidTypeAccess raises an **InvalidType** TDBXError (see page 10486) exception.

See Also

TDBXError (see page 10486)

55.1.75 TDBXValueTypeFlags Class

Flags used by TDBXValueType (see page 10636).

Class Hierarchy

Pascal

```
TDBXValueTypeFlags = class;
```

C++

```
class TDBXValueTypeFlags;
```

File

DBXCommon

Description

These flags provide information about a TDBXValueType (see page 10636) instance. The TDBXValueType.GetFlags method returns a bit mask of these flags logically ORed together.

This table shows the flag constants.

Constant	Description
Nullable (see page 10652)	Value is nullable
ReadOnly (see page 10653)	Value is read only
Searchable (see page 10653)	Value is searchable
AutoIncrement (see page 10652)	Value is autoincrement
ReadOnlyType (see page 10653)	Value is read only

See Also

TDBXValueType (see page 10636)

GetFlags (see page 10639)

55.1.75.1 TDBXValueTypeFlags Members

The following tables list the members exposed by TDBXValueTypeFlags.

Public Constants

	Name	Description
	AutoIncrement (see page 10652)	This is AutoIncrement, a member of class TDBXValueTypeFlags.
	Nullable (see page 10652)	This is Nullable, a member of class TDBXValueTypeFlags.
	ReadOnly (see page 10653)	This is ReadOnly, a member of class TDBXValueTypeFlags.
	ReadOnlyType (see page 10653)	This is ReadOnlyType, a member of class TDBXValueTypeFlags.
	Searchable (see page 10653)	This is Searchable, a member of class TDBXValueTypeFlags.

55.1.75.2 TDBXValueTypeFlags Constants

The constants of the TDBXValueTypeFlags class are listed here.

Public Constants

	Name	Description
	AutoIncrement (see page 10652)	This is AutoIncrement, a member of class TDBXValueTypeFlags.
	Nullable (see page 10652)	This is Nullable, a member of class TDBXValueTypeFlags.
	ReadOnly (see page 10653)	This is ReadOnly, a member of class TDBXValueTypeFlags.
	ReadOnlyType (see page 10653)	This is ReadOnlyType, a member of class TDBXValueTypeFlags.
	Searchable (see page 10653)	This is Searchable, a member of class TDBXValueTypeFlags.

55.1.75.2.1 TDBXValueTypeFlags.AutoIncrement Constant

Pascal

```
const AutoIncrement = $08;
```

C++

```
AutoIncrement = 0x08;
```

Description

This is AutoIncrement, a member of class TDBXValueTypeFlags.

55.1.75.2.2 TDBXValueTypeFlags.Nullable Constant

Pascal

```
const Nullable = $01;
```

C++

```
Nullable = 0x01;
```

Description

This is Nullable, a member of class TDBXValueTypeFlags.

55.1.75.2.3 TDBXValueTypeFlags.ReadOnly Constant

Pascal

```
const ReadOnly = $02;
```

C++

```
ReadOnly = 0x02;
```

Description

This is ReadOnly, a member of class TDBXValueTypeFlags.

55.1.75.2.4 TDBXValueTypeFlags.ReadOnlyType Constant

Pascal

```
const ReadOnlyType = $20;
```

C++

```
ReadOnlyType = 0x20;
```

Description

This is ReadOnlyType, a member of class TDBXValueTypeFlags.

55.1.75.2.5 TDBXValueTypeFlags.Searchable Constant

Pascal

```
const Searchable = $04;
```

C++

```
Searchable = 0x04;
```

Description

This is Searchable, a member of class TDBXValueTypeFlags.

55.1.76 TDBXValueTypeFlagsEx Class

Temporary extension of TDBXValueTypeFlags (see page 10651) class.

Class Hierarchy

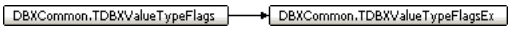

Pascal

```
TDBXValueTypeFlagsEx = class(TDBXValueTypeFlags);
```

C++

```
class TDBXValueTypeFlagsEx : public TDBXValueTypeFlags;
```

File

DBXCommon

Description

This class is an extension of the TDBXValueTypeFlags (see page 10651) class. Its flags provide information about a TDBXValueType (see page 10636) instance.

Note: Classes (see page 2685) ending in "Ex" are temporary classes that exist to avoid breaking interfaces. Their contents will be migrated to their base class in the next release.

This table contains the flag constants.

Constant	Description
ExtendedType (see page 10654)	Used internally to indicate that most recent value types should be used if applicable. Only used by the most up to date driver implementations since use of these types breaks old interfaces.

See Also

TDBXValueType (see page 10636)

TDBXValueTypeFlags (see page 10651)

55.1.76.1 TDBXValueTypeFlagsEx Members

The following tables list the members exposed by TDBXValueTypeFlagsEx.

Public Constants

	Name	Description
	AutoIncrement (see page 10652)	This is AutoIncrement, a member of class TDBXValueTypeFlags.
	ExtendedType (see page 10654)	This is ExtendedType, a member of class TDBXValueTypeFlagsEx.
	Nullable (see page 10652)	This is Nullable, a member of class TDBXValueTypeFlags.
	ReadOnly (see page 10653)	This is ReadOnly, a member of class TDBXValueTypeFlags.
	ReadOnlyType (see page 10653)	This is ReadOnlyType, a member of class TDBXValueTypeFlags.
	Searchable (see page 10653)	This is Searchable, a member of class TDBXValueTypeFlags.

55.1.76.2 TDBXValueTypeFlagsEx Constants

The constants of the TDBXValueTypeFlagsEx class are listed here.

Public Constants

	Name	Description
	ExtendedType (see page 10654)	This is ExtendedType, a member of class TDBXValueTypeFlagsEx.

55.1.76.2.1 TDBXValueTypeFlagsEx.ExtendedType Constant

Pascal

```
const ExtendedType = $40;
```

C++

```
ExtendedType = 0x40;
```

Description

This is ExtendedType, a member of class TDBXValueTypeFlagsEx.

55.1.77 TDBXWideCharsValue Class

Class Hierarchy

Pascal

```
TDBXWideCharsValue = class(TDBXWideStringValue);
```

C++

```
class TDBXWideCharsValue : public TDBXWideStringValue;
```

File

DBXCommon

Description

This is class DBXCommon.TDBXWideCharsValue.

55.1.77.1 TDBXWideCharsValue Members

The following tables list the members exposed by TDBXWideCharsValue.

Protected Methods

	Name	Description
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.
V	SetPendingValue (see page 10666)	This is SetPendingValue, a member of class TDBXWritableValue.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Methods

	Name	Description
	Create (see page 10656)	Creates an instance of TDBXWideStringValue
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
V	Destroy (see page 10656)	Destroys a TDBXWideStringValue instance.
V	GetAnsiString (see page 10629)	Gets the string value for ANSI strings.
V	GetBcd (see page 10629)	Gets the BCD value for binary-compressed decimal data types.
V	GetBoolean (see page 10629)	Gets the boolean value for boolean data types.
V	GetBytes (see page 10629)	Gets the byte array value for byte array and blob types.
V	GetDate (see page 10630)	Gets the date value for date data types.
V	GetDouble (see page 10630)	Gets the double floating point value for double data types.
V	GetInt16 (see page 10631)	Gets the smallInt value for 16-byte integer data types.
V	GetInt32 (see page 10631)	Gets the 32-byte integer value for 32-byte integer data types.
V	GetInt64 (see page 10631)	Gets the 64-byte integer value for 64-byte integer data types.
V	GetStream (see page 10657)	This is GetStream, a member of class TDBXWideCharsValue.
V	GetTime (see page 10631)	Gets the time value for time data types.
V	GetTimeStamp (see page 10632)	Gets the timestamp value for timestamp data types.
V	GetValueSize (see page 10632)	Gets the size in bytes for values.
V	GetWideString (see page 10657)	Gets the wide string value for wide string data types.

	IsNull (see page 10657)	Returns true if the value is **null**.
	SetAnsiString (see page 10663)	Sets the string value for ANSI strings.
	SetBcd (see page 10664)	Sets the BCD value for binary-compressed decimal data types.
	SetBoolean (see page 10664)	Sets the boolean value for boolean data types..
	SetDate (see page 10664)	Sets the date value for date data types..
	SetDouble (see page 10664)	Sets the double floating point value for double data types.
	SetDynamicBytes (see page 10665)	Sets the dynamic array value for byte array and blob types.
	SetInt16 (see page 10665)	Sets the the smallInt value for 16-byte integer data types.
	SetInt32 (see page 10665)	Sets the 32-byte integer value for 32-byte integer data types.
	SetInt64 (see page 10665)	Sets the 64-byte integer value for 64-byte integer data types.
	SetNull (see page 10666)	Sets a writable value of any type to null.
	SetStaticBytes (see page 10666)	Sets the open array value for byte array and blob types
	SetTime (see page 10666)	Sets the time value for time data types.
	SetTimeStamp (see page 10667)	Sets the timestamp value for timestamp data types.
	SetWideString (see page 10658)	Sets the wide string value for wide string data types.

Public Properties

	Name	Description
	ValueType (see page 10633)	Gets the value type for this value.

55.1.77.2 TDBXWideCharsValue.Create Constructor

Creates an instance of TDBXWideStringValue

Pascal

```
constructor Create(ValueType: TDBXValueType);
```

C++

```
__fastcall TDBXWideCharsValue(TDBXValueType ValueType);
```

Description

This constructor creates and initializes an instance of TDBXWideStringValue.

See Also

Destroy (see page 10660)

55.1.77.3 TDBXWideCharsValue.Destroy Destructor

Destroys a TDBXWideStringValue instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXWideCharsValue();
```

Description

This destructor destroys an instance of TDBXWideStringValue.

See Also

Create (see page 10659)

55.1.77.4 TDBXWideCharsValue Methods

The methods of the TDBXWideCharsValue class are listed here.

Public Methods

	Name	Description
	GetStream (see page 10657)	This is GetStream, a member of class TDBXWideCharsValue.
	GetWideString (see page 10657)	Gets the wide string value for wide string data types.
	IsNull (see page 10657)	Returns true if the value is **null**.
	SetWideString (see page 10658)	Sets the wide string value for wide string data types.

55.1.77.4.1 TDBXWideCharsValue.GetStream Method

Pascal

```
function GetStream: TStream; virtual;
```

C++

```
virtual __fastcall TStream GetStream();
```

Description

This is GetStream, a member of class TDBXWideCharsValue.

55.1.77.4.2 TDBXWideCharsValue.GetWideString Method

Gets the wide string value for wide string data types.

Pascal

```
function GetWideString: WideString; override;
```

C++

```
virtual __fastcall BSTR GetWideString();
```

Description

Gets the wide string value for TDBXValue instances with a data type of WideString. Throws a TDBXError exception if IsNull is true.

55.1.77.4.3 TDBXWideCharsValue.IsNull Method

Returns true if the value is **null**.

Pascal

```
function IsNull: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsNull();
```

Description

Returns true if the value is **null**. This method should be called before calling any of the Get methods. If the value is null, the Get methods will throw TDBXError exception if they are called.

55.1.77.4.4 TDBXWideCharsValue.SetWideString Method

Sets the wide string value for wide string data types.

Pascal

```
procedure SetWideString(const Value: WideString); override;
```

C++

```
virtual __fastcall SetWideString(const BSTR Value);
```

Description

Sets the wide string value for TDBXValue instances with a data type of WideString.

55.1.78 TDBXWideStringValue Class

Implementation of writable wide string values.

Class Hierarchy

Pascal

```
TDBXWideStringValue = class(TDBXWritableValue);
```

C++

```
class TDBXWideStringValue : public TDBXWritableValue;
```

File

DBXCommon

Description

Implementation of TDBXWritableValue (see page 10661) for TDBXDataTypes (see page 10447). WideStringType values.

55.1.78.1 TDBXWideStringValue Members

The following tables list the members exposed by TDBXWideStringValue.

Protected Methods

	Name	Description
	Create (see page 10659)	Creates an instance of TDBXWideStringValue (see page 10658)
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.
V	SetPendingValue (see page 10666)	This is SetPendingValue, a member of class TDBXWritableValue.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Methods

	Name	Description
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.

	Destroy (see page 10660)	Destroys a TDBXWideStringValue (see page 10658) instance.
	GetAnsiString (see page 10629)	Gets the string value for ANSI strings.
	GetBcd (see page 10629)	Gets the BCD value for binary-compressed decimal data types.
	GetBoolean (see page 10629)	Gets the boolean value for boolean data types.
	GetBytes (see page 10629)	Gets the byte array value for byte array and blob types.
	GetDate (see page 10630)	Gets the date value for date data types.
	GetDouble (see page 10630)	Gets the double floating point value for double data types.
	GetInt16 (see page 10631)	Gets the smallInt value for 16-byte integer data types.
	GetInt32 (see page 10631)	Gets the 32-byte integer value for 32-byte integer data types.
	GetInt64 (see page 10631)	Gets the 64-byte integer value for 64-byte integer data types.
	GetTime (see page 10631)	Gets the time value for time data types.
	GetTimeStamp (see page 10632)	Gets the timestamp value for timestamp data types.
	GetValueSize (see page 10632)	Gets the size in bytes for values.
	GetWideString (see page 10660)	Gets the wide string value for wide string data types.
	IsNull (see page 10660)	Returns true if the value is **null**.
	SetAnsiString (see page 10663)	Sets the string value for ANSI strings.
	SetBcd (see page 10664)	Sets the BCD value for binary-compressed decimal data types.
	SetBoolean (see page 10664)	Sets the boolean value for boolean data types..
	SetDate (see page 10664)	Sets the date value for date data types..
	SetDouble (see page 10664)	Sets the double floating point value for double data types.
	SetDynamicBytes (see page 10665)	Sets the dynamic array value for byte array and blob types.
	SetInt16 (see page 10665)	Sets the the smallInt value for 16-byte integer data types.
	SetInt32 (see page 10665)	Sets the 32-byte integer value for 32-byte integer data types.
	SetInt64 (see page 10665)	Sets the 64-byte integer value for 64-byte integer data types.
	SetNull (see page 10666)	Sets a writable value of any type to null.
	SetStaticBytes (see page 10666)	Sets the open array value for byte array and blob types
	SetTime (see page 10666)	Sets the time value for time data types.
	SetTimeStamp (see page 10667)	Sets the timestamp value for timestamp data types.
	SetWideString (see page 10661)	Sets the wide string value for wide string data types.

Public Properties

	Name	Description
	ValueType (see page 10633)	Gets the value type for this value.

55.1.78.2 TDBXWideStringValue.Create Constructor

Creates an instance of TDBXWideStringValue (see page 10658)

Pascal

```
constructor Create(ValueType: TDBXValueType);
```

C++

```
__fastcall TDBXWideStringValue(TDBXValueType ValueType);
```

Description

This constructor creates and initializes an instance of TDBXWideStringValue (see page 10658).

See Also

Destroy (see page 10660)

55.1.78.3 TDBXWideStringValue.Destroy Destructor

Destroys a TDBXWideStringValue (see page 10658) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXWideStringValue();
```

Description

This destructor destroys an instance of TDBXWideStringValue (see page 10658).

See Also

Create (see page 10659)

55.1.78.4 TDBXWideStringValue Methods

The methods of the TDBXWideStringValue class are listed here.

Public Methods

	Name	Description
	GetWideString (see page 10660)	Gets the wide string value for wide string data types.
	IsNull (see page 10660)	Returns true if the value is **null**.
	SetWideString (see page 10661)	Sets the wide string value for wide string data types.

55.1.78.4.1 TDBXWideStringValue.GetWideString Method

Gets the wide string value for wide string data types.

Pascal

```
function GetWideString: WideString; override;
```

C++

```
virtual __fastcall BSTR GetWideString();
```

Description

Gets the wide string value for TDBXValue (see page 10626) instances with a data type of WideString. Throws a TDBXError (see page 10486) exception if IsNull (see page 10660) is true.

55.1.78.4.2 TDBXWideStringValue.IsNull Method

Returns true if the value is **null**.

Pascal

```
function IsNull: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsNull();
```

Description

Returns true if the value is **null**. This method should be called before calling any of the Get methods. If the value is null, the Get

methods will throw TDBXError (see page 10486) exception if they are called.

55.1.78.4.3 TDBXWideStringValue.SetWideString Method

Sets the wide string value for wide string data types.

Pascal

```
procedure SetWideString(const Value: WideString); override;
```

C++

```
virtual __fastcall SetWideString(const BSTR Value);
```

Description

Sets the wide string value for TDBXValue (see page 10626) instances with a data type of WideString.

55.1.79 TDBXWritableValue Class

Extension of TDBXValue (see page 10626) that adds the ability to set a value.

Class Hierarchy

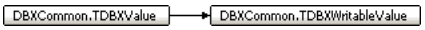

Pascal

```
TDBXWritableValue = class(TDBXValue);
```

C++

```
class TDBXWritableValue : public TDBXValue;
```

File

DBXCommon

Description

Extension of TDBXValue (see page 10626) that adds the ability to set a value. This is only used by Value. Value array property uses the readonly TDBXValue (see page 10626).

55.1.79.1 TDBXWritableValue Members

The following tables list the members exposed by TDBXWritableValue.

Protected Methods

	Name	Description
	Create (see page 10662)	Creates an instance of TDBXWritableValue (see page 10661)
	GetDbxContext (see page 10630)	Get context information for a connection associated with value.
V	SetPendingValue (see page 10666)	This is SetPendingValue, a member of class TDBXWritableValue.

Protected Properties

	Name	Description
R	DbxContext (see page 10633)	Context information for a connection associated with value.

Public Methods

	Name	Description
	CreateValue (see page 10628)	Creates values for both the TDBXReader (see page 10576) and TDBXParameterList (see page 10553) classes.
	Destroy (see page 10663)	Destroys a TDBXWritableValue (see page 10661) instance.
	GetAnsiString (see page 10629)	Gets the string value for ANSI strings.
	GetBcd (see page 10629)	Gets the BCD value for binary-compressed decimal data types.
	GetBoolean (see page 10629)	Gets the boolean value for boolean data types.
	GetBytes (see page 10629)	Gets the byte array value for byte array and blob types.
	GetDate (see page 10630)	Gets the date value for date data types.
	GetDouble (see page 10630)	Gets the double floating point value for double data types.
	GetInt16 (see page 10631)	Gets the smallInt value for 16-byte integer data types.
	GetInt32 (see page 10631)	Gets the 32-byte integer value for 32-byte integer data types.
	GetInt64 (see page 10631)	Gets the 64-byte integer value for 64-byte integer data types.
	GetTime (see page 10631)	Gets the time value for time data types.
	GetTimeStamp (see page 10632)	Gets the timestamp value for timestamp data types.
	GetValueSize (see page 10632)	Gets the size in bytes for values.
	GetWideString (see page 10632)	Gets the wide string value for wide string data types.
	IsNull (see page 10632)	Returns true if the value is **null**.
	SetAnsiString (see page 10663)	Sets the string value for ANSI strings.
	SetBcd (see page 10664)	Sets the BCD value for binary-compressed decimal data types.
	SetBoolean (see page 10664)	Sets the boolean value for boolean data types..
	SetDate (see page 10664)	Sets the date value for date data types..
	SetDouble (see page 10664)	Sets the double floating point value for double data types.
	SetDynamicBytes (see page 10665)	Sets the dynamic array value for byte array and blob types.
	SetInt16 (see page 10665)	Sets the the smallInt value for 16-byte integer data types.
	SetInt32 (see page 10665)	Sets the 32-byte integer value for 32-byte integer data types.
	SetInt64 (see page 10665)	Sets the 64-byte integer value for 64-byte integer data types.
	SetNull (see page 10666)	Sets a writable value of any type to null.
	SetStaticBytes (see page 10666)	Sets the open array value for byte array and blob types
	SetTime (see page 10666)	Sets the time value for time data types.
	SetTimeStamp (see page 10667)	Sets the timestamp value for timestamp data types.
	SetWideString (see page 10667)	Sets the wide string value for wide string data types.

Public Properties

	Name	Description
	ValueType (see page 10633)	Gets the value type for this value.

55.1.79.2 TDBXWritableValue.Create Constructor

Creates an instance of TDBXWritableValue (see page 10661)

Pascal

```
constructor Create(ValueType: TDBXValueType);
```

C++

```
__fastcall TDBXWritableValue(TDBXValueType ValueType);
```

Description

This constructor creates and initializes an instance of TDBXWritableValue (see page 10661).

See Also

Destroy (see page 10663)

55.1.79.3 TDBXWritableValue.Destroy Destructor

Destroys a TDBXWritableValue (see page 10661) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDBXWritableValue();
```

Description

This destructor destroys an instance of TDBXWritableValue (see page 10661).

See Also

Create (see page 10662)

55.1.79.4 TDBXWritableValue Methods

The methods of the TDBXWritableValue class are listed here.

Protected Methods

	Name	Description
	SetPendingValue (see page 10666)	This is SetPendingValue, a member of class TDBXWritableValue.

Public Methods

	Name	Description
	SetAnsiString (see page 10663)	Sets the string value for ANSI strings.
	SetBcd (see page 10664)	Sets the BCD value for binary-compressed decimal data types.
	SetBoolean (see page 10664)	Sets the boolean value for boolean data types..
	SetDate (see page 10664)	Sets the date value for date data types..
	SetDouble (see page 10664)	Sets the double floating point value for double data types.
	SetDynamicBytes (see page 10665)	Sets the dynamic array value for byte array and blob types.
	SetInt16 (see page 10665)	Sets the the smallInt value for 16-byte integer data types.
	SetInt32 (see page 10665)	Sets the 32-byte integer value for 32-byte integer data types.
	SetInt64 (see page 10665)	Sets the 64-byte integer value for 64-byte integer data types.
	SetNull (see page 10666)	Sets a writable value of any type to null.
	SetStaticBytes (see page 10666)	Sets the open array value for byte array and blob types
	SetTime (see page 10666)	Sets the time value for time data types.
	SetTimeStamp (see page 10667)	Sets the timestamp value for timestamp data types.
	SetWideString (see page 10667)	Sets the wide string value for wide string data types.

55.1.79.4.1 TDBXWritableValue.SetAnsiString Method

Sets the string value for ANSI strings.

Pascal

```
procedure SetAnsiString(const Value: String); virtual;
```

C++

```
virtual __fastcall SetAnsiString(const AnsiString Value);
```

Description

Sets the string value for TDBXValue (see page 10626) instances with a data type of AnsiStringType.

55.1.79.4.2 TDBXWritableValue.SetBcd Method

Sets the BCD value for binary-compressed decimal data types.

Pascal

```
procedure SetBcd(const Value: TBcd); virtual;
```

C++

```
virtual __fastcall SetBcd(const TBcd Value);
```

Description

Sets the TBcd value for TDBXValue (see page 10626) instances with a with a data type of BcdType.

55.1.79.4.3 TDBXWritableValue.SetBoolean Method

Sets the boolean value for boolean data types..

Pascal

```
procedure SetBoolean(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetBoolean(Boolean Value);
```

Description

Sets the boolean value for TDBXValue (see page 10626) instances with a data types of BooleanType..

55.1.79.4.4 TDBXWritableValue.SetDate Method

Sets the date value for date data types..

Pascal

```
procedure SetDate(Value: TDBXDate); virtual;
```

C++

```
virtual __fastcall SetDate(TDBXDate Value);
```

Description

Sets the date value for TDBXValue (see page 10626) instances with a data type of DateType.

55.1.79.4.5 TDBXWritableValue.SetDouble Method

Sets the double floating point value for double data types.

Pascal

```
procedure SetDouble(Value: Double); virtual;
```

C++

```
virtual __fastcall SetDouble(Double Value);
```

Description

Sets the double floating point value for TDBXValue (see page 10626) instances with a data type of DoubleType.

55.1.79.4.6 TDBXWritableValue.SetDynamicBytes Method

Sets the dynamic array value for byte array and blob types.

Pascal

```
procedure SetDynamicBytes(Offset: Int64; const Buffer: TBytes; BufferOffset: Int64; Length:
Int64); virtual;
```

C++

```
virtual __fastcall SetDynamicBytes(Int64 Offset, const TBytes Buffer, Int64 BufferOffset,
Int64 Length);
```

Description

Sets the dynamic array value for TDBXValue (see page 10626) instances with a data type of BytesType and .BlobType.

55.1.79.4.7 TDBXWritableValue.SetInt16 Method

Sets the the smallInt value for 16-byte integer data types.

Pascal

```
procedure SetInt16(Value: SmallInt); virtual;
```

C++

```
virtual __fastcall SetInt16(SmallInt Value);
```

Description

Sets the the smallInt value for TDBXValue (see page 10626) instances with a data type of Int16Type.

55.1.79.4.8 TDBXWritableValue.SetInt32 Method

Sets the 32-byte integer value for 32-byte integer data types.

Pascal

```
procedure SetInt32(Value: TInt32); virtual;
```

C++

```
virtual __fastcall SetInt32(TInt32 Value);
```

Description

Sets the 32-byte integer value for TDBXValue (see page 10626) instances with a data type of Int32Type.

55.1.79.4.9 TDBXWritableValue.SetInt64 Method

Sets the 64-byte integer value for 64-byte integer data types.

Pascal

```
procedure SetInt64(Value: Int64); virtual;
```

C++

```
virtual __fastcall SetInt64(Int64 Value);
```

Description

Sets the 64-byte integer value for TDBXValue (see page 10626) instances with a data type of Int64Type.

55.1.79.4.10 TDBXWritableValue.SetNull Method

Sets a writable value of any type to null.

Pascal

```
procedure SetNull; virtual;
```

C++

```
virtual __fastcall SetNull();
```

Description

Sets a TDBXWritableValue (see page 10661) instance of any type to null.

55.1.79.4.11 TDBXWritableValue.SetPendingValue Method

Pascal

```
procedure SetPendingValue; virtual;
```

C++

```
virtual __fastcall SetPendingValue();
```

Description

This is SetPendingValue, a member of class TDBXWritableValue.

55.1.79.4.12 TDBXWritableValue.SetStaticBytes Method

Sets the open array value for byte array and blob types

Pascal

```
procedure SetStaticBytes(Offset: Int64; const Buffer: array of Byte; BufferOffset: Int64;
Length: Int64); virtual;
```

C++

```
virtual __fastcall SetStaticBytes(Int64 Offset, const array of Byte Buffer, Int64
BufferOffset, Int64 Length);
```

Description

Sets the open array value for TDBXValue (see page 10626) instances with a data type of BytesType and .BlobType.

55.1.79.4.13 TDBXWritableValue.SetTime Method

Sets the time value for time data types.

Pascal

```
procedure SetTime(Value: TDBXTime); virtual;
```

C++

```
virtual __fastcall SetTime(TDBXTime Value);
```

Description

Sets the time value for TDBXValue (see page 10626) instances with a data type of TimeType.

55.1.79.4.14 TDBXWritableValue.SetTimeStamp Method

Sets the timestamp value for timestamp data types.

Pascal

```
procedure SetTimeStamp(const Value: TSQLTimeStamp); virtual;
```

C++

```
virtual __fastcall SetTimeStamp(const TSQLTimeStamp Value);
```

Description

Sets the timestamp value for TDBXValue (see page 10626) instances with a data types of TimeStampType.

55.1.79.4.15 TDBXWritableValue.SetWideString Method

Sets the wide string value for wide string data types.

Pascal

```
procedure SetWideString(const Value: WideString); virtual;
```

C++

```
virtual __fastcall SetWideString(const BSTR Value);
```

Description

Sets the wide string value for TDBXValue (see page 10626) instances with a data type of WideString.

55.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TDBXDriverDef (see page 10668)	Collection of parameters needed when loading a driver.

55.2.1 DBXCommon.TDBXDriverDef Record

Collection of parameters needed when loading a driver.

Pascal

```
TDBXDriverDef = packed record
  FDriverName: WideString;
  FDriverProperties: TDBXProperties;
  FDBXContext: TDBXContext;
end;
```

C++

```
struct TDBXDriverDef {
  BSTR FDriverName;
  TDBXProperties FDriverProperties;
  TDBXContext FDBXContext;
};
```

File

DBXCommon

Description

Used by driver implementors. Collection of parameters needed when loading a driver. By collecting these in a single record, parameters can be added without necessarily breaking older drivers.

55.3 Types

The following table lists types in this documentation.

Types

Name	Description
TDBXChars (see page 10669)	This is type DBXCommon.TDBXChars.
TDBXCreateCommandEvent (see page 10669)	This is type DBXCommon.TDBXCreateCommandEvent.
TDBXDate (see page 10670)	This is type DBXCommon.TDBXDate.
TDBXDriverLoaderClass (see page 10670)	This is type DBXCommon.TDBXDriverLoaderClass.
TDBXErrorCode (see page 10670)	This is type DBXCommon.TDBXErrorCode.
TDBXErrorEvent (see page 10671)	Event handler for TDBXConnection.OnErrorEvent property.
TDBXIsolation (see page 10671)	This is type DBXCommon.TDBXIsolation.
TDBXParameterDirection (see page 10671)	This is type DBXCommon.TDBXParameterDirection.
TDBXTime (see page 10672)	This is type DBXCommon.TDBXTime.
TDBXTraceEvent (see page 10672)	Tracing information for drivers.
TDBXTraceFlag (see page 10673)	This is type DBXCommon.TDBXTraceFlag.
TDBXType (see page 10673)	Type for TDBXDataTypes (see page 10447).
TDBXValueArray (see page 10673)	This is type DBXCommon.TDBXValueArray.
TWideStringArray (see page 10674)	This is type DBXCommon.TWideStringArray.

55.3.1 DBXCommon.TDBXChars Type

Pascal

```
TDBXChars = array of Char;
```

C++

```
array of Char TDBXChars;
```

File

DBXCommon

Description

This is type DBXCommon.TDBXChars.

55.3.2 DBXCommon.TDBXCreateCommandEvent Type

Pascal

```
TDBXCreateCommandEvent = function (DbxContext: TDBXContext; Connection: TDBXConnection;
MorphicCommand: TDBXCommand): TDBXCommand of object;
```

C++

```
(DbxContext: TDBXContext; Connection: TDBXConnection; MorphicCommand: TDBXCommand):
TDBXCommand ( TDBXCreateCommandEvent)();
```

File

DBXCommon

Description

This is type DBXCommon.TDBXCreateCommandEvent.

55.3.3 DBXCommon.TDBXDate Type

Pascal

```
TDBXDate = TInt32;
```

C++

```
TInt32 TDBXDate;
```

File

DBXCommon

Description

This is type DBXCommon.TDBXDate.

55.3.4 DBXCommon.TDBXDriverLoaderClass Type

Pascal

```
TDBXDriverLoaderClass = class of TDBXDriverLoader;
```

C++

```
TDBXDriverLoader TDBXDriverLoaderClass;
```

File

DBXCommon

Description

This is type DBXCommon.TDBXDriverLoaderClass.

55.3.5 DBXCommon.TDBXErrorCode Type

Pascal

```
TDBXErrorCode = TInt32;
```

C++

```
TInt32 TDBXErrorCode;
```

File

DBXCommon

Description

This is type DBXCommon.TDBXErrorCode.

55.3.6 DBXCommon.TDBXErrorEvent Type

Event handler for TDBXConnection.OnErrorEvent property.

Pascal

```
TDBXErrorEvent = procedure (DBXError: TDBXError) of object;
```

C++

```
(DBXError: TDBXError) ( TDBXErrorEvent)();
```

File

DBXCommon

Description

TDBXErrorEvent is an event handler that can be set on the TDBXConnection.OnErrorEvent property. This event is called just before a TDBXError (see page 10486) exception is raised. This allows implementors to perform processing before the exception is raised. dbExpress VCL components use this event to raise an exception of a different type.

See Also

TDBXConnection (see page 10398)

TDBXError (see page 10486)

OnErrorEvent (see page 10411)

55.3.7 DBXCommon.TDBXIsolation Type

Pascal

```
TDBXIsolation = TInt32;
```

C++

```
TInt32 TDBXIsolation;
```

File

DBXCommon

Description

This is type DBXCommon.TDBXIsolation.

55.3.8 DBXCommon.TDBXParameterDirection Type

Pascal

```
TDBXParameterDirection = TInt32;
```

C++

```
TInt32 TDBXParameterDirection;
```

File

DBXCommon

Description

This is type DBXCommon.TDBXParameterDirection.

55.3.9 DBXCommon.TDBXTime Type

Pascal

```
TDBXTime = TInt32;
```

C++

```
TInt32 TDBXTime;
```

File

DBXCommon

Description

This is type DBXCommon.TDBXTime.

55.3.10 DBXCommon.TDBXTraceEvent Type

Tracing information for drivers.

Pascal

```
TDBXTraceEvent = function (TraceInfo: TDBXTraceInfo): CBRType of object;
```

C++

```
(TraceInfo: TDBXTraceInfo): CBRType ( TDBXTraceEvent)();
```

File

DBXCommon

Description

This event provides tracing information for DBX database drivers. TDBXConnection (see page 10398) has a TDBXTraceEvent property called OnTrace. The TDBXTrace delegate driver uses this event to provide DBX driver framework level tracing.

See Also

TDBXConnection (see page 10398)

OnTrace (see page 10412)

55.3.11 DBXCommon.TDBXTraceFlag Type

Pascal

```
TDBXTraceFlag = TInt32;
```

C++

```
TInt32 TDBXTraceFlag;
```

File

DBXCommon

Description

This is type DBXCommon.TDBXTraceFlag.

55.3.12 DBXCommon.TDBXType Type

Type for TDBXDataTypes (see page 10447).

Pascal

```
TDBXType = TInt32;
```

C++

```
TInt32 TDBXType;
```

File

DBXCommon

Description

TDBXType is a type for TDBXDataTypes (see page 10447).

See Also

TDBXDataTypes (see page 10447)

55.3.13 DBXCommon.TDBXValueArray Type

Pascal

```
TDBXValueArray = array of TDBXValue;
```

C++

```
array of TDBXValue TDBXValueArray;
```

File

DBXCommon

Description

This is type DBXCommon.TDBXValueArray.

55.3.14 DBXCommon.TWideStringArray Type

Pascal

```
TWideStringArray = array of WideString;
```

C++

```
array of WideString TWideStringArray;
```

File

DBXCommon

Description

This is type DBXCommon.TWideStringArray.

55.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
DBXDefaultRowSetSize (see page 10675)	This is constant DBXCommon.DBXDefaultRowSetSize.
DBXVersion25 (see page 10676)	This is constant DBXCommon.DBXVersion25.
DBXVersion30 (see page 10676)	This is constant DBXCommon.DBXVersion30.
DBXVersion40 (see page 10676)	This is constant DBXCommon.DBXVersion40.
SAlreadyPrepared (see page 10677)	This is constant DBXCommon.SAlreadyPrepared.
SConnectionFactoryInitFailed (see page 10677)	This is constant DBXCommon.SConnectionFactoryInitFailed.
SDefaultErrorMessage (see page 10677)	This is constant DBXCommon.SDefaultErrorMessage.
SDllLoadError (see page 10678)	This is constant DBXCommon.SDllLoadError.
SDllProcLoadError (see page 10678)	This is constant DBXCommon.SDllProcLoadError.
SDriverLoadError (see page 10678)	This is constant DBXCommon.SDriverLoadError.
SInvalidArgument (see page 10679)	This is constant DBXCommon.SInvalidArgument.
SInvalidDelegationDepth (see page 10679)	This is constant DBXCommon.SInvalidDelegationDepth.
SInvalidOrdinal (see page 10679)	This is constant DBXCommon.SInvalidOrdinal.
SInvalidTransaction (see page 10680)	This is constant DBXCommon.SInvalidTransaction.
SInvalidTypeAccess (see page 10680)	This is constant DBXCommon.SInvalidTypeAccess.
SNotImplemented (see page 10680)	This is constant DBXCommon.SNotImplemented.
SReaderClosed (see page 10681)	This is constant DBXCommon.SReaderClosed.
SReaderNew (see page 10681)	This is constant DBXCommon.SReaderNew.
SReadOnlyParameter (see page 10681)	This is constant DBXCommon.SReadOnlyParameter.
SReadOnlyType (see page 10682)	This is constant DBXCommon.SReadOnlyType.
SRequiredProperty (see page 10682)	This is constant DBXCommon.SRequiredProperty.
SSetFactoryOnce (see page 10682)	This is constant DBXCommon.SSetFactoryOnce.
SUnknownDriver (see page 10683)	This is constant DBXCommon.SUnknownDriver.
TDBXConnectionFile (see page 10683)	This is constant DBXCommon.TDBXConnectionFile.
TDBXDriverFile (see page 10683)	This is constant DBXCommon.TDBXDriverFile.
TDBXRegistryConnectionValue (see page 10684)	This is constant DBXCommon.TDBXRegistryConnectionValue.
TDBXRegistryDriverValue (see page 10684)	This is constant DBXCommon.TDBXRegistryDriverValue.
TDBXRegistryKey (see page 10684)	This is constant DBXCommon.TDBXRegistryKey.

55.4.1 DBXCommon.DBXDefaultRowSetSize Constant

Pascal

```
DBXDefaultRowSetSize = 20;
```

C++

```
DBXDefaultRowSetSize = 20;
```

File

DBXCommon

Description

This is constant DBXCommon.DBXDefaultRowSetSize.

55.4.2 DBXCommon.DBXVersion25 Constant

Pascal

```
DBXVersion25 = '2.5';
```

C++

```
DBXVersion25 = '2.5';
```

File

DBXCommon

Description

This is constant DBXCommon.DBXVersion25.

55.4.3 DBXCommon.DBXVersion30 Constant

Pascal

```
DBXVersion30 = '3.0';
```

C++

```
DBXVersion30 = '3.0';
```

File

DBXCommon

Description

This is constant DBXCommon.DBXVersion30.

55.4.4 DBXCommon.DBXVersion40 Constant

Pascal

```
DBXVersion40 = '4.0';
```

C++

```
DBXVersion40 = '4.0';
```

File

DBXCommon

Description

This is constant DBXCommon.DBXVersion40.

55.4.5 DBXCommon.SAlreadyPrepared Constant

Pascal

```
SAlreadyPrepared = 'Command can only be prepared once';
```

C++

```
SAlreadyPrepared = 'Command can only be prepared once';
```

File

DBXCommon

Description

This is constant DBXCommon.SAlreadyPrepared.

55.4.6 DBXCommon.SConnectionFactoryInitFailed Constant

Pascal

```
SConnectionFactoryInitFailed = 'Cannot find connection files from application directory (%s)
or the system registry (%s).';
```

C++

```
SConnectionFactoryInitFailed = 'Cannot find connection files from application directory (%s)
or the system registry (%s).';
```

File

DBXCommon

Description

This is constant DBXCommon.SConnectionFactoryInitFailed.

55.4.7 DBXCommon.SDefaultErrorMessage Constant

Pascal

```
SDefaultErrorMessage = 'DBX Error:  %s';
```

C++

```
SDefaultErrorMessage = 'DBX Error:  %s';
```

File

DBXCommon

Description

This is constant DBXCommon.SDefaultErrorMessage.

55.4.8 DBXCommon.SDllLoadError Constant

Pascal

```
SDllLoadError = 'Unable to load %s (ErrorCode %d).  It may be missing from the system path.';
```

C++

```
SDllLoadError = 'Unable to load %s (ErrorCode %d).  It may be missing from the system path.';
```

File

DBXCommon

Description

This is constant DBXCommon.SDllLoadError.

55.4.9 DBXCommon.SDllProcLoadError Constant

Pascal

```
SDllProcLoadError = 'Unable to find procedure %s';
```

C++

```
SDllProcLoadError = 'Unable to find procedure %s';
```

File

DBXCommon

Description

This is constant DBXCommon.SDllProcLoadError.

55.4.10 DBXCommon.SDriverLoadError Constant

Pascal

```
SDriverLoadError = '%s driver cannot be loaded.  Make sure your project either uses the %s
unit or uses packages so the %s package can be loaded dynamically';
```

C++

```
SDriverLoadError = '%s driver cannot be loaded.  Make sure your project either uses the %s
unit or uses packages so the %s package can be loaded dynamically';
```

File

DBXCommon

Description

This is constant DBXCommon.SDriverLoadError.

55.4.11 DBXCommon.SInvalidArgument Constant

Pascal

```
SInvalidArgument = 'Invalid argument:  %s';
```

C++

```
SInvalidArgument = 'Invalid argument:  %s';
```

File

DBXCommon

Description

This is constant DBXCommon.SInvalidArgument.

55.4.12 DBXCommon.SInvalidDelegationDepth Constant

Pascal

```
SInvalidDelegationDepth = 'Cannot delegate a connection more than 16 times:  %s';
```

C++

```
SInvalidDelegationDepth = 'Cannot delegate a connection more than 16 times:  %s';
```

File

DBXCommon

Description

This is constant DBXCommon.SInvalidDelegationDepth.

55.4.13 DBXCommon.SInvalidOrdinal Constant

Pascal

```
SInvalidOrdinal = 'Invalid Ordinal:  %d';
```

C++

```
SInvalidOrdinal = 'Invalid Ordinal:  %d';
```

File

DBXCommon

Description

This is constant DBXCommon.SInvalidOrdinal.

55.4.14 DBXCommon.SInvalidTransaction Constant

Pascal

```
SInvalidTransaction = 'Invalid transaction Object';
```

C++

```
SInvalidTransaction = 'Invalid transaction Object';
```

File

DBXCommon

Description

This is constant DBXCommon.SInvalidTransaction.

55.4.15 DBXCommon.SInvalidTypeAccess Constant

Pascal

```
SInvalidTypeAccess = '%s value type cannot be accessed as %s value type';
```

C++

```
SInvalidTypeAccess = '%s value type cannot be accessed as %s value type';
```

File

DBXCommon

Description

This is constant DBXCommon.SInvalidTypeAccess.

55.4.16 DBXCommon.SNotImplemented Constant

Pascal

```
SNotImplemented = 'Feature not implemented';
```

C++

```
SNotImplemented = 'Feature not implemented';
```

File

DBXCommon

Description

This is constant DBXCommon.SNotImplemented.

55.4.17 DBXCommon.SReaderClosed Constant

Pascal

```
SReaderClosed = 'Reader has no more rows';
```

C++

```
SReaderClosed = 'Reader has no more rows';
```

File

DBXCommon

Description

This is constant DBXCommon.SReaderClosed.

55.4.18 DBXCommon.SReaderNew Constant

Pascal

```
SReaderNew = 'Reader Next method has not been called';
```

C++

```
SReaderNew = 'Reader Next method has not been called';
```

File

DBXCommon

Description

This is constant DBXCommon.SReaderNew.

55.4.19 DBXCommon.SReadOnlyParameter Constant

Pascal

```
SReadOnlyParameter = '%s parameter cannot be modified';
```

C++

```
SReadOnlyParameter = '%s parameter cannot be modified';
```

File

DBXCommon

Description

This is constant DBXCommon.SReadOnlyParameter.

55.4.20 DBXCommon.SReadOnlyType Constant

Pascal

```
SReadOnlyType = '%s type cannot be modified';
```

C++

```
SReadOnlyType = '%s type cannot be modified';
```

File

DBXCommon

Description

This is constant DBXCommon.SReadOnlyType.

55.4.21 DBXCommon.SRequiredProperty Constant

Pascal

```
SRequiredProperty = '%s property not set';
```

C++

```
SRequiredProperty = '%s property not set';
```

File

DBXCommon

Description

This is constant DBXCommon.SRequiredProperty.

55.4.22 DBXCommon.SSetFactoryOnce Constant

Pascal

```
SSetFactoryOnce = 'Cannot set the ConnectionFactory singleton more than once';
```

C++

```
SSetFactoryOnce = 'Cannot set the ConnectionFactory singleton more than once';
```

File

DBXCommon

Description

This is constant DBXCommon.SSetFactoryOnce.

55.4.23 DBXCommon.SUnknownDriver Constant

Pascal

```
SUnknownDriver = 'Unknown driver:  %s';
```

C++

```
SUnknownDriver = 'Unknown driver:  %s';
```

File

DBXCommon

Description

This is constant DBXCommon.SUnknownDriver.

55.4.24 DBXCommon.TDBXConnectionFile Constant

Pascal

```
TDBXConnectionFile = 'dbxconnections.ini';
```

C++

```
TDBXConnectionFile = 'dbxconnections.ini';
```

File

DBXCommon

Description

This is constant DBXCommon.TDBXConnectionFile.

55.4.25 DBXCommon.TDBXDriverFile Constant

Pascal

```
TDBXDriverFile = 'dbxdrivers.ini';
```

C++

```
TDBXDriverFile = 'dbxdrivers.ini';
```

File

DBXCommon

Description

This is constant DBXCommon.TDBXDriverFile.

55.4.26 DBXCommon.TDBXRegistryConnectionValue Constant

Pascal

```
TDBXRegistryConnectionValue = 'Connection Registry File';
```

C++

```
TDBXRegistryConnectionValue = 'Connection Registry File';
```

File

DBXCommon

Description

This is constant DBXCommon.TDBXRegistryConnectionValue.

55.4.27 DBXCommon.TDBXRegistryDriverValue Constant

Pascal

```
TDBXRegistryDriverValue = 'Driver Registry File';
```

C++

```
TDBXRegistryDriverValue = 'Driver Registry File';
```

File

DBXCommon

Description

This is constant DBXCommon.TDBXRegistryDriverValue.

55.4.28 DBXCommon.TDBXRegistryKey Constant

Pascal

```
TDBXRegistryKey = '\Software\Borland\BDS\5.0\dbExpress';
```

C++

```
TDBXRegistryKey = '\Software\Borland\BDS\5.0\dbExpress';
```

File

DBXCommon

Description

This is constant DBXCommon.TDBXRegistryKey.

56 DBXMetaDataNames Namespace

This is namespace DBXMetaDataNames.

Topics

Name	Description
Classes (see page 10687)	The following table lists classes in this documentation.

Classes

	Name	Description
	TDBXCatalogsColumns (see page 10688)	Accesses columns by name in the **Catalogs** metadata.
	TDBXCatalogsIndex (see page 10689)	Accesses columns by ordinal in the **Catalogs** metadata.
	TDBXColumnConstraintsColumns (see page 10690)	Internal use only
	TDBXColumnConstraintsIndex (see page 10692)	Internal use only
	TDBXColumnsColumns (see page 10695)	Accesses columns by name in the **Columns** metadata.
	TDBXColumnsIndex (see page 10701)	Accesses columns by ordinal in the **Columns** metadata.
	TDBXDataTypesColumns (see page 10707)	Accesses columns by name in the **DataTypes** metadata.
	TDBXDataTypesIndex (see page 10716)	Accesses columns by ordinal in the **DataTypes** metadata.
	TDBXForeignKeyColumnsColumns (see page 10725)	Accesses columns by name in the **ForeignKeyColumns** metadata.
	TDBXForeignKeyColumnsIndex (see page 10729)	Accesses columns by ordinal in the **ForeignKeyColumns** metadata.
	TDBXForeignKeysColumns (see page 10734)	Accesses columns by name in the **ForeignKeys** metadata.
	TDBXForeignKeysIndex (see page 10736)	Accesses columns by ordinal in the **ForeignKeys** metadata.
	TDBXIndexColumnsColumns (see page 10738)	Accesses columns by name in the **IndexColumns** metadata.
	TDBXIndexColumnsIndex (see page 10741)	Accesses columns by ordinal in the **IndexColumns** metadata.
	TDBXIndexesColumns (see page 10744)	Accesses columns by name in the **Indexes** metadata.
	TDBXIndexesIndex (see page 10748)	Accesses columns by ordinal in the **Indexes** metadata.
	TDBXMetaDataCollectionIndex (see page 10751)	Represents a collection of metadata accessed by ordinal.
	TDBXMetaDataCollectionName (see page 10758)	Represents a collection of metadata accessed by name.
	TDBXPackageProcedureParametersColumns (see page 10766)	Accesses columns by name in the **PackageProcedureParameters** metadata.
	TDBXPackageProcedureParametersIndex (see page 10772)	Accesses columns by ordinal in the **PackageProcedureParameters** metadata.
	TDBXPackageProceduresColumns (see page 10778)	Accesses columns by name in the **PackageProcedures** metadata.
	TDBXPackageProceduresIndex (see page 10781)	Accesses columns by ordinal in the **PackageProcedures** metadata.
	TDBXPackagesColumns (see page 10783)	Accesses columns by name in the **Packages** metadata.
	TDBXPackagesIndex (see page 10785)	Accesses columns by ordinal in the **Packages** metadata.
	TDBXPackageSourcesColumns (see page 10787)	Accesses columns by name in the **PackageSources** metadata.
	TDBXPackageSourcesIndex (see page 10789)	Accesses columns by ordinal in the **PackageSources** metadata.

	TDBXProcedureParametersColumns (see page 10792)	Accesses columns by name in the **ProcedureParameters** metadata.
	TDBXProcedureParametersIndex (see page 10797)	Accesses columns by ordinal in the **ProcedureParameters** metadata.
	TDBXProceduresColumns (see page 10803)	Accesses columns by name in the **Procedures** metadata.
	TDBXProceduresIndex (see page 10805)	Accesses columns by ordinal in the **Procedures** metadata.
	TDBXProcedureSourcesColumns (see page 10807)	Accesses columns by name in the **ProcedureSources** metadata.
	TDBXProcedureSourcesIndex (see page 10810)	Accesses columns by ordinal in the **ProcedureSources** metadata.
	TDBXReservedWordsColumns (see page 10813)	Accesses columns by name in the **ReservedWords** metadata.
	TDBXReservedWordsIndex (see page 10814)	Accesses columns by ordinal in the **ReservedWords** metadata.
	TDBXRolesColumns (see page 10816)	Accesses columns by name in the **Roles** metadata.
	TDBXRolesIndex (see page 10817)	Accesses columns by ordinal in the **Roles** metadata.
	TDBXSchemasColumns (see page 10818)	Accesses columns by name in the **Schemas** metadata.
	TDBXSchemasIndex (see page 10820)	Accesses columns by ordinal in the **Schemas** metadata.
	TDBXSynonymsColumns (see page 10822)	Accesses columns by name in the **Synonyms** metadata.
	TDBXSynonymsIndex (see page 10824)	Accesses columns by ordinal in the **Synonyms** metadata.
	TDBXTablesColumns (see page 10827)	Accesses columns by name in the **Tables** metadata.
	TDBXTablesIndex (see page 10829)	Accesses columns by ordinal in the **Tables** metadata.
	TDBXUsersColumns (see page 10832)	Accesses columns by name in the **Users** metadata.
	TDBXUsersIndex (see page 10833)	Accesses columns by ordinal in the **Users** metadata.
	TDBXViewsColumns (see page 10834)	Accesses columns by name in the **Views** metadata.
	TDBXViewsIndex (see page 10836)	Accesses columns by ordinal in the **Views** metadata.

56.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TDBXCatalogsColumns (see page 10688)	Accesses columns by name in the **Catalogs** metadata.
	TDBXCatalogsIndex (see page 10689)	Accesses columns by ordinal in the **Catalogs** metadata.
	TDBXColumnConstraintsColumns (see page 10690)	Internal use only
	TDBXColumnConstraintsIndex (see page 10692)	Internal use only
	TDBXColumnsColumns (see page 10695)	Accesses columns by name in the **Columns** metadata.
	TDBXColumnsIndex (see page 10701)	Accesses columns by ordinal in the **Columns** metadata.
	TDBXDataTypesColumns (see page 10707)	Accesses columns by name in the **DataTypes** metadata.
	TDBXDataTypesIndex (see page 10716)	Accesses columns by ordinal in the **DataTypes** metadata.
	TDBXForeignKeyColumnsColumns (see page 10725)	Accesses columns by name in the **ForeignKeyColumns** metadata.
	TDBXForeignKeyColumnsIndex (see page 10729)	Accesses columns by ordinal in the **ForeignKeyColumns** metadata.
	TDBXForeignKeysColumns (see page 10734)	Accesses columns by name in the **ForeignKeys** metadata.
	TDBXForeignKeysIndex (see page 10736)	Accesses columns by ordinal in the **ForeignKeys** metadata.
	TDBXIndexColumnsColumns (see page 10738)	Accesses columns by name in the **IndexColumns** metadata.
	TDBXIndexColumnsIndex (see page 10741)	Accesses columns by ordinal in the **IndexColumns** metadata.
	TDBXIndexesColumns (see page 10744)	Accesses columns by name in the **Indexes** metadata.
	TDBXIndexesIndex (see page 10748)	Accesses columns by ordinal in the **Indexes** metadata.
	TDBXMetaDataCollectionIndex (see page 10751)	Represents a collection of metadata accessed by ordinal.
	TDBXMetaDataCollectionName (see page 10758)	Represents a collection of metadata accessed by name.
	TDBXPackageProcedureParametersColumns (see page 10766)	Accesses columns by name in the **PackageProcedureParameters** metadata.
	TDBXPackageProcedureParametersIndex (see page 10772)	Accesses columns by ordinal in the **PackageProcedureParameters** metadata.
	TDBXPackageProceduresColumns (see page 10778)	Accesses columns by name in the **PackageProcedures** metadata.
	TDBXPackageProceduresIndex (see page 10781)	Accesses columns by ordinal in the **PackageProcedures** metadata.
	TDBXPackagesColumns (see page 10783)	Accesses columns by name in the **Packages** metadata.
	TDBXPackagesIndex (see page 10785)	Accesses columns by ordinal in the **Packages** metadata.
	TDBXPackageSourcesColumns (see page 10787)	Accesses columns by name in the **PackageSources** metadata.
	TDBXPackageSourcesIndex (see page 10789)	Accesses columns by ordinal in the **PackageSources** metadata.
	TDBXProcedureParametersColumns (see page 10792)	Accesses columns by name in the **ProcedureParameters** metadata.
	TDBXProcedureParametersIndex (see page 10797)	Accesses columns by ordinal in the **ProcedureParameters** metadata.
	TDBXProceduresColumns (see page 10803)	Accesses columns by name in the **Procedures** metadata.
	TDBXProceduresIndex (see page 10805)	Accesses columns by ordinal in the **Procedures** metadata.
	TDBXProcedureSourcesColumns (see page 10807)	Accesses columns by name in the **ProcedureSources** metadata.
	TDBXProcedureSourcesIndex (see page 10810)	Accesses columns by ordinal in the **ProcedureSources** metadata.
	TDBXReservedWordsColumns (see page 10813)	Accesses columns by name in the **ReservedWords** metadata.
	TDBXReservedWordsIndex (see page 10814)	Accesses columns by ordinal in the **ReservedWords** metadata.
	TDBXRolesColumns (see page 10816)	Accesses columns by name in the **Roles** metadata.
	TDBXRolesIndex (see page 10817)	Accesses columns by ordinal in the **Roles** metadata.
	TDBXSchemasColumns (see page 10818)	Accesses columns by name in the **Schemas** metadata.
	TDBXSchemasIndex (see page 10820)	Accesses columns by ordinal in the **Schemas** metadata.
	TDBXSynonymsColumns (see page 10822)	Accesses columns by name in the **Synonyms** metadata.
	TDBXSynonymsIndex (see page 10824)	Accesses columns by ordinal in the **Synonyms** metadata.
	TDBXTablesColumns (see page 10827)	Accesses columns by name in the **Tables** metadata.
	TDBXTablesIndex (see page 10829)	Accesses columns by ordinal in the **Tables** metadata.
	TDBXUsersColumns (see page 10832)	Accesses columns by name in the **Users** metadata.

	TDBXUsersIndex (see page 10833)	Accesses columns by ordinal in the **Users** metadata.
	TDBXViewsColumns (see page 10834)	Accesses columns by name in the **Views** metadata.
	TDBXViewsIndex (see page 10836)	Accesses columns by ordinal in the **Views** metadata.

56.1.1 TDBXCatalogsColumns Class

Accesses columns by name in the **Catalogs** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXCatalogsColumns

Pascal

```
TDBXCatalogsColumns = class;
```

C++

```
class TDBXCatalogsColumns;
```

File

DBXMetaDataNames

Description

TDBXCatalogsColumns is used to access columns by name in the **Catalogs** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetCatalogs when reading metadata. To retrieve the metadata column, use this constant to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the column.

Column Name	Data Type of Column Data	Description
CatalogName (see page 10689)	String	Catalog name.

See Also

TDBXCatalogsIndex (see page 10689)

TDBXMetaDataCommandsEx (see page 10531)

56.1.1.1 TDBXCatalogsColumns Members

The following tables list the members exposed by TDBXCatalogsColumns.

Public Constants

	Name	Description
	CatalogName (see page 10689)	This is CatalogName, a member of class TDBXCatalogsColumns.

56.1.1.2 TDBXCatalogsColumns Constants

The constants of the TDBXCatalogsColumns class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10689)	This is CatalogName, a member of class TDBXCatalogsColumns.

56.1.1.2.1 TDBXCatalogsColumns.CatalogName Constant

Pascal

```
const CatalogName = 'CatalogName';
```

C++

```
CatalogName = 'CatalogName';
```

Description

This is CatalogName, a member of class TDBXCatalogsColumns.

56.1.2 TDBXCatalogsIndex Class

Accesses columns by ordinal in the **Catalogs** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXCatalogsIndex

Pascal

```
TDBXCatalogsIndex = class;
```

C++

```
class TDBXCatalogsIndex;
```

File

DBXMetaDataNames

Description

TDBXCatalogsIndex is used to access columns by ordinal in the **Catalogs** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetCatalogs when reading metadata. To retrieve a metadata column, use this constant to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the column.

Column Ordinal Name	Data Type of Column Data	Description
CatalogName (see page 10690)	String	Catalog name.

See Also

TDBXCatalogsColumns (see page 10688)

TDBXMetaDataCommandsEx (see page 10531)

56.1.2.1 TDBXCatalogsIndex Members

The following tables list the members exposed by TDBXCatalogsIndex.

Public Constants

	Name	Description
	CatalogName (see page 10690)	This is CatalogName, a member of class TDBXCatalogsIndex.
	Last (see page 10690)	This is Last, a member of class TDBXCatalogsIndex.

56.1.2.2 TDBXCatalogsIndex Constants

The constants of the TDBXCatalogsIndex class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10690)	This is CatalogName, a member of class TDBXCatalogsIndex.
	Last (see page 10690)	This is Last, a member of class TDBXCatalogsIndex.

56.1.2.2.1 TDBXCatalogsIndex.CatalogName Constant

Pascal

```
const CatalogName = 0;
```

C++

```
CatalogName = 0;
```

Description

This is CatalogName, a member of class TDBXCatalogsIndex.

56.1.2.2.2 TDBXCatalogsIndex.Last Constant

Pascal

```
const Last = 0;
```

C++

```
Last = 0;
```

Description

This is Last, a member of class TDBXCatalogsIndex.

56.1.3 TDBXColumnConstraintsColumns Class

Internal use only

Class Hierarchy

Pascal

```
TDBXColumnConstraintsColumns = class;
```

C++

```
class TDBXColumnConstraintsColumns;
```

File

DBXMetaDataNames

Description

This class is used internally.

56.1.3.1 TDBXColumnConstraintsColumns Members

The following tables list the members exposed by TDBXColumnConstraintsColumns.

Public Constants

	Name	Description
	CatalogName (see page 10691)	This is CatalogName, a member of class TDBXColumnConstraintsColumns.
	ColumnName (see page 10691)	This is ColumnName, a member of class TDBXColumnConstraintsColumns.
	ConstraintName (see page 10692)	This is ConstraintName, a member of class TDBXColumnConstraintsColumns.
	SchemaName (see page 10692)	This is SchemaName, a member of class TDBXColumnConstraintsColumns.
	TableName (see page 10692)	This is TableName, a member of class TDBXColumnConstraintsColumns.

56.1.3.2 TDBXColumnConstraintsColumns Constants

The constants of the TDBXColumnConstraintsColumns class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10691)	This is CatalogName, a member of class TDBXColumnConstraintsColumns.
	ColumnName (see page 10691)	This is ColumnName, a member of class TDBXColumnConstraintsColumns.
	ConstraintName (see page 10692)	This is ConstraintName, a member of class TDBXColumnConstraintsColumns.
	SchemaName (see page 10692)	This is SchemaName, a member of class TDBXColumnConstraintsColumns.
	TableName (see page 10692)	This is TableName, a member of class TDBXColumnConstraintsColumns.

56.1.3.2.1 TDBXColumnConstraintsColumns.CatalogName Constant

Pascal

```
const CatalogName = 'CatalogName';
```

C++

```
CatalogName = 'CatalogName';
```

Description

This is CatalogName, a member of class TDBXColumnConstraintsColumns.

56.1.3.2.2 TDBXColumnConstraintsColumns.ColumnName Constant

Pascal

```
const ColumnName = 'ColumnName';
```

C++

```
ColumnName = 'ColumnName';
```

Description

This is ColumnName, a member of class TDBXColumnConstraintsColumns.

56.1.3.2.3 TDBXColumnConstraintsColumns.ConstraintName Constant

Pascal

```
const ConstraintName = 'ConstraintName';
```

C++

```
ConstraintName = 'ConstraintName';
```

Description

This is ConstraintName, a member of class TDBXColumnConstraintsColumns.

56.1.3.2.4 TDBXColumnConstraintsColumns.SchemaName Constant

Pascal

```
const SchemaName = 'SchemaName';
```

C++

```
SchemaName = 'SchemaName';
```

Description

This is SchemaName, a member of class TDBXColumnConstraintsColumns.

56.1.3.2.5 TDBXColumnConstraintsColumns.TableName Constant

Pascal

```
const TableName = 'TableName';
```

C++

```
TableName = 'TableName';
```

Description

This is TableName, a member of class TDBXColumnConstraintsColumns.

56.1.4 TDBXColumnConstraintsIndex Class

Internal use only

Class Hierarchy

Pascal

```
TDBXColumnConstraintsIndex = class;
```

C++

```
class TDBXColumnConstraintsIndex;
```

File

DBXMetaDataNames

Description

This class is used internally.

56.1.4.1 TDBXColumnConstraintsIndex Members

The following tables list the members exposed by TDBXColumnConstraintsIndex.

Public Constants

	Name	Description
	CatalogName (see page 10693)	This is CatalogName, a member of class TDBXColumnConstraintsIndex.
	ColumnName (see page 10693)	This is ColumnName, a member of class TDBXColumnConstraintsIndex.
	ConstraintName (see page 10694)	This is ConstraintName, a member of class TDBXColumnConstraintsIndex.
	Last (see page 10694)	This is Last, a member of class TDBXColumnConstraintsIndex.
	SchemaName (see page 10694)	This is SchemaName, a member of class TDBXColumnConstraintsIndex.
	TableName (see page 10694)	This is TableName, a member of class TDBXColumnConstraintsIndex.

56.1.4.2 TDBXColumnConstraintsIndex Constants

The constants of the TDBXColumnConstraintsIndex class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10693)	This is CatalogName, a member of class TDBXColumnConstraintsIndex.
	ColumnName (see page 10693)	This is ColumnName, a member of class TDBXColumnConstraintsIndex.
	ConstraintName (see page 10694)	This is ConstraintName, a member of class TDBXColumnConstraintsIndex.
	Last (see page 10694)	This is Last, a member of class TDBXColumnConstraintsIndex.
	SchemaName (see page 10694)	This is SchemaName, a member of class TDBXColumnConstraintsIndex.
	TableName (see page 10694)	This is TableName, a member of class TDBXColumnConstraintsIndex.

56.1.4.2.1 TDBXColumnConstraintsIndex.CatalogName Constant

Pascal

```
const CatalogName = 0;
```

C++

```
CatalogName = 0;
```

Description

This is CatalogName, a member of class TDBXColumnConstraintsIndex.

56.1.4.2.2 TDBXColumnConstraintsIndex.ColumnName Constant

Pascal

```
const ColumnName = 4;
```

C++

```
ColumnName = 4;
```

Description

This is ColumnName, a member of class TDBXColumnConstraintsIndex.

56.1.4.2.3 TDBXColumnConstraintsIndex.ConstraintName Constant

Pascal

```
const ConstraintName = 3;
```

C++

```
ConstraintName = 3;
```

Description

This is ConstraintName, a member of class TDBXColumnConstraintsIndex.

56.1.4.2.4 TDBXColumnConstraintsIndex.Last Constant

Pascal

```
const Last = 4;
```

C++

```
Last = 4;
```

Description

This is Last, a member of class TDBXColumnConstraintsIndex.

56.1.4.2.5 TDBXColumnConstraintsIndex.SchemaName Constant

Pascal

```
const SchemaName = 1;
```

C++

```
SchemaName = 1;
```

Description

This is SchemaName, a member of class TDBXColumnConstraintsIndex.

56.1.4.2.6 TDBXColumnConstraintsIndex.TableName Constant

Pascal

```
const TableName = 2;
```

C++

```
TableName = 2;
```

Description

This is TableName, a member of class TDBXColumnConstraintsIndex.

56.1.5 TDBXColumnsColumns Class

Accesses columns by name in the **Columns** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXColumnsColumns

Pascal

```
TDBXColumnsColumns = class;
```

C++

```
class TDBXColumnsColumns;
```

File

DBXMetaDataNames

Description

TDBXColumnsColumns is used to access columns by name in the **Columns** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetColumns when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Name	Data Type of Column Data	Description
CatalogName (see page 10697)	String	Catalog name.
SchemaName (see page 10700)	String	Schema name.
TableName (see page 10700)	String	Table name.
ColumnName (see page 10697)	String	Column name.
TypeName (see page 10700)	String	Database specific data type name.
Precision (see page 10700)	Int32	Precision (see page 10700) of this type. For a string data type, this is the maximum number of characters. For a decimal types, this is the maximum number digits including digits from the scale.
Scale (see page 10700)	Int32	Scale (see page 10700) of a decimal type.
Ordinal (see page 10699)	Int32	Position in the rows of the table starting with 1.
DefaultValue (see page 10698)	String	Default value.
IsNullable (see page 10698)	Boolean	True if accepts NULL values; false otherwise.

IsAutoIncrement (see page 10698)	Boolean	Is an autoincrement value.
MaxInline (see page 10699)	Int32	The maximal number of bytes to write into row. The rest of the data is handled as a BLOB. (Blackfish SQL only).
DbxDataType (see page 10697)	Int32	dbExpress data type.
IsFixedLength (see page 10698)	Boolean	True if holds fixed length data; false otherwise.
IsUnicode (see page 10699)	Boolean	Can hold Unicode characters.
IsLong (see page 10698)	Boolean	Is a BLOB data type.
IsUnsigned (see page 10699)	Boolean	Is an unsigned numeric type.

See Also

TDBXColumnsIndex (see page 10701)

TDBXMetaDataCommands (see page 10525)

56.1.5.1 TDBXColumnsColumns Members

The following tables list the members exposed by TDBXColumnsColumns.

Public Constants

	Name	Description
	CatalogName (see page 10697)	This is CatalogName, a member of class TDBXColumnsColumns.
	ColumnName (see page 10697)	This is ColumnName, a member of class TDBXColumnsColumns.
	DbxDataType (see page 10697)	This is DbxDataType, a member of class TDBXColumnsColumns.
	DefaultValue (see page 10698)	This is DefaultValue, a member of class TDBXColumnsColumns.
	IsAutoIncrement (see page 10698)	This is IsAutoIncrement, a member of class TDBXColumnsColumns.
	IsFixedLength (see page 10698)	This is IsFixedLength, a member of class TDBXColumnsColumns.
	IsLong (see page 10698)	This is IsLong, a member of class TDBXColumnsColumns.
	IsNullable (see page 10698)	This is IsNullable, a member of class TDBXColumnsColumns.
	IsUnicode (see page 10699)	This is IsUnicode, a member of class TDBXColumnsColumns.
	IsUnsigned (see page 10699)	This is IsUnsigned, a member of class TDBXColumnsColumns.
	MaxInline (see page 10699)	This is MaxInline, a member of class TDBXColumnsColumns.
	Ordinal (see page 10699)	This is Ordinal, a member of class TDBXColumnsColumns.
	Precision (see page 10700)	This is Precision, a member of class TDBXColumnsColumns.
	Scale (see page 10700)	This is Scale, a member of class TDBXColumnsColumns.
	SchemaName (see page 10700)	This is SchemaName, a member of class TDBXColumnsColumns.
	TableName (see page 10700)	This is TableName, a member of class TDBXColumnsColumns.
	TypeName (see page 10700)	This is TypeName, a member of class TDBXColumnsColumns.

56.1.5.2 TDBXColumnsColumns Constants

The constants of the TDBXColumnsColumns class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10697)	This is CatalogName, a member of class TDBXColumnsColumns.
	ColumnName (see page 10697)	This is ColumnName, a member of class TDBXColumnsColumns.
	DbxDataType (see page 10697)	This is DbxDataType, a member of class TDBXColumnsColumns.
	DefaultValue (see page 10698)	This is DefaultValue, a member of class TDBXColumnsColumns.
	IsAutoIncrement (see page 10698)	This is IsAutoIncrement, a member of class TDBXColumnsColumns.
	IsFixedLength (see page 10698)	This is IsFixedLength, a member of class TDBXColumnsColumns.
	IsLong (see page 10698)	This is IsLong, a member of class TDBXColumnsColumns.
	IsNullable (see page 10698)	This is IsNullable, a member of class TDBXColumnsColumns.
	IsUnicode (see page 10699)	This is IsUnicode, a member of class TDBXColumnsColumns.
	IsUnsigned (see page 10699)	This is IsUnsigned, a member of class TDBXColumnsColumns.
	MaxInline (see page 10699)	This is MaxInline, a member of class TDBXColumnsColumns.
	Ordinal (see page 10699)	This is Ordinal, a member of class TDBXColumnsColumns.
	Precision (see page 10700)	This is Precision, a member of class TDBXColumnsColumns.
	Scale (see page 10700)	This is Scale, a member of class TDBXColumnsColumns.
	SchemaName (see page 10700)	This is SchemaName, a member of class TDBXColumnsColumns.
	TableName (see page 10700)	This is TableName, a member of class TDBXColumnsColumns.
	TypeName (see page 10700)	This is TypeName, a member of class TDBXColumnsColumns.

56.1.5.2.1 TDBXColumnsColumns.CatalogName Constant

Pascal

```
const CatalogName = 'CatalogName';
```

C++

```
CatalogName = 'CatalogName';
```

Description

This is CatalogName, a member of class TDBXColumnsColumns.

56.1.5.2.2 TDBXColumnsColumns.ColumnName Constant

Pascal

```
const ColumnName = 'ColumnName';
```

C++

```
ColumnName = 'ColumnName';
```

Description

This is ColumnName, a member of class TDBXColumnsColumns.

56.1.5.2.3 TDBXColumnsColumns.DbxDataType Constant

Pascal

```
const DbxDataType = 'DbxDataType';
```

C++

```
DbxDataType = 'DbxDataType';
```

Description

This is DbxDataType, a member of class TDBXColumnsColumns.

56.1.5.2.4 TDBXColumnsColumns.DefaultValue Constant

Pascal

```
const DefaultValue = 'DefaultValue';
```

C++

```
DefaultValue = 'DefaultValue';
```

Description

This is DefaultValue, a member of class TDBXColumnsColumns.

56.1.5.2.5 TDBXColumnsColumns.IsAutoIncrement Constant

Pascal

```
const IsAutoIncrement = 'IsAutoIncrement';
```

C++

```
IsAutoIncrement = 'IsAutoIncrement';
```

Description

This is IsAutoIncrement, a member of class TDBXColumnsColumns.

56.1.5.2.6 TDBXColumnsColumns.IsFixedLength Constant

Pascal

```
const IsFixedLength = 'IsFixedLength';
```

C++

```
IsFixedLength = 'IsFixedLength';
```

Description

This is IsFixedLength, a member of class TDBXColumnsColumns.

56.1.5.2.7 TDBXColumnsColumns.IsLong Constant

Pascal

```
const IsLong = 'IsLong';
```

C++

```
IsLong = 'IsLong';
```

Description

This is IsLong, a member of class TDBXColumnsColumns.

56.1.5.2.8 TDBXColumnsColumns.IsNullable Constant

Pascal

```
const IsNullable = 'IsNullable';
```

C++

```
IsNullable = 'IsNullable';
```

Description

This is IsNullable, a member of class TDBXColumnsColumns.

56.1.5.2.9 TDBXColumnsColumns.IsUnicode Constant

Pascal

```
const IsUnicode = 'IsUnicode';
```

C++

```
IsUnicode = 'IsUnicode';
```

Description

This is IsUnicode, a member of class TDBXColumnsColumns.

56.1.5.2.10 TDBXColumnsColumns.IsUnsigned Constant

Pascal

```
const IsUnsigned = 'IsUnsigned';
```

C++

```
IsUnsigned = 'IsUnsigned';
```

Description

This is IsUnsigned, a member of class TDBXColumnsColumns.

56.1.5.2.11 TDBXColumnsColumns.MaxInline Constant

Pascal

```
const MaxInline = 'MaxInline';
```

C++

```
MaxInline = 'MaxInline';
```

Description

This is MaxInline, a member of class TDBXColumnsColumns.

56.1.5.2.12 TDBXColumnsColumns.Ordinal Constant

Pascal

```
const Ordinal = 'Ordinal';
```

C++

```
Ordinal = 'Ordinal';
```

Description

This is Ordinal, a member of class TDBXColumnsColumns.

56.1.5.2.13 TDBXColumnsColumns.Precision Constant

Pascal

```
const Precision = 'Precision';
```

C++

```
Precision = 'Precision';
```

Description

This is Precision, a member of class TDBXColumnsColumns.

56.1.5.2.14 TDBXColumnsColumns.Scale Constant

Pascal

```
const Scale = 'Scale';
```

C++

```
Scale = 'Scale';
```

Description

This is Scale, a member of class TDBXColumnsColumns.

56.1.5.2.15 TDBXColumnsColumns.SchemaName Constant

Pascal

```
const SchemaName = 'SchemaName';
```

C++

```
SchemaName = 'SchemaName';
```

Description

This is SchemaName, a member of class TDBXColumnsColumns.

56.1.5.2.16 TDBXColumnsColumns.TableName Constant

Pascal

```
const TableName = 'TableName';
```

C++

```
TableName = 'TableName';
```

Description

This is TableName, a member of class TDBXColumnsColumns.

56.1.5.2.17 TDBXColumnsColumns.TypeName Constant

Pascal

```
const TypeName = 'TypeName';
```

C++

```
TypeName = 'TypeName';
```

Description

This is TypeName, a member of class TDBXColumnsColumns.

56.1.6 TDBXColumnsIndex Class

Accesses columns by ordinal in the **Columns** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXColumnsIndex

Pascal

```
TDBXColumnsIndex = class;
```

C++

```
class TDBXColumnsIndex;
```

File

DBXMetaDataNames

Description

TDBXColumnsIndex is used to access columns by ordinal in the **Columns** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetColumns when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Ordinal (see page 10706) Name	Data Type of Column Data	Description
CatalogName (see page 10703)	String	Catalog name.
SchemaName (see page 10706)	String	Schema name.
TableName (see page 10707)	String	Table name.
ColumnName (see page 10703)	String	Column name.
TypeName (see page 10707)	String	Database specific data type name.
Precision (see page 10706)	Int32	Precision (see page 10706) of this type. For a string data type, this is the maximum number of characters. For a decimal types, this is the maximum number digits including digits from the scale.
Scale (see page 10706)	Int32	Scale (see page 10706) of a decimal type.
Ordinal (see page 10706)	Int32	Position in the rows of the table starting with 1.
DefaultValue (see page 10704)	String	Default value.

IsNullable (see page 10705)	Boolean	True if accepts NULL values; false otherwise.
IsAutoIncrement (see page 10704)	Boolean	Is an autoincrement value.
MaxInline (see page 10705)	Int32	The maximal number of bytes to write into row. The rest of the data is handled as a BLOB. (Blackfish SQL only).
DbxDataType (see page 10703)	Int32	dbExpress data type.
IsFixedLength (see page 10704)	Boolean	True if holds fixed length data; false otherwise.
IsUnicode (see page 10705)	Boolean	Can hold Unicode characters.
IsLong (see page 10704)	Boolean	Is a BLOB data type.
IsUnsigned (see page 10705)	Boolean	Is an unsigned numeric type.

See Also

TDBXColumnsColumns (see page 10695)

TDBXMetaDataCommands (see page 10525)

56.1.6.1 TDBXColumnsIndex Members

The following tables list the members exposed by TDBXColumnsIndex.

Public Constants

	Name	Description
	CatalogName (see page 10703)	This is CatalogName, a member of class TDBXColumnsIndex.
	ColumnName (see page 10703)	This is ColumnName, a member of class TDBXColumnsIndex.
	DbxDataType (see page 10703)	This is DbxDataType, a member of class TDBXColumnsIndex.
	DefaultValue (see page 10704)	This is DefaultValue, a member of class TDBXColumnsIndex.
	IsAutoIncrement (see page 10704)	This is IsAutoIncrement, a member of class TDBXColumnsIndex.
	IsFixedLength (see page 10704)	This is IsFixedLength, a member of class TDBXColumnsIndex.
	IsLong (see page 10704)	This is IsLong, a member of class TDBXColumnsIndex.
	IsNullable (see page 10705)	This is IsNullable, a member of class TDBXColumnsIndex.
	IsUnicode (see page 10705)	This is IsUnicode, a member of class TDBXColumnsIndex.
	IsUnsigned (see page 10705)	This is IsUnsigned, a member of class TDBXColumnsIndex.
	Last (see page 10705)	This is Last, a member of class TDBXColumnsIndex.
	MaxInline (see page 10705)	This is MaxInline, a member of class TDBXColumnsIndex.
	Ordinal (see page 10706)	This is Ordinal, a member of class TDBXColumnsIndex.
	Precision (see page 10706)	This is Precision, a member of class TDBXColumnsIndex.
	Scale (see page 10706)	This is Scale, a member of class TDBXColumnsIndex.
	SchemaName (see page 10706)	This is SchemaName, a member of class TDBXColumnsIndex.
	TableName (see page 10707)	This is TableName, a member of class TDBXColumnsIndex.
	TypeName (see page 10707)	This is TypeName, a member of class TDBXColumnsIndex.

56.1.6.2 TDBXColumnsIndex Constants

The constants of the TDBXColumnsIndex class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10703)	This is CatalogName, a member of class TDBXColumnsIndex.
	ColumnName (see page 10703)	This is ColumnName, a member of class TDBXColumnsIndex.
	DbxDataType (see page 10703)	This is DbxDataType, a member of class TDBXColumnsIndex.
	DefaultValue (see page 10704)	This is DefaultValue, a member of class TDBXColumnsIndex.
	IsAutoIncrement (see page 10704)	This is IsAutoIncrement, a member of class TDBXColumnsIndex.
	IsFixedLength (see page 10704)	This is IsFixedLength, a member of class TDBXColumnsIndex.
	IsLong (see page 10704)	This is IsLong, a member of class TDBXColumnsIndex.
	IsNullable (see page 10705)	This is IsNullable, a member of class TDBXColumnsIndex.
	IsUnicode (see page 10705)	This is IsUnicode, a member of class TDBXColumnsIndex.
	IsUnsigned (see page 10705)	This is IsUnsigned, a member of class TDBXColumnsIndex.
	Last (see page 10705)	This is Last, a member of class TDBXColumnsIndex.
	MaxInline (see page 10705)	This is MaxInline, a member of class TDBXColumnsIndex.
	Ordinal (see page 10706)	This is Ordinal, a member of class TDBXColumnsIndex.
	Precision (see page 10706)	This is Precision, a member of class TDBXColumnsIndex.
	Scale (see page 10706)	This is Scale, a member of class TDBXColumnsIndex.
	SchemaName (see page 10706)	This is SchemaName, a member of class TDBXColumnsIndex.
	TableName (see page 10707)	This is TableName, a member of class TDBXColumnsIndex.
	TypeName (see page 10707)	This is TypeName, a member of class TDBXColumnsIndex.

56.1.6.2.1 TDBXColumnsIndex.CatalogName Constant

Pascal

```
const CatalogName = 0;
```

C++

```
CatalogName = 0;
```

Description

This is CatalogName, a member of class TDBXColumnsIndex.

56.1.6.2.2 TDBXColumnsIndex.ColumnName Constant

Pascal

```
const ColumnName = 3;
```

C++

```
ColumnName = 3;
```

Description

This is ColumnName, a member of class TDBXColumnsIndex.

56.1.6.2.3 TDBXColumnsIndex.DbxDataType Constant

Pascal

```
const DbxDataType = 12;
```

C++

```
DbxDataType = 12;
```

Description

This is DbxDataType, a member of class TDBXColumnsIndex.

56.1.6.2.4 TDBXColumnsIndex.DefaultValue Constant

Pascal

```
const DefaultValue = 8;
```

C++

```
DefaultValue = 8;
```

Description

This is DefaultValue, a member of class TDBXColumnsIndex.

56.1.6.2.5 TDBXColumnsIndex.IsAutoIncrement Constant

Pascal

```
const IsAutoIncrement = 10;
```

C++

```
IsAutoIncrement = 10;
```

Description

This is IsAutoIncrement, a member of class TDBXColumnsIndex.

56.1.6.2.6 TDBXColumnsIndex.IsFixedLength Constant

Pascal

```
const IsFixedLength = 13;
```

C++

```
IsFixedLength = 13;
```

Description

This is IsFixedLength, a member of class TDBXColumnsIndex.

56.1.6.2.7 TDBXColumnsIndex.IsLong Constant

Pascal

```
const IsLong = 15;
```

C++

```
IsLong = 15;
```

Description

This is IsLong, a member of class TDBXColumnsIndex.

56.1.6.2.8 TDBXColumnsIndex.IsNullable Constant

Pascal

```
const IsNullable = 9;
```

C++

```
IsNullable = 9;
```

Description

This is IsNullable, a member of class TDBXColumnsIndex.

56.1.6.2.9 TDBXColumnsIndex.IsUnicode Constant

Pascal

```
const IsUnicode = 14;
```

C++

```
IsUnicode = 14;
```

Description

This is IsUnicode, a member of class TDBXColumnsIndex.

56.1.6.2.10 TDBXColumnsIndex.IsUnsigned Constant

Pascal

```
const IsUnsigned = 16;
```

C++

```
IsUnsigned = 16;
```

Description

This is IsUnsigned, a member of class TDBXColumnsIndex.

56.1.6.2.11 TDBXColumnsIndex.Last Constant

Pascal

```
const Last = 16;
```

C++

```
Last = 16;
```

Description

This is Last, a member of class TDBXColumnsIndex.

56.1.6.2.12 TDBXColumnsIndex.MaxInline Constant

Pascal

```
const MaxInline = 11;
```

C++

```
MaxInline = 11;
```

Description

This is MaxInline, a member of class TDBXColumnsIndex.

56.1.6.2.13 TDBXColumnsIndex.Ordinal Constant

Pascal

```
const Ordinal = 7;
```

C++

```
Ordinal = 7;
```

Description

This is Ordinal, a member of class TDBXColumnsIndex.

56.1.6.2.14 TDBXColumnsIndex.Precision Constant

Pascal

```
const Precision = 5;
```

C++

```
Precision = 5;
```

Description

This is Precision, a member of class TDBXColumnsIndex.

56.1.6.2.15 TDBXColumnsIndex.Scale Constant

Pascal

```
const Scale = 6;
```

C++

```
Scale = 6;
```

Description

This is Scale, a member of class TDBXColumnsIndex.

56.1.6.2.16 TDBXColumnsIndex.SchemaName Constant

Pascal

```
const SchemaName = 1;
```

C++

```
SchemaName = 1;
```

Description

This is SchemaName, a member of class TDBXColumnsIndex.

56.1.6.2.17 TDBXColumnsIndex.TableName Constant

Pascal

```
const TableName = 2;
```

C++

```
TableName = 2;
```

Description

This is TableName, a member of class TDBXColumnsIndex.

56.1.6.2.18 TDBXColumnsIndex.TypeName Constant

Pascal

```
const TypeName = 4;
```

C++

```
TypeName = 4;
```

Description

This is TypeName, a member of class TDBXColumnsIndex.

56.1.7 TDBXDataTypesColumns Class

Accesses columns by name in the **DataTypes** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXDataTypesColumns

Pascal

```
TDBXDataTypesColumns = class;
```

C++

```
class TDBXDataTypesColumns;
```

File

DBXMetaDataNames

Description

TDBXDataTypesColumns is used to access columns by name in the **DataTypes** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetDataTypes when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Name	Data Type of Column Data	Description
TypeName (see page 10716)	String	Database specific data type name.

DbxDataType (see page 10711)	Int32	dbExpress data type.
ColumnSize (see page 10710)	Int64	Length of this type. For a string data type, this is the maximum number of characters. For a floating point number, this is the number of significant bits in the mantissa. For other numeric types, this is the maximum number of significant digits. For date, time, and timestamp types, this is the length of the string representation.
CreateFormat (see page 10710)	String	Create format as used in CREATE TABLE.
CreateParameters (see page 10711)	String	Parameters for the create format.
DataType (see page 10711)	String	Name of language type representing this data type.
IsAutoIncrementable (see page 10711)	Boolean	Can be used for an autoincrement column.
IsBestMatch (see page 10711)	Boolean	Is best match for language type.
IsCaseSensitive (see page 10712)	Boolean	Is case sensitive.
IsFixedLength (see page 10712)	Boolean	Is fixed length.
IsFixedPrecisionScale (see page 10712)	Boolean	Is an exact numeric.
IsLong (see page 10713)	Boolean	Is a BLOB type.
IsNullable (see page 10713)	Boolean	Can be assigned a value of NULL.
IsSearchable (see page 10713)	Boolean	Can be searched in the SQL database.
IsSearchableWithLike (see page 10713)	Boolean	Can be searched with the SQL LIKE operator.
IsUnsigned (see page 10714)	Boolean	Is an unsigned numeric.
MaximumScale (see page 10715)	Int16	Maximum possible scale value.
MinimumScale (see page 10715)	Int16	Minimum possible scale value.
IsConcurrencyType (see page 10712)	Boolean	Has special update semantics.
MaximumVersion (see page 10715)	String	Last database product version where this type was available.
MinimumVersion (see page 10715)	String	First database product version where this type was available.
IsLiteralSupported (see page 10713)	Boolean	Can be represented as a SQL literal.
LiteralPrefix (see page 10714)	String	Prefix of SQL literal.

LiteralSuffix (see page 10714)	String	Suffix of SQL literal.
IsUnicode (see page 10714)	Boolean	Can hold Unicode character data.
ProviderDbType (see page 10715)	Int32	Same as the DbxDataType (see page 10711) column. Reserved for .NET applications.

See Also

TDBXDataTypesIndex (see page 10716)

TDBXMetaDataCommands (see page 10525)

56.1.7.1 TDBXDataTypesColumns Members

The following tables list the members exposed by TDBXDataTypesColumns.

Public Constants

	Name	Description
	ColumnSize (see page 10710)	This is ColumnSize, a member of class TDBXDataTypesColumns.
	CreateFormat (see page 10710)	This is CreateFormat, a member of class TDBXDataTypesColumns.
	CreateParameters (see page 10711)	This is CreateParameters, a member of class TDBXDataTypesColumns.
	DataType (see page 10711)	This is DataType, a member of class TDBXDataTypesColumns.
	DbxDataType (see page 10711)	This is DbxDataType, a member of class TDBXDataTypesColumns.
	IsAutoIncrementable (see page 10711)	This is IsAutoIncrementable, a member of class TDBXDataTypesColumns.
	IsBestMatch (see page 10711)	This is IsBestMatch, a member of class TDBXDataTypesColumns.
	IsCaseSensitive (see page 10712)	This is IsCaseSensitive, a member of class TDBXDataTypesColumns.
	IsConcurrencyType (see page 10712)	This is IsConcurrencyType, a member of class TDBXDataTypesColumns.
	IsFixedLength (see page 10712)	This is IsFixedLength, a member of class TDBXDataTypesColumns.
	IsFixedPrecisionScale (see page 10712)	This is IsFixedPrecisionScale, a member of class TDBXDataTypesColumns.
	IsLiteralSupported (see page 10713)	This is IsLiteralSupported, a member of class TDBXDataTypesColumns.
	IsLong (see page 10713)	This is IsLong, a member of class TDBXDataTypesColumns.
	IsNullable (see page 10713)	This is IsNullable, a member of class TDBXDataTypesColumns.
	IsSearchable (see page 10713)	This is IsSearchable, a member of class TDBXDataTypesColumns.
	IsSearchableWithLike (see page 10713)	This is IsSearchableWithLike, a member of class TDBXDataTypesColumns.
	IsUnicode (see page 10714)	This is IsUnicode, a member of class TDBXDataTypesColumns.
	IsUnsigned (see page 10714)	This is IsUnsigned, a member of class TDBXDataTypesColumns.
	LiteralPrefix (see page 10714)	This is LiteralPrefix, a member of class TDBXDataTypesColumns.
	LiteralSuffix (see page 10714)	This is LiteralSuffix, a member of class TDBXDataTypesColumns.
	MaximumScale (see page 10715)	This is MaximumScale, a member of class TDBXDataTypesColumns.
	MaximumVersion (see page 10715)	This is MaximumVersion, a member of class TDBXDataTypesColumns.
	MinimumScale (see page 10715)	This is MinimumScale, a member of class TDBXDataTypesColumns.
	MinimumVersion (see page 10715)	This is MinimumVersion, a member of class TDBXDataTypesColumns.
	ProviderDbType (see page 10715)	This is ProviderDbType, a member of class TDBXDataTypesColumns.
	TypeName (see page 10716)	This is TypeName, a member of class TDBXDataTypesColumns.

56.1.7.2 TDBXDataTypesColumns Constants

The constants of the TDBXDataTypesColumns class are listed here.

Public Constants

	Name	Description
	ColumnSize (see page 10710)	This is ColumnSize, a member of class TDBXDataTypesColumns.
	CreateFormat (see page 10710)	This is CreateFormat, a member of class TDBXDataTypesColumns.
	CreateParameters (see page 10711)	This is CreateParameters, a member of class TDBXDataTypesColumns.
	DataType (see page 10711)	This is DataType, a member of class TDBXDataTypesColumns.
	DbxDataType (see page 10711)	This is DbxDataType, a member of class TDBXDataTypesColumns.
	IsAutoIncrementable (see page 10711)	This is IsAutoIncrementable, a member of class TDBXDataTypesColumns.
	IsBestMatch (see page 10711)	This is IsBestMatch, a member of class TDBXDataTypesColumns.
	IsCaseSensitive (see page 10712)	This is IsCaseSensitive, a member of class TDBXDataTypesColumns.
	IsConcurrencyType (see page 10712)	This is IsConcurrencyType, a member of class TDBXDataTypesColumns.
	IsFixedLength (see page 10712)	This is IsFixedLength, a member of class TDBXDataTypesColumns.
	IsFixedPrecisionScale (see page 10712)	This is IsFixedPrecisionScale, a member of class TDBXDataTypesColumns.
	IsLiteralSupported (see page 10713)	This is IsLiteralSupported, a member of class TDBXDataTypesColumns.
	IsLong (see page 10713)	This is IsLong, a member of class TDBXDataTypesColumns.
	IsNullable (see page 10713)	This is IsNullable, a member of class TDBXDataTypesColumns.
	IsSearchable (see page 10713)	This is IsSearchable, a member of class TDBXDataTypesColumns.
	IsSearchableWithLike (see page 10713)	This is IsSearchableWithLike, a member of class TDBXDataTypesColumns.
	IsUnicode (see page 10714)	This is IsUnicode, a member of class TDBXDataTypesColumns.
	IsUnsigned (see page 10714)	This is IsUnsigned, a member of class TDBXDataTypesColumns.
	LiteralPrefix (see page 10714)	This is LiteralPrefix, a member of class TDBXDataTypesColumns.
	LiteralSuffix (see page 10714)	This is LiteralSuffix, a member of class TDBXDataTypesColumns.
	MaximumScale (see page 10715)	This is MaximumScale, a member of class TDBXDataTypesColumns.
	MaximumVersion (see page 10715)	This is MaximumVersion, a member of class TDBXDataTypesColumns.
	MinimumScale (see page 10715)	This is MinimumScale, a member of class TDBXDataTypesColumns.
	MinimumVersion (see page 10715)	This is MinimumVersion, a member of class TDBXDataTypesColumns.
	ProviderDbType (see page 10715)	This is ProviderDbType, a member of class TDBXDataTypesColumns.
	TypeName (see page 10716)	This is TypeName, a member of class TDBXDataTypesColumns.

56.1.7.2.1 TDBXDataTypesColumns.ColumnSize Constant

Pascal

```
const ColumnSize = 'ColumnSize';
```

C++

```
ColumnSize = 'ColumnSize';
```

Description

This is ColumnSize, a member of class TDBXDataTypesColumns.

56.1.7.2.2 TDBXDataTypesColumns.CreateFormat Constant

Pascal

```
const CreateFormat = 'CreateFormat';
```

C++

```
CreateFormat = 'CreateFormat';
```

Description

This is CreateFormat, a member of class TDBXDataTypesColumns.

56.1.7.2.3 TDBXDataTypesColumns.CreateParameters Constant

Pascal

```
const CreateParameters = 'CreateParameters';
```

C++

```
CreateParameters = 'CreateParameters';
```

Description

This is CreateParameters, a member of class TDBXDataTypesColumns.

56.1.7.2.4 TDBXDataTypesColumns.DataType Constant

Pascal

```
const DataType = 'DataType';
```

C++

```
DataType = 'DataType';
```

Description

This is DataType, a member of class TDBXDataTypesColumns.

56.1.7.2.5 TDBXDataTypesColumns.DbxDataType Constant

Pascal

```
const DbxDataType = 'DbxDataType';
```

C++

```
DbxDataType = 'DbxDataType';
```

Description

This is DbxDataType, a member of class TDBXDataTypesColumns.

56.1.7.2.6 TDBXDataTypesColumns.IsAutoIncrementable Constant

Pascal

```
const IsAutoIncrementable = 'IsAutoIncrementable';
```

C++

```
IsAutoIncrementable = 'IsAutoIncrementable';
```

Description

This is IsAutoIncrementable, a member of class TDBXDataTypesColumns.

56.1.7.2.7 TDBXDataTypesColumns.IsBestMatch Constant

Pascal

```
const IsBestMatch = 'IsBestMatch';
```

C++

```
IsBestMatch = 'IsBestMatch';
```

Description

This is IsBestMatch, a member of class TDBXDataTypesColumns.

56.1.7.2.8 TDBXDataTypesColumns.IsCaseSensitive Constant

Pascal

```
const IsCaseSensitive = 'IsCaseSensitive';
```

C++

```
IsCaseSensitive = 'IsCaseSensitive';
```

Description

This is IsCaseSensitive, a member of class TDBXDataTypesColumns.

56.1.7.2.9 TDBXDataTypesColumns.IsConcurrencyType Constant

Pascal

```
const IsConcurrencyType = 'IsConcurrencyType';
```

C++

```
IsConcurrencyType = 'IsConcurrencyType';
```

Description

This is IsConcurrencyType, a member of class TDBXDataTypesColumns.

56.1.7.2.10 TDBXDataTypesColumns.IsFixedLength Constant

Pascal

```
const IsFixedLength = 'IsFixedLength';
```

C++

```
IsFixedLength = 'IsFixedLength';
```

Description

This is IsFixedLength, a member of class TDBXDataTypesColumns.

56.1.7.2.11 TDBXDataTypesColumns.IsFixedPrecisionScale Constant

Pascal

```
const IsFixedPrecisionScale = 'IsFixedPrecisionScale';
```

C++

```
IsFixedPrecisionScale = 'IsFixedPrecisionScale';
```

Description

This is IsFixedPrecisionScale, a member of class TDBXDataTypesColumns.

56.1.7.2.12 TDBXDataTypesColumns.IsLiteralSupported Constant

Pascal

```
const IsLiteralSupported = 'IsLiteralSupported';
```

C++

```
IsLiteralSupported = 'IsLiteralSupported';
```

Description

This is IsLiteralSupported, a member of class TDBXDataTypesColumns.

56.1.7.2.13 TDBXDataTypesColumns.IsLong Constant

Pascal

```
const IsLong = 'IsLong';
```

C++

```
IsLong = 'IsLong';
```

Description

This is IsLong, a member of class TDBXDataTypesColumns.

56.1.7.2.14 TDBXDataTypesColumns.IsNullable Constant

Pascal

```
const IsNullable = 'IsNullable';
```

C++

```
IsNullable = 'IsNullable';
```

Description

This is IsNullable, a member of class TDBXDataTypesColumns.

56.1.7.2.15 TDBXDataTypesColumns.IsSearchable Constant

Pascal

```
const IsSearchable = 'IsSearchable';
```

C++

```
IsSearchable = 'IsSearchable';
```

Description

This is IsSearchable, a member of class TDBXDataTypesColumns.

56.1.7.2.16 TDBXDataTypesColumns.IsSearchableWithLike Constant

Pascal

```
const IsSearchableWithLike = 'IsSearchableWithLike';
```

C++

```
IsSearchableWithLike = 'IsSearchableWithLike';
```

Description

This is IsSearchableWithLike, a member of class TDBXDataTypesColumns.

56.1.7.2.17 TDBXDataTypesColumns.IsUnicode Constant

Pascal

```
const IsUnicode = 'IsUnicode';
```

C++

```
IsUnicode = 'IsUnicode';
```

Description

This is IsUnicode, a member of class TDBXDataTypesColumns.

56.1.7.2.18 TDBXDataTypesColumns.IsUnsigned Constant

Pascal

```
const IsUnsigned = 'IsUnsigned';
```

C++

```
IsUnsigned = 'IsUnsigned';
```

Description

This is IsUnsigned, a member of class TDBXDataTypesColumns.

56.1.7.2.19 TDBXDataTypesColumns.LiteralPrefix Constant

Pascal

```
const LiteralPrefix = 'LiteralPrefix';
```

C++

```
LiteralPrefix = 'LiteralPrefix';
```

Description

This is LiteralPrefix, a member of class TDBXDataTypesColumns.

56.1.7.2.20 TDBXDataTypesColumns.LiteralSuffix Constant

Pascal

```
const LiteralSuffix = 'LiteralSuffix';
```

C++

```
LiteralSuffix = 'LiteralSuffix';
```

Description

This is LiteralSuffix, a member of class TDBXDataTypesColumns.

56.1.7.2.21 TDBXDataTypesColumns.MaximumScale Constant

Pascal

```
const MaximumScale = 'MaximumScale';
```

C++

```
MaximumScale = 'MaximumScale';
```

Description

This is MaximumScale, a member of class TDBXDataTypesColumns.

56.1.7.2.22 TDBXDataTypesColumns.MaximumVersion Constant

Pascal

```
const MaximumVersion = 'MaximumVersion';
```

C++

```
MaximumVersion = 'MaximumVersion';
```

Description

This is MaximumVersion, a member of class TDBXDataTypesColumns.

56.1.7.2.23 TDBXDataTypesColumns.MinimumScale Constant

Pascal

```
const MinimumScale = 'MinimumScale';
```

C++

```
MinimumScale = 'MinimumScale';
```

Description

This is MinimumScale, a member of class TDBXDataTypesColumns.

56.1.7.2.24 TDBXDataTypesColumns.MinimumVersion Constant

Pascal

```
const MinimumVersion = 'MinimumVersion';
```

C++

```
MinimumVersion = 'MinimumVersion';
```

Description

This is MinimumVersion, a member of class TDBXDataTypesColumns.

56.1.7.2.25 TDBXDataTypesColumns.ProviderDbType Constant

Pascal

```
const ProviderDbType = 'ProviderDbType';
```

C++

```
ProviderDbType = 'ProviderDbType';
```

Description

This is ProviderDbType, a member of class TDBXDataTypesColumns.

56.1.7.2.26 TDBXDataTypesColumns.TypeName Constant

Pascal

```
const TypeName = 'TypeName';
```

C++

```
TypeName = 'TypeName';
```

Description

This is TypeName, a member of class TDBXDataTypesColumns.

56.1.8 TDBXDataTypesIndex Class

Accesses columns by ordinal in the **DataTypes** metadata.

Class Hierarchy

Pascal

```
TDBXDataTypesIndex = class;
```

C++

```
class TDBXDataTypesIndex;
```

File

DBXMetaDataNames

Description

TDBXDataTypesIndex is used to access columns by ordinal in the **DataTypes** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetDataTypes when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Ordinal Name	Data Type of Column Data	Description
TypeName (see page 10725)	String	Database specific data type name.
DbxDataType (see page 10720)	Int32	dbExpress data type.
ColumnSize (see page 10719)	Int64	Length of this type. For a string data type, this is the maximum number of characters. For a floating point number, this is the number of significant bits in the mantissa. For other numeric types, this is the maximum number of significant digits. For date, time, and timestamp types, this is the length of the string representation.

CreateFormat (see page 10719)	String	Create format as used in CREATE TABLE.
CreateParameters (see page 10719)	String	Parameters for the create format.
DataType (see page 10720)	String	Name of language type representing this data type.
IsAutoIncrementable (see page 10720)	Boolean	Can be used for an autoincrement column.
IsBestMatch (see page 10720)	Boolean	Is best match for language type.
IsCaseSensitive (see page 10720)	Boolean	Is case sensitive.
IsFixedLength (see page 10721)	Boolean	Is fixed length.
IsFixedPrecisionScale (see page 10721)	Boolean	Is an exact numeric.
IsLong (see page 10722)	Boolean	Is a BLOB type.
IsNullable (see page 10722)	Boolean	Can be assigned a value of NULL.
IsSearchable (see page 10722)	Boolean	Can be searched in the SQL database.
IsSearchableWithLike (see page 10722)	Boolean	Can be searched with the SQL LIKE operator.
IsUnsigned (see page 10723)	Boolean	Is an unsigned numeric.
MaximumScale (see page 10724)	Int16	Maximum possible scale value.
MinimumScale (see page 10724)	Int16	Minimum possible scale value.
IsConcurrencyType (see page 10721)	Boolean	Has special update semantics.
MaximumVersion (see page 10724)	String	Last (see page 10723) database product version where this type was available.
MinimumVersion (see page 10724)	String	First database product version where this type was available.
IsLiteralSupported (see page 10721)	Boolean	Can be represented as a SQL literal.
LiteralPrefix (see page 10723)	String	Prefix of SQL literal.
LiteralSuffix (see page 10723)	String	Suffix of SQL literal.
IsUnicode (see page 10722)	Boolean	Can hold Unicode character data.
ProviderDbType (see page 10724)	Int32	Same as the DbxDataType (see page 10720) column. Reserved for .NET applications.

See Also

TDBXDataTypesColumns (see page 10707)

TDBXMetaDataCommands (see page 10525)

56.1.8.1 TDBXDataTypesIndex Members

The following tables list the members exposed by TDBXDataTypesIndex.

Public Constants

	Name	Description
	ColumnSize (see page 10719)	This is ColumnSize, a member of class TDBXDataTypesIndex.
	CreateFormat (see page 10719)	This is CreateFormat, a member of class TDBXDataTypesIndex.
	CreateParameters (see page 10719)	This is CreateParameters, a member of class TDBXDataTypesIndex.
	DataType (see page 10720)	This is DataType, a member of class TDBXDataTypesIndex.
	DbxDataType (see page 10720)	This is DbxDataType, a member of class TDBXDataTypesIndex.
	IsAutoIncrementable (see page 10720)	This is IsAutoIncrementable, a member of class TDBXDataTypesIndex.
	IsBestMatch (see page 10720)	This is IsBestMatch, a member of class TDBXDataTypesIndex.
	IsCaseSensitive (see page 10720)	This is IsCaseSensitive, a member of class TDBXDataTypesIndex.
	IsConcurrencyType (see page 10721)	This is IsConcurrencyType, a member of class TDBXDataTypesIndex.
	IsFixedLength (see page 10721)	This is IsFixedLength, a member of class TDBXDataTypesIndex.
	IsFixedPrecisionScale (see page 10721)	This is IsFixedPrecisionScale, a member of class TDBXDataTypesIndex.
	IsLiteralSupported (see page 10721)	This is IsLiteralSupported, a member of class TDBXDataTypesIndex.
	IsLong (see page 10722)	This is IsLong, a member of class TDBXDataTypesIndex.
	IsNullable (see page 10722)	This is IsNullable, a member of class TDBXDataTypesIndex.
	IsSearchable (see page 10722)	This is IsSearchable, a member of class TDBXDataTypesIndex.
	IsSearchableWithLike (see page 10722)	This is IsSearchableWithLike, a member of class TDBXDataTypesIndex.
	IsUnicode (see page 10722)	This is IsUnicode, a member of class TDBXDataTypesIndex.
	IsUnsigned (see page 10723)	This is IsUnsigned, a member of class TDBXDataTypesIndex.
	Last (see page 10723)	This is Last, a member of class TDBXDataTypesIndex.
	LiteralPrefix (see page 10723)	This is LiteralPrefix, a member of class TDBXDataTypesIndex.
	LiteralSuffix (see page 10723)	This is LiteralSuffix, a member of class TDBXDataTypesIndex.
	MaximumScale (see page 10724)	This is MaximumScale, a member of class TDBXDataTypesIndex.
	MaximumVersion (see page 10724)	This is MaximumVersion, a member of class TDBXDataTypesIndex.
	MinimumScale (see page 10724)	This is MinimumScale, a member of class TDBXDataTypesIndex.
	MinimumVersion (see page 10724)	This is MinimumVersion, a member of class TDBXDataTypesIndex.
	ProviderDbType (see page 10724)	This is ProviderDbType, a member of class TDBXDataTypesIndex.
	TypeName (see page 10725)	This is TypeName, a member of class TDBXDataTypesIndex.

56.1.8.2 TDBXDataTypesIndex Constants

The constants of the TDBXDataTypesIndex class are listed here.

Public Constants

	Name	Description
	ColumnSize (see page 10719)	This is ColumnSize, a member of class TDBXDataTypesIndex.
	CreateFormat (see page 10719)	This is CreateFormat, a member of class TDBXDataTypesIndex.
	CreateParameters (see page 10719)	This is CreateParameters, a member of class TDBXDataTypesIndex.
	DataType (see page 10720)	This is DataType, a member of class TDBXDataTypesIndex.
	DbxDataType (see page 10720)	This is DbxDataType, a member of class TDBXDataTypesIndex.
	IsAutoIncrementable (see page 10720)	This is IsAutoIncrementable, a member of class TDBXDataTypesIndex.
	IsBestMatch (see page 10720)	This is IsBestMatch, a member of class TDBXDataTypesIndex.

	IsCaseSensitive (see page 10720)	This is IsCaseSensitive, a member of class TDBXDataTypesIndex.
	IsConcurrencyType (see page 10721)	This is IsConcurrencyType, a member of class TDBXDataTypesIndex.
	IsFixedLength (see page 10721)	This is IsFixedLength, a member of class TDBXDataTypesIndex.
	IsFixedPrecisionScale (see page 10721)	This is IsFixedPrecisionScale, a member of class TDBXDataTypesIndex.
	IsLiteralSupported (see page 10721)	This is IsLiteralSupported, a member of class TDBXDataTypesIndex.
	IsLong (see page 10722)	This is IsLong, a member of class TDBXDataTypesIndex.
	IsNullable (see page 10722)	This is IsNullable, a member of class TDBXDataTypesIndex.
	IsSearchable (see page 10722)	This is IsSearchable, a member of class TDBXDataTypesIndex.
	IsSearchableWithLike (see page 10722)	This is IsSearchableWithLike, a member of class TDBXDataTypesIndex.
	IsUnicode (see page 10722)	This is IsUnicode, a member of class TDBXDataTypesIndex.
	IsUnsigned (see page 10723)	This is IsUnsigned, a member of class TDBXDataTypesIndex.
	Last (see page 10723)	This is Last, a member of class TDBXDataTypesIndex.
	LiteralPrefix (see page 10723)	This is LiteralPrefix, a member of class TDBXDataTypesIndex.
	LiteralSuffix (see page 10723)	This is LiteralSuffix, a member of class TDBXDataTypesIndex.
	MaximumScale (see page 10724)	This is MaximumScale, a member of class TDBXDataTypesIndex.
	MaximumVersion (see page 10724)	This is MaximumVersion, a member of class TDBXDataTypesIndex.
	MinimumScale (see page 10724)	This is MinimumScale, a member of class TDBXDataTypesIndex.
	MinimumVersion (see page 10724)	This is MinimumVersion, a member of class TDBXDataTypesIndex.
	ProviderDbType (see page 10724)	This is ProviderDbType, a member of class TDBXDataTypesIndex.
	TypeName (see page 10725)	This is TypeName, a member of class TDBXDataTypesIndex.

56.1.8.2.1 TDBXDataTypesIndex.ColumnSize Constant

Pascal

```
const ColumnSize = 2;
```

C++

```
ColumnSize = 2;
```

Description

This is ColumnSize, a member of class TDBXDataTypesIndex.

56.1.8.2.2 TDBXDataTypesIndex.CreateFormat Constant

Pascal

```
const CreateFormat = 3;
```

C++

```
CreateFormat = 3;
```

Description

This is CreateFormat, a member of class TDBXDataTypesIndex.

56.1.8.2.3 TDBXDataTypesIndex.CreateParameters Constant

Pascal

```
const CreateParameters = 4;
```

C++

```
CreateParameters = 4;
```

Description

This is CreateParameters, a member of class TDBXDataTypesIndex.

56.1.8.2.4 TDBXDataTypesIndex.DataType Constant

Pascal

```
const DataType = 5;
```

C++

```
DataType = 5;
```

Description

This is DataType, a member of class TDBXDataTypesIndex.

56.1.8.2.5 TDBXDataTypesIndex.DbxDataType Constant

Pascal

```
const DbxDataType = 1;
```

C++

```
DbxDataType = 1;
```

Description

This is DbxDataType, a member of class TDBXDataTypesIndex.

56.1.8.2.6 TDBXDataTypesIndex.IsAutoIncrementable Constant

Pascal

```
const IsAutoIncrementable = 6;
```

C++

```
IsAutoIncrementable = 6;
```

Description

This is IsAutoIncrementable, a member of class TDBXDataTypesIndex.

56.1.8.2.7 TDBXDataTypesIndex.IsBestMatch Constant

Pascal

```
const IsBestMatch = 7;
```

C++

```
IsBestMatch = 7;
```

Description

This is IsBestMatch, a member of class TDBXDataTypesIndex.

56.1.8.2.8 TDBXDataTypesIndex.IsCaseSensitive Constant

Pascal

```
const IsCaseSensitive = 8;
```

C++

```
IsCaseSensitive = 8;
```

Description

This is IsCaseSensitive, a member of class TDBXDataTypesIndex.

56.1.8.2.9 TDBXDataTypesIndex.IsConcurrencyType Constant

Pascal

```
const IsConcurrencyType = 18;
```

C++

```
IsConcurrencyType = 18;
```

Description

This is IsConcurrencyType, a member of class TDBXDataTypesIndex.

56.1.8.2.10 TDBXDataTypesIndex.IsFixedLength Constant

Pascal

```
const IsFixedLength = 9;
```

C++

```
IsFixedLength = 9;
```

Description

This is IsFixedLength, a member of class TDBXDataTypesIndex.

56.1.8.2.11 TDBXDataTypesIndex.IsFixedPrecisionScale Constant

Pascal

```
const IsFixedPrecisionScale = 10;
```

C++

```
IsFixedPrecisionScale = 10;
```

Description

This is IsFixedPrecisionScale, a member of class TDBXDataTypesIndex.

56.1.8.2.12 TDBXDataTypesIndex.IsLiteralSupported Constant

Pascal

```
const IsLiteralSupported = 21;
```

C++

```
IsLiteralSupported = 21;
```

Description

This is IsLiteralSupported, a member of class TDBXDataTypesIndex.

56.1.8.2.13 TDBXDataTypesIndex.IsLong Constant

Pascal

```
const IsLong = 11;
```

C++

```
IsLong = 11;
```

Description

This is IsLong, a member of class TDBXDataTypesIndex.

56.1.8.2.14 TDBXDataTypesIndex.IsNullable Constant

Pascal

```
const IsNullable = 12;
```

C++

```
IsNullable = 12;
```

Description

This is IsNullable, a member of class TDBXDataTypesIndex.

56.1.8.2.15 TDBXDataTypesIndex.IsSearchable Constant

Pascal

```
const IsSearchable = 13;
```

C++

```
IsSearchable = 13;
```

Description

This is IsSearchable, a member of class TDBXDataTypesIndex.

56.1.8.2.16 TDBXDataTypesIndex.IsSearchableWithLike Constant

Pascal

```
const IsSearchableWithLike = 14;
```

C++

```
IsSearchableWithLike = 14;
```

Description

This is IsSearchableWithLike, a member of class TDBXDataTypesIndex.

56.1.8.2.17 TDBXDataTypesIndex.IsUnicode Constant

Pascal

```
const IsUnicode = 24;
```

C++

```
IsUnicode = 24;
```

Description

This is IsUnicode, a member of class TDBXDataTypesIndex.

56.1.8.2.18 TDBXDataTypesIndex.IsUnsigned Constant

Pascal

```
const IsUnsigned = 15;
```

C++

```
IsUnsigned = 15;
```

Description

This is IsUnsigned, a member of class TDBXDataTypesIndex.

56.1.8.2.19 TDBXDataTypesIndex.Last Constant

Pascal

```
const Last = 25;
```

C++

```
Last = 25;
```

Description

This is Last, a member of class TDBXDataTypesIndex.

56.1.8.2.20 TDBXDataTypesIndex.LiteralPrefix Constant

Pascal

```
const LiteralPrefix = 22;
```

C++

```
LiteralPrefix = 22;
```

Description

This is LiteralPrefix, a member of class TDBXDataTypesIndex.

56.1.8.2.21 TDBXDataTypesIndex.LiteralSuffix Constant

Pascal

```
const LiteralSuffix = 23;
```

C++

```
LiteralSuffix = 23;
```

Description

This is LiteralSuffix, a member of class TDBXDataTypesIndex.

56.1.8.2.22 TDBXDataTypesIndex.MaximumScale Constant

Pascal

```
const MaximumScale = 16;
```

C++

```
MaximumScale = 16;
```

Description

This is MaximumScale, a member of class TDBXDataTypesIndex.

56.1.8.2.23 TDBXDataTypesIndex.MaximumVersion Constant

Pascal

```
const MaximumVersion = 19;
```

C++

```
MaximumVersion = 19;
```

Description

This is MaximumVersion, a member of class TDBXDataTypesIndex.

56.1.8.2.24 TDBXDataTypesIndex.MinimumScale Constant

Pascal

```
const MinimumScale = 17;
```

C++

```
MinimumScale = 17;
```

Description

This is MinimumScale, a member of class TDBXDataTypesIndex.

56.1.8.2.25 TDBXDataTypesIndex.MinimumVersion Constant

Pascal

```
const MinimumVersion = 20;
```

C++

```
MinimumVersion = 20;
```

Description

This is MinimumVersion, a member of class TDBXDataTypesIndex.

56.1.8.2.26 TDBXDataTypesIndex.ProviderDbType Constant

Pascal

```
const ProviderDbType = 25;
```

C++

```
ProviderDbType = 25;
```

Description

This is ProviderDbType, a member of class TDBXDataTypesIndex.

56.1.8.2.27 TDBXDataTypesIndex.TypeName Constant

Pascal

```
const TypeName = 0;
```

C++

```
TypeName = 0;
```

Description

This is TypeName, a member of class TDBXDataTypesIndex.

56.1.9 TDBXForeignKeyColumnsColumns Class

Accesses columns by name in the **ForeignKeyColumns** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXForeignKeyColumnsColumns

Pascal

```
TDBXForeignKeyColumnsColumns = class;
```

C++

```
class TDBXForeignKeyColumnsColumns;
```

File

DBXMetaDataNames

Description

TDBXForeignKeyColumnsColumns is used to access columns by name in the **ForeignKeyColumns** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetForeignKeyColumns when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Name	Data Type of Column Data	Description
CatalogName (see page 10727)	String	Catalog name of table with foreign key.
SchemaName (see page 10729)	String	Schema name of table with foreign key.
TableName (see page 10729)	String	Table name of table with foreign key.
ForeignKeyName (see page 10727)	String	Foreign key name.

ColumnName (see page 10727)	String	Column that refers to a column in another table.
PrimaryCatalogName (see page 10728)	String	Catalog name of table being referred to.
PrimarySchemaName (see page 10728)	String	Schema name of table being referred to.
PrimaryTableName (see page 10729)	String	Name of table being referred to.
PrimaryKeyName (see page 10728)	String	Name of key being referred to. This is usually the primary key.
PrimaryColumnName (see page 10728)	String	Column being referred to.
Ordinal (see page 10727)	Int32	Position of column reference in foreign key.

See Also

TDBXForeignKeysIndex (see page 10736)

TDBXMetaDataCommands (see page 10525)

56.1.9.1 TDBXForeignKeyColumnsColumns Members

The following tables list the members exposed by TDBXForeignKeyColumnsColumns.

Public Constants

	Name	Description
	CatalogName (see page 10727)	This is CatalogName, a member of class TDBXForeignKeyColumnsColumns.
	ColumnName (see page 10727)	This is ColumnName, a member of class TDBXForeignKeyColumnsColumns.
	ForeignKeyName (see page 10727)	This is ForeignKeyName, a member of class TDBXForeignKeyColumnsColumns.
	Ordinal (see page 10727)	This is Ordinal, a member of class TDBXForeignKeyColumnsColumns.
	PrimaryCatalogName (see page 10728)	This is PrimaryCatalogName, a member of class TDBXForeignKeyColumnsColumns.
	PrimaryColumnName (see page 10728)	This is PrimaryColumnName, a member of class TDBXForeignKeyColumnsColumns.
	PrimaryKeyName (see page 10728)	This is PrimaryKeyName, a member of class TDBXForeignKeyColumnsColumns.
	PrimarySchemaName (see page 10728)	This is PrimarySchemaName, a member of class TDBXForeignKeyColumnsColumns.
	PrimaryTableName (see page 10729)	This is PrimaryTableName, a member of class TDBXForeignKeyColumnsColumns.
	SchemaName (see page 10729)	This is SchemaName, a member of class TDBXForeignKeyColumnsColumns.
	TableName (see page 10729)	This is TableName, a member of class TDBXForeignKeyColumnsColumns.

56.1.9.2 TDBXForeignKeyColumnsColumns Constants

The constants of the TDBXForeignKeyColumnsColumns class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10727)	This is CatalogName, a member of class TDBXForeignKeyColumnsColumns.
	ColumnName (see page 10727)	This is ColumnName, a member of class TDBXForeignKeyColumnsColumns.
	ForeignKeyName (see page 10727)	This is ForeignKeyName, a member of class TDBXForeignKeyColumnsColumns.
	Ordinal (see page 10727)	This is Ordinal, a member of class TDBXForeignKeyColumnsColumns.

	PrimaryCatalogName (see page 10728)	This is PrimaryCatalogName, a member of class TDBXForeignKeyColumnsColumns.
	PrimaryColumnName (see page 10728)	This is PrimaryColumnName, a member of class TDBXForeignKeyColumnsColumns.
	PrimaryKeyName (see page 10728)	This is PrimaryKeyName, a member of class TDBXForeignKeyColumnsColumns.
	PrimarySchemaName (see page 10728)	This is PrimarySchemaName, a member of class TDBXForeignKeyColumnsColumns.
	PrimaryTableName (see page 10729)	This is PrimaryTableName, a member of class TDBXForeignKeyColumnsColumns.
	SchemaName (see page 10729)	This is SchemaName, a member of class TDBXForeignKeyColumnsColumns.
	TableName (see page 10729)	This is TableName, a member of class TDBXForeignKeyColumnsColumns.

56.1.9.2.1 TDBXForeignKeyColumnsColumns.CatalogName Constant

Pascal

```
const CatalogName = 'CatalogName';
```

C++

```
CatalogName = 'CatalogName';
```

Description

This is CatalogName, a member of class TDBXForeignKeyColumnsColumns.

56.1.9.2.2 TDBXForeignKeyColumnsColumns.ColumnName Constant

Pascal

```
const ColumnName = 'ColumnName';
```

C++

```
ColumnName = 'ColumnName';
```

Description

This is ColumnName, a member of class TDBXForeignKeyColumnsColumns.

56.1.9.2.3 TDBXForeignKeyColumnsColumns.ForeignKeyName Constant

Pascal

```
const ForeignKeyName = 'ForeignKeyName';
```

C++

```
ForeignKeyName = 'ForeignKeyName';
```

Description

This is ForeignKeyName, a member of class TDBXForeignKeyColumnsColumns.

56.1.9.2.4 TDBXForeignKeyColumnsColumns.Ordinal Constant

Pascal

```
const Ordinal = 'Ordinal';
```

C++

```
Ordinal = 'Ordinal';
```

Description

This is Ordinal, a member of class TDBXForeignKeyColumnsColumns.

56.1.9.2.5 TDBXForeignKeyColumnsColumns.PrimaryCatalogName Constant

Pascal

```
const PrimaryCatalogName = 'PrimaryCatalogName';
```

C++

```
PrimaryCatalogName = 'PrimaryCatalogName';
```

Description

This is PrimaryCatalogName, a member of class TDBXForeignKeyColumnsColumns.

56.1.9.2.6 TDBXForeignKeyColumnsColumns.PrimaryColumnName Constant

Pascal

```
const PrimaryColumnName = 'PrimaryColumnName';
```

C++

```
PrimaryColumnName = 'PrimaryColumnName';
```

Description

This is PrimaryColumnName, a member of class TDBXForeignKeyColumnsColumns.

56.1.9.2.7 TDBXForeignKeyColumnsColumns.PrimaryKeyName Constant

Pascal

```
const PrimaryKeyName = 'PrimaryKeyName';
```

C++

```
PrimaryKeyName = 'PrimaryKeyName';
```

Description

This is PrimaryKeyName, a member of class TDBXForeignKeyColumnsColumns.

56.1.9.2.8 TDBXForeignKeyColumnsColumns.PrimarySchemaName Constant

Pascal

```
const PrimarySchemaName = 'PrimarySchemaName';
```

C++

```
PrimarySchemaName = 'PrimarySchemaName';
```

Description

This is PrimarySchemaName, a member of class TDBXForeignKeyColumnsColumns.

56.1.9.2.9 TDBXForeignKeyColumnsColumns.PrimaryTableName Constant

Pascal

```
const PrimaryTableName = 'PrimaryTableName';
```

C++

```
PrimaryTableName = 'PrimaryTableName';
```

Description

This is PrimaryTableName, a member of class TDBXForeignKeyColumnsColumns.

56.1.9.2.10 TDBXForeignKeyColumnsColumns.SchemaName Constant

Pascal

```
const SchemaName = 'SchemaName';
```

C++

```
SchemaName = 'SchemaName';
```

Description

This is SchemaName, a member of class TDBXForeignKeyColumnsColumns.

56.1.9.2.11 TDBXForeignKeyColumnsColumns.TableName Constant

Pascal

```
const TableName = 'TableName';
```

C++

```
TableName = 'TableName';
```

Description

This is TableName, a member of class TDBXForeignKeyColumnsColumns.

56.1.10 TDBXForeignKeyColumnsIndex Class

Accesses columns by ordinal in the **ForeignKeyColumns** metadata.

Class Hierarchy

Pascal

```
TDBXForeignKeyColumnsIndex = class;
```

C++

```
class TDBXForeignKeyColumnsIndex;
```

File

DBXMetaDataNames

Description

TDBXForeignKeyColumnsIndex is used to access columns by ordinal in the **ForeignKeyColumns** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetForeignKeyColumns when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Ordinal (see page 10732) Name	Data Type of Column Data	Description
CatalogName (see page 10731)	String	Catalog name of table with foreign key.
SchemaName (see page 10733)	String	Schema name of table with foreign key.
TableName (see page 10734)	String	Table name of table with foreign key.
ForeignKeyName (see page 10732)	String	Foreign key name.
ColumnName (see page 10731)	String	Column that refers to a column in another table.
PrimaryCatalogName (see page 10732)	String	Catalog name of table being referred to.
PrimarySchemaName (see page 10733)	String	Schema name of table being referred to.
PrimaryTableName (see page 10733)	String	Name of table being referred to.
PrimaryKeyName (see page 10733)	String	Name of key being referred to. This is usually the primary key.
PrimaryColumnName (see page 10732)	String	Column being referred to.
Ordinal (see page 10732)	Int32	Position of column reference in foreign key.

See Also

TDBXForeignKeyColumnsColumns (see page 10725)

TDBXMetaDataCommands (see page 10525)

56.1.10.1 TDBXForeignKeyColumnsIndex Members

The following tables list the members exposed by TDBXForeignKeyColumnsIndex.

Public Constants

	Name	Description
	CatalogName (see page 10731)	This is CatalogName, a member of class TDBXForeignKeyColumnsIndex.
	ColumnName (see page 10731)	This is ColumnName, a member of class TDBXForeignKeyColumnsIndex.
	ForeignKeyName (see page 10732)	This is ForeignKeyName, a member of class TDBXForeignKeyColumnsIndex.
	Last (see page 10732)	This is Last, a member of class TDBXForeignKeyColumnsIndex.
	Ordinal (see page 10732)	This is Ordinal, a member of class TDBXForeignKeyColumnsIndex.
	PrimaryCatalogName (see page 10732)	This is PrimaryCatalogName, a member of class TDBXForeignKeyColumnsIndex.

	PrimaryColumnName (see page 10732)	This is PrimaryColumnName, a member of class TDBXForeignKeyColumnsIndex.
	PrimaryKeyName (see page 10733)	This is PrimaryKeyName, a member of class TDBXForeignKeyColumnsIndex.
	PrimarySchemaName (see page 10733)	This is PrimarySchemaName, a member of class TDBXForeignKeyColumnsIndex.
	PrimaryTableName (see page 10733)	This is PrimaryTableName, a member of class TDBXForeignKeyColumnsIndex.
	SchemaName (see page 10733)	This is SchemaName, a member of class TDBXForeignKeyColumnsIndex.
	TableName (see page 10734)	This is TableName, a member of class TDBXForeignKeyColumnsIndex.

56.1.10.2 TDBXForeignKeyColumnsIndex Constants

The constants of the TDBXForeignKeyColumnsIndex class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10731)	This is CatalogName, a member of class TDBXForeignKeyColumnsIndex.
	ColumnName (see page 10731)	This is ColumnName, a member of class TDBXForeignKeyColumnsIndex.
	ForeignKeyName (see page 10732)	This is ForeignKeyName, a member of class TDBXForeignKeyColumnsIndex.
	Last (see page 10732)	This is Last, a member of class TDBXForeignKeyColumnsIndex.
	Ordinal (see page 10732)	This is Ordinal, a member of class TDBXForeignKeyColumnsIndex.
	PrimaryCatalogName (see page 10732)	This is PrimaryCatalogName, a member of class TDBXForeignKeyColumnsIndex.
	PrimaryColumnName (see page 10732)	This is PrimaryColumnName, a member of class TDBXForeignKeyColumnsIndex.
	PrimaryKeyName (see page 10733)	This is PrimaryKeyName, a member of class TDBXForeignKeyColumnsIndex.
	PrimarySchemaName (see page 10733)	This is PrimarySchemaName, a member of class TDBXForeignKeyColumnsIndex.
	PrimaryTableName (see page 10733)	This is PrimaryTableName, a member of class TDBXForeignKeyColumnsIndex.
	SchemaName (see page 10733)	This is SchemaName, a member of class TDBXForeignKeyColumnsIndex.
	TableName (see page 10734)	This is TableName, a member of class TDBXForeignKeyColumnsIndex.

56.1.10.2.1 TDBXForeignKeyColumnsIndex.CatalogName Constant

Pascal

```
const CatalogName = 0;
```

C++

```
CatalogName = 0;
```

Description

This is CatalogName, a member of class TDBXForeignKeyColumnsIndex.

56.1.10.2.2 TDBXForeignKeyColumnsIndex.ColumnName Constant

Pascal

```
const ColumnName = 4;
```

C++

```
ColumnName = 4;
```

Description

This is ColumnName, a member of class TDBXForeignKeyColumnsIndex.

56.1.10.2.3 TDBXForeignKeyColumnsIndex.ForeignKeyName Constant

Pascal

```
const ForeignKeyName = 3;
```

C++

```
ForeignKeyName = 3;
```

Description

This is ForeignKeyName, a member of class TDBXForeignKeyColumnsIndex.

56.1.10.2.4 TDBXForeignKeyColumnsIndex.Last Constant

Pascal

```
const Last = 10;
```

C++

```
Last = 10;
```

Description

This is Last, a member of class TDBXForeignKeyColumnsIndex.

56.1.10.2.5 TDBXForeignKeyColumnsIndex.Ordinal Constant

Pascal

```
const Ordinal = 10;
```

C++

```
Ordinal = 10;
```

Description

This is Ordinal, a member of class TDBXForeignKeyColumnsIndex.

56.1.10.2.6 TDBXForeignKeyColumnsIndex.PrimaryCatalogName Constant

Pascal

```
const PrimaryCatalogName = 5;
```

C++

```
PrimaryCatalogName = 5;
```

Description

This is PrimaryCatalogName, a member of class TDBXForeignKeyColumnsIndex.

56.1.10.2.7 TDBXForeignKeyColumnsIndex.PrimaryColumnName Constant

Pascal

```
const PrimaryColumnName = 9;
```

C++

```
PrimaryColumnName = 9;
```

Description

This is PrimaryColumnName, a member of class TDBXForeignKeyColumnsIndex.

56.1.10.2.8 TDBXForeignKeyColumnsIndex.PrimaryKeyName Constant

Pascal

```
const PrimaryKeyName = 8;
```

C++

```
PrimaryKeyName = 8;
```

Description

This is PrimaryKeyName, a member of class TDBXForeignKeyColumnsIndex.

56.1.10.2.9 TDBXForeignKeyColumnsIndex.PrimarySchemaName Constant

Pascal

```
const PrimarySchemaName = 6;
```

C++

```
PrimarySchemaName = 6;
```

Description

This is PrimarySchemaName, a member of class TDBXForeignKeyColumnsIndex.

56.1.10.2.10 TDBXForeignKeyColumnsIndex.PrimaryTableName Constant

Pascal

```
const PrimaryTableName = 7;
```

C++

```
PrimaryTableName = 7;
```

Description

This is PrimaryTableName, a member of class TDBXForeignKeyColumnsIndex.

56.1.10.2.11 TDBXForeignKeyColumnsIndex.SchemaName Constant

Pascal

```
const SchemaName = 1;
```

C++

```
SchemaName = 1;
```

Description

This is SchemaName, a member of class TDBXForeignKeyColumnsIndex.

56.1.10.2.12 TDBXForeignKeyColumnsIndex.TableName Constant

Pascal

```
const TableName = 2;
```

C++

```
TableName = 2;
```

Description

This is TableName, a member of class TDBXForeignKeyColumnsIndex.

56.1.11 TDBXForeignKeysColumns Class

Accesses columns by name in the **ForeignKeys** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXForeignKeysColumns

Pascal

```
TDBXForeignKeysColumns = class;
```

C++

```
class TDBXForeignKeysColumns;
```

File

DBXMetaDataNames

Description

TDBXForeignKeysColumns is used to access columns by name in the **ForeignKeys** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetForeignKeys when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Name	Data Type of Column Data	Description
CatalogName (see page 10735)	String	Catalog name.
SchemaName (see page 10735)	String	Schema name.
TableName (see page 10736)	String	Table name.
ForeignKeyName (see page 10735)	String	Foreign key name.

See Also

TDBXForeignKeysIndex (see page 10736)

TDBXMetaDataCommands (see page 10525)

56.1.11.1 TDBXForeignKeysColumns Members

The following tables list the members exposed by TDBXForeignKeysColumns.

Public Constants

	Name	Description
	CatalogName (see page 10735)	This is CatalogName, a member of class TDBXForeignKeysColumns.
	ForeignKeyName (see page 10735)	This is ForeignKeyName, a member of class TDBXForeignKeysColumns.
	SchemaName (see page 10735)	This is SchemaName, a member of class TDBXForeignKeysColumns.
	TableName (see page 10736)	This is TableName, a member of class TDBXForeignKeysColumns.

56.1.11.2 TDBXForeignKeysColumns Constants

The constants of the TDBXForeignKeysColumns class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10735)	This is CatalogName, a member of class TDBXForeignKeysColumns.
	ForeignKeyName (see page 10735)	This is ForeignKeyName, a member of class TDBXForeignKeysColumns.
	SchemaName (see page 10735)	This is SchemaName, a member of class TDBXForeignKeysColumns.
	TableName (see page 10736)	This is TableName, a member of class TDBXForeignKeysColumns.

56.1.11.2.1 TDBXForeignKeysColumns.CatalogName Constant

Pascal

```
const CatalogName = 'CatalogName';
```

C++

```
CatalogName = 'CatalogName';
```

Description

This is CatalogName, a member of class TDBXForeignKeysColumns.

56.1.11.2.2 TDBXForeignKeysColumns.ForeignKeyName Constant

Pascal

```
const ForeignKeyName = 'ForeignKeyName';
```

C++

```
ForeignKeyName = 'ForeignKeyName';
```

Description

This is ForeignKeyName, a member of class TDBXForeignKeysColumns.

56.1.11.2.3 TDBXForeignKeysColumns.SchemaName Constant

Pascal

```
const SchemaName = 'SchemaName';
```

C++

```
SchemaName = 'SchemaName';
```

Description

This is SchemaName, a member of class TDBXForeignKeysColumns.

56.1.11.2.4 TDBXForeignKeysColumns.TableName Constant

Pascal

```
const TableName = 'TableName';
```

C++

```
TableName = 'TableName';
```

Description

This is TableName, a member of class TDBXForeignKeysColumns.

56.1.12 TDBXForeignKeysIndex Class

Accesses columns by ordinal in the **ForeignKeys** metadata.

Class Hierarchy

Pascal

```
TDBXForeignKeysIndex = class;
```

C++

```
class TDBXForeignKeysIndex;
```

File

DBXMetaDataNames

Description

TDBXForeignKeysIndex is used to access columns by ordinal in the **ForeignKeys** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetForeignKeys when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Ordinal Name	Data Type of Column Data	Description
CatalogName (see page 10737)	String	Catalog name.
SchemaName (see page 10738)	String	Schema name.
TableName (see page 10738)	String	Table name.
ForeignKeyName (see page 10737)	String	Foreign key name.

See Also

TDBXForeignKeysColumns (see page 10734)

TDBXMetaDataCommands (see page 10525)

56.1.12.1 TDBXForeignKeysIndex Members

The following tables list the members exposed by TDBXForeignKeysIndex.

Public Constants

	Name	Description
	CatalogName (see page 10737)	This is CatalogName, a member of class TDBXForeignKeysIndex.
	ForeignKeyName (see page 10737)	This is ForeignKeyName, a member of class TDBXForeignKeysIndex.
	Last (see page 10737)	This is Last, a member of class TDBXForeignKeysIndex.
	SchemaName (see page 10738)	This is SchemaName, a member of class TDBXForeignKeysIndex.
	TableName (see page 10738)	This is TableName, a member of class TDBXForeignKeysIndex.

56.1.12.2 TDBXForeignKeysIndex Constants

The constants of the TDBXForeignKeysIndex class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10737)	This is CatalogName, a member of class TDBXForeignKeysIndex.
	ForeignKeyName (see page 10737)	This is ForeignKeyName, a member of class TDBXForeignKeysIndex.
	Last (see page 10737)	This is Last, a member of class TDBXForeignKeysIndex.
	SchemaName (see page 10738)	This is SchemaName, a member of class TDBXForeignKeysIndex.
	TableName (see page 10738)	This is TableName, a member of class TDBXForeignKeysIndex.

56.1.12.2.1 TDBXForeignKeysIndex.CatalogName Constant

Pascal

```
const CatalogName = 0;
```

C++

```
CatalogName = 0;
```

Description

This is CatalogName, a member of class TDBXForeignKeysIndex.

56.1.12.2.2 TDBXForeignKeysIndex.ForeignKeyName Constant

Pascal

```
const ForeignKeyName = 3;
```

C++

```
ForeignKeyName = 3;
```

Description

This is ForeignKeyName, a member of class TDBXForeignKeysIndex.

56.1.12.2.3 TDBXForeignKeysIndex.Last Constant

Pascal

```
const Last = 3;
```

C++

```
Last = 3;
```

Description

This is Last, a member of class TDBXForeignKeysIndex.

56.1.12.2.4 TDBXForeignKeysIndex.SchemaName Constant

Pascal

```
const SchemaName = 1;
```

C++

```
SchemaName = 1;
```

Description

This is SchemaName, a member of class TDBXForeignKeysIndex.

56.1.12.2.5 TDBXForeignKeysIndex.TableName Constant

Pascal

```
const TableName = 2;
```

C++

```
TableName = 2;
```

Description

This is TableName, a member of class TDBXForeignKeysIndex.

56.1.13 TDBXIndexColumnsColumns Class

Accesses columns by name in the **IndexColumns** metadata.

Class Hierarchy

Pascal

```
TDBXIndexColumnsColumns = class;
```

C++

```
class TDBXIndexColumnsColumns;
```

File

DBXMetaDataNames

Description

TDBXIndexColumnsColumns is used to access columns by name in the **IndexColumns** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetIndexColumns when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Name	Data Type of Column Data	Description
CatalogName (see page 10740)	String	Catalog name.
SchemaName (see page 10741)	String	Schema name.
TableName (see page 10741)	String	Table name.
IndexName (see page 10740)	String	Index name.
ColumnName (see page 10740)	String	Column in index.
Ordinal (see page 10740)	Int32	Position of column in index starting with 1.
IsAscending (see page 10740)	Boolean	Is a descending index column.

See Also

TDBXIndexColumnsIndex (see page 10741)

TDBXMetaDataCommands (see page 10525)

56.1.13.1 TDBXIndexColumnsColumns Members

The following tables list the members exposed by TDBXIndexColumnsColumns.

Public Constants

	Name	Description
	CatalogName (see page 10740)	This is CatalogName, a member of class TDBXIndexColumnsColumns.
	ColumnName (see page 10740)	This is ColumnName, a member of class TDBXIndexColumnsColumns.
	IndexName (see page 10740)	This is IndexName, a member of class TDBXIndexColumnsColumns.
	IsAscending (see page 10740)	This is IsAscending, a member of class TDBXIndexColumnsColumns.
	Ordinal (see page 10740)	This is Ordinal, a member of class TDBXIndexColumnsColumns.
	SchemaName (see page 10741)	This is SchemaName, a member of class TDBXIndexColumnsColumns.
	TableName (see page 10741)	This is TableName, a member of class TDBXIndexColumnsColumns.

56.1.13.2 TDBXIndexColumnsColumns Constants

The constants of the TDBXIndexColumnsColumns class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10740)	This is CatalogName, a member of class TDBXIndexColumnsColumns.
	ColumnName (see page 10740)	This is ColumnName, a member of class TDBXIndexColumnsColumns.
	IndexName (see page 10740)	This is IndexName, a member of class TDBXIndexColumnsColumns.
	IsAscending (see page 10740)	This is IsAscending, a member of class TDBXIndexColumnsColumns.
	Ordinal (see page 10740)	This is Ordinal, a member of class TDBXIndexColumnsColumns.
	SchemaName (see page 10741)	This is SchemaName, a member of class TDBXIndexColumnsColumns.
	TableName (see page 10741)	This is TableName, a member of class TDBXIndexColumnsColumns.

56.1.13.2.1 TDBXIndexColumnsColumns.CatalogName Constant

Pascal

```
const CatalogName = 'CatalogName';
```

C++

```
CatalogName = 'CatalogName';
```

Description

This is CatalogName, a member of class TDBXIndexColumnsColumns.

56.1.13.2.2 TDBXIndexColumnsColumns.ColumnName Constant

Pascal

```
const ColumnName = 'ColumnName';
```

C++

```
ColumnName = 'ColumnName';
```

Description

This is ColumnName, a member of class TDBXIndexColumnsColumns.

56.1.13.2.3 TDBXIndexColumnsColumns.IndexName Constant

Pascal

```
const IndexName = 'IndexName';
```

C++

```
IndexName = 'IndexName';
```

Description

This is IndexName, a member of class TDBXIndexColumnsColumns.

56.1.13.2.4 TDBXIndexColumnsColumns.IsAscending Constant

Pascal

```
const IsAscending = 'IsAscending';
```

C++

```
IsAscending = 'IsAscending';
```

Description

This is IsAscending, a member of class TDBXIndexColumnsColumns.

56.1.13.2.5 TDBXIndexColumnsColumns.Ordinal Constant

Pascal

```
const Ordinal = 'Ordinal';
```

C++

```
Ordinal = 'Ordinal';
```

Description

This is Ordinal, a member of class TDBXIndexColumnsColumns.

56.1.13.2.6 TDBXIndexColumnsColumns.SchemaName Constant

Pascal

```
const SchemaName = 'SchemaName';
```

C++

```
SchemaName = 'SchemaName';
```

Description

This is SchemaName, a member of class TDBXIndexColumnsColumns.

56.1.13.2.7 TDBXIndexColumnsColumns.TableName Constant

Pascal

```
const TableName = 'TableName';
```

C++

```
TableName = 'TableName';
```

Description

This is TableName, a member of class TDBXIndexColumnsColumns.

56.1.14 TDBXIndexColumnsIndex Class

Accesses columns by ordinal in the **IndexColumns** metadata.

Class Hierarchy

Pascal

```
TDBXIndexColumnsIndex = class;
```

C++

```
class TDBXIndexColumnsIndex;
```

File

DBXMetaDataNames

Description

TDBXIndexColumnsIndex is used to access columns by ordinal in the **IndexColumns** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetIndexColumns when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Ordinal (see page 10744) Name	Data Type of Column Data	Description
CatalogName (see page 10743)	String	Catalog name.
SchemaName (see page 10744)	String	Schema name.
TableName (see page 10744)	String	Table name.
IndexName (see page 10743)	String	Index name.
ColumnName (see page 10743)	String	Column in index.
Ordinal (see page 10744)	Int32	Position of column in index starting with 1.
IsAscending (see page 10743)	Boolean	Is a descending index column.

See Also

TDBXIndexColumnsColumns (see page 10738)

TDBXMetaDataCommands (see page 10525)

56.1.14.1 TDBXIndexColumnsIndex Members

The following tables list the members exposed by TDBXIndexColumnsIndex.

Public Constants

	Name	Description
	CatalogName (see page 10743)	This is CatalogName, a member of class TDBXIndexColumnsIndex.
	ColumnName (see page 10743)	This is ColumnName, a member of class TDBXIndexColumnsIndex.
	IndexName (see page 10743)	This is IndexName, a member of class TDBXIndexColumnsIndex.
	IsAscending (see page 10743)	This is IsAscending, a member of class TDBXIndexColumnsIndex.
	Last (see page 10743)	This is Last, a member of class TDBXIndexColumnsIndex.
	Ordinal (see page 10744)	This is Ordinal, a member of class TDBXIndexColumnsIndex.
	SchemaName (see page 10744)	This is SchemaName, a member of class TDBXIndexColumnsIndex.
	TableName (see page 10744)	This is TableName, a member of class TDBXIndexColumnsIndex.

56.1.14.2 TDBXIndexColumnsIndex Constants

The constants of the TDBXIndexColumnsIndex class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10743)	This is CatalogName, a member of class TDBXIndexColumnsIndex.
	ColumnName (see page 10743)	This is ColumnName, a member of class TDBXIndexColumnsIndex.
	IndexName (see page 10743)	This is IndexName, a member of class TDBXIndexColumnsIndex.
	IsAscending (see page 10743)	This is IsAscending, a member of class TDBXIndexColumnsIndex.
	Last (see page 10743)	This is Last, a member of class TDBXIndexColumnsIndex.
	Ordinal (see page 10744)	This is Ordinal, a member of class TDBXIndexColumnsIndex.
	SchemaName (see page 10744)	This is SchemaName, a member of class TDBXIndexColumnsIndex.
	TableName (see page 10744)	This is TableName, a member of class TDBXIndexColumnsIndex.

56.1.14.2.1 TDBXIndexColumnsIndex.CatalogName Constant

Pascal

```
const CatalogName = 0;
```

C++

```
CatalogName = 0;
```

Description

This is CatalogName, a member of class TDBXIndexColumnsIndex.

56.1.14.2.2 TDBXIndexColumnsIndex.ColumnName Constant

Pascal

```
const ColumnName = 4;
```

C++

```
ColumnName = 4;
```

Description

This is ColumnName, a member of class TDBXIndexColumnsIndex.

56.1.14.2.3 TDBXIndexColumnsIndex.IndexName Constant

Pascal

```
const IndexName = 3;
```

C++

```
IndexName = 3;
```

Description

This is IndexName, a member of class TDBXIndexColumnsIndex.

56.1.14.2.4 TDBXIndexColumnsIndex.IsAscending Constant

Pascal

```
const IsAscending = 6;
```

C++

```
IsAscending = 6;
```

Description

This is IsAscending, a member of class TDBXIndexColumnsIndex.

56.1.14.2.5 TDBXIndexColumnsIndex.Last Constant

Pascal

```
const Last = 6;
```

C++

```
Last = 6;
```

Description

This is Last, a member of class TDBXIndexColumnsIndex.

56.1.14.2.6 TDBXIndexColumnsIndex.Ordinal Constant

Pascal

```
const Ordinal = 5;
```

C++

```
Ordinal = 5;
```

Description

This is Ordinal, a member of class TDBXIndexColumnsIndex.

56.1.14.2.7 TDBXIndexColumnsIndex.SchemaName Constant

Pascal

```
const SchemaName = 1;
```

C++

```
SchemaName = 1;
```

Description

This is SchemaName, a member of class TDBXIndexColumnsIndex.

56.1.14.2.8 TDBXIndexColumnsIndex.TableName Constant

Pascal

```
const TableName = 2;
```

C++

```
TableName = 2;
```

Description

This is TableName, a member of class TDBXIndexColumnsIndex.

56.1.15 TDBXIndexesColumns Class

Accesses columns by name in the **Indexes** metadata.

Class Hierarchy

Pascal

```
TDBXIndexesColumns = class;
```

C++

```
class TDBXIndexesColumns;
```

File

DBXMetaDataNames

Description

TDBXIndexesColumns is used to access columns by name in the **Indexes** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetIndexes when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Name	Data Type of Column Data	Description
CatalogName (see page 10746)	String	Catalog name.
SchemaName (see page 10747)	String	Schema name.
TableName (see page 10747)	String	Table name.
IndexName (see page 10746)	String	Index name.
ConstraintName (see page 10746)	String	Constraint name; may be NULL.
IsPrimary (see page 10747)	Boolean	Is primary index of table.
IsUnique (see page 10747)	Boolean	Is unique index of table.
IsAscending (see page 10746)	Boolean	Is a descending index (Interbase only).

See Also

TDBXIndexesIndex (see page 10748)

TDBXMetaDataCommands (see page 10525)

56.1.15.1 TDBXIndexesColumns Members

The following tables list the members exposed by TDBXIndexesColumns.

Public Constants

	Name	Description
	CatalogName (see page 10746)	This is CatalogName, a member of class TDBXIndexesColumns.
	ConstraintName (see page 10746)	This is ConstraintName, a member of class TDBXIndexesColumns.
	IndexName (see page 10746)	This is IndexName, a member of class TDBXIndexesColumns.
	IsAscending (see page 10746)	This is IsAscending, a member of class TDBXIndexesColumns.
	IsPrimary (see page 10747)	This is IsPrimary, a member of class TDBXIndexesColumns.
	IsUnique (see page 10747)	This is IsUnique, a member of class TDBXIndexesColumns.
	SchemaName (see page 10747)	This is SchemaName, a member of class TDBXIndexesColumns.
	TableName (see page 10747)	This is TableName, a member of class TDBXIndexesColumns.

56.1.15.2 TDBXIndexesColumns Constants

The constants of the TDBXIndexesColumns class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10746)	This is CatalogName, a member of class TDBXIndexesColumns.
	ConstraintName (see page 10746)	This is ConstraintName, a member of class TDBXIndexesColumns.
	IndexName (see page 10746)	This is IndexName, a member of class TDBXIndexesColumns.
	IsAscending (see page 10746)	This is IsAscending, a member of class TDBXIndexesColumns.
	IsPrimary (see page 10747)	This is IsPrimary, a member of class TDBXIndexesColumns.
	IsUnique (see page 10747)	This is IsUnique, a member of class TDBXIndexesColumns.
	SchemaName (see page 10747)	This is SchemaName, a member of class TDBXIndexesColumns.
	TableName (see page 10747)	This is TableName, a member of class TDBXIndexesColumns.

56.1.15.2.1 TDBXIndexesColumns.CatalogName Constant

Pascal

```
const CatalogName = 'CatalogName';
```

C++

```
CatalogName = 'CatalogName';
```

Description

This is CatalogName, a member of class TDBXIndexesColumns.

56.1.15.2.2 TDBXIndexesColumns.ConstraintName Constant

Pascal

```
const ConstraintName = 'ConstraintName';
```

C++

```
ConstraintName = 'ConstraintName';
```

Description

This is ConstraintName, a member of class TDBXIndexesColumns.

56.1.15.2.3 TDBXIndexesColumns.IndexName Constant

Pascal

```
const IndexName = 'IndexName';
```

C++

```
IndexName = 'IndexName';
```

Description

This is IndexName, a member of class TDBXIndexesColumns.

56.1.15.2.4 TDBXIndexesColumns.IsAscending Constant

Pascal

```
const IsAscending = 'IsAscending';
```

C++

```
IsAscending = 'IsAscending';
```

Description

This is IsAscending, a member of class TDBXIndexesColumns.

56.1.15.2.5 TDBXIndexesColumns.IsPrimary Constant

Pascal

```
const IsPrimary = 'IsPrimary';
```

C++

```
IsPrimary = 'IsPrimary';
```

Description

This is IsPrimary, a member of class TDBXIndexesColumns.

56.1.15.2.6 TDBXIndexesColumns.IsUnique Constant

Pascal

```
const IsUnique = 'IsUnique';
```

C++

```
IsUnique = 'IsUnique';
```

Description

This is IsUnique, a member of class TDBXIndexesColumns.

56.1.15.2.7 TDBXIndexesColumns.SchemaName Constant

Pascal

```
const SchemaName = 'SchemaName';
```

C++

```
SchemaName = 'SchemaName';
```

Description

This is SchemaName, a member of class TDBXIndexesColumns.

56.1.15.2.8 TDBXIndexesColumns.TableName Constant

Pascal

```
const TableName = 'TableName';
```

C++

```
TableName = 'TableName';
```

Description

This is TableName, a member of class TDBXIndexesColumns.

56.1.16 TDBXIndexesIndex Class

Accesses columns by ordinal in the **Indexes** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXIndexesIndex

Pascal

```
TDBXIndexesIndex = class;
```

C++

```
class TDBXIndexesIndex;
```

File

DBXMetaDataNames

Description

TDBXIndexesIndex is used to access columns by ordinal in the **Indexes** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetIndexes when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Ordinal Name	Data Type of Column Data	Description
CatalogName (see page 10749)	String	Catalog name.
SchemaName (see page 10751)	String	Schema name.
TableName (see page 10751)	String	Table name.
IndexName (see page 10749)	String	Index name.
ConstraintName (see page 10749)	String	Constraint name; may be NULL.
IsPrimary (see page 10750)	Boolean	Is primary index of table.
IsUnique (see page 10750)	Boolean	Is unique index of table.
IsAscending (see page 10750)	Boolean	Is a descending index (Interbase only).

See Also

TDBXIndexesColumns (see page 10744)

TDBXMetaDataCommands (see page 10525)

56.1.16.1 TDBXIndexesIndex Members

The following tables list the members exposed by TDBXIndexesIndex.

Public Constants

	Name	Description
	CatalogName (see page 10749)	This is CatalogName, a member of class TDBXIndexesIndex.
	ConstraintName (see page 10749)	This is ConstraintName, a member of class TDBXIndexesIndex.

	IndexName (see page 10749)	This is IndexName, a member of class TDBXIndexesIndex.
	IsAscending (see page 10750)	This is IsAscending, a member of class TDBXIndexesIndex.
	IsPrimary (see page 10750)	This is IsPrimary, a member of class TDBXIndexesIndex.
	IsUnique (see page 10750)	This is IsUnique, a member of class TDBXIndexesIndex.
	Last (see page 10750)	This is Last, a member of class TDBXIndexesIndex.
	SchemaName (see page 10751)	This is SchemaName, a member of class TDBXIndexesIndex.
	TableName (see page 10751)	This is TableName, a member of class TDBXIndexesIndex.

56.1.16.2 TDBXIndexesIndex Constants

The constants of the TDBXIndexesIndex class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10749)	This is CatalogName, a member of class TDBXIndexesIndex.
	ConstraintName (see page 10749)	This is ConstraintName, a member of class TDBXIndexesIndex.
	IndexName (see page 10749)	This is IndexName, a member of class TDBXIndexesIndex.
	IsAscending (see page 10750)	This is IsAscending, a member of class TDBXIndexesIndex.
	IsPrimary (see page 10750)	This is IsPrimary, a member of class TDBXIndexesIndex.
	IsUnique (see page 10750)	This is IsUnique, a member of class TDBXIndexesIndex.
	Last (see page 10750)	This is Last, a member of class TDBXIndexesIndex.
	SchemaName (see page 10751)	This is SchemaName, a member of class TDBXIndexesIndex.
	TableName (see page 10751)	This is TableName, a member of class TDBXIndexesIndex.

56.1.16.2.1 TDBXIndexesIndex.CatalogName Constant

Pascal

```
const CatalogName = 0;
```

C++

```
CatalogName = 0;
```

Description

This is CatalogName, a member of class TDBXIndexesIndex.

56.1.16.2.2 TDBXIndexesIndex.ConstraintName Constant

Pascal

```
const ConstraintName = 4;
```

C++

```
ConstraintName = 4;
```

Description

This is ConstraintName, a member of class TDBXIndexesIndex.

56.1.16.2.3 TDBXIndexesIndex.IndexName Constant

Pascal

```
const IndexName = 3;
```

C++

```
IndexName = 3;
```

Description

This is IndexName, a member of class TDBXIndexesIndex.

56.1.16.2.4 TDBXIndexesIndex.IsAscending Constant

Pascal

```
const IsAscending = 7;
```

C++

```
IsAscending = 7;
```

Description

This is IsAscending, a member of class TDBXIndexesIndex.

56.1.16.2.5 TDBXIndexesIndex.IsPrimary Constant

Pascal

```
const IsPrimary = 5;
```

C++

```
IsPrimary = 5;
```

Description

This is IsPrimary, a member of class TDBXIndexesIndex.

56.1.16.2.6 TDBXIndexesIndex.IsUnique Constant

Pascal

```
const IsUnique = 6;
```

C++

```
IsUnique = 6;
```

Description

This is IsUnique, a member of class TDBXIndexesIndex.

56.1.16.2.7 TDBXIndexesIndex.Last Constant

Pascal

```
const Last = 7;
```

C++

```
Last = 7;
```

Description

This is Last, a member of class TDBXIndexesIndex.

56.1.16.2.8 TDBXIndexesIndex.SchemaName Constant

Pascal

```
const SchemaName = 1;
```

C++

```
SchemaName = 1;
```

Description

This is SchemaName, a member of class TDBXIndexesIndex.

56.1.16.2.9 TDBXIndexesIndex.TableName Constant

Pascal

```
const TableName = 2;
```

C++

```
TableName = 2;
```

Description

This is TableName, a member of class TDBXIndexesIndex.

56.1.17 TDBXMetaDataCollectionIndex Class

Represents a collection of metadata accessed by ordinal.

Class Hierarchy

DBXMetaDataNames.TDBXMetaDataCollectionIndex

Pascal

```
TDBXMetaDataCollectionIndex = class;
```

C++

```
class TDBXMetaDataCollectionIndex;
```

File

DBXMetaDataNames

Description

TDBXMetaDataCollectionIndex represents a collection of metadata accessed by ordinal.

This table describes the metadata collection ordinal names.

Collection Ordinal Name	Description
DataTypes (see page 10754)	Represents the data types supported by a specific database.
Catalogs (see page 10753)	Represents catalogs in a database.
Schemas (see page 10757)	Represents schemas in a database.
Tables (see page 10758)	Represents tables in a database.

Views (see page 10758)	Represents views in a database.
Synonyms (see page 10757)	Represents synonyms in a database.
Columns (see page 10754)	Represents table columns in a database.
ColumnConstraints (see page 10754)	Represents column constraints in a database. This is used only internally.
Indexes (see page 10755)	Represents table indexes in a database.
IndexColumns (see page 10755)	Represents index columns in a database.
ForeignKeys (see page 10755)	Represents foreign keys in a database.
ForeignKeyColumns (see page 10754)	Represents foreign key columns in a database.
Procedures (see page 10756)	Represents procedures in a database.
ProcedureSources (see page 10757)	Represents procedure source code in a database.
ProcedureParameters (see page 10756)	Represents procedure parameters in a database.
Packages (see page 10756)	Represents packages in a database.
PackageProcedures (see page 10755)	Represents package procedures in a database.
PackageProcedureParameters (see page 10755)	Represents package procedure parameters in a database.
PackageSources (see page 10756)	Represents package source code in a database.
Users (see page 10758)	Represents users in a database.
Roles (see page 10757)	Represents roles in a database.
ReservedWords (see page 10757)	Represents reserved words in a database.

See Also

TDBXMetaDataCollectionName (see page 10758)

56.1.17.1 TDBXMetaDataCollectionIndex Members

The following tables list the members exposed by TDBXMetaDataCollectionIndex.

Public Constants

	Name	Description
	Catalogs (see page 10753)	This is Catalogs, a member of class TDBXMetaDataCollectionIndex.
	ColumnConstraints (see page 10754)	This is ColumnConstraints, a member of class TDBXMetaDataCollectionIndex.
	Columns (see page 10754)	This is Columns, a member of class TDBXMetaDataCollectionIndex.
	DataTypes (see page 10754)	This is DataTypes, a member of class TDBXMetaDataCollectionIndex.
	ForeignKeyColumns (see page 10754)	This is ForeignKeyColumns, a member of class TDBXMetaDataCollectionIndex.

	ForeignKeys (see page 10755)	This is ForeignKeys, a member of class TDBXMetaDataCollectionIndex.
	IndexColumns (see page 10755)	This is IndexColumns, a member of class TDBXMetaDataCollectionIndex.
	Indexes (see page 10755)	This is Indexes, a member of class TDBXMetaDataCollectionIndex.
	PackageProcedureParameters (see page 10755)	This is PackageProcedureParameters, a member of class TDBXMetaDataCollectionIndex.
	PackageProcedures (see page 10755)	This is PackageProcedures, a member of class TDBXMetaDataCollectionIndex.
	Packages (see page 10756)	This is Packages, a member of class TDBXMetaDataCollectionIndex.
	PackageSources (see page 10756)	This is PackageSources, a member of class TDBXMetaDataCollectionIndex.
	ProcedureParameters (see page 10756)	This is ProcedureParameters, a member of class TDBXMetaDataCollectionIndex.
	Procedures (see page 10756)	This is Procedures, a member of class TDBXMetaDataCollectionIndex.
	ProcedureSources (see page 10757)	This is ProcedureSources, a member of class TDBXMetaDataCollectionIndex.
	ReservedWords (see page 10757)	This is ReservedWords, a member of class TDBXMetaDataCollectionIndex.
	Roles (see page 10757)	This is Roles, a member of class TDBXMetaDataCollectionIndex.
	Schemas (see page 10757)	This is Schemas, a member of class TDBXMetaDataCollectionIndex.
	Synonyms (see page 10757)	This is Synonyms, a member of class TDBXMetaDataCollectionIndex.
	Tables (see page 10758)	This is Tables, a member of class TDBXMetaDataCollectionIndex.
	Users (see page 10758)	This is Users, a member of class TDBXMetaDataCollectionIndex.
	Views (see page 10758)	This is Views, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2 TDBXMetaDataCollectionIndex Constants

The constants of the TDBXMetaDataCollectionIndex class are listed here.

Public Constants

	Name	Description
	Catalogs (see page 10753)	This is Catalogs, a member of class TDBXMetaDataCollectionIndex.
	ColumnConstraints (see page 10754)	This is ColumnConstraints, a member of class TDBXMetaDataCollectionIndex.
	Columns (see page 10754)	This is Columns, a member of class TDBXMetaDataCollectionIndex.
	DataTypes (see page 10754)	This is DataTypes, a member of class TDBXMetaDataCollectionIndex.
	ForeignKeyColumns (see page 10754)	This is ForeignKeyColumns, a member of class TDBXMetaDataCollectionIndex.
	ForeignKeys (see page 10755)	This is ForeignKeys, a member of class TDBXMetaDataCollectionIndex.
	IndexColumns (see page 10755)	This is IndexColumns, a member of class TDBXMetaDataCollectionIndex.
	Indexes (see page 10755)	This is Indexes, a member of class TDBXMetaDataCollectionIndex.
	PackageProcedureParameters (see page 10755)	This is PackageProcedureParameters, a member of class TDBXMetaDataCollectionIndex.
	PackageProcedures (see page 10755)	This is PackageProcedures, a member of class TDBXMetaDataCollectionIndex.
	Packages (see page 10756)	This is Packages, a member of class TDBXMetaDataCollectionIndex.
	PackageSources (see page 10756)	This is PackageSources, a member of class TDBXMetaDataCollectionIndex.
	ProcedureParameters (see page 10756)	This is ProcedureParameters, a member of class TDBXMetaDataCollectionIndex.
	Procedures (see page 10756)	This is Procedures, a member of class TDBXMetaDataCollectionIndex.
	ProcedureSources (see page 10757)	This is ProcedureSources, a member of class TDBXMetaDataCollectionIndex.
	ReservedWords (see page 10757)	This is ReservedWords, a member of class TDBXMetaDataCollectionIndex.
	Roles (see page 10757)	This is Roles, a member of class TDBXMetaDataCollectionIndex.
	Schemas (see page 10757)	This is Schemas, a member of class TDBXMetaDataCollectionIndex.
	Synonyms (see page 10757)	This is Synonyms, a member of class TDBXMetaDataCollectionIndex.
	Tables (see page 10758)	This is Tables, a member of class TDBXMetaDataCollectionIndex.
	Users (see page 10758)	This is Users, a member of class TDBXMetaDataCollectionIndex.
	Views (see page 10758)	This is Views, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.1 TDBXMetaDataCollectionIndex.Catalogs Constant

Pascal

```
const Catalogs = 2;
```

C++

```
Catalogs = 2;
```

Description

This is Catalogs, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.2 TDBXMetaDataCollectionIndex.ColumnConstraints Constant

Pascal

```
const ColumnConstraints = 8;
```

C++

```
ColumnConstraints = 8;
```

Description

This is ColumnConstraints, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.3 TDBXMetaDataCollectionIndex.Columns Constant

Pascal

```
const Columns = 7;
```

C++

```
Columns = 7;
```

Description

This is Columns, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.4 TDBXMetaDataCollectionIndex.DataTypes Constant

Pascal

```
const DataTypes = 1;
```

C++

```
DataTypes = 1;
```

Description

This is DataTypes, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.5 TDBXMetaDataCollectionIndex.ForeignKeyColumns Constant

Pascal

```
const ForeignKeyColumns = 12;
```

C++

```
ForeignKeyColumns = 12;
```

Description

This is ForeignKeyColumns, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.6 TDBXMetaDataCollectionIndex.ForeignKeys Constant

Pascal

```
const ForeignKeys = 11;
```

C++

```
ForeignKeys = 11;
```

Description

This is ForeignKeys, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.7 TDBXMetaDataCollectionIndex.IndexColumns Constant

Pascal

```
const IndexColumns = 10;
```

C++

```
IndexColumns = 10;
```

Description

This is IndexColumns, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.8 TDBXMetaDataCollectionIndex.Indexes Constant

Pascal

```
const Indexes = 9;
```

C++

```
Indexes = 9;
```

Description

This is Indexes, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.9 TDBXMetaDataCollectionIndex.PackageProcedureParameters Constant

Pascal

```
const PackageProcedureParameters = 18;
```

C++

```
PackageProcedureParameters = 18;
```

Description

This is PackageProcedureParameters, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.10 TDBXMetaDataCollectionIndex.PackageProcedures Constant

Pascal

```
const PackageProcedures = 17;
```

C++

```
PackageProcedures = 17;
```

Description

This is PackageProcedures, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.11 TDBXMetaDataCollectionIndex.Packages Constant

Pascal

```
const Packages = 16;
```

C++

```
Packages = 16;
```

Description

This is Packages, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.12 TDBXMetaDataCollectionIndex.PackageSources Constant

Pascal

```
const PackageSources = 19;
```

C++

```
PackageSources = 19;
```

Description

This is PackageSources, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.13 TDBXMetaDataCollectionIndex.ProcedureParameters Constant

Pascal

```
const ProcedureParameters = 15;
```

C++

```
ProcedureParameters = 15;
```

Description

This is ProcedureParameters, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.14 TDBXMetaDataCollectionIndex.Procedures Constant

Pascal

```
const Procedures = 13;
```

C++

```
Procedures = 13;
```

Description

This is Procedures, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.15 TDBXMetaDataCollectionIndex.ProcedureSources Constant

Pascal

```
const ProcedureSources = 14;
```

C++

```
ProcedureSources = 14;
```

Description

This is ProcedureSources, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.16 TDBXMetaDataCollectionIndex.ReservedWords Constant

Pascal

```
const ReservedWords = 22;
```

C++

```
ReservedWords = 22;
```

Description

This is ReservedWords, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.17 TDBXMetaDataCollectionIndex.Roles Constant

Pascal

```
const Roles = 21;
```

C++

```
Roles = 21;
```

Description

This is Roles, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.18 TDBXMetaDataCollectionIndex.Schemas Constant

Pascal

```
const Schemas = 3;
```

C++

```
Schemas = 3;
```

Description

This is Schemas, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.19 TDBXMetaDataCollectionIndex.Synonyms Constant

Pascal

```
const Synonyms = 6;
```

C++

```
Synonyms = 6;
```

Description

This is Synonyms, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.20 TDBXMetaDataCollectionIndex.Tables Constant

Pascal

```
const Tables = 4;
```

C++

```
Tables = 4;
```

Description

This is Tables, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.21 TDBXMetaDataCollectionIndex.Users Constant

Pascal

```
const Users = 20;
```

C++

```
Users = 20;
```

Description

This is Users, a member of class TDBXMetaDataCollectionIndex.

56.1.17.2.22 TDBXMetaDataCollectionIndex.Views Constant

Pascal

```
const Views = 5;
```

C++

```
Views = 5;
```

Description

This is Views, a member of class TDBXMetaDataCollectionIndex.

56.1.18 TDBXMetaDataCollectionName Class

Represents a collection of metadata accessed by name.

Class Hierarchy

Pascal

```
TDBXMetaDataCollectionName = class;
```

C++

```
class TDBXMetaDataCollectionName;
```

File

DBXMetaDataNames

Description

TDBXMetaDataCollectionName represents a collection of metadata accessed by name.

This table describes the metadata collection names.

Collection Name	Description
DataTypes (see page 10762)	Represents the data types supported by a specific database.
Catalogs (see page 10761)	Represents catalogs in a database.
Schemas (see page 10765)	Represents schemas in a database.
Tables (see page 10765)	Represents tables in a database.
Views (see page 10766)	Represents views in a database.
Synonyms (see page 10765)	Represents synonyms in a database.
Columns (see page 10761)	Represents table columns in a database.
ColumnConstraints (see page 10761)	Represents column constraints in a database. This is used only internally.
Indexes (see page 10762)	Represents table indexes in a database.
IndexColumns (see page 10762)	Represents index columns in a database.
ForeignKeys (see page 10762)	Represents foreign keys in a database.
ForeignKeyColumns (see page 10762)	Represents foreign key columns in a database.
Procedures (see page 10764)	Represents procedures in a database.
ProcedureSources (see page 10764)	Represents procedure source code in a database.
ProcedureParameters (see page 10764)	Represents procedure parameters in a database.
Packages (see page 10763)	Represents packages in a database.
PackageProcedures (see page 10763)	Represents package procedures in a database.
PackageProcedureParameters (see page 10763)	Represents package procedure parameters in a database.
PackageSources (see page 10763)	Represents package source code in a database.
Users (see page 10765)	Represents users in a database.
Roles (see page 10764)	Represents roles in a database.

ReservedWords (see page 10764)	Represents reserved words in a database.

See Also

TDBXMetaDataCollectionIndex (see page 10751)

56.1.18.1 TDBXMetaDataCollectionName Members

The following tables list the members exposed by TDBXMetaDataCollectionName.

Public Constants

	Name	Description
	Catalogs (see page 10761)	This is Catalogs, a member of class TDBXMetaDataCollectionName.
	ColumnConstraints (see page 10761)	This is ColumnConstraints, a member of class TDBXMetaDataCollectionName.
	Columns (see page 10761)	This is Columns, a member of class TDBXMetaDataCollectionName.
	DataTypes (see page 10762)	This is DataTypes, a member of class TDBXMetaDataCollectionName.
	ForeignKeyColumns (see page 10762)	This is ForeignKeyColumns, a member of class TDBXMetaDataCollectionName.
	ForeignKeys (see page 10762)	This is ForeignKeys, a member of class TDBXMetaDataCollectionName.
	IndexColumns (see page 10762)	This is IndexColumns, a member of class TDBXMetaDataCollectionName.
	Indexes (see page 10762)	This is Indexes, a member of class TDBXMetaDataCollectionName.
	PackageProcedureParameters (see page 10763)	This is PackageProcedureParameters, a member of class TDBXMetaDataCollectionName.
	PackageProcedures (see page 10763)	This is PackageProcedures, a member of class TDBXMetaDataCollectionName.
	Packages (see page 10763)	This is Packages, a member of class TDBXMetaDataCollectionName.
	PackageSources (see page 10763)	This is PackageSources, a member of class TDBXMetaDataCollectionName.
	ProcedureParameters (see page 10764)	This is ProcedureParameters, a member of class TDBXMetaDataCollectionName.
	Procedures (see page 10764)	This is Procedures, a member of class TDBXMetaDataCollectionName.
	ProcedureSources (see page 10764)	This is ProcedureSources, a member of class TDBXMetaDataCollectionName.
	ReservedWords (see page 10764)	This is ReservedWords, a member of class TDBXMetaDataCollectionName.
	Roles (see page 10764)	This is Roles, a member of class TDBXMetaDataCollectionName.
	Schemas (see page 10765)	This is Schemas, a member of class TDBXMetaDataCollectionName.
	Synonyms (see page 10765)	This is Synonyms, a member of class TDBXMetaDataCollectionName.
	Tables (see page 10765)	This is Tables, a member of class TDBXMetaDataCollectionName.
	Users (see page 10765)	This is Users, a member of class TDBXMetaDataCollectionName.
	Views (see page 10766)	This is Views, a member of class TDBXMetaDataCollectionName.

56.1.18.2 TDBXMetaDataCollectionName Constants

The constants of the TDBXMetaDataCollectionName class are listed here.

Public Constants

	Name	Description
	Catalogs (see page 10761)	This is Catalogs, a member of class TDBXMetaDataCollectionName.
	ColumnConstraints (see page 10761)	This is ColumnConstraints, a member of class TDBXMetaDataCollectionName.
	Columns (see page 10761)	This is Columns, a member of class TDBXMetaDataCollectionName.
	DataTypes (see page 10762)	This is DataTypes, a member of class TDBXMetaDataCollectionName.
	ForeignKeyColumns (see page 10762)	This is ForeignKeyColumns, a member of class TDBXMetaDataCollectionName.
	ForeignKeys (see page 10762)	This is ForeignKeys, a member of class TDBXMetaDataCollectionName.
	IndexColumns (see page 10762)	This is IndexColumns, a member of class TDBXMetaDataCollectionName.
	Indexes (see page 10762)	This is Indexes, a member of class TDBXMetaDataCollectionName.

	PackageProcedureParameters (see page 10763)	This is PackageProcedureParameters, a member of class TDBXMetaDataCollectionName.
	PackageProcedures (see page 10763)	This is PackageProcedures, a member of class TDBXMetaDataCollectionName.
	Packages (see page 10763)	This is Packages, a member of class TDBXMetaDataCollectionName.
	PackageSources (see page 10763)	This is PackageSources, a member of class TDBXMetaDataCollectionName.
	ProcedureParameters (see page 10764)	This is ProcedureParameters, a member of class TDBXMetaDataCollectionName.
	Procedures (see page 10764)	This is Procedures, a member of class TDBXMetaDataCollectionName.
	ProcedureSources (see page 10764)	This is ProcedureSources, a member of class TDBXMetaDataCollectionName.
	ReservedWords (see page 10764)	This is ReservedWords, a member of class TDBXMetaDataCollectionName.
	Roles (see page 10764)	This is Roles, a member of class TDBXMetaDataCollectionName.
	Schemas (see page 10765)	This is Schemas, a member of class TDBXMetaDataCollectionName.
	Synonyms (see page 10765)	This is Synonyms, a member of class TDBXMetaDataCollectionName.
	Tables (see page 10765)	This is Tables, a member of class TDBXMetaDataCollectionName.
	Users (see page 10765)	This is Users, a member of class TDBXMetaDataCollectionName.
	Views (see page 10766)	This is Views, a member of class TDBXMetaDataCollectionName.

56.1.18.2.1 TDBXMetaDataCollectionName.Catalogs Constant

Pascal

```
const Catalogs = 'Catalogs';
```

C++

```
Catalogs = 'Catalogs';
```

Description

This is Catalogs, a member of class TDBXMetaDataCollectionName.

56.1.18.2.2 TDBXMetaDataCollectionName.ColumnConstraints Constant

Pascal

```
const ColumnConstraints = 'ColumnConstraints';
```

C++

```
ColumnConstraints = 'ColumnConstraints';
```

Description

This is ColumnConstraints, a member of class TDBXMetaDataCollectionName.

56.1.18.2.3 TDBXMetaDataCollectionName.Columns Constant

Pascal

```
const Columns = 'Columns';
```

C++

```
Columns = 'Columns';
```

Description

This is Columns, a member of class TDBXMetaDataCollectionName.

56.1.18.2.4 TDBXMetaDataCollectionName.DataTypes Constant

Pascal

```
const DataTypes = 'DataTypes';
```

C++

```
DataTypes = 'DataTypes';
```

Description

This is DataTypes, a member of class TDBXMetaDataCollectionName.

56.1.18.2.5 TDBXMetaDataCollectionName.ForeignKeyColumns Constant

Pascal

```
const ForeignKeyColumns = 'ForeignKeyColumns';
```

C++

```
ForeignKeyColumns = 'ForeignKeyColumns';
```

Description

This is ForeignKeyColumns, a member of class TDBXMetaDataCollectionName.

56.1.18.2.6 TDBXMetaDataCollectionName.ForeignKeys Constant

Pascal

```
const ForeignKeys = 'ForeignKeys';
```

C++

```
ForeignKeys = 'ForeignKeys';
```

Description

This is ForeignKeys, a member of class TDBXMetaDataCollectionName.

56.1.18.2.7 TDBXMetaDataCollectionName.IndexColumns Constant

Pascal

```
const IndexColumns = 'IndexColumns';
```

C++

```
IndexColumns = 'IndexColumns';
```

Description

This is IndexColumns, a member of class TDBXMetaDataCollectionName.

56.1.18.2.8 TDBXMetaDataCollectionName.Indexes Constant

Pascal

```
const Indexes = 'Indexes';
```

C++

```
Indexes = 'Indexes';
```

Description

This is Indexes, a member of class TDBXMetaDataCollectionName.

56.1.18.2.9 TDBXMetaDataCollectionName.PackageProcedureParameters Constant

Pascal

```
const PackageProcedureParameters = 'PackageProcedureParameters';
```

C++

```
PackageProcedureParameters = 'PackageProcedureParameters';
```

Description

This is PackageProcedureParameters, a member of class TDBXMetaDataCollectionName.

56.1.18.2.10 TDBXMetaDataCollectionName.PackageProcedures Constant

Pascal

```
const PackageProcedures = 'PackageProcedures';
```

C++

```
PackageProcedures = 'PackageProcedures';
```

Description

This is PackageProcedures, a member of class TDBXMetaDataCollectionName.

56.1.18.2.11 TDBXMetaDataCollectionName.Packages Constant

Pascal

```
const Packages = 'Packages';
```

C++

```
Packages = 'Packages';
```

Description

This is Packages, a member of class TDBXMetaDataCollectionName.

56.1.18.2.12 TDBXMetaDataCollectionName.PackageSources Constant

Pascal

```
const PackageSources = 'PackageSources';
```

C++

```
PackageSources = 'PackageSources';
```

Description

This is PackageSources, a member of class TDBXMetaDataCollectionName.

56.1.18.2.13 TDBXMetaDataCollectionName.ProcedureParameters Constant

Pascal

```
const ProcedureParameters = 'ProcedureParameters';
```

C++

```
ProcedureParameters = 'ProcedureParameters';
```

Description

This is ProcedureParameters, a member of class TDBXMetaDataCollectionName.

56.1.18.2.14 TDBXMetaDataCollectionName.Procedures Constant

Pascal

```
const Procedures = 'Procedures';
```

C++

```
Procedures = 'Procedures';
```

Description

This is Procedures, a member of class TDBXMetaDataCollectionName.

56.1.18.2.15 TDBXMetaDataCollectionName.ProcedureSources Constant

Pascal

```
const ProcedureSources = 'ProcedureSources';
```

C++

```
ProcedureSources = 'ProcedureSources';
```

Description

This is ProcedureSources, a member of class TDBXMetaDataCollectionName.

56.1.18.2.16 TDBXMetaDataCollectionName.ReservedWords Constant

Pascal

```
const ReservedWords = 'ReservedWords';
```

C++

```
ReservedWords = 'ReservedWords';
```

Description

This is ReservedWords, a member of class TDBXMetaDataCollectionName.

56.1.18.2.17 TDBXMetaDataCollectionName.Roles Constant

Pascal

```
const Roles = 'Roles';
```

C++

```
Roles = 'Roles';
```

Description

This is Roles, a member of class TDBXMetaDataCollectionName.

56.1.18.2.18 TDBXMetaDataCollectionName.Schemas Constant

Pascal

```
const Schemas = 'Schemas';
```

C++

```
Schemas = 'Schemas';
```

Description

This is Schemas, a member of class TDBXMetaDataCollectionName.

56.1.18.2.19 TDBXMetaDataCollectionName.Synonyms Constant

Pascal

```
const Synonyms = 'Synonyms';
```

C++

```
Synonyms = 'Synonyms';
```

Description

This is Synonyms, a member of class TDBXMetaDataCollectionName.

56.1.18.2.20 TDBXMetaDataCollectionName.Tables Constant

Pascal

```
const Tables = 'Tables';
```

C++

```
Tables = 'Tables';
```

Description

This is Tables, a member of class TDBXMetaDataCollectionName.

56.1.18.2.21 TDBXMetaDataCollectionName.Users Constant

Pascal

```
const Users = 'Users';
```

C++

```
Users = 'Users';
```

Description

This is Users, a member of class TDBXMetaDataCollectionName.

56.1.18.2.22 TDBXMetaDataCollectionName.Views Constant

Pascal

```
const Views = 'Views';
```

C++

```
Views = 'Views';
```

Description

This is Views, a member of class TDBXMetaDataCollectionName.

56.1.19 TDBXPackageProcedureParametersColumns Class

Accesses columns by name in the **PackageProcedureParameters** metadata.

Class Hierarchy

Pascal

```
TDBXPackageProcedureParametersColumns = class;
```

C++

```
class TDBXPackageProcedureParametersColumns;
```

File

DBXMetaDataNames

Description

TDBXPackageProcedureParametersColumns is used to access columns by name in the **PackageProcedureParameters** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetPackageProcedureParameters when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Name	Data Type of Column Data	Description
CatalogName (see page 10768)	String	Catalog name.
SchemaName (see page 10772)	String	Schema name.
PackageName (see page 10770)	String	Package name.
ProcedureName (see page 10771)	String	Procedure name from package.
ParameterName (see page 10771)	String	Parameter name.
ParameterMode (see page 10770)	String	Parameter mode: IN, OUT, INOUT, RESULT.

TypeName (see page 10772)	String	Database specific data type name.
Precision (see page 10771)	Int32	Precision (see page 10771) of this type. For string data type, this is maximum number of characters. For decimal types, this is maximum number of digits, including digits from the scale.
Scale (see page 10771)	Int32	Scale (see page 10771) of a decimal type.
Ordinal (see page 10770)	Int32	Position in rows of table, starting with 1.
IsNullable (see page 10769)	Boolean	Can accept NULL values.
DbxDataType (see page 10769)	Int32	dbExpress data type.
IsFixedLength (see page 10769)	Boolean	Holds fixed length data.
IsUnicode (see page 10770)	Boolean	Can hold Unicode characters.
IsLong (see page 10769)	Boolean	Is BLOB data type.
IsUnsigned (see page 10770)	Boolean	Is unsigned numeric type.

See Also

TDBXMetaDataCommandsEx (see page 10531)

TDBXPackageProcedureParametersIndex (see page 10772)

56.1.19.1 TDBXPackageProcedureParametersColumns Members

The following tables list the members exposed by TDBXPackageProcedureParametersColumns.

Public Constants

	Name	Description
	CatalogName (see page 10768)	This is CatalogName, a member of class TDBXPackageProcedureParametersColumns.
	DbxDataType (see page 10769)	This is DbxDataType, a member of class TDBXPackageProcedureParametersColumns.
	IsFixedLength (see page 10769)	This is IsFixedLength, a member of class TDBXPackageProcedureParametersColumns.
	IsLong (see page 10769)	This is IsLong, a member of class TDBXPackageProcedureParametersColumns.
	IsNullable (see page 10769)	This is IsNullable, a member of class TDBXPackageProcedureParametersColumns.
	IsUnicode (see page 10770)	This is IsUnicode, a member of class TDBXPackageProcedureParametersColumns.
	IsUnsigned (see page 10770)	This is IsUnsigned, a member of class TDBXPackageProcedureParametersColumns.
	Ordinal (see page 10770)	This is Ordinal, a member of class TDBXPackageProcedureParametersColumns.
	PackageName (see page 10770)	This is PackageName, a member of class TDBXPackageProcedureParametersColumns.
	ParameterMode (see page 10770)	This is ParameterMode, a member of class TDBXPackageProcedureParametersColumns.

	ParameterName (see page 10771)	This is ParameterName, a member of class TDBXPackageProcedureParametersColumns.
	Precision (see page 10771)	This is Precision, a member of class TDBXPackageProcedureParametersColumns.
	ProcedureName (see page 10771)	This is ProcedureName, a member of class TDBXPackageProcedureParametersColumns.
	Scale (see page 10771)	This is Scale, a member of class TDBXPackageProcedureParametersColumns.
	SchemaName (see page 10772)	This is SchemaName, a member of class TDBXPackageProcedureParametersColumns.
	TypeName (see page 10772)	This is TypeName, a member of class TDBXPackageProcedureParametersColumns.

56.1.19.2 TDBXPackageProcedureParametersColumns Constants

The constants of the TDBXPackageProcedureParametersColumns class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10768)	This is CatalogName, a member of class TDBXPackageProcedureParametersColumns.
	DbxDataType (see page 10769)	This is DbxDataType, a member of class TDBXPackageProcedureParametersColumns.
	IsFixedLength (see page 10769)	This is IsFixedLength, a member of class TDBXPackageProcedureParametersColumns.
	IsLong (see page 10769)	This is IsLong, a member of class TDBXPackageProcedureParametersColumns.
	IsNullable (see page 10769)	This is IsNullable, a member of class TDBXPackageProcedureParametersColumns.
	IsUnicode (see page 10770)	This is IsUnicode, a member of class TDBXPackageProcedureParametersColumns.
	IsUnsigned (see page 10770)	This is IsUnsigned, a member of class TDBXPackageProcedureParametersColumns.
	Ordinal (see page 10770)	This is Ordinal, a member of class TDBXPackageProcedureParametersColumns.
	PackageName (see page 10770)	This is PackageName, a member of class TDBXPackageProcedureParametersColumns.
	ParameterMode (see page 10770)	This is ParameterMode, a member of class TDBXPackageProcedureParametersColumns.
	ParameterName (see page 10771)	This is ParameterName, a member of class TDBXPackageProcedureParametersColumns.
	Precision (see page 10771)	This is Precision, a member of class TDBXPackageProcedureParametersColumns.
	ProcedureName (see page 10771)	This is ProcedureName, a member of class TDBXPackageProcedureParametersColumns.
	Scale (see page 10771)	This is Scale, a member of class TDBXPackageProcedureParametersColumns.
	SchemaName (see page 10772)	This is SchemaName, a member of class TDBXPackageProcedureParametersColumns.
	TypeName (see page 10772)	This is TypeName, a member of class TDBXPackageProcedureParametersColumns.

56.1.19.2.1 TDBXPackageProcedureParametersColumns.CatalogName Constant

Pascal

```
const CatalogName = 'CatalogName';
```

C++

```
CatalogName = 'CatalogName';
```

Description

This is CatalogName, a member of class TDBXPackageProcedureParametersColumns.

56.1.19.2.2 TDBXPackageProcedureParametersColumns.DbxDataType Constant

Pascal

```
const DbxDataType = 'DbxDataType';
```

C++

```
DbxDataType = 'DbxDataType';
```

Description

This is DbxDataType, a member of class TDBXPackageProcedureParametersColumns.

56.1.19.2.3 TDBXPackageProcedureParametersColumns.IsFixedLength Constant

Pascal

```
const IsFixedLength = 'IsFixedLength';
```

C++

```
IsFixedLength = 'IsFixedLength';
```

Description

This is IsFixedLength, a member of class TDBXPackageProcedureParametersColumns.

56.1.19.2.4 TDBXPackageProcedureParametersColumns.IsLong Constant

Pascal

```
const IsLong = 'IsLong';
```

C++

```
IsLong = 'IsLong';
```

Description

This is IsLong, a member of class TDBXPackageProcedureParametersColumns.

56.1.19.2.5 TDBXPackageProcedureParametersColumns.IsNullable Constant

Pascal

```
const IsNullable = 'IsNullable';
```

C++

```
IsNullable = 'IsNullable';
```

Description

This is IsNullable, a member of class TDBXPackageProcedureParametersColumns.

56.1.19.2.6 TDBXPackageProcedureParametersColumns.IsUnicode Constant

Pascal

```
const IsUnicode = 'IsUnicode';
```

C++

```
IsUnicode = 'IsUnicode';
```

Description

This is IsUnicode, a member of class TDBXPackageProcedureParametersColumns.

56.1.19.2.7 TDBXPackageProcedureParametersColumns.IsUnsigned Constant

Pascal

```
const IsUnsigned = 'IsUnsigned';
```

C++

```
IsUnsigned = 'IsUnsigned';
```

Description

This is IsUnsigned, a member of class TDBXPackageProcedureParametersColumns.

56.1.19.2.8 TDBXPackageProcedureParametersColumns.Ordinal Constant

Pascal

```
const Ordinal = 'Ordinal';
```

C++

```
Ordinal = 'Ordinal';
```

Description

This is Ordinal, a member of class TDBXPackageProcedureParametersColumns.

56.1.19.2.9 TDBXPackageProcedureParametersColumns.PackageName Constant

Pascal

```
const PackageName = 'PackageName';
```

C++

```
PackageName = 'PackageName';
```

Description

This is PackageName, a member of class TDBXPackageProcedureParametersColumns.

56.1.19.2.10 TDBXPackageProcedureParametersColumns.ParameterMode Constant

Pascal

```
const ParameterMode = 'ParameterMode';
```

C++

```
ParameterMode = 'ParameterMode';
```

Description

This is ParameterMode, a member of class TDBXPackageProcedureParametersColumns.

56.1.19.2.11 TDBXPackageProcedureParametersColumns.ParameterName Constant

Pascal

```
const ParameterName = 'ParameterName';
```

C++

```
ParameterName = 'ParameterName';
```

Description

This is ParameterName, a member of class TDBXPackageProcedureParametersColumns.

56.1.19.2.12 TDBXPackageProcedureParametersColumns.Precision Constant

Pascal

```
const Precision = 'Precision';
```

C++

```
Precision = 'Precision';
```

Description

This is Precision, a member of class TDBXPackageProcedureParametersColumns.

56.1.19.2.13 TDBXPackageProcedureParametersColumns.ProcedureName Constant

Pascal

```
const ProcedureName = 'ProcedureName';
```

C++

```
ProcedureName = 'ProcedureName';
```

Description

This is ProcedureName, a member of class TDBXPackageProcedureParametersColumns.

56.1.19.2.14 TDBXPackageProcedureParametersColumns.Scale Constant

Pascal

```
const Scale = 'Scale';
```

C++

```
Scale = 'Scale';
```

Description

This is Scale, a member of class TDBXPackageProcedureParametersColumns.

56.1.19.2.15 TDBXPackageProcedureParametersColumns.SchemaName Constant

Pascal

```
const SchemaName = 'SchemaName';
```

C++

```
SchemaName = 'SchemaName';
```

Description

This is SchemaName, a member of class TDBXPackageProcedureParametersColumns.

56.1.19.2.16 TDBXPackageProcedureParametersColumns.TypeName Constant

Pascal

```
const TypeName = 'TypeName';
```

C++

```
TypeName = 'TypeName';
```

Description

This is TypeName, a member of class TDBXPackageProcedureParametersColumns.

56.1.20 TDBXPackageProcedureParametersIndex Class

Accesses columns by ordinal in the **PackageProcedureParameters** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXPackageProcedureParametersIndex

Pascal

```
TDBXPackageProcedureParametersIndex = class;
```

C++

```
class TDBXPackageProcedureParametersIndex;
```

File

DBXMetaDataNames

Description

TDBXPackageProcedureParametersIndex is used to access columns by ordinal in the **PackageProcedureParameters** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetPackageProcedureParameters when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Ordinal (see page 10776) Name	Data Type of Column Data	Description
CatalogName (see page 10775)	String	Catalog name.
SchemaName (see page 10778)	String	Schema name.
PackageName (see page 10777)	String	Package name.
ProcedureName (see page 10778)	String	Procedure name from package.
ParameterName (see page 10777)	String	Parameter name.
ParameterMode (see page 10777)	String	Parameter mode: IN, OUT, INOUT, RESULT.
TypeName (see page 10778)	String	Database specific data type name.
Precision (see page 10777)	Int32	Precision (see page 10777) of this type. For string data type, this is maximum number of characters. For decimal types, this is maximum number of digits, including digits from the scale.
Scale (see page 10778)	Int32	Scale (see page 10778) of a decimal type.
Ordinal (see page 10776)	Int32	Position in rows of table, starting with 1.
IsNullable (see page 10776)	Boolean	Can accept NULL values.
DbxDataType (see page 10775)	Int32	dbExpress data type.
IsFixedLength (see page 10775)	Boolean	Holds fixed length data.
IsUnicode (see page 10776)	Boolean	Can hold Unicode characters.
IsLong (see page 10775)	Boolean	Is BLOB data type.
IsUnsigned (see page 10776)	Boolean	Is unsigned numeric type.

See Also

TDBXMetaDataCommandsEx (see page 10531)

TDBXPackageProcedureParametersColumns (see page 10766)

56.1.20.1 TDBXPackageProcedureParametersIndex Members

The following tables list the members exposed by TDBXPackageProcedureParametersIndex.

Public Constants

	Name	Description
	CatalogName (see page 10775)	This is CatalogName, a member of class TDBXPackageProcedureParametersIndex.
	DbxDataType (see page 10775)	This is DbxDataType, a member of class TDBXPackageProcedureParametersIndex.
	IsFixedLength (see page 10775)	This is IsFixedLength, a member of class TDBXPackageProcedureParametersIndex.
	IsLong (see page 10775)	This is IsLong, a member of class TDBXPackageProcedureParametersIndex.
	IsNullable (see page 10776)	This is IsNullable, a member of class TDBXPackageProcedureParametersIndex.
	IsUnicode (see page 10776)	This is IsUnicode, a member of class TDBXPackageProcedureParametersIndex.
	IsUnsigned (see page 10776)	This is IsUnsigned, a member of class TDBXPackageProcedureParametersIndex.
	Last (see page 10776)	This is Last, a member of class TDBXPackageProcedureParametersIndex.
	Ordinal (see page 10776)	This is Ordinal, a member of class TDBXPackageProcedureParametersIndex.
	PackageName (see page 10777)	This is PackageName, a member of class TDBXPackageProcedureParametersIndex.
	ParameterMode (see page 10777)	This is ParameterMode, a member of class TDBXPackageProcedureParametersIndex.
	ParameterName (see page 10777)	This is ParameterName, a member of class TDBXPackageProcedureParametersIndex.
	Precision (see page 10777)	This is Precision, a member of class TDBXPackageProcedureParametersIndex.
	ProcedureName (see page 10778)	This is ProcedureName, a member of class TDBXPackageProcedureParametersIndex.
	Scale (see page 10778)	This is Scale, a member of class TDBXPackageProcedureParametersIndex.
	SchemaName (see page 10778)	This is SchemaName, a member of class TDBXPackageProcedureParametersIndex.
	TypeName (see page 10778)	This is TypeName, a member of class TDBXPackageProcedureParametersIndex.

56.1.20.2 TDBXPackageProcedureParametersIndex Constants

The constants of the TDBXPackageProcedureParametersIndex class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10775)	This is CatalogName, a member of class TDBXPackageProcedureParametersIndex.
	DbxDataType (see page 10775)	This is DbxDataType, a member of class TDBXPackageProcedureParametersIndex.
	IsFixedLength (see page 10775)	This is IsFixedLength, a member of class TDBXPackageProcedureParametersIndex.
	IsLong (see page 10775)	This is IsLong, a member of class TDBXPackageProcedureParametersIndex.
	IsNullable (see page 10776)	This is IsNullable, a member of class TDBXPackageProcedureParametersIndex.
	IsUnicode (see page 10776)	This is IsUnicode, a member of class TDBXPackageProcedureParametersIndex.
	IsUnsigned (see page 10776)	This is IsUnsigned, a member of class TDBXPackageProcedureParametersIndex.
	Last (see page 10776)	This is Last, a member of class TDBXPackageProcedureParametersIndex.
	Ordinal (see page 10776)	This is Ordinal, a member of class TDBXPackageProcedureParametersIndex.
	PackageName (see page 10777)	This is PackageName, a member of class TDBXPackageProcedureParametersIndex.
	ParameterMode (see page 10777)	This is ParameterMode, a member of class TDBXPackageProcedureParametersIndex.
	ParameterName (see page 10777)	This is ParameterName, a member of class TDBXPackageProcedureParametersIndex.
	Precision (see page 10777)	This is Precision, a member of class TDBXPackageProcedureParametersIndex.
	ProcedureName (see page 10778)	This is ProcedureName, a member of class TDBXPackageProcedureParametersIndex.
	Scale (see page 10778)	This is Scale, a member of class TDBXPackageProcedureParametersIndex.

	SchemaName (see page 10778)	This is SchemaName, a member of class TDBXPackageProcedureParametersIndex.
	TypeName (see page 10778)	This is TypeName, a member of class TDBXPackageProcedureParametersIndex.

56.1.20.2.1 TDBXPackageProcedureParametersIndex.CatalogName Constant

Pascal

```
const CatalogName = 0;
```

C++

```
CatalogName = 0;
```

Description

This is CatalogName, a member of class TDBXPackageProcedureParametersIndex.

56.1.20.2.2 TDBXPackageProcedureParametersIndex.DbxDataType Constant

Pascal

```
const DbxDataType = 11;
```

C++

```
DbxDataType = 11;
```

Description

This is DbxDataType, a member of class TDBXPackageProcedureParametersIndex.

56.1.20.2.3 TDBXPackageProcedureParametersIndex.IsFixedLength Constant

Pascal

```
const IsFixedLength = 12;
```

C++

```
IsFixedLength = 12;
```

Description

This is IsFixedLength, a member of class TDBXPackageProcedureParametersIndex.

56.1.20.2.4 TDBXPackageProcedureParametersIndex.IsLong Constant

Pascal

```
const IsLong = 14;
```

C++

```
IsLong = 14;
```

Description

This is IsLong, a member of class TDBXPackageProcedureParametersIndex.

56.1.20.2.5 TDBXPackageProcedureParametersIndex.IsNullable Constant

Pascal

```
const IsNullable = 10;
```

C++

```
IsNullable = 10;
```

Description

This is IsNullable, a member of class TDBXPackageProcedureParametersIndex.

56.1.20.2.6 TDBXPackageProcedureParametersIndex.IsUnicode Constant

Pascal

```
const IsUnicode = 13;
```

C++

```
IsUnicode = 13;
```

Description

This is IsUnicode, a member of class TDBXPackageProcedureParametersIndex.

56.1.20.2.7 TDBXPackageProcedureParametersIndex.IsUnsigned Constant

Pascal

```
const IsUnsigned = 15;
```

C++

```
IsUnsigned = 15;
```

Description

This is IsUnsigned, a member of class TDBXPackageProcedureParametersIndex.

56.1.20.2.8 TDBXPackageProcedureParametersIndex.Last Constant

Pascal

```
const Last = 15;
```

C++

```
Last = 15;
```

Description

This is Last, a member of class TDBXPackageProcedureParametersIndex.

56.1.20.2.9 TDBXPackageProcedureParametersIndex.Ordinal Constant

Pascal

```
const Ordinal = 9;
```

C++

```
Ordinal = 9;
```

Description

This is Ordinal, a member of class TDBXPackageProcedureParametersIndex.

56.1.20.2.10 TDBXPackageProcedureParametersIndex.PackageName Constant

Pascal

```
const PackageName = 2;
```

C++

```
PackageName = 2;
```

Description

This is PackageName, a member of class TDBXPackageProcedureParametersIndex.

56.1.20.2.11 TDBXPackageProcedureParametersIndex.ParameterMode Constant

Pascal

```
const ParameterMode = 5;
```

C++

```
ParameterMode = 5;
```

Description

This is ParameterMode, a member of class TDBXPackageProcedureParametersIndex.

56.1.20.2.12 TDBXPackageProcedureParametersIndex.ParameterName Constant

Pascal

```
const ParameterName = 4;
```

C++

```
ParameterName = 4;
```

Description

This is ParameterName, a member of class TDBXPackageProcedureParametersIndex.

56.1.20.2.13 TDBXPackageProcedureParametersIndex.Precision Constant

Pascal

```
const Precision = 7;
```

C++

```
Precision = 7;
```

Description

This is Precision, a member of class TDBXPackageProcedureParametersIndex.

56.1.20.2.14 TDBXPackageProcedureParametersIndex.ProcedureName Constant

Pascal

```
const ProcedureName = 3;
```

C++

```
ProcedureName = 3;
```

Description

This is ProcedureName, a member of class TDBXPackageProcedureParametersIndex.

56.1.20.2.15 TDBXPackageProcedureParametersIndex.Scale Constant

Pascal

```
const Scale = 8;
```

C++

```
Scale = 8;
```

Description

This is Scale, a member of class TDBXPackageProcedureParametersIndex.

56.1.20.2.16 TDBXPackageProcedureParametersIndex.SchemaName Constant

Pascal

```
const SchemaName = 1;
```

C++

```
SchemaName = 1;
```

Description

This is SchemaName, a member of class TDBXPackageProcedureParametersIndex.

56.1.20.2.17 TDBXPackageProcedureParametersIndex.TypeName Constant

Pascal

```
const TypeName = 6;
```

C++

```
TypeName = 6;
```

Description

This is TypeName, a member of class TDBXPackageProcedureParametersIndex.

56.1.21 TDBXPackageProceduresColumns Class

Accesses columns by name in the **PackageProcedures** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXPackageProceduresColumns

Pascal

```
TDBXPackageProceduresColumns = class;
```

C++

```
class TDBXPackageProceduresColumns;
```

File

DBXMetaDataNames

Description

TDBXPackageProceduresColumns is used to access columns by name in the **PackageProcedures** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetPackageProcedures when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Name	Data Type of Column Data	Description
CatalogName (see page 10780)	String	Catalog name.
SchemaName (see page 10781)	String	Schema name.
PackageName (see page 10780)	String	Package name.
ProcedureName (see page 10780)	String	Procedure name from package.
ProcedureType (see page 10780)	String	Procedure type: FUNCTION, PROCEDURE.

See Also

TDBXMetaDataCommandsEx (see page 10531)

TDBXPackageProceduresIndex (see page 10781)

56.1.21.1 TDBXPackageProceduresColumns Members

The following tables list the members exposed by TDBXPackageProceduresColumns.

Public Constants

	Name	Description
	CatalogName (see page 10780)	This is CatalogName, a member of class TDBXPackageProceduresColumns.
	PackageName (see page 10780)	This is PackageName, a member of class TDBXPackageProceduresColumns.
	ProcedureName (see page 10780)	This is ProcedureName, a member of class TDBXPackageProceduresColumns.
	ProcedureType (see page 10780)	This is ProcedureType, a member of class TDBXPackageProceduresColumns.
	SchemaName (see page 10781)	This is SchemaName, a member of class TDBXPackageProceduresColumns.

56.1.21.2 TDBXPackageProceduresColumns Constants

The constants of the TDBXPackageProceduresColumns class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10780)	This is CatalogName, a member of class TDBXPackageProceduresColumns.
	PackageName (see page 10780)	This is PackageName, a member of class TDBXPackageProceduresColumns.
	ProcedureName (see page 10780)	This is ProcedureName, a member of class TDBXPackageProceduresColumns.
	ProcedureType (see page 10780)	This is ProcedureType, a member of class TDBXPackageProceduresColumns.
	SchemaName (see page 10781)	This is SchemaName, a member of class TDBXPackageProceduresColumns.

56.1.21.2.1 TDBXPackageProceduresColumns.CatalogName Constant

Pascal

```
const CatalogName = 'CatalogName';
```

C++

```
CatalogName = 'CatalogName';
```

Description

This is CatalogName, a member of class TDBXPackageProceduresColumns.

56.1.21.2.2 TDBXPackageProceduresColumns.PackageName Constant

Pascal

```
const PackageName = 'PackageName';
```

C++

```
PackageName = 'PackageName';
```

Description

This is PackageName, a member of class TDBXPackageProceduresColumns.

56.1.21.2.3 TDBXPackageProceduresColumns.ProcedureName Constant

Pascal

```
const ProcedureName = 'ProcedureName';
```

C++

```
ProcedureName = 'ProcedureName';
```

Description

This is ProcedureName, a member of class TDBXPackageProceduresColumns.

56.1.21.2.4 TDBXPackageProceduresColumns.ProcedureType Constant

Pascal

```
const ProcedureType = 'ProcedureType';
```

C++

```
ProcedureType = 'ProcedureType';
```

Description

This is ProcedureType, a member of class TDBXPackageProceduresColumns.

56.1.21.2.5 TDBXPackageProceduresColumns.SchemaName Constant

Pascal

```
const SchemaName = 'SchemaName';
```

C++

```
SchemaName = 'SchemaName';
```

Description

This is SchemaName, a member of class TDBXPackageProceduresColumns.

56.1.22 TDBXPackageProceduresIndex Class

Accesses columns by ordinal in the **PackageProcedures** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXPackageProceduresIndex

Pascal

```
TDBXPackageProceduresIndex = class;
```

C++

```
class TDBXPackageProceduresIndex;
```

File

DBXMetaDataNames

Description

TDBXPackageProceduresIndex is used to access columns by ordinal in the **PackageProcedures** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetPackageProcedures when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Ordinal Name	Data Type of Column Data	Description
CatalogName (see page 10782)	String	Catalog name.
SchemaName (see page 10783)	String	Schema name.
PackageName (see page 10783)	String	Package name.
ProcedureName (see page 10783)	String	Procedure name from package.
ProcedureType (see page 10783)	String	Procedure type: FUNCTION, PROCEDURE.

See Also

TDBXMetaDataCommandsEx (see page 10531)

TDBXPackageProceduresColumns (see page 10778)

56.1.22.1 TDBXPackageProceduresIndex Members

The following tables list the members exposed by TDBXPackageProceduresIndex.

Public Constants

	Name	Description
	CatalogName (see page 10782)	This is CatalogName, a member of class TDBXPackageProceduresIndex.
	Last (see page 10782)	This is Last, a member of class TDBXPackageProceduresIndex.
	PackageName (see page 10783)	This is PackageName, a member of class TDBXPackageProceduresIndex.
	ProcedureName (see page 10783)	This is ProcedureName, a member of class TDBXPackageProceduresIndex.
	ProcedureType (see page 10783)	This is ProcedureType, a member of class TDBXPackageProceduresIndex.
	SchemaName (see page 10783)	This is SchemaName, a member of class TDBXPackageProceduresIndex.

56.1.22.2 TDBXPackageProceduresIndex Constants

The constants of the TDBXPackageProceduresIndex class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10782)	This is CatalogName, a member of class TDBXPackageProceduresIndex.
	Last (see page 10782)	This is Last, a member of class TDBXPackageProceduresIndex.
	PackageName (see page 10783)	This is PackageName, a member of class TDBXPackageProceduresIndex.
	ProcedureName (see page 10783)	This is ProcedureName, a member of class TDBXPackageProceduresIndex.
	ProcedureType (see page 10783)	This is ProcedureType, a member of class TDBXPackageProceduresIndex.
	SchemaName (see page 10783)	This is SchemaName, a member of class TDBXPackageProceduresIndex.

56.1.22.2.1 TDBXPackageProceduresIndex.CatalogName Constant

Pascal

```
const CatalogName = 0;
```

C++

```
CatalogName = 0;
```

Description

This is CatalogName, a member of class TDBXPackageProceduresIndex.

56.1.22.2.2 TDBXPackageProceduresIndex.Last Constant

Pascal

```
const Last = 4;
```

C++

```
Last = 4;
```

Description

This is Last, a member of class TDBXPackageProceduresIndex.

56.1.22.2.3 TDBXPackageProceduresIndex.PackageName Constant

Pascal

```
const PackageName = 2;
```

C++

```
PackageName = 2;
```

Description

This is PackageName, a member of class TDBXPackageProceduresIndex.

56.1.22.2.4 TDBXPackageProceduresIndex.ProcedureName Constant

Pascal

```
const ProcedureName = 3;
```

C++

```
ProcedureName = 3;
```

Description

This is ProcedureName, a member of class TDBXPackageProceduresIndex.

56.1.22.2.5 TDBXPackageProceduresIndex.ProcedureType Constant

Pascal

```
const ProcedureType = 4;
```

C++

```
ProcedureType = 4;
```

Description

This is ProcedureType, a member of class TDBXPackageProceduresIndex.

56.1.22.2.6 TDBXPackageProceduresIndex.SchemaName Constant

Pascal

```
const SchemaName = 1;
```

C++

```
SchemaName = 1;
```

Description

This is SchemaName, a member of class TDBXPackageProceduresIndex.

56.1.23 TDBXPackagesColumns Class

Accesses columns by name in the **Packages** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXPackagesColumns

Pascal

```
TDBXPackagesColumns = class;
```

C++

```
class TDBXPackagesColumns;
```

File

DBXMetaDataNames

Description

TDBXPackagesColumns is used to access columns by name in the **Packages** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetPackages when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Name	Data Type of Column Data	Description
CatalogName (see page 10785)	String	Catalog name.
SchemaName (see page 10785)	String	Schema name.
PackageName (see page 10785)	String	Package name.

See Also

TDBXMetaDataCommands (see page 10525)

TDBXPackagesIndex (see page 10785)

56.1.23.1 TDBXPackagesColumns Members

The following tables list the members exposed by TDBXPackagesColumns.

Public Constants

	Name	Description
	CatalogName (see page 10785)	This is CatalogName, a member of class TDBXPackagesColumns.
	PackageName (see page 10785)	This is PackageName, a member of class TDBXPackagesColumns.
	SchemaName (see page 10785)	This is SchemaName, a member of class TDBXPackagesColumns.

56.1.23.2 TDBXPackagesColumns Constants

The constants of the TDBXPackagesColumns class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10785)	This is CatalogName, a member of class TDBXPackagesColumns.
	PackageName (see page 10785)	This is PackageName, a member of class TDBXPackagesColumns.
	SchemaName (see page 10785)	This is SchemaName, a member of class TDBXPackagesColumns.

56.1.23.2.1 TDBXPackagesColumns.CatalogName Constant

Pascal

```
const CatalogName = 'CatalogName';
```

C++

```
CatalogName = 'CatalogName';
```

Description

This is CatalogName, a member of class TDBXPackagesColumns.

56.1.23.2.2 TDBXPackagesColumns.PackageName Constant

Pascal

```
const PackageName = 'PackageName';
```

C++

```
PackageName = 'PackageName';
```

Description

This is PackageName, a member of class TDBXPackagesColumns.

56.1.23.2.3 TDBXPackagesColumns.SchemaName Constant

Pascal

```
const SchemaName = 'SchemaName';
```

C++

```
SchemaName = 'SchemaName';
```

Description

This is SchemaName, a member of class TDBXPackagesColumns.

56.1.24 TDBXPackagesIndex Class

Accesses columns by ordinal in the **Packages** metadata.

Class Hierarchy

Pascal

```
TDBXPackagesIndex = class;
```

C++

```
class TDBXPackagesIndex;
```

File

DBXMetaDataNames

Description

TDBXPackagesIndex is used to access columns by ordinal in the **Packages** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetPackages when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Ordinal Name	Data Type of Column Data	Description
CatalogName (see page 10786)	String	Catalog name.
SchemaName (see page 10787)	String	Schema name.
PackageName (see page 10787)	String	Package name.

See Also

TDBXMetaDataCommands (see page 10525)

TDBXPackagesColumns (see page 10783)

56.1.24.1 TDBXPackagesIndex Members

The following tables list the members exposed by TDBXPackagesIndex.

Public Constants

	Name	Description
	CatalogName (see page 10786)	This is CatalogName, a member of class TDBXPackagesIndex.
	Last (see page 10787)	This is Last, a member of class TDBXPackagesIndex.
	PackageName (see page 10787)	This is PackageName, a member of class TDBXPackagesIndex.
	SchemaName (see page 10787)	This is SchemaName, a member of class TDBXPackagesIndex.

56.1.24.2 TDBXPackagesIndex Constants

The constants of the TDBXPackagesIndex class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10786)	This is CatalogName, a member of class TDBXPackagesIndex.
	Last (see page 10787)	This is Last, a member of class TDBXPackagesIndex.
	PackageName (see page 10787)	This is PackageName, a member of class TDBXPackagesIndex.
	SchemaName (see page 10787)	This is SchemaName, a member of class TDBXPackagesIndex.

56.1.24.2.1 TDBXPackagesIndex.CatalogName Constant

Pascal

```
const CatalogName = 0;
```

C++

```
CatalogName = 0;
```

Description

This is CatalogName, a member of class TDBXPackagesIndex.

56.1.24.2.2 TDBXPackagesIndex.Last Constant

Pascal

```
const Last = 2;
```

C++

```
Last = 2;
```

Description

This is Last, a member of class TDBXPackagesIndex.

56.1.24.2.3 TDBXPackagesIndex.PackageName Constant

Pascal

```
const PackageName = 2;
```

C++

```
PackageName = 2;
```

Description

This is PackageName, a member of class TDBXPackagesIndex.

56.1.24.2.4 TDBXPackagesIndex.SchemaName Constant

Pascal

```
const SchemaName = 1;
```

C++

```
SchemaName = 1;
```

Description

This is SchemaName, a member of class TDBXPackagesIndex.

56.1.25 TDBXPackageSourcesColumns Class

Accesses columns by name in the **PackageSources** metadata.

Class Hierarchy

Pascal

```
TDBXPackageSourcesColumns = class;
```

C++

```
class TDBXPackageSourcesColumns;
```

File

DBXMetaDataNames

Description

TDBXPackageSourcesColumns is used to access columns by name in the **PackageSources** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetPackageSources when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Name	Data Type of Column Data	Description
CatalogName (see page 10788)	String	Catalog name.
SchemaName (see page 10789)	String	Schema name.
PackageName (see page 10789)	String	Package name.
Definition (see page 10789)	String	Package source code.

See Also

TDBXMetaDataCommandsEx (see page 10531)

TDBXPackageSourcesIndex (see page 10789)

56.1.25.1 TDBXPackageSourcesColumns Members

The following tables list the members exposed by TDBXPackageSourcesColumns.

Public Constants

	Name	Description
	CatalogName (see page 10788)	This is CatalogName, a member of class TDBXPackageSourcesColumns.
	Definition (see page 10789)	This is Definition, a member of class TDBXPackageSourcesColumns.
	PackageName (see page 10789)	This is PackageName, a member of class TDBXPackageSourcesColumns.
	SchemaName (see page 10789)	This is SchemaName, a member of class TDBXPackageSourcesColumns.

56.1.25.2 TDBXPackageSourcesColumns Constants

The constants of the TDBXPackageSourcesColumns class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10788)	This is CatalogName, a member of class TDBXPackageSourcesColumns.
	Definition (see page 10789)	This is Definition, a member of class TDBXPackageSourcesColumns.
	PackageName (see page 10789)	This is PackageName, a member of class TDBXPackageSourcesColumns.
	SchemaName (see page 10789)	This is SchemaName, a member of class TDBXPackageSourcesColumns.

56.1.25.2.1 TDBXPackageSourcesColumns.CatalogName Constant

Pascal

```
const CatalogName = 'CatalogName';
```

C++

```
CatalogName = 'CatalogName';
```

Description

This is CatalogName, a member of class TDBXPackageSourcesColumns.

56.1.25.2.2 TDBXPackageSourcesColumns.Definition Constant

Pascal

```
const Definition = 'Definition';
```

C++

```
Definition = 'Definition';
```

Description

This is Definition, a member of class TDBXPackageSourcesColumns.

56.1.25.2.3 TDBXPackageSourcesColumns.PackageName Constant

Pascal

```
const PackageName = 'PackageName';
```

C++

```
PackageName = 'PackageName';
```

Description

This is PackageName, a member of class TDBXPackageSourcesColumns.

56.1.25.2.4 TDBXPackageSourcesColumns.SchemaName Constant

Pascal

```
const SchemaName = 'SchemaName';
```

C++

```
SchemaName = 'SchemaName';
```

Description

This is SchemaName, a member of class TDBXPackageSourcesColumns.

56.1.26 TDBXPackageSourcesIndex Class

Accesses columns by ordinal in the **PackageSources** metadata.

Class Hierarchy

Pascal

```
TDBXPackageSourcesIndex = class;
```

C++

```
class TDBXPackageSourcesIndex;
```

File

DBXMetaDataNames

Description

TDBXPackageSourcesIndex is used to access columns by ordinal in the **PackageSources** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetPackageSources when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Ordinal Name	Data Type of Column Data	Description
CatalogName (see page 10790)	String	Catalog name.
SchemaName (see page 10791)	String	Schema name.
PackageName (see page 10791)	String	Package name.
Definition (see page 10791)	String	Package source code.

See Also

TDBXMetaDataCommandsEx (see page 10531)

TDBXPackageSourcesColumns (see page 10787)

56.1.26.1 TDBXPackageSourcesIndex Members

The following tables list the members exposed by TDBXPackageSourcesIndex.

Public Constants

	Name	Description
	CatalogName (see page 10790)	This is CatalogName, a member of class TDBXPackageSourcesIndex.
	Definition (see page 10791)	This is Definition, a member of class TDBXPackageSourcesIndex.
	Last (see page 10791)	This is Last, a member of class TDBXPackageSourcesIndex.
	PackageName (see page 10791)	This is PackageName, a member of class TDBXPackageSourcesIndex.
	SchemaName (see page 10791)	This is SchemaName, a member of class TDBXPackageSourcesIndex.

56.1.26.2 TDBXPackageSourcesIndex Constants

The constants of the TDBXPackageSourcesIndex class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10790)	This is CatalogName, a member of class TDBXPackageSourcesIndex.
	Definition (see page 10791)	This is Definition, a member of class TDBXPackageSourcesIndex.
	Last (see page 10791)	This is Last, a member of class TDBXPackageSourcesIndex.
	PackageName (see page 10791)	This is PackageName, a member of class TDBXPackageSourcesIndex.
	SchemaName (see page 10791)	This is SchemaName, a member of class TDBXPackageSourcesIndex.

56.1.26.2.1 TDBXPackageSourcesIndex.CatalogName Constant

Pascal

```
const CatalogName = 0;
```

C++

```
CatalogName = 0;
```

Description

This is CatalogName, a member of class TDBXPackageSourcesIndex.

56.1.26.2.2 TDBXPackageSourcesIndex.Definition Constant

Pascal

```
const Definition = 3;
```

C++

```
Definition = 3;
```

Description

This is Definition, a member of class TDBXPackageSourcesIndex.

56.1.26.2.3 TDBXPackageSourcesIndex.Last Constant

Pascal

```
const Last = 3;
```

C++

```
Last = 3;
```

Description

This is Last, a member of class TDBXPackageSourcesIndex.

56.1.26.2.4 TDBXPackageSourcesIndex.PackageName Constant

Pascal

```
const PackageName = 2;
```

C++

```
PackageName = 2;
```

Description

This is PackageName, a member of class TDBXPackageSourcesIndex.

56.1.26.2.5 TDBXPackageSourcesIndex.SchemaName Constant

Pascal

```
const SchemaName = 1;
```

C++

```
SchemaName = 1;
```

Description

This is SchemaName, a member of class TDBXPackageSourcesIndex.

56.1.27 TDBXProcedureParametersColumns Class

Accesses columns by name in the **ProcedureParameters** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXProcedureParametersColumns

Pascal

```
TDBXProcedureParametersColumns = class;
```

C++

```
class TDBXProcedureParametersColumns;
```

File

DBXMetaDataNames

Description

TDBXProcedureParametersColumns is used to access columns by name in the **ProcedureParameters** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetProcedureParameters when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Name	Data Type of Column Data	Description
CatalogName (see page 10794)	String	Catalog name.
SchemaName (see page 10797)	String	Schema name.
ProcedureName (see page 10796)	String	Procedure name.
ParameterName (see page 10796)	String	Parameter name.
ParameterMode (see page 10796)	String	Parameter mode: IN, OUT, INOUT, RESULT.
TypeName (see page 10797)	String	Database specific data type name.
Precision (see page 10796)	Int32	Precision (see page 10796) of this type. For string data type, this is maximum number of characters. For decimal types, this is maximum number of digits, including digits from the scale.
Scale (see page 10796)	Int32	Scale (see page 10796) of a decimal type.
Ordinal (see page 10795)	Int32	Position in rows of table, starting with 1.
IsNullable (see page 10795)	Boolean	Can accept NULL values.

DbxDataType (see page 10794)	Int32	dbExpress data type.
IsFixedLength (see page 10794)	Boolean	Holds fixed length data.
IsUnicode (see page 10795)	Boolean	Can hold Unicode characters.
IsLong (see page 10794)	Boolean	Is BLOB data type.
IsUnsigned (see page 10795)	Boolean	Is unsigned numeric type.

See Also

TDBXMetaDataCommands (see page 10525)

TDBXProcedureParametersIndex (see page 10797)

56.1.27.1 TDBXProcedureParametersColumns Members

The following tables list the members exposed by TDBXProcedureParametersColumns.

Public Constants

	Name	Description
	CatalogName (see page 10794)	This is CatalogName, a member of class TDBXProcedureParametersColumns.
	DbxDataType (see page 10794)	This is DbxDataType, a member of class TDBXProcedureParametersColumns.
	IsFixedLength (see page 10794)	This is IsFixedLength, a member of class TDBXProcedureParametersColumns.
	IsLong (see page 10794)	This is IsLong, a member of class TDBXProcedureParametersColumns.
	IsNullable (see page 10795)	This is IsNullable, a member of class TDBXProcedureParametersColumns.
	IsUnicode (see page 10795)	This is IsUnicode, a member of class TDBXProcedureParametersColumns.
	IsUnsigned (see page 10795)	This is IsUnsigned, a member of class TDBXProcedureParametersColumns.
	Ordinal (see page 10795)	This is Ordinal, a member of class TDBXProcedureParametersColumns.
	ParameterMode (see page 10796)	This is ParameterMode, a member of class TDBXProcedureParametersColumns.
	ParameterName (see page 10796)	This is ParameterName, a member of class TDBXProcedureParametersColumns.
	Precision (see page 10796)	This is Precision, a member of class TDBXProcedureParametersColumns.
	ProcedureName (see page 10796)	This is ProcedureName, a member of class TDBXProcedureParametersColumns.
	Scale (see page 10796)	This is Scale, a member of class TDBXProcedureParametersColumns.
	SchemaName (see page 10797)	This is SchemaName, a member of class TDBXProcedureParametersColumns.
	TypeName (see page 10797)	This is TypeName, a member of class TDBXProcedureParametersColumns.

56.1.27.2 TDBXProcedureParametersColumns Constants

The constants of the TDBXProcedureParametersColumns class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10794)	This is CatalogName, a member of class TDBXProcedureParametersColumns.
	DbxDataType (see page 10794)	This is DbxDataType, a member of class TDBXProcedureParametersColumns.
	IsFixedLength (see page 10794)	This is IsFixedLength, a member of class TDBXProcedureParametersColumns.
	IsLong (see page 10794)	This is IsLong, a member of class TDBXProcedureParametersColumns.
	IsNullable (see page 10795)	This is IsNullable, a member of class TDBXProcedureParametersColumns.
	IsUnicode (see page 10795)	This is IsUnicode, a member of class TDBXProcedureParametersColumns.

	IsUnsigned (see page 10795)	This is IsUnsigned, a member of class TDBXProcedureParametersColumns.
	Ordinal (see page 10795)	This is Ordinal, a member of class TDBXProcedureParametersColumns.
	ParameterMode (see page 10796)	This is ParameterMode, a member of class TDBXProcedureParametersColumns.
	ParameterName (see page 10796)	This is ParameterName, a member of class TDBXProcedureParametersColumns.
	Precision (see page 10796)	This is Precision, a member of class TDBXProcedureParametersColumns.
	ProcedureName (see page 10796)	This is ProcedureName, a member of class TDBXProcedureParametersColumns.
	Scale (see page 10796)	This is Scale, a member of class TDBXProcedureParametersColumns.
	SchemaName (see page 10797)	This is SchemaName, a member of class TDBXProcedureParametersColumns.
	TypeName (see page 10797)	This is TypeName, a member of class TDBXProcedureParametersColumns.

56.1.27.2.1 TDBXProcedureParametersColumns.CatalogName Constant

Pascal

```
const CatalogName = 'CatalogName';
```

C++

```
CatalogName = 'CatalogName';
```

Description

This is CatalogName, a member of class TDBXProcedureParametersColumns.

56.1.27.2.2 TDBXProcedureParametersColumns.DbxDataType Constant

Pascal

```
const DbxDataType = 'DbxDataType';
```

C++

```
DbxDataType = 'DbxDataType';
```

Description

This is DbxDataType, a member of class TDBXProcedureParametersColumns.

56.1.27.2.3 TDBXProcedureParametersColumns.IsFixedLength Constant

Pascal

```
const IsFixedLength = 'IsFixedLength';
```

C++

```
IsFixedLength = 'IsFixedLength';
```

Description

This is IsFixedLength, a member of class TDBXProcedureParametersColumns.

56.1.27.2.4 TDBXProcedureParametersColumns.IsLong Constant

Pascal

```
const IsLong = 'IsLong';
```

C++

```
IsLong = 'IsLong';
```

Description

This is IsLong, a member of class TDBXProcedureParametersColumns.

56.1.27.2.5 TDBXProcedureParametersColumns.IsNullable Constant

Pascal

```
const IsNullable = 'IsNullable';
```

C++

```
IsNullable = 'IsNullable';
```

Description

This is IsNullable, a member of class TDBXProcedureParametersColumns.

56.1.27.2.6 TDBXProcedureParametersColumns.IsUnicode Constant

Pascal

```
const IsUnicode = 'IsUnicode';
```

C++

```
IsUnicode = 'IsUnicode';
```

Description

This is IsUnicode, a member of class TDBXProcedureParametersColumns.

56.1.27.2.7 TDBXProcedureParametersColumns.IsUnsigned Constant

Pascal

```
const IsUnsigned = 'IsUnsigned';
```

C++

```
IsUnsigned = 'IsUnsigned';
```

Description

This is IsUnsigned, a member of class TDBXProcedureParametersColumns.

56.1.27.2.8 TDBXProcedureParametersColumns.Ordinal Constant

Pascal

```
const Ordinal = 'Ordinal';
```

C++

```
Ordinal = 'Ordinal';
```

Description

This is Ordinal, a member of class TDBXProcedureParametersColumns.

56.1.27.2.9 TDBXProcedureParametersColumns.ParameterMode Constant

Pascal

```
const ParameterMode = 'ParameterMode';
```

C++

```
ParameterMode = 'ParameterMode';
```

Description

This is ParameterMode, a member of class TDBXProcedureParametersColumns.

56.1.27.2.10 TDBXProcedureParametersColumns.ParameterName Constant

Pascal

```
const ParameterName = 'ParameterName';
```

C++

```
ParameterName = 'ParameterName';
```

Description

This is ParameterName, a member of class TDBXProcedureParametersColumns.

56.1.27.2.11 TDBXProcedureParametersColumns.Precision Constant

Pascal

```
const Precision = 'Precision';
```

C++

```
Precision = 'Precision';
```

Description

This is Precision, a member of class TDBXProcedureParametersColumns.

56.1.27.2.12 TDBXProcedureParametersColumns.ProcedureName Constant

Pascal

```
const ProcedureName = 'ProcedureName';
```

C++

```
ProcedureName = 'ProcedureName';
```

Description

This is ProcedureName, a member of class TDBXProcedureParametersColumns.

56.1.27.2.13 TDBXProcedureParametersColumns.Scale Constant

Pascal

```
const Scale = 'Scale';
```

C++

```
Scale = 'Scale';
```

Description

This is Scale, a member of class TDBXProcedureParametersColumns.

56.1.27.2.14 TDBXProcedureParametersColumns.SchemaName Constant

Pascal

```
const SchemaName = 'SchemaName';
```

C++

```
SchemaName = 'SchemaName';
```

Description

This is SchemaName, a member of class TDBXProcedureParametersColumns.

56.1.27.2.15 TDBXProcedureParametersColumns.TypeName Constant

Pascal

```
const TypeName = 'TypeName';
```

C++

```
TypeName = 'TypeName';
```

Description

This is TypeName, a member of class TDBXProcedureParametersColumns.

56.1.28 TDBXProcedureParametersIndex Class

Accesses columns by ordinal in the **ProcedureParameters** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXProcedureParametersIndex

Pascal

```
TDBXProcedureParametersIndex = class;
```

C++

```
class TDBXProcedureParametersIndex;
```

File

DBXMetaDataNames

Description

TDBXProcedureParametersIndex is used to access columns by ordinal in the **ProcedureParameters** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetProcedureParameters when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Ordinal (see page 10801) Name	Data Type of Column Data	Description
CatalogName (see page 10799)	String	Catalog name.
SchemaName (see page 10802)	String	Schema name.
ProcedureName (see page 10802)	String	Procedure name.
ParameterName (see page 10802)	String	Parameter name.
ParameterMode (see page 10801)	String	Parameter mode: IN, OUT, INOUT, RESULT.
TypeName (see page 10803)	String	Database specific data type name.
Precision (see page 10802)	Int32	Precision (see page 10802) of this type. For string data type, this is maximum number of characters. For decimal types, this is maximum number of digits, including digits from the scale.
Scale (see page 10802)	Int32	Scale (see page 10802) of a decimal type.
Ordinal (see page 10801)	Int32	Position in rows of table, starting with 1.
IsNullable (see page 10800)	Boolean	Can accept NULL values.
DbxDataType (see page 10800)	Int32	dbExpress data type.
IsFixedLength (see page 10800)	Boolean	Holds fixed length data.
IsUnicode (see page 10800)	Boolean	Can hold Unicode characters.
IsLong (see page 10800)	Boolean	Is BLOB data type.
IsUnsigned (see page 10801)	Boolean	Is unsigned numeric type.

See Also

TDBXMetaDataCommands (see page 10525)

TDBXProcedureParametersColumns (see page 10792)

56.1.28.1 TDBXProcedureParametersIndex Members

The following tables list the members exposed by TDBXProcedureParametersIndex.

Public Constants

	Name	Description
	CatalogName (see page 10799)	This is CatalogName, a member of class TDBXProcedureParametersIndex.
	DbxDataType (see page 10800)	This is DbxDataType, a member of class TDBXProcedureParametersIndex.

	IsFixedLength (see page 10800)	This is IsFixedLength, a member of class TDBXProcedureParametersIndex.
	IsLong (see page 10800)	This is IsLong, a member of class TDBXProcedureParametersIndex.
	IsNullable (see page 10800)	This is IsNullable, a member of class TDBXProcedureParametersIndex.
	IsUnicode (see page 10800)	This is IsUnicode, a member of class TDBXProcedureParametersIndex.
	IsUnsigned (see page 10801)	This is IsUnsigned, a member of class TDBXProcedureParametersIndex.
	Last (see page 10801)	This is Last, a member of class TDBXProcedureParametersIndex.
	Ordinal (see page 10801)	This is Ordinal, a member of class TDBXProcedureParametersIndex.
	ParameterMode (see page 10801)	This is ParameterMode, a member of class TDBXProcedureParametersIndex.
	ParameterName (see page 10802)	This is ParameterName, a member of class TDBXProcedureParametersIndex.
	Precision (see page 10802)	This is Precision, a member of class TDBXProcedureParametersIndex.
	ProcedureName (see page 10802)	This is ProcedureName, a member of class TDBXProcedureParametersIndex.
	Scale (see page 10802)	This is Scale, a member of class TDBXProcedureParametersIndex.
	SchemaName (see page 10802)	This is SchemaName, a member of class TDBXProcedureParametersIndex.
	TypeName (see page 10803)	This is TypeName, a member of class TDBXProcedureParametersIndex.

56.1.28.2 TDBXProcedureParametersIndex Constants

The constants of the TDBXProcedureParametersIndex class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10799)	This is CatalogName, a member of class TDBXProcedureParametersIndex.
	DbxDataType (see page 10800)	This is DbxDataType, a member of class TDBXProcedureParametersIndex.
	IsFixedLength (see page 10800)	This is IsFixedLength, a member of class TDBXProcedureParametersIndex.
	IsLong (see page 10800)	This is IsLong, a member of class TDBXProcedureParametersIndex.
	IsNullable (see page 10800)	This is IsNullable, a member of class TDBXProcedureParametersIndex.
	IsUnicode (see page 10800)	This is IsUnicode, a member of class TDBXProcedureParametersIndex.
	IsUnsigned (see page 10801)	This is IsUnsigned, a member of class TDBXProcedureParametersIndex.
	Last (see page 10801)	This is Last, a member of class TDBXProcedureParametersIndex.
	Ordinal (see page 10801)	This is Ordinal, a member of class TDBXProcedureParametersIndex.
	ParameterMode (see page 10801)	This is ParameterMode, a member of class TDBXProcedureParametersIndex.
	ParameterName (see page 10802)	This is ParameterName, a member of class TDBXProcedureParametersIndex.
	Precision (see page 10802)	This is Precision, a member of class TDBXProcedureParametersIndex.
	ProcedureName (see page 10802)	This is ProcedureName, a member of class TDBXProcedureParametersIndex.
	Scale (see page 10802)	This is Scale, a member of class TDBXProcedureParametersIndex.
	SchemaName (see page 10802)	This is SchemaName, a member of class TDBXProcedureParametersIndex.
	TypeName (see page 10803)	This is TypeName, a member of class TDBXProcedureParametersIndex.

56.1.28.2.1 TDBXProcedureParametersIndex.CatalogName Constant

Pascal

```
const CatalogName = 0;
```

C++

```
CatalogName = 0;
```

Description

This is CatalogName, a member of class TDBXProcedureParametersIndex.

56.1.28.2.2 TDBXProcedureParametersIndex.DbxDataType Constant

Pascal

```
const DbxDataType = 10;
```

C++

```
DbxDataType = 10;
```

Description

This is DbxDataType, a member of class TDBXProcedureParametersIndex.

56.1.28.2.3 TDBXProcedureParametersIndex.IsFixedLength Constant

Pascal

```
const IsFixedLength = 11;
```

C++

```
IsFixedLength = 11;
```

Description

This is IsFixedLength, a member of class TDBXProcedureParametersIndex.

56.1.28.2.4 TDBXProcedureParametersIndex.IsLong Constant

Pascal

```
const IsLong = 13;
```

C++

```
IsLong = 13;
```

Description

This is IsLong, a member of class TDBXProcedureParametersIndex.

56.1.28.2.5 TDBXProcedureParametersIndex.IsNullable Constant

Pascal

```
const IsNullable = 9;
```

C++

```
IsNullable = 9;
```

Description

This is IsNullable, a member of class TDBXProcedureParametersIndex.

56.1.28.2.6 TDBXProcedureParametersIndex.IsUnicode Constant

Pascal

```
const IsUnicode = 12;
```

C++

```
IsUnicode = 12;
```

Description

This is IsUnicode, a member of class TDBXProcedureParametersIndex.

56.1.28.2.7 TDBXProcedureParametersIndex.IsUnsigned Constant

Pascal

```
const IsUnsigned = 14;
```

C++

```
IsUnsigned = 14;
```

Description

This is IsUnsigned, a member of class TDBXProcedureParametersIndex.

56.1.28.2.8 TDBXProcedureParametersIndex.Last Constant

Pascal

```
const Last = 14;
```

C++

```
Last = 14;
```

Description

This is Last, a member of class TDBXProcedureParametersIndex.

56.1.28.2.9 TDBXProcedureParametersIndex.Ordinal Constant

Pascal

```
const Ordinal = 8;
```

C++

```
Ordinal = 8;
```

Description

This is Ordinal, a member of class TDBXProcedureParametersIndex.

56.1.28.2.10 TDBXProcedureParametersIndex.ParameterMode Constant

Pascal

```
const ParameterMode = 4;
```

C++

```
ParameterMode = 4;
```

Description

This is ParameterMode, a member of class TDBXProcedureParametersIndex.

56.1.28.2.11 TDBXProcedureParametersIndex.ParameterName Constant

Pascal

```
const ParameterName = 3;
```

C++

```
ParameterName = 3;
```

Description

This is ParameterName, a member of class TDBXProcedureParametersIndex.

56.1.28.2.12 TDBXProcedureParametersIndex.Precision Constant

Pascal

```
const Precision = 6;
```

C++

```
Precision = 6;
```

Description

This is Precision, a member of class TDBXProcedureParametersIndex.

56.1.28.2.13 TDBXProcedureParametersIndex.ProcedureName Constant

Pascal

```
const ProcedureName = 2;
```

C++

```
ProcedureName = 2;
```

Description

This is ProcedureName, a member of class TDBXProcedureParametersIndex.

56.1.28.2.14 TDBXProcedureParametersIndex.Scale Constant

Pascal

```
const Scale = 7;
```

C++

```
Scale = 7;
```

Description

This is Scale, a member of class TDBXProcedureParametersIndex.

56.1.28.2.15 TDBXProcedureParametersIndex.SchemaName Constant

Pascal

```
const SchemaName = 1;
```

C++

```
SchemaName = 1;
```

Description

This is SchemaName, a member of class TDBXProcedureParametersIndex.

56.1.28.2.16 TDBXProcedureParametersIndex.TypeName Constant

Pascal

```
const TypeName = 5;
```

C++

```
TypeName = 5;
```

Description

This is TypeName, a member of class TDBXProcedureParametersIndex.

56.1.29 TDBXProceduresColumns Class

Accesses columns by name in the **Procedures** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXProceduresColumns

Pascal

```
TDBXProceduresColumns = class;
```

C++

```
class TDBXProceduresColumns;
```

File

DBXMetaDataNames

Description

TDBXProceduresColumns is used to access columns by name in the **Procedures** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetProcedures when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Name	Data Type of Column Data	Description
CatalogName (see page 10804)	String	Catalog name.
SchemaName (see page 10805)	String	Schema name.
ProcedureName (see page 10804)	String	Procedure name.
ProcedureType (see page 10804)	String	Procedure type: FUNCTION or PROCEDURE.

See Also

TDBXMetaDataCommands (see page 10525)

TDBXProceduresIndex (see page 10805)

56.1.29.1 TDBXProceduresColumns Members

The following tables list the members exposed by TDBXProceduresColumns.

Public Constants

	Name	Description
	CatalogName (see page 10804)	This is CatalogName, a member of class TDBXProceduresColumns.
	ProcedureName (see page 10804)	This is ProcedureName, a member of class TDBXProceduresColumns.
	ProcedureType (see page 10804)	This is ProcedureType, a member of class TDBXProceduresColumns.
	SchemaName (see page 10805)	This is SchemaName, a member of class TDBXProceduresColumns.

56.1.29.2 TDBXProceduresColumns Constants

The constants of the TDBXProceduresColumns class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10804)	This is CatalogName, a member of class TDBXProceduresColumns.
	ProcedureName (see page 10804)	This is ProcedureName, a member of class TDBXProceduresColumns.
	ProcedureType (see page 10804)	This is ProcedureType, a member of class TDBXProceduresColumns.
	SchemaName (see page 10805)	This is SchemaName, a member of class TDBXProceduresColumns.

56.1.29.2.1 TDBXProceduresColumns.CatalogName Constant

Pascal

```
const CatalogName = 'CatalogName';
```

C++

```
CatalogName = 'CatalogName';
```

Description

This is CatalogName, a member of class TDBXProceduresColumns.

56.1.29.2.2 TDBXProceduresColumns.ProcedureName Constant

Pascal

```
const ProcedureName = 'ProcedureName';
```

C++

```
ProcedureName = 'ProcedureName';
```

Description

This is ProcedureName, a member of class TDBXProceduresColumns.

56.1.29.2.3 TDBXProceduresColumns.ProcedureType Constant

Pascal

```
const ProcedureType = 'ProcedureType';
```

C++

```
ProcedureType = 'ProcedureType';
```

Description

This is ProcedureType, a member of class TDBXProceduresColumns.

56.1.29.2.4 TDBXProceduresColumns.SchemaName Constant

Pascal

```
const SchemaName = 'SchemaName';
```

C++

```
SchemaName = 'SchemaName';
```

Description

This is SchemaName, a member of class TDBXProceduresColumns.

56.1.30 TDBXProceduresIndex Class

Accesses columns by ordinal in the **Procedures** metadata.

Class Hierarchy

Pascal

```
TDBXProceduresIndex = class;
```

C++

```
class TDBXProceduresIndex;
```

File

DBXMetaDataNames

Description

TDBXProceduresIndex is used to access columns by ordinal in the **Procedures** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetProcedures when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Ordinal Name	Data Type of Column Data	Description
CatalogName (see page 10806)	String	Catalog name.
SchemaName (see page 10807)	String	Schema name.
ProcedureName (see page 10807)	String	Procedure name.
ProcedureType (see page 10807)	String	Procedure type: FUNCTION or PROCEDURE.

See Also

TDBXMetaDataCommands (see page 10525)

TDBXProceduresColumns (see page 10803)

56.1.30.1 TDBXProceduresIndex Members

The following tables list the members exposed by TDBXProceduresIndex.

Public Constants

	Name	Description
	CatalogName (see page 10806)	This is CatalogName, a member of class TDBXProceduresIndex.
	Last (see page 10806)	This is Last, a member of class TDBXProceduresIndex.
	ProcedureName (see page 10807)	This is ProcedureName, a member of class TDBXProceduresIndex.
	ProcedureType (see page 10807)	This is ProcedureType, a member of class TDBXProceduresIndex.
	SchemaName (see page 10807)	This is SchemaName, a member of class TDBXProceduresIndex.

56.1.30.2 TDBXProceduresIndex Constants

The constants of the TDBXProceduresIndex class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10806)	This is CatalogName, a member of class TDBXProceduresIndex.
	Last (see page 10806)	This is Last, a member of class TDBXProceduresIndex.
	ProcedureName (see page 10807)	This is ProcedureName, a member of class TDBXProceduresIndex.
	ProcedureType (see page 10807)	This is ProcedureType, a member of class TDBXProceduresIndex.
	SchemaName (see page 10807)	This is SchemaName, a member of class TDBXProceduresIndex.

56.1.30.2.1 TDBXProceduresIndex.CatalogName Constant

Pascal

```
const CatalogName = 0;
```

C++

```
CatalogName = 0;
```

Description

This is CatalogName, a member of class TDBXProceduresIndex.

56.1.30.2.2 TDBXProceduresIndex.Last Constant

Pascal

```
const Last = 3;
```

C++

```
Last = 3;
```

Description

This is Last, a member of class TDBXProceduresIndex.

56.1.30.2.3 TDBXProceduresIndex.ProcedureName Constant

Pascal

```
const ProcedureName = 2;
```

C++

```
ProcedureName = 2;
```

Description

This is ProcedureName, a member of class TDBXProceduresIndex.

56.1.30.2.4 TDBXProceduresIndex.ProcedureType Constant

Pascal

```
const ProcedureType = 3;
```

C++

```
ProcedureType = 3;
```

Description

This is ProcedureType, a member of class TDBXProceduresIndex.

56.1.30.2.5 TDBXProceduresIndex.SchemaName Constant

Pascal

```
const SchemaName = 1;
```

C++

```
SchemaName = 1;
```

Description

This is SchemaName, a member of class TDBXProceduresIndex.

56.1.31 TDBXProcedureSourcesColumns Class

Accesses columns by name in the **ProcedureSources** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXProcedureSourcesColumns

Pascal

```
TDBXProcedureSourcesColumns = class;
```

C++

```
class TDBXProcedureSourcesColumns;
```

File

DBXMetaDataNames

Description

TDBXProcedureSourcesColumns is used to access columns by name in the **ProcedureSources** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetProcedureSources when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Name	Data Type of Column Data	Description
CatalogName (see page 10809)	String	Catalog name.
SchemaName (see page 10810)	String	Schema name.
ProcedureName (see page 10809)	String	Procedure name.
ProcedureType (see page 10809)	String	Procedure type: FUNCTION or PROCEDURE.
Definition (see page 10809)	String	Source of SQL procedure.
ExternalDefinition (see page 10809)	String	Reference to procedure external to database.

See Also

TDBXMetaDataCommandsEx (see page 10531)

TDBXProcedureSourcesIndex (see page 10810)

56.1.31.1 TDBXProcedureSourcesColumns Members

The following tables list the members exposed by TDBXProcedureSourcesColumns.

Public Constants

	Name	Description
	CatalogName (see page 10809)	This is CatalogName, a member of class TDBXProcedureSourcesColumns.
	Definition (see page 10809)	This is Definition, a member of class TDBXProcedureSourcesColumns.
	ExternalDefinition (see page 10809)	This is ExternalDefinition, a member of class TDBXProcedureSourcesColumns.
	ProcedureName (see page 10809)	This is ProcedureName, a member of class TDBXProcedureSourcesColumns.
	ProcedureType (see page 10809)	This is ProcedureType, a member of class TDBXProcedureSourcesColumns.
	SchemaName (see page 10810)	This is SchemaName, a member of class TDBXProcedureSourcesColumns.

56.1.31.2 TDBXProcedureSourcesColumns Constants

The constants of the TDBXProcedureSourcesColumns class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10809)	This is CatalogName, a member of class TDBXProcedureSourcesColumns.
	Definition (see page 10809)	This is Definition, a member of class TDBXProcedureSourcesColumns.
	ExternalDefinition (see page 10809)	This is ExternalDefinition, a member of class TDBXProcedureSourcesColumns.
	ProcedureName (see page 10809)	This is ProcedureName, a member of class TDBXProcedureSourcesColumns.
	ProcedureType (see page 10809)	This is ProcedureType, a member of class TDBXProcedureSourcesColumns.

	SchemaName (see page 10810)	This is SchemaName, a member of class TDBXProcedureSourcesColumns.

56.1.31.2.1 TDBXProcedureSourcesColumns.CatalogName Constant

Pascal

```
const CatalogName = 'CatalogName';
```

C++

```
CatalogName = 'CatalogName';
```

Description

This is CatalogName, a member of class TDBXProcedureSourcesColumns.

56.1.31.2.2 TDBXProcedureSourcesColumns.Definition Constant

Pascal

```
const Definition = 'Definition';
```

C++

```
Definition = 'Definition';
```

Description

This is Definition, a member of class TDBXProcedureSourcesColumns.

56.1.31.2.3 TDBXProcedureSourcesColumns.ExternalDefinition Constant

Pascal

```
const ExternalDefinition = 'ExternalDefinition';
```

C++

```
ExternalDefinition = 'ExternalDefinition';
```

Description

This is ExternalDefinition, a member of class TDBXProcedureSourcesColumns.

56.1.31.2.4 TDBXProcedureSourcesColumns.ProcedureName Constant

Pascal

```
const ProcedureName = 'ProcedureName';
```

C++

```
ProcedureName = 'ProcedureName';
```

Description

This is ProcedureName, a member of class TDBXProcedureSourcesColumns.

56.1.31.2.5 TDBXProcedureSourcesColumns.ProcedureType Constant

Pascal

```
const ProcedureType = 'ProcedureType';
```

C++

```
ProcedureType = 'ProcedureType';
```

Description

This is ProcedureType, a member of class TDBXProcedureSourcesColumns.

56.1.31.2.6 TDBXProcedureSourcesColumns.SchemaName Constant

Pascal

```
const SchemaName = 'SchemaName';
```

C++

```
SchemaName = 'SchemaName';
```

Description

This is SchemaName, a member of class TDBXProcedureSourcesColumns.

56.1.32 TDBXProcedureSourcesIndex Class

Accesses columns by ordinal in the **ProcedureSources** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXProcedureSourcesIndex

Pascal

```
TDBXProcedureSourcesIndex = class;
```

C++

```
class TDBXProcedureSourcesIndex;
```

File

DBXMetaDataNames

Description

TDBXProcedureSourcesIndex is used to access columns by ordinal in the **ProcedureSources** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetProcedureSources when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Ordinal Name	Data Type of Column Data	Description
CatalogName (see page 10811)	String	Catalog name.
SchemaName (see page 10813)	String	Schema name.
ProcedureName (see page 10812)	String	Procedure name.
ProcedureType (see page 10812)	String	Procedure type: FUNCTION or PROCEDURE.
Definition (see page 10812)	String	Source of SQL procedure.

ExternalDefinition (see page 10812)	String	Reference to procedure external to database.

See Also

TDBXMetaDataCommandsEx (see page 10531)

TDBXProcedureSourcesColumns (see page 10807)

56.1.32.1 TDBXProcedureSourcesIndex Members

The following tables list the members exposed by TDBXProcedureSourcesIndex.

Public Constants

	Name	Description
	CatalogName (see page 10811)	This is CatalogName, a member of class TDBXProcedureSourcesIndex.
	Definition (see page 10812)	This is Definition, a member of class TDBXProcedureSourcesIndex.
	ExternalDefinition (see page 10812)	This is ExternalDefinition, a member of class TDBXProcedureSourcesIndex.
	Last (see page 10812)	This is Last, a member of class TDBXProcedureSourcesIndex.
	ProcedureName (see page 10812)	This is ProcedureName, a member of class TDBXProcedureSourcesIndex.
	ProcedureType (see page 10812)	This is ProcedureType, a member of class TDBXProcedureSourcesIndex.
	SchemaName (see page 10813)	This is SchemaName, a member of class TDBXProcedureSourcesIndex.

56.1.32.2 TDBXProcedureSourcesIndex Constants

The constants of the TDBXProcedureSourcesIndex class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10811)	This is CatalogName, a member of class TDBXProcedureSourcesIndex.
	Definition (see page 10812)	This is Definition, a member of class TDBXProcedureSourcesIndex.
	ExternalDefinition (see page 10812)	This is ExternalDefinition, a member of class TDBXProcedureSourcesIndex.
	Last (see page 10812)	This is Last, a member of class TDBXProcedureSourcesIndex.
	ProcedureName (see page 10812)	This is ProcedureName, a member of class TDBXProcedureSourcesIndex.
	ProcedureType (see page 10812)	This is ProcedureType, a member of class TDBXProcedureSourcesIndex.
	SchemaName (see page 10813)	This is SchemaName, a member of class TDBXProcedureSourcesIndex.

56.1.32.2.1 TDBXProcedureSourcesIndex.CatalogName Constant

Pascal

```
const CatalogName = 0;
```

C++

```
CatalogName = 0;
```

Description

This is CatalogName, a member of class TDBXProcedureSourcesIndex.

56.1.32.2.2 TDBXProcedureSourcesIndex.Definition Constant

Pascal

```
const Definition = 4;
```

C++

```
Definition = 4;
```

Description

This is Definition, a member of class TDBXProcedureSourcesIndex.

56.1.32.2.3 TDBXProcedureSourcesIndex.ExternalDefinition Constant

Pascal

```
const ExternalDefinition = 5;
```

C++

```
ExternalDefinition = 5;
```

Description

This is ExternalDefinition, a member of class TDBXProcedureSourcesIndex.

56.1.32.2.4 TDBXProcedureSourcesIndex.Last Constant

Pascal

```
const Last = 5;
```

C++

```
Last = 5;
```

Description

This is Last, a member of class TDBXProcedureSourcesIndex.

56.1.32.2.5 TDBXProcedureSourcesIndex.ProcedureName Constant

Pascal

```
const ProcedureName = 2;
```

C++

```
ProcedureName = 2;
```

Description

This is ProcedureName, a member of class TDBXProcedureSourcesIndex.

56.1.32.2.6 TDBXProcedureSourcesIndex.ProcedureType Constant

Pascal

```
const ProcedureType = 3;
```

C++

```
ProcedureType = 3;
```

Description

This is ProcedureType, a member of class TDBXProcedureSourcesIndex.

56.1.32.2.7 TDBXProcedureSourcesIndex.SchemaName Constant

Pascal

```
const SchemaName = 1;
```

C++

```
SchemaName = 1;
```

Description

This is SchemaName, a member of class TDBXProcedureSourcesIndex.

56.1.33 TDBXReservedWordsColumns Class

Accesses columns by name in the **ReservedWords** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXReservedWordsColumns

Pascal

```
TDBXReservedWordsColumns = class;
```

C++

```
class TDBXReservedWordsColumns;
```

File

DBXMetaDataNames

Description

TDBXReservedWordsColumns is used to access columns by name in the **ReservedWords** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetReservedWords when reading metadata. To retrieve a metadata column, use this constant to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the column.

Column Name	Data Type of Column Data	Description
ReservedWord (see page 10814)	String	An identifier reserved for SQL syntax. These identifiers should not be used as object names, such as tables and columns. Some databases allows their use if they are quoted.

See Also

TDBXMetaDataCommandsEx (see page 10531)

TDBXReservedWordsIndex (see page 10814)

56.1.33.1 TDBXReservedWordsColumns Members

The following tables list the members exposed by TDBXReservedWordsColumns.

Public Constants

	Name	Description
	ReservedWord (see page 10814)	This is ReservedWord, a member of class TDBXReservedWordsColumns.

56.1.33.2 TDBXReservedWordsColumns Constants

The constants of the TDBXReservedWordsColumns class are listed here.

Public Constants

	Name	Description
	ReservedWord (see page 10814)	This is ReservedWord, a member of class TDBXReservedWordsColumns.

56.1.33.2.1 TDBXReservedWordsColumns.ReservedWord Constant

Pascal

```
const ReservedWord = 'ReservedWord';
```

C++

```
ReservedWord = 'ReservedWord';
```

Description

This is ReservedWord, a member of class TDBXReservedWordsColumns.

56.1.34 TDBXReservedWordsIndex Class

Accesses columns by ordinal in the **ReservedWords** metadata.

Class Hierarchy

Pascal

```
TDBXReservedWordsIndex = class;
```

C++

```
class TDBXReservedWordsIndex;
```

File

DBXMetaDataNames

Description

TDBXReservedWordsIndex is used to access columns by ordinal in the **ReservedWords** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetReservedWords when reading metadata. To retrieve a metadata column, use this constant to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the column.

Column Ordinal Name	Data Type of Column Data	Description
ReservedWord (see page 10815)	String	An identifier reserved for SQL syntax. These identifiers should not be used as object names, such as tables and columns. Some databases allows their use if they are quoted.

See Also

TDBXMetaDataCommandsEx (see page 10531)

TDBXReservedWordsColumns (see page 10813)

56.1.34.1 TDBXReservedWordsIndex Members

The following tables list the members exposed by TDBXReservedWordsIndex.

Public Constants

	Name	Description
	Last (see page 10815)	This is Last, a member of class TDBXReservedWordsIndex.
	ReservedWord (see page 10815)	This is ReservedWord, a member of class TDBXReservedWordsIndex.

56.1.34.2 TDBXReservedWordsIndex Constants

The constants of the TDBXReservedWordsIndex class are listed here.

Public Constants

	Name	Description
	Last (see page 10815)	This is Last, a member of class TDBXReservedWordsIndex.
	ReservedWord (see page 10815)	This is ReservedWord, a member of class TDBXReservedWordsIndex.

56.1.34.2.1 TDBXReservedWordsIndex.Last Constant

Pascal

```
const Last = 0;
```

C++

```
Last = 0;
```

Description

This is Last, a member of class TDBXReservedWordsIndex.

56.1.34.2.2 TDBXReservedWordsIndex.ReservedWord Constant

Pascal

```
const ReservedWord = 0;
```

C++

```
ReservedWord = 0;
```

Description

This is ReservedWord, a member of class TDBXReservedWordsIndex.

56.1.35 TDBXRolesColumns Class

Accesses columns by name in the **Roles** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXRolesColumns

Pascal

```
TDBXRolesColumns = class;
```

C++

```
class TDBXRolesColumns;
```

File

DBXMetaDataNames

Description

TDBXReservedWordsColumns (see page 10813) is used to access columns by name in the **Roles** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetRoles when reading metadata. To retrieve a metadata column, use this constant to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the column.

Column Name	Data Type of Column Data	Description
RoleName (see page 10817)	String	Role name.

See Also

TDBXMetaDataCommandsEx (see page 10531)

TDBXRolesIndex (see page 10817)

56.1.35.1 TDBXRolesColumns Members

The following tables list the members exposed by TDBXRolesColumns.

Public Constants

	Name	Description
	RoleName (see page 10817)	This is RoleName, a member of class TDBXRolesColumns.

56.1.35.2 TDBXRolesColumns Constants

The constants of the TDBXRolesColumns class are listed here.

Public Constants

	Name	Description
	RoleName (see page 10817)	This is RoleName, a member of class TDBXRolesColumns.

56.1.35.2.1 TDBXRolesColumns.RoleName Constant

Pascal

```
const RoleName = 'RoleName';
```

C++

```
RoleName = 'RoleName';
```

Description

This is RoleName, a member of class TDBXRolesColumns.

56.1.36 TDBXRolesIndex Class

Accesses columns by ordinal in the **Roles** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXRolesIndex

Pascal

```
TDBXRolesIndex = class;
```

C++

```
class TDBXRolesIndex;
```

File

DBXMetaDataNames

Description

TDBXRolesIndex is used to access columns by ordinal in the **Roles** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetRoles when reading metadata. To retrieve a metadata column, use this constant to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the column.

Column Ordinal Name	Data Type of Column Data	Description
RoleName (see page 10818)	String	Role name.

See Also

TDBXMetaDataCommandsEx (see page 10531)

TDBXRolesColumns (see page 10816)

56.1.36.1 TDBXRolesIndex Members

The following tables list the members exposed by TDBXRolesIndex.

Public Constants

	Name	Description
	Last (see page 10818)	This is Last, a member of class TDBXRolesIndex.
	RoleName (see page 10818)	This is RoleName, a member of class TDBXRolesIndex.

56.1.36.2 TDBXRolesIndex Constants

The constants of the TDBXRolesIndex class are listed here.

Public Constants

	Name	Description
	Last (see page 10818)	This is Last, a member of class TDBXRolesIndex.
	RoleName (see page 10818)	This is RoleName, a member of class TDBXRolesIndex.

56.1.36.2.1 TDBXRolesIndex.Last Constant

Pascal

```
const Last = 0;
```

C++

```
Last = 0;
```

Description

This is Last, a member of class TDBXRolesIndex.

56.1.36.2.2 TDBXRolesIndex.RoleName Constant

Pascal

```
const RoleName = 0;
```

C++

```
RoleName = 0;
```

Description

This is RoleName, a member of class TDBXRolesIndex.

56.1.37 TDBXSchemasColumns Class

Accesses columns by name in the **Schemas** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXSchemasColumns

Pascal

```
TDBXSchemasColumns = class;
```

C++

```
class TDBXSchemasColumns;
```

File

DBXMetaDataNames

Description

TDBXSchemasColumns is used to access columns by name in the **Schemas** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetSchemas when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Name	Data Type of Column Data	Description
CatalogName (see page 10819)	String	Catalog name.
SchemaName (see page 10820)	String	Schema name.

See Also

TDBXMetaDataCommandsEx (see page 10531)

TDBXSchemasIndex (see page 10820)

56.1.37.1 TDBXSchemasColumns Members

The following tables list the members exposed by TDBXSchemasColumns.

Public Constants

	Name	Description
	CatalogName (see page 10819)	This is CatalogName, a member of class TDBXSchemasColumns.
	SchemaName (see page 10820)	This is SchemaName, a member of class TDBXSchemasColumns.

56.1.37.2 TDBXSchemasColumns Constants

The constants of the TDBXSchemasColumns class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10819)	This is CatalogName, a member of class TDBXSchemasColumns.
	SchemaName (see page 10820)	This is SchemaName, a member of class TDBXSchemasColumns.

56.1.37.2.1 TDBXSchemasColumns.CatalogName Constant

Pascal

```
const CatalogName = 'CatalogName';
```

C++

```
CatalogName = 'CatalogName';
```

Description

This is CatalogName, a member of class TDBXSchemasColumns.

56.1.37.2.2 TDBXSchemasColumns.SchemaName Constant

Pascal

```
const SchemaName = 'SchemaName';
```

C++

```
SchemaName = 'SchemaName';
```

Description

This is SchemaName, a member of class TDBXSchemasColumns.

56.1.38 TDBXSchemasIndex Class

Accesses columns by ordinal in the **Schemas** metadata.

Class Hierarchy

Pascal

```
TDBXSchemasIndex = class;
```

C++

```
class TDBXSchemasIndex;
```

File

DBXMetaDataNames

Description

TDBXSchemasIndex is used to access columns by ordinal in the **Schemas** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetSchemas when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Ordinal Name	Data Type of Column Data	Description
CatalogName (see page 10821)	String	Catalog name.
SchemaName (see page 10821)	String	Schema name.

See Also

TDBXMetaDataCommandsEx (see page 10531)

TDBXSchemasColumns (see page 10818)

56.1.38.1 TDBXSchemasIndex Members

The following tables list the members exposed by TDBXSchemasIndex.

Public Constants

	Name	Description
	CatalogName (see page 10821)	This is CatalogName, a member of class TDBXSchemasIndex.
	Last (see page 10821)	This is Last, a member of class TDBXSchemasIndex.
	SchemaName (see page 10821)	This is SchemaName, a member of class TDBXSchemasIndex.

56.1.38.2 TDBXSchemasIndex Constants

The constants of the TDBXSchemasIndex class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10821)	This is CatalogName, a member of class TDBXSchemasIndex.
	Last (see page 10821)	This is Last, a member of class TDBXSchemasIndex.
	SchemaName (see page 10821)	This is SchemaName, a member of class TDBXSchemasIndex.

56.1.38.2.1 TDBXSchemasIndex.CatalogName Constant

Pascal

```
const CatalogName = 0;
```

C++

```
CatalogName = 0;
```

Description

This is CatalogName, a member of class TDBXSchemasIndex.

56.1.38.2.2 TDBXSchemasIndex.Last Constant

Pascal

```
const Last = 1;
```

C++

```
Last = 1;
```

Description

This is Last, a member of class TDBXSchemasIndex.

56.1.38.2.3 TDBXSchemasIndex.SchemaName Constant

Pascal

```
const SchemaName = 1;
```

C++

```
SchemaName = 1;
```

Description

This is SchemaName, a member of class TDBXSchemasIndex.

56.1.39 TDBXSynonymsColumns Class

Accesses columns by name in the **Synonyms** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXSynonymsColumns

Pascal

```
TDBXSynonymsColumns = class;
```

C++

```
class TDBXSynonymsColumns;
```

File

DBXMetaDataNames

Description

TDBXSynonymsColumns is used to access columns by name in the **Synonyms** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetSynonyms when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Name	Data Type of Column Data	Description
CatalogName (see page 10823)	String	Catalog name.
SchemaName (see page 10823)	String	Schema name.
SynonymName (see page 10823)	String	Synonym name.
TableCatalogName (see page 10824)	String	Catalog name of table this synonym refers to.
TableSchemaName (see page 10824)	String	Schema name of table this synonym refers to.
TableName (see page 10824)	String	Name of table this synonym refers to.

See Also

TDBXMetaDataCommandsEx (see page 10531)

TDBXSynonymsIndex (see page 10824)

56.1.39.1 TDBXSynonymsColumns Members

The following tables list the members exposed by TDBXSynonymsColumns.

Public Constants

	Name	Description
	CatalogName (see page 10823)	This is CatalogName, a member of class TDBXSynonymsColumns.
	SchemaName (see page 10823)	This is SchemaName, a member of class TDBXSynonymsColumns.
	SynonymName (see page 10823)	This is SynonymName, a member of class TDBXSynonymsColumns.
	TableCatalogName (see page 10824)	This is TableCatalogName, a member of class TDBXSynonymsColumns.
	TableName (see page 10824)	This is TableName, a member of class TDBXSynonymsColumns.
	TableSchemaName (see page 10824)	This is TableSchemaName, a member of class TDBXSynonymsColumns.

56.1.39.2 TDBXSynonymsColumns Constants

The constants of the TDBXSynonymsColumns class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10823)	This is CatalogName, a member of class TDBXSynonymsColumns.
	SchemaName (see page 10823)	This is SchemaName, a member of class TDBXSynonymsColumns.
	SynonymName (see page 10823)	This is SynonymName, a member of class TDBXSynonymsColumns.
	TableCatalogName (see page 10824)	This is TableCatalogName, a member of class TDBXSynonymsColumns.
	TableName (see page 10824)	This is TableName, a member of class TDBXSynonymsColumns.
	TableSchemaName (see page 10824)	This is TableSchemaName, a member of class TDBXSynonymsColumns.

56.1.39.2.1 TDBXSynonymsColumns.CatalogName Constant

Pascal

```
const CatalogName = 'CatalogName';
```

C++

```
CatalogName = 'CatalogName';
```

Description

This is CatalogName, a member of class TDBXSynonymsColumns.

56.1.39.2.2 TDBXSynonymsColumns.SchemaName Constant

Pascal

```
const SchemaName = 'SchemaName';
```

C++

```
SchemaName = 'SchemaName';
```

Description

This is SchemaName, a member of class TDBXSynonymsColumns.

56.1.39.2.3 TDBXSynonymsColumns.SynonymName Constant

Pascal

```
const SynonymName = 'SynonymName';
```

C++

```
SynonymName = 'SynonymName';
```

Description

This is SynonymName, a member of class TDBXSynonymsColumns.

56.1.39.2.4 TDBXSynonymsColumns.TableCatalogName Constant

Pascal

```
const TableCatalogName = 'TableCatalogName';
```

C++

```
TableCatalogName = 'TableCatalogName';
```

Description

This is TableCatalogName, a member of class TDBXSynonymsColumns.

56.1.39.2.5 TDBXSynonymsColumns.TableName Constant

Pascal

```
const TableName = 'TableName';
```

C++

```
TableName = 'TableName';
```

Description

This is TableName, a member of class TDBXSynonymsColumns.

56.1.39.2.6 TDBXSynonymsColumns.TableSchemaName Constant

Pascal

```
const TableSchemaName = 'TableSchemaName';
```

C++

```
TableSchemaName = 'TableSchemaName';
```

Description

This is TableSchemaName, a member of class TDBXSynonymsColumns.

56.1.40 TDBXSynonymsIndex Class

Accesses columns by ordinal in the **Synonyms** metadata.

Class Hierarchy

Pascal

```
TDBXSynonymsIndex = class;
```

C++

```
class TDBXSynonymsIndex;
```

File

DBXMetaDataNames

Description

TDBXSynonymsIndex is used to access columns by ordinal in the **Synonyms** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetSynonyms when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Ordinal Name	Data Type of Column Data	Description
CatalogName (see page 10826)	String	Catalog name.
SchemaName (see page 10826)	String	Schema name.
SynonymName (see page 10826)	String	Synonym name.
TableCatalogName (see page 10827)	String	Catalog name of table this synonym refers to.
TableSchemaName (see page 10827)	String	Schema name of table this synonym refers to.
TableName (see page 10827)	String	Name of table this synonym refers to.

See Also

TDBXMetaDataCommandsEx (see page 10531)

TDBXSynonymsColumns (see page 10822)

56.1.40.1 TDBXSynonymsIndex Members

The following tables list the members exposed by TDBXSynonymsIndex.

Public Constants

	Name	Description
	CatalogName (see page 10826)	This is CatalogName, a member of class TDBXSynonymsIndex.
	Last (see page 10826)	This is Last, a member of class TDBXSynonymsIndex.
	SchemaName (see page 10826)	This is SchemaName, a member of class TDBXSynonymsIndex.
	SynonymName (see page 10826)	This is SynonymName, a member of class TDBXSynonymsIndex.
	TableCatalogName (see page 10827)	This is TableCatalogName, a member of class TDBXSynonymsIndex.
	TableName (see page 10827)	This is TableName, a member of class TDBXSynonymsIndex.
	TableSchemaName (see page 10827)	This is TableSchemaName, a member of class TDBXSynonymsIndex.

56.1.40.2 TDBXSynonymsIndex Constants

The constants of the TDBXSynonymsIndex class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10826)	This is CatalogName, a member of class TDBXSynonymsIndex.
	Last (see page 10826)	This is Last, a member of class TDBXSynonymsIndex.
	SchemaName (see page 10826)	This is SchemaName, a member of class TDBXSynonymsIndex.
	SynonymName (see page 10826)	This is SynonymName, a member of class TDBXSynonymsIndex.
	TableCatalogName (see page 10827)	This is TableCatalogName, a member of class TDBXSynonymsIndex.
	TableName (see page 10827)	This is TableName, a member of class TDBXSynonymsIndex.
	TableSchemaName (see page 10827)	This is TableSchemaName, a member of class TDBXSynonymsIndex.

56.1.40.2.1 TDBXSynonymsIndex.CatalogName Constant

Pascal

```
const CatalogName = 0;
```

C++

```
CatalogName = 0;
```

Description

This is CatalogName, a member of class TDBXSynonymsIndex.

56.1.40.2.2 TDBXSynonymsIndex.Last Constant

Pascal

```
const Last = 5;
```

C++

```
Last = 5;
```

Description

This is Last, a member of class TDBXSynonymsIndex.

56.1.40.2.3 TDBXSynonymsIndex.SchemaName Constant

Pascal

```
const SchemaName = 1;
```

C++

```
SchemaName = 1;
```

Description

This is SchemaName, a member of class TDBXSynonymsIndex.

56.1.40.2.4 TDBXSynonymsIndex.SynonymName Constant

Pascal

```
const SynonymName = 2;
```

C++

```
SynonymName = 2;
```

Description

This is SynonymName, a member of class TDBXSynonymsIndex.

56.1.40.2.5 TDBXSynonymsIndex.TableCatalogName Constant

Pascal

```
const TableCatalogName = 3;
```

C++

```
TableCatalogName = 3;
```

Description

This is TableCatalogName, a member of class TDBXSynonymsIndex.

56.1.40.2.6 TDBXSynonymsIndex.TableName Constant

Pascal

```
const TableName = 5;
```

C++

```
TableName = 5;
```

Description

This is TableName, a member of class TDBXSynonymsIndex.

56.1.40.2.7 TDBXSynonymsIndex.TableSchemaName Constant

Pascal

```
const TableSchemaName = 4;
```

C++

```
TableSchemaName = 4;
```

Description

This is TableSchemaName, a member of class TDBXSynonymsIndex.

56.1.41 TDBXTablesColumns Class

Accesses columns by name in the **Tables** metadata.

Class Hierarchy

Pascal

```
TDBXTablesColumns = class;
```

C++

```
class TDBXTablesColumns;
```

File

DBXMetaDataNames

Description

TDBXTablesColumns is used to access columns by name in the **Tables** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetTables when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Name	Data Type of Column Data	Description
CatalogName (see page 10828)	String	Catalog name.
SchemaName (see page 10829)	String	Schema name.
TableName (see page 10829)	String	Table name.
TableType (see page 10829)	String	Table type: TABLE, VIEW, SYNONYM, SYSTEM_TABLE.

See Also

TDBXMetaDataCommands (see page 10525)

TDBXTablesIndex (see page 10829)

56.1.41.1 TDBXTablesColumns Members

The following tables list the members exposed by TDBXTablesColumns.

Public Constants

	Name	Description
	CatalogName (see page 10828)	This is CatalogName, a member of class TDBXTablesColumns.
	SchemaName (see page 10829)	This is SchemaName, a member of class TDBXTablesColumns.
	TableName (see page 10829)	This is TableName, a member of class TDBXTablesColumns.
	TableType (see page 10829)	This is TableType, a member of class TDBXTablesColumns.

56.1.41.2 TDBXTablesColumns Constants

The constants of the TDBXTablesColumns class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10828)	This is CatalogName, a member of class TDBXTablesColumns.
	SchemaName (see page 10829)	This is SchemaName, a member of class TDBXTablesColumns.
	TableName (see page 10829)	This is TableName, a member of class TDBXTablesColumns.
	TableType (see page 10829)	This is TableType, a member of class TDBXTablesColumns.

56.1.41.2.1 TDBXTablesColumns.CatalogName Constant

Pascal

```
const CatalogName = 'CatalogName';
```

C++

```
CatalogName = 'CatalogName';
```

Description

This is CatalogName, a member of class TDBXTablesColumns.

56.1.41.2.2 TDBXTablesColumns.SchemaName Constant

Pascal

```
const SchemaName = 'SchemaName';
```

C++

```
SchemaName = 'SchemaName';
```

Description

This is SchemaName, a member of class TDBXTablesColumns.

56.1.41.2.3 TDBXTablesColumns.TableName Constant

Pascal

```
const TableName = 'TableName';
```

C++

```
TableName = 'TableName';
```

Description

This is TableName, a member of class TDBXTablesColumns.

56.1.41.2.4 TDBXTablesColumns.TableType Constant

Pascal

```
const TableType = 'TableType';
```

C++

```
TableType = 'TableType';
```

Description

This is TableType, a member of class TDBXTablesColumns.

56.1.42 TDBXTablesIndex Class

Accesses columns by ordinal in the **Tables** metadata.

Class Hierarchy

Pascal

```
TDBXTablesIndex = class;
```

C++

```
class TDBXTablesIndex;
```

File

DBXMetaDataNames

Description

TDBXTablesIndex is used to access columns by ordinal in the **Tables** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetTables when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Ordinal Name	Data Type of Column Data	Description
CatalogName (see page 10831)	String	Catalog name.
SchemaName (see page 10831)	String	Schema name.
TableName (see page 10831)	String	Table name.
TableType (see page 10831)	String	Table type: TABLE, VIEW, SYNONYM, SYSTEM_TABLE.

See Also

TDBXMetaDataCommands (see page 10525)

TDBXTablesColumns (see page 10827)

56.1.42.1 TDBXTablesIndex Members

The following tables list the members exposed by TDBXTablesIndex.

Public Constants

	Name	Description
	CatalogName (see page 10831)	This is CatalogName, a member of class TDBXTablesIndex.
	Last (see page 10831)	This is Last, a member of class TDBXTablesIndex.
	SchemaName (see page 10831)	This is SchemaName, a member of class TDBXTablesIndex.
	TableName (see page 10831)	This is TableName, a member of class TDBXTablesIndex.
	TableType (see page 10831)	This is TableType, a member of class TDBXTablesIndex.

56.1.42.2 TDBXTablesIndex Constants

The constants of the TDBXTablesIndex class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10831)	This is CatalogName, a member of class TDBXTablesIndex.
	Last (see page 10831)	This is Last, a member of class TDBXTablesIndex.
	SchemaName (see page 10831)	This is SchemaName, a member of class TDBXTablesIndex.
	TableName (see page 10831)	This is TableName, a member of class TDBXTablesIndex.
	TableType (see page 10831)	This is TableType, a member of class TDBXTablesIndex.

56.1.42.2.1 TDBXTablesIndex.CatalogName Constant

Pascal

```
const CatalogName = 0;
```

C++

```
CatalogName = 0;
```

Description

This is CatalogName, a member of class TDBXTablesIndex.

56.1.42.2.2 TDBXTablesIndex.Last Constant

Pascal

```
const Last = 3;
```

C++

```
Last = 3;
```

Description

This is Last, a member of class TDBXTablesIndex.

56.1.42.2.3 TDBXTablesIndex.SchemaName Constant

Pascal

```
const SchemaName = 1;
```

C++

```
SchemaName = 1;
```

Description

This is SchemaName, a member of class TDBXTablesIndex.

56.1.42.2.4 TDBXTablesIndex.TableName Constant

Pascal

```
const TableName = 2;
```

C++

```
TableName = 2;
```

Description

This is TableName, a member of class TDBXTablesIndex.

56.1.42.2.5 TDBXTablesIndex.TableType Constant

Pascal

```
const TableType = 3;
```

C++

```
TableType = 3;
```

Description

This is TableType, a member of class TDBXTablesIndex.

56.1.43 TDBXUsersColumns Class

Accesses columns by name in the **Users** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXUsersColumns

Pascal

```
TDBXUsersColumns = class;
```

C++

```
class TDBXUsersColumns;
```

File

DBXMetaDataNames

Description

TDBXUsersColumns is used to access columns by name in the **Users** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetUsers when reading metadata. To retrieve a metadata column, use this constant to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the column.

Column Name	Data Type of Column Data	Description
UserName (see page 10833)	String	User name.

See Also

TDBXMetaDataCommands (see page 10525)

TDBXUsersIndex (see page 10833)

56.1.43.1 TDBXUsersColumns Members

The following tables list the members exposed by TDBXUsersColumns.

Public Constants

	Name	Description
	UserName (see page 10833)	This is UserName, a member of class TDBXUsersColumns.

56.1.43.2 TDBXUsersColumns Constants

The constants of the TDBXUsersColumns class are listed here.

Public Constants

	Name	Description
	UserName (see page 10833)	This is UserName, a member of class TDBXUsersColumns.

56.1.43.2.1 TDBXUsersColumns.UserName Constant

Pascal

```
const UserName = 'UserName';
```

C++

```
UserName = 'UserName';
```

Description

This is UserName, a member of class TDBXUsersColumns.

56.1.44 TDBXUsersIndex Class

Accesses columns by ordinal in the **Users** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXUsersIndex

Pascal

```
TDBXUsersIndex = class;
```

C++

```
class TDBXUsersIndex;
```

File

DBXMetaDataNames

Description

TDBXUsersIndex is used to access columns by ordinal in the **Users** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommands.GetUsers when reading metadata. To retrieve a metadata column, use this constant to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the column.

Column Ordinal Name	Data Type of Column Data	Description
UserName (see page 10834)	String	User name.

See Also

TDBXMetaDataCommands (see page 10525)

TDBXUsersColumns (see page 10832)

56.1.44.1 TDBXUsersIndex Members

The following tables list the members exposed by TDBXUsersIndex.

Public Constants

	Name	Description
	Last (see page 10834)	This is Last, a member of class TDBXUsersIndex.
	UserName (see page 10834)	This is UserName, a member of class TDBXUsersIndex.

56.1.44.2 TDBXUsersIndex Constants

The constants of the TDBXUsersIndex class are listed here.

Public Constants

	Name	Description
	Last (see page 10834)	This is Last, a member of class TDBXUsersIndex.
	UserName (see page 10834)	This is UserName, a member of class TDBXUsersIndex.

56.1.44.2.1 TDBXUsersIndex.Last Constant

Pascal

```
const Last = 0;
```

C++

```
Last = 0;
```

Description

This is Last, a member of class TDBXUsersIndex.

56.1.44.2.2 TDBXUsersIndex.UserName Constant

Pascal

```
const UserName = 0;
```

C++

```
UserName = 0;
```

Description

This is UserName, a member of class TDBXUsersIndex.

56.1.45 TDBXViewsColumns Class

Accesses columns by name in the **Views** metadata.

Class Hierarchy

DBXMetaDataNames.TDBXViewsColumns

Pascal

```
TDBXViewsColumns = class;
```

C++

```
class TDBXViewsColumns;
```

File

DBXMetaDataNames

Description

TDBXViewsColumns is used to access columns by name in the **Views** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetViews when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Name	Data Type of Column Data	Description
CatalogName (see page 10835)	String	Catalog name.
SchemaName (see page 10836)	String	Schema name.
ViewName (see page 10836)	String	View name.
Definition (see page 10836)	String	SQL that was used to create the view.

See Also

TDBXMetaDataCommandsEx (see page 10531)

TDBXViewsIndex (see page 10836)

56.1.45.1 TDBXViewsColumns Members

The following tables list the members exposed by TDBXViewsColumns.

Public Constants

	Name	Description
	CatalogName (see page 10835)	This is CatalogName, a member of class TDBXViewsColumns.
	Definition (see page 10836)	This is Definition, a member of class TDBXViewsColumns.
	SchemaName (see page 10836)	This is SchemaName, a member of class TDBXViewsColumns.
	ViewName (see page 10836)	This is ViewName, a member of class TDBXViewsColumns.

56.1.45.2 TDBXViewsColumns Constants

The constants of the TDBXViewsColumns class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10835)	This is CatalogName, a member of class TDBXViewsColumns.
	Definition (see page 10836)	This is Definition, a member of class TDBXViewsColumns.
	SchemaName (see page 10836)	This is SchemaName, a member of class TDBXViewsColumns.
	ViewName (see page 10836)	This is ViewName, a member of class TDBXViewsColumns.

56.1.45.2.1 TDBXViewsColumns.CatalogName Constant

Pascal

```
const CatalogName = 'CatalogName';
```

C++

```
CatalogName = 'CatalogName';
```

Description

This is CatalogName, a member of class TDBXViewsColumns.

56.1.45.2.2 TDBXViewsColumns.Definition Constant

Pascal

```
const Definition = 'Definition';
```

C++

```
Definition = 'Definition';
```

Description

This is Definition, a member of class TDBXViewsColumns.

56.1.45.2.3 TDBXViewsColumns.SchemaName Constant

Pascal

```
const SchemaName = 'SchemaName';
```

C++

```
SchemaName = 'SchemaName';
```

Description

This is SchemaName, a member of class TDBXViewsColumns.

56.1.45.2.4 TDBXViewsColumns.ViewName Constant

Pascal

```
const ViewName = 'ViewName';
```

C++

```
ViewName = 'ViewName';
```

Description

This is ViewName, a member of class TDBXViewsColumns.

56.1.46 TDBXViewsIndex Class

Accesses columns by ordinal in the **Views** metadata.

Class Hierarchy

Pascal

```
TDBXViewsIndex = class;
```

C++

```
class TDBXViewsIndex;
```

File

DBXMetaDataNames

Description

TDBXViewsIndex is used to access columns by ordinal in the **Views** metadata collection. These columns are returned when TDBXCommand.Text' is TDBXMetaDataCommandsEx.GetViews when reading metadata. To retrieve metadata columns, use these constants to index the TDBXReader instance returned by TDBXCommand.ExecuteQuery.

This table describes the columns.

Column Ordinal Name	Data Type of Column Data	Description
CatalogName (see page 10838)	String	Catalog name.
SchemaName (see page 10838)	String	Schema name.
ViewName (see page 10838)	String	View name.
Definition (see page 10838)	String	SQL that was used to create the view.

See Also

TDBXMetaDataCommandsEx (see page 10531)

TDBXViewsColumns (see page 10834)

56.1.46.1 TDBXViewsIndex Members

The following tables list the members exposed by TDBXViewsIndex.

Public Constants

	Name	Description
	CatalogName (see page 10838)	This is CatalogName, a member of class TDBXViewsIndex.
	Definition (see page 10838)	This is Definition, a member of class TDBXViewsIndex.
	Last (see page 10838)	This is Last, a member of class TDBXViewsIndex.
	SchemaName (see page 10838)	This is SchemaName, a member of class TDBXViewsIndex.
	ViewName (see page 10838)	This is ViewName, a member of class TDBXViewsIndex.

56.1.46.2 TDBXViewsIndex Constants

The constants of the TDBXViewsIndex class are listed here.

Public Constants

	Name	Description
	CatalogName (see page 10838)	This is CatalogName, a member of class TDBXViewsIndex.
	Definition (see page 10838)	This is Definition, a member of class TDBXViewsIndex.
	Last (see page 10838)	This is Last, a member of class TDBXViewsIndex.
	SchemaName (see page 10838)	This is SchemaName, a member of class TDBXViewsIndex.
	ViewName (see page 10838)	This is ViewName, a member of class TDBXViewsIndex.

56.1.46.2.1 TDBXViewsIndex.CatalogName Constant

Pascal

```
const CatalogName = 0;
```

C++

```
CatalogName = 0;
```

Description

This is CatalogName, a member of class TDBXViewsIndex.

56.1.46.2.2 TDBXViewsIndex.Definition Constant

Pascal

```
const Definition = 3;
```

C++

```
Definition = 3;
```

Description

This is Definition, a member of class TDBXViewsIndex.

56.1.46.2.3 TDBXViewsIndex.Last Constant

Pascal

```
const Last = 3;
```

C++

```
Last = 3;
```

Description

This is Last, a member of class TDBXViewsIndex.

56.1.46.2.4 TDBXViewsIndex.SchemaName Constant

Pascal

```
const SchemaName = 1;
```

C++

```
SchemaName = 1;
```

Description

This is SchemaName, a member of class TDBXViewsIndex.

56.1.46.2.5 TDBXViewsIndex.ViewName Constant

Pascal

```
const ViewName = 2;
```

C++

```
ViewName = 2;
```

Description

This is ViewName, a member of class TDBXViewsIndex.

57 DBXpressWeb Namespace

This is namespace DBXpressWeb.

Topics

Name	Description
Classes (see page 10842)	The following table lists classes in this documentation.

Classes

	Name	Description
	TSQLQueryTableProducer (see page 10842)	TSQLQueryTableProducer assembles a sequence of HTML commands to display the result set of a query in a tabular format.

57.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TSQLQueryTableProducer (see page 10842)	TSQLQueryTableProducer assembles a sequence of HTML commands to display the result set of a query in a tabular format.

57.1.1 TSQLQueryTableProducer Class

TSQLQueryTableProducer assembles a sequence of HTML commands to display the result set of a query in a tabular format.

Class Hierarchy

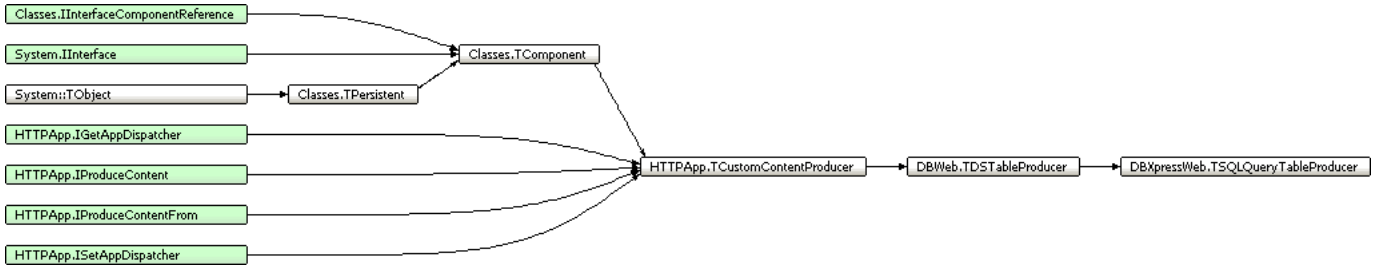

Pascal

```
TSQLQueryTableProducer = class(TDSTableProducer);
```

C++

```
class TSQLQueryTableProducer : public TDSTableProducer;
```

File

DBXpressWeb

Description

Use TSQLQueryTableProducer to produce an HTML table for the records of a parameterized query. The TSQLQuery object obtains its parameters from an HTTP request message. If the request is a GET message, the parameters for the query are the QueryFields property of the request object. If the request is a POST message, the parameters for the query are the ContentFields property of the request object.

To produce an HTML table for a query that has no parameters, use the TDataSetTableProducer object instead.

See the WebServ demo for an example of how to use TSQLQueryTableProducer.

See Also

TDataSet (see page 6894)

TQueryTableProducer (see page 8152)

TWebResponse (see page 15191)

TDataSetPageProducer (see page 11807)

57.1.1.1 TSQLQueryTableProducer Members

The following tables list the members exposed by TSQLQueryTableProducer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AcquireLayoutLock (see page 10296)	Calls BeginLayout (see page 10297) when the HTML table is not currently being laid out.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BeginLayout (see page 10297)	Increments a counter that indicates the number of times the computation of the column order was begun but not completed.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ColumnHeader (see page 10298)	Returns the string of HTML <TH> tags that define the column headers for the table.
	DefineFieldMap (see page 10298)	Builds the Fields array in DataLink (see page 10309).
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoCreateContent (see page 10299)	This is DoCreateContent, a member of class TDSTableProducer.
	DoFormatCell (see page 10299)	Generates an OnFormatCell (see page 10314) event.
	DoGetCaption (see page 10300)	Generates an OnGetCaption event.
	EndLayout (see page 10300)	Decrements a counter that indicates the number of times the computation of the column order has been begun but not completed.
	FormatCell (see page 10301)	Returns an HTML tag of a specified type that reflects a set of specified options.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDataSet (see page 10847)	Provides the abstract prototype of a method to access the DataSet property.
	GetDataSource (see page 10302)	Returns the value of the DataSource (see page 10309) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	LayoutChanged (see page 10303)	Causes the Columns (see page 10308) property to be recomputed when the dataset changes or changes are made to individual column objects.
	LinkActive (see page 10303)	Recomputes the Columns (see page 10308) property when the dataset becomes active.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 10848)	Adjusts the properties of TDSTableProducer when its helper objects are destroyed.

	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RowHeader (see page 10304)	Returns the HTML <TR> tag that defines each row of the HTML table.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetColumns (see page 10304)	Sets the Columns (see page 10308) property.
V	SetDataSet (see page 10848)	Provides the abstract prototype of a method to set the DataSet property.
	SetDataSource (see page 10305)	Sets the DataSource (see page 10309) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StoreColumns (see page 10305)	Indicates whether the Columns (see page 10308) property is stored when the TDSTableProducer (see page 10290) is saved.
	TableCaption (see page 10306)	Returns the HTML <Caption (see page 10307)> tag that defines the caption for the HTML table.
	TableHeader (see page 10306)	Returns the HTML <Table> tag for the HTML table.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	DataLink (see page 10309)	Manages the link to the dataset.
	DataSource (see page 10309)	Represents the link to the DataSet (see page 10309) of the TDSTableProducer (see page 10290).
R	InternalDataSource (see page 10311)	Represents the link to the dataset.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 10297)	Increments a counter that indicates the number of changes to the contents of the HTML table that have been initiated but not completed.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 10847)	Returns the HTML representation of the records in the dataset.
	ContentFromStream (see page 15107)	Returns the content produced in response to input read from a stream.
	ContentFromString (see page 15107)	Returns the content produced in response to an input string.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 10295)	Creates an instance of TDSTableProducer (see page 10290).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 10295)	Destroys an instance of TDSTableProducer (see page 10290).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 10301)	Decrements a counter that indicates the number of changes to the contents of the HTML table.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DataSet (see page 10309)	Specifies the dataset from which the data in the HTML table is taken.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.
	Editor (see page 10310)	Writes changes made in the property editor at design time to the TDSTableProducer (see page 10290) object.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCreateContent (see page 10853)	Occurs in descendants of TDSTableProducer when the Content method needs to generate the HTML table image.
	OnFormatCell (see page 10854)	Occurs when the HTML for a specified cell specified is about to be generated.
	OnGetTableCaption (see page 10854)	Occurs when the HTML table producer is about to generate the HTML for the table caption.

Published Properties

	Name	Description
	Caption (see page 10849)	Specifies the caption string that appears with the HTML table.
	CaptionAlignment (see page 10849)	Specifies where the Caption string should appear relative to the HTML table.
	Columns (see page 10850)	Describes the field bindings and display attributes of the columns in the HTML table.
	Footer (see page 10850)	Specifies a set of HTML commands that appear in the HTML document following the HTML table.
	Header (see page 10851)	Specifies a set of HTML commands that appear in the HTML document preceding the HTML table.
	MaxRows (see page 10851)	Specifies the maximum number of rows that appear in the HTML table.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Query (see page 10852)	Specifies the TSQLQuery object that TSQLQueryTableProducer (see page 10842) runs to obtain its data.
	RowAttributes (see page 10852)	Specifies the display attributes of all the rows in the HTML table.
	TableAttributes (see page 10853)	Specifies the display attributes that pertain to the entire HTML table produced by TDSTableProducer.
	Tag (see page 2808)	Stores an integer value as part of a component.

57.1.1.2 TSQLQueryTableProducer Methods

The methods of the TSQLQueryTableProducer class are listed here.

Protected Methods

	Name	Description
	GetDataSet (see page 10847)	Provides the abstract prototype of a method to access the DataSet property.
	Notification (see page 10848)	Adjusts the properties of TDSTableProducer when its helper objects are destroyed.
	SetDataSet (see page 10848)	Provides the abstract prototype of a method to set the DataSet property.

Public Methods

	Name	Description
	Content (see page 10847)	Returns the HTML representation of the records in the dataset.

57.1.1.2.1 TSQLQueryTableProducer.Content Method

Returns the HTML representation of the records in the dataset.

Pascal

```
function Content: string; override;
```

C++

```
virtual __fastcall AnsiString Content();
```

Description

Call Content to retrieve the HTML representation specified by the properties of the TSQLQueryTableProducer (see page 10842) object. Content calls the OnCreateContent (see page 10853) event handler. If the OnCreateContent (see page 10853) event handler indicates that the Content method should continue, Content returns the Header (see page 10851) property, followed by an HTML table image of the records in the dataset, followed by the Footer (see page 10850) property. If the OnCreateContent (see page 10853) event handler indicates that the Content method should not continue, Content returns an empty string.

The string returned by Content can be used as the Content property of the response to an HTTP request message.

See Also

Footer (see page 10310)

Header (see page 10311)

OnCreateContent (see page 10313)

TWebRequest (see page 15161)

Content (see page 15207)

57.1.1.2.2 TSQLQueryTableProducer.GetDataSet Method

Provides the abstract prototype of a method to access the DataSet property.

Pascal

```
function GetDataSet: TDataSet; override;
```

C++

```
virtual __fastcall TDataSet GetDataSet();
```

Description

Descendants of TDSTableProducer implement the GetDataSet method to access their specific implementation of the DataSet property. Descendants of TDSTableProducer restrict the DataSet property to different classes of TDataSet objects.

See Also

DataSet (see page 10309)

SetDataSet (see page 10305)

57.1.1.2.3 TSQLQueryTableProducer.Notification Method

Adjusts the properties of TDSTableProducer when its helper objects are destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification overrides the inherited method to respond to messages that the DataSource or one of its field objects is about to be destroyed. Notification sets the DataSource property to nil (Delphi) or NULL (C++), or unbinds the column that references the field component.

See Also

DataSource (see page 10309)

Columns (see page 10308)

57.1.1.2.4 TSQLQueryTableProducer.SetDataSet Method

Provides the abstract prototype of a method to set the DataSet property.

Pascal

```
procedure SetDataSet(ADataSet: TDataSet); override;
```

C++

```
virtual __fastcall SetDataSet(TDataSet ADataSet);
```

Description

Descendants of TDSTableProducer override SetDataSet to ensure that the DataSet property can only be set to an appropriate descendant of TDataSet.

See Also

DataSet (see page 10309)

GetDataSet (see page 10302)

57.1.1.3 TSQLQueryTableProducer Properties

The properties of the TSQLQueryTableProducer class are listed here.

Published Properties

	Name	Description
	Caption (see page 10849)	Specifies the caption string that appears with the HTML table.
	CaptionAlignment (see page 10849)	Specifies where the Caption string should appear relative to the HTML table.
	Columns (see page 10850)	Describes the field bindings and display attributes of the columns in the HTML table.
	Footer (see page 10850)	Specifies a set of HTML commands that appear in the HTML document following the HTML table.
	Header (see page 10851)	Specifies a set of HTML commands that appear in the HTML document preceding the HTML table.
	MaxRows (see page 10851)	Specifies the maximum number of rows that appear in the HTML table.
	Query (see page 10852)	Specifies the TSQLQuery object that TSQLQueryTableProducer (see page 10842) runs to obtain its data.
	RowAttributes (see page 10852)	Specifies the display attributes of all the rows in the HTML table.
	TableAttributes (see page 10853)	Specifies the display attributes that pertain to the entire HTML table produced by TDSTableProducer.

57.1.1.3.1 TSQLQueryTableProducer.Caption Property

Specifies the caption string that appears with the HTML table.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Set Caption to add a caption to the HTML table that represents the dataset. Caption indicates the text that should appear inside the HTML CAPTION tag.

See Also

CaptionAlignment (see page 10308)

OnGetTableCaption (see page 10314)

Embedding Tables in HTML Documents

57.1.1.3.2 TSQLQueryTableProducer.CaptionAlignment Property

Specifies where the Caption string should appear relative to the HTML table.

Pascal

```
property CaptionAlignment: THTMLCaptionAlignment;
```

C++

```
__property THTMLCaptionAlignment CaptionAlignment;
```

Description

Set CaptionAlignment to specify whether the Caption should appear above or below the HTML table.

CaptionAlignment is the value of the ALIGN option on the HTML CAPTION tag.

See Also

Caption (see page 10307)

Embedding Tables in HTML Documents

57.1.1.3.3 TSQLQueryTableProducer.Columns Property

Describes the field bindings and display attributes of the columns in the HTML table.

Pascal

```
property Columns: THTMLTableColumns;
```

C++

```
__property THTMLTableColumns Columns;
```

Description

Use Columns to read or set the field bindings and display attributes of the columns in the HTML table. Each THTMLTableColumn object specifies the display attributes of the corresponding column. Each column object also specifies the name of the field the column represents. The field binding of a column designates a field within the dataset specified by the DataSet property.

Columns can be set at design time through a property editor, or programmatically at runtime.

See Also

DataSet (see page 10309)

DefineFieldMap (see page 10298)

Editor (see page 10310)

RowAttributes (see page 10312)

SetColumns (see page 10304)

StoreColumns (see page 10305)

TableAttributes (see page 10313)

THTMLTableColumn (see page 10318)

THTMLTableColumns (see page 10327)

Specifying the Columns

57.1.1.3.4 TSQLQueryTableProducer.Footer Property

Specifies a set of HTML commands that appear in the HTML document following the HTML table.

Pascal

```
property Footer: TStrings;
```

C++

```
__property TStrings Footer;
```

Description

Set Footer to embed the representation of DataSet in a larger document. The Content method of TDSTableProducer descendants returns any Header commands, followed by the HTML table that represents the DataSet, followed by any Footer commands.

See Also

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

DataSet (see page 10309)

Header (see page 10311)

Embedding Tables in HTML Documents

57.1.1.3.5 TSQLQueryTableProducer.Header Property

Specifies a set of HTML commands that appear in the HTML document preceding the HTML table.

Pascal

```
property Header: TStrings;
```

C++

```
__property TStrings Header;
```

Description

Set Header to embed the representation of DataSet in a larger document. The Content method of TDSTableProducer descendants returns any Header commands, followed by the HTML table that represents the DataSet, followed by any Footer commands.

See Also

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

DataSet (see page 10309)

Footer (see page 10310)

Embedding Tables in HTML Documents

57.1.1.3.6 TSQLQueryTableProducer.MaxRows Property

Specifies the maximum number of rows that appear in the HTML table.

Pascal

```
property MaxRows: Integer;
```

C++

```
__property int MaxRows;
```

Description

Set MaxRows to change the upper limit on the size of the HTML table produced to represent the DataSet. MaxRows is initialized to 20.

If DataSet contains more than MaxRows records, only the first MaxRows records appear in the HTML commands produced by Content. If DataSet contains fewer than MaxRows records, the HTML table produced by TDSTableProducer will contain only enough rows to display all the records in DataSet.

See Also

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

DataSet (see page 10309)

Specifying the Row Attributes

57.1.1.3.7 TSQLQueryTableProducer.Query Property

Specifies the TSQLQuery object that TSQLQueryTableProducer (see page 10842) runs to obtain its data.

Pascal

```
property Query: TSQLQuery;
```

C++

```
__property TSQLQuery Query;
```

Description

Set Query to specify the query to run to obtain the data for the HTML table. Use Query to set the query for the TSQLQueryTableProducer (see page 10842) rather than the DataSource property: TSQLQueryTableProducer (see page 10842) generates its data source internally.

The individual field bindings specified by the THTMLTableColumn objects in the Columns (see page 10850) property refer to fields in DataSet.

Note: Query is identical to the DataSet property, except Query does not use the TDataSet base class of the query object.

See Also

DataSet (see page 10309)

57.1.1.3.8 TSQLQueryTableProducer.RowAttributes Property

Specifies the display attributes of all the rows in the HTML table.

Pascal

```
property RowAttributes: THTMLTableRowAttributes;
```

C++

```
__property THTMLTableRowAttributes RowAttributes;
```

Description

Set RowAttributes to a THTMLTableRowAttributes object that describes the display attributes of the rows in the HTML table. Setting RowAttributes calls the Assign method of the THTMLTableRowAttributes object stored internally by TDSTableProducer. Thus, the memory for any row attributes object assigned to the RowAttributes property is never owned or freed by TDSTableProducer.

Note: Not all the properties of the row attributes object are copied when one row attributes object is assigned to another. To ensure that all the attributes represented by RowAttributes are set appropriately, read RowAttributes and set the individual properties of the THTMLTableRowAttributes object.

See Also

TableAttributes (see page 10313)

Title (see page 10327)

AssignTo (see page 15297)

Specifying the Row Attributes

57.1.1.3.9 TSQLQueryTableProducer.TableAttributes Property

Specifies the display attributes that pertain to the entire HTML table produced by TDSTableProducer.

Pascal

```
property TableAttributes: THTMLTableAttributes;
```

C++

```
__property THTMLTableAttributes TableAttributes;
```

Description

Set TableAttributes to a THTMLTableAttributes object that describes the display attributes for the HTML TABLE tag. Setting TableAttributes calls the Assign method of the THTMLTableAttributes object stored internally by TDSTableProducer. Thus, the memory for any table attributes object assigned to the TableAttributes property is never owned or freed by TDSTableProducer.

Note: Not all the properties of the table attributes object are copied when one table attributes object is assigned to another. To ensure that all the attributes represented by TableAttributes are set appropriately, read TableAttributes and set the individual properties of the THTMLTableAttributes object.

See Also

RowAttributes (see page 10312)

AssignTo (see page 15291)

Title (see page 10327)

Specifying the Table Attributes

57.1.1.4 TSQLQueryTableProducer Events

The events of the TSQLQueryTableProducer class are listed here.

Published Events

	Name	Description
	OnCreateContent (see page 10853)	Occurs in descendants of TDSTableProducer when the Content method needs to generate the HTML table image.
	OnFormatCell (see page 10854)	Occurs when the HTML for a specified cell specified is about to be generated.
	OnGetTableCaption (see page 10854)	Occurs when the HTML table producer is about to generate the HTML for the table caption.

57.1.1.4.1 TSQLQueryTableProducer.OnCreateContent Event

Occurs in descendants of TDSTableProducer when the Content method needs to generate the HTML table image.

Pascal

```
property OnCreateContent: TCreateContentEvent;
```

C++

```
__property TCreateContentEvent OnCreateContent;
```

Description

Write an OnCreateContent event handler to conditionally prevent the Content method from producing any HTML commands. Set the Continue parameter to false to make Content return an empty string.

See Also

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

docreatecontent (see page 10299)

57.1.1.4.2 TSQLQueryTableProducer.OnFormatCell Event

Occurs when the HTML for a specified cell specified is about to be generated.

Pascal

```
property OnFormatCell: THTMLFormatCellEvent;
```

C++

```
__property THTMLFormatCellEvent OnFormatCell;
```

Description

Write an OnFormatCell event handler to change the appearance or contents of individual cells in the HTML table from the default values produced by TDSTableProducer.

See Also

Columns (see page 10308)

Content (see page 10282)

Content (see page 10847)

Content (see page 8157)

DoFormatCell (see page 10299)

DisplayText (see page 7162)

Align (see page 10324)

BgColor (see page 10325)

Custom (see page 10325)

VAlign (see page 10327)

57.1.1.4.3 TSQLQueryTableProducer.OnGetTableCaption Event

Occurs when the HTML table producer is about to generate the HTML for the table caption.

Pascal

```
property OnGetTableCaption: THTMLGetTableCaptionEvent;
```

C++

```
__property THTMLGetTableCaptionEvent OnGetTableCaption;
```

Description

Write an OnGetTableCaption event handler to change the value or placement of the HTML table caption. The Caption and Alignment parameters are originally set to the values of the Caption and CaptionAlignment properties, respectively. The OnGetTableCaption event handler can change these values to alter the caption for the HTML table returned by the Content

method.

See Also

Caption (see page 10307)

CaptionAlignment (see page 10308)

Content (see page 15106)

DoGetCaption (see page 10300)

58 DdeMan Namespace

This is namespace DdeMan.

Topics

Name	Description
Classes (see page 10858)	The following table lists classes in this documentation.
Functions (see page 10910)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 10911)	The following table lists structs, records, enums in this documentation.
Types (see page 10912)	The following table lists types in this documentation.
Variables (see page 10913)	The following table lists variables in this documentation.

Classes

	Name	Description
	TDdeClientConv (see page 10858)	TDdeClientConv represents a DDE conversation with a DDE server application.
	TDdeClientItem (see page 10878)	TDDEClientItem defines an item in a DDE conversation.
	TDdeMgr (see page 10886)	This is class DdeMan.TDdeMgr.
	TDdeServerConv (see page 10891)	TDdeServerConv represents a DDE topic of conversation with a DDE client.
	TDdeServerItem (see page 10898)	TDdeServerItem defines the item of a DDE conversation.

Functions

	Name	Description
	GetPasteLinkInfo (see page 10910)	This is function DdeMan.GetPasteLinkInfo.

Structs, Records, Enums

	Name	Description
	TDataMode (see page 10911)	TDataMode indicates whether a stage in a DDE conversation happens automatically.

Types

Name	Description
TMacroEvent (see page 10912)	TMacroEvent is the type for event handlers that implement the response of a DDE server to a macro.

Variables

Name	Description
ddeMgr (see page 10913)	This is variable DdeMan.ddeMgr.

58.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TDdeClientConv (see page 10858)	TDdeClientConv represents a DDE conversation with a DDE server application.
	TDdeClientItem (see page 10878)	TDDEClientItem defines an item in a DDE conversation.
	TDdeMgr (see page 10886)	This is class DdeMan.TDdeMgr.
	TDdeServerConv (see page 10891)	TDdeServerConv represents a DDE topic of conversation with a DDE client.
	TDdeServerItem (see page 10898)	TDdeServerItem defines the item of a DDE conversation.

58.1.1 TDdeClientConv Class

TDdeClientConv represents a DDE conversation with a DDE server application.

Class Hierarchy

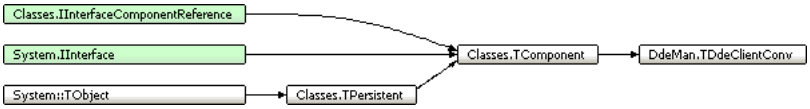

Pascal

```
TDdeClientConv = class(TComponent);
```

C++

```
class TDdeClientConv : public TComponent;
```

File

DdeMan

Description

Use TDdeClientConv with a TDdeClientItem (see page 10878) object to enable an application to act as a DDE client. TDdeClientConv provides many properties, events, and methods that provide services to

Link to a specified DDE server with a specified DDE topic.

Send data or macro commands to the DDE server after a link has been established.

Specify a format for the data that is sent to the DDE server.

Allow event handlers to respond when the DDE link is opened or closed.

TDdeClientItem (see page 10878) objects are associated with a TDdeClientConv by setting their DdeConv property.

See Also

TDdeClientItem (see page 10878)

TDdeServerConv (see page 10891)

TDdeServerItem (see page 10898)

58.1.1.1 TDdeClientConv Members

The following tables list the members exposed by TDdeClientConv.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeLink (see page 10863)	This is ChangeLink, a member of class TDdeClientConv.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ClearItems (see page 10864)	This is ClearItems, a member of class TDdeClientConv.
	Close (see page 10864)	This is Close, a member of class TDdeClientConv.
	CreateDdeConv (see page 10864)	This is CreateDdeConv, a member of class TDdeClientConv.
V	DefineProperties (see page 10865)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetCliItemByCtrl (see page 10866)	This is GetCliItemByCtrl, a member of class TDdeClientConv.
	GetCliItemByName (see page 10867)	This is GetCliItemByName, a member of class TDdeClientConv.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 10867)	Initializes the component after the form file has been read into memory.
V	Notification (see page 10867)	Forwards notification messages to all owned components.
	OnAttach (see page 10868)	This is OnAttach, a member of class TDdeClientConv.
	OnDetach (see page 10868)	This is OnDetach, a member of class TDdeClientConv.
	OnSetItem (see page 10868)	This is OnSetItem, a member of class TDdeClientConv.
	Open (see page 10869)	This is Open, a member of class TDdeClientConv.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadLinkInfo (see page 10871)	This is ReadLinkInfo, a member of class TDdeClientConv.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteLinkInfo (see page 10873)	This is WriteLinkInfo, a member of class TDdeClientConv.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CloseLink (see page 10864)	Terminates an ongoing DDE conversation.
V	Create (see page 10862)	Creates an instance of TDdeClientConv (see page 10858).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 10862)	Destroys an instance of TDdeClientConv (see page 10858).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteMacro (see page 10865)	Sends a macro command string to the DDE server application.
	ExecuteMacroLines (see page 10866)	Sends a list of several macro commands to a DDE server application.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.

	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	OpenLink (see page 10869)	Initiates a new DDE conversation.
	PasteLink (see page 10869)	Opens a DDE conversation with the current Clipboard object.
	PokeData (see page 10870)	Sends data to a DDE server application.
	PokeDataLines (see page 10870)	Sends multiple lines of data to a DDE server application.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RequestData (see page 10871)	Requests data from a DDE server.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetLink (see page 10872)	Specifies the service and topic of a DDE conversation and attempts to open the link if ConnectMode (see page 10873) is ddeAutomatic.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartAdvise (see page 10872)	Starts an advise loop for all of the TDdeClientItems associated with the current DDE conversation.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Conv (see page 10874)	Indicates the handle for the current DDE conversation.
	DataMode (see page 10874)	Governs whether portions of the DDE conversation execute automatically.
R	DdeFmt (see page 10874)	Represents the clipboard format used for submitting or requesting a data item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WaitStat (see page 10877)	Indicates whether the server is in the process of servicing a DDE macro that prohibits asynchronous DDE transactions.

Published Events

	Name	Description
	OnClose (see page 10877)	Occurs when the DDE conversation terminates.
	OnOpen (see page 10878)	Occurs immediately after the DDE link to the server has been established.

Published Properties

	Name	Description
	ConnectMode (see page 10873)	Determines whether the link to the DDE server is established automatically when the form is opened.
	DdeService (see page 10875)	Specifies the DDE server application to be linked to a DDE client.
	DdeTopic (see page 10875)	Specifies the topic of a DDE conversation
	FormatChars (see page 10876)	Determines if certain characters are filtered out of text data transferred from a DDE server application.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ServiceApplication (see page 10876)	Specifies the main executable file name (with path, if necessary) of a DDE server application, without the .EXE extension.
	Tag (see page 2808)	Stores an integer value as part of a component.

58.1.1.2 TDdeClientConv.Create Constructor

Creates an instance of TDdeClientConv (see page 10858).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDdeClientConv(TComponent * AOwner);
```

Description

Call Create to create an instance of TDdeClientConv (see page 10858). After calling the inherited constructor, Create creates the helper objects used to manage individual DDE client items.

See Also

Create (see page 2766)

TDdeClientItem (see page 10878)

58.1.1.3 TDdeClientConv.Destroy Destructor

Destroys an instance of TDdeClientConv (see page 10858).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDdeClientConv();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TDdeClientConv (see page 10858) reference is not nil, and only then calls Destroy.

Destroy closes the link to the server and frees up all helper objects before calling the inherited destructor.

See Also

CloseLink (see page 10864)

Free (see page 22300)

58.1.1.4 TDdeClientConv Methods

The methods of the TDdeClientConv class are listed here.

Protected Methods

	Name	Description
	ChangeLink (see page 10863)	This is ChangeLink, a member of class TDdeClientConv.
	ClearItems (see page 10864)	This is ClearItems, a member of class TDdeClientConv.
	Close (see page 10864)	This is Close, a member of class TDdeClientConv.
	CreateDdeConv (see page 10864)	This is CreateDdeConv, a member of class TDdeClientConv.
V	DefineProperties (see page 10865)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	GetCliItemByCtrl (see page 10866)	This is GetCliItemByCtrl, a member of class TDdeClientConv.
	GetCliItemByName (see page 10867)	This is GetCliItemByName, a member of class TDdeClientConv.
V	Loaded (see page 10867)	Initializes the component after the form file has been read into memory.
V	Notification (see page 10867)	Forwards notification messages to all owned components.
	OnAttach (see page 10868)	This is OnAttach, a member of class TDdeClientConv.
	OnDetach (see page 10868)	This is OnDetach, a member of class TDdeClientConv.
	OnSetItem (see page 10868)	This is OnSetItem, a member of class TDdeClientConv.
	Open (see page 10869)	This is Open, a member of class TDdeClientConv.
	ReadLinkInfo (see page 10871)	This is ReadLinkInfo, a member of class TDdeClientConv.
	WriteLinkInfo (see page 10873)	This is WriteLinkInfo, a member of class TDdeClientConv.

Public Methods

	Name	Description
	CloseLink (see page 10864)	Terminates an ongoing DDE conversation.
	ExecuteMacro (see page 10865)	Sends a macro command string to the DDE server application.
	ExecuteMacroLines (see page 10866)	Sends a list of several macro commands to a DDE server application.
	OpenLink (see page 10869)	Initiates a new DDE conversation.
	PasteLink (see page 10869)	Opens a DDE conversation with the current Clipboard object.
	PokeData (see page 10870)	Sends data to a DDE server application.
	PokeDataLines (see page 10870)	Sends multiple lines of data to a DDE server application.
	RequestData (see page 10871)	Requests data from a DDE server.
	SetLink (see page 10872)	Specifies the service and topic of a DDE conversation and attempts to open the link if ConnectMode (see page 10873) is ddeAutomatic.
	StartAdvise (see page 10872)	Starts an advise loop for all of the TDdeClientItems associated with the current DDE conversation.

58.1.1.4.1 TDdeClientConv.ChangeLink Method

Pascal

```
function ChangeLink(const App: string; const Topic: string; const Item: string): Boolean;
```

C++

```
__fastcall Boolean ChangeLink(const AnsiString App, const AnsiString Topic, const AnsiString
Item);
```

Description

This is ChangeLink, a member of class TDdeClientConv.

58.1.1.4.2 TDdeClientConv.ClearItems Method

Pascal

```
procedure ClearItems;
```

C++

```
__fastcall ClearItems();
```

Description

This is ClearItems, a member of class TDdeClientConv.

58.1.1.4.3 TDdeClientConv.Close Method

Pascal

```
procedure Close; dynamic;
```

C++

```
__fastcall Close();
```

Description

This is Close, a member of class TDdeClientConv.

58.1.1.4.4 TDdeClientConv.CloseLink Method

Terminates an ongoing DDE conversation.

Pascal

```
procedure CloseLink;
```

C++

```
__fastcall CloseLink();
```

Description

Use CloseLink when all communication with the DDE server is finished. After a link is closed, no DDE communication can take place between the DDE client and server until another link is opened.

See Also

OnClose (see page 10877)

OpenLink (see page 10869)

58.1.1.4.5 TDdeClientConv.CreateDdeConv Method

Pascal

```
function CreateDdeConv(FHszApp: HSZ; FHszTopic: HSZ): Boolean;
```

C++

```
__fastcall Boolean CreateDdeConv(HSZ FHszApp, HSZ FHszTopic);
```

Description

This is CreateDdeConv, a member of class TDdeClientConv.

58.1.1.4.6 TDdeClientConv.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor (see page 2839)

DefineBinaryProperty (see page 2837)

DefineProperties (see page 2931)

DefineProperty (see page 2838)

58.1.1.4.7 TDdeClientConv.ExecuteMacro Method

Sends a macro command string to the DDE server application.

Pascal

```
function ExecuteMacro(Cmd: PChar; waitFlg: Boolean): Boolean;
```

C++

```
__fastcall Boolean ExecuteMacro(const char * Cmd, Boolean waitFlg);
```

Description

Call ExecuteMacro to send a single macro command to the server application. To send multiple macro commands, use ExecuteMacroLines (see page 10866) instead. ExecuteMacro returns true if the macro was successfully passed to the DDE server application. If ExecuteMacro was unable to send a command string, ExecuteMacro returns false.

Cmd is a null-terminated string that contains the macro to be executed by the DDE server application. The actual value of Cmd depends on the DDE server application. See the documentation of the DDE server application for the command strings it will accept.

WaitFlg determines if this DDE client should wait until the DDE server application finishes executing the macro before allowing another DDE transaction to succeed. If WaitFlg is set to true, subsequent calls to ExecuteMacro, ExecuteMacroLines (see page 10866), PokeData (see page 10870), PokeDataLines (see page 10870), and RequestData (see page 10871) will fail until the DDE server application completes the macro.

Attempting to execute a macro or poke data before a DDE server application completes a currently executing macro may cause the executing macro to fail or to produce unpredictable results. See the documentation of the DDE server application for the results of sending command strings or poking data before macro execution has completed.

Note: ExecuteMacro returns true if the macro command was successfully passed to the DDE server. A true value does not ensure that the macro command will execute successfully once it has been accepted by the server.

See Also

ExecuteMacroLines (see page 10866)

PokeData (see page 10870)

PokeDataLines (see page 10870)

RequestData (see page 10871)

WaitStat (see page 10877)

58.1.1.4.8 TDdeClientConv.ExecuteMacroLines Method

Sends a list of several macro commands to a DDE server application.

Pascal

```
function ExecuteMacroLines(Cmd: TStrings; waitFlg: Boolean): Boolean;
```

C++

```
__fastcall Boolean ExecuteMacroLines(TStrings Cmd, Boolean waitFlg);
```

Description

Call ExecuteMacroLines to send several macro commands to the server application. To send a single macro command, use ExecuteMacro (see page 10865) instead. ExecuteMacroLines returns true if the macro list was successfully passed to the DDE server application. If ExecuteMacroLines is unable to send a list of command strings, it returns false.

Cmd contains the list of macros to be executed by the DDE server application. WaitFlg determines if this DDE client should wait until the DDE server application finishes executing all the macros before allowing another DDE transaction to be sent. If WaitFlg is set to true, subsequent calls to ExecuteMacro (see page 10865), ExecuteMacroLines, PokeData (see page 10870), PokeDataLines (see page 10870), and RequestData (see page 10871) will fail until the DDE server application completes all of the macros.

See Also

ExecuteMacro (see page 10865)

PokeData (see page 10870)

PokeDataLines (see page 10870)

RequestData (see page 10871)

WaitStat (see page 10877)

58.1.1.4.9 TDdeClientConv.GetCliItemByCtrl Method

Pascal

```
function GetCliItemByCtrl(ACtrl: TDdeClientItem): TPersistent;
```

C++

```
__fastcall TPersistent * GetCliItemByCtrl(TDdeClientItem ACtrl);
```

Description

This is GetCliItemByCtrl, a member of class TDdeClientConv.

58.1.1.4.10 TDdeClientConv.GetCliItemByName Method

Pascal

```
function GetCliItemByName(const ItemName: string): TPersistent;
```

C++

```
__fastcall TPersistent * GetCliItemByName(const AnsiString ItemName);
```

Description

This is GetCliItemByName, a member of class TDdeClientConv.

58.1.1.4.11 TDdeClientConv.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState (see page 2804)

Create (see page 2766)

TReader (see page 2933)

58.1.1.4.12 TDdeClientConv.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

58.1.1.4.13 TDdeClientConv.OnAttach Method

Pascal

```
procedure OnAttach(aCtrl: TDdeClientItem);
```

C++

```
__fastcall OnAttach(TDdeClientItem aCtrl);
```

Description

This is OnAttach, a member of class TDdeClientConv.

58.1.1.4.14 TDdeClientConv.OnDetach Method

Pascal

```
procedure OnDetach(aCtrl: TDdeClientItem);
```

C++

```
__fastcall OnDetach(TDdeClientItem aCtrl);
```

Description

This is OnDetach, a member of class TDdeClientConv.

58.1.1.4.15 TDdeClientConv.OnSetItem Method

Pascal

```
function OnSetItem(aCtrl: TDdeClientItem; const S: string): Boolean;
```

C++

```
__fastcall Boolean OnSetItem(TDdeClientItem aCtrl, const AnsiString S);
```

Description

This is OnSetItem, a member of class TDdeClientConv.

58.1.1.4.16 TDdeClientConv.Open Method

Pascal

```
procedure Open; dynamic;
```

C++

```
__fastcall Open();
```

Description

This is Open, a member of class TDdeClientConv.

58.1.1.4.17 TDdeClientConv.OpenLink Method

Initiates a new DDE conversation.

Pascal

```
function OpenLink: Boolean;
```

C++

```
__fastcall Boolean OpenLink();
```

Description

Call OpenLink to establish a new DDE conversation with the DdeService (see page 10875). If the server application is not running, OpenLink will launch the ServiceApplication (see page 10876). If the conversation is successfully opened, an OnOpen (see page 10878) event occurs and OpenLink returns true. If the conversation isn't successfully opened, OpenLink returns false.

Note: If ConnectMode (see page 10873) is ddeAutomatic, OpenLink is called automatically when the DdeTopic (see page 10875) and DdeService (see page 10875) are specified by SetLink (see page 10872).

See Also

ConnectMode (see page 10873)

DdeTopic (see page 10875)

DdeService (see page 10875)

OnOpen (see page 10878)

PasteLink (see page 10869)

ServiceApplication (see page 10876)

SetLink (see page 10872)

58.1.1.4.18 TDdeClientConv.PasteLink Method

Opens a DDE conversation with the current Clipboard object.

Pascal

```
function PasteLink: Boolean;
```

C++

```
__fastcall Boolean PasteLink();
```

Description

Call PasteLink to establish a DDE link to an object on the Clipboard. PasteLink sets the DdeTopic (see page 10875) or DdeService (see page 10875) properties from the Clipboard. PasteLink returns true if the link is successfully established.

See Also

DdeTopic (see page 10875)

DdeService (see page 10875)

OpenLink (see page 10869)

58.1.1.4.19 TDdeClientConv.PokeData Method

Sends data to a DDE server application.

Pascal

```
function PokeData(const Item: string; Data: PChar): Boolean;
```

C++

```
__fastcall Boolean PokeData(const AnsiString Item, const char * Data);
```

Description

Use PokeData to transfer text data to a DDE server that supports poked data. Item specifies the linked item in the DDE server. Data is a null-terminated string that specifies the text data to transfer.

The value of the DDE item depends on the linked DDE server application. Item is typically a selectable portion of text, such as a spreadsheet cell or a database field in an edit box. If the DDE server is a Delphi C++Builder application, Item is the name of the linked DDE server component.

The usual direction of data flow is from the DDE server to the DDE client application. Some DDE server applications won't accept poked data. PokeData returns true if the data was successfully transferred, false if the data was not successfully transferred.

To poke a string list rather than a single string, use the PokeDataLines (see page 10870) method.

Note: If either the ExecuteMacro (see page 10865) or ExecuteMacroLines (see page 10866) method was called with its WaitFlg parameter set to true, PokeData will automatically fail until the server application has completed executing the macro. The WaitStat (see page 10877) property indicates when PokeData will attempt to poke data to the server.

See Also

ExecuteMacro (see page 10865)

ExecuteMacroLines (see page 10866)

PokeDataLines (see page 10870)

WaitStat (see page 10877)

58.1.1.4.20 TDdeClientConv.PokeDataLines Method

Sends multiple lines of data to a DDE server application.

Pascal

```
function PokeDataLines(const Item: string; Data: TStrings): Boolean;
```

C++

```
__fastcall Boolean PokeDataLines(const AnsiString Item, TStrings Data);
```

Description

Use PokeDataLines to transfer multiple lines of text data to a DDE server that supports poked data. Item specifies the linked item in the DDE server. Data is a set of strings that specifies the text data to transfer.

The value of the DDE item depends on the linked DDE server application. Item is typically a selectable portion of text, such as a spreadsheet cell or a database field in an edit box.

The usual direction of data flow is from the DDE server to the DDE client application. Some DDE server applications won't accept poked data. PokeData (see page 10870) returns true if the data was successfully transferred, false if the data was not successfully transferred.

To poke a single string rather than a string list, use the PokeData (see page 10870) method.

Note: If either the ExecuteMacro (see page 10865) or ExecuteMacroLines (see page 10866) method was called with its WaitFlg parameter set to true, PokeData (see page 10870) will automatically fail until the server application has completed executing the macro. The WaitStat (see page 10877) property indicates when PokeData (see page 10870) will attempt to poke data to the server.

See Also

ExecuteMacro (see page 10865)

ExecuteMacroLines (see page 10866)

PokeData (see page 10870)

WaitStat (see page 10877)

58.1.1.4.21 TDdeClientConv.ReadLinkInfo Method

Pascal

```
procedure ReadLinkInfo(Reader: TReader);
```

C++

```
__fastcall ReadLinkInfo(TReader Reader);
```

Description

This is ReadLinkInfo, a member of class TDdeClientConv.

58.1.1.4.22 TDdeClientConv.RequestData Method

Requests data from a DDE server.

Pascal

```
function RequestData(const Item: string): PChar;
```

C++

```
__fastcall const char * RequestData(const AnsiString Item);
```

Description

Call RequestData to send a request to the DDE server for the data named by Item. RequestData returns the value of the named

DDE server item.

Note: RequestData automatically allocates memory to store the returned data, but applications must dispose of this string after processing it. This is done with the StrDispose procedure.

If the DDE server contains DDE items that can be continually updated, data can be received and updated automatically by creating TDdeClientItem (see page 10878) objects for those items.

The value of the DDE item depends on the linked DDE server application. Item is typically a selectable portion of text, such as a spreadsheet cell or a database field in an edit box.

Note: See the documentation for the DDE server application for specific information about specifying a DDE item.

See Also

StrDispose (see page 22947)

TDdeClientItem (see page 10878)

58.1.1.4.23 TDdeClientConv.SetLink Method

Specifies the service and topic of a DDE conversation and attempts to open the link if ConnectMode (see page 10873) is ddeAutomatic.

Pascal

```
function SetLink(const Service: string; const Topic: string): Boolean;
```

C++

```
__fastcall Boolean SetLink(const AnsiString Service, const AnsiString Topic);
```

Description

Call SetLink to fully specify a DDE conversation. If ConnectMode (see page 10873) is ddeManual, calling SetLink is equivalent to setting the DdeTopic (see page 10875) and DdeService (see page 10875) properties, and clearing all data from any associated TDdeClientItem (see page 10878) objects. If ConnectMode (see page 10873) is ddeAutomatic, SetLink opens the link to the DDE server as well.

The Service and Topic parameters specify the values for the DdeTopic (see page 10875) and DdeService (see page 10875) properties, respectively.

See Also

ConnectMode (see page 10873)

DdeService (see page 10875)

DdeTopic (see page 10875)

OpenLink (see page 10869)

58.1.1.4.24 TDdeClientConv.StartAdvise Method

Starts an advise loop for all of the TDdeClientItems associated with the current DDE conversation.

Pascal

```
function StartAdvise: Boolean;
```

C++

```
__fastcall Boolean StartAdvise();
```

Description

Call StartAdvise to start an advise loop for all of the TDdeClientItems associated with the current DDE conversation. StartAdvise returns true if loop startup is successful.

See Also

TDdeClientItem (see page 10878)

58.1.1.4.25 TDdeClientConv.WriteLinkInfo Method

Pascal

```
procedure WriteLinkInfo(Writer: TWriter);
```

C++

```
__fastcall WriteLinkInfo(TWriter Writer);
```

Description

This is WriteLinkInfo, a member of class TDdeClientConv.

58.1.1.5 TDdeClientConv Properties

The properties of the TDdeClientConv class are listed here.

Public Properties

	Name	Description
R	Conv (see page 10874)	Indicates the handle for the current DDE conversation.
	DataMode (see page 10874)	Governs whether portions of the DDE conversation execute automatically.
R	DdeFmt (see page 10874)	Represents the clipboard format used for submitting or requesting a data item.
R	WaitStat (see page 10877)	Indicates whether the server is in the process of servicing a DDE macro that prohibits asynchronous DDE transactions.

Published Properties

	Name	Description
	ConnectMode (see page 10873)	Determines whether the link to the DDE server is established automatically when the form is opened.
	DdeService (see page 10875)	Specifies the DDE server application to be linked to a DDE client.
	DdeTopic (see page 10875)	Specifies the topic of a DDE conversation
	FormatChars (see page 10876)	Determines if certain characters are filtered out of text data transferred from a DDE server application.
	ServiceApplication (see page 10876)	Specifies the main executable file name (with path, if necessary) of a DDE server application, without the .EXE extension.

58.1.1.5.1 TDdeClientConv.ConnectMode Property

Determines whether the link to the DDE server is established automatically when the form is opened.

Pascal

```
property ConnectMode: TDataMode;
```

C++

```
__property TDataMode ConnectMode;
```

Description

Use ConnectMode to change the way a DDE conversation is established. If ConnectMode is ddeAutomatic, the link is

established when it is defined by SetLink (see page 10872). If ConnectMode is ddeManual, the application must call the OpenLink (see page 10869) method or the PasteLink (see page 10869) method to establish the DDE conversation.

See Also

OpenLink (see page 10869)

PasteLink (see page 10869)

SetLink (see page 10872)

58.1.1.5.2 TDdeClientConv.Conv Property

Indicates the handle for the current DDE conversation.

Pascal

```
property Conv: HConv;
```

C++

```
__property HConv Conv;
```

Description

Read Conv to get a handle to the DDE conversation when it is required for a Windows API function call. Conv is available once a DDE conversation has been established by OpenLink (see page 10869) or PasteLink (see page 10869).

See Also

OpenLink (see page 10869)

PasteLink (see page 10869)

58.1.1.5.3 TDdeClientConv.DataMode Property

Governs whether portions of the DDE conversation execute automatically.

Pascal

```
property DataMode: TDataMode;
```

C++

```
__property TDataMode DataMode;
```

Description

DataMode is not implemented for TDdeClientConv (see page 10858). Derived objects can use this property to automatically execute portions of a DDE conversation, such as a RequestData (see page 10871) or PokeData (see page 10870) call.

Note: To determine the way the DDE conversation is established, use ConnectMode (see page 10873) instead.

See Also

ConnectMode (see page 10873)

PokeData (see page 10870)

RequestData (see page 10871)

58.1.1.5.4 TDdeClientConv.DdeFmt Property

Represents the clipboard format used for submitting or requesting a data item.

Pascal

```
property DdeFmt: Integer;
```

C++

```
__property int DdeFmt;
```

Description

Use DdeFmt to specify the format of DDE data items when it has not been specified by the server.

See Also

ServiceApplication (see page 10876)

TDdeClientItem (see page 10878)

58.1.1.5.5 TDdeClientConv.DdeService Property

Specifies the DDE server application to be linked to a DDE client.

Pascal

```
property DdeService: string;
```

C++

```
__property AnsiString DdeService;
```

Description

Use DdeService to specify the server application to link to. Typically, DdeService is the file name (with path, if necessary) of the DDE server application's main executable file without the .EXE extension. If the DDE server is a CLX application, DdeService is the project name without the .DPR or .EXE extension. For example, to link to a TDdeServerConv (see page 10891) component in PROJ1.DPR, set DdeService to PROJ1.

At design time, specify DdeService by typing the DDE server application name in the object inspector or by choosing Paste Link in the DDE Info dialog box. To establish a link at runtime, specify the service and topic with the SetLink (see page 10872) method.

Note: The necessary value for DdeService depends on the individual server application. See the documentation of the DDE server application for specific information about how to specify the server application.

See Also

DdeTopic (see page 10875)

ServiceApplication (see page 10876)

SetLink (see page 10872)

TDdeServerConv (see page 10891)

58.1.1.5.6 TDdeClientConv.DdeTopic Property

Specifies the topic of a DDE conversation

Pascal

```
property DdeTopic: string;
```

C++

```
__property AnsiString DdeTopic;
```

Description

Use DdeTopic to indicate the server-defined topic name of the DDE conversation. Typically, DdeTopic is a file name (with path, if necessary) used by the server application. If the DDE client is linked to a TDdeServerConv (see page 10891) component, DdeTopic is the name of the server conversation component rather than the form caption.

At design time, specify DdeTopic either by typing the DdeTopic name in the object inspector or by choosing Paste Link in the DDE Info dialog box. To establish a link at runtime, specify the service and topic with the SetLink (see page 10872) method.

Note: The server application defines the topic names available for a DDE conversation. See the documentation for the DDE server application for specific information about the names of available topics.

See Also

DdeService (see page 10875)

SetLink (see page 10872)

TDdeServerConv (see page 10891)

58.1.1.5.7 TDdeClientConv.FormatChars Property

Determines if certain characters are filtered out of text data transferred from a DDE server application.

Pascal

```
property FormatChars: Boolean;
```

C++

```
__property Boolean FormatChars;
```

Description

Use FormatChars with DDE server applications that transfer backspaces, linefeeds, carriage returns, and tabs with the text data. These formatting characters can cause incorrect spacing, line breaks, or characters in the DDE client data. To prevent the formatting characters from interfering with the formatting of the client data, set FormatChars to false. To include the formatting characters sent by the DDE server, set FormatChars to true.

See Also

RequestData (see page 10871)

58.1.1.5.8 TDdeClientConv.ServiceApplication Property

Specifies the main executable file name (with path, if necessary) of a DDE server application, without the .EXE extension.

Pascal

```
property ServiceApplication: string;
```

C++

```
__property AnsiString ServiceApplication;
```

Description

Set ServiceApplication to the name of the server application. Typically, this is the same value as the DdeService (see page 10875) property. Sometimes, however, DdeService (see page 10875) is a value other than the DDE server application's executable file name.

If the server application is not running when the TDdeClientConv (see page 10858) tries to establish a DDE conversation, OpenLink (see page 10869) uses the ServiceApplication property to launch the server.

See Also

DdeService (see page 10875)

OpenLink (see page 10869)

58.1.1.5.9 TDdeClientConv.WaitStat Property

Indicates whether the server is in the process of servicing a DDE macro that prohibits asynchronous DDE transactions.

Pascal

```
property WaitStat: Boolean;
```

C++

```
__property Boolean WaitStat;
```

Description

Read WaitStat to learn whether a DDE transaction can be started. When WaitStat is true, the server is processing the last macro, and attempts to execute a macro, request data, or poke data will fail. When WaitStat is false, the server is free to execute the next transaction in the DDE conversation.

See Also

ExecuteMacro (see page 10865)

ExecuteMacroLines (see page 10866)

PokeData (see page 10870)

PokeDataLines (see page 10870)

RequestData (see page 10871)

58.1.1.6 TDdeClientConv Events

The events of the TDdeClientConv class are listed here.

Published Events

	Name	Description
	OnClose (see page 10877)	Occurs when the DDE conversation terminates.
	OnOpen (see page 10878)	Occurs immediately after the DDE link to the server has been established.

58.1.1.6.1 TDdeClientConv.OnClose Event

Occurs when the DDE conversation terminates.

Pascal

```
property OnClose: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClose;
```

Description

Write an OnClose event handler to take specific action when the DDE conversation ends. The DDE conversation will end if the server terminates the conversation, if the link is closed or changed, or if the TDdeClientConv (see page 10858) object is destroyed.

See Also

CloseLink (see page 10864)

Destroy (see page 10862)

SetLink (see page 10872)

58.1.1.6.2 TDdeClientConv.OnOpen Event

Occurs immediately after the DDE link to the server has been established.

Pascal

```
property OnOpen: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnOpen;
```

Description

Write an OnOpen event handler to take specific action whenever a link is established to the DDE server. The DDE link is established by the OnOpen method or the PasteLink (see page 10869) method. SetLink (see page 10872) will also establish a DDE link if the ConnectMode (see page 10873) is ddeAutomatic. OnOpen will not be called if the link fails, such as when the server application can't be launched.

See Also

ConnectMode (see page 10873)

OpenLink (see page 10869)

PasteLink (see page 10869)

SetLink (see page 10872)

58.1.2 TDdeClientItem Class

TDDEClientItem defines an item in a DDE conversation.

Class Hierarchy

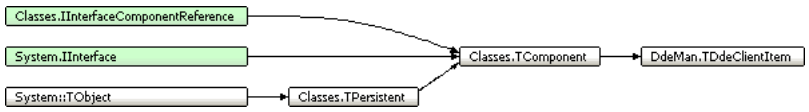

Pascal

```
TDdeClientItem = class(TComponent);
```

C++

```
class TDdeClientItem : public TComponent;
```

File

DdeMan

Description

Use TDDEClientItem with TDDEClientConv to enable an application to act as a DDE client. A DDE client item specifies a single

item supported by a DDE server. A single TDDEClientConv object may be associated with multiple client items. If the DDE client conversation has established a link with a DDE server, the server will automatically and continually update all client items until the conversation is terminated.

See Also

TDdeClientConv (see page 10858)

TDdeServerItem (see page 10898)

58.1.2.1 TDdeClientItem Members

The following tables list the members exposed by TDdeClientItem.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 10882)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 10881)	Creates an instance of TDDEClientItem.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 10882)	Destroys an instance of TDDEClientItem.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 10885)	Occurs immediately after the DDE server updates the Lines (see page 10884) property.

Published Properties

	Name	Description
	DdeConv (see page 10883)	Specifies the DDE client conversation to associate with the DDE client item.
	DdeItem (see page 10883)	Specifies the item of a DDE conversation.
	Lines (see page 10884)	Contains the text data to exchange in a DDE conversation.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Text (see page 10884)	Contains the text data to exchange in a DDE conversation.

58.1.2.2 TDdeClientItem.Create Constructor

Creates an instance of TDDEClientItem.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDdeClientItem(TComponent * AOwner);
```

Description

Call Create to instantiate a TDDEClientItem at runtime. After calling the inherited constructor, Create creates a helper object to manage the Lines (see page 10884) property.

See Also

Lines (see page 10884)

Create (see page 2766)

58.1.2.3 TDdeClientItem.Destroy Destructor

Destroys an instance of TDDEClientItem.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDdeClientItem();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TDDEClientItem is not nil, and only then calls Destroy.

Destroy frees the memory associated with the Lines (see page 10884) property before calling the inherited destructor.

See Also

Free (see page 22300)

Lines (see page 10884)

58.1.2.4 TDdeClientItem Methods

The methods of the TDdeClientItem class are listed here.

Protected Methods

	Name	Description
	Notification (see page 10882)	Forwards notification messages to all owned components.

58.1.2.4.1 TDdeClientItem.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

58.1.2.5 TDdeClientItem Properties

The properties of the TDdeClientItem class are listed here.

Published Properties

	Name	Description
	DdeConv (see page 10883)	Specifies the DDE client conversation to associate with the DDE client item.
	DdeItem (see page 10883)	Specifies the item of a DDE conversation.
	Lines (see page 10884)	Contains the text data to exchange in a DDE conversation.
	Text (see page 10884)	Contains the text data to exchange in a DDE conversation.

58.1.2.5.1 TDdeClientItem.DdeConv Property

Specifies the DDE client conversation to associate with the DDE client item.

Pascal

```
property DdeConv: TDdeClientConv;
```

C++

```
__property TDdeClientConv DdeConv;
```

Description

Set DdeConv to associate the DDE client item with a DDE client conversation. The value of DdeConv is the DDE client conversation object that defines the DDE conversation. The DdeConv establishes the link to the server that provides the Text (see page 10884) for the DDE client item.

See Also

Lines (see page 10884)

TDDeClientConv

Text (see page 10884)

58.1.2.5.2 TDdeClientItem.DdeItem Property

Specifies the item of a DDE conversation.

Pascal

```
property DdeItem: string;
```

C++

```
__property AnsiString DdeItem;
```

Description

Set the DdeItem property to the name of a DDE item supported by the DDE server. The value of DdeItem depends on the linked DDE server application. DdeItem is typically a selectable portion of text, such as a spreadsheet cell or a database field in an edit box. If the DDE server is a Delphi C++Builder application, DdeItem is the name of the linked DDE server item.

At design time, specify DdeItem by typing the item string in the object inspector. Alternately, DdeItem can be specified from the DDE conversation by pasting a link using the DDE Info dialog box, which is called by clicking the ellipsis (...) button for DdeService or DdeTopic in the object inspector. After choosing Paste Link in the DDE Info dialog box, choose the item from a list of possible items for DdeItem in the object inspector when link information is on the Clipboard.

Note: See the documentation for the DDE server application for the specific information about specifying DDE items.

See Also

DdeService (see page 10875)

DdeTopic (see page 10875)

TDdeServerItem (see page 10898)

58.1.2.5.3 TDdeClientItem.Lines Property

Contains the text data to exchange in a DDE conversation.

Pascal

```
property Lines: TStrings;
```

C++

```
__property TStrings Lines;
```

Description

Read Lines to get the text that is updated by the DDE server application. Lines is updated automatically by the DDE server. When Lines is changed by the server, an OnChange (see page 10885) event occurs.

Set the Lines property to poke data to the DDE server.

Lines corresponds to the Text (see page 10884) property. Whenever the value of Lines or Text (see page 10884) is updated, the other is also changed so that the first line of Lines is always equal to Text (see page 10884).

Note: Lines of data can also be sent to the DDE server by calling the PokeDataLines method of the DDE client conversation.

See Also

PokeDataLines (see page 10870)

Text (see page 10884)

58.1.2.5.4 TDdeClientItem.Text Property

Contains the text data to exchange in a DDE conversation.

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

Read Text to get the text that is updated by the DDE server application. Text is updated automatically by the DDE server. When Text is changed by the server, an OnChange (see page 10885) event occurs.

Set the Text property to poke data to the DDE server.

Text corresponds to the Lines (see page 10884) property. Whenever the value of Lines (see page 10884) or Text is updated, the other is also changed so that the first line of Lines (see page 10884) is always equal to Text.

For text data longer than the 255 character limit of Text, use the Lines (see page 10884) property.

Note: Data can also be sent to the DDE server by calling the PokeData method of the DDE client conversation.

See Also

Lines (see page 10884)

PokeData (see page 10870)

58.1.2.6 TDdeClientItem Events

The events of the TDdeClientItem class are listed here.

Published Events

	Name	Description
	OnChange (see page 10885)	Occurs immediately after the DDE server updates the Lines (see page 10884) property.

58.1.2.6.1 TDdeClientItem.OnChange Event

Occurs immediately after the DDE server updates the Lines (see page 10884) property.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action when the DDE server updates the text for this DDE client item. The new value for the DDE client item can be found by reading the Lines (see page 10884) or Text (see page 10884) property.

See Also

Lines (see page 10884)

Text (see page 10884)

58.1.3 TDdeMgr Class

Class Hierarchy

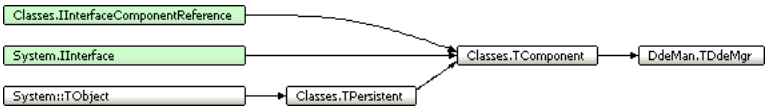

Pascal

```
TDdeMgr = class(TComponent);
```

C++

```
class TDdeMgr : public TComponent;
```

File

DdeMan

Description

This is class DdeMan.TDdeMgr.

58.1.3.1 TDdeMgr Members

The following tables list the members exposed by TDdeMgr.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.

	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 10889)	Allocates memory and constructs a safely initialized instance of a component.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 10889)	Disposes of the component and its owned components.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetExeName (see page 10890)	This is GetExeName, a member of class TDdeMgr.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	AppName (see page 10891)	This is AppName, a member of class TDdeMgr.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DdeInstId (see page 10891)	This is DdeInstId, a member of class TDdeMgr.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	LinkClipFmt (see page 10891)	This is LinkClipFmt, a member of class TDdeMgr.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

58.1.3.2 TDdeMgr.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDdeMgr(TComponent * AOwner);
```

Description

All objects have a Create method that constructs the object. TComponent redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner, as indicated by the AOwner parameter

sets the ComponentStyle property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components (see page 2799)

ComponentStyle (see page 2805)

Create (see page 22287)

Destroy (see page 2766)

Free (see page 22300)

Owner (see page 2807)

58.1.3.3 TDdeMgr.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDdeMgr();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

Destroy (see page 22287)

DestroyComponents (see page 2771)

Destroying (see page 2772)

Free (see page 22300)

Notification (see page 2783)

Owner (see page 2807)

Release (see page 13885)

Components (see page 2799)

58.1.3.4 TDdeMgr Methods

The methods of the TDdeMgr class are listed here.

Public Methods

	Name	Description
	GetExeName (see page 10890)	This is GetExeName, a member of class TDdeMgr.

58.1.3.4.1 TDdeMgr.GetExeName Method

Pascal

```
function GetExeName: string;
```

C++

```
__fastcall AnsiString GetExeName();
```

Description

This is GetExeName, a member of class TDdeMgr.

58.1.3.5 TDdeMgr Properties

The properties of the TDdeMgr class are listed here.

Public Properties

	Name	Description
	AppName (see page 10891)	This is AppName, a member of class TDdeMgr.
	DdeInstId (see page 10891)	This is DdeInstId, a member of class TDdeMgr.
R	LinkClipFmt (see page 10891)	This is LinkClipFmt, a member of class TDdeMgr.

58.1.3.5.1 TDdeMgr.AppName Property

Pascal

```
property AppName: string;
```

C++

```
__property AnsiString AppName;
```

Description

This is AppName, a member of class TDdeMgr.

58.1.3.5.2 TDdeMgr.DdeInstId Property

Pascal

```
property DdeInstId: LongInt;
```

C++

```
__property LongInt DdeInstId;
```

Description

This is DdeInstId, a member of class TDdeMgr.

58.1.3.5.3 TDdeMgr.LinkClipFmt Property

Pascal

```
property LinkClipFmt: Word;
```

C++

```
__property Word LinkClipFmt;
```

Description

This is LinkClipFmt, a member of class TDdeMgr.

58.1.4 TDdeServerConv Class

TDdeServerConv represents a DDE topic of conversation with a DDE client.

Class Hierarchy

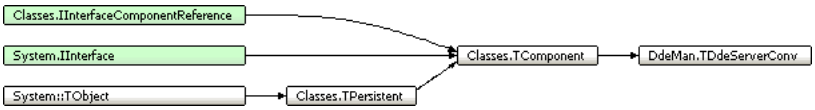

Pascal

```
TDdeServerConv = class(TComponent);
```

C++

```
class TDdeServerConv : public TComponent;
```

File

DdeMan

Description

Use TDdeServerConv to establish a DDE topic with a DDE client application. Individual items under the DDE topic are represented by TDdeServerItem (see page 10898) objects. These items are linked to the DDE server conversation by setting their ServerConv property.

Using a TDdeServerConv component is optional. Without a TDdeServerConv object, the client can still enter into a DDE conversation and receive an update directly from a TDdeServerItem (see page 10898) object. However, a DDE server conversation is required to handle any macros that the DDE client may send.

The Name property of the DDE server conversation provides the topic used by DDE clients to establish a link to the DDE server conversation. DDE server items that are not associated with a DDE server conversation use the Caption property of the form for the topic name. Use a TDdeServerConv object when establishing multiple DDE topics in a single form, or when the Caption of the form may not be unique or constant at runtime.

See Also

TDdeClientConv (see page 10858)

TDdeClientItem (see page 10878)

TDdeServerItem (see page 10898)

TForm (see page 13992)

58.1.4.1 TDdeServerConv Members

The following tables list the members exposed by TDdeServerConv.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Connect (see page 10896)	This is Connect, a member of class TDdeServerConv.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	Disconnect (see page 10896)	This is Disconnect, a member of class TDdeServerConv.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.

	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 10895)	Creates an instance of TDdeServerConv (see page 10891).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 10895)	Destroys an instance of TDdeServerConv (see page 10891).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.

V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteMacro (see page 10896)	Responds to macros sent by the DDE client.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnClose (see page 10897)	Occurs when the DDE client terminates the DDE conversation.
	OnExecuteMacro (see page 10897)	Occurs when the DDE client sends a macro to the DDE server conversation.
	OnOpen (see page 10898)	Occurs immediately after the DDE link to the client has been established.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

58.1.4.2 TDdeServerConv.Create Constructor

Creates an instance of TDdeServerConv (see page 10891).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDdeServerConv(TComponent * AOwner);
```

Description

Call Create to create an instance of TDdeServerConv (see page 10891) at runtime. For TDdeServerConv (see page 10891) objects placed on forms at design time, Create is called automatically.

See Also

Create (see page 2766)

58.1.4.3 TDdeServerConv.Destroy Destructor

Destroys an instance of TDdeServerConv (see page 10891).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDdeServerConv();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TDdeServerConv (see page 10891) reference is not nil and only then calls Destroy.

Destroy removes the DDE server conversation topic from the application, but does not terminate any currently open DDE conversations. Instead, the DDE server items that were associated with the DDE server conversation become associated with the form instead, and continue their link to the DDE client.

See Also

Free (see page 22300)

TDdeServerItem (see page 10898)

58.1.4.4 TDdeServerConv Methods

The methods of the TDdeServerConv class are listed here.

Protected Methods

	Name	Description
	Connect (see page 10896)	This is Connect, a member of class TDdeServerConv.
	Disconnect (see page 10896)	This is Disconnect, a member of class TDdeServerConv.

Public Methods

	Name	Description
	ExecuteMacro (see page 10896)	Responds to macros sent by the DDE client.

58.1.4.4.1 TDdeServerConv.Connect Method

Pascal

```
procedure Connect; dynamic;
```

C++

```
__fastcall Connect();
```

Description

This is Connect, a member of class TDdeServerConv.

58.1.4.4.2 TDdeServerConv.Disconnect Method

Pascal

```
procedure Disconnect; dynamic;
```

C++

```
__fastcall Disconnect();
```

Description

This is Disconnect, a member of class TDdeServerConv.

58.1.4.4.3 TDdeServerConv.ExecuteMacro Method

Responds to macros sent by the DDE client.

Pascal

```
function ExecuteMacro(Data: HDdeData): LongInt;
```

C++

```
__fastcall LongInt ExecuteMacro(HDdeData Data);
```

Description

ExecuteMacro provides the only way for a DDE server application to respond to macros sent by DDE clients. ExecuteMacro is called when the DDE client sends a Macro over the DDE link. Data is the handle to the DDE data that contains the macro.

ExecuteMacro retrieves the strings of the macro from Data, and calls the OnExecuteMacro (see page 10897) event handler. If there is no OnExecuteMacro (see page 10897) event handler, the macro is ignored. Whether or not there is an OnExecuteMacro (see page 10897) event handler, the DDE client always receives notification that the macro was successfully

transmitted, unless Data contains no macro.

See Also

OnExecuteMacro (see page 10897)

58.1.4.5 TDdeServerConv Events

The events of the TDdeServerConv class are listed here.

Published Events

	Name	Description
	OnClose (see page 10897)	Occurs when the DDE client terminates the DDE conversation.
	OnExecuteMacro (see page 10897)	Occurs when the DDE client sends a macro to the DDE server conversation.
	OnOpen (see page 10898)	Occurs immediately after the DDE link to the client has been established.

58.1.4.5.1 TDdeServerConv.OnClose Event

Occurs when the DDE client terminates the DDE conversation.

Pascal

```
property OnClose: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClose;
```

Description

Write an OnClose event handler to take specific action when the DDE conversation ends. The DDE conversation will end if the client terminates the conversation. It does not end if the TDdeServerConv (see page 10891) object is destroyed. In that case, the DDE server items simply remove themselves from the DDE server conversation, and continue the link to the DDE client.

The Sender parameter is the DDE server conversation, not the DDE client.

See Also

Destroy (see page 10895)

OnOpen (see page 10898)

58.1.4.5.2 TDdeServerConv.OnExecuteMacro Event

Occurs when the DDE client sends a macro to the DDE server conversation.

Pascal

```
property OnExecuteMacro: TMacroEvent;
```

C++

```
__property TMacroEvent OnExecuteMacro;
```

Description

Write an OnExecuteMacro event handler to implement a response to macros sent by the DDE client.

See Also

ExecuteMacro (see page 10896)

58.1.4.5.3 TDdeServerConv.OnOpen Event

Occurs immediately after the DDE link to the client has been established.

Pascal

```
property OnOpen: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnOpen;
```

Description

Write an OnOpen event handler to take specific action whenever a link is established to a DDE client. The Sender parameter is the DDE server conversation, not the DDE client.

See Also

OnClose (see page 10897)

58.1.5 TDdeServerItem Class

TDdeServerItem defines the item of a DDE conversation.

Class Hierarchy

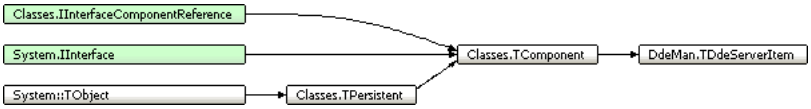

Pascal

```
TDdeServerItem = class(TComponent);
```

C++

```
class TDdeServerItem : public TComponent;
```

File

DdeMan

Description

Use TDdeServerItem by itself, or, optionally, with a TDdeServerConv (see page 10891) object to enable an application to function as a DDE server. DDE clients link to an individual DDE server item by referring to the Name property. Once a link to a DDE client is established, the DDE server item can update the client to changes in the item data, or receive data that the DDE client pokes to the item.

See Also

TDdeClientConv (see page 10858)

TDdeClientItem (see page 10878)

TDdeServerConv (see page 10891)

58.1.5.1 TDdeServerItem Members

The following tables list the members exposed by TDdeServerItem.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetText (see page 10904)	This is GetText, a member of class TDdeServerItem.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 10904)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetLines (see page 10905)	This is SetLines, a member of class TDdeServerItem.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetServerConv (see page 10905)	This is SetServerConv, a member of class TDdeServerItem.
	SetText (see page 10905)	This is SetText, a member of class TDdeServerItem.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	Change (see page 10903)	Generates an OnChange (see page 10908) event.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CopyToClipboard (see page 10903)	Copies the data specified in the Text (see page 10907) or Lines (see page 10906) property to the Windows Clipboard, along with DDE link information.
	Create (see page 10901)	Creates an instance of TDdeServerItem (see page 10898).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 10902)	Destroys an instance of TDdeServerItem (see page 10898).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PokeData (see page 10904)	Processes data that has been poked to the DDE server item by a DDE client.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Fmt (see page 10906)	Indicates the clipboard format the DDE client item uses to pass data to the DDE client.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnChange (see page 10908)	Occurs when the Lines (see page 10906) property changes, before the changes have been sent to the DDE client.
	OnPokeData (see page 10908)	Occurs after a DDE client changes the value of the Lines (see page 10906) property.

Published Properties

	Name	Description
	Lines (see page 10906)	Contains the data to exchange in a DDE conversation.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ServerConv (see page 10907)	Specifies the DDE server conversation to associate with the DDE server item.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Text (see page 10907)	Contains the data to exchange in a DDE conversation.

58.1.5.2 TDdeServerItem.Create Constructor

Creates an instance of TDdeServerItem (see page 10898).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDdeServerItem(TComponent * AOwner);
```

Description

Call Create to instantiate a DDE server item at runtime. TDdeServerItem (see page 10898) objects placed on forms at design time are created automatically.

After calling the inherited constructor, Create initializes the Fmt (see page 10906) property to CF_TEXT, and creates a helper object to manage the Lines (see page 10906) property.

See Also

Fmt (see page 10906)

Lines (see page 10906)

Create (see page 2766)

58.1.5.3 TDdeServerItem.Destroy Destructor

Destroys an instance of TDdeServerItem (see page 10898).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDdeServerItem();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TDdeServerItem (see page 10898) is not nil, and only then calls Destroy.

Destroy frees the memory associated with the Lines (see page 10906) property before calling the inherited destructor.

See Also

Lines (see page 10906)

Free (see page 22300)

58.1.5.4 TDdeServerItem Methods

The methods of the TDdeServerItem class are listed here.

Protected Methods

	Name	Description
	GetText (see page 10904)	This is GetText, a member of class TDdeServerItem.
V	Notification (see page 10904)	Forwards notification messages to all owned components.
	SetLines (see page 10905)	This is SetLines, a member of class TDdeServerItem.
	SetServerConv (see page 10905)	This is SetServerConv, a member of class TDdeServerItem.
	SetText (see page 10905)	This is SetText, a member of class TDdeServerItem.

Public Methods

	Name	Description
	Change (see page 10903)	Generates an OnChange (see page 10908) event.

	CopyToClipboard (see page 10903)	Copies the data specified in the Text (see page 10907) or Lines (see page 10906) property to the Windows Clipboard, along with DDE link information.
	PokeData (see page 10904)	Processes data that has been poked to the DDE server item by a DDE client.

58.1.5.4.1 TDdeServerItem.Change Method

Generates an OnChange (see page 10908) event.

Pascal

```
procedure Change; dynamic;
```

C++

```
__fastcall Change();
```

Description

Call Change to trigger an OnChange (see page 10908) event when not setting the Lines (see page 10906) or Text (see page 10907) property. Use Change in conjunction with the OnChange (see page 10908) event to enable the DDE server item to exchange data that is not compatible with the strings used by the Lines (see page 10906) and Text (see page 10907) properties.

See Also

Lines (see page 10906)

OnChange (see page 10908)

Text (see page 10907)

58.1.5.4.2 TDdeServerItem.CopyToClipboard Method

Copies the data specified in the Text (see page 10907) or Lines (see page 10906) property to the Windows Clipboard, along with DDE link information.

Pascal

```
procedure CopyToClipboard;
```

C++

```
__fastcall CopyToClipboard();
```

Description

Use CopyToClipboard to enable DDE clients to link to the DDE server item by executing an Edit | Paste Link command, or its equivalent in the command structure of the DDE client application.

CopyToClipboard can be used to create a DDE link at runtime only. To create a link at design time, select the TDdeServerItem (see page 10898) object and choose Edit | Copy from the menu. Then, activate the DDE client application and paste the link according to the rules of the DDE client application. See the documentation for the DDE client application for specific information about pasting the link.

See Also

Lines (see page 10906)

PasteLink (see page 10869)

Text (see page 10907)

58.1.5.4.3 TDdeServerItem.GetText Method

Pascal

```
function GetText: string;
```

C++

```
__fastcall AnsiString GetText();
```

Description

This is GetText, a member of class TDdeServerItem.

58.1.5.4.4 TDdeServerItem.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

58.1.5.4.5 TDdeServerItem.PokeData Method

Processes data that has been poked to the DDE server item by a DDE client.

Pascal

```
function PokeData(Data: HDdeData): LongInt;
```

C++

```
__fastcall LongInt PokeData(HDdeData Data);
```

Description

Call PokeData to inform the DDE server item that the DDE conversation holds data that has been poked from a DDE client. Data is the DDE handle to the poked data.

PokeData does the following:

1Uses the Data handle to obtain the data that was poked.

2Sets the Lines (see page 10906) property to the new data value.

3Triggers an OnChange (see page 10908) event.

4Sends the new value of the data back to the DDE client.

5Triggers an OnPokeData (see page 10908) event.

See Also

Lines (see page 10906)

OnChange (see page 10908)

OnPokeData (see page 10908)

58.1.5.4.6 TDdeServerItem.SetLines Method

Pascal

```
procedure SetLines(Value: TStrings);
```

C++

```
__fastcall SetLines(TStrings Value);
```

Description

This is SetLines, a member of class TDdeServerItem.

58.1.5.4.7 TDdeServerItem.SetServerConv Method

Pascal

```
procedure SetServerConv(SConv: TDdeServerConv);
```

C++

```
__fastcall SetServerConv(TDdeServerConv SConv);
```

Description

This is SetServerConv, a member of class TDdeServerItem.

58.1.5.4.8 TDdeServerItem.SetText Method

Pascal

```
procedure SetText(const Item: string);
```

C++

```
__fastcall SetText(const AnsiString Item);
```

Description

This is SetText, a member of class TDdeServerItem.

58.1.5.5 TDdeServerItem Properties

The properties of the TDdeServerItem class are listed here.

Public Properties

	Name	Description
R	Fmt (see page 10906)	Indicates the clipboard format the DDE client item uses to pass data to the DDE client.

Published Properties

	Name	Description
	Lines (see page 10906)	Contains the data to exchange in a DDE conversation.
	ServerConv (see page 10907)	Specifies the DDE server conversation to associate with the DDE server item.
	Text (see page 10907)	Contains the data to exchange in a DDE conversation.

58.1.5.5.1 TDdeServerItem.Fmt Property

Indicates the clipboard format the DDE client item uses to pass data to the DDE client.

Pascal

```
property Fmt: Integer;
```

C++

```
__property int Fmt;
```

Description

Set Fmt in an OnChange (see page 10908) event handler to specify a clipboard format for the data that is passed to the DDE client if it is not text. The DDE data that is sent to the DDE client includes this format so that clients can correctly interpret the data. Read Fmt when the client pokes data to the DDE server item to determine the clipboard format the client used to format the data.

See Also

Lines (see page 10906)

OnChange (see page 10908)

OnPokeData (see page 10908)

DdeFmt (see page 10874)

Text (see page 10907)

58.1.5.5.2 TDdeServerItem.Lines Property

Contains the data to exchange in a DDE conversation.

Pascal

```
property Lines: TStrings;
```

C++

```
__property TStrings Lines;
```

Description

Set Lines to the data the DDE server item sends to the DDE client. Whenever the Lines property changes, an OnChange (see

page 10908) event occurs and then the new value is sent to the DDE client. If the DDE server item accepts poked data, Lines receives the data whenever the DDE client pokes data to this item. When Lines is changed by a DDE client poking data, an OnPokeData (see page 10908) event occurs after the OnChange (see page 10908) event.

Usually Lines contains text, and setting Lines sets the Fmt (see page 10906) property to CF_TEXT. However, Delphi C++Builder strings can contain non-text data. If Lines does not specify text data, Fmt (see page 10906) must be changed to indicate the format of the data in an OnChange (see page 10908) event handler.

Lines corresponds to the Text (see page 10907) property. Whenever the value of Lines or Text (see page 10907) is changed, the other is updated so that the first line of Lines is always equal to Text (see page 10907).

See Also

Fmt (see page 10906)

OnChange (see page 10908)

OnPokeData (see page 10908)

Text (see page 10907)

58.1.5.5.3 TDdeServerItem.ServerConv Property

Specifies the DDE server conversation to associate with the DDE server item.

Pascal

```
property ServerConv: TDdeServerConv;
```

C++

```
__property TDdeServerConv ServerConv;
```

Description

Set ServerConv to associate the DDE server item with a DDE topic represented by the DDE server conversation. If ServerConv is not set, the DDE topic of the conversation is the value of the Caption property of the form containing the TDdeServerItem (see page 10898) component.

58.1.5.5.4 TDdeServerItem.Text Property

Contains the data to exchange in a DDE conversation.

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

Set Text to the data the DDE server item sends to the DDE client. Whenever the Text property changes, an OnChange (see page 10908) event occurs and then the new value is sent to the DDE client. If the DDE server item accepts poked data, Text receives the data whenever the DDE client pokes data to this item. When Text is changed by a DDE client poking data, an OnPokeData (see page 10908) event occurs after the OnChange (see page 10908) event.

Usually Text contains text, and setting Text sets the Fmt (see page 10906) property to CF_TEXT. However, Text can contain non-text data. If Text does not specify text data, Fmt (see page 10906) must be changed to indicate the format of the data in an OnChange (see page 10908) event handler.

Text corresponds to the Lines (see page 10906) property. Whenever the value of Lines (see page 10906) or Text is

changed, the other is updated so that the first line of Lines (see page 10906) is always equal to Text.

See Also

Fmt (see page 10906)

Lines (see page 10906)

OnChange (see page 10908)

OnPokeData (see page 10908)

58.1.5.6 TDdeServerItem Events

The events of the TDdeServerItem class are listed here.

Published Events

	Name	Description
	OnChange (see page 10908)	Occurs when the Lines (see page 10906) property changes, before the changes have been sent to the DDE client.
	OnPokeData (see page 10908)	Occurs after a DDE client changes the value of the Lines (see page 10906) property.

58.1.5.6.1 TDdeServerItem.OnChange Event

Occurs when the Lines (see page 10906) property changes, before the changes have been sent to the DDE client.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action when the DDE server item is about to update the DDE client. The new value for the DDE client item can be found by reading the Lines (see page 10906) or Text (see page 10907) property.

If the value of Lines (see page 10906) does not contain text data, an OnChange event handler must be assigned to change the value of Fmt (see page 10906). OnChange is called after Fmt (see page 10906) gets set to CF_TEXT by setting the Lines (see page 10906) property, but before the data gets sent to the client.

See Also

Change (see page 10903)

Fmt (see page 10906)

Lines (see page 10906)

Text (see page 10907)

58.1.5.6.2 TDdeServerItem.OnPokeData Event

Occurs after a DDE client changes the value of the Lines (see page 10906) property.

Pascal

```
property OnPokeData: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPokeData;
```

Description

Write an OnPokeData event handler to take specific action when the DDE client pokes data to this DDE server item. The new value for the item can be found by reading the Lines (see page 10906) or Text (see page 10907) property.

When a DDE client pokes data to a DDE server items, the PokeData (see page 10904) method is called. PokeData (see page 10904) causes the following events.

1First, the new value is assigned to the Lines (see page 10906) (Text (see page 10907)) property.

2An OnChange (see page 10908) event occurs.

3The new value is sent back to the DDE client.

4An OnPokeData event occurs.

See Also

Lines (see page 10906)

OnChange (see page 10908)

PokeData (see page 10904)

Text (see page 10907)

58.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetPasteLinkInfo (see page 10910)	This is function DdeMan.GetPasteLinkInfo.

58.2.1 DdeMan.GetPasteLinkInfo Function

Pascal

```
function GetPasteLinkInfo(var Service: string; var Topic: string; var Item: string): Boolean;
```

C++

```
Boolean GetPasteLinkInfo(AnsiString Service, AnsiString Topic, AnsiString Item);
```

File

DdeMan

Description

This is function DdeMan.GetPasteLinkInfo.

58.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TDataMode (see page 10911)	TDataMode indicates whether a stage in a DDE conversation happens automatically.

58.3.1 DdeMan.TDataMode Enumeration

TDataMode indicates whether a stage in a DDE conversation happens automatically.

Pascal

```
TDataMode = (
  ddeAutomatic,
  ddeManual
);
```

C++

```
enum TDataMode {
  ddeAutomatic,
  ddeManual
};
```

File

DdeMan

Description

The following table lists the possible values:

Value	Meaning
ddeAutomatic	The stage in the DDE conversation is initiated automatically.
axrIncludeDescendants	The application must call a method to initiate the stage in the DDE converstation.

58.4 Types

The following table lists types in this documentation.

Types

Name	Description
TMacroEvent (see page 10912)	TMacroEvent is the type for event handlers that implement the response of a DDE server to a macro.

58.4.1 DdeMan.TMacroEvent Type

TMacroEvent is the type for event handlers that implement the response of a DDE server to a macro.

Pascal

```
TMacroEvent = procedure (Sender: TObject; Msg: TStrings) of object;
```

C++

```
(Sender: TObject; Msg: TStrings) ( TMacroEvent)();
```

File

DdeMan

Description

Sender parameter is the DDE server conversation.

Msg gives the macro string for the server to execute.

58.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
ddeMgr (see page 10913)	This is variable DdeMan.ddeMgr.

58.5.1 DdeMan.ddeMgr Variable

Pascal

```
ddeMgr: TDdeMgr;
```

C++

```
TDdeMgr ddeMgr;
```

File

DdeMan

Description

This is variable DdeMan.ddeMgr.

59 DesignEditors Namespace

This is namespace DesignEditors.

Topics

Name	Description
Classes (see page 10917)	The following table lists classes in this documentation.
Interfaces (see page 11088)	The following table lists interfaces in this documentation.
Functions (see page 11090)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 11095)	The following table lists structs, records, enums in this documentation.
Types (see page 11096)	The following table lists types in this documentation.
Variables (see page 11098)	The following table lists variables in this documentation.
Constants (see page 11099)	The following table lists constants in this documentation.

Classes

	Name	Description
	TBoolProperty (see page 10918)	This is class DesignEditors.TBoolProperty.
	TCharProperty (see page 10918)	This is class DesignEditors.TCharProperty.
	TClassProperty (see page 10922)	This is class DesignEditors.TClassProperty.
	TComponentEditor (see page 10927)	TComponentEditor provides the default implementation for the IComponentEditor interface.
	TComponentGuidelines (see page 10935)	This is class DesignEditors.TComponentGuidelines.
	TComponentNameProperty (see page 10939)	This is class DesignEditors.TComponentNameProperty.
	TComponentProperty (see page 10944)	This is class DesignEditors.TComponentProperty.
	TCustomModule (see page 10952)	This is class DesignEditors.TCustomModule.
	TDateProperty (see page 10957)	This is class DesignEditors.TDateProperty.
	TDateTimeProperty (see page 10962)	This is class DesignEditors.TDateTimeProperty.
	TDefaultEditor (see page 10967)	TDefaultEditor is the component editor for components that do not explicitly register a component editor.
	TDesignerOleDropTarget (see page 10970)	This is class DesignEditors.TDesignerOleDropTarget.
	TEditActionSelectionEditor (see page 10973)	This is class DesignEditors.TEditActionSelectionEditor.
	TEnumProperty (see page 10978)	This is class DesignEditors.TEnumProperty.
	TFloatProperty (see page 10983)	This is class DesignEditors.TFloatProperty.
	TInt64Property (see page 10988)	This is class DesignEditors.TInt64Property.
	TIntegerProperty (see page 10993)	This is class DesignEditors.TIntegerProperty.
	TInterfaceProperty (see page 10998)	This is class DesignEditors.TInterfaceProperty.
	TMethodProperty (see page 11004)	This is class DesignEditors.TMethodProperty.

	TNestedProperty (see page 11011)	This is class DesignEditors.TNestedProperty.
	TOrdinalProperty (see page 11015)	This is class DesignEditors.TOrdinalProperty.
	TPropertyEditor (see page 11019)	TPropertyEditor is a base class for property editors that implements the IProperty interface.
	TSelectionEditor (see page 11046)	This is class DesignEditors.TSelectionEditor.
	TSetElementProperty (see page 11049)	This is class DesignEditors.TSetElementProperty.
	TSetProperty (see page 11056)	This is class DesignEditors.TSetProperty.
	TStringProperty (see page 11061)	This is class DesignEditors.TStringProperty.
	TTimeProperty (see page 11066)	This is class DesignEditors.TTimeProperty.
	TVariantProperty (see page 11071)	This is class DesignEditors.TVariantProperty.
	TWideCharProperty (see page 11077)	This is class DesignEditors.TWideCharProperty.
	TWideStringProperty (see page 11082)	This is class DesignEditors.TWideStringProperty.

Constants

Name	Description
sClassNotApplicable (see page 11099)	This is constant DesignEditors.sClassNotApplicable.
sNotAvailable (see page 11099)	This is constant DesignEditors.sNotAvailable.

Functions

	Name	Description
	AncestorNameMatches (see page 11090)	This is function DesignEditors.AncestorNameMatches.
	ClassInheritsFrom (see page 11090)	This is function DesignEditors.ClassInheritsFrom.
	FreeEditorGroup (see page 11091)	This is function DesignEditors.FreeEditorGroup.
	GetComponentEditor (see page 11091)	This is function DesignEditors.GetComponentEditor.
	GetComponentProperties (see page 11091)	This is function DesignEditors.GetComponentProperties.
	GetSelectionEditors (see page 11092)	This is function DesignEditors.GetSelectionEditors.
	GetSelectionEditors (see page 11092)	This is function DesignEditors.GetSelectionEditors.
	GetSelectionEditors (see page 11092)	This is function DesignEditors.GetSelectionEditors.
	GuidelinesFor (see page 11093)	This is function DesignEditors.GuidelinesFor.
	NewEditorGroup (see page 11093)	This is function DesignEditors.NewEditorGroup.
	NotifyGroupChange (see page 11093)	This is function DesignEditors.NotifyGroupChange.
	PossibleStream (see page 11094)	This is function DesignEditors.PossibleStream.
	UnnotifyGroupChange (see page 11094)	This is function DesignEditors.UnnotifyGroupChange.

Interfaces

	Name	Description
	IDesignGroupChange (see page 11088)	This is class DesignEditors.IDesignGroupChange.

Structs, Records, Enums

	Name	Description
	TInstProp (see page 11095)	This is record DesignEditors.TInstProp.

Types

Name	Description
PInstPropList (see page 11096)	This is type DesignEditors.PInstPropList.
TBit (see page 11096)	This is type DesignEditors.TBit.
TGetTopLevelComponentFunc (see page 11097)	This is type DesignEditors.TGetTopLevelComponentFunc.
TGroupChangeProc (see page 11097)	This is type DesignEditors.TGroupChangeProc.
TInstPropList (see page 11097)	This is type DesignEditors.TInstPropList.

Variables

Name	Description
GetTopLevelComponentFunc (see page 11098)	This is variable DesignEditors.GetTopLevelComponentFunc.

59.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TBoolProperty (see page 10918)	This is class DesignEditors.TBoolProperty.
	TCharProperty (see page 10918)	This is class DesignEditors.TCharProperty.
	TClassProperty (see page 10922)	This is class DesignEditors.TClassProperty.
	TComponentEditor (see page 10927)	TComponentEditor provides the default implementation for the IComponentEditor interface.
	TComponentGuidelines (see page 10935)	This is class DesignEditors.TComponentGuidelines.
	TComponentNameProperty (see page 10939)	This is class DesignEditors.TComponentNameProperty.
	TComponentProperty (see page 10944)	This is class DesignEditors.TComponentProperty.
	TCustomModule (see page 10952)	This is class DesignEditors.TCustomModule.
	TDateProperty (see page 10957)	This is class DesignEditors.TDateProperty.
	TDateTimeProperty (see page 10962)	This is class DesignEditors.TDateTimeProperty.
	TDefaultEditor (see page 10967)	TDefaultEditor is the component editor for components that do not explicitly register a component editor.
	TDesignerOleDropTarget (see page 10970)	This is class DesignEditors.TDesignerOleDropTarget.
	TEditActionSelectionEditor (see page 10973)	This is class DesignEditors.TEditActionSelectionEditor.
	TEnumProperty (see page 10978)	This is class DesignEditors.TEnumProperty.
	TFloatProperty (see page 10983)	This is class DesignEditors.TFloatProperty.
	TInt64Property (see page 10988)	This is class DesignEditors.TInt64Property.
	TIntegerProperty (see page 10993)	This is class DesignEditors.TIntegerProperty.
	TInterfaceProperty (see page 10998)	This is class DesignEditors.TInterfaceProperty.
	TMethodProperty (see page 11004)	This is class DesignEditors.TMethodProperty.
	TNestedProperty (see page 11011)	This is class DesignEditors.TNestedProperty.
	TOrdinalProperty (see page 11015)	This is class DesignEditors.TOrdinalProperty.
	TPropertyEditor (see page 11019)	TPropertyEditor is a base class for property editors that implements the IProperty interface.
	TSelectionEditor (see page 11046)	This is class DesignEditors.TSelectionEditor.
	TSetElementProperty (see page 11049)	This is class DesignEditors.TSetElementProperty.
	TSetProperty (see page 11056)	This is class DesignEditors.TSetProperty.
	TStringProperty (see page 11061)	This is class DesignEditors.TStringProperty.
	TTimeProperty (see page 11066)	This is class DesignEditors.TTimeProperty.
	TVariantProperty (see page 11071)	This is class DesignEditors.TVariantProperty.
	TWideCharProperty (see page 11077)	This is class DesignEditors.TWideCharProperty.
	TWideStringProperty (see page 11082)	This is class DesignEditors.TWideStringProperty.

59.1.1 TBoolProperty Class

Class Hierarchy

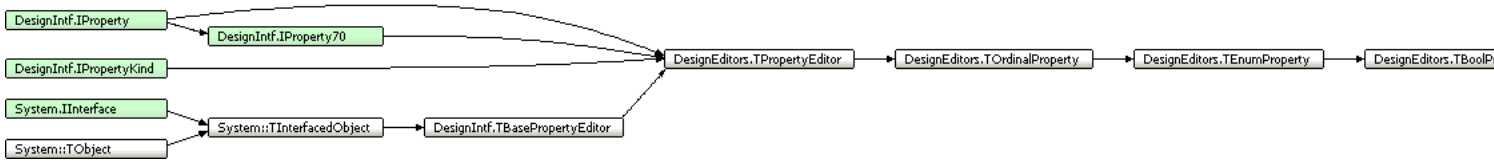

Pascal

```
TBoolProperty = class(TEnumProperty);
```

C++

```
class TBoolProperty : public TEnumProperty;
```

File

DesignEditors

Description

This is class DesignEditors.TBoolProperty.

59.1.2 TCharProperty Class

Class Hierarchy

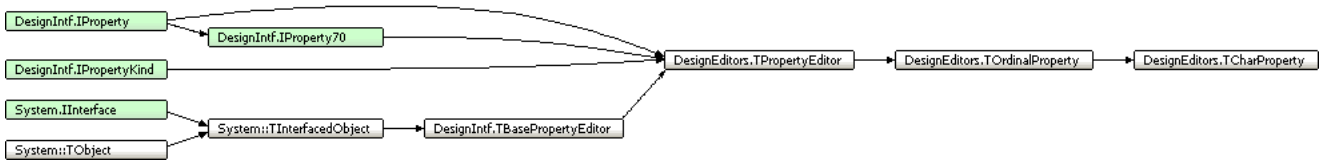

Pascal

```
TCharProperty = class(TOrdinalProperty);
```

C++

```
class TCharProperty : public TOrdinalProperty;
```

File

DesignEditors

Description

This is class DesignEditors.TCharProperty.

59.1.2.1 TCharProperty Members

The following tables list the members exposed by TCharProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
A	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AllEqual (see page 11018)	Indicates whether all selected components have the same value for the property.
V	AllEqual (see page 11024)	Indicates whether all selected components have the same value for the property.

	AllEqual (see page 11199)	Indicates whether all selected components have the same value for the property.
V	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
V	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
V	GetAttributes (see page 11026)	Describes the property so the Object Inspector provides the appropriate controls.
	GetAttributes (see page 11200)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
V	GetEditLimit (see page 11018)	Returns the maximum length, in characters, of the Value property.
V	GetEditLimit (see page 11027)	Returns the maximum length, in characters, of the Value (see page 11045) property.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
V	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
V	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
V	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
V	GetValue (see page 10921)	Returns the property value formatted as a string.
V	GetValues (see page 11035)	Provides the enumerated values of the property to a callback procedure.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.

	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetValue (see page 10922)	Responds when the user changes the value of the property.
	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
	Designer (see page 11044)	Provides access to the form designer object in the IDE.
	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.2.2 TCharProperty Methods

The methods of the TCharProperty class are listed here.

Public Methods

	Name	Description
	GetValue (see page 10921)	Returns the property value formatted as a string.
	SetValue (see page 10922)	Responds when the user changes the value of the property.

59.1.2.2.1 TCharProperty.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raised an exception.

As implemented in TPropertyEditor, GetValue returns the string "(unknown)". Descendant classes override this method to provide an appropriate value string or raise an exception.

GetValue is the read access method of the Value property.

Note: To obtain an error string when the property value is unavailable rather than raising an exception, you can call the GetEditValue method. GetEditValue is protected, but can be called through the IProperty interface in code that is not a method of a descendant class.

See Also

setvalue (see page 11042)

Value (see page 11045)

GetVisualValue (see page 11037)

GetEditValue (see page 11027)

59.1.2.2.2 TCharProperty.SetValue Method

Responds when the user changes the value of the property.

Pascal

```
procedure SetValue(const Value: string); override;
```

C++

```
virtual __fastcall SetValue(const AnsiString Value);
```

Description

The Object Inspector calls SetValue when the user has finished changing the property value. SetValue applies the value specified by the Value parameter to all components that it is currently editing. If the value can't be set, SetValue raises an exception that describes the problem.

Note: If the property is changed only through a dialog brought up by the Edit method, the Object Inspector does not call SetValue. Rather, the Edit method should handle the setting of the value on all components.

See Also

Edit (see page 11200)

59.1.3 TClassProperty Class

Class Hierarchy

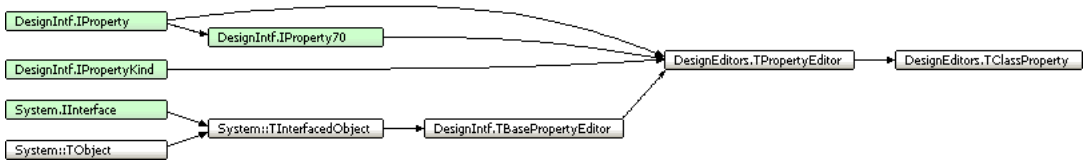

Pascal

```
TClassProperty = class(TPropertyEditor);
```

C++

```
class TClassProperty : public TPropertyEditor;
```

File

DesignEditors

Description

This is class DesignEditors.TClassProperty.

59.1.3.1 TClassProperty Members

The following tables list the members exposed by TClassProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.

	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
	AfterConstruction (see page 22279)	Responds after the last constructor executes.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AllEqual (see page 11024)	Indicates whether all selected components have the same value for the property.
	AllEqual (see page 11199)	Indicates whether all selected components have the same value for the property.
	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttributes (see page 10926)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
	GetEditLimit (see page 11027)	Returns the maximum length, in characters, of the Value (see page 11045) property.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
V	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
V	GetProperties (see page 10926)	Provides the subproperties of the property to a callback procedure.
V	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
V	GetValue (see page 10927)	Returns the property value formatted as a string.
V	GetValues (see page 11035)	Provides the enumerated values of the property to a callback procedure.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.
V	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetValue (see page 11204)	Responds when the user changes the value of the property.
V	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.3.2 TClassProperty Methods

The methods of the TClassProperty class are listed here.

Public Methods

	Name	Description
	GetAttributes (see page 10926)	Describes the property so the Object Inspector provides the appropriate controls.
	GetProperties (see page 10926)	Provides the subproperties of the property to a callback procedure.
	GetValue (see page 10927)	Returns the property value formatted as a string.

59.1.3.2.1 TClassProperty.GetAttributes Method

Describes the property so the Object Inspector provides the appropriate controls.

Pascal

```
function GetAttributes: TPropertyAttributes; override;
```

C++

```
virtual __fastcall TPropertyAttributes GetAttributes();
```

Description

The Object Inspector calls GetAttributes to determine what controls to display for the property and what property editor methods to call. The GetAttributes method of TPropertyEditor returns [paMultiSelect, paRevertable]. Override GetAttributes to return the set of values that corresponds to the attributes of the associated property.

Before the Object Inspector calls the Activate method, GetAttributes does not need to provide accurate information about any attributes other than paSubProperties, paVCL, and paMultiSelect.

See Also

Activate (see page 11024)

Edit (see page 11025)

GetProperties (see page 11033)

GetValues (see page 11035)

setvalue (see page 11042)

Value (see page 11045)

ListDrawValue

ListMeasureHeight

ListMeasureWidth

59.1.3.2.2 TClassProperty.GetProperties Method

Provides the subproperties of the property to a callback procedure.

Pascal

```
procedure GetProperties(Proc: TGetPropProc); override;
```

C++

```
virtual __fastcall GetProperties(TGetPropProc Proc);
```

Description

The Object Inspector calls GetProperties when the property is expanded to show subproperties. The GetProperties method of TPropertyEditor does nothing. Property editors that return paSubProperties from the GetAttributes method should override GetProperties to create an appropriate property editor and pass its IProperty interface to the Proc parameter for every

subproperty (or nested property) of the property.

See Also

GetAttributes (see page 11026)

59.1.3.2.3 TClassProperty.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raised an exception.

As implemented in TPropertyEditor, GetValue returns the string "(unknown)". Descendant classes override this method to provide an appropriate value string or raise an exception.

GetValue is the read access method of the Value property.

Note: To obtain an error string when the property value is unavailable rather than raising an exception, you can call the GetEditValue method. GetEditValue is protected, but can be called through the IProperty interface in code that is not a method of a descendant class.

See Also

setvalue (see page 11042)

Value (see page 11045)

GetVisualValue (see page 11037)

GetEditValue (see page 11027)

59.1.4 TComponentEditor Class

TComponentEditor provides the default implementation for the IComponentEditor interface.

Class Hierarchy

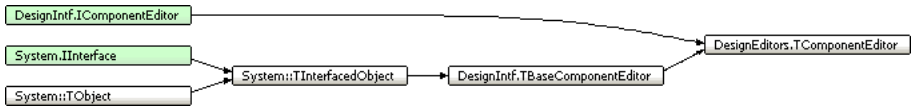

Pascal

```
TComponentEditor = class(TBaseComponentEditor, IComponentEditor);
```

C++

```
class TComponentEditor : public TBaseComponentEditor, public IComponentEditor;
```

File

DesignEditors

Description

Use TComponentEditor or one of its descendants as a base class when defining a design-time editor for working with a new component class. Although you can create component editors that descend directly from TBaseComponentEditor, it is convenient to use TComponentEditor, because this class supplies a default implementation of the IComonentEditor interface.

Create (see page 10929) a new component editor class for a component class to add verbs to its context menu, change its default double-click behavior, or allow the component to use a new clipboard format.

Once the component editor is defined, it can be registered to work with a particular component class. A registered component editor is created for each component of that class when it is selected in the form designer. When the component is double-clicked, the Edit (see page 10930) method of the component editor is called. When the user right-clicks the component, the GetVerbCount (see page 10933) and GetVerb (see page 10932) methods of the editor are called to build context menu. To create the association between a component editor and a component class, call RegisterComponentEditor.

See Also

RegisterComponentEditor

TDefaultEditor (see page 10967)

59.1.4.1 TComponentEditor Members

The following tables list the members exposed by TComponentEditor.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Copy (see page 10930)	Provides an opportunity to copy the component to the clipboard in custom formats.

V	Create (see page 10929)	Creates a new TBaseComponentEditor object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Edit (see page 10930)	Responds when the user double-clicks the component in the form designer.
V	ExecuteVerb (see page 10931)	Performs the action of a specified verb.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetComponent (see page 10931)	Returns the value of the Component (see page 10934) property.
	GetDesigner (see page 10932)	Returns the interface for the form designer.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetVerb (see page 10932)	Returns the string that corresponds to a specified position of the component's context menu.
V	GetVerbCount (see page 10933)	Returns the number of menu strings added to the context menu by the component editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsInInlined (see page 10933)	Indicates whether the component has the csInline flag set in its ComponentState property.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	PrepareItem (see page 10933)	Allows the component editor to modify menu items as they are added to the context menu.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Component (see page 10934)	Provides access to the component in the form designer that uses this component editor.
R	Designer (see page 10935)	Provides access to the form designer in the IDE.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

59.1.4.2 TComponentEditor.Create Constructor

Creates a new TBaseComponentEditor object.

Pascal

```
constructor Create(AComponent: TComponent; ADesigner: IDesigner); override;
```

C++

```
virtual __fastcall TComponentEditor(TComponent * AComponent, IDesigner ADesigner);
```

Description

Do not create instances of a component editor. Registered component editors are created automatically for selected objects in the form designer.

AComponent is the instance of the component for which the editor is created.

ADesigner is an interface that the component editor can use to interact with the forms designer.

59.1.4.3 TComponentEditor Methods

The methods of the TComponentEditor class are listed here.

Public Methods

	Name	Description
	Copy (see page 10930)	Provides an opportunity to copy the component to the clipboard in custom formats.
	Edit (see page 10930)	Responds when the user double-clicks the component in the form designer.
	ExecuteVerb (see page 10931)	Performs the action of a specified verb.
	GetComponent (see page 10931)	Returns the value of the Component (see page 10934) property.
	GetDesigner (see page 10932)	Returns the interface for the form designer.
	GetVerb (see page 10932)	Returns the string that corresponds to a specified position of the component's context menu.
	GetVerbCount (see page 10933)	Returns the number of menu strings added to the context menu by the component editor.
	IsInInlined (see page 10933)	Indicates whether the component has the csInline flag set in its ComponentState property.
	PrepareItem (see page 10933)	Allows the component editor to modify menu items as they are added to the context menu.

59.1.4.3.1 TComponentEditor.Copy Method

Provides an opportunity to copy the component to the clipboard in custom formats.

Pascal

```
procedure Copy; virtual;
```

C++

```
virtual __fastcall Copy();
```

Description

Do not call the Copy method. It is called automatically when the user tries to copy the component to the clipboard. When Copy is called, the component's filed image is already on the Clipboard. Override Copy to copy a different clipboard format, which is ignored by the designer but might be recognized by another application.

See Also

Assign (see page 3331)

SetAsHandle (see page 3343)

SetFormat

AsText (see page 3347)

TClipboard (see page 3328)

59.1.4.3.2 TComponentEditor.Edit Method

Responds when the user double-clicks the component in the form designer.

Pascal

```
procedure Edit; virtual;
```

C++

```
virtual __fastcall Edit();
```

Description

The form designer calls Edit when the user double-clicks the component. Edit executes the first verb supplied by the component editor. That is, if GetVerbCount (see page 10933) returns a value greater than zero, Edit calls

```
ExecuteVerb(0);
```

Override the Edit method to implement another response when the user double-clicks the component. For example, the Edit method can bring up a dialog or design expert.

See Also

GetVerbCount (see page 10933)

ExecuteVerb (see page 10931)

59.1.4.3.3 TComponentEditor.ExecuteVerb Method

Performs the action of a specified verb.

Pascal

```
procedure ExecuteVerb(Index: Integer); virtual;
```

C++

```
virtual __fastcall ExecuteVerb(int Index);
```

Description

The form designer calls ExecuteVerb when the user selects the verb that is returned by GetVerb (see page 10932)(Index). This verb is the string that appears in the corresponding position of the component's context menu.

TComponentEditor (see page 10927) does not supply any verbs for its component. As a result, ExecuteVerb does nothing. Override GetVerb (see page 10932) to provide a set of verbs, or menu strings, that are added to the context menu which appears when the user right-clicks the component. Override ExecuteVerb to implement a response to each of these verbs.

See Also

getverb (see page 10932)

GetVerbCount (see page 10933)

59.1.4.3.4 TComponentEditor.GetComponent Method

Returns the value of the Component (see page 10934) property.

Pascal

```
function GetComponent: TComponent;
```

C++

```
__fastcall TComponent * GetComponent();
```

Description

Use GetComponent to read from and make changes to the component associated with the component editor. Calling GetComponent is equivalent to reading the Component (see page 10934) property, but unlike the Component (see page

10934) property, is available through the IComponentEditor interface.

A component editor is created for each instance of the registered component class when it is selected in the form designer. GetComponent returns the selected instance.

Note: When the component editor makes changes to the component, it must call the Modified method of the form designer that is specified by the Designer (see page 10935) property.

See Also

Component (see page 10934)

Designer (see page 10935)

IComponentEditor (see page 11138)

59.1.4.3.5 TComponentEditor.GetDesigner Method

Returns the interface for the form designer.

Pascal

```
function GetDesigner: IDesigner;
```

C++

```
__fastcall IDesigner GetDesigner();
```

Description

GetDesigner is the read implementation for the Designer (see page 10935) property.

See Also

Designer (see page 10935)

IDesigner (see page 11149)

59.1.4.3.6 TComponentEditor.GetVerb Method

Returns the string that corresponds to a specified position of the component's context menu.

Pascal

```
function GetVerb(Index: Integer): string; virtual;
```

C++

```
virtual __fastcall AnsiString GetVerb(int Index);
```

Description

The form designer calls GetVerb iteratively to build the context menu that appears when the user right-clicks the component.

When implementing GetVerb, add an ampersand (&) to a verb to cause the following character to appear underlined in the context menu and to act as a shortcut key for selecting the menu item. Be sure to add ellipsis (...) to the end of a verb if its corresponding action is to bring up a dialog.

See Also

executeverb (see page 11140)

GetVerbCount (see page 11142)

59.1.4.3.7 TComponentEditor.GetVerbCount Method

Returns the number of menu strings added to the context menu by the component editor.

Pascal

```
function GetVerbCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetVerbCount();
```

Description

The form designer calls GetVerbCount when the user right-clicks the component in order to determine the number of menu items that should be added to the context menu. It then adds the menu items by calling GetVerb (see page 10932) for every value from 0 to GetVerbCount - 1.

TComponentEditor (see page 10927) does not supply any verbs for its component. As a result, GetVerbCount returns 0. Override GetVerbCount when providing a set of verbs, or menu strings, to be added to the context menu. When overriding GetVerbCount, override GetVerb (see page 10932) to return the actual strings, and override ExecuteVerb (see page 10931) to implement a response to each of these verbs. For example, if adding two verbs to the context menu, GetVerbCount should return 2, GetVerb (see page 10932) should return a value for an Index of 0 and for an Index of 1, and ExecuteVerb (see page 10931) should perform the corresponding actions for an Index of 0 and for an Index of 1.

See Also

getverb (see page 10932)

ExecuteVerb (see page 10931)

PrepareItem (see page 10933)

59.1.4.3.8 TComponentEditor.IsInInlined Method

Indicates whether the component has the csInline flag set in its ComponentState property.

Pascal

```
function IsInInlined: Boolean;
```

C++

```
__fastcall Boolean IsInInlined();
```

Description

IsInInlined indicates whether the component being edited is inside a top-level component that can be modified at design time and also embedded in a form.

For instance, when a component is used on a Frame and that Frame is dropped on a Form, IsInInlined returns true for the component when it is embedded in the Frame on the Form, but returns false when the component that is only on the Frame.

See Also

ComponentState (see page 2804)

59.1.4.3.9 TComponentEditor.PrepareItem Method

Allows the component editor to modify menu items as they are added to the context menu.

Pascal

```
procedure PrepareItem(Index: Integer; const AItem: IMenuItem); virtual;
```

C++

```
virtual __fastcall PrepareItem(int Index, const IMenuItem AItem);
```

Description

The form designer calls PrepareItem for each verb when it constructs the context menu. This method lets the component editor make any modifications to the menu item for each verb, including adding subitems or removing the item from the context menu.

Index identifies the current verb that the form designer is adding to the context menu. This is the same index as appears for the command in the GetVerb (see page 10932) and ExecuteVerb (see page 10931) methods.

AItem is an interface to the menu item created by the form designer for the current verb. To add subitems, call the AddItem method of this interface. To disable the item, set the Enabled property to false. To suppress the menu item altogether, set its Visible property to false.

As implemented in TComponentEditor (see page 10927), PrepareItem does nothing. Override this method to customize the context menu before it is displayed.

See Also

getverb (see page 10932)

GetVerbCount (see page 10933)

ExecuteVerb (see page 10931)

59.1.4.4 TComponentEditor Properties

The properties of the TComponentEditor class are listed here.

Public Properties

	Name	Description
R	Component (see page 10934)	Provides access to the component in the form designer that uses this component editor.
R	Designer (see page 10935)	Provides access to the form designer in the IDE.

59.1.4.4.1 TComponentEditor.Component Property

Provides access to the component in the form designer that uses this component editor.

Pascal

```
property Component: TComponent;
```

C++

```
__property TComponent * Component;
```

Description

Use Component to read from and make changes to the component associated with the component editor. A component editor is created for each instance of the registered component class when it is selected in the form designer. Component is the selected instance.

Note: When the component editor makes changes to the component, it must call the Modified method of the form designer that is specified by the Designer (see page 10935) property.

See Also

create (see page 10929)

Designer (see page 10935)

Modified

59.1.4.4.2 TComponentEditor.Designer Property

Provides access to the form designer in the IDE.

Pascal

```
property Designer: IDesigner;
```

C++

```
__property IDesigner Designer;
```

Description

Use Designer to read from or make changes to the current form or module in the IDE. The IDesigner interface has methods that can access the components in the current unit and other units linked in by the uses clause (Delphi) or defined in included header files (C++). Other IDesigner methods create or alter the methods in the current unit and so on.

Note: When the component editor makes changes to the component, it must call the Modified method of the IDesigner interface.

See Also

Component (see page 10934)

Modified

59.1.5 TComponentGuidelines Class

Class Hierarchy

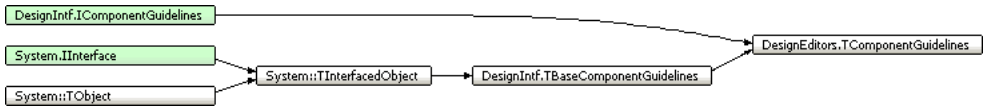

Pascal

```
TComponentGuidelines = class(TBaseComponentGuidelines, IComponentGuidelines);
```

C++

```
class TComponentGuidelines : public TBaseComponentGuidelines, public IComponentGuidelines;
```

File

DesignEditors

Description

This is class DesignEditors.TComponentGuidelines.

59.1.5.1 TComponentGuidelines Members

The following tables list the members exposed by TComponentGuidelines.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
V	CalcHorzPos (see page 10937)	This is CalcHorzPos, a member of class TComponentGuidelines.
V	CalcVertPos (see page 10938)	This is CalcVertPos, a member of class TComponentGuidelines.
V	GetCount (see page 10938)	This is GetCount, a member of class TComponentGuidelines.
V	GetDesignerGuideOffset (see page 10938)	This is GetDesignerGuideOffset, a member of class TComponentGuidelines.
V	GetDesignerGuideType (see page 10938)	This is GetDesignerGuideType, a member of class TComponentGuidelines.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Protected Properties

	Name	Description
R	Component (see page 10939)	This is Component, a member of class TComponentGuidelines.
R	Container (see page 10939)	This is Container, a member of class TComponentGuidelines.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11109)	This is Create, a member of class TBaseComponentGuidelines.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 10938)	This is Initialize, a member of class TComponentGuidelines.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Count (see page 11145)	This is Count, a member of class IComponentGuidelines.
R	Designer (see page 11110)	This is Designer, a member of class TBaseComponentGuidelines.
R	DesignerGuideOffsets (see page 11145)	This is DesignerGuideOffsets, a member of class IComponentGuidelines.
R	DesignerGuideTypes (see page 11145)	This is DesignerGuideTypes, a member of class IComponentGuidelines.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

59.1.5.2 TComponentGuidelines Methods

The methods of the TComponentGuidelines class are listed here.

Protected Methods

	Name	Description
V	CalcHorzPos (see page 10937)	This is CalcHorzPos, a member of class TComponentGuidelines.
V	CalcVertPos (see page 10938)	This is CalcVertPos, a member of class TComponentGuidelines.
V	GetCount (see page 10938)	This is GetCount, a member of class TComponentGuidelines.
V	GetDesignerGuideOffset (see page 10938)	This is GetDesignerGuideOffset, a member of class TComponentGuidelines.
V	GetDesignerGuideType (see page 10938)	This is GetDesignerGuideType, a member of class TComponentGuidelines.

Public Methods

	Name	Description
V	Initialize (see page 10938)	This is Initialize, a member of class TComponentGuidelines.

59.1.5.2.1 TComponentGuidelines.CalcHorzPos Method

Pascal

```
function CalcHorzPos(APos: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int CalcHorzPos(int APos);
```

Description

This is CalcHorzPos, a member of class TComponentGuidelines.

59.1.5.2.2 TComponentGuidelines.CalcVertPos Method

Pascal

```
function CalcVertPos(APos: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int CalcVertPos(int APos);
```

Description

This is CalcVertPos, a member of class TComponentGuidelines.

59.1.5.2.3 TComponentGuidelines.GetCount Method

Pascal

```
function GetCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetCount();
```

Description

This is GetCount, a member of class TComponentGuidelines.

59.1.5.2.4 TComponentGuidelines.GetDesignerGuideOffset Method

Pascal

```
function GetDesignerGuideOffset(Index: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int GetDesignerGuideOffset(int Index);
```

Description

This is GetDesignerGuideOffset, a member of class TComponentGuidelines.

59.1.5.2.5 TComponentGuidelines.GetDesignerGuideType Method

Pascal

```
function GetDesignerGuideType(Index: Integer): TDesignerGuideType; virtual;
```

C++

```
virtual __fastcall TDesignerGuideType GetDesignerGuideType(int Index);
```

Description

This is GetDesignerGuideType, a member of class TComponentGuidelines.

59.1.5.2.6 TComponentGuidelines.Initialize Method

Pascal

```
procedure Initialize(AComponent: TComponent; AContainer: TComponent); override;
```

C++

```
virtual __fastcall Initialize(TComponent * AComponent, TComponent * AContainer);
```

Description

This is Initialize, a member of class TComponentGuidelines.

59.1.5.3 TComponentGuidelines Properties

The properties of the TComponentGuidelines class are listed here.

Protected Properties

	Name	Description
R	Component (see page 10939)	This is Component, a member of class TComponentGuidelines.
R	Container (see page 10939)	This is Container, a member of class TComponentGuidelines.

59.1.5.3.1 TComponentGuidelines.Component Property

Pascal

```
property Component: TComponent;
```

C++

```
__property TComponent * Component;
```

Description

This is Component, a member of class TComponentGuidelines.

59.1.5.3.2 TComponentGuidelines.Container Property

Pascal

```
property Container: TComponent;
```

C++

```
__property TComponent * Container;
```

Description

This is Container, a member of class TComponentGuidelines.

59.1.6 TComponentNameProperty Class

Class Hierarchy

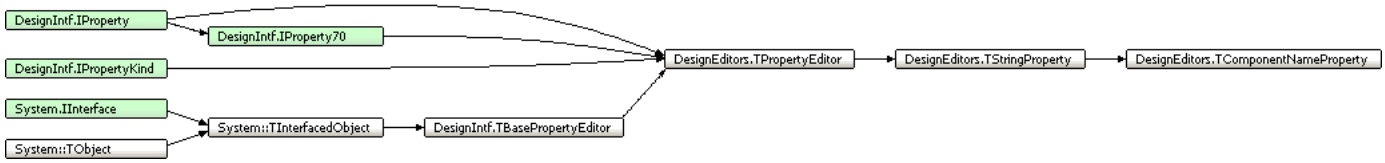

Pascal

```
TComponentNameProperty = class(TStringProperty);
```

C++

```
class TComponentNameProperty : public TStringProperty;
```

File

DesignEditors

Description

This is class DesignEditors.TComponentNameProperty.

59.1.6.1 TComponentNameProperty Members

The following tables list the members exposed by TComponentNameProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
A	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.

	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AllEqual (see page 11024)	Indicates whether all selected components have the same value for the property.
V	AllEqual (see page 11065)	Indicates whether all selected components have the same value for the property.
	AllEqual (see page 11199)	Indicates whether all selected components have the same value for the property.
V	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
V	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
V	GetAttributes (see page 10943)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
V	GetEditLimit (see page 10943)	Returns the maximum length, in characters, of the Value property.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.

V	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
V	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
V	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
V	GetValue (see page 11035)	Returns the property value formatted as a string.
V	GetValue (see page 11065)	Returns the property value formatted as a string.
	GetValue (see page 11203)	Returns the property value formatted as a string.
V	GetValues (see page 11035)	Provides the enumerated values of the property to a callback procedure.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.
V	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetValue (see page 11066)	Responds when the user changes the value of the property.
	SetValue (see page 11204)	Responds when the user changes the value of the property.
V	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.6.2 TComponentNameProperty Methods

The methods of the TComponentNameProperty class are listed here.

Public Methods

	Name	Description
	GetAttributes (see page 10943)	Describes the property so the Object Inspector provides the appropriate controls.
	GetEditLimit (see page 10943)	Returns the maximum length, in characters, of the Value property.

59.1.6.2.1 TComponentNameProperty.GetAttributes Method

Describes the property so the Object Inspector provides the appropriate controls.

Pascal

```
function GetAttributes: TPropertyAttributes; override;
```

C++

```
virtual __fastcall TPropertyAttributes GetAttributes();
```

Description

The Object Inspector calls GetAttributes to determine what controls to display for the property and what property editor methods to call. The GetAttributes method of TPropertyEditor returns [paMultiSelect, paRevertable]. Override GetAttributes to return the set of values that corresponds to the attributes of the associated property.

Before the Object Inspector calls the Activate method, GetAttributes does not need to provide accurate information about any attributes other than paSubProperties, paVCL, and paMultiSelect.

See Also

Activate (see page 11024)

Edit (see page 11025)

GetProperties (see page 11033)

GetValues (see page 11035)

setvalue (see page 11042)

Value (see page 11045)

ListDrawValue

ListMeasureHeight

ListMeasureWidth

59.1.6.2.2 TComponentNameProperty.GetEditLimit Method

Returns the maximum length, in characters, of the Value property.

Pascal

```
function GetEditLimit: Integer; override;
```

C++

```
virtual __fastcall int GetEditLimit();
```

Description

The Object Inspector calls GetEditLimit to determine how many characters the user can type when entering a value for the property. GetEditLimit returns 255. Override this method when the maximum length of the property string is some other value.

See Also

Value (see page 11045)

59.1.7 TComponentProperty Class

Class Hierarchy

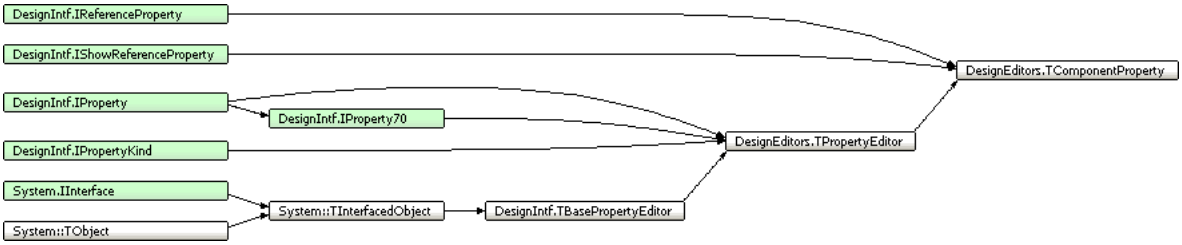

Pascal

```
TComponentProperty = class(TPropertyEditor, IReferenceProperty, IShowReferenceProperty);
```

C++

```
class TComponentProperty : public TPropertyEditor, public IReferenceProperty, public
IShowReferenceProperty;
```

File

DesignEditors

Description

This is class DesignEditors.TComponentProperty.

59.1.7.1 TComponentProperty Members

The following tables list the members exposed by TComponentProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	FilterFunc (see page 10948)	This is FilterFunc, a member of class TComponentProperty.
V	GetComponentReference (see page 10949)	This is GetComponentReference, a member of class TComponentProperty.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.

	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
V	GetSelections (see page 10950)	This is GetSelections, a member of class TComponentProperty.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
A	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.
V	ShowReferenceProperty (see page 10951)	This is ShowReferenceProperty, a member of class TComponentProperty.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AllEqual (see page 10947)	Indicates whether all selected components have the same value for the property.
V	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
V	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.

	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Edit (see page 10948)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
V	GetAttributes (see page 10948)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
V	GetEditLimit (see page 10949)	Returns the maximum length, in characters, of the Value property.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
V	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
V	GetProperties (see page 10949)	Provides the subproperties of the property to a callback procedure.
V	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
V	GetValue (see page 10950)	Returns the property value formatted as a string.
V	GetValues (see page 10951)	Provides the enumerated values of the property to a callback procedure.
V	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetValue (see page 10951)	Responds when the user changes the value of the property.
V	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.7.2 TComponentProperty Methods

The methods of the TComponentProperty class are listed here.

Protected Methods

	Name	Description
	FilterFunc (see page 10948)	This is FilterFunc, a member of class TComponentProperty.
V	GetComponentReference (see page 10949)	This is GetComponentReference, a member of class TComponentProperty.
V	GetSelections (see page 10950)	This is GetSelections, a member of class TComponentProperty.
V	ShowReferenceProperty (see page 10951)	This is ShowReferenceProperty, a member of class TComponentProperty.

Public Methods

	Name	Description
V	AllEqual (see page 10947)	Indicates whether all selected components have the same value for the property.
V	Edit (see page 10948)	Responds when the user double-clicks the property in the Object Inspector.
V	GetAttributes (see page 10948)	Describes the property so the Object Inspector provides the appropriate controls.
V	GetEditLimit (see page 10949)	Returns the maximum length, in characters, of the Value property.
V	GetProperties (see page 10949)	Provides the subproperties of the property to a callback procedure.
V	GetValue (see page 10950)	Returns the property value formatted as a string.
V	GetValues (see page 10951)	Provides the enumerated values of the property to a callback procedure.
V	SetValue (see page 10951)	Responds when the user changes the value of the property.

59.1.7.2.1 TComponentProperty.AllEqual Method

Indicates whether all selected components have the same value for the property.

Pascal

```
function AllEqual: Boolean; override;
```

C++

```
virtual __fastcall Boolean AllEqual();
```

Description

The Object Inspector calls AllEqual when there is more than one component selected to determine whether a value should appear for this property. If this method returns true, the Object Inspector displays a value for the property. If this method returns false, the line for the property in the Object Inspector remains blank.

In TPropertyEditor, AllEqual always returns false if more than one selected component has the associated property. Override this method to check the properties of individual components and determine whether a property value can be displayed in the Object Inspector.

Note: AllEqual is called only when the value returned by GetAttributes includes paMultiSelect.

See Also

GetAttributes (see page 11026)

GetComponent (see page 11026)

GetValue (see page 11035)

59.1.7.2.2 TComponentProperty.Edit Method

Responds when the user double-clicks the property in the Object Inspector.

Pascal

```
procedure Edit; override;
```

C++

```
virtual __fastcall Edit();
```

Description

The Object Inspector calls Edit when the ellipsis button (...) is pressed or the property is double-clicked. The Edit method of TPropertyEditor cycles through the possible enumerated values of the property when the GetValues method returns one or more property strings to the callback method.

Override the Edit method to implement a response to the ellipsis button or double-clicking the property. For example, use the Edit method to bring up a dialog for editing the property in some more meaningful fashion than typing a text value in the Object Inspector.

See Also

GetValues (see page 11035)

Value (see page 11045)

59.1.7.2.3 TComponentProperty.FilterFunc Method

Pascal

```
function FilterFunc(const ATestEditor: IProperty): Boolean;
```

C++

```
__fastcall Boolean FilterFunc(const IProperty ATestEditor);
```

Description

This is FilterFunc, a member of class TComponentProperty.

59.1.7.2.4 TComponentProperty.GetAttributes Method

Describes the property so the Object Inspector provides the appropriate controls.

Pascal

```
function GetAttributes: TPropertyAttributes; override;
```

C++

```
virtual __fastcall TPropertyAttributes GetAttributes();
```

Description

The Object Inspector calls GetAttributes to determine what controls to display for the property and what property editor methods

to call. The GetAttributes method of TPropertyEditor returns [paMultiSelect, paRevertable]. Override GetAttributes to return the set of values that corresponds to the attributes of the associated property.

Before the Object Inspector calls the Activate method, GetAttributes does not need to provide accurate information about any attributes other than paSubProperties, paVCL, and paMultiSelect.

See Also

Activate (see page 11024)

Edit (see page 11025)

GetProperties (see page 11033)

GetValues (see page 11035)

setvalue (see page 11042)

Value (see page 11045)

ListDrawValue

ListMeasureHeight

ListMeasureWidth

59.1.7.2.5 TComponentProperty.GetComponentReference Method

Pascal

```
function GetComponentReference: TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * GetComponentReference();
```

Description

This is GetComponentReference, a member of class TComponentProperty.

59.1.7.2.6 TComponentProperty.GetEditLimit Method

Returns the maximum length, in characters, of the Value property.

Pascal

```
function GetEditLimit: Integer; override;
```

C++

```
virtual __fastcall int GetEditLimit();
```

Description

The Object Inspector calls GetEditLimit to determine how many characters the user can type when entering a value for the property. GetEditLimit returns 255. Override this method when the maximum length of the property string is some other value.

See Also

Value (see page 11045)

59.1.7.2.7 TComponentProperty.GetProperties Method

Provides the subproperties of the property to a callback procedure.

Pascal

```
procedure GetProperties(Proc: TGetPropProc); override;
```

C++

```
virtual __fastcall GetProperties(TGetPropProc Proc);
```

Description

The Object Inspector calls GetProperties when the property is expanded to show subproperties. The GetProperties method of TPropertyEditor does nothing. Property editors that return paSubProperties from the GetAttributes method should override GetProperties to create an appropriate property editor and pass its IProperty interface to the Proc parameter for every subproperty (or nested property) of the property.

See Also

GetAttributes (see page 11026)

59.1.7.2.8 TComponentProperty.GetSelections Method

Pascal

```
function GetSelections: IDesignerSelections; virtual;
```

C++

```
virtual __fastcall IDesignerSelections GetSelections();
```

Description

This is GetSelections, a member of class TComponentProperty.

59.1.7.2.9 TComponentProperty.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raised an exception.

As implemented in TPropertyEditor, GetValue returns the string "(unknown)". Descendant classes override this method to provide an appropriate value string or raise an exception.

GetValue is the read access method of the Value property.

Note: To obtain an error string when the property value is unavailable rather than raising an exception, you can call the GetEditValue method. GetEditValue is protected, but can be called through the IProperty interface in code that is not a method of a descendant class.

See Also

setvalue (see page 11042)

Value (see page 11045)

GetVisualValue (see page 11037)

GetEditValue (see page 11027)

59.1.7.2.10 TComponentProperty.GetValues Method

Provides the enumerated values of the property to a callback procedure.

Pascal

```
procedure GetValues(Proc: TGetStrProc); override;
```

C++

```
virtual __fastcall GetValues(TGetStrProc Proc);
```

Description

The Object Inspector calls GetValues when the drop-down button is clicked to show enumerated values. The GetValues method of TPropertyEditor does nothing. Property editors that return paValueList from the GetAttributes method should override GetValues to call the Proc parameter for every string that can represent a valid enumerated value. Even if the GetAttributes method does not return paValueList, property editors for enumerated types should still override GetValues to enumerate all possible values.

See Also

GetAttributes (see page 11026)

Value (see page 11045)

59.1.7.2.11 TComponentProperty.SetValue Method

Responds when the user changes the value of the property.

Pascal

```
procedure SetValue(const Value: string); override;
```

C++

```
virtual __fastcall SetValue(const AnsiString Value);
```

Description

The Object Inspector calls SetValue when the user has finished changing the property value. SetValue applies the value specified by the Value parameter to all components that it is currently editing. If the value can't be set, SetValue raises an exception that describes the problem.

Note: If the property is changed only through a dialog brought up by the Edit method, the Object Inspector does not call SetValue. Rather, the Edit method should handle the setting of the value on all components.

See Also

Edit (see page 11200)

59.1.7.2.12 TComponentProperty.ShowReferenceProperty Method

Pascal

```
function ShowReferenceProperty: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ShowReferenceProperty();
```

Description

This is ShowReferenceProperty, a member of class TComponentProperty.

59.1.8 TCustomModule Class

Class Hierarchy

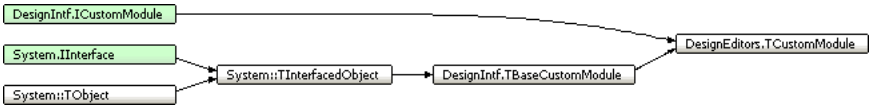

Pascal

```
TCustomModule = class(TBaseCustomModule, ICustomModule);
```

C++

```
class TCustomModule : public TBaseCustomModule, public ICustomModule;
```

File

DesignEditors

Description

This is class DesignEditors.TCustomModule.

59.1.8.1 TCustomModule Members

The following tables list the members exposed by TCustomModule.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 10953)	This is Create, a member of class TCustomModule.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	DesignClass (see page 11112)	This is DesignClass, a member of class TBaseCustomModule.
V	Destroy (see page 10954)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	ExecuteVerb (see page 10955)	This is ExecuteVerb, a member of class TCustomModule.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
V	GetAttributes (see page 10955)	This is GetAttributes, a member of class TCustomModule.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetVerb (see page 10955)	This is GetVerb, a member of class TCustomModule.
V	GetVerbCount (see page 10955)	This is GetVerbCount, a member of class TCustomModule.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	Nestable (see page 10955)	This is Nestable, a member of class TCustomModule.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	PrepareItem (see page 10956)	This is PrepareItem, a member of class TCustomModule.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Saving (see page 10956)	This is Saving, a member of class TCustomModule.
V	ValidateComponent (see page 10956)	This is ValidateComponent, a member of class TCustomModule.
V	ValidateComponentClass (see page 10956)	This is ValidateComponentClass, a member of class TCustomModule.

Public Properties

	Name	Description
R	Designer (see page 10957)	This is Designer, a member of class TCustomModule.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	Root (see page 10957)	This is Root, a member of class TCustomModule.

59.1.8.2 TCustomModule.Create Constructor

Pascal

```
constructor Create(ARoot: TComponent; const ADesigner: IDesigner); override;
```

C++

```
virtual __fastcall TCustomModule(TComponent * ARoot, const IDesigner ADesigner);
```

Description

This is Create, a member of class TCustomModule.

59.1.8.3 TCustomModule.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomModule();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

59.1.8.4 TCustomModule Methods

The methods of the TCustomModule class are listed here.

Public Methods

	Name	Description
	ExecuteVerb (see page 10955)	This is ExecuteVerb, a member of class TCustomModule.
	GetAttributes (see page 10955)	This is GetAttributes, a member of class TCustomModule.
	GetVerb (see page 10955)	This is GetVerb, a member of class TCustomModule.
	GetVerbCount (see page 10955)	This is GetVerbCount, a member of class TCustomModule.
	Nestable (see page 10955)	This is Nestable, a member of class TCustomModule.
	PrepareItem (see page 10956)	This is PrepareItem, a member of class TCustomModule.
	Saving (see page 10956)	This is Saving, a member of class TCustomModule.
	ValidateComponent (see page 10956)	This is ValidateComponent, a member of class TCustomModule.
	ValidateComponentClass (see page 10956)	This is ValidateComponentClass, a member of class TCustomModule.

59.1.8.4.1 TCustomModule.ExecuteVerb Method

Pascal

```
procedure ExecuteVerb(Index: Integer); virtual;
```

C++

```
virtual __fastcall ExecuteVerb(int Index);
```

Description

This is ExecuteVerb, a member of class TCustomModule.

59.1.8.4.2 TCustomModule.GetAttributes Method

Pascal

```
function GetAttributes: TCustomModuleAttributes; virtual;
```

C++

```
virtual __fastcall TCustomModuleAttributes GetAttributes();
```

Description

This is GetAttributes, a member of class TCustomModule.

59.1.8.4.3 TCustomModule.GetVerb Method

Pascal

```
function GetVerb(Index: Integer): string; virtual;
```

C++

```
virtual __fastcall AnsiString GetVerb(int Index);
```

Description

This is GetVerb, a member of class TCustomModule.

59.1.8.4.4 TCustomModule.GetVerbCount Method

Pascal

```
function GetVerbCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetVerbCount();
```

Description

This is GetVerbCount, a member of class TCustomModule.

59.1.8.4.5 TCustomModule.Nestable Method

Pascal

```
function Nestable: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean Nestable();
```

Description

This is Nestable, a member of class TCustomModule.

59.1.8.4.6 TCustomModule.PrepareItem Method

Pascal

```
procedure PrepareItem(Index: Integer; const AItem: IMenuItem); virtual;
```

C++

```
virtual __fastcall PrepareItem(int Index, const IMenuItem AItem);
```

Description

This is PrepareItem, a member of class TCustomModule.

59.1.8.4.7 TCustomModule.Saving Method

Pascal

```
procedure Saving; virtual;
```

C++

```
virtual __fastcall Saving();
```

Description

This is Saving, a member of class TCustomModule.

59.1.8.4.8 TCustomModule.ValidateComponent Method

Pascal

```
procedure ValidateComponent(Component: TComponent); virtual;
```

C++

```
virtual __fastcall ValidateComponent(TComponent * Component);
```

Description

This is ValidateComponent, a member of class TCustomModule.

59.1.8.4.9 TCustomModule.ValidateComponentClass Method

Pascal

```
function ValidateComponentClass(ComponentClass: TComponentClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ValidateComponentClass(TComponentClass ComponentClass);
```

Description

This is ValidateComponentClass, a member of class TCustomModule.

59.1.8.5 TCustomModule Properties

The properties of the TCustomModule class are listed here.

Public Properties

	Name	Description
R	Designer (see page 10957)	This is Designer, a member of class TCustomModule.
R	Root (see page 10957)	This is Root, a member of class TCustomModule.

59.1.8.5.1 TCustomModule.Designer Property

Pascal

```
property Designer: IDesigner;
```

C++

```
__property IDesigner Designer;
```

Description

This is Designer, a member of class TCustomModule.

59.1.8.5.2 TCustomModule.Root Property

Pascal

```
property Root: TComponent;
```

C++

```
__property TComponent * Root;
```

Description

This is Root, a member of class TCustomModule.

59.1.9 TDateProperty Class

Class Hierarchy

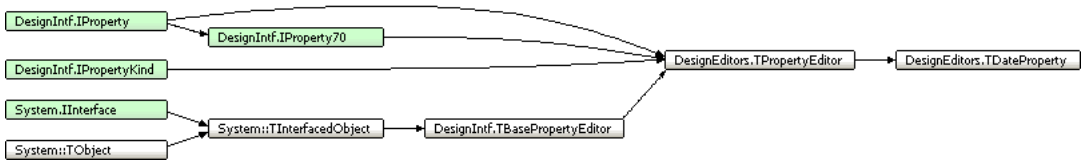

Pascal

```
TDateProperty = class(TPropertyEditor);
```

C++

```
class TDateProperty : public TPropertyEditor;
```

File

DesignEditors

Description

This is class DesignEditors.TDateProperty.

59.1.9.1 TDateProperty Members

The following tables list the members exposed by TDateProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
A	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.

	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
	AfterConstruction (see page 22279)	Responds after the last constructor executes.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AllEqual (see page 11024)	Indicates whether all selected components have the same value for the property.
	AllEqual (see page 11199)	Indicates whether all selected components have the same value for the property.
	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttributes (see page 10960)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
	GetEditLimit (see page 11027)	Returns the maximum length, in characters, of the Value (see page 11045) property.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
	GetValue (see page 10961)	Returns the property value formatted as a string.

V	GetValues (see page 11035)	Provides the enumerated values of the property to a callback procedure.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.
V	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetValue (see page 10962)	Responds when the user changes the value of the property.
V	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.9.2 TDateProperty Methods

The methods of the TDateProperty class are listed here.

Public Methods

	Name	Description
V	GetAttributes (see page 10960)	Describes the property so the Object Inspector provides the appropriate controls.
V	GetValue (see page 10961)	Returns the property value formatted as a string.
V	SetValue (see page 10962)	Responds when the user changes the value of the property.

59.1.9.2.1 TDateProperty.GetAttributes Method

Describes the property so the Object Inspector provides the appropriate controls.

Pascal

```
function GetAttributes: TPropertyAttributes; override;
```

C++

```
virtual __fastcall TPropertyAttributes GetAttributes();
```

Description

The Object Inspector calls GetAttributes to determine what controls to display for the property and what property editor methods to call. The GetAttributes method of TPropertyEditor returns [paMultiSelect, paRevertable]. Override GetAttributes to return the set of values that corresponds to the attributes of the associated property.

Before the Object Inspector calls the Activate method, GetAttributes does not need to provide accurate information about any attributes other than paSubProperties, paVCL, and paMultiSelect.

See Also

Activate (see page 11024)

Edit (see page 11025)

GetProperties (see page 11033)

GetValues (see page 11035)

setvalue (see page 11042)

Value (see page 11045)

ListDrawValue

ListMeasureHeight

ListMeasureWidth

59.1.9.2.2 TDateProperty.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raised an exception.

As implemented in TPropertyEditor, GetValue returns the string "(unknown)". Descendant classes override this method to provide an appropriate value string or raise an exception.

GetValue is the read access method of the Value property.

Note: To obtain an error string when the property value is unavailable rather than raising an exception, you can call the GetEditValue method. GetEditValue is protected, but can be called through the IProperty interface in code that is not a method of a descendant class.

See Also

setvalue (see page 11042)

Value (see page 11045)

GetVisualValue (see page 11037)

GetEditValue (see page 11027)

59.1.9.2.3 TDateProperty.SetValue Method

Responds when the user changes the value of the property.

Pascal

```
procedure SetValue(const Value: string); override;
```

C++

```
virtual __fastcall SetValue(const AnsiString Value);
```

Description

The Object Inspector calls SetValue when the user has finished changing the property value. SetValue applies the value specified by the Value parameter to all components that it is currently editing. If the value can't be set, SetValue raises an exception that describes the problem.

Note: If the property is changed only through a dialog brought up by the Edit method, the Object Inspector does not call SetValue. Rather, the Edit method should handle the setting of the value on all components.

See Also

Edit (see page 11200)

59.1.10 TDateTimeProperty Class

Class Hierarchy

Pascal

```
TDateTimeProperty = class(TPropertyEditor);
```

C++

```
class TDateTimeProperty : public TPropertyEditor;
```

File

DesignEditors

Description

This is class DesignEditors.TDateTimeProperty.

59.1.10.1 TDateTimeProperty Members

The following tables list the members exposed by TDateTimeProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
A	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AllEqual (see page 11024)	Indicates whether all selected components have the same value for the property.
	AllEqual (see page 11199)	Indicates whether all selected components have the same value for the property.
V	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.

	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttributes (see page 10965)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
	GetEditLimit (see page 11027)	Returns the maximum length, in characters, of the Value (see page 11045) property.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
	GetValue (see page 10966)	Returns the property value formatted as a string.
	GetValues (see page 11035)	Provides the enumerated values of the property to a callback procedure.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.
	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetValue (see page 10966)	Responds when the user changes the value of the property.
V	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.10.2 TDateTimeProperty Methods

The methods of the TDateTimeProperty class are listed here.

Public Methods

	Name	Description
V	GetAttributes (see page 10965)	Describes the property so the Object Inspector provides the appropriate controls.
V	GetValue (see page 10966)	Returns the property value formatted as a string.
V	SetValue (see page 10966)	Responds when the user changes the value of the property.

59.1.10.2.1 TDateTimeProperty.GetAttributes Method

Describes the property so the Object Inspector provides the appropriate controls.

Pascal

```
function GetAttributes: TPropertyAttributes; override;
```

C++

```
virtual __fastcall TPropertyAttributes GetAttributes();
```

Description

The Object Inspector calls GetAttributes to determine what controls to display for the property and what property editor methods to call. The GetAttributes method of TPropertyEditor returns [paMultiSelect, paRevertable]. Override GetAttributes to return the set of values that corresponds to the attributes of the associated property.

Before the Object Inspector calls the Activate method, GetAttributes does not need to provide accurate information about any attributes other than paSubProperties, paVCL, and paMultiSelect.

See Also

Activate (see page 11024)

Edit (see page 11025)

GetProperties (see page 11033)

GetValues (see page 11035)

setvalue (see page 11042)

Value (see page 11045)

ListDrawValue

ListMeasureHeight

ListMeasureWidth

59.1.10.2.2 TDateTimeProperty.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raised an exception.

As implemented in TPropertyEditor, GetValue returns the string "(unknown)". Descendant classes override this method to provide an appropriate value string or raise an exception.

GetValue is the read access method of the Value property.

Note: To obtain an error string when the property value is unavailable rather than raising an exception, you can call the GetEditValue method. GetEditValue is protected, but can be called through the IProperty interface in code that is not a method of a descendant class.

See Also

setvalue (see page 11042)

Value (see page 11045)

GetVisualValue (see page 11037)

GetEditValue (see page 11027)

59.1.10.2.3 TDateTimeProperty.SetValue Method

Responds when the user changes the value of the property.

Pascal

```
procedure SetValue(const Value: string); override;
```

C++

```
virtual __fastcall SetValue(const AnsiString Value);
```

Description

The Object Inspector calls SetValue when the user has finished changing the property value. SetValue applies the value specified by the Value parameter to all components that it is currently editing. If the value can't be set, SetValue raises an exception that describes the problem.

Note: If the property is changed only through a dialog brought up by the Edit method, the Object Inspector does not call SetValue. Rather, the Edit method should handle the setting of the value on all components.

See Also

Edit (see page 11200)

59.1.11 TDefaultEditor Class

TDefaultEditor is the component editor for components that do not explicitly register a component editor.

Class Hierarchy

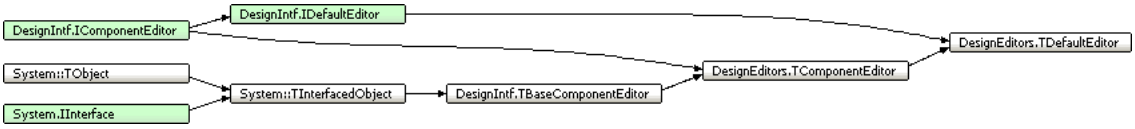

Pascal

```
TDefaultEditor = class(TComponentEditor, IDefaultEditor);
```

C++

```
class TDefaultEditor : public TComponentEditor, public IDefaultEditor;
```

File

DesignEditors

Description

When a component class does not have a registered property editor, TDefaultEditor controls its response to right clicks, double clicks, and clipboard commands.

TDefaultEditor does not add any custom menu items to the context menu that appears when the user right-clicks a component in the designer, nor does it add any clipboard formats to the standard component format. However, TDefaultEditor does override the Edit (see page 10969) method to supply a default response when the user double clicks a component.

The Edit (see page 10969) method in TDefaultEditor searches the component for events. If it finds an OnCreate event, it brings up the code editor for the OnCreate event handler. If there is no OnCreate event, it brings up the code editor for the OnChange event handler. If there is no OnChange event, it brings up the code editor for the OnClick event handler. If it finds none of these events, it brings up the code editor for the first event it finds.

Use TDefaultEditor as a base class when creating component editors that bring up the code editor as the default response to a double click.

See Also

RegisterComponentEditor

TComponentEditor (see page 10927)

59.1.11.1 TDefaultEditor Members

The following tables list the members exposed by TDefaultEditor.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
V	EditProperty (see page 10970)	This is EditProperty, a member of class TDefaultEditor.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Copy (see page 10930)	Provides an opportunity to copy the component to the clipboard in custom formats.
	Copy (see page 11140)	Provides an opportunity to copy the component to the clipboard in custom formats.
V	Create (see page 10929)	Creates a new TBaseComponentEditor object.
V	Create (see page 11107)	Creates a new TBaseComponentEditor (see page 11105) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Edit (see page 10969)	Responds when the user double-clicks the component in the form designer.
V	ExecuteVerb (see page 10931)	Performs the action of a specified verb.
	ExecuteVerb (see page 11140)	This is ExecuteVerb, a member of class IComponentEditor.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetComponent (see page 10931)	Returns the value of the Component (see page 10934) property.

	GetComponent (see page 11141)	Returns a reference to the component on which the editor operates.
	GetDesigner (see page 10932)	Returns the interface for the form designer.
	GetDesigner (see page 11141)	Returns the interface for the form designer.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetVerb (see page 10932)	Returns the string that corresponds to a specified position of the component's context menu.
	GetVerb (see page 11141)	Returns the string that corresponds to a specified position of the component's context menu.
V	GetVerbCount (see page 10933)	Returns the number of menu strings added to the context menu by the component editor.
	GetVerbCount (see page 11142)	Returns the number of menu strings that the component editor adds to the context menu.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsInInlined (see page 10933)	Indicates whether the component has the csInline flag set in its ComponentState property.
	IsInInlined (see page 11142)	Indicates whether the component has the csInline flag set in its ComponentState property.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	PrepareItem (see page 10933)	Allows the component editor to modify menu items as they are added to the context menu.
	PrepareItem (see page 11142)	Allows the component editor to modify menu items as they are added to the context menu.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Component (see page 10934)	Provides access to the component in the form designer that uses this component editor.
R	Designer (see page 10935)	Provides access to the form designer in the IDE.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

59.1.11.2 TDefaultEditor Methods

The methods of the TDefaultEditor class are listed here.

Protected Methods

	Name	Description
V	EditProperty (see page 10970)	This is EditProperty, a member of class TDefaultEditor.

Public Methods

	Name	Description
V	Edit (see page 10969)	Responds when the user double-clicks the component in the form designer.

59.1.11.2.1 TDefaultEditor.Edit Method

Responds when the user double-clicks the component in the form designer.

Pascal

```
procedure Edit; override;
```

C++

```
virtual __fastcall Edit();
```

Description

The Object Inspector calls Edit when the user double-clicks the component. Edit searches the component for events. If it finds an OnCreate event, it brings up the code editor for the OnCreate event handler. If there is no OnCreate event, it brings up the code editor for the OnChange event handler. If there is no OnChange event, it brings up the code editor for the OnClick event handler. If it finds none of these events, it brings up the code editor for the first event it finds.

Note: Do not override Edit to change the event handler that your component editor displays in the code editor. Instead, override the protected EditProperty (see page 10970) method.

See Also

Component (see page 10934)

editproperty (see page 10970)

59.1.11.2.2 TDefaultEditor.EditProperty Method

Pascal

```
procedure EditProperty(const Prop: IProperty; var Continue: Boolean); virtual;
```

C++

```
virtual __fastcall EditProperty(const IProperty Prop, Boolean Continue);
```

Description

This is EditProperty, a member of class TDefaultEditor.

59.1.12 TDesignerOleDropTarget Class

Class Hierarchy

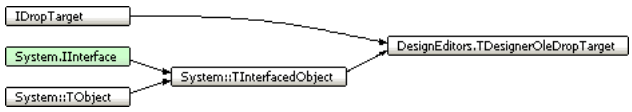

Pascal

```
TDesignerOleDropTarget = class(TInterfacedObject, IDropTarget);
```

C++

```
class TDesignerOleDropTarget : public TInterfacedObject, public IDropTarget;
```

File

DesignEditors

Description

This is class DesignEditors.TDesignerOleDropTarget.

59.1.12.1 TDesignerOleDropTarget Members

The following tables list the members exposed by TDesignerOleDropTarget.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	DragEnter (see page 10972)	This is DragEnter, a member of class TDesignerOleDropTarget.
	DragLeave (see page 10972)	This is DragLeave, a member of class TDesignerOleDropTarget.
	DragOver (see page 10973)	This is DragOver, a member of class TDesignerOleDropTarget.
	Drop (see page 10973)	This is Drop, a member of class TDesignerOleDropTarget.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 10972)	This is Create, a member of class TDesignerOleDropTarget.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

59.1.12.2 TDesignerOleDropTarget.Create Constructor

Pascal

```
constructor Create(const AOldDropTarget: IDropTarget = nil);
```

C++

```
__fastcall TDesignerOleDropTarget(const IDropTarget AOldDropTarget = nil);
```

Description

This is Create, a member of class TDesignerOleDropTarget.

59.1.12.3 TDesignerOleDropTarget Methods

The methods of the TDesignerOleDropTarget class are listed here.

Protected Methods

	Name	Description
	DragEnter (see page 10972)	This is DragEnter, a member of class TDesignerOleDropTarget.
	DragLeave (see page 10972)	This is DragLeave, a member of class TDesignerOleDropTarget.
	DragOver (see page 10973)	This is DragOver, a member of class TDesignerOleDropTarget.
	Drop (see page 10973)	This is Drop, a member of class TDesignerOleDropTarget.

59.1.12.3.1 TDesignerOleDropTarget.DragEnter Method

Pascal

```
function DragEnter(const dataObj: IDataObject; grfKeyState: Longint; pt: TPoint; var dwEffect:
Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT DragEnter(const IDataObject dataObj, Longint grfKeyState, TPoint
pt, Longint dwEffect);
```

Description

This is DragEnter, a member of class TDesignerOleDropTarget.

59.1.12.3.2 TDesignerOleDropTarget.DragLeave Method

Pascal

```
function DragLeave: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT DragLeave();
```

Description

This is DragLeave, a member of class TDesignerOleDropTarget.

59.1.12.3.3 TDesignerOleDropTarget.DragOver Method

Pascal

```
function DragOver(grfKeyState: Longint; pt: TPoint; var dwEffect: Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT DragOver(Longint grfKeyState, TPoint pt, Longint dwEffect);
```

Description

This is DragOver, a member of class TDesignerOleDropTarget.

59.1.12.3.4 TDesignerOleDropTarget.Drop Method

Pascal

```
function Drop(const dataObj: IDataObject; grfKeyState: Longint; pt: TPoint; var dwEffect:
Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Drop(const IDataObject dataObj, Longint grfKeyState, TPoint pt,
Longint dwEffect);
```

Description

This is Drop, a member of class TDesignerOleDropTarget.

59.1.13 TEditActionSelectionEditor Class

Class Hierarchy

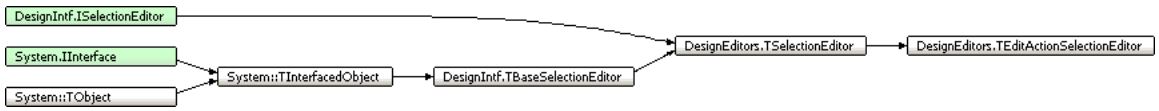

Pascal

```
TEditActionSelectionEditor = class(TSelectionEditor);
```

C++

```
class TEditActionSelectionEditor : public TSelectionEditor;
```

File

DesignEditors

Description

This is class DesignEditors.TEditActionSelectionEditor.

59.1.13.1 TEditActionSelectionEditor Members

The following tables list the members exposed by TEditActionSelectionEditor.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	EditAction (see page 10975)	This is EditAction, a member of class TEditActionSelectionEditor.
	GetEditState (see page 10976)	This is GetEditState, a member of class TEditActionSelectionEditor.
	HandleCopy (see page 10976)	This is HandleCopy, a member of class TEditActionSelectionEditor.
	HandleCut (see page 10977)	This is HandleCut, a member of class TEditActionSelectionEditor.
	HandleDelete (see page 10977)	This is HandleDelete, a member of class TEditActionSelectionEditor.
	HandlePaste (see page 10977)	This is HandlePaste, a member of class TEditActionSelectionEditor.
	HandleSelectAll (see page 10977)	This is HandleSelectAll, a member of class TEditActionSelectionEditor.
	HandleUndo (see page 10977)	This is HandleUndo, a member of class TEditActionSelectionEditor.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11047)	This is Create, a member of class TSelectionEditor.
V	Create (see page 11117)	This is Create, a member of class TBaseSelectionEditor.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	ExecuteVerb (see page 11048)	This is ExecuteVerb, a member of class TSelectionEditor.
	ExecuteVerb (see page 11214)	This is ExecuteVerb, a member of class ISelectionEditor.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetVerb (see page 10976)	This is GetVerb, a member of class TEditActionSelectionEditor.
V	GetVerbCount (see page 10976)	This is GetVerbCount, a member of class TEditActionSelectionEditor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	PrepareItem (see page 10978)	This is PrepareItem, a member of class TEditActionSelectionEditor.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	RequiresUnits (see page 11049)	This is RequiresUnits, a member of class TSelectionEditor.
	RequiresUnits (see page 11215)	This is RequiresUnits, a member of class ISelectionEditor.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Designer (see page 11049)	This is Designer, a member of class TSelectionEditor.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

59.1.13.2 TEditActionSelectionEditor Methods

The methods of the TEditActionSelectionEditor class are listed here.

Protected Methods

	Name	Description
	EditAction (see page 10975)	This is EditAction, a member of class TEditActionSelectionEditor.
	GetEditState (see page 10976)	This is GetEditState, a member of class TEditActionSelectionEditor.
	HandleCopy (see page 10976)	This is HandleCopy, a member of class TEditActionSelectionEditor.
	HandleCut (see page 10977)	This is HandleCut, a member of class TEditActionSelectionEditor.
	HandleDelete (see page 10977)	This is HandleDelete, a member of class TEditActionSelectionEditor.
	HandlePaste (see page 10977)	This is HandlePaste, a member of class TEditActionSelectionEditor.
	HandleSelectAll (see page 10977)	This is HandleSelectAll, a member of class TEditActionSelectionEditor.
	HandleUndo (see page 10977)	This is HandleUndo, a member of class TEditActionSelectionEditor.

Public Methods

	Name	Description
V	GetVerb (see page 10976)	This is GetVerb, a member of class TEditActionSelectionEditor.
V	GetVerbCount (see page 10976)	This is GetVerbCount, a member of class TEditActionSelectionEditor.
V	PrepareItem (see page 10978)	This is PrepareItem, a member of class TEditActionSelectionEditor.

59.1.13.2.1 TEditActionSelectionEditor.EditAction Method

Pascal

```
procedure EditAction(Action: TEditAction);
```

C++

```
__fastcall EditAction(TEditAction Action);
```

Description

This is EditAction, a member of class TEditActionSelectionEditor.

59.1.13.2.2 TEditActionSelectionEditor.GetEditState Method

Pascal

```
function GetEditState: TEditState;
```

C++

```
__fastcall TEditState GetEditState();
```

Description

This is GetEditState, a member of class TEditActionSelectionEditor.

59.1.13.2.3 TEditActionSelectionEditor.GetVerb Method

Pascal

```
function GetVerb(Index: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetVerb(int Index);
```

Description

This is GetVerb, a member of class TEditActionSelectionEditor.

59.1.13.2.4 TEditActionSelectionEditor.GetVerbCount Method

Pascal

```
function GetVerbCount: Integer; override;
```

C++

```
virtual __fastcall int GetVerbCount();
```

Description

This is GetVerbCount, a member of class TEditActionSelectionEditor.

59.1.13.2.5 TEditActionSelectionEditor.HandleCopy Method

Pascal

```
procedure HandleCopy(Sender: TObject);
```

C++

```
__fastcall HandleCopy(TObject * Sender);
```

Description

This is HandleCopy, a member of class TEditActionSelectionEditor.

59.1.13.2.6 TEditActionSelectionEditor.HandleCut Method

Pascal

```
procedure HandleCut(Sender: TObject);
```

C++

```
__fastcall HandleCut(TObject * Sender);
```

Description

This is HandleCut, a member of class TEditActionSelectionEditor.

59.1.13.2.7 TEditActionSelectionEditor.HandleDelete Method

Pascal

```
procedure HandleDelete(Sender: TObject);
```

C++

```
__fastcall HandleDelete(TObject * Sender);
```

Description

This is HandleDelete, a member of class TEditActionSelectionEditor.

59.1.13.2.8 TEditActionSelectionEditor.HandlePaste Method

Pascal

```
procedure HandlePaste(Sender: TObject);
```

C++

```
__fastcall HandlePaste(TObject * Sender);
```

Description

This is HandlePaste, a member of class TEditActionSelectionEditor.

59.1.13.2.9 TEditActionSelectionEditor.HandleSelectAll Method

Pascal

```
procedure HandleSelectAll(Sender: TObject);
```

C++

```
__fastcall HandleSelectAll(TObject * Sender);
```

Description

This is HandleSelectAll, a member of class TEditActionSelectionEditor.

59.1.13.2.10 TEditActionSelectionEditor.HandleUndo Method

Pascal

```
procedure HandleUndo(Sender: TObject);
```

C++

```
__fastcall HandleUndo(TObject * Sender);
```

Description

This is HandleUndo, a member of class TEditActionSelectionEditor.

59.1.13.2.11 TEditActionSelectionEditor.PrepareItem Method

Pascal

```
procedure PrepareItem(Index: Integer; const AItem: IMenuItem); override;
```

C++

```
virtual __fastcall PrepareItem(int Index, const IMenuItem AItem);
```

Description

This is PrepareItem, a member of class TEditActionSelectionEditor.

59.1.14 TEnumProperty Class

Class Hierarchy

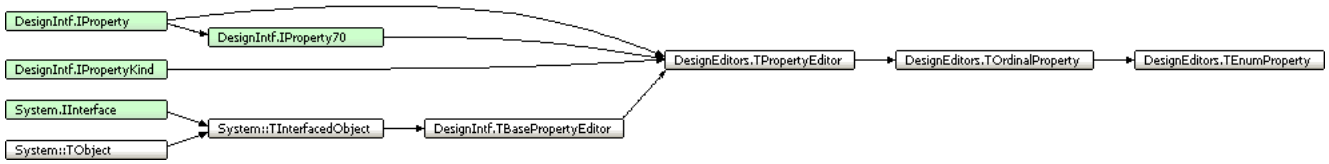

Pascal

```
TEnumProperty = class(TOrdinalProperty);
```

C++

```
class TEnumProperty : public TOrdinalProperty;
```

File

DesignEditors

Description

This is class DesignEditors.TEnumProperty.

59.1.14.1 TEnumProperty Members

The following tables list the members exposed by TEnumProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.

	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
A	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AllEqual (see page 11018)	Indicates whether all selected components have the same value for the property.
V	AllEqual (see page 11024)	Indicates whether all selected components have the same value for the property.
	AllEqual (see page 11199)	Indicates whether all selected components have the same value for the property.
V	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
V	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
V	GetAttributes (see page 10981)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
V	GetEditLimit (see page 11018)	Returns the maximum length, in characters, of the Value property.
V	GetEditLimit (see page 11027)	Returns the maximum length, in characters, of the Value (see page 11045) property.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
V	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
V	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
V	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
V	GetValue (see page 10982)	Returns the property value formatted as a string.
V	GetValues (see page 10982)	Provides the enumerated values of the property to a callback procedure.
V	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetValue (see page 10983)	Responds when the user changes the value of the property.
V	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.14.2 TEnumProperty Methods

The methods of the TEnumProperty class are listed here.

Public Methods

	Name	Description
V	GetAttributes (see page 10981)	Describes the property so the Object Inspector provides the appropriate controls.
V	GetValue (see page 10982)	Returns the property value formatted as a string.
V	GetValues (see page 10982)	Provides the enumerated values of the property to a callback procedure.
V	SetValue (see page 10983)	Responds when the user changes the value of the property.

59.1.14.2.1 TEnumProperty.GetAttributes Method

Describes the property so the Object Inspector provides the appropriate controls.

Pascal

```
function GetAttributes: TPropertyAttributes; override;
```

C++

```
virtual __fastcall TPropertyAttributes GetAttributes();
```

Description

The Object Inspector calls GetAttributes to determine what controls to display for the property and what property editor methods to call. The GetAttributes method of TPropertyEditor returns [paMultiSelect, paRevertable]. Override GetAttributes to return the set of values that corresponds to the attributes of the associated property.

Before the Object Inspector calls the Activate method, GetAttributes does not need to provide accurate information about any attributes other than paSubProperties, paVCL, and paMultiSelect.

See Also

Activate (see page 11024)

Edit (see page 11025)

GetProperties (see page 11033)

GetValues (see page 11035)

setvalue (see page 11042)

Value (see page 11045)

ListDrawValue

ListMeasureHeight

ListMeasureWidth

59.1.14.2.2 TEnumProperty.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raised an exception.

As implemented in TPropertyEditor, GetValue returns the string "(unknown)". Descendant classes override this method to provide an appropriate value string or raise an exception.

GetValue is the read access method of the Value property.

Note: To obtain an error string when the property value is unavailable rather than raising an exception, you can call the GetEditValue method. GetEditValue is protected, but can be called through the IProperty interface in code that is not a method of a descendant class.

See Also

setvalue (see page 11042)

Value (see page 11045)

GetVisualValue (see page 11037)

GetEditValue (see page 11027)

59.1.14.2.3 TEnumProperty.GetValues Method

Provides the enumerated values of the property to a callback procedure.

Pascal

```
procedure GetValues(Proc: TGetStrProc); override;
```

C++

```
virtual __fastcall GetValues(TGetStrProc Proc);
```

Description

The Object Inspector calls GetValues when the drop-down button is clicked to show enumerated values. The GetValues method of TPropertyEditor does nothing. Property editors that return paValueList from the GetAttributes method should override GetValues to call the Proc parameter for every string that can represent a valid enumerated value. Even if the GetAttributes method does not return paValueList, property editors for enumerated types should still override GetValues to enumerate all possible values.

See Also

GetAttributes (see page 11026)

Value (see page 11045)

59.1.14.2.4 TEnumProperty.SetValue Method

Responds when the user changes the value of the property.

Pascal

```
procedure SetValue(const Value: string); override;
```

C++

```
virtual __fastcall SetValue(const AnsiString Value);
```

Description

The Object Inspector calls SetValue when the user has finished changing the property value. SetValue applies the value specified by the Value parameter to all components that it is currently editing. If the value can't be set, SetValue raises an exception that describes the problem.

Note: If the property is changed only through a dialog brought up by the Edit method, the Object Inspector does not call SetValue. Rather, the Edit method should handle the setting of the value on all components.

See Also

Edit (see page 11200)

59.1.15 TFloatProperty Class

Class Hierarchy

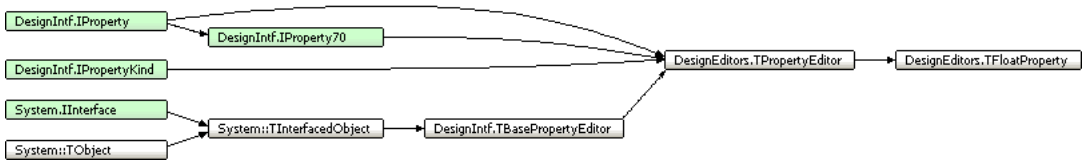

Pascal

```
TFloatProperty = class(TPropertyEditor);
```

C++

```
class TFloatProperty : public TPropertyEditor;
```

File

DesignEditors

Description

This is class DesignEditors.TFloatProperty.

59.1.15.1 TFloatProperty Members

The following tables list the members exposed by TFloatProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
A	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AllEqual (see page 10987)	Indicates whether all selected components have the same value for the property.
V	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
V	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
V	GetAttributes (see page 11026)	Describes the property so the Object Inspector provides the appropriate controls.
	GetAttributes (see page 11200)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
V	GetEditLimit (see page 11027)	Returns the maximum length, in characters, of the Value (see page 11045) property.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
V	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
V	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.

	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
V	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
V	GetValue (see page 10987)	Returns the property value formatted as a string.
V	GetValues (see page 11035)	Provides the enumerated values of the property to a callback procedure.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.
V	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetValue (see page 10988)	Responds when the user changes the value of the property.
V	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.15.2 TFloatProperty Methods

The methods of the TFloatProperty class are listed here.

Public Methods

	Name	Description
V	AllEqual (see page 10987)	Indicates whether all selected components have the same value for the property.
V	GetValue (see page 10987)	Returns the property value formatted as a string.
V	SetValue (see page 10988)	Responds when the user changes the value of the property.

59.1.15.2.1 TFloatProperty.AllEqual Method

Indicates whether all selected components have the same value for the property.

Pascal

```
function AllEqual: Boolean; override;
```

C++

```
virtual __fastcall Boolean AllEqual();
```

Description

The Object Inspector calls AllEqual when there is more than one component selected to determine whether a value should appear for this property. If this method returns true, the Object Inspector displays a value for the property. If this method returns false, the line for the property in the Object Inspector remains blank.

In TPropertyEditor, AllEqual always returns false if more than one selected component has the associated property. Override this method to check the properties of individual components and determine whether a property value can be displayed in the Object Inspector.

Note: AllEqual is called only when the value returned by GetAttributes includes paMultiSelect.

See Also

GetAttributes (see page 11026)

GetComponent (see page 11026)

GetValue (see page 11035)

59.1.15.2.2 TFloatProperty.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raised an exception.

As implemented in TPropertyEditor, GetValue returns the string "(unknown)". Descendant classes override this method to provide an appropriate value string or raise an exception.

GetValue is the read access method of the Value property.

Note: To obtain an error string when the property value is unavailable rather than raising an exception, you can call the GetEditValue method. GetEditValue is protected, but can be called through the IProperty interface in code that is not a method of a descendant class.

See Also

setvalue (see page 11042)

Value (see page 11045)

GetVisualValue (see page 11037)

GetEditValue (see page 11027)

59.1.15.2.3 TFloatProperty.SetValue Method

Responds when the user changes the value of the property.

Pascal

```
procedure SetValue(const Value: string); override;
```

C++

```
virtual __fastcall SetValue(const AnsiString Value);
```

Description

The Object Inspector calls SetValue when the user has finished changing the property value. SetValue applies the value specified by the Value parameter to all components that it is currently editing. If the value can't be set, SetValue raises an exception that describes the problem.

Note: If the property is changed only through a dialog brought up by the Edit method, the Object Inspector does not call SetValue. Rather, the Edit method should handle the setting of the value on all components.

See Also

Edit (see page 11200)

59.1.16 TInt64Property Class

Class Hierarchy

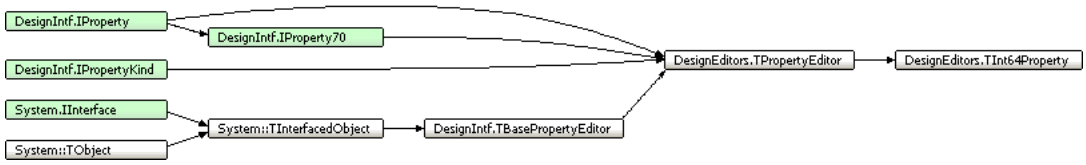

Pascal

```
TInt64Property = class(TPropertyEditor);
```

C++

```
class TInt64Property : public TPropertyEditor;
```

File

DesignEditors

Description

This is class DesignEditors.TInt64Property.

59.1.16.1 TInt64Property Members

The following tables list the members exposed by TInt64Property.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
A	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AllEqual (see page 10991)	Indicates whether all selected components have the same value for the property.

	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttributes (see page 11026)	Describes the property so the Object Inspector provides the appropriate controls.
	GetAttributes (see page 11200)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
	GetEditLimit (see page 10992)	Returns the maximum length, in characters, of the Value property.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
	GetValue (see page 10992)	Returns the property value formatted as a string.
	GetValues (see page 11035)	Provides the enumerated values of the property to a callback procedure.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.
	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetValue (see page 10993)	Responds when the user changes the value of the property.
V	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.16.2 TInt64Property Methods

The methods of the TInt64Property class are listed here.

Public Methods

	Name	Description
V	AllEqual (see page 10991)	Indicates whether all selected components have the same value for the property.
V	GetEditLimit (see page 10992)	Returns the maximum length, in characters, of the Value property.
V	GetValue (see page 10992)	Returns the property value formatted as a string.
V	SetValue (see page 10993)	Responds when the user changes the value of the property.

59.1.16.2.1 TInt64Property.AllEqual Method

Indicates whether all selected components have the same value for the property.

Pascal

```
function AllEqual: Boolean; override;
```

C++

```
virtual __fastcall Boolean AllEqual();
```

Description

The Object Inspector calls AllEqual when there is more than one component selected to determine whether a value should

appear for this property. If this method returns true, the Object Inspector displays a value for the property. If this method returns false, the line for the property in the Object Inspector remains blank.

In TPropertyEditor, AllEqual always returns false if more than one selected component has the associated property. Override this method to check the properties of individual components and determine whether a property value can be displayed in the Object Inspector.

Note: AllEqual is called only when the value returned by GetAttributes includes paMultiSelect.

See Also

GetAttributes (see page 11026)

GetComponent (see page 11026)

GetValue (see page 11035)

59.1.16.2.2 TInt64Property.GetEditLimit Method

Returns the maximum length, in characters, of the Value property.

Pascal

```
function GetEditLimit: Integer; override;
```

C++

```
virtual __fastcall int GetEditLimit();
```

Description

The Object Inspector calls GetEditLimit to determine how many characters the user can type when entering a value for the property. GetEditLimit returns 255. Override this method when the maximum length of the property string is some other value.

See Also

Value (see page 11045)

59.1.16.2.3 TInt64Property.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raised an exception.

As implemented in TPropertyEditor, GetValue returns the string "(unknown)". Descendant classes override this method to provide an appropriate value string or raise an exception.

GetValue is the read access method of the Value property.

Note: To obtain an error string when the property value is unavailable rather than raising an exception, you can call the GetEditValue method. GetEditValue is protected, but can be called through the IProperty interface in code that is not a method of a descendant class.

See Also

setvalue (see page 11042)

Value (see page 11045)

GetVisualValue (see page 11037)

GetEditValue (see page 11027)

59.1.16.2.4 TInt64Property.SetValue Method

Responds when the user changes the value of the property.

Pascal

```
procedure SetValue(const Value: string); override;
```

C++

```
virtual __fastcall SetValue(const AnsiString Value);
```

Description

The Object Inspector calls SetValue when the user has finished changing the property value. SetValue applies the value specified by the Value parameter to all components that it is currently editing. If the value can't be set, SetValue raises an exception that describes the problem.

Note: If the property is changed only through a dialog brought up by the Edit method, the Object Inspector does not call SetValue. Rather, the Edit method should handle the setting of the value on all components.

See Also

Edit (see page 11200)

59.1.17 TIntegerProperty Class

Class Hierarchy

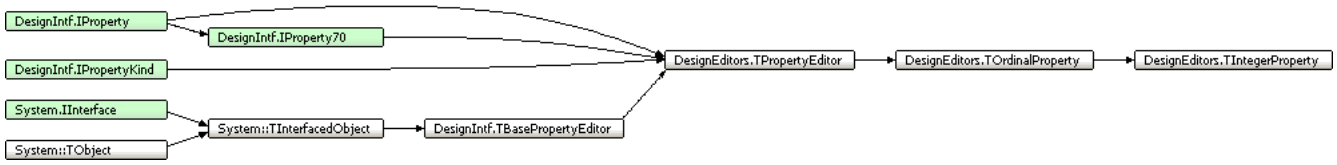

Pascal

```
TIntegerProperty = class(TOrdinalProperty);
```

C++

```
class TIntegerProperty : public TOrdinalProperty;
```

File

DesignEditors

Description

This is class DesignEditors.TIntegerProperty.

59.1.17.1 TIntegerProperty Members

The following tables list the members exposed by TIntegerProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
A	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.

	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
	AfterConstruction (see page 22279)	Responds after the last constructor executes.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AllEqual (see page 11018)	Indicates whether all selected components have the same value for the property.
	AllEqual (see page 11024)	Indicates whether all selected components have the same value for the property.
	AllEqual (see page 11199)	Indicates whether all selected components have the same value for the property.
	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttributes (see page 11026)	Describes the property so the Object Inspector provides the appropriate controls.
	GetAttributes (see page 11200)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
	GetEditLimit (see page 11018)	Returns the maximum length, in characters, of the Value property.
	GetEditLimit (see page 11027)	Returns the maximum length, in characters, of the Value (see page 11045) property.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.

	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
V	GetValue (see page 10996)	Returns the property value formatted as a string.
V	GetValues (see page 11035)	Provides the enumerated values of the property to a callback procedure.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.
V	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetValue (see page 10997)	Responds when the user changes the value of the property.
V	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.17.2 TIntegerProperty Methods

The methods of the TIntegerProperty class are listed here.

Public Methods

	Name	Description
V	GetValue (see page 10996)	Returns the property value formatted as a string.
V	SetValue (see page 10997)	Responds when the user changes the value of the property.

59.1.17.2.1 TIntegerProperty.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raised an exception.

As implemented in TPropertyEditor, GetValue returns the string "(unknown)". Descendant classes override this method to provide an appropriate value string or raise an exception.

GetValue is the read access method of the Value property.

Note: To obtain an error string when the property value is unavailable rather than raising an exception, you can call the GetEditValue method. GetEditValue is protected, but can be called through the IProperty interface in code that is not a method of a descendant class.

See Also

setvalue (see page 11042)

Value (see page 11045)

GetVisualValue (see page 11037)

GetEditValue (see page 11027)

59.1.17.2.2 TIntegerProperty.SetValue Method

Responds when the user changes the value of the property.

Pascal

```
procedure SetValue(const Value: string); override;
```

C++

```
virtual __fastcall SetValue(const AnsiString Value);
```

Description

The Object Inspector calls SetValue when the user has finished changing the property value. SetValue applies the value specified by the Value parameter to all components that it is currently editing. If the value can't be set, SetValue raises an exception that describes the problem.

Note: If the property is changed only through a dialog brought up by the Edit method, the Object Inspector does not call SetValue. Rather, the Edit method should handle the setting of the value on all components.

See Also

Edit (see page 11200)

59.1.18 TInterfaceProperty Class

Class Hierarchy

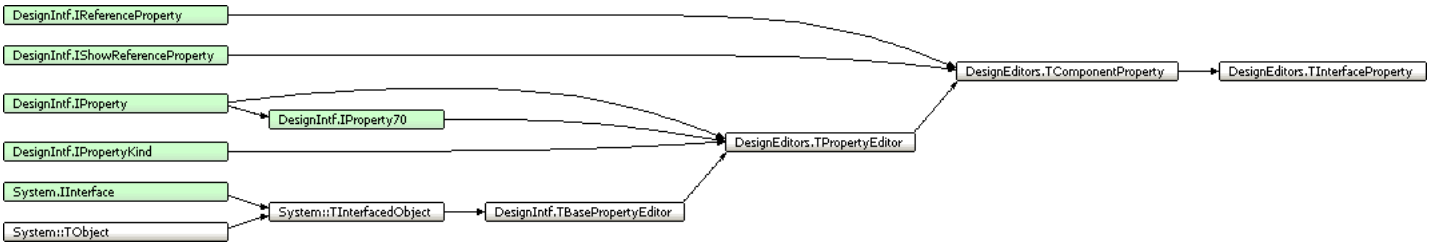

Pascal

```
TInterfaceProperty = class(TComponentProperty);
```

C++

```
class TInterfaceProperty : public TComponentProperty;
```

File

DesignEditors

Description

This is class DesignEditors.TInterfaceProperty.

59.1.18.1 TInterfaceProperty Members

The following tables list the members exposed by TInterfaceProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	FilterFunc (see page 10948)	This is FilterFunc, a member of class TComponentProperty.
	GetComponent (see page 11002)	This is GetComponent, a member of class TInterfaceProperty.
V	GetComponentReference (see page 11002)	This is GetComponentReference, a member of class TInterfaceProperty.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.

	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
V	GetSelections (see page 11002)	This is GetSelections, a member of class TInterfaceProperty.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	ReceiveComponentNames (see page 11003)	This is ReceiveComponentNames, a member of class TInterfaceProperty.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
A	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.
V	ShowReferenceProperty (see page 10951)	This is ShowReferenceProperty, a member of class TComponentProperty.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AllEqual (see page 11001)	Indicates whether all selected components have the same value for the property.
V	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
V	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 10948)	Responds when the user double-clicks the property in the Object Inspector.
	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttributes (see page 10948)	Describes the property so the Object Inspector provides the appropriate controls.
	GetAttributes (see page 11026)	Describes the property so the Object Inspector provides the appropriate controls.
	GetAttributes (see page 11200)	Describes the property so the Object Inspector provides the appropriate controls.
	GetEditLimit (see page 10949)	Returns the maximum length, in characters, of the Value property.
	GetEditLimit (see page 11027)	Returns the maximum length, in characters, of the Value (see page 11045) property.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
	GetProperties (see page 10949)	Provides the subproperties of the property to a callback procedure.
	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
	GetValue (see page 10950)	Returns the property value formatted as a string.
	GetValue (see page 11035)	Returns the property value formatted as a string.
	GetValue (see page 11203)	Returns the property value formatted as a string.
	GetValues (see page 11002)	Provides the enumerated values of the property to a callback procedure.
	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.

	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetValue (see page 11003)	Responds when the user changes the value of the property.
	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ShowReferenceProperty (see page 11219)	This is ShowReferenceProperty, a member of class IShowReferenceProperty.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
	Designer (see page 11044)	Provides access to the form designer object in the IDE.
	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.18.2 TInterfaceProperty Methods

The methods of the TInterfaceProperty class are listed here.

Protected Methods

	Name	Description
	GetComponent (see page 11002)	This is GetComponent, a member of class TInterfaceProperty.
	GetComponentReference (see page 11002)	This is GetComponentReference, a member of class TInterfaceProperty.
	GetSelections (see page 11002)	This is GetSelections, a member of class TInterfaceProperty.
	ReceiveComponentNames (see page 11003)	This is ReceiveComponentNames, a member of class TInterfaceProperty.

Public Methods

	Name	Description
	AllEqual (see page 11001)	Indicates whether all selected components have the same value for the property.
	GetValues (see page 11002)	Provides the enumerated values of the property to a callback procedure.
	SetValue (see page 11003)	Responds when the user changes the value of the property.

59.1.18.2.1 TInterfaceProperty.AllEqual Method

Indicates whether all selected components have the same value for the property.

Pascal

```
function AllEqual: Boolean; override;
```

C++

```
virtual __fastcall Boolean AllEqual();
```

Description

The Object Inspector calls AllEqual when there is more than one component selected to determine whether a value should appear for this property. If this method returns true, the Object Inspector displays a value for the property. If this method returns false, the line for the property in the Object Inspector remains blank.

In TPropertyEditor, AllEqual always returns false if more than one selected component has the associated property. Override this method to check the properties of individual components and determine whether a property value can be displayed in the Object Inspector.

Note: AllEqual is called only when the value returned by GetAttributes includes paMultiSelect.

See Also

GetAttributes (see page 11026)

GetComponent (see page 11026)

GetValue (see page 11035)

59.1.18.2.2 TInterfaceProperty.GetComponent Method

Pascal

```
function GetComponent(const AInterface: IInterface): TComponent;
```

C++

```
__fastcall TComponent * GetComponent(const IInterface AInterface);
```

Description

This is GetComponent, a member of class TInterfaceProperty.

59.1.18.2.3 TInterfaceProperty.GetComponentReference Method

Pascal

```
function GetComponentReference: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetComponentReference();
```

Description

This is GetComponentReference, a member of class TInterfaceProperty.

59.1.18.2.4 TInterfaceProperty.GetSelections Method

Pascal

```
function GetSelections: IDesignerSelections; override;
```

C++

```
virtual __fastcall IDesignerSelections GetSelections();
```

Description

This is GetSelections, a member of class TInterfaceProperty.

59.1.18.2.5 TInterfaceProperty.GetValues Method

Provides the enumerated values of the property to a callback procedure.

Pascal

```
procedure GetValues(Proc: TGetStrProc); override;
```

C++

```
virtual __fastcall GetValues(TGetStrProc Proc);
```

Description

The Object Inspector calls GetValues when the drop-down button is clicked to show enumerated values. The GetValues method of TPropertyEditor does nothing. Property editors that return paValueList from the GetAttributes method should override GetValues to call the Proc parameter for every string that can represent a valid enumerated value. Even if the GetAttributes method does not return paValueList, property editors for enumerated types should still override GetValues to enumerate all possible values.

See Also

GetAttributes (see page 11026)

Value (see page 11045)

59.1.18.2.6 TInterfaceProperty.ReceiveComponentNames Method

Pascal

```
procedure ReceiveComponentNames(const S: string);
```

C++

```
__fastcall ReceiveComponentNames(const AnsiString S);
```

Description

This is ReceiveComponentNames, a member of class TInterfaceProperty.

59.1.18.2.7 TInterfaceProperty.SetValue Method

Responds when the user changes the value of the property.

Pascal

```
procedure SetValue(const Value: string); override;
```

C++

```
virtual __fastcall SetValue(const AnsiString Value);
```

Description

The Object Inspector calls SetValue when the user has finished changing the property value. SetValue applies the value specified by the Value parameter to all components that it is currently editing. If the value can't be set, SetValue raises an exception that describes the problem.

Note: If the property is changed only through a dialog brought up by the Edit method, the Object Inspector does not call SetValue. Rather, the Edit method should handle the setting of the value on all components.

See Also

Edit (see page 11200)

59.1.19 TMethodProperty Class

Class Hierarchy

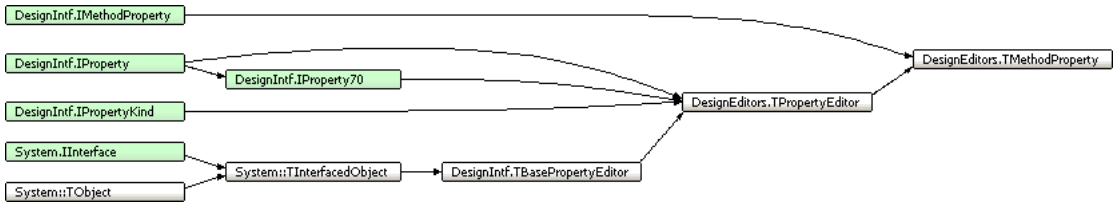

Pascal

```
TMethodProperty = class(TPropertyEditor, IMethodProperty);
```

C++

```
class TMethodProperty : public TPropertyEditor, public IMethodProperty;
```

File

DesignEditors

Description

This is class DesignEditors.TMethodProperty.

59.1.19.1 TMethodProperty Members

The following tables list the members exposed by TMethodProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.

	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
	AfterConstruction (see page 22279)	Responds after the last constructor executes.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AllEqual (see page 11007)	Indicates whether all selected components have the same value for the property.
	AllNamed (see page 11008)	This is AllNamed, a member of class TMethodProperty.
	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 11008)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
V	GetAttributes (see page 11008)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
V	GetEditLimit (see page 11009)	Returns the maximum length, in characters, of the Value property.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
V	GetFormMethodName (see page 11009)	This is GetFormMethodName, a member of class TMethodProperty.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
V	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
V	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
V	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
	GetTrimmedEventName (see page 11009)	This is GetTrimmedEventName, a member of class TMethodProperty.
V	GetValue (see page 11010)	Returns the property value formatted as a string.
V	GetValues (see page 11010)	Provides the enumerated values of the property to a callback procedure.
V	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetValue (see page 11011)	Responds when the user changes the value of the property.
V	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.

R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.19.2 TMethodProperty Methods

The methods of the TMethodProperty class are listed here.

Public Methods

	Name	Description
V	AllEqual (see page 11007)	Indicates whether all selected components have the same value for the property.
V	AllNamed (see page 11008)	This is AllNamed, a member of class TMethodProperty.
V	Edit (see page 11008)	Responds when the user double-clicks the property in the Object Inspector.
V	GetAttributes (see page 11008)	Describes the property so the Object Inspector provides the appropriate controls.
V	GetEditLimit (see page 11009)	Returns the maximum length, in characters, of the Value property.
V	GetFormMethodName (see page 11009)	This is GetFormMethodName, a member of class TMethodProperty.
	GetTrimmedEventName (see page 11009)	This is GetTrimmedEventName, a member of class TMethodProperty.
V	GetValue (see page 11010)	Returns the property value formatted as a string.
V	GetValues (see page 11010)	Provides the enumerated values of the property to a callback procedure.
V	SetValue (see page 11011)	Responds when the user changes the value of the property.

59.1.19.2.1 TMethodProperty.AllEqual Method

Indicates whether all selected components have the same value for the property.

Pascal

```
function AllEqual: Boolean; override;
```

C++

```
virtual __fastcall Boolean AllEqual();
```

Description

The Object Inspector calls AllEqual when there is more than one component selected to determine whether a value should appear for this property. If this method returns true, the Object Inspector displays a value for the property. If this method returns false, the line for the property in the Object Inspector remains blank.

In TPropertyEditor, AllEqual always returns false if more than one selected component has the associated property. Override this method to check the properties of individual components and determine whether a property value can be displayed in the Object Inspector.

Note: AllEqual is called only when the value returned by GetAttributes includes paMultiSelect.

See Also

GetAttributes (see page 11026)

GetComponent (see page 11026)

GetValue (see page 11035)

59.1.19.2.2 TMethodProperty.AllNamed Method

Pascal

```
function AllNamed: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean AllNamed();
```

Description

This is AllNamed, a member of class TMethodProperty.

59.1.19.2.3 TMethodProperty.Edit Method

Responds when the user double-clicks the property in the Object Inspector.

Pascal

```
procedure Edit; override;
```

C++

```
virtual __fastcall Edit();
```

Description

The Object Inspector calls Edit when the ellipsis button (...) is pressed or the property is double-clicked. The Edit method of TPropertyEditor cycles through the possible enumerated values of the property when the GetValues method returns one or more property strings to the callback method.

Override the Edit method to implement a response to the ellipsis button or double-clicking the property. For example, use the Edit method to bring up a dialog for editing the property in some more meaningful fashion than typing a text value in the Object Inspector.

See Also

GetValues (see page 11035)

Value (see page 11045)

59.1.19.2.4 TMethodProperty.GetAttributes Method

Describes the property so the Object Inspector provides the appropriate controls.

Pascal

```
function GetAttributes: TPropertyAttributes; override;
```

C++

```
virtual __fastcall TPropertyAttributes GetAttributes();
```

Description

The Object Inspector calls GetAttributes to determine what controls to display for the property and what property editor methods to call. The GetAttributes method of TPropertyEditor returns [paMultiSelect, paRevertable]. Override GetAttributes to return the set of values that corresponds to the attributes of the associated property.

Before the Object Inspector calls the Activate method, GetAttributes does not need to provide accurate information about any attributes other than paSubProperties, paVCL, and paMultiSelect.

See Also

Activate (see page 11024)

Edit (see page 11025)

GetProperties (see page 11033)

GetValues (see page 11035)

setvalue (see page 11042)

Value (see page 11045)

ListDrawValue

ListMeasureHeight

ListMeasureWidth

59.1.19.2.5 TMethodProperty.GetEditLimit Method

Returns the maximum length, in characters, of the Value property.

Pascal

```
function GetEditLimit: Integer; override;
```

C++

```
virtual __fastcall int GetEditLimit();
```

Description

The Object Inspector calls GetEditLimit to determine how many characters the user can type when entering a value for the property. GetEditLimit returns 255. Override this method when the maximum length of the property string is some other value.

See Also

Value (see page 11045)

59.1.19.2.6 TMethodProperty.GetFormMethodName Method

Pascal

```
function GetFormMethodName: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetFormMethodName();
```

Description

This is GetFormMethodName, a member of class TMethodProperty.

59.1.19.2.7 TMethodProperty.GetTrimmedEventName Method

Pascal

```
function GetTrimmedEventName: string;
```

C++

```
__fastcall AnsiString GetTrimmedEventName();
```

Description

This is GetTrimmedEventName, a member of class TMethodProperty.

59.1.19.2.8 TMethodProperty.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raised an exception.

As implemented in TPropertyEditor, GetValue returns the string "(unknown)". Descendant classes override this method to provide an appropriate value string or raise an exception.

GetValue is the read access method of the Value property.

Note: To obtain an error string when the property value is unavailable rather than raising an exception, you can call the GetEditValue method. GetEditValue is protected, but can be called through the IProperty interface in code that is not a method of a descendant class.

See Also

setvalue (see page 11042)

Value (see page 11045)

GetVisualValue (see page 11037)

GetEditValue (see page 11027)

59.1.19.2.9 TMethodProperty.GetValues Method

Provides the enumerated values of the property to a callback procedure.

Pascal

```
procedure GetValues(Proc: TGetStrProc); override;
```

C++

```
virtual __fastcall GetValues(TGetStrProc Proc);
```

Description

The Object Inspector calls GetValues when the drop-down button is clicked to show enumerated values. The GetValues method of TPropertyEditor does nothing. Property editors that return paValueList from the GetAttributes method should override GetValues to call the Proc parameter for every string that can represent a valid enumerated value. Even if the GetAttributes method does not return paValueList, property editors for enumerated types should still override GetValues to enumerate all possible values.

See Also

GetAttributes (see page 11026)

Value (see page 11045)

59.1.19.2.10 TMethodProperty.SetValue Method

Responds when the user changes the value of the property.

Pascal

```
procedure SetValue(const AValue: string); override;
```

C++

```
virtual __fastcall SetValue(const AnsiString AValue);
```

Description

The Object Inspector calls SetValue when the user has finished changing the property value. SetValue applies the value specified by the Value parameter to all components that it is currently editing. If the value can't be set, SetValue raises an exception that describes the problem.

Note: If the property is changed only through a dialog brought up by the Edit method, the Object Inspector does not call SetValue. Rather, the Edit method should handle the setting of the value on all components.

See Also

Edit (see page 11200)

59.1.20 TNestedProperty Class

Class Hierarchy

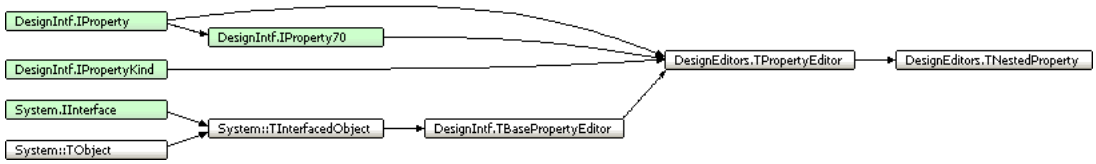

Pascal

```
TNestedProperty = class(TPropertyEditor);
```

C++

```
class TNestedProperty : public TPropertyEditor;
```

File

DesignEditors

Description

This is class DesignEditors.TNestedProperty.

59.1.20.1 TNestedProperty Members

The following tables list the members exposed by TNestedProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
A	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AllEqual (see page 11024)	Indicates whether all selected components have the same value for the property.
	AllEqual (see page 11199)	Indicates whether all selected components have the same value for the property.
V	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.

	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11014)	This is Create, a member of class TNestedProperty.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 11014)	Destroys an instance of TPropertyEditor.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttributes (see page 11026)	Describes the property so the Object Inspector provides the appropriate controls.
	GetAttributes (see page 11200)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
	GetEditLimit (see page 11027)	Returns the maximum length, in characters, of the Value (see page 11045) property.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
	GetValue (see page 11035)	Returns the property value formatted as a string.
	GetValue (see page 11203)	Returns the property value formatted as a string.
	GetValues (see page 11035)	Provides the enumerated values of the property to a callback procedure.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.
	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetValue (see page 11204)	Responds when the user changes the value of the property.
V	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.20.2 TNestedProperty.Create Constructor

Pascal

```
constructor Create(Parent: TPropertyEditor); reintroduce;
```

C++

```
__fastcall TNestedProperty(TPropertyEditor Parent);
```

Description

This is Create, a member of class TNestedProperty.

59.1.20.3 TNestedProperty.Destroy Destructor

Destroys an instance of TPropertyEditor.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TNestedProperty();
```

Description

Do not call Destroy. The Object Inspector destroys the property editor automatically when it is finished.

Destroy frees the internal objects used to store information about the selected components and their property settings. Override Destroy to free any other internal structures created by the property editor.

See Also

Free (see page 22300)

59.1.21 TOrdinalProperty Class

Class Hierarchy

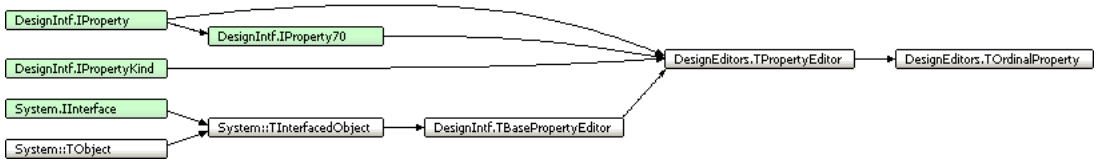

Pascal

```
TOrdinalProperty = class(TPropertyEditor);
```

C++

```
class TOrdinalProperty : public TPropertyEditor;
```

File

DesignEditors

Description

This is class DesignEditors.TOrdinalProperty.

59.1.21.1 TOrdinalProperty Members

The following tables list the members exposed by TOrdinalProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.

	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
A	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AllEqual (see page 11018)	Indicates whether all selected components have the same value for the property.
V	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
V	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.

	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
V	GetAttributes (see page 11026)	Describes the property so the Object Inspector provides the appropriate controls.
	GetAttributes (see page 11200)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
V	GetEditLimit (see page 11018)	Returns the maximum length, in characters, of the Value property.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
V	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
V	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
V	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
V	GetValue (see page 11035)	Returns the property value formatted as a string.
	GetValue (see page 11203)	Returns the property value formatted as a string.
V	GetValues (see page 11035)	Provides the enumerated values of the property to a callback procedure.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.
V	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetValue (see page 11204)	Responds when the user changes the value of the property.
V	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.21.2 TOrdinalProperty Methods

The methods of the TOrdinalProperty class are listed here.

Public Methods

	Name	Description
V	AllEqual (see page 11018)	Indicates whether all selected components have the same value for the property.
V	GetEditLimit (see page 11018)	Returns the maximum length, in characters, of the Value property.

59.1.21.2.1 TOrdinalProperty.AllEqual Method

Indicates whether all selected components have the same value for the property.

Pascal

```
function AllEqual: Boolean; override;
```

C++

```
virtual __fastcall Boolean AllEqual();
```

Description

The Object Inspector calls AllEqual when there is more than one component selected to determine whether a value should appear for this property. If this method returns true, the Object Inspector displays a value for the property. If this method returns false, the line for the property in the Object Inspector remains blank.

In TPropertyEditor, AllEqual always returns false if more than one selected component has the associated property. Override this method to check the properties of individual components and determine whether a property value can be displayed in the Object Inspector.

Note: AllEqual is called only when the value returned by GetAttributes includes paMultiSelect.

See Also

GetAttributes (see page 11026)

GetComponent (see page 11026)

GetValue (see page 11035)

59.1.21.2.2 TOrdinalProperty.GetEditLimit Method

Returns the maximum length, in characters, of the Value property.

Pascal

```
function GetEditLimit: Integer; override;
```

C++

```
virtual __fastcall int GetEditLimit();
```

Description

The Object Inspector calls GetEditLimit to determine how many characters the user can type when entering a value for the property. GetEditLimit returns 255. Override this method when the maximum length of the property string is some other value.

See Also

Value (see page 11045)

59.1.22 TPropertyEditor Class

TPropertyEditor is a base class for property editors that implements the IProperty interface.

Class Hierarchy

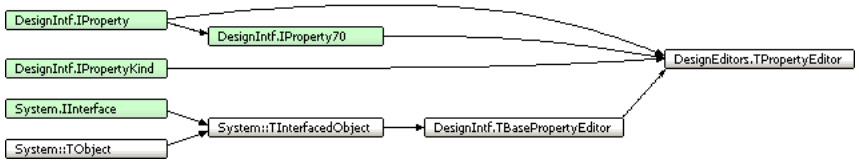

Pascal

```
TPropertyEditor = class(TBasePropertyEditor, IProperty, IPropertyKind, IProperty70);
```

C++

```
class TPropertyEditor : public TBasePropertyEditor, public IProperty, public IPropertyKind,
public IProperty70;
```

File

DesignEditors

Description

Use TPropertyEditor or one of its descendants as a base class when defining a property editor for a specialized property in a new component class. TPropertyEditor has several descendants that can be used to set the values of properties with simple types. Create (see page 11022) specialized property editors to allow users to set more complicated property values from the Object Inspector.

Unlike its ancestor, TBasePropertyEditor, TPropertyEditor contains no abstract (or, in C++ terminology, pure virtual) methods and provides a default implementation for the IProperty interface that the Object Inspector uses to communicate with property editors.

Individual descendants of TPropertyEditor are displayed based on the type of the property being edited. Individual property editors can be associated with all properties of a specified type, or with a particular property of a particular component class. To create the association between a property editor and a property type, call RegisterPropertyEditor. Once the property editor is registered, it appears when the user attempts to edit an appropriate property using the Object Inspector.

See Also

RegisterPropertyEditor

IProperty (see page 11197)

59.1.22.1 TPropertyEditor Members

The following tables list the members exposed by TPropertyEditor.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	AllEqual (see page 11024)	Indicates whether all selected components have the same value for the property.
	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttributes (see page 11026)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
	GetEditLimit (see page 11027)	Returns the maximum length, in characters, of the Value (see page 11045) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetName (see page 11032)	Returns the name of the property.
	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.
	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetValue (see page 11035)	Returns the property value formatted as a string.
	GetValues (see page 11035)	Provides the enumerated values of the property to a callback procedure.
	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.22.2 TPropertyEditor.Create Constructor

Creates a new TPropertyEditor (see page 11019) object.

Pascal

```
constructor Create(const ADesigner: IDesigner; APropCount: Integer); override;
```

C++

```
virtual __fastcall TPropertyEditor(const IDesigner ADesigner, int APropCount);
```

Description

Do not create instances of a property editor. Registered property editors are created automatically for selected objects in the form designer.

ADesigner is an interface that the property editor can use to interact with the forms designer. It becomes the value of the Designer (see page 11044) property.

APropCount indicates the number of components whose properties are to be edited using the property editor. It becomes the value of the PropCount (see page 11045) property.

See Also

Initialize (see page 11038)

Activate (see page 11024)

Designer (see page 11044)

PropCount (see page 11045)

59.1.22.3 TPropertyEditor.Destroy Destructor

Destroys an instance of TPropertyEditor (see page 11019).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPropertyEditor();
```

Description

Do not call Destroy. The Object Inspector destroys the property editor automatically when it is finished.

Destroy frees the internal objects used to store information about the selected components and their property settings. Override Destroy to free any other internal structures created by the property editor.

See Also

Free (see page 22300)

59.1.22.4 TPropertyEditor Methods

The methods of the TPropertyEditor class are listed here.

Protected Methods

	Name	Description
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
V	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
V	AllEqual (see page 11024)	Indicates whether all selected components have the same value for the property.
V	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
V	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
V	GetAttributes (see page 11026)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
V	GetEditLimit (see page 11027)	Returns the maximum length, in characters, of the Value (see page 11045) property.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
V	GetName (see page 11032)	Returns the name of the property.

	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.
	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetValue (see page 11035)	Returns the property value formatted as a string.
	GetValues (see page 11035)	Provides the enumerated values of the property to a callback procedure.
	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	Revert (see page 11039)	Restores all object properties to their original values.
	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

59.1.22.4.1 TPropertyEditor.Activate Method

Responds when the property is selected in the Object Inspector.

Pascal

```
procedure Activate; virtual;
```

C++

```
virtual __fastcall Activate();
```

Description

Do not call the Activate method. It is called automatically when the property is selected. As implemented in TPropertyEditor (see page 11019), Activate does nothing. Override this method to determine attributes of the property dynamically when the property editor is displayed in the Object Inspector.

Before Activate is called, the GetAttributes (see page 11026) method does not need to provide accurate information about any attributes other than paSubProperties, paVCL, and paMultiSelect. Any other attributes can be computed in an override of the Activate method and stored so that subsequent calls to GetAttributes (see page 11026) can provide accurate information.

See Also

GetAttributes (see page 11026)

Initialize (see page 11038)

59.1.22.4.2 TPropertyEditor.AllEqual Method

Indicates whether all selected components have the same value for the property.

Pascal

```
function AllEqual: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean AllEqual();
```

Description

The Object Inspector calls AllEqual when there is more than one component selected to determine whether a value should appear for this property. If this method returns true, the Object Inspector displays a value for the property. If this method returns false, the line for the property in the Object Inspector remains blank.

In TPropertyEditor (see page 11019), AllEqual always returns false if more than one selected component has the associated property. Override this method to check the properties of individual components and determine whether a property value can be displayed in the Object Inspector.

Note: AllEqual is called only when the value returned by GetAttributes (see page 11026) includes paMultiSelect.

See Also

GetAttributes (see page 11026)

GetComponent (see page 11026)

GetValue (see page 11035)

59.1.22.4.3 TPropertyEditor.AutoFill Method

Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.

Pascal

```
function AutoFill: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean AutoFill();
```

Description

If GetAttributes returns paValueList, the Object Inspector calls AutoFill to verify that automatic typing completion should be permitted. If AutoFill returns true, the Object Inspector uses the values provided by GetValues to fill in partially typed entries. For example, if the legal values for a property are prSmall and prLarge, the user can type "prS" and the Object Inspector automatically fills in "prSmall."

See Also

GetAttributes (see page 11200)

GetValues (see page 11203)

59.1.22.4.4 TPropertyEditor.Edit Method

Responds when the user double-clicks the property in the Object Inspector.

Pascal

```
procedure Edit; virtual;
```

C++

```
virtual __fastcall Edit();
```

Description

The Object Inspector calls Edit when the ellipsis button (...) is pressed or the property is double-clicked. The Edit method of TPropertyEditor (see page 11019) cycles through the possible enumerated values of the property when the GetValues (see page 11035) method returns one or more property strings to the callback method.

Override the Edit method to implement a response to the ellipsis button or double-clicking the property. For example, use the Edit method to bring up a dialog for editing the property in some more meaningful fashion than typing a text value in the Object Inspector.

See Also

GetValues (see page 11035)

Value (see page 11045)

59.1.22.4.5 TPropertyEditor.GetAttributes Method

Describes the property so the Object Inspector provides the appropriate controls.

Pascal

```
function GetAttributes: TPropertyAttributes; virtual;
```

C++

```
virtual __fastcall TPropertyAttributes GetAttributes();
```

Description

The Object Inspector calls GetAttributes to determine what controls to display for the property and what property editor methods to call. The GetAttributes method of TPropertyEditor (see page 11019) returns [paMultiSelect, paRevertable]. Override GetAttributes to return the set of values that corresponds to the attributes of the associated property.

Before the Object Inspector calls the Activate (see page 11024) method, GetAttributes does not need to provide accurate information about any attributes other than paSubProperties, paVCL, and paMultiSelect.

See Also

Activate (see page 11024)

Edit (see page 11025)

GetProperties (see page 11033)

GetValues (see page 11035)

setvalue (see page 11042)

Value (see page 11045)

ListDrawValue

ListMeasureHeight

ListMeasureWidth

59.1.22.4.6 TPropertyEditor.GetComponent Method

Returns the specified object from the set of all objects currently being edited by the property editor.

Pascal

```
function GetComponent(Index: Integer): TPersistent;
```

C++

```
__fastcall TPersistent * GetComponent(int Index);
```

Description

Call GetComponent to access any of the objects with a property currently being edited by the property editor. The Index parameter is used to specify individual objects in the internal list maintained by the property editor. Index should be a value between 0 and PropCount (see page 11045) - 1, inclusive.

Use GetComponent to iterate over all objects currently being edited.

See Also

PropCount (see page 11045)

HasInstance (see page 11038)

59.1.22.4.7 TPropertyEditor.GetEditLimit Method

Returns the maximum length, in characters, of the Value (see page 11045) property.

Pascal

```
function GetEditLimit: Integer; virtual;
```

C++

```
virtual __fastcall int GetEditLimit();
```

Description

The Object Inspector calls GetEditLimit to determine how many characters the user can type when entering a value for the property. GetEditLimit returns 255. Override this method when the maximum length of the property string is some other value.

See Also

Value (see page 11045)

59.1.22.4.8 GetEditValue Method

This is the overview for the GetEditValue method overload.

Overload List

	Name	Description
	TPropertyEditor.GetEditValue (string) (see page 11027)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	TPropertyEditor.GetEditValue (WideString) (see page 11027)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.

59.1.22.4.8.1 TPropertyEditor.GetEditValue Method (WideString)

Retrieves a string to represent the property's value, or, if the value is not available, an error message.

Pascal

```
function GetEditValue(out Value: string): Boolean; overload;
function GetEditValue(out Value: WideString): Boolean; overload;
```

C++

```
__fastcall Boolean GetEditValue(AnsiString Value);
__fastcall Boolean GetEditValue(BSTR Value);
```

Description

The Object Inspector calls GetEditValue (through the IProperty interface) to obtain the string it displays as the property value.

GetValue (see page 11035) formats the string displayed in the Object Inspector to represent the property value and returns it as the Value (see page 11045) parameter. Unlike the GetValue (see page 11035) method, which is the read access method for the Value (see page 11045) property, GetEditValue does not raise an exception when the property value is unavailable. Instead, it assigns an error message to Value (see page 11045) and returns false.

GetValue (see page 11035) returns true if Value (see page 11045) returns a string representation of the property value.

Note: GetEditValue is not a virtual method. To provide a method that supplies the string that appears in the Object Inspector, override the GetValue (see page 11035) method instead. GetEditValue calls GetValue (see page 11035) to fetch this string.

See Also

GetValue (see page 11035)

59.1.22.4.9 TPropertyEditor.GetFloatValue Method

Returns the property value of the first object currently being edited.

Pascal

```
function GetFloatValue: Extended;
```

C++

```
__fastcall Extended GetFloatValue();
```

Description

Call GetFloatValue to obtain the current value of the property for the first object in the internal list of objects being edited. Call GetFloatValue only if the property is a float type.

If more than one object is currently being edited, GetFloatValue returns the value for the object returned by GetComponent (see page 11026)(0). To obtain the current property value for every object currently being edited, use the GetFloatValueAt (see page 11028) method.

See Also

GetComponent (see page 11026)

GetFloatValueAt (see page 11028)

59.1.22.4.10 TPropertyEditor.GetFloatValueAt Method

Returns the property value of the specified object.

Pascal

```
function GetFloatValueAt(Index: Integer): Extended;
```

C++

```
__fastcall Extended GetFloatValueAt(int Index);
```

Description

Call GetFloatValueAt to obtain the current value of the property for an object in the internal list of objects being edited. Call GetFloatValueAt only if the property is a float type. The Index parameter identifies which object's value should be returned. It corresponds to the Index parameter of the GetComponent (see page 11026) method.

Use GetFloatValueAt to iterate over all current values, for example when implementing the AllEqual (see page 11024) method.

See Also

GetComponent (see page 11026)

GetFloatValue (see page 11028)

AllEqual (see page 11024)

59.1.22.4.11 TPropertyEditor.GetInt64Value Method

Returns the property value of the first object currently being edited.

Pascal

```
function GetInt64Value: Int64;
```

C++

```
__fastcall Int64 GetInt64Value();
```

Description

Call GetInt64Value to obtain the current value of the property for the first object in the internal list of objects being edited. Call GetInt64Value only if the property is 64-bit integer type.

If more than one object is currently being edited, GetInt64Value returns the value for the object returned by GetComponent (see page 11026)(0). To obtain the current property value for every object currently being edited, use the GetInt64ValueAt (see page 11029) method.

See Also

GetComponent (see page 11026)

GetInt64ValueAt (see page 11029)

59.1.22.4.12 TPropertyEditor.GetInt64ValueAt Method

Returns the property value of the specified object.

Pascal

```
function GetInt64ValueAt(Index: Integer): Int64;
```

C++

```
__fastcall Int64 GetInt64ValueAt(int Index);
```

Description

Call GetInt64ValueAt to obtain the current value of the property for an object in the internal list of objects being edited. Call GetInt64ValueAt only if the property is a 64-bit integer type. The Index parameter identifies which object's value should be returned. It corresponds to the Index parameter of the GetComponent (see page 11026) method.

Use GetInt64ValueAt to iterate over all current values, for example when implementing the AllEqual (see page 11024) method.

See Also

GetComponent (see page 11026)

GetInt64Value (see page 11028)

AllEqual (see page 11024)

59.1.22.4.13 TPropertyEditor.GetIntfValue Method

Returns the property value of the first object currently being edited.

Pascal

```
function GetIntfValue: IInterface;
```

C++

```
__fastcall IInterface GetIntfValue();
```

Description

Call GetIntfValue to obtain the current value of the property for the first object in the internal list of objects being edited. Call

GetIntfValue only if the property is a component interface type.

If more than one object is currently being edited, GetIntfValue returns the value for the object returned by GetComponent (see page 11026)(0). To obtain the current property value for every object currently being edited, use the GetIntfValueAt (see page 11030) method.

See Also

GetComponent (see page 11026)

GetIntfValueAt (see page 11030)

59.1.22.4.14 TPropertyEditor.GetIntfValueAt Method

Returns the property value of the specified object.

Pascal

```
function GetIntfValueAt(Index: Integer): IInterface;
```

C++

```
__fastcall IInterface GetIntfValueAt(int Index);
```

Description

Call GetIntfValueAt to obtain the current value of the property for an object in the internal list of objects being edited. Call GetIntfValueAt only if the property is a component interface type. The Index parameter identifies which object's value should be returned. It corresponds to the Index parameter of the GetComponent (see page 11026) method.

Use GetIntfValueAt to iterate over all current values, for example when implementing the AllEqual (see page 11024) method.

See Also

GetComponent (see page 11026)

GetIntfValue (see page 11029)

AllEqual (see page 11024)

59.1.22.4.15 TPropertyEditor.GetIsDefault Method

Pascal

```
function GetIsDefault: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetIsDefault();
```

Description

This is GetIsDefault, a member of class TPropertyEditor.

59.1.22.4.16 TPropertyEditor.GetKind Method

Pascal

```
function GetKind: TTypeKind;
```

C++

```
__fastcall TTypeKind GetKind();
```

Description

This is GetKind, a member of class TPropertyEditor.

59.1.22.4.17 TPropertyEditor.GetMethodValue Method

Returns the property value of the first object currently being edited.

Pascal

```
function GetMethodValue: TMethod;
```

C++

```
__fastcall TMethod GetMethodValue();
```

Description

Call GetMethodValue to obtain the current value of the property for the first object in the internal list of objects being edited. Call GetMethodValue only if the property is a method, such as the DefWndProc property of TWinControl (Windows) or TWidgetControl (Linux), or an event handler.

If more than one object is currently being edited, GetMethodValue returns the value for the object returned by GetComponent (see page 11026)(0). To obtain the current property value for every object currently being edited, use the GetMethodValueAt (see page 11031) method.

See Also

GetComponent (see page 11026)

GetMethodValueAt (see page 11031)

59.1.22.4.18 TPropertyEditor.GetMethodValueAt Method

Returns the property value of the specified object.

Pascal

```
function GetMethodValueAt(Index: Integer): TMethod;
```

C++

```
__fastcall TMethod GetMethodValueAt(int Index);
```

Description

Call GetMethodValueAt to obtain the current value of the property for an object in the internal list of objects being edited. Call GetMethodValueAt only if the property is a method, such as the DefWndProc property of TWinControl (Windows) or TWidgetControl (Linux), or an event handler. The Index parameter identifies which object's value should be returned. It corresponds to the Index parameter of the GetComponent (see page 11026) method.

Use GetMethodValueAt to iterate over all current values, for example when implementing the AllEqual (see page 11024) method.

See Also

GetComponent (see page 11026)

GetMethodValue (see page 11031)

AllEqual (see page 11024)

59.1.22.4.19 TPropertyEditor.GetName Method

Returns the name of the property.

Pascal

```
function GetName: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetName();
```

Description

The Object Inspector calls GetName to obtain the property name that it displays. GetName returns the name of the property from the property type information with all underbars replaced by spaces. Override this method if the Object Inspector should display a string other than the property name.

See Also

PropDrawName

59.1.22.4.20 TPropertyEditor.GetOrdValue Method

Returns the property value of the first object currently being edited.

Pascal

```
function GetOrdValue: Longint;
```

C++

```
__fastcall Longint GetOrdValue();
```

Description

Call GetOrdValue to obtain the current value of the property for the first object in the internal list of objects being edited. Call GetOrdValue only if the property is an ordinal type.

If more than one object is currently being edited, GetOrdValue returns the value for the object returned by GetComponent (see page 11026)(0). To obtain the current property value for every object currently being edited, use the GetOrdValueAt (see page 11032) method.

See Also

GetComponent (see page 11026)

GetOrdValueAt (see page 11032)

59.1.22.4.21 TPropertyEditor.GetOrdValueAt Method

Returns the property value of the specified object.

Pascal

```
function GetOrdValueAt(Index: Integer): Longint;
```

C++

```
__fastcall Longint GetOrdValueAt(int Index);
```

Description

Call GetOrdValueAt to obtain the current value of the property for an object in the internal list of objects being edited. Call GetOrdValueAt only if the property is an ordinal type. The Index parameter identifies which object's value should be returned. It

corresponds to the Index parameter of the GetComponent (see page 11026) method.

Use GetOrdValueAt to iterate over all current values, for example when implementing the AllEqual (see page 11024) method.

See Also

GetComponent (see page 11026)

GetOrdValue (see page 11032)

AllEqual (see page 11024)

59.1.22.4.22 TPropertyEditor.GetProperties Method

Provides the subproperties of the property to a callback procedure.

Pascal

```
procedure GetProperties(Proc: TGetPropProc); virtual;
```

C++

```
virtual __fastcall GetProperties(TGetPropProc Proc);
```

Description

The Object Inspector calls GetProperties when the property is expanded to show subproperties. The GetProperties method of TPropertyEditor (see page 11019) does nothing. Property editors that return paSubProperties from the GetAttributes (see page 11026) method should override GetProperties to create an appropriate property editor and pass its IProperty interface to the Proc parameter for every subproperty (or nested property) of the property.

See Also

GetAttributes (see page 11026)

59.1.22.4.23 TPropertyEditor.GetPropInfo Method

Returns the property information record for the object being edited.

Pascal

```
function GetPropInfo: PPropInfo; virtual;
```

C++

```
virtual __fastcall PPropInfo GetPropInfo();
```

Description

Call GetPropInfo to access the property information record that describes the property being edited, including its access functions. If more than one object is being edited, GetPropInfo returns the property information record for the object returned by GetComponent (see page 11026)(0).

If only the type information record is desired, use the GetPropType (see page 11033) method instead.

See Also

GetComponent (see page 11026)

GetPropType (see page 11033)

59.1.22.4.24 TPropertyEditor.GetPropType Method

Returns the type information record for the object being edited.

Pascal

```
function GetPropType: PTypeInfo;
```

C++

```
__fastcall PTypeInfo GetPropType();
```

Description

Call GetPropType to obtain access to the type information record that describes the type of the property being edited. If more than one object is being edited, GetPropType returns the type information record for the object returned by GetComponent (see page 11026)(0).

If information about the property access functions and storage details is desired as well, use the GetPropInfo (see page 11033) method instead.

See Also

GetComponent (see page 11026)

GetPropInfo (see page 11033)

59.1.22.4.25 TPropertyEditor.GetStrValue Method

Returns the property value of the first object currently being edited.

Pascal

```
function GetStrValue: string;
```

C++

```
__fastcall AnsiString GetStrValue();
```

Description

Call GetStrValue to obtain the current value of the property for the first object in the internal list of objects being edited. Only call GetStrValue if the property is a string type.

If more than one object is currently being edited, GetStrValue returns the value for the object returned by GetComponent (see page 11026)(0); To obtain the current property value for every object currently being edited, use the GetStrValueAt (see page 11034) method.

See Also

GetComponent (see page 11026)

GetStrValueAt (see page 11034)

59.1.22.4.26 TPropertyEditor.GetStrValueAt Method

Returns the property value of the specified object.

Pascal

```
function GetStrValueAt(Index: Integer): string;
```

C++

```
__fastcall AnsiString GetStrValueAt(int Index);
```

Description

Call GetStrValueAt to obtain the current value of the property for an object in the internal list of objects being edited. Call GetStrValueAt only if the property is an ordinal type. The Index parameter identifies which object's value should be returned. It

corresponds to the Index parameter of the GetComponent (see page 11026) method.

Use GetStrValueAt to iterate over all current values, for example when implementing the AllEqual (see page 11024) method.

See Also

GetComponent (see page 11026)

GetStrValue (see page 11034)

AllEqual (see page 11024)

59.1.22.4.27 TPropertyEditor.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raised an exception.

As implemented in TPropertyEditor (see page 11019), GetValue returns the string "(unknown)". Descendant classes override this method to provide an appropriate value string or raise an exception.

GetValue is the read access method of the Value (see page 11045) property.

Note: To obtain an error string when the property value is unavailable rather than raising an exception, you can call the GetEditValue (see page 11027) method. GetEditValue (see page 11027) is protected, but can be called through the IProperty interface in code that is not a method of a descendant class.

See Also

setvalue (see page 11042)

Value (see page 11045)

GetVisualValue (see page 11037)

GetEditValue (see page 11027)

59.1.22.4.28 TPropertyEditor.GetValues Method

Provides the enumerated values of the property to a callback procedure.

Pascal

```
procedure GetValues(Proc: TGetStrProc); virtual;
```

C++

```
virtual __fastcall GetValues(TGetStrProc Proc);
```

Description

The Object Inspector calls GetValues when the drop-down button is clicked to show enumerated values. The GetValues method of TPropertyEditor (see page 11019) does nothing. Property editors that return paValueList from the GetAttributes (see page 11026) method should override GetValues to call the Proc parameter for every string that can represent a valid enumerated value. Even if the GetAttributes (see page 11026) method does not return paValueList, property editors for enumerated types

should still override GetValues to enumerate all possible values.

See Also

GetAttributes (see page 11026)

Value (see page 11045)

59.1.22.4.29 TPropertyEditor.GetValueW Method

Pascal

```
function GetValueW: WideString; virtual;
```

C++

```
virtual __fastcall BSTR GetValueW();
```

Description

This is GetValueW, a member of class TPropertyEditor.

59.1.22.4.30 TPropertyEditor.GetVarValue Method

Returns the property value of the first object currently being edited.

Pascal

```
function GetVarValue: Variant;
```

C++

```
__fastcall Variant GetVarValue();
```

Description

Call GetVarValue to obtain the current value of the property for the first object in the internal list of objects being edited. Call GetVarValue only if the property value can be represented by a variant.

If more than one object is currently being edited, GetVarValue returns the value for the object returned by GetComponent (see page 11026)(0); To obtain the current property value for every object currently being edited, use the GetVarValueAt (see page 11036) method.

See Also

GetComponent (see page 11026)

GetVarValueAt (see page 11036)

59.1.22.4.31 TPropertyEditor.GetVarValueAt Method

Returns the property value of the specified object.

Pascal

```
function GetVarValueAt(Index: Integer): Variant;
```

C++

```
__fastcall Variant GetVarValueAt(int Index);
```

Description

Call GetVarValueAt to obtain the current value of the property for an object in the internal list of objects being edited. Call GetVarValueAt only if the property value can be represented as a variant. The Index parameter identifies which object's value

should be returned. It corresponds to the Index parameter of the GetComponent (see page 11026) method.

Use GetVarValueAt to iterate over all current values, for example when implementing the AllEqual (see page 11024) method.

See Also

GetComponent (see page 11026)

GetVarValue (see page 11036)

AllEqual (see page 11024)

59.1.22.4.32 TPropertyEditor.GetVisualValue Method

Returns the property value as it appears in the Object Inspector.

Pascal

```
function GetVisualValue: string;
```

C++

```
__fastcall AnsiString GetVisualValue();
```

Description

GetVisualValue returns the string that the Object Inspector displays to represent the property value. If only one item is selected or all multi-selected items have the same value, GetVisualValue returns the string formatted by GetValue (see page 11035). Otherwise, GetVisualValue returns an empty string.

See Also

GetValue (see page 11035)

GetEditValue (see page 11027)

59.1.22.4.33 TPropertyEditor.GetWideStrValue Method

Pascal

```
function GetWideStrValue: WideString;
```

C++

```
__fastcall BSTR GetWideStrValue();
```

Description

This is GetWideStrValue, a member of class TPropertyEditor.

59.1.22.4.34 TPropertyEditor.GetWideStrValueAt Method

Pascal

```
function GetWideStrValueAt(Index: Integer): WideString;
```

C++

```
__fastcall BSTR GetWideStrValueAt(int Index);
```

Description

This is GetWideStrValueAt, a member of class TPropertyEditor.

59.1.22.4.35 TPropertyEditor.HasInstance Method

Indicates whether a specified object is currently being edited by the property editor.

Pascal

```
function HasInstance(Instance: TPersistent): Boolean;
```

C++

```
__fastcall Boolean HasInstance(TPersistent * Instance);
```

Description

HasInstance returns true if the Instance parameter references one of the objects that the GetComponent (see page 11026) method returns, false otherwise.

See Also

GetComponent (see page 11026)

59.1.22.4.36 TPropertyEditor.Initialize Method

Initializes the property editor after it is created but before it is used.

Pascal

```
procedure Initialize; override;
```

C++

```
virtual __fastcall Initialize();
```

Description

Do not call the Initialize method. It is called automatically by the Object Inspector to allow property editors to perform any necessary initialization before they are used. The Initialize method of TPropertyEditor (see page 11019) does nothing. Override the Initialize method to perform any special initialization, such as the creation of helper objects.

Property editors are often created and discarded without being used because they do not represent a common property across all selected components. Since Initialize is called only after it is determined that the property editor will be used, it is more efficient to initialize property values and create helper objects in the Initialize method rather than the constructor.

See Also

Activate (see page 11024)

Create (see page 22287)

59.1.22.4.37 TPropertyEditor.Modified Method

Informs the designer that the property value has changed.

Pascal

```
procedure Modified;
```

C++

```
__fastcall Modified();
```

Description

Call Modified after changing the property value so that the designer is aware that something has changed.

It is not necessary to call Modified if changes are made using the SetFloatValue (see page 11039), SetMethodValue (see page 11040), SetOrdValue (see page 11041), SetStrValue (see page 11042), or SetVarValue (see page 11043) methods. These methods automatically call Modified.

See Also

Designer (see page 11044)

SetFloatValue (see page 11039)

SetMethodValue (see page 11040)

SetOrdValue (see page 11041)

SetVarValue (see page 11043)

Value (see page 11045)

59.1.22.4.38 TPropertyEditor.Revert Method

Restores all object properties to their original values.

Pascal

```
procedure Revert;
```

C++

```
__fastcall Revert();
```

Description

Call Revert to restore the original property values that each of the edited objects had before the user began making modification with the Object Inspector. Revert will not work if GetAttributes (see page 11026) returns paAutoUpdate.

See Also

GetAttributes (see page 11026)

59.1.22.4.39 TPropertyEditor.SetFloatValue Method

Changes the property value for every object being edited.

Pascal

```
procedure SetFloatValue(Value: Extended);
```

C++

```
__fastcall SetFloatValue(Extended Value);
```

Description

Call SetFloatValue to change the property setting for every object that can be returned by the GetComponent (see page 11026) method. Do not call SetFloatValue unless the property editor represents a float value. After changing a property value using SetFloatValue, it is not necessary to call Modified (see page 11038).

See Also

GetComponent (see page 11026)

Modified (see page 11038)

setvalue (see page 11042)

59.1.22.4.40 TPropertyEditor.SetInt64Value Method

Changes the property value for every object being edited.

Pascal

```
procedure SetInt64Value(Value: Int64);
```

C++

```
__fastcall SetInt64Value(Int64 Value);
```

Description

Call SetInt64Value to change the property setting for every object that can be returned by the GetComponent (see page 11026) method. Do not call SetInt64Value unless the property editor represents a 64-bit integer value. After changing a property value using SetInt64Value, it is not necessary to call Modified (see page 11038).

See Also

GetComponent (see page 11026)

Modified (see page 11038)

setvalue (see page 11042)

59.1.22.4.41 TPropertyEditor.SetIntfValue Method

Changes the property value for every object being edited.

Pascal

```
procedure SetIntfValue(const Value: IInterface);
```

C++

```
__fastcall SetIntfValue(const IInterface Value);
```

Description

Call SetIntfValue to change the property setting for every object that can be returned by the GetComponent (see page 11026) method. Do not call SetIntfValue unless the property editor represents a component interface value. After changing a property value using SetIntfValue, it is not necessary to call Modified (see page 11038).

See Also

GetComponent (see page 11026)

Modified (see page 11038)

setvalue (see page 11042)

59.1.22.4.42 TPropertyEditor.SetMethodValue Method

Changes the property value for every object being edited.

Pascal

```
procedure SetMethodValue(const Value: TMethod);
```

C++

```
__fastcall SetMethodValue(const TMethod Value);
```

Description

Call SetMethodValue to change the property setting for every object that can be returned by the GetComponent (see page 11026) method. Do not call SetMethodValue unless the property represents a method. After changing a property value using SetMethodValue, it is not necessary to call Modified (see page 11038).

See Also

GetComponent (see page 11026)

Modified (see page 11038)

setvalue (see page 11042)

59.1.22.4.43 TPropertyEditor.SetOrdValue Method

Changes the property value for every object being edited.

Pascal

```
procedure SetOrdValue(Value: Longint);
```

C++

```
__fastcall SetOrdValue(Longint Value);
```

Description

Call SetOrdValue to change the property setting for every object that can be returned by the GetComponent (see page 11026) method. Do not call SetOrdValue unless the property represents an ordinal value. After changing a property value using SetOrdValue, it is not necessary to call Modified (see page 11038).

See Also

GetComponent (see page 11026)

Modified (see page 11038)

setvalue (see page 11042)

59.1.22.4.44 TPropertyEditor.SetPropEntry Method

Informs the property editor of a particular object whose property is to be edited.

Pascal

```
procedure SetPropEntry(Index: Integer; AInstance: TPersistent; APropInfo: PPropInfo); override;
```

C++

```
virtual __fastcall SetPropEntry(int Index, TPersistent * AInstance, PPropInfo APropInfo);
```

Description

The Object Inspector calls SetPropEntry for each object that has a property that the property editor is editing. This allows the property editor to store information about the objects being edited so that it can get and set property values.

Index is the index of the object being edited. This value can range from 0 to one less than the value of the PropCount (see page 11045) property.

AInstance is a reference to the object being edited. This is later used to supply the value that the GetComponent (see page 11026) method returns.

APropInfo is a pointer to the property information record for the property on AInstance. This is later used to supply the value that the GetPropInfo (see page 11033) method returns.

See Also

GetComponent (see page 11026)

GetPropInfo (see page 11033)

PropCount (see page 11045)

59.1.22.4.45 TPropertyEditor.SetStrValue Method

Changes the property value for every object being edited.

Pascal

```
procedure SetStrValue(const Value: string);
```

C++

```
__fastcall SetStrValue(const AnsiString Value);
```

Description

Call SetStrValue to change the property setting for every object that can be returned by the GetComponent (see page 11026) method. Do not call SetStrValue unless the property represents a string value. After changing a property value using SetStrValue, it is not necessary to call Modified (see page 11038).

See Also

GetComponent (see page 11026)

Modified (see page 11038)

setvalue (see page 11042)

59.1.22.4.46 SetValue Method

This is the overview for the SetValue method overload.

Overload List

	Name	Description
	TPropertyEditor.SetValue (string) (see page 11043)	Responds when the user changes the value of the property.
	TPropertyEditor.SetValue (WideString) (see page 11042)	This is SetValue, a member of class TPropertyEditor.

59.1.22.4.46.1 TPropertyEditor.SetValue Method (WideString)

Pascal

```
procedure SetValue(const Value: WideString); virtual; overload;
```

C++

```
virtual __fastcall SetValue(const BSTR Value);
```

Description

This is SetValue, a member of class TPropertyEditor.

59.1.22.4.46.2 TPropertyEditor.SetValue Method (string)

Responds when the user changes the value of the property.

Pascal

```
procedure SetValue(const Value: string); virtual; overload;
```

C++

```
virtual __fastcall SetValue(const AnsiString Value);
```

Description

The Object Inspector calls SetValue when the user has finished changing the property value. SetValue applies the value specified by the Value parameter to all components that it is currently editing. If the value can't be set, SetValue raises an exception that describes the problem.

Note: If the property is changed only through a dialog brought up by the Edit method, the Object Inspector does not call SetValue. Rather, the Edit method should handle the setting of the value on all components.

See Also

Edit (see page 11200)

59.1.22.4.47 TPropertyEditor.SetVarValue Method

Changes the property value for every object being edited.

Pascal

```
procedure SetVarValue(const Value: Variant);
```

C++

```
__fastcall SetVarValue(const Variant Value);
```

Description

Call SetVarValue to change the property setting for every object that can be returned by the GetComponent (see page 11026) method. Do not call SetVarValue unless the property can be represented by a variant value. After changing a property value using SetVarValue, it is not necessary to call Modified (see page 11038).

See Also

GetComponent (see page 11026)

Modified (see page 11038)

setvalue (see page 11042)

59.1.22.4.48 TPropertyEditor.SetWideStrValue Method

Pascal

```
procedure SetWideStrValue(const Value: WideString);
```

C++

```
__fastcall SetWideStrValue(const BSTR Value);
```

Description

This is SetWideStrValue, a member of class TPropertyEditor.

59.1.22.4.49 TPropertyEditor.ValueAvailable Method

Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Pascal

```
function ValueAvailable: Boolean;
```

C++

```
__fastcall Boolean ValueAvailable();
```

Description

The Object Inspector calls ValueAvailable to determine which components can be edited for a particular property or which properties can be displayed for a particular component.

See Also

ComponentStyle (see page 2805)

59.1.22.5 TPropertyEditor Properties

The properties of the TPropertyEditor class are listed here.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.22.5.1 TPropertyEditor.Designer Property

Provides access to the form designer object in the IDE.

Pascal

```
property Designer: IDesigner;
```

C++

```
__property IDesigner Designer;
```

Description

Use Designer to read from or make changes to the current form or module in the IDE. The IDesigner interface has methods that can access the components in the current unit and other units linked in by the uses clause (Delphi) or included headers (C++), create or alter the methods in the current unit, and so on.

59.1.22.5.2 TPropertyEditor.PrivateDirectory Property

Specifies the location where the property editor can store auxiliary files.

Pascal

```
property PrivateDirectory: string;
```

C++

```
__property AnsiString PrivateDirectory;
```

Description

Use PrivateDirectory to store executables used by the property editor or auxiliary files that store state information, examples, templates, and so on.

Note: On Windows, PrivateDirectory is the location specified in the registry under the key

Note: HKEY_CURRENT_USER\Software\Borland (see page 1761)\Delphi\x.0\Globals\PrivateDir

Note: HKEY_CURRENT_USER\Software\Borland (see page 1761)\CBuilder\x.0\Globals\PrivateDir

The property editor can use the directory to read files, write files, or store DLLs (Windows) or shared objects (Linux) and executables.

59.1.22.5.3 TPropertyEditor.PropCount Property

Indicates the number of components whose properties are being edited.

Pascal

```
property PropCount: Integer;
```

C++

```
__property int PropCount;
```

Description

Use PropCount to determine the valid range of indexes to use when calling the GetComponent (see page 11026) method. This value corresponds to the number of components selected when the property editor is created. Typically, it is not necessary to access individual components because the Getxxx and Setxxx methods propagate values appropriately.

See Also

GetComponent (see page 11026)

setvalue (see page 11042)

59.1.22.5.4 TPropertyEditor.Value Property

Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

Pascal

```
property Value: string;
```

C++

```
__property AnsiString Value;
```

Description

Read Value to determine the current property setting as it is displayed in the Object Inspector. Set Value to apply a value to all selected components for which the property is appropriate.

The Value property is implemented using the GetValue (see page 11035) and SetValue (see page 11042) methods. Descendants of TPropertyEditor (see page 11019) must override these methods to implement the Value property.

See Also

GetValue (see page 11035)

setvalue (see page 11042)

59.1.23 TSelectionEditor Class

Class Hierarchy

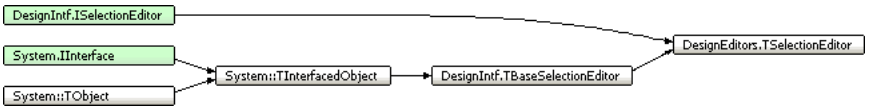

Pascal

```
TSelectionEditor = class(TBaseSelectionEditor, ISelectionEditor);
```

C++

```
class TSelectionEditor : public TBaseSelectionEditor, public ISelectionEditor;
```

File

DesignEditors

Description

This is class DesignEditors.TSelectionEditor.

59.1.23.1 TSelectionEditor Members

The following tables list the members exposed by TSelectionEditor.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11047)	This is Create, a member of class TSelectionEditor.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	ExecuteVerb (see page 11048)	This is ExecuteVerb, a member of class TSelectionEditor.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetVerb (see page 11048)	This is GetVerb, a member of class TSelectionEditor.
V	GetVerbCount (see page 11048)	This is GetVerbCount, a member of class TSelectionEditor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	PrepareItem (see page 11048)	This is PrepareItem, a member of class TSelectionEditor.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	RequiresUnits (see page 11049)	This is RequiresUnits, a member of class TSelectionEditor.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Designer (see page 11049)	This is Designer, a member of class TSelectionEditor.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

59.1.23.2 TSelectionEditor.Create Constructor

Pascal

```
constructor Create(const ADesigner: IDesigner); override;
```

C++

```
virtual __fastcall TSelectionEditor(const IDesigner ADesigner);
```

Description

This is Create, a member of class TSelectionEditor.

59.1.23.3 TSelectionEditor Methods

The methods of the TSelectionEditor class are listed here.

Public Methods

	Name	Description
	ExecuteVerb (see page 11048)	This is ExecuteVerb, a member of class TSelectionEditor.
	GetVerb (see page 11048)	This is GetVerb, a member of class TSelectionEditor.
	GetVerbCount (see page 11048)	This is GetVerbCount, a member of class TSelectionEditor.
	PrepareItem (see page 11048)	This is PrepareItem, a member of class TSelectionEditor.
	RequiresUnits (see page 11049)	This is RequiresUnits, a member of class TSelectionEditor.

59.1.23.3.1 TSelectionEditor.ExecuteVerb Method

Pascal

```
procedure ExecuteVerb(Index: Integer; const List: IDesignerSelections); virtual;
```

C++

```
virtual __fastcall ExecuteVerb(int Index, const IDesignerSelections List);
```

Description

This is ExecuteVerb, a member of class TSelectionEditor.

59.1.23.3.2 TSelectionEditor.GetVerb Method

Pascal

```
function GetVerb(Index: Integer): string; virtual;
```

C++

```
virtual __fastcall AnsiString GetVerb(int Index);
```

Description

This is GetVerb, a member of class TSelectionEditor.

59.1.23.3.3 TSelectionEditor.GetVerbCount Method

Pascal

```
function GetVerbCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetVerbCount();
```

Description

This is GetVerbCount, a member of class TSelectionEditor.

59.1.23.3.4 TSelectionEditor.PrepareItem Method

Pascal

```
procedure PrepareItem(Index: Integer; const AItem: IMenuItem); virtual;
```

C++

```
virtual __fastcall PrepareItem(int Index, const IMenuItem AItem);
```

Description

This is PrepareItem, a member of class TSelectionEditor.

59.1.23.3.5 TSelectionEditor.RequiresUnits Method

Pascal

```
procedure RequiresUnits(Proc: TGetStrProc); virtual;
```

C++

```
virtual __fastcall RequiresUnits(TGetStrProc Proc);
```

Description

This is RequiresUnits, a member of class TSelectionEditor.

59.1.23.4 TSelectionEditor Properties

The properties of the TSelectionEditor class are listed here.

Public Properties

	Name	Description
	Designer (see page 11049)	This is Designer, a member of class TSelectionEditor.

59.1.23.4.1 TSelectionEditor.Designer Property

Pascal

```
property Designer: IDesigner;
```

C++

```
__property IDesigner Designer;
```

Description

This is Designer, a member of class TSelectionEditor.

59.1.24 TSetElementProperty Class

Class Hierarchy

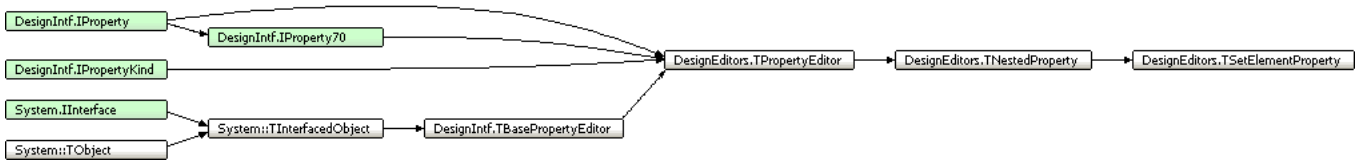

Pascal

```
TSetElementProperty = class(TNestedProperty);
```

C++

```
class TSetElementProperty : public TNestedProperty;
```

File

DesignEditors

Description

This is class DesignEditors.TSetElementProperty.

59.1.24.1 TSetElementProperty Members

The following tables list the members exposed by TSetElementProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	Create (see page 11052)	This is Create, a member of class TSetElementProperty.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11054)	This is GetIsDefault, a member of class TSetElementProperty.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.

	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Protected Properties

	Name	Description
	Element (see page 11056)	This is Element, a member of class TSetElementProperty.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
	AfterConstruction (see page 22279)	Responds after the last constructor executes.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AllEqual (see page 11053)	Indicates whether all selected components have the same value for the property.
	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 11014)	Destroys an instance of TPropertyEditor.
	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttributes (see page 11053)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
	GetEditLimit (see page 11027)	Returns the maximum length, in characters, of the Value (see page 11045) property.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.

	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
V	GetName (see page 11054)	Returns the name of the property.
V	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
V	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
V	GetValue (see page 11055)	Returns the property value formatted as a string.
V	GetValues (see page 11055)	Provides the enumerated values of the property to a callback procedure.
V	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetValue (see page 11056)	Responds when the user changes the value of the property.
V	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.24.2 TSetElementProperty.Create Constructor

Pascal

```
constructor Create(Parent: TPropertyEditor; AElement: Integer); reintroduce;
```

C++

```
__fastcall TSetElementProperty(TPropertyEditor Parent, int AElement);
```

Description

This is Create, a member of class TSetElementProperty.

59.1.24.3 TSetElementProperty Methods

The methods of the TSetElementProperty class are listed here.

Protected Methods

	Name	Description
	GetIsDefault (see page 11054)	This is GetIsDefault, a member of class TSetElementProperty.

Public Methods

	Name	Description
	AllEqual (see page 11053)	Indicates whether all selected components have the same value for the property.
	GetAttributes (see page 11053)	Describes the property so the Object Inspector provides the appropriate controls.
	GetName (see page 11054)	Returns the name of the property.
	GetValue (see page 11055)	Returns the property value formatted as a string.
	GetValues (see page 11055)	Provides the enumerated values of the property to a callback procedure.
	SetValue (see page 11056)	Responds when the user changes the value of the property.

59.1.24.3.1 TSetElementProperty.AllEqual Method

Indicates whether all selected components have the same value for the property.

Pascal

```
function AllEqual: Boolean; override;
```

C++

```
virtual __fastcall Boolean AllEqual();
```

Description

The Object Inspector calls AllEqual when there is more than one component selected to determine whether a value should appear for this property. If this method returns true, the Object Inspector displays a value for the property. If this method returns false, the line for the property in the Object Inspector remains blank.

In TPropertyEditor, AllEqual always returns false if more than one selected component has the associated property. Override this method to check the properties of individual components and determine whether a property value can be displayed in the Object Inspector.

Note: AllEqual is called only when the value returned by GetAttributes includes paMultiSelect.

See Also

GetAttributes (see page 11026)

GetComponent (see page 11026)

GetValue (see page 11035)

59.1.24.3.2 TSetElementProperty.GetAttributes Method

Describes the property so the Object Inspector provides the appropriate controls.

Pascal

```
function GetAttributes: TPropertyAttributes; override;
```

C++

```
virtual __fastcall TPropertyAttributes GetAttributes();
```

Description

The Object Inspector calls GetAttributes to determine what controls to display for the property and what property editor methods to call. The GetAttributes method of TPropertyEditor returns [paMultiSelect, paRevertable]. Override GetAttributes to return the set of values that corresponds to the attributes of the associated property.

Before the Object Inspector calls the Activate method, GetAttributes does not need to provide accurate information about any attributes other than paSubProperties, paVCL, and paMultiSelect.

See Also

Activate (see page 11024)

Edit (see page 11025)

GetProperties (see page 11033)

GetValues (see page 11035)

setvalue (see page 11042)

Value (see page 11045)

ListDrawValue

ListMeasureHeight

ListMeasureWidth

59.1.24.3.3 TSetElementProperty.GetIsDefault Method

Pascal

```
function GetIsDefault: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsDefault();
```

Description

This is GetIsDefault, a member of class TSetElementProperty.

59.1.24.3.4 TSetElementProperty.GetName Method

Returns the name of the property.

Pascal

```
function GetName: string; override;
```

C++

```
virtual __fastcall AnsiString GetName();
```

Description

The Object Inspector calls GetName to obtain the property name that it displays. GetName returns the name of the property from the property type information with all underbars replaced by spaces. Override this method if the Object Inspector should display

a string other than the property name.

See Also

PropDrawName

59.1.24.3.5 TSetElementProperty.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raised an exception.

As implemented in TPropertyEditor, GetValue returns the string "(unknown)". Descendant classes override this method to provide an appropriate value string or raise an exception.

GetValue is the read access method of the Value property.

Note: To obtain an error string when the property value is unavailable rather than raising an exception, you can call the GetEditValue method. GetEditValue is protected, but can be called through the IProperty interface in code that is not a method of a descendant class.

See Also

setvalue (see page 11042)

Value (see page 11045)

GetVisualValue (see page 11037)

GetEditValue (see page 11027)

59.1.24.3.6 TSetElementProperty.GetValues Method

Provides the enumerated values of the property to a callback procedure.

Pascal

```
procedure GetValues(Proc: TGetStrProc); override;
```

C++

```
virtual __fastcall GetValues(TGetStrProc Proc);
```

Description

The Object Inspector calls GetValues when the drop-down button is clicked to show enumerated values. The GetValues method of TPropertyEditor does nothing. Property editors that return paValueList from the GetAttributes method should override GetValues to call the Proc parameter for every string that can represent a valid enumerated value. Even if the GetAttributes method does not return paValueList, property editors for enumerated types should still override GetValues to enumerate all possible values.

See Also

GetAttributes (see page 11026)

Value (see page 11045)

59.1.24.3.7 TSetElementProperty.SetValue Method

Responds when the user changes the value of the property.

Pascal

```
procedure SetValue(const Value: string); override;
```

C++

```
virtual __fastcall SetValue(const AnsiString Value);
```

Description

The Object Inspector calls SetValue when the user has finished changing the property value. SetValue applies the value specified by the Value parameter to all components that it is currently editing. If the value can't be set, SetValue raises an exception that describes the problem.

Note: If the property is changed only through a dialog brought up by the Edit method, the Object Inspector does not call SetValue. Rather, the Edit method should handle the setting of the value on all components.

See Also

Edit (see page 11200)

59.1.24.4 TSetElementProperty Properties

The properties of the TSetElementProperty class are listed here.

Protected Properties

	Name	Description
	Element (see page 11056)	This is Element, a member of class TSetElementProperty.

59.1.24.4.1 TSetElementProperty.Element Property

Pascal

```
property Element: Integer;
```

C++

```
__property int Element;
```

Description

This is Element, a member of class TSetElementProperty.

59.1.25 TSetProperty Class

Class Hierarchy

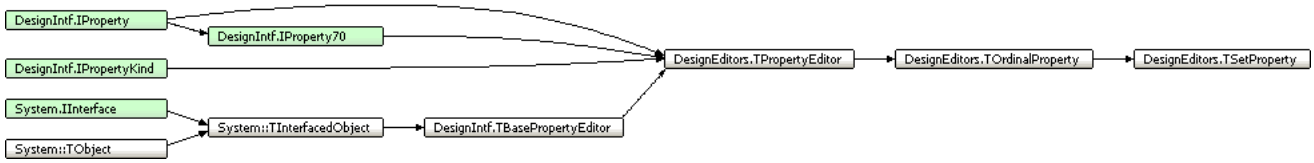

Pascal

```
TSetProperty = class(TOrdinalProperty);
```

C++

```
class TSetProperty : public TOrdinalProperty;
```

File

DesignEditors

Description

This is class DesignEditors.TSetProperty.

59.1.25.1 TSetProperty Members

The following tables list the members exposed by TSetProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.

	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
	AfterConstruction (see page 22279)	Responds after the last constructor executes.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AllEqual (see page 11018)	Indicates whether all selected components have the same value for the property.
	AllEqual (see page 11024)	Indicates whether all selected components have the same value for the property.
	AllEqual (see page 11199)	Indicates whether all selected components have the same value for the property.
	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttributes (see page 11060)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
	GetEditLimit (see page 11018)	Returns the maximum length, in characters, of the Value property.
	GetEditLimit (see page 11027)	Returns the maximum length, in characters, of the Value (see page 11045) property.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
V	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
V	GetProperties (see page 11060)	Provides the subproperties of the property to a callback procedure.
V	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
V	GetValue (see page 11061)	Returns the property value formatted as a string.
V	GetValues (see page 11035)	Provides the enumerated values of the property to a callback procedure.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.
V	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetValue (see page 11204)	Responds when the user changes the value of the property.
V	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.25.2 TSetProperty Methods

The methods of the TSetProperty class are listed here.

Public Methods

	Name	Description
	GetAttributes (see page 11060)	Describes the property so the Object Inspector provides the appropriate controls.
	GetProperties (see page 11060)	Provides the subproperties of the property to a callback procedure.
	GetValue (see page 11061)	Returns the property value formatted as a string.

59.1.25.2.1 TSetProperty.GetAttributes Method

Describes the property so the Object Inspector provides the appropriate controls.

Pascal

```
function GetAttributes: TPropertyAttributes; override;
```

C++

```
virtual __fastcall TPropertyAttributes GetAttributes();
```

Description

The Object Inspector calls GetAttributes to determine what controls to display for the property and what property editor methods to call. The GetAttributes method of TPropertyEditor returns [paMultiSelect, paRevertable]. Override GetAttributes to return the set of values that corresponds to the attributes of the associated property.

Before the Object Inspector calls the Activate method, GetAttributes does not need to provide accurate information about any attributes other than paSubProperties, paVCL, and paMultiSelect.

See Also

Activate (see page 11024)

Edit (see page 11025)

GetProperties (see page 11033)

GetValues (see page 11035)

setvalue (see page 11042)

Value (see page 11045)

ListDrawValue

ListMeasureHeight

ListMeasureWidth

59.1.25.2.2 TSetProperty.GetProperties Method

Provides the subproperties of the property to a callback procedure.

Pascal

```
procedure GetProperties(Proc: TGetPropProc); override;
```

C++

```
virtual __fastcall GetProperties(TGetPropProc Proc);
```

Description

The Object Inspector calls GetProperties when the property is expanded to show subproperties. The GetProperties method of TPropertyEditor does nothing. Property editors that return paSubProperties from the GetAttributes method should override GetProperties to create an appropriate property editor and pass its IProperty interface to the Proc parameter for every

subproperty (or nested property) of the property.

See Also

GetAttributes (see page 11026)

59.1.25.2.3 TSetProperty.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raised an exception.

As implemented in TPropertyEditor, GetValue returns the string "(unknown)". Descendant classes override this method to provide an appropriate value string or raise an exception.

GetValue is the read access method of the Value property.

Note: To obtain an error string when the property value is unavailable rather than raising an exception, you can call the GetEditValue method. GetEditValue is protected, but can be called through the IProperty interface in code that is not a method of a descendant class.

See Also

setvalue (see page 11042)

Value (see page 11045)

GetVisualValue (see page 11037)

GetEditValue (see page 11027)

59.1.26 TStringProperty Class

Class Hierarchy

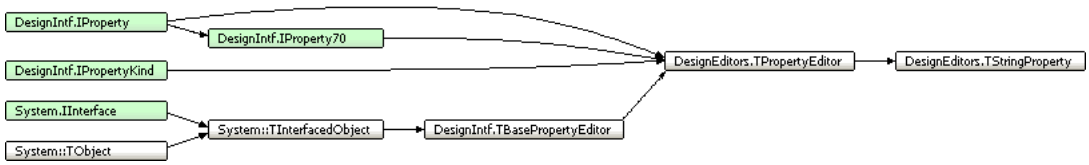

Pascal

```
TStringProperty = class(TPropertyEditor);
```

C++

```
class TStringProperty : public TPropertyEditor;
```

File

DesignEditors

Description

This is class DesignEditors.TStringProperty.

59.1.26.1 TStringProperty Members

The following tables list the members exposed by TStringProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
A	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AllEqual (see page 11065)	Indicates whether all selected components have the same value for the property.
V	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
V	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
V	GetAttributes (see page 11026)	Describes the property so the Object Inspector provides the appropriate controls.
	GetAttributes (see page 11200)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
V	GetEditLimit (see page 11065)	Returns the maximum length, in characters, of the Value property.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
V	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
V	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
V	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.

	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
V	GetValue (see page 11065)	Returns the property value formatted as a string.
V	GetValues (see page 11035)	Provides the enumerated values of the property to a callback procedure.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.
V	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetValue (see page 11066)	Responds when the user changes the value of the property.
V	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.26.2 TStringProperty Methods

The methods of the TStringProperty class are listed here.

Public Methods

	Name	Description
V	AllEqual (see page 11065)	Indicates whether all selected components have the same value for the property.
V	GetEditLimit (see page 11065)	Returns the maximum length, in characters, of the Value property.
V	GetValue (see page 11065)	Returns the property value formatted as a string.
V	SetValue (see page 11066)	Responds when the user changes the value of the property.

59.1.26.2.1 TStringProperty.AllEqual Method

Indicates whether all selected components have the same value for the property.

Pascal

```
function AllEqual: Boolean; override;
```

C++

```
virtual __fastcall Boolean AllEqual();
```

Description

The Object Inspector calls AllEqual when there is more than one component selected to determine whether a value should appear for this property. If this method returns true, the Object Inspector displays a value for the property. If this method returns false, the line for the property in the Object Inspector remains blank.

In TPropertyEditor, AllEqual always returns false if more than one selected component has the associated property. Override this method to check the properties of individual components and determine whether a property value can be displayed in the Object Inspector.

Note: AllEqual is called only when the value returned by GetAttributes includes paMultiSelect.

See Also

GetAttributes (see page 11026)

GetComponent (see page 11026)

GetValue (see page 11035)

59.1.26.2.2 TStringProperty.GetEditLimit Method

Returns the maximum length, in characters, of the Value property.

Pascal

```
function GetEditLimit: Integer; override;
```

C++

```
virtual __fastcall int GetEditLimit();
```

Description

The Object Inspector calls GetEditLimit to determine how many characters the user can type when entering a value for the property. GetEditLimit returns 255. Override this method when the maximum length of the property string is some other value.

See Also

Value (see page 11045)

59.1.26.2.3 TStringProperty.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raised an exception.

As implemented in TPropertyEditor, GetValue returns the string "(unknown)". Descendant classes override this method to provide an appropriate value string or raise an exception.

GetValue is the read access method of the Value property.

Note: To obtain an error string when the property value is unavailable rather than raising an exception, you can call the GetEditValue method. GetEditValue is protected, but can be called through the IProperty interface in code that is not a method of a descendant class.

See Also

setvalue (see page 11042)

Value (see page 11045)

GetVisualValue (see page 11037)

GetEditValue (see page 11027)

59.1.26.2.4 TStringProperty.SetValue Method

Responds when the user changes the value of the property.

Pascal

```
procedure SetValue(const Value: string); override;
```

C++

```
virtual __fastcall SetValue(const AnsiString Value);
```

Description

The Object Inspector calls SetValue when the user has finished changing the property value. SetValue applies the value specified by the Value parameter to all components that it is currently editing. If the value can't be set, SetValue raises an exception that describes the problem.

Note: If the property is changed only through a dialog brought up by the Edit method, the Object Inspector does not call SetValue. Rather, the Edit method should handle the setting of the value on all components.

See Also

Edit (see page 11200)

59.1.27 TTimeProperty Class

Class Hierarchy

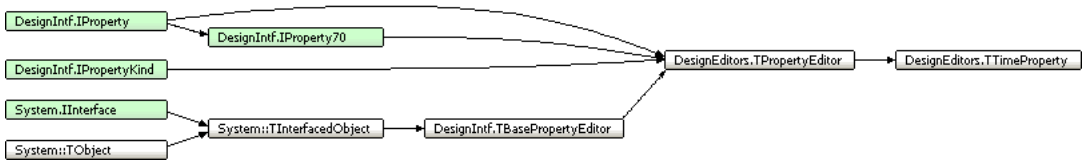

Pascal

```
TTimeProperty = class(TPropertyEditor);
```

C++

```
class TTimeProperty : public TPropertyEditor;
```

File

DesignEditors

Description

This is class DesignEditors.TTimeProperty.

59.1.27.1 TTimeProperty Members

The following tables list the members exposed by TTimeProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.

	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
	AfterConstruction (see page 22279)	Responds after the last constructor executes.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AllEqual (see page 11024)	Indicates whether all selected components have the same value for the property.
	AllEqual (see page 11199)	Indicates whether all selected components have the same value for the property.
	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttributes (see page 11070)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
	GetEditLimit (see page 11027)	Returns the maximum length, in characters, of the Value (see page 11045) property.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
V	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
V	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
V	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
V	GetValue (see page 11070)	Returns the property value formatted as a string.
V	GetValues (see page 11035)	Provides the enumerated values of the property to a callback procedure.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.
V	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetValue (see page 11071)	Responds when the user changes the value of the property.
V	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.27.2 TTimeProperty Methods

The methods of the TTimeProperty class are listed here.

Public Methods

	Name	Description
	GetAttributes (see page 11070)	Describes the property so the Object Inspector provides the appropriate controls.
	GetValue (see page 11070)	Returns the property value formatted as a string.
	SetValue (see page 11071)	Responds when the user changes the value of the property.

59.1.27.2.1 TTimeProperty.GetAttributes Method

Describes the property so the Object Inspector provides the appropriate controls.

Pascal

```
function GetAttributes: TPropertyAttributes; override;
```

C++

```
virtual __fastcall TPropertyAttributes GetAttributes();
```

Description

The Object Inspector calls GetAttributes to determine what controls to display for the property and what property editor methods to call. The GetAttributes method of TPropertyEditor returns [paMultiSelect, paRevertable]. Override GetAttributes to return the set of values that corresponds to the attributes of the associated property.

Before the Object Inspector calls the Activate method, GetAttributes does not need to provide accurate information about any attributes other than paSubProperties, paVCL, and paMultiSelect.

See Also

Activate (see page 11024)

Edit (see page 11025)

GetProperties (see page 11033)

GetValues (see page 11035)

setvalue (see page 11042)

Value (see page 11045)

ListDrawValue

ListMeasureHeight

ListMeasureWidth

59.1.27.2.2 TTimeProperty.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raised an exception.

As implemented in TPropertyEditor, GetValue returns the string "(unknown)". Descendant classes override this method to

provide an appropriate value string or raise an exception.

GetValue is the read access method of the Value property.

Note: To obtain an error string when the property value is unavailable rather than raising an exception, you can call the GetEditValue method. GetEditValue is protected, but can be called through the IProperty interface in code that is not a method of a descendant class.

See Also

setvalue (see page 11042)

Value (see page 11045)

GetVisualValue (see page 11037)

GetEditValue (see page 11027)

59.1.27.2.3 TTimeProperty.SetValue Method

Responds when the user changes the value of the property.

Pascal

```
procedure SetValue(const Value: string); override;
```

C++

```
virtual __fastcall SetValue(const AnsiString Value);
```

Description

The Object Inspector calls SetValue when the user has finished changing the property value. SetValue applies the value specified by the Value parameter to all components that it is currently editing. If the value can't be set, SetValue raises an exception that describes the problem.

Note: If the property is changed only through a dialog brought up by the Edit method, the Object Inspector does not call SetValue. Rather, the Edit method should handle the setting of the value on all components.

See Also

Edit (see page 11200)

59.1.28 TVariantProperty Class

Class Hierarchy

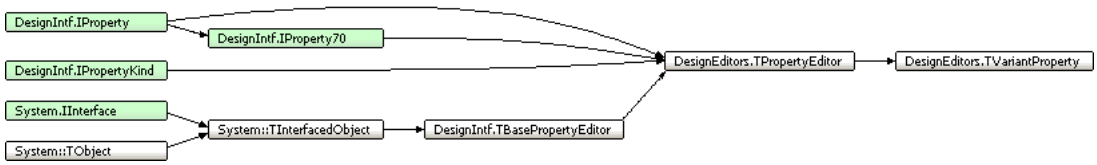

Pascal

```
TVariantProperty = class(TPropertyEditor);
```

C++

```
class TVariantProperty : public TPropertyEditor;
```

File

DesignEditors

Description

This is class DesignEditors.TVariantProperty.

59.1.28.1 TVariantProperty Members

The following tables list the members exposed by TVariantProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
A	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.

	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AllEqual (see page 11024)	Indicates whether all selected components have the same value for the property.
	AllEqual (see page 11199)	Indicates whether all selected components have the same value for the property.
V	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
V	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
V	GetAttributes (see page 11075)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
V	GetEditLimit (see page 11027)	Returns the maximum length, in characters, of the Value (see page 11045) property.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
V	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.

V	GetProperties (see page 11075)	Provides the subproperties of the property to a callback procedure.
V	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
V	GetValue (see page 11076)	Returns the property value formatted as a string.
V	GetValues (see page 11035)	Provides the enumerated values of the property to a callback procedure.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.
V	GetValueW (see page 11036)	This is GetValueW, a member of class TPropertyEditor.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetValue (see page 11076)	Responds when the user changes the value of the property.
V	SetValue (see page 11042)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.28.2 TVariantProperty Methods

The methods of the TVariantProperty class are listed here.

Public Methods

	Name	Description
V	GetAttributes (see page 11075)	Describes the property so the Object Inspector provides the appropriate controls.
V	GetProperties (see page 11075)	Provides the subproperties of the property to a callback procedure.
V	GetValue (see page 11076)	Returns the property value formatted as a string.
V	SetValue (see page 11076)	Responds when the user changes the value of the property.

59.1.28.2.1 TVariantProperty.GetAttributes Method

Describes the property so the Object Inspector provides the appropriate controls.

Pascal

```
function GetAttributes: TPropertyAttributes; override;
```

C++

```
virtual __fastcall TPropertyAttributes GetAttributes();
```

Description

The Object Inspector calls GetAttributes to determine what controls to display for the property and what property editor methods to call. The GetAttributes method of TPropertyEditor returns [paMultiSelect, paRevertable]. Override GetAttributes to return the set of values that corresponds to the attributes of the associated property.

Before the Object Inspector calls the Activate method, GetAttributes does not need to provide accurate information about any attributes other than paSubProperties, paVCL, and paMultiSelect.

See Also

Activate (see page 11024)

Edit (see page 11025)

GetProperties (see page 11033)

GetValues (see page 11035)

setvalue (see page 11042)

Value (see page 11045)

ListDrawValue

ListMeasureHeight

ListMeasureWidth

59.1.28.2.2 TVariantProperty.GetProperties Method

Provides the subproperties of the property to a callback procedure.

Pascal

```
procedure GetProperties(Proc: TGetPropProc); override;
```

C++

```
virtual __fastcall GetProperties(TGetPropProc Proc);
```

Description

The Object Inspector calls GetProperties when the property is expanded to show subproperties. The GetProperties method of TPropertyEditor does nothing. Property editors that return paSubProperties from the GetAttributes method should override GetProperties to create an appropriate property editor and pass its IProperty interface to the Proc parameter for every subproperty (or nested property) of the property.

See Also

GetAttributes (see page 11026)

59.1.28.2.3 TVariantProperty.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raised an exception.

As implemented in TPropertyEditor, GetValue returns the string "(unknown)". Descendant classes override this method to provide an appropriate value string or raise an exception.

GetValue is the read access method of the Value property.

Note: To obtain an error string when the property value is unavailable rather than raising an exception, you can call the GetEditValue method. GetEditValue is protected, but can be called through the IProperty interface in code that is not a method of a descendant class.

See Also

setvalue (see page 11042)

Value (see page 11045)

GetVisualValue (see page 11037)

GetEditValue (see page 11027)

59.1.28.2.4 TVariantProperty.SetValue Method

Responds when the user changes the value of the property.

Pascal

```
procedure SetValue(const Value: string); override;
```

C++

```
virtual __fastcall SetValue(const AnsiString Value);
```

Description

The Object Inspector calls SetValue when the user has finished changing the property value. SetValue applies the value specified by the Value parameter to all components that it is currently editing. If the value can't be set, SetValue raises an exception that describes the problem.

Note: If the property is changed only through a dialog brought up by the Edit method, the Object Inspector does not call SetValue. Rather, the Edit method should handle the setting of the value on all components.

See Also

Edit (see page 11200)

59.1.29 TWideCharProperty Class

Class Hierarchy

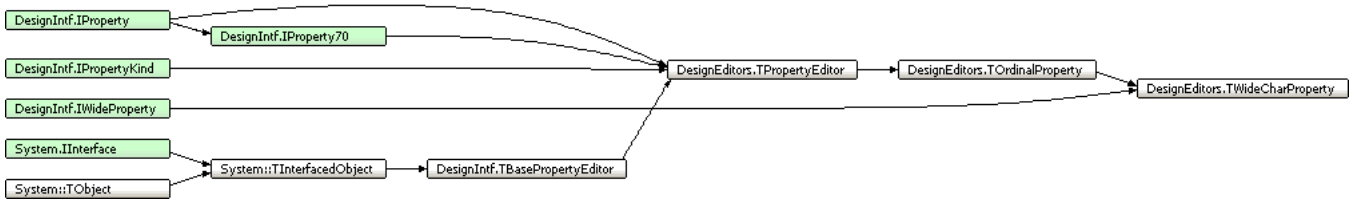

Pascal

```
TWideCharProperty = class(TOrdinalProperty, IWideProperty);
```

C++

```
class TWideCharProperty : public TOrdinalProperty, public IWideProperty;
```

File

DesignEditors

Description

This is class DesignEditors.TWideCharProperty.

59.1.29.1 TWideCharProperty Members

The following tables list the members exposed by TWideCharProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.
	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.

	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
A	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AllEqual (see page 11018)	Indicates whether all selected components have the same value for the property.
V	AllEqual (see page 11024)	Indicates whether all selected components have the same value for the property.
	AllEqual (see page 11199)	Indicates whether all selected components have the same value for the property.
V	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
V	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.

	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
V	GetAttributes (see page 11026)	Describes the property so the Object Inspector provides the appropriate controls.
	GetAttributes (see page 11200)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
V	GetEditLimit (see page 11018)	Returns the maximum length, in characters, of the Value property.
V	GetEditLimit (see page 11027)	Returns the maximum length, in characters, of the Value (see page 11045) property.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetEditValue (see page 11220)	This is GetEditValue, a member of class IWideProperty.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
V	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
V	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
V	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
V	GetValue (see page 11080)	Returns the property value formatted as a string.
V	GetValues (see page 11035)	Provides the enumerated values of the property to a callback procedure.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.
V	GetValueW (see page 11081)	This is GetValueW, a member of class TWideCharProperty.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetValue (see page 11042)	This is the overview for the SetValue method overload.
V	SetValue (see page 11081)	This is the overview for the SetValue method overload.
	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.29.2 TWideCharProperty Methods

The methods of the TWideCharProperty class are listed here.

Public Methods

	Name	Description
V	GetValue (see page 11080)	Returns the property value formatted as a string.
V	GetValueW (see page 11081)	This is GetValueW, a member of class TWideCharProperty.
V	SetValue (see page 11081)	This is the overview for the SetValue method overload.

59.1.29.2.1 TWideCharProperty.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raised an exception.

As implemented in TPropertyEditor, GetValue returns the string "(unknown)". Descendant classes override this method to provide an appropriate value string or raise an exception.

GetValue is the read access method of the Value property.

Note: To obtain an error string when the property value is unavailable rather than raising an exception, you can call the GetEditValue method. GetEditValue is protected, but can be called through the IProperty interface in code that is not a method of a descendant class.

See Also

setvalue (see page 11042)

Value (see page 11045)

GetVisualValue (see page 11037)

GetEditValue (see page 11027)

59.1.29.2.2 TWideCharProperty.GetValueW Method

Pascal

```
function GetValueW: WideString; override;
```

C++

```
virtual __fastcall BSTR GetValueW();
```

Description

This is GetValueW, a member of class TWideCharProperty.

59.1.29.2.3 SetValue Method

This is the overview for the SetValue method overload.

Overload List

	Name	Description
	TWideCharProperty.SetValue (string) (see page 11081)	Responds when the user changes the value of the property.
	TWideCharProperty.SetValue (WideString) (see page 11081)	This is SetValue, a member of class TWideCharProperty.

59.1.29.2.3.1 TWideCharProperty.SetValue Method (WideString)

Pascal

```
procedure SetValue(const Value: WideString); override;
```

C++

```
virtual __fastcall SetValue(const BSTR Value);
```

Description

This is SetValue, a member of class TWideCharProperty.

59.1.29.2.3.2 TWideCharProperty.SetValue Method (string)

Responds when the user changes the value of the property.

Pascal

```
procedure SetValue(const Value: string); override;
```

C++

```
virtual __fastcall SetValue(const AnsiString Value);
```

Description

The Object Inspector calls SetValue when the user has finished changing the property value. SetValue applies the value specified by the Value parameter to all components that it is currently editing. If the value can't be set, SetValue raises an exception that describes the problem.

Note: If the property is changed only through a dialog brought up by the Edit method, the Object Inspector does not call SetValue. Rather, the Edit method should handle the setting of the value on all components.

See Also

Edit (see page 11200)

59.1.30 TWideStringProperty Class

Class Hierarchy

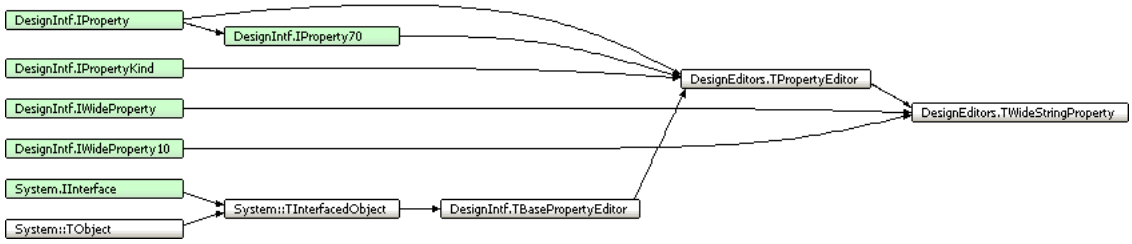

Pascal

```
TWideStringProperty = class(TPropertyEditor, IWideProperty, IWideProperty10);
```

C++

```
class TWideStringProperty : public TPropertyEditor, public IWideProperty, public
IWideProperty10;
```

File

DesignEditors

Description

This is class DesignEditors.TWideStringProperty.

59.1.30.1 TWideStringProperty Members

The following tables list the members exposed by TWideStringProperty.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetEditValue (see page 11027)	This is the overview for the GetEditValue method overload.
	GetFloatValue (see page 11028)	Returns the property value of the first object currently being edited.
	GetFloatValueAt (see page 11028)	Returns the property value of the specified object.
	GetInt64Value (see page 11028)	Returns the property value of the first object currently being edited.
	GetInt64ValueAt (see page 11029)	Returns the property value of the specified object.
	GetIntfValue (see page 11029)	Returns the property value of the first object currently being edited.
	GetIntfValueAt (see page 11030)	Returns the property value of the specified object.
V	GetIsDefault (see page 11030)	This is GetIsDefault, a member of class TPropertyEditor.

	GetMethodValue (see page 11031)	Returns the property value of the first object currently being edited.
	GetMethodValueAt (see page 11031)	Returns the property value of the specified object.
	GetOrdValue (see page 11032)	Returns the property value of the first object currently being edited.
	GetOrdValueAt (see page 11032)	Returns the property value of the specified object.
	GetStrValue (see page 11034)	Returns the property value of the first object currently being edited.
	GetStrValueAt (see page 11034)	Returns the property value of the specified object.
	GetVarValue (see page 11036)	Returns the property value of the first object currently being edited.
	GetVarValueAt (see page 11036)	Returns the property value of the specified object.
	GetWideStrValue (see page 11037)	This is GetWideStrValue, a member of class TPropertyEditor.
	GetWideStrValueAt (see page 11037)	This is GetWideStrValueAt, a member of class TPropertyEditor.
	HasInstance (see page 11038)	Indicates whether a specified object is currently being edited by the property editor.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	Modified (see page 11038)	Informs the designer that the property value has changed.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFloatValue (see page 11039)	Changes the property value for every object being edited.
	SetInt64Value (see page 11040)	Changes the property value for every object being edited.
	SetIntfValue (see page 11040)	Changes the property value for every object being edited.
	SetMethodValue (see page 11040)	Changes the property value for every object being edited.
	SetOrdValue (see page 11041)	Changes the property value for every object being edited.
V	SetPropEntry (see page 11041)	Informs the property editor of a particular object whose property is to be edited.
A	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.
	SetStrValue (see page 11042)	Changes the property value for every object being edited.
	SetVarValue (see page 11043)	Changes the property value for every object being edited.
	SetWideStrValue (see page 11043)	This is SetWideStrValue, a member of class TPropertyEditor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	Activate (see page 11024)	Responds when the property is selected in the Object Inspector.
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AllEqual (see page 11085)	Indicates whether all selected components have the same value for the property.
V	AutoFill (see page 11025)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11022)	Creates a new TPropertyEditor (see page 11019) object.
V	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 11022)	Destroys an instance of TPropertyEditor (see page 11019).
V	Destroy (see page 22287)	Disposes of an object instance.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 11025)	Responds when the user double-clicks the property in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttributes (see page 11026)	Describes the property so the Object Inspector provides the appropriate controls.
	GetAttributes (see page 11200)	Describes the property so the Object Inspector provides the appropriate controls.
	GetComponent (see page 11026)	Returns the specified object from the set of all objects currently being edited by the property editor.
	GetEditLimit (see page 11027)	Returns the maximum length, in characters, of the Value (see page 11045) property.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetEditValue (see page 11220)	This is GetEditValue, a member of class IWideProperty.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.
	GetKind (see page 11030)	This is GetKind, a member of class TPropertyEditor.
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.
	GetName (see page 11032)	Returns the name of the property.
	GetName (see page 11202)	Returns the name of the property.
	GetName (see page 11221)	This is GetName, a member of class IWideProperty10.
	GetNameW (see page 11086)	This is GetNameW, a member of class TWideStringProperty.
	GetProperties (see page 11033)	Provides the subproperties of the property to a callback procedure.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
	GetPropInfo (see page 11033)	Returns the property information record for the object being edited.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11033)	Returns the type information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
	GetValue (see page 11086)	Returns the property value formatted as a string.
	GetValues (see page 11086)	This is GetValues, a member of class TWideStringProperty.
	GetValueW (see page 11087)	This is GetValueW, a member of class TWideStringProperty.
	GetVisualValue (see page 11037)	Returns the property value as it appears in the Object Inspector.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	Initialize (see page 11038)	Initializes the property editor after it is created but before it is used.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Revert (see page 11039)	Restores all object properties to their original values.
	Revert (see page 11204)	Restores all object properties to their original values.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetValue (see page 11087)	This is SetValue, a member of class TWideStringProperty.
	SetValue (see page 11042)	This is the overview for the SetValue method overload.

	ValueAvailable (see page 11044)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	Designer (see page 11044)	Provides access to the form designer object in the IDE.
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.
R	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.
R	PrivateDirectory (see page 11044)	Specifies the location where the property editor can store auxiliary files.
R	PropCount (see page 11045)	Indicates the number of components whose properties are being edited.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Value (see page 11045)	Represents the current value of the property (as returned by GetValue (see page 11035)) as a string.

59.1.30.2 TWideStringProperty Methods

The methods of the TWideStringProperty class are listed here.

Public Methods

	Name	Description
V	AllEqual (see page 11085)	Indicates whether all selected components have the same value for the property.
	GetNameW (see page 11086)	This is GetNameW, a member of class TWideStringProperty.
V	GetValue (see page 11086)	Returns the property value formatted as a string.
V	GetValues (see page 11086)	This is GetValues, a member of class TWideStringProperty.
V	GetValueW (see page 11087)	This is GetValueW, a member of class TWideStringProperty.
V	SetValue (see page 11087)	This is SetValue, a member of class TWideStringProperty.

59.1.30.2.1 TWideStringProperty.AllEqual Method

Indicates whether all selected components have the same value for the property.

Pascal

```
function AllEqual: Boolean; override;
```

C++

```
virtual __fastcall Boolean AllEqual();
```

Description

The Object Inspector calls AllEqual when there is more than one component selected to determine whether a value should appear for this property. If this method returns true, the Object Inspector displays a value for the property. If this method returns false, the line for the property in the Object Inspector remains blank.

In TPropertyEditor, AllEqual always returns false if more than one selected component has the associated property. Override this method to check the properties of individual components and determine whether a property value can be displayed in the Object Inspector.

Note: AllEqual is called only when the value returned by GetAttributes includes paMultiSelect.

See Also

GetAttributes (see page 11026)

GetComponent (see page 11026)

GetValue (see page 11035)

59.1.30.2.2 TWideStringProperty.GetNameW Method

Pascal

```
function GetNameW: WideString;
```

C++

```
__fastcall BSTR GetNameW();
```

Description

This is GetNameW, a member of class TWideStringProperty.

59.1.30.2.3 TWideStringProperty.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raised an exception.

As implemented in TPropertyEditor, GetValue returns the string "(unknown)". Descendant classes override this method to provide an appropriate value string or raise an exception.

GetValue is the read access method of the Value property.

Note: To obtain an error string when the property value is unavailable rather than raising an exception, you can call the GetEditValue method. GetEditValue is protected, but can be called through the IProperty interface in code that is not a method of a descendant class.

See Also

setvalue (see page 11042)

Value (see page 11045)

GetVisualValue (see page 11037)

GetEditValue (see page 11027)

59.1.30.2.4 TWideStringProperty.GetValues Method

Pascal

```
procedure GetValues(Proc: TGetWideStrProc); virtual;
```

C++

```
virtual __fastcall GetValues(TGetWideStrProc Proc);
```

Description

This is GetValues, a member of class TWideStringProperty.

59.1.30.2.5 TWideStringProperty.GetValueW Method

Pascal

```
function GetValueW: WideString; override;
```

C++

```
virtual __fastcall BSTR GetValueW();
```

Description

This is GetValueW, a member of class TWideStringProperty.

59.1.30.2.6 TWideStringProperty.SetValue Method

Pascal

```
procedure SetValue(const Value: WideString); virtual;
```

C++

```
virtual __fastcall SetValue(const BSTR Value);
```

Description

This is SetValue, a member of class TWideStringProperty.

59.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IDesignGroupChange (see page 11088)	This is class DesignEditors.IDesignGroupChange.

59.2.1 IDesignGroupChange Interface

Class Hierarchy

DesignEditors.IDesignGroupChange

Pascal

```
IDesignGroupChange = interface;
```

C++

```
__interface IDesignGroupChange;
```

File

DesignEditors

Description

This is class DesignEditors.IDesignGroupChange.

59.2.1.1 IDesignGroupChange Members

The following tables list the members exposed by IDesignGroupChange.

Public Methods

	Name	Description
	FreeEditorGroup (see page 11089)	This is FreeEditorGroup, a member of class IDesignGroupChange.

59.2.1.2 IDesignGroupChange Methods

The methods of the IDesignGroupChange class are listed here.

Public Methods

	Name	Description
	FreeEditorGroup (see page 11089)	This is FreeEditorGroup, a member of class IDesignGroupChange.

59.2.1.2.1 IDesignGroupChange.FreeEditorGroup Method

Pascal

```
procedure FreeEditorGroup(Group: Integer);
```

C++

```
__fastcall FreeEditorGroup(int Group);
```

Description

This is FreeEditorGroup, a member of class IDesignGroupChange.

59.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AncestorNameMatches (see page 11090)	This is function DesignEditors.AncestorNameMatches.
	ClassInheritsFrom (see page 11090)	This is function DesignEditors.ClassInheritsFrom.
	FreeEditorGroup (see page 11091)	This is function DesignEditors.FreeEditorGroup.
	GetComponentEditor (see page 11091)	This is function DesignEditors.GetComponentEditor.
	GetComponentProperties (see page 11091)	This is function DesignEditors.GetComponentProperties.
	GetSelectionEditors (see page 11092)	This is function DesignEditors.GetSelectionEditors.
	GetSelectionEditors (see page 11092)	This is function DesignEditors.GetSelectionEditors.
	GetSelectionEditors (see page 11092)	This is function DesignEditors.GetSelectionEditors.
	GuidelinesFor (see page 11093)	This is function DesignEditors.GuidelinesFor.
	NewEditorGroup (see page 11093)	This is function DesignEditors.NewEditorGroup.
	NotifyGroupChange (see page 11093)	This is function DesignEditors.NotifyGroupChange.
	PossibleStream (see page 11094)	This is function DesignEditors.PossibleStream.
	UnnotifyGroupChange (see page 11094)	This is function DesignEditors.UnnotifyGroupChange.

59.3.1 DesignEditors.AncestorNameMatches Function

Pascal

```
function AncestorNameMatches(ClassType: TClass; AncestorClass: TClass): Boolean;
```

C++

```
Boolean AncestorNameMatches(TClass ClassType, TClass AncestorClass);
```

File

DesignEditors

Description

This is function DesignEditors.AncestorNameMatches.

59.3.2 DesignEditors.ClassInheritsFrom Function

Pascal

```
function ClassInheritsFrom(ClassType: TClass; const ClassName: string): Boolean;
```

C++

```
Boolean ClassInheritsFrom(TClass ClassType, const AnsiString ClassName);
```

File

DesignEditors

Description

This is function DesignEditors.ClassInheritsFrom.

59.3.3 DesignEditors.FreeEditorGroup Function

Pascal

```
procedure FreeEditorGroup(Group: Integer);
```

C++

```
FreeEditorGroup(int Group);
```

File

DesignEditors

Description

This is function DesignEditors.FreeEditorGroup.

59.3.4 DesignEditors.GetComponentEditor Function

Pascal

```
function GetComponentEditor(Component: TComponent; const Designer: IDesigner):
IComponentEditor;
```

C++

```
IComponentEditor GetComponentEditor(TComponent * Component, const IDesigner Designer);
```

File

DesignEditors

Description

This is function DesignEditors.GetComponentEditor.

59.3.5 DesignEditors.GetComponentProperties Function

Pascal

```
procedure GetComponentProperties(const Components: IDesignerSelections; Filter: TTypeKinds;
const Designer: IDesigner; Proc: TGetPropProc; EditorFilterFunc: TPropertyEditorFilterFunc =
nil);
```

C++

```
GetComponentProperties(const IDesignerSelections Components, TTypeKinds Filter, const
IDesigner Designer, TGetPropProc Proc, TPropertyEditorFilterFunc EditorFilterFunc = nil);
```

File

DesignEditors

Description

This is function DesignEditors.GetComponentProperties.

59.3.6 DesignEditors.GetSelectionEditors Function

Pascal

```
function GetSelectionEditors(const Designer: IDesigner): ISelectionEditorList; overload;
```

C++

```
ISelectionEditorList GetSelectionEditors(const IDesigner Designer);
```

File

DesignEditors

Description

This is function DesignEditors.GetSelectionEditors.

59.3.7 DesignEditors.GetSelectionEditors Function

Pascal

```
function GetSelectionEditors(const Designer: IDesigner; const Selections:
IDesignerSelections): ISelectionEditorList; overload;
```

C++

```
ISelectionEditorList GetSelectionEditors(const IDesigner Designer, const IDesignerSelections
Selections);
```

File

DesignEditors

Description

This is function DesignEditors.GetSelectionEditors.

59.3.8 DesignEditors.GetSelectionEditors Function

Pascal

```
function GetSelectionEditors(const Designer: IDesigner; Component: TComponent):
ISelectionEditorList; overload;
```

C++

```
ISelectionEditorList GetSelectionEditors(const IDesigner Designer, TComponent * Component);
```

File

DesignEditors

Description

This is function DesignEditors.GetSelectionEditors.

59.3.9 DesignEditors.GuidelinesFor Function

Pascal

```
function GuidelinesFor(const ADesigner: IDesigner; AComponent: TComponent; AContainer:
TComponent): IComponentGuidelines;
```

C++

```
IComponentGuidelines GuidelinesFor(const IDesigner ADesigner, TComponent * AComponent,
TComponent * AContainer);
```

File

DesignEditors

Description

This is function DesignEditors.GuidelinesFor.

59.3.10 DesignEditors.NewEditorGroup Function

Pascal

```
function NewEditorGroup: Integer;
```

C++

```
int NewEditorGroup();
```

File

DesignEditors

Description

This is function DesignEditors.NewEditorGroup.

59.3.11 DesignEditors.NotifyGroupChange Function

Pascal

```
procedure NotifyGroupChange(AProc: TGroupChangeProc);
```

C++

```
NotifyGroupChange(TGroupChangeProc AProc);
```

File

DesignEditors

Description

This is function DesignEditors.NotifyGroupChange.

59.3.12 DesignEditors.PossibleStream Function

Pascal

```
function PossibleStream(const S: string): Boolean;
```

C++

```
Boolean PossibleStream(const AnsiString S);
```

File

DesignEditors

Description

This is function DesignEditors.PossibleStream.

59.3.13 DesignEditors.UnnotifyGroupChange Function

Pascal

```
procedure UnnotifyGroupChange(AProc: TGroupChangeProc);
```

C++

```
UnnotifyGroupChange(TGroupChangeProc AProc);
```

File

DesignEditors

Description

This is function DesignEditors.UnnotifyGroupChange.

59.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TInstProp (see page 11095)	This is record DesignEditors.TInstProp.

59.4.1 DesignEditors.TInstProp Record

Pascal

```
TInstProp = record
  Instance: TPersistent;
  PropInfo: PPropInfo;
end;
```

C++

```
struct TInstProp {
  TPersistent * Instance;
  PPropInfo PropInfo;
};
```

File

DesignEditors

Description

This is record DesignEditors.TInstProp.

59.5 Types

The following table lists types in this documentation.

Types

Name	Description
PInstPropList (see page 11096)	This is type DesignEditors.PInstPropList.
TBit (see page 11096)	This is type DesignEditors.TBit.
TGetTopLevelComponentFunc (see page 11097)	This is type DesignEditors.TGetTopLevelComponentFunc.
TGroupChangeProc (see page 11097)	This is type DesignEditors.TGroupChangeProc.
TInstPropList (see page 11097)	This is type DesignEditors.TInstPropList.

59.5.1 DesignEditors.PInstPropList Type

Pascal

```
PInstPropList = ^TInstPropList;
```

C++

```
^TInstPropList PInstPropList;
```

File

DesignEditors

Description

This is type DesignEditors.PInstPropList.

59.5.2 DesignEditors.TBit Type

Pascal

```
TBit = 0..31;
```

C++

```
0..31 TBit;
```

File

DesignEditors

Description

This is type DesignEditors.TBit.

59.5.3 DesignEditors.TGetTopLevelComponentFunc Type

Pascal

```
TGetTopLevelComponentFunc = function (Ignoring: TComponent = nil): TComponent;
```

C++

```
(Ignoring: TComponent = nil): TComponent ( TGetTopLevelComponentFunc)();
```

File

DesignEditors

Description

This is type DesignEditors.TGetTopLevelComponentFunc.

59.5.4 DesignEditors.TGroupChangeProc Type

Pascal

```
TGroupChangeProc = procedure (AGroup: Integer);
```

C++

```
(AGroup: Integer) ( TGroupChangeProc)();
```

File

DesignEditors

Description

This is type DesignEditors.TGroupChangeProc.

59.5.5 DesignEditors.TInstPropList Type

Pascal

```
TInstPropList = array[0..1023] of TInstProp;
```

C++

```
array[0..1023] of TInstProp TInstPropList;
```

File

DesignEditors

Description

This is type DesignEditors.TInstPropList.

59.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
GetTopLevelComponentFunc (see page 11098)	This is variable DesignEditors.GetTopLevelComponentFunc.

59.6.1 DesignEditors.GetTopLevelComponentFunc Variable

Pascal

```
GetTopLevelComponentFunc: TGetTopLevelComponentFunc;
```

C++

```
TGetTopLevelComponentFunc GetTopLevelComponentFunc;
```

File

DesignEditors

Description

This is variable DesignEditors.GetTopLevelComponentFunc.

59.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
sClassNotApplicable (see page 11099)	This is constant DesignEditors.sClassNotApplicable.
sNotAvailable (see page 11099)	This is constant DesignEditors.sNotAvailable.

59.7.1 DesignEditors.sClassNotApplicable Constant

Pascal

```
sClassNotApplicable = 'Class %s is not applicable to this module';
```

C++

```
sClassNotApplicable = 'Class %s is not applicable to this module';
```

File

DesignEditors

Description

This is constant DesignEditors.sClassNotApplicable.

59.7.2 DesignEditors.sNotAvailable Constant

Pascal

```
sNotAvailable = '(Not available)';
```

C++

```
sNotAvailable = '(Not available)';
```

File

DesignEditors

Description

This is constant DesignEditors.sNotAvailable.

60 DesignIntf Namespace

This is namespace DesignIntf.

Topics

Name	Description
Classes (see page 11105)	The following table lists classes in this documentation.
Interfaces (see page 11134)	The following table lists interfaces in this documentation.
Functions (see page 11224)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 11233)	The following table lists structs, records, enums in this documentation.
Types (see page 11238)	The following table lists types in this documentation.
Variables (see page 11249)	The following table lists variables in this documentation.
Constants (see page 11255)	The following table lists constants in this documentation.

Classes

	Name	Description
	TBaseComponentEditor (see page 11105)	TBaseComponentEditor is the base class for all component editors.
	TBaseComponentGuidelines (see page 11107)	This is class DesignIntf.TBaseComponentGuidelines.
	TBaseCustomModule (see page 11110)	This is class DesignIntf.TBaseCustomModule.
	TBasePropertyEditor (see page 11112)	TBasePropertyEditor is the base class for all property editors.
	TBaseSelectionEditor (see page 11116)	This is class DesignIntf.TBaseSelectionEditor.
	TClassWrapper (see page 11117)	This is class DesignIntf.TClassWrapper.
	TDesignerSelections (see page 11121)	TDesignerSelections maintains a list of persistent objects.
	TDragTarget (see page 11127)	This is class DesignIntf.TDragTarget.
	TEventInfo (see page 11129)	This is class DesignIntf.TEventInfo.

Constants

Name	Description
MaxIdentLength (see page 11255)	This is constant DesignIntf.MaxIdentLength.
sActionCategoryName (see page 11256)	This is constant DesignIntf.sActionCategoryName.
sAppearanceCategoryName (see page 11256)	This is constant DesignIntf.sAppearanceCategoryName.
sBehaviorCategoryName (see page 11256)	This is constant DesignIntf.sBehaviorCategoryName.
sDatabaseCategoryName (see page 11257)	This is constant DesignIntf.sDatabaseCategoryName.
sDataCategoryName (see page 11257)	This is constant DesignIntf.sDataCategoryName.
sDesignCategoryName (see page 11257)	This is constant DesignIntf.sDesignCategoryName.
sDragNDropCategoryName (see page 11258)	This is constant DesignIntf.sDragNDropCategoryName.
sFocusCategoryName (see page 11258)	This is constant DesignIntf.sFocusCategoryName.

sHelpCategoryName (see page 11258)	This is constant DesignIntf.sHelpCategoryName.
sInputCategoryName (see page 11259)	This is constant DesignIntf.sInputCategoryName.
sInvalidFilter (see page 11259)	This is constant DesignIntf.sInvalidFilter.
sKeyCategoryName (see page 11259)	This is constant DesignIntf.sKeyCategoryName.
sLayoutCategoryName (see page 11260)	This is constant DesignIntf.sLayoutCategoryName.
sLegacyCategoryName (see page 11260)	This is constant DesignIntf.sLegacyCategoryName.
sLinkageCategoryName (see page 11260)	This is constant DesignIntf.sLinkageCategoryName.
sLocaleCategoryName (see page 11261)	This is constant DesignIntf.sLocaleCategoryName.
sLocalizableCategoryName (see page 11261)	This is constant DesignIntf.sLocalizableCategoryName.
sMiscellaneousCategoryName (see page 11261)	This is constant DesignIntf.sMiscellaneousCategoryName.
sMouseCategoryName (see page 11262)	This is constant DesignIntf.sMouseCategoryName.
sPropertyChangeCategoryName (see page 11262)	This is constant DesignIntf.sPropertyChangeCategoryName.
sVisualCategoryName (see page 11262)	This is constant DesignIntf.sVisualCategoryName.
sWindowStyleName (see page 11263)	This is constant DesignIntf.sWindowStyleName.

Functions

	Name	Description
	CreateSelectionList (see page 11224)	This is function DesignIntf.CreateSelectionList.
	EnableDemandLoadReport (see page 11225)	This is function DesignIntf.EnableDemandLoadReport.
	ForceDemandLoadState (see page 11225)	This is function DesignIntf.ForceDemandLoadState.
	PersistentToDesignObject (see page 11225)	This is function DesignIntf.PersistentToDesignObject.
	RegisterComponentEditor (see page 11226)	This is function DesignIntf.RegisterComponentEditor.
	RegisterComponentGuidelines (see page 11226)	This is function DesignIntf.RegisterComponentGuidelines.
	RegisterCustomModule (see page 11226)	This is function DesignIntf.RegisterCustomModule.
	RegisterDesignDragObject (see page 11227)	This is function DesignIntf.RegisterDesignDragObject.
	RegisterDesignNotification (see page 11227)	This is function DesignIntf.RegisterDesignNotification.
	RegisterDragTarget (see page 11227)	This is function DesignIntf.RegisterDragTarget.
	RegisterIDropTarget (see page 11228)	This is function DesignIntf.RegisterIDropTarget.
	RegisterPropertiesInCategory (see page 11228)	This is function DesignIntf.RegisterPropertiesInCategory.
	RegisterPropertiesInCategory (see page 11228)	This is function DesignIntf.RegisterPropertiesInCategory.
	RegisterPropertiesInCategory (see page 11229)	This is function DesignIntf.RegisterPropertiesInCategory.
	RegisterPropertyEditor (see page 11229)	This is function DesignIntf.RegisterPropertyEditor.
	RegisterPropertyInCategory (see page 11229)	This is function DesignIntf.RegisterPropertyInCategory.
	RegisterPropertyInCategory (see page 11230)	This is function DesignIntf.RegisterPropertyInCategory.
	RegisterPropertyInCategory (see page 11230)	This is function DesignIntf.RegisterPropertyInCategory.
	RegisterPropertyInCategory (see page 11230)	This is function DesignIntf.RegisterPropertyInCategory.
	RegisterPropertyMapper (see page 11231)	This is function DesignIntf.RegisterPropertyMapper.
	RegisterSelectionEditor (see page 11231)	This is function DesignIntf.RegisterSelectionEditor.
	SetPropertyEditorGroup (see page 11231)	This is function DesignIntf.SetPropertyEditorGroup.
	UnlistPublishedProperty (see page 11232)	This is function DesignIntf.UnlistPublishedProperty.
	UnregisterDesignNotification (see page 11232)	This is function DesignIntf.UnregisterDesignNotification.

Interfaces

	Name	Description
	IActivatable (see page 11134)	This is class DesignIntf.IActivatable.
	IClass (see page 11135)	This is class DesignIntf.IClass.
	IComponentEditor (see page 11138)	IComponentEditor is the interface that the form designer uses to communicate with component editors.
	IComponentGuidelines (see page 11143)	This is class DesignIntf.IComponentGuidelines.
	ICustomModule (see page 11145)	This is class DesignIntf.ICustomModule.
	IDefaultEditor (see page 11148)	This is class DesignIntf.IDefaultEditor.
	IDesigner (see page 11149)	IDesigner is an interface to the form designer in the IDE.
	IDesigner60 (see page 11151)	This is class DesignIntf.IDesigner60.
	IDesigner70 (see page 11167)	This is class DesignIntf.IDesigner70.
	IDesigner80 (see page 11169)	This is class DesignIntf.IDesigner80.

	IDesignerSelections (see page 11172)	IDesignerSelections represents a list of persistent objects.
	IDesignNotification (see page 11176)	This is class DesignIntf.IDesignNotification.
	IDesignObject (see page 11178)	This is class DesignIntf.IDesignObject.
	IDesignPersistent (see page 11183)	This is class DesignIntf.IDesignPersistent.
	IDesignWindow (see page 11184)	This is class DesignIntf.IDesignWindow.
	IEditHandler (see page 11186)	This is class DesignIntf.IEditHandler.
	IEventInfo (see page 11187)	This is class DesignIntf.IEventInfo.
	IHostForm (see page 11189)	This is class DesignIntf.IHostForm.
	IImplementation (see page 11196)	This is class DesignIntf.IImplementation.
	IMethodProperty (see page 11197)	This is class DesignIntf.IMethodProperty.
	IProperty (see page 11197)	IProperty is the interface that the Object Inspector uses to communicate with property editors.
	IProperty70 (see page 11205)	This is class DesignIntf.IProperty70.
	IProperty80 (see page 11207)	This is class DesignIntf.IProperty80.
	IPropertyControl (see page 11208)	This is class DesignIntf.IPropertyControl.
	IPropertyDescription (see page 11209)	This is class DesignIntf.IPropertyDescription.
	IPropertyHost (see page 11210)	This is class DesignIntf.IPropertyHost.
	IPropertyKind (see page 11211)	This is class DesignIntf.IPropertyKind.
	IReferenceProperty (see page 11212)	This is class DesignIntf.IReferenceProperty.
	ISelectionEditor (see page 11213)	This is class DesignIntf.ISelectionEditor.
	ISelectionEditorList (see page 11215)	This is class DesignIntf.ISelectionEditorList.
	ISelectionPropertyFilter (see page 11217)	This is class DesignIntf.ISelectionPropertyFilter.
	IShowReferenceProperty (see page 11218)	This is class DesignIntf.IShowReferenceProperty.
	IWideProperty (see page 11219)	This is class DesignIntf.IWideProperty.
	IWideProperty10 (see page 11221)	This is class DesignIntf.IWideProperty10.
	IWidePropertyDescription (see page 11222)	This is class DesignIntf.IWidePropertyDescription.

Structs, Records, Enums

	Name	Description
	TCustomModuleAttribute (see page 11233)	This is record DesignIntf.TCustomModuleAttribute.
	TDemandLoadState (see page 11233)	This is record DesignIntf.TDemandLoadState.
	TDesignerGuideType (see page 11234)	This is record DesignIntf.TDesignerGuideType.
	TEditAction (see page 11234)	This is record DesignIntf.TEditAction.
	TEditStates (see page 11235)	This is record DesignIntf.TEditStates.
	TPropertyAttribute (see page 11236)	This is record DesignIntf.TPropertyAttribute.
	TPropKind (see page 11237)	This is record DesignIntf.TPropKind.

Types

Name	Description
IDesignEditQuery (see page 11238)	This is type DesignIntf.IDesignEditQuery.
TComponentEditorClass (see page 11239)	This is type DesignIntf.TComponentEditorClass.
TComponentGuidelinesClass (see page 11239)	This is type DesignIntf.TComponentGuidelinesClass.
TCustomModuleAttributes (see page 11239)	This is type DesignIntf.TCustomModuleAttributes.
TCustomModuleClass (see page 11240)	This is type DesignIntf.TCustomModuleClass.
TDragTargetClass (see page 11240)	This is type DesignIntf.TDragTargetClass.
TEditState (see page 11240)	This is type DesignIntf.TEditState.
TEnableDemandLoadReportProc (see page 11241)	This is type DesignIntf.TEnableDemandLoadReportProc.
TForceDemandLoadStateProc (see page 11241)	This is type DesignIntf.TForceDemandLoadStateProc.
TGetDesignerEvent (see page 11241)	This is type DesignIntf.TGetDesignerEvent.
TGetPropProc (see page 11242)	This is type DesignIntf.TGetPropProc.
TGetWideStrProc (see page 11242)	This is type DesignIntf.TGetWideStrProc.
TPropertyAttributes (see page 11242)	This is type DesignIntf.TPropertyAttributes.
TPropertyEditorClass (see page 11243)	This is type DesignIntf.TPropertyEditorClass.
TPropertyEditorFilterFunc (see page 11243)	This is type DesignIntf.TPropertyEditorFilterFunc.
TPropertyMapperFunc (see page 11243)	This is type DesignIntf.TPropertyMapperFunc.

TRegisterComponentEditorProc (see page 11244)	This is type DesignIntf.TRegisterComponentEditorProc.
TRegisterComponentGuidelines (see page 11244)	This is type DesignIntf.TRegisterComponentGuidelines.
TRegisterCustomModuleProc (see page 11244)	This is type DesignIntf.TRegisterCustomModuleProc.
TRegisterDesignDragObject (see page 11245)	This is type DesignIntf.TRegisterDesignDragObject.
TRegisterDesignNotification (see page 11245)	This is type DesignIntf.TRegisterDesignNotification.
TRegisterDragTargetProc (see page 11245)	This is type DesignIntf.TRegisterDragTargetProc.
TRegisterIDropTarget (see page 11246)	This is type DesignIntf.TRegisterIDropTarget.
TRegisterPropertyEditorProc (see page 11246)	This is type DesignIntf.TRegisterPropertyEditorProc.
TRegisterPropertyInCategoryProc (see page 11246)	This is type DesignIntf.TRegisterPropertyInCategoryProc.
TRegisterPropertyMapperProc (see page 11247)	This is type DesignIntf.TRegisterPropertyMapperProc.
TRegisterSelectionEditorProc (see page 11247)	This is type DesignIntf.TRegisterSelectionEditorProc.
TSelectionEditorClass (see page 11247)	This is type DesignIntf.TSelectionEditorClass.
TSetPropertyEditorGroupProc (see page 11248)	This is type DesignIntf.TSetPropertyEditorGroupProc.

Variables

Name	Description
EnableDemandLoadReportProc (see page 11249)	This is variable DesignIntf.EnableDemandLoadReportProc.
ForceDemandLoadStateProc (see page 11249)	This is variable DesignIntf.ForceDemandLoadStateProc.
GReferenceExpandable (see page 11250)	This is variable DesignIntf.GReferenceExpandable.
GShowReadOnlyProps (see page 11250)	This is variable DesignIntf.GShowReadOnlyProps.
RegisterComponentEditorProc (see page 11250)	This is variable DesignIntf.RegisterComponentEditorProc.
RegisterComponentGuidelinesProc (see page 11251)	This is variable DesignIntf.RegisterComponentGuidelinesProc.
RegisterCustomModuleProc (see page 11251)	This is variable DesignIntf.RegisterCustomModuleProc.
RegisterDesignDragObjectProc (see page 11251)	This is variable DesignIntf.RegisterDesignDragObjectProc.
RegisterDesignNotificationProc (see page 11252)	This is variable DesignIntf.RegisterDesignNotificationProc.
RegisterDragTargetProc (see page 11252)	This is variable DesignIntf.RegisterDragTargetProc.
RegisterIDropTargetProc (see page 11252)	This is variable DesignIntf.RegisterIDropTargetProc.
RegisterPropertyEditorProc (see page 11253)	This is variable DesignIntf.RegisterPropertyEditorProc.
RegisterPropertyInCategoryProc (see page 11253)	This is variable DesignIntf.RegisterPropertyInCategoryProc.
RegisterPropertyMapperProc (see page 11253)	This is variable DesignIntf.RegisterPropertyMapperProc.
RegisterSelectionEditorProc (see page 11254)	This is variable DesignIntf.RegisterSelectionEditorProc.
SetPropertyEditorGroupProc (see page 11254)	This is variable DesignIntf.SetPropertyEditorGroupProc.
UnregisterDesignNotificationProc (see page 11254)	This is variable DesignIntf.UnregisterDesignNotificationProc.

60.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TBaseComponentEditor (see page 11105)	TBaseComponentEditor is the base class for all component editors.
	TBaseComponentGuidelines (see page 11107)	This is class DesignIntf.TBaseComponentGuidelines.
	TBaseCustomModule (see page 11110)	This is class DesignIntf.TBaseCustomModule.
	TBasePropertyEditor (see page 11112)	TBasePropertyEditor is the base class for all property editors.
	TBaseSelectionEditor (see page 11116)	This is class DesignIntf.TBaseSelectionEditor.
	TClassWrapper (see page 11117)	This is class DesignIntf.TClassWrapper.
	TDesignerSelections (see page 11121)	TDesignerSelections maintains a list of persistent objects.
	TDragTarget (see page 11127)	This is class DesignIntf.TDragTarget.
	TEventInfo (see page 11129)	This is class DesignIntf.TEventInfo.

60.1.1 TBaseComponentEditor Class

TBaseComponentEditor is the base class for all component editors.

Class Hierarchy

Pascal

```
TBaseComponentEditor = class(TInterfacedObject);
```

C++

```
class TBaseComponentEditor : public TInterfacedObject;
```

File

DesignIntf

Description

Use TBaseComponentEditor or one of its descendants as a base class when defining a design-time editor for working with a new component class. All component editors must descend from TBaseComponentEditor and, in addition, they must support the IComponentEditor (see page 11138) interface.

Create (see page 11107) a new component editor class for a component class to add verbs to its context menu, change its default double-click behavior, or allow the component to use a new clipboard format.

TBaseComponentEditor does not implement any support for the IComponentEditor (see page 11138) interface. This support must be supplied by descendant classes. For example, TBaseComponentEditor has an immediate descendant, TComponentEditor, which supplies a default implementation of the IComponentEditor (see page 11138) interface. Typically, component writers use TComponentEditor or TDefaultEditor as a base class when defining new component editors. However, you can derive a component editor directly from TBaseComponentEditor if you want to provide your own implementation of IComponentEditor (see page 11138)..

Once the component editor is defined, it can be registered to work with a particular component class. A registered component editor is created for each component of that class when it is selected in the form designer. To create the association between a component editor and a component class, call RegisterComponentEditor (see page 11226).

See Also

RegisterComponentEditor (see page 11226)

IComponentEditor (see page 11138)

TComponentEditor (see page 10927)

60.1.1.1 TBaseComponentEditor Members

The following tables list the members exposed by TBaseComponentEditor.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11107)	Creates a new TBaseComponentEditor (see page 11105) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

60.1.1.2 TBaseComponentEditor.Create Constructor

Creates a new TBaseComponentEditor (see page 11105) object.

Pascal

```
constructor Create(AComponent: TComponent; ADesigner: IDesigner); virtual;
```

C++

```
virtual __fastcall TBaseComponentEditor(TComponent * AComponent, IDesigner ADesigner);
```

Description

Do not create instances of a component editor. Registered component editors are created automatically for selected objects in the form designer.

AComponent is the instance of the component for which the editor is created.

ADesigner is an interface that the component editor can use to interact with the forms designer.

60.1.2 TBaseComponentGuidelines Class

Class Hierarchy

Pascal

```
TBaseComponentGuidelines = class(TInterfacedObject);
```

C++

```
class TBaseComponentGuidelines : public TInterfacedObject;
```

File

DesignIntf

Description

This is class DesignIntf.TBaseComponentGuidelines.

60.1.2.1 TBaseComponentGuidelines Members

The following tables list the members exposed by TBaseComponentGuidelines.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11109)	This is Create, a member of class TBaseComponentGuidelines.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
A	Initialize (see page 11109)	This is Initialize, a member of class TBaseComponentGuidelines.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Designer (see page 11110)	This is Designer, a member of class TBaseComponentGuidelines.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

60.1.2.2 TBaseComponentGuidelines.Create Constructor

Pascal

```
constructor Create(const ADesigner: IDesigner); virtual;
```

C++

```
virtual __fastcall TBaseComponentGuidelines(const IDesigner ADesigner);
```

Description

This is Create, a member of class TBaseComponentGuidelines.

60.1.2.3 TBaseComponentGuidelines Methods

The methods of the TBaseComponentGuidelines class are listed here.

Public Methods

	Name	Description
A	Initialize (see page 11109)	This is Initialize, a member of class TBaseComponentGuidelines.

60.1.2.3.1 TBaseComponentGuidelines.Initialize Method

Pascal

```
procedure Initialize(AComponent: TComponent; AContainer: TComponent); virtual; abstract;
```

C++

```
virtual __fastcall Initialize(TComponent * AComponent, TComponent * AContainer) = 0;
```

Description

This is Initialize, a member of class TBaseComponentGuidelines.

60.1.2.4 TBaseComponentGuidelines Properties

The properties of the TBaseComponentGuidelines class are listed here.

Public Properties

	Name	Description
R	Designer (see page 11110)	This is Designer, a member of class TBaseComponentGuidelines.

60.1.2.4.1 TBaseComponentGuidelines.Designer Property

Pascal

```
property Designer: IDesigner;
```

C++

```
__property IDesigner Designer;
```

Description

This is Designer, a member of class TBaseComponentGuidelines.

60.1.3 TBaseCustomModule Class

Class Hierarchy

Pascal

```
TBaseCustomModule = class(TInterfacedObject);
```

C++

```
class TBaseCustomModule : public TInterfacedObject;
```

File

DesignIntf

Description

This is class DesignIntf.TBaseCustomModule.

60.1.3.1 TBaseCustomModule Members

The following tables list the members exposed by TBaseCustomModule.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.

V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11111)	This is Create, a member of class TBaseCustomModule.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	DesignClass (see page 11112)	This is DesignClass, a member of class TBaseCustomModule.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

60.1.3.2 TBaseCustomModule.Create Constructor

Pascal

```
constructor Create(ARoot: TComponent; const Designer: IDesigner); virtual;
```

C++

```
virtual __fastcall TBaseCustomModule(TComponent * ARoot, const IDesigner Designer);
```

Description

This is Create, a member of class TBaseCustomModule.

60.1.3.3 TBaseCustomModule Methods

The methods of the TBaseCustomModule class are listed here.

Public Methods

	Name	Description
	DesignClass (see page 11112)	This is DesignClass, a member of class TBaseCustomModule.

60.1.3.3.1 TBaseCustomModule.DesignClass Method

Pascal

```
class function DesignClass: TComponentClass; virtual;
```

C++

```
virtual __fastcall TComponentClass DesignClass();
```

Description

This is DesignClass, a member of class TBaseCustomModule.

60.1.4 TBasePropertyEditor Class

TBasePropertyEditor is the base class for all property editors.

Class Hierarchy

Pascal

```
TBasePropertyEditor = class(TInterfacedObject);
```

C++

```
class TBasePropertyEditor : public TInterfacedObject;
```

File

DesignIntf

Description

Use TBasePropertyEditor or one of its descendants as a base class when defining a property editor for a specialized property in a new component class. All property editors must descend from TBasePropertyEditor and, in addition, they must support the IProperty (see page 11197) interface.

TBasePropertyEditor defines two abstract (pure virtual) protected methods that the Object Inspector calls to inform the property editor of information it may find useful. It does not implement any support for the IProperty (see page 11197) interface. This support must be supplied by descendant classes. For example, TBasePropertyEditor has an immediate descendant, TPropertyEditor, which supplies a default implementation of the IProperty (see page 11197) interface as well as support for fetching property values of different types. Typically, component writers use TPropertyEditor as a base class when defining new property editors. However, you can derive a property editor directly from TBasePropertyEditor, and avoid the overhead of all methods that handle types that are not needed for your particular property.

Property editors can be associated with all properties of a specified type, or with a particular property of a particular component

class. To create the association between a property editor and a property type, call RegisterPropertyEditor (see page 11229). Once the property editor is registered, it appears when the user attempts to edit an appropriate property using the Object Inspector.

See Also

RegisterPropertyEditor (see page 11229)

IProperty (see page 11197)

TPropertyEditor (see page 11019)

60.1.4.1 TBasePropertyEditor Members

The following tables list the members exposed by TBasePropertyEditor.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
A	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11114)	Creates a new TBasePropertyEditor (see page 11112) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

60.1.4.2 TBasePropertyEditor.Create Constructor

Creates a new TBasePropertyEditor (see page 11112) object.

Pascal

```
constructor Create(const ADesigner: IDesigner; APropCount: Integer); virtual;
```

C++

```
virtual __fastcall TBasePropertyEditor(const IDesigner ADesigner, int APropCount);
```

Description

Do not create instances of a property editor. Registered property editors are created automatically for selected objects in the form designer.

ADesigner is an interface that the property editor can use to interact with the forms designer.

APropCount indicates the number of components whose properties are to be edited using the property editor.

See Also

Initialize (see page 11114)

60.1.4.3 TBasePropertyEditor Methods

The methods of the TBasePropertyEditor class are listed here.

Protected Methods

	Name	Description
A	Initialize (see page 11114)	Initializes the property editor after it is created but before it is used.
A	SetPropEntry (see page 11115)	Informs the property editor of a particular object whose property is to be edited.

60.1.4.3.1 TBasePropertyEditor.Initialize Method

Initializes the property editor after it is created but before it is used.

Pascal

```
procedure Initialize; virtual; abstract;
```

C++

```
virtual __fastcall Initialize() = 0;
```

Description

Do not call the Initialize method. It is called automatically by the Object Inspector to allow property editors to perform any necessary initialization before they are used.

In TBasePropertyEditor (see page 11112), Initialize is abstract (pure virtual). Descendant classes must override the Initialize method to provide an implementation. This method can be used to perform any special initialization, such as the creation of helper objects.

Property editors are often created and discarded without being used because they do not represent a common property across all selected components. Since Initialize is called only after it is determined that the property editor will be used, it is more efficient to initialize property values and create helper objects in the Initialize method rather than the constructor.

See Also

Create (see page 11114)

60.1.4.3.2 TBasePropertyEditor.SetPropEntry Method

Informs the property editor of a particular object whose property is to be edited.

Pascal

```
procedure SetPropEntry(Index: Integer; AInstance: TPersistent; APropInfo: PPropInfo); virtual;
abstract;
```

C++

```
virtual __fastcall SetPropEntry(int Index, TPersistent * AInstance, PPropInfo APropInfo) = 0;
```

Description

The Object Inspector calls SetPropEntry for each object that has a property that the property editor is editing. This allows the property editor to store information about the objects being edited so that it can get and set property values.

Index is the index of the object being edited. This value can range from 0 to one less than the APropCount parameter that is passed to the constructor.

AInstance is a reference to the object being edited.

APropInfo is a pointer to the property information record for the property on AInstance. This can be used to access the methods that get or set the property value.

In TBasePropertyEditor (see page 11112), SetPropEntry is abstract (pure virtual). Descendant classes must override this method to provide an implementation.

See Also

Create (see page 11114)

60.1.5 TBaseSelectionEditor Class

Class Hierarchy

Pascal

```
TBaseSelectionEditor = class(TInterfacedObject);
```

C++

```
class TBaseSelectionEditor : public TInterfacedObject;
```

File

DesignIntf

Description

This is class DesignIntf.TBaseSelectionEditor.

60.1.5.1 TBaseSelectionEditor Members

The following tables list the members exposed by TBaseSelectionEditor.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11117)	This is Create, a member of class TBaseSelectionEditor.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

60.1.5.2 TBaseSelectionEditor.Create Constructor

Pascal

```
constructor Create(const ADesigner: IDesigner); virtual;
```

C++

```
virtual __fastcall TBaseSelectionEditor(const IDesigner ADesigner);
```

Description

This is Create, a member of class TBaseSelectionEditor.

60.1.6 TClassWrapper Class

Class Hierarchy

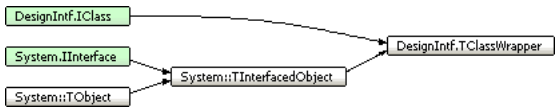

Pascal

```
TClassWrapper = class(TInterfacedObject, IClass);
```

C++

```
class TClassWrapper : public TInterfacedObject, public IClass;
```

File

DesignIntf

Description

This is class DesignIntf.TClassWrapper.

60.1.6.1 TClassWrapper Members

The following tables list the members exposed by TClassWrapper.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	ClassNameIs (see page 11119)	Determines whether an object is of a specific type.
	GetClassName (see page 11120)	This is GetClassName, a member of class TClassWrapper.
	GetClassParent (see page 11120)	This is GetClassParent, a member of class TClassWrapper.
	GetUnitName (see page 11120)	This is GetUnitName, a member of class TClassWrapper.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11119)	This is Create, a member of class TClassWrapper.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ClassName (see page 11137)	This is ClassName, a member of class IClass.
R	ClassParent (see page 11138)	This is ClassParent, a member of class IClass.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	UnitName (see page 11138)	This is UnitName, a member of class IClass.

60.1.6.2 TClassWrapper.Create Constructor

Pascal

```
constructor Create(AClass: TClass);
```

C++

```
__fastcall TClassWrapper(TClass AClass);
```

Description

This is Create, a member of class TClassWrapper.

60.1.6.3 TClassWrapper Methods

The methods of the TClassWrapper class are listed here.

Protected Methods

	Name	Description
	ClassNameIs (see page 11119)	Determines whether an object is of a specific type.
	GetClassName (see page 11120)	This is GetClassName, a member of class TClassWrapper.
	GetClassParent (see page 11120)	This is GetClassParent, a member of class TClassWrapper.
	GetUnitName (see page 11120)	This is GetUnitName, a member of class TClassWrapper.

60.1.6.3.1 TClassWrapper.ClassNameIs Method

Determines whether an object is of a specific type.

Pascal

```
function ClassNameIs(const AClassName: string): Boolean;
```

C++

```
__fastcall Boolean ClassNameIs(const AnsiString AClassName);
```

Description

System::TObject::ClassNameIs determines whether an object instance or class reference has a class name that matches a specified string. This is useful to query objects across modules or shared libraries.

Note: In C++ code, call System::TObject::ClassNameIs as a method to compare an object's class name. Use the global static function to compare the class name from a metaclass object.

See Also

System::TObject::ClassName (see page 22290)

System::TObject::InheritsFrom (see page 22305)

60.1.6.3.2 TClassWrapper.GetClassName Method

Pascal

```
function GetClassName: string;
```

C++

```
__fastcall AnsiString GetClassName();
```

Description

This is GetClassName, a member of class TClassWrapper.

60.1.6.3.3 TClassWrapper.GetClassParent Method

Pascal

```
function GetClassParent: IClass;
```

C++

```
__fastcall IClass GetClassParent();
```

Description

This is GetClassParent, a member of class TClassWrapper.

60.1.6.3.4 TClassWrapper.GetUnitName Method

Pascal

```
function GetUnitName: string;
```

C++

```
__fastcall AnsiString GetUnitName();
```

Description

This is GetUnitName, a member of class TClassWrapper.

60.1.7 TDesignerSelections Class

TDesignerSelections maintains a list of persistent objects.

Class Hierarchy

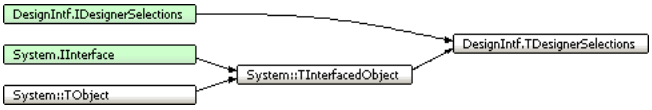

Pascal

```
TDesignerSelections = class(TInterfacedObject, IDesignerSelections);
```

C++

```
class TDesignerSelections : public TInterfacedObject, public IDesignerSelections;
```

File

DesignIntf

Description

The form designer and its property editors and component editors use TDesignerSelections to manage lists of persistent objects in the IDE. For example, the form designer uses a TDesignerSelections to represent the set of components currently selected in the form designer.

TDesignerSelections implements the IDesignerSelections (see page 11172) interface. TDesignerSelections introduces properties and methods to

Add (see page 11124) objects to the list.

Compare two lists of objects.

Access individual objects in the list.

Note: TDesignerSelections does not allow objects to be rearranged or deleted from the list once they are added. To maintain a list of objects that allows the objects to be rearranged or deleted, use the TList object, or the Objects property of a descendant of TStrings.

See Also

TList (see page 2868)

TStrings (see page 3029)

TPersistent (see page 2928)

IDesignerSelections (see page 11172)

TInterfaceList (see page 2855)

60.1.7.1 TDesignerSelections Members

The following tables list the members exposed by TDesignerSelections.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	Add (see page 11124)	Adds a new object to the end of the list.
	Equals (see page 11125)	Indicates whether the selections list is identical to another selections list.
	Get (see page 11125)	Returns a specified item from the list.
	GetCount (see page 11125)	Returns the value of the Count (see page 11126) property.
	GetDesignObject (see page 11126)	This is GetDesignObject, a member of class TDesignerSelections.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Protected Properties

	Name	Description
R	Count (see page 11126)	Indicates the number of entries in the list.
R	Items (see page 11126)	Provides indexed access to the objects in the list.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Copy (see page 11123)	Creates an instance of TDesignerSelections (see page 11121) that is a copy of another list.
V	Create (see page 11123)	Creates an instance of TDesignerSelections (see page 11121).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 11124)	Destroys an instance of TDesignerSelections (see page 11121).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	DesignObjects (see page 11175)	This is DesignObjects, a member of class IDesignerSelections.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

60.1.7.2 TDesignerSelections.Copy Constructor

Creates an instance of TDesignerSelections (see page 11121) that is a copy of another list.

Pascal

```
constructor Copy(const Selections: IDesignerSelections);
```

C++

```
__fastcall TDesignerSelections(const IDesignerSelections Selections);
```

Description

Call Copy to create a TDesignerSelections (see page 11121) object that represents the same list of selected objects as those in the Selections parameter.

See Also

Create (see page 11123)

60.1.7.3 TDesignerSelections.Create Constructor

Creates an instance of TDesignerSelections (see page 11121).

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TDesignerSelections();
```

Description

Call Create to create an empty component list. After creating the list, use the Add (see page 11124) method (via the IDesignerSelections (see page 11172) interface) to add objects to the list.

See Also

Add (see page 11124)

60.1.7.4 TDesignerSelections.Destroy Destructor

Destroys an instance of TDesignerSelections (see page 11121).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDesignerSelections();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TDesignerSelections (see page 11121) reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

60.1.7.5 TDesignerSelections Methods

The methods of the TDesignerSelections class are listed here.

Protected Methods

	Name	Description
	Add (see page 11124)	Adds a new object to the end of the list.
	Equals (see page 11125)	Indicates whether the selections list is identical to another selections list.
	Get (see page 11125)	Returns a specified item from the list.
	GetCount (see page 11125)	Returns the value of the Count (see page 11126) property.
	GetDesignObject (see page 11126)	This is GetDesignObject, a member of class TDesignerSelections.

60.1.7.5.1 TDesignerSelections.Add Method

Adds a new object to the end of the list.

Pascal

```
function Add(const Item: TPersistent): Integer;
```

C++

```
__fastcall int Add(const TPersistent * Item);
```

Description

Call Add to insert a new object at the end of the Items (see page 11126) array. Add returns the index of the new item, where the first item in the list has an index of 0. Add increases the value of Count (see page 11126) to reflect the addition of a new object.

Note: Add is a protected method. To call this method you must use the IDesignerSelections (see page 11172) interface.

See Also

Count (see page 11126)

Items (see page 11126)

60.1.7.5.2 TDesignerSelections.Equals Method

Indicates whether the selections list is identical to another selections list.

Pascal

```
function Equals(const List: IDesignerSelections): Boolean;
```

C++

```
__fastcall Boolean Equals(const IDesignerSelections List);
```

Description

Call Equals to determine whether the list specified by the List parameter contains exactly the same objects in the same order as this list. Equals returns false if the other list

Contains objects not in the Items (see page 11126) array.

Contains a subset of the objects in the Items (see page 11126) array.

Contains the same objects but in a different order.

See Also

Items (see page 11126)

60.1.7.5.3 TDesignerSelections.Get Method

Returns a specified item from the list.

Pascal

```
function Get(Index: Integer): TPersistent;
```

C++

```
__fastcall TPersistent * Get(int Index);
```

Description

Get is the protected read access method for the Items (see page 11126) property.

Index identifies the desired object by its position in the list, where 0 identifies the first position. Get returns the specified object.

See Also

Items (see page 11126)

60.1.7.5.4 TDesignerSelections.GetCount Method

Returns the value of the Count (see page 11126) property.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the protected read access method for the Count (see page 11126) property.

See Also

Count (see page 11126)

60.1.7.5.5 TDesignerSelections.GetDesignObject Method

Pascal

```
function GetDesignObject(Index: Integer): IDesignObject;
```

C++

```
__fastcall IDesignObject GetDesignObject(int Index);
```

Description

This is GetDesignObject, a member of class TDesignerSelections.

60.1.7.6 TDesignerSelections Properties

The properties of the TDesignerSelections class are listed here.

Protected Properties

	Name	Description
R	Count (see page 11126)	Indicates the number of entries in the list.
R	Items (see page 11126)	Provides indexed access to the objects in the list.

60.1.7.6.1 TDesignerSelections.Count Property

Indicates the number of entries in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read Count to determine the number of entries in the Items (see page 11126) array. Use Count as an upper limit when iterating over all the objects in the list.

Count is a read-only property. To increase the number of objects in the list, use the Add (see page 11124) method. Objects can not be removed from the list once they have been added.

Note: Count is a protected property. To access this property you must use the IDesignerSelections (see page 11172) interface.

See Also

Items (see page 11126)

Add (see page 11124)

60.1.7.6.2 TDesignerSelections.Items Property

Provides indexed access to the objects in the list.

Pascal

```
property Items [Index: Integer]: TPersistent;
```

C++

```
__property TPersistent * Items[int Index];
```

Description

Use Items to access a specific object in the array. The Index parameter indicates the index of the object, where 0 is the index of the first object, 1 is the index of the second object, and so on. Use Items with the Count (see page 11126) property to iterate through all of the objects in the list.

To add objects to the Items array, use the Add (see page 11124) method. Once objects have been added, they can't be deleted or rearranged.

Note: Items is a protected property. To access this property you must use the IDesignerSelections (see page 11172) interface.

See Also

Count (see page 11126)

Add (see page 11124)

operator_sb

60.1.8 TDragTarget Class

Class Hierarchy

DesignIntf.TDragTarget

Pascal

```
TDragTarget = class;
```

C++

```
class TDragTarget;
```

File

DesignIntf

Description

This is class DesignIntf.TDragTarget.

60.1.8.1 TDragTarget Members

The following tables list the members exposed by TDragTarget.

Public Methods

	Name	Description
V	Create (see page 11128)	This is Create, a member of class TDragTarget.
A	DragDrop (see page 11128)	This is DragDrop, a member of class TDragTarget.
A	DragOver (see page 11128)	This is DragOver, a member of class TDragTarget.

Public Properties

	Name	Description
	Designer (see page 11129)	This is Designer, a member of class TDragTarget.

60.1.8.2 TDragTarget.Create Constructor

Pascal

```
constructor Create(const ADesigner: IDesigner); virtual;
```

C++

```
virtual __fastcall TDragTarget(const IDesigner ADesigner);
```

Description

This is Create, a member of class TDragTarget.

60.1.8.3 TDragTarget Methods

The methods of the TDragTarget class are listed here.

Public Methods

	Name	Description
	DragDrop (see page 11128)	This is DragDrop, a member of class TDragTarget.
	DragOver (see page 11128)	This is DragOver, a member of class TDragTarget.

60.1.8.3.1 TDragTarget.DragDrop Method

Pascal

```
procedure DragDrop(Target: TObject; Source: TObject; X: Integer; Y: Integer); virtual;
abstract;
```

C++

```
virtual __fastcall DragDrop(TObject * Target, TObject * Source, int X, int Y) = 0;
```

Description

This is DragDrop, a member of class TDragTarget.

60.1.8.3.2 TDragTarget.DragOver Method

Pascal

```
function DragOver(Target: TObject; Source: TObject; X: Integer; Y: Integer; State:
TDragState): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean DragOver(TObject * Target, TObject * Source, int X, int Y,
TDragState State) = 0;
```

Description

This is DragOver, a member of class TDragTarget.

60.1.8.4 TDragTarget Properties

The properties of the TDragTarget class are listed here.

Public Properties

	Name	Description
	Designer (see page 11129)	This is Designer, a member of class TDragTarget.

60.1.8.4.1 TDragTarget.Designer Property

Pascal

```
property Designer: IDesigner;
```

C++

```
__property IDesigner Designer;
```

Description

This is Designer, a member of class TDragTarget.

60.1.9 TEventInfo Class

Class Hierarchy

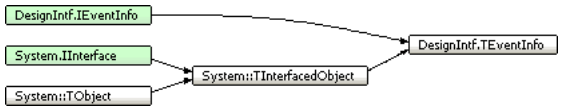

Pascal

```
TEventInfo = class(TInterfacedObject, IEventInfo);
```

C++

```
class TEventInfo : public TInterfacedObject, public IEventInfo;
```

File

DesignIntf

Description

This is class DesignIntf.TEventInfo.

60.1.9.1 TEventInfo Members

The following tables list the members exposed by TEventInfo.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetMethodKind (see page 11132)	This is GetMethodKind, a member of class TEventInfo.
	GetParamCount (see page 11132)	This is GetParamCount, a member of class TEventInfo.
	GetParamFlags (see page 11132)	This is GetParamFlags, a member of class TEventInfo.
	GetParamName (see page 11132)	This is GetParamName, a member of class TEventInfo.
	GetParamType (see page 11132)	This is GetParamType, a member of class TEventInfo.
	GetResultType (see page 11133)	This is GetResultType, a member of class TEventInfo.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11131)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

60.1.9.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TEventInfo.Create (PPropInfo) (see page 11131)	This is Create, a member of class TEventInfo.
	TEventInfo.Create (PTypeData) (see page 11131)	This is Create, a member of class TEventInfo.

60.1.9.2.1 TEventInfo.Create Constructor (PPropInfo)

Pascal

```
constructor Create(APropInfo: PPropInfo); overload;
```

C++

```
__fastcall TEventInfo(PPropInfo APropInfo);
```

Description

This is Create, a member of class TEventInfo.

60.1.9.2.2 TEventInfo.Create Constructor (PTypeData)

Pascal

```
constructor Create(ATypeData: PTypeData); overload;
```

C++

```
__fastcall TEventInfo(PTypeData ATypeData);
```

Description

This is Create, a member of class TEventInfo.

60.1.9.3 TEventInfo Methods

The methods of the TEventInfo class are listed here.

Protected Methods

	Name	Description
	GetMethodKind (see page 11132)	This is GetMethodKind, a member of class TEventInfo.
	GetParamCount (see page 11132)	This is GetParamCount, a member of class TEventInfo.
	GetParamFlags (see page 11132)	This is GetParamFlags, a member of class TEventInfo.
	GetParamName (see page 11132)	This is GetParamName, a member of class TEventInfo.
	GetParamType (see page 11132)	This is GetParamType, a member of class TEventInfo.

	GetResultType (see page 11133)	This is GetResultType, a member of class TEventInfo.

60.1.9.3.1 TEventInfo.GetMethodKind Method

Pascal

```
function GetMethodKind: TMethodKind;
```

C++

```
__fastcall TMethodKind GetMethodKind();
```

Description

This is GetMethodKind, a member of class TEventInfo.

60.1.9.3.2 TEventInfo.GetParamCount Method

Pascal

```
function GetParamCount: Integer;
```

C++

```
__fastcall int GetParamCount();
```

Description

This is GetParamCount, a member of class TEventInfo.

60.1.9.3.3 TEventInfo.GetParamFlags Method

Pascal

```
function GetParamFlags(Index: Integer): TParamFlags;
```

C++

```
__fastcall TParamFlags GetParamFlags(int Index);
```

Description

This is GetParamFlags, a member of class TEventInfo.

60.1.9.3.4 TEventInfo.GetParamName Method

Pascal

```
function GetParamName(Index: Integer): string;
```

C++

```
__fastcall AnsiString GetParamName(int Index);
```

Description

This is GetParamName, a member of class TEventInfo.

60.1.9.3.5 TEventInfo.GetParamType Method

Pascal

```
function GetParamType(Index: Integer): string;
```

C++

```
__fastcall AnsiString GetParamType(int Index);
```

Description

This is GetParamType, a member of class TEventInfo.

60.1.9.3.6 TEventInfo.GetResultType Method

Pascal

```
function GetResultType: string;
```

C++

```
__fastcall AnsiString GetResultType();
```

Description

This is GetResultType, a member of class TEventInfo.

60.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IActivatable (see page 11134)	This is class DesignIntf.IActivatable.
	IClass (see page 11135)	This is class DesignIntf.IClass.
	IComponentEditor (see page 11138)	IComponentEditor is the interface that the form designer uses to communicate with component editors.
	IComponentGuidelines (see page 11143)	This is class DesignIntf.IComponentGuidelines.
	ICustomModule (see page 11145)	This is class DesignIntf.ICustomModule.
	IDefaultEditor (see page 11148)	This is class DesignIntf.IDefaultEditor.
	IDesigner (see page 11149)	IDesigner is an interface to the form designer in the IDE.
	IDesigner60 (see page 11151)	This is class DesignIntf.IDesigner60.
	IDesigner70 (see page 11167)	This is class DesignIntf.IDesigner70.
	IDesigner80 (see page 11169)	This is class DesignIntf.IDesigner80.
	IDesignerSelections (see page 11172)	IDesignerSelections represents a list of persistent objects.
	IDesignNotification (see page 11176)	This is class DesignIntf.IDesignNotification.
	IDesignObject (see page 11178)	This is class DesignIntf.IDesignObject.
	IDesignPersistent (see page 11183)	This is class DesignIntf.IDesignPersistent.
	IDesignWindow (see page 11184)	This is class DesignIntf.IDesignWindow.
	IEditHandler (see page 11186)	This is class DesignIntf.IEditHandler.
	IEventInfo (see page 11187)	This is class DesignIntf.IEventInfo.
	IHostForm (see page 11189)	This is class DesignIntf.IHostForm.
	IImplementation (see page 11196)	This is class DesignIntf.IImplementation.
	IMethodProperty (see page 11197)	This is class DesignIntf.IMethodProperty.
	IProperty (see page 11197)	IProperty is the interface that the Object Inspector uses to communicate with property editors.
	IProperty70 (see page 11205)	This is class DesignIntf.IProperty70.
	IProperty80 (see page 11207)	This is class DesignIntf.IProperty80.
	IPropertyControl (see page 11208)	This is class DesignIntf.IPropertyControl.
	IPropertyDescription (see page 11209)	This is class DesignIntf.IPropertyDescription.
	IPropertyHost (see page 11210)	This is class DesignIntf.IPropertyHost.
	IPropertyKind (see page 11211)	This is class DesignIntf.IPropertyKind.
	IReferenceProperty (see page 11212)	This is class DesignIntf.IReferenceProperty.
	ISelectionEditor (see page 11213)	This is class DesignIntf.ISelectionEditor.
	ISelectionEditorList (see page 11215)	This is class DesignIntf.ISelectionEditorList.
	ISelectionPropertyFilter (see page 11217)	This is class DesignIntf.ISelectionPropertyFilter.
	IShowReferenceProperty (see page 11218)	This is class DesignIntf.IShowReferenceProperty.
	IWideProperty (see page 11219)	This is class DesignIntf.IWideProperty.
	IWideProperty10 (see page 11221)	This is class DesignIntf.IWideProperty10.
	IWidePropertyDescription (see page 11222)	This is class DesignIntf.IWidePropertyDescription.

60.2.1 IActivatable Interface

Class Hierarchy

DesignIntf.IActivatable

Pascal

```
IActivatable = interface;
```

C++

```
__interface IActivatable;
```

File

DesignIntf

Description

This is class DesignIntf.IActivatable.

60.2.1.1 IActivatable Members

The following tables list the members exposed by IActivatable.

Public Methods

	Name	Description
	Activate (see page 11135)	This is Activate, a member of class IActivatable.

60.2.1.2 IActivatable Methods

The methods of the IActivatable class are listed here.

Public Methods

	Name	Description
	Activate (see page 11135)	This is Activate, a member of class IActivatable.

60.2.1.2.1 IActivatable.Activate Method

Pascal

```
procedure Activate;
```

C++

```
__fastcall Activate();
```

Description

This is Activate, a member of class IActivatable.

60.2.2 IClass Interface

Class Hierarchy

DesignIntf.IClass

Pascal

```
IClass = interface;
```

C++

```
__interface IClass;
```

File

DesignIntf

Description

This is class DesignIntf.IClass.

60.2.2.1 IClass Members

The following tables list the members exposed by IClass.

Public Methods

	Name	Description
	ClassNameIs (see page 11136)	This is ClassNameIs, a member of class IClass.
	GetClassName (see page 11137)	This is GetClassName, a member of class IClass.
	GetClassParent (see page 11137)	This is GetClassParent, a member of class IClass.
	GetUnitName (see page 11137)	This is GetUnitName, a member of class IClass.

Public Properties

	Name	Description
R	ClassName (see page 11137)	This is ClassName, a member of class IClass.
R	ClassParent (see page 11138)	This is ClassParent, a member of class IClass.
R	UnitName (see page 11138)	This is UnitName, a member of class IClass.

60.2.2.2 IClass Methods

The methods of the IClass class are listed here.

Public Methods

	Name	Description
	ClassNameIs (see page 11136)	This is ClassNameIs, a member of class IClass.
	GetClassName (see page 11137)	This is GetClassName, a member of class IClass.
	GetClassParent (see page 11137)	This is GetClassParent, a member of class IClass.
	GetUnitName (see page 11137)	This is GetUnitName, a member of class IClass.

60.2.2.2.1 IClass.ClassNameIs Method

Pascal

```
function ClassNameIs(const AClassName: string): Boolean;
```

C++

```
__fastcall Boolean ClassNameIs(const AnsiString AClassName);
```

Description

This is ClassNameIs, a member of class IClass.

60.2.2.2.2 IClass.GetClassName Method

Pascal

```
function GetClassName: string;
```

C++

```
__fastcall AnsiString GetClassName();
```

Description

This is GetClassName, a member of class IClass.

60.2.2.2.3 IClass.GetClassParent Method

Pascal

```
function GetClassParent: IClass;
```

C++

```
__fastcall IClass GetClassParent();
```

Description

This is GetClassParent, a member of class IClass.

60.2.2.2.4 IClass.GetUnitName Method

Pascal

```
function GetUnitName: string;
```

C++

```
__fastcall AnsiString GetUnitName();
```

Description

This is GetUnitName, a member of class IClass.

60.2.2.3 IClass Properties

The properties of the IClass class are listed here.

Public Properties

	Name	Description
R	ClassName (see page 11137)	This is ClassName, a member of class IClass.
R	ClassParent (see page 11138)	This is ClassParent, a member of class IClass.
R	UnitName (see page 11138)	This is UnitName, a member of class IClass.

60.2.2.3.1 IClass.ClassName Property

Pascal

```
property ClassName: string;
```

C++

```
__property AnsiString ClassName;
```

Description

This is ClassName, a member of class IClass.

60.2.2.3.2 IClass.ClassParent Property

Pascal

```
property ClassParent: IClass;
```

C++

```
__property IClass ClassParent;
```

Description

This is ClassParent, a member of class IClass.

60.2.2.3.3 IClass.UnitName Property

Pascal

```
property UnitName: string;
```

C++

```
__property AnsiString UnitName;
```

Description

This is UnitName, a member of class IClass.

60.2.3 IComponentEditor Interface

IComponentEditor is the interface that the form designer uses to communicate with component editors.

Class Hierarchy

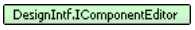

Pascal

```
IComponentEditor = interface;
```

C++

```
__interface IComponentEditor;
```

File

DesignIntf

Description

When you create your own component editors, they must meet two criteria:

They must be descendants of TBaseComponentEditor (see page 11105).

They must implement the IComponentEditor interface.

The forms designer uses the methods on the IComponentEditor interface to interact with component editors.

Typically, component editors are derived from TComponentEditor, which supplies a default implementation for

IComponentEditor, or TDefaultEditor, which also supplies a default implementation of the edit method to bring up an event handler in the edit window.

Note: C++ method declarations that use IComponentEditor use the _di_IComponentEditor type instead. This type is a DelphiInterface wrapper around the IComponentEditor interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IComponentEditor >
_di_IComponentEditor;
```

See Also

TDefaultEditor (see page 10967)

TBaseComponentEditor (see page 11105)

TComponentEditor (see page 10927)

60.2.3.1 IComponentEditor Members

The following tables list the members exposed by IComponentEditor.

Public Methods

	Name	Description
	Copy (see page 11140)	Provides an opportunity to copy the component to the clipboard in custom formats.
	Edit (see page 11140)	Responds when the user double-clicks the component in the form designer.
	ExecuteVerb (see page 11140)	This is ExecuteVerb, a member of class IComponentEditor.
	GetComponent (see page 11141)	Returns a reference to the component on which the editor operates.
	GetDesigner (see page 11141)	Returns the interface for the form designer.
	GetVerb (see page 11141)	Returns the string that corresponds to a specified position of the component's context menu.
	GetVerbCount (see page 11142)	Returns the number of menu strings that the component editor adds to the context menu.
	IsInInlined (see page 11142)	Indicates whether the component has the csInline flag set in its ComponentState property.
	PrepareItem (see page 11142)	Allows the component editor to modify menu items as they are added to the context menu.

60.2.3.2 IComponentEditor Methods

The methods of the IComponentEditor class are listed here.

Public Methods

	Name	Description
	Copy (see page 11140)	Provides an opportunity to copy the component to the clipboard in custom formats.
	Edit (see page 11140)	Responds when the user double-clicks the component in the form designer.
	ExecuteVerb (see page 11140)	This is ExecuteVerb, a member of class IComponentEditor.
	GetComponent (see page 11141)	Returns a reference to the component on which the editor operates.
	GetDesigner (see page 11141)	Returns the interface for the form designer.
	GetVerb (see page 11141)	Returns the string that corresponds to a specified position of the component's context menu.
	GetVerbCount (see page 11142)	Returns the number of menu strings that the component editor adds to the context menu.
	IsInInlined (see page 11142)	Indicates whether the component has the csInline flag set in its ComponentState property.

	PrepareItem (see page 11142)	Allows the component editor to modify menu items as they are added to the context menu.

60.2.3.2.1 IComponentEditor.Copy Method

Provides an opportunity to copy the component to the clipboard in custom formats.

Pascal

```
procedure Copy;
```

C++

```
__fastcall Copy();
```

Description

The forms designer calls Copy when the user tries to copy the component to the clipboard. When Copy is called, the component's filed image is already on the Clipboard.

Implement Copy to copy a different clipboard format, which is ignored by the designer but might be recognized by another application.

See Also

Assign (see page 3331)

SetAsHandleTClipboard_SetFormat

AsText (see page 3347)

TClipboard (see page 3328)

60.2.3.2.2 IComponentEditor.Edit Method

Responds when the user double-clicks the component in the form designer.

Pascal

```
procedure Edit;
```

C++

```
__fastcall Edit();
```

Description

The form designer calls Edit when the user double-clicks the component.

Implement the Edit method to provide a response when the user double-clicks the component. For example, the Edit method can bring up a dialog or design expert.

See Also

GetVerbCount (see page 11142)

executeverb (see page 11140)

60.2.3.2.3 IComponentEditor.ExecuteVerb Method

Pascal

```
procedure ExecuteVerb(Index: Integer);
```

C++

```
__fastcall ExecuteVerb(int Index);
```

Description

This is ExecuteVerb, a member of class IComponentEditor.

60.2.3.2.4 IComponentEditor.GetComponent Method

Returns a reference to the component on which the editor operates.

Pascal

```
function GetComponent: TComponent;
```

C++

```
__fastcall TComponent * GetComponent();
```

Description

A component editor is created for each instance of the registered component class when it is selected in the form designer. GetComponent returns the selected instance. This component is associated with a component editor through its constructor.

See Also

Create (see page 11107)

60.2.3.2.5 IComponentEditor.GetDesigner Method

Returns the interface for the form designer.

Pascal

```
function GetDesigner: IDesigner;
```

C++

```
__fastcall IDesigner GetDesigner();
```

Description

GetDesigner returns the interface of the form designer. This interface is supplied to a component editor as a parameter to the constructor.

See Also

Create (see page 11107)

60.2.3.2.6 IComponentEditor.GetVerb Method

Returns the string that corresponds to a specified position of the component's context menu.

Pascal

```
function GetVerb(Index: Integer): string;
```

C++

```
__fastcall AnsiString GetVerb(int Index);
```

Description

The form designer calls GetVerb iteratively to build the context menu that appears when the user right-clicks the component.

When implementing GetVerb, add an ampersand (&) to a verb to cause the following character to appear underlined in the

context menu and to act as a shortcut key for selecting the menu item. Be sure to add ellipsis (...) to the end of a verb if its corresponding action is to bring up a dialog.

See Also

executeverb (see page 11140)

GetVerbCount (see page 11142)

60.2.3.2.7 IComponentEditor.GetVerbCount Method

Returns the number of menu strings that the component editor adds to the context menu.

Pascal

```
function GetVerbCount: Integer;
```

C++

```
__fastcall int GetVerbCount();
```

Description

The form designer calls GetVerbCount when the user right-clicks the component in order to determine the number of menu items that should be added to the context menu. It then adds the menu items by calling GetVerb (see page 11141) for every value from 0 to GetVerbCount - 1.

See Also

GetVerb (see page 11141)

executeverb (see page 11140)

60.2.3.2.8 IComponentEditor.IsInInlined Method

Indicates whether the component has the csInline flag set in its ComponentState property.

Pascal

```
function IsInInlined: Boolean;
```

C++

```
__fastcall Boolean IsInInlined();
```

Description

IsInInlined indicates whether the component being edited is inside a top-level component that can be modified at design time and also embedded in a form.

For instance, when a component is used on a Frame and that Frame is dropped on a Form, IsInInlined returns true for the component when it is embedded in the Frame on the Form, but returns false when the component that is only on the Frame.

See Also

ComponentState (see page 2804)

60.2.3.2.9 IComponentEditor.PrepareItem Method

Allows the component editor to modify menu items as they are added to the context menu.

Pascal

```
procedure PrepareItem(Index: Integer; const AItem: IMenuItem);
```

C++

```
__fastcall PrepareItem(int Index, const IMenuItem AItem);
```

Description

The form designer calls PrepareItem for each verb when it constructs the context menu. This method lets the component editor make any modifications to the menu item for each verb, including adding subitems or removing the item from the context menu.

Index identifies the current verb that the form designer is adding to the context menu. This is the same index as appears for the command in the GetVerb (see page 11141) and ExecuteVerb (see page 11140) methods.

AItem is an interface to the menu item created by the form designer for the current verb. To add subitems, call the AddItem method of this interface. To disable the item, set the Enabled property to false. To suppress the menu item altogether, set its Visible property to false.

See Also

GetVerb (see page 11141)

GetVerbCount (see page 11142)

executeverb (see page 11140)

60.2.4 IComponentGuidelines Interface

Class Hierarchy

DesignIntf.IComponentGuidelines

Pascal

```
IComponentGuidelines = interface;
```

C++

```
__interface IComponentGuidelines;
```

File

DesignIntf

Description

This is class DesignIntf.IComponentGuidelines.

60.2.4.1 IComponentGuidelines Members

The following tables list the members exposed by IComponentGuidelines.

Public Methods

	Name	Description
	GetCount (see page 11144)	This is GetCount, a member of class IComponentGuidelines.
	GetDesignerGuideOffset (see page 11144)	This is GetDesignerGuideOffset, a member of class IComponentGuidelines.
	GetDesignerGuideType (see page 11144)	This is GetDesignerGuideType, a member of class IComponentGuidelines.

Public Properties

	Name	Description
R	Count (see page 11145)	This is Count, a member of class IComponentGuidelines.

	DesignerGuideOffsets (see page 11145)	This is DesignerGuideOffsets, a member of class IComponentGuidelines.
	DesignerGuideTypes (see page 11145)	This is DesignerGuideTypes, a member of class IComponentGuidelines.

60.2.4.2 IComponentGuidelines Methods

The methods of the IComponentGuidelines class are listed here.

Public Methods

	Name	Description
	GetCount (see page 11144)	This is GetCount, a member of class IComponentGuidelines.
	GetDesignerGuideOffset (see page 11144)	This is GetDesignerGuideOffset, a member of class IComponentGuidelines.
	GetDesignerGuideType (see page 11144)	This is GetDesignerGuideType, a member of class IComponentGuidelines.

60.2.4.2.1 IComponentGuidelines.GetCount Method

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

This is GetCount, a member of class IComponentGuidelines.

60.2.4.2.2 IComponentGuidelines.GetDesignerGuideOffset Method

Pascal

```
function GetDesignerGuideOffset(Index: Integer): Integer;
```

C++

```
__fastcall int GetDesignerGuideOffset(int Index);
```

Description

This is GetDesignerGuideOffset, a member of class IComponentGuidelines.

60.2.4.2.3 IComponentGuidelines.GetDesignerGuideType Method

Pascal

```
function GetDesignerGuideType(Index: Integer): TDesignerGuideType;
```

C++

```
__fastcall TDesignerGuideType GetDesignerGuideType(int Index);
```

Description

This is GetDesignerGuideType, a member of class IComponentGuidelines.

60.2.4.3 IComponentGuidelines Properties

The properties of the IComponentGuidelines class are listed here.

Public Properties

	Name	Description
R	Count (see page 11145)	This is Count, a member of class IComponentGuidelines.
R	DesignerGuideOffsets (see page 11145)	This is DesignerGuideOffsets, a member of class IComponentGuidelines.
R	DesignerGuideTypes (see page 11145)	This is DesignerGuideTypes, a member of class IComponentGuidelines.

60.2.4.3.1 IComponentGuidelines.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class IComponentGuidelines.

60.2.4.3.2 IComponentGuidelines.DesignerGuideOffsets Property

Pascal

```
property DesignerGuideOffsets [Index: Integer]: Integer;
```

C++

```
__property int DesignerGuideOffsets[int Index];
```

Description

This is DesignerGuideOffsets, a member of class IComponentGuidelines.

60.2.4.3.3 IComponentGuidelines.DesignerGuideTypes Property

Pascal

```
property DesignerGuideTypes [Index: Integer]: TDesignerGuideType;
```

C++

```
__property TDesignerGuideType DesignerGuideTypes[int Index];
```

Description

This is DesignerGuideTypes, a member of class IComponentGuidelines.

60.2.5 ICustomModule Interface

Class Hierarchy

DesignIntf.ICustomModule

Pascal

```
ICustomModule = interface;
```

C++

```
__interface ICustomModule;
```

File

DesignIntf

Description

This is class DesignIntf.ICustomModule.

60.2.5.1 ICustomModule Members

The following tables list the members exposed by ICustomModule.

Public Methods

	Name	Description
	ExecuteVerb (see page 11146)	This is ExecuteVerb, a member of class ICustomModule.
	GetAttributes (see page 11147)	This is GetAttributes, a member of class ICustomModule.
	GetVerb (see page 11147)	This is GetVerb, a member of class ICustomModule.
	GetVerbCount (see page 11147)	This is GetVerbCount, a member of class ICustomModule.
	Nestable (see page 11147)	This is Nestable, a member of class ICustomModule.
	PrepareItem (see page 11147)	This is PrepareItem, a member of class ICustomModule.
	Saving (see page 11148)	This is Saving, a member of class ICustomModule.
	ValidateComponent (see page 11148)	This is ValidateComponent, a member of class ICustomModule.
	ValidateComponentClass (see page 11148)	This is ValidateComponentClass, a member of class ICustomModule.

60.2.5.2 ICustomModule Methods

The methods of the ICustomModule class are listed here.

Public Methods

	Name	Description
	ExecuteVerb (see page 11146)	This is ExecuteVerb, a member of class ICustomModule.
	GetAttributes (see page 11147)	This is GetAttributes, a member of class ICustomModule.
	GetVerb (see page 11147)	This is GetVerb, a member of class ICustomModule.
	GetVerbCount (see page 11147)	This is GetVerbCount, a member of class ICustomModule.
	Nestable (see page 11147)	This is Nestable, a member of class ICustomModule.
	PrepareItem (see page 11147)	This is PrepareItem, a member of class ICustomModule.
	Saving (see page 11148)	This is Saving, a member of class ICustomModule.
	ValidateComponent (see page 11148)	This is ValidateComponent, a member of class ICustomModule.
	ValidateComponentClass (see page 11148)	This is ValidateComponentClass, a member of class ICustomModule.

60.2.5.2.1 ICustomModule.ExecuteVerb Method

Pascal

```
procedure ExecuteVerb(Index: Integer);
```

C++

```
__fastcall ExecuteVerb(int Index);
```

Description

This is ExecuteVerb, a member of class ICustomModule.

60.2.5.2.2 ICustomModule.GetAttributes Method

Pascal

```
function GetAttributes: TCustomModuleAttributes;
```

C++

```
__fastcall TCustomModuleAttributes GetAttributes();
```

Description

This is GetAttributes, a member of class ICustomModule.

60.2.5.2.3 ICustomModule.GetVerb Method

Pascal

```
function GetVerb(Index: Integer): string;
```

C++

```
__fastcall AnsiString GetVerb(int Index);
```

Description

This is GetVerb, a member of class ICustomModule.

60.2.5.2.4 ICustomModule.GetVerbCount Method

Pascal

```
function GetVerbCount: Integer;
```

C++

```
__fastcall int GetVerbCount();
```

Description

This is GetVerbCount, a member of class ICustomModule.

60.2.5.2.5 ICustomModule.Nestable Method

Pascal

```
function Nestable: Boolean;
```

C++

```
__fastcall Boolean Nestable();
```

Description

This is Nestable, a member of class ICustomModule.

60.2.5.2.6 ICustomModule.PrepareItem Method

Pascal

```
procedure PrepareItem(Index: Integer; const AItem: IMenuItem);
```

C++

```
__fastcall PrepareItem(int Index, const IMenuItem AItem);
```

Description

This is PrepareItem, a member of class ICustomModule.

60.2.5.2.7 ICustomModule.Saving Method

Pascal

```
procedure Saving;
```

C++

```
__fastcall Saving();
```

Description

This is Saving, a member of class ICustomModule.

60.2.5.2.8 ICustomModule.ValidateComponent Method

Pascal

```
procedure ValidateComponent(Component: TComponent);
```

C++

```
__fastcall ValidateComponent(TComponent * Component);
```

Description

This is ValidateComponent, a member of class ICustomModule.

60.2.5.2.9 ICustomModule.ValidateComponentClass Method

Pascal

```
function ValidateComponentClass(ComponentClass: TComponentClass): Boolean;
```

C++

```
__fastcall Boolean ValidateComponentClass(TComponentClass ComponentClass);
```

Description

This is ValidateComponentClass, a member of class ICustomModule.

60.2.6 IDefaultEditor Interface

Class Hierarchy

Pascal

```
IDefaultEditor = interface(IComponentEditor);
```

C++

```
__interface IDefaultEditor : public IComponentEditor;
```

File

DesignIntf

Description

This is class DesignIntf.IDefaultEditor.

60.2.7 IDesigner Interface

IDesigner is an interface to the form designer in the IDE.

Class Hierarchy

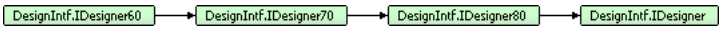

Pascal

```
IDesigner = interface(IDesigner80);
```

C++

```
__interface IDesigner : public IDesigner80;
```

File

DesignIntf

Description

Use IDesigner to interact with the current unit when implementing a property editor or component editor. Access IDesigner through the Designer property of the property editor or component editor. IDesigner introduces several methods that are useful for managing standard menu items and for creating and manipulating components and methods such as event handlers.

Note: C++ method declarations that use IDesigner use the _di_IDesigner type instead. This is a DelphiInterface wrapper around the IDesigner interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IDesigner > _di_IDesigner;
```

See Also

TComponentEditor (see page 10927)

TPropertyEditor (see page 11019)

60.2.7.1 IDesigner Members

The following tables list the members exposed by IDesigner.

Public Methods

	Name	Description
	Activate (see page 11154)	This is Activate, a member of class IDesigner60.
	CanPaste (see page 11154)	This is CanPaste, a member of class IDesigner60.
	ChainCall (see page 11154)	This is the overview for the ChainCall method overload.
	ClearSelection (see page 11155)	This is ClearSelection, a member of class IDesigner60.
	CopySelection (see page 11155)	This is CopySelection, a member of class IDesigner60.
	CreateComponent (see page 11156)	This is CreateComponent, a member of class IDesigner60.
	CreateCurrentComponent (see page 11156)	This is CreateCurrentComponent, a member of class IDesigner60.
	CreateMethod (see page 11156)	This is CreateMethod, a member of class IDesigner60.
	CreateMethod (see page 11171)	This is CreateMethod, a member of class IDesigner80.

	CutSelection (see page 11156)	This is CutSelection, a member of class IDesigner60.
	DeleteSelection (see page 11157)	This is DeleteSelection, a member of class IDesigner60.
	Edit (see page 11157)	This is Edit, a member of class IDesigner60.
	FindRootAncestor (see page 11168)	This is FindRootAncestor, a member of class IDesigner70.
	GetActiveClassGroup (see page 11169)	This is GetActiveClassGroup, a member of class IDesigner70.
	GetAncestorDesigner (see page 11157)	This is GetAncestorDesigner, a member of class IDesigner60.
	GetBaseRegKey (see page 11157)	This is GetBaseRegKey, a member of class IDesigner60.
	GetComponent (see page 11157)	This is GetComponent, a member of class IDesigner60.
	GetComponentName (see page 11158)	This is GetComponentName, a member of class IDesigner60.
	GetComponentNames (see page 11158)	This is GetComponentNames, a member of class IDesigner60.
	GetDesignerExtension (see page 11151)	This is GetDesignerExtension, a member of class IDesigner.
	GetIDEOptions (see page 11158)	This is GetIDEOptions, a member of class IDesigner60.
	GetIsDormant (see page 11158)	This is GetIsDormant, a member of class IDesigner60.
	GetMethodName (see page 11159)	This is GetMethodName, a member of class IDesigner60.
	GetMethods (see page 11159)	This is GetMethods, a member of class IDesigner60.
	GetMethods (see page 11171)	This is GetMethods, a member of class IDesigner80.
	GetObject (see page 11159)	This is GetObject, a member of class IDesigner60.
	GetObjectName (see page 11159)	This is GetObjectName, a member of class IDesigner60.
	GetObjectNames (see page 11159)	This is GetObjectNames, a member of class IDesigner60.
	GetPathAndBaseExeName (see page 11160)	This is GetPathAndBaseExeName, a member of class IDesigner60.
	GetPrivateDirectory (see page 11160)	This is GetPrivateDirectory, a member of class IDesigner60.
	GetProjectModules (see page 11160)	This is GetProjectModules, a member of class IDesigner60.
	GetRoot (see page 11160)	This is GetRoot, a member of class IDesigner60.
	GetRootClassName (see page 11161)	This is GetRootClassName, a member of class IDesigner60.
	GetScrollRanges (see page 11161)	This is GetScrollRanges, a member of class IDesigner60.
	GetSelections (see page 11161)	This is GetSelections, a member of class IDesigner60.
	GetShiftState (see page 11161)	This is GetShiftState, a member of class IDesigner60.
	IsComponentHidden (see page 11161)	This is IsComponentHidden, a member of class IDesigner60.
	IsComponentLinkable (see page 11162)	This is IsComponentLinkable, a member of class IDesigner60.
	IsSourceReadOnly (see page 11162)	This is IsSourceReadOnly, a member of class IDesigner60.
	MakeComponentLinkable (see page 11162)	This is MakeComponentLinkable, a member of class IDesigner60.
	MethodExists (see page 11162)	This is MethodExists, a member of class IDesigner60.
	MethodFromAncestor (see page 11163)	This is MethodFromAncestor, a member of class IDesigner60.
	ModalEdit (see page 11163)	This is ModalEdit, a member of class IDesigner60.
	Modified (see page 11163)	This is Modified, a member of class IDesigner60.
	ModuleFileNames (see page 11163)	This is ModuleFileNames, a member of class IDesigner60.
	NoSelection (see page 11163)	This is NoSelection, a member of class IDesigner60.
	PasteSelection (see page 11164)	This is PasteSelection, a member of class IDesigner60.
	RenameMethod (see page 11164)	This is RenameMethod, a member of class IDesigner60.
	Resurrect (see page 11164)	This is Resurrect, a member of class IDesigner60.
	Revert (see page 11164)	This is Revert, a member of class IDesigner60.
	SelectComponent (see page 11165)	This is SelectComponent, a member of class IDesigner60.
	SelectComponent (see page 11171)	This is SelectComponent, a member of class IDesigner80.
	SelectItemName (see page 11165)	This is SelectItemName, a member of class IDesigner60.
	SetSelections (see page 11165)	This is SetSelections, a member of class IDesigner60.
	ShowMethod (see page 11165)	This is ShowMethod, a member of class IDesigner60.
	UniqueName (see page 11165)	This is UniqueName, a member of class IDesigner60.

Public Properties

	Name	Description
R	ActiveClassGroup (see page 11169)	This is ActiveClassGroup, a member of class IDesigner70.
R	AncestorDesigner (see page 11166)	This is AncestorDesigner, a member of class IDesigner60.
R	DesignerExtention (see page 11151)	This is DesignerExtention, a member of class IDesigner.
R	IsDormant (see page 11166)	This is IsDormant, a member of class IDesigner60.

	Root (see page 11166)	This is Root, a member of class IDesigner60.

60.2.7.2 IDesigner Methods

The methods of the IDesigner class are listed here.

Public Methods

	Name	Description
	GetDesignerExtension (see page 11151)	This is GetDesignerExtension, a member of class IDesigner.

60.2.7.2.1 IDesigner.GetDesignerExtension Method

Pascal

```
function GetDesignerExtension: string;
```

C++

```
__fastcall AnsiString GetDesignerExtension();
```

Description

This is GetDesignerExtension, a member of class IDesigner.

60.2.7.3 IDesigner Properties

The properties of the IDesigner class are listed here.

Public Properties

	Name	Description
	DesignerExtention (see page 11151)	This is DesignerExtention, a member of class IDesigner.

60.2.7.3.1 IDesigner.DesignerExtention Property

Pascal

```
property DesignerExtention: string;
```

C++

```
__property AnsiString DesignerExtention;
```

Description

This is DesignerExtention, a member of class IDesigner.

60.2.8 IDesigner60 Interface

Class Hierarchy

DesignIntf.IDesigner60

Pascal

```
IDesigner60 = interface;
```

C++

```
__interface IDesigner60;
```

File

DesignIntf

Description

This is class DesignIntf.IDesigner60.

60.2.8.1 IDesigner60 Members

The following tables list the members exposed by IDesigner60.

Public Methods

	Name	Description
	Activate (see page 11154)	This is Activate, a member of class IDesigner60.
	CanPaste (see page 11154)	This is CanPaste, a member of class IDesigner60.
	ChainCall (see page 11154)	This is the overview for the ChainCall method overload.
	ClearSelection (see page 11155)	This is ClearSelection, a member of class IDesigner60.
	CopySelection (see page 11155)	This is CopySelection, a member of class IDesigner60.
	CreateComponent (see page 11156)	This is CreateComponent, a member of class IDesigner60.
	CreateCurrentComponent (see page 11156)	This is CreateCurrentComponent, a member of class IDesigner60.
	CreateMethod (see page 11156)	This is CreateMethod, a member of class IDesigner60.
	CutSelection (see page 11156)	This is CutSelection, a member of class IDesigner60.
	DeleteSelection (see page 11157)	This is DeleteSelection, a member of class IDesigner60.
	Edit (see page 11157)	This is Edit, a member of class IDesigner60.
	GetAncestorDesigner (see page 11157)	This is GetAncestorDesigner, a member of class IDesigner60.
	GetBaseRegKey (see page 11157)	This is GetBaseRegKey, a member of class IDesigner60.
	GetComponent (see page 11157)	This is GetComponent, a member of class IDesigner60.
	GetComponentName (see page 11158)	This is GetComponentName, a member of class IDesigner60.
	GetComponentNames (see page 11158)	This is GetComponentNames, a member of class IDesigner60.
	GetIDEOptions (see page 11158)	This is GetIDEOptions, a member of class IDesigner60.
	GetIsDormant (see page 11158)	This is GetIsDormant, a member of class IDesigner60.
	GetMethodName (see page 11159)	This is GetMethodName, a member of class IDesigner60.
	GetMethods (see page 11159)	This is GetMethods, a member of class IDesigner60.
	GetObject (see page 11159)	This is GetObject, a member of class IDesigner60.
	GetObjectName (see page 11159)	This is GetObjectName, a member of class IDesigner60.
	GetObjectNames (see page 11159)	This is GetObjectNames, a member of class IDesigner60.
	GetPathAndBaseExeName (see page 11160)	This is GetPathAndBaseExeName, a member of class IDesigner60.
	GetPrivateDirectory (see page 11160)	This is GetPrivateDirectory, a member of class IDesigner60.
	GetProjectModules (see page 11160)	This is GetProjectModules, a member of class IDesigner60.
	GetRoot (see page 11160)	This is GetRoot, a member of class IDesigner60.
	GetRootClassName (see page 11161)	This is GetRootClassName, a member of class IDesigner60.
	GetScrollRanges (see page 11161)	This is GetScrollRanges, a member of class IDesigner60.
	GetSelections (see page 11161)	This is GetSelections, a member of class IDesigner60.
	GetShiftState (see page 11161)	This is GetShiftState, a member of class IDesigner60.
	IsComponentHidden (see page 11161)	This is IsComponentHidden, a member of class IDesigner60.
	IsComponentLinkable (see page 11162)	This is IsComponentLinkable, a member of class IDesigner60.
	IsSourceReadOnly (see page 11162)	This is IsSourceReadOnly, a member of class IDesigner60.
	MakeComponentLinkable (see page 11162)	This is MakeComponentLinkable, a member of class IDesigner60.
	MethodExists (see page 11162)	This is MethodExists, a member of class IDesigner60.
	MethodFromAncestor (see page 11163)	This is MethodFromAncestor, a member of class IDesigner60.
	ModalEdit (see page 11163)	This is ModalEdit, a member of class IDesigner60.

	Modified (see page 11163)	This is Modified, a member of class IDesigner60.
	ModuleFileNames (see page 11163)	This is ModuleFileNames, a member of class IDesigner60.
	NoSelection (see page 11163)	This is NoSelection, a member of class IDesigner60.
	PasteSelection (see page 11164)	This is PasteSelection, a member of class IDesigner60.
	RenameMethod (see page 11164)	This is RenameMethod, a member of class IDesigner60.
	Resurrect (see page 11164)	This is Resurrect, a member of class IDesigner60.
	Revert (see page 11164)	This is Revert, a member of class IDesigner60.
	SelectComponent (see page 11165)	This is SelectComponent, a member of class IDesigner60.
	SelectItemName (see page 11165)	This is SelectItemName, a member of class IDesigner60.
	SetSelections (see page 11165)	This is SetSelections, a member of class IDesigner60.
	ShowMethod (see page 11165)	This is ShowMethod, a member of class IDesigner60.
	UniqueName (see page 11165)	This is UniqueName, a member of class IDesigner60.

Public Properties

	Name	Description
R	AncestorDesigner (see page 11166)	This is AncestorDesigner, a member of class IDesigner60.
R	IsDormant (see page 11166)	This is IsDormant, a member of class IDesigner60.
R	Root (see page 11166)	This is Root, a member of class IDesigner60.

60.2.8.2 IDesigner60 Methods

The methods of the IDesigner60 class are listed here.

Public Methods

	Name	Description
	Activate (see page 11154)	This is Activate, a member of class IDesigner60.
	CanPaste (see page 11154)	This is CanPaste, a member of class IDesigner60.
	ChainCall (see page 11154)	This is the overview for the ChainCall method overload.
	ClearSelection (see page 11155)	This is ClearSelection, a member of class IDesigner60.
	CopySelection (see page 11155)	This is CopySelection, a member of class IDesigner60.
	CreateComponent (see page 11156)	This is CreateComponent, a member of class IDesigner60.
	CreateCurrentComponent (see page 11156)	This is CreateCurrentComponent, a member of class IDesigner60.
	CreateMethod (see page 11156)	This is CreateMethod, a member of class IDesigner60.
	CutSelection (see page 11156)	This is CutSelection, a member of class IDesigner60.
	DeleteSelection (see page 11157)	This is DeleteSelection, a member of class IDesigner60.
	Edit (see page 11157)	This is Edit, a member of class IDesigner60.
	GetAncestorDesigner (see page 11157)	This is GetAncestorDesigner, a member of class IDesigner60.
	GetBaseRegKey (see page 11157)	This is GetBaseRegKey, a member of class IDesigner60.
	GetComponent (see page 11157)	This is GetComponent, a member of class IDesigner60.
	GetComponentName (see page 11158)	This is GetComponentName, a member of class IDesigner60.
	GetComponentNames (see page 11158)	This is GetComponentNames, a member of class IDesigner60.
	GetIDEOptions (see page 11158)	This is GetIDEOptions, a member of class IDesigner60.
	GetIsDormant (see page 11158)	This is GetIsDormant, a member of class IDesigner60.
	GetMethodName (see page 11159)	This is GetMethodName, a member of class IDesigner60.
	GetMethods (see page 11159)	This is GetMethods, a member of class IDesigner60.
	GetObject (see page 11159)	This is GetObject, a member of class IDesigner60.
	GetObjectName (see page 11159)	This is GetObjectName, a member of class IDesigner60.
	GetObjectNames (see page 11159)	This is GetObjectNames, a member of class IDesigner60.
	GetPathAndBaseExeName (see page 11160)	This is GetPathAndBaseExeName, a member of class IDesigner60.
	GetPrivateDirectory (see page 11160)	This is GetPrivateDirectory, a member of class IDesigner60.
	GetProjectModules (see page 11160)	This is GetProjectModules, a member of class IDesigner60.
	GetRoot (see page 11160)	This is GetRoot, a member of class IDesigner60.
	GetRootClassName (see page 11161)	This is GetRootClassName, a member of class IDesigner60.

	GetScrollRanges (see page 11161)	This is GetScrollRanges, a member of class IDesigner60.
	GetSelections (see page 11161)	This is GetSelections, a member of class IDesigner60.
	GetShiftState (see page 11161)	This is GetShiftState, a member of class IDesigner60.
	IsComponentHidden (see page 11161)	This is IsComponentHidden, a member of class IDesigner60.
	IsComponentLinkable (see page 11162)	This is IsComponentLinkable, a member of class IDesigner60.
	IsSourceReadOnly (see page 11162)	This is IsSourceReadOnly, a member of class IDesigner60.
	MakeComponentLinkable (see page 11162)	This is MakeComponentLinkable, a member of class IDesigner60.
	MethodExists (see page 11162)	This is MethodExists, a member of class IDesigner60.
	MethodFromAncestor (see page 11163)	This is MethodFromAncestor, a member of class IDesigner60.
	ModalEdit (see page 11163)	This is ModalEdit, a member of class IDesigner60.
	Modified (see page 11163)	This is Modified, a member of class IDesigner60.
	ModuleFileNames (see page 11163)	This is ModuleFileNames, a member of class IDesigner60.
	NoSelection (see page 11163)	This is NoSelection, a member of class IDesigner60.
	PasteSelection (see page 11164)	This is PasteSelection, a member of class IDesigner60.
	RenameMethod (see page 11164)	This is RenameMethod, a member of class IDesigner60.
	Resurrect (see page 11164)	This is Resurrect, a member of class IDesigner60.
	Revert (see page 11164)	This is Revert, a member of class IDesigner60.
	SelectComponent (see page 11165)	This is SelectComponent, a member of class IDesigner60.
	SelectItemName (see page 11165)	This is SelectItemName, a member of class IDesigner60.
	SetSelections (see page 11165)	This is SetSelections, a member of class IDesigner60.
	ShowMethod (see page 11165)	This is ShowMethod, a member of class IDesigner60.
	UniqueName (see page 11165)	This is UniqueName, a member of class IDesigner60.

60.2.8.2.1 IDesigner60.Activate Method

Pascal

```
procedure Activate;
```

C++

```
__fastcall Activate();
```

Description

This is Activate, a member of class IDesigner60.

60.2.8.2.2 IDesigner60.CanPaste Method

Pascal

```
function CanPaste: Boolean;
```

C++

```
__fastcall Boolean CanPaste();
```

Description

This is CanPaste, a member of class IDesigner60.

60.2.8.2.3 ChainCall Method

This is the overview for the ChainCall method overload.

Overload List

	Name	Description
	IDesigner60.ChainCall (string, string, string, IEventInfo) (see page 11155)	This is ChainCall, a member of class IDesigner60.
	IDesigner60.ChainCall (string, string, string, PTypeData) (see page 11155)	This is ChainCall, a member of class IDesigner60.

60.2.8.2.3.1 IDesigner60.ChainCall Method (string, string, string, IEventInfo)

Pascal

```
procedure ChainCall(const MethodName: string; const InstanceName: string; const
InstanceMethod: string; const AEventInfo: IEventInfo); overload;
```

C++

```
__fastcall ChainCall(const AnsiString MethodName, const AnsiString InstanceName, const
AnsiString InstanceMethod, const IEventInfo AEventInfo);
```

Description

This is ChainCall, a member of class IDesigner60.

60.2.8.2.3.2 IDesigner60.ChainCall Method (string, string, string, PTypeData)

Pascal

```
procedure ChainCall(const MethodName: string; const InstanceName: string; const
InstanceMethod: string; TypeData: PTypeData); overload;
```

C++

```
__fastcall ChainCall(const AnsiString MethodName, const AnsiString InstanceName, const
AnsiString InstanceMethod, PTypeData TypeData);
```

Description

This is ChainCall, a member of class IDesigner60.

60.2.8.2.4 IDesigner60.ClearSelection Method

Pascal

```
procedure ClearSelection;
```

C++

```
__fastcall ClearSelection();
```

Description

This is ClearSelection, a member of class IDesigner60.

60.2.8.2.5 IDesigner60.CopySelection Method

Pascal

```
procedure CopySelection;
```

C++

```
__fastcall CopySelection();
```

Description

This is CopySelection, a member of class IDesigner60.

60.2.8.2.6 IDesigner60.CreateComponent Method

Pascal

```
function CreateComponent(ComponentClass: TComponentClass; Parent: TComponent; Left: Integer;
Top: Integer; Width: Integer; Height: Integer): TComponent;
```

C++

```
__fastcall TComponent * CreateComponent(TComponentClass ComponentClass, TComponent * Parent,
int Left, int Top, int Width, int Height);
```

Description

This is CreateComponent, a member of class IDesigner60.

60.2.8.2.7 IDesigner60.CreateCurrentComponent Method

Pascal

```
function CreateCurrentComponent(Parent: TComponent; const Rect: TRect): TComponent;
```

C++

```
__fastcall TComponent * CreateCurrentComponent(TComponent * Parent, const TRect Rect);
```

Description

This is CreateCurrentComponent, a member of class IDesigner60.

60.2.8.2.8 IDesigner60.CreateMethod Method

Pascal

```
function CreateMethod(const Name: string; TypeData: PTypeData): TMethod; overload;
```

C++

```
__fastcall TMethod CreateMethod(const AnsiString Name, PTypeData TypeData);
```

Description

This is CreateMethod, a member of class IDesigner60.

60.2.8.2.9 IDesigner60.CutSelection Method

Pascal

```
procedure CutSelection;
```

C++

```
__fastcall CutSelection();
```

Description

This is CutSelection, a member of class IDesigner60.

60.2.8.2.10 IDesigner60.DeleteSelection Method

Pascal

```
procedure DeleteSelection(ADoAll: Boolean = False);
```

C++

```
__fastcall DeleteSelection(Boolean ADoAll = False);
```

Description

This is DeleteSelection, a member of class IDesigner60.

60.2.8.2.11 IDesigner60.Edit Method

Pascal

```
procedure Edit(const Component: TComponent);
```

C++

```
__fastcall Edit(const TComponent * Component);
```

Description

This is Edit, a member of class IDesigner60.

60.2.8.2.12 IDesigner60.GetAncestorDesigner Method

Pascal

```
function GetAncestorDesigner: IDesigner;
```

C++

```
__fastcall IDesigner GetAncestorDesigner();
```

Description

This is GetAncestorDesigner, a member of class IDesigner60.

60.2.8.2.13 IDesigner60.GetBaseRegKey Method

Pascal

```
function GetBaseRegKey: string;
```

C++

```
__fastcall AnsiString GetBaseRegKey();
```

Description

This is GetBaseRegKey, a member of class IDesigner60.

60.2.8.2.14 IDesigner60.GetComponent Method

Pascal

```
function GetComponent(const Name: string): TComponent;
```

C++

```
__fastcall TComponent * GetComponent(const AnsiString Name);
```

Description

This is GetComponent, a member of class IDesigner60.

60.2.8.2.15 IDesigner60.GetComponentName Method

Pascal

```
function GetComponentName(Component: TComponent): string;
```

C++

```
__fastcall AnsiString GetComponentName(TComponent * Component);
```

Description

This is GetComponentName, a member of class IDesigner60.

60.2.8.2.16 IDesigner60.GetComponentNames Method

Pascal

```
procedure GetComponentNames(TypeData: PTypeData; Proc: TGetStrProc);
```

C++

```
__fastcall GetComponentNames(PTypeData TypeData, TGetStrProc Proc);
```

Description

This is GetComponentNames, a member of class IDesigner60.

60.2.8.2.17 IDesigner60.GetIDEOptions Method

Pascal

```
function GetIDEOptions: TCustomIniFile;
```

C++

```
__fastcall TCustomIniFile GetIDEOptions();
```

Description

This is GetIDEOptions, a member of class IDesigner60.

60.2.8.2.18 IDesigner60.GetIsDormant Method

Pascal

```
function GetIsDormant: Boolean;
```

C++

```
__fastcall Boolean GetIsDormant();
```

Description

This is GetIsDormant, a member of class IDesigner60.

60.2.8.2.19 IDesigner60.GetMethodName Method

Pascal

```
function GetMethodName(const Method: TMethod): string;
```

C++

```
__fastcall AnsiString GetMethodName(const TMethod Method);
```

Description

This is GetMethodName, a member of class IDesigner60.

60.2.8.2.20 IDesigner60.GetMethods Method

Pascal

```
procedure GetMethods(TypeData: PTypeData; Proc: TGetStrProc); overload;
```

C++

```
__fastcall GetMethods(PTypeData TypeData, TGetStrProc Proc);
```

Description

This is GetMethods, a member of class IDesigner60.

60.2.8.2.21 IDesigner60.GetObject Method

Pascal

```
function GetObject(const Name: string): TPersistent;
```

C++

```
__fastcall TPersistent * GetObject(const AnsiString Name);
```

Description

This is GetObject, a member of class IDesigner60.

60.2.8.2.22 IDesigner60.GetObjectName Method

Pascal

```
function GetObjectName(Instance: TPersistent): string;
```

C++

```
__fastcall AnsiString GetObjectName(TPersistent * Instance);
```

Description

This is GetObjectName, a member of class IDesigner60.

60.2.8.2.23 IDesigner60.GetObjectNames Method

Pascal

```
procedure GetObjectNames(TypeData: PTypeData; Proc: TGetStrProc);
```

C++

```
__fastcall GetObjectNames(PTypeData TypeData, TGetStrProc Proc);
```

Description

This is GetObjectNames, a member of class IDesigner60.

60.2.8.2.24 IDesigner60.GetPathAndBaseExeName Method

Pascal

```
function GetPathAndBaseExeName: string;
```

C++

```
__fastcall AnsiString GetPathAndBaseExeName();
```

Description

This is GetPathAndBaseExeName, a member of class IDesigner60.

60.2.8.2.25 IDesigner60.GetPrivateDirectory Method

Pascal

```
function GetPrivateDirectory: string;
```

C++

```
__fastcall AnsiString GetPrivateDirectory();
```

Description

This is GetPrivateDirectory, a member of class IDesigner60.

60.2.8.2.26 IDesigner60.GetProjectModules Method

Pascal

```
procedure GetProjectModules(Proc: TGetModuleProc);
```

C++

```
__fastcall GetProjectModules(TGetModuleProc Proc);
```

Description

This is GetProjectModules, a member of class IDesigner60.

60.2.8.2.27 IDesigner60.GetRoot Method

Pascal

```
function GetRoot: TComponent;
```

C++

```
__fastcall TComponent * GetRoot();
```

Description

This is GetRoot, a member of class IDesigner60.

60.2.8.2.28 IDesigner60.GetRootClassName Method

Pascal

```
function GetRootClassName: string;
```

C++

```
__fastcall AnsiString GetRootClassName();
```

Description

This is GetRootClassName, a member of class IDesigner60.

60.2.8.2.29 IDesigner60.GetScrollRanges Method

Pascal

```
function GetScrollRanges(const ScrollPosition: TPoint): TPoint;
```

C++

```
__fastcall TPoint GetScrollRanges(const TPoint ScrollPosition);
```

Description

This is GetScrollRanges, a member of class IDesigner60.

60.2.8.2.30 IDesigner60.GetSelections Method

Pascal

```
procedure GetSelections(const List: IDesignerSelections);
```

C++

```
__fastcall GetSelections(const IDesignerSelections List);
```

Description

This is GetSelections, a member of class IDesigner60.

60.2.8.2.31 IDesigner60.GetShiftState Method

Pascal

```
function GetShiftState: TShiftState;
```

C++

```
__fastcall TShiftState GetShiftState();
```

Description

This is GetShiftState, a member of class IDesigner60.

60.2.8.2.32 IDesigner60.IsComponentHidden Method

Pascal

```
function IsComponentHidden(Component: TComponent): Boolean;
```

C++

```
__fastcall Boolean IsComponentHidden(TComponent * Component);
```

Description

This is IsComponentHidden, a member of class IDesigner60.

60.2.8.2.33 IDesigner60.IsComponentLinkable Method

Pascal

```
function IsComponentLinkable(Component: TComponent): Boolean;
```

C++

```
__fastcall Boolean IsComponentLinkable(TComponent * Component);
```

Description

This is IsComponentLinkable, a member of class IDesigner60.

60.2.8.2.34 IDesigner60.IsSourceReadOnly Method

Pascal

```
function IsSourceReadOnly: Boolean;
```

C++

```
__fastcall Boolean IsSourceReadOnly();
```

Description

This is IsSourceReadOnly, a member of class IDesigner60.

60.2.8.2.35 IDesigner60.MakeComponentLinkable Method

Pascal

```
procedure MakeComponentLinkable(Component: TComponent);
```

C++

```
__fastcall MakeComponentLinkable(TComponent * Component);
```

Description

This is MakeComponentLinkable, a member of class IDesigner60.

60.2.8.2.36 IDesigner60.MethodExists Method

Pascal

```
function MethodExists(const Name: string): Boolean;
```

C++

```
__fastcall Boolean MethodExists(const AnsiString Name);
```

Description

This is MethodExists, a member of class IDesigner60.

60.2.8.2.37 IDesigner60.MethodFromAncestor Method

Pascal

```
function MethodFromAncestor(const Method: TMethod): Boolean;
```

C++

```
__fastcall Boolean MethodFromAncestor(const TMethod Method);
```

Description

This is MethodFromAncestor, a member of class IDesigner60.

60.2.8.2.38 IDesigner60.ModalEdit Method

Pascal

```
procedure ModalEdit(EditKey: Char; const ReturnWindow: IActivatable);
```

C++

```
__fastcall ModalEdit(Char EditKey, const IActivatable ReturnWindow);
```

Description

This is ModalEdit, a member of class IDesigner60.

60.2.8.2.39 IDesigner60.Modified Method

Pascal

```
procedure Modified;
```

C++

```
__fastcall Modified();
```

Description

This is Modified, a member of class IDesigner60.

60.2.8.2.40 IDesigner60.ModuleFileNames Method

Pascal

```
procedure ModuleFileNames(var ImplFileName: string; var IntfFileName: string; var
FormFileName: string);
```

C++

```
__fastcall ModuleFileNames(AnsiString ImplFileName, AnsiString IntfFileName, AnsiString
FormFileName);
```

Description

This is ModuleFileNames, a member of class IDesigner60.

60.2.8.2.41 IDesigner60.NoSelection Method

Pascal

```
procedure NoSelection;
```

C++

```
__fastcall NoSelection();
```

Description

This is NoSelection, a member of class IDesigner60.

60.2.8.2.42 IDesigner60.PasteSelection Method

Pascal

```
procedure PasteSelection;
```

C++

```
__fastcall PasteSelection();
```

Description

This is PasteSelection, a member of class IDesigner60.

60.2.8.2.43 IDesigner60.RenameMethod Method

Pascal

```
procedure RenameMethod(const CurName: string; const NewName: string);
```

C++

```
__fastcall RenameMethod(const AnsiString CurName, const AnsiString NewName);
```

Description

This is RenameMethod, a member of class IDesigner60.

60.2.8.2.44 IDesigner60.Resurrect Method

Pascal

```
procedure Resurrect;
```

C++

```
__fastcall Resurrect();
```

Description

This is Resurrect, a member of class IDesigner60.

60.2.8.2.45 IDesigner60.Revert Method

Pascal

```
procedure Revert(Instance: TPersistent; PropInfo: PPropInfo);
```

C++

```
__fastcall Revert(TPersistent * Instance, PPropInfo PropInfo);
```

Description

This is Revert, a member of class IDesigner60.

60.2.8.2.46 IDesigner60.SelectComponent Method

Pascal

```
procedure SelectComponent(Instance: TPersistent); overload;
```

C++

```
__fastcall SelectComponent(TPersistent * Instance);
```

Description

This is SelectComponent, a member of class IDesigner60.

60.2.8.2.47 IDesigner60.SelectItemName Method

Pascal

```
procedure SelectItemName(const PropertyName: string);
```

C++

```
__fastcall SelectItemName(const AnsiString PropertyName);
```

Description

This is SelectItemName, a member of class IDesigner60.

60.2.8.2.48 IDesigner60.SetSelections Method

Pascal

```
procedure SetSelections(const List: IDesignerSelections);
```

C++

```
__fastcall SetSelections(const IDesignerSelections List);
```

Description

This is SetSelections, a member of class IDesigner60.

60.2.8.2.49 IDesigner60.ShowMethod Method

Pascal

```
procedure ShowMethod(const Name: string);
```

C++

```
__fastcall ShowMethod(const AnsiString Name);
```

Description

This is ShowMethod, a member of class IDesigner60.

60.2.8.2.50 IDesigner60.UniqueName Method

Pascal

```
function UniqueName(const BaseName: string): string;
```

C++

```
__fastcall AnsiString UniqueName(const AnsiString BaseName);
```

Description

This is UniqueName, a member of class IDesigner60.

60.2.8.3 IDesigner60 Properties

The properties of the IDesigner60 class are listed here.

Public Properties

	Name	Description
R	AncestorDesigner (see page 11166)	This is AncestorDesigner, a member of class IDesigner60.
R	IsDormant (see page 11166)	This is IsDormant, a member of class IDesigner60.
R	Root (see page 11166)	This is Root, a member of class IDesigner60.

60.2.8.3.1 IDesigner60.AncestorDesigner Property

Pascal

```
property AncestorDesigner: IDesigner;
```

C++

```
__property IDesigner AncestorDesigner;
```

Description

This is AncestorDesigner, a member of class IDesigner60.

60.2.8.3.2 IDesigner60.IsDormant Property

Pascal

```
property IsDormant: Boolean;
```

C++

```
__property Boolean IsDormant;
```

Description

This is IsDormant, a member of class IDesigner60.

60.2.8.3.3 IDesigner60.Root Property

Pascal

```
property Root: TComponent;
```

C++

```
__property TComponent * Root;
```

Description

This is Root, a member of class IDesigner60.

60.2.9 IDesigner70 Interface

Class Hierarchy

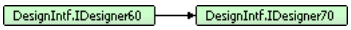

Pascal

```
IDesigner70 = interface(IDesigner60);
```

C++

```
__interface IDesigner70 : public IDesigner60;
```

File

DesignIntf

Description

This is class DesignIntf.IDesigner70.

60.2.9.1 IDesigner70 Members

The following tables list the members exposed by IDesigner70.

Public Methods

	Name	Description
	Activate (see page 11154)	This is Activate, a member of class IDesigner60.
	CanPaste (see page 11154)	This is CanPaste, a member of class IDesigner60.
	ChainCall (see page 11154)	This is the overview for the ChainCall method overload.
	ClearSelection (see page 11155)	This is ClearSelection, a member of class IDesigner60.
	CopySelection (see page 11155)	This is CopySelection, a member of class IDesigner60.
	CreateComponent (see page 11156)	This is CreateComponent, a member of class IDesigner60.
	CreateCurrentComponent (see page 11156)	This is CreateCurrentComponent, a member of class IDesigner60.
	CreateMethod (see page 11156)	This is CreateMethod, a member of class IDesigner60.
	CutSelection (see page 11156)	This is CutSelection, a member of class IDesigner60.
	DeleteSelection (see page 11157)	This is DeleteSelection, a member of class IDesigner60.
	Edit (see page 11157)	This is Edit, a member of class IDesigner60.
	FindRootAncestor (see page 11168)	This is FindRootAncestor, a member of class IDesigner70.
	GetActiveClassGroup (see page 11169)	This is GetActiveClassGroup, a member of class IDesigner70.
	GetAncestorDesigner (see page 11157)	This is GetAncestorDesigner, a member of class IDesigner60.
	GetBaseRegKey (see page 11157)	This is GetBaseRegKey, a member of class IDesigner60.
	GetComponent (see page 11157)	This is GetComponent, a member of class IDesigner60.
	GetComponentName (see page 11158)	This is GetComponentName, a member of class IDesigner60.
	GetComponentNames (see page 11158)	This is GetComponentNames, a member of class IDesigner60.
	GetIDEOptions (see page 11158)	This is GetIDEOptions, a member of class IDesigner60.
	GetIsDormant (see page 11158)	This is GetIsDormant, a member of class IDesigner60.
	GetMethodName (see page 11159)	This is GetMethodName, a member of class IDesigner60.
	GetMethods (see page 11159)	This is GetMethods, a member of class IDesigner60.
	GetObject (see page 11159)	This is GetObject, a member of class IDesigner60.
	GetObjectName (see page 11159)	This is GetObjectName, a member of class IDesigner60.
	GetObjectNames (see page 11159)	This is GetObjectNames, a member of class IDesigner60.
	GetPathAndBaseExeName (see page 11160)	This is GetPathAndBaseExeName, a member of class IDesigner60.
	GetPrivateDirectory (see page 11160)	This is GetPrivateDirectory, a member of class IDesigner60.

	GetProjectModules (see page 11160)	This is GetProjectModules, a member of class IDesigner60.
	GetRoot (see page 11160)	This is GetRoot, a member of class IDesigner60.
	GetRootClassName (see page 11161)	This is GetRootClassName, a member of class IDesigner60.
	GetScrollRanges (see page 11161)	This is GetScrollRanges, a member of class IDesigner60.
	GetSelections (see page 11161)	This is GetSelections, a member of class IDesigner60.
	GetShiftState (see page 11161)	This is GetShiftState, a member of class IDesigner60.
	IsComponentHidden (see page 11161)	This is IsComponentHidden, a member of class IDesigner60.
	IsComponentLinkable (see page 11162)	This is IsComponentLinkable, a member of class IDesigner60.
	IsSourceReadOnly (see page 11162)	This is IsSourceReadOnly, a member of class IDesigner60.
	MakeComponentLinkable (see page 11162)	This is MakeComponentLinkable, a member of class IDesigner60.
	MethodExists (see page 11162)	This is MethodExists, a member of class IDesigner60.
	MethodFromAncestor (see page 11163)	This is MethodFromAncestor, a member of class IDesigner60.
	ModalEdit (see page 11163)	This is ModalEdit, a member of class IDesigner60.
	Modified (see page 11163)	This is Modified, a member of class IDesigner60.
	ModuleFileNames (see page 11163)	This is ModuleFileNames, a member of class IDesigner60.
	NoSelection (see page 11163)	This is NoSelection, a member of class IDesigner60.
	PasteSelection (see page 11164)	This is PasteSelection, a member of class IDesigner60.
	RenameMethod (see page 11164)	This is RenameMethod, a member of class IDesigner60.
	Resurrect (see page 11164)	This is Resurrect, a member of class IDesigner60.
	Revert (see page 11164)	This is Revert, a member of class IDesigner60.
	SelectComponent (see page 11165)	This is SelectComponent, a member of class IDesigner60.
	SelectItemName (see page 11165)	This is SelectItemName, a member of class IDesigner60.
	SetSelections (see page 11165)	This is SetSelections, a member of class IDesigner60.
	ShowMethod (see page 11165)	This is ShowMethod, a member of class IDesigner60.
	UniqueName (see page 11165)	This is UniqueName, a member of class IDesigner60.

Public Properties

	Name	Description
R	ActiveClassGroup (see page 11169)	This is ActiveClassGroup, a member of class IDesigner70.
R	AncestorDesigner (see page 11166)	This is AncestorDesigner, a member of class IDesigner60.
R	IsDormant (see page 11166)	This is IsDormant, a member of class IDesigner60.
R	Root (see page 11166)	This is Root, a member of class IDesigner60.

60.2.9.2 IDesigner70 Methods

The methods of the IDesigner70 class are listed here.

Public Methods

	Name	Description
	FindRootAncestor (see page 11168)	This is FindRootAncestor, a member of class IDesigner70.
	GetActiveClassGroup (see page 11169)	This is GetActiveClassGroup, a member of class IDesigner70.

60.2.9.2.1 IDesigner70.FindRootAncestor Method

Pascal

```
function FindRootAncestor(const AClassName: string): TComponent;
```

C++

```
__fastcall TComponent * FindRootAncestor(const AnsiString AClassName);
```

Description

This is FindRootAncestor, a member of class IDesigner70.

60.2.9.2.2 IDesigner70.GetActiveClassGroup Method

Pascal

```
function GetActiveClassGroup: TPersistentClass;
```

C++

```
__fastcall TPersistentClass GetActiveClassGroup();
```

Description

This is GetActiveClassGroup, a member of class IDesigner70.

60.2.9.3 IDesigner70 Properties

The properties of the IDesigner70 class are listed here.

Public Properties

	Name	Description
	ActiveClassGroup (see page 11169)	This is ActiveClassGroup, a member of class IDesigner70.

60.2.9.3.1 IDesigner70.ActiveClassGroup Property

Pascal

```
property ActiveClassGroup: TPersistentClass;
```

C++

```
__property TPersistentClass ActiveClassGroup;
```

Description

This is ActiveClassGroup, a member of class IDesigner70.

60.2.10 IDesigner80 Interface

Class Hierarchy

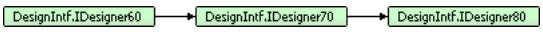

Pascal

```
IDesigner80 = interface(IDesigner70);
```

C++

```
__interface IDesigner80 : public IDesigner70;
```

File

DesignIntf

Description

This is class DesignIntf.IDesigner80.

60.2.10.1 IDesigner80 Members

The following tables list the members exposed by IDesigner80.

Public Methods

	Name	Description
	Activate (see page 11154)	This is Activate, a member of class IDesigner60.
	CanPaste (see page 11154)	This is CanPaste, a member of class IDesigner60.
	ChainCall (see page 11154)	This is the overview for the ChainCall method overload.
	ClearSelection (see page 11155)	This is ClearSelection, a member of class IDesigner60.
	CopySelection (see page 11155)	This is CopySelection, a member of class IDesigner60.
	CreateComponent (see page 11156)	This is CreateComponent, a member of class IDesigner60.
	CreateCurrentComponent (see page 11156)	This is CreateCurrentComponent, a member of class IDesigner60.
	CreateMethod (see page 11171)	This is CreateMethod, a member of class IDesigner80.
	CutSelection (see page 11156)	This is CutSelection, a member of class IDesigner60.
	DeleteSelection (see page 11157)	This is DeleteSelection, a member of class IDesigner60.
	Edit (see page 11157)	This is Edit, a member of class IDesigner60.
	FindRootAncestor (see page 11168)	This is FindRootAncestor, a member of class IDesigner70.
	GetActiveClassGroup (see page 11169)	This is GetActiveClassGroup, a member of class IDesigner70.
	GetAncestorDesigner (see page 11157)	This is GetAncestorDesigner, a member of class IDesigner60.
	GetBaseRegKey (see page 11157)	This is GetBaseRegKey, a member of class IDesigner60.
	GetComponent (see page 11157)	This is GetComponent, a member of class IDesigner60.
	GetComponentName (see page 11158)	This is GetComponentName, a member of class IDesigner60.
	GetComponentNames (see page 11158)	This is GetComponentNames, a member of class IDesigner60.
	GetIDEOptions (see page 11158)	This is GetIDEOptions, a member of class IDesigner60.
	GetIsDormant (see page 11158)	This is GetIsDormant, a member of class IDesigner60.
	GetMethodName (see page 11159)	This is GetMethodName, a member of class IDesigner60.
	GetMethods (see page 11171)	This is GetMethods, a member of class IDesigner80.
	GetObject (see page 11159)	This is GetObject, a member of class IDesigner60.
	GetObjectName (see page 11159)	This is GetObjectName, a member of class IDesigner60.
	GetObjectNames (see page 11159)	This is GetObjectNames, a member of class IDesigner60.
	GetPathAndBaseExeName (see page 11160)	This is GetPathAndBaseExeName, a member of class IDesigner60.
	GetPrivateDirectory (see page 11160)	This is GetPrivateDirectory, a member of class IDesigner60.
	GetProjectModules (see page 11160)	This is GetProjectModules, a member of class IDesigner60.
	GetRoot (see page 11160)	This is GetRoot, a member of class IDesigner60.
	GetRootClassName (see page 11161)	This is GetRootClassName, a member of class IDesigner60.
	GetScrollRanges (see page 11161)	This is GetScrollRanges, a member of class IDesigner60.
	GetSelections (see page 11161)	This is GetSelections, a member of class IDesigner60.
	GetShiftState (see page 11161)	This is GetShiftState, a member of class IDesigner60.
	IsComponentHidden (see page 11161)	This is IsComponentHidden, a member of class IDesigner60.
	IsComponentLinkable (see page 11162)	This is IsComponentLinkable, a member of class IDesigner60.
	IsSourceReadOnly (see page 11162)	This is IsSourceReadOnly, a member of class IDesigner60.
	MakeComponentLinkable (see page 11162)	This is MakeComponentLinkable, a member of class IDesigner60.
	MethodExists (see page 11162)	This is MethodExists, a member of class IDesigner60.
	MethodFromAncestor (see page 11163)	This is MethodFromAncestor, a member of class IDesigner60.
	ModalEdit (see page 11163)	This is ModalEdit, a member of class IDesigner60.
	Modified (see page 11163)	This is Modified, a member of class IDesigner60.
	ModuleFileNames (see page 11163)	This is ModuleFileNames, a member of class IDesigner60.
	NoSelection (see page 11163)	This is NoSelection, a member of class IDesigner60.
	PasteSelection (see page 11164)	This is PasteSelection, a member of class IDesigner60.
	RenameMethod (see page 11164)	This is RenameMethod, a member of class IDesigner60.
	Resurrect (see page 11164)	This is Resurrect, a member of class IDesigner60.
	Revert (see page 11164)	This is Revert, a member of class IDesigner60.

	SelectComponent (see page 11171)	This is SelectComponent, a member of class IDesigner80.
	SelectItemName (see page 11165)	This is SelectItemName, a member of class IDesigner60.
	SetSelections (see page 11165)	This is SetSelections, a member of class IDesigner60.
	ShowMethod (see page 11165)	This is ShowMethod, a member of class IDesigner60.
	UniqueName (see page 11165)	This is UniqueName, a member of class IDesigner60.

Public Properties

	Name	Description
R	ActiveClassGroup (see page 11169)	This is ActiveClassGroup, a member of class IDesigner70.
R	AncestorDesigner (see page 11166)	This is AncestorDesigner, a member of class IDesigner60.
R	IsDormant (see page 11166)	This is IsDormant, a member of class IDesigner60.
R	Root (see page 11166)	This is Root, a member of class IDesigner60.

60.2.10.2 IDesigner80 Methods

The methods of the IDesigner80 class are listed here.

Public Methods

	Name	Description
	CreateMethod (see page 11171)	This is CreateMethod, a member of class IDesigner80.
	GetMethods (see page 11171)	This is GetMethods, a member of class IDesigner80.
	SelectComponent (see page 11171)	This is SelectComponent, a member of class IDesigner80.

60.2.10.2.1 IDesigner80.CreateMethod Method

Pascal

```
function CreateMethod(const Name: string; const AEventInfo: IEventInfo): TMethod; overload;
```

C++

```
__fastcall TMethod CreateMethod(const AnsiString Name, const IEventInfo AEventInfo);
```

Description

This is CreateMethod, a member of class IDesigner80.

60.2.10.2.2 IDesigner80.GetMethods Method

Pascal

```
procedure GetMethods(const AEventInfo: IEventInfo; Proc: TGetStrProc); overload;
```

C++

```
__fastcall GetMethods(const IEventInfo AEventInfo, TGetStrProc Proc);
```

Description

This is GetMethods, a member of class IDesigner80.

60.2.10.2.3 IDesigner80.SelectComponent Method

Pascal

```
procedure SelectComponent(const ADesignObject: IDesignObject); overload;
```

C++

```
__fastcall SelectComponent(const IDesignObject ADesignObject);
```

Description

This is SelectComponent, a member of class IDesigner80.

60.2.11 IDesignerSelections Interface

IDesignerSelections represents a list of persistent objects.

Class Hierarchy

DesignIntf.IDesignerSelections

Pascal

```
IDesignerSelections = interface;
```

C++

```
__interface IDesignerSelections;
```

File

DesignIntf

Description

The form designer uses IDesignerSelections to get or set a list of those objects that are selected.

IDesignerSelections provides properties and methods to

Add (see page 11173) objects to the list.

Compare two lists of objects.

Access individual objects in the list.

Note: C++ method declarations that use IDesignerSelections use the _di_IDesignerSelections type instead. This is a DelphiInterface wrapper around the IDesignerSelections interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IDesignerSelections >
_di_IDesignerSelections;
```

See Also

TDesignerSelections (see page 11121)

60.2.11.1 IDesignerSelections Members

The following tables list the members exposed by IDesignerSelections.

Public Methods

	Name	Description
	Add (see page 11173)	Adds a new object to the end of the list.
	Equals (see page 11173)	Indicates whether the selections list is identical to another selections list.
	Get (see page 11174)	Returns a specified item from the list.
	GetCount (see page 11174)	Returns the value of the Count (see page 11175) property.
	GetDesignObject (see page 11174)	This is GetDesignObject, a member of class IDesignerSelections.

Public Properties

	Name	Description
R	Count (see page 11175)	Indicates the number of entries in the list.
R	DesignObjects (see page 11175)	This is DesignObjects, a member of class IDesignerSelections.
R	Items (see page 11175)	Provides indexed access to the objects in the list.

60.2.11.2 IDesignerSelections Methods

The methods of the IDesignerSelections class are listed here.

Public Methods

	Name	Description
	Add (see page 11173)	Adds a new object to the end of the list.
	Equals (see page 11173)	Indicates whether the selections list is identical to another selections list.
	Get (see page 11174)	Returns a specified item from the list.
	GetCount (see page 11174)	Returns the value of the Count (see page 11175) property.
	GetDesignObject (see page 11174)	This is GetDesignObject, a member of class IDesignerSelections.

60.2.11.2.1 IDesignerSelections.Add Method

Adds a new object to the end of the list.

Pascal

```
function Add(const Item: TPersistent): Integer;
```

C++

```
__fastcall int Add(const TPersistent * Item);
```

Description

Call Add to insert a new object at the end of the Items (see page 11175) array. Add returns the index of the new item, where the first item in the list has an index of 0. Add increases the value of Count (see page 11175) to reflect the addition of a new object.

See Also

Count (see page 11175)

Items (see page 11175)

60.2.11.2.2 IDesignerSelections.Equals Method

Indicates whether the selections list is identical to another selections list.

Pascal

```
function Equals(const List: IDesignerSelections): Boolean;
```

C++

```
__fastcall Boolean Equals(const IDesignerSelections List);
```

Description

Call Equals to determine whether the list specified by the List parameter contains exactly the same objects in the same order as this list. Equals returns false if the other list

Contains objects not in the Items (see page 11175) array.

Contains a subset of the objects in the Items (see page 11175) array.

Contains the same objects but in a different order.

See Also

Items (see page 11175)

60.2.11.2.3 IDesignerSelections.Get Method

Returns a specified item from the list.

Pascal

```
function Get(Index: Integer): TPersistent;
```

C++

```
__fastcall TPersistent * Get(int Index);
```

Description

Get is the protected read access method for the Items (see page 11175) property.

Index identifies the desired object by its position in the list, where 0 identifies the first position. Get returns the specified object.

See Also

Items (see page 11175)

60.2.11.2.4 IDesignerSelections.GetCount Method

Returns the value of the Count (see page 11175) property.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the protected read access method for the Count (see page 11175) property.

See Also

Count (see page 11175)

60.2.11.2.5 IDesignerSelections.GetDesignObject Method

Pascal

```
function GetDesignObject(Index: Integer): IDesignObject;
```

C++

```
__fastcall IDesignObject GetDesignObject(int Index);
```

Description

This is GetDesignObject, a member of class IDesignerSelections.

60.2.11.3 IDesignerSelections Properties

The properties of the IDesignerSelections class are listed here.

Public Properties

	Name	Description
R	Count (see page 11175)	Indicates the number of entries in the list.
R	DesignObjects (see page 11175)	This is DesignObjects, a member of class IDesignerSelections.
R	Items (see page 11175)	Provides indexed access to the objects in the list.

60.2.11.3.1 IDesignerSelections.Count Property

Indicates the number of entries in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read Count to determine the number of entries in the Items (see page 11175) array. Use Count as an upper limit when iterating over all the objects in the list.

Count is a read-only property. To increase the number of objects in the list, use the Add (see page 11173) method. Objects can not be removed from the list once they have been added.

See Also

Items (see page 11175)

Add (see page 11173)

60.2.11.3.2 IDesignerSelections.DesignObjects Property

Pascal

```
property DesignObjects [Index: Integer]: IDesignObject;
```

C++

```
__property IDesignObject DesignObjects[int Index];
```

Description

This is DesignObjects, a member of class IDesignerSelections.

60.2.11.3.3 IDesignerSelections.Items Property

Provides indexed access to the objects in the list.

Pascal

```
property Items [Index: Integer]: TPersistent;
```

C++

```
__property TPersistent * Items[int Index];
```

Description

Use Items to access a specific object in the list. The Index parameter indicates the index of the object, where 0 is the index of the first object, 1 is the index of the second object, and so on. Use Items with the Count (see page 11175) property to iterate through all of the objects in the list.

To add objects to the Items array, use the Add (see page 11173) method. Once objects have been added, they can't be deleted or rearranged.

See Also

Count (see page 11175)

Add (see page 11173)

60.2.12 IDesignNotification Interface

Class Hierarchy

DesignIntf.IDesignNotification

Pascal

```
IDesignNotification = interface;
```

C++

```
__interface IDesignNotification;
```

File

DesignIntf

Description

This is class DesignIntf.IDesignNotification.

60.2.12.1 IDesignNotification Members

The following tables list the members exposed by IDesignNotification.

Public Methods

	Name	Description
	DesignerClosed (see page 11177)	This is DesignerClosed, a member of class IDesignNotification.
	DesignerOpened (see page 11177)	This is DesignerOpened, a member of class IDesignNotification.
	ItemDeleted (see page 11177)	This is ItemDeleted, a member of class IDesignNotification.
	ItemInserted (see page 11177)	This is ItemInserted, a member of class IDesignNotification.
	ItemsModified (see page 11178)	This is ItemsModified, a member of class IDesignNotification.
	SelectionChanged (see page 11178)	This is SelectionChanged, a member of class IDesignNotification.

60.2.12.2 IDesignNotification Methods

The methods of the IDesignNotification class are listed here.

Public Methods

	Name	Description
	DesignerClosed (see page 11177)	This is DesignerClosed, a member of class IDesignNotification.
	DesignerOpened (see page 11177)	This is DesignerOpened, a member of class IDesignNotification.
	ItemDeleted (see page 11177)	This is ItemDeleted, a member of class IDesignNotification.
	ItemInserted (see page 11177)	This is ItemInserted, a member of class IDesignNotification.
	ItemsModified (see page 11178)	This is ItemsModified, a member of class IDesignNotification.
	SelectionChanged (see page 11178)	This is SelectionChanged, a member of class IDesignNotification.

60.2.12.2.1 IDesignNotification.DesignerClosed Method

Pascal

```
procedure DesignerClosed(const ADesigner: IDesigner; AGoingDormant: Boolean);
```

C++

```
__fastcall DesignerClosed(const IDesigner ADesigner, Boolean AGoingDormant);
```

Description

This is DesignerClosed, a member of class IDesignNotification.

60.2.12.2.2 IDesignNotification.DesignerOpened Method

Pascal

```
procedure DesignerOpened(const ADesigner: IDesigner; AResurrecting: Boolean);
```

C++

```
__fastcall DesignerOpened(const IDesigner ADesigner, Boolean AResurrecting);
```

Description

This is DesignerOpened, a member of class IDesignNotification.

60.2.12.2.3 IDesignNotification.ItemDeleted Method

Pascal

```
procedure ItemDeleted(const ADesigner: IDesigner; AItem: TPersistent);
```

C++

```
__fastcall ItemDeleted(const IDesigner ADesigner, TPersistent * AItem);
```

Description

This is ItemDeleted, a member of class IDesignNotification.

60.2.12.2.4 IDesignNotification.ItemInserted Method

Pascal

```
procedure ItemInserted(const ADesigner: IDesigner; AItem: TPersistent);
```

C++

```
__fastcall ItemInserted(const IDesigner ADesigner, TPersistent * AItem);
```

Description

This is ItemInserted, a member of class IDesignNotification.

60.2.12.2.5 IDesignNotification.ItemsModified Method

Pascal

```
procedure ItemsModified(const ADesigner: IDesigner);
```

C++

```
__fastcall ItemsModified(const IDesigner ADesigner);
```

Description

This is ItemsModified, a member of class IDesignNotification.

60.2.12.2.6 IDesignNotification.SelectionChanged Method

Pascal

```
procedure SelectionChanged(const ADesigner: IDesigner; const ASelection: IDesignerSelections);
```

C++

```
__fastcall SelectionChanged(const IDesigner ADesigner, const IDesignerSelections ASelection);
```

Description

This is SelectionChanged, a member of class IDesignNotification.

60.2.13 IDesignObject Interface

Class Hierarchy

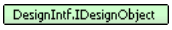

Pascal

```
IDesignObject = interface;
```

C++

```
__interface IDesignObject;
```

File

DesignIntf

Description

This is class DesignIntf.IDesignObject.

60.2.13.1 IDesignObject Members

The following tables list the members exposed by IDesignObject.

Public Methods

	Name	Description
	Equals (see page 11179)	This is the overview for the Equals method overload.
	GetClassName (see page 11180)	This is GetClassName, a member of class IDesignObject.
	GetClassType (see page 11180)	This is GetClassType, a member of class IDesignObject.
	GetComponentIndex (see page 11180)	This is GetComponentIndex, a member of class IDesignObject.
	GetComponentName (see page 11180)	This is GetComponentName, a member of class IDesignObject.
	GetIsComponent (see page 11181)	This is GetIsComponent, a member of class IDesignObject.
	GetNamePath (see page 11181)	This is GetNamePath, a member of class IDesignObject.

Public Properties

	Name	Description
R	ClassName (see page 11181)	This is ClassName, a member of class IDesignObject.
R	ClassType (see page 11182)	This is ClassType, a member of class IDesignObject.
R	ComponentIndex (see page 11182)	This is ComponentIndex, a member of class IDesignObject.
R	ComponentName (see page 11182)	This is ComponentName, a member of class IDesignObject.
R	IsComponent (see page 11182)	This is IsComponent, a member of class IDesignObject.
R	NamePath (see page 11182)	This is NamePath, a member of class IDesignObject.

60.2.13.2 IDesignObject Methods

The methods of the IDesignObject class are listed here.

Public Methods

	Name	Description
	Equals (see page 11179)	This is the overview for the Equals method overload.
	GetClassName (see page 11180)	This is GetClassName, a member of class IDesignObject.
	GetClassType (see page 11180)	This is GetClassType, a member of class IDesignObject.
	GetComponentIndex (see page 11180)	This is GetComponentIndex, a member of class IDesignObject.
	GetComponentName (see page 11180)	This is GetComponentName, a member of class IDesignObject.
	GetIsComponent (see page 11181)	This is GetIsComponent, a member of class IDesignObject.
	GetNamePath (see page 11181)	This is GetNamePath, a member of class IDesignObject.

60.2.13.2.1 Equals Method

This is the overview for the Equals method overload.

Overload List

	Name	Description
	IDesignObject.Equals (IDesignObject) (see page 11179)	This is Equals, a member of class IDesignObject.
	IDesignObject.Equals (TObject) (see page 11180)	This is Equals, a member of class IDesignObject.

60.2.13.2.1.1 IDesignObject.Equals Method (IDesignObject)

Pascal

```
function Equals(const ADesignObject: IDesignObject): Boolean; overload;
```

C++

```
__fastcall Boolean Equals(const IDesignObject ADesignObject);
```

Description

This is Equals, a member of class IDesignObject.

60.2.13.2.1.2 IDesignObject.Equals Method (TObject)

Pascal

```
function Equals(Obj: TObject): Boolean; overload;
```

C++

```
__fastcall Boolean Equals(TObject * Obj);
```

Description

This is Equals, a member of class IDesignObject.

60.2.13.2.2 IDesignObject.GetClassName Method

Pascal

```
function GetClassName: string;
```

C++

```
__fastcall AnsiString GetClassName();
```

Description

This is GetClassName, a member of class IDesignObject.

60.2.13.2.3 IDesignObject.GetClassType Method

Pascal

```
function GetClassType: IClass;
```

C++

```
__fastcall IClass GetClassType();
```

Description

This is GetClassType, a member of class IDesignObject.

60.2.13.2.4 IDesignObject.GetComponentIndex Method

Pascal

```
function GetComponentIndex: Integer;
```

C++

```
__fastcall int GetComponentIndex();
```

Description

This is GetComponentIndex, a member of class IDesignObject.

60.2.13.2.5 IDesignObject.GetComponentName Method

Pascal

```
function GetComponentName: string;
```

C++

```
__fastcall AnsiString GetComponentName();
```

Description

This is GetComponentName, a member of class IDesignObject.

60.2.13.2.6 IDesignObject.GetIsComponent Method

Pascal

```
function GetIsComponent: Boolean;
```

C++

```
__fastcall Boolean GetIsComponent();
```

Description

This is GetIsComponent, a member of class IDesignObject.

60.2.13.2.7 IDesignObject.GetNamePath Method

Pascal

```
function GetNamePath: string;
```

C++

```
__fastcall AnsiString GetNamePath();
```

Description

This is GetNamePath, a member of class IDesignObject.

60.2.13.3 IDesignObject Properties

The properties of the IDesignObject class are listed here.

Public Properties

	Name	Description
R	ClassName (see page 11181)	This is ClassName, a member of class IDesignObject.
R	ClassType (see page 11182)	This is ClassType, a member of class IDesignObject.
R	ComponentIndex (see page 11182)	This is ComponentIndex, a member of class IDesignObject.
R	ComponentName (see page 11182)	This is ComponentName, a member of class IDesignObject.
R	IsComponent (see page 11182)	This is IsComponent, a member of class IDesignObject.
R	NamePath (see page 11182)	This is NamePath, a member of class IDesignObject.

60.2.13.3.1 IDesignObject.ClassName Property

Pascal

```
property ClassName: string;
```

C++

```
__property AnsiString ClassName;
```

Description

This is ClassName, a member of class IDesignObject.

60.2.13.3.2 IDesignObject.ClassType Property

Pascal

```
property ClassType: IClass;
```

C++

```
__property IClass ClassType;
```

Description

This is ClassType, a member of class IDesignObject.

60.2.13.3.3 IDesignObject.ComponentIndex Property

Pascal

```
property ComponentIndex: Integer;
```

C++

```
__property int ComponentIndex;
```

Description

This is ComponentIndex, a member of class IDesignObject.

60.2.13.3.4 IDesignObject.ComponentName Property

Pascal

```
property ComponentName: string;
```

C++

```
__property AnsiString ComponentName;
```

Description

This is ComponentName, a member of class IDesignObject.

60.2.13.3.5 IDesignObject.IsComponent Property

Pascal

```
property IsComponent: Boolean;
```

C++

```
__property Boolean IsComponent;
```

Description

This is IsComponent, a member of class IDesignObject.

60.2.13.3.6 IDesignObject.NamePath Property

Pascal

```
property NamePath: string;
```

C++

```
__property AnsiString NamePath;
```

Description

This is NamePath, a member of class IDesignObject.

60.2.14 IDesignPersistent Interface

Class Hierarchy

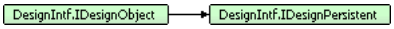

Pascal

```
IDesignPersistent = interface(IDesignObject);
```

C++

```
__interface IDesignPersistent : public IDesignObject;
```

File

DesignIntf

Description

This is class DesignIntf.IDesignPersistent.

60.2.14.1 IDesignPersistent Members

The following tables list the members exposed by IDesignPersistent.

Public Methods

	Name	Description
	Equals (see page 11179)	This is the overview for the Equals method overload.
	GetClassName (see page 11180)	This is GetClassName, a member of class IDesignObject.
	GetClassType (see page 11180)	This is GetClassType, a member of class IDesignObject.
	GetComponentIndex (see page 11180)	This is GetComponentIndex, a member of class IDesignObject.
	GetComponentName (see page 11180)	This is GetComponentName, a member of class IDesignObject.
	GetIsComponent (see page 11181)	This is GetIsComponent, a member of class IDesignObject.
	GetNamePath (see page 11181)	This is GetNamePath, a member of class IDesignObject.
	GetPersistent (see page 11184)	This is GetPersistent, a member of class IDesignPersistent.

Public Properties

	Name	Description
R	ClassName (see page 11181)	This is ClassName, a member of class IDesignObject.
R	ClassType (see page 11182)	This is ClassType, a member of class IDesignObject.
R	ComponentIndex (see page 11182)	This is ComponentIndex, a member of class IDesignObject.
R	ComponentName (see page 11182)	This is ComponentName, a member of class IDesignObject.
R	IsComponent (see page 11182)	This is IsComponent, a member of class IDesignObject.
R	NamePath (see page 11182)	This is NamePath, a member of class IDesignObject.
R	Persistent (see page 11184)	This is Persistent, a member of class IDesignPersistent.

60.2.14.2 IDesignPersistent Methods

The methods of the IDesignPersistent class are listed here.

Public Methods

	Name	Description
	GetPersistent (see page 11184)	This is GetPersistent, a member of class IDesignPersistent.

60.2.14.2.1 IDesignPersistent.GetPersistent Method

Pascal

```
function GetPersistent: TPersistent;
```

C++

```
__fastcall TPersistent * GetPersistent();
```

Description

This is GetPersistent, a member of class IDesignPersistent.

60.2.14.3 IDesignPersistent Properties

The properties of the IDesignPersistent class are listed here.

Public Properties

	Name	Description
R	Persistent (see page 11184)	This is Persistent, a member of class IDesignPersistent.

60.2.14.3.1 IDesignPersistent.Persistent Property

Pascal

```
property Persistent: TPersistent;
```

C++

```
__property TPersistent * Persistent;
```

Description

This is Persistent, a member of class IDesignPersistent.

60.2.15 IDesignWindow Interface

Class Hierarchy

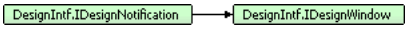

Pascal

```
IDesignWindow = interface(IDesignNotification);
```

C++

```
__interface IDesignWindow : public IDesignNotification;
```

File

DesignIntf

Description

This is class DesignIntf.IDesignWindow.

60.2.15.1 IDesignWindow Members

The following tables list the members exposed by IDesignWindow.

Public Methods

	Name	Description
	DesignerClosed (see page 11177)	This is DesignerClosed, a member of class IDesignNotification.
	DesignerOpened (see page 11177)	This is DesignerOpened, a member of class IDesignNotification.
	ItemDeleted (see page 11177)	This is ItemDeleted, a member of class IDesignNotification.
	ItemInserted (see page 11177)	This is ItemInserted, a member of class IDesignNotification.
	ItemsModified (see page 11178)	This is ItemsModified, a member of class IDesignNotification.
	SelectionChanged (see page 11178)	This is SelectionChanged, a member of class IDesignNotification.
	WindowHide (see page 11185)	This is WindowHide, a member of class IDesignWindow.
	WindowShow (see page 11185)	This is WindowShow, a member of class IDesignWindow.

60.2.15.2 IDesignWindow Methods

The methods of the IDesignWindow class are listed here.

Public Methods

	Name	Description
	WindowHide (see page 11185)	This is WindowHide, a member of class IDesignWindow.
	WindowShow (see page 11185)	This is WindowShow, a member of class IDesignWindow.

60.2.15.2.1 IDesignWindow.WindowHide Method

Pascal

```
procedure WindowHide;
```

C++

```
__fastcall WindowHide();
```

Description

This is WindowHide, a member of class IDesignWindow.

60.2.15.2.2 IDesignWindow.WindowShow Method

Pascal

```
procedure WindowShow;
```

C++

```
__fastcall WindowShow();
```

Description

This is WindowShow, a member of class IDesignWindow.

60.2.16 IEditHandler Interface

Class Hierarchy

DesignIntf.IEditHandler

Pascal

```
IEditHandler = interface;
```

C++

```
__interface IEditHandler;
```

File

DesignIntf

Description

This is class DesignIntf.IEditHandler.

60.2.16.1 IEditHandler Members

The following tables list the members exposed by IEditHandler.

Public Methods

	Name	Description
	EditAction (see page 11186)	This is EditAction, a member of class IEditHandler.
	GetEditState (see page 11187)	This is GetEditState, a member of class IEditHandler.

60.2.16.2 IEditHandler Methods

The methods of the IEditHandler class are listed here.

Public Methods

	Name	Description
	EditAction (see page 11186)	This is EditAction, a member of class IEditHandler.
	GetEditState (see page 11187)	This is GetEditState, a member of class IEditHandler.

60.2.16.2.1 IEditHandler.EditAction Method

Pascal

```
function EditAction(Action: TEditAction): Boolean;
```

C++

```
__fastcall Boolean EditAction(TEditAction Action);
```

Description

This is EditAction, a member of class IEditHandler.

60.2.16.2.2 IEditHandler.GetEditState Method

Pascal

```
function GetEditState: TEditState;
```

C++

```
__fastcall TEditState GetEditState();
```

Description

This is GetEditState, a member of class IEditHandler.

60.2.17 IEventInfo Interface

Class Hierarchy

DesignIntf.IEventInfo

Pascal

```
IEventInfo = interface;
```

C++

```
__interface IEventInfo;
```

File

DesignIntf

Description

This is class DesignIntf.IEventInfo.

60.2.17.1 IEventInfo Members

The following tables list the members exposed by IEventInfo.

Public Methods

	Name	Description
	GetMethodKind (see page 11188)	This is GetMethodKind, a member of class IEventInfo.
	GetParamCount (see page 11188)	This is GetParamCount, a member of class IEventInfo.
	GetParamFlags (see page 11188)	This is GetParamFlags, a member of class IEventInfo.
	GetParamName (see page 11188)	This is GetParamName, a member of class IEventInfo.
	GetParamType (see page 11189)	This is GetParamType, a member of class IEventInfo.
	GetResultType (see page 11189)	This is GetResultType, a member of class IEventInfo.

60.2.17.2 IEventInfo Methods

The methods of the IEventInfo class are listed here.

Public Methods

	Name	Description
	GetMethodKind (see page 11188)	This is GetMethodKind, a member of class IEventInfo.

	GetParamCount (see page 11188)	This is GetParamCount, a member of class IEventInfo.
	GetParamFlags (see page 11188)	This is GetParamFlags, a member of class IEventInfo.
	GetParamName (see page 11188)	This is GetParamName, a member of class IEventInfo.
	GetParamType (see page 11189)	This is GetParamType, a member of class IEventInfo.
	GetResultType (see page 11189)	This is GetResultType, a member of class IEventInfo.

60.2.17.2.1 IEventInfo.GetMethodKind Method

Pascal

```
function GetMethodKind: TMethodKind;
```

C++

```
__fastcall TMethodKind GetMethodKind();
```

Description

This is GetMethodKind, a member of class IEventInfo.

60.2.17.2.2 IEventInfo.GetParamCount Method

Pascal

```
function GetParamCount: Integer;
```

C++

```
__fastcall int GetParamCount();
```

Description

This is GetParamCount, a member of class IEventInfo.

60.2.17.2.3 IEventInfo.GetParamFlags Method

Pascal

```
function GetParamFlags(Index: Integer): TParamFlags;
```

C++

```
__fastcall TParamFlags GetParamFlags(int Index);
```

Description

This is GetParamFlags, a member of class IEventInfo.

60.2.17.2.4 IEventInfo.GetParamName Method

Pascal

```
function GetParamName(Index: Integer): string;
```

C++

```
__fastcall AnsiString GetParamName(int Index);
```

Description

This is GetParamName, a member of class IEventInfo.

60.2.17.2.5 IEventInfo.GetParamType Method

Pascal

```
function GetParamType(Index: Integer): string;
```

C++

```
__fastcall AnsiString GetParamType(int Index);
```

Description

This is GetParamType, a member of class IEventInfo.

60.2.17.2.6 IEventInfo.GetResultType Method

Pascal

```
function GetResultType: string;
```

C++

```
__fastcall AnsiString GetResultType();
```

Description

This is GetResultType, a member of class IEventInfo.

60.2.18 IHostForm Interface

Class Hierarchy

DesignIntf.IHostForm

Pascal

```
IHostForm = interface;
```

C++

```
__interface IHostForm;
```

File

DesignIntf

Description

This is class DesignIntf.IHostForm.

60.2.18.1 IHostForm Members

The following tables list the members exposed by IHostForm.

Public Methods

	Name	Description
	BringToFront (see page 11191)	This is BringToFront, a member of class IHostForm.
	CheckPosChanged (see page 11191)	This is CheckPosChanged, a member of class IHostForm.
	DeinitializeDesigner (see page 11191)	This is DeinitializeDesigner, a member of class IHostForm.

	GetCanPrint (see page 11191)	This is GetCanPrint, a member of class IHostForm.
	GetCaption (see page 11191)	This is GetCaption, a member of class IHostForm.
	GetDesignerState (see page 11192)	This is GetDesignerState, a member of class IHostForm.
	GetFont (see page 11192)	This is GetFont, a member of class IHostForm.
	GetForm (see page 11192)	This is GetForm, a member of class IHostForm.
	GetFormImage (see page 11192)	This is GetFormImage, a member of class IHostForm.
	GetScrollPos (see page 11193)	This is GetScrollPos, a member of class IHostForm.
	GetVisible (see page 11193)	This is GetVisible, a member of class IHostForm.
	GetWindowState (see page 11193)	This is GetWindowState, a member of class IHostForm.
	HideWindow (see page 11193)	This is HideWindow, a member of class IHostForm.
	IsMenuKey (see page 11193)	This is IsMenuKey, a member of class IHostForm.
	SetCaption (see page 11194)	This is SetCaption, a member of class IHostForm.
	SetDesigner (see page 11194)	This is SetDesigner, a member of class IHostForm.
	SetDesigning (see page 11194)	This is SetDesigning, a member of class IHostForm.
	SetFormDefaults (see page 11194)	This is SetFormDefaults, a member of class IHostForm.
	Show (see page 11195)	This is Show, a member of class IHostForm.
	ShowWindow (see page 11195)	This is ShowWindow, a member of class IHostForm.
	Unmodify (see page 11195)	This is Unmodify, a member of class IHostForm.

Public Properties

	Name	Description
	Caption (see page 11195)	This is Caption, a member of class IHostForm.
R	Visible (see page 11196)	This is Visible, a member of class IHostForm.

60.2.18.2 IHostForm Methods

The methods of the IHostForm class are listed here.

Public Methods

	Name	Description
	BringToFront (see page 11191)	This is BringToFront, a member of class IHostForm.
	CheckPosChanged (see page 11191)	This is CheckPosChanged, a member of class IHostForm.
	DeinitializeDesigner (see page 11191)	This is DeinitializeDesigner, a member of class IHostForm.
	GetCanPrint (see page 11191)	This is GetCanPrint, a member of class IHostForm.
	GetCaption (see page 11191)	This is GetCaption, a member of class IHostForm.
	GetDesignerState (see page 11192)	This is GetDesignerState, a member of class IHostForm.
	GetFont (see page 11192)	This is GetFont, a member of class IHostForm.
	GetForm (see page 11192)	This is GetForm, a member of class IHostForm.
	GetFormImage (see page 11192)	This is GetFormImage, a member of class IHostForm.
	GetScrollPos (see page 11193)	This is GetScrollPos, a member of class IHostForm.
	GetVisible (see page 11193)	This is GetVisible, a member of class IHostForm.
	GetWindowState (see page 11193)	This is GetWindowState, a member of class IHostForm.
	HideWindow (see page 11193)	This is HideWindow, a member of class IHostForm.
	IsMenuKey (see page 11193)	This is IsMenuKey, a member of class IHostForm.
	SetCaption (see page 11194)	This is SetCaption, a member of class IHostForm.
	SetDesigner (see page 11194)	This is SetDesigner, a member of class IHostForm.
	SetDesigning (see page 11194)	This is SetDesigning, a member of class IHostForm.
	SetFormDefaults (see page 11194)	This is SetFormDefaults, a member of class IHostForm.
	Show (see page 11195)	This is Show, a member of class IHostForm.
	ShowWindow (see page 11195)	This is ShowWindow, a member of class IHostForm.
	Unmodify (see page 11195)	This is Unmodify, a member of class IHostForm.

60.2.18.2.1 IHostForm.BringToFront Method

Pascal

```
procedure BringToFront;
```

C++

```
__fastcall BringToFront();
```

Description

This is BringToFront, a member of class IHostForm.

60.2.18.2.2 IHostForm.CheckPosChanged Method

Pascal

```
procedure CheckPosChanged;
```

C++

```
__fastcall CheckPosChanged();
```

Description

This is CheckPosChanged, a member of class IHostForm.

60.2.18.2.3 IHostForm.DeinitializeDesigner Method

Pascal

```
procedure DeinitializeDesigner(ARoot: TComponent);
```

C++

```
__fastcall DeinitializeDesigner(TComponent * ARoot);
```

Description

This is DeinitializeDesigner, a member of class IHostForm.

60.2.18.2.4 IHostForm.GetCanPrint Method

Pascal

```
function GetCanPrint: Boolean;
```

C++

```
__fastcall Boolean GetCanPrint();
```

Description

This is GetCanPrint, a member of class IHostForm.

60.2.18.2.5 IHostForm.GetCaption Method

Pascal

```
function GetCaption: string;
```

C++

```
__fastcall AnsiString GetCaption();
```

Description

This is GetCaption, a member of class IHostForm.

60.2.18.2.6 IHostForm.GetDesignerState Method

Pascal

```
function GetDesignerState: TDesignerState;
```

C++

```
__fastcall TDesignerState GetDesignerState();
```

Description

This is GetDesignerState, a member of class IHostForm.

60.2.18.2.7 IHostForm.GetFont Method

Pascal

```
function GetFont: TPersistent;
```

C++

```
__fastcall TPersistent * GetFont();
```

Description

This is GetFont, a member of class IHostForm.

60.2.18.2.8 IHostForm.GetForm Method

Pascal

```
function GetForm: TComponent;
```

C++

```
__fastcall TComponent * GetForm();
```

Description

This is GetForm, a member of class IHostForm.

60.2.18.2.9 IHostForm.GetFormImage Method

Pascal

```
function GetFormImage: TObject;
```

C++

```
__fastcall TObject * GetFormImage();
```

Description

This is GetFormImage, a member of class IHostForm.

60.2.18.2.10 IHostForm.GetScrollPos Method

Pascal

```
function GetScrollPos(Horiz: Boolean): Integer;
```

C++

```
__fastcall int GetScrollPos(Boolean Horiz);
```

Description

This is GetScrollPos, a member of class IHostForm.

60.2.18.2.11 IHostForm.GetVisible Method

Pascal

```
function GetVisible: Boolean;
```

C++

```
__fastcall Boolean GetVisible();
```

Description

This is GetVisible, a member of class IHostForm.

60.2.18.2.12 IHostForm.GetWindowState Method

Pascal

```
function GetWindowState: TShowState;
```

C++

```
__fastcall TShowState GetWindowState();
```

Description

This is GetWindowState, a member of class IHostForm.

60.2.18.2.13 IHostForm.HideWindow Method

Pascal

```
procedure HideWindow;
```

C++

```
__fastcall HideWindow();
```

Description

This is HideWindow, a member of class IHostForm.

60.2.18.2.14 IHostForm.IsMenuKey Method

Pascal

```
function IsMenuKey(var Message: TWMKey): Boolean;
```

C++

```
__fastcall Boolean IsMenuKey(TWMKey Message);
```

Description

This is IsMenuKey, a member of class IHostForm.

60.2.18.2.15 IHostForm.SetCaption Method

Pascal

```
procedure SetCaption(const ACaption: string);
```

C++

```
__fastcall SetCaption(const AnsiString ACaption);
```

Description

This is SetCaption, a member of class IHostForm.

60.2.18.2.16 IHostForm.SetDesigner Method

Pascal

```
procedure SetDesigner(const ADesigner: IInterface);
```

C++

```
__fastcall SetDesigner(const IInterface ADesigner);
```

Description

This is SetDesigner, a member of class IHostForm.

60.2.18.2.17 IHostForm.SetDesigning Method

Pascal

```
procedure SetDesigning(DesignMode: Boolean);
```

C++

```
__fastcall SetDesigning(Boolean DesignMode);
```

Description

This is SetDesigning, a member of class IHostForm.

60.2.18.2.18 IHostForm.SetFormDefaults Method

Pascal

```
procedure SetFormDefaults(ARoot: TComponent; const ARootName: string; X: Integer; Y: Integer;
Scale: Boolean);
```

C++

```
__fastcall SetFormDefaults(TComponent * ARoot, const AnsiString ARootName, int X, int Y,
Boolean Scale);
```

Description

This is SetFormDefaults, a member of class IHostForm.

60.2.18.2.19 IHostForm.Show Method

Pascal

```
procedure Show;
```

C++

```
__fastcall Show();
```

Description

This is Show, a member of class IHostForm.

60.2.18.2.20 IHostForm.ShowWindow Method

Pascal

```
procedure ShowWindow(AShowState: TShowState);
```

C++

```
__fastcall ShowWindow(TShowState AShowState);
```

Description

This is ShowWindow, a member of class IHostForm.

60.2.18.2.21 IHostForm.Unmodify Method

Pascal

```
procedure Unmodify;
```

C++

```
__fastcall Unmodify();
```

Description

This is Unmodify, a member of class IHostForm.

60.2.18.3 IHostForm Properties

The properties of the IHostForm class are listed here.

Public Properties

	Name	Description
	Caption (see page 11195)	This is Caption, a member of class IHostForm.
R	Visible (see page 11196)	This is Visible, a member of class IHostForm.

60.2.18.3.1 IHostForm.Caption Property

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

This is Caption, a member of class IHostForm.

60.2.18.3.2 IHostForm.Visible Property

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

This is Visible, a member of class IHostForm.

60.2.19 IImplementation Interface

Class Hierarchy

DesignIntf.IImplementation

Pascal

```
IImplementation = interface;
```

C++

```
__interface IImplementation;
```

File

DesignIntf

Description

This is class DesignIntf.IImplementation.

60.2.19.1 IImplementation Members

The following tables list the members exposed by IImplementation.

Public Methods

	Name	Description
	GetInstance (see page 11197)	This is GetInstance, a member of class IImplementation.

60.2.19.2 IImplementation Methods

The methods of the IImplementation class are listed here.

Public Methods

	Name	Description
	GetInstance (see page 11197)	This is GetInstance, a member of class IImplementation.

60.2.19.2.1 IImplementation.GetInstance Method

Pascal

```
function GetInstance: TObject;
```

C++

```
__fastcall TObject * GetInstance();
```

Description

This is GetInstance, a member of class IImplementation.

60.2.20 IMethodProperty Interface

Class Hierarchy

DesignIntf.IMethodProperty

Pascal

```
IMethodProperty = interface;
```

C++

```
__interface IMethodProperty;
```

File

DesignIntf

Description

This is class DesignIntf.IMethodProperty.

60.2.21 IProperty Interface

IProperty is the interface that the Object Inspector uses to communicate with property editors.

Class Hierarchy

DesignIntf.IProperty

Pascal

```
IProperty = interface;
```

C++

```
__interface IProperty;
```

File

DesignIntf

Description

When you create your own property editors, they must meet two criteria:

They must be descendants of TBasePropertyEditor (see page 11112).

They must implement the IProperty interface.

The Object Inspector uses the methods on the IProperty interface to interact with property editors.

Typically, property editors are derived from TPropertyEditor, which supplies a default implementation for IProperty.

Note: C++ method declarations that use IPropertyServer use the _di_IPropertyServer type instead. This type is a DelphiInterface wrapper around the IProperty interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IProperty >  _di_IProperty;
```

See Also

RegisterPropertyEditor (see page 11229)

TBasePropertyEditor (see page 11112)

TPropertyEditor (see page 11019)

60.2.21.1 IProperty Members

The following tables list the members exposed by IProperty.

Public Methods

	Name	Description
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
	AllEqual (see page 11199)	Indicates whether all selected components have the same value for the property.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	GetAttributes (see page 11200)	Describes the property so the Object Inspector provides the appropriate controls.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetName (see page 11202)	Returns the name of the property.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
	GetValue (see page 11203)	Returns the property value formatted as a string.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	Revert (see page 11204)	Restores all object properties to their original values.
	SetValue (see page 11204)	Responds when the user changes the value of the property.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

60.2.21.2 IProperty Methods

The methods of the IProperty class are listed here.

Public Methods

	Name	Description
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
	AllEqual (see page 11199)	Indicates whether all selected components have the same value for the property.

	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	GetAttributes (see page 11200)	Describes the property so the Object Inspector provides the appropriate controls.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetName (see page 11202)	Returns the name of the property.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
	GetValue (see page 11203)	Returns the property value formatted as a string.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	Revert (see page 11204)	Restores all object properties to their original values.
	SetValue (see page 11204)	Responds when the user changes the value of the property.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

60.2.21.2.1 IProperty.Activate Method

Responds when the property is selected in the Object Inspector.

Pascal

```
procedure Activate;
```

C++

```
__fastcall Activate();
```

Description

The Object Inspector calls Activate when the property is selected. This method lets a property editor determine attributes of the property dynamically when it is displayed in the Object Inspector.

Before Activate is called, the GetAttributes (see page 11200) method does not need to provide accurate information about any attributes other than paSubProperties, paVCL, and paMultiSelect. Any other attributes can be computed in the Activate method and stored so that subsequent calls to GetAttributes (see page 11200) can provide accurate information.

See Also

GetAttributes (see page 11200)

60.2.21.2.2 IProperty.AllEqual Method

Indicates whether all selected components have the same value for the property.

Pascal

```
function AllEqual: Boolean;
```

C++

```
__fastcall Boolean AllEqual();
```

Description

The Object Inspector calls AllEqual when there is more than one component selected to determine whether a value should appear for this property. If this method returns true, the Object Inspector displays a value for the property. If this method returns false, the line for the property in the Object Inspector remains blank.

Note: AllEqual is called only when the value returned by GetAttributes (see page 11200) includes paMultiSelect.

See Also

GetAttributes (see page 11200)

GetValue (see page 11203)

60.2.21.2.3 IProperty.AutoFill Method

Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.

Pascal

```
function AutoFill: Boolean;
```

C++

```
__fastcall Boolean AutoFill();
```

Description

If GetAttributes (see page 11200) returns paValueList, the Object Inspector calls AutoFill to verify that automatic typing completion should be permitted. If AutoFill returns true, the Object Inspector uses the values provided by GetValues (see page 11203) to fill in partially typed entries. For example, if the legal values for a property are prSmall and prLarge, the user can type "prS" and the Object Inspector automatically fills in "prSmall."

See Also

GetAttributes (see page 11200)

GetValues (see page 11203)

60.2.21.2.4 IProperty.Edit Method

Responds when the user double-clicks the property in the Object Inspector.

Pascal

```
procedure Edit; overload;
```

C++

```
__fastcall Edit();
```

Description

The Object Inspector calls Edit when the ellipsis button (...) is pressed or the property is double-clicked. Property editors implement this method to respond. For example, the Edit method can bring up a dialog for editing the property or it can change the value to some default.

See Also

GetValues (see page 11203)

GetValue (see page 11203)

60.2.21.2.5 IProperty.GetAttributes Method

Describes the property so the Object Inspector provides the appropriate controls.

Pascal

```
function GetAttributes: TPropertyAttributes;
```

C++

```
__fastcall TPropertyAttributes GetAttributes();
```

Description

The Object Inspector calls GetAttributes to determine what controls to display for the property and what property editor methods to call. GetAttributes returns a set of flags that provide this information.

Before the Object Inspector calls the Activate (see page 11199) method, GetAttributes does not need to provide accurate information about any attributes other than paSubProperties, paVCL, and paMultiSelect.

See Also

Activate (see page 11199)

Edit (see page 11200)

GetProperties (see page 11202)

GetValues (see page 11203)

SetValue (see page 11204)

60.2.21.2.6 IProperty.GetEditLimit Method

Returns the maximum length, in characters, of the value.

Pascal

```
function GetEditLimit: Integer;
```

C++

```
__fastcall int GetEditLimit();
```

Description

The Object Inspector calls GetEditLimit to determine how many characters the user can type when entering a value for the property.

60.2.21.2.7 IProperty.GetEditValue Method

Retrieves a string to represent the property's value, or, if the value is not available, an error message.

Pascal

```
function GetEditValue(out Value: string): Boolean;
```

C++

```
__fastcall Boolean GetEditValue(AnsiString Value);
```

Description

The Object Inspector calls GetEditValue to obtain the string it displays as the property value.

GetValue (see page 11203) formats the string displayed in the Object Inspector to represent the property value and returns it as the Value parameter. Unlike the GetValue (see page 11203) method, GetEditValue does not raise an exception when the property value is unavailable. Instead, it assigns an error message to Value and returns false.

GetValue (see page 11203) returns true if Value returns a string representation of the property value.

See Also

GetValue (see page 11203)

60.2.21.2.8 IProperty.GetName Method

Returns the name of the property.

Pascal

```
function GetName: string;
```

C++

```
__fastcall AnsiString GetName();
```

Description

The Object Inspector calls GetName to obtain the property name that it displays in the left column.

60.2.21.2.9 IProperty.GetProperties Method

Provides the subproperties of the property to a callback procedure.

Pascal

```
procedure GetProperties(Proc: TGetPropProc);
```

C++

```
__fastcall GetProperties(TGetPropProc Proc);
```

Description

The Object Inspector calls GetProperties when the property is expanded to show subproperties. Property editors that return paSubProperties from the GetAttributes (see page 11200) method should implement GetProperties to create an appropriate property editor and pass its IProperty (see page 11197) interface to the Proc parameter for every subproperty (or nested property) of the property.

See Also

GetAttributes (see page 11200)

60.2.21.2.10 IProperty.GetPropInfo Method

Returns the property information record for the object being edited.

Pascal

```
function GetPropInfo: PPropInfo;
```

C++

```
__fastcall PPropInfo GetPropInfo();
```

Description

Call GetPropInfo to access the property information record that describes the property being edited, including its access functions. If more than one object is being edited, GetPropInfo returns the property information record for a representative component.

See Also

GetPropType (see page 11203)

60.2.21.2.11 IProperty.GetPropType Method

Returns the type information record for the object being edited.

Pascal

```
function GetPropType: PTypeInfo;
```

C++

```
__fastcall PTypeInfo GetPropType();
```

Description

Call GetPropType to obtain access to the type information record that describes the type of the property being edited. If more than one object is being edited, GetPropType returns the type information record for a representative component.

See Also

GetPropInfo (see page 11202)

60.2.21.2.12 IProperty.GetValue Method

Returns the property value formatted as a string.

Pascal

```
function GetValue: string;
```

C++

```
__fastcall AnsiString GetValue();
```

Description

GetValue formats the value of the property, formatted as a string. If the value is not available, GetValue raises an exception.

See Also

SetValue (see page 11204)

GetEditValue (see page 11201)

60.2.21.2.13 IProperty.GetValues Method

Provides the enumerated values of the property to a callback procedure.

Pascal

```
procedure GetValues(Proc: TGetStrProc);
```

C++

```
__fastcall GetValues(TGetStrProc Proc);
```

Description

The Object Inspector calls GetValues when the drop-down button is clicked to show enumerated values. Property editors that return paValueList from the GetAttributes (see page 11200) method implement GetValues to call the Proc parameter for every string that can represent a valid enumerated value. Even if the GetAttributes (see page 11200) method does not return paValueList, property editors for enumerated types should still implement GetValues to enumerate all possible values.

See Also

GetAttributes (see page 11200)

GetValue (see page 11203)

60.2.21.2.14 IProperty.HasInstance Method

Indicates whether a specified object is currently being edited by the property editor.

Pascal

```
function HasInstance(Instance: TPersistent): Boolean;
```

C++

```
__fastcall Boolean HasInstance(TPersistent * Instance);
```

Description

HasInstance returns true if the Instance parameter references one of the objects that the property editor is currently editing, false otherwise.

60.2.21.2.15 IProperty.Revert Method

Restores all object properties to their original values.

Pascal

```
procedure Revert;
```

C++

```
__fastcall Revert();
```

Description

Call Revert to restore the original property values that each of the edited objects had before the user began making modification with the Object Inspector. Revert may not work if GetAttributes (see page 11200) returns paAutoUpdate.

See Also

GetAttributes (see page 11200)

60.2.21.2.16 IProperty.SetValue Method

Responds when the user changes the value of the property.

Pascal

```
procedure SetValue(const Value: string);
```

C++

```
__fastcall SetValue(const AnsiString Value);
```

Description

The Object Inspector calls SetValue when the user has finished changing the property value. SetValue applies the value specified by the Value parameter to all components that it is currently editing. If the value can't be set, SetValue raises an exception that describes the problem.

Note: If the property is changed only through a dialog brought up by the Edit (see page 11200) method, the Object Inspector does not call SetValue. Rather, the Edit (see page 11200) method should handle the setting of the value on all components.

See Also

Edit (see page 11200)

60.2.21.2.17 IProperty.ValueAvailable Method

Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Pascal

```
function ValueAvailable: Boolean;
```

C++

```
__fastcall Boolean ValueAvailable();
```

Description

The Object Inspector calls ValueAvailable to determine which components can be edited for a particular property or which properties can be displayed for a particular component.

See Also

ComponentStyle (see page 2805)

60.2.22 IProperty70 Interface

Class Hierarchy

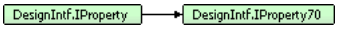

Pascal

```
IProperty70 = interface(IProperty);
```

C++

```
__interface IProperty70 : public IProperty;
```

File

DesignIntf

Description

This is class DesignIntf.IProperty70.

60.2.22.1 IProperty70 Members

The following tables list the members exposed by IProperty70.

Public Methods

	Name	Description
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
	AllEqual (see page 11199)	Indicates whether all selected components have the same value for the property.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	Edit (see page 11200)	Responds when the user double-clicks the property in the Object Inspector.
	GetAttributes (see page 11200)	Describes the property so the Object Inspector provides the appropriate controls.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.

	GetName (see page 11202)	Returns the name of the property.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
	GetValue (see page 11203)	Returns the property value formatted as a string.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	Revert (see page 11204)	Restores all object properties to their original values.
	SetValue (see page 11204)	Responds when the user changes the value of the property.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

Public Properties

	Name	Description
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.

60.2.22.2 IProperty70 Methods

The methods of the IProperty70 class are listed here.

Public Methods

	Name	Description
	GetIsDefault (see page 11206)	This is GetIsDefault, a member of class IProperty70.

60.2.22.2.1 IProperty70.GetIsDefault Method

Pascal

```
function GetIsDefault: Boolean;
```

C++

```
__fastcall Boolean GetIsDefault();
```

Description

This is GetIsDefault, a member of class IProperty70.

60.2.22.3 IProperty70 Properties

The properties of the IProperty70 class are listed here.

Public Properties

	Name	Description
R	IsDefault (see page 11206)	This is IsDefault, a member of class IProperty70.

60.2.22.3.1 IProperty70.IsDefault Property

Pascal

```
property IsDefault: Boolean;
```

C++

```
__property Boolean IsDefault;
```

Description

This is IsDefault, a member of class IProperty70.

60.2.23 IProperty80 Interface

Class Hierarchy

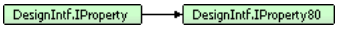

Pascal

```
IProperty80 = interface(IProperty);
```

C++

```
__interface IProperty80 : public IProperty;
```

File

DesignIntf

Description

This is class DesignIntf.IProperty80.

60.2.23.1 IProperty80 Members

The following tables list the members exposed by IProperty80.

Public Methods

	Name	Description
	Activate (see page 11199)	Responds when the property is selected in the Object Inspector.
	AllEqual (see page 11199)	Indicates whether all selected components have the same value for the property.
	AutoFill (see page 11200)	Returns true to indicate that values can be typed incrementally—and automatically completed—in the Object Inspector.
	Edit (see page 11208)	This is Edit, a member of class IProperty80.
	GetAttributes (see page 11200)	Describes the property so the Object Inspector provides the appropriate controls.
	GetEditLimit (see page 11201)	Returns the maximum length, in characters, of the value.
	GetEditValue (see page 11201)	Retrieves a string to represent the property's value, or, if the value is not available, an error message.
	GetName (see page 11202)	Returns the name of the property.
	GetProperties (see page 11202)	Provides the subproperties of the property to a callback procedure.
	GetPropInfo (see page 11202)	Returns the property information record for the object being edited.
	GetPropType (see page 11203)	Returns the type information record for the object being edited.
	GetValue (see page 11203)	Returns the property value formatted as a string.
	GetValues (see page 11203)	Provides the enumerated values of the property to a callback procedure.
	HasInstance (see page 11204)	Indicates whether a specified object is currently being edited by the property editor.
	Revert (see page 11204)	Restores all object properties to their original values.
	SetValue (see page 11204)	Responds when the user changes the value of the property.
	ValueAvailable (see page 11205)	Indicates whether all components being edited have a value for the property if their ComponentStyle includes csCheckPropAvail.

60.2.23.2 IProperty80 Methods

The methods of the IProperty80 class are listed here.

Public Methods

	Name	Description
	Edit (see page 11208)	This is Edit, a member of class IProperty80.

60.2.23.2.1 IProperty80.Edit Method

Pascal

```
procedure Edit(const Host: IPropertyHost; DblClick: Boolean);
```

C++

```
__fastcall Edit(const IPropertyHost Host, Boolean DblClick);
```

Description

This is Edit, a member of class IProperty80.

60.2.24 IPropertyControl Interface

Class Hierarchy

DesignIntf.IPropertyControl

Pascal

```
IPropertyControl = interface;
```

C++

```
__interface IPropertyControl;
```

File

DesignIntf

Description

This is class DesignIntf.IPropertyControl.

60.2.24.1 IPropertyControl Members

The following tables list the members exposed by IPropertyControl.

Public Methods

	Name	Description
	ClearCache (see page 11209)	This is ClearCache, a member of class IPropertyControl.

60.2.24.2 IPropertyControl Methods

The methods of the IPropertyControl class are listed here.

Public Methods

	Name	Description
	ClearCache (see page 11209)	This is ClearCache, a member of class IPropertyControl.

60.2.24.2.1 IPropertyControl.ClearCache Method

Pascal

```
procedure ClearCache;
```

C++

```
__fastcall ClearCache();
```

Description

This is ClearCache, a member of class IPropertyControl.

60.2.25 IPropertyDescription Interface

Class Hierarchy

DesignIntf.IPropertyDescription

Pascal

```
IPropertyDescription = interface;
```

C++

```
__interface IPropertyDescription;
```

File

DesignIntf

Description

This is class DesignIntf.IPropertyDescription.

60.2.25.1 IPropertyDescription Members

The following tables list the members exposed by IPropertyDescription.

Public Methods

	Name	Description
	GetDescription (see page 11210)	This is GetDescription, a member of class IPropertyDescription.

60.2.25.2 IPropertyDescription Methods

The methods of the IPropertyDescription class are listed here.

Public Methods

	Name	Description
	GetDescription (see page 11210)	This is GetDescription, a member of class IPropertyDescription.

60.2.25.2.1 IPropertyDescription.GetDescription Method

Pascal

```
function GetDescription: string;
```

C++

```
__fastcall AnsiString GetDescription();
```

Description

This is GetDescription, a member of class IPropertyDescription.

60.2.26 IPropertyHost Interface

Class Hierarchy

DesignIntf.IPropertyHost

Pascal

```
IPropertyHost = interface;
```

C++

```
__interface IPropertyHost;
```

File

DesignIntf

Description

This is class DesignIntf.IPropertyHost.

60.2.26.1 IPropertyHost Members

The following tables list the members exposed by IPropertyHost.

Public Methods

	Name	Description
	CloseDropDown (see page 11211)	This is CloseDropDown, a member of class IPropertyHost.
	DropDownControl (see page 11211)	This is DropDownControl, a member of class IPropertyHost.

60.2.26.2 IPropertyHost Methods

The methods of the IPropertyHost class are listed here.

Public Methods

	Name	Description
	CloseDropDown (see page 11211)	This is CloseDropDown, a member of class IPropertyHost.
	DropDownControl (see page 11211)	This is DropDownControl, a member of class IPropertyHost.

60.2.26.2.1 IPropertyHost.CloseDropDown Method

Pascal

```
procedure CloseDropDown;
```

C++

```
__fastcall CloseDropDown();
```

Description

This is CloseDropDown, a member of class IPropertyHost.

60.2.26.2.2 IPropertyHost.DropDownControl Method

Pascal

```
procedure DropDownControl(Control: TPersistent);
```

C++

```
__fastcall DropDownControl(TPersistent * Control);
```

Description

This is DropDownControl, a member of class IPropertyHost.

60.2.27 IPropertyKind Interface

Class Hierarchy

DesignIntf.IPropertyKind

Pascal

```
IPropertyKind = interface;
```

C++

```
__interface IPropertyKind;
```

File

DesignIntf

Description

This is class DesignIntf.IPropertyKind.

60.2.27.1 IPropertyKind Members

The following tables list the members exposed by IPropertyKind.

Public Methods

	Name	Description
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.

Public Properties

	Name	Description
	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.

60.2.27.2 IPropertyKind Methods

The methods of the IPropertyKind class are listed here.

Public Methods

	Name	Description
	GetKind (see page 11212)	This is GetKind, a member of class IPropertyKind.

60.2.27.2.1 IPropertyKind.GetKind Method

Pascal

```
function GetKind: TTypeKind;
```

C++

```
__fastcall TTypeKind GetKind();
```

Description

This is GetKind, a member of class IPropertyKind.

60.2.27.3 IPropertyKind Properties

The properties of the IPropertyKind class are listed here.

Public Properties

	Name	Description
	Kind (see page 11212)	This is Kind, a member of class IPropertyKind.

60.2.27.3.1 IPropertyKind.Kind Property

Pascal

```
property Kind: TTypeKind;
```

C++

```
__property TTypeKind Kind;
```

Description

This is Kind, a member of class IPropertyKind.

60.2.28 IReferenceProperty Interface

Class Hierarchy

DesignIntf.IReferenceProperty

Pascal

```
IReferenceProperty = interface;
```

C++

```
__interface IReferenceProperty;
```

File

DesignIntf

Description

This is class DesignIntf.IReferenceProperty.

60.2.28.1 IReferenceProperty Members

The following tables list the members exposed by IReferenceProperty.

Public Methods

	Name	Description
	GetComponentReference (see page 11213)	This is GetComponentReference, a member of class IReferenceProperty.

60.2.28.2 IReferenceProperty Methods

The methods of the IReferenceProperty class are listed here.

Public Methods

	Name	Description
	GetComponentReference (see page 11213)	This is GetComponentReference, a member of class IReferenceProperty.

60.2.28.2.1 IReferenceProperty.GetComponentReference Method

Pascal

```
function GetComponentReference: TComponent;
```

C++

```
__fastcall TComponent * GetComponentReference();
```

Description

This is GetComponentReference, a member of class IReferenceProperty.

60.2.29 ISelectionEditor Interface

Class Hierarchy

DesignIntf.ISelectionEditor

Pascal

```
ISelectionEditor = interface;
```

C++

```
__interface ISelectionEditor;
```

File

DesignIntf

Description

This is class DesignIntf.ISelectionEditor.

60.2.29.1 ISelectionEditor Members

The following tables list the members exposed by ISelectionEditor.

Public Methods

	Name	Description
	ExecuteVerb (see page 11214)	This is ExecuteVerb, a member of class ISelectionEditor.
	GetVerb (see page 11214)	This is GetVerb, a member of class ISelectionEditor.
	GetVerbCount (see page 11215)	This is GetVerbCount, a member of class ISelectionEditor.
	PrepareItem (see page 11215)	This is PrepareItem, a member of class ISelectionEditor.
	RequiresUnits (see page 11215)	This is RequiresUnits, a member of class ISelectionEditor.

60.2.29.2 ISelectionEditor Methods

The methods of the ISelectionEditor class are listed here.

Public Methods

	Name	Description
	ExecuteVerb (see page 11214)	This is ExecuteVerb, a member of class ISelectionEditor.
	GetVerb (see page 11214)	This is GetVerb, a member of class ISelectionEditor.
	GetVerbCount (see page 11215)	This is GetVerbCount, a member of class ISelectionEditor.
	PrepareItem (see page 11215)	This is PrepareItem, a member of class ISelectionEditor.
	RequiresUnits (see page 11215)	This is RequiresUnits, a member of class ISelectionEditor.

60.2.29.2.1 ISelectionEditor.ExecuteVerb Method

Pascal

```
procedure ExecuteVerb(Index: Integer; const List: IDesignerSelections);
```

C++

```
__fastcall ExecuteVerb(int Index, const IDesignerSelections List);
```

Description

This is ExecuteVerb, a member of class ISelectionEditor.

60.2.29.2.2 ISelectionEditor.GetVerb Method

Pascal

```
function GetVerb(Index: Integer): string;
```

C++

```
__fastcall AnsiString GetVerb(int Index);
```

Description

This is GetVerb, a member of class ISelectionEditor.

60.2.29.2.3 ISelectionEditor.GetVerbCount Method

Pascal

```
function GetVerbCount: Integer;
```

C++

```
__fastcall int GetVerbCount();
```

Description

This is GetVerbCount, a member of class ISelectionEditor.

60.2.29.2.4 ISelectionEditor.PrepareItem Method

Pascal

```
procedure PrepareItem(Index: Integer; const AItem: IMenuItem);
```

C++

```
__fastcall PrepareItem(int Index, const IMenuItem AItem);
```

Description

This is PrepareItem, a member of class ISelectionEditor.

60.2.29.2.5 ISelectionEditor.RequiresUnits Method

Pascal

```
procedure RequiresUnits(Proc: TGetStrProc);
```

C++

```
__fastcall RequiresUnits(TGetStrProc Proc);
```

Description

This is RequiresUnits, a member of class ISelectionEditor.

60.2.30 ISelectionEditorList Interface

Class Hierarchy

DesignIntf.ISelectionEditorList

Pascal

```
ISelectionEditorList = interface;
```

C++

```
__interface ISelectionEditorList;
```

File

DesignIntf

Description

This is class DesignIntf.ISelectionEditorList.

60.2.30.1 ISelectionEditorList Members

The following tables list the members exposed by ISelectionEditorList.

Public Methods

	Name	Description
	Get (see page 11216)	This is Get, a member of class ISelectionEditorList.
	GetCount (see page 11216)	This is GetCount, a member of class ISelectionEditorList.

Public Properties

	Name	Description
R	Count (see page 11217)	This is Count, a member of class ISelectionEditorList.
R	Items (see page 11217)	This is Items, a member of class ISelectionEditorList.

60.2.30.2 ISelectionEditorList Methods

The methods of the ISelectionEditorList class are listed here.

Public Methods

	Name	Description
	Get (see page 11216)	This is Get, a member of class ISelectionEditorList.
	GetCount (see page 11216)	This is GetCount, a member of class ISelectionEditorList.

60.2.30.2.1 ISelectionEditorList.Get Method

Pascal

```
function Get(Index: Integer): ISelectionEditor;
```

C++

```
__fastcall ISelectionEditor Get(int Index);
```

Description

This is Get, a member of class ISelectionEditorList.

60.2.30.2.2 ISelectionEditorList.GetCount Method

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

This is GetCount, a member of class ISelectionEditorList.

60.2.30.3 ISelectionEditorList Properties

The properties of the ISelectionEditorList class are listed here.

Public Properties

	Name	Description
R	Count (see page 11217)	This is Count, a member of class ISelectionEditorList.
R	Items (see page 11217)	This is Items, a member of class ISelectionEditorList.

60.2.30.3.1 ISelectionEditorList.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class ISelectionEditorList.

60.2.30.3.2 ISelectionEditorList.Items Property

Pascal

```
property Items [Index: Integer]: ISelectionEditor;
```

C++

```
__property ISelectionEditor Items[int Index];
```

Description

This is Items, a member of class ISelectionEditorList.

60.2.31 ISelectionPropertyFilter Interface

Class Hierarchy

DesignIntf.ISelectionPropertyFilter

Pascal

```
ISelectionPropertyFilter = interface;
```

C++

```
__interface ISelectionPropertyFilter;
```

File

DesignIntf

Description

This is class DesignIntf.ISelectionPropertyFilter.

60.2.31.1 ISelectionPropertyFilter Members

The following tables list the members exposed by ISelectionPropertyFilter.

Public Methods

	Name	Description
	FilterProperties (see page 11218)	This is FilterProperties, a member of class ISelectionPropertyFilter.

60.2.31.2 ISelectionPropertyFilter Methods

The methods of the ISelectionPropertyFilter class are listed here.

Public Methods

	Name	Description
	FilterProperties (see page 11218)	This is FilterProperties, a member of class ISelectionPropertyFilter.

60.2.31.2.1 ISelectionPropertyFilter.FilterProperties Method

Pascal

```
procedure FilterProperties(const ASelection: IDesignerSelections; const ASelectionProperties:
IInterfaceList);
```

C++

```
__fastcall FilterProperties(const IDesignerSelections ASelection, const IInterfaceList
ASelectionProperties);
```

Description

This is FilterProperties, a member of class ISelectionPropertyFilter.

60.2.32 IShowReferenceProperty Interface

Class Hierarchy

DesignIntf.IShowReferenceProperty

Pascal

```
IShowReferenceProperty = interface;
```

C++

```
__interface IShowReferenceProperty;
```

File

DesignIntf

Description

This is class DesignIntf.IShowReferenceProperty.

60.2.32.1 IShowReferenceProperty Members

The following tables list the members exposed by IShowReferenceProperty.

Public Methods

	Name	Description
	ShowReferenceProperty (see page 11219)	This is ShowReferenceProperty, a member of class IShowReferenceProperty.

60.2.32.2 IShowReferenceProperty Methods

The methods of the IShowReferenceProperty class are listed here.

Public Methods

	Name	Description
	ShowReferenceProperty (see page 11219)	This is ShowReferenceProperty, a member of class IShowReferenceProperty.

60.2.32.2.1 IShowReferenceProperty.ShowReferenceProperty Method

Pascal

```
function ShowReferenceProperty: Boolean;
```

C++

```
__fastcall Boolean ShowReferenceProperty();
```

Description

This is ShowReferenceProperty, a member of class IShowReferenceProperty.

60.2.33 IWideProperty Interface

Class Hierarchy

DesignIntf.IWideProperty

Pascal

```
IWideProperty = interface;
```

C++

```
__interface IWideProperty;
```

File

DesignIntf

Description

This is class DesignIntf.IWideProperty.

60.2.33.1 IWideProperty Members

The following tables list the members exposed by IWideProperty.

Public Methods

	Name	Description
	GetEditValue (see page 11220)	This is GetEditValue, a member of class IWideProperty.
	GetValue (see page 11220)	This is GetValue, a member of class IWideProperty.
	SetValue (see page 11220)	This is SetValue, a member of class IWideProperty.

60.2.33.2 IWideProperty Methods

The methods of the IWideProperty class are listed here.

Public Methods

	Name	Description
	GetEditValue (see page 11220)	This is GetEditValue, a member of class IWideProperty.
	GetValue (see page 11220)	This is GetValue, a member of class IWideProperty.
	SetValue (see page 11220)	This is SetValue, a member of class IWideProperty.

60.2.33.2.1 IWideProperty.GetEditValue Method

Pascal

```
function GetEditValue(out Value: widestring): Boolean;
```

C++

```
__fastcall Boolean GetEditValue(BSTR Value);
```

Description

This is GetEditValue, a member of class IWideProperty.

60.2.33.2.2 IWideProperty.GetValue Method

Pascal

```
function GetValue: WideString;
```

C++

```
__fastcall BSTR GetValue();
```

Description

This is GetValue, a member of class IWideProperty.

60.2.33.2.3 IWideProperty.SetValue Method

Pascal

```
procedure SetValue(const Value: WideString);
```

C++

```
__fastcall SetValue(const BSTR Value);
```

Description

This is SetValue, a member of class IWideProperty.

60.2.34 IWideProperty10 Interface

Class Hierarchy

DesignIntf.IWideProperty10

Pascal

```
IWideProperty10 = interface;
```

C++

```
__interface IWideProperty10;
```

File

DesignIntf

Description

This is class DesignIntf.IWideProperty10.

60.2.34.1 IWideProperty10 Members

The following tables list the members exposed by IWideProperty10.

Public Methods

	Name	Description
	GetName (see page 11221)	This is GetName, a member of class IWideProperty10.
	GetValues (see page 11222)	This is GetValues, a member of class IWideProperty10.

60.2.34.2 IWideProperty10 Methods

The methods of the IWideProperty10 class are listed here.

Public Methods

	Name	Description
	GetName (see page 11221)	This is GetName, a member of class IWideProperty10.
	GetValues (see page 11222)	This is GetValues, a member of class IWideProperty10.

60.2.34.2.1 IWideProperty10.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IWideProperty10.

60.2.34.2.2 IWideProperty10.GetValues Method

Pascal

```
procedure GetValues(Proc: TGetWideStrProc);
```

C++

```
__fastcall GetValues(TGetWideStrProc Proc);
```

Description

This is GetValues, a member of class IWideProperty10.

60.2.35 IWidePropertyDescription Interface

Class Hierarchy

DesignIntf.IWidePropertyDescription

Pascal

```
IWidePropertyDescription = interface;
```

C++

```
__interface IWidePropertyDescription;
```

File

DesignIntf

Description

This is class DesignIntf.IWidePropertyDescription.

60.2.35.1 IWidePropertyDescription Members

The following tables list the members exposed by IWidePropertyDescription.

Public Methods

	Name	Description
	GetDescription (see page 11223)	This is GetDescription, a member of class IWidePropertyDescription.

60.2.35.2 IWidePropertyDescription Methods

The methods of the IWidePropertyDescription class are listed here.

Public Methods

	Name	Description
	GetDescription (see page 11223)	This is GetDescription, a member of class IWidePropertyDescription.

60.2.35.2.1 IWidePropertyDescription.GetDescription Method

Pascal

```
function GetDescription: WideString;
```

C++

```
__fastcall BSTR GetDescription();
```

Description

This is GetDescription, a member of class IWidePropertyDescription.

60.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CreateSelectionList (see page 11224)	This is function DesignIntf.CreateSelectionList.
	EnableDemandLoadReport (see page 11225)	This is function DesignIntf.EnableDemandLoadReport.
	ForceDemandLoadState (see page 11225)	This is function DesignIntf.ForceDemandLoadState.
	PersistentToDesignObject (see page 11225)	This is function DesignIntf.PersistentToDesignObject.
	RegisterComponentEditor (see page 11226)	This is function DesignIntf.RegisterComponentEditor.
	RegisterComponentGuidelines (see page 11226)	This is function DesignIntf.RegisterComponentGuidelines.
	RegisterCustomModule (see page 11226)	This is function DesignIntf.RegisterCustomModule.
	RegisterDesignDragObject (see page 11227)	This is function DesignIntf.RegisterDesignDragObject.
	RegisterDesignNotification (see page 11227)	This is function DesignIntf.RegisterDesignNotification.
	RegisterDragTarget (see page 11227)	This is function DesignIntf.RegisterDragTarget.
	RegisterIDropTarget (see page 11228)	This is function DesignIntf.RegisterIDropTarget.
	RegisterPropertiesInCategory (see page 11228)	This is function DesignIntf.RegisterPropertiesInCategory.
	RegisterPropertiesInCategory (see page 11228)	This is function DesignIntf.RegisterPropertiesInCategory.
	RegisterPropertiesInCategory (see page 11229)	This is function DesignIntf.RegisterPropertiesInCategory.
	RegisterPropertyEditor (see page 11229)	This is function DesignIntf.RegisterPropertyEditor.
	RegisterPropertyInCategory (see page 11229)	This is function DesignIntf.RegisterPropertyInCategory.
	RegisterPropertyInCategory (see page 11230)	This is function DesignIntf.RegisterPropertyInCategory.
	RegisterPropertyInCategory (see page 11230)	This is function DesignIntf.RegisterPropertyInCategory.
	RegisterPropertyInCategory (see page 11230)	This is function DesignIntf.RegisterPropertyInCategory.
	RegisterPropertyMapper (see page 11231)	This is function DesignIntf.RegisterPropertyMapper.
	RegisterSelectionEditor (see page 11231)	This is function DesignIntf.RegisterSelectionEditor.
	SetPropertyEditorGroup (see page 11231)	This is function DesignIntf.SetPropertyEditorGroup.
	UnlistPublishedProperty (see page 11232)	This is function DesignIntf.UnlistPublishedProperty.
	UnregisterDesignNotification (see page 11232)	This is function DesignIntf.UnregisterDesignNotification.

60.3.1 DesignIntf.CreateSelectionList Function

Pascal

```
function CreateSelectionList: IDesignerSelections;
```

C++

```
IDesignerSelections CreateSelectionList();
```

File

DesignIntf

Description

This is function DesignIntf.CreateSelectionList.

60.3.2 DesignIntf.EnableDemandLoadReport Function

Pascal

```
procedure EnableDemandLoadReport(Detailed: Boolean);
```

C++

```
EnableDemandLoadReport(Boolean Detailed);
```

File

DesignIntf

Description

This is function DesignIntf.EnableDemandLoadReport.

60.3.3 DesignIntf.ForceDemandLoadState Function

Pascal

```
procedure ForceDemandLoadState(DemandLoadState: TDemandLoadState);
```

C++

```
ForceDemandLoadState(TDemandLoadState DemandLoadState);
```

File

DesignIntf

Description

This is function DesignIntf.ForceDemandLoadState.

60.3.4 DesignIntf.PersistentToDesignObject Function

Pascal

```
function PersistentToDesignObject(APersistent: TPersistent): IDesignObject;
```

C++

```
IDesignObject PersistentToDesignObject(TPersistent * APersistent);
```

File

DesignIntf

Description

This is function DesignIntf.PersistentToDesignObject.

60.3.5 DesignIntf.RegisterComponentEditor Function

Pascal

```
procedure RegisterComponentEditor(ComponentClass: TComponentClass; ComponentEditor:
TComponentEditorClass);
```

C++

```
RegisterComponentEditor(TComponentClass ComponentClass, TComponentEditorClass ComponentEditor);
```

File

DesignIntf

Description

This is function DesignIntf.RegisterComponentEditor.

60.3.6 DesignIntf.RegisterComponentGuidelines Function

Pascal

```
procedure RegisterComponentGuidelines(AComponentClass: TComponentClass;
AComponentGuidelinesClass: TComponentGuidelinesClass);
```

C++

```
RegisterComponentGuidelines(TComponentClass AComponentClass, TComponentGuidelinesClass
AComponentGuidelinesClass);
```

File

DesignIntf

Description

This is function DesignIntf.RegisterComponentGuidelines.

60.3.7 DesignIntf.RegisterCustomModule Function

Pascal

```
procedure RegisterCustomModule(ComponentBaseClass: TComponentClass; CustomModuleClass:
TCustomModuleClass);
```

C++

```
RegisterCustomModule(TComponentClass ComponentBaseClass, TCustomModuleClass CustomModuleClass);
```

File

DesignIntf

Description

This is function DesignIntf.RegisterCustomModule.

60.3.8 DesignIntf.RegisterDesignDragObject Function

Pascal

```
procedure RegisterDesignDragObject(DragObjectClass: TDragObjectClass);
```

C++

```
RegisterDesignDragObject(TDragObjectClass DragObjectClass);
```

File

DesignIntf

Description

This is function DesignIntf.RegisterDesignDragObject.

60.3.9 DesignIntf.RegisterDesignNotification Function

Pascal

```
procedure RegisterDesignNotification(const DesignNotification: IDesignNotification);
```

C++

```
RegisterDesignNotification(const IDesignNotification DesignNotification);
```

File

DesignIntf

Description

This is function DesignIntf.RegisterDesignNotification.

60.3.10 DesignIntf.RegisterDragTarget Function

Pascal

```
procedure RegisterDragTarget(const SourceName: string; TargetClass: TDragTargetClass);
```

C++

```
RegisterDragTarget(const AnsiString SourceName, TDragTargetClass TargetClass);
```

File

DesignIntf

Description

This is function DesignIntf.RegisterDragTarget.

60.3.11 DesignIntf.RegisterIDropTarget Function

Pascal

```
procedure RegisterIDropTarget(const ADropTarget: IDropTarget);
```

C++

```
RegisterIDropTarget(const IDropTarget ADropTarget);
```

File

DesignIntf

Description

This is function DesignIntf.RegisterIDropTarget.

60.3.12 DesignIntf.RegisterPropertiesInCategory Function

Pascal

```
procedure RegisterPropertiesInCategory(const CategoryName: string; PropertyType: PTypeInfo;
const Filters: array of string); overload;
```

C++

```
RegisterPropertiesInCategory(const AnsiString CategoryName, PTypeInfo PropertyType, const
array of string Filters);
```

File

DesignIntf

Description

This is function DesignIntf.RegisterPropertiesInCategory.

60.3.13 DesignIntf.RegisterPropertiesInCategory Function

Pascal

```
procedure RegisterPropertiesInCategory(const CategoryName: string; ComponentClass: TClass;
const Filters: array of string); overload;
```

C++

```
RegisterPropertiesInCategory(const AnsiString CategoryName, TClass ComponentClass, const array
of string Filters);
```

File

DesignIntf

Description

This is function DesignIntf.RegisterPropertiesInCategory.

60.3.14 DesignIntf.RegisterPropertiesInCategory Function

Pascal

```
procedure RegisterPropertiesInCategory(const CategoryName: string; const Filters: array of
const); overload;
```

C++

```
RegisterPropertiesInCategory(const AnsiString CategoryName, const array of const Filters);
```

File

DesignIntf

Description

This is function DesignIntf.RegisterPropertiesInCategory.

60.3.15 DesignIntf.RegisterPropertyEditor Function

Pascal

```
procedure RegisterPropertyEditor(PropertyType: PTypeInfo; ComponentClass: TClass; const
PropertyName: string; EditorClass: TPropertyEditorClass);
```

C++

```
RegisterPropertyEditor(PTypeInfo PropertyType, TClass ComponentClass, const AnsiString
PropertyName, TPropertyEditorClass EditorClass);
```

File

DesignIntf

Description

This is function DesignIntf.RegisterPropertyEditor.

60.3.16 DesignIntf.RegisterPropertyInCategory Function

Pascal

```
procedure RegisterPropertyInCategory(const CategoryName: string; PropertyType: PTypeInfo);
overload;
```

C++

```
RegisterPropertyInCategory(const AnsiString CategoryName, PTypeInfo PropertyType);
```

File

DesignIntf

Description

This is function DesignIntf.RegisterPropertyInCategory.

60.3.17 DesignIntf.RegisterPropertyInCategory Function

Pascal

```
procedure RegisterPropertyInCategory(const CategoryName: string; PropertyType: PTypeInfo;
const PropertyName: string); overload;
```

C++

```
RegisterPropertyInCategory(const AnsiString CategoryName, PTypeInfo PropertyType, const
AnsiString PropertyName);
```

File

DesignIntf

Description

This is function DesignIntf.RegisterPropertyInCategory.

60.3.18 DesignIntf.RegisterPropertyInCategory Function

Pascal

```
procedure RegisterPropertyInCategory(const CategoryName: string; ComponentClass: TClass; const
PropertyName: string); overload;
```

C++

```
RegisterPropertyInCategory(const AnsiString CategoryName, TClass ComponentClass, const
AnsiString PropertyName);
```

File

DesignIntf

Description

This is function DesignIntf.RegisterPropertyInCategory.

60.3.19 DesignIntf.RegisterPropertyInCategory Function

Pascal

```
procedure RegisterPropertyInCategory(const CategoryName: string; const PropertyName: string);
overload;
```

C++

```
RegisterPropertyInCategory(const AnsiString CategoryName, const AnsiString PropertyName);
```

File

DesignIntf

Description

This is function DesignIntf.RegisterPropertyInCategory.

60.3.20 DesignIntf.RegisterPropertyMapper Function

Pascal

```
procedure RegisterPropertyMapper(Mapper: TPropertyMapperFunc);
```

C++

```
RegisterPropertyMapper(TPropertyMapperFunc Mapper);
```

File

DesignIntf

Description

This is function DesignIntf.RegisterPropertyMapper.

60.3.21 DesignIntf.RegisterSelectionEditor Function

Pascal

```
procedure RegisterSelectionEditor(AClass: TClass; AEditor: TSelectionEditorClass);
```

C++

```
RegisterSelectionEditor(TClass AClass, TSelectionEditorClass AEditor);
```

File

DesignIntf

Description

This is function DesignIntf.RegisterSelectionEditor.

60.3.22 DesignIntf.SetPropertyEditorGroup Function

Pascal

```
procedure SetPropertyEditorGroup(EditorClass: TPropertyEditorClass; GroupClass:
TPersistentClass);
```

C++

```
SetPropertyEditorGroup(TPropertyEditorClass EditorClass, TPersistentClass GroupClass);
```

File

DesignIntf

Description

This is function DesignIntf.SetPropertyEditorGroup.

60.3.23 DesignIntf.UnlistPublishedProperty Function

Pascal

```
procedure UnlistPublishedProperty(ComponentClass: TClass; const PropertyName: string);
```

C++

```
UnlistPublishedProperty(TClass ComponentClass, const AnsiString PropertyName);
```

File

DesignIntf

Description

This is function DesignIntf.UnlistPublishedProperty.

60.3.24 DesignIntf.UnregisterDesignNotification Function

Pascal

```
procedure UnregisterDesignNotification(const DesignNotification: IDesignNotification);
```

C++

```
UnregisterDesignNotification(const IDesignNotification DesignNotification);
```

File

DesignIntf

Description

This is function DesignIntf.UnregisterDesignNotification.

60.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCustomModuleAttribute (see page 11233)	This is record DesignIntf.TCustomModuleAttribute.
	TDemandLoadState (see page 11233)	This is record DesignIntf.TDemandLoadState.
	TDesignerGuideType (see page 11234)	This is record DesignIntf.TDesignerGuideType.
	TEditAction (see page 11234)	This is record DesignIntf.TEditAction.
	TEditStates (see page 11235)	This is record DesignIntf.TEditStates.
	TPropertyAttribute (see page 11236)	This is record DesignIntf.TPropertyAttribute.
	TPropKind (see page 11237)	This is record DesignIntf.TPropKind.

60.4.1 DesignIntf.TCustomModuleAttribute Enumeration

Pascal

```
TCustomModuleAttribute = (
  cmaVirtualSize
);
```

C++

```
enum TCustomModuleAttribute {
  cmaVirtualSize
};
```

File

DesignIntf

Description

This is record DesignIntf.TCustomModuleAttribute.

60.4.2 DesignIntf.TDemandLoadState Enumeration

Pascal

```
TDemandLoadState = (
  dlDefault,
  dlDisable,
  dlEnable
);
```

C++

```
enum TDemandLoadState {
  dlDefault,
  dlDisable,
  dlEnable
};
```

File

DesignIntf

Description

This is record DesignIntf.TDemandLoadState.

60.4.3 DesignIntf.TDesignerGuideType Enumeration

Pascal

```
TDesignerGuideType = (
  gtAlignLeft,
  gtAlignTop,
  gtAlignRight,
  gtAlignBottom,
  gtMarginLeft,
  gtMarginTop,
  gtMarginRight,
  gtMarginBottom,
  gtPaddingLeft,
  gtPaddingTop,
  gtPaddingRight,
  gtPaddingBottom,
  gtBaseline
);
```

C++

```
enum TDesignerGuideType {
  gtAlignLeft,
  gtAlignTop,
  gtAlignRight,
  gtAlignBottom,
  gtMarginLeft,
  gtMarginTop,
  gtMarginRight,
  gtMarginBottom,
  gtPaddingLeft,
  gtPaddingTop,
  gtPaddingRight,
  gtPaddingBottom,
  gtBaseline
};
```

File

DesignIntf

Description

This is record DesignIntf.TDesignerGuideType.

60.4.4 DesignIntf.TEditAction Enumeration

Pascal

```
TEditAction = (
  eaUndo,
```

```
    eaRedo,
    eaCut,
    eaCopy,
    eaPaste,
    eaDelete,
    eaSelectAll,
    eaPrint,
    eaElide
  );
```

C++

```
  enum TEditAction {
    eaUndo,
    eaRedo,
    eaCut,
    eaCopy,
    eaPaste,
    eaDelete,
    eaSelectAll,
    eaPrint,
    eaElide
  };
```

File

DesignIntf

Description

This is record DesignIntf.TEditAction.

60.4.5 DesignIntf.TEditStates Enumeration

Pascal

```
  TEditStates = (
    esCanUndo,
    esCanRedo,
    esCanCut,
    esCanCopy,
    esCanPaste,
    esCanDelete,
    esCanEditOle,
    esCanPrint,
    esCanSelectAll,
    esCanCreateTemplate,
    esCanElide
  );
```

C++

```
  enum TEditStates {
    esCanUndo,
    esCanRedo,
    esCanCut,
    esCanCopy,
    esCanPaste,
    esCanDelete,
    esCanEditOle,
    esCanPrint,
    esCanSelectAll,
    esCanCreateTemplate,
    esCanElide
  };
```

File

DesignIntf

Description

This is record DesignIntf.TEditStates.

60.4.6 DesignIntf.TPropertyAttribute Enumeration

Pascal

```
TPropertyAttribute = (
  paValueList,
  paSubProperties,
  paDialog,
  paMultiSelect,
  paAutoUpdate,
  paSortList,
  paReadOnly,
  paRevertable,
  paFullWidthName,
  paVolatileSubProperties,
  paVCL,
  paNotNestable,
  paDisplayReadOnly,
  paCustomDropDown,
  paValueEditable
);
```

C++

```
enum TPropertyAttribute {
  paValueList,
  paSubProperties,
  paDialog,
  paMultiSelect,
  paAutoUpdate,
  paSortList,
  paReadOnly,
  paRevertable,
  paFullWidthName,
  paVolatileSubProperties,
  paVCL,
  paNotNestable,
  paDisplayReadOnly,
  paCustomDropDown,
  paValueEditable
};
```

File

DesignIntf

Description

This is record DesignIntf.TPropertyAttribute.

60.4.7 DesignIntf.TPropKind Enumeration

Pascal

```
TPropKind = (
  pkProperties,
  pkEvents
);
```

C++

```
enum TPropKind {
  pkProperties,
  pkEvents
};
```

File

DesignIntf

Description

This is record DesignIntf.TPropKind.

60.5 Types

The following table lists types in this documentation.

Types

Name	Description
IDesignEditQuery (see page 11238)	This is type DesignIntf.IDesignEditQuery.
TComponentEditorClass (see page 11239)	This is type DesignIntf.TComponentEditorClass.
TComponentGuidelinesClass (see page 11239)	This is type DesignIntf.TComponentGuidelinesClass.
TCustomModuleAttributes (see page 11239)	This is type DesignIntf.TCustomModuleAttributes.
TCustomModuleClass (see page 11240)	This is type DesignIntf.TCustomModuleClass.
TDragTargetClass (see page 11240)	This is type DesignIntf.TDragTargetClass.
TEditState (see page 11240)	This is type DesignIntf.TEditState.
TEnableDemandLoadReportProc (see page 11241)	This is type DesignIntf.TEnableDemandLoadReportProc.
TForceDemandLoadStateProc (see page 11241)	This is type DesignIntf.TForceDemandLoadStateProc.
TGetDesignerEvent (see page 11241)	This is type DesignIntf.TGetDesignerEvent.
TGetPropProc (see page 11242)	This is type DesignIntf.TGetPropProc.
TGetWideStrProc (see page 11242)	This is type DesignIntf.TGetWideStrProc.
TPropertyAttributes (see page 11242)	This is type DesignIntf.TPropertyAttributes.
TPropertyEditorClass (see page 11243)	This is type DesignIntf.TPropertyEditorClass.
TPropertyEditorFilterFunc (see page 11243)	This is type DesignIntf.TPropertyEditorFilterFunc.
TPropertyMapperFunc (see page 11243)	This is type DesignIntf.TPropertyMapperFunc.
TRegisterComponentEditorProc (see page 11244)	This is type DesignIntf.TRegisterComponentEditorProc.
TRegisterComponentGuidelines (see page 11244)	This is type DesignIntf.TRegisterComponentGuidelines.
TRegisterCustomModuleProc (see page 11244)	This is type DesignIntf.TRegisterCustomModuleProc.
TRegisterDesignDragObject (see page 11245)	This is type DesignIntf.TRegisterDesignDragObject.
TRegisterDesignNotification (see page 11245)	This is type DesignIntf.TRegisterDesignNotification.
TRegisterDragTargetProc (see page 11245)	This is type DesignIntf.TRegisterDragTargetProc.
TRegisterIDropTarget (see page 11246)	This is type DesignIntf.TRegisterIDropTarget.
TRegisterPropertyEditorProc (see page 11246)	This is type DesignIntf.TRegisterPropertyEditorProc.
TRegisterPropertyInCategoryProc (see page 11246)	This is type DesignIntf.TRegisterPropertyInCategoryProc.
TRegisterPropertyMapperProc (see page 11247)	This is type DesignIntf.TRegisterPropertyMapperProc.
TRegisterSelectionEditorProc (see page 11247)	This is type DesignIntf.TRegisterSelectionEditorProc.
TSelectionEditorClass (see page 11247)	This is type DesignIntf.TSelectionEditorClass.
TSetPropertyEditorGroupProc (see page 11248)	This is type DesignIntf.TSetPropertyEditorGroupProc.

60.5.1 DesignIntf.IDesignEditQuery Type

Pascal

```
IDesignEditQuery = IEditHandler;
```

C++

```
IEditHandler IDesignEditQuery;
```

File

DesignIntf

Description

This is type DesignIntf.IDesignEditQuery.

60.5.2 DesignIntf.TComponentEditorClass Type

Pascal

```
TComponentEditorClass = class of TBaseComponentEditor;
```

C++

```
TBaseComponentEditor TComponentEditorClass;
```

File

DesignIntf

Description

This is type DesignIntf.TComponentEditorClass.

60.5.3 DesignIntf.TComponentGuidelinesClass Type

Pascal

```
TComponentGuidelinesClass = class of TBaseComponentGuidelines;
```

C++

```
TBaseComponentGuidelines TComponentGuidelinesClass;
```

File

DesignIntf

Description

This is type DesignIntf.TComponentGuidelinesClass.

60.5.4 DesignIntf.TCustomModuleAttributes Type

Pascal

```
TCustomModuleAttributes = set of TCustomModuleAttribute;
```

C++

```
TCustomModuleAttribute TCustomModuleAttributes;
```

File

DesignIntf

Description

This is type DesignIntf.TCustomModuleAttributes.

60.5.5 DesignIntf.TCustomModuleClass Type

Pascal

```
TCustomModuleClass = class of TBaseCustomModule;
```

C++

```
TBaseCustomModule TCustomModuleClass;
```

File

DesignIntf

Description

This is type DesignIntf.TCustomModuleClass.

60.5.6 DesignIntf.TDragTargetClass Type

Pascal

```
TDragTargetClass = class of TDragTarget;
```

C++

```
TDragTarget TDragTargetClass;
```

File

DesignIntf

Description

This is type DesignIntf.TDragTargetClass.

60.5.7 DesignIntf.TEditState Type

Pascal

```
TEditState = set of TEditStates;
```

C++

```
TEditStates TEditState;
```

File

DesignIntf

Description

This is type DesignIntf.TEditState.

60.5.8 DesignIntf.TEnableDemandLoadReportProc Type

Pascal

```
TEnableDemandLoadReportProc = procedure (Detailed: Boolean);
```

C++

```
(Detailed: Boolean) ( TEnableDemandLoadReportProc)();
```

File

DesignIntf

Description

This is type DesignIntf.TEnableDemandLoadReportProc.

60.5.9 DesignIntf.TForceDemandLoadStateProc Type

Pascal

```
TForceDemandLoadStateProc = procedure (DemandLoadState: TDemandLoadState);
```

C++

```
(DemandLoadState: TDemandLoadState) ( TForceDemandLoadStateProc)();
```

File

DesignIntf

Description

This is type DesignIntf.TForceDemandLoadStateProc.

60.5.10 DesignIntf.TGetDesignerEvent Type

Pascal

```
TGetDesignerEvent = procedure (Sender: TObject; out ADesigner: IDesigner) of object;
```

C++

```
(Sender: TObject; out ADesigner: IDesigner) ( TGetDesignerEvent)();
```

File

DesignIntf

Description

This is type DesignIntf.TGetDesignerEvent.

60.5.11 DesignIntf.TGetPropProc Type

Pascal

```
TGetPropProc = procedure (const Prop: IProperty) of object;
```

C++

```
(const Prop: IProperty) ( TGetPropProc)();
```

File

DesignIntf

Description

This is type DesignIntf.TGetPropProc.

60.5.12 DesignIntf.TGetWideStrProc Type

Pascal

```
TGetWideStrProc = procedure (const S: WideString) of object;
```

C++

```
(const S: WideString) ( TGetWideStrProc)();
```

File

DesignIntf

Description

This is type DesignIntf.TGetWideStrProc.

60.5.13 DesignIntf.TPropertyAttributes Type

Pascal

```
TPropertyAttributes = set of TPropertyAttribute;
```

C++

```
TPropertyAttribute TPropertyAttributes;
```

File

DesignIntf

Description

This is type DesignIntf.TPropertyAttributes.

60.5.14 DesignIntf.TPropertyEditorClass Type

Pascal

```
TPropertyEditorClass = class of TBasePropertyEditor;
```

C++

```
TBasePropertyEditor TPropertyEditorClass;
```

File

DesignIntf

Description

This is type DesignIntf.TPropertyEditorClass.

60.5.15 DesignIntf.TPropertyEditorFilterFunc Type

Pascal

```
TPropertyEditorFilterFunc = function (const ATestEditor: IProperty): Boolean of object;
```

C++

```
(const ATestEditor: IProperty): Boolean ( TPropertyEditorFilterFunc)();
```

File

DesignIntf

Description

This is type DesignIntf.TPropertyEditorFilterFunc.

60.5.16 DesignIntf.TPropertyMapperFunc Type

Pascal

```
TPropertyMapperFunc = function (Obj: TPersistent; PropInfo: PPropInfo): TPropertyEditorClass;
```

C++

```
(Obj: TPersistent; PropInfo: PPropInfo): TPropertyEditorClass ( TPropertyMapperFunc)();
```

File

DesignIntf

Description

This is type DesignIntf.TPropertyMapperFunc.

60.5.17 DesignIntf.TRegisterComponentEditorProc Type

Pascal

```
TRegisterComponentEditorProc = procedure (ComponentClass: TComponentClass; ComponentEditor:
TComponentEditorClass);
```

C++

```
(ComponentClass: TComponentClass; ComponentEditor: TComponentEditorClass) (
TRegisterComponentEditorProc)();
```

File

DesignIntf

Description

This is type DesignIntf.TRegisterComponentEditorProc.

60.5.18 DesignIntf.TRegisterComponentGuidelines Type

Pascal

```
TRegisterComponentGuidelines = procedure (AComponentClass: TComponentClass;
AComponentGuidelineClass: TComponentGuidelinesClass);
```

C++

```
(AComponentClass: TComponentClass; AComponentGuidelineClass: TComponentGuidelinesClass) (
TRegisterComponentGuidelines)();
```

File

DesignIntf

Description

This is type DesignIntf.TRegisterComponentGuidelines.

60.5.19 DesignIntf.TRegisterCustomModuleProc Type

Pascal

```
TRegisterCustomModuleProc = procedure (Group: Integer; ComponentBaseClass: TComponentClass;
CustomModuleClass: TCustomModuleClass);
```

C++

```
(Group: Integer; ComponentBaseClass: TComponentClass; CustomModuleClass: TCustomModuleClass) (
TRegisterCustomModuleProc)();
```

File

DesignIntf

Description

This is type DesignIntf.TRegisterCustomModuleProc.

60.5.20 DesignIntf.TRegisterDesignDragObject Type

Pascal

```
TRegisterDesignDragObject = procedure (DragObjectClass: TDragObjectClass);
```

C++

```
(DragObjectClass: TDragObjectClass) ( TRegisterDesignDragObject)();
```

File

DesignIntf

Description

This is type DesignIntf.TRegisterDesignDragObject.

60.5.21 DesignIntf.TRegisterDesignNotification Type

Pascal

```
TRegisterDesignNotification = procedure (const DesignNotification: IDesignNotification);
```

C++

```
(const DesignNotification: IDesignNotification) ( TRegisterDesignNotification)();
```

File

DesignIntf

Description

This is type DesignIntf.TRegisterDesignNotification.

60.5.22 DesignIntf.TRegisterDragTargetProc Type

Pascal

```
TRegisterDragTargetProc = procedure (const SourceName: string; TargetClass: TDragTargetClass);
```

C++

```
(const SourceName: string; TargetClass: TDragTargetClass) ( TRegisterDragTargetProc)();
```

File

DesignIntf

Description

This is type DesignIntf.TRegisterDragTargetProc.

60.5.23 DesignIntf.TRegisterIDropTarget Type

Pascal

```
TRegisterIDropTarget = procedure (const ADropTarget: IDropTarget);
```

C++

```
(const ADropTarget: IDropTarget) ( TRegisterIDropTarget)();
```

File

DesignIntf

Description

This is type DesignIntf.TRegisterIDropTarget.

60.5.24 DesignIntf.TRegisterPropertyEditorProc Type

Pascal

```
TRegisterPropertyEditorProc = procedure (PropertyType: PTypeInfo; ComponentClass: TClass;
const PropertyName: string; EditorClass: TPropertyEditorClass);
```

C++

```
(PropertyType: PTypeInfo; ComponentClass: TClass; const PropertyName: string; EditorClass:
TPropertyEditorClass) ( TRegisterPropertyEditorProc)();
```

File

DesignIntf

Description

This is type DesignIntf.TRegisterPropertyEditorProc.

60.5.25 DesignIntf.TRegisterPropertyInCategoryProc Type

Pascal

```
TRegisterPropertyInCategoryProc = procedure (const CategoryName: string; ComponentClass:
TClass; PropertyType: PTypeInfo; const PropertyName: string);
```

C++

```
(const CategoryName: string; ComponentClass: TClass; PropertyType: PTypeInfo; const
PropertyName: string) ( TRegisterPropertyInCategoryProc)();
```

File

DesignIntf

Description

This is type DesignIntf.TRegisterPropertyInCategoryProc.

60.5.26 DesignIntf.TRegisterPropertyMapperProc Type

Pascal

```
TRegisterPropertyMapperProc = procedure (Mapper: TPropertyMapperFunc);
```

C++

```
(Mapper: TPropertyMapperFunc) ( TRegisterPropertyMapperProc)();
```

File

DesignIntf

Description

This is type DesignIntf.TRegisterPropertyMapperProc.

60.5.27 DesignIntf.TRegisterSelectionEditorProc Type

Pascal

```
TRegisterSelectionEditorProc = procedure (AClass: TClass; AEditor: TSelectionEditorClass);
```

C++

```
(AClass: TClass; AEditor: TSelectionEditorClass) ( TRegisterSelectionEditorProc)();
```

File

DesignIntf

Description

This is type DesignIntf.TRegisterSelectionEditorProc.

60.5.28 DesignIntf.TSelectionEditorClass Type

Pascal

```
TSelectionEditorClass = class of TBaseSelectionEditor;
```

C++

```
TBaseSelectionEditor TSelectionEditorClass;
```

File

DesignIntf

Description

This is type DesignIntf.TSelectionEditorClass.

60.5.29 DesignIntf.TSetPropertyEditorGroupProc Type

Pascal

```
TSetPropertyEditorGroupProc = procedure (EditorClass: TPropertyEditorClass; GroupClass:
TPersistentClass);
```

C++

```
(EditorClass: TPropertyEditorClass; GroupClass: TPersistentClass) (
TSetPropertyEditorGroupProc)();
```

File

DesignIntf

Description

This is type DesignIntf.TSetPropertyEditorGroupProc.

60.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
EnableDemandLoadReportProc (see page 11249)	This is variable DesignIntf.EnableDemandLoadReportProc.
ForceDemandLoadStateProc (see page 11249)	This is variable DesignIntf.ForceDemandLoadStateProc.
GReferenceExpandable (see page 11250)	This is variable DesignIntf.GReferenceExpandable.
GShowReadOnlyProps (see page 11250)	This is variable DesignIntf.GShowReadOnlyProps.
RegisterComponentEditorProc (see page 11250)	This is variable DesignIntf.RegisterComponentEditorProc.
RegisterComponentGuidelinesProc (see page 11251)	This is variable DesignIntf.RegisterComponentGuidelinesProc.
RegisterCustomModuleProc (see page 11251)	This is variable DesignIntf.RegisterCustomModuleProc.
RegisterDesignDragObjectProc (see page 11251)	This is variable DesignIntf.RegisterDesignDragObjectProc.
RegisterDesignNotificationProc (see page 11252)	This is variable DesignIntf.RegisterDesignNotificationProc.
RegisterDragTargetProc (see page 11252)	This is variable DesignIntf.RegisterDragTargetProc.
RegisterIDropTargetProc (see page 11252)	This is variable DesignIntf.RegisterIDropTargetProc.
RegisterPropertyEditorProc (see page 11253)	This is variable DesignIntf.RegisterPropertyEditorProc.
RegisterPropertyInCategoryProc (see page 11253)	This is variable DesignIntf.RegisterPropertyInCategoryProc.
RegisterPropertyMapperProc (see page 11253)	This is variable DesignIntf.RegisterPropertyMapperProc.
RegisterSelectionEditorProc (see page 11254)	This is variable DesignIntf.RegisterSelectionEditorProc.
SetPropertyEditorGroupProc (see page 11254)	This is variable DesignIntf.SetPropertyEditorGroupProc.
UnregisterDesignNotificationProc (see page 11254)	This is variable DesignIntf.UnregisterDesignNotificationProc.

60.6.1 DesignIntf.EnableDemandLoadReportProc Variable

Pascal

```
EnableDemandLoadReportProc: TEnableDemandLoadReportProc;
```

C++

```
TEnableDemandLoadReportProc EnableDemandLoadReportProc;
```

File

DesignIntf

Description

This is variable DesignIntf.EnableDemandLoadReportProc.

60.6.2 DesignIntf.ForceDemandLoadStateProc Variable

Pascal

```
ForceDemandLoadStateProc: TForceDemandLoadStateProc;
```

C++

```
TForceDemandLoadStateProc ForceDemandLoadStateProc;
```

File

DesignIntf

Description

This is variable DesignIntf.ForceDemandLoadStateProc.

60.6.3 DesignIntf.GReferenceExpandable Variable

Pascal

```
GReferenceExpandable: Boolean = True;
```

C++

```
Boolean GReferenceExpandable = True;
```

File

DesignIntf

Description

This is variable DesignIntf.GReferenceExpandable.

60.6.4 DesignIntf.GShowReadOnlyProps Variable

Pascal

```
GShowReadOnlyProps: Boolean = True;
```

C++

```
Boolean GShowReadOnlyProps = True;
```

File

DesignIntf

Description

This is variable DesignIntf.GShowReadOnlyProps.

60.6.5 DesignIntf.RegisterComponentEditorProc Variable

Pascal

```
RegisterComponentEditorProc: TRegisterComponentEditorProc;
```

C++

```
TRegisterComponentEditorProc RegisterComponentEditorProc;
```

File

DesignIntf

Description

This is variable DesignIntf.RegisterComponentEditorProc.

60.6.6 DesignIntf.RegisterComponentGuidelinesProc Variable

Pascal

```
RegisterComponentGuidelinesProc: TRegisterComponentGuidelines;
```

C++

```
TRegisterComponentGuidelines RegisterComponentGuidelinesProc;
```

File

DesignIntf

Description

This is variable DesignIntf.RegisterComponentGuidelinesProc.

60.6.7 DesignIntf.RegisterCustomModuleProc Variable

Pascal

```
RegisterCustomModuleProc: TRegisterCustomModuleProc;
```

C++

```
TRegisterCustomModuleProc RegisterCustomModuleProc;
```

File

DesignIntf

Description

This is variable DesignIntf.RegisterCustomModuleProc.

60.6.8 DesignIntf.RegisterDesignDragObjectProc Variable

Pascal

```
RegisterDesignDragObjectProc: TRegisterDesignDragObject;
```

C++

```
TRegisterDesignDragObject RegisterDesignDragObjectProc;
```

File

DesignIntf

Description

This is variable DesignIntf.RegisterDesignDragObjectProc.

60.6.9 DesignIntf.RegisterDesignNotificationProc Variable

Pascal

```
RegisterDesignNotificationProc: TRegisterDesignNotification;
```

C++

```
TRegisterDesignNotification RegisterDesignNotificationProc;
```

File

DesignIntf

Description

This is variable DesignIntf.RegisterDesignNotificationProc.

60.6.10 DesignIntf.RegisterDragTargetProc Variable

Pascal

```
RegisterDragTargetProc: TRegisterDragTargetProc;
```

C++

```
TRegisterDragTargetProc RegisterDragTargetProc;
```

File

DesignIntf

Description

This is variable DesignIntf.RegisterDragTargetProc.

60.6.11 DesignIntf.RegisterIDropTargetProc Variable

Pascal

```
RegisterIDropTargetProc: TRegisterIDropTarget;
```

C++

```
TRegisterIDropTarget RegisterIDropTargetProc;
```

File

DesignIntf

Description

This is variable DesignIntf.RegisterIDropTargetProc.

60.6.12 DesignIntf.RegisterPropertyEditorProc Variable

Pascal

```
RegisterPropertyEditorProc: TRegisterPropertyEditorProc;
```

C++

```
TRegisterPropertyEditorProc RegisterPropertyEditorProc;
```

File

DesignIntf

Description

This is variable DesignIntf.RegisterPropertyEditorProc.

60.6.13 DesignIntf.RegisterPropertyInCategoryProc Variable

Pascal

```
RegisterPropertyInCategoryProc: TRegisterPropertyInCategoryProc;
```

C++

```
TRegisterPropertyInCategoryProc RegisterPropertyInCategoryProc;
```

File

DesignIntf

Description

This is variable DesignIntf.RegisterPropertyInCategoryProc.

60.6.14 DesignIntf.RegisterPropertyMapperProc Variable

Pascal

```
RegisterPropertyMapperProc: TRegisterPropertyMapperProc;
```

C++

```
TRegisterPropertyMapperProc RegisterPropertyMapperProc;
```

File

DesignIntf

Description

This is variable DesignIntf.RegisterPropertyMapperProc.

60.6.15 DesignIntf.RegisterSelectionEditorProc Variable

Pascal

```
RegisterSelectionEditorProc: TRegisterSelectionEditorProc;
```

C++

```
TRegisterSelectionEditorProc RegisterSelectionEditorProc;
```

File

DesignIntf

Description

This is variable DesignIntf.RegisterSelectionEditorProc.

60.6.16 DesignIntf.SetPropertyEditorGroupProc Variable

Pascal

```
SetPropertyEditorGroupProc: TSetPropertyEditorGroupProc;
```

C++

```
TSetPropertyEditorGroupProc SetPropertyEditorGroupProc;
```

File

DesignIntf

Description

This is variable DesignIntf.SetPropertyEditorGroupProc.

60.6.17 DesignIntf.UnregisterDesignNotificationProc Variable

Pascal

```
UnregisterDesignNotificationProc: TRegisterDesignNotification;
```

C++

```
TRegisterDesignNotification UnregisterDesignNotificationProc;
```

File

DesignIntf

Description

This is variable DesignIntf.UnregisterDesignNotificationProc.

60.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
MaxIdentLength (see page 11255)	This is constant DesignIntf.MaxIdentLength.
sActionCategoryName (see page 11256)	This is constant DesignIntf.sActionCategoryName.
sAppearanceCategoryName (see page 11256)	This is constant DesignIntf.sAppearanceCategoryName.
sBehaviorCategoryName (see page 11256)	This is constant DesignIntf.sBehaviorCategoryName.
sDatabaseCategoryName (see page 11257)	This is constant DesignIntf.sDatabaseCategoryName.
sDataCategoryName (see page 11257)	This is constant DesignIntf.sDataCategoryName.
sDesignCategoryName (see page 11257)	This is constant DesignIntf.sDesignCategoryName.
sDragNDropCategoryName (see page 11258)	This is constant DesignIntf.sDragNDropCategoryName.
sFocusCategoryName (see page 11258)	This is constant DesignIntf.sFocusCategoryName.
sHelpCategoryName (see page 11258)	This is constant DesignIntf.sHelpCategoryName.
sInputCategoryName (see page 11259)	This is constant DesignIntf.sInputCategoryName.
sInvalidFilter (see page 11259)	This is constant DesignIntf.sInvalidFilter.
sKeyCategoryName (see page 11259)	This is constant DesignIntf.sKeyCategoryName.
sLayoutCategoryName (see page 11260)	This is constant DesignIntf.sLayoutCategoryName.
sLegacyCategoryName (see page 11260)	This is constant DesignIntf.sLegacyCategoryName.
sLinkageCategoryName (see page 11260)	This is constant DesignIntf.sLinkageCategoryName.
sLocaleCategoryName (see page 11261)	This is constant DesignIntf.sLocaleCategoryName.
sLocalizableCategoryName (see page 11261)	This is constant DesignIntf.sLocalizableCategoryName.
sMiscellaneousCategoryName (see page 11261)	This is constant DesignIntf.sMiscellaneousCategoryName.
sMouseCategoryName (see page 11262)	This is constant DesignIntf.sMouseCategoryName.
sPropertyChangeCategoryName (see page 11262)	This is constant DesignIntf.sPropertyChangeCategoryName.
sVisualCategoryName (see page 11262)	This is constant DesignIntf.sVisualCategoryName.
sWindowStyleName (see page 11263)	This is constant DesignIntf.sWindowStyleName.

60.7.1 DesignIntf.MaxIdentLength Constant

Pascal

```
MaxIdentLength = 63;
```

C++

```
MaxIdentLength = 63;
```

File

DesignIntf

Description

This is constant DesignIntf.MaxIdentLength.

60.7.2 DesignIntf.sActionCategoryName Constant

Pascal

```
sActionCategoryName = 'Action';
```

C++

```
sActionCategoryName = 'Action';
```

File

DesignIntf

Description

This is constant DesignIntf.sActionCategoryName.

60.7.3 DesignIntf.sAppearanceCategoryName Constant

Pascal

```
sAppearanceCategoryName = 'Appearance';
```

C++

```
sAppearanceCategoryName = 'Appearance';
```

File

DesignIntf

Description

This is constant DesignIntf.sAppearanceCategoryName.

60.7.4 DesignIntf.sBehaviorCategoryName Constant

Pascal

```
sBehaviorCategoryName = 'Behavior';
```

C++

```
sBehaviorCategoryName = 'Behavior';
```

File

DesignIntf

Description

This is constant DesignIntf.sBehaviorCategoryName.

60.7.5 DesignIntf.sDatabaseCategoryName Constant

Pascal

```
sDatabaseCategoryName = 'Database';
```

C++

```
sDatabaseCategoryName = 'Database';
```

File

DesignIntf

Description

This is constant DesignIntf.sDatabaseCategoryName.

60.7.6 DesignIntf.sDataCategoryName Constant

Pascal

```
sDataCategoryName = 'Data';
```

C++

```
sDataCategoryName = 'Data';
```

File

DesignIntf

Description

This is constant DesignIntf.sDataCategoryName.

60.7.7 DesignIntf.sDesignCategoryName Constant

Pascal

```
sDesignCategoryName = 'Design';
```

C++

```
sDesignCategoryName = 'Design';
```

File

DesignIntf

Description

This is constant DesignIntf.sDesignCategoryName.

60.7.8 DesignIntf.sDragNDropCategoryName Constant

Pascal

```
sDragNDropCategoryName = 'Drag Drop/Docking';
```

C++

```
sDragNDropCategoryName = 'Drag Drop/Docking';
```

File

DesignIntf

Description

This is constant DesignIntf.sDragNDropCategoryName.

60.7.9 DesignIntf.sFocusCategoryName Constant

Pascal

```
sFocusCategoryName = 'Focus';
```

C++

```
sFocusCategoryName = 'Focus';
```

File

DesignIntf

Description

This is constant DesignIntf.sFocusCategoryName.

60.7.10 DesignIntf.sHelpCategoryName Constant

Pascal

```
sHelpCategoryName = 'Help and Hints';
```

C++

```
sHelpCategoryName = 'Help and Hints';
```

File

DesignIntf

Description

This is constant DesignIntf.sHelpCategoryName.

60.7.11 DesignIntf.sInputCategoryName Constant

Pascal

```
sInputCategoryName = 'Input';
```

C++

```
sInputCategoryName = 'Input';
```

File

DesignIntf

Description

This is constant DesignIntf.sInputCategoryName.

60.7.12 DesignIntf.sInvalidFilter Constant

Pascal

```
sInvalidFilter = 'Property filters may only be name, class or type based (%d:%d)';
```

C++

```
sInvalidFilter = 'Property filters may only be name, class or type based (%d:%d)';
```

File

DesignIntf

Description

This is constant DesignIntf.sInvalidFilter.

60.7.13 DesignIntf.sKeyCategoryName Constant

Pascal

```
sKeyCategoryName = 'Key';
```

C++

```
sKeyCategoryName = 'Key';
```

File

DesignIntf

Description

This is constant DesignIntf.sKeyCategoryName.

60.7.14 DesignIntf.sLayoutCategoryName Constant

Pascal

```
sLayoutCategoryName = 'Layout';
```

C++

```
sLayoutCategoryName = 'Layout';
```

File

DesignIntf

Description

This is constant DesignIntf.sLayoutCategoryName.

60.7.15 DesignIntf.sLegacyCategoryName Constant

Pascal

```
sLegacyCategoryName = 'Legacy';
```

C++

```
sLegacyCategoryName = 'Legacy';
```

File

DesignIntf

Description

This is constant DesignIntf.sLegacyCategoryName.

60.7.16 DesignIntf.sLinkageCategoryName Constant

Pascal

```
sLinkageCategoryName = 'Linkage';
```

C++

```
sLinkageCategoryName = 'Linkage';
```

File

DesignIntf

Description

This is constant DesignIntf.sLinkageCategoryName.

60.7.17 DesignIntf.sLocaleCategoryName Constant

Pascal

```
sLocaleCategoryName = 'Locale';
```

C++

```
sLocaleCategoryName = 'Locale';
```

File

DesignIntf

Description

This is constant DesignIntf.sLocaleCategoryName.

60.7.18 DesignIntf.sLocalizableCategoryName Constant

Pascal

```
sLocalizableCategoryName = 'Localizable';
```

C++

```
sLocalizableCategoryName = 'Localizable';
```

File

DesignIntf

Description

This is constant DesignIntf.sLocalizableCategoryName.

60.7.19 DesignIntf.sMiscellaneousCategoryName Constant

Pascal

```
sMiscellaneousCategoryName = 'Miscellaneous';
```

C++

```
sMiscellaneousCategoryName = 'Miscellaneous';
```

File

DesignIntf

Description

This is constant DesignIntf.sMiscellaneousCategoryName.

60.7.20 DesignIntf.sMouseCategoryName Constant

Pascal

```
sMouseCategoryName = 'Mouse';
```

C++

```
sMouseCategoryName = 'Mouse';
```

File

DesignIntf

Description

This is constant DesignIntf.sMouseCategoryName.

60.7.21 DesignIntf.sPropertyChangeCategoryName Constant

Pascal

```
sPropertyChangeCategoryName = 'Property Changed';
```

C++

```
sPropertyChangeCategoryName = 'Property Changed';
```

File

DesignIntf

Description

This is constant DesignIntf.sPropertyChangeCategoryName.

60.7.22 DesignIntf.sVisualCategoryName Constant

Pascal

```
sVisualCategoryName = 'Visual';
```

C++

```
sVisualCategoryName = 'Visual';
```

File

DesignIntf

Description

This is constant DesignIntf.sVisualCategoryName.

60.7.23 DesignIntf.sWindowStyleName Constant

Pascal

```
sWindowStyleName = 'Window Style';
```

C++

```
sWindowStyleName = 'Window Style';
```

File

DesignIntf

Description

This is constant DesignIntf.sWindowStyleName.

61 Dialogs Namespace

This is namespace Dialogs.

Topics

Name	Description
Classes (see page 11269)	The following table lists classes in this documentation.
Functions (see page 11531)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 11548)	The following table lists structs, records, enums in this documentation.
Types (see page 11568)	The following table lists types in this documentation.
Variables (see page 11585)	The following table lists variables in this documentation.
Constants (see page 11586)	The following table lists constants in this documentation.

Classes

	Name	Description
	EPlatformVersionException (see page 11269)	This is class Dialogs.EPlatformVersionException.
	TColorDialog (see page 11270)	TColorDialog displays a modal color-selection dialog.
	TCommonDialog (see page 11277)	TCommonDialog is the ancestor of all components that represent Windows common dialog boxes.
	TCustomFileDialog (see page 11288)	Abstract base class for Vista style file dialogs.
	TCustomFileOpenDialog (see page 11308)	Base class for Vista style file open dialogs.
	TCustomFileSaveDialog (see page 11314)	Base class for Vista style file save dialogs.
	TCustomTaskDialog (see page 11318)	Base class for Task Dialogs.
	TFavoriteLinkItem (see page 11337)	Item in a list of favorite links.
	TFavoriteLinkItems (see page 11340)	List of favorite links.
	TFavoriteLinkItemsEnumerator (see page 11343)	Enumerator for list of favorite items.
	TFileOpenDialog (see page 11346)	Class for Vista style file open dialogs.
	TFileSaveDialog (see page 11357)	Class for Vista style file save dialogs.
	TFileTypeItem (see page 11368)	File type specification.
	TFileTypeItems (see page 11372)	Collection of file type specifications.
	TFindDialog (see page 11376)	TFindDialog displays a Find (see page 11383) dialog that allows users to search for text in a file.
	TFontDialog (see page 11394)	TFontDialog displays a font-selection dialog.
	TOpenDialog (see page 11407)	TOpenDialog displays a file-selection dialog.
	TPageSetupDialog (see page 11441)	TPageSetupDialog displays a page setup dialog.
	TPrintDialog (see page 11458)	TPrintDialog displays a Print dialog.
	TPrinterSetupDialog (see page 11478)	TPrinterSetupDialog displays a print-setup dialog for configuring printers.

	TReplaceDialog (see page 11482)	TReplaceDialog displays a search-and-replace dialog.
	TSaveDialog (see page 11489)	TSaveDialog displays a "Save As" dialog for saving files.
	TTaskDialog (see page 11494)	Class for Task Dialog properties.
	TTaskDialogBaseButtonItem (see page 11507)	Base button class for Task Dialogs.
	TTaskDialogButtonItem (see page 11513)	A single Task Dialog button item.
	TTaskDialogButtons (see page 11517)	A collection of buttons appearing on a Task Dialog.
	TTaskDialogButtonsEnumerator (see page 11521)	Enumerates the button collection for a Task Dialog.
	TTaskDialogProgressBar (see page 11523)	Progress bar class for Task Dialogs. in one, concise sentence.
	TTaskDialogRadioButtonItem (see page 11527)	A single radio button item for Task Dialogs..

Constants

Name	Description
MaxCustomColors (see page 11586)	The number of custom colors on a Color Dialog.
mbAbortIgnore (see page 11586)	Used to define a dialog with Abort and Ignore buttons.
mbAbortRetryIgnore (see page 11587)	Used to define a dialog with Abort, Retry and Ignore buttons.
mbOKCancel (see page 11588)	Used to define a dialog with OK and Cancel buttons.
mbYesAllNoAllCancel (see page 11588)	This is constant Dialogs.mbYesAllNoAllCancel.
mbYesNo (see page 11589)	Used to define a dialog with Yes and No buttons.
mbYesNoCancel (see page 11589)	Used to define a dialog with Yes, No and Cancel buttons.
tdiError (see page 11590)	This is constant Dialogs.tdiError.
tdiInformation (see page 11590)	This is constant Dialogs.tdiInformation.
tdiNone (see page 11591)	This is constant Dialogs.tdiNone.
tdiShield (see page 11591)	This is constant Dialogs.tdiShield.
tdiWarning (see page 11591)	This is constant Dialogs.tdiWarning.

Functions

	Name	Description
	CreateMessageDialog (see page 11531)	Creates a specified message dialog.
	CreateMessageDialog (see page 11531)	Creates a specified message dialog.
	InputBox (see page 11532)	Displays an input dialog box that lets the user enter a string, double, or integer.
	InputQuery (see page 11533)	Displays an input dialog box that lets the user enter a string, double, or integer.
	MessageDlg (see page 11535)	Displays a message dialog box in the center of the screen.
	MessageDlg (see page 11535)	Displays a message dialog box in the center of the screen.
	MessageDlgPos (see page 11537)	Displays a message dialog box at the specified screen coordinates.
	MessageDlgPos (see page 11537)	Displays a message dialog box at the specified screen coordinates.
	MessageDlgPosHelp (see page 11539)	Displays a message dialog box whose help is supplied in a named help file.
	MessageDlgPosHelp (see page 11539)	Displays a message dialog box whose help is supplied in a named help file.
	PromptForFileName (see page 11540)	Displays an open or save dialog, where the user can specify a file name.
	ShowMessage (see page 11541)	Displays a message box with an OK button.
	ShowMessageFmt (see page 11542)	Displays a message box with a formatted message.
	ShowMessagePos (see page 11543)	Displays a message box at a specified location.
	TaskMessageDlg (see page 11543)	Displays a message task dialog box in the center of the screen.
	TaskMessageDlg (see page 11543)	Displays a message task dialog box in the center of the screen.
	TaskMessageDlgPos (see page 11545)	Displays a message task dialog box at the specified screen coordinates.
	TaskMessageDlgPos (see page 11545)	Displays a message task dialog box at the specified screen coordinates.
	TaskMessageDlgPosHelp (see page 11546)	Displays a message task dialog box whose help is supplied in a named help file.
	TaskMessageDlgPosHelp (see page 11546)	Displays a message task dialog box whose help is supplied in a named help file.

Structs, Records, Enums

	Name	Description
	TColorDialogOption (see page 11548)	TColorDialogOption and TColorDialogOptions (see page 11568) specify options for a TColorDialog (see page 11270) component.
	TFileDialogOption (see page 11549)	The settings in TFileDialogOption determine the behavior of a TCustomFileDialog (see page 11288) file dialog. TFileDialogOptions (see page 11571) is a set of TFileDialogOption. These options are the same as the options for the Microsoft IFileDialog::GetOptions and IFileDialog::SetOptions functions.

	TFileDialogOverwriteResponse (see page 11551)	TFileDialogOverwriteResponse indicates the application's response to an overwrite request during a save operation for a TCustomFileSaveDialog (see page 11314) dialog. These are the same values as in the FDE_OVERWRITE_RESPONSE enumerated type used in the Microsoft IFileDialogEvents::OnOverwrite method.
	TFileDialogShareViolationResponse (see page 11552)	TFileDialogShareViolationResponse indicates the application's response to a sharing violation during a file open or save for a TCustomFileDialog (see page 11288) dialog. These are the same values as in the FDE_SHAREVIOLATION_RESPONSE enumerated type used in the Microsoft IFileDialogEvents::OnShareViolation method.
	TFileEditStyle (see page 11552)	TFileEditStyle indicates the type of control that appears in a file selection dialog for the file name.
	TFindOption (see page 11553)	TFindOption and TFindOptions (see page 11573) determine the controls that appear in a find dialog.
	TFontDialogDevice (see page 11554)	TFontDialogDevice indicates the source from which a TFontDialog (see page 11394) instance obtains its list of fonts.
	TFontDialogOption (see page 11555)	TFontDialogOption and TFontDialogOptions (see page 11574) are flags that influence the appearance and behavior of a font dialog.
	TMsgDlgBtn (see page 11556)	TMsgDlgBtn defines a set of values used by MessageDlg (see page 11535) and MessageDlgPos (see page 11537) functions.
	TMsgDlgType (see page 11558)	TMsgDlgType defines values describing the type of message box.
	TOpenOption (see page 11559)	TOpenOption and TOpenOptions (see page 11576) determine the behavior of a file selection dialog.
	TOpenOptionEx (see page 11561)	TOpenOptionEx and TOpenOptionsEx (see page 11578) determine the behavior of a file selection dialog.
	TPageMeasureUnits (see page 11561)	Specifies page setup measure unit types.
	TPageSetupDialogOption (see page 11562)	This is record Dialogs.TPageSetupDialogOption.
	TPageType (see page 11562)	TPageType indicates the type of page being configured for printing.
	TPrintDialogOption (see page 11563)	TPrintDialogOption and TPrintDialogOptions (see page 11580) specify options for a TPrintDialog (see page 11458) component.
	TPrinterKind (see page 11564)	This is record Dialogs.TPrinterKind.
	TPrintRange (see page 11564)	TPrintRange indicates how the user specified the set of pages to print.
	TProgressBarState (see page 11565)	TProgressBarState defines a set of values used by the State property.
	TTaskDialogCommonButton (see page 11565)	Buttons (see page 2289) for Task Dialogs.
	TTaskDialogFlag (see page 11566)	Flags for Task Dialogs.

Types

Name	Description
TColorDialogOptions (see page 11568)	TColorDialogOption (see page 11548) and TColorDialogOptions specify options for a TColorDialog (see page 11270) component.
TCustomColors (see page 11569)	TCustomColors is an array that holds the color values for your custom colors.
TFDApplyEvent (see page 11569)	TFDApplyEvent is the type for event handlers that respond when the user clicks the Apply button in a nonmodal font dialog.
TFileDialogCloseEvent (see page 11570)	Event for user clicking OK.
TFileDialogFolderChangingEvent (see page 11570)	Event before a directory is opened or closed from the dialog.
TFileDialogOptions (see page 11571)	The settings in TFileDialogOptions determine the behavior of a TCustomFileDialog (see page 11288) file dialog. TFileDialogOptions is a set of TFileDialogOptions. These options are the same as the options for the Microsoft IFileDialog::GetOptions and IFileDialog::SetOptions functions.
TFileDialogOverwriteEvent (see page 11572)	Event just before a file would be overwritten.
TFileDialogShareViolationEvent (see page 11572)	Event when sharing violations arise from open or save operations.
TFindOptions (see page 11573)	TFindOption (see page 11553) and TFindOptions determine the controls that appear in a find dialog.
TFindReplaceFunc (see page 11574)	This is type Dialogs.TFindReplaceFunc.
TFontDialogOptions (see page 11574)	TFontDialogOption (see page 11555) and TFontDialogOptions are flags that influence the appearance and behavior of a font dialog.
TIncludeItemEvent (see page 11575)	This is type Dialogs.TIncludeItemEvent.
TMsgDlgButtons (see page 11575)	TMsgDlgButtons defines a set of values used by MessageDlg (see page 11535) and MessageDlgPos (see page 11537).
TOFNotifyEx (see page 11576)	This is type Dialogs.TOFNotifyEx.
TOpenOptions (see page 11576)	TOpenOption (see page 11559) and TOpenOptions determine the behavior of a file selection dialog.

TOpenOptionsEx (see page 11578)	TOpenOptionEx (see page 11561) and TOpenOptionsEx determine the behavior of a file selection dialog.
TPageSetupBeforePaintEvent (see page 11578)	TPageSetupBeforePaintEvent defines the class method called by the BeforePaint event.
TPageSetupDialogOptions (see page 11579)	Specifies page setup dialog options.
TPaintPageEvent (see page 11579)	TPageSetupBeforePaintEvent (see page 11578) defines the class method called by various TPageSetupDialog (see page 11441) events.
TPrintDialogOptions (see page 11580)	TPrintDialogOption (see page 11563) and TPrintDialogOptions specify options for a TPrintDialog (see page 11458) component.
TPrinterOrientation (see page 11581)	TPrinterOrientation indicates the orientation of pages when printing.
TTaskDialogButtonList (see page 11581)	This is type Dialogs.TTaskDialogButtonList.
TTaskDialogCommonButtons (see page 11582)	An array of Buttons (see page 2289) for Task Dialogs.
TTaskDialogFlags (see page 11582)	Flags for Task Dialogs.
TTaskDialogIcon (see page 11583)	This is type Dialogs.TTaskDialogIcon.
TTaskDlgClickEvent (see page 11583)	Event for a button click in a Task Dialog.
TTaskDlgTimerEvent (see page 11584)	Event for the Task Dialog timer.

Variables

Name	Description
ForceCurrentDirectory (see page 11585)	Indicates whether open and save dialogs should display the current directory if no initial directory is assigned.
UseLatestCommonDialogs (see page 11585)	This is variable Dialogs.UseLatestCommonDialogs.

61.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EPlatformVersionException (see page 11269)	This is class Dialogs.EPlatformVersionException.
	TColorDialog (see page 11270)	TColorDialog displays a modal color-selection dialog.
	TCommonDialog (see page 11277)	TCommonDialog is the ancestor of all components that represent Windows common dialog boxes.
	TCustomFileDialog (see page 11288)	Abstract base class for Vista style file dialogs.
	TCustomFileOpenDialog (see page 11308)	Base class for Vista style file open dialogs.
	TCustomFileSaveDialog (see page 11314)	Base class for Vista style file save dialogs.
	TCustomTaskDialog (see page 11318)	Base class for Task Dialogs (see page 11265).
	TFavoriteLinkItem (see page 11337)	Item in a list of favorite links.
	TFavoriteLinkItems (see page 11340)	List of favorite links.
	TFavoriteLinkItemsEnumerator (see page 11343)	Enumerator for list of favorite items.
	TFileOpenDialog (see page 11346)	Class for Vista style file open dialogs.
	TFileSaveDialog (see page 11357)	Class for Vista style file save dialogs.
	TFileTypeItem (see page 11368)	File type specification.
	TFileTypeItems (see page 11372)	Collection of file type specifications.
	TFindDialog (see page 11376)	TFindDialog displays a Find (see page 11383) dialog that allows users to search for text in a file.
	TFontDialog (see page 11394)	TFontDialog displays a font-selection dialog.
	TOpenDialog (see page 11407)	TOpenDialog displays a file-selection dialog.
	TPageSetupDialog (see page 11441)	TPageSetupDialog displays a page setup dialog.
	TPrintDialog (see page 11458)	TPrintDialog displays a Print dialog.
	TPrinterSetupDialog (see page 11478)	TPrinterSetupDialog displays a print-setup dialog for configuring printers.
	TReplaceDialog (see page 11482)	TReplaceDialog displays a search-and-replace dialog.
	TSaveDialog (see page 11489)	TSaveDialog displays a "Save As" dialog for saving files.
	TTaskDialog (see page 11494)	Class for Task Dialog properties.
	TTaskDialogBaseButtonItem (see page 11507)	Base button class for Task Dialogs (see page 11265).
	TTaskDialogButtonItem (see page 11513)	A single Task Dialog button item.
	TTaskDialogButtons (see page 11517)	A collection of buttons appearing on a Task Dialog.
	TTaskDialogButtonsEnumerator (see page 11521)	Enumerates the button collection for a Task Dialog.
	TTaskDialogProgressBar (see page 11523)	Progress bar class for Task Dialogs (see page 11265). in one, concise sentence.
	TTaskDialogRadioButtonItem (see page 11527)	A single radio button item for Task Dialogs (see page 11265)..

61.1.1 EPlatformVersionException Class

Class Hierarchy

Pascal

```
EPlatformVersionException = class(Exception);
```

C++

```
class EPlatformVersionException : public Exception;
```

File

Dialogs

Description

This is class Dialogs.EPlatformVersionException.

61.1.2 TColorDialog Class

TColorDialog displays a modal color-selection dialog.

Class Hierarchy

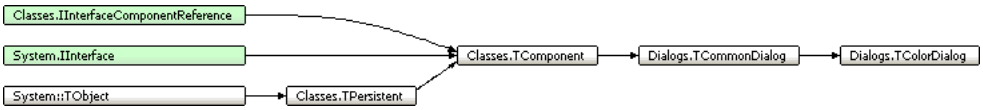

Pascal

```
TColorDialog = class(TCommonDialog);
```

C++

```
class TColorDialog : public TCommonDialog;
```

File

Dialogs

Description

The TColorDialog component displays a Windows dialog box for selecting colors. The dialog does not appear at runtime until it is activated by a call to the Execute (see page 11274) method. When the user selects a color and clicks OK, the dialog closes and the selected color is stored in the Color (see page 11275) property.

61.1.2.1 TColorDialog Members

The following tables list the members exposed by TColorDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoClose (see page 11282)	Generates an OnClose (see page 11287) event.
	DoShow (see page 11283)	Generates an OnShow (see page 11288) event.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	MessageHook (see page 11284)	Provides special Windows message processing for the dialog.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	TaskModalDialog (see page 11284)	Ensures that Windows messages are properly routed when the dialog executes.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WndProc (see page 11285)	Responds to Windows messages sent to the dialog.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Template (see page 11287)	Points to a custom template for the dialog.
	TemplateModule (see page 11287)	This is TemplateModule, a member of class TCommonDialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.

	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11273)	Creates and initializes a color-selection dialog.
	DefaultHandler (see page 11282)	Sends unhandled messages to Windows for processing.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 11274)	Destroys the color-selection dialog and frees its memory.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 11274)	Displays the color-selection dialog.
	Execute (see page 11283)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Handle (see page 11286)	Specifies the window handle of the dialog.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnClose (see page 11287)	Occurs when the dialog closes.
	OnShow (see page 11288)	Occurs when the dialog opens.

Published Properties

	Name	Description
	Color (see page 11275)	Returns the selected color.
	Ctl3D (see page 11276)	Determines whether the dialog has a three-dimensional look.
	CustomColors (see page 11276)	Determines which custom colors are available in the dialog box.
	HelpContext (see page 11286)	Specifies the context number for online Help.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 11277)	Specifies options and default appearance for the dialog.
	Tag (see page 2808)	Stores an integer value as part of a component.

61.1.2.2 TColorDialog.Create Constructor

Creates and initializes a color-selection dialog.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TColorDialog(TComponent * AOwner);
```

Description

Create generates a TColorDialog (see page 11270) instance.

AOwner is the component, typically a form or data module, that is responsible for freeing the TColorDialog (see page 11270) instance. It becomes the value of the Owner property.

Note: Creating a TColorDialog (see page 11270) instance does not cause the dialog to appear to the user. To display the color dialog, use the Execute (see page 11274) method.

See Also

Execute (see page 11274)

Owner (see page 2807)

61.1.2.3 TColorDialog.Destroy Destructor

Destroys the color-selection dialog and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TColorDialog();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free verifies that the color-selection dialog is not nil before calling Destroy.

See Also

Create (see page 11273)

Free (see page 22300)

61.1.2.4 TColorDialog Methods

The methods of the TColorDialog class are listed here.

Public Methods

	Name	Description
	Execute (see page 11274)	Displays the color-selection dialog.

61.1.2.4.1 TColorDialog.Execute Method

Displays the color-selection dialog.

Pascal

```
function Execute(ParentWnd: HWND): Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute(HWND ParentWnd);
```

Description

Execute opens the color-selection dialog, returning true when the user selects a color and clicks OK, or false when the user cancels.

```
if ColorDialog1.Execute then
  Shape1.Brush.Color := ColorDialog1.Color
else
  Shape1.Brush.Color := clGray;
if (ColorDialog1->Execute())
  Shape1->Brush->Color = ColorDialog1->Color;
else
  Shape1->Brush->Color = clGray;
```

C++ Examples:

```
/*
The following code changes the color of a shape control
```

```
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

Color (see page 11275)

61.1.2.5 TColorDialog Properties

The properties of the TColorDialog class are listed here.

Published Properties

	Name	Description
	Color (see page 11275)	Returns the selected color.
	Ctl3D (see page 11276)	Determines whether the dialog has a three-dimensional look.
	CustomColors (see page 11276)	Determines which custom colors are available in the dialog box.
	Options (see page 11277)	Specifies options and default appearance for the dialog.

61.1.2.5.1 TColorDialog.Color Property

Returns the selected color.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

When the user selects a color in the dialog box and clicks OK, the selected color becomes the value of the Color property. To make a default color of choice appear in the dialog when it opens, assign a value to Color in the Object Inspector or in program code, otherwise the default is clBlack.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

CustomColors (see page 11276)

61.1.2.5.2 TColorDialog.Ctl3D Property

Determines whether the dialog has a three-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It has no effect on 32-bit versions of Windows and on NT 4.0 and later.

Ctl3D determined whether controls in the dialog had a flat or beveled look. It required CTL3D32.DLL to be installed in the System32 directory.

See Also

Ctl3D (see page 6198)

61.1.2.5.3 TColorDialog.CustomColors Property

Determines which custom colors are available in the dialog box.

Pascal

```
property CustomColors: TStrings;
```

C++

```
__property TStrings CustomColors;
```

Description

Each custom color is represented as a string of the form ColorX=HexValue. For example, the following string sets the first custom color.

```
ColorA=808022
```

Up to 16 custom colors (ColorA through ColorP) can be set.

See Also

Color (see page 11275)

61.1.2.5.4 TColorDialog.Options Property

Specifies options and default appearance for the dialog.

Pascal

```
property Options: TColorDialogOptions;
```

C++

```
__property TColorDialogOptions Options;
```

Description

Use the Options property to customize the appearance and functionality of the dialog. By default, all options are off.

61.1.3 TCommonDialog Class

TCommonDialog is the ancestor of all components that represent Windows common dialog boxes.

Class Hierarchy

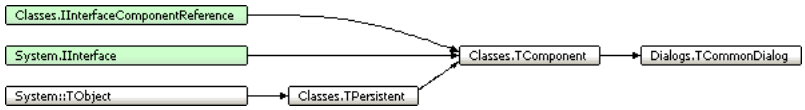

Pascal

```
TCommonDialog = class(TComponent);
```

C++

```
class TCommonDialog : public TComponent;
```

File

Dialogs

Description

TCommonDialog is an abstract class that represents Windows common dialog boxes. Its descendants handle such tasks as finding and opening files, setting fonts and colors, and printing. The dialogs do not appear at runtime until activated by a call to the Execute (see page 11283) method. Except for TFindDialog (see page 11376) and TReplaceDialog (see page 11482), these dialogs are all modal.

The TCommonDialog descendants are nonvisual components that generate a single function call to COMMDLG.DLL, and are

therefore less easily modified than other components. New components that encapsulate common-dialog functions should descend from TCommonDialog. Most other dialog boxes should descend from TForm.

See Also

TColorDialog (see page 11270)

TFindDialog (see page 11376)

TFontDialog (see page 11394)

TForm (see page 13992)

TOpenDialog (see page 11407)

TOpenPictureDialog (see page 13282)

TPrintDialog (see page 11458)

TPrinterSetupDialog (see page 11478)

TReplaceDialog (see page 11482)

TSaveDialog (see page 11489)

TSavePictureDialog (see page 13300)

61.1.3.1 TCommonDialog Members

The following tables list the members exposed by TCommonDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoClose (see page 11282)	Generates an OnClose (see page 11287) event.
	DoShow (see page 11283)	Generates an OnShow (see page 11288) event.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	MessageHook (see page 11284)	Provides special Windows message processing for the dialog.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	TaskModalDialog (see page 11284)	Ensures that Windows messages are properly routed when the dialog executes.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WndProc (see page 11285)	Responds to Windows messages sent to the dialog.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Template (see page 11287)	Points to a custom template for the dialog.
	TemplateModule (see page 11287)	This is TemplateModule, a member of class TCommonDialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11281)	Creates and initializes a TCommonDialog (see page 11277) instance.
V	DefaultHandler (see page 11282)	Sends unhandled messages to Windows for processing.
V	Destroy (see page 11281)	Destroys the TCommonDialog (see page 11277) instance and frees its memory.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Execute (see page 11283)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Handle (see page 11286)	Specifies the window handle of the dialog.

R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnClose (see page 11287)	Occurs when the dialog closes.
	OnShow (see page 11288)	Occurs when the dialog opens.

Published Properties

	Name	Description
	Ctl3D (see page 11285)	Determines whether the dialog has a three-dimensional look.
	HelpContext (see page 11286)	Specifies the context number for online Help.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

61.1.3.2 TCommonDialog.Create Constructor

Creates and initializes a TCommonDialog (see page 11277) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCommonDialog(TComponent * AOwner);
```

Description

Call Create to instantiate a common dialog component at runtime. Dialogs (see page 11265) added to forms or data modules at design time are created automatically.

After calling the inherited constructor, Create performs the following actions:

Placeholder #1.

See Also

Ctl3D (see page 11285)

Destroy (see page 11281)

Create (see page 2766)

61.1.3.3 TCommonDialog.Destroy Destructor

Destroys the TCommonDialog (see page 11277) instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCommonDialog();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free verifies that the dialog reference is not nil before calling Destroy.

See Also

Free (see page 22300)

Create (see page 11281)

61.1.3.4 TCommonDialog Methods

The methods of the TCommonDialog class are listed here.

Protected Methods

	Name	Description
	DoClose (see page 11282)	Generates an OnClose (see page 11287) event.
	DoShow (see page 11283)	Generates an OnShow (see page 11288) event.
V	MessageHook (see page 11284)	Provides special Windows message processing for the dialog.
V	TaskModalDialog (see page 11284)	Ensures that Windows messages are properly routed when the dialog executes.
V	WndProc (see page 11285)	Responds to Windows messages sent to the dialog.

Public Methods

	Name	Description
V	DefaultHandler (see page 11282)	Sends unhandled messages to Windows for processing.
V	Execute (see page 11283)	This is the overview for the Execute method overload.

61.1.3.4.1 TCommonDialog.DefaultHandler Method

Sends unhandled messages to Windows for processing.

Pascal

```
procedure DefaultHandler(var Message); override;
```

C++

```
virtual __fastcall DefaultHandler( Message);
```

Description

DefaultHandler is called by Dispatch when it cannot find a method for a particular message. DefaultHandler sends these messages to the dialog's window procedure, where they are handled by Windows.

See Also

WndProc (see page 11285)

Dispatch

MessageHook (see page 11284)

61.1.3.4.2 TCommonDialog.DoClose Method

Generates an OnClose (see page 11287) event.

Pascal

```
procedure DoClose; dynamic;
```

C++

```
__fastcall DoClose();
```

Description

DoClose is called automatically when the user closes the dialog. Override this method to provide additional processing other than calling the OnClose (see page 11287) event handler. For example, to perform shutdown operations that use protected methods of the dialog class, override DoClose rather than writing an OnClose (see page 11287) event handler (which can access only public methods).

See Also

OnClose (see page 11287)

61.1.3.4.3 TCommonDialog.DoShow Method

Generates an OnShow (see page 11288) event.

Pascal

```
procedure DoShow; dynamic;
```

C++

```
__fastcall DoShow();
```

Description

DoShow is called automatically when the dialog is first displayed. Override this method to provide additional processing other than calling the OnShow (see page 11288) event handler. For example, to perform initializations that use protected methods of the dialog class, override DoShow rather than writing an OnShow (see page 11288) event handler (which can access only public methods).

See Also

OnShow (see page 11288)

61.1.3.4.4 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
V	TCommonDialog.Execute () (see page 11283)	Displays the dialog box.
A	TCommonDialog.Execute (HWND) (see page 11283)	Displays the dialog box.

61.1.3.4.4.1 TCommonDialog.Execute Method ()

Displays the dialog box.

Pascal

```
function Execute: Boolean; virtual; overload;
function Execute(ParentWnd: HWND): Boolean; virtual; abstract; overload;
```

C++

```
virtual __fastcall Boolean Execute();
virtual __fastcall Boolean Execute(HWND ParentWnd) = 0;
```

Description

TCommonDialog (see page 11277) introduces the abstract Execute method, which descendant classes use to open the dialog instance at runtime. Generally, Execute returns true when the user makes a selection and clicks OK, and returns false when the user closes the dialog without making a selection.

See Also

TaskModalDialog (see page 11284)

61.1.3.4.5 TCommonDialog.MessageHook Method

Provides special Windows message processing for the dialog.

Pascal

```
function MessageHook(var Msg: TMessage): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean MessageHook(TMessage Msg);
```

Description

MessageHook intercepts messages sent to the dialog window before the dialog's window procedure gets them. When MessageHook handles a message, it returns true to indicate that the message should not be sent on for further processing. If the window procedure should handle the message, MessageHook returns false.

Override MessageHook to provide extra message processing. For example, if the Template (see page 11287) property substitutes a dialog template with additional controls, use MessageHook to handle messages from those dialog controls.

In TCommonDialog (see page 11277), MessageHook handles Help messages by calling the application's HelpContext (see page 11286) method if the HelpContext (see page 11286) property is defined. Descendants of TCommonDialog (see page 11277) override MessageHook to add additional message processing.

See Also

Template (see page 11287)

HelpContext (see page 13679)

HelpContext (see page 11286)

61.1.3.4.6 TCommonDialog.TaskModalDialog Method

Ensures that Windows messages are properly routed when the dialog executes.

Pascal

```
function TaskModalDialog(DialogFunc: Pointer; var DialogData): Bool; virtual;
```

C++

```
virtual __fastcall Bool TaskModalDialog(void * DialogFunc,  DialogData);
```

Description

Descendants of TCommonDialog (see page 11277) call TaskModalDialog from their Execute (see page 11283) methods.

TaskModalDialog

Disables other windows while the dialog is displayed.

Ensures that application windows are activated and deactivated appropriately.

Ensures that the MessageHook (see page 11284) receives Windows messages before the dialog's window procedure.

Calls the dialog function specified as DialogFunc, passing in the dialog-specific data provided by DialogData.

See Also

Execute (see page 11283)

61.1.3.4.7 TCommonDialog.WndProc Method

Responds to Windows messages sent to the dialog.

Pascal

```
procedure WndProc(var Message: TMessage); virtual;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc is the main window procedure for the dialog. As implemented in TCommonDialog (see page 11277), it simply calls the Dispatch method, where any message handlers respond to messages before the DefaultHandler (see page 11282) method sends them on to the Windows message handler.

Override WndProc to subclass the dialog's window procedure. To add additional message processing without interfering with the existing message handling of the dialog, override MessageHook (see page 11284) instead.

See Also

DefaultHandler (see page 11282)

Dispatch

MessageHook (see page 11284)

61.1.3.5 TCommonDialog Properties

The properties of the TCommonDialog class are listed here.

Protected Properties

	Name	Description
	Template (see page 11287)	Points to a custom template for the dialog.
	TemplateModule (see page 11287)	This is TemplateModule, a member of class TCommonDialog.

Public Properties

	Name	Description
R	Handle (see page 11286)	Specifies the window handle of the dialog.

Published Properties

	Name	Description
	Ctl3D (see page 11285)	Determines whether the dialog has a three-dimensional look.
	HelpContext (see page 11286)	Specifies the context number for online Help.

61.1.3.5.1 TCommonDialog.Ctl3D Property

Determines whether the dialog has a three-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It has no effect on 32-bit versions of Windows and on NT 4.0 and later.

Ctl3D determined whether controls in the dialog had a flat or beveled look. It required CTL3D32.DLL to be installed in the System32 directory.

See Also

Ctl3D (see page 6198)

61.1.3.5.2 TCommonDialog.Handle Property

Specifies the window handle of the dialog.

Pascal

```
property Handle: HWnd;
```

C++

```
__property HWnd Handle;
```

Description

The Handle property, which provides access to the dialog's window handle after a call to Execute (see page 11283), can be used in OnShow (see page 11288) and OnClose (see page 11287) event handlers. Handle is reset to 0 after the dialog closes.

See Also

Execute (see page 11283)

OnClose (see page 11287)

OnShow (see page 11288)

61.1.3.5.3 TCommonDialog.HelpContext Property

Specifies the context number for online Help.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

The HelpContext property is an integer value that determines which Help screen appears when the user requests context-sensitive online Help. If HelpContext is set to the default value of 0, the dialog inherits the Help context of its parent.

See Also

HelpContext (see page 5826)

61.1.3.5.4 TCommonDialog.Template Property

Points to a custom template for the dialog.

Pascal

```
property Template: PChar;
```

C++

```
__property const char * Template;
```

Description

Template points to a resource identifier (a null-terminated string or an integer created with MAKEINTRESOURCE) that identifies a Windows dialog resource in the current HInstance module. The Execute (see page 11283) method uses this resource to customize the appearance of the dialog.

See Also

Execute (see page 11283)

Hinstance

TOpenDialog (see page 11407)

TPrintDialog (see page 11458)

MessageHook (see page 11284)

61.1.3.5.5 TCommonDialog.TemplateModule Property

Pascal

```
property TemplateModule: HINST;
```

C++

```
__property HINST TemplateModule;
```

Description

This is TemplateModule, a member of class TCommonDialog.

61.1.3.6 TCommonDialog Events

The events of the TCommonDialog class are listed here.

Published Events

	Name	Description
	OnClose (see page 11287)	Occurs when the dialog closes.
	OnShow (see page 11288)	Occurs when the dialog opens.

61.1.3.6.1 TCommonDialog.OnClose Event

Occurs when the dialog closes.

Pascal

```
property OnClose: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClose;
```

Description

Write an OnClose event handler to perform special processing when the dialog closes.

See Also

Handle (see page 11286)

61.1.3.6.2 TCommonDialog.OnShow Event

Occurs when the dialog opens.

Pascal

```
property OnShow: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnShow;
```

Description

Write an OnShow event handler to perform special processing when the dialog is displayed.

See Also

Handle (see page 11286)

Execute (see page 11283)

61.1.4 TCustomFileDialog Class

Abstract base class for Vista style file dialogs.

Class Hierarchy

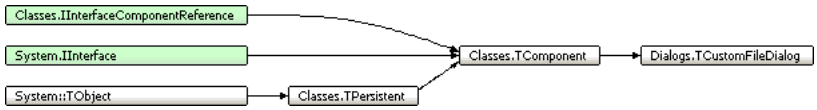

Pascal

```
TCustomFileDialog = class(TComponent);
```

C++

```
class TCustomFileDialog : public TComponent;
```

File

Dialogs

Description

File dialogs, such as open and save, extend this abstract base class for Microsoft Windows Vista style file dialogs. This is a wrapper class for the Windows Vista common item dialog and implements the IFileDialog interface.

See Also

TCustomFileOpenDialog (see page 11308)

TCustomFileSaveDialog (see page 11314)

61.1.4.1 TCustomFileDialog Members

The following tables list the members exposed by TCustomFileDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FileOkClick (see page 11297)	This is FileOkClick, a member of class TCustomFileDialog.
	FolderChange (see page 11297)	This is FolderChange, a member of class TCustomFileDialog.
	FolderChanging (see page 11297)	This is FolderChanging, a member of class TCustomFileDialog.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Overwrite (see page 11298)	This is Overwrite, a member of class TCustomFileDialog.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SelectionChange (see page 11298)	This is SelectionChange, a member of class TCustomFileDialog.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	ShareViolation (see page 11299)	This is ShareViolation, a member of class TCustomFileDialog.
	TypeChange (see page 11299)	This is TypeChange, a member of class TCustomFileDialog.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnExecute (see page 11305)	Occurs when a file dialog gets executed.
	OnFileOkClick (see page 11306)	Occurs when the user clicks the OK button.
	OnFolderChange (see page 11306)	Occurs when a directory is opened or closed from the dialog.
	OnFolderChanging (see page 11307)	Occurs before a directory is opened or closed from the dialog.
	OnOverwrite (see page 11307)	Occurs just before a file would be overwritten.
	OnSelectionChange (see page 11307)	Occurs when the user changes the selection in the dialog.
	OnShareViolation (see page 11308)	Occurs when sharing violations arise from open or save operations.
	OnTypeChange (see page 11308)	Occurs when the user changes the file type selection in the dialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11292)	Creates and initializes an instance.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 11292)	Destroys an instance and frees the associated resources.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 11296)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.

	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	ClientGuid (see page 11299)	GUID associated with a dialog's persisted state.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DefaultExtension (see page 11300)	Default extension to be added to file names.
	DefaultFolder (see page 11300)	Default folder if no recently used folder.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dialog (see page 11300)	Pointer to dialog.
	FavoriteLinks (see page 11301)	List of places available for the user to open or save items.
	FileName (see page 11301)	Text in dialog's File name edit box.
	FileNameLabel (see page 11301)	Text of label next to file name edit box.
R	Files (see page 11302)	List of selected file names.
	FileTypeIndex (see page 11302)	Determines which filter is selected by default when the dialog opens.
	FileTypes (see page 11303)	Determines the file type masks (filters) available in the dialog.
R	Handle (see page 11303)	Specifies the window handle of the dialog.
	OkButtonLabel (see page 11303)	Text of the Open or Save button.
	Options (see page 11304)	List of options controlling dialog.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.

R	ShellItem (see page 11304)	Current shell item.
R	ShellItems (see page 11304)	Array of current shell items.
	Title (see page 11305)	Dialog (see page 11300) title.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

Strict protected Methods

	Name	Description
A	CreateFileDialog (see page 11293)	Create (see page 11292) a file dialog.
	DoOnExecute (see page 11294)	Call OnExecute (see page 11305) event method if defined.
	DoOnFileOkClick (see page 11294)	Call OnFileOkClick (see page 11306) event method if defined.
	DoOnFolderChange (see page 11294)	Call OnFolderChange (see page 11306) event method if defined.
	DoOnFolderChanging (see page 11295)	Call OnFolderChanging (see page 11307) event method if defined.
	DoOnOverwrite (see page 11295)	Call OnOverwrite (see page 11307) event method if defined.
	DoOnSelectionChange (see page 11295)	Call OnSelectionChange (see page 11307) event method if defined.
	DoOnShareViolation (see page 11296)	Call OnShareViolation (see page 11308) event method if defined.
	DoOnTypeChange (see page 11296)	Call OnTypeChange (see page 11308) event method if defined.
	GetFileNames (see page 11298)	This is GetFileNames, a member of class TCustomFileDialog.
	GetItemName (see page 11298)	This is GetItemName, a member of class TCustomFileDialog.
V	GetResults (see page 11298)	This is GetResults, a member of class TCustomFileDialog.

61.1.4.2 TCustomFileDialog.Create Constructor

Creates and initializes an instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomFileDialog(TComponent * AOwner);
```

Description

This method creates and initializes an instance of TCustomFileDialog (see page 11288).

See Also

Destroy (see page 11292)

61.1.4.3 TCustomFileDialog.Destroy Destructor

Destroys an instance and frees the associated resources.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomFileDialog();
```

Description

This method destroys an instance of TCustomFileDialog (see page 11288) and frees the associated resources.

See Also

Create (see page 11292)

61.1.4.4 TCustomFileDialog Methods

The methods of the TCustomFileDialog class are listed here.

Protected Methods

	Name	Description
	FileOkClick (see page 11297)	This is FileOkClick, a member of class TCustomFileDialog.
	FolderChange (see page 11297)	This is FolderChange, a member of class TCustomFileDialog.
	FolderChanging (see page 11297)	This is FolderChanging, a member of class TCustomFileDialog.
	Overwrite (see page 11298)	This is Overwrite, a member of class TCustomFileDialog.
	SelectionChange (see page 11298)	This is SelectionChange, a member of class TCustomFileDialog.
	ShareViolation (see page 11299)	This is ShareViolation, a member of class TCustomFileDialog.
	TypeChange (see page 11299)	This is TypeChange, a member of class TCustomFileDialog.

Public Methods

	Name	Description
V	Execute (see page 11296)	This is the overview for the Execute method overload.

Strict protected Methods

	Name	Description
A	CreateFileDialog (see page 11293)	Create (see page 11292) a file dialog.
	DoOnExecute (see page 11294)	Call OnExecute (see page 11305) event method if defined.
	DoOnFileOkClick (see page 11294)	Call OnFileOkClick (see page 11306) event method if defined.
	DoOnFolderChange (see page 11294)	Call OnFolderChange (see page 11306) event method if defined.
	DoOnFolderChanging (see page 11295)	Call OnFolderChanging (see page 11307) event method if defined.
	DoOnOverwrite (see page 11295)	Call OnOverwrite (see page 11307) event method if defined.
	DoOnSelectionChange (see page 11295)	Call OnSelectionChange (see page 11307) event method if defined.
	DoOnShareViolation (see page 11296)	Call OnShareViolation (see page 11308) event method if defined.
	DoOnTypeChange (see page 11296)	Call OnTypeChange (see page 11308) event method if defined.
	GetFileNames (see page 11298)	This is GetFileNames, a member of class TCustomFileDialog.
	GetItemName (see page 11298)	This is GetItemName, a member of class TCustomFileDialog.
V	GetResults (see page 11298)	This is GetResults, a member of class TCustomFileDialog.

61.1.4.4.1 TCustomFileDialog.CreateFileDialog Method

Create (see page 11292) a file dialog.

Pascal

```
function CreateFileDialog: IFileDialog; virtual; abstract;
```

C++

```
virtual __fastcall IFileDialog CreateFileDialog() = 0;
```

Description

This method creates a file dialog. Since it is an abstract method that is not implemented, it must be overridden and implemented by any class extending TCustomFileDialog (see page 11288). The implementation should create a Windows common item dialog, usually using the Windows function CoCreateInstance.

61.1.4.4.2 TCustomFileDialog.DoOnExecute Method

Call OnExecute (see page 11305) event method if defined.

Pascal

```
procedure DoOnExecute; dynamic;
```

C++

```
__fastcall DoOnExecute();
```

Description

DoOnExecute calls the OnExecute (see page 11305) event, if the event is defined. DoOnExecute is called by the Execute (see page 11296) method, so this event occurs whenever an instance of TCustomFileDialog (see page 11288) gets executed. Execute (see page 11296) causes the dialog to be displayed.

See Also

Execute (see page 11296)

OnExecute (see page 11305)

61.1.4.4.3 TCustomFileDialog.DoOnFileOkClick Method

Call OnFileOkClick (see page 11306) event method if defined.

Pascal

```
function DoOnFileOkClick: Boolean; dynamic;
```

C++

```
__fastcall Boolean DoOnFileOkClick();
```

Description

DoOnFileOkClick calls the OnFileOkClick (see page 11306) event, if the event is defined. DoOnFileOkClick is called whenever the user clicks the OK button. The event occurs just before the dialog returns with a result.

See Also

OnFileOkClick (see page 11306)

61.1.4.4.4 TCustomFileDialog.DoOnFolderChange Method

Call OnFolderChange (see page 11306) event method if defined.

Pascal

```
procedure DoOnFolderChange; dynamic;
```

C++

```
__fastcall DoOnFolderChange();
```

Description

DoOnFolderChange calls the OnFolderChange (see page 11306) event, if the event is defined. DoOnFolderChange is called before a directory is opened or closed from the dialog. Such changes happen when the user double-clicks on a directory, clicks the Up arrow, or uses the list box at the top of the dialog to navigate through the directory structure. The event occurs before the change.

See Also

OnFolderChange (see page 11306)

61.1.4.4.5 TCustomFileDialog.DoOnFolderChanging Method

Call OnFolderChanging (see page 11307) event method if defined.

Pascal

```
function DoOnFolderChanging: Boolean; dynamic;
```

C++

```
__fastcall Boolean DoOnFolderChanging();
```

Description

DoOnFolderChanging calls the OnFolderChanging (see page 11307) event, if the event is defined. DoOnFolderChanging is called when a directory is opened or closed from the dialog. This can happen when the user double-clicks on a directory, clicks the Up arrow, or uses the list box at the top of the dialog to navigate through the directory structure. The event occurs after the change.

See Also

OnFolderChanging (see page 11307)

61.1.4.4.6 TCustomFileDialog.DoOnOverwrite Method

Call OnOverwrite (see page 11307) event method if defined.

Pascal

```
procedure DoOnOverwrite(var Response: TFileDialogOverwriteResponse); dynamic;
```

C++

```
__fastcall DoOnOverwrite(TFileDialogOverwriteResponse Response);
```

Description

DoOnOverwrite calls the OnOverwrite (see page 11307) event, if the event is defined. DoOnOverwrite is called just before a file would be overwritten, so this applies only to save dialogs.

See Also

OnOverwrite (see page 11307)

61.1.4.4.7 TCustomFileDialog.DoOnSelectionChange Method

Call OnSelectionChange (see page 11307) event method if defined.

Pascal

```
procedure DoOnSelectionChange; dynamic;
```

C++

```
__fastcall DoOnSelectionChange();
```

Description

DoOnSelectionChange calls the OnSelectionChange (see page 11307) event, if the event is defined. DoOnSelectionChange is called when the user changes the selection in the dialog.

See Also

OnSelectionChange (see page 11307)

61.1.4.4.8 TCustomFileDialog.DoOnShareViolation Method

Call OnShareViolation (see page 11308) event method if defined.

Pascal

```
procedure DoOnShareViolation(var Response: TFileDialogShareViolationResponse); dynamic;
```

C++

```
__fastcall DoOnShareViolation(TFileDialogShareViolationResponse Response);
```

Description

DoOnShareViolation calls the OnShareViolation (see page 11308) event, if the event is defined. DoOnShareViolation is called when sharing violations that arise from Open or Save operations occur.

See Also

OnShareViolation (see page 11308)

61.1.4.4.9 TCustomFileDialog.DoOnTypeChange Method

Call OnTypeChange (see page 11308) event method if defined.

Pascal

```
procedure DoOnTypeChange; dynamic;
```

C++

```
__fastcall DoOnTypeChange();
```

Description

DoOnTypeChange calls the OnTypeChange (see page 11308) event, if the event is defined. DoOnTypeChange is called when the user changes the file type selection in the dialog.

See Also

OnTypeChange (see page 11308)

61.1.4.4.10 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
	TCustomFileDialog.Execute () (see page 11296)	Displays the dialog box.
	TCustomFileDialog.Execute (HWND) (see page 11296)	Displays the dialog box.

61.1.4.4.10.1 TCustomFileDialog.Execute Method ()

Displays the dialog box.

Pascal

```
function Execute: Boolean; virtual; overload;
function Execute(ParentWnd: HWND): Boolean; virtual; overload;
```

C++

```
virtual __fastcall Boolean Execute();
virtual __fastcall Boolean Execute(HWND ParentWnd);
```

Description

Execute opens the dialog instance at runtime. Generally, Execute returns true when the user makes a selection and clicks OK, and returns false when the user closes the dialog without making a selection.

If the window handle parameter is not supplied, the parent window of the application is used as the parent window of the dialog. Execute calls DoOnExecute (see page 11294), which triggers the OnExecute (see page 11305) event.

See Also

DoOnExecute (see page 11294)

OnExecute (see page 11305)

61.1.4.4.11 TCustomFileDialog.FileOkClick Method

Pascal

```
function FileOkClick: HResult; dynamic;
```

C++

```
__fastcall HRESULT FileOkClick();
```

Description

This is FileOkClick, a member of class TCustomFileDialog.

61.1.4.4.12 TCustomFileDialog.FolderChange Method

Pascal

```
function FolderChange: HResult; dynamic;
```

C++

```
__fastcall HRESULT FolderChange();
```

Description

This is FolderChange, a member of class TCustomFileDialog.

61.1.4.4.13 TCustomFileDialog.FolderChanging Method

Pascal

```
function FolderChanging(psiFolder: IShellItem): HResult; dynamic;
```

C++

```
__fastcall HRESULT FolderChanging(IShellItem psiFolder);
```

Description

This is FolderChanging, a member of class TCustomFileDialog.

61.1.4.4.14 TCustomFileDialog.GetFileNames Method

Pascal

```
function GetFileNames(Items: IShellItemArray): HResult; dynamic;
```

C++

```
__fastcall HRESULT GetFileNames(IShellItemArray Items);
```

Description

This is GetFileNames, a member of class TCustomFileDialog.

61.1.4.4.15 TCustomFileDialog.GetItemName Method

Pascal

```
function GetItemName(Item: IShellItem; var ItemName: TFileName): HResult; dynamic;
```

C++

```
__fastcall HRESULT GetItemName(IShellItem Item, TFileName ItemName);
```

Description

This is GetItemName, a member of class TCustomFileDialog.

61.1.4.4.16 TCustomFileDialog.GetResults Method

Pascal

```
function GetResults: HResult; virtual;
```

C++

```
virtual __fastcall HRESULT GetResults();
```

Description

This is GetResults, a member of class TCustomFileDialog.

61.1.4.4.17 TCustomFileDialog.Overwrite Method

Pascal

```
function Overwrite(psiFile: IShellItem; var Response: Cardinal): HResult; dynamic;
```

C++

```
__fastcall HRESULT Overwrite(IShellItem psiFile, unsigned Response);
```

Description

This is Overwrite, a member of class TCustomFileDialog.

61.1.4.4.18 TCustomFileDialog.SelectionChange Method

Pascal

```
function SelectionChange: HResult; dynamic;
```

C++

```
__fastcall HRESULT SelectionChange();
```

Description

This is SelectionChange, a member of class TCustomFileDialog.

61.1.4.4.19 TCustomFileDialog.ShareViolation Method

Pascal

```
function ShareViolation(psiFile: IShellItem; var Response: Cardinal): HResult; dynamic;
```

C++

```
__fastcall HRESULT ShareViolation(IShellItem psiFile, unsigned Response);
```

Description

This is ShareViolation, a member of class TCustomFileDialog.

61.1.4.4.20 TCustomFileDialog.TypeChange Method

Pascal

```
function TypeChange: HResult; dynamic;
```

C++

```
__fastcall HRESULT TypeChange();
```

Description

This is TypeChange, a member of class TCustomFileDialog.

61.1.4.5 TCustomFileDialog Properties

The properties of the TCustomFileDialog class are listed here.

Public Properties

	Name	Description
	ClientGuid (see page 11299)	GUID associated with a dialog's persisted state.
	DefaultExtension (see page 11300)	Default extension to be added to file names.
	DefaultFolder (see page 11300)	Default folder if no recently used folder.
R	Dialog (see page 11300)	Pointer to dialog.
	FavoriteLinks (see page 11301)	List of places available for the user to open or save items.
	FileName (see page 11301)	Text in dialog's File name edit box.
	FileNameLabel (see page 11301)	Text of label next to file name edit box.
R	Files (see page 11302)	List of selected file names.
	FileTypeIndex (see page 11302)	Determines which filter is selected by default when the dialog opens.
	FileTypes (see page 11303)	Determines the file type masks (filters) available in the dialog.
R	Handle (see page 11303)	Specifies the window handle of the dialog.
	OkButtonLabel (see page 11303)	Text of the Open or Save button.
	Options (see page 11304)	List of options controlling dialog.
R	ShellItem (see page 11304)	Current shell item.
R	ShellItems (see page 11304)	Array of current shell items.
	Title (see page 11305)	Dialog (see page 11300) title.

61.1.4.5.1 TCustomFileDialog.ClientGuid Property

GUID associated with a dialog's persisted state.

Pascal

```
property ClientGuid: string;
```

C++

```
__property AnsiString ClientGuid;
```

Description

ClientGuid holds a GUID associated with a dialog's persisted state. Persisted states for a dialog can include such things as its position and size. Having a GUID allows an application to remember these persisted states for different versions of the dialog.

61.1.4.5.2 TCustomFileDialog.DefaultExtension Property

Default extension to be added to file names.

Pascal

```
property DefaultExtension: string;
```

C++

```
__property AnsiString DefaultExtension;
```

Description

DefaultExtension contains the default extension to add to file names.

See Also

DefaultFolder (see page 11300)

FileName (see page 11301)

Files (see page 11302)

FileTypes (see page 11303)

61.1.4.5.3 TCustomFileDialog.DefaultFolder Property

Default folder if no recently used folder.

Pascal

```
property DefaultFolder: string;
```

C++

```
__property AnsiString DefaultFolder;
```

Description

DefaultFolder contains the name of the default folder if there is no recently used folder.

See Also

DefaultExtension (see page 11300)

61.1.4.5.4 TCustomFileDialog.Dialog Property

Pointer to dialog.

Pascal

```
property Dialog: IFileDialog;
```

C++

```
__property IFileDialog Dialog;
```

Description

Dialog contains a pointer to the dialog itself.

See Also

Handle (see page 11303)

61.1.4.5.5 TCustomFileDialog.FavoriteLinks Property

List of places available for the user to open or save items.

Pascal

```
property FavoriteLinks: TFavoriteLinkItems;
```

C++

```
__property TFavoriteLinkItems FavoriteLinks;
```

Description

FavoriteLinks contains a list of places available for the user to open or save items. This is an instance of TFavoriteLinkItems (see page 11340).

See Also

TFavoriteLinkItems (see page 11340)

61.1.4.5.6 TCustomFileDialog.FileName Property

Text in dialog's File name edit box.

Pascal

```
property FileName: TFileName;
```

C++

```
__property TFileName FileName;
```

Description

FileName contains the text currently entered in the dialog's File name edit box.

See Also

FileNameLabel (see page 11301)

FileTypes (see page 11303)

ShellItem (see page 11304)

61.1.4.5.7 TCustomFileDialog.FileNameLabel Property

Text of label next to file name edit box.

Pascal

```
property FileNameLabel: string;
```

C++

```
__property AnsiString FileNameLabel;
```

Description

FileNameLabel holds the text of the label next to the file name edit box in the dialog.

See Also

FileName (see page 11301)

FileTypes (see page 11303)

OkButtonLabel (see page 11303)

61.1.4.5.8 TCustomFileDialog.Files Property

List of selected file names.

Pascal

```
property Files: TStrings;
```

C++

```
__property TStrings Files;
```

Description

Files is a string list that contains each selected file name with its full directory path. (To let users select multiple file names, set the fdoAllowMultiSelect flag in Options (see page 11304).) Use properties and methods for string lists to traverse this list of files and read individual items.

The example below assigns the list of files in Files to the Items property of a TListBox component.

```
ListBox1.Items.Assign(OpenDialog1.Files);
ListBox1->Items->Assign(OpenDialog1->Files);
```

See Also

FileName (see page 11301)

ShellItems (see page 11304)

61.1.4.5.9 TCustomFileDialog.FileTypeIndex Property

Determines which filter is selected by default when the dialog opens.

Pascal

```
property FileTypeIndex: Cardinal;
```

C++

```
__property unsigned FileTypeIndex;
```

Description

FileTypeIndex determines which of the file types in FileTypes (see page 11303) is selected by default when the dialog opens. Set FileTypeIndex to 1 to choose the first file type in the list as the default, or set FileTypeIndex to 2 to choose the second file type as the default, and so forth. If the value of FileTypeIndex is out or range, the first file type listed in FileTypes (see page 11303) is the default.

See Also

FileTypes (see page 11303)

61.1.4.5.10 TCustomFileDialog.FileTypes Property

Determines the file type masks (filters) available in the dialog.

Pascal

```
property FileTypes: TFileTypeItems;
```

C++

```
__property TFileTypeItems FileTypes;
```

Description

FileTypes is an instance of TFileTypeItems (see page 11372) containing a list of file types, each of which is an instance of TFileTypeItem (see page 11368).

To add file types to a dialog, get the TFileTypeItems (see page 11372) instance from FileTypes and use the TFileTypeItems.Add method as in this sample:

```
with FileOpenDialog1.FileTypes.Add do
begin
  DisplayName := 'All Files';
  FileMask := '*.*';
end;
```

See Also

TFileTypeItem (see page 11368)

TFileTypeItems (see page 11372)

FileTypeIndex (see page 11302)

61.1.4.5.11 TCustomFileDialog.Handle Property

Specifies the window handle of the dialog.

Pascal

```
property Handle: HWnd;
```

C++

```
__property HWnd Handle;
```

Description

The Handle property contains the dialog's window handle after a call to Execute (see page 11296).

See Also

Dialog (see page 11300)

Execute (see page 11296)

61.1.4.5.12 TCustomFileDialog.OkButtonLabel Property

Text of the Open or Save button.

Pascal

```
property OkButtonLabel: string;
```

C++

```
__property AnsiString OkButtonLabel;
```

Description

OkButtonLabel contains text of the Open or Save button.

See Also

FileNameLabel (see page 11301)

61.1.4.5.13 TCustomFileDialog.Options Property

List of options controlling dialog.

Pascal

```
property Options: TFileDialogOptions;
```

C++

```
__property TFileDialogOptions Options;
```

Description

Options is a set of TFileDialogOption (see page 11549) options controlling the dialog's behavior.

See Also

TFileDialogOption

61.1.4.5.14 TCustomFileDialog.ShellItem Property

Current shell item.

Pascal

```
property ShellItem: IShellItem;
```

C++

```
__property IShellItem ShellItem;
```

Description

ShellItem holds the most recently accessed shell item in the dialog from a user's action, such as selecting a file or folder. Shell items are container-like objects in Windows, such as folders. See the Microsoft SDK for information on shell items.

During an event, ShellItem contains the shell item pertinent to the event, such as a file a user selected; after the event is processed, ShellItem is `nil`. After the user clicks OK, ShellItem contains the shell item result from the dialog.

See Also

FileName (see page 11301)

ShellItems (see page 11304)

61.1.4.5.15 TCustomFileDialog.ShellItems Property

Array of current shell items.

Pascal

```
property ShellItems: IShellItemArray;
```

C++

```
__property IShellItemArray ShellItems;
```

Description

ShellItems holds the most recently accessed shell items in the dialog from a user's action, such as selecting files. Shell items are container-like objects in Windows, such as folders. See the Microsoft SDK for information on shell items. The fdoAllowMultiSelect option must be set in Options (see page 11304) so that multiple items can be selected for ShellItems to be used.

During an event, ShellItems contains the shell items pertinent to the event, such as an array of files a user selected; after the event is processed, ShellItems is `nil`. After the user clicks OK, ShellItems contains the shell items result from the dialog.

See Also

Files (see page 11302)

ShellItem (see page 11304)

61.1.4.5.16 TCustomFileDialog.Title Property

Dialog (see page 11300) title.

Pascal

```
property Title: string;
```

C++

```
__property AnsiString Title;
```

Description

Title contains the dialog's title.

See Also

FileNameLabel (see page 11301)

OkButtonLabel (see page 11303)

61.1.4.6 TCustomFileDialog Events

The events of the TCustomFileDialog class are listed here.

Public Events

	Name	Description
	OnExecute (see page 11305)	Occurs when a file dialog gets executed.
	OnFileOkClick (see page 11306)	Occurs when the user clicks the OK button.
	OnFolderChange (see page 11306)	Occurs when a directory is opened or closed from the dialog.
	OnFolderChanging (see page 11307)	Occurs before a directory is opened or closed from the dialog.
	OnOverwrite (see page 11307)	Occurs just before a file would be overwritten.
	OnSelectionChange (see page 11307)	Occurs when the user changes the selection in the dialog.
	OnShareViolation (see page 11308)	Occurs when sharing violations arise from open or save operations.
	OnTypeChange (see page 11308)	Occurs when the user changes the file type selection in the dialog.

61.1.4.6.1 TCustomFileDialog.OnExecute Event

Occurs when a file dialog gets executed.

Pascal

```
property OnExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExecute;
```

Description

OnExecute occurs when an instance of TCustomFileDialog (see page 11288) gets executed by the Execute (see page 11296) method. Execute (see page 11296) causes the dialog to be displayed. The DoOnExecute (see page 11294) method triggers this event.

See Also

DoOnExecute (see page 11294)

Execute (see page 11296)

61.1.4.6.2 TCustomFileDialog.OnFileOkClick Event

Occurs when the user clicks the OK button.

Pascal

```
property OnFileOkClick: TFileDialogCloseEvent;
```

C++

```
__property TFileDialogCloseEvent OnFileOkClick;
```

Description

OnFileOkClick occurs when the user clicks the OK button. The event occurs just before the dialog returns with a result. If an event handler is defined, the property contains a pointer to a TFileDialogCloseEvent (see page 11570). The DoOnFileOkClick (see page 11294) method triggers this event.

See Also

DoOnFileOkClick (see page 11294)

TFileDialogCloseEvent

61.1.4.6.3 TCustomFileDialog.OnFolderChange Event

Occurs when a directory is opened or closed from the dialog.

Pascal

```
property OnFolderChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnFolderChange;
```

Description

The OnFolderChange event occurs when the user changes the directory whose contents are displayed in the dialog. This can happen when the user double-clicks on a directory, clicks the Up arrow, or uses the list box at the top of the dialog to navigate through the directory structure. The event occurs after the folder changes. The DoOnFolderChange (see page 11294) method triggers this event.

See Also

DoOnFolderChange (see page 11294)

61.1.4.6.4 TCustomFileDialog.OnFolderChanging Event

Occurs before a directory is opened or closed from the dialog.

Pascal

```
property OnFolderChanging: TFileDialogFolderChangingEvent;
```

C++

```
__property TFileDialogFolderChangingEvent OnFolderChanging;
```

Description

OnFileOkClick (see page 11306) occurs before a directory is opened or closed from the dialog. This can happen when the user double-clicks on a directory, clicks the Up arrow, or uses the list box at the top of the dialog to navigate through the directory structure. If an event handler is defined, the property contains a pointer to a TFileDialogFolderChangingEvent (see page 11570). The DoOnFolderChanging (see page 11295) method triggers this event.

See Also

DoOnFolderChanging (see page 11295)

TFileDialogFolderChangingEvent

61.1.4.6.5 TCustomFileDialog.OnOverwrite Event

Occurs just before a file would be overwritten.

Pascal

```
property OnOverwrite: TFileDialogOverwriteEvent;
```

C++

```
__property TFileDialogOverwriteEvent OnOverwrite;
```

Description

OnOverwrite occurs just before a file would be overwritten, so this applies only to save dialogs. If an event handler is defined, the property contains a pointer to a TFileDialogOverwriteEvent (see page 11572). The DoOnOverwrite (see page 11295) method triggers this event.

See Also

DoOnOverwrite (see page 11295)

TFileDialogOverwriteEvent

61.1.4.6.6 TCustomFileDialog.OnSelectionChange Event

Occurs when the user changes the selection in the dialog.

Pascal

```
property OnSelectionChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSelectionChange;
```

Description

OnSelectionChange occurs when the user changes the selection in the dialog. The DoOnSelectionChange (see page 11295) method triggers this event.

See Also

DoOnSelectionChange (see page 11295)

61.1.4.6.7 TCustomFileDialog.OnShareViolation Event

Occurs when sharing violations arise from open or save operations.

Pascal

```
property OnShareViolation: TFileDialogShareViolationEvent;
```

C++

```
__property TFileDialogShareViolationEvent OnShareViolation;
```

Description

OnShareViolation occurs when sharing violations arise from open or save operations. If an event handler is defined, the property contains a pointer to a TFileDialogShareViolationEvent (see page 11572). The DoOnShareViolation (see page 11296) method triggers this event.

See Also

DoOnShareViolation (see page 11296)

TFileDialogShareViolationEvent

61.1.4.6.8 TCustomFileDialog.OnTypeChange Event

Occurs when the user changes the file type selection in the dialog.

Pascal

```
property OnTypeChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnTypeChange;
```

Description

OnTypeChange occurs when the user changes the file type selection in the dialog. The DoOnTypeChange (see page 11296) method triggers this event.

See Also

DoOnTypeChange (see page 11296)

61.1.5 TCustomFileOpenDialog Class

Base class for Vista style file open dialogs.

Class Hierarchy

Pascal

```
TCustomFileOpenDialog = class(TCustomFileDialog);
```

C++

```
class TCustomFileOpenDialog : public TCustomFileDialog;
```

File

Dialogs

Description

This class extends TCustomFileDialog (see page 11288). File open dialogs extend this base class for Microsoft Windows Vista style file open dialogs. This is a wrapper class for the Windows Vista CLSID_FileOpenDialog and implements the IFileOpenDialog interface.

See Also

TCustomFileDialog (see page 11288)

61.1.5.1 TCustomFileOpenDialog Members

The following tables list the members exposed by TCustomFileOpenDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CreateFileDialog (see page 11313)	Create (see page 11292) an open file dialog.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FileOkClick (see page 11297)	This is FileOkClick, a member of class TCustomFileDialog.
	FolderChange (see page 11297)	This is FolderChange, a member of class TCustomFileDialog.
	FolderChanging (see page 11297)	This is FolderChanging, a member of class TCustomFileDialog.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetResults (see page 11313)	Get user's choices in dialog.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.

	Notification (see page 2783)	Forwards notification messages to all owned components.
	Overwrite (see page 11298)	This is Overwrite, a member of class TCustomFileDialog.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SelectionChange (see page 11313)	Respond to a selection change.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	ShareViolation (see page 11299)	This is ShareViolation, a member of class TCustomFileDialog.
	TypeChange (see page 11299)	This is TypeChange, a member of class TCustomFileDialog.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnExecute (see page 11305)	Occurs when a file dialog gets executed.
	OnFileOkClick (see page 11306)	Occurs when the user clicks the OK button.
	OnFolderChange (see page 11306)	Occurs when a directory is opened or closed from the dialog.
	OnFolderChanging (see page 11307)	Occurs before a directory is opened or closed from the dialog.
	OnOverwrite (see page 11307)	Occurs just before a file would be overwritten.
	OnSelectionChange (see page 11307)	Occurs when the user changes the selection in the dialog.
	OnShareViolation (see page 11308)	Occurs when sharing violations arise from open or save operations.
	OnTypeChange (see page 11308)	Occurs when the user changes the file type selection in the dialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.

	Create (see page 11292)	Creates and initializes an instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 11292)	Destroys an instance and frees the associated resources.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 11296)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	ClientGuid (see page 11299)	GUID associated with a dialog's persisted state.

R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DefaultExtension (see page 11300)	Default extension to be added to file names.
	DefaultFolder (see page 11300)	Default folder if no recently used folder.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dialog (see page 11300)	Pointer to dialog.
	FavoriteLinks (see page 11301)	List of places available for the user to open or save items.
	FileName (see page 11301)	Text in dialog's File name edit box.
	FileNameLabel (see page 11301)	Text of label next to file name edit box.
R	Files (see page 11302)	List of selected file names.
	FileTypeIndex (see page 11302)	Determines which filter is selected by default when the dialog opens.
	FileTypes (see page 11303)	Determines the file type masks (filters) available in the dialog.
R	Handle (see page 11303)	Specifies the window handle of the dialog.
	OkButtonLabel (see page 11303)	Text of the Open or Save button.
	Options (see page 11304)	List of options controlling dialog.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	ShellItem (see page 11304)	Current shell item.
R	ShellItems (see page 11304)	Array of current shell items.
	Title (see page 11305)	Dialog (see page 11300) title.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

Strict protected Methods

	Name	Description
	DoOnExecute (see page 11294)	Call OnExecute (see page 11305) event method if defined.
	DoOnFileOkClick (see page 11294)	Call OnFileOkClick (see page 11306) event method if defined.
	DoOnFolderChange (see page 11294)	Call OnFolderChange (see page 11306) event method if defined.
	DoOnFolderChanging (see page 11295)	Call OnFolderChanging (see page 11307) event method if defined.
	DoOnOverwrite (see page 11295)	Call OnOverwrite (see page 11307) event method if defined.
	DoOnSelectionChange (see page 11295)	Call OnSelectionChange (see page 11307) event method if defined.
	DoOnShareViolation (see page 11296)	Call OnShareViolation (see page 11308) event method if defined.
	DoOnTypeChange (see page 11296)	Call OnTypeChange (see page 11308) event method if defined.
	GetFileNames (see page 11298)	This is GetFileNames, a member of class TCustomFileDialog.
	GetItemName (see page 11298)	This is GetItemName, a member of class TCustomFileDialog.

61.1.5.2 TCustomFileOpenDialog Methods

The methods of the TCustomFileOpenDialog class are listed here.

Protected Methods

	Name	Description
V	CreateFileDialog (see page 11313)	Create (see page 11292) an open file dialog.
V	GetResults (see page 11313)	Get user's choices in dialog.

	SelectionChange (see page 11313)	Respond to a selection change.

61.1.5.2.1 TCustomFileOpenDialog.CreateFileDialog Method

Create (see page 11292) an open file dialog.

Pascal

```
function CreateFileDialog: IFileDialog; override;
```

C++

```
virtual __fastcall IFileDialog CreateFileDialog();
```

Description

This method creates a Windows CLSID_FileOpenDialog using the Windows function CoCreateInstance.

See Also

TCustomFileDialog (see page 11288)

CreateFileDialog (see page 11293)

61.1.5.2.2 TCustomFileOpenDialog.GetResults Method

Get user's choices in dialog.

Pascal

```
function GetResults: HResult; override;
```

C++

```
virtual __fastcall HRESULT GetResults();
```

Description

GetResults obtains the user's choices in a dialog. The fdoAllowMultiSelect option must be set in Options (see page 11304) to allow multiple items to be selected. This returns S_OK if successful or an error value otherwise.

See Also

Options (see page 11304)

61.1.5.2.3 TCustomFileOpenDialog.SelectionChange Method

Respond to a selection change.

Pascal

```
function SelectionChange: HResult; override;
```

C++

```
virtual __fastcall HRESULT SelectionChange();
```

Description

SelectionChange performs the appropriate processing for a change to the selection in the dialog. TCustomFileDialog.DoOnSelectionChange (see page 11295) is called. The ShellItem (see page 11304) and ShellItems (see page 11304) properties are updated.

See Also

TCustomFileDialog (see page 11288)

ShellItem (see page 11304)

ShellItems (see page 11304)

DoOnSelectionChange (see page 11295)

61.1.6 TCustomFileSaveDialog Class

Base class for Vista style file save dialogs.

Class Hierarchy

Pascal

```
TCustomFileSaveDialog = class(TCustomFileDialog);
```

C++

```
class TCustomFileSaveDialog : public TCustomFileDialog;
```

File

Dialogs

Description

This class extends TCustomFileDialog (see page 11288). File save dialogs extend this base class for Microsoft Windows Vista style file save dialogs. This is a wrapper class for the Windows Vista CLSID_FileSaveDialog and implements the IFileSaveDialog interface.

See Also

TCustomFileDialog (see page 11288)

61.1.6.1 TCustomFileSaveDialog Members

The following tables list the members exposed by TCustomFileSaveDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CreateFileDialog (see page 11318)	Create (see page 11292) an save file dialog.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FileOkClick (see page 11297)	This is FileOkClick, a member of class TCustomFileDialog.
	FolderChange (see page 11297)	This is FolderChange, a member of class TCustomFileDialog.
	FolderChanging (see page 11297)	This is FolderChanging, a member of class TCustomFileDialog.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Overwrite (see page 11298)	This is Overwrite, a member of class TCustomFileDialog.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SelectionChange (see page 11298)	This is SelectionChange, a member of class TCustomFileDialog.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	ShareViolation (see page 11299)	This is ShareViolation, a member of class TCustomFileDialog.
	TypeChange (see page 11299)	This is TypeChange, a member of class TCustomFileDialog.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnExecute (see page 11305)	Occurs when a file dialog gets executed.
	OnFileOkClick (see page 11306)	Occurs when the user clicks the OK button.
	OnFolderChange (see page 11306)	Occurs when a directory is opened or closed from the dialog.
	OnFolderChanging (see page 11307)	Occurs before a directory is opened or closed from the dialog.
	OnOverwrite (see page 11307)	Occurs just before a file would be overwritten.
	OnSelectionChange (see page 11307)	Occurs when the user changes the selection in the dialog.
	OnShareViolation (see page 11308)	Occurs when sharing violations arise from open or save operations.
	OnTypeChange (see page 11308)	Occurs when the user changes the file type selection in the dialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 11292)	Creates and initializes an instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 11292)	Destroys an instance and frees the associated resources.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Execute (see page 11296)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	ClientGuid (see page 11299)	GUID associated with a dialog's persisted state.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DefaultExtension (see page 11300)	Default extension to be added to file names.
	DefaultFolder (see page 11300)	Default folder if no recently used folder.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dialog (see page 11300)	Pointer to dialog.
	FavoriteLinks (see page 11301)	List of places available for the user to open or save items.
	FileName (see page 11301)	Text in dialog's File name edit box.
	FileNameLabel (see page 11301)	Text of label next to file name edit box.
R	Files (see page 11302)	List of selected file names.
	FileTypeIndex (see page 11302)	Determines which filter is selected by default when the dialog opens.
	FileTypes (see page 11303)	Determines the file type masks (filters) available in the dialog.
R	Handle (see page 11303)	Specifies the window handle of the dialog.
	OkButtonLabel (see page 11303)	Text of the Open or Save button.
	Options (see page 11304)	List of options controlling dialog.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	ShellItem (see page 11304)	Current shell item.
R	ShellItems (see page 11304)	Array of current shell items.
	Title (see page 11305)	Dialog (see page 11300) title.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

Strict protected Methods

	Name	Description
	DoOnExecute (see page 11294)	Call OnExecute (see page 11305) event method if defined.
	DoOnFileOkClick (see page 11294)	Call OnFileOkClick (see page 11306) event method if defined.
	DoOnFolderChange (see page 11294)	Call OnFolderChange (see page 11306) event method if defined.
	DoOnFolderChanging (see page 11295)	Call OnFolderChanging (see page 11307) event method if defined.

	DoOnOverwrite (see page 11295)	Call OnOverwrite (see page 11307) event method if defined.
	DoOnSelectionChange (see page 11295)	Call OnSelectionChange (see page 11307) event method if defined.
	DoOnShareViolation (see page 11296)	Call OnShareViolation (see page 11308) event method if defined.
	DoOnTypeChange (see page 11296)	Call OnTypeChange (see page 11308) event method if defined.
	GetFileNames (see page 11298)	This is GetFileNames, a member of class TCustomFileDialog.
	GetItemName (see page 11298)	This is GetItemName, a member of class TCustomFileDialog.
V	GetResults (see page 11298)	This is GetResults, a member of class TCustomFileDialog.

61.1.6.2 TCustomFileSaveDialog Methods

The methods of the TCustomFileSaveDialog class are listed here.

Protected Methods

	Name	Description
V	CreateFileDialog (see page 11318)	Create (see page 11292) an save file dialog.

61.1.6.2.1 TCustomFileSaveDialog.CreateFileDialog Method

Create (see page 11292) an save file dialog.

Pascal

```
function CreateFileDialog: IFileDialog; override;
```

C++

```
virtual __fastcall IFileDialog CreateFileDialog();
```

Description

This method creates a Windows CLSID_FileSaveDialog using the Windows function CoCreateInstance.

See Also

TCustomFileDialog (see page 11288)

CreateFileDialog (see page 11293)

61.1.7 TCustomTaskDialog Class

Base class for Task Dialogs (see page 11265).

Class Hierarchy

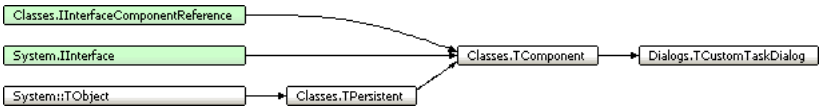

Pascal

```
TCustomTaskDialog = class(TComponent);
```

C++

```
class TCustomTaskDialog : public TComponent;
```

File

Dialogs

Description

TCustomTaskDialog provides the base class for Task Dialogs (see page 11265). Derived classes should inherit from TTaskDialog (see page 11494) instead of TCustomTaskDialog.

61.1.7.1 TCustomTaskDialog Members

The following tables list the members exposed by TCustomTaskDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CallbackProc (see page 11323)	An application-defined function used with Task Dialogs (see page 11265).
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnButtonClicked (see page 11334)	Button (see page 11328) click event.
	OnDialogConstructed (see page 11335)	Dialog constructed event.
	OnDialogCreated (see page 11335)	Task Dialog creation event.
	OnDialogDestroyed (see page 11335)	Task dialog destruction event..
	OnExpanded (see page 11335)	Task dialog expanded event.
	OnHyperlinkClicked (see page 11336)	Task dialog hyperlink clicked event.
	OnNavigated (see page 11336)	Task Dialog navigation event.
	OnRadioButtonClicked (see page 11336)	Task Dialog radio button click event.
	OnTimer (see page 11336)	Task dialog timer event.
	OnVerificationClicked (see page 11337)	Task Dialog verification check box event.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11322)	Creates a custom Task Dialog instance.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 11322)	Destroys an instance of a custom Task Dialog.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 11326)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Button (see page 11328)	Selected button.
	Buttons (see page 11328)	The collection of all buttons on the Task Dialog.
	Caption (see page 11328)	Caption for the Task Dialog.
	CommonButtons (see page 11328)	Specifies the push buttons displayed in the Task Dialog.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	CustomFooterIcon (see page 11329)	Indicates the custom footer icon, if any, for the Task Dialog.
	CustomMainIcon (see page 11329)	Indicates the custom main icon, if any, for the Task Dialog.
	DefaultButton (see page 11329)	The default button for this Task Dialog.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	ExpandButtonCaption (see page 11329)	Additional information for this button.
R	Expanded (see page 11330)	Indicates whether the Task Dialog is expanded.
	ExpandedText (see page 11330)	Text (see page 11333) to display when the Task Dialog is expanded.
	Flags (see page 11330)	Flags for this Task Dialog.
	FooterIcon (see page 11331)	Indicates which standard footer icon to display.
	FooterText (see page 11331)	String to be used in the footer area of the Task Dialog.
R	Handle (see page 11331)	Handle to the parent window.
	HelpContext (see page 11332)	Help context for the Task Dialog.
	MainIcon (see page 11332)	Indicates which standard main icon to display.

	ModalResult (see page 11332)	Modal result.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ProgressBar (see page 11332)	Progress bar for this Task Dialog.
R	RadioButton (see page 11333)	Selected radio button.
	RadioButtons (see page 11333)	Collection of all radio buttons on this Task Dialog.
	Text (see page 11333)	The main text on this Task Dialog..
	Title (see page 11333)	The title of this Task Dialog.
R	URL (see page 11334)	URL for the Task Dialog.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	VerificationText (see page 11334)	Text (see page 11333) used to label the verification check box.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

Strict protected Methods

	Name	Description
	DoExecute (see page 11323)	Executes the Task Dialog.
	DoOnButtonClicked (see page 11324)	Performs the click function of the Task Dialog.
	DoOnDialogContructed (see page 11324)	Performs the on-construction actions of the Task Dialog.
	DoOnDialogCreated (see page 11324)	Performs the on-creation actrions of a Task Dialog.
	DoOnDialogDestroyed (see page 11324)	Performs the on-destruction actions of a Task Dialog.
	DoOnExpandButtonClicked (see page 11325)	Performs the expansion action of a Task Dialog when the **Expand** button is pushed.
	DoOnHelp (see page 11325)	Performs the help function for a Task Dialog.
	DoOnHyperlinkClicked (see page 11325)	Performs the hyperlink click action of a Task Dialog.
	DoOnNavigated (see page 11325)	Performs the navigation action of a Task Dialog.
	DoOnRadioButtonClicked (see page 11326)	Performs the radio button click action of a Task Dialog.
	DoOnTimer (see page 11326)	Performs the timer action of a Task Dialog.
	DoOnVerificationClicked (see page 11326)	Performs the verification check action of a Task Dialog.

61.1.7.2 TCustomTaskDialog.Create Constructor

Creates a custom Task Dialog instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomTaskDialog(TComponent * AOwner);
```

Description

Use Create to create an instance of TCustomTaskDialog (see page 11318). Derived classes should inherit from the TTaskDialog (see page 11494) instead of this one.

61.1.7.3 TCustomTaskDialog.Destroy Destructor

Destroys an instance of a custom Task Dialog.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomTaskDialog();
```

Description

Use Destroy to destroy an instance of TCustomTaskDialog (see page 11318).

61.1.7.4 TCustomTaskDialog Methods

The methods of the TCustomTaskDialog class are listed here.

Protected Methods

	Name	Description
	CallbackProc (see page 11323)	An application-defined function used with Task Dialogs (see page 11265).

Public Methods

	Name	Description
	Execute (see page 11326)	This is the overview for the Execute method overload.

Strict protected Methods

	Name	Description
	DoExecute (see page 11323)	Executes the Task Dialog.
	DoOnButtonClicked (see page 11324)	Performs the click function of the Task Dialog.
	DoOnDialogContructed (see page 11324)	Performs the on-construction actions of the Task Dialog.
	DoOnDialogCreated (see page 11324)	Performs the on-creation actrions of a Task Dialog.
	DoOnDialogDestroyed (see page 11324)	Performs the on-destruction actions of a Task Dialog.
	DoOnExpandButtonClicked (see page 11325)	Performs the expansion action of a Task Dialog when the **Expand** button is pushed.
	DoOnHelp (see page 11325)	Performs the help function for a Task Dialog.
	DoOnHyperlinkClicked (see page 11325)	Performs the hyperlink click action of a Task Dialog.
	DoOnNavigated (see page 11325)	Performs the navigation action of a Task Dialog.
	DoOnRadioButtonClicked (see page 11326)	Performs the radio button click action of a Task Dialog.
	DoOnTimer (see page 11326)	Performs the timer action of a Task Dialog.
	DoOnVerificationClicked (see page 11326)	Performs the verification check action of a Task Dialog.

61.1.7.4.1 TCustomTaskDialog.CallbackProc Method

An application-defined function used with Task Dialogs (see page 11265).

Pascal

```
function CallbackProc(hwnd: HWND; msg: UINT; wParam: WPARAM; lParam: LPARAM; lpRefData:
LONG_PTR): HResult; dynamic;
```

C++

```
__fastcall HRESULT CallbackProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam, LONG_PTR
lpRefData);
```

Description

Use CallbackProc as an application-defined function used with the TaskDialogIndirect function. It receives messages from the Task Dialog when various events occur. An application must register this callback function. .

61.1.7.4.2 TCustomTaskDialog.DoExecute Method

Executes the Task Dialog.

Pascal

```
function DoExecute(ParentWnd: HWND): Boolean; dynamic;
```

C++

```
__fastcall Boolean DoExecute(HWND ParentWnd);
```

Description

Use DoExecute to perform the execution of the Task Dialog.

61.1.7.4.3 TCustomTaskDialog.DoOnButtonClicked Method

Performs the click function of the Task Dialog.

Pascal

```
procedure DoOnButtonClicked(AModalResult: Integer; var CanClose: Boolean); dynamic;
```

C++

```
__fastcall DoOnButtonClicked(int AModalResult, Boolean CanClose);
```

Description

Use DoOnButtonClicked to perform the click action of the Task Dialog.

61.1.7.4.4 TCustomTaskDialog.DoOnDialogContructed Method

Performs the on-construction actions of the Task Dialog.

Pascal

```
procedure DoOnDialogContructed; dynamic;
```

C++

```
__fastcall DoOnDialogContructed();
```

Description

Use DoOnDialogContructed to perform the on-construction actions of the Task Dialog.

61.1.7.4.5 TCustomTaskDialog.DoOnDialogCreated Method

Performs the on-creation actrions of a Task Dialog.

Pascal

```
procedure DoOnDialogCreated; dynamic;
```

C++

```
__fastcall DoOnDialogCreated();
```

Description

Use DoOnDialogCreated to perform the on-creation actions of the Task Dialog.

61.1.7.4.6 TCustomTaskDialog.DoOnDialogDestroyed Method

Performs the on-destruction actions of a Task Dialog.

Pascal

```
procedure DoOnDialogDestroyed; dynamic;
```

C++

```
__fastcall DoOnDialogDestroyed();
```

Description

Use DoOnDialogDestroyed to perform the on-destruction actions of a Task Dialog.

61.1.7.4.7 TCustomTaskDialog.DoOnExpandButtonClicked Method

Performs the expansion action of a Task Dialog when the **Expand** button is pushed.

Pascal

```
procedure DoOnExpandButtonClicked(Expanded: Boolean); dynamic;
```

C++

```
__fastcall DoOnExpandButtonClicked(Boolean Expanded);
```

Description

Use DoOnExpandButtonClicked to perform the expansion action of a Task Dialog.

61.1.7.4.8 TCustomTaskDialog.DoOnHelp Method

Performs the help function for a Task Dialog.

Pascal

```
procedure DoOnHelp; dynamic;
```

C++

```
__fastcall DoOnHelp();
```

Description

Use DoOnHelp to perform the help function for a Task Dialog.

61.1.7.4.9 TCustomTaskDialog.DoOnHyperlinkClicked Method

Performs the hyperlink click action of a Task Dialog.

Pascal

```
procedure DoOnHyperlinkClicked(const AURL: string); dynamic;
```

C++

```
__fastcall DoOnHyperlinkClicked(const AnsiString AURL);
```

Description

Use DoOnHyperlinkClicked to perform the hyperlink click action of a Task Dialog.

61.1.7.4.10 TCustomTaskDialog.DoOnNavigated Method

Performs the navigation action of a Task Dialog.

Pascal

```
procedure DoOnNavigated; dynamic;
```

C++

```
__fastcall DoOnNavigated();
```

Description

Use DoOnNavigated to perform the navigation action of a Task Dialog.

61.1.7.4.11 TCustomTaskDialog.DoOnRadioButtonClicked Method

Performs the radio button click action of a Task Dialog.

Pascal

```
procedure DoOnRadioButtonClicked(ButtonID: Integer); dynamic;
```

C++

```
__fastcall DoOnRadioButtonClicked(int ButtonID);
```

Description

Use DoOnRadioButtonClicked to perform the radion button click action of a Task Dialog.

61.1.7.4.12 TCustomTaskDialog.DoOnTimer Method

Performs the timer action of a Task Dialog.

Pascal

```
procedure DoOnTimer(TickCount: Cardinal; var Reset: Boolean); dynamic;
```

C++

```
__fastcall DoOnTimer(unsigned TickCount, Boolean Reset);
```

Description

Use DoOnTimer to perform the timer action of the Task Dialog.

61.1.7.4.13 TCustomTaskDialog.DoOnVerificationClicked Method

Performs the verification check action of a Task Dialog.

Pascal

```
procedure DoOnVerificationClicked(Checked: Boolean); dynamic;
```

C++

```
__fastcall DoOnVerificationClicked(Boolean Checked);
```

Description

Use DoOnVerificationClicked to perform the verification check action of a Task Dialog.

61.1.7.4.14 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
	TCustomTaskDialog.Execute () (see page 11327)	Displays the Task Dialog.
	TCustomTaskDialog.Execute (HWND) (see page 11327)	Displays the Task Dialog.

61.1.7.4.14.1 TCustomTaskDialog.Execute Method ()

Displays the Task Dialog.

Pascal

```
function Execute: Boolean; overload;
function Execute(ParentWnd: HWND): Boolean; overload;
```

C++

```
__fastcall Boolean Execute();
__fastcall Boolean Execute(HWND ParentWnd);
```

Description

Use Execute to display the Task Dialog.

61.1.7.5 TCustomTaskDialog Properties

The properties of the TCustomTaskDialog class are listed here.

Public Properties

	Name	Description
	Button (see page 11328)	Selected button.
	Buttons (see page 11328)	The collection of all buttons on the Task Dialog.
	Caption (see page 11328)	Caption for the Task Dialog.
	CommonButtons (see page 11328)	Specifies the push buttons displayed in the Task Dialog.
	CustomFooterIcon (see page 11329)	Indicates the custom footer icon, if any, for the Task Dialog.
	CustomMainIcon (see page 11329)	Indicates the custom main icon, if any, for the Task Dialog.
	DefaultButton (see page 11329)	The default button for this Task Dialog.
	ExpandButtonCaption (see page 11329)	Additional information for this button.
R	Expanded (see page 11330)	Indicates whether the Task Dialog is expanded.
	ExpandedText (see page 11330)	Text (see page 11333) to display when the Task Dialog is expanded.
	Flags (see page 11330)	Flags for this Task Dialog.
	FooterIcon (see page 11331)	Indicates which standard footer icon to display.
	FooterText (see page 11331)	String to be used in the footer area of the Task Dialog.
R	Handle (see page 11331)	Handle to the parent window.
	HelpContext (see page 11332)	Help context for the Task Dialog.
	MainIcon (see page 11332)	Indicates which standard main icon to display.
	ModalResult (see page 11332)	Modal result.
	ProgressBar (see page 11332)	Progress bar for this Task Dialog.
R	RadioButton (see page 11333)	Selected radio button.
	RadioButtons (see page 11333)	Collection of all radio buttons on this Task Dialog.
	Text (see page 11333)	The main text on this Task Dialog..
	Title (see page 11333)	The title of this Task Dialog.
R	URL (see page 11334)	URL for the Task Dialog.

	VerificationText (see page 11334)	Text (see page 11333) used to label the verification check box.

61.1.7.5.1 TCustomTaskDialog.Button Property

Selected button.

Pascal

```
property Button: TTaskDialogButtonItem;
```

C++

```
__property TTaskDialogButtonItem Button;
```

Description

Button is the selected button on the Task Dialog.

61.1.7.5.2 TCustomTaskDialog.Buttons Property

The collection of all buttons on the Task Dialog.

Pascal

```
property Buttons: TTaskDialogButtons;
```

C++

```
__property TTaskDialogButtons Buttons;
```

Description

Buttons contains all of the buttons for the Task Dialog.

61.1.7.5.3 TCustomTaskDialog.Caption Property

Caption for the Task Dialog.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption contains the caption for the Task Dialog.

61.1.7.5.4 TCustomTaskDialog.CommonButtons Property

Specifies the push buttons displayed in the Task Dialog.

Pascal

```
property CommonButtons: TTaskDialogCommonButtons;
```

C++

```
__property TTaskDialogCommonButtons CommonButtons;
```

Description

CommonButtons specifies the common push buttons for the Task Dialog. If no buttons are specified, the **OK** button is displayed.

61.1.7.5.5 TCustomTaskDialog.CustomFooterIcon Property

Indicates the custom footer icon, if any, for the Task Dialog.

Pascal

```
property CustomFooterIcon: TIcon;
```

C++

```
__property TIcon CustomFooterIcon;
```

Description

CustomFooterIcon points to the custom footer icon for this Task Dialog. If **tfUseHiconFooter** is set in Flags (see page 11330).

61.1.7.5.6 TCustomTaskDialog.CustomMainIcon Property

Indicates the custom main icon, if any, for the Task Dialog.

Pascal

```
property CustomMainIcon: TIcon;
```

C++

```
__property TIcon CustomMainIcon;
```

Description

CustomMainIcon points to the custom main icon for this Task Dialog. If **tfUseHiconMain** is set in Flags (see page 11330).

61.1.7.5.7 TCustomTaskDialog.DefaultButton Property

The default button for this Task Dialog.

Pascal

```
property DefaultButton: TTaskDialogCommonButton;
```

C++

```
__property TTaskDialogCommonButton DefaultButton;
```

Description

DefaultButton indicates the default button for this Task Dialog.

61.1.7.5.8 TCustomTaskDialog.ExpandButtonCaption Property

Additional information for this button.

Pascal

```
property ExpandButtonCaption: string;
```

C++

```
__property AnsiString ExpandButtonCaption;
```

Description

ExpandButtonCaption contains additional text to be displayed when the caption is expanded.

61.1.7.5.9 TCustomTaskDialog.Expanded Property

Indicates whether the Task Dialog is expanded.

Pascal

```
property Expanded: Boolean;
```

C++

```
__property Boolean Expanded;
```

Description

Expanded is **True** when the Task Dialog is expanded.

61.1.7.5.10 TCustomTaskDialog.ExpandedText Property

Text (see page 11333) to display when the Task Dialog is expanded.

Pascal

```
property ExpandedText: string;
```

C++

```
__property AnsiString ExpandedText;
```

Description

ExpandedText contains additional information to display when the Task Dialog is expanded.

61.1.7.5.11 TCustomTaskDialog.Flags Property

Flags for this Task Dialog.

Pascal

```
property Flags: TTaskDialogFlags;
```

C++

```
__property TTaskDialogFlags Flags;
```

Description

Flags provides flags for TCustomTaskDialog (see page 11318).

Flag	Description
tfEnableHyperlinks	If set, content, footer, and expanded text can include hyperlinks.
tfUseHiconMain	if set, use the custom main icon.
tfUseHiconFooter	If set, use the custom footer icon.
tfAllowDialogCancellation	If set, permits Task Dialog to be closed in the absence of a **Cancel** button.
tfUseCommandLinks	If set, buttons are displayed as command links using a standard dialog glyph.
tfUseCommandLinksNoIcon	If set, buttons are displayed as command links without a glyph.
tfExpandFooterArea	If set, display expanded text in the footer.
tfExpandedByDefault	If set, expanded text is displayed when the Task Dialog opens.

tfVerificationFlagChecked	If set, the verification check box is initially checked.
tfShowProgressBar	If set, display the progress bar.
tfShowMarqueeProgressBar	If set, display marquee progress bar.
tfCallbackTimer	If set, dialogs callback will be called every 200 milliseconds.
tfPositionRelativeToWindow	If set, Task Dialog is centered with respect to parent window.
tfRtlLayout	If set, text reads right-to-left.
tfNoDefaultRadioButton	If set, there is no default radio button.
tfCanBeMinimized	If set, the Task Dialog can be minimized.

61.1.7.5.12 TCustomTaskDialog.FooterIcon Property

Indicates which standard footer icon to display.

Pascal

```
property FooterIcon: TTaskDialogIcon;
```

C++

```
__property TTaskDialogIcon FooterIcon;
```

Description

FooterIcon indicates which standard footer icon to display in the Task Dialog. Legal values include **tdiNone (see page 11591)**, **tdiWarning (see page 11591)**, **tdiError (see page 11590)**, **tdiInformation (see page 11590)**, and **tdiShield (see page 11591)**.

61.1.7.5.13 TCustomTaskDialog.FooterText Property

String to be used in the footer area of the Task Dialog.

Pascal

```
property FooterText: string;
```

C++

```
__property AnsiString FooterText;
```

Description

FooterText provides a string to be used in the footer area of the Task Dialog.

61.1.7.5.14 TCustomTaskDialog.Handle Property

Handle to the parent window.

Pascal

```
property Handle: HWND;
```

C++

```
__property HWND Handle;
```

Description

Handle provides a handle to the parent window. This field can be null.

61.1.7.5.15 TCustomTaskDialog.HelpContext Property

Help context for the Task Dialog.

Pascal

```
property HelpContext: Integer;
```

C++

```
__property int HelpContext;
```

Description

HelpContext provides the help context for the Task Dialog

61.1.7.5.16 TCustomTaskDialog.MainIcon Property

Indicates which standard main icon to display.

Pascal

```
property MainIcon: TTaskDialogIcon;
```

C++

```
__property TTaskDialogIcon MainIcon;
```

Description

MainIcon indicates which standard main icon to display in the Task Dialog. Legal values include **tdiNone (see page 11591)**, **tdiWarning (see page 11591)**, **tdiError (see page 11590)**, **tdiInformation (see page 11590)**, and **tdiShield (see page 11591)**.

61.1.7.5.17 TCustomTaskDialog.ModalResult Property

Modal result.

Pascal

```
property ModalResult: TModalResult;
```

C++

```
__property TModalResult ModalResult;
```

Description

ModalResult contains the modal result of the Task Dialog: mrYes, mrNo, mrOk, mrCancel mrAbort, mrRetry, mrIgnore, mrAll, mrNoToAll, mrYesToAll.

61.1.7.5.18 TCustomTaskDialog.ProgressBar Property

Progress bar for this Task Dialog.

Pascal

```
property ProgressBar: TTaskDialogProgressBar;
```

C++

```
__property TTaskDialogProgressBar ProgressBar;
```

Description

ProgressBar is the TTaskDialogProgressBar (see page 11523) for this Task Dialog.

61.1.7.5.19 TCustomTaskDialog.RadioButton Property

Selected radio button.

Pascal

```
property RadioButton: TTaskDialogRadioButtonItem;
```

C++

```
__property TTaskDialogRadioButtonItem RadioButton;
```

Description

RadioButton is the selected radio button on this Task Dialog.

61.1.7.5.20 TCustomTaskDialog.RadioButtons Property

Collection of all radio buttons on this Task Dialog.

Pascal

```
property RadioButtons: TTaskDialogButtons;
```

C++

```
__property TTaskDialogButtons RadioButtons;
```

Description

RadioButtons is the collection of radio buttons for this class dialog.

61.1.7.5.21 TCustomTaskDialog.Text Property

The main text on this Task Dialog..

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

Text contains the main text to be displayed in this Task Dialog.

61.1.7.5.22 TCustomTaskDialog.Title Property

The title of this Task Dialog.

Pascal

```
property Title: string;
```

C++

```
__property AnsiString Title;
```

Description

Title contains the title of this Task Dialog.

61.1.7.5.23 TCustomTaskDialog.URL Property

URL for the Task Dialog.

Pascal

```
property URL: string;
```

C++

```
__property AnsiString URL;
```

Description

URL contains the URL for the Task Dialog.

61.1.7.5.24 TCustomTaskDialog.VerificationText Property

Text (see page 11333) used to label the verification check box.

Pascal

```
property VerificationText: string;
```

C++

```
__property AnsiString VerificationText;
```

Description

VerificationText contains text use to label the verification check box.

61.1.7.6 TCustomTaskDialog Events

The events of the TCustomTaskDialog class are listed here.

Public Events

	Name	Description
	OnButtonClicked (see page 11334)	Button (see page 11328) click event.
	OnDialogConstructed (see page 11335)	Dialog constructed event.
	OnDialogCreated (see page 11335)	Task Dialog creation event.
	OnDialogDestroyed (see page 11335)	Task dialog destruction event..
	OnExpanded (see page 11335)	Task dialog expanded event.
	OnHyperlinkClicked (see page 11336)	Task dialog hyperlink clicked event.
	OnNavigated (see page 11336)	Task Dialog navigation event.
	OnRadioButtonClicked (see page 11336)	Task Dialog radio button click event.
	OnTimer (see page 11336)	Task dialog timer event.
	OnVerificationClicked (see page 11337)	Task Dialog verification check box event.

61.1.7.6.1 TCustomTaskDialog.OnButtonClicked Event

Button (see page 11328) click event.

Pascal

```
property OnButtonClicked: TTaskDlgClickEvent;
```

C++

```
__property TTaskDlgClickEvent OnButtonClicked;
```

Description

OnButtonClicked is the button click event for Task Dialog.

61.1.7.6.2 TCustomTaskDialog.OnDialogConstructed Event

Dialog constructed event.

Pascal

```
property OnDialogConstructed: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDialogConstructed;
```

Description

OnDialogConstructed is the dialog constructed event.

61.1.7.6.3 TCustomTaskDialog.OnDialogCreated Event

Task Dialog creation event.

Pascal

```
property OnDialogCreated: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDialogCreated;
```

Description

OnDialogCreated is the Task Dialog creation event.

61.1.7.6.4 TCustomTaskDialog.OnDialogDestroyed Event

Task dialog destruction event..

Pascal

```
property OnDialogDestroyed: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDialogDestroyed;
```

Description

OnDialogDestroyed is the Task Dialog destroy event.

61.1.7.6.5 TCustomTaskDialog.OnExpanded Event

Task dialog expanded event.

Pascal

```
property OnExpanded: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExpanded;
```

Description

OnExpanded is the Task Dialog expanded event.

61.1.7.6.6 TCustomTaskDialog.OnHyperlinkClicked Event

Task dialog hyperlink clicked event.

Pascal

```
property OnHyperlinkClicked: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnHyperlinkClicked;
```

Description

OnHyperlinkClicked is the Task Dialog hyperlink clicked event.

61.1.7.6.7 TCustomTaskDialog.OnNavigated Event

Task Dialog navigation event.

Pascal

```
property OnNavigated: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnNavigated;
```

Description

OnNavigated is the Task Dialog navigation event.

61.1.7.6.8 TCustomTaskDialog.OnRadioButtonClicked Event

Task Dialog radio button click event.

Pascal

```
property OnRadioButtonClicked: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnRadioButtonClicked;
```

Description

OnRadioButtonClicked

61.1.7.6.9 TCustomTaskDialog.OnTimer Event

Task dialog timer event.

Pascal

```
property OnTimer: TTaskDlgTimerEvent;
```

C++

```
__property TTaskDlgTimerEvent OnTimer;
```

Description

OnTimer is the Task Dialog timer event.

61.1.7.6.10 TCustomTaskDialog.OnVerificationClicked Event

Task Dialog verification check box event.

Pascal

```
property OnVerificationClicked: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnVerificationClicked;
```

Description

OnVerificationClicked is the Task Dialog verification check box event.

61.1.8 TFavoriteLinkItem Class

Item in a list of favorite links.

Class Hierarchy

Pascal

```
TFavoriteLinkItem = class(TCollectionItem);
```

C++

```
class TFavoriteLinkItem : public TCollectionItem;
```

File

Dialogs

Description

This class represents Items in a list of favorite links.

See Also

TCollectionItem (see page 2755)

TFavoriteLinkItem

FavoriteLinks (see page 11301)

61.1.8.1 TFavoriteLinkItem Members

The following tables list the members exposed by TFavoriteLinkItem.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 11339)	Get displayed name for item.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2757)	Creates and initializes a TCollectionItem (see page 2755) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	Location (see page 11339)	Displayed name for item.

61.1.8.2 TFavoriteLinkItem Methods

The methods of the TFavoriteLinkItem class are listed here.

Protected Methods

	Name	Description
V	GetDisplayName (see page 11339)	Get displayed name for item.

61.1.8.2.1 TFavoriteLinkItem.GetDisplayName Method

Get displayed name for item.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName obtains the name that appears in the file dialog for a favorite item.

See Also

TFavoriteLinkItem (see page 11337)

61.1.8.3 TFavoriteLinkItem Properties

The properties of the TFavoriteLinkItem class are listed here.

Published Properties

	Name	Description
	Location (see page 11339)	Displayed name for item.

61.1.8.3.1 TFavoriteLinkItem.Location Property

Displayed name for item.

Pascal

```
property Location: string;
```

C++

```
__property AnsiString Location;
```

Description

Location contains the name that appears in the file dialog for a favorite item.

See Also

TFavoriteLinkItem (see page 11337)

GetDisplayName (see page 11339)

61.1.9 TFavoriteLinkItems Class

List of favorite links.

Class Hierarchy

Pascal

```
TFavoriteLinkItems = class(TCollection);
```

C++

```
class TFavoriteLinkItems : public TCollection;
```

File

Dialogs

Description

This class represents a list of favorite links.

See Also

TCollection (see page 2731)

TFavoriteLinkItem (see page 11337)

FavoriteLinks (see page 11301)

61.1.9.1 TFavoriteLinkItems Members

The following tables list the members exposed by TFavoriteLinkItems.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.

	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 11342)	Add item to list of favorite links.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 2734)	Creates and initializes a collection.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 11342)	Get an enumerator for favorite items list.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 11343)	List of favorite items.

61.1.9.2 TFavoriteLinkItems Methods

The methods of the TFavoriteLinkItems class are listed here.

Public Methods

	Name	Description
	Add (see page 11342)	Add item to list of favorite links.
	GetEnumerator (see page 11342)	Get an enumerator for favorite items list.

61.1.9.2.1 TFavoriteLinkItems.Add Method

Add item to list of favorite links.

Pascal

```
function Add: TFavoriteLinkItem;
```

C++

```
__fastcall TFavoriteLinkItem Add();
```

Description

Add adds an item to the list of favorite links.

61.1.9.2.2 TFavoriteLinkItems.GetEnumerator Method

Get an enumerator for favorite items list.

Pascal

```
function GetEnumerator: TFavoriteLinkItemsEnumerator;
```

C++

```
__fastcall TFavoriteLinkItemsEnumerator GetEnumerator();
```

Description

GetEnumerator obtains a TFavoriteLinkItemsEnumerator (see page 11343) instance to enumerate the favorite items list.

See Also

TFavoriteLinkItemsEnumerator (see page 11343)

61.1.9.3 TFavoriteLinkItems Properties

The properties of the TFavoriteLinkItems class are listed here.

Public Properties

	Name	Description
	Items (see page 11343)	List of favorite items.

61.1.9.3.1 TFavoriteLinkItems.Items Property

List of favorite items.

Pascal

```
property Items [Index: Integer]: TFavoriteLinkItem;
```

C++

```
__property TFavoriteLinkItem Items[int Index];
```

Description

Items contains the list of favorite items and can be indexed by item number.

See Also

TFavoriteLinkItem (see page 11337)

61.1.10 TFavoriteLinkItemsEnumerator Class

Enumerator for list of favorite items.

Class Hierarchy

Dialogs.TFavoriteLinkItemsEnumerator

Pascal

```
TFavoriteLinkItemsEnumerator = class;
```

C++

```
class TFavoriteLinkItemsEnumerator;
```

File

Dialogs

Description

This class allows you to enumerate the list of favorite items.

See Also

TFavoriteLinkItem (see page 11337)

TFavoriteLinkItems (see page 11340)

61.1.10.1 TFavoriteLinkItemsEnumerator Members

The following tables list the members exposed by TFavoriteLinkItemsEnumerator.

Public Methods

	Name	Description
	Create (see page 11344)	Create a favorite items enumerator.
	GetCurrent (see page 11344)	Returns current enumerated favorite item.
	MoveNext (see page 11345)	Move to next item in favorite items enumeration.

Public Properties

	Name	Description
R	Current (see page 11345)	Current enumerated favorite item.

61.1.10.2 TFavoriteLinkItemsEnumerator.Create Constructor

Create a favorite items enumerator.

Pascal

```
constructor Create(ACollection: TFavoriteLinkItems);
```

C++

```
__fastcall TFavoriteLinkItemsEnumerator(TFavoriteLinkItems ACollection);
```

Description

Create creates and initializes a favorite items enumerator instance for the supplied TFavoriteLinkItems (see page 11340) list.

See Also

TFavoriteLinkItems (see page 11340)

61.1.10.3 TFavoriteLinkItemsEnumerator Methods

The methods of the TFavoriteLinkItemsEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 11344)	Returns current enumerated favorite item.
	MoveNext (see page 11345)	Move to next item in favorite items enumeration.

61.1.10.3.1 TFavoriteLinkItemsEnumerator.GetCurrent Method

Returns current enumerated favorite item.

Pascal

```
function GetCurrent: TFavoriteLinkItem;
```

C++

```
__fastcall TFavoriteLinkItem GetCurrent();
```

Description

GetCurrent obtains the current enumerated favorite item.

See Also

TFavoriteLinkItem (see page 11337)

Current

MoveNext (see page 11345)

61.1.10.3.2 TFavoriteLinkItemsEnumerator.MoveNext Method

Move to next item in favorite items enumeration.

Pascal

```
function MoveNext: Boolean;
```

C++

```
__fastcall Boolean MoveNext();
```

Description

MoveNext goes to the next item in the favorite items enumeration.

See Also

TFavoriteLinkItem (see page 11337)

Current

GetCurrent (see page 11344)

61.1.10.4 TFavoriteLinkItemsEnumerator Properties

The properties of the TFavoriteLinkItemsEnumerator class are listed here.

Public Properties

	Name	Description
R	Current (see page 11345)	Current enumerated favorite item.

61.1.10.4.1 TFavoriteLinkItemsEnumerator.Current Property

Current enumerated favorite item.

Pascal

```
property Current: TFavoriteLinkItem;
```

C++

```
__property TFavoriteLinkItem Current;
```

Description

Current allows you to obtain the current enumerated favorite item.

See Also

TFavoriteLinkItem (see page 11337)

MoveNext (see page 11345)

61.1.11 TFileOpenDialog Class

Class for Vista style file open dialogs.

Class Hierarchy

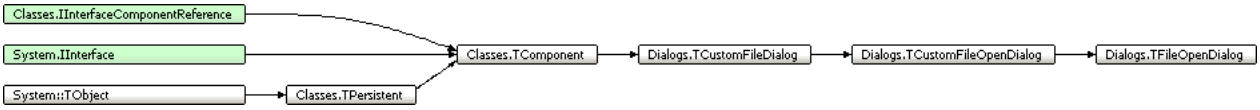

Pascal

```
TFileOpenDialog = class(TCustomFileOpenDialog);
```

C++

```
class TFileOpenDialog : public TCustomFileOpenDialog;
```

File

Dialogs

Description

This class extends TCustomFileOpenDialog (see page 11308). It allows creating Microsoft Windows Vista style file open dialogs CLSID_FileOpenDialog and implements the IFileOpenDialog interface.

See Also

TCustomFileOpenDialog (see page 11308)

61.1.11.1 TFileOpenDialog Members

The following tables list the members exposed by TFileOpenDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CreateFileDialog (see page 11313)	Create (see page 11292) an open file dialog.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FileOkClick (see page 11297)	This is FileOkClick, a member of class TCustomFileDialog.
	FolderChange (see page 11297)	This is FolderChange, a member of class TCustomFileDialog.
	FolderChanging (see page 11297)	This is FolderChanging, a member of class TCustomFileDialog.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.

	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetResults (see page 11313)	Get user's choices in dialog.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	Overwrite (see page 11298)	This is Overwrite, a member of class TCustomFileDialog.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SelectionChange (see page 11298)	This is SelectionChange, a member of class TCustomFileDialog.
V	SelectionChange (see page 11313)	Respond to a selection change.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	ShareViolation (see page 11299)	This is ShareViolation, a member of class TCustomFileDialog.
	TypeChange (see page 11299)	This is TypeChange, a member of class TCustomFileDialog.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnOverwrite (see page 11307)	Occurs just before a file would be overwritten.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 11292)	Creates and initializes an instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 11292)	Destroys an instance and frees the associated resources.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Execute (see page 11296)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dialog (see page 11300)	Pointer to dialog.
R	Files (see page 11302)	List of selected file names.
R	Handle (see page 11303)	Specifies the window handle of the dialog.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	ShellItem (see page 11304)	Current shell item.
R	ShellItems (see page 11304)	Array of current shell items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnExecute (see page 11354)	Occurs when a file dialog gets executed.
	OnFileOkClick (see page 11355)	Occurs when the user clicks the OK button.
	OnFolderChange (see page 11355)	Occurs when a directory is opened or closed from the dialog.
	OnFolderChanging (see page 11355)	Occurs before a directory is opened or closed from the dialog.
	OnSelectionChange (see page 11356)	Occurs when the user changes the selection in the dialog.
	OnShareViolation (see page 11356)	Occurs when sharing violations arise from Open or Save operations.
	OnTypeChange (see page 11356)	Occurs when the user changes the file type selection in the dialog.

Published Properties

	Name	Description
	ClientGuid (see page 11350)	GUID associated with a dialog's persisted state.
	DefaultExtension (see page 11350)	Default extension to be added to file names.
	DefaultFolder (see page 11351)	Default folder if no recently used folder.
	FavoriteLinks (see page 11351)	List of places available for the user to open or save items.
	FileName (see page 11351)	Text in dialog's File name edit box.
	FileNameLabel (see page 11352)	Text of label next to file name edit box.
	FileTypeIndex (see page 11352)	Determines which filter is selected by default when the dialog opens.
	FileTypes (see page 11352)	Determines the file type masks (filters) available in the dialog.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OkButtonLabel (see page 11353)	Text of the Open or Save button.
	Options (see page 11353)	List of options controlling dialog.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Title (see page 11354)	Dialog (see page 11300) title.

Strict protected Methods

	Name	Description
A	CreateFileDialog (see page 11293)	Create (see page 11292) a file dialog.
	DoOnExecute (see page 11294)	Call OnExecute (see page 11305) event method if defined.
	DoOnFileOkClick (see page 11294)	Call OnFileOkClick (see page 11306) event method if defined.
	DoOnFolderChange (see page 11294)	Call OnFolderChange (see page 11306) event method if defined.
	DoOnFolderChanging (see page 11295)	Call OnFolderChanging (see page 11307) event method if defined.
	DoOnOverwrite (see page 11295)	Call OnOverwrite (see page 11307) event method if defined.

	DoOnSelectionChange (see page 11295)	Call OnSelectionChange (see page 11307) event method if defined.
	DoOnShareViolation (see page 11296)	Call OnShareViolation (see page 11308) event method if defined.
	DoOnTypeChange (see page 11296)	Call OnTypeChange (see page 11308) event method if defined.
	GetFileNames (see page 11298)	This is GetFileNames, a member of class TCustomFileDialog.
	GetItemName (see page 11298)	This is GetItemName, a member of class TCustomFileDialog.
	GetResults (see page 11298)	This is GetResults, a member of class TCustomFileDialog.

61.1.11.2 TFileOpenDialog Properties

The properties of the TFileOpenDialog class are listed here.

Published Properties

	Name	Description
	ClientGuid (see page 11350)	GUID associated with a dialog's persisted state.
	DefaultExtension (see page 11350)	Default extension to be added to file names.
	DefaultFolder (see page 11351)	Default folder if no recently used folder.
	FavoriteLinks (see page 11351)	List of places available for the user to open or save items.
	FileName (see page 11351)	Text in dialog's File name edit box.
	FileNameLabel (see page 11352)	Text of label next to file name edit box.
	FileTypeIndex (see page 11352)	Determines which filter is selected by default when the dialog opens.
	FileTypes (see page 11352)	Determines the file type masks (filters) available in the dialog.
	OkButtonLabel (see page 11353)	Text of the Open or Save button.
	Options (see page 11353)	List of options controlling dialog.
	Title (see page 11354)	Dialog (see page 11300) title.

61.1.11.2.1 TFileOpenDialog.ClientGuid Property

GUID associated with a dialog's persisted state.

Pascal

```
property ClientGuid: string;
```

C++

```
__property AnsiString ClientGuid;
```

Description

ClientGuid holds a GUID associated with a dialog's persisted state. Persisted states for a dialog can include such things as its position and size. Having a GUID allows an application to remember these persisted states for different versions of the dialog.

61.1.11.2.2 TFileOpenDialog.DefaultExtension Property

Default extension to be added to file names.

Pascal

```
property DefaultExtension: string;
```

C++

```
__property AnsiString DefaultExtension;
```

Description

DefaultExtension contains the default extension to add to file names.

See Also

DefaultFolder (see page 11351)

FileName (see page 11351)

Files

FileTypes (see page 11352)

61.1.11.2.3 TFileOpenDialog.DefaultFolder Property

Default folder if no recently used folder.

Pascal

```
property DefaultFolder: string;
```

C++

```
__property AnsiString DefaultFolder;
```

Description

DefaultFolder contains the name of the default folder if there is no recently used folder.

See Also

DefaultExtension (see page 11350)

61.1.11.2.4 TFileOpenDialog.FavoriteLinks Property

List of places available for the user to open or save items.

Pascal

```
property FavoriteLinks: TFavoriteLinkItems;
```

C++

```
__property TFavoriteLinkItems FavoriteLinks;
```

Description

FavoriteLinks contains a list of places available for the user to open or save items. This is an instance of TFavoriteLinkItems (see page 11340).

See Also

TFavoriteLinkItems (see page 11340)

61.1.11.2.5 TFileOpenDialog.FileName Property

Text in dialog's File name edit box.

Pascal

```
property FileName: TFileName;
```

C++

```
__property TFileName FileName;
```

Description

FileName contains the text currently entered in the dialog's File name edit box.

See Also

FileNameLabel (see page 11352)

FileTypes (see page 11352)

ShellItem

61.1.11.2.6 TFileOpenDialog.FileNameLabel Property

Text of label next to file name edit box.

Pascal

```
property FileNameLabel: string;
```

C++

```
__property AnsiString FileNameLabel;
```

Description

FileNameLabel holds the text of the label next to the file name edit box in the dialog.

See Also

FileName (see page 11351)

FileTypes (see page 11352)

OkButtonLabel (see page 11353)

61.1.11.2.7 TFileOpenDialog.FileTypeIndex Property

Determines which filter is selected by default when the dialog opens.

Pascal

```
property FileTypeIndex: Cardinal;
```

C++

```
__property unsigned FileTypeIndex;
```

Description

FileTypeIndex determines which of the file types in FileTypes (see page 11352) is selected by default when the dialog opens. Set FileTypeIndex to 1 to choose the first file type in the list as the default, or set FileTypeIndex to 2 to choose the second file type as the default, and so forth. If the value of FileTypeIndex is out or range, the first file type listed in FileTypes (see page 11352) is the default.

See Also

FileTypes (see page 11352)

61.1.11.2.8 TFileOpenDialog.FileTypes Property

Determines the file type masks (filters) available in the dialog.

Pascal

```
property FileTypes: TFileTypeItems;
```

C++

```
__property TFileTypeItems FileTypes;
```

Description

FileTypes is an instance of TFileTypeItems (see page 11372) containing a list of file types, each of which is an instance of TFileTypeItem (see page 11368).

To add file types to a dialog, get the TFileTypeItems (see page 11372) instance from FileTypes and use the TFileTypeItems.Add method as in this sample:

```
with FileOpenDialog1.FileTypes.Add do
begin
  DisplayName := 'All Files';
  FileMask := '*.*';
end;
```

See Also

TFileTypeItem (see page 11368)

TFileTypeItems (see page 11372)

FileTypeIndex (see page 11352)

61.1.11.2.9 TFileOpenDialog.OkButtonLabel Property

Text of the Open or Save button.

Pascal

```
property OkButtonLabel: string;
```

C++

```
__property AnsiString OkButtonLabel;
```

Description

OkButtonLabel contains text of the Open or Save button.

See Also

FileNameLabel (see page 11352)

61.1.11.2.10 TFileOpenDialog.Options Property

List of options controlling dialog.

Pascal

```
property Options: TFileDialogOptions;
```

C++

```
__property TFileDialogOptions Options;
```

Description

Options is a set of TFileDialogOption (see page 11549) options controlling the dialog's behavior.

See Also

TFileDialogOption

61.1.11.2.11 TFileOpenDialog.Title Property

Dialog (see page 11300) title.

Pascal

```
property Title: string;
```

C++

```
__property AnsiString Title;
```

Description

Title contains the dialog's title.

See Also

FileNameLabel (see page 11352)

OkButtonLabel (see page 11353)

61.1.11.3 TFileOpenDialog Events

The events of the TFileOpenDialog class are listed here.

Published Events

	Name	Description
	OnExecute (see page 11354)	Occurs when a file dialog gets executed.
	OnFileOkClick (see page 11355)	Occurs when the user clicks the OK button.
	OnFolderChange (see page 11355)	Occurs when a directory is opened or closed from the dialog.
	OnFolderChanging (see page 11355)	Occurs before a directory is opened or closed from the dialog.
	OnSelectionChange (see page 11356)	Occurs when the user changes the selection in the dialog.
	OnShareViolation (see page 11356)	Occurs when sharing violations arise from Open or Save operations.
	OnTypeChange (see page 11356)	Occurs when the user changes the file type selection in the dialog.

61.1.11.3.1 TFileOpenDialog.OnExecute Event

Occurs when a file dialog gets executed.

Pascal

```
property OnExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExecute;
```

Description

OnExecute occurs when an instance of TFileOpenDialog (see page 11346) gets executed by the Execute (see page 11296) method. Execute (see page 11296) causes the dialog to be displayed. The DoOnExecute (see page 11294) method triggers this event.

See Also

DoOnExecute (see page 11294)

Execute (see page 11296)

61.1.11.3.2 TFileOpenDialog.OnFileOkClick Event

Occurs when the user clicks the OK button.

Pascal

```
property OnFileOkClick: TFileDialogCloseEvent;
```

C++

```
__property TFileDialogCloseEvent OnFileOkClick;
```

Description

OnFileOkClick occurs when whenever the user clicks the OK button. The event occurs just before the dialog returns with a result. The DoOnFileOkClick (see page 11294) method triggers this event.

See Also

DoOnFileOkClick (see page 11294)

61.1.11.3.3 TFileOpenDialog.OnFolderChange Event

Occurs when a directory is opened or closed from the dialog.

Pascal

```
property OnFolderChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnFolderChange;
```

Description

The OnFolderChange event occurs when the user changes the directory whose contents are displayed in the dialog. This can happen when the user double-clicks on a directory, clicks the Up arrow, or uses the list box at the top of the dialog to navigate through the directory structure. The event occurs after the folder changes. The DoOnFolderChange (see page 11294) method triggers this event.

See Also

DoOnFolderChange (see page 11294)

61.1.11.3.4 TFileOpenDialog.OnFolderChanging Event

Occurs before a directory is opened or closed from the dialog.

Pascal

```
property OnFolderChanging: TFileDialogFolderChangingEvent;
```

C++

```
__property TFileDialogFolderChangingEvent OnFolderChanging;
```

Description

OnFileOkClick (see page 11355) occurs before a directory is opened or closed from the dialog. This can happen when the user double-clicks on a directory, clicks the Up arrow, or uses the list box at the top of the dialog to navigate through the directory structure. The DoOnFolderChanging (see page 11295) method triggers this event.

See Also

DoOnFolderChanging (see page 11295)

61.1.11.3.5 TFileOpenDialog.OnSelectionChange Event

Occurs when the user changes the selection in the dialog.

Pascal

```
property OnSelectionChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSelectionChange;
```

Description

OnSelectionChange occurs when the user changes the selection in the dialog. The DoOnSelectionChange (see page 11295) method triggers this event.

See Also

DoOnSelectionChange (see page 11295)

61.1.11.3.6 TFileOpenDialog.OnShareViolation Event

Occurs when sharing violations arise from Open or Save operations.

Pascal

```
property OnShareViolation: TFileDialogShareViolationEvent;
```

C++

```
__property TFileDialogShareViolationEvent OnShareViolation;
```

Description

OnShareViolation occurs when sharing violations arise from Open or Save operations. The DoOnShareViolation (see page 11296) method triggers this event.

See Also

DoOnShareViolation (see page 11296)

61.1.11.3.7 TFileOpenDialog.OnTypeChange Event

Occurs when the user changes the file type selection in the dialog.

Pascal

```
property OnTypeChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnTypeChange;
```

Description

OnTypeChange occurs when the user changes the file type selection in the dialog. The DoOnTypeChange (see page 11296) method triggers this event.

See Also

DoOnTypeChange (see page 11296)

61.1.12 TFileSaveDialog Class

Class for Vista style file save dialogs.

Class Hierarchy

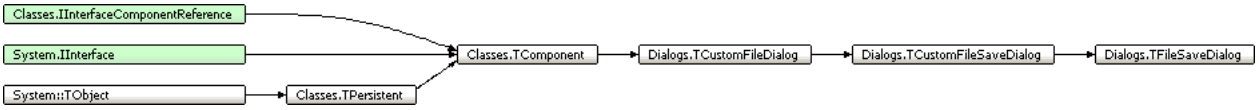

Pascal

```
TFileSaveDialog = class(TCustomFileSaveDialog);
```

C++

```
class TFileSaveDialog : public TCustomFileSaveDialog;
```

File

Dialogs

Description

This class extends TCustomFileSaveDialog (see page 11314). It allows creating Microsoft Windows Vista style file save dialogs CLSID_FileSaveDialog and implements the IFileSaveDialog interface.

See Also

TCustomFileSaveDialog (see page 11314)

61.1.12.1 TFileSaveDialog Members

The following tables list the members exposed by TFileSaveDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CreateFileDialog (see page 11318)	Create (see page 11292) an save file dialog.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FileOkClick (see page 11297)	This is FileOkClick, a member of class TCustomFileDialog.
	FolderChange (see page 11297)	This is FolderChange, a member of class TCustomFileDialog.
	FolderChanging (see page 11297)	This is FolderChanging, a member of class TCustomFileDialog.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.

	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	Overwrite (see page 11298)	This is Overwrite, a member of class TCustomFileDialog.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SelectionChange (see page 11298)	This is SelectionChange, a member of class TCustomFileDialog.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	ShareViolation (see page 11299)	This is ShareViolation, a member of class TCustomFileDialog.
	TypeChange (see page 11299)	This is TypeChange, a member of class TCustomFileDialog.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 11292)	Creates and initializes an instance.

	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 11292)	Destroys an instance and frees the associated resources.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Execute (see page 11296)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dialog (see page 11300)	Pointer to dialog.
R	Files (see page 11302)	List of selected file names.
R	Handle (see page 11303)	Specifies the window handle of the dialog.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	ShellItem (see page 11304)	Current shell item.
R	ShellItems (see page 11304)	Array of current shell items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnExecute (see page 11365)	Occurs when a file dialog gets executed.
	OnFileOkClick (see page 11365)	Occurs when the user clicks the OK button.
	OnFolderChange (see page 11366)	Occurs when a directory is opened or closed from the dialog.
	OnFolderChanging (see page 11366)	Occurs before a directory is opened or closed from the dialog.
	OnOverwrite (see page 11366)	Occurs just before a file would be overwritten.
	OnSelectionChange (see page 11367)	Occurs when the user changes the selection in the dialog.
	OnShareViolation (see page 11367)	Occurs when sharing violations arise from Open or Save operations.
	OnTypeChange (see page 11367)	Occurs when the user changes the file type selection in the dialog.

Published Properties

	Name	Description
	ClientGuid (see page 11361)	GUID associated with a dialog's persisted state.
	DefaultExtension (see page 11361)	Default extension to be added to file names.
	DefaultFolder (see page 11362)	Default folder if no recently used folder.
	FavoriteLinks (see page 11362)	List of places available for the user to open or save items.
	FileName (see page 11362)	Text in dialog's File name edit box.
	FileNameLabel (see page 11363)	Text of label next to file name edit box.
	FileTypeIndex (see page 11363)	Determines which filter is selected by default when the dialog opens.
	FileTypes (see page 11363)	Determines the file type masks (filters) available in the dialog.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OkButtonLabel (see page 11364)	Text of the Open or Save button.
	Options (see page 11364)	List of options controlling dialog.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Title (see page 11364)	Dialog (see page 11300) title.

Strict protected Methods

	Name	Description
A	CreateFileDialog (see page 11293)	Create (see page 11292) a file dialog.
	DoOnExecute (see page 11294)	Call OnExecute (see page 11305) event method if defined.
	DoOnFileOkClick (see page 11294)	Call OnFileOkClick (see page 11306) event method if defined.
	DoOnFolderChange (see page 11294)	Call OnFolderChange (see page 11306) event method if defined.
	DoOnFolderChanging (see page 11295)	Call OnFolderChanging (see page 11307) event method if defined.
	DoOnOverwrite (see page 11295)	Call OnOverwrite (see page 11307) event method if defined.
	DoOnSelectionChange (see page 11295)	Call OnSelectionChange (see page 11307) event method if defined.
	DoOnShareViolation (see page 11296)	Call OnShareViolation (see page 11308) event method if defined.
	DoOnTypeChange (see page 11296)	Call OnTypeChange (see page 11308) event method if defined.
	GetFileNames (see page 11298)	This is GetFileNames, a member of class TCustomFileDialog.

	GetItemName (see page 11298)	This is GetItemName, a member of class TCustomFileDialog.
	GetResults (see page 11298)	This is GetResults, a member of class TCustomFileDialog.

61.1.12.2 TFileSaveDialog Properties

The properties of the TFileSaveDialog class are listed here.

Published Properties

	Name	Description
	ClientGuid (see page 11361)	GUID associated with a dialog's persisted state.
	DefaultExtension (see page 11361)	Default extension to be added to file names.
	DefaultFolder (see page 11362)	Default folder if no recently used folder.
	FavoriteLinks (see page 11362)	List of places available for the user to open or save items.
	FileName (see page 11362)	Text in dialog's File name edit box.
	FileNameLabel (see page 11363)	Text of label next to file name edit box.
	FileTypeIndex (see page 11363)	Determines which filter is selected by default when the dialog opens.
	FileTypes (see page 11363)	Determines the file type masks (filters) available in the dialog.
	OkButtonLabel (see page 11364)	Text of the Open or Save button.
	Options (see page 11364)	List of options controlling dialog.
	Title (see page 11364)	Dialog (see page 11300) title.

61.1.12.2.1 TFileSaveDialog.ClientGuid Property

GUID associated with a dialog's persisted state.

Pascal

```
property ClientGuid: string;
```

C++

```
__property AnsiString ClientGuid;
```

Description

ClientGuid holds a GUID associated with a dialog's persisted state. Persisted states for a dialog can include such things as its position and size. Having a GUID allows an application to remember these persisted states for different versions of the dialog.

61.1.12.2.2 TFileSaveDialog.DefaultExtension Property

Default extension to be added to file names.

Pascal

```
property DefaultExtension: string;
```

C++

```
__property AnsiString DefaultExtension;
```

Description

DefaultExtension contains the default extension to add to file names.

See Also

DefaultFolder (see page 11362)

FileName (see page 11362)

Files

FileTypes (see page 11363)

61.1.12.2.3 TFileSaveDialog.DefaultFolder Property

Default folder if no recently used folder.

Pascal

```
property DefaultFolder: string;
```

C++

```
__property AnsiString DefaultFolder;
```

Description

DefaultFolder contains the name of the default folder if there is no recently used folder.

See Also

DefaultExtension (see page 11361)

61.1.12.2.4 TFileSaveDialog.FavoriteLinks Property

List of places available for the user to open or save items.

Pascal

```
property FavoriteLinks: TFavoriteLinkItems;
```

C++

```
__property TFavoriteLinkItems FavoriteLinks;
```

Description

FavoriteLinks contains a list of places available for the user to open or save items. This is an instance of TFavoriteLinkItems (see page 11340).

See Also

TFavoriteLinkItems (see page 11340)

61.1.12.2.5 TFileSaveDialog.FileName Property

Text in dialog's File name edit box.

Pascal

```
property FileName: TFileName;
```

C++

```
__property TFileName FileName;
```

Description

FileName contains the text currently entered in the dialog's File name edit box.

See Also

FileNameLabel (see page 11363)

FileTypes (see page 11363)

ShellItem

61.1.12.2.6 TFileSaveDialog.FileNameLabel Property

Text of label next to file name edit box.

Pascal

```
property FileNameLabel: string;
```

C++

```
__property AnsiString FileNameLabel;
```

Description

FileNameLabel holds the text of the label next to the file name edit box in the dialog.

See Also

FileName (see page 11362)

FileTypes (see page 11363)

OkButtonLabel (see page 11364)

61.1.12.2.7 TFileSaveDialog.FileTypeIndex Property

Determines which filter is selected by default when the dialog opens.

Pascal

```
property FileTypeIndex: Cardinal;
```

C++

```
__property unsigned FileTypeIndex;
```

Description

FileTypeIndex determines which of the file types in FileTypes (see page 11363) is selected by default when the dialog opens. Set FileTypeIndex to 1 to choose the first file type in the list as the default, or set FileTypeIndex to 2 to choose the second file type as the default, and so forth. If the value of FileTypeIndex is out or range, the first file type listed in FileTypes (see page 11363) is the default.

See Also

FileTypes (see page 11363)

61.1.12.2.8 TFileSaveDialog.FileTypes Property

Determines the file type masks (filters) available in the dialog.

Pascal

```
property FileTypes: TFileTypeItems;
```

C++

```
__property TFileTypeItems FileTypes;
```

Description

FileTypes is an instance of TFileTypeItems (see page 11372) containing a list of file types, each of which is an instance of TFileTypeItem (see page 11368).

To add file types to a dialog, get the TFileTypeItems (see page 11372) instance from FileTypes and use the

TFileTypeItems.Add method as in this sample:

```
with FileOpenDialog1.FileTypes.Add do
begin
  DisplayName := 'All Files';
  FileMask := '*.*';
end;
```

See Also

TFileTypeItem (see page 11368)

TFileTypeItems (see page 11372)

FileTypeIndex (see page 11363)

61.1.12.2.9 TFileSaveDialog.OkButtonLabel Property

Text of the Open or Save button.

Pascal

```
property OkButtonLabel: string;
```

C++

```
__property AnsiString OkButtonLabel;
```

Description

OkButtonLabel contains text of the Open or Save button.

See Also

FileNameLabel (see page 11363)

61.1.12.2.10 TFileSaveDialog.Options Property

List of options controlling dialog.

Pascal

```
property Options: TFileDialogOptions;
```

C++

```
__property TFileDialogOptions Options;
```

Description

Options is a set of TFileDialogOption (see page 11549) options controlling the dialog's behavior.

See Also

TFileDialogOption

61.1.12.2.11 TFileSaveDialog.Title Property

Dialog (see page 11300) title.

Pascal

```
property Title: string;
```

C++

```
__property AnsiString Title;
```

Description

Title contains the dialog's title.

See Also

FileNameLabel (see page 11363)

OkButtonLabel (see page 11364)

61.1.12.3 TFileSaveDialog Events

The events of the TFileSaveDialog class are listed here.

Published Events

	Name	Description
	OnExecute (see page 11365)	Occurs when a file dialog gets executed.
	OnFileOkClick (see page 11365)	Occurs when the user clicks the OK button.
	OnFolderChange (see page 11366)	Occurs when a directory is opened or closed from the dialog.
	OnFolderChanging (see page 11366)	Occurs before a directory is opened or closed from the dialog.
	OnOverwrite (see page 11366)	Occurs just before a file would be overwritten.
	OnSelectionChange (see page 11367)	Occurs when the user changes the selection in the dialog.
	OnShareViolation (see page 11367)	Occurs when sharing violations arise from Open or Save operations.
	OnTypeChange (see page 11367)	Occurs when the user changes the file type selection in the dialog.

61.1.12.3.1 TFileSaveDialog.OnExecute Event

Occurs when a file dialog gets executed.

Pascal

```
property OnExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExecute;
```

Description

OnExecute occurs when an instance of TFileSaveDialog (see page 11357) gets executed by the Execute (see page 11296) method. Execute (see page 11296) causes the dialog to be displayed. The DoOnExecute (see page 11294) method triggers this event.

See Also

DoOnExecute (see page 11294)

Execute (see page 11296)

61.1.12.3.2 TFileSaveDialog.OnFileOkClick Event

Occurs when the user clicks the OK button.

Pascal

```
property OnFileOkClick: TFileDialogCloseEvent;
```

C++

```
__property TFileDialogCloseEvent OnFileOkClick;
```

Description

OnFileOkClick occurs when whenever the user clicks the OK button. The event occurs just before the dialog returns with a result. The DoOnFileOkClick (see page 11294) method triggers this event.

See Also

DoOnFileOkClick (see page 11294)

61.1.12.3.3 TFileSaveDialog.OnFolderChange Event

Occurs when a directory is opened or closed from the dialog.

Pascal

```
property OnFolderChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnFolderChange;
```

Description

The OnFolderChange event occurs when the user changes the directory whose contents are displayed in the dialog. This can happen when the user double-clicks on a directory, clicks the Up arrow, or uses the list box at the top of the dialog to navigate through the directory structure. The event occurs after the folder changes. The DoOnFolderChange (see page 11294) method triggers this event.

See Also

DoOnFolderChange (see page 11294)

61.1.12.3.4 TFileSaveDialog.OnFolderChanging Event

Occurs before a directory is opened or closed from the dialog.

Pascal

```
property OnFolderChanging: TFileDialogFolderChangingEvent;
```

C++

```
__property TFileDialogFolderChangingEvent OnFolderChanging;
```

Description

OnFileOkClick (see page 11365) occurs before a directory is opened or closed from the dialog. This can happen when the user double-clicks on a directory, clicks the Up arrow, or uses the list box at the top of the dialog to navigate through the directory structure. The DoOnFolderChanging (see page 11295) method triggers this event.

See Also

DoOnFolderChanging (see page 11295)

61.1.12.3.5 TFileSaveDialog.OnOverwrite Event

Occurs just before a file would be overwritten.

Pascal

```
property OnOverwrite: TFileDialogOverwriteEvent;
```

C++

```
__property TFileDialogOverwriteEvent OnOverwrite;
```

Description

OnOverwrite occurs just before a file would be overwritten, so this applies only to save dialogs. The DoOnOverwrite (see page 11295) method triggers this event.

See Also

DoOnOverwrite (see page 11295)

61.1.12.3.6 TFileSaveDialog.OnSelectionChange Event

Occurs when the user changes the selection in the dialog.

Pascal

```
property OnSelectionChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSelectionChange;
```

Description

OnSelectionChange occurs when the user changes the selection in the dialog. The DoOnSelectionChange (see page 11295) method triggers this event.

See Also

DoOnSelectionChange (see page 11295)

61.1.12.3.7 TFileSaveDialog.OnShareViolation Event

Occurs when sharing violations arise from Open or Save operations.

Pascal

```
property OnShareViolation: TFileDialogShareViolationEvent;
```

C++

```
__property TFileDialogShareViolationEvent OnShareViolation;
```

Description

OnShareViolation occurs when sharing violations arise from Open or Save operations. The DoOnShareViolation (see page 11296) method triggers this event.

See Also

DoOnShareViolation (see page 11296)

61.1.12.3.8 TFileSaveDialog.OnTypeChange Event

Occurs when the user changes the file type selection in the dialog.

Pascal

```
property OnTypeChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnTypeChange;
```

Description

OnTypeChange occurs when the user changes the file type selection in the dialog. The DoOnTypeChange (see page 11296) method triggers this event.

See Also

DoOnTypeChange (see page 11296)

61.1.13 TFileTypeItem Class

File type specification.

Class Hierarchy

Pascal

```
TFileTypeItem = class(TCollectionItem);
```

C++

```
class TFileTypeItem : public TCollectionItem;
```

File

Dialogs

Description

The class TFileTypeItem represents a type of file that a dialog handles.

A file type includes both a name describing the type of file (DisplayName (see page 11371) property) and a mask typically describing the file's extension (FileMask (see page 11372) property). For instance, a name might be “PDF files” and the filter “*.pdf”.

The property TCustomFileDialog.FileTypes is an instance of TFileTypeItems (see page 11372) containing a list of file types, each of which is an instance of TFileTypeItem

See Also

TCustomFileDialog (see page 11288)

TFileTypeItems (see page 11372)

FileTypes (see page 11303)

FileTypeIndex (see page 11302)

61.1.13.1 TFileTypeItem Members

The following tables list the members exposed by TFileTypeItem.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	GetDisplayName (see page 11370)	Get display name for file type.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11370)	Creates and initializes an instance.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 11370)	Destroys an instance and frees the associated resources.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
R	DisplayNameWStr (see page 11371)	Display name in Unicode for file type.
R	FileMaskWStr (see page 11372)	Mask (see page 15797) in Unicode for file type.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	DisplayName (see page 11371)	Display name for file type.

	FileMask (see page 11372)	Mask (see page 15797) for file type.

61.1.13.2 TFileTypeItem.Create Constructor

Creates and initializes an instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TFileTypeItem(TCollection Collection);
```

Description

This method creates and initializes an instance of TFileTypeItem (see page 11368).

See Also

Destroy (see page 11370)

61.1.13.3 TFileTypeItem.Destroy Destructor

Destroys an instance and frees the associated resources.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TFileTypeItem();
```

Description

This method destroys an instance of TFileTypeItem (see page 11368) and frees the associated resources.

See Also

Create (see page 11370)

61.1.13.4 TFileTypeItem Methods

The methods of the TFileTypeItem class are listed here.

Protected Methods

	Name	Description
	GetDisplayName (see page 11370)	Get display name for file type.

61.1.13.4.1 TFileTypeItem.GetDisplayName Method

Get display name for file type.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName returns the name that appears in the file types list for the file type.

See Also

DisplayName (see page 11371)

DisplayNameWStr (see page 11371)

61.1.13.5 TFileTypeItem Properties

The properties of the TFileTypeItem class are listed here.

Public Properties

	Name	Description
R	DisplayNameWStr (see page 11371)	Display name in Unicode for file type.
R	FileMaskWStr (see page 11372)	Mask (see page 15797) in Unicode for file type.

Published Properties

	Name	Description
	DisplayName (see page 11371)	Display name for file type.
	FileMask (see page 11372)	Mask (see page 15797) for file type.

61.1.13.5.1 TFileTypeItem.DisplayName Property

Display name for file type.

Pascal

```
property DisplayName: string;
```

C++

```
__property AnsiString DisplayName;
```

Description

DisplayName contains the name that appears in the file types list for the file type. For instance, a name could be “Text files”.

See Also

DisplayNameWStr (see page 11371)

61.1.13.5.2 TFileTypeItem.DisplayNameWStr Property

Display name in Unicode for file type.

Pascal

```
property DisplayNameWStr: LPCWSTR;
```

C++

```
__property LPCWSTR DisplayNameWStr;
```

Description

DisplayName (see page 11371) contains the file type's name that appears in the file types list in Unicode. For instance, a name could be “Text files”.

See Also

DisplayName (see page 11371)

61.1.13.5.3 TFileTypeItem.FileMask Property

Mask (see page 15797) for file type.

Pascal

```
property FileMask: string;
```

C++

```
__property AnsiString FileMask;
```

Description

FileMask contains the mask for the file type. For instance, a mask for text files could be “*.txt”.

See Also

FileMaskWStr (see page 11372)

61.1.13.5.4 TFileTypeItem.FileMaskWStr Property

Mask (see page 15797) in Unicode for file type.

Pascal

```
property FileMaskWStr: LPCWSTR;
```

C++

```
__property LPCWSTR FileMaskWStr;
```

Description

FileMask (see page 11372) contains the file type's mask in Unicode. For instance, a mask for text files could be “*.txt”.

See Also

FileMask (see page 11372)

61.1.14 TFileTypeItems Class

Collection of file type specifications.

Class Hierarchy

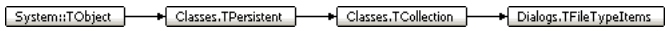

Pascal

```
TFileTypeItems = class(TCollection);
```

C++

```
class TFileTypeItems : public TCollection;
```

File

Dialogs

Description

The class TFileTypeItems represents a collection of file types for a dialog. A file type includes both a name describing the type of file and a mask typically describing the file's extension. For instance, a name might be “PDF files” and the filter “*.pdf”.

The property TCustomFileDialog.FileTypes is an instance of TFileTypeItems containing a list of file types, each of which is an instance of TFileTypeItem (see page 11368)

To add file types to a dialog, get the TFileTypeItems instance from TCustomFileDialog.FileTypes and use the TFileTypeItems.Add (see page 11375) method as in this sample:

```
with FileOpenDialog1.FileTypes.Add do
begin
  DisplayName := ‘All Files’;
  FileMask := ‘*.*’;
end;
```

See Also

TCustomFileDialog (see page 11288)

TFileTypeItem (see page 11368)

DisplayName (see page 11371)

FileMask (see page 11372)

FileTypes (see page 11303)

FileTypeIndex (see page 11302)

61.1.14.1 TFileTypeItems Members

The following tables list the members exposed by TFileTypeItems.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.

R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 11375)	Add a file type to the collection.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 2734)	Creates and initializes a collection.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FilterSpecArray (see page 11375)	Get an array of file types.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 11376)	List of file types.

61.1.14.2 TFileTypeItems Methods

The methods of the TFileTypeItems class are listed here.

Public Methods

	Name	Description
	Add (see page 11375)	Add a file type to the collection.
	FilterSpecArray (see page 11375)	Get an array of file types.

61.1.14.2.1 TFileTypeItems.Add Method

Add a file type to the collection.

Pascal

```
function Add: TFileTypeItem;
```

C++

```
__fastcall TFileTypeItem Add();
```

Description

Add adds a file type to the file types collection. After you add the file type, you need to set its properties as in this sample:

```
with FileOpenDialog1.FileTypes.Add do
begin
  DisplayName := 'All Files';
  FileMask := '*.*';
end;
```

See Also

TFileTypeItem (see page 11368)

DisplayName (see page 11371)

FileMask (see page 11372)

61.1.14.2.2 TFileTypeItems.FilterSpecArray Method

Get an array of file types.

Pascal

```
function FilterSpecArray: TComdlgFilterSpecArray;
```

C++

```
__fastcall TComdlgFilterSpecArray FilterSpecArray();
```

Description

FilterSpecArray returns a TComdlgFilterSpecArray record with file type information. TComdlgFilterSpecArray is the same as the COMDLG_FILTERSPEC struct in the Microsoft IFileDialog interface for common item dialogs.

61.1.14.3 TFileTypeItems Properties

The properties of the TFileTypeItems class are listed here.

Public Properties

	Name	Description
	Items (see page 11376)	List of file types.

61.1.14.3.1 TFileTypeItems.Items Property

List of file types.

Pascal

```
property Items [Index: Integer]: TFileTypeItem;
```

C++

```
__property TFileTypeItem Items[int Index];
```

Description

Items contains a list of the dialog's file types and can be indexed by a cardinal integer.

61.1.15 TFindDialog Class

TFindDialog displays a Find (see page 11383) dialog that allows users to search for text in a file.

Class Hierarchy

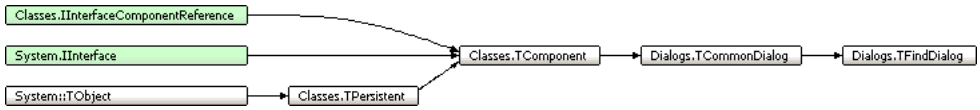

Pascal

```
TFindDialog = class(TCommonDialog);
```

C++

```
class TFindDialog : public TCommonDialog;
```

File

Dialogs

Description

The TFindDialog component displays a modeless Windows dialog box that prompts the user for a search string. The dialog does not appear at runtime until it is activated by a call to the Execute (see page 11381) method.

See Also

TReplaceDialog (see page 11482)

61.1.15.1 TFindDialog Members

The following tables list the members exposed by TFindDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoClose (see page 11282)	Generates an OnClose (see page 11287) event.
	DoShow (see page 11283)	Generates an OnShow (see page 11288) event.
	Find (see page 11383)	This is Find, a member of class TFindDialog.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	MessageHook (see page 11383)	Provides special Windows message processing for the dialog.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	Replace (see page 11384)	This is Replace, a member of class TFindDialog.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	TaskModalDialog (see page 11284)	Ensures that Windows messages are properly routed when the dialog executes.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WndProc (see page 11285)	Responds to Windows messages sent to the dialog.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Template (see page 11287)	Points to a custom template for the dialog.
	TemplateModule (see page 11287)	This is TemplateModule, a member of class TCommonDialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CloseDialog (see page 11381)	Closes the Find (see page 11383) dialog box.
V	Create (see page 11380)	Creates and initializes a TFindDialog (see page 11376) instance.
V	DefaultHandler (see page 11282)	Sends unhandled messages to Windows for processing.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 11380)	Destroys the TFindDialog (see page 11376) and frees its memory.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Execute (see page 11381)	Displays the Find (see page 11383) dialog.
V	Execute (see page 11283)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Handle (see page 11286)	Specifies the window handle of the dialog.
	Left (see page 11386)	Determines the X coordinate (horizontal position) of the Find (see page 11383) dialog's upper left corner.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Position (see page 11390)	Determines the position of the Find (see page 11383) dialog on the screen.
	Top (see page 11392)	Determines the Y coordinate (vertical position) of the Find (see page 11383) dialog's upper left corner.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnClose (see page 11287)	Occurs when the dialog closes.
	OnFind (see page 11392)	Occurs when the user clicks the Find (see page 11383) Next button.
	OnShow (see page 11288)	Occurs when the dialog opens.

Published Properties

	Name	Description
	Ctl3D (see page 11285)	Determines whether the dialog has a three-dimensional look.
	FindText (see page 11384)	Contains the search string.
	HelpContext (see page 11286)	Specifies the context number for online Help.

	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 11386)	Determines the appearance and behavior of the Find (see page 11383) dialog.
	Tag (see page 2808)	Stores an integer value as part of a component.

61.1.15.2 TFindDialog.Create Constructor

Creates and initializes a TFindDialog (see page 11376) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TFindDialog(TComponent * AOwner);
```

Description

The Create method generates a TFindDialog (see page 11376) instance, but the new dialog does not appear on the form at runtime until the Execute (see page 11381) method is called.

AOwner is the component that is responsible for freeing the dialog. It becomes the value of the Owner property.

See Also

Owner (see page 2807)

Execute (see page 11381)

Destroy (see page 11380)

61.1.15.3 TFindDialog.Destroy Destructor

Destroys the TFindDialog (see page 11376) and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TFindDialog();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free checks that the Find (see page 11383) dialog reference is not nil before calling Destroy.

See Also

Free (see page 22300)

Create (see page 11380)

61.1.15.4 TFindDialog Methods

The methods of the TFindDialog class are listed here.

Protected Methods

	Name	Description
	Find (see page 11383)	This is Find, a member of class TFindDialog.

	MessageHook (see page 11383)	Provides special Windows message processing for the dialog.
	Replace (see page 11384)	This is Replace, a member of class TFindDialog.

Public Methods

	Name	Description
	CloseDialog (see page 11381)	Closes the Find (see page 11383) dialog box.
	Execute (see page 11381)	Displays the Find (see page 11383) dialog.

61.1.15.4.1 TFindDialog.CloseDialog Method

Closes the Find (see page 11383) dialog box.

Pascal

```
procedure CloseDialog;
```

C++

```
__fastcall CloseDialog();
```

Description

CloseDialog closes the dialog box and frees its window handle, but leaves most properties of the TFindDialog (see page 11376) (including FindText (see page 11384)) unchanged.

61.1.15.4.2 TFindDialog.Execute Method

Displays the Find (see page 11383) dialog.

Pascal

```
function Execute(ParentWnd: HWND): Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute(HWND ParentWnd);
```

Description

Execute opens the Find (see page 11383) dialog, returning true when a window handle is successfully allocated. If the dialog is already open when Execute is called, Execute brings the dialog to the top, activates its parent form, and returns true.

C++ Examples:

```
/*
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FindDialog1->Position = Point(RichEdit1->Left + RichEdit1->Width, RichEdit1->Top);
  FindDialog1->Execute();
}

void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  int FoundAt, StartPos, ToEnd;
  TSearchTypes mySearchTypes;
  // begin the search after the current selection
  // if there is one
  // otherwise, begin at the start of the text
```

```
  if (FindDialog1->Options.Contains(frMatchCase))
    mySearchTypes << stMatchCase;
  if (FindDialog1->Options.Contains(frWholeWord))
    mySearchTypes << stWholeWord;
  if (RichEdit1->SelLength)
    StartPos = RichEdit1->SelStart + RichEdit1->SelLength;
  else
    StartPos = 0;
  // ToEnd is the length from StartPos
  // to the end of the text in the rich edit control
  ToEnd = RichEdit1->Text.Length() - StartPos;
  FoundAt = RichEdit1->FindText(FindDialog1->FindText, StartPos, ToEnd, mySearchTypes);
  if (FoundAt != -1)
  {
    RichEdit1->SetFocus();
    RichEdit1->SelStart = FoundAt;
    RichEdit1->SelLength = FindDialog1->FindText.Length();
  }
  else Beep();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("../Overview.RTF");  // starts in Debug
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FindDialog1.Position :=
    Point(RichEdit1.Left + RichEdit1.Width, RichEdit1.Top);
  FindDialog1.Execute;
end;

procedure TForm1.FindDialog1Find(Sender: TObject);
var
  FoundAt: LongInt;
  StartPos, ToEnd: Integer;
  mySearchTypes : TSearchTypes;
  myFindOptions : TFindOptions;
begin
  with RichEdit1 do
  begin
    if frMatchCase in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stMatchCase];
    if frWholeWord in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stWholeWord];
    { begin the search after the current selection if there is one }
    { otherwise, begin at the start of the text }
    if SelLength <> 0 then
      StartPos := SelStart + SelLength
    else
      StartPos := 0;
    { ToEnd is the length from StartPos to the end of the
      text in the rich edit control }
    ToEnd := Length(Text) - StartPos;
```

```
      FoundAt :=
        FindText(FindDialog1.FindText, StartPos, ToEnd, mySearchTypes);
      if FoundAt <> -1 then
      begin
        SetFocus;
        SelStart := FoundAt;
        SelLength := Length(FindDialog1.FindText);
      end
      else Beep;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const Path = 'OverView.RTF';
  begin
    RichEdit1.PlainText := False;
    RichEdit1.Lines.LoadFromFile(Path);
    RichEdit1.ScrollBars := ssVertical;
  end;
```

See Also

Create (see page 11380)

61.1.15.4.3 TFindDialog.Find Method

Pascal

```
procedure Find; dynamic;
```

C++

```
__fastcall Find();
```

Description

This is Find, a member of class TFindDialog.

61.1.15.4.4 TFindDialog.MessageHook Method

Provides special Windows message processing for the dialog.

Pascal

```
function MessageHook(var Msg: TMessage): Boolean; override;
```

C++

```
virtual __fastcall Boolean MessageHook(TMessage Msg);
```

Description

MessageHook intercepts messages sent to the dialog window before the dialog's window procedure gets them. When MessageHook handles a message, it returns true to indicate that the message should not be sent on for further processing. If the window procedure should handle the message, MessageHook returns false.

Override MessageHook to provide extra message processing. For example, if the Template property substitutes a dialog template with additional controls, use MessageHook to handle messages from those dialog controls.

In TCommonDialog, MessageHook handles Help messages by calling the application's HelpContext method if the HelpContext property is defined. Descendants of TCommonDialog override MessageHook to add additional message processing.

See Also

Template (see page 11287)

HelpContext (see page 13679)

HelpContext (see page 11286)

61.1.15.4.5 TFindDialog.Replace Method

Pascal

```
procedure Replace; dynamic;
```

C++

```
__fastcall Replace();
```

Description

This is Replace, a member of class TFindDialog.

61.1.15.5 TFindDialog Properties

The properties of the TFindDialog class are listed here.

Public Properties

	Name	Description
	Left (see page 11386)	Determines the X coordinate (horizontal position) of the Find (see page 11383) dialog's upper left corner.
	Position (see page 11390)	Determines the position of the Find (see page 11383) dialog on the screen.
	Top (see page 11392)	Determines the Y coordinate (vertical position) of the Find (see page 11383) dialog's upper left corner.

Published Properties

	Name	Description
	FindText (see page 11384)	Contains the search string.
	Options (see page 11386)	Determines the appearance and behavior of the Find (see page 11383) dialog.

61.1.15.5.1 TFindDialog.FindText Property

Contains the search string.

Pascal

```
property FindText: string;
```

C++

```
__property AnsiString FindText;
```

Description

FindText contains the text string that the user wants to search for. When the user clicks the Find (see page 11383) Next button, the text in the dialog's edit box is automatically assigned to FindText. To make a default text string appear in the edit box when the dialog opens, assign a value to FindText in the Object Inspector or in program code.

C++ Examples:

```
/*
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
```

```
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FindDialog1->Position = Point(RichEdit1->Left + RichEdit1->Width, RichEdit1->Top);
  FindDialog1->Execute();
}

void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  int FoundAt, StartPos, ToEnd;
  TSearchTypes mySearchTypes;
  // begin the search after the current selection
  // if there is one
  // otherwise, begin at the start of the text
  if (FindDialog1->Options.Contains(frMatchCase))
    mySearchTypes << stMatchCase;
  if (FindDialog1->Options.Contains(frWholeWord))
    mySearchTypes << stWholeWord;
  if (RichEdit1->SelLength)
    StartPos = RichEdit1->SelStart + RichEdit1->SelLength;
  else
    StartPos = 0;
  // ToEnd is the length from StartPos
  // to the end of the text in the rich edit control
  ToEnd = RichEdit1->Text.Length() - StartPos;
  FoundAt = RichEdit1->FindText(FindDialog1->FindText, StartPos, ToEnd, mySearchTypes);
  if (FoundAt != -1)
  {
    RichEdit1->SetFocus();
    RichEdit1->SelStart = FoundAt;
    RichEdit1->SelLength = FindDialog1->FindText.Length();
  }
  else Beep();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("../Overview.RTF");  // starts in Debug
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FindDialog1.Position :=
    Point(RichEdit1.Left + RichEdit1.Width, RichEdit1.Top);
  FindDialog1.Execute;
end;

procedure TForm1.FindDialog1Find(Sender: TObject);
var
  FoundAt: LongInt;
  StartPos, ToEnd: Integer;
  mySearchTypes : TSearchTypes;
  myFindOptions : TFindOptions;
begin
  with RichEdit1 do
```

```
    begin
      if frMatchCase in FindDialog1.Options then
         mySearchTypes := mySearchTypes + [stMatchCase];
      if frWholeWord in FindDialog1.Options then
         mySearchTypes := mySearchTypes + [stWholeWord];
      { begin the search after the current selection if there is one }
      { otherwise, begin at the start of the text }
      if SelLength <> 0 then
        StartPos := SelStart + SelLength
      else
        StartPos := 0;
      { ToEnd is the length from StartPos to the end of the
        text in the rich edit control }
      ToEnd := Length(Text) - StartPos;
      FoundAt :=
        FindText(FindDialog1.FindText, StartPos, ToEnd, mySearchTypes);
      if FoundAt <> -1 then
      begin
        SetFocus;
        SelStart := FoundAt;
        SelLength := Length(FindDialog1.FindText);
      end
      else Beep;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const Path = 'OverView.RTF';
  begin
    RichEdit1.PlainText := False;
    RichEdit1.Lines.LoadFromFile(Path);
    RichEdit1.ScrollBars := ssVertical;
  end;
```

61.1.15.5.2 TFindDialog.Left Property

Determines the X coordinate (horizontal position) of the Find (see page 11383) dialog's upper left corner.

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

Left holds the same value as the first integer in Position (see page 11390). This property is maintained for compatibility with older applications.

See Also

Position (see page 11390)

Top (see page 11392)

61.1.15.5.3 TFindDialog.Options Property

Determines the appearance and behavior of the Find (see page 11383) dialog.

Pascal

```
property Options: TFindOptions;
```

C++

```
__property TFindOptions Options;
```

Description

Use Options to

Determine which controls in the dialog are disabled (grayed).

Determine which controls do not appear in the dialog.

Specify settings for the search direction, case-sensitivity, and "whole word" attributes of the search.

By default, all options except frDown are off.

C++ Examples:

```
/*
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FindDialog1->Position = Point(RichEdit1->Left + RichEdit1->Width, RichEdit1->Top);
  FindDialog1->Execute();
}

void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  int FoundAt, StartPos, ToEnd;
  TSearchTypes mySearchTypes;
  // begin the search after the current selection
  // if there is one
  // otherwise, begin at the start of the text
  if (FindDialog1->Options.Contains(frMatchCase))
    mySearchTypes << stMatchCase;
  if (FindDialog1->Options.Contains(frWholeWord))
    mySearchTypes << stWholeWord;
  if (RichEdit1->SelLength)
    StartPos = RichEdit1->SelStart + RichEdit1->SelLength;
  else
    StartPos = 0;
  // ToEnd is the length from StartPos
  // to the end of the text in the rich edit control
  ToEnd = RichEdit1->Text.Length() - StartPos;
  FoundAt = RichEdit1->FindText(FindDialog1->FindText, StartPos, ToEnd, mySearchTypes);
  if (FoundAt != -1)
  {
    RichEdit1->SetFocus();
    RichEdit1->SelStart = FoundAt;
    RichEdit1->SelLength = FindDialog1->FindText.Length();
  }
  else Beep();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("../Overview.RTF");  // starts in Debug
  RichEdit1->ScrollBars = ssVertical;
}
/*
```

```
The following code calls the user-defined routine DoReplace
if the Replace button was clicked, or calls the user-defined
routine DoReplaceAll if the Replace All button was clicked.
Place a TReplaceDialog, a TButton and a TRichEdit in the form.
*/
void __fastcall TForm1::ReplaceDialog1Replace(TObject *Sender)
{
  TReplaceDialog *dlg = (TReplaceDialog *)Sender;
  if (dlg->Options.Contains(frReplace))
    DoReplace(dlg->FindText, dlg->ReplaceText);
  else if (dlg->Options.Contains(frReplaceAll))
    DoReplaceAll(dlg->FindText, dlg->ReplaceText);
}

void __fastcall TForm1::DoReplace(AnsiString FindText, AnsiString ReplaceText)
{
  ShowMessage(
    "DoReplace: FindText: " + ReplaceDialog1->FindText +
    " ReplaceText: " + ReplaceDialog1->ReplaceText);
};

void __fastcall TForm1::DoReplaceAll(AnsiString FindText, AnsiString ReplaceText)
{
  ShowMessage(
    "DoReplaceAll: FindText: " + ReplaceDialog1->FindText +
    " ReplaceText: " + ReplaceDialog1->ReplaceText);
};
```

Delphi Examples:

```
{
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FindDialog1.Position :=
    Point(RichEdit1.Left + RichEdit1.Width, RichEdit1.Top);
  FindDialog1.Execute;
end;

procedure TForm1.FindDialog1Find(Sender: TObject);
var
  FoundAt: LongInt;
  StartPos, ToEnd: Integer;
  mySearchTypes : TSearchTypes;
  myFindOptions : TFindOptions;
begin
  with RichEdit1 do
  begin
    if frMatchCase in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stMatchCase];
    if frWholeWord in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stWholeWord];
    { begin the search after the current selection if there is one }
    { otherwise, begin at the start of the text }
    if SelLength <> 0 then
      StartPos := SelStart + SelLength
    else
      StartPos := 0;
    { ToEnd is the length from StartPos to the end of the
      text in the rich edit control }
    ToEnd := Length(Text) - StartPos;
```

```
      FoundAt :=
        FindText(FindDialog1.FindText, StartPos, ToEnd, mySearchTypes);
      if FoundAt <> -1 then
      begin
        SetFocus;
        SelStart := FoundAt;
        SelLength := Length(FindDialog1.FindText);
      end
      else Beep;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const Path = 'OverView.RTF';
  begin
    RichEdit1.PlainText := False;
    RichEdit1.Lines.LoadFromFile(Path);
    RichEdit1.ScrollBars := ssVertical;
  end;
  {
  The following code calls the user-defined routine DoReplace
  if the Replace button was clicked, or calls the user-defined
  routine DoReplaceAll if the Replace All button was clicked.
  Place a TReplaceDialog, a TButton and a TRichEdit in the form.
  }
  procedure TForm1.ReplaceDialog1Replace(Sender: TObject);
  // var FindText, ReplaceText: String;
  begin
    with Sender as TReplaceDialog do
    begin
    if frReplace in Options then
      DoReplace(ReplaceDialog1.FindText, ReplaceDialog1.ReplaceText)
    else if frReplaceAll in Options then
      DoReplaceAll(ReplaceDialog1.FindText, ReplaceDialog1.ReplaceText);
    end;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    ReplaceDialog1.Execute;
  end;

  procedure TForm1.DoReplace(FindText, ReplaceText: String);
  begin
    ShowMessage(
      'DoReplace: FindText: ' + ReplaceDialog1.FindText +
      ' ReplaceText: ' + ReplaceDialog1.ReplaceText);
  end;

  procedure TForm1.DoReplaceAll(FindText, ReplaceText: String);
  begin
    ShowMessage(
      'DoReplaceAll: FindText: ' + ReplaceDialog1.FindText +
      ' ReplaceText: ' + ReplaceDialog1.ReplaceText);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const Path = 'OverView.RTF';
  begin
    RichEdit1.PlainText := False;
    RichEdit1.Lines.LoadFromFile(Path);
    RichEdit1.ScrollBars := ssVertical;
  end;
```

See Also

FindText (see page 11384)

TReplaceDialog (see page 11482)

OnReplace (see page 11488)

ReplaceText (see page 11487)

61.1.15.5.4 TFindDialog.Position Property

Determines the position of the Find (see page 11383) dialog on the screen.

Pascal

```
property Position: TPoint;
```

C++

```
__property TPoint Position;
```

Description

Position gives the pixel coordinates of the dialog's upper left corner.

C++ Examples:

```
/*
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FindDialog1->Position = Point(RichEdit1->Left + RichEdit1->Width, RichEdit1->Top);
  FindDialog1->Execute();
}

void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  int FoundAt, StartPos, ToEnd;
  TSearchTypes mySearchTypes;
  // begin the search after the current selection
  // if there is one
  // otherwise, begin at the start of the text
  if (FindDialog1->Options.Contains(frMatchCase))
    mySearchTypes << stMatchCase;
  if (FindDialog1->Options.Contains(frWholeWord))
    mySearchTypes << stWholeWord;
  if (RichEdit1->SelLength)
    StartPos = RichEdit1->SelStart + RichEdit1->SelLength;
  else
    StartPos = 0;
  // ToEnd is the length from StartPos
  // to the end of the text in the rich edit control
  ToEnd = RichEdit1->Text.Length() - StartPos;
  FoundAt = RichEdit1->FindText(FindDialog1->FindText, StartPos, ToEnd, mySearchTypes);
  if (FoundAt != -1)
  {
    RichEdit1->SetFocus();
    RichEdit1->SelStart = FoundAt;
    RichEdit1->SelLength = FindDialog1->FindText.Length();
  }
  else Beep();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("../Overview.RTF");  // starts in Debug
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FindDialog1.Position :=
    Point(RichEdit1.Left + RichEdit1.Width, RichEdit1.Top);
  FindDialog1.Execute;
end;

procedure TForm1.FindDialog1Find(Sender: TObject);
var
  FoundAt: LongInt;
  StartPos, ToEnd: Integer;
  mySearchTypes : TSearchTypes;
  myFindOptions : TFindOptions;
begin
  with RichEdit1 do
  begin
    if frMatchCase in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stMatchCase];
    if frWholeWord in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stWholeWord];
    { begin the search after the current selection if there is one }
    { otherwise, begin at the start of the text }
    if SelLength <> 0 then
      StartPos := SelStart + SelLength
    else
      StartPos := 0;
    { ToEnd is the length from StartPos to the end of the
      text in the rich edit control }
    ToEnd := Length(Text) - StartPos;
    FoundAt :=
      FindText(FindDialog1.FindText, StartPos, ToEnd, mySearchTypes);
    if FoundAt <> -1 then
    begin
      SetFocus;
      SelStart := FoundAt;
      SelLength := Length(FindDialog1.FindText);
    end
    else Beep;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
const Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := False;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
end;
```

See Also

Left (see page 11386)

Top (see page 11392)

61.1.15.5.5 TFindDialog.Top Property

Determines the Y coordinate (vertical position) of the Find (see page 11383) dialog's upper left corner.

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

Top holds the same value as the second integer in Position (see page 11390). This property is maintained for compatibility with older applications.

See Also

Left (see page 11386)

Position (see page 11390)

61.1.15.6 TFindDialog Events

The events of the TFindDialog class are listed here.

Published Events

	Name	Description
	OnFind (see page 11392)	Occurs when the user clicks the Find (see page 11383) Next button.

61.1.15.6.1 TFindDialog.OnFind Event

Occurs when the user clicks the Find (see page 11383) Next button.

Pascal

```
property OnFind: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnFind;
```

Description

Write an OnFind event handler to perform the requested search when the user clicks Find (see page 11383) Next in the dialog. Use the Options (see page 11386) flags to determine how the search is conducted.

C++ Examples:

```
/*
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
```

```
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FindDialog1->Position = Point(RichEdit1->Left + RichEdit1->Width, RichEdit1->Top);
  FindDialog1->Execute();
}

void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  int FoundAt, StartPos, ToEnd;
  TSearchTypes mySearchTypes;
  // begin the search after the current selection
  // if there is one
  // otherwise, begin at the start of the text
  if (FindDialog1->Options.Contains(frMatchCase))
    mySearchTypes << stMatchCase;
  if (FindDialog1->Options.Contains(frWholeWord))
    mySearchTypes << stWholeWord;
  if (RichEdit1->SelLength)
    StartPos = RichEdit1->SelStart + RichEdit1->SelLength;
  else
    StartPos = 0;
  // ToEnd is the length from StartPos
  // to the end of the text in the rich edit control
  ToEnd = RichEdit1->Text.Length() - StartPos;
  FoundAt = RichEdit1->FindText(FindDialog1->FindText, StartPos, ToEnd, mySearchTypes);
  if (FoundAt != -1)
  {
    RichEdit1->SetFocus();
    RichEdit1->SelStart = FoundAt;
    RichEdit1->SelLength = FindDialog1->FindText.Length();
  }
  else Beep();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("../Overview.RTF");  // starts in Debug
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FindDialog1.Position :=
    Point(RichEdit1.Left + RichEdit1.Width, RichEdit1.Top);
  FindDialog1.Execute;
end;

procedure TForm1.FindDialog1Find(Sender: TObject);
var
  FoundAt: LongInt;
  StartPos, ToEnd: Integer;
  mySearchTypes : TSearchTypes;
  myFindOptions : TFindOptions;
begin
  with RichEdit1 do
```

```
    begin
      if frMatchCase in FindDialog1.Options then
         mySearchTypes := mySearchTypes + [stMatchCase];
      if frWholeWord in FindDialog1.Options then
         mySearchTypes := mySearchTypes + [stWholeWord];
      { begin the search after the current selection if there is one }
      { otherwise, begin at the start of the text }
      if SelLength <> 0 then
        StartPos := SelStart + SelLength
      else
        StartPos := 0;
      { ToEnd is the length from StartPos to the end of the
        text in the rich edit control }
      ToEnd := Length(Text) - StartPos;
      FoundAt :=
        FindText(FindDialog1.FindText, StartPos, ToEnd, mySearchTypes);
      if FoundAt <> -1 then
      begin
        SetFocus;
        SelStart := FoundAt;
        SelLength := Length(FindDialog1.FindText);
      end
      else Beep;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const Path = 'OverView.RTF';
  begin
    RichEdit1.PlainText := False;
    RichEdit1.Lines.LoadFromFile(Path);
    RichEdit1.ScrollBars := ssVertical;
  end;
```

See Also

FindText (see page 11384)

Options (see page 11386)

61.1.16 TFontDialog Class

TFontDialog displays a font-selection dialog.

Class Hierarchy

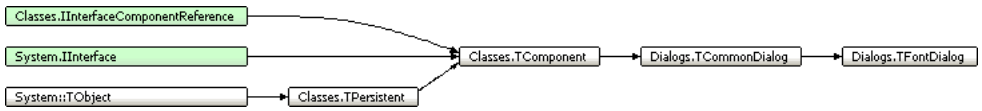

Pascal

```
TFontDialog = class(TCommonDialog);
```

C++

```
class TFontDialog : public TCommonDialog;
```

File

Dialogs

Description

TFontDialog displays a modal Windows dialog box for selecting fonts. The dialog does not appear at runtime until it is activated by a call to the Execute (see page 11399) method. When the user selects a font and clicks OK, the dialog closes and the selected font is stored in the Font (see page 11401) property.

61.1.16.1 TFontDialog Members

The following tables list the members exposed by TFontDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	Apply (see page 11399)	This is Apply, a member of class TFontDialog.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoClose (see page 11282)	Generates an OnClose (see page 11287) event.
	DoShow (see page 11283)	Generates an OnShow (see page 11288) event.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	MessageHook (see page 11284)	Provides special Windows message processing for the dialog.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	TaskModalDialog (see page 11284)	Ensures that Windows messages are properly routed when the dialog executes.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WndProc (see page 11400)	Responds to Windows messages sent to the dialog.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Template (see page 11287)	Points to a custom template for the dialog.
	TemplateModule (see page 11287)	This is TemplateModule, a member of class TCommonDialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11398)	Creates and initializes a TFontDialog (see page 11394) instance.
	DefaultHandler (see page 11282)	Sends unhandled messages to Windows for processing.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 11398)	Destroys the TFontDialog (see page 11394) and frees its memory.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 11399)	Displays the Font (see page 11401) dialog.
	Execute (see page 11283)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Handle (see page 11286)	Specifies the window handle of the dialog.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnApply (see page 11406)	Occurs when the user clicks the Apply (see page 11399) button.
	OnClose (see page 11287)	Occurs when the dialog closes.
	OnShow (see page 11288)	Occurs when the dialog opens.

Published Properties

	Name	Description
	Ctl3D (see page 11285)	Determines whether the dialog has a three-dimensional look.
	Device (see page 11401)	Specifies the device from which to retrieve the list of available fonts.
	Font (see page 11401)	Indicates the selected font.
	HelpContext (see page 11286)	Specifies the context number for online Help.

	MaxFontSize (see page 11403)	Determines the largest font size available in the Font (see page 11401) dialog.
	MinFontSize (see page 11404)	Determines the smallest font size available in the Font (see page 11401) dialog.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 11405)	Determines the appearance and behavior of the Font (see page 11401) dialog.
	Tag (see page 2808)	Stores an integer value as part of a component.

61.1.16.2 TFontDialog.Create Constructor

Creates and initializes a TFontDialog (see page 11394) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TFontDialog(TComponent * AOwner);
```

Description

The Create method generates a TFontDialog (see page 11394) instance, but the new dialog does not appear on the form at runtime until the Execute (see page 11399) method is called.

AOwner is the component, typically a form or data module, that is responsible for freeing the TFontDialog (see page 11394) instance. It becomes the value of the Owner property.

See Also

Owner (see page 2807)

Execute (see page 11399)

Destroy (see page 11398)

61.1.16.3 TFontDialog.Destroy Destructor

Destroys the TFontDialog (see page 11394) and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TFontDialog();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free verifies that the TFontDialog (see page 11394) reference is not nil before calling Destroy.

See Also

Free (see page 22300)

Create (see page 11398)

61.1.16.4 TFontDialog Methods

The methods of the TFontDialog class are listed here.

Protected Methods

	Name	Description
	Apply (see page 11399)	This is Apply, a member of class TFontDialog.
V	WndProc (see page 11400)	Responds to Windows messages sent to the dialog.

Public Methods

	Name	Description
V	Execute (see page 11399)	Displays the Font (see page 11401) dialog.

61.1.16.4.1 TFontDialog.Apply Method

Pascal

```
procedure Apply(Wnd: HWND); dynamic;
```

C++

```
__fastcall Apply(HWND Wnd);
```

Description

This is Apply, a member of class TFontDialog.

61.1.16.4.2 TFontDialog.Execute Method

Displays the Font (see page 11401) dialog.

Pascal

```
function Execute(ParentWnd: HWND): Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute(HWND ParentWnd);
```

Description

Execute opens the Font (see page 11401) dialog, returning true when the user selects a font and clicks OK.

```
FontDialog1.Font := Form1.Font;
if FontDialog1.Execute then
for i := 0 to (Form1.ControlCount - 1) do
if (Form1.Controls[i] is TButton) then
      (Form1.Controls[i] as TButton).Font :=
        FontDialog1.Font
FontDialog1->Font = Form1->Font;
if (FontDialog1->Execute())
  Button1->Font = FontDialog1->Font;
```

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
```

61.1.16.4.3 TFontDialog.WndProc Method

Responds to Windows messages sent to the dialog.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc is the main window procedure for the dialog. As implemented in TCommonDialog, it simply calls the Dispatch method, where any message handlers respond to messages before the DefaultHandler method sends them on to the Windows message handler.

Override WndProc to subclass the dialog's window procedure. To add additional message processing without interfering with the existing message handling of the dialog, override MessageHook instead.

See Also

DefaultHandler (see page 11282)

Dispatch

MessageHook (see page 11284)

61.1.16.5 TFontDialog Properties

The properties of the TFontDialog class are listed here.

Published Properties

	Name	Description
	Device (see page 11401)	Specifies the device from which to retrieve the list of available fonts.
	Font (see page 11401)	Indicates the selected font.
	MaxFontSize (see page 11403)	Determines the largest font size available in the Font (see page 11401) dialog.
	MinFontSize (see page 11404)	Determines the smallest font size available in the Font (see page 11401) dialog.
	Options (see page 11405)	Determines the appearance and behavior of the Font (see page 11401) dialog.

61.1.16.5.1 TFontDialog.Device Property

Specifies the device from which to retrieve the list of available fonts.

Pascal

```
property Device: TFontDialogDevice;
```

C++

```
__property TFontDialogDevice Device;
```

Description

Use Device to specify whether you want the dialog to list screen fonts (fdScreen), printer fonts (fdPrinter), or both screen and printer fonts (fdBoth).

61.1.16.5.2 TFontDialog.Font Property

Indicates the selected font.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

When the user selects a font in the dialog box and clicks OK, the selected font becomes the value of the Font property. To make a default font appear in the dialog when it opens, assign a value to Font in the Object Inspector or in program code.

```
FontDialog1.Font := Form1.Font;
if FontDialog1.Execute then
for i := 0 to (Form1.ControlCount - 1) do
if (Form1.Controls[i] is TButton) then
      (Form1.Controls[i] as TButton).Font :=
        FontDialog1.Font
FontDialog1->Font = Form1->Font;
if (FontDialog1->Execute())
  Button1->Font = FontDialog1->Font;
```

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;
```

```
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
```

See Also

TFont (see page 14497)

61.1.16.5.3 TFontDialog.MaxFontSize Property

Determines the largest font size available in the Font (see page 11401) dialog.

Pascal

```
property MaxFontSize: Integer;
```

C++

```
__property int MaxFontSize;
```

Description

Assign a value to MaxFontSize to limit the font sizes available in the dialog.

MaxFontSize is inoperative unless the fdLimitSize flag is set in Options (see page 11405).

C++ Examples:

```
/*
This example uses a Font dialog box, a button, and a label
on a form. When the user clicks the button, the Font dialog
box appears. The font sizes available are within the range
of 8 to 14. When the user chooses OK, the selected font is
applied to the caption of the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options.Clear();
  FontDialog1->Options << fdLimitSize;
  FontDialog1->MaxFontSize = 14;
  FontDialog1->MinFontSize = 8;
  if (FontDialog1->Execute())
    Label1->Font = FontDialog1->Font;
}
```

Delphi Examples:

```
{
This example uses a Font dialog box, a button, and a label
on a form. When the user clicks the button, the Font dialog
box appears. The font sizes available are within the range
of 8 to 14. When the user chooses OK, the selected font is
applied to the caption of the label.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  FontDialog1.Options := [fdLimitSize];
  FontDialog1.MaxFontSize := 14;
  FontDialog1.MinFontSize := 8;
  if FontDialog1.Execute then
    Label1.Font := FontDialog1.Font;
end;
```

See Also

MinFontSize (see page 11404)

Options (see page 11405)

61.1.16.5.4 TFontDialog.MinFontSize Property

Determines the smallest font size available in the Font (see page 11401) dialog.

Pascal

```
property MinFontSize: Integer;
```

C++

```
__property int MinFontSize;
```

Description

Assign a value to MinFontSize to limit the font sizes available in the dialog.

MinFontSize is inoperative unless the fdLimitSize flag is set in Options (see page 11405).

C++ Examples:

```
/*
This example uses a Font dialog box, a button, and a label
on a form. When the user clicks the button, the Font dialog
box appears. The font sizes available are within the range
of 8 to 14. When the user chooses OK, the selected font is
applied to the caption of the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options.Clear();
  FontDialog1->Options << fdLimitSize;
  FontDialog1->MaxFontSize = 14;
  FontDialog1->MinFontSize = 8;
  if (FontDialog1->Execute())
    Label1->Font = FontDialog1->Font;
}
```

Delphi Examples:

```
{
This example uses a Font dialog box, a button, and a label
on a form. When the user clicks the button, the Font dialog
box appears. The font sizes available are within the range
of 8 to 14. When the user chooses OK, the selected font is
applied to the caption of the label.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FontDialog1.Options := [fdLimitSize];
  FontDialog1.MaxFontSize := 14;
  FontDialog1.MinFontSize := 8;
  if FontDialog1.Execute then
    Label1.Font := FontDialog1.Font;
```

```
end;
```

See Also

MaxFontSize (see page 11403)

Options (see page 11405)

61.1.16.5.5 TFontDialog.Options Property

Determines the appearance and behavior of the Font (see page 11401) dialog.

Pascal

```
property Options: TFontDialogOptions;
```

C++

```
__property TFontDialogOptions Options;
```

Description

Use the Options property to customize the appearance and functionality of the dialog. Use these flags to indicate which fonts should appear and what controls appear in the dialog.

By default, all options except fdEffects are off.

C++ Examples:

```
/*
This example uses a Font dialog box, a button, and a label
on a form. When the user clicks the button, the Font dialog
box appears. The font sizes available are within the range
of 8 to 14. When the user chooses OK, the selected font is
applied to the caption of the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options.Clear();
  FontDialog1->Options << fdLimitSize;
  FontDialog1->MaxFontSize = 14;
  FontDialog1->MinFontSize = 8;
  if (FontDialog1->Execute())
    Label1->Font = FontDialog1->Font;
}
```

Delphi Examples:

```
{
This example uses a Font dialog box, a button, and a label
on a form. When the user clicks the button, the Font dialog
box appears. The font sizes available are within the range
of 8 to 14. When the user chooses OK, the selected font is
applied to the caption of the label.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FontDialog1.Options := [fdLimitSize];
  FontDialog1.MaxFontSize := 14;
  FontDialog1.MinFontSize := 8;
  if FontDialog1.Execute then
    Label1.Font := FontDialog1.Font;
end;
```

See Also

OnApply (see page 11406)

MaxFontSize (see page 11403)

MinFontSize (see page 11404)

61.1.16.6 TFontDialog Events

The events of the TFontDialog class are listed here.

Published Events

	Name	Description
	OnApply (see page 11406)	Occurs when the user clicks the Apply (see page 11399) button.

61.1.16.6.1 TFontDialog.OnApply Event

Occurs when the user clicks the Apply (see page 11399) button.

Pascal

```
property OnApply: TFDApplyEvent;
```

C++

```
__property TFDApplyEvent OnApply;
```

Description

The Apply (see page 11399) button appears in the dialog if an event handler is assigned to OnApply or if the fdApplyButton flag is set in Options (see page 11405). Use an OnApply event handler to apply the selected font without closing the Font (see page 11401) dialog box.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
```

```
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
```

See Also

Font (see page 11401)

Handle (see page 11286)

Options (see page 11405)

Font (see page 5823)

61.1.17 TOpenDialog Class

TOpenDialog displays a file-selection dialog.

Class Hierarchy

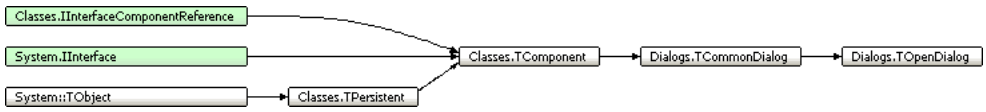

Pascal

```
TOpenDialog = class(TCommonDialog);
```

C++

```
class TOpenDialog : public TCommonDialog;
```

File

Dialogs

Description

TOpenDialog displays a modal Windows dialog box for selecting and opening files. The dialog does not appear at runtime until it is activated by a call to the Execute (see page 11415) method. When the user clicks Open, the dialog closes and the selected file or files are stored in the Files (see page 11429) property.

See Also

TOpenPictureDialog (see page 13282)

TSaveDialog (see page 11489)

TSavePictureDialog (see page 13300)

61.1.17.1 TOpenDialog Members

The following tables list the members exposed by TOpenDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanClose (see page 11413)	This is CanClose, a member of class TOpenDialog.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 11413)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoCanClose (see page 11413)	This is DoCanClose, a member of class TOpenDialog.
	DoClose (see page 11282)	Generates an OnClose (see page 11287) event.
	DoExecute (see page 11413)	This is the overview for the DoExecute method overload.
	DoFolderChange (see page 11414)	This is DoFolderChange, a member of class TOpenDialog.
	DoIncludeItem (see page 11414)	This is DoIncludeItem, a member of class TOpenDialog.
	DoSelectionChange (see page 11415)	This is DoSelectionChange, a member of class TOpenDialog.
	DoShow (see page 11283)	Generates an OnShow (see page 11288) event.
	DoTypeChange (see page 11415)	This is DoTypeChange, a member of class TOpenDialog.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetFileNames (see page 11417)	This is GetFileNames, a member of class TOpenDialog.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetStaticRect (see page 11417)	Returns the coordinates of the reserved area of the dialog box.

	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	MessageHook (see page 11284)	Provides special Windows message processing for the dialog.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	TaskModalDialog (see page 11284)	Ensures that Windows messages are properly routed when the dialog executes.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WndProc (see page 11420)	Responds to Windows messages sent to the dialog.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Template (see page 11287)	Points to a custom template for the dialog.
	TemplateModule (see page 11287)	This is TemplateModule, a member of class TCommonDialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11411)	Creates and initializes a TOpenDialog (see page 11407) instance.
V	DefaultHandler (see page 11282)	Sends unhandled messages to Windows for processing.

	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 11412)	Destroys the TOpenDialog (see page 11407) and frees its memory.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 11415)	Displays the file-selection dialog.
	Execute (see page 11283)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	FileEditStyle (see page 11423)	Determines the style of the file-selection dialog. (Obsolete.)
R	Files (see page 11429)	List of selected file names.
R	Handle (see page 11286)	Specifies the window handle of the dialog.
	HistoryList (see page 11436)	Maintains a list of previously selected files. (Obsolete.)
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCanClose (see page 11439)	Occurs when the user tries to close the dialog without canceling.
	OnClose (see page 11287)	Occurs when the dialog closes.
	OnFolderChange (see page 11440)	Occurs when a directory is opened or closed from the dialog.
	OnIncludeItem (see page 11440)	Occurs before the dialog adds a file to the file list box.
	OnSelectionChange (see page 11441)	Occurs when file names displayed in the dialog are changed.
	OnShow (see page 11288)	Occurs when the dialog opens.
	OnTypeChange (see page 11441)	Occurs when the file types displayed in the dialog are changed.

Published Properties

	Name	Description
	Ctl3D (see page 11285)	Determines whether the dialog has a three-dimensional look.
	DefaultExt (see page 11421)	Specifies a default file extension.
	FileName (see page 11424)	Indicates the name and directory path of the last file selected.
	Filter (see page 11431)	Determines the file masks (filters) available in the dialog.
	FilterIndex (see page 11434)	Determines which filter is selected by default when the dialog opens.
	HelpContext (see page 11286)	Specifies the context number for online Help.
	InitialDir (see page 11436)	Determines the current directory when the dialog opens.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 11436)	Determines the appearance and behavior of the file-selection dialog.
	OptionsEx (see page 11438)	Augments the Options (see page 11436) property with additional flags that determine the appearance and behavior of the file-selection dialog.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Title (see page 11438)	Specifies the text in the dialog's title bar.

61.1.17.2 TOpenDialog.Create Constructor

Creates and initializes a TOpenDialog (see page 11407) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TOpenDialog(TComponent * AOwner);
```

Description

Create generates a TOpenDialog (see page 11407) instance, but the new dialog does not appear on the form at runtime until the Execute (see page 11415) method is called.

AOwner is the component that is responsible for freeing the TOpenDialog (see page 11407) instance. It becomes the value of the Owner property.

See Also

Execute (see page 11415)

Owner (see page 2807)

Destroy (see page 11412)

61.1.17.3 TOpenDialog.Destroy Destructor

Destroys the TOpenDialog (see page 11407) and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TOpenDialog();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free verifies that the TOpenDialog (see page 11407) reference is not nil before calling Destroy.

See Also

Free (see page 22300)

Create (see page 11411)

61.1.17.4 TOpenDialog Methods

The methods of the TOpenDialog class are listed here.

Protected Methods

	Name	Description
	CanClose (see page 11413)	This is CanClose, a member of class TOpenDialog.
V	DefineProperties (see page 11413)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoCanClose (see page 11413)	This is DoCanClose, a member of class TOpenDialog.
	DoExecute (see page 11413)	This is the overview for the DoExecute method overload.
	DoFolderChange (see page 11414)	This is DoFolderChange, a member of class TOpenDialog.
	DoIncludeItem (see page 11414)	This is DoIncludeItem, a member of class TOpenDialog.
	DoSelectionChange (see page 11415)	This is DoSelectionChange, a member of class TOpenDialog.
	DoTypeChange (see page 11415)	This is DoTypeChange, a member of class TOpenDialog.
	GetFileNames (see page 11417)	This is GetFileNames, a member of class TOpenDialog.
V	GetStaticRect (see page 11417)	Returns the coordinates of the reserved area of the dialog box.
V	WndProc (see page 11420)	Responds to Windows messages sent to the dialog.

Public Methods

	Name	Description
V	Execute (see page 11415)	Displays the file-selection dialog.

61.1.17.4.1 TOpenDialog.CanClose Method

Pascal

```
function CanClose(var OpenFileName: TOpenFileName): Boolean;
```

C++

```
__fastcall Boolean CanClose(TOpenFileName OpenFileName);
```

Description

This is CanClose, a member of class TOpenDialog.

61.1.17.4.2 TOpenDialog.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor (see page 2839)

DefineBinaryProperty (see page 2837)

DefineProperties (see page 2931)

DefineProperty (see page 2838)

61.1.17.4.3 TOpenDialog.DoCanClose Method

Pascal

```
function DoCanClose: Boolean; dynamic;
```

C++

```
__fastcall Boolean DoCanClose();
```

Description

This is DoCanClose, a member of class TOpenDialog.

61.1.17.4.4 DoExecute Method

This is the overview for the DoExecute method overload.

Overload List

	Name	Description
	TOpenDialog.DoExecute (Pointer) (see page 11414)	This is DoExecute, a member of class TOpenDialog.
	TOpenDialog.DoExecute (Pointer, HWND) (see page 11414)	This is DoExecute, a member of class TOpenDialog.

61.1.17.4.4.1 TOpenDialog.DoExecute Method (Pointer)

Pascal

```
function DoExecute(Func: Pointer): Bool; overload;
```

C++

```
__fastcall Bool DoExecute(void * Func);
```

Description

This is DoExecute, a member of class TOpenDialog.

61.1.17.4.4.2 TOpenDialog.DoExecute Method (Pointer, HWND)

Pascal

```
function DoExecute(Func: Pointer; ParentWnd: HWND): Bool; overload;
```

C++

```
__fastcall Bool DoExecute(void * Func, HWND ParentWnd);
```

Description

This is DoExecute, a member of class TOpenDialog.

61.1.17.4.5 TOpenDialog.DoFolderChange Method

Pascal

```
procedure DoFolderChange; dynamic;
```

C++

```
__fastcall DoFolderChange();
```

Description

This is DoFolderChange, a member of class TOpenDialog.

61.1.17.4.6 TOpenDialog.DoIncludeItem Method

Pascal

```
procedure DoIncludeItem(const OFN: TOFNotifyEx; var Include: Boolean); dynamic;
```

C++

```
__fastcall DoIncludeItem(const TOFNotifyEx OFN, Boolean Include);
```

Description

This is DoIncludeItem, a member of class TOpenDialog.

61.1.17.4.7 TOpenDialog.DoSelectionChange Method

Pascal

```
procedure DoSelectionChange; dynamic;
```

C++

```
__fastcall DoSelectionChange();
```

Description

This is DoSelectionChange, a member of class TOpenDialog.

61.1.17.4.8 TOpenDialog.DoTypeChange Method

Pascal

```
procedure DoTypeChange; dynamic;
```

C++

```
__fastcall DoTypeChange();
```

Description

This is DoTypeChange, a member of class TOpenDialog.

61.1.17.4.9 TOpenDialog.Execute Method

Displays the file-selection dialog.

Pascal

```
function Execute(ParentWnd: HWND): Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute(HWND ParentWnd);
```

Description

Execute opens the file-selection dialog, returning true when the user selects a file and clicks Open. If the user clicks Cancel, Execute returns false.

```
if OpenDialog1.Execute then
  Memo1.Lines.LoadFromFile(OpenDialog1.FileName)
else
  Memo1.Lines.Clear;
if (OpenDialog1->Execute())
  Memo1->Lines->LoadFromFile(OpenDialog1->FileName);
else
  Memo1->Clear();
```

C++ Examples:

```
/*
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  OpenPictureDialog1->DefaultExt = GraphicExtension(__classid(TIcon));
  OpenPictureDialog1->FileName = GraphicFileMask(__classid(TIcon));
  OpenPictureDialog1->Filter = GraphicFilter(__classid(TIcon));
  OpenPictureDialog1->Options.Clear();
  OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
  while (true)
  {
    if (OpenPictureDialog1->Execute())
    {
      if (!OpenPictureDialog1->Options.Contains(ofExtensionDifferent))
      {
        Application->Icon->LoadFromFile(OpenPictureDialog1->FileName);
        break;
      }
      else // reset Options to remove ofExtensionDifferent
      {
        OpenPictureDialog1->Options.Clear();
        OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
      }
    }
    else // user cancelled
      break;
  }
}
```

Delphi Examples:

```
{
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
}
procedure TForm1.Button1Click(Sender: TObject);
var
Done: Boolean;
filenamestring : String;
begin
  OpenPictureDialog1.DefaultExt := GraphicExtension(TIcon);
  filenamestring := GraphicFileMask(TIcon);
  OpenPictureDialog1.FileName := filenamestring;
  OpenPictureDialog1.Filter := GraphicFilter(TIcon);
  OpenPictureDialog1.Options := [ ofFileMustExist, ofHideReadOnly, ofNoChangeDir ];
  Done := False;
  while not Done do
  begin
  if OpenPictureDialog1.Execute then
    begin
    if not (ofExtensionDifferent in OpenPictureDialog1.Options) then
      begin
      Application.Icon.LoadFromFile(OpenPictureDialog1.FileName);
      Done := True;
      end
    else
//      OpenPictureDialog1.Options := OpenPictureDialog1.Options - ofExtensionDifferent;
    end
  else { User cancelled }
    Done := True;
  end;
end;
{
Use a TOpenDialog to select a file.  Place the first line of
the file into a TEdit.
}
procedure TForm1.Edit1Click(Sender: TObject);
```

```
var
  F: TextFile;
  S: string;
begin
  if OpenDialog1.Execute then            { Display Open dialog box }
  begin
    AssignFile(F, OpenDialog1.FileName); { File selected in dialog }
    Reset(F);
    Readln(F, S);                        { Read first line of file }
    Edit1.Text := S;                     { Put string in a TEdit control }
    CloseFile(F);
  end;
end;
```

See Also

Files (see page 11429)

FileName (see page 11424)

61.1.17.4.10 TOpenDialog.GetFileNames Method

Pascal

```
procedure GetFileNames(var OpenFileName: TOpenFileName);
```

C++

```
__fastcall GetFileNames(TOpenFileName OpenFileName);
```

Description

This is GetFileNames, a member of class TOpenDialog.

61.1.17.4.11 TOpenDialog.GetStaticRect Method

Returns the coordinates of the reserved area of the dialog box.

Pascal

```
function GetStaticRect: TRect; virtual;
```

C++

```
virtual __fastcall TRect GetStaticRect();
```

Description

GetStaticRect is provided for components that descend from TOpenDialog (see page 11407) and require the placement of new controls alongside the standard ones inherited from the parent class. For Explorer-style dialogs, GetStaticRect returns the size and location of the standard controls within the dialog. For older versions of Windows, or if ofOldStyleDialog is enabled in Options (see page 11436), GetStaticRect returns the client dimensions of the entire dialog box.

C++ Examples:

```
/*
The following example is taken from TOpenPictureDialog. It
calculates the preview rectangle based on the difference
between the client area of the dialog and the static area of
the standard controls.
*/

#include <ExtDlgs.hpp>

class MyOpenPictureDialog : public TOpenPictureDialog
{
```

```
private:    // User declarations
    TPanel *FPicturePanel;  // can't use the private versions in TOpenPictureDialog
    TLabel *FPictureLabel;
    TSpeedButton *FPreviewButton;
    TPanel *FPaintPanel;
    TImage *FImageCtrl;
    AnsiString FSavedFilename;
protected:  // User declarations
    DYNAMIC void __fastcall DoShow(void);
public:     // User declarations
    __fastcall MyOpenPictureDialog(TComponent* Owner);
    virtual bool __fastcall  Execute(HWND ParentWnd){ return TOpenDialog::Execute(ParentWnd); }
};

__fastcall MyOpenPictureDialog::MyOpenPictureDialog(TComponent* Owner)
    : TOpenPictureDialog(Owner)
{
}

void __fastcall MyOpenPictureDialog::DoShow(void)
{
  RECT *PreviewRect = new RECT();
  TRect StaticRect;
  FPicturePanel = new TPanel(this);
  FPictureLabel = new TLabel(this);
  FPreviewButton = new TSpeedButton(this);
  FPaintPanel = new TPanel(this);
  FImageCtrl = new TImage(this);
  // Set preview area to entire dialog
  GetClientRect(Handle, PreviewRect);
  StaticRect = GetStaticRect();
  // Move preview area to right of static area
  PreviewRect->left = StaticRect.Left + (StaticRect.Right - StaticRect.Left);
  PreviewRect->top += 4;
  FPicturePanel->BoundsRect.Left = PreviewRect->left;
  FPicturePanel->BoundsRect.Right = PreviewRect->right;
  FPicturePanel->BoundsRect.Top = PreviewRect->top;
  FPicturePanel->BoundsRect.Bottom = PreviewRect->bottom;

  FPreviewButton->Left = FPaintPanel->BoundsRect.Right - FPreviewButton->Width - 2;
  FImageCtrl->Picture = NULL;
  FSavedFilename = "";
  FPaintPanel->Caption = "";
  FPicturePanel->ParentWindow = Handle;
  delete PreviewRect;
  delete FPicturePanel;
  delete FPreviewButton;
  delete FPaintPanel;
  delete FImageCtrl;
//  TOpenDialog::DoShow();   // the above code is equivalent to the inherited version
}

TForm1 *Form1;
MyOpenPictureDialog *OpenPictureDialog1;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (OpenPictureDialog1->Execute(GetParentHandle()))
      Image1->Picture->LoadFromFile(OpenPictureDialog1->FileName);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  OpenPictureDialog1 =  new MyOpenPictureDialog(Form1);
}
```

Delphi Examples:

```
{
The following example is overrides the
TOpenPictureDialog.DoShow method and implements it here.
DoShow calculates the preview rectangle based on the
difference between the client area of the dialog and the
static area of the standard controls.
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    Image1: TImage;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  MyOpenPictureDialog = class(TOpenPictureDialog)
  private
    FPicturePanel: TPanel; // can't use the private versions in TOpenPictureDialog
    FPictureLabel: TLabel;
    FPreviewButton: TSpeedButton;
    FPaintPanel: TPanel;
    FImageCtrl: TImage;
    FSavedFilename: string;
  public
    { Public declarations }
  protected
    procedure DoShow; override;
  end;
var
  Form1: TForm1;
  OpenPictureDialog1: MyOpenPictureDialog;

implementation

{$R *.dfm}

procedure MyOpenPictureDialog.DoShow;
var
  PreviewRect, StaticRect: TRect;
begin
  FPicturePanel:= TPanel.Create(Self);
  FPictureLabel:= TLabel.Create(Self);
  FPreviewButton:= TSpeedButton.Create(Self);
  FPaintPanel:= TPanel.Create(Self);
  FImageCtrl:= TImage.Create(Self);
  // Set preview area to entire dialog
  GetClientRect(Handle, PreviewRect);
  StaticRect := GetStaticRect;
  // Move preview area to right of static area
  PreviewRect.Left := StaticRect.Left + (StaticRect.Right - StaticRect.Left);
  Inc(PreviewRect.Top, 4);
  FPicturePanel.BoundsRect := PreviewRect;
  FPreviewButton.Left := FPaintPanel.BoundsRect.Right - FPreviewButton.Width - 2;
  FImageCtrl.Picture := nil;
  FSavedFilename := '';
  FPaintPanel.Caption := '';
  FPicturePanel.ParentWindow := Handle;
//  TCommonDialog.DoShow;  // this is the one TOpenPictureDialog.DoShow calls
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  if OpenPictureDialog1.Execute then
```

```
      Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  OpenPictureDialog1:= MyOpenPictureDialog.Create(Form1);
//  OpenPictureDialog1.Parent:= Form1;
//  OpenPictureDialog1.visible := false;
end;
```

See Also

Options (see page 11436)

Template (see page 11287)

TOpenPictureDialog (see page 13282)

61.1.17.4.12 TOpenDialog.WndProc Method

Responds to Windows messages sent to the dialog.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc is the main window procedure for the dialog. As implemented in TCommonDialog, it simply calls the Dispatch method, where any message handlers respond to messages before the DefaultHandler method sends them on to the Windows message handler.

Override WndProc to subclass the dialog's window procedure. To add additional message processing without interfering with the existing message handling of the dialog, override MessageHook instead.

See Also

DefaultHandler (see page 11282)

Dispatch

MessageHook (see page 11284)

61.1.17.5 TOpenDialog Properties

The properties of the TOpenDialog class are listed here.

Public Properties

	Name	Description
	FileEditStyle (see page 11423)	Determines the style of the file-selection dialog. (Obsolete.)
R	Files (see page 11429)	List of selected file names.
	HistoryList (see page 11436)	Maintains a list of previously selected files. (Obsolete.)

Published Properties

	Name	Description
	DefaultExt (see page 11421)	Specifies a default file extension.
	FileName (see page 11424)	Indicates the name and directory path of the last file selected.

	Filter (see page 11431)	Determines the file masks (filters) available in the dialog.
	FilterIndex (see page 11434)	Determines which filter is selected by default when the dialog opens.
	InitialDir (see page 11436)	Determines the current directory when the dialog opens.
	Options (see page 11436)	Determines the appearance and behavior of the file-selection dialog.
	OptionsEx (see page 11438)	Augments the Options (see page 11436) property with additional flags that determine the appearance and behavior of the file-selection dialog.
	Title (see page 11438)	Specifies the text in the dialog's title bar.

61.1.17.5.1 TOpenDialog.DefaultExt Property

Specifies a default file extension.

Pascal

```
property DefaultExt: string;
```

C++

```
__property AnsiString DefaultExt;
```

Description

DefaultExt specifies a file extension that is appended automatically to the selected file name, unless the selected file name already includes a registered extension. If the user selects a file name with an extension that is unregistered, DefaultExt is appended to the unregistered extension.

Extensions longer than three characters are not supported. Do not include the period (.) that divides the file name and its extension.

C++ Examples:

```
/*
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString NewFileName, OldFileName;
  TFileStream *NewFile, *OldFile;
  SavePictureDialog1->DefaultExt = GraphicExtension(__classid(Graphics::TBitmap));
  SavePictureDialog1->Filter = GraphicFilter(__classid(Graphics::TBitmap));
  OldFileName = "factory.bmp";
  if (SavePictureDialog1->Execute())
  {
    NewFileName = SavePictureDialog1->FileName;
    OldFile = new TFileStream(OldFileName, (fmOpenRead | fmShareDenyWrite));
    try
    {
      NewFile = new TFileStream(NewFileName, (fmCreate | fmShareDenyRead));
      try
      {
        NewFile->CopyFrom(OldFile, OldFile->Size);
      }
      __finally
      {
        FreeAndNil(&NewFile);
      };
    }
    __finally
    {
      FreeAndNil(&OldFile);
    };
```

```
  };
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Image1->Picture->LoadFromFile("../factory.bmp");
}
/*
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenPictureDialog1->DefaultExt = GraphicExtension(__classid(TIcon));
  OpenPictureDialog1->FileName = GraphicFileMask(__classid(TIcon));
  OpenPictureDialog1->Filter = GraphicFilter(__classid(TIcon));
  OpenPictureDialog1->Options.Clear();
  OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
  while (true)
  {
    if (OpenPictureDialog1->Execute())
    {
      if (!OpenPictureDialog1->Options.Contains(ofExtensionDifferent))
      {
        Application->Icon->LoadFromFile(OpenPictureDialog1->FileName);
        break;
      }
      else // reset Options to remove ofExtensionDifferent
      {
        OpenPictureDialog1->Options.Clear();
        OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
      }
    }
    else // user cancelled
      break;
  }
}
```

Delphi Examples:

```
{
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
}
procedure TForm1.Button1Click(Sender: TObject);
var NewFileName, OldFileName: string;
    NewFile, OldFile: TFileStream;
begin
  SavePictureDialog1.DefaultExt := GraphicExtension(TBitmap);
  SavePictureDialog1.Filter := GraphicFilter(TBitmap);
  OldFileName := 'factory.bmp';
  if SavePictureDialog1.Execute then
  begin
    NewFileName := SavePictureDialog1.FileName;
    OldFile := TFileStream.Create(
      OldFileName, fmOpenRead or fmShareDenyWrite);
    try
      NewFile := TFileStream.Create(
        NewFileName, fmCreate or fmShareDenyRead);
      try
```

```
        NewFile.CopyFrom(OldFile, OldFile.Size);
      finally
        FreeAndNil(NewFile);
      end;
    finally
      FreeAndNil(OldFile);
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Picture.LoadFromFile('factory.bmp');
end;
{
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
}
procedure TForm1.Button1Click(Sender: TObject);
var
Done: Boolean;
filenamestring : String;
begin
  OpenPictureDialog1.DefaultExt := GraphicExtension(TIcon);
  filenamestring := GraphicFileMask(TIcon);
  OpenPictureDialog1.FileName := filenamestring;
  OpenPictureDialog1.Filter := GraphicFilter(TIcon);
  OpenPictureDialog1.Options := [ ofFileMustExist, ofHideReadOnly, ofNoChangeDir ];
  Done := False;
  while not Done do
  begin
  if OpenPictureDialog1.Execute then
    begin
    if not (ofExtensionDifferent in OpenPictureDialog1.Options) then
      begin
      Application.Icon.LoadFromFile(OpenPictureDialog1.FileName);
      Done := True;
      end
    else
//      OpenPictureDialog1.Options := OpenPictureDialog1.Options - ofExtensionDifferent;
    end
  else { User cancelled }
    Done := True;
  end;
end;
```

61.1.17.5.2 TOpenDialog.FileEditStyle Property

Determines the style of the file-selection dialog. (Obsolete.)

Pascal

```
property FileEditStyle: TFileEditStyle;
```

C++

```
__property TFileEditStyle FileEditStyle;
```

Description

FileEditStyle is maintained for compatibility with older versions of the VCL. It has no effect.

61.1.17.5.3 TOpenDialog.FileName Property

Indicates the name and directory path of the last file selected.

Pascal

```
property FileName: TFileName;
```

C++

```
__property TFileName FileName;
```

Description

The FileName property returns the name and complete directory path of the most recently selected file. The value of FileName is the same as the first item in the Files (see page 11429) property.

To make a file name appear by default in the dialog's edit box, assign a value to FileName in the Object Inspector or in program code. Programmatic changes to FileName have no effect while the dialog is active.

```
if OpenDialog1.Execute then
  Memo1.Lines.LoadFromFile(OpenDialog1.FileName)
else
  Memo1.Lines.Clear;
if (OpenDialog1->Execute())
  Memo1->Lines->LoadFromFile(OpenDialog1->FileName);
else
  Memo1->Clear();
```

C++ Examples:

```
/*
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString NewFileName, OldFileName;
  TFileStream *NewFile, *OldFile;
  SavePictureDialog1->DefaultExt = GraphicExtension(__classid(Graphics::TBitmap));
  SavePictureDialog1->Filter = GraphicFilter(__classid(Graphics::TBitmap));
  OldFileName = "factory.bmp";
  if (SavePictureDialog1->Execute())
  {
    NewFileName = SavePictureDialog1->FileName;
    OldFile = new TFileStream(OldFileName, (fmOpenRead | fmShareDenyWrite));
    try
    {
      NewFile = new TFileStream(NewFileName, (fmCreate | fmShareDenyRead));
      try
      {
        NewFile->CopyFrom(OldFile, OldFile->Size);
      }
      __finally
      {
        FreeAndNil(&NewFile);
      };
    }
    __finally
    {
      FreeAndNil(&OldFile);
    };
  };
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Image1->Picture->LoadFromFile("../factory.bmp");
}
/*
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
*/
void __fastcall TForm1::Add_a_fileClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    int index = TabControl1->Tabs->Add(OpenDialog1->FileName);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[index]);
  }
}

void __fastcall TForm1::Assign_filesClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TabControl1->Tabs->Assign(OpenDialog1->Files);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[TabControl1->TabIndex]);
  }
}
/*
Place the following code in the tab control’s OnChange event
handler:
*/
void __fastcall TForm1::TabControl1Change(TObject *Sender)
{
    Memo1->Lines->LoadFromFile(
      TabControl1->Tabs->Strings[TabControl1->TabIndex]);
}
/*
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenPictureDialog1->DefaultExt = GraphicExtension(__classid(TIcon));
  OpenPictureDialog1->FileName = GraphicFileMask(__classid(TIcon));
  OpenPictureDialog1->Filter = GraphicFilter(__classid(TIcon));
  OpenPictureDialog1->Options.Clear();
  OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
  while (true)
  {
    if (OpenPictureDialog1->Execute())
    {
      if (!OpenPictureDialog1->Options.Contains(ofExtensionDifferent))
```

```
        {
          Application->Icon->LoadFromFile(OpenPictureDialog1->FileName);
          break;
        }
        else // reset Options to remove ofExtensionDifferent
        {
          OpenPictureDialog1->Options.Clear();
          OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
        }
      }
      else // user cancelled
        break;
    }
  }
  /*
  Click the button to open a TOpenDialog, then select a file
  to delete.
  */
  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    OpenDialog1->Title = "Delete File";
    if (OpenDialog1->Execute())
    {
      if (FileExists(OpenDialog1->FileName))
        DeleteFile(OpenDialog1->FileName);
    }
  }
```

Delphi Examples:

```
{
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
}
procedure TForm1.Button1Click(Sender: TObject);
var NewFileName, OldFileName: string;
    NewFile, OldFile: TFileStream;
begin
  SavePictureDialog1.DefaultExt := GraphicExtension(TBitmap);
  SavePictureDialog1.Filter := GraphicFilter(TBitmap);
  OldFileName := 'factory.bmp';
  if SavePictureDialog1.Execute then
  begin
    NewFileName := SavePictureDialog1.FileName;
    OldFile := TFileStream.Create(
      OldFileName, fmOpenRead or fmShareDenyWrite);
    try
      NewFile := TFileStream.Create(
        NewFileName, fmCreate or fmShareDenyRead);
      try
        NewFile.CopyFrom(OldFile, OldFile.Size);
      finally
        FreeAndNil(NewFile);
      end;
    finally
      FreeAndNil(OldFile);
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Picture.LoadFromFile('factory.bmp');
end;
```

```
{
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
}
procedure TForm1.Add_a_fileClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    index:= TabControl1.Tabs.Add(OpenDialog1.FileName);
    Memo1.Lines.LoadFromFile(TabControl1.Tabs[index]);
  end;
end;

procedure TForm1.Assign_filesClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    TabControl1.Tabs.Assign(OpenDialog1.Files);
    Memo1.Lines.LoadFromFile(
      TabControl1.Tabs[TabControl1.TabIndex]);
  end;
end;

{
Place the following code in the tab control's OnChange event
handler:
}
procedure TForm1.TabControl1Change(Sender: TObject);
begin
  with TabControl1 do
    Memo1.Lines.LoadFromFile(Tabs[TabIndex]);
end;
{
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
}
procedure TForm1.Button1Click(Sender: TObject);
var
Done: Boolean;
filenamestring : String;
begin
  OpenPictureDialog1.DefaultExt := GraphicExtension(TIcon);
  filenamestring := GraphicFileMask(TIcon);
  OpenPictureDialog1.FileName := filenamestring;
  OpenPictureDialog1.Filter := GraphicFilter(TIcon);
  OpenPictureDialog1.Options := [ ofFileMustExist, ofHideReadOnly, ofNoChangeDir ];
  Done := False;
  while not Done do
```

```
    begin
    if OpenPictureDialog1.Execute then
      begin
      if not (ofExtensionDifferent in OpenPictureDialog1.Options) then
        begin
        Application.Icon.LoadFromFile(OpenPictureDialog1.FileName);
        Done := True;
        end
      else
//      OpenPictureDialog1.Options := OpenPictureDialog1.Options - ofExtensionDifferent;
      end
    else { User cancelled }
      Done := True;
    end;
  end;
  {
  Use a TOpenDialog to select a file.  Place the first line of
  the file into a TEdit.
  }
  procedure TForm1.Edit1Click(Sender: TObject);
  var
    F: TextFile;
    S: string;
  begin
    if OpenDialog1.Execute then            { Display Open dialog box }
    begin
      AssignFile(F, OpenDialog1.FileName); { File selected in dialog }
      Reset(F);
      Readln(F, S);                        { Read first line of file }
      Edit1.Text := S;                     { Put string in a TEdit control }
      CloseFile(F);
    end;
  end;
  {
  Click the button to open a TOpenDialog, then select a file
  to delete.
  }

  procedure TForm1.Button1Click(Sender: TObject);
  var
    F: Textfile;
  begin
    OpenDialog1.Title := 'Delete File';
    if OpenDialog1.Execute then
    begin
      AssignFile(F, OpenDialog1.FileName);
      try
        Reset(F);
        if MessageDlg('Erase ' + OpenDialog1.FileName + '?',
          mtConfirmation, [mbYes, mbNo], 0) = mrYes then
        begin
          CloseFile(F);
          Erase(F);
        end;
      except
        on EInOutError do
          MessageDlg('File I/O error.', mtError, [mbOk], 0);
      end;
    end;
  end;
```

See Also

Files (see page 11429)

Execute (see page 11415)

61.1.17.5.4 TOpenDialog.Files Property

List of selected file names.

Pascal

```
property Files: TStrings;
```

C++

```
__property TStrings Files;
```

Description

Files is a string list that contains each selected file name with its full directory path. (To let users select multiple file names, set the ofAllowMultiSelect flag in Options (see page 11436).) Use properties and methods for string lists to traverse this list of files and read individual items.

The example below assigns the list of files in Files to the Items property of a TListBox component.

```
ListBox1.Items.Assign(OpenDialog1.Files);
ListBox1->Items->Assign(OpenDialog1->Files);
```

C++ Examples:

```
/*
This example uses an Open dialog box, a memo, and a button
on a form. When the user clicks the button, the Open dialog
box appears. When the user selects files in the dialog box
and chooses the OK button, the first line from each of the
files is added to the memo. Choose multiple files using the
CNTL key or the SHIFT key.
*/

#include <stdio.h>     // for FILE

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FILE *stream;
  char FirstLine[512];

  OpenDialog1->Options.Clear();
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist;
  OpenDialog1->Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
  OpenDialog1->FilterIndex = 2; // start the dialog showing all files
  if (OpenDialog1->Execute())
  {
    for (int I = 0; I < OpenDialog1->Files->Count; I ++)
    {
      stream = fopen(OpenDialog1->Files->Strings[I].c_str(), "r");
      if (stream)
      {
        // read the first line from the file
        fgets(FirstLine, sizeof(FirstLine), stream);
        Memo1->Lines->Append(FirstLine);
        fclose(stream);
      }
    }
  }
}
/*
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
```

```
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
*/
void __fastcall TForm1::Add_a_fileClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    int index = TabControl1->Tabs->Add(OpenDialog1->FileName);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[index]);
  }
}

void __fastcall TForm1::Assign_filesClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TabControl1->Tabs->Assign(OpenDialog1->Files);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[TabControl1->TabIndex]);
  }
}
/*
Place the following code in the tab control's OnChange event
handler:
*/
void __fastcall TForm1::TabControl1Change(TObject *Sender)
{
    Memo1->Lines->LoadFromFile(
      TabControl1->Tabs->Strings[TabControl1->TabIndex]);
}
```

Delphi Examples:

```
{
This example uses an Open dialog box, a memo, and a button
on a form. When the user clicks the button, the Open dialog
box appears. When the user selects files in the dialog box
and chooses the OK button, the first line from each of the
files is added to the memo. Choose multiple files using the
CNTL key or the SHIFT key.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: integer;
  F: TextFile;
  FirstLine: string;
begin
  OpenDialog1.Options := [ofAllowMultiSelect, ofFileMustExist];
  OpenDialog1.Filter := 'Text files (*.txt)|*.txt|All files (*.*)|*.*';
  OpenDialog1.FilterIndex := 2; { start the dialog showing all files }
  if OpenDialog1.Execute then
    with OpenDialog1.Files do
      for I := 0 to Count - 1 do
      begin
        AssignFile(F, Strings[I]);  { next file in Files property }
        Reset(F);
        Readln(F, FirstLine);  { Read the first line out of the file }
        Memo1.Lines.Append(FirstLine);  { Add the line to the memo }
        CloseFile(F);
      end;
end;
{
This example uses a tab control to display the contents of
```

```
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
}
procedure TForm1.Add_a_fileClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    index:= TabControl1.Tabs.Add(OpenDialog1.FileName);
    Memo1.Lines.LoadFromFile(TabControl1.Tabs[index]);
  end;
end;

procedure TForm1.Assign_filesClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    TabControl1.Tabs.Assign(OpenDialog1.Files);
    Memo1.Lines.LoadFromFile(
      TabControl1.Tabs[TabControl1.TabIndex]);
  end;
end;

{
Place the following code in the tab control's OnChange event
handler:
}
procedure TForm1.TabControl1Change(Sender: TObject);
begin
  with TabControl1 do
    Memo1.Lines.LoadFromFile(Tabs[TabIndex]);
end;
```

See Also

FileName (see page 11424)

Options (see page 11436)

TStrings (see page 3029)

61.1.17.5.5 TOpenDialog.Filter Property

Determines the file masks (filters) available in the dialog.

Pascal

```
property Filter: string;
```

C++

```
__property AnsiString Filter;
```

Description

The file-selection dialog includes a drop-down list of file types under the edit box. When the user picks a file type from the list, only files of the selected type are displayed in the dialog.

To configure file masks at design time, click on the ellipsis marks (...) to the right of the Filter property in the Object Inspector. This opens the Filter editor. In the left column of the Filter editor, under Filter Name, type a brief description of each file type that will be available at runtime. In the right column, under Filter, type the file mask corresponding to each description. For example, the description "Text files" might appear to the left of the mask "*.txt", and the description "Pascal source files" might appear to the left of the mask "*.pas". Since the description appears in the drop-down list at runtime, it is often helpful to show the mask explicitly in the description (for example, "Text files (*.txt)").

To create file masks in program code, assign a value to the Filter property that consists of a description and a mask separated by a vertical bar (pipe) character. Do not include spaces around the vertical bar. For example,

```
OpenDialog1.Filter := 'Text files (*.txt)|*.TXT';
OpenDialog1->Filter = "Text files (*.txt)|*.TXT";
```

Multiple filters should be separated by vertical bars. For example,

```
OpenDialog1.Filter := 'Text files (*.txt)|*.TXT|Pascal files (*.pas)|*.PAS';
OpenDialog1->Filter = "Text files (*.txt)|*.TXT|Pascal files (*.pas)|*.PAS";
```

To include multiple masks in a single filter, separate the masks with semicolons. This works both in the Object Inspector and in program code. For example,

```
OpenDialog1.Filter := 'Pascal files|*.PAS;*.DPK;*.DPR';
OpenDialog1->Filter = "Pascal files|*.PAS;*.DPK;*.DPR";
```

If no value is assigned to Filter, the dialog displays all file types.

C++ Examples:

```
/*
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString NewFileName, OldFileName;
  TFileStream *NewFile, *OldFile;
  SavePictureDialog1->DefaultExt = GraphicExtension(__classid(Graphics::TBitmap));
  SavePictureDialog1->Filter = GraphicFilter(__classid(Graphics::TBitmap));
  OldFileName = "factory.bmp";
  if (SavePictureDialog1->Execute())
  {
    NewFileName = SavePictureDialog1->FileName;
    OldFile = new TFileStream(OldFileName, (fmOpenRead | fmShareDenyWrite));
    try
    {
      NewFile = new TFileStream(NewFileName, (fmCreate | fmShareDenyRead));
      try
      {
        NewFile->CopyFrom(OldFile, OldFile->Size);
      }
      __finally
      {
        FreeAndNil(&NewFile);
      };
    }
    __finally
    {
```

```
      FreeAndNil(&OldFile);
    };
  };
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Image1->Picture->LoadFromFile("../factory.bmp");
}
/*
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenPictureDialog1->DefaultExt = GraphicExtension(__classid(TIcon));
  OpenPictureDialog1->FileName = GraphicFileMask(__classid(TIcon));
  OpenPictureDialog1->Filter = GraphicFilter(__classid(TIcon));
  OpenPictureDialog1->Options.Clear();
  OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
  while (true)
  {
    if (OpenPictureDialog1->Execute())
    {
      if (!OpenPictureDialog1->Options.Contains(ofExtensionDifferent))
      {
        Application->Icon->LoadFromFile(OpenPictureDialog1->FileName);
        break;
      }
      else // reset Options to remove ofExtensionDifferent
      {
        OpenPictureDialog1->Options.Clear();
        OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
      }
    }
    else // user cancelled
      break;
  }
}
```

Delphi Examples:

```
{
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
}
procedure TForm1.Button1Click(Sender: TObject);
var NewFileName, OldFileName: string;
    NewFile, OldFile: TFileStream;
begin
  SavePictureDialog1.DefaultExt := GraphicExtension(TBitmap);
  SavePictureDialog1.Filter := GraphicFilter(TBitmap);
  OldFileName := 'factory.bmp';
  if SavePictureDialog1.Execute then
  begin
    NewFileName := SavePictureDialog1.FileName;
    OldFile := TFileStream.Create(
      OldFileName, fmOpenRead or fmShareDenyWrite);
    try
      NewFile := TFileStream.Create(
```

```
        NewFileName, fmCreate or fmShareDenyRead);
      try
        NewFile.CopyFrom(OldFile, OldFile.Size);
      finally
        FreeAndNil(NewFile);
      end;
    finally
      FreeAndNil(OldFile);
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Picture.LoadFromFile('factory.bmp');
end;
{
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
}
procedure TForm1.Button1Click(Sender: TObject);
var
Done: Boolean;
filenamestring : String;
begin
  OpenPictureDialog1.DefaultExt := GraphicExtension(TIcon);
  filenamestring := GraphicFileMask(TIcon);
  OpenPictureDialog1.FileName := filenamestring;
  OpenPictureDialog1.Filter := GraphicFilter(TIcon);
  OpenPictureDialog1.Options := [ ofFileMustExist, ofHideReadOnly, ofNoChangeDir ];
  Done := False;
  while not Done do
  begin
  if OpenPictureDialog1.Execute then
    begin
    if not (ofExtensionDifferent in OpenPictureDialog1.Options) then
      begin
      Application.Icon.LoadFromFile(OpenPictureDialog1.FileName);
      Done := True;
      end
    else
//      OpenPictureDialog1.Options := OpenPictureDialog1.Options - ofExtensionDifferent;
    end
  else { User cancelled }
    Done := True;
  end;
end;
```

See Also

FilterIndex (see page 11434)

GraphicFilter (see page 14624)

61.1.17.5.6 TOpenDialog.FilterIndex Property

Determines which filter is selected by default when the dialog opens.

Pascal

```
property FilterIndex: Integer;
```

C++

```
__property int FilterIndex;
```

Description

FilterIndex determines which of the file types in Filter (see page 11431) is selected by default when the dialog opens. Set FilterIndex to 1 to choose the first file type in the list as the default, or set FilterIndex to 2 to choose the second file type as the default, and so forth. If the value of FilterIndex is out or range, the first file type listed in Filter (see page 11431) is the default.

C++ Examples:

```
/*
This example uses an Open dialog box, a memo, and a button
on a form. When the user clicks the button, the Open dialog
box appears. When the user selects files in the dialog box
and chooses the OK button, the first line from each of the
files is added to the memo. Choose multiple files using the
CNTL key or the SHIFT key.
*/

#include <stdio.h>     // for FILE

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FILE *stream;
  char FirstLine[512];

  OpenDialog1->Options.Clear();
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist;
  OpenDialog1->Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
  OpenDialog1->FilterIndex = 2; // start the dialog showing all files
  if (OpenDialog1->Execute())
  {
    for (int I = 0; I < OpenDialog1->Files->Count; I ++)
    {
      stream = fopen(OpenDialog1->Files->Strings[I].c_str(), "r");
      if (stream)
      {
        // read the first line from the file
        fgets(FirstLine, sizeof(FirstLine), stream);
        Memo1->Lines->Append(FirstLine);
        fclose(stream);
      }
    }
  }
}
```

Delphi Examples:

```
{
This example uses an Open dialog box, a memo, and a button
on a form. When the user clicks the button, the Open dialog
box appears. When the user selects files in the dialog box
and chooses the OK button, the first line from each of the
files is added to the memo. Choose multiple files using the
CNTL key or the SHIFT key.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: integer;
  F: TextFile;
  FirstLine: string;
begin
  OpenDialog1.Options := [ofAllowMultiSelect, ofFileMustExist];
  OpenDialog1.Filter := 'Text files (*.txt)|*.txt|All files (*.*)|*.*';
  OpenDialog1.FilterIndex := 2; { start the dialog showing all files }
```

```
    if OpenDialog1.Execute then
      with OpenDialog1.Files do
        for I := 0 to Count - 1 do
        begin
          AssignFile(F, Strings[I]);  { next file in Files property }
          Reset(F);
          Readln(F, FirstLine);  { Read the first line out of the file }
          Memo1.Lines.Append(FirstLine);  { Add the line to the memo }
          CloseFile(F);
        end;
  end;
```

See Also

Filter (see page 11431)

61.1.17.5.7 TOpenDialog.HistoryList Property

Maintains a list of previously selected files. (Obsolete.)

Pascal

```
property HistoryList: TStrings;
```

C++

```
__property TStrings HistoryList;
```

Description

HistoryList is maintained for compatibility with older versions of TOpenDialog (see page 11407). It is not used.

61.1.17.5.8 TOpenDialog.InitialDir Property

Determines the current directory when the dialog opens.

Pascal

```
property InitialDir: string;
```

C++

```
__property AnsiString InitialDir;
```

Description

InitialDir determines the default directory displayed in the file-selection dialog when it opens. For example, to point the dialog at the WINDOWS\SYSTEM directory, set the value of InitialDir to C:\WINDOWS\SYSTEM.

If no value is assigned to InitialDir, or if the specified directory does not exist, the initial directory is controlled by the global ForceCurrentDirectory (see page 11585) variable. If ForceCurrentDirectory (see page 11585) is true, the dialog opens with the current working directory displayed. Otherwise, the dialog opens with either the current working directory or the My Documents directory, depending on the version of Windows.

See Also

ForceCurrentDirectory (see page 11585)

61.1.17.5.9 TOpenDialog.Options Property

Determines the appearance and behavior of the file-selection dialog.

Pascal

```
property Options: TOpenOptions;
```

C++

```
__property TOpenOptions Options;
```

Description

Use the Options property to customize the appearance and functionality of the dialog.

C++ Examples:

```
/*
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenPictureDialog1->DefaultExt = GraphicExtension(__classid(TIcon));
  OpenPictureDialog1->FileName = GraphicFileMask(__classid(TIcon));
  OpenPictureDialog1->Filter = GraphicFilter(__classid(TIcon));
  OpenPictureDialog1->Options.Clear();
  OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
  while (true)
  {
    if (OpenPictureDialog1->Execute())
    {
      if (!OpenPictureDialog1->Options.Contains(ofExtensionDifferent))
      {
        Application->Icon->LoadFromFile(OpenPictureDialog1->FileName);
        break;
      }
      else // reset Options to remove ofExtensionDifferent
      {
        OpenPictureDialog1->Options.Clear();
        OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
      }
    }
    else // user cancelled
      break;
  }
}
```

Delphi Examples:

```
{
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
}
procedure TForm1.Button1Click(Sender: TObject);
var
Done: Boolean;
filenamestring : String;
begin
  OpenPictureDialog1.DefaultExt := GraphicExtension(TIcon);
  filenamestring := GraphicFileMask(TIcon);
  OpenPictureDialog1.FileName := filenamestring;
  OpenPictureDialog1.Filter := GraphicFilter(TIcon);
  OpenPictureDialog1.Options := [ ofFileMustExist, ofHideReadOnly, ofNoChangeDir ];
  Done := False;
  while not Done do
```

```
    begin
    if OpenPictureDialog1.Execute then
      begin
      if not (ofExtensionDifferent in OpenPictureDialog1.Options) then
        begin
        Application.Icon.LoadFromFile(OpenPictureDialog1.FileName);
        Done := True;
        end
      else
//      OpenPictureDialog1.Options := OpenPictureDialog1.Options - ofExtensionDifferent;
      end
    else { User cancelled }
      Done := True;
    end;
  end;
```

See Also

DefaultExt (see page 11421)

FileName (see page 11424)

TSaveDialog (see page 11489)

TSavePictureDialog (see page 13300)

61.1.17.5.10 TOpenDialog.OptionsEx Property

Augments the Options (see page 11436) property with additional flags that determine the appearance and behavior of the file-selection dialog.

Pascal

```
property OptionsEx: TOpenOptionsEx;
```

C++

```
__property TOpenOptionsEx OptionsEx;
```

Description

Use the OptionsEx property to further customize the file open dialog beyond the options covered by the Options (see page 11436) property.

See Also

Options (see page 11436)

61.1.17.5.11 TOpenDialog.Title Property

Specifies the text in the dialog's title bar.

Pascal

```
property Title: string;
```

C++

```
__property AnsiString Title;
```

Description

Use Title to specify the text that appears in the file-selection dialog's title bar. If no value is assigned to Title, the dialog has the title "Open".

C++ Examples:

```
/*
Click the button to open a TOpenDialog, then select a file
to delete.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenDialog1->Title = "Delete File";
  if (OpenDialog1->Execute())
  {
    if (FileExists(OpenDialog1->FileName))
      DeleteFile(OpenDialog1->FileName);
  }
}
```

Delphi Examples:

```
{
Click the button to open a TOpenDialog, then select a file
to delete.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  F: Textfile;
begin
  OpenDialog1.Title := 'Delete File';
  if OpenDialog1.Execute then
  begin
    AssignFile(F, OpenDialog1.FileName);
    try
      Reset(F);
      if MessageDlg('Erase ' + OpenDialog1.FileName + '?',
        mtConfirmation, [mbYes, mbNo], 0) = mrYes then
      begin
        CloseFile(F);
        Erase(F);
      end;
    except
      on EInOutError do
        MessageDlg('File I/O error.', mtError, [mbOk], 0);
    end;
  end;
end;
```

61.1.17.6 TOpenDialog Events

The events of the TOpenDialog class are listed here.

Published Events

	Name	Description
	OnCanClose (see page 11439)	Occurs when the user tries to close the dialog without canceling.
	OnFolderChange (see page 11440)	Occurs when a directory is opened or closed from the dialog.
	OnIncludeItem (see page 11440)	Occurs before the dialog adds a file to the file list box.
	OnSelectionChange (see page 11441)	Occurs when file names displayed in the dialog are changed.
	OnTypeChange (see page 11441)	Occurs when the file types displayed in the dialog are changed.

61.1.17.6.1 TOpenDialog.OnCanClose Event

Occurs when the user tries to close the dialog without canceling.

Pascal

```
property OnCanClose: TCloseQueryEvent;
```

C++

```
__property TCloseQueryEvent OnCanClose;
```

Description

Write an OnCanClose event handler to provide custom validation of the value of FileName (see page 11424). File selection dialogs provide a number of built-in validations, such as checking for invalid characters, prompting for confirmation before overwriting, checking whether a file or path exists, and so on. These validations can be specified using the Options (see page 11436) property. However, applications can provide additional validation of file names in an OnCanClose event handler.

Set the CanClose (see page 11413) parameter to false to prevent the dialog from closing. The OnCanClose event handler is responsible for telling the user why the dialog doesn't close.

Note: OnCanClose does not occur under Windows NT 3.51 unless the new shell is installed.

See Also

FileName (see page 11424)

OnClose (see page 11287)

Options (see page 11436)

61.1.17.6.2 TOpenDialog.OnFolderChange Event

Occurs when a directory is opened or closed from the dialog.

Pascal

```
property OnFolderChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnFolderChange;
```

Description

The OnFolderChange event occurs when the user changes the directory whose contents are displayed in the dialog. This can happen when the user double-clicks on a directory, clicks the Up arrow, or uses the list box at the top of the dialog to navigate through the directory structure.

61.1.17.6.3 TOpenDialog.OnIncludeItem Event

Occurs before the dialog adds a file to the file list box.

Pascal

```
property OnIncludeItem: TIncludeItemEvent;
```

C++

```
__property TIncludeItemEvent OnIncludeItem;
```

Description

Write an OnIncludeItem event handler to programmatically filter the items that appear in the shell folder's item list.

This event does not occur unless the Options (see page 11436) property includes ofEnableIncludeNotify.

See Also

Options (see page 11436)

61.1.17.6.4 TOpenDialog.OnSelectionChange Event

Occurs when file names displayed in the dialog are changed.

Pascal

```
property OnSelectionChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSelectionChange;
```

Description

The OnSelectionChange event occurs when the user does something to change the list displayed in the dialog. This can include opening the file-selection dialog, highlighting a file or directory, selecting a new filter, selecting a new directory, or creating a new folder.

See Also

FileName (see page 11424)

61.1.17.6.5 TOpenDialog.OnTypeChange Event

Occurs when the file types displayed in the dialog are changed.

Pascal

```
property OnTypeChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnTypeChange;
```

Description

The OnTypeChange event occurs when the user selects a new filter from the Files (see page 11429) of Type list box at the bottom of the dialog.

See Also

Filter (see page 11431)

61.1.18 TPageSetupDialog Class

TPageSetupDialog displays a page setup dialog.

Class Hierarchy

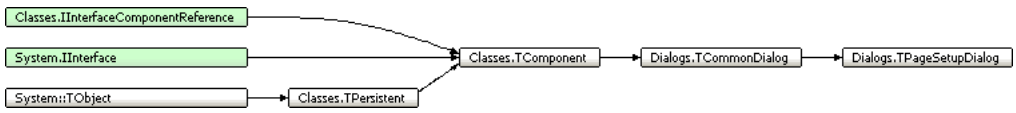

Pascal

```
TPageSetupDialog = class(TCommonDialog);
```

C++

```
class TPageSetupDialog : public TCommonDialog;
```

File

Dialogs

Description

TPageSetupDialog displays a modal Windows dialog box for setting up the print page. The dialog does not appear at runtime until it is activated by a call to the Execute (see page 11446) method. When the user selects a font and clicks OK, the dialog closes and the selected font is stored in the Font property.

The page setup dialog includes a sample page that is drawn when the dialog first appears and again as the user reconfigures the page setup. Each time the sample page is drawn, the following events occur. Defining handlers for these events allows you to customize the way the sample page is drawn. Events are listed in the order they occur:

Event	Occurs
BeforePaint (see page 11455)	Before drawing begins. Use to initialize values for other event handlers.
OnDrawFullPage (see page 11456)	Before drawing full page.
OnDrawMinMargin (see page 11457)	Before drawing minimum margins.
OnDrawMargin (see page 11457)	Before drawing margins.
OnDrawGreekText (see page 11456)	Before drawing "Greek" text.
OnDrawEnvStamp (see page 11455)	Before drawing stamp (envelope paper types only).
OnDrawRetAddress (see page 11458)	Before drawing return address (envelope paper types only).

Most of the event handlers can prevent some or all following event handlers from being called. Refer to individual event handler topics for more information.

61.1.18.1 TPageSetupDialog Members

The following tables list the members exposed by TPageSetupDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoClose (see page 11282)	Generates an OnClose (see page 11287) event.
	DoShow (see page 11283)	Generates an OnShow (see page 11288) event.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	MessageHook (see page 11284)	Provides special Windows message processing for the dialog.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	TaskModalDialog (see page 11284)	Ensures that Windows messages are properly routed when the dialog executes.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WndProc (see page 11285)	Responds to Windows messages sent to the dialog.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Template (see page 11287)	Points to a custom template for the dialog.
	TemplateModule (see page 11287)	This is TemplateModule, a member of class TCommonDialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.

	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11446)	Creates and initializes a TPageSetupDialog (see page 11441) instance.
	DefaultHandler (see page 11282)	Sends unhandled messages to Windows for processing.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 11281)	Destroys the TCommonDialog (see page 11277) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 11446)	Displays the page setup dialog.
	Execute (see page 11283)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetDefaults (see page 11446)	Displays the page setup dialog.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Handle (see page 11286)	Specifies the window handle of the dialog.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	PageSetupDlgRec (see page 11454)	Contain setup data structure.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	BeforePaint (see page 11455)	Occurs before dialog draws sample page.
	OnClose (see page 11287)	Occurs when the dialog closes.
	OnDrawEnvStamp (see page 11455)	Occurs before dialog draws stamp.
	OnDrawFullPage (see page 11456)	Occurs before any page drawing.
	OnDrawGreekText (see page 11456)	Occurs before drawing "Greek" text.
	OnDrawMargin (see page 11457)	Occurs before drawing margins.
	OnDrawMinMargin (see page 11457)	Occurs before drawing minimum margins.
	OnDrawRetAddress (see page 11458)	Occurs before dialog draws return address.
	OnShow (see page 11288)	Occurs when the dialog opens.

Published Properties

	Name	Description
	Ctl3D (see page 11285)	Determines whether the dialog has a three-dimensional look.
	HelpContext (see page 11286)	Specifies the context number for online Help.
	MarginBottom (see page 11447)	Specifies bottom margin.
	MarginLeft (see page 11448)	Specifies left margin
	MarginRight (see page 11448)	Specifies right margin.
	MarginTop (see page 11449)	Specifies top margin.
	MinMarginBottom (see page 11450)	Specifies minimum bottom margin.
	MinMarginLeft (see page 11450)	Specifies minimum left margin.
	MinMarginRight (see page 11451)	Specifies minimum right margin.
	MinMarginTop (see page 11452)	Specifies minimum top margin.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 11452)	Specifies dialog options.
	PageHeight (see page 11453)	Specifies page height.
	PageWidth (see page 11454)	Specifies page height.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Units (see page 11454)	Specifies units used by user.

61.1.18.2 TPageSetupDialog.Create Constructor

Creates and initializes a TPageSetupDialog (see page 11441) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TPageSetupDialog(TComponent * AOwner);
```

Description

The Create method generates a TPageSetupDialog (see page 11441) instance, but the new dialog does not appear on the form at runtime until the Execute (see page 11446) method is called.

AOwner is the component, typically a form or data module, that is responsible for freeing the TPageSetupDialog (see page 11441) instance. It becomes the value of the Owner property.

See Also

Owner (see page 2807)

Execute (see page 11446)

61.1.18.3 TPageSetupDialog Methods

The methods of the TPageSetupDialog class are listed here.

Public Methods

	Name	Description
	Execute (see page 11446)	Displays the page setup dialog.
	GetDefaults (see page 11446)	Displays the page setup dialog.

61.1.18.3.1 TPageSetupDialog.Execute Method

Displays the page setup dialog.

Pascal

```
function Execute(ParentWnd: HWND): Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute(HWND ParentWnd);
```

Description

Execute opens the page setup dialog. Execute returns true if the user completed the page setup operation, or false if the user canceled it.

61.1.18.3.2 TPageSetupDialog.GetDefaults Method

Displays the page setup dialog.

Pascal

```
function GetDefaults: Boolean;
```

C++

```
__fastcall Boolean GetDefaults();
```

Description

GetDefaults obtains the default settings for the current printer, initializes the printer, and sets the page margin options accordingly. GetDefaults returns true if this operation succeeds, false otherwise.

61.1.18.4 TPageSetupDialog Properties

The properties of the TPageSetupDialog class are listed here.

Public Properties

	Name	Description
R	PageSetupDlgRec (see page 11454)	Contain setup data structure.

Published Properties

	Name	Description
	MarginBottom (see page 11447)	Specifies bottom margin.
	MarginLeft (see page 11448)	Specifies left margin
	MarginRight (see page 11448)	Specifies right margin.
	MarginTop (see page 11449)	Specifies top margin.
	MinMarginBottom (see page 11450)	Specifies minimum bottom margin.
	MinMarginLeft (see page 11450)	Specifies minimum left margin.
	MinMarginRight (see page 11451)	Specifies minimum right margin.
	MinMarginTop (see page 11452)	Specifies minimum top margin.
	Options (see page 11452)	Specifies dialog options.
	PageHeight (see page 11453)	Specifies page height.
	PageWidth (see page 11454)	Specifies page height.
	Units (see page 11454)	Specifies units used by user.

61.1.18.4.1 TPageSetupDialog.MarginBottom Property

Specifies bottom margin.

Pascal

```
property MarginBottom: Integer;
```

C++

```
__property int MarginBottom;
```

Description

If Options (see page 11452) includes psoMargins, MarginBottom is the initial value presented to the user as the bottom margin. If the user clicks the OK button (indicated by Execute (see page 11446) returning true), MarginBottom is assigned the bottom margin value specified by the user.

If the user is entering values in inches, MarginBottom expresses the margin in thousandths of an inch. If the user is entering values in millimeters, MarginBottom expresses the margin in hundredths of a millimeter.

See Also

MarginLeft (see page 11448)

MarginRight (see page 11448)

MarginTop (see page 11449)

MinMarginLeft (see page 11450)

MinMarginRight (see page 11451)

MinMarginBottom (see page 11450)

MinMarginTop (see page 11452)

Execute (see page 11446)

Options (see page 11452)

Units (see page 11454)

61.1.18.4.2 TPageSetupDialog.MarginLeft Property

Specifies left margin

Pascal

```
property MarginLeft: Integer;
```

C++

```
__property int MarginLeft;
```

Description

If Options (see page 11452) includes psoMargins, MarginLeft is the initial value presented to the user as the left margin. If the user clicks the OK button (indicated by Execute (see page 11446) returning true), MarginLeft is assigned the left margin value specified by the user.

If the user is entering values in inches, MarginLeft expresses the margin in thousandths of an inch. If the user is entering values in millimeters, MarginLeft expresses the margin in hundredths of a millimeter.

See Also

MarginBottom (see page 11447)

MarginRight (see page 11448)

MarginTop (see page 11449)

MinMarginLeft (see page 11450)

MinMarginRight (see page 11451)

MinMarginLeft (see page 11450)

MinMarginTop (see page 11452)

Execute (see page 11446)

Options (see page 11452)

Units (see page 11454)

61.1.18.4.3 TPageSetupDialog.MarginRight Property

Specifies right margin.

Pascal

```
property MarginRight: Integer;
```

C++

```
__property int MarginRight;
```

Description

If Options (see page 11452) includes psoMargins, MarginRight is the initial value presented to the user as the right margin. If the user clicks the OK button (indicated by Execute (see page 11446) returning true), MarginRight is assigned the right margin value specified by the user.

If the user is entering values in inches, MarginRight expresses the margin in thousandths of an inch. If the user is entering values in millimeters, MarginRight expresses the margin in hundredths of a millimeter.

See Also

MarginLeft (see page 11448)

MarginBottom (see page 11447)

MarginTop (see page 11449)

MinMarginLeft (see page 11450)

MinMarginRight (see page 11451)

MinMarginRight (see page 11451)

MinMarginTop (see page 11452)

Execute (see page 11446)

Options (see page 11452)

Units (see page 11454)

61.1.18.4.4 TPageSetupDialog.MarginTop Property

Specifies top margin.

Pascal

```
property MarginTop: Integer;
```

C++

```
__property int MarginTop;
```

Description

If Options (see page 11452) includes psoMargins, MarginTop is the initial value presented to the user as the top margin. If the user clicks the OK button (indicated by Execute (see page 11446) returning true), MarginTop is assigned the top margin value specified by the user.

If the user is entering values in inches, MarginTop expresses the margin in thousandths of an inch. If the user is entering values in millimeters, MarginTop expresses the margin in hundredths of a millimeter.

See Also

MarginLeft (see page 11448)

MarginRight (see page 11448)

MarginBottom (see page 11447)

MinMarginLeft (see page 11450)

MinMarginRight (see page 11451)

MinMarginTop (see page 11452)

MinMarginTop (see page 11452)

Execute (see page 11446)

Options (see page 11452)

Units (see page 11454)

61.1.18.4.5 TPageSetupDialog.MinMarginBottom Property

Specifies minimum bottom margin.

Pascal

```
property MinMarginBottom: Integer;
```

C++

```
__property int MinMarginBottom;
```

Description

If Options (see page 11452) includes psoMinMargins, MinMarginBottom is the minimum value the user can enter for the bottom margin.

If the user is entering values in inches, MinMarginBottom expresses the minimum margin in thousandths of an inch. If the user is entering values in millimeters, MinMarginBottom expresses the minimum margin in hundredths of a millimeter.

See Also

MarginLeft (see page 11448)

MarginRight (see page 11448)

MarginTop (see page 11449)

MinMarginLeft (see page 11450)

MinMarginRight (see page 11451)

MarginBottom (see page 11447)

MinMarginTop (see page 11452)

Execute (see page 11446)

Options (see page 11452)

Units (see page 11454)

61.1.18.4.6 TPageSetupDialog.MinMarginLeft Property

Specifies minimum left margin.

Pascal

```
property MinMarginLeft: Integer;
```

C++

```
__property int MinMarginLeft;
```

Description

If Options (see page 11452) includes psoMinMargins, MinMarginLeft is the minimum value the user can enter for the left

margin.

If the user is entering values in inches, MinMarginLeft expresses the minimum margin in thousandths of an inch. If the user is entering values in millimeters, MinMarginLeft expresses the minimum margin in hundredths of a millimeter.

See Also

MarginLeft (see page 11448)

MarginRight (see page 11448)

MarginTop (see page 11449)

MinMarginBottom (see page 11450)

MinMarginRight (see page 11451)

MarginBottom (see page 11447)

MinMarginTop (see page 11452)

Execute (see page 11446)

Options (see page 11452)

Units (see page 11454)

61.1.18.4.7 TPageSetupDialog.MinMarginRight Property

Specifies minimum right margin.

Pascal

```
property MinMarginRight: Integer;
```

C++

```
__property int MinMarginRight;
```

Description

If Options (see page 11452) includes psoMinMargins, MinMarginRight is the minimum value the user can enter for the right margin.

If the user is entering values in inches, MinMarginRight expresses the minimum margin in thousandths of an inch. If the user is entering values in millimeters, MinMarginRight expresses the minimum margin in hundredths of a millimeter.

See Also

MarginLeft (see page 11448)

MarginRight (see page 11448)

MarginTop (see page 11449)

MinMarginLeft (see page 11450)

MinMarginBottom (see page 11450)

MarginBottom (see page 11447)

MinMarginTop (see page 11452)

Execute (see page 11446)

Options (see page 11452)

Units (see page 11454)

61.1.18.4.8 TPageSetupDialog.MinMarginTop Property

Specifies minimum top margin.

Pascal

```
property MinMarginTop: Integer;
```

C++

```
__property int MinMarginTop;
```

Description

If Options (see page 11452) includes psoMinMargins, MinMarginTop is the minimum value the user can enter for the top margin.

If the user is entering values in inches, MinMarginTop expresses the minimum margin in thousandths of an inch. If the user is entering values in millimeters, MinMarginTop expresses the minimum margin in hundredths of a millimeter.

See Also

MarginLeft (see page 11448)

MarginRight (see page 11448)

MarginTop (see page 11449)

MinMarginLeft (see page 11450)

MinMarginRight (see page 11451)

MarginBottom (see page 11447)

MinMarginBottom (see page 11450)

Execute (see page 11446)

Options (see page 11452)

Units (see page 11454)

61.1.18.4.9 TPageSetupDialog.Options Property

Specifies dialog options.

Pascal

```
property Options: TPageSetupDialogOptions;
```

C++

```
__property TPageSetupDialogOptions Options;
```

Description

Options specifies the display options for the printer dialog. These are:

Option	Meaning
psoDefaultMinMargins	Sets the minimum values that the user can specify for the page margins to be the minimum margins allowed by the printer.
psoDisableMargins	Disables the margin controls, preventing the user from setting the margins.
psoDisableOrientation	Disables the orientation controls, preventing the user from setting the page orientation.

psoDisablePagePainting	Prevents default sample page drawing. Does not disable event calls.
psoDisablePaper	Disables the paper controls, preventing the user from setting page parameters such as the paper size and source.
psoDisablePrinter	Disables the Printer button, preventing the user from invoking a dialog box that contains additional printer setup information.
psoMargins	Sets the initial margins from MarginLeft (see page 11448), MarginRight (see page 11448), MarginBottom (see page 11447), and MarginTop (see page 11449) properties. Otherwise, initial margins are one inch each.
psoMinMargins	Sets the minimum margins from MinMarginLeft (see page 11450), MinMarginRight (see page 11451), MinMarginBottom (see page 11450), and MinMarginTop (see page 11452). Otherwise, minimum margins are determined by the printer driver.
psoShowHelp	Displays Help button and Help title bar button.
psoWarning	Prevents display of warning when there is no default printer.
psoNoNetworkButton	Hides and disables Network button.

See Also

MarginLeft (see page 11448)

MarginRight (see page 11448)

MarginBottom (see page 11447)

MarginTop (see page 11449)

MinMarginLeft (see page 11450)

MinMarginRight (see page 11451)

MinMarginBottom (see page 11450)

MinMarginTop (see page 11452)

61.1.18.4.10 TPageSetupDialog.PageHeight Property

Specifies page height.

Pascal

```
property PageHeight: Integer;
```

C++

```
__property int PageHeight;
```

Description

PageHeight contains the height of the page type selected by the user.

If the user is entering values in inches, PageHeight expresses the page height in thousandths of an inch. If the user is entering values in millimeters, PageHeight expresses the page height in hundredths of a millimeter.

See Also

PageWidth (see page 11454)

61.1.18.4.11 TPageSetupDialog.PageSetupDlgRec Property

Contain setup data structure.

Pascal

```
property PageSetupDlgRec: TPageSetupDlg;
```

C++

```
__property TPageSetupDlg PageSetupDlgRec;
```

Description

PageSetupDlgRec contains the Windows page setup data structure.

61.1.18.4.12 TPageSetupDialog.PageWidth Property

Specifies page height.

Pascal

```
property PageWidth: Integer;
```

C++

```
__property int PageWidth;
```

Description

PageWidth contains the width of the page type selected by the user.

If the user is entering values in inches, PageWidth expresses the page width in thousandths of an inch. If the user is entering values in millimeters, PageWidth expresses the page width in hundredths of a millimeter.

See Also

PageHeight (see page 11453)

61.1.18.4.13 TPageSetupDialog.Units Property

Specifies units used by user.

Pascal

```
property Units: TPageMeasureUnits;
```

C++

```
__property TPageMeasureUnits Units;
```

Description

Units determines the measurement units used by the user. Possible settings:

Constant	Meaning
pmDefault	User must specify values in units determined by locale settings.
pmMillimeters	User must specify values in millimeters.
pmInches	User must specify values in inches.

61.1.18.5 TPageSetupDialog Events

The events of the TPageSetupDialog class are listed here.

Published Events

	Name	Description
	BeforePaint (see page 11455)	Occurs before dialog draws sample page.
	OnDrawEnvStamp (see page 11455)	Occurs before dialog draws stamp.
	OnDrawFullPage (see page 11456)	Occurs before any page drawing.
	OnDrawGreekText (see page 11456)	Occurs before drawing "Greek" text.
	OnDrawMargin (see page 11457)	Occurs before drawing margins.
	OnDrawMinMargin (see page 11457)	Occurs before drawing minimum margins.
	OnDrawRetAddress (see page 11458)	Occurs before dialog draws return address.

61.1.18.5.1 TPageSetupDialog.BeforePaint Event

Occurs before dialog draws sample page.

Pascal

```
property BeforePaint: TPageSetupBeforePaintEvent;
```

C++

```
__property TPageSetupBeforePaintEvent BeforePaint;
```

Description

Write an BeforePaint event handler to initialize data used by the other event handlers.

PaperSize indicates a standard Windows paper size. See Paper size constants.

Orientation indicates the paper orientation currently selected.

PageType indicates the kind of paper currently selected.

Set DoneDrawing to true to disable further painting. This results in a blank sample page, with none of the other event handlers being called.

61.1.18.5.2 TPageSetupDialog.OnDrawEnvStamp Event

Occurs before dialog draws stamp.

Pascal

```
property OnDrawEnvStamp: TPaintPageEvent;
```

C++

```
__property TPaintPageEvent OnDrawEnvStamp;
```

Description

Write an OnDrawEnvStamp event handler to customize the way the page setup dialog draws the stamp. The PageRect parameter indicates the boundaries of the stamp.

If the handler sets the DoneDrawing parameter to true, default stamp drawing is suppressed.

OnDrawEnvStamp only occurs for envelope paper types.

61.1.18.5.3 TPageSetupDialog.OnDrawFullPage Event

Occurs before any page drawing.

Pascal

```
property OnDrawFullPage: TPaintPageEvent;
```

C++

```
__property TPaintPageEvent OnDrawFullPage;
```

Description

Write an OnDrawFullPage event handler to customize the way the entire page is drawn. The PageRect parameter indicates the boundaries of the page.

If the handler sets the DoneDrawing parameter to true, all default page drawing is suppressed and none of the subsequent event handlers (OnDrawMinMargin (see page 11457), OnDrawMargin (see page 11457), OnDrawGreekText (see page 11456), OnDrawEnvStamp (see page 11455), and OnDrawRetAddress (see page 11458)) are called. All drawing of the sample page is done by the OnDrawFullPage event handler.

See Also

OnDrawMinMargin (see page 11457)

OnDrawMargin (see page 11457)

OnDrawGreekText (see page 11456)

OnDrawEnvStamp (see page 11455)

OnDrawRetAddress (see page 11458)

61.1.18.5.4 TPageSetupDialog.OnDrawGreekText Event

Occurs before drawing "Greek" text.

Pascal

```
property OnDrawGreekText: TPaintPageEvent;
```

C++

```
__property TPaintPageEvent OnDrawGreekText;
```

Description

Write an OnDrawGreekText event handler to customize the way "Greek" text is drawn. The PageRect parameter indicates the current margins of the page.

If the handler sets the DoneDrawing parameter to true, default drawing for the margins and "Greek" text is suppressed. This does not suppress special drawing features for envelope paper types: drawing the return address or stamp, and calling the OnDrawEnvStamp (see page 11455), and OnDrawRetAddress (see page 11458) event handlers.

See Also

OnDrawEnvStamp (see page 11455)

OnDrawRetAddress (see page 11458)

61.1.18.5.5 TPageSetupDialog.OnDrawMargin Event

Occurs before drawing margins.

Pascal

```
property OnDrawMargin: TPaintPageEvent;
```

C++

```
__property TPaintPageEvent OnDrawMargin;
```

Description

Write an OnDrawMargin event handler to customize the way the margins are drawn. The PageRect parameter indicates the current margins of the page.

If the handler sets the DoneDrawing parameter to true, default drawing for the margins and "Greek" text is suppressed and the OnDrawGreekText (see page 11456) event handler is not called. However, this does not suppress special drawing features for envelope paper types: drawing the return address or stamp, and calling the OnDrawEnvStamp (see page 11455), and OnDrawRetAddress (see page 11458) event handlers.

See Also

OnDrawGreekText (see page 11456)

OnDrawEnvStamp (see page 11455)

OnDrawRetAddress (see page 11458)

61.1.18.5.6 TPageSetupDialog.OnDrawMinMargin Event

Occurs before drawing minimum margins.

Pascal

```
property OnDrawMinMargin: TPaintPageEvent;
```

C++

```
__property TPaintPageEvent OnDrawMinMargin;
```

Description

Write an OnDrawMinMargin event handler to customize the way the minimum margins are drawn. The PageRect parameter indicates the minimum margins of the page.

If the handler sets the DoneDrawing parameter to true, default drawing for the minimum margins, margins, and "Greek" text is suppressed and the OnDrawMargin (see page 11457) and OnDrawGreekText (see page 11456) event handlers are not called. However, this does not suppress special drawing features for envelope paper types: drawing the return address or stamp, and calling the OnDrawEnvStamp (see page 11455), and OnDrawRetAddress (see page 11458) event handlers.

See Also

OnDrawMargin (see page 11457)

OnDrawGreekText (see page 11456)

OnDrawEnvStamp (see page 11455)

OnDrawRetAddress (see page 11458)

61.1.18.5.7 TPageSetupDialog.OnDrawRetAddress Event

Occurs before dialog draws return address.

Pascal

```
property OnDrawRetAddress: TPaintPageEvent;
```

C++

```
__property TPaintPageEvent OnDrawRetAddress;
```

Description

Write an OnDrawRetAddress event handler to customize the way the page setup dialog draws the return address. The PageRect parameter indicates the boundaries of the return address

If the handler sets the DoneDrawing parameter to true, default return address drawing is suppressed.

OnDrawRetAddress only occurs for envelope paper types.

61.1.19 TPrintDialog Class

TPrintDialog displays a Print dialog.

Class Hierarchy

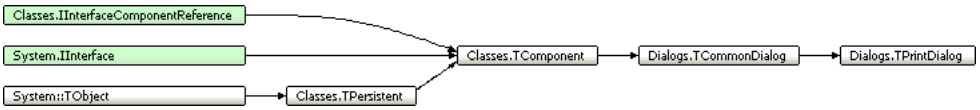

Pascal

```
TPrintDialog = class(TCommonDialog);
```

C++

```
class TPrintDialog : public TCommonDialog;
```

File

Dialogs

Description

The TPrintDialog component displays a standard Windows dialog box for sending jobs to a printer. The dialog is modal and does not appear at runtime until it is activated by a call to the Execute (see page 11462) method.

See Also

TPrinterSetupDialog (see page 11478)

61.1.19.1 TPrintDialog Members

The following tables list the members exposed by TPrintDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoClose (see page 11282)	Generates an OnClose (see page 11287) event.
	DoShow (see page 11283)	Generates an OnShow (see page 11288) event.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	MessageHook (see page 11284)	Provides special Windows message processing for the dialog.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	TaskModalDialog (see page 11284)	Ensures that Windows messages are properly routed when the dialog executes.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WndProc (see page 11285)	Responds to Windows messages sent to the dialog.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Template (see page 11287)	Points to a custom template for the dialog.

	TemplateModule (see page 11287)	This is TemplateModule, a member of class TCommonDialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 11281)	Creates and initializes a TCommonDialog (see page 11277) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 11282)	Sends unhandled messages to Windows for processing.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 11281)	Destroys the TCommonDialog (see page 11277) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Execute (see page 11462)	Displays the Print dialog box.
V	Execute (see page 11283)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Handle (see page 11286)	Specifies the window handle of the dialog.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnClose (see page 11287)	Occurs when the dialog closes.
	OnShow (see page 11288)	Occurs when the dialog opens.

Published Properties

	Name	Description
	Collate (see page 11464)	Indicates whether the Collate check box is selected.
	Copies (see page 11464)	Indicates the number copies selected in the Print dialog.
	Ctl3D (see page 11285)	Determines whether the dialog has a three-dimensional look.
	FromPage (see page 11465)	Indicates the page on which the print job is to begin.
	HelpContext (see page 11286)	Specifies the context number for online Help.
	MaxPage (see page 11467)	Determines the greatest page number the user can enter when specifying pages to print.
	MinPage (see page 11469)	Determines the smallest page number the user can enter when specifying pages to print.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 11471)	Determines the appearance and behavior of the Print dialog.
	PrintRange (see page 11473)	Indicates the type of print range selected in the dialog.
	PrintToFile (see page 11475)	Indicates whether the Print To File check box is selected.
	Tag (see page 2808)	Stores an integer value as part of a component.
	ToPage (see page 11476)	Indicates the page on which the print job is to end.

61.1.19.2 TPrintDialog Methods

The methods of the TPrintDialog class are listed here.

Public Methods

	Name	Description
	Execute (see page 11462)	Displays the Print dialog box.

61.1.19.2.1 TPrintDialog.Execute Method

Displays the Print dialog box.

Pascal

```
function Execute(ParentWnd: HWND): Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute(HWND ParentWnd);
```

Description

Execute opens the Print dialog, returning true when the user clicks Print and false when the user clicks Cancel.

C++ Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PrintDialog1->Options.Clear();
  PrintDialog1->Options << poPageNums << poSelection;
  PrintDialog1->FromPage = 1;
  PrintDialog1->MinPage = 1;
  PrintDialog1->ToPage = PageControl1->PageCount;
  PrintDialog1->MaxPage = PageControl1->PageCount;
  if (PrintDialog1->Execute())
  {
    int Start, Stop;
    // determine the range the user wants to print
    switch (PrintDialog1->PrintRange)
    {
      case prSelection:
        Start = PageControl1->ActivePage->PageIndex;
        Stop = Start;
        break;
      case prPageNums:
        Start = PrintDialog1->FromPage - 1;
        Stop =  PrintDialog1->ToPage - 1;
        break;
      default:  // prAllPages
        Start = PrintDialog1->MinPage - 1;
        Stop = PrintDialog1->MaxPage - 1;
        break;
    }
```

```
    // now, print the pages
    Printer()->BeginDoc();
    for (int i = Start; i <= Stop; i++)
    {
      PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
      if (i != Stop)
        Printer()->NewPage();
    }
    Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
}
procedure TForm1.Button1Click(Sender:TObject);
var
  I, Start, Stop: Integer;
begin
PrintDialog1.Options := [poPageNums, poSelection];
PrintDialog1.FromPage := 1;
PrintDialog1.MinPage := 1;
PrintDialog1.ToPage := PageControl1.PageCount;
PrintDialog1.MaxPage := PageControl1.PageCount;

if PrintDialog1.Execute() then
  begin
    { determine the range the user wants to print }
    with PrintDialog1 do
    begin
      if PrintRange = prAllPages then
        begin
        Start := MinPage - 1;
        Stop := MaxPage - 1;
      end
      else if PrintRange = prSelection then
      begin
        Start := PageControl1.ActivePage.PageIndex;
        Stop := Start;
        end
      else  { PrintRange = prPageNums }
      begin
        Start := FromPage - 1;
        Stop := ToPage - 1;
      end;
    end;
    { now, print the pages }
    with Printer do
    begin
      BeginDoc;
      for I := Start to Stop do
      begin
        PageControl1.Pages[I].PaintTo(Handle, 10, 10);
        if I <> Stop then
          NewPage;
      end;
      EndDoc;
    end;
```

```
  end;
end;
```

61.1.19.3 TPrintDialog Properties

The properties of the TPrintDialog class are listed here.

Published Properties

	Name	Description
	Collate (see page 11464)	Indicates whether the Collate check box is selected.
	Copies (see page 11464)	Indicates the number copies selected in the Print dialog.
	FromPage (see page 11465)	Indicates the page on which the print job is to begin.
	MaxPage (see page 11467)	Determines the greatest page number the user can enter when specifying pages to print.
	MinPage (see page 11469)	Determines the smallest page number the user can enter when specifying pages to print.
	Options (see page 11471)	Determines the appearance and behavior of the Print dialog.
	PrintRange (see page 11473)	Indicates the type of print range selected in the dialog.
	PrintToFile (see page 11475)	Indicates whether the Print To File check box is selected.
	ToPage (see page 11476)	Indicates the page on which the print job is to end.

61.1.19.3.1 TPrintDialog.Collate Property

Indicates whether the Collate check box is selected.

Pascal

```
property Collate: Boolean;
```

C++

```
__property Boolean Collate;
```

Description

Collate is true whenever the Collate check box is selected in the Print dialog. To make the dialog open with the check box selected, set Collate to true in the Object Inspector or in program code.

See Also

Copies (see page 11464)

61.1.19.3.2 TPrintDialog.Copies Property

Indicates the number copies selected in the Print dialog.

Pascal

```
property Copies: Integer;
```

C++

```
__property int Copies;
```

Description

If Copies is 0 or 1, the dialog will have 1 in the Number of Copies field. To make the dialog open with 2 or more copies selected, set the value of Copies in the Object Inspector or in program code.

See Also

Collate (see page 11464)

61.1.19.3.3 TPrintDialog.FromPage Property

Indicates the page on which the print job is to begin.

Pascal

```
property FromPage: Integer;
```

C++

```
__property int FromPage;
```

Description

The FromPage property corresponds to the From field in the Print dialog.

C++ Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PrintDialog1->Options.Clear();
  PrintDialog1->Options << poPageNums << poSelection;
  PrintDialog1->FromPage = 1;
  PrintDialog1->MinPage = 1;
  PrintDialog1->ToPage = PageControl1->PageCount;
  PrintDialog1->MaxPage = PageControl1->PageCount;
  if (PrintDialog1->Execute())
  {
    int Start, Stop;
    // determine the range the user wants to print
    switch (PrintDialog1->PrintRange)
    {
      case prSelection:
        Start = PageControl1->ActivePage->PageIndex;
        Stop = Start;
        break;
      case prPageNums:
        Start = PrintDialog1->FromPage - 1;
        Stop =  PrintDialog1->ToPage - 1;
        break;
      default:  // prAllPages
        Start = PrintDialog1->MinPage - 1;
        Stop = PrintDialog1->MaxPage - 1;
        break;
    }
    // now, print the pages
    Printer()->BeginDoc();
    for (int i = Start; i <= Stop; i++)
    {
      PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
      if (i != Stop)
        Printer()->NewPage();
    }
    Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
}
procedure TForm1.Button1Click(Sender:TObject);
var
  I, Start, Stop: Integer;
begin
PrintDialog1.Options := [poPageNums, poSelection];
PrintDialog1.FromPage := 1;
PrintDialog1.MinPage := 1;
PrintDialog1.ToPage := PageControl1.PageCount;
PrintDialog1.MaxPage := PageControl1.PageCount;

if PrintDialog1.Execute() then
  begin
    { determine the range the user wants to print }
    with PrintDialog1 do
    begin
      if PrintRange = prAllPages then
        begin
        Start := MinPage - 1;
        Stop := MaxPage - 1;
      end
      else if PrintRange = prSelection then
      begin
        Start := PageControl1.ActivePage.PageIndex;
        Stop := Start;
        end
      else  { PrintRange = prPageNums }
      begin
        Start := FromPage - 1;
        Stop := ToPage - 1;
      end;
    end;
    { now, print the pages }
    with Printer do
    begin
      BeginDoc;
      for I := Start to Stop do
      begin
        PageControl1.Pages[I].PaintTo(Handle, 10, 10);
        if I <> Stop then
          NewPage;
      end;
      EndDoc;
    end;
  end;
end;
```

See Also

MaxPage (see page 11467)

MinPage (see page 11469)

Options (see page 11471)

PrintRange (see page 11473)

ToPage (see page 11476)

61.1.19.3.4 TPrintDialog.MaxPage Property

Determines the greatest page number the user can enter when specifying pages to print.

Pascal

```
property MaxPage: Integer;
```

C++

```
__property int MaxPage;
```

Description

If the user tries to enter a number greater than MaxPage, an error message appears.

Note: The user can specify page numbers only if the poPageNums flag is set in Options (see page 11471). If poPageNums is on, MaxPage must be set to a value greater than MinPage (see page 11469) or the Print dialog will not open.

C++ Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PrintDialog1->Options.Clear();
  PrintDialog1->Options << poPageNums << poSelection;
  PrintDialog1->FromPage = 1;
  PrintDialog1->MinPage = 1;
  PrintDialog1->ToPage = PageControl1->PageCount;
  PrintDialog1->MaxPage = PageControl1->PageCount;
  if (PrintDialog1->Execute())
  {
    int Start, Stop;
    // determine the range the user wants to print
    switch (PrintDialog1->PrintRange)
    {
      case prSelection:
        Start = PageControl1->ActivePage->PageIndex;
        Stop = Start;
        break;
      case prPageNums:
        Start = PrintDialog1->FromPage - 1;
        Stop =  PrintDialog1->ToPage - 1;
        break;
      default:  // prAllPages
        Start = PrintDialog1->MinPage - 1;
        Stop = PrintDialog1->MaxPage - 1;
        break;
    }
    // now, print the pages
    Printer()->BeginDoc();
    for (int i = Start; i <= Stop; i++)
    {
```

```
      PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
      if (i != Stop)
        Printer()->NewPage();
    }
    Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
}
procedure TForm1.Button1Click(Sender:TObject);
var
  I, Start, Stop: Integer;
begin
PrintDialog1.Options := [poPageNums, poSelection];
PrintDialog1.FromPage := 1;
PrintDialog1.MinPage := 1;
PrintDialog1.ToPage := PageControl1.PageCount;
PrintDialog1.MaxPage := PageControl1.PageCount;

if PrintDialog1.Execute() then
  begin
    { determine the range the user wants to print }
    with PrintDialog1 do
    begin
      if PrintRange = prAllPages then
        begin
        Start := MinPage - 1;
        Stop := MaxPage - 1;
      end
      else if PrintRange = prSelection then
      begin
        Start := PageControl1.ActivePage.PageIndex;
        Stop := Start;
        end
      else  { PrintRange = prPageNums }
      begin
        Start := FromPage - 1;
        Stop := ToPage - 1;
      end;
    end;
    { now, print the pages }
    with Printer do
    begin
      BeginDoc;
      for I := Start to Stop do
      begin
        PageControl1.Pages[I].PaintTo(Handle, 10, 10);
        if I <> Stop then
          NewPage;
      end;
      EndDoc;
    end;
  end;
end;
```

See Also

FromPage (see page 11465)

MinPage (see page 11469)

Options (see page 11471)

ToPage (see page 11476)

61.1.19.3.5 TPrintDialog.MinPage Property

Determines the smallest page number the user can enter when specifying pages to print.

Pascal

```
property MinPage: Integer;
```

C++

```
__property int MinPage;
```

Description

If the user tries to enter a number smaller than MinPage, a error message appears.

Note: The user can specify page numbers only if the poPageNums flag is set in Options (see page 11471). If poPageNums is on, MaxPage (see page 11467) must be set to a value greater than MinPage or the Print dialog will not open.

C++ Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PrintDialog1->Options.Clear();
  PrintDialog1->Options << poPageNums << poSelection;
  PrintDialog1->FromPage = 1;
  PrintDialog1->MinPage = 1;
  PrintDialog1->ToPage = PageControl1->PageCount;
  PrintDialog1->MaxPage = PageControl1->PageCount;
  if (PrintDialog1->Execute())
  {
    int Start, Stop;
    // determine the range the user wants to print
    switch (PrintDialog1->PrintRange)
    {
      case prSelection:
        Start = PageControl1->ActivePage->PageIndex;
        Stop = Start;
        break;
      case prPageNums:
        Start = PrintDialog1->FromPage - 1;
        Stop =  PrintDialog1->ToPage - 1;
        break;
      default:  // prAllPages
```

```
        Start = PrintDialog1->MinPage - 1;
        Stop = PrintDialog1->MaxPage - 1;
        break;
    }
    // now, print the pages
    Printer()->BeginDoc();
    for (int i = Start; i <= Stop; i++)
    {
      PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
      if (i != Stop)
        Printer()->NewPage();
    }
    Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
}
procedure TForm1.Button1Click(Sender:TObject);
var
  I, Start, Stop: Integer;
begin
PrintDialog1.Options := [poPageNums, poSelection];
PrintDialog1.FromPage := 1;
PrintDialog1.MinPage := 1;
PrintDialog1.ToPage := PageControl1.PageCount;
PrintDialog1.MaxPage := PageControl1.PageCount;

if PrintDialog1.Execute() then
  begin
    { determine the range the user wants to print }
    with PrintDialog1 do
    begin
      if PrintRange = prAllPages then
        begin
        Start := MinPage - 1;
        Stop := MaxPage - 1;
      end
      else if PrintRange = prSelection then
      begin
        Start := PageControl1.ActivePage.PageIndex;
        Stop := Start;
        end
      else  { PrintRange = prPageNums }
      begin
        Start := FromPage - 1;
        Stop := ToPage - 1;
      end;
    end;
    { now, print the pages }
    with Printer do
    begin
      BeginDoc;
      for I := Start to Stop do
      begin
        PageControl1.Pages[I].PaintTo(Handle, 10, 10);
        if I <> Stop then
```

```
            NewPage;
        end;
        EndDoc;
      end;
    end;
  end;
```

See Also

FromPage (see page 11465)

MaxPage (see page 11467)

Options (see page 11471)

ToPage (see page 11476)

61.1.19.3.6 TPrintDialog.Options Property

Determines the appearance and behavior of the Print dialog.

Pascal

```
property Options: TPrintDialogOptions;
```

C++

```
__property TPrintDialogOptions Options;
```

Description

Use Options to specify which controls appear in the dialog and which controls appear but are disabled. By default, all options flags are off.

C++ Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PrintDialog1->Options.Clear();
  PrintDialog1->Options << poPageNums << poSelection;
  PrintDialog1->FromPage = 1;
  PrintDialog1->MinPage = 1;
  PrintDialog1->ToPage = PageControl1->PageCount;
  PrintDialog1->MaxPage = PageControl1->PageCount;
  if (PrintDialog1->Execute())
  {
    int Start, Stop;
    // determine the range the user wants to print
    switch (PrintDialog1->PrintRange)
    {
      case prSelection:
        Start = PageControl1->ActivePage->PageIndex;
        Stop = Start;
        break;
      case prPageNums:
        Start = PrintDialog1->FromPage - 1;
        Stop =  PrintDialog1->ToPage - 1;
```

```
        break;
      default:  // prAllPages
        Start = PrintDialog1->MinPage - 1;
        Stop = PrintDialog1->MaxPage - 1;
        break;
    }
    // now, print the pages
    Printer()->BeginDoc();
    for (int i = Start; i <= Stop; i++)
    {
      PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
      if (i != Stop)
        Printer()->NewPage();
    }
    Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
}
procedure TForm1.Button1Click(Sender:TObject);
var
  I, Start, Stop: Integer;
begin
PrintDialog1.Options := [poPageNums, poSelection];
PrintDialog1.FromPage := 1;
PrintDialog1.MinPage := 1;
PrintDialog1.ToPage := PageControl1.PageCount;
PrintDialog1.MaxPage := PageControl1.PageCount;

if PrintDialog1.Execute() then
  begin
    { determine the range the user wants to print }
    with PrintDialog1 do
    begin
      if PrintRange = prAllPages then
        begin
        Start := MinPage - 1;
        Stop := MaxPage - 1;
      end
      else if PrintRange = prSelection then
      begin
        Start := PageControl1.ActivePage.PageIndex;
        Stop := Start;
        end
      else  { PrintRange = prPageNums }
      begin
        Start := FromPage - 1;
        Stop := ToPage - 1;
      end;
    end;
    { now, print the pages }
    with Printer do
    begin
      BeginDoc;
      for I := Start to Stop do
      begin
```

```
          PageControl1.Pages[I].PaintTo(Handle, 10, 10);
          if I <> Stop then
            NewPage;
        end;
        EndDoc;
      end;
    end;
  end;
```

See Also

FromPage (see page 11465)

MaxPage (see page 11467)

MinPage (see page 11469)

ToPage (see page 11476)

61.1.19.3.7 TPrintDialog.PrintRange Property

Indicates the type of print range selected in the dialog.

Pascal

```
property PrintRange: TPrintRange;
```

C++

```
__property TPrintRange PrintRange;
```

Description

The value of PrintRange corresponds the All, Selection, and Pages (From/To) radio buttons in the Print dialog. To determine which radio button is selected by default when the dialog opens, set the value of PrintRange in the Object Inspector or in program code.

To enable the Pages radio button, set the poPageNums flag in Options (see page 11471). To enable the Selection radio button, set the poSelection flag in Options (see page 11471) and set MaxPage (see page 11467) to a greater value than MinPage (see page 11469).

C++ Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PrintDialog1->Options.Clear();
  PrintDialog1->Options << poPageNums << poSelection;
  PrintDialog1->FromPage = 1;
  PrintDialog1->MinPage = 1;
  PrintDialog1->ToPage = PageControl1->PageCount;
  PrintDialog1->MaxPage = PageControl1->PageCount;
  if (PrintDialog1->Execute())
  {
    int Start, Stop;
    // determine the range the user wants to print
    switch (PrintDialog1->PrintRange)
```

```
      {
        case prSelection:
          Start = PageControl1->ActivePage->PageIndex;
          Stop = Start;
          break;
        case prPageNums:
          Start = PrintDialog1->FromPage - 1;
          Stop =  PrintDialog1->ToPage - 1;
          break;
        default:  // prAllPages
          Start = PrintDialog1->MinPage - 1;
          Stop = PrintDialog1->MaxPage - 1;
          break;
      }
      // now, print the pages
      Printer()->BeginDoc();
      for (int i = Start; i <= Stop; i++)
      {
        PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
        if (i != Stop)
          Printer()->NewPage();
      }
      Printer()->EndDoc();
    }
  }
```

Delphi Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
}
procedure TForm1.Button1Click(Sender:TObject);
var
  I, Start, Stop: Integer;
begin
PrintDialog1.Options := [poPageNums, poSelection];
PrintDialog1.FromPage := 1;
PrintDialog1.MinPage := 1;
PrintDialog1.ToPage := PageControl1.PageCount;
PrintDialog1.MaxPage := PageControl1.PageCount;

if PrintDialog1.Execute() then
  begin
    { determine the range the user wants to print }
    with PrintDialog1 do
    begin
      if PrintRange = prAllPages then
        begin
        Start := MinPage - 1;
        Stop := MaxPage - 1;
      end
      else if PrintRange = prSelection then
      begin
        Start := PageControl1.ActivePage.PageIndex;
        Stop := Start;
        end
      else  { PrintRange = prPageNums }
      begin
        Start := FromPage - 1;
        Stop := ToPage - 1;
```

```
      end;
    end;
    { now, print the pages }
    with Printer do
    begin
      BeginDoc;
      for I := Start to Stop do
      begin
        PageControl1.Pages[I].PaintTo(Handle, 10, 10);
        if I <> Stop then
          NewPage;
      end;
      EndDoc;
    end;
  end;
end;
```

See Also

MaxPage (see page 11467)

MinPage (see page 11469)

Options (see page 11471)

61.1.19.3.8 TPrintDialog.PrintToFile Property

Indicates whether the Print To File check box is selected.

Pascal

```
property PrintToFile: Boolean;
```

C++

```
__property Boolean PrintToFile;
```

Description

PrintToFile is true whenever the Print To File check box is selected in the dialog. To make the dialog open with the check box selected, set PrintToFile to true in the Object Inspector or in program code.

To make the Print To File check box appear in the dialog, set the poPrintToFile flag in Options (see page 11471). To disable (gray) the Print To File check box, set the poDisablePrintToFile flag.

C++ Examples:

```
/*
This example displays a print dialog box with a Print to
File check box, and then prints the contents of a rich edit
control to the indicated destination:
Verify that the file is saved by the SaveDialog.
*/
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
  PrintDialog1->Options << poPrintToFile;
  PrintDialog1->PrintToFile = true;
  if (PrintDialog1->Execute())
  {
    if (PrintDialog1->PrintToFile)
    {
      SaveDialog1->Title = "Print to File:";
      if (SaveDialog1->Execute())
        RichEdit1->Lines->SaveToFile(SaveDialog1->FileName);
      else
        RichEdit1->Print("");
    }
```

```
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
// you may need to change this path to suit your environment
  char const *Path = "..\\overview.rtf";
  RichEdit1->PlainText = false;
  RichEdit1->Lines->LoadFromFile(Path);
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example displays a print dialog box with a Print to
File check box, and then prints the contents of a rich edit
control to the indicated destination:
Verify that the file is saved by the SaveDialog.
}
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
with PrintDialog1 do
  begin
  Options := [poPrintToFile];
  PrintToFile := True;
  if Execute then
    begin
    if PrintToFile then
      begin
      SaveDialog1.Title := 'Print to File: ';
      if SaveDialog1.Execute then
        RichEdit1.Lines.SaveToFile(SaveDialog1.FileName);
      end
    else
      RichEdit1.Print('');
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
const
  // you may need to change this path to suit your environment
  Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := False;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
end;
```

See Also

Options (see page 11471)

61.1.19.3.9 TPrintDialog.ToPage Property

Indicates the page on which the print job is to end.

Pascal

```
property ToPage: Integer;
```

C++

```
__property int ToPage;
```

Description

The ToPage property corresponds to the To field in the Print dialog.

C++ Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PrintDialog1->Options.Clear();
  PrintDialog1->Options << poPageNums << poSelection;
  PrintDialog1->FromPage = 1;
  PrintDialog1->MinPage = 1;
  PrintDialog1->ToPage = PageControl1->PageCount;
  PrintDialog1->MaxPage = PageControl1->PageCount;
  if (PrintDialog1->Execute())
  {
    int Start, Stop;
    // determine the range the user wants to print
    switch (PrintDialog1->PrintRange)
    {
      case prSelection:
        Start = PageControl1->ActivePage->PageIndex;
        Stop = Start;
        break;
      case prPageNums:
        Start = PrintDialog1->FromPage - 1;
        Stop =  PrintDialog1->ToPage - 1;
        break;
      default:  // prAllPages
        Start = PrintDialog1->MinPage - 1;
        Stop = PrintDialog1->MaxPage - 1;
        break;
    }
    // now, print the pages
    Printer()->BeginDoc();
    for (int i = Start; i <= Stop; i++)
    {
      PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
      if (i != Stop)
        Printer()->NewPage();
    }
    Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
}
```

```
procedure TForm1.Button1Click(Sender:TObject);
var
  I, Start, Stop: Integer;
begin
PrintDialog1.Options := [poPageNums, poSelection];
PrintDialog1.FromPage := 1;
PrintDialog1.MinPage := 1;
PrintDialog1.ToPage := PageControl1.PageCount;
PrintDialog1.MaxPage := PageControl1.PageCount;

if PrintDialog1.Execute() then
  begin
    { determine the range the user wants to print }
    with PrintDialog1 do
    begin
      if PrintRange = prAllPages then
        begin
        Start := MinPage - 1;
        Stop := MaxPage - 1;
      end
      else if PrintRange = prSelection then
      begin
        Start := PageControl1.ActivePage.PageIndex;
        Stop := Start;
        end
      else  { PrintRange = prPageNums }
      begin
        Start := FromPage - 1;
        Stop := ToPage - 1;
      end;
    end;
    { now, print the pages }
    with Printer do
    begin
      BeginDoc;
      for I := Start to Stop do
      begin
        PageControl1.Pages[I].PaintTo(Handle, 10, 10);
        if I <> Stop then
          NewPage;
      end;
      EndDoc;
    end;
  end;
end;
```

See Also

FromPage (see page 11465)

MaxPage (see page 11467)

MinPage (see page 11469)

61.1.20 TPrinterSetupDialog Class

TPrinterSetupDialog displays a print-setup dialog for configuring printers.

Class Hierarchy

Pascal

```
TPrinterSetupDialog = class(TCommonDialog);
```

C++

```
class TPrinterSetupDialog : public TCommonDialog;
```

File

Dialogs

Description

TPrinterSetupDialog displays a modal Windows dialog box for configuring printers. The contents of the dialog vary depending on the printer driver selected.

The dialog does not appear at runtime until it is activated by a call to the Execute (see page 11482) method.

See Also

TPrintDialog (see page 11458)

61.1.20.1 TPrinterSetupDialog Members

The following tables list the members exposed by TPrinterSetupDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoClose (see page 11282)	Generates an OnClose (see page 11287) event.
	DoShow (see page 11283)	Generates an OnShow (see page 11288) event.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	MessageHook (see page 11284)	Provides special Windows message processing for the dialog.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	TaskModalDialog (see page 11284)	Ensures that Windows messages are properly routed when the dialog executes.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WndProc (see page 11285)	Responds to Windows messages sent to the dialog.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Template (see page 11287)	Points to a custom template for the dialog.
	TemplateModule (see page 11287)	This is TemplateModule, a member of class TCommonDialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 11281)	Creates and initializes a TCommonDialog (see page 11277) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.

	DefaultHandler (see page 11282)	Sends unhandled messages to Windows for processing.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 11281)	Destroys the TCommonDialog (see page 11277) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 11482)	Displays the printer configuration dialog box.
	Execute (see page 11283)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.

R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Handle (see page 11286)	Specifies the window handle of the dialog.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnClose (see page 11287)	Occurs when the dialog closes.
	OnShow (see page 11288)	Occurs when the dialog opens.

Published Properties

	Name	Description
	Ctl3D (see page 11285)	Determines whether the dialog has a three-dimensional look.
	HelpContext (see page 11286)	Specifies the context number for online Help.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

61.1.20.2 TPrinterSetupDialog Methods

The methods of the TPrinterSetupDialog class are listed here.

Public Methods

	Name	Description
V	Execute (see page 11482)	Displays the printer configuration dialog box.

61.1.20.2.1 TPrinterSetupDialog.Execute Method

Displays the printer configuration dialog box.

Pascal

```
function Execute(ParentWnd: HWND): Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute(HWND ParentWnd);
```

Description

Execute opens the print-setup dialog. If the user clicks OK, Execute configures the printers as specified in the dialog and returns true. If the user clicks Cancel, Execute returns false.

61.1.21 TReplaceDialog Class

TReplaceDialog displays a search-and-replace dialog.

Class Hierarchy

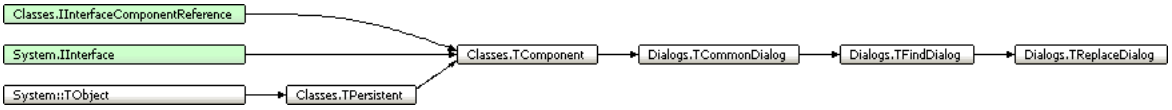

Pascal

```
TReplaceDialog = class(TFindDialog);
```

C++

```
class TReplaceDialog : public TFindDialog;
```

File

Dialogs

Description

TReplaceDialog is a special version of TFindDialog (see page 11376) that prompts the user for both a search string and a replace string. Like the Find (see page 11383) dialog, the Replace (see page 11384) dialog is modeless and does not appear at runtime until activated by a call to the Execute (see page 11381) method.

See Also

TFindDialog (see page 11376)

61.1.21.1 TReplaceDialog Members

The following tables list the members exposed by TReplaceDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoClose (see page 11282)	Generates an OnClose (see page 11287) event.
	DoShow (see page 11283)	Generates an OnShow (see page 11288) event.
	Find (see page 11383)	This is Find, a member of class TFindDialog.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.

	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	MessageHook (see page 11284)	Provides special Windows message processing for the dialog.
V	MessageHook (see page 11383)	Provides special Windows message processing for the dialog.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	Replace (see page 11384)	This is Replace, a member of class TFindDialog.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	TaskModalDialog (see page 11284)	Ensures that Windows messages are properly routed when the dialog executes.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WndProc (see page 11285)	Responds to Windows messages sent to the dialog.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Template (see page 11287)	Points to a custom template for the dialog.
	TemplateModule (see page 11287)	This is TemplateModule, a member of class TCommonDialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

	CloseDialog (see page 11381)	Closes the Find (see page 11383) dialog box.
V	Create (see page 11486)	Creates and initializes a TReplaceDialog (see page 11482) instance.
V	DefaultHandler (see page 11282)	Sends unhandled messages to Windows for processing.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 11281)	Destroys the TCommonDialog (see page 11277) instance and frees its memory.
V	Destroy (see page 11380)	Destroys the TFindDialog (see page 11376) and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Execute (see page 11381)	Displays the Find (see page 11383) dialog.
V	Execute (see page 11283)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Handle (see page 11286)	Specifies the window handle of the dialog.
	Left (see page 11386)	Determines the X coordinate (horizontal position) of the Find (see page 11383) dialog's upper left corner.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Position (see page 11390)	Determines the position of the Find (see page 11383) dialog on the screen.
	Top (see page 11392)	Determines the Y coordinate (vertical position) of the Find (see page 11383) dialog's upper left corner.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnClose (see page 11287)	Occurs when the dialog closes.
	OnFind (see page 11392)	Occurs when the user clicks the Find (see page 11383) Next button.
	OnReplace (see page 11488)	Occurs when the user clicks the Replace (see page 11384) or Replace (see page 11384) All button.
	OnShow (see page 11288)	Occurs when the dialog opens.

Published Properties

	Name	Description
	Ctl3D (see page 11285)	Determines whether the dialog has a three-dimensional look.
	FindText (see page 11384)	Contains the search string.
	HelpContext (see page 11286)	Specifies the context number for online Help.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 11386)	Determines the appearance and behavior of the Find (see page 11383) dialog.
	ReplaceText (see page 11487)	Contains the text that should replace FindText (see page 11384).
	Tag (see page 2808)	Stores an integer value as part of a component.

61.1.21.2 TReplaceDialog.Create Constructor

Creates and initializes a TReplaceDialog (see page 11482) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TReplaceDialog(TComponent * AOwner);
```

Description

The Create method generates a TReplaceDialog (see page 11482) instance, but the new dialog does not appear on the form at runtime until the Execute (see page 11381) method is called.

AOwner is the component responsible for freeing the TReplaceDialog (see page 11482) instance. It becomes the value of the Owner property.

See Also

Execute (see page 11381)

Owner (see page 2807)

61.1.21.3 TReplaceDialog Properties

The properties of the TReplaceDialog class are listed here.

Published Properties

	Name	Description
	ReplaceText (see page 11487)	Contains the text that should replace FindText (see page 11384).

61.1.21.3.1 TReplaceDialog.ReplaceText Property

Contains the text that should replace FindText (see page 11384).

Pascal

```
property ReplaceText;
```

C++

```
__property ReplaceText;
```

Description

FindText (see page 11384), taken from the Find (see page 11383) What edit box, contains the text string that the user wants to search for. ReplaceText, taken from the Replace (see page 11384) With edit box, contains the text string that will replace FindText (see page 11384). To make a default text string appear in the Replace (see page 11384) With edit box when the dialog opens, assign a value to ReplaceText in the Object Inspector or in program code.

C++ Examples:

```
/*
The following event handler searches a TMemo object called
Memo1 and replaces FindText with ReplaceText. It uses
TMemo's SelStart, SelLength, and SelText properties.
*/
void __fastcall TForm1::ReplaceDialog1Replace(TObject *Sender)
{
  TReplaceDialog *dlg = (TReplaceDialog *)Sender;
  /* perform a global case-sensitive search for FindText in Memo1 */
  int SelPos = Memo1->Lines->Text.Pos(dlg->FindText);
  if (SelPos > 0)
  {
    Memo1->SelStart = SelPos - 1;
    Memo1->SelLength = dlg->FindText.Length();
    // Replace selected text with ReplaceText
    Memo1->SelText = dlg->ReplaceText;
  }
  else
    MessageBeep(0);
}
```

Delphi Examples:

```
{
The following event handler searches a TMemo object called
Memo1 and replaces FindText with ReplaceText. It uses
TMemo's SelStart, SelLength, and SelText properties.
}
```

```
procedure TForm1.ReplaceDialog1Replace(Sender: TObject);
var
  SelPos: Integer;
begin
  with TReplaceDialog(Sender) do
  begin
  { Perform a global case-sensitive search for FindText in Memo1 }
    SelPos := Pos(FindText, Memo1.Lines.Text);
    if SelPos > 0 then
    begin
      Memo1.SelStart := SelPos - 1;
      Memo1.SelLength := Length(FindText);
      { Replace selected text with ReplaceText }
      Memo1.SelText := ReplaceText;
    end
    else MessageDlg(Concat('Could not find "', FindText, '" in Memo1.'), mtError, [mbOk], 0);
  end;
end;
```

See Also

FindText (see page 11384)

61.1.21.4 TReplaceDialog Events

The events of the TReplaceDialog class are listed here.

Published Events

	Name	Description
	OnReplace (see page 11488)	Occurs when the user clicks the Replace (see page 11384) or Replace (see page 11384) All button.

61.1.21.4.1 TReplaceDialog.OnReplace Event

Occurs when the user clicks the Replace (see page 11384) or Replace (see page 11384) All button.

Pascal

```
property OnReplace;
```

C++

```
__property OnReplace;
```

Description

OnReplace occurs whenever the user clicks Replace (see page 11384) or Replace (see page 11384) All. In addition, when the user clicks Replace (see page 11384), the frReplace flag is set (and the frReplaceAll flag is turned off) in Options (see page 11386). When the user clicks ReplaceAll, the frReplaceAll flag is set (and the frReplace flag is turned off) in Options (see page 11386).

Write an OnReplace event handler that searches for the text specified in FindText (see page 11384) and replaces it with the text in ReplaceText (see page 11487). Use the Options (see page 11386) flags to determine how the search is conducted.

C++ Examples:

```
/*
The following event handler searches a TMemo object called
Memo1 and replaces FindText with ReplaceText. It uses
TMemo's SelStart, SelLength, and SelText properties.
*/
void __fastcall TForm1::ReplaceDialog1Replace(TObject *Sender)
{
```

```
  TReplaceDialog *dlg = (TReplaceDialog *)Sender;
  /* perform a global case-sensitive search for FindText in Memo1 */
  int SelPos = Memo1->Lines->Text.Pos(dlg->FindText);
  if (SelPos > 0)
  {
    Memo1->SelStart = SelPos - 1;
    Memo1->SelLength = dlg->FindText.Length();
    // Replace selected text with ReplaceText
    Memo1->SelText = dlg->ReplaceText;
  }
  else
    MessageBeep(0);
}
```

Delphi Examples:

```
{
The following event handler searches a TMemo object called
Memo1 and replaces FindText with ReplaceText. It uses
TMemo's SelStart, SelLength, and SelText properties.
}
procedure TForm1.ReplaceDialog1Replace(Sender: TObject);
var
  SelPos: Integer;
begin
  with TReplaceDialog(Sender) do
  begin
  { Perform a global case-sensitive search for FindText in Memo1 }
    SelPos := Pos(FindText, Memo1.Lines.Text);
    if SelPos > 0 then
    begin
      Memo1.SelStart := SelPos - 1;
      Memo1.SelLength := Length(FindText);
      { Replace selected text with ReplaceText }
      Memo1.SelText := ReplaceText;
    end
    else MessageDlg(Concat('Could not find "', FindText, '" in Memo1.'), mtError, [mbOk], 0);
  end;
end;
```

See Also

OnFind (see page 11392)

Options (see page 11386)

ReplaceText (see page 11487)

FindText (see page 11384)

61.1.22 TSaveDialog Class

TSaveDialog displays a "Save As" dialog for saving files.

Class Hierarchy

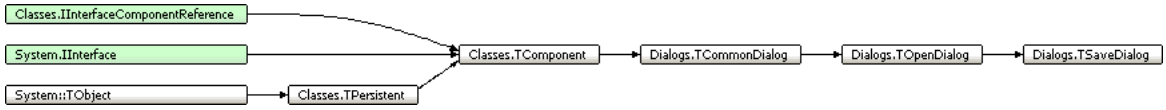

Pascal

```
TSaveDialog = class(TOpenDialog);
```

C++

```
class TSaveDialog : public TOpenDialog;
```

File

Dialogs

Description

TSaveDialog displays a modal Windows dialog box for selecting file names and saving files. The dialog does not appear at runtime until it is activated by a call to the Execute (see page 11494) method. When the user clicks Save, the dialog closes and the selected file name is stored in the FileName (see page 11424) property.

Delphi Examples:

```
{
This example reads an entire file into a buffer with one
command and then writes it into a saved file.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FromF, ToF: file;
  NumRead, NumWritten: Integer;
  Buf: array[1..2048] of Char;
begin
  if OpenDialog1.Execute then     { Display Open dialog box }
  begin
    AssignFile(FromF, OpenDialog1.FileName);
    Reset(FromF, 1);    { Record size = 1 }
    if SaveDialog1.Execute then      { Display Save dialog box}
    begin
      AssignFile(ToF, SaveDialog1.FileName);    { Open output file }
      Rewrite(ToF, 1);  { Record size = 1 }
      Canvas.TextOut(10, 10, 'Copying ' + IntToStr(FileSize(FromF))
        + ' bytes...');
      repeat
        System.BlockRead(FromF, Buf, SizeOf(Buf), NumRead);
        BlockWrite(ToF, Buf, NumRead, NumWritten);
      until (NumRead = 0) or (NumWritten <> NumRead);
      CloseFile(FromF);
      CloseFile(ToF);
    end;
  end;
end;
```

See Also

TSavePictureDialog (see page 13300)

TOpenDialog (see page 11407)

61.1.22.1 TSaveDialog Members

The following tables list the members exposed by TSaveDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.

	_Release (see page 2769)	Called when an application releases a component interface.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanClose (see page 11413)	This is CanClose, a member of class TOpenDialog.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 11413)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoCanClose (see page 11413)	This is DoCanClose, a member of class TOpenDialog.
	DoClose (see page 11282)	Generates an OnClose (see page 11287) event.
	DoExecute (see page 11413)	This is the overview for the DoExecute method overload.
	DoFolderChange (see page 11414)	This is DoFolderChange, a member of class TOpenDialog.
	DoIncludeItem (see page 11414)	This is DoIncludeItem, a member of class TOpenDialog.
	DoSelectionChange (see page 11415)	This is DoSelectionChange, a member of class TOpenDialog.
	DoShow (see page 11283)	Generates an OnShow (see page 11288) event.
	DoTypeChange (see page 11415)	This is DoTypeChange, a member of class TOpenDialog.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetFileNames (see page 11417)	This is GetFileNames, a member of class TOpenDialog.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetStaticRect (see page 11417)	Returns the coordinates of the reserved area of the dialog box.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	MessageHook (see page 11284)	Provides special Windows message processing for the dialog.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	TaskModalDialog (see page 11284)	Ensures that Windows messages are properly routed when the dialog executes.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.

	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WndProc (see page 11285)	Responds to Windows messages sent to the dialog.
	WndProc (see page 11420)	Responds to Windows messages sent to the dialog.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Template (see page 11287)	Points to a custom template for the dialog.
	TemplateModule (see page 11287)	This is TemplateModule, a member of class TCommonDialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 11281)	Creates and initializes a TCommonDialog (see page 11277) instance.
	Create (see page 11411)	Creates and initializes a TOpenDialog (see page 11407) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 11282)	Sends unhandled messages to Windows for processing.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 11281)	Destroys the TCommonDialog (see page 11277) instance and frees its memory.
	Destroy (see page 11412)	Destroys the TOpenDialog (see page 11407) and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 11494)	Displays the Save As dialog box.
	Execute (see page 11283)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	FileEditStyle (see page 11423)	Determines the style of the file-selection dialog. (Obsolete.)
R	Files (see page 11429)	List of selected file names.
R	Handle (see page 11286)	Specifies the window handle of the dialog.
	HistoryList (see page 11436)	Maintains a list of previously selected files. (Obsolete.)
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCanClose (see page 11439)	Occurs when the user tries to close the dialog without canceling.
	OnClose (see page 11287)	Occurs when the dialog closes.
	OnFolderChange (see page 11440)	Occurs when a directory is opened or closed from the dialog.
	OnIncludeItem (see page 11440)	Occurs before the dialog adds a file to the file list box.
	OnSelectionChange (see page 11441)	Occurs when file names displayed in the dialog are changed.
	OnShow (see page 11288)	Occurs when the dialog opens.
	OnTypeChange (see page 11441)	Occurs when the file types displayed in the dialog are changed.

Published Properties

	Name	Description
	Ctl3D (see page 11285)	Determines whether the dialog has a three-dimensional look.
	DefaultExt (see page 11421)	Specifies a default file extension.
	FileName (see page 11424)	Indicates the name and directory path of the last file selected.
	Filter (see page 11431)	Determines the file masks (filters) available in the dialog.
	FilterIndex (see page 11434)	Determines which filter is selected by default when the dialog opens.
	HelpContext (see page 11286)	Specifies the context number for online Help.
	InitialDir (see page 11436)	Determines the current directory when the dialog opens.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 11436)	Determines the appearance and behavior of the file-selection dialog.
	OptionsEx (see page 11438)	Augments the Options (see page 11436) property with additional flags that determine the appearance and behavior of the file-selection dialog.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Title (see page 11438)	Specifies the text in the dialog's title bar.

61.1.22.2 TSaveDialog Methods

The methods of the TSaveDialog class are listed here.

Public Methods

	Name	Description
	Execute (see page 11494)	Displays the Save As dialog box.

61.1.22.2.1 TSaveDialog.Execute Method

Displays the Save As dialog box.

Pascal

```
function Execute(ParentWnd: HWND): Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute(HWND ParentWnd);
```

Description

Execute opens the Save As dialog, returning true when the user selects a file name and clicks Save. If the user cancels the save operation, Execute returns a false value.

```
if SaveDialog1.Execute then
  Memo1.Lines.SaveToFile(SaveDialog1.FileName);
if (SaveDialog1->Execute())
  Memo1->Lines->SaveToFile(SaveDialog1->FileName);
```

61.1.23 TTaskDialog Class

Class for Task Dialog properties.

Class Hierarchy

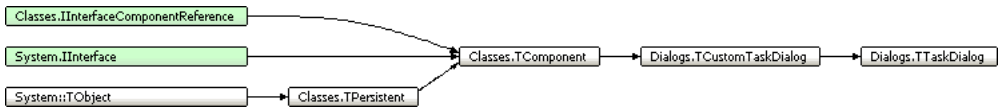

Pascal

```
TTaskDialog = class(TCustomTaskDialog);
```

C++

```
class TTaskDialog : public TCustomTaskDialog;
```

File

Dialogs

Description

TTaskDialog is a class for Task Dialogs (see page 11265).

Note: Task Dialogs (see page 11265) require Vista.

61.1.23.1 TTaskDialog Members

The following tables list the members exposed by TTaskDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CallbackProc (see page 11323)	An application-defined function used with Task Dialogs (see page 11265).
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 11322)	Creates a custom Task Dialog instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 11322)	Destroys an instance of a custom Task Dialog.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 11326)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.

	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Button (see page 11328)	Selected button.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Expanded (see page 11330)	Indicates whether the Task Dialog is expanded.
R	Handle (see page 11331)	Handle to the parent window.
	ModalResult (see page 11332)	Modal result.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	RadioButton (see page 11333)	Selected radio button.
R	URL (see page 11334)	URL for the Task Dialog.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnButtonClicked (see page 11504)	Button (see page 11328) click event.
	OnDialogConstructed (see page 11504)	Dialog constructed event.
	OnDialogCreated (see page 11505)	Task Dialog creation event.

	OnDialogDestroyed (see page 11505)	Task dialog destruction event..
	OnExpanded (see page 11505)	Task dialog expanded event.
	OnHyperlinkClicked (see page 11505)	Task dialog hyperlink clicked event.
	OnNavigated (see page 11506)	Task Dialog navigation event.
	OnRadioButtonClicked (see page 11506)	Task Dialog radio button click event.
	OnTimer (see page 11506)	Task dialog timer event.
	OnVerificationClicked (see page 11506)	Task Dialog verification check box event.

Published Properties

	Name	Description
	Buttons (see page 11499)	The collection of all buttons on the Task Dialog.
	Caption (see page 11499)	Caption for the Task Dialog.
	CommonButtons (see page 11499)	Specifies the push buttons displayed in the Task Dialog.
	CustomFooterIcon (see page 11500)	Indicates the custom footer icon, if any, for the Task Dialog.
	CustomMainIcon (see page 11500)	Indicates the custom main icon, if any, for the Task Dialog.
	DefaultButton (see page 11500)	The default button for this Task Dialog.
	ExpandButtonCaption (see page 11500)	Additional information for this button.
	ExpandedText (see page 11501)	Text (see page 11503) to display when the Task Dialog is expanded.
	Flags (see page 11501)	Flags for this Task Dialog.
	FooterIcon (see page 11502)	Indicates which standard footer icon to display.
	FooterText (see page 11502)	String to be used in the footer area of the Task Dialog.
	HelpContext (see page 11502)	Help context for the Task Dialog.
	MainIcon (see page 11502)	Indicates which standard main icon to display.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ProgressBar (see page 11503)	Progress bar for this Task Dialog.
	RadioButtons (see page 11503)	Collection of all radio buttons on this Task Dialog.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Text (see page 11503)	The main text on this Task Dialog..
	Title (see page 11503)	The title of this Task Dialog.
	VerificationText (see page 11504)	Text (see page 11503) used to label the verification check box.

Strict protected Methods

	Name	Description
	DoExecute (see page 11323)	Executes the Task Dialog.
	DoOnButtonClicked (see page 11324)	Performs the click function of the Task Dialog.
	DoOnDialogContructed (see page 11324)	Performs the on-construction actions of the Task Dialog.
	DoOnDialogCreated (see page 11324)	Performs the on-creation actrions of a Task Dialog.
	DoOnDialogDestroyed (see page 11324)	Performs the on-destruction actions of a Task Dialog.
	DoOnExpandButtonClicked (see page 11325)	Performs the expansion action of a Task Dialog when the **Expand** button is pushed.
	DoOnHelp (see page 11325)	Performs the help function for a Task Dialog.
	DoOnHyperlinkClicked (see page 11325)	Performs the hyperlink click action of a Task Dialog.
	DoOnNavigated (see page 11325)	Performs the navigation action of a Task Dialog.
	DoOnRadioButtonClicked (see page 11326)	Performs the radio button click action of a Task Dialog.
	DoOnTimer (see page 11326)	Performs the timer action of a Task Dialog.
	DoOnVerificationClicked (see page 11326)	Performs the verification check action of a Task Dialog.

61.1.23.2 TTaskDialog Properties

The properties of the TTaskDialog class are listed here.

Published Properties

	Name	Description
	Buttons (see page 11499)	The collection of all buttons on the Task Dialog.

	Caption (see page 11499)	Caption for the Task Dialog.
	CommonButtons (see page 11499)	Specifies the push buttons displayed in the Task Dialog.
	CustomFooterIcon (see page 11500)	Indicates the custom footer icon, if any, for the Task Dialog.
	CustomMainIcon (see page 11500)	Indicates the custom main icon, if any, for the Task Dialog.
	DefaultButton (see page 11500)	The default button for this Task Dialog.
	ExpandButtonCaption (see page 11500)	Additional information for this button.
	ExpandedText (see page 11501)	Text (see page 11503) to display when the Task Dialog is expanded.
	Flags (see page 11501)	Flags for this Task Dialog.
	FooterIcon (see page 11502)	Indicates which standard footer icon to display.
	FooterText (see page 11502)	String to be used in the footer area of the Task Dialog.
	HelpContext (see page 11502)	Help context for the Task Dialog.
	MainIcon (see page 11502)	Indicates which standard main icon to display.
	ProgressBar (see page 11503)	Progress bar for this Task Dialog.
	RadioButtons (see page 11503)	Collection of all radio buttons on this Task Dialog.
	Text (see page 11503)	The main text on this Task Dialog..
	Title (see page 11503)	The title of this Task Dialog.
	VerificationText (see page 11504)	Text (see page 11503) used to label the verification check box.

61.1.23.2.1 TTaskDialog.Buttons Property

The collection of all buttons on the Task Dialog.

Pascal

```
property Buttons: TTaskDialogButtons;
```

C++

```
__property TTaskDialogButtons Buttons;
```

Description

Buttons contains all of the buttons for the Task Dialog.

61.1.23.2.2 TTaskDialog.Caption Property

Caption for the Task Dialog.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption contains the caption for the Task Dialog.

61.1.23.2.3 TTaskDialog.CommonButtons Property

Specifies the push buttons displayed in the Task Dialog.

Pascal

```
property CommonButtons: TTaskDialogCommonButtons;
```

C++

```
__property TTaskDialogCommonButtons CommonButtons;
```

Description

CommonButtons specifies the common push buttons for the Task Dialog. If no buttons are specified, the **OK** button is displayed.

61.1.23.2.4 TTaskDialog.CustomFooterIcon Property

Indicates the custom footer icon, if any, for the Task Dialog.

Pascal

```
property CustomFooterIcon: TIcon;
```

C++

```
__property TIcon CustomFooterIcon;
```

Description

CustomFooterIcon points to the custom footer icon for this Task Dialog. If **tfUseHiconFooter** is set in Flags (see page 11501).

61.1.23.2.5 TTaskDialog.CustomMainIcon Property

Indicates the custom main icon, if any, for the Task Dialog.

Pascal

```
property CustomMainIcon: TIcon;
```

C++

```
__property TIcon CustomMainIcon;
```

Description

CustomMainIcon points to the custom main icon for this Task Dialog. If **tfUseHiconMain** is set in Flags (see page 11501).

61.1.23.2.6 TTaskDialog.DefaultButton Property

The default button for this Task Dialog.

Pascal

```
property DefaultButton: TTaskDialogCommonButton;
```

C++

```
__property TTaskDialogCommonButton DefaultButton;
```

Description

DefaultButton indicates the default button for this Task Dialog.

61.1.23.2.7 TTaskDialog.ExpandButtonCaption Property

Additional information for this button.

Pascal

```
property ExpandButtonCaption: string;
```

C++

```
__property AnsiString ExpandButtonCaption;
```

Description

ExpandButtonCaption contains additional text to be displayed when the caption is expanded.

61.1.23.2.8 TTaskDialog.ExpandedText Property

Text (see page 11503) to display when the Task Dialog is expanded.

Pascal

```
property ExpandedText: string;
```

C++

```
__property AnsiString ExpandedText;
```

Description

ExpandedText contains additional information to display when the Task Dialog is expanded.

61.1.23.2.9 TTaskDialog.Flags Property

Flags for this Task Dialog.

Pascal

```
property Flags: TTaskDialogFlags;
```

C++

```
__property TTaskDialogFlags Flags;
```

Description

Flags provides flags for TTaskDialog (see page 11494).

Flag	Description
tfEnableHyperlinks	If set, content, footer, and expanded text can include hyperlinks.
tfUseHiconMain	if set, use the custom main icon.
tfUseHiconFooter	If set, use the custom footer icon.
tfAllowDialogCancellation	If set, permits Task Dialog to be closed in the absence of a **Cancel** button.
tfUseCommandLinks	If set, buttons are displayed as command links using a standard dialog glyph.
tfUseCommandLinksNoIcon	If set, buttons are displayed as command links without a glyph.
tfExpandFooterArea	If set, display expanded text in the footer.
tfExpandedByDefault	If set, expanded text is displayed when the Task Dialog opens.
tfVerificationFlagChecked	If set, the verification check box is initially checked.
tfShowProgressBar	If set, display the progress bar.
tfShowMarqueeProgressBar	If set, display marquee progress bar.
tfCallbackTimer	If set, dialogs callback will be called every 200 milliseconds.
tfPositionRelativeToWindow	If set, Task Dialog is centered with respect to parent window.
tfRtlLayout	If set, text reads right-to-left.
tfNoDefaultRadioButton	If set, there is no default radio button.

tfCanBeMinimized	If set, the Task Dialog can be minimized.

61.1.23.2.10 TTaskDialog.FooterIcon Property

Indicates which standard footer icon to display.

Pascal

```
property FooterIcon: TTaskDialogIcon;
```

C++

```
__property TTaskDialogIcon FooterIcon;
```

Description

FooterIcon indicates which standard footer icon to display in the Task Dialog. Legal values include **tdiNone (see page 11591)**, **tdiWarning (see page 11591)**, **tdiError (see page 11590)**, **tdiInformation (see page 11590)**, and **tdiShield (see page 11591)**.

61.1.23.2.11 TTaskDialog.FooterText Property

String to be used in the footer area of the Task Dialog.

Pascal

```
property FooterText: string;
```

C++

```
__property AnsiString FooterText;
```

Description

FooterText provides a string to be used in the footer area of the Task Dialog.

61.1.23.2.12 TTaskDialog.HelpContext Property

Help context for the Task Dialog.

Pascal

```
property HelpContext: Integer;
```

C++

```
__property int HelpContext;
```

Description

HelpContext provides the help context for the Task Dialog

61.1.23.2.13 TTaskDialog.MainIcon Property

Indicates which standard main icon to display.

Pascal

```
property MainIcon: TTaskDialogIcon;
```

C++

```
__property TTaskDialogIcon MainIcon;
```

Description

MainIcon indicates which standard main icon to display in the Task Dialog. Legal values include **tdiNone (see page 11591)**, **tdiWarning (see page 11591)**, **tdiError (see page 11590)**, **tdiInformation (see page 11590)**, and **tdiShield (see page 11591)**.

61.1.23.2.14 TTaskDialog.ProgressBar Property

Progress bar for this Task Dialog.

Pascal

```
property ProgressBar: TTaskDialogProgressBar;
```

C++

```
__property TTaskDialogProgressBar ProgressBar;
```

Description

ProgressBar is the TTaskDialogProgressBar (see page 11523) for this Task Dialog.

61.1.23.2.15 TTaskDialog.RadioButtons Property

Collection of all radio buttons on this Task Dialog.

Pascal

```
property RadioButtons: TTaskDialogButtons;
```

C++

```
__property TTaskDialogButtons RadioButtons;
```

Description

RadioButtons is the collection of radio buttons for this class dialog.

61.1.23.2.16 TTaskDialog.Text Property

The main text on this Task Dialog..

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

Text contains the main text to be displayed in this Task Dialog.

61.1.23.2.17 TTaskDialog.Title Property

The title of this Task Dialog.

Pascal

```
property Title: string;
```

C++

```
__property AnsiString Title;
```

Description

Title contains the title of this Task Dialog.

61.1.23.2.18 TTaskDialog.VerificationText Property

Text (see page 11503) used to label the verification check box.

Pascal

```
property VerificationText: string;
```

C++

```
__property AnsiString VerificationText;
```

Description

VerificationText contains text use to label the verification check box.

61.1.23.3 TTaskDialog Events

The events of the TTaskDialog class are listed here.

Published Events

	Name	Description
	OnButtonClicked (see page 11504)	Button (see page 11328) click event.
	OnDialogConstructed (see page 11504)	Dialog constructed event.
	OnDialogCreated (see page 11505)	Task Dialog creation event.
	OnDialogDestroyed (see page 11505)	Task dialog destruction event..
	OnExpanded (see page 11505)	Task dialog expanded event.
	OnHyperlinkClicked (see page 11505)	Task dialog hyperlink clicked event.
	OnNavigated (see page 11506)	Task Dialog navigation event.
	OnRadioButtonClicked (see page 11506)	Task Dialog radio button click event.
	OnTimer (see page 11506)	Task dialog timer event.
	OnVerificationClicked (see page 11506)	Task Dialog verification check box event.

61.1.23.3.1 TTaskDialog.OnButtonClicked Event

Button (see page 11328) click event.

Pascal

```
property OnButtonClicked: TTaskDlgClickEvent;
```

C++

```
__property TTaskDlgClickEvent OnButtonClicked;
```

Description

OnButtonClicked is the button click event for Task Dialog.

61.1.23.3.2 TTaskDialog.OnDialogConstructed Event

Dialog constructed event.

Pascal

```
property OnDialogConstructed: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDialogConstructed;
```

Description

OnDialogConstructed is the dialog constructed event.

61.1.23.3.3 TTaskDialog.OnDialogCreated Event

Task Dialog creation event.

Pascal

```
property OnDialogCreated: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDialogCreated;
```

Description

OnDialogCreated is the Task Dialog creation event.

61.1.23.3.4 TTaskDialog.OnDialogDestroyed Event

Task dialog destruction event..

Pascal

```
property OnDialogDestroyed: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDialogDestroyed;
```

Description

OnDialogDestroyed is the Task Dialog destroy event.

61.1.23.3.5 TTaskDialog.OnExpanded Event

Task dialog expanded event.

Pascal

```
property OnExpanded: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExpanded;
```

Description

OnExpanded is the Task Dialog expanded event.

61.1.23.3.6 TTaskDialog.OnHyperlinkClicked Event

Task dialog hyperlink clicked event.

Pascal

```
property OnHyperlinkClicked: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnHyperlinkClicked;
```

Description

OnHyperlinkClicked is the Task Dialog hyperlink clicked event.

61.1.23.3.7 TTaskDialog.OnNavigated Event

Task Dialog navigation event.

Pascal

```
property OnNavigated: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnNavigated;
```

Description

OnNavigated is the Task Dialog navigation event.

61.1.23.3.8 TTaskDialog.OnRadioButtonClicked Event

Task Dialog radio button click event.

Pascal

```
property OnRadioButtonClicked: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnRadioButtonClicked;
```

Description

OnRadioButtonClicked

61.1.23.3.9 TTaskDialog.OnTimer Event

Task dialog timer event.

Pascal

```
property OnTimer: TTaskDlgTimerEvent;
```

C++

```
__property TTaskDlgTimerEvent OnTimer;
```

Description

OnTimer is the Task Dialog timer event.

61.1.23.3.10 TTaskDialog.OnVerificationClicked Event

Task Dialog verification check box event.

Pascal

```
property OnVerificationClicked: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnVerificationClicked;
```

Description

OnVerificationClicked is the Task Dialog verification check box event.

61.1.24 TTaskDialogBaseButtonItem Class

Base button class for Task Dialogs (see page 11265).

Class Hierarchy

Pascal

```
TTaskDialogBaseButtonItem = class(TCollectionItem);
```

C++

```
class TTaskDialogBaseButtonItem : public TCollectionItem;
```

File

Dialogs

Description

TTaskDialogBaseButtonItem is the base button class for Task Dialog buttons. This class implements the basic button functions for all Task dialog button classes

61.1.24.1 TTaskDialogBaseButtonItem Members

The following tables list the members exposed by TTaskDialogBaseButtonItem.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Click (see page 11509)	Clicks the button.
V	Create (see page 11509)	Creates a button instance.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 11509)	Destroys a button.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetInitialState (see page 11511)	Set the initial state of the button.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
	ModalResult (see page 11512)	Modal result of pushing this button.
R	TextWStr (see page 11513)	Descriptive button text.

Published Properties

	Name	Description
	Caption (see page 11511)	The caption for the button.
	Default (see page 11512)	Indicates whether this button is the default.
	Enabled (see page 11512)	Indicates whether this button is enabled.

Strict protected Methods

	Name	Description
V	DoButtonClick (see page 11510)	Click (see page 11509) the button.
V	DoSetEnabled (see page 11510)	Enables or disables the button..
V	GetButtonText (see page 11510)	Fetch the button text.
V	GetDisplayName (see page 11510)	Get display name..

Strict protected Properties

	Name	Description
R	Client (see page 11512)	Client area of the button.

61.1.24.2 TTaskDialogBaseButtonItem.Create Constructor

Creates a button instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TTaskDialogBaseButtonItem(TCollection Collection);
```

Description

Use Create to create an instance of TTaskDialogBaseButtonItem (see page 11507).

61.1.24.3 TTaskDialogBaseButtonItem.Destroy Destructor

Destroys a button.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTaskDialogBaseButtonItem();
```

Description

Use Destroy to destroy an instance of TTaskDialogBaseButtonItem (see page 11507).

61.1.24.4 TTaskDialogBaseButtonItem Methods

The methods of the TTaskDialogBaseButtonItem class are listed here.

Public Methods

	Name	Description
	Click (see page 11509)	Clicks the button.
V	SetInitialState (see page 11511)	Set the initial state of the button.

Strict protected Methods

	Name	Description
V	DoButtonClick (see page 11510)	Click (see page 11509) the button.
V	DoSetEnabled (see page 11510)	Enables or disables the button..
V	GetButtonText (see page 11510)	Fetch the button text.
V	GetDisplayName (see page 11510)	Get display name..

61.1.24.4.1 TTaskDialogBaseButtonItem.Click Method

Clicks the button.

Pascal

```
procedure Click;
```

C++

```
__fastcall Click();
```

Description

Use Click to perform the action of clicking the button.

61.1.24.4.2 TTaskDialogBaseButtonItem.DoButtonClick Method

Click (see page 11509) the button.

Pascal

```
procedure DoButtonClick; virtual;
```

C++

```
virtual __fastcall DoButtonClick();
```

Description

DoButtonClick performs the basic operation of clicking a button. In general, use the public method Click (see page 11509) to perform this action.

61.1.24.4.3 TTaskDialogBaseButtonItem.DoSetEnabled Method

Enables or disables the button..

Pascal

```
procedure DoSetEnabled; virtual;
```

C++

```
virtual __fastcall DoSetEnabled();
```

Description

Use DoSetEnabled to enable or disable the button.

61.1.24.4.4 TTaskDialogBaseButtonItem.GetButtonText Method

Fetch the button text.

Pascal

```
function GetButtonText: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetButtonText();
```

Description

Use GetButtonText to get the button text.

61.1.24.4.5 TTaskDialogBaseButtonItem.GetDisplayName Method

Get display name..

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

Use GetDisplayName to fetch the display name for the button.

61.1.24.4.6 TTaskDialogBaseButtonItem.SetInitialState Method

Set the initial state of the button.

Pascal

```
procedure SetInitialState; virtual;
```

C++

```
virtual __fastcall SetInitialState();
```

Description

Use SetInitialState to set the initial state of the button.

61.1.24.5 TTaskDialogBaseButtonItem Properties

The properties of the TTaskDialogBaseButtonItem class are listed here.

Public Properties

	Name	Description
	ModalResult (see page 11512)	Modal result of pushing this button.
R	TextWStr (see page 11513)	Descriptive button text.

Published Properties

	Name	Description
	Caption (see page 11511)	The caption for the button.
	Default (see page 11512)	Indicates whether this button is the default.
	Enabled (see page 11512)	Indicates whether this button is enabled.

Strict protected Properties

	Name	Description
R	Client (see page 11512)	Client area of the button.

61.1.24.5.1 TTaskDialogBaseButtonItem.Caption Property

The caption for the button.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

This property contains the caption for the button.

61.1.24.5.2 TTaskDialogBaseButtonItem.Client Property

Client area of the button.

Pascal

```
property Client: TCustomTaskDialog;
```

C++

```
__property TCustomTaskDialog Client;
```

Description

Client describes the client area of the button

61.1.24.5.3 TTaskDialogBaseButtonItem.Default Property

Indicates whether this button is the default.

Pascal

```
property Default: Boolean;
```

C++

```
__property Boolean Default;
```

Description

Default is **True** if this is the default button in the Task Dialog window.

61.1.24.5.4 TTaskDialogBaseButtonItem.Enabled Property

Indicates whether this button is enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled is **True** if this button is enabled.

61.1.24.5.5 TTaskDialogBaseButtonItem.ModalResult Property

Modal result of pushing this button.

Pascal

```
property ModalResult: TModalResult;
```

C++

```
__property TModalResult ModalResult;
```

Description

ModalResult defines the modal result of pushing this button: mrYes, mrNo, mrOk, mrCancel mrAbort, mrRetry, mrIgnore, mrAll, mrNoToAll, mrYesToAll.

61.1.24.5.6 TTaskDialogBaseButtonItem.TextWStr Property

Descriptive button text.

Pascal

```
property TextWStr: LPCWSTR;
```

C++

```
__property LPCWSTR TextWStr;
```

Description

TextWStr contains the descriptive text for this button.

61.1.25 TTaskDialogButtonItem Class

A single Task Dialog button item.

Class Hierarchy

Pascal

```
TTaskDialogButtonItem = class(TTaskDialogBaseButtonItem);
```

C++

```
class TTaskDialogButtonItem : public TTaskDialogBaseButtonItem;
```

File

Dialogs

Description

An instance of the TTaskDialogButtonItem class contains information about a single Task Dialog button..

61.1.25.1 TTaskDialogButtonItem Members

The following tables list the members exposed by TTaskDialogButtonItem.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Click (see page 11509)	Clicks the button.
V	Create (see page 11515)	Creates a button instance.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 11509)	Destroys a button.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetInitialState (see page 11516)	Set the initial state of the button.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
R	TextWStr (see page 11513)	Descriptive button text.

Published Properties

	Name	Description
	Caption (see page 11511)	The caption for the button.
	CommandLinkHint (see page 11516)	Command link hint.

	Default (see page 11512)	Indicates whether this button is the default.
	ElevationRequired (see page 11516)	Elevation required.
	Enabled (see page 11512)	Indicates whether this button is enabled.
	ModalResult (see page 11517)	Modal result of pushing this button.

Strict protected Methods

	Name	Description
V	DoButtonClick (see page 11510)	Click (see page 11509) the button.
V	DoSetEnabled (see page 11510)	Enables or disables the button..
V	GetButtonText (see page 11515)	Fetch the button text.
V	GetDisplayName (see page 11510)	Get display name..

Strict protected Properties

	Name	Description
R	Client (see page 11512)	Client area of the button.

61.1.25.2 TTaskDialogButtonItem.Create Constructor

Creates a button instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TTaskDialogButtonItem(TCollection Collection);
```

Description

Use Create to create an instance of TTaskDialogButtonItem (see page 11513).

61.1.25.3 TTaskDialogButtonItem Methods

The methods of the TTaskDialogButtonItem class are listed here.

Public Methods

	Name	Description
V	SetInitialState (see page 11516)	Set the initial state of the button.

Strict protected Methods

	Name	Description
V	GetButtonText (see page 11515)	Fetch the button text.

61.1.25.3.1 TTaskDialogButtonItem.GetButtonText Method

Fetch the button text.

Pascal

```
function GetButtonText: string; override;
```

C++

```
virtual __fastcall AnsiString GetButtonText();
```

Description

Use GetButtonText to get the button text.

61.1.25.3.2 TTaskDialogButtonItem.SetInitialState Method

Set the initial state of the button.

Pascal

```
procedure SetInitialState; override;
```

C++

```
virtual __fastcall SetInitialState();
```

Description

Use SetInitialState to set the initial state of the button.

61.1.25.4 TTaskDialogButtonItem Properties

The properties of the TTaskDialogButtonItem class are listed here.

Published Properties

	Name	Description
	CommandLinkHint (see page 11516)	Command link hint.
	ElevationRequired (see page 11516)	Elevation required.
	ModalResult (see page 11517)	Modal result of pushing this button.

61.1.25.4.1 TTaskDialogButtonItem.CommandLinkHint Property

Command link hint.

Pascal

```
property CommandLinkHint: string;
```

C++

```
__property AnsiString CommandLinkHint;
```

Description

A hint for this button.

61.1.25.4.2 TTaskDialogButtonItem.ElevationRequired Property

Elevation required.

Pascal

```
property ElevationRequired: Boolean;
```

C++

```
__property Boolean ElevationRequired;
```

Description

ElevationRequired is **True** if security elevation is required in the process following a button click. Setting this property indicates that this button should display a User Account Control (UAC) shield icon.

61.1.25.4.3 TTaskDialogButtonItem.ModalResult Property

Modal result of pushing this button.

Pascal

```
property ModalResult: TModalResult;
```

C++

```
__property TModalResult ModalResult;
```

Description

ModalResult defines the modal result of pushing this button: mrYes, mrNo, mrOk, mrCancel mrAbort, mrRetry, mrIgnore, mrAll, mrNoToAll, mrYesToAll.

61.1.26 TTaskDialogButtons Class

A collection of buttons appearing on a Task Dialog.

Class Hierarchy

Pascal

```
TTaskDialogButtons = class(TOwnedCollection);
```

C++

```
class TTaskDialogButtons : public TOwnedCollection;
```

File

Dialogs

Description

TTaskDialogButtons represents the collection of buttons appearing on a Task Dialog.

61.1.26.1 TTaskDialogButtons Members

The following tables list the members exposed by TTaskDialogButtons.

Protected Methods

	Name	Description
	Added (see page 2736)	Responds when items are added to the collection.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.

V	GetOwner (see page 2915)	Returns the owner of an object.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 11519)	Add a button to the collection.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	Buttons (see page 11519)	Returns an array of the buttons defined for this Task Dialog.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 2734)	Creates and initializes a collection.
	Create (see page 2914)	Creates and initializes a TOwnedCollection (see page 2912) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindButton (see page 11520)	Find a button by modal result.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 11520)	Get an enumerator for the Task Dialog button collection.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetInitialState (see page 11520)	Sets the initial state for a collection of Task Dialog buttons.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
	DefaultButton (see page 11521)	The default button for the Task Dialog.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 11521)	An array of buttons for the Task Dialog

61.1.26.2 TTaskDialogButtons Methods

The methods of the TTaskDialogButtons class are listed here.

Public Methods

	Name	Description
	Add (see page 11519)	Add a button to the collection.
	Buttons (see page 11519)	Returns an array of the buttons defined for this Task Dialog.
	FindButton (see page 11520)	Find a button by modal result.
	GetEnumerator (see page 11520)	Get an enumerator for the Task Dialog button collection.
	SetInitialState (see page 11520)	Sets the initial state for a collection of Task Dialog buttons.

61.1.26.2.1 TTaskDialogButtons.Add Method

Add a button to the collection.

Pascal

```
function Add: TTaskDialogBaseButtonItem;
```

C++

```
__fastcall TTaskDialogBaseButtonItem Add();
```

Description

Use Add to add a button to Items (see page 11521).

61.1.26.2.2 TTaskDialogButtons.Buttons Method

Returns an array of the buttons defined for this Task Dialog.

Pascal

```
function Buttons: PTaskDialogButton;
```

C++

```
__fastcall PTaskDialogButton Buttons();
```

Description

Use Buttons to retrieve an array of the modal results and text strings of the buttons defined for this Task Dialog.

61.1.26.2.3 TTaskDialogButtons.FindButton Method

Find a button by modal result.

Pascal

```
function FindButton(AModalResult: TModalResult): TTaskDialogBaseButtonItem;
```

C++

```
__fastcall TTaskDialogBaseButtonItem FindButton(TModalResult AModalResult);
```

Description

Use FindButton to find the button in this Task Dialog that corresponds to a particular modal result value.

61.1.26.2.4 TTaskDialogButtons.GetEnumerator Method

Get an enumerator for the Task Dialog button collection.

Pascal

```
function GetEnumerator: TTaskDialogButtonsEnumerator;
```

C++

```
__fastcall TTaskDialogButtonsEnumerator GetEnumerator();
```

Description

Use GetEnumerator to fetch an enumerator for TTaskDialogButtons (see page 11517).

61.1.26.2.5 TTaskDialogButtons.SetInitialState Method

Sets the initial state for a collection of Task Dialog buttons.

Pascal

```
procedure SetInitialState; dynamic;
```

C++

```
__fastcall SetInitialState();
```

Description

Use SetInitialState to set the initial state for TTaskDialogButtons (see page 11517)

61.1.26.3 TTaskDialogButtons Properties

The properties of the TTaskDialogButtons class are listed here.

Public Properties

	Name	Description
	DefaultButton (see page 11521)	The default button for the Task Dialog.

	Items (see page 11521)	An array of buttons for the Task Dialog

61.1.26.3.1 TTaskDialogButtons.DefaultButton Property

The default button for the Task Dialog.

Pascal

```
property DefaultButton: TTaskDialogBaseButtonItem;
```

C++

```
__property TTaskDialogBaseButtonItem DefaultButton;
```

Description

DefaultButton represents the default button for the Task Dialog.

61.1.26.3.2 TTaskDialogButtons.Items Property

An array of buttons for the Task Dialog

Pascal

```
property Items [Index: Integer]: TTaskDialogBaseButtonItem;
```

C++

```
__property TTaskDialogBaseButtonItem Items[int Index];
```

Description

Itemsis an array of all the TTaskDialogBaseButton used in this Task Dialog.

61.1.27 TTaskDialogButtonsEnumerator Class

Enumerates the button collection for a Task Dialog.

Class Hierarchy

Pascal

```
TTaskDialogButtonsEnumerator = class;
```

C++

```
class TTaskDialogButtonsEnumerator;
```

File

Dialogs

Description

TTaskDialogButtonsEnumerator provides an enumerator for the collection TTaskDialogButtons (see page 11517)

61.1.27.1 TTaskDialogButtonsEnumerator Members

The following tables list the members exposed by TTaskDialogButtonsEnumerator.

Public Methods

	Name	Description
	Create (see page 11522)	Creates a Task Dialog buttons enumerator.
	GetCurrent (see page 11522)	Gets the current button.
	MoveNext (see page 11522)	Moves the enumerator to the next button in the collection.

Public Properties

	Name	Description
R	Current (see page 11523)	The current button.

61.1.27.2 TTaskDialogButtonsEnumerator.Create Constructor

Creates a Task Dialog buttons enumerator.

Pascal

```
constructor Create(ACollection: TTaskDialogButtons);
```

C++

```
__fastcall TTaskDialogButtonsEnumerator(TTaskDialogButtons ACollection);
```

Description

Creates an instance of TTaskDialogButtonsEnumerator (see page 11521).

61.1.27.3 TTaskDialogButtonsEnumerator Methods

The methods of the TTaskDialogButtonsEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 11522)	Gets the current button.
	MoveNext (see page 11522)	Moves the enumerator to the next button in the collection.

61.1.27.3.1 TTaskDialogButtonsEnumerator.GetCurrent Method

Gets the current button.

Pascal

```
function GetCurrent: TTaskDialogBaseButtonItem;
```

C++

```
__fastcall TTaskDialogBaseButtonItem GetCurrent();
```

Description

Use GetCurrent to get the current button in TTaskDialogButtons (see page 11517).

61.1.27.3.2 TTaskDialogButtonsEnumerator.MoveNext Method

Moves the enumerator to the next button in the collection.

Pascal

```
function MoveNext: Boolean;
```

C++

```
__fastcall Boolean MoveNext();
```

Description

Use MoveNext to move the TTaskDialogButtonsEnumerator (see page 11521) to the next button in TTaskDialogButtons (see page 11517).

61.1.27.4 TTaskDialogButtonsEnumerator Properties

The properties of the TTaskDialogButtonsEnumerator class are listed here.

Public Properties

	Name	Description
R	Current (see page 11523)	The current button.

61.1.27.4.1 TTaskDialogButtonsEnumerator.Current Property

The current button.

Pascal

```
property Current: TTaskDialogBaseButtonItem;
```

C++

```
__property TTaskDialogBaseButtonItem Current;
```

Description

Current contains the TTaskDialogButtonsEnumerator (see page 11521) current button in the TTaskDialogButtons (see page 11517) collection.

61.1.28 TTaskDialogProgressBar Class

Progress bar class for Task Dialogs (see page 11265). in one, concise sentence.

Class Hierarchy

Pascal

```
TTaskDialogProgressBar = class(TPersistent);
```

C++

```
class TTaskDialogProgressBar : public TPersistent;
```

File

Dialogs

Description

TTaskDialogProgressBar provides properties and methods to manage progress bars for Task Dialogs (see page 11265).

61.1.28.1 TTaskDialogProgressBar Members

The following tables list the members exposed by TTaskDialogProgressBar.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11525)	Creates an instance of a class dialog progress bar.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	Initialize (see page 11525)	Initialize an instance of a task dialog progress bar.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Published Properties

	Name	Description
	MarqueeSpeed (see page 11526)	Controls (see page 5711) the marquee speed of an indeterminate progress bar.
	Max (see page 11526)	Specifies the upper limit of the value of the progress bar.
	Min (see page 11526)	Specifies the minimum value of the progress bar.

	Position (see page 11526)	Specifies the current position of the progress bar.
	State (see page 11527)	The state of the progress bar.

61.1.28.2 TTaskDialogProgressBar.Create Constructor

Creates an instance of a class dialog progress bar.

Pascal

```
constructor Create(AClient: TCustomTaskDialog);
```

C++

```
__fastcall TTaskDialogProgressBar(TCustomTaskDialog AClient);
```

Description

Use Create to create an instance of TTaskDialogProgressBar (see page 11523).

61.1.28.3 TTaskDialogProgressBar Methods

The methods of the TTaskDialogProgressBar class are listed here.

Public Methods

	Name	Description
	Initialize (see page 11525)	Initialize an instance of a task dialog progress bar.

61.1.28.3.1 TTaskDialogProgressBar.Initialize Method

Initialize an instance of a task dialog progress bar.

Pascal

```
procedure Initialize;
```

C++

```
__fastcall Initialize();
```

Description

Use Initialize to initialize an instance of TTaskDialogProgressBar (see page 11523).

61.1.28.4 TTaskDialogProgressBar Properties

The properties of the TTaskDialogProgressBar class are listed here.

Published Properties

	Name	Description
	MarqueeSpeed (see page 11526)	Controls (see page 5711) the marquee speed of an indeterminate progress bar.
	Max (see page 11526)	Specifies the upper limit of the value of the progress bar.
	Min (see page 11526)	Specifies the minimum value of the progress bar.
	Position (see page 11526)	Specifies the current position of the progress bar.
	State (see page 11527)	The state of the progress bar.

61.1.28.4.1 TTaskDialogProgressBar.MarqueeSpeed Property

Controls (see page 5711) the marquee speed of an indeterminate progress bar.

Pascal

```
property MarqueeSpeed: Cardinal;
```

C++

```
__property unsigned MarqueeSpeed;
```

Description

MarqueeSpeed controls the time it takes for the progress block to scroll across the progress bar in milliseconds..

61.1.28.4.2 TTaskDialogProgressBar.Max Property

Specifies the upper limit of the value of the progress bar.

Pascal

```
property Max: Integer;
```

C++

```
__property int Max;
```

Description

Max controls the maximum value of the progress bar.

61.1.28.4.3 TTaskDialogProgressBar.Min Property

Specifies the minimum value of the progress bar.

Pascal

```
property Min: Integer;
```

C++

```
__property int Min;
```

Description

Min controls the minimum value of the progress bar.

61.1.28.4.4 TTaskDialogProgressBar.Position Property

Specifies the current position of the progress bar.

Pascal

```
property Position: Integer;
```

C++

```
__property int Position;
```

Description

Position describes the current position of the progress bar.

61.1.28.4.5 TTaskDialogProgressBar.State Property

The state of the progress bar.

Pascal

```
property State: TProgressBarState;
```

C++

```
__property TProgressBarState State;
```

Description

State describes the state of the progress bar: pbsNormal, pbsError, or pbsPaused.

61.1.29 TTaskDialogRadioButtonItem Class

A single radio button item for Task Dialogs (see page 11265)..

Class Hierarchy

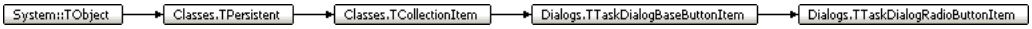

Pascal

```
TTaskDialogRadioButtonItem = class(TTaskDialogBaseButtonItem);
```

C++

```
class TTaskDialogRadioButtonItem : public TTaskDialogBaseButtonItem;
```

File

Dialogs

Description

An instance of TTaskDialogRadioButtonItem represents a single radio button for a Task Dialog.

61.1.29.1 TTaskDialogRadioButtonItem Members

The following tables list the members exposed by TTaskDialogRadioButtonItem.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Click (see page 11509)	Clicks the button.
V	Create (see page 11529)	Creates a single Task dialog radio button.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 11509)	Destroys a button.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetInitialState (see page 11511)	Set the initial state of the button.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
	ModalResult (see page 11512)	Modal result of pushing this button.
R	TextWStr (see page 11513)	Descriptive button text.

Published Properties

	Name	Description
	Caption (see page 11511)	The caption for the button.

	Default (see page 11512)	Indicates whether this button is the default.
	Enabled (see page 11512)	Indicates whether this button is enabled.

Strict protected Methods

	Name	Description
V	DoButtonClick (see page 11529)	Performs the click actions for the radio button.
V	DoSetEnabled (see page 11529)	Enable or disable the radio button.
V	GetButtonText (see page 11510)	Fetch the button text.
V	GetDisplayName (see page 11510)	Get display name..

Strict protected Properties

	Name	Description
R	Client (see page 11512)	Client area of the button.

61.1.29.2 TTaskDialogRadioButtonItem.Create Constructor

Creates a single Task dialog radio button.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TTaskDialogRadioButtonItem(TCollection Collection);
```

Description

Use Create to create a single instance of the TTaskDialogRadioButtonItem (see page 11527) class.

61.1.29.3 TTaskDialogRadioButtonItem Methods

The methods of the TTaskDialogRadioButtonItem class are listed here.

Strict protected Methods

	Name	Description
V	DoButtonClick (see page 11529)	Performs the click actions for the radio button.
V	DoSetEnabled (see page 11529)	Enable or disable the radio button.

61.1.29.3.1 TTaskDialogRadioButtonItem.DoButtonClick Method

Performs the click actions for the radio button.

Pascal

```
procedure DoButtonClick; override;
```

C++

```
virtual __fastcall DoButtonClick();
```

Description

Use DoButtonClick to perform the actions specified on the click of the radio button.

61.1.29.3.2 TTaskDialogRadioButtonItem.DoSetEnabled Method

Enable or disable the radio button.

Pascal

```
procedure DoSetEnabled; override;
```

C++

```
virtual __fastcall DoSetEnabled();
```

Description

Use DoSetEnabled to enable or disable the radio button.

61.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CreateMessageDialog (see page 11531)	Creates a specified message dialog.
	CreateMessageDialog (see page 11531)	Creates a specified message dialog.
	InputBox (see page 11532)	Displays an input dialog box that lets the user enter a string, double, or integer.
	InputQuery (see page 11533)	Displays an input dialog box that lets the user enter a string, double, or integer.
	MessageDlg (see page 11535)	Displays a message dialog box in the center of the screen.
	MessageDlg (see page 11535)	Displays a message dialog box in the center of the screen.
	MessageDlgPos (see page 11537)	Displays a message dialog box at the specified screen coordinates.
	MessageDlgPos (see page 11537)	Displays a message dialog box at the specified screen coordinates.
	MessageDlgPosHelp (see page 11539)	Displays a message dialog box whose help is supplied in a named help file.
	MessageDlgPosHelp (see page 11539)	Displays a message dialog box whose help is supplied in a named help file.
	PromptForFileName (see page 11540)	Displays an open or save dialog, where the user can specify a file name.
	ShowMessage (see page 11541)	Displays a message box with an OK button.
	ShowMessageFmt (see page 11542)	Displays a message box with a formatted message.
	ShowMessagePos (see page 11543)	Displays a message box at a specified location.
	TaskMessageDlg (see page 11543)	Displays a message task dialog box in the center of the screen.
	TaskMessageDlg (see page 11543)	Displays a message task dialog box in the center of the screen.
	TaskMessageDlgPos (see page 11545)	Displays a message task dialog box at the specified screen coordinates.
	TaskMessageDlgPos (see page 11545)	Displays a message task dialog box at the specified screen coordinates.
	TaskMessageDlgPosHelp (see page 11546)	Displays a message task dialog box whose help is supplied in a named help file.
	TaskMessageDlgPosHelp (see page 11546)	Displays a message task dialog box whose help is supplied in a named help file.

61.2.1 Dialogs.CreateMessageDialog Function

Creates a specified message dialog.

Pascal

```
function CreateMessageDialog(const Msg: string; DlgType: TMsgDlgType; Buttons:
TMsgDlgButtons): TForm; overload;
function CreateMessageDialog(const Msg: string; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons;
DefaultButton: TMsgDlgBtn): TForm; overload;
```

C++

```
TForm * CreateMessageDialog(const AnsiString Msg, TMsgDlgType DlgType, TMsgDlgButtons Buttons);
TForm * CreateMessageDialog(const AnsiString Msg, TMsgDlgType DlgType, TMsgDlgButtons Buttons,
TMsgDlgBtn DefaultButton);
```

File

Dialogs

Description

Call CreateMessageDialog to create a message dialog that will be used several times in an application. CreateMessageDialog returns a dialog of the type specified by the DlgType parameter and with the buttons indicated by the Buttons (see page 2289) parameter. Users can copy messages created using CreateMessageDialog to the clipboard by pressing Ctrl+C.

Note: CreateMessageDialog does not display the dialog form. The application must display this form when it is needed by calling its ShowModal method.

See Also

MessageDlg (see page 11535)

MessageDlgPos (see page 11537)

MessageDlgPosHelp (see page 11539)

ShowMessage (see page 11541)

ShowMessageFmt (see page 11542)

TCustomForm

TCustomForm

61.2.2 Dialogs.InputBox Function

Displays an input dialog box that lets the user enter a string, double, or integer.

Pascal

```
function InputBox(const ACaption: string; const APrompt: string; const ADefault: string):
string;
```

C++

```
AnsiString InputBox(const AnsiString ACaption, const AnsiString APrompt, const AnsiString
ADefault);
```

File

Dialogs

Description

Call InputBox to bring up an input dialog box ready for the user to enter a string, double, or integer in its edit box.

ACaption is the caption of the dialog box.

APrompt is the text that prompts the user to enter input in the edit box.

ADefault is the value that appears in the edit box when the dialog box first appears.

AMin is the minimum value the user can enter into the edit box.

AMax is the maximum value the user can enter into the edit box.

Decimals does nothing.

Increment controls the amount by which the value in the spin control changes when the user clicks the up or down arrow. (It has no effect on values that the user types).

If the user chooses the Cancel button, InputBox returns the default value. If the user chooses the OK button, InputBox returns the value in the edit box.

Use the InputBox function when there is a default value that should be used when the user chooses the Cancel button (or presses Esc) to exit the dialog. If the application needs to know whether the user chooses OK or Cancel, use the InputQuery (see page 11533) function instead.

C++ Examples:

```
/*
This example displays an input dialog box when the user
clicks a button on the form. The input dialog box includes a
prompt string and a default string. The string the user enters
in the dialog box is stored in the InputString variable and
the text edit.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString InputString = InputBox("Input Box", "Prompt", "Default string");
  Edit1->Text = InputString;
}
```

Delphi Examples:

```
{
This example displays an input dialog box when the user
clicks a button on the form. The input dialog box includes a
prompt string and a default string. The string the user enters
in the dialog box is stored in the InputString variable and
the text edit.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  InputString: string;
begin
  InputString:= Dialogs.InputBox('Input Box', 'Prompt', 'Default string');
  Edit1.Text:= InputString;
end;
```

See Also

InputQuery (see page 11533)

MessageDlg (see page 11535)

MessageDlgPos (see page 11537)

61.2.3 Dialogs.InputQuery Function

Displays an input dialog box that lets the user enter a string, double, or integer.

Pascal

```
function InputQuery(const ACaption: string; const APrompt: string; var Value: string): Boolean;
```

C++

```
Boolean InputQuery(const AnsiString ACaption, const AnsiString APrompt, AnsiString Value);
```

File

Dialogs

Description

Call InputQuery to bring up an input dialog box ready for the user to enter a string, double, or integer in its edit box.

ACaption is the caption of the dialog box.

APrompt is the text that prompts the user to enter input in the edit box.

Value is the value that appears in the edit box when the dialog box first appears and which returns the value that the user enters.

AMin is the minimum value the user can enter into the edit box.

AMax is the maximum value the user can enter into the edit box.

Decimals does nothing.

Increment controls the amount by which the value in the spin control changes when the user clicks the up or down arrow. (It has no effect on values that the user types).

InputQuery returns true if the user chooses OK, and false if the user chooses Cancel or presses the Esc key.

If a default value should be used when the user cancels out of the dialog, use InputBox (see page 11532) instead.

C++ Examples:

```
/*
This example uses a button and a label on the form. When the
user clicks the button, the input box displays. If the user
chooses OK, the string that appears in the edit box of the
dialog box displays as the caption of the label on the form.
If the user chooses Cancel, the dialog box closes and the
caption of the label remains unchanged.
*/
#include <dialogs.hpp>
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString NewString = "Default String";
  Label1->Caption = NewString;
  if (InputQuery("Input Box", "Prompt", NewString))
    // NewString has been changed by the user, who clicked ok
    Label1->Caption = NewString;
}
```

Delphi Examples:

```
{
This example uses a button and a label on the form. When the
user clicks the button, the input box displays. If the user
chooses OK, the string that appears in the edit box of the
dialog box displays as the caption of the label on the form.
If the user chooses Cancel, the dialog box closes and the
caption of the label remains unchanged.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  NewString: string;
  ClickedOK: Boolean;
begin
  NewString := 'Default String';
  Label1.Caption := NewString;
  ClickedOK := Dialogs.InputQuery('Input Box', 'Prompt', NewString);
  if ClickedOK then            { NewString contains new input string }
    Label1.Caption := 'The new string is ''' + NewString + '''';
end;
```

See Also

InputBox (see page 11532)

MessageDlg (see page 11535)

MessageDlgPos (see page 11537)

61.2.4 Dialogs.MessageDlg Function

Displays a message dialog box in the center of the screen.

Pascal

```
function MessageDlg(const Msg: string; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons; HelpCtx:
Longint): Integer; overload;
function MessageDlg(const Msg: string; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons; HelpCtx:
Longint; DefaultButton: TMsgDlgBtn): Integer; overload;
```

C++

```
int MessageDlg(const AnsiString Msg, TMsgDlgType DlgType, TMsgDlgButtons Buttons, Longint
HelpCtx);
int MessageDlg(const AnsiString Msg, TMsgDlgType DlgType, TMsgDlgButtons Buttons, Longint
HelpCtx, TMsgDlgBtn DefaultButton);
```

File

Dialogs

Description

Call MessageDlg to bring up a message box and obtain the user's response.

Msg is the content of the message that appears.

DlgType indicates the purpose of the dialog.

Buttons (see page 2289) indicates what buttons should appear in the message box. For a three button message box, use the syntax containing Button1, Button2, and Button3.

Button1, **Button2**, and **Button3** indicate what types of buttons to use for a three button message box. The resulting buttons appear in order.

HelpCtx specifies the context ID for the help topic that should appear when the user clicks the help button or presses F1 while the dialog is displayed.

DefaultBtn specifies which button from among those specified by Buttons (see page 2289) (or Button1, Button2, and Button3) is the default button for the dialog. If DefaultBtn is mbNone, there is no default button.

MessageDlg returns the value of the button the user selected. The following table lists the TMsgDlgBtn (see page 11556) values for each type of button that can appear in the message box, and the corresponding value that is returned if the user selects that button:

TMsgDlgBtn (see page 11556) Value	Corresponding return value
mbOK	mrOK
mbCancel	mrCancel
mbYes	mrYes
mbNo	mrNo
mbAbort	mrAbort
mbRetry	mrRetry
mbIgnore	mrIgnore
mbAll	mrAll
mbNoToAll	mrNoToAll

mbYesToAll	mrYesToAll

Note: If the user types Ctrl+C in the message box, the text of the message is copied to the clipboard.

C++ Examples:

```
/*
The following code adds a new object to a list in a list
object and then removes it:
*/
class TMyClass : public TComponent
{
__published:    // IDE-managed Components
private:    // User declarations
public:     // User declarations
  AnsiString MyString;
    __fastcall TMyClass(TComponent* Owner, AnsiString mystr);
};

__fastcall TMyClass::TMyClass(TComponent* Owner, AnsiString mystr)
    : TComponent(Owner)
{
  MyString = mystr;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TList *list = new TList();
  TMyClass *TheObject = new TMyClass(Form1, "This is an object.");
  try
  {
    list->Add(TheObject); // add AnsiString instance to list
    MessageDlg("The list has " + IntToStr(list->Count) + " objects",
               mtInformation, TMsgDlgButtons() << mbOK, 0);
    list->Remove(TheObject);
    MessageDlg("The list has " + IntToStr(list->Count) + " objects",
               mtInformation, TMsgDlgButtons() << mbOK, 0);
  }
  __finally
  {
    delete list;
  }
}
```

Delphi Examples:

```
{
This example uses a button on a form. When the user clicks
the button, a message box appears, asking if the user wants
to exit the application. If the user chooses Yes, another
dialog box appears informing the user the application is
about to end. When user chooses OK, the application ends.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if Dialogs.MessageDlg('Welcome to my Delphi application.  Exit now?',
    mtConfirmation, [mbYes, mbNo], 0, mbYes) = mrYes then
  begin
    Dialogs.MessageDlg('Exiting the Delphi application.', mtInformation,
      [mbOk], 0, mbOk);
    Close;
  end;
end;
```

See Also

Remove and MessageDlg example

TCustomForm (see page 13849)

MessageDlgPos (see page 11537)

ShowMessage (see page 11541)

ShowMessageFmt (see page 11542)

TCustomForm (see page 13849)

CreateMessageDialog (see page 11531)

61.2.5 Dialogs.MessageDlgPos Function

Displays a message dialog box at the specified screen coordinates.

Pascal

```
function MessageDlgPos(const Msg: string; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons;
HelpCtx: Longint; X: Integer; Y: Integer): Integer; overload;
function MessageDlgPos(const Msg: string; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons;
HelpCtx: Longint; X: Integer; Y: Integer; DefaultButton: TMsgDlgBtn): Integer; overload;
```

C++

```
int MessageDlgPos(const AnsiString Msg, TMsgDlgType DlgType, TMsgDlgButtons Buttons, Longint
HelpCtx, int X, int Y);
int MessageDlgPos(const AnsiString Msg, TMsgDlgType DlgType, TMsgDlgButtons Buttons, Longint
HelpCtx, int X, int Y, TMsgDlgBtn DefaultButton);
```

File

Dialogs

Description

Call MessageDlgPos to bring up a message box at a particular location when you do not need to specify a caption. (If you need to specify the caption as well, use the MessageDlg (see page 11535) function).

Msg is the content of the message that appears.

DlgType indicates the purpose of the dialog.

Buttons (see page 2289) indicates what buttons should appear in the message box.

HelpCtx specifies the context ID for the help topic that should appear when the user clicks the help button or presses F1 while the dialog is displayed.

X and **Y** specify the screen coordinates where the dialog should appear. A value of –1 means that the message box can appear anywhere in the specified dimension.

DefaultBtn specifies which button from among those specified by Buttons (see page 2289) is the default button for the dialog. If DefaultBtn is mbNone, there is no default button.

MessageDlgPos returns the value of the button the user selected. The following table lists the TMsgDlgBtn (see page 11556) values for each type of button that can appear in the message box, and the corresponding value that is returned if the user selects that button:

TMsgDlgBtn (see page 11556) Value	Corresponding return value
mbOK	mrOK
mbCancel	mrCancel
mbYes	mrYes
mbNo	mrNo
mbAbort	mrAbort
mbRetry	mrRetry
mbIgnore	mrIgnore
mbAll	mrAll
mbNoToAll	mrNoToAll
mbYesToAll	mrYesToAll

Note: If the user types Ctrl+C in the message box, the text of the message is copied to the clipboard.

C++ Examples:

```
/*
This example uses a button on a form. When the user clicks
the button, a message box appears with a Yes, No, and Cancel
button on it:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // can't do TMsgDlgButtons() << mbYesNoCancel
  MessageDlgPos("Are you there?", mtConfirmation, mbYesNoCancel, 0, 200, 200, mbYes);
}
```

Delphi Examples:

```
{
This example uses a button on a form. When the user clicks
the button, a message box appears with a Yes, No, and Cancel
button on it:
}

procedure TForm1.Button1Click(Sender: TObject);

begin
  // can't do TMsgDlgButtons() << mbYesNoCancel
  Dialogs.MessageDlgPos('Are you there?', mtConfirmation, mbYesNoCancel, 0, 200, 200, mbYes);
end;
```

See Also

Remove and MessageDlg example

MessageDlg (see page 11535)

ShowMessage (see page 11541)

ShowMessageFmt (see page 11542)

CreateMessageDialog (see page 11531)

61.2.6 Dialogs.MessageDlgPosHelp Function

Displays a message dialog box whose help is supplied in a named help file.

Pascal

```
function MessageDlgPosHelp(const Msg: string; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons;
HelpCtx: Longint; X: Integer; Y: Integer; const HelpFileName: string): Integer; overload;
function MessageDlgPosHelp(const Msg: string; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons;
HelpCtx: Longint; X: Integer; Y: Integer; const HelpFileName: string; DefaultButton:
TMsgDlgBtn): Integer; overload;
```

C++

```
int MessageDlgPosHelp(const AnsiString Msg, TMsgDlgType DlgType, TMsgDlgButtons Buttons,
Longint HelpCtx, int X, int Y, const AnsiString HelpFileName);
int MessageDlgPosHelp(const AnsiString Msg, TMsgDlgType DlgType, TMsgDlgButtons Buttons,
Longint HelpCtx, int X, int Y, const AnsiString HelpFileName, TMsgDlgBtn DefaultButton);
```

File

Dialogs

Description

Call MessageDlgPosHelp to bring up a message box when the associated help comes from a help file other than the default help file.

Msg is the content of the message that appears.

DlgType indicates the purpose of the dialog.

Buttons (see page 2289) indicates what buttons should appear in the message box.

HelpCtx specifies the context ID for the help topic in the file specified by HelpFileName. This topic appears when the user clicks the help button or presses F1 while the dialog is displayed.

X and Y specify the screen coordinates where the dialog should appear.

HelpFileName specifies the help file that contains the help topic for this message box.

MessageDlgPosHelp returns the value of the button the user selected. The following table lists the TMsgDlgBtn (see page 11556) values for each type of button that can appear in the message box, and the corresponding value that is returned if the user selects that button:

TMsgDlgBtn (see page 11556) Value	Corresponding return value
mbOK	mrOk
mbCancel	mrCancel
mbYes	mrYes
mbNo	mrNo
mbAbort	mrAbort
mbRetry	mrRetry
mbIgnore	mrIgnore
mbAll	mrAll
mbNoToAll	mrNoToAll
mbYesToAll	mrYesToAll

Note: If the user types Ctrl+C in the message box, the text of the message is copied to the clipboard.

See Also

MessageDlg (see page 11535)

ShowMessage (see page 11541)

ShowMessageFmt (see page 11542)

61.2.7 Dialogs.PromptForFileName Function

Displays an open or save dialog, where the user can specify a file name.

Pascal

```
function PromptForFileName(var AFileName: string; const AFilter: string = ''; const
ADefaultExt: string = ''; const ATitle: string = ''; const AInitialDir: string = '';
SaveDialog: Boolean = False): Boolean;
```

C++

```
Boolean PromptForFileName(AnsiString AFileName, const AnsiString AFilter = '', const
AnsiString ADefaultExt = '', const AnsiString ATitle = '', const AnsiString AInitialDir = '',
Boolean SaveDialog = False);
```

File

Dialogs

Description

Call PromptForFileName to display an open or save dialog.

AFilename is the name of the file. The value supplied by the caller is used to initialize the file name in the dialog. On exit, AFileName contains the name specified in the dialog.

AFilter determines the file masks (filters) available in the dialog. If AFilter is set to an empty string, the dialog uses the string 'All files (*.*)', which causes the dialog to display all files in the current directory. For information on filter strings, see Filter.

ADefaultExt specifies a file extension that is appended automatically to the specified file name, unless the selected file name already includes a registered extension. If the user specifies a file name with an extension that is unregistered, ADefaultExt is appended to the unregistered extension.

ATitle specifies the title for the dialog. If ATitle is set to an empty string, the dialog uses 'Open' or 'Save As', depending on the value of the SaveDialog parameter.

AInitialDir specifies the directory displayed in the dialog when it opens. If AInitialDir is an empty string, or if the specified directory does not exist, the initial directory is the current working directory.

SaveDialog specifies whether the file selection should be a Save As dialog (true) or an Open dialog (false).

Options specifies file dialog options.

PromptForFileName returns true if the user selected a file name, false if the user exited by clicking the cancel button.

See Also

TOpenDialog (see page 11407)

TSaveDialog (see page 11489)

SelectDirectory

61.2.8 Dialogs.ShowMessage Function

Displays a message box with an OK button.

Pascal

```
procedure ShowMessage(const Msg: string);
```

C++

```
ShowMessage(const AnsiString Msg);
```

File

Dialogs

Description

Call ShowMessage to display a simple message box with an OK button. The name of the application's executable file appears as the caption of the message box.

Msg parameter is the message string that appears in the message box.

Params lists the values to insert into Msg if the Msg string includes formatting specifiers. For more information about how messages are formatted, see Format Strings.

Params_size is the index of the last value in Params (one less than the number of elements).

Note: To display a message in a message box with other buttons, or with an icon, use the MessageDlg (see page 11535) function.

Note: If the user types Ctrl+C in the message box, the text of the message is copied to the clipboard.

C++ Examples:

```
/*
The following example uses an edit control and a button on a
form. When the button is clicked, the current directory is
searched for the filename specified in the edit control. A
message box indicates whether the file is found.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString asFileName = FileSearch(Edit1->Text, GetCurrentDir());
  if (asFileName.IsEmpty())
    ShowMessage(AnsiString("Couldn't find ") + Edit1->Text + ".");
  else
    ShowMessage(AnsiString("Found ") + asFileName + ".");
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Label2->Caption = GetCurrentDir();
}
```

Delphi Examples:

```
{
```

```
The following example uses an edit control and a button on a
form. When the button is clicked, the current directory is
searched for the filename specified in the edit control. A
message box indicates whether the file is found.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FileToFind: string;
begin
  FileToFind := SysUtils.FileSearch(Edit1.Text, GetCurrentDir);
  if FileToFind = '' then
    ShowMessage('Couldn''t find ' + Edit1.Text + '.')
  else
    ShowMessage('Found ' + FileToFind + '.');
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Label2.Caption := GetCurrentDir;
end;
```

See Also

MessageDlg (see page 11535)

MessageDlgPos (see page 11537)

ShowMessageFmt (see page 11542)

TApplication

ShowMessagePos (see page 11543)

61.2.9 Dialogs.ShowMessageFmt Function

Displays a message box with a formatted message.

Pascal

```
procedure ShowMessageFmt(const Msg: string; Params: array of const);
```

C++

```
ShowMessageFmt(const AnsiString Msg, array of const Params);
```

File

Dialogs

Description

Call ShowMessageFmt to display a simple message box with an OK button. The name of the application's executable file appears as the caption of the message box.

The Msg parameter is a format string for the message that appears in the message box. The Params parameter provides the parameters that are assembled into the message. The Params_Size parameter gives the index of the element of the Params array (one less than the number of parameters).

For more information about how messages are formatted, see Format Strings.

To display a message in a message box with other buttons, or with an icon, use the MessageDlg (see page 11535) function.

Note: If the user types Ctrl+C in the message box, the text of the message is copied to the clipboard.

See Also

MessageDlg (see page 11535)

ShowMessage (see page 11541)

ShowMessagePos (see page 11543)

61.2.10 Dialogs.ShowMessagePos Function

Displays a message box at a specified location.

Pascal

```
procedure ShowMessagePos(const Msg: string; X: Integer; Y: Integer);
```

C++

```
ShowMessagePos(const AnsiString Msg, int X, int Y);
```

File

Dialogs

Description

Call ShowMessagePos to display a simple message box with an OK button at a specified location. The name of the application's executable file appears as the caption of the message box.

The Msg parameter is the message that appears in the message box.

X and Y specify the screen coordinates where the message box should appear. A value of –1 means that the message box can appear anywhere in the specified dimension.

To display a message in a message box with other buttons, or with an icon, use the MessageDlg (see page 11535) function.

Note: If the user types Ctrl+C in the message box, the text of the message is copied to the clipboard.

See Also

MessageDlg (see page 11535)

MessageDlgPos (see page 11537)

ShowMessage (see page 11541)

ShowMessageFmt (see page 11542)

61.2.11 Dialogs.TaskMessageDlg Function

Displays a message task dialog box in the center of the screen.

Pascal

```
function TaskMessageDlg(const Title: string; const Msg: string; DlgType: TMsgDlgType; Buttons:
TMsgDlgButtons; HelpCtx: Longint): Integer; overload;
function TaskMessageDlg(const Title: string; const Msg: string; DlgType: TMsgDlgType; Buttons:
```

```
TMsgDlgButtons; HelpCtx: Longint; DefaultButton: TMsgDlgBtn): Integer; overload;
```

C++

```
int TaskMessageDlg(const AnsiString Title, const AnsiString Msg, TMsgDlgType DlgType,
TMsgDlgButtons Buttons, Longint HelpCtx);
int TaskMessageDlg(const AnsiString Title, const AnsiString Msg, TMsgDlgType DlgType,
TMsgDlgButtons Buttons, Longint HelpCtx, TMsgDlgBtn DefaultButton);
```

File

Dialogs

Description

Call TaskMessageDlg to bring up a message box and obtain the user's response.

Caption specifies the caption for the message box when DlgType is mtCustom. If this parameter is not used, the caption for custom message boxes is the title of the application.

Msg is the content of the message that appears.

DlgType indicates the purpose of the dialog.

Buttons (see page 2289) indicates what buttons should appear in the message box. For a three button message box, use the syntax containing Button1, Button2, and Button3.

Button1, Button2, and Button3 indicate what types of buttons to use for a three button message box. The resulting buttons appear in order.

HelpCtx specifies the context ID for the help topic that should appear when the user clicks the help button or presses F1 while the dialog is displayed.

X and Y specify the screen coordinates where the dialog should appear. A value of –1 means that the message box can appear anywhere in the specified dimension.

DefaultBtn specifies which button from among those specified by Buttons (see page 2289) (or Button1, Button2, and Button3) is the default button for the dialog. If DefaultBtn is mbNone, there is no default button.

Bitmap is an image that appears on the face of the message dialog. If Bitmap is nil (Delphi) or NULL (C++), there is no image unless DlgType is mtConfirmation.

MessageDlg (see page 11535) returns the value of the button the user selected. The following table lists the TMsgDlgBtn (see page 11556) values for each type of button that can appear in the message box, and the corresponding value that is returned if the user selects that button:

TMsgDlgBtn (see page 11556) Value	Corresponding return value
mbOk	mrOk
mbOK	mrOk
mbCancel	mrCancel
mbYes	mrYes
mbNo	mrNo
mbAbort	mrAbort
mbRetry	mrRetry
mbIgnore	mrIgnore
mbAll	mrAll
mbNoToAll	mrNoToAll
mbYesToAll	mrYesToAll

Note: If the user types Ctrl+C in the message box, the text of the message is copied to the clipboard.

61.2.12 Dialogs.TaskMessageDlgPos Function

Displays a message task dialog box at the specified screen coordinates.

Pascal

```
function TaskMessageDlgPos(const Title: string; const Msg: string; DlgType: TMsgDlgType;
Buttons: TMsgDlgButtons; HelpCtx: Longint; X: Integer; Y: Integer): Integer; overload;
function TaskMessageDlgPos(const Title: string; const Msg: string; DlgType: TMsgDlgType;
Buttons: TMsgDlgButtons; HelpCtx: Longint; X: Integer; Y: Integer; DefaultButton: TMsgDlgBtn):
Integer; overload;
```

C++

```
int TaskMessageDlgPos(const AnsiString Title, const AnsiString Msg, TMsgDlgType DlgType,
TMsgDlgButtons Buttons, Longint HelpCtx, int X, int Y);
int TaskMessageDlgPos(const AnsiString Title, const AnsiString Msg, TMsgDlgType DlgType,
TMsgDlgButtons Buttons, Longint HelpCtx, int X, int Y, TMsgDlgBtn DefaultButton);
```

File

Dialogs

Description

Call TaskMessageDlgPos to bring up a message box at a particular location when you do not need to specify a caption. (If you need to specify the caption as well, use the MessageDlg (see page 11535) function).

Msg is the content of the message that appears.

DlgType indicates the purpose of the dialog.

Buttons (see page 2289) indicates what buttons should appear in the message box.

HelpCtx specifies the context ID for the help topic that should appear when the user clicks the help button or presses F1 while the dialog is displayed.

X and Y specify the screen coordinates where the dialog should appear. A value of –1 means that the message box can appear anywhere in the specified dimension.

DefaultBtn specifies which button from among those specified by Buttons (see page 2289) is the default button for the dialog. If DefaultBtn is mbNone, there is no default button.

Bitmap is an image that appears on the face of the message dialog. If Bitmap is nil (Delphi) or NULL (C++), there is no image unless DlgType is mtConfirmation.

MessageDlgPos (see page 11537) returns the value of the button the user selected. The following table lists the TMsgDlgBtn (see page 11556) values for each type of button that can appear in the message box, and the corresponding value that is returned if the user selects that button:

TMsgDlgBtn (see page 11556) Value	Corresponding return value
mbOk	mrOk
mbOK	mrOk
mbCancel	mrCancel
mbYes	mrYes

mbNo	mrNo
mbAbort	mrAbort
mbRetry	mrRetry
mbIgnore	mrIgnore
mbAll	mrAll
mbNoToAll	mrNoToAll
mbYesToAll	mrYesToAll

Note: If the user types Ctrl+C in the message box, the text of the message is copied to the clipboard.

61.2.13 Dialogs.TaskMessageDlgPosHelp Function

Displays a message task dialog box whose help is supplied in a named help file.

Pascal

```
function TaskMessageDlgPosHelp(const Title: string; const Msg: string; DlgType: TMsgDlgType;
Buttons: TMsgDlgButtons; HelpCtx: Longint; X: Integer; Y: Integer; const HelpFileName:
string): Integer; overload;
function TaskMessageDlgPosHelp(const Title: string; const Msg: string; DlgType: TMsgDlgType;
Buttons: TMsgDlgButtons; HelpCtx: Longint; X: Integer; Y: Integer; const HelpFileName: string;
DefaultButton: TMsgDlgBtn): Integer; overload;
```

C++

```
int TaskMessageDlgPosHelp(const AnsiString Title, const AnsiString Msg, TMsgDlgType DlgType,
TMsgDlgButtons Buttons, Longint HelpCtx, int X, int Y, const AnsiString HelpFileName);
int TaskMessageDlgPosHelp(const AnsiString Title, const AnsiString Msg, TMsgDlgType DlgType,
TMsgDlgButtons Buttons, Longint HelpCtx, int X, int Y, const AnsiString HelpFileName,
TMsgDlgBtn DefaultButton);
```

File

Dialogs

Description

Call TaskMessageDlgPosHelp to bring up a message box when the associated help comes from a help file other than the default help file.

Msg is the content of the message that appears.

DlgType indicates the purpose of the dialog.

Buttons (see page 2289) indicates what buttons should appear in the message box.

HelpCtx specifies the context ID for the help topic in the file specified by HelpFileName. This topic appears when the user clicks the help button or presses F1 while the dialog is displayed.

X and Y specify the screen coordinates where the dialog should appear.

HelpFileName specifies the help file that contains the help topic for this message box.

MessageDlgPosHelp (see page 11539) returns the value of the button the user selected. The following table lists the TMsgDlgBtn (see page 11556) values for each type of button that can appear in the message box, and the corresponding value that is returned if the user selects that button:

TMsgDlgBtn (see page 11556) Value	Corresponding return value
mbOK	mrOk
mbCancel	mrCancel
mbYes	mrYes
mbNo	mrNo
mbAbort	mrAbort
mbRetry	mrRetry
mbIgnore	mrIgnore
mbAll	mrAll
mbNoToAll	mrNoToAll
mbYesToAll	mrYesToAll

Note: If the user types Ctrl+C in the message box, the text of the message is copied to the clipboard.

61.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TColorDialogOption (see page 11548)	TColorDialogOption and TColorDialogOptions (see page 11568) specify options for a TColorDialog (see page 11270) component.
	TFileDialogOption (see page 11549)	The settings in TFileDialogOption determine the behavior of a TCustomFileDialog (see page 11288) file dialog. TFileDialogOptions (see page 11571) is a set of TFileDialogOption. These options are the same as the options for the Microsoft IFileDialog::GetOptions and IFileDialog::SetOptions functions.
	TFileDialogOverwriteResponse (see page 11551)	TFileDialogOverwriteResponse indicates the application's response to an overwrite request during a save operation for a TCustomFileSaveDialog (see page 11314) dialog. These are the same values as in the FDE_OVERWRITE_RESPONSE enumerated type used in the Microsoft IFileDialogEvents::OnOverwrite method.
	TFileDialogShareViolationResponse (see page 11552)	TFileDialogShareViolationResponse indicates the application's response to a sharing violation during a file open or save for a TCustomFileDialog (see page 11288) dialog. These are the same values as in the FDE_SHAREVIOLATION_RESPONSE enumerated type used in the Microsoft IFileDialogEvents::OnShareViolation method.
	TFileEditStyle (see page 11552)	TFileEditStyle indicates the type of control that appears in a file selection dialog for the file name.
	TFindOption (see page 11553)	TFindOption and TFindOptions (see page 11573) determine the controls that appear in a find dialog.
	TFontDialogDevice (see page 11554)	TFontDialogDevice indicates the source from which a TFontDialog (see page 11394) instance obtains its list of fonts.
	TFontDialogOption (see page 11555)	TFontDialogOption and TFontDialogOptions (see page 11574) are flags that influence the appearance and behavior of a font dialog.
	TMsgDlgBtn (see page 11556)	TMsgDlgBtn defines a set of values used by MessageDlg (see page 11535) and MessageDlgPos (see page 11537) functions.
	TMsgDlgType (see page 11558)	TMsgDlgType defines values describing the type of message box.
	TOpenOption (see page 11559)	TOpenOption and TOpenOptions (see page 11576) determine the behavior of a file selection dialog.
	TOpenOptionEx (see page 11561)	TOpenOptionEx and TOpenOptionsEx (see page 11578) determine the behavior of a file selection dialog.
	TPageMeasureUnits (see page 11561)	Specifies page setup measure unit types.
	TPageSetupDialogOption (see page 11562)	This is record Dialogs.TPageSetupDialogOption.
	TPageType (see page 11562)	TPageType indicates the type of page being configured for printing.
	TPrintDialogOption (see page 11563)	TPrintDialogOption and TPrintDialogOptions (see page 11580) specify options for a TPrintDialog (see page 11458) component.
	TPrinterKind (see page 11564)	This is record Dialogs.TPrinterKind.
	TPrintRange (see page 11564)	TPrintRange indicates how the user specified the set of pages to print.
	TProgressBarState (see page 11565)	TProgressBarState defines a set of values used by the State property.
	TTaskDialogCommonButton (see page 11565)	Buttons (see page 2289) for Task Dialogs (see page 11265).
	TTaskDialogFlag (see page 11566)	Flags for Task Dialogs (see page 11265).

61.3.1 Dialogs.TColorDialogOption Enumeration

TColorDialogOption and TColorDialogOptions (see page 11568) specify options for a TColorDialog (see page 11270) component.

Pascal

```
TColorDialogOption = (
  cdFullOpen,
  cdPreventFullOpen,
  cdShowHelp,
  cdSolidColor,
  cdAnyColor
);
```

C++

```
enum TColorDialogOption {
  cdFullOpen,
  cdPreventFullOpen,
  cdShowHelp,
  cdSolidColor,
  cdAnyColor
};
```

File

Dialogs

Description

TColorDialogOptions (see page 11568) is a set of TColorDialogOption values. The possible values of TColorDialogOption are

Value	Meaning
cdFullOpen	Display custom color options when the dialog opens.
cdPreventFullOpen	Disable the Define Custom Colors button in the dialog, so that the user cannot define new colors.
cdShowHelp	Add a Help button to the color dialog.
cdSolidColor	Direct Windows to use the nearest solid color to the color chosen.
cdAnyColor	Allow the user to select non-solid colors (which may have to be approximated by dithering).

61.3.2 Dialogs.TFileDialogOption Enumeration

The settings in TFileDialogOption determine the behavior of a TCustomFileDialog (see page 11288) file dialog. TFileDialogOptions (see page 11571) is a set of TFileDialogOption. These options are the same as the options for the Microsoft IFileDialog::GetOptions and IFileDialog::SetOptions functions.

Pascal

```
TFileDialogOption = (
  fdoOverWritePrompt,
  fdoStrictFileTypes,
  fdoNoChangeDir,
  fdoPickFolders,
  fdoForceFileSystem,
  fdoAllNonStorageItems,
  fdoNoValidate,
  fdoAllowMultiSelect,
  fdoPathMustExist,
  fdoFileMustExist,
  fdoCreatePrompt,
  fdoShareAware,
  fdoNoReadOnlyReturn,
  fdoNoTestFileCreate,
  fdoHideMRUPlaces,
```

```
    fdoHidePinnedPlaces,
    fdoNoDereferenceLinks,
    fdoDontAddToRecent,
    fdoForceShowHidden,
    fdoDefaultNoMiniMode,
    fdoForcePreviewPaneOn
  );
```

C++

```
  enum TFileDialogOption {
    fdoOverWritePrompt,
    fdoStrictFileTypes,
    fdoNoChangeDir,
    fdoPickFolders,
    fdoForceFileSystem,
    fdoAllNonStorageItems,
    fdoNoValidate,
    fdoAllowMultiSelect,
    fdoPathMustExist,
    fdoFileMustExist,
    fdoCreatePrompt,
    fdoShareAware,
    fdoNoReadOnlyReturn,
    fdoNoTestFileCreate,
    fdoHideMRUPlaces,
    fdoHidePinnedPlaces,
    fdoNoDereferenceLinks,
    fdoDontAddToRecent,
    fdoForceShowHidden,
    fdoDefaultNoMiniMode,
    fdoForcePreviewPaneOn
  };
```

File

Dialogs

Description

The following table lists the possible values:

Value	Meaning
fdoOverWritePrompt	Prompt before overwriting an existing file of the same name when saving a file. This is a default for save dialogs.
fdoStrictFileTypes	The file extension of a saved file being must match the selected file type.
fdoNoChangeDir	Unused.
fdoPickFolders	Choose folders rather than files.
fdoForceFileSystem	Returned items must be file system items.
fdoAllNonStorageItems	Allow users to choose any item in the Shell namespace. This flag cannot be combined with fdoForceFileSystem.
fdoNoValidate	Do not check for situations preventing applications from opening selected files, such as sharing violations or access denied errors.
fdoAllowMultiSelect	Allow selecting multiple items in an open dialog.
fdoPathMustExist	Items returned must be in an existing folder. This is a default.
fdoFileMustExist	Items returned must exist. This is a default value for open dialogs.
fdoCreatePrompt	Prompt for creation if returned item in save dialog does not exist. This does not create the item.
fdoShareAware	For a sharing violation opening a file, call the application back for guidance. This flag is overridden by fdoNoValidate.

fdoNoReadOnlyReturn	Do not return read-only items.
fdoNoTestFileCreate	Do not test creation of returned item from save dialogs. If not set, the calling application must handle errors discovered in the creation test.
fdoHideMRUPlaces	Hide places of recently opened or saved items.
fdoHidePinnedPlaces	Hide pinned places from which users can choose.
fdoNoDereferenceLinks	Shortcuts are not treated as their target items, allowing applications to open .lnk files.
fdoDontAddToRecent	Do not add the item being opened or saved to the list of recent places.
fdoForceShowHidden	Show hidden items.
fdoDefaultNoMiniMode	Open save dialog box in expanded mode in which users can browse folders. Expanded mode is set and unset by clicking the button in the lower-left corner of a save dialog box.
fdoForcePreviewPaneOn	Display the preview pane.

61.3.3 Dialogs.TFileDialogOverwriteResponse Enumeration

TFileDialogOverwriteResponse indicates the application's response to an overwrite request during a save operation for a TCustomFileSaveDialog (see page 11314) dialog. These are the same values as in the FDE_OVERWRITE_RESPONSE enumerated type used in the Microsoft IFileDialogEvents::OnOverwrite method.

Pascal

```
TFileDialogOverwriteResponse = (
  forDefault = FDEOR_DEFAULT,
  forAccept = FDEOR_ACCEPT,
  forRefuse = FDEOR_REFUSE
);
```

C++

```
enum TFileDialogOverwriteResponse {
  forDefault = FDEOR_DEFAULT,
  forAccept = FDEOR_ACCEPT,
  forRefuse = FDEOR_REFUSE
};
```

File

Dialogs

Description

The following table lists the possible values:

Value	Meaning
forDefault	The application displays a dialog asking the user whether or not to overwrite.
forAccept	The application returns the file name, allowing the overwrite.
forRefuse	The application does not return the file name, refusing the overwrite.

61.3.4 Dialogs.TFileDialogShareViolationResponse Enumeration

TFileDialogShareViolationResponse indicates the application's response to a sharing violation during a file open or save for a TCustomFileDialog (see page 11288) dialog. These are the same values as in the FDE_SHAREVIOLATION_RESPONSE enumerated type used in the Microsoft IFileDialogEvents::OnShareViolation method.

Pascal

```
TFileDialogShareViolationResponse = (
  fsrDefault = FDESVR_DEFAULT,
  fsrAccept = FDESVR_ACCEPT,
  fsrRefuse = FDESVR_REFUSE
);
```

C++

```
enum TFileDialogShareViolationResponse {
  fsrDefault = FDESVR_DEFAULT,
  fsrAccept = FDESVR_ACCEPT,
  fsrRefuse = FDESVR_REFUSE
};
```

File

Dialogs

Description

The following table lists the possible values:

Value	Meaning
fsrDefault	The application displays a dialog telling the user the file is in use allowing selection of another file.
fsrAccept	The application returns the file name.
fsrRefuse	The application does not return the file name.

61.3.5 Dialogs.TFileEditStyle Enumeration

TFileEditStyle indicates the type of control that appears in a file selection dialog for the file name.

Pascal

```
TFileEditStyle = (
  fsEdit,
  fsComboBox
);
```

C++

```
enum TFileEditStyle {
  fsEdit,
  fsComboBox
};
```

File

Dialogs

Description

TFileEditStyle can have one of the following values:

Value	Meaning
fsEdit	The control for entering the file name is an edit control.
fsComboBox	The control for entering the file name is a combo box.

61.3.6 Dialogs.TFindOption Enumeration

TFindOption and TFindOptions (see page 11573) determine the controls that appear in a find dialog.

Pascal

```
TFindOption = (
  frDown,
  frFindNext,
  frHideMatchCase,
  frHideWholeWord,
  frHideUpDown,
  frMatchCase,
  frDisableMatchCase,
  frDisableUpDown,
  frDisableWholeWord,
  frReplace,
  frReplaceAll,
  frWholeWord,
  frShowHelp
);
```

C++

```
enum TFindOption {
  frDown,
  frFindNext,
  frHideMatchCase,
  frHideWholeWord,
  frHideUpDown,
  frMatchCase,
  frDisableMatchCase,
  frDisableUpDown,
  frDisableWholeWord,
  frReplace,
  frReplaceAll,
  frWholeWord,
  frShowHelp
};
```

File

Dialogs

Description

TFindOption values determine which controls are enabled in a TFindDialog (see page 11376) instance, and how the controls are initialized.

TFindOptions (see page 11573) is a set of TFindOption values.

The following table lists the possible values:

Value	Meaning
frDisableMatchCase	Disables (grays) the Match Case check box in a find dialog.
frDisableUpDown	Disables (grays) the Up and Down buttons, which determine the direction of the search.
frDisableWholeWord	Disables (grays) the Match Whole Word check box of find dialog.
frDown	Selects the Down button by default when the dialog opens. If the frDown flags is off, Up is selected when the dialog opens. (By default, frDown is on.)
frFindNext	This flag is turned on when the user clicks the Find Next button and turned off when the dialog closes.
frHideMatchCase	Removes the Match Case check box from the dialog.
frHideWholeWord	Removes the Match Whole Word check box from the dialog.
frHideUpDown	Removes the Up and Down buttons from the dialog.
frMatchCase	This flag is turned on (off) when the user selects (deselects) the Match Case check box. To select the check box by default when the dialog opens, set frMatchCase at design time.
frReplace	Applies to TReplaceDialog (see page 11482) only. This flag is set by the system to indicate that the application should replace the current occurrence (and only the current occurrence) of the FindText string with the ReplaceText string. Not used in search routines.
frReplaceAll	Applies to TReplaceDialog (see page 11482) only. This flag is set by the system to indicate that the application should replace all occurrences of the FindText string with the ReplaceText string.
frShowHelp	Displays a Help button in the dialog.
frWholeWord	This flag is turned on (off) when the user selects (deselects) the Match Whole Word check box. To select the check box by default when the dialog opens, set frWholeWord at design time.

61.3.7 Dialogs.TFontDialogDevice Enumeration

TFontDialogDevice indicates the source from which a TFontDialog (see page 11394) instance obtains its list of fonts.

Pascal

```
TFontDialogDevice = (
  fdScreen,
  fdPrinter,
  fdBoth
);
```

C++

```
enum TFontDialogDevice {
  fdScreen,
  fdPrinter,
  fdBoth
};
```

File

Dialogs

Description

These are the possible values of TFondDialogDevice:

Value	Meaning
fdScreen	Retrieve fonts from the screen.
fdPrinter	Retrieve fonts from the printer.
fdBoth	Retrieve fonts from both the screen and the printer.

61.3.8 Dialogs.TFontDialogOption Enumeration

TFontDialogOption and TFontDialogOptions (see page 11574) are flags that influence the appearance and behavior of a font dialog.

Pascal

```
TFontDialogOption = (
  fdAnsiOnly,
  fdTrueTypeOnly,
  fdEffects,
  fdFixedPitchOnly,
  fdForceFontExist,
  fdNoFaceSel,
  fdNoOEMFonts,
  fdNoSimulations,
  fdNoSizeSel,
  fdNoStyleSel,
  fdNoVectorFonts,
  fdShowHelp,
  fdWysiwyg,
  fdLimitSize,
  fdScalableOnly,
  fdApplyButton
);
```

C++

```
enum TFontDialogOption {
  fdAnsiOnly,
  fdTrueTypeOnly,
  fdEffects,
  fdFixedPitchOnly,
  fdForceFontExist,
  fdNoFaceSel,
  fdNoOEMFonts,
  fdNoSimulations,
  fdNoSizeSel,
  fdNoStyleSel,
  fdNoVectorFonts,
  fdShowHelp,
  fdWysiwyg,
  fdLimitSize,
  fdScalableOnly,
  fdApplyButton
};
```

File

Dialogs

Description

TFontDialogOptions (see page 11574) is a set of TFontDialogOption values. These are the possible values of TFontDialogOption:

Value	Meaning
fdAnsiOnly	Displays only fonts that use the Windows character set. Symbol fonts are not listed in the dialog.
fdApplyButton	Displays an Apply button in the dialog, whether or not there is an OnApply event handler.
fdEffects	Displays the Effects check boxes (Strikeout and Underline) and the Color list box in the dialog.
fdFixedPitchOnly	Displays only monospaced fonts in the dialog. Proportionally spaced fonts are not listed, nor are TrueType fonts that do not have the fixed-pitch flag set in the style header.
fdForceFontExist	Allows the user to enter only fonts that are displayed in the dialog (that is, listed in the Font combo box). If the user tries to enter another font name, an error message appears.
fdLimitSize	Enables the MaxFontSize and MinFontSize properties, limiting the range of font sizes that appear in the dialog if these properties have values.
fdNoFaceSel	Causes the dialog to open without a preselected font name in the Font combo box.
fdNoOEMFonts	Removes OEM fonts from the dialog's combo box. Lists only non-OEM fonts.
fdScalableOnly	Displays only scalable fonts in the dialog. Non-scalable (bitmap) fonts are removed from the list.
fdNoSimulations	Displays only fonts and font styles that are directly supported by the font definition file. GDI-synthesized bold and italic styles (for bitmap fonts) are not listed.
fdNoSizeSel	Causes the dialog to open without a preselected size in the Size combo box.
fdNoStyleSel	Causes the dialog to open without a preselected style in the Font Style combo box.
fdNoVectorFonts	Removes vector fonts from the dialog's combo box; lists only non-vector fonts. (Vector fonts are Windows 1.0 fonts, such as Roman or Script, which resemble pen-plotter output.)
fdShowHelp	Displays a Help button in the dialog.
fdTrueTypeOnly	Displays only TrueType fonts in the dialog. Other fonts are not listed.
fdWysiwyg	Displays only fonts that are available to both the printer and the screen. Device-specific fonts are not listed in the dialog.

61.3.9 Dialogs.TMsgDlgBtn Enumeration

TMsgDlgBtn defines a set of values used by MessageDlg (see page 11535) and MessageDlgPos (see page 11537) functions.

Pascal

```
TMsgDlgBtn = (
  mbYes,
  mbNo,
  mbOK,
  mbCancel,
  mbAbort,
  mbRetry,
  mbIgnore,
  mbAll,
  mbNoToAll,
  mbYesToAll,
  mbHelp
);
```

C++

```
enum TMsgDlgBtn {
  mbYes,
```

```
    mbNo,
    mbOK,
    mbCancel,
    mbAbort,
    mbRetry,
    mbIgnore,
    mbAll,
    mbNoToAll,
    mbYesToAll,
    mbHelp
  };
```

File

Dialogs

Description

The TMsgDlgBtn enumeration defines the set of values a button in a message box can have. The TMsgDlgBtn values are used by the MessageDlg (see page 11535) and MessageDlgPos (see page 11537) functions.

The following tables lists the possible values:

Value	Meaning
mbYes	A button with 'Yes' on its face.
mbNo	A button the text 'No' on its face.
mbOK	A button the text 'OK' on its face.
mbCancel	A button with the text 'Cancel' on its face.
mbAbort	A button with the text 'Abort' on its face
mbRetry	A button with the text 'Retry' on its face
mbIgnore	A button the text 'Ignore' on its face
mbAll	A button with the text 'All' on its face
mbNoToAll	A button with the text 'No to All' on its face
mbYesToAll	A button with the text 'Yes to All' on its face
mbHelp	A button with the text 'Help' on its face
mbYesNoCancel (see page 11589)	mbYes, mbNo, and mbCancel
mbYesNoAllCancel	mbYes, mbYesToAll, mbNo, mbNoToAll, and mbCancel
mbOKCancel (see page 11588)	mbOK and mbCancel
mbAbortRetryIgnore (see page 11587)	mbAbort, mbRetry, and mbIgnore
mbAbortIgnore (see page 11586)	mbAbort, mbIgnore

In addition, the Dialogs (see page 11265) unit defines the following constants for commonly used TMsgDlgButton values:

Constant	Meaning
mbYesNoCancel (see page 11589)	mbYes, mbNo, and mbCancel
mbYesNoAllCancel	mbYes, mbYesToAll, mbNo, mbNoToAll, and mbCancel
mbOKCancel (see page 11588)	mbOK and mbCancel

mbAbortRetryIgnore (see page 11587)	mbAbort, mbRetry, and mbIgnore
mbAbortIgnore (see page 11586)	mbAbort, mbIgnore

See Also

MessageDlg (see page 11535)

MessageDlgPos (see page 11537)

TMsgDlgButtons (see page 11575)

61.3.10 Dialogs.TMsgDlgType Enumeration

TMsgDlgType defines values describing the type of message box.

Pascal

```
TMsgDlgType = (
  mtWarning,
  mtError,
  mtInformation,
  mtConfirmation,
  mtCustom
);
```

C++

```
enum TMsgDlgType {
  mtWarning,
  mtError,
  mtInformation,
  mtConfirmation,
  mtCustom
};
```

File

Dialogs

Description

The TMsgDlgType type defines the values describing the type of a message box. The TMsgDlgType is used by the MessageDlg (see page 11535) and MessageDlgPos (see page 11537) functions. The following table lists the possible values:

Value	Meaning
mtWarning	A message box containing a yellow exclamation point symbol.
mtError	A message box containing a red stop sign.
mtInformation	A message box containing a blue "i".
mtConfirmation	A message box containing a green question mark.
mtCustom	A message box containing no bitmap. The caption of the message box is the name of the application's executable file.

61.3.11 Dialogs.TOpenOption Enumeration

TOpenOption and TOpenOptions (see page 11576) determine the behavior of a file selection dialog.

Pascal

```
TOpenOption = (
  ofReadOnly,
  ofOverwritePrompt,
  ofHideReadOnly,
  ofNoChangeDir,
  ofShowHelp,
  ofNoValidate,
  ofAllowMultiSelect,
  ofExtensionDifferent,
  ofPathMustExist,
  ofFileMustExist,
  ofCreatePrompt,
  ofShareAware,
  ofNoReadOnlyReturn,
  ofNoTestFileCreate,
  ofNoNetworkButton,
  ofNoLongNames,
  ofOldStyleDialog,
  ofNoDereferenceLinks,
  ofEnableIncludeNotify,
  ofEnableSizing,
  ofDontAddToRecent,
  ofForceShowHidden
);
```

C++

```
enum TOpenOption {
  ofReadOnly,
  ofOverwritePrompt,
  ofHideReadOnly,
  ofNoChangeDir,
  ofShowHelp,
  ofNoValidate,
  ofAllowMultiSelect,
  ofExtensionDifferent,
  ofPathMustExist,
  ofFileMustExist,
  ofCreatePrompt,
  ofShareAware,
  ofNoReadOnlyReturn,
  ofNoTestFileCreate,
  ofNoNetworkButton,
  ofNoLongNames,
  ofOldStyleDialog,
  ofNoDereferenceLinks,
  ofEnableIncludeNotify,
  ofEnableSizing,
  ofDontAddToRecent,
  ofForceShowHidden
};
```

File

Dialogs

Description

TOpenOption values determine the appearance and behavior of a file selection dialog. TOpenOptions (see page 11576) is a set of TOpenOption values.

The following table lists the possible values:

Value	Meaning
ofReadOnly	Selects the Open As Read Only check box by default when the dialog opens.
ofOverwritePrompt	Generates a warning message if the user tries to select a file name that is already in use, asking whether to overwrite the existing file. (Use with save dialogs.)
ofHideReadOnly	Removes the Open As Read Only check box from the dialog.
ofNoChangeDir	After the user clicks OK, resets the current directory to whatever it was before the file-selection dialog opened.
ofShowHelp	Displays a Help button in the dialog.
ofNoValidate	Disables checking for invalid characters in file names. Allows selection of file names with invalid characters.
ofAllowMultiSelect	Allows users to select more than one file in the dialog.
ofExtensionDifferent	This flag is turned on at runtime whenever the selected filename has an extension that differs from DefaultExt. If you use this flag in an application, remember to reset it.
ofPathMustExist	Generates an error message if the user tries to select a file name with a nonexistent directory path.
ofFileMustExist	Generates an error message if the user tries to select a nonexistent file. (only applies to Open dialogs).
ofCreatePrompt	Generates a warning message if the user tries to select a nonexistent file, asking whether to create a new file with the specified name.
ofShareAware	Ignores sharing errors and allows files to be selected even when sharing violations occur.
ofNoReadOnlyReturn	Generates an error message if the user tries to select a read-only file.
ofNoTestFileCreate	Disables checking for network file protection and inaccessibility of disk drives. Applies only when the user tries to save a file in a create-no-modify shared network directory.
ofNoNetworkButton	Removes the Network button (which opens a Map Network Drive dialog) from the file-selection dialog. Applies only if the ofOldStyleDialog flag is on.
ofNoLongNames	Displays 8.3-character file names only. This option is only valid if Options also includes ofOldStyleDialog.
ofOldStyleDialog	Creates the older style of file-selection dialog.
ofNoDereferenceLinks	Disables dereferencing of Windows shortcuts. If the user selects a shortcut, assigns to FileName the path and file name of the shortcut itself (the .LNK file), rather than the file linked to the shortcut.
ofEnableIncludeNotify	(Windows 2000 and later.) Sends CDN_INCLUDEITEM notification messages to the dialog when the user opens a folder. A notification is sent for each item in the newly opened folder. You can use these messages to control which items appear in the folder's item list.
ofEnableSizing	(Windows 98 and later) Lets the Explorer-style dialog be resized with the mouse or keyboard. By default, the dialog allows this resizing regardless of the value of this option. It is only required if you provide a hook procedure or custom template. (Old style dialogs never permit resizing.)
ofDontAddToRecent	Prevents the file from being added to the list of recently opened files.
ofForceShowHidden	Ensures that hidden files are visible in the dialog.

61.3.12 Dialogs.TOpenOptionEx Enumeration

TOpenOptionEx and TOpenOptionsEx (see page 11578) determine the behavior of a file selection dialog.

Pascal

```
TOpenOptionEx = (
  ofExNoPlacesBar
);
```

C++

```
enum TOpenOptionEx {
  ofExNoPlacesBar
};
```

File

Dialogs

Description

TOpenOptionEx values determine the appearance and behavior of a file selection dialog. TOpenOptionsEx (see page 11578) is a set of TOpenOptionEx values.

61.3.13 Dialogs.TPageMeasureUnits Enumeration

Specifies page setup measure unit types.

Pascal

```
TPageMeasureUnits = (
  pmDefault,
  pmMillimeters,
  pmInches
);
```

C++

```
enum TPageMeasureUnits {
  pmDefault,
  pmMillimeters,
  pmInches
};
```

File

Dialogs

Description

Units determines the measurement units used by the user. TPageMeasureUnits enumerates the possible unit types :

Constant	Meaning
pmDefault	User must specify values in units determined by locale settings.
pmMillimeters	User must specify values in millimeters.
pmInches	User must specify values in inches.

See Also

Units (see page 11454)

61.3.14 Dialogs.TPageSetupDialogOption Enumeration

Pascal

```
TPageSetupDialogOption = (
  psoDefaultMinMargins,
  psoDisableMargins,
  psoDisableOrientation,
  psoDisablePagePainting,
  psoDisablePaper,
  psoDisablePrinter,
  psoMargins,
  psoMinMargins,
  psoShowHelp,
  psoWarning,
  psoNoNetworkButton
);
```

C++

```
enum TPageSetupDialogOption {
  psoDefaultMinMargins,
  psoDisableMargins,
  psoDisableOrientation,
  psoDisablePagePainting,
  psoDisablePaper,
  psoDisablePrinter,
  psoMargins,
  psoMinMargins,
  psoShowHelp,
  psoWarning,
  psoNoNetworkButton
};
```

File

Dialogs

Description

This is record Dialogs.TPageSetupDialogOption.

61.3.15 Dialogs.TPageType Enumeration

TPageType indicates the type of page being configured for printing.

Pascal

```
TPageType = (
  ptEnvelope,
  ptPaper
);
```

C++

```
enum TPageType {
```

```
  ptEnvelope,
  ptPaper
};
```

File

Dialogs

Description

TPageType has one of the following values:

Value	Meaning
ptEnvelope	Page is an envelope.
ptPaper	Page is a sheet of paper.

61.3.16 Dialogs.TPrintDialogOption Enumeration

TPrintDialogOption and TPrintDialogOptions (see page 11580) specify options for a TPrintDialog (see page 11458) component.

Pascal

```
TPrintDialogOption = (
  poPrintToFile,
  poPageNums,
  poSelection,
  poWarning,
  poHelp,
  poDisablePrintToFile
);
```

C++

```
enum TPrintDialogOption {
  poPrintToFile,
  poPageNums,
  poSelection,
  poWarning,
  poHelp,
  poDisablePrintToFile
};
```

File

Dialogs

Description

TPrintDialogOptions (see page 11580) is a set of TPrintDialogOption values. The possible values of TPrintDialogOption are

Value	Meaning
poDisablePrintToFile	Disables (grays) the Print To File Check box. (Applicable only if the poPrintToFile flag is set.)
poHelp	Displays a Help button in the dialog. May not work in some versions of Windows 95.
poPageNums	Enables the Pages radio button, allowing the user to specify a page range.
poPrintToFile	Displays a Print To File check box in the dialog.
poSelection	Enables the Selection radio button, allowing the user to print selected (highlighted) text.
poWarning	Generates a warning message if the user tries to send a job to an uninstalled printer.

61.3.17 Dialogs.TPrinterKind Enumeration

Pascal

```
TPrinterKind = (
  pkDotMatrix,
  pkHPPCL
);
```

C++

```
enum TPrinterKind {
  pkDotMatrix,
  pkHPPCL
};
```

File

Dialogs

Description

This is record Dialogs.TPrinterKind.

61.3.18 Dialogs.TPrintRange Enumeration

TPrintRange indicates how the user specified the set of pages to print.

Pascal

```
TPrintRange = (
  prAllPages,
  prSelection,
  prPageNums
);
```

C++

```
enum TPrintRange {
  prAllPages,
  prSelection,
  prPageNums
};
```

File

Dialogs

Description

The possible values of TPrintRange are

Value	Meaning
prAllPages	The All radio button is selected.
prSelection	The Selection radio button is selected.
prPageNums	The Pages radio button is selected.

61.3.19 Dialogs.TProgressBarState Enumeration

TProgressBarState defines a set of values used by the State property.

Pascal

```
TProgressBarState = (
  pbsNormal,
  pbsError,
  pbsPaused
);
```

C++

```
enum TProgressBarState {
  pbsNormal,
  pbsError,
  pbsPaused
};
```

File

Dialogs

Description

The State property defines the state of a dialog task bar. The TProgressBarState enumeration defines the set of state values :

Value	Meaning
pbsNormal	The dialog is still in process.
pbsError	An error has occured in the dialog.
pbsPaused	The dialog process has been paused.

See Also

State (see page 11527)

61.3.20 Dialogs.TTaskDialogCommonButton Enumeration

Buttons (see page 2289) for Task Dialogs (see page 11265).

Pascal

```
TTaskDialogCommonButton = (
  tcbOk,
  tcbYes,
  tcbNo,
  tcbCancel,
  tcbRetry,
  tcbClose
);
```

C++

```
enum TTaskDialogCommonButton {
  tcbOk,
  tcbYes,
  tcbNo,
```

```
    tcbCancel,
    tcbRetry,
    tcbClose
  };
```

File

Dialogs

Description

The following predefined buttons can appear on Task Dialogs (see page 11265):

Button	Description
tcbOk	OK button.
tcbYes	Yes button.
tcbNo	No button.
tcbCancel	Cancel button.
tcbRetry	Retry button.
tcbClose	Close button.

61.3.21 Dialogs.TTaskDialogFlag Enumeration

Flags for Task Dialogs (see page 11265).

Pascal

```
  TTaskDialogFlag = (
    tfEnableHyperlinks,
    tfUseHiconMain,
    tfUseHiconFooter,
    tfAllowDialogCancellation,
    tfUseCommandLinks,
    tfUseCommandLinksNoIcon,
    tfExpandFooterArea,
    tfExpandedByDefault,
    tfVerificationFlagChecked,
    tfShowProgressBar,
    tfShowMarqueeProgressBar,
    tfCallbackTimer,
    tfPositionRelativeToWindow,
    tfRtlLayout,
    tfNoDefaultRadioButton,
    tfCanBeMinimized
  );
```

C++

```
  enum TTaskDialogFlag {
    tfEnableHyperlinks,
    tfUseHiconMain,
    tfUseHiconFooter,
    tfAllowDialogCancellation,
    tfUseCommandLinks,
    tfUseCommandLinksNoIcon,
    tfExpandFooterArea,
    tfExpandedByDefault,
    tfVerificationFlagChecked,
    tfShowProgressBar,
```

```
    tfShowMarqueeProgressBar,
    tfCallbackTimer,
    tfPositionRelativeToWindow,
    tfRtlLayout,
    tfNoDefaultRadioButton,
    tfCanBeMinimized
  };
```

File

Dialogs

Description

The following flags specify options for Task Dialogs (see page 11265):

Flag	Description
tfEnableHyperlinks	If set, content, footer, and expanded text can include hyperlinks.
tfUseHiconMain	if set, use the custom main icon.
tfUseHiconFooter	If set, use the custom footer icon.
tfAllowDialogCancellation	If set, permits Task Dialog to be closed in the absence of a **Cancel** button.
tfUseCommandLinks	If set, buttons are displayed as command links using a standard dialog glyph.
tfUseCommandLinksNoIcon	If set, buttons are displayed as command links without a glyph.
tfExpandFooterArea	If set, display expanded text in the footer.
tfExpandedByDefault	If set, expanded text is displayed when the Task Dialog opens.
tfVerificationFlagChecked	If set, the verification check box is initially checked.
tfShowProgressBar	If set, display the progress bar.
tfShowMarqueeProgressBar	If set, display marquee progress bar.
tfCallbackTimer	If set, dialogs callback will be called every 200 milliseconds.
tfPositionRelativeToWindow	If set, Task Dialog is centered with respect to parent window.
tfRtlLayout	If set, text reads right-to-left.
tfNoDefaultRadioButton	If set, there is no default radio button.
tfCanBeMinimized	If set, the Task Dialog can be minimized.

61.4 Types

The following table lists types in this documentation.

Types

Name	Description
TColorDialogOptions (see page 11568)	TColorDialogOption (see page 11548) and TColorDialogOptions specify options for a TColorDialog (see page 11270) component.
TCustomColors (see page 11569)	TCustomColors is an array that holds the color values for your custom colors.
TFDApplyEvent (see page 11569)	TFDApplyEvent is the type for event handlers that respond when the user clicks the Apply button in a nonmodal font dialog.
TFileDialogCloseEvent (see page 11570)	Event for user clicking OK.
TFileDialogFolderChangingEvent (see page 11570)	Event before a directory is opened or closed from the dialog.
TFileDialogOptions (see page 11571)	The settings in TFileDialogOptions determine the behavior of a TCustomFileDialog (see page 11288) file dialog. TFileDialogOptions is a set of TFileDialogOptions. These options are the same as the options for the Microsoft IFileDialog::GetOptions and IFileDialog::SetOptions functions.
TFileDialogOverwriteEvent (see page 11572)	Event just before a file would be overwritten.
TFileDialogShareViolationEvent (see page 11572)	Event when sharing violations arise from open or save operations.
TFindOptions (see page 11573)	TFindOption (see page 11553) and TFindOptions determine the controls that appear in a find dialog.
TFindReplaceFunc (see page 11574)	This is type Dialogs.TFindReplaceFunc.
TFontDialogOptions (see page 11574)	TFontDialogOption (see page 11555) and TFontDialogOptions are flags that influence the appearance and behavior of a font dialog.
TIncludeItemEvent (see page 11575)	This is type Dialogs.TIncludeItemEvent.
TMsgDlgButtons (see page 11575)	TMsgDlgButtons defines a set of values used by MessageDlg (see page 11535) and MessageDlgPos (see page 11537).
TOFNotifyEx (see page 11576)	This is type Dialogs.TOFNotifyEx.
TOpenOptions (see page 11576)	TOpenOption (see page 11559) and TOpenOptions determine the behavior of a file selection dialog.
TOpenOptionsEx (see page 11578)	TOpenOptionEx (see page 11561) and TOpenOptionsEx determine the behavior of a file selection dialog.
TPageSetupBeforePaintEvent (see page 11578)	TPageSetupBeforePaintEvent defines the class method called by the BeforePaint event.
TPageSetupDialogOptions (see page 11579)	Specifies page setup dialog options.
TPaintPageEvent (see page 11579)	TPageSetupBeforePaintEvent (see page 11578) defines the class method called by various TPageSetupDialog (see page 11441) events.
TPrintDialogOptions (see page 11580)	TPrintDialogOption (see page 11563) and TPrintDialogOptions specify options for a TPrintDialog (see page 11458) component.
TPrinterOrientation (see page 11581)	TPrinterOrientation indicates the orientation of pages when printing.
TTaskDialogButtonList (see page 11581)	This is type Dialogs.TTaskDialogButtonList.
TTaskDialogCommonButtons (see page 11582)	An array of Buttons (see page 2289) for Task Dialogs (see page 11265).
TTaskDialogFlags (see page 11582)	Flags for Task Dialogs (see page 11265).
TTaskDialogIcon (see page 11583)	This is type Dialogs.TTaskDialogIcon.
TTaskDlgClickEvent (see page 11583)	Event for a button click in a Task Dialog.
TTaskDlgTimerEvent (see page 11584)	Event for the Task Dialog timer.

61.4.1 Dialogs.TColorDialogOptions Type

TColorDialogOption (see page 11548) and TColorDialogOptions specify options for a TColorDialog (see page 11270) component.

Pascal

```
TColorDialogOptions = set of TColorDialogOption;
```

C++

```
TColorDialogOption TColorDialogOptions;
```

File

Dialogs

Description

TColorDialogOptions is a set of TColorDialogOption (see page 11548) values. The possible values of TColorDialogOption (see page 11548) are

Value	Meaning
cdFullOpen	Display custom color options when the dialog opens.
cdPreventFullOpen	Disable the Define Custom Colors button in the dialog, so that the user cannot define new colors.
cdShowHelp	Add a Help button to the color dialog.
cdSolidColor	Direct Windows to use the nearest solid color to the color chosen.
cdAnyColor	Allow the user to select non-solid colors (which may have to be approximated by dithering).

61.4.2 Dialogs.TCustomColors Type

TCustomColors is an array that holds the color values for your custom colors.

Pascal

```
TCustomColors = array[0..MaxCustomColors - 1] of Longint;
```

C++

```
array[0..MaxCustomColors - 1] of Longint TCustomColors;
```

File

Dialogs

Description

TCustomColors is an array that holds the color values for the custom colors you create using TColorDialog (see page 11270). As many as 16 custom colors (ColorA through ColorP) can be set.

61.4.3 Dialogs.TFDApplyEvent Type

TFDApplyEvent is the type for event handlers that respond when the user clicks the Apply button in a nonmodal font dialog.

Pascal

```
TFDApplyEvent = procedure (Sender: TObject; Wnd: HWND) of object;
```

C++

```
(Sender: TObject; Wnd: HWND) ( TFDApplyEvent)();
```

File

Dialogs

Description

Sender is the TFontDialog (see page 11394) component that displays the dialog.

Wnd is the Font dialog's window handle. Wnd has the same value as the Handle property of Sender.

61.4.4 Dialogs.TFileDialogCloseEvent Type

Event for user clicking OK.

Pascal

```
TFileDialogCloseEvent = procedure (Sender: TObject; var CanClose: Boolean) of object;
```

C++

```
(Sender: TObject; var CanClose: Boolean) ( TFileDialogCloseEvent)();
```

File

Dialogs

Description

TFileDialogCloseEvent is the event for the user clicking OK. If defined, the OnFileOkClick property contains a pointer to this event.

See Also

TCustomFileDialog (see page 11288)

OnFileOkClick (see page 11306)

61.4.5 Dialogs.TFileDialogFolderChangingEvent Type

Event before a directory is opened or closed from the dialog.

Pascal

```
TFileDialogFolderChangingEvent = procedure (Sender: TObject; var CanChange: Boolean) of object;
```

C++

```
(Sender: TObject; var CanChange: Boolean) ( TFileDialogFolderChangingEvent)();
```

File

Dialogs

Description

TFileDialogFolderChangingEvent is the event that occurs just before a directory is opened or closed from the dialog. If defined, the OnFolderChanging property contains a pointer to this event.

See Also

TCustomFileDialog (see page 11288)

OnFolderChanging (see page 11307)

61.4.6 Dialogs.TFileDialogOptions Type

The settings in TFileDialogOptions determine the behavior of a TCustomFileDialog (see page 11288) file dialog. TFileDialogOptions is a set of TFileDialogOptions. These options are the same as the options for the Microsoft IFileDialog::GetOptions and IFileDialog::SetOptions functions.

Pascal

```
TFileDialogOptions = set of TFileDialogOption;
```

C++

```
TFileDialogOption TFileDialogOptions;
```

File

Dialogs

Description

The following table lists the possible values:

Value	Meaning
fdoOverWritePrompt	Prompt before overwriting an existing file of the same name when saving a file. This is a default for save dialogs.
fdoStrictFileTypes	The file extension of a saved file being must match the selected file type.
fdoNoChangeDir	Unused.
fdoPickFolders	Choose folders rather than files.
fdoForceFileSystem	Returned items must be file system items.
fdoAllNonStorageItems	Allow users to choose any item in the Shell namespace. This flag cannot be combined with fdoForceFileSystem.
fdoNoValidate	Do not check for situations preventing applications from opening selected files, such as sharing violations or access denied errors.
fdoAllowMultiSelect	Allow selecting multiple items in an open dialog.
fdoPathMustExist	Items returned must be in an existing folder. This is a default.
fdoFileMustExist	Items returned must exist. This is a default value for open dialogs.
fdoCreatePrompt	Prompt for creation if returned item in save dialog does not exist. This does not create the item.
fdoShareAware	For a sharing violation opening a file, call the application back for guidance. This flag is overridden by fdoNoValidate.
fdoNoReadOnlyReturn	Do not return read-only items.
fdoNoTestFileCreate	Do not test creation of returned item from save dialogs. If not set, the calling application must handle errors discovered in the creation test.
fdoHideMRUPlaces	Hide places of recently opened or saved items.
fdoHidePinnedPlaces	Hide pinned places from which users can choose.
fdoNoDereferenceLinks	Shortcuts are not treated as their target items, allowing applications to open .lnk files.
fdoDontAddToRecent	Do not add the item being opened or saved to the list of recent places.
fdoForceShowHidden	Show hidden items.

fdoDefaultNoMiniMode	Open save dialog box in expanded mode in which users can browse folders. Expanded mode is set and unset by clicking the button in the lower-left corner of a save dialog box.
fdoForcePreviewPaneOn	Display the preview pane.

61.4.7 Dialogs.TFileDialogOverwriteEvent Type

Event just before a file would be overwritten.

Pascal

```
TFileDialogOverwriteEvent = procedure (Sender: TObject; var Response:
TFileDialogOverwriteResponse) of object;
```

C++

```
(Sender: TObject; var Response: TFileDialogOverwriteResponse) ( TFileDialogOverwriteEvent)();
```

File

Dialogs

Description

TFileDialogOverwriteEvent is the event just before a file would be overwritten from a save operation. If defined, the OnOverwrite property contains a pointer to this event.

See Also

TCustomFileDialog (see page 11288)

OnOverwrite (see page 11307)

61.4.8 Dialogs.TFileDialogShareViolationEvent Type

Event when sharing violations arise from open or save operations.

Pascal

```
TFileDialogShareViolationEvent = procedure (Sender: TObject; var Response:
TFileDialogShareViolationResponse) of object;
```

C++

```
(Sender: TObject; var Response: TFileDialogShareViolationResponse) (
TFileDialogShareViolationEvent)();
```

File

Dialogs

Description

TFileDialogShareViolationEvent is the event for sharing violations that arise from open or save operations. If defined, the OnShareViolation property contains a pointer to this event.

See Also

TCustomFileDialog (see page 11288)

OnShareViolation (see page 11308)

61.4.9 Dialogs.TFindOptions Type

TFindOption (see page 11553) and TFindOptions determine the controls that appear in a find dialog.

Pascal

```
TFindOptions = set of TFindOption;
```

C++

```
TFindOption TFindOptions;
```

File

Dialogs

Description

TFindOption (see page 11553) values determine which controls are enabled in a TFindDialog (see page 11376) instance, and how the controls are initialized.

TFindOptions is a set of TFindOption (see page 11553) values.

The following table lists the possible values:

Value	Meaning
frDisableMatchCase	Disables (grays) the Match Case check box in a find dialog.
frDisableUpDown	Disables (grays) the Up and Down buttons, which determine the direction of the search.
frDisableWholeWord	Disables (grays) the Match Whole Word check box of find dialog.
frDown	Selects the Down button by default when the dialog opens. If the frDown flags is off, Up is selected when the dialog opens. (By default, frDown is on.)
frFindNext	This flag is turned on when the user clicks the Find Next button and turned off when the dialog closes.
frHideMatchCase	Removes the Match Case check box from the dialog.
frHideWholeWord	Removes the Match Whole Word check box from the dialog.
frHideUpDown	Removes the Up and Down buttons from the dialog.
frMatchCase	This flag is turned on (off) when the user selects (deselects) the Match Case check box. To select the check box by default when the dialog opens, set frMatchCase at design time.
frReplace	Applies to TReplaceDialog (see page 11482) only. This flag is set by the system to indicate that the application should replace the current occurrence (and only the current occurrence) of the FindText string with the ReplaceText string. Not used in search routines.
frReplaceAll	Applies to TReplaceDialog (see page 11482) only. This flag is set by the system to indicate that the application should replace all occurrences of the FindText string with the ReplaceText string.
frShowHelp	Displays a Help button in the dialog.
frWholeWord	This flag is turned on (off) when the user selects (deselects) the Match Whole Word check box. To select the check box by default when the dialog opens, set frWholeWord at design time.

61.4.10 Dialogs.TFindReplaceFunc Type

Pascal

```
TFindReplaceFunc = function (var FindReplace: TFindReplace): HWnd stdcall;
```

C++

```
(var FindReplace: TFindReplace): HWnd stdcall ( TFindReplaceFunc)();
```

File

Dialogs

Description

This is type Dialogs.TFindReplaceFunc.

61.4.11 Dialogs.TFontDialogOptions Type

TFontDialogOption (see page 11555) and TFontDialogOptions are flags that influence the appearance and behavior of a font dialog.

Pascal

```
TFontDialogOptions = set of TFontDialogOption;
```

C++

```
TFontDialogOption TFontDialogOptions;
```

File

Dialogs

Description

TFontDialogOptions is a set of TFontDialogOption (see page 11555) values. These are the possible values of TFontDialogOption (see page 11555):

Value	Meaning
fdAnsiOnly	Displays only fonts that use the Windows character set. Symbol fonts are not listed in the dialog.
fdApplyButton	Displays an Apply button in the dialog, whether or not there is an OnApply event handler.
fdEffects	Displays the Effects check boxes (Strikeout and Underline) and the Color list box in the dialog.
fdFixedPitchOnly	Displays only monospaced fonts in the dialog. Proportionally spaced fonts are not listed, nor are TrueType fonts that do not have the fixed-pitch flag set in the style header.
fdForceFontExist	Allows the user to enter only fonts that are displayed in the dialog (that is, listed in the Font combo box). If the user tries to enter another font name, an error message appears.
fdLimitSize	Enables the MaxFontSize and MinFontSize properties, limiting the range of font sizes that appear in the dialog if these properties have values.
fdNoFaceSel	Causes the dialog to open without a preselected font name in the Font combo box.
fdNoOEMFonts	Removes OEM fonts from the dialog's combo box. Lists only non-OEM fonts.
fdScalableOnly	Displays only scalable fonts in the dialog. Non-scalable (bitmap) fonts are removed from the list.

fdNoSimulations	Displays only fonts and font styles that are directly supported by the font definition file. GDI-synthesized bold and italic styles (for bitmap fonts) are not listed.
fdNoSizeSel	Causes the dialog to open without a preselected size in the Size combo box.
fdNoStyleSel	Causes the dialog to open without a preselected style in the Font Style combo box.
fdNoVectorFonts	Removes vector fonts from the dialog's combo box; lists only non-vector fonts. (Vector fonts are Windows 1.0 fonts, such as Roman or Script, which resemble pen-plotter output.)
fdShowHelp	Displays a Help button in the dialog.
fdTrueTypeOnly	Displays only TrueType fonts in the dialog. Other fonts are not listed.
fdWysiwyg	Displays only fonts that are available to both the printer and the screen. Device-specific fonts are not listed in the dialog.

61.4.12 Dialogs.TIncludeItemEvent Type

Pascal

```
TIncludeItemEvent = procedure (const OFN: TOFNotifyEx; var Include: Boolean) of object;
```

C++

```
(const OFN: TOFNotifyEx; var Include: Boolean) ( TIncludeItemEvent)();
```

File

Dialogs

Description

This is type Dialogs.TIncludeItemEvent.

61.4.13 Dialogs.TMsgDlgButtons Type

TMsgDlgButtons defines a set of values used by MessageDlg (see page 11535) and MessageDlgPos (see page 11537).

Pascal

```
TMsgDlgButtons = set of TMsgDlgBtn;
```

C++

```
TMsgDlgBtn TMsgDlgButtons;
```

File

Dialogs

Description

The TMsgDlgButtons type defines the set of values a button in a message box can have. The TMsgDlgButtons type is used by the MessageDlg (see page 11535) and MessageDlgPos (see page 11537) functions. The following tables lists the possible values:

Value	Meaning
mbYes	A button with 'Yes' on its face.

mbNo	A button the text 'No' on its face.
mbOK	A button the text 'OK' on its face.
mbCancel	A button with the text 'Cancel' on its face.
mbAbort	A button with the text 'Abort' on its face
mbRetry	A button with the text 'Retry' on its face
mbIgnore	A button the text 'Ignore' on its face
mbAll	A button with the text 'All' on its face
mbNoToAll	A button with the text 'No to All' on its face
mbYesToAll	A button with the text 'Yes to All' on its face
mbHelp	A button with the text 'Help' on its face

In addition, the Dialogs (see page 11265) unit defines the following constants for commonly used TMsgDlgButtons values:

Constant	Meaning
mbYesNoCancel (see page 11589)	mbYes, mbNo, and mbCancel
mbYesNoAllCancel	mbYes, mbYesToAll, mbNo, mbNoToAll, and mbCancel
mbOKCancel (see page 11588)	mbOK and mbCancel
mbAbortRetryIgnore (see page 11587)	mbAbort, mbRetry, and mbIgnore
mbAbortIgnore (see page 11586)	mbAbort, mbIgnore

61.4.14 Dialogs.TOFNotifyEx Type

Pascal

```
TOFNotifyEx = type CommDlg.TOFNotifyEx;
```

C++

```
type CommDlg.TOFNotifyEx TOFNotifyEx;
```

File

Dialogs

Description

This is type Dialogs.TOFNotifyEx.

61.4.15 Dialogs.TOpenOptions Type

TOpenOption (see page 11559) and TOpenOptions determine the behavior of a file selection dialog.

Pascal

```
TOpenOptions = set of TOpenOption;
```

C++

```
TOpenOption TOpenOptions;
```

File

Dialogs

Description

TOpenOption (see page 11559) values determine the appearance and behavior of a file selection dialog. TOpenOptions is a set of TOpenOption (see page 11559) values.

The following table lists the possible values:

Value	Meaning
ofReadOnly	Selects the Open As Read Only check box by default when the dialog opens.
ofOverwritePrompt	Generates a warning message if the user tries to select a file name that is already in use, asking whether to overwrite the existing file. (Use with save dialogs.)
ofHideReadOnly	Removes the Open As Read Only check box from the dialog.
ofNoChangeDir	After the user clicks OK, resets the current directory to whatever it was before the file-selection dialog opened.
ofShowHelp	Displays a Help button in the dialog.
ofNoValidate	Disables checking for invalid characters in file names. Allows selection of file names with invalid characters.
ofAllowMultiSelect	Allows users to select more than one file in the dialog.
ofExtensionDifferent	This flag is turned on at runtime whenever the selected filename has an extension that differs from DefaultExt. If you use this flag in an application, remember to reset it.
ofPathMustExist	Generates an error message if the user tries to select a file name with a nonexistent directory path.
ofFileMustExist	Generates an error message if the user tries to select a nonexistent file. (only applies to Open dialogs).
ofCreatePrompt	Generates a warning message if the user tries to select a nonexistent file, asking whether to create a new file with the specified name.
ofShareAware	Ignores sharing errors and allows files to be selected even when sharing violations occur.
ofNoReadOnlyReturn	Generates an error message if the user tries to select a read-only file.
ofNoTestFileCreate	Disables checking for network file protection and inaccessibility of disk drives. Applies only when the user tries to save a file in a create-no-modify shared network directory.
ofNoNetworkButton	Removes the Network button (which opens a Map Network Drive dialog) from the file-selection dialog. Applies only if the ofOldStyleDialog flag is on.
ofNoLongNames	Displays 8.3-character file names only. This option is only valid if Options also includes ofOldStyleDialog.
ofOldStyleDialog	Creates the older style of file-selection dialog.
ofNoDereferenceLinks	Disables dereferencing of Windows shortcuts. If the user selects a shortcut, assigns to FileName the path and file name of the shortcut itself (the .LNK file), rather than the file linked to the shortcut.
ofEnableIncludeNotify	(Windows 2000 and later.) Sends CDN_INCLUDEITEM notification messages to the dialog when the user opens a folder. A notification is sent for each item in the newly opened folder. You can use these messages to control which items appear in the folder's item list.
ofEnableSizing	(Windows 98 and later) Lets the Explorer-style dialog be resized with the mouse or keyboard. By default, the dialog allows this resizing regardless of the value of this option. It is only required if you provide a hook procedure or custom template. (Old style dialogs never permit resizing.)
ofDontAddToRecent	Prevents the file from being added to the list of recently opened files.

ofForceShowHidden	Ensures that hidden files are visible in the dialog.

61.4.16 Dialogs.TOpenOptionsEx Type

TOpenOptionEx (see page 11561) and TOpenOptionsEx determine the behavior of a file selection dialog.

Pascal

```
TOpenOptionsEx = set of TOpenOptionEx;
```

C++

```
TOpenOptionEx TOpenOptionsEx;
```

File

Dialogs

Description

TOpenOptionEx (see page 11561) values determine the appearance and behavior of a file selection dialog. TOpenOptionsEx is a set of TOpenOptionEx (see page 11561) values.

61.4.17 Dialogs.TPageSetupBeforePaintEvent Type

TPageSetupBeforePaintEvent defines the class method called by the BeforePaint event.

Pascal

```
TPageSetupBeforePaintEvent = procedure (Sender: TObject; const PaperSize: SmallInt; const
Orientation: TPrinterOrientation; const PageType: TPageType; var DoneDrawing: Boolean) of
object;
```

C++

```
(Sender: TObject; const PaperSize: SmallInt; const Orientation: TPrinterOrientation; const
PageType: TPageType; var DoneDrawing: Boolean) ( TPageSetupBeforePaintEvent)();
```

File

Dialogs

Description

The BeforePaint event occurs before dialog draws sample page. Write a BeforePaint event handler to initialize data used by the other event handlers.

PaperSize indicates a standard Windows paper size. See Paper size constants.

Orientation indicates the paper orientation currently selected.

PageType indicates the kind of paper currently selected.

Set **DoneDrawing** to true to disable further painting. This results in a blank sample page, with none of the other event handlers being called.

See Also

BeforePaint (see page 11455)

TPageSetupDialog (see page 11441)

61.4.18 Dialogs.TPageSetupDialogOptions Type

Specifies page setup dialog options.

Pascal

```
TPageSetupDialogOptions = set of TPageSetupDialogOption;
```

C++

```
TPageSetupDialogOption TPageSetupDialogOptions;
```

File

Dialogs

Description

Options specifies the options for the page setup dialog. TPageSetupDialogOptions enumerates the possible unit types :

Option	Meaning
psoDefaultMinMargins	Sets the minimum values that the user can specify for the page margins to be the minimum margins allowed by the printer.
psoDisableMargins	Disables the margin controls, preventing the user from setting the margins.
psoDisableOrientation	Disables the orientation controls, preventing the user from setting the page orientation.
psoDisablePagePainting	Prevents default sample page drawing. Does not disable event calls.
psoDisablePaper	Disables the paper controls, preventing the user from setting page parameters such as the paper size and source.
psoDisablePrinter	Disables the Printer button, preventing the user from invoking a dialog box that contains additional printer setup information.
psoMargins	Sets the initial margins from MarginLeft, MarginRight, MarginBottom, and MarginTop properties. Otherwise, initial margins are one inch each.
psoMinMargins	Sets the minimum margins from MinMarginLeft, MinMarginRight, MinMarginBottom, and MinMarginTop. Otherwise, minimum margins are determined by the printer driver.
psoShowHelp	Displays Help button and Help title bar button.
psoWarning	Prevents display of warning when there is no default printer.
psoNoNetworkButton	Hides and disables Network button.

See Also

Options (see page 11452)

61.4.19 Dialogs.TPaintPageEvent Type

TPageSetupBeforePaintEvent (see page 11578) defines the class method called by various TPageSetupDialog (see page 11441) events.

Pascal

```
TPaintPageEvent = procedure (Sender: TObject; Canvas: TCanvas; PageRect: TRect; var
DoneDrawing: Boolean) of object;
```

C++

```
(Sender: TObject; Canvas: TCanvas; PageRect: TRect; var DoneDrawing: Boolean) (
TPaintPageEvent)();
```

File

Dialogs

Description

The TPaintPageEvent class method is called by a number of events that are involved in the painting of the Page Setup Dialog illustrative page image.

Sender is a reference to the calling TPageSetupDialog (see page 11441) instance.

Canvas is a reference to the Page Setup Dialog canvas.

PageRect gives the bounds of the item being drawn that resulted in the event.

Set **DoneDrawing** to true to disable further painting. This results in a blank sample page, with none of the other event handlers being called.

See Also

OnDrawFullPage (see page 11456)

OnDrawMinMargin (see page 11457)

OnDrawMargin (see page 11457)

OnDrawGreekText (see page 11456)

OnDrawEnvStamp (see page 11455)

OnDrawRetAddress (see page 11458)

TPageSetupDialog (see page 11441)

61.4.20 Dialogs.TPrintDialogOptions Type

TPrintDialogOption (see page 11563) and TPrintDialogOptions specify options for a TPrintDialog (see page 11458) component.

Pascal

```
TPrintDialogOptions = set of TPrintDialogOption;
```

C++

```
TPrintDialogOption TPrintDialogOptions;
```

File

Dialogs

Description

TPrintDialogOptions is a set of TPrintDialogOption (see page 11563) values. The possible values of TPrintDialogOption (see page 11563) are

Value	Meaning
poDisablePrintToFile	Disables (grays) the Print To File Check box. (Applicable only if the poPrintToFile flag is set.)
poHelp	Displays a Help button in the dialog. May not work in some versions of Windows 95.
poPageNums	Enables the Pages radio button, allowing the user to specify a page range.
poPrintToFile	Displays a Print To File check box in the dialog.
poSelection	Enables the Selection radio button, allowing the user to print selected (highlighted) text.
poWarning	Generates a warning message if the user tries to send a job to an uninstalled printer.

61.4.21 Dialogs.TPrinterOrientation Type

TPrinterOrientation indicates the orientation of pages when printing.

Pascal

```
TPrinterOrientation = Printers.TPrinterOrientation;
```

C++

```
Printers.TPrinterOrientation TPrinterOrientation;
```

File

Dialogs

Description

These are the possible values for TPrinterOrientation:

Value	Meaning
poPortrait	The job prints vertically on the page.
poLandscape	The job prints horizontally on the page.

61.4.22 Dialogs.TTaskDialogButtonList Type

Pascal

```
TTaskDialogButtonList = array of TTaskDialogButton;
```

C++

```
array of TTaskDialogButton TTaskDialogButtonList;
```

File

Dialogs

Description

This is type Dialogs.TTaskDialogButtonList.

61.4.23 Dialogs.TTaskDialogCommonButtons Type

An array of Buttons (see page 2289) for Task Dialogs (see page 11265).

Pascal

```
TTaskDialogCommonButtons = set of TTaskDialogCommonButton;
```

C++

```
TTaskDialogCommonButton TTaskDialogCommonButtons;
```

File

Dialogs

Description

The following predefined buttons can appear on Task Dialogs (see page 11265):

Button	Description
tcbOk	OK button.
tcbYes	Yes button.
tcbNo	No button.
tcbCancel	Cancel button.
tcbRetry	Retry button.
tcbClose	Close button.

61.4.24 Dialogs.TTaskDialogFlags Type

Flags for Task Dialogs (see page 11265).

Pascal

```
TTaskDialogFlags = set of TTaskDialogFlag;
```

C++

```
TTaskDialogFlag TTaskDialogFlags;
```

File

Dialogs

Description

The following flags specify options for Task Dialogs (see page 11265):

Flag	Description
tfEnableHyperlinks	If set, content, footer, and expanded text can include hyperlinks.
tfUseHiconMain	if set, use the custom main icon.
tfUseHiconFooter	If set, use the custom footer icon.

tfAllowDialogCancellation	If set, permits Task Dialog to be closed in the absence of a **Cancel** button.
tfUseCommandLinks	If set, buttons are displayed as command links using a standard dialog glyph.
tfUseCommandLinksNoIcon	If set, buttons are displayed as command links without a glyph.
tfExpandFooterArea	If set, display expanded text in the footer.
tfExpandedByDefault	If set, expanded text is displayed when the Task Dialog opens.
tfVerificationFlagChecked	If set, the verification check box is initially checked.
tfShowProgressBar	If set, display the progress bar.
tfShowMarqueeProgressBar	If set, display marquee progress bar.
tfCallbackTimer	If set, dialogs callback will be called every 200 milliseconds.
tfPositionRelativeToWindow	If set, Task Dialog is centered with respect to parent window.
tfRtlLayout	If set, text reads right-to-left.
tfNoDefaultRadioButton	If set, there is no default radio button.
tfCanBeMinimized	If set, the Task Dialog can be minimized.

61.4.25 Dialogs.TTaskDialogIcon Type

Pascal

```
TTaskDialogIcon = Low(Integer)..High(Integer);
```

C++

```
Low(Integer)..High(Integer) TTaskDialogIcon;
```

File

Dialogs

Description

This is type Dialogs.TTaskDialogIcon.

61.4.26 Dialogs.TTaskDlgClickEvent Type

Event for a button click in a Task Dialog.

Pascal

```
TTaskDlgClickEvent = procedure (Sender: TObject; ModalResult: TModalResult; var CanClose:
Boolean) of object;
```

C++

```
(Sender: TObject; ModalResult: TModalResult; var CanClose: Boolean) ( TTaskDlgClickEvent)();
```

File

Dialogs

Description

OnButtonClicked is the event for a button click in a Task Dialog.

61.4.27 Dialogs.TTaskDlgTimerEvent Type

Event for the Task Dialog timer.

Pascal

```
TTaskDlgTimerEvent = procedure (Sender: TObject; TickCount: Cardinal; var Reset: Boolean) of
object;
```

C++

```
(Sender: TObject; TickCount: Cardinal; var Reset: Boolean) ( TTaskDlgTimerEvent)();
```

File

Dialogs

Description

OnTimer is the timer event for a Task Dialog.

61.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
ForceCurrentDirectory (see page 11585)	Indicates whether open and save dialogs should display the current directory if no initial directory is assigned.
UseLatestCommonDialogs (see page 11585)	This is variable Dialogs.UseLatestCommonDialogs.

61.5.1 Dialogs.ForceCurrentDirectory Variable

Indicates whether open and save dialogs should display the current directory if no initial directory is assigned.

Pascal

```
ForceCurrentDirectory: Boolean = False;
```

C++

```
Boolean ForceCurrentDirectory = False;
```

File

Dialogs

Description

ForceCurrentDirectory indicates how open and save dialogs should initialize the directory if the InitialDir property is not set to a valid directory. When ForceCurrentDirectory is **True**, the dialog opens with the current directory assigned. When ForceCurrentDirectory is **False**, the initial directory is either the current directory or the My Documents directory, depending on the version of Windows.

See Also

TOpenDialog (see page 11407)

61.5.2 Dialogs.UseLatestCommonDialogs Variable

Pascal

```
UseLatestCommonDialogs: Boolean = False;
```

C++

```
Boolean UseLatestCommonDialogs = False;
```

File

Dialogs

Description

This is variable Dialogs.UseLatestCommonDialogs.

61.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
MaxCustomColors (see page 11586)	The number of custom colors on a Color Dialog.
mbAbortIgnore (see page 11586)	Used to define a dialog with Abort and Ignore buttons.
mbAbortRetryIgnore (see page 11587)	Used to define a dialog with Abort, Retry and Ignore buttons.
mbOKCancel (see page 11588)	Used to define a dialog with OK and Cancel buttons.
mbYesAllNoAllCancel (see page 11588)	This is constant Dialogs.mbYesAllNoAllCancel.
mbYesNo (see page 11589)	Used to define a dialog with Yes and No buttons.
mbYesNoCancel (see page 11589)	Used to define a dialog with Yes, No and Cancel buttons.
tdiError (see page 11590)	This is constant Dialogs.tdiError.
tdiInformation (see page 11590)	This is constant Dialogs.tdiInformation.
tdiNone (see page 11591)	This is constant Dialogs.tdiNone.
tdiShield (see page 11591)	This is constant Dialogs.tdiShield.
tdiWarning (see page 11591)	This is constant Dialogs.tdiWarning.

61.6.1 Dialogs.MaxCustomColors Constant

The number of custom colors on a Color Dialog.

Pascal

```
MaxCustomColors = 16;
```

C++

```
MaxCustomColors = 16;
```

File

Dialogs

Description

The CustomColors property is used to define colors to be displayed in some or all of the 16 custom color boxes on a TColorDialog (see page 11270). This limit of 16 is defined by MaxCustomColors.

See Also

CustomColors (see page 11276)

TColorDialog (see page 11270)

61.6.2 Dialogs.mbAbortIgnore Constant

Used to define a dialog with Abort and Ignore buttons.

Pascal

```
mbAbortIgnore = [mbAbort, mbIgnore];
```

C++

```
mbAbortIgnore = [mbAbort, mbIgnore];
```

File

Dialogs

Description

The TMsgDlgButtons (see page 11575) type defines the set of values a button in a message box can have. mbAbortIgnore value defines a dialog with Abort and Ignore buttons.

See Also

mbAbortRetryIgnore (see page 11587)

mbOKCancel (see page 11588)

mbYesNoAllCancel

mbYesNo (see page 11589)

mbYesNoCancel (see page 11589)

MessageDlg (see page 11535)

MessageDlgPos (see page 11537)

MessageDlgPosHelp (see page 11539)

TMsgDlgButtons (see page 11575)

61.6.3 Dialogs.mbAbortRetryIgnore Constant

Used to define a dialog with Abort, Retry and Ignore buttons.

Pascal

```
mbAbortRetryIgnore = [mbAbort, mbRetry, mbIgnore];
```

C++

```
mbAbortRetryIgnore = [mbAbort, mbRetry, mbIgnore];
```

File

Dialogs

Description

The TMsgDlgButtons (see page 11575) type defines the set of values a button in a message box can have. mbAbortRetryIgnore value defines a dialog with Abort, Retry and Ignore buttons.

See Also

mbAbortIgnore (see page 11586)

mbOKCancel (see page 11588)

mbYesNoAllCancel

mbYesNo (see page 11589)

mbYesNoCancel (see page 11589)

MessageDlg (see page 11535)

MessageDlgPos (see page 11537)

MessageDlgPosHelp (see page 11539)

TMsgDlgButtons (see page 11575)

61.6.4 Dialogs.mbOKCancel Constant

Used to define a dialog with OK and Cancel buttons.

Pascal

```
mbOKCancel = [mbOK, mbCancel];
```

C++

```
mbOKCancel = [mbOK, mbCancel];
```

File

Dialogs

Description

The TMsgDlgButtons (see page 11575) type defines the set of values a button in a message box can have. mbOKCancel value defines a dialog with OK and Cancel buttons.

See Also

mbAbortIgnore (see page 11586)

mbAbortRetryIgnore (see page 11587)

mbYesNoAllCancel

mbYesNo (see page 11589)

mbYesNoCancel (see page 11589)

MessageDlg (see page 11535)

MessageDlgPos (see page 11537)

MessageDlgPosHelp (see page 11539)

TMsgDlgButtons (see page 11575)

61.6.5 Dialogs.mbYesAllNoAllCancel Constant

Pascal

```
mbYesAllNoAllCancel = [mbYes, mbYesToAll, mbNo, mbNoToAll, mbCancel];
```

C++

```
mbYesAllNoAllCancel = [mbYes, mbYesToAll, mbNo, mbNoToAll, mbCancel];
```

File

Dialogs

Description

This is constant Dialogs.mbYesAllNoAllCancel.

61.6.6 Dialogs.mbYesNo Constant

Used to define a dialog with Yes and No buttons.

Pascal

```
mbYesNo = [mbYes, mbNo];
```

C++

```
mbYesNo = [mbYes, mbNo];
```

File

Dialogs

Description

The TMsgDlgButtons (see page 11575) type defines the set of values a button in a message box can have. mbYesNo value defines a dialog with Yes and No buttons.

See Also

mbAbortIgnore (see page 11586)

mbAbortRetryIgnore (see page 11587)

mbOKCancel (see page 11588)

mbYesNoAllCancel

mbYesNoCancel (see page 11589)

MessageDlg (see page 11535)

MessageDlgPos (see page 11537)

MessageDlgPosHelp (see page 11539)

TMsgDlgButtons (see page 11575)

61.6.7 Dialogs.mbYesNoCancel Constant

Used to define a dialog with Yes, No and Cancel buttons.

Pascal

```
mbYesNoCancel = [mbYes, mbNo, mbCancel];
```

C++

```
mbYesNoCancel = [mbYes, mbNo, mbCancel];
```

File

Dialogs

Description

The TMsgDlgButtons (see page 11575) type defines the set of values a button in a message box can have. mbYesNoCancel value defines a dialog with Yes, No and Cancel buttons.

See Also

mbAbortIgnore (see page 11586)

mbAbortRetryIgnore (see page 11587)

mbOKCancel (see page 11588)

mbYesNo (see page 11589)

mbYesNoAllCancel

MessageDlg (see page 11535)

MessageDlgPos (see page 11537)

MessageDlgPosHelp (see page 11539)

TMsgDlgButtons (see page 11575)

61.6.8 Dialogs.tdiError Constant

Pascal

```
tdiError = 2;
```

C++

```
tdiError = 2;
```

File

Dialogs

Description

This is constant Dialogs.tdiError.

61.6.9 Dialogs.tdiInformation Constant

Pascal

```
tdiInformation = 3;
```

C++

```
tdiInformation = 3;
```

File

Dialogs

Description

This is constant Dialogs.tdiInformation.

61.6.10 Dialogs.tdiNone Constant

Pascal

```
tdiNone = 0;
```

C++

```
tdiNone = 0;
```

File

Dialogs

Description

This is constant Dialogs.tdiNone.

61.6.11 Dialogs.tdiShield Constant

Pascal

```
tdiShield = 4;
```

C++

```
tdiShield = 4;
```

File

Dialogs

Description

This is constant Dialogs.tdiShield.

61.6.12 Dialogs.tdiWarning Constant

Pascal

```
tdiWarning = 1;
```

C++

```
tdiWarning = 1;
```

File

Dialogs

Description

This is constant Dialogs.tdiWarning.

62 DockTabSet Namespace

This is namespace DockTabSet.

Topics

Name	Description
Classes (see page 11594)	The following table lists classes in this documentation.
Interfaces (see page 11643)	The following table lists interfaces in this documentation.

Classes

	Name	Description
	ETabDockException (see page 11594)	The exception class for docktab control.
	TCaptionedTabDockTree (see page 11594)	Defines the captioned tab dock tree that can be instantiated to create a new docktab control.
	TDockClientInfo (see page 11598)	Provides the properties describing the dock client.
	TDockTabSet (see page 11604)	A docking tab set that has the appearance of notebook dividers.
	TTabDockPanel (see page 11625)	The panel for the TTabDock object.

Interfaces

	Name	Description
	IAlternateDockHost (see page 11643)	Implement this interface to allow items to be redocked from one area to another, such as when using a pin control.

62.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ETabDockException (see page 11594)	The exception class for docktab control.
	TCaptionedTabDockTree (see page 11594)	Defines the captioned tab dock tree that can be instantiated to create a new docktab control.
	TDockClientInfo (see page 11598)	Provides the properties describing the dock client.
	TDockTabSet (see page 11604)	A docking tab set that has the appearance of notebook dividers.
	TTabDockPanel (see page 11625)	The panel for the TTabDock object.

62.1.1 ETabDockException Class

The exception class for docktab control.

Class Hierarchy

Pascal

```
ETabDockException = class(Exception);
```

C++

```
class ETabDockException : public Exception;
```

File

DockTabSet

62.1.2 TCaptionedTabDockTree Class

Defines the captioned tab dock tree that can be instantiated to create a new docktab control.

Class Hierarchy

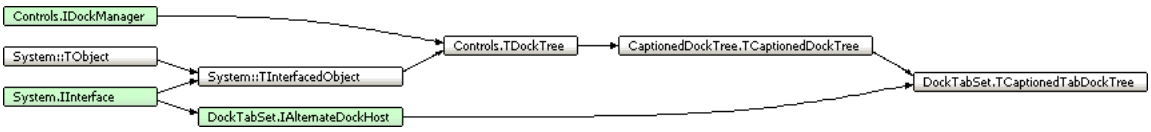

Pascal

```
TCaptionedTabDockTree = class(TCaptionedDockTree, IAlternateDockHost);
```

C++

```
class TCaptionedTabDockTree : public TCaptionedDockTree, public IAlternateDockHost;
```

File

DockTabSet

62.1.2.1 TCaptionedTabDockTree Members

The following tables list the members exposed by TCaptionedTabDockTree.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
V	AdjustCaptionRect (see page 2402)	This is AdjustCaptionRect, a member of class TCaptionedDockTree.
V	AdjustDockRect (see page 2402)	Adjusts the boundaries of the docking zone where a specified control is docked.
V	AdjustDockRect (see page 5987)	Adjusts the boundaries of the docking zone where a specified control is docked.
V	AdjustFrameRect (see page 5988)	This is AdjustFrameRect, a member of class TDockTree.
	BeginUpdate (see page 5988)	Responds when the docked controls are about to change.
	EndUpdate (see page 5988)	Responds when the docked controls have finished changing.
	FindControlAtPos (see page 5989)	This is FindControlAtPos, a member of class TDockTree.
	GetControlBounds (see page 5989)	Returns the dimensions allotted to a control that is already docked.
V	HitTest (see page 5990)	Indicates which docked control is at a specified mouse position.
V	InsertControl (see page 5990)	Handles the docking of a control into the associated DockSite (see page 5996).
	InternalCaptionHitTest (see page 2403)	This is InternalCaptionHitTest, a member of class TCaptionedDockTree.
V	LoadFromStream (see page 5991)	Creates docking zones for a set of docked controls read from a stream.
V	MouseDown (see page 5991)	This is MouseDown, a member of class TDockTree.
V	MouseMove (see page 5991)	This is MouseMove, a member of class TDockTree.
V	MouseUp (see page 11597)	This is MouseUp, a member of class TCaptionedTabDockTree.
V	PaintDockFrame (see page 2404)	Paints the elements of a docking zone that surround a docked control.
V	PaintDockFrame (see page 5992)	Paints the elements of a docking zone that surround a docked control.
V	PositionDockRect (see page 5993)	Computes the dimensions of a new docking zone.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
V	ReferenceFromOrient (see page 5993)	This is ReferenceFromOrient, a member of class TDockTree.
V	RemoveControl (see page 5993)	Handles the undocking of a control from the associated DockSite (see page 5996).
V	ResetBounds (see page 5994)	Recomputes the layout of docked controls.
V	SaveToStream (see page 5994)	Saves a list of the set of docked controls and their positions to a stream.
	SetReplacingControl (see page 5995)	Prepares the dock tree for replacing one docked control with another.
V	ShowHint (see page 5995)	This is ShowHint, a member of class TDockTree.
	UpdateAll (see page 5995)	Makes any internal adjustments to reflect the current state of the docked controls.
V	WndProc (see page 2404)	This is WndProc, a member of class TCaptionedDockTree.
V	WndProc (see page 5996)	This is WndProc, a member of class TDockTree.
V	ZoneCaptionHitTest (see page 2404)	This is ZoneCaptionHitTest, a member of class TCaptionedDockTree.
V	ZoneCaptionHitTest (see page 5996)	This is ZoneCaptionHitTest, a member of class TDockTree.

Protected Properties

	Name	Description
R	DockCaptionDrawer (see page 2405)	This is DockCaptionDrawer, a member of class TCaptionedDockTree.
R	DockCaptionOrientation (see page 2405)	This is DockCaptionOrientation, a member of class TCaptionedDockTree.

	DockSite (see page 5996)	Specifies the windowed control that uses this dock tree to manage docking operations.
	RelativeSizes (see page 5997)	This is RelativeSizes, a member of class TDockTree.
R	TopZone (see page 5997)	This is TopZone, a member of class TDockTree.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 6239)	Responds when the docked controls are about to change.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 5985)	Creates and initializes a TDockTree (see page 5983) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 2400)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2401)	Frees the memory associated with the TDockTree object.
V	Destroy (see page 5986)	Frees the memory associated with the TDockTree (see page 5983) object.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 6240)	Responds when the docked controls have finished changing.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetControlBounds (see page 6240)	Returns the dimensions allotted to a control that is already docked.
V	GetDockCaptionDrawer (see page 2403)	This is GetDockCaptionDrawer, a member of class TCaptionedDockTree.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetParentFormState (see page 2403)	Returns the state of the form that contains the control.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertControl (see page 6240)	Handles the docking of a control into the associated container control.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LoadFromStream (see page 6241)	Lays out a set of docked controls read from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintSite (see page 6241)	Paints the elements that surround all docked controls.
V	PaintSite (see page 5992)	Paints the elements that surround all docked controls.

	PositionDockRect (see page 6242)	Computes the dimensions of a new docking zone.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RemoveControl (see page 6242)	Handles the undocking of a control from the associated container control.
	ResetBounds (see page 6243)	Recomputes the layout of docked controls.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToStream (see page 6245)	Saves a list of the set of docked controls and their positions to a stream.
	SetAlternateDockHost (see page 11597)	Sets an item to dock on an alternate host, assuming that an alternate host is defined.
	SetReplacingControl (see page 6245)	This is SetReplacingControl, a member of class IDockManager.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

62.1.2.2 TCaptionedTabDockTree Methods

The methods of the TCaptionedTabDockTree class are listed here.

Protected Methods

	Name	Description
V	MouseUp (see page 11597)	This is MouseUp, a member of class TCaptionedTabDockTree.

Public Methods

	Name	Description
	SetAlternateDockHost (see page 11597)	Sets an item to dock on an alternate host, assuming that an alternate host is defined.

62.1.2.2.1 TCaptionedTabDockTree.MouseUp Method

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer; var
Handled: Boolean); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y, Boolean
Handled);
```

Description

This is MouseUp, a member of class TCaptionedTabDockTree.

62.1.2.2.2 TCaptionedTabDockTree.SetAlternateDockHost Method

Sets an item to dock on an alternate host, assuming that an alternate host is defined.

Pascal

```
procedure SetAlternateDockHost(const DockHost: TWinControl);
```

C++

```
__fastcall SetAlternateDockHost(const TWinControl * DockHost);
```

62.1.3 TDockClientInfo Class

Provides the properties describing the dock client.

Class Hierarchy

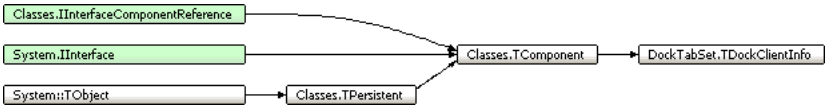

Pascal

```
TDockClientInfo = class(TComponent);
```

C++

```
class TDockClientInfo : public TComponent;
```

File

DockTabSet

62.1.3.1 TDockClientInfo Members

The following tables list the members exposed by TDockClientInfo.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 11602)	Forwards notification messages to all owned components.

	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11601)	This is Create, a member of class TDockClientInfo.
	CreateTab (see page 11602)	This is CreateTab, a member of class TDockClientInfo.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 11601)	Disposes of the component and its owned components.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClient (see page 11603)	Controls (see page 5711) how child controls are docked
R	DockTabSet (see page 11603)	Specifies the name of the current DockTabSet.
	ImageIndex (see page 11603)	Specifies an image's index number in the Images list.
R	ListIndex (see page 11603)	Lists the index of the tab in the collection of tabs that are part of the TDockTabSet (see page 11604).
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	ParentPanel (see page 11604)	Specifies the parent TDockTabSet (see page 11604) that is returned in the TDockClientInfo (see page 11598) class.
	TabIndex (see page 11604)	This is TabIndex, a member of class TDockClientInfo.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.

	Tag (see page 2808)	Stores an integer value as part of a component.

62.1.3.2 TDockClientInfo.Create Constructor

Pascal

```
constructor Create(const ADockTabSet: TDockTabSet; const ADockClient: TWinControl);
reintroduce;
```

C++

```
__fastcall TDockClientInfo(const TDockTabSet ADockTabSet, const TWinControl * ADockClient);
```

Description

This is Create, a member of class TDockClientInfo.

62.1.3.3 TDockClientInfo.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDockClientInfo();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

Destroy (see page 22287)

DestroyComponents (see page 2771)

Destroying (see page 2772)

Free (see page 22300)

Notification (see page 2783)

Owner (see page 2807)

Release (see page 13885)

Components (see page 2799)

62.1.3.4 TDockClientInfo Methods

The methods of the TDockClientInfo class are listed here.

Protected Methods

	Name	Description
	Notification (see page 11602)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
	CreateTab (see page 11602)	This is CreateTab, a member of class TDockClientInfo.

62.1.3.4.1 TDockClientInfo.CreateTab Method

Pascal

```
procedure CreateTab(AlwaysCreate: Boolean = False);
```

C++

```
__fastcall CreateTab(Boolean AlwaysCreate = False);
```

Description

This is CreateTab, a member of class TDockClientInfo.

62.1.3.4.2 TDockClientInfo.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

62.1.3.5 TDockClientInfo Properties

The properties of the TDockClientInfo class are listed here.

Public Properties

	Name	Description
R	DockClient (see page 11603)	Controls (see page 5711) how child controls are docked
R	DockTabSet (see page 11603)	Specifies the name of the current DockTabSet.
	ImageIndex (see page 11603)	Specifies an image's index number in the Images list.
R	ListIndex (see page 11603)	Lists the index of the tab in the collection of tabs that are part of the TDockTabSet (see page 11604).
R	ParentPanel (see page 11604)	Specifies the parent TDockTabSet (see page 11604) that is returned in the TDockClientInfo (see page 11598) class.
	TabIndex (see page 11604)	This is TabIndex, a member of class TDockClientInfo.

62.1.3.5.1 TDockClientInfo.DockClient Property

Controls (see page 5711) how child controls are docked

Pascal

```
property DockClient: TWinControl;
```

C++

```
__property TWinControl * DockClient;
```

62.1.3.5.2 TDockClientInfo.DockTabSet Property

Specifies the name of the current DockTabSet.

Pascal

```
property DockTabSet: TDockTabSet;
```

C++

```
__property TDockTabSet DockTabSet;
```

62.1.3.5.3 TDockClientInfo.ImageIndex Property

Specifies an image's index number in the Images list.

Pascal

```
property ImageIndex: Integer;
```

C++

```
__property int ImageIndex;
```

62.1.3.5.4 TDockClientInfo.ListIndex Property

Lists the index of the tab in the collection of tabs that are part of the TDockTabSet (see page 11604).

Pascal

```
property ListIndex: Integer;
```

C++

```
__property int ListIndex;
```

62.1.3.5.5 TDockClientInfo.ParentPanel Property

Specifies the parent TDockTabSet (see page 11604) that is returned in the TDockClientInfo (see page 11598) class.

Pascal

```
property ParentPanel: TTabDockPanel;
```

C++

```
__property TTabDockPanel ParentPanel;
```

62.1.3.5.6 TDockClientInfo.TabIndex Property

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

This is TabIndex, a member of class TDockClientInfo.

62.1.4 TDockTabSet Class

A docking tab set that has the appearance of notebook dividers.

Class Hierarchy

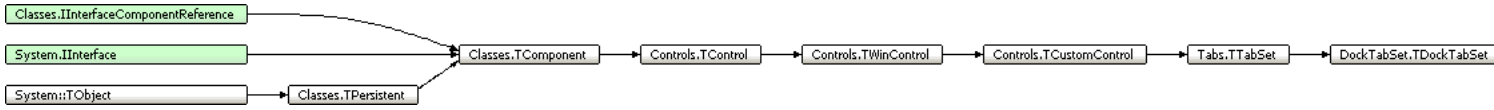

Pascal

```
TDockTabSet = class(TTabSet);
```

C++

```
class TDockTabSet : public TTabSet;
```

File

DockTabSet

Description

Inherits from TTabSet. Use TDockTabSet to add a control with multiple tab settings to a form. Unlike a page control, TDockTabSet is not made up of several pages that contain different controls. Instead, TDockTabSet is a single object. When the current tab changes, the tab control must directly update its contents to reflect the change in an OnChange event handler. Disabling a TDockTabSet object disables all tabs because it is one control.

62.1.4.1 TDockTabSet Members

The following tables list the members exposed by TDockTabSet.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetImageIndex (see page 11624)	This is OnGetImageIndex, a member of class TDockTabSet.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanChange (see page 23171)	This is CanChange, a member of class TTabSet.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	CMHintShow (see page 23171)	This is CMHintShow, a member of class TTabSet.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 23172)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.

	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 11617)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 11618)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 11619)	This is DockReplaceDockClient, a member of class TDockTabSet.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 11619)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawTab (see page 23172)	This is DrawTab, a member of class TTabSet.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindDockClientInfo (see page 11619)	This is FindDockClientInfo, a member of class TDockTabSet.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetChildren (see page 23172)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.

	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetImageIndex (see page 11620)	This is GetImageIndex, a member of class TDockTabSet.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTabName (see page 11620)	This is GetTabName, a member of class TDockTabSet.
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MeasureTab (see page 23174)	This is MeasureTab, a member of class TTabSet.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseDown (see page 23175)	OnMouseDown event dispatcher.
	MouseMove (see page 11620)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 11621)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 5920)	Renders the image of a custom control.
	Paint (see page 23176)	Renders the image of a custom control.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.

	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetTabIndex (see page 11621)	This is SetTabIndex, a member of class TDockTabSet.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShouldAutoHide (see page 11622)	This is ShouldAutoHide, a member of class TDockTabSet.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
V	ShowDockClientInfo (see page 11622)	This is ShowDockClientInfo, a member of class TDockTabSet.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.

V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	Images (see page 11623)	This is Images, a member of class TDockTabSet.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
R	Scroller (see page 23196)	This is Scroller, a member of class TTabSet.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.

	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 11616)	Constructs a tab set control.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 11617)	Frees the memory associated with the TTabSet object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	HideCurrentDockClient (see page 11620)	Makes the control invisible.
	IndexOfDockClient (see page 11620)	Specifies the index of the client in a collection.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ItemAtPos (see page 23173)	Returns the index of the tab indicated by the coordinates of a point on the control.
	ItemRect (see page 23173)	Returns the rectangle that surrounds the item specified in the Item parameter.
	ItemWidth (see page 23174)	Returns the width, in pixels, of a specified tab in the set.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MinClientRect (see page 23174)	This is the overview for the MinClientRect method overload.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectNext (see page 23177)	Selects the next tab in a tab set control, scrolling if necessary to bring the selected tab into view.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	ShowDockClient (see page 11622)	Makes the control visible
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	Canvas (see page 23183)	Specifies the TCanvas object that presents a drawing surface for the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.

R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
	FirstIndex (see page 23189)	Specifies the tab that appears in the leftmost visible position in the tab set control.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 23204)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 23204)	Occurs when an object with custom alignment is aligned.
	OnChange (see page 23205)	Occurs just before a new tab is selected.
	OnClick (see page 23205)	Occurs when the user clicks the control.
	OnDockDrop (see page 11624)	Occurs when the control is released from the mouse control and dropped onto the client.
	OnDockOver (see page 11624)	Occurs when the tabbed control is hovering over the dock site.
	OnDragDrop (see page 23207)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 23210)	Occurs when the user drags an object over a control.
	OnDrawTab (see page 23212)	Occurs when a tab in an owner-drawn tab set control needs to redisplay.
	OnEndDock (see page 23213)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 23213)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 23214)	Occurs when a control receives the input focus.
	OnExit (see page 23216)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 11625)	Returns information about the site or client control on which the tab control is docked.
	OnMeasureTab (see page 23218)	Occurs when an application needs to redisplay a tab in an owner-draw tab set.
	OnMouseActivate (see page 23219)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 23220)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 23222)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 23222)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 23222)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 23225)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 11625)	Occurs when the user begins to drag a control to a docked position.
	OnStartDrag (see page 23228)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnTabAdded (see page 11625)	Occurs when a tab is added to the tab control.
	OnTabRemoved (see page 11625)	Occurs when a tab is removed from the tab control.
	OnUnDock (see page 11625)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 23178)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 23182)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 23182)	Determines if scroll buttons automatically appear in a tab set control if there isn't room in the control to display all the tabs.
	AutoSelect (see page 11622)	Determines whether the control is automatically selected when it gets focus.
	BackgroundColor (see page 23183)	Determines the background color of the tab set control.
	Constraints (see page 23185)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DestinationDockSite (see page 11623)	Specifies the docking destination of the control.
	DitherBackground (see page 23186)	Determines whether the background color set with the BackgroundColor (see page 23183) property is dithered.
	DockSite (see page 11623)	Specifies the windowed control that uses this dock tree to manage docking operations.
	DragCursor (see page 23186)	Indicates the image used to represent the mouse pointer when the control is being dragged.

	DragKind (see page 23186)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 23187)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 23188)	Controls whether the control responds to mouse, keyboard, and timer events.
	EndMargin (see page 23189)	Determines how far, in pixels, the rightmost tab appears from the right edge of the tab set control.
	Font (see page 23190)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBackground (see page 23191)	Determines whether control uses parent's theme background.
	ParentShowHint (see page 23191)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 23192)	Identifies the pop-up menu associated with the control.
	SelectedColor (see page 23197)	Determines the color of the selected tab in the tab set control.
	ShowHint (see page 23197)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	ShrinkToFit (see page 23198)	This is ShrinkToFit, a member of class TTabSet.
	SoftTop (see page 23199)	Specifies whether a line appears at the edge of the tab set, setting off the non-current tabs.
	StartMargin (see page 23199)	Determines how far, in pixels, the first tab appears from the left edge of the tab set control.
	Style (see page 23199)	Determines whether the tab set is owner-drawn.
	TabHeight (see page 23200)	Determines the height of the tabs in an owner-drawn tab set control.
	TabIndex (see page 23200)	Determines which tab of a tab set control is currently selected.
	TabPosition (see page 23201)	This is TabPosition, a member of class TTabSet.
	Tabs (see page 23201)	Contains the list of text strings that appear on the tabs.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	UnselectedColor (see page 23201)	Determines the color of the tabs that aren't currently selected in the tab set control.
	Visible (see page 23201)	Determines whether the component appears onscreen.
R	VisibleTabs (see page 23203)	Contains the number of tabs currently visible in the tab set control.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

62.1.4.2 TDockTabSet.Create Constructor

Constructs a tab set control.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDockTabSet(TComponent * AOwner);
```

Description

Create sets the initial values of the control, including creating a TTabList object that holds the list of tabs for the control. The AOwner parameter specifies the component (usually a form) that is responsible for freeing the tab set.

See Also

Destroy (see page 23170)

62.1.4.3 TDockTabSet.Destroy Destructor

Frees the memory associated with the TTabSet object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDockTabSet();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks that the tab set is not nil before calling Destroy. Destroy frees the TTabList object that holds the list of tabs.

See Also

Create (see page 23170)

Free (see page 22300)

62.1.4.4 TDockTabSet Methods

The methods of the TDockTabSet class are listed here.

Protected Methods

	Name	Description
	DoAddDockClient (see page 11617)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 11618)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 11619)	This is DockReplaceDockClient, a member of class TDockTabSet.
	DoRemoveDockClient (see page 11619)	Responds when a control is undocked from the windowed control.
	FindDockClientInfo (see page 11619)	This is FindDockClientInfo, a member of class TDockTabSet.
	GetImageIndex (see page 11620)	This is GetImageIndex, a member of class TDockTabSet.
	GetTabName (see page 11620)	This is GetTabName, a member of class TDockTabSet.
	MouseMove (see page 11620)	Respond to mouse moving over control area..
	Notification (see page 11621)	Responds to notifications that components are being created or destroyed.
	SetTabIndex (see page 11621)	This is SetTabIndex, a member of class TDockTabSet.
	ShouldAutoHide (see page 11622)	This is ShouldAutoHide, a member of class TDockTabSet.
	ShowDockClientInfo (see page 11622)	This is ShowDockClientInfo, a member of class TDockTabSet.

Public Methods

	Name	Description
	HideCurrentDockClient (see page 11620)	Makes the control invisible.
	IndexOfDockClient (see page 11620)	Specifies the index of the client in a collection.
	ShowDockClient (see page 11622)	Makes the control visible

62.1.4.4.1 TDockTabSet.DoAddDockClient Method

Performs actions when a control is docked to the windowed control.

Pascal

```
procedure DoAddDockClient(Client: TControl; const ARect: TRect); override;
```

C++

```
virtual __fastcall DoAddDockClient(TControl * Client, const TRect ARect);
```

Description

DoAddDockClient is called automatically when a client control is docked to the windowed control. As implemented in TWinControl, DoAddDockClient sets the client's Parent property to the control.

Client is the control that has been docked.

ARect describes the boundaries of the region where Client has been docked.

Override DoAddDockClient to perform actions when a control is docked to the windowed control.

Note: DoAddDockClient is called in response to a message sent by the DockDrop method, before the OnDockDrop event.

See Also

Parent (see page 5832)

DockClients (see page 6201)

ManualDock (see page 5771)

DoRemoveDockClient (see page 6147)

DockDrop (see page 6142)

OnDockDrop (see page 6214)

62.1.4.4.2 TDockTabSet.DockOver Method

Responds when a docking client is dragged over the windowed control.

Pascal

```
procedure DockOver(Source: TDragDockObject; X: Integer; Y: Integer; State: TDragState; var
Accept: Boolean); override;
```

C++

```
virtual __fastcall DockOver(TDragDockObject Source, int X, int Y, TDragState State, Boolean
Accept);
```

Description

DockOver is called automatically for dock sites when a dockable object is dragged over them. As implemented in TWinControl, DockOver positions the docking rectangle and then calls DoDockOver to generate an OnDockOver event.

Override DockOver to perform additional actions when another control is dragged over the control.

DockOver is called only if DockSite is true.

See Also

OnDockOver (see page 6216)

DockSite (see page 6204)

DoDockOver (see page 6143)

62.1.4.4.3 TDockTabSet.DockReplaceDockClient Method

Pascal

```
function DockReplaceDockClient(Client: TControl; NewDockSite: TWinControl; DropControl:
TControl; ControlSide: TAlign; ReplacementClient: TControl): Boolean; override;
```

C++

```
virtual __fastcall Boolean DockReplaceDockClient(TControl * Client, TWinControl * NewDockSite,
TControl * DropControl, TAlign ControlSide, TControl * ReplacementClient);
```

Description

This is DockReplaceDockClient, a member of class TDockTabSet.

62.1.4.4.4 TDockTabSet.DoRemoveDockClient Method

Responds when a control is undocked from the windowed control.

Pascal

```
procedure DoRemoveDockClient(Client: TControl); override;
```

C++

```
virtual __fastcall DoRemoveDockClient(TControl * Client);
```

Description

DoRemoveDockClient is called automatically when a client control is undocked from the windowed control. As implemented in TWinControl, DoRemoveDockClient does nothing.

Client is the control that has been undocked.

Override DoRemoveDockClient to perform any necessary adjustments when a control is undocked from the windowed control.

Note: DoRemoveDockClient occurs at the end of the undocking process. To make changes when undocking starts, override the DoUnDock method instead.

See Also

DockClients (see page 6201)

ManualDock (see page 5771)

DoAddDockClient (see page 6142)

DoUnDock (see page 6147)

62.1.4.4.5 TDockTabSet.FindDockClientInfo Method

Pascal

```
function FindDockClientInfo(const Client: TControl): TDockClientInfo;
```

C++

```
__fastcall TDockClientInfo FindDockClientInfo(const TControl * Client);
```

Description

This is FindDockClientInfo, a member of class TDockTabSet.

62.1.4.4.6 TDockTabSet.GetImageIndex Method

Pascal

```
function GetImageIndex(TabIndex: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetImageIndex(int TabIndex);
```

Description

This is GetImageIndex, a member of class TDockTabSet.

62.1.4.4.7 TDockTabSet.GetTabName Method

Pascal

```
function GetTabName(const Client: TControl): string; virtual;
```

C++

```
virtual __fastcall AnsiString GetTabName(const TControl * Client);
```

Description

This is GetTabName, a member of class TDockTabSet.

62.1.4.4.8 TDockTabSet.HideCurrentDockClient Method

Makes the control invisible.

Pascal

```
procedure HideCurrentDockClient(const AnimateSpeed: Integer = 200);
```

C++

```
__fastcall HideCurrentDockClient(const int AnimateSpeed = 200);
```

62.1.4.4.9 TDockTabSet.IndexOfDockClient Method

Specifies the index of the client in a collection.

Pascal

```
function IndexOfDockClient(const Client: TControl): Integer;
```

C++

```
__fastcall int IndexOfDockClient(const TControl * Client);
```

62.1.4.4.10 TDockTabSet.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown (see page 5773)

MouseUp (see page 5774)

OnMouseMove (see page 5865)

62.1.4.4.11 TDockTabSet.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

62.1.4.4.12 TDockTabSet.SetTabIndex Method

Pascal

```
procedure SetTabIndex(Value: Integer); override;
```

C++

```
virtual __fastcall SetTabIndex(int Value);
```

Description

This is SetTabIndex, a member of class TDockTabSet.

62.1.4.4.13 TDockTabSet.ShouldAutoHide Method

Pascal

```
function ShouldAutoHide(const Client: TWinControl): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ShouldAutoHide(const TWinControl * Client);
```

Description

This is ShouldAutoHide, a member of class TDockTabSet.

62.1.4.4.14 TDockTabSet.ShowDockClient Method

Makes the control visible

Pascal

```
procedure ShowDockClient(const Client: TControl);
```

C++

```
__fastcall ShowDockClient(const TControl * Client);
```

62.1.4.4.15 TDockTabSet.ShowDockClientInfo Method

Pascal

```
procedure ShowDockClientInfo(const DockClientInfo: TDockClientInfo); virtual;
```

C++

```
virtual __fastcall ShowDockClientInfo(const TDockClientInfo DockClientInfo);
```

Description

This is ShowDockClientInfo, a member of class TDockTabSet.

62.1.4.5 TDockTabSet Properties

The properties of the TDockTabSet class are listed here.

Protected Properties

	Name	Description
	Images (see page 11623)	This is Images, a member of class TDockTabSet.

Published Properties

	Name	Description
	AutoSelect (see page 11622)	Determines whether the control is automatically selected when it gets focus.
	DestinationDockSite (see page 11623)	Specifies the docking destination of the control.
	DockSite (see page 11623)	Specifies the windowed control that uses this dock tree to manage docking operations.

62.1.4.5.1 TDockTabSet.AutoSelect Property

Determines whether the control is automatically selected when it gets focus.

Pascal

```
property AutoSelect: Boolean;
```

C++

```
__property Boolean AutoSelect;
```

Description

When this property is set to **True** the runtime behavior is that a tab on the control gets focus and is selected when the user moves the mouse pointer over the tab. If you want to test this, create a simple form application, drop a TDockTabSet (see page 11604) control onto the form, set the TStrings of the Tabs (see page 23133) property to `foo` and `bar`, set AutoSelect to **True**, then add the following code to a OnClick event:

```
procedure TForm1.DockTabSet1Click(Sender: TObject);
 var whichTab: string;
begin

  if DockTabSet1.TabIndex = 1 then
     whichTab := 'bar'
  else
     whichTab := 'foo';
     ShowMessage('Clicked '+ whichTab);
end;
```

62.1.4.5.2 TDockTabSet.DestinationDockSite Property

Specifies the docking destination of the control.

Pascal

```
property DestinationDockSite: TWinControl;
```

C++

```
__property TWinControl * DestinationDockSite;
```

Description

For example, the destination dock site of a tabbed control might be an instance of a form.

62.1.4.5.3 TDockTabSet.DockSite Property

Specifies the windowed control that uses this dock tree to manage docking operations.

Pascal

```
property DockSite: Boolean;
```

C++

```
__property Boolean DockSite;
```

62.1.4.5.4 TDockTabSet.Images Property

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

This is Images, a member of class TDockTabSet.

62.1.4.6 TDockTabSet Events

The events of the TDockTabSet class are listed here.

Protected Events

	Name	Description
	OnGetImageIndex (see page 11624)	This is OnGetImageIndex, a member of class TDockTabSet.

Published Events

	Name	Description
	OnDockDrop (see page 11624)	Occurs when the control is released from the mouse control and dropped onto the client.
	OnDockOver (see page 11624)	Occurs when the tabbed control is hovering over the dock site.
	OnGetSiteInfo (see page 11625)	Returns information about the site or client control on which the tab control is docked.
	OnStartDock (see page 11625)	Occurs when the user begins to drag a control to a docked position.
	OnTabAdded (see page 11625)	Occurs when a tab is added to the tab control.
	OnTabRemoved (see page 11625)	Occurs when a tab is removed from the tab control.
	OnUnDock (see page 11625)	Occurs when the application tries to undock a control that is docked to the windowed control.

62.1.4.6.1 TDockTabSet.OnDockDrop Event

Occurs when the control is released from the mouse control and dropped onto the client.

Pascal

```
property OnDockDrop: TDockDropEvent;
```

C++

```
__property TDockDropEvent OnDockDrop;
```

62.1.4.6.2 TDockTabSet.OnDockOver Event

Occurs when the tabbed control is hovering over the dock site.

Pascal

```
property OnDockOver: TDockOverEvent;
```

C++

```
__property TDockOverEvent OnDockOver;
```

62.1.4.6.3 TDockTabSet.OnGetImageIndex Event

Pascal

```
property OnGetImageIndex: TTabGetImageEvent;
```

C++

```
__property TTabGetImageEvent OnGetImageIndex;
```

Description

This is OnGetImageIndex, a member of class TDockTabSet.

62.1.4.6.4 TDockTabSet.OnGetSiteInfo Event

Returns information about the site or client control on which the tab control is docked.

Pascal

```
property OnGetSiteInfo: TGetSiteInfoEvent;
```

C++

```
__property TGetSiteInfoEvent OnGetSiteInfo;
```

62.1.4.6.5 TDockTabSet.OnStartDock Event

Occurs when the user begins to drag a control to a docked position.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

62.1.4.6.6 TDockTabSet.OnTabAdded Event

Occurs when a tab is added to the tab control.

Pascal

```
property OnTabAdded: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnTabAdded;
```

62.1.4.6.7 TDockTabSet.OnTabRemoved Event

Occurs when a tab is removed from the tab control.

Pascal

```
property OnTabRemoved: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnTabRemoved;
```

62.1.4.6.8 TDockTabSet.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock: TUnDockEvent;
```

C++

```
__property TUnDockEvent OnUnDock;
```

62.1.5 TTabDockPanel Class

The panel for the TTabDock object.

Class Hierarchy

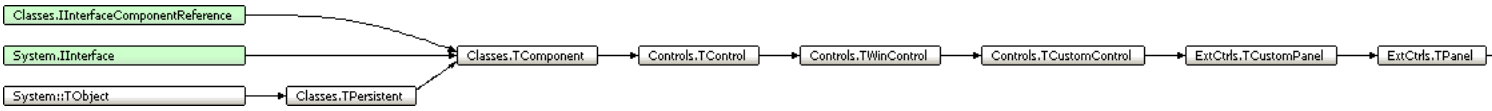

Pascal

```
TTabDockPanel = class(TPanel);
```

C++

```
class TTabDockPanel : public TPanel;
```

File

DockTabSet

Description

Use TDockTabSet (see page 11604) instead of TTabDock, when possible.

62.1.5.1 TTabDockPanel Members

The following tables list the members exposed by TTabDockPanel.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 12537)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAnimation (see page 11639)	This is BeginAnimation, a member of class TTabDockPanel.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanAutoSize (see page 12537)	Adjusts the way the panel responds to resize attempts.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.

	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	CMShowingChanged (see page 11640)	This is CMShowingChanged, a member of class TTabDockPanel.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 12537)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.

	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EndAnimation (see page 11640)	This is EndAnimation, a member of class TTabDockPanel.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.

	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 5920)	Renders the image of a custom control.
V	Paint (see page 12538)	Renders the image of the panel.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.

	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBackground (see page 12539)	This is SetParentBackground, a member of class TCustomPanel.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShouldAutoHide (see page 11640)	This is ShouldAutoHide, a member of class TTabDockPanel.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMEraseBkgnd (see page 11640)	This is WMEraseBkgnd, a member of class TTabDockPanel.
	WMNCPaint (see page 11641)	This is WMNCPaint, a member of class TTabDockPanel.
	WMPaint (see page 11641)	This is WMPaint, a member of class TTabDockPanel.
	WMPrint (see page 11641)	This is WMPrint, a member of class TTabDockPanel.
	WMPrintClient (see page 11641)	This is WMPrintClient, a member of class TTabDockPanel.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	Alignment (see page 12539)	Determines how the caption is aligned within the panel.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelInner (see page 12540)	Determines the style of the inner bevel of a panel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelOuter (see page 12541)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BevelWidth (see page 12542)	Determines the width, in pixels, of both the inner and outer bevels of a panel.

	BorderStyle (see page 12543)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	BorderWidth (see page 12543)	Specifies the distance, in pixels, between the outer and inner bevels.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Color (see page 12544)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
R	DockClientInfo (see page 11642)	This is DockClientInfo, a member of class TTabDockPanel.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	FullRepaint (see page 12545)	Determines how the panel repaints itself when it is resized.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	Locked (see page 12545)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 12546)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
R	SplitterPanel (see page 11642)	This is SplitterPanel, a member of class TTabDockPanel.
	SplitterWidth (see page 11642)	This is SplitterWidth, a member of class TTabDockPanel.
	Text (see page 5842)	Contains a text string associated with the control.
	VerticalAlignment (see page 12547)	Sets the vertical position of the caption.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 11638)	This is Create, a member of class TTabDockPanel.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 11638)	Destroys an instance of TCustomControl.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
V	GetControlsAlignment (see page 12538)	Returns the alignment of text in the panel.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.

	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UpdateDockCaptionPin (see page 11640)	Updates the pin state for the DockCaption of the TTabDockPanel (see page 11625).
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AnimateSpeed (see page 11642)	Sets the animation speed on the panel and pin.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.

	DockManager (see page 13060)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentBackground (see page 12546)	Determines whether control uses parent's theme background.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 13080)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 13081)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 13082)	Occurs when an attempt is made to resize the control.
	OnClick (see page 13082)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 13084)	Adjust resize constraints.
	OnContextPopup (see page 13084)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 13085)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 13085)	Occurs when another control is docked to the control.
	OnDockOver (see page 13087)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 13090)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13092)	Occurs when the user drags an object over a control.
	OnEndDock (see page 13094)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

	OnEndDrag (see page 13095)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 13096)	Occurs when a control receives the input focus.
	OnExit (see page 13098)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 13100)	Returns the control's docking information.
	OnMouseActivate (see page 13102)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 13103)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 13105)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 13105)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 13105)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 13108)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 13110)	Occurs immediately after the control is resized.
	OnStartDock (see page 13111)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 13111)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 13112)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 13044)	Determines how the control aligns within its container (parent control).
	Alignment (see page 13048)	Determines how the caption is aligned within the panel.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 13048)	Specifies how the control is anchored to its parent.
	AutoSize (see page 13049)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 13049)	Specifies which edges of the control are beveled.
	BevelInner (see page 13050)	Determines the style of the inner bevel of a panel.
	BevelKind (see page 13051)	Specifies the control's bevel style.
	BevelOuter (see page 13051)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 13052)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BiDiMode (see page 13053)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 13054)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 13054)	Specifies the distance, in pixels, between the outer and inner bevels.
	Caption (see page 13055)	Specifies a text string that identifies the control to the user.
	Color (see page 13058)	Specifies the background color of the control.
	Constraints (see page 13059)	Specifies the size constraints for the control.
	Ctl3D (see page 13059)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DockSite (see page 13062)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 13063)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 13063)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 13063)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 13065)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 13066)	Controls the attributes of text written on or in the control.
	FullRepaint (see page 13067)	Determines how the panel repaints itself when it is resized.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.

	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Locked (see page 13067)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Padding (see page 13067)	Specifies the padding of a control.
	ParentBackground (see page 13068)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 13068)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 13068)	Determines where a control looks for its color information.
	ParentCtl3D (see page 13069)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 13069)	Determines where a control looks for its font information.
	ParentShowHint (see page 13070)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 13070)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 13075)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 13076)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 13077)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	UseDockManager (see page 13077)	Specifies whether the docking manager is used in drag-and-dock operations.
	VerticalAlignment (see page 13078)	Sets the vertical position of the caption.
	Visible (see page 13078)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

62.1.5.2 TTabDockPanel.Create Constructor

Pascal

```
constructor Create(AOwner: TComponent; DockClientInfo: TDockClientInfo); reintroduce;
```

C++

```
__fastcall TTabDockPanel(TComponent * AOwner, TDockClientInfo DockClientInfo);
```

Description

This is Create, a member of class TTabDockPanel.

62.1.5.3 TTabDockPanel.Destroy Destructor

Destroys an instance of TCustomControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTabDockPanel();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the control is not nil, and only then calls Destroy.

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Destroys the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

Free (see page 22300)

delete

Create (see page 5918)

62.1.5.4 TTabDockPanel Methods

The methods of the TTabDockPanel class are listed here.

Protected Methods

	Name	Description
	BeginAnimation (see page 11639)	This is BeginAnimation, a member of class TTabDockPanel.
	CMShowingChanged (see page 11640)	This is CMShowingChanged, a member of class TTabDockPanel.
	EndAnimation (see page 11640)	This is EndAnimation, a member of class TTabDockPanel.
	ShouldAutoHide (see page 11640)	This is ShouldAutoHide, a member of class TTabDockPanel.
	WMEraseBkgnd (see page 11640)	This is WMEraseBkgnd, a member of class TTabDockPanel.
	WMNCPaint (see page 11641)	This is WMNCPaint, a member of class TTabDockPanel.
	WMPaint (see page 11641)	This is WMPaint, a member of class TTabDockPanel.
	WMPrint (see page 11641)	This is WMPrint, a member of class TTabDockPanel.
	WMPrintClient (see page 11641)	This is WMPrintClient, a member of class TTabDockPanel.

Public Methods

	Name	Description
	UpdateDockCaptionPin (see page 11640)	Updates the pin state for the DockCaption of the TTabDockPanel (see page 11625).

62.1.5.4.1 TTabDockPanel.BeginAnimation Method

Pascal

```
procedure BeginAnimation;
```

C++

```
__fastcall BeginAnimation();
```

Description

This is BeginAnimation, a member of class TTabDockPanel.

62.1.5.4.2 TTabDockPanel.CMShowingChanged Method

Pascal

```
procedure CMShowingChanged(var Message: TMessage); message CM_SHOWINGCHANGED;
```

C++

```
__fastcall CMShowingChanged(TMessage Message);
```

Description

This is CMShowingChanged, a member of class TTabDockPanel.

62.1.5.4.3 TTabDockPanel.EndAnimation Method

Pascal

```
procedure EndAnimation;
```

C++

```
__fastcall EndAnimation();
```

Description

This is EndAnimation, a member of class TTabDockPanel.

62.1.5.4.4 TTabDockPanel.ShouldAutoHide Method

Pascal

```
function ShouldAutoHide: Boolean;
```

C++

```
__fastcall Boolean ShouldAutoHide();
```

Description

This is ShouldAutoHide, a member of class TTabDockPanel.

62.1.5.4.5 TTabDockPanel.UpdateDockCaptionPin Method

Updates the pin state for the DockCaption of the TTabDockPanel (see page 11625).

Pascal

```
procedure UpdateDockCaptionPin;
```

C++

```
__fastcall UpdateDockCaptionPin();
```

62.1.5.4.6 TTabDockPanel.WMEraseBkgnd Method

Pascal

```
procedure WMEraseBkgnd(var Message: TWMEraseBkgnd); message WM_ERASEBKGND;
```

C++

```
__fastcall WMEraseBkgnd(TWMEraseBkgnd Message);
```

Description

This is WMEraseBkgnd, a member of class TTabDockPanel.

62.1.5.4.7 TTabDockPanel.WMNCPaint Method

Pascal

```
procedure WMNCPaint(var Message: TWMNCPaint); message WM_NCPAINT;
```

C++

```
__fastcall WMNCPaint(TWMNCPaint Message);
```

Description

This is WMNCPaint, a member of class TTabDockPanel.

62.1.5.4.8 TTabDockPanel.WMPaint Method

Pascal

```
procedure WMPaint(var Message: TWMPaint); message WM_PAINT;
```

C++

```
__fastcall WMPaint(TWMPaint Message);
```

Description

This is WMPaint, a member of class TTabDockPanel.

62.1.5.4.9 TTabDockPanel.WMPrint Method

Pascal

```
procedure WMPrint(var Message: TWMPrint); message WM_PRINT;
```

C++

```
__fastcall WMPrint(TWMPrint Message);
```

Description

This is WMPrint, a member of class TTabDockPanel.

62.1.5.4.10 TTabDockPanel.WMPrintClient Method

Pascal

```
procedure WMPrintClient(var Message: TWMPrintClient); message WM_PRINTCLIENT;
```

C++

```
__fastcall WMPrintClient(TWMPrintClient Message);
```

Description

This is WMPrintClient, a member of class TTabDockPanel.

62.1.5.5 TTabDockPanel Properties

The properties of the TTabDockPanel class are listed here.

Protected Properties

	Name	Description
R	DockClientInfo (see page 11642)	This is DockClientInfo, a member of class TTabDockPanel.
R	SplitterPanel (see page 11642)	This is SplitterPanel, a member of class TTabDockPanel.
	SplitterWidth (see page 11642)	This is SplitterWidth, a member of class TTabDockPanel.

Public Properties

	Name	Description
	AnimateSpeed (see page 11642)	Sets the animation speed on the panel and pin.

62.1.5.5.1 TTabDockPanel.AnimateSpeed Property

Sets the animation speed on the panel and pin.

Pascal

```
property AnimateSpeed: Integer;
```

C++

```
__property int AnimateSpeed;
```

62.1.5.5.2 TTabDockPanel.DockClientInfo Property

Pascal

```
property DockClientInfo: TDockClientInfo;
```

C++

```
__property TDockClientInfo DockClientInfo;
```

Description

This is DockClientInfo, a member of class TTabDockPanel.

62.1.5.5.3 TTabDockPanel.SplitterPanel Property

Pascal

```
property SplitterPanel: TPanel;
```

C++

```
__property TPanel SplitterPanel;
```

Description

This is SplitterPanel, a member of class TTabDockPanel.

62.1.5.5.4 TTabDockPanel.SplitterWidth Property

Pascal

```
property SplitterWidth: Integer;
```

C++

```
__property int SplitterWidth;
```

Description

This is SplitterWidth, a member of class TTabDockPanel.

62.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAlternateDockHost (see page 11643)	Implement this interface to allow items to be redocked from one area to another, such as when using a pin control.

62.2.1 IAlternateDockHost Interface

Implement this interface to allow items to be redocked from one area to another, such as when using a pin control.

Class Hierarchy

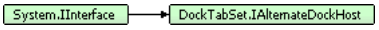

Pascal

```
IAlternateDockHost = interface(IInterface);
```

C++

```
__interface IAlternateDockHost : public IInterface;
```

File

DockTabSet

62.2.1.1 IAlternateDockHost Members

The following tables list the members exposed by IAlternateDockHost.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	SetAlternateDockHost (see page 11644)	Sets an item to dock on an alternate host, assuming that an alternate host is defined.

62.2.1.2 IAlternateDockHost Methods

The methods of the IAlternateDockHost class are listed here.

Public Methods

	Name	Description
	SetAlternateDockHost (see page 11644)	Sets an item to dock on an alternate host, assuming that an alternate host is defined.

62.2.1.2.1 IAlternateDockHost.SetAlternateDockHost Method

Sets an item to dock on an alternate host, assuming that an alternate host is defined.

Pascal

```
procedure SetAlternateDockHost(const DockHost: TWinControl);
```

C++

```
__fastcall SetAlternateDockHost(const TWinControl * DockHost);
```

63 DRIntf Namespace

This is namespace DRIntf.

Topics

Name	Description
Functions (see page 11647)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 11661)	The following table lists structs, records, enums in this documentation.
Types (see page 11662)	The following table lists types in this documentation.
Constants (see page 11664)	The following table lists constants in this documentation.

Constants

Name	Description
NullAttrID (see page 11664)	This is constant DRIntf.NullAttrID.
NullDatabaseID (see page 11664)	This is constant DRIntf.NullDatabaseID.
NullFieldID (see page 11665)	This is constant DRIntf.NullFieldID.
NullID (see page 11665)	This is constant DRIntf.NullID.
NullTableID (see page 11665)	This is constant DRIntf.NullTableID.

Functions

	Name	Description
	AssociateAttr (see page 11647)	Sets up an association between an attribute ID and a field ID.
	CreateField (see page 11648)	Creates a new field from information stored in the Data Dictionary.
	CurrentDictionary (see page 11649)	This is function DRIntf.CurrentDictionary.
	DictionaryActive (see page 11649)	Indicates whether there is an active Data Dictionary.
	DictionaryDeactivate (see page 11649)	Deactivates the Data Dictionary.
	FindAttrID (see page 11650)	Returns the ID of a specified attribute set.
	FindDatabaseID (see page 11650)	Returns the ID of a specified database.
	FindFieldID (see page 11651)	Returns the ID of a specified field.
	FindTableID (see page 11652)	Returns the ID of a specified table.
	GetAttrID (see page 11652)	Returns the ID of the attribute set associated with a field.
	GetAttrName (see page 11653)	Returns the name of an attribute set given its ID.
	GetAttrNames (see page 11653)	Executes a callback for every attribute set name.
	GetControlClass (see page 11654)	Returns the class associated with an attribute set.
	HasConstraints (see page 11654)	Indicates whether there are any data constraints for a dataset stored in the Data Dictionary.
	IsNullID (see page 11655)	Indicates whether a Data Dictionary Identifier is null.

	NewAttr (see page 11656)	Creates an attribute set based on the properties of a field component.
	QualifyTableName (see page 11656)	Returns the fully qualified name of a table in the Data Dictionary.
	QualifyTableNameByName (see page 11657)	Returns the fully qualified name of a table in the Data Dictionary.
	UnassociateAttr (see page 11657)	Removes the association between an attribute ID and a field ID.
	UpdateAttr (see page 11658)	Updates an attribute set to match the properties of a field component.
	UpdateConstraints (see page 11659)	Replaces all imported constraints for a dataset with the current constraints in the Data Dictionary.
	UpdateDataset (see page 11659)	Updates a dataset and all its field components to match the Data Dictionary.
	UpdateField (see page 11660)	Updates a field component to match the associated attribute set in the Data Dictionary.

Structs, Records, Enums

	Name	Description
	TID (see page 11661)	This is record DRIntf.TID.

Types

Name	Description
TAttrID (see page 11662)	This is type DRIntf.TAttrID.
TDatabaseID (see page 11662)	This is type DRIntf.TDatabaseID.
TFieldID (see page 11663)	This is type DRIntf.TFieldID.
TTableID (see page 11663)	This is type DRIntf.TTableID.

63.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AssociateAttr (see page 11647)	Sets up an association between an attribute ID and a field ID.
	CreateField (see page 11648)	Creates a new field from information stored in the Data Dictionary.
	CurrentDictionary (see page 11649)	This is function DRIntf.CurrentDictionary.
	DictionaryActive (see page 11649)	Indicates whether there is an active Data Dictionary.
	DictionaryDeactivate (see page 11649)	Deactivates the Data Dictionary.
	FindAttrID (see page 11650)	Returns the ID of a specified attribute set.
	FindDatabaseID (see page 11650)	Returns the ID of a specified database.
	FindFieldID (see page 11651)	Returns the ID of a specified field.
	FindTableID (see page 11652)	Returns the ID of a specified table.
	GetAttrID (see page 11652)	Returns the ID of the attribute set associated with a field.
	GetAttrName (see page 11653)	Returns the name of an attribute set given its ID.
	GetAttrNames (see page 11653)	Executes a callback for every attribute set name.
	GetControlClass (see page 11654)	Returns the class associated with an attribute set.
	HasConstraints (see page 11654)	Indicates whether there are any data constraints for a dataset stored in the Data Dictionary.
	IsNullID (see page 11655)	Indicates whether a Data Dictionary Identifier is null.
	NewAttr (see page 11656)	Creates an attribute set based on the properties of a field component.
	QualifyTableName (see page 11656)	Returns the fully qualified name of a table in the Data Dictionary.
	QualifyTableNameByName (see page 11657)	Returns the fully qualified name of a table in the Data Dictionary.
	UnassociateAttr (see page 11657)	Removes the association between an attribute ID and a field ID.
	UpdateAttr (see page 11658)	Updates an attribute set to match the properties of a field component.
	UpdateConstraints (see page 11659)	Replaces all imported constraints for a dataset with the current constraints in the Data Dictionary.
	UpdateDataset (see page 11659)	Updates a dataset and all its field components to match the Data Dictionary.
	UpdateField (see page 11660)	Updates a field component to match the associated attribute set in the Data Dictionary.

63.1.1 DRIntf.AssociateAttr Function

Sets up an association between an attribute ID and a field ID.

Pascal

```
procedure AssociateAttr(const AttrID: TAttrID; const FieldID: TFieldID);
```

C++

```
AssociateAttr(const TAttrID AttrID, const TFieldID FieldID);
```

File

DRIntf

Description

Call AssociateAttr to associate an attribute set in the Data Dictionary with a field in the field dictionary.

See Also

UnassociateAttr (see page 11657)

FindFieldID (see page 11651)

FindAttrID (see page 11650)

GetAttrID (see page 11652)

NewAttr (see page 11656)

The Data Dictionary

63.1.2 DRIntf.CreateField Function

Creates a new field from information stored in the Data Dictionary.

Pascal

```
function CreateField(Owner: TComponent; Dataset: TDataset; const FullName: string; const
Origin: string; const FieldID: TFieldID; const AttrID: TAttrID): TField;
```

C++

```
TField CreateField(TComponent * Owner, TDataset Dataset, const AnsiString FullName, const
AnsiString Origin, const TFieldID FieldID, const TAttrID AttrID);
```

File

DRIntf

Description

Call CreateField to create a field component based on the information in the Data Dictionary.

The Owner parameter specifies the component responsible for freeing the field.

The Dataset parameter specifies the dataset that includes the field component.

The FullName and Origin parameters specify the values of the field component's properties of the same names. Origin can be an empty string.

The FieldID and AttrID parameters specify the field in the field dictionary and an attribute set to use when creating the field. Both can be 0. If FieldID is specified and AttrID is not, the attribute set associated with FieldID is used.

See Also

TField

TField

TField

AssociateAttr (see page 11647)

FindFieldID (see page 11651)

FindAttrID (see page 11650)

GetControlclass

The Data Dictionary

63.1.3 DRIntf.CurrentDictionary Function

Pascal

```
function CurrentDictionary: HDBIDR;
```

C++

```
HDBIDR CurrentDictionary();
```

File

DRIntf

Description

This is function DRIntf.CurrentDictionary.

63.1.4 DRIntf.DictionaryActive Function

Indicates whether there is an active Data Dictionary.

Pascal

```
function DictionaryActive: Boolean;
```

C++

```
Boolean DictionaryActive();
```

File

DRIntf

Description

Call DictionaryActive to check whether the Data Dictionary is active. If the dictionary is not active, applications can't store or retrieve attribute sets.

DictionaryActive returns true if the dictionary is active, false otherwise.

See Also

DictionaryDeactivate (see page 11649)

The Data Dictionary

63.1.5 DRIntf.DictionaryDeactivate Function

Deactivates the Data Dictionary.

Pascal

```
procedure DictionaryDeactivate;
```

C++

```
DictionaryDeactivate();
```

File

DRIntf

Description

Call DictionaryDeactivate to disable the Data Dictionary.

See Also

DictionaryActive (see page 11649)

The Data Dictionary

63.1.6 DRIntf.FindAttrID Function

Returns the ID of a specified attribute set.

Pascal

```
function FindAttrID(const AttrName: string): TAttrID;
```

C++

```
TAttrID FindAttrID(const AnsiString AttrName);
```

File

DRIntf

Description

Call FindAttrID to get the attribute ID for an attribute set that is stored in the Data Dictionary. AttrName is the name of the attribute set. If the Data Dictionary does not contain an attribute set with the specified name, FindAttrID returns a null ID.

See Also

NewAttr (see page 11656)

GetAttrID (see page 11652)

GetAttrName (see page 11653)

GetAttrNames (see page 11653)

IsNullID (see page 11655)

The Data Dictionary

63.1.7 DRIntf.FindDatabaseID Function

Returns the ID of a specified database.

Pascal

```
function FindDatabaseID(const Database: string): TDatabaseID;
```

C++

```
TDatabaseID FindDatabaseID(const AnsiString Database);
```

File

DRIntf

Description

Call FindDatabaseID to get the database ID for the database specified by the Database parameter. This ID can be used for locating the database's tables in the Data Dictionary.

If the named database is not in the Data Dictionary, FindDatabaseID returns a null ID.

See Also

FindTableID (see page 11652)

IsNullID (see page 11655)

The Data Dictionary

63.1.8 DRIntf.FindFieldID Function

Returns the ID of a specified field.

Pascal

```
function FindFieldID(const Table: TTableID; const Field: string): TFieldID;
```

C++

```
TFieldID FindFieldID(const TTableID Table, const AnsiString Field);
```

File

DRIntf

Description

Call FindFieldID to get the ID for a field that is stored in the field dictionary. Use this field ID to get or set an associated attribute set or to use as a basis when creating a new field component.

The Table parameter specifies the ID of the table that contains the field. The Field parameter specifies the name of the field.

If the Data Dictionary does not contain an entry for the specified field, FindFieldID returns a null ID.

See Also

GetAttrID (see page 11652)

FindTableID (see page 11652)

IsNullID (see page 11655)

UpdateAttr (see page 11658)

createField

UpdateField (see page 11660)

UnassociateAttr (see page 11657)

AssociateAttr (see page 11647)

The Data Dictionary

63.1.9 DRIntf.FindTableID Function

Returns the ID of a specified table.

Pascal

```
function FindTableID(const Database: TDatabaseID; const Table: string): TTableID;
```

C++

```
TTableID FindTableID(const TDatabaseID Database, const AnsiString Table);
```

File

DRIntf

Description

Call FindTableID to get the ID for a table in the database specified by the Database parameter. This ID can be used for locating the database's fields in the Data Dictionary.

The Database parameter specifies the database in which to locate the table. This ID can be obtained by calling FindDatabaseID (see page 11650).

The Table parameter is the name of table.

If the database ID is invalid or the named table is not in the Data Dictionary, FindTableID returns a null ID.

See Also

FindDatabaseID (see page 11650)

FindFieldID (see page 11651)

IsNullID (see page 11655)

QualifyTablename

The Data Dictionary

63.1.10 DRIntf.GetAttrID Function

Returns the ID of the attribute set associated with a field.

Pascal

```
function GetAttrID(const FieldID: TFieldID): TAttrID;
```

C++

```
TAttrID GetAttrID(const TFieldID FieldID);
```

File

DRIntf

Description

Call GetAttrID to get the ID for the attribute set associated with a field. The FieldID parameter is the Data Dictionary ID for the field whose attribute set is desired.

If the FieldID is not valid or the field does not have an associated attribute set, GetAttrID returns a null ID.

See Also

FindAttrID (see page 11650)

AssociateAttr (see page 11647)

FindFieldID (see page 11651)

IsNullID (see page 11655)

The Data Dictionary

63.1.11 DRIntf.GetAttrName Function

Returns the name of an attribute set given its ID.

Pascal

```
function GetAttrName(const AttrID: TAttrID): string;
```

C++

```
AnsiString GetAttrName(const TAttrID AttrID);
```

File

DRIntf

Description

Call GetAttrName to get the name of an attribute set from its ID.

See Also

FindAttrID (see page 11650)

NewAttr (see page 11656)

GetAttrNames (see page 11653)

GetAttrID (see page 11652)

The Data Dictionary

63.1.12 DRIntf.GetAttrNames Function

Executes a callback for every attribute set name.

Pascal

```
procedure GetAttrNames(Proc: TGetStrProc);
```

C++

```
GetAttrNames(TGetStrProc Proc);
```

File

DRIntf

Description

Call GetAttrNames to perform a callback for every attribute set in the Data Dictionary. The Proc parameter is called for every attribute set with its parameter set to the name of the attribute set.

The callback can obtain an attribute ID from the attribute set's name by calling FindAttrID (see page 11650).

See Also

NewAttr (see page 11656)

GetAttrName (see page 11653)

FindAttrID (see page 11650)

The Data Dictionary

63.1.13 DRIntf.GetControlClass Function

Returns the class associated with an attribute set.

Pascal

```
function GetControlClass(const AttrID: TAttrID): string;
```

C++

```
AnsiString GetControlClass(const TAttrID AttrID);
```

File

DRIntf

Description

Call GetControlClass to get name of the control class (a TField descendant) associated with an attribute set. If no class is associated with the attribute set, GetControlClass returns an empty string.

Note: To obtain a class reference from the class name, use the GetClass (Delphi) or __classid (C++) function.

See Also

TObject

CreateField (see page 11648)

FindAttrID (see page 11650)

GetClass

</linkItem> <linkItem type="concept" library="win32"> <link xlinkHref="BDETheDataDictionary.xml">The Data Dictionary</link> </linkItem> </linkList> </api>

63.1.14 DRIntf.HasConstraints Function

Indicates whether there are any data constraints for a dataset stored in the Data Dictionary.

Pascal

```
function HasConstraints(Dataset: TDBDataset): Boolean;
```

C++

```
Boolean HasConstraints(TDBDataset Dataset);
```

File

DRIntf

Description

Call HasConstraints to determine whether the Data Dictionary has any constraints recorded for the specified dataset.

See Also

UpdateConstraints (see page 11659)

TDataSet

TField

UpdateDataset (see page 11659)

The Data Dictionary

63.1.15 DRIntf.IsNullID Function

Indicates whether a Data Dictionary Identifier is null.

Pascal

```
function IsNullID(const ID): Boolean;
```

C++

```
Boolean IsNullID(const  ID);
```

File

DRIntf

Description

Call IsNullID to determine whether the ID for an entity in the Data Dictionary is null. IsNullID returns true if the ID is null, false otherwise.

Use IsNullID to test the return value of functions that look up items in the Data Dictionary.

See Also

FindDatabaseID (see page 11650)

FindFieldID (see page 11651)

FindTableID (see page 11652)

FindAttrID (see page 11650)

GetAttrID (see page 11652)

TAttrID (see page 11662)

The Data Dictionary

63.1.16 DRIntf.NewAttr Function

Creates an attribute set based on the properties of a field component.

Pascal

```
function NewAttr(Field: TField; const FieldID: TFieldID; const AttrName: string; const Parent:
TAttrID): TAttrID;
```

C++

```
TAttrID NewAttr(TField Field, const TFieldID FieldID, const AnsiString AttrName, const TAttrID
Parent);
```

File

DRIntf

Description

Call NewAttr create an attribute set in the data dictionary.

The attributes in the attribute set reflect the current properties of the field component specified by the Field parameter.

If a FieldID is provided, the attribute set is associated with Field ID. To create an attribute set without associating it with a field in the Data Dictionary, set the FieldID parameter to NullFieldID (see page 11665).

The AttrName parameter specifies the name under which the attribute set is stored.

The Parent parameter specifies a parent attribute set from which the new attribute set inherits values. Only field properties that differ from the values in the parent attribute set are saved with the new attribute set. If Parent is NullAttrID (see page 11664), all field properties are saved in the new attribute set.

NewAttr returns the attribute ID for the new attribute set.

See Also

FindAttrID (see page 11650)

FindFieldID (see page 11651)

GetAttrName (see page 11653)

UpdateAttr (see page 11658)

AssociateAttr (see page 11647)

The Data Dictionary

63.1.17 DRIntf.QualifyTableName Function

Returns the fully qualified name of a table in the Data Dictionary.

Pascal

```
function QualifyTableName(Database: TDatabase; const TableName: string): string;
```

C++

```
AnsiString QualifyTableName(TDatabase Database, const AnsiString TableName);
```

File

DRIntf

Description

Call QualifyTableName to add user name information to a database table name. Specify the database that contains the table using the Database parameter. Specify the name of a table within that database using the TableName parameter. QualifyTableName returns the fully qualified name of the database table if the table is stored in the Data Dictionary qualified by user name.

See Also

QualifyTableNameByName (see page 11657)

The Data Dictionary

63.1.18 DRIntf.QualifyTableNameByName Function

Returns the fully qualified name of a table in the Data Dictionary.

Pascal

```
function QualifyTableNameByName(const SessionName: string; const DatabaseName: string; const
TableName: string): string;
```

C++

```
AnsiString QualifyTableNameByName(const AnsiString SessionName, const AnsiString DatabaseName,
const AnsiString TableName);
```

File

DRIntf

Description

Call QualifyTableNameByName to add user name information to a database table name. Specify the BDE session that manages the database connection with the Session Name parameter. Specify the database that contains the table using the Database parameter. Specify the name of a table within that database using the TableName parameter. QualifyTableNameByName returns the fully qualified name of the database table if the table is stored in the Data Dictionary qualified by user name.

See Also

QualifyTablename

TSession

TDatabase

The Data Dictionary

63.1.19 DRIntf.UnassociateAttr Function

Removes the association between an attribute ID and a field ID.

Pascal

```
procedure UnassociateAttr(const FieldID: TFieldID);
```

C++

```
UnassociateAttr(const TFieldID FieldID);
```

File

DRIntf

Description

Call UnassociateAttr to end the association between the field specified by FieldID and its current attribute set in the Data Dictionary.

See Also

AssociateAttr (see page 11647)

FindFieldID (see page 11651)

The Data Dictionary

63.1.20 DRIntf.UpdateAttr Function

Updates an attribute set to match the properties of a field component.

Pascal

```
procedure UpdateAttr(Field: TField; const FieldID: TFieldID; const AttrID: TAttrID);
```

C++

```
UpdateAttr(TField Field, const TFieldID FieldID, const TAttrID AttrID);
```

File

DRIntf

Description

Call UpdateAttr to update an attribute set so that it matches the properties of the field component specified by the Field parameter. If the field is in the field dictionary, only those attributes that differ from the field identified by FieldID are saved in the attribute set.

The AttrID parameter identifies the attribute set to update. If AttrID is NULL, the attribute set associated with the FieldID parameter is updated.

See Also

FindFieldID (see page 11651)

GetAttrID (see page 11652)

FindAttrID (see page 11650)

NewAttr (see page 11656)

UpdateField (see page 11660)

AssociateAttr (see page 11647)

The Data Dictionary

63.1.21 DRIntf.UpdateConstraints Function

Replaces all imported constraints for a dataset with the current constraints in the Data Dictionary.

Pascal

```
procedure UpdateConstraints(Dataset: TDBDataset; Constraints: TCheckConstraints);
```

C++

```
UpdateConstraints(TDBDataset Dataset, TCheckConstraints Constraints);
```

File

DRIntf

Description

Call UpdateConstraints to ensure that the constraints for a dataset match the current values stored in the Data Dictionary. Specify the dataset to which the constraints apply as the DataSet parameter. Specify the TCheckConstraints object that represents that dataset's constraints as the Constraints parameter.

UpdateConstraints removes all constraints with a non-empty ImportedConstraint property and replaces them with new constraints that match the values in the Data Dictionary.

UpdateConstraints is equivalent to calling "Read From Dictionary" in the Constraints property editor.

See Also

HasConstraints (see page 11654)

UpdateDataset (see page 11659)

TDataSet

TCheckConstraint

The Data Dictionary

63.1.22 DRIntf.UpdateDataset Function

Updates a dataset and all its field components to match the Data Dictionary.

Pascal

```
procedure UpdateDataset(Dataset: TDBDataset);
```

C++

```
UpdateDataset(TDBDataset Dataset);
```

File

DRIntf

Description

Call UpdateDataset to ensure that a dataset reflects the current settings in the data dictionary. Specify the dataset to update as the Dataset parameter.

If Dataset is a TTable, UpdateDataset updates the Constraints property so that all imported constraints match the Data

Dictionary and updates all the field components to match the associated attribute sets.

If Dataset is a TQuery or TStoredProc, UpdateDataset updates the field components to match the associated attribute sets and parses the query to obtain any field constraints included in the constituent tables.

Calling UpdateDataset is equivalent to using "Retrieve attributes" for all fields in the fields editor followed by "Read From Dictionary" in the Constraints property editor.

See Also

UpdateConstraints (see page 11659)

UpdateField (see page 11660)

TDataset

TDataset

The Data Dictionary

63.1.23 DRIntf.UpdateField Function

Updates a field component to match the associated attribute set in the Data Dictionary.

Pascal

```
procedure UpdateField(Field: TField; const FieldID: TFieldID; const AttrID: TAttrID);
```

C++

```
UpdateField(TField Field, const TFieldID FieldID, const TAttrID AttrID);
```

File

DRIntf

Description

Call UpdateField to change the properties of a field component so that they match the associated attribute set in the Data Dictionary.

Specify the field to update as the Field parameter.

Specify the ID of the corresponding field in the Data Dictionary using the FieldID parameter. This value can be obtained using FindFieldID (see page 11651).

Specify the attribute set that acts as a source using the AttrID parameter.

See Also

UpdateDataset (see page 11659)

AssociateAttr (see page 11647)

UpdateAttr (see page 11658)

FindFieldID (see page 11651)

FindAttrID (see page 11650)

The Data Dictionary

63.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TID (see page 11661)	This is record DRIntf.TID.

63.2.1 DRIntf.TID Record

Pascal

```
TID = packed record
  Reserved: array[1..6] of Byte;
end;
```

C++

```
struct TID {
  array[1..6] of Byte Reserved;
};
```

File

DRIntf

Description

This is record DRIntf.TID.

63.3 Types

The following table lists types in this documentation.

Types

Name	Description
TAttrID (see page 11662)	This is type DRIntf.TAttrID.
TDatabaseID (see page 11662)	This is type DRIntf.TDatabaseID.
TFieldID (see page 11663)	This is type DRIntf.TFieldID.
TTableID (see page 11663)	This is type DRIntf.TTableID.

63.3.1 DRIntf.TAttrID Type

Pascal

```
TAttrID = type TID;
```

C++

```
type TID TAttrID;
```

File

DRIntf

Description

This is type DRIntf.TAttrID.

63.3.2 DRIntf.TDatabaseID Type

Pascal

```
TDatabaseID = type TID;
```

C++

```
type TID TDatabaseID;
```

File

DRIntf

Description

This is type DRIntf.TDatabaseID.

63.3.3 DRIntf.TFieldID Type

Pascal

```
TFieldID = type TID;
```

C++

```
type TID TFieldID;
```

File

DRIntf

Description

This is type DRIntf.TFieldID.

63.3.4 DRIntf.TTableID Type

Pascal

```
TTableID = type TID;
```

C++

```
type TID TTableID;
```

File

DRIntf

Description

This is type DRIntf.TTableID.

63.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
NullAttrID (see page 11664)	This is constant DRIntf.NullAttrID.
NullDatabaseID (see page 11664)	This is constant DRIntf.NullDatabaseID.
NullFieldID (see page 11665)	This is constant DRIntf.NullFieldID.
NullID (see page 11665)	This is constant DRIntf.NullID.
NullTableID (see page 11665)	This is constant DRIntf.NullTableID.

63.4.1 DRIntf.NullAttrID Constant

Pascal

```
NullAttrID: TAttrID = ();
```

C++

```
TAttrID NullAttrID = ();
```

File

DRIntf

Description

This is constant DRIntf.NullAttrID.

63.4.2 DRIntf.NullDatabaseID Constant

Pascal

```
NullDatabaseID: TDatabaseID = ();
```

C++

```
TDatabaseID NullDatabaseID = ();
```

File

DRIntf

Description

This is constant DRIntf.NullDatabaseID.

63.4.3 DRIntf.NullFieldID Constant

Pascal

```
NullFieldID: TFieldID = ();
```

C++

```
TFieldID NullFieldID = ();
```

File

DRIntf

Description

This is constant DRIntf.NullFieldID.

63.4.4 DRIntf.NullID Constant

Pascal

```
NullID: TID = ();
```

C++

```
TID NullID = ();
```

File

DRIntf

Description

This is constant DRIntf.NullID.

63.4.5 DRIntf.NullTableID Constant

Pascal

```
NullTableID: TTableID = ();
```

C++

```
TTableID NullTableID = ();
```

File

DRIntf

Description

This is constant DRIntf.NullTableID.

64 DSIntf Namespace

This is namespace DSIntf.

Topics

Name	Description
Interfaces (see page 11673)	The following table lists interfaces in this documentation.
Functions (see page 11702)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 11705)	The following table lists structs, records, enums in this documentation.
Types (see page 11713)	The following table lists types in this documentation.
Constants (see page 11723)	The following table lists constants in this documentation.

Constants

Name	Description
BLANK_NOTCHANGED (see page 11727)	This is constant DSIntf.BLANK_NOTCHANGED.
BLANK_NULL (see page 11727)	This is constant DSIntf.BLANK_NULL.
BlobTypeMap (see page 11727)	This is constant DSIntf.BlobTypeMap.
CLSID_DSBase (see page 11728)	This is constant DSIntf.CLSID_DSBase.
CLSID_DSCursor (see page 11728)	This is constant DSIntf.CLSID_DSCursor.
CLSID_DSWriter (see page 11728)	This is constant DSIntf.CLSID_DSWriter.
DataTypeMap (see page 11729)	This is constant DSIntf.DataTypeMap.
DBERR_ABORTED (see page 11729)	This is constant DSIntf.DBERR_ABORTED.
DBERR_BLOBNOTFETCHED (see page 11729)	This is constant DSIntf.DBERR_BLOBNOTFETCHED.
DBERR_BOF (see page 11730)	This is constant DSIntf.DBERR_BOF.
DBERR_CANCELLED (see page 11730)	This is constant DSIntf.DBERR_CANCELLED.
DBERR_CANNOTAPPEND (see page 11730)	This is constant DSIntf.DBERR_CANNOTAPPEND.
DBERR_DATAPACKETMISMATCH (see page 11731)	This is constant DSIntf.DBERR_DATAPACKETMISMATCH.
DBERR_DELTAISEMPTY (see page 11731)	This is constant DSIntf.DBERR_DELTAISEMPTY.
DBERR_DETAILSNOTFETCHED (see page 11731)	This is constant DSIntf.DBERR_DETAILSNOTFETCHED.
DBERR_EOF (see page 11732)	This is constant DSIntf.DBERR_EOF.
DBERR_FLYAWAY_WRONGORDER (see page 11732)	This is constant DSIntf.DBERR_FLYAWAY_WRONGORDER.
DBERR_LINKFIELDSNOTUNIQUE (see page 11732)	This is constant DSIntf.DBERR_LINKFIELDSNOTUNIQUE.
DBERR_NEWERVERSIONREQ (see page 11733)	This is constant DSIntf.DBERR_NEWERVERSIONREQ.
DBERR_NOCASCADEDUPDATES (see page 11733)	This is constant DSIntf.DBERR_NOCASCADEDUPDATES.
DBERR_NOMASTERRECORD (see page 11733)	This is constant DSIntf.DBERR_NOMASTERRECORD.
DBERR_NOMETADATA (see page 11734)	This is constant DSIntf.DBERR_NOMETADATA.
DBERR_NONE (see page 11734)	This is constant DSIntf.DBERR_NONE.

DBERR_NOSUCHINDEX (see page 11734)	This is constant DSIntf.DBERR_NOSUCHINDEX.
DBERR_NOTHINGTOUNDO (see page 11735)	This is constant DSIntf.DBERR_NOTHINGTOUNDO.
dsArrayFldType (see page 11735)	This is constant DSIntf.dsArrayFldType.
dsCASCADEDELETES (see page 11735)	This is constant DSIntf.dsCASCADEDELETES.
dsCASCADEUPDATES (see page 11736)	This is constant DSIntf.dsCASCADEUPDATES.
dscbrABORT (see page 11736)	This is constant DSIntf.dscbrABORT.
dscbrAPPLY (see page 11736)	This is constant DSIntf.dscbrAPPLY.
dscbrCANCEL (see page 11737)	This is constant DSIntf.dscbrCANCEL.
dscbrCORRECT (see page 11737)	This is constant DSIntf.dscbrCORRECT.
dscbrIGNORE (see page 11737)	This is constant DSIntf.dscbrIGNORE.
dscbrMERGE (see page 11738)	This is constant DSIntf.dscbrMERGE.
dscbrREFRESH (see page 11738)	This is constant DSIntf.dscbrREFRESH.
dscbrSKIP (see page 11738)	This is constant DSIntf.dscbrSKIP.
dsCompArrayFldType (see page 11739)	This is constant DSIntf.dsCompArrayFldType.
dsDELAYEDBIT (see page 11739)	This is constant DSIntf.dsDELAYEDBIT.
dsDetUpd (see page 11739)	This is constant DSIntf.dsDetUpd.
dsEmbeddedFldType (see page 11740)	This is constant DSIntf.dsEmbeddedFldType.
dsfldADT (see page 11740)	This is constant DSIntf.dsfldADT.
dsfldARRAY (see page 11740)	This is constant DSIntf.dsfldARRAY.
dsfldBCD (see page 11741)	This is constant DSIntf.dsfldBCD.
dsfldBOOL (see page 11741)	This is constant DSIntf.dsfldBOOL.
dsfldBYTES (see page 11741)	This is constant DSIntf.dsfldBYTES.
dsfldDATE (see page 11742)	This is constant DSIntf.dsfldDATE.
dsfldDATETIME (see page 11742)	This is constant DSIntf.dsfldDATETIME.
dsfldEMBEDDEDTBL (see page 11742)	This is constant DSIntf.dsfldEMBEDDEDTBL.
dsfldFLOATIEEE (see page 11743)	This is constant DSIntf.dsfldFLOATIEEE.
dsfldFMTBCD (see page 11743)	This is constant DSIntf.dsfldFMTBCD.
dsfldINT (see page 11743)	This is constant DSIntf.dsfldINT.
dsfldREF (see page 11744)	This is constant DSIntf.dsfldREF.
dsfldTIME (see page 11744)	This is constant DSIntf.dsfldTIME.
dsfldTIMESTAMP (see page 11744)	This is constant DSIntf.dsfldTIMESTAMP.
dsfldUINT (see page 11745)	This is constant DSIntf.dsfldUINT.
dsfldUNICODE (see page 11745)	This is constant DSIntf.dsfldUNICODE.
dsfldUNKNOWN (see page 11745)	This is constant DSIntf.dsfldUNKNOWN.
dsfldZSTRING (see page 11746)	This is constant DSIntf.dsfldZSTRING.
dsIncInDelta (see page 11746)	This is constant DSIntf.dsIncInDelta.
dsIsNotVisible (see page 11746)	This is constant DSIntf.dsIsNotVisible.
dsIsVisible (see page 11747)	This is constant DSIntf.dsIsVisible.
dskeyCASEINSENSITIVE (see page 11747)	This is constant DSIntf.dskeyCASEINSENSITIVE.
dskeyDESCENDING (see page 11747)	This is constant DSIntf.dskeyDESCENDING.
dsPseudoFldType (see page 11748)	This is constant DSIntf.dsPseudoFldType.
dsRecDeleted (see page 11748)	This is constant DSIntf.dsRecDeleted.
dsRecModified (see page 11748)	This is constant DSIntf.dsRecModified.
dsRecNew (see page 11749)	This is constant DSIntf.dsRecNew.
dsRecOrg (see page 11749)	This is constant DSIntf.dsRecOrg.
dsRecUnmodified (see page 11749)	This is constant DSIntf.dsRecUnmodified.
dsSizeBitsLen (see page 11750)	This is constant DSIntf.dsSizeBitsLen.
dsSizeBitsMask (see page 11750)	This is constant DSIntf.dsSizeBitsMask.
dsTypeBitsMask (see page 11750)	This is constant DSIntf.dsTypeBitsMask.
dsUnused (see page 11751)	This is constant DSIntf.dsUnused.
dsVaryingFldType (see page 11751)	This is constant DSIntf.dsVaryingFldType.
ERRBASE_ALC (see page 11751)	This is constant DSIntf.ERRBASE_ALC.
ERRCAT_ALC (see page 11752)	This is constant DSIntf.ERRCAT_ALC.
ERRCODE_ABORTED (see page 11752)	This is constant DSIntf.ERRCODE_ABORTED.

ERRCODE_BLOBNOTFETCHED (see page 11752)	This is constant DSIntf.ERRCODE_BLOBNOTFETCHED.
ERRCODE_CANCELLED (see page 11753)	This is constant DSIntf.ERRCODE_CANCELLED.
ERRCODE_CANNOTAPPEND (see page 11753)	This is constant DSIntf.ERRCODE_CANNOTAPPEND.
ERRCODE_DATAPACKETMISMATCH (see page 11753)	This is constant DSIntf.ERRCODE_DATAPACKETMISMATCH.
ERRCODE_DELTAISEMPTY (see page 11754)	This is constant DSIntf.ERRCODE_DELTAISEMPTY.
ERRCODE_DETAILSNOTFETCHED (see page 11754)	This is constant DSIntf.ERRCODE_DETAILSNOTFETCHED.
ERRCODE_FLYAWAY_WRONGORDER (see page 11754)	This is constant DSIntf.ERRCODE_FLYAWAY_WRONGORDER.
ERRCODE_LINKFIELDSNOTUNIQUE (see page 11755)	This is constant DSIntf.ERRCODE_LINKFIELDSNOTUNIQUE.
ERRCODE_NEWERVERSIONREQ (see page 11755)	This is constant DSIntf.ERRCODE_NEWERVERSIONREQ.
ERRCODE_NOCASCADEDUPDATES (see page 11755)	This is constant DSIntf.ERRCODE_NOCASCADEDUPDATES.
ERRCODE_NOMASTERRECORD (see page 11756)	This is constant DSIntf.ERRCODE_NOMASTERRECORD.
ERRCODE_NOMETADATA (see page 11756)	This is constant DSIntf.ERRCODE_NOMETADATA.
ERRCODE_NOTHINGTOUNDO (see page 11756)	This is constant DSIntf.ERRCODE_NOTHINGTOUNDO.
FieldTypeMap (see page 11757)	This is constant DSIntf.FieldTypeMap.
fldADT (see page 11757)	This is constant DSIntf.fldADT.
fldARRAY (see page 11757)	This is constant DSIntf.fldARRAY.
fldAttrHIDDEN (see page 11758)	This is constant DSIntf.fldAttrHIDDEN.
fldAttrLINK (see page 11758)	This is constant DSIntf.fldAttrLINK.
fldAttrREADONLY (see page 11758)	This is constant DSIntf.fldAttrREADONLY.
fldAttrREQUIRED (see page 11759)	This is constant DSIntf.fldAttrREQUIRED.
fldBCD (see page 11759)	This is constant DSIntf.fldBCD.
fldBLOB (see page 11759)	This is constant DSIntf.fldBLOB.
fldBOOL (see page 11760)	This is constant DSIntf.fldBOOL.
fldBYTES (see page 11760)	This is constant DSIntf.fldBYTES.
fldCURSOR (see page 11760)	This is constant DSIntf.fldCURSOR.
fldDATE (see page 11761)	This is constant DSIntf.fldDATE.
fldDATETIME (see page 11761)	This is constant DSIntf.fldDATETIME.
fldFLOAT (see page 11761)	This is constant DSIntf.fldFLOAT.
fldFLOATIEEE (see page 11762)	This is constant DSIntf.fldFLOATIEEE.
fldFMTBCD (see page 11762)	This is constant DSIntf.fldFMTBCD.
fldINT16 (see page 11762)	This is constant DSIntf.fldINT16.
fldINT32 (see page 11763)	This is constant DSIntf.fldINT32.
fldINT64 (see page 11763)	This is constant DSIntf.fldINT64.
fldLOCKINFO (see page 11763)	This is constant DSIntf.fldLOCKINFO.
fldREF (see page 11764)	This is constant DSIntf.fldREF.
fldstACCOLEOBJ (see page 11764)	This is constant DSIntf.fldstACCOLEOBJ.
fldstAUTOINC (see page 11764)	This is constant DSIntf.fldstAUTOINC.
fldstBFILE (see page 11765)	This is constant DSIntf.fldstBFILE.
fldstBINARY (see page 11765)	This is constant DSIntf.fldstBINARY.
fldstDBSOLEOBJ (see page 11765)	This is constant DSIntf.fldstDBSOLEOBJ.
fldstFIXED (see page 11766)	This is constant DSIntf.fldstFIXED.
fldstFMTMEMO (see page 11766)	This is constant DSIntf.fldstFMTMEMO.
fldstGRAPHIC (see page 11766)	This is constant DSIntf.fldstGRAPHIC.
fldstGUID (see page 11767)	This is constant DSIntf.fldstGUID.
fldstHBINARY (see page 11767)	This is constant DSIntf.fldstHBINARY.
fldstHMEMO (see page 11767)	This is constant DSIntf.fldstHMEMO.
fldstMEMO (see page 11768)	This is constant DSIntf.fldstMEMO.
fldstMONEY (see page 11768)	This is constant DSIntf.fldstMONEY.
fldstOLEOBJ (see page 11768)	This is constant DSIntf.fldstOLEOBJ.
fldstORAINTERVAL (see page 11769)	This is constant DSIntf.fldstORAINTERVAL.
fldstORATIMESTAMP (see page 11769)	This is constant DSIntf.fldstORATIMESTAMP.
fldstPASSWORD (see page 11769)	This is constant DSIntf.fldstPASSWORD.
fldstReference (see page 11770)	This is constant DSIntf.fldstReference.
fldstTYPEDBINARY (see page 11770)	This is constant DSIntf.fldstTYPEDBINARY.

fldstWIDEMEMO (see page 11770)	This is constant DSIntf.fldstWIDEMEMO.
FldSubTypeMap (see page 11771)	This is constant DSIntf.FldSubTypeMap.
fldTABLE (see page 11771)	This is constant DSIntf.fldTABLE.
fldTIME (see page 11771)	This is constant DSIntf.fldTIME.
fldTIMESTAMP (see page 11772)	This is constant DSIntf.fldTIMESTAMP.
fldUINT16 (see page 11772)	This is constant DSIntf.fldUINT16.
fldUINT32 (see page 11772)	This is constant DSIntf.fldUINT32.
fldUINT64 (see page 11773)	This is constant DSIntf.fldUINT64.
fldUNICODE (see page 11773)	This is constant DSIntf.fldUNICODE.
fldUNKNOWN (see page 11773)	This is constant DSIntf.fldUNKNOWN.
fldVARBYTES (see page 11774)	This is constant DSIntf.fldVARBYTES.
fldWIDESTRING (see page 11774)	This is constant DSIntf.fldWIDESTRING.
fldZSTRING (see page 11774)	This is constant DSIntf.fldZSTRING.
MAXKEYFIELDS (see page 11775)	This is constant DSIntf.MAXKEYFIELDS.
MAXLOGFLDTYPES (see page 11775)	This is constant DSIntf.MAXLOGFLDTYPES.
mdALLOWLINKCHANGE (see page 11775)	This is constant DSIntf.mdALLOWLINKCHANGE.
mdCASCADEDEL (see page 11776)	This is constant DSIntf.mdCASCADEDEL.
mdCASCADEMOD (see page 11776)	This is constant DSIntf.mdCASCADEMOD.
PACKETVERSION_1 (see page 11776)	This is constant DSIntf.PACKETVERSION_1.
PACKETVERSION_2 (see page 11777)	This is constant DSIntf.PACKETVERSION_2.
PACKETVERSION_3 (see page 11777)	This is constant DSIntf.PACKETVERSION_3.
szAUTOINCVALUE (see page 11777)	This is constant DSIntf.szAUTOINCVALUE.
szBDEDEFCL (see page 11778)	This is constant DSIntf.szBDEDEFCL.
szBDEDEFX (see page 11778)	This is constant DSIntf.szBDEDEFX.
szBDEDOMCL (see page 11778)	This is constant DSIntf.szBDEDOMCL.
szBDEDOMX (see page 11779)	This is constant DSIntf.szBDEDOMX.
szBDERECCL (see page 11779)	This is constant DSIntf.szBDERECCL.
szBDERECX (see page 11779)	This is constant DSIntf.szBDERECX.
szCHANGE_LOG (see page 11780)	This is constant DSIntf.szCHANGE_LOG.
szCHANGEINDEX (see page 11780)	This is constant DSIntf.szCHANGEINDEX.
szCONSTRAINTS (see page 11780)	This is constant DSIntf.szCONSTRAINTS.
szDATASET_CONTEXT (see page 11781)	This is constant DSIntf.szDATASET_CONTEXT.
szDATASET_DELTA (see page 11781)	This is constant DSIntf.szDATASET_DELTA.
szDECIMALS (see page 11781)	This is constant DSIntf.szDECIMALS.
szDEFAULT_ORDER (see page 11782)	This is constant DSIntf.szDEFAULT_ORDER.
szDISABLE_DELETES (see page 11782)	This is constant DSIntf.szDISABLE_DELETES.
szDISABLE_EDITS (see page 11782)	This is constant DSIntf.szDISABLE_EDITS.
szDISABLE_INSERTS (see page 11783)	This is constant DSIntf.szDISABLE_INSERTS.
szELEMENTS (see page 11783)	This is constant DSIntf.szELEMENTS.
szFIELDNAME (see page 11783)	This is constant DSIntf.szFIELDNAME.
szFIELDPROPS (see page 11784)	This is constant DSIntf.szFIELDPROPS.
szLCID (see page 11784)	This is constant DSIntf.szLCID.
szMAXVALUE (see page 11784)	This is constant DSIntf.szMAXVALUE.
szMD_FIELDLINKS (see page 11785)	This is constant DSIntf.szMD_FIELDLINKS.
szMD_SEMANTICS (see page 11785)	This is constant DSIntf.szMD_SEMANTICS.
szMINVALUE (see page 11785)	This is constant DSIntf.szMINVALUE.
szNO_RESET_CALL (see page 11786)	This is constant DSIntf.szNO_RESET_CALL.
szORIGIN (see page 11786)	This is constant DSIntf.szORIGIN.
szPRIMARY_KEY (see page 11786)	This is constant DSIntf.szPRIMARY_KEY.
szPROVFLAGS (see page 11787)	This is constant DSIntf.szPROVFLAGS.
szREADONLY (see page 11787)	This is constant DSIntf.szREADONLY.
szSERVER_COL (see page 11787)	This is constant DSIntf.szSERVER_COL.
szSERVERCALC (see page 11788)	This is constant DSIntf.szSERVERCALC.
szstACCOLEOBJ (see page 11788)	This is constant DSIntf.szstACCOLEOBJ.

szstADTNESTEDTABLE (see page 11788)	This is constant DSIntf.szstADTNESTEDTABLE.
szstAUTOINC (see page 11789)	This is constant DSIntf.szstAUTOINC.
szstBINARY (see page 11789)	This is constant DSIntf.szstBINARY.
szstDBSOLEOBJ (see page 11789)	This is constant DSIntf.szstDBSOLEOBJ.
szstFIXEDCHAR (see page 11790)	This is constant DSIntf.szstFIXEDCHAR.
szstFMTMEMO (see page 11790)	This is constant DSIntf.szstFMTMEMO.
szstGRAPHIC (see page 11790)	This is constant DSIntf.szstGRAPHIC.
szstGUID (see page 11791)	This is constant DSIntf.szstGUID.
szstHBINARY (see page 11791)	This is constant DSIntf.szstHBINARY.
szstHMEMO (see page 11791)	This is constant DSIntf.szstHMEMO.
szstMEMO (see page 11792)	This is constant DSIntf.szstMEMO.
szstMONEY (see page 11792)	This is constant DSIntf.szstMONEY.
szstOLEOBJ (see page 11792)	This is constant DSIntf.szstOLEOBJ.
szstREFNESTEDTABLE (see page 11793)	This is constant DSIntf.szstREFNESTEDTABLE.
szstTYPEDBINARY (see page 11793)	This is constant DSIntf.szstTYPEDBINARY.
szstWideMEMO (see page 11793)	This is constant DSIntf.szstWideMEMO.
szSUBTYPE (see page 11794)	This is constant DSIntf.szSUBTYPE.
szTABLE_NAME (see page 11794)	This is constant DSIntf.szTABLE_NAME.
szTYPENAME (see page 11794)	This is constant DSIntf.szTYPENAME.
szUNIQUE_KEY (see page 11795)	This is constant DSIntf.szUNIQUE_KEY.
szUPDATEMODE (see page 11795)	This is constant DSIntf.szUPDATEMODE.
szWIDTH (see page 11795)	This is constant DSIntf.szWIDTH.
xmlOFF (see page 11796)	This is constant DSIntf.xmlOFF.
xmlON (see page 11796)	This is constant DSIntf.xmlON.
xmlUNTYPED (see page 11796)	This is constant DSIntf.xmlUNTYPED.
xmlUTF8 (see page 11797)	This is constant DSIntf.xmlUTF8.
xmlXMLDATATYPED (see page 11797)	This is constant DSIntf.xmlXMLDATATYPED.
xmlXMLSCHEMA (see page 11797)	This is constant DSIntf.xmlXMLSCHEMA.
xmlXMLUTF8 (see page 11798)	This is constant DSIntf.xmlXMLUTF8.

Functions

	Name	Description
	CreateDbClientObject (see page 11702)	This is function DSIntf.CreateDbClientObject.
	DataPacketSize (see page 11702)	This is function DSIntf.DataPacketSize.
	DataPacketToVariant (see page 11703)	This is function DSIntf.DataPacketToVariant.
	FreeDataPacket (see page 11703)	This is function DSIntf.FreeDataPacket.
	RegisterMidasLib (see page 11703)	This is function DSIntf.RegisterMidasLib.
	StringToVariantArray (see page 11704)	This is function DSIntf.StringToVariantArray.
	VariantArrayToString (see page 11704)	This is function DSIntf.VariantArrayToString.
	VarToDataPacket (see page 11704)	This is function DSIntf.VarToDataPacket.

Interfaces

	Name	Description
	IDSBase (see page 11673)	This is class DSIntf.IDSBase.
	IDSCursor (see page 11682)	This is class DSIntf.IDSCursor.
	IDSWriter (see page 11697)	This is class DSIntf.IDSWriter.

Structs, Records, Enums

	Name	Description
	AGGSTATE (see page 11705)	This is record DSIntf.AGGSTATE.
	AGGVALUE (see page 11706)	This is record DSIntf.AGGVALUE.
	CURProp (see page 11706)	This is record DSIntf.CURProp.
	DBSearchCond (see page 11707)	This is record DSIntf.DBSearchCond.
	DSBOOKMRK (see page 11707)	This is record DSIntf.DSBOOKMRK.
	DSFLDDesc (see page 11708)	This is record DSIntf.DSFLDDesc.

	DSIDXDesc (see page 11708)	This is record DSIntf.DSIDXDesc.
	DSProp (see page 11709)	This is record DSIntf.DSProp.
	DSProps (see page 11710)	This is record DSIntf.DSProps.
	GROUPSTATE (see page 11711)	This is record DSIntf.GROUPSTATE.
	TDSDataPacketFldDesc (see page 11711)	This is record DSIntf.TDSDataPacketFldDesc.
	TPcktAttrArea (see page 11712)	This is record DSIntf.TPcktAttrArea.
	TPcktFldStatus (see page 11712)	This is record DSIntf.TPcktFldStatus.

Types

Name	Description
DBResult (see page 11713)	This is type DSIntf.DBResult.
DSAttr (see page 11714)	This is type DSIntf.DSAttr.
dsCBRType (see page 11714)	This is type DSIntf.dsCBRType.
DSKEY (see page 11714)	This is type DSIntf.DSKEY.
DSKEYBOOL (see page 11715)	This is type DSIntf.DSKEYBOOL.
hDSAggregate (see page 11715)	This is type DSIntf.hDSAggregate.
hDSFilter (see page 11715)	This is type DSIntf.hDSFilter.
MIDASNAME (see page 11716)	This is type DSIntf.MIDASNAME.
MIDASPATH (see page 11716)	This is type DSIntf.MIDASPATH.
pDSAttr (see page 11716)	This is type DSIntf.pDSAttr.
pDSBOOKMRK (see page 11717)	This is type DSIntf.pDSBOOKMRK.
pdsCBRType (see page 11717)	This is type DSIntf.pdsCBRType.
PDSDataPacketFldDesc (see page 11717)	This is type DSIntf.PDSDataPacketFldDesc.
pDSFLDDesc (see page 11718)	This is type DSIntf.pDSFLDDesc.
pDSIDXDesc (see page 11718)	This is type DSIntf.pDSIDXDesc.
pDSProps (see page 11718)	This is type DSIntf.pDSProps.
pfCHANGECallBack (see page 11719)	This is type DSIntf.pfCHANGECallBack.
pfDSCalcField (see page 11719)	This is type DSIntf.pfDSCalcField.
pfDSFilter (see page 11719)	This is type DSIntf.pfDSFilter.
pfDSReconcile (see page 11720)	This is type DSIntf.pfDSReconcile.
pfDSReconcile_MD (see page 11720)	This is type DSIntf.pfDSReconcile_MD.
pfFLDComp (see page 11720)	This is type DSIntf.pfFLDComp.
phDSAggregate (see page 11721)	This is type DSIntf.phDSAggregate.
phDSFilter (see page 11721)	This is type DSIntf.phDSFilter.
PLongBool (see page 11721)	This is type DSIntf.PLongBool.
pSAVEPOINT (see page 11722)	This is type DSIntf.pSAVEPOINT.
SAVEPOINT (see page 11722)	This is type DSIntf.SAVEPOINT.
TDataPacket (see page 11722)	This is type DSIntf.TDataPacket.

64.1 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IDSBase (see page 11673)	This is class DSIntf.IDSBase.
	IDSCursor (see page 11682)	This is class DSIntf.IDSCursor.
	IDSWriter (see page 11697)	This is class DSIntf.IDSWriter.

64.1.1 IDSBase Interface

Class Hierarchy

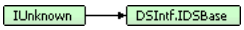

Pascal

```
IDSBase = interface(IUnknown);
```

C++

```
__interface IDSBase : public IUnknown;
```

File

DSIntf

Description

This is class DSIntf.IDSBase.

64.1.1.1 IDSBase Members

The following tables list the members exposed by IDSBase.

Public Methods

	Name	Description
	AcceptChanges (see page 11675)	This is AcceptChanges, a member of class IDSBase.
	AddField (see page 11675)	This is AddField, a member of class IDSBase.
	AddOptParameter (see page 11675)	This is AddOptParameter, a member of class IDSBase.
	AppendData (see page 11675)	This is AppendData, a member of class IDSBase.
	Clone (see page 11676)	This is Clone, a member of class IDSBase.
	Create (see page 11676)	This is Create, a member of class IDSBase.
	CreateIndex (see page 11676)	This is CreateIndex, a member of class IDSBase.
	DropOptParameter (see page 11676)	This is DropOptParameter, a member of class IDSBase.
	FldCmp (see page 11677)	This is FldCmp, a member of class IDSBase.
	GetDelta (see page 11677)	This is GetDelta, a member of class IDSBase.
	GetEmbeddedDS (see page 11677)	This is GetEmbeddedDS, a member of class IDSBase.
	GetErrorString (see page 11677)	This is GetErrorString, a member of class IDSBase.
	GetFieldDescs (see page 11677)	This is GetFieldDescs, a member of class IDSBase.

	GetIndexDescs (see page 11678)	This is GetIndexDescs, a member of class IDSBase.
	GetOptParameter (see page 11678)	This is GetOptParameter, a member of class IDSBase.
	GetProp (see page 11678)	This is GetProp, a member of class IDSBase.
	GetProps (see page 11678)	This is GetProps, a member of class IDSBase.
	MakeEmbedded (see page 11679)	This is MakeEmbedded, a member of class IDSBase.
	PutBlank (see page 11679)	This is PutBlank, a member of class IDSBase.
	Reconcile (see page 11679)	This is Reconcile, a member of class IDSBase.
	Reconcile_MD (see page 11679)	This is Reconcile_MD, a member of class IDSBase.
	Refresh (see page 11680)	This is Refresh, a member of class IDSBase.
	RefreshRecords (see page 11680)	This is RefreshRecords, a member of class IDSBase.
	ReleaseBlobs (see page 11680)	This is ReleaseBlobs, a member of class IDSBase.
	RemoveIndex (see page 11680)	This is RemoveIndex, a member of class IDSBase.
	Reserved1 (see page 11681)	This is Reserved1, a member of class IDSBase.
	Reserved2 (see page 11681)	This is Reserved2, a member of class IDSBase.
	Reserved3 (see page 11681)	This is Reserved3, a member of class IDSBase.
	Reset (see page 11681)	This is Reset, a member of class IDSBase.
	RollBack (see page 11681)	This is RollBack, a member of class IDSBase.
	SetFieldCalculation (see page 11682)	This is SetFieldCalculation, a member of class IDSBase.
	SetProp (see page 11682)	This is SetProp, a member of class IDSBase.
	StreamDS (see page 11682)	This is StreamDS, a member of class IDSBase.

64.1.1.2 IDSBase Methods

The methods of the IDSBase class are listed here.

Public Methods

	Name	Description
	AcceptChanges (see page 11675)	This is AcceptChanges, a member of class IDSBase.
	AddField (see page 11675)	This is AddField, a member of class IDSBase.
	AddOptParameter (see page 11675)	This is AddOptParameter, a member of class IDSBase.
	AppendData (see page 11675)	This is AppendData, a member of class IDSBase.
	Clone (see page 11676)	This is Clone, a member of class IDSBase.
	Create (see page 11676)	This is Create, a member of class IDSBase.
	CreateIndex (see page 11676)	This is CreateIndex, a member of class IDSBase.
	DropOptParameter (see page 11676)	This is DropOptParameter, a member of class IDSBase.
	FldCmp (see page 11677)	This is FldCmp, a member of class IDSBase.
	GetDelta (see page 11677)	This is GetDelta, a member of class IDSBase.
	GetEmbeddedDS (see page 11677)	This is GetEmbeddedDS, a member of class IDSBase.
	GetErrorString (see page 11677)	This is GetErrorString, a member of class IDSBase.
	GetFieldDescs (see page 11677)	This is GetFieldDescs, a member of class IDSBase.
	GetIndexDescs (see page 11678)	This is GetIndexDescs, a member of class IDSBase.
	GetOptParameter (see page 11678)	This is GetOptParameter, a member of class IDSBase.
	GetProp (see page 11678)	This is GetProp, a member of class IDSBase.
	GetProps (see page 11678)	This is GetProps, a member of class IDSBase.
	MakeEmbedded (see page 11679)	This is MakeEmbedded, a member of class IDSBase.
	PutBlank (see page 11679)	This is PutBlank, a member of class IDSBase.
	Reconcile (see page 11679)	This is Reconcile, a member of class IDSBase.
	Reconcile_MD (see page 11679)	This is Reconcile_MD, a member of class IDSBase.
	Refresh (see page 11680)	This is Refresh, a member of class IDSBase.
	RefreshRecords (see page 11680)	This is RefreshRecords, a member of class IDSBase.
	ReleaseBlobs (see page 11680)	This is ReleaseBlobs, a member of class IDSBase.
	RemoveIndex (see page 11680)	This is RemoveIndex, a member of class IDSBase.
	Reserved1 (see page 11681)	This is Reserved1, a member of class IDSBase.

	Reserved2 (see page 11681)	This is Reserved2, a member of class IDSBase.
	Reserved3 (see page 11681)	This is Reserved3, a member of class IDSBase.
	Reset (see page 11681)	This is Reset, a member of class IDSBase.
	RollBack (see page 11681)	This is RollBack, a member of class IDSBase.
	SetFieldCalculation (see page 11682)	This is SetFieldCalculation, a member of class IDSBase.
	SetProp (see page 11682)	This is SetProp, a member of class IDSBase.
	StreamDS (see page 11682)	This is StreamDS, a member of class IDSBase.

64.1.1.2.1 IDSBase.AcceptChanges Method

Pascal

```
function AcceptChanges: DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult AcceptChanges();
```

Description

This is AcceptChanges, a member of class IDSBase.

64.1.1.2.2 IDSBase.AddField Method

Pascal

```
function AddField(pFldDes: pDSFLDDesc): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult AddField(pDSFLDDesc pFldDes);
```

Description

This is AddField, a member of class IDSBase.

64.1.1.2.3 IDSBase.AddOptParameter Method

Pascal

```
function AddOptParameter(iFldNo: LongWord; pszAttr: PChar; iType: LongWord; iLen: LongWord;
pValue: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult AddOptParameter(LongWord iFldNo, const char * pszAttr, LongWord
iType, LongWord iLen, void * pValue);
```

Description

This is AddOptParameter, a member of class IDSBase.

64.1.1.2.4 IDSBase.AppendData Method

Pascal

```
function AppendData(Packet: PSafeArray; bEof: LongBool): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult AppendData(PSafeArray Packet, LongBool bEof);
```

Description

This is AppendData, a member of class IDSBase.

64.1.1.2.5 IDSBase.Clone Method

Pascal

```
function Clone(iPType: LongWord; bRecurse: LongBool; bCloneOptParams: LongBool; var DataSet:
IDSBase): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult Clone(LongWord iPType, LongBool bRecurse, LongBool
bCloneOptParams, IDSBase DataSet);
```

Description

This is Clone, a member of class IDSBase.

64.1.1.2.6 IDSBase.Create Method

Pascal

```
function Create(iFields: LongWord; pFldDes: pDSFLDDesc; pszName: PChar): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult Create(LongWord iFields, pDSFLDDesc pFldDes, const char *
pszName);
```

Description

This is Create, a member of class IDSBase.

64.1.1.2.7 IDSBase.CreateIndex Method

Pascal

```
function CreateIndex(const IdxDesc: DSIDXDesc): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult CreateIndex(const DSIDXDesc IdxDesc);
```

Description

This is CreateIndex, a member of class IDSBase.

64.1.1.2.8 IDSBase.DropOptParameter Method

Pascal

```
function DropOptParameter(iFldNo: LongWord; pName: PChar): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult DropOptParameter(LongWord iFldNo, const char * pName);
```

Description

This is DropOptParameter, a member of class IDSBase.

64.1.1.2.9 IDSBase.FldCmp Method

Pascal

```
function FldCmp(iFldType: LongWord; pFld1: Pointer; pFld2: Pointer; iUnits1: LongWord;
iUnits2: LongWord): Integer; stdcall;
```

C++

```
__fastcall __stdcall int FldCmp(LongWord iFldType, void * pFld1, void * pFld2, LongWord
iUnits1, LongWord iUnits2);
```

Description

This is FldCmp, a member of class IDSBase.

64.1.1.2.10 IDSBase.GetDelta Method

Pascal

```
function GetDelta(out DsDelta: IDSBase): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetDelta(IDSBase DsDelta);
```

Description

This is GetDelta, a member of class IDSBase.

64.1.1.2.11 IDSBase.GetEmbeddedDS Method

Pascal

```
function GetEmbeddedDS(iFieldID: LongWord; out DsDet: IDSBase): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetEmbeddedDS(LongWord iFieldID, IDSBase DsDet);
```

Description

This is GetEmbeddedDS, a member of class IDSBase.

64.1.1.2.12 IDSBase.GetErrorString Method

Pascal

```
function GetErrorString(iErrCode: DBResult; pString: PChar): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetErrorString(DBResult iErrCode, const char * pString);
```

Description

This is GetErrorString, a member of class IDSBase.

64.1.1.2.13 IDSBase.GetFieldDescs Method

Pascal

```
function GetFieldDescs(Fields: pDSFLDDesc): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetFieldDescs(pDSFLDDesc Fields);
```

Description

This is GetFieldDescs, a member of class IDSBase.

64.1.1.2.14 IDSBase.GetIndexDescs Method

Pascal

```
function GetIndexDescs(p1: PDSIDXDesc): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetIndexDescs(PDSIDXDesc p1);
```

Description

This is GetIndexDescs, a member of class IDSBase.

64.1.1.2.15 IDSBase.GetOptParameter Method

Pascal

```
function GetOptParameter(iNo: LongWord; iFldNo: LongWord; var ppName: Pointer; var piType:
LongWord; var piLen: LongWord; var ppValue: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetOptParameter(LongWord iNo, LongWord iFldNo, void * ppName,
LongWord piType, LongWord piLen, void * ppValue);
```

Description

This is GetOptParameter, a member of class IDSBase.

64.1.1.2.16 IDSBase.GetProp Method

Pascal

```
function GetProp(eProp: DSProp; piPropValue: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetProp(DSProp eProp, void * piPropValue);
```

Description

This is GetProp, a member of class IDSBase.

64.1.1.2.17 IDSBase.GetProps Method

Pascal

```
function GetProps(var Prop: DSProps): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetProps(DSProps Prop);
```

Description

This is GetProps, a member of class IDSBase.

64.1.1.2.18 IDSBase.MakeEmbedded Method

Pascal

```
function MakeEmbedded(DsDet: IDSBase; iFieldsLink: LongWord; piFieldsM: PLongWord; piFieldsD:
PLongWord; pMasterFieldName: PChar; pDetailFieldName: PChar): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult MakeEmbedded(IDSBase DsDet, LongWord iFieldsLink, PLongWord
piFieldsM, PLongWord piFieldsD, const char * pMasterFieldName, const char * pDetailFieldName);
```

Description

This is MakeEmbedded, a member of class IDSBase.

64.1.1.2.19 IDSBase.PutBlank Method

Pascal

```
function PutBlank(pRecBuf: Pointer; iRecNo: LongWord; iFldNo: LongWord; iBlankValue:
LongWord): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult PutBlank(void * pRecBuf, LongWord iRecNo, LongWord iFldNo,
LongWord iBlankValue);
```

Description

This is PutBlank, a member of class IDSBase.

64.1.1.2.20 IDSBase.Reconcile Method

Pascal

```
function Reconcile(DeltaPacket: PSafeArray; ErrorPacket: PSafeArray; iClientData: LongWord;
pfReconcile: pfDSReconcile): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult Reconcile(PSafeArray DeltaPacket, PSafeArray ErrorPacket,
LongWord iClientData, pfDSReconcile pfReconcile);
```

Description

This is Reconcile, a member of class IDSBase.

64.1.1.2.21 IDSBase.Reconcile_MD Method

Pascal

```
function Reconcile_MD(pDsRec: IDSBase; pDeltaPacket: PSafeArray; pErrorPacket: PSafeArray;
iClientData: LongWord; pfReconcile_MD: pfDSReconcile_MD): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult Reconcile_MD(IDSBase pDsRec, PSafeArray pDeltaPacket, PSafeArray
pErrorPacket, LongWord iClientData, pfDSReconcile_MD pfReconcile_MD);
```

Description

This is Reconcile_MD, a member of class IDSBase.

64.1.1.2.22 IDSBase.Refresh Method

Pascal

```
function Refresh(NewPacket: PSafeArray; iClientData: LongWord; pfReconcile: pfDSReconcile):
DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult Refresh(PSafeArray NewPacket, LongWord iClientData,
pfDSReconcile pfReconcile);
```

Description

This is Refresh, a member of class IDSBase.

64.1.1.2.23 IDSBase.RefreshRecords Method

Pascal

```
function RefreshRecords(NewDataPacket: PSafeArray; iRecNo: LongWord; iClientData: LongWord;
pfReconcile: pfDSReconcile): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult RefreshRecords(PSafeArray NewDataPacket, LongWord iRecNo,
LongWord iClientData, pfDSReconcile pfReconcile);
```

Description

This is RefreshRecords, a member of class IDSBase.

64.1.1.2.24 IDSBase.ReleaseBlobs Method

Pascal

```
function ReleaseBlobs(iBlobId: LongWord): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult ReleaseBlobs(LongWord iBlobId);
```

Description

This is ReleaseBlobs, a member of class IDSBase.

64.1.1.2.25 IDSBase.RemoveIndex Method

Pascal

```
function RemoveIndex(pszName: PChar): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult RemoveIndex(const char * pszName);
```

Description

This is RemoveIndex, a member of class IDSBase.

64.1.1.2.26 IDSBase.Reserved1 Method

Pascal

```
function Reserved1(Input: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult Reserved1(void * Input);
```

Description

This is Reserved1, a member of class IDSBase.

64.1.1.2.27 IDSBase.Reserved2 Method

Pascal

```
function Reserved2(Input: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult Reserved2(void * Input);
```

Description

This is Reserved2, a member of class IDSBase.

64.1.1.2.28 IDSBase.Reserved3 Method

Pascal

```
function Reserved3(Input: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult Reserved3(void * Input);
```

Description

This is Reserved3, a member of class IDSBase.

64.1.1.2.29 IDSBase.Reset Method

Pascal

```
function Reset: DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult Reset();
```

Description

This is Reset, a member of class IDSBase.

64.1.1.2.30 IDSBase.RollBack Method

Pascal

```
function RollBack(iSavePoint: SAVEPOINT): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult RollBack(SAVEPOINT iSavePoint);
```

Description

This is RollBack, a member of class IDSBase.

64.1.1.2.31 IDSBase.SetFieldCalculation Method

Pascal

```
function SetFieldCalculation(iClientData: LongWord; pfCalc: pfDSCalcField): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult SetFieldCalculation(LongWord iClientData, pfDSCalcField pfCalc);
```

Description

This is SetFieldCalculation, a member of class IDSBase.

64.1.1.2.32 IDSBase.SetProp Method

Pascal

```
function SetProp(eProp: DSProp; iPropValue: LongWord): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult SetProp(DSProp eProp, LongWord iPropValue);
```

Description

This is SetProp, a member of class IDSBase.

64.1.1.2.33 IDSBase.StreamDS Method

Pascal

```
function StreamDS(out Packet: PSafeArray): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult StreamDS(PSafeArray Packet);
```

Description

This is StreamDS, a member of class IDSBase.

64.1.2 IDSCursor Interface

Class Hierarchy

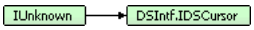

Pascal

```
IDSCursor = interface(IUnknown);
```

C++

```
__interface IDSCursor : public IUnknown;
```

File

DSIntf

Description

This is class DSIntf.IDSCursor.

64.1.2.1 IDSCursor Members

The following tables list the members exposed by IDSCursor.

Public Methods

	Name	Description
	AddAggregate (see page 11685)	This is AddAggregate, a member of class IDSCursor.
	AddFilter (see page 11685)	This is AddFilter, a member of class IDSCursor.
	AddFilterCallBack (see page 11685)	This is AddFilterCallBack, a member of class IDSCursor.
	CloneCursor (see page 11686)	This is CloneCursor, a member of class IDSCursor.
	CompareBookMarks (see page 11686)	This is CompareBookMarks, a member of class IDSCursor.
	DeleteRecord (see page 11686)	This is DeleteRecord, a member of class IDSCursor.
	DropAggregate (see page 11686)	This is DropAggregate, a member of class IDSCursor.
	DropFilter (see page 11687)	This is DropFilter, a member of class IDSCursor.
	DropRange (see page 11687)	This is DropRange, a member of class IDSCursor.
	ExtractKey (see page 11687)	This is ExtractKey, a member of class IDSCursor.
	GetAggregateDesc (see page 11687)	This is GetAggregateDesc, a member of class IDSCursor.
	GetAggregateValue (see page 11687)	This is GetAggregateValue, a member of class IDSCursor.
	GetBlob (see page 11688)	This is GetBlob, a member of class IDSCursor.
	GetBlobLen (see page 11688)	This is GetBlobLen, a member of class IDSCursor.
	GetCurrentBookMark (see page 11688)	This is GetCurrentBookMark, a member of class IDSCursor.
	GetCurrentRecord (see page 11688)	This is GetCurrentRecord, a member of class IDSCursor.
	GetCursorProps (see page 11689)	This is GetCursorProps, a member of class IDSCursor.
	GetField (see page 11689)	This is GetField, a member of class IDSCursor.
	GetFieldDescs (see page 11689)	This is GetFieldDescs, a member of class IDSCursor.
	GetIndexDescs (see page 11689)	This is GetIndexDescs, a member of class IDSCursor.
	GetProp (see page 11690)	This is GetProp, a member of class IDSCursor.
	GetRecordAttribute (see page 11690)	This is GetRecordAttribute, a member of class IDSCursor.
	GetRecordBlock (see page 11690)	This is GetRecordBlock, a member of class IDSCursor.
	GetRecordCount (see page 11690)	This is GetRecordCount, a member of class IDSCursor.
	GetRecordForKey (see page 11690)	This is GetRecordForKey, a member of class IDSCursor.
	GetRecordNumber (see page 11691)	This is GetRecordNumber, a member of class IDSCursor.
	GetRowRequestPacket (see page 11691)	This is GetRowRequestPacket, a member of class IDSCursor.
	GetSequenceNumber (see page 11691)	This is GetSequenceNumber, a member of class IDSCursor.
	GetSubGroupState (see page 11691)	This is GetSubGroupState, a member of class IDSCursor.
	InitCursor (see page 11692)	This is InitCursor, a member of class IDSCursor.
	InitRecord (see page 11692)	This is InitRecord, a member of class IDSCursor.
	InsertRecord (see page 11692)	This is InsertRecord, a member of class IDSCursor.
	LinkCursors (see page 11692)	This is LinkCursors, a member of class IDSCursor.
	LocateWithFilter (see page 11693)	This is LocateWithFilter, a member of class IDSCursor.
	ModifyRecord (see page 11693)	This is ModifyRecord, a member of class IDSCursor.
	MoveRelative (see page 11693)	This is MoveRelative, a member of class IDSCursor.
	MoveToBOF (see page 11693)	This is MoveToBOF, a member of class IDSCursor.
	MoveToBookMark (see page 11693)	This is MoveToBookMark, a member of class IDSCursor.
	MoveToEOF (see page 11694)	This is MoveToEOF, a member of class IDSCursor.
	MoveToKey (see page 11694)	This is MoveToKey, a member of class IDSCursor.
	MoveToNextSubGroup (see page 11694)	This is MoveToNextSubGroup, a member of class IDSCursor.
	MoveToSeqNo (see page 11694)	This is MoveToSeqNo, a member of class IDSCursor.
	PutBlob (see page 11695)	This is PutBlob, a member of class IDSCursor.

	PutField (see page 11695)	This is PutField, a member of class IDSCursor.
	RefreshRecord (see page 11695)	This is RefreshRecord, a member of class IDSCursor.
	ResyncWithMaster (see page 11695)	This is ResyncWithMaster, a member of class IDSCursor.
	RevertRecord (see page 11695)	This is RevertRecord, a member of class IDSCursor.
	SetNotifyCallBack (see page 11696)	This is SetNotifyCallBack, a member of class IDSCursor.
	SetProp (see page 11696)	This is SetProp, a member of class IDSCursor.
	SetRange (see page 11696)	This is SetRange, a member of class IDSCursor.
	SortOnFields (see page 11696)	This is SortOnFields, a member of class IDSCursor.
	UndoLastChange (see page 11697)	This is UndoLastChange, a member of class IDSCursor.
	UseIndexOrder (see page 11697)	This is UseIndexOrder, a member of class IDSCursor.
	VerifyField (see page 11697)	This is VerifyField, a member of class IDSCursor.

64.1.2.2 IDSCursor Methods

The methods of the IDSCursor class are listed here.

Public Methods

	Name	Description
	AddAggregate (see page 11685)	This is AddAggregate, a member of class IDSCursor.
	AddFilter (see page 11685)	This is AddFilter, a member of class IDSCursor.
	AddFilterCallBack (see page 11685)	This is AddFilterCallBack, a member of class IDSCursor.
	CloneCursor (see page 11686)	This is CloneCursor, a member of class IDSCursor.
	CompareBookMarks (see page 11686)	This is CompareBookMarks, a member of class IDSCursor.
	DeleteRecord (see page 11686)	This is DeleteRecord, a member of class IDSCursor.
	DropAggregate (see page 11686)	This is DropAggregate, a member of class IDSCursor.
	DropFilter (see page 11687)	This is DropFilter, a member of class IDSCursor.
	DropRange (see page 11687)	This is DropRange, a member of class IDSCursor.
	ExtractKey (see page 11687)	This is ExtractKey, a member of class IDSCursor.
	GetAggregateDesc (see page 11687)	This is GetAggregateDesc, a member of class IDSCursor.
	GetAggregateValue (see page 11687)	This is GetAggregateValue, a member of class IDSCursor.
	GetBlob (see page 11688)	This is GetBlob, a member of class IDSCursor.
	GetBlobLen (see page 11688)	This is GetBlobLen, a member of class IDSCursor.
	GetCurrentBookMark (see page 11688)	This is GetCurrentBookMark, a member of class IDSCursor.
	GetCurrentRecord (see page 11688)	This is GetCurrentRecord, a member of class IDSCursor.
	GetCursorProps (see page 11689)	This is GetCursorProps, a member of class IDSCursor.
	GetField (see page 11689)	This is GetField, a member of class IDSCursor.
	GetFieldDescs (see page 11689)	This is GetFieldDescs, a member of class IDSCursor.
	GetIndexDescs (see page 11689)	This is GetIndexDescs, a member of class IDSCursor.
	GetProp (see page 11690)	This is GetProp, a member of class IDSCursor.
	GetRecordAttribute (see page 11690)	This is GetRecordAttribute, a member of class IDSCursor.
	GetRecordBlock (see page 11690)	This is GetRecordBlock, a member of class IDSCursor.
	GetRecordCount (see page 11690)	This is GetRecordCount, a member of class IDSCursor.
	GetRecordForKey (see page 11690)	This is GetRecordForKey, a member of class IDSCursor.
	GetRecordNumber (see page 11691)	This is GetRecordNumber, a member of class IDSCursor.
	GetRowRequestPacket (see page 11691)	This is GetRowRequestPacket, a member of class IDSCursor.
	GetSequenceNumber (see page 11691)	This is GetSequenceNumber, a member of class IDSCursor.
	GetSubGroupState (see page 11691)	This is GetSubGroupState, a member of class IDSCursor.
	InitCursor (see page 11692)	This is InitCursor, a member of class IDSCursor.
	InitRecord (see page 11692)	This is InitRecord, a member of class IDSCursor.
	InsertRecord (see page 11692)	This is InsertRecord, a member of class IDSCursor.
	LinkCursors (see page 11692)	This is LinkCursors, a member of class IDSCursor.
	LocateWithFilter (see page 11693)	This is LocateWithFilter, a member of class IDSCursor.
	ModifyRecord (see page 11693)	This is ModifyRecord, a member of class IDSCursor.

	MoveRelative (see page 11693)	This is MoveRelative, a member of class IDSCursor.
	MoveToBOF (see page 11693)	This is MoveToBOF, a member of class IDSCursor.
	MoveToBookMark (see page 11693)	This is MoveToBookMark, a member of class IDSCursor.
	MoveToEOF (see page 11694)	This is MoveToEOF, a member of class IDSCursor.
	MoveToKey (see page 11694)	This is MoveToKey, a member of class IDSCursor.
	MoveToNextSubGroup (see page 11694)	This is MoveToNextSubGroup, a member of class IDSCursor.
	MoveToSeqNo (see page 11694)	This is MoveToSeqNo, a member of class IDSCursor.
	PutBlob (see page 11695)	This is PutBlob, a member of class IDSCursor.
	PutField (see page 11695)	This is PutField, a member of class IDSCursor.
	RefreshRecord (see page 11695)	This is RefreshRecord, a member of class IDSCursor.
	ResyncWithMaster (see page 11695)	This is ResyncWithMaster, a member of class IDSCursor.
	RevertRecord (see page 11695)	This is RevertRecord, a member of class IDSCursor.
	SetNotifyCallBack (see page 11696)	This is SetNotifyCallBack, a member of class IDSCursor.
	SetProp (see page 11696)	This is SetProp, a member of class IDSCursor.
	SetRange (see page 11696)	This is SetRange, a member of class IDSCursor.
	SortOnFields (see page 11696)	This is SortOnFields, a member of class IDSCursor.
	UndoLastChange (see page 11697)	This is UndoLastChange, a member of class IDSCursor.
	UseIndexOrder (see page 11697)	This is UseIndexOrder, a member of class IDSCursor.
	VerifyField (see page 11697)	This is VerifyField, a member of class IDSCursor.

64.1.2.2.1 IDSCursor.AddAggregate Method

Pascal

```
function AddAggregate(iFlds: LongWord; iCanLen: LongWord; pCanExpr: Pointer; var hAgg:
hDSAggregate): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult AddAggregate(LongWord iFlds, LongWord iCanLen, void * pCanExpr,
hDSAggregate hAgg);
```

Description

This is AddAggregate, a member of class IDSCursor.

64.1.2.2.2 IDSCursor.AddFilter Method

Pascal

```
function AddFilter(pcanExpr: Pointer; iLen: LongWord; var hFilter: hDSFilter): DBResult;
stdcall;
```

C++

```
__fastcall __stdcall DBResult AddFilter(void * pcanExpr, LongWord iLen, hDSFilter hFilter);
```

Description

This is AddFilter, a member of class IDSCursor.

64.1.2.2.3 IDSCursor.AddFilterCallBack Method

Pascal

```
function AddFilterCallBack(iClientData: LongWord; pfFilter: pfDSFilter; var hFilter:
hDSFilter): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult AddFilterCallBack(LongWord iClientData, pfDSFilter pfFilter,
hDSFilter hFilter);
```

Description

This is AddFilterCallBack, a member of class IDSCursor.

64.1.2.2.4 IDSCursor.CloneCursor Method

Pascal

```
function CloneCursor(Cursor: IDSCursor): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult CloneCursor(IDSCursor Cursor);
```

Description

This is CloneCursor, a member of class IDSCursor.

64.1.2.2.5 IDSCursor.CompareBookMarks Method

Pascal

```
function CompareBookMarks(pBookMark1: Pointer; pBookMark2: Pointer; var iCmp: Integer):
DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult CompareBookMarks(void * pBookMark1, void * pBookMark2, int iCmp);
```

Description

This is CompareBookMarks, a member of class IDSCursor.

64.1.2.2.6 IDSCursor.DeleteRecord Method

Pascal

```
function DeleteRecord: DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult DeleteRecord();
```

Description

This is DeleteRecord, a member of class IDSCursor.

64.1.2.2.7 IDSCursor.DropAggregate Method

Pascal

```
function DropAggregate(hAgg: hDSAggregate): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult DropAggregate(hDSAggregate hAgg);
```

Description

This is DropAggregate, a member of class IDSCursor.

64.1.2.2.8 IDSCursor.DropFilter Method

Pascal

```
function DropFilter(hFilter: hDSFilter): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult DropFilter(hDSFilter hFilter);
```

Description

This is DropFilter, a member of class IDSCursor.

64.1.2.2.9 IDSCursor.DropRange Method

Pascal

```
function DropRange: DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult DropRange();
```

Description

This is DropRange, a member of class IDSCursor.

64.1.2.2.10 IDSCursor.ExtractKey Method

Pascal

```
function ExtractKey(pRecBuf: Pointer; pKeyBuf: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult ExtractKey(void * pRecBuf, void * pKeyBuf);
```

Description

This is ExtractKey, a member of class IDSCursor.

64.1.2.2.11 IDSCursor.GetAggregateDesc Method

Pascal

```
function GetAggregateDesc(hAgg: hDSAggregate; var Desc: DSFLDDesc): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetAggregateDesc(hDSAggregate hAgg, DSFLDDesc Desc);
```

Description

This is GetAggregateDesc, a member of class IDSCursor.

64.1.2.2.12 IDSCursor.GetAggregateValue Method

Pascal

```
function GetAggregateValue(hAgg: hDSAggregate; pValue: Pointer; var bBlank: LongBool):
DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetAggregateValue(hDSAggregate hAgg, void * pValue, LongBool
bBlank);
```

Description

This is GetAggregateValue, a member of class IDSCursor.

64.1.2.2.13 IDSCursor.GetBlob Method

Pascal

```
function GetBlob(pRecBuf: Pointer; iFieldNo: LongWord; iOffSet: LongWord; pBuf: Pointer; var
iLength: LongWord): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetBlob(void * pRecBuf, LongWord iFieldNo, LongWord iOffSet,
void * pBuf, LongWord iLength);
```

Description

This is GetBlob, a member of class IDSCursor.

64.1.2.2.14 IDSCursor.GetBlobLen Method

Pascal

```
function GetBlobLen(pRecBuf: Pointer; iFieldNo: LongWord; var iLength: LongWord): DBResult;
stdcall;
```

C++

```
__fastcall __stdcall DBResult GetBlobLen(void * pRecBuf, LongWord iFieldNo, LongWord iLength);
```

Description

This is GetBlobLen, a member of class IDSCursor.

64.1.2.2.15 IDSCursor.GetCurrentBookMark Method

Pascal

```
function GetCurrentBookMark(pBookMark: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetCurrentBookMark(void * pBookMark);
```

Description

This is GetCurrentBookMark, a member of class IDSCursor.

64.1.2.2.16 IDSCursor.GetCurrentRecord Method

Pascal

```
function GetCurrentRecord(pRecBuf: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetCurrentRecord(void * pRecBuf);
```

Description

This is GetCurrentRecord, a member of class IDSCursor.

64.1.2.2.17 IDSCursor.GetCursorProps Method

Pascal

```
function GetCursorProps(var p1: DSProps): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetCursorProps(DSProps p1);
```

Description

This is GetCursorProps, a member of class IDSCursor.

64.1.2.2.18 IDSCursor.GetField Method

Pascal

```
function GetField(pRecBuf: Pointer; iFieldNo: LongWord; pFldBuf: Pointer; var bBlank:
LongBool): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetField(void * pRecBuf, LongWord iFieldNo, void * pFldBuf,
LongBool bBlank);
```

Description

This is GetField, a member of class IDSCursor.

64.1.2.2.19 IDSCursor.GetFieldDescs Method

Pascal

```
function GetFieldDescs(p1: pDSFLDDesc): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetFieldDescs(pDSFLDDesc p1);
```

Description

This is GetFieldDescs, a member of class IDSCursor.

64.1.2.2.20 IDSCursor.GetIndexDescs Method

Pascal

```
function GetIndexDescs(bCurrentOnly: LongBool; var IdxDesc: DSIDXDesc): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetIndexDescs(LongBool bCurrentOnly, DSIDXDesc IdxDesc);
```

Description

This is GetIndexDescs, a member of class IDSCursor.

64.1.2.2.21 IDSCursor.GetProp Method

Pascal

```
function GetProp(eProp: CURProp; piPropValue: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetProp(CURProp eProp, void * piPropValue);
```

Description

This is GetProp, a member of class IDSCursor.

64.1.2.2.22 IDSCursor.GetRecordAttribute Method

Pascal

```
function GetRecordAttribute(var Attr: DSAttr): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetRecordAttribute(DSAttr Attr);
```

Description

This is GetRecordAttribute, a member of class IDSCursor.

64.1.2.2.23 IDSCursor.GetRecordBlock Method

Pascal

```
function GetRecordBlock(piRecs: PLongWord; pRecBuf: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetRecordBlock(PLongWord piRecs, void * pRecBuf);
```

Description

This is GetRecordBlock, a member of class IDSCursor.

64.1.2.2.24 IDSCursor.GetRecordCount Method

Pascal

```
function GetRecordCount(var iRecs: Integer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetRecordCount(int iRecs);
```

Description

This is GetRecordCount, a member of class IDSCursor.

64.1.2.2.25 IDSCursor.GetRecordForKey Method

Pascal

```
function GetRecordForKey(iFields: LongWord; iPartLen: LongWord; pKey: Pointer; pRecBuf:
Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetRecordForKey(LongWord iFields, LongWord iPartLen, void *
pKey, void * pRecBuf);
```

Description

This is GetRecordForKey, a member of class IDSCursor.

64.1.2.2.26 IDSCursor.GetRecordNumber Method

Pascal

```
function GetRecordNumber(var iRecNo: LongWord): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetRecordNumber(LongWord iRecNo);
```

Description

This is GetRecordNumber, a member of class IDSCursor.

64.1.2.2.27 IDSCursor.GetRowRequestPacket Method

Pascal

```
function GetRowRequestPacket(bKeysOnly: LongBool; bFetchAllBlobs: LongBool; bFetchAllDetails:
LongBool; bInclMetaData: LongBool; out Packet: PSafeArray): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetRowRequestPacket(LongBool bKeysOnly, LongBool bFetchAllBlobs,
LongBool bFetchAllDetails, LongBool bInclMetaData, PSafeArray Packet);
```

Description

This is GetRowRequestPacket, a member of class IDSCursor.

64.1.2.2.28 IDSCursor.GetSequenceNumber Method

Pascal

```
function GetSequenceNumber(var iSeq: LongWord): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetSequenceNumber(LongWord iSeq);
```

Description

This is GetSequenceNumber, a member of class IDSCursor.

64.1.2.2.29 IDSCursor.GetSubGroupState Method

Pascal

```
function GetSubGroupState(iFields: LongWord; var iSubGroupState: GROUPSTATE): DBResult;
stdcall;
```

C++

```
__fastcall __stdcall DBResult GetSubGroupState(LongWord iFields, GROUPSTATE iSubGroupState);
```

Description

This is GetSubGroupState, a member of class IDSCursor.

64.1.2.2.30 IDSCursor.InitCursor Method

Pascal

```
function InitCursor(DataSet: IDSBase): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult InitCursor(IDSBase DataSet);
```

Description

This is InitCursor, a member of class IDSCursor.

64.1.2.2.31 IDSCursor.InitRecord Method

Pascal

```
function InitRecord(pRecBuf: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult InitRecord(void * pRecBuf);
```

Description

This is InitRecord, a member of class IDSCursor.

64.1.2.2.32 IDSCursor.InsertRecord Method

Pascal

```
function InsertRecord(pRecBuf: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult InsertRecord(void * pRecBuf);
```

Description

This is InsertRecord, a member of class IDSCursor.

64.1.2.2.33 IDSCursor.LinkCursors Method

Pascal

```
function LinkCursors(iFieldsM: LongWord; piFieldsM: PLongWord; piFieldsD: PLongWord; hCurDet:
IDSCursor): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult LinkCursors(LongWord iFieldsM, PLongWord piFieldsM, PLongWord
piFieldsD, IDSCursor hCurDet);
```

Description

This is LinkCursors, a member of class IDSCursor.

64.1.2.2.34 IDSCursor.LocateWithFilter Method

Pascal

```
function LocateWithFilter(pCanExpr: Pointer; iLen: LongWord): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult LocateWithFilter(void * pCanExpr, LongWord iLen);
```

Description

This is LocateWithFilter, a member of class IDSCursor.

64.1.2.2.35 IDSCursor.ModifyRecord Method

Pascal

```
function ModifyRecord(pRecBuf: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult ModifyRecord(void * pRecBuf);
```

Description

This is ModifyRecord, a member of class IDSCursor.

64.1.2.2.36 IDSCursor.MoveRelative Method

Pascal

```
function MoveRelative(i: Integer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult MoveRelative(int i);
```

Description

This is MoveRelative, a member of class IDSCursor.

64.1.2.2.37 IDSCursor.MoveToBOF Method

Pascal

```
function MoveToBOF: DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult MoveToBOF();
```

Description

This is MoveToBOF, a member of class IDSCursor.

64.1.2.2.38 IDSCursor.MoveToBookMark Method

Pascal

```
function MoveToBookMark(pBookMark: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult MoveToBookMark(void * pBookMark);
```

Description

This is MoveToBookMark, a member of class IDSCursor.

64.1.2.2.39 IDSCursor.MoveToEOF Method

Pascal

```
function MoveToEOF: DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult MoveToEOF();
```

Description

This is MoveToEOF, a member of class IDSCursor.

64.1.2.2.40 IDSCursor.MoveToKey Method

Pascal

```
function MoveToKey(SearchCond: DBSearchCond; iFields: LongWord; iPartLen: LongWord; pRecBuf:
Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult MoveToKey(DBSearchCond SearchCond, LongWord iFields, LongWord
iPartLen, void * pRecBuf);
```

Description

This is MoveToKey, a member of class IDSCursor.

64.1.2.2.41 IDSCursor.MoveToNextSubGroup Method

Pascal

```
function MoveToNextSubGroup(iFields: LongWord): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult MoveToNextSubGroup(LongWord iFields);
```

Description

This is MoveToNextSubGroup, a member of class IDSCursor.

64.1.2.2.42 IDSCursor.MoveToSeqNo Method

Pascal

```
function MoveToSeqNo(i: LongWord): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult MoveToSeqNo(LongWord i);
```

Description

This is MoveToSeqNo, a member of class IDSCursor.

64.1.2.2.43 IDSCursor.PutBlob Method

Pascal

```
function PutBlob(pRecBuf: Pointer; iFieldNo: LongWord; iOffSet: LongWord; pBuf: Pointer;
iLength: LongWord): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult PutBlob(void * pRecBuf, LongWord iFieldNo, LongWord iOffSet,
void * pBuf, LongWord iLength);
```

Description

This is PutBlob, a member of class IDSCursor.

64.1.2.2.44 IDSCursor.PutField Method

Pascal

```
function PutField(pRecBuf: Pointer; iFieldNo: LongWord; pFldBuf: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult PutField(void * pRecBuf, LongWord iFieldNo, void * pFldBuf);
```

Description

This is PutField, a member of class IDSCursor.

64.1.2.2.45 IDSCursor.RefreshRecord Method

Pascal

```
function RefreshRecord(Packet: PSafeArray): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult RefreshRecord(PSafeArray Packet);
```

Description

This is RefreshRecord, a member of class IDSCursor.

64.1.2.2.46 IDSCursor.ResyncWithMaster Method

Pascal

```
function ResyncWithMaster: DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult ResyncWithMaster();
```

Description

This is ResyncWithMaster, a member of class IDSCursor.

64.1.2.2.47 IDSCursor.RevertRecord Method

Pascal

```
function RevertRecord: DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult RevertRecord();
```

Description

This is RevertRecord, a member of class IDSCursor.

64.1.2.2.48 IDSCursor.SetNotifyCallBack Method

Pascal

```
function SetNotifyCallBack(iClientData: LongWord; pfCallBack: pfCHANGECallBack): DBResult;
stdcall;
```

C++

```
__fastcall __stdcall DBResult SetNotifyCallBack(LongWord iClientData, pfCHANGECallBack
pfCallBack);
```

Description

This is SetNotifyCallBack, a member of class IDSCursor.

64.1.2.2.49 IDSCursor.SetProp Method

Pascal

```
function SetProp(eProp: CURProp; iPropValue: LongWord): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult SetProp(CURProp eProp, LongWord iPropValue);
```

Description

This is SetProp, a member of class IDSCursor.

64.1.2.2.50 IDSCursor.SetRange Method

Pascal

```
function SetRange(iFields: LongWord; pKey1: Pointer; bKey1Incl: LongBool; pKey2: Pointer;
bKey2Incl: LongBool): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult SetRange(LongWord iFields, void * pKey1, LongBool bKey1Incl,
void * pKey2, LongBool bKey2Incl);
```

Description

This is SetRange, a member of class IDSCursor.

64.1.2.2.51 IDSCursor.SortOnFields Method

Pascal

```
function SortOnFields(iFields: LongWord; piFields: PLongWord; pDescending: PLongBool;
pCaseInsensitive: PLongBool): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult SortOnFields(LongWord iFields, PLongWord piFields, PLongBool
pDescending, PLongBool pCaseInsensitive);
```

Description

This is SortOnFields, a member of class IDSCursor.

64.1.2.2.52 IDSCursor.UndoLastChange Method

Pascal

```
function UndoLastChange(bFollowChange: LongBool): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult UndoLastChange(LongBool bFollowChange);
```

Description

This is UndoLastChange, a member of class IDSCursor.

64.1.2.2.53 IDSCursor.UseIndexOrder Method

Pascal

```
function UseIndexOrder(pszName: PChar): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult UseIndexOrder(const char * pszName);
```

Description

This is UseIndexOrder, a member of class IDSCursor.

64.1.2.2.54 IDSCursor.VerifyField Method

Pascal

```
function VerifyField(iFieldNo: LongWord; pFldBuf: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult VerifyField(LongWord iFieldNo, void * pFldBuf);
```

Description

This is VerifyField, a member of class IDSCursor.

64.1.3 IDSWriter Interface

Class Hierarchy

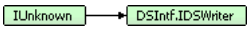

Pascal

```
IDSWriter = interface(IUnknown);
```

C++

```
__interface IDSWriter : public IUnknown;
```

File

DSIntf

Description

This is class DSIntf.IDSWriter.

64.1.3.1 IDSWriter Members

The following tables list the members exposed by IDSWriter.

Public Methods

	Name	Description
	AddAttribute (see page 11698)	This is AddAttribute, a member of class IDSWriter.
	AddColumnDesc (see page 11699)	This is AddColumnDesc, a member of class IDSWriter.
	EndOfNestedRows (see page 11699)	This is EndOfNestedRows, a member of class IDSWriter.
	GetColumnCount (see page 11699)	This is GetColumnCount, a member of class IDSWriter.
	GetColumnDescs (see page 11699)	This is GetColumnDescs, a member of class IDSWriter.
	GetDataPacket (see page 11700)	This is GetDataPacket, a member of class IDSWriter.
	GetErrorString (see page 11700)	This is GetErrorString, a member of class IDSWriter.
	GetSize (see page 11700)	This is GetSize, a member of class IDSWriter.
	GetXMLMode (see page 11700)	This is GetXMLMode, a member of class IDSWriter.
	Init (see page 11700)	This is Init, a member of class IDSWriter.
	Init_Sequential (see page 11701)	This is Init_Sequential, a member of class IDSWriter.
	PutField (see page 11701)	This is PutField, a member of class IDSWriter.
	Reset (see page 11701)	This is Reset, a member of class IDSWriter.
	SetXMLMode (see page 11701)	This is SetXMLMode, a member of class IDSWriter.

64.1.3.2 IDSWriter Methods

The methods of the IDSWriter class are listed here.

Public Methods

	Name	Description
	AddAttribute (see page 11698)	This is AddAttribute, a member of class IDSWriter.
	AddColumnDesc (see page 11699)	This is AddColumnDesc, a member of class IDSWriter.
	EndOfNestedRows (see page 11699)	This is EndOfNestedRows, a member of class IDSWriter.
	GetColumnCount (see page 11699)	This is GetColumnCount, a member of class IDSWriter.
	GetColumnDescs (see page 11699)	This is GetColumnDescs, a member of class IDSWriter.
	GetDataPacket (see page 11700)	This is GetDataPacket, a member of class IDSWriter.
	GetErrorString (see page 11700)	This is GetErrorString, a member of class IDSWriter.
	GetSize (see page 11700)	This is GetSize, a member of class IDSWriter.
	GetXMLMode (see page 11700)	This is GetXMLMode, a member of class IDSWriter.
	Init (see page 11700)	This is Init, a member of class IDSWriter.
	Init_Sequential (see page 11701)	This is Init_Sequential, a member of class IDSWriter.
	PutField (see page 11701)	This is PutField, a member of class IDSWriter.
	Reset (see page 11701)	This is Reset, a member of class IDSWriter.
	SetXMLMode (see page 11701)	This is SetXMLMode, a member of class IDSWriter.

64.1.3.2.1 IDSWriter.AddAttribute Method

Pascal

```
function AddAttribute(AttrArea: TPcktAttrArea; Attr: PChar; AttrType: LongWord; Len: LongWord;
Value: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult AddAttribute(TPcktAttrArea AttrArea, const char * Attr, LongWord
AttrType, LongWord Len, void * Value);
```

Description

This is AddAttribute, a member of class IDSWriter.

64.1.3.2.2 IDSWriter.AddColumnDesc Method

Pascal

```
function AddColumnDesc(const FldDes: TDSDataPacketFldDesc): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult AddColumnDesc(const TDSDataPacketFldDesc FldDes);
```

Description

This is AddColumnDesc, a member of class IDSWriter.

64.1.3.2.3 IDSWriter.EndOfNestedRows Method

Pascal

```
function EndOfNestedRows: DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult EndOfNestedRows();
```

Description

This is EndOfNestedRows, a member of class IDSWriter.

64.1.3.2.4 IDSWriter.GetColumnCount Method

Pascal

```
function GetColumnCount(var Count: LongWord): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetColumnCount(LongWord Count);
```

Description

This is GetColumnCount, a member of class IDSWriter.

64.1.3.2.5 IDSWriter.GetColumnDescs Method

Pascal

```
function GetColumnDescs(FieldDes: PDSDataPacketFldDesc): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetColumnDescs(PDSDataPacketFldDesc FieldDes);
```

Description

This is GetColumnDescs, a member of class IDSWriter.

64.1.3.2.6 IDSWriter.GetDataPacket Method

Pascal

```
function GetDataPacket(var SA: PSafeArray): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetDataPacket(PSafeArray SA);
```

Description

This is GetDataPacket, a member of class IDSWriter.

64.1.3.2.7 IDSWriter.GetErrorString Method

Pascal

```
function GetErrorString(iErrCode: DBResult; pString: PChar): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetErrorString(DBResult iErrCode, const char * pString);
```

Description

This is GetErrorString, a member of class IDSWriter.

64.1.3.2.8 IDSWriter.GetSize Method

Pascal

```
function GetSize(var DataPacketSize: LongWord): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult GetSize(LongWord DataPacketSize);
```

Description

This is GetSize, a member of class IDSWriter.

64.1.3.2.9 IDSWriter.GetXMLMode Method

Pascal

```
function GetXMLMode: LongWord; stdcall;
```

C++

```
__fastcall __stdcall LongWord GetXMLMode();
```

Description

This is GetXMLMode, a member of class IDSWriter.

64.1.3.2.10 IDSWriter.Init Method

Pascal

```
function Init(Version: LongWord; Columns: Word; FieldDesc: PDSDataPacketFldDesc): DBResult;
stdcall;
```

C++

```
__fastcall __stdcall DBResult Init(LongWord Version, Word Columns, PDSDataPacketFldDesc
FieldDesc);
```

Description

This is Init, a member of class IDSWriter.

64.1.3.2.11 IDSWriter.Init_Sequential Method

Pascal

```
function Init_Sequential(Version: LongWord; Columns: Word): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult Init_Sequential(LongWord Version, Word Columns);
```

Description

This is Init_Sequential, a member of class IDSWriter.

64.1.3.2.12 IDSWriter.PutField Method

Pascal

```
function PutField(FldStatus: TPcktFldStatus; Len: LongWord; Src: Pointer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult PutField(TPcktFldStatus FldStatus, LongWord Len, void * Src);
```

Description

This is PutField, a member of class IDSWriter.

64.1.3.2.13 IDSWriter.Reset Method

Pascal

```
function Reset: DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult Reset();
```

Description

This is Reset, a member of class IDSWriter.

64.1.3.2.14 IDSWriter.SetXMLMode Method

Pascal

```
function SetXMLMode(iMode: Integer): DBResult; stdcall;
```

C++

```
__fastcall __stdcall DBResult SetXMLMode(int iMode);
```

Description

This is SetXMLMode, a member of class IDSWriter.

64.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CreateDbClientObject (see page 11702)	This is function DSIntf.CreateDbClientObject.
	DataPacketSize (see page 11702)	This is function DSIntf.DataPacketSize.
	DataPacketToVariant (see page 11703)	This is function DSIntf.DataPacketToVariant.
	FreeDataPacket (see page 11703)	This is function DSIntf.FreeDataPacket.
	RegisterMidasLib (see page 11703)	This is function DSIntf.RegisterMidasLib.
	StringToVariantArray (see page 11704)	This is function DSIntf.StringToVariantArray.
	VariantArrayToString (see page 11704)	This is function DSIntf.VariantArrayToString.
	VarToDataPacket (see page 11704)	This is function DSIntf.VarToDataPacket.

64.2.1 DSIntf.CreateDbClientObject Function

Pascal

```
procedure CreateDbClientObject(const CLSID: TGUID; const IID: TGUID; out Obj);
```

C++

```
CreateDbClientObject(const TGUID CLSID, const TGUID IID,  Obj);
```

File

DSIntf

Description

This is function DSIntf.CreateDbClientObject.

64.2.2 DSIntf.DataPacketSize Function

Pascal

```
function DataPacketSize(const DataPacket: TDataPacket): Integer;
```

C++

```
int DataPacketSize(const TDataPacket DataPacket);
```

File

DSIntf

Description

This is function DSIntf.DataPacketSize.

64.2.3 DSIntf.DataPacketToVariant Function

Pascal

```
procedure DataPacketToVariant(const DataPacket: TDataPacket; var V: OleVariant);
```

C++

```
DataPacketToVariant(const TDataPacket DataPacket, OleVariant V);
```

File

DSIntf

Description

This is function DSIntf.DataPacketToVariant.

64.2.4 DSIntf.FreeDataPacket Function

Pascal

```
procedure FreeDataPacket(var DataPacket: TDataPacket);
```

C++

```
FreeDataPacket(TDataPacket DataPacket);
```

File

DSIntf

Description

This is function DSIntf.FreeDataPacket.

64.2.5 DSIntf.RegisterMidasLib Function

Pascal

```
procedure RegisterMidasLib(GetClassProc: Pointer);
```

C++

```
RegisterMidasLib(void * GetClassProc);
```

File

DSIntf

Description

This is function DSIntf.RegisterMidasLib.

64.2.6 DSIntf.StringToVariantArray Function

Pascal

```
function StringToVariantArray(const S: string): OleVariant;
```

C++

```
OleVariant StringToVariantArray(const AnsiString S);
```

File

DSIntf

Description

This is function DSIntf.StringToVariantArray.

64.2.7 DSIntf.VariantArrayToString Function

Pascal

```
function VariantArrayToString(const V: OleVariant): string;
```

C++

```
AnsiString VariantArrayToString(const OleVariant V);
```

File

DSIntf

Description

This is function DSIntf.VariantArrayToString.

64.2.8 DSIntf.VarToDataPacket Function

Pascal

```
function VarToDataPacket(const V: OleVariant): TDataPacket;
```

C++

```
TDataPacket VarToDataPacket(const OleVariant V);
```

File

DSIntf

Description

This is function DSIntf.VarToDataPacket.

64.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	AGGSTATE (see page 11705)	This is record DSIntf.AGGSTATE.
	CURProp (see page 11706)	This is record DSIntf.CURProp.
	DBSearchCond (see page 11707)	This is record DSIntf.DBSearchCond.
	DSProp (see page 11709)	This is record DSIntf.DSProp.
	GROUPSTATE (see page 11711)	This is record DSIntf.GROUPSTATE.
	TPcktAttrArea (see page 11712)	This is record DSIntf.TPcktAttrArea.
	TPcktFldStatus (see page 11712)	This is record DSIntf.TPcktFldStatus.

Records

	Name	Description
	AGGVALUE (see page 11706)	This is record DSIntf.AGGVALUE.
	DSBOOKMRK (see page 11707)	This is record DSIntf.DSBOOKMRK.
	DSFLDDesc (see page 11708)	This is record DSIntf.DSFLDDesc.
	DSIDXDesc (see page 11708)	This is record DSIntf.DSIDXDesc.
	DSProps (see page 11710)	This is record DSIntf.DSProps.
	TDSDataPacketFldDesc (see page 11711)	This is record DSIntf.TDSDataPacketFldDesc.

64.3.1 DSIntf.AGGSTATE Enumeration

Pascal

```
AGGSTATE = (
  aggSTATEUNDEF,
  aggSTATEOK,
  aggSTATEBLANK,
  aggSTATEERROR
);
```

C++

```
enum AGGSTATE {
  aggSTATEUNDEF,
  aggSTATEOK,
  aggSTATEBLANK,
  aggSTATEERROR
};
```

File

DSIntf

Description

This is record DSIntf.AGGSTATE.

64.3.2 DSIntf.AGGVALUE Record

Pascal

```
AGGVALUE = packed record
  iCnt: Integer;
  eAggState: AGGSTATE;
  Value: record;
end;
```

C++

```
struct AGGVALUE {
  int iCnt;
  AGGSTATE eAggState;
  record Value;
};
```

File

DSIntf

Description

This is record DSIntf.AGGVALUE.

64.3.3 DSIntf.CURProp Enumeration

Pascal

```
CURProp = (
  curpropCONSTRAINT_ERROR_MESSAGE,
  curpropDSBASE,
  curpropSETCRACK,
  curpropGETORG_RECBUF
);
```

C++

```
enum CURProp {
  curpropCONSTRAINT_ERROR_MESSAGE,
  curpropDSBASE,
  curpropSETCRACK,
  curpropGETORG_RECBUF
};
```

File

DSIntf

Description

This is record DSIntf.CURProp.

64.3.4 DSIntf.DBSearchCond Enumeration

Pascal

```
DBSearchCond = (
  keySEARCHEQ,
  keySEARCHGT,
  keySEARCHGEQ
);
```

C++

```
enum DBSearchCond {
  keySEARCHEQ,
  keySEARCHGT,
  keySEARCHGEQ
};
```

File

DSIntf

Description

This is record DSIntf.DBSearchCond.

64.3.5 DSIntf.DSBOOKMRK Record

Pascal

```
DSBOOKMRK = packed record
  iPos: Integer;
  iState: Integer;
  iRecNo: Integer;
  iSeqNo: Integer;
  iOrderID: Integer;
end;
```

C++

```
struct DSBOOKMRK {
  int iPos;
  int iState;
  int iRecNo;
  int iSeqNo;
  int iOrderID;
};
```

File

DSIntf

Description

This is record DSIntf.DSBOOKMRK.

64.3.6 DSIntf.DSFLDDesc Record

Pascal

```
DSFLDDesc = packed record
  szName: MIDASNAME;
  iFldType: Integer;
  iFldSubType: Integer;
  iUnits1: Integer;
  iUnits2: Integer;
  iFldLen: Integer;
  iFldOffsInRec: Integer;
  iNullOffsInRec: Integer;
  iFieldID: Word;
  iFieldIDParent: Word;
  bCalculated: LongBool;
  iFldAttr: Integer;
  iOptParameters: Integer;
end;
```

C++

```
struct DSFLDDesc {
  MIDASNAME szName;
  int iFldType;
  int iFldSubType;
  int iUnits1;
  int iUnits2;
  int iFldLen;
  int iFldOffsInRec;
  int iNullOffsInRec;
  Word iFieldID;
  Word iFieldIDParent;
  LongBool bCalculated;
  int iFldAttr;
  int iOptParameters;
};
```

File

DSIntf

Description

This is record DSIntf.DSFLDDesc.

64.3.7 DSIntf.DSIDXDesc Record

Pascal

```
DSIDXDesc = packed record
  szName: MIDASNAME;
  iFields: Integer;
  iKeyFields: DSKEY;
  iKeyLen: Integer;
  bUnique: LongBool;
  bDescending: DSKEYBOOL;
  bCaseInsensitive: DSKEYBOOL;
end;
```

C++

```
struct DSIDXDesc {
  MIDASNAME szName;
  int iFields;
  DSKEY iKeyFields;
  int iKeyLen;
  LongBool bUnique;
  DSKEYBOOL bDescending;
  DSKEYBOOL bCaseInsensitive;
};
```

File

DSIntf

Description

This is record DSIntf.DSIDXDesc.

64.3.8 DSIntf.DSProp Enumeration

Pascal

```
DSProp = (
  dspropLOGCHANGES,
  dspropREADONLY,
  dspropNOOFCHANGES,
  dspropCONSTRAINTS_DISABLED,
  dspropDSISPARTIAL,
  dspropRECORDSINDS,
  dspropAUTOINC_DISABLED,
  dspropISDELTA,
  dspropDONTINCLMETADATA,
  dspropINCLBLOBSINDELTA,
  dspropGETSAVEPOINT,
  dspropCOMPRESSARRAYS,
  dspropMD_SEMANTICS,
  dspropFIELD_FULLNAME,
  dspropFIELDID_FORNAME,
  dspropFIELDID_FORPARENT,
  dspropCHANGEINDEX_VIEW,
  dspropGETUNIQUEINDEX,
  dspropREMOTE_UPDATEMODE,
  dspropXML_STREAMMODE,
  dspropDISABLEDELETES,
  dspropDISABLEINSERTS,
  dspropDISABLEEDITS,
  dspropDISABLESTRINGTRIM,
  dspropDATAHASCHANGED
);
```

C++

```
enum DSProp {
  dspropLOGCHANGES,
  dspropREADONLY,
  dspropNOOFCHANGES,
  dspropCONSTRAINTS_DISABLED,
  dspropDSISPARTIAL,
  dspropRECORDSINDS,
  dspropAUTOINC_DISABLED,
  dspropISDELTA,
  dspropDONTINCLMETADATA,
```

```
    dspropINCLBLOBSINDELTA,
    dspropGETSAVEPOINT,
    dspropCOMPRESSARRAYS,
    dspropMD_SEMANTICS,
    dspropFIELD_FULLNAME,
    dspropFIELDID_FORNAME,
    dspropFIELDID_FORPARENT,
    dspropCHANGEINDEX_VIEW,
    dspropGETUNIQUEINDEX,
    dspropREMOTE_UPDATEMODE,
    dspropXML_STREAMMODE,
    dspropDISABLEDELETES,
    dspropDISABLEINSERTS,
    dspropDISABLEEDITS,
    dspropDISABLESTRINGTRIM,
    dspropDATAHASCHANGED
  };
```

File

DSIntf

Description

This is record DSIntf.DSProp.

64.3.9 DSIntf.DSProps Record

Pascal

```
  DSProps = packed record
    szName: MIDASPATH;
    iFields: Integer;
    iRecBufSize: Integer;
    iBookMarkSize: Integer;
    bReadOnly: LongBool;
    iIndexes: Integer;
    iOptParams: Integer;
    bDelta: LongBool;
    iLCID: Integer;
    iUnused: packed array[0..7] of Integer;
  end;
```

C++

```
  struct DSProps {
    MIDASPATH szName;
    int iFields;
    int iRecBufSize;
    int iBookMarkSize;
    LongBool bReadOnly;
    int iIndexes;
    int iOptParams;
    LongBool bDelta;
    int iLCID;
    packed array[0..7] of Integer iUnused;
  };
```

File

DSIntf

Description

This is record DSIntf.DSProps.

64.3.10 DSIntf.GROUPSTATE Enumeration

Pascal

```
GROUPSTATE = (
  grSTATEMIDDLE,
  grSTATEFIRST,
  grSTATELAST,
  grSTATEFIRSTLAST
);
```

C++

```
enum GROUPSTATE {
  grSTATEMIDDLE,
  grSTATEFIRST,
  grSTATELAST,
  grSTATEFIRSTLAST
};
```

File

DSIntf

Description

This is record DSIntf.GROUPSTATE.

64.3.11 DSIntf.TDSDataPacketFldDesc Record

Pascal

```
TDSDataPacketFldDesc = packed record
  szFieldName: MIDASNAME;
  iFieldType: Integer;
  iAttributes: Word;
end;
```

C++

```
struct TDSDataPacketFldDesc {
  MIDASNAME szFieldName;
  int iFieldType;
  Word iAttributes;
};
```

File

DSIntf

Description

This is record DSIntf.TDSDataPacketFldDesc.

64.3.12 DSIntf.TPcktAttrArea Enumeration

Pascal

```
TPcktAttrArea = (
  fldAttrArea,
  pcktAttrArea
);
```

C++

```
enum TPcktAttrArea {
  fldAttrArea,
  pcktAttrArea
};
```

File

DSIntf

Description

This is record DSIntf.TPcktAttrArea.

64.3.13 DSIntf.TPcktFldStatus Enumeration

Pascal

```
TPcktFldStatus = (
  fldIsChanged,
  fldIsNull,
  fldIsUnChanged
);
```

C++

```
enum TPcktFldStatus {
  fldIsChanged,
  fldIsNull,
  fldIsUnChanged
};
```

File

DSIntf

Description

This is record DSIntf.TPcktFldStatus.

64.4 Types

The following table lists types in this documentation.

Types

Name	Description
DBResult (see page 11713)	This is type DSIntf.DBResult.
DSAttr (see page 11714)	This is type DSIntf.DSAttr.
dsCBRType (see page 11714)	This is type DSIntf.dsCBRType.
DSKEY (see page 11714)	This is type DSIntf.DSKEY.
DSKEYBOOL (see page 11715)	This is type DSIntf.DSKEYBOOL.
hDSAggregate (see page 11715)	This is type DSIntf.hDSAggregate.
hDSFilter (see page 11715)	This is type DSIntf.hDSFilter.
MIDASNAME (see page 11716)	This is type DSIntf.MIDASNAME.
MIDASPATH (see page 11716)	This is type DSIntf.MIDASPATH.
pDSAttr (see page 11716)	This is type DSIntf.pDSAttr.
pDSBOOKMRK (see page 11717)	This is type DSIntf.pDSBOOKMRK.
pdsCBRType (see page 11717)	This is type DSIntf.pdsCBRType.
PDSDataPacketFldDesc (see page 11717)	This is type DSIntf.PDSDataPacketFldDesc.
pDSFLDDesc (see page 11718)	This is type DSIntf.pDSFLDDesc.
pDSIDXDesc (see page 11718)	This is type DSIntf.pDSIDXDesc.
pDSProps (see page 11718)	This is type DSIntf.pDSProps.
pfCHANGECallBack (see page 11719)	This is type DSIntf.pfCHANGECallBack.
pfDSCalcField (see page 11719)	This is type DSIntf.pfDSCalcField.
pfDSFilter (see page 11719)	This is type DSIntf.pfDSFilter.
pfDSReconcile (see page 11720)	This is type DSIntf.pfDSReconcile.
pfDSReconcile_MD (see page 11720)	This is type DSIntf.pfDSReconcile_MD.
pfFLDComp (see page 11720)	This is type DSIntf.pfFLDComp.
phDSAggregate (see page 11721)	This is type DSIntf.phDSAggregate.
phDSFilter (see page 11721)	This is type DSIntf.phDSFilter.
PLongBool (see page 11721)	This is type DSIntf.PLongBool.
pSAVEPOINT (see page 11722)	This is type DSIntf.pSAVEPOINT.
SAVEPOINT (see page 11722)	This is type DSIntf.SAVEPOINT.
TDataPacket (see page 11722)	This is type DSIntf.TDataPacket.

64.4.1 DSIntf.DBResult Type

Pascal

```
DBResult = Word;
```

C++

```
Word DBResult;
```

File

DSIntf

Description

This is type DSIntf.DBResult.

64.4.2 DSIntf.DSAttr Type

Pascal

```
DSAttr = type Byte;
```

C++

```
type Byte DSAttr;
```

File

DSIntf

Description

This is type DSIntf.DSAttr.

64.4.3 DSIntf.dsCBRType Type

Pascal

```
dsCBRType = Integer;
```

C++

```
int dsCBRType;
```

File

DSIntf

Description

This is type DSIntf.dsCBRType.

64.4.4 DSIntf.DSKEY Type

Pascal

```
DSKEY = array[0..MAXKEYFIELDS-1] of Integer;
```

C++

```
array[0..MAXKEYFIELDS-1] of Integer DSKEY;
```

File

DSIntf

Description

This is type DSIntf.DSKEY.

64.4.5 DSIntf.DSKEYBOOL Type

Pascal

```
DSKEYBOOL = array[0..MAXKEYFIELDS-1] of LongBool;
```

C++

```
array[0..MAXKEYFIELDS-1] of LongBool DSKEYBOOL;
```

File

DSIntf

Description

This is type DSIntf.DSKEYBOOL.

64.4.6 DSIntf.hDSAggregate Type

Pascal

```
hDSAggregate = type Integer;
```

C++

```
type Integer hDSAggregate;
```

File

DSIntf

Description

This is type DSIntf.hDSAggregate.

64.4.7 DSIntf.hDSFilter Type

Pascal

```
hDSFilter = type Pointer;
```

C++

```
type Pointer hDSFilter;
```

File

DSIntf

Description

This is type DSIntf.hDSFilter.

64.4.8 DSIntf.MIDASNAME Type

Pascal

```
MIDASNAME = array [0..31] of Char;
```

C++

```
array [0..31] of Char MIDASNAME;
```

File

DSIntf

Description

This is type DSIntf.MIDASNAME.

64.4.9 DSIntf.MIDASPATH Type

Pascal

```
MIDASPATH = array [0..260] of Char;
```

C++

```
array [0..260] of Char MIDASPATH;
```

File

DSIntf

Description

This is type DSIntf.MIDASPATH.

64.4.10 DSIntf.pDSAttr Type

Pascal

```
pDSAttr = ^DSAttr;
```

C++

```
^DSAttr pDSAttr;
```

File

DSIntf

Description

This is type DSIntf.pDSAttr.

64.4.11 DSIntf.pDSBOOKMRK Type

Pascal

```
pDSBOOKMRK = ^DSBOOKMRK;
```

C++

```
^DSBOOKMRK pDSBOOKMRK;
```

File

DSIntf

Description

This is type DSIntf.pDSBOOKMRK.

64.4.12 DSIntf.pdsCBRType Type

Pascal

```
pdsCBRType = ^dsCBRType;
```

C++

```
^dsCBRType pdsCBRType;
```

File

DSIntf

Description

This is type DSIntf.pdsCBRType.

64.4.13 DSIntf.PDSDataPacketFldDesc Type

Pascal

```
PDSDataPacketFldDesc = ^TDSDataPacketFldDesc;
```

C++

```
^TDSDataPacketFldDesc PDSDataPacketFldDesc;
```

File

DSIntf

Description

This is type DSIntf.PDSDataPacketFldDesc.

64.4.14 DSIntf.pDSFLDDesc Type

Pascal

```
pDSFLDDesc = ^DSFLDDesc;
```

C++

```
^DSFLDDesc pDSFLDDesc;
```

File

DSIntf

Description

This is type DSIntf.pDSFLDDesc.

64.4.15 DSIntf.pDSIDXDesc Type

Pascal

```
pDSIDXDesc = ^DSIDXDesc;
```

C++

```
^DSIDXDesc pDSIDXDesc;
```

File

DSIntf

Description

This is type DSIntf.pDSIDXDesc.

64.4.16 DSIntf.pDSProps Type

Pascal

```
pDSProps = ^DSProps;
```

C++

```
^DSProps pDSProps;
```

File

DSIntf

Description

This is type DSIntf.pDSProps.

64.4.17 DSIntf.pfCHANGECallBack Type

Pascal

```
pfCHANGECallBack = procedure ( iClientData : Integer );
```

C++

```
( iClientData : Integer ) ( pfCHANGECallBack)();
```

File

DSIntf

Description

This is type DSIntf.pfCHANGECallBack.

64.4.18 DSIntf.pfDSCalcField Type

Pascal

```
pfDSCalcField = function ( iClientData : Integer; pRecBuf : Pointer ): DBResult;
```

C++

```
( iClientData : Integer; pRecBuf : Pointer ): DBResult ( pfDSCalcField)();
```

File

DSIntf

Description

This is type DSIntf.pfDSCalcField.

64.4.19 DSIntf.pfDSFilter Type

Pascal

```
pfDSFilter = function ( iClientData : Integer; pRecBuf : Pointer ): LongBool;
```

C++

```
( iClientData : Integer; pRecBuf : Pointer ): LongBool ( pfDSFilter)();
```

File

DSIntf

Description

This is type DSIntf.pfDSFilter.

64.4.20 DSIntf.pfDSReconcile Type

Pascal

```
pfDSReconcile = function ( iClientData : Integer; iRslt : Integer; iAction : DSAttr; iResponse
: dsCBRType; iErrCode : Integer; pErrMessage : PChar; pErrContext : PChar; pRecUpd : Pointer;
pRecOrg : Pointer; pRecConflict : Pointer ): dsCBRType;
```

C++

```
( iClientData : Integer; iRslt : Integer; iAction : DSAttr; iResponse : dsCBRType; iErrCode :
Integer; pErrMessage : PChar; pErrContext : PChar; pRecUpd : Pointer; pRecOrg : Pointer;
pRecConflict : Pointer ): dsCBRType ( pfDSReconcile)();
```

File

DSIntf

Description

This is type DSIntf.pfDSReconcile.

64.4.21 DSIntf.pfDSReconcile_MD Type

Pascal

```
pfDSReconcile_MD = function ( iClientData : Integer; iRslt : Integer; iAction : DSAttr;
iErrResponse : dsCBRType; iErrCode : Integer; pErrMessage : PChar; pErrContext : PChar;
pRecUpd : PByte; pRecOrg : PByte; pRecConflict : PByte; iLevels : Integer; piFieldIDs :
PInteger ): dsCBRType;
```

C++

```
( iClientData : Integer; iRslt : Integer; iAction : DSAttr; iErrResponse : dsCBRType; iErrCode
: Integer; pErrMessage : PChar; pErrContext : PChar; pRecUpd : PByte; pRecOrg : PByte;
pRecConflict : PByte; iLevels : Integer; piFieldIDs : PInteger ): dsCBRType (
pfDSReconcile_MD)();
```

File

DSIntf

Description

This is type DSIntf.pfDSReconcile_MD.

64.4.22 DSIntf.pfFLDComp Type

Pascal

```
pfFLDComp = function ( iClientData : Integer; pVal1 : Pointer; pVal2 : Pointer ): Integer;
```

C++

```
( iClientData : Integer; pVal1 : Pointer; pVal2 : Pointer ): Integer ( pfFLDComp)();
```

File

DSIntf

Description

This is type DSIntf.pfFLDComp.

64.4.23 DSIntf.phDSAggregate Type

Pascal

```
phDSAggregate = ^hDSAggregate;
```

C++

```
^hDSAggregate phDSAggregate;
```

File

DSIntf

Description

This is type DSIntf.phDSAggregate.

64.4.24 DSIntf.phDSFilter Type

Pascal

```
phDSFilter = ^hDSFilter;
```

C++

```
^hDSFilter phDSFilter;
```

File

DSIntf

Description

This is type DSIntf.phDSFilter.

64.4.25 DSIntf.PLongBool Type

Pascal

```
PLongBool = ^LongBool;
```

C++

```
^LongBool PLongBool;
```

File

DSIntf

Description

This is type DSIntf.PLongBool.

64.4.26 DSIntf.pSAVEPOINT Type

Pascal

```
pSAVEPOINT = ^SAVEPOINT;
```

C++

```
^SAVEPOINT pSAVEPOINT;
```

File

DSIntf

Description

This is type DSIntf.pSAVEPOINT.

64.4.27 DSIntf.SAVEPOINT Type

Pascal

```
SAVEPOINT = type Integer;
```

C++

```
type Integer SAVEPOINT;
```

File

DSIntf

Description

This is type DSIntf.SAVEPOINT.

64.4.28 DSIntf.TDataPacket Type

Pascal

```
TDataPacket = PSafeArray;
```

C++

```
PSafeArray TDataPacket;
```

File

DSIntf

Description

This is type DSIntf.TDataPacket.

64.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
BLANK_NOTCHANGED (see page 11727)	This is constant DSIntf.BLANK_NOTCHANGED.
BLANK_NULL (see page 11727)	This is constant DSIntf.BLANK_NULL.
BlobTypeMap (see page 11727)	This is constant DSIntf.BlobTypeMap.
CLSID_DSBase (see page 11728)	This is constant DSIntf.CLSID_DSBase.
CLSID_DSCursor (see page 11728)	This is constant DSIntf.CLSID_DSCursor.
CLSID_DSWriter (see page 11728)	This is constant DSIntf.CLSID_DSWriter.
DataTypeMap (see page 11729)	This is constant DSIntf.DataTypeMap.
DBERR_ABORTED (see page 11729)	This is constant DSIntf.DBERR_ABORTED.
DBERR_BLOBNOTFETCHED (see page 11729)	This is constant DSIntf.DBERR_BLOBNOTFETCHED.
DBERR_BOF (see page 11730)	This is constant DSIntf.DBERR_BOF.
DBERR_CANCELLED (see page 11730)	This is constant DSIntf.DBERR_CANCELLED.
DBERR_CANNOTAPPEND (see page 11730)	This is constant DSIntf.DBERR_CANNOTAPPEND.
DBERR_DATAPACKETMISMATCH (see page 11731)	This is constant DSIntf.DBERR_DATAPACKETMISMATCH.
DBERR_DELTAISEMPTY (see page 11731)	This is constant DSIntf.DBERR_DELTAISEMPTY.
DBERR_DETAILSNOTFETCHED (see page 11731)	This is constant DSIntf.DBERR_DETAILSNOTFETCHED.
DBERR_EOF (see page 11732)	This is constant DSIntf.DBERR_EOF.
DBERR_FLYAWAY_WRONGORDER (see page 11732)	This is constant DSIntf.DBERR_FLYAWAY_WRONGORDER.
DBERR_LINKFIELDSNOTUNIQUE (see page 11732)	This is constant DSIntf.DBERR_LINKFIELDSNOTUNIQUE.
DBERR_NEWERVERSIONREQ (see page 11733)	This is constant DSIntf.DBERR_NEWERVERSIONREQ.
DBERR_NOCASCADEDUPDATES (see page 11733)	This is constant DSIntf.DBERR_NOCASCADEDUPDATES.
DBERR_NOMASTERRECORD (see page 11733)	This is constant DSIntf.DBERR_NOMASTERRECORD.
DBERR_NOMETADATA (see page 11734)	This is constant DSIntf.DBERR_NOMETADATA.
DBERR_NONE (see page 11734)	This is constant DSIntf.DBERR_NONE.
DBERR_NOSUCHINDEX (see page 11734)	This is constant DSIntf.DBERR_NOSUCHINDEX.
DBERR_NOTHINGTOUNDO (see page 11735)	This is constant DSIntf.DBERR_NOTHINGTOUNDO.
dsArrayFldType (see page 11735)	This is constant DSIntf.dsArrayFldType.
dsCASCADEDELETES (see page 11735)	This is constant DSIntf.dsCASCADEDELETES.
dsCASCADEUPDATES (see page 11736)	This is constant DSIntf.dsCASCADEUPDATES.
dscbrABORT (see page 11736)	This is constant DSIntf.dscbrABORT.
dscbrAPPLY (see page 11736)	This is constant DSIntf.dscbrAPPLY.
dscbrCANCEL (see page 11737)	This is constant DSIntf.dscbrCANCEL.
dscbrCORRECT (see page 11737)	This is constant DSIntf.dscbrCORRECT.
dscbrIGNORE (see page 11737)	This is constant DSIntf.dscbrIGNORE.
dscbrMERGE (see page 11738)	This is constant DSIntf.dscbrMERGE.
dscbrREFRESH (see page 11738)	This is constant DSIntf.dscbrREFRESH.
dscbrSKIP (see page 11738)	This is constant DSIntf.dscbrSKIP.
dsCompArrayFldType (see page 11739)	This is constant DSIntf.dsCompArrayFldType.
dsDELAYEDBIT (see page 11739)	This is constant DSIntf.dsDELAYEDBIT.
dsDetUpd (see page 11739)	This is constant DSIntf.dsDetUpd.
dsEmbeddedFldType (see page 11740)	This is constant DSIntf.dsEmbeddedFldType.
dsfldADT (see page 11740)	This is constant DSIntf.dsfldADT.
dsfldARRAY (see page 11740)	This is constant DSIntf.dsfldARRAY.
dsfldBCD (see page 11741)	This is constant DSIntf.dsfldBCD.
dsfldBOOL (see page 11741)	This is constant DSIntf.dsfldBOOL.

dsfldBYTES (see page 11741)	This is constant DSIntf.dsfldBYTES.
dsfldDATE (see page 11742)	This is constant DSIntf.dsfldDATE.
dsfldDATETIME (see page 11742)	This is constant DSIntf.dsfldDATETIME.
dsfldEMBEDDEDTBL (see page 11742)	This is constant DSIntf.dsfldEMBEDDEDTBL.
dsfldFLOATIEEE (see page 11743)	This is constant DSIntf.dsfldFLOATIEEE.
dsfldFMTBCD (see page 11743)	This is constant DSIntf.dsfldFMTBCD.
dsfldINT (see page 11743)	This is constant DSIntf.dsfldINT.
dsfldREF (see page 11744)	This is constant DSIntf.dsfldREF.
dsfldTIME (see page 11744)	This is constant DSIntf.dsfldTIME.
dsfldTIMESTAMP (see page 11744)	This is constant DSIntf.dsfldTIMESTAMP.
dsfldUINT (see page 11745)	This is constant DSIntf.dsfldUINT.
dsfldUNICODE (see page 11745)	This is constant DSIntf.dsfldUNICODE.
dsfldUNKNOWN (see page 11745)	This is constant DSIntf.dsfldUNKNOWN.
dsfldZSTRING (see page 11746)	This is constant DSIntf.dsfldZSTRING.
dsIncInDelta (see page 11746)	This is constant DSIntf.dsIncInDelta.
dsIsNotVisible (see page 11746)	This is constant DSIntf.dsIsNotVisible.
dsIsVisible (see page 11747)	This is constant DSIntf.dsIsVisible.
dskeyCASEINSENSITIVE (see page 11747)	This is constant DSIntf.dskeyCASEINSENSITIVE.
dskeyDESCENDING (see page 11747)	This is constant DSIntf.dskeyDESCENDING.
dsPseudoFldType (see page 11748)	This is constant DSIntf.dsPseudoFldType.
dsRecDeleted (see page 11748)	This is constant DSIntf.dsRecDeleted.
dsRecModified (see page 11748)	This is constant DSIntf.dsRecModified.
dsRecNew (see page 11749)	This is constant DSIntf.dsRecNew.
dsRecOrg (see page 11749)	This is constant DSIntf.dsRecOrg.
dsRecUnmodified (see page 11749)	This is constant DSIntf.dsRecUnmodified.
dsSizeBitsLen (see page 11750)	This is constant DSIntf.dsSizeBitsLen.
dsSizeBitsMask (see page 11750)	This is constant DSIntf.dsSizeBitsMask.
dsTypeBitsMask (see page 11750)	This is constant DSIntf.dsTypeBitsMask.
dsUnused (see page 11751)	This is constant DSIntf.dsUnused.
dsVaryingFldType (see page 11751)	This is constant DSIntf.dsVaryingFldType.
ERRBASE_ALC (see page 11751)	This is constant DSIntf.ERRBASE_ALC.
ERRCAT_ALC (see page 11752)	This is constant DSIntf.ERRCAT_ALC.
ERRCODE_ABORTED (see page 11752)	This is constant DSIntf.ERRCODE_ABORTED.
ERRCODE_BLOBNOTFETCHED (see page 11752)	This is constant DSIntf.ERRCODE_BLOBNOTFETCHED.
ERRCODE_CANCELLED (see page 11753)	This is constant DSIntf.ERRCODE_CANCELLED.
ERRCODE_CANNOTAPPEND (see page 11753)	This is constant DSIntf.ERRCODE_CANNOTAPPEND.
ERRCODE_DATAPACKETMISMATCH (see page 11753)	This is constant DSIntf.ERRCODE_DATAPACKETMISMATCH.
ERRCODE_DELTAISEMPTY (see page 11754)	This is constant DSIntf.ERRCODE_DELTAISEMPTY.
ERRCODE_DETAILSNOTFETCHED (see page 11754)	This is constant DSIntf.ERRCODE_DETAILSNOTFETCHED.
ERRCODE_FLYAWAY_WRONGORDER (see page 11754)	This is constant DSIntf.ERRCODE_FLYAWAY_WRONGORDER.
ERRCODE_LINKFIELDSNOTUNIQUE (see page 11755)	This is constant DSIntf.ERRCODE_LINKFIELDSNOTUNIQUE.
ERRCODE_NEWERVERSIONREQ (see page 11755)	This is constant DSIntf.ERRCODE_NEWERVERSIONREQ.
ERRCODE_NOCASCADEDUPDATES (see page 11755)	This is constant DSIntf.ERRCODE_NOCASCADEDUPDATES.
ERRCODE_NOMASTERRECORD (see page 11756)	This is constant DSIntf.ERRCODE_NOMASTERRECORD.
ERRCODE_NOMETADATA (see page 11756)	This is constant DSIntf.ERRCODE_NOMETADATA.
ERRCODE_NOTHINGTOUNDO (see page 11756)	This is constant DSIntf.ERRCODE_NOTHINGTOUNDO.
FieldTypeMap (see page 11757)	This is constant DSIntf.FieldTypeMap.
fldADT (see page 11757)	This is constant DSIntf.fldADT.
fldARRAY (see page 11757)	This is constant DSIntf.fldARRAY.
fldAttrHIDDEN (see page 11758)	This is constant DSIntf.fldAttrHIDDEN.
fldAttrLINK (see page 11758)	This is constant DSIntf.fldAttrLINK.
fldAttrREADONLY (see page 11758)	This is constant DSIntf.fldAttrREADONLY.
fldAttrREQUIRED (see page 11759)	This is constant DSIntf.fldAttrREQUIRED.
fldBCD (see page 11759)	This is constant DSIntf.fldBCD.

fldBLOB (see page 11759)	This is constant DSIntf.fldBLOB.
fldBOOL (see page 11760)	This is constant DSIntf.fldBOOL.
fldBYTES (see page 11760)	This is constant DSIntf.fldBYTES.
fldCURSOR (see page 11760)	This is constant DSIntf.fldCURSOR.
fldDATE (see page 11761)	This is constant DSIntf.fldDATE.
fldDATETIME (see page 11761)	This is constant DSIntf.fldDATETIME.
fldFLOAT (see page 11761)	This is constant DSIntf.fldFLOAT.
fldFLOATIEEE (see page 11762)	This is constant DSIntf.fldFLOATIEEE.
fldFMTBCD (see page 11762)	This is constant DSIntf.fldFMTBCD.
fldINT16 (see page 11762)	This is constant DSIntf.fldINT16.
fldINT32 (see page 11763)	This is constant DSIntf.fldINT32.
fldINT64 (see page 11763)	This is constant DSIntf.fldINT64.
fldLOCKINFO (see page 11763)	This is constant DSIntf.fldLOCKINFO.
fldREF (see page 11764)	This is constant DSIntf.fldREF.
fldstACCOLEOBJ (see page 11764)	This is constant DSIntf.fldstACCOLEOBJ.
fldstAUTOINC (see page 11764)	This is constant DSIntf.fldstAUTOINC.
fldstBFILE (see page 11765)	This is constant DSIntf.fldstBFILE.
fldstBINARY (see page 11765)	This is constant DSIntf.fldstBINARY.
fldstDBSOLEOBJ (see page 11765)	This is constant DSIntf.fldstDBSOLEOBJ.
fldstFIXED (see page 11766)	This is constant DSIntf.fldstFIXED.
fldstFMTMEMO (see page 11766)	This is constant DSIntf.fldstFMTMEMO.
fldstGRAPHIC (see page 11766)	This is constant DSIntf.fldstGRAPHIC.
fldstGUID (see page 11767)	This is constant DSIntf.fldstGUID.
fldstHBINARY (see page 11767)	This is constant DSIntf.fldstHBINARY.
fldstHMEMO (see page 11767)	This is constant DSIntf.fldstHMEMO.
fldstMEMO (see page 11768)	This is constant DSIntf.fldstMEMO.
fldstMONEY (see page 11768)	This is constant DSIntf.fldstMONEY.
fldstOLEOBJ (see page 11768)	This is constant DSIntf.fldstOLEOBJ.
fldstORAINTERVAL (see page 11769)	This is constant DSIntf.fldstORAINTERVAL.
fldstORATIMESTAMP (see page 11769)	This is constant DSIntf.fldstORATIMESTAMP.
fldstPASSWORD (see page 11769)	This is constant DSIntf.fldstPASSWORD.
fldstReference (see page 11770)	This is constant DSIntf.fldstReference.
fldstTYPEDBINARY (see page 11770)	This is constant DSIntf.fldstTYPEDBINARY.
fldstWIDEMEMO (see page 11770)	This is constant DSIntf.fldstWIDEMEMO.
FldSubTypeMap (see page 11771)	This is constant DSIntf.FldSubTypeMap.
fldTABLE (see page 11771)	This is constant DSIntf.fldTABLE.
fldTIME (see page 11771)	This is constant DSIntf.fldTIME.
fldTIMESTAMP (see page 11772)	This is constant DSIntf.fldTIMESTAMP.
fldUINT16 (see page 11772)	This is constant DSIntf.fldUINT16.
fldUINT32 (see page 11772)	This is constant DSIntf.fldUINT32.
fldUINT64 (see page 11773)	This is constant DSIntf.fldUINT64.
fldUNICODE (see page 11773)	This is constant DSIntf.fldUNICODE.
fldUNKNOWN (see page 11773)	This is constant DSIntf.fldUNKNOWN.
fldVARBYTES (see page 11774)	This is constant DSIntf.fldVARBYTES.
fldWIDESTRING (see page 11774)	This is constant DSIntf.fldWIDESTRING.
fldZSTRING (see page 11774)	This is constant DSIntf.fldZSTRING.
MAXKEYFIELDS (see page 11775)	This is constant DSIntf.MAXKEYFIELDS.
MAXLOGFLDTYPES (see page 11775)	This is constant DSIntf.MAXLOGFLDTYPES.
mdALLOWLINKCHANGE (see page 11775)	This is constant DSIntf.mdALLOWLINKCHANGE.
mdCASCADEDEL (see page 11776)	This is constant DSIntf.mdCASCADEDEL.
mdCASCADEMOD (see page 11776)	This is constant DSIntf.mdCASCADEMOD.
PACKETVERSION_1 (see page 11776)	This is constant DSIntf.PACKETVERSION_1.
PACKETVERSION_2 (see page 11777)	This is constant DSIntf.PACKETVERSION_2.
PACKETVERSION_3 (see page 11777)	This is constant DSIntf.PACKETVERSION_3.

szAUTOINCVALUE (see page 11777)	This is constant DSIntf.szAUTOINCVALUE.
szBDEDEFCL (see page 11778)	This is constant DSIntf.szBDEDEFCL.
szBDEDEFX (see page 11778)	This is constant DSIntf.szBDEDEFX.
szBDEDOMCL (see page 11778)	This is constant DSIntf.szBDEDOMCL.
szBDEDOMX (see page 11779)	This is constant DSIntf.szBDEDOMX.
szBDERECCL (see page 11779)	This is constant DSIntf.szBDERECCL.
szBDERECX (see page 11779)	This is constant DSIntf.szBDERECX.
szCHANGE_LOG (see page 11780)	This is constant DSIntf.szCHANGE_LOG.
szCHANGEINDEX (see page 11780)	This is constant DSIntf.szCHANGEINDEX.
szCONSTRAINTS (see page 11780)	This is constant DSIntf.szCONSTRAINTS.
szDATASET_CONTEXT (see page 11781)	This is constant DSIntf.szDATASET_CONTEXT.
szDATASET_DELTA (see page 11781)	This is constant DSIntf.szDATASET_DELTA.
szDECIMALS (see page 11781)	This is constant DSIntf.szDECIMALS.
szDEFAULT_ORDER (see page 11782)	This is constant DSIntf.szDEFAULT_ORDER.
szDISABLE_DELETES (see page 11782)	This is constant DSIntf.szDISABLE_DELETES.
szDISABLE_EDITS (see page 11782)	This is constant DSIntf.szDISABLE_EDITS.
szDISABLE_INSERTS (see page 11783)	This is constant DSIntf.szDISABLE_INSERTS.
szELEMENTS (see page 11783)	This is constant DSIntf.szELEMENTS.
szFIELDNAME (see page 11783)	This is constant DSIntf.szFIELDNAME.
szFIELDPROPS (see page 11784)	This is constant DSIntf.szFIELDPROPS.
szLCID (see page 11784)	This is constant DSIntf.szLCID.
szMAXVALUE (see page 11784)	This is constant DSIntf.szMAXVALUE.
szMD_FIELDLINKS (see page 11785)	This is constant DSIntf.szMD_FIELDLINKS.
szMD_SEMANTICS (see page 11785)	This is constant DSIntf.szMD_SEMANTICS.
szMINVALUE (see page 11785)	This is constant DSIntf.szMINVALUE.
szNO_RESET_CALL (see page 11786)	This is constant DSIntf.szNO_RESET_CALL.
szORIGIN (see page 11786)	This is constant DSIntf.szORIGIN.
szPRIMARY_KEY (see page 11786)	This is constant DSIntf.szPRIMARY_KEY.
szPROVFLAGS (see page 11787)	This is constant DSIntf.szPROVFLAGS.
szREADONLY (see page 11787)	This is constant DSIntf.szREADONLY.
szSERVER_COL (see page 11787)	This is constant DSIntf.szSERVER_COL.
szSERVERCALC (see page 11788)	This is constant DSIntf.szSERVERCALC.
szstACCOLEOBJ (see page 11788)	This is constant DSIntf.szstACCOLEOBJ.
szstADTNESTEDTABLE (see page 11788)	This is constant DSIntf.szstADTNESTEDTABLE.
szstAUTOINC (see page 11789)	This is constant DSIntf.szstAUTOINC.
szstBINARY (see page 11789)	This is constant DSIntf.szstBINARY.
szstDBSOLEOBJ (see page 11789)	This is constant DSIntf.szstDBSOLEOBJ.
szstFIXEDCHAR (see page 11790)	This is constant DSIntf.szstFIXEDCHAR.
szstFMTMEMO (see page 11790)	This is constant DSIntf.szstFMTMEMO.
szstGRAPHIC (see page 11790)	This is constant DSIntf.szstGRAPHIC.
szstGUID (see page 11791)	This is constant DSIntf.szstGUID.
szstHBINARY (see page 11791)	This is constant DSIntf.szstHBINARY.
szstHMEMO (see page 11791)	This is constant DSIntf.szstHMEMO.
szstMEMO (see page 11792)	This is constant DSIntf.szstMEMO.
szstMONEY (see page 11792)	This is constant DSIntf.szstMONEY.
szstOLEOBJ (see page 11792)	This is constant DSIntf.szstOLEOBJ.
szstREFNESTEDTABLE (see page 11793)	This is constant DSIntf.szstREFNESTEDTABLE.
szstTYPEDBINARY (see page 11793)	This is constant DSIntf.szstTYPEDBINARY.
szstWideMEMO (see page 11793)	This is constant DSIntf.szstWideMEMO.
szSUBTYPE (see page 11794)	This is constant DSIntf.szSUBTYPE.
szTABLE_NAME (see page 11794)	This is constant DSIntf.szTABLE_NAME.
szTYPENAME (see page 11794)	This is constant DSIntf.szTYPENAME.
szUNIQUE_KEY (see page 11795)	This is constant DSIntf.szUNIQUE_KEY.
szUPDATEMODE (see page 11795)	This is constant DSIntf.szUPDATEMODE.

szWIDTH (see page 11795)	This is constant DSIntf.szWIDTH.
xmlOFF (see page 11796)	This is constant DSIntf.xmlOFF.
xmlON (see page 11796)	This is constant DSIntf.xmlON.
xmlUNTYPED (see page 11796)	This is constant DSIntf.xmlUNTYPED.
xmlUTF8 (see page 11797)	This is constant DSIntf.xmlUTF8.
xmlXMLDATATYPED (see page 11797)	This is constant DSIntf.xmlXMLDATATYPED.
xmlXMLSCHEMA (see page 11797)	This is constant DSIntf.xmlXMLSCHEMA.
xmlXMLUTF8 (see page 11798)	This is constant DSIntf.xmlXMLUTF8.

64.5.1 DSIntf.BLANK_NOTCHANGED Constant

Pascal

```
BLANK_NOTCHANGED = 2;
```

C++

```
BLANK_NOTCHANGED = 2;
```

File

DSIntf

Description

This is constant DSIntf.BLANK_NOTCHANGED.

64.5.2 DSIntf.BLANK_NULL Constant

Pascal

```
BLANK_NULL = 1;
```

C++

```
BLANK_NULL = 1;
```

File

DSIntf

Description

This is constant DSIntf.BLANK_NULL.

64.5.3 DSIntf.BlobTypeMap Constant

Pascal

```
BlobTypeMap: array[fldstMEMO..fldstBFILE] of TFieldType = ( ftMemo, ftBlob, ftFmtMemo,
ftParadoxOle, ftGraphic, ftDBaseOle, ftTypedBinary, ftBlob, ftBlob, ftBlob, ftWideMemo,
ftOraClob, ftOraBlob, ftBlob, ftBlob);
```

C++

```
array[fldstMEMO..fldstBFILE] of TFieldType BlobTypeMap = ( ftMemo, ftBlob, ftFmtMemo,
ftParadoxOle, ftGraphic, ftDBaseOle, ftTypedBinary, ftBlob, ftBlob, ftBlob, ftWideMemo,
```

```
ftOraClob, ftOraBlob, ftBlob, ftBlob);
```

File

DSIntf

Description

This is constant DSIntf.BlobTypeMap.

64.5.4 DSIntf.CLSID_DSBase Constant

Pascal

```
CLSID_DSBase: TGUID = '{9E8D2FA1-591C-11D0-BF52-0020AF32BD64}';
```

C++

```
TGUID CLSID_DSBase = '{9E8D2FA1-591C-11D0-BF52-0020AF32BD64}';
```

File

DSIntf

Description

This is constant DSIntf.CLSID_DSBase.

64.5.5 DSIntf.CLSID_DSCursor Constant

Pascal

```
CLSID_DSCursor: TGUID = '{9E8D2FA3-591C-11D0-BF52-0020AF32BD64}';
```

C++

```
TGUID CLSID_DSCursor = '{9E8D2FA3-591C-11D0-BF52-0020AF32BD64}';
```

File

DSIntf

Description

This is constant DSIntf.CLSID_DSCursor.

64.5.6 DSIntf.CLSID_DSWriter Constant

Pascal

```
CLSID_DSWriter: TGUID = '{9E8D2FA5-591C-11D0-BF52-0020AF32BD64}';
```

C++

```
TGUID CLSID_DSWriter = '{9E8D2FA5-591C-11D0-BF52-0020AF32BD64}';
```

File

DSIntf

Description

This is constant DSIntf.CLSID_DSWriter.

64.5.7 DSIntf.DataTypeMap Constant

Pascal

```
DataTypeMap: array[0..MAXLOGFLDTYPES - 1] of TFieldType = ( ftUnknown, ftString, ftDate,
ftBlob, ftBoolean, ftSmallint, ftInteger, ftFloat, ftBCD, ftBytes, ftTime, ftDateTime, ftWord,
ftInteger, ftUnknown, ftVarBytes, ftUnknown, ftUnknown, ftLargeInt, ftLargeInt, ftADT,
ftArray, ftReference, ftDataSet, ftTimeStamp, ftFMTBcd, ftWideString);
```

C++

```
array[0..MAXLOGFLDTYPES - 1] of TFieldType DataTypeMap = ( ftUnknown, ftString, ftDate,
ftBlob, ftBoolean, ftSmallint, ftInteger, ftFloat, ftBCD, ftBytes, ftTime, ftDateTime, ftWord,
ftInteger, ftUnknown, ftVarBytes, ftUnknown, ftUnknown, ftLargeInt, ftLargeInt, ftADT,
ftArray, ftReference, ftDataSet, ftTimeStamp, ftFMTBcd, ftWideString);
```

File

DSIntf

Description

This is constant DSIntf.DataTypeMap.

64.5.8 DSIntf.DBERR_ABORTED Constant

Pascal

```
DBERR_ABORTED = ERRBASE_ALC + ERRCODE_ABORTED;
```

C++

```
DBERR_ABORTED = ERRBASE_ALC + ERRCODE_ABORTED;
```

File

DSIntf

Description

This is constant DSIntf.DBERR_ABORTED.

64.5.9 DSIntf.DBERR_BLOBNOTFETCHED Constant

Pascal

```
DBERR_BLOBNOTFETCHED = ERRBASE_ALC + ERRCODE_BLOBNOTFETCHED;
```

C++

```
DBERR_BLOBNOTFETCHED = ERRBASE_ALC + ERRCODE_BLOBNOTFETCHED;
```

File

DSIntf

Description

This is constant DSIntf.DBERR_BLOBNOTFETCHED.

64.5.10 DSIntf.DBERR_BOF Constant

Pascal

```
DBERR_BOF = $2201;
```

C++

```
DBERR_BOF = 0x2201;
```

File

DSIntf

Description

This is constant DSIntf.DBERR_BOF.

64.5.11 DSIntf.DBERR_CANCELLED Constant

Pascal

```
DBERR_CANCELLED = ERRBASE_ALC + ERRCODE_CANCELLED;
```

C++

```
DBERR_CANCELLED = ERRBASE_ALC + ERRCODE_CANCELLED;
```

File

DSIntf

Description

This is constant DSIntf.DBERR_CANCELLED.

64.5.12 DSIntf.DBERR_CANNOTAPPEND Constant

Pascal

```
DBERR_CANNOTAPPEND = ERRBASE_ALC + ERRCODE_CANNOTAPPEND;
```

C++

```
DBERR_CANNOTAPPEND = ERRBASE_ALC + ERRCODE_CANNOTAPPEND;
```

File

DSIntf

Description

This is constant DSIntf.DBERR_CANNOTAPPEND.

64.5.13 DSIntf.DBERR_DATAPACKETMISMATCH Constant

Pascal

```
DBERR_DATAPACKETMISMATCH = ERRBASE_ALC + ERRCODE_DATAPACKETMISMATCH;
```

C++

```
DBERR_DATAPACKETMISMATCH = ERRBASE_ALC + ERRCODE_DATAPACKETMISMATCH;
```

File

DSIntf

Description

This is constant DSIntf.DBERR_DATAPACKETMISMATCH.

64.5.14 DSIntf.DBERR_DELTAISEMPTY Constant

Pascal

```
DBERR_DELTAISEMPTY = ERRBASE_ALC + ERRCODE_DELTAISEMPTY;
```

C++

```
DBERR_DELTAISEMPTY = ERRBASE_ALC + ERRCODE_DELTAISEMPTY;
```

File

DSIntf

Description

This is constant DSIntf.DBERR_DELTAISEMPTY.

64.5.15 DSIntf.DBERR_DETAILSNOTFETCHED Constant

Pascal

```
DBERR_DETAILSNOTFETCHED = ERRBASE_ALC + ERRCODE_DETAILSNOTFETCHED;
```

C++

```
DBERR_DETAILSNOTFETCHED = ERRBASE_ALC + ERRCODE_DETAILSNOTFETCHED;
```

File

DSIntf

Description

This is constant DSIntf.DBERR_DETAILSNOTFETCHED.

64.5.16 DSIntf.DBERR_EOF Constant

Pascal

```
DBERR_EOF = $2202;
```

C++

```
DBERR_EOF = 0x2202;
```

File

DSIntf

Description

This is constant DSIntf.DBERR_EOF.

64.5.17 DSIntf.DBERR_FLYAWAY_WRONGORDER Constant

Pascal

```
DBERR_FLYAWAY_WRONGORDER = ERRBASE_ALC + ERRCODE_FLYAWAY_WRONGORDER;
```

C++

```
DBERR_FLYAWAY_WRONGORDER = ERRBASE_ALC + ERRCODE_FLYAWAY_WRONGORDER;
```

File

DSIntf

Description

This is constant DSIntf.DBERR_FLYAWAY_WRONGORDER.

64.5.18 DSIntf.DBERR_LINKFIELDSNOTUNIQUE Constant

Pascal

```
DBERR_LINKFIELDSNOTUNIQUE = ERRBASE_ALC + ERRCODE_LINKFIELDSNOTUNIQUE;
```

C++

```
DBERR_LINKFIELDSNOTUNIQUE = ERRBASE_ALC + ERRCODE_LINKFIELDSNOTUNIQUE;
```

File

DSIntf

Description

This is constant DSIntf.DBERR_LINKFIELDSNOTUNIQUE.

64.5.19 DSIntf.DBERR_NEWERVERSIONREQ Constant

Pascal

```
DBERR_NEWERVERSIONREQ = ERRBASE_ALC + ERRCODE_NEWERVERSIONREQ;
```

C++

```
DBERR_NEWERVERSIONREQ = ERRBASE_ALC + ERRCODE_NEWERVERSIONREQ;
```

File

DSIntf

Description

This is constant DSIntf.DBERR_NEWERVERSIONREQ.

64.5.20 DSIntf.DBERR_NOCASCADEDUPDATES Constant

Pascal

```
DBERR_NOCASCADEDUPDATES = ERRBASE_ALC + ERRCODE_NOCASCADEDUPDATES;
```

C++

```
DBERR_NOCASCADEDUPDATES = ERRBASE_ALC + ERRCODE_NOCASCADEDUPDATES;
```

File

DSIntf

Description

This is constant DSIntf.DBERR_NOCASCADEDUPDATES.

64.5.21 DSIntf.DBERR_NOMASTERRECORD Constant

Pascal

```
DBERR_NOMASTERRECORD = ERRBASE_ALC + ERRCODE_NOMASTERRECORD;
```

C++

```
DBERR_NOMASTERRECORD = ERRBASE_ALC + ERRCODE_NOMASTERRECORD;
```

File

DSIntf

Description

This is constant DSIntf.DBERR_NOMASTERRECORD.

64.5.22 DSIntf.DBERR_NOMETADATA Constant

Pascal

```
DBERR_NOMETADATA = ERRBASE_ALC + ERRCODE_NOMETADATA;
```

C++

```
DBERR_NOMETADATA = ERRBASE_ALC + ERRCODE_NOMETADATA;
```

File

DSIntf

Description

This is constant DSIntf.DBERR_NOMETADATA.

64.5.23 DSIntf.DBERR_NONE Constant

Pascal

```
DBERR_NONE = 0;
```

C++

```
DBERR_NONE = 0;
```

File

DSIntf

Description

This is constant DSIntf.DBERR_NONE.

64.5.24 DSIntf.DBERR_NOSUCHINDEX Constant

Pascal

```
DBERR_NOSUCHINDEX = $270D;
```

C++

```
DBERR_NOSUCHINDEX = 0x270D;
```

File

DSIntf

Description

This is constant DSIntf.DBERR_NOSUCHINDEX.

64.5.25 DSIntf.DBERR_NOTHINGTOUNDO Constant

Pascal

```
DBERR_NOTHINGTOUNDO = ERRBASE_ALC + ERRCODE_NOTHINGTOUNDO;
```

C++

```
DBERR_NOTHINGTOUNDO = ERRBASE_ALC + ERRCODE_NOTHINGTOUNDO;
```

File

DSIntf

Description

This is constant DSIntf.DBERR_NOTHINGTOUNDO.

64.5.26 DSIntf.dsArrayFldType Constant

Pascal

```
dsArrayFldType = $00800000;
```

C++

```
dsArrayFldType = 0x00800000;
```

File

DSIntf

Description

This is constant DSIntf.dsArrayFldType.

64.5.27 DSIntf.dsCASCADEDELETES Constant

Pascal

```
dsCASCADEDELETES = 1;
```

C++

```
dsCASCADEDELETES = 1;
```

File

DSIntf

Description

This is constant DSIntf.dsCASCADEDELETES.

64.5.28 DSIntf.dsCASCADEUPDATES Constant

Pascal

```
dsCASCADEUPDATES = 2;
```

C++

```
dsCASCADEUPDATES = 2;
```

File

DSIntf

Description

This is constant DSIntf.dsCASCADEUPDATES.

64.5.29 DSIntf.dscbrABORT Constant

Pascal

```
dscbrABORT = 2;
```

C++

```
dscbrABORT = 2;
```

File

DSIntf

Description

This is constant DSIntf.dscbrABORT.

64.5.30 DSIntf.dscbrAPPLY Constant

Pascal

```
dscbrAPPLY = 4;
```

C++

```
dscbrAPPLY = 4;
```

File

DSIntf

Description

This is constant DSIntf.dscbrAPPLY.

64.5.31 DSIntf.dscbrCANCEL Constant

Pascal

```
dscbrCANCEL = 5;
```

C++

```
dscbrCANCEL = 5;
```

File

DSIntf

Description

This is constant DSIntf.dscbrCANCEL.

64.5.32 DSIntf.dscbrCORRECT Constant

Pascal

```
dscbrCORRECT = 4;
```

C++

```
dscbrCORRECT = 4;
```

File

DSIntf

Description

This is constant DSIntf.dscbrCORRECT.

64.5.33 DSIntf.dscbrIGNORE Constant

Pascal

```
dscbrIGNORE = 5;
```

C++

```
dscbrIGNORE = 5;
```

File

DSIntf

Description

This is constant DSIntf.dscbrIGNORE.

64.5.34 DSIntf.dscbrMERGE Constant

Pascal

```
dscbrMERGE = 3;
```

C++

```
dscbrMERGE = 3;
```

File

DSIntf

Description

This is constant DSIntf.dscbrMERGE.

64.5.35 DSIntf.dscbrREFRESH Constant

Pascal

```
dscbrREFRESH = 6;
```

C++

```
dscbrREFRESH = 6;
```

File

DSIntf

Description

This is constant DSIntf.dscbrREFRESH.

64.5.36 DSIntf.dscbrSKIP Constant

Pascal

```
dscbrSKIP = 1;
```

C++

```
dscbrSKIP = 1;
```

File

DSIntf

Description

This is constant DSIntf.dscbrSKIP.

64.5.37 DSIntf.dsCompArrayFldType Constant

Pascal

```
dsCompArrayFldType = $02000000;
```

C++

```
dsCompArrayFldType = 0x02000000;
```

File

DSIntf

Description

This is constant DSIntf.dsCompArrayFldType.

64.5.38 DSIntf.dsDELAYEDBIT Constant

Pascal

```
dsDELAYEDBIT = $80000000;
```

C++

```
dsDELAYEDBIT = 0x80000000;
```

File

DSIntf

Description

This is constant DSIntf.dsDELAYEDBIT.

64.5.39 DSIntf.dsDetUpd Constant

Pascal

```
dsDetUpd = $0040;
```

C++

```
dsDetUpd = 0x0040;
```

File

DSIntf

Description

This is constant DSIntf.dsDetUpd.

64.5.40 DSIntf.dsEmbeddedFldType Constant

Pascal

```
dsEmbeddedFldType = $04000000;
```

C++

```
dsEmbeddedFldType = 0x04000000;
```

File

DSIntf

Description

This is constant DSIntf.dsEmbeddedFldType.

64.5.41 DSIntf.dsfldADT Constant

Pascal

```
dsfldADT = 12;
```

C++

```
dsfldADT = 12;
```

File

DSIntf

Description

This is constant DSIntf.dsfldADT.

64.5.42 DSIntf.dsfldARRAY Constant

Pascal

```
dsfldARRAY = 13;
```

C++

```
dsfldARRAY = 13;
```

File

DSIntf

Description

This is constant DSIntf.dsfldARRAY.

64.5.43 DSIntf.dsfldBCD Constant

Pascal

```
dsfldBCD = 5;
```

C++

```
dsfldBCD = 5;
```

File

DSIntf

Description

This is constant DSIntf.dsfldBCD.

64.5.44 DSIntf.dsfldBOOL Constant

Pascal

```
dsfldBOOL = 3;
```

C++

```
dsfldBOOL = 3;
```

File

DSIntf

Description

This is constant DSIntf.dsfldBOOL.

64.5.45 DSIntf.dsfldBYTES Constant

Pascal

```
dsfldBYTES = 11;
```

C++

```
dsfldBYTES = 11;
```

File

DSIntf

Description

This is constant DSIntf.dsfldBYTES.

64.5.46 DSIntf.dsfldDATE Constant

Pascal

```
dsfldDATE = 6;
```

C++

```
dsfldDATE = 6;
```

File

DSIntf

Description

This is constant DSIntf.dsfldDATE.

64.5.47 DSIntf.dsfldDATETIME Constant

Pascal

```
dsfldDATETIME = 17;
```

C++

```
dsfldDATETIME = 17;
```

File

DSIntf

Description

This is constant DSIntf.dsfldDATETIME.

64.5.48 DSIntf.dsfldEMBEDDEDTBL Constant

Pascal

```
dsfldEMBEDDEDTBL = 14;
```

C++

```
dsfldEMBEDDEDTBL = 14;
```

File

DSIntf

Description

This is constant DSIntf.dsfldEMBEDDEDTBL.

64.5.49 DSIntf.dsfldFLOATIEEE Constant

Pascal

```
dsfldFLOATIEEE = 4;
```

C++

```
dsfldFLOATIEEE = 4;
```

File

DSIntf

Description

This is constant DSIntf.dsfldFLOATIEEE.

64.5.50 DSIntf.dsfldFMTBCD Constant

Pascal

```
dsfldFMTBCD = 18;
```

C++

```
dsfldFMTBCD = 18;
```

File

DSIntf

Description

This is constant DSIntf.dsfldFMTBCD.

64.5.51 DSIntf.dsfldINT Constant

Pascal

```
dsfldINT = 1;
```

C++

```
dsfldINT = 1;
```

File

DSIntf

Description

This is constant DSIntf.dsfldINT.

64.5.52 DSIntf.dsfldREF Constant

Pascal

```
dsfldREF = 15;
```

C++

```
dsfldREF = 15;
```

File

DSIntf

Description

This is constant DSIntf.dsfldREF.

64.5.53 DSIntf.dsfldTIME Constant

Pascal

```
dsfldTIME = 7;
```

C++

```
dsfldTIME = 7;
```

File

DSIntf

Description

This is constant DSIntf.dsfldTIME.

64.5.54 DSIntf.dsfldTIMESTAMP Constant

Pascal

```
dsfldTIMESTAMP = 8;
```

C++

```
dsfldTIMESTAMP = 8;
```

File

DSIntf

Description

This is constant DSIntf.dsfldTIMESTAMP.

64.5.55 DSIntf.dsfldUINT Constant

Pascal

```
dsfldUINT = 2;
```

C++

```
dsfldUINT = 2;
```

File

DSIntf

Description

This is constant DSIntf.dsfldUINT.

64.5.56 DSIntf.dsfldUNICODE Constant

Pascal

```
dsfldUNICODE = 10;
```

C++

```
dsfldUNICODE = 10;
```

File

DSIntf

Description

This is constant DSIntf.dsfldUNICODE.

64.5.57 DSIntf.dsfldUNKNOWN Constant

Pascal

```
dsfldUNKNOWN = 0;
```

C++

```
dsfldUNKNOWN = 0;
```

File

DSIntf

Description

This is constant DSIntf.dsfldUNKNOWN.

64.5.58 DSIntf.dsfldZSTRING Constant

Pascal

```
dsfldZSTRING = 9;
```

C++

```
dsfldZSTRING = 9;
```

File

DSIntf

Description

This is constant DSIntf.dsfldZSTRING.

64.5.59 DSIntf.dsIncInDelta Constant

Pascal

```
dsIncInDelta = $80000000;
```

C++

```
dsIncInDelta = 0x80000000;
```

File

DSIntf

Description

This is constant DSIntf.dsIncInDelta.

64.5.60 DSIntf.dsIsNotVisible Constant

Pascal

```
dsIsNotVisible = dsRecDeleted or dsRecOrg or dsUnused;
```

C++

```
dsIsNotVisible = dsRecDeleted or dsRecOrg or dsUnused;
```

File

DSIntf

Description

This is constant DSIntf.dsIsNotVisible.

64.5.61 DSIntf.dsIsVisible Constant

Pascal

```
dsIsVisible = not (dsRecDeleted or dsRecOrg or dsUnused);
```

C++

```
dsIsVisible = not (dsRecDeleted or dsRecOrg or dsUnused);
```

File

DSIntf

Description

This is constant DSIntf.dsIsVisible.

64.5.62 DSIntf.dskeyCASEINSENSITIVE Constant

Pascal

```
dskeyCASEINSENSITIVE = $4000;
```

C++

```
dskeyCASEINSENSITIVE = 0x4000;
```

File

DSIntf

Description

This is constant DSIntf.dskeyCASEINSENSITIVE.

64.5.63 DSIntf.dskeyDESCENDING Constant

Pascal

```
dskeyDESCENDING = $8000;
```

C++

```
dskeyDESCENDING = 0x8000;
```

File

DSIntf

Description

This is constant DSIntf.dskeyDESCENDING.

64.5.64 DSIntf.dsPseudoFldType Constant

Pascal

```
dsPseudoFldType = $01000000;
```

C++

```
dsPseudoFldType = 0x01000000;
```

File

DSIntf

Description

This is constant DSIntf.dsPseudoFldType.

64.5.65 DSIntf.dsRecDeleted Constant

Pascal

```
dsRecDeleted = $0002;
```

C++

```
dsRecDeleted = 0x0002;
```

File

DSIntf

Description

This is constant DSIntf.dsRecDeleted.

64.5.66 DSIntf.dsRecModified Constant

Pascal

```
dsRecModified = $0008;
```

C++

```
dsRecModified = 0x0008;
```

File

DSIntf

Description

This is constant DSIntf.dsRecModified.

64.5.67 DSIntf.dsRecNew Constant

Pascal

```
dsRecNew = $0004;
```

C++

```
dsRecNew = 0x0004;
```

File

DSIntf

Description

This is constant DSIntf.dsRecNew.

64.5.68 DSIntf.dsRecOrg Constant

Pascal

```
dsRecOrg = $0001;
```

C++

```
dsRecOrg = 0x0001;
```

File

DSIntf

Description

This is constant DSIntf.dsRecOrg.

64.5.69 DSIntf.dsRecUnmodified Constant

Pascal

```
dsRecUnmodified = $0000;
```

C++

```
dsRecUnmodified = 0x0000;
```

File

DSIntf

Description

This is constant DSIntf.dsRecUnmodified.

64.5.70 DSIntf.dsSizeBitsLen Constant

Pascal

```
dsSizeBitsLen = 16;
```

C++

```
dsSizeBitsLen = 16;
```

File

DSIntf

Description

This is constant DSIntf.dsSizeBitsLen.

64.5.71 DSIntf.dsSizeBitsMask Constant

Pascal

```
dsSizeBitsMask = $0000FFFF;
```

C++

```
dsSizeBitsMask = 0x0000FFFF;
```

File

DSIntf

Description

This is constant DSIntf.dsSizeBitsMask.

64.5.72 DSIntf.dsTypeBitsMask Constant

Pascal

```
dsTypeBitsMask = $003F0000;
```

C++

```
dsTypeBitsMask = 0x003F0000;
```

File

DSIntf

Description

This is constant DSIntf.dsTypeBitsMask.

64.5.73 DSIntf.dsUnused Constant

Pascal

```
dsUnused = $0020;
```

C++

```
dsUnused = 0x0020;
```

File

DSIntf

Description

This is constant DSIntf.dsUnused.

64.5.74 DSIntf.dsVaryingFldType Constant

Pascal

```
dsVaryingFldType = $00400000;
```

C++

```
dsVaryingFldType = 0x00400000;
```

File

DSIntf

Description

This is constant DSIntf.dsVaryingFldType.

64.5.75 DSIntf.ERRBASE_ALC Constant

Pascal

```
ERRBASE_ALC = $4000;
```

C++

```
ERRBASE_ALC = 0x4000;
```

File

DSIntf

Description

This is constant DSIntf.ERRBASE_ALC.

64.5.76 DSIntf.ERRCAT_ALC Constant

Pascal

```
ERRCAT_ALC = $40;
```

C++

```
ERRCAT_ALC = 0x40;
```

File

DSIntf

Description

This is constant DSIntf.ERRCAT_ALC.

64.5.77 DSIntf.ERRCODE_ABORTED Constant

Pascal

```
ERRCODE_ABORTED = 6;
```

C++

```
ERRCODE_ABORTED = 6;
```

File

DSIntf

Description

This is constant DSIntf.ERRCODE_ABORTED.

64.5.78 DSIntf.ERRCODE_BLOBNOTFETCHED Constant

Pascal

```
ERRCODE_BLOBNOTFETCHED = 9;
```

C++

```
ERRCODE_BLOBNOTFETCHED = 9;
```

File

DSIntf

Description

This is constant DSIntf.ERRCODE_BLOBNOTFETCHED.

64.5.79 DSIntf.ERRCODE_CANCELLED Constant

Pascal

```
ERRCODE_CANCELLED = 7;
```

C++

```
ERRCODE_CANCELLED = 7;
```

File

DSIntf

Description

This is constant DSIntf.ERRCODE_CANCELLED.

64.5.80 DSIntf.ERRCODE_CANNOTAPPEND Constant

Pascal

```
ERRCODE_CANNOTAPPEND = 4;
```

C++

```
ERRCODE_CANNOTAPPEND = 4;
```

File

DSIntf

Description

This is constant DSIntf.ERRCODE_CANNOTAPPEND.

64.5.81 DSIntf.ERRCODE_DATAPACKETMISMATCH Constant

Pascal

```
ERRCODE_DATAPACKETMISMATCH = 5;
```

C++

```
ERRCODE_DATAPACKETMISMATCH = 5;
```

File

DSIntf

Description

This is constant DSIntf.ERRCODE_DATAPACKETMISMATCH.

64.5.82 DSIntf.ERRCODE_DELTAISEMPTY Constant

Pascal

```
ERRCODE_DELTAISEMPTY = 1;
```

C++

```
ERRCODE_DELTAISEMPTY = 1;
```

File

DSIntf

Description

This is constant DSIntf.ERRCODE_DELTAISEMPTY.

64.5.83 DSIntf.ERRCODE_DETAILSNOTFETCHED Constant

Pascal

```
ERRCODE_DETAILSNOTFETCHED = 10;
```

C++

```
ERRCODE_DETAILSNOTFETCHED = 10;
```

File

DSIntf

Description

This is constant DSIntf.ERRCODE_DETAILSNOTFETCHED.

64.5.84 DSIntf.ERRCODE_FLYAWAY_WRONGORDER Constant

Pascal

```
ERRCODE_FLYAWAY_WRONGORDER = 13;
```

C++

```
ERRCODE_FLYAWAY_WRONGORDER = 13;
```

File

DSIntf

Description

This is constant DSIntf.ERRCODE_FLYAWAY_WRONGORDER.

64.5.85 DSIntf.ERRCODE_LINKFIELDSNOTUNIQUE Constant

Pascal

```
ERRCODE_LINKFIELDSNOTUNIQUE = 12;
```

C++

```
ERRCODE_LINKFIELDSNOTUNIQUE = 12;
```

File

DSIntf

Description

This is constant DSIntf.ERRCODE_LINKFIELDSNOTUNIQUE.

64.5.86 DSIntf.ERRCODE_NEWERVERSIONREQ Constant

Pascal

```
ERRCODE_NEWERVERSIONREQ = 8;
```

C++

```
ERRCODE_NEWERVERSIONREQ = 8;
```

File

DSIntf

Description

This is constant DSIntf.ERRCODE_NEWERVERSIONREQ.

64.5.87 DSIntf.ERRCODE_NOCASCADEDUPDATES Constant

Pascal

```
ERRCODE_NOCASCADEDUPDATES = 14;
```

C++

```
ERRCODE_NOCASCADEDUPDATES = 14;
```

File

DSIntf

Description

This is constant DSIntf.ERRCODE_NOCASCADEDUPDATES.

64.5.88 DSIntf.ERRCODE_NOMASTERRECORD Constant

Pascal

```
ERRCODE_NOMASTERRECORD = 11;
```

C++

```
ERRCODE_NOMASTERRECORD = 11;
```

File

DSIntf

Description

This is constant DSIntf.ERRCODE_NOMASTERRECORD.

64.5.89 DSIntf.ERRCODE_NOMETADATA Constant

Pascal

```
ERRCODE_NOMETADATA = 3;
```

C++

```
ERRCODE_NOMETADATA = 3;
```

File

DSIntf

Description

This is constant DSIntf.ERRCODE_NOMETADATA.

64.5.90 DSIntf.ERRCODE_NOTHINGTOUNDO Constant

Pascal

```
ERRCODE_NOTHINGTOUNDO = 2;
```

C++

```
ERRCODE_NOTHINGTOUNDO = 2;
```

File

DSIntf

Description

This is constant DSIntf.ERRCODE_NOTHINGTOUNDO.

64.5.91 DSIntf.FieldTypeMap Constant

Pascal

```
FieldTypeMap: TFieldMap = ( fldUNKNOWN, fldZSTRING, fldINT16, fldINT32, fldUINT16, fldBOOL,
fldFLOAT, fldFLOAT, fldBCD, fldDATE, fldTIME, fldTIMESTAMP, fldBYTES, fldVARBYTES, fldINT32,
fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldCURSOR, fldZSTRING,
fldWIDESTRING, fldINT64, fldADT, fldArray, fldREF, fldTABLE, fldBLOB, fldBLOB, fldUNKNOWN,
fldUNKNOWN, fldUNKNOWN, fldZSTRING, fldDATETIME, fldFMTBCD, fldWIDESTRING, fldBLOB,
fldDATETIME, fldZSTRING);
```

C++

```
TFieldMap FieldTypeMap = ( fldUNKNOWN, fldZSTRING, fldINT16, fldINT32, fldUINT16, fldBOOL,
fldFLOAT, fldFLOAT, fldBCD, fldDATE, fldTIME, fldTIMESTAMP, fldBYTES, fldVARBYTES, fldINT32,
fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldBLOB, fldCURSOR, fldZSTRING,
fldWIDESTRING, fldINT64, fldADT, fldArray, fldREF, fldTABLE, fldBLOB, fldBLOB, fldUNKNOWN,
fldUNKNOWN, fldUNKNOWN, fldZSTRING, fldDATETIME, fldFMTBCD, fldWIDESTRING, fldBLOB,
fldDATETIME, fldZSTRING);
```

File

DSIntf

Description

This is constant DSIntf.FieldTypeMap.

64.5.92 DSIntf.fldADT Constant

Pascal

```
fldADT = 20;
```

C++

```
fldADT = 20;
```

File

DSIntf

Description

This is constant DSIntf.fldADT.

64.5.93 DSIntf.fldARRAY Constant

Pascal

```
fldARRAY = 21;
```

C++

```
fldARRAY = 21;
```

File

DSIntf

Description

This is constant DSIntf.fldARRAY.

64.5.94 DSIntf.fldAttrHIDDEN Constant

Pascal

```
fldAttrHIDDEN = $0001;
```

C++

```
fldAttrHIDDEN = 0x0001;
```

File

DSIntf

Description

This is constant DSIntf.fldAttrHIDDEN.

64.5.95 DSIntf.fldAttrLINK Constant

Pascal

```
fldAttrLINK = $0008;
```

C++

```
fldAttrLINK = 0x0008;
```

File

DSIntf

Description

This is constant DSIntf.fldAttrLINK.

64.5.96 DSIntf.fldAttrREADONLY Constant

Pascal

```
fldAttrREADONLY = $0002;
```

C++

```
fldAttrREADONLY = 0x0002;
```

File

DSIntf

Description

This is constant DSIntf.fldAttrREADONLY.

64.5.97 DSIntf.fldAttrREQUIRED Constant

Pascal

```
fldAttrREQUIRED = $0004;
```

C++

```
fldAttrREQUIRED = 0x0004;
```

File

DSIntf

Description

This is constant DSIntf.fldAttrREQUIRED.

64.5.98 DSIntf.fldBCD Constant

Pascal

```
fldBCD = 8;
```

C++

```
fldBCD = 8;
```

File

DSIntf

Description

This is constant DSIntf.fldBCD.

64.5.99 DSIntf.fldBLOB Constant

Pascal

```
fldBLOB = 3;
```

C++

```
fldBLOB = 3;
```

File

DSIntf

Description

This is constant DSIntf.fldBLOB.

64.5.100 DSIntf.fldBOOL Constant

Pascal

```
fldBOOL = 4;
```

C++

```
fldBOOL = 4;
```

File

DSIntf

Description

This is constant DSIntf.fldBOOL.

64.5.101 DSIntf.fldBYTES Constant

Pascal

```
fldBYTES = 9;
```

C++

```
fldBYTES = 9;
```

File

DSIntf

Description

This is constant DSIntf.fldBYTES.

64.5.102 DSIntf.fldCURSOR Constant

Pascal

```
fldCURSOR = 17;
```

C++

```
fldCURSOR = 17;
```

File

DSIntf

Description

This is constant DSIntf.fldCURSOR.

64.5.103 DSIntf.fldDATE Constant

Pascal

```
fldDATE = 2;
```

C++

```
fldDATE = 2;
```

File

DSIntf

Description

This is constant DSIntf.fldDATE.

64.5.104 DSIntf.fldDATETIME Constant

Pascal

```
fldDATETIME = 24;
```

C++

```
fldDATETIME = 24;
```

File

DSIntf

Description

This is constant DSIntf.fldDATETIME.

64.5.105 DSIntf.fldFLOAT Constant

Pascal

```
fldFLOAT = 7;
```

C++

```
fldFLOAT = 7;
```

File

DSIntf

Description

This is constant DSIntf.fldFLOAT.

64.5.106 DSIntf.fldFLOATIEEE Constant

Pascal

```
fldFLOATIEEE = 14;
```

C++

```
fldFLOATIEEE = 14;
```

File

DSIntf

Description

This is constant DSIntf.fldFLOATIEEE.

64.5.107 DSIntf.fldFMTBCD Constant

Pascal

```
fldFMTBCD = 25;
```

C++

```
fldFMTBCD = 25;
```

File

DSIntf

Description

This is constant DSIntf.fldFMTBCD.

64.5.108 DSIntf.fldINT16 Constant

Pascal

```
fldINT16 = 5;
```

C++

```
fldINT16 = 5;
```

File

DSIntf

Description

This is constant DSIntf.fldINT16.

64.5.109 DSIntf.fldINT32 Constant

Pascal

```
fldINT32 = 6;
```

C++

```
fldINT32 = 6;
```

File

DSIntf

Description

This is constant DSIntf.fldINT32.

64.5.110 DSIntf.fldINT64 Constant

Pascal

```
fldINT64 = 18;
```

C++

```
fldINT64 = 18;
```

File

DSIntf

Description

This is constant DSIntf.fldINT64.

64.5.111 DSIntf.fldLOCKINFO Constant

Pascal

```
fldLOCKINFO = 16;
```

C++

```
fldLOCKINFO = 16;
```

File

DSIntf

Description

This is constant DSIntf.fldLOCKINFO.

64.5.112 DSIntf.fldREF Constant

Pascal

```
fldREF = 22;
```

C++

```
fldREF = 22;
```

File

DSIntf

Description

This is constant DSIntf.fldREF.

64.5.113 DSIntf.fldstACCOLEOBJ Constant

Pascal

```
fldstACCOLEOBJ = 30;
```

C++

```
fldstACCOLEOBJ = 30;
```

File

DSIntf

Description

This is constant DSIntf.fldstACCOLEOBJ.

64.5.114 DSIntf.fldstAUTOINC Constant

Pascal

```
fldstAUTOINC = 29;
```

C++

```
fldstAUTOINC = 29;
```

File

DSIntf

Description

This is constant DSIntf.fldstAUTOINC.

64.5.115 DSIntf.fldstBFILE Constant

Pascal

```
fldstBFILE = 36;
```

C++

```
fldstBFILE = 36;
```

File

DSIntf

Description

This is constant DSIntf.fldstBFILE.

64.5.116 DSIntf.fldstBINARY Constant

Pascal

```
fldstBINARY = 23;
```

C++

```
fldstBINARY = 23;
```

File

DSIntf

Description

This is constant DSIntf.fldstBINARY.

64.5.117 DSIntf.fldstDBSOLEOBJ Constant

Pascal

```
fldstDBSOLEOBJ = 27;
```

C++

```
fldstDBSOLEOBJ = 27;
```

File

DSIntf

Description

This is constant DSIntf.fldstDBSOLEOBJ.

64.5.118 DSIntf.fldstFIXED Constant

Pascal

```
fldstFIXED = 31;
```

C++

```
fldstFIXED = 31;
```

File

DSIntf

Description

This is constant DSIntf.fldstFIXED.

64.5.119 DSIntf.fldstFMTMEMO Constant

Pascal

```
fldstFMTMEMO = 24;
```

C++

```
fldstFMTMEMO = 24;
```

File

DSIntf

Description

This is constant DSIntf.fldstFMTMEMO.

64.5.120 DSIntf.fldstGRAPHIC Constant

Pascal

```
fldstGRAPHIC = 26;
```

C++

```
fldstGRAPHIC = 26;
```

File

DSIntf

Description

This is constant DSIntf.fldstGRAPHIC.

64.5.121 DSIntf.fldstGUID Constant

Pascal

```
fldstGUID = 38;
```

C++

```
fldstGUID = 38;
```

File

DSIntf

Description

This is constant DSIntf.fldstGUID.

64.5.122 DSIntf.fldstHBINARY Constant

Pascal

```
fldstHBINARY = 34;
```

C++

```
fldstHBINARY = 34;
```

File

DSIntf

Description

This is constant DSIntf.fldstHBINARY.

64.5.123 DSIntf.fldstHMEMO Constant

Pascal

```
fldstHMEMO = 33;
```

C++

```
fldstHMEMO = 33;
```

File

DSIntf

Description

This is constant DSIntf.fldstHMEMO.

64.5.124 DSIntf.fldstMEMO Constant

Pascal

```
fldstMEMO = 22;
```

C++

```
fldstMEMO = 22;
```

File

DSIntf

Description

This is constant DSIntf.fldstMEMO.

64.5.125 DSIntf.fldstMONEY Constant

Pascal

```
fldstMONEY = 21;
```

C++

```
fldstMONEY = 21;
```

File

DSIntf

Description

This is constant DSIntf.fldstMONEY.

64.5.126 DSIntf.fldstOLEOBJ Constant

Pascal

```
fldstOLEOBJ = 25;
```

C++

```
fldstOLEOBJ = 25;
```

File

DSIntf

Description

This is constant DSIntf.fldstOLEOBJ.

64.5.127 DSIntf.fldstORAINTERVAL Constant

Pascal

```
fldstORAINTERVAL = 40;
```

C++

```
fldstORAINTERVAL = 40;
```

File

DSIntf

Description

This is constant DSIntf.fldstORAINTERVAL.

64.5.128 DSIntf.fldstORATIMESTAMP Constant

Pascal

```
fldstORATIMESTAMP = 39;
```

C++

```
fldstORATIMESTAMP = 39;
```

File

DSIntf

Description

This is constant DSIntf.fldstORATIMESTAMP.

64.5.129 DSIntf.fldstPASSWORD Constant

Pascal

```
fldstPASSWORD = 1;
```

C++

```
fldstPASSWORD = 1;
```

File

DSIntf

Description

This is constant DSIntf.fldstPASSWORD.

64.5.130 DSIntf.fldstReference Constant

Pascal

```
fldstReference = 70;
```

C++

```
fldstReference = 70;
```

File

DSIntf

Description

This is constant DSIntf.fldstReference.

64.5.131 DSIntf.fldstTYPEDBINARY Constant

Pascal

```
fldstTYPEDBINARY = 28;
```

C++

```
fldstTYPEDBINARY = 28;
```

File

DSIntf

Description

This is constant DSIntf.fldstTYPEDBINARY.

64.5.132 DSIntf.fldstWIDEMEMO Constant

Pascal

```
fldstWIDEMEMO = 32;
```

C++

```
fldstWIDEMEMO = 32;
```

File

DSIntf

Description

This is constant DSIntf.fldstWIDEMEMO.

64.5.133 DSIntf.FldSubTypeMap Constant

Pascal

```
FldSubTypeMap: array[TFieldType] of Word = ( 0, 0, 0, 0, 0, 0, 0, fldstMONEY, 0, 0, 0, 0, 0,
0, fldstAUTOINC, fldstBINARY, fldstMEMO, fldstGRAPHIC, fldstFMTMEMO, fldstOLEOBJ,
fldstDBSOLEOBJ, fldstTYPEDBINARY, 0, fldstFIXED, 0, 0, 0, 0, 0, 0, fldstHBINARY, fldstHMEMO,
0, 0, 0, fldstGUID, fldDATETIME, 0, fldstFIXED, fldstWIDEMEMO, fldstORATIMESTAMP,
fldstORAINTERVAL);
```

C++

```
array[TFieldType] of Word FldSubTypeMap = ( 0, 0, 0, 0, 0, 0, 0, fldstMONEY, 0, 0, 0, 0, 0, 0,
fldstAUTOINC, fldstBINARY, fldstMEMO, fldstGRAPHIC, fldstFMTMEMO, fldstOLEOBJ, fldstDBSOLEOBJ,
fldstTYPEDBINARY, 0, fldstFIXED, 0, 0, 0, 0, 0, 0, fldstHBINARY, fldstHMEMO, 0, 0, 0,
fldstGUID, fldDATETIME, 0, fldstFIXED, fldstWIDEMEMO, fldstORATIMESTAMP, fldstORAINTERVAL);
```

File

DSIntf

Description

This is constant DSIntf.FldSubTypeMap.

64.5.134 DSIntf.fldTABLE Constant

Pascal

```
fldTABLE = 23;
```

C++

```
fldTABLE = 23;
```

File

DSIntf

Description

This is constant DSIntf.fldTABLE.

64.5.135 DSIntf.fldTIME Constant

Pascal

```
fldTIME = 10;
```

C++

```
fldTIME = 10;
```

File

DSIntf

Description

This is constant DSIntf.fldTIME.

64.5.136 DSIntf.fldTIMESTAMP Constant

Pascal

```
fldTIMESTAMP = 11;
```

C++

```
fldTIMESTAMP = 11;
```

File

DSIntf

Description

This is constant DSIntf.fldTIMESTAMP.

64.5.137 DSIntf.fldUINT16 Constant

Pascal

```
fldUINT16 = 12;
```

C++

```
fldUINT16 = 12;
```

File

DSIntf

Description

This is constant DSIntf.fldUINT16.

64.5.138 DSIntf.fldUINT32 Constant

Pascal

```
fldUINT32 = 13;
```

C++

```
fldUINT32 = 13;
```

File

DSIntf

Description

This is constant DSIntf.fldUINT32.

64.5.139 DSIntf.fldUINT64 Constant

Pascal

```
fldUINT64 = 19;
```

C++

```
fldUINT64 = 19;
```

File

DSIntf

Description

This is constant DSIntf.fldUINT64.

64.5.140 DSIntf.fldUNICODE Constant

Pascal

```
fldUNICODE = $1007;
```

C++

```
fldUNICODE = 0x1007;
```

File

DSIntf

Description

This is constant DSIntf.fldUNICODE.

64.5.141 DSIntf.fldUNKNOWN Constant

Pascal

```
fldUNKNOWN = 0;
```

C++

```
fldUNKNOWN = 0;
```

File

DSIntf

Description

This is constant DSIntf.fldUNKNOWN.

64.5.142 DSIntf.fldVARBYTES Constant

Pascal

```
fldVARBYTES = 15;
```

C++

```
fldVARBYTES = 15;
```

File

DSIntf

Description

This is constant DSIntf.fldVARBYTES.

64.5.143 DSIntf.fldWIDESTRING Constant

Pascal

```
fldWIDESTRING = 26;
```

C++

```
fldWIDESTRING = 26;
```

File

DSIntf

Description

This is constant DSIntf.fldWIDESTRING.

64.5.144 DSIntf.fldZSTRING Constant

Pascal

```
fldZSTRING = 1;
```

C++

```
fldZSTRING = 1;
```

File

DSIntf

Description

This is constant DSIntf.fldZSTRING.

64.5.145 DSIntf.MAXKEYFIELDS Constant

Pascal

```
MAXKEYFIELDS = 16;
```

C++

```
MAXKEYFIELDS = 16;
```

File

DSIntf

Description

This is constant DSIntf.MAXKEYFIELDS.

64.5.146 DSIntf.MAXLOGFLDTYPES Constant

Pascal

```
MAXLOGFLDTYPES = 27;
```

C++

```
MAXLOGFLDTYPES = 27;
```

File

DSIntf

Description

This is constant DSIntf.MAXLOGFLDTYPES.

64.5.147 DSIntf.mdALLOWLINKCHANGE Constant

Pascal

```
mdALLOWLINKCHANGE = $0010;
```

C++

```
mdALLOWLINKCHANGE = 0x0010;
```

File

DSIntf

Description

This is constant DSIntf.mdALLOWLINKCHANGE.

64.5.148 DSIntf.mdCASCADEDEL Constant

Pascal

```
mdCASCADEDEL = $0004;
```

C++

```
mdCASCADEDEL = 0x0004;
```

File

DSIntf

Description

This is constant DSIntf.mdCASCADEDEL.

64.5.149 DSIntf.mdCASCADEMOD Constant

Pascal

```
mdCASCADEMOD = $0008;
```

C++

```
mdCASCADEMOD = 0x0008;
```

File

DSIntf

Description

This is constant DSIntf.mdCASCADEMOD.

64.5.150 DSIntf.PACKETVERSION_1 Constant

Pascal

```
PACKETVERSION_1 = 1;
```

C++

```
PACKETVERSION_1 = 1;
```

File

DSIntf

Description

This is constant DSIntf.PACKETVERSION_1.

64.5.151 DSIntf.PACKETVERSION_2 Constant

Pascal

```
PACKETVERSION_2 = 2;
```

C++

```
PACKETVERSION_2 = 2;
```

File

DSIntf

Description

This is constant DSIntf.PACKETVERSION_2.

64.5.152 DSIntf.PACKETVERSION_3 Constant

Pascal

```
PACKETVERSION_3 = 3;
```

C++

```
PACKETVERSION_3 = 3;
```

File

DSIntf

Description

This is constant DSIntf.PACKETVERSION_3.

64.5.153 DSIntf.szAUTOINCVALUE Constant

Pascal

```
szAUTOINCVALUE = 'AUTOINCVALUE';
```

C++

```
szAUTOINCVALUE = 'AUTOINCVALUE';
```

File

DSIntf

Description

This is constant DSIntf.szAUTOINCVALUE.

64.5.154 DSIntf.szBDEDEFCL Constant

Pascal

```
szBDEDEFCL = 'BDEDEFAULT_CL';
```

C++

```
szBDEDEFCL = 'BDEDEFAULT_CL';
```

File

DSIntf

Description

This is constant DSIntf.szBDEDEFCL.

64.5.155 DSIntf.szBDEDEFX Constant

Pascal

```
szBDEDEFX = 'BDEDEFAULT_X';
```

C++

```
szBDEDEFX = 'BDEDEFAULT_X';
```

File

DSIntf

Description

This is constant DSIntf.szBDEDEFX.

64.5.156 DSIntf.szBDEDOMCL Constant

Pascal

```
szBDEDOMCL = 'BDEDOMAIN_CL';
```

C++

```
szBDEDOMCL = 'BDEDOMAIN_CL';
```

File

DSIntf

Description

This is constant DSIntf.szBDEDOMCL.

64.5.157 DSIntf.szBDEDOMX Constant

Pascal

```
szBDEDOMX = 'BDEDOMAIN_X';
```

C++

```
szBDEDOMX = 'BDEDOMAIN_X';
```

File

DSIntf

Description

This is constant DSIntf.szBDEDOMX.

64.5.158 DSIntf.szBDERECCL Constant

Pascal

```
szBDERECCL = 'BDERECORD_CL';
```

C++

```
szBDERECCL = 'BDERECORD_CL';
```

File

DSIntf

Description

This is constant DSIntf.szBDERECCL.

64.5.159 DSIntf.szBDERECX Constant

Pascal

```
szBDERECX = 'BDERECORD_X';
```

C++

```
szBDERECX = 'BDERECORD_X';
```

File

DSIntf

Description

This is constant DSIntf.szBDERECX.

64.5.160 DSIntf.szCHANGE_LOG Constant

Pascal

```
szCHANGE_LOG = 'CHANGE_LOG';
```

C++

```
szCHANGE_LOG = 'CHANGE_LOG';
```

File

DSIntf

Description

This is constant DSIntf.szCHANGE_LOG.

64.5.161 DSIntf.szCHANGEINDEX Constant

Pascal

```
szCHANGEINDEX = 'CHANGEINDEX';
```

C++

```
szCHANGEINDEX = 'CHANGEINDEX';
```

File

DSIntf

Description

This is constant DSIntf.szCHANGEINDEX.

64.5.162 DSIntf.szCONSTRAINTS Constant

Pascal

```
szCONSTRAINTS = 'CONSTRAINTS';
```

C++

```
szCONSTRAINTS = 'CONSTRAINTS';
```

File

DSIntf

Description

This is constant DSIntf.szCONSTRAINTS.

64.5.163 DSIntf.szDATASET_CONTEXT Constant

Pascal

```
szDATASET_CONTEXT = 'DATASET_CONTEXT';
```

C++

```
szDATASET_CONTEXT = 'DATASET_CONTEXT';
```

File

DSIntf

Description

This is constant DSIntf.szDATASET_CONTEXT.

64.5.164 DSIntf.szDATASET_DELTA Constant

Pascal

```
szDATASET_DELTA = 'DATASET_DELTA';
```

C++

```
szDATASET_DELTA = 'DATASET_DELTA';
```

File

DSIntf

Description

This is constant DSIntf.szDATASET_DELTA.

64.5.165 DSIntf.szDECIMALS Constant

Pascal

```
szDECIMALS = 'DECIMALS';
```

C++

```
szDECIMALS = 'DECIMALS';
```

File

DSIntf

Description

This is constant DSIntf.szDECIMALS.

64.5.166 DSIntf.szDEFAULT_ORDER Constant

Pascal

```
szDEFAULT_ORDER = 'DEFAULT_ORDER';
```

C++

```
szDEFAULT_ORDER = 'DEFAULT_ORDER';
```

File

DSIntf

Description

This is constant DSIntf.szDEFAULT_ORDER.

64.5.167 DSIntf.szDISABLE_DELETES Constant

Pascal

```
szDISABLE_DELETES = 'DISABLE_DELETES';
```

C++

```
szDISABLE_DELETES = 'DISABLE_DELETES';
```

File

DSIntf

Description

This is constant DSIntf.szDISABLE_DELETES.

64.5.168 DSIntf.szDISABLE_EDITS Constant

Pascal

```
szDISABLE_EDITS = 'DISABLE_EDITS';
```

C++

```
szDISABLE_EDITS = 'DISABLE_EDITS';
```

File

DSIntf

Description

This is constant DSIntf.szDISABLE_EDITS.

64.5.169 DSIntf.szDISABLE_INSERTS Constant

Pascal

```
szDISABLE_INSERTS = 'DISABLE_INSERTS';
```

C++

```
szDISABLE_INSERTS = 'DISABLE_INSERTS';
```

File

DSIntf

Description

This is constant DSIntf.szDISABLE_INSERTS.

64.5.170 DSIntf.szELEMENTS Constant

Pascal

```
szELEMENTS = 'ELEMENTS';
```

C++

```
szELEMENTS = 'ELEMENTS';
```

File

DSIntf

Description

This is constant DSIntf.szELEMENTS.

64.5.171 DSIntf.szFIELDNAME Constant

Pascal

```
szFIELDNAME = 'FIELDNAME';
```

C++

```
szFIELDNAME = 'FIELDNAME';
```

File

DSIntf

Description

This is constant DSIntf.szFIELDNAME.

64.5.172 DSIntf.szFIELDPROPS Constant

Pascal

```
szFIELDPROPS = 'FIELDPROPS';
```

C++

```
szFIELDPROPS = 'FIELDPROPS';
```

File

DSIntf

Description

This is constant DSIntf.szFIELDPROPS.

64.5.173 DSIntf.szLCID Constant

Pascal

```
szLCID = 'LCID';
```

C++

```
szLCID = 'LCID';
```

File

DSIntf

Description

This is constant DSIntf.szLCID.

64.5.174 DSIntf.szMAXVALUE Constant

Pascal

```
szMAXVALUE = 'MAXVALUE';
```

C++

```
szMAXVALUE = 'MAXVALUE';
```

File

DSIntf

Description

This is constant DSIntf.szMAXVALUE.

64.5.175 DSIntf.szMD_FIELDLINKS Constant

Pascal

```
szMD_FIELDLINKS = 'MD_FIELDLINKS';
```

C++

```
szMD_FIELDLINKS = 'MD_FIELDLINKS';
```

File

DSIntf

Description

This is constant DSIntf.szMD_FIELDLINKS.

64.5.176 DSIntf.szMD_SEMANTICS Constant

Pascal

```
szMD_SEMANTICS = 'MD_SEMANTICS';
```

C++

```
szMD_SEMANTICS = 'MD_SEMANTICS';
```

File

DSIntf

Description

This is constant DSIntf.szMD_SEMANTICS.

64.5.177 DSIntf.szMINVALUE Constant

Pascal

```
szMINVALUE = 'MINVALUE';
```

C++

```
szMINVALUE = 'MINVALUE';
```

File

DSIntf

Description

This is constant DSIntf.szMINVALUE.

64.5.178 DSIntf.szNO_RESET_CALL Constant

Pascal

```
szNO_RESET_CALL = 'NO_RESET_CALL';
```

C++

```
szNO_RESET_CALL = 'NO_RESET_CALL';
```

File

DSIntf

Description

This is constant DSIntf.szNO_RESET_CALL.

64.5.179 DSIntf.szORIGIN Constant

Pascal

```
szORIGIN = 'ORIGIN';
```

C++

```
szORIGIN = 'ORIGIN';
```

File

DSIntf

Description

This is constant DSIntf.szORIGIN.

64.5.180 DSIntf.szPRIMARY_KEY Constant

Pascal

```
szPRIMARY_KEY = 'PRIMARY_KEY';
```

C++

```
szPRIMARY_KEY = 'PRIMARY_KEY';
```

File

DSIntf

Description

This is constant DSIntf.szPRIMARY_KEY.

64.5.181 DSIntf.szPROVFLAGS Constant

Pascal

```
szPROVFLAGS = 'PROVFLAGS';
```

C++

```
szPROVFLAGS = 'PROVFLAGS';
```

File

DSIntf

Description

This is constant DSIntf.szPROVFLAGS.

64.5.182 DSIntf.szREADONLY Constant

Pascal

```
szREADONLY = 'READONLY';
```

C++

```
szREADONLY = 'READONLY';
```

File

DSIntf

Description

This is constant DSIntf.szREADONLY.

64.5.183 DSIntf.szSERVER_COL Constant

Pascal

```
szSERVER_COL = 'SERVER_COL';
```

C++

```
szSERVER_COL = 'SERVER_COL';
```

File

DSIntf

Description

This is constant DSIntf.szSERVER_COL.

64.5.184 DSIntf.szSERVERCALC Constant

Pascal

```
szSERVERCALC = 'SERVER_CALC';
```

C++

```
szSERVERCALC = 'SERVER_CALC';
```

File

DSIntf

Description

This is constant DSIntf.szSERVERCALC.

64.5.185 DSIntf.szstACCOLEOBJ Constant

Pascal

```
szstACCOLEOBJ = 'AccessOle';
```

C++

```
szstACCOLEOBJ = 'AccessOle';
```

File

DSIntf

Description

This is constant DSIntf.szstACCOLEOBJ.

64.5.186 DSIntf.szstADTNESTEDTABLE Constant

Pascal

```
szstADTNESTEDTABLE = 'ADTNestedTable';
```

C++

```
szstADTNESTEDTABLE = 'ADTNestedTable';
```

File

DSIntf

Description

This is constant DSIntf.szstADTNESTEDTABLE.

64.5.187 DSIntf.szstAUTOINC Constant

Pascal

```
szstAUTOINC = 'Autoinc';
```

C++

```
szstAUTOINC = 'Autoinc';
```

File

DSIntf

Description

This is constant DSIntf.szstAUTOINC.

64.5.188 DSIntf.szstBINARY Constant

Pascal

```
szstBINARY = 'Binary';
```

C++

```
szstBINARY = 'Binary';
```

File

DSIntf

Description

This is constant DSIntf.szstBINARY.

64.5.189 DSIntf.szstDBSOLEOBJ Constant

Pascal

```
szstDBSOLEOBJ = 'dBASEOle';
```

C++

```
szstDBSOLEOBJ = 'dBASEOle';
```

File

DSIntf

Description

This is constant DSIntf.szstDBSOLEOBJ.

64.5.190 DSIntf.szstFIXEDCHAR Constant

Pascal

```
szstFIXEDCHAR = 'FixedChar';
```

C++

```
szstFIXEDCHAR = 'FixedChar';
```

File

DSIntf

Description

This is constant DSIntf.szstFIXEDCHAR.

64.5.191 DSIntf.szstFMTMEMO Constant

Pascal

```
szstFMTMEMO = 'Formatted';
```

C++

```
szstFMTMEMO = 'Formatted';
```

File

DSIntf

Description

This is constant DSIntf.szstFMTMEMO.

64.5.192 DSIntf.szstGRAPHIC Constant

Pascal

```
szstGRAPHIC = 'Graphics';
```

C++

```
szstGRAPHIC = 'Graphics';
```

File

DSIntf

Description

This is constant DSIntf.szstGRAPHIC.

64.5.193 DSIntf.szstGUID Constant

Pascal

```
szstGUID = 'Guid';
```

C++

```
szstGUID = 'Guid';
```

File

DSIntf

Description

This is constant DSIntf.szstGUID.

64.5.194 DSIntf.szstHBINARY Constant

Pascal

```
szstHBINARY = 'HBinary';
```

C++

```
szstHBINARY = 'HBinary';
```

File

DSIntf

Description

This is constant DSIntf.szstHBINARY.

64.5.195 DSIntf.szstHMEMO Constant

Pascal

```
szstHMEMO = 'HMemo';
```

C++

```
szstHMEMO = 'HMemo';
```

File

DSIntf

Description

This is constant DSIntf.szstHMEMO.

64.5.196 DSIntf.szstMEMO Constant

Pascal

```
szstMEMO = 'Text';
```

C++

```
szstMEMO = 'Text';
```

File

DSIntf

Description

This is constant DSIntf.szstMEMO.

64.5.197 DSIntf.szstMONEY Constant

Pascal

```
szstMONEY = 'Money';
```

C++

```
szstMONEY = 'Money';
```

File

DSIntf

Description

This is constant DSIntf.szstMONEY.

64.5.198 DSIntf.szstOLEOBJ Constant

Pascal

```
szstOLEOBJ = 'Ole';
```

C++

```
szstOLEOBJ = 'Ole';
```

File

DSIntf

Description

This is constant DSIntf.szstOLEOBJ.

64.5.199 DSIntf.szstREFNESTEDTABLE Constant

Pascal

```
szstREFNESTEDTABLE = 'Reference';
```

C++

```
szstREFNESTEDTABLE = 'Reference';
```

File

DSIntf

Description

This is constant DSIntf.szstREFNESTEDTABLE.

64.5.200 DSIntf.szstTYPEDBINARY Constant

Pascal

```
szstTYPEDBINARY = 'TypedBinary';
```

C++

```
szstTYPEDBINARY = 'TypedBinary';
```

File

DSIntf

Description

This is constant DSIntf.szstTYPEDBINARY.

64.5.201 DSIntf.szstWideMEMO Constant

Pascal

```
szstWideMEMO = 'WideText';
```

C++

```
szstWideMEMO = 'WideText';
```

File

DSIntf

Description

This is constant DSIntf.szstWideMEMO.

64.5.202 DSIntf.szSUBTYPE Constant

Pascal

```
szSUBTYPE = 'SUBTYPE';
```

C++

```
szSUBTYPE = 'SUBTYPE';
```

File

DSIntf

Description

This is constant DSIntf.szSUBTYPE.

64.5.203 DSIntf.szTABLE_NAME Constant

Pascal

```
szTABLE_NAME = 'TABLE_NAME';
```

C++

```
szTABLE_NAME = 'TABLE_NAME';
```

File

DSIntf

Description

This is constant DSIntf.szTABLE_NAME.

64.5.204 DSIntf.szTYPENAME Constant

Pascal

```
szTYPENAME = 'TYPENAME';
```

C++

```
szTYPENAME = 'TYPENAME';
```

File

DSIntf

Description

This is constant DSIntf.szTYPENAME.

64.5.205 DSIntf.szUNIQUE_KEY Constant

Pascal

```
szUNIQUE_KEY = 'UNIQUE_KEY';
```

C++

```
szUNIQUE_KEY = 'UNIQUE_KEY';
```

File

DSIntf

Description

This is constant DSIntf.szUNIQUE_KEY.

64.5.206 DSIntf.szUPDATEMODE Constant

Pascal

```
szUPDATEMODE = 'UPDATEMODE';
```

C++

```
szUPDATEMODE = 'UPDATEMODE';
```

File

DSIntf

Description

This is constant DSIntf.szUPDATEMODE.

64.5.207 DSIntf.szWIDTH Constant

Pascal

```
szWIDTH = 'WIDTH';
```

C++

```
szWIDTH = 'WIDTH';
```

File

DSIntf

Description

This is constant DSIntf.szWIDTH.

64.5.208 DSIntf.xmlOFF Constant

Pascal

```
xmlOFF = 0;
```

C++

```
xmlOFF = 0;
```

File

DSIntf

Description

This is constant DSIntf.xmlOFF.

64.5.209 DSIntf.xmlON Constant

Pascal

```
xmlON = xmlXMLSCHEMA or xmlUNTYPED;
```

C++

```
xmlON = xmlXMLSCHEMA or xmlUNTYPED;
```

File

DSIntf

Description

This is constant DSIntf.xmlON.

64.5.210 DSIntf.xmlUNTYPED Constant

Pascal

```
xmlUNTYPED = 1;
```

C++

```
xmlUNTYPED = 1;
```

File

DSIntf

Description

This is constant DSIntf.xmlUNTYPED.

64.5.211 DSIntf.xmlUTF8 Constant

Pascal

```
xmlUTF8 = xmlON or xmlXMLUTF8;
```

C++

```
xmlUTF8 = xmlON or xmlXMLUTF8;
```

File

DSIntf

Description

This is constant DSIntf.xmlUTF8.

64.5.212 DSIntf.xmlXMLDATATYPED Constant

Pascal

```
xmlXMLDATATYPED = 2;
```

C++

```
xmlXMLDATATYPED = 2;
```

File

DSIntf

Description

This is constant DSIntf.xmlXMLDATATYPED.

64.5.213 DSIntf.xmlXMLSCHEMA Constant

Pascal

```
xmlXMLSCHEMA = 4;
```

C++

```
xmlXMLSCHEMA = 4;
```

File

DSIntf

Description

This is constant DSIntf.xmlXMLSCHEMA.

64.5.214 DSIntf.xmlXMLUTF8 Constant

Pascal

```
xmlXMLUTF8 = 8;
```

C++

```
xmlXMLUTF8 = 8;
```

File

DSIntf

Description

This is constant DSIntf.xmlXMLUTF8.

65 DSProd Namespace

This is namespace DSProd.

Topics

Name	Description
Classes (see page 11800)	The following table lists classes in this documentation.

Classes

	Name	Description
	TCustomDataSetPageProducer (see page 11800)	TCustomDataSetPageProducer is the ancestor of the data set page producer classes.
	TDataSetPageProducer (see page 11807)	TDataSetPageProducer produces a string of HTML commands based on an input template.

65.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCustomDataSetPageProducer (see page 11800)	TCustomDataSetPageProducer is the ancestor of the data set page producer classes.
	TDataSetPageProducer (see page 11807)	TDataSetPageProducer produces a string of HTML commands based on an input template.

65.1.1 TCustomDataSetPageProducer Class

TCustomDataSetPageProducer is the ancestor of the data set page producer classes.

Class Hierarchy

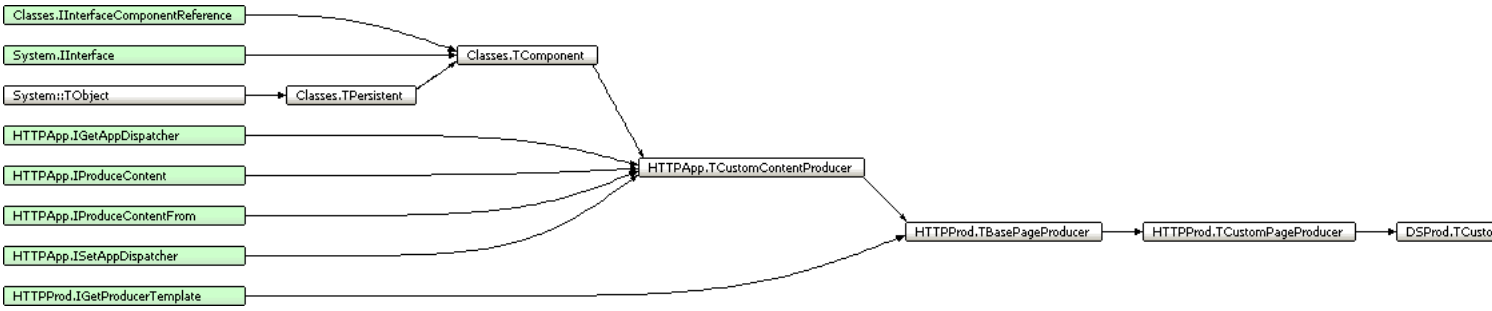

Pascal

```
TCustomDataSetPageProducer = class(TCustomPageProducer);
```

C++

```
class TCustomDataSetPageProducer : public TCustomPageProducer;
```

File

DSProd

Description

Do not create instances of TCustomDataSetPageProducer. Use TCustomDataSetPageProducer as a base class when declaring non-visual components that can appear on the component palette and be used in the form designer. Properties and methods of TCustomDataSetPageProducer provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TDataSetPageProducer (see page 11807)

TPageProducer (see page 15309)

65.1.1.1 TCustomDataSetPageProducer Members

The following tables list the members exposed by TCustomDataSetPageProducer.

Protected Events

	Name	Description
	OnHTMLTag (see page 15280)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoTagEvent (see page 11805)	This is DoTagEvent, a member of class TCustomDataSetPageProducer.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDataSet (see page 11805)	This is GetDataSet, a member of class TCustomDataSetPageProducer.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetProducerTemplateFileName (see page 15276)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.
	GetProducerTemplateStream (see page 15276)	This is GetProducerTemplateStream, a member of class TBasePageProducer.
V	GetScriptEngine (see page 15277)	This is GetScriptEngine, a member of class TBasePageProducer.
	GetTagID (see page 15277)	This is GetTagID, a member of class TBasePageProducer.
V	GetTemplateFileName (see page 15277)	This is GetTemplateFileName, a member of class TBasePageProducer.
V	GetTemplateFileName (see page 15286)	This is GetTemplateFileName, a member of class TCustomPageProducer.
V	GetTemplateStream (see page 15277)	This is GetTemplateStream, a member of class TBasePageProducer.
V	GetTemplateStream (see page 15286)	This is GetTemplateStream, a member of class TCustomPageProducer.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleScriptTag (see page 15277)	This is HandleScriptTag, a member of class TBasePageProducer.
V	HandleTag (see page 15278)	This is HandleTag, a member of class TBasePageProducer.
V	HandleTag (see page 15286)	This is HandleTag, a member of class TCustomPageProducer.
	ImplHandleTag (see page 15278)	This is ImplHandleTag, a member of class TBasePageProducer.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 11805)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.

	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ServerScriptFromStream (see page 15278)	This is ServerScriptFromStream, a member of class TBasePageProducer.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 11806)	This is SetDataSet, a member of class TCustomDataSetPageProducer.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UseScriptEngine (see page 15278)	This is UseScriptEngine, a member of class TBasePageProducer.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 11804)	Returns the translated HTML template.
	ContentFromStream (see page 15107)	Returns the content produced in response to input read from a stream.
	ContentFromStream (see page 15275)	Returns the content produced in response to input read from a stream.
	ContentFromString (see page 15107)	Returns the content produced in response to an input string.
	ContentFromString (see page 15275)	Returns the content produced in response to an input string.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 15274)	Creates an instance of TBasePageProducer (see page 15269).
	Create (see page 15285)	Creates an instance of TCustomPageProducer (see page 15281).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 15285)	Destroys an instance of TCustomPageProducer (see page 15281).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProducerTemplateFileName (see page 15320)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.
	GetProducerTemplateStream (see page 15320)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DataSet (see page 11806)	Refers to the page producer's DataSet object.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.
	HTMLDoc (see page 15287)	Specifies the HTML template used to generate the value returned by the Content (see page 15274) method.
	HTMLFile (see page 15287)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content (see page 15274) method.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ScriptEngine (see page 15279)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 15279)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebModuleContext (see page 15279)	Provides access to the Web module context object.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

65.1.1.2 TCustomDataSetPageProducer Methods

The methods of the TCustomDataSetPageProducer class are listed here.

Protected Methods

	Name	Description
V	DoTagEvent (see page 11805)	This is DoTagEvent, a member of class TCustomDataSetPageProducer.
V	GetDataSet (see page 11805)	This is GetDataSet, a member of class TCustomDataSetPageProducer.
V	Notification (see page 11805)	Overrides the inherited method to respond to messages that the dispatcher is going away.
V	SetDataSet (see page 11806)	This is SetDataSet, a member of class TCustomDataSetPageProducer.

Public Methods

	Name	Description
V	Content (see page 11804)	Returns the translated HTML template.

65.1.1.2.1 TCustomDataSetPageProducer.Content Method

Returns the translated HTML template.

Pascal

```
function Content: string; override;
```

C++

```
virtual __fastcall AnsiString Content();
```

Description

Call Content to get the HTML commands that result from converting the HTML template. Content reads the template from HTMLDoc or HTMLFile, depending on which property is set. Content replaces each tag in the template that has the form <#Fieldname> with the value of the field in DataSet (see page 11806) that has the specified name. Any HTML-transparent tags in the template that do not have tag names which match a field in DataSet (see page 11806) are passed on to the OnHTMLTag event handler.

See Also

DataSet (see page 11806)

OnHTMLTag (see page 15280)

ContentFromString (see page 15275)

ContentFromStream (see page 15275)

HTMLDoc (see page 15287)

HTMLFile (see page 15287)

65.1.1.2.2 TCustomDataSetPageProducer.DoTagEvent Method

Pascal

```
procedure DoTagEvent(Tag: TTag; const TagString: string; TagParams: TStrings; var ReplaceText:
string); override;
```

C++

```
virtual __fastcall DoTagEvent(TTag Tag, const AnsiString TagString, TStrings TagParams,
AnsiString ReplaceText);
```

Description

This is DoTagEvent, a member of class TCustomDataSetPageProducer.

65.1.1.2.3 TCustomDataSetPageProducer.GetDataSet Method

Pascal

```
function GetDataSet: TDataSet; virtual;
```

C++

```
virtual __fastcall TDataSet GetDataSet();
```

Description

This is GetDataSet, a member of class TCustomDataSetPageProducer.

65.1.1.2.4 TCustomDataSetPageProducer.Notification Method

Overrides the inherited method to respond to messages that the dispatcher is going away.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

After calling the inherited method, Notification checks whether this call was triggered by the removal of the dispatcher. If so, the Dispatcher property is set to nil. NULL.

See Also

DispatcherComponent (see page 15110)

Dispatcher (see page 15109)

65.1.1.2.5 TCustomDataSetPageProducer.SetDataSet Method

Pascal

```
procedure SetDataSet(ADataSet: TDataSet); virtual;
```

C++

```
virtual __fastcall SetDataSet(TDataSet ADataSet);
```

Description

This is SetDataSet, a member of class TCustomDataSetPageProducer.

65.1.1.3 TCustomDataSetPageProducer Properties

The properties of the TCustomDataSetPageProducer class are listed here.

Public Properties

	Name	Description
	DataSet (see page 11806)	Refers to the page producer's DataSet object.

65.1.1.3.1 TCustomDataSetPageProducer.DataSet Property

Refers to the page producer's DataSet object.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

DataSet is a reference to the page producer's data set. Use this property to directly access the data set object, which is derived from TDataSet.

See Also

TDataSet (see page 6894)

Using Dataset Page Producers

65.1.2 TDataSetPageProducer Class

TDataSetPageProducer produces a string of HTML commands based on an input template.

Class Hierarchy

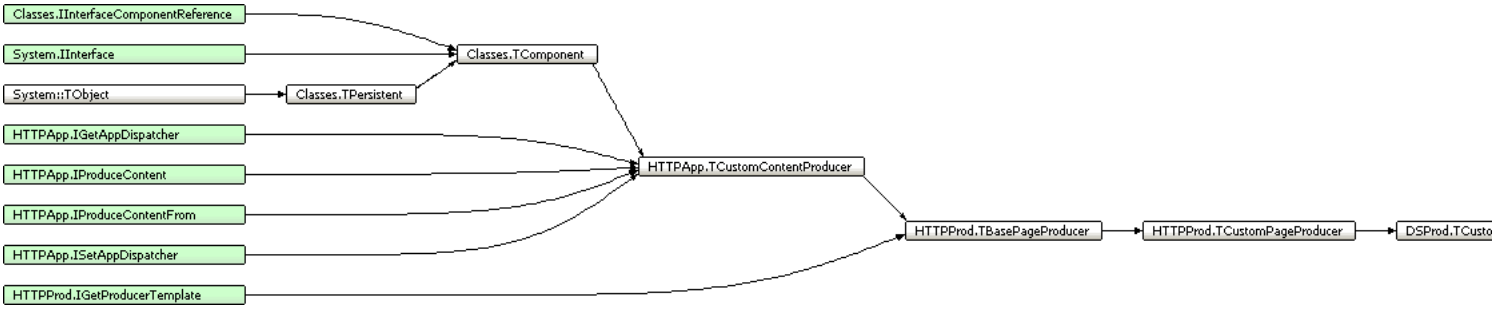

Pascal

```
TDataSetPageProducer = class(TCustomDataSetPageProducer);
```

C++

```
class TDataSetPageProducer : public TCustomDataSetPageProducer;
```

File

DSProd

Description

Use TDataSetPageProducer to convert an HTML template that includes field data into a string of HTML commands that can be interpreted by a client application such as a Web browser. HTML templates include HTML commands and HTML-transparent tags that are replaced with customized content by the dataset page producer in an OnHTMLTag (see page 11814) event handler.

Dataset page producers differ from other page producer components in that they replace HTML-transparent tags that match fields in a dataset with the current value of those fields.

See Also

TDataSetTableProducer (see page 10277)

TQueryTableProducer (see page 8152)

TSQLQueryTableProducer (see page 10842)

TPageProducer (see page 15309)

TAdapterPageProducer (see page 19802)

TInetXPageProducer (see page 17145)

The Web Module

Using Dataset Page Producers

65.1.2.1 TDataSetPageProducer Members

The following tables list the members exposed by TDataSetPageProducer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoTagEvent (see page 11805)	This is DoTagEvent, a member of class TCustomDataSetPageProducer.
	DoTagEvent (see page 15276)	This is DoTagEvent, a member of class TBasePageProducer.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDataSet (see page 11805)	This is GetDataSet, a member of class TCustomDataSetPageProducer.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetProducerTemplateFileName (see page 15276)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.
	GetProducerTemplateStream (see page 15276)	This is GetProducerTemplateStream, a member of class TBasePageProducer.
V	GetScriptEngine (see page 15277)	This is GetScriptEngine, a member of class TBasePageProducer.
	GetTagID (see page 15277)	This is GetTagID, a member of class TBasePageProducer.
V	GetTemplateFileName (see page 15277)	This is GetTemplateFileName, a member of class TBasePageProducer.
V	GetTemplateFileName (see page 15286)	This is GetTemplateFileName, a member of class TCustomPageProducer.
V	GetTemplateStream (see page 15277)	This is GetTemplateStream, a member of class TBasePageProducer.
V	GetTemplateStream (see page 15286)	This is GetTemplateStream, a member of class TCustomPageProducer.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleScriptTag (see page 15277)	This is HandleScriptTag, a member of class TBasePageProducer.
V	HandleTag (see page 15278)	This is HandleTag, a member of class TBasePageProducer.
V	HandleTag (see page 15286)	This is HandleTag, a member of class TCustomPageProducer.
	ImplHandleTag (see page 15278)	This is ImplHandleTag, a member of class TBasePageProducer.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 11805)	Overrides the inherited method to respond to messages that the dispatcher is going away.
V	Notification (see page 15108)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.

	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ServerScriptFromStream (see page 15278)	This is ServerScriptFromStream, a member of class TBasePageProducer.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetDataSet (see page 11806)	This is SetDataSet, a member of class TCustomDataSetPageProducer.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
V	UseScriptEngine (see page 15278)	This is UseScriptEngine, a member of class TBasePageProducer.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Content (see page 11804)	Returns the translated HTML template.
V	Content (see page 15106)	Returns the content produced by the content producer.
V	Content (see page 15274)	Returns the content produced by the content producer.
V	ContentFromStream (see page 15107)	Returns the content produced in response to input read from a stream.
V	ContentFromStream (see page 15275)	Returns the content produced in response to input read from a stream.
V	ContentFromString (see page 15107)	Returns the content produced in response to an input string.
V	ContentFromString (see page 15275)	Returns the content produced in response to an input string.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 15274)	Creates an instance of TBasePageProducer (see page 15269).
V	Create (see page 15285)	Creates an instance of TCustomPageProducer (see page 15281).

	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 15285)	Destroys an instance of TCustomPageProducer (see page 15281).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProducerTemplateFileName (see page 15320)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.
	GetProducerTemplateStream (see page 15320)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	StripParamQuotes (see page 15279)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebModuleContext (see page 15279)	Provides access to the Web module context object.

Published Events

	Name	Description
	OnHTMLTag (see page 11814)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Published Properties

	Name	Description
	DataSet (see page 11811)	Refers to the page producer's DataSet object.
	HTMLDoc (see page 11812)	Specifies the HTML template used to generate the value returned by the Content method.
	HTMLFile (see page 11812)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ScriptEngine (see page 11813)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	Tag (see page 2808)	Stores an integer value as part of a component.

65.1.2.2 TDataSetPageProducer Properties

The properties of the TDataSetPageProducer class are listed here.

Published Properties

	Name	Description
	DataSet (see page 11811)	Refers to the page producer's DataSet object.
	HTMLDoc (see page 11812)	Specifies the HTML template used to generate the value returned by the Content method.
	HTMLFile (see page 11812)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	ScriptEngine (see page 11813)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.

65.1.2.2.1 TDataSetPageProducer.DataSet Property

Refers to the page producer's DataSet object.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

DataSet is a reference to the page producer's data set. Use this property to directly access the data set object, which is derived from TDataSet.

See Also

TDataSet (see page 6894)

Using Dataset Page Producers

65.1.2.2.2 TDataSetPageProducer.HTMLDoc Property

Specifies the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLDoc: TStrings;
```

C++

```
__property TStrings HTMLDoc;
```

Description

Set HTMLDoc to a set of strings that make up the HTML template. Each string is a sequence of one or more HTML commands or HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLDoc clears the HTMLFile property.

See Also

Content (see page 15274)

HTMLFile (see page 15287)

Specifying the HTML Template

65.1.2.2.3 TDataSetPageProducer.HTMLFile Property

Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLFile: TFileName;
```

C++

```
__property TFileName HTMLFile;
```

Description

Set HTMLFile to the name of the file that contains an HTML template. The HTML template is a sequence of HTML commands and HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLFile clears the HTMLDoc property.

Note: When distributing applications that contain page producers which use the HTMLFile property, be sure to distribute the file along with the application. Also, be sure that the value of HTMLFile does not include a hard-coded path name that will not be available on the user's disk.

See Also

Content (see page 15274)

HTMLDoc (see page 15287)

Specifying the HTML Template

65.1.2.2.4 TDataSetPageProducer.ScriptEngine Property

Names the script engine that generates or interprets script embedded in the documents that this producer creates.

Pascal

```
property ScriptEngine: string;
```

C++

```
__property AnsiString ScriptEngine;
```

Description

ScriptEngine indicates the scripting language that the producer uses in its generated documents. For example, if the producer generates code that includes jscript, the value of ScriptEngine is 'JSCRIPT'.

When ScriptEngine is defined, the page producer generates its content using a special script-producer component. When ScriptEngine is not defined, the page producer uses only the OnHTMLTag event, plus any other built-in translations provided by a descendant class.

See Also

OnHTMLTag (see page 15280)

65.1.2.3 TDataSetPageProducer Events

The events of the TDataSetPageProducer class are listed here.

Published Events

	Name	Description
	OnHTMLTag (see page 11814)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

65.1.2.3.1 TDataSetPageProducer.OnHTMLTag Event

Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Pascal

```
property OnHTMLTag: THTMLTagEvent;
```

C++

```
__property THTMLTagEvent OnHTMLTag;
```

Description

Write an OnHTMLTag event handler to convert an HTML-transparent tag into the corresponding HTML commands that can be interpreted by a Web browser. If an OnHTMLTag event is not supplied, the HTML-transparent tags in the HTML template will be replaced by empty strings.

Note: The sequence of HTML that the event handler produces for an HTML-transparent tag can contain HTML-transparent tags for another page producer to convert. For example, one page producer might have a template that represents the format of the final HTTP response. It could read the HTTP request and for each section in the request assemble a set of HTML-transparent tags with parameters based on the request. Another page producer might take the content of the first page producer, and interpret those parameters.

See Also

Names (see page 3075)

Values (see page 3086)

Customizing the InternetExpress Page Producer Template

Chaining Page Producers Together

Converting HTML-transparent Tags

66 ExptIntf Namespace

This is namespace ExptIntf.

Topics

Name	Description
Classes (see page 11817)	The following table lists classes in this documentation.
Functions (see page 11821)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 11822)	The following table lists structs, records, enums in this documentation.
Types (see page 11823)	The following table lists types in this documentation.
Variables (see page 11825)	The following table lists variables in this documentation.
Constants (see page 11826)	The following table lists constants in this documentation.

Classes

	Name	Description
	TIExpert (see page 11817)	This is class ExptIntf.TIExpert.

Constants

Name	Description
ExpertEntryPoint (see page 11826)	This is constant ExptIntf.ExpertEntryPoint.
isExperts (see page 11826)	This is constant ExptIntf.isExperts.
ValidExpertVersion (see page 11826)	This is constant ExptIntf.ValidExpertVersion.

Functions

	Name	Description
	RegisterLibraryExpert (see page 11821)	Registers an expert so that it is installed when a package is installed.

Structs, Records, Enums

	Name	Description
	TExpertStyle (see page 11822)	This is record ExptIntf.TExpertStyle.

Types

Name	Description
TExpertInitProc (see page 11823)	This is type ExptIntf.TExpertInitProc.
TExpertRegisterProc (see page 11823)	This is type ExptIntf.TExpertRegisterProc.
TExpertState (see page 11824)	This is type ExptIntf.TExpertState.
TExpertTerminateProc (see page 11824)	This is type ExptIntf.TExpertTerminateProc.

Variables

Name	Description
LibraryExpertProc (see page 11825)	This is variable ExptIntf.LibraryExpertProc.
ToolServices (see page 11825)	This is variable ExptIntf.ToolServices.

66.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TIExpert (see page 11817)	This is class ExptIntf.TIExpert.

66.1.1 TIExpert Class

Class Hierarchy

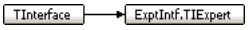

Pascal

```
TIExpert = class(TInterface);
```

C++

```
class TIExpert : public TInterface;
```

File

ExptIntf

Description

This is class ExptIntf.TIExpert.

66.1.1.1 TIExpert Members

The following tables list the members exposed by TIExpert.

Public Methods

	Name	Description
	Execute (see page 11818)	This is Execute, a member of class TIExpert.
	GetAuthor (see page 11818)	This is GetAuthor, a member of class TIExpert.
	GetComment (see page 11818)	This is GetComment, a member of class TIExpert.
	GetGlyph (see page 11818)	This is GetGlyph, a member of class TIExpert.
	GetIDString (see page 11819)	This is GetIDString, a member of class TIExpert.
	GetMenuText (see page 11819)	This is GetMenuText, a member of class TIExpert.
	GetName (see page 11819)	This is GetName, a member of class TIExpert.
	GetPage (see page 11819)	This is GetPage, a member of class TIExpert.
	GetState (see page 11820)	This is GetState, a member of class TIExpert.
	GetStyle (see page 11820)	This is GetStyle, a member of class TIExpert.

66.1.1.2 TIExpert Methods

The methods of the TIExpert class are listed here.

Public Methods

	Name	Description
	Execute (see page 11818)	This is Execute, a member of class TIExpert.
	GetAuthor (see page 11818)	This is GetAuthor, a member of class TIExpert.
	GetComment (see page 11818)	This is GetComment, a member of class TIExpert.
	GetGlyph (see page 11818)	This is GetGlyph, a member of class TIExpert.
	GetIDString (see page 11819)	This is GetIDString, a member of class TIExpert.
	GetMenuText (see page 11819)	This is GetMenuText, a member of class TIExpert.
	GetName (see page 11819)	This is GetName, a member of class TIExpert.
	GetPage (see page 11819)	This is GetPage, a member of class TIExpert.
	GetState (see page 11820)	This is GetState, a member of class TIExpert.
	GetStyle (see page 11820)	This is GetStyle, a member of class TIExpert.

66.1.1.2.1 TIExpert.Execute Method

Pascal

```
procedure Execute; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall Execute() = 0;
```

Description

This is Execute, a member of class TIExpert.

66.1.1.2.2 TIExpert.GetAuthor Method

Pascal

```
function GetAuthor: string; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall AnsiString GetAuthor() = 0;
```

Description

This is GetAuthor, a member of class TIExpert.

66.1.1.2.3 TIExpert.GetComment Method

Pascal

```
function GetComment: string; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall AnsiString GetComment() = 0;
```

Description

This is GetComment, a member of class TIExpert.

66.1.1.2.4 TIExpert.GetGlyph Method

Pascal

```
function GetGlyph: HICON; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall HICON GetGlyph() = 0;
```

Description

This is GetGlyph, a member of class TIExpert.

66.1.1.2.5 TIExpert.GetIDString Method

Pascal

```
function GetIDString: string; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall AnsiString GetIDString() = 0;
```

Description

This is GetIDString, a member of class TIExpert.

66.1.1.2.6 TIExpert.GetMenuText Method

Pascal

```
function GetMenuText: string; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall AnsiString GetMenuText() = 0;
```

Description

This is GetMenuText, a member of class TIExpert.

66.1.1.2.7 TIExpert.GetName Method

Pascal

```
function GetName: string; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall AnsiString GetName() = 0;
```

Description

This is GetName, a member of class TIExpert.

66.1.1.2.8 TIExpert.GetPage Method

Pascal

```
function GetPage: string; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall AnsiString GetPage() = 0;
```

Description

This is GetPage, a member of class TIExpert.

66.1.1.2.9 TIExpert.GetState Method

Pascal

```
function GetState: TExpertState; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall TExpertState GetState() = 0;
```

Description

This is GetState, a member of class TIExpert.

66.1.1.2.10 TIExpert.GetStyle Method

Pascal

```
function GetStyle: TExpertStyle; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall TExpertStyle GetStyle() = 0;
```

Description

This is GetStyle, a member of class TIExpert.

66.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	RegisterLibraryExpert (see page 11821)	Registers an expert so that it is installed when a package is installed.

66.2.1 ExptIntf.RegisterLibraryExpert Function

Registers an expert so that it is installed when a package is installed.

Pascal

```
procedure RegisterLibraryExpert(Expert: TIExpert);
```

C++

```
RegisterLibraryExpert(TIExpert Expert);
```

File

ExptIntf

Description

Call RegisterLibrary to register an expert for a package so that it appears in the New Items dialog. If the package already has a registered expert, RegisterLibraryExpert does nothing.

66.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TExpertStyle (see page 11822)	This is record ExptIntf.TExpertStyle.

66.3.1 ExptIntf.TExpertStyle Enumeration

Pascal

```
TExpertStyle = (
  esStandard,
  esForm,
  esProject,
  esAddIn
);
```

C++

```
enum TExpertStyle {
  esStandard,
  esForm,
  esProject,
  esAddIn
};
```

File

ExptIntf

Description

This is record ExptIntf.TExpertStyle.

66.4 Types

The following table lists types in this documentation.

Types

Name	Description
TExpertInitProc (see page 11823)	This is type ExptIntf.TExpertInitProc.
TExpertRegisterProc (see page 11823)	This is type ExptIntf.TExpertRegisterProc.
TExpertState (see page 11824)	This is type ExptIntf.TExpertState.
TExpertTerminateProc (see page 11824)	This is type ExptIntf.TExpertTerminateProc.

66.4.1 ExptIntf.TExpertInitProc Type

Pascal

```
TExpertInitProc = function (ToolServices: TIToolServices; RegisterProc: TExpertRegisterProc;
var Terminate: TExpertTerminateProc): Boolean stdcall;
```

C++

```
(ToolServices: TIToolServices; RegisterProc: TExpertRegisterProc; var Terminate:
TExpertTerminateProc): Boolean stdcall ( TExpertInitProc)();
```

File

ExptIntf

Description

This is type ExptIntf.TExpertInitProc.

66.4.2 ExptIntf.TExpertRegisterProc Type

Pascal

```
TExpertRegisterProc = function (Expert: TIExpert): Boolean;
```

C++

```
(Expert: TIExpert): Boolean ( TExpertRegisterProc)();
```

File

ExptIntf

Description

This is type ExptIntf.TExpertRegisterProc.

66.4.3 ExptIntf.TExpertState Type

Pascal

```
TExpertState = set of (esEnabled, esChecked);
```

C++

```
(esEnabled, esChecked) TExpertState;
```

File

ExptIntf

Description

This is type ExptIntf.TExpertState.

66.4.4 ExptIntf.TExpertTerminateProc Type

Pascal

```
TExpertTerminateProc = procedure;
```

C++

```
( TExpertTerminateProc)();
```

File

ExptIntf

Description

This is type ExptIntf.TExpertTerminateProc.

66.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
LibraryExpertProc (see page 11825)	This is variable ExptIntf.LibraryExpertProc.
ToolServices (see page 11825)	This is variable ExptIntf.ToolServices.

66.5.1 ExptIntf.LibraryExpertProc Variable

Pascal

```
LibraryExpertProc: TExpertRegisterProc = nil;
```

C++

```
TExpertRegisterProc LibraryExpertProc = nil;
```

File

ExptIntf

Description

This is variable ExptIntf.LibraryExpertProc.

66.5.2 ExptIntf.ToolServices Variable

Pascal

```
ToolServices: TIToolServices = nil;
```

C++

```
TIToolServices ToolServices = nil;
```

File

ExptIntf

Description

This is variable ExptIntf.ToolServices.

66.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
ExpertEntryPoint (see page 11826)	This is constant ExptIntf.ExpertEntryPoint.
isExperts (see page 11826)	This is constant ExptIntf.isExperts.
ValidExpertVersion (see page 11826)	This is constant ExptIntf.ValidExpertVersion.

66.6.1 ExptIntf.ExpertEntryPoint Constant

Pascal

```
ExpertEntryPoint = 'INITEXPERT0017';
```

C++

```
ExpertEntryPoint = 'INITEXPERT0017';
```

File

ExptIntf

Description

This is constant ExptIntf.ExpertEntryPoint.

66.6.2 ExptIntf.isExperts Constant

Pascal

```
isExperts = 'Experts';
```

C++

```
isExperts = 'Experts';
```

File

ExptIntf

Description

This is constant ExptIntf.isExperts.

66.6.3 ExptIntf.ValidExpertVersion Constant

Pascal

```
ValidExpertVersion = 3;
```

C++

```
ValidExpertVersion = 3;
```

File

ExptIntf

Description

This is constant ExptIntf.ValidExpertVersion.

67 ExtActns Namespace

This is namespace ExtActns.

Topics

Name	Description
Classes (see page 11831)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 12096)	The following table lists structs, records, enums in this documentation.
Types (see page 12099)	The following table lists types in this documentation.

Classes

	Name	Description
	TBrowseURL (see page 11832)	TBrowseURL is the standard action for launching the default browser to display a specified URL (see page 11841).
	TCustomFileRun (see page 11844)	TCustomFileRun is the base class for actions that perform a specified operation on a file.
	TDownLoadURL (see page 11857)	TDownLoadURL is the standard action for saving the contents of a specified URL (see page 11869) to a file.
	TFileRun (see page 11872)	TFileRun is the standard action for performing a specified operation on a file.
	TListControlAction (see page 11886)	TListControlAction is the base class for action objects that operate on the selection of a list control (list box or list view).
	TListControlClearSelection (see page 11893)	TListControlClearSelection deselects all selected items in a list control (list view or list box).
	TListControlCopySelection (see page 11906)	TListControlCopySelection copies the selected items in a list control (list view or list box) to another (destination) list control.
	TListControlDeleteSelection (see page 11919)	TListControlDeleteSelection deletes the selected items in a list control (list view or listbox).
	TListControlMoveSelection (see page 11931)	TListControlMoveSelection moves the selected items in a list control (list view or list box) to another (destination) list control.
	TListControlSelectAll (see page 11937)	TListControlSelectAll selects all the items in a list control (list view or listbox).
	TNextTab (see page 11949)	TNextTab moves a tabbed control to its next page.
	TOpenPicture (see page 11965)	TOpenPicture is the standard action for displaying the open picture dialog.
	TPreviousTab (see page 11977)	TPreviousTab moves a page control to its previous page.
	TRichEditAction (see page 11992)	TRichEditAction is the base class for action objects that apply rich text attributes to a rich edit control.
	TRichEditAlignCenter (see page 11998)	TRichEditAlignCenter centers the text horizontally in the current paragraph of a rich edit control.
	TRichEditAlignLeft (see page 12005)	TRichEditAlignLeft left-justifies the text in the current paragraph of a rich edit control.

	TRichEditAlignRight (see page 12011)	TRichEditAlignRight right-justifies the text in the current paragraph of a rich edit control.
	TRichEditBold (see page 12017)	TRichEditBold toggles the bold attribute of the currently selected text in a rich edit control.
	TRichEditBullets (see page 12023)	TRichEditBullets toggles whether the current paragraph of a rich edit control is bulleted.
	TRichEditItalic (see page 12029)	TRichEditItalic toggles the italic attribute of the currently selected text in a rich edit control.
	TRichEditStrikeOut (see page 12036)	TRichEditStrikeOut toggles the strikeout attribute of the currently selected text in a rich edit control.
	TRichEditUnderline (see page 12042)	TRichEditUnderline toggles the underline attribute of the currently selected text in a rich edit control.
	TSavePicture (see page 12048)	TSavePicture is the standard action for displaying the save picture dialog.
	TSendMail (see page 12061)	TSendMail is the standard action for sending an email message.
	TTabAction (see page 12073)	TTabAction is the base class for action objects designed to respond to the buttons in a tabbed dialog.
	TURLAction (see page 12083)	TURLAction is the base class for actions that perform some function based on an URL (see page 12093).

Structs, Records, Enums

	Name	Description
	TShowCmd (see page 12096)	This is record ExtActns.TShowCmd.
	TURLDownloadStatus (see page 12097)	This is record ExtActns.TURLDownloadStatus.

Types

Name	Description
TDownloadProgressEvent (see page 12099)	This is type ExtActns.TDownloadProgressEvent.
TValidateTabEvent (see page 12099)	This is type ExtActns.TValidateTabEvent.

67.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TBrowseURL (see page 11832)	TBrowseURL is the standard action for launching the default browser to display a specified URL (see page 11841).
	TCustomFileRun (see page 11844)	TCustomFileRun is the base class for actions that perform a specified operation on a file.
	TDownLoadURL (see page 11857)	TDownLoadURL is the standard action for saving the contents of a specified URL (see page 11869) to a file.
	TFileRun (see page 11872)	TFileRun is the standard action for performing a specified operation on a file.
	TListControlAction (see page 11886)	TListControlAction is the base class for action objects that operate on the selection of a list control (list box or list view).
	TListControlClearSelection (see page 11893)	TListControlClearSelection deselects all selected items in a list control (list view or list box).
	TListControlCopySelection (see page 11906)	TListControlCopySelection copies the selected items in a list control (list view or list box) to another (destination) list control.
	TListControlDeleteSelection (see page 11919)	TListControlDeleteSelection deletes the selected items in a list control (list view or listbox).
	TListControlMoveSelection (see page 11931)	TListControlMoveSelection moves the selected items in a list control (list view or list box) to another (destination) list control.
	TListControlSelectAll (see page 11937)	TListControlSelectAll selects all the items in a list control (list view or listbox).
	TNextTab (see page 11949)	TNextTab moves a tabbed control to its next page.
	TOpenPicture (see page 11965)	TOpenPicture is the standard action for displaying the open picture dialog.
	TPreviousTab (see page 11977)	TPreviousTab moves a page control to its previous page.
	TRichEditAction (see page 11992)	TRichEditAction is the base class for action objects that apply rich text attributes to a rich edit control.
	TRichEditAlignCenter (see page 11998)	TRichEditAlignCenter centers the text horizontally in the current paragraph of a rich edit control.
	TRichEditAlignLeft (see page 12005)	TRichEditAlignLeft left-justifies the text in the current paragraph of a rich edit control.
	TRichEditAlignRight (see page 12011)	TRichEditAlignRight right-justifies the text in the current paragraph of a rich edit control.
	TRichEditBold (see page 12017)	TRichEditBold toggles the bold attribute of the currently selected text in a rich edit control.
	TRichEditBullets (see page 12023)	TRichEditBullets toggles whether the current paragraph of a rich edit control is bulleted.
	TRichEditItalic (see page 12029)	TRichEditItalic toggles the italic attribute of the currently selected text in a rich edit control.
	TRichEditStrikeOut (see page 12036)	TRichEditStrikeOut toggles the strikeout attribute of the currently selected text in a rich edit control.
	TRichEditUnderline (see page 12042)	TRichEditUnderline toggles the underline attribute of the currently selected text in a rich edit control.
	TSavePicture (see page 12048)	TSavePicture is the standard action for displaying the save picture dialog.
	TSendMail (see page 12061)	TSendMail is the standard action for sending an email message.
	TTabAction (see page 12073)	TTabAction is the base class for action objects designed to respond to the buttons in a tabbed dialog.
	TURLAction (see page 12083)	TURLAction is the base class for actions that perform some function based on an URL (see page 12093).

67.1.1 TBrowseURL Class

TBrowseURL is the standard action for launching the default browser to display a specified URL (see page 11841).

Class Hierarchy

Pascal

```
TBrowseURL = class(TURLAction);
```

C++

```
class TBrowseURL : public TURLAction;
```

File

ExtActns

Description

Add TBrowseURL to an action manager or action list to let your application launch a browser to display a specific Web page in response to user actions. Controls (see page 5711) linked to this action cause the application to launch the default browser, as specified in the system registry, and display the Web page specified by the URL (see page 11841) property. You should assign a value to the URL (see page 11841) property before the user invokes this action.

See Also

TFileRun (see page 11872)

TSendMail (see page 12061)

TDownLoadURL (see page 11857)

67.1.1.1 TBrowseURL Members

The following tables list the members exposed by TBrowseURL.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.

	AssignTo (see page 303)	Copies the properties of this action to a destination object.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 11836)	Launches the default Web browser to display the Web page specified by URL (see page 11841).
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 12088)	Indicates whether the action can execute.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.
V	UpdateTarget (see page 12088)	Enables or disables the action, depending on whether the URL (see page 12093) property is set.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterBrowse (see page 11842)	Occurs after the default browser launches and loads the specified URL (see page 11841).
	BeforeBrowse (see page 11842)	Occurs immediately before the default browser launches and loads the specified URL (see page 11841).
	OnHint (see page 11842)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11843)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 11837)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Enabled (see page 11838)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 11838)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11838)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11839)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11839)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11840)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 11840)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 11840)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	URL (see page 11841)	Specifies the Uniform Resource Locator (URL) on which this action performs its function.
	Visible (see page 11841)	Specifies the Visible property for client controls and menu items.

67.1.1.2 TBrowseURL Methods

The methods of the TBrowseURL class are listed here.

Public Methods

	Name	Description
V	ExecuteTarget (see page 11836)	Launches the default Web browser to display the Web page specified by URL (see page 11841).

67.1.1.2.1 TBrowseURL.ExecuteTarget Method

Launches the default Web browser to display the Web page specified by URL (see page 11841).

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

ExecuteTarget is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and launches the default Web browser, passing in the URL (see page 11841) specified by the URL (see page 11841) property so that the browser displays the associated Web page.

See Also

URL (see page 12093)

67.1.1.3 TBrowseURL Properties

The properties of the TBrowseURL class are listed here.

Published Properties

	Name	Description
	Caption (see page 11837)	Represents the caption of client controls and menu items.
	Enabled (see page 11838)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 11838)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11838)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11839)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11839)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11840)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 11840)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 11840)	Specifies the ShortCut property for client menu items.
	URL (see page 11841)	Specifies the Uniform Resource Locator (URL) on which this action performs its function.
	Visible (see page 11841)	Specifies the Visible property for client controls and menu items.

67.1.1.3.1 TBrowseURL.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

67.1.1.3.2 TBrowseURL.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

67.1.1.3.3 TBrowseURL.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

67.1.1.3.4 TBrowseURL.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the

HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

67.1.1.3.5 TBrowseURL.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

67.1.1.3.6 TBrowseURL.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

67.1.1.3.7 TBrowseURL.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

67.1.1.3.8 TBrowseURL.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

67.1.1.3.9 TBrowseURL.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

67.1.1.3.10 TBrowseURL.URL Property

Specifies the Uniform Resource Locator (URL) on which this action performs its function.

Pascal

```
property URL: String;
```

C++

```
__property AnsiString URL;
```

Description

URL is the Uniform Resource Locator (URL) with which this action works. Each TURLAction descendant performs a different function that uses this URL.

This value is not assigned automatically: it is the responsibility of the application to supply a value for URL. If no value is supplied, TURLAction automatically disables itself.

See Also

UpdateTarget (see page 12088)

67.1.1.3.11 TBrowseURL.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

67.1.1.4 TBrowseURL Events

The events of the TBrowseURL class are listed here.

Published Events

	Name	Description
	AfterBrowse (see page 11842)	Occurs after the default browser launches and loads the specified URL (see page 11841).
	BeforeBrowse (see page 11842)	Occurs immediately before the default browser launches and loads the specified URL (see page 11841).
	OnHint (see page 11842)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11843)	Occurs when the application is idle or when the action list updates.

67.1.1.4.1 TBrowseURL.AfterBrowse Event

Occurs after the default browser launches and loads the specified URL (see page 11841).

Pascal

```
property AfterBrowse: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterBrowse;
```

Description

Write an AfterBrowse event handler to take application-specific actions immediately after TBrowseURL (see page 11832) launches the default browser and loads a specified URL (see page 11841).

See Also

BeforeBrowse (see page 11842)

67.1.1.4.2 TBrowseURL.BeforeBrowse Event

Occurs immediately before the default browser launches and loads the specified URL (see page 11841).

Pascal

```
property BeforeBrowse: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeBrowse;
```

Description

Write a BeforeBrowse event handler to take application-specific actions immediately before TBrowseURL (see page 11832) launches the default browser and loads a specified URL (see page 11841).

See Also

AfterBrowse (see page 11842)

67.1.1.4.3 TBrowseURL.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

67.1.1.4.4 TBrowseURL.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
```

```
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

67.1.2 TCustomFileRun Class

TCustomFileRun is the base class for actions that perform a specified operation on a file.

Class Hierarchy

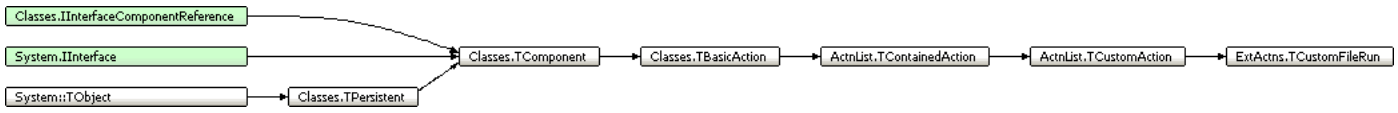

Pascal

```
TCustomFileRun = class(TCustomAction);
```

C++

```
class TCustomFileRun : public TCustomAction;
```

File

ExtActns

Description

Use TCustomFileRun as a base class when defining a pre-defined action that uses ShellExecute to perform an operation on a file. TCustomFileRun implements properties for specifying

A target file or the option of letting the user select one using a browse dialog.

The operation to perform on the target file.

Parameters (see page 11855) to supply to an external application that performs the desired operation.

The startup directory for the external application.

The state of the window that appears when performing the specified operation.

When the ExecuteTarget (see page 11849) method is called, TCustomFileRun uses these properties to call the Windows API ShellExecute, which performs the specified operation if it is listed as applicable to the target file type in the system registry. Depending on the operation, this may involve launching an external application, calling a dynamic-link library (DLL), or displaying a system dialog (such as the file properties dialog).

Do not add TCustomFileRun to an action list or action manager. To add a generic file run action, use TFileRun (see page

11872) instead.

See Also

TFileRun (see page 11872)

67.1.2.1 TCustomFileRun Members

The following tables list the members exposed by TCustomFileRun.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 11850)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11849)	Creates an instance of TCustomFileRun (see page 11844).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.

	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 11849)	Performs the specified operation on a target file.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 11850)	Indicates whether the action can execute.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetupBrowseDialog (see page 11851)	Creates an instance of TOpenDialog and assigns it as the value of the BrowseDlg (see page 11853) property.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.

V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 11851)	Enables or disables the action, depending on current conditions.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Browse (see page 11852)	Controls (see page 5711) whether an open dialog automatically appears to let users browse for the target file.
	BrowseDlg (see page 11853)	Specifies the dialog that allows users to select the target file to run.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Directory (see page 11853)	Specifies the default directory for the externally-launched application.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	FileName (see page 11853)	Specifies the target file to run.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
R	HInst (see page 11854)	Indicates the Windows error code that is returned when TCustomFileRun (see page 11844) launches the external application.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
	Operation (see page 11855)	Indicates what TCustomFileRun (see page 11844) should do with the target file.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parameters (see page 11855)	Specifies parameters to be passed to the application when the target file is an executable.
	ParentControl (see page 11856)	This is ParentControl, a member of class TCustomFileRun.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	ShowCmd (see page 11856)	Specifies how the launched application window appears.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Properties

	Name	Description
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.

	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

67.1.2.2 TCustomFileRun.Create Constructor

Creates an instance of TCustomFileRun (see page 11844).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomFileRun(TComponent * AOwner);
```

Description

Most applications do not need to call the TCustomFileRun (see page 11844) constructor because actions are added at design time to an action list or action manager.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TCustomFileRun (see page 11844) instance.

Create initializes the action's properties and creates a default TOpenDialog instance to act as the value of the BrowseDlg (see page 11853) property.

See Also

Owner (see page 2807)

SetupBrowseDialog (see page 11851)

BrowseDlg (see page 11853)

67.1.2.3 TCustomFileRun Methods

The methods of the TCustomFileRun class are listed here.

Protected Methods

	Name	Description
	Notification (see page 11850)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	ExecuteTarget (see page 11849)	Performs the specified operation on a target file.
	HandlesTarget (see page 11850)	Indicates whether the action can execute.
	SetupBrowseDialog (see page 11851)	Creates an instance of TOpenDialog and assigns it as the value of the BrowseDlg (see page 11853) property.
	UpdateTarget (see page 11851)	Enables or disables the action, depending on current conditions.

67.1.2.3.1 TCustomFileRun.ExecuteTarget Method

Performs the specified operation on a target file.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and performs the operation specified by the Operation (see page 11855) property on the target file specified by the FileName (see page 11853) property or chosen by the user if the value of the Browse (see page 11852) property is true. The values of Parameters (see page 11855), ShowCmd (see page 11856), and Directory (see page 11853), as well as the window handle for ParentControl (see page 11856) are passed to Windows with the request to perform the specified operation.

Note: If the attempt to perform the operation fails, the HInst (see page 11854) property contains an error code that indicates what was wrong. If the attempt succeeds, the HInst (see page 11854) property is greater than 32.

See Also

FileName (see page 11853)

Browse (see page 11852)

Operation (see page 11855)

Parameters (see page 11855)

ShowCmd (see page 11856)

Directory (see page 11853)

parentcontrol (see page 11856)

HInst (see page 11854)

67.1.2.3.2 TCustomFileRun.HandlesTarget Method

Indicates whether the action can execute.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method lets the action object indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

HandlesTarget always returns true, because TCustomFileRun (see page 11844) does not use the target object.

67.1.2.3.3 TCustomFileRun.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TBasicAction overrides the Notification method to check whether the component that is the value of the ActionComponent property is about to be freed. If so, it sets the ActionComponent property to nil (Delphi) or NULL (C++).

See Also

ActionComponent (see page 2715)

67.1.2.3.4 TCustomFileRun.SetupBrowseDialog Method

Creates an instance of TOpenDialog and assigns it as the value of the BrowseDlg (see page 11853) property.

Pascal

```
procedure SetupBrowseDialog;
```

C++

```
__fastcall SetupBrowseDialog();
```

Description

SetupBrowseDialog generates the default open dialog instance for the dialog that appears when the Browse (see page 11852) property is true. It assigns this newly created dialog component as the value of the BrowseDlg (see page 11853) property.

Most applications do not need to call SetupBrowseDialog because it is called automatically by the TCustomFileRun (see page 11844) constructor.

See Also

BrowseDlg (see page 11853)

TOpenDialog (see page 11407)

Browse (see page 11852)

67.1.2.3.5 TCustomFileRun.UpdateTarget Method

Enables or disables the action, depending on current conditions.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when application is idle so that the action can update itself to reflect current conditions.

In TCustomFileRun (see page 11844), UpdateTarget checks whether it can obtain a target file. If the FileName (see page

11853) property specifies a target file, or if the Browse (see page 11852) property is true, indicating that users can select a target file, then UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

FileName (see page 11853)

Browse (see page 11852)

67.1.2.4 TCustomFileRun Properties

The properties of the TCustomFileRun class are listed here.

Public Properties

	Name	Description
	Browse (see page 11852)	Controls (see page 5711) whether an open dialog automatically appears to let users browse for the target file.
	BrowseDlg (see page 11853)	Specifies the dialog that allows users to select the target file to run.
	Directory (see page 11853)	Specifies the default directory for the externally-launched application.
	FileName (see page 11853)	Specifies the target file to run.
R	HInst (see page 11854)	Indicates the Windows error code that is returned when TCustomFileRun (see page 11844) launches the external application.
	Operation (see page 11855)	Indicates what TCustomFileRun (see page 11844) should do with the target file.
	Parameters (see page 11855)	Specifies parameters to be passed to the application when the target file is an executable.
	ParentControl (see page 11856)	This is ParentControl, a member of class TCustomFileRun.
	ShowCmd (see page 11856)	Specifies how the launched application window appears.

67.1.2.4.1 TCustomFileRun.Browse Property

Controls (see page 5711) whether an open dialog automatically appears to let users browse for the target file.

Pascal

```
property Browse: Boolean;
```

C++

```
__property Boolean Browse;
```

Description

Browse controls whether TCustomFileRun (see page 11844) displays the dialog specified by BrowseDlg (see page 11853) when it fires, letting users select the target file to be launched. When Browse is true, firing TCustomFileRun (see page 11844) displays the open dialog specified by BrowseDlg (see page 11853), and then (if the user does not cancel) performs the operation specified by Operation (see page 11855) on the chosen file. When Browse is false, your application must specify the target file by explicitly setting the FileName (see page 11853) property.

Note: When Browse is true, TCustomFileRun (see page 11844) ignores the FileName (see page 11853) property.

See Also

BrowseDlg (see page 11853)

FileName (see page 11853)

Operation (see page 11855)

67.1.2.4.2 TCustomFileRun.BrowseDlg Property

Specifies the dialog that allows users to select the target file to run.

Pascal

```
property BrowseDlg: TOpenDialog;
```

C++

```
__property TOpenDialog BrowseDlg;
```

Description

Use BrowseDlg when the Browse (see page 11852) property is true to configure the dialog that users use to select a target file. By expanding BrowseDlg in the Object Inspector at design time, or by setting its properties at runtime, you can customize the way the dialog first appears when the action fires.

By default, BrowseDlg is an internally-generated TOpenDialog object. When using the default dialog, there is no need to explicitly add a dialog component to your application. If you want to use a specialized TOpenDialog descendant (such as TOpenPictureDialog), however, set BrowseDlg to an open dialog component you place in the same form or data module as the TCustomFileRun (see page 11844) action.

See Also

Browse (see page 11852)

FileName (see page 11853)

TOpenDialog (see page 11407)

67.1.2.4.3 TCustomFileRun.Directory Property

Specifies the default directory for the externally-launched application.

Pascal

```
property Directory: TFileName;
```

C++

```
__property TFileName Directory;
```

Description

Set Directory to specify a default (startup) directory for the external application that is launched when the action fires.

See Also

Parameters (see page 11855)

FileName (see page 11853)

67.1.2.4.4 TCustomFileRun.FileName Property

Specifies the target file to run.

Pascal

```
property FileName: TFileName;
```

C++

```
__property TFileName FileName;
```

Description

Set FileName to specify the file on which TCustomFileRun (see page 11844) performs the operation specified by Operation (see page 11855) when the action fires. FileName can specify a file that is opened using an external application (such as a .txt file that causes a text editor to launch), or it can specify an executable file for the application itself. When the action fires in response to a user action, TCustomFileRun (see page 11844) launches an external application or dll to perform Operation (see page 11855) on the file specified by FileName.

Note:

See Also

Browse (see page 11852)

BrowseDlg (see page 11853)

Parameters (see page 11855)

Operation (see page 11855)

67.1.2.4.5 TCustomFileRun.HInst Property

Indicates the Windows error code that is returned when TCustomFileRun (see page 11844) launches the external application.

Pascal

```
property HInst: HInst;
```

C++

```
__property HInst HInst;
```

Description

TCustomFileRun (see page 11844) sets HInst to the Windows return value when it launches the external application. If this value is less than 32, it represents an error code that indicates why the application failed to launch. This value is only available after the action has executed.

The following table lists the possible error values (defined in shellapi.pas) that can be the value of HInst when it is less than 32.

Value	Meaning
0	The operating system is out of memory or resources.
ERROR_FILE_NOT_FOUND	The specified file was not found.
ERROR_PATH_NOT_FOUND	The specified path was not found.
ERROR_BAD_FORMAT	The .exe file is invalid (does not have the correct internal format).
SE_ERR_ACCESSDENIED	The operating system denied access to the specified file.
SE_ERR_ASSOCINCOMPLETE	The file name association is incomplete or invalid.
SE_ERR_DDEBUSY	A DDE transaction could not be completed because other DDE transactions were being processed.
SE_ERR_DDEFAIL	A DDE transaction failed.
SE_ERR_DDETIMEOUT	A DDE transaction could not be completed because the request timed out.
SE_ERR_DLLNOTFOUND	A specified dynamic-link library was not found.
SE_ERR_FNF	The specified file was not found.
SE_ERR_NOASSOC	There is no application associated with the given file name extension or TCustomFileRun (see page 11844) attempted to print an unprintable file.

SE_ERR_OOM	There was not enough memory to complete the operation.
SE_ERR_PNF	The specified path was not found.
SE_ERR_SHARE	A sharing violation occurred.

67.1.2.4.6 TCustomFileRun.Operation Property

Indicates what TCustomFileRun (see page 11844) should do with the target file.

Pascal

```
property Operation: String;
```

C++

```
__property AnsiString Operation;
```

Description

Set Operation to indicate what should be done with the target file when TCustomFileRun (see page 11844) fires. The possible values for Operation depend on the particular file or folder specified by FileName (see page 11853) or selected by the user when Browse (see page 11852) is true. The possible operations for a particular file or folder are listed in the system Registry (see page 18633).

The following strings are usually valid:

Value	Meaning
edit	TCustomFileRun (see page 11844) opens an editor. If FileName (see page 11853) is not an editable file, the action fails.
explore	TCustomFileRun (see page 11844) launches the Windows explorer for the folder specified by FileName (see page 11853).
open	TCustomFileRun (see page 11844) opens the file specified by FileName (see page 11853). FileName (see page 11853) can be an executable file, a file associated with an application that can open it, or a folder.
print	TCustomFileRun (see page 11844) prints the file specified by FileName (see page 11853). If FileName (see page 11853) is a binary file (for example, if it is an executable), the action fails.
properties	TCustomFileRun (see page 11844) displays the file or folder's properties in a Properties dialog.

Note: If Operation is an empty string, Windows tries to use the "open" operation. If the specified file does not support the "open" operation, Windows tries to use the default operation. If there is no "open" operation and no default operation, and if the application is running on Windows 2000 or later, Windows uses the first verb listed in the Registry (see page 18633).

See Also

FileName (see page 11853)

Browse (see page 11852)

67.1.2.4.7 TCustomFileRun.Parameters Property

Specifies parameters to be passed to the application when the target file is an executable.

Pascal

```
property Parameters: String;
```

C++

```
__property AnsiString Parameters;
```

Description

Use Parameters to specify any parameters that should be passed to the external application when it is launched. Only set Parameters if FileName (see page 11853) is the name of an Executable file or the user is browsing for an executable file. If the target file holds data associated with the application to launch, Parameters should be an empty string.

See Also

FileName (see page 11853)

Operation (see page 11855)

67.1.2.4.8 TCustomFileRun.ParentControl Property

Pascal

```
property ParentControl: TWinControl;
```

C++

```
__property TWinControl * ParentControl;
```

Description

This is ParentControl, a member of class TCustomFileRun.

67.1.2.4.9 TCustomFileRun.ShowCmd Property

Specifies how the launched application window appears.

Pascal

```
property ShowCmd: TShowCmd;
```

C++

```
__property TShowCmd ShowCmd;
```

Description

Use ShowCmd to indicate how you want the application window to appear when TCustomFileRun (see page 11844) performs the specified operation on its target file. ShowCmd can have any of the following values:

Value	Meaning
scHide	The external application window is hidden. If it was active, another window is activated.
scMaximize	The external application window is maximized (but not necessarily activated)
scMinimize	The external application window is minimized (but not necessarily activated).
scRestore	The external application window is activated. If it was minimized or maximized, Windows restores it to its original size and position.
scShow	The external application window is activated in its current size and position.
scShowDefault	The external application window is shown in the default startup state specified by that external application.
scShowMaximized	The external application window is activated and maximized.
scShowMinimized	The external application window is activated and minimized.
scShowMinNoActive	The external application window is minimized. The active window does not change.
scShowNA	The external application window is displayed (after being hidden) but activation does not change.

scShowNoActivate	The external application window is displayed in its most recent size and position, but activation does not change.
scShowNormal	The external application window is activated and restored to its original size and position if it was maximized or minimized.

See Also

Operation (see page 11855)

67.1.3 TDownLoadURL Class

TDownLoadURL is the standard action for saving the contents of a specified URL (see page 11869) to a file.

Class Hierarchy

Pascal

```
TDownLoadURL = class(TURLAction, IBindStatusCallback);
```

C++

```
class TDownLoadURL : public TURLAction, public IBindStatusCallback;
```

File

ExtActns

Description

Add TDownLoadURL to an action manager or action list to let your application save the contents of a specified URL (see page 11869) to a file. Controls (see page 5711) linked to this action cause the application to lookup the resource specified by the URL (see page 11869) property and save it to the file specified by the Filename (see page 11866) property. While TDownLoadURL is writing to the specified file, it periodically generates an OnDownloadProgress (see page 11870) event, so that you can provide users with feedback about the process.

You should assign a value to the URL (see page 11869) property and to the FileName property before the user invokes this action.

See Also

TFileRun (see page 11872)

TSendMail (see page 12061)

TBrowseURL (see page 11832)

67.1.3.1 TDownLoadURL Members

The following tables list the members exposed by TDownLoadURL.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetBindInfo (see page 11863)	This is GetBindInfo, a member of class TDownLoadURL.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPriority (see page 11863)	This is GetPriority, a member of class TDownLoadURL.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	OnDataAvailable (see page 11863)	This is OnDataAvailable, a member of class TDownLoadURL.
	OnLowResource (see page 11863)	This is OnLowResource, a member of class TDownLoadURL.
	OnObjectAvailable (see page 11864)	This is OnObjectAvailable, a member of class TDownLoadURL.
	OnProgress (see page 11864)	This is OnProgress, a member of class TDownLoadURL.
	OnStartBinding (see page 11864)	This is OnStartBinding, a member of class TDownLoadURL.
	OnStopBinding (see page 11864)	This is OnStopBinding, a member of class TDownLoadURL.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.

	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 11862)	Creates the file specified by the Filename (see page 11866) property and downloads the contents of URL (see page 11869) to it.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 12088)	Indicates whether the action can execute.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.
V	UpdateTarget (see page 12088)	Enables or disables the action, depending on whether the URL (see page 12093) property is set.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterDownload (see page 11870)	Occurs after TDownLoadURL (see page 11857) downloads the specified resource and writes it to a file.
	BeforeDownload (see page 11870)	Occurs before TDownLoadURL (see page 11857) downloads the specified resource and writes it to a file.
	OnDownloadProgress (see page 11870)	Occurs periodically while TDownLoadURL (see page 11857) is downloading the specified resource and writing it to a file.
	OnHint (see page 11871)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11871)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 11865)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Enabled (see page 11865)	Indicates whether client controls and menu items are enabled.
	Filename (see page 11866)	Specifies the file to which this action copies the resource identified by the URL (see page 11869) property.
	HelpContext (see page 11866)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11866)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11867)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11867)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11868)	Indicates the ImageIndex property for client controls and menu items.

	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 11868)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 11869)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	URL (see page 11869)	Specifies the Uniform Resource Locator (URL) on which this action performs its function.
	Visible (see page 11869)	Specifies the Visible property for client controls and menu items.

67.1.3.2 TDownLoadURL Methods

The methods of the TDownLoadURL class are listed here.

Protected Methods

	Name	Description
	GetBindInfo (see page 11863)	This is GetBindInfo, a member of class TDownLoadURL.
	GetPriority (see page 11863)	This is GetPriority, a member of class TDownLoadURL.
	OnDataAvailable (see page 11863)	This is OnDataAvailable, a member of class TDownLoadURL.
	OnLowResource (see page 11863)	This is OnLowResource, a member of class TDownLoadURL.
	OnObjectAvailable (see page 11864)	This is OnObjectAvailable, a member of class TDownLoadURL.
	OnProgress (see page 11864)	This is OnProgress, a member of class TDownLoadURL.
	OnStartBinding (see page 11864)	This is OnStartBinding, a member of class TDownLoadURL.
	OnStopBinding (see page 11864)	This is OnStopBinding, a member of class TDownLoadURL.

Public Methods

	Name	Description
	ExecuteTarget (see page 11862)	Creates the file specified by the Filename (see page 11866) property and downloads the contents of URL (see page 11869) to it.

67.1.3.2.1 TDownLoadURL.ExecuteTarget Method

Creates the file specified by the Filename (see page 11866) property and downloads the contents of URL (see page 11869) to it.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

ExecuteTarget is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and looks up (on the internet) the URL (see page 11869) specified by the URL (see page 11869) property, creates the file specified by the Filename (see page 11866) property, and writes the bits found at the specified URL (see page 11869) to the file.

Periodically throughout this download process, TDownLoadURL (see page 11857) generates an OnDownloadProgress (see page 11870) event.

See Also

URL (see page 12093)

Filename (see page 11866)

OnDownloadProgress (see page 11870)

67.1.3.2.2 TDownLoadURL.GetBindInfo Method

Pascal

```
function GetBindInfo(out grfBINDF: Cardinal; var bindinfo: _tagBINDINFO): HRESULT; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetBindInfo(unsigned grfBINDF, _tagBINDINFO bindinfo);
```

Description

This is GetBindInfo, a member of class TDownLoadURL.

67.1.3.2.3 TDownLoadURL.GetPriority Method

Pascal

```
function GetPriority(out nPriority): HRESULT; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetPriority( nPriority);
```

Description

This is GetPriority, a member of class TDownLoadURL.

67.1.3.2.4 TDownLoadURL.OnDataAvailable Method

Pascal

```
function OnDataAvailable(grfBSCF: Cardinal; dwSize: Cardinal; formatetc: PFormatEtc; stgmed:
PStgMedium): HRESULT; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnDataAvailable(unsigned grfBSCF, unsigned dwSize, PFormatEtc
formatetc, PStgMedium stgmed);
```

Description

This is OnDataAvailable, a member of class TDownLoadURL.

67.1.3.2.5 TDownLoadURL.OnLowResource Method

Pascal

```
function OnLowResource(reserved: Cardinal): HRESULT; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnLowResource(unsigned reserved);
```

Description

This is OnLowResource, a member of class TDownLoadURL.

67.1.3.2.6 TDownLoadURL.OnObjectAvailable Method

Pascal

```
function OnObjectAvailable(const iid: TGUID; punk: IUnknown): HRESULT; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnObjectAvailable(const TGUID iid, IUnknown punk);
```

Description

This is OnObjectAvailable, a member of class TDownLoadURL.

67.1.3.2.7 TDownLoadURL.OnProgress Method

Pascal

```
function OnProgress(ulProgress: Cardinal; ulProgressMax: Cardinal; ulStatusCode: Cardinal;
szStatusText: PWideChar): HRESULT; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnProgress(unsigned ulProgress, unsigned ulProgressMax, unsigned
ulStatusCode, PWideChar szStatusText);
```

Description

This is OnProgress, a member of class TDownLoadURL.

67.1.3.2.8 TDownLoadURL.OnStartBinding Method

Pascal

```
function OnStartBinding(dwReserved: Cardinal; pib: IBinding): HRESULT; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnStartBinding(unsigned dwReserved, IBinding pib);
```

Description

This is OnStartBinding, a member of class TDownLoadURL.

67.1.3.2.9 TDownLoadURL.OnStopBinding Method

Pascal

```
function OnStopBinding(hresult: HRESULT; szError: PWideChar): HRESULT; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnStopBinding(HRESULT hresult, PWideChar szError);
```

Description

This is OnStopBinding, a member of class TDownLoadURL.

67.1.3.3 TDownLoadURL Properties

The properties of the TDownLoadURL class are listed here.

Published Properties

	Name	Description
	Caption (see page 11865)	Represents the caption of client controls and menu items.
	Enabled (see page 11865)	Indicates whether client controls and menu items are enabled.
	Filename (see page 11866)	Specifies the file to which this action copies the resource identified by the URL (see page 11869) property.
	HelpContext (see page 11866)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11866)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11867)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11867)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11868)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 11868)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 11869)	Specifies the ShortCut property for client menu items.
	URL (see page 11869)	Specifies the Uniform Resource Locator (URL) on which this action performs its function.
	Visible (see page 11869)	Specifies the Visible property for client controls and menu items.

67.1.3.3.1 TDownLoadURL.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

67.1.3.3.2 TDownLoadURL.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

67.1.3.3.3 TDownLoadURL.Filename Property

Specifies the file to which this action copies the resource identified by the URL (see page 11869) property.

Pascal

```
property Filename: TFilename;
```

C++

```
__property TFilename Filename;
```

Description

Filename is the name of the file that TDownLoadURL (see page 11857) creates to hold the contents of the URL (see page 11869) specified by the URL (see page 11869) property. When the user triggers this action, TDownLoadURL (see page 11857) looks up URL (see page 11869) on the internet, creates the file specified by Filename, and writes the bits it finds at URL (see page 11869) to this file.

This value is not assigned automatically: it is the responsibility of the application to supply a value for Filename.

See Also

ExecuteTarget (see page 11862)

URL (see page 12093)

67.1.3.3.4 TDownLoadURL.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

67.1.3.3.5 TDownLoadURL.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

67.1.3.3.6 TDownLoadURL.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

67.1.3.3.7 TDownLoadURL.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

67.1.3.3.8 TDownLoadURL.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

67.1.3.3.9 TDownLoadURL.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

67.1.3.3.10 TDownLoadURL.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

67.1.3.3.11 TDownLoadURL.URL Property

Specifies the Uniform Resource Locator (URL) on which this action performs its function.

Pascal

```
property URL: String;
```

C++

```
__property AnsiString URL;
```

Description

URL is the Uniform Resource Locator (URL) with which this action works. Each TURLAction descendant performs a different function that uses this URL.

This value is not assigned automatically: it is the responsibility of the application to supply a value for URL. If no value is supplied, TURLAction automatically disables itself.

See Also

UpdateTarget (see page 12088)

67.1.3.3.12 TDownLoadURL.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

67.1.3.4 TDownLoadURL Events

The events of the TDownLoadURL class are listed here.

Published Events

	Name	Description
	AfterDownload (see page 11870)	Occurs after TDownLoadURL (see page 11857) downloads the specified resource and writes it to a file.
	BeforeDownload (see page 11870)	Occurs before TDownLoadURL (see page 11857) downloads the specified resource and writes it to a file.
	OnDownloadProgress (see page 11870)	Occurs periodically while TDownLoadURL (see page 11857) is downloading the specified resource and writing it to a file.
	OnHint (see page 11871)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11871)	Occurs when the application is idle or when the action list updates.

67.1.3.4.1 TDownLoadURL.AfterDownload Event

Occurs after TDownLoadURL (see page 11857) downloads the specified resource and writes it to a file.

Pascal

```
property AfterDownload: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterDownload;
```

Description

Write an AfterDownload event handler to perform a specific action after downloading the specified resource.

67.1.3.4.2 TDownLoadURL.BeforeDownload Event

Occurs before TDownLoadURL (see page 11857) downloads the specified resource and writes it to a file.

Pascal

```
property BeforeDownload: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeDownload;
```

Description

Write a BeforeDownload event handler to perform a specific action before downloading the specified resource.

67.1.3.4.3 TDownLoadURL.OnDownloadProgress Event

Occurs periodically while TDownLoadURL (see page 11857) is downloading the specified resource and writing it to a file.

Pascal

```
property OnDownloadProgress: TDownloadProgressEvent;
```

C++

```
__property TDownloadProgressEvent OnDownloadProgress;
```

Description

OnDownloadProgress occurs periodically while the contents of the specified URL (see page 11869) are downloaded to a file.

Sender is the action that is copying the contents of an URL (see page 11869) to a file.

Progress indicates the current progress of the download operation relative to the expected maximum indicated in the ProgressMax parameter.

ProgressMax indicates the expected maximum value of the Progress parameter. This value can change during the download operation as more information becomes available. A value of zero means that the maximum value is unknown.

StatusCode indicates the current stage of the download process or other information about the download process.

StatusText is a string that corresponds to StatusCode. It can be used for displaying the status of the download to users.

Cancel allows the event handler to cancel the download operation. On entry to the event handler, Cancel is false. If the event handler changes Cancel to true, the download operation is aborted.

See Also

Filename (see page 11866)

URL (see page 12093)

67.1.3.4.4 TDownLoadURL.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

67.1.3.4.5 TDownLoadURL.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
```

```
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

67.1.4 TFileRun Class

TFileRun is the standard action for performing a specified operation on a file.

Class Hierarchy

Pascal

```
TFileRun = class(TCustomFileRun);
```

C++

```
class TFileRun : public TCustomFileRun;
```

File

ExtActns

Description

Add TFileRun to an action list to let your application launch an external application or dll in response to user actions. Controls (see page 5711) linked to this action cause your application to perform the operation specified by the Operation (see page 11882) property on the file specified by the FileName (see page 11880) property.

Before this action fires, configure its properties to indicate what application to launch and how to launch it:

To indicate the target file to run, set the FileName (see page 11880) property to the name of a file on which the external application operates or to the name of the application's executable file. If you want to let the user select the file name instead when the action fires, set the Browse (see page 11878) property to true, and an open dialog appears when the action fires.

To indicate what action to perform on the specified file, set the Operation (see page 11882) property.

To supply parameters to the external application when it is launched, use the Parameters (see page 11882) property.

To specify a default directory for the external application, use the Directory (see page 11879) property.

To control how the external application's window appears (minimized, maximized, hidden, and so on), use the ShowCmd (see page 11884) property.

If the external application needs a parent window (to parent a dialog such as a properties dialog), or if you want to allow TFileRun to display error messages, use the ParentControl (see page 11883) property.

See Also

TBrowseURL (see page 11832)

TSendMail (see page 12061)

TFileOpenWith (see page 20619)

TDownloadURL

67.1.4.1 TFileRun Members

The following tables list the members exposed by TFileRun.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 11850)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 11849)	Creates an instance of TCustomFileRun (see page 11844).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".

	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
V	ExecuteTarget (see page 11849)	Performs the specified operation on a target file.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 11850)	Indicates whether the action can execute.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetupBrowseDialog (see page 11851)	Creates an instance of TOpenDialog and assigns it as the value of the BrowseDlg (see page 11853) property.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.
V	UpdateTarget (see page 11851)	Enables or disables the action, depending on current conditions.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
R	HInst (see page 11854)	Indicates the Windows error code that is returned when TCustomFileRun (see page 11844) launches the external application.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnHint (see page 11885)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11885)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Browse (see page 11878)	Controls whether an open dialog automatically appears to let users browse for the target file.
	BrowseDlg (see page 11879)	Specifies the dialog that allows users to select the target file to run.
	Caption (see page 11879)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Directory (see page 11879)	Specifies the default directory for the externally-launched application.
	Enabled (see page 11880)	Indicates whether client controls and menu items are enabled.
	FileName (see page 11880)	Specifies the target file to run.
	HelpContext (see page 11881)	Indicates the help context ID for client controls and menu items.
	Hint (see page 11881)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11881)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Operation (see page 11882)	Indicates what TCustomFileRun should do with the target file.
	Parameters (see page 11882)	Specifies parameters to be passed to the application when the target file is an executable.
	ParentControl (see page 11883)	This is ParentControl, a member of class TFileRun.
	SecondaryShortCuts (see page 11883)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 11883)	Specifies the ShortCut property for client menu items.
	ShowCmd (see page 11884)	Specifies how the launched application window appears.
	Tag (see page 2808)	Stores an integer value as part of a component.

	Visible (see page 11884)	Specifies the Visible property for client controls and menu items.

67.1.4.2 TFileRun Properties

The properties of the TFileRun class are listed here.

Published Properties

	Name	Description
	Browse (see page 11878)	Controls whether an open dialog automatically appears to let users browse for the target file.
	BrowseDlg (see page 11879)	Specifies the dialog that allows users to select the target file to run.
	Caption (see page 11879)	Represents the caption of client controls and menu items.
	Directory (see page 11879)	Specifies the default directory for the externally-launched application.
	Enabled (see page 11880)	Indicates whether client controls and menu items are enabled.
	FileName (see page 11880)	Specifies the target file to run.
	HelpContext (see page 11881)	Indicates the help context ID for client controls and menu items.
	Hint (see page 11881)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11881)	Indicates the ImageIndex property for client controls and menu items.
	Operation (see page 11882)	Indicates what TCustomFileRun should do with the target file.
	Parameters (see page 11882)	Specifies parameters to be passed to the application when the target file is an executable.
	ParentControl (see page 11883)	This is ParentControl, a member of class TFileRun.
	SecondaryShortCuts (see page 11883)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 11883)	Specifies the ShortCut property for client menu items.
	ShowCmd (see page 11884)	Specifies how the launched application window appears.
	Visible (see page 11884)	Specifies the Visible property for client controls and menu items.

67.1.4.2.1 TFileRun.Browse Property

Controls whether an open dialog automatically appears to let users browse for the target file.

Pascal

```
property Browse: Boolean;
```

C++

```
__property Boolean Browse;
```

Description

Browse controls whether TCustomFileRun displays the dialog specified by BrowseDlg when it fires, letting users select the target file to be launched. When Browse is true, firing TCustomFileRun displays the open dialog specified by BrowseDlg, and then (if the user does not cancel) performs the operation specified by Operation on the chosen file. When Browse is false, your application must specify the target file by explicitly setting the FileName property.

Note: When Browse is true, TCustomFileRun ignores the FileName property.

See Also

BrowseDlg (see page 11853)

FileName (see page 11853)

Operation (see page 11855)

67.1.4.2.2 TFileRun.BrowseDlg Property

Specifies the dialog that allows users to select the target file to run.

Pascal

```
property BrowseDlg: TOpenDialog;
```

C++

```
__property TOpenDialog BrowseDlg;
```

Description

Use BrowseDlg when the Browse property is true to configure the dialog that users use to select a target file. By expanding BrowseDlg in the Object Inspector at design time, or by setting its properties at runtime, you can customize the way the dialog first appears when the action fires.

By default, BrowseDlg is an internally-generated TOpenDialog object. When using the default dialog, there is no need to explicitly add a dialog component to your application. If you want to use a specialized TOpenDialog descendant (such as TOpenPictureDialog), however, set BrowseDlg to an open dialog component you place in the same form or data module as the TCustomFileRun action.

See Also

Browse (see page 11852)

FileName (see page 11853)

TOpenDialog (see page 11407)

67.1.4.2.3 TFileRun.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

67.1.4.2.4 TFileRun.Directory Property

Specifies the default directory for the externally-launched application.

Pascal

```
property Directory: TFileName;
```

C++

```
__property TFileName Directory;
```

Description

Set Directory to specify a default (startup) directory for the external application that is launched when the action fires.

See Also

Parameters (see page 11855)

FileName (see page 11853)

67.1.4.2.5 TFileRun.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

67.1.4.2.6 TFileRun.FileName Property

Specifies the target file to run.

Pascal

```
property FileName: TFileName;
```

C++

```
__property TFileName FileName;
```

Description

Set FileName to specify the file on which TCustomFileRun performs the operation specified by Operation when the action fires. FileName can specify a file that is opened using an external application (such as a .txt file that causes a text editor to launch), or it can specify an executable file for the application itself. When the action fires in response to a user action, TCustomFileRun launches an external application or dll to perform Operation on the file specified by FileName.

Note:

See Also

Browse (see page 11852)

BrowseDlg (see page 11853)

Parameters (see page 11855)

Operation (see page 11855)

67.1.4.2.7 TFileRun.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

67.1.4.2.8 TFileRun.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

67.1.4.2.9 TFileRun.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an

image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

67.1.4.2.10 TFileRun.Operation Property

Indicates what TCustomFileRun should do with the target file.

Pascal

```
property Operation: String;
```

C++

```
__property AnsiString Operation;
```

Description

Set Operation to indicate what should be done with the target file when TCustomFileRun fires. The possible values for Operation depend on the particular file or folder specified by FileName or selected by the user when Browse is true. The possible operations for a particular file or folder are listed in the system Registry.

The following strings are usually valid:

Value	Meaning
edit	TCustomFileRun opens an editor. If FileName is not an editable file, the action fails.
explore	TCustomFileRun launches the Windows explorer for the folder specified by FileName.
open	TCustomFileRun opens the file specified by FileName. FileName can be an executable file, a file associated with an application that can open it, or a folder.
print	TCustomFileRun prints the file specified by FileName. If FileName is a binary file (for example, if it is an executable), the action fails.
properties	TCustomFileRun displays the file or folder's properties in a Properties dialog.

Note: If Operation is an empty string, Windows tries to use the "open" operation. If the specified file does not support the "open" operation, Windows tries to use the default operation. If there is no "open" operation and no default operation, and if the application is running on Windows 2000 or later, Windows uses the first verb listed in the Registry.

See Also

FileName (see page 11853)

Browse (see page 11852)

67.1.4.2.11 TFileRun.Parameters Property

Specifies parameters to be passed to the application when the target file is an executable.

Pascal

```
property Parameters: String;
```

C++

```
__property AnsiString Parameters;
```

Description

Use Parameters to specify any parameters that should be passed to the external application when it is launched. Only set Parameters if FileName is the name of an Executable file or the user is browsing for an executable file. If the target file holds data associated with the application to launch, Parameters should be an empty string.

See Also

FileName (see page 11853)

Operation (see page 11855)

67.1.4.2.12 TFileRun.ParentControl Property

Pascal

```
property ParentControl: TWinControl;
```

C++

```
__property TWinControl * ParentControl;
```

Description

This is ParentControl, a member of class TFileRun.

67.1.4.2.13 TFileRun.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

67.1.4.2.14 TFileRun.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

67.1.4.2.15 TFileRun.ShowCmd Property

Specifies how the launched application window appears.

Pascal

```
property ShowCmd: TShowCmd;
```

C++

```
__property TShowCmd ShowCmd;
```

Description

Use ShowCmd to indicate how you want the application window to appear when TCustomFileRun performs the specified operation on its target file. ShowCmd can have any of the following values:

Value	Meaning
scHide	The external application window is hidden. If it was active, another window is activated.
scMaximize	The external application window is maximized (but not necessarily activated)
scMinimize	The external application window is minimized (but not necessarily activated).
scRestore	The external application window is activated. If it was minimized or maximized, Windows restores it to its original size and position.
scShow	The external application window is activated in its current size and position.
scShowDefault	The external application window is shown in the default startup state specified by that external application.
scShowMaximized	The external application window is activated and maximized.
scShowMinimized	The external application window is activated and minimized.
scShowMinNoActive	The external application window is minimized. The active window does not change.
scShowNA	The external application window is displayed (after being hidden) but activation does not change.
scShowNoActivate	The external application window is displayed in its most recent size and position, but activation does not change.
scShowNormal	The external application window is activated and restored to its original size and position if it was maximized or minimized.

See Also

Operation (see page 11855)

67.1.4.2.16 TFileRun.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

67.1.4.3 TFileRun Events

The events of the TFileRun class are listed here.

Published Events

	Name	Description
	OnHint (see page 11885)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11885)	Occurs when the application is idle or when the action list updates.

67.1.4.3.1 TFileRun.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

67.1.4.3.2 TFileRun.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

67.1.5 TListControlAction Class

TListControlAction is the base class for action objects that operate on the selection of a list control (list box or list view).

Class Hierarchy

Pascal

```
TListControlAction = class(TCustomAction);
```

C++

```
class TListControlAction : public TCustomAction;
```

File

ExtActns

Description

TListControlAction implements the common behavior for actions that work with list controls (list boxes and list views). Each descendant class performs a specific operation on the target list control. Do not use TListControlAction directly in an application. Instead, use a TListControlAction descendant such as TListControlSelectAll (see page 11937), TListControlClearSelection (see page 11893), TListControlDeleteSelection (see page 11919), TListControlCopySelection (see page 11906), or TListControlMoveSelection (see page 11931).

TListControlAction makes the target list control available to descendant classes as the value of the ListControl (see page 11893) property. TListControlAction descendants then override the ExecuteTarget method to perform their specific function on the target list control.

See Also

TListControlClearSelection (see page 11893)

TListControlCopySelection (see page 11906)

TListControlMoveSelection (see page 11931)

TListCOntrolDeleteSelection

TListControlSelectAll (see page 11937)

TCustomListControl (see page 5928)

67.1.5.1 TListControlAction Members

The following tables list the members exposed by TListControlAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsEnabled (see page 11892)	This is IsEnabled, a member of class TListControlAction.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 11892)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.

	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 11891)	Creates an instance of TListControlAction (see page 11886).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 11892)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.

	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
	ListControl (see page 11893)	Provides access to the target of the action.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Properties

	Name	Description
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

67.1.5.2 TListControlAction.Create Constructor

Creates an instance of TListControlAction (see page 11886).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TListControlAction(TComponent * AOwner);
```

Description

Most applications do not need to call the TListControlAction (see page 11886) constructor because actions are added at design time to an action list or action manager.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TListControlAction (see page 11886) instance.

See Also

Owner (see page 2807)

67.1.5.3 TListControlAction Methods

The methods of the TListControlAction class are listed here.

Protected Methods

	Name	Description
	IsEnabled (see page 11892)	This is IsEnabled, a member of class TListControlAction.
V	Notification (see page 11892)	Responds when components are created or destroyed.

Public Methods

	Name	Description
V	HandlesTarget (see page 11892)	Indicates whether the action can execute its function.

67.1.5.3.1 TListControlAction.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the currently active object.

HandlesTarget checks that the target is a TCustomListControl descendant (or that the ListControl (see page 11893) property is set if Target is nil (Delphi) or NULL (C++)). HandlesTarget returns true if there is a list control to act as target, false otherwise.

See Also

ExecuteTarget (see page 2712)

TCustomListControl (see page 5928)

ListControl (see page 11893)

67.1.5.3.2 TListControlAction.IsEnabled Method

Pascal

```
function IsEnabled(Target: TObject): Boolean;
```

C++

```
__fastcall Boolean IsEnabled(TObject * Target);
```

Description

This is IsEnabled, a member of class TListControlAction.

67.1.5.3.3 TListControlAction.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TBasicAction overrides the Notification method to check whether the component that is the value of the ActionComponent property is about to be freed. If so, it sets the ActionComponent property to nil (Delphi) or NULL (C++).

See Also

ActionComponent (see page 2715)

67.1.5.4 TListControlAction Properties

The properties of the TListControlAction class are listed here.

Public Properties

	Name	Description
	ListControl (see page 11893)	Provides access to the target of the action.

67.1.5.4.1 TListControlAction.ListControl Property

Provides access to the target of the action.

Pascal

```
property ListControl: TCustomListControl;
```

C++

```
__property TCustomListControl ListControl;
```

Description

ListControl is the list control on which the action performs its function. It is a TCustomListControl descendant, such a TListView or TListBox.

Set ListControl to associate the action with a particular control. If ListControl is not explicitly set, it is assumed to be the active listbox or list view when the action fires.

See Also

TListView (see page 4285)

TListBox (see page 21598)

67.1.6 TListControlClearSelection Class

TListControlClearSelection deselects all selected items in a list control (list view or list box).

Class Hierarchy

Pascal

```
TListControlClearSelection = class(TListControlAction);
```

C++

```
class TListControlClearSelection : public TListControlAction;
```

File

ExtActns

Description

Add TListControlClearSelection to an action list to allow users to deselect any selected items of a list control by using menu items or toolbar buttons. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to deselect all selected items in the target control.

TListControlClearSelection is designed to work with a list control (TCustomListControl descendant) as a target. TListControlClearSelection automatically disables itself if the target control is not a list control or if that list box or list view has no selected items.

See Also

TListControlCopySelection

TListControlMoveSelection

TListControlDeleteSelection

TListControlSelectAll

TCustomListControl (see page 5928)

67.1.6.1 TListControlClearSelection Members

The following tables list the members exposed by TListControlClearSelection.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.

	AssignTo (see page 303)	Copies the properties of this action to a destination object.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsEnabled (see page 11892)	This is IsEnabled, a member of class TListControlAction.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 11892)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 11891)	Creates an instance of TListControlAction (see page 11886).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 11898)	Deselects the currently selected items in the target list control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 11892)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 11899)	Enables or disables the action, depending on the state of the target control.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.

	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnHint (see page 11904)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11904)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 11900)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Enabled (see page 11900)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 11900)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11901)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11901)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11902)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11902)	Indicates the ImageIndex property for client controls and menu items.
	ListControl (see page 11902)	Provides access to the target of the action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 11903)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 11903)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 11904)	Specifies the Visible property for client controls and menu items.

67.1.6.2 TListControlClearSelection Methods

The methods of the TListControlClearSelection class are listed here.

Public Methods

	Name	Description
V	ExecuteTarget (see page 11898)	Deselects the currently selected items in the target list control.
V	UpdateTarget (see page 11899)	Enables or disables the action, depending on the state of the target control.

67.1.6.2.1 TListControlClearSelection.ExecuteTarget Method

Deselects the currently selected items in the target list control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires. It should always be a TCustomListControl descendant (because HandlesTarget (see page 11892) returns false if it is not).

ExecuteTarget sets the ListControl (see page 11902) property to Target it is not already set. It then calls the list control's ClearSelection method, deselecting any selected items.

See Also

TCustomListControl (see page 5928)

ListControl (see page 11893)

ClearSelection (see page 5940)

67.1.6.2.2 TListControlClearSelection.UpdateTarget Method

Enables or disables the action, depending on the state of the target control.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TListControlClearSelection (see page 11893), UpdateTarget checks whether the current target object (specified by the ListControl (see page 11902) property if it is set, otherwise by the Target parameter) is a list control that includes at least one selected item. If so, UpdateTarget sets Enabled (see page 11900) to true. If not, UpdateTarget sets Enabled (see page 11900) to false.

See Also

Enabled (see page 310)

SelCount (see page 5957)

ListControl (see page 11893)

67.1.6.3 TListControlClearSelection Properties

The properties of the TListControlClearSelection class are listed here.

Published Properties

	Name	Description
	Caption (see page 11900)	Represents the caption of client controls and menu items.
	Enabled (see page 11900)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 11900)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11901)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11901)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11902)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11902)	Indicates the ImageIndex property for client controls and menu items.
	ListControl (see page 11902)	Provides access to the target of the action.
	SecondaryShortCuts (see page 11903)	Specifies the short cuts (in addition to ShortCut) for triggering clients.

	ShortCut (see page 11903)	Specifies the ShortCut property for client menu items.
	Visible (see page 11904)	Specifies the Visible property for client controls and menu items.

67.1.6.3.1 TListControlClearSelection.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

67.1.6.3.2 TListControlClearSelection.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

67.1.6.3.3 TListControlClearSelection.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value.

Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

67.1.6.3.4 TListControlClearSelection.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

67.1.6.3.5 TListControlClearSelection.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

67.1.6.3.6 TListControlClearSelection.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

67.1.6.3.7 TListControlClearSelection.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

67.1.6.3.8 TListControlClearSelection.ListControl Property

Provides access to the target of the action.

Pascal

```
property ListControl: TCustomListControl;
```

C++

```
__property TCustomListControl ListControl;
```

Description

ListControl is the list control on which the action performs its function. It is a TCustomListControl descendant, such a TListView or TListBox.

Set ListControl to associate the action with a particular control. If ListControl is not explicitly set, it is assumed to be the active listbox or list view when the action fires.

See Also

TListView (see page 4285)

TListBox (see page 21598)

67.1.6.3.9 TListControlClearSelection.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

67.1.6.3.10 TListControlClearSelection.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

67.1.6.3.11 TListControlClearSelection.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

67.1.6.4 TListControlClearSelection Events

The events of the TListControlClearSelection class are listed here.

Published Events

	Name	Description
	OnHint (see page 11904)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11904)	Occurs when the application is idle or when the action list updates.

67.1.6.4.1 TListControlClearSelection.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

67.1.6.4.2 TListControlClearSelection.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

67.1.7 TListControlCopySelection Class

TListControlCopySelection copies the selected items in a list control (list view or list box) to another (destination) list control.

Class Hierarchy

Pascal

```
TListControlCopySelection = class(TListControlAction);
```

C++

```
class TListControlCopySelection : public TListControlAction;
```

File

ExtActns

Description

Add TListControlCopySelection to an action list to allow users to copy the selected items of a list control to another list control by using menu items or toolbar buttons. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to copy all selected items in the target control to another (destination) list control.

Your application must explicitly set the Destination (see page 11913) property to indicate the list to which items are copied.

TListControlCopySelection is designed to work with a list control (TCustomListControl descendant) as a target. This target is the source list control, not the destination. TListControlCopySelection automatically disables itself if the target control is not a list control, if that list box or list view has no selected items, or if there is no destination list control assigned.

See Also

TListControlClearSelection

TListControlMoveSelection

TListControlDeleteSelection

TListControlSelectAll

TCustomListControl (see page 5928)

67.1.7.1 TListControlCopySelection Members

The following tables list the members exposed by TListControlCopySelection.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsEnabled (see page 11892)	This is IsEnabled, a member of class TListControlAction.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 11892)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 11891)	Creates an instance of TListControlAction (see page 11886).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 11910)	Copies the selected items in the target list control to the list control specified by Destination (see page 11913).

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 11911)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 11912)	Enables or disables the action, depending on the state of the target control.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnHint (see page 11917)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11918)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 11912)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Destination (see page 11913)	Specifies the list control to which items are copied.
	Enabled (see page 11913)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 11914)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11914)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11915)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11915)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11915)	Indicates the ImageIndex property for client controls and menu items.
	ListControl (see page 11916)	Provides access to the target of the action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 11916)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 11916)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 11917)	Specifies the Visible property for client controls and menu items.

67.1.7.2 TListControlCopySelection Methods

The methods of the TListControlCopySelection class are listed here.

Public Methods

	Name	Description
V	ExecuteTarget (see page 11910)	Copies the selected items in the target list control to the list control specified by Destination (see page 11913).
V	HandlesTarget (see page 11911)	Indicates whether the action can execute its function.
V	UpdateTarget (see page 11912)	Enables or disables the action, depending on the state of the target control.

67.1.7.2.1 TListControlCopySelection.ExecuteTarget Method

Copies the selected items in the target list control to the list control specified by Destination (see page 11913).

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires. It should always be a TCustomListControl descendant (because HandlesTarget (see page 11911) returns false if it is not).

ExecuteTarget sets the ListControl (see page 11916) property to Target it is not already set. It then calls the list control's ClearSelection method, copying any selected items to the list box or list view specified by the Destination (see page 11913) property.

See Also

TCustomListControl (see page 5928)

ListControl (see page 11893)

CopySelection (see page 5941)

Destination

67.1.7.2.2 TListControlCopySelection.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the currently active object.

HandlesTarget checks that the target is a TCustomListControl descendant (or that the ListControl (see page 11916) property is set if Target is nil (Delphi) or NULL (C++)). It also ensures that the Destination (see page 11913) property has been asssigned. HandlesTarget returns true if there is a list control to act as target and a destination control, false otherwise.

See Also

ExecuteTarget (see page 2712)

TCustomListControl (see page 5928)

ListControl (see page 11893)

Destination

67.1.7.2.3 TListControlCopySelection.UpdateTarget Method

Enables or disables the action, depending on the state of the target control.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TListControlCopySelection (see page 11906), UpdateTarget checks whether the current target object (specified by the ListControl (see page 11916) property if it is set, otherwise by the Target parameter) is a list control that includes at least one selected item. If so, UpdateTarget sets Enabled (see page 11913) to true. If not, UpdateTarget sets Enabled (see page 11913) to false.

See Also

Enabled (see page 310)

SelCount (see page 5957)

ListControl (see page 11893)

67.1.7.3 TListControlCopySelection Properties

The properties of the TListControlCopySelection class are listed here.

Published Properties

	Name	Description
	Caption (see page 11912)	Represents the caption of client controls and menu items.
	Destination (see page 11913)	Specifies the list control to which items are copied.
	Enabled (see page 11913)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 11914)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11914)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11915)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11915)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11915)	Indicates the ImageIndex property for client controls and menu items.
	ListControl (see page 11916)	Provides access to the target of the action.
	SecondaryShortCuts (see page 11916)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 11916)	Specifies the ShortCut property for client menu items.
	Visible (see page 11917)	Specifies the Visible property for client controls and menu items.

67.1.7.3.1 TListControlCopySelection.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

67.1.7.3.2 TListControlCopySelection.Destination Property

Specifies the list control to which items are copied.

Pascal

```
property Destination: TCustomListControl;
```

C++

```
__property TCustomListControl Destination;
```

Description

Destination is the list control to which the action copies selected items from its target. It is a TCustomListControl descendant, such a TListView or TListBox.

You must explicitly set Destination to indicate the list control that receives the new items. These items are copied from the list control specified by the ListControl (see page 11916) property. Destination and ListControl (see page 11916) do not need to be the same type of control, as long as they are both TCustomListControl descendants.

See Also

TListView (see page 4285)

TListBox (see page 21598)

ListControl (see page 11893)

67.1.7.3.3 TListControlCopySelection.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

67.1.7.3.4 TListControlCopySelection.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

67.1.7.3.5 TListControlCopySelection.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

67.1.7.3.6 TListControlCopySelection.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

67.1.7.3.7 TListControlCopySelection.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

67.1.7.3.8 TListControlCopySelection.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

67.1.7.3.9 TListControlCopySelection.ListControl Property

Provides access to the target of the action.

Pascal

```
property ListControl: TCustomListControl;
```

C++

```
__property TCustomListControl ListControl;
```

Description

ListControl is the list control on which the action performs its function. It is a TCustomListControl descendant, such a TListView or TListBox.

Set ListControl to associate the action with a particular control. If ListControl is not explicitly set, it is assumed to be the active listbox or list view when the action fires.

See Also

TListView (see page 4285)

TListBox (see page 21598)

67.1.7.3.10 TListControlCopySelection.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

67.1.7.3.11 TListControlCopySelection.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

67.1.7.3.12 TListControlCopySelection.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

67.1.7.4 TListControlCopySelection Events

The events of the TListControlCopySelection class are listed here.

Published Events

	Name	Description
	OnHint (see page 11917)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11918)	Occurs when the application is idle or when the action list updates.

67.1.7.4.1 TListControlCopySelection.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The

OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

67.1.7.4.2 TListControlCopySelection.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
```

```
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

67.1.8 TListControlDeleteSelection Class

TListControlDeleteSelection deletes the selected items in a list control (list view or listbox).

Class Hierarchy

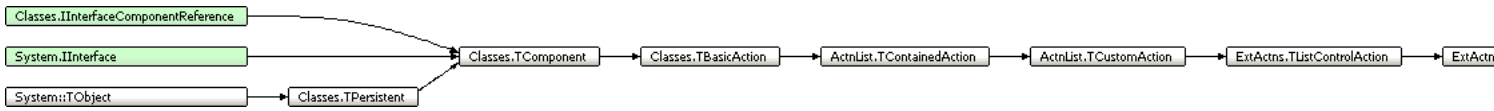

Pascal

```
TListControlDeleteSelection = class(TListControlAction);
```

C++

```
class TListControlDeleteSelection : public TListControlAction;
```

File

ExtActns

Description

Add TListControlDeleteSelection to an action list to allow users to delete the selected items of a list control by using menu items or toolbar buttons. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to delete all selected items in the target control (a listbox or list view).

TListControlDeleteSelection is designed to work with a list control (TCustomListControl descendant) as a target. TListControlDeleteSelection automatically disables itself if the target control is not a list control or if that list box or list view has no selected items.

See Also

TListControlClearSelection

TListControlMoveSelection

TListControlDeleteSelection

TListControlSelectAll

TCustomListControl (see page 5928)

67.1.8.1 TListControlDeleteSelection Members

The following tables list the members exposed by TListControlDeleteSelection.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsEnabled (see page 11892)	This is IsEnabled, a member of class TListControlAction.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 11892)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 11891)	Creates an instance of TListControlAction (see page 11886).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.

	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 11924)	Deletes the selected items in the target list control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 11892)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.

V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 11924)	Enables or disables the action, depending on the state of the target control.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnHint (see page 11929)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11930)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 11925)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Enabled (see page 11925)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 11926)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11926)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11927)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11927)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11927)	Indicates the ImageIndex property for client controls and menu items.
	ListControl (see page 11928)	Provides access to the target of the action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 11928)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 11929)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 11929)	Specifies the Visible property for client controls and menu items.

67.1.8.2 TListControlDeleteSelection Methods

The methods of the TListControlDeleteSelection class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 11924)	Deletes the selected items in the target list control.
	UpdateTarget (see page 11924)	Enables or disables the action, depending on the state of the target control.

67.1.8.2.1 TListControlDeleteSelection.ExecuteTarget Method

Deletes the selected items in the target list control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires. It should always be a TCustomListControl descendant (because HandlesTarget (see page 11892) returns false if it is not).

ExecuteTarget sets the ListControl (see page 11928) property to Target it is not already set. It then calls the list control's DeleteSelected method, deleting any selected items.

See Also

TCustomListControl (see page 5928)

ListControl (see page 11893)

DeleteSelected (see page 5941)

67.1.8.2.2 TListControlDeleteSelection.UpdateTarget Method

Enables or disables the action, depending on the state of the target control.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TListControlDeleteSelection (see page 11919), UpdateTarget checks whether the current target object (specified by the ListControl (see page 11928) property if it is set, otherwise by the Target parameter) is a list control that includes at least one selected item. If so, UpdateTarget sets Enabled (see page 11925) to true. If not, UpdateTarget sets Enabled (see page 11925) to false.

See Also

Enabled (see page 310)

SelCount (see page 5957)

ListControl (see page 11893)

67.1.8.3 TListControlDeleteSelection Properties

The properties of the TListControlDeleteSelection class are listed here.

Published Properties

	Name	Description
	Caption (see page 11925)	Represents the caption of client controls and menu items.
	Enabled (see page 11925)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 11926)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11926)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11927)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11927)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11927)	Indicates the ImageIndex property for client controls and menu items.
	ListControl (see page 11928)	Provides access to the target of the action.
	SecondaryShortCuts (see page 11928)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 11929)	Specifies the ShortCut property for client menu items.
	Visible (see page 11929)	Specifies the Visible property for client controls and menu items.

67.1.8.3.1 TListControlDeleteSelection.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

67.1.8.3.2 TListControlDeleteSelection.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

67.1.8.3.3 TListControlDeleteSelection.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

67.1.8.3.4 TListControlDeleteSelection.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

67.1.8.3.5 TListControlDeleteSelection.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

67.1.8.3.6 TListControlDeleteSelection.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

67.1.8.3.7 TListControlDeleteSelection.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

67.1.8.3.8 TListControlDeleteSelection.ListControl Property

Provides access to the target of the action.

Pascal

```
property ListControl: TCustomListControl;
```

C++

```
__property TCustomListControl ListControl;
```

Description

ListControl is the list control on which the action performs its function. It is a TCustomListControl descendant, such a TListView or TListBox.

Set ListControl to associate the action with a particular control. If ListControl is not explicitly set, it is assumed to be the active listbox or list view when the action fires.

See Also

TListView (see page 4285)

TListBox (see page 21598)

67.1.8.3.9 TListControlDeleteSelection.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

67.1.8.3.10 TListControlDeleteSelection.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

67.1.8.3.11 TListControlDeleteSelection.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

67.1.8.4 TListControlDeleteSelection Events

The events of the TListControlDeleteSelection class are listed here.

Published Events

	Name	Description
	OnHint (see page 11929)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11930)	Occurs when the application is idle or when the action list updates.

67.1.8.4.1 TListControlDeleteSelection.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

67.1.8.4.2 TListControlDeleteSelection.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
```

```
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

67.1.9 TListControlMoveSelection Class

TListControlMoveSelection moves the selected items in a list control (list view or list box) to another (destination) list control.

Class Hierarchy

Pascal

```
TListControlMoveSelection = class(TListControlCopySelection);
```

C++

```
class TListControlMoveSelection : public TListControlCopySelection;
```

File

ExtActns

Description

Add TListControlMoveSelection to an action list to allow users to move the selected items of a list control to another list control by using menu items or toolbar buttons. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to move all selected items in the target control to another (destination) list control.

Your application must explicitly set the Destination (see page 11913) property to indicate the list to which items are copied.

TListControlMoveSelection is designed to work with a list control (TCustomListControl descendant) as a target. This target is the source list control, not the destination. TListControlMoveSelection automatically disables itself if the target control is not a list control, if that list box or list view has no selected items, or if there is no destination list control assigned.

See Also

TListControlClearSelection

TListControlCopySelection

TListControlDeleteSelection

TListControlSelectAll

TCustomListControl (see page 5928)

67.1.9.1 TListControlMoveSelection Members

The following tables list the members exposed by TListControlMoveSelection.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsEnabled (see page 11892)	This is IsEnabled, a member of class TListControlAction.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 11892)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.

	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.

	Create (see page 11891)	Creates an instance of TListControlAction (see page 11886).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 11936)	Moves the selected items in the target list control to the list control specified by Destination (see page 11913).
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 11892)	Indicates whether the action can execute its function.
	HandlesTarget (see page 11911)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.
V	UpdateTarget (see page 11912)	Enables or disables the action, depending on the state of the target control.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
	ListControl (see page 11893)	Provides access to the target of the action.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnHint (see page 11917)	Occurs when the mouse pauses over a client control or menu item.

	OnUpdate (see page 11918)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 11912)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Destination (see page 11913)	Specifies the list control to which items are copied.
	Enabled (see page 11913)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 11914)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11914)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11915)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11915)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11915)	Indicates the ImageIndex property for client controls and menu items.
	ListControl (see page 11916)	Provides access to the target of the action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 11916)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 11916)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 11917)	Specifies the Visible property for client controls and menu items.

67.1.9.2 TListControlMoveSelection Methods

The methods of the TListControlMoveSelection class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 11936)	Moves the selected items in the target list control to the list control specified by Destination (see page 11913).

67.1.9.2.1 TListControlMoveSelection.ExecuteTarget Method

Moves the selected items in the target list control to the list control specified by Destination (see page 11913).

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires. It should always be a TCustomListControl descendant (because HandlesTarget (see page 11911) returns false if it is not).

ExecuteTarget sets the ListControl (see page 11916) property to Target it is not already set. It then calls the list control's MoveSelection method, copying any selected items to the list box or list view specified by the Destination (see page 11913) property and removing them from the target (source list control).

See Also

TCustomListControl (see page 5928)

ListControl (see page 11893)

MoveSelection (see page 5942)

Destination

67.1.10 TListControlSelectAll Class

TListControlSelectAll selects all the items in a list control (list view or listbox).

Class Hierarchy

Pascal

```
TListControlSelectAll = class(TListControlAction);
```

C++

```
class TListControlSelectAll : public TListControlAction;
```

File

ExtActns

Description

Add TListControlSelectAll to an action list to allow users to select all the items of a list control by using menu items or toolbar buttons. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to select all items in the target control (a listbox or list view).

TListControlSelectAll is designed to work with a list control (TCustomListControl descendant) as a target. TListControlSelectAll automatically disables itself if the target control is not a list control, if that target control does not permit multiple selection, or if all of the items in the target control are already selected.

See Also

TListControlClearSelection

TListControlMoveSelection

TListControlSelectAll

TListControlSelectAll

TCustomListControl (see page 5928)

67.1.10.1 TListControlSelectAll Members

The following tables list the members exposed by TListControlSelectAll.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsEnabled (see page 11892)	This is IsEnabled, a member of class TListControlAction.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 11892)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 11891)	Creates an instance of TListControlAction (see page 11886).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".

	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 11942)	Selects all the items in the target list control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 11892)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 11942)	Enables or disables the action, depending on the state of the target control.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.

	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnHint (see page 11947)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11948)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 11943)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Enabled (see page 11943)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 11944)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11944)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11945)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11945)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11945)	Indicates the ImageIndex property for client controls and menu items.
	ListControl (see page 11946)	Provides access to the target of the action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 11946)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 11946)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 11947)	Specifies the Visible property for client controls and menu items.

67.1.10.2 TListControlSelectAll Methods

The methods of the TListControlSelectAll class are listed here.

Public Methods

	Name	Description
V	ExecuteTarget (see page 11942)	Selects all the items in the target list control.
V	UpdateTarget (see page 11942)	Enables or disables the action, depending on the state of the target control.

67.1.10.2.1 TListControlSelectAll.ExecuteTarget Method

Selects all the items in the target list control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires. It should always be a TCustomListControl descendant (because HandlesTarget (see page 11892) returns false if it is not).

ExecuteTarget sets the ListControl (see page 11946) property to Target it is not already set. It then calls the list control's SelectAll method, selecting all of its items.

See Also

TCustomListControl (see page 5928)

ListControl (see page 11893)

SelectAll (see page 5943)

67.1.10.2.2 TListControlSelectAll.UpdateTarget Method

Enables or disables the action, depending on the state of the target control.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TListControlSelectAll (see page 11937), UpdateTarget checks whether the current target object (specified by the ListControl (see page 11946) property if it is set, otherwise by the Target parameter) is a list control that includes allows multiple selections and that includes at least one unselected item. If so, UpdateTarget sets Enabled (see page 11943) to true. If not, UpdateTarget sets Enabled (see page 11943) to false.

See Also

Enabled (see page 310)

MultiSelect (see page 5957)

ListControl (see page 11893)

67.1.10.3 TListControlSelectAll Properties

The properties of the TListControlSelectAll class are listed here.

Published Properties

	Name	Description
	Caption (see page 11943)	Represents the caption of client controls and menu items.
	Enabled (see page 11943)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 11944)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11944)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11945)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11945)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11945)	Indicates the ImageIndex property for client controls and menu items.
	ListControl (see page 11946)	Provides access to the target of the action.
	SecondaryShortCuts (see page 11946)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 11946)	Specifies the ShortCut property for client menu items.
	Visible (see page 11947)	Specifies the Visible property for client controls and menu items.

67.1.10.3.1 TListControlSelectAll.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

67.1.10.3.2 TListControlSelectAll.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

67.1.10.3.3 TListControlSelectAll.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

67.1.10.3.4 TListControlSelectAll.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

67.1.10.3.5 TListControlSelectAll.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

67.1.10.3.6 TListControlSelectAll.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

67.1.10.3.7 TListControlSelectAll.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an

image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

67.1.10.3.8 TListControlSelectAll.ListControl Property

Provides access to the target of the action.

Pascal

```
property ListControl: TCustomListControl;
```

C++

```
__property TCustomListControl ListControl;
```

Description

ListControl is the list control on which the action performs its function. It is a TCustomListControl descendant, such a TListView or TListBox.

Set ListControl to associate the action with a particular control. If ListControl is not explicitly set, it is assumed to be the active listbox or list view when the action fires.

See Also

TListView (see page 4285)

TListBox (see page 21598)

67.1.10.3.9 TListControlSelectAll.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

67.1.10.3.10 TListControlSelectAll.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

67.1.10.3.11 TListControlSelectAll.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

67.1.10.4 TListControlSelectAll Events

The events of the TListControlSelectAll class are listed here.

Published Events

	Name	Description
	OnHint (see page 11947)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11948)	Occurs when the application is idle or when the action list updates.

67.1.10.4.1 TListControlSelectAll.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The

OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

67.1.10.4.2 TListControlSelectAll.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
```

```
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

67.1.11 TNextTab Class

TNextTab moves a tabbed control to its next page.

Class Hierarchy

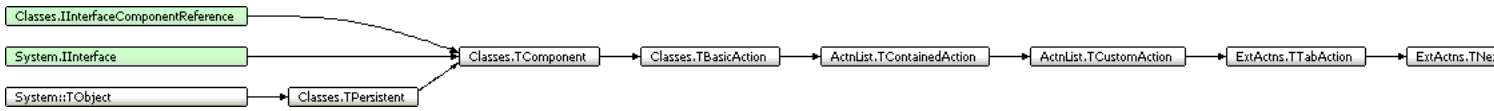

Pascal

```
TNextTab = class(TTabAction);
```

C++

```
class TNextTab : public TTabAction;
```

File

ExtActns

Description

Add TNextTab to an action list to provide the response to the "Next" and "Finish" buttons or menu items in a tabbed dialog. Controls (see page 5711) such as menu items and tool buttons linked to this action cause a different response, depending on the value of the Wrap (see page 11960) property and on whether the target control is on the last page:

If the target page control is not on the last page, or if Wrap (see page 11960) is true, client controls cause this action to change the current page to the next page. (When Wrap (see page 11960) is true, the next page after the last tab is the first tabbed page).

If Wrap (see page 11960) is false and the target page control is on the last page, this action changes the caption of client controls (for example, from "Next" to "Finish") and, when fired, responds by generating an OnFinish (see page 11962) event.

Before you can use TNextTab, you must assign a target page control or tab control as the value of the TabControl (see page 11960) property. Once the target control is assigned, TNextTab automatically disables itself if the target control is not enabled.

See Also

TPreviousTab (see page 11977)

TPageControl (see page 4459)

TTabControl (see page 4841)

67.1.11.1 TNextTab Members

The following tables list the members exposed by TNextTab.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 12079)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SelectNextTab (see page 12079)	This is SelectNextTab, a member of class TTabAction.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 12078)	Creates an instance of TTabAction (see page 12073).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.

	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 11954)	Changes the selected tab in the target tab control to the next one, or generates an OnFinish (see page 11962) event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 12078)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 11955)	Updates the action to reflect current conditions.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterTabChange (see page 11961)	Occurs immediately after the action changes the page of the associated tab control.
	BeforeTabChange (see page 11962)	Occurs immediately before the action changes the page of the associated tab control.
	OnFinish (see page 11962)	Occurs when the action fires and the target page control is on its last page.
	OnHint (see page 11963)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11963)	Occurs when the application is idle or when the action list updates.
	OnValidateTab (see page 11964)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 11956)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Enabled (see page 11956)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 11956)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11957)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11957)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11958)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11958)	Indicates the ImageIndex property for client controls and menu items.
	LastTabCaption (see page 11958)	Specifies the caption when the target page control is on its last page and Wrap (see page 11960) is false.
	Name (see page 2805)	Specifies the name of the component as referenced in code.

	ShortCut (see page 11959)	Specifies the ShortCut property for client menu items.
	SkipHiddenTabs (see page 11959)	Indicates whether the tab action should consider hidden tabbed pages when moving to the next page.
	TabControl (see page 11960)	Specifies the target of the action.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 11960)	Specifies the Visible property for client controls and menu items.
	Wrap (see page 11960)	Indicates whether the tab action should "wrap" when navigating to the next tab page.

67.1.11.2 TNextTab Methods

The methods of the TNextTab class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 11954)	Changes the selected tab in the target tab control to the next one, or generates an OnFinish (see page 11962) event.
	UpdateTarget (see page 11955)	Updates the action to reflect current conditions.

67.1.11.2.1 TNextTab.ExecuteTarget Method

Changes the selected tab in the target tab control to the next one, or generates an OnFinish (see page 11962) event.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

ExecuteTarget causes the action to perform its default response based on the state of the target tab control.

Target is the currently active object when the action fires. TNextTab (see page 11949) ignores this parameter and uses the TabControl (see page 11960) property to identify the target control.

ExecuteTarget performs differently, depending on whether the target control is on the last page and on the value of the Wrap (see page 11960) property:

If Wrap (see page 11960) is true or the target control is not on the last page, ExecuteTarget generates a BeforeTabChange (see page 11962) event, changes the target control to the next page, and generates an AfterTabChange (see page 11961) event. If Wrap (see page 11960) is true and the target control is on the last page, this causes the target control to move to the first page. Note that the SkipHiddenTabs (see page 11959) property determines whether the next page can include a hidden page.

If Wrap (see page 11960) is false and the target control is on the last page, ExecuteTarget generates an OnFinish (see page 11962) event.

See Also

ExecuteTarget (see page 2712)

ActivePageIndex (see page 4484)

TabControl (see page 12080)

OnFinish (see page 11962)

BeforeTabChange (see page 12082)

AfterTabChange (see page 12082)

SkipHiddenTabs (see page 12080)

Wrap (see page 12081)

TabIndex (see page 3946)

67.1.11.2.2 TNextTab.UpdateTarget Method

Updates the action to reflect current conditions.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TNextTab (see page 11949), UpdateTarget first checks whether the TabControl (see page 11960) property specifies an enabled control that can act as a target. If not, UpdateTarget sets the Enabled (see page 11956) property to false.

If the target control is enabled, UpdateTarget sets the Enabled (see page 11956) property to true and checks whether the target control is on its last page. If so, (and if Wrap (see page 11960) is false) it changes the Caption (see page 11956) property to the value of LastTabCaption (see page 11958). If not, it restores the Caption (see page 11956) property to its original value.

See Also

Enabled (see page 310)

Caption (see page 308)

LastTabCaption (see page 11958)

TabControl (see page 12080)

Wrap (see page 12081)

67.1.11.3 TNextTab Properties

The properties of the TNextTab class are listed here.

Published Properties

	Name	Description
	Caption (see page 11956)	Represents the caption of client controls and menu items.
	Enabled (see page 11956)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 11956)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11957)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11957)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11958)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11958)	Indicates the ImageIndex property for client controls and menu items.
	LastTabCaption (see page 11958)	Specifies the caption when the target page control is on its last page and Wrap (see page 11960) is false.

	ShortCut (see page 11959)	Specifies the ShortCut property for client menu items.
	SkipHiddenTabs (see page 11959)	Indicates whether the tab action should consider hidden tabbed pages when moving to the next page.
	TabControl (see page 11960)	Specifies the target of the action.
	Visible (see page 11960)	Specifies the Visible property for client controls and menu items.
	Wrap (see page 11960)	Indicates whether the tab action should "wrap" when navigating to the next tab page.

67.1.11.3.1 TNextTab.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

67.1.11.3.2 TNextTab.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

67.1.11.3.3 TNextTab.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

67.1.11.3.4 TNextTab.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

67.1.11.3.5 TNextTab.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

67.1.11.3.6 TNextTab.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

67.1.11.3.7 TNextTab.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

67.1.11.3.8 TNextTab.LastTabCaption Property

Specifies the caption when the target page control is on its last page and Wrap (see page 11960) is false.

Pascal

```
property LastTabCaption: String;
```

C++

```
__property AnsiString LastTabCaption;
```

Description

Set LastTabCaption to the caption of target controls when the control specified by the TabControl (see page 11960) property is on its last page and the action can't tab beyond the last page. When the target page control is not on the last page, the caption for client controls is given by the Caption (see page 11956) property. When the target page control is on the last page (and Wrap (see page 11960) is false), TNextTab (see page 11949) caches the original value of Caption (see page 11956) and changes the Caption (see page 11956) property to LastTabCaption. If the target page control returns to a previous page, TNextTab (see page 11949) then restores the original value of the Caption (see page 11956) property.

LastTabCaption is designed for client controls such as the button in a tabbed dialog that changes from a "Next" button to a "Finish" button when the tabbed dialog reaches the last page.

See Also

Caption (see page 308)

TabControl (see page 12080)

Wrap (see page 12081)

67.1.11.3.9 TNextTab.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

67.1.11.3.10 TNextTab.SkipHiddenTabs Property

Indicates whether the tab action should consider hidden tabbed pages when moving to the next page.

Pascal

```
property SkipHiddenTabs: Boolean;
```

C++

```
__property Boolean SkipHiddenTabs;
```

Description

Set SkipHiddenTabs to true if you do not want the action to move to a tab page if its TabVisible property is false. Set SkipHiddenTabs to false if the tab action can move to and make visible a hidden page.

Note: SkipHiddenTabs only has an effect when the TabControl property identifies a TPageControl object. (TTabControl does not support hidden pages).

See Also

ExecuteTarget (see page 11954)

ExecuteTarget (see page 11982)

TabVisible (see page 4956)

TPageControl (see page 4459)

67.1.11.3.11 TNextTab.TabControl Property

Specifies the target of the action.

Pascal

```
property TabControl: TCustomTabControl;
```

C++

```
__property TCustomTabControl TabControl;
```

Description

Set TabControl to the control on which the action operates when its ExecuteTarget method is called. Typically, this is a page control or tab control object that implements the pages of a tabbed dialog.

An application must explicitly assign a value to the TabControl property before the TTabAction descendant is enabled. The action does not infer the target page control when it fires.

See Also

ExecuteTarget (see page 11954)

ExecuteTarget (see page 11982)

67.1.11.3.12 TNextTab.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

67.1.11.3.13 TNextTab.Wrap Property

Indicates whether the tab action should "wrap" when navigating to the next tab page.

Pascal

```
property Wrap: Boolean;
```

C++

```
__property Boolean Wrap;
```

Description

When Wrap is true, the tab action can move previous to the first tabbed page or after the last tabbed page in the associated control. When Wrap is false, an attempt to move beyond the first or last page does not result in any page change, although it still results in a BeforeTabChange and an AfterTabChange event or in an OnFinish event.

See Also

ExecuteTarget (see page 11954)

ExecuteTarget (see page 11982)

TabControl (see page 12080)

BeforeTabChange (see page 12082)

AfterTabChange (see page 12082)

OnFinish (see page 11962)

67.1.11.4 TNextTab Events

The events of the TNextTab class are listed here.

Published Events

	Name	Description
	AfterTabChange (see page 11961)	Occurs immediately after the action changes the page of the associated tab control.
	BeforeTabChange (see page 11962)	Occurs immediately before the action changes the page of the associated tab control.
	OnFinish (see page 11962)	Occurs when the action fires and the target page control is on its last page.
	OnHint (see page 11963)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11963)	Occurs when the application is idle or when the action list updates.
	OnValidateTab (see page 11964)	Occurs when the application is idle or when the action list updates.

67.1.11.4.1 TNextTab.AfterTabChange Event

Occurs immediately after the action changes the page of the associated tab control.

Pascal

```
property AfterTabChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterTabChange;
```

Description

Write an AfterTabChange event handler to perform any additional tasks after the tab action changes the current page of the associated tab control. Typically, this event is used to initialize the current page of the tab control.

Note: AfterTabChange only occurs if the action tries to change the page of the tab control (even if it is not successful). If the action attempts some other task (for example, when TNextPage is on the last page of the tab control), AfterTabChange does not occur.

See Also

BeforeTabChange (see page 12082)

TabControl (see page 12080)

Wrap (see page 12081)

67.1.11.4.2 TNextTab.BeforeTabChange Event

Occurs immediately before the action changes the page of the associated tab control.

Pascal

```
property BeforeTabChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeTabChange;
```

Description

Write a BeforeTabChange event handler to perform any additional tasks before the tab action changes the current page of the associated tab control. Typically, this event is used to save or respond to the changes the user made on the current page before it changes.

Note: BeforeTabChange only occurs if the action tries to change the page of the tab control (even if it is not successful). If the action attempts some other task (for example, when TNextPage is on the last page of the tab control), BeforeTabChange does not occur.

See Also

AfterTabChange (see page 12082)

TabControl (see page 12080)

Wrap (see page 12081)

67.1.11.4.3 TNextTab.OnFinish Event

Occurs when the action fires and the target page control is on its last page.

Pascal

```
property OnFinish: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnFinish;
```

Description

Write an OnFinish event handler to respond when the user indicates that they are finished with the tabbed dialog by clicking a client control when the tabbed dialog is on its last page. OnFinish only occurs when the target control is on its last page and the Wrap (see page 11960) property is false. When the target page control is on any other page, or when Wrap (see page 11960) is true, TNextTab (see page 11949) generates an BeforeTabChange (see page 11962) event, changes the current page of the target control, and generates an AfterTabChange (see page 11961) event.

Note: Use the OnFinish event rather than the action's OnExecute event. If you supply an event handler to the OnExecute event, it blocks the default response by this action, and the action won't change the current page when the target tabbed dialog is not on its last page.

See Also

AfterTabChange (see page 12082)

BeforeTabChange (see page 12082)

TabControl (see page 12080)

Wrap (see page 12081)

67.1.11.4.4 TNextTab.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

67.1.11.4.5 TNextTab.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
```

```
    // Indicate whether ToolBar1 is currently visible.
    TAction* pa = dynamic_cast<TAction*>(Sender);
    if (pa)
    {
      if (pa->Checked != ToolBar1->Visible)
      {
        lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
      }
      pa->Checked = ToolBar1->Visible;
    }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

67.1.11.4.6 TNextTab.OnValidateTab Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnValidateTab: TValidateTabEvent;
```

C++

```
__property TValidateTabEvent OnValidateTab;
```

Description

Use the OnValidateTab event rather than the OnUpdate event to update the properties of a tab action when the application is idle. Unlike the OnUpdate event, which prevents the default behavior of the UpdateTarget method, OnValidateTab occurs after UpdateTarget has set the Enabled property to reflect whether the target tab control is enabled.

OnValidateTab can be used to update any of the properties of the tab action, but it is intended primarily for enabling or disabling the action, depending on the contents of the current page of the target tab control.

Sender is the tab action component to update in this event.

TabControl is the target tab control.

Enabled reflects the Enabled property of this action. Set Enabled in the event handler to enable or disable the action depending on current conditions.

See Also

OnUpdate (see page 2718)

UpdateTarget (see page 12079)

TabControl (see page 12080)

Enabled (see page 310)

67.1.12 TOpenPicture Class

TOpenPicture is the standard action for displaying the open picture dialog.

Class Hierarchy

Pascal

```
TOpenPicture = class(TCommonDialogAction);
```

C++

```
class TOpenPicture : public TCommonDialogAction;
```

File

ExtActns

Description

Add TOpenPicture to an action list to add a graphics file selection dialog to your application. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to display the open picture dialog when invoked. Use the OnAccept (see page 11975) event to respond after the user selects a file name in the dialog or the OnCancel (see page 11975) event to respond when the user cancels from the dialog. In the event handler, you can read the file name from the dialog specified by the Dialog (see page 11971) property.

See Also

TFileOpen (see page 20605)

TOpenPictureDialog (see page 13282)

TSavePicture (see page 12048)

67.1.12.1 TOpenPicture Members

The following tables list the members exposed by TOpenPicture.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDialog (see page 20539)	This is FDialog, a member of class TCommonDialogAction.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAccept (see page 20540)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 20540)	This is DoCancel, a member of class TCommonDialogAction.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDialogClass (see page 11970)	This is GetDialogClass, a member of class TOpenPicture.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20541)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.

	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetupDialog (see page 20542)	This is SetupDialog, a member of class TCommonDialogAction.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	BeforeExecute (see page 20543)	Occurs immediately before the dialog appears.
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 20538)	Creates an instance of TCommonDialogAction.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20538)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
V	ExecuteTarget (see page 20540)	Executes the associated dialog and generates an OnAccept (see page 20543) or OnCancel (see page 20543) event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 20541)	Indicates whether the action can execute its dialog.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.

V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
R	ExecuteResult (see page 20542)	Indicates the result of executing the associated dialog.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnAccept (see page 11975)	This is OnAccept, a member of class TOpenPicture.
	OnCancel (see page 11975)	This is OnCancel, a member of class TOpenPicture.
	OnHint (see page 11975)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11976)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 11970)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
R	Dialog (see page 11971)	Provides access the open picture dialog that the action executes.
	Enabled (see page 11971)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 11971)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11972)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11972)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11973)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11973)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 11974)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 11974)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 11974)	Specifies the Visible property for client controls and menu items.

67.1.12.2 TOpenPicture Methods

The methods of the TOpenPicture class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 11970)	This is GetDialogClass, a member of class TOpenPicture.

67.1.12.2.1 TOpenPicture.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TOpenPicture.

67.1.12.3 TOpenPicture Properties

The properties of the TOpenPicture class are listed here.

Published Properties

	Name	Description
	Caption (see page 11970)	Represents the caption of client controls and menu items.
	Dialog (see page 11971)	Provides access the open picture dialog that the action executes.
	Enabled (see page 11971)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 11971)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11972)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11972)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11973)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11973)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 11974)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 11974)	Specifies the ShortCut property for client menu items.
	Visible (see page 11974)	Specifies the Visible property for client controls and menu items.

67.1.12.3.1 TOpenPicture.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

67.1.12.3.2 TOpenPicture.Dialog Property

Provides access the open picture dialog that the action executes.

Pascal

```
property Dialog: TOpenPictureDialog;
```

C++

```
__property TOpenPictureDialog Dialog;
```

Description

Use Dialog to configure the graphics file selection dialog before the action fires or to read the selected file name after the action executes the dialog. For example, you can set the dialog's default extension or initially selected file at design time using the Object Inspector or read the dialog's FileName property from an OnAccept (see page 11975) event handler to use the file that the user selected.

When using an open picture action, you do not place a TOpenPicture (see page 11965) Dialog component on the form or data module at design time. Instead, you add the TOpenPicture (see page 11965) object to your action list and then use the Dialog property. The dialog component is created by the TOpenPicture (see page 11965) constructor.

See Also

onaccept (see page 20543)

oncancel (see page 20543)

FileName (see page 11424)

67.1.12.3.3 TOpenPicture.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

67.1.12.3.4 TOpenPicture.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

67.1.12.3.5 TOpenPicture.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

67.1.12.3.6 TOpenPicture.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

67.1.12.3.7 TOpenPicture.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

67.1.12.3.8 TOpenPicture.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

67.1.12.3.9 TOpenPicture.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

67.1.12.3.10 TOpenPicture.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

67.1.12.3.11 TOpenPicture.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

67.1.12.4 TOpenPicture Events

The events of the TOpenPicture class are listed here.

Published Events

	Name	Description
	OnAccept (see page 11975)	This is OnAccept, a member of class TOpenPicture.
	OnCancel (see page 11975)	This is OnCancel, a member of class TOpenPicture.
	OnHint (see page 11975)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11976)	Occurs when the application is idle or when the action list updates.

67.1.12.4.1 TOpenPicture.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TOpenPicture.

67.1.12.4.2 TOpenPicture.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TOpenPicture.

67.1.12.4.3 TOpenPicture.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some

application-specific way.

67.1.12.4.4 TOpenPicture.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
```

```
  to Action1.
  }
  procedure TMainForm.Action1Update(Sender: TObject);
  begin
    if ((Sender as TAction).Checked <> ToolBar1.Visible) then
      lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
      { Indicate whether ToolBar1 is currently visible }
      (Sender as TAction).Checked := ToolBar1.Visible;
  end;
```

See Also

UpdateAction (see page 2790)

67.1.13 TPreviousTab Class

TPreviousTab moves a page control to its previous page.

Class Hierarchy

Pascal

```
TPreviousTab = class(TTabAction);
```

C++

```
class TPreviousTab : public TTabAction;
```

File

ExtActns

Description

Add TPreviousTab to an action list to allow users to move to the prior page of a page control using menu items or toolbar buttons. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application change the current page of a page control to the previous one. Typically, these client controls are the "Prior" button or menu item in a tabbed dialog.

Before you can use TPreviousTab, you must assign a target tab control as the value of the TabControl (see page 11987) property. Once the target tab control is assigned, TPreviousTab automatically disables itself if the target control is not enabled or if it is on the first page and wrapping is not permitted.

See Also

TNextTab (see page 11949)

TPageControl (see page 4459)

TTabControl (see page 4841)

67.1.13.1 TPreviousTab Members

The following tables list the members exposed by TPreviousTab.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 12079)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SelectNextTab (see page 12079)	This is SelectNextTab, a member of class TTabAction.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.

	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 12078)	Creates an instance of TTabAction (see page 12073).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 11982)	Changes the selected page in the target control to the previous one.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 12078)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
	Update (see page 295)	Indicates whether the action updates itself.
	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateTarget (see page 11982)	Enables or disables the action, depending on the status of the target control.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterTabChange (see page 11989)	Occurs immediately after the action changes the page of the associated tab control.
	BeforeTabChange (see page 11989)	Occurs immediately before the action changes the page of the associated tab control.
	OnHint (see page 11989)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11990)	Occurs when the application is idle or when the action list updates.
	OnValidateTab (see page 11991)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 11983)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Enabled (see page 11984)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 11984)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11985)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11985)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11986)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11986)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ShortCut (see page 11986)	Specifies the ShortCut property for client menu items.
	SkipHiddenTabs (see page 11987)	Indicates whether the tab action should consider hidden tabbed pages when moving to the next page.
	TabControl (see page 11987)	Specifies the target of the action.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 11987)	Specifies the Visible property for client controls and menu items.
	Wrap (see page 11988)	Indicates whether the tab action should "wrap" when navigating to the next tab page.

67.1.13.2 TPreviousTab Methods

The methods of the TPreviousTab class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 11982)	Changes the selected page in the target control to the previous one.
	UpdateTarget (see page 11982)	Enables or disables the action, depending on the status of the target control.

67.1.13.2.1 TPreviousTab.ExecuteTarget Method

Changes the selected page in the target control to the previous one.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

ExecuteTarget causes the action to perform its default response based on the state of the target tab control.

Target is the currently active object when the action fires. TPreviousTab (see page 11977) ignores this parameter and uses the TabControl (see page 11987) property to identify the target control.

ExecuteTarget

1. Generates a BeforeTabChange (see page 11989) event.
2. Moves the control identified by TabControl (see page 11987) to the previous page. If TabControl (see page 11987) is on the first page and the Wrap (see page 11988) property is true, this moves to the last page. If Wrap (see page 11988) is false, ExecuteTarget does not change the page when the tab control is on the first page. Note that the SkipHiddenTabs (see page 11987) property controls whether hidden pages are considered when locating the previous page.
3. Generates an AfterTabChange (see page 11989) event.

See Also

ExecuteTarget (see page 2712)

ActivePageIndex (see page 4484)

TabControl (see page 12080)

BeforeTabChange (see page 12082)

AfterTabChange (see page 12082)

Wrap (see page 12081)

TabIndex (see page 3946)

SkipHiddenTabs (see page 12080)

67.1.13.2.2 TPreviousTab.UpdateTarget Method

Enables or disables the action, depending on the status of the target control.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

Target is the currently active object.

In TPreviousTab (see page 11977), UpdateTarget ignores the Target parameter and checks whether the target control specified by the TabControl (see page 11987) property is enabled. If so, it sets Enabled (see page 11984) to true if the first page is not currently selected or if Wrap (see page 11988) is true. It sets Enabled (see page 11984) to false if TabControl (see page 11987) is not enabled or if the first page is selected and Wrap (see page 11988) is false.

See Also

Enabled (see page 310)

TabControl (see page 12080)

Wrap (see page 12081)

67.1.13.3 TPreviousTab Properties

The properties of the TPreviousTab class are listed here.

Published Properties

	Name	Description
	Caption (see page 11983)	Represents the caption of client controls and menu items.
	Enabled (see page 11984)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 11984)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 11985)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 11985)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 11986)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 11986)	Indicates the ImageIndex property for client controls and menu items.
	ShortCut (see page 11986)	Specifies the ShortCut property for client menu items.
	SkipHiddenTabs (see page 11987)	Indicates whether the tab action should consider hidden tabbed pages when moving to the next page.
	TabControl (see page 11987)	Specifies the target of the action.
	Visible (see page 11987)	Specifies the Visible property for client controls and menu items.
	Wrap (see page 11988)	Indicates whether the tab action should "wrap" when navigating to the next tab page.

67.1.13.3.1 TPreviousTab.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

67.1.13.3.2 TPreviousTab.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

67.1.13.3.3 TPreviousTab.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

67.1.13.3.4 TPreviousTab.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

67.1.13.3.5 TPreviousTab.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

67.1.13.3.6 TPreviousTab.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

67.1.13.3.7 TPreviousTab.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

67.1.13.3.8 TPreviousTab.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

67.1.13.3.9 TPreviousTab.SkipHiddenTabs Property

Indicates whether the tab action should consider hidden tabbed pages when moving to the next page.

Pascal

```
property SkipHiddenTabs: Boolean;
```

C++

```
__property Boolean SkipHiddenTabs;
```

Description

Set SkipHiddenTabs to true if you do not want the action to move to a tab page if its TabVisible property is false. Set SkipHiddenTabs to false if the tab action can move to and make visible a hidden page.

Note: SkipHiddenTabs only has an effect when the TabControl property identifies a TPageControl object. (TTabControl does not support hidden pages).

See Also

ExecuteTarget (see page 11954)

ExecuteTarget (see page 11982)

TabVisible (see page 4956)

TPageControl (see page 4459)

67.1.13.3.10 TPreviousTab.TabControl Property

Specifies the target of the action.

Pascal

```
property TabControl: TCustomTabControl;
```

C++

```
__property TCustomTabControl TabControl;
```

Description

Set TabControl to the control on which the action operates when its ExecuteTarget method is called. Typically, this is a page control or tab control object that implements the pages of a tabbed dialog.

An application must explicitly assign a value to the TabControl property before the TTabAction descendant is enabled. The action does not infer the target page control when it fires.

See Also

ExecuteTarget (see page 11954)

ExecuteTarget (see page 11982)

67.1.13.3.11 TPreviousTab.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

67.1.13.3.12 TPreviousTab.Wrap Property

Indicates whether the tab action should "wrap" when navigating to the next tab page.

Pascal

```
property Wrap: Boolean;
```

C++

```
__property Boolean Wrap;
```

Description

When Wrap is true, the tab action can move previous to the first tabbed page or after the last tabbed page in the associated control. When Wrap is false, an attempt to move beyond the first or last page does not result in any page change, although it still results in a BeforeTabChange and an AfterTabChange event or in an OnFinish event.

See Also

ExecuteTarget (see page 11954)

ExecuteTarget (see page 11982)

TabControl (see page 12080)

BeforeTabChange (see page 12082)

AfterTabChange (see page 12082)

OnFinish (see page 11962)

67.1.13.4 TPreviousTab Events

The events of the TPreviousTab class are listed here.

Published Events

	Name	Description
	AfterTabChange (see page 11989)	Occurs immediately after the action changes the page of the associated tab control.
	BeforeTabChange (see page 11989)	Occurs immediately before the action changes the page of the associated tab control.
	OnHint (see page 11989)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 11990)	Occurs when the application is idle or when the action list updates.
	OnValidateTab (see page 11991)	Occurs when the application is idle or when the action list updates.

67.1.13.4.1 TPreviousTab.AfterTabChange Event

Occurs immediately after the action changes the page of the associated tab control.

Pascal

```
property AfterTabChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterTabChange;
```

Description

Write an AfterTabChange event handler to perform any additional tasks after the tab action changes the current page of the associated tab control. Typically, this event is used to initialize the current page of the tab control.

Note: AfterTabChange only occurs if the action tries to change the page of the tab control (even if it is not successful). If the action attempts some other task (for example, when TNextPage is on the last page of the tab control), AfterTabChange does not occur.

See Also

BeforeTabChange (see page 12082)

TabControl (see page 12080)

Wrap (see page 12081)

67.1.13.4.2 TPreviousTab.BeforeTabChange Event

Occurs immediately before the action changes the page of the associated tab control.

Pascal

```
property BeforeTabChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeTabChange;
```

Description

Write a BeforeTabChange event handler to perform any additional tasks before the tab action changes the current page of the associated tab control. Typically, this event is used to save or respond to the changes the user made on the current page before it changes.

Note: BeforeTabChange only occurs if the action tries to change the page of the tab control (even if it is not successful). If the action attempts some other task (for example, when TNextPage is on the last page of the tab control), BeforeTabChange does not occur.

See Also

AfterTabChange (see page 12082)

TabControl (see page 12080)

Wrap (see page 12081)

67.1.13.4.3 TPreviousTab.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

67.1.13.4.4 TPreviousTab.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
```

```
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

67.1.13.4.5 TPreviousTab.OnValidateTab Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnValidateTab: TValidateTabEvent;
```

C++

```
__property TValidateTabEvent OnValidateTab;
```

Description

Use the OnValidateTab event rather than the OnUpdate event to update the properties of a tab action when the application is idle. Unlike the OnUpdate event, which prevents the default behavior of the UpdateTarget method, OnValidateTab occurs after UpdateTarget has set the Enabled property to reflect whether the target tab control is enabled.

OnValidateTab can be used to update any of the properties of the tab action, but it is intended primarily for enabling or disabling the action, depending on the contents of the current page of the target tab control.

Sender is the tab action component to update in this event.

TabControl is the target tab control.

Enabled reflects the Enabled property of this action. Set Enabled in the event handler to enable or disable the action depending on current conditions.

See Also

OnUpdate (see page 2718)

UpdateTarget (see page 12079)

TabControl (see page 12080)

Enabled (see page 310)

67.1.14 TRichEditAction Class

TRichEditAction is the base class for action objects that apply rich text attributes to a rich edit control.

Class Hierarchy

Pascal

```
TRichEditAction = class(TEditAction);
```

C++

```
class TRichEditAction : public TEditAction;
```

File

ExtActns

Description

TRichEditAction implements the common behavior for actions that assign text attributes to a rich edit control. Each descendant class represents a specific text attribute. Do not use TRichEditAction directly in an application. Instead, use a TRichEditAction descendant such as TRichEditBold (see page 12017), TRichEditItalic (see page 12029), TRichEditUnderline (see page 12042), TRichEditStrikeOut (see page 12036), TRichEditAlignCenter (see page 11998), TRichEditAlignLeft (see page 12005), TRichEditAlignRight (see page 12011), or TRichEditBullets (see page 12023).

TRichEditAction ensures that the action is only works with a TCustomRichEdit descendant as a target. TRichEditAction descendants toggle their specific text attribute in the target rich edit control: if the attribute is set, it is removed, and if it is not set (or set for only some of the text), the action sets it. If the target rich edit control has a selection, the attribute is toggled for the selected text only. If there is no selection, the attribute is toggled for all of the text.

See Also

TCustomRichEdit (see page 3854)

TRichEditBold (see page 12017)

TRichEditItalic (see page 12029)

TRichEditUnderline (see page 12042)

TRichEditAlignCenter (see page 11998)

TRichEditAlignLeft (see page 12005)

TRichEditAlignRight (see page 12011)

TRichEditBullets (see page 12023)

TRichEditStrikeOut (see page 12036)

67.1.14.1 TRichEditAction Members

The following tables list the members exposed by TRichEditAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CurrText (see page 11997)	This is CurrText, a member of class TRichEditAction.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetControl (see page 20549)	Returns the target of the edit action.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20550)	Responds when objects are added or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFontStyle (see page 11998)	This is SetFontStyle, a member of class TRichEditAction.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 11997)	Creates an instance of TRichEditAction (see page 11992).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 20549)	Frees an instance of TCustomAction.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 11998)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.
V	UpdateTarget (see page 20551)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Control (see page 20551)	Provides access to the target of the action.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.

	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

67.1.14.2 TRichEditAction.Create Constructor

Creates an instance of TRichEditAction (see page 11992).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TRichEditAction(TComponent * AOwner);
```

Description

Most applications do not need to call the TRichEditAction (see page 11992) constructor because actions are added at design time to an action list or action manager.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TRichEditAction (see page 11992) instance.

See Also

Owner (see page 2807)

67.1.14.3 TRichEditAction Methods

The methods of the TRichEditAction class are listed here.

Protected Methods

	Name	Description
	CurrText (see page 11997)	This is CurrText, a member of class TRichEditAction.
	SetFontStyle (see page 11998)	This is SetFontStyle, a member of class TRichEditAction.

Public Methods

	Name	Description
	HandlesTarget (see page 11998)	Indicates whether the action can execute its function.

67.1.14.3.1 TRichEditAction.CurrText Method

Pascal

```
function CurrText(Edit: TCustomRichEdit): TTextAttributes;
```

C++

```
__fastcall TTextAttributes CurrText(TCustomRichEdit Edit);
```

Description

This is CurrText, a member of class TRichEditAction.

67.1.14.3.2 TRichEditAction.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the currently active object.

HandlesTarget returns true if Target is a TCustomRichEdit descendant, false otherwise.

See Also

ExecuteTarget (see page 2712)

TCustomRichEdit (see page 3854)

67.1.14.3.3 TRichEditAction.SetFontStyle Method

Pascal

```
procedure SetFontStyle(Edit: TCustomRichEdit; Style: TFontStyle);
```

C++

```
__fastcall SetFontStyle(TCustomRichEdit Edit, TFontStyle Style);
```

Description

This is SetFontStyle, a member of class TRichEditAction.

67.1.15 TRichEditAlignCenter Class

TRichEditAlignCenter centers the text horizontally in the current paragraph of a rich edit control.

Class Hierarchy

Pascal

```
TRichEditAlignCenter = class(TRichEditAction);
```

C++

```
class TRichEditAlignCenter : public TRichEditAction;
```

File

ExtActns

Description

Add TRichEditAlignCenter to an action list to let users center the text in a rich edit control. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to center the text in all paragraphs that contain selected text (or the paragraph with the cursor if no text is selected).

TRichEditAlignCenter is designed to work with a rich edit control (TCustomRichEdit descendant) as a target. It automatically disables itself if the active control in a form is not a rich edit control, and adds or removes a check mark to reflect whether the text in the target edit control is centered.

See Also

TCustomRichEdit (see page 3854)

TRichEditItalic (see page 12029)

TRichEditUnderline (see page 12042)

TRichEditStrikeOut (see page 12036)

TRichEditAlignRight (see page 12011)

TRichEditAlignLeft (see page 12005)

TRichEditBullets (see page 12023)

TRichEditBold (see page 12017)

67.1.15.1 TRichEditAlignCenter Members

The following tables list the members exposed by TRichEditAlignCenter.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.

	CurrText (see page 11997)	This is CurrText, a member of class TRichEditAction.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetControl (see page 20549)	Returns the target of the edit action.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 20550)	Responds when objects are added or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFontStyle (see page 11998)	This is SetFontStyle, a member of class TRichEditAction.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 11997)	Creates an instance of TRichEditAction (see page 11992).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 20549)	Frees an instance of TCustomAction.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 12004)	Centers the text in the current paragraph of the target rich edit control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 11998)	Indicates whether the action can execute its function.
	HandlesTarget (see page 20550)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
	Update (see page 295)	Indicates whether the action updates itself.
	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateTarget (see page 12004)	Configures the action, depending on current conditions.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
	ComObject (see page 2793)	Specifies the interface reference implemented by the component.

R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Control (see page 20551)	Provides access to the target of the action.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

67.1.15.2 TRichEditAlignCenter Methods

The methods of the TRichEditAlignCenter class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 12004)	Centers the text in the current paragraph of the target rich edit control.
	UpdateTarget (see page 12004)	Configures the action, depending on current conditions.

67.1.15.2.1 TRichEditAlignCenter.ExecuteTarget Method

Centers the text in the current paragraph of the target rich edit control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is a rich edit control, and if so, sets its Alignment property to taCenter.

See Also

Paragraph (see page 3884)

Alignment (see page 4609)

67.1.15.2.2 TRichEditAlignCenter.UpdateTarget Method

Configures the action, depending on current conditions.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TRichEditAlignCenter (see page 11998), UpdateTarget sets Enabled to true and sets the Checked property to true if the text in the current paragraph is centered.

See Also

Enabled (see page 310)

Checked (see page 308)

67.1.16 TRichEditAlignLeft Class

TRichEditAlignLeft left-justifies the text in the current paragraph of a rich edit control.

Class Hierarchy

Pascal

```
TRichEditAlignLeft = class(TRichEditAction);
```

C++

```
class TRichEditAlignLeft : public TRichEditAction;
```

File

ExtActns

Description

Add TRichEditAlignLeft to an action list to let users left-justify the text in a rich edit control. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to align the text in all paragraphs that contain selected text (or the paragraph with the cursor if no text is selected) to the left margin.

TRichEditAlignLeft is designed to work with a rich edit control (TCustomRichEdit descendant) as a target. It automatically disables itself if the active control in a form is not a rich edit control, and adds or removes a check mark to indicate whether the text in the target control is left-aligned.

See Also

TCustomRichEdit (see page 3854)

TRichEditItalic (see page 12029)

TRichEditUnderline (see page 12042)

TRichEditStrikeOut (see page 12036)

TRichEditAlignRight (see page 12011)

TRichEditAlignCenter (see page 11998)

TRichEditBullets (see page 12023)

TRichEditBold (see page 12017)

67.1.16.1 TRichEditAlignLeft Members

The following tables list the members exposed by TRichEditAlignLeft.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CurrText (see page 11997)	This is CurrText, a member of class TRichEditAction.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetControl (see page 20549)	Returns the target of the edit action.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20550)	Responds when objects are added or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFontStyle (see page 11998)	This is SetFontStyle, a member of class TRichEditAction.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.

	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 11997)	Creates an instance of TRichEditAction (see page 11992).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20549)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 12010)	Left-justifies the text in the current paragraph of the target rich edit control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 11998)	Indicates whether the action can execute its function.
	HandlesTarget (see page 20550)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
	Update (see page 295)	Indicates whether the action updates itself.
	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 12010)	Configures the action, depending on current conditions.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Control (see page 20551)	Provides access to the target of the action.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.

	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

67.1.16.2 TRichEditAlignLeft Methods

The methods of the TRichEditAlignLeft class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 12010)	Left-justifies the text in the current paragraph of the target rich edit control.
	UpdateTarget (see page 12010)	Configures the action, depending on current conditions.

67.1.16.2.1 TRichEditAlignLeft.ExecuteTarget Method

Left-justifies the text in the current paragraph of the target rich edit control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is a rich edit control, and if so, sets its Alignment property to taLeftJustify.

See Also

Paragraph (see page 3884)

Alignment (see page 4609)

67.1.16.2.2 TRichEditAlignLeft.UpdateTarget Method

Configures the action, depending on current conditions.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update

themselves according to current conditions.

In TRichEditAlignLeft (see page 12005), UpdateTarget sets Enabled to true and sets the Checked property to true if the text in the current paragraph is left justified.

See Also

Enabled (see page 310)

Checked (see page 308)

67.1.17 TRichEditAlignRight Class

TRichEditAlignRight right-justifies the text in the current paragraph of a rich edit control.

Class Hierarchy

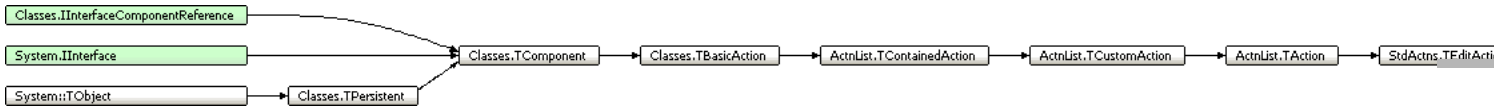

Pascal

```
TRichEditAlignRight = class(TRichEditAction);
```

C++

```
class TRichEditAlignRight : public TRichEditAction;
```

File

ExtActns

Description

Add TRichEditAlignRight to an action list to let users right-justify the text in a rich edit control. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to align the text in all paragraphs that contain selected text (or the paragraph with the cursor if no text is selected) to the right margin.

TRichEditAlignRight is designed to work with a rich edit control (TCustomRichEdit descendant) as a target. It automatically disables itself if the active control in a form is not a rich edit control, and adds or removes a check mark to indicate whether the current paragraph of the target control is right-aligned.

See Also

TCustomRichEdit (see page 3854)

TRichEditItalic (see page 12029)

TRichEditUnderline (see page 12042)

TRichEditStrikeOut (see page 12036)

TRichEditAlignLeft (see page 12005)

TRichEditAlignCenter (see page 11998)

TRichEditBullets (see page 12023)

TRichEditBold (see page 12017)

67.1.17.1 TRichEditAlignRight Members

The following tables list the members exposed by TRichEditAlignRight.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CurrText (see page 11997)	This is CurrText, a member of class TRichEditAction.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetControl (see page 20549)	Returns the target of the edit action.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20550)	Responds when objects are added or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFontStyle (see page 11998)	This is SetFontStyle, a member of class TRichEditAction.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 11997)	Creates an instance of TRichEditAction (see page 11992).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.

	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20549)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 12016)	Right-justifies the text in the current paragraph of the target rich edit control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 11998)	Indicates whether the action can execute its function.
	HandlesTarget (see page 20550)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 12017)	Configures the action, depending on current conditions.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Control (see page 20551)	Provides access to the target of the action.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

67.1.17.2 TRichEditAlignRight Methods

The methods of the TRichEditAlignRight class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 12016)	Right-justifies the text in the current paragraph of the target rich edit control.
	UpdateTarget (see page 12017)	Configures the action, depending on current conditions.

67.1.17.2.1 TRichEditAlignRight.ExecuteTarget Method

Right-justifies the text in the current paragraph of the target rich edit control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is a rich edit control, and if so, sets its Alignment property to taRightJustify.

See Also

Paragraph (see page 3884)

Alignment (see page 4609)

67.1.17.2.2 TRichEditAlignRight.UpdateTarget Method

Configures the action, depending on current conditions.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TRichEditAlignRight (see page 12011), UpdateTarget sets Enabled to true and sets the Checked property to true if the text in the current paragraph is right justified.

See Also

Enabled (see page 310)

Checked (see page 308)

67.1.18 TRichEditBold Class

TRichEditBold toggles the bold attribute of the currently selected text in a rich edit control.

Class Hierarchy

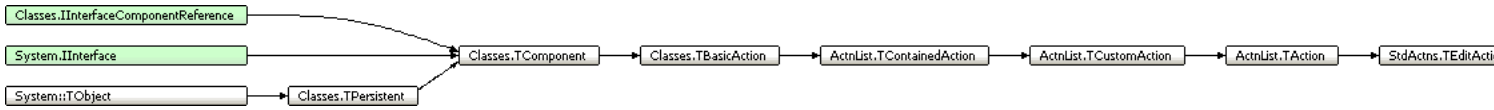

Pascal

```
TRichEditBold = class(TRichEditAction);
```

C++

```
class TRichEditBold : public TRichEditAction;
```

File

ExtActns

Description

Add TRichEditBold to an action list to let users control whether the selected text in a rich edit control is bold. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to toggle the bold attribute of selected text (or of all text if there is no text selected).

TRichEditBold is designed to work with a rich edit control (TCustomRichEdit descendant) as a target. It automatically disables itself if the active control in a form is not a rich edit control, and adds or removes a check mark to indicate whether the target control's text is bold.

See Also

TCustomRichEdit (see page 3854)

TRichEditItalic (see page 12029)

TRichEditUnderline (see page 12042)

TRichEditBullets (see page 12023)

TRichEditAlignRight (see page 12011)

TRichEditAlignLeft (see page 12005)

TRichEditAlignCenter (see page 11998)

TRichEditStrikeOut (see page 12036)

67.1.18.1 TRichEditBold Members

The following tables list the members exposed by TRichEditBold.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CurrText (see page 11997)	This is CurrText, a member of class TRichEditAction.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetControl (see page 20549)	Returns the target of the edit action.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.

	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 20550)	Responds when objects are added or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFontStyle (see page 11998)	This is SetFontStyle, a member of class TRichEditAction.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 11997)	Creates an instance of TRichEditAction (see page 11992).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20549)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 12022)	Toggles the fsBold style of the target rich edit control's text attributes.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 11998)	Indicates whether the action can execute its function.
	HandlesTarget (see page 20550)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 12023)	Sets the Checked property to reflect whether the target control's text is bold.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Control (see page 20551)	Provides access to the target of the action.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.

	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

67.1.18.2 TRichEditBold Methods

The methods of the TRichEditBold class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 12022)	Toggles the fsBold style of the target rich edit control's text attributes.
	UpdateTarget (see page 12023)	Sets the Checked property to reflect whether the target control's text is bold.

67.1.18.2.1 TRichEditBold.ExecuteTarget Method

Toggles the fsBold style of the target rich edit control's text attributes.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

If the target rich edit control has a selection, ExecuteTarget toggles the fsBold style of the selected text attributes (SelAttributes property), adding the fsBold property if the selected text is not all bold and removing it if the selected text is entirely bold. Note that if the selected text contains both bold and non-bold text, this results in the text becoming entirely bold.

If the target rich edit control has no selection, ExecuteTarget toggles the default fsBold style of the control. This changes the bold attribute of all text that has not had a specific text attribute applied.

See Also

Style (see page 4992)

SelAttributes (see page 3887)

DefAttributes (see page 3877)

67.1.18.2.2 TRichEditBold.UpdateTarget Method

Sets the Checked property to reflect whether the target control's text is bold.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

Target is the currently active object when the action fires.

UpdateTarget sets the Checked property to true if Target is a rich edit control whose selected text is bold, otherwise it sets Checked to false.

See Also

Style (see page 4992)

SelAttributes (see page 3887)

Checked (see page 308)

67.1.19 TRichEditBullets Class

TRichEditBullets toggles whether the current paragraph of a rich edit control is bulleted.

Class Hierarchy

Pascal

```
TRichEditBullets = class(TRichEditAction);
```

C++

```
class TRichEditBullets : public TRichEditAction;
```

File

ExtActns

Description

Add TRichEditBullets to an action list to let users control whether the current paragraphs in a rich edit control are bulleted. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to toggle the bulleted attribute of all paragraphs that contain selected text (or the paragraph with the cursor if no text is selected).

TRichEditBullets is designed to work with a rich edit control (TCustomRichEdit descendant) as a target. It automatically disables itself if the active control in a form is not a rich edit control and adds or removes a check mark to indicate whether that control's paragraphs are bulleted.

See Also

TCustomRichEdit (see page 3854)

TRichEditItalic (see page 12029)

TRichEditUnderline (see page 12042)

TRichEditStrikeOut (see page 12036)

TRichEditAlignRight (see page 12011)

TRichEditAlignLeft (see page 12005)

TRichEditAlignCenter (see page 11998)

TRichEditBold (see page 12017)

67.1.19.1 TRichEditBullets Members

The following tables list the members exposed by TRichEditBullets.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CurrText (see page 11997)	This is CurrText, a member of class TRichEditAction.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetControl (see page 20549)	Returns the target of the edit action.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 20550)	Responds when objects are added or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFontStyle (see page 11998)	This is SetFontStyle, a member of class TRichEditAction.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 11997)	Creates an instance of TRichEditAction (see page 11992).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 20549)	Frees an instance of TCustomAction.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 12029)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 11998)	Indicates whether the action can execute its function.
V	HandlesTarget (see page 20550)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 12029)	Configures the action, depending on current conditions.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Control (see page 20551)	Provides access to the target of the action.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

67.1.19.2 TRichEditBullets Methods

The methods of the TRichEditBullets class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 12029)	Introduces an interface for invoking an action on a target client component or control.
	UpdateTarget (see page 12029)	Configures the action, depending on current conditions.

67.1.19.2.1 TRichEditBullets.ExecuteTarget Method

Introduces an interface for invoking an action on a target client component or control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

ExecuteTarget does nothing in TBasicAction. ExecuteTarget was introduced in TBasicAction so that descendants can override it to initiate the action on the target. For example, an edit action that performs copying might copy the contents of an edit control to the clipboard.

See Also

OnExecute (see page 2716)

HandlesTarget (see page 2713)

67.1.19.2.2 TRichEditBullets.UpdateTarget Method

Configures the action, depending on current conditions.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In UpdateTarget, UpdateTarget sets Enabled to T true and sets the Checked property to T true if the current paragraph is bulleted or to F false if the current paragraph is not bulleted.

See Also

Enabled (see page 310)

Checked (see page 308)

67.1.20 TRichEditItalic Class

TRichEditItalic toggles the italic attribute of the currently selected text in a rich edit control.

Class Hierarchy

Pascal

```
TRichEditItalic = class(TRichEditAction);
```

C++

```
class TRichEditItalic : public TRichEditAction;
```

File

ExtActns

Description

Add TRichEditItalic to an action list to let users control whether the selected text in a rich edit control is italic. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to toggle the italic attribute of selected text (or of all text if there is no text selected).

TRichEditItalic is designed to work with a rich edit control (TCustomRichEdit descendant) as a target. It automatically disables itself if the active control in a form is not a rich edit control, and adds or removes a check mark to reflect whether the target control's text is italic.

See Also

TCustomRichEdit (see page 3854)

TRichEditBold (see page 12017)

TRichEditUnderline (see page 12042)

TRichEditBullets (see page 12023)

TRichEditAlignRight (see page 12011)

TRichEditAlignLeft (see page 12005)

TRichEditAlignCenter (see page 11998)

TRichEditStrikeout

67.1.20.1 TRichEditItalic Members

The following tables list the members exposed by TRichEditItalic.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CurrText (see page 11997)	This is CurrText, a member of class TRichEditAction.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetControl (see page 20549)	Returns the target of the edit action.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20550)	Responds when objects are added or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFontStyle (see page 11998)	This is SetFontStyle, a member of class TRichEditAction.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 11997)	Creates an instance of TRichEditAction (see page 11992).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 20549)	Frees an instance of TCustomAction.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".

	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 12035)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 11998)	Indicates whether the action can execute its function.
V	HandlesTarget (see page 20550)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 12035)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Control (see page 20551)	Provides access to the target of the action.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.

	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

67.1.20.2 TRichEditItalic Methods

The methods of the TRichEditItalic class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 12035)	Introduces an interface for invoking an action on a target client component or control.
	UpdateTarget (see page 12035)	Enables or disables the action, depending on the currently active control in the form.

67.1.20.2.1 TRichEditItalic.ExecuteTarget Method

Introduces an interface for invoking an action on a target client component or control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

ExecuteTarget does nothing in TBasicAction. ExecuteTarget was introduced in TBasicAction so that descendants can override it to initiate the action on the target. For example, an edit action that performs copying might copy the contents of an edit control to the clipboard.

See Also

OnExecute (see page 2716)

HandlesTarget (see page 2713)

67.1.20.2.2 TRichEditItalic.UpdateTarget Method

Enables or disables the action, depending on the currently active control in the form.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the potential target changes so that actions can update themselves accordingly.

In TEditAction, UpdateTarget checks whether the current target object (specified by the Target parameter) is an edit control and,

if the action's function requires it, that some of its text is selected. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

67.1.21 TRichEditStrikeOut Class

TRichEditStrikeOut toggles the strikeout attribute of the currently selected text in a rich edit control.

Class Hierarchy

Pascal

```
TRichEditStrikeOut = class(TRichEditAction);
```

C++

```
class TRichEditStrikeOut : public TRichEditAction;
```

File

ExtActns

Description

Add TRichEditStrikeOut to an action list to let users control whether the selected text in a rich edit control has a line through it. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to toggle the strikeout attribute of selected text (or of all text if there is no text selected).

TRichEditStrikeOut is designed to work with a rich edit control (TCustomRichEdit descendant) as a target. It automatically disables itself if the active control in a form is not a rich edit control and adds or removes a check mark to indicate whether the target control's text has the strike out attribute.

See Also

TCustomRichEdit (see page 3854)

TRichEditItalic (see page 12029)

TRichEditUnderline (see page 12042)

TRichEditBullets (see page 12023)

TRichEditAlignRight (see page 12011)

TRichEditAlignLeft (see page 12005)

TRichEditAlignCenter (see page 11998)

67.1.21.1 TRichEditStrikeOut Members

The following tables list the members exposed by TRichEditStrikeOut.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CurrText (see page 11997)	This is CurrText, a member of class TRichEditAction.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetControl (see page 20549)	Returns the target of the edit action.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20550)	Responds when objects are added or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFontStyle (see page 11998)	This is SetFontStyle, a member of class TRichEditAction.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 11997)	Creates an instance of TRichEditAction (see page 11992).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.

	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20549)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 12041)	Toggles the fsStrikeOut style of the target rich edit control's text attributes.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 11998)	Indicates whether the action can execute its function.
	HandlesTarget (see page 20550)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.

V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 12042)	Sets the Checked property to reflect whether the target control's text includes a strikeout line.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Control (see page 20551)	Provides access to the target of the action.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.

	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

67.1.21.2 TRichEditStrikeOut Methods

The methods of the TRichEditStrikeOut class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 12041)	Toggles the fsStrikeOut style of the target rich edit control's text attributes.
	UpdateTarget (see page 12042)	Sets the Checked property to reflect whether the target control's text includes a strikeout line.

67.1.21.2.1 TRichEditStrikeOut.ExecuteTarget Method

Toggles the fsStrikeOut style of the target rich edit control's text attributes.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

If the target rich edit control has a selection, ExecuteTarget toggles the fsStrikeOut style of the selected text attributes (SelAttributes property), adding the fsStrikeOut property if the selected text is not all struck out and removing it if the selected text is entirely struck out. Note that if the selected text contains both strikeout and non-strikeout text, this results in the text becoming entirely struck out.

If the target rich edit control has no selection, ExecuteTarget toggles the default fsStrikeOut style of the control. This changes the strikeout attribute of all text that has not had a specific text attribute applied.

See Also

Style (see page 4992)

SelAttributes (see page 3887)

DefAttributes (see page 3877)

67.1.21.2.2 TRichEditStrikeOut.UpdateTarget Method

Sets the Checked property to reflect whether the target control's text includes a strikeout line.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

Target is the currently active object when the action fires.

UpdateTarget checks whether Target is a rich edit control with the strikeout font attribute on its selected text. If the selected text is struck out, UpdateTarget sets the Checked property to true, otherwise it sets Checked to false.

See Also

Style (see page 4992)

SelAttributes (see page 3887)

Checked (see page 308)

67.1.22 TRichEditUnderline Class

TRichEditUnderline toggles the underline attribute of the currently selected text in a rich edit control.

Class Hierarchy

Pascal

```
TRichEditUnderline = class(TRichEditAction);
```

C++

```
class TRichEditUnderline : public TRichEditAction;
```

File

ExtActns

Description

Add TRichEditUnderline to an action list to let users control whether the selected text in a rich edit control is underlined. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to toggle the underline attribute of selected text (or of all text if there is no text selected).

TRichEditUnderline is designed to work with a rich edit control (TCustomRichEdit descendant) as a target. It automatically disables itself if the active control in a form is not a rich edit control, and adds or removes a check to indicate whether the target

control's text is underlined.

See Also

TCustomRichEdit (see page 3854)

TRichEditBold (see page 12017)

TRichEditUnderline

TRichEditStrikeOut (see page 12036)

TRichEditBullets (see page 12023)

TRichEditAlignRight (see page 12011)

TRichEditAlignLeft (see page 12005)

TRichEditAlignCenter (see page 11998)

TRichEditStrikeOut (see page 12036)

67.1.22.1 TRichEditUnderline Members

The following tables list the members exposed by TRichEditUnderline.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CurrText (see page 11997)	This is CurrText, a member of class TRichEditAction.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetControl (see page 20549)	Returns the target of the edit action.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.

	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 20550)	Responds when objects are added or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFontStyle (see page 11998)	This is SetFontStyle, a member of class TRichEditAction.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.

	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 11997)	Creates an instance of TRichEditAction (see page 11992).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20549)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 12047)	Toggles the fsUnderline style of the target rich edit control's text attributes.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.

V	HandlesTarget (see page 11998)	Indicates whether the action can execute its function.
V	HandlesTarget (see page 20550)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 12048)	Sets the Checked property to reflect whether the target control's text is underlined.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Control (see page 20551)	Provides access to the target of the action.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.

	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

67.1.22.2 TRichEditUnderline Methods

The methods of the TRichEditUnderline class are listed here.

Public Methods

	Name	Description
V	ExecuteTarget (see page 12047)	Toggles the fsUnderline style of the target rich edit control's text attributes.
V	UpdateTarget (see page 12048)	Sets the Checked property to reflect whether the target control's text is underlined.

67.1.22.2.1 TRichEditUnderline.ExecuteTarget Method

Toggles the fsUnderline style of the target rich edit control's text attributes.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

If the target rich edit control has a selection, ExecuteTarget toggles the fsUnderline style of the selected text attributes (SelAttributes property), adding the fsUnderline property if the selected text is not all underlined and removing it if the selected text is entirely underlined. Note that if the selected text contains both underlined and non-underlined text, this results in the text becoming entirely underlined.

If the target rich edit control has no selection, ExecuteTarget toggles the default fsUnderline style of the control. This changes the underlined attribute of all text that has not had a specific text attribute applied.

See Also

Style (see page 4992)

SelAttributes (see page 3887)

DefAttributes (see page 3877)

67.1.22.2.2 TRichEditUnderline.UpdateTarget Method

Sets the Checked property to reflect whether the target control's text is underlined.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

Target is the currently active object when the action fires.

UpdateTarget checks the selected text of the target rich edit control (which is specified by the Control property or the Target parameter) is underlined. If the selected text is underlined, UpdateTarget sets the Checked property to true, otherwise it sets Checked to false.

See Also

Style (see page 4992)

SelAttributes (see page 3887)

Checked (see page 308)

67.1.23 TSavePicture Class

TSavePicture is the standard action for displaying the save picture dialog.

Class Hierarchy

Pascal

```
TSavePicture = class(TCommonDialogAction);
```

C++

```
class TSavePicture : public TCommonDialogAction;
```

File

ExtActns

Description

Add TSavePicture to an action list to add a graphics "save as" dialog to your application. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to display the save picture dialog when invoked. Use the OnAccept (see page 12059) event to respond after the user selects a file name in the dialog or the OnCancel (see page 12059) event to respond when the user cancels from the dialog. In the event handler, you can read the file name from the dialog specified by the Dialog (see page 12054) property.

See Also

TFileSaveAs (see page 20654)

TSavePictureDialog (see page 13300)

TSavePicture

67.1.23.1 TSavePicture Members

The following tables list the members exposed by TSavePicture.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDialog (see page 20539)	This is FDialog, a member of class TCommonDialogAction.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.

	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoAccept (see page 20540)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 20540)	This is DoCancel, a member of class TCommonDialogAction.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDialogClass (see page 12053)	This is GetDialogClass, a member of class TSavePicture.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 20541)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetupDialog (see page 20542)	This is SetupDialog, a member of class TCommonDialogAction.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	BeforeExecute (see page 20543)	Occurs immediately before the dialog appears.
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 20538)	Creates an instance of TCommonDialogAction.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 20538)	Frees an instance of TCustomAction.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
V	ExecuteTarget (see page 20540)	Executes the associated dialog and generates an OnAccept (see page 20543) or OnCancel (see page 20543) event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 20541)	Indicates whether the action can execute its dialog.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
R	ExecuteResult (see page 20542)	Indicates the result of executing the associated dialog.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnAccept (see page 12059)	This is OnAccept, a member of class TSavePicture.
	OnCancel (see page 12059)	This is OnCancel, a member of class TSavePicture.
	OnHint (see page 12059)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 12059)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 12054)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
R	Dialog (see page 12054)	Provides access the save picture dialog that the action executes.
	Enabled (see page 12055)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 12055)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 12056)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 12056)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 12057)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 12057)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 12057)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 12058)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 12058)	Specifies the Visible property for client controls and menu items.

67.1.23.2 TSavePicture Methods

The methods of the TSavePicture class are listed here.

Protected Methods

	Name	Description
V	GetDialogClass (see page 12053)	This is GetDialogClass, a member of class TSavePicture.

67.1.23.2.1 TSavePicture.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TSavePicture.

67.1.23.3 TSavePicture Properties

The properties of the TSavePicture class are listed here.

Published Properties

	Name	Description
	Caption (see page 12054)	Represents the caption of client controls and menu items.
R	Dialog (see page 12054)	Provides access the save picture dialog that the action executes.
	Enabled (see page 12055)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 12055)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 12056)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 12056)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 12057)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 12057)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 12057)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 12058)	Specifies the ShortCut property for client menu items.
	Visible (see page 12058)	Specifies the Visible property for client controls and menu items.

67.1.23.3.1 TSavePicture.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

67.1.23.3.2 TSavePicture.Dialog Property

Provides access the save picture dialog that the action executes.

Pascal

```
property Dialog: TSavePictureDialog;
```

C++

```
__property TSavePictureDialog Dialog;
```

Description

Use Dialog to configure the graphics file "save as" dialog before the action fires or to read the specified file name after the action executes the dialog. For example, you can set the dialog's default extension or initial file name at design time using the Object Inspector or read the dialog's FileName property from an OnAccept (see page 12059) event handler to use the file name that the user selects.

When using a save picture action, you do not place a TSavePicture (see page 12048) Dialog component on the form or data module at design time. Instead, you add the TSavePicture (see page 12048) object to your action list and then use the Dialog property. The dialog component is created by the TSavePicture (see page 12048) constructor.

See Also

onaccept (see page 20543)

oncancel (see page 20543)

FileName (see page 11424)

67.1.23.3.3 TSavePicture.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

67.1.23.3.4 TSavePicture.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

67.1.23.3.5 TSavePicture.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

67.1.23.3.6 TSavePicture.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

67.1.23.3.7 TSavePicture.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

67.1.23.3.8 TSavePicture.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

67.1.23.3.9 TSavePicture.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key

combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

67.1.23.3.10 TSavePicture.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

67.1.23.3.11 TSavePicture.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

67.1.23.4 TSavePicture Events

The events of the TSavePicture class are listed here.

Published Events

	Name	Description
	OnAccept (see page 12059)	This is OnAccept, a member of class TSavePicture.
	OnCancel (see page 12059)	This is OnCancel, a member of class TSavePicture.

	OnHint (see page 12059)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 12059)	Occurs when the application is idle or when the action list updates.

67.1.23.4.1 TSavePicture.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TSavePicture.

67.1.23.4.2 TSavePicture.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TSavePicture.

67.1.23.4.3 TSavePicture.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

67.1.23.4.4 TSavePicture.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

67.1.24 TSendMail Class

TSendMail is the standard action for sending an email message.

Class Hierarchy

Pascal

```
TSendMail = class(TCustomAction);
```

C++

```
class TSendMail : public TCustomAction;
```

File

ExtActns

Description

Add TSendMail to an action list to let your application send email messages in response to user actions. Controls (see page 5711) linked to this action cause the application to send a MAPI mail message containing the message specified in the Text (see page 12071) property.

See Also

TFileRun (see page 11872)

TSendMail

67.1.24.1 TSendMail Members

The following tables list the members exposed by TSendMail.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.

	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 12065)	Frees the memory associated with the TSendMail (see page 12061) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 12065)	Displays a dialog where the user can compose an email message and then sends the message.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.

V	HandlesTarget (see page 12066)	Indicates whether the action can execute.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 12066)	Enables the action.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnHint (see page 12071)	Occurs when the mouse pauses over a client control or menu item.

	OnUpdate (see page 12072)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 12067)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Enabled (see page 12067)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 12068)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 12068)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 12069)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 12069)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 12069)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 12070)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 12070)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Text (see page 12071)	Specifies the content of the mail message.
	Visible (see page 12071)	Specifies the Visible property for client controls and menu items.

67.1.24.2 TSendMail.Destroy Destructor

Frees the memory associated with the TSendMail (see page 12061) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSendMail();
```

Description

Do not explicitly destroy actions. Action objects are created and destroyed by the action manager or action list that uses them.

See Also

Free (see page 22300)

67.1.24.3 TSendMail Methods

The methods of the TSendMail class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 12065)	Displays a dialog where the user can compose an email message and then sends the message.
	HandlesTarget (see page 12066)	Indicates whether the action can execute.
	UpdateTarget (see page 12066)	Enables the action.

67.1.24.3.1 TSendMail.ExecuteTarget Method

Displays a dialog where the user can compose an email message and then sends the message.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter, and displays a dialog where the user can compose an email message. The body of the message is initialized to the value of the Text (see page 12071) property. When the user exits the dialog, the message is sent.

ExecuteTarget always spawns a temporary MAPI session for every mail message, which lasts only for the duration of the message. If necessary, the user is prompted with a login dialog.

See Also

Text (see page 12071)

67.1.24.3.2 TSendMail.HandlesTarget Method

Indicates whether the action can execute.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

HandlesTarget ignores the Target parameter and always returns true, because the ability to send mail is not dependant on the state of the application.

67.1.24.3.3 TSendMail.UpdateTarget Method

Enables the action.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically the application is idle so that actions can update themselves according to current conditions.

In TSendMail (see page 12061), UpdateTarget ignores the Target parameter and always sets the Enabled (see page

12067) property to true.

See Also

Enabled (see page 310)

67.1.24.4 TSendMail Properties

The properties of the TSendMail class are listed here.

Published Properties

	Name	Description
	Caption (see page 12067)	Represents the caption of client controls and menu items.
	Enabled (see page 12067)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 12068)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 12068)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 12069)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 12069)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 12069)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 12070)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 12070)	Specifies the ShortCut property for client menu items.
	Text (see page 12071)	Specifies the content of the mail message.
	Visible (see page 12071)	Specifies the Visible property for client controls and menu items.

67.1.24.4.1 TSendMail.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

67.1.24.4.2 TSendMail.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

67.1.24.4.3 TSendMail.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

67.1.24.4.4 TSendMail.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

67.1.24.4.5 TSendMail.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

67.1.24.4.6 TSendMail.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

67.1.24.4.7 TSendMail.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

67.1.24.4.8 TSendMail.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

67.1.24.4.9 TSendMail.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

67.1.24.4.10 TSendMail.Text Property

Specifies the content of the mail message.

Pascal

```
property Text: TStrings;
```

C++

```
__property TStrings Text;
```

Description

Set Text to the content of the mail message. Because Text is a TStrings object, you can use the string list editor at design time to compose the message. Use the TStrings object's LoadFromFile method to load the value of Text from a file.

When the action executes, TSendMail (see page 12061) displays a dialog for the user to compose the mail message. In that dialog, the body of the message is initialized to Text.

See Also

ExecuteTarget (see page 12065)

LoadFromFile (see page 3061)

67.1.24.4.11 TSendMail.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

67.1.24.5 TSendMail Events

The events of the TSendMail class are listed here.

Published Events

	Name	Description
	OnHint (see page 12071)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 12072)	Occurs when the application is idle or when the action list updates.

67.1.24.5.1 TSendMail.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

67.1.24.5.2 TSendMail.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
```

```
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

67.1.25 TTabAction Class

TTabAction is the base class for action objects designed to respond to the buttons in a tabbed dialog.

Class Hierarchy

Pascal

```
TTabAction = class(TCustomAction);
```

C++

```
class TTabAction : public TCustomAction;
```

File

ExtActns

Description

TTabAction implements the common behavior for actions that operate on tabbed controls (TCustomTabControl descendants). Each descendant class is designed to perform a specific function in a tabbed dialog such as a wizard. Do not use TTabAction directly in an application. Instead, use a TTabAction descendant such as TNextTab (see page 11949) or TPreviousTab (see page 11977).

TTabAction introduces the TabControl (see page 12080) property, which specifies the page control or tab control on which this action operates. It ensures that the action is only enabled when the TabControl (see page 12080) property is assigned to an enabled TPageControl or TTabControl object.

See Also

TPreviousTab (see page 11977)

TNextTab (see page 11949)

TPageControl (see page 4459)

TCustomTabControl (see page 3921)

TTabControl (see page 4841)

67.1.25.1 TTabAction Members

The following tables list the members exposed by TTabAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 12079)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SelectNextTab (see page 12079)	This is SelectNextTab, a member of class TTabAction.

	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	AfterTabChange (see page 12082)	Occurs immediately after the action changes the page of the associated tab control.
	BeforeTabChange (see page 12082)	Occurs immediately before the action changes the page of the associated tab control.
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.
	OnValidateTab (see page 12083)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

	Create (see page 12078)	Creates an instance of TTabAction (see page 12073).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 12078)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 12079)	Enables or disables the action, depending on the target tab control.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	SkipHiddenTabs (see page 12080)	Indicates whether the tab action should consider hidden tabbed pages when moving to the next page.
	TabControl (see page 12080)	Specifies the target of the action.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.
	Wrap (see page 12081)	Indicates whether the tab action should "wrap" when navigating to the next tab page.

Published Properties

	Name	Description
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

67.1.25.2 TTabAction.Create Constructor

Creates an instance of TTabAction (see page 12073).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TTabAction(TComponent * AOwner);
```

Description

Do not call the TTabAction (see page 12073) constructor. Actions are added at design time to an action list or action manager.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TTabAction (see page 12073) instance.

See Also

Owner (see page 2807)

67.1.25.3 TTabAction Methods

The methods of the TTabAction class are listed here.

Protected Methods

	Name	Description
	Notification (see page 12079)	Responds when components are created or destroyed.
	SelectNextTab (see page 12079)	This is SelectNextTab, a member of class TTabAction.

Public Methods

	Name	Description
	HandlesTarget (see page 12078)	Indicates whether the action can execute its function.
	UpdateTarget (see page 12079)	Enables or disables the action, depending on the target tab control.

67.1.25.3.1 TTabAction.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides an action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the currently active object.

In TTabAction (see page 12073), HandlesTarget ignores the Target parameter, and checks, instead, whether the TabControl (see page 12080) property has been set. If a target tab control has been assigned to the TabControl (see page 12080) property, HandlesTarget returns true. Otherwise, it returns false.

See Also

ExecuteTarget (see page 2712)

TabControl (see page 12080)

67.1.25.3.2 TTabAction.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TBasicAction overrides the Notification method to check whether the component that is the value of the ActionComponent property is about to be freed. If so, it sets the ActionComponent property to nil (Delphi) or NULL (C++).

See Also

ActionComponent (see page 2715)

67.1.25.3.3 TTabAction.SelectNextTab Method

Pascal

```
procedure SelectNextTab(GoForward: Boolean = True);
```

C++

```
__fastcall SelectNextTab(Boolean GoForward = True);
```

Description

This is SelectNextTab, a member of class TTabAction.

67.1.25.3.4 TTabAction.UpdateTarget Method

Enables or disables the action, depending on the target tab control.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to the current conditions.

Target is the potential target object identified by the application.

In TTabAction (see page 12073), UpdateTarget ignores the Target parameter and checks, instead, whether the TabControl (see page 12080) property is set. If TabControl (see page 12080) identifies an enabled page control or tab control that has

more than one page, UpdateTarget sets the Enabled property to true. If not, it sets Enabled to false.

After setting the Enabled property, UpdateTarget generates an OnValidateTab (see page 12083) event, where application-specific code can further determine whether the action should be enabled.

See Also

Enabled (see page 310)

TabControl (see page 12080)

Enabled (see page 5821)

OnValidateTab (see page 12083)

67.1.25.4 TTabAction Properties

The properties of the TTabAction class are listed here.

Public Properties

	Name	Description
	SkipHiddenTabs (see page 12080)	Indicates whether the tab action should consider hidden tabbed pages when moving to the next page.
	TabControl (see page 12080)	Specifies the target of the action.
	Wrap (see page 12081)	Indicates whether the tab action should "wrap" when navigating to the next tab page.

67.1.25.4.1 TTabAction.SkipHiddenTabs Property

Indicates whether the tab action should consider hidden tabbed pages when moving to the next page.

Pascal

```
property SkipHiddenTabs: Boolean;
```

C++

```
__property Boolean SkipHiddenTabs;
```

Description

Set SkipHiddenTabs to true if you do not want the action to move to a tab page if its TabVisible property is false. Set SkipHiddenTabs to false if the tab action can move to and make visible a hidden page.

Note: SkipHiddenTabs only has an effect when the TabControl (see page 12080) property identifies a TPageControl object. (TTabControl does not support hidden pages).

See Also

ExecuteTarget (see page 11954)

ExecuteTarget (see page 11982)

TabVisible (see page 4956)

TPageControl (see page 4459)

67.1.25.4.2 TTabAction.TabControl Property

Specifies the target of the action.

Pascal

```
property TabControl: TCustomTabControl;
```

C++

```
__property TCustomTabControl TabControl;
```

Description

Set TabControl to the control on which the action operates when its ExecuteTarget method is called. Typically, this is a page control or tab control object that implements the pages of a tabbed dialog.

An application must explicitly assign a value to the TabControl property before the TTabAction (see page 12073) descendant is enabled. The action does not infer the target page control when it fires.

See Also

ExecuteTarget (see page 11954)

ExecuteTarget (see page 11982)

67.1.25.4.3 TTabAction.Wrap Property

Indicates whether the tab action should "wrap" when navigating to the next tab page.

Pascal

```
property Wrap: Boolean;
```

C++

```
__property Boolean Wrap;
```

Description

When Wrap is true, the tab action can move previous to the first tabbed page or after the last tabbed page in the associated control. When Wrap is false, an attempt to move beyond the first or last page does not result in any page change, although it still results in a BeforeTabChange (see page 12082) and an AfterTabChange (see page 12082) event or in an OnFinish event.

See Also

ExecuteTarget (see page 11954)

ExecuteTarget (see page 11982)

TabControl (see page 12080)

BeforeTabChange (see page 12082)

AfterTabChange (see page 12082)

OnFinish (see page 11962)

67.1.25.5 TTabAction Events

The events of the TTabAction class are listed here.

Public Events

	Name	Description
	AfterTabChange (see page 12082)	Occurs immediately after the action changes the page of the associated tab control.
	BeforeTabChange (see page 12082)	Occurs immediately before the action changes the page of the associated tab control.

	OnValidateTab (see page 12083)	Occurs when the application is idle or when the action list updates.

67.1.25.5.1 TTabAction.AfterTabChange Event

Occurs immediately after the action changes the page of the associated tab control.

Pascal

```
property AfterTabChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterTabChange;
```

Description

Write an AfterTabChange event handler to perform any additional tasks after the tab action changes the current page of the associated tab control. Typically, this event is used to initialize the current page of the tab control.

Note: AfterTabChange only occurs if the action tries to change the page of the tab control (even if it is not successful). If the action attempts some other task (for example, when TNextPage is on the last page of the tab control), AfterTabChange does not occur.

See Also

BeforeTabChange (see page 12082)

TabControl (see page 12080)

Wrap (see page 12081)

67.1.25.5.2 TTabAction.BeforeTabChange Event

Occurs immediately before the action changes the page of the associated tab control.

Pascal

```
property BeforeTabChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeTabChange;
```

Description

Write a BeforeTabChange event handler to perform any additional tasks before the tab action changes the current page of the associated tab control. Typically, this event is used to save or respond to the changes the user made on the current page before it changes.

Note: BeforeTabChange only occurs if the action tries to change the page of the tab control (even if it is not successful). If the action attempts some other task (for example, when TNextPage is on the last page of the tab control), BeforeTabChange does not occur.

See Also

AfterTabChange (see page 12082)

TabControl (see page 12080)

Wrap (see page 12081)

67.1.25.5.3 TTabAction.OnValidateTab Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnValidateTab: TValidateTabEvent;
```

C++

```
__property TValidateTabEvent OnValidateTab;
```

Description

Use the OnValidateTab event rather than the OnUpdate event to update the properties of a tab action when the application is idle. Unlike the OnUpdate event, which prevents the default behavior of the UpdateTarget (see page 12079) method, OnValidateTab occurs after UpdateTarget (see page 12079) has set the Enabled property to reflect whether the target tab control is enabled.

OnValidateTab can be used to update any of the properties of the tab action, but it is intended primarily for enabling or disabling the action, depending on the contents of the current page of the target tab control.

Sender is the tab action component to update in this event.

TabControl (see page 12080) is the target tab control.

Enabled reflects the Enabled property of this action. Set Enabled in the event handler to enable or disable the action depending on current conditions.

See Also

OnUpdate (see page 2718)

UpdateTarget (see page 12079)

TabControl (see page 12080)

Enabled (see page 310)

67.1.26 TURLAction Class

TURLAction is the base class for actions that perform some function based on an URL (see page 12093).

Class Hierarchy

Pascal

```
TURLAction = class(TCustomAction);
```

C++

```
class TURLAction : public TCustomAction;
```

File

ExtActns

Description

Use TURLAction as a base class when creating an action component that uses a Uniform Resource Locator (URL (see page 12093)). TURLAction introduces the URL (see page 12093) property to get or set the URL (see page 12093) that it works with, but does not actually do anything with that URL (see page 12093). Rather, TURLAction descendants, such as TBrowseURL (see page 11832) or TDownLoadURL (see page 11857), override the ExecuteTarget method to perform some operation that uses the value of the URL (see page 12093) property.

TURLAction automatically disables itself if the URL (see page 12093) property has not been assigned.

See Also

TBrowseURL (see page 11832)

TDownLoadURL (see page 11857)

67.1.26.1 TURLAction Members

The following tables list the members exposed by TURLAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 12088)	Indicates whether the action can execute.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 12088)	Enables or disables the action, depending on whether the URL (see page 12093) property is set.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnHint (see page 12093)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 12094)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 12089)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Enabled (see page 12089)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 12090)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 12090)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 12091)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 12091)	Indicates the Help hint for client controls and menu items.

	ImageIndex (see page 12091)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 12092)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 12092)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	URL (see page 12093)	Specifies the Uniform Resource Locator (URL) on which this action performs its function.
	Visible (see page 12093)	Specifies the Visible property for client controls and menu items.

67.1.26.2 TURLAction Methods

The methods of the TURLAction class are listed here.

Public Methods

	Name	Description
	HandlesTarget (see page 12088)	Indicates whether the action can execute.
	UpdateTarget (see page 12088)	Enables or disables the action, depending on whether the URL (see page 12093) property is set.

67.1.26.2.1 TURLAction.HandlesTarget Method

Indicates whether the action can execute.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

HandlesTarget always returns true, because TURLAction (see page 12083) does not use the target object.

67.1.26.2.2 TURLAction.UpdateTarget Method

Enables or disables the action, depending on whether the URL (see page 12093) property is set.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically the application is idle so that actions can update themselves according to current conditions.

In TURLAction (see page 12083), UpdateTarget checks whether the URL (see page 12093) property has been set. If so, UpdateTarget sets Enabled (see page 12089) to true. If not, UpdateTarget sets Enabled (see page 12089) to false.

See Also

Enabled (see page 310)

URL (see page 12093)

67.1.26.3 TURLAction Properties

The properties of the TURLAction class are listed here.

Published Properties

	Name	Description
	Caption (see page 12089)	Represents the caption of client controls and menu items.
	Enabled (see page 12089)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 12090)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 12090)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 12091)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 12091)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 12091)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 12092)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 12092)	Specifies the ShortCut property for client menu items.
	URL (see page 12093)	Specifies the Uniform Resource Locator (URL) on which this action performs its function.
	Visible (see page 12093)	Specifies the Visible property for client controls and menu items.

67.1.26.3.1 TURLAction.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

67.1.26.3.2 TURLAction.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

67.1.26.3.3 TURLAction.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

67.1.26.3.4 TURLAction.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

67.1.26.3.5 TURLAction.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

67.1.26.3.6 TURLAction.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

67.1.26.3.7 TURLAction.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

67.1.26.3.8 TURLAction.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

67.1.26.3.9 TURLAction.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

67.1.26.3.10 TURLAction.URL Property

Specifies the Uniform Resource Locator (URL) on which this action performs its function.

Pascal

```
property URL: String;
```

C++

```
__property AnsiString URL;
```

Description

URL is the Uniform Resource Locator (URL) with which this action works. Each TURLAction (see page 12083) descendant performs a different function that uses this URL.

This value is not assigned automatically: it is the responsibility of the application to supply a value for URL. If no value is supplied, TURLAction (see page 12083) automatically disables itself.

See Also

UpdateTarget (see page 12088)

67.1.26.3.11 TURLAction.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

67.1.26.4 TURLAction Events

The events of the TURLAction class are listed here.

Published Events

	Name	Description
	OnHint (see page 12093)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 12094)	Occurs when the application is idle or when the action list updates.

67.1.26.4.1 TURLAction.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

67.1.26.4.2 TURLAction.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
```

```
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

67.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TShowCmd (see page 12096)	This is record ExtActns.TShowCmd.
	TURLDownloadStatus (see page 12097)	This is record ExtActns.TURLDownloadStatus.

67.2.1 ExtActns.TShowCmd Enumeration

Pascal

```
TShowCmd = (
  scHide,
  scMaximize,
  scMinimize,
  scRestore,
  scShow,
  scShowDefault,
  scShowMaximized,
  scShowMinimized,
  scShowMinNoActive,
  scShowNA,
  scShowNoActivate,
  scShowNormal
);
```

C++

```
enum TShowCmd {
  scHide,
  scMaximize,
  scMinimize,
  scRestore,
  scShow,
  scShowDefault,
  scShowMaximized,
  scShowMinimized,
  scShowMinNoActive,
  scShowNA,
  scShowNoActivate,
  scShowNormal
};
```

File

ExtActns

Description

This is record ExtActns.TShowCmd.

67.2.2 ExtActns.TURLDownloadStatus Enumeration

Pascal

```
TURLDownloadStatus = (
  dsFindingResource,
  dsConnecting,
  dsRedirecting,
  dsBeginDownloadData,
  dsDownloadingData,
  dsEndDownloadData,
  dsBeginDownloadComponents,
  dsInstallingComponents,
  dsEndDownloadComponents,
  dsUsingCachedCopy,
  dsSendingRequest,
  dsClassIDAvailable,
  dsMIMETypeAvailable,
  dsCacheFileNameAvailable,
  dsBeginSyncOperation,
  dsEndSyncOperation,
  dsBeginUploadData,
  dsUploadingData,
  dsEndUploadData,
  dsProtocolClassID,
  dsEncoding,
  dsVerifiedMIMETypeAvailable,
  dsClassInstallLocation,
  dsDecoding,
  dsLoadingMIMEHandler,
  dsContentDispositionAttach,
  dsFilterReportMIMEType,
  dsCLSIDCanInstantiate,
  dsIUnKnownAvailable,
  dsDirectBind,
  dsRawMIMEType,
  dsProxyDetecting,
  dsAcceptRanges,
  dsCookieSent,
  dsCompactPolicyReceived,
  dsCookieSuppressed,
  dsCookieStateUnknown,
  dsCookieStateAccept,
  dsCookeStateReject,
  dsCookieStatePrompt,
  dsCookieStateLeash,
  dsCookieStateDowngrade,
  dsPolicyHREF,
  dsP3PHeader,
  dsSessionCookieReceived,
  dsPersistentCookieReceived,
  dsSessionCookiesAllowed
);
```

C++

```
enum TURLDownloadStatus {
  dsFindingResource,
  dsConnecting,
  dsRedirecting,
  dsBeginDownloadData,
  dsDownloadingData,
  dsEndDownloadData,
```

```
    dsBeginDownloadComponents,
    dsInstallingComponents,
    dsEndDownloadComponents,
    dsUsingCachedCopy,
    dsSendingRequest,
    dsClassIDAvailable,
    dsMIMETypeAvailable,
    dsCacheFileNameAvailable,
    dsBeginSyncOperation,
    dsEndSyncOperation,
    dsBeginUploadData,
    dsUploadingData,
    dsEndUploadData,
    dsProtocolClassID,
    dsEncoding,
    dsVerifiedMIMETypeAvailable,
    dsClassInstallLocation,
    dsDecoding,
    dsLoadingMIMEHandler,
    dsContentDispositionAttach,
    dsFilterReportMIMEType,
    dsCLSIDCanInstantiate,
    dsIUnKnownAvailable,
    dsDirectBind,
    dsRawMIMEType,
    dsProxyDetecting,
    dsAcceptRanges,
    dsCookieSent,
    dsCompactPolicyReceived,
    dsCookieSuppressed,
    dsCookieStateUnknown,
    dsCookieStateAccept,
    dsCookeStateReject,
    dsCookieStatePrompt,
    dsCookieStateLeash,
    dsCookieStateDowngrade,
    dsPolicyHREF,
    dsP3PHeader,
    dsSessionCookieReceived,
    dsPersistentCookieReceived,
    dsSessionCookiesAllowed
  };
```

File

ExtActns

Description

This is record ExtActns.TURLDownloadStatus.

67.3 Types

The following table lists types in this documentation.

Types

Name	Description
TDownloadProgressEvent (see page 12099)	This is type ExtActns.TDownloadProgressEvent.
TValidateTabEvent (see page 12099)	This is type ExtActns.TValidateTabEvent.

67.3.1 ExtActns.TDownloadProgressEvent Type

Pascal

```
TDownloadProgressEvent = procedure (Sender: TDownLoadURL; Progress, ProgressMax: Cardinal;
StatusCode: TURLDownloadStatus; StatusText: String; var Cancel: Boolean) of object;
```

C++

```
(Sender: TDownLoadURL; Progress, ProgressMax: Cardinal; StatusCode: TURLDownloadStatus;
StatusText: String; var Cancel: Boolean) ( TDownloadProgressEvent)();
```

File

ExtActns

Description

This is type ExtActns.TDownloadProgressEvent.

67.3.2 ExtActns.TValidateTabEvent Type

Pascal

```
TValidateTabEvent = procedure (Sender: TTabAction; TabControl: TCustomTabControl; var Enabled:
Boolean) of object;
```

C++

```
(Sender: TTabAction; TabControl: TCustomTabControl; var Enabled: Boolean) (
TValidateTabEvent)();
```

File

ExtActns

Description

This is type ExtActns.TValidateTabEvent.

68 ExtCtrls Namespace

This is namespace ExtCtrls.

Topics

Name	Description
Classes (see page 12104)	The following table lists classes in this documentation.
Functions (see page 13262)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 13264)	The following table lists structs, records, enums in this documentation.
Types (see page 13272)	The following table lists types in this documentation.
Constants (see page 13279)	The following table lists constants in this documentation.

Classes

	Name	Description
	EGridPanelException (see page 12105)	The base class for runtime exceptions that occur in grid panels.
	TBevel (see page 12105)	TBevel represents a beveled outline.
	TBoundLabel (see page 12123)	TBoundLabel is the type of a labeled edit control's label.
	TCellCollection (see page 12172)	A collection of TCellItem (see page 12177) objects.
	TCellItem (see page 12177)	Represents an item in a TCellCollection (see page 12172).
	TColorBox (see page 12182)	TColorBox represents a combo box that lets users select a color.
	TColorListBox (see page 12246)	TColorListBox represents a list box that lets users select a color.
	TColumnCollection (see page 12311)	Controls (see page 5711) the **Collection Editor** for columns within a grid panel.
	TColumnItem (see page 12315)	This is class ExtCtrls.TColumnItem.
	TControlBar (see page 12316)	TControlBar manages the layout of toolbar components.
	TControlCollection (see page 12393)	This is class ExtCtrls.TControlCollection.
	TControlItem (see page 12400)	This is class ExtCtrls.TControlItem.
	TCustomColorBox (see page 12406)	TCustomColorBox represents a combo box that lets users select a color.
	TCustomColorListBox (see page 12426)	This is class ExtCtrls.TCustomColorListBox.
	TCustomControlBar (see page 12446)	TCustomControlBar is the base class for control bar components.
	TCustomFlowPanel (see page 12475)	The base class for all flow panel components.
	TCustomGridPanel (see page 12489)	The base class for all grid panel components.
	TCustomLabeledEdit (see page 12508)	TCustomLabeledEdit is the base class for edit controls that have an integrated label.
	TCustomPanel (see page 12525)	TCustomPanel is the base class for all panel components.
	TCustomRadioGroup (see page 12547)	TCustomRadioGroup is the base class for radio-group components.
	TCustomTrayIcon (see page 12564)	The base class for TTrayIcon (see page 13252).

	TFlowPanel (see page 12579)	TFlowPanel implements a flow panel control in which components are placed in pre-defined locations.
	TGridPanel (see page 12660)	TGridPanel implements a grid panel control in which each component is placed within a cell on a grid.
	THeader (see page 12744)	THeader is a sectioned visual control that displays text and allows each section to be resized with the mouse.
	TImage (see page 12777)	TImage displays a graphical image.
	TLabeledEdit (see page 12833)	TLabeledEdit is an edit control that has an associated label.
	TNotebook (see page 12902)	TNotebook displays multiple pages, each with its own set of controls.
	TPage (see page 12961)	This is class ExtCtrls.TPage.
	TPaintBox (see page 12981)	TPaintBox provides a canvas that applications can use for rendering an image.
	TPanel (see page 13032)	TPanel implements a generic panel control.
	TRadioGroup (see page 13115)	TRadioGroup represents a group of radio buttons that function together.
	TRowCollection (see page 13167)	Controls (see page 5711) the **Collection Editor** for rows within a grid panel.
	TRowItem (see page 13171)	This is class ExtCtrls.TRowItem.
	TShape (see page 13172)	TShape represents a geometric shape that can be drawn on a form.
	TSplitter (see page 13210)	TSplitter divides the client area of a form into resizable panes.
	TTimer (see page 13242)	TTimer encapsulates the Windows API timer functions.
	TTrayIcon (see page 13252)	Creates an icon in the system tray next to the clock.

Constants

Name	Description
NoColorSelected (see page 13279)	This is constant ExtCtrls.NoColorSelected.
WM_SYSTEM_TRAY_MESSAGE (see page 13279)	This is constant ExtCtrls.WM_SYSTEM_TRAY_MESSAGE.

Functions

	Name	Description
	Frame3D (see page 13262)	Draws a three dimensional frame on a canvas.
	NotebookHandlesNeeded (see page 13263)	This is function ExtCtrls.NotebookHandlesNeeded.

Structs, Records, Enums

	Name	Description
	TBalloonFlags (see page 13264)	A type specified when showing a balloon hint that puts one of the default icons on the balloon message.
	TBandDrawingStyle (see page 13265)	This is record ExtCtrls.TBandDrawingStyle.
	TBandPaintOption (see page 13265)	TBandPaintOption and TBandPaintOptions (see page 13274) specify which parts of a band are painted.
	TBevelShape (see page 13266)	TBevelShape determines the overall shape of a bevel control.
	TBevelStyle (see page 13267)	TBevelStyle indicates whether a bevel appears raised or lowered.
	TColorBoxStyles (see page 13267)	This is record ExtCtrls.TColorBoxStyles.
	TCornerEdge (see page 13268)	The size of the cornering for the edge of the bands in a TControlBar (see page 12316).
	TExpandStyle (see page 13268)	Specifies how a grid panel expands.
	TFlowStyle (see page 13269)	This is record ExtCtrls.TFlowStyle.
	TLabelPosition (see page 13269)	TLabelPosition specifies the position of the label in a labeled edit control.
	TResizeStyle (see page 13270)	This is record ExtCtrls.TResizeStyle.
	TShapeType (see page 13270)	TShapeType determines the shape of a TShape (see page 13172) control.
	TSizeStyle (see page 13271)	This is record ExtCtrls.TSizeStyle.

Types

Name	Description
NaturalNumber (see page 13272)	This is type ExtCtrls.NaturalNumber.
TBandDragEvent (see page 13272)	This is type ExtCtrls.TBandDragEvent.
TBandInfoEvent (see page 13273)	This is type ExtCtrls.TBandInfoEvent.
TBandMoveEvent (see page 13273)	This is type ExtCtrls.TBandMoveEvent.
TBandPaintEvent (see page 13273)	This is type ExtCtrls.TBandPaintEvent.

TBandPaintOptions (see page 13274)	This is type ExtCtrls.TBandPaintOptions.
TBeginBandMoveEvent (see page 13274)	This is type ExtCtrls.TBeginBandMoveEvent.
TCanResizeEvent (see page 13274)	This is type ExtCtrls.TCanResizeEvent.
TCellSpan (see page 13275)	This is type ExtCtrls.TCellSpan.
TColorBoxStyle (see page 13275)	TColorBoxStyle and TColorBoxStyles (see page 13267) specify how a color box control displays its selections.
TEndBandMoveEvent (see page 13276)	This is type ExtCtrls.TEndBandMoveEvent.
TGetColorsEvent (see page 13276)	This is type ExtCtrls.TGetColorsEvent.
TLBGetColorsEvent (see page 13276)	This is type ExtCtrls.TLBGetColorsEvent.
TPanelBevel (see page 13277)	TPanelBevel specifies a bevel's cut.
TRowSize (see page 13277)	This is type ExtCtrls.TRowSize.
TSectionEvent (see page 13278)	TSectionEvent is the type of event handlers that respond when a section of a THeader (see page 12744) control is resized.
TSplitterCanResizeEvent (see page 13278)	This is type ExtCtrls.TSplitterCanResizeEvent.

68.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EGridPanelException (see page 12105)	The base class for runtime exceptions that occur in grid panels.
	TBevel (see page 12105)	TBevel represents a beveled outline.
	TBoundLabel (see page 12123)	TBoundLabel is the type of a labeled edit control's label.
	TCellCollection (see page 12172)	A collection of TCellItem (see page 12177) objects.
	TCellItem (see page 12177)	Represents an item in a TCellCollection (see page 12172).
	TColorBox (see page 12182)	TColorBox represents a combo box that lets users select a color.
	TColorListBox (see page 12246)	TColorListBox represents a list box that lets users select a color.
	TColumnCollection (see page 12311)	Controls (see page 5711) the **Collection Editor** for columns within a grid panel.
	TColumnItem (see page 12315)	This is class ExtCtrls.TColumnItem.
	TControlBar (see page 12316)	TControlBar manages the layout of toolbar components.
	TControlCollection (see page 12393)	This is class ExtCtrls.TControlCollection.
	TControlItem (see page 12400)	This is class ExtCtrls.TControlItem.
	TCustomColorBox (see page 12406)	TCustomColorBox represents a combo box that lets users select a color.
	TCustomColorListBox (see page 12426)	This is class ExtCtrls.TCustomColorListBox.
	TCustomControlBar (see page 12446)	TCustomControlBar is the base class for control bar components.
	TCustomFlowPanel (see page 12475)	The base class for all flow panel components.
	TCustomGridPanel (see page 12489)	The base class for all grid panel components.
	TCustomLabeledEdit (see page 12508)	TCustomLabeledEdit is the base class for edit controls that have an integrated label.
	TCustomPanel (see page 12525)	TCustomPanel is the base class for all panel components.
	TCustomRadioGroup (see page 12547)	TCustomRadioGroup is the base class for radio-group components.
	TCustomTrayIcon (see page 12564)	The base class for TTrayIcon (see page 13252).
	TFlowPanel (see page 12579)	TFlowPanel implements a flow panel control in which components are placed in pre-defined locations.
	TGridPanel (see page 12660)	TGridPanel implements a grid panel control in which each component is placed within a cell on a grid.
	THeader (see page 12744)	THeader is a sectioned visual control that displays text and allows each section to be resized with the mouse.
	TImage (see page 12777)	TImage displays a graphical image.
	TLabeledEdit (see page 12833)	TLabeledEdit is an edit control that has an associated label.
	TNotebook (see page 12902)	TNotebook displays multiple pages, each with its own set of controls.
	TPage (see page 12961)	This is class ExtCtrls.TPage.
	TPaintBox (see page 12981)	TPaintBox provides a canvas that applications can use for rendering an image.
	TPanel (see page 13032)	TPanel implements a generic panel control.
	TRadioGroup (see page 13115)	TRadioGroup represents a group of radio buttons that function together.
	TRowCollection (see page 13167)	Controls (see page 5711) the **Collection Editor** for rows within a grid panel.
	TRowItem (see page 13171)	This is class ExtCtrls.TRowItem.
	TShape (see page 13172)	TShape represents a geometric shape that can be drawn on a form.
	TSplitter (see page 13210)	TSplitter divides the client area of a form into resizable panes.
	TTimer (see page 13242)	TTimer encapsulates the Windows API timer functions.
	TTrayIcon (see page 13252)	Creates an icon in the system tray next to the clock.

68.1.1 EGridPanelException Class

The base class for runtime exceptions that occur in grid panels.

Class Hierarchy

Pascal

```
EGridPanelException = class(Exception);
```

C++

```
class EGridPanelException : public Exception;
```

File

ExtCtrls

Description

Use EGridPanelException as the base class for creating custom exceptions that occur in grid panels.

68.1.2 TBevel Class

TBevel represents a beveled outline.

Class Hierarchy

Pascal

```
TBevel = class(TGraphicControl);
```

C++

```
class TBevel : public TGraphicControl;
```

File

ExtCtrls

Description

Use TBevel to create beveled boxes, frames, or lines. The bevel can appear raised or lowered.

See Also

TShape (see page 13172)

Frame3D (see page 13262)

Bevels

Graphic Controls

68.1.2.1 TBevel Members

The following tables list the members exposed by TBevel.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.

	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.

	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 12113)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 12113)	Creates and initializes an instance of TBevel (see page 12105).
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.

	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.

	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Align (see page 12114)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 12117)	Specifies how the control is anchored to its parent.
	Constraints (see page 12118)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentShowHint (see page 12118)	Determines where a control looks to find out if its Help Hint should be shown.
	Shape (see page 12119)	Determines the shape of the bevel.
	ShowHint (see page 12119)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 12121)	Determines whether the bevel appears raised or lowered.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 12121)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.2.2 TBevel.Create Constructor

Creates and initializes an instance of TBevel (see page 12105).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBevel(TComponent * AOwner);
```

Description

Call Create to instantiate a bevel at runtime. Bevels placed on forms at design time are created automatically.

See Also

Shape (see page 12119)

Style (see page 12121)

68.1.2.3 TBevel Methods

The methods of the TBevel class are listed here.

Protected Methods

	Name	Description
	Paint (see page 12113)	Render control surface.

68.1.2.3.1 TBevel.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas (see page 6053)

68.1.2.4 TBevel Properties

The properties of the TBevel class are listed here.

Published Properties

	Name	Description
	Align (see page 12114)	Determines how the control aligns within its container (parent control).
	Anchors (see page 12117)	Specifies how the control is anchored to its parent.
	Constraints (see page 12118)	Specifies the size constraints for the control.
	ParentShowHint (see page 12118)	Determines where a control looks to find out if its Help Hint should be shown.
	Shape (see page 12119)	Determines the shape of the bevel.
	ShowHint (see page 12119)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 12121)	Determines whether the bevel appears raised or lowered.
	Visible (see page 12121)	Determines whether the component appears onscreen.

68.1.2.4.1 TBevel.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
```

```
        pts[i]->Brush->Color = colorPalette[i];
    }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
```

```
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

68.1.2.4.2 TBevel.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

68.1.2.4.3 TBevel.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

68.1.2.4.4 TBevel.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

68.1.2.4.5 TBevel.Shape Property

Determines the shape of the bevel.

Pascal

```
property Shape: TBevelShape;
```

C++

```
__property TBevelShape Shape;
```

Description

Set Shape to specify whether the bevel appears as a line, box, frame, or space. For shapes that can appear either raised or lowered, the Style (see page 12121) property indicates which effect is used.

The default value for Shape is bsBox.

See Also

ClientRect (see page 5810)

Style (see page 12121)

68.1.2.4.6 TBevel.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
```

```
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

68.1.2.4.7 TBevel.Style Property

Determines whether the bevel appears raised or lowered.

Pascal

```
property Style: TBevelStyle;
```

C++

```
__property TBevelStyle Style;
```

Description

Set Style to indicate whether the bevel should create a raised or a lowered effect. When the Shape (see page 12119) property is bsBox, the entire client area appears raised or lowered. For all other values of Shape (see page 12119), the bevel displays a raised or lowered line along the edge or edges of the client area. The default value of Style is bsLowered.

See Also

Shape (see page 12119)

68.1.2.4.8 TBevel.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
```

```
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

68.1.3 TBoundLabel Class

TBoundLabel is the type of a labeled edit control's label.

Class Hierarchy

Pascal

```
TBoundLabel = class(TCustomLabel);
```

C++

```
class TBoundLabel : public TCustomLabel;
```

File

ExtCtrls

Description

TBoundLabel implements the label of a labeled edit control. Unlike other label controls such as TLabel, TBoundLabel "belongs" to the labeled edit control (it is a subcomponent of the edit control). This means that the bound label's Owner is the edit control rather than the form, and the edit control is responsible for loading and saving the bound label.

Use the properties of TBoundLabel to specify how you want the labeled edit control's label to appear. For example, you can use the Caption (see page 12133) property to specify the text that appears in the label. You can use the Alignment and Layout (see page 12141) properties to specify the placement of that caption within the label.

See Also

TLabeledEdit (see page 12833)

TLabel (see page 21544)

TComponent (see page 2762)

68.1.3.1 TBoundLabel Members

The following tables list the members exposed by TBoundLabel.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.

	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	AdjustBounds (see page 12131)	Resizes the label when the text changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDrawText (see page 21296)	Renders the text in the client area.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.

	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetLabelText (see page 21297)	Returns the value of the Caption (see page 21303) property.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	Loaded (see page 21297)	Adjusts the size of the label when it is first loaded. .
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 21298)	Responds to notifications when objects are about to be created or destroyed.
	Paint (see page 6052)	Render control surface.
	Paint (see page 21298)	Draws the image of the label.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.

	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetAutoSize (see page 21298)	Sets the AutoSize (see page 21299) property.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	Alignment (see page 21299)	Controls (see page 5711) the horizontal placement of the text within the label.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AutoSize (see page 21299)	Determines whether the size of the label automatically resizes to accommodate the text.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	EllipsisPosition (see page 21305)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	FocusControl (see page 21306)	Designates a windowed control associated with the label.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.

	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 12131)	Creates a new TBoundLabel (see page 12123) object.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Canvas (see page 21300)	Provides the drawing surface used by the graphic control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Events

	Name	Description
	OnClick (see page 12154)	Occurs when the user clicks the control.
	OnContextPopup (see page 12156)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 12157)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 12157)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 12159)	Occurs when the user drags an object over a control.
	OnEndDock (see page 12162)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 12162)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 12163)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 12164)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 12166)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 12168)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 12171)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 12171)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	BiDiMode (see page 12132)	Specifies the bi-directional mode for the control.
	Caption (see page 12133)	Specifies a text string that identifies the control to the user.
	Color (see page 12135)	Specifies the background color of the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 12136)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 12137)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 12137)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 12139)	Controls the attributes of text written on or in the control.
	Height (see page 12140)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Layout (see page 12141)	Specifies the vertical placement of the text within the label.
R	Left (see page 12141)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 12142)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 12142)	Determines where a control looks for its color information.
	ParentFont (see page 12143)	Determines where a control looks for its font information.
	ParentShowHint (see page 12143)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 12144)	Identifies the pop-up menu associated with the control.
	ShowAccelChar (see page 12148)	Determines how an ampersand in the label text is displayed.
	ShowHint (see page 12149)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

	Tag (see page 2808)	Stores an integer value as part of a component.
R	Top (see page 12151)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Transparent (see page 12152)	Specifies whether controls that sit below the label on a form can be seen through the label.
	Width (see page 12153)	Specifies the horizontal size of the control or form in pixels.
	WordWrap (see page 12153)	Specifies whether the label text wraps when it is too long for the width of the label.

68.1.3.2 TBoundLabel.Create Constructor

Creates a new TBoundLabel (see page 12123) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBoundLabel(TComponent * AOwner);
```

Description

Do not create instances of TBoundLabel (see page 12123). Bound labels are instantiated by the labeled edit controls to which they belong.

AOwner specifies the labeled edit control that acts as the bound label's Owner.

See Also

TLabeledEdit (see page 12833)

Owner (see page 2807)

68.1.3.3 TBoundLabel Methods

The methods of the TBoundLabel class are listed here.

Protected Methods

	Name	Description
V	AdjustBounds (see page 12131)	Resizes the label when the text changes.

68.1.3.3.1 TBoundLabel.AdjustBounds Method

Resizes the label when the text changes.

Pascal

```
procedure AdjustBounds; override;
```

C++

```
virtual __fastcall AdjustBounds();
```

Description

When the text in the label or the font changes, AdjustBounds is implicitly called. It redraws the text in the client area, and then resizes the label if the AutoSize property is true.

When WordWrap is true, the width of the label is fixed. If AutoSize is also true, changes to the text cause the label to change in height. When AutoSize is true and WordWrap is false, the height of the label is determined by the font, and changes to the text cause the label to change in width.

Override AdjustBounds to make additional adjustments when the text changes.

See Also

DoDrawText (see page 21296)

AutoSize (see page 21299)

68.1.3.4 TBoundLabel Properties

The properties of the TBoundLabel class are listed here.

Published Properties

	Name	Description
	BiDiMode (see page 12132)	Specifies the bi-directional mode for the control.
	Caption (see page 12133)	Specifies a text string that identifies the control to the user.
	Color (see page 12135)	Specifies the background color of the control.
	DragCursor (see page 12136)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 12137)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 12137)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 12139)	Controls the attributes of text written on or in the control.
	Height (see page 12140)	Specifies the vertical size of the control in pixels.
	Layout (see page 12141)	Specifies the vertical placement of the text within the label.
R	Left (see page 12141)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	ParentBiDiMode (see page 12142)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 12142)	Determines where a control looks for its color information.
	ParentFont (see page 12143)	Determines where a control looks for its font information.
	ParentShowHint (see page 12143)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 12144)	Identifies the pop-up menu associated with the control.
	ShowAccelChar (see page 12148)	Determines how an ampersand in the label text is displayed.
	ShowHint (see page 12149)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
R	Top (see page 12151)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Transparent (see page 12152)	Specifies whether controls that sit below the label on a form can be seen through the label.
	Width (see page 12153)	Specifies the horizontal size of the control or form in pixels.
	WordWrap (see page 12153)	Specifies whether the label text wraps when it is too long for the width of the label.

68.1.3.4.1 TBoundLabel.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

68.1.3.4.2 TBoundLabel.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
```

```
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

68.1.3.4.3 TBoundLabel.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

68.1.3.4.4 TBoundLabel.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

68.1.3.4.5 TBoundLabel.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

68.1.3.4.6 TBoundLabel.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
```

```
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

68.1.3.4.7 TBoundLabel.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

68.1.3.4.8 TBoundLabel.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
HideSelection, Height example
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
```

```
        SelLength := Length(Text);
        if (I = 1) then
          HideSelection := False;
      end;
  end;
```

See Also

ClientHeight (see page 5808)

Width (see page 5849)

Using TDBListBox and TDBComboBox

68.1.3.4.9 TBoundLabel.Layout Property

Specifies the vertical placement of the text within the label.

Pascal

```
property Layout: TTextLayout;
```

C++

```
__property TTextLayout Layout;
```

Description

Set Layout to specify how the text of the label is placed within the ClientRect of the label control. Layout is the vertical analog to the Alignment property.

See Also

Alignment (see page 21299)

ClientRect (see page 5810)

68.1.3.4.10 TBoundLabel.Left Property

Specifies the horizontal coordinate of the left edge of a component relative to its parent.

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

Use the Left property to determine where the left side of the control begins or to reposition the left side of the control.

If the control is contained in another control, the Left and Top properties are relative to the parent control. If the control is contained directly by the form, the property values are relative to the form. For forms, the value of the Left property is relative to the screen in pixels.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
```

```
{
  Timer1->Interval = 100;
  if (ActiveControl)
    ActiveControl->Left = ActiveControl->Left + 1;
}
```

Delphi Examples:

```
{
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  if ActiveControl <> nil then
    ActiveControl.Left := ActiveControl.Left + 1;
end;
```

See Also

Top (see page 5844)

68.1.3.4.11 TBoundLabel.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

68.1.3.4.12 TBoundLabel.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

68.1.3.4.13 TBoundLabel.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

68.1.3.4.14 TBoundLabel.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show

them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

68.1.3.4.15 TBoundLabel.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
```

```
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
```

```
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
```

```
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

68.1.3.4.16 TBoundLabel.ShowAccelChar Property

Determines how an ampersand in the label text is displayed.

Pascal

```
property ShowAccelChar: Boolean;
```

C++

```
__property Boolean ShowAccelChar;
```

Description

Set ShowAccelChar to true to allow the label to display an underlined accelerator key value. When ShowAccelChar is true, any character preceded by an ampersand (&) appears underlined. If the FocusControl property is set, the windowed control specified by the FocusControl property receives input focus when the user types that underlined character. To display an ampersand when ShowAccelChar is true, use two ampersands (&&) to stand for the single ampersand that is displayed.

Set ShowAccelChar to false to display the label text with all ampersands appearing as ampersands. When ShowAccelChar is false, the value of the FocusControl property is not used.

C++ Examples:

```
/*
This example uses two labels on a form. The first label has
a caption with an accelerator character in it. The second
label also includes an ampersand, but it does not appear as
an accelerator character.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Label1->ShowAccelChar = true;
  Label1->Caption = "An &Underlined character appears here";
  Label2->ShowAccelChar = false;
  Label2->Caption = "An ampersand (&) appears here";
}
```

Delphi Examples:

```
{
This example uses two labels on a form. The first label has
a caption with an accelerator character in it. The second
label also includes an ampersand, but it does not appear as
an accelerator character.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Label1.ShowAccelChar := True;
  Label1.Caption := 'An &Underlined character appears here';
  Label2.ShowAccelChar := False;
  Label2.Caption := 'An ampersand (&) appears here';
end;
```

See Also

FocusControl (see page 21306)

GetLabelText (see page 21297)

68.1.3.4.17 TBoundLabel.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
```

```
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

68.1.3.4.18 TBoundLabel.Top Property

Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

Use Top to locate the top of the control or reposition the control to a different Y coordinate. The Top property, like the Left property, is the position of the control relative to its container. Thus, if a control is contained in a TPanel, the Left and Top properties are relative to the panel. If the control is contained directly by the form, it is relative to the form. For forms, the value of the Top property is relative to the screen in pixels.

C++ Examples:

```
/*
This example uses an updown control and a group box on a
form, with several controls contained within the group box.
When the updown control or is clicked, the controls within
the group box are moved in the appropriate direction.
*/
void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  int I;
  TControl *ChildControl;

  for (I = 0; I < GroupBox1->ControlCount; I++)
  {
    ChildControl = GroupBox1->Controls[I];
    if (Button == btNext)
      ChildControl->Top = ChildControl->Top + 15;
    else
      ChildControl->Top = ChildControl->Top - 15;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorarray[5] = {clYellow, clGreen, clBlue, clLime, clFuchsia};
```

```
  GroupBox1->Brush->Color = clRed;
  for (int i = 0; i < GroupBox1->ControlCount; i++)
    ((TWinControl *)GroupBox1->Controls[i])->Brush->Color = colorarray[i];
}
```

Delphi Examples:

```
{
This example uses an updown control and a group box on a
form, with several controls contained within the group box.
When the updown control or is clicked, the controls within
the group box are moved in the appropriate direction.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
const
  colorarray : Array[0..4] of TColor = (clYellow, clGreen, clBlue, clLime, clFuchsia);
begin
   GroupBox1.Brush.Color := clRed;
  for I:= 0 to GroupBox1.ControlCount -1 do
  begin
    TWinControl(GroupBox1.Controls[I]).Brush.Color := colorarray[I];
  end;
end;

procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
var
  I: Integer;
  ChildControl: TControl;
begin
  for I:= 0 to GroupBox1.ControlCount -1 do
  begin
    ChildControl := GroupBox1.Controls[I];
    if Button = btNext then
      ChildControl.Top := ChildControl.Top + 15
    else
      ChildControl.Top := ChildControl.Top - 15;
  end;
end;
```

See Also

Left (see page 5830)

68.1.3.4.19 TBoundLabel.Transparent Property

Specifies whether controls that sit below the label on a form can be seen through the label.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Set Transparent to true to prevent the label from obscuring other controls on the form. For example, if the label is used to add text to a graphic, set Transparent to true so that the label does not stand out as a separate object.

Writing text so that the background is transparent is slower than writing text when Transparent is false. If the label is not obscuring a complicated image, performance can be improved by setting the background color of the label to match the object beneath it and setting Transparent to false.

See Also

Color (see page 5812)

ParentColor (see page 5834)

68.1.3.4.20 TBoundLabel.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

See Also

ClientWidth (see page 5811)

Height (see page 5825)

Left (see page 5830)

68.1.3.4.21 TBoundLabel.WordWrap Property

Specifies whether the label text wraps when it is too long for the width of the label.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the label to display multiple line of text. When WordWrap is true, text that is too wide for the label control wraps at the right margin and continues in additional lines.

Set WordWrap to false to limit the label to a single line. When WordWrap is false, text that is too wide for the label appears truncated.

See Also

ClientWidth (see page 5811)

GetLabelText (see page 21297)

68.1.3.5 TBoundLabel Events

The events of the TBoundLabel class are listed here.

Published Events

	Name	Description
	OnClick (see page 12154)	Occurs when the user clicks the control.
	OnContextPopup (see page 12156)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 12157)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 12157)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 12159)	Occurs when the user drags an object over a control.
	OnEndDock (see page 12162)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 12162)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 12163)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 12164)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 12166)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 12168)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 12171)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 12171)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

68.1.3.5.1 TBoundLabel.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
```

```
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

68.1.3.5.2 TBoundLabel.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

68.1.3.5.3 TBoundLabel.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

68.1.3.5.4 TBoundLabel.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

68.1.3.5.5 TBoundLabel.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
```

```
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code comes from an application that contains a list box
  and three labels, each with a different font and color.  The
  DragMode property for each of the labels is dmAutomatic.
  The user can select a label and drag it to a list box and
  drop it. When the label is dropped, the items in the list
  box assume the color and font of the dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.3.5.6 TBoundLabel.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.3.5.7 TBoundLabel.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
```

```
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

68.1.3.5.8 TBoundLabel.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control.

However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.3.5.9 TBoundLabel.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
```

```
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
```

```
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.3.5.10 TBoundLabel.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
```

```
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
```

```
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.3.5.11 TBoundLabel.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
  property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
```

```
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

68.1.3.5.12 TBoundLabel.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

68.1.3.5.13 TBoundLabel.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

68.1.4 TCellCollection Class

A collection of TCellItem (see page 12177) objects.

Class Hierarchy

Pascal

```
TCellCollection = class(TOwnedCollection);
```

C++

```
class TCellCollection : public TOwnedCollection;
```

File

ExtCtrls

Description

Use TCellCollection to represent all the cell items in a grid panel.

68.1.4.1 TCellCollection Members

The following tables list the members exposed by TCellCollection.

Protected Methods

	Name	Description
	Added (see page 2736)	Responds when items are added to the collection.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.

V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
V	GetAttr (see page 12174)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
V	GetAttrCount (see page 12175)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 12175)	Returns a specified item in the collection.
V	GetItemAttr (see page 12176)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2915)	Returns the owner of an object.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 12176)	This is SetItem, a member of class TCellCollection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 12176)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 2736)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 2734)	Creates and initializes a collection.
	Create (see page 2914)	Creates and initializes a TOwnedCollection (see page 2912) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 12176)	Returns the Owner of the cell collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 12177)	Lists the items of the cell collection.

68.1.4.2 TCellCollection Methods

The methods of the TCellCollection class are listed here.

Protected Methods

	Name	Description
V	GetAttr (see page 12174)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
V	GetAttrCount (see page 12175)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 12175)	Returns a specified item in the collection.
V	GetItemAttr (see page 12176)	Returns the value of a custom attribute assigned to one of the collection's items.
	SetItem (see page 12176)	This is SetItem, a member of class TCellCollection.
V	Update (see page 12176)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Owner (see page 12176)	Returns the Owner of the cell collection.

68.1.4.2.1 TCellCollection.GetAttr Method

Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.

Pascal

```
function GetAttr(Index: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetAttr(int Index);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. The GetAttr method returns the name of an attribute.

Index identifies the attribute whose name is requested. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

As implemented by TCollection, GetAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttrCount (see page 2744)

GetItemAttr (see page 2745)

68.1.4.2.2 TCellCollection.GetAttrCount Method

Returns the number of custom attributes associated with items in the collection.

Pascal

```
function GetAttrCount: Integer; override;
```

C++

```
virtual __fastcall int GetAttrCount();
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetAttrCount returns the number of distinct attributes assigned to each item in the collection.

As implemented in TCollection, GetAttrCount always returns 0, because TCollection defines no custom attributes.

See Also

GetAttr (see page 2743)

GetItemAttr (see page 2745)

68.1.4.2.3 TCellCollection.GetItem Method

Returns a specified item in the collection.

Pascal

```
function GetItem(Index: Integer): TCellItem;
```

C++

```
__fastcall TCellItem GetItem(int Index);
```

Description

GetItem is the protected read implementation of the Items property.

See Also

Items (see page 2751)

68.1.4.2.4 TCellCollection.GetItemAttr Method

Returns the value of a custom attribute assigned to one of the collection's items.

Pascal

```
function GetItemAttr(Index: Integer; ItemIndex: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetItemAttr(int Index, int ItemIndex);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetItemAttr returns the value of one of these attributes for a specified item in the collection.

Index identifies which attribute's value is desired. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

ItemIndex identifies the item whose attribute value is desired. This is an index into the Items property array.

As implemented in TCollection, GetItemAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttr (see page 2743)

GetAttrCount (see page 2744)

68.1.4.2.5 TCellCollection.Owner Method

Returns the Owner of the cell collection.

Pascal

```
function Owner: TCustomGridPanel;
```

C++

```
__fastcall TCustomGridPanel Owner();
```

Description

Use the Owner method to obtain a reference to the list view that owns this cell collection.

68.1.4.2.6 TCellCollection.SetItem Method

Pascal

```
procedure SetItem(Index: Integer; Value: TCellItem);
```

C++

```
__fastcall SetItem(int Index, TCellItem Value);
```

Description

This is SetItem, a member of class TCellCollection.

68.1.4.2.7 TCellCollection.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed (see page 2740)

Changed (see page 2758)

68.1.4.3 TCellCollection Properties

The properties of the TCellCollection class are listed here.

Public Properties

	Name	Description
	Items (see page 12177)	Lists the items of the cell collection.

68.1.4.3.1 TCellCollection.Items Property

Lists the items of the cell collection.

Pascal

```
property Items [Index: Integer]: TCellItem;
```

C++

```
__property TCellItem Items[int Index];
```

Description

Use the Items property to list the items of TCellCollection (see page 12172).

68.1.5 TCellItem Class

Represents an item in a TCellCollection (see page 12172).

Class Hierarchy

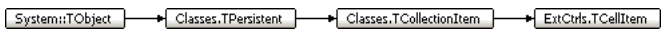

Pascal

```
TCellItem = class(TCollectionItem);
```

C++

```
class TCellItem : public TCollectionItem;
```

File

ExtCtrls

Description

The TCellItem class represents a cell on a grid panel.

68.1.5.1 TCellItem Members

The following tables list the members exposed by TCellItem.

Protected Methods

	Name	Description
	AssignTo (see page 12180)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.
	SetSizeStyle (see page 12180)	This is SetSizeStyle, a member of class TCellItem.
	SetValue (see page 12181)	This is SetValue, a member of class TCellItem.

Protected Properties

	Name	Description
	AutoAdded (see page 12181)	This is AutoAdded, a member of class TCellItem.
	Size (see page 12181)	This is Size, a member of class TCellItem.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 12179)	Creates and initializes a TCollectionItem instance.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	SizeStyle (see page 12181)	Specifies the manner in which column size or row size is specified.
	Value (see page 12182)	Sets the size of the cell item.

68.1.5.2 TCellItem.Create Constructor

Creates and initializes a TCollectionItem instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TCellItem(TCollection Collection);
```

Description

Do not call Create to instantiate a collection item. Instead, call the Add method of the collection to which the item should belong.

Collection identifies the TCollection instance to which the new item belongs.

See Also

Add (see page 2736)

Collection (see page 2761)

68.1.5.3 TCellItem Methods

The methods of the TCellItem class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 12180)	Copies the properties of an object to a destination object.
	SetSizeStyle (see page 12180)	This is SetSizeStyle, a member of class TCellItem.
	SetValue (see page 12181)	This is SetValue, a member of class TCellItem.

68.1.5.3.1 TCellItem.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign (see page 2930)

EConvertError (see page 22724)

68.1.5.3.2 TCellItem.SetSizeStyle Method

Pascal

```
procedure SetSizeStyle(Value: TSizeStyle);
```

C++

```
__fastcall SetSizeStyle(TSizeStyle Value);
```

Description

This is SetSizeStyle, a member of class TCellItem.

68.1.5.3.3 TCellItem.SetValue Method

Pascal

```
procedure SetValue(Value: Double);
```

C++

```
__fastcall SetValue(Double Value);
```

Description

This is SetValue, a member of class TCellItem.

68.1.5.4 TCellItem Properties

The properties of the TCellItem class are listed here.

Protected Properties

	Name	Description
	AutoAdded (see page 12181)	This is AutoAdded, a member of class TCellItem.
	Size (see page 12181)	This is Size, a member of class TCellItem.

Published Properties

	Name	Description
	SizeStyle (see page 12181)	Specifies the manner in which column size or row size is specified.
	Value (see page 12182)	Sets the size of the cell item.

68.1.5.4.1 TCellItem.AutoAdded Property

Pascal

```
property AutoAdded: Boolean;
```

C++

```
__property Boolean AutoAdded;
```

Description

This is AutoAdded, a member of class TCellItem.

68.1.5.4.2 TCellItem.Size Property

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

This is Size, a member of class TCellItem.

68.1.5.4.3 TCellItem.SizeStyle Property

Specifies the manner in which column size or row size is specified.

Pascal

```
property SizeStyle: TSizeStyle;
```

C++

```
__property TSizeStyle SizeStyle;
```

Description

Use the SizeStyle property to specify whether the size of the row or column in a grid is an absolute value, a percentage of the grid size, or generated automatically.

Style	Description
ssAbsolute	The number specified for the Value (see page 12182) property is interpreted as the number of pixels for the row or column.
ssAuto	The number specified for the Value (see page 12182) property is ignored, and the row or column is sized automatically.
ssPercent	The number specified for the Value (see page 12182) property is interpreted as a percentage of the entire grid panel.

68.1.5.4.4 TCellItem.Value Property

Sets the size of the cell item.

Pascal

```
property Value: Double;
```

C++

```
__property Double Value;
```

Description

Use the Value property to specify the size of the column or row in the grid. This value can be set automatically or expressed as a percentage of the grid size or as an absolute value, depending on the SizeStyle (see page 12181). For example, if SizeStyle (see page 12181) is set to ssPercent, and you specify a value of 60, the column occupies approximately 60% of the grid panes width or a row occupies approximately 60% of the grid panel height.

68.1.6 TColorBox Class

TColorBox represents a combo box that lets users select a color.

Class Hierarchy

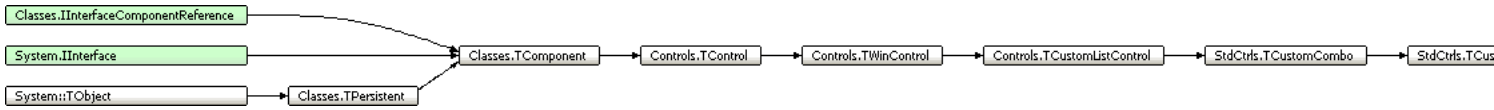

Pascal

```
TColorBox = class(TCustomColorBox);
```

C++

```
class TColorBox : public TCustomColorBox;
```

File

ExtCtrls

Description

Use TColorBox to provide the user with a drop-down combo box from which to select a color. Use the Style (see page 12217) property to specify which colors the color box should list. Each color is displayed next to a small square that is drawn in the corresponding color.

Use the Selected (see page 12215) property to access the color that the user selects. The Colors (see page 12424) property provides access to the entire list of colors, while the ColorNames (see page 12423) property lists their names.

See Also

TComboBox (see page 21064)

TColorDialog (see page 11270)

68.1.6.1 TColorBox Members

The following tables list the members exposed by TColorBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDrawItem (see page 21224)	Occurs when an item in an owner-draw combo box needs to be displayed.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMeasureItem (see page 21225)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefEditProc (see page 21167)	This is FDefEditProc, a member of class TCustomCombo.
	FDefListProc (see page 21167)	This is FDefListProc, a member of class TCustomCombo.

	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FDropHandle (see page 21168)	This is FDropHandle, a member of class TCustomCombo.
	FDroppingDown (see page 21168)	This is FDroppingDown, a member of class TCustomCombo.
	FEditHandle (see page 21168)	This is FEditHandle, a member of class TCustomCombo.
	FEditInstance (see page 21168)	This is FEditInstance, a member of class TCustomCombo.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFocusChanged (see page 21168)	This is FFocusChanged, a member of class TCustomCombo.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FIsFocused (see page 21169)	This is FIsFocused, a member of class TCustomCombo.
	FListHandle (see page 21169)	This is FListHandle, a member of class TCustomCombo.
	FListInstance (see page 21169)	This is FListInstance, a member of class TCustomCombo.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveIndex (see page 21169)	This is FSaveIndex, a member of class TCustomCombo.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustDropDown (see page 21171)	Adjusts the size and position of the drop down list.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21171)	Generates an OnChange (see page 21197) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	CloseUp (see page 12419)	Generates an OnCloseUp event.
	CloseUp (see page 21172)	Generates an OnCloseUp (see page 21199) event.
V	ComboWndProc (see page 21173)	Responds to Windows messages sent to the edit control or drop-down list.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.

	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 21215)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 12420)	Creates the window used to implement the color box.
	CreateWnd (see page 21174)	Creates the Windows control that represents the combo box.
	CreateWnd (see page 21215)	Creates the Windows control that represents the combo box.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWindowHandle (see page 21174)	Destroys all of the windows controls that implement the combo box.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21215)	Destroys the Windows control that represents the combo box.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.

	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawItem (see page 12420)	Renders an individual item in the color box.
V	DrawItem (see page 21216)	Generates an OnDrawItem (see page 21224) event.
	DropDown (see page 21175)	Generates an OnDropDown (see page 21199) event.
V	EditWndProc (see page 21175)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetDroppedDown (see page 21176)	Returns the value of the DroppedDown (see page 21185) property.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
A	GetItemCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
V	GetItemCount (see page 21216)	Returns the number of items in the Items (see page 21187) property list.
A	GetItemHt (see page 21177)	Returns the value of the ItemHeight (see page 21186) property.
V	GetItemHt (see page 21216)	Returns the value of the ItemHeight (see page 21186) property.
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
V	GetItemIndex (see page 21177)	Returns the value of the ItemIndex property.
A	GetItemsClass (see page 21177)	Indicates the class that implements the Items (see page 21187) property.
V	GetItemsClass (see page 21217)	Returns the class used to implement the Items (see page 21187) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelLength (see page 21178)	Returns the value of the SelLength (see page 21190) property.
	GetSelStart (see page 21178)	Returns the value of the SelStart (see page 21193) property.
	GetSelText (see page 21217)	Returns the value of the SelText (see page 21222) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.

	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
V	KeyDown (see page 12420)	Generates an OnKeyDown event.
	KeyPress (see page 6163)	Respond to keyboard input.
V	KeyPress (see page 12421)	Generates an OnKeyPress event.
V	KeyPress (see page 21217)	Responds to WM_CHAR messages sent to the combo box.
	KeyUp (see page 6164)	Respond to released key.
	ListWndProc (see page 21178)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 12421)	Initializes the combo box after its form is loaded into memory.
V	Loaded (see page 21179)	Initializes the combo box after its form is loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
V	MeasureItem (see page 21218)	Generates an OnMeasureItem (see page 21225) event.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	PickCustomColor (see page 12422)	This is PickCustomColor, a member of class TCustomColorBox.
	PopulateList (see page 12422)	This is PopulateList, a member of class TCustomColorBox.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.

	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
V	Select (see page 12422)	Generates an OnSelect event.
	Select (see page 21179)	Generates an OnSelect (see page 21201) event.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectItem (see page 21218)	Selects a specified item from the drop-down list.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetDropDownCount (see page 21180)	Sets the value of the DropDownCount (see page 21185) property.
	SetDroppedDown (see page 21180)	Sets the value of the DroppedDown (see page 21185) property.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetItemHeight (see page 21181)	Sets the value of the ItemHeight (see page 21186) property.
A	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
V	SetItemIndex (see page 21181)	Sets the value of the ItemIndex property.
V	SetItems (see page 21181)	Sets the value of the Items (see page 21187) property.
	SetMaxLength (see page 21182)	Sets the value of the MaxLength (see page 21189) property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelLength (see page 21182)	Sets the value of the SelLength (see page 21190) property.
	SetSelStart (see page 21182)	Sets the value of the SelStart (see page 21193) property.
	SetStyle (see page 12422)	Sets the Style (see page 12425) property.
V	SetStyle (see page 21219)	Sets the Style (see page 21224) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.

	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WndProc (see page 21183)	Processes messages received by the combo box.
V	WndProc (see page 21219)	Processes messages received by the combo box.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
R	EditHandle (see page 21185)	Returns the window handle for the edit portion of the combo box.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	ItemCount (see page 21186)	Specifies the number of items in the drop-down list.
R	ListHandle (see page 21189)	Provides access to the window handle of the list of items in the combo box.
	MaxLength (see page 21189)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Sorted (see page 21222)	Determines whether the list portion of the combo box is alphabetized.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
A	AddItem (see page 5940)	Adds an item to the list control.
V	AddItem (see page 21170)	Adds an item to the drop-down list of items.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 5940)	Deletes all of the items from the list control.
V	Clear (see page 21172)	Deletes all text from the edit box and all items from the Items (see page 21187) list.
A	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
V	ClearSelection (see page 21172)	Deselects any selected items in the drop-down list.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
A	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
V	CopySelection (see page 21173)	Copies the selected item in the drop-down list to another list control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 12419)	Creates a new TCustomColorBox (see page 12406) object.
V	Create (see page 21166)	Creates and initializes a TCustomCombo (see page 21154) instance.
V	Create (see page 21214)	Creates and initializes a TCustomComboBox (see page 21201) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	DeleteSelected (see page 5941)	Deletes all the selected items.
V	DeleteSelected (see page 21174)	Deletes the selected item from the drop-down list.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 21166)	Destroys the combo box and frees its memory.
V	Destroy (see page 21214)	Destroys the combo box and frees its memory.

	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Focused (see page 21175)	Indicates whether the combo box has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
A	SelectAll (see page 5943)	Selects all items or all text in the selected item.
V	SelectAll (see page 21180)	Selects the entire block of text in the edit box.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AutoCloseUp (see page 21220)	Specifies whether the drop-down closes up automatically when the user selects an item.
	AutoCompleteDelay (see page 21220)	Specifies the delay between a key press and an attempt to autocomplete the field.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 21184)	Provides access to the drawing surface of the combo box.
	CharCase (see page 21221)	Determines the case of the text in the combo box.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ColorNames (see page 12423)	Lists the names of the colors in the color box.
R	Colors (see page 12424)	Lists the colors in the color box.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DroppedDown (see page 21185)	Indicates whether the drop-down list is currently displayed.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	Items (see page 21187)	Provides access to the list of items (strings) in the list portion of the combo box.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.

R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelLength (see page 21190)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 21193)	Specifies the position of the first selected character in the edit portion of the combo box.
	SelText (see page 21222)	Represents the selected text in the edit region.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 12221)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 12223)	Occurs when the user clicks the control.
	OnCloseUp (see page 12225)	Occurs when the drop-down list closes up due to some user action.
	OnContextPopup (see page 12225)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 12226)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 12226)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 12228)	Occurs when the user drags an object over a control.
	OnDropDown (see page 12231)	Occurs when the user opens the drop-down list.
	OnEndDock (see page 12232)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 12232)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 12233)	Occurs when a control receives the input focus.
	OnExit (see page 12236)	Occurs when the input focus shifts away from one control to another.
	OnGetColors (see page 12237)	Occurs when the control is populated with colors.
	OnKeyDown (see page 12238)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 12240)	Occurs when a key is pressed.
	OnKeyUp (see page 12242)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 12243)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 12244)	Occurs when the user moves the mouse outside of a control.
	OnSelect (see page 12244)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 12244)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 12245)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 12196)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 12200)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 12200)	Positions to matching list items as you type.
	AutoDropDown (see page 12201)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 12201)	Specifies which edges of the control are beveled.
	BevelInner (see page 12201)	Specifies the cut of the inner bevel.

	BevelKind (see page 12202)	Specifies the control's bevel style.
	BevelOuter (see page 12202)	Specifies the cut of the outer bevel.
	BiDiMode (see page 12203)	Specifies the bi-directional mode for the control.
	Color (see page 12203)	Specifies the background color of the control.
	Constraints (see page 12204)	Specifies the size constraints for the control.
	Ctl3D (see page 12205)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DefaultColorColor (see page 12205)	Specifies the color displayed in the color box for clDefault.
	DropDownCount (see page 12205)	Specifies the maximum number of items displayed in the drop-down list.
	Enabled (see page 12206)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 12206)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ItemHeight (see page 12207)	Specifies the height, in pixels, of the items in the drop-down list.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NoneColorColor (see page 12208)	Specifies the color displayed in the color box for clNone.
	ParentBiDiMode (see page 12208)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 12209)	Determines where a control looks for its color information.
	ParentCtl3D (see page 12209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 12210)	Determines where a control looks for its font information.
	ParentShowHint (see page 12210)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 12211)	Identifies the pop-up menu associated with the control.
	Selected (see page 12215)	Specifies the currently selected color.
	ShowHint (see page 12215)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 12217)	Controls what colors are listed and how their names are formatted.
	TabOrder (see page 12217)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 12218)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 12218)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.6.2 TColorBox Properties

The properties of the TColorBox class are listed here.

Published Properties

	Name	Description
	Align (see page 12196)	Determines how the control aligns within its container (parent control).
	Anchors (see page 12200)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 12200)	Positions to matching list items as you type.
	AutoDropDown (see page 12201)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.

	BevelEdges (see page 12201)	Specifies which edges of the control are beveled.
	BevelInner (see page 12201)	Specifies the cut of the inner bevel.
	BevelKind (see page 12202)	Specifies the control's bevel style.
	BevelOuter (see page 12202)	Specifies the cut of the outer bevel.
	BiDiMode (see page 12203)	Specifies the bi-directional mode for the control.
	Color (see page 12203)	Specifies the background color of the control.
	Constraints (see page 12204)	Specifies the size constraints for the control.
	Ctl3D (see page 12205)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColorColor (see page 12205)	Specifies the color displayed in the color box for clDefault.
	DropDownCount (see page 12205)	Specifies the maximum number of items displayed in the drop-down list.
	Enabled (see page 12206)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 12206)	Controls the attributes of text written on or in the control.
	ItemHeight (see page 12207)	Specifies the height, in pixels, of the items in the drop-down list.
	NoneColorColor (see page 12208)	Specifies the color displayed in the color box for clNone.
	ParentBiDiMode (see page 12208)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 12209)	Determines where a control looks for its color information.
	ParentCtl3D (see page 12209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 12210)	Determines where a control looks for its font information.
	ParentShowHint (see page 12210)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 12211)	Identifies the pop-up menu associated with the control.
	Selected (see page 12215)	Specifies the currently selected color.
	ShowHint (see page 12215)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 12217)	Controls what colors are listed and how their names are formatted.
	TabOrder (see page 12217)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 12218)	Determines if the user can tab to a control.
	Visible (see page 12218)	Determines whether the component appears onscreen.

68.1.6.2.1 TColorBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
```

```
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

```
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

68.1.6.2.2 TColorBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

68.1.6.2.3 TColorBox.AutoComplete Property

Positions to matching list items as you type.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

AutoComplete will position to matching list values as you type, saving you keystrokes.

As you type, the ItemIndex property is updated with the list entry that matches the characters you have entered. As you type, different list entries may match, and ItemIndex changes accordingly.

There is a delay of AutoCompleteDelay milliseconds between the characters typed and the autocomplete operation.

See Also

AutoCompleteDelay (see page 21220)

68.1.6.2.4 TColorBox.AutoDropDown Property

Specifies whether the drop-down list drops down automatically in response to user keystrokes.

Pascal

```
property AutoDropDown: Boolean;
```

C++

```
__property Boolean AutoDropDown;
```

Description

When AutoDropDown is true, the combo box automatically drops down its list when the user starts typing a string while the combo box has focus.

When AutoDropDown is false, the user must explicitly use the drop-down button to drop down the combo box list.

See Also

autocomplete (see page 21220)

OnDropDown (see page 21199)

DroppedDown (see page 21185)

68.1.6.2.5 TColorBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

68.1.6.2.6 TColorBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

68.1.6.2.7 TColorBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

68.1.6.2.8 TColorBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

68.1.6.2.9 TColorBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

68.1.6.2.10 TColorBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
```

```
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

68.1.6.2.11 TColorBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

68.1.6.2.12 TColorBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

68.1.6.2.13 TColorBox.DefaultColorColor Property

Specifies the color displayed in the color box for clDefault.

Pascal

```
property DefaultColorColor: TColor;
```

C++

```
__property TColor DefaultColorColor;
```

Description

Set DefaultColorColor to indicate what color the color box should use when drawing the icon for clDefault. This property only affects the color box if the Style property includes cbSystemColors and cbIncludeDefault.

Note: DefaultColorColor only affects the way clDefault appears in the color box. When you assign clDefault to a control, the control determines its own default color.

See Also

Colors (see page 12424)

NoneColorColor (see page 12424)

Style (see page 12425)

68.1.6.2.14 TColorBox.DropDownCount Property

Specifies the maximum number of items displayed in the drop-down list.

Pascal

```
property DropDownCount: Integer;
```

C++

```
__property int DropDownCount;
```

Description

By default, the drop-down list is long enough to contain eight items without requiring the user to scroll to see them all. To make the drop-down list smaller or larger, specify a number larger or smaller than eight as the DropDownCount value.

If the DropDownCount value is larger than the number of items in the Items property, the drop-down list will be just large enough to hold all the possible choices and no more. If the DropDownCount value is smaller then the number of item in the Items property, the drop down list will display a scroll bar.

See Also

Items (see page 21187)

ItemCount (see page 21186)

68.1.6.2.15 TColorBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

68.1.6.2.16 TColorBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name,

Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

68.1.6.2.17 TColorBox.ItemHeight Property

Specifies the height, in pixels, of the items in the drop-down list.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Read ItemHeight to determine the height of the items in the drop-down list.

The effect of setting ItemHeight varies, depending on the type of combo box and the style. For example, TComboBox lets you set the item height only when Style is set to csOwnerDrawFixed or csOwnerDrawVariable. If Style is csOwnerDrawVariable, ItemHeight is the default height for drawing list items, but this value can be overridden by an OnMeasureItem event handler.

TComboBoxEx does not let you change the value of ItemHeight.

Note: Combo boxes that do not support changes to ItemHeight do not raise an exception when you attempt to change ItemHeight; they simply ignore the attempt.

See Also

Items (see page 21187)

OnMeasureItem (see page 21225)

Style (see page 21224)

68.1.6.2.18 TColorBox.NoneColorColor Property

Specifies the color displayed in the color box for clNone.

Pascal

```
property NoneColorColor: TColor;
```

C++

```
__property TColor NoneColorColor;
```

Description

Set NoneColorColor to indicate what color the color box should use when drawing the icon for clNone. This property only affects the color box if the Style property includes cbSystemColors and cbIncludeNone.

Note: NoneColorColor only affects the way clNone appears in the color box. When you assign clNone to a control, the control determines how to interpret clNone.

See Also

Colors (see page 12424)

DefaultColorColor (see page 12424)

Style (see page 12425)

68.1.6.2.19 TColorBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

68.1.6.2.20 TColorBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

68.1.6.2.21 TColorBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

68.1.6.2.22 TColorBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

68.1.6.2.23 TColorBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

68.1.6.2.24 TColorBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      const int num_items = 4;
      for (int index = 0; index < num_items; ++index)
      {
          TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
          PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
          NewItem->Caption = "Menu Item " + IntToStr(index);
          NewItem->Tag = index;
          NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
          TNotifyEvent();
      }
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
          {
              case 0:
              {
                  ShowMessage("first item clicked");
                  break;
              }
              case 1:
              {
                  ShowMessage("second item clicked");
                  break;
              }
              case 2:
              {
                  ShowMessage("third item clicked");
                  break;
              }
              case 3:
              {
```

```
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
```

```
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
```

```
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

68.1.6.2.25 TColorBox.Selected Property

Specifies the currently selected color.

Pascal

```
property Selected: TColor;
```

C++

```
__property TColor Selected;
```

Description

Use Selected to get or set the selected color as a TColor value.

See Also

Colors (see page 12424)

ItemIndex

68.1.6.2.26 TColorBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
```

```
    Edit1.Hint := 'Enter your name';
    Edit1.ShowHint := True;
    with ListBox1 do
    begin
      for I := 1 to 10 do
        Items.Add('Item ' + IntToStr(I));
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

68.1.6.2.27 TColorBox.Style Property

Controls what colors are listed and how their names are formatted.

Pascal

```
property Style: TComboBoxStyle;
```

C++

```
__property TColorBoxStyle Style;__property TComboBoxStyle Style;
```

Description

Use Style to configure what colors appear in the list and how the names appear. Style is a set that includes 0 or more of the TColorBoxStyles flags.

See Also

DefaultColorColor (see page 12424)

NoneColorColor (see page 12424)

ColorNames (see page 12423)

Colors (see page 12424)

TColor (see page 14648)

TColorDialog (see page 11270)

68.1.6.2.28 TColorBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

68.1.6.2.29 TColorBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

68.1.6.2.30 TColorBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
```

```
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

68.1.6.3 TColorBox Events

The events of the TColorBox class are listed here.

Published Events

	Name	Description
	OnChange (see page 12221)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 12223)	Occurs when the user clicks the control.
	OnCloseUp (see page 12225)	Occurs when the drop-down list closes up due to some user action.
	OnContextPopup (see page 12225)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 12226)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 12226)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 12228)	Occurs when the user drags an object over a control.
	OnDropDown (see page 12231)	Occurs when the user opens the drop-down list.
	OnEndDock (see page 12232)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 12232)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 12233)	Occurs when a control receives the input focus.
	OnExit (see page 12236)	Occurs when the input focus shifts away from one control to another.
	OnGetColors (see page 12237)	Occurs when the control is populated with colors.
	OnKeyDown (see page 12238)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 12240)	Occurs when a key is pressed.
	OnKeyUp (see page 12242)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 12243)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 12244)	Occurs when the user moves the mouse outside of a control.
	OnSelect (see page 12244)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 12244)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 12245)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

68.1.6.3.1 TColorBox.OnChange Event

Occurs when the user changes the text displayed in the edit region.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action immediately after the user edits the text in the edit region or selects an item from the list. The Text property gives the new value in the edit region.

Note: OnChange only occurs in response to user actions. Changing the Text property programmatically does not trigger an OnChange event.

C++ Examples:

```
/*
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
*/
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  if (pCB)
    PageControl1->ActivePage = (TTabSheet *)(pCB->Items->Objects[pCB->ItemIndex]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
    ComboBox1->Items->AddObject(PageControl1->Pages[i]->Name, PageControl1->Pages[i]);
    ComboBox1->ItemIndex = 0;
  }
}
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
```

```
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to PageControl1.PageCount - 1 do
    ComboBox1.Items.AddObject(PageControl1.Pages[i].Name,
      PageControl1.Pages[i]);
  ComboBox1.ItemIndex := 0;
end;

procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  if (Sender is TComboBox) then
    with (Sender as TComboBox) do
      PageControl1.ActivePage := TTabSheet(Items.Objects[ItemIndex]);
end;
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
```

```
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

See Also

Text (see page 5842)

OnCloseUp (see page 21199)

68.1.6.3.2 TColorBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

68.1.6.3.3 TColorBox.OnCloseUp Event

Occurs when the drop-down list closes up due to some user action.

Pascal

```
property OnCloseUp: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCloseUp;
```

Description

Write an OnCloseUp event handler to implement special processing that needs to occur when the drop-down list closes up. For example, an OnCloseUp event handler can check whether the user changed the selected item while the list was dropped down and respond accordingly.

See Also

OnDropDown (see page 21199)

DroppedDown (see page 21185)

68.1.6.3.4 TColorBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

68.1.6.3.5 TColorBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

68.1.6.3.6 TColorBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
```

```
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:
```

```
procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

68.1.6.3.7 TColorBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler

is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;
```

```
procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.6.3.8 TColorBox.OnDropDown Event

Occurs when the user opens the drop-down list.

Pascal

```
property OnDropDown: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDropDown;
```

Description

Write an OnDropDown event handler to implement special processing that needs to occur only when the drop-down list is activated.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if (((TForm1 *)Screen->CustomForms[i])->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
```

```
    I: Integer;
  begin
    with ComboBox1 do
    begin
      Items.BeginUpdate; { prevent repaints until done }
      Items.Clear; { empty the list of any old values }
      for I := 0 to Screen.CustomFormCount - 1 do
      begin
        if (Screen.CustomForms[I].Visible) then
          Items.Add(Screen.CustomForms[I].Caption); { add form name }
      end;
      Items.EndUpdate; {reenable painting }
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ComboBox1.Sorted := True; { make sure the form names are sorted }
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Form2.Show;
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    Form3.Show;
  end;
```

See Also

DroppedDown (see page 21185)

OnCloseUp (see page 21199)

68.1.6.3.9 TColorBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.6.3.10 TColorBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

68.1.6.3.11 TColorBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
```

```
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
```

```
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

68.1.6.3.12 TColorBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
```

```
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

68.1.6.3.13 TColorBox.OnGetColors Event

Occurs when the control is populated with colors.

Pascal

```
property OnGetColors: TGetColorsEvent;
```

C++

```
__property TGetColorsEvent OnGetColors;
```

68.1.6.3.14 TColorBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
```

```
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
```

```
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

68.1.6.3.15 TColorBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

68.1.6.3.16 TColorBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
```

```
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

68.1.6.3.17 TColorBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

68.1.6.3.18 TColorBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

68.1.6.3.19 TColorBox.OnSelect Event

Occurs when the user selects a string in the drop-down list.

Pascal

```
property OnSelect: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSelect;
```

Description

Write an OnSelect event handler to respond when the user changes the selected item in the drop-down list. OnSelect occurs after the selection has changed and the Text property reflects the new item.

See Also

autocomplete (see page 21220)

ItemIndex

Text (see page 5842)

68.1.6.3.20 TColorBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is

automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

68.1.6.3.21 TColorBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

68.1.7 TColorListBox Class

TColorListBox represents a list box that lets users select a color.

Class Hierarchy

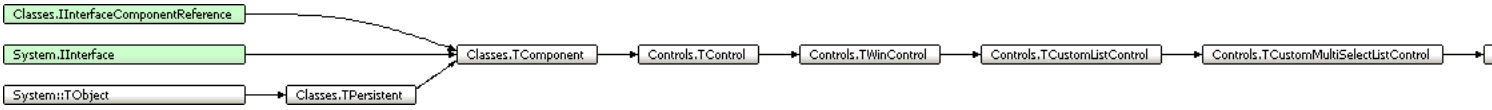

Pascal

```
TColorListBox = class(TCustomColorListBox);
```

C++

```
class TColorListBox : public TCustomColorListBox;
```

File

ExtCtrls

Description

TCustomColorBox (see page 12406) is the base class for TColorListBox and for combo boxes that display a list of colors. The list box displays a list of all the colors for which the Graphics (see page 14387) unit defines a constant. For a list of these colors, see TColor. Each color is displayed next to a small square that is drawn in the corresponding color.

Do not use TCustomColorBox (see page 12406) in applications. It is intended to act as a base class for color selection controls. Rather, to add a list box control with selectable colors, use the TColorListBox component. To use a drop-down combo box that lets users select a color, use the TColorBox (see page 12182) component, which appears on the Additional page of the component palette.

68.1.7.1 TColorListBox Members

The following tables list the members exposed by TColorListBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnData (see page 21357)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 21357)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 21358)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDrawItem (see page 21358)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMeasureItem (see page 21360)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.

	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FMoving (see page 21321)	This is FMoving, a member of class TCustomListBox.
	FMultiSelect (see page 5956)	This is FMultiSelect, a member of class TCustomMultiSelectListControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.

	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 21324)	Initializes the window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 12439)	Creates a Windows control corresponding to the list box component.
	CreateWnd (see page 21325)	Creates a Windows control corresponding to the list box component.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DeleteString (see page 21326)	Deletes a string from the list box.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21326)	Saves the list box items in a string list before freeing the underlying Windows control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFindData (see page 21326)	Returns the index of a string in the list box.
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetData (see page 21327)	Returns a string in the list box given its index.
	DoGetDataObject (see page 21327)	Returns the object associated with an item in a virtual list box.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.

V	DragCanceled (see page 21328)	Sends a mouse message canceling the drag-and-drop operation.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawItem (see page 12439)	Generates an OnDrawItem event.
V	DrawItem (see page 21328)	Generates an OnDrawItem (see page 21358) event.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetCount (see page 21329)	Returns the value of the Count (see page 21346) property.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetItemData (see page 21329)	Returns a 32-bit value associated with the specified item in a list box.
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
V	GetItemIndex (see page 21329)	Returns the value of the ItemIndex property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
A	GetSelCount (see page 5956)	Returns the value of the SelCount (see page 5957) property.
V	GetSelCount (see page 21330)	Returns the number of selected items.
	GetSelected (see page 21330)	Indicates whether a particular item is selected.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InternalGetItemData (see page 21330)	Retrieves the data associated with an item in the list box.
	InternalSetItemData (see page 21330)	Associates an integer with a specified item in the list box.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.

	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
V	KeyDown (see page 12440)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
V	KeyPress (see page 12441)	Responds when the user presses a key.
V	KeyPress (see page 21334)	Responds when the user presses a key.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 12441)	Initializes the control after it is loaded from a stream.
V	LoadRecreateItems (see page 21334)	This is LoadRecreateItems, a member of class TCustomListBox.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
V	MeasureItem (see page 21335)	Generates an OnMeasureItem (see page 21360) event.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	PickCustomColor (see page 12441)	This is PickCustomColor, a member of class TCustomColorListBox.
	PopulateList (see page 12442)	This is PopulateList, a member of class TCustomColorListBox.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetContent (see page 21335)	Removes all of the items from a list box.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
V	SaveRecreateItems (see page 21336)	This is SaveRecreateItems, a member of class TCustomListBox.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.

	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItemData (see page 21336)	Sets the 32-bit value associated with the specified item in a list box.
A	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
V	SetItemIndex (see page 21336)	Gives focus to the item with a specified index.
A	SetMultiSelect (see page 5956)	Sets the value of the MultiSelect (see page 5957) property.
V	SetMultiSelect (see page 21337)	Sets the MultiSelect property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetStyle (see page 12442)	This is SetStyle, a member of class TCustomColorListBox.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.

	WndProc (see page 6187)	Provides specific message responses for the control.
	WndProc (see page 21337)	Handles Windows messages.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 21343)	Determines whether the list box has a border.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Columns (see page 21346)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	ExtendedSelect (see page 21346)	This is ExtendedSelect, a member of class TCustomListBox.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 21347)	Determines whether the list box displays the partial items.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Sorted (see page 21353)	Specifies whether the items in a list box are arranged alphabetically.
	TabWidth (see page 21355)	Specifies the size of the tabs in the list box.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddItem (see page 5940)	Adds an item to the list control.
	AddItem (see page 21322)	Adds an item to the list box
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 5940)	Deletes all of the items from the list control.
V	Clear (see page 21322)	Deletes all items from the list box.
A	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
V	ClearSelection (see page 21323)	Deselects all selected items in the list box, if possible.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
A	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
V	CopySelection (see page 21323)	Copies the selected items to another list-type control
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 12438)	Instantiates a list box control.
V	Create (see page 21320)	Instantiates a list box control.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	DeleteSelected (see page 5941)	Deletes all the selected items.
V	DeleteSelected (see page 21325)	Deletes the selected items from the list box
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 21320)	Destroys the instance of a list box control.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ItemAtPos (see page 21331)	Returns the index of the list box item indicated by the coordinates of a point on the control.
	ItemRect (see page 21333)	Returns the rectangle that surrounds the item specified in the Item parameter.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
A	SelectAll (see page 5943)	Selects all items or all text in the selected item.
V	SelectAll (see page 21336)	Selects the entire block of text in the list box.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AutoCompleteDelay (see page 21342)	Specifies the delay between a key press and an attempt to autocomplete the field.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 21343)	Provides a drawing surface when implementing a handler for the OnDrawItem (see page 21358) event of an owner-draw list box.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ColorNames (see page 12442)	This is ColorNames, a member of class TCustomColorListBox.
R	Colors (see page 12443)	This is Colors, a member of class TCustomColorListBox.

R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Count (see page 21346)	Indicates the number of items in the list box.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	Items (see page 21348)	Contains the strings that appear in the list box.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	MultiSelect (see page 5957)	Specifies whether the user can select more than one item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ScrollWidth (see page 21351)	Specifies the width, in pixels, by which the list box can scroll horizontally.
R	SelCount (see page 5957)	Indicates the number of selected items.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TopIndex (see page 21356)	Specifies the index number of the item that appears at the top of the list box.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 12283)	Occurs when the user clicks the control.

	OnContextPopup (see page 12285)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 12286)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 12286)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 12288)	Occurs when the user drags an object over a control.
	OnEndDock (see page 12291)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 12291)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 12292)	Occurs when a control receives the input focus.
	OnExit (see page 12295)	Occurs when the input focus shifts away from one control to another.
	OnGetColors (see page 12296)	This is OnGetColors, a member of class TColorListBox.
	OnKeyDown (see page 12296)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 12299)	Occurs when a key is pressed.
	OnKeyUp (see page 12300)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 12302)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 12303)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 12305)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 12305)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 12306)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 12308)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 12310)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 12311)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 12259)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 12262)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 12263)	Determines whether the user can give focus to items by typing in the list.
	BevelEdges (see page 12263)	Specifies which edges of the control are beveled.
	BevelInner (see page 12264)	Specifies the cut of the inner bevel.
	BevelKind (see page 12264)	Specifies the control's bevel style.
	BevelOuter (see page 12264)	Specifies the cut of the outer bevel.
	BiDiMode (see page 12265)	Specifies the bi-directional mode for the control.
	Color (see page 12265)	Sets the background color of the TColorListBox (see page 12246) control.
	Constraints (see page 12266)	Specifies the size constraints for the control.
	Ctl3D (see page 12266)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DefaultColorColor (see page 12267)	This is DefaultColorColor, a member of class TColorListBox.
	Enabled (see page 12267)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 12267)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ItemHeight (see page 12268)	Specifies the height in pixels of an item in an owner-draw list box.

	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NoneColorColor (see page 12269)	This is NoneColorColor, a member of class TColorListBox.
	ParentBiDiMode (see page 12269)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 12270)	Determines where a control looks for its color information.
	ParentCtl3D (see page 12270)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 12270)	Determines where a control looks for its font information.
	ParentShowHint (see page 12271)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 12271)	Identifies the pop-up menu associated with the control.
	Selected (see page 12276)	Indicates whether a particular item is selected.
	ShowHint (see page 12278)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 12279)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabOrder (see page 12280)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 12281)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 12281)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.7.2 TColorListBox Properties

The properties of the TColorListBox class are listed here.

Published Properties

	Name	Description
	Align (see page 12259)	Determines how the control aligns within its container (parent control).
	Anchors (see page 12262)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 12263)	Determines whether the user can give focus to items by typing in the list.
	BevelEdges (see page 12263)	Specifies which edges of the control are beveled.
	BevelInner (see page 12264)	Specifies the cut of the inner bevel.
	BevelKind (see page 12264)	Specifies the control's bevel style.
	BevelOuter (see page 12264)	Specifies the cut of the outer bevel.
	BiDiMode (see page 12265)	Specifies the bi-directional mode for the control.
	Color (see page 12265)	Sets the background color of the TColorListBox (see page 12246) control.
	Constraints (see page 12266)	Specifies the size constraints for the control.
	Ctl3D (see page 12266)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColorColor (see page 12267)	This is DefaultColorColor, a member of class TColorListBox.
	Enabled (see page 12267)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 12267)	Controls the attributes of text written on or in the control.
	ItemHeight (see page 12268)	Specifies the height in pixels of an item in an owner-draw list box.
	NoneColorColor (see page 12269)	This is NoneColorColor, a member of class TColorListBox.
	ParentBiDiMode (see page 12269)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 12270)	Determines where a control looks for its color information.
	ParentCtl3D (see page 12270)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 12270)	Determines where a control looks for its font information.
	ParentShowHint (see page 12271)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 12271)	Identifies the pop-up menu associated with the control.

	Selected (see page 12276)	Indicates whether a particular item is selected.
	ShowHint (see page 12278)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 12279)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabOrder (see page 12280)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 12281)	Determines if the user can tab to a control.
	Visible (see page 12281)	Determines whether the component appears onscreen.

68.1.7.2.1 TColorListBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
```

```
    TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

    // Line up the directory list box on the left of the form.
    pdlb->Parent = Form1;
    pdlb->Align = alLeft;
    pdlb->Width = Form1->ClientWidth/3;

    // Now use the splitter to divide the directory pane from the file pane.
    ps->Parent = Form1;
    ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
    ps->Align = pdlb->Align; // Give same alignment as directory list.

    // Each pane must be at least one quarter of the form's width.
    ps->MinSize = Form1->ClientWidth/4;

    // Finally, create the last pane - a file list box.
    pflb->Parent = Form1;
    pflb->Align = alClient;
    pdlb->FileList = pflb; // Link the file list box to the directory list box.
  }
  /*
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  */

  #include <ComCtrls.hpp>
  #include <Controls.hpp>

  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    TProgressBar *ProgressBar = new TProgressBar(this);
    ProgressBar->Parent = this;
    ProgressBar->Align = alBottom;
    TButton *Button1 = new TButton(this);
    Button1->Parent = this;
  }
  /*
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  */
  #include <Comctrls.hpp>
  TPageControl* ppc;
  const int MAXTABS = 3;
  TTabSheet* pts[MAXTABS];
  const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
  int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
      ppc = new TPageControl(this);
      ppc->Parent = this;
      ppc->Align = alClient;
      for (int i=0;i<iTabTitles;i++)
    {
          pts[i] = new TTabSheet(this);
          pts[i]->PageControl = ppc;
          pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
          pts[i]->Caption = ppcTabTitles[i];
          pts[i]->Brush->Color = colorPalette[i];
    }
  }//---------------------------------------------------------------------------
  void __fastcall TForm1::FormDestroy(TObject *Sender)
  {
```

```
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
```

```
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

68.1.7.2.2 TColorListBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

68.1.7.2.3 TColorListBox.AutoComplete Property

Determines whether the user can give focus to items by typing in the list.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

Use AutoComplete to specify whether the list box automatically updates the ItemIndex property in response to user keystrokes. When AutoComplete is true, the item with focus changes to the first item that begins with the sequence of characters that the user typed. As the user types more characters, ItemIndex may change to a later item because the string to match becomes longer.

See Also

ItemIndex

68.1.7.2.4 TColorListBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

68.1.7.2.5 TColorListBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

68.1.7.2.6 TColorListBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

68.1.7.2.7 TColorListBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

68.1.7.2.8 TColorListBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

68.1.7.2.9 TColorListBox.Color Property

Sets the background color of the TColorListBox (see page 12246) control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

The color property is set to white, by default.

68.1.7.2.10 TColorListBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

68.1.7.2.11 TColorListBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

68.1.7.2.12 TColorListBox.DefaultColorColor Property

Pascal

```
property DefaultColorColor: TColor;
```

C++

```
__property TColor DefaultColorColor;
```

Description

This is DefaultColorColor, a member of class TColorListBox.

68.1.7.2.13 TColorListBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

68.1.7.2.14 TColorListBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name,

Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

68.1.7.2.15 TColorListBox.ItemHeight Property

Specifies the height in pixels of an item in an owner-draw list box.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Use ItemHeight to control the height of an item in a list box when the Style property is lbOwnerDrawFixed. The ItemHeight property can also affect the height of the list box itself. When the IntegralHeight property is true, the height of the list box is always an increment of the ItemHeight value.

If the Style property is lbStandard or lbOwnerDrawVariable, the value of ItemHeight is ignored.

See Also

DrawItem (see page 21328)

IntegralHeight (see page 21347)

Items (see page 21348)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

Style (see page 21354)

Using TDBListBox and TDBComboBox

68.1.7.2.16 TColorListBox.NoneColorColor Property

Pascal

```
property NoneColorColor: TColor;
```

C++

```
__property TColor NoneColorColor;
```

Description

This is NoneColorColor, a member of class TColorListBox.

68.1.7.2.17 TColorListBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

68.1.7.2.18 TColorListBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

68.1.7.2.19 TColorListBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

68.1.7.2.20 TColorListBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

68.1.7.2.21 TColorListBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

68.1.7.2.22 TColorListBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
```

```
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
```

```
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
```

```
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
```

```
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

68.1.7.2.23 TColorListBox.Selected Property

Indicates whether a particular item is selected.

Pascal

```
property Selected [Index: Integer]: Boolean;
```

C++

```
__property TColor Selected;__property Boolean Selected[int Index];
```

Description

Use Selected to query the state of a particular item in the list box. If the item specified by the Index parameter is selected in the list box, the value of the Selected property is true. If the specified item is not selected, Selected is false. Set Selected to change the currently selected item.

The Index parameter is the item referenced by its position in the list box, with the first item having an Index value of 0.

An item is selected when the user highlights it. More than one item in the list box can be selected by setting the MultiSelect property to true. In this case, the Selected item is the one that has focus when the list box has input focus.

C++ Examples:

```
/*
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
```

```
size of the deleted file.  Continue will continue listing
file sizes in a multiple selection that includes the deleted
file.  Break will stop.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
*/
#include <stdio.h>     // for FILE, fopen, fstat, fileno, fread and fclose
#include <sys\stat.h>  // for fstat and the stat struct

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < FileListBox1->Items->Count; i++)
  {
    if (FileListBox1->Selected[i])
    {
      if (!FileExists(FileListBox1->Items->Strings[i]))
      {
        MessageBeep(0);
        if (Application->MessageBox(
          AnsiString("File " + FileListBox1->Items->Strings[i] + "not found").c_str(),
          NULL,
          MB_OKCANCEL) == IDCANCEL)
          break;
        else
          continue;
      }
      FILE *F = fopen(FileListBox1->Items->Strings[i].c_str(),"r");
      struct stat statbuf;
      fstat(fileno(F), &statbuf);
      ListBox1->Items->Add(
          FileListBox1->Items->Strings[i] + ": " + IntToStr(statbuf.st_size));
      fclose(F);
    }
  }
}
```

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], 1);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode 1.', mtError, [mbOk], 0);
```

```
          Continue;
        end
        else
          FileClose(filehandle);

        AssignFile(F, FileListBox1.Items.Strings[i]);
        Reset(F, 1);
        ListBox1.Items.Add(
          FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
        CloseFile(F);
      end;
    finally
    { do something here }
    end;
  end;
end;
```

See Also

extendedselect (see page 21346)

Items (see page 21348)

MultiSelect

SelCount

68.1.7.2.24 TColorListBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;
```

```
  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

68.1.7.2.25 TColorListBox.Style Property

Determines whether the list box is standard or owner-draw and whether it is virtual.

Pascal

```
property Style: TListBoxStyle;
```

C++

```
__property TColorBoxStyle Style;__property TListBoxStyle Style;
```

Description

Use Style to specify whether the list box is a standard list box that displays text strings, or an owner-draw list box that can display graphical images. Owner-draw list boxes let you display items in some nonstandard way. In this case, you must write the code to paint items in the list box.

In addition, Style controls whether the list box is virtual, which means that items are supplied dynamically using event handlers. By default, Style is lbStandard, meaning that the list box is not virtual and it displays each item as a string.

See Also

DrawItem (see page 21328)

ItemHeight (see page 21347)

Items (see page 21348)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

OnData (see page 21357)

OnDataFind (see page 21357)

OnDataObject (see page 21358)

68.1.7.2.26 TColorListBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore

cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

68.1.7.2.27 TColorListBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

68.1.7.2.28 TColorListBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
```

```
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

68.1.7.3 TColorListBox Events

The events of the TColorListBox class are listed here.

Published Events

	Name	Description
	OnClick (see page 12283)	Occurs when the user clicks the control.
	OnContextPopup (see page 12285)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 12286)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 12286)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 12288)	Occurs when the user drags an object over a control.
	OnEndDock (see page 12291)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 12291)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 12292)	Occurs when a control receives the input focus.
	OnExit (see page 12295)	Occurs when the input focus shifts away from one control to another.
	OnGetColors (see page 12296)	This is OnGetColors, a member of class TColorListBox.
	OnKeyDown (see page 12296)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 12299)	Occurs when a key is pressed.
	OnKeyUp (see page 12300)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 12302)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 12303)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 12305)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 12305)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 12306)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 12308)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 12310)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 12311)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

68.1.7.3.1 TColorListBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.

- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
```

```
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

68.1.7.3.2 TColorListBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the

default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

68.1.7.3.3 TColorListBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

68.1.7.3.4 TColorListBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
```

```
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

68.1.7.3.5 TColorListBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true

to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.7.3.6 TColorListBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.7.3.7 TColorListBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
```

```
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

68.1.7.3.8 TColorListBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter

event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

68.1.7.3.9 TColorListBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
```

```
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

68.1.7.3.10 TColorListBox.OnGetColors Event

Pascal

```
property OnGetColors: TLBGetColorsEvent;
```

C++

```
__property TLBGetColorsEvent OnGetColors;
```

Description

This is OnGetColors, a member of class TColorListBox.

68.1.7.3.11 TColorListBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
```

```
      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

68.1.7.3.12 TColorListBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys

that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

68.1.7.3.13 TColorListBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
```

```
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

68.1.7.3.14 TColorListBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the

OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.7.3.15 TColorListBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
```

```
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
```

```
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.7.3.16 TColorListBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

68.1.7.3.17 TColorListBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

68.1.7.3.18 TColorListBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
```

```
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
```

```
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.7.3.19 TColorListBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
```

```
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
```

```
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

68.1.7.3.20 TColorListBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

68.1.7.3.21 TColorListBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

68.1.8 TColumnCollection Class

Controls (see page 5711) the **Collection Editor** for columns within a grid panel.

Class Hierarchy

System::TObject → Classes.TPersistent → Classes.TCollection → Classes.TOwnedCollection → ExtCtrls.TCellCollection → ExtCtrls.TColumnCollection

Pascal

```
TColumnCollection = class(TCellCollection);
```

C++

```
class TColumnCollection : public TCellCollection;
```

File

ExtCtrls

Description

TColumnCollection controls the **Collection Editor** for columns within a grid panel.

See Also

Collection Editor

68.1.8.1 TColumnCollection Members

The following tables list the members exposed by TColumnCollection.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
V	GetAttr (see page 12174)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
V	GetAttrCount (see page 12175)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItem (see page 12175)	Returns a specified item in the collection.
V	GetItemAttr (see page 12314)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2915)	Returns the owner of an object.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Notify (see page 12315)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
	SetItem (see page 12176)	This is SetItem, a member of class TCellCollection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.
V	Update (see page 12176)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 12314)	Adds a column to the grid panel.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 12314)	This is Create, a member of class TColumnCollection.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
	Owner (see page 12176)	Returns the Owner of the cell collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.

R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 2751)	Lists the items in the collection.
	Items (see page 12177)	Lists the items of the cell collection.

68.1.8.2 TColumnCollection.Create Constructor

Pascal

```
constructor Create(AOwner: TPersistent);
```

C++

```
__fastcall TColumnCollection(TPersistent * AOwner);
```

Description

This is Create, a member of class TColumnCollection.

68.1.8.3 TColumnCollection Methods

The methods of the TColumnCollection class are listed here.

Protected Methods

	Name	Description
V	GetItemAttr (see page 12314)	Returns the value of a custom attribute assigned to one of the collection's items.
V	Notify (see page 12315)	Responds when items are added to or removed from the collection.

Public Methods

	Name	Description
	Add (see page 12314)	Adds a column to the grid panel.

68.1.8.3.1 TColumnCollection.Add Method

Adds a column to the grid panel.

Pascal

```
function Add: TColumnItem;
```

C++

```
__fastcall TColumnItem Add();
```

Description

Use the Add method to add a new column to the grid panel.

68.1.8.3.2 TColumnCollection.GetItemAttr Method

Returns the value of a custom attribute assigned to one of the collection's items.

Pascal

```
function GetItemAttr(Index: Integer; ItemIndex: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetItemAttr(int Index, int ItemIndex);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetItemAttr returns the value of one of these attributes for a specified item in the collection.

Index identifies which attribute's value is desired. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

ItemIndex identifies the item whose attribute value is desired. This is an index into the Items property array.

As implemented in TCollection, GetItemAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttr (see page 2743)

GetAttrCount (see page 2744)

68.1.8.3.3 TColumnCollection.Notify Method

Responds when items are added to or removed from the collection.

Pascal

```
procedure Notify(Item: TCollectionItem; Action: TCollectionNotification); override;
```

C++

```
virtual __fastcall Notify(TCollectionItem Item, TCollectionNotification Action);
```

Description

Notify is called automatically when the items in the collection change.

Item is the item that was just added to or that is about to be removed from the collection.

Action indicates whether item was added, is about to be removed, or is about to be deleted:

As implemented in TCollection, Notify calls Added when Action is cnAdded and calls Deleting when Action is cnDeleting. TCollection ignores the cnExtracting action. Descendant classes can override Notify to modify this behavior.

See Also

Added (see page 2736)

Deleting (see page 2741)

68.1.9 TColumnItem Class

Class Hierarchy

Pascal

```
TColumnItem = class(TCellItem);
```

C++

```
class TColumnItem : public TCellItem;
```

File

ExtCtrls

Description

This is class ExtCtrls.TColumnItem.

68.1.10 TControlBar Class

TControlBar manages the layout of toolbar components.

Class Hierarchy

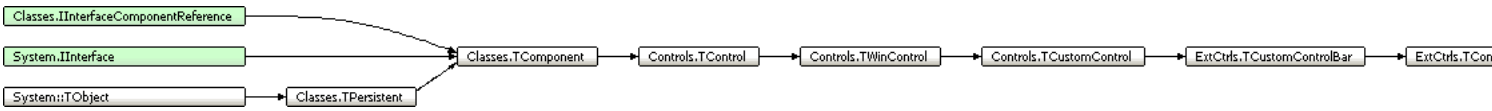

Pascal

```
TControlBar = class(TCustomControlBar);
```

C++

```
class TControlBar : public TCustomControlBar;
```

File

ExtCtrls

Description

Use TControlBar as a docking site for toolbar components. Control bars contain child controls (usually TToolBar objects) that can be moved and resized independently. As with a cool bar, each child control resides on its own individual band that the user positions by dragging its sizing grip.

Tip: When docking TToolBar objects to a control bar, you may find it useful to use TToolDockObject.

Note: If you want to use TActionToolBar on a TControlBar component, you should set the AutoSize (see page 12333) property to **False**. When the AutoSize (see page 12333) property is set to True (default), the TActionToolBar control wraps its controls and expands itself in size, accordingly. This does not allow the TActionToolBar component to display properly on a TControlBar component. When the AutoSize (see page 12333) property is set to **False**, the TActionToolBar hides items on the end of the bar and displays a dropdown button that provides access to those items. The control displays in the proper size.

See Also

TCoolBar (see page 3603)

TToolBar (see page 4994)

68.1.10.1 TControlBar Members

The following tables list the members exposed by TControlBar.

Protected Events

	Name	Description
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.

	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 12459)	Aligns controls in the control bar.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanAutoSize (see page 12459)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.

	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 12460)	Sets the control bar's window flags.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
V	DoBandMove (see page 12460)	Generates an OnBandMove (see page 12473) event.
V	DoBandPaint (see page 12460)	This is DoBandPaint, a member of class TCustomControlBar.
	DoBeginBandMove (see page 12461)	This is DoBeginBandMove, a member of class TCustomControlBar.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockOver (see page 12461)	Generates an OnDockOver event and ensures proper docking of floating controls.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndBandMove (see page 12461)	This is DoEndBandMove, a member of class TCustomControlBar.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoPaletteChange (see page 12461)	Prepares the control bar to realize a new palette.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.

	DragControl (see page 12462)	Starts a drag operation on a control band.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetControlInfo (see page 12463)	Generates an OnBandInfo (see page 12473) event.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPalette (see page 12463)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetSiteInfo (see page 12464)	Generates an OnGetSiteInfo event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HitTest (see page 12464)	Returns the control at the specified coordinates.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.

	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
V	MouseDown (see page 12465)	Handles mouse button clicks.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
V	MouseMove (see page 12465)	Handles mouse movements.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	MouseUp (see page 12465)	Handles releasing of a mouse button.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 5920)	Renders the image of a custom control.
V	Paint (see page 12466)	Paints the control bar.
V	PaintControlFrame (see page 12466)	Paints a control band's frame.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.

	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.

	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 5918)	Creates an instance of TCustomControl (see page 5908).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 12457)	Creates and initializes an instance of TCustomControlBar (see page 12446).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 12458)	Destroys the TCustomControlBar (see page 12446) instance.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	FlipChildren (see page 12462)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.

	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	StickControls (see page 12467)	Places controls in their final positions after a control is moved in or dropped on the control bar.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.

	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	Canvas (see page 12336)	Specifies the TCanvas object that presents a drawing surface for the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 12357)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 12357)	Occurs when an object with custom alignment is aligned.
	OnBandDrag (see page 12358)	Occurs when the user starts a drag operation on a specified control.
	OnBandInfo (see page 12358)	Occurs when the control bar adjusts the docking position of a band.
	OnBandMove (see page 12359)	This is OnBandMove, a member of class TControlBar.
	OnBandPaint (see page 12359)	This is OnBandPaint, a member of class TControlBar.
	OnBeginBandMove (see page 12359)	This is OnBeginBandMove, a member of class TControlBar.
	OnCanResize (see page 12359)	Occurs when an attempt is made to resize the control.
	OnClick (see page 12360)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 12362)	Adjust resize constraints.
	OnContextPopup (see page 12362)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 12363)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 12363)	Occurs when another control is docked to the control.
	OnDockOver (see page 12365)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 12367)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 12370)	Occurs when the user drags an object over a control.
	OnEndBandMove (see page 12372)	This is OnEndBandMove, a member of class TControlBar.
	OnEndDock (see page 12372)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 12373)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 12374)	Occurs when a control receives the input focus.
	OnExit (see page 12376)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 12378)	Returns the control's docking information.
	OnMouseActivate (see page 12380)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 12381)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 12383)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 12383)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 12384)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 12386)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 12388)	Occurs when the control bar is painted.
	OnResize (see page 12389)	Occurs immediately after the control is resized.
	OnStartDock (see page 12389)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 12390)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 12390)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 12328)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 12332)	Specifies how the control is anchored to its parent.
	AutoDock (see page 12333)	Determines whether the control bar temporarily docks drag clients to provide visual feedback.
	AutoDrag (see page 12333)	Specifies whether control bands can be undocked by dragging.
	AutoSize (see page 12333)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 12334)	Specifies which edges of the control are beveled.

	BevelInner (see page 12334)	Specifies the cut of the inner bevel.
	BevelKind (see page 12335)	Specifies the control's bevel style.
	BevelOuter (see page 12335)	Specifies the cut of the outer bevel.
	BevelWidth (see page 12336)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 12336)	Specifies the width of the control's border.
	Color (see page 12338)	Specifies the background color of the control.
	Constraints (see page 12339)	Specifies the size constraints for the control.
	CornerEdge (see page 12340)	The size of the cornering for the edge of the bands.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DockSite (see page 12340)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 12341)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 12341)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 12342)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DrawingStyle (see page 12343)	This is DrawingStyle, a member of class TControlBar.
	Enabled (see page 12343)	Controls whether the control responds to mouse, keyboard, and timer events.
	GradientDirection (see page 12344)	Horizontal or Vertical.
	GradientEndColor (see page 12344)	Stores the color that the gradient is painted with when it ends.
	GradientStartColor (see page 12344)	Stores the color that the gradient is painted when it starts.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBackground (see page 12345)	Determines whether control uses parent's theme background.
	ParentColor (see page 12345)	Determines where a control looks for its color information.
	ParentCtl3D (see page 12345)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 12346)	Determines where a control looks for its font information.
	ParentShowHint (see page 12346)	Determines where a control looks to find out if its Help Hint should be shown.
	Picture (see page 12347)	This is Picture, a member of class TControlBar.
	PopupMenu (see page 12347)	Identifies the pop-up menu associated with the control.
	RowSize (see page 12351)	This is RowSize, a member of class TControlBar.
	RowSnap (see page 12352)	This is RowSnap, a member of class TControlBar.
	ShowHint (see page 12352)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 12353)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 12354)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 12354)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.10.2 TControlBar Properties

The properties of the TControlBar class are listed here.

Public Properties

	Name	Description
	Canvas (see page 12336)	Specifies the TCanvas object that presents a drawing surface for the control.

Published Properties

	Name	Description
	Align (see page 12328)	Determines how the control aligns within its container (parent control).
	Anchors (see page 12332)	Specifies how the control is anchored to its parent.
	AutoDock (see page 12333)	Determines whether the control bar temporarily docks drag clients to provide visual feedback.
	AutoDrag (see page 12333)	Specifies whether control bands can be undocked by dragging.
	AutoSize (see page 12333)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 12334)	Specifies which edges of the control are beveled.
	BevelInner (see page 12334)	Specifies the cut of the inner bevel.
	BevelKind (see page 12335)	Specifies the control's bevel style.
	BevelOuter (see page 12335)	Specifies the cut of the outer bevel.
	BevelWidth (see page 12336)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 12336)	Specifies the width of the control's border.
	Color (see page 12338)	Specifies the background color of the control.
	Constraints (see page 12339)	Specifies the size constraints for the control.
	CornerEdge (see page 12340)	The size of the cornering for the edge of the bands.
	DockSite (see page 12340)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 12341)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 12341)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 12342)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DrawingStyle (see page 12343)	This is DrawingStyle, a member of class TControlBar.
	Enabled (see page 12343)	Controls whether the control responds to mouse, keyboard, and timer events.
	GradientDirection (see page 12344)	Horizontal or Vertical.
	GradientEndColor (see page 12344)	Stores the color that the gradient is painted with when it ends.
	GradientStartColor (see page 12344)	Stores the color that the gradient is painted when it starts.
	ParentBackground (see page 12345)	Determines whether control uses parent's theme background.
	ParentColor (see page 12345)	Determines where a control looks for its color information.
	ParentCtl3D (see page 12345)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 12346)	Determines where a control looks for its font information.
	ParentShowHint (see page 12346)	Determines where a control looks to find out if its Help Hint should be shown.
	Picture (see page 12347)	This is Picture, a member of class TControlBar.
	PopupMenu (see page 12347)	Identifies the pop-up menu associated with the control.
	RowSize (see page 12351)	This is RowSize, a member of class TControlBar.
	RowSnap (see page 12352)	This is RowSnap, a member of class TControlBar.
	ShowHint (see page 12352)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 12353)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 12354)	Determines if the user can tab to a control.
	Visible (see page 12354)	Determines whether the component appears onscreen.

68.1.10.2.1 TControlBar.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
```

```
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
```

```
    Split: TSplitter;
    Files: TFileListBox;
    Dirs: TDirectoryListBox;
  begin
    { first add a directory list box to the form }
    Dirs := TDirectoryListBox.Create(Form1);
    { line it up on the left of the form }
    Dirs.Parent := Form1;
    Dirs.Align := alLeft;
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
```

```
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
    end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

68.1.10.2.2 TControlBar.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

68.1.10.2.3 TControlBar.AutoDock Property

Determines whether the control bar temporarily docks drag clients to provide visual feedback.

Pascal

```
property AutoDock: Boolean;
```

C++

```
__property Boolean AutoDock;
```

Description

If AutoDock is true (the default), when a dockable object is dragged over the control bar, the control bar temporarily docks the object so that the user can see the impact of the drag operation. That is, the user can see the dock position and how other docked controls will be repositioned to make room. When the user releases the mouse, the dragged object is docked at the current spot, and any temporary dockings are canceled. However, if the user cancels the drag-and-dock operation, the dragged object reverts to its last temporary dock site rather than to its location at the start of the drag operation.

If AutoDock is false, the control bar does not temporarily dock dragged objects. Instead, the user only sees the dragging rectangle as it passes over the control bar, with no feedback as to the adjustments the control bar would make if the dragged object were released.

See Also

DockClients (see page 6201)

DockSite (see page 6204)

OnDockOver (see page 6216)

68.1.10.2.4 TControlBar.AutoDrag Property

Specifies whether control bands can be undocked by dragging.

Pascal

```
property AutoDrag: Boolean;
```

C++

```
__property Boolean AutoDrag;
```

Description

Use AutoDrag to specify whether control bands are automatically undocked when the user drags them. If AutoDrag is true, control bands (TToolBar objects) are undocked when dragged off the control bar. Otherwise, control bands are kept within the control bar.

See Also

DragMode (see page 5819)

OnBandDrag (see page 12472)

onbandmove (see page 12473)

68.1.10.2.5 TControlBar.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints (see page 5812)

CanAutoSize (see page 5741)

68.1.10.2.6 TControlBar.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

68.1.10.2.7 TControlBar.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

68.1.10.2.8 TControlBar.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

68.1.10.2.9 TControlBar.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

68.1.10.2.10 TControlBar.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

68.1.10.2.11 TControlBar.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics or text drawn by the control is clipped to the area within the border.

See Also

ClientRect

BevelOuter (see page 6194)

68.1.10.2.12 TControlBar.Canvas Property

Specifies the TCanvas object that presents a drawing surface for the control.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control. Canvas encapsulates a Windows device context, providing all the tools and methods needed for drawing and painting.

Canvas is a protected property that is usually redeclared as public in descendants of TCustomControl.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas,Rect.Left,Rect.Top,index);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(index) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TStringGrid *myStringGrid;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TStringGrid")
  {
    myStringGrid = (TStringGrid *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myStringGrid->Canvas->MoveTo(myStringGrid->ClientRect.Left, myStringGrid->ClientRect.Top);
    myStringGrid->Canvas->LineTo(myStringGrid->ClientRect.Right,
myStringGrid->ClientRect.Bottom);
    myStringGrid = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
```

```
    Rect: TRect; State: TGridDrawState);
  var
    index: Integer;
  begin
    index := ARow * DrawGrid1.ColCount + ACol;
    DrawGrid1.Canvas.Brush.Color := clWhite;
    DrawGrid1.Canvas.FillRect(Rect);
    if (gdFocused in State) then
    begin
      DrawGrid1.Canvas.DrawFocusRect(Rect);
      Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
    end;
    ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  end;

  procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  begin
    CanSelect:= True;
  end;
  {
  The following code uses ClientRect to find and draw a line
  from the top left to the bottom right of the current
  control.  Select a TControl in the ListBox list.  This
  example assumes that the current object is a descendent of
  TControl and has a public Canvas property.
  }
    if (ListBox1.Items.Objects[Index] is TStringGrid) then
    begin
      myStringGrid:= (ListBox1.Items.Objects[Index] as TStringGrid);
      with myStringGrid.ClientRect do
      begin // make sure we are using the TControl's canvas and not the form's!
        myStringGrid.Canvas.MoveTo(Left, Top);
        myStringGrid.Canvas.LineTo(Right, Bottom);
      end;
    myStringGrid:= nil;
    end;
```

See Also

Paint (see page 5920)

68.1.10.2.13 TControlBar.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
```

```
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

68.1.10.2.14 TControlBar.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

68.1.10.2.15 TControlBar.CornerEdge Property

The size of the cornering for the edge of the bands.

Pascal

```
property CornerEdge: TCornerEdge;
```

C++

```
__property TCornerEdge CornerEdge;
```

Description

This property stores the size of the cornering for the edge of the bands.

68.1.10.2.16 TControlBar.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite: Boolean;
```

C++

```
__property Boolean DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
```

```
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

68.1.10.2.17 TControlBar.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

68.1.10.2.18 TControlBar.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

68.1.10.2.19 TControlBar.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
```

```
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

68.1.10.2.20 TControlBar.DrawingStyle Property

Pascal

```
property DrawingStyle: TBandDrawingStyle;
```

C++

```
__property TBandDrawingStyle DrawingStyle;
```

Description

This is DrawingStyle, a member of class TControlBar.

68.1.10.2.21 TControlBar.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

68.1.10.2.22 TControlBar.GradientDirection Property

Horizontal or Vertical.

Pascal

```
property GradientDirection: TGradientDirection;
```

C++

```
__property TGradientDirection GradientDirection;
```

Description

Stores the direction of the gradient as Horizontal or Vertical.

68.1.10.2.23 TControlBar.GradientEndColor Property

Stores the color that the gradient is painted with when it ends.

Pascal

```
property GradientEndColor: TColor;
```

C++

```
__property TColor GradientEndColor;
```

Description

Use this property to store the color that you want to use to end your drawing. (This blends with the start color.)

68.1.10.2.24 TControlBar.GradientStartColor Property

Stores the color that the gradient is painted when it starts.

Pascal

```
property GradientStartColor: TColor;
```

C++

```
__property TColor GradientStartColor;
```

Description

Use this property to store the color that you want to use to initialize the drawing.

68.1.10.2.25 TControlBar.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground: Boolean;
```

C++

```
__property Boolean ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

68.1.10.2.26 TControlBar.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

68.1.10.2.27 TControlBar.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

68.1.10.2.28 TControlBar.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

68.1.10.2.29 TControlBar.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

68.1.10.2.30 TControlBar.Picture Property

Pascal

```
property Picture: TPicture;
```

C++

```
__property TPicture Picture;
```

Description

This is Picture, a member of class TControlBar.

68.1.10.2.31 TControlBar.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
```

```
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->CopyToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Cut1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      const int num_items = 4;
      for (int index = 0; index < num_items; ++index)
      {
          TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
          PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
          NewItem->Caption = "Menu Item " + IntToStr(index);
          NewItem->Tag = index;
          NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
          TNotifyEvent();
      }
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
```

```
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
```

```
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
```

```
procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

68.1.10.2.32 TControlBar.RowSize Property

Pascal

```
property RowSize: TRowSize;
```

C++

```
__property TRowSize RowSize;
```

Description

This is RowSize, a member of class TControlBar.

68.1.10.2.33 TControlBar.RowSnap Property

Pascal

```
property RowSnap: Boolean;
```

C++

```
__property Boolean RowSnap;
```

Description

This is RowSnap, a member of class TControlBar.

68.1.10.2.34 TControlBar.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
```

```
    {
      strcpy(string, "Item");
      itoa(i, index, 10);
      strcat(string, index);
      ListBox1->Items->Add(string);
    }
    ListBox1->Hint = "Select an item";
    ListBox1->ShowHint = true;
    Application->ShowHint = true;
  }
  /*
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  */
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

68.1.10.2.35 TControlBar.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

68.1.10.2.36 TControlBar.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

68.1.10.2.37 TControlBar.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
```

```
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

68.1.10.3 TControlBar Events

The events of the TControlBar class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 12357)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 12357)	Occurs when an object with custom alignment is aligned.
	OnBandDrag (see page 12358)	Occurs when the user starts a drag operation on a specified control.
	OnBandInfo (see page 12358)	Occurs when the control bar adjusts the docking position of a band.
	OnBandMove (see page 12359)	This is OnBandMove, a member of class TControlBar.
	OnBandPaint (see page 12359)	This is OnBandPaint, a member of class TControlBar.
	OnBeginBandMove (see page 12359)	This is OnBeginBandMove, a member of class TControlBar.
	OnCanResize (see page 12359)	Occurs when an attempt is made to resize the control.
	OnClick (see page 12360)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 12362)	Adjust resize constraints.
	OnContextPopup (see page 12362)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 12363)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 12363)	Occurs when another control is docked to the control.
	OnDockOver (see page 12365)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 12367)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 12370)	Occurs when the user drags an object over a control.
	OnEndBandMove (see page 12372)	This is OnEndBandMove, a member of class TControlBar.
	OnEndDock (see page 12372)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 12373)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 12374)	Occurs when a control receives the input focus.
	OnExit (see page 12376)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 12378)	Returns the control's docking information.
	OnMouseActivate (see page 12380)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 12381)	Occurs when the user presses a mouse button with the mouse pointer over a control.

	OnMouseEnter (see page 12383)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 12383)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 12384)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 12386)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 12388)	Occurs when the control bar is painted.
	OnResize (see page 12389)	Occurs immediately after the control is resized.
	OnStartDock (see page 12389)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 12390)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 12390)	Occurs when the application tries to undock a control that is docked to the windowed control.

68.1.10.3.1 TControlBar.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore: TAlignInsertBeforeEvent;
```

C++

```
__property TAlignInsertBeforeEvent OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore (see page 6137)

Align (see page 5798)

TAlignInsertBeforeEvent (see page 6293)

68.1.10.3.2 TControlBar.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition: TAlignPositionEvent;
```

C++

```
__property TAlignPositionEvent OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition (see page 6138)

Align (see page 5798)

Anchors (see page 5802)

TAlignInfo (see page 6269)

TAlignPositionEvent (see page 6293)

68.1.10.3.3 TControlBar.OnBandDrag Event

Occurs when the user starts a drag operation on a specified control.

Pascal

```
property OnBandDrag: TBandDragEvent;
```

C++

```
__property TBandDragEvent OnBandDrag;
```

Description

Use OnBandDrag to conditionally allow the user to start dragging a control in the control bar.

The event handler indicates the control that the user is trying to drag. Change its Drag parameter to false to prevent the drag operation. If Drag remains true on exit from the event handler, the drag operation proceeds.

See Also

OnStartDock (see page 5871)

onbandmove (see page 12473)

68.1.10.3.4 TControlBar.OnBandInfo Event

Occurs when the control bar adjusts the docking position of a band.

Pascal

```
property OnBandInfo: TBandInfoEvent;
```

C++

```
__property TBandInfoEvent OnBandInfo;
```

Description

Write code in the OnBandInfo event handler to provide information to the control bar that it can use when arranging its bands.

The OnBandInfo event handler has several parameters that can be used to return information about how the band should be positioned, including its size, inset, and number of rows.

See Also

TBDockHeight (see page 5842)

onbandmove (see page 12473)

LRDockWidth (see page 5831)

Height (see page 5825)

Width (see page 5849)

68.1.10.3.5 TControlBar.OnBandMove Event

Pascal

```
property OnBandMove: TBandMoveEvent;
```

C++

```
__property TBandMoveEvent OnBandMove;
```

Description

This is OnBandMove, a member of class TControlBar.

68.1.10.3.6 TControlBar.OnBandPaint Event

Pascal

```
property OnBandPaint: TBandPaintEvent;
```

C++

```
__property TBandPaintEvent OnBandPaint;
```

Description

This is OnBandPaint, a member of class TControlBar.

68.1.10.3.7 TControlBar.OnBeginBandMove Event

Pascal

```
property OnBeginBandMove: TBeginBandMoveEvent;
```

C++

```
__property TBeginBandMoveEvent OnBeginBandMove;
```

Description

This is OnBeginBandMove, a member of class TControlBar.

68.1.10.3.8 TControlBar.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize: TCanResizeEvent;
```

C++

```
__property TCanResizeEvent OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize (see page 5853)

SetBounds (see page 5783)

OnResize (see page 5870)

68.1.10.3.9 TControlBar.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
```

```
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

68.1.10.3.10 TControlBar.OnConstrainedResize Event

Adjust resize constraints.

Pascal

```
property OnConstrainedResize: TConstrainedResizeEvent;
```

C++

```
__property TConstrainedResizeEvent OnConstrainedResize;
```

Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

Note: The OnConstrainedResize handler is called immediately after the OnCanResize handler.

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

Constraints (see page 5812)

OnResize (see page 5870)

CanAutoSize (see page 5741)

68.1.10.3.11 TControlBar.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

68.1.10.3.12 TControlBar.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

68.1.10.3.13 TControlBar.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop: TDockDropEvent;
```

C++

```
__property TDockDropEvent OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
```

```
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockOver (see page 6216)

68.1.10.3.14 TControlBar.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver: TDockOverEvent;
```

C++

```
__property TDockOverEvent OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
```

```
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnGetSiteInfo (see page 6223)

TDragDockObject (see page 6009)

68.1.10.3.15 TControlBar.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

68.1.10.3.16 TControlBar.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.10.3.17 TControlBar.OnEndBandMove Event

Pascal

```
property OnEndBandMove: TEndBandMoveEvent;
```

C++

```
__property TEndBandMoveEvent OnEndBandMove;
```

Description

This is OnEndBandMove, a member of class TControlBar.

68.1.10.3.18 TControlBar.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.10.3.19 TControlBar.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
```

```
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

68.1.10.3.20 TControlBar.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}
```

```
void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
```

```
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

68.1.10.3.21 TControlBar.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit

event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
```

```
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

68.1.10.3.22 TControlBar.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo: TGetSiteInfoEvent;
```

C++

```
__property TGetSiteInfoEvent OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}
```

```
void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;
```

```
procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

TDragDockObject (see page 6009)

68.1.10.3.23 TControlBar.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done

at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.10.3.24 TControlBar.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
```

```
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
```

```
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.10.3.25 TControlBar.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

68.1.10.3.26 TControlBar.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse

outside of a control when the parent top-level form is not active.

68.1.10.3.27 TControlBar.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
```

```
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
```

```
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.10.3.28 TControlBar.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;
```

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

68.1.10.3.29 TControlBar.OnPaint Event

Occurs when the control bar is painted.

Pascal

```
property OnPaint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPaint;
```

Description

Write an OnPaint event handler to change the way the control bar is rendered. When a control bar receives a Windows paint message, it first draws the background picture, if any, and then generates an OnPaint event. After the OnPaint event exits, the control bar generates an OnBandPaint event for each band, finally rendering the grabbers and frames for each band.

Use the Canvas property to draw the control bar's image. Use the ClientRect property to obtain the dimensions on which to draw.

If the custom drawing is intended to change the appearance of individual bands, use the OnBandPaint event instead.

See Also

onbandpaint (see page 12473)

Canvas (see page 5921)

ClientRect

picture (see page 12471)

68.1.10.3.30 TControlBar.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

68.1.10.3.31 TControlBar.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is

automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

68.1.10.3.32 TControlBar.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

68.1.10.3.33 TControlBar.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock: TUnDockEvent;
```

C++

```
__property TUnDockEvent OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
```

```
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
```

```
    var Accept: Boolean);
  var
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

68.1.11 TControlCollection Class

Class Hierarchy

Pascal

```
TControlCollection = class(TOwnedCollection);
```

C++

```
class TControlCollection : public TOwnedCollection;
```

File

ExtCtrls

Description

This is class ExtCtrls.TControlCollection.

68.1.11.1 TControlCollection Members

The following tables list the members exposed by TControlCollection.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetControl (see page 12397)	This is GetControl, a member of class TControlCollection.
	GetControlItem (see page 12397)	This is GetControlItem, a member of class TControlCollection.
	GetItem (see page 12397)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2915)	Returns the owner of an object.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetControl (see page 12398)	This is SetControl, a member of class TControlCollection.
	SetItem (see page 12398)	This is SetItem, a member of class TControlCollection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 12398)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 12396)	Creates a new TCollectionItem instance and adds it to the Items array.
	AddControl (see page 12396)	This is AddControl, a member of class TControlCollection.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 12395)	This is Create, a member of class TControlCollection.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	IndexOf (see page 12397)	This is IndexOf, a member of class TControlCollection.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 12398)	Returns the Owner of the collection.
	RemoveControl (see page 12398)	This is RemoveControl, a member of class TControlCollection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	ControlItems (see page 12399)	This is ControlItems, a member of class TControlCollection.
	Controls (see page 12399)	This is Controls, a member of class TControlCollection.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 12400)	Lists the items in the collection.

68.1.11.2 TControlCollection.Create Constructor

Pascal

```
constructor Create(AOwner: TPersistent);
```

C++

```
__fastcall TControlCollection(TPersistent * AOwner);
```

Description

This is Create, a member of class TControlCollection.

68.1.11.3 TControlCollection Methods

The methods of the TControlCollection class are listed here.

Protected Methods

	Name	Description
	GetControl (see page 12397)	This is GetControl, a member of class TControlCollection.
	GetControlItem (see page 12397)	This is GetControlItem, a member of class TControlCollection.
	GetItem (see page 12397)	Returns a specified item in the collection.
	SetControl (see page 12398)	This is SetControl, a member of class TControlCollection.
	SetItem (see page 12398)	This is SetItem, a member of class TControlCollection.
V	Update (see page 12398)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 12396)	Creates a new TCollectionItem instance and adds it to the Items array.
	AddControl (see page 12396)	This is AddControl, a member of class TControlCollection.
	IndexOf (see page 12397)	This is IndexOf, a member of class TControlCollection.
	Owner (see page 12398)	Returns the Owner of the collection.
	RemoveControl (see page 12398)	This is RemoveControl, a member of class TControlCollection.

68.1.11.3.1 TControlCollection.Add Method

Creates a new TCollectionItem instance and adds it to the Items array.

Pascal

```
function Add: TControlItem;
```

C++

```
__fastcall TControlItem Add();
```

Description

Call Add to create an item in the collection. The new item is placed at the end of the Items array.

Add returns the new collection item.

See Also

Items (see page 2751)

Insert (see page 2746)

Create (see page 2734)

Delete (see page 2741)

68.1.11.3.2 TControlCollection.AddControl Method

Pascal

```
procedure AddControl(AControl: TControl; AColumn: Integer = -1; ARow: Integer = -1);
```

C++

```
__fastcall AddControl(TControl * AControl, int AColumn = -1, int ARow = -1);
```

Description

This is AddControl, a member of class TControlCollection.

68.1.11.3.3 TControlCollection.GetControl Method

Pascal

```
function GetControl(AColumn: Integer; ARow: Integer): TControl;
```

C++

```
__fastcall TControl * GetControl(int AColumn, int ARow);
```

Description

This is GetControl, a member of class TControlCollection.

68.1.11.3.4 TControlCollection.GetControlItem Method

Pascal

```
function GetControlItem(AColumn: Integer; ARow: Integer): TControlItem;
```

C++

```
__fastcall TControlItem GetControlItem(int AColumn, int ARow);
```

Description

This is GetControlItem, a member of class TControlCollection.

68.1.11.3.5 TControlCollection.GetItem Method

Returns a specified item in the collection.

Pascal

```
function GetItem(Index: Integer): TControlItem;
```

C++

```
__fastcall TControlItem GetItem(int Index);
```

Description

GetItem is the protected read implementation of the Items property.

See Also

Items (see page 2751)

68.1.11.3.6 TControlCollection.IndexOf Method

Pascal

```
function IndexOf(AControl: TControl): Integer;
```

C++

```
__fastcall int IndexOf(TControl * AControl);
```

Description

This is IndexOf, a member of class TControlCollection.

68.1.11.3.7 TControlCollection.Owner Method

Returns the Owner of the collection.

Pascal

```
function Owner: TCustomGridPanel;
```

C++

```
__fastcall TCustomGridPanel Owner();
```

Description

Call Owner to obtain a reference to the object that owns this collection. Typically, the owner uses the collection to implement one of its properties.

68.1.11.3.8 TControlCollection.RemoveControl Method

Pascal

```
procedure RemoveControl(AControl: TControl);
```

C++

```
__fastcall RemoveControl(TControl * AControl);
```

Description

This is RemoveControl, a member of class TControlCollection.

68.1.11.3.9 TControlCollection.SetControl Method

Pascal

```
procedure SetControl(AColumn: Integer; ARow: Integer; Value: TControl);
```

C++

```
__fastcall SetControl(int AColumn, int ARow, TControl * Value);
```

Description

This is SetControl, a member of class TControlCollection.

68.1.11.3.10 TControlCollection.SetItem Method

Pascal

```
procedure SetItem(Index: Integer; Value: TControlItem);
```

C++

```
__fastcall SetItem(int Index, TControlItem Value);
```

Description

This is SetItem, a member of class TControlCollection.

68.1.11.3.11 TControlCollection.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed (see page 2740)

Changed (see page 2758)

68.1.11.4 TControlCollection Properties

The properties of the TControlCollection class are listed here.

Public Properties

	Name	Description
R	ControlItems (see page 12399)	This is ControlItems, a member of class TControlCollection.
	Controls (see page 12399)	This is Controls, a member of class TControlCollection.
	Items (see page 12400)	Lists the items in the collection.

68.1.11.4.1 TControlCollection.ControlItems Property

Pascal

```
property ControlItems [AColumn, ARow: Integer]: TControlItem;
```

C++

```
__property TControlItem ControlItems[int AColumn, ARow];
```

Description

This is ControlItems, a member of class TControlCollection.

68.1.11.4.2 TControlCollection.Controls Property

Pascal

```
property Controls [AColumn, ARow: Integer]: TControl;
```

C++

```
__property TControl * Controls[int AColumn, ARow];
```

Description

This is Controls, a member of class TControlCollection.

68.1.11.4.3 TControlCollection.Items Property

Lists the items in the collection.

Pascal

```
property Items [Index: Integer]: TControlItem;
```

C++

```
__property TControlItem Items[int Index];
```

Description

Use Items to access individual items in the collection. The value of the Index parameter corresponds to the Index property of TCollectionItem. It represents the position of the item in the collection.

See Also

Count (see page 2749)

ItemClass (see page 2751)

Index (see page 2762)

Add (see page 2736)

Insert (see page 2746)

68.1.12 TControlItem Class

Class Hierarchy

Pascal

```
TControlItem = class(TCollectionItem);
```

C++

```
class TControlItem : public TCollectionItem;
```

File

ExtCtrls

Description

This is class ExtCtrls.TControlItem.

68.1.12.1 TControlItem Members

The following tables list the members exposed by TControlItem.

Protected Methods

	Name	Description
	AssignTo (see page 12403)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	InternalSetLocation (see page 12403)	This is InternalSetLocation, a member of class TControlItem.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Protected Properties

	Name	Description
	GridPanel (see page 12405)	This is GridPanel, a member of class TControlItem.
	Pushed (see page 12405)	This is Pushed, a member of class TControlItem.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 12402)	Creates and initializes a TCollectionItem instance.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 12402)	Destroys the TCollectionItem instance and frees its memory.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetLocation (see page 12404)	This is SetLocation, a member of class TControlItem.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	Column (see page 12404)	This is Column, a member of class TControlItem.
	ColumnSpan (see page 12404)	This is ColumnSpan, a member of class TControlItem.
	Control (see page 12405)	This is Control, a member of class TControlItem.
	Row (see page 12405)	This is Row, a member of class TControlItem.
	RowSpan (see page 12406)	This is RowSpan, a member of class TControlItem.

68.1.12.2 TControlItem.Create Constructor

Creates and initializes a TCollectionItem instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TControlItem(TCollection Collection);
```

Description

Do not call Create to instantiate a collection item. Instead, call the Add method of the collection to which the item should belong.

Collection identifies the TCollection instance to which the new item belongs.

See Also

Add (see page 2736)

Collection (see page 2761)

68.1.12.3 TControlItem.Destroy Destructor

Destroys the TCollectionItem instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TControlItem();
```

Description

Destroy is called indirectly by TCollection's Clear or Delete method.

See Also

Clear (see page 2741)

Delete (see page 2741)

Free (see page 22300)

68.1.12.4 TControlItem Methods

The methods of the TControlItem class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 12403)	Copies the properties of an object to a destination object.
	InternalSetLocation (see page 12403)	This is InternalSetLocation, a member of class TControlItem.

Public Methods

	Name	Description
	SetLocation (see page 12404)	This is SetLocation, a member of class TControlItem.

68.1.12.4.1 TControlItem.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign (see page 2930)

EConvertError (see page 22724)

68.1.12.4.2 TControlItem.InternalSetLocation Method

Pascal

```
procedure InternalSetLocation(AColumn: Integer; ARow: Integer; APushed: Boolean; MoveExisting:
```

```
Boolean);
```

C++

```
__fastcall InternalSetLocation(int AColumn, int ARow, Boolean APushed, Boolean MoveExisting);
```

Description

This is InternalSetLocation, a member of class TControlItem.

68.1.12.4.3 TControlItem.SetLocation Method

Pascal

```
procedure SetLocation(AColumn: Integer; ARow: Integer; APushed: Boolean = False);
```

C++

```
__fastcall SetLocation(int AColumn, int ARow, Boolean APushed = False);
```

Description

This is SetLocation, a member of class TControlItem.

68.1.12.5 TControlItem Properties

The properties of the TControlItem class are listed here.

Protected Properties

	Name	Description
R	GridPanel (see page 12405)	This is GridPanel, a member of class TControlItem.
R	Pushed (see page 12405)	This is Pushed, a member of class TControlItem.

Published Properties

	Name	Description
	Column (see page 12404)	This is Column, a member of class TControlItem.
	ColumnSpan (see page 12404)	This is ColumnSpan, a member of class TControlItem.
	Control (see page 12405)	This is Control, a member of class TControlItem.
	Row (see page 12405)	This is Row, a member of class TControlItem.
	RowSpan (see page 12406)	This is RowSpan, a member of class TControlItem.

68.1.12.5.1 TControlItem.Column Property

Pascal

```
property Column: Integer;
```

C++

```
__property int Column;
```

Description

This is Column, a member of class TControlItem.

68.1.12.5.2 TControlItem.ColumnSpan Property

Pascal

```
property ColumnSpan: TCellSpan;
```

C++

```
__property TCellSpan ColumnSpan;
```

Description

This is ColumnSpan, a member of class TControlItem.

68.1.12.5.3 TControlItem.Control Property

Pascal

```
property Control: TControl;
```

C++

```
__property TControl * Control;
```

Description

This is Control, a member of class TControlItem.

68.1.12.5.4 TControlItem.GridPanel Property

Pascal

```
property GridPanel: TCustomGridPanel;
```

C++

```
__property TCustomGridPanel GridPanel;
```

Description

This is GridPanel, a member of class TControlItem.

68.1.12.5.5 TControlItem.Pushed Property

Pascal

```
property Pushed: Boolean;
```

C++

```
__property Boolean Pushed;
```

Description

This is Pushed, a member of class TControlItem.

68.1.12.5.6 TControlItem.Row Property

Pascal

```
property Row: Integer;
```

C++

```
__property int Row;
```

Description

This is Row, a member of class TControlItem.

68.1.12.5.7 TControlItem.RowSpan Property

Pascal

```
property RowSpan: TCellSpan;
```

C++

```
__property TCellSpan RowSpan;
```

Description

This is RowSpan, a member of class TControlItem.

68.1.13 TCustomColorBox Class

TCustomColorBox represents a combo box that lets users select a color.

Class Hierarchy

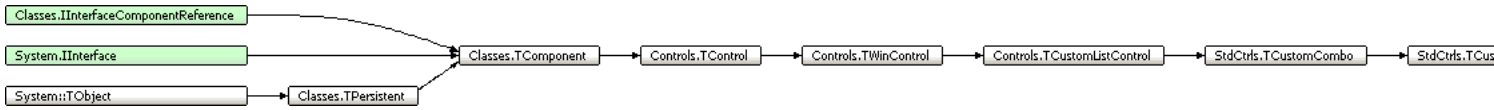

Pascal

```
TCustomColorBox = class(TCustomComboBox);
```

C++

```
class TCustomColorBox : public TCustomComboBox;
```

File

ExtCtrls

Description

TCustomColorBox is the base class for drop-down combo boxes that display a list of colors. The combo box displays a list of all the colors for which the Graphics (see page 14387) unit defines a constant. For a list of these colors, see TColor. Each color is displayed next to a small square that is drawn in the corresponding color.

Do not use TCustomColorBox in applications. It is intended to act as a base class for color selection controls. Rather, to add a drop-down combo box that lets users select a color, use the TColorBox (see page 12182) component, which appears on the Additional page of the component palette.

See Also

TComboBox (see page 21064)

TColorDialog (see page 11270)

TColorBox (see page 12182)

68.1.13.1 TCustomColorBox Members

The following tables list the members exposed by TCustomColorBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnChange (see page 21197)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnCloseUp (see page 21199)	Occurs when the drop-down list closes up due to some user action.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 21224)	Occurs when an item in an owner-draw combo box needs to be displayed.
	OnDropDown (see page 21199)	Occurs when the user opens the drop-down list.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 21225)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnSelect (see page 21201)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.

	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefEditProc (see page 21167)	This is FDefEditProc, a member of class TCustomCombo.
	FDefListProc (see page 21167)	This is FDefListProc, a member of class TCustomCombo.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FDropHandle (see page 21168)	This is FDropHandle, a member of class TCustomCombo.
	FDroppingDown (see page 21168)	This is FDroppingDown, a member of class TCustomCombo.
	FEditHandle (see page 21168)	This is FEditHandle, a member of class TCustomCombo.
	FEditInstance (see page 21168)	This is FEditInstance, a member of class TCustomCombo.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFocusChanged (see page 21168)	This is FFocusChanged, a member of class TCustomCombo.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FIsFocused (see page 21169)	This is FIsFocused, a member of class TCustomCombo.
	FListHandle (see page 21169)	This is FListHandle, a member of class TCustomCombo.
	FListInstance (see page 21169)	This is FListInstance, a member of class TCustomCombo.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveIndex (see page 21169)	This is FSaveIndex, a member of class TCustomCombo.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustDropDown (see page 21171)	Adjusts the size and position of the drop down list.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21171)	Generates an OnChange (see page 21197) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.

	CloseUp (see page 12419)	Generates an OnCloseUp event.
	ComboWndProc (see page 21173)	Responds to Windows messages sent to the edit control or drop-down list.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 21215)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 12420)	Creates the window used to implement the color box.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWindowHandle (see page 21174)	Destroys all of the windows controls that implement the combo box.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21215)	Destroys the Windows control that represents the combo box.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.

	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawItem (see page 12420)	Renders an individual item in the color box.
	DropDown (see page 21175)	Generates an OnDropDown (see page 21199) event.
V	EditWndProc (see page 21175)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetDroppedDown (see page 21176)	Returns the value of the DroppedDown (see page 21185) property.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
A	GetItemCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
V	GetItemCount (see page 21216)	Returns the number of items in the Items (see page 21187) property list.
A	GetItemHt (see page 21177)	Returns the value of the ItemHeight (see page 21186) property.
V	GetItemHt (see page 21216)	Returns the value of the ItemHeight (see page 21186) property.
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
V	GetItemIndex (see page 21177)	Returns the value of the ItemIndex property.
A	GetItemsClass (see page 21177)	Indicates the class that implements the Items (see page 21187) property.
V	GetItemsClass (see page 21217)	Returns the class used to implement the Items (see page 21187) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelLength (see page 21178)	Returns the value of the SelLength (see page 21190) property.
	GetSelStart (see page 21178)	Returns the value of the SelStart (see page 21193) property.
	GetSelText (see page 21217)	Returns the value of the SelText (see page 21222) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
V	KeyDown (see page 12420)	Generates an OnKeyDown event.
V	KeyPress (see page 12421)	Generates an OnKeyPress event.
	KeyUp (see page 6164)	Respond to released key.
	ListWndProc (see page 21178)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
V	Loaded (see page 12421)	Initializes the combo box after its form is loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
V	MeasureItem (see page 21218)	Generates an OnMeasureItem (see page 21225) event.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	PickCustomColor (see page 12422)	This is PickCustomColor, a member of class TCustomColorBox.
	PopulateList (see page 12422)	This is PopulateList, a member of class TCustomColorBox.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
V	Select (see page 12422)	Generates an OnSelect event.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectItem (see page 21218)	Selects a specified item from the drop-down list.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.

	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetDropDownCount (see page 21180)	Sets the value of the DropDownCount (see page 21185) property.
	SetDroppedDown (see page 21180)	Sets the value of the DroppedDown (see page 21185) property.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetItemHeight (see page 21181)	Sets the value of the ItemHeight (see page 21186) property.
A	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
V	SetItemIndex (see page 21181)	Sets the value of the ItemIndex property.
V	SetItems (see page 21181)	Sets the value of the Items (see page 21187) property.
	SetMaxLength (see page 21182)	Sets the value of the MaxLength (see page 21189) property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelLength (see page 21182)	Sets the value of the SelLength (see page 21190) property.
	SetSelStart (see page 21182)	Sets the value of the SelStart (see page 21193) property.
	SetStyle (see page 12422)	Sets the Style (see page 12425) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.

V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WndProc (see page 21183)	Processes messages received by the combo box.
V	WndProc (see page 21219)	Processes messages received by the combo box.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownCount (see page 21185)	Specifies the maximum number of items displayed in the drop-down list.
R	EditHandle (see page 21185)	Returns the window handle for the edit portion of the combo box.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	ItemCount (see page 21186)	Specifies the number of items in the drop-down list.
	ItemHeight (see page 21186)	Specifies the height, in pixels, of the items in the drop-down list.
R	ListHandle (see page 21189)	Provides access to the window handle of the list of items in the combo box.
	MaxLength (see page 21189)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 21190)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Sorted (see page 21222)	Determines whether the list portion of the combo box is alphabetized.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.

	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnGetColors (see page 12426)	Occurs when the control is populated with colors.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
A	AddItem (see page 5940)	Adds an item to the list control.
V	AddItem (see page 21170)	Adds an item to the drop-down list of items.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 5940)	Deletes all of the items from the list control.
V	Clear (see page 21172)	Deletes all text from the edit box and all items from the Items (see page 21187) list.
A	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
V	ClearSelection (see page 21172)	Deselects any selected items in the drop-down list.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
A	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
V	CopySelection (see page 21173)	Copies the selected item in the drop-down list to another list control.
V	Create (see page 12419)	Creates a new TCustomColorBox (see page 12406) object.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	DeleteSelected (see page 5941)	Deletes all the selected items.
V	DeleteSelected (see page 21174)	Deletes the selected item from the drop-down list.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 21166)	Destroys the combo box and frees its memory.

	Destroy (see page 21214)	Destroys the combo box and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Focused (see page 21175)	Indicates whether the combo box has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.

	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
A	SelectAll (see page 5943)	Selects all items or all text in the selected item.
V	SelectAll (see page 21180)	Selects the entire block of text in the edit box.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.

	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AutoCloseUp (see page 21220)	Specifies whether the drop-down closes up automatically when the user selects an item.
	AutoComplete (see page 21220)	Positions to matching list items as you type.
	AutoCompleteDelay (see page 21220)	Specifies the delay between a key press and an attempt to autocomplete the field.
	AutoDropDown (see page 21221)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 21184)	Provides access to the drawing surface of the combo box.
	CharCase (see page 21221)	Determines the case of the text in the combo box.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ColorNames (see page 12423)	Lists the names of the colors in the color box.
R	Colors (see page 12424)	Lists the colors in the color box.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DefaultColorColor (see page 12424)	Specifies the color displayed in the color box for clDefault.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DroppedDown (see page 21185)	Indicates whether the drop-down list is currently displayed.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.

R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	Items (see page 21187)	Provides access to the list of items (strings) in the list portion of the combo box.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	NoneColorColor (see page 12424)	Specifies the color displayed in the color box for clNone.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	Selected (see page 12425)	Specifies the currently selected color.
	SelLength (see page 21190)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 21193)	Specifies the position of the first selected character in the edit portion of the combo box.
	SelText (see page 21222)	Represents the selected text in the edit region.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	Style (see page 12425)	Controls (see page 5711) what colors are listed and how their names are formatted.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TabStop (see page 21196)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.13.2 TCustomColorBox.Create Constructor

Creates a new TCustomColorBox (see page 12406) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomColorBox(TComponent * AOwner);
```

Description

Call Create to instantiate a color box at runtime. TCustomColorBox (see page 12406) objects placed on a form at design time are created automatically.

AOwner specifies the owner of the color box. This component (typically the form) is responsible for freeing the color box.

See Also

Owner (see page 2807)

68.1.13.3 TCustomColorBox Methods

The methods of the TCustomColorBox class are listed here.

Protected Methods

	Name	Description
	CloseUp (see page 12419)	Generates an OnCloseUp event.
	CreateWnd (see page 12420)	Creates the window used to implement the color box.
	DrawItem (see page 12420)	Renders an individual item in the color box.
	KeyDown (see page 12420)	Generates an OnKeyDown event.
	KeyPress (see page 12421)	Generates an OnKeyPress event.
	Loaded (see page 12421)	Initializes the combo box after its form is loaded into memory.
	PickCustomColor (see page 12422)	This is PickCustomColor, a member of class TCustomColorBox.
	PopulateList (see page 12422)	This is PopulateList, a member of class TCustomColorBox.
	Select (see page 12422)	Generates an OnSelect event.
	SetStyle (see page 12422)	Sets the Style (see page 12425) property.

68.1.13.3.1 TCustomColorBox.CloseUp Method

Generates an OnCloseUp event.

Pascal

```
procedure CloseUp; override;
```

C++

```
virtual __fastcall CloseUp();
```

Description

Applications can't call this protected method. It is called automatically when the drop-down list of the combo box closes.

CloseUp generates an OnCloseUp event so that applications can respond when the list closes.

See Also

OnCloseUp (see page 21199)

68.1.13.3.2 TCustomColorBox.CreateWnd Method

Creates the window used to implement the color box.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

Applications can't call this protected method. It is called automatically when the color box is first created or when the window must be destroyed and recreated to reflect property changes.

TCustomColorBox (see page 12406) overrides the inherited CreateWnd method to populate the combo box with the colors specified by the Style (see page 12425) property.

See Also

CreateWnd (see page 6137)

Style (see page 12425)

68.1.13.3.3 TCustomColorBox.DrawItem Method

Renders an individual item in the color box.

Pascal

```
procedure DrawItem(Index: Integer; Rect: TRect; State: TOwnerDrawState); override;
```

C++

```
virtual __fastcall DrawItem(int Index, TRect Rect, TOwnerDrawState State);
```

Description

Applications can't call this protected method. It is called automatically when the color box needs to paint one of the items in the list. DrawItem draws a small box that displays the color that the item represents, followed by the name of the color.

Index is the index of the item to draw, where 0 specifies the first item, 1 specifies the second item, and so on.

Rect indicates the boundaries of the item on the color box's canvas, in client coordinates.

State indicates state information that can affect the way the item is drawn. In TCustomColorBox (see page 12406), DrawItem uses State to determine whether an item is selected, but ignores all other state information.

See Also

Items (see page 21187)

68.1.13.3.4 TCustomColorBox.KeyDown Method

Generates an OnKeyDown event.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

Applications can't call this protected method. It is called automatically when the user first presses a key while the color box has focus.

Key is the key code of the key that was pressed.

Shift indicates the state of the Ctrl, Shift, and Alt keys when Key was pressed.

TCustomColorBox (see page 12406) overrides the inherited KeyDown method to prepare for the selection of a new color. It then calls the inherited method, which generates an OnKeyDown event.

See Also

OnKeyDown (see page 6225)

68.1.13.3.5 TCustomColorBox.KeyPress Method

Generates an OnKeyPress event.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

Applications can't call this protected method. It is called automatically when the user presses a key while the color box has focus.

Key is the key code of the key that was pressed.

TCustomColorBox (see page 12406) overrides the inherited KeyPress method to display a color selection dialog if the user presses the Enter key while the custom color item has focus. Otherwise, KeyPress calls the inherited method, which generates an OnKeyPress event.

See Also

OnKeyPress (see page 6227)

TColorDialog (see page 11270)

68.1.13.3.6 TCustomColorBox.Loaded Method

Initializes the combo box after its form is loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The component streaming system calls this method after it loads the combo box's form from a stream.

TCustomCombo overrides the inherited method to select the item specified by the ItemIndex property once the form is fully loaded.

See Also

Loaded (see page 2782)

ItemIndex

68.1.13.3.7 TCustomColorBox.PickCustomColor Method

Pascal

```
function PickCustomColor: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean PickCustomColor();
```

Description

This is PickCustomColor, a member of class TCustomColorBox.

68.1.13.3.8 TCustomColorBox.PopulateList Method

Pascal

```
procedure PopulateList;
```

C++

```
__fastcall PopulateList();
```

Description

This is PopulateList, a member of class TCustomColorBox.

68.1.13.3.9 TCustomColorBox.Select Method

Generates an OnSelect event.

Pascal

```
procedure Select; override;
```

C++

```
virtual __fastcall Select();
```

Description

Applications can't call this protected method. It is called automatically when the user selects a color.

TCustomColorBox (see page 12406) overrides the inherited method to display a color selection dialog if the user selects the custom color item. After checking for a custom color, TCustomColorBox (see page 12406) calls the inherited method, which generates an OnSelect event.

See Also

OnSelect (see page 21201)

68.1.13.3.10 TCustomColorBox.SetStyle Method

Sets the Style (see page 12425) property.

Pascal

```
procedure SetStyle(AStyle: TColorBoxStyle); reintroduce;
```

C++

```
__fastcall SetStyle(TColorBoxStyle AStyle);
```

Description

SetStyle is the protected write implementation of the Style (see page 12425) property. It repopulates the list to reflect the new value of Style (see page 12425), and disables the color box if the new value for Style (see page 12425) results in an empty color box.

See Also

Style (see page 12425)

68.1.13.4 TCustomColorBox Properties

The properties of the TCustomColorBox class are listed here.

Public Properties

	Name	Description
R	ColorNames (see page 12423)	Lists the names of the colors in the color box.
R	Colors (see page 12424)	Lists the colors in the color box.
	DefaultColorColor (see page 12424)	Specifies the color displayed in the color box for clDefault.
	NoneColorColor (see page 12424)	Specifies the color displayed in the color box for clNone.
	Selected (see page 12425)	Specifies the currently selected color.
	Style (see page 12425)	Controls (see page 5711) what colors are listed and how their names are formatted.

68.1.13.4.1 TCustomColorBox.ColorNames Property

Lists the names of the colors in the color box.

Pascal

```
property ColorNames [Index: Integer]: string;
```

C++

```
__property AnsiString ColorNames[int Index];
```

Description

Use ColorNames to obtain the name of one of the colors listed in the color box.

Index is the index of the name, where 0 is the index of the first color in the box, 1 is the index of the second color, and so on.

The value of each name is either the name of a color constant defined in the Graphics (see page 14387) unit (such as clBlack), or it is the name of the color with the "cl" stripped off (Black). The Style (see page 12425) property determines which name is used for each color.

The Style (see page 12425) property also controls which colors are included in the list.

See Also

Colors (see page 12424)

Items (see page 21187)

ItemIndex

Style (see page 12425)

68.1.13.4.2 TCustomColorBox.Colors Property

Lists the colors in the color box.

Pascal

```
property Colors [Index: Integer]: TColor;
```

C++

```
__property TColor Colors[int Index];
```

Description

Use Colors to obtain the value of a color in the color box.

Index is the index of the color, where 0 is the index of the first color in the box, 1 is the index of the second color, and so on.

Note: The Style (see page 12425) property controls which colors are included in the list.

See Also

ColorNames (see page 12423)

Selected (see page 12425)

DefaultColorColor (see page 12424)

NoneColorColor (see page 12424)

Style (see page 12425)

68.1.13.4.3 TCustomColorBox.DefaultColorColor Property

Specifies the color displayed in the color box for clDefault.

Pascal

```
property DefaultColorColor: TColor;
```

C++

```
__property TColor DefaultColorColor;
```

Description

Set DefaultColorColor to indicate what color the color box should use when drawing the icon for clDefault. This property only affects the color box if the Style (see page 12425) property includes cbSystemColors and cbIncludeDefault.

Note: DefaultColorColor only affects the way clDefault appears in the color box. When you assign clDefault to a control, the control determines its own default color.

See Also

Colors (see page 12424)

NoneColorColor (see page 12424)

Style (see page 12425)

68.1.13.4.4 TCustomColorBox.NoneColorColor Property

Specifies the color displayed in the color box for clNone.

Pascal

```
property NoneColorColor: TColor;
```

C++

```
__property TColor NoneColorColor;
```

Description

Set NoneColorColor to indicate what color the color box should use when drawing the icon for clNone. This property only affects the color box if the Style (see page 12425) property includes cbSystemColors and cbIncludeNone.

Note: NoneColorColor only affects the way clNone appears in the color box. When you assign clNone to a control, the control determines how to interpret clNone.

See Also

Colors (see page 12424)

DefaultColorColor (see page 12424)

Style (see page 12425)

68.1.13.4.5 TCustomColorBox.Selected Property

Specifies the currently selected color.

Pascal

```
property Selected: TColor;
```

C++

```
__property TColor Selected;
```

Description

Use Selected to get or set the selected color as a TColor value.

See Also

Colors (see page 12424)

ItemIndex

68.1.13.4.6 TCustomColorBox.Style Property

Controls (see page 5711) what colors are listed and how their names are formatted.

Pascal

```
property Style: TColorBoxStyle;
```

C++

```
__property TColorBoxStyle Style;
```

Description

Use Style to configure what colors appear in the list and how the names appear. Style is a set that includes 0 or more of the TColorBoxStyles (see page 13267) flags.

See Also

DefaultColorColor (see page 12424)

NoneColorColor (see page 12424)

ColorNames (see page 12423)

Colors (see page 12424)

TColor (see page 14648)

TColorDialog (see page 11270)

68.1.13.5 TCustomColorBox Events

The events of the TCustomColorBox class are listed here.

Public Events

	Name	Description
	OnGetColors (see page 12426)	Occurs when the control is populated with colors.

68.1.13.5.1 TCustomColorBox.OnGetColors Event

Occurs when the control is populated with colors.

Pascal

```
property OnGetColors: TGetColorsEvent;
```

C++

```
__property TGetColorsEvent OnGetColors;
```

68.1.14 TCustomColorListBox Class

Class Hierarchy

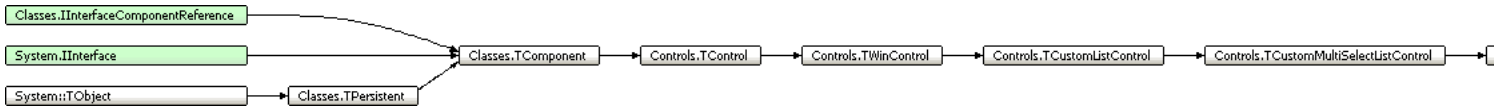

Pascal

```
TCustomColorListBox = class(TCustomListBox);
```

C++

```
class TCustomColorListBox : public TCustomListBox;
```

File

ExtCtrls

Description

This is class ExtCtrls.TCustomColorListBox.

68.1.14.1 TCustomColorListBox Members

The following tables list the members exposed by TCustomColorListBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnData (see page 21357)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 21357)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 21358)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 21358)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 21360)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.

	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FMoving (see page 21321)	This is FMoving, a member of class TCustomListBox.
	FMultiSelect (see page 5956)	This is FMultiSelect, a member of class TCustomMultiSelectListControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 21324)	Initializes the window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 12439)	Creates a Windows control corresponding to the list box component.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.

	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DeleteString (see page 21326)	Deletes a string from the list box.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
V	DestroyWnd (see page 21326)	Saves the list box items in a string list before freeing the underlying Windows control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFindData (see page 21326)	Returns the index of a string in the list box.
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetData (see page 21327)	Returns a string in the list box given its index.
	DoGetDataObject (see page 21327)	Returns the object associated with an item in a virtual list box.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragCanceled (see page 21328)	Sends a mouse message canceling the drag-and-drop operation.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawItem (see page 12439)	Generates an OnDrawItem event.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.

	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetCount (see page 21329)	Returns the value of the Count (see page 21346) property.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetItemData (see page 21329)	Returns a 32-bit value associated with the specified item in a list box.
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
V	GetItemIndex (see page 21329)	Returns the value of the ItemIndex property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
A	GetSelCount (see page 5956)	Returns the value of the SelCount (see page 5957) property.
V	GetSelCount (see page 21330)	Returns the number of selected items.
	GetSelected (see page 21330)	Indicates whether a particular item is selected.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InternalGetItemData (see page 21330)	Retrieves the data associated with an item in the list box.
	InternalSetItemData (see page 21330)	Associates an integer with a specified item in the list box.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
V	KeyDown (see page 12440)	Respond to key press events.
V	KeyPress (see page 12441)	Responds when the user presses a key.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 12441)	Initializes the control after it is loaded from a stream.
V	LoadRecreateItems (see page 21334)	This is LoadRecreateItems, a member of class TCustomListBox.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
V	MeasureItem (see page 21335)	Generates an OnMeasureItem (see page 21360) event.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.

	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	PickCustomColor (see page 12441)	This is PickCustomColor, a member of class TCustomColorListBox.
	PopulateList (see page 12442)	This is PopulateList, a member of class TCustomColorListBox.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetContent (see page 21335)	Removes all of the items from a list box.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
V	SaveRecreateItems (see page 21336)	This is SaveRecreateItems, a member of class TCustomListBox.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItemData (see page 21336)	Sets the 32-bit value associated with the specified item in a list box.

	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
	SetItemIndex (see page 21336)	Gives focus to the item with a specified index.
	SetMultiSelect (see page 5956)	Sets the value of the MultiSelect (see page 5957) property.
	SetMultiSelect (see page 21337)	Sets the MultiSelect property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetStyle (see page 12442)	This is SetStyle, a member of class TCustomColorListBox.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WndProc (see page 21337)	Handles Windows messages.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 21343)	Determines whether the list box has a border.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.

	Columns (see page 21346)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	ExtendedSelect (see page 21346)	This is ExtendedSelect, a member of class TCustomListBox.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 21347)	Determines whether the list box displays the partial items.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	ItemHeight (see page 21347)	Specifies the height in pixels of an item in an owner-draw list box.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 21350)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Sorted (see page 21353)	Specifies whether the items in a list box are arranged alphabetically.
	TabWidth (see page 21355)	Specifies the size of the tabs in the list box.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnGetColors (see page 12446)	This is OnGetColors, a member of class TCustomColorListBox.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddItem (see page 5940)	Adds an item to the list control.
	AddItem (see page 21322)	Adds an item to the list box
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 5940)	Deletes all of the items from the list control.
V	Clear (see page 21322)	Deletes all items from the list box.
A	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
V	ClearSelection (see page 21323)	Deselects all selected items in the list box, if possible.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
A	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
V	CopySelection (see page 21323)	Copies the selected items to another list-type control
V	Create (see page 12438)	Instantiates a list box control.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	DeleteSelected (see page 5941)	Deletes all the selected items.
V	DeleteSelected (see page 21325)	Deletes the selected items from the list box
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 21320)	Destroys the instance of a list box control.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ItemAtPos (see page 21331)	Returns the index of the list box item indicated by the coordinates of a point on the control.
	ItemRect (see page 21333)	Returns the rectangle that surrounds the item specified in the Item parameter.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.

	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
A	SelectAll (see page 5943)	Selects all items or all text in the selected item.
V	SelectAll (see page 21336)	Selects the entire block of text in the list box.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 21342)	Determines whether the user can give focus to items by typing in the list.
	AutoCompleteDelay (see page 21342)	Specifies the delay between a key press and an attempt to autocomplete the field.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 21343)	Provides a drawing surface when implementing a handler for the OnDrawItem (see page 21358) event of an owner-draw list box.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ColorNames (see page 12442)	This is ColorNames, a member of class TCustomColorListBox.
R	Colors (see page 12443)	This is Colors, a member of class TCustomColorListBox.

R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Count (see page 21346)	Indicates the number of items in the list box.
	DefaultColorColor (see page 12443)	This is DefaultColorColor, a member of class TCustomColorListBox.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	Items (see page 21348)	Contains the strings that appear in the list box.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	MultiSelect (see page 5957)	Specifies whether the user can select more than one item.
	NoneColorColor (see page 12443)	This is NoneColorColor, a member of class TCustomColorListBox.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ScrollWidth (see page 21351)	Specifies the width, in pixels, by which the list box can scroll horizontally.
R	SelCount (see page 5957)	Indicates the number of selected items.
	Selected (see page 12443)	Indicates whether a particular item is selected.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	Style (see page 12445)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.

	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TopIndex (see page 21356)	Specifies the index number of the item that appears at the top of the list box.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	TabStop (see page 21355)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.14.2 TCustomColorListBox.Create Constructor

Instantiates a list box control.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomColorListBox(TComponent * AOwner);
```

Description

Call Create to instantiate a list box control at runtime. For list boxes created at design time, Create is called automatically.AOwner is the component that is responsible for freeing the list box. It becomes the value of the Owner property. If AOwner is nil, the caller is responsible for explicitly freeing the control.

See Also

Owner (see page 2807)

68.1.14.3 TCustomColorListBox Methods

The methods of the TCustomColorListBox class are listed here.

Protected Methods

	Name	Description
	CreateWnd (see page 12439)	Creates a Windows control corresponding to the list box component.
	DrawItem (see page 12439)	Generates an OnDrawItem event.
	KeyDown (see page 12440)	Respond to key press events.
	KeyPress (see page 12441)	Responds when the user presses a key.
	Loaded (see page 12441)	Initializes the control after it is loaded from a stream.
	PickCustomColor (see page 12441)	This is PickCustomColor, a member of class TCustomColorListBox.
	PopulateList (see page 12442)	This is PopulateList, a member of class TCustomColorListBox.
	SetStyle (see page 12442)	This is SetStyle, a member of class TCustomColorListBox.

68.1.14.3.1 TCustomColorListBox.CreateWnd Method

Creates a Windows control corresponding to the list box component.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the list box must be displayed. CreateWnd calls the inherited CreateWnd method, which calls

CreateParams to initialize the window-creation parameters.

CreateWindowHandle to create the window handle for the control.

CreateWnd additionally sets up the list box position, tabs, items, and columns.

See Also

Columns (see page 21346)

CreateHandle (see page 6135)

CreateParams (see page 21324)

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

ItemIndex

Items (see page 21348)

TabWidth (see page 21355)

TopIndex (see page 21356)

68.1.14.3.2 TCustomColorListBox.DrawItem Method

Generates an OnDrawItem event.

Pascal

```
procedure DrawItem(Index: Integer; Rect: TRect; State: TOwnerDrawState); override;
```

C++

```
virtual __fastcall DrawItem(int Index, TRect Rect, TOwnerDrawState State);
```

Description

DrawItem is called whenever a visual aspect of an owner-draw list box changes. The list box calls DrawItem for each visible item in its list, passing the index of the item in the Index parameter.

By default, the DrawItem method for a list box calls any event handler attached to the component's OnDrawItem event. If there is no handler attached, DrawItem fills the rectangle passed in Rect and draws any text associated with the indexed item. The default drawing ignores the State parameter, although State is passed along to attached event handlers.

See Also

ItemHeight (see page 21347)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

Style (see page 21354)

68.1.14.3.3 TCustomColorListBox.KeyDown Method

Respond to key press events.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-down message (WM_KEYDOWN) from Windows, its message handler calls the DoKeyDown method. If DoKeyDown determines that the control should, in fact, process the character, it calls KeyDown, passing the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

See Also

KeyUp (see page 6164)

OnKeyDown (see page 6225)

DoKeyDown (see page 6145)

DoKeyPress (see page 6146)

DoKeyUp (see page 6146)

68.1.14.3.4 TCustomColorListBox.KeyPress Method

Responds when the user presses a key.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress is called automatically when the list box receives a key-press message (WM_CHAR). It generates an OnKeyPress event, and then, if the AutoComplete property is true, updates ItemIndex to reflect the next character that the user typed.

The Key parameter is the character represented by the key that is pressed, not a Windows virtual key code.

See Also

OnKeyPress (see page 6227)

68.1.14.3.5 TCustomColorListBox.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded (see page 2782)

Action (see page 5797)

ActionChange (see page 5737)

68.1.14.3.6 TCustomColorListBox.PickCustomColor Method

Pascal

```
function PickCustomColor: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean PickCustomColor();
```

Description

This is PickCustomColor, a member of class TCustomColorListBox.

68.1.14.3.7 TCustomColorListBox.PopulateList Method

Pascal

```
procedure PopulateList;
```

C++

```
__fastcall PopulateList();
```

Description

This is PopulateList, a member of class TCustomColorListBox.

68.1.14.3.8 TCustomColorListBox.SetStyle Method

Pascal

```
procedure SetStyle(AStyle: TColorBoxStyle); reintroduce;
```

C++

```
__fastcall SetStyle(TColorBoxStyle AStyle);
```

Description

This is SetStyle, a member of class TCustomColorListBox.

68.1.14.4 TCustomColorListBox Properties

The properties of the TCustomColorListBox class are listed here.

Public Properties

	Name	Description
R	ColorNames (see page 12442)	This is ColorNames, a member of class TCustomColorListBox.
R	Colors (see page 12443)	This is Colors, a member of class TCustomColorListBox.
	DefaultColorColor (see page 12443)	This is DefaultColorColor, a member of class TCustomColorListBox.
	NoneColorColor (see page 12443)	This is NoneColorColor, a member of class TCustomColorListBox.
	Selected (see page 12443)	Indicates whether a particular item is selected.
	Style (see page 12445)	Determines whether the list box is standard or owner-draw and whether it is virtual.

68.1.14.4.1 TCustomColorListBox.ColorNames Property

Pascal

```
property ColorNames [Index: Integer]: string;
```

C++

```
__property AnsiString ColorNames[int Index];
```

Description

This is ColorNames, a member of class TCustomColorListBox.

68.1.14.4.2 TCustomColorListBox.Colors Property

Pascal

```
property Colors [Index: Integer]: TColor;
```

C++

```
__property TColor Colors[int Index];
```

Description

This is Colors, a member of class TCustomColorListBox.

68.1.14.4.3 TCustomColorListBox.DefaultColorColor Property

Pascal

```
property DefaultColorColor: TColor;
```

C++

```
__property TColor DefaultColorColor;
```

Description

This is DefaultColorColor, a member of class TCustomColorListBox.

68.1.14.4.4 TCustomColorListBox.NoneColorColor Property

Pascal

```
property NoneColorColor: TColor;
```

C++

```
__property TColor NoneColorColor;
```

Description

This is NoneColorColor, a member of class TCustomColorListBox.

68.1.14.4.5 TCustomColorListBox.Selected Property

Indicates whether a particular item is selected.

Pascal

```
property Selected: TColor;
```

C++

```
__property TColor Selected;
```

Description

Use Selected to query the state of a particular item in the list box. If the item specified by the Index parameter is selected in the list box, the value of the Selected property is true. If the specified item is not selected, Selected is false. Set Selected to change the currently selected item.

The Index parameter is the item referenced by its position in the list box, with the first item having an Index value of 0.

An item is selected when the user highlights it. More than one item in the list box can be selected by setting the MultiSelect property to true. In this case, the Selected item is the one that has focus when the list box has input focus.

C++ Examples:

```
/*
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Continue will continue listing
file sizes in a multiple selection that includes the deleted
file.  Break will stop.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
*/
#include <stdio.h>     // for FILE, fopen, fstat, fileno, fread and fclose
#include <sys\stat.h>  // for fstat and the stat struct

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < FileListBox1->Items->Count; i++)
  {
    if (FileListBox1->Selected[i])
    {
      if (!FileExists(FileListBox1->Items->Strings[i]))
      {
        MessageBeep(0);
        if (Application->MessageBox(
          AnsiString("File " + FileListBox1->Items->Strings[i] + "not found").c_str(),
          NULL,
          MB_OKCANCEL) == IDCANCEL)
          break;
        else
          continue;
      }
      FILE *F = fopen(FileListBox1->Items->Strings[i].c_str(),"r");
      struct stat statbuf;
      fstat(fileno(F), &statbuf);
      ListBox1->Items->Add(
          FileListBox1->Items->Strings[i] + ": " + IntToStr(statbuf.st_size));
      fclose(F);
    }
  }
}
```

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
```

```
          MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                     ' not found', mtError, [mbOk], 0);
          Continue;
        end;
        filehandle:=  FileOpen(FileListBox1.Items.Strings[i], 1);
        if (filehandle = -1) then
        begin
          MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                     ' cannot be opened with access mode 1.', mtError, [mbOk], 0);
          Continue;
        end
        else
          FileClose(filehandle);

        AssignFile(F, FileListBox1.Items.Strings[i]);
        Reset(F, 1);
        ListBox1.Items.Add(
          FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
        CloseFile(F);
      end;
    finally
    { do something here }
    end;
  end;
end;
```

See Also

extendedselect (see page 21346)

Items (see page 21348)

MultiSelect

SelCount

68.1.14.4.6 TCustomColorListBox.Style Property

Determines whether the list box is standard or owner-draw and whether it is virtual.

Pascal

```
property Style: TColorBoxStyle;
```

C++

```
__property TColorBoxStyle Style;
```

Description

Use Style to specify whether the list box is a standard list box that displays text strings, or an owner-draw list box that can display graphical images. Owner-draw list boxes let you display items in some nonstandard way. In this case, you must write the code to paint items in the list box.

In addition, Style controls whether the list box is virtual, which means that items are supplied dynamically using event handlers. By default, Style is lbStandard, meaning that the list box is not virtual and it displays each item as a string.

See Also

DrawItem (see page 21328)

ItemHeight (see page 21347)

Items (see page 21348)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

OnData (see page 21357)

OnDataFind (see page 21357)

OnDataObject (see page 21358)

68.1.14.5 TCustomColorListBox Events

The events of the TCustomColorListBox class are listed here.

Public Events

	Name	Description
	OnGetColors (see page 12446)	This is OnGetColors, a member of class TCustomColorListBox.

68.1.14.5.1 TCustomColorListBox.OnGetColors Event

Pascal

```
property OnGetColors: TLBGetColorsEvent;
```

C++

```
__property TLBGetColorsEvent OnGetColors;
```

Description

This is OnGetColors, a member of class TCustomColorListBox.

68.1.15 TCustomControlBar Class

TCustomControlBar is the base class for control bar components.

Class Hierarchy

Pascal

```
TCustomControlBar = class(TCustomControl);
```

C++

```
class TCustomControlBar : public TCustomControl;
```

File

ExtCtrls

Description

Use TCustomControlBar as a base class when defining control bar components. Control bars are designed to manage the layout of toolbar controls, although they can contain other controls as well. They provide a convenient docking site in applications that have dockable toolbars.

Do not create instances of TCustomControlBar. To add a control bar to a form, use TControlBar (see page 12316) instead.

See Also

TControlBar (see page 12316)

TToolBar (see page 4994)

68.1.15.1 TCustomControlBar Members

The following tables list the members exposed by TCustomControlBar.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnBandDrag (see page 12472)	Occurs when the user starts a drag operation on a specified control.
	OnBandInfo (see page 12473)	Occurs when the control bar adjusts the docking position of a band.
	OnBandMove (see page 12473)	This is OnBandMove, a member of class TCustomControlBar.
	OnBandPaint (see page 12473)	This is OnBandPaint, a member of class TCustomControlBar.
	OnBeginBandMove (see page 12474)	This is OnBeginBandMove, a member of class TCustomControlBar.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndBandMove (see page 12474)	This is OnEndBandMove, a member of class TCustomControlBar.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnPaint (see page 12474)	Occurs when the control bar is painted.

	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 12459)	Aligns controls in the control bar.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 12459)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.

	CreateParams (see page 12460)	Sets the control bar's window flags.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DoBandMove (see page 12460)	Generates an OnBandMove (see page 12473) event.
	DoBandPaint (see page 12460)	This is DoBandPaint, a member of class TCustomControlBar.
	DoBeginBandMove (see page 12461)	This is DoBeginBandMove, a member of class TCustomControlBar.
	DockOver (see page 12461)	Generates an OnDockOver event and ensures proper docking of floating controls.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndBandMove (see page 12461)	This is DoEndBandMove, a member of class TCustomControlBar.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoPaletteChange (see page 12461)	Prepares the control bar to realize a new palette.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragControl (see page 12462)	Starts a drag operation on a control band.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.

	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetControlInfo (see page 12463)	Generates an OnBandInfo (see page 12473) event.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetPalette (see page 12463)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
V	GetSiteInfo (see page 12464)	Generates an OnGetSiteInfo event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HitTest (see page 12464)	Returns the control at the specified coordinates.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 12465)	Handles mouse button clicks.
V	MouseMove (see page 12465)	Handles mouse movements.
V	MouseUp (see page 12465)	Handles releasing of a mouse button.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.

	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 12466)	Paints the control bar.
	PaintControlFrame (see page 12466)	Paints a control band's frame.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.

V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoDock (see page 12467)	Determines whether the control bar temporarily docks drag clients to provide visual feedback.
	AutoDrag (see page 12468)	Specifies whether control bands can be undocked by dragging.
	AutoSize (see page 12468)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 12469)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	CornerEdge (see page 12469)	This is CornerEdge, a member of class TCustomControlBar.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DockSite (see page 12469)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DrawingStyle (see page 12470)	This is DrawingStyle, a member of class TCustomControlBar.

	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GradientDirection (see page 12471)	This is GradientDirection, a member of class TCustomControlBar.
	GradientEndColor (see page 12471)	This is GradientEndColor, a member of class TCustomControlBar.
	GradientStartColor (see page 12471)	This is GradientStartColor, a member of class TCustomControlBar.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	RowSize (see page 12472)	This is RowSize, a member of class TCustomControlBar.
	RowSnap (see page 12472)	This is RowSnap, a member of class TCustomControlBar.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 12457)	Creates and initializes an instance of TCustomControlBar (see page 12446).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.

	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 12458)	Destroys the TCustomControlBar (see page 12446) instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
V	FlipChildren (see page 12462)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	StickControls (see page 12467)	Places controls in their final positions after a control is moved in or dropped on the control bar.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.

	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.

	ParentWindow (see page 6210)	Reference to parent's underlying control.
	Picture (see page 12471)	This is Picture, a member of class TCustomControlBar.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.15.2 TCustomControlBar.Create Constructor

Creates and initializes an instance of TCustomControlBar (see page 12446).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomControlBar(TComponent * AOwner);
```

Description

Call Create to instantiate a control bar at runtime. Control bars added to forms at design time are created automatically.

AOwner is a component, typically the form, that is responsible for freeing the control bar instance. It becomes the value of the Owner property.

See Also

Owner (see page 2807)

68.1.15.3 TCustomControlBar.Destroy Destructor

Destroys the TCustomControlBar (see page 12446) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomControlBar();
```

Description

Do not call Destroy. Instead, call Free.

Destroy frees the memory allocated for the TCustomControlBar (see page 12446) properties.

See Also

Free (see page 22300)

68.1.15.4 TCustomControlBar Methods

The methods of the TCustomControlBar class are listed here.

Protected Methods

	Name	Description
	AlignControls (see page 12459)	Aligns controls in the control bar.
	CanAutoSize (see page 12459)	Indicates how the control can be resized.
	CreateParams (see page 12460)	Sets the control bar's window flags.
	DoBandMove (see page 12460)	Generates an OnBandMove (see page 12473) event.
	DoBandPaint (see page 12460)	This is DoBandPaint, a member of class TCustomControlBar.
	DoBeginBandMove (see page 12461)	This is DoBeginBandMove, a member of class TCustomControlBar.
	DockOver (see page 12461)	Generates an OnDockOver event and ensures proper docking of floating controls.
	DoEndBandMove (see page 12461)	This is DoEndBandMove, a member of class TCustomControlBar.
	DoPaletteChange (see page 12461)	Prepares the control bar to realize a new palette.
	DragControl (see page 12462)	Starts a drag operation on a control band.
	GetControlInfo (see page 12463)	Generates an OnBandInfo (see page 12473) event.
	GetPalette (see page 12463)	Returns the handle to a palette for use by the control.
	GetSiteInfo (see page 12464)	Generates an OnGetSiteInfo event
	HitTest (see page 12464)	Returns the control at the specified coordinates.
	MouseDown (see page 12465)	Handles mouse button clicks.
	MouseMove (see page 12465)	Handles mouse movements.
	MouseUp (see page 12465)	Handles releasing of a mouse button.
	Paint (see page 12466)	Paints the control bar.
	PaintControlFrame (see page 12466)	Paints a control band's frame.

Public Methods

	Name	Description
	FlipChildren (see page 12462)	Reverses the positions of child controls.
	StickControls (see page 12467)	Places controls in their final positions after a control is moved in or dropped on the control bar.

68.1.15.4.1 TCustomControlBar.AlignControls Method

Aligns controls in the control bar.

Pascal

```
procedure AlignControls(AControl: TControl; var ARect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect ARect);
```

Description

Use AlignControls to align all controls within an area of the control bar. The alignment of child controls is based on their Align property.

AControl is a child control that takes precedence in alignment over other, similarly-aligned controls. This parameter can be nil (Delphi) or NULL (C++).

Rect specifies the area in which to align the controls.

TCustomControlBar (see page 12446) overrides the inherited method to enforce the alignment of child controls in a manner appropriate to a control bar. AlignControls determines the best alignment for each control according to its size and the available space.

68.1.15.4.2 TCustomControlBar.CanAutoSize Method

Indicates how the control can be resized.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize property is true and an attempt is made to resize the control. It allows controls to implement the AutoSize property before the resize sequence that begins with the OnCanResize and OnConstrainedResize events and ends with the OnResize event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the control. CanAutoSize adjusts these values so that the control's size adjusts to its contents. If the adjustment results in values of NewWidth and NewHeight that differ from the control's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width and Height, CanAutoSize returns false, and the resize sequence is aborted.

As implemented in TWinControl, CanAutoSize returns true, adjusting NewWidth and NewHeight so that all the control's children fit.

See Also

Constraints (see page 5812)

ConstrainedResize (see page 6133)

AdjustSize (see page 6129)

68.1.15.4.3 TCustomControlBar.CreateParams Method

Sets the control bar's window flags.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. CreateParams fills in the parameter record that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

In TCustomControlBar (see page 12446), CreateParams overrides the inherited method in order to turn off the CS_HREDRAW and CS_VREDRAW windows flags.

See Also

CreateWnd (see page 6137)

CreateWindowHandle (see page 6137)

68.1.15.4.4 TCustomControlBar.DoBandMove Method

Generates an OnBandMove (see page 12473) event.

Pascal

```
procedure DoBandMove(Control: TControl; var ARect: TRect); virtual;
```

C++

```
virtual __fastcall DoBandMove(TControl * Control, TRect ARect);
```

Description

Do not call DoBandMove. It is called automatically when a band is moved in the control bar.

Control is the band that is about to be moved.

ARect is the area to which the control is about to be moved. This value can be changed by the OnBandMove (see page 12473) event handler.

Override DoBandMove to perform actions in addition to calling the OnBandMove (see page 12473) event handler when a band is moved.

See Also

onbandmove (see page 12473)

68.1.15.4.5 TCustomControlBar.DoBandPaint Method

Pascal

```
procedure DoBandPaint(Control: TControl; Canvas: TCanvas; var ARect: TRect; var Options:
TBandPaintOptions); virtual;
```

C++

```
virtual __fastcall DoBandPaint(TControl * Control, TCanvas Canvas, TRect ARect,
TBandPaintOptions Options);
```

Description

This is DoBandPaint, a member of class TCustomControlBar.

68.1.15.4.6 TCustomControlBar.DoBeginBandMove Method

Pascal

```
function DoBeginBandMove(Control: TControl): Boolean; dynamic;
```

C++

```
__fastcall Boolean DoBeginBandMove(TControl * Control);
```

Description

This is DoBeginBandMove, a member of class TCustomControlBar.

68.1.15.4.7 TCustomControlBar.DockOver Method

Generates an OnDockOver event and ensures proper docking of floating controls.

Pascal

```
procedure DockOver(Source: TDragDockObject; X: Integer; Y: Integer; State: TDragState; var
Accept: Boolean); override;
```

C++

```
virtual __fastcall DockOver(TDragDockObject Source, int X, int Y, TDragState State, Boolean
Accept);
```

Description

DockOver overrides the inherited method to ensure proper docking of floating controls that are accepted by the OnDockOver event handler.

See Also

OnDockOver (see page 6216)

68.1.15.4.8 TCustomControlBar.DoEndBandMove Method

Pascal

```
procedure DoEndBandMove(Control: TControl); dynamic;
```

C++

```
__fastcall DoEndBandMove(TControl * Control);
```

Description

This is DoEndBandMove, a member of class TCustomControlBar.

68.1.15.4.9 TCustomControlBar.DoPaletteChange Method

Prepares the control bar to realize a new palette.

Pascal

```
function DoPaletteChange: Boolean;
```

C++

```
__fastcall Boolean DoPaletteChange();
```

Description

Call DoPaletteChange to prepare the control bar to realize a new palette. DoPaletteChange returns true if the control bar should be redrawn to reflect the new palette, and false otherwise.

DoPaletteChange is called when the Picture (see page 12471) property changes.

See Also

picture (see page 12471)

getpalette (see page 12463)

68.1.15.4.10 TCustomControlBar.DragControl Method

Starts a drag operation on a control band.

Pascal

```
function DragControl(AControl: TControl; X: Integer; Y: Integer; KeepCapture: Boolean =
False): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DragControl(TControl * AControl, int X, int Y, Boolean KeepCapture
= False);
```

Description

Call DragControl to start a drag operation on a control band.

If AutoDrag (see page 12468) is true, DragControl is called automatically when a control is undocked from the control bar.

AControl is the control on the band that should be dragged.

X and Y are the current positions of the mouse pointer.

KeepCapture indicates whether the control bar should capture the mouse when processing the drag.

See Also

AutoDrag (see page 12468)

Capture (see page 6099)

68.1.15.4.11 TCustomControlBar.FlipChildren Method

Reverses the positions of child controls.

Pascal

```
procedure FlipChildren(AllLevels: Boolean); override;
```

C++

```
virtual __fastcall FlipChildren(Boolean AllLevels);
```

Description

Call FlipChildren to flip the control's children; that is, to move children on the left side of the control to the right side and vice

versa. FlipChildren reverses the position of all child controls and adjusts their Align properties, if necessary, to enforce the new position.

AllLevels specifies whether FlipChildren should be called recursively on the control's children.

FlipChildren can be used to reverse the layout of an application when it is running in Middle Eastern locales where users read from right to left instead of left to right: Call UseRightToLeftAlignment to determine whether the BiDiMode property dictates a reversal of alignment based on the system locale.

Note: FlipChildren does the same thing as the Flip Children command on the control's context menu in the IDE.

See Also

BiDiMode (see page 5803)

UseRightToLeftAlignment (see page 5790)

68.1.15.4.12 TCustomControlBar.GetControlInfo Method

Generates an OnBandInfo (see page 12473) event.

Pascal

```
procedure GetControlInfo(AControl: TControl; var Insets: TRect; var PreferredSize: Integer;
var RowCount: Integer); virtual;
```

C++

```
virtual __fastcall GetControlInfo(TControl * AControl, TRect Insets, int PreferredSize, int
RowCount);
```

Description

GetControlInfo is called automatically when the control bar needs to arrange the position of its bands. It generates an OnBandInfo (see page 12473) event to obtain information about a control band.

AControl is the control associated with the band being positioned.

Insets returns the margin between the edges of the control and the edges of its band.

PreferredSize returns the control's preferred width. (The control's width is reduced if it can't fit inside the control bar.)

RowCount returns the number of control rows that the control spans in the control bar.

See Also

OnBandInfo (see page 12473)

68.1.15.4.13 TCustomControlBar.GetPalette Method

Returns the handle to a palette for use by the control.

Pascal

```
function GetPalette: HPALETTE; override;
```

C++

```
virtual __fastcall HPALETTE GetPalette();
```

Description

As implemented in TControl, GetPalette returns a null handle, indicating that the control does not have a palette. Override GetPalette to implement derived objects that use a palette. Create a palette handle for the control by a call to the Windows API function CreatePalette. GetPalette should return the palette handle for the control.

Specifying the palette for a control tells the application that the control's palette needs to be realized and designates the palette to use for realization.

GetPalette is called only if the run-time video mode requires color palette support, for example, in 256-color mode.

See Also

PaletteChanged (see page 5775)

68.1.15.4.14 TCustomControlBar.GetSiteInfo Method

Generates an OnGetSiteInfo event

Pascal

```
procedure GetSiteInfo(Client: TControl; var InfluenceRect: TRect; MousePos: TPoint; var
CanDock: Boolean); override;
```

C++

```
virtual __fastcall GetSiteInfo(TControl * Client, TRect InfluenceRect, TPoint MousePos,
Boolean CanDock);
```

Description

GetSiteInfo is called automatically during drag-and-dock operations to initialize a TDragDockObject with information about where a dragged object can be docked, if at all.

Client is a potential docking client.

InfluenceRect returns the area in which the client would be docked.

MousePos is the current mouse position.

CanDock returns whether the client can dock to the windowed control.

As implemented in TWinControl, GetSiteInfo generates an influence rectangle equal to BoundsRect expanded by 10 pixels on every side, and then calls the OnGetSiteInfo event handler.

See Also

OnGetSiteInfo (see page 6223)

BoundsRect (see page 5803)

68.1.15.4.15 TCustomControlBar.HitTest Method

Returns the control at the specified coordinates.

Pascal

```
function HitTest(X: Integer; Y: Integer): TControl;
```

C++

```
__fastcall TControl * HitTest(int X, int Y);
```

Description

Call HitTest to get the control at the specified coordinates.

X and Y are the coordinates.

See Also

MouseDown (see page 12465)

68.1.15.4.16 TCustomControlBar.MouseDown Method

Handles mouse button clicks.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

MouseDown is called automatically when the user clicks the control bar with the mouse. After generating an OnMouseDown event, MouseDown determines whether a band is being dragged and if so, computes its position.

Button indicates which button was clicked.

Shift indicates the state of the Alt, Ctrl, and Shift keys and the state of the mouse buttons.

X and Y are the position where the mouse was clicked.

See Also

OnMouseDown (see page 5862)

HitTest (see page 12464)

68.1.15.4.17 TCustomControlBar.MouseMove Method

Handles mouse movements.

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

MouseMove generates an OnMouseMove event and then performs special processing if a control is being undocked from the control bar or moved within the control bar.

Shift indicates the state of the Alt, Ctrl, and Shift keys and the state of the mouse buttons.

X and Y are the current position of the mouse.

See Also

OnMouseMove (see page 5865)

AutoDrag (see page 12468)

68.1.15.4.18 TCustomControlBar.MouseUp Method

Handles releasing of a mouse button.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

MouseUp handles the dropping of a control onto the control bar and, once the control (if any) is docked, generates an OnMouseUp event.

Button indicates which button was released.

Shift indicates the state of the Alt, Ctrl, and Shift keys and the state of the mouse buttons.

X and Y are the position where the mouse was released.

See Also

OnMouseUp (see page 5867)

68.1.15.4.19 TCustomControlBar.Paint Method

Paints the control bar.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Paint is called automatically when the control bar needs to paint itself.

To customize the painting, either override the Paint method, or specify options in the OnPaint (see page 12474) and OnBandPaint (see page 12473) event handlers.

See Also

OnPaint (see page 12474)

onbandpaint (see page 12473)

PaintControlFrame (see page 12466)

68.1.15.4.20 TCustomControlBar.PaintControlFrame Method

Paints a control band's frame.

Pascal

```
procedure PaintControlFrame(Canvas: TCanvas; AControl: TControl; var ARect: TRect); virtual;
```

C++

```
virtual __fastcall PaintControlFrame(TCanvas Canvas, TControl * AControl, TRect ARect);
```

Description

The Paint (see page 12466) method calls PaintControlFrame for each band to render its image. PaintControlFrame generates an OnBandPaint (see page 12473) event, and then draws the frame and grabber for the band, depending on the values returned by the event handler.

Canvas is the control bar's canvas.

AControl is the control on the band to be painted.

ARect indicates the dimensions of the band.

See Also

OnPaint (see page 12474)

onbandpaint (see page 12473)

PaintControlFrame

68.1.15.4.21 TCustomControlBar.StickControls Method

Places controls in their final positions after a control is moved in or dropped on the control bar.

Pascal

```
procedure StickControls; virtual;
```

C++

```
virtual __fastcall StickControls();
```

Description

Override StickControls to specify how controls are set in their final positions after a control is moved in or dropped on the control bar.

See Also

rowsnap (see page 12472)

68.1.15.5 TCustomControlBar Properties

The properties of the TCustomControlBar class are listed here.

Protected Properties

	Name	Description
	AutoDock (see page 12467)	Determines whether the control bar temporarily docks drag clients to provide visual feedback.
	AutoDrag (see page 12468)	Specifies whether control bands can be undocked by dragging.
	AutoSize (see page 12468)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelKind (see page 12469)	Specifies the control's bevel style.
	CornerEdge (see page 12469)	This is CornerEdge, a member of class TCustomControlBar.
	DockSite (see page 12469)	Specifies whether the control can be the target of drag-and-dock operations.
	DrawingStyle (see page 12470)	This is DrawingStyle, a member of class TCustomControlBar.
	GradientDirection (see page 12471)	This is GradientDirection, a member of class TCustomControlBar.
	GradientEndColor (see page 12471)	This is GradientEndColor, a member of class TCustomControlBar.
	GradientStartColor (see page 12471)	This is GradientStartColor, a member of class TCustomControlBar.
	RowSize (see page 12472)	This is RowSize, a member of class TCustomControlBar.
	RowSnap (see page 12472)	This is RowSnap, a member of class TCustomControlBar.

Public Properties

	Name	Description
	Picture (see page 12471)	This is Picture, a member of class TCustomControlBar.

68.1.15.5.1 TCustomControlBar.AutoDock Property

Determines whether the control bar temporarily docks drag clients to provide visual feedback.

Pascal

```
property AutoDock: Boolean;
```

C++

```
__property Boolean AutoDock;
```

Description

If AutoDock is true (the default), when a dockable object is dragged over the control bar, the control bar temporarily docks the object so that the user can see the impact of the drag operation. That is, the user can see the dock position and how other docked controls will be repositioned to make room. When the user releases the mouse, the dragged object is docked at the current spot, and any temporary dockings are canceled. However, if the user cancels the drag-and-dock operation, the dragged object reverts to its last temporary dock site rather than to its location at the start of the drag operation.

If AutoDock is false, the control bar does not temporarily dock dragged objects. Instead, the user only sees the dragging rectangle as it passes over the control bar, with no feedback as to the adjustments the control bar would make if the dragged object were released.

See Also

DockClients (see page 6201)

DockSite (see page 6204)

OnDockOver (see page 6216)

68.1.15.5.2 TCustomControlBar.AutoDrag Property

Specifies whether control bands can be undocked by dragging.

Pascal

```
property AutoDrag: Boolean;
```

C++

```
__property Boolean AutoDrag;
```

Description

Use AutoDrag to specify whether control bands are automatically undocked when the user drags them. If AutoDrag is true, control bands (TToolBar objects) are undocked when dragged off the control bar. Otherwise, control bands are kept within the control bar.

See Also

DragMode (see page 5819)

OnBandDrag (see page 12472)

onbandmove (see page 12473)

68.1.15.5.3 TCustomControlBar.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints (see page 5812)

CanAutoSize (see page 5741)

68.1.15.5.4 TCustomControlBar.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

68.1.15.5.5 TCustomControlBar.CornerEdge Property

Pascal

```
property CornerEdge: TCornerEdge;
```

C++

```
__property TCornerEdge CornerEdge;
```

Description

This is CornerEdge, a member of class TCustomControlBar.

68.1.15.5.6 TCustomControlBar.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite: Boolean;
```

C++

```
__property Boolean DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

68.1.15.5.7 TCustomControlBar.DrawingStyle Property

Pascal

```
property DrawingStyle: TBandDrawingStyle;
```

C++

```
__property TBandDrawingStyle DrawingStyle;
```

Description

This is DrawingStyle, a member of class TCustomControlBar.

68.1.15.5.8 TCustomControlBar.GradientDirection Property

Pascal

```
property GradientDirection: TGradientDirection;
```

C++

```
__property TGradientDirection GradientDirection;
```

Description

This is GradientDirection, a member of class TCustomControlBar.

68.1.15.5.9 TCustomControlBar.GradientEndColor Property

Pascal

```
property GradientEndColor: TColor;
```

C++

```
__property TColor GradientEndColor;
```

Description

This is GradientEndColor, a member of class TCustomControlBar.

68.1.15.5.10 TCustomControlBar.GradientStartColor Property

Pascal

```
property GradientStartColor: TColor;
```

C++

```
__property TColor GradientStartColor;
```

Description

This is GradientStartColor, a member of class TCustomControlBar.

68.1.15.5.11 TCustomControlBar.Picture Property

Pascal

```
property Picture: TPicture;
```

C++

```
__property TPicture Picture;
```

Description

This is Picture, a member of class TCustomControlBar.

68.1.15.5.12 TCustomControlBar.RowSize Property

Pascal

```
property RowSize: TRowSize;
```

C++

```
__property TRowSize RowSize;
```

Description

This is RowSize, a member of class TCustomControlBar.

68.1.15.5.13 TCustomControlBar.RowSnap Property

Pascal

```
property RowSnap: Boolean;
```

C++

```
__property Boolean RowSnap;
```

Description

This is RowSnap, a member of class TCustomControlBar.

68.1.15.6 TCustomControlBar Events

The events of the TCustomControlBar class are listed here.

Protected Events

	Name	Description
	OnBandDrag (see page 12472)	Occurs when the user starts a drag operation on a specified control.
	OnBandInfo (see page 12473)	Occurs when the control bar adjusts the docking position of a band.
	OnBandMove (see page 12473)	This is OnBandMove, a member of class TCustomControlBar.
	OnBandPaint (see page 12473)	This is OnBandPaint, a member of class TCustomControlBar.
	OnBeginBandMove (see page 12474)	This is OnBeginBandMove, a member of class TCustomControlBar.
	OnEndBandMove (see page 12474)	This is OnEndBandMove, a member of class TCustomControlBar.
	OnPaint (see page 12474)	Occurs when the control bar is painted.

68.1.15.6.1 TCustomControlBar.OnBandDrag Event

Occurs when the user starts a drag operation on a specified control.

Pascal

```
property OnBandDrag: TBandDragEvent;
```

C++

```
__property TBandDragEvent OnBandDrag;
```

Description

Use OnBandDrag to conditionally allow the user to start dragging a control in the control bar.

The event handler indicates the control that the user is trying to drag. Change its Drag parameter to false to prevent the drag operation. If Drag remains true on exit from the event handler, the drag operation proceeds.

See Also

OnStartDock (see page 5871)

onbandmove (see page 12473)

68.1.15.6.2 TCustomControlBar.OnBandInfo Event

Occurs when the control bar adjusts the docking position of a band.

Pascal

```
property OnBandInfo: TBandInfoEvent;
```

C++

```
__property TBandInfoEvent OnBandInfo;
```

Description

Write code in the OnBandInfo event handler to provide information to the control bar that it can use when arranging its bands. The OnBandInfo event handler has several parameters that can be used to return information about how the band should be positioned, including its size, inset, and number of rows.

See Also

TBDockHeight (see page 5842)

onbandmove (see page 12473)

LRDockWidth (see page 5831)

Height (see page 5825)

Width (see page 5849)

68.1.15.6.3 TCustomControlBar.OnBandMove Event

Pascal

```
property OnBandMove: TBandMoveEvent;
```

C++

```
__property TBandMoveEvent OnBandMove;
```

Description

This is OnBandMove, a member of class TCustomControlBar.

68.1.15.6.4 TCustomControlBar.OnBandPaint Event

Pascal

```
property OnBandPaint: TBandPaintEvent;
```

C++

```
__property TBandPaintEvent OnBandPaint;
```

Description

This is OnBandPaint, a member of class TCustomControlBar.

68.1.15.6.5 TCustomControlBar.OnBeginBandMove Event

Pascal

```
property OnBeginBandMove: TBeginBandMoveEvent;
```

C++

```
__property TBeginBandMoveEvent OnBeginBandMove;
```

Description

This is OnBeginBandMove, a member of class TCustomControlBar.

68.1.15.6.6 TCustomControlBar.OnEndBandMove Event

Pascal

```
property OnEndBandMove: TEndBandMoveEvent;
```

C++

```
__property TEndBandMoveEvent OnEndBandMove;
```

Description

This is OnEndBandMove, a member of class TCustomControlBar.

68.1.15.6.7 TCustomControlBar.OnPaint Event

Occurs when the control bar is painted.

Pascal

```
property OnPaint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPaint;
```

Description

Write an OnPaint event handler to change the way the control bar is rendered. When a control bar receives a Windows paint message, it first draws the background picture, if any, and then generates an OnPaint event. After the OnPaint event exits, the control bar generates an OnBandPaint (see page 12473) event for each band, finally rendering the grabbers and frames for each band.

Use the Canvas property to draw the control bar's image. Use the ClientRect property to obtain the dimensions on which to draw.

If the custom drawing is intended to change the appearance of individual bands, use the OnBandPaint (see page 12473) event instead.

See Also

onbandpaint (see page 12473)

Canvas (see page 5921)

ClientRect

picture (see page 12471)

68.1.16 TCustomFlowPanel Class

The base class for all flow panel components.

Class Hierarchy

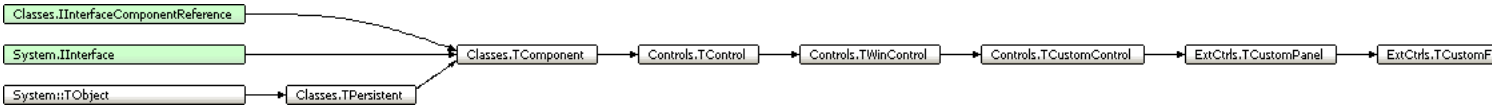

Pascal

```
TCustomFlowPanel = class(TCustomPanel);
```

C++

```
class TCustomFlowPanel : public TCustomPanel;
```

File

ExtCtrls

Description

Use TCustomFlowPanel as a base class when defining flow panels that include other controls as constituent parts.

See Also

TFlowPanel (see page 12579)

68.1.16.1 TCustomFlowPanel Members

The following tables list the members exposed by TCustomFlowPanel.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.

	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 12537)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 12487)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.

	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanAutoSize (see page 12537)	Adjusts the way the panel responds to resize attempts.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 12537)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.

	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.

	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 5920)	Renders the image of a custom control.
V	Paint (see page 12538)	Renders the image of the panel.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.

	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBackground (see page 12539)	This is SetParentBackground, a member of class TCustomPanel.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	Alignment (see page 12539)	Determines how the caption is aligned within the panel.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelInner (see page 12540)	Determines the style of the inner bevel of a panel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelOuter (see page 12541)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BevelWidth (see page 12542)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BorderStyle (see page 12543)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 6195)	Specifies the width of the control's border.

	BorderWidth (see page 12543)	Specifies the distance, in pixels, between the outer and inner bevels.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Color (see page 12544)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	FullRepaint (see page 12545)	Determines how the panel repaints itself when it is resized.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	Locked (see page 12545)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 12546)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	VerticalAlignment (see page 12547)	Sets the vertical position of the caption.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 12486)	Creates and initializes an instance of TCustomPanel.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 12486)	Destroys an instance of TCustomControl.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetChildren (see page 12487)	Calls a specified method for each child of the control.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlIndex (see page 12488)	This is GetControlIndex, a member of class TCustomFlowPanel.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
V	GetControlsAlignment (see page 12538)	Returns the alignment of text in the panel.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.

V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetControlIndex (see page 12488)	This is SetControlIndex, a member of class TCustomFlowPanel.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AutoWrap (see page 12488)	Specifies whether controls are automatically wrapped to fit on the flow panel.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	FlowStyle (see page 12489)	Specifies the starting point and direction for controls added to a flow panel.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentBackground (see page 12546)	Determines whether control uses parent's theme background.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.16.2 TCustomFlowPanel.Create Constructor

Creates and initializes an instance of TCustomPanel.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomFlowPanel(TComponent * AOwner);
```

Description

Call Create to instantiate a panel at runtime. Panels placed in forms at design time do not need to be explicitly created because they are created automatically.

AOwner is the component, typically a form, that is responsible for freeing the panel.

See Also

Owner (see page 2807)

68.1.16.3 TCustomFlowPanel.Destroy Destructor

Destroys an instance of TCustomControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomFlowPanel();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the control is not nil, and only then calls Destroy.

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Destroys the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

Free (see page 22300)

delete

Create (see page 5918)

68.1.16.4 TCustomFlowPanel Methods

The methods of the TCustomFlowPanel class are listed here.

Protected Methods

	Name	Description
	AlignControls (see page 12487)	Aligns any controls for which the control is the parent within a specified area of the control.

Public Methods

	Name	Description
	GetChildren (see page 12487)	Calls a specified method for each child of the control.
	GetControlIndex (see page 12488)	This is GetControlIndex, a member of class TCustomFlowPanel.
	SetControlIndex (see page 12488)	This is SetControlIndex, a member of class TCustomFlowPanel.

68.1.16.4.1 TCustomFlowPanel.AlignControls Method

Aligns any controls for which the control is the parent within a specified area of the control.

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

Use AlignControls to align all controls within an area of a form, panel, group box, scroll box, or any container control. Specify the area in which to align the controls as the value of the Rect parameter.

AlignControls uses the Align property value for each child control to determine how to align it. The AControl parameter can be nil (Delphi) or NULL (C++). If you specify a control in AControl, that control takes precedence in alignment over other, similarly-aligned controls.

See Also

Align (see page 5798)

68.1.16.4.2 TCustomFlowPanel.GetChildren Method

Calls a specified method for each child of the control.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the streaming system that loads and saves components. Applications seldom need to call this routine.

GetChildren executes the callback specified by the Proc parameter for every child control listed by the Controls property that lists the Root parameter as its Owner.

Override GetChildren to limit or augment which child controls are saved with the control. When overriding GetChildren, call the procedure passed as the Proc parameter for every child control that should be saved. The Root parameter indicates the

component (usually a form) that owns the control.

See Also

Controls (see page 6197)

Owner (see page 2807)

68.1.16.4.3 TCustomFlowPanel.GetControlIndex Method

Pascal

```
function GetControlIndex(AControl: TControl): Integer;
```

C++

```
__fastcall int GetControlIndex(TControl * AControl);
```

Description

This is GetControlIndex, a member of class TCustomFlowPanel.

68.1.16.4.4 TCustomFlowPanel.SetControlIndex Method

Pascal

```
procedure SetControlIndex(AControl: TControl; Index: Integer);
```

C++

```
__fastcall SetControlIndex(TControl * AControl, int Index);
```

Description

This is SetControlIndex, a member of class TCustomFlowPanel.

68.1.16.5 TCustomFlowPanel Properties

The properties of the TCustomFlowPanel class are listed here.

Public Properties

	Name	Description
	AutoWrap (see page 12488)	Specifies whether controls are automatically wrapped to fit on the flow panel.
	FlowStyle (see page 12489)	Specifies the starting point and direction for controls added to a flow panel.

68.1.16.5.1 TCustomFlowPanel.AutoWrap Property

Specifies whether controls are automatically wrapped to fit on the flow panel.

Pascal

```
property AutoWrap: Boolean;
```

C++

```
__property Boolean AutoWrap;
```

Description

If set to True, a new control is automatically placed in a location on the flow panel in which it is visible. If set to False, a new control continues in the same direction as the previous control, even if it would no longer be visible on the panel. You can increase the size of the panel to view controls that are not automatically wrapped to the next line or column.

68.1.16.5.2 TCustomFlowPanel.FlowStyle Property

Specifies the starting point and direction for controls added to a flow panel.

Pascal

```
property FlowStyle: TFlowStyle;
```

C++

```
__property TFlowStyle FlowStyle;
```

Description

Use FlowStyle to set the starting position for the first control added to a flow panel, as well as the subsequent position for additional controls.

Flow Style	Description
BottomTopLeftRight	The first control is at the bottom left corner of the panel. Subsequent controls are added from bottom to top and then from left to right.
BottomTopRightLeft	The first control is at the bottom right corner of the panel. Subsequent controls are added from bottom to top and then from right to left.
LeftRightBottomTop	The first control is at the bottom left corner of the panel. Subsequent controls are added from left to right and then from bottom to top.
LeftRightTopBottom	The first control is at the top left corner of the panel. Subsequent controls are added from left to right and then from top to bottom. This is the default setting.
RightLeftBottomTop	The first control is at the bottom right corner of the panel. Subsequent controls are added from right to left and then from bottom to top.
RightLeftTopBottom	The first control is at the top right corner of the panel. Subsequent controls are added from right to left and then from top to bottom.
TopBottomLeftRight	The first control is at the top left corner of the panel. Subsequent controls are added from top to bottom and then from left to right.
TopBottomRightLeft	The first control is at the top right corner of the panel. Subsequent controls are added from top to bottom and then from right to left.

68.1.17 TCustomGridPanel Class

The base class for all grid panel components.

Class Hierarchy

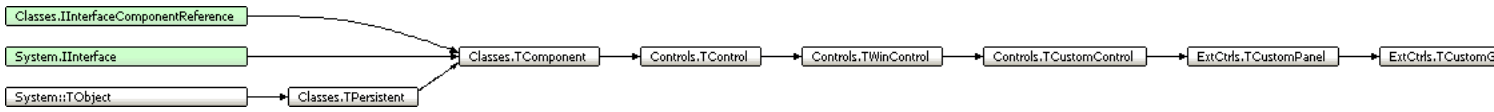

Pascal

```
TCustomGridPanel = class(TCustomPanel);
```

C++

```
class TCustomGridPanel : public TCustomPanel;
```

File

ExtCtrls

Description

Use TCustomGridPanel as a base class when defining grid panels that include other controls as constituent parts.

See Also

TGridPanel (see page 12660)

68.1.17.1 TCustomGridPanel Members

The following tables list the members exposed by TCustomGridPanel.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.

	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 12537)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 12502)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	AutoAddColumn (see page 12502)	This is AutoAddColumn, a member of class TCustomGridPanel.
	AutoAddRow (see page 12503)	This is AutoAddRow, a member of class TCustomGridPanel.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanAutoSize (see page 12537)	Adjusts the way the panel responds to resize attempts.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	CellIndexToCell (see page 12503)	This is CellIndexToCell, a member of class TCustomGridPanel.
	CellToCellIndex (see page 12503)	This is CellToCellIndex, a member of class TCustomGridPanel.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.

	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 12537)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.

	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 12504)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 12505)	Renders the image of the panel.

	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveEmptyAutoAddColumns (see page 12505)	This is RemoveEmptyAutoAddColumns, a member of class TCustomGridPanel.
	RemoveEmptyAutoAddRows (see page 12505)	This is RemoveEmptyAutoAddRows, a member of class TCustomGridPanel.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBackground (see page 12539)	This is SetParentBackground, a member of class TCustomPanel.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.

	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 12505)	This is UpdateControlOriginalParentSize, a member of class TCustomGridPanel.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	Alignment (see page 12539)	Determines how the caption is aligned within the panel.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelInner (see page 12540)	Determines the style of the inner bevel of a panel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelOuter (see page 12541)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BevelWidth (see page 12542)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BorderStyle (see page 12543)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	BorderWidth (see page 12543)	Specifies the distance, in pixels, between the outer and inner bevels.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Color (see page 12544)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.

	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	FullRepaint (see page 12545)	Determines how the panel repaints itself when it is resized.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	Locked (see page 12545)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 12546)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	VerticalAlignment (see page 12547)	Sets the vertical position of the caption.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 12500)	Creates and initializes an instance of TCustomPanel.

	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 12501)	Destroys an instance of TCustomControl.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetChildren (see page 12503)	Calls a specified method for each child of the control.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
V	GetControlsAlignment (see page 12538)	Returns the alignment of text in the panel.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.

	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsColumnEmpty (see page 12504)	This is IsColumnEmpty, a member of class TCustomGridPanel.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	IsRowEmpty (see page 12504)	This is IsRowEmpty, a member of class TCustomGridPanel.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.

V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	CellCount (see page 12506)	This is CellCount, a member of class TCustomGridPanel.
R	CellRect (see page 12506)	Returns the screen coordinates of a cell in the grid.
R	CellSize (see page 12507)	This is CellSize, a member of class TCustomGridPanel.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	ColumnCollection (see page 12507)	Sets the number and size of the columns on the grid panel.
R	ColumnSpanIndex (see page 12507)	This is ColumnSpanIndex, a member of class TCustomGridPanel.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlCollection (see page 12507)	This is ControlCollection, a member of class TCustomGridPanel.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
	ExpandStyle (see page 12507)	Specifies whether rows or columns are added to a filled grid panel to accommodate additional controls.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.

R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentBackground (see page 12546)	Determines whether control uses parent's theme background.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	RowCollection (see page 12508)	Sets the number and size of the rows on the grid panel.
R	RowSpanIndex (see page 12508)	This is RowSpanIndex, a member of class TCustomGridPanel.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.17.2 TCustomGridPanel.Create Constructor

Creates and initializes an instance of TCustomPanel.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomGridPanel(TComponent * AOwner);
```

Description

Call Create to instantiate a panel at runtime. Panels placed in forms at design time do not need to be explicitly created because they are created automatically.

AOwner is the component, typically a form, that is responsible for freeing the panel.

See Also

Owner (see page 2807)

68.1.17.3 TCustomGridPanel.Destroy Destructor

Destroys an instance of TCustomControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomGridPanel();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the control is not nil, and only then calls Destroy.

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Destroys the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

Free (see page 22300)

delete

Create (see page 5918)

68.1.17.4 TCustomGridPanel Methods

The methods of the TCustomGridPanel class are listed here.

Protected Methods

	Name	Description
	AlignControls (see page 12502)	Aligns any controls for which the control is the parent within a specified area of the control.
	AutoAddColumn (see page 12502)	This is AutoAddColumn, a member of class TCustomGridPanel.
	AutoAddRow (see page 12503)	This is AutoAddRow, a member of class TCustomGridPanel.
	CellIndexToCell (see page 12503)	This is CellIndexToCell, a member of class TCustomGridPanel.
	CellToCellIndex (see page 12503)	This is CellToCellIndex, a member of class TCustomGridPanel.
	Loaded (see page 12504)	Initializes the control after it is loaded from a stream.
	Paint (see page 12505)	Renders the image of the panel.
	RemoveEmptyAutoAddColumns (see page 12505)	This is RemoveEmptyAutoAddColumns, a member of class TCustomGridPanel.
	RemoveEmptyAutoAddRows (see page 12505)	This is RemoveEmptyAutoAddRows, a member of class TCustomGridPanel.
	UpdateControlOriginalParentSize (see page 12505)	This is UpdateControlOriginalParentSize, a member of class TCustomGridPanel.

Public Methods

	Name	Description
	GetChildren (see page 12503)	Calls a specified method for each child of the control.
	IsColumnEmpty (see page 12504)	This is IsColumnEmpty, a member of class TCustomGridPanel.
	IsRowEmpty (see page 12504)	This is IsRowEmpty, a member of class TCustomGridPanel.

68.1.17.4.1 TCustomGridPanel.AlignControls Method

Aligns any controls for which the control is the parent within a specified area of the control.

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

Use AlignControls to align all controls within an area of a form, panel, group box, scroll box, or any container control. Specify the area in which to align the controls as the value of the Rect parameter.

AlignControls uses the Align property value for each child control to determine how to align it. The AControl parameter can be nil (Delphi) or NULL (C++). If you specify a control in AControl, that control takes precedence in alignment over other, similarly-aligned controls.

See Also

Align (see page 5798)

68.1.17.4.2 TCustomGridPanel.AutoAddColumn Method

Pascal

```
function AutoAddColumn: TColumnItem;
```

C++

```
__fastcall TColumnItem AutoAddColumn();
```

Description

This is AutoAddColumn, a member of class TCustomGridPanel.

68.1.17.4.3 TCustomGridPanel.AutoAddRow Method

Pascal

```
function AutoAddRow: TRowItem;
```

C++

```
__fastcall TRowItem AutoAddRow();
```

Description

This is AutoAddRow, a member of class TCustomGridPanel.

68.1.17.4.4 TCustomGridPanel.CellIndexToCell Method

Pascal

```
procedure CellIndexToCell(AIndex: Integer; var AColumn: Integer; var ARow: Integer);
```

C++

```
__fastcall CellIndexToCell(int AIndex, int AColumn, int ARow);
```

Description

This is CellIndexToCell, a member of class TCustomGridPanel.

68.1.17.4.5 TCustomGridPanel.CellToCellIndex Method

Pascal

```
function CellToCellIndex(AColumn: Integer; ARow: Integer): Integer;
```

C++

```
__fastcall int CellToCellIndex(int AColumn, int ARow);
```

Description

This is CellToCellIndex, a member of class TCustomGridPanel.

68.1.17.4.6 TCustomGridPanel.GetChildren Method

Calls a specified method for each child of the control.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the streaming system that loads and saves components. Applications seldom need to call this routine.

GetChildren executes the callback specified by the Proc parameter for every child control listed by the Controls property that lists the Root parameter as its Owner.

Override GetChildren to limit or augment which child controls are saved with the control. When overriding GetChildren, call the procedure passed as the Proc parameter for every child control that should be saved. The Root parameter indicates the component (usually a form) that owns the control.

See Also

Controls (see page 6197)

Owner (see page 2807)

68.1.17.4.7 TCustomGridPanel.IsColumnEmpty Method

Pascal

```
function IsColumnEmpty(AColumn: Integer): Boolean;
```

C++

```
__fastcall Boolean IsColumnEmpty(int AColumn);
```

Description

This is IsColumnEmpty, a member of class TCustomGridPanel.

68.1.17.4.8 TCustomGridPanel.IsRowEmpty Method

Pascal

```
function IsRowEmpty(ARow: Integer): Boolean;
```

C++

```
__fastcall Boolean IsRowEmpty(int ARow);
```

Description

This is IsRowEmpty, a member of class TCustomGridPanel.

68.1.17.4.9 TCustomGridPanel.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded (see page 2782)

Action (see page 5797)

ActionChange (see page 5737)

68.1.17.4.10 TCustomGridPanel.Paint Method

Renders the image of the panel.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Override the Paint method to change the way the panel is drawn. The Paint method for TCustomPanel draws the bevels of the panel, sets the brush and font for the panel, and fills in the interior of the panel. Finally, Paint writes the value of Caption on the panel.

See Also

BevelInner (see page 12540)

BevelOuter (see page 12541)

Brush (see page 6195)

Caption (see page 5806)

Canvas (see page 5921)

68.1.17.4.11 TCustomGridPanel.RemoveEmptyAutoAddColumns Method

Pascal

```
procedure RemoveEmptyAutoAddColumns;
```

C++

```
__fastcall RemoveEmptyAutoAddColumns();
```

Description

This is RemoveEmptyAutoAddColumns, a member of class TCustomGridPanel.

68.1.17.4.12 TCustomGridPanel.RemoveEmptyAutoAddRows Method

Pascal

```
procedure RemoveEmptyAutoAddRows;
```

C++

```
__fastcall RemoveEmptyAutoAddRows();
```

Description

This is RemoveEmptyAutoAddRows, a member of class TCustomGridPanel.

68.1.17.4.13 TCustomGridPanel.UpdateControlOriginalParentSize Method

Pascal

```
procedure UpdateControlOriginalParentSize(AControl: TControl; var AOriginalParentSize:
TPoint); override;
```

C++

```
virtual __fastcall UpdateControlOriginalParentSize(TControl * AControl, TPoint
AOriginalParentSize);
```

Description

This is UpdateControlOriginalParentSize, a member of class TCustomGridPanel.

68.1.17.5 TCustomGridPanel Properties

The properties of the TCustomGridPanel class are listed here.

Public Properties

	Name	Description
R	CellCount (see page 12506)	This is CellCount, a member of class TCustomGridPanel.
R	CellRect (see page 12506)	Returns the screen coordinates of a cell in the grid.
R	CellSize (see page 12507)	This is CellSize, a member of class TCustomGridPanel.
	ColumnCollection (see page 12507)	Sets the number and size of the columns on the grid panel.
R	ColumnSpanIndex (see page 12507)	This is ColumnSpanIndex, a member of class TCustomGridPanel.
	ControlCollection (see page 12507)	This is ControlCollection, a member of class TCustomGridPanel.
	ExpandStyle (see page 12507)	Specifies whether rows or columns are added to a filled grid panel to accommodate additional controls.
	RowCollection (see page 12508)	Sets the number and size of the rows on the grid panel.
R	RowSpanIndex (see page 12508)	This is RowSpanIndex, a member of class TCustomGridPanel.

68.1.17.5.1 TCustomGridPanel.CellCount Property

Pascal

```
property CellCount: Integer;
```

C++

```
__property int CellCount;
```

Description

This is CellCount, a member of class TCustomGridPanel.

68.1.17.5.2 TCustomGridPanel.CellRect Property

Returns the screen coordinates of a cell in the grid.

Pascal

```
property CellRect [AColumn, ARow: Integer]: TRect;
```

C++

```
__property TRect CellRect[int AColumn, ARow];
```

Description

Use the CellRect property to return the screen coordinates of a cell in the grid.

68.1.17.5.3 TCustomGridPanel.CellSize Property

Pascal

```
property CellSize [AColumn, ARow: Integer]: TPoint;
```

C++

```
__property TPoint CellSize[int AColumn, ARow];
```

Description

This is CellSize, a member of class TCustomGridPanel.

68.1.17.5.4 TCustomGridPanel.ColumnCollection Property

Sets the number and size of the columns on the grid panel.

Pascal

```
property ColumnCollection: TColumnCollection;
```

C++

```
__property TColumnCollection ColumnCollection;
```

Description

Use the ColumnCollection property to set the number of columns as well as the size of each column. You can use the SizeStyle property to specify whether column sizes are based on a percentage of the available width, based on an absolute value, or automatically sized.

68.1.17.5.5 TCustomGridPanel.ColumnSpanIndex Property

Pascal

```
property ColumnSpanIndex [AColumn, ARow: Integer]: Integer;
```

C++

```
__property int ColumnSpanIndex[int AColumn, ARow];
```

Description

This is ColumnSpanIndex, a member of class TCustomGridPanel.

68.1.17.5.6 TCustomGridPanel.ControlCollection Property

Pascal

```
property ControlCollection: TControlCollection;
```

C++

```
__property TControlCollection ControlCollection;
```

Description

This is ControlCollection, a member of class TCustomGridPanel.

68.1.17.5.7 TCustomGridPanel.ExpandStyle Property

Specifies whether rows or columns are added to a filled grid panel to accommodate additional controls.

Pascal

```
property ExpandStyle: TExpandStyle;
```

C++

```
__property TExpandStyle ExpandStyle;
```

Description

Use the ExpandStyle property to specify how a grid panel is expanded when each cell of the grid contains a control, and a new control is added.

68.1.17.5.8 TCustomGridPanel.RowCollection Property

Sets the number and size of the rows on the grid panel.

Pascal

```
property RowCollection: TRowCollection;
```

C++

```
__property TRowCollection RowCollection;
```

Description

Use the RowCollection property to set the number of rows as well as the size of each row. You can use the SizeStyle property to specify whether row sizes are based on a percentage of the available width, based on an absolute value, or automatically sized.

68.1.17.5.9 TCustomGridPanel.RowSpanIndex Property

Pascal

```
property RowSpanIndex [AColumn, ARow: Integer]: Integer;
```

C++

```
__property int RowSpanIndex[int AColumn, ARow];
```

Description

This is RowSpanIndex, a member of class TCustomGridPanel.

68.1.18 TCustomLabeledEdit Class

TCustomLabeledEdit is the base class for edit controls that have an integrated label.

Class Hierarchy

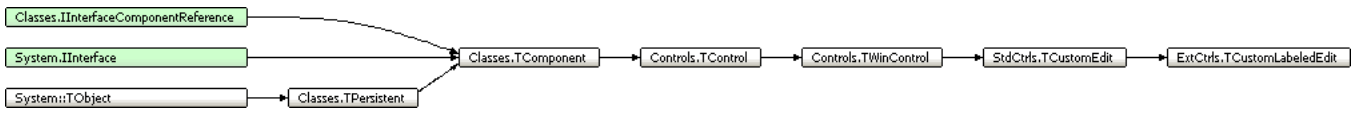

Pascal

```
TCustomLabeledEdit = class(TCustomEdit);
```

C++

```
class TCustomLabeledEdit : public TCustomEdit;
```

File

ExtCtrls

Description

TCustomLabeledEdit is the base class for edit controls that have an associated label. This class introduces new properties for managing a label that sits on one side of the edit control.

Do not create instances of TCustomLabeledEdit. Use TCustomLabeledEdit as a base class when declaring edit controls that include a label. Properties and methods of TCustomLabeledEdit provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TEdit (see page 21402)

TLabeledEdit (see page 12833)

TBoundLabel (see page 12123)

68.1.18.1 TCustomLabeledEdit Members

The following tables list the members exposed by TCustomLabeledEdit.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnChange (see page 21275)	Occurs when the text for the edit control may have changed.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.

	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21245)	Generates an OnChange (see page 21275) event.

	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	CMBidimodechanged (see page 12521)	This is CMBidimodechanged, a member of class TCustomLabeledEdit.
	CMEnabledchanged (see page 12521)	Responds when the Enabled property changes.
	CMVisiblechanged (see page 12521)	Responds when the Visible property changes.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 21248)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWindowHandle (see page 21248)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CreateWnd (see page 21248)	Creates a Windows control corresponding to the edit control.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
V	DestroyWnd (see page 21251)	Destroys the Windows control that corresponds to the edit control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.

	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
V	DoSetMaxLength (see page 21251)	Sets the MaxLength (see page 21263) property.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
V	GetSelLength (see page 21252)	Returns the SelLength (see page 21265) property.
V	GetSelStart (see page 21252)	Returns the SelStart (see page 21269) property.
V	GetSelText (see page 21252)	Returns the SelText (see page 21272) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.

	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 12522)	Responds to notifications when objects are created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetAutoSize (see page 21257)	Sets the AutoSize (see page 21260) property.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.

	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 12523)	Sets the name of the control.
V	SetParent (see page 12523)	Sets the value of the Parent property.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetSelLength (see page 21257)	Sets the SelLength (see page 21265) property.
V	SetSelStart (see page 21257)	Sets the SelStart (see page 21269) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSelect (see page 21260)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AutoSize (see page 21260)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 21261)	Determines whether the edit control has a single line border around the client area.

	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	CharCase (see page 21262)	Determines the case of the text within the edit control.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	HideSelection (see page 21262)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MaxLength (see page 21263)	Specifies the maximum number of characters the user can enter into the edit control.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	OEMConvert (see page 21264)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 21264)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PasswordChar (see page 21265)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.

	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 21245)	Deletes all text from the edit control.
	ClearSelection (see page 21245)	Removes the selected text from the edit control.
	ClearUndo (see page 21246)	Clears the undo buffer so that no changes to the text can be backed out.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopyToClipboard (see page 21246)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
V	Create (see page 12520)	Creates an instance of TCustomLabeledEdit (see page 12508).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CutToClipboard (see page 21249)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 21250)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetSelTextBuf (see page 21253)	Copies the selected text into a buffer and returns the number of characters copied.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	PasteFromClipboard (see page 21254)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectAll (see page 21255)	Selects all text in the edit control.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 12522)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSelTextBuf (see page 21258)	Replaces the selected text with a null-terminated string.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	SetupInternalLabel (see page 12523)	Initializes the control that implements the EditLabel (see page 12524) property.
	Show (see page 5788)	Makes a control visible.
	Undo (see page 21259)	Backs out all changes in the undo buffer.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	CanUndo (see page 21262)	Indicates whether the edit control contains changes that can be backed out.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	EditLabel (see page 12524)	Specifies the control that implements the label.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LabelPosition (see page 12524)	Specifies the position of the label relative to the edit control.
	LabelSpacing (see page 12525)	Specifies the distance, in pixels, between the label and the edit region.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Modified (see page 21263)	Indicates whether the user edited the text of the edit control.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ReadOnly (see page 21265)	Determines whether the user can change the text of the edit control.
	SelLength (see page 21265)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 21269)	Specifies the position of the first selected character in the text.
	SelText (see page 21272)	Specifies the selected portion of the edit control's text.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	Text (see page 21273)	Contains a text string associated with the control.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	TabStop (see page 21273)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.18.2 TCustomLabeledEdit.Create Constructor

Creates an instance of TCustomLabeledEdit (see page 12508).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomLabeledEdit(TComponent * AOwner);
```

Description

Call Create to instantiate a labeled edit control. AOwner specifies a component, typically the form, that is responsible for freeing the labeled edit control.

See Also

Owner (see page 2807)

68.1.18.3 TCustomLabeledEdit Methods

The methods of the TCustomLabeledEdit class are listed here.

Protected Methods

	Name	Description
	CMBidimodechanged (see page 12521)	This is CMBidimodechanged, a member of class TCustomLabeledEdit.
	CMEnabledchanged (see page 12521)	Responds when the Enabled property changes.
	CMVisiblechanged (see page 12521)	Responds when the Visible property changes.
V	Notification (see page 12522)	Responds to notifications when objects are created or destroyed.
V	SetName (see page 12523)	Sets the name of the control.
V	SetParent (see page 12523)	Sets the value of the Parent property.

Public Methods

	Name	Description
	SetBounds (see page 12522)	Sets the windowed control's boundary properties all at once.
	SetupInternalLabel (see page 12523)	Initializes the control that implements the EditLabel (see page 12524) property.

68.1.18.3.1 TCustomLabeledEdit.CMBidimodechanged Method

Pascal

```
procedure CMBidimodechanged(var Message: TMessage); message CM_BIDIMODECHANGED;
```

C++

```
__fastcall CMBidimodechanged(TMessage Message);
```

Description

This is CMBidimodechanged, a member of class TCustomLabeledEdit.

68.1.18.3.2 TCustomLabeledEdit.CMEnabledchanged Method

Responds when the Enabled property changes.

Pascal

```
procedure CMEnabledchanged(var Message: TMessage); message CM_ENABLEDCHANGED;
```

C++

```
__fastcall CMEnabledchanged(TMessage Message);
```

Description

CMEnabledchanged is called automatically when an application changes the Enabled property.

Message is a CM_ENABLEDCHANGED message that notifies the labeled edit of the change.

The labeled edit control responds by changing the Enabled property of the associated label to the value of its own Enabled property.

See Also

Enabled (see page 5821)

EditLabel (see page 12524)

68.1.18.3.3 TCustomLabeledEdit.CMVisiblechanged Method

Responds when the Visible property changes.

Pascal

```
procedure CMVisiblechanged(var Message: TMessage); message CM_VISIBLECHANGED;
```

C++

```
__fastcall CMVisiblechanged(TMessage Message);
```

Description

CMVisiblechanged is called automatically when an application changes the Visible property.

Message is a CM_VISIBLECHANGED message that notifies the labeled edit of the change.

The labeled edit control responds by changing the Visible property of the associated label to the value of its own Visible property.

See Also

Visible (see page 5847)

EditLabel (see page 12524)

68.1.18.3.4 TCustomLabeledEdit.Notification Method

Responds to notifications when objects are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

This protected method is called automatically immediately after objects are created and immediately before they are destroyed. TCustomLabeledEdit (see page 12508) overrides this method to update the EditLabel (see page 12524) property when it receives a notification that the label component is about to be destroyed.

AComponent is the component that was just created or that is about to be destroyed.

Operation indicates whether the notification indicates the creation or destruction of AComponent.

See Also

EditLabel (see page 12524)

68.1.18.3.5 TCustomLabeledEdit.SetBounds Method

Sets the windowed control's boundary properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to change all of the control's boundary properties at once. The same effect can be achieved by setting the Left, Top, Width, and Height properties separately. By setting all four properties at once, SetBounds ensures that the control will not repaint between changes

Specify the values for the Left, Top, Width, and Height properties as the value of the ALeft, ATop, AWidth, and AHeight parameters, respectively.

See Also

Height (see page 5825)

Left (see page 5830)

Top (see page 5844)

Width (see page 5849)

68.1.18.3.6 TCustomLabeledEdit.SetName Method

Sets the name of the control.

Pascal

```
procedure SetName(const Value: TComponentName); override;
```

C++

```
virtual __fastcall SetName(const TComponentName Value);
```

Description

SetName is the protected write implementation of the Name property. TControl updates the Text or Caption property, if appropriate, after changing the value of Name.

Override SetName to execute additional code when the value of the Name property changes.

See Also

Name

Caption (see page 5806)

Text (see page 5842)

68.1.18.3.7 TCustomLabeledEdit.SetParent Method

Sets the value of the Parent property.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected property setter for the Parent property. TCustomLabeledEdit (see page 12508) overrides SetParent to ensure that the label has the same parent as the edit control.

See Also

Parent (see page 5832)

EditLabel (see page 12524)

68.1.18.3.8 TCustomLabeledEdit.SetupInternalLabel Method

Initializes the control that implements the EditLabel (see page 12524) property.

Pascal

```
procedure SetupInternalLabel;
```

C++

```
__fastcall SetupInternalLabel();
```

Description

Applications do not need to call SetupInternalLabel because this method is called when you set the EditLabel (see page 12524) property.

SetupInternalLabel ensures that there is a control to implement the EditLabel (see page 12524) property, adds the labeled edit control to the list of components that should be notified when the label is freed, and sets the label's FocusControl property to the labeled edit control.

See Also

EditLabel (see page 12524)

FocusControl (see page 21306)

68.1.18.4 TCustomLabeledEdit Properties

The properties of the TCustomLabeledEdit class are listed here.

Public Properties

	Name	Description
R	EditLabel (see page 12524)	Specifies the control that implements the label.
	LabelPosition (see page 12524)	Specifies the position of the label relative to the edit control.
	LabelSpacing (see page 12525)	Specifies the distance, in pixels, between the label and the edit region.

68.1.18.4.1 TCustomLabeledEdit.EditLabel Property

Specifies the control that implements the label.

Pascal

```
property EditLabel: TBoundLabel;
```

C++

```
__property TBoundLabel EditLabel;
```

Description

Use EditLabel to work with the label that is associated with this labeled edit control. Use this label's properties to specify the caption that appears on the label, the position of the caption within the label, the font used to write that caption, and so on. At design time, expand the EditLabel property in the Object Inspector to set its properties.

See Also

LabelPosition (see page 12524)

LabelSpacing (see page 12525)

68.1.18.4.2 TCustomLabeledEdit.LabelPosition Property

Specifies the position of the label relative to the edit control.

Pascal

```
property LabelPosition: TLabelPosition;
```

C++

```
__property TLabelPosition LabelPosition;
```

Description

Set LabelPosition to indicate where the label sits relative to the edit control.

See Also

EditLabel (see page 12524)

LabelSpacing (see page 12525)

68.1.18.4.3 TCustomLabeledEdit.LabelSpacing Property

Specifies the distance, in pixels, between the label and the edit region.

Pascal

```
property LabelSpacing: Integer;
```

C++

```
__property int LabelSpacing;
```

Description

Set LabelSpacing to indicate the distance, in pixels, between the label and the edit control.

See Also

EditLabel (see page 12524)

LabelPosition (see page 12524)

68.1.19 TCustomPanel Class

TCustomPanel is the base class for all panel components.

Class Hierarchy

Pascal

```
TCustomPanel = class(TCustomControl);
```

C++

```
class TCustomPanel : public TCustomControl;
```

File

ExtCtrls

Description

Use TCustomPanel as a base class when defining objects that include other controls as constituent parts. Panel components are controls can contain other controls, grouping them together.

When a panel is aligned with the form using the Align property, it maintains the same relative position to the form even when the form is resized. For example, a panel can be aligned so that it always remains on the top of the form, even when the user changes the shape and size of the form. This makes panels useful for implementing components that act like tool bars or status bars. For example, construct a tool bar or tool palette by adding speed buttons that execute commands or set modes to a panel .

See Also

TForm (see page 13992)

TPanel (see page 13032)

TCustomFrame (see page 13978)

TGroupBox (see page 21474)

68.1.19.1 TCustomPanel Members

The following tables list the members exposed by TCustomPanel.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.

	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 12537)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 12537)	Adjusts the way the panel responds to resize attempts.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 12537)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.

	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 12538)	Renders the image of the panel.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.

	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 12539)	This is SetParentBackground, a member of class TCustomPanel.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	Alignment (see page 12539)	Determines how the caption is aligned within the panel.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 12540)	Determines the style of the inner bevel of a panel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 12541)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 12542)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BorderStyle (see page 12543)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 12543)	Specifies the distance, in pixels, between the outer and inner bevels.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 12544)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	FullRepaint (see page 12545)	Determines how the panel repaints itself when it is resized.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	Locked (see page 12545)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 12546)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	VerticalAlignment (see page 12547)	Sets the vertical position of the caption.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 12536)	Creates and initializes an instance of TCustomPanel (see page 12525).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.

	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
V	GetControlsAlignment (see page 12538)	Returns the alignment of text in the panel.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.

	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentBackground (see page 12546)	Determines whether control uses parent's theme background.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.

	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.19.2 TCustomPanel.Create Constructor

Creates and initializes an instance of TCustomPanel (see page 12525).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomPanel(TComponent * AOwner);
```

Description

Call Create to instantiate a panel at runtime. Panels placed in forms at design time do not need to be explicitly created because they are created automatically.

AOwner is the component, typically a form, that is responsible for freeing the panel.

See Also

Owner (see page 2807)

68.1.19.3 TCustomPanel Methods

The methods of the TCustomPanel class are listed here.

Protected Methods

	Name	Description
	AdjustClientRect (see page 12537)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	CanAutoSize (see page 12537)	Adjusts the way the panel responds to resize attempts.
	CreateParams (see page 12537)	Initializes a window-creation parameter record passed in the Params parameter.
	Paint (see page 12538)	Renders the image of the panel.
	SetParentBackground (see page 12539)	This is SetParentBackground, a member of class TCustomPanel.

Public Methods

	Name	Description
	GetControlsAlignment (see page 12538)	Returns the alignment of text in the panel.

68.1.19.3.1 TCustomPanel.AdjustClientRect Method

Adjusts the ClientRect property for the idiosyncrasies of the window.

Pascal

```
procedure AdjustClientRect(var Rect: TRect); override;
```

C++

```
virtual __fastcall AdjustClientRect(TRect Rect);
```

Description

AdjustClientRect is called internally when the panel needs accurate information on where to place child controls within the client area. AdjustClientRect reduces ClientRect by the width of the inner and outer bevels so that child objects are not placed on the beveled region.

See Also

ClientRect

Controls (see page 6197)

68.1.19.3.2 TCustomPanel.CanAutoSize Method

Adjusts the way the panel responds to resize attempts.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize property is true and an attempt is made to resize the panel. It allows the panel to implement the AutoSize property before the resize sequence that begins with the OnCanResize and OnConstrainedResize events and ends with the OnResize event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the panel. CanAutoSize adjusts these values so that the panel's size adjusts to its contents. If the adjustment results in values of NewWidth and NewHeight that differ from the panel's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width and Height, CanAutoSize returns false, and the resize sequence is aborted.

See Also

AutoSize (see page 5803)

AdjustSize (see page 6129)

ConstrainedResize (see page 6133)

OnCanResize (see page 5851)

68.1.19.3.3 TCustomPanel.CreateParams Method

Initializes a window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

CreateParams is called automatically to initialize the window-creation parameters whenever the window for the panel needs to be created. CreateParams overrides the inherited method to implement the BorderStyle (see page 12543) property.

See Also

BorderStyle (see page 12543)

CreateParams (see page 6135)

68.1.19.3.4 TCustomPanel.GetControlsAlignment Method

Returns the alignment of text in the panel.

Pascal

```
function GetControlsAlignment: TAlignment; override;
```

C++

```
virtual __fastcall TAlignment GetControlsAlignment();
```

Description

GetControlsAlignment returns the value of the Alignment (see page 12539) property.

See Also

Alignment (see page 12539)

UseRightToLeftAlignment (see page 5790)

68.1.19.3.5 TCustomPanel.Paint Method

Renders the image of the panel.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Override the Paint method to change the way the panel is drawn. The Paint method for TCustomPanel (see page 12525) draws the bevels of the panel, sets the brush and font for the panel, and fills in the interior of the panel. Finally, Paint writes the value of Caption on the panel.

See Also

BevelInner (see page 12540)

BevelOuter (see page 12541)

Brush (see page 6195)

Caption (see page 5806)

Canvas (see page 5921)

68.1.19.3.6 TCustomPanel.SetParentBackground Method

Pascal

```
procedure SetParentBackground(Value: Boolean); override;
```

C++

```
virtual __fastcall SetParentBackground(Boolean Value);
```

Description

This is SetParentBackground, a member of class TCustomPanel.

68.1.19.4 TCustomPanel Properties

The properties of the TCustomPanel class are listed here.

Protected Properties

	Name	Description
	Alignment (see page 12539)	Determines how the caption is aligned within the panel.
	BevelInner (see page 12540)	Determines the style of the inner bevel of a panel.
	BevelOuter (see page 12541)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 12542)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BorderStyle (see page 12543)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 12543)	Specifies the distance, in pixels, between the outer and inner bevels.
	Color (see page 12544)	Specifies the background color of the control.
	FullRepaint (see page 12545)	Determines how the panel repaints itself when it is resized.
	Locked (see page 12545)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	ParentColor (see page 12546)	Determines where a control looks for its color information.
	VerticalAlignment (see page 12547)	Sets the vertical position of the caption.

Public Properties

	Name	Description
	ParentBackground (see page 12546)	Determines whether control uses parent's theme background.

68.1.19.4.1 TCustomPanel.Alignment Property

Determines how the caption is aligned within the panel.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use the Alignment property to specify if the caption of a panel appears on the left, right, or center of the panel. For example, to use a panel as a status bar that displays the TApplication. ::Hint property as the panel's Caption property, set Alignment to taLeftJustify to place the hint text on the left side of the panel.

Note: Use the VerticalAlignment (see page 12547) property to control the vertical position of the caption.

See Also

Hint (see page 13712)

Caption (see page 5806)

68.1.19.4.2 TCustomPanel.BevelInner Property

Determines the style of the inner bevel of a panel.

Pascal

```
property BevelInner: TPanelBevel;
```

C++

```
__property TPanelBevel BevelInner;
```

Description

Use BevelInner to provide the panel with a three-dimensional beveled look. A panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel, separated by the number of pixels specified in the BorderWidth (see page 12543) property. The width of the bevels, if they are drawn, is the BevelWidth (see page 12542) property.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelOuter (see page 12541)

BevelWidth (see page 12542)

BorderWidth (see page 12543)

68.1.19.4.3 TCustomPanel.BevelOuter Property

Determines the style of the outer bevel of a panel.

Pascal

```
property BevelOuter: TPanelBevel;
```

C++

```
__property TPanelBevel BevelOuter;
```

Description

Use BevelOuter to provide the panel with a three-dimensional beveled look. A panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel, separated by the number of pixels specified in the BorderWidth (see page 12543) property. The width of the outer bevel is specified in the BevelWidth (see page 12542) property in pixels.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
```

```
end;
```

See Also

BevelInner (see page 12540)

BevelWidth (see page 12542)

BorderWidth (see page 12543)

68.1.19.4.4 TCustomPanel.BevelWidth Property

Determines the width, in pixels, of both the inner and outer bevels of a panel.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify how wide the inner or outer bevel should be. Do not confuse BevelWidth, which is the width of the bevels, with BorderWidth (see page 12543), which is the space between the bevels.

If both the BevelInner (see page 12540) and BevelOuter (see page 12541) properties are bvNone, BevelWidth has no effect. To remove both bevels, set the BevelInner (see page 12540) and BevelOuter (see page 12541) properties to bvNone, rather than setting the BevelWidth to 0, as this involves less overhead when painting.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
```

```
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelInner (see page 12540)

BevelOuter (see page 12541)

BorderWidth (see page 12543)

68.1.19.4.5 TCustomPanel.BorderStyle Property

Determines the style of the line drawn around the perimeter of the panel control.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to specify whether the panel has a single line drawn around it. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

Do not confuse the line drawn around the panel with the BorderWidth (see page 12543) of the panel. The BorderWidth (see page 12543) of the panel is the distance between the outer and inner bevels.

See Also

BorderWidth (see page 12543)

68.1.19.4.6 TCustomPanel.BorderWidth Property

Specifies the distance, in pixels, between the outer and inner bevels.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to specify how wide the border around the panel should be. A value of 0 (zero) means no border should appear.

The border of a panel is the area between the outer and inner bevels. It is visible only if the inner bevel is raised or lowered, but affects the inset of the caption within the panel even if BevelInner (see page 12540) is bvNone. If the Alignment (see page 12539) property is not taCenter, the Caption will be aligned to the inner edge of the border. This edge is BorderWidth pixels in from the outer bevel if BevelInner (see page 12540) is bvNone. It is the inner edge of the inner bevel otherwise.

Do not confuse the border of the panel with line drawn around the panel itself. The line around the panel is specified by the BorderStyle (see page 12543) property.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

Alignment (see page 12539)

BevelInner (see page 12540)

BevelOuter (see page 12541)

BorderStyle (see page 12543)

Caption (see page 5806)

68.1.19.4.7 TCustomPanel.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

68.1.19.4.8 TCustomPanel.FullRepaint Property

Determines how the panel repaints itself when it is resized.

Pascal

```
property FullRepaint: Boolean;
```

C++

```
__property Boolean FullRepaint;
```

Description

FullRepaint controls how the panel responds when it is resized. When FullRepaint is true, the entire panel, including the beveled border repaints when the size changes. When FullRepaint is false, only the area inside the beveled border repaints.

See Also

Invalidate (see page 6161)

68.1.19.4.9 TCustomPanel.Locked Property

Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.

Pascal

```
property Locked: Boolean;
```

C++

```
__property Boolean Locked;
```

Description

Set the Locked property to false when the panel is used as a toolbar by a TOleContainer component. Any panels aligned to a TOleContainer are assumed to be replaceable toolbars unless the Locked property is true. Prevent the panel from being replaced by an OLE server's toolbar by setting the panel's Locked property to true.

Note: An OLE application's toolbars must be in panels aligned to a side of the form (that is, their Align properties must be set to alTop, alBottom, alLeft, or alRight).

See Also

Align (see page 5798)

TOleContainer (see page 17921)

68.1.19.4.10 TCustomPanel.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground: Boolean;
```

C++

```
__property Boolean ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

68.1.19.4.11 TCustomPanel.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

68.1.19.4.12 TCustomPanel.VerticalAlignment Property

Sets the vertical position of the caption.

Pascal

```
property VerticalAlignment: TVerticalAlignment;
```

C++

```
__property TVerticalAlignment VerticalAlignment;
```

Description

Use the VerticalAlignment property to control the position of the caption within the panel. Choose one of the following values.

Value	Description
taAlignBottom	The caption displays at the bottom of the panel.
taAlignTop	The caption displays at the top of the panel.
taVerticalCenter	The caption displays at the center of the panel.

Note: Use the Alignment (see page 12539) property to control the horizontal position of the caption.

See Also

Caption

68.1.20 TCustomRadioGroup Class

TCustomRadioGroup is the base class for radio-group components.

Class Hierarchy

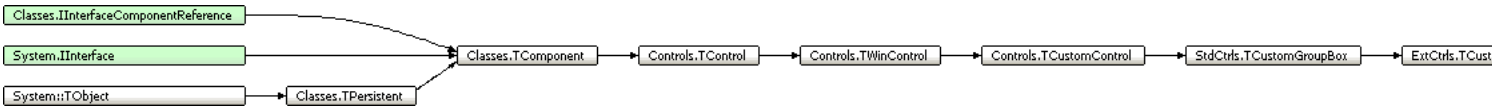

Pascal

```
TCustomRadioGroup = class(TCustomGroupBox);
```

C++

```
class TCustomRadioGroup : public TCustomGroupBox;
```

File

ExtCtrls

Description

Radio groups are collections of radio buttons in a single group box. TCustomRadioGroup is the class from which radio-group components—including TRadioGroup (see page 13115) and TDBRadioGroup—descend.

Radio buttons that are contained directly in the same control component are said to be "grouped." When the user checks a radio button, all other radio buttons in its group become unchecked. Hence, two radio buttons on a form can be checked at the same time only if they are placed in separate containers, such as group boxes.

See Also

TRadioGroup (see page 13115)

TDBRadioGroup (see page 9283)

TRadioButton (see page 21753)

68.1.20.1 TCustomRadioGroup Members

The following tables list the members exposed by TCustomRadioGroup.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.

	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 21287)	Adjusts the ClientRect property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanModify (see page 12559)	Determines whether the user can select a different radio button in the group.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.

	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 21287)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.

	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 12560)	Prevents the streaming system from loading and saving child controls in the usual fashion.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 12560)	Initializes the radio group after it is loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 5920)	Renders the image of a custom control.
	Paint (see page 21287)	Renders the onscreen image of the group box.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.

	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 12560)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.

	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Columns (see page 12562)	Specifies number of columns in the radio group.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	ItemIndex (see page 12562)	Indicates which radio button in the group is currently selected.
	Items (see page 12563)	Lists the radio buttons in the radio group.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.

	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 12558)	Creates and initializes a TCustomRadioGroup (see page 12547) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 12558)	Destroys the TCustomRadioGroup (see page 12547) instance and the radio buttons it contains.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.

	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
V	FlipChildren (see page 12559)	Prevents the default behavior of flipping the child radio buttons.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Buttons (see page 12561)	Access individual TRadioButton objects.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.

R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.20.2 TCustomRadioGroup.Create Constructor

Creates and initializes a TCustomRadioGroup (see page 12547) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomRadioGroup(TComponent * AOwner);
```

Description

Call Create to instantiate a radio group at runtime. Radio group components added to forms at design time are created automatically.

AOwner is the component that is responsible for freeing the radio group instance. It becomes the value of the Owner property.

See Also

Owner (see page 2807)

68.1.20.3 TCustomRadioGroup.Destroy Destructor

Destroys the TCustomRadioGroup (see page 12547) instance and the radio buttons it contains.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomRadioGroup();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the radio group is not nil, and only then calls Destroy.

Destroy frees the memory for each item in the group, then calls the inherited destructor.

See Also

Free (see page 22300)

68.1.20.4 TCustomRadioGroup Methods

The methods of the TCustomRadioGroup class are listed here.

Protected Methods

	Name	Description
	CanModify (see page 12559)	Determines whether the user can select a different radio button in the group.
	GetChildren (see page 12560)	Prevents the streaming system from loading and saving child controls in the usual fashion.
	Loaded (see page 12560)	Initializes the radio group after it is loaded into memory.
	ReadState (see page 12560)	Prepares the control for having its properties assigned values from a stream.

Public Methods

	Name	Description
	FlipChildren (see page 12559)	Prevents the default behavior of flipping the child radio buttons.

68.1.20.4.1 TCustomRadioGroup.CanModify Method

Determines whether the user can select a different radio button in the group.

Pascal

```
function CanModify: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanModify();
```

Description

Event handlers call CanModify to verify that the user is allowed to select a different radio button in the group. As defined in TCustomRadioGroup (see page 12547), this method always returns true; override it to return false in situations where the user should not be able to make changes.

See Also

CanModify

68.1.20.4.2 TCustomRadioGroup.FlipChildren Method

Prevents the default behavior of flipping the child radio buttons.

Pascal

```
procedure FlipChildren(AllLevels: Boolean); override;
```

C++

```
virtual __fastcall FlipChildren(Boolean AllLevels);
```

Description

FlipChildren overrides the inherited FlipChildren method to disable flipping the radio buttons.

To flip the children of a radio group, use the BiDiMode property.

See Also

BiDiMode (see page 5803)

68.1.20.4.3 TCustomRadioGroup.GetChildren Method

Prevents the streaming system from loading and saving child controls in the usual fashion.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called in the streaming sequence that loads and saves VCL components. Because radio groups load and save their buttons using the ReadState (see page 12560) and WriteState methods, TCustomRadioGroup (see page 12547) overrides TWinControl to disable the GetChildren method.

See Also

GetChildren (see page 6153)

GetChildren

GetParentComponent

readstate (see page 12560)

WriteState

68.1.20.4.4 TCustomRadioGroup.Loaded Method

Initializes the radio group after it is loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call Loaded. The component streaming system calls Loaded automatically after the radio group's form is loaded into memory so that it can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method to ensure that the buttons in the group are properly arranged.

See Also

Items (see page 12563)

68.1.20.4.5 TCustomRadioGroup.ReadState Method

Prepares the control for having its properties assigned values from a stream.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

Override ReadState to change the preparations the control makes as it readies itself to have its property values assigned from a stream. For example, a control might destroy temporary internal data structures or objects before new instances are loaded from the stream. Be sure to include a call to the inherited method when overriding ReadState.

As implemented in TWinControl, ReadState disables control alignment until all the child controls have been read from the stream, calls the inherited ReadState method, and re-enables the alignment of all the controls. The method then builds the tab order list of controls and synchronizes its visibility with that of its parent control.

See Also

Parent (see page 5832)

ReadState (see page 5778)

68.1.20.5 TCustomRadioGroup Properties

The properties of the TCustomRadioGroup class are listed here.

Protected Properties

	Name	Description
	Columns (see page 12562)	Specifies number of columns in the radio group.
	ItemIndex (see page 12562)	Indicates which radio button in the group is currently selected.
	Items (see page 12563)	Lists the radio buttons in the radio group.

Public Properties

	Name	Description
R	Buttons (see page 12561)	Access individual TRadioButton objects.

68.1.20.5.1 TCustomRadioGroup.Buttons Property

Access individual TRadioButton objects.

Pascal

```
property Buttons [Index: Integer]: TRadioButton;
```

C++

```
__property TRadioButton Buttons[int Index];
```

Description

Buttons provides direct access to the individual TRadioButton objects created by the control. Buttons is a read-only property. This means you can access individual TRadioButton objects, but you cannot add or delete TRadioButton objects. To modify the button list, change the Items (see page 12563) property.

Index is the zero-origin button index. If Index is equal to or greater than the number of buttons, an error is raised.

See Also

Items (see page 12563)

68.1.20.5.2 TCustomRadioGroup.Columns Property

Specifies number of columns in the radio group.

Pascal

```
property Columns: Integer;
```

C++

```
__property int Columns;
```

Description

Columns determines the number of columns in the radio group if Orientation is orVertical (the default), and the number of rows if Orientation is orHorizontal. Its value can range from 1 to 16. The default value is 1, which means that the radio buttons are arranged in a single line.

68.1.20.5.3 TCustomRadioGroup.ItemIndex Property

Indicates which radio button in the group is currently selected.

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

Description

ItemIndex holds the index of the selected radio button in the Items (see page 12563) list. (The first button is 0.) The value of ItemIndex changes at runtime as the user selects radio buttons. If you want one of the buttons to appear selected when the application starts, assign that button to ItemIndex at design time; otherwise, leave ItemIndex set to the default value of -1, which means that no button is selected.

C++ Examples:

```
/*
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Items->Add("Vertical");
  RadioGroup1->Items->Add("Horizontal");
  RadioGroup1->ItemIndex = 2;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Vertical")
    ScrollBar1->Kind = sbVertical;
  else if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Horizontal")
    ScrollBar1->Kind = sbHorizontal;
}
```

Delphi Examples:

```
{
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  RadioGroup1.Items.Add('Vertical');
  RadioGroup1.Items.Add('Horizontal');
  RadioGroup1.ItemIndex := 2;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Vertical' then
    ScrollBar1.Kind := sbVertical;
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Horizontal' then
    ScrollBar1.Kind := sbHorizontal;
end;
```

See Also

Items (see page 12563)

68.1.20.5.4 TCustomRadioGroup.Items Property

Lists the radio buttons in the radio group.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Items holds a TStrings object that lists the captions of the radio buttons in the group. In TCustomRadioGroup (see page 12547) descendants, where this property is published, radio buttons can be added or removed by editing the Items list from the Object Inspector.

Note: A TRadioButton object is created automatically for each string in Items. You cannot directly add or remove TRadioButton objects, but you can access individual TButton objects through the Buttons (see page 12561) property.

C++ Examples:

```
/*
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Items->Add("Vertical");
  RadioGroup1->Items->Add("Horizontal");
  RadioGroup1->ItemIndex = 2;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Vertical")
    ScrollBar1->Kind = sbVertical;
  else if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Horizontal")
    ScrollBar1->Kind = sbHorizontal;
}
```

Delphi Examples:

```
{
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  RadioGroup1.Items.Add('Vertical');
  RadioGroup1.Items.Add('Horizontal');
  RadioGroup1.ItemIndex := 2;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Vertical' then
    ScrollBar1.Kind := sbVertical;
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Horizontal' then
    ScrollBar1.Kind := sbHorizontal;
end;
```

See Also

ItemIndex (see page 12562)

TRadioButton (see page 21753)

Buttons (see page 12561)

68.1.21 TCustomTrayIcon Class

The base class for TTrayIcon (see page 13252).

Class Hierarchy

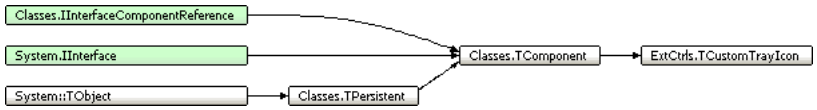

Pascal

```
TCustomTrayIcon = class(TComponent);
```

C++

```
class TCustomTrayIcon : public TComponent;
```

File

ExtCtrls

Description

TCustomTrayIcon is the base class for TTrayIcon (see page 13252), which places an icon in the system tray next to the clock.

68.1.21.1 TCustomTrayIcon Members

The following tables list the members exposed by TCustomTrayIcon.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoOnAnimate (see page 12569)	This is DoOnAnimate, a member of class TCustomTrayIcon.
	GetAnimateInterval (see page 12569)	This is GetAnimateInterval, a member of class TCustomTrayIcon.
	GetBalloonTimeout (see page 12569)	This is GetBalloonTimeout, a member of class TCustomTrayIcon.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAnimate (see page 12570)	This is SetAnimate, a member of class TCustomTrayIcon.
	SetAnimateInterval (see page 12570)	This is SetAnimateInterval, a member of class TCustomTrayIcon.
	SetBalloonHint (see page 12571)	This is SetBalloonHint, a member of class TCustomTrayIcon.
	SetBalloonTimeout (see page 12571)	This is SetBalloonTimeout, a member of class TCustomTrayIcon.
	SetBalloonTitle (see page 12571)	This is SetBalloonTitle, a member of class TCustomTrayIcon.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHint (see page 12571)	This is SetHint, a member of class TCustomTrayIcon.
	SetIcon (see page 12572)	This is SetIcon, a member of class TCustomTrayIcon.
V	SetIconIndex (see page 12572)	This is SetIconIndex, a member of class TCustomTrayIcon.
	SetIconList (see page 12572)	This is SetIconList, a member of class TCustomTrayIcon.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetVisible (see page 12572)	This is SetVisible, a member of class TCustomTrayIcon.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WindowProc (see page 12573)	This is WindowProc, a member of class TCustomTrayIcon.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Data (see page 12575)	This is Data, a member of class TCustomTrayIcon.

Public Events

	Name	Description
	OnAnimate (see page 12577)	Occurs each time the icon changes when the tray icon is animating.
	OnClick (see page 12577)	Occurs when the user clicks the tray icon.
	OnDblClick (see page 12578)	Occurs when the user double clicks the tray icon.
	OnMouseDown (see page 12578)	Occurs when the user presses a mouse button while the mouse pointer is over a control.
	OnMouseMove (see page 12578)	Occurs when the user moves the mouse pointer while the mouse pointer is over the tray icon.
	OnMouseUp (see page 12578)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 12568)	Creates and initializes a TCustomTrayIcon (see page 12564) instance.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 12568)	Frees the memory associated with the TCustomTrayIcon (see page 12564) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 12570)	This is the overview for the Refresh method overload.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetDefaultIcon (see page 12571)	Sets the tray icon to the default application icon.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	ShowBalloonHint (see page 12573)	Displays the balloon hint.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Animate (see page 12573)	Specifies whether the icon in the system tray is static or animated.
	AnimateInterval (see page 12574)	Controls (see page 5711) the length of time between image changes for the tray icon image.
	BalloonFlags (see page 12574)	Specifies an icon for the balloon hint.
	BalloonHint (see page 12574)	Specifies the text within a balloon hint.
	BalloonTimeout (see page 12575)	Specifies the length of time that the balloon displays.
	BalloonTitle (see page 12575)	Specifies the text that appears as the balloon title.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.

	Hint (see page 12575)	Specifies the text that appears when the user moves the mouse over the control.
	Icon (see page 12576)	Specifies the default icon shown in the system tray.
	IconIndex (see page 12576)	This is an index into the Icons (see page 12576) image list which is the icon currently being displayed in the system tray.
	Icons (see page 12576)	Specifies the list of images to be displayed for the icon in the system tray.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	PopupMenu (see page 12576)	Identifies the pop-up menu associated with the tray icon.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 12577)	Determines whether the icon appears in the system tray.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

68.1.21.2 TCustomTrayIcon.Create Constructor

Creates and initializes a TCustomTrayIcon (see page 12564) instance.

Pascal

```
constructor Create(Owner: TComponent); override;
```

C++

```
virtual __fastcall TCustomTrayIcon(TComponent * Owner);
```

Description

Creates and initializes a TCustomTrayIcon (see page 12564) instance.

68.1.21.3 TCustomTrayIcon.Destroy Destructor

Frees the memory associated with the TCustomTrayIcon (see page 12564) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomTrayIcon();
```

Description

Frees the memory associated with the TCustomTrayIcon (see page 12564) object.

68.1.21.4 TCustomTrayIcon Methods

The methods of the TCustomTrayIcon class are listed here.

Protected Methods

	Name	Description
V	DoOnAnimate (see page 12569)	This is DoOnAnimate, a member of class TCustomTrayIcon.
	GetAnimateInterval (see page 12569)	This is GetAnimateInterval, a member of class TCustomTrayIcon.
	GetBalloonTimeout (see page 12569)	This is GetBalloonTimeout, a member of class TCustomTrayIcon.
	SetAnimate (see page 12570)	This is SetAnimate, a member of class TCustomTrayIcon.

	SetAnimateInterval (see page 12570)	This is SetAnimateInterval, a member of class TCustomTrayIcon.
	SetBalloonHint (see page 12571)	This is SetBalloonHint, a member of class TCustomTrayIcon.
	SetBalloonTimeout (see page 12571)	This is SetBalloonTimeout, a member of class TCustomTrayIcon.
	SetBalloonTitle (see page 12571)	This is SetBalloonTitle, a member of class TCustomTrayIcon.
	SetHint (see page 12571)	This is SetHint, a member of class TCustomTrayIcon.
	SetIcon (see page 12572)	This is SetIcon, a member of class TCustomTrayIcon.
V	SetIconIndex (see page 12572)	This is SetIconIndex, a member of class TCustomTrayIcon.
	SetIconList (see page 12572)	This is SetIconList, a member of class TCustomTrayIcon.
V	SetVisible (see page 12572)	This is SetVisible, a member of class TCustomTrayIcon.
V	WindowProc (see page 12573)	This is WindowProc, a member of class TCustomTrayIcon.

Public Methods

	Name	Description
	Refresh (see page 12570)	This is the overview for the Refresh method overload.
	SetDefaultIcon (see page 12571)	Sets the tray icon to the default application icon.
V	ShowBalloonHint (see page 12573)	Displays the balloon hint.

68.1.21.4.1 TCustomTrayIcon.DoOnAnimate Method

Pascal

```
procedure DoOnAnimate(Sender: TObject); virtual;
```

C++

```
virtual __fastcall DoOnAnimate(TObject * Sender);
```

Description

This is DoOnAnimate, a member of class TCustomTrayIcon.

68.1.21.4.2 TCustomTrayIcon.GetAnimateInterval Method

Pascal

```
function GetAnimateInterval: Cardinal;
```

C++

```
__fastcall unsigned GetAnimateInterval();
```

Description

This is GetAnimateInterval, a member of class TCustomTrayIcon.

68.1.21.4.3 TCustomTrayIcon.GetBalloonTimeout Method

Pascal

```
function GetBalloonTimeout: Integer;
```

C++

```
__fastcall int GetBalloonTimeout();
```

Description

This is GetBalloonTimeout, a member of class TCustomTrayIcon.

68.1.21.4.4 Refresh Method

This is the overview for the Refresh method overload.

Overload List

	Name	Description
	TCustomTrayIcon.Refresh () (see page 12570)	Repaints the tray icon on the screen.
	TCustomTrayIcon.Refresh (Integer) (see page 12570)	Repaints the tray icon on the screen.

68.1.21.4.4.1 TCustomTrayIcon.Refresh Method ()

Repaints the tray icon on the screen.

Pascal

```
function Refresh(Message: Integer): Boolean; overload;
procedure Refresh; overload;
```

C++

```
__fastcall Boolean Refresh(int Message);
__fastcall Refresh();
```

Description

Repaints the tray icon on the screen.

68.1.21.4.5 TCustomTrayIcon.SetAnimate Method

Pascal

```
procedure SetAnimate(Value: Boolean);
```

C++

```
__fastcall SetAnimate(Boolean Value);
```

Description

This is SetAnimate, a member of class TCustomTrayIcon.

68.1.21.4.6 TCustomTrayIcon.SetAnimateInterval Method

Pascal

```
procedure SetAnimateInterval(Value: Cardinal);
```

C++

```
__fastcall SetAnimateInterval(unsigned Value);
```

Description

This is SetAnimateInterval, a member of class TCustomTrayIcon.

68.1.21.4.7 TCustomTrayIcon.SetBalloonHint Method

Pascal

```
procedure SetBalloonHint(const Value: string);
```

C++

```
__fastcall SetBalloonHint(const AnsiString Value);
```

Description

This is SetBalloonHint, a member of class TCustomTrayIcon.

68.1.21.4.8 TCustomTrayIcon.SetBalloonTimeout Method

Pascal

```
procedure SetBalloonTimeout(Value: Integer);
```

C++

```
__fastcall SetBalloonTimeout(int Value);
```

Description

This is SetBalloonTimeout, a member of class TCustomTrayIcon.

68.1.21.4.9 TCustomTrayIcon.SetBalloonTitle Method

Pascal

```
procedure SetBalloonTitle(const Value: string);
```

C++

```
__fastcall SetBalloonTitle(const AnsiString Value);
```

Description

This is SetBalloonTitle, a member of class TCustomTrayIcon.

68.1.21.4.10 TCustomTrayIcon.SetDefaultIcon Method

Sets the tray icon to the default application icon.

Pascal

```
procedure SetDefaultIcon;
```

C++

```
__fastcall SetDefaultIcon();
```

Description

This method sets the tray icon to the default application icon if the Icon (see page 12576) property is unassigned. This method is generally not used.

68.1.21.4.11 TCustomTrayIcon.SetHint Method

Pascal

```
procedure SetHint(const Value: string);
```

C++

```
__fastcall SetHint(const AnsiString Value);
```

Description

This is SetHint, a member of class TCustomTrayIcon.

68.1.21.4.12 TCustomTrayIcon.SetIcon Method

Pascal

```
procedure SetIcon(Value: TIcon);
```

C++

```
__fastcall SetIcon(TIcon Value);
```

Description

This is SetIcon, a member of class TCustomTrayIcon.

68.1.21.4.13 TCustomTrayIcon.SetIconIndex Method

Pascal

```
procedure SetIconIndex(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetIconIndex(int Value);
```

Description

This is SetIconIndex, a member of class TCustomTrayIcon.

68.1.21.4.14 TCustomTrayIcon.SetIconList Method

Pascal

```
procedure SetIconList(Value: TImageList);
```

C++

```
__fastcall SetIconList(TImageList Value);
```

Description

This is SetIconList, a member of class TCustomTrayIcon.

68.1.21.4.15 TCustomTrayIcon.SetVisible Method

Pascal

```
procedure SetVisible(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetVisible(Boolean Value);
```

Description

This is SetVisible, a member of class TCustomTrayIcon.

68.1.21.4.16 TCustomTrayIcon.ShowBalloonHint Method

Displays the balloon hint.

Pascal

```
procedure ShowBalloonHint; virtual;
```

C++

```
virtual __fastcall ShowBalloonHint();
```

Description

Displays the balloon hint.

68.1.21.4.17 TCustomTrayIcon.WindowProc Method

Pascal

```
procedure WindowProc(var Message: TMessage); virtual;
```

C++

```
virtual __fastcall WindowProc(TMessage Message);
```

Description

This is WindowProc, a member of class TCustomTrayIcon.

68.1.21.5 TCustomTrayIcon Properties

The properties of the TCustomTrayIcon class are listed here.

Protected Properties

	Name	Description
	Data (see page 12575)	This is Data, a member of class TCustomTrayIcon.

Public Properties

	Name	Description
	Animate (see page 12573)	Specifies whether the icon in the system tray is static or animated.
	AnimateInterval (see page 12574)	Controls (see page 5711) the length of time between image changes for the tray icon image.
	BalloonFlags (see page 12574)	Specifies an icon for the balloon hint.
	BalloonHint (see page 12574)	Specifies the text within a balloon hint.
	BalloonTimeout (see page 12575)	Specifies the length of time that the balloon displays.
	BalloonTitle (see page 12575)	Specifies the text that appears as the balloon title.
	Hint (see page 12575)	Specifies the text that appears when the user moves the mouse over the control.
	Icon (see page 12576)	Specifies the default icon shown in the system tray.
	IconIndex (see page 12576)	This is an index into the Icons (see page 12576) image list which is the icon currently being displayed in the system tray.
	Icons (see page 12576)	Specifies the list of images to be displayed for the icon in the system tray.
	PopupMenu (see page 12576)	Identifies the pop-up menu associated with the tray icon.
	Visible (see page 12577)	Determines whether the icon appears in the system tray.

68.1.21.5.1 TCustomTrayIcon.Animate Property

Specifies whether the icon in the system tray is static or animated.

Pascal

```
property Animate: Boolean;
```

C++

```
__property Boolean Animate;
```

Description

When set to True, the icon shown in the system tray displays, in sequence, the images in the image list specified by the Icons (see page 12576) property. The images change only if the specified image list contains images. The images change on a rotating basis by the interval set by the AnimateInterval (see page 12574) property. When set to False the animation stops.

68.1.21.5.2 TCustomTrayIcon.AnimateInterval Property

Controls (see page 5711) the length of time between image changes for the tray icon image.

Pascal

```
property AnimateInterval: Cardinal;
```

C++

```
__property unsigned AnimateInterval;
```

Description

Use AnimateInterval to specify the length of time between image changes in the tray icon. The default interval is 1000 ms (1 second).

68.1.21.5.3 TCustomTrayIcon.BalloonFlags Property

Specifies an icon for the balloon hint.

Pascal

```
property BalloonFlags: TBalloonFlags;
```

C++

```
__property TBalloonFlags BalloonFlags;
```

Description

The following values specify an icon to be displayed in the balloon hint:

- bfNone
- bfInfo
- bfWarning
- bfError

68.1.21.5.4 TCustomTrayIcon.BalloonHint Property

Specifies the text within a balloon hint.

Pascal

```
property BalloonHint: string;
```

C++

```
__property AnsiString BalloonHint;
```

Description

Use BalloonHint to specify the text string that appears when a balloon hint is shown.

68.1.21.5.5 TCustomTrayIcon.BalloonTimeout Property

Specifies the length of time that the balloon displays.

Pascal

```
property BalloonTimeout: Integer;
```

C++

```
__property int BalloonTimeout;
```

Description

Use BalloonTimeout to set the length of time that a balloon displays. By default, this value is set to 3000 ms (3 seconds).

68.1.21.5.6 TCustomTrayIcon.BalloonTitle Property

Specifies the text that appears as the balloon title.

Pascal

```
property BalloonTitle: string;
```

C++

```
__property AnsiString BalloonTitle;
```

Description

Specifies the text that appears as the balloon title.

68.1.21.5.7 TCustomTrayIcon.Data Property

Pascal

```
property Data: TNotifyIconData;
```

C++

```
__property TNotifyIconData Data;
```

Description

This is Data, a member of class TCustomTrayIcon.

68.1.21.5.8 TCustomTrayIcon.Hint Property

Specifies the text that appears when the user moves the mouse over the control.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Specifies the text that appears when the user moves the mouse over the control.

68.1.21.5.9 TCustomTrayIcon.Icon Property

Specifies the default icon shown in the system tray.

Pascal

```
property Icon: TIcon;
```

C++

```
__property TIcon Icon;
```

Description

Use the Icon property to set the default icon to be shown in the system tray. If you do not specify a value, the application icon is used.

68.1.21.5.10 TCustomTrayIcon.IconIndex Property

This is an index into the Icons (see page 12576) image list which is the icon currently being displayed in the system tray.

Pascal

```
property IconIndex: Integer;
```

C++

```
__property int IconIndex;
```

Description

This is an index into the Icons (see page 12576) image list which is the icon currently being displayed in the system tray.

68.1.21.5.11 TCustomTrayIcon.Icons Property

Specifies the list of images to be displayed for the icon in the system tray.

Pascal

```
property Icons: TImageList;
```

C++

```
__property TImageList Icons;
```

Description

Use this property to specify the list of images used for the icon in the system tray. These images are displayed when the Animate (see page 12573) property is set to True.

68.1.21.5.12 TCustomTrayIcon.PopupMenu Property

Identifies the pop-up menu associated with the tray icon.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Use PopupMenuOn to specify when the menu is displayed.

68.1.21.5.13 TCustomTrayIcon.Visible Property

Determines whether the icon appears in the system tray.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Determines whether the icon appears in the system tray.

68.1.21.6 TCustomTrayIcon Events

The events of the TCustomTrayIcon class are listed here.

Public Events

	Name	Description
	OnAnimate (see page 12577)	Occurs each time the icon changes when the tray icon is animating.
	OnClick (see page 12577)	Occurs when the user clicks the tray icon.
	OnDblClick (see page 12578)	Occurs when the user double clicks the tray icon.
	OnMouseDown (see page 12578)	Occurs when the user presses a mouse button while the mouse pointer is over a control.
	OnMouseMove (see page 12578)	Occurs when the user moves the mouse pointer while the mouse pointer is over the tray icon.
	OnMouseUp (see page 12578)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

68.1.21.6.1 TCustomTrayIcon.OnAnimate Event

Occurs each time the icon changes when the tray icon is animating.

Pascal

```
property OnAnimate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAnimate;
```

Description

Occurs each time the icon changes when the tray icon is animating.

68.1.21.6.2 TCustomTrayIcon.OnClick Event

Occurs when the user clicks the tray icon.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Occurs when the user clicks the tray icon.

68.1.21.6.3 TCustomTrayIcon.OnDblClick Event

Occurs when the user double clicks the tray icon.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Occurs when the user double clicks the tray icon.

68.1.21.6.4 TCustomTrayIcon.OnMouseDown Event

Occurs when the user presses a mouse button while the mouse pointer is over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Occurs when the user presses a mouse button while the mouse pointer is over a control.

68.1.21.6.5 TCustomTrayIcon.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over the tray icon.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Occurs when the user moves the mouse pointer while the mouse pointer is over the tray icon.

68.1.21.6.6 TCustomTrayIcon.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

68.1.22 TFlowPanel Class

TFlowPanel implements a flow panel control in which components are placed in pre-defined locations.

Class Hierarchy

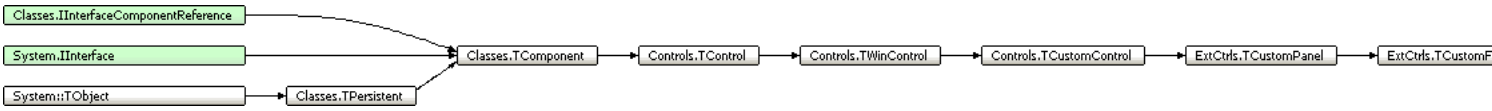

Pascal

```
TFlowPanel = class(TCustomFlowPanel);
```

C++

```
class TFlowPanel : public TCustomFlowPanel;
```

File

ExtCtrls

Description

Use TFlowPanel to put an empty flow panel on a form. The major difference between a traditional panel and a flow panel is the way in which controls are placed. With a traditional panel, you place a control (such as a button) in a specific location. You can freely move that control to any location within the panel using the mouse. In a flow panel, each control is placed in a specific location, regardless of where you place it with the mouse. The automatic location is controlled by the FlowStyle (see page 12610) property. For example, using the default FlowStyle (see page 12610) property of LeftRightTopBottom, the first control you add to the flow panel snaps to the top left corner. The second control that you add snaps next to the first control, and so on.

See Also

TPanel (see page 13032)

68.1.22.1 TFlowPanel Members

The following tables list the members exposed by TFlowPanel.

Protected Events

	Name	Description
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.

	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 12537)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 12487)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanAutoSize (see page 12537)	Adjusts the way the panel responds to resize attempts.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 12537)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.

	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.

	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 5920)	Renders the image of a custom control.
	Paint (see page 12538)	Renders the image of the panel.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.

	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBackground (see page 12539)	This is SetParentBackground, a member of class TCustomPanel.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.

	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 5918)	Creates an instance of TCustomControl (see page 5908).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 12486)	Creates and initializes an instance of TCustomPanel.
V	Create (see page 12536)	Creates and initializes an instance of TCustomPanel (see page 12525).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 12486)	Destroys an instance of TCustomControl.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetChildren (see page 12487)	Calls a specified method for each child of the control.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlIndex (see page 12488)	This is GetControlIndex, a member of class TCustomFlowPanel.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetControlsAlignment (see page 12538)	Returns the alignment of text in the panel.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.

V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetControlIndex (see page 12488)	This is SetControlIndex, a member of class TCustomFlowPanel.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.

	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 12626)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 12626)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 12627)	Occurs when an attempt is made to resize the control.
	OnClick (see page 12627)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 12629)	Adjust resize constraints.
	OnContextPopup (see page 12630)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 12631)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 12631)	Occurs when another control is docked to the control.
	OnDockOver (see page 12633)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 12635)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 12637)	Occurs when the user drags an object over a control.
	OnEndDock (see page 12640)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 12640)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 12641)	Occurs when a control receives the input focus.
	OnExit (see page 12643)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 12645)	Returns the control's docking information.
	OnMouseActivate (see page 12647)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 12648)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 12650)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 12651)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 12651)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

	OnMouseUp (see page 12653)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 12655)	Occurs immediately after the control is resized.
	OnStartDock (see page 12656)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 12657)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 12657)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 12591)	Determines how the control aligns within its container (parent control).
	Alignment (see page 12594)	Determines how the caption is aligned within the panel.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 12595)	Specifies how the control is anchored to its parent.
	AutoSize (see page 12595)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AutoWrap (see page 12596)	Specifies whether controls are automatically wrapped to fit on the flow panel.
	BevelEdges (see page 12596)	Specifies which edges of the control are beveled.
	BevelInner (see page 12596)	Determines the style of the inner bevel of a panel.
	BevelKind (see page 12597)	Specifies the control's bevel style.
	BevelOuter (see page 12598)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 12599)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BiDiMode (see page 12600)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 12600)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 12601)	Specifies the distance, in pixels, between the outer and inner bevels.
	Caption (see page 12602)	Specifies a text string that identifies the control to the user.
	Color (see page 12605)	Specifies the background color of the control.
	Constraints (see page 12605)	Specifies the size constraints for the control.
	Ctl3D (see page 12606)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DockSite (see page 12606)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 12607)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 12608)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 12608)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 12610)	Controls whether the control responds to mouse, keyboard, and timer events.
	FlowStyle (see page 12610)	Specifies the starting point and direction for controls added to a flow panel.
	Font (see page 12611)	Controls the attributes of text written on or in the control.
	FullRepaint (see page 12612)	Determines how the panel repaints itself when it is resized.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Locked (see page 12612)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Padding (see page 12613)	Specifies the padding of a control.
	ParentBackground (see page 12613)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 12613)	Specifies whether the control uses its parent's BiDiMode.

	ParentColor (see page 12614)	Determines where a control looks for its color information.
	ParentCtl3D (see page 12614)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 12614)	Determines where a control looks for its font information.
	ParentShowHint (see page 12615)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 12616)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 12620)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 12622)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 12622)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	UseDockManager (see page 12623)	Specifies whether the docking manager is used in drag-and-dock operations.
	VerticalAlignment (see page 12623)	Sets the vertical position of the caption.
	Visible (see page 12623)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.22.2 TFlowPanel Properties

The properties of the TFlowPanel class are listed here.

Published Properties

	Name	Description
	Align (see page 12591)	Determines how the control aligns within its container (parent control).
	Alignment (see page 12594)	Determines how the caption is aligned within the panel.
	Anchors (see page 12595)	Specifies how the control is anchored to its parent.
	AutoSize (see page 12595)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AutoWrap (see page 12596)	Specifies whether controls are automatically wrapped to fit on the flow panel.
	BevelEdges (see page 12596)	Specifies which edges of the control are beveled.
	BevelInner (see page 12596)	Determines the style of the inner bevel of a panel.
	BevelKind (see page 12597)	Specifies the control's bevel style.
	BevelOuter (see page 12598)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 12599)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BiDiMode (see page 12600)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 12600)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 12601)	Specifies the distance, in pixels, between the outer and inner bevels.
	Caption (see page 12602)	Specifies a text string that identifies the control to the user.
	Color (see page 12605)	Specifies the background color of the control.
	Constraints (see page 12605)	Specifies the size constraints for the control.
	Ctl3D (see page 12606)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 12606)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 12607)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 12608)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 12608)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 12610)	Controls whether the control responds to mouse, keyboard, and timer events.
	FlowStyle (see page 12610)	Specifies the starting point and direction for controls added to a flow panel.
	Font (see page 12611)	Controls the attributes of text written on or in the control.
	FullRepaint (see page 12612)	Determines how the panel repaints itself when it is resized.
	Locked (see page 12612)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	Padding (see page 12613)	Specifies the padding of a control.

	ParentBackground (see page 12613)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 12613)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 12614)	Determines where a control looks for its color information.
	ParentCtl3D (see page 12614)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 12614)	Determines where a control looks for its font information.
	ParentShowHint (see page 12615)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 12616)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 12620)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 12622)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 12622)	Determines if the user can tab to a control.
	UseDockManager (see page 12623)	Specifies whether the docking manager is used in drag-and-dock operations.
	VerticalAlignment (see page 12623)	Sets the vertical position of the caption.
	Visible (see page 12623)	Determines whether the component appears onscreen.

68.1.22.2.1 TFlowPanel.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
```

```
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
```

```
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
```

```
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

68.1.22.2.2 TFlowPanel.Alignment Property

Determines how the caption is aligned within the panel.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use the Alignment property to specify if the caption of a panel appears on the left, right, or center of the panel. For example, to use a panel as a status bar that displays the TApplication. ::Hint property as the panel's Caption property, set Alignment to taLeftJustify to place the hint text on the left side of the panel.

Note: Use the VerticalAlignment property to control the vertical position of the caption.

See Also

Hint (see page 13712)

Caption (see page 5806)

68.1.22.2.3 TFlowPanel.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

68.1.22.2.4 TFlowPanel.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints (see page 5812)

CanAutoSize (see page 5741)

68.1.22.2.5 TFlowPanel.AutoWrap Property

Specifies whether controls are automatically wrapped to fit on the flow panel.

Pascal

```
property AutoWrap: Boolean;
```

C++

```
__property Boolean AutoWrap;
```

Description

If set to True, a new control is automatically placed in a location on the flow panel in which it is visible. If set to False, a new control continues in the same direction as the previous control, even if it would no longer be visible on the panel. You can increase the size of the panel to view controls that are not automatically wrapped to the next line or column.

68.1.22.2.6 TFlowPanel.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

68.1.22.2.7 TFlowPanel.BevelInner Property

Determines the style of the inner bevel of a panel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TPanelBevel BevelInner;__property TBevelCut BevelInner;
```

Description

Use BevelInner to provide the panel with a three-dimensional beveled look. A panel component has two bevels, an outer bevel

drawn next to the border of the control, and an inner bevel drawn inside the outer bevel, separated by the number of pixels specified in the BorderWidth property. The width of the bevels, if they are drawn, is the BevelWidth property.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelOuter (see page 12541)

BevelWidth (see page 12542)

BorderWidth (see page 12543)

68.1.22.2.8 TFlowPanel.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

68.1.22.2.9 TFlowPanel.BevelOuter Property

Determines the style of the outer bevel of a panel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TPanelBevel BevelOuter;__property TBevelCut BevelOuter;
```

Description

Use BevelOuter to provide the panel with a three-dimensional beveled look. A panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel, separated by the number of pixels specified in the BorderWidth property. The width of the outer bevel is specified in the BevelWidth property in pixels.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
```

```
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelInner (see page 12540)

BevelWidth (see page 12542)

BorderWidth (see page 12543)

68.1.22.2.10 TFlowPanel.BevelWidth Property

Determines the width, in pixels, of both the inner and outer bevels of a panel.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify how wide the inner or outer bevel should be. Do not confuse BevelWidth, which is the width of the bevels, with BorderWidth, which is the space between the bevels.

If both the BevelInner and BevelOuter properties are bvNone, BevelWidth has no effect. To remove both bevels, set the BevelInner and BevelOuter properties to bvNone, rather than setting the BevelWidth to 0, as this involves less overhead when painting.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
```

```
      Align := alBottom;
      BevelInner := bvLowered;
      BevelOuter := bvRaised;
      BorderWidth := 1;
      BevelWidth := 1;
    end;
  end;
```

See Also

BevelInner (see page 12540)

BevelOuter (see page 12541)

BorderWidth (see page 12543)

68.1.22.2.11 TFlowPanel.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

68.1.22.2.12 TFlowPanel.BorderStyle Property

Determines the style of the line drawn around the perimeter of the panel control.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to specify whether the panel has a single line drawn around it. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

Do not confuse the line drawn around the panel with the BorderWidth of the panel. The BorderWidth of the panel is the distance between the outer and inner bevels.

See Also

BorderWidth (see page 12543)

68.1.22.2.13 TFlowPanel.BorderWidth Property

Specifies the distance, in pixels, between the outer and inner bevels.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to specify how wide the border around the panel should be. A value of 0 (zero) means no border should appear.

The border of a panel is the area between the outer and inner bevels. It is visible only if the inner bevel is raised or lowered, but affects the inset of the caption within the panel even if BevelInner is bvNone. If the Alignment property is not taCenter, the Caption will be aligned to the inner edge of the border. This edge is BorderWidth pixels in from the outer bevel if BevelInner is bvNone. It is the inner edge of the inner bevel otherwise.

Do not confuse the border of the panel with line drawn around the panel itself. The line around the panel is specified by the BorderStyle property.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
```

```
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

Alignment (see page 12539)

BevelInner (see page 12540)

BevelOuter (see page 12541)

BorderStyle (see page 12543)

Caption (see page 5806)

68.1.22.2.14 TFlowPanel.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
```

```
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

68.1.22.2.15 TFlowPanel.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

68.1.22.2.16 TFlowPanel.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

68.1.22.2.17 TFlowPanel.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

68.1.22.2.18 TFlowPanel.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite: Boolean;
```

C++

```
__property Boolean DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

68.1.22.2.19 TFlowPanel.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

68.1.22.2.20 TFlowPanel.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

68.1.22.2.21 TFlowPanel.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
```

```
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

68.1.22.2.22 TFlowPanel.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

68.1.22.2.23 TFlowPanel.FlowStyle Property

Specifies the starting point and direction for controls added to a flow panel.

Pascal

```
property FlowStyle: TFlowStyle;
```

C++

```
__property TFlowStyle FlowStyle;
```

Description

Use FlowStyle to set the starting position for the first control added to a flow panel, as well as the subsequent position for additional controls.

Flow Style	Description
BottomTopLeftRight	The first control is at the bottom left corner of the panel. Subsequent controls are added from bottom to top and then from left to right.

BottomTopRightLeft	The first control is at the bottom right corner of the panel. Subsequent controls are added from bottom to top and then from right to left.
LeftRightBottomTop	The first control is at the bottom left corner of the panel. Subsequent controls are added from left to right and then from bottom to top.
LeftRightTopBottom	The first control is at the top left corner of the panel. Subsequent controls are added from left to right and then from top to bottom. This is the default setting.
RightLeftBottomTop	The first control is at the bottom right corner of the panel. Subsequent controls are added from right to left and then from bottom to top.
RightLeftTopBottom	The first control is at the top right corner of the panel. Subsequent controls are added from right to left and then from top to bottom.
TopBottomLeftRight	The first control is at the top left corner of the panel. Subsequent controls are added from top to bottom and then from left to right.
TopBottomRightLeft	The first control is at the top right corner of the panel. Subsequent controls are added from top to bottom and then from right to left.

68.1.22.2.24 TFlowPanel.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
```

```
  corresponding font name in the combo box.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    i : Integer;
  begin
    for i := 0 to Screen.Fonts.Count - 1 do
     ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
  //  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
    RichEdit1.Lines.Text := 'How is it to be you?';
  end;

  procedure TForm1.ComboBox1Click(Sender: TObject);
  begin
    RichEdit1.Font.Name := ComboBox1.Text;
  end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

68.1.22.2.25 TFlowPanel.FullRepaint Property

Determines how the panel repaints itself when it is resized.

Pascal

```
property FullRepaint: Boolean;
```

C++

```
__property Boolean FullRepaint;
```

Description

FullRepaint controls how the panel responds when it is resized. When FullRepaint is true, the entire panel, including the beveled border repaints when the size changes. When FullRepaint is false, only the area inside the beveled border repaints.

See Also

Invalidate (see page 6161)

68.1.22.2.26 TFlowPanel.Locked Property

Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.

Pascal

```
property Locked: Boolean;
```

C++

```
__property Boolean Locked;
```

Description

Set the Locked property to false when the panel is used as a toolbar by a TOleContainer component. Any panels aligned to a TOleContainer are assumed to be replaceable toolbars unless the Locked property is true. Prevent the panel from being replaced by an OLE server's toolbar by setting the panel's Locked property to true.

Note: An OLE application's toolbars must be in panels aligned to a side of the form (that is, their Align properties must be set to alTop, alBottom, alLeft, or alRight).

See Also

Align (see page 5798)

TOleContainer (see page 17921)

68.1.22.2.27 TFlowPanel.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding: TPadding;
```

C++

```
__property TPadding Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding (see page 6103)

68.1.22.2.28 TFlowPanel.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground: Boolean;
```

C++

```
__property Boolean ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

68.1.22.2.29 TFlowPanel.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

68.1.22.2.30 TFlowPanel.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

68.1.22.2.31 TFlowPanel.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

68.1.22.2.32 TFlowPanel.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

68.1.22.2.33 TFlowPanel.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

68.1.22.2.34 TFlowPanel.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
```

```
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
```

```
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
```

```
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

68.1.22.2.35 TFlowPanel.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

68.1.22.2.36 TFlowPanel.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

68.1.22.2.37 TFlowPanel.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

68.1.22.2.38 TFlowPanel.UseDockManager Property

Specifies whether the docking manager is used in drag-and-dock operations.

Pascal

```
property UseDockManager: Boolean;
```

C++

```
__property Boolean UseDockManager;
```

Description

Use UseDockManager to get or set whether a docking manager is used in drag-and-dock operations.

The docking manager handles the positioning of docked controls and any painting of docking zones around those controls. If you do not use a docking manager, docked controls are aligned to the windowed control based on the closest edge to where they are released. When you do not use a docking manager, there is no region for a user to grab to automatically undock a docked control.

See Also

DockManager (see page 6202)

68.1.22.2.39 TFlowPanel.VerticalAlignment Property

Sets the vertical position of the caption.

Pascal

```
property VerticalAlignment: TVerticalAlignment;
```

C++

```
__property TVerticalAlignment VerticalAlignment;
```

Description

Use the VerticalAlignment property to control the position of the caption within the panel. Choose one of the following values.

Value	Description
taAlignBottom	The caption displays at the bottom of the panel.
taAlignTop	The caption displays at the top of the panel.
taVerticalCenter	The caption displays at the center of the panel.

Note: Use the Alignment property to control the horizontal position of the caption.

See Also

Caption

68.1.22.2.40 TFlowPanel.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

68.1.22.3 TFlowPanel Events

The events of the TFlowPanel class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 12626)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 12626)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 12627)	Occurs when an attempt is made to resize the control.
	OnClick (see page 12627)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 12629)	Adjust resize constraints.
	OnContextPopup (see page 12630)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 12631)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 12631)	Occurs when another control is docked to the control.
	OnDockOver (see page 12633)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 12635)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 12637)	Occurs when the user drags an object over a control.
	OnEndDock (see page 12640)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 12640)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 12641)	Occurs when a control receives the input focus.
	OnExit (see page 12643)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 12645)	Returns the control's docking information.
	OnMouseActivate (see page 12647)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 12648)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 12650)	Occurs when the user moves the mouse into a control.

	OnMouseLeave (see page 12651)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 12651)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 12653)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 12655)	Occurs immediately after the control is resized.
	OnStartDock (see page 12656)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 12657)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 12657)	Occurs when the application tries to undock a control that is docked to the windowed control.

68.1.22.3.1 TFlowPanel.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore: TAlignInsertBeforeEvent;
```

C++

```
__property TAlignInsertBeforeEvent OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore (see page 6137)

Align (see page 5798)

TAlignInsertBeforeEvent (see page 6293)

68.1.22.3.2 TFlowPanel.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition: TAlignPositionEvent;
```

C++

```
__property TAlignPositionEvent OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition (see page 6138)

Align (see page 5798)

Anchors (see page 5802)

TAlignInfo (see page 6269)

TAlignPositionEvent (see page 6293)

68.1.22.3.3 TFlowPanel.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize: TCanResizeEvent;
```

C++

```
__property TCanResizeEvent OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize (see page 5853)

SetBounds (see page 5783)

OnResize (see page 5870)

68.1.22.3.4 TFlowPanel.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
```

```
  edit control placed somewhere on the client area of the
  sheet.  Unchecking the checkbox will cause the client area
  (not the tab area) of the Tab Control to become invisible.
  }
  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    PageControl1.ActivePage.Visible := CheckBox1.Checked;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    CheckBox1.Checked := PageControl1.ActivePage.Visible;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

68.1.22.3.5 TFlowPanel.OnConstrainedResize Event

Adjust resize constraints.

Pascal

```
property OnConstrainedResize: TConstrainedResizeEvent;
```

C++

```
__property TConstrainedResizeEvent OnConstrainedResize;
```

Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being

applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

Note: The OnConstrainedResize handler is called immediately after the OnCanResize handler.

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

Constraints (see page 5812)

OnResize (see page 5870)

CanAutoSize (see page 5741)

68.1.22.3.6 TFlowPanel.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

68.1.22.3.7 TFlowPanel.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

68.1.22.3.8 TFlowPanel.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop: TDockDropEvent;
```

C++

```
__property TDockDropEvent OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
```

```
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
```

```
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockOver (see page 6216)

68.1.22.3.9 TFlowPanel.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver: TDockOverEvent;
```

C++

```
__property TDockOverEvent OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
```

```
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
```

```
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnGetSiteInfo (see page 6223)

TDragDockObject (see page 6009)

68.1.22.3.10 TFlowPanel.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
```

```
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:
```

```
procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

68.1.22.3.11 TFlowPanel.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler

is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;
```

```
procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.22.3.12 TFlowPanel.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.22.3.13 TFlowPanel.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

68.1.22.3.14 TFlowPanel.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
```

```
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

68.1.22.3.15 TFlowPanel.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

68.1.22.3.16 TFlowPanel.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo: TGetSiteInfoEvent;
```

C++

```
__property TGetSiteInfoEvent OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
```

```
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
```

```
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

TDragDockObject (see page 6009)

68.1.22.3.17 TFlowPanel.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of

whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.22.3.18 TFlowPanel.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
```

```
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
```

```
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.22.3.19 TFlowPanel.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

68.1.22.3.20 TFlowPanel.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

68.1.22.3.21 TFlowPanel.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;
```

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.22.3.22 TFlowPanel.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
```

```
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
```

```
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

68.1.22.3.23 TFlowPanel.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

68.1.22.3.24 TFlowPanel.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

68.1.22.3.25 TFlowPanel.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

68.1.22.3.26 TFlowPanel.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock: TUnDockEvent;
```

C++

```
__property TUnDockEvent OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
```

```
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
```

```
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
```

```
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

68.1.23 TGridPanel Class

TGridPanel implements a grid panel control in which each component is placed within a cell on a grid.

Class Hierarchy

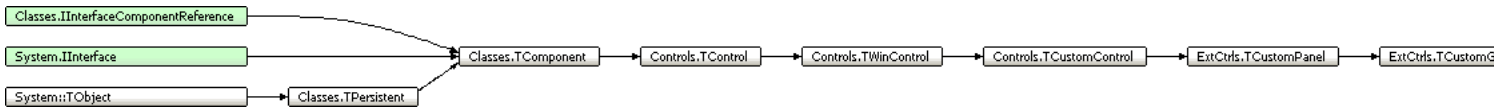

Pascal

```
TGridPanel = class(TCustomGridPanel);
```

C++

```
class TGridPanel : public TCustomGridPanel;
```

File

ExtCtrls

Description

Use TGridPanel to put an empty grid panel on a form. The major difference between a traditional panel and a grid panel is the way in which controls are placed. With a traditional panel, you place a control (such as a button) in a specific location. You can freely move that control to any location within the panel using the mouse. In a grid panel, you specify the number of rows and columns on the panel, then each component is placed into the next available cell in the grid, regardless of where you place it with the mouse. If each grid contains a control and you add more controls, the location for the new controls is determined by the ExpandStyle (see page 12694) property. For example, using the default ExpandStyle (see page 12694) property of AddRows, a new row is added to the grid, and each cell in the row can contain a new control.

See Also

TPanel (see page 13032)

68.1.23.1 TGridPanel Members

The following tables list the members exposed by TGridPanel.

Protected Events

	Name	Description
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 12537)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 12502)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	AutoAddColumn (see page 12502)	This is AutoAddColumn, a member of class TCustomGridPanel.
	AutoAddRow (see page 12503)	This is AutoAddRow, a member of class TCustomGridPanel.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.

	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanAutoSize (see page 12537)	Adjusts the way the panel responds to resize attempts.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	CellIndexToCell (see page 12503)	This is CellIndexToCell, a member of class TCustomGridPanel.
	CellToCellIndex (see page 12503)	This is CellToCellIndex, a member of class TCustomGridPanel.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 12537)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.

	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.

	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 12504)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 5920)	Renders the image of a custom control.
V	Paint (see page 12505)	Renders the image of the panel.
V	Paint (see page 12538)	Renders the image of the panel.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveEmptyAutoAddColumns (see page 12505)	This is RemoveEmptyAutoAddColumns, a member of class TCustomGridPanel.
	RemoveEmptyAutoAddRows (see page 12505)	This is RemoveEmptyAutoAddRows, a member of class TCustomGridPanel.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBackground (see page 12539)	This is SetParentBackground, a member of class TCustomPanel.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
V	UpdateControlOriginalParentSize (see page 12505)	This is UpdateControlOriginalParentSize, a member of class TCustomGridPanel.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.

	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 5731)	Creates an instance of TControl (see page 5724).
	Create (see page 5918)	Creates an instance of TCustomControl (see page 5908).
	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 12500)	Creates and initializes an instance of TCustomPanel.
	Create (see page 12536)	Creates and initializes an instance of TCustomPanel (see page 12525).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).

	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 12501)	Destroys an instance of TCustomControl.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetChildren (see page 12503)	Calls a specified method for each child of the control.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetControlsAlignment (see page 12538)	Returns the alignment of text in the panel.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsColumnEmpty (see page 12504)	This is IsColumnEmpty, a member of class TCustomGridPanel.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	IsRowEmpty (see page 12504)	This is IsRowEmpty, a member of class TCustomGridPanel.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.

	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	CellCount (see page 12506)	This is CellCount, a member of class TCustomGridPanel.
R	CellRect (see page 12506)	Returns the screen coordinates of a cell in the grid.
R	CellSize (see page 12507)	This is CellSize, a member of class TCustomGridPanel.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ColumnSpanIndex (see page 12507)	This is ColumnSpanIndex, a member of class TCustomGridPanel.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 12688)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	RowSpanIndex (see page 12508)	This is RowSpanIndex, a member of class TCustomGridPanel.

R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 12709)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 12710)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 12711)	Occurs when an attempt is made to resize the control.
	OnClick (see page 12711)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 12713)	Adjust resize constraints.
	OnContextPopup (see page 12713)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 12714)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 12714)	Occurs when another control is docked to the control.
	OnDockOver (see page 12716)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 12719)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 12721)	Occurs when the user drags an object over a control.
	OnEndDock (see page 12723)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 12724)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 12725)	Occurs when a control receives the input focus.
	OnExit (see page 12727)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 12729)	Returns the control's docking information.
	OnMouseActivate (see page 12731)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 12732)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 12734)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 12734)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 12734)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 12737)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 12739)	Occurs immediately after the control is resized.
	OnStartDock (see page 12740)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 12740)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 12741)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 12673)	Determines how the control aligns within its container (parent control).
	Alignment (see page 12676)	Determines how the caption is aligned within the panel.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 12677)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 12677)	Specifies which edges of the control are beveled.
	BevelInner (see page 12677)	Determines the style of the inner bevel of a panel.
	BevelKind (see page 12678)	Specifies the control's bevel style.
	BevelOuter (see page 12679)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 12680)	Determines the width, in pixels, of both the inner and outer bevels of a panel.

	BiDiMode (see page 12681)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 12681)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 12682)	Specifies the distance, in pixels, between the outer and inner bevels.
	Caption (see page 12683)	Specifies a text string that identifies the control to the user.
	Color (see page 12686)	Specifies the background color of the control.
	ColumnCollection (see page 12687)	Sets the number and size of the columns on the grid panel.
	Constraints (see page 12687)	Specifies the size constraints for the control.
	ControlCollection (see page 12687)	This is ControlCollection, a member of class TGridPanel.
	Ctl3D (see page 12688)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DockSite (see page 12690)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 12691)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 12692)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 12692)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 12693)	Controls whether the control responds to mouse, keyboard, and timer events.
	ExpandStyle (see page 12694)	Specifies whether rows or columns are added to a filled grid panel to accommodate additional controls.
	Font (see page 12694)	Controls the attributes of text written on or in the control.
	FullRepaint (see page 12695)	Determines how the panel repaints itself when it is resized.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Locked (see page 12696)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Padding (see page 12696)	Specifies the padding of a control.
	ParentBackground (see page 12696)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 12697)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 12697)	Determines where a control looks for its color information.
	ParentCtl3D (see page 12697)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 12698)	Determines where a control looks for its font information.
	ParentShowHint (see page 12698)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 12699)	Identifies the pop-up menu associated with the control.
	RowCollection (see page 12703)	Sets the number and size of the rows on the grid panel.
	ShowHint (see page 12704)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 12705)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 12706)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	UseDockManager (see page 12706)	Specifies whether the docking manager is used in drag-and-dock operations.
	VerticalAlignment (see page 12707)	Sets the vertical position of the caption.
	Visible (see page 12707)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.23.2 TGridPanel Properties

The properties of the TGridPanel class are listed here.

Public Properties

	Name	Description
	DockManager (see page 12688)	Specifies the control's docking manager interface.

Published Properties

	Name	Description
	Align (see page 12673)	Determines how the control aligns within its container (parent control).
	Alignment (see page 12676)	Determines how the caption is aligned within the panel.
	Anchors (see page 12677)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 12677)	Specifies which edges of the control are beveled.
	BevelInner (see page 12677)	Determines the style of the inner bevel of a panel.
	BevelKind (see page 12678)	Specifies the control's bevel style.
	BevelOuter (see page 12679)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 12680)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BiDiMode (see page 12681)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 12681)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 12682)	Specifies the distance, in pixels, between the outer and inner bevels.
	Caption (see page 12683)	Specifies a text string that identifies the control to the user.
	Color (see page 12686)	Specifies the background color of the control.
	ColumnCollection (see page 12687)	Sets the number and size of the columns on the grid panel.
	Constraints (see page 12687)	Specifies the size constraints for the control.
	ControlCollection (see page 12687)	This is ControlCollection, a member of class TGridPanel.
	Ctl3D (see page 12688)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 12690)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 12691)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 12692)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 12692)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 12693)	Controls whether the control responds to mouse, keyboard, and timer events.
	ExpandStyle (see page 12694)	Specifies whether rows or columns are added to a filled grid panel to accommodate additional controls.
	Font (see page 12694)	Controls the attributes of text written on or in the control.
	FullRepaint (see page 12695)	Determines how the panel repaints itself when it is resized.
	Locked (see page 12696)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	Padding (see page 12696)	Specifies the padding of a control.
	ParentBackground (see page 12696)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 12697)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 12697)	Determines where a control looks for its color information.
	ParentCtl3D (see page 12697)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 12698)	Determines where a control looks for its font information.
	ParentShowHint (see page 12698)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 12699)	Identifies the pop-up menu associated with the control.
	RowCollection (see page 12703)	Sets the number and size of the rows on the grid panel.
	ShowHint (see page 12704)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 12705)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 12706)	Determines if the user can tab to a control.
	UseDockManager (see page 12706)	Specifies whether the docking manager is used in drag-and-dock operations.

	VerticalAlignment (see page 12707)	Sets the vertical position of the caption.
	Visible (see page 12707)	Determines whether the component appears onscreen.

68.1.23.2.1 TGridPanel.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;
```

```
  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
```

```
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

68.1.23.2.2 TGridPanel.Alignment Property

Determines how the caption is aligned within the panel.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use the Alignment property to specify if the caption of a panel appears on the left, right, or center of the panel. For example, to use a panel as a status bar that displays the TApplication. ::Hint property as the panel's Caption property, set Alignment to taLeftJustify to place the hint text on the left side of the panel.

Note: Use the VerticalAlignment property to control the vertical position of the caption.

See Also

Hint (see page 13712)

Caption (see page 5806)

68.1.23.2.3 TGridPanel.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

68.1.23.2.4 TGridPanel.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

68.1.23.2.5 TGridPanel.BevelInner Property

Determines the style of the inner bevel of a panel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TPanelBevel BevelInner;__property TBevelCut BevelInner;
```

Description

Use BevelInner to provide the panel with a three-dimensional beveled look. A panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel, separated by the number of pixels specified in the BorderWidth property. The width of the bevels, if they are drawn, is the BevelWidth property.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelOuter (see page 12541)

BevelWidth (see page 12542)

BorderWidth (see page 12543)

68.1.23.2.6 TGridPanel.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

68.1.23.2.7 TGridPanel.BevelOuter Property

Determines the style of the outer bevel of a panel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TPanelBevel BevelOuter;__property TBevelCut BevelOuter;
```

Description

Use BevelOuter to provide the panel with a three-dimensional beveled look. A panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel, separated by the number of pixels specified in the BorderWidth property. The width of the outer bevel is specified in the BevelWidth property in pixels.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
```

```
  the panel the appearance of a status line by changing the
  value of the BevelInner, BevelOuter, BevelWidth, and
  BorderWidth properties.
  }
  procedure TForm1.CreateStatusLineClick(Sender: TObject);
  begin
    with Panel1 do
    begin
      Align := alBottom;
      BevelInner := bvLowered;
      BevelOuter := bvRaised;
      BorderWidth := 1;
      BevelWidth := 1;
    end;
  end;
```

See Also

BevelInner (see page 12540)

BevelWidth (see page 12542)

BorderWidth (see page 12543)

68.1.23.2.8 TGridPanel.BevelWidth Property

Determines the width, in pixels, of both the inner and outer bevels of a panel.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify how wide the inner or outer bevel should be. Do not confuse BevelWidth, which is the width of the bevels, with BorderWidth, which is the space between the bevels.

If both the BevelInner and BevelOuter properties are bvNone, BevelWidth has no effect. To remove both bevels, set the BevelInner and BevelOuter properties to bvNone, rather than setting the BevelWidth to 0, as this involves less overhead when painting.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
```

```
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelInner (see page 12540)

BevelOuter (see page 12541)

BorderWidth (see page 12543)

68.1.23.2.9 TGridPanel.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

68.1.23.2.10 TGridPanel.BorderStyle Property

Determines the style of the line drawn around the perimeter of the panel control.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to specify whether the panel has a single line drawn around it. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

Do not confuse the line drawn around the panel with the BorderWidth of the panel. The BorderWidth of the panel is the distance between the outer and inner bevels.

See Also

BorderWidth (see page 12543)

68.1.23.2.11 TGridPanel.BorderWidth Property

Specifies the distance, in pixels, between the outer and inner bevels.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to specify how wide the border around the panel should be. A value of 0 (zero) means no border should appear.

The border of a panel is the area between the outer and inner bevels. It is visible only if the inner bevel is raised or lowered, but affects the inset of the caption within the panel even if BevelInner is bvNone. If the Alignment property is not taCenter, the Caption will be aligned to the inner edge of the border. This edge is BorderWidth pixels in from the outer bevel if BevelInner is bvNone. It is the inner edge of the inner bevel otherwise.

Do not confuse the border of the panel with line drawn around the panel itself. The line around the panel is specified by the BorderStyle property.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
```

```
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

Alignment (see page 12539)

BevelInner (see page 12540)

BevelOuter (see page 12541)

BorderStyle (see page 12543)

Caption (see page 5806)

68.1.23.2.12 TGridPanel.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
```

```
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
```

```
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
```

```
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

68.1.23.2.13 TGridPanel.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

68.1.23.2.14 TGridPanel.ColumnCollection Property

Sets the number and size of the columns on the grid panel.

Pascal

```
property ColumnCollection: TColumnCollection;
```

C++

```
__property TColumnCollection ColumnCollection;
```

Description

Use the ColumnCollection property to set the number of columns as well as the size of each column. You can use the SizeStyle property to specify whether column sizes are based on a percentage of the available width, based on an absolute value, or automatically sized.

68.1.23.2.15 TGridPanel.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

68.1.23.2.16 TGridPanel.ControlCollection Property

Pascal

```
property ControlCollection: TControlCollection;
```

C++

```
__property TControlCollection ControlCollection;
```

Description

This is ControlCollection, a member of class TGridPanel.

68.1.23.2.17 TGridPanel.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

68.1.23.2.18 TGridPanel.DockManager Property

Specifies the control's docking manager interface.

Pascal

```
property DockManager: IDockManager;
```

C++

```
__property IDockManager DockManager;
```

Description

Use DockManager to specify the control's docking manager. The docking manager handles the layout of docking zones (where controls are docked) and any painting associated with docking zones.

If you set DockSite and UseDockManager to true, but do not assign a value to DockManager, the windowed control generates a default docking manager, using the global DefaultDockTreeClass variable.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/
```

```
void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;
```

```
procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

UseDockManager (see page 6212)

DefaultDockTreeClass (see page 6321)

68.1.23.2.19 TGridPanel.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite: Boolean;
```

C++

```
__property Boolean DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
```

```
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

68.1.23.2.20 TGridPanel.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

68.1.23.2.21 TGridPanel.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

68.1.23.2.22 TGridPanel.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
```

```
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

68.1.23.2.23 TGridPanel.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

68.1.23.2.24 TGridPanel.ExpandStyle Property

Specifies whether rows or columns are added to a filled grid panel to accommodate additional controls.

Pascal

```
property ExpandStyle: TExpandStyle;
```

C++

```
__property TExpandStyle ExpandStyle;
```

Description

Use the ExpandStyle property to specify how a grid panel is expanded when each cell of the grid contains a control, and a new control is added.

68.1.23.2.25 TGridPanel.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
```

```
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

68.1.23.2.26 TGridPanel.FullRepaint Property

Determines how the panel repaints itself when it is resized.

Pascal

```
property FullRepaint: Boolean;
```

C++

```
__property Boolean FullRepaint;
```

Description

FullRepaint controls how the panel responds when it is resized. When FullRepaint is true, the entire panel, including the beveled border repaints when the size changes. When FullRepaint is false, only the area inside the beveled border repaints.

See Also

Invalidate (see page 6161)

68.1.23.2.27 TGridPanel.Locked Property

Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.

Pascal

```
property Locked: Boolean;
```

C++

```
__property Boolean Locked;
```

Description

Set the Locked property to false when the panel is used as a toolbar by a TOleContainer component. Any panels aligned to a TOleContainer are assumed to be replaceable toolbars unless the Locked property is true. Prevent the panel from being replaced by an OLE server's toolbar by setting the panel's Locked property to true.

Note: An OLE application's toolbars must be in panels aligned to a side of the form (that is, their Align properties must be set to alTop, alBottom, alLeft, or alRight).

See Also

Align (see page 5798)

TOleContainer (see page 17921)

68.1.23.2.28 TGridPanel.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding: TPadding;
```

C++

```
__property TPadding Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding (see page 6103)

68.1.23.2.29 TGridPanel.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground: Boolean;
```

C++

```
__property Boolean ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

68.1.23.2.30 TGridPanel.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

68.1.23.2.31 TGridPanel.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

68.1.23.2.32 TGridPanel.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

68.1.23.2.33 TGridPanel.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

68.1.23.2.34 TGridPanel.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

68.1.23.2.35 TGridPanel.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
```

```
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Cut1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      const int num_items = 4;
      for (int index = 0; index < num_items; ++index)
      {
          TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
          PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
          NewItem->Caption = "Menu Item " + IntToStr(index);
          NewItem->Tag = index;
          NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
          TNotifyEvent();
      }
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
          {
```

```
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
```

```
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
```

```
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

68.1.23.2.36 TGridPanel.RowCollection Property

Sets the number and size of the rows on the grid panel.

Pascal

```
property RowCollection: TRowCollection;
```

C++

```
__property TRowCollection RowCollection;
```

Description

Use the RowCollection property to set the number of rows as well as the size of each row. You can use the SizeStyle property to specify whether row sizes are based on a percentage of the available width, based on an absolute value, or automatically sized.

68.1.23.2.37 TGridPanel.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
```

```
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

68.1.23.2.38 TGridPanel.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

68.1.23.2.39 TGridPanel.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

68.1.23.2.40 TGridPanel.UseDockManager Property

Specifies whether the docking manager is used in drag-and-dock operations.

Pascal

```
property UseDockManager: Boolean;
```

C++

```
__property Boolean UseDockManager;
```

Description

Use UseDockManager to get or set whether a docking manager is used in drag-and-dock operations.

The docking manager handles the positioning of docked controls and any painting of docking zones around those controls. If you do not use a docking manager, docked controls are aligned to the windowed control based on the closest edge to where they are released. When you do not use a docking manager, there is no region for a user to grab to automatically undock a docked control.

See Also

DockManager (see page 6202)

68.1.23.2.41 TGridPanel.VerticalAlignment Property

Sets the vertical position of the caption.

Pascal

```
property VerticalAlignment: TVerticalAlignment;
```

C++

```
__property TVerticalAlignment VerticalAlignment;
```

Description

Use the VerticalAlignment property to control the position of the caption within the panel. Choose one of the following values.

Value	Description
taAlignBottom	The caption displays at the bottom of the panel.
taAlignTop	The caption displays at the top of the panel.
taVerticalCenter	The caption displays at the center of the panel.

Note: Use the Alignment property to control the horizontal position of the caption.

See Also

Caption

68.1.23.2.42 TGridPanel.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

68.1.23.3 TGridPanel Events

The events of the TGridPanel class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 12709)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 12710)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 12711)	Occurs when an attempt is made to resize the control.
	OnClick (see page 12711)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 12713)	Adjust resize constraints.
	OnContextPopup (see page 12713)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 12714)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 12714)	Occurs when another control is docked to the control.
	OnDockOver (see page 12716)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 12719)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 12721)	Occurs when the user drags an object over a control.
	OnEndDock (see page 12723)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 12724)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 12725)	Occurs when a control receives the input focus.
	OnExit (see page 12727)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 12729)	Returns the control's docking information.
	OnMouseActivate (see page 12731)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 12732)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 12734)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 12734)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 12734)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 12737)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 12739)	Occurs immediately after the control is resized.
	OnStartDock (see page 12740)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 12740)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 12741)	Occurs when the application tries to undock a control that is docked to the windowed control.

68.1.23.3.1 TGridPanel.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore: TAlignInsertBeforeEvent;
```

C++

```
__property TAlignInsertBeforeEvent OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore (see page 6137)

Align (see page 5798)

TAlignInsertBeforeEvent (see page 6293)

68.1.23.3.2 TGridPanel.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition: TAlignPositionEvent;
```

C++

```
__property TAlignPositionEvent OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition (see page 6138)

Align (see page 5798)

Anchors (see page 5802)

TAlignInfo (see page 6269)

TAlignPositionEvent (see page 6293)

68.1.23.3.3 TGridPanel.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize: TCanResizeEvent;
```

C++

```
__property TCanResizeEvent OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize (see page 5853)

SetBounds (see page 5783)

OnResize (see page 5870)

68.1.23.3.4 TGridPanel.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
```

```
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

68.1.23.3.5 TGridPanel.OnConstrainedResize Event

Adjust resize constraints.

Pascal

```
property OnConstrainedResize: TConstrainedResizeEvent;
```

C++

```
__property TConstrainedResizeEvent OnConstrainedResize;
```

Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

Note: The OnConstrainedResize handler is called immediately after the OnCanResize handler.

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

Constraints (see page 5812)

OnResize (see page 5870)

CanAutoSize (see page 5741)

68.1.23.3.6 TGridPanel.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

68.1.23.3.7 TGridPanel.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

68.1.23.3.8 TGridPanel.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop: TDockDropEvent;
```

C++

```
__property TDockDropEvent OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
```

```
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockOver (see page 6216)

68.1.23.3.9 TGridPanel.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver: TDockOverEvent;
```

C++

```
__property TDockOverEvent OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
```

```
      ShowMessage("GetSiteInfo CanDock went bad!");
    }
  }
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnGetSiteInfo (see page 6223)

TDragDockObject (see page 6009)

68.1.23.3.10 TGridPanel.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

68.1.23.3.11 TGridPanel.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}
```

```
/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
```

```
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.23.3.12 TGridPanel.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.23.3.13 TGridPanel.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

68.1.23.3.14 TGridPanel.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
```

```
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
```

```
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

68.1.23.3.15 TGridPanel.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK

button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

68.1.23.3.16 TGridPanel.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo: TGetSiteInfoEvent;
```

C++

```
__property TGetSiteInfoEvent OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
```

```
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

TDragDockObject (see page 6009)

68.1.23.3.17 TGridPanel.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.23.3.18 TGridPanel.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
```

```
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
```

```
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.23.3.19 TGridPanel.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

68.1.23.3.20 TGridPanel.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

68.1.23.3.21 TGridPanel.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
```

```
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
```

```
      end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.23.3.22 TGridPanel.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}
```

```
// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
```

```
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

68.1.23.3.23 TGridPanel.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

68.1.23.3.24 TGridPanel.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

68.1.23.3.25 TGridPanel.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible

for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

68.1.23.3.26 TGridPanel.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock: TUnDockEvent;
```

C++

```
__property TUnDockEvent OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/
```

```
void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
```

```
    if Client is TDockableForm then
      TDockableForm(Client).DockSite := True;
    if (DockClientCount = 2) and (NewTarget <> Self) then
      PostMessage(Self.Handle, WM_CLOSE, 0, 0);
    UpdateCaption(Client);
  end;
  {
  The following example is taken from the docking demo. It
  shows how set permissions to dock dockable objects onto a
  docking site.
  }

  procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer);
  begin
    //OnDockDrop gets called AFTER the client has actually docked,
    //so we check for DockClientCount = 1 before making the dock panel visible.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, True, nil);
    (Sender as TPanel).DockManager.ResetBounds(True);
    //Make DockManager repaints it's clients.
  end;

  procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer; State: TDragState;
    var Accept: Boolean);
  var
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

68.1.24 THeader Class

THeader is a sectioned visual control that displays text and allows each section to be resized with the mouse.

Class Hierarchy

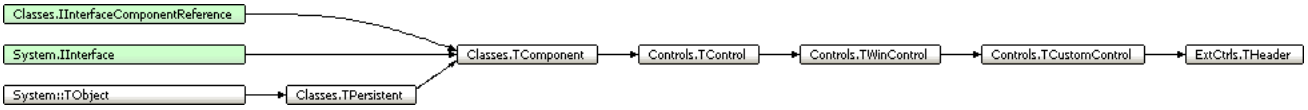

Pascal

```
THeader = class(TCustomControl);
```

C++

```
class THeader : public TCustomControl;
```

File

ExtCtrls

Description

THeader is provided for backward compatibility. New applications should use THeaderControl instead.

68.1.24.1 THeader Members

The following tables list the members exposed by THeader.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.

	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 12755)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.

	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.

	MouseDown (see page 12756)	OnMouseDown event dispatcher.
	MouseMove (see page 12756)	Respond to mouse moving over control area..
	MouseUp (see page 12757)	OnMouseUp event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 12757)	Renders the image of a custom control.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.

V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	Sized (see page 12758)	This is Sized, a member of class THeader.
	Sizing (see page 12758)	This is Sizing, a member of class THeader.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.

	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 12754)	Creates a new THeader (see page 12744) object.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 12755)	Frees the memory associated with the THeader (see page 12744) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.

	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.

	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.

R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SectionWidth (see page 12771)	Determines the width, in pixels, of the sections of a header.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnContextPopup (see page 12776)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnSized (see page 12777)	Occurs after the header has been resized.
	OnSizing (see page 12777)	Occurs before the header is resized.

Published Properties

	Name	Description
	Align (see page 12759)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AllowResize (see page 12762)	Determines if the user can modify the size of the header at runtime with the mouse.
	Anchors (see page 12763)	Specifies how the control is anchored to its parent.
	BorderStyle (see page 12763)	Determines whether the header control has a border.
	Constraints (see page 12763)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Enabled (see page 12764)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 12764)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentFont (see page 12766)	Determines where a control looks for its font information.
	ParentShowHint (see page 12766)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 12767)	Identifies the pop-up menu associated with the control.
	Sections (see page 12771)	Lists the strings that contain the text for the sections of a header.
	ShowHint (see page 12772)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 12773)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 12774)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 12774)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.24.2 THeader.Create Constructor

Creates a new THeader (see page 12744) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall THeader(TComponent * AOwner);
```

Description

Call Create to instantiate a header at runtime. Headers added to forms at design time are created automatically.

The AOwner parameter initializes the Owner property of the header.

See Also

Owner (see page 2807)

68.1.24.3 THeader.Destroy Destructor

Frees the memory associated with the THeader (see page 12744) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THeader();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks whether the THeader (see page 12744) reference is nil before calling Destroy.

See Also

Free (see page 22300)

68.1.24.4 THeader Methods

The methods of the THeader class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 12755)	Initializes a window-creation parameter data structure.
	MouseDown (see page 12756)	OnMouseDown event dispatcher.
	MouseMove (see page 12756)	Respond to mouse moving over control area..
	MouseUp (see page 12757)	OnMouseUp event dispatcher.
	Paint (see page 12757)	Renders the image of a custom control.
	Sized (see page 12758)	This is Sized, a member of class THeader.
	Sizing (see page 12758)	This is Sizing, a member of class THeader.

68.1.24.4.1 THeader.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call

the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

68.1.24.4.2 THeader.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove (see page 5773)

MouseUp (see page 5774)

OnMouseDown (see page 5862)

68.1.24.4.3 THeader.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown (see page 5773)

MouseUp (see page 5774)

OnMouseMove (see page 5865)

68.1.24.4.4 THeader.MouseUp Method

OnMouseUp event dispatcher.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user releases a previously pressed mouse button while the cursor's hotspot is over the control.

A control calls MouseUp in response to any of the Windows mouse-up messages (WM_LBUTTONUP, WM_MBUTTONUP, WM_RBUTTONUP), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

See Also

MouseDown (see page 5773)

MouseMove (see page 5773)

OnMouseUp (see page 5867)

68.1.24.4.5 THeader.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device

context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas (see page 5921)

PaintWindow (see page 5920)

GetDeviceContext (see page 6155)

ClipRect (see page 14486)

68.1.24.4.6 THeader.Sized Method

Pascal

```
procedure Sized(ASection: Integer; AWidth: Integer); dynamic;
```

C++

```
__fastcall Sized(int ASection, int AWidth);
```

Description

This is Sized, a member of class THeader.

68.1.24.4.7 THeader.Sizing Method

Pascal

```
procedure Sizing(ASection: Integer; AWidth: Integer); dynamic;
```

C++

```
__fastcall Sizing(int ASection, int AWidth);
```

Description

This is Sizing, a member of class THeader.

68.1.24.5 THeader Properties

The properties of the THeader class are listed here.

Public Properties

	Name	Description
	SectionWidth (see page 12771)	Determines the width, in pixels, of the sections of a header.

Published Properties

	Name	Description
	Align (see page 12759)	Determines how the control aligns within its container (parent control).
	AllowResize (see page 12762)	Determines if the user can modify the size of the header at runtime with the mouse.
	Anchors (see page 12763)	Specifies how the control is anchored to its parent.
	BorderStyle (see page 12763)	Determines whether the header control has a border.
	Constraints (see page 12763)	Specifies the size constraints for the control.
	Enabled (see page 12764)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 12764)	Controls the attributes of text written on or in the control.

	ParentFont (see page 12766)	Determines where a control looks for its font information.
	ParentShowHint (see page 12766)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 12767)	Identifies the pop-up menu associated with the control.
	Sections (see page 12771)	Lists the strings that contain the text for the sections of a header.
	ShowHint (see page 12772)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 12773)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 12774)	Determines if the user can tab to a control.
	Visible (see page 12774)	Determines whether the component appears onscreen.

68.1.24.5.1 THeader.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
```

```
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

68.1.24.5.2 THeader.AllowResize Property

Determines if the user can modify the size of the header at runtime with the mouse.

Pascal

```
property AllowResize: Boolean;
```

C++

```
__property Boolean AllowResize;
```

Description

If AllowResize is false, the sections within a header can't be resized. If AllowResize is true, clicking a border of a header section and dragging it left or right changes the width of the section. The default value is true.

68.1.24.5.3 THeader.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

68.1.24.5.4 THeader.BorderStyle Property

Determines whether the header control has a border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

68.1.24.5.5 THeader.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

68.1.24.5.6 THeader.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

68.1.24.5.7 THeader.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

68.1.24.5.8 THeader.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

68.1.24.5.9 THeader.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

68.1.24.5.10 THeader.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      const int num_items = 4;
      for (int index = 0; index < num_items; ++index)
      {
          TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
          PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
          NewItem->Caption = "Menu Item " + IntToStr(index);
          NewItem->Tag = index;
          NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
          TNotifyEvent();
      }
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
          {
              case 0:
              {
                  ShowMessage("first item clicked");
                  break;
              }
              case 1:
              {
                  ShowMessage("second item clicked");
                  break;
              }
              case 2:
              {
                  ShowMessage("third item clicked");
                  break;
              }
              case 3:
              {
```

```
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
```

```
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
```

```
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

68.1.24.5.11 THeader.Sections Property

Lists the strings that contain the text for the sections of a header.

Pascal

```
property Sections: TStrings;
```

C++

```
__property TStrings Sections;
```

Description

The number of lines of the string list determines the number of sections of the header. If the string list is empty, the header has one blank section. If this string list contains one or more lines, the text of each line is in its own section. The first line is in the leftmost section, the second line is in the next section to the right, and so on.

68.1.24.5.12 THeader.SectionWidth Property

Determines the width, in pixels, of the sections of a header.

Pascal

```
property SectionWidth [X: Integer]: Integer;
```

C++

```
__property int SectionWidth[int X];
```

Description

Use SectionWidth to specify the width of each header section. X is an index into the sections, from 0 to the number of sections - 1. For example, the index of the first section is 0, the second section is 1, and so on.

68.1.24.5.13 THeader.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
```

```
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

68.1.24.5.14 THeader.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

68.1.24.5.15 THeader.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

68.1.24.5.16 THeader.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
```

```
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

68.1.24.6 THeader Events

The events of the THeader class are listed here.

Published Events

	Name	Description
	OnContextPopup (see page 12776)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnSized (see page 12777)	Occurs after the header has been resized.
	OnSizing (see page 12777)	Occurs before the header is resized.

68.1.24.6.1 THeader.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

68.1.24.6.2 THeader.OnSized Event

Occurs after the header has been resized.

Pascal

```
property OnSized: TSectionEvent;
```

C++

```
__property TSectionEvent OnSized;
```

Description

Write an OnSized event handler to perform special processing after the header has been resized. In this event, the AWidth parameter is the width of the section before it was resized. Use the SectionWidth (see page 12771) property to determine the new size of the section.

See Also

OnSizing (see page 12777)

SectionWidth (see page 12771)

68.1.24.6.3 THeader.OnSizing Event

Occurs before the header is resized.

Pascal

```
property OnSizing: TSectionEvent;
```

C++

```
__property TSectionEvent OnSizing;
```

Description

Write an OnSizing event handler to perform special processing immediately before the header is resized. In this event, AWidth is the width that the section will have after it is resized. Use the SectionWidth (see page 12771) property to determine the current size of the section.

See Also

OnSized (see page 12777)

SectionWidth (see page 12771)

68.1.25 TImage Class

TImage displays a graphical image.

Class Hierarchy

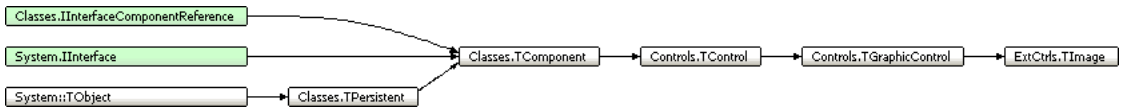

Pascal

```
TImage = class(TGraphicControl);
```

C++

```
class TImage : public TGraphicControl;
```

File

ExtCtrls

Description

Use TImage to display a graphical image on a form. Use the TPicture object in the Picture (see page 12802) property to specify the actual bitmap, con, metafile, or other graphic object displayed by TImage. Properties and methods of TPicture can be used for such things as loading an image from file, clearing the image in the TImage, and assigning an image for another control. TImage introduces several properties to determine how the image is displayed within the boundaries of the TImage object.

To add an image to a form or data module so that it is available for display by other controls, such as action lists and bitmap buttons, use a TImageList control instead.

See Also

TImageList (see page 6078)

TPicture (see page 14584)

TBitmap (see page 14396)

TIcon (see page 14537)

TMetafile (see page 14549)

Adding Graphical Objects to a String List

Adding Images to an Application

Graphic Controls

Images

68.1.25.1 TImage Members

The following tables list the members exposed by TImage.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.

	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 12786)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestRect (see page 12787)	This is DestRect, a member of class TImage.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoPaletteChange (see page 12787)	This is DoPaletteChange, a member of class TImage.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.

	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 12787)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Paint (see page 12788)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	Progress (see page 12788)	This is Progress, a member of class TImage.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.

	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.

	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 12785)	Creates and initializes an instance of TImage (see page 12777).
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 12786)	Destroys an instance of TImage (see page 12777).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.

	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Canvas (see page 12794)	Provides a drawing surface for embellishing bitmap images.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.

	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnClick (see page 12814)	Occurs when the user clicks the control.
	OnContextPopup (see page 12816)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 12817)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 12817)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 12819)	Occurs when the user drags an object over a control.
	OnEndDock (see page 12822)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 12822)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 12823)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 12824)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 12826)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 12826)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 12827)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 12829)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnProgress (see page 12831)	Occurs periodically during slow operations that affect the image.
	OnStartDock (see page 12832)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 12832)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 12789)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 12793)	Specifies how the control is anchored to its parent.
	AutoSize (see page 12793)	Specifies whether the control sizes itself automatically to accommodate the dimensions of the image.

	Center (see page 12797)	Indicates whether the image is centered in the image control.
	Constraints (see page 12797)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 12798)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 12798)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 12799)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 12800)	Controls whether the control responds to mouse, keyboard, and timer events.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	IncrementalDisplay (see page 12801)	Specifies whether successive approximations of the image should be drawn during slow operations.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentShowHint (see page 12801)	Determines where a control looks to find out if its Help Hint should be shown.
	Picture (see page 12802)	Specifies the image that appears on the image control.
	PopupMenu (see page 12803)	Identifies the pop-up menu associated with the control.
	Proportional (see page 12808)	Indicates whether the image should be changed, without distortion, so that it fits the bounds of the image control.
	ShowHint (see page 12808)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Stretch (see page 12810)	Indicates whether the image should be changed so that it exactly fits the bounds of the image control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Transparent (see page 12811)	Specifies whether the background of the image obscures objects below the image object.
	Visible (see page 12812)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.25.2 TImage.Create Constructor

Creates and initializes an instance of TImage (see page 12777).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TImage(TComponent * AOwner);
```

Description

Call Create to instantiate a TImage (see page 12777) object at runtime. Images placed on forms at design time are created automatically.

AOwner is the component that is responsible for freeing the TImage (see page 12777) instance. It becomes the value of the Owner property.

See Also

Owner (see page 2807)

68.1.25.3 TImage.Destroy Destructor

Destroys an instance of TImage (see page 12777).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TImage();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TImage (see page 12777) reference is not nil, and only then calls Destroy.

As the control is destroyed, it frees the TPicture object in the Picture (see page 12802) property.

See Also

Free (see page 22300)

68.1.25.4 TImage Methods

The methods of the TImage class are listed here.

Protected Methods

	Name	Description
V	CanAutoSize (see page 12786)	Indicates whether the control can resize itself automatically to accommodate its contents.
	DestRect (see page 12787)	This is DestRect, a member of class TImage.
	DoPaletteChange (see page 12787)	This is DoPaletteChange, a member of class TImage.
V	GetPalette (see page 12787)	Returns the handle to a palette for use by the control.
V	Paint (see page 12788)	Render control surface.
	Progress (see page 12788)	This is Progress, a member of class TImage.

68.1.25.4.1 TImage.CanAutoSize Method

Indicates whether the control can resize itself automatically to accommodate its contents.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize property is true and an attempt is made to resize the control. It allows controls to implement the AutoSize property before the resize sequence that begins with the OnCanResize and OnConstrainedResize events and ends with the OnResize event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the control.

CanAutoSize adjusts these values so that the control's size adjusts to its contents. If the adjustment results in values of NewWidth and NewHeight that differ from the control's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width and Height, CanAutoSize returns true, and the resize sequence is aborted.

As implemented in TControl, CanAutoSize always returns true without changing NewWidth or NewHeight. Override CanAutoSize to implement the AutoSize property for a particular control class's contents.

See Also

AutoSize (see page 5803)

OnResize (see page 5870)

OnCanResize (see page 5851)

OnConstrainedResize (see page 5853)

Width (see page 5849)

Height (see page 5825)

68.1.25.4.2 TImage.DestRect Method

Pascal

```
function DestRect: TRect;
```

C++

```
__fastcall TRect DestRect();
```

Description

This is DestRect, a member of class TImage.

68.1.25.4.3 TImage.DoPaletteChange Method

Pascal

```
function DoPaletteChange: Boolean;
```

C++

```
__fastcall Boolean DoPaletteChange();
```

Description

This is DoPaletteChange, a member of class TImage.

68.1.25.4.4 TImage.GetPalette Method

Returns the handle to a palette for use by the control.

Pascal

```
function GetPalette: HPALETTE; override;
```

C++

```
virtual __fastcall HPALETTE GetPalette();
```

Description

As implemented in TControl, GetPalette returns a null handle, indicating that the control does not have a palette. Override

GetPalette to implement derived objects that use a palette. Create a palette handle for the control by a call to the Windows API function CreatePalette. GetPalette should return the palette handle for the control.

Specifying the palette for a control tells the application that the control's palette needs to be realized and designates the palette to use for realization.

GetPalette is called only if the run-time video mode requires color palette support, for example, in 256-color mode.

See Also

PaletteChanged (see page 5775)

68.1.25.4.5 TImage.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas (see page 6053)

68.1.25.4.6 TImage.Progress Method

Pascal

```
procedure Progress(Sender: TObject; Stage: TProgressStage; PercentDone: Byte; RedrawNow:
Boolean; const R: TRect; const Msg: string); dynamic;
```

C++

```
__fastcall Progress(TObject * Sender, TProgressStage Stage, Byte PercentDone, Boolean
RedrawNow, const TRect R, const AnsiString Msg);
```

Description

This is Progress, a member of class TImage.

68.1.25.5 TImage Properties

The properties of the TImage class are listed here.

Public Properties

	Name	Description
R	Canvas (see page 12794)	Provides a drawing surface for embellishing bitmap images.

Published Properties

	Name	Description
	Align (see page 12789)	Determines how the control aligns within its container (parent control).
	Anchors (see page 12793)	Specifies how the control is anchored to its parent.
	AutoSize (see page 12793)	Specifies whether the control sizes itself automatically to accommodate the dimensions of the image.
	Center (see page 12797)	Indicates whether the image is centered in the image control.
	Constraints (see page 12797)	Specifies the size constraints for the control.
	DragCursor (see page 12798)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 12798)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 12799)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 12800)	Controls whether the control responds to mouse, keyboard, and timer events.
	IncrementalDisplay (see page 12801)	Specifies whether successive approximations of the image should be drawn during slow operations.
	ParentShowHint (see page 12801)	Determines where a control looks to find out if its Help Hint should be shown.
	Picture (see page 12802)	Specifies the image that appears on the image control.
	PopupMenu (see page 12803)	Identifies the pop-up menu associated with the control.
	Proportional (see page 12808)	Indicates whether the image should be changed, without distortion, so that it fits the bounds of the image control.
	ShowHint (see page 12808)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Stretch (see page 12810)	Indicates whether the image should be changed so that it exactly fits the bounds of the image control.
	Transparent (see page 12811)	Specifies whether the background of the image obscures objects below the image object.
	Visible (see page 12812)	Determines whether the component appears onscreen.

68.1.25.5.1 TImage.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of

the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
```

```
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form’s width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
```

```
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

68.1.25.5.2 TImage.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

68.1.25.5.3 TImage.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate the dimensions of the image.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the TImage (see page 12777) control adjusts its own dimensions to match the dimensions of the image that the Picture (see page 12802) property specifies. When AutoSize is true, the TImage (see page 12777) control forces its own height and width to match those of the picture it represents. When AutoSize is false (the default), the Center (see page 12797) and Stretch (see page 12810) properties determine how the image control deals with discrepancies between the size of the picture and the size of the image control.

See Also

Constraints (see page 5812)

Picture (see page 12802)

Width (see page 14602)

Height (see page 14600)

Height (see page 5825)

Width (see page 5849)

Center (see page 12797)

Stretch (see page 12810)

68.1.25.5.4 TImage.Canvas Property

Provides a drawing surface for embellishing bitmap images.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Canvas is the TCanvas object on which the image for the TImage (see page 12777) component is drawn. The painting of the image specified in the Picture (see page 12802) property is automatic and does not require manual intercession for the picture to appear.

You can use properties and methods of TCanvas to do such things as draw lines, draw images, and write text to the TImage (see page 12777) control's canvas.

Note: Canvas is only available if the Picture (see page 12802) property represents a TBitmap object. Trying to read Canvas when Picture (see page 12802) represents another type of graphic image causes TImage (see page 12777) to raise an EInvalidOperation exception.

C++ Examples:

```
/*
This example creates a region and selects this region as the
clipping rectangle for the Image component's canvas. It then
sets the canvas's brush color to red and calls FillRect
using the ClipRect as the area to fill. Lastly, the ClipRect
is reset to the original value that it contained by calling
SelectClipRect with nil as the second parameter and deletes
the region.  Place a TImage object in the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    HRGN MyRgn;

    MyRgn = ::CreateRectRgn(100,100,200,200);
    ::SelectClipRgn(Image1->Canvas->Handle,MyRgn);
    Image1->Canvas->Brush->Color = clRed;
    Image1->Canvas->FillRect(Image1->Canvas->ClipRect);
    Image1->Invalidate();
    ::SelectClipRgn(Image1->Canvas->Handle,NULL);
    ::DeleteObject(MyRgn);
}
/*
This example converts a specified icon to a bitmap. To run
this example, add an image, a button, and an Open dialog to
a form. Name the button ConvertIcon2Bitmap, and add the
following code as its OnClick event handler.
*/
void __fastcall TForm1::ConvertIcon2BitmapClick(TObject *Sender)
```

```
{
  OpenDialog1->DefaultExt = ".ICO";
  OpenDialog1->Filter = "icons (*.ico)|*.ICO";
  OpenDialog1->Options << ofOverwritePrompt << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TIcon *pi = new TIcon();
    try
    {
      AnsiString as;
      pi->LoadFromFile(OpenDialog1->FileName);
      as = ChangeFileExt(OpenDialog1->FileName,".BMP");
      Image1->Width = pi->Width;
      Image1->Height = pi->Height;
      Image1->Canvas->Draw(0,0,pi);
      Image1->Picture->SaveToFile(as);
      ShowMessage(OpenDialog1->FileName + " Saved to " + as);
    }
    __finally
    {
      delete pi;
    }
  }
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TImage *myImage;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TImage")
  {
    myImage = (TImage *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myImage->Canvas->MoveTo(myImage->ClientRect.Left, myImage->ClientRect.Top);
    myImage->Canvas->LineTo(myImage->ClientRect.Right, myImage->ClientRect.Bottom);
    myImage = NULL;
  }
}
/*
The following example paints a cross-hatched ellipse onto
Image1 which is a TImage on the form when the user clicks on
Button1.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TCanvas *pCanvas = Image1->Canvas;
  pCanvas->Brush->Color = clRed;
  pCanvas->Brush->Style = bsDiagCross;
  pCanvas->Ellipse(0, 0, Image1->Width, Image1->Height);
}
```

Delphi Examples:

```
{
This example creates a region and selects this region as the
clipping rectangle for the Image component's canvas. It then
sets the canvas's brush color to red and calls FillRect
using the ClipRect as the area to fill. Lastly, the ClipRect
is reset to the original value that it contained by calling
SelectClipRect with nil as the second parameter and deletes
```

```
the region.  Place a TImage object in the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
    MyRgn: HRGN;
begin
    MyRgn := CreateRectRgn(100,100,200,200);
    SelectClipRgn(Image1.Canvas.Handle,MyRgn);
    Image1.Canvas.Brush.Color := clRed;
    Image1.Canvas.FillRect(Image1.Canvas.ClipRect);
    Image1.Invalidate;
    SelectClipRgn(Image1.Canvas.Handle, HRGN(nil));
    DeleteObject(MyRgn);
end;
{
This example converts a specified icon to a bitmap. To run
this example, add an image, a button, and an Open dialog to
a form. Name the button ConvertIcon2Bitmap, and add the
following code as its OnClick event handler.
}
procedure TForm1.ConvertIcon2BitmapClick(Sender: TObject);
var
  s : string;
  Icon: TIcon;
begin
  OpenDialog1.DefaultExt := '.ICO';
  OpenDialog1.Filter := 'icons (*.ico)|*.ICO';
  OpenDialog1.Options := [ofOverwritePrompt, ofFileMustExist, ofHideReadOnly ];
  if OpenDialog1.Execute then
  begin
    Icon := TIcon.Create;
    try
      Icon.Loadfromfile(OpenDialog1.FileName);
      s:= ChangeFileExt(OpenDialog1.FileName,'.BMP');
      Image1.Width := Icon.Width;
      Image1.Height := Icon.Height;
      Image1.Canvas.Draw(0,0,Icon);
      Image1.Picture.SaveToFile(s);
      ShowMessage(OpenDialog1.FileName + ' Saved to ' + s);
    finally
      Icon.Free;
    end;
  end;
end;
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TImage) then
  begin
    myImage:= (ListBox1.Items.Objects[Index] as TImage);
    with myImage.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myImage.Canvas.MoveTo(Left, Top);
      myImage.Canvas.LineTo(Right, Bottom);
    end;
  myImage:= nil;
  end;
{
The following example paints a cross-hatched ellipse onto
Image1 which is a TImage on the form when the user clicks on
Button1.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Image1 do begin
    Canvas.Brush.Color := clRed;
    Canvas.Brush.Style := bsDiagCross;
    Canvas.Ellipse(0, 0, Image1.Width, Image1.Height);
  end;
end;
```

See Also

EInvalidOperation (see page 2700)

Picture (see page 12802)

TCanvas (see page 14441)

68.1.25.5.5 TImage.Center Property

Indicates whether the image is centered in the image control.

Pascal

```
property Center: Boolean;
```

C++

```
__property Boolean Center;
```

Description

When the image does not fit perfectly within the image control, use Center to specify how the image is positioned. When Center is true, the image is centered in the control. When Center is false, the upper left corner of the image is positioned at the upper left corner of the control.

The default value for Center is false.

Note: Center has no effect if the AutoSize (see page 12793) property is true or if the Stretch (see page 12810) property is true and Picture (see page 12802) does not specify an icon.

See Also

AutoSize (see page 5803)

Picture (see page 12802)

Stretch (see page 12810)

Proportional (see page 12808)

68.1.25.5.6 TImage.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

68.1.25.5.7 TImage.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

68.1.25.5.8 TImage.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

68.1.25.5.9 TImage.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
```

```
    AttachMode: TNodeAttachMode;
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

68.1.25.5.10 TImage.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

68.1.25.5.11 TImage.IncrementalDisplay Property

Specifies whether successive approximations of the image should be drawn during slow operations.

Pascal

```
property IncrementalDisplay: Boolean;
```

C++

```
__property Boolean IncrementalDisplay;
```

Description

Set IncrementalDisplay to true on large compressed images that take a long time to load or on images that may be altered by a computationally slow process. When IncrementalDisplay is true, partial renderings of the image are drawn periodically during slow operations such as loading. Set IncrementalDisplay to false to wait until the entire operation is complete before drawing the image.

For some classes of graphic objects, an OnProgress (see page 12831) event occurs at various points during slow operations. If IncrementalDisplay is true, the OnProgress (see page 12831) event occurs immediately after the approximate image is drawn. If IncrementalDisplay is false, OnProgress (see page 12831) occurs but the image is not updated until the entire image has been loaded.

The default value for IncrementalDisplay is false.

See Also

OnProgress (see page 12831)

68.1.25.5.12 TImage.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

68.1.25.5.13 TImage.Picture Property

Specifies the image that appears on the image control.

Pascal

```
property Picture: TPicture;
```

C++

```
__property TPicture Picture;
```

Description

Use Picture to specify the image for the TImage (see page 12777) component. Picture is a reference to a TPicture object. Use properties and methods of TPicture to specify a bitmap, icon, metafile, or user-defined graphic to be displayed by the image control.

Setting Picture at design time brings up the Picture Editor, which can be used to specify the file that contains the image.

C++ Examples:

```
/*
This example uses an image, a button, and a shape component
on a form. When the user clicks the button, an image of the
form is stored in the FormImage variable and copied to the
Clipboard. The image of the form is then copied back to
the image component, producing an interesting result,
especially if the button is clicked multiple times.  Add
ExtCtrls, StdCtrls, and Clipbrd to the uses clause.
}
*/

#include <Clipbrd.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *FormImage = GetFormImage();
  try
  {
    Clipboard()->Assign(FormImage);
    Image1->Picture->Assign(Clipboard());
  }
  __finally
  {
    delete FormImage;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Shape1->Shape = stEllipse;
  Shape1->Brush->Color = clLime;
  Image1->Stretch = true;
}
/*
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  PaintBox1->Canvas->Draw(0,0, Image1->Picture->Graphic);
}
```

Delphi Examples:

```
{
This example uses an image, a button, and a shape component
on a form. When the user clicks the button, an image of the
form is stored in the FormImage variable and copied to the
Clipboard. The image of the form is then copied back to
the image component, producing an interesting result,
especially if the button is clicked multiple times.  Add
ExtCtrls, StdCtrls, and Clipbrd to the uses clause.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FormImage: TBitmap;
begin
  FormImage := GetFormImage;
  try
    Clipboard.Assign(FormImage);
    Image1.Picture.Assign(Clipboard);
  finally
    FormImage.Free;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Shape1.Shape := stEllipse;
  Shape1.Brush.Color := clLime;
  Image1.Stretch := True;
end;
{
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  PaintBox1.Canvas.Draw(0,0, Image1.Picture.Graphic);
end;
```

See Also

Graphic (see page 14600)

68.1.25.5.14 TImage.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
```

```
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
```

```
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
```

```
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
```

```
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

68.1.25.5.15 TImage.Proportional Property

Indicates whether the image should be changed, without distortion, so that it fits the bounds of the image control.

Pascal

```
property Proportional: Boolean;
```

C++

```
__property Boolean Proportional;
```

Description

Set Proportional to true to ensure that the image can be fully displayed in the image control without any distortion such as occurs with the Stretch (see page 12810) property. When Proportional is true, images that are too large to fit in the image control are scaled down (while maintaining the same aspect ratio) until they fit in the image control. Images that are too small are displayed normally. That is, Proportional can reduce the magnification of the image, but does not increase it.

When the image control resizes, the image resizes also.

To resize the image so that it fits exactly in the image control, even if that causes distortion, use the Stretch (see page 12810) property instead.

To resize the control to the image rather than resizing the image to the control, use the AutoSize (see page 12793) property instead.

The default value for Proportional is false.

Note: Proportional has no effect if the Picture (see page 12802) property contains an icon.

See Also

AutoSize (see page 12793)

Picture (see page 12802)

Stretch (see page 12810)

68.1.25.5.16 TImage.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
```

```
    begin
      for I := 1 to 10 do
        Items.Add('Item ' + IntToStr(I));
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

68.1.25.5.17 TImage.Stretch Property

Indicates whether the image should be changed so that it exactly fits the bounds of the image control.

Pascal

```
property Stretch: Boolean;
```

C++

```
__property Boolean Stretch;
```

Description

Set Stretch to true to cause the image to assume the size and shape of the image control. When the image control resizes, the image resizes also. Stretch resizes the height and width of the image independently. Thus, unlike a simple change in magnification, Stretch can distort the image if the image control is not the same shape as the image.

To resize the image without any distortion, use the Proportional (see page 12808) property instead.

To resize the control to the image rather than resizing the image to the control, use the AutoSize (see page 12793) property instead.

The default value for Stretch is false.

Note: Stretch has no effect if the Picture (see page 12802) property contains an icon.

C++ Examples:

```
/*
This example converts a bitmap image to a jpeg file by using
the Assign method.  This example requires a button and two
images.  One image must have it's Picture property loaded
with a bitmap at design time.
*/

#include <Jpeg.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  //Requires "jpeg.hpp" to be included in the source file
```

```
  TJPEGImage *jp = new TJPEGImage();
  try
  {
    jp->Assign(Image1->Picture->Bitmap);
    jp->SaveToFile("..\\factory.jpg");
    Image2->Stretch = True;
    Image2->Picture->LoadFromFile("..\\factory.jpg");
  }
  __finally
  {
    delete jp;
  }
}
```

Delphi Examples:

```
{
This example assigns a bitmap contained in an ImageList to
an Image component's Picture's Bitmap property and displays
the bitmap.  TImage Stretch will fit the bitmap to the image
size.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   ImageList1.GetBitmap(2,Image1.Picture.Bitmap);
   Image1.Stretch := True;
end;
{
This example converts a bitmap image to a jpeg file by using
the Assign method.  This example requires a button and two
images.  One image must have it's Picture property loaded
with a bitmap at design time.
}
uses Jpeg;

procedure TForm1.Button1Click(Sender: TObject);
var
  jp: TJPEGImage;  //Requires the "jpeg" unit added to "uses" clause.
begin
  jp := TJPEGImage.Create;
  try
    with jp do
    begin
      Assign(Image1.Picture.Bitmap);
      SaveToFile('factory.jpg');
      Image2.Stretch := True;
      Image2.Picture.LoadFromFile('factory.jpg');
    end;
  finally
    jp.Free;
  end;
end;
```

See Also

AutoSize (see page 5803)

Picture (see page 12802)

Proportional (see page 12808)

68.1.25.5.18 TImage.Transparent Property

Specifies whether the background of the image obscures objects below the image object.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Set Transparent to true to allow objects behind the TImage (see page 12777) object to show through the background of the image. Set Transparent to false to make the background of the image opaque. Setting Transparent sets the Transparent property of the Picture (see page 12802).

Note: Transparent has no effect unless the Picture (see page 12802) property specifies a TBitmap object.

See Also

Picture (see page 12802)

Transparent (see page 14529)

Graphic (see page 14600)

68.1.25.5.19 TImage.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
```

```
      pPage->PageControl = PageControl1;
      pPage->Caption = AnsiString("Page") + IntToStr(i);
      TEdit *pEdit = new TEdit(this);
      pEdit->Parent = pPage;
      pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
      pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
    }
    PageControl1Change(Sender);
  }
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

68.1.25.6 TImage Events

The events of the TImage class are listed here.

Published Events

	Name	Description
	OnClick (see page 12814)	Occurs when the user clicks the control.
	OnContextPopup (see page 12816)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 12817)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 12817)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 12819)	Occurs when the user drags an object over a control.
	OnEndDock (see page 12822)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 12822)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 12823)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 12824)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 12826)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 12826)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 12827)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 12829)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnProgress (see page 12831)	Occurs periodically during slow operations that affect the image.
	OnStartDock (see page 12832)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 12832)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

68.1.25.6.1 TImage.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.

- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
```

```
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

68.1.25.6.2 TImage.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

68.1.25.6.3 TImage.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

68.1.25.6.4 TImage.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

68.1.25.6.5 TImage.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
```

```
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code comes from an application that contains a list box
  and three labels, each with a different font and color.  The
  DragMode property for each of the labels is dmAutomatic.
  The user can select a label and drag it to a list box and
  drop it. When the label is dropped, the items in the list
  box assume the color and font of the dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.25.6.6 TImage.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.25.6.7 TImage.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
```

```
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

68.1.25.6.8 TImage.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control.

However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.25.6.9 TImage.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
```

```
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
```

```
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.25.6.10 TImage.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

68.1.25.6.11 TImage.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

68.1.25.6.12 TImage.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}
```

```
// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
```

```
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.25.6.13 TImage.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
```

```
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;
```

```
procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

68.1.25.6.14 TImage.OnProgress Event

Occurs periodically during slow operations that affect the image.

Pascal

```
property OnProgress: TProgressEvent;
```

C++

```
__property TProgressEvent OnProgress;
```

Description

Write an OnProgress event handler to provide the user with feedback during slow operations such as loading large compressed images.

Note: The PercentDone parameter of the event handler is only an approximation. With some image formats, the value of PercentDone may actually decrease from the value in previous events, as the graphic object discovers there is more work to do.

See Also

IncrementalDisplay (see page 12801)

OnProgress (see page 14603)

TProgressBar (see page 4615)

68.1.25.6.15 TImage.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

68.1.25.6.16 TImage.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

68.1.26 TLabeledEdit Class

TLabeledEdit is an edit control that has an associated label.

Class Hierarchy

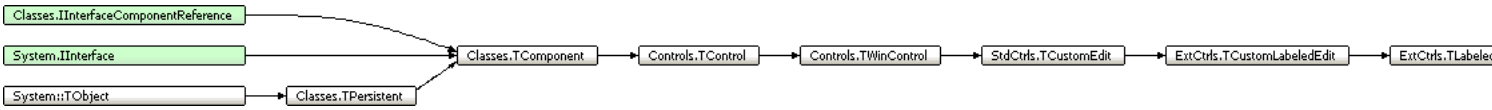

Pascal

```
TLabeledEdit = class(TCustomLabeledEdit);
```

C++

```
class TLabeledEdit : public TCustomLabeledEdit;
```

File

ExtCtrls

Description

Use TLabeledEdit to add an edit control with a label to a form. TLabeledEdit works with an associated label control, which is the value of its EditLabel (see page 12855) property. The position of that label relative to the edit control is determined by the LabelPosition (see page 12858) and LabelSpacing (see page 12859) properties.

Use the Text (see page 12870) property to work with the text in the control, and the label's Caption property to work with the text in the label.

See Also

TEdit (see page 21402)

TBoundLabel (see page 12123)

TMaskEdit (see page 15824)

68.1.26.1 TLabeledEdit Members

The following tables list the members exposed by TLabeledEdit.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.

	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21245)	Generates an OnChange (see page 21275) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	CMBidimodechanged (see page 12521)	This is CMBidimodechanged, a member of class TCustomLabeledEdit.
	CMEnabledchanged (see page 12521)	Responds when the Enabled property changes.
	CMVisiblechanged (see page 12521)	Responds when the Visible property changes.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 21248)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWindowHandle (see page 21248)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 21248)	Creates a Windows control corresponding to the edit control.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21251)	Destroys the Windows control that corresponds to the edit control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.

	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoSetMaxLength (see page 21251)	Sets the MaxLength (see page 21263) property.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelLength (see page 21252)	Returns the SelLength (see page 21265) property.
	GetSelStart (see page 21252)	Returns the SelStart (see page 21269) property.
	GetSelText (see page 21252)	Returns the SelText (see page 21272) property.

	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 12522)	Responds to notifications when objects are created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.

	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetAutoSize (see page 21257)	Sets the AutoSize (see page 21260) property.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetName (see page 12523)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParent (see page 12523)	Sets the value of the Parent property.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetSelLength (see page 21257)	Sets the SelLength (see page 21265) property.
V	SetSelStart (see page 21257)	Sets the SelStart (see page 21269) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 21245)	Deletes all text from the edit control.
	ClearSelection (see page 21245)	Removes the selected text from the edit control.
	ClearUndo (see page 21246)	Clears the undo buffer so that no changes to the text can be backed out.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopyToClipboard (see page 21246)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 12520)	Creates an instance of TCustomLabeledEdit (see page 12508).
V	Create (see page 21244)	Creates an instance of TCustomEdit (see page 21232).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.

	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CutToClipboard (see page 21249)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 21250)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
V	GetSelTextBuf (see page 21253)	Copies the selected text into a buffer and returns the number of characters copied.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.

	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	PasteFromClipboard (see page 21254)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectAll (see page 21255)	Selects all text in the edit control.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.

V	SetBounds (see page 12522)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSelTextBuf (see page 21258)	Replaces the selected text with a null-terminated string.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	SetupInternalLabel (see page 12523)	Initializes the control that implements the EditLabel (see page 12524) property.
	Show (see page 5788)	Makes a control visible.
	Undo (see page 21259)	Backs out all changes in the undo buffer.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	CanUndo (see page 21262)	Indicates whether the edit control contains changes that can be backed out.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.

R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Modified (see page 21263)	Indicates whether the user edited the text of the edit control.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelLength (see page 21265)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 21269)	Specifies the position of the first selected character in the text.
	SelText (see page 21272)	Specifies the selected portion of the edit control's text.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 12873)	Occurs when the text for the edit control may have changed.
	OnClick (see page 12874)	Occurs when the user clicks the control.
	OnContextPopup (see page 12876)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 12876)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 12877)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 12879)	Occurs when the user drags an object over a control.
	OnEndDock (see page 12881)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 12882)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 12883)	Occurs when a control receives the input focus.
	OnExit (see page 12885)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 12887)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 12890)	Occurs when a key is pressed.
	OnKeyUp (see page 12891)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 12892)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 12893)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 12896)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 12896)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 12896)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 12898)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 12901)	Occurs when the user begins to drag a control with a DragKind of dkDock.

	OnStartDrag (see page 12901)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 12846)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 12846)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 12847)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 12848)	Specifies which edges of the control are beveled.
	BevelInner (see page 12848)	Specifies the cut of the inner bevel.
	BevelKind (see page 12849)	Specifies the control's bevel style.
	BevelOuter (see page 12849)	Specifies the cut of the outer bevel.
	BiDiMode (see page 12849)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 12850)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 12850)	Determines the case of the text within the edit control.
	Color (see page 12851)	Specifies the background color of the control.
	Constraints (see page 12851)	Specifies the size constraints for the control.
	Ctl3D (see page 12852)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 12853)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 12853)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 12853)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EditLabel (see page 12855)	Specifies the control that implements the label.
	Enabled (see page 12855)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 12856)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	HideSelection (see page 12857)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 12857)	Determines the behavior of the input method editor (IME).
	ImeName (see page 12858)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	LabelPosition (see page 12858)	Specifies the position of the label relative to the edit control.
	LabelSpacing (see page 12859)	Specifies the distance, in pixels, between the label and the edit region.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	MaxLength (see page 12859)	Specifies the maximum number of characters the user can enter into the edit control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OEMConvert (see page 12860)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBiDiMode (see page 12860)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 12860)	Determines where a control looks for its color information.
	ParentCtl3D (see page 12861)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 12861)	Determines where a control looks for its font information.

	ParentShowHint (see page 12862)	Determines where a control looks to find out if its Help Hint should be shown.
	PasswordChar (see page 12862)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 12863)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 12867)	Determines whether the user can change the text of the edit control.
	ShowHint (see page 12868)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 12869)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 12870)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Text (see page 12870)	Contains a text string associated with the control.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 12871)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.26.2 TLabeledEdit Properties

The properties of the TLabeledEdit class are listed here.

Published Properties

	Name	Description
	Anchors (see page 12846)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 12846)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 12847)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 12848)	Specifies which edges of the control are beveled.
	BevelInner (see page 12848)	Specifies the cut of the inner bevel.
	BevelKind (see page 12849)	Specifies the control's bevel style.
	BevelOuter (see page 12849)	Specifies the cut of the outer bevel.
	BiDiMode (see page 12849)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 12850)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 12850)	Determines the case of the text within the edit control.
	Color (see page 12851)	Specifies the background color of the control.
	Constraints (see page 12851)	Specifies the size constraints for the control.
	Ctl3D (see page 12852)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 12853)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 12853)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 12853)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EditLabel (see page 12855)	Specifies the control that implements the label.
	Enabled (see page 12855)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 12856)	Controls the attributes of text written on or in the control.
	HideSelection (see page 12857)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 12857)	Determines the behavior of the input method editor (IME).
	ImeName (see page 12858)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	LabelPosition (see page 12858)	Specifies the position of the label relative to the edit control.
	LabelSpacing (see page 12859)	Specifies the distance, in pixels, between the label and the edit region.
	MaxLength (see page 12859)	Specifies the maximum number of characters the user can enter into the edit control.
	OEMConvert (see page 12860)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.

	ParentBiDiMode (see page 12860)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 12860)	Determines where a control looks for its color information.
	ParentCtl3D (see page 12861)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 12861)	Determines where a control looks for its font information.
	ParentShowHint (see page 12862)	Determines where a control looks to find out if its Help Hint should be shown.
	PasswordChar (see page 12862)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 12863)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 12867)	Determines whether the user can change the text of the edit control.
	ShowHint (see page 12868)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 12869)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 12870)	Determines if the user can tab to a control.
	Text (see page 12870)	Contains a text string associated with the control.
	Visible (see page 12871)	Determines whether the component appears onscreen.

68.1.26.2.1 TLabeledEdit.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

68.1.26.2.2 TLabeledEdit.AutoSelect Property

Determines whether all the text in the edit control is automatically selected when the control gets focus.

Pascal

```
property AutoSelect: Boolean;
```

C++

```
__property Boolean AutoSelect;
```

Description

Set AutoSelect to select all the text when the edit control gets focus. AutoSelect only applies to single-line edit controls.

Use AutoSelect when the user is more likely to replace the text in the edit control than to append to it.

See Also

SelectAll (see page 21255)

SelLength (see page 21265)

SelStart (see page 21269)

SelText (see page 21272)

SetSelTextBuf (see page 21258)

68.1.26.2.3 TLabeledEdit.AutoSize Property

Determines whether the height of the edit control automatically resizes to accommodate the text.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;__property Boolean AutoSize;
```

Description

Use AutoSize to make the edit control adjust its size automatically so the client area accommodates the height of the text. When AutoSize is false, the edit control has a fixed height. When AutoSize is true, the size of the control is readjusted whenever a change occurs that could affect the height of the control, such as a change to the font or border style.

Note: AutoSize only adjusts the height of the edit control. Even with AutoSize set to true, text in the edit control may appear truncated if it extends beyond the width of the control.

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit1->AutoSize = true;
  Edit1->Font->Size = 20;
  Edit1->Text = "The edit box is taller now";
}
```

Delphi Examples:

```
{
{
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  Edit1.AutoSize := True;
  Edit1.Font.Size := 20;
  Edit1.Text := 'The edit box is taller now';
end;
```

See Also

BorderStyle (see page 21261)

ClientHeight (see page 5808)

ControlStyle (see page 5814)

Height (see page 5825)

68.1.26.2.4 TLabeledEdit.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

68.1.26.2.5 TLabeledEdit.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

68.1.26.2.6 TLabeledEdit.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

68.1.26.2.7 TLabeledEdit.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

68.1.26.2.8 TLabeledEdit.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

68.1.26.2.9 TLabeledEdit.BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the edit control stands out. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the edit control has a single-line border around the client area. If BorderStyle is bsNone, there will be no border.

68.1.26.2.10 TLabeledEdit.CharCase Property

Determines the case of the text within the edit control.

Pascal

```
property CharCase: TEditCharCase;
```

C++

```
__property TEditCharCase CharCase;
```

Description

Use CharCase to force the contents of the edit control to assume a particular case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types them into the edit control. Changing the CharCase property to ecLowerCase or ecUpperCase changes the actual contents of the text, not just the appearance. Any case information is lost and can't be recaptured by changing CharCase to ecNormal.

See Also

Text (see page 5842)

68.1.26.2.11 TLabeledEdit.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

68.1.26.2.12 TLabeledEdit.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

68.1.26.2.13 TLabeledEdit.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

68.1.26.2.14 TLabeledEdit.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

68.1.26.2.15 TLabeledEdit.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

68.1.26.2.16 TLabeledEdit.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

68.1.26.2.17 TLabeledEdit.EditLabel Property

Specifies the control that implements the label.

Pascal

```
property EditLabel: TBoundLabel;
```

C++

```
__property TBoundLabel EditLabel;
```

Description

Use EditLabel to work with the label that is associated with this labeled edit control. Use this label's properties to specify the caption that appears on the label, the position of the caption within the label, the font used to write that caption, and so on. At design time, expand the EditLabel property in the Object Inspector to set its properties.

See Also

LabelPosition (see page 12524)

LabelSpacing (see page 12525)

68.1.26.2.18 TLabeledEdit.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

68.1.26.2.19 TLabeledEdit.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

68.1.26.2.20 TLabeledEdit.HideSelection Property

Determines whether the visual indication of the selected text remains when focus shifts to another control.

Pascal

```
property HideSelection: Boolean;
```

C++

```
__property Boolean HideSelection;
```

Description

Set HideSelection to false to provide visual feedback of the selected portion of the text even when the edit control does not have focus. Set HideSelection to true to show the selection only when the edit control has focus. HideSelection does not affect the actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms with many edit controls look too busy.

See Also

SelLength (see page 21265)

SelStart (see page 21269)

SelText (see page 21272)

SetSelTextBuf (see page 21258)

68.1.26.2.21 TLabeledEdit.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

68.1.26.2.22 TLabeledEdit.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

68.1.26.2.23 TLabeledEdit.LabelPosition Property

Specifies the position of the label relative to the edit control.

Pascal

```
property LabelPosition: TLabelPosition;
```

C++

```
__property TLabelPosition LabelPosition;
```

Description

Set LabelPosition to indicate where the label sits relative to the edit control.

See Also

EditLabel (see page 12524)

LabelSpacing (see page 12525)

68.1.26.2.24 TLabeledEdit.LabelSpacing Property

Specifies the distance, in pixels, between the label and the edit region.

Pascal

```
property LabelSpacing: Integer;
```

C++

```
__property int LabelSpacing;
```

Description

Set LabelSpacing to indicate the distance, in pixels, between the label and the edit control.

See Also

EditLabel (see page 12524)

LabelPosition (see page 12524)

68.1.26.2.25 TLabeledEdit.MaxLength Property

Specifies the maximum number of characters the user can enter into the edit control.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the edit control. A value of 0 indicates that there is no application-defined limit on the length.

Use MaxLength to limit the length of the text in an edit control if that text will be copied into a fixed-length buffer.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

Note: The unit of MaxLength is characters (not bytes) if the application is using themes.

See Also

Text (see page 5842)

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

68.1.26.2.26 TLabeledEdit.OEMConvert Property

Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.

Pascal

```
property OEMConvert: Boolean;
```

C++

```
__property Boolean OEMConvert;
```

Description

To ensure that any characters typed in the edit control can be unambiguously converted to the OEM character set, set OEMConvert to true. This causes all characters to be converted from ANSI to OEM and then back to ANSI as they are typed. Set OEMConvert to false to avoid the overhead of this conversion when it does not matter whether the text can be unambiguously mapped to an OEM string.

OEMConvert is most useful for edit controls used to enter file names when the application does not use Unicode file names.

See Also

Text (see page 5842)

68.1.26.2.27 TLabeledEdit.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

68.1.26.2.28 TLabeledEdit.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

68.1.26.2.29 TLabeledEdit.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

68.1.26.2.30 TLabeledEdit.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

68.1.26.2.31 TLabeledEdit.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

68.1.26.2.32 TLabeledEdit.PasswordChar Property

Indicates the character, if any, to display in place of the actual characters typed in the control.

Pascal

```
property PasswordChar: Char;
```

C++

```
__property Char PasswordChar;
```

Description

Use the PasswordChar property to create an edit control that displays a special character in place of any entered text. If PasswordChar is set to the null character (ANSI character zero), the edit control displays its text normally. If PasswordChar is any other character, the edit control displays PasswordChar in place of each character typed. PasswordChar affects the appearance of the edit control only. The value of the Text property reflects the actual characters that are typed.

See Also

Text (see page 5842)

68.1.26.2.33 TLabeledEdit.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
```

```
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
```

```
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
```

```
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

68.1.26.2.34 TLabeledEdit.ReadOnly Property

Determines whether the user can change the text of the edit control.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

Setting ReadOnly to true ensures that the text is not altered, while still allowing the user to select text. The selected text can then be manipulated by the application, or copied to the Clipboard.

See Also

CopyToClipboard (see page 21246)

GetSelTextBuf (see page 21253)

68.1.26.2.35 TLabeledEdit.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

68.1.26.2.36 TLabeledEdit.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

68.1.26.2.37 TLabeledEdit.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

68.1.26.2.38 TLabeledEdit.Text Property

Contains a text string associated with the control.

Pascal

```
property Text: TCaption;
```

C++

```
__property TCaption Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
  Size++;                         //Add room for null character
```

```
  char *Buffer = new char[Size];  //Creates Buffer dynamic variable
  Edit1->GetTextBuf(Buffer,Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer;
  delete [] Buffer;
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
  Inc(Size);                      {Add room for null character}
  GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
  Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
end;
```

See Also

Caption (see page 5806)

Lines (see page 21380)

68.1.26.2.39 TLabeledEdit.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
```

```
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

68.1.26.3 TLabeledEdit Events

The events of the TLabeledEdit class are listed here.

Published Events

	Name	Description
	OnChange (see page 12873)	Occurs when the text for the edit control may have changed.
	OnClick (see page 12874)	Occurs when the user clicks the control.
	OnContextPopup (see page 12876)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 12876)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 12877)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 12879)	Occurs when the user drags an object over a control.
	OnEndDock (see page 12881)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 12882)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 12883)	Occurs when a control receives the input focus.
	OnExit (see page 12885)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 12887)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 12890)	Occurs when a key is pressed.
	OnKeyUp (see page 12891)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 12892)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 12893)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 12896)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 12896)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 12896)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 12898)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 12901)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 12901)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

68.1.26.3.1 TLabeledEdit.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change (see page 21245)

Text (see page 5842)

68.1.26.3.2 TLabeledEdit.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
```

```
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

68.1.26.3.3 TLabeledEdit.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

68.1.26.3.4 TLabeledEdit.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

68.1.26.3.5 TLabeledEdit.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
```

```
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

68.1.26.3.6 TLabeledEdit.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
```

```
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.26.3.7 TLabeledEdit.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.26.3.8 TLabeledEdit.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

68.1.26.3.9 TLabeledEdit.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
```

```
    ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
```

```
    else if myActiveControl is TRichEdit then
      with myActiveControl as TRichEdit do
        SelAttributes.Assign(TFontDialog(Sender).Font)
    else
      Beep;
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

68.1.26.3.10 TLabeledEdit.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
```

```
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

68.1.26.3.11 TLabeledEdit.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
```

```
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

```
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

68.1.26.3.12 TLabeledEdit.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

68.1.26.3.13 TLabeledEdit.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

68.1.26.3.14 TLabeledEdit.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.26.3.15 TLabeledEdit.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
```

```
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.26.3.16 TLabeledEdit.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

68.1.26.3.17 TLabeledEdit.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

68.1.26.3.18 TLabeledEdit.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
```

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.26.3.19 TLabeledEdit.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

68.1.26.3.20 TLabeledEdit.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

68.1.26.3.21 TLabeledEdit.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

68.1.27 TNotebook Class

TNotebook displays multiple pages, each with its own set of controls.

Class Hierarchy

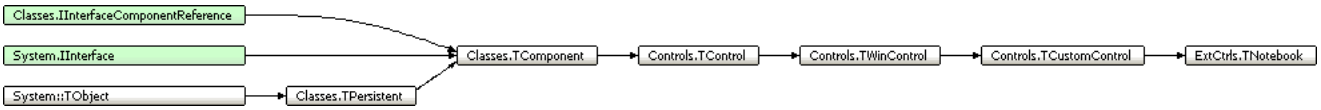

Pascal

```
TNotebook = class(TCustomControl);
```

C++

```
class TNotebook : public TCustomControl;
```

File

ExtCtrls

Description

Notebook components are frequently used with tab set controls (TTabSet) to let the user select pages in the notebook by clicking a tab. TNotebook is provided for backward compatibility. New applications should use TPageControl instead.

68.1.27.1 TNotebook Members

The following tables list the members exposed by TNotebook.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.

	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.

	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 12913)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.

	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 12914)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 12914)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 5920)	Renders the image of a custom control.

	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 12915)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 12915)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.

	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 12912)	Constructs a TNotebook (see page 12902) component.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 12913)	Destroys the TNotebook (see page 12902) component.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.

R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 12938)	Occurs when the user clicks the control.
	OnContextPopup (see page 12940)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 12941)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 12941)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 12943)	Occurs when the user drags an object over a control.
	OnEndDock (see page 12946)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 12946)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 12947)	Occurs when a control receives the input focus.
	OnExit (see page 12950)	Occurs when the input focus shifts away from one control to another.
	OnMouseActivate (see page 12951)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 12952)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 12954)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 12955)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 12955)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 12957)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPageChanged (see page 12959)	Occurs when the user selects a new page in the notebook.

	OnStartDock (see page 12960)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 12960)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	ActivePage (see page 12916)	Determines which page displays in the notebook.
	Align (see page 12917)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 12920)	Specifies how the control is anchored to its parent.
	Color (see page 12921)	Specifies the background color of the control.
	Constraints (see page 12921)	Specifies the size constraints for the control.
	Ctl3D (see page 12922)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 12923)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 12923)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 12923)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 12925)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 12925)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	PageIndex (see page 12926)	Determines which page displays in the notebook control.
	Pages (see page 12927)	Contains the strings that identify the individual pages of the notebook control.
	ParentColor (see page 12927)	Determines where a control looks for its color information.
	ParentCtl3D (see page 12928)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 12928)	Determines where a control looks for its font information.
	ParentShowHint (see page 12928)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 12929)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 12934)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 12935)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 12936)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 12936)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.27.2 TNotebook.Create Constructor

Constructs a TNotebook (see page 12902) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TNotebook(TComponent * AOwner);
```

Description

Call Create to instantiate a notebook at runtime. Notebook components placed in forms at design time are created automatically.

The AOwner parameter specifies the component (usually the form) that is responsible for freeing the notebook.

68.1.27.3 TNotebook.Destroy Destructor

Destroys the TNotebook (see page 12902) component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TNotebook();
```

Description

Do not call Destroy directly in an application. Instead, use Free, which checks that the TNotebook (see page 12902) reference is not nil before calling Destroy.

See Also

Free (see page 22300)

68.1.27.4 TNotebook Methods

The methods of the TNotebook class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 12913)	Initializes a window-creation parameter data structure.
	GetChildOwner (see page 12914)	Returns the owner of a child component being read from a stream.
	GetChildren (see page 12914)	Calls a specified method for each child of the control.
	ReadState (see page 12915)	Prepares the control for having its properties assigned values from a stream.
	ShowControl (see page 12915)	Ensures that a specified child control is visible.

68.1.27.4.1 TNotebook.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override

CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

68.1.27.4.2 TNotebook.GetChildOwner Method

Returns the owner of a child component being read from a stream.

Pascal

```
function GetChildOwner: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetChildOwner();
```

Description

GetChildOwner is used internally by the component streaming system. It is rarely necessary to call it directly.

In TComponent, GetChildOwner always returns nil (Delphi) or NULL (C++), indicating that the owner is the root component currently being read (usually a form or data module). The Owner of a component is responsible for destroying it.

See Also

Create (see page 2766)

Owner (see page 2807)

68.1.27.4.3 TNotebook.GetChildren Method

Calls a specified method for each child of the control.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the streaming system that loads and saves components. Applications seldom need to call this routine.

GetChildren executes the callback specified by the Proc parameter for every child control listed by the Controls property that lists the Root parameter as its Owner.

Override GetChildren to limit or augment which child controls are saved with the control. When overriding GetChildren, call the procedure passed as the Proc parameter for every child control that should be saved. The Root parameter indicates the component (usually a form) that owns the control.

See Also

Controls (see page 6197)

Owner (see page 2807)

68.1.27.4.4 TNotebook.ReadState Method

Prepares the control for having its properties assigned values from a stream.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

Override ReadState to change the preparations the control makes as it readies itself to have its property values assigned from a stream. For example, a control might destroy temporary internal data structures or objects before new instances are loaded from the stream. Be sure to include a call to the inherited method when overriding ReadState.

As implemented in TWinControl, ReadState disables control alignment until all the child controls have been read from the stream, calls the inherited ReadState method, and re-enables the alignment of all the controls. The method then builds the tab order list of controls and synchronizes its visibility with that of its parent control.

See Also

Parent (see page 5832)

ReadState (see page 5778)

68.1.27.4.5 TNotebook.ShowControl Method

Ensures that a specified child control is visible.

Pascal

```
procedure ShowControl(AControl: TControl); override;
```

C++

```
virtual __fastcall ShowControl(TControl * AControl);
```

Description

Call ShowControl to display a child control. Specify the control you want to ensure will be visible as the value of the AControl parameter.

As implemented in TWinControl, ShowControl simply calls the ShowControl method of the control's parent. Derived objects override ShowControl to perform whatever actions are necessary to allow the indicated child object to be visible. Possible actions include scrolling the particular child object into view, or changing to the appropriate notebook page.

See Also

Show (see page 5788)

Parent (see page 5832)

68.1.27.5 TNotebook Properties

The properties of the TNotebook class are listed here.

Published Properties

	Name	Description
	ActivePage (see page 12916)	Determines which page displays in the notebook.
	Align (see page 12917)	Determines how the control aligns within its container (parent control).
	Anchors (see page 12920)	Specifies how the control is anchored to its parent.
	Color (see page 12921)	Specifies the background color of the control.
	Constraints (see page 12921)	Specifies the size constraints for the control.
	Ctl3D (see page 12922)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 12923)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 12923)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 12923)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 12925)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 12925)	Controls the attributes of text written on or in the control.
	PageIndex (see page 12926)	Determines which page displays in the notebook control.
	Pages (see page 12927)	Contains the strings that identify the individual pages of the notebook control.
	ParentColor (see page 12927)	Determines where a control looks for its color information.
	ParentCtl3D (see page 12928)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 12928)	Determines where a control looks for its font information.
	ParentShowHint (see page 12928)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 12929)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 12934)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 12935)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 12936)	Determines if the user can tab to a control.
	Visible (see page 12936)	Determines whether the component appears onscreen.

68.1.27.5.1 TNotebook.ActivePage Property

Determines which page displays in the notebook.

Pascal

```
property ActivePage: string;
```

C++

```
__property AnsiString ActivePage;
```

Description

ActivePage determines which page displays in the notebook. The value of ActivePage must be one of the strings contained in the Pages (see page 12927) property.

To get or set the active page by index rather than string, use the PageIndex (see page 12926) property instead.

See Also

PageIndex (see page 12926)

Pages (see page 12927)

68.1.27.5.2 TNotebook.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
```

```
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
```

```
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
```

```
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

68.1.27.5.3 TNotebook.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

68.1.27.5.4 TNotebook.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

68.1.27.5.5 TNotebook.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

68.1.27.5.6 TNotebook.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

68.1.27.5.7 TNotebook.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

68.1.27.5.8 TNotebook.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

68.1.27.5.9 TNotebook.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

68.1.27.5.10 TNotebook.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

68.1.27.5.11 TNotebook.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

68.1.27.5.12 TNotebook.PageIndex Property

Determines which page displays in the notebook control.

Pascal

```
property PageIndex: Integer;
```

C++

```
__property int PageIndex;
```

Description

The value of PageIndex determines which page displays in the notebook. Changing the PageIndex value changes the page in the control.

Each string in the Pages (see page 12927) property is automatically assigned a PageIndex value when the page is created. The first page receives a value of 0, the second has a value of 1, and so on. If you delete a string from the Pages (see page 12927) property, the PageIndex values are reassigned so that the values always begin with 0 and continue to increase without any gaps between values.

To get or set the current page using its string rather than its index, use the ActivePage (see page 12916) property instead.

See Also

ActivePage (see page 12916)

Pages (see page 12927)

68.1.27.5.13 TNotebook.Pages Property

Contains the strings that identify the individual pages of the notebook control.

Pascal

```
property Pages: TStrings;
```

C++

```
__property TStrings Pages;
```

Description

The Pages property contains the strings that identify the individual pages of the notebook control. Both these controls create a separate page for each string in the Pages property. For example, if Pages contains three strings, First, Second, and Third, the control has three separate pages.

You can access the various pages in a notebook or tabbed notebook control with either the ActivePage (see page 12916) or PageIndex (see page 12926) property.

See Also

ActivePage (see page 12916)

PageIndex (see page 12926)

68.1.27.5.14 TNotebook.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

68.1.27.5.15 TNotebook.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

68.1.27.5.16 TNotebook.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

68.1.27.5.17 TNotebook.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

68.1.27.5.18 TNotebook.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
```

```
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
```

```
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
```

```
    Edit1.PopupMenu := PopupMenu1;
    Edit2.PopupMenu := PopupMenu1;
  end;

  {
  These are the cut, copy, and paste OnClick events for the commands
  on the pop-up menu. Double click on the pop-up menu and add three
  TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
  The names will be automatically set to "Cut1", "Copy1", and
  "Paste1".  Then double click on the OnClick events and they will
  be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
  Note: only the selected portion of the string in the edit box will
  be copied, cut or replaced.
  }
  procedure TForm1.Copy1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
     Edit1.CopyToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
```

```
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

68.1.27.5.19 TNotebook.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
```

```
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

68.1.27.5.20 TNotebook.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

68.1.27.5.21 TNotebook.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

68.1.27.5.22 TNotebook.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
```

```
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

68.1.27.6 TNotebook Events

The events of the TNotebook class are listed here.

Published Events

	Name	Description
	OnClick (see page 12938)	Occurs when the user clicks the control.
	OnContextPopup (see page 12940)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 12941)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 12941)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 12943)	Occurs when the user drags an object over a control.
	OnEndDock (see page 12946)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 12946)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 12947)	Occurs when a control receives the input focus.
	OnExit (see page 12950)	Occurs when the input focus shifts away from one control to another.
	OnMouseActivate (see page 12951)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 12952)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 12954)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 12955)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 12955)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 12957)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPageChanged (see page 12959)	Occurs when the user selects a new page in the notebook.
	OnStartDock (see page 12960)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 12960)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

68.1.27.6.1 TNotebook.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.

- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

68.1.27.6.2 TNotebook.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple

times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

68.1.27.6.3 TNotebook.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

68.1.27.6.4 TNotebook.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
```

```
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
```

```
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

68.1.27.6.5 TNotebook.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor

property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
```

```
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.27.6.6 TNotebook.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.27.6.7 TNotebook.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
```

```
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

68.1.27.6.8 TNotebook.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter

event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
```

```
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

68.1.27.6.9 TNotebook.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

68.1.27.6.10 TNotebook.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.27.6.11 TNotebook.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
```

```
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
```

```
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.27.6.12 TNotebook.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

68.1.27.6.13 TNotebook.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

68.1.27.6.14 TNotebook.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;
```

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.27.6.15 TNotebook.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
```

```
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
```

```
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

68.1.27.6.16 TNotebook.OnPageChanged Event

Occurs when the user selects a new page in the notebook.

Pascal

```
property OnPageChanged: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPageChanged;
```

Description

Use the OnPageChanged event when you want special processing to occur when the active page changes.

68.1.27.6.17 TNotebook.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

68.1.27.6.18 TNotebook.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

68.1.28 TPage Class

Class Hierarchy

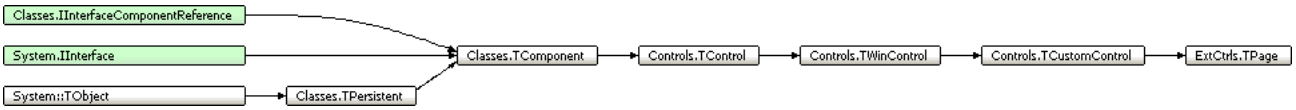

Pascal

```
TPage = class(TCustomControl);
```

C++

```
class TPage : public TCustomControl;
```

File

ExtCtrls

Description

This is class ExtCtrls.TPage.

68.1.28.1 TPage Members

The following tables list the members exposed by TPage.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.

	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.

	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.

	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 12972)	Renders the image of a custom control.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 12973)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.

	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.

	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 12971)	Creates an instance of TCustomControl.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.

	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.

	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 12980)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 12980)	Occurs when an object with custom alignment is aligned.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Caption (see page 12973)	Specifies a text string that identifies the control to the user.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 12976)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	TabOrder (see page 12977)	Indicates the position of the control in its parent's tab order.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 12978)	Determines whether the component appears onscreen.
	Width (see page 12979)	Specifies the horizontal size of the control or form in pixels.

68.1.28.2 TPage.Create Constructor

Creates an instance of TCustomControl.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TPage(TComponent * AOwner);
```

Description

Calling Create constructs and initializes an instance of TCustomControl. However, you should never attempt to instantiate a TCustomControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants. Controls placed on forms at design time are created automatically.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes a TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create (see page 6123)

Destroy (see page 5919)

Canvas (see page 5921)

Owner (see page 2807)

68.1.28.3 TPage Methods

The methods of the TPage class are listed here.

Protected Methods

	Name	Description
	Paint (see page 12972)	Renders the image of a custom control.
	ReadState (see page 12973)	Prepares the control for having its properties assigned values from a stream.

68.1.28.3.1 TPage.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas (see page 5921)

PaintWindow (see page 5920)

GetDeviceContext (see page 6155)

ClipRect (see page 14486)

68.1.28.3.2 TPage.ReadState Method

Prepares the control for having its properties assigned values from a stream.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

Override ReadState to change the preparations the control makes as it readies itself to have its property values assigned from a stream. For example, a control might destroy temporary internal data structures or objects before new instances are loaded from the stream. Be sure to include a call to the inherited method when overriding ReadState.

As implemented in TWinControl, ReadState disables control alignment until all the child controls have been read from the stream, calls the inherited ReadState method, and re-enables the alignment of all the controls. The method then builds the tab order list of controls and synchronizes its visibility with that of its parent control.

See Also

Parent (see page 5832)

ReadState (see page 5778)

68.1.28.4 TPage Properties

The properties of the TPage class are listed here.

Published Properties

	Name	Description
	Caption (see page 12973)	Specifies a text string that identifies the control to the user.
	Height (see page 12976)	Specifies the vertical size of the control in pixels.
	TabOrder (see page 12977)	Indicates the position of the control in its parent's tab order.
	Visible (see page 12978)	Determines whether the component appears onscreen.
	Width (see page 12979)	Specifies the horizontal size of the control or form in pixels.

68.1.28.4.1 TPage.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
```

```
          Parent := ToolBar;
          Caption := ButtonCaptions[I];
          if (ButtonCaptions[I] = '|') then
            Style := tbsSeparator
          else
            Style := tbsButton;
        end;
      end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ToolBar1 := TToolBar.Create(Self);
    ToolBar1.Parent := Self;
    AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
    ToolBar1.ShowCaptions := True;
    ToolBar1.Height := 40;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I: Integer;
  begin
    for I := ToolBar1.ButtonCount - 1 downto 0 do
      ToolBar1.Buttons[I].Free;
    ToolBar1.Free;
  end;
```

68.1.28.4.2 TPage.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
```

```
  }
}
```

Delphi Examples:

```
{
HideSelection, Height example
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
        HideSelection := False;
    end;
end;
```

See Also

ClientHeight (see page 5808)

Width (see page 5849)

Using TDBListBox and TDBComboBox

68.1.28.4.3 TPage.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of

the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

68.1.28.4.4 TPage.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

68.1.28.4.5 TPage.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

See Also

ClientWidth (see page 5811)

Height (see page 5825)

Left (see page 5830)

68.1.28.5 TPage Events

The events of the TPage class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 12980)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 12980)	Occurs when an object with custom alignment is aligned.

68.1.28.5.1 TPage.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore: TAlignInsertBeforeEvent;
```

C++

```
__property TAlignInsertBeforeEvent OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore (see page 6137)

Align (see page 5798)

TAlignInsertBeforeEvent (see page 6293)

68.1.28.5.2 TPage.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition: TAlignPositionEvent;
```

C++

```
__property TAlignPositionEvent OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition (see page 6138)

Align (see page 5798)

Anchors (see page 5802)

TAlignInfo (see page 6269)

TAlignPositionEvent (see page 6293)

68.1.29 TPaintBox Class

TPaintBox provides a canvas that applications can use for rendering an image.

Class Hierarchy

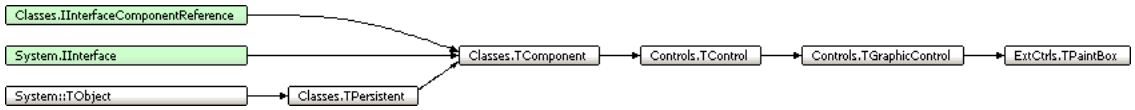

Pascal

```
TPaintBox = class(TGraphicControl);
```

C++

```
class TPaintBox : public TGraphicControl;
```

File

ExtCtrls

Description

Use TPaintBox to add custom images to a form. Unlike TImage (see page 12777), which displays an image that is stored in a bitmap, icon, or metafile, TPaintBox requires an application to draw the image directly on a canvas. Use the OnPaint (see page 13028) event handler to draw on the paint box's Canvas (see page 12994), the drawing surface of the paint box.

See Also

TCanvas (see page 14441)

TImage (see page 12777)

Graphic Controls

Paint Boxes

68.1.29.1 TPaintBox Members

The following tables list the members exposed by TPaintBox.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.

	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.

	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Paint (see page 12989)	Generates an OnPaint (see page 13028) event.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.

	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 12989)	Creates an instance of TPaintBox (see page 12981).
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.

	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Canvas (see page 12994)	Provides the drawing surface used by the graphic control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnClick (see page 13011)	Occurs when the user clicks the control.
	OnContextPopup (see page 13013)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 13014)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 13014)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13016)	Occurs when the user drags an object over a control.
	OnEndDock (see page 13019)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 13019)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 13020)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 13021)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 13023)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 13024)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 13024)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 13026)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 13028)	OnPaint occurs when the paintbox receives a Windows paint message.
	OnStartDock (see page 13030)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 13031)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 12990)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 12994)	Specifies how the control is anchored to its parent.
	Color (see page 12996)	Specifies the background color of the control.
	Constraints (see page 12997)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 12998)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 12998)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 12998)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 13000)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 13000)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentColor (see page 13001)	Determines where a control looks for its color information.
	ParentFont (see page 13002)	Determines where a control looks for its font information.
	ParentShowHint (see page 13002)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 13003)	Identifies the pop-up menu associated with the control.

	ShowHint (see page 13008)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 13009)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.29.2 TPaintBox.Create Constructor

Creates an instance of TPaintBox (see page 12981).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TPaintBox(TComponent * AOwner);
```

Description

Call Create to instantiate a TPaintBox (see page 12981) object at runtime. TPaintBox (see page 12981) objects added to forms at design time are created automatically.

Call Create to instantiate a TImage (see page 12777) object at runtime. Images placed on forms at design time are created automatically.

AOwner is the component that is responsible for freeing the paintbox instance. It becomes the value of the Owner property.

See Also

Owner (see page 2807)

68.1.29.3 TPaintBox Methods

The methods of the TPaintBox class are listed here.

Protected Methods

	Name	Description
	Paint (see page 12989)	Generates an OnPaint (see page 13028) event.

68.1.29.3.1 TPaintBox.Paint Method

Generates an OnPaint (see page 13028) event.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Paint responds to WM_PAINT messages from Windows by initializing the canvas and calling the OnPaint (see page 13028) event handler if it is assigned.

See Also

OnPaint (see page 13028)

68.1.29.4 TPaintBox Properties

The properties of the TPaintBox class are listed here.

Public Properties

	Name	Description
	Canvas (see page 12994)	Provides the drawing surface used by the graphic control.

Published Properties

	Name	Description
	Align (see page 12990)	Determines how the control aligns within its container (parent control).
	Anchors (see page 12994)	Specifies how the control is anchored to its parent.
	Color (see page 12996)	Specifies the background color of the control.
	Constraints (see page 12997)	Specifies the size constraints for the control.
	DragCursor (see page 12998)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 12998)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 12998)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 13000)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 13000)	Controls the attributes of text written on or in the control.
	ParentColor (see page 13001)	Determines where a control looks for its color information.
	ParentFont (see page 13002)	Determines where a control looks for its font information.
	ParentShowHint (see page 13002)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 13003)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 13008)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Visible (see page 13009)	Determines whether the component appears onscreen.

68.1.29.4.1 TPaintBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
```

```
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

```
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

68.1.29.4.2 TPaintBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

68.1.29.4.3 TPaintBox.Canvas Property

Provides the drawing surface used by the graphic control.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control.

Since a TGraphicControl instance does not have its own Windows screen object, it obtains its device context from its parent control.

C++ Examples:

```
/*
When you do custom drawing, such as shown in the code for
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
   PaintBox1->Canvas->Brush->Color = clRed;
   PaintBox1->Canvas->FillRect(PaintBox1->Canvas->ClipRect);
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}

/*
In order for the drawing to be persistent, you need to also
include an OnPaint event handler that tells the PaintBox how
to redraw itself when it needs to. In the code below, an
ellipse will be drawn everytime the PaintBox renders itself,
but the PaintBox will only paint the client area of the
control red after the Button is clicked. This client area
will only stay red until it is invalidated, but the ellipse
will persist. This behavior is different from the TImage
component, because the TImage component maintains an
internal bitmap that stores this drawing information for
you.
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
   PaintBox1->Canvas->Brush->Color = clBlue;
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TLabel *myLabel;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TLabel")
  {
    myLabel = (TLabel *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myLabel->Canvas->MoveTo(myLabel->ClientRect.Left, myLabel->ClientRect.Top);
    myLabel->Canvas->LineTo(myLabel->ClientRect.Right, myLabel->ClientRect.Bottom);
    myLabel = NULL;
  }
}
/*
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PaintBox1->Canvas->Draw(0,0, Image1->Picture->Graphic);
}
```

Delphi Examples:

```
{
When you do custom drawing, such as shown in the code for
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   PaintBox1.Canvas.Brush.Color := clRed;
   PaintBox1.Canvas.FillRect(PaintBox1.Canvas.ClipRect);
   PaintBox1.Canvas.Brush.Color := clBlue;
```

```
    PaintBox1.Canvas.Ellipse(0,0,100,50);
  end;
  {
  In order for the drawing to be persistent, you need to also
  include an OnPaint event handler that tells the PaintBox how
  to redraw itself when it needs to. In the code below, an
  ellipse will be drawn everytime the PaintBox renders itself,
  but the PaintBox will only paint the client area of the
  control red after the Button is clicked. This client area
  will only stay red until it is invalidated, but the ellipse
  will persist. This behavior is different from the TImage
  component, because the TImage component maintains an
  internal bitmap that stores this drawing information for
  you.
  }
  procedure TForm1.PaintBox1Paint(Sender: TObject);
  begin
    PaintBox1.Canvas.Brush.Color := clBlue;
    PaintBox1.Canvas.Ellipse(0,0,100,50);
  end;
  {
  The following code uses ClientRect to find and draw a line
  from the top left to the bottom right of the current
  control.  Select a TControl in the ListBox list.  This
  example assumes that the current object is a descendent of
  TControl and has a public Canvas property.
  }
    if (ListBox1.Items.Objects[Index] is TLabel) then
    begin
      myLabel:= (ListBox1.Items.Objects[Index] as TLabel);
      with myLabel.ClientRect do
      begin // make sure we are using the TControl's canvas and not the form's!
        myLabel.Canvas.MoveTo(Left, Top);
        myLabel.Canvas.LineTo(Right, Bottom);
      end;
    myLabel:= nil;
    end;
  {
  The following code draws the graphic in Picture1 in the
  top-left corner PaintBox1.  Place a TPaintBox in the form
  and a TImage in the form that contains a picture.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    PaintBox1.Canvas.Draw(0,0, Image1.Picture.Graphic);
  end;
```

68.1.29.4.4 TPaintBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

68.1.29.4.5 TPaintBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

68.1.29.4.6 TPaintBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

68.1.29.4.7 TPaintBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

68.1.29.4.8 TPaintBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
```

```
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

68.1.29.4.9 TPaintBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

68.1.29.4.10 TPaintBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

68.1.29.4.11 TPaintBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

68.1.29.4.12 TPaintBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

68.1.29.4.13 TPaintBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

68.1.29.4.14 TPaintBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
```

```
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
```

```
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;
```

```
{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);
```

```
    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

68.1.29.4.15 TPaintBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

68.1.29.4.16 TPaintBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
```

```
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

68.1.29.5 TPaintBox Events

The events of the TPaintBox class are listed here.

Published Events

	Name	Description
	OnClick (see page 13011)	Occurs when the user clicks the control.
	OnContextPopup (see page 13013)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 13014)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 13014)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13016)	Occurs when the user drags an object over a control.
	OnEndDock (see page 13019)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 13019)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 13020)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 13021)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 13023)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 13024)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 13024)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 13026)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 13028)	OnPaint occurs when the paintbox receives a Windows paint message.
	OnStartDock (see page 13030)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 13031)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

68.1.29.5.1 TPaintBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check

box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.

- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
```

```
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

68.1.29.5.2 TPaintBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

68.1.29.5.3 TPaintBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

68.1.29.5.4 TPaintBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}
```

```
/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
```

```
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

68.1.29.5.5 TPaintBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;
```

```
procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.29.5.6 TPaintBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.29.5.7 TPaintBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
```

```
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

68.1.29.5.8 TPaintBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears

to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.29.5.9 TPaintBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
```

```
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
```

```
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.29.5.10 TPaintBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

68.1.29.5.11 TPaintBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

68.1.29.5.12 TPaintBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;
```

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.29.5.13 TPaintBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
```

```
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
```

```
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

68.1.29.5.14 TPaintBox.OnPaint Event

OnPaint occurs when the paintbox receives a Windows paint message.

Pascal

```
property OnPaint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPaint;
```

Description

Write an OnPaint event handler to draw the image of the paint box. The OnPaint event fires when the system repaints the TPaintBox (see page 12981) control. If there is a handler for this event, its code is executed when the OnPaint event fires.

Use properties and methods of the canvas (such as LineTo, Draw, and TextOut) to programmatically construct the image that the paintbox presents on a form. This can consist of a number of operations including drawing lines, images, and shapes. A TPaintBox (see page 12981) itself has no visible manifestation, so if an OnPaint event handler is not provided or no drawing is done in the event handler, the paintbox can not be seen on the form at runtime.

C++ Examples:

```
/*
When you do custom drawing, such as shown in the code for
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   PaintBox1->Canvas->Brush->Color = clRed;
   PaintBox1->Canvas->FillRect(PaintBox1->Canvas->ClipRect);
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}

/*
In order for the drawing to be persistent, you need to also
include an OnPaint event handler that tells the PaintBox how
to redraw itself when it needs to. In the code below, an
ellipse will be drawn everytime the PaintBox renders itself,
but the PaintBox will only paint the client area of the
control red after the Button is clicked. This client area
will only stay red until it is invalidated, but the ellipse
will persist. This behavior is different from the TImage
component, because the TImage component maintains an
internal bitmap that stores this drawing information for
you.
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
   PaintBox1->Canvas->Brush->Color = clBlue;
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}
/*
This example paints a white five-pointed star in a paintbox
control:
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
  TPaintBox *pPB = (TPaintBox *)Sender;
  TPoint points[6];
  pPB->Canvas->Pen->Color = clWhite;
  points[0].x = 40;
  points[0].y = 10;
  points[1].x = 20;
  points[1].y = 60;
  points[2].x = 70;
  points[2].y = 30;
  points[3].x = 10;
  points[3].y = 30;
  points[4].x = 60;
  points[4].y = 60;
  points[5].x = 40;
  points[5].y = 10;
  pPB->Canvas->Polyline(points,5);
}
```

Delphi Examples:

```
{
When you do custom drawing, such as shown in the code for
```

```
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   PaintBox1.Canvas.Brush.Color := clRed;
   PaintBox1.Canvas.FillRect(PaintBox1.Canvas.ClipRect);
   PaintBox1.Canvas.Brush.Color := clBlue;
   PaintBox1.Canvas.Ellipse(0,0,100,50);
end;
{
In order for the drawing to be persistent, you need to also
include an OnPaint event handler that tells the PaintBox how
to redraw itself when it needs to. In the code below, an
ellipse will be drawn everytime the PaintBox renders itself,
but the PaintBox will only paint the client area of the
control red after the Button is clicked. This client area
will only stay red until it is invalidated, but the ellipse
will persist. This behavior is different from the TImage
component, because the TImage component maintains an
internal bitmap that stores this drawing information for
you.
}
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
   PaintBox1.Canvas.Brush.Color := clBlue;
   PaintBox1.Canvas.Ellipse(0,0,100,50);
end;
{
This example paints a white five-pointed star in a paintbox
control:
}
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
  with Sender as TPaintBox do
  begin
    Canvas.Pen.Color := clWhite;
    Canvas.Polyline([Point(40, 10), Point(20, 60), Point(70, 30),
    Point(10, 30), Point(60, 60), Point(40, 10)]);
  end;
end;
```

See Also

Canvas (see page 6053)

TCanvas (see page 14441)

68.1.29.5.15 TPaintBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the

appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

68.1.29.5.16 TPaintBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

68.1.30 TPanel Class

TPanel implements a generic panel control.

Class Hierarchy

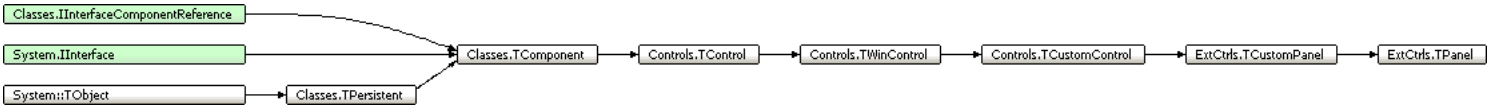

Pascal

```
TPanel = class(TCustomPanel);
```

C++

```
class TPanel : public TCustomPanel;
```

File

ExtCtrls

Description

Use TPanel to put an empty panel on a form. Panels have properties for providing a beveled border around the control, as well as methods to help manage the placement of child controls embedded in the panel.

You can also use panels to group controls together, similar to the way you can use a group box, but with a beveled border (or no border) rather than the group box outline. Panels are typically used for groups of controls within a single form. If you intend to use the same grouping in other forms, you may want to use a frame instead.

Panels with no borders are useful as docking sites when writing applications that use drag-and-dock.

Although you can use a panel to implement a status bar or tool bar, it is recommended that you use the TToolBar and TStatusBar classes instead.

See Also

TToolBar (see page 4994)

TStatusBar (see page 4769)

TGroupBox (see page 21474)

Grouping Controls

Panels

Adding a Cool Bar Component

Adding a Speed Button to a Panel

Adding a Toolbar Using a Panel Component

Adding a Toolbar Using the Toolbar Component

Adding a Tool Button

Adding Hidden Toolbars

Allowing Toggle Buttons

Allowing Toggled Tool Buttons

Assigning a Menu to a Tool Button

Assigning a Speed Button's Glyph

Assigning Images to Tool Buttons

Creating a Group of Speed Buttons

Creating Groups of Tool Buttons

Designing Toolbars and Cool Bars

Hiding and Showing Toolbars

Responding to Clicks

Setting the Initial Condition of a Speed Button

Setting Tool Button Appearance and Initial Conditions

68.1.30.1 TPanel Members

The following tables list the members exposed by TPanel.

Protected Events

	Name	Description
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.

	AdjustClientRect (see page 12537)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanAutoSize (see page 12537)	Adjusts the way the panel responds to resize attempts.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 12537)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.

	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event

	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 5920)	Renders the image of a custom control.
V	Paint (see page 12538)	Renders the image of the panel.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.

	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBackground (see page 12539)	This is SetParentBackground, a member of class TCustomPanel.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 5918)	Creates an instance of TCustomControl (see page 5908).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 12536)	Creates and initializes an instance of TCustomPanel (see page 12525).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.

	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetControlsAlignment (see page 12538)	Returns the alignment of text in the panel.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.

	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetDesignVisible (see page 6178)	Makes a control visible at design time.
	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Update (see page 5789)	Processes any pending paint messages immediately.
	Update (see page 6183)	Forces the control to update.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 13060)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.

	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 13080)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 13081)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 13082)	Occurs when an attempt is made to resize the control.
	OnClick (see page 13082)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 13084)	Adjust resize constraints.
	OnContextPopup (see page 13084)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 13085)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 13085)	Occurs when another control is docked to the control.
	OnDockOver (see page 13087)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 13090)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13092)	Occurs when the user drags an object over a control.
	OnEndDock (see page 13094)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 13095)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 13096)	Occurs when a control receives the input focus.
	OnExit (see page 13098)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 13100)	Returns the control's docking information.
	OnMouseActivate (see page 13102)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 13103)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 13105)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 13105)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 13105)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 13108)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 13110)	Occurs immediately after the control is resized.
	OnStartDock (see page 13111)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 13111)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 13112)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 13044)	Determines how the control aligns within its container (parent control).
	Alignment (see page 13048)	Determines how the caption is aligned within the panel.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 13048)	Specifies how the control is anchored to its parent.
	AutoSize (see page 13049)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 13049)	Specifies which edges of the control are beveled.
	BevelInner (see page 13050)	Determines the style of the inner bevel of a panel.
	BevelKind (see page 13051)	Specifies the control's bevel style.
	BevelOuter (see page 13051)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 13052)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BiDiMode (see page 13053)	Specifies the bi-directional mode for the control.

	BorderStyle (see page 13054)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 13054)	Specifies the distance, in pixels, between the outer and inner bevels.
	Caption (see page 13055)	Specifies a text string that identifies the control to the user.
	Color (see page 13058)	Specifies the background color of the control.
	Constraints (see page 13059)	Specifies the size constraints for the control.
	Ctl3D (see page 13059)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DockSite (see page 13062)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 13063)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 13063)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 13063)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 13065)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 13066)	Controls the attributes of text written on or in the control.
	FullRepaint (see page 13067)	Determines how the panel repaints itself when it is resized.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Locked (see page 13067)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Padding (see page 13067)	Specifies the padding of a control.
	ParentBackground (see page 13068)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 13068)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 13068)	Determines where a control looks for its color information.
	ParentCtl3D (see page 13069)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 13069)	Determines where a control looks for its font information.
	ParentShowHint (see page 13070)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 13070)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 13075)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 13076)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 13077)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	UseDockManager (see page 13077)	Specifies whether the docking manager is used in drag-and-dock operations.
	VerticalAlignment (see page 13078)	Sets the vertical position of the caption.
	Visible (see page 13078)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.30.2 TPanel Properties

The properties of the TPanel class are listed here.

Public Properties

	Name	Description
	DockManager (see page 13060)	Specifies the control's docking manager interface.

Published Properties

	Name	Description
	Align (see page 13044)	Determines how the control aligns within its container (parent control).
	Alignment (see page 13048)	Determines how the caption is aligned within the panel.
	Anchors (see page 13048)	Specifies how the control is anchored to its parent.
	AutoSize (see page 13049)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 13049)	Specifies which edges of the control are beveled.
	BevelInner (see page 13050)	Determines the style of the inner bevel of a panel.
	BevelKind (see page 13051)	Specifies the control's bevel style.
	BevelOuter (see page 13051)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 13052)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BiDiMode (see page 13053)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 13054)	Determines the style of the line drawn around the perimeter of the panel control.
	BorderWidth (see page 13054)	Specifies the distance, in pixels, between the outer and inner bevels.
	Caption (see page 13055)	Specifies a text string that identifies the control to the user.
	Color (see page 13058)	Specifies the background color of the control.
	Constraints (see page 13059)	Specifies the size constraints for the control.
	Ctl3D (see page 13059)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 13062)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 13063)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 13063)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 13063)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 13065)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 13066)	Controls the attributes of text written on or in the control.
	FullRepaint (see page 13067)	Determines how the panel repaints itself when it is resized.
	Locked (see page 13067)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	Padding (see page 13067)	Specifies the padding of a control.
	ParentBackground (see page 13068)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 13068)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 13068)	Determines where a control looks for its color information.
	ParentCtl3D (see page 13069)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 13069)	Determines where a control looks for its font information.
	ParentShowHint (see page 13070)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 13070)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 13075)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 13076)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 13077)	Determines if the user can tab to a control.
	UseDockManager (see page 13077)	Specifies whether the docking manager is used in drag-and-dock operations.
	VerticalAlignment (see page 13078)	Sets the vertical position of the caption.
	Visible (see page 13078)	Determines whether the component appears onscreen.

68.1.30.2.1 TPanel.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;
```

```
  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
```

```
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
```

```
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

68.1.30.2.2 TPanel.Alignment Property

Determines how the caption is aligned within the panel.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use the Alignment property to specify if the caption of a panel appears on the left, right, or center of the panel. For example, to use a panel as a status bar that displays the TApplication. ::Hint property as the panel's Caption property, set Alignment to taLeftJustify to place the hint text on the left side of the panel.

Note: Use the VerticalAlignment property to control the vertical position of the caption.

See Also

Hint (see page 13712)

Caption (see page 5806)

68.1.30.2.3 TPanel.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

68.1.30.2.4 TPanel.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints (see page 5812)

CanAutoSize (see page 5741)

68.1.30.2.5 TPanel.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

68.1.30.2.6 TPanel.BevelInner Property

Determines the style of the inner bevel of a panel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TPanelBevel BevelInner;__property TBevelCut BevelInner;
```

Description

Use BevelInner to provide the panel with a three-dimensional beveled look. A panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel, separated by the number of pixels specified in the BorderWidth property. The width of the bevels, if they are drawn, is the BevelWidth property.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
```

```
      BevelOuter := bvRaised;
      BorderWidth := 1;
      BevelWidth := 1;
    end;
  end;
```

See Also

BevelOuter (see page 12541)

BevelWidth (see page 12542)

BorderWidth (see page 12543)

68.1.30.2.7 TPanel.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

68.1.30.2.8 TPanel.BevelOuter Property

Determines the style of the outer bevel of a panel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TPanelBevel BevelOuter;__property TBevelCut BevelOuter;
```

Description

Use BevelOuter to provide the panel with a three-dimensional beveled look. A panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel, separated by the number of pixels specified in the BorderWidth property. The width of the outer bevel is specified in the BevelWidth property in pixels.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
```

```
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelInner (see page 12540)

BevelWidth (see page 12542)

BorderWidth (see page 12543)

68.1.30.2.9 TPanel.BevelWidth Property

Determines the width, in pixels, of both the inner and outer bevels of a panel.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify how wide the inner or outer bevel should be. Do not confuse BevelWidth, which is the width of the bevels, with BorderWidth, which is the space between the bevels.

If both the BevelInner and BevelOuter properties are bvNone, BevelWidth has no effect. To remove both bevels, set the BevelInner and BevelOuter properties to bvNone, rather than setting the BevelWidth to 0, as this involves less overhead when painting.

C++ Examples:

```
/*
```

```
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelInner (see page 12540)

BevelOuter (see page 12541)

BorderWidth (see page 12543)

68.1.30.2.10 TPanel.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency,

ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

68.1.30.2.11 TPanel.BorderStyle Property

Determines the style of the line drawn around the perimeter of the panel control.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to specify whether the panel has a single line drawn around it. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

Do not confuse the line drawn around the panel with the BorderWidth of the panel. The BorderWidth of the panel is the distance between the outer and inner bevels.

See Also

BorderWidth (see page 12543)

68.1.30.2.12 TPanel.BorderWidth Property

Specifies the distance, in pixels, between the outer and inner bevels.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to specify how wide the border around the panel should be. A value of 0 (zero) means no border should appear.

The border of a panel is the area between the outer and inner bevels. It is visible only if the inner bevel is raised or lowered, but affects the inset of the caption within the panel even if BevelInner is bvNone. If the Alignment property is not taCenter, the Caption will be aligned to the inner edge of the border. This edge is BorderWidth pixels in from the outer bevel if BevelInner is bvNone. It is the inner edge of the inner bevel otherwise.

Do not confuse the border of the panel with line drawn around the panel itself. The line around the panel is specified by the BorderStyle property.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

Alignment (see page 12539)

BevelInner (see page 12540)

BevelOuter (see page 12541)

BorderStyle (see page 12543)

Caption (see page 5806)

68.1.30.2.13 TPanel.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of

character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
```

```
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
```

```
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

68.1.30.2.14 TPanel.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

68.1.30.2.15 TPanel.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

68.1.30.2.16 TPanel.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

68.1.30.2.17 TPanel.DockManager Property

Specifies the control's docking manager interface.

Pascal

```
property DockManager: IDockManager;
```

C++

```
__property IDockManager DockManager;
```

Description

Use DockManager to specify the control's docking manager. The docking manager handles the layout of docking zones (where controls are docked) and any painting associated with docking zones.

If you set DockSite and UseDockManager to true, but do not assign a value to DockManager, the windowed control generates a default docking manager, using the global DefaultDockTreeClass variable.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}
```

```
void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;
```

```
procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

UseDockManager (see page 6212)

DefaultDockTreeClass (see page 6321)

68.1.30.2.18 TPanel.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite: Boolean;
```

C++

```
__property Boolean DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
```

```
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

68.1.30.2.19 TPanel.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

68.1.30.2.20 TPanel.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

68.1.30.2.21 TPanel.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
```

```
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

68.1.30.2.22 TPanel.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

68.1.30.2.23 TPanel.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

68.1.30.2.24 TPanel.FullRepaint Property

Determines how the panel repaints itself when it is resized.

Pascal

```
property FullRepaint: Boolean;
```

C++

```
__property Boolean FullRepaint;
```

Description

FullRepaint controls how the panel responds when it is resized. When FullRepaint is true, the entire panel, including the beveled border repaints when the size changes. When FullRepaint is false, only the area inside the beveled border repaints.

See Also

Invalidate (see page 6161)

68.1.30.2.25 TPanel.Locked Property

Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.

Pascal

```
property Locked: Boolean;
```

C++

```
__property Boolean Locked;
```

Description

Set the Locked property to false when the panel is used as a toolbar by a TOleContainer component. Any panels aligned to a TOleContainer are assumed to be replaceable toolbars unless the Locked property is true. Prevent the panel from being replaced by an OLE server's toolbar by setting the panel's Locked property to true.

Note: An OLE application's toolbars must be in panels aligned to a side of the form (that is, their Align properties must be set to alTop, alBottom, alLeft, or alRight).

See Also

Align (see page 5798)

TOleContainer (see page 17921)

68.1.30.2.26 TPanel.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding: TPadding;
```

C++

```
__property TPadding Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding (see page 6103)

68.1.30.2.27 TPanel.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground: Boolean;
```

C++

```
__property Boolean ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

68.1.30.2.28 TPanel.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

68.1.30.2.29 TPanel.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

68.1.30.2.30 TPanel.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

68.1.30.2.31 TPanel.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

68.1.30.2.32 TPanel.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

68.1.30.2.33 TPanel.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
```

```
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
```

```
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
```

```
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
```

```
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
```

```
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

68.1.30.2.34 TPanel.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
```

```
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

68.1.30.2.35 TPanel.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

68.1.30.2.36 TPanel.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

68.1.30.2.37 TPanel.UseDockManager Property

Specifies whether the docking manager is used in drag-and-dock operations.

Pascal

```
property UseDockManager: Boolean;
```

C++

```
__property Boolean UseDockManager;
```

Description

Use UseDockManager to get or set whether a docking manager is used in drag-and-dock operations.

The docking manager handles the positioning of docked controls and any painting of docking zones around those controls. If you do not use a docking manager, docked controls are aligned to the windowed control based on the closest edge to where they are released. When you do not use a docking manager, there is no region for a user to grab to automatically undock a docked control.

See Also

DockManager (see page 6202)

68.1.30.2.38 TPanel.VerticalAlignment Property

Sets the vertical position of the caption.

Pascal

```
property VerticalAlignment: TVerticalAlignment;
```

C++

```
__property TVerticalAlignment VerticalAlignment;
```

Description

Use the VerticalAlignment property to control the position of the caption within the panel. Choose one of the following values.

Value	Description
taAlignBottom	The caption displays at the bottom of the panel.
taAlignTop	The caption displays at the top of the panel.
taVerticalCenter	The caption displays at the center of the panel.

Note: Use the Alignment property to control the horizontal position of the caption.

See Also

Caption

68.1.30.2.39 TPanel.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

68.1.30.3 TPanel Events

The events of the TPanel class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 13080)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 13081)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 13082)	Occurs when an attempt is made to resize the control.
	OnClick (see page 13082)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 13084)	Adjust resize constraints.
	OnContextPopup (see page 13084)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 13085)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 13085)	Occurs when another control is docked to the control.
	OnDockOver (see page 13087)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 13090)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13092)	Occurs when the user drags an object over a control.
	OnEndDock (see page 13094)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 13095)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 13096)	Occurs when a control receives the input focus.
	OnExit (see page 13098)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 13100)	Returns the control's docking information.
	OnMouseActivate (see page 13102)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 13103)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 13105)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 13105)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 13105)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 13108)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 13110)	Occurs immediately after the control is resized.
	OnStartDock (see page 13111)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 13111)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 13112)	Occurs when the application tries to undock a control that is docked to the windowed control.

68.1.30.3.1 TPanel.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore: TAlignInsertBeforeEvent;
```

C++

```
__property TAlignInsertBeforeEvent OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore (see page 6137)

Align (see page 5798)

TAlignInsertBeforeEvent (see page 6293)

68.1.30.3.2 TPanel.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition: TAlignPositionEvent;
```

C++

```
__property TAlignPositionEvent OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition (see page 6138)

Align (see page 5798)

Anchors (see page 5802)

TAlignInfo (see page 6269)

TAlignPositionEvent (see page 6293)

68.1.30.3.3 TPanel.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize: TCanResizeEvent;
```

C++

```
__property TCanResizeEvent OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize (see page 5853)

SetBounds (see page 5783)

OnResize (see page 5870)

68.1.30.3.4 TPanel.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
```

```
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

68.1.30.3.5 TPanel.OnConstrainedResize Event

Adjust resize constraints.

Pascal

```
property OnConstrainedResize: TConstrainedResizeEvent;
```

C++

```
__property TConstrainedResizeEvent OnConstrainedResize;
```

Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

Note: The OnConstrainedResize handler is called immediately after the OnCanResize handler.

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

Constraints (see page 5812)

OnResize (see page 5870)

CanAutoSize (see page 5741)

68.1.30.3.6 TPanel.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

68.1.30.3.7 TPanel.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

68.1.30.3.8 TPanel.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop: TDockDropEvent;
```

C++

```
__property TDockDropEvent OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
```

```
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockOver (see page 6216)

68.1.30.3.9 TPanel.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver: TDockOverEvent;
```

C++

```
__property TDockOverEvent OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
```

```
      ShowMessage("GetSiteInfo CanDock went bad!");
    }
  }
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnGetSiteInfo (see page 6223)

TDragDockObject (see page 6009)

68.1.30.3.10 TPanel.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

68.1.30.3.11 TPanel.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}
```

```
/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
```

```
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.30.3.12 TPanel.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.30.3.13 TPanel.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

68.1.30.3.14 TPanel.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
```

```
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
```

```
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

68.1.30.3.15 TPanel.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK

button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

68.1.30.3.16 TPanel.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo: TGetSiteInfoEvent;
```

C++

```
__property TGetSiteInfoEvent OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
```

```
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

TDragDockObject (see page 6009)

68.1.30.3.17 TPanel.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.30.3.18 TPanel.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
```

```
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
```

```
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.30.3.19 TPanel.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

68.1.30.3.20 TPanel.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

68.1.30.3.21 TPanel.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
```

```
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
```

```
      end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.30.3.22 TPanel.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}
```

```
// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
```

```
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

68.1.30.3.23 TPanel.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

68.1.30.3.24 TPanel.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

68.1.30.3.25 TPanel.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible

for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

68.1.30.3.26 TPanel.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock: TUnDockEvent;
```

C++

```
__property TUnDockEvent OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/
```

```
void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
```

```
    if Client is TDockableForm then
      TDockableForm(Client).DockSite := True;
    if (DockClientCount = 2) and (NewTarget <> Self) then
      PostMessage(Self.Handle, WM_CLOSE, 0, 0);
    UpdateCaption(Client);
  end;
  {
  The following example is taken from the docking demo. It
  shows how set permissions to dock dockable objects onto a
  docking site.
  }

  procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer);
  begin
    //OnDockDrop gets called AFTER the client has actually docked,
    //so we check for DockClientCount = 1 before making the dock panel visible.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, True, nil);
    (Sender as TPanel).DockManager.ResetBounds(True);
    //Make DockManager repaints it's clients.
  end;

  procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer; State: TDragState;
    var Accept: Boolean);
  var
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

68.1.31 TRadioGroup Class

TRadioGroup represents a group of radio buttons that function together.

Class Hierarchy

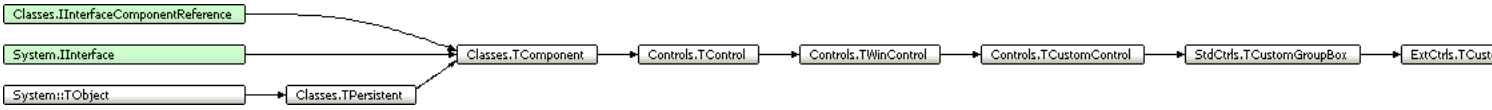

Pascal

```
TRadioGroup = class(TCustomRadioGroup);
```

C++

```
class TRadioGroup : public TCustomRadioGroup;
```

File

ExtCtrls

Description

A TRadioGroup object is a special group box that contains only radio buttons. Radio buttons that are placed directly in the same control component are said to be "grouped." When the user checks a radio button, all other radio buttons in its group become unchecked. Hence, two radio buttons on a form can be checked at the same time only if they are placed in separate containers, such as group boxes.

To add radio buttons to a TRadioGroup, edit the Items (see page 13141) property in the Object Inspector. Each string in Items (see page 13141) makes a radio button appear in the group box with the string as its caption. The value of the ItemIndex (see page 13140) property determines which radio button is currently selected.

Display the radio buttons in a single column or in multiple columns by setting the Columns (see page 13135) property.

Note: Setting the radio group's BiDiMode (see page 13131) to bdRightToLeft automatically flips its radio buttons. The FlipChildren (see page 12559) method has no effect.

See Also

TDBRadioGroup (see page 9283)

TRadioButton (see page 21753)

Group Boxes and Radio Groups

Grouping Controls

68.1.31.1 TRadioGroup Members

The following tables list the members exposed by TRadioGroup.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.

	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 21287)	Adjusts the ClientRect property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.

	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanModify (see page 12559)	Determines whether the user can select a different radio button in the group.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 21287)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.

	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetChildren (see page 12560)	Prevents the streaming system from loading and saving child controls in the usual fashion.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 12560)	Initializes the radio group after it is loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 5920)	Renders the image of a custom control.
V	Paint (see page 21287)	Renders the onscreen image of the group box.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
V	ReadState (see page 12560)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.

	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.

	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 5918)	Creates an instance of TCustomControl (see page 5908).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 12558)	Creates and initializes a TCustomRadioGroup (see page 12547) instance.
V	Create (see page 21286)	Creates and initializes a TCustomGroupBox (see page 21275) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.

	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 12558)	Destroys the TCustomRadioGroup (see page 12547) instance and the radio buttons it contains.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	FlipChildren (see page 12559)	Prevents the default behavior of flipping the child radio buttons.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.

	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.

V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Buttons (see page 12561)	Access individual TRadioButton objects.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.

	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 13153)	Occurs when the user clicks the control.
	OnContextPopup (see page 13155)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 13156)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13158)	Occurs when the user drags an object over a control.
	OnEndDock (see page 13161)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 13161)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 13162)	Occurs when a control receives the input focus.
	OnExit (see page 13164)	Occurs when the input focus shifts away from one control to another.
	OnStartDock (see page 13166)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 13167)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 13127)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 13130)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 13131)	Specifies the bi-directional mode for the control.
	Caption (see page 13131)	Specifies a text string that identifies the control to the user.
	Color (see page 13134)	Specifies the background color of the control.
	Columns (see page 13135)	Specifies number of columns in the radio group.
	Constraints (see page 13135)	Specifies the size constraints for the control.
	Ctl3D (see page 13135)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 13136)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 13136)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 13137)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 13138)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 13139)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ItemIndex (see page 13140)	Indicates which radio button in the group is currently selected.
	Items (see page 13141)	Lists the radio buttons in the radio group.

	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBackground (see page 13142)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 13142)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 13143)	Determines where a control looks for its color information.
	ParentCtl3D (see page 13143)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 13143)	Determines where a control looks for its font information.
	ParentShowHint (see page 13144)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 13144)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 13149)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 13150)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 13151)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 13151)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.31.2 TRadioGroup Properties

The properties of the TRadioGroup class are listed here.

Published Properties

	Name	Description
	Align (see page 13127)	Determines how the control aligns within its container (parent control).
	Anchors (see page 13130)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 13131)	Specifies the bi-directional mode for the control.
	Caption (see page 13131)	Specifies a text string that identifies the control to the user.
	Color (see page 13134)	Specifies the background color of the control.
	Columns (see page 13135)	Specifies number of columns in the radio group.
	Constraints (see page 13135)	Specifies the size constraints for the control.
	Ctl3D (see page 13135)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 13136)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 13136)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 13137)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 13138)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 13139)	Controls the attributes of text written on or in the control.
	ItemIndex (see page 13140)	Indicates which radio button in the group is currently selected.
	Items (see page 13141)	Lists the radio buttons in the radio group.
	ParentBackground (see page 13142)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 13142)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 13143)	Determines where a control looks for its color information.
	ParentCtl3D (see page 13143)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 13143)	Determines where a control looks for its font information.
	ParentShowHint (see page 13144)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 13144)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 13149)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 13150)	Indicates the position of the control in its parent's tab order.

	TabStop (see page 13151)	Determines if the user can tab to a control.
	Visible (see page 13151)	Determines whether the component appears onscreen.

68.1.31.2.1 TRadioGroup.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;
```

```
  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form’s width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
```

```
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

68.1.31.2.2 TRadioGroup.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other

aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

68.1.31.2.3 TRadioGroup.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

68.1.31.2.4 TRadioGroup.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

```
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
```

```
    ToolBar1.ShowCaptions := True;
    ToolBar1.Height := 40;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I: Integer;
  begin
    for I := ToolBar1.ButtonCount - 1 downto 0 do
      ToolBar1.Buttons[I].Free;
    ToolBar1.Free;
  end;
```

68.1.31.2.5 TRadioGroup.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

68.1.31.2.6 TRadioGroup.Columns Property

Specifies number of columns in the radio group.

Pascal

```
property Columns: Integer;
```

C++

```
__property int Columns;
```

Description

Columns determines the number of columns in the radio group if Orientation is orVertical (the default), and the number of rows if Orientation is orHorizontal. Its value can range from 1 to 16. The default value is 1, which means that the radio buttons are arranged in a single line.

68.1.31.2.7 TRadioGroup.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

68.1.31.2.8 TRadioGroup.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

68.1.31.2.9 TRadioGroup.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

68.1.31.2.10 TRadioGroup.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

68.1.31.2.11 TRadioGroup.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
```

```
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

68.1.31.2.12 TRadioGroup.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

68.1.31.2.13 TRadioGroup.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;
```

```
procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

68.1.31.2.14 TRadioGroup.ItemIndex Property

Indicates which radio button in the group is currently selected.

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

Description

ItemIndex holds the index of the selected radio button in the Items list. (The first button is 0.) The value of ItemIndex changes at runtime as the user selects radio buttons. If you want one of the buttons to appear selected when the application starts, assign that button to ItemIndex at design time; otherwise, leave ItemIndex set to the default value of -1, which means that no button is selected.

C++ Examples:

```
/*
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Items->Add("Vertical");
  RadioGroup1->Items->Add("Horizontal");
  RadioGroup1->ItemIndex = 2;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Vertical")
    ScrollBar1->Kind = sbVertical;
  else if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Horizontal")
    ScrollBar1->Kind = sbHorizontal;
}
```

Delphi Examples:

```
{
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  RadioGroup1.Items.Add('Vertical');
  RadioGroup1.Items.Add('Horizontal');
  RadioGroup1.ItemIndex := 2;
```

```
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Vertical' then
    ScrollBar1.Kind := sbVertical;
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Horizontal' then
    ScrollBar1.Kind := sbHorizontal;
end;
```

See Also

Items (see page 12563)

68.1.31.2.15 TRadioGroup.Items Property

Lists the radio buttons in the radio group.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Items holds a TStrings object that lists the captions of the radio buttons in the group. In TCustomRadioGroup descendants, where this property is published, radio buttons can be added or removed by editing the Items list from the Object Inspector.

Note: A TRadioButton object is created automatically for each string in Items. You cannot directly add or remove TRadioButton objects, but you can access individual TButton objects through the Buttons property.

C++ Examples:

```
/*
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Items->Add("Vertical");
  RadioGroup1->Items->Add("Horizontal");
  RadioGroup1->ItemIndex = 2;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Vertical")
    ScrollBar1->Kind = sbVertical;
  else if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Horizontal")
    ScrollBar1->Kind = sbHorizontal;
}
```

Delphi Examples:

```
{
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    RadioGroup1.Items.Add('Vertical');
    RadioGroup1.Items.Add('Horizontal');
    RadioGroup1.ItemIndex := 2;
  end;

  procedure TForm1.RadioGroup1Click(Sender: TObject);
  begin
    if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Vertical' then
      ScrollBar1.Kind := sbVertical;
    if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Horizontal' then
      ScrollBar1.Kind := sbHorizontal;
  end;
```

See Also

ItemIndex (see page 12562)

TRadioButton (see page 21753)

Buttons (see page 12561)

68.1.31.2.16 TRadioGroup.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground: Boolean;
```

C++

```
__property Boolean ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

68.1.31.2.17 TRadioGroup.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

68.1.31.2.18 TRadioGroup.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

68.1.31.2.19 TRadioGroup.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

68.1.31.2.20 TRadioGroup.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

68.1.31.2.21 TRadioGroup.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

68.1.31.2.22 TRadioGroup.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
```

```
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      const int num_items = 4;
      for (int index = 0; index < num_items; ++index)
      {
          TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
          PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
          NewItem->Caption = "Menu Item " + IntToStr(index);
          NewItem->Tag = index;
          NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
          TNotifyEvent();
      }
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
          {
              case 0:
              {
                  ShowMessage("first item clicked");
                  break;
              }
              case 1:
              {
                  ShowMessage("second item clicked");
                  break;
              }
              case 2:
              {
                  ShowMessage("third item clicked");
                  break;
              }
              case 3:
              {
                  ShowMessage("fourth item clicked");
                  break;
              }
          }
      }
  }

  /*
```

```
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
```

```
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
```

```
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

68.1.31.2.23 TRadioGroup.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

68.1.31.2.24 TRadioGroup.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

68.1.31.2.25 TRadioGroup.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

68.1.31.2.26 TRadioGroup.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

68.1.31.3 TRadioGroup Events

The events of the TRadioGroup class are listed here.

Published Events

	Name	Description
	OnClick (see page 13153)	Occurs when the user clicks the control.
	OnContextPopup (see page 13155)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 13156)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13158)	Occurs when the user drags an object over a control.
	OnEndDock (see page 13161)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 13161)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 13162)	Occurs when a control receives the input focus.
	OnExit (see page 13164)	Occurs when the input focus shifts away from one control to another.
	OnStartDock (see page 13166)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 13167)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

68.1.31.3.1 TRadioGroup.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
```

```
  edit control placed somewhere on the client area of the
  sheet.  Unchecking the checkbox will cause the client area
  (not the tab area) of the Tab Control to become invisible.
  }
  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    PageControl1.ActivePage.Visible := CheckBox1.Checked;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    CheckBox1.Checked := PageControl1.ActivePage.Visible;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

68.1.31.3.2 TRadioGroup.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

68.1.31.3.3 TRadioGroup.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
```

```
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:
```

```
procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

68.1.31.3.4 TRadioGroup.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler

is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;
```

```
procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.31.3.5 TRadioGroup.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.31.3.6 TRadioGroup.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

68.1.31.3.7 TRadioGroup.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
```

```
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

68.1.31.3.8 TRadioGroup.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

68.1.31.3.9 TRadioGroup.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

68.1.31.3.10 TRadioGroup.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

68.1.32 TRowCollection Class

Controls (see page 5711) the **Collection Editor** for rows within a grid panel.

Class Hierarchy

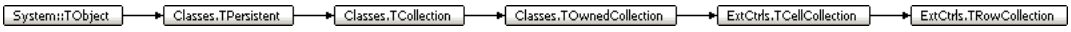

Pascal

```
TRowCollection = class(TCellCollection);
```

C++

```
class TRowCollection : public TCellCollection;
```

File

ExtCtrls

Description

TRowCollection controls the **Collection Editor** for rows within a grid panel.

See Also

Collection Editor

68.1.32.1 TRowCollection Members

The following tables list the members exposed by TRowCollection.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
V	GetAttr (see page 12174)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
V	GetAttrCount (see page 12175)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItem (see page 12175)	Returns a specified item in the collection.
V	GetItemAttr (see page 13170)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2915)	Returns the owner of an object.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Notify (see page 13171)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
	SetItem (see page 12176)	This is SetItem, a member of class TCellCollection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.
V	Update (see page 12176)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 13170)	Adds a row to the grid panel.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 13170)	This is Create, a member of class TRowCollection.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
	Owner (see page 12176)	Returns the Owner of the cell collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 2751)	Lists the items in the collection.
	Items (see page 12177)	Lists the items of the cell collection.

68.1.32.2 TRowCollection.Create Constructor

Pascal

```
constructor Create(AOwner: TPersistent);
```

C++

```
__fastcall TRowCollection(TPersistent * AOwner);
```

Description

This is Create, a member of class TRowCollection.

68.1.32.3 TRowCollection Methods

The methods of the TRowCollection class are listed here.

Protected Methods

	Name	Description
	GetItemAttr (see page 13170)	Returns the value of a custom attribute assigned to one of the collection's items.
	Notify (see page 13171)	Responds when items are added to or removed from the collection.

Public Methods

	Name	Description
	Add (see page 13170)	Adds a row to the grid panel.

68.1.32.3.1 TRowCollection.Add Method

Adds a row to the grid panel.

Pascal

```
function Add: TRowItem;
```

C++

```
__fastcall TRowItem Add();
```

Description

Use the Add method to add a row to the grid panel.

68.1.32.3.2 TRowCollection.GetItemAttr Method

Returns the value of a custom attribute assigned to one of the collection's items.

Pascal

```
function GetItemAttr(Index: Integer; ItemIndex: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetItemAttr(int Index, int ItemIndex);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetItemAttr returns the value of one of these attributes for a specified item in the collection.

Index identifies which attribute's value is desired. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

ItemIndex identifies the item whose attribute value is desired. This is an index into the Items property array.

As implemented in TCollection, GetItemAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttr (see page 2743)

GetAttrCount (see page 2744)

68.1.32.3.3 TRowCollection.Notify Method

Responds when items are added to or removed from the collection.

Pascal

```
procedure Notify(Item: TCollectionItem; Action: TCollectionNotification); override;
```

C++

```
virtual __fastcall Notify(TCollectionItem Item, TCollectionNotification Action);
```

Description

Notify is called automatically when the items in the collection change.

Item is the item that was just added to or that is about to be removed from the collection.

Action indicates whether item was added, is about to be removed, or is about to be deleted:

As implemented in TCollection, Notify calls Added when Action is cnAdded and calls Deleting when Action is cnDeleting. TCollection ignores the cnExtracting action. Descendant classes can override Notify to modify this behavior.

See Also

Added (see page 2736)

Deleting (see page 2741)

68.1.33 TRowItem Class

Class Hierarchy

Pascal

```
TRowItem = class(TCellItem);
```

C++

```
class TRowItem : public TCellItem;
```

File

ExtCtrls

Description

This is class ExtCtrls.TRowItem.

68.1.34 TShape Class

TShape represents a geometric shape that can be drawn on a form.

Class Hierarchy

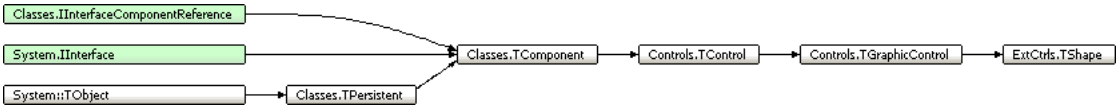

Pascal

```
TShape = class(TGraphicControl);
```

C++

```
class TShape : public TGraphicControl;
```

File

ExtCtrls

Description

Add a TShape object to a form to draw a simple geometric shape on the form. TShape introduces properties to describe the pen used to outline the shape and the brush used to fill it.

If the shape is only part of the image of a custom control, use the methods of the control's canvas instead.

See Also

TBrush (see page 14430)

TCanvas (see page 14441)

TPen (see page 14571)

TBevel (see page 12105)

Graphic Controls

Shapes

68.1.34.1 TShape Members

The following tables list the members exposed by TShape.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.

	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Paint (see page 13180)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.

	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 13179)	Creates an instance of TShape (see page 13172).
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 13180)	Destroys of an instance of TShape (see page 13172).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.

	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.

R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnContextPopup (see page 13194)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 13195)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13197)	Occurs when the user drags an object over a control.
	OnEndDock (see page 13199)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 13200)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 13201)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 13202)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 13204)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 13204)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 13205)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 13207)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 13209)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 13210)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Methods

	Name	Description
	StyleChanged (see page 13181)	This is StyleChanged, a member of class TShape.

Published Properties

	Name	Description
	Align (see page 13181)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 13185)	Specifies how the control is anchored to its parent.
	Brush (see page 13185)	Specifies the color and pattern used for filling the shape control.
	Constraints (see page 13186)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 13186)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 13187)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 13187)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 13189)	Controls whether the control responds to mouse, keyboard, and timer events.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentShowHint (see page 13189)	Determines where a control looks to find out if its Help Hint should be shown.
	Pen (see page 13190)	Specifies the pen used to outline the shape control.
	Shape (see page 13190)	Specifies the shape of the control.
	ShowHint (see page 13191)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 13192)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

68.1.34.2 TShape.Create Constructor

Creates an instance of TShape (see page 13172).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TShape(TComponent * AOwner);
```

Description

Call TShape (see page 13172) indirectly, using the new keyword, to instantiate a TShape (see page 13172) object at runtime. Shapes added to a form at design time are created automatically.

AOwner is the component, typically a form, that is responsible for freeing the shape object.

See Also

Owner (see page 2807)

68.1.34.3 TShape.Destroy Destructor

Destroys of an instance of TShape (see page 13172).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TShape();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the object is not nil before it calls Destroy.

See Also

Free (see page 22300)

68.1.34.4 TShape Methods

The methods of the TShape class are listed here.

Protected Methods

	Name	Description
	Paint (see page 13180)	Render control surface.

Published Methods

	Name	Description
	StyleChanged (see page 13181)	This is StyleChanged, a member of class TShape.

68.1.34.4.1 TShape.Paint Method

Render control surface.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when a lightweight control needs update its display area. Code in the Paint method uses the Canvas property to render the control surface.

As implemented in TGraphicControl, Paint does nothing. Specific rendering logic is provided by Paint methods in descendant control classes.

See Also

Canvas (see page 6053)

68.1.34.4.2 TShape.StyleChanged Method

Pascal

```
procedure StyleChanged(Sender: TObject);
```

C++

```
__fastcall StyleChanged(TObject * Sender);
```

Description

This is StyleChanged, a member of class TShape.

68.1.34.5 TShape Properties

The properties of the TShape class are listed here.

Published Properties

	Name	Description
	Align (see page 13181)	Determines how the control aligns within its container (parent control).
	Anchors (see page 13185)	Specifies how the control is anchored to its parent.
	Brush (see page 13185)	Specifies the color and pattern used for filling the shape control.
	Constraints (see page 13186)	Specifies the size constraints for the control.
	DragCursor (see page 13186)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 13187)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 13187)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 13189)	Controls whether the control responds to mouse, keyboard, and timer events.
	ParentShowHint (see page 13189)	Determines where a control looks to find out if its Help Hint should be shown.
	Pen (see page 13190)	Specifies the pen used to outline the shape control.
	Shape (see page 13190)	Specifies the shape of the control.
	ShowHint (see page 13191)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Visible (see page 13192)	Determines whether the component appears onscreen.

68.1.34.5.1 TShape.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
```

```
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
```

```
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
```

```
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

68.1.34.5.2 TShape.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

68.1.34.5.3 TShape.Brush Property

Specifies the color and pattern used for filling the shape control.

Pascal

```
property Brush: TBrush;
```

C++

```
__property TBrush Brush;
```

Description

Brush is a TBrush object that represents the color and pattern that fill the interior of the shape. Use the properties and methods of TBrush to change the color or pattern of the shape object:

```
Shape1.Brush.Color := clWhite;
Shape1->Brush->Color = clWhite;
```

Note: Setting the Brush property assigns the properties of another TBrush object to the shape's brush. The shape object does not take ownership of the assigned object.

See Also

Pen (see page 13190)

TBrush (see page 14430)

68.1.34.5.4 TShape.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

68.1.34.5.5 TShape.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

68.1.34.5.6 TShape.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

68.1.34.5.7 TShape.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
```

```
      AttachMode = naAddChild;
    else if (HT.Contains(htNowhere))
      AttachMode = naAdd;
    else if (HT.Contains(htOnIndent))
      AttachMode = naInsert;
    else
      return;
    TreeView1->Selected->MoveTo(Item, AttachMode);
  }

  void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
        int X, int Y, TDragState State, bool &Accept)
  {
    Accept = (Source == (TObject *)TreeView1);
  }
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

68.1.34.5.8 TShape.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

68.1.34.5.9 TShape.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

68.1.34.5.10 TShape.Pen Property

Specifies the pen used to outline the shape control.

Pascal

```
property Pen: TPen;
```

C++

```
__property TPen Pen;
```

Description

Read Pen to obtain the TPen object that represents the color and style of the line used to outline the shape. Use the properties of Pen to change the color or style.

Pen is a TPen object that represents the color and style of the line used to outline the shape. Use the properties and methods of TPen to change the color or pattern of the shape object:

Shape1.Pen.Width := 1;

Shape1.Pen.Color := Red;

Shape1->Pen->Width = 1;

Shape1->Pen->Color = clRed;

Note: Setting the Pen property assigns the properties of another TPen object to the shape's pen. The shape object does not take ownership of the assigned object.

See Also

Brush (see page 13185)

TPen (see page 14571)

68.1.34.5.11 TShape.Shape Property

Specifies the shape of the control.

Pascal

```
property Shape: TShapeType;
```

C++

```
__property TShapeType Shape;
```

Description

Set Shape to the geometric shape that should be drawn on the form. The Shape property has these possible values:

Value	Meaning
stCircle	The shape is a circle.
stEllipse	The shape is an ellipse.
stRectangle	The shape is a rectangle. (Default)
stRoundRect	The shape is a rectangle with rounded corners.
stRoundSquare	The shape is a square with rounded corners.

stSquare	The shape is a square.

68.1.34.5.12 TShape.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
```

```
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

68.1.34.5.13 TShape.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
```

```
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

68.1.34.6 TShape Events

The events of the TShape class are listed here.

Published Events

	Name	Description
	OnContextPopup (see page 13194)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 13195)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13197)	Occurs when the user drags an object over a control.
	OnEndDock (see page 13199)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 13200)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 13201)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 13202)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 13204)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 13204)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 13205)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 13207)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 13209)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 13210)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

68.1.34.6.1 TShape.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result,

when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

68.1.34.6.2 TShape.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
```

```
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
```

```
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

68.1.34.6.3 TShape.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/
```

```
// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.34.6.4 TShape.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

68.1.34.6.5 TShape.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
```

```
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

68.1.34.6.6 TShape.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.34.6.7 TShape.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}
```

```
// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
```

```
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.34.6.8 TShape.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

68.1.34.6.9 TShape.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

68.1.34.6.10 TShape.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

68.1.34.6.11 TShape.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
```

```
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
```

```
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

68.1.34.6.12 TShape.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

68.1.34.6.13 TShape.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

68.1.35 TSplitter Class

TSplitter divides the client area of a form into resizable panes.

Class Hierarchy

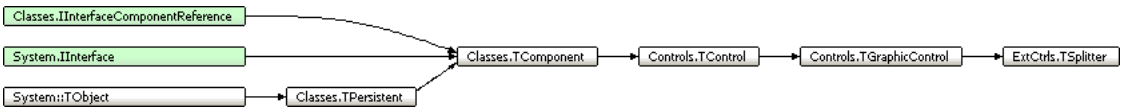

Pascal

```
TSplitter = class(TGraphicControl);
```

C++

```
class TSplitter : public TGraphicControl;
```

File

ExtCtrls

Description

Add a splitter to a form between two aligned controls to allow users to resize the controls at runtime. The splitter sits between a control aligned to one edge of the form and the controls that fill up the rest of the client area. Give the splitter the same alignment as the control that is anchored to the edge of the form. When the user moves the splitter, it resizes the anchored control. This, in turn, changes the client area of the form, and the controls that fill up the rest of the client area resize accordingly.

Use each control on the form as a separate pane. After each pane is placed, place a splitter with the same alignment to allow that pane to be resized. The last pane to be placed on the form should be client-aligned, so that it resizes automatically to fill up the remaining space after all other panes are resized.

See Also

TForm (see page 13992)

Splitter Controls

68.1.35.1 TSplitter Members

The following tables list the members exposed by TSplitter.

Protected Events

	Name	Description
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.

	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 13220)	Determines how the splitter responds to mouse moves.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DoCanResize (see page 13221)	Generates an OnCanResize (see page 13240) event and enforces the MinSize (see page 13235) property.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.

	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 13221)	Dynamically locates the control that can be resized by the splitter.
V	MouseMove (see page 13222)	Provides visual feedback of how the neighboring control would be resized.
V	MouseUp (see page 13222)	Resizes the neighboring control when the user releases the mouse.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 13222)	Draws the image of the splitter.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	RequestAlign (see page 13223)	Changes the cursor when the alignment of the splitter changes.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.

	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	StopSizing (see page 13223)	Cleans up after the splitter has resized its neighboring control.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.

	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Create (see page 13218)	Creates an instance of TSplitter (see page 13210).
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 13220)	Frees the memory associated with the splitter.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Canvas (see page 13228)	Provides the drawing surface used by the graphic control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCanResize (see page 13240)	Occurs when the user tries to move the splitter with the mouse.
	OnMoved (see page 13240)	Occurs after the user has retiled the form using the splitter.
	OnPaint (see page 13241)	Occurs when the splitter needs to draw itself.

Published Properties

	Name	Description
	Align (see page 13224)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AutoSnap (see page 13228)	Determines whether neighboring objects are resized to zero when the splitter is used to make them smaller than MinSize (see page 13235).
	Beveled (see page 13228)	Determines whether the splitter looks beveled along the edge that moves.
	Color (see page 13231)	Specifies the background color of the control.
	Constraints (see page 13231)	Specifies the size constraints for the control.
	Cursor (see page 13232)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	MinSize (see page 13235)	Specifies the minimum size, in pixels, of the panes on either side of the splitter.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentColor (see page 13236)	Determines where a control looks for its color information.
	ResizeStyle (see page 13237)	Specifies the effect of moving the splitter.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 13237)	Determines whether the component appears onscreen.
	Width (see page 13239)	Specifies the horizontal size of the control or form in pixels.

68.1.35.2 TSplitter.Create Constructor

Creates an instance of TSplitter (see page 13210).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSplitter(TComponent * AOwner);
```

Description

Call Create to instantiate a splitter at runtime. TSplitter (see page 13210) objects placed on forms at design time are created automatically.

AOwner specifies the form whose surface the splitter divides.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
```

```
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form’s width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

See Also

Owner (see page 2807)

68.1.35.3 TSplitter.Destroy Destructor

Frees the memory associated with the splitter.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSplitter();
```

Description

Do not call Destroy directly. Instead,call Free, which checks that the splitter is not nil before calling Destroy.

See Also

Free (see page 22300)

68.1.35.4 TSplitter Methods

The methods of the TSplitter class are listed here.

Protected Methods

	Name	Description
	CanResize (see page 13220)	Determines how the splitter responds to mouse moves.
	DoCanResize (see page 13221)	Generates an OnCanResize (see page 13240) event and enforces the MinSize (see page 13235) property.
	MouseDown (see page 13221)	Dynamically locates the control that can be resized by the splitter.
	MouseMove (see page 13222)	Provides visual feedback of how the neighboring control would be resized.
	MouseUp (see page 13222)	Resizes the neighboring control when the user releases the mouse.
	Paint (see page 13222)	Draws the image of the splitter.
	RequestAlign (see page 13223)	Changes the cursor when the alignment of the splitter changes.
	StopSizing (see page 13223)	Cleans up after the splitter has resized its neighboring control.

68.1.35.4.1 TSplitter.CanResize Method

Determines how the splitter responds to mouse moves.

Pascal

```
function CanResize(var NewSize: Integer): Boolean; virtual; reintroduce;
```

C++

```
virtual __fastcall Boolean CanResize(int NewSize);
```

Description

CanResize is called by the DoCanResize (see page 13221) method to generate an OnCanResize (see page 13240) event. The NewSize parameter indicates the size the associated control will have if the splitter is allowed to perform the resize. CanResize returns a boolean value that indicates whether the value specified by NewSize should be applied by redrawing the line (when ResizeStyle (see page 13237) is rsLine) or resizing the associated control (when ResizeStyle (see page 13237) is rsUpdate);

Unless an OnCanResize (see page 13240) event handler sets the value of NewSize and the return value, CanResize returns true without changing the NewSize parameter.

Note: CanResize is not called if the value of ResizeStyle (see page 13237) is rsNone when the user presses the mouse button.

See Also

OnCanResize (see page 13240)

ResizeStyle (see page 13237)

DoCanResize (see page 13221)

68.1.35.4.2 TSplitter.DoCanResize Method

Generates an OnCanResize (see page 13240) event and enforces the MinSize (see page 13235) property.

Pascal

```
function DoCanResize(var NewSize: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoCanResize(int NewSize);
```

Description

DoCanResize is called automatically when the user moves the mouse after pressing the mouse button on the splitter. The NewSize parameter indicates the size the associated control will have if the splitter is allowed to perform the resize. DoCanResize returns a boolean value that indicates whether the value specified by NewSize should be applied by redrawing the line (when ResizeStyle (see page 13237) is rsLine) or resizing the associated control (when ResizeStyle (see page 13237) is rsUpdate).

DoCanResize calls CanResize (see page 13220) to generate an OnCanResize (see page 13240) event. If, after calling the event handler, NewSize is less than the value of MinSize (see page 13235), NewSize is changed to zero.

Note: DoCanResize is not called if the value of ResizeStyle (see page 13237) is rsNone when the user presses the mouse button.

68.1.35.4.3 TSplitter.MouseDown Method

Dynamically locates the control that can be resized by the splitter.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

After calling the parent class's method, MouseDown locates the control that it is about to resize. Because this control is located dynamically, an application can delete or add aligned controls beside a splitter at runtime and the splitter will resize whichever control is currently beside it.

See Also

MouseDown (see page 5773)

MouseMove (see page 13222)

MouseUp (see page 13222)

68.1.35.4.4 TSplitter.MouseMove Method

Provides visual feedback of how the neighboring control would be resized.

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

After calling the inherited method, MouseMove computes the new boundaries of the neighboring control and draws a line where the new edge would be. The control is not actually resized until the mouse is released.

See Also

OnCanResize (see page 13240)

ResizeStyle (see page 13237)

CanResize (see page 13220)

MinSize (see page 13235)

MouseMove (see page 5773)

MouseDown (see page 13221)

MouseUp (see page 13222)

68.1.35.4.5 TSplitter.MouseUp Method

Resizes the neighboring control when the user releases the mouse.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

After calling the inherited method, MouseUp resizes the neighboring control that was located when the mouse went down. After the control is resized, MouseUp generates an OnMoved (see page 13240) event.

See Also

MouseUp (see page 5774)

MouseDown (see page 13221)

MouseMove (see page 13222)

68.1.35.4.6 TSplitter.Paint Method

Draws the image of the splitter.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method is called automatically when the splitter receives WM_PAINT messages from its Parent. Paint draws the background color of the splitter, and if Beveled (see page 13228) is true, draws a bevel along the movable edges. After drawing the background and bevel, Paint generates an OnPaint (see page 13241) event.

See Also

Beveled (see page 13228)

Color (see page 5812)

Parent (see page 5832)

OnPaint (see page 13241)

68.1.35.4.7 TSplitter.RequestAlign Method

Changes the cursor when the alignment of the splitter changes.

Pascal

```
procedure RequestAlign; override;
```

C++

```
virtual __fastcall RequestAlign();
```

Description

RequestAlign changes the cursor from crHSplit to crVSplit when the Align (see page 13224) property changes to alBottom or alTop. It changes the cursor from crVSplit to crHSplit when the Align (see page 13224) property changes to any other value.

If the Cursor (see page 13232) property is not crHSplit or crVSplit, RequestAlign has no effect.

Override RequestAlign to suppress the automatic cursor change or to implement a different association between alignment and the splitter's cursor.

See Also

Align (see page 5798)

Cursor (see page 5815)

68.1.35.4.8 TSplitter.StopSizing Method

Cleans up after the splitter has resized its neighboring control.

Pascal

```
procedure StopSizing; dynamic;
```

C++

```
__fastcall StopSizing();
```

Description

This protected procedure is called when the user releases the mouse or hits the Escape key while using the splitter to resize a

control. StopSizing makes sure that the resize line is no longer visible and generates an OnMoved (see page 13240) event.

See Also

OnMoved (see page 13240)

68.1.35.5 TSplitter Properties

The properties of the TSplitter class are listed here.

Public Properties

	Name	Description
	Canvas (see page 13228)	Provides the drawing surface used by the graphic control.

Published Properties

	Name	Description
	Align (see page 13224)	Determines how the control aligns within its container (parent control).
	AutoSnap (see page 13228)	Determines whether neighboring objects are resized to zero when the splitter is used to make them smaller than MinSize (see page 13235).
	Beveled (see page 13228)	Determines whether the splitter looks beveled along the edge that moves.
	Color (see page 13231)	Specifies the background color of the control.
	Constraints (see page 13231)	Specifies the size constraints for the control.
	Cursor (see page 13232)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	MinSize (see page 13235)	Specifies the minimum size, in pixels, of the panes on either side of the splitter.
	ParentColor (see page 13236)	Determines where a control looks for its color information.
	ResizeStyle (see page 13237)	Specifies the effect of moving the splitter.
	Visible (see page 13237)	Determines whether the component appears onscreen.
	Width (see page 13239)	Specifies the horizontal size of the control or form in pixels.

68.1.35.5.1 TSplitter.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls

into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
```

```
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form’s width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
```

```
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

68.1.35.5.2 TSplitter.AutoSnap Property

Determines whether neighboring objects are resized to zero when the splitter is used to make them smaller than MinSize (see page 13235).

Pascal

```
property AutoSnap: Boolean;
```

C++

```
__property Boolean AutoSnap;
```

Description

Set AutoSnap to false to prevent the splitter from resizing neighboring objects to zero when the user tries to make them smaller than MinSize (see page 13235). When AutoSnap is false, such resize attempts are simply blocked, leaving the neighboring object with a size of MinSize (see page 13235). The default value of AutoSnap is true.

See Also

MinSize (see page 13235)

68.1.35.5.3 TSplitter.Beveled Property

Determines whether the splitter looks beveled along the edge that moves.

Pascal

```
property Beveled: Boolean;
```

C++

```
__property Boolean Beveled;
```

Description

Set Beveled to false to remove the beveled edges from the splitter. Set Beveled to true to give the edge a 3D beveled look.

The bevel on a splitter runs in a single direction only: along the edges the user moves to resize neighboring objects. Thus, if the Align (see page 13224) property is alLeft or alRight, the vertical edges of the splitter are beveled. If the Align (see page 13224) property is alTop or alBottom, the horizontal edges of the splitter are beveled. The default value of Beveled is false.

See Also

Align (see page 5798)

68.1.35.5.4 TSplitter.Canvas Property

Provides the drawing surface used by the graphic control.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control.

Since a TGraphicControl instance does not have its own Windows screen object, it obtains its device context from its parent control.

C++ Examples:

```
/*
When you do custom drawing, such as shown in the code for
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   PaintBox1->Canvas->Brush->Color = clRed;
   PaintBox1->Canvas->FillRect(PaintBox1->Canvas->ClipRect);
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}

/*
In order for the drawing to be persistent, you need to also
include an OnPaint event handler that tells the PaintBox how
to redraw itself when it needs to. In the code below, an
ellipse will be drawn everytime the PaintBox renders itself,
but the PaintBox will only paint the client area of the
control red after the Button is clicked. This client area
will only stay red until it is invalidated, but the ellipse
will persist. This behavior is different from the TImage
component, because the TImage component maintains an
internal bitmap that stores this drawing information for
you.
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
   PaintBox1->Canvas->Brush->Color = clBlue;
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TLabel *myLabel;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TLabel")
  {
    myLabel = (TLabel *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myLabel->Canvas->MoveTo(myLabel->ClientRect.Left, myLabel->ClientRect.Top);
    myLabel->Canvas->LineTo(myLabel->ClientRect.Right, myLabel->ClientRect.Bottom);
    myLabel = NULL;
  }
}
/*
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
*/
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PaintBox1->Canvas->Draw(0,0, Image1->Picture->Graphic);
}
```

Delphi Examples:

```
{
When you do custom drawing, such as shown in the code for
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   PaintBox1.Canvas.Brush.Color := clRed;
   PaintBox1.Canvas.FillRect(PaintBox1.Canvas.ClipRect);
   PaintBox1.Canvas.Brush.Color := clBlue;
   PaintBox1.Canvas.Ellipse(0,0,100,50);
end;
{
In order for the drawing to be persistent, you need to also
include an OnPaint event handler that tells the PaintBox how
to redraw itself when it needs to. In the code below, an
ellipse will be drawn everytime the PaintBox renders itself,
but the PaintBox will only paint the client area of the
control red after the Button is clicked. This client area
will only stay red until it is invalidated, but the ellipse
will persist. This behavior is different from the TImage
component, because the TImage component maintains an
internal bitmap that stores this drawing information for
you.
}
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
   PaintBox1.Canvas.Brush.Color := clBlue;
   PaintBox1.Canvas.Ellipse(0,0,100,50);
end;
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TLabel) then
  begin
    myLabel:= (ListBox1.Items.Objects[Index] as TLabel);
    with myLabel.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myLabel.Canvas.MoveTo(Left, Top);
      myLabel.Canvas.LineTo(Right, Bottom);
    end;
  myLabel:= nil;
  end;
{
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  PaintBox1.Canvas.Draw(0,0, Image1.Picture.Graphic);
end;
```

68.1.35.5.5 TSplitter.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

68.1.35.5.6 TSplitter.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

68.1.35.5.7 TSplitter.Cursor Property

Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.

Pascal

```
property Cursor: TCursor;
```

C++

```
__property TCursor Cursor;
```

Description

Change the value of Cursor to provide feedback to the user when the mouse pointer enters the control. The value of Cursor is the index of the cursor in the list of cursors maintained by the global variable, Screen. In addition to the built-in cursors provided by TScreen, applications can add custom cursors to the list.

C++ Examples:

```
/*
This example shows how to add custom cursors to an
application.  The following code makes this cursor available
to the application via the constant crMyCursor, and sets it
as the global cursor to the application.
*/
Graphics::TBitmap *bmpMask, *bmpColor;
TIconInfo *iconInfo;

const crMyCursor = 5;

  void __fastcall TForm1::Button2Click(TObject *Sender)
{
  bmpMask = new Graphics::TBitmap();
```

```
  bmpColor = new Graphics::TBitmap();
  iconInfo = new TIconInfo();

  bmpMask->LoadFromFile("../SquareMask.bmp");
  bmpColor->LoadFromFile("../Square.bmp");

  iconInfo->fIcon = false;
  iconInfo->xHotspot = 15;
  iconInfo->yHotspot = 15;
  iconInfo->hbmMask = bmpMask->Handle;
  iconInfo->hbmColor = bmpColor->Handle;

  Screen->Cursors[crMyCursor] = CreateIconIndirect(iconInfo);

  Screen->Cursor = crMyCursor;

  delete bmpMask;
  delete bmpColor;
  delete iconInfo;
}
```

Delphi Examples:

```
{
This example uses a splitter and two image controls to
create a "slide show" effect.  The example displays an image
(the current "slide"), which can be hidden behind a screen
that looks like a venetian blind. When the blind is pulled
down to completely cover the current slide, the slide changes
to a different image.  When the blind is raised, the new
image appears.
To prepare this example, follow these steps:
1. Place an image control on the form.
Set the picture property to a bitmap of a venetian blind,
Set the Stretch property to True.
Set the Align property to alTop.
2. Add a splitter control to the form.
Set the Align property to alTop.
Set the Cursor property to crVSplit.
Set the OnMoved event handler to the code that appears below.
Set the MinSize property to 1.
3. Add a second image control to the form.
Set the Align property to alClient.
Set the Stretch property to True.
4. Select the form and add the OnCreate and OnDestroy event
   handlers that appear below.
5. Add the global variables below to the unit.
}
var
  CurImage: Integer;
  BMPs: Array[1..5] of TBitmap;

procedure TForm1.Splitter1Moved(Sender: TObject);
begin
  { check if the blind is closed }
  if Image2.Height <= 1 then
  begin
    { the blind is closed, change the current slide }
    CurImage := CurImage + 1;
    if CurImage = 6 then
       CurImage := 1;
    Image2.Picture.Bitmap := BMPs[CurImage];
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    { Initialize the array of slides.
      These must be freed in the OnDestroy event handler. }
    BMPs[1] := TBitmap.Create;
    BMPs[1].LoadFromFile('Factory.bmp');
    BMPs[2] := TBitmap.Create;
    BMPs[2].LoadFromFile('Rhododendron.bmp');
    BMPs[3] := TBitmap.Create;
    BMPs[3].LoadFromFile('littleB_64.bmp');
    BMPs[4] := TBitmap.Create;
    BMPs[4].LoadFromFile('Soap bubbles.bmp');
    BMPs[5] := TBitmap.Create;
    BMPs[5].LoadFromFile('Gone Fishing.bmp');
    CurImage := 1;
    Image2.Picture.Bitmap := BMPs[1];
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I: Integer;
  begin
    { clean up }
    for I := 1 to 5 do
      BMPs[I].Free;
  end;
  {
  This example shows how to add custom cursors to an
  application.  The following code makes this cursor available
  to the application via the constant crMyCursor, and sets it
  as the global cursor to the application.
  }
  var
    bmpMask : TBitmap;
    bmpColor : TBitmap;
    iconInfo : TIconInfo;
  const
    crMyCursor = 5;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    bmpMask := TBitmap.Create;
    bmpColor := TBitmap.Create;

    bmpMask.LoadFromFile('SquareMask.bmp');
    bmpColor.LoadFromFile('Square.bmp');

    with iconInfo do
    begin
      fIcon := false;
      xHotspot := 15;
      yHotspot := 15;
      hbmMask := bmpMask.Handle;
      hbmColor := bmpColor.Handle;
    end;

    Screen.Cursors[crMyCursor] := CreateIconIndirect(iconInfo);

    Screen.Cursor := crMyCursor;

    bmpMask.Free;
    bmpColor.Free;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    DestroyIcon(Screen.Cursors[crMyCursor]);
  end;
```

See Also

DragCursor (see page 5818)

Cursors (see page 14212)

Screen (see page 14385)

ProcessMessages (see page 13691)

68.1.35.5.8 TSplitter.MinSize Property

Specifies the minimum size, in pixels, of the panes on either side of the splitter.

Pascal

```
property MinSize: NaturalNumber;
```

C++

```
__property NaturalNumber MinSize;
```

Description

Set MinSize to provide a minimum size the splitter must leave when resizing its neighboring control. For example, if the Align (see page 13224) property is alLeft or alRight, the splitter cannot resize the regions to its left or right any smaller than MinSize pixels. If the Align (see page 13224) property is alTop or alBottom, the splitter cannot resize the regions above or below it any smaller than MinSize pixels. The default value of MinSize is 30.

Note: Always set MinSize to a value less than half the client width of its parent. When MinSize is half the client width of the splitter's parent, the splitter cannot move because to do so would be to resize one of the panes less than MinSize pixels.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.
```

```
    // Each pane must be at least one quarter of the form's width.
    ps->MinSize = Form1->ClientWidth/4;

    // Finally, create the last pane - a file list box.
    pflb->Parent = Form1;
    pflb->Align = alClient;
    pdlb->FileList = pflb; // Link the file list box to the directory list box.
  }
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form’s width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

See Also

Align (see page 5798)

68.1.35.5.9 TSplitter.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its

own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

68.1.35.5.10 TSplitter.ResizeStyle Property

Specifies the effect of moving the splitter.

Pascal

```
property ResizeStyle: TResizeStyle;
```

C++

```
__property TResizeStyle ResizeStyle;
```

Description

Use ResizeStyle to determine what happens when the user moves the splitter with the mouse.

ResizeStyle can have one of the values in the following table:

Value	Meaning
rsNone	Moving the splitter with the mouse has no effect.
rsLine	A line is drawn indicating the new position of the splitter that would result if the user released the mouse at the current position. The splitter does not actually move and no controls are resized.
rsUpdate	The control against which the splitter is aligned is resized based on the current mouse position, and the splitter is moved to that position.
rsPattern	A pattern is drawn indicating the new position of the splitter that would result if the user released the mouse at the current position. The splitter does not actually move and no controls are resized. This is the default value for ResizeStyle.

See Also

OnCanResize (see page 13240)

OnMoved (see page 13240)

68.1.35.5.11 TSplitter.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
```

```
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

68.1.35.5.12 TSplitter.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

See Also

ClientWidth (see page 5811)

Height (see page 5825)

Left (see page 5830)

68.1.35.6 TSplitter Events

The events of the TSplitter class are listed here.

Published Events

	Name	Description
	OnCanResize (see page 13240)	Occurs when the user tries to move the splitter with the mouse.
	OnMoved (see page 13240)	Occurs after the user has retiled the form using the splitter.
	OnPaint (see page 13241)	Occurs when the splitter needs to draw itself.

68.1.35.6.1 TSplitter.OnCanResize Event

Occurs when the user tries to move the splitter with the mouse.

Pascal

```
property OnCanResize: TCanResizeEvent;
```

C++

```
__property TCanResizeEvent OnCanResize;
```

Description

Write an OnCanResize event handler to conditionally prevent the user from resizing neighboring components using the splitter.

The NewSize parameter indicates the size the associated control will have if the splitter is allowed to perform the resize. This value can be changed if the attempted resize can be partially allowed. The Accept parameter indicates whether the resize should be permitted at all. Set Accept to false to disallow any movement of the splitter.

See Also

OnMoved (see page 13240)

ResizeStyle (see page 13237)

68.1.35.6.2 TSplitter.OnMoved Event

Occurs after the user has retiled the form using the splitter.

Pascal

```
property OnMoved: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMoved;
```

Description

Write an OnMoved event handler to take specific action when the user finishes dragging the edge of the neighboring control using the splitter. For example, after resizing a control on the edge of the form, use an OnMoved event handler to update the layout of the new client area of the form. Use the ClientRect property of the form to determine the new size remaining.

Delphi Examples:

```
{
This example uses a splitter and two image controls to
create a "slide show" effect.  The example displays an image
(the current "slide"), which can be hidden behind a screen
that looks like a venetian blind. When the blind is pulled
down to completely cover the current slide, the slide changes
to a different image.  When the blind is raised, the new
image appears.
To prepare this example, follow these steps:
1. Place an image control on the form.
Set the picture property to a bitmap of a venetian blind,
Set the Stretch property to True.
Set the Align property to alTop.
2. Add a splitter control to the form.
Set the Align property to alTop.
Set the Cursor property to crVSplit.
Set the OnMoved event handler to the code that appears below.
Set the MinSize property to 1.
3. Add a second image control to the form.
Set the Align property to alClient.
```

```
  Set the Stretch property to True.
  4. Select the form and add the OnCreate and OnDestroy event
     handlers that appear below.
  5. Add the global variables below to the unit.
  }
  var
    CurImage: Integer;
    BMPs: Array[1..5] of TBitmap;

  procedure TForm1.Splitter1Moved(Sender: TObject);
  begin
    { check if the blind is closed }
    if Image2.Height <= 1 then
    begin
      { the blind is closed, change the current slide }
      CurImage := CurImage + 1;
      if CurImage = 6 then
         CurImage := 1;
      Image2.Picture.Bitmap := BMPs[CurImage];
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    { Initialize the array of slides.
      These must be freed in the OnDestroy event handler. }
    BMPs[1] := TBitmap.Create;
    BMPs[1].LoadFromFile('Factory.bmp');
    BMPs[2] := TBitmap.Create;
    BMPs[2].LoadFromFile('Rhododendron.bmp');
    BMPs[3] := TBitmap.Create;
    BMPs[3].LoadFromFile('littleB_64.bmp');
    BMPs[4] := TBitmap.Create;
    BMPs[4].LoadFromFile('Soap bubbles.bmp');
    BMPs[5] := TBitmap.Create;
    BMPs[5].LoadFromFile('Gone Fishing.bmp');
    CurImage := 1;
    Image2.Picture.Bitmap := BMPs[1];
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I: Integer;
  begin
    { clean up }
    for I := 1 to 5 do
      BMPs[I].Free;
  end;
```

See Also

ClientRect

68.1.35.6.3 TSplitter.OnPaint Event

Occurs when the splitter needs to draw itself.

Pascal

```
property OnPaint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPaint;
```

Description

Write an OnPaint event handler to customize the appearance of the splitter. For example, use an OnPaint event handler to give a splitter the appearance of the splitters that appear in cool bar objects. Use the Canvas (see page 13228) property as a drawing surface for painting the splitter. Use the ClientRect property to determine the coordinates of the canvas on which to paint the splitter.

When a splitter object receives a WM_PAINT message, it paints its background before generating the OnPaint event. If the Beveled (see page 13228) property is true, the beveled appearance will also be added before the OnPaint event. Most applications set Beveled (see page 13228) to false when using the OnPaint event to customize the appearance of the splitter.

See Also

ClientRect

Canvas (see page 6053)

Beveled (see page 13228)

68.1.36 TTimer Class

TTimer encapsulates the Windows API timer functions.

Class Hierarchy

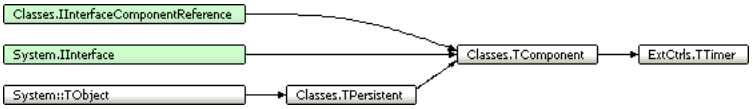

Pascal

```
TTimer = class(TComponent);
```

C++

```
class TTimer : public TComponent;
```

File

ExtCtrls

Description

TTimer is used to simplify calling the Windows API timer functions SetTimer and KillTimer, and to simplify processing the WM_TIMER messages. Use one timer component for each timer in the application.

The execution of the timer occurs through its OnTimer (see page 13251) event. TTimer has an Interval (see page 13249) property that determines how often the timer's OnTimer (see page 13251) event occurs. Interval (see page 13249) corresponds to the parameter for the Windows API SetTimer function.

Warning: Caution:Limitations on the total number of timers system-wide are system dependent.

68.1.36.1 TTimer Members

The following tables list the members exposed by TTimer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Timer (see page 13246)	This is Timer, a member of class TTimer.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.

	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 13245)	Instantiates a timer object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 13245)	Disposes of a timer object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.

V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnTimer (see page 13251)	Occurs when a specified amount of time, determined by the Interval (see page 13249) property, has passed.

Published Properties

	Name	Description
	Enabled (see page 13246)	Controls (see page 5711) whether the timer generates OnTimer (see page 13251) events periodically.
	Interval (see page 13249)	Determines the amount of time, in milliseconds, that passes before the timer component initiates another OnTimer (see page 13251) event.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

68.1.36.2 TTimer.Create Constructor

Instantiates a timer object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TTimer(TComponent * AOwner);
```

Description

Call Create to instantiate a timer at runtime. Timers added to forms or data modules at design time are created automatically.

AOwner specifies the component, typically a form or data module, that is responsible for freeing the timer.

See Also

Owner (see page 2807)

68.1.36.3 TTimer.Destroy Destructor

Disposes of a timer object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTimer();
```

Description

Do not call Destroy directly in an application. Instead, an application should call Free. Free verifies that the timer is not nil before it calls Destroy.

Destroy deactivates the timer by setting Enabled (see page 13246) to False before freeing the resources required by the timer.

See Also

Create (see page 13245)

68.1.36.4 TTimer Methods

The methods of the TTimer class are listed here.

Protected Methods

	Name	Description
	Timer (see page 13246)	This is Timer, a member of class TTimer.

68.1.36.4.1 TTimer.Timer Method

Pascal

```
procedure Timer; dynamic;
```

C++

```
__fastcall Timer();
```

Description

This is Timer, a member of class TTimer.

68.1.36.5 TTimer Properties

The properties of the TTimer class are listed here.

Published Properties

	Name	Description
	Enabled (see page 13246)	Controls (see page 5711) whether the timer generates OnTimer (see page 13251) events periodically.
	Interval (see page 13249)	Determines the amount of time, in milliseconds, that passes before the timer component initiates another OnTimer (see page 13251) event.

68.1.36.5.1 TTimer.Enabled Property

Controls (see page 5711) whether the timer generates OnTimer (see page 13251) events periodically.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to enable or disable the timer. If Enabled is true, the timer responds normally. If Enabled is false, the timer does not generate OnTimer (see page 13251) events. The default is true.

C++ Examples:

```
/*
This example uses a timer on a form. When the application
runs and the user minimizes the application, the timer
starts and the application returns to its normal size when
an OnTimer event occurs.  The timer then shuts down until
the next time the form is minimized.  Be sure to declare the
AppStartTimer method as a public method of TForm1.  Place a
TTimer object in the form and enter Timer1Timer in the OnTimer
event.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnMinimize = AppStartTimer;
  Timer1->Interval = 1000;
}

void __fastcall TForm1::AppStartTimer(TObject *Sender)
{
  Timer1->Enabled = true;
}

//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Application->Restore();
  Timer1->Enabled = false;
}
/*
This code ejects the CD from the CD-ROM player and shuts
down the media player after 10 seconds. For the code to run
correctly, you must have your CD audio device installed
correctly, and the device must have software-ejecting
capabilities.
*/

Word TimeOver;

void __fastcall TForm1::FormClick(TObject *Sender)
{
  MediaPlayer1->DeviceType = dtCDAudio;
  MediaPlayer1->Open();
  MediaPlayer1->Play();
  Timer1->Enabled = true;
  TimeOver = 0;
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  if (TimeOver == 10)
  {
    MediaPlayer1->Eject();
    MediaPlayer1->Close();
    // disable the timer, as we are done
    Timer1->Enabled = false;
  }
```

```
  else
    TimeOver++;
}
```

Delphi Examples:

```
{
This example uses a timer on a form. When the application
runs and the user minimizes the application, the timer
starts and the application returns to its normal size when
an OnTimer event occurs.  The timer then shuts down until
the next time the form is minimized.  Be sure to declare the
AppStartTimer method as a public method of TForm1.  Place a
TTimer object in the form and enter Timer1Timer in the OnTimer
event.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnMinimize := AppStartTimer;
end;

procedure TForm1.AppStartTimer(Sender: TObject);
begin
  Timer1.Enabled := True;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Application.Restore;
  Timer1.Enabled := False;
end;
{
This code ejects the CD from the CD-ROM player and shuts
down the media player after 10 seconds. For the code to run
correctly, you must have your CD audio device installed
correctly, and the device must have software-ejecting
capabilities.
}
var
  TimerOver: Word;

procedure TForm1.FormClick(Sender: TObject);
begin
  MediaPlayer1.DeviceType := dtCDAudio;
  MediaPlayer1.Open;
  MediaPlayer1.Play;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  if TimeOver = 10 then
  begin
    MediaPlayer1.Eject;
    MediaPlayer1.Close;
    { disable the timer, as we are done }
    Timer1.Enabled := False;
  end
  else
    if (MediaPlayer1.Mode = mpOpen) then
     Inc(TimeOver);
end;
```

See Also

OnTimer (see page 13251)

68.1.36.5.2 TTimer.Interval Property

Determines the amount of time, in milliseconds, that passes before the timer component initiates another OnTimer (see page 13251) event.

Pascal

```
property Interval: Cardinal;
```

C++

```
__property unsigned Interval;
```

Description

Interval determines how frequently the OnTimer (see page 13251) event occurs. Each time the specified interval passes, the OnTimer (see page 13251) event occurs.

Use Interval to specify any cardinal value as the interval between OnTimer (see page 13251) events. The default value is 1000 (one second).

Note: A 0 value is valid, however the timer will not call an OnTimer (see page 13251) event for a value of 0.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Timer1->Interval = 100;
  if (ActiveControl)
    ActiveControl->Left = ActiveControl->Left + 1;
}
/*
This code uses a button, a timer and a maskedit on a
form. When the user clicks the button, the form
disappears for the period of time specified in the
Interval property of the timer control, then the form
reappears:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Timer1->Enabled = true;
  Hide();
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Visible = true;
  Timer1->Enabled = false;
}
/*
This example uses a timer on a form. When the application
runs and the user minimizes the application, the timer
starts and the application returns to its normal size when
an OnTimer event occurs.  The timer then shuts down until
the next time the form is minimized.  Be sure to declare the
AppStartTimer method as a public method of TForm1.  Place a
TTimer object in the form and enter Timer1Timer in the OnTimer
```

```
event.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnMinimize = AppStartTimer;
  Timer1->Interval = 1000;
}

void __fastcall TForm1::AppStartTimer(TObject *Sender)
{
  Timer1->Enabled = true;
}

//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Application->Restore();
  Timer1->Enabled = false;
}
```

Delphi Examples:

```
{
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  if ActiveControl <> nil then
    ActiveControl.Left := ActiveControl.Left + 1;
end;
{
This code uses a button, a timer and a maskedit on a
form. When the user clicks the button, the form
disappears for the period of time specified in the
Interval property of the timer control, then the form
reappears:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Timer1.Interval := StrtoInt(MaskEdit1.Text);
  Timer1.Enabled := True;
  Hide;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Visible := True;
  Timer1.Enabled := False;
end;
{
This example uses a timer on a form. When the application
runs and the user minimizes the application, the timer
starts and the application returns to its normal size when
an OnTimer event occurs.  The timer then shuts down until
the next time the form is minimized.  Be sure to declare the
AppStartTimer method as a public method of TForm1.  Place a
TTimer object in the form and enter Timer1Timer in the OnTimer
event.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnMinimize := AppStartTimer;
```

```
end;

procedure TForm1.AppStartTimer(Sender: TObject);
begin
  Timer1.Enabled := True;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Application.Restore;
  Timer1.Enabled := False;
end;
```

See Also

OnTimer (see page 13251)

68.1.36.6 TTimer Events

The events of the TTimer class are listed here.

Published Events

	Name	Description
	OnTimer (see page 13251)	Occurs when a specified amount of time, determined by the Interval (see page 13249) property, has passed.

68.1.36.6.1 TTimer.OnTimer Event

Occurs when a specified amount of time, determined by the Interval (see page 13249) property, has passed.

Pascal

```
property OnTimer: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnTimer;
```

Description

Write an OnTimer event handler to execute an action at regular intervals.

The Interval (see page 13249) property of a timer determines how frequently the OnTimer event occurs. Each time the specified interval passes, the OnTimer event occurs.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Timer1->Interval = 100;
  if (ActiveControl)
    ActiveControl->Left = ActiveControl->Left + 1;
}
```

Delphi Examples:

```
{
```

```
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  if ActiveControl <> nil then
    ActiveControl.Left := ActiveControl.Left + 1;
end;
```

See Also

Interval (see page 13249)

68.1.37 TTrayIcon Class

Creates an icon in the system tray next to the clock.

Class Hierarchy

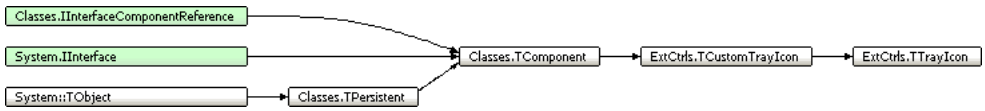

Pascal

```
TTrayIcon = class(TCustomTrayIcon);
```

C++

```
class TTrayIcon : public TCustomTrayIcon;
```

File

ExtCtrls

Description

Use the TTrayIcon component to create an icon in the system tray next to the clock. TTrayIcon has events for mouse clicks, properties for menus and animation of the icon, and methods for showing balloon hints.

68.1.37.1 TTrayIcon Members

The following tables list the members exposed by TTrayIcon.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.

	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoOnAnimate (see page 12569)	This is DoOnAnimate, a member of class TCustomTrayIcon.
	GetAnimateInterval (see page 12569)	This is GetAnimateInterval, a member of class TCustomTrayIcon.
	GetBalloonTimeout (see page 12569)	This is GetBalloonTimeout, a member of class TCustomTrayIcon.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAnimate (see page 12570)	This is SetAnimate, a member of class TCustomTrayIcon.
	SetAnimateInterval (see page 12570)	This is SetAnimateInterval, a member of class TCustomTrayIcon.
	SetBalloonHint (see page 12571)	This is SetBalloonHint, a member of class TCustomTrayIcon.
	SetBalloonTimeout (see page 12571)	This is SetBalloonTimeout, a member of class TCustomTrayIcon.
	SetBalloonTitle (see page 12571)	This is SetBalloonTitle, a member of class TCustomTrayIcon.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHint (see page 12571)	This is SetHint, a member of class TCustomTrayIcon.
	SetIcon (see page 12572)	This is SetIcon, a member of class TCustomTrayIcon.
V	SetIconIndex (see page 12572)	This is SetIconIndex, a member of class TCustomTrayIcon.
	SetIconList (see page 12572)	This is SetIconList, a member of class TCustomTrayIcon.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetVisible (see page 12572)	This is SetVisible, a member of class TCustomTrayIcon.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WindowProc (see page 12573)	This is WindowProc, a member of class TCustomTrayIcon.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Data (see page 12575)	This is Data, a member of class TCustomTrayIcon.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 12568)	Creates and initializes a TCustomTrayIcon (see page 12564) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 12568)	Frees the memory associated with the TCustomTrayIcon (see page 12564) object.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 12570)	This is the overview for the Refresh method overload.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetDefaultIcon (see page 12571)	Sets the tray icon to the default application icon.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	ShowBalloonHint (see page 12573)	Displays the balloon hint.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnAnimate (see page 13260)	Occurs each time the icon changes when the tray icon is animating.
	OnClick (see page 13260)	Occurs when the user clicks the tray icon.
	OnDblClick (see page 13260)	Occurs when the user double clicks the tray icon.
	OnMouseDown (see page 13260)	Occurs when the user presses a mouse button while the mouse pointer is over a control.
	OnMouseMove (see page 13261)	Occurs when the user moves the mouse pointer while the mouse pointer is over the tray icon.
	OnMouseUp (see page 13261)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Published Properties

	Name	Description
	Animate (see page 13256)	Specifies whether the icon in the system tray is static or animated.
	AnimateInterval (see page 13256)	Controls the length of time between image changes for the tray icon image.
	BalloonFlags (see page 13257)	Specifies an icon for the balloon hint.
	BalloonHint (see page 13257)	Specifies the text within a balloon hint.
	BalloonTimeout (see page 13257)	Specifies the length of time that the balloon displays.
	BalloonTitle (see page 13257)	Specifies the text that appears as the balloon title.
	Hint (see page 13258)	Specifies the text that appears when the user moves the mouse over the control.

	Icon (see page 13258)	Specifies the default icon shown in the system tray.
	IconIndex (see page 13258)	This is an index into the Icons image list which is the icon currently being displayed in the system tray.
	Icons (see page 13258)	Specifies the list of images to be displayed for the icon in the system tray.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	PopupMenu (see page 13259)	Identifies the pop-up menu associated with the tray icon.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 13259)	Determines whether the icon appears in the system tray.

68.1.37.2 TTrayIcon Properties

The properties of the TTrayIcon class are listed here.

Published Properties

	Name	Description
	Animate (see page 13256)	Specifies whether the icon in the system tray is static or animated.
	AnimateInterval (see page 13256)	Controls the length of time between image changes for the tray icon image.
	BalloonFlags (see page 13257)	Specifies an icon for the balloon hint.
	BalloonHint (see page 13257)	Specifies the text within a balloon hint.
	BalloonTimeout (see page 13257)	Specifies the length of time that the balloon displays.
	BalloonTitle (see page 13257)	Specifies the text that appears as the balloon title.
	Hint (see page 13258)	Specifies the text that appears when the user moves the mouse over the control.
	Icon (see page 13258)	Specifies the default icon shown in the system tray.
	IconIndex (see page 13258)	This is an index into the Icons image list which is the icon currently being displayed in the system tray.
	Icons (see page 13258)	Specifies the list of images to be displayed for the icon in the system tray.
	PopupMenu (see page 13259)	Identifies the pop-up menu associated with the tray icon.
	Visible (see page 13259)	Determines whether the icon appears in the system tray.

68.1.37.2.1 TTrayIcon.Animate Property

Specifies whether the icon in the system tray is static or animated.

Pascal

```
property Animate: Boolean;
```

C++

```
__property Boolean Animate;
```

Description

When set to True, the icon shown in the system tray displays, in sequence, the images in the image list specified by the Icons property. The images change only if the specified image list contains images. The images change on a rotating basis by the interval set by the AnimateInterval property. When set to False the animation stops.

68.1.37.2.2 TTrayIcon.AnimateInterval Property

Controls the length of time between image changes for the tray icon image.

Pascal

```
property AnimateInterval: Cardinal;
```

C++

```
__property unsigned AnimateInterval;
```

Description

Use AnimateInterval to specify the length of time between image changes in the tray icon. The default interval is 1000 ms (1 second).

68.1.37.2.3 TTrayIcon.BalloonFlags Property

Specifies an icon for the balloon hint.

Pascal

```
property BalloonFlags: TBalloonFlags;
```

C++

```
__property TBalloonFlags BalloonFlags;
```

Description

The following values specify an icon to be displayed in the balloon hint:

- bfNone
- bfInfo
- bfWarning
- bfError

68.1.37.2.4 TTrayIcon.BalloonHint Property

Specifies the text within a balloon hint.

Pascal

```
property BalloonHint: string;
```

C++

```
__property AnsiString BalloonHint;
```

Description

Use BalloonHint to specify the text string that appears when a balloon hint is shown.

68.1.37.2.5 TTrayIcon.BalloonTimeout Property

Specifies the length of time that the balloon displays.

Pascal

```
property BalloonTimeout: Integer;
```

C++

```
__property int BalloonTimeout;
```

Description

Use BalloonTimeout to set the length of time that a balloon displays. By default, this value is set to 3000 ms (3 seconds).

68.1.37.2.6 TTrayIcon.BalloonTitle Property

Specifies the text that appears as the balloon title.

Pascal

```
property BalloonTitle: string;
```

C++

```
__property AnsiString BalloonTitle;
```

Description

Specifies the text that appears as the balloon title.

68.1.37.2.7 TTrayIcon.Hint Property

Specifies the text that appears when the user moves the mouse over the control.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Specifies the text that appears when the user moves the mouse over the control.

68.1.37.2.8 TTrayIcon.Icon Property

Specifies the default icon shown in the system tray.

Pascal

```
property Icon: TIcon;
```

C++

```
__property TIcon Icon;
```

Description

Use the Icon property to set the default icon to be shown in the system tray. If you do not specify a value, the application icon is used.

68.1.37.2.9 TTrayIcon.IconIndex Property

This is an index into the Icons image list which is the icon currently being displayed in the system tray.

Pascal

```
property IconIndex: Integer;
```

C++

```
__property int IconIndex;
```

Description

This is an index into the Icons image list which is the icon currently being displayed in the system tray.

68.1.37.2.10 TTrayIcon.Icons Property

Specifies the list of images to be displayed for the icon in the system tray.

Pascal

```
property Icons: TImageList;
```

C++

```
__property TImageList Icons;
```

Description

Use this property to specify the list of images used for the icon in the system tray. These images are displayed when the Animate property is set to True.

68.1.37.2.11 TTrayIcon.PopupMenu Property

Identifies the pop-up menu associated with the tray icon.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Use PopupMenuOn to specify when the menu is displayed.

68.1.37.2.12 TTrayIcon.Visible Property

Determines whether the icon appears in the system tray.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Determines whether the icon appears in the system tray.

68.1.37.3 TTrayIcon Events

The events of the TTrayIcon class are listed here.

Published Events

	Name	Description
	OnAnimate (see page 13260)	Occurs each time the icon changes when the tray icon is animating.
	OnClick (see page 13260)	Occurs when the user clicks the tray icon.
	OnDblClick (see page 13260)	Occurs when the user double clicks the tray icon.
	OnMouseDown (see page 13260)	Occurs when the user presses a mouse button while the mouse pointer is over a control.
	OnMouseMove (see page 13261)	Occurs when the user moves the mouse pointer while the mouse pointer is over the tray icon.
	OnMouseUp (see page 13261)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

68.1.37.3.1 TTrayIcon.OnAnimate Event

Occurs each time the icon changes when the tray icon is animating.

Pascal

```
property OnAnimate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAnimate;
```

Description

Occurs each time the icon changes when the tray icon is animating.

68.1.37.3.2 TTrayIcon.OnClick Event

Occurs when the user clicks the tray icon.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Occurs when the user clicks the tray icon.

68.1.37.3.3 TTrayIcon.OnDblClick Event

Occurs when the user double clicks the tray icon.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Occurs when the user double clicks the tray icon.

68.1.37.3.4 TTrayIcon.OnMouseDown Event

Occurs when the user presses a mouse button while the mouse pointer is over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Occurs when the user presses a mouse button while the mouse pointer is over a control.

68.1.37.3.5 TTrayIcon.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over the tray icon.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Occurs when the user moves the mouse pointer while the mouse pointer is over the tray icon.

68.1.37.3.6 TTrayIcon.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

68.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	Frame3D (see page 13262)	Draws a three dimensional frame on a canvas.
	NotebookHandlesNeeded (see page 13263)	This is function ExtCtrls.NotebookHandlesNeeded.

68.2.1 ExtCtrls.Frame3D Function

Draws a three dimensional frame on a canvas.

Pascal

```
procedure Frame3D(Canvas: TCanvas; var Rect: TRect; TopColor: TColor; BottomColor: TColor;
Width: Integer);
```

C++

```
Frame3D(TCanvas Canvas, TRect Rect, TColor TopColor, TColor BottomColor, int Width);
```

File

ExtCtrls

Description

Use Frame3D to create three-dimensional effects when drawing directly on a canvas.

The Canvas parameter is the canvas to draw on.

The Rect parameter specifies the position on the canvas where the three-dimensional rectangle should appear. It returns the dimensions of the rectangle that lies inside the frame.

The TopColor and BottomColor parameters indicate the colors that appear on the left and top or the right and bottom of the rectangle respectively.

The Width parameter specifies the width of the frame.

To create a raised effect, the TopColor should be a light color and the BottomColor should be a shadow color. To create a depressed effect, the TopColor should be the shadow color and the BottomColor should be the light color. To create a beveled effect, draw a frame immediately inside another frame with the colors reversed.

See Also

TWinControl

TWinControl

TWincontrol

TWinControl

TWInControl

TBevel

TFrameControl

68.2.2 ExtCtrls.NotebookHandlesNeeded Function

Pascal

```
procedure NotebookHandlesNeeded(Notebook: TNotebook);
```

C++

```
NotebookHandlesNeeded(TNotebook Notebook);
```

File

ExtCtrls

Description

This is function ExtCtrls.NotebookHandlesNeeded.

68.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TBalloonFlags (see page 13264)	A type specified when showing a balloon hint that puts one of the default icons on the balloon message.
	TBandDrawingStyle (see page 13265)	This is record ExtCtrls.TBandDrawingStyle.
	TBandPaintOption (see page 13265)	TBandPaintOption and TBandPaintOptions (see page 13274) specify which parts of a band are painted.
	TBevelShape (see page 13266)	TBevelShape determines the overall shape of a bevel control.
	TBevelStyle (see page 13267)	TBevelStyle indicates whether a bevel appears raised or lowered.
	TColorBoxStyles (see page 13267)	This is record ExtCtrls.TColorBoxStyles.
	TCornerEdge (see page 13268)	The size of the cornering for the edge of the bands in a TControlBar (see page 12316).
	TExpandStyle (see page 13268)	Specifies how a grid panel expands.
	TFlowStyle (see page 13269)	This is record ExtCtrls.TFlowStyle.
	TLabelPosition (see page 13269)	TLabelPosition specifies the position of the label in a labeled edit control.
	TResizeStyle (see page 13270)	This is record ExtCtrls.TResizeStyle.
	TShapeType (see page 13270)	TShapeType determines the shape of a TShape (see page 13172) control.
	TSizeStyle (see page 13271)	This is record ExtCtrls.TSizeStyle.

68.3.1 ExtCtrls.TBalloonFlags Enumeration

A type specified when showing a balloon hint that puts one of the default icons on the balloon message.

Pascal

```
TBalloonFlags = (
  bfNone = NIIF_NONE,
  bfInfo = NIIF_INFO,
  bfWarning = NIIF_WARNING,
  bfError = NIIF_ERROR
);
```

C++

```
enum TBalloonFlags {
  bfNone = NIIF_NONE,
  bfInfo = NIIF_INFO,
  bfWarning = NIIF_WARNING,
  bfError = NIIF_ERROR
};
```

File

ExtCtrls

Description

You can set TBalloonFlags to one of the following values:

- bfNone
- bfInfo

- bfWarning
- bfError

68.3.2 ExtCtrls.TBandDrawingStyle Enumeration

Pascal

```
TBandDrawingStyle = (
  dsNormal,
  dsGradient
);
```

C++

```
enum TBandDrawingStyle {
  dsNormal,
  dsGradient
};
```

File

ExtCtrls

Description

This is record ExtCtrls.TBandDrawingStyle.

68.3.3 ExtCtrls.TBandPaintOption Enumeration

TBandPaintOption and TBandPaintOptions (see page 13274) specify which parts of a band are painted.

Pascal

```
TBandPaintOption = (
  bpoGrabber,
  bpoFrame,
  bpoGradient,
  bpoRoundRect
);
```

C++

```
enum TBandPaintOption {
  bpoGrabber,
  bpoFrame,
  bpoGradient,
  bpoRoundRect
};
```

File

ExtCtrls

Description

TBandPaintOptions (see page 13274) has one or more of the following values:

Value	Meaning
bpoGrabber	The grabber is painted.

bpoFrame	The frame is painted.
bpoGradient	The gradient is painted.
bpoRoundRect	The round rectangle is painted.

68.3.4 ExtCtrls.TBevelShape Enumeration

TBevelShape determines the overall shape of a bevel control.

Pascal

```
TBevelShape = (
  bsBox,
  bsFrame,
  bsTopLine,
  bsBottomLine,
  bsLeftLine,
  bsRightLine,
  bsSpacer
);
```

C++

```
enum TBevelShape {
  bsBox,
  bsFrame,
  bsTopLine,
  bsBottomLine,
  bsLeftLine,
  bsRightLine,
  bsSpacer
};
```

File

ExtCtrls

Description

These are the possible values of the TBevelShape type:

Value	Meaning
bsBox	The entire client area appears raised or lowered.
bsFrame	The client area is outlined by a raised or lowered frame.
bsTopLine	The bevel displays a line at the top of the client area.
bsBottomLine	The bevel displays a line at the bottom of the client area.
bsLeftLine	The bevel displays a line at the left side of the client area.
bsRightLine	The bevel displays a line at the right side of the client area.
bsSpacer	The bevel is an empty space.

68.3.5 ExtCtrls.TBevelStyle Enumeration

TBevelStyle indicates whether a bevel appears raised or lowered.

Pascal

```
TBevelStyle = (
  bsLowered,
  bsRaised
);
```

C++

```
enum TBevelStyle {
  bsLowered,
  bsRaised
};
```

File

ExtCtrls

Description

These are the possible values of the TBevelStyle type:

Value	Meaning
bsLowered	The bevel is lowered.
bsRaised	The bevel is raised.

68.3.6 ExtCtrls.TColorBoxStyles Enumeration

Pascal

```
TColorBoxStyles = (
  cbStandardColors,
  cbExtendedColors,
  cbSystemColors,
  cbIncludeNone,
  cbIncludeDefault,
  cbCustomColor,
  cbPrettyNames,
  cbCustomColors
);
```

C++

```
enum TColorBoxStyles {
  cbStandardColors,
  cbExtendedColors,
  cbSystemColors,
  cbIncludeNone,
  cbIncludeDefault,
  cbCustomColor,
  cbPrettyNames,
  cbCustomColors
};
```

File

ExtCtrls

Description

This is record ExtCtrls.TColorBoxStyles.

68.3.7 ExtCtrls.TCornerEdge Enumeration

The size of the cornering for the edge of the bands in a TControlBar (see page 12316).

Pascal

```
TCornerEdge = (
  ceNone,
  ceSmall,
  ceMedium,
  ceLarge
);
```

C++

```
enum TCornerEdge {
  ceNone,
  ceSmall,
  ceMedium,
  ceLarge
};
```

File

ExtCtrls

Description

This type contains the size of the cornering for the edge of the bands in a TControlBar (see page 12316).

68.3.8 ExtCtrls.TExpandStyle Enumeration

Specifies how a grid panel expands.

Pascal

```
TExpandStyle = (
  emAddRows,
  emAddColumns,
  emFixedSize
);
```

C++

```
enum TExpandStyle {
  emAddRows,
  emAddColumns,
  emFixedSize
};
```

File

ExtCtrls

Description

Use TExpandStyle to specify whether a grid panel adds rows or columns when new controls are added.

68.3.9 ExtCtrls.TFlowStyle Enumeration

Pascal

```
TFlowStyle = (
  fsLeftRightTopBottom,
  fsRightLeftTopBottom,
  fsLeftRightBottomTop,
  fsRightLeftBottomTop,
  fsTopBottomLeftRight,
  fsBottomTopLeftRight,
  fsTopBottomRightLeft,
  fsBottomTopRightLeft
);
```

C++

```
enum TFlowStyle {
  fsLeftRightTopBottom,
  fsRightLeftTopBottom,
  fsLeftRightBottomTop,
  fsRightLeftBottomTop,
  fsTopBottomLeftRight,
  fsBottomTopLeftRight,
  fsTopBottomRightLeft,
  fsBottomTopRightLeft
};
```

File

ExtCtrls

Description

This is record ExtCtrls.TFlowStyle.

68.3.10 ExtCtrls.TLabelPosition Enumeration

TLabelPosition specifies the position of the label in a labeled edit control.

Pascal

```
TLabelPosition = (
  lpAbove,
  lpBelow,
  lpLeft,
  lpRight
);
```

C++

```
enum TLabelPosition {
  lpAbove,
  lpBelow,
  lpLeft,
  lpRight
```

```
};
```

File

ExtCtrls

Description

The following are possible values of TLabelPosition:

Value	Meaning
lpAbove	The label appears above the edit control.
lpBelow	The label appears below the edit control.
lpLeft	The label appears to the left of the edit control.
lpRight	The label appears to the right of the edit control.

68.3.11 ExtCtrls.TResizeStyle Enumeration

Pascal

```
TResizeStyle = (
  rsNone,
  rsLine,
  rsUpdate,
  rsPattern
);
```

C++

```
enum TResizeStyle {
  rsNone,
  rsLine,
  rsUpdate,
  rsPattern
};
```

File

ExtCtrls

Description

This is record ExtCtrls.TResizeStyle.

68.3.12 ExtCtrls.TShapeType Enumeration

TShapeType determines the shape of a TShape (see page 13172) control.

Pascal

```
TShapeType = (
  stRectangle,
  stSquare,
  stRoundRect,
  stRoundSquare,
  stEllipse,
  stCircle
```

```
);
```

C++

```
enum TShapeType {
  stRectangle,
  stSquare,
  stRoundRect,
  stRoundSquare,
  stEllipse,
  stCircle
};
```

File

ExtCtrls

Description

These are the possible values of the TShapeType type:

Value	Meaning
stCircle	The shape is a circle.
stEllipse	The shape is an ellipse.
stRectangle	The shape is a rectangle. (Default)
stRoundRect	The shape is a rectangle with rounded corners.
stRoundSquare	The shape is a square with rounded corners.
stSquare	The shape is a square.

68.3.13 ExtCtrls.TSizeStyle Enumeration

Pascal

```
TSizeStyle = (
  ssAbsolute,
  ssPercent,
  ssAuto
);
```

C++

```
enum TSizeStyle {
  ssAbsolute,
  ssPercent,
  ssAuto
};
```

File

ExtCtrls

Description

This is record ExtCtrls.TSizeStyle.

68.4 Types

The following table lists types in this documentation.

Types

Name	Description
NaturalNumber (see page 13272)	This is type ExtCtrls.NaturalNumber.
TBandDragEvent (see page 13272)	This is type ExtCtrls.TBandDragEvent.
TBandInfoEvent (see page 13273)	This is type ExtCtrls.TBandInfoEvent.
TBandMoveEvent (see page 13273)	This is type ExtCtrls.TBandMoveEvent.
TBandPaintEvent (see page 13273)	This is type ExtCtrls.TBandPaintEvent.
TBandPaintOptions (see page 13274)	This is type ExtCtrls.TBandPaintOptions.
TBeginBandMoveEvent (see page 13274)	This is type ExtCtrls.TBeginBandMoveEvent.
TCanResizeEvent (see page 13274)	This is type ExtCtrls.TCanResizeEvent.
TCellSpan (see page 13275)	This is type ExtCtrls.TCellSpan.
TColorBoxStyle (see page 13275)	TColorBoxStyle and TColorBoxStyles (see page 13267) specify how a color box control displays its selections.
TEndBandMoveEvent (see page 13276)	This is type ExtCtrls.TEndBandMoveEvent.
TGetColorsEvent (see page 13276)	This is type ExtCtrls.TGetColorsEvent.
TLBGetColorsEvent (see page 13276)	This is type ExtCtrls.TLBGetColorsEvent.
TPanelBevel (see page 13277)	TPanelBevel specifies a bevel's cut.
TRowSize (see page 13277)	This is type ExtCtrls.TRowSize.
TSectionEvent (see page 13278)	TSectionEvent is the type of event handlers that respond when a section of a THeader (see page 12744) control is resized.
TSplitterCanResizeEvent (see page 13278)	This is type ExtCtrls.TSplitterCanResizeEvent.

68.4.1 ExtCtrls.NaturalNumber Type

Pascal

```
NaturalNumber = 1..High(Integer);
```

C++

```
1..High(Integer) NaturalNumber;
```

File

ExtCtrls

Description

This is type ExtCtrls.NaturalNumber.

68.4.2 ExtCtrls.TBandDragEvent Type

Pascal

```
TBandDragEvent = procedure (Sender: TObject; Control: TControl; var Drag: Boolean) of object;
```

C++

```
(Sender: TObject; Control: TControl; var Drag: Boolean) ( TBandDragEvent)();
```

File

ExtCtrls

Description

This is type ExtCtrls.TBandDragEvent.

68.4.3 ExtCtrls.TBandInfoEvent Type

Pascal

```
TBandInfoEvent = procedure (Sender: TObject; Control: TControl; var Insets: TRect; var
PreferredSize, RowCount: Integer) of object;
```

C++

```
(Sender: TObject; Control: TControl; var Insets: TRect; var PreferredSize, RowCount: Integer)
( TBandInfoEvent)();
```

File

ExtCtrls

Description

This is type ExtCtrls.TBandInfoEvent.

68.4.4 ExtCtrls.TBandMoveEvent Type

Pascal

```
TBandMoveEvent = procedure (Sender: TObject; Control: TControl; var ARect: TRect) of object;
```

C++

```
(Sender: TObject; Control: TControl; var ARect: TRect) ( TBandMoveEvent)();
```

File

ExtCtrls

Description

This is type ExtCtrls.TBandMoveEvent.

68.4.5 ExtCtrls.TBandPaintEvent Type

Pascal

```
TBandPaintEvent = procedure (Sender: TObject; Control: TControl; Canvas: TCanvas; var ARect:
TRect; var Options: TBandPaintOptions) of object;
```

C++

```
(Sender: TObject; Control: TControl; Canvas: TCanvas; var ARect: TRect; var Options:
TBandPaintOptions) ( TBandPaintEvent)();
```

File

ExtCtrls

Description

This is type ExtCtrls.TBandPaintEvent.

68.4.6 ExtCtrls.TBandPaintOptions Type

Pascal

```
TBandPaintOptions = set of TBandPaintOption;
```

C++

```
TBandPaintOption TBandPaintOptions;
```

File

ExtCtrls

Description

This is type ExtCtrls.TBandPaintOptions.

68.4.7 ExtCtrls.TBeginBandMoveEvent Type

Pascal

```
TBeginBandMoveEvent = procedure (Sender: TObject; Control: TControl; var AllowMove: Boolean)
of object;
```

C++

```
(Sender: TObject; Control: TControl; var AllowMove: Boolean) ( TBeginBandMoveEvent)();
```

File

ExtCtrls

Description

This is type ExtCtrls.TBeginBandMoveEvent.

68.4.8 ExtCtrls.TCanResizeEvent Type

Pascal

```
TCanResizeEvent = TSplitterCanResizeEvent;
```

C++

```
TSplitterCanResizeEvent TCanResizeEvent;
```

File

ExtCtrls

Description

This is type ExtCtrls.TCanResizeEvent.

68.4.9 ExtCtrls.TCellSpan Type

Pascal

```
TCellSpan = 1..MaxInt;
```

C++

```
1..MaxInt TCellSpan;
```

File

ExtCtrls

Description

This is type ExtCtrls.TCellSpan.

68.4.10 ExtCtrls.TColorBoxStyle Type

TColorBoxStyle and TColorBoxStyles (see page 13267) specify how a color box control displays its selections.

Pascal

```
TColorBoxStyle = set of TColorBoxStyles;
```

C++

```
TColorBoxStyles TColorBoxStyle;
```

File

ExtCtrls

Description

TColorBoxStyle is a set of TColorBoxStyles (see page 13267) values. The possible values of TColorBoxStyles (see page 13267) are

Style	Meaning when included
cbStandardColors	The color box lists the first 16 color constants from the Graphics (see page 14387) unit. These are the constants such as clRed or clBlack, that represent fixed RGB values.
cbExtendedColors	The color box includes clMoneyGreen, clSkyBlue, clCream, and clMedGray.
cbSystemColors	The color box includes the symbolic color constants that reflect the values specified in the Windows control panel.
cbIncludeNone	The color box includes clNone. This option only has an effect if Style includes cbSystemColors.
cbIncludeDefault	The color box includes clNone. This option only has an effect if Style includes cbSystemColors.

cbCustomColor	The first entry in the color box represents a custom color. When the user selects the Custom entry, the color box displays the Color selection dialog, where the user can specify a custom color.
cbPrettyNames	The color box lists the names as words rather than as the color constant strings. (for example 'Black' rather than 'clBlack').

68.4.11 ExtCtrls.TEndBandMoveEvent Type

Pascal

```
TEndBandMoveEvent = procedure (Sender: TObject; Control: TControl) of object;
```

C++

```
(Sender: TObject; Control: TControl) ( TEndBandMoveEvent)();
```

File

ExtCtrls

Description

This is type ExtCtrls.TEndBandMoveEvent.

68.4.12 ExtCtrls.TGetColorsEvent Type

Pascal

```
TGetColorsEvent = procedure (Sender: TCustomColorBox; Items: TStrings) of object;
```

C++

```
(Sender: TCustomColorBox; Items: TStrings) ( TGetColorsEvent)();
```

File

ExtCtrls

Description

This is type ExtCtrls.TGetColorsEvent.

68.4.13 ExtCtrls.TLBGetColorsEvent Type

Pascal

```
TLBGetColorsEvent = procedure (Sender: TCustomColorListBox; Items: TStrings) of object;
```

C++

```
(Sender: TCustomColorListBox; Items: TStrings) ( TLBGetColorsEvent)();
```

File

ExtCtrls

Description

This is type ExtCtrls.TLBGetColorsEvent.

68.4.14 ExtCtrls.TPanelBevel Type

TPanelBevel specifies a bevel's cut.

Pascal

```
TPanelBevel = TBevelCut;
```

C++

```
TBevelCut TPanelBevel;
```

File

ExtCtrls

Description

TPanelBevel can have one of the following values:

Value	Meaning
bvNone	The bevel does not exist.
bvLowered	The bevel appears lowered.
bvRaised	The bevel appears raised.
bvSpace	The bevel appears as a space if its kind is not bkTile. Otherwise, the bevel appears raised.

See Also

TWinControl

TWinControl

68.4.15 ExtCtrls.TRowSize Type

Pascal

```
TRowSize = 1..MaxInt;
```

C++

```
1..MaxInt TRowSize;
```

File

ExtCtrls

Description

This is type ExtCtrls.TRowSize.

68.4.16 ExtCtrls.TSectionEvent Type

TSectionEvent is the type of event handlers that respond when a section of a THeader (see page 12744) control is resized.

Pascal

```
TSectionEvent = procedure (Sender: TObject; ASection, AWidth: Integer) of object;
```

C++

```
(Sender: TObject; ASection, AWidth: Integer) ( TSectionEvent)();
```

File

ExtCtrls

Description

Sender is the header control.

ASection is the index of the section that was resized.

AWidth is the width of the section.

68.4.17 ExtCtrls.TSplitterCanResizeEvent Type

Pascal

```
TSplitterCanResizeEvent = procedure (Sender: TObject; var NewSize: Integer; var Accept:
Boolean) of object;
```

C++

```
(Sender: TObject; var NewSize: Integer; var Accept: Boolean) ( TSplitterCanResizeEvent)();
```

File

ExtCtrls

Description

This is type ExtCtrls.TSplitterCanResizeEvent.

68.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
NoColorSelected (see page 13279)	This is constant ExtCtrls.NoColorSelected.
WM_SYSTEM_TRAY_MESSAGE (see page 13279)	This is constant ExtCtrls.WM_SYSTEM_TRAY_MESSAGE.

68.5.1 ExtCtrls.NoColorSelected Constant

Pascal

```
NoColorSelected = TColor($FF000000);
```

C++

```
NoColorSelected = TColor($FF000000);
```

File

ExtCtrls

Description

This is constant ExtCtrls.NoColorSelected.

68.5.2 ExtCtrls.WM_SYSTEM_TRAY_MESSAGE Constant

Pascal

```
WM_SYSTEM_TRAY_MESSAGE = WM_USER + 1;
```

C++

```
WM_SYSTEM_TRAY_MESSAGE = WM_USER + 1;
```

File

ExtCtrls

Description

This is constant ExtCtrls.WM_SYSTEM_TRAY_MESSAGE.

69 ExtDlgs Namespace

This is namespace ExtDlgs.

Topics

Name	Description
Classes (see page 13282)	The following table lists classes in this documentation.
Constants (see page 13310)	The following table lists constants in this documentation.

Classes

	Name	Description
	TOpenPictureDialog (see page 13282)	TOpenPictureDialog displays a graphics-file selection dialog.
	TOpenTextFileDialog (see page 13292)	This is class ExtDlgs.TOpenTextFileDialog.
	TSavePictureDialog (see page 13300)	TSavePictureDialog displays a "Save As" dialog for saving graphics files.
	TSaveTextFileDialog (see page 13305)	This is class ExtDlgs.TSaveTextFileDialog.

Constants

Name	Description
DefaultEncodingNames (see page 13310)	This is constant ExtDlgs.DefaultEncodingNames.

69.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TOpenPictureDialog (see page 13282)	TOpenPictureDialog displays a graphics-file selection dialog.
	TOpenTextFileDialog (see page 13292)	This is class ExtDlgs.TOpenTextFileDialog.
	TSavePictureDialog (see page 13300)	TSavePictureDialog displays a "Save As" dialog for saving graphics files.
	TSaveTextFileDialog (see page 13305)	This is class ExtDlgs.TSaveTextFileDialog.

69.1.1 TOpenPictureDialog Class

TOpenPictureDialog displays a graphics-file selection dialog.

Class Hierarchy

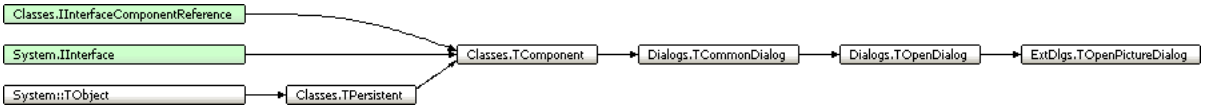

Pascal

```
TOpenPictureDialog = class(TOpenDialog);
```

C++

```
class TOpenPictureDialog : public TOpenDialog;
```

File

ExtDlgs

Description

TOpenPictureDialog displays a modal Windows dialog box for selecting and opening graphics files. This component is just like TOpenDialog, except that it includes a rectangular preview region. If the selected image can be read by TPicture, it is displayed in the preview region; supported file types include bitmap (.BMP), icon (.ICO), Windows metafile (.WMF), and enhanced Windows metafile (.EMF). If the selected image cannot be displayed, "(None)" appears in the preview region.

If the user selects a file of an unrecognized type, TPicture raises an EInvalidGraphic exception.

See Also

EInvalidGraphic (see page 14395)

TOpenDialog (see page 11407)

TPicture (see page 14584)

TSavePictureDialog (see page 13300)

GraphicFilter

69.1.1.1 TOpenPictureDialog Members

The following tables list the members exposed by TOpenPictureDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanClose (see page 11413)	This is CanClose, a member of class TOpenDialog.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 11413)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoCanClose (see page 11413)	This is DoCanClose, a member of class TOpenDialog.
V	DoClose (see page 13287)	Generates an OnClose event.
	DoExecute (see page 11413)	This is the overview for the DoExecute method overload.
	DoFolderChange (see page 11414)	This is DoFolderChange, a member of class TOpenDialog.
	DoIncludeItem (see page 11414)	This is DoIncludeItem, a member of class TOpenDialog.
V	DoSelectionChange (see page 13287)	This is DoSelectionChange, a member of class TOpenPictureDialog.
V	DoShow (see page 13287)	Generates an OnShow event.
	DoTypeChange (see page 11415)	This is DoTypeChange, a member of class TOpenDialog.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetFileNames (see page 11417)	This is GetFileNames, a member of class TOpenDialog.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetStaticRect (see page 11417)	Returns the coordinates of the reserved area of the dialog box.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	MessageHook (see page 11284)	Provides special Windows message processing for the dialog.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	PreviewClick (see page 13288)	This is PreviewClick, a member of class TOpenPictureDialog.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	TaskModalDialog (see page 11284)	Ensures that Windows messages are properly routed when the dialog executes.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WndProc (see page 11285)	Responds to Windows messages sent to the dialog.
V	WndProc (see page 11420)	Responds to Windows messages sent to the dialog.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ImageCtrl (see page 13292)	This is ImageCtrl, a member of class TOpenPictureDialog.
R	PictureLabel (see page 13292)	This is PictureLabel, a member of class TOpenPictureDialog.
	Template (see page 11287)	Points to a custom template for the dialog.
	TemplateModule (see page 11287)	This is TemplateModule, a member of class TCommonDialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 13286)	Creates and initializes a TOpenPictureDialog (see page 13282) instance.
V	DefaultHandler (see page 11282)	Sends unhandled messages to Windows for processing.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 11281)	Destroys the TCommonDialog (see page 11277) instance and frees its memory.
V	Destroy (see page 11412)	Destroys the TOpenDialog (see page 11407) and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.

V	Execute (see page 13288)	Displays the file-selection dialog.
V	Execute (see page 11283)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	FileEditStyle (see page 11423)	Determines the style of the file-selection dialog. (Obsolete.)
R	Files (see page 11429)	List of selected file names.
R	Handle (see page 11286)	Specifies the window handle of the dialog.

	HistoryList (see page 11436)	Maintains a list of previously selected files. (Obsolete.)
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCanClose (see page 11439)	Occurs when the user tries to close the dialog without canceling.
	OnClose (see page 11287)	Occurs when the dialog closes.
	OnFolderChange (see page 11440)	Occurs when a directory is opened or closed from the dialog.
	OnIncludeItem (see page 11440)	Occurs before the dialog adds a file to the file list box.
	OnSelectionChange (see page 11441)	Occurs when file names displayed in the dialog are changed.
	OnShow (see page 11288)	Occurs when the dialog opens.
	OnTypeChange (see page 11441)	Occurs when the file types displayed in the dialog are changed.

Published Properties

	Name	Description
	Ctl3D (see page 11285)	Determines whether the dialog has a three-dimensional look.
	DefaultExt (see page 11421)	Specifies a default file extension.
	FileName (see page 11424)	Indicates the name and directory path of the last file selected.
	Filter (see page 13289)	Determines the file masks (filters) available in the dialog.
	FilterIndex (see page 11434)	Determines which filter is selected by default when the dialog opens.
	HelpContext (see page 11286)	Specifies the context number for online Help.
	InitialDir (see page 11436)	Determines the current directory when the dialog opens.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 11436)	Determines the appearance and behavior of the file-selection dialog.
	OptionsEx (see page 11438)	Augments the Options (see page 11436) property with additional flags that determine the appearance and behavior of the file-selection dialog.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Title (see page 11438)	Specifies the text in the dialog's title bar.

69.1.1.2 TOpenPictureDialog.Create Constructor

Creates and initializes a TOpenPictureDialog (see page 13282) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TOpenPictureDialog(TComponent * AOwner);
```

Description

The Create method generates a TOpenPictureDialog (see page 13282) instance, but the dialog does not appear at runtime until the Execute (see page 13288) method is called. Create initializes the Filter (see page 13289) property by defining a filter for each file type that TPicture supports.

See Also

Execute (see page 13288)

Filter (see page 11431)

TPicture (see page 14584)

69.1.1.3 TOpenPictureDialog Methods

The methods of the TOpenPictureDialog class are listed here.

Protected Methods

	Name	Description
	DoClose (see page 13287)	Generates an OnClose event.
	DoSelectionChange (see page 13287)	This is DoSelectionChange, a member of class TOpenPictureDialog.
	DoShow (see page 13287)	Generates an OnShow event.
	PreviewClick (see page 13288)	This is PreviewClick, a member of class TOpenPictureDialog.

Public Methods

	Name	Description
	Execute (see page 13288)	Displays the file-selection dialog.

69.1.1.3.1 TOpenPictureDialog.DoClose Method

Generates an OnClose event.

Pascal

```
procedure DoClose; override;
```

C++

```
virtual __fastcall DoClose();
```

Description

DoClose is called automatically when the user closes the dialog. Override this method to provide additional processing other than calling the OnClose event handler. For example, to perform shutdown operations that use protected methods of the dialog class, override DoClose rather than writing an OnClose event handler (which can access only public methods).

See Also

OnClose (see page 11287)

69.1.1.3.2 TOpenPictureDialog.DoSelectionChange Method

Pascal

```
procedure DoSelectionChange; override;
```

C++

```
virtual __fastcall DoSelectionChange();
```

Description

This is DoSelectionChange, a member of class TOpenPictureDialog.

69.1.1.3.3 TOpenPictureDialog.DoShow Method

Generates an OnShow event.

Pascal

```
procedure DoShow; override;
```

C++

```
virtual __fastcall DoShow();
```

Description

DoShow is called automatically when the dialog is first displayed. Override this method to provide additional processing other than calling the OnShow event handler. For example, to perform initializations that use protected methods of the dialog class, override DoShow rather than writing an OnShow event handler (which can access only public methods).

See Also

OnShow (see page 11288)

69.1.1.3.4 TOpenPictureDialog.Execute Method

Displays the file-selection dialog.

Pascal

```
function Execute(ParentWnd: HWND): Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute(HWND ParentWnd);
```

Description

Execute opens the file-selection dialog, returning true when the user selects a file and clicks Open. If the user clicks Cancel, Execute returns false.

69.1.1.3.5 TOpenPictureDialog.PreviewClick Method

Pascal

```
procedure PreviewClick(Sender: TObject); virtual;
```

C++

```
virtual __fastcall PreviewClick(TObject * Sender);
```

Description

This is PreviewClick, a member of class TOpenPictureDialog.

69.1.1.4 TOpenPictureDialog Properties

The properties of the TOpenPictureDialog class are listed here.

Protected Properties

	Name	Description
R	ImageCtrl (see page 13292)	This is ImageCtrl, a member of class TOpenPictureDialog.
R	PictureLabel (see page 13292)	This is PictureLabel, a member of class TOpenPictureDialog.

Published Properties

	Name	Description
	Filter (see page 13289)	Determines the file masks (filters) available in the dialog.

69.1.1.4.1 TOpenPictureDialog.Filter Property

Determines the file masks (filters) available in the dialog.

Pascal

```
property Filter: string;
```

C++

```
__property AnsiString Filter;
```

Description

The file-selection dialog includes a drop-down list of file types under the edit box. When the user picks a file type from the list, only files of the selected type are displayed in the dialog.

To configure file masks at design time, click on the ellipsis marks (...) to the right of the Filter property in the Object Inspector. This opens the Filter editor. In the left column of the Filter editor, under Filter Name, type a brief description of each file type that will be available at runtime. In the right column, under Filter, type the file mask corresponding to each description. For example, the description "Text files" might appear to the left of the mask "*.txt", and the description "Pascal source files" might appear to the left of the mask "*.pas". Since the description appears in the drop-down list at runtime, it is often helpful to show the mask explicitly in the description (for example, "Text files (*.txt)").

To create file masks in program code, assign a value to the Filter property that consists of a description and a mask separated by a vertical bar (pipe) character. Do not include spaces around the vertical bar. For example,

```
OpenDialog1.Filter := 'Text files (*.txt)|*.TXT';
OpenDialog1->Filter = "Text files (*.txt)|*.TXT";
```

Multiple filters should be separated by vertical bars. For example,

```
OpenDialog1.Filter := 'Text files (*.txt)|*.TXT|Pascal files (*.pas)|*.PAS';
OpenDialog1->Filter = "Text files (*.txt)|*.TXT|Pascal files (*.pas)|*.PAS";
```

To include multiple masks in a single filter, separate the masks with semicolons. This works both in the Object Inspector and in program code. For example,

```
OpenDialog1.Filter := 'Pascal files|*.PAS;*.DPK;*.DPR';
OpenDialog1->Filter = "Pascal files|*.PAS;*.DPK;*.DPR";
```

If no value is assigned to Filter, the dialog displays all file types.

C++ Examples:

```
/*
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString NewFileName, OldFileName;
  TFileStream *NewFile, *OldFile;
  SavePictureDialog1->DefaultExt = GraphicExtension(__classid(Graphics::TBitmap));
  SavePictureDialog1->Filter = GraphicFilter(__classid(Graphics::TBitmap));
  OldFileName = "factory.bmp";
  if (SavePictureDialog1->Execute())
  {
    NewFileName = SavePictureDialog1->FileName;
    OldFile = new TFileStream(OldFileName, (fmOpenRead | fmShareDenyWrite));
    try
    {
```

```
        NewFile = new TFileStream(NewFileName, (fmCreate | fmShareDenyRead));
        try
        {
          NewFile->CopyFrom(OldFile, OldFile->Size);
        }
        __finally
        {
          FreeAndNil(&NewFile);
        };
      }
      __finally
      {
        FreeAndNil(&OldFile);
      };
    };
  }

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    Image1->Picture->LoadFromFile("../factory.bmp");
  }
  /*
  The following code allows the user to use a dialog box to
  redefine the icon for the application at run time. When the
  user clicks Button1, OpenDialog1 executes and the user
  specifies an icon filename. The Icon is then assigned to the
  application.  At runtime, click Button1 to select the icon,
  then minimize the application to see the icon.
  */
  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    OpenPictureDialog1->DefaultExt = GraphicExtension(__classid(TIcon));
    OpenPictureDialog1->FileName = GraphicFileMask(__classid(TIcon));
    OpenPictureDialog1->Filter = GraphicFilter(__classid(TIcon));
    OpenPictureDialog1->Options.Clear();
    OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
    while (true)
    {
      if (OpenPictureDialog1->Execute())
      {
        if (!OpenPictureDialog1->Options.Contains(ofExtensionDifferent))
        {
          Application->Icon->LoadFromFile(OpenPictureDialog1->FileName);
          break;
        }
        else // reset Options to remove ofExtensionDifferent
        {
          OpenPictureDialog1->Options.Clear();
          OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
        }
      }
      else // user cancelled
        break;
    }
  }
```

Delphi Examples:

```
  {
  This code displays a Save Picture dialog box with the
  TBitmap default extension that is added automatically to
  files that are given no extension. Confirm that the
  bitmap file has been saved under the filename and path
  specified.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var NewFileName, OldFileName: string;
```

```
    NewFile, OldFile: TFileStream;
begin
  SavePictureDialog1.DefaultExt := GraphicExtension(TBitmap);
  SavePictureDialog1.Filter := GraphicFilter(TBitmap);
  OldFileName := 'factory.bmp';
  if SavePictureDialog1.Execute then
  begin
    NewFileName := SavePictureDialog1.FileName;
    OldFile := TFileStream.Create(
      OldFileName, fmOpenRead or fmShareDenyWrite);
    try
      NewFile := TFileStream.Create(
        NewFileName, fmCreate or fmShareDenyRead);
      try
        NewFile.CopyFrom(OldFile, OldFile.Size);
      finally
        FreeAndNil(NewFile);
      end;
    finally
      FreeAndNil(OldFile);
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Picture.LoadFromFile('factory.bmp');
end;
{
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
}
procedure TForm1.Button1Click(Sender: TObject);
var
Done: Boolean;
filenamestring : String;
begin
  OpenPictureDialog1.DefaultExt := GraphicExtension(TIcon);
  filenamestring := GraphicFileMask(TIcon);
  OpenPictureDialog1.FileName := filenamestring;
  OpenPictureDialog1.Filter := GraphicFilter(TIcon);
  OpenPictureDialog1.Options := [ ofFileMustExist, ofHideReadOnly, ofNoChangeDir ];
  Done := False;
  while not Done do
  begin
  if OpenPictureDialog1.Execute then
    begin
    if not (ofExtensionDifferent in OpenPictureDialog1.Options) then
      begin
      Application.Icon.LoadFromFile(OpenPictureDialog1.FileName);
      Done := True;
      end
    else
//      OpenPictureDialog1.Options := OpenPictureDialog1.Options - ofExtensionDifferent;
    end
  else { User cancelled }
    Done := True;
  end;
end;
```

See Also

FilterIndex (see page 11434)

GraphicFilter (see page 14624)

69.1.1.4.2 TOpenPictureDialog.ImageCtrl Property

Pascal

```
property ImageCtrl: TImage;
```

C++

```
__property TImage ImageCtrl;
```

Description

This is ImageCtrl, a member of class TOpenPictureDialog.

69.1.1.4.3 TOpenPictureDialog.PictureLabel Property

Pascal

```
property PictureLabel: TLabel;
```

C++

```
__property TLabel PictureLabel;
```

Description

This is PictureLabel, a member of class TOpenPictureDialog.

69.1.2 TOpenTextFileDialog Class

Class Hierarchy

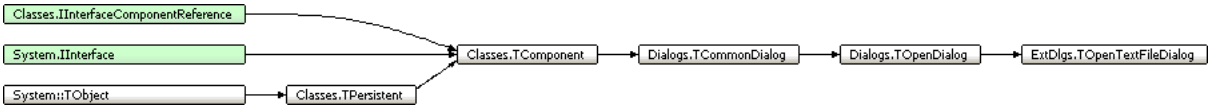

Pascal

```
TOpenTextFileDialog = class(TOpenDialog);
```

C++

```
class TOpenTextFileDialog : public TOpenDialog;
```

File

ExtDlgs

Description

This is class ExtDlgs.TOpenTextFileDialog.

69.1.2.1 TOpenTextFileDialog Members

The following tables list the members exposed by TOpenTextFileDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanClose (see page 11413)	This is CanClose, a member of class TOpenDialog.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 11413)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoCanClose (see page 11413)	This is DoCanClose, a member of class TOpenDialog.
V	DoClose (see page 13297)	Generates an OnClose event.
	DoExecute (see page 11413)	This is the overview for the DoExecute method overload.
	DoFolderChange (see page 11414)	This is DoFolderChange, a member of class TOpenDialog.
	DoIncludeItem (see page 11414)	This is DoIncludeItem, a member of class TOpenDialog.
	DoSelectionChange (see page 11415)	This is DoSelectionChange, a member of class TOpenDialog.
V	DoShow (see page 13297)	Generates an OnShow event.
	DoTypeChange (see page 11415)	This is DoTypeChange, a member of class TOpenDialog.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetFileNames (see page 11417)	This is GetFileNames, a member of class TOpenDialog.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetStaticRect (see page 11417)	Returns the coordinates of the reserved area of the dialog box.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	MessageHook (see page 11284)	Provides special Windows message processing for the dialog.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	TaskModalDialog (see page 11284)	Ensures that Windows messages are properly routed when the dialog executes.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WndProc (see page 11285)	Responds to Windows messages sent to the dialog.
V	WndProc (see page 11420)	Responds to Windows messages sent to the dialog.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Template (see page 11287)	Points to a custom template for the dialog.
	TemplateModule (see page 11287)	This is TemplateModule, a member of class TCommonDialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 13296)	Creates and initializes a TOpenDialog instance.
V	DefaultHandler (see page 11282)	Sends unhandled messages to Windows for processing.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 13296)	Destroys the TOpenDialog and frees its memory.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Execute (see page 13298)	Displays the file-selection dialog.
V	Execute (see page 11283)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	FileEditStyle (see page 11423)	Determines the style of the file-selection dialog. (Obsolete.)
R	Files (see page 11429)	List of selected file names.
R	Handle (see page 11286)	Specifies the window handle of the dialog.
	HistoryList (see page 11436)	Maintains a list of previously selected files. (Obsolete.)
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCanClose (see page 11439)	Occurs when the user tries to close the dialog without canceling.
	OnClose (see page 11287)	Occurs when the dialog closes.
	OnFolderChange (see page 11440)	Occurs when a directory is opened or closed from the dialog.
	OnIncludeItem (see page 11440)	Occurs before the dialog adds a file to the file list box.
	OnSelectionChange (see page 11441)	Occurs when file names displayed in the dialog are changed.
	OnShow (see page 11288)	Occurs when the dialog opens.

	OnTypeChange (see page 11441)	Occurs when the file types displayed in the dialog are changed.

Published Properties

	Name	Description
	Ctl3D (see page 11285)	Determines whether the dialog has a three-dimensional look.
	DefaultExt (see page 11421)	Specifies a default file extension.
	EncodingIndex (see page 13300)	This is EncodingIndex, a member of class TOpenTextFileDialog.
	Encodings (see page 13300)	This is Encodings, a member of class TOpenTextFileDialog.
	FileName (see page 11424)	Indicates the name and directory path of the last file selected.
	Filter (see page 11431)	Determines the file masks (filters) available in the dialog.
	FilterIndex (see page 11434)	Determines which filter is selected by default when the dialog opens.
	HelpContext (see page 11286)	Specifies the context number for online Help.
	InitialDir (see page 11436)	Determines the current directory when the dialog opens.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 11436)	Determines the appearance and behavior of the file-selection dialog.
	OptionsEx (see page 11438)	Augments the Options (see page 11436) property with additional flags that determine the appearance and behavior of the file-selection dialog.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Title (see page 11438)	Specifies the text in the dialog's title bar.

69.1.2.2 TOpenTextFileDialog.Create Constructor

Creates and initializes a TOpenDialog instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TOpenTextFileDialog(TComponent * AOwner);
```

Description

Create generates a TOpenDialog instance, but the new dialog does not appear on the form at runtime until the Execute method is called.

AOwner is the component that is responsible for freeing the TOpenDialog instance. It becomes the value of the Owner property.

See Also

Execute (see page 11415)

Owner (see page 2807)

Destroy (see page 11412)

69.1.2.3 TOpenTextFileDialog.Destroy Destructor

Destroys the TOpenDialog and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TOpenTextFileDialog();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free verifies that the TOpenDialog reference is not nil before calling Destroy.

See Also

Free (see page 22300)

Create (see page 11411)

69.1.2.4 TOpenTextFileDialog Methods

The methods of the TOpenTextFileDialog class are listed here.

Protected Methods

	Name	Description
	DoClose (see page 13297)	Generates an OnClose event.
	DoShow (see page 13297)	Generates an OnShow event.

Public Methods

	Name	Description
	Execute (see page 13298)	Displays the file-selection dialog.

69.1.2.4.1 TOpenTextFileDialog.DoClose Method

Generates an OnClose event.

Pascal

```
procedure DoClose; override;
```

C++

```
virtual __fastcall DoClose();
```

Description

DoClose is called automatically when the user closes the dialog. Override this method to provide additional processing other than calling the OnClose event handler. For example, to perform shutdown operations that use protected methods of the dialog class, override DoClose rather than writing an OnClose event handler (which can access only public methods).

See Also

OnClose (see page 11287)

69.1.2.4.2 TOpenTextFileDialog.DoShow Method

Generates an OnShow event.

Pascal

```
procedure DoShow; override;
```

C++

```
virtual __fastcall DoShow();
```

Description

DoShow is called automatically when the dialog is first displayed. Override this method to provide additional processing other

than calling the OnShow event handler. For example, to perform initializations that use protected methods of the dialog class, override DoShow rather than writing an OnShow event handler (which can access only public methods).

See Also

OnShow (see page 11288)

69.1.2.4.3 TOpenTextFileDialog.Execute Method

Displays the file-selection dialog.

Pascal

```
function Execute(ParentWnd: HWND): Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute(HWND ParentWnd);
```

Description

Execute opens the file-selection dialog, returning true when the user selects a file and clicks Open. If the user clicks Cancel, Execute returns false.

```
if OpenDialog1.Execute then
  Memo1.Lines.LoadFromFile(OpenDialog1.FileName)
else
  Memo1.Lines.Clear;
if (OpenDialog1->Execute())
  Memo1->Lines->LoadFromFile(OpenDialog1->FileName);
else
  Memo1->Clear();
```

C++ Examples:

```
/*
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenPictureDialog1->DefaultExt = GraphicExtension(__classid(TIcon));
  OpenPictureDialog1->FileName = GraphicFileMask(__classid(TIcon));
  OpenPictureDialog1->Filter = GraphicFilter(__classid(TIcon));
  OpenPictureDialog1->Options.Clear();
  OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
  while (true)
  {
    if (OpenPictureDialog1->Execute())
    {
      if (!OpenPictureDialog1->Options.Contains(ofExtensionDifferent))
      {
        Application->Icon->LoadFromFile(OpenPictureDialog1->FileName);
        break;
      }
      else // reset Options to remove ofExtensionDifferent
      {
        OpenPictureDialog1->Options.Clear();
        OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
      }
    }
    else // user cancelled
      break;
```

```
  }
}
```

Delphi Examples:

```
{
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
}
procedure TForm1.Button1Click(Sender: TObject);
var
Done: Boolean;
filenamestring : String;
begin
  OpenPictureDialog1.DefaultExt := GraphicExtension(TIcon);
  filenamestring := GraphicFileMask(TIcon);
  OpenPictureDialog1.FileName := filenamestring;
  OpenPictureDialog1.Filter := GraphicFilter(TIcon);
  OpenPictureDialog1.Options := [ ofFileMustExist, ofHideReadOnly, ofNoChangeDir ];
  Done := False;
  while not Done do
  begin
  if OpenPictureDialog1.Execute then
    begin
    if not (ofExtensionDifferent in OpenPictureDialog1.Options) then
      begin
      Application.Icon.LoadFromFile(OpenPictureDialog1.FileName);
      Done := True;
      end
    else
//      OpenPictureDialog1.Options := OpenPictureDialog1.Options - ofExtensionDifferent;
    end
  else { User cancelled }
    Done := True;
  end;
end;
{
Use a TOpenDialog to select a file.  Place the first line of
the file into a TEdit.
}
procedure TForm1.Edit1Click(Sender: TObject);
var
  F: TextFile;
  S: string;
begin
  if OpenDialog1.Execute then            { Display Open dialog box }
  begin
    AssignFile(F, OpenDialog1.FileName); { File selected in dialog }
    Reset(F);
    Readln(F, S);                        { Read first line of file }
    Edit1.Text := S;                     { Put string in a TEdit control }
    CloseFile(F);
  end;
end;
```

See Also

Files (see page 11429)

FileName (see page 11424)

69.1.2.5 TOpenTextFileDialog Properties

The properties of the TOpenTextFileDialog class are listed here.

Published Properties

	Name	Description
	EncodingIndex (see page 13300)	This is EncodingIndex, a member of class TOpenTextFileDialog.
	Encodings (see page 13300)	This is Encodings, a member of class TOpenTextFileDialog.

69.1.2.5.1 TOpenTextFileDialog.EncodingIndex Property

Pascal

```
property EncodingIndex: Integer;
```

C++

```
__property int EncodingIndex;
```

Description

This is EncodingIndex, a member of class TOpenTextFileDialog.

69.1.2.5.2 TOpenTextFileDialog.Encodings Property

Pascal

```
property Encodings: TStrings;
```

C++

```
__property TStrings Encodings;
```

Description

This is Encodings, a member of class TOpenTextFileDialog.

69.1.3 TSavePictureDialog Class

TSavePictureDialog displays a "Save As" dialog for saving graphics files.

Class Hierarchy

Pascal

```
TSavePictureDialog = class(TOpenPictureDialog);
```

C++

```
class TSavePictureDialog : public TOpenPictureDialog;
```

File

ExtDlgs

Description

TSavePictureDialog displays a modal Windows dialog box for selecting file names and saving graphics files. This component is just like TSaveDialog, except that it includes a rectangular preview region. If the selected image can be read by TPicture, it is displayed in the preview region; supported file types include bitmap (.BMP), icon (.ICO), Windows metafile (.WMF), and enhanced Windows metafile (.EMF). If the selected image cannot be displayed, "(None)" appears in the preview region.

See Also

TOpenPictureDialog (see page 13282)

TPicture (see page 14584)

TSaveDialog (see page 11489)

69.1.3.1 TSavePictureDialog Members

The following tables list the members exposed by TSavePictureDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanClose (see page 11413)	This is CanClose, a member of class TOpenDialog.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 11413)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoCanClose (see page 11413)	This is DoCanClose, a member of class TOpenDialog.
	DoClose (see page 11282)	Generates an OnClose (see page 11287) event.
V	DoClose (see page 13287)	Generates an OnClose event.
	DoExecute (see page 11413)	This is the overview for the DoExecute method overload.
	DoFolderChange (see page 11414)	This is DoFolderChange, a member of class TOpenDialog.
	DoIncludeItem (see page 11414)	This is DoIncludeItem, a member of class TOpenDialog.
	DoSelectionChange (see page 11415)	This is DoSelectionChange, a member of class TOpenDialog.
V	DoSelectionChange (see page 13287)	This is DoSelectionChange, a member of class TOpenPictureDialog.
	DoShow (see page 11283)	Generates an OnShow (see page 11288) event.
V	DoShow (see page 13287)	Generates an OnShow event.
	DoTypeChange (see page 11415)	This is DoTypeChange, a member of class TOpenDialog.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetFileNames (see page 11417)	This is GetFileNames, a member of class TOpenDialog.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.

V	GetStaticRect (see page 11417)	Returns the coordinates of the reserved area of the dialog box.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	MessageHook (see page 11284)	Provides special Windows message processing for the dialog.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	PreviewClick (see page 13288)	This is PreviewClick, a member of class TOpenPictureDialog.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	TaskModalDialog (see page 11284)	Ensures that Windows messages are properly routed when the dialog executes.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WndProc (see page 11285)	Responds to Windows messages sent to the dialog.
V	WndProc (see page 11420)	Responds to Windows messages sent to the dialog.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ImageCtrl (see page 13292)	This is ImageCtrl, a member of class TOpenPictureDialog.
R	PictureLabel (see page 13292)	This is PictureLabel, a member of class TOpenPictureDialog.
	Template (see page 11287)	Points to a custom template for the dialog.
	TemplateModule (see page 11287)	This is TemplateModule, a member of class TCommonDialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 11281)	Creates and initializes a TCommonDialog (see page 11277) instance.
V	Create (see page 11411)	Creates and initializes a TOpenDialog (see page 11407) instance.
V	Create (see page 13286)	Creates and initializes a TOpenPictureDialog (see page 13282) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 11282)	Sends unhandled messages to Windows for processing.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 11281)	Destroys the TCommonDialog (see page 11277) instance and frees its memory.
V	Destroy (see page 11412)	Destroys the TOpenDialog (see page 11407) and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Execute (see page 13305)	Displays the Save As dialog box.
V	Execute (see page 11283)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	FileEditStyle (see page 11423)	Determines the style of the file-selection dialog. (Obsolete.)
R	Files (see page 11429)	List of selected file names.
R	Handle (see page 11286)	Specifies the window handle of the dialog.
	HistoryList (see page 11436)	Maintains a list of previously selected files. (Obsolete.)
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCanClose (see page 11439)	Occurs when the user tries to close the dialog without canceling.
	OnClose (see page 11287)	Occurs when the dialog closes.
	OnFolderChange (see page 11440)	Occurs when a directory is opened or closed from the dialog.
	OnIncludeItem (see page 11440)	Occurs before the dialog adds a file to the file list box.
	OnSelectionChange (see page 11441)	Occurs when file names displayed in the dialog are changed.
	OnShow (see page 11288)	Occurs when the dialog opens.
	OnTypeChange (see page 11441)	Occurs when the file types displayed in the dialog are changed.

Published Properties

	Name	Description
	Ctl3D (see page 11285)	Determines whether the dialog has a three-dimensional look.
	DefaultExt (see page 11421)	Specifies a default file extension.
	FileName (see page 11424)	Indicates the name and directory path of the last file selected.
	Filter (see page 11431)	Determines the file masks (filters) available in the dialog.
	Filter (see page 13289)	Determines the file masks (filters) available in the dialog.
	FilterIndex (see page 11434)	Determines which filter is selected by default when the dialog opens.
	HelpContext (see page 11286)	Specifies the context number for online Help.
	InitialDir (see page 11436)	Determines the current directory when the dialog opens.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 11436)	Determines the appearance and behavior of the file-selection dialog.
	OptionsEx (see page 11438)	Augments the Options (see page 11436) property with additional flags that determine the appearance and behavior of the file-selection dialog.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Title (see page 11438)	Specifies the text in the dialog's title bar.

69.1.3.2 TSavePictureDialog Methods

The methods of the TSavePictureDialog class are listed here.

Public Methods

	Name	Description
	Execute (see page 13305)	Displays the Save As dialog box.

69.1.3.2.1 TSavePictureDialog.Execute Method

Displays the Save As dialog box.

Pascal

```
function Execute: Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute();
```

Description

Execute opens the Save As dialog, returning true when the user selects a file name and clicks Save.

69.1.4 TSaveTextFileDialog Class

Class Hierarchy

Pascal

```
TSaveTextFileDialog = class(TOpenTextFileDialog);
```

C++

```
class TSaveTextFileDialog : public TOpenTextFileDialog;
```

File

ExtDlgs

Description

This is class ExtDlgs.TSaveTextFileDialog.

69.1.4.1 TSaveTextFileDialog Members

The following tables list the members exposed by TSaveTextFileDialog.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanClose (see page 11413)	This is CanClose, a member of class TOpenDialog.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 11413)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoCanClose (see page 11413)	This is DoCanClose, a member of class TOpenDialog.
	DoClose (see page 11282)	Generates an OnClose (see page 11287) event.
V	DoClose (see page 13297)	Generates an OnClose event.
	DoExecute (see page 11413)	This is the overview for the DoExecute method overload.
	DoFolderChange (see page 11414)	This is DoFolderChange, a member of class TOpenDialog.
	DoIncludeItem (see page 11414)	This is DoIncludeItem, a member of class TOpenDialog.
	DoSelectionChange (see page 11415)	This is DoSelectionChange, a member of class TOpenDialog.
	DoShow (see page 11283)	Generates an OnShow (see page 11288) event.
V	DoShow (see page 13297)	Generates an OnShow event.
	DoTypeChange (see page 11415)	This is DoTypeChange, a member of class TOpenDialog.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetFileNames (see page 11417)	This is GetFileNames, a member of class TOpenDialog.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetStaticRect (see page 11417)	Returns the coordinates of the reserved area of the dialog box.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	MessageHook (see page 11284)	Provides special Windows message processing for the dialog.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	TaskModalDialog (see page 11284)	Ensures that Windows messages are properly routed when the dialog executes.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WndProc (see page 11285)	Responds to Windows messages sent to the dialog.
	WndProc (see page 11420)	Responds to Windows messages sent to the dialog.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Template (see page 11287)	Points to a custom template for the dialog.
	TemplateModule (see page 11287)	This is TemplateModule, a member of class TCommonDialog.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 11281)	Creates and initializes a TCommonDialog (see page 11277) instance.
	Create (see page 11411)	Creates and initializes a TOpenDialog (see page 11407) instance.
	Create (see page 13296)	Creates and initializes a TOpenDialog instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 11282)	Sends unhandled messages to Windows for processing.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 11281)	Destroys the TCommonDialog (see page 11277) instance and frees its memory.
	Destroy (see page 11412)	Destroys the TOpenDialog (see page 11407) and frees its memory.
	Destroy (see page 13296)	Destroys the TOpenDialog and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 13309)	Displays the dialog box.
	Execute (see page 11283)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	FileEditStyle (see page 11423)	Determines the style of the file-selection dialog. (Obsolete.)
R	Files (see page 11429)	List of selected file names.
R	Handle (see page 11286)	Specifies the window handle of the dialog.
	HistoryList (see page 11436)	Maintains a list of previously selected files. (Obsolete.)
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCanClose (see page 11439)	Occurs when the user tries to close the dialog without canceling.
	OnClose (see page 11287)	Occurs when the dialog closes.
	OnFolderChange (see page 11440)	Occurs when a directory is opened or closed from the dialog.
	OnIncludeItem (see page 11440)	Occurs before the dialog adds a file to the file list box.
	OnSelectionChange (see page 11441)	Occurs when file names displayed in the dialog are changed.
	OnShow (see page 11288)	Occurs when the dialog opens.
	OnTypeChange (see page 11441)	Occurs when the file types displayed in the dialog are changed.

Published Properties

	Name	Description
	Ctl3D (see page 11285)	Determines whether the dialog has a three-dimensional look.
	DefaultExt (see page 11421)	Specifies a default file extension.
	EncodingIndex (see page 13300)	This is EncodingIndex, a member of class TOpenTextFileDialog.
	Encodings (see page 13300)	This is Encodings, a member of class TOpenTextFileDialog.
	FileName (see page 11424)	Indicates the name and directory path of the last file selected.
	Filter (see page 11431)	Determines the file masks (filters) available in the dialog.
	FilterIndex (see page 11434)	Determines which filter is selected by default when the dialog opens.
	HelpContext (see page 11286)	Specifies the context number for online Help.
	InitialDir (see page 11436)	Determines the current directory when the dialog opens.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 11436)	Determines the appearance and behavior of the file-selection dialog.
	OptionsEx (see page 11438)	Augments the Options (see page 11436) property with additional flags that determine the appearance and behavior of the file-selection dialog.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Title (see page 11438)	Specifies the text in the dialog's title bar.

69.1.4.2 TSaveTextFileDialog Methods

The methods of the TSaveTextFileDialog class are listed here.

Public Methods

	Name	Description
	Execute (see page 13309)	Displays the dialog box.

69.1.4.2.1 TSaveTextFileDialog.Execute Method

Displays the dialog box.

Pascal

```
function Execute: Boolean; override;
```

C++

```
virtual __fastcall Boolean Execute();
```

Description

TCommonDialog introduces the abstract Execute method, which descendant classes use to open the dialog instance at runtime. Generally, Execute returns true when the user makes a selection and clicks OK, and returns false when the user closes the dialog without making a selection.

See Also

TaskModalDialog (see page 11284)

69.2 Constants

The following table lists constants in this documentation.

Constants

Name	Description
DefaultEncodingNames (see page 13310)	This is constant ExtDlgs.DefaultEncodingNames.

69.2.1 ExtDlgs.DefaultEncodingNames Constant

Pascal

```
DefaultEncodingNames: array[0..5] of string = (SANSIEncoding, SASCIIEncoding,
SUnicodeEncoding, SBigEndianEncoding, SUTF8Encoding, SUTF7Encoding);
```

C++

```
array[0..5] of string DefaultEncodingNames = (SANSIEncoding, SASCIIEncoding, SUnicodeEncoding,
SBigEndianEncoding, SUTF8Encoding, SUTF7Encoding);
```

File

ExtDlgs

Description

This is constant ExtDlgs.DefaultEncodingNames.

70 FileCtrl Namespace

This is namespace FileCtrl.

Topics

Name	Description
Classes (see page 13313)	The following table lists classes in this documentation.
Functions (see page 13616)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 13620)	The following table lists structs, records, enums in this documentation.
Types (see page 13624)	The following table lists types in this documentation.
Constants (see page 13626)	The following table lists constants in this documentation.

Classes

	Name	Description
	TDirectoryListBox (see page 13313)	TDirectoryListBox represents a list box control that is aware of the directory structure of the current drive.
	TDriveComboBox (see page 13396)	TDriveComboBox implements a specialized combo box that displays all the drives available when the application runs.
	TFileListBox (see page 13465)	TFileListBox is a specialized list box that lists all the files in a specified directory.
	TFilterComboBox (see page 13547)	TFilterComboBox is a specialized combo box that presents the user with a choice of file filters.

Constants

Name	Description
WNTYPE_DRIVE (see page 13626)	This is constant FileCtrl.WNTYPE_DRIVE.

Functions

	Name	Description
	DirectoryExists (see page 13616)	Determines whether a specified directory exists.
	ForceDirectories (see page 13617)	Creates a new directory, including the creation of parent directories as needed.
	MinimizeName (see page 13617)	Shortens a fully qualified path name so that it can be drawn with a specified length limit.
	ProcessPath (see page 13618)	Parses a file name into its constituent parts.
	SelectDirectory (see page 13619)	Brings up a dialog to allow the user to enter a directory name.
	SelectDirectory (see page 13619)	Brings up a dialog to allow the user to enter a directory name.

Structs, Records, Enums

	Name	Description
	TDriveType (see page 13620)	This is record FileCtrl.TDriveType.
	TFileAttr (see page 13620)	TFileAttr and TFileType (see page 13624) determine which files a file list box displays.
	TSelectDirExtOpt (see page 13621)	This is record FileCtrl.TSelectDirExtOpt.
	TSelectDirOpt (see page 13622)	TSelectDirOpt and TSelectDirOpts (see page 13625) determine how a directory selection dialog responds when the user enters a nonexistent directory.
	TTextCase (see page 13623)	TTextCase indicates the case of the volume name or drive letter that appears in a dialog box.

Types

Name	Description
TFileType (see page 13624)	TFileType describes possible combinations of file attributes.
TSelectDirExtOpts (see page 13624)	This is type FileCtrl.TSelectDirExtOpts.
TSelectDirOpts (see page 13625)	This is type FileCtrl.TSelectDirOpts.

70.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TDirectoryListBox (see page 13313)	TDirectoryListBox represents a list box control that is aware of the directory structure of the current drive.
	TDriveComboBox (see page 13396)	TDriveComboBox implements a specialized combo box that displays all the drives available when the application runs.
	TFileListBox (see page 13465)	TFileListBox is a specialized list box that lists all the files in a specified directory.
	TFilterComboBox (see page 13547)	TFilterComboBox is a specialized combo box that presents the user with a choice of file filters.

70.1.1 TDirectoryListBox Class

TDirectoryListBox represents a list box control that is aware of the directory structure of the current drive.

Class Hierarchy

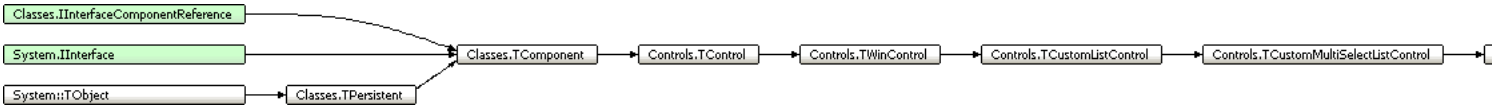

Pascal

```
TDirectoryListBox = class(TCustomListBox);
```

C++

```
class TDirectoryListBox : public TCustomListBox;
```

File

FileCtrl

Description

TDirectoryListBox implements the generic behavior introduced in TCustomListBox. TDirectoryListBox publishes many of the properties inherited from TCustomListBox and introduces new behavior to let the user change directories or drives.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, and
a label on a form. When the user uses the directory list box
to change directories, a message appears and the color of
the form changes if the file AUTOEXEC.BAT is in the new
directory. The code is written in the OnChange event of the
directory list box:
*/
void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
  if (FileListBox1->Items->IndexOf("AUTOEXEC.BAT") > -1)
  {
```

```
    Color = clYellow;
    Label1->Caption = "You are in the root directory!";
  }
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, and
a label on a form. When the user uses the directory list box
to change directories, a message appears and the color of
the form changes if the file AUTOEXEC.BAT is in the new
directory. The code is written in the OnChange event of the
directory list box:
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
  if FileListBox1.Items.IndexOf('AUTOEXEC.BAT') > -1 then
  begin
    Color := clYellow;
    Label1.Caption := 'You are in the root directory!';
  end
  else
  begin
    Color := clBtnFace;
    Label1.Caption := '';
  end;
end;
```

See Also

TDriveComboBox (see page 13396)

TFileListBox (see page 13465)

TFilterComboBox (see page 13547)

TCustomListBox (see page 21308)

70.1.1.1 TDirectoryListBox Members

The following tables list the members exposed by TDirectoryListBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnData (see page 21357)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 21357)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 21358)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDrawItem (see page 21358)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMeasureItem (see page 21360)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.

	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	ClosedBMP (see page 13327)	This is ClosedBMP, a member of class TDirectoryListBox.
	CurrentBMP (see page 13327)	This is CurrentBMP, a member of class TDirectoryListBox.
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDirectory (see page 13327)	This is FDirectory, a member of class TDirectoryListBox.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FMoving (see page 21321)	This is FMoving, a member of class TCustomListBox.
	FMultiSelect (see page 5956)	This is FMultiSelect, a member of class TCustomMultiSelectListControl.
	FOnChange (see page 13328)	This is FOnChange, a member of class TDirectoryListBox.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	OpenedBMP (see page 13328)	This is OpenedBMP, a member of class TDirectoryListBox.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
V	BuildList (see page 13329)	This is BuildList, a member of class TDirectoryListBox.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.

	Change (see page 13329)	This is Change, a member of class TDirectoryListBox.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 21324)	Initializes the window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 13329)	Creates a Windows control corresponding to the list box component.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 13330)	OnDblClick event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DeleteString (see page 21326)	Deletes a string from the list box.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21326)	Saves the list box items in a string list before freeing the underlying Windows control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFindData (see page 21326)	Returns the index of a string in the list box.
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetData (see page 21327)	Returns a string in the list box given its index.
	DoGetDataObject (see page 21327)	Returns the object associated with an item in a virtual list box.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.

	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragCanceled (see page 21328)	Sends a mouse message canceling the drag-and-drop operation.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawItem (see page 13330)	Generates an OnDrawItem event.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetCount (see page 21329)	Returns the value of the Count (see page 21346) property.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetItemData (see page 21329)	Returns a 32-bit value associated with the specified item in a list box.
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
V	GetItemIndex (see page 21329)	Returns the value of the ItemIndex property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
A	GetSelCount (see page 5956)	Returns the value of the SelCount (see page 5957) property.
V	GetSelCount (see page 21330)	Returns the number of selected items.
	GetSelected (see page 21330)	Indicates whether a particular item is selected.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event

	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InternalGetItemData (see page 21330)	Retrieves the data associated with an item in the list box.
	InternalSetItemData (see page 21330)	Associates an integer with a specified item in the list box.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
V	KeyPress (see page 13331)	Responds when the user presses a key.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	LoadRecreateItems (see page 21334)	This is LoadRecreateItems, a member of class TCustomListBox.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
V	MeasureItem (see page 21335)	Generates an OnMeasureItem (see page 21360) event.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 13332)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadBitmaps (see page 13333)	This is ReadBitmaps, a member of class TDirectoryListBox.
	ReadDirectoryNames (see page 13333)	This is ReadDirectoryNames, a member of class TDirectoryListBox.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetContent (see page 21335)	Removes all of the items from a list box.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
V	SaveRecreateItems (see page 21336)	This is SaveRecreateItems, a member of class TCustomListBox.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.

	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItemData (see page 21336)	Sets the 32-bit value associated with the specified item in a list box.
A	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
V	SetItemIndex (see page 21336)	Gives focus to the item with a specified index.
A	SetMultiSelect (see page 5956)	Sets the value of the MultiSelect (see page 5957) property.
V	SetMultiSelect (see page 21337)	Sets the MultiSelect property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.

	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WndProc (see page 21337)	Handles Windows messages.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 21343)	Determines whether the list box has a border.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	ExtendedSelect (see page 21346)	This is ExtendedSelect, a member of class TCustomListBox.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Sorted (see page 21353)	Specifies whether the items in a list box are arranged alphabetically.
	Style (see page 21354)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabWidth (see page 21355)	Specifies the size of the tabs in the list box.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddItem (see page 5940)	Adds an item to the list control.
	AddItem (see page 21322)	Adds an item to the list box
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 5940)	Deletes all of the items from the list control.
V	Clear (see page 21322)	Deletes all items from the list box.
A	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
V	ClearSelection (see page 21323)	Deselects all selected items in the list box, if possible.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
A	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
V	CopySelection (see page 21323)	Copies the selected items to another list-type control
V	Create (see page 13326)	Creates and initializes a directory list box.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	DeleteSelected (see page 5941)	Deletes all the selected items.
V	DeleteSelected (see page 21325)	Deletes the selected items from the list box
V	Destroy (see page 13326)	Disposes of an instance of a directory list box.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DisplayCase (see page 13330)	Preserves or converts the case of the string passed to it.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FileCompareText (see page 13331)	Compares two strings, taking into account the case-sensitivity of the file system.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.

	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItemPath (see page 13331)	This is GetItemPath, a member of class TDirectoryListBox.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ItemAtPos (see page 21331)	Returns the index of the list box item indicated by the coordinates of a point on the control.
	ItemRect (see page 21333)	Returns the rectangle that surrounds the item specified in the Item parameter.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	OpenCurrent (see page 13332)	Opens the selected directory.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)

V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
A	SelectAll (see page 5943)	Selects all items or all text in the selected item.
V	SelectAll (see page 21336)	Selects the entire block of text in the list box.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Update (see page 13333)	Refreshes the directory list.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AutoCompleteDelay (see page 21342)	Specifies the delay between a key press and an attempt to autocomplete the field.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 21343)	Provides a drawing surface when implementing a handler for the OnDrawItem (see page 21358) event of an owner-draw list box.
R	CaseSensitive (see page 13342)	Indicates whether the system supports case-sensitive directory information.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.

	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Count (see page 21346)	Indicates the number of items in the list box.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Directory (see page 13344)	Determines the current directory for the directory list box control.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Drive (see page 13349)	Determines the drive for which the list box displays the directory structure.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	Items (see page 21348)	Contains the strings that appear in the list box.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	MultiSelect (see page 5957)	Specifies whether the user can select more than one item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	PreserveCase (see page 13362)	Indicates whether the file system preserves the case of file names when writing to disk.
	ScrollWidth (see page 21351)	Specifies the width, in pixels, by which the list box can scroll horizontally.
R	SelCount (see page 5957)	Indicates the number of selected items.
	Selected (see page 21351)	Indicates whether a particular item is selected.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TopIndex (see page 21356)	Specifies the index number of the item that appears at the top of the list box.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 13367)	Occurs when the user selects a new directory with the mouse or keyboard when the user moves the selection bar and presses enter.
	OnClick (see page 13369)	Occurs when the user clicks the control.
	OnContextPopup (see page 13371)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 13371)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

	OnDragDrop (see page 13372)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13374)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 13376)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 13377)	Occurs when a control receives the input focus.
	OnExit (see page 13380)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 13381)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 13384)	Occurs when a key is pressed.
	OnKeyUp (see page 13385)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 13387)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 13388)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 13390)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 13391)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 13391)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 13393)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDrag (see page 13395)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 13336)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 13339)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 13340)	Determines whether the user can give focus to items by typing in the list.
	BevelEdges (see page 13340)	Specifies which edges of the control are beveled.
	BevelInner (see page 13340)	Specifies the cut of the inner bevel.
	BevelKind (see page 13341)	Specifies the control's bevel style.
	BevelOuter (see page 13341)	Specifies the cut of the outer bevel.
	Color (see page 13342)	Specifies the background color of the control.
	Columns (see page 13343)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	Constraints (see page 13343)	Specifies the size constraints for the control.
	Ctl3D (see page 13344)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DirLabel (see page 13346)	Provides a simple way to display the current directory as the caption of a label control.
	DragCursor (see page 13347)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 13347)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 13350)	Controls whether the control responds to mouse, keyboard, and timer events.
	FileList (see page 13350)	Provides a simple way to connect a directory list box with a file list box.
	Font (see page 13353)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 13354)	Determines the behavior of the input method editor (IME).
	ImeName (see page 13354)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 13355)	Determines whether the list box displays the partial items.

	ItemHeight (see page 13355)	Specifies the height in pixels of an item in an owner-draw list box.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentColor (see page 13356)	Determines where a control looks for its color information.
	ParentCtl3D (see page 13356)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 13357)	Determines where a control looks for its font information.
	ParentShowHint (see page 13357)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 13358)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 13363)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 13364)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 13365)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 13365)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

70.1.1.2 TDirectoryListBox.Create Constructor

Creates and initializes a directory list box.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDirectoryListBox(TComponent * AOwner);
```

Description

Call Create to instantiate a directory list box at runtime. For directory list boxes created at design time, Create is called automatically.

See Also

Create (see page 21320)

Sorted (see page 21353)

Style (see page 21354)

ParentWindow (see page 6210)

70.1.1.3 TDirectoryListBox.Destroy Destructor

Disposes of an instance of a directory list box.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDirectoryListBox();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free checks that the list box reference is not nil before it calls Destroy.

Destroy frees the bitmaps used for the Open, Close and Current folders, and then calls the inherited Destroy method.

See Also

Destroy (see page 21320)

70.1.1.4 TDirectoryListBox Fields

The fields of the TDirectoryListBox class are listed here.

Protected Fields

	Name	Description
	ClosedBMP (see page 13327)	This is ClosedBMP, a member of class TDirectoryListBox.
	CurrentBMP (see page 13327)	This is CurrentBMP, a member of class TDirectoryListBox.
	FDirectory (see page 13327)	This is FDirectory, a member of class TDirectoryListBox.
	FOnChange (see page 13328)	This is FOnChange, a member of class TDirectoryListBox.
	OpenedBMP (see page 13328)	This is OpenedBMP, a member of class TDirectoryListBox.

70.1.1.4.1 TDirectoryListBox.ClosedBMP Field

Pascal

```
ClosedBMP: TBitmap;
```

C++

```
TBitmap ClosedBMP;
```

Description

This is ClosedBMP, a member of class TDirectoryListBox.

70.1.1.4.2 TDirectoryListBox.CurrentBMP Field

Pascal

```
CurrentBMP: TBitmap;
```

C++

```
TBitmap CurrentBMP;
```

Description

This is CurrentBMP, a member of class TDirectoryListBox.

70.1.1.4.3 TDirectoryListBox.FDirectory Field

Pascal

```
FDirectory: string;
```

C++

```
AnsiString FDirectory;
```

Description

This is FDirectory, a member of class TDirectoryListBox.

70.1.1.4.4 TDirectoryListBox.FOnChange Field

Pascal

```
FOnChange: TNotifyEvent;
```

C++

```
TNotifyEvent FOnChange;
```

Description

This is FOnChange, a member of class TDirectoryListBox.

70.1.1.4.5 TDirectoryListBox.OpenedBMP Field

Pascal

```
OpenedBMP: TBitmap;
```

C++

```
TBitmap OpenedBMP;
```

Description

This is OpenedBMP, a member of class TDirectoryListBox.

70.1.1.5 TDirectoryListBox Methods

The methods of the TDirectoryListBox class are listed here.

Protected Methods

	Name	Description
	BuildList (see page 13329)	This is BuildList, a member of class TDirectoryListBox.
	Change (see page 13329)	This is Change, a member of class TDirectoryListBox.
	CreateWnd (see page 13329)	Creates a Windows control corresponding to the list box component.
	DblClick (see page 13330)	OnDblClick event dispatcher.
	DrawItem (see page 13330)	Generates an OnDrawItem event.
	KeyPress (see page 13331)	Responds when the user presses a key.
	Notification (see page 13332)	Responds to notifications that components are being created or destroyed.
	ReadBitmaps (see page 13333)	This is ReadBitmaps, a member of class TDirectoryListBox.
	ReadDirectoryNames (see page 13333)	This is ReadDirectoryNames, a member of class TDirectoryListBox.

Public Methods

	Name	Description
	DisplayCase (see page 13330)	Preserves or converts the case of the string passed to it.
	FileCompareText (see page 13331)	Compares two strings, taking into account the case-sensitivity of the file system.
	GetItemPath (see page 13331)	This is GetItemPath, a member of class TDirectoryListBox.
	OpenCurrent (see page 13332)	Opens the selected directory.
	Update (see page 13333)	Refreshes the directory list.

70.1.1.5.1 TDirectoryListBox.BuildList Method

Pascal

```
procedure BuildList; virtual;
```

C++

```
virtual __fastcall BuildList();
```

Description

This is BuildList, a member of class TDirectoryListBox.

70.1.1.5.2 TDirectoryListBox.Change Method

Pascal

```
procedure Change; virtual;
```

C++

```
virtual __fastcall Change();
```

Description

This is Change, a member of class TDirectoryListBox.

70.1.1.5.3 TDirectoryListBox.CreateWnd Method

Creates a Windows control corresponding to the list box component.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the list box must be displayed. CreateWnd calls the inherited CreateWnd method, which calls

CreateParams to initialize the window-creation parameters.

CreateWindowHandle to create the window handle for the control.

CreateWnd additionally sets up the list box position, tabs, items, and columns.

See Also

Columns (see page 21346)

CreateHandle (see page 6135)

CreateParams (see page 21324)

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

ItemIndex

Items (see page 21348)

TabWidth (see page 21355)

TopIndex (see page 21356)

70.1.1.5.4 TDirectoryListBox.DblClick Method

OnDblClick event dispatcher.

Pascal

```
procedure DblClick; override;
```

C++

```
virtual __fastcall DblClick();
```

Description

DblClick calls any event handler attached to the OnDblClick event. Override DblClick to provide other responses in addition to the inherited event-handler call when the user double-clicks the control.

The DblClick method is triggered by left mouse-button double-click message (WM_LBUTTONDBLCLK) from Windows.

See Also

Click (see page 5743)

OnDblClick (see page 5854)

70.1.1.5.5 TDirectoryListBox.DisplayCase Method

Preserves or converts the case of the string passed to it.

Pascal

```
function DisplayCase(const S: String): String;
```

C++

```
__fastcall AnsiString DisplayCase(const AnsiString S);
```

Description

DisplayCase converts S to all-lower-case (using the currently installed language driver) if neither PreserveCase (see page 13362) nor CaseSensitive (see page 13342) is true. Otherwise, it returns S unmodified.

See Also

CaseSensitive (see page 13342)

PreserveCase (see page 13362)

70.1.1.5.6 TDirectoryListBox.DrawItem Method

Generates an OnDrawItem event.

Pascal

```
procedure DrawItem(Index: Integer; Rect: TRect; State: TOwnerDrawState); override;
```

C++

```
virtual __fastcall DrawItem(int Index, TRect Rect, TOwnerDrawState State);
```

Description

DrawItem is called whenever a visual aspect of an owner-draw list box changes. The list box calls DrawItem for each visible item in its list, passing the index of the item in the Index parameter.

By default, the DrawItem method for a list box calls any event handler attached to the component's OnDrawItem event. If there is no handler attached, DrawItem fills the rectangle passed in Rect and draws any text associated with the indexed item. The default drawing ignores the State parameter, although State is passed along to attached event handlers.

See Also

ItemHeight (see page 21347)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

Style (see page 21354)

70.1.1.5.7 TDirectoryListBox.FileCompareText Method

Compares two strings, taking into account the case-sensitivity of the file system.

Pascal

```
function FileCompareText(const A: String; const B: String): Integer;
```

C++

```
__fastcall int FileCompareText(const AnsiString A, const AnsiString B);
```

Description

If CaseSensitive (see page 13342) is false, FileCompareText compares the strings passed to it case-insensitively. Otherwise, it uses AnsiCompareStr to compare the strings.

See Also

AnsiCompareStr (see page 22789)

CaseSensitive (see page 13342)

70.1.1.5.8 TDirectoryListBox.GetItemPath Method

Pascal

```
function GetItemPath(Index: Integer): string;
```

C++

```
__fastcall AnsiString GetItemPath(int Index);
```

Description

This is GetItemPath, a member of class TDirectoryListBox.

70.1.1.5.9 TDirectoryListBox.KeyPress Method

Responds when the user presses a key.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress is called automatically when the list box receives a key-press message (WM_CHAR). It generates an OnKeyPress event, and then, if the AutoComplete property is true, updates ItemIndex to reflect the next character that the user typed.

The Key parameter is the character represented by the key that is pressed, not a Windows virtual key code.

See Also

OnKeyPress (see page 6227)

70.1.1.5.10 TDirectoryListBox.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

70.1.1.5.11 TDirectoryListBox.OpenCurrent Method

Opens the selected directory.

Pascal

```
procedure OpenCurrent;
```

C++

```
__fastcall OpenCurrent();
```

Description

OpenCurrent opens the directory selected in the directory list box, as if the user had double-clicked the directory.

C++ Examples:

```
/*
This example uses a directory list box, a button, and a
label on a form. When the user selects a directory in the
directory list box and clicks the button, the selected
directory opens, and the path of the second directory
displayed in the list box appears as the caption of the
label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  DirectoryListBox1->OpenCurrent();
  Label1->Caption = DirectoryListBox1->GetItemPath(1);
}
```

Delphi Examples:

```
{
This example uses a directory list box, a button, and a
label on a form. When the user selects a directory in the
directory list box and clicks the button, the selected
directory opens, and the path of the second directory
displayed in the list box appears as the caption of the
label.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  DirectoryListBox1.OpenCurrent;
  Label1.Caption := DirectoryListBox1.GetItemPath(1);
end;
```

70.1.1.5.12 TDirectoryListBox.ReadBitmaps Method

Pascal

```
procedure ReadBitmaps; virtual;
```

C++

```
virtual __fastcall ReadBitmaps();
```

Description

This is ReadBitmaps, a member of class TDirectoryListBox.

70.1.1.5.13 TDirectoryListBox.ReadDirectoryNames Method

Pascal

```
function ReadDirectoryNames(const ParentDirectory: string; DirectoryList: TStringList):
Integer;
```

C++

```
__fastcall int ReadDirectoryNames(const AnsiString ParentDirectory, TStringList DirectoryList);
```

Description

This is ReadDirectoryNames, a member of class TDirectoryListBox.

70.1.1.5.14 TDirectoryListBox.Update Method

Refreshes the directory list.

Pascal

```
procedure Update; reintroduce;
```

C++

```
__fastcall Update();
```

Description

Call Update to refresh the directory list after new directories have been created or existing directories deleted.

C++ Examples:

```
/*
The following sample code sets the directory of
DirectoryListBox1 to c:\\WINDOWS when the form is created.
When Button1 is pressed, a subdirectory called
c:\\WINDOWS\\mytemp is added, but note that it is not updated
in DirectoryListBox1 until Button2 is pressed and Update is
called.
Note: A EInOutError will occur if you add a directory that
already exists.  Also, you must add the new directory to
your current directory to see the magic happen.  Also,
capitalization matters in C++ and the file spearators must
be "\\".
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ForceDirectories(Edit1->Text);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  DirectoryListBox1->Update();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  DirectoryListBox1->Directory = "c:\\WINDOWS";
}
```

Delphi Examples:

```
{
The following sample code sets the directory of
DirectoryListBox1 to C:\WINDOWS when the form is created.
When Button1 is pressed, a subdirectory called
mytemp is added to C:\WINDOWS, but note that it is not
updated in DirectoryListBox1 until Button2 is pressed
and Update is called.
Note: A EInOutError will occur if you add a directory that
already exists.  Also, you must add the new directory to
your current path to see the magic happen.
}

procedure TForm1.Button1Click(Sender: TObject);
begin
  MkDir(Edit1.Text);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  DirectoryListBox1.Update;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
  DirectoryListBox1.Directory := 'c:\WINDOWS';
end;
```

70.1.1.6 TDirectoryListBox Properties

The properties of the TDirectoryListBox class are listed here.

Public Properties

	Name	Description
R	CaseSensitive (see page 13342)	Indicates whether the system supports case-sensitive directory information.
	Directory (see page 13344)	Determines the current directory for the directory list box control.
	Drive (see page 13349)	Determines the drive for which the list box displays the directory structure.
R	PreserveCase (see page 13362)	Indicates whether the file system preserves the case of file names when writing to disk.

Published Properties

	Name	Description
	Align (see page 13336)	Determines how the control aligns within its container (parent control).
	Anchors (see page 13339)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 13340)	Determines whether the user can give focus to items by typing in the list.
	BevelEdges (see page 13340)	Specifies which edges of the control are beveled.
	BevelInner (see page 13340)	Specifies the cut of the inner bevel.
	BevelKind (see page 13341)	Specifies the control's bevel style.
	BevelOuter (see page 13341)	Specifies the cut of the outer bevel.
	Color (see page 13342)	Specifies the background color of the control.
	Columns (see page 13343)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	Constraints (see page 13343)	Specifies the size constraints for the control.
	Ctl3D (see page 13344)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DirLabel (see page 13346)	Provides a simple way to display the current directory as the caption of a label control.
	DragCursor (see page 13347)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 13347)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 13350)	Controls whether the control responds to mouse, keyboard, and timer events.
	FileList (see page 13350)	Provides a simple way to connect a directory list box with a file list box.
	Font (see page 13353)	Controls the attributes of text written on or in the control.
	ImeMode (see page 13354)	Determines the behavior of the input method editor (IME).
	ImeName (see page 13354)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 13355)	Determines whether the list box displays the partial items.
	ItemHeight (see page 13355)	Specifies the height in pixels of an item in an owner-draw list box.
	ParentColor (see page 13356)	Determines where a control looks for its color information.
	ParentCtl3D (see page 13356)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 13357)	Determines where a control looks for its font information.
	ParentShowHint (see page 13357)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 13358)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 13363)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 13364)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 13365)	Determines if the user can tab to a control.
	Visible (see page 13365)	Determines whether the component appears onscreen.

70.1.1.6.1 TDirectoryListBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
```

```
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
```

```
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
```

```
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

70.1.1.6.2 TDirectoryListBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

70.1.1.6.3 TDirectoryListBox.AutoComplete Property

Determines whether the user can give focus to items by typing in the list.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

Use AutoComplete to specify whether the list box automatically updates the ItemIndex property in response to user keystrokes. When AutoComplete is true, the item with focus changes to the first item that begins with the sequence of characters that the user typed. As the user types more characters, ItemIndex may change to a later item because the string to match becomes longer.

See Also

ItemIndex

70.1.1.6.4 TDirectoryListBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

70.1.1.6.5 TDirectoryListBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

70.1.1.6.6 TDirectoryListBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

70.1.1.6.7 TDirectoryListBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

70.1.1.6.8 TDirectoryListBox.CaseSensitive Property

Indicates whether the system supports case-sensitive directory information.

Pascal

```
property CaseSensitive: Boolean;
```

C++

```
__property Boolean CaseSensitive;
```

Description

Read CaseSensitive to ascertain whether the current file system supports case-sensitive file names. CaseSensitive returns true if it does and false if it does not.

See Also

DisplayCase (see page 13330)

FileCompareText (see page 13331)

PreserveCase (see page 13362)

70.1.1.6.9 TDirectoryListBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

70.1.1.6.10 TDirectoryListBox.Columns Property

Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.

Pascal

```
property Columns: Integer;
```

C++

```
__property int Columns;
```

Description

Use Columns to specify the number of columns, in a multi-column list box, that are visible without having to use the horizontal scroll bar.

Multi-column list boxes have a horizontal scroll bar that allows users to view multiple columns as they wrap. The default value for Columns is 0, meaning that the list box is not multi-column. That is, users can scroll only vertically and the list of items will not wrap.

For Columns values greater than 0, multiple columns accommodate the items as they wrap beyond the bottom of the list box. The Columns property specifies the number of columns that are visible without having to horizontally scroll the list box.

The width of each column depends upon both the Width property and the number of Columns.

See Also

ItemHeight (see page 21347)

Width (see page 5849)

70.1.1.6.11 TDirectoryListBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

70.1.1.6.12 TDirectoryListBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

70.1.1.6.13 TDirectoryListBox.Directory Property

Determines the current directory for the directory list box control.

Pascal

```
property Directory: string;
```

C++

```
__property AnsiString Directory;
```

Description

The directory list box displays the value of the Directory property as the current directory in the list box. The example shows how a directory list box and a file list box can work together through their Directory properties.

If the form contains a file list box, you can set the FileList (see page 13350) property to have the file list box automatically display the files from the directory specified by the Directory property.

C++ Examples:

```
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

See Also

DirLabel (see page 13346)

Drive (see page 13349)

FileList (see page 13350)

70.1.1.6.14 TDirectoryListBox.DirLabel Property

Provides a simple way to display the current directory as the caption of a label control.

Pascal

```
property DirLabel: TLabel;
```

C++

```
__property TLabel DirLabel;
```

Description

Use DirLabel to specify a label. When the current directory changes in the directory list box, the change is reflected in the caption of the label.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FileListBox1->FileEdit = Edit1;
  FilterComboBox1->FileList = FileListBox1;
  DirectoryListBox1->FileList = FileListBox1;
  DirectoryListBox1->DirLabel = Label1;
  DriveComboBox1->DirList = DirectoryListBox1;
  Button1->Default = true;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FileListBox1->ApplyFilePath(Edit1->Text);
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FileListBox1.ApplyFilePath(Edit1.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileEdit := Edit1;
```

```
    FilterComboBox1.FileList := FileListBox1;
    DirectoryListBox1.FileList := FileListBox1;
    DirectoryListBox1.DirLabel := Label1;
    DriveComboBox1.DirList := DirectoryListBox1;
    Button1.Default := True;
  end;
```

See Also

Caption (see page 5806)

Directory (see page 13344)

DirList (see page 13422)

FileEdit (see page 13501)

FileList (see page 13350)

70.1.1.6.15 TDirectoryListBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

70.1.1.6.16 TDirectoryListBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
```

```
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

70.1.1.6.17 TDirectoryListBox.Drive Property

Determines the drive for which the list box displays the directory structure.

Pascal

```
property Drive: Char;
```

C++

```
__property Char Drive;
```

Description

When the value of Drive changes, the Directory (see page 13344) value changes to the current directory on the specified drive.

C++ Examples:

```
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
```

```
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

See Also

Directory (see page 13344)

DirList (see page 13422)

Text (see page 5842)

70.1.1.6.18 TDirectoryListBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

70.1.1.6.19 TDirectoryListBox.FileList Property

Provides a simple way to connect a directory list box with a file list box.

Pascal

```
property FileList: TFileListBox;
```

C++

```
__property TFileListBox FileList;
```

Description

Use FileList to specify a file list box in which to display the files of the current directory.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FileListBox1->FileEdit = Edit1;
  FilterComboBox1->FileList = FileListBox1;
  DirectoryListBox1->FileList = FileListBox1;
  DirectoryListBox1->DirLabel = Label1;
  DriveComboBox1->DirList = DirectoryListBox1;
  Button1->Default = true;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FileListBox1->ApplyFilePath(Edit1->Text);
}
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FileListBox1.ApplyFilePath(Edit1.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileEdit := Edit1;
  FilterComboBox1.FileList := FileListBox1;
  DirectoryListBox1.FileList := FileListBox1;
  DirectoryListBox1.DirLabel := Label1;
  DriveComboBox1.DirList := DirectoryListBox1;
  Button1.Default := True;
end;
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

See Also

DirLabel (see page 13346)

Directory (see page 13344)

DirList (see page 13422)

FileEdit (see page 13501)

Filter (see page 13575)

Mask (see page 13506)

Mask (see page 13579)

70.1.1.6.20 TDirectoryListBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

70.1.1.6.21 TDirectoryListBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

70.1.1.6.22 TDirectoryListBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector

provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

70.1.1.6.23 TDirectoryListBox.IntegralHeight Property

Determines whether the list box displays the partial items.

Pascal

```
property IntegralHeight: Boolean;
```

C++

```
__property Boolean IntegralHeight;
```

Description

IntegralHeight controls the height of the list box itself.

When IntegralHeight is true, and the Style property is lsOwnerDrawFixed, the list box height will always be a multiple of the ItemHeight property value. In this case, items will never be partially displayed.

If IntegralHeight is false, the list box height can be set to any value, but the last item may not be fully displayed.

If the Style property value of the list box is lbOwnerDrawVariable, setting the IntegralHeight property to true has no effect.

See Also

ItemHeight (see page 21347)

Items (see page 21348)

Style (see page 21354)

Using TDBListBox and TDBComboBox

70.1.1.6.24 TDirectoryListBox.ItemHeight Property

Specifies the height in pixels of an item in an owner-draw list box.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Use ItemHeight to control the height of an item in a list box when the Style property is lbOwnerDrawFixed. The ItemHeight property can also affect the height of the list box itself. When the IntegralHeight property is true, the height of the list box is always an increment of the ItemHeight value.

If the Style property is lbStandard or lbOwnerDrawVariable, the value of ItemHeight is ignored.

See Also

DrawItem (see page 21328)

IntegralHeight (see page 21347)

Items (see page 21348)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

Style (see page 21354)

Using TDBListBox and TDBComboBox

70.1.1.6.25 TDirectoryListBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

70.1.1.6.26 TDirectoryListBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

70.1.1.6.27 TDirectoryListBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

70.1.1.6.28 TDirectoryListBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

70.1.1.6.29 TDirectoryListBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
```

```
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
```

```
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
```

```
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
```

```
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

70.1.1.6.30 TDirectoryListBox.PreserveCase Property

Indicates whether the file system preserves the case of file names when writing to disk.

Pascal

```
property PreserveCase: Boolean;
```

C++

```
__property Boolean PreserveCase;
```

Description

Read PreserveCase to ascertain whether the current file system preserves the case of file names. PreserveCase returns true if it does and false if it does not.

See Also

CaseSensitive (see page 13342)

DisplayCase (see page 13330)

70.1.1.6.31 TDirectoryListBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
```

```
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

70.1.1.6.32 TDirectoryListBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

70.1.1.6.33 TDirectoryListBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

70.1.1.6.34 TDirectoryListBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
```

```
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

70.1.1.7 TDirectoryListBox Events

The events of the TDirectoryListBox class are listed here.

Published Events

	Name	Description
	OnChange (see page 13367)	Occurs when the user selects a new directory with the mouse or keyboard when the user moves the selection bar and presses enter.
	OnClick (see page 13369)	Occurs when the user clicks the control.
	OnContextPopup (see page 13371)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 13371)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 13372)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13374)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 13376)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 13377)	Occurs when a control receives the input focus.
	OnExit (see page 13380)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 13381)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 13384)	Occurs when a key is pressed.
	OnKeyUp (see page 13385)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 13387)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 13388)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 13390)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 13391)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 13391)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 13393)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDrag (see page 13395)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

70.1.1.7.1 TDirectoryListBox.OnChange Event

Occurs when the user selects a new directory with the mouse or keyboard when the user moves the selection bar and presses enter.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to perform an action whenever a new directory is selected.

C++ Examples:

```
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
/*
This example uses a file list box, a directory list box, and
a label on a form. When the user uses the directory list box
to change directories, a message appears and the color of
the form changes if the file AUTOEXEC.BAT is in the new
directory. The code is written in the OnChange event of the
directory list box:
*/
void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
  if (FileListBox1->Items->IndexOf("AUTOEXEC.BAT") > -1)
  {
    Color = clYellow;
    Label1->Caption = "You are in the root directory!";
  }
}
```

Delphi Examples:

```
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
```

```
    FileListBox1.Drive := DriveComboBox1.Drive;
    FileListBox1.Directory := DirectoryListBox1.Directory;
  end;
  {
  This example uses a file list box, a directory list box, and
  a label on a form. When the user uses the directory list box
  to change directories, a message appears and the color of
  the form changes if the file AUTOEXEC.BAT is in the new
  directory. The code is written in the OnChange event of the
  directory list box:
  }
  procedure TForm1.DirectoryListBox1Change(Sender: TObject);
  begin
    FileListBox1.Directory := DirectoryListBox1.Directory;
    if FileListBox1.Items.IndexOf('AUTOEXEC.BAT') > -1 then
    begin
      Color := clYellow;
      Label1.Caption := 'You are in the root directory!';
    end
    else
    begin
      Color := clBtnFace;
      Label1.Caption := '';
    end;
  end;
```

70.1.1.7.2 TDirectoryListBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
```

```
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

70.1.1.7.3 TDirectoryListBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

70.1.1.7.4 TDirectoryListBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

70.1.1.7.5 TDirectoryListBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
```

```
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.
```

```
procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

70.1.1.7.6 TDirectoryListBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
```

```
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
```

```
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

70.1.1.7.7 TDirectoryListBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

70.1.1.7.8 TDirectoryListBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
```

```
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
```

```
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

70.1.1.7.9 TDirectoryListBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
```

```
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

70.1.1.7.10 TDirectoryListBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
```

```
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
```

```
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

70.1.1.7.11 TDirectoryListBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an

OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

70.1.1.7.12 TDirectoryListBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
```

```
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

70.1.1.7.13 TDirectoryListBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level

form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

70.1.1.7.14 TDirectoryListBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
```

```
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;
```

```
procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

70.1.1.7.15 TDirectoryListBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

70.1.1.7.16 TDirectoryListBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

70.1.1.7.17 TDirectoryListBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;
```

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

70.1.1.7.18 TDirectoryListBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
```

```
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
```

```
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

70.1.1.7.19 TDirectoryListBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it

contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

70.1.2 TDriveComboBox Class

TDriveComboBox implements a specialized combo box that displays all the drives available when the application runs.

Class Hierarchy

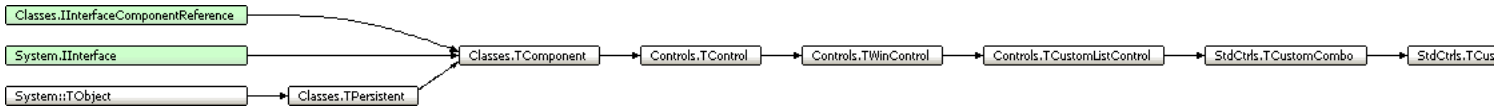

Pascal

```
TDriveComboBox = class(TCustomComboBox);
```

C++

```
class TDriveComboBox : public TCustomComboBox;
```

File

FileCtrl

Description

Add TDriveComboBox to a form to allow users to select a drive. Use TDriveComboBox along with TFileListBox (see page 13465), TFilterComboBox (see page 13547), and TDirectoryListBox (see page 13313) to add full file selection capabilities to a form.

To add a standard Windows 95 file open or save dialog to an application, use TOpenDialog or TSaveDialog instead.

See Also

TDirectoryListBox (see page 13313)

TFileListBox (see page 13465)

TFilterComboBox (see page 13547)

TOpenDialog (see page 11407)

TSaveDialog (see page 11489)

70.1.2.1 TDriveComboBox Members

The following tables list the members exposed by TDriveComboBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnCloseUp (see page 21199)	Occurs when the drop-down list closes up due to some user action.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDrawItem (see page 21224)	Occurs when an item in an owner-draw combo box needs to be displayed.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMeasureItem (see page 21225)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnSelect (see page 21201)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	CDROMBMP (see page 13410)	This is CDROMBMP, a member of class TDriveComboBox.
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefEditProc (see page 21167)	This is FDefEditProc, a member of class TCustomCombo.
	FDefListProc (see page 21167)	This is FDefListProc, a member of class TCustomCombo.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FDropHandle (see page 21168)	This is FDropHandle, a member of class TCustomCombo.
	FDroppingDown (see page 21168)	This is FDroppingDown, a member of class TCustomCombo.

	FEditHandle (see page 21168)	This is FEditHandle, a member of class TCustomCombo.
	FEditInstance (see page 21168)	This is FEditInstance, a member of class TCustomCombo.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFocusChanged (see page 21168)	This is FFocusChanged, a member of class TCustomCombo.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FIsFocused (see page 21169)	This is FIsFocused, a member of class TCustomCombo.
	FixedBMP (see page 13410)	This is FixedBMP, a member of class TDriveComboBox.
	FListHandle (see page 21169)	This is FListHandle, a member of class TCustomCombo.
	FListInstance (see page 21169)	This is FListInstance, a member of class TCustomCombo.
	FloppyBMP (see page 13410)	This is FloppyBMP, a member of class TDriveComboBox.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveIndex (see page 21169)	This is FSaveIndex, a member of class TCustomCombo.
	NetworkBMP (see page 13410)	This is NetworkBMP, a member of class TDriveComboBox.
	RAMBMP (see page 13411)	This is RAMBMP, a member of class TDriveComboBox.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustDropDown (see page 21171)	Adjusts the size and position of the drop down list.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
V	BuildList (see page 13411)	This is BuildList, a member of class TDriveComboBox.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21171)	Generates an OnChange (see page 21197) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
V	Click (see page 13411)	Respond to user click.
	CloseUp (see page 21172)	Generates an OnCloseUp (see page 21199) event.
V	ComboWndProc (see page 21173)	Responds to Windows messages sent to the edit control or drop-down list.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.

	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 21215)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 13412)	Creates the Windows control that represents the combo box.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWindowHandle (see page 21174)	Destroys all of the windows controls that implement the combo box.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21215)	Destroys the Windows control that represents the combo box.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawItem (see page 13412)	Generates an OnDrawItem event.

	DropDown (see page 21175)	Generates an OnDropDown (see page 21199) event.
V	EditWndProc (see page 21175)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetDroppedDown (see page 21176)	Returns the value of the DroppedDown (see page 21185) property.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
A	GetItemCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
V	GetItemCount (see page 21216)	Returns the number of items in the Items (see page 21187) property list.
A	GetItemHt (see page 21177)	Returns the value of the ItemHeight (see page 21186) property.
V	GetItemHt (see page 21216)	Returns the value of the ItemHeight (see page 21186) property.
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
V	GetItemIndex (see page 21177)	Returns the value of the ItemIndex property.
A	GetItemsClass (see page 21177)	Indicates the class that implements the Items (see page 21187) property.
V	GetItemsClass (see page 21217)	Returns the class used to implement the Items (see page 21187) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelLength (see page 21178)	Returns the value of the SelLength (see page 21190) property.
	GetSelStart (see page 21178)	Returns the value of the SelStart (see page 21193) property.
	GetSelText (see page 21217)	Returns the value of the SelText (see page 21222) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyPress (see page 21217)	Responds to WM_CHAR messages sent to the combo box.
	KeyUp (see page 6164)	Respond to released key.
	ListWndProc (see page 21178)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	Loaded (see page 21179)	Initializes the combo box after its form is loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MeasureItem (see page 21218)	Generates an OnMeasureItem (see page 21225) event.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 13413)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Select (see page 21179)	Generates an OnSelect (see page 21201) event.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectItem (see page 21218)	Selects a specified item from the drop-down list.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.

	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropDownCount (see page 21180)	Sets the value of the DropDownCount (see page 21185) property.
	SetDroppedDown (see page 21180)	Sets the value of the DroppedDown (see page 21185) property.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItemHeight (see page 21181)	Sets the value of the ItemHeight (see page 21186) property.
	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
	SetItemIndex (see page 21181)	Sets the value of the ItemIndex property.
	SetItems (see page 21181)	Sets the value of the Items (see page 21187) property.
	SetMaxLength (see page 21182)	Sets the value of the MaxLength (see page 21189) property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelLength (see page 21182)	Sets the value of the SelLength (see page 21190) property.
	SetSelStart (see page 21182)	Sets the value of the SelStart (see page 21193) property.
	SetStyle (see page 21219)	Sets the Style (see page 21224) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.

V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WndProc (see page 21183)	Processes messages received by the combo box.
V	WndProc (see page 21219)	Processes messages received by the combo box.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DropDownCount (see page 21185)	Specifies the maximum number of items displayed in the drop-down list.
R	EditHandle (see page 21185)	Returns the window handle for the edit portion of the combo box.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	ItemCount (see page 21186)	Specifies the number of items in the drop-down list.
	ItemHeight (see page 21186)	Specifies the height, in pixels, of the items in the drop-down list.
R	ListHandle (see page 21189)	Provides access to the window handle of the list of items in the combo box.
	MaxLength (see page 21189)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Sorted (see page 21222)	Determines whether the list portion of the combo box is alphabetized.
	Style (see page 21224)	Determines the display style of the combo box.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
A	AddItem (see page 5940)	Adds an item to the list control.
V	AddItem (see page 21170)	Adds an item to the drop-down list of items.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 5940)	Deletes all of the items from the list control.
V	Clear (see page 21172)	Deletes all text from the edit box and all items from the Items (see page 21187) list.
A	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
V	ClearSelection (see page 21172)	Deselects any selected items in the drop-down list.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
A	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
V	CopySelection (see page 21173)	Copies the selected item in the drop-down list to another list control.
V	Create (see page 13409)	Creates and initializes an instance of TDriveComboBox (see page 13396).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	DeleteSelected (see page 5941)	Deletes all the selected items.
V	DeleteSelected (see page 21174)	Deletes the selected item from the drop-down list.
V	Destroy (see page 13409)	Destroys an instance of TDriveComboBox (see page 13396).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
V	Focused (see page 21175)	Indicates whether the combo box has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.

	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.

V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
A	SelectAll (see page 5943)	Selects all items or all text in the selected item.
V	SelectAll (see page 21180)	Selects the entire block of text in the edit box.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AutoCloseUp (see page 21220)	Specifies whether the drop-down closes up automatically when the user selects an item.
	AutoCompleteDelay (see page 21220)	Specifies the delay between a key press and an attempt to autocomplete the field.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 21184)	Provides access to the drawing surface of the combo box.
	CharCase (see page 21221)	Determines the case of the text in the combo box.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.

R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Drive (see page 13426)	Indicates the currently selected drive displayed in the edit portion of the combo box.
	DroppedDown (see page 21185)	Indicates whether the drop-down list is currently displayed.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	Items (see page 21187)	Provides access to the list of items (strings) in the list portion of the combo box.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelLength (see page 21190)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 21193)	Specifies the position of the first selected character in the edit portion of the combo box.
	SelText (see page 21222)	Represents the selected text in the edit region.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	Text (see page 13438)	Contains a text string associated with the control.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 13442)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 13444)	Occurs when the user clicks the control.
	OnContextPopup (see page 13446)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 13446)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 13447)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13449)	Occurs when the user drags an object over a control.
	OnDropDown (see page 13451)	Occurs when the user opens the drop-down list.
	OnEndDrag (see page 13453)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

	OnEnter (see page 13454)	Occurs when a control receives the input focus.
	OnExit (see page 13456)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 13458)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 13461)	Occurs when a key is pressed.
	OnKeyUp (see page 13462)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 13464)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 13464)	Occurs when the user moves the mouse outside of a control.
	OnStartDrag (see page 13464)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 13414)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 13417)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 13418)	Positions to matching list items as you type.
	AutoDropDown (see page 13418)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 13419)	Specifies which edges of the control are beveled.
	BevelInner (see page 13419)	Specifies the cut of the inner bevel.
	BevelKind (see page 13420)	Specifies the control's bevel style.
	BevelOuter (see page 13420)	Specifies the cut of the outer bevel.
	Color (see page 13420)	Specifies the background color of the control.
	Constraints (see page 13421)	Specifies the size constraints for the control.
	Ctl3D (see page 13422)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DirList (see page 13422)	Connects the drive combo box to a directory list box.
	DragCursor (see page 13424)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 13424)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 13427)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 13427)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 13428)	Determines the behavior of the input method editor (IME).
	ImeName (see page 13429)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentColor (see page 13429)	Determines where a control looks for its color information.
	ParentCtl3D (see page 13430)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 13430)	Determines where a control looks for its font information.
	ParentShowHint (see page 13431)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 13431)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 13436)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 13437)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 13438)	Determines if the user can tab to a control.

	Tag (see page 2808)	Stores an integer value as part of a component.
	TextCase (see page 13439)	Determines if the volume name in the Text (see page 13438) property appears in uppercase or lowercase.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 13440)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

70.1.2.2 TDriveComboBox.Create Constructor

Creates and initializes an instance of TDriveComboBox (see page 13396).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDriveComboBox(TComponent * AOwner);
```

Description

Call Create to create a TDriveComboBox (see page 13396) object at runtime. Drive (see page 13426) combo boxes placed on a form at design time are created automatically.

After allocating the memory for the combo box, Create fills the combo box with all the drives on the system, and initializes the Drive (see page 13426) property to the current drive.

See Also

Drive (see page 13426)

Create (see page 21214)

70.1.2.3 TDriveComboBox.Destroy Destructor

Destroys an instance of TDriveComboBox (see page 13396).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDriveComboBox();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the drive combo box reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

70.1.2.4 TDriveComboBox Fields

The fields of the TDriveComboBox class are listed here.

Protected Fields

	Name	Description
	CDROMBMP (see page 13410)	This is CDROMBMP, a member of class TDriveComboBox.
	FixedBMP (see page 13410)	This is FixedBMP, a member of class TDriveComboBox.
	FloppyBMP (see page 13410)	This is FloppyBMP, a member of class TDriveComboBox.
	NetworkBMP (see page 13410)	This is NetworkBMP, a member of class TDriveComboBox.
	RAMBMP (see page 13411)	This is RAMBMP, a member of class TDriveComboBox.

70.1.2.4.1 TDriveComboBox.CDROMBMP Field

Pascal

```
CDROMBMP: TBitmap;
```

C++

```
TBitmap CDROMBMP;
```

Description

This is CDROMBMP, a member of class TDriveComboBox.

70.1.2.4.2 TDriveComboBox.FixedBMP Field

Pascal

```
FixedBMP: TBitmap;
```

C++

```
TBitmap FixedBMP;
```

Description

This is FixedBMP, a member of class TDriveComboBox.

70.1.2.4.3 TDriveComboBox.FloppyBMP Field

Pascal

```
FloppyBMP: TBitmap;
```

C++

```
TBitmap FloppyBMP;
```

Description

This is FloppyBMP, a member of class TDriveComboBox.

70.1.2.4.4 TDriveComboBox.NetworkBMP Field

Pascal

```
NetworkBMP: TBitmap;
```

C++

```
TBitmap NetworkBMP;
```

Description

This is NetworkBMP, a member of class TDriveComboBox.

70.1.2.4.5 TDriveComboBox.RAMBMP Field

Pascal

```
RAMBMP: TBitmap;
```

C++

```
TBitmap RAMBMP;
```

Description

This is RAMBMP, a member of class TDriveComboBox.

70.1.2.5 TDriveComboBox Methods

The methods of the TDriveComboBox class are listed here.

Protected Methods

	Name	Description
	BuildList (see page 13411)	This is BuildList, a member of class TDriveComboBox.
	Click (see page 13411)	Respond to user click.
	CreateWnd (see page 13412)	Creates the Windows control that represents the combo box.
	DrawItem (see page 13412)	Generates an OnDrawItem event.
	Notification (see page 13413)	Responds to notifications that components are being created or destroyed.

70.1.2.5.1 TDriveComboBox.BuildList Method

Pascal

```
procedure BuildList; virtual;
```

C++

```
virtual __fastcall BuildList();
```

Description

This is BuildList, a member of class TDriveComboBox.

70.1.2.5.2 TDriveComboBox.Click Method

Respond to user click.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Click is called automatically when the user left-clicks the control. Component or application code can call Click to simulate a user mouse click. This is often done in menu actions and hotkey handlers.

As implemented in TControl, Click queries whether the OnClick event handler exists and is different from the OnExecute handler for the control's Action. If this is true, the OnClick event handler is called. If this is false, and the Action OnExecute event handler exists, the Action's Execute method is called. Override Click to provide additional behavior.

See Also

Action (see page 5797)

OnExecute (see page 2716)

Execute (see page 2712)

DblClick (see page 5747)

OnClick (see page 5851)

70.1.2.5.3 TDriveComboBox.CreateWnd Method

Creates the Windows control that represents the combo box.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

The CreateHandle method calls CreateWnd to handle the particulars of creating the combo box window. TCustomCombo overrides the inherited method to ensure that the newly-created window and the TCustomCombo properties correctly reflect each other.

The Windows control for the combo box can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. Thus, CreateWnd may be called more than once during the lifetime of the TCustomCombo instance.

See Also

CreateHandle (see page 6135)

70.1.2.5.4 TDriveComboBox.DrawItem Method

Generates an OnDrawItem event.

Pascal

```
procedure DrawItem(Index: Integer; Rect: TRect; State: TOwnerDrawState); override;
```

C++

```
virtual __fastcall DrawItem(int Index, TRect Rect, TOwnerDrawState State);
```

Description

When Style is csOwnerDrawFixed or csOwnerDrawVariable, DrawItem is called whenever an item in the list needs to be drawn. If an OnDrawItem event handler is assigned, DrawItem generates an OnDrawItem event, passing the Index, Rect, and State parameters to the event handler. Otherwise, OnDrawItem fills the rectangle passed in Rect and draws any text associated with the indexed item, ignoring the State parameter.

Override DrawItem to provide class-specific drawing code for the items in the list.

See Also

OnDrawItem (see page 21224)

Style (see page 21224)

70.1.2.5.5 TDriveComboBox.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

70.1.2.6 TDriveComboBox Properties

The properties of the TDriveComboBox class are listed here.

Public Properties

	Name	Description
	Drive (see page 13426)	Indicates the currently selected drive displayed in the edit portion of the combo box.
	Text (see page 13438)	Contains a text string associated with the control.

Published Properties

	Name	Description
	Align (see page 13414)	Determines how the control aligns within its container (parent control).
	Anchors (see page 13417)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 13418)	Positions to matching list items as you type.
	AutoDropDown (see page 13418)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 13419)	Specifies which edges of the control are beveled.
	BevelInner (see page 13419)	Specifies the cut of the inner bevel.
	BevelKind (see page 13420)	Specifies the control's bevel style.
	BevelOuter (see page 13420)	Specifies the cut of the outer bevel.
	Color (see page 13420)	Specifies the background color of the control.
	Constraints (see page 13421)	Specifies the size constraints for the control.
	Ctl3D (see page 13422)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DirList (see page 13422)	Connects the drive combo box to a directory list box.
	DragCursor (see page 13424)	Indicates the image used to represent the mouse pointer when the control is being dragged.

	DragMode (see page 13424)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 13427)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 13427)	Controls the attributes of text written on or in the control.
	ImeMode (see page 13428)	Determines the behavior of the input method editor (IME).
	ImeName (see page 13429)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	ParentColor (see page 13429)	Determines where a control looks for its color information.
	ParentCtl3D (see page 13430)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 13430)	Determines where a control looks for its font information.
	ParentShowHint (see page 13431)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 13431)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 13436)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 13437)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 13438)	Determines if the user can tab to a control.
	TextCase (see page 13439)	Determines if the volume name in the Text (see page 13438) property appears in uppercase or lowercase.
	Visible (see page 13440)	Determines whether the component appears onscreen.

70.1.2.6.1 TDriveComboBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
```

```
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
```

```
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
```

```
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

70.1.2.6.2 TDriveComboBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

70.1.2.6.3 TDriveComboBox.AutoComplete Property

Positions to matching list items as you type.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

AutoComplete will position to matching list values as you type, saving you keystrokes.

As you type, the ItemIndex property is updated with the list entry that matches the characters you have entered. As you type, different list entries may match, and ItemIndex changes accordingly.

There is a delay of AutoCompleteDelay milliseconds between the characters typed and the autocomplete operation.

See Also

AutoCompleteDelay (see page 21220)

70.1.2.6.4 TDriveComboBox.AutoDropDown Property

Specifies whether the drop-down list drops down automatically in response to user keystrokes.

Pascal

```
property AutoDropDown: Boolean;
```

C++

```
__property Boolean AutoDropDown;
```

Description

When AutoDropDown is true, the combo box automatically drops down its list when the user starts typing a string while the combo box has focus.

When AutoDropDown is false, the user must explicitly use the drop-down button to drop down the combo box list.

See Also

autocomplete (see page 21220)

OnDropDown (see page 21199)

DroppedDown (see page 21185)

70.1.2.6.5 TDriveComboBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

70.1.2.6.6 TDriveComboBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

70.1.2.6.7 TDriveComboBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

70.1.2.6.8 TDriveComboBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

70.1.2.6.9 TDriveComboBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

70.1.2.6.10 TDriveComboBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

70.1.2.6.11 TDriveComboBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

70.1.2.6.12 TDriveComboBox.DirList Property

Connects the drive combo box to a directory list box.

Pascal

```
property DirList: TDirectoryListBox;
```

C++

```
__property TDirectoryListBox DirList;
```

Description

Set DirList to a directory list box control that should update when the user changes the drive. When a new drive is selected in the drive combo box, the specified directory list box updates to display the directory structure and the current directory on the new drive.

Note: If the form contains a file list box as well, it can be updated automatically using the FileList property of the directory list box object.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FileListBox1->FileEdit = Edit1;
  FilterComboBox1->FileList = FileListBox1;
  DirectoryListBox1->FileList = FileListBox1;
  DirectoryListBox1->DirLabel = Label1;
  DriveComboBox1->DirList = DirectoryListBox1;
  Button1->Default = true;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FileListBox1->ApplyFilePath(Edit1->Text);
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FileListBox1.ApplyFilePath(Edit1.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileEdit := Edit1;
  FilterComboBox1.FileList := FileListBox1;
  DirectoryListBox1.FileList := FileListBox1;
  DirectoryListBox1.DirLabel := Label1;
  DriveComboBox1.DirList := DirectoryListBox1;
  Button1.Default := True;
end;
```

See Also

TDirectoryListBox (see page 13313)

TFileListBox (see page 13465)

70.1.2.6.13 TDriveComboBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

70.1.2.6.14 TDriveComboBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
```

```
    else
      return;
    TreeView1->Selected->MoveTo(Item, AttachMode);
  }

  void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
        int X, int Y, TDragState State, bool &Accept)
  {
    Accept = (Source == (TObject *)TreeView1);
  }
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

70.1.2.6.15 TDriveComboBox.Drive Property

Indicates the currently selected drive displayed in the edit portion of the combo box.

Pascal

```
property Drive: Char;
```

C++

```
__property Char Drive;
```

Description

Set Drive to initialize the combo box to a particular drive. Read Drive to obtain the drive that was selected by the user.

When the user uses the drive combo box to select a new drive, the selected drive becomes the value of the Drive property. The value of the Text (see page 13438) property also changes to the new volume name when the Drive property value changes.

C++ Examples:

```
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
```

```
end;
```

See Also

Text (see page 5842)

70.1.2.6.16 TDriveComboBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

70.1.2.6.17 TDriveComboBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}
```

```
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

70.1.2.6.18 TDriveComboBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

70.1.2.6.19 TDriveComboBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

70.1.2.6.20 TDriveComboBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

70.1.2.6.21 TDriveComboBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

70.1.2.6.22 TDriveComboBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

70.1.2.6.23 TDriveComboBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

70.1.2.6.24 TDriveComboBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
```

```
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
```

```
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
```

```
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
```

```
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
```

```
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

70.1.2.6.25 TDriveComboBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
```

```
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

70.1.2.6.26 TDriveComboBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

70.1.2.6.27 TDriveComboBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

70.1.2.6.28 TDriveComboBox.Text Property

Contains a text string associated with the control.

Pascal

```
property Text: TCaption;
```

C++

```
__property TCaption Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
```

```
  copies the text in an edit box into a null-terminated string, and
  puts this string in another edit box when the user clicks the
  button on the form.
  */
  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    int Size = Edit1->GetTextLen(); //Get length of string in Edit1
    Size++;                        //Add room for null character
    char *Buffer = new char[Size];  //Creates Buffer dynamic variable
    Edit1->GetTextBuf(Buffer,Size); //Puts Edit1->Text into Buffer
    Edit2->Text = Buffer;
    delete [] Buffer;
  }
  /*
  Note: the same thing could be accomplished more simply as follows:
  */
  void __fastcall TForm1::Button2Click(TObject *Sender)
  {
    Edit2->Text = Edit1->Text;
  }
```

Delphi Examples:

```
  {
  This example requires two edit boxes and a button.  This example
  copies the text in an edit box into a null-terminated string, and
  puts this string in another edit box when the user clicks the
  button on the form.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    Buffer: PChar;
    Size: Byte;
  begin
    Size := Edit1.GetTextLen;       {Get length of string in Edit1}
    Inc(Size);                      {Add room for null character}
    GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
    Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
    Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
    FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
  end;

  {
  Note: the same thing could be accomplished more simply as follows:
  }
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    Edit2.Text := Edit1.Text;
  end;
```

See Also

Caption (see page 5806)

Lines (see page 21380)

70.1.2.6.29 TDriveComboBox.TextCase Property

Determines if the volume name in the Text (see page 13438) property appears in uppercase or lowercase.

Pascal

```
  property TextCase: TTextCase;
```

C++

```
  __property TTextCase TextCase;
```

Description

Set TextCase to the case that should be used to represent the drive letter that is the value of the Text (see page 13438) property.

Note: Changes made to the TextCase property at design time don't appear until the application runs.

See Also

Drive (see page 13426)

Text (see page 5842)

70.1.2.6.30 TDriveComboBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

70.1.2.7 TDriveComboBox Events

The events of the TDriveComboBox class are listed here.

Published Events

	Name	Description
	OnChange (see page 13442)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 13444)	Occurs when the user clicks the control.
	OnContextPopup (see page 13446)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 13446)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 13447)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13449)	Occurs when the user drags an object over a control.
	OnDropDown (see page 13451)	Occurs when the user opens the drop-down list.

	OnEndDrag (see page 13453)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 13454)	Occurs when a control receives the input focus.
	OnExit (see page 13456)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 13458)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 13461)	Occurs when a key is pressed.
	OnKeyUp (see page 13462)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 13464)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 13464)	Occurs when the user moves the mouse outside of a control.
	OnStartDrag (see page 13464)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

70.1.2.7.1 TDriveComboBox.OnChange Event

Occurs when the user changes the text displayed in the edit region.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action immediately after the user edits the text in the edit region or selects an item from the list. The Text property gives the new value in the edit region.

Note: OnChange only occurs in response to user actions. Changing the Text property programmatically does not trigger an OnChange event.

C++ Examples:

```
/*
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
*/
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  if (pCB)
    PageControl1->ActivePage = (TTabSheet *)(pCB->Items->Objects[pCB->ItemIndex]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
```

```
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
    ComboBox1->Items->AddObject(PageControl1->Pages[i]->Name, PageControl1->Pages[i]);
    ComboBox1->ItemIndex = 0;
  }
}
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to PageControl1.PageCount - 1 do
    ComboBox1.Items.AddObject(PageControl1.Pages[i].Name,
      PageControl1.Pages[i]);
  ComboBox1.ItemIndex := 0;
end;

procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  if (Sender is TComboBox) then
    with (Sender as TComboBox) do
      PageControl1.ActivePage := TTabSheet(Items.Objects[ItemIndex]);
end;
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
```

```
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

See Also

Text (see page 5842)

OnCloseUp (see page 21199)

70.1.2.7.2 TDriveComboBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
```

```
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

70.1.2.7.3 TDriveComboBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

70.1.2.7.4 TDriveComboBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

70.1.2.7.5 TDriveComboBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
```

```
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

70.1.2.7.6 TDriveComboBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
```

```
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

70.1.2.7.7 TDriveComboBox.OnDropDown Event

Occurs when the user opens the drop-down list.

Pascal

```
property OnDropDown: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDropDown;
```

Description

Write an OnDropDown event handler to implement special processing that needs to occur only when the drop-down list is activated.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if (((TForm1 *)Screen->CustomForms[i])->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
```

```
      end;
      Items.EndUpdate; {reenable painting }
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ComboBox1.Sorted := True; { make sure the form names are sorted }
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Form2.Show;
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    Form3.Show;
  end;
```

See Also

DroppedDown (see page 21185)

OnCloseUp (see page 21199)

70.1.2.7.8 TDriveComboBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
```

```
  message displayed depends on whether the dragged label
  control was successfully dropped into and accepted by a
  another control.
  }
  procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
  var
    S: string;
  begin
    S := (Sender as TLabel).Name + ' was dropped... and ';
    if Target <> nil then
      S := S + 'accepted!'
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

70.1.2.7.9 TDriveComboBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
```

```
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

70.1.2.7.10 TDriveComboBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
```

```
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

70.1.2.7.11 TDriveComboBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
```

```
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
  {
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
```

```
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

70.1.2.7.12 TDriveComboBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
```

```
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

70.1.2.7.13 TDriveComboBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
```

```
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

70.1.2.7.14 TDriveComboBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

70.1.2.7.15 TDriveComboBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

70.1.2.7.16 TDriveComboBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these

controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

70.1.3 TFileListBox Class

TFileListBox is a specialized list box that lists all the files in a specified directory.

Class Hierarchy

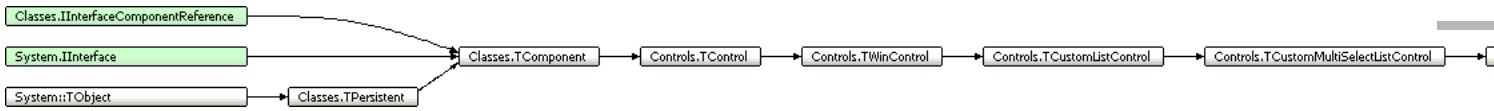

Pascal

```
TFileListBox = class(TCustomListBox);
```

C++

```
class TFileListBox : public TCustomListBox;
```

File

FileCtrl

Description

Add TFileListBox to a form to allow users to select a file. Use TFileListBox along with TDriveComboBox (see page 13396), TFilterComboBox (see page 13547), and TDirectoryListBox (see page 13313) to add full file selection capabilities to a form.

Note: To add a standard Windows file open or save dialog to an application, use TOpenDialog or TSaveDialog instead.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, and
a label on a form. When the user uses the directory list box
to change directories, a message appears and the color of
the form changes if the file AUTOEXEC.BAT is in the new
directory. The code is written in the OnChange event of the
directory list box:
*/
void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
  if (FileListBox1->Items->IndexOf("AUTOEXEC.BAT") > -1)
  {
    Color = clYellow;
    Label1->Caption = "You are in the root directory!";
  }
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, and
a label on a form. When the user uses the directory list box
to change directories, a message appears and the color of
the form changes if the file AUTOEXEC.BAT is in the new
directory. The code is written in the OnChange event of the
directory list box:
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
  if FileListBox1.Items.IndexOf('AUTOEXEC.BAT') > -1 then
  begin
    Color := clYellow;
    Label1.Caption := 'You are in the root directory!';
  end
  else
  begin
    Color := clBtnFace;
    Label1.Caption := '';
  end;
end;
```

See Also

TDirectoryListBox (see page 13313)

TDriveComboBox (see page 13396)

TFilterComboBox (see page 13547)

TOpenDialog (see page 11407)

TSaveDialog (see page 11489)

70.1.3.1 TFileListBox Members

The following tables list the members exposed by TFileListBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnData (see page 21357)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 21357)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 21358)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDrawItem (see page 21358)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMeasureItem (see page 21360)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.

	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	DirBMP (see page 13479)	This is DirBMP, a member of class TFileListBox.
	ExeBMP (see page 13480)	This is ExeBMP, a member of class TFileListBox.
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDirectory (see page 13480)	This is FDirectory, a member of class TFileListBox.
	FDirList (see page 13480)	This is FDirList, a member of class TFileListBox.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFileEdit (see page 13480)	This is FFileEdit, a member of class TFileListBox.
	FFileType (see page 13480)	This is FFileType, a member of class TFileListBox.
	FFilterCombo (see page 13481)	This is FFilterCombo, a member of class TFileListBox.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FLastSel (see page 13481)	This is FLastSel, a member of class TFileListBox.
	FMask (see page 13481)	This is FMask, a member of class TFileListBox.
	FMoving (see page 21321)	This is FMoving, a member of class TCustomListBox.
	FMultiSelect (see page 5956)	This is FMultiSelect, a member of class TCustomMultiSelectListControl.
	FOnChange (see page 13481)	This is FOnChange, a member of class TFileListBox.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FShowGlyphs (see page 13482)	This is FShowGlyphs, a member of class TFileListBox.
	UnknownBMP (see page 13482)	This is UnknownBMP, a member of class TFileListBox.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.

	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 13484)	This is Change, a member of class TFileListBox.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 13484)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 21324)	Initializes the window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 13484)	Creates a Windows control corresponding to the list box component.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DeleteString (see page 21326)	Deletes a string from the list box.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21326)	Saves the list box items in a string list before freeing the underlying Windows control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFindData (see page 21326)	Returns the index of a string in the list box.
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetData (see page 21327)	Returns a string in the list box given its index.
	DoGetDataObject (see page 21327)	Returns the object associated with an item in a virtual list box.

	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragCanceled (see page 21328)	Sends a mouse message canceling the drag-and-drop operation.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawItem (see page 13485)	Generates an OnDrawItem event.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetCount (see page 21329)	Returns the value of the Count (see page 21346) property.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFilePath (see page 13486)	This is GetFilePath, a member of class TFileListBox.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetItemData (see page 21329)	Returns a 32-bit value associated with the specified item in a list box.
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
V	GetItemIndex (see page 21329)	Returns the value of the ItemIndex property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.

	GetSelCount (see page 5956)	Returns the value of the SelCount (see page 5957) property.
	GetSelCount (see page 21330)	Returns the number of selected items.
	GetSelected (see page 21330)	Indicates whether a particular item is selected.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InternalGetItemData (see page 21330)	Retrieves the data associated with an item in the list box.
	InternalSetItemData (see page 21330)	Associates an integer with a specified item in the list box.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyPress (see page 21334)	Responds when the user presses a key.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	LoadRecreateItems (see page 21334)	This is LoadRecreateItems, a member of class TCustomListBox.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MeasureItem (see page 21335)	Generates an OnMeasureItem (see page 21360) event.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 13486)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadBitmaps (see page 13486)	This is ReadBitmaps, a member of class TFileListBox.
	ReadFileNames (see page 13487)	This is ReadFileNames, a member of class TFileListBox.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetContent (see page 21335)	Removes all of the items from a list box.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.

	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
V	SaveRecreateItems (see page 21336)	This is SaveRecreateItems, a member of class TCustomListBox.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItemData (see page 21336)	Sets the 32-bit value associated with the specified item in a list box.
A	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
V	SetItemIndex (see page 21336)	Gives focus to the item with a specified index.
A	SetMultiSelect (see page 5956)	Sets the value of the MultiSelect (see page 5957) property.
V	SetMultiSelect (see page 21337)	Sets the MultiSelect property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WndProc (see page 21337)	Handles Windows messages.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 21343)	Determines whether the list box has a border.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Columns (see page 21346)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Sorted (see page 21353)	Specifies whether the items in a list box are arranged alphabetically.
	Style (see page 21354)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabWidth (see page 21355)	Specifies the size of the tabs in the list box.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
A	AddItem (see page 5940)	Adds an item to the list control.
V	AddItem (see page 21322)	Adds an item to the list box
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	ApplyFilePath (see page 13482)	Updates the file list box to represent a fully qualified file name.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 5940)	Deletes all of the items from the list control.
V	Clear (see page 21322)	Deletes all items from the list box.
A	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
V	ClearSelection (see page 21323)	Deselects all selected items in the list box, if possible.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
A	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
V	CopySelection (see page 21323)	Copies the selected items to another list-type control
V	Create (see page 13478)	Creates an instance of TFileListBox (see page 13465).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	DeleteSelected (see page 5941)	Deletes all the selected items.
V	DeleteSelected (see page 21325)	Deletes the selected items from the list box
V	Destroy (see page 13479)	Destroys an instance of TFileListBox (see page 13465).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ItemAtPos (see page 21331)	Returns the index of the list box item indicated by the coordinates of a point on the control.
	ItemRect (see page 21333)	Returns the rectangle that surrounds the item specified in the Item parameter.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	MoveSelection (see page 5942)	Moves all selected items to another list control.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
A	SelectAll (see page 5943)	Selects all items or all text in the selected item.
V	SelectAll (see page 21336)	Selects the entire block of text in the list box.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Update (see page 13487)	Refreshes the directory list for the file list box.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AutoCompleteDelay (see page 21342)	Specifies the delay between a key press and an attempt to autocomplete the field.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 21343)	Provides a drawing surface when implementing a handler for the OnDrawItem (see page 21358) event of an owner-draw list box.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.

R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Count (see page 21346)	Indicates the number of items in the list box.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Directory (see page 13496)	Specifies the directory that contains all the files listed in the file list box.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Drive (see page 13499)	Specifies the drive which contains the files listed in the list box.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
	FileName (see page 13502)	Indicates the name of the selected file in the list box, including the path name.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	Items (see page 21348)	Contains the strings that appear in the list box.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ScrollWidth (see page 21351)	Specifies the width, in pixels, by which the list box can scroll horizontally.
R	SelCount (see page 5957)	Indicates the number of selected items.
	Selected (see page 21351)	Indicates whether a particular item is selected.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TopIndex (see page 21356)	Specifies the index number of the item that appears at the top of the list box.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 13519)	Occurs immediately after the user changes the selected file in the list.
	OnClick (see page 13520)	Occurs when the user clicks the control.
	OnContextPopup (see page 13521)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

	OnDblClick (see page 13522)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 13522)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13525)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 13527)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 13528)	Occurs when a control receives the input focus.
	OnExit (see page 13531)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 13532)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 13535)	Occurs when a key is pressed.
	OnKeyUp (see page 13536)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 13538)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 13539)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 13541)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 13541)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 13541)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 13544)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDrag (see page 13546)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 13488)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 13492)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 13492)	Determines whether the user can give focus to items by typing in the list.
	BevelEdges (see page 13493)	Specifies which edges of the control are beveled.
	BevelInner (see page 13493)	Specifies the cut of the inner bevel.
	BevelKind (see page 13493)	Specifies the control's bevel style.
	BevelOuter (see page 13494)	Specifies the cut of the outer bevel.
	Color (see page 13494)	Specifies the background color of the control.
	Constraints (see page 13495)	Specifies the size constraints for the control.
	Ctl3D (see page 13496)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 13497)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 13498)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 13500)	Controls whether the control responds to mouse, keyboard, and timer events.
	ExtendedSelect (see page 13501)	This is ExtendedSelect, a member of class TFileListBox.
	FileEdit (see page 13501)	Links the directory list box to an edit control that displays the currently selected file.
	FileType (see page 13503)	Determines which files are displayed in the file list box based on the attributes of the files.
	Font (see page 13503)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 13504)	Determines the behavior of the input method editor (IME).

	ImeName (see page 13505)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 13505)	Determines whether the list box displays the partial items.
	ItemHeight (see page 13506)	Specifies the height in pixels of an item in an owner-draw list box.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Mask (see page 13506)	Limits which files are displayed in the file list box.
	MultiSelect (see page 13507)	Specifies whether the user can select more than one item.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentColor (see page 13508)	Determines where a control looks for its color information.
	ParentCtl3D (see page 13508)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 13509)	Determines where a control looks for its font information.
	ParentShowHint (see page 13509)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 13510)	Identifies the pop-up menu associated with the control.
	ShowGlyphs (see page 13514)	Determines whether glyphs (bitmaps) appear next to the file names listed in the file list box.
	ShowHint (see page 13515)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 13516)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 13517)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 13517)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

70.1.3.2 TFileListBox.Create Constructor

Creates an instance of TFileListBox (see page 13465).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TFileListBox(TComponent * AOwner);
```

Description

Call Create to create a TFileListBox (see page 13465) object at runtime. File list boxes placed on a form at design time are created automatically.

See Also

Directory (see page 13496)

Drive (see page 13499)

FileType (see page 13503)

Mask (see page 13506)

MultiSelect

Sorted (see page 21353)

Create (see page 21320)

Width (see page 5849)

70.1.3.3 TFileListBox.Destroy Destructor

Destroys an instance of TFileListBox (see page 13465).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TFileListBox();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the file list box reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

70.1.3.4 TFileListBox Fields

The fields of the TFileListBox class are listed here.

Protected Fields

	Name	Description
	DirBMP (see page 13479)	This is DirBMP, a member of class TFileListBox.
	ExeBMP (see page 13480)	This is ExeBMP, a member of class TFileListBox.
	FDirectory (see page 13480)	This is FDirectory, a member of class TFileListBox.
	FDirList (see page 13480)	This is FDirList, a member of class TFileListBox.
	FFileEdit (see page 13480)	This is FFileEdit, a member of class TFileListBox.
	FFileType (see page 13480)	This is FFileType, a member of class TFileListBox.
	FFilterCombo (see page 13481)	This is FFilterCombo, a member of class TFileListBox.
	FLastSel (see page 13481)	This is FLastSel, a member of class TFileListBox.
	FMask (see page 13481)	This is FMask, a member of class TFileListBox.
	FOnChange (see page 13481)	This is FOnChange, a member of class TFileListBox.
	FShowGlyphs (see page 13482)	This is FShowGlyphs, a member of class TFileListBox.
	UnknownBMP (see page 13482)	This is UnknownBMP, a member of class TFileListBox.

70.1.3.4.1 TFileListBox.DirBMP Field

Pascal

```
DirBMP: TBitmap;
```

C++

```
TBitmap DirBMP;
```

Description

This is DirBMP, a member of class TFileListBox.

70.1.3.4.2 TFileListBox.ExeBMP Field

Pascal

```
ExeBMP: TBitmap;
```

C++

```
TBitmap ExeBMP;
```

Description

This is ExeBMP, a member of class TFileListBox.

70.1.3.4.3 TFileListBox.FDirectory Field

Pascal

```
FDirectory: string;
```

C++

```
AnsiString FDirectory;
```

Description

This is FDirectory, a member of class TFileListBox.

70.1.3.4.4 TFileListBox.FDirList Field

Pascal

```
FDirList: TDirectoryListBox;
```

C++

```
TDirectoryListBox FDirList;
```

Description

This is FDirList, a member of class TFileListBox.

70.1.3.4.5 TFileListBox.FFileEdit Field

Pascal

```
FFileEdit: TEdit;
```

C++

```
TEdit FFileEdit;
```

Description

This is FFileEdit, a member of class TFileListBox.

70.1.3.4.6 TFileListBox.FFileType Field

Pascal

```
FFileType: TFileType;
```

C++

```
TFileType FFileType;
```

Description

This is FFileType, a member of class TFileListBox.

70.1.3.4.7 TFileListBox.FFilterCombo Field

Pascal

```
FFilterCombo: TFilterComboBox;
```

C++

```
TFilterComboBox FFilterCombo;
```

Description

This is FFilterCombo, a member of class TFileListBox.

70.1.3.4.8 TFileListBox.FLastSel Field

Pascal

```
FLastSel: Integer;
```

C++

```
int FLastSel;
```

Description

This is FLastSel, a member of class TFileListBox.

70.1.3.4.9 TFileListBox.FMask Field

Pascal

```
FMask: string;
```

C++

```
AnsiString FMask;
```

Description

This is FMask, a member of class TFileListBox.

70.1.3.4.10 TFileListBox.FOnChange Field

Pascal

```
FOnChange: TNotifyEvent;
```

C++

```
TNotifyEvent FOnChange;
```

Description

This is FOnChange, a member of class TFileListBox.

70.1.3.4.11 TFileListBox.FShowGlyphs Field

Pascal

```
FShowGlyphs: Boolean;
```

C++

```
Boolean FShowGlyphs;
```

Description

This is FShowGlyphs, a member of class TFileListBox.

70.1.3.4.12 TFileListBox.UnknownBMP Field

Pascal

```
UnknownBMP: TBitmap;
```

C++

```
TBitmap UnknownBMP;
```

Description

This is UnknownBMP, a member of class TFileListBox.

70.1.3.5 TFileListBox Methods

The methods of the TFileListBox class are listed here.

Protected Methods

	Name	Description
	Change (see page 13484)	This is Change, a member of class TFileListBox.
	Click (see page 13484)	Respond to user click.
	CreateWnd (see page 13484)	Creates a Windows control corresponding to the list box component.
	DrawItem (see page 13485)	Generates an OnDrawItem event.
	GetFilePath (see page 13486)	This is GetFilePath, a member of class TFileListBox.
	Notification (see page 13486)	Responds to notifications that components are being created or destroyed.
	ReadBitmaps (see page 13486)	This is ReadBitmaps, a member of class TFileListBox.
	ReadFileNames (see page 13487)	This is ReadFileNames, a member of class TFileListBox.

Public Methods

	Name	Description
	ApplyFilePath (see page 13482)	Updates the file list box to represent a fully qualified file name.
	Update (see page 13487)	Refreshes the directory list for the file list box.

70.1.3.5.1 TFileListBox.ApplyFilePath Method

Updates the file list box to represent a fully qualified file name.

Pascal

```
procedure ApplyFilePath(const EditText: string); virtual;
```

C++

```
virtual __fastcall ApplyFilePath(const AnsiString EditText);
```

Description

Call ApplyFilePath to change the Directory (see page 13496), Drive (see page 13499), FileName (see page 13502), and Mask (see page 13506) properties so that they match the values implied by the EditText parameter. For example, if an edit control is linked to the file list box using the FileEdit (see page 13501) property, call ApplyFilePath from the OnChange (see page 13519) event handler of the edit control to update the file list box whenever the user types a value in the edit control.

ApplyFilePath updates the Mask (see page 13506) property only if the EditText parameter includes wildcards. If the form contains a TDirectoryListBox (see page 13313) object that specifies this file list box object in its FileList property, the Directory (see page 13496) and Drive (see page 13499) properties of the directory list box object are updated as well. If the form also includes a TDriveComboBox (see page 13396) object that specifies the directory list box in its DirList property, the Drive (see page 13499) property of the drive combo box will also be updated. However, the Mask (see page 13506) property of any TFilterComboBox (see page 13547) will not be updated.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FileListBox1->FileEdit = Edit1;
  FilterComboBox1->FileList = FileListBox1;
  DirectoryListBox1->FileList = FileListBox1;
  DirectoryListBox1->DirLabel = Label1;
  DriveComboBox1->DirList = DirectoryListBox1;
  Button1->Default = true;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FileListBox1->ApplyFilePath(Edit1->Text);
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FileListBox1.ApplyFilePath(Edit1.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileEdit := Edit1;
  FilterComboBox1.FileList := FileListBox1;
  DirectoryListBox1.FileList := FileListBox1;
  DirectoryListBox1.DirLabel := Label1;
  DriveComboBox1.DirList := DirectoryListBox1;
  Button1.Default := True;
end;
```

See Also

Directory (see page 13496)

Drive (see page 13499)

FileEdit (see page 13501)

FileName (see page 13502)

Mask (see page 13506)

TDirectoryListBox (see page 13313)

TDriveComboBox (see page 13396)

TFilterComboBox (see page 13547)

70.1.3.5.2 TFileListBox.Change Method

Pascal

```
procedure Change; virtual;
```

C++

```
virtual __fastcall Change();
```

Description

This is Change, a member of class TFileListBox.

70.1.3.5.3 TFileListBox.Click Method

Respond to user click.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Click is called automatically when the user left-clicks the control. Component or application code can call Click to simulate a user mouse click. This is often done in menu actions and hotkey handlers.

As implemented in TControl, Click queries whether the OnClick event handler exists and is different from the OnExecute handler for the control's Action. If this is true, the OnClick event handler is called. If this is false, and the Action OnExecute event handler exists, the Action's Execute method is called. Override Click to provide additional behavior.

See Also

Action (see page 5797)

OnExecute (see page 2716)

Execute (see page 2712)

DblClick (see page 5747)

OnClick (see page 5851)

70.1.3.5.4 TFileListBox.CreateWnd Method

Creates a Windows control corresponding to the list box component.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the list box must be displayed. CreateWnd calls the inherited CreateWnd method, which calls

CreateParams to initialize the window-creation parameters.

CreateWindowHandle to create the window handle for the control.

CreateWnd additionally sets up the list box position, tabs, items, and columns.

See Also

Columns (see page 21346)

CreateHandle (see page 6135)

CreateParams (see page 21324)

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

ItemIndex

Items (see page 21348)

TabWidth (see page 21355)

TopIndex (see page 21356)

70.1.3.5.5 TFileListBox.DrawItem Method

Generates an OnDrawItem event.

Pascal

```
procedure DrawItem(Index: Integer; Rect: TRect; State: TOwnerDrawState); override;
```

C++

```
virtual __fastcall DrawItem(int Index, TRect Rect, TOwnerDrawState State);
```

Description

DrawItem is called whenever a visual aspect of an owner-draw list box changes. The list box calls DrawItem for each visible item in its list, passing the index of the item in the Index parameter.

By default, the DrawItem method for a list box calls any event handler attached to the component's OnDrawItem event. If there is no handler attached, DrawItem fills the rectangle passed in Rect and draws any text associated with the indexed item. The default drawing ignores the State parameter, although State is passed along to attached event handlers.

See Also

ItemHeight (see page 21347)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

Style (see page 21354)

70.1.3.5.6 TFileListBox.GetFilePath Method

Pascal

```
function GetFilePath: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetFilePath();
```

Description

This is GetFilePath, a member of class TFileListBox.

70.1.3.5.7 TFileListBox.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

70.1.3.5.8 TFileListBox.ReadBitmaps Method

Pascal

```
procedure ReadBitmaps; virtual;
```

C++

```
virtual __fastcall ReadBitmaps();
```

Description

This is ReadBitmaps, a member of class TFileListBox.

70.1.3.5.9 TFileListBox.ReadFileNames Method

Pascal

```
procedure ReadFileNames; virtual;
```

C++

```
virtual __fastcall ReadFileNames();
```

Description

This is ReadFileNames, a member of class TFileListBox.

70.1.3.5.10 TFileListBox.Update Method

Refreshes the directory list for the file list box.

Pascal

```
procedure Update; reintroduce;
```

C++

```
__fastcall Update();
```

Description

Call Update whenever the files listed by the file list box may be out of date. For example, if an application creates, renames, or deletes files, calling Update will ensure that those changes are reflected in the file list box.

The list box is updated automatically when the Drive (see page 13499), Directory (see page 13496), FileType (see page 13503), or Mask (see page 13506) property is changed.

See Also

Directory (see page 13496)

Drive (see page 13499)

FileType (see page 13503)

Mask (see page 13506)

70.1.3.6 TFileListBox Properties

The properties of the TFileListBox class are listed here.

Public Properties

	Name	Description
	Directory (see page 13496)	Specifies the directory that contains all the files listed in the file list box.
	Drive (see page 13499)	Specifies the drive which contains the files listed in the list box.
	FileName (see page 13502)	Indicates the name of the selected file in the list box, including the path name.

Published Properties

	Name	Description
	Align (see page 13488)	Determines how the control aligns within its container (parent control).
	Anchors (see page 13492)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 13492)	Determines whether the user can give focus to items by typing in the list.
	BevelEdges (see page 13493)	Specifies which edges of the control are beveled.

	BevelInner (see page 13493)	Specifies the cut of the inner bevel.
	BevelKind (see page 13493)	Specifies the control's bevel style.
	BevelOuter (see page 13494)	Specifies the cut of the outer bevel.
	Color (see page 13494)	Specifies the background color of the control.
	Constraints (see page 13495)	Specifies the size constraints for the control.
	Ctl3D (see page 13496)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 13497)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 13498)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 13500)	Controls whether the control responds to mouse, keyboard, and timer events.
	ExtendedSelect (see page 13501)	This is ExtendedSelect, a member of class TFileListBox.
	FileEdit (see page 13501)	Links the directory list box to an edit control that displays the currently selected file.
	FileType (see page 13503)	Determines which files are displayed in the file list box based on the attributes of the files.
	Font (see page 13503)	Controls the attributes of text written on or in the control.
	ImeMode (see page 13504)	Determines the behavior of the input method editor (IME).
	ImeName (see page 13505)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 13505)	Determines whether the list box displays the partial items.
	ItemHeight (see page 13506)	Specifies the height in pixels of an item in an owner-draw list box.
	Mask (see page 13506)	Limits which files are displayed in the file list box.
	MultiSelect (see page 13507)	Specifies whether the user can select more than one item.
	ParentColor (see page 13508)	Determines where a control looks for its color information.
	ParentCtl3D (see page 13508)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 13509)	Determines where a control looks for its font information.
	ParentShowHint (see page 13509)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 13510)	Identifies the pop-up menu associated with the control.
	ShowGlyphs (see page 13514)	Determines whether glyphs (bitmaps) appear next to the file names listed in the file list box.
	ShowHint (see page 13515)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 13516)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 13517)	Determines if the user can tab to a control.
	Visible (see page 13517)	Determines whether the component appears onscreen.

70.1.3.6.1 TFileListBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
```

```
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
```

```
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
```

```
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

70.1.3.6.2 TFileListBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

70.1.3.6.3 TFileListBox.AutoComplete Property

Determines whether the user can give focus to items by typing in the list.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

Use AutoComplete to specify whether the list box automatically updates the ItemIndex property in response to user keystrokes. When AutoComplete is true, the item with focus changes to the first item that begins with the sequence of characters that the user typed. As the user types more characters, ItemIndex may change to a later item because the string to match becomes

longer.

See Also

ItemIndex

70.1.3.6.4 TFileListBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

70.1.3.6.5 TFileListBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

70.1.3.6.6 TFileListBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

70.1.3.6.7 TFileListBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

70.1.3.6.8 TFileListBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

70.1.3.6.9 TFileListBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict,

the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

70.1.3.6.10 TFileListBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

70.1.3.6.11 TFileListBox.Directory Property

Specifies the directory that contains all the files listed in the file list box.

Pascal

```
property Directory: string;
```

C++

```
__property AnsiString Directory;
```

Description

Set Directory to update the file list box so that it displays the files in a new directory. If the form includes a TDirectoryListBox (see page 13313) object, set the FileList property of the directory list box to have the Directory property updated automatically when the user selects a new directory.

C++ Examples:

```
/*
The following example assumes that a drive combo box, a file
```

```
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

See Also

Drive (see page 13499)

Directory (see page 13344)

FileList (see page 13350)

70.1.3.6.12 TFileListBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

70.1.3.6.13 TFileListBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

70.1.3.6.14 TFileListBox.Drive Property

Specifies the drive which contains the files listed in the list box.

Pascal

```
property Drive: char;
```

C++

```
__property char Drive;
```

Description

Set the value of Drive to the drive that contains the files that the list box should display. When the value of Drive changes, the

Directory (see page 13496) value also changes to the current directory on the specified drive.

If the form includes a TDirectoryListBox (see page 13313) object, set the FileList property of the directory list box to have the Drive property updated automatically when the Drive property of the directory list box changes.

C++ Examples:

```
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

See Also

Directory (see page 13496)

Drive (see page 13349)

FileList (see page 13350)

70.1.3.6.15 TFileListBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

70.1.3.6.16 TFileListBox.ExtendedSelect Property

Pascal

```
property ExtendedSelect: Boolean;
```

C++

```
__property Boolean ExtendedSelect;
```

Description

This is ExtendedSelect, a member of class TFileListBox.

70.1.3.6.17 TFileListBox.FileEdit Property

Links the directory list box to an edit control that displays the currently selected file.

Pascal

```
property FileEdit: TEdit;
```

C++

```
__property TEdit FileEdit;
```

Description

Set FileEdit to an edit control that displays the selected file from the file list box, as is commonly done in Open and Save dialog boxes. If no file is selected in the file list box, the text of the edit control is the current value of the file list box's Mask (see page 13506) property.

Note: If the user changes the text of the edit control, the results are not automatically reflected in the file list box. Use the ApplyFilePath (see page 13482) method to update the file list box when the edit control's text changes.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
```

```
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FileListBox1->FileEdit = Edit1;
  FilterComboBox1->FileList = FileListBox1;
  DirectoryListBox1->FileList = FileListBox1;
  DirectoryListBox1->DirLabel = Label1;
  DriveComboBox1->DirList = DirectoryListBox1;
  Button1->Default = true;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FileListBox1->ApplyFilePath(Edit1->Text);
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FileListBox1.ApplyFilePath(Edit1.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileEdit := Edit1;
  FilterComboBox1.FileList := FileListBox1;
  DirectoryListBox1.FileList := FileListBox1;
  DirectoryListBox1.DirLabel := Label1;
  DriveComboBox1.DirList := DirectoryListBox1;
  Button1.Default := True;
end;
```

See Also

ApplyFilePath (see page 13482)

FileName (see page 13502)

Mask (see page 13506)

70.1.3.6.18 TFileListBox.FileName Property

Indicates the name of the selected file in the list box, including the path name.

Pascal

```
property FileName: string;
```

C++

```
__property AnsiString FileName;
```

Description

Set FileName to initialize the file list box to a particular file. Read FileName to obtain the name of the file the user selected.

If the FileEdit (see page 13501) property is set, the linked edit control displays the value of FileName whenever the FileName

property changes.

See Also

FileEdit (see page 13501)

FileType (see page 13503)

70.1.3.6.19 TFileListBox.FileType Property

Determines which files are displayed in the file list box based on the attributes of the files.

Pascal

```
property FileType: TFileType;
```

C++

```
__property TFileType FileType;
```

Description

Set FileType to the type of files the list box should display. FileType can specify more than one type of file. For example, if the value of FileType is a set containing the values ftReadOnly and ftHidden, files that have either the read-only or the hidden attributes are displayed in the list box.

To set the FileType property using the Object Inspector, click the FileType property to see the attribute values. Then set each value to True or False true or false to specify whether it is included in the FileType set.

Note: The files that are listed can also be limited by the form of their names. To restrict the files to particular names, use the Mask (see page 13506) property or include a TFilterComboBox (see page 13547) on the form.

See Also

Mask (see page 13506)

TFilterComboBox (see page 13547)

70.1.3.6.20 TFileListBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

70.1.3.6.21 TFileListBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

70.1.3.6.22 TFileListBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

70.1.3.6.23 TFileListBox.IntegralHeight Property

Determines whether the list box displays the partial items.

Pascal

```
property IntegralHeight: Boolean;
```

C++

```
__property Boolean IntegralHeight;
```

Description

IntegralHeight controls the height of the list box itself.

When IntegralHeight is true, and the Style property is lsOwnerDrawFixed, the list box height will always be a multiple of the ItemHeight property value. In this case, items will never be partially displayed.

If IntegralHeight is false, the list box height can be set to any value, but the last item may not be fully displayed.

If the Style property value of the list box is lbOwnerDrawVariable, setting the IntegralHeight property to true has no effect.

See Also

ItemHeight (see page 21347)

Items (see page 21348)

Style (see page 21354)

Using TDBListBox and TDBComboBox

70.1.3.6.24 TFileListBox.ItemHeight Property

Specifies the height in pixels of an item in an owner-draw list box.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Use ItemHeight to control the height of an item in a list box when the Style property is lbOwnerDrawFixed. The ItemHeight property can also affect the height of the list box itself. When the IntegralHeight property is true, the height of the list box is always an increment of the ItemHeight value.

If the Style property is lbStandard or lbOwnerDrawVariable, the value of ItemHeight is ignored.

See Also

DrawItem (see page 21328)

IntegralHeight (see page 21347)

Items (see page 21348)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

Style (see page 21354)

Using TDBListBox and TDBComboBox

70.1.3.6.25 TFileListBox.Mask Property

Limits which files are displayed in the file list box.

Pascal

```
property Mask: string;
```

C++

```
__property AnsiString Mask;
```

Description

Set Mask to a regular expression to limit the list box to files that match the mask. The value of the mask is a file name that may include wildcards. The asterisk (*) is a wildcard which matches any number of arbitrary characters. The question mark (?) is a wildcard which matches a single arbitrary character. The file mask *.* displays all files, which is the default value.

To specify multiple file masks, separate the file mask specifications with semicolons. For example, *.DLL;*.EXE.

If the form includes a TFilterComboBox (see page 13547) object, set the FileList property of the filter combo box to have the Mask property updated automatically when the Mask property of the filter combo box changes.

C++ Examples:

```
/*
This example uses a filter combo box and a file list box on
a form. When the user selects a filter in the filter combo
box, the file list box displays only files which match the
selected mask:
*/
void __fastcall TForm1::FilterComboBox1Change(TObject *Sender)
{
  FileListBox1->Mask = FilterComboBox1->Mask;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FilterComboBox1->Filter = "All files (*.*)|*.*| Executable files (*.exe)|*.exe";
}
```

Delphi Examples:

```
{
This example uses a filter combo box and a file list box on
a form. When the user selects a filter in the filter combo
box, the file list box displays only files which match the
selected mask:
}
procedure TForm1.FilterComboBox1Change(Sender: TObject);
begin
  FileListBox1.Mask := FilterComboBox1.Mask;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FilterComboBox1.Filter := 'All files (*.*)|*.*| Pascal files (*.pas)|*.pas';
end;
```

See Also

FileEdit (see page 13501)

FileName (see page 13502)

FileList (see page 13574)

Mask (see page 13579)

70.1.3.6.26 TFileListBox.MultiSelect Property

Specifies whether the user can select more than one item.

Pascal

```
property MultiSelect: Boolean;
```

C++

```
__property Boolean MultiSelect;
```

Description

Use MultiSelect to specify whether the user can select more than one item. When MultiSelect is true, the user can select multiple items in the control, and the SelCount property indicates the number of selected items. When MultiSelect is false, the user can only select one item, which is indicated by the ItemIndex property.

Note: To implement the MultiSelect property in a TCustomMultiSelectListControl descendant, override the protected SetMultiSelect method.

See Also

SelCount (see page 5957)

ItemIndex (see page 5944)

70.1.3.6.27 TFileListBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

70.1.3.6.28 TFileListBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

70.1.3.6.29 TFileListBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

70.1.3.6.30 TFileListBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

70.1.3.6.31 TFileListBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
```

```
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
```

```
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
```

```
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

70.1.3.6.32 TFileListBox.ShowGlyphs Property

Determines whether glyphs (bitmaps) appear next to the file names listed in the file list box.

Pascal

```
property ShowGlyphs: Boolean;
```

C++

```
__property Boolean ShowGlyphs;
```

Description

Set ShowGlyphs to true to display icons next to the files in the list that indicate the file type. Set ShowGlyphs to false to list only the file names.

70.1.3.6.33 TFileListBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

70.1.3.6.34 TFileListBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

70.1.3.6.35 TFileListBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

70.1.3.6.36 TFileListBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
```

```
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

70.1.3.7 TFileListBox Events

The events of the TFileListBox class are listed here.

Published Events

	Name	Description
	OnChange (see page 13519)	Occurs immediately after the user changes the selected file in the list.
	OnClick (see page 13520)	Occurs when the user clicks the control.
	OnContextPopup (see page 13521)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 13522)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 13522)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13525)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 13527)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 13528)	Occurs when a control receives the input focus.
	OnExit (see page 13531)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 13532)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 13535)	Occurs when a key is pressed.
	OnKeyUp (see page 13536)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 13538)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 13539)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 13541)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 13541)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 13541)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 13544)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDrag (see page 13546)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

70.1.3.7.1 TFileListBox.OnChange Event

Occurs immediately after the user changes the selected file in the list.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action when the FileName (see page 13502) property changes. If the FileEdit (see page 13501) property has been set, the edit control is updated before the OnChange event.

See Also

FileEdit (see page 13501)

FileName (see page 13502)

70.1.3.7.2 TFileListBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
```

```
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

70.1.3.7.3 TFileListBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

70.1.3.7.4 TFileListBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

70.1.3.7.5 TFileListBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

70.1.3.7.6 TFileListBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

70.1.3.7.7 TFileListBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
```

```
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

70.1.3.7.8 TFileListBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter

event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

70.1.3.7.9 TFileListBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
```

```
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

70.1.3.7.10 TFileListBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly

fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
```

```
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
```

```
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

70.1.3.7.11 TFileListBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/
```

```
void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

70.1.3.7.12 TFileListBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
```

```
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
```

```
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

70.1.3.7.13 TFileListBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

70.1.3.7.14 TFileListBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
```

```
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
      }
      else
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
```

```
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

70.1.3.7.15 TFileListBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

70.1.3.7.16 TFileListBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

70.1.3.7.17 TFileListBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
```

```
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

70.1.3.7.18 TFileListBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
```

```
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
```

```
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

70.1.3.7.19 TFileListBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

70.1.4 TFilterComboBox Class

TFilterComboBox is a specialized combo box that presents the user with a choice of file filters.

Class Hierarchy

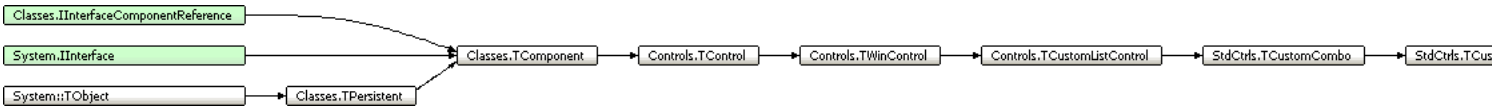

Pascal

```
TFilterComboBox = class(TCustomComboBox);
```

C++

```
class TFilterComboBox : public TCustomComboBox;
```

File

FileCtrl

Description

Add a filter combo box to a form that includes a file selection control to provide a set of predefined file filters. Use TFilterComboBox along with TDriveComboBox (see page 13396), TDirectoryListBox (see page 13313), and TFileListBox (see page 13465) to add full file selection capabilities to a form.

To add a standard Win32 file open or save dialog to an application, use TOpenDialog or TSaveDialog instead.

See Also

TDirectoryListBox (see page 13313)

TDriveComboBox (see page 13396)

TFileListBox (see page 13465)

70.1.4.1 TFilterComboBox Members

The following tables list the members exposed by TFilterComboBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnCloseUp (see page 21199)	Occurs when the drop-down list closes up due to some user action.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDrawItem (see page 21224)	Occurs when an item in an owner-draw combo box needs to be displayed.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.

	OnMeasureItem (see page 21225)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnSelect (see page 21201)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefEditProc (see page 21167)	This is FDefEditProc, a member of class TCustomCombo.
	FDefListProc (see page 21167)	This is FDefListProc, a member of class TCustomCombo.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FDropHandle (see page 21168)	This is FDropHandle, a member of class TCustomCombo.
	FDroppingDown (see page 21168)	This is FDroppingDown, a member of class TCustomCombo.
	FEditHandle (see page 21168)	This is FEditHandle, a member of class TCustomCombo.
	FEditInstance (see page 21168)	This is FEditInstance, a member of class TCustomCombo.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFocusChanged (see page 21168)	This is FFocusChanged, a member of class TCustomCombo.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FIsFocused (see page 21169)	This is FIsFocused, a member of class TCustomCombo.
	FListHandle (see page 21169)	This is FListHandle, a member of class TCustomCombo.
	FListInstance (see page 21169)	This is FListInstance, a member of class TCustomCombo.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveIndex (see page 21169)	This is FSaveIndex, a member of class TCustomCombo.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustDropDown (see page 21171)	Adjusts the size and position of the drop down list.

	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	BuildList (see page 13560)	This is BuildList, a member of class TFilterComboBox.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 13560)	Generates an OnChange event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 13561)	Respond to user click.
	CloseUp (see page 21172)	Generates an OnCloseUp (see page 21199) event.
	ComboWndProc (see page 21173)	Responds to Windows messages sent to the edit control or drop-down list.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 21215)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 13561)	Creates the Windows control that represents the combo box.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWindowHandle (see page 21174)	Destroys all of the windows controls that implement the combo box.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21215)	Destroys the Windows control that represents the combo box.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.

	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawItem (see page 21216)	Generates an OnDrawItem (see page 21224) event.
	DropDown (see page 21175)	Generates an OnDropDown (see page 21199) event.
V	EditWndProc (see page 21175)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetDroppedDown (see page 21176)	Returns the value of the DroppedDown (see page 21185) property.

V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
A	GetItemCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
V	GetItemCount (see page 21216)	Returns the number of items in the Items (see page 21187) property list.
A	GetItemHt (see page 21177)	Returns the value of the ItemHeight (see page 21186) property.
V	GetItemHt (see page 21216)	Returns the value of the ItemHeight (see page 21186) property.
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
V	GetItemIndex (see page 21177)	Returns the value of the ItemIndex property.
A	GetItemsClass (see page 21177)	Indicates the class that implements the Items (see page 21187) property.
V	GetItemsClass (see page 21217)	Returns the class used to implement the Items (see page 21187) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelLength (see page 21178)	Returns the value of the SelLength (see page 21190) property.
	GetSelStart (see page 21178)	Returns the value of the SelStart (see page 21193) property.
	GetSelText (see page 21217)	Returns the value of the SelText (see page 21222) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
V	KeyPress (see page 21217)	Responds to WM_CHAR messages sent to the combo box.
	KeyUp (see page 6164)	Respond to released key.
	ListWndProc (see page 21178)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 21179)	Initializes the combo box after its form is loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
V	MeasureItem (see page 21218)	Generates an OnMeasureItem (see page 21225) event.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 13562)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.

V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Select (see page 21179)	Generates an OnSelect (see page 21201) event.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectItem (see page 21218)	Selects a specified item from the drop-down list.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetDropDownCount (see page 21180)	Sets the value of the DropDownCount (see page 21185) property.
	SetDroppedDown (see page 21180)	Sets the value of the DroppedDown (see page 21185) property.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetItemHeight (see page 21181)	Sets the value of the ItemHeight (see page 21186) property.
A	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
V	SetItemIndex (see page 21181)	Sets the value of the ItemIndex property.
V	SetItems (see page 21181)	Sets the value of the Items (see page 21187) property.
	SetMaxLength (see page 21182)	Sets the value of the MaxLength (see page 21189) property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.

	SetSelLength (see page 21182)	Sets the value of the SelLength (see page 21190) property.
	SetSelStart (see page 21182)	Sets the value of the SelStart (see page 21193) property.
V	SetStyle (see page 21219)	Sets the Style (see page 21224) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WndProc (see page 21183)	Processes messages received by the combo box.
V	WndProc (see page 21219)	Processes messages received by the combo box.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DropDownCount (see page 21185)	Specifies the maximum number of items displayed in the drop-down list.
R	EditHandle (see page 21185)	Returns the window handle for the edit portion of the combo box.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	ItemCount (see page 21186)	Specifies the number of items in the drop-down list.
	ItemHeight (see page 21186)	Specifies the height, in pixels, of the items in the drop-down list.
R	ListHandle (see page 21189)	Provides access to the window handle of the list of items in the combo box.
	MaxLength (see page 21189)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.

	Sorted (see page 21222)	Determines whether the list portion of the combo box is alphabetized.
	Style (see page 21224)	Determines the display style of the combo box.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddItem (see page 5940)	Adds an item to the list control.
	AddItem (see page 21170)	Adds an item to the drop-down list of items.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 5940)	Deletes all of the items from the list control.
	Clear (see page 21172)	Deletes all text from the edit box and all items from the Items (see page 21187) list.
	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
	ClearSelection (see page 21172)	Deselects any selected items in the drop-down list.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
	CopySelection (see page 21173)	Copies the selected item in the drop-down list to another list control.
	Create (see page 13559)	Creates and initializes an instance of TFilterComboBox (see page 13547).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DeleteSelected (see page 5941)	Deletes all the selected items.
	DeleteSelected (see page 21174)	Deletes the selected item from the drop-down list.
	Destroy (see page 13560)	Destroys an instance of TFilterComboBox (see page 13547).
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
V	Focused (see page 21175)	Indicates whether the combo box has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.

	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	MoveSelection (see page 5942)	Moves all selected items to another list control.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectAll (see page 5943)	Selects all items or all text in the selected item.
	SelectAll (see page 21180)	Selects the entire block of text in the edit box.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetDesignVisible (see page 6178)	Makes a control visible at design time.
	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Update (see page 5789)	Processes any pending paint messages immediately.
	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AutoCloseUp (see page 21220)	Specifies whether the drop-down closes up automatically when the user selects an item.
	AutoCompleteDelay (see page 21220)	Specifies the delay between a key press and an attempt to autocomplete the field.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 21184)	Provides access to the drawing surface of the combo box.
	CharCase (see page 21221)	Determines the case of the text in the combo box.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DroppedDown (see page 21185)	Indicates whether the drop-down list is currently displayed.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	Items (see page 21187)	Provides access to the list of items (strings) in the list portion of the combo box.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	Mask (see page 13579)	Specifies the mask associated with the string selected in the filter combo box.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.

R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelLength (see page 21190)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 21193)	Specifies the position of the first selected character in the edit portion of the combo box.
	SelText (see page 21222)	Represents the selected text in the edit region.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	Text (see page 13588)	Contains a text string associated with the control.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 13592)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 13594)	Occurs when the user clicks the control.
	OnContextPopup (see page 13596)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 13596)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 13597)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13599)	Occurs when the user drags an object over a control.
	OnDropDown (see page 13601)	Occurs when the user opens the drop-down list.
	OnEndDrag (see page 13603)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 13604)	Occurs when a control receives the input focus.
	OnExit (see page 13606)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 13608)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 13611)	Occurs when a key is pressed.
	OnKeyUp (see page 13612)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 13614)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 13614)	Occurs when the user moves the mouse outside of a control.
	OnStartDrag (see page 13614)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 13563)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 13567)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 13567)	Positions to matching list items as you type.
	AutoDropDown (see page 13567)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 13568)	Specifies which edges of the control are beveled.
	BevelInner (see page 13568)	Specifies the cut of the inner bevel.
	BevelKind (see page 13569)	Specifies the control's bevel style.
	BevelOuter (see page 13569)	Specifies the cut of the outer bevel.
	Color (see page 13570)	Specifies the background color of the control.
	Constraints (see page 13570)	Specifies the size constraints for the control.

	Ctl3D (see page 13571)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 13571)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 13572)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 13573)	Controls whether the control responds to mouse, keyboard, and timer events.
	FileList (see page 13574)	Links the filter combo box to a file list box control.
	Filter (see page 13575)	Contains all the file masks displayed in the filter combo box.
	Font (see page 13576)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 13577)	Determines the behavior of the input method editor (IME).
	ImeName (see page 13578)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentColor (see page 13580)	Determines where a control looks for its color information.
	ParentCtl3D (see page 13580)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 13580)	Determines where a control looks for its font information.
	ParentShowHint (see page 13581)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 13581)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 13586)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 13587)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 13588)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 13590)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

70.1.4.2 TFilterComboBox.Create Constructor

Creates and initializes an instance of TFilterComboBox (see page 13547).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TFilterComboBox(TComponent * AOwner);
```

Description

Call Create to create a TFilterComboBox (see page 13547) objectat runtime. Filter (see page 13575) combo boxes placed on a form at design time are created automatically.

See Also

Mask (see page 13579)

Style (see page 21224)

Create (see page 21214)

70.1.4.3 TFilterComboBox.Destroy Destructor

Destroys an instance of TFilterComboBox (see page 13547).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TFilterComboBox();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the filter combo box is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

70.1.4.4 TFilterComboBox Methods

The methods of the TFilterComboBox class are listed here.

Protected Methods

	Name	Description
	BuildList (see page 13560)	This is BuildList, a member of class TFilterComboBox.
V	Change (see page 13560)	Generates an OnChange event.
V	Click (see page 13561)	Respond to user click.
V	CreateWnd (see page 13561)	Creates the Windows control that represents the combo box.
V	Notification (see page 13562)	Responds to notifications that components are being created or destroyed.

70.1.4.4.1 TFilterComboBox.BuildList Method

Pascal

```
procedure BuildList;
```

C++

```
__fastcall BuildList();
```

Description

This is BuildList, a member of class TFilterComboBox.

70.1.4.4.2 TFilterComboBox.Change Method

Generates an OnChange event.

Pascal

```
procedure Change; override;
```

C++

```
virtual __fastcall Change();
```

Description

Change is called automatically in response to notifications that the text has changed. It responds by generating an OnChange event. Override this method to add additional processing when the text changes, or to block the OnChange event.

See Also

OnChange (see page 21197)

Changed (see page 5742)

70.1.4.4.3 TFilterComboBox.Click Method

Respond to user click.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Click is called automatically when the user left-clicks the control. Component or application code can call Click to simulate a user mouse click. This is often done in menu actions and hotkey handlers.

As implemented in TControl, Click queries whether the OnClick event handler exists and is different from the OnExecute handler for the control's Action. If this is true, the OnClick event handler is called. If this is false, and the Action OnExecute event handler exists, the Action's Execute method is called. Override Click to provide additional behavior.

See Also

Action (see page 5797)

OnExecute (see page 2716)

Execute (see page 2712)

DblClick (see page 5747)

OnClick (see page 5851)

70.1.4.4.4 TFilterComboBox.CreateWnd Method

Creates the Windows control that represents the combo box.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

The CreateHandle method calls CreateWnd to handle the particulars of creating the combo box window. TCustomCombo overrides the inherited method to ensure that the newly-created window and the TCustomCombo properties correctly reflect each other.

The Windows control for the combo box can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. Thus, CreateWnd may be called more than once during the lifetime of the TCustomCombo instance.

See Also

CreateHandle (see page 6135)

70.1.4.4.5 TFilterComboBox.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

70.1.4.5 TFilterComboBox Properties

The properties of the TFilterComboBox class are listed here.

Public Properties

	Name	Description
R	Mask (see page 13579)	Specifies the mask associated with the string selected in the filter combo box.
	Text (see page 13588)	Contains a text string associated with the control.

Published Properties

	Name	Description
	Align (see page 13563)	Determines how the control aligns within its container (parent control).
	Anchors (see page 13567)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 13567)	Positions to matching list items as you type.
	AutoDropDown (see page 13567)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 13568)	Specifies which edges of the control are beveled.
	BevelInner (see page 13568)	Specifies the cut of the inner bevel.
	BevelKind (see page 13569)	Specifies the control's bevel style.
	BevelOuter (see page 13569)	Specifies the cut of the outer bevel.

	Color (see page 13570)	Specifies the background color of the control.
	Constraints (see page 13570)	Specifies the size constraints for the control.
	Ctl3D (see page 13571)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 13571)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 13572)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 13573)	Controls whether the control responds to mouse, keyboard, and timer events.
	FileList (see page 13574)	Links the filter combo box to a file list box control.
	Filter (see page 13575)	Contains all the file masks displayed in the filter combo box.
	Font (see page 13576)	Controls the attributes of text written on or in the control.
	ImeMode (see page 13577)	Determines the behavior of the input method editor (IME).
	ImeName (see page 13578)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	ParentColor (see page 13580)	Determines where a control looks for its color information.
	ParentCtl3D (see page 13580)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 13580)	Determines where a control looks for its font information.
	ParentShowHint (see page 13581)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 13581)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 13586)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 13587)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 13588)	Determines if the user can tab to a control.
	Visible (see page 13590)	Determines whether the component appears onscreen.

70.1.4.5.1 TFilterComboBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
```

```
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}
```

```
var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

70.1.4.5.2 TFilterComboBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

70.1.4.5.3 TFilterComboBox.AutoComplete Property

Positions to matching list items as you type.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

AutoComplete will position to matching list values as you type, saving you keystrokes.

As you type, the ItemIndex property is updated with the list entry that matches the characters you have entered. As you type, different list entries may match, and ItemIndex changes accordingly.

There is a delay of AutoCompleteDelay milliseconds between the characters typed and the autocomplete operation.

See Also

AutoCompleteDelay (see page 21220)

70.1.4.5.4 TFilterComboBox.AutoDropDown Property

Specifies whether the drop-down list drops down automatically in response to user keystrokes.

Pascal

```
property AutoDropDown: Boolean;
```

C++

```
__property Boolean AutoDropDown;
```

Description

When AutoDropDown is true, the combo box automatically drops down its list when the user starts typing a string while the combo box has focus.

When AutoDropDown is false, the user must explicitly use the drop-down button to drop down the combo box list.

See Also

autocomplete (see page 21220)

OnDropDown (see page 21199)

DroppedDown (see page 21185)

70.1.4.5.5 TFilterComboBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

70.1.4.5.6 TFilterComboBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

70.1.4.5.7 TFilterComboBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

70.1.4.5.8 TFilterComboBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

70.1.4.5.9 TFilterComboBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

70.1.4.5.10 TFilterComboBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

70.1.4.5.11 TFilterComboBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

70.1.4.5.12 TFilterComboBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

70.1.4.5.13 TFilterComboBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}
```

```
void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

70.1.4.5.14 TFilterComboBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

70.1.4.5.15 TFilterComboBox.FileList Property

Links the filter combo box to a file list box control.

Pascal

```
property FileList: TFileListBox;
```

C++

```
__property TFileListBox FileList;
```

Description

If the form includes a TFileListBox (see page 13465) object to display a list of files to choose from, set FileList to allow the filter combo box to limit which files are displayed in the list box. Once the two controls are connected, whenever a new filter is selected using a filter combo box, the file list box displays the files that match the selected filter.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FileListBox1->FileEdit = Edit1;
  FilterComboBox1->FileList = FileListBox1;
  DirectoryListBox1->FileList = FileListBox1;
  DirectoryListBox1->DirLabel = Label1;
  DriveComboBox1->DirList = DirectoryListBox1;
  Button1->Default = true;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FileListBox1->ApplyFilePath(Edit1->Text);
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FileListBox1.ApplyFilePath(Edit1.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileEdit := Edit1;
  FilterComboBox1.FileList := FileListBox1;
  DirectoryListBox1.FileList := FileListBox1;
  DirectoryListBox1.DirLabel := Label1;
  DriveComboBox1.DirList := DirectoryListBox1;
  Button1.Default := True;
end;
```

See Also

Mask (see page 13579)

70.1.4.5.16 TFilterComboBox.Filter Property

Contains all the file masks displayed in the filter combo box.

Pascal

```
property Filter: string;
```

C++

```
__property AnsiString Filter;
```

Description

Set the Filter property to specify the names that appear in the filter combo box, along with the masks associated with each name. Each mask is a file name that may include wildcards. The asterisk (*) is a wildcard which matches any number of arbitrary characters. The question mark (?) is a wildcard which matches a single arbitrary character. When the application runs, the file filter the user selects in the filter combo box becomes the value of the Mask (see page 13579) property.

To create the value of Filter for a single file type, follow these steps:

1Type some meaningful text to indicate the type of file.

2Type a | character (vertical bar).

3Type the file filter.

Don't put in any spaces around the | character in the string.

Here's an example:

```
FilterComboBox1.Filter := 'Text files|*.TXT';
FilterComboBox1->Filter = "Text files|*.TXT";
```

If Filter is set to this string, the string "Text (see page 13588) files" appears in the filter combo box. When the user selects "Text (see page 13588) files", the mask is set to "*.TXT".

To specify multiple file filters, extend the value of Filter, separating each file name/mask combination with a | character. Filter

can specify an any number of file filters, as long as the string is less than 255 characters.

Here's an example of two file filters that could be specified as the value of the Filter property:

```
'Text files (*.TXT)|*.TXT|Pascal files (*.PAS)|*.PAS'
"Text files (*.TXT)|*.TXT|C++ files (*.CPP)|*.CPP"
```

The previous example includes the file filters in parentheses in the text parts. This isn't required, but it's a common convention that helps users understand what to expect when they select a file filter.

The mask portion can include multiple file specifiers if they are separated with semicolons:

```
'All files|*.TXT;*.PAS;*.WB1'
"All files|*.TXT;*.CPP;*.RC"
```

C++ Examples:

```
/*
This example uses a filter combo box and a file list box on
a form. When the user selects a filter in the filter combo
box, the file list box displays only files which match the
selected mask:
*/
void __fastcall TForm1::FilterComboBox1Change(TObject *Sender)
{
  FileListBox1->Mask = FilterComboBox1->Mask;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FilterComboBox1->Filter = "All files (*.*)|*.*| Executable files (*.exe)|*.exe";
}
```

Delphi Examples:

```
{
This example uses a filter combo box and a file list box on
a form. When the user selects a filter in the filter combo
box, the file list box displays only files which match the
selected mask:
}
procedure TForm1.FilterComboBox1Change(Sender: TObject);
begin
  FileListBox1.Mask := FilterComboBox1.Mask;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FilterComboBox1.Filter := 'All files (*.*)|*.*| Pascal files (*.pas)|*.pas';
end;
```

See Also

Mask (see page 13579)

70.1.4.5.17 TFilterComboBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

70.1.4.5.18 TFilterComboBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

70.1.4.5.19 TFilterComboBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

70.1.4.5.20 TFilterComboBox.Mask Property

Specifies the mask associated with the string selected in the filter combo box.

Pascal

```
property Mask: string;
```

C++

```
__property AnsiString Mask;
```

Description

Read Mask to obtain the mask that was selected by the user.

When the user uses the filter combo box to select a new filter, the mask associated with the selected file type becomes the value of the Mask property. If a file list box is linked to the filter combo box with the FileList (see page 13574) property, the file list will update when the value of Mask changes.

C++ Examples:

```
/*
This example uses a filter combo box and a file list box on
a form. When the user selects a filter in the filter combo
box, the file list box displays only files which match the
selected mask:
*/
void __fastcall TForm1::FilterComboBox1Change(TObject *Sender)
{
  FileListBox1->Mask = FilterComboBox1->Mask;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FilterComboBox1->Filter = "All files (*.*)|*.*| Executable files (*.exe)|*.exe";
}
```

Delphi Examples:

```
{
This example uses a filter combo box and a file list box on
a form. When the user selects a filter in the filter combo
box, the file list box displays only files which match the
selected mask:
}
procedure TForm1.FilterComboBox1Change(Sender: TObject);
begin
  FileListBox1.Mask := FilterComboBox1.Mask;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FilterComboBox1.Filter := 'All files (*.*)|*.*| Pascal files (*.pas)|*.pas';
end;
```

See Also

FileList (see page 13574)

Filter (see page 13575)

Mask (see page 13506)

70.1.4.5.21 TFilterComboBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

70.1.4.5.22 TFilterComboBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

70.1.4.5.23 TFilterComboBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

70.1.4.5.24 TFilterComboBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

70.1.4.5.25 TFilterComboBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
```

```
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
```

```
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
```

```
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
```

```
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

70.1.4.5.26 TFilterComboBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

70.1.4.5.27 TFilterComboBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

70.1.4.5.28 TFilterComboBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

70.1.4.5.29 TFilterComboBox.Text Property

Contains a text string associated with the control.

Pascal

```
property Text: TCaption;
```

C++

```
__property TCaption Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
  Size++;                        //Add room for null character
  char *Buffer = new char[Size];  //Creates Buffer dynamic variable
  Edit1->GetTextBuf(Buffer,Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer;
  delete [] Buffer;
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
  Inc(Size);                      {Add room for null character}
  GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
  Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
```

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
end;
```

See Also

Caption (see page 5806)

Lines (see page 21380)

70.1.4.5.30 TFilterComboBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
```

```
  This examples requires a TPageControl and a TCheckBox.  The
  form OnCreate event handler populates each sheet with an
  edit control placed somewhere on the client area of the
  sheet.  Unchecking the checkbox will cause the client area
  (not the tab area) of the Tab Control to become invisible.
  }
  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    PageControl1.ActivePage.Visible := CheckBox1.Checked;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    CheckBox1.Checked := PageControl1.ActivePage.Visible;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

70.1.4.6 TFilterComboBox Events

The events of the TFilterComboBox class are listed here.

Published Events

	Name	Description
	OnChange (see page 13592)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 13594)	Occurs when the user clicks the control.
	OnContextPopup (see page 13596)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 13596)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 13597)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13599)	Occurs when the user drags an object over a control.
	OnDropDown (see page 13601)	Occurs when the user opens the drop-down list.
	OnEndDrag (see page 13603)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

	OnEnter (see page 13604)	Occurs when a control receives the input focus.
	OnExit (see page 13606)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 13608)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 13611)	Occurs when a key is pressed.
	OnKeyUp (see page 13612)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 13614)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 13614)	Occurs when the user moves the mouse outside of a control.
	OnStartDrag (see page 13614)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

70.1.4.6.1 TFilterComboBox.OnChange Event

Occurs when the user changes the text displayed in the edit region.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action immediately after the user edits the text in the edit region or selects an item from the list. The Text property gives the new value in the edit region.

Note: OnChange only occurs in response to user actions. Changing the Text property programmatically does not trigger an OnChange event.

C++ Examples:

```
/*
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
*/
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  if (pCB)
    PageControl1->ActivePage = (TTabSheet *)(pCB->Items->Objects[pCB->ItemIndex]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
    ComboBox1->Items->AddObject(PageControl1->Pages[i]->Name, PageControl1->Pages[i]);
```

```
    ComboBox1->ItemIndex = 0;
  }
}
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to PageControl1.PageCount - 1 do
    ComboBox1.Items.AddObject(PageControl1.Pages[i].Name,
      PageControl1.Pages[i]);
  ComboBox1.ItemIndex := 0;
end;

procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  if (Sender is TComboBox) then
    with (Sender as TComboBox) do
      PageControl1.ActivePage := TTabSheet(Items.Objects[ItemIndex]);
end;
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
```

```
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

See Also

Text (see page 5842)

OnCloseUp (see page 21199)

70.1.4.6.2 TFilterComboBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
```

```
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

70.1.4.6.3 TFilterComboBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

70.1.4.6.4 TFilterComboBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

70.1.4.6.5 TFilterComboBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
```

```
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

70.1.4.6.6 TFilterComboBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
```

```
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

70.1.4.6.7 TFilterComboBox.OnDropDown Event

Occurs when the user opens the drop-down list.

Pascal

```
property OnDropDown: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDropDown;
```

Description

Write an OnDropDown event handler to implement special processing that needs to occur only when the drop-down list is activated.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if (((TForm1 *)Screen->CustomForms[i])->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
```

```
    end;
    Items.EndUpdate; {reenable painting }
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.Show;
end;
```

See Also

DroppedDown (see page 21185)

OnCloseUp (see page 21199)

70.1.4.6.8 TFilterComboBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
```

```
  message displayed depends on whether the dragged label
  control was successfully dropped into and accepted by a
  another control.
  }
  procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
  var
    S: string;
  begin
    S := (Sender as TLabel).Name + ' was dropped... and ';
    if Target <> nil then
      S := S + 'accepted!'
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

70.1.4.6.9 TFilterComboBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
```

```
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

70.1.4.6.10 TFilterComboBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
```

```
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

70.1.4.6.11 TFilterComboBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
```

```
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
  {
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
```

```
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

70.1.4.6.12 TFilterComboBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
```

```
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

70.1.4.6.13 TFilterComboBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
```

```
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

70.1.4.6.14 TFilterComboBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

70.1.4.6.15 TFilterComboBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

70.1.4.6.16 TFilterComboBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these

controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

70.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DirectoryExists (see page 13616)	Determines whether a specified directory exists.
	ForceDirectories (see page 13617)	Creates a new directory, including the creation of parent directories as needed.
	MinimizeName (see page 13617)	Shortens a fully qualified path name so that it can be drawn with a specified length limit.
	ProcessPath (see page 13618)	Parses a file name into its constituent parts.
	SelectDirectory (see page 13619)	Brings up a dialog to allow the user to enter a directory name.
	SelectDirectory (see page 13619)	Brings up a dialog to allow the user to enter a directory name.

70.2.1 FileCtrl.DirectoryExists Function

Determines whether a specified directory exists.

Pascal

```
function DirectoryExists(const Name: string): Boolean;
```

C++

```
Boolean DirectoryExists(const AnsiString Name);
```

File

FileCtrl

Description

Call DirectoryExists to determine whether the directory specified by the Name parameter exists. If the directory exists, the function returns true. If the directory does not exist, the function returns false.

If a full path name is entered, DirectoryExists searches for the directory along the designated path. Otherwise, the Name parameter is interpreted as a relative path name from the current directory.

Note: The FileCtrl (see page 13311) unit (Windows only) also contains a DirectoryExists function. However, the FileCtrl (see page 13311) version is deprecated, and the SysUtils (see page 22709) version preferred, even if the code does not need to be cross-platform.

See Also

ForceDirectories (see page 22895)

SelectDirectory

FileExists (see page 22860)

70.2.2 FileCtrl.ForceDirectories Function

Creates a new directory, including the creation of parent directories as needed.

Pascal

```
function ForceDirectories(const Dir: string): Boolean;
```

C++

```
Boolean ForceDirectories(const AnsiString Dir);
```

File

FileCtrl

Description

ForceDirectories creates a new directory as specified in Dir, which must be a fully-qualified path name. If the directories given in the path do not yet exist, ForceDirectories attempts to create them.

ForceDirectories returns true if it successfully creates all necessary directories, false if it could not create a needed directory.

Warning: Important:Do not call ForceDirectories with an empty string. Doing so causes ForceDirectories to raise an exception.

Note: The FileCtrl (see page 13311) unit (Windows only) also contains a ForceDirectories function. However, the FileCtrl (see page 13311) version is deprecated, and the SysUtils (see page 22709) version preferred, even if the code does not need to be cross-platform.

See Also

ExtractFileDir (see page 22843)

GetCurrentDir (see page 22910)

RemoveDir (see page 22933)

SetCurrentDir (see page 22940)

CreateDir (see page 22817)

DirectoryExists (see page 22832)

SelectDirectory

70.2.3 FileCtrl.MinimizeName Function

Shortens a fully qualified path name so that it can be drawn with a specified length limit.

Pascal

```
function MinimizeName(const Filename: TFileName; Canvas: TCanvas; MaxLen: Integer): TFileName;
```

C++

```
TFileName MinimizeName(const TFileName Filename, TCanvas Canvas, int MaxLen);
```

File

FileCtrl

Description

Use MinimizeName to obtain a version of a file name and path that can be drawn on a limited size drawing surface. MinimizeName replaces directories in the path portion of the file name with dots until the resulting name fits the specified number of pixels in length.

MinimizeName returns the shortened name.

Filename is the fully qualified path name that (possibly) needs to be shortened.

Canvas is the drawing surface on which the resulting name is to be rendered. It is used to determine font metrics.

MaxLen is the length, in pixels, available for drawing the file name on the canvas.

See Also

AdjustLineBreaks

WrapText

70.2.4 FileCtrl.ProcessPath Function

Parses a file name into its constituent parts.

Pascal

```
procedure ProcessPath(const EditText: string; var Drive: Char; var DirPart: string; var
FilePart: string);
```

C++

```
ProcessPath(const AnsiString EditText, Char Drive, AnsiString DirPart, AnsiString FilePart);
```

File

FileCtrl

Description

Call ProcessPath to parse a file name into its drive, path, and file name.

Pass the full file name that should be parsed as the EditText parameter. EditText must refer to an existing file or directory name.

ProcessPath returns the indicated drive, directory, and file name as the Drive, DirPart, and FilePart parameters.

See Also

ExtractFileDir

ExtractFileDrive

ExtractFileExt

ExtractFileName

ExtractFilePath

70.2.5 FileCtrl.SelectDirectory Function

Brings up a dialog to allow the user to enter a directory name.

Pascal

```
function SelectDirectory(var Directory: string; Options: TSelectDirOpts; HelpCtx: Longint):
Boolean; overload;
function SelectDirectory(const Caption: string; const Root: WideString; var Directory: string;
Options: TSelectDirExtOpts = [sdNewUI]; Parent: TWinControl = nil): Boolean; overload;
```

C++

```
Boolean SelectDirectory(AnsiString Directory, TSelectDirOpts Options, Longint HelpCtx);
Boolean SelectDirectory(const AnsiString Caption, const BSTR Root, AnsiString Directory,
TSelectDirExtOpts Options = [sdNewUI], TWinControl * Parent = nil);
```

File

FileCtrl

Description

Call SelectDirectory to let the user enter a directory name.

Use the first syntax to call the Select Directory dialog box. The directory passed to the function with the Directory parameter appears as the currently selected directory when the dialog box appears. The name of the directory the user selects becomes the value of Directory when the function returns.

The HelpCtx parameter is the help context ID number.

Use the second syntax to display the Windows directory browser. The Caption parameter specifies a caption for the dialog. The Root parameter specifies the root directory from which to browse. The selected directory is returned as the Directory parameter. When using this syntax, SelectDirectory does not change the value of the current directory.

Warning: You can't use the same variable for the Root parameter and the Directory parameter.

The Options parameter is a set of values. If Options is the empty set, the user can only select a directory that already exists. No edit box is provided for the user to enter a new directory name. If Options is not empty, the included values determine how the dialog responds when the user types a nonexistent directory name.

With either syntax, SelectDirectory returns true if the user selected a directory and chose OK, and false if the user chose Cancel or closed the dialog box without selecting a directory.

See Also

DirectoryExists (see page 13616)

ForceDirectories (see page 13617)

70.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TDriveType (see page 13620)	This is record FileCtrl.TDriveType.
	TFileAttr (see page 13620)	TFileAttr and TFileType (see page 13624) determine which files a file list box displays.
	TSelectDirExtOpt (see page 13621)	This is record FileCtrl.TSelectDirExtOpt.
	TSelectDirOpt (see page 13622)	TSelectDirOpt and TSelectDirOpts (see page 13625) determine how a directory selection dialog responds when the user enters a nonexistent directory.
	TTextCase (see page 13623)	TTextCase indicates the case of the volume name or drive letter that appears in a dialog box.

70.3.1 FileCtrl.TDriveType Enumeration

Pascal

```
TDriveType = (
  dtUnknown,
  dtNoDrive,
  dtFloppy,
  dtFixed,
  dtNetwork,
  dtCDROM,
  dtRAM
);
```

C++

```
enum TDriveType {
  dtUnknown,
  dtNoDrive,
  dtFloppy,
  dtFixed,
  dtNetwork,
  dtCDROM,
  dtRAM
};
```

File

FileCtrl

Description

This is record FileCtrl.TDriveType.

70.3.2 FileCtrl.TFileAttr Enumeration

TFileAttr and TFileType (see page 13624) determine which files a file list box displays.

Pascal

```
TFileAttr = (
  ftReadOnly,
  ftHidden,
  ftSystem,
  ftVolumeID,
  ftDirectory,
  ftArchive,
  ftNormal
);
```

C++

```
enum TFileAttr {
  ftReadOnly,
  ftHidden,
  ftSystem,
  ftVolumeID,
  ftDirectory,
  ftArchive,
  ftNormal
};
```

File

FileCtrl

Description

TFileType (see page 13624) is a set of TFileAttr values. These are the possible TFileAttr values:

Value	Meaning
ftReadOnly	The list box displays files with the read-only attribute.
ftHidden	The list box displays files with the hidden attribute.
ftSystem	The list box displays system files.
ftVolumeID	The list box displays the volume name.
ftDirectory	The list box displays directories.
ftArchive	The list box displays files with archive attribute.
ftNormal	The list box displays files with no special attributes.

See Also

SelectDirectory

70.3.3 FileCtrl.TSelectDirExtOpt Enumeration

Pascal

```
TSelectDirExtOpt = (
  sdNewFolder,
  sdShowEdit,
  sdShowShares,
  sdNewUI,
  sdShowFiles,
  sdValidateDir
);
```

C++

```
enum TSelectDirExtOpt {
  sdNewFolder,
  sdShowEdit,
  sdShowShares,
  sdNewUI,
  sdShowFiles,
  sdValidateDir
};
```

File

FileCtrl

Description

This is record FileCtrl.TSelectDirExtOpt.

70.3.4 FileCtrl.TSelectDirOpt Enumeration

TSelectDirOpt and TSelectDirOpts (see page 13625) determine how a directory selection dialog responds when the user enters a nonexistent directory.

Pascal

```
TSelectDirOpt = (
  sdAllowCreate,
  sdPerformCreate,
  sdPrompt
);
```

C++

```
enum TSelectDirOpt {
  sdAllowCreate,
  sdPerformCreate,
  sdPrompt
};
```

File

FileCtrl

Description

These are the possible values that can be added to the set of options:

Value	Meaning
sdAllowCreate	An edit box allows the user to type in the name of a directory that does not exist. This option does not create a directory: the application must read the name of the selected directory and create it if desired.
sdPerformCreate	Used only in combination with sdAllowCreate. If the user enters a directory name that does not exist, the directory selection dialog creates it.
sdPrompt	Used only in combination with sdAllowCreate. Displays a message box that informs the user when the entered directory does not exist and asks if the directory should be created. If the user chooses OK, the directory is created if the option set includes sdPerformCreate. If the option set does not include sdPerformCreate, the directory is not created: the application must read the directory name and create it.

See Also

SelectDirectory

70.3.5 FileCtrl.TTextCase Enumeration

TTextCase indicates the case of the volume name or drive letter that appears in a dialog box.

Pascal

```
TTextCase = (
  tcLowerCase,
  tcUpperCase
);
```

C++

```
enum TTextCase {
  tcLowerCase,
  tcUpperCase
};
```

File

FileCtrl

Description

TTextCase can have one of the following values:

Value	Meaning
tcLowerCase	The volume name is displayed in lowercase letters.
tcUpperCase	The volume name is displayed in uppercase letters.

70.4 Types

The following table lists types in this documentation.

Types

Name	Description
TFileType (see page 13624)	TFileType describes possible combinations of file attributes.
TSelectDirExtOpts (see page 13624)	This is type FileCtrl.TSelectDirExtOpts.
TSelectDirOpts (see page 13625)	This is type FileCtrl.TSelectDirOpts.

70.4.1 FileCtrl.TFileType Type

TFileType describes possible combinations of file attributes.

Pascal

```
TFileType = set of TFileAttr;
```

C++

```
TFileAttr TFileType;
```

File

FileCtrl

Description

TFileType describes possible combinations of file attributes.

See Also

TFileListView

70.4.2 FileCtrl.TSelectDirExtOpts Type

Pascal

```
TSelectDirExtOpts = set of TSelectDirExtOpt;
```

C++

```
TSelectDirExtOpt TSelectDirExtOpts;
```

File

FileCtrl

Description

This is type FileCtrl.TSelectDirExtOpts.

70.4.3 FileCtrl.TSelectDirOpts Type

Pascal

```
TSelectDirOpts = set of TSelectDirOpt;
```

C++

```
TSelectDirOpt TSelectDirOpts;
```

File

FileCtrl

Description

This is type FileCtrl.TSelectDirOpts.

70.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
WNTYPE_DRIVE (see page 13626)	This is constant FileCtrl.WNTYPE_DRIVE.

70.5.1 FileCtrl.WNTYPE_DRIVE Constant

Pascal

```
WNTYPE_DRIVE = 1;
```

C++

```
WNTYPE_DRIVE = 1;
```

File

FileCtrl

Description

This is constant FileCtrl.WNTYPE_DRIVE.

71 FMTBcd Namespace

This is namespace FMTBcd.

Topics

Name	Description
Classes (see page 13630)	The following table lists classes in this documentation.
Functions (see page 13632)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 13649)	The following table lists structs, records, enums in this documentation.
Types (see page 13650)	The following table lists types in this documentation.
Constants (see page 13651)	The following table lists constants in this documentation.

Classes

	Name	Description
	EBcdException (see page 13630)	EBcdException is the exception class for errors that occur when working with BCD values.
	EBcdOverflowException (see page 13630)	EBcdOverflowException is the exception class for errors that occur when a Variant can't create a BCD value due to the requested precision and number of decimals.

Constants

Name	Description
_DefaultDecimals (see page 13651)	This is constant FMTBcd._DefaultDecimals.
_NoDecimal (see page 13651)	This is constant FMTBcd._NoDecimal.
DefaultFMTBcdScale (see page 13652)	This is constant FMTBcd.DefaultFMTBcdScale.
MaxBcdPrecision (see page 13652)	This is constant FMTBcd.MaxBcdPrecision.
MaxBcdScale (see page 13652)	This is constant FMTBcd.MaxBcdScale.
MaxFMTBcdDigits (see page 13653)	This is constant FMTBcd.MaxFMTBcdDigits.
MaxFMTBcdFractionSize (see page 13653)	This is constant FMTBcd.MaxFMTBcdFractionSize.
MaxStringDigits (see page 13653)	This is constant FMTBcd.MaxStringDigits.
NullBcd (see page 13654)	Represents a NULL TBcd (see page 13649) value.
SizeOfFraction (see page 13654)	This is constant FMTBcd.SizeOfFraction.
SizeOfTBCD (see page 13654)	This is constant FMTBcd.SizeOfTBCD.

Functions

	Name	Description
	BcdAdd (see page 13633)	Adds two TBcd (see page 13649) values.
	BcdCompare (see page 13633)	Compares two TBcd (see page 13649) values.

	BcdDivide (see page 13634)	Divides one BCD value by another.
	BcdDivide (see page 13634)	Divides one BCD value by another.
	BcdDivide (see page 13634)	Divides one BCD value by another.
	BcdDivide (see page 13634)	Divides one BCD value by another.
	BcdFromBytes (see page 13635)	This is function FMTBcd.BcdFromBytes.
	BcdMultiply (see page 13635)	Multiplies two BCD values.
	BcdMultiply (see page 13635)	Multiplies two BCD values.
	BcdMultiply (see page 13635)	Multiplies two BCD values.
	BcdMultiply (see page 13635)	Multiplies two BCD values.
	BcdPrecision (see page 13636)	Returns the precision of a binary-coded decimal (BCD) value.
	BcdScale (see page 13636)	Returns the number of decimal places for a binary-coded decimal (BCD) value.
	BcdSubtract (see page 13637)	Subtracts one BCD value from another.
	BcdToBytes (see page 13637)	This is function FMTBcd.BcdToBytes.
	BCDToCurr (see page 13637)	Converts a binary-coded decimal value (BCD) to the corresponding Currency value.
	BcdToDouble (see page 13638)	Converts a binary-coded decimal value (BCD) to the corresponding double precision floating point value.
	BcdToInteger (see page 13638)	Converts a binary-coded decimal value (BCD) to an integer.
	BcdToStr (see page 13639)	Returns a string representation of a binary-coded decimal (BCD) value.
	BcdToStrF (see page 13639)	Returns a string representation of a binary-coded decimal (BCD) value that reflects specified formatting.
	CurrToBCD (see page 13640)	Converts a Currency value to the corresponding binary-coded decimal (BCD) value.
	DoubleToBcd (see page 13641)	Converts a double precision floating point value to the corresponding binary-coded decimal (BCD) value.
	DoubleToBcd (see page 13641)	Converts a double precision floating point value to the corresponding binary-coded decimal (BCD) value.
	FormatBcd (see page 13641)	Returns a string representation of a binary-coded decimal (BCD) value that reflects specified formatting.
	IntegerToBcd (see page 13642)	Converts an integer value to the corresponding binary-coded decimal (BCD) value.
	IsBcdNegative (see page 13642)	Indicates whether a binary-coded decimal (BCD) value is less than 0.
	NormalizeBcd (see page 13643)	Converts a BCD value into another BCD value with a specified precision and number of decimals.
	OldBCDToCurr (see page 13643)	This is function FMTBcd.OldBCDToCurr.
	OldCurrToBCD (see page 13644)	This is function FMTBcd.OldCurrToBCD.
	RoundAt (see page 13644)	This is function FMTBcd.RoundAt.
	StrToBcd (see page 13644)	Converts a string to the corresponding binary-coded decimal (BCD) value.
	TryStrToBcd (see page 13645)	Converts a string to the corresponding binary-coded decimal (BCD) value.
	VarFMTBcd (see page 13646)	Returns the TVarType constant that identifies the internal type of a Variant as a TBcd (see page 13649) value.
	VarFMTBcdCreate (see page 13646)	Creates a new Variant that stores its value internally as a TBcd (see page 13649) value.
	VarFMTBcdCreate (see page 13646)	Creates a new Variant that stores its value internally as a TBcd (see page 13649) value.
	VarFMTBcdCreate (see page 13646)	Creates a new Variant that stores its value internally as a TBcd (see page 13649) value.
	VarFMTBcdCreate (see page 13646)	Creates a new Variant that stores its value internally as a TBcd (see page 13649) value.
	VarFMTBcdCreate (see page 13646)	Creates a new Variant that stores its value internally as a TBcd (see page 13649) value.
	VarIsFMTBcd (see page 13647)	Indicates whether a specified Variant represents a TBcd (see page 13649) value.
	VarToBcd (see page 13647)	Converts a Variant into a TBcd (see page 13649) type.

Structs, Records, Enums

	Name	Description
	TBcd (see page 13649)	TBcd stores a binary coded decimal value.

Types

Name	Description
PBcd (see page 13650)	This is type FMTBcd.PBcd.
TNibbleArray (see page 13650)	This is type FMTBcd.TNibbleArray.

71.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EBcdException (see page 13630)	EBcdException is the exception class for errors that occur when working with BCD values.
	EBcdOverflowException (see page 13630)	EBcdOverflowException is the exception class for errors that occur when a Variant can't create a BCD value due to the requested precision and number of decimals.

71.1.1 EBcdException Class

EBcdException is the exception class for errors that occur when working with BCD values.

Class Hierarchy

Pascal

```
EBcdException = class(Exception);
```

C++

```
class EBcdException : public Exception;
```

File

FMTBcd

Description

EBcdException is raised when

An attempt to convert a Variant or string to a BCD value is unsuccessful.

An application tries to create a Variant that represents a BCD value, starting with a value that can't be successfully converted.

An application tries to divide by a BCD type whose value is 0.

See Also

VarFMTBcdCreate (see page 13646)

TBcd (see page 13649)

StrToBCD

BCDDivide

71.1.2 EBcdOverflowException Class

EBcdOverflowException is the exception class for errors that occur when a Variant can't create a BCD value due to the

requested precision and number of decimals.

Class Hierarchy

Pascal

```
EBcdOverflowException = class(EBcdException);
```

C++

```
class EBcdOverflowException : public EBcdException;
```

File

FMTBcd

Description

EBcdOverflowException is raised when

A call to VarFMTBcdCreate (see page 13646) tries to create a BCD value with a precision and number of decimals that can't be supported.

An application tries to cast a Variant to a BCD value but it can't be expressed using the default precision and number of decimals.

See Also

VarFMTBcdCreate (see page 13646)

TBcd (see page 13649)

71.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	BcdAdd (see page 13633)	Adds two TBcd (see page 13649) values.
	BcdCompare (see page 13633)	Compares two TBcd (see page 13649) values.
	BcdDivide (see page 13634)	Divides one BCD value by another.
	BcdDivide (see page 13634)	Divides one BCD value by another.
	BcdDivide (see page 13634)	Divides one BCD value by another.
	BcdDivide (see page 13634)	Divides one BCD value by another.
	BcdFromBytes (see page 13635)	This is function FMTBcd.BcdFromBytes.
	BcdMultiply (see page 13635)	Multiplies two BCD values.
	BcdMultiply (see page 13635)	Multiplies two BCD values.
	BcdMultiply (see page 13635)	Multiplies two BCD values.
	BcdMultiply (see page 13635)	Multiplies two BCD values.
	BcdPrecision (see page 13636)	Returns the precision of a binary-coded decimal (BCD) value.
	BcdScale (see page 13636)	Returns the number of decimal places for a binary-coded decimal (BCD) value.
	BcdSubtract (see page 13637)	Subtracts one BCD value from another.
	BcdToBytes (see page 13637)	This is function FMTBcd.BcdToBytes.
	BCDToCurr (see page 13637)	Converts a binary-coded decimal value (BCD) to the corresponding Currency value.
	BcdToDouble (see page 13638)	Converts a binary-coded decimal value (BCD) to the corresponding double precision floating point value.
	BcdToInteger (see page 13638)	Converts a binary-coded decimal value (BCD) to an integer.
	BcdToStr (see page 13639)	Returns a string representation of a binary-coded decimal (BCD) value.
	BcdToStrF (see page 13639)	Returns a string representation of a binary-coded decimal (BCD) value that reflects specified formatting.
	CurrToBCD (see page 13640)	Converts a Currency value to the corresponding binary-coded decimal (BCD) value.
	DoubleToBcd (see page 13641)	Converts a double precision floating point value to the corresponding binary-coded decimal (BCD) value.
	DoubleToBcd (see page 13641)	Converts a double precision floating point value to the corresponding binary-coded decimal (BCD) value.
	FormatBcd (see page 13641)	Returns a string representation of a binary-coded decimal (BCD) value that reflects specified formatting.
	IntegerToBcd (see page 13642)	Converts an integer value to the corresponding binary-coded decimal (BCD) value.
	IsBcdNegative (see page 13642)	Indicates whether a binary-coded decimal (BCD) value is less than 0.
	NormalizeBcd (see page 13643)	Converts a BCD value into another BCD value with a specified precision and number of decimals.
	OldBCDToCurr (see page 13643)	This is function FMTBcd.OldBCDToCurr.
	OldCurrToBCD (see page 13644)	This is function FMTBcd.OldCurrToBCD.
	RoundAt (see page 13644)	This is function FMTBcd.RoundAt.
	StrToBcd (see page 13644)	Converts a string to the corresponding binary-coded decimal (BCD) value.
	TryStrToBcd (see page 13645)	Converts a string to the corresponding binary-coded decimal (BCD) value.
	VarFMTBcd (see page 13646)	Returns the TVarType constant that identifies the internal type of a Variant as a TBcd (see page 13649) value.
	VarFMTBcdCreate (see page 13646)	Creates a new Variant that stores its value internally as a TBcd (see page 13649) value.
	VarFMTBcdCreate (see page 13646)	Creates a new Variant that stores its value internally as a TBcd (see page 13649) value.

	VarFMTBcdCreate (see page 13646)	Creates a new Variant that stores its value internally as a TBcd (see page 13649) value.
	VarFMTBcdCreate (see page 13646)	Creates a new Variant that stores its value internally as a TBcd (see page 13649) value.
	VarFMTBcdCreate (see page 13646)	Creates a new Variant that stores its value internally as a TBcd (see page 13649) value.
	VarIsFMTBcd (see page 13647)	Indicates whether a specified Variant represents a TBcd (see page 13649) value.
	VarToBcd (see page 13647)	Converts a Variant into a TBcd (see page 13649) type.

71.2.1 FMTBcd.BcdAdd Function

Adds two TBcd (see page 13649) values.

Pascal

```
procedure BcdAdd(const bcdIn1: TBcd; const bcdIn2: TBcd; var bcdOut: TBcd); deprecated;
```

C++

```
BcdAdd(const TBcd bcdIn1, const TBcd bcdIn2, TBcd bcdOut);
```

File

FMTBcd

Description

BcdAdd adds bcdIn1 to bcdIn2 and returns the result as bcdOut.

The return value is given a precision and number of decimal places that is sufficient to accurately represent the result of the addition. You can convert this to a specific precision and number of decimal places using the NormalizeBcd (see page 13643) procedure.

See Also

BcdSubtract

BcdMultiply

BcdDivide

NormalizeBcd

71.2.2 FMTBcd.BcdCompare Function

Compares two TBcd (see page 13649) values.

Pascal

```
function BcdCompare(const bcd1: TBcd; const bcd2: TBcd): Integer;
```

C++

```
int BcdCompare(const TBcd bcd1, const TBcd bcd2);
```

File

FMTBcd

Description

Call BcdCompare to compare two TBcd (see page 13649) values. The return value is indicated in the following table:

Return value	Condition
-1	bcd1< bcd2
0	bcd1= bcd2
1	bcd1 > bcd2

See Also

CompareValue

71.2.3 FMTBcd.BcdDivide Function

Divides one BCD value by another.

Pascal

```
procedure BcdDivide(Dividend: string; Divisor: string; var bcdOut: TBcd); overload; deprecated;
procedure BcdDivide(const Dividend: TBcd; const Divisor: TBcd; var bcdOut: TBcd); overload;
deprecated;
procedure BcdDivide(const Dividend: TBcd; const Divisor: Double; var bcdOut: TBcd); overload;
deprecated;
procedure BcdDivide(const Dividend: TBcd; const Divisor: string; var bcdOut: TBcd); overload;
deprecated;
```

C++

```
BcdDivide(AnsiString Dividend, AnsiString Divisor, TBcd bcdOut);
BcdDivide(const TBcd Dividend, const TBcd Divisor, TBcd bcdOut);
BcdDivide(const TBcd Dividend, const Double Divisor, TBcd bcdOut);
BcdDivide(const TBcd Dividend, const AnsiString Divisor, TBcd bcdOut);
```

File

FMTBcd

Description

BcdDivide divides Dividend by Divisor and returns the result as bcdOut.

The return value is given a precision and number of decimal places that is sufficient to accurately represent the result of the division, if possible. (The number of decimal places in the result, however, will never exceed 10.) You can convert this to a specific precision and number of decimal places using the NormalizeBcd (see page 13643) procedure.

See Also

BcdSubtract

BcdMultiply

BcdAdd

NormalizeBcd

71.2.4 FMTBcd.BcdFromBytes Function

Pascal

```
function BcdFromBytes(const AValue: TBytes): TBcd;
```

C++

```
TBcd BcdFromBytes(const TBytes AValue);
```

File

FMTBcd

Description

This is function FMTBcd.BcdFromBytes.

71.2.5 FMTBcd.BcdMultiply Function

Multiplies two BCD values.

Pascal

```
procedure BcdMultiply(const bcdIn1: TBcd; const bcdIn2: TBcd; var bcdOut: TBcd); overload;
deprecated;
procedure BcdMultiply(const bcdIn: TBcd; const DoubleIn: Double; var bcdOut: TBcd); overload;
deprecated;
procedure BcdMultiply(const bcdIn: TBcd; const StringIn: string; var bcdOut: TBcd); overload;
deprecated;
procedure BcdMultiply(StringIn1: string; StringIn2: string; var bcdOut: TBcd); overload;
deprecated;
```

C++

```
BcdMultiply(const TBcd bcdIn1, const TBcd bcdIn2, TBcd bcdOut);
BcdMultiply(const TBcd bcdIn, const Double DoubleIn, TBcd bcdOut);
BcdMultiply(const TBcd bcdIn, const AnsiString StringIn, TBcd bcdOut);
BcdMultiply(AnsiString StringIn1, AnsiString StringIn2, TBcd bcdOut);
```

File

FMTBcd

Description

BcdMultiply multiplies the first two arguments and returns the result as bcdOut.

The return value is given a precision and number of decimal places that is sufficient to accurately represent the result of the multiplication, if possible. You can convert this to a specific precision and number of decimal places using the NormalizeBcd (see page 13643) procedure.

See Also

BcdSubtract

BcdDivide

BcdAdd

NormalizeBcd

71.2.6 FMTBcd.BcdPrecision Function

Returns the precision of a binary-coded decimal (BCD) value.

Pascal

```
function BcdPrecision(const Bcd: TBcd): Word;
```

C++

```
Word BcdPrecision(const TBcd Bcd);
```

File

FMTBcd

Description

Use BcdPrecision to determine the number of significant digits in a specified BCD value.

See Also

BcdScale

IsBcdNegative

NormalizeBcd

71.2.7 FMTBcd.BcdScale Function

Returns the number of decimal places for a binary-coded decimal (BCD) value.

Pascal

```
function BcdScale(const Bcd: TBcd): Word;
```

C++

```
Word BcdScale(const TBcd Bcd);
```

File

FMTBcd

Description

Use BcdScale to determine the number of decimal places in a specified BCD value.

See Also

BcdPrecision

IsBcdNegative

NormalizeBcd

71.2.8 FMTBcd.BcdSubtract Function

Subtracts one BCD value from another.

Pascal

```
procedure BcdSubtract(const bcdIn1: TBcd; const bcdIn2: TBcd; var bcdOut: TBcd); deprecated;
```

C++

```
BcdSubtract(const TBcd bcdIn1, const TBcd bcdIn2, TBcd bcdOut);
```

File

FMTBcd

Description

BcdSubtract subtracts bcdIn2 from bcdIn1 and returns the result as bcdOut.

The return value is given a precision and number of decimal places that is sufficient to accurately represent the result of the subtraction. You can convert this to a specific precision and number of decimal places using the NormalizeBcd (see page 13643) procedure.

See Also

BcdMultiply

BcdDivide

BcdAdd

NormalizeBcd

71.2.9 FMTBcd.BcdToBytes Function

Pascal

```
function BcdToBytes(const Value: TBcd): TBytes;
```

C++

```
TBytes BcdToBytes(const TBcd Value);
```

File

FMTBcd

Description

This is function FMTBcd.BcdToBytes.

71.2.10 FMTBcd.BCDToCurr Function

Converts a binary-coded decimal value (BCD) to the corresponding Currency value.

Pascal

```
function BCDToCurr(const BCD: TBcd; var Curr: Currency): Boolean;
```

C++

```
Boolean BCDToCurr(const TBcd BCD, Currency Curr);
```

File

FMTBcd

Description

Use BCDToCurr to convert a binary-coded decimal value, such as is stored in a physical BCD database field, into a Currency value, which is the representation used by TBCDField objects. BCDToCurr returns true if the BCD value passed as the BCD parameter is successfully converted into a Currency object which is passed out as the Curr parameter.

See Also

CurrToBCD

TBCDField

TField

71.2.11 FMTBcd.BcdToDouble Function

Converts a binary-coded decimal value (BCD) to the corresponding double precision floating point value.

Pascal

```
function BcdToDouble(const Bcd: TBcd): Double;
```

C++

```
Double BcdToDouble(const TBcd Bcd);
```

File

FMTBcd

Description

Use BcdToDouble to convert a binary-coded decimal value into a double precision floating point value. If the Bcd parameter represents a value that can't be expressed in double precision, BcdToDouble raises an exception.

See Also

DoubleToBcd

BCDToCurr

BCDToInteger

71.2.12 FMTBcd.BcdToInteger Function

Converts a binary-coded decimal value (BCD) to an integer.

Pascal

```
function BcdToInteger(const Bcd: TBcd; Truncate: Boolean = False): Integer;
```

C++

```
int BcdToInteger(const TBcd Bcd, Boolean Truncate = False);
```

File

FMTBcd

Description

Use BcdToInteger to convert a binary-coded decimal value into an integer.

Bcd is the value to convert.

Truncate indicates how to handle values that include a decimal portion. If Truncate is true, BcdToInteger truncates the value specified by Bcd and converts only the integer portion. If Truncate is false, BcdToInteger raises an exception if the Bcd parameter does not represent an integer value.

See Also

IntegerToBcd

BCDToCurr

BCDToDouble

71.2.13 FMTBcd.BcdToStr Function

Returns a string representation of a binary-coded decimal (BCD) value.

Pascal

```
function BcdToStr(const Bcd: TBcd): string; overload;
```

C++

```
AnsiString BcdToStr(const TBcd Bcd);
```

File

FMTBcd

Description

Use BcdToStr to convert a binary-coded decimal value into a string. BcdToStr formats the string to display all of the digits stored in the BCD value.

See Also

BcdToStrF

StrToBcd

TryStrToBcd

71.2.14 FMTBcd.BcdToStrF Function

Returns a string representation of a binary-coded decimal (BCD) value that reflects specified formatting.

Pascal

```
function BcdToStrF(const Bcd: TBcd; Format: TFloatFormat; const Precision: Integer; const
Digits: Integer): string;
```

C++

```
AnsiString BcdToStrF(const TBcd Bcd, TFloatFormat Format, const int Precision, const int
Digits);
```

File

FMTBcd

Description

Use BcdToStrF to convert a binary-coded decimal value into a string.

Value is the BCD value to convert to a string.

Format is the formatting code that indicates the overall type of formatting to use.

Precision indicates the maximum number of significant digits in the resulting string.

Digits indicates the maximum number of digits past the decimal point in the resulting string.

See Also

BcdToStr

StrToBcd

FloatToStrF

FormatBcd

TryStrToBcd

71.2.15 FMTBcd.CurrToBCD Function

Converts a Currency value to the corresponding binary-coded decimal (BCD) value.

Pascal

```
function CurrToBCD(const Curr: Currency; var BCD: TBcd; Precision: Integer = 32; Decimals:
Integer = 4): Boolean;
```

C++

```
Boolean CurrToBCD(const Currency Curr, TBcd BCD, int Precision = 32, int Decimals = 4);
```

File

FMTBcd

Description

Use CurrToBCD to convert a Currency value, such as is used by TBCDField objects, into a binary-coded decimal value, such as is stored in a physical BCD database field. CurrToBCD returns true if the Currency object passed as the Curr parameter is successfully converted into a BCD value and passed out as the BCD parameter. The Precision parameter specifies the number of digits represented in the BCD value, and the Decimals parameter specifies the number of those digits that are after the decimal point.

See Also

BCDToCurr

TBCDField

TField

71.2.16 FMTBcd.DoubleToBcd Function

Converts a double precision floating point value to the corresponding binary-coded decimal (BCD) value.

Pascal

```
function DoubleToBcd(const AValue: Double): TBcd; overload;
procedure DoubleToBcd(const AValue: Double; var bcd: TBcd); overload;
```

C++

```
TBcd DoubleToBcd(const Double AValue);
DoubleToBcd(const Double AValue, TBcd bcd);
```

File

FMTBcd

Description

Use DoubleToBcd to convert a double precision floating point value into a binary-coded decimal value. DoubleToBcd infers the number of decimal places and significant digits from the specified value aValue, using only as many as necessary. It never sets the number of significant digits to a value greater than 15.

aValue is the value to convert.

bcd returns the resulting BCD value when using the second syntax. When using the first syntax, the BCD value is the return value of the function.

See Also

BCDToDouble

CurrToBcd

IntegerToBcd

71.2.17 FMTBcd.FormatBcd Function

Returns a string representation of a binary-coded decimal (BCD) value that reflects specified formatting.

Pascal

```
function FormatBcd(const Format: string; Bcd: TBcd): string;
```

C++

```
AnsiString FormatBcd(const AnsiString Format, TBcd Bcd);
```

File

FMTBcd

Description

Use FormatBcd to convert a binary-coded decimal value into a string.

Format indicates how the resulting string should be formatted. It uses the same format specifiers as pertain to the FormatFloat function.

Bcd is the value to convert to a string.

See Also

BcdToStrF

FormatFloat

BcdToStr

TryStrToBcd

StrToBcd

71.2.18 FMTBcd.IntegerToBcd Function

Converts an integer value to the corresponding binary-coded decimal (BCD) value.

Pascal

```
function IntegerToBcd(const AValue: Integer): TBcd;
```

C++

```
TBcd IntegerToBcd(const int AValue);
```

File

FMTBcd

Description

Use IntegerToBcd to convert an integer into a binary-coded decimal value.

See Also

BCDToInteger

CurrToBcd

DoubleToBcd

71.2.19 FMTBcd.IsBcdNegative Function

Indicates whether a binary-coded decimal (BCD) value is less than 0.

Pascal

```
function IsBcdNegative(const Bcd: TBcd): Boolean;
```

C++

```
Boolean IsBcdNegative(const TBcd Bcd);
```

File

FMTBcd

Description

IsBcdNegative returns true if Bcd is less than 0, false if Bcd is greater than or equal to 0.

See Also

BcdPrecision

BcdScale

NormalizeBcd

71.2.20 FMTBcd.NormalizeBcd Function

Converts a BCD value into another BCD value with a specified precision and number of decimals.

Pascal

```
function NormalizeBcd(const InBcd: TBcd; var OutBcd: TBcd; const Prec: Word; const Scale:
Word): Boolean;
```

C++

```
Boolean NormalizeBcd(const TBcd InBcd, TBcd OutBcd, const Word Prec, const Word Scale);
```

File

FMTBcd

Description

Call NormalizeBcd to give a TBcd (see page 13649) value a specific precision and number of decimal places. If the specified precision or number of decimal places is less than that of the original BCD value, NormalizeBcd truncates the value (as opposed to rounding).

InBcd is the BCD value to convert.

OutBcd returns the resulting BCD value with the specified precision and decimal places.

Prec is the number of significant digits in OutBcd.

Scale is the number of decimal places in OutBcd.

NormalizeBcd returns true if the conversion can be made with no loss of precision. It returns false if the conversion required some digits to be truncated.

71.2.21 FMTBcd.OldBCDToCurr Function

Pascal

```
function OldBCDToCurr(const BCD: TBcd; var Curr: Currency): Boolean;
```

C++

```
Boolean OldBCDToCurr(const TBcd BCD, Currency Curr);
```

File

FMTBcd

Description

This is function FMTBcd.OldBCDToCurr.

71.2.22 FMTBcd.OldCurrToBCD Function

Pascal

```
function OldCurrToBCD(const Curr: Currency; var BCD: TBcd; Precision: Integer = 32; Decimals:
Integer = 4): Boolean;
```

C++

```
Boolean OldCurrToBCD(const Currency Curr, TBcd BCD, int Precision = 32, int Decimals = 4);
```

File

FMTBcd

Description

This is function FMTBcd.OldCurrToBCD.

71.2.23 FMTBcd.RoundAt Function

Pascal

```
function RoundAt(const Value: string; Position: SmallInt): string;
```

C++

```
AnsiString RoundAt(const AnsiString Value, SmallInt Position);
```

File

FMTBcd

Description

This is function FMTBcd.RoundAt.

71.2.24 FMTBcd.StrToBcd Function

Converts a string to the corresponding binary-coded decimal (BCD) value.

Pascal

```
function StrToBcd(const AValue: string): TBcd;
```

C++

```
TBcd StrToBcd(const AnsiString AValue);
```

File

FMTBcd

Description

Use StrToBcd to convert a string into a binary-coded decimal value. The decimals and precision are inferred from the digits present in the string.

StrToBcd converts the string specified by aValue and returns the resulting BCD value. If aValue does not represent a value that can be expressed as a TBcd (see page 13649), StrToBcd raises an exception.

Tip: To convert a string to a Bcd value without raising an exception for invalid strings, use the TryStrToBcd (see page 13645) function instead.

See Also

BcdToStr

BcdToStrF

FormatBcd

TryStrToBcd

71.2.25 FMTBcd.TryStrToBcd Function

Converts a string to the corresponding binary-coded decimal (BCD) value.

Pascal

```
function TryStrToBcd(const AValue: string; var Bcd: TBcd): Boolean;
```

C++

```
Boolean TryStrToBcd(const AnsiString AValue, TBcd Bcd);
```

File

FMTBcd

Description

Use TryStrToBcd to convert a string into a binary-coded decimal value. The decimals and precision are inferred from the digits present in the string.

TryStrToBcd converts the string specified by aValue and returns the resulting BCD value as Bcd.

TryStrToBcd returns true if it successfully converts aValue to a TBcd (see page 13649) value. If aValue can't be successfully converted to a TBcd (see page 13649) value, TryStrToBcd returns false.

See Also

BcdToStr

BcdToStrF

FormatBcd

StrToBcd

71.2.26 FMTBcd.VarFMTBcd Function

Returns the TVarType constant that identifies the internal type of a Variant as a TBcd (see page 13649) value.

Pascal

```
function VarFMTBcd: TVarType;
```

C++

```
TVarType VarFMTBcd();
```

File

FMTBcd

Description

VarFMTBcd returns the Variant type code that identifies a TBcd (see page 13649)-valued Variant. This is the value that the VarType function returns when the Variant stores its value as a TBcd (see page 13649) value.

The Variant type codes for custom Variants (see page 23567) are dynamically allocated when the application loads the unit that implements them. VarFMTBcd allows the application to identify the Variant type code that is currently used for custom Variants (see page 23567) that represent a BCD value. This type code, however, may not match the type code used by another application that uses a different set of custom Variant types.

See Also

VarIsFMTBcd

VarType

TBcd

VarToBcd

71.2.27 FMTBcd.VarFMTBcdCreate Function

Creates a new Variant that stores its value internally as a TBcd (see page 13649) value.

Pascal

```
procedure VarFMTBcdCreate(var ADest: Variant; const ABcd: TBcd); overload;
function VarFMTBcdCreate: Variant; overload;
function VarFMTBcdCreate(const AValue: string; Precision: Word; Scale: Word): Variant;
overload;
function VarFMTBcdCreate(const AValue: Double; Precision: Word = 18; Scale: Word = 4):
Variant; overload;
function VarFMTBcdCreate(const ABcd: TBcd): Variant; overload;
```

C++

```
VarFMTBcdCreate(Variant ADest, const TBcd ABcd);
Variant VarFMTBcdCreate();
Variant VarFMTBcdCreate(const AnsiString AValue, Word Precision, Word Scale);
Variant VarFMTBcdCreate(const Double AValue, Word Precision = 18, Word Scale = 4);
Variant VarFMTBcdCreate(const TBcd ABcd);
```

File

FMTBcd

Description

Call VarBcdCreate to obtain a Variant that stores its value as a TBcd (see page 13649) value.

Using the first syntax, the Variant returned as aDest has the same value as ABcd.

Using the second syntax (passing no arguments), the returned Variant has a value of NullBcd (see page 13654).

Using the third syntax, the returned Variant has the same value as ABcd.

Using the fourth syntax, the returned Variant converts AValue (if necessary) to extract the value.

Using the remaining syntaxes, the AValue parameter is converted into a TBcd (see page 13649) value with the specified Precision and Scale and the result is the value of the Variant.

See Also

VarIsFMTBcd

VarToBcd

NullBcd

71.2.28 FMTBcd.VarIsFMTBcd Function

Indicates whether a specified Variant represents a TBcd (see page 13649) value.

Pascal

```
function VarIsFMTBcd(const AValue: Variant): Boolean; overload;
```

C++

```
Boolean VarIsFMTBcd(const Variant AValue);
```

File

FMTBcd

Description

VarIsBcd returns true if the Variant aValue stores its value internally as a TBcd (see page 13649) value. VarIsBcd returns false otherwise.

See Also

VarToBcd

TBcd

VarFMTBcd

71.2.29 FMTBcd.VarToBcd Function

Converts a Variant into a TBcd (see page 13649) type.

Pascal

```
function VarToBcd(const AValue: Variant): TBcd;
```

C++

```
TBcd VarToBcd(const Variant AValue);
```

File

FMTBcd

Description

Use VarDoubleAsBcd to obtain a TBcd (see page 13649) value from a Variant.

aValue is the Variant to convert.

See Also

VarIsFMTBcd

71.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TBcd (see page 13649)	TBcd stores a binary coded decimal value.

71.3.1 FMTBcd.TBcd Record

TBcd stores a binary coded decimal value.

Pascal

```
TBcd = packed record
  Precision: Byte;
  SignSpecialPlaces: Byte;
  Fraction: packed array [0..31] of Byte;
end;
```

C++

```
struct TBcd {
  Byte Precision;
  Byte SignSpecialPlaces;
  packed array [0..31] of Byte Fraction;
};
```

File

FMTBcd

Description

TBcd represents a binary coded decimal value. It contains the following fields:

Field	Contents
Precision	The number of digits in the decimal version of the value.
SignSpecialPlaces	The sign bit (0 is positive, any other value negative)
Fraction	An array of BCD nibbles, 00 to 99 per byte, high nibble first. Only the first Precision nibbles are used.

You can create a Variant that represents a TBcd value using the VarFMTBcdCreate (see page 13646) function. In fact, the easiest way to manipulate TBcd values is to create Variants (see page 23567) for them and use the built-in operators provided by the Variant type.

71.4 Types

The following table lists types in this documentation.

Types

Name	Description
PBcd (see page 13650)	This is type FMTBcd.PBcd.
TNibbleArray (see page 13650)	This is type FMTBcd.TNibbleArray.

71.4.1 FMTBcd.PBcd Type

Pascal

```
PBcd = ^TBcd;
```

C++

```
^TBcd PBcd;
```

File

FMTBcd

Description

This is type FMTBcd.PBcd.

71.4.2 FMTBcd.TNibbleArray Type

Pascal

```
TNibbleArray = array[0..63] of Byte;
```

C++

```
array[0..63] of Byte TNibbleArray;
```

File

FMTBcd

Description

This is type FMTBcd.TNibbleArray.

71.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
_DefaultDecimals (see page 13651)	This is constant FMTBcd._DefaultDecimals.
_NoDecimal (see page 13651)	This is constant FMTBcd._NoDecimal.
DefaultFMTBcdScale (see page 13652)	This is constant FMTBcd.DefaultFMTBcdScale.
MaxBcdPrecision (see page 13652)	This is constant FMTBcd.MaxBcdPrecision.
MaxBcdScale (see page 13652)	This is constant FMTBcd.MaxBcdScale.
MaxFMTBcdDigits (see page 13653)	This is constant FMTBcd.MaxFMTBcdDigits.
MaxFMTBcdFractionSize (see page 13653)	This is constant FMTBcd.MaxFMTBcdFractionSize.
MaxStringDigits (see page 13653)	This is constant FMTBcd.MaxStringDigits.
NullBcd (see page 13654)	Represents a NULL TBcd (see page 13649) value.
SizeOfFraction (see page 13654)	This is constant FMTBcd.SizeOfFraction.
SizeOfTBCD (see page 13654)	This is constant FMTBcd.SizeOfTBCD.

71.5.1 FMTBcd._DefaultDecimals Constant

Pascal

```
_DefaultDecimals = 10;
```

C++

```
_DefaultDecimals = 10;
```

File

FMTBcd

Description

This is constant FMTBcd._DefaultDecimals.

71.5.2 FMTBcd._NoDecimal Constant

Pascal

```
_NoDecimal = -255;
```

C++

```
_NoDecimal = -255;
```

File

FMTBcd

Description

This is constant FMTBcd._NoDecimal.

71.5.3 FMTBcd.DefaultFMTBcdScale Constant

Pascal

```
DefaultFMTBcdScale = 6;
```

C++

```
DefaultFMTBcdScale = 6;
```

File

FMTBcd

Description

This is constant FMTBcd.DefaultFMTBcdScale.

71.5.4 FMTBcd.MaxBcdPrecision Constant

Pascal

```
MaxBcdPrecision = 18;
```

C++

```
MaxBcdPrecision = 18;
```

File

FMTBcd

Description

This is constant FMTBcd.MaxBcdPrecision.

71.5.5 FMTBcd.MaxBcdScale Constant

Pascal

```
MaxBcdScale = 4;
```

C++

```
MaxBcdScale = 4;
```

File

FMTBcd

Description

This is constant FMTBcd.MaxBcdScale.

71.5.6 FMTBcd.MaxFMTBcdDigits Constant

Pascal

```
MaxFMTBcdDigits = 32;
```

C++

```
MaxFMTBcdDigits = 32;
```

File

FMTBcd

Description

This is constant FMTBcd.MaxFMTBcdDigits.

71.5.7 FMTBcd.MaxFMTBcdFractionSize Constant

Pascal

```
MaxFMTBcdFractionSize = 64;
```

C++

```
MaxFMTBcdFractionSize = 64;
```

File

FMTBcd

Description

This is constant FMTBcd.MaxFMTBcdFractionSize.

71.5.8 FMTBcd.MaxStringDigits Constant

Pascal

```
MaxStringDigits = 100;
```

C++

```
MaxStringDigits = 100;
```

File

FMTBcd

Description

This is constant FMTBcd.MaxStringDigits.

71.5.9 FMTBcd.NullBcd Constant

Represents a NULL TBcd (see page 13649) value.

Pascal

```
NullBcd: TBcd = (Precision: 0; SignSpecialPlaces: 0; Fraction:
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0));
```

C++

```
TBcd NullBcd = (Precision: 0; SignSpecialPlaces: 0; Fraction:
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0));
```

File

FMTBcd

Description

Use NullBcd to indicate an unknown or missing TBcd (see page 13649) value. NullBcd is a TBcd (see page 13649) with all its fields set to 0.

See Also

Null

71.5.10 FMTBcd.SizeOfFraction Constant

Pascal

```
SizeOfFraction = 32;
```

C++

```
SizeOfFraction = 32;
```

File

FMTBcd

Description

This is constant FMTBcd.SizeOfFraction.

71.5.11 FMTBcd.SizeOfTBCD Constant

Pascal

```
SizeOfTBCD = 2 + SizeOfFraction;
```

C++

```
SizeOfTBCD = 2 + SizeOfFraction;
```

File

FMTBcd

Description

This is constant FMTBcd.SizeOfTBCD.

72 Forms Namespace

This is namespace Forms.

Topics

Name	Description
Classes (see page 13660)	The following table lists classes in this documentation.
Interfaces (see page 14334)	The following table lists interfaces in this documentation.
Functions (see page 14342)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 14353)	The following table lists structs, records, enums in this documentation.
Types (see page 14369)	The following table lists types in this documentation.
Variables (see page 14383)	The following table lists variables in this documentation.

Classes

	Name	Description
	TApplication (see page 13660)	TApplication is the type used for a GUI windowed application.
	TControlScrollBar (see page 13744)	TControlScrollBar displays a horizontal or vertical scroll bar.
	TCustomActiveForm (see page 13759)	TCustomActiveForm is the base class for Active (see page 13896) Forms.
	TCustomDockForm (see page 13829)	TCustomDockForm is the default floating dock site class for TControl objects.
	TCustomForm (see page 13849)	TCustomForm is the base class from which to derive a window such as a form or dialog.
	TCustomFrame (see page 13978)	TCustomFrame is the base class from which TFrame (see page 14114) descends.
	TForm (see page 13992)	TForm represents a standard application window (form).
	TFrame (see page 14114)	TFrame is a container for components; it can be nested within forms or other frames.
	TGlassFrame (see page 14186)	TGlassFrame controls Windows Vista Aero glass effects.
	TMonitor (see page 14191)	TMonitor represents a monitor on which an application runs.
	TScreen (see page 14197)	TScreen represents the state of the screen in which an application runs.
	TScrollBox (see page 14238)	TScrollBox represents a scrolling area (a scroll box) in a window.
	TScrollingWinControl (see page 14312)	TScrollingWinControl is the base class for controls that support scrolling.

Functions

	Name	Description
	AllocateHWnd (see page 14342)	Creates a window that implements a specified window procedure.
	DeallocateHWnd (see page 14343)	Frees a window that was created using AllocateHWnd (see page 14342).
	DisableTaskWindows (see page 14343)	This is function Forms.DisableTaskWindows.
	DoneCtl3D (see page 14344)	This is function Forms.DoneCtl3D.

	EnableTaskWindows (see page 14344)	This is function Forms.EnableTaskWindows.
	ForegroundTask (see page 14344)	Indicates whether the current thread is running in the foreground.
	FreeObjectInstance (see page 14345)	This is function Forms.FreeObjectInstance.
	GetApplicationMainFormOnTaskBar (see page 14345)	This is function Forms.GetApplicationMainFormOnTaskBar.
	GetCustomFormGlassFrame (see page 14345)	This is function Forms.GetCustomFormGlassFrame.
	GetParentForm (see page 14346)	Returns the form or property page that contains a specified control.
	InitCtl3D (see page 14346)	This is function Forms.InitCtl3D.
	IsAccel (see page 14346)	Indicates whether a particular character is an accelerator character (or hot key) within a given menu caption or other text string.
	KeyboardStateToShiftState (see page 14347)	Convert array of virtual keys to TShiftState.
	KeyboardStateToShiftState (see page 14347)	Convert array of virtual keys to TShiftState.
	KeyDataToShiftState (see page 14348)	Convert keyboard message data to TShiftState.
	KeysToShiftState (see page 14348)	Convert mouse event parameter to TShiftState.
	MakeObjectInstance (see page 14349)	This is function Forms.MakeObjectInstance.
	RestoreFocusState (see page 14349)	This is function Forms.RestoreFocusState.
	SaveFocusState (see page 14349)	This is function Forms.SaveFocusState.
	SetApplicationMainFormOnTaskBar (see page 14350)	This is function Forms.SetApplicationMainFormOnTaskBar.
	SetAutoSubClass (see page 14350)	This is function Forms.SetAutoSubClass.
	SetCustomFormGlassFrame (see page 14350)	This is function Forms.SetCustomFormGlassFrame.
	Subclass3DDlg (see page 14351)	This is function Forms.Subclass3DDlg.
	Subclass3DWnd (see page 14351)	This is function Forms.Subclass3DWnd.
	ValidParentForm (see page 14351)	Returns the form or property page that contains a specified control.

Interfaces

	Name	Description
	IDesignerHook (see page 14334)	IDesignerHook is an interface that allows component writers to interact with the form designer in the IDE.
	IOleForm (see page 14340)	IOleForm is designed to communicate with OLE containers. To embed a VCL form in an OLE container, CodeGear recommends using ActiveForms and TOleControl instead of IOleForm.

Structs, Records, Enums

	Name	Description
	TActiveFormBorderStyle (see page 14353)	Specifies the style of an active form's border.
	TBorderIcon (see page 14354)	TBorderIcons (see page 14370) and TBorderIcons (see page 14370) specify the buttons that appear on the title bar of a form window.
	TCloseAction (see page 14355)	TCloseAction describes how a form should respond when it is closed.
	TCMHintShow (see page 14355)	This is record Forms.TCMHintShow.
	TCMHintShowPause (see page 14356)	This is record Forms.TCMHintShowPause.
	TCursorRec (see page 14356)	This is record Forms.TCursorRec.
	TDefaultMonitor (see page 14357)	Specifies the monitor on which the form appears.
	TFormBorderStyle (see page 14357)	TFormBorderStyle is the type of the BorderStyle property for windows-only forms and TBorderStyle (see page 14371) is the type of the BorderStyle property for other controls in windows-only applications.
	TFormStyle (see page 14358)	TFormStyle describes form style.
	THintInfo (see page 14359)	THintInfo is used to define the appearance and behavior of the Help window.
	TMonitorDefaultTo (see page 14361)	TMonitorDefautlTo specifies which monitor to select when there is no match to the specified criteria.
	TPopupForm (see page 14362)	This is record Forms.TPopupForm.
	TPopupMode (see page 14362)	TPopupMode enumerates the possible values of a Form's PopupMode property.
	TPopupWnd (see page 14363)	This is record Forms.TPopupWnd.
	TPosition (see page 14363)	TPosition describes the positioning of a form.
	TPrintScale (see page 14364)	Represents the proportions of a printed form.
	TScrollBarKind (see page 14365)	TScrollBarKind indicates the orientation of a scrollbar.
	TScrollBarStyle (see page 14366)	TScrollBarStyle specifies a scroll bar style.
	TShowAction (see page 14366)	This is record Forms.TShowAction.
	TTileMode (see page 14367)	Represents how MDI child forms are arranged when the Tile method is called.

	TTimerMode (see page 14367)	This is record Forms.TTimerMode.
	TWindowState (see page 14368)	TWindowState indicates whether a form is maximized, minimized, or normally sized.

Types

Name	Description
PCursorRec (see page 14369)	This is type Forms.PCursorRec.
PHintInfo (see page 14370)	PHintInfo declares a pointer to a Help Window appearance record.
TBorderIcons (see page 14370)	TBorderIcons specifies the buttons that appear on the title bar of a form window.
TBorderStyle (see page 14371)	TBorderStyle is the type of the BorderStyle property for framed controls.
TCloseEvent (see page 14371)	TCloseEvent is the type of a form's OnClose event handle.
TCloseQueryEvent (see page 14372)	TCloseQueryEvent is used for event handlers that are called when a window is about to close.
TCustomFormClass (see page 14372)	TCustomFormClass defines the metaclass for TCustomForm (see page 13849).
TCustomFrameClass (see page 14373)	TCustomFrameClass defines the metaclass for TCustomFrame (see page 13978).
TExceptionEvent (see page 14373)	TExceptionEvent is the type of event handlers that respond when the application traps an unhandled exception.
TFocusState (see page 14374)	This is type Forms.TFocusState.
TFormClass (see page 14374)	TFormClass defines the metaclass for TForm (see page 13992).
TFormState (see page 14374)	TFormState describes the possible state of a form.
TGetHandleEvent (see page 14375)	This is type Forms.TGetHandleEvent.
THelpEvent (see page 14375)	THelpEvent is the type of an OnHelp event handler.
TIdleEvent (see page 14376)	TIdleEvent is the type of event handler that perform actions when the application is idle.
TMessageEvent (see page 14376)	TMessageEvent is the type of an OnMessage event handler.
TPopupFormArray (see page 14377)	This is type Forms.TPopupFormArray.
TPopupWndArray (see page 14377)	This is type Forms.TPopupWndArray.
TScrollBarInc (see page 14377)	TScrollBarInc represents a number of logical units on a TControlScrollBar (see page 13744).
TSetLayeredWindowAttributes (see page 14378)	This is type Forms.TSetLayeredWindowAttributes.
TSettingChangeEvent (see page 14378)	TSettingChangeEvent is the type of an OnSettingChange event handler.
TShortCutEvent (see page 14381)	TShortCutEvent is the type of an OnShortCut event handler.
TShowHintEvent (see page 14381)	TShowHintEvent is the type of the OnShowHint event handler.
TWindowHook (see page 14382)	TWindowHook is the type for a dialog procedure.

Variables

Name	Description
Application (see page 14383)	Represents application-level information.
Ctl3DBtnWndProc (see page 14383)	This is variable Forms.Ctl3DBtnWndProc.
Ctl3DCtlColorEx (see page 14384)	This is variable Forms.Ctl3DCtlColorEx.
Ctl3DDlgFramePaint (see page 14384)	This is variable Forms.Ctl3DDlgFramePaint.
HintWindowClass (see page 14384)	Specifies the class for the window used to display help hints.
Screen (see page 14385)	Represents a screen device.
SetLayeredWindowAttributes (see page 14385)	This is variable Forms.SetLayeredWindowAttributes.

72.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TApplication (see page 13660)	TApplication is the type used for a GUI windowed application.
	TControlScrollBar (see page 13744)	TControlScrollBar displays a horizontal or vertical scroll bar.
	TCustomActiveForm (see page 13759)	TCustomActiveForm is the base class for Active (see page 13896) Forms (see page 13657).
	TCustomDockForm (see page 13829)	TCustomDockForm is the default floating dock site class for TControl objects.
	TCustomForm (see page 13849)	TCustomForm is the base class from which to derive a window such as a form or dialog.
	TCustomFrame (see page 13978)	TCustomFrame is the base class from which TFrame (see page 14114) descends.
	TForm (see page 13992)	TForm represents a standard application window (form).
	TFrame (see page 14114)	TFrame is a container for components; it can be nested within forms or other frames.
	TGlassFrame (see page 14186)	TGlassFrame controls Windows Vista Aero glass effects.
	TMonitor (see page 14191)	TMonitor represents a monitor on which an application runs.
	TScreen (see page 14197)	TScreen represents the state of the screen in which an application runs.
	TScrollBox (see page 14238)	TScrollBox represents a scrolling area (a scroll box) in a window.
	TScrollingWinControl (see page 14312)	TScrollingWinControl is the base class for controls that support scrolling.

72.1.1 TApplication Class

TApplication is the type used for a GUI windowed application.

Class Hierarchy

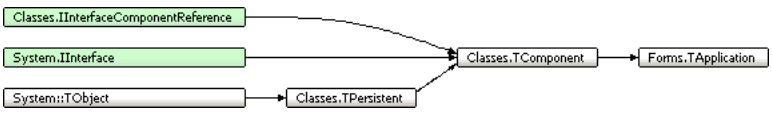

Pascal

```
TApplication = class(TComponent);
```

C++

```
class TApplication : public TComponent;
```

File

Forms

Description

TApplication encapsulates a windowed application. The methods and properties introduced in TApplication reflect the fundamentals established in the Windows operating system to create, run, sustain, and destroy an application. TApplication thereby simplifies the interface between the developer and the Windows environment. For this purpose, TApplication encapsulates behavior for

Windows message processing.

Context-sensitive online help.

Menu accelerator and key processing.

Exception handling.

Managing the fundamental parts defined by the Windows operating system for an application, such as MainWindow, WindowClass, and so on

Each GUI application automatically declares an Application (see page 14383) variable as the instance of the application. If the application is not a Web server application, control panel applet, or NT service application, this variable is of type TApplication.

TApplication does not appear on the Component palette, nor is it available in the form designer to visually manipulate; so it has no published properties. Nevertheless, some of its public properties can be set or modified at design time in the Forms (see page 13657) and Application (see page 14383) pages of the Project|Options dialog box. In addition, you can add a TApplicationEvents component to any form in the project to intercept the application's events, and supply event handlers using the IDE.

Note: For application-wide properties and methods that affect the display, see TScreen (see page 14197).

See Also

Application (see page 14383)

Exception (see page 22750)

TComponent (see page 2762)

TControl (see page 5724)

TForm (see page 13992)

TWinControl (see page 6112)

TScreen (see page 14197)

TApplicationEvents (see page 1326)

Console Applications

Creating Applications

MDI Applications

SDI Applications

User Interface Models

GUI Applications

Controlling Application Behavior

Working at the Application Level

72.1.1.1 TApplication Members

The following tables list the members exposed by TApplication.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Idle (see page 13682)	This is Idle, a member of class TApplication.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDlgMsg (see page 13685)	This is IsDlgMsg, a member of class TApplication.
	IsHintMsg (see page 13685)	This is IsHintMsg, a member of class TApplication.
	IsKeyMsg (see page 13685)	This is IsKeyMsg, a member of class TApplication.
	IsMDIMsg (see page 13685)	This is IsMDIMsg, a member of class TApplication.
	IsPreProcessMessage (see page 13685)	This is IsPreProcessMessage, a member of class TApplication.
	IsShortCut (see page 13686)	This is IsShortCut, a member of class TApplication.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnActionExecute (see page 13725)	Occurs when an action's Execute method is called and its action list has not already handled it.
	OnActionUpdate (see page 13726)	Occurs when an action's Update method is called and its action list has not already handled it.
	OnActivate (see page 13726)	Occurs when an application becomes active.
	OnDeactivate (see page 13727)	Occurs when an application becomes inactive.
	OnException (see page 13728)	Occurs when an unhandled exception occurs in the application.
	OnGetActiveFormHandle (see page 13729)	This is OnGetActiveFormHandle, a member of class TApplication.
	OnGetMainFormHandle (see page 13729)	This is OnGetMainFormHandle, a member of class TApplication.
	OnHelp (see page 13730)	Occurs when the application receives a request for help.
	OnHint (see page 13731)	Occurs when the mouse pointer moves over a control or menu item that can display a Help Hint (see page 13712).
	OnIdle (see page 13733)	Occurs when an application becomes idle.
	OnMessage (see page 13737)	Occurs when the application receives a Windows message.
	OnMinimize (see page 13739)	Occurs when an application is minimized.
	OnModalBegin (see page 13740)	Occurs when a modal form is opened.
	OnModalEnd (see page 13740)	Occurs when a modal form is closed.
	OnRestore (see page 13741)	Occurs when the previously minimized application is restored to its normal size.
	OnSettingChange (see page 13741)	Occurs when Windows notifies the application that a system-wide setting has changed.
	OnShortCut (see page 13741)	Occurs when the user presses a key (before the OnKeyDown event).
	OnShowHint (see page 13742)	Occurs when the application is about to display the hint window for a Help Hint (see page 13712).

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActivateHint (see page 13668)	Displays a hint window for the control at a specified position.
	AddPopupForm (see page 13669)	This is AddPopupForm, a member of class TApplication.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BringToFront (see page 13669)	Sets the last active window as the topmost window on the desktop above all other applications.
	CancelHint (see page 13669)	Cancels the display of a hint for a control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ControlDestroyed (see page 13670)	Notifies the application when a control is destroyed.
V	Create (see page 13666)	Creates a new application.
	CreateForm (see page 13670)	Creates a new form.
	CreateHandle (see page 13672)	Generates a main window for the application if it does not already have one.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 13667)	Destroys the application instance and all of its associated memory.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoApplicationIdle (see page 13673)	Processes messages in the Message queue for certain events.
	ExecuteAction (see page 13673)	Generates an OnActionExecute (see page 13725) event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandleException (see page 13674)	Provides default handling of exceptions for the application.
	HandleMessage (see page 13674)	Interrupts the execution of an application while Windows processes a message in the Windows message queue.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HelpCommand (see page 13678)	Provides quick access to any of the Help commands in the WinHelp API (application programming interface).
	HelpContext (see page 13679)	Displays a specified help topic.
	HelpJump (see page 13679)	Displays a specified help topic.
	HelpKeyword (see page 13680)	Displays a specified help topic.
	HelpShowTableOfContents (see page 13680)	Shows the Table of Contents within the help system.
	HideHint (see page 13681)	Hides the current hint.
	HintMouseMessage (see page 13681)	Regulates the display of hint windows.
	HookMainWindow (see page 13681)	Enables a native Windows dialog box to receive messages sent to the application's main window.
	HookSynchronizeWakeup (see page 13682)	This is HookSynchronizeWakeup, a member of class TApplication.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	Initialize (see page 13682)	Provides an opportunity to initialize subsystems.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 13686)	Specifies whether the application adjusts its interface for Middle Eastern locales.
	MessageBox (see page 13686)	Displays a specified message to the user.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	Minimize (see page 13688)	Shrinks an application to the Windows task bar.
	ModalFinished (see page 13689)	Track closing of modal form.
	ModalStarted (see page 13689)	Track opening of modal form.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NormalizeAllTopMosts (see page 13690)	Makes forms that have been designated as topmost forms (their FormStyle is fsStayOnTop) behave as if they were not topmost forms.
	NormalizeTopMosts (see page 13690)	Makes forms that have been designated as topmost forms (their FormStyle is fsStayOnTop) behave as if they were not topmost forms.
	ProcessMessages (see page 13691)	Interrupts the execution of an application so that it can process the message queue.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemovePopupForm (see page 13694)	This is RemovePopupForm, a member of class TApplication.
	Restore (see page 13694)	Restores a minimized application to its normal size.
	RestoreTopMosts (see page 13695)	Restores forms designated as fsStayOnTop to be topmost again.
	Run (see page 13696)	Executes the application.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	ShowException (see page 13698)	Displays a message box for exceptions that are not caught by application code.
	Terminate (see page 13699)	Ends application execution.
	UnhookMainWindow (see page 13701)	Releases a dialog procedure previously hooked by a call to the HookMainWindow (see page 13681) method.
	UnhookSynchronizeWakeup (see page 13701)	Removes the method that was assigned to the WakeMainThread variable by HookSynchronizeWakeup (see page 13682).
	UpdateAction (see page 13701)	Generates an OnActionUpdate (see page 13726) event.
	UseRightToLeftAlignment (see page 13702)	Specifies whether the application object is in a right-to-left alignment mode.
	UseRightToLeftReading (see page 13702)	Specifies whether the application object is in a right-to-left text mode.
	UseRightToLeftScrollBar (see page 13703)	Specifies whether the vertical scroll bar appears on the left side controls.

Public Properties

	Name	Description
	ActionUpdateDelay (see page 13704)	Delays the call to DoActionIdle.
R	Active (see page 13704)	Specifies whether the application is active and has focus.
R	ActiveFormHandle (see page 13705)	Gives the Windows handle of the currently active form.
	AllowTesting (see page 13705)	Represents information for the IDE.
	AutoDragDocking (see page 13705)	Indicates whether dragged windows are automatically docked.
	BiDiKeyboard (see page 13706)	Specifies the name of the keyboard map that should be used in middle-east locales when text reads from right to left.
	BiDiMode (see page 13706)	Indicates the layout of the application when running under Middle Eastern versions of Windows.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurrentHelpFile (see page 13707)	Indicates the current help file.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DialogHandle (see page 13707)	Provides a mechanism for using non-VCL dialog boxes in an application.
	EnumAllWindowsOnActivateHint (see page 13708)	This is EnumAllWindowsOnActivateHint, a member of class TApplication.
R	ExeName (see page 13709)	Contains the file name of the application's executable file including path information.
	Handle (see page 13710)	Provides access to the window handle of the main form (window) of the application.
	HelpFile (see page 13711)	Specifies the name of the default file the application uses to display Help.
R	HelpSystem (see page 13711)	Provides access to the interface for working with the help system.
	Hint (see page 13712)	Specifies the text string that appears in the Help Hint box.
	HintColor (see page 13715)	Determines the color of the hint boxes for the Help Hints for the application.
	HintHidePause (see page 13715)	Specifies the time interval to wait before hiding the Help Hint (see page 13712) if the mouse has not moved from the control or menu item.

	HintPause (see page 13715)	Specifies the time interval that passes before the control's Help Hint (see page 13712) appears when the user places the mouse pointer on a control or menu item.
	HintShortCuts (see page 13716)	Specifies whether the application's pop-up hints include shortcut information.
	HintShortPause (see page 13716)	Specifies the time period to wait before bringing up a hint if another hint has already been shown.
	Icon (see page 13717)	Determines which icon represents the application on the Windows task bar.
R	MainForm (see page 13718)	Identifies which form in the application is the main window.
R	MainFormHandle (see page 13719)	Gives the Windows handle of the application main form.
	MainFormOnTaskBar (see page 13719)	Controls (see page 5711) how Windows taskbar buttons are handled by VCL.
R	ModalLevel (see page 13720)	This is ModalLevel, a member of class TApplication.
	ModalPopupMode (see page 13720)	This is ModalPopupMode, a member of class TApplication.
	NonBiDiKeyboard (see page 13720)	Specifies the name of the keyboard map that should be used unless text reads from right to left.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	PopupControlWnd (see page 13721)	This is PopupControlWnd, a member of class TApplication.
	ShowHint (see page 13721)	Determines whether Help Hints are enabled or disabled for the entire application.
	ShowMainForm (see page 13722)	Determines whether the application shows its main form on startup.
R	Terminated (see page 13723)	Reports whether the application is in the process of shutting down.
	Title (see page 13723)	Contains the text that appears below the icon representing the application when it is minimized.
	UpdateFormatSettings (see page 13724)	Specifies whether format settings are updated automatically when the user alters the system configuration.
	UpdateMetricSettings (see page 13724)	Specifies whether the settings used for the hint window font and the icon title font are updated.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

72.1.1.2 TApplication.Create Constructor

Creates a new application.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TApplication(TComponent * AOwner);
```

Description

Do not call Create directly. Each GUI application automatically creates an instance of an application object.

For TApplication (see page 13660), the Create method initializes default values for some of its properties. It also prepares the application to run by setting up objects such as hint windows, the application icon, the main window handle, the instance handle, lists for managing child windows of the main window (form), and so on.

See Also

CreateForm (see page 13670)

CreateHandle (see page 13672)

Destroy (see page 13667)

Handle (see page 13710)

MainForm (see page 13718)

Run (see page 13696)

72.1.1.3 TApplication.Destroy Destructor

Destroys the application instance and all of its associated memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TApplication();
```

Description

Never call Application.Destroy or Application.Free. Instead, use the application's Terminate (see page 13699) method to end the application.

Destroy signals that the application has been deactivated before calling the inherited destructor. It then frees memory for hooked windows, the application instance, other object instances and lists that were created at application startup or that accumulated during execution

Since the Application (see page 14383) owns all forms (usually), it destroys these forms in its destructor.

See Also

Terminate (see page 13699)

Terminated (see page 13723)

OnDeactivate (see page 13727)

72.1.1.4 TApplication Methods

The methods of the TApplication class are listed here.

Protected Methods

	Name	Description
	Idle (see page 13682)	This is Idle, a member of class TApplication.
	IsDlgMsg (see page 13685)	This is IsDlgMsg, a member of class TApplication.
	IsHintMsg (see page 13685)	This is IsHintMsg, a member of class TApplication.
	IsKeyMsg (see page 13685)	This is IsKeyMsg, a member of class TApplication.
	IsMDIMsg (see page 13685)	This is IsMDIMsg, a member of class TApplication.
	IsPreProcessMessage (see page 13685)	This is IsPreProcessMessage, a member of class TApplication.
	IsShortCut (see page 13686)	This is IsShortCut, a member of class TApplication.

Public Methods

	Name	Description
	ActivateHint (see page 13668)	Displays a hint window for the control at a specified position.
	AddPopupForm (see page 13669)	This is AddPopupForm, a member of class TApplication.
	BringToFront (see page 13669)	Sets the last active window as the topmost window on the desktop above all other applications.
	CancelHint (see page 13669)	Cancels the display of a hint for a control.
	ControlDestroyed (see page 13670)	Notifies the application when a control is destroyed.
	CreateForm (see page 13670)	Creates a new form.

	CreateHandle (see page 13672)	Generates a main window for the application if it does not already have one.
	DoApplicationIdle (see page 13673)	Processes messages in the Message queue for certain events.
	ExecuteAction (see page 13673)	Generates an OnActionExecute (see page 13725) event.
	HandleException (see page 13674)	Provides default handling of exceptions for the application.
	HandleMessage (see page 13674)	Interrupts the execution of an application while Windows processes a message in the Windows message queue.
	HelpCommand (see page 13678)	Provides quick access to any of the Help commands in the WinHelp API (application programming interface).
	HelpContext (see page 13679)	Displays a specified help topic.
	HelpJump (see page 13679)	Displays a specified help topic.
	HelpKeyword (see page 13680)	Displays a specified help topic.
	HelpShowTableOfContents (see page 13680)	Shows the Table of Contents within the help system.
	HideHint (see page 13681)	Hides the current hint.
	HintMouseMessage (see page 13681)	Regulates the display of hint windows.
	HookMainWindow (see page 13681)	Enables a native Windows dialog box to receive messages sent to the application's main window.
	HookSynchronizeWakeup (see page 13682)	This is HookSynchronizeWakeup, a member of class TApplication.
	Initialize (see page 13682)	Provides an opportunity to initialize subsystems.
	IsRightToLeft (see page 13686)	Specifies whether the application adjusts its interface for Middle Eastern locales.
	MessageBox (see page 13686)	Displays a specified message to the user.
	Minimize (see page 13688)	Shrinks an application to the Windows task bar.
	ModalFinished (see page 13689)	Track closing of modal form.
	ModalStarted (see page 13689)	Track opening of modal form.
	NormalizeAllTopMosts (see page 13690)	Makes forms that have been designated as topmost forms (their FormStyle is fsStayOnTop) behave as if they were not topmost forms.
	NormalizeTopMosts (see page 13690)	Makes forms that have been designated as topmost forms (their FormStyle is fsStayOnTop) behave as if they were not topmost forms.
	ProcessMessages (see page 13691)	Interrupts the execution of an application so that it can process the message queue.
	RemovePopupForm (see page 13694)	This is RemovePopupForm, a member of class TApplication.
	Restore (see page 13694)	Restores a minimized application to its normal size.
	RestoreTopMosts (see page 13695)	Restores forms designated as fsStayOnTop to be topmost again.
	Run (see page 13696)	Executes the application.
	ShowException (see page 13698)	Displays a message box for exceptions that are not caught by application code.
	Terminate (see page 13699)	Ends application execution.
	UnhookMainWindow (see page 13701)	Releases a dialog procedure previously hooked by a call to the HookMainWindow (see page 13681) method.
	UnhookSynchronizeWakeup (see page 13701)	Removes the method that was assigned to the WakeMainThread variable by HookSynchronizeWakeup (see page 13682).
	UpdateAction (see page 13701)	Generates an OnActionUpdate (see page 13726) event.
	UseRightToLeftAlignment (see page 13702)	Specifies whether the application object is in a right-to-left alignment mode.
	UseRightToLeftReading (see page 13702)	Specifies whether the application object is in a right-to-left text mode.
	UseRightToLeftScrollBar (see page 13703)	Specifies whether the vertical scroll bar appears on the left side controls.

72.1.1.4.1 TApplication.ActivateHint Method

Displays a hint window for the control at a specified position.

Pascal

```
procedure ActivateHint(CursorPos: TPoint);
```

C++

```
__fastcall ActivateHint(TPoint CursorPos);
```

Description

ActivateHint locates the control or menu item at the position specified by CursorPos, where CursorPos represents a screen coordinates in pixels. After locating the control, ActivateHint displays the control's Hint (see page 13712) in a hint window.

See Also

THintWindow (see page 6055)

Hint (see page 13712)

OnHint (see page 13731)

Hint (see page 5827)

72.1.1.4.2 TApplication.AddPopupForm Method

Pascal

```
function AddPopupForm(APopupForm: TCustomForm): Integer;
```

C++

```
__fastcall int AddPopupForm(TCustomForm APopupForm);
```

Description

This is AddPopupForm, a member of class TApplication.

72.1.1.4.3 TApplication.BringToFront Method

Sets the last active window as the topmost window on the desktop above all other applications.

Pascal

```
procedure BringToFront;
```

C++

```
__fastcall BringToFront();
```

Description

Use BringToFront to find the window owned by the MainForm (see page 13718) that was last active and bring it to the foreground. BringToFront also tests to see if the window is visible and enabled before making it the topmost window.

See Also

BringToFront (see page 5740)

MainForm (see page 13718)

72.1.1.4.4 TApplication.CancelHint Method

Cancels the display of a hint for a control.

Pascal

```
procedure CancelHint;
```

C++

```
__fastcall CancelHint();
```

Description

Applications rarely, if ever, call CancelHint directly. CancelHint is used internally to cancel the display of hints at appropriate times, such as when a window is not valid or when the application closes. CancelHint calls a series of routines that clean up the display and timing issues of the hint window for a control.

See Also

HideHint (see page 13681)

Hint (see page 13712)

72.1.1.4.5 TApplication.ControlDestroyed Method

Notifies the application when a control is destroyed.

Pascal

```
procedure ControlDestroyed(Control: TControl);
```

C++

```
__fastcall ControlDestroyed(TControl * Control);
```

Description

ControlDestroyed is for internal use and is called automatically by TControl. Do not call it directly.

ControlDestroyed notifies the application that the control passed as the Control parameter has been destroyed, so that the properties which refer to controls, such as MainForm (see page 13718) and ActiveCustomForm, can be updated to specify a valid object or nil (Delphi) or NULL (C++).

See Also

Destroy (see page 13667)

MainForm (see page 13718)

ActiveCustomForm (see page 14207)

72.1.1.4.6 TApplication.CreateForm Method

Creates a new form.

Pascal

```
procedure CreateForm(InstanceClass: TComponentClass; var Reference);
```

C++

```
__fastcall CreateForm(TComponentClass InstanceClass,  Reference);
```

Description

Call CreateForm to dynamically create a form at runtime. Developers do not need to add code for creating most forms, because typically one or more calls to CreateForm are added automatically to the project's source when using the form designer.

CreateForm creates a new form of the type specified by the FormClass parameter and assigns it to the variable given by the Reference parameter. The owner of the new form is the Application (see page 14383) object.

Note: By default, the form created by the first call to CreateForm in a project becomes the application's main form.

C++ Examples:

```
/*
StepBy, CreateForm, Run, Show, Update, Initialize example
This example shows progress of loading forms as an application starts up. The code example is
placed in the project source (*.dpr) file. To see project source, select the View | Project
Source menu item.
You will need to set up your project with the following steps before using the code example:
```

```
o          Add four additional forms to a default project.
o          Place a TProgressBar on Form5
o          Take the Project|Options|Forms menu option and place Form5 on the available forms
list.
o          Change the code of your project (*.dpr) file to look like the example.
*/
#include <vcl\vcl.h>
#include "Unit5.h"
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("TAppCreateForm.cpp", Form1);
USERES("Project1.res");
USEFORM("Unit2.cpp", Form2);
USEFORM("Unit3.cpp", Form3);
USEFORM("Unit4.cpp", Form4);
USEFORM("Unit5.cpp", Form5);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
  try
  {
    Application->Initialize();
    SetApplicationMainFormOnTaskBar(Application, true);
    Application->CreateForm(__classid(TForm5), &Form5);
    Form5->ProgressBar1->Max = 100;
    Form5->Show();   // show a splash screen contain ProgressBar control
    Form5->Update(); // force display of Form5
    Application->CreateForm(__classid(TForm1), &Form1);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form1 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm2), &Form2);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form2 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm3), &Form3);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form3 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm4), &Form4);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form4 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
//  delete Form5;
    Application->Run();
  }
  catch (Exception &exception)
  {
    Application->ShowException(&exception);
  }
  return 0;
}
```

Delphi Examples:

```
{
This example shows progress of loading forms as an
application starts up. The code example is placed in the
project source (*.dpr) file.  To see project source, select
the View | Project Source menu item.
You will need to set up your project with the following steps
before using the code example:
    Add four additional forms to a default project.
```

```
      Place a TProgressBar on Form5
      Take the Project|Options|Forms menu option and place
            Form5 on the available forms list.
      Change the code of your project (*.dpr) file to look
            like the example.
  }
  begin
    Application.Initialize;
    with TForm5.Create(nil) do
    try
      Application.MainFormOnTaskbar := True;
      ProgressBar1.Max := 100;
      Show;  // show a splash screen contain ProgressBar control
      Update; // force display of Form5
      Application.CreateForm(TForm1, Form1);
      ProgressBar1.StepBy(25);
      Label1.Caption :=  'Form1 loaded successfully.';
      Update; // force display of Form5
      Sleep(3000);
      Application.CreateForm(TForm2, Form2);
      ProgressBar1.StepBy(25);
      Label1.Caption :=  'Form2 loaded successfully.';
      Update; // force display of Form5
      Sleep(3000);
      Application.CreateForm(TForm3, Form3);
      ProgressBar1.StepBy(25);
      Label1.Caption :=  'Form3 loaded successfully.';
      Update; // force display of Form5
      Sleep(3000);
      Application.CreateForm(TForm4, Form4);
      ProgressBar1.StepBy(25);
      Label1.Caption :=  'Form4 loaded successfully.';
      Update; // force display of Form5
      Sleep(3000);
    finally
      Free;
    end;
    Application.Run;
```

See Also

CreateHandle (see page 13672)

Handle (see page 13710)

MainForm (see page 13718)

Run (see page 13696)

72.1.1.4.7 TApplication.CreateHandle Method

Generates a main window for the application if it does not already have one.

Pascal

```
procedure CreateHandle;
```

C++

```
__fastcall CreateHandle();
```

Description

Do not call CreateHandle directly. In EXEs, the application's constructor calls CreateHandle. In DLLs it does not.

CreateHandle encapsulates much of what traditional Windows programs write in WinMain. For example, CreateHandle registers the Window Class, creates the application window and determines how the application window is shown.

See Also

Create (see page 13666)

CreateForm (see page 13670)

Handle (see page 13710)

MainForm (see page 13718)

Run (see page 13696)

72.1.1.4.8 TApplication.DoApplicationIdle Method

Processes messages in the Message queue for certain events.

Pascal

```
procedure DoApplicationIdle;
```

C++

```
__fastcall DoApplicationIdle();
```

Description

The DoApplicationIdle method forces a call to the protected method TApplication.Idle (see page 13682). Idle (see page 13682) triggers the event OnIdle (see page 13733), processes the events OnMouseEnter and OnMouseLeave, and updates the hints.

72.1.1.4.9 TApplication.ExecuteAction Method

Generates an OnActionExecute (see page 13725) event.

Pascal

```
function ExecuteAction(Action: TBasicAction): Boolean; reintroduce;
```

C++

```
__fastcall Boolean ExecuteAction(TBasicAction Action);
```

Description

When the user invokes an action, a series of calls ensues to respond to that action. First, the action list that contains the action generates an OnExecute event. If the action list does not handle the OnExecute event, then the action is routed to the Application (see page 14383) object's ExecuteAction method, which invokes the OnActionExecute (see page 13725) event handler. (The application's ExecuteAction method responds to all unhandled actions in the application.) If the OnActionExecute (see page 13725) event handler does not handle the action, then it is routed to the action's OnExecute event handler. If that does not handle the action, the active control's ExecuteAction method is called, followed (if necessary) by the active form's ExecuteAction method.

Do not call ExecuteAction. It is called automatically when the user invokes an action. The Action parameter specifies the action that was invoked. ExecuteAction checks whether the application knows how to perform its function, and if so, executes the action and returns true. Otherwise, ExecuteAction returns false, and the default processing for the action continues.

See Also

OnActionExecute (see page 13725)

UpdateAction (see page 13701)

Execute (see page 2712)

HandlesTarget (see page 2713)

ExecuteAction

72.1.1.4.10 TApplication.HandleException Method

Provides default handling of exceptions for the application.

Pascal

```
procedure HandleException(Sender: TObject);
```

C++

```
__fastcall HandleException(TObject * Sender);
```

Description

If an exception passes through all the try blocks in the application code, the application automatically calls HandleException, which displays a dialog box indicating an error occurred. Unless the exception object is EAbort, HandleException calls the OnException (see page 13728) event handler, if one exists. Otherwise, it calls ShowException (see page 13698) to display a message dialog box indicating an error occurred.

To assign other exception handling code for the application, use the OnException (see page 13728) event handler.

See Also

EAbort (see page 22722)

OnException (see page 13728)

ShowException (see page 13698)

72.1.1.4.11 TApplication.HandleMessage Method

Interrupts the execution of an application while Windows processes a message in the Windows message queue.

Pascal

```
procedure HandleMessage;
```

C++

```
__fastcall HandleMessage();
```

Description

HandleMessage interrupts the execution of the application so that Windows can process a single message from the Windows message queue before returning control to the application. If the message queue is empty, HandleMessage generates an OnIdle (see page 13733) event and starts the process of updating the actions in the application.

Note: If the application goes idle, HandleMessage may take a long time to return. Therefore, do not call HandleMessage when waiting for something message-based while priority actions are also being processed. Instead, call ProcessMessages (see page 13691) when processing more than just messages.

C++ Examples:

```
/*
The following code demonstrates background processing using
the OnIdle event, and using the HandleMessage method to
permit messages or background processing to get through.
Note: You must add MyIdleHandler to the Form1 methods.
*/
```

```
// global variables to show the order of events
int  XPos, YPos, Delta;

// This is a utility procedure to display messages
// add this at the beginning of the unit

void StatusMsg(TForm *Form, TCanvas *Canvas, char *Message, bool Bkg)
{
  if (!Bkg)
    Canvas->Font->Style = TFontStyles() << fsBold; // foreground messages are bold
  Canvas->TextOut(XPos, YPos, Message);
  if (!Bkg)
    Canvas->Font->Style = TFontStyles();
  // change Xpos and YPos to prepare for the next message
  YPos += Delta;
  if (YPos >= Form->ClientHeight - 10)
  {
    YPos = 10;
    XPos += 180;
  }
  if (XPos >= Form->ClientWidth - 100)
  {
    if (Canvas->Font->Color == clRed)
      Canvas->Font->Color = clBlack;
    else
      Canvas->Font->Color = clRed;
    XPos = 10;
  }
}

/*
This is the OnCreate event handler for Form1.  It
initializes global values and sets the application's OnIdle
event handler
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Button1->Caption = "Do not yield";
  Button2->Caption  = "Handle Message";
  Application->OnIdle = MyIdleHandler;
  XPos = 10;
  YPos = 10;
  if (Canvas->Font->Height > 0)
    Delta = Canvas->Font->Height + 1;
  else
    Delta = 1 - Canvas->Font->Height;
}

/*
This is the OnIdle event handler. It is set in the Form's
OnCreate event handler, so you need only add it as a private
method of the form.  Usually it would perform some background
processing for the application.  This one simply writes a
message to let you know when it's there.
*/
void __fastcall TForm1::MyIdleHandler(TObject *Sender, bool &Done)
{
  StatusMsg(Form1, Canvas, "This represents a background process.", true);
}

/*
This is the OnClick event handler for Button1.  It
simulates a lengthy process
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  int X, Y;
  StatusMsg(Form1, Canvas, "The Button1Click handler is starting", false);
  for (int I = 0; I < 100; I++)
  {
    randomize();
    for (int J = 0; J < 100; J++)
      Y = random(J);
    X = random(I);
  }
  StatusMsg(Form1, Canvas, "The Button1Click handler is done", false);
}

/*
This is the OnClick event handler for button2.  It
simulates a lengthy process that allows other messages to be
processed.
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  int X, Y;
  StatusMsg(Form1, Canvas, "The Button2Click handler is starting", false);
  for (int I = 0; I < 100; I++)
  {
    randomize();
    for (int J = 0; J < 100; J++)
    {
      Y = random(J);
    }
    // Yield to OnIdle or other messages
    Application->HandleMessage();
    X = random(I);
  }
  StatusMsg(Form1, Canvas, "The Button2Click handler is done", false);
}
```

Delphi Examples:

```
{
The following code demonstrates background processing using
the OnIdle event, and using the HandleMessage method to
permit messages or background processing to get through.
Note: You must add MyIdleHandler to the Form1 methods.
}
var
{ global variables to show the order of events. }
  XPos, YPos, Delta: integer;

{ This is a utility procedure to display messages }
{ add this at the beginning of the implementation section }
procedure StatusMsg(
  MyForm : TForm1; Canvas : TCanvas; Message : string; Bkg : Boolean);
begin
  if not bkg then
    Canvas.Font.Style := [fsBold]; {foreground messages are bold }
  Canvas.TextOut(XPos, YPos, Message);
  if not bkg then
    Canvas.Font.Style := [];
  { change Xpos and YPos to prepare for the next message }
  Ypos := Ypos + Delta;
  if YPos >= MyForm.ClientHeight - 10 then
  begin
    YPos := 10;
    Xpos := Xpos + 180;
  end;
  if (Xpos >= MyForm.ClientWidth - 100) then
  begin
    if (Canvas.Font.Color = clRed) then
```

```
      Canvas.Font.Color := clBlack
    else
      Canvas.Font.Color := clRed;
    Xpos := 10;
  end;
end;

{This is the Form's OnCreate event handler.  }
{ It initializes global variables and sets the  the OnIdle
event handler }
procedure TForm1.FormCreate(Sender: TObject);
begin
  Button1.Caption := 'Do not yield';
  Button2.Caption := 'Handle Message';
  Application.OnIdle:= MyIdleHandler;
  XPos := 10;
  YPos := 10;
  Delta := Abs(Canvas.Font.Height) + 1;
 end;

{
This is the OnIdle event handler. It is set in the Form's
OnCreate event handler, so you need only add it as a private
method of the form.  Usually it would perform some
background processing for the application.  This one simply
writes a message to let you know when it's there.
}
procedure TForm1.MyIdleHandler(Sender: TObject; var Done: Boolean);
begin
  StatusMsg(
    Form1, Canvas, 'This represents a background process.', True);
end;

{
Set this method as the OnClick event handler of Button1.  It
performs a calculation without yielding to other messages or
idle time.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, X, Y: Word;
begin
  StatusMsg(
    Form1, Canvas, 'The synchronous handler is starting', False);
  I := 0;
  J := 0;
  while I < 10 do
  begin
    Randomize;
    while J < 10 do
    begin
      Y := Random(J);
      Inc(J);
    end;
    X := Random(I);
    Inc(I);
  end;
  StatusMsg(
    Form1, Canvas, 'The synchronous handler is done', False);
end;

{
Set this method as the OnClick event handler of Button2.  It
performs a calculation but calls HandleMessage to allow idle
time or asynchronous message processing.
}
```

```
procedure TForm1.Button2Click(Sender: TObject);
var
  I, J, X, Y: Word;
begin
 StatusMsg(
    Form1, Canvas, 'The asynchronous handler is starting', False);
  I := 0;
  J := 0;
  while I < 10 do
  begin
    Randomize;
    while J < 10 do
    begin
      Y := Random(J);
      Inc(J);
    end;
    X := Random(I);
    Inc(I);
    { yield to OnIdle or other messages }
    Application.HandleMessage;
  end;
  StatusMsg(
    Form1, Canvas, 'The asynchronous handler is done', False);
end;
```

See Also

OnIdle (see page 13733)

ProcessMessages (see page 13691)

UpdateAction (see page 13701)

72.1.1.4.12 TApplication.HelpCommand Method

Provides quick access to any of the Help commands in the WinHelp API (application programming interface).

Pascal

```
function HelpCommand(Command: Integer; Data: Longint): Boolean;
```

C++

```
__fastcall Boolean HelpCommand(int Command, Longint Data);
```

Description

Use HelpCommand to send a command to WinHelp. Before sending the help command to the Windows help engine, HelpCommand generates an OnHelp (see page 13730) event on the active form or on TApplication (see page 13660). Finally, the command is forwarded to WinHelp if there is no OnHelp (see page 13730) event handler or if the OnHelp (see page 13730) event handler indicates that WinHelp should be called.

For information about Windows help commands and the data passed to them, see the WinHelp topic in the Windows Help system.

See Also

HelpContext (see page 13679)

HelpFile (see page 13711)

HelpJump (see page 13679)

OnHelp (see page 13730)

OnHelp (see page 13959)

72.1.1.4.13 TApplication.HelpContext Method

Displays a specified help topic.

Pascal

```
function HelpContext(Context: THelpContext): Boolean;
```

C++

```
__fastcall Boolean HelpContext(THelpContext Context);
```

Description

Use HelpContext to bring up the help topic specified by the Context parameter from the file specified in the CurrentHelpFile (see page 13707) property. HelpContext generates an OnHelp (see page 13730) event (either on the active form or on the application object itself). If there is no OnHelp (see page 13730) event handler, or if the event handler indicates that WinHelp should be called, then HelpContext invokes the Windows help engine, passing it the context ID.

For example, if the Context value is 714, the HelpContext method displays the screen with the context help ID of 714 in the help file specified by CurrentHelpFile (see page 13707).

HelpContext returns false if CurrentHelpFile (see page 13707) is an empty string, meaning the application has no help file assigned. In all other cases, HelpContext returns true.

See Also

HelpCommand (see page 13678)

HelpContext (see page 5826)

HelpFile (see page 13711)

HelpJump (see page 13679)

OnHelp (see page 13730)

CurrentHelpFile (see page 13707)

HelpKeyword (see page 13680)

Help and Hint Properties

72.1.1.4.14 TApplication.HelpJump Method

Displays a specified help topic.

Pascal

```
function HelpJump(const JumpID: string): Boolean;
```

C++

```
__fastcall Boolean HelpJump(const AnsiString JumpID);
```

Description

Call HelpJump to bring up a topic identified by its symbolic name. HelpJump displays the topic identified by the JumpID parameter from the file specified in the Current HelpFile (see page 13711) property. HelpJump generates an OnHelp (see page 13730) event (either on the active form or on the application object itself). If there is no OnHelp (see page 13730) event handler, or if the event handler indicates that WinHelp should be called, then HelpJump invokes the Windows help engine, passing it the topic name.

HelpJump returns false if CurrentHelpFile (see page 13707) is an empty string, meaning the application has no help file

assigned. In all other cases, HelpJump returns true.

See Also

HelpCommand (see page 13678)

HelpContext (see page 13679)

HelpFile (see page 13711)

OnHelp (see page 13730)

72.1.1.4.15 TApplication.HelpKeyword Method

Displays a specified help topic.

Pascal

```
function HelpKeyword(const Keyword: string): Boolean;
```

C++

```
__fastcall Boolean HelpKeyword(const AnsiString Keyword);
```

Description

Use HelpKeyword to bring up the help topic specified by the Keyword parameter from the file specified in the CurrentHelpFile (see page 13707) property. HelpKeyword invokes the current help system, passing it the string specified by Keyword. If the current system is WinHelp, this displays the topic that has an Alink of Keyword.

HelpKeyword returns false if the current help system does not support the keyword lookups.

See Also

HelpCommand (see page 13678)

HelpKeyword (see page 5826)

HelpFile (see page 13711)

HelpJump (see page 13679)

OnHelp (see page 13730)

CurrentHelpFile (see page 13707)

72.1.1.4.16 TApplication.HelpShowTableOfContents Method

Shows the Table of Contents within the help system.

Pascal

```
function HelpShowTableOfContents: Boolean;
```

C++

```
__fastcall Boolean HelpShowTableOfContents();
```

Description

Use HelpShowTableOfContents to bring up the table of contents for the help system specified in the CurrentHelpFile (see page 13707) property. This method is preferred to calling "Application.HelpSystem.ShowTableOfContents", which cannot be used as the first call into the help system.

72.1.1.4.17 TApplication.HideHint Method

Hides the current hint.

Pascal

```
procedure HideHint;
```

C++

```
__fastcall HideHint();
```

Description

Applications rarely, if ever, call HideHint directly. HideHint is used internally to hide the hint window at the appropriate times according to predetermined conditions and timer values. For example, CancelHint (see page 13669) calls HideHint.

See Also

CancelHint (see page 13669)

HintPause (see page 13715)

HintShortPause (see page 13716)

72.1.1.4.18 TApplication.HintMouseMessage Method

Regulates the display of hint windows.

Pascal

```
procedure HintMouseMessage(Control: TControl; var Message: TMessage);
```

C++

```
__fastcall HintMouseMessage(TControl * Control, TMessage Message);
```

Description

This method is called automatically when the mouse passes over a control with a Hint (see page 13712). Applications rarely, if ever, call HintMouseMessage directly. HintMouseMessage is used internally to manage the display of the hint windows.

See Also

CancelHint (see page 13669)

HintPause (see page 13715)

HintShortPause (see page 13716)

72.1.1.4.19 TApplication.HookMainWindow Method

Enables a native Windows dialog box to receive messages sent to the application's main window.

Pascal

```
procedure HookMainWindow(Hook: TWindowHook);
```

C++

```
__fastcall HookMainWindow(TWindowHook Hook);
```

Description

Use HookMainWindow to ensure that a native Windows dialog box behaves correctly as a child of the application, not as a stand-alone window. For example, switching among applications with Alt+Tab treats the application as a single task after calling

HookMainWindow, rather than treating the native Windows dialog box as a separate task.

When the window identified by the Handle (see page 13710) property receives relevant dialog messages, it passes them to the dialog procedure passed as the Hook parameter.

There is no problem with leaving a dialog box hooked into the main window, even for extended periods. However, should the dialog box close, call the UnhookMainWindow (see page 13701) method to release the hook.

See Also

Handle (see page 13710)

UnhookMainWindow (see page 13701)

72.1.1.4.20 TApplication.HookSynchronizeWakeup Method

Pascal

```
procedure HookSynchronizeWakeup;
```

C++

```
__fastcall HookSynchronizeWakeup();
```

Description

This is HookSynchronizeWakeup, a member of class TApplication.

72.1.1.4.21 TApplication.Idle Method

Pascal

```
procedure Idle(const Msg: TMsg);
```

C++

```
__fastcall Idle(const TMsg Msg);
```

Description

This is Idle, a member of class TApplication.

72.1.1.4.22 TApplication.Initialize Method

Provides an opportunity to initialize subsystems.

Pascal

```
procedure Initialize;
```

C++

```
__fastcall Initialize();
```

Description

Initialize is the first method called by the project source file. It calls the InitProc procedure pointer. By default, the call to Initialize for the application does nothing because the default InitProc pointer is nil (Delphi) or NULL (C++) .

To use Initialize, the InitProc pointer must be predefined. This can be accomplished in one of two ways:

In Delphi, you can include a unit that assigns a procedure to InitProc in its initialization section, such as the ComObj (see page 5525) unit. You can make this assignment in the initialization section of any of your units.

In both Delphi and C++, you can create a custom initialization procedure that assigns a value to the InitProc pointer, and add a

call to this procedure to the project source prior to the call to Initialize. (In Delphi, you can add it to the initialization section of the unit in which it is declared. In C++, you can use the pragma startup directive in that unit.)

Warning: Only one instance of InitProc can be defined in an application. If more than one unit assigns a value to InitProc, only the last assignment will work. You can, however, call the previous value of InitProc from an initialization procedure, so that all initialization procedures are executed.

For projects that do not assign a value to InitProc, the call to Initialize can be safely deleted from the project source.

Note: Although Initialize is the first method called in the main project source code, it is not the first code that is executed in a GUI application. For example, in Delphi, the application first executes the initialization section of all the units used by the application.

C++ Examples:

```
/*
StepBy, CreateForm, Run, Show, Update, Initialize example
This example shows progress of loading forms as an application starts up. The code example is
placed in the project source (*.dpr) file. To see project source, select the View | Project
Source menu item.
You will need to set up your project with the following steps before using the code example:
o          Add four additional forms to a default project.
o          Place a TProgressBar on Form5
o          Take the Project|Options|Forms menu option and place Form5 on the available forms
list.
o          Change the code of your project (*.dpr) file to look like the example.
*/
#include <vcl\vcl.h>
#include "Unit5.h"
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("TAppCreateForm.cpp", Form1);
USERES("Project1.res");
USEFORM("Unit2.cpp", Form2);
USEFORM("Unit3.cpp", Form3);
USEFORM("Unit4.cpp", Form4);
USEFORM("Unit5.cpp", Form5);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
  try
  {
    Application->Initialize();
    SetApplicationMainFormOnTaskBar(Application, true);
    Application->CreateForm(__classid(TForm5), &Form5);
    Form5->ProgressBar1->Max = 100;
    Form5->Show();   // show a splash screen contain ProgressBar control
    Form5->Update(); // force display of Form5
    Application->CreateForm(__classid(TForm1), &Form1);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form1 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm2), &Form2);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form2 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm3), &Form3);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form3 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
```

```
      Application->CreateForm(__classid(TForm4), &Form4);
      Form5->ProgressBar1->StepBy(25);
      Form5->Label1->Caption =  "Form4 loaded successfully.";
      Form5->Update(); // force display of Form5
      Sleep(3000);
//    delete Form5;
      Application->Run();
    }
    catch (Exception &exception)
    {
      Application->ShowException(&exception);
    }
    return 0;
  }
```

Delphi Examples:

```
{
This example shows progress of loading forms as an
application starts up. The code example is placed in the
project source (*.dpr) file.  To see project source, select
the View | Project Source menu item.
You will need to set up your project with the following steps
before using the code example:
    Add four additional forms to a default project.
    Place a TProgressBar on Form5
    Take the Project|Options|Forms menu option and place
          Form5 on the available forms list.
    Change the code of your project (*.dpr) file to look
          like the example.
}
begin
  Application.Initialize;
  with TForm5.Create(nil) do
  try
    Application.MainFormOnTaskbar := True;
    ProgressBar1.Max := 100;
    Show;  // show a splash screen contain ProgressBar control
    Update; // force display of Form5
    Application.CreateForm(TForm1, Form1);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form1 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm2, Form2);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form2 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm3, Form3);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form3 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm4, Form4);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form4 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
  finally
    Free;
  end;
  Application.Run;
```

See Also

InitProc (see page 22646)

72.1.1.4.23 TApplication.IsDlgMsg Method

Pascal

```
function IsDlgMsg(var Msg: TMsg): Boolean;
```

C++

```
__fastcall Boolean IsDlgMsg(TMsg Msg);
```

Description

This is IsDlgMsg, a member of class TApplication.

72.1.1.4.24 TApplication.IsHintMsg Method

Pascal

```
function IsHintMsg(var Msg: TMsg): Boolean;
```

C++

```
__fastcall Boolean IsHintMsg(TMsg Msg);
```

Description

This is IsHintMsg, a member of class TApplication.

72.1.1.4.25 TApplication.IsKeyMsg Method

Pascal

```
function IsKeyMsg(var Msg: TMsg): Boolean;
```

C++

```
__fastcall Boolean IsKeyMsg(TMsg Msg);
```

Description

This is IsKeyMsg, a member of class TApplication.

72.1.1.4.26 TApplication.IsMDIMsg Method

Pascal

```
function IsMDIMsg(var Msg: TMsg): Boolean;
```

C++

```
__fastcall Boolean IsMDIMsg(TMsg Msg);
```

Description

This is IsMDIMsg, a member of class TApplication.

72.1.1.4.27 TApplication.IsPreProcessMessage Method

Pascal

```
function IsPreProcessMessage(var Msg: TMsg): Boolean;
```

C++

```
__fastcall Boolean IsPreProcessMessage(TMsg Msg);
```

Description

This is IsPreProcessMessage, a member of class TApplication.

72.1.1.4.28 TApplication.IsRightToLeft Method

Specifies whether the application adjusts its interface for Middle Eastern locales.

Pascal

```
function IsRightToLeft: Boolean;
```

C++

```
__fastcall Boolean IsRightToLeft();
```

Description

Use IsRightToLeft to determine whether the application makes any adjustments of control alignment, text direction, or scroll bar placement. IsRightToLeft returns true if the application's BiDiMode (see page 13706) indicates that any of these should be reversed and the application is running on a system with a Middle Eastern locale.

See Also

BiDiMode (see page 13706)

userighttoleftreading (see page 13702)

72.1.1.4.29 TApplication.IsShortCut Method

Pascal

```
function IsShortCut(var Message: TWMKey): Boolean;
```

C++

```
__fastcall Boolean IsShortCut(TWMKey Message);
```

Description

This is IsShortCut, a member of class TApplication.

72.1.1.4.30 TApplication.MessageBox Method

Displays a specified message to the user.

Pascal

```
function MessageBox(const Text: PChar; const Caption: PChar; Flags: Longint = MB_OK): Integer;
```

C++

```
__fastcall int MessageBox(const const char * Text, const const char * Caption, Longint Flags =
MB_OK);
```

Description

Use MessageBox to display a generic dialog box a message and one or more buttons. Caption is the caption of the dialog box and is optional.

MessageBox is an encapsulation of the Windows API MessageBox function. TApplication (see page 13660)'s encapsulation of MessageBox automatically supplies the missing window handle parameter needed for the Windows API function.

The value of the Text parameter is the message, which can be longer than 255 characters if necessary. Long messages are automatically wrapped in the message box.

The value of the Caption parameter is the caption that appears in the title bar of the dialog box. Captions can be longer than 255 characters, but don't wrap. A long caption results in a wide message box.

The Flags parameter specifies what buttons appear on the message box and the behavior (possible return values). The following table lists the possible values. These values can be combined to obtain the desired effect.

Value	Meaning
MB_ABORTRETRYIGNORE	The message box contains three push buttons: Abort, Retry, and Ignore.
MB_OK	The message box contains one push button: OK. This is the default.
MB_OKCANCEL	The message box contains two push buttons: OK and Cancel.
MB_RETRYCANCEL	The message box contains two push buttons: Retry and Cancel.
MB_YESNO	The message box contains two push buttons: Yes and No.
MB_YESNOCANCEL	The message box contains three push buttons: Yes, No, and Cancel.

MessageBox returns 0 if there isn't enough memory to create the message box. Otherwise it returns one of the following values:

ValueNumeric valueMeaning

Value	Numeric value	Meaning
IDOK	1	The user chose the OK button.
IDCANCEL	2	The user chose the Cancel button.
IDABORT	3	The user chose the Abort button.
IDRETRY	4	The user chose the Retry button.
IDIGNORE	5	The user chose the Ignore button.
IDYES	6	The user chose the Yes button.
IDNO	7	The user chose the No button.

C++ Examples:

```
/*
The following code causes "stay on top" forms to allow a
MessageBox to appear on top. After the message box is
closed, the topmost forms are restored so that they continue
to float to the top.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Application->NormalizeTopMosts();
#ifdef _DELPHI_STRING_UNICODE
Application->MessageBox(L"This should be on top.", L"Look", MB_OKCANCEL);
#else
Application->MessageBox("This should be on top.", "Look", MB_OKCANCEL);
#endif
Application->RestoreTopMosts();
}
```

Delphi Examples:

```
{
The following code causes "stay on top" forms to allow a
MessageBox to appear on top. After the message box is
closed, the topmost forms are restored so that they continue
to float to the top.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  with Application do
  begin
    NormalizeTopMosts;
    MessageBox('This should be on top.', 'Look', MB_OK);    // [smbOK]
    RestoreTopMosts;
  end;
end;
```

See Also

MessageDlgPosHelp (see page 11539)

ShowMessage (see page 11541)

MessageDlg (see page 11535)

72.1.1.4.31 TApplication.Minimize Method

Shrinks an application to the Windows task bar.

Pascal

```
procedure Minimize;
```

C++

```
__fastcall Minimize();
```

Description

Call Minimize to minimize the application. When the user minimizes the application, the Minimize method is automatically called.

Note: Don't confuse the Minimize method, which minimizes an entire application, with minimizing a form or window. To minimize, maximize, or restore a window or form, change the value of its WindowState property.

C++ Examples:

```
/*
The following code minimizes an application when it's
deactivated. Note that AppDeactivate should be declared a
method of TForm1.  Click anywhere outside of the form
to deactivate it.
*/
void __fastcall TForm1::AppDeactivate(TObject *Sender)
{
  Application->Minimize();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnDeactivate = AppDeactivate;
}
```

Delphi Examples:

```
{
The following code minimizes an application when it's
deactivated. Note that AppDeactivate should be declared a
method of TForm1.  Click anywhere outside of the form
to deactivate it.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnDeactivate := AppDeactivate;
end;
```

```
procedure TForm1.AppDeactivate(Sender: TObject);
begin
  Application.Minimize;
end;
```

See Also

Icon (see page 13717)

OnMinimize (see page 13739)

Restore (see page 13694)

WindowState (see page 13942)

72.1.1.4.32 TApplication.ModalFinished Method

Track closing of modal form.

Pascal

```
procedure ModalFinished;
```

C++

```
__fastcall ModalFinished();
```

Description

Call ModalFinished to indicate that a modal form is being closed. ModalFinished is called by the ShowModal method of TCustomForm (see page 13849).

ModalFinished calls the OnModalEnd (see page 13740) event handler, if defined.

See Also

OnModalEnd (see page 13740)

ModalStarted (see page 13689)

ShowModal (see page 13890)

72.1.1.4.33 TApplication.ModalStarted Method

Track opening of modal form.

Pascal

```
procedure ModalStarted;
```

C++

```
__fastcall ModalStarted();
```

Description

Call ModalStarted to indicate that a modal form is being opened. ModalStarted is called by the ShowModal method of TCustomForm (see page 13849).

ModalStarted calls the OnModalStart event handler, if defined.

See Also

OnModalStart

ModalFinished (see page 13689)

ShowModal (see page 13890)

72.1.1.4.34 TApplication.NormalizeAllTopMosts Method

Makes forms that have been designated as topmost forms (their FormStyle is fsStayOnTop) behave as if they were not topmost forms.

Pascal

```
procedure NormalizeAllTopMosts;
```

C++

```
__fastcall NormalizeAllTopMosts();
```

Description

Use NormalizeAllTopMosts to allow other windows appear on top of a topmost form. Normally, topmost forms remain on top of other windows, and so may obscure other windows.

NormalizeAllTopMosts behaves like the NormalizeTopMosts (see page 13690) method, except that it also includes the application's Main Window.

See Also

FormStyle (see page 13912)

NormalizeTopMosts (see page 13690)

RestoreTopMosts (see page 13695)

MainForm (see page 13718)

72.1.1.4.35 TApplication.NormalizeTopMosts Method

Makes forms that have been designated as topmost forms (their FormStyle is fsStayOnTop) behave as if they were not topmost forms.

Pascal

```
procedure NormalizeTopMosts;
```

C++

```
__fastcall NormalizeTopMosts();
```

Description

Use NormalizeTopMosts to allow a message box or dialog box that is displayed using the Windows API functions (such as MessageBox (see page 13686) and MessageDlg) directly, appear on top of a topmost form. Otherwise the topmost form remains on top, and may obscure the message box.

Calling NormalizeTopMosts is not necessary when using functions to display message boxes.

To return the forms designated as fsStayOnTop to be topmost again, call RestoreTopMosts (see page 13695).

C++ Examples:

```
/*
The following code causes "stay on top" forms to allow a
MessageBox to appear on top. After the message box is
closed, the topmost forms are restored so that they continue
to float to the top.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Application->NormalizeTopMosts();
```

```
#ifdef _DELPHI_STRING_UNICODE
Application->MessageBox(L"This should be on top.", L"Look", MB_OKCANCEL);
#else
Application->MessageBox("This should be on top.", "Look", MB_OKCANCEL);
#endif
Application->RestoreTopMosts();
}
```

Delphi Examples:

```
{
The following code causes "stay on top" forms to allow a
MessageBox to appear on top. After the message box is
closed, the topmost forms are restored so that they continue
to float to the top.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Application do
  begin
    NormalizeTopMosts;
    MessageBox('This should be on top.', 'Look', MB_OK);    // [smbOK]
    RestoreTopMosts;
  end;
end;
```

See Also

FormStyle (see page 13912)

RestoreTopMosts (see page 13695)

NormalizeAllTopMosts (see page 13690)

MessageBox (see page 13686)

MessageDlg (see page 11535)

72.1.1.4.36 TApplication.ProcessMessages Method

Interrupts the execution of an application so that it can process the message queue.

Pascal

```
procedure ProcessMessages;
```

C++

```
__fastcall ProcessMessages();
```

Description

Call ProcessMessages to permit the application to process messages that are currently in the message queue. ProcessMessages cycles the Windows message loop until it is empty, and then returns control to the application.

Note: Neglecting message processing affects only the application calling ProcessMessages, not other applications. In lengthy operations, calling ProcessMessages periodically allows the application to respond to paint and other messages.

Note: ProcessMessages does not allow the application to go idle, whereas HandleMessage (see page 13674) does.

C++ Examples:

```
/*
This example uses two buttons that are long enough to
```

```
accommodate lengthy captions on a form. When the user clicks
the button with the caption Ignore Messages, the code begins
to generate a long series of random numbers. If the user
tries to resize the form while the handler is running,
nothing happens until the handler is finished. When the user
clicks the button with the caption Process Messages, more
random numbers are generated, but the application can still
respond to a series of events, such as resizing the form
and printing text.
Note: How quickly these event handlers run depends on the
microprocessor of your computer. A message appears on the
form informing you when the handler has finished executing.
*/

int magicnumber = 20;

void __fastcall TForm1::FormCreate(TObject* Sender)
{
  Button1->Caption = "Ignore Messages";
  Button2->Caption = "Handle Message";
}

void __fastcall TForm1::Button1Click(TObject* Sender)
{
  int x, y;
  Canvas->TextOut(10, 10, "The Button1Click handler has started");
  for (int i = 0; i < magicnumber; i++)
 {
    Randomize();
    for (int j = 0; j < magicnumber; j++)
    {
      Sleep(10);
      y = random(j);
    }
    x = random(i);
  }
  Canvas->TextOut(10, 10, "The Button1Click handler is finished   ");
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  int x, y;
  Canvas->TextOut(10, 10, "The Button2Click handler has started");
  for (int i = 0; i < magicnumber; i++)
  {
    Randomize();
    for (int j = 0; j < magicnumber; j++)
    {
      y = random(j);
      Sleep(10);
      Application->ProcessMessages();
    }
    x = random(i);
  }
  Canvas->TextOut(10, 10, "The Button2Click handler is finished   ");
}
```

Delphi Examples:

```
{
This example uses two buttons that are long enough to
accommodate lengthy captions on a form. When the user clicks
the button with the caption Ignore Messages, the code begins
to generate a long series of random numbers. If the user
tries to resize the form while the handler is running,
nothing happens until the handler is finished. When the user
clicks the button with the caption Process Messages, more
```

```
  random numbers are generated, but the application can still
  respond to a series of events, such as resizing the form
  and printing text.
  Note: How quickly these event handlers run depends on the
  microprocessor of your computer. A message appears on the
  form informing you when the handler has finished executing.
  }
  const magicnumber = 500;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Button1.Caption := 'Ignore Messages';
    Button2.Caption := 'Process Messages';
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, J: Integer;
    X, Y: Word;
  begin
    I := 0;
    J := 0;
    Canvas.TextOut(10, 10, 'The Button1Click handler has started');
    while I < magicnumber do
    begin
      Randomize;
      while J < magicnumber do
      begin
        Sleep(10);
        Y := Random(J);
        Inc(J);
      end;
      X := Random(I);
      Inc(I);
    end;
    Canvas.TextOut(10, 10, 'The Button1Click handler is finished   ');
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  var
    I, J: Integer;
    X, Y: Word;
  begin
    I := 0;
    J := 0;
    Canvas.TextOut(10, 10, 'The Button2Click handler has started');
    while I < magicnumber do
    begin
      Randomize;
      while J < magicnumber do
      begin
        Y := Random(J);
        Inc(J);
        Sleep(10);
        Application.ProcessMessages;
      end;
      X := Random(I);
      Inc(I);
    end;
    Canvas.TextOut(10, 10, 'The Button2Click handler is finished   ');
  end;
```

See Also

HandleMessage (see page 13674)

OnMessage (see page 13737)

72.1.1.4.37 TApplication.RemovePopupForm Method

Pascal

```
procedure RemovePopupForm(APopupForm: TCustomForm);
```

C++

```
__fastcall RemovePopupForm(TCustomForm APopupForm);
```

Description

This is RemovePopupForm, a member of class TApplication.

72.1.1.4.38 TApplication.Restore Method

Restores a minimized application to its normal size.

Pascal

```
procedure Restore;
```

C++

```
__fastcall Restore();
```

Description

Use Restore to restore the application to its previous size before it was minimized. When the user restores the application to normal size, Restore is automatically called.

Note: Don't confuse the Restore method, which restores the entire application, with restoring a form or window to its original size. To minimize, maximize, and restore a window or form, change the value of its WindowState property.

C++ Examples:

```
/*
This example uses a timer on a form. When the application
runs and the user minimizes the application, the timer
starts and the application returns to its normal size when
an OnTimer event occurs.  The timer then shuts down until
the next time the form is minimized.  Be sure to declare the
AppStartTimer method as a public method of TForm1.  Place a
TTimer object in the form and enter Timer1Timer in the OnTimer
event.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnMinimize = AppStartTimer;
  Timer1->Interval = 1000;
}

void __fastcall TForm1::AppStartTimer(TObject *Sender)
{
  Timer1->Enabled = true;
}

//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Application->Restore();
  Timer1->Enabled = false;
}
```

Delphi Examples:

```
{
This example uses a timer on a form. When the application
runs and the user minimizes the application, the timer
starts and the application returns to its normal size when
an OnTimer event occurs.  The timer then shuts down until
the next time the form is minimized.  Be sure to declare the
AppStartTimer method as a public method of TForm1.  Place a
TTimer object in the form and enter Timer1Timer in the OnTimer
event.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnMinimize := AppStartTimer;
end;

procedure TForm1.AppStartTimer(Sender: TObject);
begin
  Timer1.Enabled := True;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Application.Restore;
  Timer1.Enabled := False;
end;
```

See Also

WindowState (see page 13942)

72.1.1.4.39 TApplication.RestoreTopMosts Method

Restores forms designated as fsStayOnTop to be topmost again.

Pascal

```
procedure RestoreTopMosts;
```

C++

```
__fastcall RestoreTopMosts();
```

Description

Use RestoreTopMosts to return forms that were originally designated as topmost forms (FormStyle is fsStayOnTop), then temporarily changed to be non-topmost forms with the NormalizeTopMosts (see page 13690) or NormalizeAllTopMosts (see page 13690) method, to a topmost position.

C++ Examples:

```
/*
The following code causes "stay on top" forms to allow a
MessageBox to appear on top. After the message box is
closed, the topmost forms are restored so that they continue
to float to the top.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Application->NormalizeTopMosts();
#ifdef _DELPHI_STRING_UNICODE
Application->MessageBox(L"This should be on top.", L"Look", MB_OKCANCEL);
#else
Application->MessageBox("This should be on top.", "Look", MB_OKCANCEL);
#endif
```

```
Application->RestoreTopMosts();
}
```

Delphi Examples:

```
{
The following code causes "stay on top" forms to allow a
MessageBox to appear on top. After the message box is
closed, the topmost forms are restored so that they continue
to float to the top.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Application do
  begin
    NormalizeTopMosts;
    MessageBox('This should be on top.', 'Look', MB_OK);    // [smbOK]
    RestoreTopMosts;
  end;
end;
```

See Also

NormalizeTopMosts (see page 13690)

NormalizeAllTopMosts (see page 13690)

72.1.1.4.40 TApplication.Run Method

Executes the application.

Pascal

```
procedure Run;
```

C++

```
__fastcall Run();
```

Description

Do not call Run. When creating a new project, The IDE automatically creates a main program block in the project file that calls the Run method. When the application is executed, the application's Run method is called.

Run contains the application's main message loop. The call to Run does not return until the application's message loop terminates.

C++ Examples:

```
/*
StepBy, CreateForm, Run, Show, Update, Initialize example
This example shows progress of loading forms as an application starts up. The code example is
placed in the project source (*.dpr) file. To see project source, select the View | Project
Source menu item.
You will need to set up your project with the following steps before using the code example:
o          Add four additional forms to a default project.
o          Place a TProgressBar on Form5
o          Take the Project|Options|Forms menu option and place Form5 on the available forms
list.
o          Change the code of your project (*.dpr) file to look like the example.
*/
#include <vcl\vcl.h>
#include "Unit5.h"
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("TAppCreateForm.cpp", Form1);
USERES("Project1.res");
```

```
USEFORM("Unit2.cpp", Form2);
USEFORM("Unit3.cpp", Form3);
USEFORM("Unit4.cpp", Form4);
USEFORM("Unit5.cpp", Form5);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
  try
  {
    Application->Initialize();
    SetApplicationMainFormOnTaskBar(Application, true);
    Application->CreateForm(__classid(TForm5), &Form5);
    Form5->ProgressBar1->Max = 100;
    Form5->Show();   // show a splash screen contain ProgressBar control
    Form5->Update(); // force display of Form5
    Application->CreateForm(__classid(TForm1), &Form1);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form1 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm2), &Form2);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form2 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm3), &Form3);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form3 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm4), &Form4);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form4 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
//  delete Form5;
    Application->Run();
  }
  catch (Exception &exception)
  {
    Application->ShowException(&exception);
  }
  return 0;
}
```

Delphi Examples:

```
{
This example shows progress of loading forms as an
application starts up. The code example is placed in the
project source (*.dpr) file.  To see project source, select
the View | Project Source menu item.
You will need to set up your project with the following steps
before using the code example:
    Add four additional forms to a default project.
    Place a TProgressBar on Form5
    Take the Project|Options|Forms menu option and place
          Form5 on the available forms list.
    Change the code of your project (*.dpr) file to look
          like the example.
}
begin
  Application.Initialize;
  with TForm5.Create(nil) do
  try
    Application.MainFormOnTaskbar := True;
    ProgressBar1.Max := 100;
```

```
    Show;  // show a splash screen contain ProgressBar control
    Update; // force display of Form5
    Application.CreateForm(TForm1, Form1);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form1 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm2, Form2);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form2 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm3, Form3);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form3 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm4, Form4);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form4 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
  finally
    Free;
  end;
  Application.Run;
```

See Also

Create (see page 13666)

CreateForm (see page 13670)

CreateHandle (see page 13672)

MainForm (see page 13718)

Terminate (see page 13699)

72.1.1.4.41 TApplication.ShowException Method

Displays a message box for exceptions that are not caught by application code.

Pascal

```
procedure ShowException(E: Exception);
```

C++

```
__fastcall ShowException(Exception E);
```

Description

The HandleException (see page 13674) method calls ShowException by default if no handler is specified for the OnException (see page 13728) event. If an exception handler filters exceptions, ShowException is called for those exceptions that are not filtered out.

To specify how exceptions are handled for an application, write a handler for the OnException (see page 13728) event.

Note: Calling ShowException is rarely necessary in a component-based application, since the default exception handler calls ShowException automatically.

C++ Examples:

```
/*
```

```
In addition to displaying the exception message, which
happens by default, the following code shuts down the
application when an exception is not caught and handled.
AppException should be declared a method of TForm1.
*/

#include <System.hpp>
#include <SysUtils.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnException = AppException;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::AppException(TObject *Sender, Exception *E)
{
  Application->ShowException(E);
  Application->Terminate();
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  throw(Exception("Hardware error: Divide by 0"));
}
```

Delphi Examples:

```
{
In addition to displaying the exception message, which
happens by default, the following code shuts down the
application when an exception is not caught and handled.
AppException should be declared a method of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnException := AppException;
end;
procedure TForm1.AppException(Sender: TObject; E: Exception);
begin
  Application.ShowException(E);
  Application.Terminate;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  raise EPasswordInvalid.Create('Incorrect password entered');
end;
```

See Also

HandleException (see page 13674)

OnException (see page 13728)

72.1.1.4.42 TApplication.Terminate Method

Ends application execution.

Pascal

```
procedure Terminate;
```

C++

```
__fastcall Terminate();
```

Description

Call Terminate to end the application programmatically. By calling Terminate rather than freeing the application object, you allow the application to shut down in an orderly fashion.

Terminate calls the Windows API PostQuitMessage function to perform an orderly shutdown of the application. Terminate is not immediate.

Terminate is called automatically on a WM_QUIT message and when the main form closes.

C++ Examples:

```
/*
In addition to displaying the exception message, which
happens by default, the following code shuts down the
application when an exception is not caught and handled.
AppException should be declared a method of TForm1.
*/

#include <System.hpp>
#include <SysUtils.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnException = AppException;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::AppException(TObject *Sender, Exception *E)
{
  Application->ShowException(E);
  Application->Terminate();
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  throw(Exception("Hardware error: Divide by 0"));
}
```

Delphi Examples:

```
{
In addition to displaying the exception message, which
happens by default, the following code shuts down the
application when an exception is not caught and handled.
AppException should be declared a method of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnException := AppException;
end;
procedure TForm1.AppException(Sender: TObject; E: Exception);
begin
  Application.ShowException(E);
  Application.Terminate;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  raise EPasswordInvalid.Create('Incorrect password entered');
end;
```

See Also

Terminated (see page 13723)

72.1.1.4.43 TApplication.UnhookMainWindow Method

Releases a dialog procedure previously hooked by a call to the HookMainWindow (see page 13681) method.

Pascal

```
procedure UnhookMainWindow(Hook: TWindowHook);
```

C++

```
__fastcall UnhookMainWindow(TWindowHook Hook);
```

Description

Use UnhookMainWindow to release the hooked window. Specify the dialog procedure as the value of the Hook parameter.

The TWindowHook (see page 14382) type is the parameter type used for UnhookMainWindow. It is a method-pointer type used for the dialog-procedures of non-VCL dialog boxes. The dialog procedure is similar to a window procedure for a window, in that it processes messages for the dialog box, but its syntax is somewhat different.

For more information about Windows hooks see the SetWindowsEx and UnhookWindowsEx functions in the Windows Help File.

See Also

HookMainWindow (see page 13681)

72.1.1.4.44 TApplication.UnhookSynchronizeWakeup Method

Removes the method that was assigned to the WakeMainThread variable by HookSynchronizeWakeup (see page 13682).

Pascal

```
procedure UnhookSynchronizeWakeup;
```

C++

```
__fastcall UnhookSynchronizeWakeup();
```

Description

Applications rarely, if ever, call UnhookSynchronizeWakeup directly. UnhookSynchronizeWakeup is called from the TApplication (see page 13660) destructor. It resets the WakeMainThread variable to nil (Delphi) or NULL (C++) if that variable was set by the HookSynchronizeWakeup (see page 13682) method.

Warning: You should not modify WakeMainThread for GUI applications because a working handler has been assigned. Console applications may assign a handler to WakeMainThread if the application needs to be notified of a thread synchronization call. If your console application assigns a handler, this is an optimization that only accelerates deliver of notification that a synchronization request is available. You must use CheckSynchronize to perforrm the synchronization itself.

See Also

hooksynchronizewakeup (see page 13682)

WakeMainThread

Synchronize (see page 3126)

72.1.1.4.45 TApplication.UpdateAction Method

Generates an OnActionUpdate (see page 13726) event.

Pascal

```
function UpdateAction(Action: TBasicAction): Boolean; reintroduce;
```

C++

```
__fastcall Boolean UpdateAction(TBasicAction Action);
```

Description

When the application is idle, it makes a series of calls to update the properties (such as whether it is enabled, checked, and so on) of every action that is linked to a visible control or menu item. First, that action list that contains the action generates an OnUpdate event. If the action list does not handle the OnUpdate event, then the action is routed to the Application (see page 14383) object's UpdateAction method, which invokes the OnActionUpdate (see page 13726) event handler. (The application's UpdateAction method responds to all unhandled actions in the application.) If the OnActionUpdate (see page 13726) event handler does not update the action, then it is routed to the action's OnUpdate event handler. If that does not update the action, the active control's UpdateAction method is called, followed (if necessary) by the active form's UpdateAction method.

Do not call UpdateAction. It is called automatically when the application is idle. The Action parameter specifies the action to be updated. UpdateAction returns true if the action's properties are updated and require no further processing. Otherwise, UpdateAction returns false, and the default action processing continues.

See Also

OnActionUpdate (see page 13726)

ExecuteAction (see page 13673)

UpdateTarget (see page 2715)

HandlesTarget (see page 2713)

InitiateAction (see page 5769)

72.1.1.4.46 TApplication.UseRightToLeftAlignment Method

Specifies whether the application object is in a right-to-left alignment mode.

Pascal

```
function UseRightToLeftAlignment: Boolean;
```

C++

```
__fastcall Boolean UseRightToLeftAlignment();
```

Description

Use UseRightToLeftAlignment to determine whether the application’s controls alignment is reversed so that they are aligned from right to left. UseRightToLeftAlignment returns true for Middle Eastern locales if the BiDiMode (see page 13706) property is bdRightToLeft. Otherwise, it returns false.

See Also

BiDiMode

72.1.1.4.47 TApplication.UseRightToLeftReading Method

Specifies whether the application object is in a right-to-left text mode.

Pascal

```
function UseRightToLeftReading: Boolean;
```

C++

```
__fastcall Boolean UseRightToLeftReading();
```

Description

Call UseRightToLeftReading to determine which style and text drawing flags (used in ExTextOut or DrawText) to set for the application's controls.

UseRightToLeftReading returns true for for Middle Eastern locales if the BiDiMode (see page 13706) property is not bdLeftToRight. Otherwise, it returns false.

See Also

BiDiMode

72.1.1.4.48 TApplication.UseRightToLeftScrollBar Method

Specifies whether the vertical scroll bar appears on the left side controls.

Pascal

```
function UseRightToLeftScrollBar: Boolean;
```

C++

```
__fastcall Boolean UseRightToLeftScrollBar();
```

Description

Call UseRightToLeftScrollBar to determine whether the vertical scroll bars appear on the left sides of the application's controls. When UseRightToLeftScrollBar returns true, the vertical scroll bars appear on the left sides of the controls. When UseRightToLeftScrollBar returns false, the vertical scroll bars appear on the right.

UseRightToLeftScrollBar returns true for Middle Eastern locales if BiDiMode (see page 13706) is bdRightToLeft or bdRightToLeftNoAlign.

See Also

BiDiMode

72.1.1.5 TApplication Properties

The properties of the TApplication class are listed here.

Public Properties

	Name	Description
	ActionUpdateDelay (see page 13704)	Delays the call to DoActionIdle.
R	Active (see page 13704)	Specifies whether the application is active and has focus.
R	ActiveFormHandle (see page 13705)	Gives the Windows handle of the currently active form.
	AllowTesting (see page 13705)	Represents information for the IDE.
	AutoDragDocking (see page 13705)	Indicates whether dragged windows are automatically docked.
	BiDiKeyboard (see page 13706)	Specifies the name of the keyboard map that should be used in middle-east locales when text reads from right to left.
	BiDiMode (see page 13706)	Indicates the layout of the application when running under Middle Eastern versions of Windows.
R	CurrentHelpFile (see page 13707)	Indicates the current help file.
	DialogHandle (see page 13707)	Provides a mechanism for using non-VCL dialog boxes in an application.
	EnumAllWindowsOnActivateHint (see page 13708)	This is EnumAllWindowsOnActivateHint, a member of class TApplication.
R	ExeName (see page 13709)	Contains the file name of the application's executable file including path information.
	Handle (see page 13710)	Provides access to the window handle of the main form (window) of the application.
	HelpFile (see page 13711)	Specifies the name of the default file the application uses to display Help.

R	HelpSystem (see page 13711)	Provides access to the interface for working with the help system.
	Hint (see page 13712)	Specifies the text string that appears in the Help Hint box.
	HintColor (see page 13715)	Determines the color of the hint boxes for the Help Hints for the application.
	HintHidePause (see page 13715)	Specifies the time interval to wait before hiding the Help Hint (see page 13712) if the mouse has not moved from the control or menu item.
	HintPause (see page 13715)	Specifies the time interval that passes before the control's Help Hint (see page 13712) appears when the user places the mouse pointer on a control or menu item.
	HintShortCuts (see page 13716)	Specifies whether the application's pop-up hints include shortcut information.
	HintShortPause (see page 13716)	Specifies the time period to wait before bringing up a hint if another hint has already been shown.
	Icon (see page 13717)	Determines which icon represents the application on the Windows task bar.
R	MainForm (see page 13718)	Identifies which form in the application is the main window.
R	MainFormHandle (see page 13719)	Gives the Windows handle of the application main form.
	MainFormOnTaskBar (see page 13719)	Controls (see page 5711) how Windows taskbar buttons are handled by VCL.
R	ModalLevel (see page 13720)	This is ModalLevel, a member of class TApplication.
	ModalPopupMode (see page 13720)	This is ModalPopupMode, a member of class TApplication.
	NonBiDiKeyboard (see page 13720)	Specifies the name of the keyboard map that should be used unless text reads from right to left.
R	PopupControlWnd (see page 13721)	This is PopupControlWnd, a member of class TApplication.
	ShowHint (see page 13721)	Determines whether Help Hints are enabled or disabled for the entire application.
	ShowMainForm (see page 13722)	Determines whether the application shows its main form on startup.
R	Terminated (see page 13723)	Reports whether the application is in the process of shutting down.
	Title (see page 13723)	Contains the text that appears below the icon representing the application when it is minimized.
	UpdateFormatSettings (see page 13724)	Specifies whether format settings are updated automatically when the user alters the system configuration.
	UpdateMetricSettings (see page 13724)	Specifies whether the settings used for the hint window font and the icon title font are updated.

72.1.1.5.1 TApplication.ActionUpdateDelay Property

Delays the call to DoActionIdle.

Pascal

```
property ActionUpdateDelay: Integer;
```

C++

```
__property int ActionUpdateDelay;
```

Description

Use ActionUpdateDelay to delay the call to DoActionIdle. The following example demonstrates the usefulness of this property:

1. Create (see page 13666) an MDI project with a main form, a MDI Child and a TAction class.
2. Link the TAction OnUpdate event to a method.
3. Run (see page 13696) the program and maximize the MDI child.

If you open Windows Task Manager, you can see that CPU usage for the MDI application is nearly 100%. However, if you set Application.ActionUpdateDelay to a non-zero value, the CPU usage for the application is significantly lower than 100%.

See Also

Building a VCL Forms MDI Application Using a Wizard

72.1.1.5.2 TApplication.Active Property

Specifies whether the application is active and has focus.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Active is true while the application is active and false if it is not. Active is set to true in the constructor.

An application is active if the form or application has focus. An application becomes inactive when a window from a different application is about to become activated. When the application closes, Active is set to false in the destructor.

Note: Active is read-only.

See Also

OnActivate (see page 13726)

OnDeactivate (see page 13727)

72.1.1.5.3 TApplication.ActiveFormHandle Property

Gives the Windows handle of the currently active form.

Pascal

```
property ActiveFormHandle: HWND;
```

C++

```
__property HWND ActiveFormHandle;
```

Description

ActiveFormHandle gives the Windows handle of the currently active form.

See Also

MainFormHandle (see page 13719)

72.1.1.5.4 TApplication.AllowTesting Property

Represents information for the IDE.

Pascal

```
property AllowTesting: Boolean;
```

C++

```
__property Boolean AllowTesting;
```

Description

Do not use AllowTesting; it is for internal use only.

72.1.1.5.5 TApplication.AutoDragDocking Property

Indicates whether dragged windows are automatically docked.

Pascal

```
property AutoDragDocking: Boolean;
```

C++

```
__property Boolean AutoDragDocking;
```

Description

Set AutoDragDocking to enable drag-and-dock behavior when windows are dragged. When AutoDragDocking is true, dragged windows are automatically docked when the mouse is released over a docking site. When AutoDragDocking is false, dragged windows are only docked when the Ctrl key is pressed.

72.1.1.5.6 TApplication.BiDiKeyboard Property

Specifies the name of the keyboard map that should be used in middle-east locales when text reads from right to left.

Pascal

```
property BiDiKeyboard: string;
```

C++

```
__property AnsiString BiDiKeyboard;
```

Description

This name is a string composed of the hexadecimal value of the language identifier (low word) and a device identifier (high word). For example, Saudi Arabian Arabic has a language identifier of 0x0401, so its primary layout is named "00000401".

This keyboard layout is only used when the application runs in a middle eastern locale and the value of BiDiMode (see page 13706) indicates that text should read from right to left.

See Also

BiDiMode (see page 13706)

NonBiDiKeyboard (see page 13720)

72.1.1.5.7 TApplication.BiDiMode Property

Indicates the layout of the application when running under Middle Eastern versions of Windows.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to specify the bi-directional mode for the application. The bi-directional mode controls the direction in which text appears (left-to-right or right-to-left), the placement of vertical scroll bars, and the default alignment of text when the application runs in Middle Eastern locales.

The application's BiDiMode is used in the ShowMessage procedure and the MessageBox (see page 13686) method. It also serves as the ParentBiDiMode for forms and for popup menus that are not activated by a control.

See Also

ParentBiDiMode

ShowMessage (see page 11541)

MessageBox (see page 13686)

72.1.1.5.8 TApplication.CurrentHelpFile Property

Indicates the current help file.

Pascal

```
property CurrentHelpFile: string;
```

C++

```
__property AnsiString CurrentHelpFile;
```

Description

Read CurrentHelpFile to determine the name of the help file that is currently used by the application when it executes a help command (via the HelpCommand (see page 13678) method) or brings up context sensitive help (via the HelpContext (see page 13679) or HelpJump (see page 13679) method). If CurrentHelpFile is an empty string, the application ignores attempts to display Help.

If the active form has an associated help file, this is the value of CurrentHelpFile. Otherwise, CurrentHelpFile reflects the value of the application's HelpFile (see page 13711) property.

See Also

PropertyName

ActiveCustomForm (see page 14207)

HelpCommand (see page 13678)

HelpJump (see page 13679)

HelpSystem (see page 13711)

HelpContext (see page 13679)

72.1.1.5.9 TApplication.DialogHandle Property

Provides a mechanism for using non-VCL dialog boxes in an application.

Pascal

```
property DialogHandle: HWnd;
```

C++

```
__property HWnd DialogHandle;
```

Description

Use DialogHandle when displaying a modeless dialog box that was created using the CreateDialog API function. Assigning the handle of the modeless dialog box to DialogHandle allows the dialog to see messages from the application's message loop.

Assign the handle of the modeless dialog box to DialogHandle when it receives an activation message (WM_NCACTIVATE) and set DialogHandle to zero when the dialog box receives a deactivation message.

Delphi Examples:

```
{
Here is replacement of the window procedure for Delphi's
encapsulation of the Windows search-and-replace common dialog
box. Click on Button2 to activate an open dialog and send
a WM_NCACTIVATE to FindReplaceWndProc.
}
function FindReplaceWndProc(Wnd: HWND; Msg, WParam, LParam: Longint): Longint; stdcall;
```

```
    function CallDefWndProc: Longint;
    begin
      Result := CallWindowProc(Pointer(GetProp(Wnd,
        MakeIntAtom(WndProcPtrAtom))), Wnd, Msg, WParam, LParam);
    end;

  begin
    case Msg of
      WM_DESTROY:
        if Application.DialogHandle = Wnd then Application.DialogHandle := 0;
      WM_NCACTIVATE:
        if WParam <> 0 then
        begin
          if Application.DialogHandle = 0 then Application.DialogHandle := Wnd;
        end else
        begin
          if Application.DialogHandle = Wnd then Application.DialogHandle := 0;
        end;
      WM_NCDESTROY:
        begin
          Result := CallDefWndProc;
          RemoveProp(Wnd, MakeIntAtom(WndProcPtrAtom));
          Exit;
        end;
     end;
     Result := CallDefWndProc;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    ReplaceDialog1.Execute;
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    OpenDialog1.Execute;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    AtomText: array[0..31] of Char;
  begin
    WndProcPtrAtom := GlobalAddAtom(StrFmt(AtomText,
      'WndProcPtr%.8X%.8X', [HInstance, GetCurrentThreadID]));
    SetWindowLong(Application.Handle, GWL_WNDPROC, Longint(@FindReplaceWndProc));;
  end;
```

See Also

Handle (see page 13710)

72.1.1.5.10 TApplication.EnumAllWindowsOnActivateHint Property

Pascal

```
property EnumAllWindowsOnActivateHint: Boolean;
```

C++

```
__property Boolean EnumAllWindowsOnActivateHint;
```

Description

This is EnumAllWindowsOnActivateHint, a member of class TApplication.

72.1.1.5.11 TApplication.ExeName Property

Contains the file name of the application's executable file including path information.

Pascal

```
property ExeName: string;
```

C++

```
__property AnsiString ExeName;
```

Description

Use ExeName to obtain the name of the executable file for the application. ExeName is the fully-qualified name, including the path to the application's executable.

The name of the application is the root name of the project with an .EXE extension. By default, this name is PROJECT1.EXE. To change the ExeName, save the project by the desired new root name and rebuild the application. ExeName will reflect the change to the project file name.

ExeName is a read-only property.

C++ Examples:

```
/*
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Files
can be saved to and from the app directory by using relative
paths.
*/
void __fastcall TForm1::Save1Click(TObject *Sender)
{
  AnsiString NewFileName =
    ExtractFilePath(Application->ExeName) +
    ExtractFileName(Edit1->Text);
  AnsiString OldFileName =
    ExtractFilePath(Application->ExeName) +
    ExtractFileName(Edit2->Text);
  AnsiString Msg = Format("Copy %s to %s", ARRAYOFCONST((OldFileName, NewFileName)));
  if (MessageDlg(Msg, mtCustom, TMsgDlgButtons() << mbOK << mbCancel, 0) == mrOk)
  {
    TFileStream *OldFile = new TFileStream(OldFileName, fmOpenRead);
    try
    {
      TFileStream *NewFile = new TFileStream(NewFileName, fmCreate);
      try
      {
        NewFile->CopyFrom(OldFile, OldFile->Size);
      }
      __finally
      {
        FreeAndNil(&NewFile);
      }
    }
    __finally
    {
      FreeAndNil(&OldFile);
    }
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Label3->Caption = "Current directory: " + ExtractFilePath(Application->ExeName);
```

```
}
```

Delphi Examples:

```
{
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Files
can be saved to and from the app directory by using relative
paths.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Label3.Caption:= 'Current directory: ' + ExtractFilePath(Application.ExeName);
end;

procedure TForm1.Save1Click(Sender: TObject);
var
  NewFileName, OldFileName: string;
  Msg: string;
  NewFile: TFileStream;
  OldFile: TFileStream;
begin
  NewFileName :=
    ExtractFilePath(Application.ExeName) +
    ExtractFileName(Edit1.Text);
  OldFileName :=
    ExtractFilePath(Application.ExeName) +
    ExtractFileName(Edit2.Text);
  Msg := Format('Copy %s to %s?', [OldFileName, NewFileName]);
  if MessageDlg(Msg, mtCustom, mbOKCancel, 0) = mrOK then
  begin
    OldFile := TFileStream.Create(
      OldFileName, fmOpenRead or fmShareDenyWrite);
    try
      NewFile := TFileStream.Create(
        NewFileName, fmCreate or fmShareDenyRead);
      try
        NewFile.CopyFrom(OldFile, OldFile.Size);
      finally
        FreeAndNil(NewFile);
      end;
    finally
      FreeAndNil(OldFile);
    end;
  end;
end;
```

See Also

ParamStr (see page 22496)

Title (see page 13723)

72.1.1.5.12 TApplication.Handle Property

Provides access to the window handle of the main form (window) of the application.

Pascal

```
property Handle: HWnd;
```

C++

```
__property HWnd Handle;
```

Description

Use Handle when calling Windows API functions that require a parent window handle. For example, a DLL that displays its own top-level pop-up windows needs a parent window to display its windows in the application. Using the Handle property makes such windows part of the application, so that they are minimized, restored, enabled, and disabled with the application.

Note: When writing a DLL that uses VCL forms, assign the window handle of the host EXE's main window to the DLL's Application.HandleApplication->Handle property. This makes the DLL's form part of the host application. Never assign to the Handle property in an EXE.

See Also

DialogHandle (see page 13707)

MainForm (see page 13718)

72.1.1.5.13 TApplication.HelpFile Property

Specifies the name of the default file the application uses to display Help.

Pascal

```
property HelpFile: string;
```

C++

```
__property AnsiString HelpFile;
```

Description

Use HelpFile to specify a default help file for applications that use the Help system. Either assign a file name to the HelpFile property at runtime, or specify a Help File on the Application (see page 14383) page of the Project|Options dialog box at design time.

The Help system specified by HelpSystem (see page 13711) displays help topics from the file specified by CurrentHelpFile (see page 13707). HelpFile supplies the value of CurrentHelpFile (see page 13707) unless there is a help file associated with the active form.

By default, HelpFile is a null string, and the application's Help method ignores attempts to display Help unless the active form has an associated help file.

See Also

HelpCommand (see page 13678)

HelpContext (see page 13679)

HelpJump (see page 13679)

CurrentHelpFile (see page 13707)

HelpSystem (see page 13711)

72.1.1.5.14 TApplication.HelpSystem Property

Provides access to the interface for working with the help system.

Pascal

```
property HelpSystem: IHelpSystem;
```

C++

```
__property IHelpSystem HelpSystem;
```

Description

TApplication (see page 13660) uses IHelpSystem to interact with a Help system manager. This interface includes methods for displaying help topics based on a keyword or topic ID.

Applications can use the HelpSystem property to interact directly with the Help system manager, or they can use the application's help-oriented methods such as HelpCommand (see page 13678) and HelpJump (see page 13679).

See Also

HelpCommand (see page 13678)

HelpContext (see page 13679)

HelpJump (see page 13679)

CurrentHelpFile (see page 13707)

HelpFile (see page 13711)

ShowTableOfContents (see page 15027)

72.1.1.5.15 TApplication.Hint Property

Specifies the text string that appears in the Help Hint box.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

TApplication (see page 13660)'s Hint property is assigned the Hint value of the control or menu item that the mouse is currently moving over. It can also be assigned a string value that informs the user of some action, error, or other type of information. Therefore, use Hint either to:

Transfer hints from controls to another area of display, such as a status bar, using the OnHint (see page 13731) event handler. This reads the Hint property.

Indicate the status of the application while it is processing some action. This sets the Hint property.

Help Hints appear when the OnHint (see page 13731) event occurs. Therefore, if TApplication (see page 13660)'s Hint property is assigned to the status bar's caption, for example, the caption of the status bar control displays the current string value of the Hint property during an OnHint (see page 13731) event.

There are two parts to the Hint string: short and long. Short hints are used by pop-up tool tips. Long hints are used by the status bar, separated by the | character. Use the global functions GetShortHint and GetLongHint, from the Controls (see page 5711) unit to extract the long and short hints from a hint string.

Note: When setting Hint to a message informing the user of an event occurring in the application, remember that, by default, the Hint string is reset to a control's Hint when the mouse cursor moves over a control.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
```

```
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
/*
Note:   It is not actually necessary to write an OnHint
event handler to display hints on the status bar. This can
be accomplished more simply by setting the status bar's
AutoHint property and SimplePanel property.
This shows how the OnHint event handler is declared as a
public method of the form.
*/

/*
Here is the code to add to the unit file.  It provides the
implementation of the DisplayHint method and assigns it to
the Application in the form's OnCreate event handler.
Note:   It is not actually necessary to write an OnHint event
handler to display hints on the status bar. This can be
accomplished more simply by setting the status bar's AutoHint
property.
*/
void __fastcall TForm1::DisplayHint(TObject *Sender)
{
  StatusBar1->SimpleText = GetLongHint(Application->Hint);
}

// Here is the form's OnCreate event handler.
// It assigns the application's OnHint event handler at
// runtime because the Application is not available in the
// Object Inspector
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnHint = DisplayHint;
}
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
```

```
    I: Integer;
  begin
    Edit1.Hint := 'Enter your name';
    Edit1.ShowHint := True;
    with ListBox1 do
    begin
      for I := 1 to 10 do
        Items.Add('Item ' + IntToStr(I));
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
  {
  Note:   It is not actually necessary to write an OnHint
  event handler to display hints on the status bar. This can
  be accomplished more simply by setting the status bar's
  AutoHint property and SimplePanel property.
  This shows how the OnHint event handler is declared as a
  public method of the form.
  }
  type
    TForm1 = class(TForm)
      Button1: TButton;
      StatusBar1: TStatusBar;
      Edit1: TEdit;
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      procedure DisplayHint(Sender: TObject);
    end;
  var
    Form1: TForm1;
  implementation
  {$R *.dfm}

  { Here is the implementation of the OnHint event handler }
  { It displays the application's current hint in the status bar }
  procedure TForm1.DisplayHint(Sender: TObject);
  begin
    StatusBar1.SimpleText := GetLongHint(Application.Hint);
  end;
  { Here is the form's OnCreate event handler. }
  { It assign's the application's OnHint event handler at runtime }
  { because the Application is not available in the Object Inspector }
  { at design time }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Application.OnHint := DisplayHint;
  end;
```

See Also

GetLongHint (see page 6252)

GetShortHint (see page 6253)

HintColor (see page 13715)

OnHint (see page 13731)

ShowHint (see page 13721)

THintWindow (see page 6055)

72.1.1.5.16 TApplication.HintColor Property

Determines the color of the hint boxes for the Help Hints for the application.

Pascal

```
property HintColor: TColor;
```

C++

```
__property TColor HintColor;
```

Description

Use HintColor to specify the hint box color. A default color value of clInfoBk is set for the HintColor property in the constructor when the application is created. To change the HintColor, assign it a new TColor value at runtime.

Tip: To further customize the hint window, create a custom descendant of THintWindow and assign it to the global HintWindowClass (see page 14384) variable.

See Also

Hint (see page 13712)

THintWindow (see page 6055)

HintWindowClass (see page 14384)

72.1.1.5.17 TApplication.HintHidePause Property

Specifies the time interval to wait before hiding the Help Hint (see page 13712) if the mouse has not moved from the control or menu item.

Pascal

```
property HintHidePause: Integer;
```

C++

```
__property int HintHidePause;
```

Description

Use HintHidePause to specify a wait time in milliseconds, which is different from the default value of 2500 ms or 2 1/2 seconds that is set in the constructor. The Help Hint (see page 13712) for the control or menu item is specified in the Hint (see page 13712) property.

Note: The default is a predefined constant value that is several times longer than HintPause (see page 13715).

See Also

Hint (see page 13712)

HintPause (see page 13715)

HintShortPause (see page 13716)

72.1.1.5.18 TApplication.HintPause Property

Specifies the time interval that passes before the control's Help Hint (see page 13712) appears when the user places the

mouse pointer on a control or menu item.

Pascal

```
property HintPause: Integer;
```

C++

```
__property int HintPause;
```

Description

Use HintPause to change the default pause time of 500 ms or 1/2 second that is set in the constructor. When assigning a value to HintPause, specify the interval in milliseconds. The Help Hint (see page 13712) displayed at the end of the pause is specified in the Hint (see page 13712) property.

See Also

HintHidePause (see page 13715)

Hint (see page 13712)

HintShortPause (see page 13716)

72.1.1.5.19 TApplication.HintShortCuts Property

Specifies whether the application's pop-up hints include shortcut information.

Pascal

```
property HintShortCuts: Boolean;
```

C++

```
__property Boolean HintShortCuts;
```

Description

Use HintShortCuts to specify whether the application's pop-up hints include shortcut information. If HintShortCuts is true, the shortcut information is displayed. For example, if HintShortCuts is true, the hint for the Cut command might be "Cut (Ctrl+X)" instead of "Cut".

72.1.1.5.20 TApplication.HintShortPause Property

Specifies the time period to wait before bringing up a hint if another hint has already been shown.

Pascal

```
property HintShortPause: Integer;
```

C++

```
__property int HintShortPause;
```

Description

Use HintShortPause to reduce the flicker when moving the mouse quickly over a set of buttons that all have help hints on. Specify a value in milliseconds that is different from the default value of 50 ms that is set in the constructor.

See Also

HintHidePause (see page 13715)

Hint (see page 13712)

HintPause (see page 13715)

72.1.1.5.21 TApplication.Icon Property

Determines which icon represents the application on the Windows task bar.

Pascal

```
property Icon: TIcon;
```

C++

```
__property TIcon Icon;
```

Description

Use Icon to assign a specific icon to represent the application. Set the Icon property by either:

Assigning it a TIcon at runtime.

Loading an icon on the Application (see page 14383) page of the Project|Options dialog box at design time.

Icon represents the application when it is minimized and appears either on the Windows desktop or task bar.

C++ Examples:

```
/*
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
then minimize the application to see the icon.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenPictureDialog1->DefaultExt = GraphicExtension(__classid(TIcon));
  OpenPictureDialog1->FileName = GraphicFileMask(__classid(TIcon));
  OpenPictureDialog1->Filter = GraphicFilter(__classid(TIcon));
  OpenPictureDialog1->Options.Clear();
  OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
  while (true)
  {
    if (OpenPictureDialog1->Execute())
    {
      if (!OpenPictureDialog1->Options.Contains(ofExtensionDifferent))
      {
        Application->Icon->LoadFromFile(OpenPictureDialog1->FileName);
        break;
      }
      else // reset Options to remove ofExtensionDifferent
      {
        OpenPictureDialog1->Options.Clear();
        OpenPictureDialog1->Options << ofFileMustExist << ofHideReadOnly << ofNoChangeDir;
      }
    }
    else // user cancelled
      break;
  }
}
```

Delphi Examples:

```
{
The following code allows the user to use a dialog box to
redefine the icon for the application at run time. When the
user clicks Button1, OpenDialog1 executes and the user
specifies an icon filename. The Icon is then assigned to the
application.  At runtime, click Button1 to select the icon,
```

```
then minimize the application to see the icon.
}
procedure TForm1.Button1Click(Sender: TObject);
var
Done: Boolean;
filenamestring : String;
begin
  OpenPictureDialog1.DefaultExt := GraphicExtension(TIcon);
  filenamestring := GraphicFileMask(TIcon);
  OpenPictureDialog1.FileName := filenamestring;
  OpenPictureDialog1.Filter := GraphicFilter(TIcon);
  OpenPictureDialog1.Options := [ ofFileMustExist, ofHideReadOnly, ofNoChangeDir ];
  Done := False;
  while not Done do
  begin
  if OpenPictureDialog1.Execute then
    begin
    if not (ofExtensionDifferent in OpenPictureDialog1.Options) then
      begin
      Application.Icon.LoadFromFile(OpenPictureDialog1.FileName);
      Done := True;
      end
    else
//      OpenPictureDialog1.Options := OpenPictureDialog1.Options - ofExtensionDifferent;
    end
  else { User cancelled }
    Done := True;
  end;
end;
```

See Also

Minimize (see page 13688)

OnMinimize (see page 13739)

Restore (see page 13694)

Title (see page 13723)

72.1.1.5.22 TApplication.MainForm Property

Identifies which form in the application is the main window.

Pascal

```
property MainForm: TForm;
```

C++

```
__property TForm * MainForm;
```

Description

Use MainForm to determine the form that acts as the application's main window. The main form is the first form created in the main body of the application. When the main form closes, the application terminates.

When a new project is created, Form1 automatically becomes the value of the MainForm property. To assign a different form to the MainForm property, use the Forms (see page 13657) page of the Project|Options dialog box at design time. MainForm cannot be modified at runtime; it is read-only.

See Also

Create (see page 13666)

CreateForm (see page 13670)

Handle (see page 13710)

Terminate (see page 13699)

72.1.1.5.23 TApplication.MainFormHandle Property

Gives the Windows handle of the application main form.

Pascal

```
property MainFormHandle: HWND;
```

C++

```
__property HWND MainFormHandle;
```

Description

MainFormHandle gives the Windows handle of the application main form.

See Also

ActiveFormHandle (see page 13705)

72.1.1.5.24 TApplication.MainFormOnTaskBar Property

Controls (see page 5711) how Windows taskbar buttons are handled by VCL.

Pascal

```
property MainFormOnTaskBar: Boolean;
```

C++

```
__property Boolean MainFormOnTaskBar;
```

Description

The MainFormOnTaskBar property controls how Windows taskbar buttons are handled by VCL.

If the property is true, a taskbar button represents the application's main form and displays its caption. If false, a taskbar button represents the application's (hidden) main window and bears the application's Title (see page 13723).

MainFormOnTaskBar must be true to use Windows Vista Aero effects. These include live taskbar thumbnails, Dynamic Windows, Windows Flip, and Windows Flip 3D.

If a change to default behavior is required, MainFormOnTaskBar should be set in the .dpr file after Application.Initialize (see page 13682) and before main form creation. The MainFormOnTaskBar setting is intended to persist for the life of the application; later runtime changes of this property could result in unexpected behavior.

MainFormOnTaskBar defaults to `True` for applications created in Delphi 2007 and later products and `False` for earlier products.

The property can be applied to older applications. Note that it affects the Z-order of your MainForm (see page 13718) in case your application depends on this.

To update existing VCL applications, add the following line to the project's .dpr file after `Application.Initialize (see page 13682);`:

```
Application.MainFormOnTaskbar := True;
```

This line is automatically added to new projects.

For further information, see http://www.microsoft.com/windows/products/windowsvista/features/experiences/aero.mspx.

See Also

TApplication (see page 13660)

72.1.1.5.25 TApplication.ModalLevel Property

Pascal

```
property ModalLevel: Integer;
```

C++

```
__property int ModalLevel;
```

Description

This is ModalLevel, a member of class TApplication.

72.1.1.5.26 TApplication.ModalPopupMode Property

Pascal

```
property ModalPopupMode: TPopupMode;
```

C++

```
__property TPopupMode ModalPopupMode;
```

Description

This is ModalPopupMode, a member of class TApplication.

72.1.1.5.27 TApplication.NonBiDiKeyboard Property

Specifies the name of the keyboard map that should be used unless text reads from right to left.

Pascal

```
property NonBiDiKeyboard: string;
```

C++

```
__property AnsiString NonBiDiKeyboard;
```

Description

Use NonBiDiKeyboard to specify the keyboard map that should be used. This name is a string composed of the hexadecimal value of the language identifier (low word) and a device identifier (high word). For example, U.S. English has a language identifier of 0x0409, so the primary U.S. English layout is named "00000409". Variants (see page 23567) of U.S. English layout (such as the Dvorak layout) are named "00010409", "00020409", and so on.

If the application is running in a middle eastern locale and the value of BiDiMode (see page 13706) indicates that text should read from right to left, the application uses BiDiKeyboard (see page 13706) instead.

See Also

BiDiMode (see page 13706)

BiDiKeyboard (see page 13706)

72.1.1.5.28 TApplication.PopupControlWnd Property

Pascal

```
property PopupControlWnd: HWND;
```

C++

```
__property HWND PopupControlWnd;
```

Description

This is PopupControlWnd, a member of class TApplication.

72.1.1.5.29 TApplication.ShowHint Property

Determines whether Help Hints are enabled or disabled for the entire application.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

Use ShowHints to choose when to display Help Hints. If ShowHint is true, Help Hints are enabled; if ShowHint is false, Help Hints are disabled. The default value is true. Help Hints are specified in the Hint (see page 13712) property.

Setting ShowHint for the application to false disables all Help Hints, regardless of the value of the ShowHint properties for individual controls.

Tip: You can customize the hint window by creating a custom descendant of THintWindow and assigning it to the global HintWindowClass (see page 14384) variable.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
```

```
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 13712)

OnHint (see page 13731)

ParentShowHint (see page 5835)

ShowHint (see page 5840)

THintWindow (see page 6055)

72.1.1.5.30 TApplication.ShowMainForm Property

Determines whether the application shows its main form on startup.

Pascal

```
property ShowMainForm: Boolean;
```

C++

```
__property Boolean ShowMainForm;
```

Description

Use ShowMainForm to control whether and when the application shows its main form.

The default value of true is set in the constructor. When ShowMainForm is true, the main window shows automatically when the application starts. The MainForm (see page 13718) property specifies which form is the main form.

To hide the main form at startup, set ShowMainForm to false in the main project file before the call to the Run (see page

13696) method. Make sure that the Visible property of the form is also false.

See Also

Create (see page 13666)

MainForm (see page 13718)

Run (see page 13696)

Visible (see page 5847)

Hiding the Main Form

72.1.1.5.31 TApplication.Terminated Property

Reports whether the application is in the process of shutting down.

Pascal

```
property Terminated: Boolean;
```

C++

```
__property Boolean Terminated;
```

Description

Terminated is used when calling the ProcessMessages (see page 13691) method so that the application doesn't perform unnecessary actions after the shut-down process has been triggered. ProcessMessages (see page 13691) sets Terminated to true when it receives a WM_QUIT message.

An application usually receives a WM_QUIT message because the main window of the application has closed, or because the Terminate (see page 13699) method has been called.

Note: For applications using calculation-intensive loops, call ProcessMessages (see page 13691) periodically, and also check Terminated to determine whether to abort the calculation and allow the application to terminate.

See Also

ProcessMessages (see page 13691)

Terminate (see page 13699)

72.1.1.5.32 TApplication.Title Property

Contains the text that appears below the icon representing the application when it is minimized.

Pascal

```
property Title: string;
```

C++

```
__property AnsiString Title;
```

Description

Use the Title property to modify the name of the text string appearing below the icon of the application. The default value is taken from the DLL or EXE file name at runtime.

Set the Title at runtime, or enter the value of the Title property on the Application (see page 14383) page of the Project | Options dialog box.

See Also

ExeName (see page 13709)

Icon (see page 13717)

72.1.1.5.33 TApplication.UpdateFormatSettings Property

Specifies whether format settings are updated automatically when the user alters the system configuration.

Pascal

```
property UpdateFormatSettings: Boolean;
```

C++

```
__property Boolean UpdateFormatSettings;
```

Description

Use UpdateFormatSettings to control automatic updating of format settings. The default of true is set in the constructor. UpdateFormatSettings is checked when the application receives a WM_WININICHANGE message.

Using the default format settings is recommended. These settings are initialized to the Windows local settings.

See Also

variables

72.1.1.5.34 TApplication.UpdateMetricSettings Property

Specifies whether the settings used for the hint window font and the icon title font are updated.

Pascal

```
property UpdateMetricSettings: Boolean;
```

C++

```
__property Boolean UpdateMetricSettings;
```

Description

Use UpdateMetricSettings to specify whether or not the hint window font and the icon title font on the desk top should be updated to reflect the system settings when those settings are changed.

The initial value of true is set in the constructor.

72.1.1.6 TApplication Events

The events of the TApplication class are listed here.

Public Events

	Name	Description
	OnActionExecute (see page 13725)	Occurs when an action's Execute method is called and its action list has not already handled it.
	OnActionUpdate (see page 13726)	Occurs when an action's Update method is called and its action list has not already handled it.
	OnActivate (see page 13726)	Occurs when an application becomes active.
	OnDeactivate (see page 13727)	Occurs when an application becomes inactive.
	OnException (see page 13728)	Occurs when an unhandled exception occurs in the application.

	OnGetActiveFormHandle (see page 13729)	This is OnGetActiveFormHandle, a member of class TApplication.
	OnGetMainFormHandle (see page 13729)	This is OnGetMainFormHandle, a member of class TApplication.
	OnHelp (see page 13730)	Occurs when the application receives a request for help.
	OnHint (see page 13731)	Occurs when the mouse pointer moves over a control or menu item that can display a Help Hint (see page 13712).
	OnIdle (see page 13733)	Occurs when an application becomes idle.
	OnMessage (see page 13737)	Occurs when the application receives a Windows message.
	OnMinimize (see page 13739)	Occurs when an application is minimized.
	OnModalBegin (see page 13740)	Occurs when a modal form is opened.
	OnModalEnd (see page 13740)	Occurs when a modal form is closed.
	OnRestore (see page 13741)	Occurs when the previously minimized application is restored to its normal size.
	OnSettingChange (see page 13741)	Occurs when Windows notifies the application that a system-wide setting has changed.
	OnShortCut (see page 13741)	Occurs when the user presses a key (before the OnKeyDown event).
	OnShowHint (see page 13742)	Occurs when the application is about to display the hint window for a Help Hint (see page 13712).

72.1.1.6.1 TApplication.OnActionExecute Event

Occurs when an action's Execute method is called and its action list has not already handled it.

Pascal

```
property OnActionExecute: TActionEvent;
```

C++

```
__property TActionEvent OnActionExecute;
```

Description

Use the OnExecute event handler to respond when a user invokes actions whose action lists do not have OnExecute event handlers.

If the action list that contains the action does not handle it in an OnExecute event handler, then the action is routed to the Application (see page 14383) object's ExecuteAction (see page 13673) method, which invokes the OnActionExecute event handler. (The application's ExecuteAction (see page 13673) method applies to all actions in the application.)

The Handled parameter of the event handler returns false by default. If the handler handles the event, it should change Handled to true, thereby preventing further attempts to handle the action. When the event handler exits with Handled set to false, the action's OnExecute event occurs. If the action remains unhandled after that, the active control's ExecuteAction (see page 13673) method is called to allow the action to execute with an identified target. Finally, the active form's ExecuteAction (see page 13673) method is called if all other handlers do not handle the action.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

See Also

OnActionUpdate (see page 13726)

ExecuteAction (see page 13673)

OnExecute (see page 2716)

OnExecute (see page 327)

ExecuteAction

OnActionExecute (see page 1352)

What Happens When an Action Fires

72.1.1.6.2 TApplication.OnActionUpdate Event

Occurs when an action's Update method is called and its action list has not already handled it.

Pascal

```
property OnActionUpdate: TActionEvent;
```

C++

```
__property TActionEvent OnActionUpdate;
```

Description

Use the OnActionUpdate event handler to update the properties of an action when the application is idle if the action's action list does not handle it in an OnUpdate event handler.

If the action list that contains the action does not update it in an OnUpdate event handler, the action is routed to the Application (see page 14383) object's UpdateAction (see page 13701) method, which invokes the OnActionUpdate event handler. (The application's UpdateAction (see page 13701) method applies to all actions in the application.)

The Handled parameter of the event handler returns false by default. If the handler updates the event, it should change Handled to true, thereby ending the processing of the action. When the event handler exits with Handled set to false, the action's OnUpdate event occurs. If the action is not updated after that, the active control's UpdateAction (see page 13701) method is called to allow the target to update the action. Finally, the active form's UpdateAction (see page 13701) method is called if all other handlers do not handle the action.

Note: OnActionUpdate occurs after the application's OnIdle (see page 13733) event.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

See Also

OnActionExecute (see page 13725)

UpdateAction (see page 13701)

OnUpdate (see page 2718)

OnUpdate (see page 327)

UpdateAction

OnIdle (see page 13733)

OnActionUpdate (see page 1354)

72.1.1.6.3 TApplication.OnActivate Event

Occurs when an application becomes active.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to perform special processing when the application becomes active.

An application becomes active when it is initially run or when focus moves from another Windows application back to any window of the application.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

See Also

Active (see page 13704)

OnDeactivate (see page 13727)

OnActivate (see page 1355)

72.1.1.6.4 TApplication.OnDeactivate Event

Occurs when an application becomes inactive.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to perform any special processing that should occur immediately before the application is deactivated. The OnDeactivate event occurs when the user switches from the application to another Windows application.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

C++ Examples:

```
/*
The following code minimizes an application when it's
deactivated. Note that AppDeactivate should be declared a
method of TForm1.  Click anywhere outside of the form
to deactivate it.
*/
void __fastcall TForm1::AppDeactivate(TObject *Sender)
{
  Application->Minimize();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnDeactivate = AppDeactivate;
}
```

Delphi Examples:

```
{
The following code minimizes an application when it's
deactivated. Note that AppDeactivate should be declared a
method of TForm1.  Click anywhere outside of the form
to deactivate it.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnDeactivate := AppDeactivate;
end;
```

```
procedure TForm1.AppDeactivate(Sender: TObject);
begin
  Application.Minimize;
end;
```

See Also

Active (see page 13704)

OnActivate (see page 13726)

OnDeactivate (see page 1356)

72.1.1.6.5 TApplication.OnException Event

Occurs when an unhandled exception occurs in the application.

Pascal

```
property OnException: TExceptionEvent;
```

C++

```
__property TExceptionEvent OnException;
```

Description

Use OnException to change the default behavior that occurs when an exception is not handled by application code. The OnException event handler is called automatically in the HandleException (see page 13674) method.

OnException only handles exceptions that occur during message processing. Exceptions that occur before or after the execution of the application's Run (see page 13696) method do not generate OnException events.

If an exception passes through the tryblocks in the application code, the application automatically calls the HandleException (see page 13674) method. Unless the exception object is EAbort, HandleException (see page 13674) calls the OnException handler, if one exists. Otherwise, it calls ShowException (see page 13698) to display a message box indicating an error occurred.

TExceptionEvent (see page 14373) is the type of the OnException event. It points to a method that handles exceptions in the application. The Sender parameter is the object that raised the exception, and E is the exception object.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

C++ Examples:

```
/*
In addition to displaying the exception message, which
happens by default, the following code shuts down the
application when an exception is not caught and handled.
AppException should be declared a method of TForm1.
*/

#include <System.hpp>
#include <SysUtils.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnException = AppException;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::AppException(TObject *Sender, Exception *E)
{
```

```
  Application->ShowException(E);
  Application->Terminate();
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  throw(Exception("Hardware error: Divide by 0"));
}
```

Delphi Examples:

```
{
In addition to displaying the exception message, which
happens by default, the following code shuts down the
application when an exception is not caught and handled.
AppException should be declared a method of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnException := AppException;
end;
procedure TForm1.AppException(Sender: TObject; E: Exception);
begin
  Application.ShowException(E);
  Application.Terminate;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  raise EPasswordInvalid.Create('Incorrect password entered');
end;
```

See Also

HandleException (see page 13674)

ShowException (see page 13698)

OnException (see page 1357)

72.1.1.6.6 TApplication.OnGetActiveFormHandle Event

Pascal

```
property OnGetActiveFormHandle: TGetHandleEvent;
```

C++

```
__property TGetHandleEvent OnGetActiveFormHandle;
```

Description

This is OnGetActiveFormHandle, a member of class TApplication.

72.1.1.6.7 TApplication.OnGetMainFormHandle Event

Pascal

```
property OnGetMainFormHandle: TGetHandleEvent;
```

C++

```
__property TGetHandleEvent OnGetMainFormHandle;
```

Description

This is OnGetMainFormHandle, a member of class TApplication.

72.1.1.6.8 TApplication.OnHelp Event

Occurs when the application receives a request for help.

Pascal

```
property OnHelp: THelpEvent;
```

C++

```
__property THelpEvent OnHelp;
```

Description

Write an OnHelp event handler to perform special processing when the user requests help. The HelpContext (see page 13679) and the HelpJump (see page 13679) methods automatically trigger the OnHelp event.

Set CallHelp to true if the application should still invoke the help system after the event. Set CallHelp to false to prevent the default response by the help system. All application help methods go through OnHelp. The application calls the help system only if OnHelp's CallHelp parameter returns true or if no OnHelp event handler is assigned.

The event handler returns true if it succeeds, false if it fails.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

C++ Examples:

```
/*
The following code changes the Help file depending on the
active form, before the help viewer is called. AppHelp is
assigned to the OnHelp event handler of Application in the
OnCreate event of Form1.
*/
bool __fastcall TForm1::AppHelp(Word Command, int Data, bool &CallHelp)
{
  Application->HelpFile = Screen->ActiveForm->Name + ".hlp";
  CallHelp = true;
  return true;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnHelp = AppHelp;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  const DATANOTFOUND = 714;
  Application->HelpContext(DATANOTFOUND);
}
```

Delphi Examples:

```
{
The following code changes the Help file depending on the
active form, before the help viewer is called. AppHelp is
assigned to the OnHelp event handler of Application in the
OnCreate event of Form1.
}
function TForm1.AppHelp(
  Command: Word; Data: Longint; var CallHelp: Boolean) : Boolean;
var helpstring : String;
begin
```

```
    helpstring := Screen.ActiveForm.Name + '.hlp';
    MessageDlg(
      'Using ' + helpstring + ' as the help file.',
      mtInformation, [mbOK], 0);
    Application.HelpFile := helpstring;
    CallHelp := True;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  const DATANOTFOUND = 714;
  begin
    Application.HelpContext(DATANOTFOUND);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Application.OnHelp := AppHelp;
  end;
```

See Also

HelpCommand (see page 13678)

HelpContext (see page 13679)

HelpFile (see page 13711)

HelpJump (see page 13679)

OnHelp (see page 1358)

InvokeHelp

72.1.1.6.9 TApplication.OnHint Event

Occurs when the mouse pointer moves over a control or menu item that can display a Help Hint (see page 13712).

Pascal

```
property OnHint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnHint;
```

Description

Write an OnHint event handler to perform special processing when the mouse pauses over a control or menu item whose Hint (see page 13712) property is not an empty string.

A common use of the OnHint event is to display the value of a control or menu item's Hint (see page 13712) property in another control. Note that it is not necessary to use an OnHint event handler to display hints on a status bar (TStatusBar): TStatusBar will automatically display hints if you set its AutoHint property to true.

The Hint (see page 13712) property of a control can specify both a short Help Hint (see page 13712) and a (usually) longer hint that appears elsewhere because of code in an OnHint event handler.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

Tip: Component writers can respond to the automatically-fired THintAction action in a component's ExecuteAction (see page 13673) method rather than relying on an OnHint event handler.

C++ Examples:

```
/*
Note:   It is not actually necessary to write an OnHint
event handler to display hints on the status bar. This can
be accomplished more simply by setting the status bar's
AutoHint property and SimplePanel property.
This shows how the OnHint event handler is declared as a
public method of the form.
*/

/*
Here is the code to add to the unit file.  It provides the
implementation of the DisplayHint method and assigns it to
the Application in the form's OnCreate event handler.
Note:   It is not actually necessary to write an OnHint event
handler to display hints on the status bar. This can be
accomplished more simply by setting the status bar's AutoHint
property.
*/
void __fastcall TForm1::DisplayHint(TObject *Sender)
{
  StatusBar1->SimpleText = GetLongHint(Application->Hint);
}

// Here is the form's OnCreate event handler.
// It assigns the application's OnHint event handler at
// runtime because the Application is not available in the
// Object Inspector
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnHint = DisplayHint;
}
```

Delphi Examples:

```
{
Note:   It is not actually necessary to write an OnHint
event handler to display hints on the status bar. This can
be accomplished more simply by setting the status bar's
AutoHint property and SimplePanel property.
This shows how the OnHint event handler is declared as a
public method of the form.
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    StatusBar1: TStatusBar;
    Edit1: TEdit;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure DisplayHint(Sender: TObject);
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}

{ Here is the implementation of the OnHint event handler }
{ It displays the application's current hint in the status bar }
procedure TForm1.DisplayHint(Sender: TObject);
begin
  StatusBar1.SimpleText := GetLongHint(Application.Hint);
end;
{ Here is the form's OnCreate event handler. }
```

```
{ It assign's the application's OnHint event handler at runtime }
{ because the Application is not available in the Object Inspector }
{ at design time }
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnHint := DisplayHint;
end;
```

See Also

GetLongHint (see page 6252)

GetShortHint (see page 6253)

Hint (see page 13712)

ActivateHint (see page 13668)

OnHint (see page 1359)

AutoHint (see page 4786)

ExecuteAction

THintAction (see page 20707)

72.1.1.6.10 TApplication.OnIdle Event

Occurs when an application becomes idle.

Pascal

```
property OnIdle: TIdleEvent;
```

C++

```
__property TIdleEvent OnIdle;
```

Description

Write an OnIdle event handler to perform special processing when an application is idle. An application is idle when it is not processing code. For example, an application is idle when it is waiting for input from the user.

OnIdle is called only once, as the application transitions into an idle state. It is not called continuously unless the Done parameter is set to false. Applications that set Done to false consume an inordinate amount of CPU time, which affects overall system performance.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

C++ Examples:

```
/*
The following code demonstrates background processing using
the OnIdle event, and using the HandleMessage method to
permit messages or background processing to get through.
Note: You must add MyIdleHandler to the Form1 methods.
*/
// global variables to show the order of events
int  XPos, YPos, Delta;

// This is a utility procedure to display messages
// add this at the beginning of the unit

void StatusMsg(TForm *Form, TCanvas *Canvas, char *Message, bool Bkg)
```

```
{
  if (!Bkg)
    Canvas->Font->Style = TFontStyles() << fsBold; // foreground messages are bold
  Canvas->TextOut(XPos, YPos, Message);
  if (!Bkg)
    Canvas->Font->Style = TFontStyles();
  // change Xpos and YPos to prepare for the next message
  YPos += Delta;
  if (YPos >= Form->ClientHeight - 10)
  {
    YPos = 10;
    XPos += 180;
  }
  if (XPos >= Form->ClientWidth - 100)
  {
    if (Canvas->Font->Color == clRed)
      Canvas->Font->Color = clBlack;
    else
      Canvas->Font->Color = clRed;
    XPos = 10;
  }
}

/*
This is the OnCreate event handler for Form1.  It
initializes global values and sets the application's OnIdle
event handler
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Button1->Caption = "Do not yield";
  Button2->Caption  = "Handle Message";
  Application->OnIdle = MyIdleHandler;
  XPos = 10;
  YPos = 10;
  if (Canvas->Font->Height > 0)
    Delta = Canvas->Font->Height + 1;
  else
    Delta = 1 - Canvas->Font->Height;
}

/*
This is the OnIdle event handler. It is set in the Form's
OnCreate event handler, so you need only add it as a private
method of the form.  Usually it would perform some background
processing for the application.  This one simply writes a
message to let you know when it's there.
*/
void __fastcall TForm1::MyIdleHandler(TObject *Sender, bool &Done)
{
  StatusMsg(Form1, Canvas, "This represents a background process.", true);
}

/*
This is the OnClick event handler for Button1.  It
simulates a lengthy process
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int X, Y;
  StatusMsg(Form1, Canvas, "The Button1Click handler is starting", false);
  for (int I = 0; I < 100; I++)
  {
    randomize();
    for (int J = 0; J < 100; J++)
      Y = random(J);
```

```
    X = random(I);
  }
  StatusMsg(Form1, Canvas, "The Button1Click handler is done", false);
}

/*
This is the OnClick event handler for button2.  It
simulates a lengthy process that allows other messages to be
processed.
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  int X, Y;
  StatusMsg(Form1, Canvas, "The Button2Click handler is starting", false);
  for (int I = 0; I < 100; I++)
  {
    randomize();
    for (int J = 0; J < 100; J++)
    {
      Y = random(J);
    }
    // Yield to OnIdle or other messages
    Application->HandleMessage();
    X = random(I);
  }
  StatusMsg(Form1, Canvas, "The Button2Click handler is done", false);
}
```

Delphi Examples:

```
{
The following code demonstrates background processing using
the OnIdle event, and using the HandleMessage method to
permit messages or background processing to get through.
Note: You must add MyIdleHandler to the Form1 methods.
}
var
{ global variables to show the order of events. }
  XPos, YPos, Delta: integer;

{ This is a utility procedure to display messages }
{ add this at the beginning of the implementation section }
procedure StatusMsg(
  MyForm : TForm1; Canvas : TCanvas; Message : string; Bkg : Boolean);
begin
  if not bkg then
    Canvas.Font.Style := [fsBold]; {foreground messages are bold }
  Canvas.TextOut(XPos, YPos, Message);
  if not bkg then
    Canvas.Font.Style := [];
  { change Xpos and YPos to prepare for the next message }
  Ypos := Ypos + Delta;
  if YPos >= MyForm.ClientHeight - 10 then
  begin
    YPos := 10;
    Xpos := Xpos + 180;
  end;
  if (Xpos >= MyForm.ClientWidth - 100) then
  begin
    if (Canvas.Font.Color = clRed) then
      Canvas.Font.Color := clBlack
    else
      Canvas.Font.Color := clRed;
    Xpos := 10;
  end;
end;
```

```
{This is the Form's OnCreate event handler.  }
{ It initializes global variables and sets the  the OnIdle
event handler }
procedure TForm1.FormCreate(Sender: TObject);
begin
  Button1.Caption := 'Do not yield';
  Button2.Caption := 'Handle Message';
  Application.OnIdle:= MyIdleHandler;
  XPos := 10;
  YPos := 10;
  Delta := Abs(Canvas.Font.Height) + 1;
 end;

{
This is the OnIdle event handler. It is set in the Form's
OnCreate event handler, so you need only add it as a private
method of the form.  Usually it would perform some
background processing for the application.  This one simply
writes a message to let you know when it's there.
}
procedure TForm1.MyIdleHandler(Sender: TObject; var Done: Boolean);
begin
  StatusMsg(
    Form1, Canvas, 'This represents a background process.', True);
end;

{
Set this method as the OnClick event handler of Button1.  It
performs a calculation without yielding to other messages or
idle time.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, X, Y: Word;
begin
  StatusMsg(
    Form1, Canvas, 'The synchronous handler is starting', False);
  I := 0;
  J := 0;
  while I < 10 do
  begin
    Randomize;
    while J < 10 do
    begin
      Y := Random(J);
      Inc(J);
    end;
    X := Random(I);
    Inc(I);
  end;
  StatusMsg(
    Form1, Canvas, 'The synchronous handler is done', False);
end;

{
Set this method as the OnClick event handler of Button2.  It
performs a calculation but calls HandleMessage to allow idle
time or asynchronous message processing.
}
procedure TForm1.Button2Click(Sender: TObject);
var
  I, J, X, Y: Word;
begin
 StatusMsg(
    Form1, Canvas, 'The asynchronous handler is starting', False);
  I := 0;
```

```
    J := 0;
    while I < 10 do
    begin
      Randomize;
      while J < 10 do
      begin
        Y := Random(J);
        Inc(J);
      end;
      X := Random(I);
      Inc(I);
      { yield to OnIdle or other messages }
      Application.HandleMessage;
    end;
    StatusMsg(
      Form1, Canvas, 'The asynchronous handler is done', False);
  end;
```

See Also

OnIdle (see page 1361)

CheckSynchronize

OnUpdate (see page 2718)

72.1.1.6.11 TApplication.OnMessage Event

Occurs when the application receives a Windows message.

Pascal

```
property OnMessage: TMessageEvent;
```

C++

```
__property TMessageEvent OnMessage;
```

Description

Use OnMessage to trap any or all Windows messages posted to all windows in the application. The OnMessage event occurs when an application receives a Windows message. An OnMessage event handler allows an application to respond to messages other than those declared in the events for TApplication (see page 13660). If the application doesn't have a specific handler for an incoming message, the message is dispatched to the window for which it was intended, and Windows handles the message.

Note: OnMessage only receives messages that are posted to the message queue, not those sent directly with the Windows API SendMessage function.

Warning: Caution:Thousands of messages per second flow though this event. Be careful when coding the handler, because it can affect the performance of the entire application.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

C++ Examples:

```
/*
The following code handles a custom message that the
application sends to itself when a file is ready for reading.
*/
```

```
const WM_FILEREADY = WM_USER + 2000;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  const PChar Path = "../overview.rtf";
  PostMessage(Application->Handle, WM_FILEREADY, 0, int(Path));
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnMessage = AppMessage;
}

void __fastcall TForm1::AppMessage(tagMSG &Msg, bool &Handled)
{
  if (Msg.message == WM_FILEREADY)
  {
    Memo1->Lines->LoadFromFile(AnsiString((char *)Msg.lParam));
    Handled = true;
  }
/*
For all other messages, Handled remains False so that other
message handlers can respond.
*/
}
```

Delphi Examples:

```
{
The following code handles a custom message that the
application sends to itself when a file is ready for reading.
}
const WM_FILEREADY = WM_USER + 2000;
procedure TForm1.Button1Click(Sender: TObject);
const Path: PChar = 'OverView.RTF';
begin
  PostMessage(Application.Handle, WM_FILEREADY, 0, integer(Path));
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnMessage := AppMessage;
end;

procedure TForm1.AppMessage(var Msg: TMsg; var Handled: Boolean);
var msgStr: String;
begin
  if Msg.hwnd = Application.Handle then
  begin
    if Msg.message = WM_FILEREADY then
    begin
      msgStr:= PChar(Msg.lParam);
      Memo1.Lines.LoadFromFile(msgStr);
      Handled := True;
    end;
  end;
  { for all other messages, Handled remains False }
  { so that other message handlers can respond }
end;
```

See Also

HandleMessage (see page 13674)

ProcessMessages (see page 13691)

OnMessage (see page 1362)

72.1.1.6.12 TApplication.OnMinimize Event

Occurs when an application is minimized.

Pascal

```
property OnMinimize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMinimize;
```

Description

Write an OnMinimize event handler to perform special processing when the application is minimized. The application is minimized, either because the user minimizes the main window, or because of a call to the Minimize (see page 13688) method. The Icon (see page 13717) property determines the icon that represents the minimized application.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

C++ Examples:

```
/*
This example uses a timer on a form. When the application
runs and the user minimizes the application, the timer
starts and the application returns to its normal size when
an OnTimer event occurs.  The timer then shuts down until
the next time the form is minimized.  Be sure to declare the
AppStartTimer method as a public method of TForm1.  Place a
TTimer object in the form and enter Timer1Timer in the OnTimer
event.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnMinimize = AppStartTimer;
  Timer1->Interval = 1000;
}

void __fastcall TForm1::AppStartTimer(TObject *Sender)
{
  Timer1->Enabled = true;
}

//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Application->Restore();
  Timer1->Enabled = false;
}
```

Delphi Examples:

```
{
This example uses a timer on a form. When the application
runs and the user minimizes the application, the timer
starts and the application returns to its normal size when
an OnTimer event occurs.  The timer then shuts down until
the next time the form is minimized.  Be sure to declare the
AppStartTimer method as a public method of TForm1.  Place a
TTimer object in the form and enter Timer1Timer in the OnTimer
event.
}
procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
  Application.OnMinimize := AppStartTimer;
end;

procedure TForm1.AppStartTimer(Sender: TObject);
begin
  Timer1.Enabled := True;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Application.Restore;
  Timer1.Enabled := False;
end;
```

See Also

Icon (see page 13717)

Minimize (see page 13688)

OnRestore (see page 13741)

Restore (see page 13694)

OnMinimize (see page 1363)

72.1.1.6.13 TApplication.OnModalBegin Event

Occurs when a modal form is opened.

Pascal

```
property OnModalBegin: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnModalBegin;
```

Description

Write an OnModalBegin event handler to perform special processing when a modal form is opened.

See Also

OnModalEnd (see page 13740)

ModalStarted (see page 13689)

72.1.1.6.14 TApplication.OnModalEnd Event

Occurs when a modal form is closed.

Pascal

```
property OnModalEnd: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnModalEnd;
```

Description

Write an OnModalEnd event handler to perform special processing when a modal form is closed.

See Also

OnModalBegin (see page 13740)

ModalStarted (see page 13689)

72.1.1.6.15 TApplication.OnRestore Event

Occurs when the previously minimized application is restored to its normal size.

Pascal

```
property OnRestore: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnRestore;
```

Description

Write an OnRestore event handler to perform special processing when the application is restored from the minimized state in which it appears as an icon. An application is restored either because the user restores the application, or because the application calls the Restore (see page 13694) method.

Note: Don't confuse restoring an application with restoring a form or window to its original size. To minimize, maximize, and restore a window or form, change the value of the WindowState property.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

See Also

Icon (see page 13717)

Minimize (see page 13688)

Restore (see page 13694)

OnRestore (see page 1364)

WindowState (see page 13942)

72.1.1.6.16 TApplication.OnSettingChange Event

Occurs when Windows notifies the application that a system-wide setting has changed.

Pascal

```
property OnSettingChange: TSettingChangeEvent;
```

C++

```
__property TSettingChangeEvent OnSettingChange;
```

Description

Use OnSettingChange to respond when Windows informs the application that a system-wide setting or policy has changed.

72.1.1.6.17 TApplication.OnShortCut Event

Occurs when the user presses a key (before the OnKeyDown event).

Pascal

```
property OnShortCut: TShortCutEvent;
```

C++

```
__property TShortCutEvent OnShortCut;
```

Description

Use OnShortCut to dispatch shortcut keystrokes before the form or its controls handle them. When the user presses a key, the application can dispatch it as a shortcut key instead of allowing the standard keystroke processing (OnKeyDown, OnKeyPress, and OnKeyUp). Built-in shortcut processing is provided for menu shortcuts and actions associated with the form. OnShortCut lets the application implement additional shortcuts.

If the OnShortCut event handler responds to the keystroke, set its Handled parameter to true. This prevents the keystroke from being passed on to menus or actions associated with the application. It also prevents the standard keystroke processing in the same way that a menu or action shortcut does.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

See Also

OnShortCut (see page 13977)

OnKeyDown (see page 6225)

OnShortCut (see page 1366)

72.1.1.6.18 TApplication.OnShowHint Event

Occurs when the application is about to display the hint window for a Help Hint (see page 13712).

Pascal

```
property OnShowHint: TShowHintEvent;
```

C++

```
__property TShowHintEvent OnShowHint;
```

Description

Write an OnShowHint event handler to change the appearance and behavior of Help Hints.

In the event handler, set the HintStr parameter to change the text of the Help Hint (see page 13712). To obtain the text of a hint for a particular control, call the GetLongHint or GetShortHint function, and assign the result to HintStr.

Use the CanShow parameter to permit or prevent the Help Hint (see page 13712) from displaying.

Note: You can also respond to this event using the TApplicationEvents component, which allows you to assign an event handler using the IDE.

C++ Examples:

```
/*
This example uses three speed buttons on a panel. The code
changes the color, width, and position of the text in the
Help Hint for the third speed button. Set the ShowHint
property of each speed button to true and give a value to
the Hint property of each.  You must declare the DoShow
method in the type declaration of the form. Once it is
declared, write the code for the DoShow method in the
implementation part of the unit. Finally, in the OnCreate
event handler for the form, assign the method to the
OnShowHint event of the application.
```

```
*/
void __fastcall TForm1::DoShowHint(AnsiString &HintStr, bool &CanShow, THintInfo &HintInfo)
{
  if (HintInfo.HintControl == SpeedButton3)
  {
    HintInfo.HintColor = clAqua;// Changes only for this hint
    HintInfo.HintMaxWidth = 120; // Hint text word wraps if width is greater than 120
    HintInfo.HintPos.x += SpeedButton3->Width; // Move hint to right edge
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->ShowHint = true;
  Application->OnShowHint = DoShowHint;
}
```

Delphi Examples:

```
{
This example uses three speed buttons on a panel. The code
changes the color, width, and position of the text in the
Help Hint for the third speed button. Set the ShowHint
property of each speed button to true and give a value to
the Hint property of each.  You must declare the DoShow
method in the type declaration of the form. Once it is
declared, write the code for the DoShow method in the
implementation part of the unit. Finally, in the OnCreate
event handler for the form, assign the method to the
OnShowHint event of the application.
}

procedure TForm1.DoShowHint(var HintStr: String; var CanShow: Boolean; var HintInfo:
THintInfo);
begin
  if HintInfo.HintControl = SpeedButton3 then
  begin
    with HintInfo do
    begin
      HintColor := clAqua;{ Changes only for this hint }
      HintMaxWidth := 120;{Hint text word wraps if width is greater than 120 }
      Inc(HintPos.X, SpeedButton3.Width); { Move hint to right edge }
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.ShowHint := True;
  Application.OnShowHint := DoShowHint;
end;
```

See Also

GetLongHint (see page 6252)

GetShortHint (see page 6253)

HintHidePause (see page 13715)

OnHint (see page 13731)

OnShowHint (see page 1367)

72.1.2 TControlScrollBar Class

TControlScrollBar displays a horizontal or vertical scroll bar.

Class Hierarchy

Pascal

```
TControlScrollBar = class(TPersistent);
```

C++

```
class TControlScrollBar : public TPersistent;
```

File

Forms

Description

Use TControlScrollBar to control the scroll bar in a scrolling windowed control such as TForm (see page 13992) or TScrollBox (see page 14238). TControlScrollBar introduces properties that control the size of the scrollable region and the position to which that region is scrolled, and when the scroll bar appears.

See Also

HorzScrollBar

TForm (see page 13992)

TScrollBox (see page 14238)

VertScrollBar

TScrollBar (see page 21821)

72.1.2.1 TControlScrollBar Members

The following tables list the members exposed by TControlScrollBar.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 13746)	Copies the contents of a TControlScrollBar (see page 13744) instance to another TControlScrollBar (see page 13744).
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ChangeBiDiPosition (see page 13746)	Changes the position of a horizontal scroll bar to match the current bi-directional mode.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsScrollBarVisible (see page 13747)	Returns true if the scroll bar is visible.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Kind (see page 13750)	Indicates whether the scroll bar is horizontal or vertical.
R	ScrollPos (see page 13756)	Indicates the position of the thumb tab.

Published Properties

	Name	Description
	ButtonSize (see page 13748)	Specifies the sizes of the buttons in the scroll bar.
	Color (see page 13748)	Specifies the scroll bar's color.
	Increment (see page 13748)	Determines how far the display moves when the user clicks one of the small end arrows on the scroll bar.
	Margin (see page 13750)	Determines when a scroll bar is generated.
	ParentColor (see page 13751)	Specifies whether the scroll bar should.
	Position (see page 13752)	Specifies the position of the thumb tab on the scroll bar.
	Range (see page 13753)	Determines how far the scrolling region of the associated control can move.
	Size (see page 13756)	Specifies the scroll bar's size.
	Smooth (see page 13756)	Specifies whether the amount the associated control scrolls is dynamically computed.
	Style (see page 13757)	Specifies the scroll bar's style.
	ThumbSize (see page 13757)	This is ThumbSize, a member of class TControlScrollBar.
	Tracking (see page 13757)	Determines whether the form or scroll box moves before the thumb tab is released.
	Visible (see page 13758)	Determines whether the scroll bar appears.

72.1.2.2 TControlScrollBar Methods

The methods of the TControlScrollBar class are listed here.

Public Methods

	Name	Description
	Assign (see page 13746)	Copies the contents of a TControlScrollBar (see page 13744) instance to another TControlScrollBar (see page 13744).
	ChangeBiDiPosition (see page 13746)	Changes the position of a horizontal scroll bar to match the current bi-directional mode.
	IsScrollBarVisible (see page 13747)	Returns true if the scroll bar is visible.

72.1.2.2.1 TControlScrollBar.Assign Method

Copies the contents of a TControlScrollBar (see page 13744) instance to another TControlScrollBar (see page 13744).

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Assign copies the Visible (see page 13758), Range (see page 13753), Position (see page 13752), and Increment (see page 13748) properties from the source object to the TControlScrollBar (see page 13744) instance. If the Source parameter is not another scroll bar, Assign calls the TPersistent Assign method, which allows the source object to copy properties if it has an AssignTo method that knows how to copy to a control scroll bar.

See Also

Assign (see page 2930)

AssignTo (see page 2931)

Position (see page 13752)

Visible (see page 13758)

Range (see page 13753)

72.1.2.2.2 TControlScrollBar.ChangeBiDiPosition Method

Changes the position of a horizontal scroll bar to match the current bi-directional mode.

Pascal

```
procedure ChangeBiDiPosition;
```

C++

```
__fastcall ChangeBiDiPosition();
```

Description

Use ChangeBiDiPosition to change the position of a horizontal scroll bar to reflect the current bi-directional mode of its associated scrolling windowed control.

For example, TScrollingWinControl (see page 14312) calls ChangeBiDiPosition from its DoFlipChildren method to determine whether the scroll bar should be repositioned along with the child controls.

See Also

BiDiMode (see page 5803)

DoFlipChildren (see page 14327)

72.1.2.2.3 TControlScrollBar.IsScrollBarVisible Method

Returns true if the scroll bar is visible.

Pascal

```
function IsScrollBarVisible: Boolean;
```

C++

```
__fastcall Boolean IsScrollBarVisible();
```

Description

Call IsScrollBarVisible to determine whether the scroll bar is visible. IsScrollBarVisible returns true if the Visible (see page 13758) property is true and the associated scrolling windowed control contains child controls that lie within Margin (see page 13750) pixels from the edge.

See Also

Visible (see page 13758)

Margin (see page 13750)

72.1.2.3 TControlScrollBar Properties

The properties of the TControlScrollBar class are listed here.

Public Properties

	Name	Description
R	Kind (see page 13750)	Indicates whether the scroll bar is horizontal or vertical.
R	ScrollPos (see page 13756)	Indicates the position of the thumb tab.

Published Properties

	Name	Description
	ButtonSize (see page 13748)	Specifies the sizes of the buttons in the scroll bar.
	Color (see page 13748)	Specifies the scroll bar's color.
	Increment (see page 13748)	Determines how far the display moves when the user clicks one of the small end arrows on the scroll bar.
	Margin (see page 13750)	Determines when a scroll bar is generated.
	ParentColor (see page 13751)	Specifies whether the scroll bar should.
	Position (see page 13752)	Specifies the position of the thumb tab on the scroll bar.
	Range (see page 13753)	Determines how far the scrolling region of the associated control can move.
	Size (see page 13756)	Specifies the scroll bar's size.
	Smooth (see page 13756)	Specifies whether the amount the associated control scrolls is dynamically computed.
	Style (see page 13757)	Specifies the scroll bar's style.
	ThumbSize (see page 13757)	This is ThumbSize, a member of class TControlScrollBar.
	Tracking (see page 13757)	Determines whether the form or scroll box moves before the thumb tab is released.
	Visible (see page 13758)	Determines whether the scroll bar appears.

72.1.2.3.1 TControlScrollBar.ButtonSize Property

Specifies the sizes of the buttons in the scroll bar.

Pascal

```
property ButtonSize: Integer;
```

C++

```
__property int ButtonSize;
```

Description

Use ButtonSize to specify the sizes of the buttons in the scroll bar. If the scroll bar is horizontal, ButtonSize is the width of the arrow buttons in pixels. If the scroll bar is vertical, ButtonSize is the height of the arrow buttons in pixels.

See Also

Size (see page 13756)

Kind (see page 13750)

thumbsize (see page 13757)

72.1.2.3.2 TControlScrollBar.Color Property

Specifies the scroll bar's color.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to specify the scroll bar's color. When you explicitly assign a value to Color, the ParentColor (see page 13751) property is automatically set to false.

If the scroll bar's ParentColor (see page 13751) property is true, then the Color property reflects the Windows button highlight color (clBtnHighlight), and changing this color in the Windows control panel automatically updates the value of Color.

See Also

ParentColor (see page 13751)

72.1.2.3.3 TControlScrollBar.Increment Property

Determines how far the display moves when the user clicks one of the small end arrows on the scroll bar.

Pascal

```
property Increment: TScrollBarInc;
```

C++

```
__property TScrollBarInc Increment;
```

Description

Use Increment to control how the scroll bar affects the scrollable region of the control. If Increment is set to 1, each click scrolls by one position; if Increment is set to 2, each click scrolls by two positions; and so forth.

Note: Do not use the Increment property when Smooth (see page 13756) is set to true. When Smooth (see page 13756) is true, every time the range or visibility of the scroll bar changes, the value of Increment is dynamically recomputed.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
/*
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  const int iPageDelta = 10;

  if (Key == VK_NEXT)
    VertScrollBar->Position += iPageDelta;
  else if (Key == VK_PRIOR)
    VertScrollBar->Position -= iPageDelta;
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form's logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
    Range := 600;
    Position := Range - ClientWidth; { start form out fully scrolled }
    Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
    Visible := True;  { Show the scrollbar }
  end;
end;
{
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
}
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
```

```
  Shift: TShiftState);
  const
    PageDelta = 10;
  begin
    With VertScrollbar do
      if Key = VK_NEXT then
        Position := Position + PageDelta
      else if Key = VK_PRIOR then
        Position := Position - PageDelta;
  end;
```

See Also

Smooth (see page 13756)

72.1.2.3.4 TControlScrollBar.Kind Property

Indicates whether the scroll bar is horizontal or vertical.

Pascal

```
property Kind: TScrollBarKind;
```

C++

```
__property TScrollBarKind Kind;
```

Description

Kind indicates whether the scroll bar has a vertical or horizontal orientation. This value is initialized in the constructor, when the scroll bar is created by a scrolling windowed control for its VertScrollBar or HorzScrollBar property.

See Also

VertScrollBar (see page 14330)

HorzScrollBar (see page 14329)

Create

72.1.2.3.5 TControlScrollBar.Margin Property

Determines when a scroll bar is generated.

Pascal

```
property Margin: Word;
```

C++

```
__property Word Margin;
```

Description

The Margin property determines the minimum number of pixels that must separate each control from the edge of the control that uses the scroll bar. At runtime, whenever a child control is less than Margin pixels from the edge and Visible (see page 13758) is set to true, a scroll bar appears.

C++ Examples:

```
/*
This example uses a button and a label on a form and needs
the AutoScroll property set on the form. Place the label
near the left side of the form, and place the button
somewhere near the middle of the form. When the user runs
the application, a horizontal scroll bar does not appear,
because no control on the form is close enough to the right
```

```
edge.  Each time the user clicks the button, the button moves
25 pixels to the right, and the calculated Range value is
reported in the caption of the label. Repeatedly clicking the
button eventually moves the button close enough to the edge
of the form (within the Margin amount) so that a horizontal
scroll bar appears.  Note that the Range property is
continuously updated as the button moves to the right.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Button1->Left = Button1->Left + 25;
  Label1->Caption = IntToStr(HorzScrollBar->Range);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  HorzScrollBar->Margin = 25;
  HorzScrollBar->Increment = 10;
}
```

Delphi Examples:

```
{
This example uses a button and a label on a form and needs
the AutoScroll property set on the form. Place the label
near the left side of the form, and place the button
somewhere near the middle of the form. When the user runs
the application, a horizontal scroll bar does not appear,
because no control on the form is close enough to the right
edge.  Each time the user clicks the button, the button moves
25 pixels to the right, and the calculated Range value is
reported in the caption of the label. Repeatedly clicking the
button eventually moves the button close enough to the edge
of the form (within the Margin amount) so that a horizontal
scroll bar appears.  Note that the Range property is
continuously updated as the button moves to the right.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  with HorzScrollBar do
  begin
    Margin:= 25;
    Increment := 10;
    Visible := True;
  end;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.Left := Button1.Left + 25;
  Label1.Caption := IntToStr(HorzScrollBar.Range);
end;
```

See Also

Visible (see page 13758)

IsScrollBarVisible (see page 13747)

72.1.2.3.6 TControlScrollBar.ParentColor Property

Specifies whether the scroll bar should.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

Use ParentColor to indicate that the scroll bar should always reflect the Windows button highlight color. Using ParentColor differs from setting the Color (see page 13748) property to clBtnHighlight in that when ParentColor is true, changing the button highlight color in the Windows control panel automatically updates the value of Color (see page 13748).

When the value of the Color (see page 13748) property is changed, its ParentColor property is automatically set to false.

See Also

Color (see page 13748)

TColor (see page 14648)

72.1.2.3.7 TControlScrollBar.Position Property

Specifies the position of the thumb tab on the scroll bar.

Pascal

```
property Position: Integer;
```

C++

```
__property int Position;
```

Description

Use Position to programmatically scroll the client area of the associated control. The value of Position changes (from 0 to Range (see page 13753)) at runtime as the user scrolls the scroll bar.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
/*
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  const int iPageDelta = 10;

  if (Key == VK_NEXT)
```

```
    VertScrollBar->Position += iPageDelta;
  else if (Key == VK_PRIOR)
    VertScrollBar->Position -= iPageDelta;
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form’s logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
    Range := 600;
    Position := Range - ClientWidth; { start form out fully scrolled }
    Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
    Visible := True;  { Show the scrollbar }
  end;
end;
{
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
}
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
const
  PageDelta = 10;
begin
  With VertScrollbar do
    if Key = VK_NEXT then
      Position := Position + PageDelta
    else if Key = VK_PRIOR then
      Position := Position - PageDelta;
end;
```

See Also

Range (see page 13753)

ScrollPos (see page 13756)

72.1.2.3.8 TControlScrollBar.Range Property

Determines how far the scrolling region of the associated control can move.

Pascal

```
property Range: Integer;
```

C++

```
__property int Range;
```

Description

Range represents the virtual size (in pixels) of the associated control's client area. For example, if the Range of a form's horizontal scroll bar is set to 500, and the width of the form is 200, the scroll bar's Position (see page 13752) can vary from 0 to 300.

If the Range of a horizontal scroll bar is less than the width of the form or scroll box, no horizontal scroll bar appears. If the

Range of a vertical scroll bar is less than the height of the form or scroll box, no vertical scroll bar appears.

The value of Range is set automatically, but can be overridden in the Object Inspector or in program code.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
/*
This example uses a button and a label on a form and needs
the AutoScroll property set on the form. Place the label
near the left side of the form, and place the button
somewhere near the middle of the form. When the user runs
the application, a horizontal scroll bar does not appear,
because no control on the form is close enough to the right
edge.  Each time the user clicks the button, the button moves
25 pixels to the right, and the calculated Range value is
reported in the caption of the label. Repeatedly clicking the
button eventually moves the button close enough to the edge
of the form (within the Margin amount) so that a horizontal
scroll bar appears.  Note that the Range property is
continuously updated as the button moves to the right.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Button1->Left = Button1->Left + 25;
  Label1->Caption = IntToStr(HorzScrollBar->Range);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  HorzScrollBar->Margin = 25;
  HorzScrollBar->Increment = 10;
}
/*
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  const int iPageDelta = 10;

  if (Key == VK_NEXT)
    VertScrollBar->Position += iPageDelta;
  else if (Key == VK_PRIOR)
    VertScrollBar->Position -= iPageDelta;
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form's logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
    Range := 600;
    Position := Range - ClientWidth; { start form out fully scrolled }
    Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
    Visible := True;  { Show the scrollbar }
  end;
end;
{
This example uses a button and a label on a form and needs
the AutoScroll property set on the form. Place the label
near the left side of the form, and place the button
somewhere near the middle of the form. When the user runs
the application, a horizontal scroll bar does not appear,
because no control on the form is close enough to the right
edge.  Each time the user clicks the button, the button moves
25 pixels to the right, and the calculated Range value is
reported in the caption of the label. Repeatedly clicking the
button eventually moves the button close enough to the edge
of the form (within the Margin amount) so that a horizontal
scroll bar appears.  Note that the Range property is
continuously updated as the button moves to the right.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  with HorzScrollBar do
  begin
    Margin:= 25;
    Increment := 10;
    Visible := True;
  end;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.Left := Button1.Left + 25;
  Label1.Caption := IntToStr(HorzScrollBar.Range);
end;
{
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
}
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
const
  PageDelta = 10;
begin
  With VertScrollbar do
    if Key = VK_NEXT then
      Position := Position + PageDelta
    else if Key = VK_PRIOR then
      Position := Position - PageDelta;
end;
```

See Also

Position (see page 13752)

Width (see page 5849)

Height (see page 5825)

72.1.2.3.9 TControlScrollBar.ScrollPos Property

Indicates the position of the thumb tab.

Pascal

```
property ScrollPos: Integer;
```

C++

```
__property int ScrollPos;
```

Description

The read-only ScrollPos property returns the current value of Position (see page 13752) when the scroll bar is visible. If the scroll bar is not visible, ScrollPos returns 0.

See Also

Position (see page 13752)

Visible (see page 13758)

72.1.2.3.10 TControlScrollBar.Size Property

Specifies the scroll bar's size.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

Use Size to specify the width of the scroll bar in pixels.

Note: The length of the scroll bar is determined by the width (for horizontal scroll bars) or height (for vertical scroll bars) of the associated scrolling windowed control.

See Also

ButtonSize (see page 13748)

thumbsize (see page 13757)

72.1.2.3.11 TControlScrollBar.Smooth Property

Specifies whether the amount the associated control scrolls is dynamically computed.

Pascal

```
property Smooth: Boolean;
```

C++

```
__property Boolean Smooth;
```

Description

Set Smooth to true to specify that the scroll bar should dynamically adjust its scrolling increment based on the size of the associated control's client area.

When Smooth is true, clicking the end arrows of the scroll bar scrolls the associated control by approximately one tenth of its client area, and clicking the scroll bar beside the thumb tab scrolls the control by the length of the client area minus the amount scrolled by the end arrows. When Smooth is false, the Increment (see page 13748) property controls the amount of scrolling when the user clicks the end arrows, and clicking beside the thumb tab scrolls the control by the entire length of the client area.

See Also

Increment (see page 13748)

Position (see page 13752)

72.1.2.3.12 TControlScrollBar.Style Property

Specifies the scroll bar's style.

Pascal

```
property Style: TScrollBarStyle;
```

C++

```
__property TScrollBarStyle Style;
```

Description

Use Style to specify the scroll bar's style. The style determines whether the scroll bar appears three-dimensional, flat, or uses hot tracking.

72.1.2.3.13 TControlScrollBar.ThumbSize Property

Pascal

```
property ThumbSize: Integer;
```

C++

```
__property int ThumbSize;
```

Description

This is ThumbSize, a member of class TControlScrollBar.

72.1.2.3.14 TControlScrollBar.Tracking Property

Determines whether the form or scroll box moves before the thumb tab is released.

Pascal

```
property Tracking: Boolean;
```

C++

```
__property Boolean Tracking;
```

Description

Use Tracking to control whether the scroll bar and associated control repaint while the thumb tab is dragged with the mouse. If Tracking is true, the image moves as the user drags the thumb tab. If Tracking is false, the image remains static until the thumb tab is released.

Repainting while the thumb tab is dragged can provide useful feedback to the user about how far the control has scrolled, but slows performance.

72.1.2.3.15 TControlScrollBar.Visible Property

Determines whether the scroll bar appears.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

If Visible is set to true, a scroll bar appears whenever a visual component is less than Margin (see page 13750) pixels from the edge of the form or scroll box. If Visible is set to false, the scroll bar is never visible. This is useful, for example, for programmatically controlling the scroll position of a form without allowing the user to control the scroll position.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
/*
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  const int iPageDelta = 10;

  if (Key == VK_NEXT)
    VertScrollBar->Position += iPageDelta;
  else if (Key == VK_PRIOR)
    VertScrollBar->Position -= iPageDelta;
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
```

```
    with HorzScrollBar do
    begin
      { Set the range to twice the ClientWidth of the form }
      { This means that the form's logical size is twice as big }
      { as the physical window. }
      { Note that Range must always be larger than the ClientWidth }
      Range := 600;
      Position := Range - ClientWidth; { start form out fully scrolled }
      Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
      Visible := True;  { Show the scrollbar }
    end;
  end;
  {
  The following example demonstrates form scrolling with the
  PgUp and PgDn keys.
  }
  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
  const
    PageDelta = 10;
  begin
    With VertScrollbar do
      if Key = VK_NEXT then
        Position := Position + PageDelta
      else if Key = VK_PRIOR then
        Position := Position - PageDelta;
  end;
```

See Also

Margin (see page 13750)

72.1.3 TCustomActiveForm Class

TCustomActiveForm is the base class for Active (see page 13896) Forms (see page 13657).

Class Hierarchy

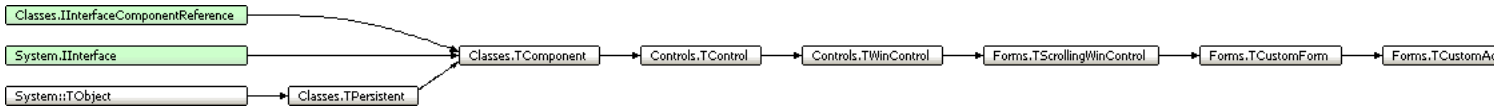

Pascal

```
TCustomActiveForm = class(TCustomForm);
```

C++

```
class TCustomActiveForm : public TCustomForm;
```

File

Forms

Description

TCustomActiveForm is used in code that is generated automatically by the ActiveForm wizard. For more information, see Generating an ActiveX control based on a VCL form.

72.1.3.1 TCustomActiveForm Members

The following tables list the members exposed by TCustomActiveForm.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnCanResize (see page 13947)	Occurs when an attempt is made to resize the control.
	OnClose (see page 13949)	Occurs when the form closes.
	OnCloseQuery (see page 13950)	Occurs when close is attempted.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnHelp (see page 13959)	Occurs when the form receives a request for help.
	OnHide (see page 13960)	Occurs when the form is hidden (that is, when its Visible (see page 13940) property is set to false).
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnResize (see page 13976)	Occurs immediately after the control is resized.
	OnShortCut (see page 13977)	Occurs when the user presses a key (before the OnKeyDown (see page 13962) event).
	OnShow (see page 13977)	Occurs when the form is shown (that is, when its Visible (see page 13940) property is set to true).
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFormState (see page 13863)	This is FFormState, a member of class TCustomForm.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.

	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	Activate (see page 13865)	Activation event dispatcher.
	ActiveChanged (see page 13865)	Active (see page 13896) state event dispatcher.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 14324)	Adjusts the ClientRect property to exclude the scroll bars.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 13866)	Aligns all child controls of the form that fall within a specified rectangle.
V	AlignControls (see page 14325)	Aligns the controls in the scrolling windowed control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
V	AutoScrollEnabled (see page 14325)	Indicates whether automatic scrolling is enabled.
V	AutoScrollInView (see page 14325)	Scrolls a control into the visible area of the scrolling windowed control.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
V	BeginAutoDrag (see page 13867)	Responds when the user starts dragging the form if DragMode is dmAutomatic.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
V	ChangeScale (see page 13867)	Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.
V	ChangeScale (see page 14326)	Resizes the scrolling windowed control by a specific ratio.
	Click (see page 5743)	Respond to user click.
	CloseModal (see page 13868)	This is CloseModal, a member of class TCustomForm.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 13773)	Updates the active form's creation parameters according to the value of the AxBorderStyle (see page 13777) property.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWindowHandle (see page 13869)	Creates a form window given a set of window-creation parameters.
V	CreateWnd (see page 6137)	Creates the underlying window.

	CreateWnd (see page 13869)	Creates the form window.
	CreateWnd (see page 14327)	Creates a Windows control corresponding to the TScrollingWinControl (see page 14312) object.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	Deactivate (see page 13870)	Generates an OnDeactivate (see page 13953) event.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DefineProperties (see page 13871)	Designates methods for storing a form's unpublished data on a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyHandle (see page 13871)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWindowHandle (see page 13872)	Destroys the window created in the CreateWindowHandle (see page 13869) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoClose (see page 13873)	Form closing event dispatcher.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoCreate (see page 13873)	Form creation event dispatcher.
	DoDestroy (see page 13873)	Form destruction event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDock (see page 13874)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoFlipChildren (see page 14327)	Flips the control's children.
	DoHide (see page 13874)	Form hide event dispatcher.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoShow (see page 13875)	Form show event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.

	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorderIconStyles (see page 13875)	This is GetBorderIconStyles, a member of class TCustomForm.
	GetBorderStyles (see page 13876)	This is GetBorderStyles, a member of class TCustomForm.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetChildren (see page 13876)	Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetClientRect (see page 13876)	Returns the value of the ClientRect property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloating (see page 13877)	Returns the value of the Floating property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetOwnerWindow (see page 13878)	This is GetOwnerWindow, a member of class TCustomForm.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 13878)	Handles exceptions that are raised by the OnCreate (see page 13951) event handler.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.

	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 13880)	Performs fix-ups when the form is first loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 13882)	Responds to notifications that objects are about to be inserted or removed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 13882)	Generates an OnPaint (see page 13975) event.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
V	PaintWindow (see page 13882)	Renders the image of the form.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
V	PaletteChanged (see page 13883)	Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	QueryInterface (see page 13884)	Returns a reference to a specified interface if that interface is supported by the form.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
V	ReadState (see page 13884)	Reads the form's published properties, stored data, and owned components from a reader object.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
V	RequestAlign (see page 13885)	Instructs the form's parent (if any) to reposition the form, enforcing its Align property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
V	Resizing (see page 13885)	Makes any required adjustments when the form changes size.
V	Resizing (see page 14328)	This is Resizing, a member of class TScrollingWinControl.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.

	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
V	SetChildOrder (see page 13886)	Changes the order in which Child appears when GetChildren (see page 13876) is called.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParent (see page 13887)	Sets the Parent property.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
V	SetParentBiDiMode (see page 13887)	Sets the ParentBiDiMode property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
V	UpdateActions (see page 13891)	Updates all actions associated with the form.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	UpdateWindowState (see page 13892)	Updates the WindowState (see page 13942) property to reflect the current state of the form.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	ValidateRename (see page 13892)	Determines if a form can rename one of its owned components.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.

V	VisibleChanging (see page 13893)	Responds just before the form changes the value of its Visible (see page 13940) property.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WndProc (see page 13893)	Provides specific message responses for the form.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
R	ActiveMDIChild (see page 13897)	Specifies the MDI child that has focus.
	AlphaBlend (see page 13898)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 13898)	Specifies the degree of translucency on a translucent form.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderIcons (see page 13899)	Specifies which icons appear on the title bar of the form.
R	ClientHandle (see page 13904)	Provides access to the handle of the form's internal MDI client window.
W	ClientHeight (see page 13905)	Specifies the height (in pixels) of the form's client area.
W	ClientWidth (see page 13905)	Specifies the width (in pixels) of the form's client area.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Ctl3D (see page 13909)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultMonitor (see page 13909)	Specifies the monitor on which the form appears.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	FormStyle (see page 13912)	Determines the form's style.
	Icon (see page 13917)	Specifies the icon that appears when the form is minimized.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	MDIChildCount (see page 13921)	Specifies the number of open MDI child forms.
R	MDIChildren (see page 13922)	Provides indexed access to all MDI child forms.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ObjectMenuItem (see page 13926)	Represents an OLE object menu item that reacts to selections of OLE objects.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentFont (see page 13927)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
R	PopupChildren (see page 13928)	This is PopupChildren, a member of class TCustomForm.
	Position (see page 13933)	Represents the size and placement of the form.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	TileMode (see page 13936)	Represents how MDI child forms are arranged when the Tile method is called.

	TransparentColor (see page 13938)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 13939)	Indicates the color on the form that appears transparent when TransparentColor (see page 13938) is true.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowMenu (see page 13941)	Specifies the Window menu for an MDI parent form.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 13866)	Construction event dispatcher.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 13866)	Destruction event dispatcher.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Close (see page 13867)	Closes the form.
V	CloseQuery (see page 13868)	Close (see page 13867) attempt event dispatcher.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 13773)	Creates a new TCustomActiveForm (see page 13759) object.
V	CreateNew (see page 13862)	Creates and initializes a new form.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 13870)	Provides message handling for all messages that the form does not fully process.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DefocusControl (see page 13871)	Removes focus from a control on the form.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 13862)	Removes the form from memory.
V	Destroy (see page 14323)	Removes the TScrollingWinControl (see page 14312) object from memory.

	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	DisableAutoRange (see page 14327)	Disables automatic scrolling.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	Dock (see page 13872)	Handles automatic docking of the form into other windows.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EnableAutoRange (see page 14328)	Reenables automatic scrolling.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	FocusControl (see page 13875)	Sets focus to a control on the form.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFormImage (see page 13877)	Returns a bitmap of the form.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	Hide (see page 13879)	Hides the form.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	IsShortCut (see page 13880)	Processes shortcut keys when the form has focus.
	MakeFullyVisible (see page 13881)	Ensures that the form is fully visible on a specified monitor.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	MouseWheelHandler (see page 13881)	Directs mouse wheel messages to the appropriate control.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	Print (see page 13883)	Prints the form.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	RecreateAsPopup (see page 13884)	This is RecreateAsPopup, a member of class TCustomForm.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	Release (see page 13885)	Destroys the form and frees its associated memory.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollInView (see page 14328)	Scrolls a control into the visible area of the scrolling windowed control.
	SendCancelMode (see page 13886)	Cancels modes on the form.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetDesignVisible (see page 6178)	Makes a control visible at design time.
	SetFocus (see page 6179)	Gives the input focus to the control.

V	SetFocus (see page 13886)	Sets focus to the form.
V	SetFocusedControl (see page 13887)	Sets focus to a control on the form.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Show (see page 13888)	Shows the form.
V	ShowModal (see page 13890)	Shows a form as a modal dialog.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UpdateGlassFrame (see page 13892)	This is UpdateGlassFrame, a member of class TCustomForm.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.
V	WantChildKey (see page 13774)	Indicates whether the form processes keyboard input for a control it owns.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Action (see page 13895)	Designates the action associated with the control.
R	Active (see page 13896)	Specifies whether the form has focus.
	ActiveOleControl (see page 13897)	Specifies the OLE control on the form that reacts to changes in focus.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 13900)	Specifies the appearance and behavior of the form border.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 13900)	Provides access to the drawing area of the form.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designer (see page 13910)	Specifies the designer interface for the form.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.

	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropTarget (see page 13910)	Specifies whether the form is the target of a drag-and-drop operation.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	FormState (see page 13911)	Indicates transitional state information about the form.
	GlassFrame (see page 13914)	This is GlassFrame, a member of class TCustomForm.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HelpFile (see page 13916)	Specifies the name of the file the form uses to display Help.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Menu (see page 13923)	Specifies the form's main menu.
	ModalResult (see page 13924)	Represents the return value of a form that is used as a modal dialog.
R	Monitor (see page 13926)	Provides access to the monitor on which the form appears.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	OleFormObject (see page 13927)	Specifies the IOleForm (see page 14340) interface for an in-place OLE object contained in the form.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PopupMode (see page 13933)	This is PopupMode, a member of class TCustomForm.
	PopupParent (see page 13933)	Sets an order for stacked forms that users cannot change.
	ScreenSnap (see page 13935)	Specifies whether form snaps to edge of screen.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	SnapBuffer (see page 13936)	Specifies distance for screen snap.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 13797)	Indicates whether the form is visible.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.
	WindowState (see page 13942)	Represents how the form appears on the screen.

Published Events

	Name	Description
	OnActivate (see page 13799)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnClick (see page 13803)	Occurs when the user clicks the control.
	OnContextPopup (see page 13805)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 13805)	Occurs when the form is created.
	OnDblClick (see page 13807)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

	OnDeactivate (see page 13807)	Occurs when the form loses focus.
	OnDestroy (see page 13808)	Occurs when the form is destroyed.
	OnDragDrop (see page 13809)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13811)	Occurs when the user drags an object over a control.
	OnKeyDown (see page 13814)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 13816)	Occurs when a key is pressed.
	OnKeyUp (see page 13817)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 13819)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 13820)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 13822)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 13823)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 13823)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 13825)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 13827)	Occurs when the form is redrawn.

Published Properties

	Name	Description
	ActiveControl (see page 13775)	Specifies the control that has focus on the form.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 13776)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 13777)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 13777)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AxBorderStyle (see page 13777)	Sets the active form's border style.
	BorderWidth (see page 13778)	Specifies the width of the control's border.
	Caption (see page 13778)	Specifies a text string that identifies the control to the user.
	Color (see page 13781)	Specifies the background color of the control.
	Constraints (see page 13782)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Font (see page 13782)	Controls the attributes of text written on or in the control.
	Height (see page 13783)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HorzScrollBar (see page 13784)	Represents the horizontal scroll bar for the scrolling windowed control.
	KeyPreview (see page 13785)	Specifies whether the form should receive keyboard events before the active control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Left (see page 13920)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 13788)	Specifies when OnCreate and OnDestroy events occur.
	Padding (see page 13789)	Specifies the padding of a control.
	PixelsPerInch (see page 13789)	Represents the proportion of the font on the system on which the form was designed.
	PopupMenu (see page 13789)	Identifies the pop-up menu associated with the control.
	PrintScale (see page 13794)	Represents the proportions of a printed form.

	Scaled (see page 13794)	Specifies whether the form is sized according to the value of the PixelsPerInch property.
	ShowHint (see page 13795)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Top (see page 13937)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	VertScrollBar (see page 13796)	Represents the vertical scroll bar for the scrolling windowed control.
	Width (see page 13798)	Specifies the horizontal size of the control or form in pixels.

72.1.3.2 TCustomActiveForm.Create Constructor

Creates a new TCustomActiveForm (see page 13759) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomActiveForm(TComponent * AOwner);
```

Description

Create is the TCustomActiveForm (see page 13759) constructor.

AOwner specifies a component, typically the global application object, that becomes the value of the Owner property. The owner is responsible for initiating streaming of the active form and for freeing it.

See Also

Owner (see page 2807)

ParentWindow (see page 6210)

72.1.3.3 TCustomActiveForm Methods

The methods of the TCustomActiveForm class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 13773)	Updates the active form's creation parameters according to the value of the AxBorderStyle (see page 13777) property.

Public Methods

	Name	Description
	WantChildKey (see page 13774)	Indicates whether the form processes keyboard input for a control it owns.

72.1.3.3.1 TCustomActiveForm.CreateParams Method

Updates the active form's creation parameters according to the value of the AxBorderStyle (see page 13777) property.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

CreateParams performs the following tasks:

Calls the inherited CreateParams method.

Removes the WS_CAPTION flag from the Style property of Params.

Adds flags to the Style and ExStyle properties of Params according to the value of the AxBorderStyle (see page 13777) property, as shown in the following table:

AxBorderStyleStyleExStyle

AxBorderStyle (see page 13777)	Style	ExStyle
afbNone	Nothing	Nothing
afbSingle	WS_BORDER	Nothing
afbSunken	Nothing	WS_EX_CLIENTEDGE
afbRaised	WS_DLGFRAME	WS_EX_WINDOWEDGE

Note: The creation parameters are not updated when the form is in design mode.

See Also

CreateParams2

72.1.3.3.2 TCustomActiveForm.WantChildKey Method

Indicates whether the form processes keyboard input for a control it owns.

Pascal

```
function WantChildKey(Child: TControl; var Message: TMessage): Boolean; override;
```

C++

```
virtual __fastcall Boolean WantChildKey(TControl * Child, TMessage Message);
```

Description

Do not call WantChildKey. It is used internally to ensure that keyboard events sent to child objects are handled correctly.

See Also

WantChildKey (see page 13893)

72.1.3.4 TCustomActiveForm Properties

The properties of the TCustomActiveForm class are listed here.

Public Properties

	Name	Description
	Visible (see page 13797)	Indicates whether the form is visible.

Published Properties

	Name	Description
	ActiveControl (see page 13775)	Specifies the control that has focus on the form.
	Anchors (see page 13776)	Specifies how the control is anchored to its parent.

	AutoScroll (see page 13777)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 13777)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AxBorderStyle (see page 13777)	Sets the active form's border style.
	BorderWidth (see page 13778)	Specifies the width of the control's border.
	Caption (see page 13778)	Specifies a text string that identifies the control to the user.
	Color (see page 13781)	Specifies the background color of the control.
	Constraints (see page 13782)	Specifies the size constraints for the control.
	Font (see page 13782)	Controls the attributes of text written on or in the control.
	Height (see page 13783)	Specifies the vertical size of the control in pixels.
	HorzScrollBar (see page 13784)	Represents the horizontal scroll bar for the scrolling windowed control.
	KeyPreview (see page 13785)	Specifies whether the form should receive keyboard events before the active control.
	OldCreateOrder (see page 13788)	Specifies when OnCreate and OnDestroy events occur.
	Padding (see page 13789)	Specifies the padding of a control.
	PixelsPerInch (see page 13789)	Represents the proportion of the font on the system on which the form was designed.
	PopupMenu (see page 13789)	Identifies the pop-up menu associated with the control.
	PrintScale (see page 13794)	Represents the proportions of a printed form.
	Scaled (see page 13794)	Specifies whether the form is sized according to the value of the PixelsPerInch property.
	ShowHint (see page 13795)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	VertScrollBar (see page 13796)	Represents the vertical scroll bar for the scrolling windowed control.
	Width (see page 13798)	Specifies the horizontal size of the control or form in pixels.

72.1.3.4.1 TCustomActiveForm.ActiveControl Property

Specifies the control that has focus on the form.

Pascal

```
property ActiveControl: TWinControl;
```

C++

```
__property TWinControl * ActiveControl;
```

Description

Use ActiveControl to get or set the control that has focus on the form. Only one control can have focus at a given time in an application.

If the form does not have focus, ActiveControl is the control on the form that will receive focus when the form receives focus.

Note: When focus shifts to another control, ActiveControl is updated before the OnExit event occurs.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Timer1->Interval = 100;
  if (ActiveControl)
```

```
    ActiveControl->Left = ActiveControl->Left + 1;
}
```

Delphi Examples:

```
{
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  if ActiveControl <> nil then
    ActiveControl.Left := ActiveControl.Left + 1;
end;
```

See Also

ActiveOleControl (see page 13897)

OnExit (see page 6221)

OnEnter (see page 6219)

Active (see page 13896)

72.1.3.4.2 TCustomActiveForm.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

72.1.3.4.3 TCustomActiveForm.AutoScroll Property

Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

If AutoScroll is true, the scroll bars appear automatically when necessary. For example, if the user resizes the control so that some of its controls are partially obscured, scroll bars appear.

If AutoScroll is false, scroll bars don't appear automatically. In this case, use the HorzScrollBar and VertScrollBar properties to make scroll bars appear.

See Also

AutoScrollInView (see page 14325)

HorzScrollBar (see page 14329)

VertScrollBar (see page 14330)

72.1.3.4.4 TCustomActiveForm.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints (see page 5812)

CanAutoSize (see page 5741)

72.1.3.4.5 TCustomActiveForm.AxBorderStyle Property

Sets the active form's border style.

Pascal

```
property AxBorderStyle: TActiveFormBorderStyle;
```

C++

```
__property TActiveFormBorderStyle AxBorderStyle;
```

Description

AxBorderStyle takes one of the following values:

ValuesMeaning

Values	Meaning	
afbNone	No border appears around the active form.	
afbSingle	A single line border appears around the active form.	
afbSunken	The active form gets a sunken border.	
afbRaised	The active form gets a raised border and no caption.	

See Also

BorderStyle (see page 13900)

72.1.3.4.6 TCustomActiveForm.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics or text drawn by the control is clipped to the area within the border.

See Also

ClientRect

BevelOuter (see page 6194)

72.1.3.4.7 TCustomActiveForm.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
```

```
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
```

```
    ToolBar1.Parent := Self;
    AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
    ToolBar1.ShowCaptions := True;
    ToolBar1.Height := 40;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I: Integer;
  begin
    for I := ToolBar1.ButtonCount - 1 downto 0 do
      ToolBar1.Buttons[I].Free;
    ToolBar1.Free;
  end;
```

72.1.3.4.8 TCustomActiveForm.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

72.1.3.4.9 TCustomActiveForm.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

72.1.3.4.10 TCustomActiveForm.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
```

```
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

72.1.3.4.11 TCustomActiveForm.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
```

```
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
HideSelection, Height example
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
        HideSelection := False;
    end;
end;
```

See Also

ClientHeight (see page 5808)

Width (see page 5849)

Using TDBListBox and TDBComboBox

72.1.3.4.12 TCustomActiveForm.HorzScrollBar Property

Represents the horizontal scroll bar for the scrolling windowed control.

Pascal

```
property HorzScrollBar: TControlScrollBar;
```

C++

```
__property TControlScrollBar HorzScrollBar;
```

Description

Use HorzScrollBar to hide, show, or manipulate the horizontal scroll bar for the scrolling windowed control.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form's logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
    Range := 600;
    Position := Range - ClientWidth; { start form out fully scrolled }
    Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
    Visible := True;  { Show the scrollbar }
  end;
end;
```

See Also

AutoScroll (see page 14329)

VertScrollBar (see page 14330)

TControlScrollBar (see page 13744)

72.1.3.4.13 TCustomActiveForm.KeyPreview Property

Specifies whether the form should receive keyboard events before the active control.

Pascal

```
property KeyPreview: Boolean;
```

C++

```
__property Boolean KeyPreview;
```

Description

If KeyPreview is true, keyboard events occur on the form before they occur on the active control. (The active control is specified by the ActiveControl property.)

If KeyPreview is false, keyboard events occur only on the active control.

Navigation keys (Tab, BackTab, the arrow keys, and so on) are unaffected by KeyPreview because they do not generate keyboard events. Similarly, when a button has focus or when its Default property is true, the Enter key is unaffected by KeyPreview because it does not generate a keyboard events.

KeyPreview is false by default.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
```

```
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
```

```
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

ActiveControl (see page 13896)

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

OnKeyPress (see page 6227)

72.1.3.4.14 TCustomActiveForm.OldCreateOrder Property

Specifies when OnCreate and OnDestroy events occur.

Pascal

```
property OldCreateOrder: Boolean;
```

C++

```
__property Boolean OldCreateOrder;
```

Description

When OldCreateOrder is false (the default) the OnCreate event occurs after all constructors are finished and the OnDestroy event occurs before any destructors are called.

This differs from earlier versions of the library (distributed with Delphi version 3 and previous and C++Builder 1), where the OnCreate event occurred when the TCustomForm constructor executed and the OnDestroy event occurred when the TCustomForm destructor executed. Applications that require the OnCreate event and OnDestroy events to occur from the constructor and destructor of the form can set OldCreateOrder to true.

See Also

OnCreate (see page 13951)

OnDestroy (see page 13953)

BeforeDestruction (see page 13866)

AfterConstruction (see page 13866)

72.1.3.4.15 TCustomActiveForm.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding: TPadding;
```

C++

```
__property TPadding Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding (see page 6103)

72.1.3.4.16 TCustomActiveForm.PixelsPerInch Property

Represents the proportion of the font on the system on which the form was designed.

Pascal

```
property PixelsPerInch: Integer;
```

C++

```
__property int PixelsPerInch;
```

Description

Use PixelsPerInch at runtime to change how the form is scaled to the current screen resolution. If PixelsPerInch is changed from its default value, the form will not have the same proportion on every screen resolution.

At design time, PixelsPerInch is set automatically when the form is saved. Changes to PixelsPerInch made in the Object Inspector only affect the form at runtime.

The form's Scaled property must be true for PixelsPerInch to have an effect.

See Also

Scaled (see page 13935)

72.1.3.4.17 TCustomActiveForm.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
```

```
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
```

```
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
```

```
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
```

```
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

72.1.3.4.18 TCustomActiveForm.PrintScale Property

Represents the proportions of a printed form.

Pascal

```
property PrintScale: TPrintScale;
```

C++

```
__property TPrintScale PrintScale;
```

Description

Use PrintScale to get or set the proportions of the printed form. PrintScale can have one of the following TPrintScale values:

Value	Meaning
poNone	No special scaling occurs; therefore, the printed form and how the form appears onscreen may appear squished or stretched.
poProportional	The form is printed so that the printed image is approximately the same visible size as on the screen (WYSIWYG). The form image is scaled so that the printed image is approximately the same visible size as on the screen.
poPrintToFit	The form is printed using the same screen proportions, but in a size that just fits the printed page.

See Also

Print (see page 13883)

72.1.3.4.19 TCustomActiveForm.Scaled Property

Specifies whether the form is sized according to the value of the PixelsPerInch property.

Pascal

```
property Scaled: Boolean;
```

C++

```
__property Boolean Scaled;
```

Description

Scaled determines whether the form adjusts itself to differences between the font used at design time and the current system font used by a system at runtime. If Scaled is true, the form resizes itself and all its child controls to maintain the relationship

between the size of controls and the height of text displayed in the default font. If Scaled is false, no such resizing occurs.

By setting Scaled to true, the form can adjust to fonts that are not scalable (that is, fonts that only supply discrete point sizes) or to differences such as forms designed using small fonts at design time that are run on systems that use large fonts.

The degree of resizing is determined by the PixelsPerInch property, which measures the proportion of the font on the system used when designing the form.

See Also

PixelsPerInch (see page 13928)

Considerations When Dynamically Resizing Forms and Controls

72.1.3.4.20 TCustomActiveForm.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
```

```
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

72.1.3.4.21 TCustomActiveForm.VertScrollBar Property

Represents the vertical scroll bar for the scrolling windowed control.

Pascal

```
property VertScrollBar: TControlScrollBar;
```

C++

```
__property TControlScrollBar VertScrollBar;
```

Description

Use VertScrollBar to hide, show, or manipulate the vertical scroll bar for the scrolling windowed control.

C++ Examples:

```
/*
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
```

```
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  const int iPageDelta = 10;

  if (Key == VK_NEXT)
    VertScrollBar->Position += iPageDelta;
  else if (Key == VK_PRIOR)
    VertScrollBar->Position -= iPageDelta;
}
```

Delphi Examples:

```
{
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
}
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
const
  PageDelta = 10;
begin
  With VertScrollbar do
    if Key = VK_NEXT then
      Position := Position + PageDelta
    else if Key = VK_PRIOR then
      Position := Position - PageDelta;
end;
```

See Also

AutoScroll (see page 14329)

HorzScrollBar (see page 14329)

TControlScrollBar (see page 13744)

72.1.3.4.22 TCustomActiveForm.Visible Property

Indicates whether the form is visible.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use Visible to determine whether the form is visible.

If Visible is true, the form is visible unless it is completely obscured by other forms. If Visible is false, the form is not visible.

The Show and ShowModal methods set Visible to true and bring the form to the front of all open windows.

C++ Examples:

```
/*
The following code uses two forms. Form1 has a button on it. The
second form is used as a tool palette. This code makes the palette
form visible, and ensures it is the top form by bringing it to the
front.  To run this example, you must include Unit2.h in this
module.
*/
#include "Unit2.h"
```

```
void __fastcall TForm1::ShowPaletteButtonClick(TObject *Sender)
{
  if (!Form2->Visible)
  {
    Form2->Visible = true;
    Form2->BringToFront();
  }
}
```

Delphi Examples:

```
{
The following code uses two forms. Form1 has a button on it. The
second form is used as a tool palette. This code makes the palette
form visible, and ensures it is the top form by bringing it to the
front.  To run this example, you must add Unit2 in the uses clause
of your unit.
}
procedure TForm1.ShowPaletteButtonClick(Sender: TObject);
begin
if Form2.Visible = False then Form2.Visible := True;
Form2.BringToFront;
end;
```

See Also

BringToFront (see page 5740)

Show (see page 13888)

ShowModal (see page 13890)

72.1.3.4.23 TCustomActiveForm.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

See Also

ClientWidth (see page 5811)

Height (see page 5825)

Left (see page 5830)

72.1.3.5 TCustomActiveForm Events

The events of the TCustomActiveForm class are listed here.

Published Events

	Name	Description
	OnActivate (see page 13799)	Occurs when the form becomes active.
	OnClick (see page 13803)	Occurs when the user clicks the control.
	OnContextPopup (see page 13805)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 13805)	Occurs when the form is created.
	OnDblClick (see page 13807)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 13807)	Occurs when the form loses focus.
	OnDestroy (see page 13808)	Occurs when the form is destroyed.
	OnDragDrop (see page 13809)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13811)	Occurs when the user drags an object over a control.
	OnKeyDown (see page 13814)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 13816)	Occurs when a key is pressed.
	OnKeyUp (see page 13817)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 13819)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 13820)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 13822)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 13823)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 13823)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 13825)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 13827)	Occurs when the form is redrawn.

72.1.3.5.1 TCustomActiveForm.OnActivate Event

Occurs when the form becomes active.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Use OnActivate to perform special processing when the form receives focus. A form becomes active when focus is transferred to it (when the user clicks on the form, for example).

Note: The OnActivate event of the application (TApplication), not the form, occurs when Windows switches control from another application.

Delphi Examples:

```
{
Here is the code on Form1 that fills the list box.
}
var
  Form1: TForm1;
  Customers: TTable;
  Blobs: TTable;
  Form1Done : Boolean;
  procedure CreateMyTable;
  procedure CreateCustSource;
```

```
    procedure CreateBlobSource;

  implementation

  {$R *.dfm}

  procedure TForm1.FormCreate(Sender: TObject);
  var
    I: Integer;
  begin
    CreateCustSource;
    CreateBlobSource;
    for I := 0 to ComponentCount - 1 do
    begin
      if (Components[I] is TDataSource) then
        ListBox1.Items.AddObject(Components[I].Name,
          Components[I] as TObject);
    end;
  end;

  procedure CreateCustSource();
  begin
    Customers:= TTable.Create(Form1);
    with Customers do
    begin
      DatabaseName := 'DBDEMOS';
      TableType := ttParadox;
      TableName := 'MyCustInfo';
      ReadOnly:= False;

  //    Don't overwrite an existing table
      if (not Customers.Exists) then CreateMyTable
      else
      begin
        if (Customers.Exists AND
          (MessageDlg('MyCustInfo table already exists.  Do you want to rebuild it?',
  mtConfirmation, [mbYes, mbNo], 0) = mrYes)) then
        begin
          Customers.Close;
          Customers.DeleteTable;
          CreateMyTable;
        end;
      end;
    end;
    Form1.CustomerDS.DataSet:= Customers;
    Customers.Active:= True;
  end;

  procedure CreateMyTable();
  var
    i: Integer;
  begin
  //      The Table component must not be active
  //      First, describe the type of table and give
  //      it a name
  //      Next, describe the fields in the table
    with Customers do
    begin
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
```

```
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
          with AddFieldDef do
          begin
            Name := 'Field3';
            DataType := ftString;
            Size := 30;
          end;
          with AddFieldDef do
          begin
            Name := 'Field4';
            DataType := ftString;
            Size := 30;
          end;
        end;

//    Take out this part to remove indexing
//    Next, describe any indexes
        with IndexDefs do
        begin
          Clear;
//          The 1st index has no name because it is
//          a Paradox primary key
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;

//      Call the CreateTable method to create the table
        CreateTable;
        Customers.Active:= True;
        for i := 1 to 20 do
          Customers.AppendRecord([i*111, i*222, i*333, i*444]);
      end;
end;

procedure CreateBlobSource;
var
  i: Integer;
begin
  Blobs:= TTable.Create(Form1);
  with Blobs do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'MyBlobInfo';
    Blobs.Active := False;
    { Don't overwrite an existing table }
//    if Blobs.Exists then
//      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
//    else
    begin
```

```
        if (Blobs.Exists) then
        begin
          Blobs.Close;
          Blobs.DeleteTable;
        end;
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
        { Next, describe the fields in the table }
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftBlob;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
  {
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
  }
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Blobs.Active:= True;
        for i := 1 to 20 do
          Blobs.AppendRecord([i*111, IntToStr(i*222)]);
      end;
    end;
    Form1.BlobDS.DataSet:= Blobs;
    Blobs.Active:= True;
  end;

  procedure TForm1.ListBox1MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
  begin
    Form1Done := True;
  end;

  initialization

  Form1Done := False;
```

```
{
The following code sets the DataSource for a DBGrid
component in Form2 to the currently selected data source in
a list provided in Form1. This allows users to specify the
desired information in Form1, and then move to Form2 to view
the information. Select an item in the listbox and then
activate Form2 by clicking in it.
}
procedure TForm2.FormActivate(Sender: TObject);
begin
  if (Form1Done) then
  begin
    DBGrid2.DataSource :=
      Form1.ListBox1.Items.Objects[Form1.ListBox1.ItemIndex] as TDataSource;
    DBNavigator2.DataSource :=
      Form1.ListBox1.Items.Objects[Form1.ListBox1.ItemIndex] as TDataSource;
  end;
end;
```

See Also

FormStyle (see page 13912)

OnDeactivate (see page 13953)

OnActivate (see page 13726)

72.1.3.5.2 TCustomActiveForm.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
```

```
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

72.1.3.5.3 TCustomActiveForm.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

72.1.3.5.4 TCustomActiveForm.OnCreate Event

Occurs when the form is created.

Pascal

```
property OnCreate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCreate;
```

Description

Use OnCreate to perform special processing when the form is created and is invoked by TCustomForm's constructor. Either implement this event or override the constructor of the form; do not do both. Any objects created in the OnCreate event should be freed by the OnDestroy event.

When a form is being created and its Visible property is true, the following events occur in the order listed:

1. OnCreate2. OnShow3. OnActivate4. OnPaint

   Note: Use of the OnCreate event is discouraged in C++ code because it can interact badly with the form's constructor (see OldCreateOrder). It is recommended that you override the form constructor instead.

C++ Examples:

```
/*
This example requires a new TPageControl, as-is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
evaluates as true.  If Button is not btNext, the previous Tab Page
is selected.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
const
  TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *page = new TTabSheet(PageControl1);
    page->PageControl = PageControl1;
    page->Caption = AnsiString("Page") + IntToStr(i);
    page->Brush->Color = colorPalette[i];
  }
}

void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  PageControl1->SelectNextPage(Button == btNext);
}
```

Delphi Examples:

```
{
This example requires a new TPageControl, as-is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
```

```
  evaluates as true.  If Button is not btNext, the previous Tab Page
  is selected.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  const
    colorPalette: Array[0..11] of TColor =
      (clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
      clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
  begin
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet #' + IntToStr(i);
        Brush.Color := colorPalette[i];
      end;
  end;

  procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
  begin
    PageControl1.SelectNextPage(Button = btNext);
  end;
```

See Also

OnDestroy (see page 13953)

OnActivate (see page 13943)

OnPaint (see page 13975)

OnShow (see page 13977)

OldCreateOrder (see page 13926)

72.1.3.5.5 TCustomActiveForm.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

72.1.3.5.6 TCustomActiveForm.OnDeactivate Event

Occurs when the form loses focus.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Use OnDeactivate to perform special processing when the form transitions from being the active form to another form in the same application becoming the active form. If activation goes to another application, this event is not triggered. To determine if another application has become active, Use the TApplication object's OnDeactivate event.

See Also

OnActivate (see page 13943)

OnDeactivate (see page 13727)

72.1.3.5.7 TCustomActiveForm.OnDestroy Event

Occurs when the form is destroyed.

Pascal

```
property OnDestroy: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDestroy;
```

Description

Use OnDestroy to perform special processing when the form is destroyed. Either implement this event or override the destructor of the class, but not both. This event should destroy any objects created in the OnCreate event.

C++ Examples:

```
/*
The following code allocates a TBitmap object that is used
by the application when the main form is first created. The
OnDestroy method frees the bitmap as the application shuts
down.
*/
Graphics::TBitmap *LogoBmp;
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  LogoBmp = new Graphics::TBitmap();
  LogoBmp->LoadFromFile("../FACTORY.BMP");
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete LogoBmp;
}

void __fastcall TForm1::FormPaint(TObject *Sender)
{
  Canvas->Brush->Bitmap = LogoBmp;
  Canvas->FillRect(Rect(0,0,100,100));
}
```

Delphi Examples:

```
{
The following code explicitly allocates memory for a pointer
in the OnCreate event of Form1, then releases the memory in
the OnDestroy event. Assume that MyPtr is a Pointer type
field of TForm1.
}
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  New(MyPtr);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Dispose(MyPtr);
end;
```

See Also

OnCreate (see page 13951)

Free (see page 22300)

Destroy (see page 13862)

OldCreateOrder (see page 13926)

72.1.3.5.8 TCustomActiveForm.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
```

```
      DestList->DoubleBuffered = true;
      DestList->Color = clWindow;
    }
  }

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    ListBox1->Items->Add("Not");
    ListBox1->Items->Add("In");
    ListBox1->Items->Add("Alphabetical");
    ListBox1->Items->Add("Order");

  }
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;
```

```
// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

72.1.3.5.9 TCustomActiveForm.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
```

```
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

72.1.3.5.10 TCustomActiveForm.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
```

```
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

72.1.3.5.11 TCustomActiveForm.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

72.1.3.5.12 TCustomActiveForm.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
```

```
    FormColor: TColor;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    KeyPreview := True;
    FormColor := Form1.Color;
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

72.1.3.5.13 TCustomActiveForm.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as

maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

72.1.3.5.14 TCustomActiveForm.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
```

```
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
```

```
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

72.1.3.5.15 TCustomActiveForm.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

72.1.3.5.16 TCustomActiveForm.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

72.1.3.5.17 TCustomActiveForm.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}
```

```
// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
```

```
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

72.1.3.5.18 TCustomActiveForm.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
```

```
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

72.1.3.5.19 TCustomActiveForm.OnPaint Event

Occurs when the form is redrawn.

Pascal

```
property OnPaint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPaint;
```

Description

Use OnPaint to perform special processing when the form is redrawn. Any special painting on the form should be done in this event. OnPaint occurs before any controls on the form are painted.

To determine which portions of the form's canvas need to be repainted, use the ClipRect property of the canvas.

If you use the form's Canvas property outside the OnPaint event, it will be erased and drawn-over by the next OnPaint event.

C++ Examples:

```
/*
The following code loads a background bitmap onto the Canvas
of the main form in the OnPaint event handler.
*/
Graphics::TBitmap *TheGraphic;

void __fastcall TForm1::FormPaint(TObject *Sender)// OnPaint event handler
{
  Form1->Canvas->Draw(0, 0, TheGraphic); // Draw the graphic on the Canvas
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TheGraphic = new Graphics::TBitmap(); // Create the bitmap object
  TheGraphic->LoadFromFile("..\\bigf.bmp"); // Load the bitmap from a file
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete TheGraphic;
}
```

Delphi Examples:

```
{
The following code loads a background bitmap onto the Canvas
of the main form in the OnPaint event handler.
}
procedure TForm1.FormPaint(Sender: TObject); { OnPaint event handler}
begin
  Form1.Canvas.Draw(0, 0, TheGraphic); { Draw the graphic on the Canvas }
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  TheGraphic := TBitmap.Create; { Create the bitmap object }
  TheGraphic.LoadFromFile('bigf.bmp'); { Load the bitmap from a file}
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  (TheGraphic as TObject).Free;
end;
```

See Also

Canvas (see page 13900)

ClipRect (see page 14486)

72.1.4 TCustomDockForm Class

TCustomDockForm is the default floating dock site class for TControl objects.

Class Hierarchy

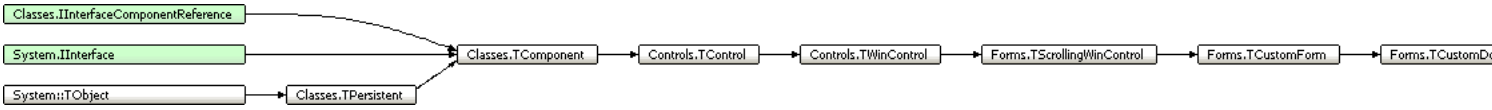

Pascal

```
TCustomDockForm = class(TCustomForm);
```

C++

```
class TCustomDockForm : public TCustomForm;
```

File

Forms

Description

When a control is created, TCustomDockForm is the type assigned to its FloatingDockSiteClass property. You can use TCustomDockForm as the basis for your own floating dock site class, and then assign the new class to the a control's FloatingDockSiteClass property.

See Also

TControl (see page 5724)

72.1.4.1 TCustomDockForm Members

The following tables list the members exposed by TCustomDockForm.

Protected Events

	Name	Description
	OnActivate (see page 13943)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnCanResize (see page 13947)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnClick (see page 13947)	Occurs when the user clicks the control.
	OnClose (see page 13949)	Occurs when the form closes.
	OnCloseQuery (see page 13950)	Occurs when close is attempted.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 13951)	Occurs when the form is created.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDblClick (see page 13953)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 13953)	Occurs when the form loses focus.

	OnDestroy (see page 13953)	Occurs when the form is destroyed.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragDrop (see page 13954)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDragOver (see page 13957)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnHelp (see page 13959)	Occurs when the form receives a request for help.
	OnHide (see page 13960)	Occurs when the form is hidden (that is, when its Visible (see page 13940) property is set to false).
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyDown (see page 13962)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyPress (see page 13964)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnKeyUp (see page 13966)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseActivate (see page 13967)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseDown (see page 13968)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseMove (see page 13971)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseUp (see page 13973)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnPaint (see page 13975)	Occurs when the form is redrawn.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnResize (see page 13976)	Occurs immediately after the control is resized.
	OnShortCut (see page 13977)	Occurs when the user presses a key (before the OnKeyDown (see page 13962) event).
	OnShow (see page 13977)	Occurs when the form is shown (that is, when its Visible (see page 13940) property is set to true).
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFormState (see page 13863)	This is FFormState, a member of class TCustomForm.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	Activate (see page 13865)	Activation event dispatcher.
	ActiveChanged (see page 13865)	Active (see page 13896) state event dispatcher.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 14324)	Adjusts the ClientRect property to exclude the scroll bars.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 13866)	Aligns all child controls of the form that fall within a specified rectangle.
V	AlignControls (see page 14325)	Aligns the controls in the scrolling windowed control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
V	AutoScrollEnabled (see page 14325)	Indicates whether automatic scrolling is enabled.
V	AutoScrollInView (see page 14325)	Scrolls a control into the visible area of the scrolling windowed control.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
V	BeginAutoDrag (see page 13867)	Responds when the user starts dragging the form if DragMode is dmAutomatic.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.

	ChangeScale (see page 13867)	Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.
	ChangeScale (see page 14326)	Resizes the scrolling windowed control by a specific ratio.
	Click (see page 5743)	Respond to user click.
	CloseModal (see page 13868)	This is CloseModal, a member of class TCustomForm.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 13868)	Initializes the window-creation parameter record when the form window is created.
	CreateParams (see page 14326)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWindowHandle (see page 13869)	Creates a form window given a set of window-creation parameters.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 13869)	Creates the form window.
	CreateWnd (see page 14327)	Creates a Windows control corresponding to the TScrollingWinControl (see page 14312) object.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	Deactivate (see page 13870)	Generates an OnDeactivate (see page 13953) event.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DefineProperties (see page 13871)	Designates methods for storing a form's unpublished data on a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyHandle (see page 13871)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWindowHandle (see page 13872)	Destroys the window created in the CreateWindowHandle (see page 13869) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 13843)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoClose (see page 13873)	Form closing event dispatcher.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoCreate (see page 13873)	Form creation event dispatcher.
	DoDestroy (see page 13873)	Form destruction event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDock (see page 13874)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.

	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
V	DoFlipChildren (see page 14327)	Flips the control's children.
	DoHide (see page 13874)	Form hide event dispatcher.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
V	DoRemoveDockClient (see page 13844)	Performs actions when a control is undocked from the dock form.
	DoShow (see page 13875)	Form show event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorderIconStyles (see page 13875)	This is GetBorderIconStyles, a member of class TCustomForm.
	GetBorderStyles (see page 13876)	This is GetBorderStyles, a member of class TCustomForm.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetChildren (see page 13876)	Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetClientRect (see page 13876)	Returns the value of the ClientRect property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloating (see page 13877)	Returns the value of the Floating property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.

	GetOwnerWindow (see page 13878)	This is GetOwnerWindow, a member of class TCustomForm.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
V	GetSiteInfo (see page 13844)	Returns docking information for the dock form.
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 13878)	Handles exceptions that are raised by the OnCreate (see page 13951) event handler.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 13845)	Specifies actions to perform when the dock form is loaded.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 13882)	Responds to notifications that objects are about to be inserted or removed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 13882)	Generates an OnPaint (see page 13975) event.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
V	PaintWindow (see page 13882)	Renders the image of the form.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
V	PaletteChanged (see page 13883)	Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	QueryInterface (see page 13884)	Returns a reference to a specified interface if that interface is supported by the form.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
V	ReadState (see page 13884)	Reads the form's published properties, stored data, and owned components from a reader object.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.

	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	RequestAlign (see page 13885)	Instructs the form's parent (if any) to reposition the form, enforcing its Align property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	Resizing (see page 13885)	Makes any required adjustments when the form changes size.
	Resizing (see page 14328)	This is Resizing, a member of class TScrollingWinControl.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetChildOrder (see page 13886)	Changes the order in which Child appears when GetChildren (see page 13876) is called.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParent (see page 13887)	Sets the Parent property.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentBiDiMode (see page 13887)	Sets the ParentBiDiMode property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateActions (see page 13891)	Updates all actions associated with the form.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.

	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	UpdateWindowState (see page 13892)	Updates the WindowState (see page 13942) property to reflect the current state of the form.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	ValidateRename (see page 13892)	Determines if a form can rename one of its owned components.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	VisibleChanging (see page 13893)	Responds just before the form changes the value of its Visible (see page 13940) property.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WndProc (see page 13893)	Provides specific message responses for the form.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
R	ActiveMDIChild (see page 13897)	Specifies the MDI child that has focus.
	AlphaBlend (see page 13898)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 13898)	Specifies the degree of translucency on a translucent form.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderIcons (see page 13899)	Specifies which icons appear on the title bar of the form.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
R	ClientHandle (see page 13904)	Provides access to the handle of the form's internal MDI client window.
W	ClientHeight (see page 13905)	Specifies the height (in pixels) of the form's client area.
W	ClientWidth (see page 13905)	Specifies the width (in pixels) of the form's client area.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Ctl3D (see page 13909)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultMonitor (see page 13909)	Specifies the monitor on which the form appears.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	Height (see page 13915)	Specifies the vertical size of the control in pixels.
	HorzScrollBar (see page 13916)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 13917)	Specifies the icon that appears when the form is minimized.

	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	MDIChildCount (see page 13921)	Specifies the number of open MDI child forms.
R	MDIChildren (see page 13922)	Provides indexed access to all MDI child forms.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ObjectMenuItem (see page 13926)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 13926)	Specifies when OnCreate (see page 13951) and OnDestroy (see page 13953) events occur.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentFont (see page 13927)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
R	PopupChildren (see page 13928)	This is PopupChildren, a member of class TCustomForm.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	PopupMenu (see page 13928)	Identifies the pop-up menu associated with the control.
	Position (see page 13933)	Represents the size and placement of the form.
	PrintScale (see page 13934)	Represents the proportions of a printed form.
	Scaled (see page 13935)	Specifies whether the form is sized according to the value of the PixelsPerInch (see page 13928) property.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	TileMode (see page 13936)	Represents how MDI child forms are arranged when the Tile method is called.
	TransparentColor (see page 13938)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 13939)	Indicates the color on the form that appears transparent when TransparentColor (see page 13938) is true.
	VertScrollBar (see page 13939)	Represents the vertical scroll bar for the scrolling windowed control.
W	Visible (see page 13940)	Indicates whether the form is visible.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	Width (see page 13941)	Specifies the horizontal size of the control or form in pixels.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowMenu (see page 13941)	Specifies the Window menu for an MDI parent form.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 13866)	Construction event dispatcher.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 13866)	Destruction event dispatcher.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.

	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Close (see page 13867)	Closes the form.
V	CloseQuery (see page 13868)	Close (see page 13867) attempt event dispatcher.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 13843)	Creates an instance of TCustomDockForm (see page 13829).
V	CreateNew (see page 13862)	Creates and initializes a new form.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 13870)	Provides message handling for all messages that the form does not fully process.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DefocusControl (see page 13871)	Removes focus from a control on the form.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 13862)	Removes the form from memory.
V	Destroy (see page 14323)	Removes the TScrollingWinControl (see page 14312) object from memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	DisableAutoRange (see page 14327)	Disables automatic scrolling.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
V	Dock (see page 13872)	Handles automatic docking of the form into other windows.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EnableAutoRange (see page 14328)	Reenables automatic scrolling.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.

	FlipChildren (see page 6150)	Reverses the positions of child controls.
	FocusControl (see page 13875)	Sets focus to a control on the form.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFormImage (see page 13877)	Returns a bitmap of the form.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	Hide (see page 13879)	Hides the form.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	IsShortCut (see page 13880)	Processes shortcut keys when the form has focus.
	MakeFullyVisible (see page 13881)	Ensures that the form is fully visible on a specified monitor.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MouseWheelHandler (see page 13881)	Directs mouse wheel messages to the appropriate control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.

	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	Print (see page 13883)	Prints the form.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	RecreateAsPopup (see page 13884)	This is RecreateAsPopup, a member of class TCustomForm.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	Release (see page 13885)	Destroys the form and frees its associated memory.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollInView (see page 14328)	Scrolls a control into the visible area of the scrolling windowed control.
	SendCancelMode (see page 13886)	Cancels modes on the form.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
V	SetFocus (see page 13886)	Sets focus to the form.
V	SetFocusedControl (see page 13887)	Sets focus to a control on the form.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Show (see page 13888)	Shows the form.
V	ShowModal (see page 13890)	Shows a form as a modal dialog.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UpdateGlassFrame (see page 13892)	This is UpdateGlassFrame, a member of class TCustomForm.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.
V	WantChildKey (see page 13893)	Indicates whether the form processes keyboard input for a control it owns.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Action (see page 13895)	Designates the action associated with the control.
R	Active (see page 13896)	Specifies whether the form has focus.
	ActiveControl (see page 13896)	Specifies the control that has focus on the form.

	ActiveOleControl (see page 13897)	Specifies the OLE control on the form that reacts to changes in focus.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 13845)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 13846)	Specifies the appearance and behavior of the form border.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 13900)	Provides access to the drawing area of the form.
	Caption (see page 13902)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 13908)	Specifies the background color of the control.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designer (see page 13910)	Specifies the designer interface for the form.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropTarget (see page 13910)	Specifies whether the form is the target of a drag-and-drop operation.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Font (see page 13910)	Controls the attributes of text written on or in the control.
R	FormState (see page 13911)	Indicates transitional state information about the form.
	FormStyle (see page 13846)	Determines the form's style.
	GlassFrame (see page 13914)	This is GlassFrame, a member of class TCustomForm.

R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HelpFile (see page 13916)	Specifies the name of the file the form uses to display Help.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	KeyPreview (see page 13918)	Specifies whether the form should receive keyboard events before the active control.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Menu (see page 13923)	Specifies the form's main menu.
	ModalResult (see page 13924)	Represents the return value of a form that is used as a modal dialog.
R	Monitor (see page 13926)	Provides access to the monitor on which the form appears.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	OleFormObject (see page 13927)	Specifies the IOleForm (see page 14340) interface for an in-place OLE object contained in the form.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PopupMode (see page 13933)	This is PopupMode, a member of class TCustomForm.
	PopupParent (see page 13933)	Sets an order for stacked forms that users cannot change.
	ScreenSnap (see page 13935)	Specifies whether form snaps to edge of screen.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	SnapBuffer (see page 13936)	Specifies distance for screen snap.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.
	WindowState (see page 13942)	Represents how the form appears on the screen.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14332)	Occurs when an object with custom alignment is aligned.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HorzScrollBar (see page 14329)	Represents the horizontal scroll bar for the scrolling windowed control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Left (see page 13920)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.

	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	PixelsPerInch (see page 13848)	Represents the proportion of the font on the system on which the form was designed.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Top (see page 13937)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	VertScrollBar (see page 14330)	Represents the vertical scroll bar for the scrolling windowed control.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

72.1.4.2 TCustomDockForm.Create Constructor

Creates an instance of TCustomDockForm (see page 13829).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomDockForm(TComponent * AOwner);
```

Description

Typically, applications do not create instances of TCustomDockForm (see page 13829). Instead, this class is instantiated automatically as needed by controls that specify it as the value of their FloatingDockSiteClass property.

AOwner is the dock form's owner.

See Also

FloatingDockSiteClass (see page 5823)

Floating (see page 5823)

72.1.4.3 TCustomDockForm Methods

The methods of the TCustomDockForm class are listed here.

Protected Methods

	Name	Description
	DoAddDockClient (see page 13843)	Performs actions when a control is docked to the windowed control.
	DoRemoveDockClient (see page 13844)	Performs actions when a control is undocked from the dock form.
	GetSiteInfo (see page 13844)	Returns docking information for the dock form.
	Loaded (see page 13845)	Specifies actions to perform when the dock form is loaded.

72.1.4.3.1 TCustomDockForm.DoAddDockClient Method

Performs actions when a control is docked to the windowed control.

Pascal

```
procedure DoAddDockClient(Client: TControl; const ARect: TRect); override;
```

C++

```
virtual __fastcall DoAddDockClient(TControl * Client, const TRect ARect);
```

Description

DoAddDockClient is called automatically when a client control is docked to the windowed control. As implemented in TWinControl, DoAddDockClient sets the client's Parent property to the control.

Client is the control that has been docked.

ARect describes the boundaries of the region where Client has been docked.

Override DoAddDockClient to perform actions when a control is docked to the windowed control.

Note: DoAddDockClient is called in response to a message sent by the DockDrop method, before the OnDockDrop event.

See Also

Parent (see page 5832)

DockClients (see page 6201)

ManualDock (see page 5771)

DoRemoveDockClient (see page 6147)

DockDrop (see page 6142)

OnDockDrop (see page 6214)

72.1.4.3.2 TCustomDockForm.DoRemoveDockClient Method

Performs actions when a control is undocked from the dock form.

Pascal

```
procedure DoRemoveDockClient(Client: TControl); override;
```

C++

```
virtual __fastcall DoRemoveDockClient(TControl * Client);
```

Description

Client is the control being undocked.

DoRemoveDockClient calls the inherited DoRemoveDockClient method. Then, if the dock form contains no more clients, DoRemoveDockClient calls the Release (see page 13885) method.

See Also

Release (see page 13885)

doadddockclient (see page 13843)

72.1.4.3.3 TCustomDockForm.GetSiteInfo Method

Returns docking information for the dock form.

Pascal

```
procedure GetSiteInfo(Client: TControl; var InfluenceRect: TRect; MousePos: TPoint; var
CanDock: Boolean); override;
```

C++

```
virtual __fastcall GetSiteInfo(TControl * Client, TRect InfluenceRect, TPoint MousePos,
Boolean CanDock);
```

Description

GetSiteInfo overrides the inherited method, setting CanDock to true if the dock form contains one or more clients.

See Also

DockClientCount (see page 6199)

72.1.4.3.4 TCustomDockForm.Loaded Method

Specifies actions to perform when the dock form is loaded.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Loaded calls the inherited method. Then Loaded sets the dock form's active control, if possible.

72.1.4.4 TCustomDockForm Properties

The properties of the TCustomDockForm class are listed here.

Public Properties

	Name	Description
	AutoScroll (see page 13845)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	BorderStyle (see page 13846)	Specifies the appearance and behavior of the form border.
	FormStyle (see page 13846)	Determines the form's style.

Published Properties

	Name	Description
	PixelsPerInch (see page 13848)	Represents the proportion of the font on the system on which the form was designed.

72.1.4.4.1 TCustomDockForm.AutoScroll Property

Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

If AutoScroll is true, the scroll bars appear automatically when necessary. For example, if the user resizes the control so that some of its controls are partially obscured, scroll bars appear.

If AutoScroll is false, scroll bars don't appear automatically. In this case, use the HorzScrollBar and VertScrollBar properties to make scroll bars appear.

See Also

AutoScrollInView (see page 14325)

HorzScrollBar (see page 14329)

VertScrollBar (see page 14330)

72.1.4.4.2 TCustomDockForm.BorderStyle Property

Specifies the appearance and behavior of the form border.

Pascal

```
property BorderStyle: TFormBorderStyle;
```

C++

```
__property TFormBorderStyle BorderStyle;
```

Description

Use BorderStyle to get or set the appearance and behavior of the form border. BorderStyle can be any of the following TForm BorderStyle values:

Value	Meaning
bsDialog	Not resizable; no minimize/maximize menu
bsSingle	Not resizable; minimize/maximize menu
bsNone	Not resizable; no visible border line
bsSizeable	Standard resizable border
bsToolWindow	like bsSingle but with a smaller caption
bsSizeToolWin	like bsSizeable with a smaller caption

Note: Changing the border style of an MDI child form to bsDialog or bsNone has no effect.

See Also

BorderIcons (see page 13899)

72.1.4.4.3 TCustomDockForm.FormStyle Property

Determines the form's style.

Pascal

```
property FormStyle: TFormStyle;
```

C++

```
__property TFormStyle FormStyle;
```

Description

Use FormStyle to get or set the form's style. FormStyle is one of the following values:

Value	Meaning
fsNormal	The form is neither an MDI parent window nor an MDI child window.
fsMDIChild	The form is an MDI child window.

fsMDIForm	The form is an MDI parent window.
fsStayOnTop	This form remains on top of the desktop and of other forms in the project, except any others that also have FormStyle set to fsStayOnTop. If one fsStayOnTop form launches another, neither form will consistently remain on top.

If the form is the main form of an MDI application, its FormStyle property must be set to fsMDIForm.

Note: It is not advisable to change FormStyle at runtime.

Warning: TGraphicControl descendants placed in the client area of a form with FormStyle set to fsMDIForm will not paint.

C++ Examples:

```
/*
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
*/
void __fastcall TForm1::MyArrangeIconsClick(TObject *Sender)
{
  Form1->ArrangeIcons();
}
/*
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
*/
void __fastcall TForm1::TileFormsClick(TObject *Sender)
{
  TileMode = tbVertical;
  Tile();
}
```

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
}
procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
```

```
    Form1.ArrangeIcons;
  end;
  {
  This example ensures that the main form of the application
  is an MDI parent:
  }
  type
    TForm1 = class(TForm)
      Edit1: TEdit;
    private
      { Private declarations }
    public
      { Public declarations }
      constructor Create(AOwner : TComponent); override;
    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  constructor TForm1.Create(AOwner : TComponent);

  begin
    inherited Create(AOwner);
    if FormStyle <> fsMDIForm then
            FormStyle := fsMDIForm;
    if FormStyle = fsMDIForm then
            Edit1.Text := 'MDI form'
    else
      //This line never executes
      Edit1.Text := 'Not an MDI form';
  end;
  {
  This example uses three forms. The first form has its
  FormStyle property set to MDIForm. The other two have their
  FormStyle properties set to MDIChild and their Visible
  properties set to true. Add a main menu component and name
  one of the menu items TileForms. This is code for the
  TileFormsClick handler.  When the user chooses the TileForms
  command, the child forms tile vertically within the MDI frame
  form.
  }
  procedure TForm1.TileFormsClick(Sender: TObject);
  begin
    TileMode := tbVertical;
    Tile;
  end;
```

See Also

ArrangeIcons (see page 14006)

Tile (see page 14009)

Cascade (see page 14007)

Windows Overview

72.1.4.4.4 TCustomDockForm.PixelsPerInch Property

Represents the proportion of the font on the system on which the form was designed.

Pascal

```
property PixelsPerInch: Integer;
```

C++

```
__property int PixelsPerInch;
```

Description

Use PixelsPerInch at runtime to change how the form is scaled to the current screen resolution. If PixelsPerInch is changed from its default value, the form will not have the same proportion on every screen resolution.

At design time, PixelsPerInch is set automatically when the form is saved. Changes to PixelsPerInch made in the Object Inspector only affect the form at runtime.

The form's Scaled property must be true for PixelsPerInch to have an effect.

See Also

Scaled (see page 13935)

72.1.5 TCustomForm Class

TCustomForm is the base class from which to derive a window such as a form or dialog.

Class Hierarchy

Pascal

```
TCustomForm = class(TScrollingWinControl);
```

C++

```
class TCustomForm : public TScrollingWinControl;
```

File

Forms

Description

Derive from TCustomForm to create a custom window. The TCustomForm descendant can contain other objects, such as TButton, TCheckBox, and TComboBox objects.

See Also

TPropertyPage (see page 1603)

TActiveForm (see page 1489)

TForm (see page 13992)

72.1.5.1 TCustomForm Members

The following tables list the members exposed by TCustomForm.

Protected Events

	Name	Description
	OnActivate (see page 13943)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 13947)	Occurs when an attempt is made to resize the control.
	OnClick (see page 13947)	Occurs when the user clicks the control.
	OnClose (see page 13949)	Occurs when the form closes.
	OnCloseQuery (see page 13950)	Occurs when close is attempted.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 13951)	Occurs when the form is created.
	OnDblClick (see page 13953)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 13953)	Occurs when the form loses focus.
	OnDestroy (see page 13953)	Occurs when the form is destroyed.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 13954)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13957)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnHelp (see page 13959)	Occurs when the form receives a request for help.
	OnHide (see page 13960)	Occurs when the form is hidden (that is, when its Visible (see page 13940) property is set to false).
	OnKeyDown (see page 13962)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 13964)	Occurs when a key is pressed.
	OnKeyUp (see page 13966)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 13967)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 13968)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 13971)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 13973)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnPaint (see page 13975)	Occurs when the form is redrawn.
	OnResize (see page 13976)	Occurs immediately after the control is resized.
	OnShortCut (see page 13977)	Occurs when the user presses a key (before the OnKeyDown (see page 13962) event).
	OnShow (see page 13977)	Occurs when the form is shown (that is, when its Visible (see page 13940) property is set to true).
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFormState (see page 13863)	This is FFormState, a member of class TCustomForm.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	Activate (see page 13865)	Activation event dispatcher.
	ActiveChanged (see page 13865)	Active (see page 13896) state event dispatcher.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 14324)	Adjusts the ClientRect property to exclude the scroll bars.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 13866)	Aligns all child controls of the form that fall within a specified rectangle.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
V	AutoScrollEnabled (see page 14325)	Indicates whether automatic scrolling is enabled.
V	AutoScrollInView (see page 14325)	Scrolls a control into the visible area of the scrolling windowed control.
V	BeginAutoDrag (see page 13867)	Responds when the user starts dragging the form if DragMode is dmAutomatic.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	ChangeScale (see page 13867)	Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.
	Click (see page 5743)	Respond to user click.
	CloseModal (see page 13868)	This is CloseModal, a member of class TCustomForm.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.

	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 13868)	Initializes the window-creation parameter record when the form window is created.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 13869)	Creates a form window given a set of window-creation parameters.
	CreateWnd (see page 13869)	Creates the form window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	Deactivate (see page 13870)	Generates an OnDeactivate (see page 13953) event.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 13871)	Designates methods for storing a form's unpublished data on a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 13871)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 13872)	Destroys the window created in the CreateWindowHandle (see page 13869) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoClose (see page 13873)	Form closing event dispatcher.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoCreate (see page 13873)	Form creation event dispatcher.
	DoDestroy (see page 13873)	Form destruction event dispatcher.
	DoDock (see page 13874)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoFlipChildren (see page 14327)	Flips the control's children.
	DoHide (see page 13874)	Form hide event dispatcher.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoShow (see page 13875)	Form show event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.

	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorderIconStyles (see page 13875)	This is GetBorderIconStyles, a member of class TCustomForm.
	GetBorderStyles (see page 13876)	This is GetBorderStyles, a member of class TCustomForm.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
V	GetChildren (see page 13876)	Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 13876)	Returns the value of the ClientRect property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 13877)	Returns the value of the Floating property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetOwnerWindow (see page 13878)	This is GetOwnerWindow, a member of class TCustomForm.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 13878)	Handles exceptions that are raised by the OnCreate (see page 13951) event handler.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 13880)	Performs fix-ups when the form is first loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.

	Notification (see page 13882)	Responds to notifications that objects are about to be inserted or removed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 13882)	Generates an OnPaint (see page 13975) event.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 13882)	Renders the image of the form.
	PaletteChanged (see page 13883)	Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 13884)	Returns a reference to a specified interface if that interface is supported by the form.
	ReadState (see page 13884)	Reads the form's published properties, stored data, and owned components from a reader object.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 13885)	Instructs the form's parent (if any) to reposition the form, enforcing its Align property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	Resizing (see page 13885)	Makes any required adjustments when the form changes size.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 13886)	Changes the order in which Child appears when GetChildren (see page 13876) is called.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 13887)	Sets the Parent property.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 13887)	Sets the ParentBiDiMode property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.

V	UpdateActions (see page 13891)	Updates all actions associated with the form.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	UpdateWindowState (see page 13892)	Updates the WindowState (see page 13942) property to reflect the current state of the form.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 13892)	Determines if a form can rename one of its owned components.
V	VisibleChanging (see page 13893)	Responds just before the form changes the value of its Visible (see page 13940) property.
V	WndProc (see page 13893)	Provides specific message responses for the form.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
R	ActiveMDIChild (see page 13897)	Specifies the MDI child that has focus.
	AlphaBlend (see page 13898)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 13898)	Specifies the degree of translucency on a translucent form.
	AutoScroll (see page 13899)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderIcons (see page 13899)	Specifies which icons appear on the title bar of the form.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	ClientHandle (see page 13904)	Provides access to the handle of the form's internal MDI client window.
W	ClientHeight (see page 13905)	Specifies the height (in pixels) of the form's client area.
W	ClientWidth (see page 13905)	Specifies the width (in pixels) of the form's client area.
	Ctl3D (see page 13909)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultMonitor (see page 13909)	Specifies the monitor on which the form appears.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	FormStyle (see page 13912)	Determines the form's style.
	Height (see page 13915)	Specifies the vertical size of the control in pixels.

	HorzScrollBar (see page 13916)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 13917)	Specifies the icon that appears when the form is minimized.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	MDIChildCount (see page 13921)	Specifies the number of open MDI child forms.
R	MDIChildren (see page 13922)	Provides indexed access to all MDI child forms.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ObjectMenuItem (see page 13926)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 13926)	Specifies when OnCreate (see page 13951) and OnDestroy (see page 13953) events occur.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 13927)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PixelsPerInch (see page 13928)	Represents the proportion of the font on the system on which the form was designed.
R	PopupChildren (see page 13928)	This is PopupChildren, a member of class TCustomForm.
	PopupMenu (see page 13928)	Identifies the pop-up menu associated with the control.
	Position (see page 13933)	Represents the size and placement of the form.
	PrintScale (see page 13934)	Represents the proportions of a printed form.
	Scaled (see page 13935)	Specifies whether the form is sized according to the value of the PixelsPerInch (see page 13928) property.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	TileMode (see page 13936)	Represents how MDI child forms are arranged when the Tile method is called.
	TransparentColor (see page 13938)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 13939)	Indicates the color on the form that appears transparent when TransparentColor (see page 13938) is true.
	VertScrollBar (see page 13939)	Represents the vertical scroll bar for the scrolling windowed control.
W	Visible (see page 13940)	Indicates whether the form is visible.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	Width (see page 13941)	Specifies the horizontal size of the control or form in pixels.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowMenu (see page 13941)	Specifies the Window menu for an MDI parent form.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 13866)	Construction event dispatcher.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 13866)	Destruction event dispatcher.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Close (see page 13867)	Closes the form.
V	CloseQuery (see page 13868)	Close (see page 13867) attempt event dispatcher.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 13861)	Creates and initializes a new TForm (see page 13992) object.
V	CreateNew (see page 13862)	Creates and initializes a new form.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 13870)	Provides message handling for all messages that the form does not fully process.
	DefocusControl (see page 13871)	Removes focus from a control on the form.
V	Destroy (see page 13862)	Removes the form from memory.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	DisableAutoRange (see page 14327)	Disables automatic scrolling.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Dock (see page 13872)	Handles automatic docking of the form into other windows.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EnableAutoRange (see page 14328)	Reenables automatic scrolling.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	FocusControl (see page 13875)	Sets focus to a control on the form.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFormImage (see page 13877)	Returns a bitmap of the form.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 13879)	Hides the form.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	IsShortCut (see page 13880)	Processes shortcut keys when the form has focus.
	MakeFullyVisible (see page 13881)	Ensures that the form is fully visible on a specified monitor.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 13881)	Directs mouse wheel messages to the appropriate control.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	Print (see page 13883)	Prints the form.
	Realign (see page 6171)	Forces the control to realign children.
	RecreateAsPopup (see page 13884)	This is RecreateAsPopup, a member of class TCustomForm.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	Release (see page 13885)	Destroys the form and frees its associated memory.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.

V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollInView (see page 14328)	Scrolls a control into the visible area of the scrolling windowed control.
	SendCancelMode (see page 13886)	Cancels modes on the form.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 13886)	Sets focus to the form.
V	SetFocusedControl (see page 13887)	Sets focus to a control on the form.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 13888)	Shows the form.
V	ShowModal (see page 13890)	Shows a form as a modal dialog.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UpdateGlassFrame (see page 13892)	This is UpdateGlassFrame, a member of class TCustomForm.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.
V	WantChildKey (see page 13893)	Indicates whether the form processes keyboard input for a control it owns.

Public Properties

	Name	Description
	Action (see page 13895)	Designates the action associated with the control.
R	Active (see page 13896)	Specifies whether the form has focus.
	ActiveControl (see page 13896)	Specifies the control that has focus on the form.
	ActiveOleControl (see page 13897)	Specifies the OLE control on the form that reacts to changes in focus.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 13900)	Specifies the appearance and behavior of the form border.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 13900)	Provides access to the drawing area of the form.
	Caption (see page 13902)	Specifies a text string that identifies the control to the user.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	Color (see page 13908)	Specifies the background color of the control.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designer (see page 13910)	Specifies the designer interface for the form.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropTarget (see page 13910)	Specifies whether the form is the target of a drag-and-drop operation.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Font (see page 13910)	Controls the attributes of text written on or in the control.
R	FormState (see page 13911)	Indicates transitional state information about the form.
	GlassFrame (see page 13914)	This is GlassFrame, a member of class TCustomForm.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HelpFile (see page 13916)	Specifies the name of the file the form uses to display Help.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	KeyPreview (see page 13918)	Specifies whether the form should receive keyboard events before the active control.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Menu (see page 13923)	Specifies the form's main menu.
	ModalResult (see page 13924)	Represents the return value of a form that is used as a modal dialog.
R	Monitor (see page 13926)	Provides access to the monitor on which the form appears.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	OleFormObject (see page 13927)	Specifies the IOleForm (see page 14340) interface for an in-place OLE object contained in the form.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PopupMode (see page 13933)	This is PopupMode, a member of class TCustomForm.
	PopupParent (see page 13933)	Sets an order for stacked forms that users cannot change.
	ScreenSnap (see page 13935)	Specifies whether form snaps to edge of screen.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	SnapBuffer (see page 13936)	Specifies distance for screen snap.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.

	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.
	WindowState (see page 13942)	Represents how the form appears on the screen.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14332)	Occurs when an object with custom alignment is aligned.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 13920)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 13937)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.

72.1.5.2 TCustomForm.Create Constructor

Creates and initializes a new TForm (see page 13992) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomForm(TComponent * AOwner);
```

Description

Use Create to instantiate a TForm (see page 13992) object or descendant of TForm (see page 13992) at runtime. AOwner is the owner of the TCustomForm (see page 13849) object.

For descendants of TCustomForm (see page 13849) that are not TForm (see page 13992) objects, use CreateNew (see page 13862) instead. Calling Create for a TCustomForm (see page 13849) descendant that is not a TForm (see page 13992) will raise an exception.

See Also

CreateNew (see page 13862)

TForm (see page 13992)

Owner (see page 2807)

72.1.5.3 TCustomForm.CreateNew Constructor

Creates and initializes a new form.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); virtual;
```

C++

```
virtual __fastcall TCustomForm(TComponent * AOwner, int Dummy = 0);
```

Description

Use CreateNew instead of Create (see page 13861) to create a form without using the associated .DFM file to initialize it. Always use CreateNew if the TCustomForm (see page 13849) descendant is not a TForm (see page 13992) object or a descendant of TForm (see page 13992).

CreateNew bypasses the streaming in of the previously-associated .DFM file. If the form contains visual components, therefore, you must stream in an external .DFM to bind the visual components with their classes. If the newly created form has an external .DFM file, then you can follow the call to CreateNew with a call to InitInheritedComponent. If you need to create the .dfm file for the new form instance, bracket the call to CreateNew with calls to WriteComponentResFile and ReadComponentResFile. The following code sequence

1. streams out an external .DFM file;2. creates a new form disassociated from any .DFM resource file;3. streams in the external .DFM file and binds it to this new form.

```
WriteComponentResFile('Temp.dfm', Form1);
...
```

Form2 := TForm.CreateNew(Application (see page 14383));

```
ReadComponentResFile('Temp.dfm', Form2);
```

Warning: Using CreateNew instead of Create (see page 13861) can cause unpredictable results because most forms are written assuming their controls will be created from the .DFM file.

CreateNew is called by the Create (see page 13861) method.

The first argument, AOwner, is the owner of the form (usually the application or another form is the owner). The second argument, Dummy, defaults to 0 and is not used by this constructor. The system passes 0 by default if this constructor is passed only the first argument, AOwner.

See Also

Create (see page 13861)

Owner (see page 2807)

ReadComponentResFile

WriteComponentResFile

72.1.5.4 TCustomForm.Destroy Destructor

Removes the form from memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomForm();
```

Description

Do not call Destroy. Instead, call Release (see page 13885), which indirectly produces a call to the Free method, which in turn calls Destroy.

See Also

OnDestroy (see page 13953)

Release (see page 13885)

Free (see page 22300)

72.1.5.5 TCustomForm Fields

The fields of the TCustomForm class are listed here.

Protected Fields

	Name	Description
	FFormState (see page 13863)	This is FFormState, a member of class TCustomForm.

72.1.5.5.1 TCustomForm.FFormState Field

Pascal

```
FFormState: TFormState;
```

C++

```
TFormState FFormState;
```

Description

This is FFormState, a member of class TCustomForm.

72.1.5.6 TCustomForm Methods

The methods of the TCustomForm class are listed here.

Protected Methods

	Name	Description
	Activate (see page 13865)	Activation event dispatcher.
	ActiveChanged (see page 13865)	Active (see page 13896) state event dispatcher.
V	AlignControls (see page 13866)	Aligns all child controls of the form that fall within a specified rectangle.
V	BeginAutoDrag (see page 13867)	Responds when the user starts dragging the form if DragMode is dmAutomatic.
V	ChangeScale (see page 13867)	Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.
	CloseModal (see page 13868)	This is CloseModal, a member of class TCustomForm.
V	CreateParams (see page 13868)	Initializes the window-creation parameter record when the form window is created.
V	CreateWindowHandle (see page 13869)	Creates a form window given a set of window-creation parameters.

	CreateWnd (see page 13869)	Creates the form window.
	Deactivate (see page 13870)	Generates an OnDeactivate (see page 13953) event.
	DefineProperties (see page 13871)	Designates methods for storing a form's unpublished data on a stream such as a form file.
	DestroyHandle (see page 13871)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 13872)	Destroys the window created in the CreateWindowHandle (see page 13869) method.
	DoClose (see page 13873)	Form closing event dispatcher.
	DoCreate (see page 13873)	Form creation event dispatcher.
	DoDestroy (see page 13873)	Form destruction event dispatcher.
	DoDock (see page 13874)	Docking event dispatcher.
	DoHide (see page 13874)	Form hide event dispatcher.
	DoShow (see page 13875)	Form show event dispatcher.
	GetBorderIconStyles (see page 13875)	This is GetBorderIconStyles, a member of class TCustomForm.
	GetBorderStyles (see page 13876)	This is GetBorderStyles, a member of class TCustomForm.
	GetChildren (see page 13876)	Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.
	GetClientRect (see page 13876)	Returns the value of the ClientRect property.
	GetFloating (see page 13877)	Returns the value of the Floating property.
	GetOwnerWindow (see page 13878)	This is GetOwnerWindow, a member of class TCustomForm.
	HandleCreateException (see page 13878)	Handles exceptions that are raised by the OnCreate (see page 13951) event handler.
	Loaded (see page 13880)	Performs fix-ups when the form is first loaded into memory.
	Notification (see page 13882)	Responds to notifications that objects are about to be inserted or removed.
	Paint (see page 13882)	Generates an OnPaint (see page 13975) event.
	PaintWindow (see page 13882)	Renders the image of the form.
	PaletteChanged (see page 13883)	Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.
	QueryInterface (see page 13884)	Returns a reference to a specified interface if that interface is supported by the form.
	ReadState (see page 13884)	Reads the form's published properties, stored data, and owned components from a reader object.
	RequestAlign (see page 13885)	Instructs the form's parent (if any) to reposition the form, enforcing its Align property.
	Resizing (see page 13885)	Makes any required adjustments when the form changes size.
	SetChildOrder (see page 13886)	Changes the order in which Child appears when GetChildren (see page 13876) is called.
	SetParent (see page 13887)	Sets the Parent property.
	SetParentBiDiMode (see page 13887)	Sets the ParentBiDiMode property.
	UpdateActions (see page 13891)	Updates all actions associated with the form.
	UpdateWindowState (see page 13892)	Updates the WindowState (see page 13942) property to reflect the current state of the form.
	ValidateRename (see page 13892)	Determines if a form can rename one of its owned components.
	VisibleChanging (see page 13893)	Responds just before the form changes the value of its Visible (see page 13940) property.
	WndProc (see page 13893)	Provides specific message responses for the form.

Public Methods

	Name	Description
	AfterConstruction (see page 13866)	Construction event dispatcher.
	BeforeDestruction (see page 13866)	Destruction event dispatcher.
	Close (see page 13867)	Closes the form.
	CloseQuery (see page 13868)	Close (see page 13867) attempt event dispatcher.
	DefaultHandler (see page 13870)	Provides message handling for all messages that the form does not fully process.
	DefocusControl (see page 13871)	Removes focus from a control on the form.
	Dock (see page 13872)	Handles automatic docking of the form into other windows.
	FocusControl (see page 13875)	Sets focus to a control on the form.
	GetFormImage (see page 13877)	Returns a bitmap of the form.

	Hide (see page 13879)	Hides the form.
	IsShortCut (see page 13880)	Processes shortcut keys when the form has focus.
	MakeFullyVisible (see page 13881)	Ensures that the form is fully visible on a specified monitor.
V	MouseWheelHandler (see page 13881)	Directs mouse wheel messages to the appropriate control.
	Print (see page 13883)	Prints the form.
	RecreateAsPopup (see page 13884)	This is RecreateAsPopup, a member of class TCustomForm.
	Release (see page 13885)	Destroys the form and frees its associated memory.
	SendCancelMode (see page 13886)	Cancels modes on the form.
V	SetFocus (see page 13886)	Sets focus to the form.
V	SetFocusedControl (see page 13887)	Sets focus to a control on the form.
	Show (see page 13888)	Shows the form.
V	ShowModal (see page 13890)	Shows a form as a modal dialog.
	UpdateGlassFrame (see page 13892)	This is UpdateGlassFrame, a member of class TCustomForm.
V	WantChildKey (see page 13893)	Indicates whether the form processes keyboard input for a control it owns.

72.1.5.6.1 TCustomForm.Activate Method

Activation event dispatcher.

Pascal

```
procedure Activate; dynamic;
```

C++

```
__fastcall Activate();
```

Description

Activation is the event dispatcher for form activations.

The form component calls Activate in response to the CM_ACTIVATE message, sent when a form gets the focus either because of a change in active windows within an application or because of the application becoming active. As defined in TCustomForm (see page 13849), Activate simply calls the OnActivate (see page 13943) event handler.

See Also

Deactivate (see page 13870)

OnActivate (see page 13943)

72.1.5.6.2 TCustomForm.ActiveChanged Method

Active (see page 13896) state event dispatcher.

Pascal

```
procedure ActiveChanged; dynamic;
```

C++

```
__fastcall ActiveChanged();
```

Description

ActiveChanged is the event dispatcher for changes in the form's active status. As implemented in TCustomForm (see page 13849), ActiveChanged does nothing. Override this method in a descendant class to implement event handling or other response to a change in the active form.

See Also

ActiveControl (see page 13896)

72.1.5.6.3 TCustomForm.AfterConstruction Method

Construction event dispatcher.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is the dispatcher for the event that occurs immediately after the form object's construction. As implemented in TCustomForm (see page 13849), AfterConstruction calls the OnCreate (see page 13951) event handler, provided the OldCreateOrder (see page 13926) property is false.

See Also

BeforeDestruction (see page 13866)

OnCreate (see page 13951)

OldCreateOrder (see page 13926)

72.1.5.6.4 TCustomForm.AlignControls Method

Aligns all child controls of the form that fall within a specified rectangle.

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

Use AlignControls to align all controls within an area of the form. Specify the area in which to align the controls as the value of the Rect parameter.

AlignControls uses the Align property value for each child control to determine how to align it. The AControl parameter can be nil (Delphi) or NULL (C++). If you specify a control in AControl, that control takes precedence in alignment over other, similarly aligned controls.

See Also

Align (see page 5798)

72.1.5.6.5 TCustomForm.BeforeDestruction Method

Destruction event dispatcher.

Pascal

```
procedure BeforeDestruction; override;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

BeforeDestruction is the dispatcher for the event that occurs immediately before the form object's destruction. As implemented in TCustomForm (see page 13849), BeforeDestruction calls the OnDestroy (see page 13953) event handler, provided the OldCreateOrder (see page 13926) property is false.

See Also

AfterConstruction (see page 13866)

OnDestroy (see page 13953)

72.1.5.6.6 TCustomForm.BeginAutoDrag Method

Responds when the user starts dragging the form if DragMode is dmAutomatic.

Pascal

```
procedure BeginAutoDrag; override;
```

C++

```
virtual __fastcall BeginAutoDrag();
```

Description

BeginAutoDrag suppresses the default behavior of the BeginAutoDrag method. This is because forms do not change the mouse cursor when dragged.

See Also

DragMode (see page 5819)

Floating

72.1.5.6.7 TCustomForm.ChangeScale Method

Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.

Pascal

```
procedure ChangeScale(M: Integer; D: Integer); override;
```

C++

```
virtual __fastcall ChangeScale(int M, int D);
```

Description

TCustomForm (see page 13849) overrides ChangeScale to rescale any scroll bars and child controls.

See Also

ChangeScale (see page 5743)

72.1.5.6.8 TCustomForm.Close Method

Closes the form.

Pascal

```
procedure Close;
```

C++

```
__fastcall Close();
```

Description

Call Close to close a form.

Attempts to close a form can be aborted by the CloseQuery (see page 13868) event dispatcher or the OnCloseQuery (see page 13950) event handler.

Note: When the main form of the application closes, the application terminates.

See Also

CloseQuery (see page 13868)

OnCloseQuery (see page 13950)

72.1.5.6.9 TCustomForm.CloseModal Method

Pascal

```
procedure CloseModal;
```

C++

```
__fastcall CloseModal();
```

Description

This is CloseModal, a member of class TCustomForm.

72.1.5.6.10 TCustomForm.CloseQuery Method

Close (see page 13867) attempt event dispatcher.

Pascal

```
function CloseQuery: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CloseQuery();
```

Description

CloseQuery is called automatically when an attempt is made to close the form. CloseQuery can allow the form to close by returning true, or prevent the form from closing by returning false.

As implemented in TCustomForm (see page 13849), CloseQuery polls any MDI children by calling their CloseQuery methods. If no child form aborts the close, CloseQuery then calls the OnCloseQuery (see page 13950) event handler, if it exists, to determine if the close should be allowed. If no such event handler exists, CloseQuery returns true.

See Also

Close (see page 13867)

OnCloseQuery (see page 13950)

72.1.5.6.11 TCustomForm.CreateParams Method

Initializes the window-creation parameter record when the form window is created.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 13869) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle (see page 13869). TCustomForm (see page 13849) overrides CreateParams to customize the way the form creates its Windows representation after calling the inherited method. You should never need to override this method in the context of a form class.

The fields of the Params parameter become the parameters to a call to the CreateWindowEx OS function.

See Also

CreateWindowHandle (see page 13869)

CreateWnd (see page 13869)

72.1.5.6.12 TCustomForm.CreateWindowHandle Method

Creates a form window given a set of window-creation parameters.

Pascal

```
procedure CreateWindowHandle(const Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateWindowHandle(const TCreateParams Params);
```

Description

The CreateWnd (see page 13869) method calls CreateWindowHandle to create the form window once it has been specified in the window-creation parameters. CreateWindowHandle creates the window by calling the CreateWindowEx API function, passing parameters from the record passed in the Params parameter. CreateWindowHandle removes the CS_HREDRAW and CS_VREDRAW class styles from the window class.

Params holds information needed when telling Windows to create a window handle.

See Also

CreateParams (see page 13868)

CreateWnd (see page 13869)

72.1.5.6.13 TCustomForm.CreateWnd Method

Creates the form window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called when the form is created, and whenever a change to the form requires the window to be recreated. CreateWnd calls (indirectly) the CreateParams (see page 13868) method to obtain the window creation parameters, and then

calls CreateWindowHandle (see page 13869) to direct Windows to create the underlying window for the form. It then initializes the newly-created window to reflect the property settings of the form.

See Also

CreateParams (see page 13868)

CreateWindowHandle (see page 13869)

Handle (see page 6207)

72.1.5.6.14 TCustomForm.Deactivate Method

Generates an OnDeactivate (see page 13953) event.

Pascal

```
procedure Deactivate; dynamic;
```

C++

```
__fastcall Deactivate();
```

Description

The form component calls Deactivate in response to the CM_DEACTIVATE message, sent when a form gets the focus either because of a change in active windows within an application or because of the application becoming active.

The default Deactivate method calls any event handler attached by the user to the OnDeactivate (see page 13953) event. Override this method to block the OnDeactivate (see page 13953) event or provide additional responses to the loss of activation.

See Also

Activate (see page 13865)

OnActivate (see page 13943)

72.1.5.6.15 TCustomForm.DefaultHandler Method

Provides message handling for all messages that the form does not fully process.

Pascal

```
procedure DefaultHandler(var Message); override;
```

C++

```
virtual __fastcall DefaultHandler( Message);
```

Description

Override DefaultHandler to change the default message handling for the form. This is seldom necessary because messages can be handled by creating message methods.

DefaultHandler passes any otherwise-unhandled messages to the form's window procedure by calling the CallWindowProc OS function.

Note: In Delphi code, calling inherited in a message-handling method results in a call to the ancestor's DefaultHandler method unless that ancestor specifies a handler for the message.

See Also

Dispatch

72.1.5.6.16 TCustomForm.DefineProperties Method

Designates methods for storing a form's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

DefineProperties calls the inherited method and calls the filer object's DefineProperty method for the PixelsPerInch (see page 13928),and TextHeight properties. TForm (see page 13992) descendants can also specify methods that read and write data other than properties by overriding the DefineProperties method.

See Also

DefineProperty (see page 2838)

DefineBinaryProperty (see page 2837)

72.1.5.6.17 TCustomForm.DefocusControl Method

Removes focus from a control on the form.

Pascal

```
procedure DefocusControl(Control: TWinControl; Removing: Boolean);
```

C++

```
__fastcall DefocusControl(TWinControl * Control, Boolean Removing);
```

Description

Use DefocusControl to remove focus from a control on the form. This method is used internally by the VCL.

Control is the control from which to remove focus. If Control is the form's active control, DefocusControl sets the form's ActiveControl (see page 13896) property to nil (Delphi) or NULL (C++).

Removing specifies whether to set focus to the control's parent.

See Also

SetFocusedControl (see page 13887)

ActiveControl (see page 13896)

SetFocus (see page 13886)

FocusedControl

72.1.5.6.18 TCustomForm.DestroyHandle Method

Destroys the control's window without destroying the control.

Pascal

```
procedure DestroyHandle; override;
```

C++

```
virtual __fastcall DestroyHandle();
```

Description

Call DestroyHandle to dispose of the window, but leave the control intact. The control can later recreate the window if needed. DestroyHandle is the converse operation to CreateHandle. Applications should call the high-level CreateHandle and DestroyHandle methods, rather than the lower-level methods of CreateWnd and DestroyWnd, whenever possible.

If the control has TWinControl objects as child controls, DestroyHandle calls each of their DestroyHandle methods before calling DestroyWnd to destroy its own handle.

See Also

CreateHandle (see page 6135)

DestroyWnd (see page 6141)

72.1.5.6.19 TCustomForm.DestroyWindowHandle Method

Destroys the window created in the CreateWindowHandle (see page 13869) method.

Pascal

```
procedure DestroyWindowHandle; override;
```

C++

```
virtual __fastcall DestroyWindowHandle();
```

Description

DestroyWindowHandle is called automatically to destroy the window associated with a form before destroying the form.

See Also

CreateWindowHandle (see page 13869)

DestroyWnd (see page 6141)

Destroy (see page 13862)

72.1.5.6.20 TCustomForm.Dock Method

Handles automatic docking of the form into other windows.

Pascal

```
procedure Dock(NewDockSite: TWinControl; ARect: TRect); override;
```

C++

```
virtual __fastcall Dock(TWinControl * NewDockSite, TRect ARect);
```

Description

Do not call Dock in application code. It is called automatically when the form is released on a docking site that will accept it. To dock a form programmatically, use the ManualDock method instead.

See Also

DragMode (see page 5819)

ManualDock (see page 5771)

72.1.5.6.21 TCustomForm.DoClose Method

Form closing event dispatcher.

Pascal

```
procedure DoClose(var Action: TCloseAction); dynamic;
```

C++

```
__fastcall DoClose(TCloseAction Action);
```

Description

DoClose is called automatically when the form is about to close. DoClose can alter closing behavior by changing the Action (see page 13895) parameter. For more information, refer to the OnClose (see page 13949) event handler.

As implemented in TCustomForm (see page 13849), DoClose simply calls the OnClose (see page 13949) event handler.

See Also

OnClose (see page 13949)

72.1.5.6.22 TCustomForm.DoCreate Method

Form creation event dispatcher.

Pascal

```
procedure DoCreate; virtual;
```

C++

```
virtual __fastcall DoCreate();
```

Description

DoCreate is called automatically after form construction is complete. The precise way DoCreate is called can be modified by setting the OldCreateOrder (see page 13926) property.

As implemented in TCustomForm (see page 13849), DoCreate simply calls the OnCreate (see page 13951) event handler.

See Also

OnCreate (see page 13951)

FormState (see page 13911)

Visible (see page 5847)

OldCreateOrder (see page 13926)

72.1.5.6.23 TCustomForm.DoDestroy Method

Form destruction event dispatcher.

Pascal

```
procedure DoDestroy; virtual;
```

C++

```
virtual __fastcall DoDestroy();
```

Description

DoDestroy is called automatically when the TCustomForm (see page 13849) object is about to be deallocated. The precise

way DoDestroy is called can be modified by setting the OldCreateOrder (see page 13926) property.

As implemented in TCustomForm (see page 13849), DoDestroy simply calls the OnDestroy (see page 13953) event handler.

See Also

OnDestroy (see page 13953)

OldCreateOrder (see page 13926)

72.1.5.6.24 TCustomForm.DoDock Method

Docking event dispatcher.

Pascal

```
procedure DoDock(NewDockSite: TWinControl; var ARect: TRect); override;
```

C++

```
virtual __fastcall DoDock(TWinControl * NewDockSite, TRect ARect);
```

Description

DoDock supplements the inherited DoDock method by updating form borders.

See Also

DoDock (see page 5750)

Floating

Dock (see page 13872)

ManualDock (see page 5771)

ManualFloat (see page 5772)

72.1.5.6.25 TCustomForm.DoHide Method

Form hide event dispatcher.

Pascal

```
procedure DoHide; dynamic;
```

C++

```
__fastcall DoHide();
```

Description

DoHide is called automatically when the form is hidden.

As implemented in TCustomForm (see page 13849), DoHide calls the OnHide (see page 13960) event handler.

See Also

OnHide (see page 13960)

DoShow (see page 13875)

Hide (see page 13879)

72.1.5.6.26 TCustomForm.DoShow Method

Form show event dispatcher.

Pascal

```
procedure DoShow; dynamic;
```

C++

```
__fastcall DoShow();
```

Description

DoShow is called automatically when the application is about to show the form. As implemented in TCustomForm (see page 13849), DoShow simply calls the OnShow (see page 13977) event handler.

See Also

DoHide (see page 13874)

OnShow (see page 13977)

72.1.5.6.27 TCustomForm.FocusControl Method

Sets focus to a control on the form.

Pascal

```
procedure FocusControl(Control: TWinControl);
```

C++

```
__fastcall FocusControl(TWinControl * Control);
```

Description

Use FocusControl to set focus to a control on the form.

Control is a TWinControl object encapsulating the control on which to set focus. FocusControl sets the ActiveControl (see page 13896) property.

See Also

TWinControl (see page 6112)

ActiveControl (see page 13896)

72.1.5.6.28 TCustomForm.GetBorderIconStyles Method

Pascal

```
procedure GetBorderIconStyles(var Style: Cardinal; var ExStyle: Cardinal); dynamic;
```

C++

```
__fastcall GetBorderIconStyles(unsigned Style, unsigned ExStyle);
```

Description

This is GetBorderIconStyles, a member of class TCustomForm.

72.1.5.6.29 TCustomForm.GetBorderStyles Method

Pascal

```
procedure GetBorderStyles(var Style: Cardinal; var ExStyle: Cardinal; var ClassStyle:
Cardinal); dynamic;
```

C++

```
__fastcall GetBorderStyles(unsigned Style, unsigned ExStyle, unsigned ClassStyle);
```

Description

This is GetBorderStyles, a member of class TCustomForm.

72.1.5.6.30 TCustomForm.GetChildren Method

Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

The component streaming system calls GetChildren to request that the form execute a callback for every child component in the form. The callback must execute in creation order (the order the child components appear in the form file). A child component is defined as any component that returns this form from its GetParentComponent method.

Proc is the callback to execute for every child component.

Root specifies the top-level component (such as a form or data module) that is currently being loaded or saved.

In TCustomForm (see page 13849), GetChildren first calls the inherited method, which executes Proc for every control that lists the form as its Parent. Then, if the Root parameter identifies this form, GetChildren executes Proc for every child component that does not have a Parent.

See Also

GetParentComponent

72.1.5.6.31 TCustomForm.GetClientRect Method

Returns the value of the ClientRect property.

Pascal

```
function GetClientRect: TRect; override;
```

C++

```
virtual __fastcall TRect GetClientRect();
```

Description

GetClientRect is the protected access function for the ClientRect property. It returns the dimensions of the form window, indented by any scroll bars and menus.

Override this method to change the way the client area of the form is defined.

See Also

GetClientOrigin (see page 6153)

ClientRect

72.1.5.6.32 TCustomForm.GetFloating Method

Returns the value of the Floating property.

Pascal

```
function GetFloating: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetFloating();
```

Description

GetFloating is the protected access function for the Floating property. Override this method in descendants that implement floating windows in some class-specific way.

See Also

Floating

72.1.5.6.33 TCustomForm.GetFormImage Method

Returns a bitmap of the form.

Pascal

```
function GetFormImage: TBitmap;
```

C++

```
__fastcall TBitmap GetFormImage();
```

Description

Use the GetFormImage method to obtain a bitmap of the form.

C++ Examples:

```
/*
This example uses an image, a button, and a shape component
on a form. When the user clicks the button, an image of the
form is stored in the FormImage variable and copied to the
Clipboard. The image of the form is then copied back to
the image component, producing an interesting result,
especially if the button is clicked multiple times.  Add
ExtCtrls, StdCtrls, and Clipbrd to the uses clause.
}
*/

#include <Clipbrd.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *FormImage = GetFormImage();
  try
  {
    Clipboard()->Assign(FormImage);
    Image1->Picture->Assign(Clipboard());
  }
```

```
    __finally
    {
      delete FormImage;
    }
  }

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    Shape1->Shape = stEllipse;
    Shape1->Brush->Color = clLime;
    Image1->Stretch = true;
  }
```

Delphi Examples:

```
{
This example uses an image, a button, and a shape component
on a form. When the user clicks the button, an image of the
form is stored in the FormImage variable and copied to the
Clipboard. The image of the form is then copied back to
the image component, producing an interesting result,
especially if the button is clicked multiple times.  Add
ExtCtrls, StdCtrls, and Clipbrd to the uses clause.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FormImage: TBitmap;
begin
  FormImage := GetFormImage;
  try
    Clipboard.Assign(FormImage);
    Image1.Picture.Assign(Clipboard);
  finally
    FormImage.Free;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Shape1.Shape := stEllipse;
  Shape1.Brush.Color := clLime;
  Image1.Stretch := True;
end;
```

See Also

Print (see page 13883)

72.1.5.6.34 TCustomForm.GetOwnerWindow Method

Pascal

```
function GetOwnerWindow: HWND; dynamic;
```

C++

```
__fastcall HWND GetOwnerWindow();
```

Description

This is GetOwnerWindow, a member of class TCustomForm.

72.1.5.6.35 TCustomForm.HandleCreateException Method

Handles exceptions that are raised by the OnCreate (see page 13951) event handler.

Pascal

```
function HandleCreateException: Boolean; dynamic;
```

C++

```
__fastcall Boolean HandleCreateException();
```

Description

The form calls HandleCreateException internally if the OnCreate (see page 13951) event handler raises an exception. HandleCreateException forwards the exception on to the application object, which generates an OnException event. If there is no OnException event handler, the application displays an exception message box.

HandleCreateException returns true if it handles the exception, false otherwise.

See Also

OnCreate (see page 13951)

OnException (see page 13728)

ShowException (see page 13698)

72.1.5.6.36 TCustomForm.Hide Method

Hides the form.

Pascal

```
procedure Hide;
```

C++

```
__fastcall Hide();
```

Description

Hide sets the Visible (see page 13940) property to false.

C++ Examples:

```
/*
This code uses a button, a timer and a maskedit on a
form. When the user clicks the button, the form
disappears for the period of time specified in the
Interval property of the timer control, then the form
reappears:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Timer1->Enabled = true;
  Hide();
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Visible = true;
  Timer1->Enabled = false;
}
```

Delphi Examples:

```
{
This code uses a button, a timer and a maskedit on a
form. When the user clicks the button, the form
disappears for the period of time specified in the
Interval property of the timer control, then the form
```

```
reappears:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Timer1.Interval := StrtoInt(MaskEdit1.Text);
  Timer1.Enabled := True;
  Hide;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Visible := True;
  Timer1.Enabled := False;
end;
```

See Also

Visible (see page 13940)

72.1.5.6.37 TCustomForm.IsShortCut Method

Processes shortcut keys when the form has focus.

Pascal

```
function IsShortCut(var Message: TWMKey): Boolean; dynamic;
```

C++

```
__fastcall Boolean IsShortCut(TWMKey Message);
```

Description

IsShortCut is called automatically when the user presses a key while the form has focus. It allows the form to dispatch shortcut keys and bypass the standard keystroke processing (OnKeyDown (see page 13962), OnKeyPress (see page 13964), and OnKeyUp (see page 13966) events).

The Message parameter encapsulates the Windows message describing the keypress. If a shortcut for the key exists, IsShortCut executes the appropriate command and returns true. Otherwise IsShortCut returns false.

IsShortCut generates an OnShortCut (see page 13977) event to allow the developer to insert shortcuts for the form. If the keystroke is not handled in an OnShortCut (see page 13977) event handler, IsShortCut checks for any shortcuts associated with the form's menus and actions.

See Also

OnShortCut (see page 13977)

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

72.1.5.6.38 TCustomForm.Loaded Method

Performs fix-ups when the form is first loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Loaded overrides the inherited method to give focus to the ActiveControl (see page 13896) if it is assigned. Before assigning focus, Loaded calls the inherited method to perform any lower-level initialization.

See Also

ActiveControl (see page 13896)

Loaded (see page 2782)

72.1.5.6.39 TCustomForm.MakeFullyVisible Method

Ensures that the form is fully visible on a specified monitor.

Pascal

```
procedure MakeFullyVisible(AMonitor: TMonitor = nil);
```

C++

```
__fastcall MakeFullyVisible(TMonitor AMonitor = nil);
```

Description

Call MakeFullyVisible to ensure that the form does not appear split over more than one monitor in a multi-monitor application.

AMonitor is the monitor on which the form should appear. If AMonitor is nil (Delphi) or NULL (C++), MakeFullyVisible uses the Monitor (see page 13926) property.

MakeFullyVisible checks whether the form fits entirely on the specified monitor. If not, it repositions the form so that it fits, if possible.

See Also

DefaultMonitor (see page 13909)

Monitor (see page 13926)

72.1.5.6.40 TCustomForm.MouseWheelHandler Method

Directs mouse wheel messages to the appropriate control.

Pascal

```
procedure MouseWheelHandler(var Message: TMessage); override;
```

C++

```
virtual __fastcall MouseWheelHandler(TMessage Message);
```

Description

MouseWheelHandler is called automatically when the form or any windowed control contained in the form receives a message from a mouse wheel. The Message parameter encapsulates the message.

As implemented in TCustomForm (see page 13849), MouseWheelHandler performs no special processing, but simply redirects the message to the active control by calling its Perform method. By default, the Perform method generates the appropriate mouse wheel events.

See Also

ActiveControl (see page 13896)

Perform (see page 5776)

OnMouseWheel (see page 5869)

OnMouseWheelUp (see page 5870)

OnMouseWheelDown (see page 5869)

72.1.5.6.41 TCustomForm.Notification Method

Responds to notifications that objects are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, forms pass along the notification to their owned components, if any. In addition, Notification makes internal adjustments if the main menu is added or deleted.

A form can, if needed, override Notification to make other adjustments when components are inserted or removed. In particular, if a form has object fields or properties that contains references to other components, it might check the notifications of component removals and invalidate those references as needed.

See Also

InsertComponent (see page 2780)

72.1.5.6.42 TCustomForm.Paint Method

Generates an OnPaint (see page 13975) event.

Pascal

```
procedure Paint; dynamic;
```

C++

```
__fastcall Paint();
```

Description

The form component calls Paint in response to the WM_PAINT message (sent when the window needs to be painted) before rendering all controls on the form.

The default Paint method calls any event handler attached by the user to the OnPaint (see page 13975) event. Override Paint to block the OnPaint (see page 13975) event or to draw additional features before or after the OnPaint (see page 13975) event.

See Also

OnPaint (see page 13975)

72.1.5.6.43 TCustomForm.PaintWindow Method

Renders the image of the form.

Pascal

```
procedure PaintWindow(DC: HDC); override;
```

C++

```
virtual __fastcall PaintWindow(HDC DC);
```

Description

Applications can't call this protected method. It is called when the form needs to be painted.

PaintWindow locks the canvas to prevent threading issues, assigns the DC parameter as the Handle of the form's canvas, and then calls the Paint (see page 13882) method to render the image of the form.

See Also

Paint (see page 13882)

Canvas (see page 13900)

Handle (see page 14489)

Lock (see page 14463)

72.1.5.6.44 TCustomForm.PaletteChanged Method

Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.

Pascal

```
function PaletteChanged(Foreground: Boolean): Boolean; override;
```

C++

```
virtual __fastcall Boolean PaletteChanged(Boolean Foreground);
```

Description

PaletteChanged is called automatically when the system palette changes. The Foreground parameter indicates whether the palette realization should treat this control's palette as being in the foreground or the background. The return value from PaletteChanged indicates whether the form actually realized its palette, returning true only if the form has an associated palette.

Most applications should not need to alter the default behavior of PaletteChanged.

See Also

PaletteChanged (see page 6170)

72.1.5.6.45 TCustomForm.Print Method

Prints the form.

Pascal

```
procedure Print;
```

C++

```
__fastcall Print();
```

Description

Call Print to print the form. Print uses the GetFormImage (see page 13877) method to obtain a bitmap of the form and draws that to the printer's HDC.

See Also

GetFormImage (see page 13877)

72.1.5.6.46 TCustomForm.QueryInterface Method

Returns a reference to a specified interface if that interface is supported by the form.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override;
```

C++

```
virtual __fastcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

For forms that support interfaces, QueryInterface calls the QueryInterface method of the interface supported by the form. QueryInterface returns a reference to the interface specified by the IID parameter, as the Obj parameter. If the form does not support the interface, the Obj parameter is nil (Delphi) or NULL (C++).

QueryInterface returns S_OK if the Obj parameter is successfully assigned.

See Also

_AddRef (see page 2769)

_Release (see page 2769)

Invoke (see page 2781)

72.1.5.6.47 TCustomForm.ReadState Method

Reads the form's published properties, stored data, and owned components from a reader object.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

ReadState is part of a sequence of calls used by the VCL streaming system that loads and saves VCL components. It reads the values of all the component's published properties, stored data, and owned components from the reader object passed in Reader. Do not call ReadState directly.

Any descendant classes overriding ReadState should end with a call to the inherited ReadState method.

See Also

TReader (see page 2933)

72.1.5.6.48 TCustomForm.RecreateAsPopup Method

Pascal

```
procedure RecreateAsPopup(AWindowHandle: HWND);
```

C++

```
__fastcall RecreateAsPopup(HWND AWindowHandle);
```

Description

This is RecreateAsPopup, a member of class TCustomForm.

72.1.5.6.49 TCustomForm.Release Method

Destroys the form and frees its associated memory.

Pascal

```
procedure Release;
```

C++

```
__fastcall Release();
```

Description

Use Release to destroy the form and free its associated memory.

Release does not destroy the form until all event handlers of the form and event handlers of components on the form have finished executing. Release also guarantees that all messages in the form's event queue are processed before the form is released. Any event handlers for the form or its children should use Release instead of Free (Delphi) or delete (C++). Failing to do so can cause a memory access error.

Note: Release returns immediately to the caller. It does not wait for the form to be freed before returning.

See Also

Free (see page 22300)

Destroy (see page 13862)

72.1.5.6.50 TCustomForm.RequestAlign Method

Instructs the form's parent (if any) to reposition the form, enforcing its Align property.

Pascal

```
procedure RequestAlign; override;
```

C++

```
virtual __fastcall RequestAlign();
```

Description

RequestAlign is called internally when changes are made to the size or position of the form, so that it remains properly aligned within a dock site or properly positioned on the screen.

See Also

Align (see page 5798)

Floating

72.1.5.6.51 TCustomForm.Resizing Method

Makes any required adjustments when the form changes size.

Pascal

```
procedure Resizing(State: TWindowState); override;
```

C++

```
virtual __fastcall Resizing(TWindowState State);
```

Description

This protected method is called automatically when the form is resized, including when it is maximized, minimized, or restored to normal size. Resizing adjusts the size and position of any child controls that are aligned to the form.

The State parameter indicates the state of the form after the resize.

72.1.5.6.52 TCustomForm.SendCancelMode Method

Cancels modes on the form.

Pascal

```
procedure SendCancelMode(Sender: TControl);
```

C++

```
__fastcall SendCancelMode(TControl * Sender);
```

Description

Use SendCancelMode to release the mouse capture and cancel scroll bar input and menu processing on the form.

See Also

ActiveControl (see page 13896)

72.1.5.6.53 TCustomForm.SetChildOrder Method

Changes the order in which Child appears when GetChildren (see page 13876) is called.

Pascal

```
procedure SetChildOrder(Child: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Child, int Order);
```

Description

Use SetChildOrder to change the order in which child components are loaded and saved. The Child moves as if it were an item in a list object: items previously below the child's old position move up, and those below the new position move down.

See Also

GetChildren (see page 13876)

72.1.5.6.54 TCustomForm.SetFocus Method

Sets focus to the form.

Pascal

```
procedure SetFocus; override;
```

C++

```
virtual __fastcall SetFocus();
```

Description

Use SetFocus to give the form input focus. SetFocus calls the SetFocus method of the active control, if one exists. To determine whether the form has focus, use the Active (see page 13896) property.

See Also

Active (see page 13896)

SetFocusedControl (see page 13887)

SetFocus (see page 6179)

72.1.5.6.55 TCustomForm.SetFocusedControl Method

Sets focus to a control on the form.

Pascal

```
function SetFocusedControl(Control: TWinControl): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean SetFocusedControl(TWinControl * Control);
```

Description

Use SetFocusedControl to give a control on the form input focus. SetFocusedControl returns false if the Control specified by the Control parameter was already in the process of receiving focus, true otherwise.

Note: A return value of true does not indicate the control has successfully received input focus. If the Control can't have focus (for example, if it is not visible), SetFocusedControl will still return true, indicating that an attempt was made.

See Also

Active (see page 13896)

SetFocus (see page 13886)

DefocusControl (see page 13871)

72.1.5.6.56 TCustomForm.SetParent Method

Sets the Parent property.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. It overrides the inherited method to regenerate the form's window when the Parent property is changed to nil (Delphi) or NULL (C++).

See Also

Parent

72.1.5.6.57 TCustomForm.SetParentBiDiMode Method

Sets the ParentBiDiMode property.

Pascal

```
procedure SetParentBiDiMode(Value: Boolean); override;
```

C++

```
virtual __fastcall SetParentBiDiMode(Boolean Value);
```

Description

SetParentBiDiMode is the protected implementation of the ParentBiDiMode property. It overrides the inherited method to obtain the BiDiMode property of the application if the form does not have a parent.

See Also

ParentBiDiMode

72.1.5.6.58 TCustomForm.Show Method

Shows the form.

Pascal

```
procedure Show;
```

C++

```
__fastcall Show();
```

Description

Use Show to set the form's Visible (see page 13940) property to true and to bring the form to the front of other forms on the screen.

C++ Examples:

```
/*
StepBy, CreateForm, Run, Show, Update, Initialize example
This example shows progress of loading forms as an application starts up. The code example is
placed in the project source (*.dpr) file. To see project source, select the View | Project
Source menu item.
You will need to set up your project with the following steps before using the code example:
o          Add four additional forms to a default project.
o          Place a TProgressBar on Form5
o          Take the Project|Options|Forms menu option and place Form5 on the available forms
list.
o          Change the code of your project (*.dpr) file to look like the example.
*/
#include <vcl\vcl.h>
#include "Unit5.h"
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("TAppCreateForm.cpp", Form1);
USERES("Project1.res");
USEFORM("Unit2.cpp", Form2);
USEFORM("Unit3.cpp", Form3);
USEFORM("Unit4.cpp", Form4);
USEFORM("Unit5.cpp", Form5);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
  try
  {
    Application->Initialize();
    SetApplicationMainFormOnTaskBar(Application, true);
    Application->CreateForm(__classid(TForm5), &Form5);
    Form5->ProgressBar1->Max = 100;
    Form5->Show();   // show a splash screen contain ProgressBar control
    Form5->Update(); // force display of Form5
    Application->CreateForm(__classid(TForm1), &Form1);
```

```
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form1 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm2), &Form2);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form2 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm3), &Form3);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form3 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
    Application->CreateForm(__classid(TForm4), &Form4);
    Form5->ProgressBar1->StepBy(25);
    Form5->Label1->Caption =  "Form4 loaded successfully.";
    Form5->Update(); // force display of Form5
    Sleep(3000);
//  delete Form5;
    Application->Run();
  }
  catch (Exception &exception)
  {
    Application->ShowException(&exception);
  }
  return 0;
}
```

Delphi Examples:

```
{
This example shows progress of loading forms as an
application starts up. The code example is placed in the
project source (*.dpr) file.  To see project source, select
the View | Project Source menu item.
You will need to set up your project with the following steps
before using the code example:
    Add four additional forms to a default project.
    Place a TProgressBar on Form5
    Take the Project|Options|Forms menu option and place
          Form5 on the available forms list.
    Change the code of your project (*.dpr) file to look
          like the example.
}
begin
  Application.Initialize;
  with TForm5.Create(nil) do
  try
    Application.MainFormOnTaskbar := True;
    ProgressBar1.Max := 100;
    Show;  // show a splash screen contain ProgressBar control
    Update; // force display of Form5
    Application.CreateForm(TForm1, Form1);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form1 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm2, Form2);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form2 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
    Application.CreateForm(TForm3, Form3);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form3 loaded successfully.';
    Update; // force display of Form5
```

```
    Sleep(3000);
    Application.CreateForm(TForm4, Form4);
    ProgressBar1.StepBy(25);
    Label1.Caption :=  'Form4 loaded successfully.';
    Update; // force display of Form5
    Sleep(3000);
  finally
    Free;
  end;
  Application.Run;
```

See Also

BringToFront (see page 5740)

Visible (see page 13940)

72.1.5.6.59 TCustomForm.ShowModal Method

Shows a form as a modal dialog.

Pascal

```
function ShowModal: Integer; virtual;
```

C++

```
virtual __fastcall int ShowModal();
```

Description

Use ShowModal to show a form as a modal form. A modal form is one where the application can't continue to run until the form is closed. Thus, ShowModal does not return until the form closes. When the form closes, it returns the value of the ModalResult (see page 13924) property.

To close a modal form, set its ModalResult (see page 13924) property to a nonzero value.

Note: If the form contains buttons with a ModalResult (see page 13924) property set to a value other than mrNone, the form automatically closes when the user clicks one of those buttons and returns the ModalResult (see page 13924) value as the return value of ShowModal.

You can check the return value against common return values using the global IsAbortResult, IsAnAllResult, IsNegativeResult, or IsPositiveResult functions.

C++ Examples:

```
/*
This code brings up the modal dialog from Unit2 when a
button is clicked.  It causes a Beep if the OK button is
clicked. Implement the modal form in Unit2 and then put
Unit2 in the uses clause for this module.
*/
#include "Unit2.h"
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (MyDialogBox1->ShowModal() == mrOk)
    MessageBeep(0);
}
/*
The following methods are used for buttons in a form that is
used as a modal dialog box. The methods cause the dialog box
to terminate when the user clicks either the OK or Cancel
button, returning mrOk or mrCancel from ShowModal,
respectively. You could also set the ModalResult value to
mrOk for the OK button and mrCancel for the Cancel button to
accomplish the same thing. When the user clicks either
```

```
button, the dialog box closes.
*/
void __fastcall TMyDialogBox1::OKButtonClick(TObject *Sender)
{
  ModalResult = mrOk;
}

void __fastcall TMyDialogBox1::CancelButtonClick(TObject *Sender)
{
  ModalResult = mrCancel;
}
```

Delphi Examples:

```
{
This code brings up the modal dialog from Unit2 when a
button is clicked.  It causes a Beep if the OK button is
clicked. Implement the modal form in Unit2 and then put
Unit2 in the uses clause for this module.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if MyDialogBox1.ShowModal = mrOK then
    Beep;
end;
{
The following methods are used for buttons in a form that is
used as a modal dialog box. The methods cause the dialog box
to terminate when the user clicks either the OK or Cancel
button, returning mrOk or mrCancel from ShowModal,
respectively. You could also set the ModalResult value to
mrOk for the OK button and mrCancel for the Cancel button to
accomplish the same thing. When the user clicks either
button, the dialog box closes.
}
procedure TMyDialogBox1.CancelButtonClick(Sender: TObject);
begin
  ModalResult := mrCancel;
end;

procedure TMyDialogBox1.OKButtonClick(Sender: TObject);
begin
  ModalResult := mrOK;
end;
```

See Also

ModalResult (see page 13924)

ModalResult (see page 20937)

IsAnALlResult

IsNegativeResult (see page 6260)

IsPositiveResult (see page 6260)

StripAllFromResult (see page 6265)

IsAbortResult (see page 6258)

72.1.5.6.60 TCustomForm.UpdateActions Method

Updates all actions associated with the form.

Pascal

```
procedure UpdateActions; virtual;
```

C++

```
virtual __fastcall UpdateActions();
```

Description

UpdateActions is called automatically when the application is idle to allow all components in the form an opportunity to update any actions for which the component is a target. This allows actions to be checked, grayed, and so on to reflect the current state of their targets.

UpdateActions updates all actions associated with the form, its menus, and the components contained in the form.

See Also

UpdateAction

InitiateAction (see page 5769)

72.1.5.6.61 TCustomForm.UpdateGlassFrame Method

Pascal

```
procedure UpdateGlassFrame(Sender: TObject);
```

C++

```
__fastcall UpdateGlassFrame(TObject * Sender);
```

Description

This is UpdateGlassFrame, a member of class TCustomForm.

72.1.5.6.62 TCustomForm.UpdateWindowState Method

Updates the WindowState (see page 13942) property to reflect the current state of the form.

Pascal

```
procedure UpdateWindowState;
```

C++

```
__fastcall UpdateWindowState();
```

Description

UpdateWindowState ensures that the WindowState (see page 13942) property reflects the current state of the form.

See Also

WindowState (see page 13942)

72.1.5.6.63 TCustomForm.ValidateRename Method

Determines if a form can rename one of its owned components.

Pascal

```
procedure ValidateRename(AComponent: TComponent; const CurName: string; const NewName:
string); override;
```

C++

```
virtual __fastcall ValidateRename(TComponent * AComponent, const AnsiString CurName, const
AnsiString NewName);
```

Description

The ValidateRename method checks to see if a form can rename one of its owned components, passed in AComponent, from its current name (CurName) to the string passed in NewName. If AComponent is nil (Delphi) or NULL (C++) or NewName is already the name of a component in the form's list, ValidateRename raises an EComponentError exception.

See Also

FindComponent (see page 2772)

72.1.5.6.64 TCustomForm.VisibleChanging Method

Responds just before the form changes the value of its Visible (see page 13940) property.

Pascal

```
procedure VisibleChanging; override;
```

C++

```
virtual __fastcall VisibleChanging();
```

Description

This protected method raises an exception if, for some reason, the form should not change its visible state.

See Also

Visible (see page 13940)

72.1.5.6.65 TCustomForm.WantChildKey Method

Indicates whether the form processes keyboard input for a control it owns.

Pascal

```
function WantChildKey(Child: TControl; var Message: TMessage): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean WantChildKey(TControl * Child, TMessage Message);
```

Description

WantChildKey is called by any control that the form owns when the control receives keyboard input. If WantChildKey returns true, the control performs no further processing in response to the keyboard input, instead deferring to its parent form to respond to the keystroke. In its default implementation, WantChildKey always returns false; it is overridden in TActiveForm to implement certain ActiveX functionality.

72.1.5.6.66 TCustomForm.WndProc Method

Provides specific message responses for the form.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc is the first method that receives messages for a form. After calling the inherited method, WndProc performs some window activation, focus, and position message handling necessary to keep in sync with Windows. Override WndProc to change how the form responds to Windows messages.

See Also

WndProc (see page 6187)

72.1.5.7 TCustomForm Properties

The properties of the TCustomForm class are listed here.

Protected Properties

	Name	Description
R	ActiveMDIChild (see page 13897)	Specifies the MDI child that has focus.
	AlphaBlend (see page 13898)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 13898)	Specifies the degree of translucency on a translucent form.
	AutoScroll (see page 13899)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	BorderIcons (see page 13899)	Specifies which icons appear on the title bar of the form.
R	ClientHandle (see page 13904)	Provides access to the handle of the form's internal MDI client window.
W	ClientHeight (see page 13905)	Specifies the height (in pixels) of the form's client area.
W	ClientWidth (see page 13905)	Specifies the width (in pixels) of the form's client area.
	Ctl3D (see page 13909)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultMonitor (see page 13909)	Specifies the monitor on which the form appears.
	FormStyle (see page 13912)	Determines the form's style.
	Height (see page 13915)	Specifies the vertical size of the control in pixels.
	HorzScrollBar (see page 13916)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 13917)	Specifies the icon that appears when the form is minimized.
R	MDIChildCount (see page 13921)	Specifies the number of open MDI child forms.
R	MDIChildren (see page 13922)	Provides indexed access to all MDI child forms.
	ObjectMenuItem (see page 13926)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 13926)	Specifies when OnCreate (see page 13951) and OnDestroy (see page 13953) events occur.
	ParentFont (see page 13927)	Determines where a control looks for its font information.
	PixelsPerInch (see page 13928)	Represents the proportion of the font on the system on which the form was designed.
R	PopupChildren (see page 13928)	This is PopupChildren, a member of class TCustomForm.
	PopupMenu (see page 13928)	Identifies the pop-up menu associated with the control.
	Position (see page 13933)	Represents the size and placement of the form.
	PrintScale (see page 13934)	Represents the proportions of a printed form.
	Scaled (see page 13935)	Specifies whether the form is sized according to the value of the PixelsPerInch (see page 13928) property.
	TileMode (see page 13936)	Represents how MDI child forms are arranged when the Tile method is called.
	TransparentColor (see page 13938)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 13939)	Indicates the color on the form that appears transparent when TransparentColor (see page 13938) is true.
	VertScrollBar (see page 13939)	Represents the vertical scroll bar for the scrolling windowed control.
W	Visible (see page 13940)	Indicates whether the form is visible.
	Width (see page 13941)	Specifies the horizontal size of the control or form in pixels.
	WindowMenu (see page 13941)	Specifies the Window menu for an MDI parent form.

Public Properties

	Name	Description
	Action (see page 13895)	Designates the action associated with the control.
R	Active (see page 13896)	Specifies whether the form has focus.
	ActiveControl (see page 13896)	Specifies the control that has focus on the form.
	ActiveOleControl (see page 13897)	Specifies the OLE control on the form that reacts to changes in focus.
	BorderStyle (see page 13900)	Specifies the appearance and behavior of the form border.
R	Canvas (see page 13900)	Provides access to the drawing area of the form.
	Caption (see page 13902)	Specifies a text string that identifies the control to the user.
	Color (see page 13908)	Specifies the background color of the control.
	Designer (see page 13910)	Specifies the designer interface for the form.
	DropTarget (see page 13910)	Specifies whether the form is the target of a drag-and-drop operation.
	Font (see page 13910)	Controls the attributes of text written on or in the control.
R	FormState (see page 13911)	Indicates transitional state information about the form.
	GlassFrame (see page 13914)	This is GlassFrame, a member of class TCustomForm.
	HelpFile (see page 13916)	Specifies the name of the file the form uses to display Help.
	KeyPreview (see page 13918)	Specifies whether the form should receive keyboard events before the active control.
	Menu (see page 13923)	Specifies the form's main menu.
	ModalResult (see page 13924)	Represents the return value of a form that is used as a modal dialog.
R	Monitor (see page 13926)	Provides access to the monitor on which the form appears.
	OleFormObject (see page 13927)	Specifies the IOleForm (see page 14340) interface for an in-place OLE object contained in the form.
	PopupMode (see page 13933)	This is PopupMode, a member of class TCustomForm.
	PopupParent (see page 13933)	Sets an order for stacked forms that users cannot change.
	ScreenSnap (see page 13935)	Specifies whether form snaps to edge of screen.
	SnapBuffer (see page 13936)	Specifies distance for screen snap.
	WindowState (see page 13942)	Represents how the form appears on the screen.

Published Properties

	Name	Description
	Left (see page 13920)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Top (see page 13937)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.

72.1.5.7.1 TCustomForm.Action Property

Designates the action associated with the control.

Pascal

```
property Action: TBasicAction;
```

C++

```
__property TBasicAction Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

72.1.5.7.2 TCustomForm.Active Property

Specifies whether the form has focus.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Use Active to determine whether the form has focus. The Active Form receives all keyboard input. If it has a title bar, it is drawn using the active colors specified in the Control panel.

If Active is true , the form has focus; if Active is false, the form does not have focus.

See Also

SetFocus (see page 13886)

OnActivate (see page 13943)

OnDeactivate (see page 13953)

ActiveControl (see page 13896)

72.1.5.7.3 TCustomForm.ActiveControl Property

Specifies the control that has focus on the form.

Pascal

```
property ActiveControl: TWinControl;
```

C++

```
__property TWinControl * ActiveControl;
```

Description

Use ActiveControl to get or set the control that has focus on the form. Only one control can have focus at a given time in an application.

If the form does not have focus, ActiveControl is the control on the form that will receive focus when the form receives focus.

Note: When focus shifts to another control, ActiveControl is updated before the OnExit event occurs.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Timer1->Interval = 100;
  if (ActiveControl)
    ActiveControl->Left = ActiveControl->Left + 1;
}
```

Delphi Examples:

```
{
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  if ActiveControl <> nil then
    ActiveControl.Left := ActiveControl.Left + 1;
end;
```

See Also

ActiveOleControl (see page 13897)

OnExit (see page 6221)

OnEnter (see page 6219)

Active (see page 13896)

72.1.5.7.4 TCustomForm.ActiveMDIChild Property

Specifies the MDI child that has focus.

Pascal

```
property ActiveMDIChild: TForm;
```

C++

```
__property TForm * ActiveMDIChild;
```

Description

Use ActiveMDIChild to get the MDI child that has focus.

If the form is not an MDI parent (that is, if the FormStyle (see page 13912) property of the form is not fsMDIForm), ActiveMDIChild returns nil (Delphi) or NULL (C++).

See Also

FormStyle (see page 13912)

MDIChildCount (see page 13921)

72.1.5.7.5 TCustomForm.ActiveOleControl Property

Specifies the OLE control on the form that reacts to changes in focus.

Pascal

```
property ActiveOleControl: TWinControl;
```

C++

```
__property TWinControl * ActiveOleControl;
```

Description

Use ActiveOLEControl to get or set an OLE control on the form that reacts to changes in focus.

If ActiveOLEControl is not nil (Delphi) or NULL (C++), the user interface for the specified OLE control is deactivated when it is not focused. Also, when the form receives or loses focus, the document window for the specified OLE control is activated or deactivated respectively.

See Also

ActiveControl (see page 13896)

TWinControl (see page 6112)

72.1.5.7.6 TCustomForm.AlphaBlend Property

Specifies whether the form is translucent.

Pascal

```
property AlphaBlend: Boolean;
```

C++

```
__property Boolean AlphaBlend;
```

Description

Set AlphaBlend to specify that the form represents a layered window that allows a translucent color. The AlphaBlendValue (see page 13898) property specifies the degree of translucency.

Note: AlphaBlend does not work on all systems. The application must be running under Windows 2000 or better, and the machine must be a P90 or better.

See Also

AlphaBlendValue (see page 13898)

TransparentColor (see page 13938)

72.1.5.7.7 TCustomForm.AlphaBlendValue Property

Specifies the degree of translucency on a translucent form.

Pascal

```
property AlphaBlendValue: Byte;
```

C++

```
__property Byte AlphaBlendValue;
```

Description

Set AlphaBlendValue to a value between 0 and 255 to indicate the degree of translucency when the AlphaBlend (see page 13898) property is true. A value of 0 indicates a completely transparent window. A value of 255 indicates complete opacity.

Note: AlphaBlendValue only has an effect when the AlphaBlend (see page 13898) property is true.

See Also

AlphaBlend (see page 13898)

TransparentColor (see page 13938)

TransparentColorValue (see page 13939)

72.1.5.7.8 TCustomForm.AutoScroll Property

Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

If AutoScroll is true, the scroll bars appear automatically when necessary. For example, if the user resizes the control so that some of its controls are partially obscured, scroll bars appear.

If AutoScroll is false, scroll bars don't appear automatically. In this case, use the HorzScrollBar and VertScrollBar properties to make scroll bars appear.

See Also

AutoScrollInView (see page 14325)

HorzScrollBar (see page 14329)

VertScrollBar (see page 14330)

72.1.5.7.9 TCustomForm.BorderIcons Property

Specifies which icons appear on the title bar of the form.

Pascal

```
property BorderIcons: TBorderIcons;
```

C++

```
__property TBorderIcons BorderIcons;
```

Description

Use BorderIcons to get or set the icons that appear on the title bar of the form. BorderIcons can include any of the following TBorderIcons (see page 14370) values:

Value	Meaning
biSystemMenu	The form has a Control menu (also known as a System (see page 22079) menu).
biMinimize	The form has a Minimize button
biMaximize	The form has a Maximize button
biHelp	If BorderStyle (see page 13900) is bsDialog or biMinimize and biMaximize are excluded, a question mark appears in the form's title bar and when clicked, the cursor changes to crHelp; otherwise,no question mark appears.

Note: Certain combinations of the BorderIcons and BorderStyle (see page 13900) properties are mutually exclusive. For example, BorderIcons biMax, biMin with BorderStyle (see page 13900) of bsDialog are mutually exclusive.

Delphi Examples:

```
{
```

```
BorderIcons example
The following code removes a form's Maximize button when the user clicks a button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
BorderIcons := BorderIcons - [biMaximize];
end;
```

See Also

BorderStyle (see page 13900)

72.1.5.7.10 TCustomForm.BorderStyle Property

Specifies the appearance and behavior of the form border.

Pascal

```
property BorderStyle: TFormBorderStyle;
```

C++

```
__property TFormBorderStyle BorderStyle;
```

Description

Use BorderStyle to get or set the appearance and behavior of the form border. BorderStyle can be any of the following TForm (see page 13992) BorderStyle values:

Value	Meaning
bsDialog	Not resizable; no minimize/maximize menu
bsSingle	Not resizable; minimize/maximize menu
bsNone	Not resizable; no visible border line
bsSizeable	Standard resizable border
bsToolWindow	like bsSingle but with a smaller caption
bsSizeToolWin	like bsSizeable with a smaller caption

Note: Changing the border style of an MDI child form to bsDialog or bsNone has no effect.

See Also

BorderIcons (see page 13899)

72.1.5.7.11 TCustomForm.Canvas Property

Provides access to the drawing area of the form.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use TCanvas to draw on the client area of the form. Canvas is often used in the OnPaint (see page 13975) event handler.

C++ Examples:

```
/*
```

```
The following code loads a bitmap from a file and assigns it
to the Brush of the Canvas of Form1:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *BrushBmp = new Graphics::TBitmap;
  try
  {
    BrushBmp->LoadFromFile("../bm1.BMP");
    Form1->Canvas->Brush->Bitmap = BrushBmp;
    Form1->Canvas->FillRect(Rect(0,0,100,100));
  }
  __finally
  {
    Form1->Canvas->Brush->Bitmap = NULL;
    delete BrushBmp;
  }
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TForm1 *myForm;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TForm1")
  {
    myForm = (TForm1 *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myForm->Canvas->MoveTo(myForm->ClientRect.Left, myForm->ClientRect.Top);
    myForm->Canvas->LineTo(myForm->ClientRect.Right, myForm->ClientRect.Bottom);
    myForm = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TForm1) then
  begin
    myForm:= (ListBox1.Items.Objects[Index] as TForm1);
    with myForm.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myForm.Canvas.MoveTo(Left, Top);
      myForm.Canvas.LineTo(Right, Bottom);
    end;
  myForm:= nil;
  end;
{
The following code loads a bitmap from a file and assigns it
to the Brush of the Canvas of Form1:
}
var
  Bitmap: TBitmap;

procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  Bitmap := TBitmap.Create;
  try
    Bitmap.LoadFromFile('bm1.BMP');
    Form1.Canvas.Brush.Bitmap := Bitmap;
    Form1.Canvas.FillRect(Rect(0,0,100,100));
  finally
    Form1.Canvas.Brush.Bitmap := nil;
    Bitmap.Free;
  end;
end;
```

See Also

TCanvas (see page 14441)

OnPaint (see page 13975)

72.1.5.7.12 TCustomForm.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
```

```
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
```

```
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

72.1.5.7.13 TCustomForm.ClientHandle Property

Provides access to the handle of the form's internal MDI client window.

Pascal

```
property ClientHandle: HWND;
```

C++

```
__property HWND ClientHandle;
```

Description

Use ClientHandle to get the handle for the form's internal MDI client window.

ClientHandle is meaningful only if the form is an MDI parent (that is, if the form's FormStyle (see page 13912) property is set to fsMDIForm).

Note: Use the Handle property instead to get the handle to the form window itself.

See Also

FormStyle (see page 13912)

Handle (see page 6207)

72.1.5.7.14 TCustomForm.ClientHeight Property

Specifies the height (in pixels) of the form's client area.

Pascal

```
property ClientHeight: Integer;
```

C++

```
__property int ClientHeight;
```

Description

Use ClientHeight to determine the height (in pixels) of the form's client area. The client area is the usable area inside the form's border, excluding the title bar, scroll bars, and so on.

Set ClientHeight to change the height of the form's window based on the desired client area. To change the height of the form's window based on the total size of the window (including the border, menu, status bar and so on), use the Height (see page 13915) property instead.

Delphi Examples:

```
{
This example reduces the height of the form's client area by
half when the user clicks the button on the form:
Note that only the client area is halved, not the entire form.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ClientHeight := ClientHeight div 2;
end;
```

See Also

ClientRect

ClientWidth (see page 13905)

Height (see page 5825)

72.1.5.7.15 TCustomForm.ClientWidth Property

Specifies the width (in pixels) of the form's client area.

Pascal

```
property ClientWidth: Integer;
```

C++

```
__property int ClientWidth;
```

Description

Use ClientWidth to determine the width (in pixels) of the form's client area. The client area is the usable area inside the form's border. Set ClientWidth to change the width of the form's window based on the desired client area. To change the width of the form's window based on the total size of the window (including the border, status bar and so on), use the Width (see page 13941) property instead.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;
```

```
  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form's logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
    Range := 600;
    Position := Range - ClientWidth; { start form out fully scrolled }
    Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
    Visible := True;  { Show the scrollbar }
  end;
end;
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

See Also

ClientHeight (see page 13905)

ClientRect

Width (see page 5849)

72.1.5.7.16 TCustomForm.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

72.1.5.7.17 TCustomForm.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

72.1.5.7.18 TCustomForm.DefaultMonitor Property

Specifies the monitor on which the form appears.

Pascal

```
property DefaultMonitor: TDefaultMonitor;
```

C++

```
__property TDefaultMonitor DefaultMonitor;
```

Description

Use DefaultMonitor to associate a form with a particular monitor in a multi-monitor application. The following table lists the possible values:

Value	Meaning
dmDesktop	No attempt is made to position the form on a specific monitor.
dmPrimary	The form is positioned on the first monitor listed in the global screen object's Monitors property.
dmMainForm	The form appears on the same monitor as the application's main form.
dmActiveForm	The form appears on the same monitor as the currently active form.

Note: DefaultMonitor has no effect if the application does not have a main form.

See Also

Monitors (see page 14228)

TMonitor (see page 14191)

Screen (see page 14385)

MainForm (see page 13718)

Active (see page 13896)

Monitor (see page 13926)

MakeFullyVisible (see page 13881)

72.1.5.7.19 TCustomForm.Designer Property

Specifies the designer interface for the form.

Pascal

```
property Designer: IDesignerHook;
```

C++

```
__property IDesignerHook Designer;
```

Description

The Designer property is used internally at design time. Never assign a value to this property. This is done automatically by the form designer. Only use this property when designing classes for use in the form designer, such as property editors and component editors.

See Also

TPropertyEditor (see page 11019)

TComponentEditor (see page 10927)

72.1.5.7.20 TCustomForm.DropTarget Property

Specifies whether the form is the target of a drag-and-drop operation.

Pascal

```
property DropTarget: Boolean;
```

C++

```
__property Boolean DropTarget;
```

Description

Use DropTarget to determine whether the form is currently the target of a drag-and-drop operation.

See Also

BeginDrag (see page 5740)

DragMode (see page 5819)

EndDrag (see page 5757)

OnDragOver (see page 5857)

72.1.5.7.21 TCustomForm.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

72.1.5.7.22 TCustomForm.FormState Property

Indicates transitional state information about the form.

Pascal

```
property FormState: TFormState;
```

C++

```
__property TFormState FormState;
```

Description

Read FormState to determine the state of the form. FormState indicates when the form is in various transitional states or when certain Windows operations have occurred. The following table lists the values that can be included in a form's state:

Value	Meaning
fsCreating	The form's constructor is currently executing.
fsVisible	The form's window is visible. This state is used to update the Visible (see page 13940) property.
fsShowing	The form's WindowState (see page 13942) property is changing. This state is used to prevent WindowState (see page 13942) changes from interfering with a transition that is in progress.
fsModal	The form was created as a modal window.
fsCreatedMDIChild	The form is an MDI parent window. This state is only set once the MDI client window has been created.
fsActivated	The form has received a CM_ACTIVATE message (because it received focus or the application became active) but has not yet called the Activate (see page 13865) method to generate an OnActivate (see page 13943) event.

See Also

Visible (see page 13940)

Create (see page 13861)

WindowState (see page 13942)

ShowModal (see page 13890)

FormStyle (see page 13912)

Activate (see page 13865)

72.1.5.7.23 TCustomForm.FormStyle Property

Determines the form's style.

Pascal

```
property FormStyle: TFormStyle;
```

C++

```
__property TFormStyle FormStyle;
```

Description

Use FormStyle to get or set the form's style. FormStyle is one of the following values:

Value	Meaning
fsNormal	The form is neither an MDI parent window nor an MDI child window.
fsMDIChild	The form is an MDI child window.
fsMDIForm	The form is an MDI parent window.

fsStayOnTop	This form remains on top of the desktop and of other forms in the project, except any others that also have FormStyle set to fsStayOnTop. If one fsStayOnTop form launches another, neither form will consistently remain on top.

If the form is the main form of an MDI application, its FormStyle property must be set to fsMDIForm.

Note: It is not advisable to change FormStyle at runtime.

Warning: TGraphicControl descendants placed in the client area of a form with FormStyle set to fsMDIForm will not paint.

C++ Examples:

```
/*
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
*/
void __fastcall TForm1::MyArrangeIconsClick(TObject *Sender)
{
  Form1->ArrangeIcons();
}
/*
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
*/
void __fastcall TForm1::TileFormsClick(TObject *Sender)
{
  TileMode = tbVertical;
  Tile();
}
```

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
}
procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
  Form1.ArrangeIcons;
```

```
end;
{
This example ensures that the main form of the application
is an MDI parent:
}
type
  TForm1 = class(TForm)
    Edit1: TEdit;
  private
    { Private declarations }
  public
    { Public declarations }
    constructor Create(AOwner : TComponent); override;
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

constructor TForm1.Create(AOwner : TComponent);

begin
  inherited Create(AOwner);
  if FormStyle <> fsMDIForm then
          FormStyle := fsMDIForm;
  if FormStyle = fsMDIForm then
          Edit1.Text := 'MDI form'
  else
    //This line never executes
    Edit1.Text := 'Not an MDI form';
end;
{
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
}
procedure TForm1.TileFormsClick(Sender: TObject);
begin
  TileMode := tbVertical;
  Tile;
end;
```

See Also

ArrangeIcons (see page 14006)

Tile (see page 14009)

Cascade (see page 14007)

Windows Overview

72.1.5.7.24 TCustomForm.GlassFrame Property

Pascal

```
property GlassFrame: TGlassFrame;
```

C++

```
__property TGlassFrame GlassFrame;
```

Description

This is GlassFrame, a member of class TCustomForm.

72.1.5.7.25 TCustomForm.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
HideSelection, Height example
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
```

```
      with TRichEdit.Create(Self) do
      begin
        Parent := Self;
        Top := Height * I;
        Text := 'This text will take the focus.';
        SelStart := 0;
        SelLength := Length(Text);
        if (I = 1) then
          HideSelection := False;
      end;
  end;
```

See Also

ClientHeight (see page 5808)

Width (see page 5849)

Using TDBListBox and TDBComboBox

72.1.5.7.26 TCustomForm.HelpFile Property

Specifies the name of the file the form uses to display Help.

Pascal

```
property HelpFile: string;
```

C++

```
__property AnsiString HelpFile;
```

Description

Use HelpFile for forms that have a help file to use with the Help system. Help topics are selected from the specified file.

See Also

HelpContext (see page 5826)

OnHelp (see page 13959)

HelpFile (see page 13711)

HelpKeyword (see page 5826)

72.1.5.7.27 TCustomForm.HorzScrollBar Property

Represents the horizontal scroll bar for the scrolling windowed control.

Pascal

```
property HorzScrollBar: TControlScrollBar;
```

C++

```
__property TControlScrollBar HorzScrollBar;
```

Description

Use HorzScrollBar to hide, show, or manipulate the horizontal scroll bar for the scrolling windowed control.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form's logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
    Range := 600;
    Position := Range - ClientWidth; { start form out fully scrolled }
    Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
    Visible := True;  { Show the scrollbar }
  end;
end;
```

See Also

AutoScroll (see page 14329)

VertScrollBar (see page 14330)

TControlScrollBar (see page 13744)

72.1.5.7.28 TCustomForm.Icon Property

Specifies the icon that appears when the form is minimized.

Pascal

```
property Icon: TIcon;
```

C++

```
__property TIcon Icon;
```

Description

Set the Icon property to provide an icon for the form. If Icon is not set, the system provides a default icon when the form is minimized.

Delphi Examples:

```
{
This code assigns an icon to a form when the form is created:
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Icon.LoadFromFile('MDIChild1.ICO');
end;
```

See Also

WindowState (see page 13942)

72.1.5.7.29 TCustomForm.KeyPreview Property

Specifies whether the form should receive keyboard events before the active control.

Pascal

```
property KeyPreview: Boolean;
```

C++

```
__property Boolean KeyPreview;
```

Description

If KeyPreview is true, keyboard events occur on the form before they occur on the active control. (The active control is specified by the ActiveControl (see page 13896) property.)

If KeyPreview is false, keyboard events occur only on the active control.

Navigation keys (Tab, BackTab, the arrow keys, and so on) are unaffected by KeyPreview because they do not generate keyboard events. Similarly, when a button has focus or when its Default property is true, the Enter key is unaffected by KeyPreview because it does not generate a keyboard events.

KeyPreview is false by default.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
```

```
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
```

```
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

ActiveControl (see page 13896)

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

OnKeyPress (see page 6227)

72.1.5.7.30 TCustomForm.Left Property

Specifies the horizontal coordinate of the left edge of a component relative to its parent.

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

Use the Left property to determine where the left side of the control begins or to reposition the left side of the control.

If the control is contained in another control, the Left and Top properties are relative to the parent control. If the control is contained directly by the form, the property values are relative to the form. For forms, the value of the Left property is relative to the screen in pixels.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Timer1->Interval = 100;
  if (ActiveControl)
    ActiveControl->Left = ActiveControl->Left + 1;
}
```

Delphi Examples:

```
{
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  if ActiveControl <> nil then
    ActiveControl.Left := ActiveControl.Left + 1;
end;
```

See Also

Top (see page 5844)

72.1.5.7.31 TCustomForm.MDIChildCount Property

Specifies the number of open MDI child forms.

Pascal

```
property MDIChildCount: Integer;
```

C++

```
__property int MDIChildCount;
```

Description

Use MDIChildCount to get the number of open MDI child forms.

MDIChildCount is meaningful only if the form is an MDI parent (that is, if the form's FormStyle (see page 13912) property is set to fsMDIForm).

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items CloseChildren. This is code for the
```

```
CloseChildrenClick handler.  When the user chooses the
CloseChildren command, all the MDI children of Form1 are
closed.
}
procedure TForm1.CloseChildrenClick(Sender: TObject);
var
  I: Integer;
begin
  with Form1 do
    for I := MDIChildCount-1 downto 0 do
      MDIChildren[I].Close;
end;

procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
  Form1.ArrangeIcons;
end;

procedure TForm1.MyCascadeClick(Sender: TObject);
begin
  Form1.Cascade;
end;
```

See Also

MDIChildren (see page 13922)

72.1.5.7.32 TCustomForm.MDIChildren Property

Provides indexed access to all MDI child forms.

Pascal

```
property MDIChildren [I: Integer]: TForm;
```

C++

```
__property TForm * MDIChildren[int I];
```

Description

Use MDIChildren to access a child form.

I is the index of the child form to access. The order of MDI children changes as different MDI forms are activated. Initially, forms are listed in creation order, with new forms being added to the end of the array. Whenever an MDI child is activated, however, it moves to the front of the list. Thus, use I to index over all forms in the array, but not to refer to a specific form.

MDIChildren is meaningful only if the form is an MDI frame (that is, if the form's FormStyle (see page 13912) property is set to fsMDIForm).

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items CloseChildren. This is code for the
CloseChildrenClick handler.  When the user chooses the
CloseChildren command, all the MDI children of Form1 are
closed.
}
procedure TForm1.CloseChildrenClick(Sender: TObject);
var
  I: Integer;
begin
```

```
  with Form1 do
    for I := MDIChildCount-1 downto 0 do
      MDIChildren[I].Close;
end;

procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
  Form1.ArrangeIcons;
end;

procedure TForm1.MyCascadeClick(Sender: TObject);
begin
  Form1.Cascade;
end;
```

See Also

MDIChildCount (see page 13921)

FormStyle (see page 13912)

72.1.5.7.33 TCustomForm.Menu Property

Specifies the form's main menu.

Pascal

```
property Menu: TMainMenu;
```

C++

```
__property TMainMenu Menu;
```

Description

Use Menu to get or set the form's main menu. At design time, Menu is set to the first TMainMenu component added to the form.

C++ Examples:

```
/*
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *Item1 = NewItem("New item", TextToShortCut("Ctrl+N"), false, true, MyOnExecute,
0, "Item1");
  Menu = NewMenu(this, "MyMenu", &Item1, 0);
}

void __fastcall TForm1::MyOnExecute(TObject *Sender)
{
  MessageDlg("Item1 has been executed.", mtInformation, TMsgDlgButtons() << mbOK, 0);
}
/*
To create a toolbar that corresponds to an existing menu:

1   Drop a ToolBar on the form and add a ToolButton for each
  top-level menu item you wish to create.
2   Set the MenuItem property of each ToolButton to correspond
  to the top level menu items.
3   Set the Grouped property of each ToolButton to true.
4   Clear the Menu property of the Form (if it is assigned)
5   Set the Visible property on the ToolBar and ToolButtons to
  true.
6   Set the ShowCaptions property on the ToolBar to true.
```

```
7   Set the MenuItem | Visible property of each ToolButton to
  true.
*/
```

Delphi Examples:

```
{
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Menu := NewMenu(Self, 'MyMenu',
            [NewItem('New item', TextToShortCut('Ctrl+N'),
            False, True, MyOnExecute, 0, 'Item1')]);
end;

procedure TForm1.MyOnExecute(Sender: TObject);
begin
  MessageDlg('Item1 has been executed.', mtInformation, [mbOK], 0);
end;
{
To create a toolbar that corresponds to an existing menu:

1   Drop a ToolBar on the form and add a ToolButton for each top-level menu item you wish to
create.
2   Set the MenuItem property of each ToolButton to correspond to the top level menu items.
3   Set the Grouped property of each ToolButton to true.
4   Clear the Menu property of the Form (if it is assigned)
5   Set the Visible property on the ToolBar and ToolButtons to true.
6   Set the ShowCaptions property on the ToolBar to true.
7   Set the MenuItem | Visible property of each ToolButton to true.
}
```

See Also

TMainMenu (see page 16056)

72.1.5.7.34 TCustomForm.ModalResult Property

Represents the return value of a form that is used as a modal dialog.

Pascal

```
property ModalResult: TModalResult;
```

C++

```
__property TModalResult ModalResult;
```

Description

Use ModalResult to close the form when it is displayed modally.

By default, ModalResult is mrNone. Set ModalResult to any nonzero value to close the form. The value assigned to ModalResult becomes the return value of the ShowModal (see page 13890) function call used to display the form.

Note: This property is modified automatically by some components such as the TButton object.

C++ Examples:

```
/*
This code brings up the modal dialog from Unit2 when a
button is clicked.  It causes a Beep if the OK button is
```

```
clicked. Implement the modal form in Unit2 and then put
Unit2 in the uses clause for this module.
*/
#include "Unit2.h"
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (MyDialogBox1->ShowModal() == mrOk)
    MessageBeep(0);
}
/*
The following methods are used for buttons in a form that is
used as a modal dialog box. The methods cause the dialog box
to terminate when the user clicks either the OK or Cancel
button, returning mrOk or mrCancel from ShowModal,
respectively. You could also set the ModalResult value to
mrOk for the OK button and mrCancel for the Cancel button to
accomplish the same thing. When the user clicks either
button, the dialog box closes.
*/
void __fastcall TMyDialogBox1::OKButtonClick(TObject *Sender)
{
  ModalResult = mrOk;
}

void __fastcall TMyDialogBox1::CancelButtonClick(TObject *Sender)
{
  ModalResult = mrCancel;
}
```

Delphi Examples:

```
{
This code brings up the modal dialog from Unit2 when a
button is clicked.  It causes a Beep if the OK button is
clicked. Implement the modal form in Unit2 and then put
Unit2 in the uses clause for this module.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if MyDialogBox1.ShowModal = mrOK then
    Beep;
end;
{
The following methods are used for buttons in a form that is
used as a modal dialog box. The methods cause the dialog box
to terminate when the user clicks either the OK or Cancel
button, returning mrOk or mrCancel from ShowModal,
respectively. You could also set the ModalResult value to
mrOk for the OK button and mrCancel for the Cancel button to
accomplish the same thing. When the user clicks either
button, the dialog box closes.
}
procedure TMyDialogBox1.CancelButtonClick(Sender: TObject);
begin
  ModalResult := mrCancel;
end;

procedure TMyDialogBox1.OKButtonClick(Sender: TObject);
begin
  ModalResult := mrOK;
end;
```

See Also

ShowModal (see page 13890)

ModalResult (see page 20937)

72.1.5.7.35 TCustomForm.Monitor Property

Provides access to the monitor on which the form appears.

Pascal

```
property Monitor: TMonitor;
```

C++

```
__property TMonitor Monitor;
```

Description

Use Monitor to access information about the monitor on which the form appears if the application runs on a multi-monitor system. The monitor is determined by the DefaultMonitor (see page 13909) property.

See Also

Monitors (see page 14228)

BoundsRect (see page 5803)

DefaultMonitor (see page 13909)

MakeFullyVisible (see page 13881)

72.1.5.7.36 TCustomForm.ObjectMenuItem Property

Represents an OLE object menu item that reacts to selections of OLE objects.

Pascal

```
property ObjectMenuItem: TMenuItem;
```

C++

```
__property TMenuItem ObjectMenuItem;
```

Description

Use ObjectMenuItem to get or set a menu item that becomes enabled or disabled when an OLE object on the form is selected or unselected respectively.

The OLE object menu item can be used for standard OLE commands, such as Activate (see page 13865) or Convert. (The implementations of these commands are provided by the OLE server.)

See Also

OleFormObject (see page 13927)

72.1.5.7.37 TCustomForm.OldCreateOrder Property

Specifies when OnCreate (see page 13951) and OnDestroy (see page 13953) events occur.

Pascal

```
property OldCreateOrder: Boolean;
```

C++

```
__property Boolean OldCreateOrder;
```

Description

When OldCreateOrder is false (the default) the OnCreate (see page 13951) event occurs after all constructors are finished

and the OnDestroy (see page 13953) event occurs before any destructors are called.

This differs from earlier versions of the library (distributed with Delphi version 3 and previous and C++Builder 1), where the OnCreate (see page 13951) event occurred when the TCustomForm (see page 13849) constructor executed and the OnDestroy (see page 13953) event occurred when the TCustomForm (see page 13849) destructor executed. Applications that require the OnCreate (see page 13951) event and OnDestroy (see page 13953) events to occur from the constructor and destructor of the form can set OldCreateOrder to true.

See Also

OnCreate (see page 13951)

OnDestroy (see page 13953)

BeforeDestruction (see page 13866)

AfterConstruction (see page 13866)

72.1.5.7.38 TCustomForm.OleFormObject Property

Specifies the IOleForm (see page 14340) interface for an in-place OLE object contained in the form.

Pascal

```
property OleFormObject: IOleForm;
```

C++

```
__property IOleForm OleFormObject;
```

Description

OleFormObject is used to communicate resize and destroy notifications when a form is used as a frame for an in-place OLE object.

See Also

ObjectMenuItem (see page 13926)

72.1.5.7.39 TCustomForm.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

72.1.5.7.40 TCustomForm.PixelsPerInch Property

Represents the proportion of the font on the system on which the form was designed.

Pascal

```
property PixelsPerInch: Integer;
```

C++

```
__property int PixelsPerInch;
```

Description

Use PixelsPerInch at runtime to change how the form is scaled to the current screen resolution. If PixelsPerInch is changed from its default value, the form will not have the same proportion on every screen resolution.

At design time, PixelsPerInch is set automatically when the form is saved. Changes to PixelsPerInch made in the Object Inspector only affect the form at runtime.

The form's Scaled (see page 13935) property must be true for PixelsPerInch to have an effect.

See Also

Scaled (see page 13935)

72.1.5.7.41 TCustomForm.PopupChildren Property

Pascal

```
property PopupChildren: TList;
```

C++

```
__property TList PopupChildren;
```

Description

This is PopupChildren, a member of class TCustomForm.

72.1.5.7.42 TCustomForm.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
```

```
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
```

```
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
```

```
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}
```

```
type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;
```

```
procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

72.1.5.7.43 TCustomForm.PopupMode Property

Pascal

```
property PopupMode: TPopupMode;
```

C++

```
__property TPopupMode PopupMode;
```

Description

This is PopupMode, a member of class TCustomForm.

72.1.5.7.44 TCustomForm.PopupParent Property

Sets an order for stacked forms that users cannot change.

Pascal

```
property PopupParent: TCustomForm;
```

C++

```
__property TCustomForm PopupParent;
```

Description

If the PopupMode (see page 13933) property is set to pmExplicit and PopupParent is nil, then the Application.MainForm is implicitly used as the PopupParent. If no Application.MainForm is assigned, then Application.Handle is used as the PopupParent.

If the PopupMode (see page 13933) property is set to pmAuto, Screen.ActiveForm is used as the PopupParent property.

72.1.5.7.45 TCustomForm.Position Property

Represents the size and placement of the form.

Pascal

```
property Position: TPosition;
```

C++

```
__property TPosition Position;
```

Description

Use Position to get or set the size and placement of the form. Position can have one of the following TPosition (see page 14363) values:

Value	Meaning
poDesigned	The form appears positioned on the screen and with the same height and width as it had at design time.
poDefault	The form appears in a position on the screen and with a height and width determined by the operating system. Each time you run the application, the form moves slightly down and to the right. The right side of the form is always near the far right side of the screen, and the bottom of the form is always near the bottom of the screen, regardless of the screen's resolution. The operating system only supports this for overlapped windows. For a pop-up or child window (that has a BorderStyle (see page 13900) of bsDialog or bsNone, for example), the window is positioned in the upper left corner of the screen.
poDefaultPosOnly	The form displays with the size you created it at design time, but the operating system chooses its position on the screen. Each time you run the application, the form moves slightly down and to the right. When the form can no longer move down and to the right and keep the same size while remaining entirely visible on the screen, the form displays at the top-left corner of the screen. The operating system only supports this for overlapped windows. For a pop-up or child window (that has a BorderStyle (see page 13900) of bsDialog or bsNone, for example), the window is positioned in the upper left corner of the screen.
poDefaultSizeOnly	The form appears in the position you left it at design time, but the operating system chooses its size. The right side of the form is always near the far right side of the screen, and the bottom of the form is always near the bottom of the screen, regardless of the screen's resolution.
poScreenCenter	The form remains the size you left it at design time, but is positioned in the center of the screen. In multi-monitor applications, the form may be moved from this center position so that it falls entirely on one monitor, as specified by the DefaultMonitor (see page 13909) property.
poDesktopCenter	The form remains the size you left it at design time, but is positioned in the center of the screen. No adjustments are made for multi-monitor applications.
poMainFormCenter	The form remains the size you left it at design time, but is positioned in the center of the application's main form. No adjustments are made for multi-monitor applications. This position should only be used with secondary forms. If set for a main form, it acts like poScreenCenter.
poOwnerFormCenter	The form remains the size you left it at design time, but is positioned in the center of the form specified by the Owner property. If the Owner property does not specify a form, this position acts like poMainFormCenter.

See Also

OnResize (see page 5870)

SetInitialBounds

72.1.5.7.46 TCustomForm.PrintScale Property

Represents the proportions of a printed form.

Pascal

```
property PrintScale: TPrintScale;
```

C++

```
__property TPrintScale PrintScale;
```

Description

Use PrintScale to get or set the proportions of the printed form. PrintScale can have one of the following TPrintScale (see page 14364) values:

Value	Meaning
poNone	No special scaling occurs; therefore, the printed form and how the form appears onscreen may appear squished or stretched.
poProportional	The form is printed so that the printed image is approximately the same visible size as on the screen (WYSIWYG). The form image is scaled so that the printed image is approximately the same visible size as on the screen.
poPrintToFit	The form is printed using the same screen proportions, but in a size that just fits the printed page.

See Also

Print (see page 13883)

72.1.5.7.47 TCustomForm.Scaled Property

Specifies whether the form is sized according to the value of the PixelsPerInch (see page 13928) property.

Pascal

```
property Scaled: Boolean;
```

C++

```
__property Boolean Scaled;
```

Description

Scaled determines whether the form adjusts itself to differences between the font used at design time and the current system font used by a system at runtime. If Scaled is true, the form resizes itself and all its child controls to maintain the relationship between the size of controls and the height of text displayed in the default font. If Scaled is false, no such resizing occurs.

By setting Scaled to true, the form can adjust to fonts that are not scalable (that is, fonts that only supply discrete point sizes) or to differences such as forms designed using small fonts at design time that are run on systems that use large fonts.

The degree of resizing is determined by the PixelsPerInch (see page 13928) property, which measures the proportion of the font on the system used when designing the form.

See Also

PixelsPerInch (see page 13928)

Considerations When Dynamically Resizing Forms and Controls

72.1.5.7.48 TCustomForm.ScreenSnap Property

Specifies whether form snaps to edge of screen.

Pascal

```
property ScreenSnap: Boolean;
```

C++

```
__property Boolean ScreenSnap;
```

Description

ScreenSnap determines whether the edges of the form snap to the edge of the screen when the user moves the form. To control the snap distance, set the SnapBuffer (see page 13936) property.

See Also

SnapBuffer (see page 13936)

72.1.5.7.49 TCustomForm.SnapBuffer Property

Specifies distance for screen snap.

Pascal

```
property SnapBuffer: Integer;
```

C++

```
__property int SnapBuffer;
```

Description

If ScreenSnap (see page 13935) is true, SnapBuffer determines the maximum number of pixels that be between the form edge and the screen edge before the form snaps to the screen edge.

See Also

ScreenSnap (see page 13935)

72.1.5.7.50 TCustomForm.TileMode Property

Represents how MDI child forms are arranged when the Tile method is called.

Pascal

```
property TileMode: TTileMode;
```

C++

```
__property TTileMode TileMode;
```

Description

TileMode can have one of the following TTileMode (see page 14367) values:

Value	Meaning
tbHorizontal	Each form stretches across the width of the parent form
tbVertical	Each form stretches along the height of the parent form

TileMode is meaningful only if the form is an MDI child (that is, if the form's FormStyle (see page 13912) property is set to fsMDIChild).

C++ Examples:

```
/*
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
*/
void __fastcall TForm1::TileFormsClick(TObject *Sender)
{
  TileMode = tbVertical;
  Tile();
}
```

Delphi Examples:

```
{
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
}
procedure TForm1.TileFormsClick(Sender: TObject);
begin
  TileMode := tbVertical;
  Tile;
end;
```

See Also

FormStyle (see page 13912)

Tile (see page 14009)

72.1.5.7.51 TCustomForm.Top Property

Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

Use Top to locate the top of the control or reposition the control to a different Y coordinate. The Top property, like the Left property, is the position of the control relative to its container. Thus, if a control is contained in a TPanel, the Left and Top properties are relative to the panel. If the control is contained directly by the form, it is relative to the form. For forms, the value of the Top property is relative to the screen in pixels.

C++ Examples:

```
/*
This example uses an updown control and a group box on a
form, with several controls contained within the group box.
When the updown control or is clicked, the controls within
the group box are moved in the appropriate direction.
*/
void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  int I;
  TControl *ChildControl;

  for (I = 0; I < GroupBox1->ControlCount; I++)
  {
    ChildControl = GroupBox1->Controls[I];
    if (Button == btNext)
      ChildControl->Top = ChildControl->Top + 15;
    else
      ChildControl->Top = ChildControl->Top - 15;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  const TColor colorarray[5] = {clYellow, clGreen, clBlue, clLime, clFuchsia};
  GroupBox1->Brush->Color = clRed;
  for (int i = 0; i < GroupBox1->ControlCount; i++)
    ((TWinControl *)GroupBox1->Controls[i])->Brush->Color = colorarray[i];
}
```

Delphi Examples:

```
{
This example uses an updown control and a group box on a
form, with several controls contained within the group box.
When the updown control or is clicked, the controls within
the group box are moved in the appropriate direction.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
const
  colorarray : Array[0..4] of TColor = (clYellow, clGreen, clBlue, clLime, clFuchsia);
begin
   GroupBox1.Brush.Color := clRed;
  for I:= 0 to GroupBox1.ControlCount -1 do
  begin
    TWinControl(GroupBox1.Controls[I]).Brush.Color := colorarray[I];
  end;
end;

procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
var
  I: Integer;
  ChildControl: TControl;
begin
  for I:= 0 to GroupBox1.ControlCount -1 do
  begin
    ChildControl := GroupBox1.Controls[I];
    if Button = btNext then
      ChildControl.Top := ChildControl.Top + 15
    else
      ChildControl.Top := ChildControl.Top - 15;
  end;
end;
```

See Also

Left (see page 5830)

72.1.5.7.52 TCustomForm.TransparentColor Property

Specifies whether a color on the form appears transparent.

Pascal

```
property TransparentColor: Boolean;
```

C++

```
__property Boolean TransparentColor;
```

Description

Use TransparentColor to indicate that one of the colors on the form should be treated as transparent, allowing windows behind the form to completely show through. The TransparentColorValue (see page 13939) property indicates the color that appears completely transparent.

Note: To make the entire form transparent, or to make it translucent rather than transparent, use the AlphaBlend (see page 13898) and AlphaBlendValue (see page 13898) properties.

Note: TransparentColor does not work on all systems. The application must be running under Windows 2000 or better, with adequate hardware.

See Also

TransparentColorValue (see page 13939)

AlphaBlend (see page 13898)

AlphaBlendValue (see page 13898)

72.1.5.7.53 TCustomForm.TransparentColorValue Property

Indicates the color on the form that appears transparent when TransparentColor (see page 13938) is true.

Pascal

```
property TransparentColorValue: TColor;
```

C++

```
__property TColor TransparentColorValue;
```

Description

Use TransparentColorValue to indicate the color that appears transparent when the TransparentColor (see page 13938) property is true.

Note: If the application uses a 16-bit color palette, TransparentColorValue must be a basic color.

See Also

TransparentColor (see page 13938)

72.1.5.7.54 TCustomForm.VertScrollBar Property

Represents the vertical scroll bar for the scrolling windowed control.

Pascal

```
property VertScrollBar: TControlScrollBar;
```

C++

```
__property TControlScrollBar VertScrollBar;
```

Description

Use VertScrollBar to hide, show, or manipulate the vertical scroll bar for the scrolling windowed control.

C++ Examples:

```
/*
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  const int iPageDelta = 10;

  if (Key == VK_NEXT)
    VertScrollBar->Position += iPageDelta;
  else if (Key == VK_PRIOR)
    VertScrollBar->Position -= iPageDelta;
```

```
}
```

Delphi Examples:

```
{
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
}
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
const
  PageDelta = 10;
begin
  With VertScrollbar do
    if Key = VK_NEXT then
      Position := Position + PageDelta
    else if Key = VK_PRIOR then
      Position := Position - PageDelta;
end;
```

See Also

AutoScroll (see page 14329)

HorzScrollBar (see page 14329)

TControlScrollBar (see page 13744)

72.1.5.7.55 TCustomForm.Visible Property

Indicates whether the form is visible.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use Visible to determine whether the form is visible.

If Visible is true, the form is visible unless it is completely obscured by other forms. If Visible is false, the form is not visible.

The Show (see page 13888) and ShowModal (see page 13890) methods set Visible to true and bring the form to the front of all open windows.

C++ Examples:

```
/*
The following code uses two forms. Form1 has a button on it. The
second form is used as a tool palette. This code makes the palette
form visible, and ensures it is the top form by bringing it to the
front.  To run this example, you must include Unit2.h in this
module.
*/
#include "Unit2.h"

void __fastcall TForm1::ShowPaletteButtonClick(TObject *Sender)
{
  if (!Form2->Visible)
  {
    Form2->Visible = true;
    Form2->BringToFront();
  }
}
```

Delphi Examples:

```
{
The following code uses two forms. Form1 has a button on it. The
second form is used as a tool palette. This code makes the palette
form visible, and ensures it is the top form by bringing it to the
front.  To run this example, you must add Unit2 in the uses clause
of your unit.
}
procedure TForm1.ShowPaletteButtonClick(Sender: TObject);
begin
if Form2.Visible = False then Form2.Visible := True;
Form2.BringToFront;
end;
```

See Also

BringToFront (see page 5740)

Show (see page 13888)

ShowModal (see page 13890)

72.1.5.7.56 TCustomForm.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

See Also

ClientWidth (see page 5811)

Height (see page 5825)

Left (see page 5830)

72.1.5.7.57 TCustomForm.WindowMenu Property

Specifies the Window menu for an MDI parent form.

Pascal

```
property WindowMenu: TMenuItem;
```

C++

```
__property TMenuItem WindowMenu;
```

Description

Use WindowMenu to get or set the Window menu for an MDI parent form. The Window menu is a standard menu in MDI applications. It contains commands that let the user manage the windows in the application. Menu (see page 13923) items usually include Cascade, Arrange Icons, Tile, and so on.

The Window menu also lists (at the bottom) the child windows that are currently open in the application. When the user selects one of these windows from the menu, the window becomes the active window in the application.

Although this menu is commonly called the Window menu, it can have any name that corresponds to an existing menu item on the menu bar. The names of open child forms are merged onto the menu automatically at run-time.

WindowMenu is meaningful only if the form is an MDI parent (that is, if the form's FormStyle (see page 13912) property is set to fsMDIForm).

See Also

FormStyle (see page 13912)

72.1.5.7.58 TCustomForm.WindowState Property

Represents how the form appears on the screen.

Pascal

```
property WindowState: TWindowState;
```

C++

```
__property TWindowState WindowState;
```

Description

Set WindowState to minimize, maximize, or restore the form window. Read WindowState to determine whether the form is minimized, maximized, or in a normal state.

See Also

Icon (see page 13917)

BorderStyle (see page 13900)

72.1.5.8 TCustomForm Events

The events of the TCustomForm class are listed here.

Protected Events

	Name	Description
	OnActivate (see page 13943)	Occurs when the form becomes active.
	OnCanResize (see page 13947)	Occurs when an attempt is made to resize the control.
	OnClick (see page 13947)	Occurs when the user clicks the control.
	OnClose (see page 13949)	Occurs when the form closes.
	OnCloseQuery (see page 13950)	Occurs when close is attempted.
	OnCreate (see page 13951)	Occurs when the form is created.
	OnDblClick (see page 13953)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 13953)	Occurs when the form loses focus.
	OnDestroy (see page 13953)	Occurs when the form is destroyed.
	OnDragDrop (see page 13954)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 13957)	Occurs when the user drags an object over a control.
	OnHelp (see page 13959)	Occurs when the form receives a request for help.
	OnHide (see page 13960)	Occurs when the form is hidden (that is, when its Visible (see page 13940) property is set to false).
	OnKeyDown (see page 13962)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 13964)	Occurs when a key is pressed.

	OnKeyUp (see page 13966)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 13967)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 13968)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 13971)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 13973)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnPaint (see page 13975)	Occurs when the form is redrawn.
	OnResize (see page 13976)	Occurs immediately after the control is resized.
	OnShortCut (see page 13977)	Occurs when the user presses a key (before the OnKeyDown (see page 13962) event).
	OnShow (see page 13977)	Occurs when the form is shown (that is, when its Visible (see page 13940) property is set to true).

72.1.5.8.1 TCustomForm.OnActivate Event

Occurs when the form becomes active.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Use OnActivate to perform special processing when the form receives focus. A form becomes active when focus is transferred to it (when the user clicks on the form, for example).

Note: The OnActivate event of the application (TApplication (see page 13660)), not the form, occurs when Windows switches control from another application.

Delphi Examples:

```
{
Here is the code on Form1 that fills the list box.
}
var
  Form1: TForm1;
  Customers: TTable;
  Blobs: TTable;
  Form1Done : Boolean;
  procedure CreateMyTable;
  procedure CreateCustSource;
  procedure CreateBlobSource;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  CreateCustSource;
  CreateBlobSource;
  for I := 0 to ComponentCount - 1 do
  begin
    if (Components[I] is TDataSource) then
```

```
      ListBox1.Items.AddObject(Components[I].Name,
        Components[I] as TObject);
  end;
end;

procedure CreateCustSource();
begin
  Customers:= TTable.Create(Form1);
  with Customers do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'MyCustInfo';
    ReadOnly:= False;

//    Don't overwrite an existing table
    if (not Customers.Exists) then CreateMyTable
    else
    begin
      if (Customers.Exists AND
        (MessageDlg('MyCustInfo table already exists.  Do you want to rebuild it?',
mtConfirmation, [mbYes, mbNo], 0) = mrYes)) then
      begin
        Customers.Close;
        Customers.DeleteTable;
        CreateMyTable;
      end;
    end;
  end;
  Form1.CustomerDS.DataSet:= Customers;
  Customers.Active:= True;
end;

procedure CreateMyTable();
var
  i: Integer;
begin
//      The Table component must not be active
//      First, describe the type of table and give
//      it a name
//      Next, describe the fields in the table
  with Customers do
  begin
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
        with AddFieldDef do
        begin
          Name := 'Field3';
          DataType := ftString;
          Size := 30;
        end;
        with AddFieldDef do
        begin
```

```
            Name := 'Field4';
            DataType := ftString;
            Size := 30;
          end;
        end;

//      Take out this part to remove indexing
//      Next, describe any indexes
        with IndexDefs do
        begin
          Clear;
//          The 1st index has no name because it is
//          a Paradox primary key
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;

//        Call the CreateTable method to create the table
        CreateTable;
        Customers.Active:= True;
        for i := 1 to 20 do
          Customers.AppendRecord([i*111, i*222, i*333, i*444]);
      end;
  end;

  procedure CreateBlobSource;
  var
    i: Integer;
  begin
    Blobs:= TTable.Create(Form1);
    with Blobs do
    begin
      DatabaseName := 'DBDEMOS';
      TableType := ttParadox;
      TableName := 'MyBlobInfo';
      Blobs.Active := False;
      { Don't overwrite an existing table }
//      if Blobs.Exists then
//        MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
//      else
      begin
        if (Blobs.Exists) then
        begin
          Blobs.Close;
          Blobs.DeleteTable;
        end;
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
        { Next, describe the fields in the table }
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
```

```
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftBlob;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
{
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
}
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
      Blobs.Active:= True;
      for i := 1 to 20 do
        Blobs.AppendRecord([i*111, IntToStr(i*222)]);
    end;
  end;
  Form1.BlobDS.DataSet:= Blobs;
  Blobs.Active:= True;
end;

procedure TForm1.ListBox1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  Form1Done := True;
end;

initialization

Form1Done := False;
{
The following code sets the DataSource for a DBGrid
component in Form2 to the currently selected data source in
a list provided in Form1. This allows users to specify the
desired information in Form1, and then move to Form2 to view
the information. Select an item in the listbox and then
activate Form2 by clicking in it.
}
procedure TForm2.FormActivate(Sender: TObject);
begin
  if (Form1Done) then
  begin
    DBGrid2.DataSource :=
      Form1.ListBox1.Items.Objects[Form1.ListBox1.ItemIndex] as TDataSource;
    DBNavigator2.DataSource :=
```

```
      Form1.ListBox1.Items.Objects[Form1.ListBox1.ItemIndex] as TDataSource;
    end;
  end;
```

See Also

FormStyle (see page 13912)

OnDeactivate (see page 13953)

OnActivate (see page 13726)

72.1.5.8.2 TCustomForm.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize: TCanResizeEvent;
```

C++

```
__property TCanResizeEvent OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize (see page 5853)

SetBounds (see page 5783)

OnResize (see page 5870)

72.1.5.8.3 TCustomForm.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.

- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

72.1.5.8.4 TCustomForm.OnClose Event

Occurs when the form closes.

Pascal

```
property OnClose: TCloseEvent;
```

C++

```
__property TCloseEvent OnClose;
```

Description

Use OnClose to perform special processing when the form closes. The OnClose event specifies which event handler to call when a form is about to close. The handler specified by OnClose might, for example, test to make sure all fields in a data-entry form have valid contents before allowing the form to close.

A form is closed by the Close (see page 13867) method or when the user chooses Close (see page 13867) from the form's system menu.

The TCloseEvent (see page 14371) type points to a method that handles the closing of a form. The value of the Action (see page 13895) parameter determines if the form actually closes. These are the possible values of Action (see page 13895):

Value	Meaning
caNone	The form is not allowed to close, so nothing happens.
caHide	The form is not closed, but just hidden. Your application can still access a hidden form.
caFree	The form is closed and all allocated memory for the form is freed.

caMinimize	The form is minimized, rather than closed. This is the default action for MDI child forms.

If a form is an MDI child form, and its BorderIcons (see page 13899) property is biMinimize, then the default Action (see page 13895) is caMinimize. If a MDI child form does not have these settings, the default Action (see page 13895) is caNone, meaning that nothing happens when the user attempts to close the form.

If a form is an SDI child form, Action (see page 13895) defaults to caHide.

To close the form and free it in an OnClose event, set Action (see page 13895) to caFree.

Note: When the application shuts down, the main form receives an OnClose event, but any child forms do not receive the OnClose event.

C++ Examples:

```
/*
This example displays a message dialog box when the user
attempts to close the form. If the user clicks the Yes
button, the form closes; otherwise, the form only minimizes.
TCustomForm.Close does not allow an action of caMinimize
for a form closing itself.  Close Form2 from Form1 (from
a button?) and implement the OnClose in Form2.  Place
"Application.CreateForm(TForm2, Form2);" in the project dpr
file to open Form2 and set the Form2 Visible property to
True.
*/
void __fastcall TForm2::FormClose(TObject *Sender, TCloseAction &Action)
{
  if (MessageDlg("Close application ?", mtConfirmation, TMsgDlgButtons() << mbYes << mbNo,0)
== mrYes)
    Action = caFree;
  else
    Action = caMinimize;
}
```

See Also

OnCreate (see page 13951)

OnCloseQuery (see page 13950)

OnDestroy (see page 13953)

72.1.5.8.5 TCustomForm.OnCloseQuery Event

Occurs when close is attempted.

Pascal

```
property OnCloseQuery: TCloseQueryEvent;
```

C++

```
__property TCloseQueryEvent OnCloseQuery;
```

Description

Use OnCloseQuery to specify the conditions under which the form can close. An OnCloseQuery event handler returns a Boolean CanClose value that determines whether a form is allowed to close. Its default value is true.

You can use an OnCloseQuery event handler to ask users if they are sure they really want the form closed immediately. For example, you can use the handler to display a message box that prompts the user to save a file before closing the form.

The TCloseQueryEvent (see page 14372) type points to the method that determines whether a form can be closed. The value of the CanClose parameter determines if the form can close or not.

C++ Examples:

```
/*
When the user attempts to close the form in this example, a
message dialog appears that asks the user if it is OK to
close the form. If the user chooses the OK button, the form
closes. If the user chooses Cancel, the form doesn't close.
Populate the OnCloseQuery event handler of the form.
*/
void __fastcall TForm1::FormCloseQuery(TObject *Sender, bool &CanClose)
{
  if (MessageDlg("Close the form?", mtConfirmation, TMsgDlgButtons() << mbOK << mbCancel,0) ==
mrCancel)
    CanClose = false;
}
```

Delphi Examples:

```
{
When the user attempts to close the form in this example, a
message dialog appears that asks the user if it is OK to
close the form. If the user chooses the OK button, the form
closes. If the user chooses Cancel, the form doesn't close.
Populate the OnCloseQuery event handler of the form.
}
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  if MessageDlg('Close the form?', mtConfirmation,
    [mbOk, mbCancel], 0) = mrCancel then
     CanClose := False;
end;
```

See Also

OnClose (see page 13949)

72.1.5.8.6 TCustomForm.OnCreate Event

Occurs when the form is created.

Pascal

```
property OnCreate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCreate;
```

Description

Use OnCreate to perform special processing when the form is created and is invoked by TCustomForm (see page 13849)'s constructor. Either implement this event or override the constructor of the form; do not do both. Any objects created in the OnCreate event should be freed by the OnDestroy (see page 13953) event.

When a form is being created and its Visible (see page 13940) property is true, the following events occur in the order listed:

1. OnCreate2. OnShow3. OnActivate4. OnPaint (see page 13975)

   Note: Use of the OnCreate event is discouraged in C++ code because it can interact badly with the form's constructor (see OldCreateOrder (see page 13926)). It is recommended that you override the form constructor instead.

C++ Examples:

```
/*
This example requires a new TPageControl, as-is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
evaluates as true.  If Button is not btNext, the previous Tab Page
is selected.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
const
  TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *page = new TTabSheet(PageControl1);
    page->PageControl = PageControl1;
    page->Caption = AnsiString("Page") + IntToStr(i);
    page->Brush->Color = colorPalette[i];
  }
}

void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  PageControl1->SelectNextPage(Button == btNext);
}
```

Delphi Examples:

```
{
This example requires a new TPageControl, as-is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
evaluates as true.  If Button is not btNext, the previous Tab Page
is selected.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
const
  colorPalette: Array[0..11] of TColor =
    (clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet #' + IntToStr(i);
      Brush.Color := colorPalette[i];
    end;
end;

procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
begin
  PageControl1.SelectNextPage(Button = btNext);
end;
```

See Also

OnDestroy (see page 13953)

OnActivate (see page 13943)

OnPaint (see page 13975)

OnShow (see page 13977)

OldCreateOrder (see page 13926)

72.1.5.8.7 TCustomForm.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

72.1.5.8.8 TCustomForm.OnDeactivate Event

Occurs when the form loses focus.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Use OnDeactivate to perform special processing when the form transitions from being the active form to another form in the same application becoming the active form. If activation goes to another application, this event is not triggered. To determine if another application has become active, Use the TApplication (see page 13660) object's OnDeactivate event.

See Also

OnActivate (see page 13943)

OnDeactivate (see page 13727)

72.1.5.8.9 TCustomForm.OnDestroy Event

Occurs when the form is destroyed.

Pascal

```
property OnDestroy: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDestroy;
```

Description

Use OnDestroy to perform special processing when the form is destroyed. Either implement this event or override the destructor of the class, but not both. This event should destroy any objects created in the OnCreate (see page 13951) event.

C++ Examples:

```
/*
The following code allocates a TBitmap object that is used
by the application when the main form is first created. The
OnDestroy method frees the bitmap as the application shuts
down.
*/
Graphics::TBitmap *LogoBmp;
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  LogoBmp = new Graphics::TBitmap();
  LogoBmp->LoadFromFile("../FACTORY.BMP");
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete LogoBmp;
}

void __fastcall TForm1::FormPaint(TObject *Sender)
{
  Canvas->Brush->Bitmap = LogoBmp;
  Canvas->FillRect(Rect(0,0,100,100));
}
```

Delphi Examples:

```
{
The following code explicitly allocates memory for a pointer
in the OnCreate event of Form1, then releases the memory in
the OnDestroy event. Assume that MyPtr is a Pointer type
field of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  New(MyPtr);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Dispose(MyPtr);
end;
```

See Also

OnCreate (see page 13951)

Free (see page 22300)

Destroy (see page 13862)

OldCreateOrder (see page 13926)

72.1.5.8.10 TCustomForm.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

72.1.5.8.11 TCustomForm.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

72.1.5.8.12 TCustomForm.OnHelp Event

Occurs when the form receives a request for help.

Pascal

```
property OnHelp: THelpEvent;
```

C++

```
__property THelpEvent OnHelp;
```

Description

Write an OnHelp event handler to perform special processing when the user requests help. The HelpContext and the HelpJump methods automatically trigger the OnHelp event.

Set the CallHelp parameter to true if the application should still invoke the help system after the event. Set CallHelp to false to prevent the default response by the help system. The form calls the help system only if OnHelp's CallHelp parameter returns true or if no OnHelp event handler is assigned.

To find the possible values of the Command and Data parameters, search for WinHelp in the Win32 Developer's Reference Help (Win32.HLP) file, which explains the WinHelp API (application programming interface). The possible values for the Data parameter depend upon the value of the Command parameter.

The event handler returns true if it succeeds, false if it fails.

See Also

HelpCommand (see page 13678)

HelpContext (see page 13679)

HelpFile (see page 13916)

HelpJump (see page 13679)

HelpFile (see page 13711)

OnHelp (see page 13730)

72.1.5.8.13 TCustomForm.OnHide Event

Occurs when the form is hidden (that is, when its Visible (see page 13940) property is set to false).

Pascal

```
property OnHide: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnHide;
```

Description

Use OnHide to perform special processing when the form is hidden (that is, when the form's Visible (see page 13940) property is set to false).

C++ Examples:

```
/*
This example uses two forms, each with a label and a button.
When the user clicks a button, the other form appears and
the current form disappears. Also, a message appears in the
label of the form that is showing, indicating that the other
form is hidden. Set the OnHide event handle for each form to
FormHide.This is the implementation section of Unit1:
*/
#include "TFormOnHide.h"
#include "Unit2.h"  // We must include unit2's header file too!

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
  Hide();
}

void __fastcall TForm1::FormHide(TObject *Sender)
{
  Form2->Label1->Caption = "Form1 is hiding";
}
/*
This is the implementation of Unit2:
*/
```

```
#include "Unit2.h"
#include "TFormOnHide.h"  // We must include unit1's header file too!

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm2 *Form2;

__fastcall TForm2::TForm2(TComponent* Owner)
    : TForm(Owner)
{
}

void __fastcall TForm2::Button1Click(TObject *Sender)
{
  Form1->Show();
  Hide();
}

void __fastcall TForm2::FormHide(TObject *Sender)
{
  Form1->Label1->Caption = "Form2 is hiding";
}
```

Delphi Examples:

```
{
This example uses two forms, each with a label and a button.
When the user clicks a button, the other form appears and
the current form disappears. Also, a message appears in the
label of the form that is showing, indicating that the other
form is hidden. Set the OnHide event handle for each form to
FormHide.This is the implementation section of Unit1:
}

implementation

{$R *.dfm}

uses Unit2;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
  Hide;
end;

procedure TForm1.FormHide(Sender: TObject);
begin
  Form2.Label1.Caption := 'Form1 is hiding';
end;
{
This is the implementation section of Unit2:
}
implementation

{$R *.dfm}

uses TFormOnHide;

procedure TForm2.Button1Click(Sender: TObject);
begin
  Form1.Show;
  Hide;
end;

procedure TForm2.FormHide(Sender: TObject);
begin
```

```
  Form1.Label1.Caption := 'Form2 is hiding';
end;
```

See Also

OnShow (see page 13977)

72.1.5.8.14 TCustomForm.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
```

```
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
```

```
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

72.1.5.8.15 TCustomForm.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

72.1.5.8.16 TCustomForm.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
```

```
  This example changes a form's color to aqua when the user
  presses the F1 key, even when a control on the form has the
  focus. When the user releases the key, the form returns to
  its original color.  Set the KeyPreview on the form to True.
  Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
  event to FormKeyUp.  Set the KeyPreview on the form to True.
  }
  var
    FormColor: TColor;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    KeyPreview := True;
    FormColor := Form1.Color;
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

72.1.5.8.17 TCustomForm.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

72.1.5.8.18 TCustomForm.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
```

```
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

72.1.5.8.19 TCustomForm.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

72.1.5.8.20 TCustomForm.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
```

```
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
```

```
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

72.1.5.8.21 TCustomForm.OnPaint Event

Occurs when the form is redrawn.

Pascal

```
property OnPaint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPaint;
```

Description

Use OnPaint to perform special processing when the form is redrawn. Any special painting on the form should be done in this event. OnPaint occurs before any controls on the form are painted.

To determine which portions of the form's canvas need to be repainted, use the ClipRect property of the canvas.

If you use the form's Canvas (see page 13900) property outside the OnPaint event, it will be erased and drawn-over by the next OnPaint event.

C++ Examples:

```
/*
The following code loads a background bitmap onto the Canvas
of the main form in the OnPaint event handler.
*/
Graphics::TBitmap *TheGraphic;

void __fastcall TForm1::FormPaint(TObject *Sender)// OnPaint event handler
{
  Form1->Canvas->Draw(0, 0, TheGraphic); // Draw the graphic on the Canvas
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TheGraphic = new Graphics::TBitmap(); // Create the bitmap object
  TheGraphic->LoadFromFile("..\\bigf.bmp"); // Load the bitmap from a file
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete TheGraphic;
}
```

Delphi Examples:

```
{
The following code loads a background bitmap onto the Canvas
of the main form in the OnPaint event handler.
}
procedure TForm1.FormPaint(Sender: TObject); { OnPaint event handler}
begin
  Form1.Canvas.Draw(0, 0, TheGraphic); { Draw the graphic on the Canvas }
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  TheGraphic := TBitmap.Create; { Create the bitmap object }
  TheGraphic.LoadFromFile('bigf.bmp'); { Load the bitmap from a file}
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  (TheGraphic as TObject).Free;
end;
```

See Also

Canvas (see page 13900)

ClipRect (see page 14486)

72.1.5.8.22 TCustomForm.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
```

```
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

72.1.5.8.23 TCustomForm.OnShortCut Event

Occurs when the user presses a key (before the OnKeyDown (see page 13962) event).

Pascal

```
property OnShortCut: TShortCutEvent;
```

C++

```
__property TShortCutEvent OnShortCut;
```

Description

Use OnShortCut to dispatch shortcut keystrokes before the form handles them. When the user presses a key, the form can dispatch it as a shortcut key instead of allowing the standard keystroke processing (OnKeyDown (see page 13962), OnKeyPress (see page 13964), and OnKeyUp (see page 13966)). Built-in shortcut processing is provided for menu shortcuts and actions associated with the form. OnShortCut allows the form to implement additional shortcuts.

If the OnShortCut implements a response to the keystroke, set the Handled parameter of the event handler to true. This prevents the keystroke from being passed on to menus or actions associated with the form. It also prevents the standard keystroke processing in the same way that a menu or action shortcut does.

See Also

OnKeyDown (see page 6225)

IsShortCut (see page 13880)

IsShortCut (see page 322)

IsShortCut (see page 16078)

72.1.5.8.24 TCustomForm.OnShow Event

Occurs when the form is shown (that is, when its Visible (see page 13940) property is set to true).

Pascal

```
property OnShow: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnShow;
```

Description

Use OnShow to perform special processing when the form is shown (that is, when the form's Visible (see page 13940) property is set to true).

C++ Examples:

```
/*
This project requires a Page Control.  You will need to
populate the Page Control by right clicking and selecting
New Page from the context menu. As the tabs are selected by
the user, the Page Control OnChange event fires, which
causes the form caption to display the captions of each of
the TTabSheets attached to the Page Control.
Note the use of the Form's OnShow event handler to set the
caption when the form first appears.  This occurs before
any changes trigger the OnChange event of the Page Control.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  Caption = AnsiString("Now working on tab: " + PageControl1->ActivePage->Caption);
}

void __fastcall TForm1::FormShow(TObject *Sender)
{
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This project requires a Page Control.  You will need to
populate the Page Control by right clicking and selecting
New Page from the context menu. As the tabs are selected by
the user, the Page Control OnChange event fires, which
causes the form caption to display the captions of each of
the TTabSheets attached to the Page Control.
Note the use of the Form's OnShow event handler to set the
caption when the form first appears.  This occurs before
any changes trigger the OnChange event of the Page Control.
}
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  Caption := ' Now working on tab:  ' + PageControl1.ActivePage.Caption;
end;

procedure TForm1.FormShow(Sender: TObject);
begin
  PageControl1Change(Sender);
end;
```

See Also

OnHide (see page 13960)

72.1.6 TCustomFrame Class

TCustomFrame is the base class from which TFrame (see page 14114) descends.

Class Hierarchy

Pascal

```
TCustomFrame = class(TScrollingWinControl);
```

C++

```
class TCustomFrame : public TScrollingWinControl;
```

File

Forms

Description

TCustomFrame is the base class for frames. A frame, like a form, is a container for controls and components. It uses the same ownership mechanism as forms for automatic instantiation and destruction of the components on it, and the same parent-child relationships for synchronization of component properties. Unlike forms, however, frames can be nested within forms or other frames, and can be saved on the Component palette for easy reuse. After a frame is created and saved, it continues to function as a unit and to inherit changes from the components (including other frames) it contains. Moreover, an embedded frame continues to inherit changes made to the frame from which it is derived.

Do not create instances of TCustomFrame. Instead, to create and work with frames, define a TCustomFrame descendant by choosing File|New|Frame, and adding the controls and components that you want your frame to encapsulate.

See Also

TForm (see page 13992)

TScrollingWinControl (see page 14312)

TFrame (see page 14114)

72.1.6.1 TCustomFrame Members

The following tables list the members exposed by TCustomFrame.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 14324)	Adjusts the ClientRect property to exclude the scroll bars.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 14325)	Aligns the controls in the scrolling windowed control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
V	AutoScrollEnabled (see page 14325)	Indicates whether automatic scrolling is enabled.
V	AutoScrollInView (see page 14325)	Scrolls a control into the visible area of the scrolling windowed control.

	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	ChangeScale (see page 14326)	Resizes the scrolling windowed control by a specific ratio.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 13990)	Initializes frame creation parameters for efficient redrawing.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 14327)	Creates a Windows control corresponding to the TScrollingWinControl (see page 14312) object.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoFlipChildren (see page 14327)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.

	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 13990)	Calls a specified method for each child of the frame.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.

	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 13991)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
V	Resizing (see page 14328)	This is Resizing, a member of class TScrollingWinControl.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.

	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 13991)	Sets the Parent property.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoScroll (see page 14329)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 13989)	Creates and initializes a TCustomFrame (see page 13978) object.

	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 14323)	Removes the TScrollingWinControl (see page 14312) object from memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	DisableAutoRange (see page 14327)	Disables automatic scrolling.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EnableAutoRange (see page 14328)	Reenables automatic scrolling.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.

	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollInView (see page 14328)	Scrolls a control into the visible area of the scrolling windowed control.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.

	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.

	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14332)	Occurs when an object with custom alignment is aligned.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HorzScrollBar (see page 14329)	Represents the horizontal scroll bar for the scrolling windowed control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	VertScrollBar (see page 14330)	Represents the vertical scroll bar for the scrolling windowed control.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

72.1.6.2 TCustomFrame.Create Constructor

Creates and initializes a TCustomFrame (see page 13978) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomFrame(TComponent * AOwner);
```

Description

Use Create to create and initialize a frame.

AOwner is the component that owns the frame.

See Also

TFrame (see page 14114)

Owner (see page 2807)

ParentWindow (see page 6210)

72.1.6.3 TCustomFrame Methods

The methods of the TCustomFrame class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 13990)	Initializes frame creation parameters for efficient redrawing.
	GetChildren (see page 13990)	Calls a specified method for each child of the frame.
	PaintWindow (see page 13991)	Renders the image of a windowed control.
	SetParent (see page 13991)	Sets the Parent property.

72.1.6.3.1 TCustomFrame.CreateParams Method

Initializes frame creation parameters for efficient redrawing.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 14327) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle.

See Also

CreateParams (see page 14326)

CreateWnd (see page 14327)

CreateWindowHandle (see page 6137)

72.1.6.3.2 TCustomFrame.GetChildren Method

Calls a specified method for each child of the frame.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the component streaming system that loads and saves forms and data modules. Applications seldom need to call this routine.

Proc is a callback to execute for every component in the frame.

Root indicates the top-level component that is the entry point for the entire streaming process. This can be a form, data module, or frame.

GetChildren executes the method specified by Proc for every component contained by the frame. It provides extra support for components that do not have parents if the Root parameter identifies this frame.

See Also

GetChildren (see page 6153)

72.1.6.3.3 TCustomFrame.PaintWindow Method

Renders the image of a windowed control.

Pascal

```
procedure PaintWindow(DC: HDC); override;
```

C++

```
virtual __fastcall PaintWindow(HDC DC);
```

Description

Call PaintWindow to repaint the control. PaintWindow sends a WM_PAINT message to the windowed control's DefaultHandler method, setting the message record's WParam field to the value passed in DC and the other parameter and result fields to zeros.

See Also

DefaultHandler (see page 6139)

PaintTo (see page 6167)

DoubleBuffered (see page 6205)

72.1.6.3.4 TCustomFrame.SetParent Method

Sets the Parent property.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. TCustomFrame (see page 13978) overrides the method inherited from TControl to

Update the parent form so that it is aware of any action lists in the frame.

Regenerate the frame's window so that it fits properly into its new parent.

See Also

Parent (see page 5832)

72.1.7 TForm Class

TForm represents a standard application window (form).

Class Hierarchy

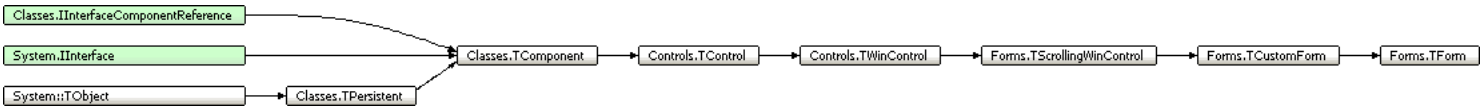

Pascal

```
TForm = class(TCustomForm);
```

C++

```
class TForm : public TCustomForm;
```

File

Forms

Description

When you create forms in the Form designer at design time, they are implemented as descendants of TForm. Forms (see page 13657) can represent the application's main window, or dialog boxes, or MDI children. A form can contain other objects, such as TButton, TCheckBox, and TComboBox objects.

Examples of forms include TLoginDialog and TPasswordDialog objects.

See Also

TLoginDialog (see page 9776)

TPasswordDialog (see page 9828)

Creating Applications

MDI Applications

SDI Applications

User Interface Models

GUI Applications

Adding Forms

Controlling When Forms Reside in Memory

Creating Forms Dynamically

Creating Modeless Forms Such as Windows

Displaying an Auto-created Form

Using Forms

Passing Additional Arguments to Forms

Retrieving Data from Forms

Retrieving Data from Modal Forms

Retrieving Data from Modeless Forms

Controlling Application Behavior

Creating a Form Instance Using a Local Variable

Using the Main Form

VCL Overview

Windows Overview

72.1.7.1 TForm Members

The following tables list the members exposed by TForm.

Protected Events

	Name	Description
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFormState (see page 13863)	This is FFormState, a member of class TCustomForm.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	Activate (see page 13865)	Activation event dispatcher.
	ActiveChanged (see page 13865)	Active (see page 13896) state event dispatcher.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.

	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustClientRect (see page 14324)	Adjusts the ClientRect property to exclude the scroll bars.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	AlignControls (see page 13866)	Aligns all child controls of the form that fall within a specified rectangle.
	AlignControls (see page 14325)	Aligns the controls in the scrolling windowed control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	AutoScrollEnabled (see page 14325)	Indicates whether automatic scrolling is enabled.
	AutoScrollInView (see page 14325)	Scrolls a control into the visible area of the scrolling windowed control.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	BeginAutoDrag (see page 13867)	Responds when the user starts dragging the form if DragMode is dmAutomatic.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	ChangeScale (see page 13867)	Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.
	ChangeScale (see page 14326)	Resizes the scrolling windowed control by a specific ratio.
	Click (see page 5743)	Respond to user click.
	CloseModal (see page 13868)	This is CloseModal, a member of class TCustomForm.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 13868)	Initializes the window-creation parameter record when the form window is created.
	CreateParams (see page 14326)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWindowHandle (see page 13869)	Creates a form window given a set of window-creation parameters.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 13869)	Creates the form window.
	CreateWnd (see page 14327)	Creates a Windows control corresponding to the TScrollingWinControl (see page 14312) object.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	Deactivate (see page 13870)	Generates an OnDeactivate (see page 13953) event.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.

	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DefineProperties (see page 13871)	Designates methods for storing a form's unpublished data on a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyHandle (see page 13871)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWindowHandle (see page 13872)	Destroys the window created in the CreateWindowHandle (see page 13869) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoClose (see page 13873)	Form closing event dispatcher.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoCreate (see page 13873)	Form creation event dispatcher.
	DoDestroy (see page 13873)	Form destruction event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDock (see page 13874)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoFlipChildren (see page 14327)	Flips the control's children.
	DoHide (see page 13874)	Form hide event dispatcher.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoShow (see page 13875)	Form show event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.

	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorderIconStyles (see page 13875)	This is GetBorderIconStyles, a member of class TCustomForm.
	GetBorderStyles (see page 13876)	This is GetBorderStyles, a member of class TCustomForm.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetChildren (see page 13876)	Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetClientRect (see page 13876)	Returns the value of the ClientRect property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloating (see page 13877)	Returns the value of the Floating property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetOwnerWindow (see page 13878)	This is GetOwnerWindow, a member of class TCustomForm.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 13878)	Handles exceptions that are raised by the OnCreate (see page 13951) event handler.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 13880)	Performs fix-ups when the form is first loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.

	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	Notification (see page 13882)	Responds to notifications that objects are about to be inserted or removed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 13882)	Generates an OnPaint (see page 13975) event.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaintWindow (see page 13882)	Renders the image of the form.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteChanged (see page 13883)	Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	QueryInterface (see page 13884)	Returns a reference to a specified interface if that interface is supported by the form.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	ReadState (see page 13884)	Reads the form's published properties, stored data, and owned components from a reader object.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	RequestAlign (see page 13885)	Instructs the form's parent (if any) to reposition the form, enforcing its Align property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	Resizing (see page 13885)	Makes any required adjustments when the form changes size.
	Resizing (see page 14328)	This is Resizing, a member of class TScrollingWinControl.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.

	SetChildOrder (see page 13886)	Changes the order in which Child appears when GetChildren (see page 13876) is called.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParent (see page 13887)	Sets the Parent property.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentBiDiMode (see page 13887)	Sets the ParentBiDiMode property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateActions (see page 13891)	Updates all actions associated with the form.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	UpdateWindowState (see page 13892)	Updates the WindowState (see page 13942) property to reflect the current state of the form.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateRename (see page 13892)	Determines if a form can rename one of its owned components.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	VisibleChanging (see page 13893)	Responds just before the form changes the value of its Visible (see page 13940) property.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WndProc (see page 13893)	Provides specific message responses for the form.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
R	PopupChildren (see page 13928)	This is PopupChildren, a member of class TCustomForm.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 13866)	Construction event dispatcher.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	ArrangeIcons (see page 14006)	Arranges the icons of minimized MDI child forms.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 13866)	Destruction event dispatcher.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	Cascade (see page 14007)	Arranges MDI child forms so they overlap.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Close (see page 13867)	Closes the form.
V	CloseQuery (see page 13868)	Close (see page 13867) attempt event dispatcher.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 13861)	Creates and initializes a new TForm (see page 13992) object.
V	Create (see page 14323)	Creates and initializes a new TScrollingWinControl (see page 14312) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	CreateNew (see page 13862)	Creates and initializes a new form.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 13870)	Provides message handling for all messages that the form does not fully process.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DefocusControl (see page 13871)	Removes focus from a control on the form.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 13862)	Removes the form from memory.
V	Destroy (see page 14323)	Removes the TScrollingWinControl (see page 14312) object from memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	DisableAutoRange (see page 14327)	Disables automatic scrolling.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
V	Dock (see page 13872)	Handles automatic docking of the form into other windows.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EnableAutoRange (see page 14328)	Reenables automatic scrolling.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.

	FocusControl (see page 13875)	Sets focus to a control on the form.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFormImage (see page 13877)	Returns a bitmap of the form.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	Hide (see page 13879)	Hides the form.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	IsShortCut (see page 13880)	Processes shortcut keys when the form has focus.
	MakeFullyVisible (see page 13881)	Ensures that the form is fully visible on a specified monitor.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MouseWheelHandler (see page 13881)	Directs mouse wheel messages to the appropriate control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 14008)	Activates the next child form in the form sequence.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.

	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	Previous (see page 14009)	Activates the previous MDI child form in the form sequence.
	Print (see page 13883)	Prints the form.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	RecreateAsPopup (see page 13884)	This is RecreateAsPopup, a member of class TCustomForm.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	Release (see page 13885)	Destroys the form and frees its associated memory.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollInView (see page 14328)	Scrolls a control into the visible area of the scrolling windowed control.
	SendCancelMode (see page 13886)	Cancels modes on the form.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
V	SetFocus (see page 13886)	Sets focus to the form.
V	SetFocusedControl (see page 13887)	Sets focus to a control on the form.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Show (see page 13888)	Shows the form.
V	ShowModal (see page 13890)	Shows a form as a modal dialog.
	Tile (see page 14009)	Arranges MDI child forms so that they are all the same size.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UpdateGlassFrame (see page 13892)	This is UpdateGlassFrame, a member of class TCustomForm.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.
V	WantChildKey (see page 13893)	Indicates whether the form processes keyboard input for a control it owns.

Public Properties

	Name	Description
R	Active (see page 13896)	Specifies whether the form has focus.
	ActiveMDIChild (see page 14013)	Specifies the MDI child that has focus.

	ActiveOleControl (see page 13897)	Specifies the OLE control on the form that reacts to changes in focus.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 13900)	Provides access to the drawing area of the form.
	ClientHandle (see page 14024)	Provides access to the handle of the form's internal MDI client window.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designer (see page 13910)	Specifies the designer interface for the form.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 14029)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropTarget (see page 13910)	Specifies whether the form is the target of a drag-and-drop operation.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	FormState (see page 13911)	Indicates transitional state information about the form.
	GlassFrame (see page 13914)	This is GlassFrame, a member of class TCustomForm.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	MDIChildCount (see page 14044)	Specifies the number of open MDI child forms.
	MDIChildren (see page 14044)	Provides indexed access to all MDI child forms.
	ModalResult (see page 13924)	Represents the return value of a form that is used as a modal dialog.
R	Monitor (see page 13926)	Provides access to the monitor on which the form appears.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	OleFormObject (see page 13927)	Specifies the IOleForm (see page 14340) interface for an in-place OLE object contained in the form.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.

	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TileMode (see page 14059)	Represents how MDI child forms are arranged when the Tile method is called.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnActivate (see page 14065)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 14069)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14069)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 14070)	Occurs when an attempt is made to resize the control.
	OnClick (see page 14070)	Occurs when the user clicks the control.
	OnClose (see page 14072)	Occurs when the form closes.
	OnCloseQuery (see page 14073)	Occurs when close is attempted.
	OnConstrainedResize (see page 14074)	Adjust resize constraints.
	OnContextPopup (see page 14075)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 14076)	Occurs when the form is created.
	OnDblClick (see page 14077)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 14078)	Occurs when the form loses focus.
	OnDestroy (see page 14078)	Occurs when the form is destroyed.
	OnDockDrop (see page 14079)	Occurs when another control is docked to the control.
	OnDockOver (see page 14081)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 14083)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 14085)	Occurs when the user drags an object over a control.
	OnEndDock (see page 14088)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnGetSiteInfo (see page 14088)	Returns the control's docking information.
	OnHelp (see page 14090)	Occurs when the form receives a request for help.
	OnHide (see page 14091)	Occurs when the form is hidden (that is, when its Visible property is set to false).
	OnKeyDown (see page 14093)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 14095)	Occurs when a key is pressed.
	OnKeyUp (see page 14096)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 14098)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 14099)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 14101)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 14102)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 14102)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 14104)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 14107)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 14107)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 14107)	Occurs when the mouse wheel is rotated upward.
	OnPaint (see page 14108)	Occurs when the form is redrawn.
	OnResize (see page 14109)	Occurs immediately after the control is resized.
	OnShortCut (see page 14109)	Occurs when the user presses a key (before the OnKeyDown event).
	OnShow (see page 14110)	Occurs when the form is shown (that is, when its Visible property is set to true).
	OnStartDock (see page 14111)	Occurs when the user begins to drag a control with a DragKind of dkDock.

	OnUnDock (see page 14111)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Action (see page 14011)	Designates the action associated with the control.
	ActiveControl (see page 14012)	Specifies the control that has focus on the form.
	Align (see page 14013)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AlphaBlend (see page 14017)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 14017)	Specifies the degree of translucency on a translucent form.
	Anchors (see page 14017)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 14018)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 14018)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BiDiMode (see page 14019)	Specifies the bi-directional mode for the control.
	BorderIcons (see page 14019)	Specifies which icons appear on the title bar of the form.
	BorderStyle (see page 14020)	Specifies the appearance and behavior of the form border.
	BorderWidth (see page 14021)	Specifies the width of the control's border.
	Caption (see page 14021)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 14024)	Specifies the height (in pixels) of the form's client area.
	ClientWidth (see page 14025)	Specifies the width (in pixels) of the form's client area.
	Color (see page 14027)	Specifies the background color of the control.
	Constraints (see page 14028)	Specifies the size constraints for the control.
	Ctl3D (see page 14028)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DefaultMonitor (see page 14029)	Specifies the monitor on which the form appears.
	DockSite (see page 14031)	Specifies whether the control can be the target of drag-and-dock operations.
	DragKind (see page 14032)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 14033)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 14034)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 14035)	Controls the attributes of text written on or in the control.
	FormStyle (see page 14036)	Determines the form's style.
	Height (see page 14038)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpFile (see page 14039)	Specifies the name of the file the form uses to display Help.
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HorzScrollBar (see page 14040)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 14041)	Specifies the icon that appears when the form is minimized.
	KeyPreview (see page 14041)	Specifies whether the form should receive keyboard events before the active control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Left (see page 13920)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Menu (see page 14045)	Specifies the form's main menu.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectMenuItem (see page 14047)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 14047)	Specifies when OnCreate and OnDestroy events occur.

	Padding (see page 14047)	Specifies the padding of a control.
	ParentBiDiMode (see page 14048)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 14048)	Determines where a control looks for its font information.
	PixelsPerInch (see page 14049)	Represents the proportion of the font on the system on which the form was designed.
	PopupMenu (see page 14049)	Identifies the pop-up menu associated with the control.
	PopupMode (see page 14054)	Controls (see page 5711) how the top-level form behaves with respect to Window's WS_POPUP style.
	PopupParent (see page 14054)	Sets an order for stacked forms that users cannot change.
	Position (see page 14055)	Represents the size and placement of the form.
	PrintScale (see page 14056)	Represents the proportions of a printed form.
	Scaled (see page 14056)	Specifies whether the form is sized according to the value of the PixelsPerInch property.
	ScreenSnap (see page 14056)	Specifies whether form snaps to edge of screen.
	ShowHint (see page 14057)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SnapBuffer (see page 14058)	Specifies distance for screen snap.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Top (see page 13937)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	TransparentColor (see page 14060)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 14060)	Indicates the color on the form that appears transparent when TransparentColor is true.
	UseDockManager (see page 14060)	Specifies whether the docking manager is used in drag-and-dock operations.
	VertScrollBar (see page 14061)	Represents the vertical scroll bar for the scrolling windowed control.
	Visible (see page 14062)	Indicates whether the form is visible.
	Width (see page 14063)	Specifies the horizontal size of the control or form in pixels.
	WindowMenu (see page 14063)	Specifies the Window menu for an MDI parent form.
	WindowState (see page 14063)	Represents how the form appears on the screen.

72.1.7.2 TForm Methods

The methods of the TForm class are listed here.

Public Methods

	Name	Description
	ArrangeIcons (see page 14006)	Arranges the icons of minimized MDI child forms.
	Cascade (see page 14007)	Arranges MDI child forms so they overlap.
	Next (see page 14008)	Activates the next child form in the form sequence.
	Previous (see page 14009)	Activates the previous MDI child form in the form sequence.
	Tile (see page 14009)	Arranges MDI child forms so that they are all the same size.

72.1.7.2.1 TForm.ArrangeIcons Method

Arranges the icons of minimized MDI child forms.

Pascal

```
procedure ArrangeIcons;
```

C++

```
__fastcall ArrangeIcons();
```

Description

Use ArrangeIcons to arrange the icons of minimized MDI child forms so they are evenly spaced and don't overlap.

ArrangeIcons works only if the form is an MDI parent form (that is, if the form's FormStyle (see page 14036) property is fsMDIForm).

C++ Examples:

```
/*
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
*/
void __fastcall TForm1::MyArrangeIconsClick(TObject *Sender)
{
  Form1->ArrangeIcons();
}
```

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
}
procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
  Form1.ArrangeIcons;
end;
```

See Also

Cascade (see page 14007)

Tile (see page 14009)

FormStyle (see page 13912)

72.1.7.2.2 TForm.Cascade Method

Arranges MDI child forms so they overlap.

Pascal

```
procedure Cascade;
```

C++

```
__fastcall Cascade();
```

Description

Use Cascade to arrange MDI child forms so they overlap.

Cascade works only if the form is an MDI parent form (that is, if the form's FormStyle (see page 14036) property is fsMDIForm).

C++ Examples:

```
/*
This example uses several forms. The first form has its
FormStyle property set to fsMDIForm. The others have their
FormStyle properties set to fsMDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyCascade. This is code for the
MyCascadeClick handler.  When the user chooses the
Cascade command, restored forms are arranged
so that they overlap.  DO NOT name the menu items "Cascade"
as that will override the TForm method.
*/
void __fastcall TForm1::MyCascadeClick(TObject *Sender)
{
  Form1->Cascade();
}
```

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to fsMDIForm. The others have their
FormStyle properties set to fsMDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyCascade. This is code for the
MyCascadeClick handler.  When the user chooses the
Cascade command, restored forms are arranged
so that they overlap.  DO NOT name the menu items "Cascade"
as that will override the TForm method.
}
procedure TForm1.MyCascadeClick(Sender: TObject);
begin
  Form1.Cascade;
end;
```

See Also

Tile (see page 14009)

ArrangeIcons (see page 14006)

FormStyle (see page 13912)

72.1.7.2.3 TForm.Next Method

Activates the next child form in the form sequence.

Pascal

```
procedure Next;
```

C++

```
__fastcall Next();
```

Description

Use Next to change the active child form of an MDI parent. For example, if you have three child forms within a parent form and Form2 is the active form, the Next method makes Form3 the active form. Calling Next again makes Form4 active. The next time your application calls Next, the sequence starts over again and Form2 becomes the active form once again.

The Next method applies only to forms that are MDI parent forms (have a FormStyle (see page 14036) property value of fsMDIForm).

See Also

Previous (see page 14009)

FormStyle (see page 13912)

72.1.7.2.4 TForm.Previous Method

Activates the previous MDI child form in the form sequence.

Pascal

```
procedure Previous;
```

C++

```
__fastcall Previous();
```

Description

Use Previous to change the active child form of an MDI parent. For example, if you have three child forms within a parent form and Form4 is the active form, the Previous method makes Form3 the active form. Calling Previous again makes Form2 active. The next time your application calls Previous, the sequence starts over again and Form4 becomes the active form once again.

The Previous method applies only to forms that are MDI parent forms (have a FormStyle (see page 14036) property value of fsMDIForm).

See Also

Next (see page 14008)

72.1.7.2.5 TForm.Tile Method

Arranges MDI child forms so that they are all the same size.

Pascal

```
procedure Tile;
```

C++

```
__fastcall Tile();
```

Description

Use Tile to arrange MDI child forms so that they are all the same size. Tiled forms completely fill up the client area of the parent form. How the forms arrange themselves depends upon the values of their TileMode (see page 14059) properties.

Tile works only if the form is an MDI parent form (that is, if the form's FormStyle (see page 14036) property is fsMDIForm).

C++ Examples:

```
/*
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
*/
void __fastcall TForm1::TileFormsClick(TObject *Sender)
{
  TileMode = tbVertical;
```

```
  Tile();
}
```

Delphi Examples:

```
{
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
}
procedure TForm1.TileFormsClick(Sender: TObject);
begin
  TileMode := tbVertical;
  Tile;
end;
```

See Also

TileMode (see page 13936)

72.1.7.3 TForm Properties

The properties of the TForm class are listed here.

Public Properties

	Name	Description
	ActiveMDIChild (see page 14013)	Specifies the MDI child that has focus.
	ClientHandle (see page 14024)	Provides access to the handle of the form's internal MDI client window.
	DockManager (see page 14029)	Specifies the control's docking manager interface.
	MDIChildCount (see page 14044)	Specifies the number of open MDI child forms.
	MDIChildren (see page 14044)	Provides indexed access to all MDI child forms.
	TileMode (see page 14059)	Represents how MDI child forms are arranged when the Tile method is called.

Published Properties

	Name	Description
	Action (see page 14011)	Designates the action associated with the control.
	ActiveControl (see page 14012)	Specifies the control that has focus on the form.
	Align (see page 14013)	Determines how the control aligns within its container (parent control).
	AlphaBlend (see page 14017)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 14017)	Specifies the degree of translucency on a translucent form.
	Anchors (see page 14017)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 14018)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 14018)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BiDiMode (see page 14019)	Specifies the bi-directional mode for the control.
	BorderIcons (see page 14019)	Specifies which icons appear on the title bar of the form.
	BorderStyle (see page 14020)	Specifies the appearance and behavior of the form border.
	BorderWidth (see page 14021)	Specifies the width of the control's border.
	Caption (see page 14021)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 14024)	Specifies the height (in pixels) of the form's client area.
	ClientWidth (see page 14025)	Specifies the width (in pixels) of the form's client area.
	Color (see page 14027)	Specifies the background color of the control.
	Constraints (see page 14028)	Specifies the size constraints for the control.

	Ctl3D (see page 14028)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultMonitor (see page 14029)	Specifies the monitor on which the form appears.
	DockSite (see page 14031)	Specifies whether the control can be the target of drag-and-dock operations.
	DragKind (see page 14032)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 14033)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 14034)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 14035)	Controls the attributes of text written on or in the control.
	FormStyle (see page 14036)	Determines the form's style.
	Height (see page 14038)	Specifies the vertical size of the control in pixels.
	HelpFile (see page 14039)	Specifies the name of the file the form uses to display Help.
	HorzScrollBar (see page 14040)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 14041)	Specifies the icon that appears when the form is minimized.
	KeyPreview (see page 14041)	Specifies whether the form should receive keyboard events before the active control.
	Menu (see page 14045)	Specifies the form's main menu.
	ObjectMenuItem (see page 14047)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 14047)	Specifies when OnCreate and OnDestroy events occur.
	Padding (see page 14047)	Specifies the padding of a control.
	ParentBiDiMode (see page 14048)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 14048)	Determines where a control looks for its font information.
	PixelsPerInch (see page 14049)	Represents the proportion of the font on the system on which the form was designed.
	PopupMenu (see page 14049)	Identifies the pop-up menu associated with the control.
	PopupMode (see page 14054)	Controls (see page 5711) how the top-level form behaves with respect to Window's WS_POPUP style.
	PopupParent (see page 14054)	Sets an order for stacked forms that users cannot change.
	Position (see page 14055)	Represents the size and placement of the form.
	PrintScale (see page 14056)	Represents the proportions of a printed form.
	Scaled (see page 14056)	Specifies whether the form is sized according to the value of the PixelsPerInch property.
	ScreenSnap (see page 14056)	Specifies whether form snaps to edge of screen.
	ShowHint (see page 14057)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SnapBuffer (see page 14058)	Specifies distance for screen snap.
	TransparentColor (see page 14060)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 14060)	Indicates the color on the form that appears transparent when TransparentColor is true.
	UseDockManager (see page 14060)	Specifies whether the docking manager is used in drag-and-dock operations.
	VertScrollBar (see page 14061)	Represents the vertical scroll bar for the scrolling windowed control.
	Visible (see page 14062)	Indicates whether the form is visible.
	Width (see page 14063)	Specifies the horizontal size of the control or form in pixels.
	WindowMenu (see page 14063)	Specifies the Window menu for an MDI parent form.
	WindowState (see page 14063)	Represents how the form appears on the screen.

72.1.7.3.1 TForm.Action Property

Designates the action associated with the control.

Pascal

```
property Action: TBasicAction;
```

C++

```
__property TBasicAction Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user

commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

72.1.7.3.2 TForm.ActiveControl Property

Specifies the control that has focus on the form.

Pascal

```
property ActiveControl: TWinControl;
```

C++

```
__property TWinControl * ActiveControl;
```

Description

Use ActiveControl to get or set the control that has focus on the form. Only one control can have focus at a given time in an application.

If the form does not have focus, ActiveControl is the control on the form that will receive focus when the form receives focus.

Note: When focus shifts to another control, ActiveControl is updated before the OnExit event occurs.

C++ Examples:

```
/*
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Timer1->Interval = 100;
  if (ActiveControl)
    ActiveControl->Left = ActiveControl->Left + 1;
}
```

Delphi Examples:

```
{
Place a TTimer object in the form and enter Timer1Timer in
the OnTimer event.  Place other controls in the form and
change the active control at runtime.  The following event
handler responds to timer events by moving the active control
one pixel to the right every 100 miliseconds:
}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  if ActiveControl <> nil then
    ActiveControl.Left := ActiveControl.Left + 1;
end;
```

See Also

ActiveOleControl (see page 13897)

OnExit (see page 6221)

OnEnter (see page 6219)

Active (see page 13896)

72.1.7.3.3 TForm.ActiveMDIChild Property

Specifies the MDI child that has focus.

Pascal

```
property ActiveMDIChild: TForm;
```

C++

```
__property TForm * ActiveMDIChild;
```

Description

Use ActiveMDIChild to get the MDI child that has focus.

If the form is not an MDI parent (that is, if the FormStyle property of the form is not fsMDIForm), ActiveMDIChild returns nil (Delphi) or NULL (C++).

See Also

FormStyle (see page 13912)

MDIChildCount (see page 13921)

72.1.7.3.4 TForm.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
```

```
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}
```

```
var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

72.1.7.3.5 TForm.AlphaBlend Property

Specifies whether the form is translucent.

Pascal

```
property AlphaBlend: Boolean;
```

C++

```
__property Boolean AlphaBlend;
```

Description

Set AlphaBlend to specify that the form represents a layered window that allows a translucent color. The AlphaBlendValue property specifies the degree of translucency.

Note: AlphaBlend does not work on all systems. The application must be running under Windows 2000 or better, and the machine must be a P90 or better.

See Also

AlphaBlendValue (see page 13898)

TransparentColor (see page 13938)

72.1.7.3.6 TForm.AlphaBlendValue Property

Specifies the degree of translucency on a translucent form.

Pascal

```
property AlphaBlendValue: Byte;
```

C++

```
__property Byte AlphaBlendValue;
```

Description

Set AlphaBlendValue to a value between 0 and 255 to indicate the degree of translucency when the AlphaBlend property is true. A value of 0 indicates a completely transparent window. A value of 255 indicates complete opacity.

Note: AlphaBlendValue only has an effect when the AlphaBlend property is true.

See Also

AlphaBlend (see page 13898)

TransparentColor (see page 13938)

TransparentColorValue (see page 13939)

72.1.7.3.7 TForm.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

72.1.7.3.8 TForm.AutoScroll Property

Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

If AutoScroll is true, the scroll bars appear automatically when necessary. For example, if the user resizes the control so that some of its controls are partially obscured, scroll bars appear.

If AutoScroll is false, scroll bars don't appear automatically. In this case, use the HorzScrollBar and VertScrollBar properties to make scroll bars appear.

See Also

AutoScrollInView (see page 14325)

HorzScrollBar (see page 14329)

VertScrollBar (see page 14330)

72.1.7.3.9 TForm.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints (see page 5812)

CanAutoSize (see page 5741)

72.1.7.3.10 TForm.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

72.1.7.3.11 TForm.BorderIcons Property

Specifies which icons appear on the title bar of the form.

Pascal

```
property BorderIcons: TBorderIcons;
```

C++

```
__property TBorderIcons BorderIcons;
```

Description

Use BorderIcons to get or set the icons that appear on the title bar of the form. BorderIcons can include any of the following TBorderIcons values:

Value	Meaning
biSystemMenu	The form has a Control menu (also known as a System menu).

biMinimize	The form has a Minimize button
biMaximize	The form has a Maximize button
biHelp	If BorderStyle is bsDialog or biMinimize and biMaximize are excluded, a question mark appears in the form's title bar and when clicked, the cursor changes to crHelp; otherwise,no question mark appears.

Note: Certain combinations of the BorderIcons and BorderStyle properties are mutually exclusive. For example, BorderIcons biMax, biMin with BorderStyle of bsDialog are mutually exclusive.

Delphi Examples:

```
{
BorderIcons example
The following code removes a form’s Maximize button when the user clicks a button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
BorderIcons := BorderIcons - [biMaximize];
end;
```

See Also

BorderStyle (see page 13900)

72.1.7.3.12 TForm.BorderStyle Property

Specifies the appearance and behavior of the form border.

Pascal

```
property BorderStyle: TFormBorderStyle;
```

C++

```
__property TFormBorderStyle BorderStyle;
```

Description

Use BorderStyle to get or set the appearance and behavior of the form border. BorderStyle can be any of the following TForm BorderStyle values:

Value	Meaning
bsDialog	Not resizable; no minimize/maximize menu
bsSingle	Not resizable; minimize/maximize menu
bsNone	Not resizable; no visible border line
bsSizeable	Standard resizable border
bsToolWindow	like bsSingle but with a smaller caption
bsSizeToolWin	like bsSizeable with a smaller caption

Note: Changing the border style of an MDI child form to bsDialog or bsNone has no effect.

See Also

BorderIcons (see page 13899)

72.1.7.3.13 TForm.BorderWidth Property

Specifies the width of the control's border.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to get or set the width of the control's border. Graphics or text drawn by the control is clipped to the area within the border.

See Also

ClientRect

BevelOuter (see page 6194)

72.1.7.3.14 TForm.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
```

```
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

72.1.7.3.15 TForm.ClientHandle Property

Provides access to the handle of the form's internal MDI client window.

Pascal

```
property ClientHandle: HWND;
```

C++

```
__property HWND ClientHandle;
```

Description

Use ClientHandle to get the handle for the form's internal MDI client window.

ClientHandle is meaningful only if the form is an MDI parent (that is, if the form's FormStyle property is set to fsMDIForm).

Note: Use the Handle property instead to get the handle to the form window itself.

See Also

FormStyle (see page 13912)

Handle (see page 6207)

72.1.7.3.16 TForm.ClientHeight Property

Specifies the height (in pixels) of the form's client area.

Pascal

```
property ClientHeight: Integer;
```

C++

```
__property int ClientHeight;
```

Description

Use ClientHeight to determine the height (in pixels) of the form's client area. The client area is the usable area inside the form's border, excluding the title bar, scroll bars, and so on.

Set ClientHeight to change the height of the form's window based on the desired client area. To change the height of the form's window based on the total size of the window (including the border, menu, status bar and so on), use the Height property instead.

Delphi Examples:

```
{
This example reduces the height of the form's client area by
half when the user clicks the button on the form:
Note that only the client area is halved, not the entire form.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ClientHeight := ClientHeight div 2;
end;
```

See Also

ClientRect

ClientWidth (see page 13905)

Height (see page 5825)

72.1.7.3.17 TForm.ClientWidth Property

Specifies the width (in pixels) of the form's client area.

Pascal

```
property ClientWidth: Integer;
```

C++

```
__property int ClientWidth;
```

Description

Use ClientWidth to determine the width (in pixels) of the form's client area. The client area is the usable area inside the form's border. Set ClientWidth to change the width of the form's window based on the desired client area. To change the width of the form's window based on the total size of the window (including the border, status bar and so on), use the Width property instead.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
```

```
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form’s logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
    Range := 600;
    Position := Range - ClientWidth; { start form out fully scrolled }
    Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
    Visible := True;  { Show the scrollbar }
  end;
end;
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form’s width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
```

```
end;
```

See Also

ClientHeight (see page 13905)

ClientRect

Width (see page 5849)

72.1.7.3.18 TForm.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

72.1.7.3.19 TForm.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

72.1.7.3.20 TForm.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

72.1.7.3.21 TForm.DefaultMonitor Property

Specifies the monitor on which the form appears.

Pascal

```
property DefaultMonitor: TDefaultMonitor;
```

C++

```
__property TDefaultMonitor DefaultMonitor;
```

Description

Use DefaultMonitor to associate a form with a particular monitor in a multi-monitor application. The following table lists the possible values:

Value	Meaning
dmDesktop	No attempt is made to position the form on a specific monitor.
dmPrimary	The form is positioned on the first monitor listed in the global screen object's Monitors property.
dmMainForm	The form appears on the same monitor as the application's main form.
dmActiveForm	The form appears on the same monitor as the currently active form.

Note: DefaultMonitor has no effect if the application does not have a main form.

See Also

Monitors (see page 14228)

TMonitor (see page 14191)

Screen (see page 14385)

MainForm (see page 13718)

Active (see page 13896)

Monitor (see page 13926)

MakeFullyVisible (see page 13881)

72.1.7.3.22 TForm.DockManager Property

Specifies the control's docking manager interface.

Pascal

```
property DockManager: IDockManager;
```

C++

```
__property IDockManager DockManager;
```

Description

Use DockManager to specify the control's docking manager. The docking manager handles the layout of docking zones (where controls are docked) and any painting associated with docking zones.

If you set DockSite and UseDockManager to true, but do not assign a value to DockManager, the windowed control generates a

default docking manager, using the global DefaultDockTreeClass variable.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
```

```
    Source: TDragDockObject; X, Y: Integer);
  begin
    //OnDockDrop gets called AFTER the client has actually docked,
    //so we check for DockClientCount = 1 before making the dock panel visible.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, True, nil);
    (Sender as TPanel).DockManager.ResetBounds(True);
    //Make DockManager repaints it's clients.
  end;

  procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer; State: TDragState;
    var Accept: Boolean);
  var
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

UseDockManager (see page 6212)

DefaultDockTreeClass (see page 6321)

72.1.7.3.23 TForm.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite: Boolean;
```

C++

```
__property Boolean DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

72.1.7.3.24 TForm.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

72.1.7.3.25 TForm.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
```

```
  drop.
  }
  procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
  var
    AnItem: TTreeNode;
    AttachMode: TNodeAttachMode;
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

72.1.7.3.26 TForm.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

72.1.7.3.27 TForm.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;
```

```
procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

72.1.7.3.28 TForm.FormStyle Property

Determines the form's style.

Pascal

```
property FormStyle: TFormStyle;
```

C++

```
__property TFormStyle FormStyle;
```

Description

Use FormStyle to get or set the form's style. FormStyle is one of the following values:

Value	Meaning
fsNormal	The form is neither an MDI parent window nor an MDI child window.
fsMDIChild	The form is an MDI child window.
fsMDIForm	The form is an MDI parent window.
fsStayOnTop	This form remains on top of the desktop and of other forms in the project, except any others that also have FormStyle set to fsStayOnTop. If one fsStayOnTop form launches another, neither form will consistently remain on top.

If the form is the main form of an MDI application, its FormStyle property must be set to fsMDIForm.

Note: It is not advisable to change FormStyle at runtime.

Warning: TGraphicControl descendants placed in the client area of a form with FormStyle set to fsMDIForm will not paint.

C++ Examples:

```
/*
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
*/
void __fastcall TForm1::MyArrangeIconsClick(TObject *Sender)
{
```

```
  Form1->ArrangeIcons();
}
/*
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
*/
void __fastcall TForm1::TileFormsClick(TObject *Sender)
{
  TileMode = tbVertical;
  Tile();
}
```

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
}
procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
  Form1.ArrangeIcons;
end;
{
This example ensures that the main form of the application
is an MDI parent:
}
type
  TForm1 = class(TForm)
    Edit1: TEdit;
  private
    { Private declarations }
  public
    { Public declarations }
    constructor Create(AOwner : TComponent); override;
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

constructor TForm1.Create(AOwner : TComponent);

begin
  inherited Create(AOwner);
  if FormStyle <> fsMDIForm then
          FormStyle := fsMDIForm;
  if FormStyle = fsMDIForm then
          Edit1.Text := 'MDI form'
  else
    //This line never executes
```

```
    Edit1.Text := 'Not an MDI form';
end;
{
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
}
procedure TForm1.TileFormsClick(Sender: TObject);
begin
  TileMode := tbVertical;
  Tile;
end;
```

See Also

ArrangeIcons (see page 14006)

Tile (see page 14009)

Cascade (see page 14007)

Windows Overview

72.1.7.3.29 TForm.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
```

```
  }
}
```

Delphi Examples:

```
{
HideSelection, Height example
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
        HideSelection := False;
    end;
end;
```

See Also

ClientHeight (see page 5808)

Width (see page 5849)

Using TDBListBox and TDBComboBox

72.1.7.3.30 TForm.HelpFile Property

Specifies the name of the file the form uses to display Help.

Pascal

```
property HelpFile: string;
```

C++

```
__property AnsiString HelpFile;
```

Description

Use HelpFile for forms that have a help file to use with the Help system. Help topics are selected from the specified file.

See Also

HelpContext (see page 5826)

OnHelp (see page 13959)

HelpFile (see page 13711)

HelpKeyword (see page 5826)

72.1.7.3.31 TForm.HorzScrollBar Property

Represents the horizontal scroll bar for the scrolling windowed control.

Pascal

```
property HorzScrollBar: TControlScrollBar;
```

C++

```
__property TControlScrollBar HorzScrollBar;
```

Description

Use HorzScrollBar to hide, show, or manipulate the horizontal scroll bar for the scrolling windowed control.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form's logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
    Range := 600;
    Position := Range - ClientWidth; { start form out fully scrolled }
    Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
    Visible := True;  { Show the scrollbar }
  end;
end;
```

See Also

AutoScroll (see page 14329)

VertScrollBar (see page 14330)

TControlScrollBar (see page 13744)

72.1.7.3.32 TForm.Icon Property

Specifies the icon that appears when the form is minimized.

Pascal

```
property Icon: TIcon;
```

C++

```
__property TIcon Icon;
```

Description

Set the Icon property to provide an icon for the form. If Icon is not set, the system provides a default icon when the form is minimized.

Delphi Examples:

```
{
This code assigns an icon to a form when the form is created:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Icon.LoadFromFile('MDIChild1.ICO');
end;
```

See Also

WindowState (see page 13942)

72.1.7.3.33 TForm.KeyPreview Property

Specifies whether the form should receive keyboard events before the active control.

Pascal

```
property KeyPreview: Boolean;
```

C++

```
__property Boolean KeyPreview;
```

Description

If KeyPreview is true, keyboard events occur on the form before they occur on the active control. (The active control is specified by the ActiveControl property.)

If KeyPreview is false, keyboard events occur only on the active control.

Navigation keys (Tab, BackTab, the arrow keys, and so on) are unaffected by KeyPreview because they do not generate keyboard events. Similarly, when a button has focus or when its Default property is true, the Enter key is unaffected by KeyPreview because it does not generate a keyboard events.

KeyPreview is false by default.

C++ Examples:

```
/*
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
```

```
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
```

```
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
  {
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

ActiveControl (see page 13896)

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

OnKeyPress (see page 6227)

72.1.7.3.34 TForm.MDIChildCount Property

Specifies the number of open MDI child forms.

Pascal

```
property MDIChildCount: Integer;
```

C++

```
__property int MDIChildCount;
```

Description

Use MDIChildCount to get the number of open MDI child forms.

MDIChildCount is meaningful only if the form is an MDI parent (that is, if the form's FormStyle property is set to fsMDIForm).

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items CloseChildren. This is code for the
CloseChildrenClick handler.  When the user chooses the
CloseChildren command, all the MDI children of Form1 are
closed.
}
procedure TForm1.CloseChildrenClick(Sender: TObject);
var
  I: Integer;
begin
  with Form1 do
    for I := MDIChildCount-1 downto 0 do
      MDIChildren[I].Close;
end;

procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
  Form1.ArrangeIcons;
end;

procedure TForm1.MyCascadeClick(Sender: TObject);
begin
  Form1.Cascade;
end;
```

See Also

MDIChildren (see page 13922)

72.1.7.3.35 TForm.MDIChildren Property

Provides indexed access to all MDI child forms.

Pascal

```
property MDIChildren [I: Integer]: TForm;
```

C++

```
__property TForm * MDIChildren[int I];
```

Description

Use MDIChildren to access a child form.

I is the index of the child form to access. The order of MDI children changes as different MDI forms are activated. Initially, forms are listed in creation order, with new forms being added to the end of the array. Whenever an MDI child is activated, however, it moves to the front of the list. Thus, use I to index over all forms in the array, but not to refer to a specific form.

MDIChildren is meaningful only if the form is an MDI frame (that is, if the form's FormStyle property is set to fsMDIForm).

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items CloseChildren. This is code for the
CloseChildrenClick handler.  When the user chooses the
CloseChildren command, all the MDI children of Form1 are
closed.
}
procedure TForm1.CloseChildrenClick(Sender: TObject);
var
  I: Integer;
begin
  with Form1 do
    for I := MDIChildCount-1 downto 0 do
      MDIChildren[I].Close;
end;

procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
  Form1.ArrangeIcons;
end;

procedure TForm1.MyCascadeClick(Sender: TObject);
begin
  Form1.Cascade;
end;
```

See Also

MDIChildCount (see page 13921)

FormStyle (see page 13912)

72.1.7.3.36 TForm.Menu Property

Specifies the form's main menu.

Pascal

```
property Menu: TMainMenu;
```

C++

```
__property TMainMenu Menu;
```

Description

Use Menu to get or set the form's main menu. At design time, Menu is set to the first TMainMenu component added to the form.

C++ Examples:

```
/*
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *Item1 = NewItem("New item", TextToShortCut("Ctrl+N"), false, true, MyOnExecute,
0, "Item1");
  Menu = NewMenu(this, "MyMenu", &Item1, 0);
}

void __fastcall TForm1::MyOnExecute(TObject *Sender)
{
  MessageDlg("Item1 has been executed.", mtInformation, TMsgDlgButtons() << mbOK, 0);
}
/*
To create a toolbar that corresponds to an existing menu:

1   Drop a ToolBar on the form and add a ToolButton for each
  top-level menu item you wish to create.
2   Set the MenuItem property of each ToolButton to correspond
  to the top level menu items.
3   Set the Grouped property of each ToolButton to true.
4   Clear the Menu property of the Form (if it is assigned)
5   Set the Visible property on the ToolBar and ToolButtons to
  true.
6   Set the ShowCaptions property on the ToolBar to true.
7   Set the MenuItem | Visible property of each ToolButton to
  true.
*/
```

Delphi Examples:

```
{
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Menu := NewMenu(Self, 'MyMenu',
            [NewItem('New item', TextToShortCut('Ctrl+N'),
            False, True, MyOnExecute, 0, 'Item1')]);
end;

procedure TForm1.MyOnExecute(Sender: TObject);
begin
  MessageDlg('Item1 has been executed.', mtInformation, [mbOK], 0);
end;
{
To create a toolbar that corresponds to an existing menu:

1   Drop a ToolBar on the form and add a ToolButton for each top-level menu item you wish to
create.
2   Set the MenuItem property of each ToolButton to correspond to the top level menu items.
3   Set the Grouped property of each ToolButton to true.
4   Clear the Menu property of the Form (if it is assigned)
5   Set the Visible property on the ToolBar and ToolButtons to true.
6   Set the ShowCaptions property on the ToolBar to true.
7   Set the MenuItem | Visible property of each ToolButton to true.
}
```

See Also

TMainMenu (see page 16056)

72.1.7.3.37 TForm.ObjectMenuItem Property

Represents an OLE object menu item that reacts to selections of OLE objects.

Pascal

```
property ObjectMenuItem: TMenuItem;
```

C++

```
__property TMenuItem ObjectMenuItem;
```

Description

Use ObjectMenuItem to get or set a menu item that becomes enabled or disabled when an OLE object on the form is selected or unselected respectively.

The OLE object menu item can be used for standard OLE commands, such as Activate or Convert. (The implementations of these commands are provided by the OLE server.)

See Also

OleFormObject (see page 13927)

72.1.7.3.38 TForm.OldCreateOrder Property

Specifies when OnCreate and OnDestroy events occur.

Pascal

```
property OldCreateOrder: Boolean;
```

C++

```
__property Boolean OldCreateOrder;
```

Description

When OldCreateOrder is false (the default) the OnCreate event occurs after all constructors are finished and the OnDestroy event occurs before any destructors are called.

This differs from earlier versions of the library (distributed with Delphi version 3 and previous and C++Builder 1), where the OnCreate event occurred when the TCustomForm constructor executed and the OnDestroy event occurred when the TCustomForm destructor executed. Applications that require the OnCreate event and OnDestroy events to occur from the constructor and destructor of the form can set OldCreateOrder to true.

See Also

OnCreate (see page 13951)

OnDestroy (see page 13953)

BeforeDestruction (see page 13866)

AfterConstruction (see page 13866)

72.1.7.3.39 TForm.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding: TPadding;
```

C++

```
__property TPadding Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding (see page 6103)

72.1.7.3.40 TForm.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

72.1.7.3.41 TForm.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

72.1.7.3.42 TForm.PixelsPerInch Property

Represents the proportion of the font on the system on which the form was designed.

Pascal

```
property PixelsPerInch: Integer;
```

C++

```
__property int PixelsPerInch;
```

Description

Use PixelsPerInch at runtime to change how the form is scaled to the current screen resolution. If PixelsPerInch is changed from its default value, the form will not have the same proportion on every screen resolution.

At design time, PixelsPerInch is set automatically when the form is saved. Changes to PixelsPerInch made in the Object Inspector only affect the form at runtime.

The form's Scaled property must be true for PixelsPerInch to have an effect.

See Also

Scaled (see page 13935)

72.1.7.3.43 TForm.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
```

```
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
```

```
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
```

```
    Edit1.PopupMenu := PopupMenu1;
    Edit2.PopupMenu := PopupMenu1;
  end;

  {
  These are the cut, copy, and paste OnClick events for the commands
  on the pop-up menu. Double click on the pop-up menu and add three
  TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
  The names will be automatically set to "Cut1", "Copy1", and
  "Paste1".  Then double click on the OnClick events and they will
  be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
  Note: only the selected portion of the string in the edit box will
  be copied, cut or replaced.
  }
  procedure TForm1.Copy1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
     Edit1.CopyToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
```

```
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

72.1.7.3.44 TForm.PopupMode Property

Controls (see page 5711) how the top-level form behaves with respect to Window's WS_POPUP style.

Pascal

```
property PopupMode: TPopupMode;
```

C++

```
__property TPopupMode PopupMode;
```

Description

The PopupMode property controls how the top-level form behaves with respect to Window's WS_POPUP style. A window that has the WS_POPUP style is always above its "owner" in Z-order. You can use the PopupMode property in conjunction with the PopupParent (see page 14054) property to avoid the appearance of a hung application caused by a modal dialog showing up behind another form on the screen.

Note: Using the PopupMode property is similar to specifying fsStayOnTop for the FormStyle (see page 14036) property, but allows better control over the layering.

The PopupMode property is automatically set to pmAuto when the ShowModal (see page 13890) method is called. However, this causes the window handle to be recreated, which is not usually desirable. To avoid the re-creation of window handles, you can explicitly set the PopupMode property to pmAuto prior to calling the ShowModal (see page 13890) method (such as at design time).

Set the PopupMode property to pmExplicit for non-modal design windows such as tool palettes and other floating tool windows. This causes the windows to always remain on top of the main form. In the case of a snapped together design (in which the window is docked to the main form), the design window remains on top of the designer.

Note: For undocked windows, the design window remains above the main form, allowing other top-level forms to obscure the design window.

Set thePopupMode property to pmNone if you want the popup window to have the pre-Delphi 8 behavior (except for the ShowModal (see page 13890) item mentioned above).

You can force the Z-Ordering of your forms and create a "stacked" appearance that the user cannot change by setting the PopupParent (see page 14054) property to an explicit TCustomForm (see page 13849). If the PopupMode property is pmExplicit and PopupParent (see page 14054) is nil, then the Application.MainForm is implicitly used as the PopupParent (see page 14054). If no Application.MainForm is assigned, then Application.Handle is used as the PopupParent (see page 14054). If the PopupMode property is set to pmAuto, Screen.ActiveForm is used as the PopupParent (see page 14054).

72.1.7.3.45 TForm.PopupParent Property

Sets an order for stacked forms that users cannot change.

Pascal

```
property PopupParent: TCustomForm;
```

C++

```
__property TCustomForm PopupParent;
```

Description

If the PopupMode property is set to pmExplicit and PopupParent is nil, then the Application.MainForm is implicitly used as the

PopupParent. If no Application.MainForm is assigned, then Application.Handle is used as the PopupParent.

If the PopupMode property is set to pmAuto, Screen.ActiveForm is used as the PopupParent property.

72.1.7.3.46 TForm.Position Property

Represents the size and placement of the form.

Pascal

```
property Position: TPosition;
```

C++

```
__property TPosition Position;
```

Description

Use Position to get or set the size and placement of the form. Position can have one of the following TPosition values:

Value	Meaning
poDesigned	The form appears positioned on the screen and with the same height and width as it had at design time.
poDefault	The form appears in a position on the screen and with a height and width determined by the operating system. Each time you run the application, the form moves slightly down and to the right. The right side of the form is always near the far right side of the screen, and the bottom of the form is always near the bottom of the screen, regardless of the screen's resolution. The operating system only supports this for overlapped windows. For a pop-up or child window (that has a BorderStyle of bsDialog or bsNone, for example), the window is positioned in the upper left corner of the screen.
poDefaultPosOnly	The form displays with the size you created it at design time, but the operating system chooses its position on the screen. Each time you run the application, the form moves slightly down and to the right. When the form can no longer move down and to the right and keep the same size while remaining entirely visible on the screen, the form displays at the top-left corner of the screen. The operating system only supports this for overlapped windows. For a pop-up or child window (that has a BorderStyle of bsDialog or bsNone, for example), the window is positioned in the upper left corner of the screen.
poDefaultSizeOnly	The form appears in the position you left it at design time, but the operating system chooses its size. The right side of the form is always near the far right side of the screen, and the bottom of the form is always near the bottom of the screen, regardless of the screen's resolution.
poScreenCenter	The form remains the size you left it at design time, but is positioned in the center of the screen. In multi-monitor applications, the form may be moved from this center position so that it falls entirely on one monitor, as specified by the DefaultMonitor property.
poDesktopCenter	The form remains the size you left it at design time, but is positioned in the center of the screen. No adjustments are made for multi-monitor applications.
poMainFormCenter	The form remains the size you left it at design time, but is positioned in the center of the application's main form. No adjustments are made for multi-monitor applications. This position should only be used with secondary forms. If set for a main form, it acts like poScreenCenter.
poOwnerFormCenter	The form remains the size you left it at design time, but is positioned in the center of the form specified by the Owner property. If the Owner property does not specify a form, this position acts like poMainFormCenter.

See Also

OnResize (see page 5870)

SetInitialBounds

72.1.7.3.47 TForm.PrintScale Property

Represents the proportions of a printed form.

Pascal

```
property PrintScale: TPrintScale;
```

C++

```
__property TPrintScale PrintScale;
```

Description

Use PrintScale to get or set the proportions of the printed form. PrintScale can have one of the following TPrintScale values:

Value	Meaning
poNone	No special scaling occurs; therefore, the printed form and how the form appears onscreen may appear squished or stretched.
poProportional	The form is printed so that the printed image is approximately the same visible size as on the screen (WYSIWYG). The form image is scaled so that the printed image is approximately the same visible size as on the screen.
poPrintToFit	The form is printed using the same screen proportions, but in a size that just fits the printed page.

See Also

Print (see page 13883)

72.1.7.3.48 TForm.Scaled Property

Specifies whether the form is sized according to the value of the PixelsPerInch property.

Pascal

```
property Scaled: Boolean;
```

C++

```
__property Boolean Scaled;
```

Description

Scaled determines whether the form adjusts itself to differences between the font used at design time and the current system font used by a system at runtime. If Scaled is true, the form resizes itself and all its child controls to maintain the relationship between the size of controls and the height of text displayed in the default font. If Scaled is false, no such resizing occurs.

By setting Scaled to true, the form can adjust to fonts that are not scalable (that is, fonts that only supply discrete point sizes) or to differences such as forms designed using small fonts at design time that are run on systems that use large fonts.

The degree of resizing is determined by the PixelsPerInch property, which measures the proportion of the font on the system used when designing the form.

See Also

PixelsPerInch (see page 13928)

Considerations When Dynamically Resizing Forms and Controls

72.1.7.3.49 TForm.ScreenSnap Property

Specifies whether form snaps to edge of screen.

Pascal

```
property ScreenSnap: Boolean;
```

C++

```
__property Boolean ScreenSnap;
```

Description

ScreenSnap determines whether the edges of the form snap to the edge of the screen when the user moves the form. To control the snap distance, set the SnapBuffer property.

See Also

SnapBuffer (see page 13936)

72.1.7.3.50 TForm.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
```

```
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

72.1.7.3.51 TForm.SnapBuffer Property

Specifies distance for screen snap.

Pascal

```
property SnapBuffer: Integer;
```

C++

```
__property int SnapBuffer;
```

Description

If ScreenSnap is true, SnapBuffer determines the maximum number of pixels that be between the form edge and the screen edge before the form snaps to the screen edge.

See Also

ScreenSnap (see page 13935)

72.1.7.3.52 TForm.TileMode Property

Represents how MDI child forms are arranged when the Tile method is called.

Pascal

```
property TileMode: TTileMode;
```

C++

```
__property TTileMode TileMode;
```

Description

TileMode can have one of the following TTileMode values:

Value	Meaning
tbHorizontal	Each form stretches across the width of the parent form
tbVertical	Each form stretches along the height of the parent form

TileMode is meaningful only if the form is an MDI child (that is, if the form's FormStyle property is set to fsMDIChild).

C++ Examples:

```
/*
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
*/
void __fastcall TForm1::TileFormsClick(TObject *Sender)
{
  TileMode = tbVertical;
  Tile();
}
```

Delphi Examples:

```
{
This example uses three forms. The first form has its
FormStyle property set to MDIForm. The other two have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items TileForms. This is code for the
TileFormsClick handler.  When the user chooses the TileForms
command, the child forms tile vertically within the MDI frame
form.
}
procedure TForm1.TileFormsClick(Sender: TObject);
begin
  TileMode := tbVertical;
  Tile;
end;
```

See Also

FormStyle (see page 13912)

Tile (see page 14009)

72.1.7.3.53 TForm.TransparentColor Property

Specifies whether a color on the form appears transparent.

Pascal

```
property TransparentColor: Boolean;
```

C++

```
__property Boolean TransparentColor;
```

Description

Use TransparentColor to indicate that one of the colors on the form should be treated as transparent, allowing windows behind the form to completely show through. The TransparentColorValue property indicates the color that appears completely transparent.

Note: To make the entire form transparent, or to make it translucent rather than transparent, use the AlphaBlend and AlphaBlendValue properties.

Note: TransparentColor does not work on all systems. The application must be running under Windows 2000 or better, with adequate hardware.

See Also

TransparentColorValue (see page 13939)

AlphaBlend (see page 13898)

AlphaBlendValue (see page 13898)

72.1.7.3.54 TForm.TransparentColorValue Property

Indicates the color on the form that appears transparent when TransparentColor is true.

Pascal

```
property TransparentColorValue: TColor;
```

C++

```
__property TColor TransparentColorValue;
```

Description

Use TransparentColorValue to indicate the color that appears transparent when the TransparentColor property is true.

Note: If the application uses a 16-bit color palette, TransparentColorValue must be a basic color.

See Also

TransparentColor (see page 13938)

72.1.7.3.55 TForm.UseDockManager Property

Specifies whether the docking manager is used in drag-and-dock operations.

Pascal

```
property UseDockManager: Boolean;
```

C++

```
__property Boolean UseDockManager;
```

Description

Use UseDockManager to get or set whether a docking manager is used in drag-and-dock operations.

The docking manager handles the positioning of docked controls and any painting of docking zones around those controls. If you do not use a docking manager, docked controls are aligned to the windowed control based on the closest edge to where they are released. When you do not use a docking manager, there is no region for a user to grab to automatically undock a docked control.

See Also

DockManager (see page 6202)

72.1.7.3.56 TForm.VertScrollBar Property

Represents the vertical scroll bar for the scrolling windowed control.

Pascal

```
property VertScrollBar: TControlScrollBar;
```

C++

```
__property TControlScrollBar VertScrollBar;
```

Description

Use VertScrollBar to hide, show, or manipulate the vertical scroll bar for the scrolling windowed control.

C++ Examples:

```
/*
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  const int iPageDelta = 10;

  if (Key == VK_NEXT)
    VertScrollBar->Position += iPageDelta;
  else if (Key == VK_PRIOR)
    VertScrollBar->Position -= iPageDelta;
}
```

Delphi Examples:

```
{
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
}
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
const
  PageDelta = 10;
begin
  With VertScrollbar do
    if Key = VK_NEXT then
      Position := Position + PageDelta
    else if Key = VK_PRIOR then
      Position := Position - PageDelta;
end;
```

See Also

AutoScroll (see page 14329)

HorzScrollBar (see page 14329)

TControlScrollBar (see page 13744)

72.1.7.3.57 TForm.Visible Property

Indicates whether the form is visible.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use Visible to determine whether the form is visible.

If Visible is true, the form is visible unless it is completely obscured by other forms. If Visible is false, the form is not visible.

The Show and ShowModal methods set Visible to true and bring the form to the front of all open windows.

C++ Examples:

```
/*
The following code uses two forms. Form1 has a button on it. The
second form is used as a tool palette. This code makes the palette
form visible, and ensures it is the top form by bringing it to the
front.  To run this example, you must include Unit2.h in this
module.
*/
#include "Unit2.h"

void __fastcall TForm1::ShowPaletteButtonClick(TObject *Sender)
{
  if (!Form2->Visible)
  {
    Form2->Visible = true;
    Form2->BringToFront();
  }
}
```

Delphi Examples:

```
{
The following code uses two forms. Form1 has a button on it. The
second form is used as a tool palette. This code makes the palette
form visible, and ensures it is the top form by bringing it to the
front.  To run this example, you must add Unit2 in the uses clause
of your unit.
}
procedure TForm1.ShowPaletteButtonClick(Sender: TObject);
begin
if Form2.Visible = False then Form2.Visible := True;
Form2.BringToFront;
end;
```

See Also

BringToFront (see page 5740)

Show (see page 13888)

ShowModal (see page 13890)

72.1.7.3.58 TForm.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

See Also

ClientWidth (see page 5811)

Height (see page 5825)

Left (see page 5830)

72.1.7.3.59 TForm.WindowMenu Property

Specifies the Window menu for an MDI parent form.

Pascal

```
property WindowMenu: TMenuItem;
```

C++

```
__property TMenuItem WindowMenu;
```

Description

Use WindowMenu to get or set the Window menu for an MDI parent form. The Window menu is a standard menu in MDI applications. It contains commands that let the user manage the windows in the application. Menu items usually include Cascade, Arrange Icons, Tile, and so on.

The Window menu also lists (at the bottom) the child windows that are currently open in the application. When the user selects one of these windows from the menu, the window becomes the active window in the application.

Although this menu is commonly called the Window menu, it can have any name that corresponds to an existing menu item on the menu bar. The names of open child forms are merged onto the menu automatically at run-time.

WindowMenu is meaningful only if the form is an MDI parent (that is, if the form's FormStyle property is set to fsMDIForm).

See Also

FormStyle (see page 13912)

72.1.7.3.60 TForm.WindowState Property

Represents how the form appears on the screen.

Pascal

```
property WindowState: TWindowState;
```

C++

```
__property TWindowState WindowState;
```

Description

Set WindowState to minimize, maximize, or restore the form window. Read WindowState to determine whether the form is minimized, maximized, or in a normal state.

See Also

Icon (see page 13917)

BorderStyle (see page 13900)

72.1.7.4 TForm Events

The events of the TForm class are listed here.

Published Events

	Name	Description
	OnActivate (see page 14065)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 14069)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14069)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 14070)	Occurs when an attempt is made to resize the control.
	OnClick (see page 14070)	Occurs when the user clicks the control.
	OnClose (see page 14072)	Occurs when the form closes.
	OnCloseQuery (see page 14073)	Occurs when close is attempted.
	OnConstrainedResize (see page 14074)	Adjust resize constraints.
	OnContextPopup (see page 14075)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 14076)	Occurs when the form is created.
	OnDblClick (see page 14077)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 14078)	Occurs when the form loses focus.
	OnDestroy (see page 14078)	Occurs when the form is destroyed.
	OnDockDrop (see page 14079)	Occurs when another control is docked to the control.
	OnDockOver (see page 14081)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 14083)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 14085)	Occurs when the user drags an object over a control.
	OnEndDock (see page 14088)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnGetSiteInfo (see page 14088)	Returns the control's docking information.
	OnHelp (see page 14090)	Occurs when the form receives a request for help.
	OnHide (see page 14091)	Occurs when the form is hidden (that is, when its Visible property is set to false).
	OnKeyDown (see page 14093)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 14095)	Occurs when a key is pressed.
	OnKeyUp (see page 14096)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 14098)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 14099)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 14101)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 14102)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 14102)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 14104)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

	OnMouseWheel (see page 14107)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 14107)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 14107)	Occurs when the mouse wheel is rotated upward.
	OnPaint (see page 14108)	Occurs when the form is redrawn.
	OnResize (see page 14109)	Occurs immediately after the control is resized.
	OnShortCut (see page 14109)	Occurs when the user presses a key (before the OnKeyDown event).
	OnShow (see page 14110)	Occurs when the form is shown (that is, when its Visible property is set to true).
	OnStartDock (see page 14111)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnUnDock (see page 14111)	Occurs when the application tries to undock a control that is docked to the windowed control.

72.1.7.4.1 TForm.OnActivate Event

Occurs when the form becomes active.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Use OnActivate to perform special processing when the form receives focus. A form becomes active when focus is transferred to it (when the user clicks on the form, for example).

Note: The OnActivate event of the application (TApplication), not the form, occurs when Windows switches control from another application.

Delphi Examples:

```
{
Here is the code on Form1 that fills the list box.
}
var
  Form1: TForm1;
  Customers: TTable;
  Blobs: TTable;
  Form1Done : Boolean;
  procedure CreateMyTable;
  procedure CreateCustSource;
  procedure CreateBlobSource;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  CreateCustSource;
  CreateBlobSource;
  for I := 0 to ComponentCount - 1 do
  begin
    if (Components[I] is TDataSource) then
      ListBox1.Items.AddObject(Components[I].Name,
        Components[I] as TObject);
  end;
end;
```

```
procedure CreateCustSource();
begin
  Customers:= TTable.Create(Form1);
  with Customers do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'MyCustInfo';
    ReadOnly:= False;

//    Don't overwrite an existing table
    if (not Customers.Exists) then CreateMyTable
    else
    begin
      if (Customers.Exists AND
        (MessageDlg('MyCustInfo table already exists.  Do you want to rebuild it?',
mtConfirmation, [mbYes, mbNo], 0) = mrYes)) then
      begin
        Customers.Close;
        Customers.DeleteTable;
        CreateMyTable;
      end;
    end;
  end;
  Form1.CustomerDS.DataSet:= Customers;
  Customers.Active:= True;
end;

procedure CreateMyTable();
var
  i: Integer;
begin
//      The Table component must not be active
//      First, describe the type of table and give
//      it a name
//      Next, describe the fields in the table
  with Customers do
  begin
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
        with AddFieldDef do
        begin
          Name := 'Field3';
          DataType := ftString;
          Size := 30;
        end;
        with AddFieldDef do
        begin
          Name := 'Field4';
          DataType := ftString;
          Size := 30;
        end;
```

```
      end;

//    Take out this part to remove indexing
//    Next, describe any indexes
      with IndexDefs do
      begin
        Clear;
//        The 1st index has no name because it is
//        a Paradox primary key
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
      end;

//      Call the CreateTable method to create the table
      CreateTable;
      Customers.Active:= True;
      for i := 1 to 20 do
        Customers.AppendRecord([i*111, i*222, i*333, i*444]);
    end;
end;

procedure CreateBlobSource;
var
  i: Integer;
begin
  Blobs:= TTable.Create(Form1);
  with Blobs do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'MyBlobInfo';
    Blobs.Active := False;
    { Don't overwrite an existing table }
//    if Blobs.Exists then
//      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
//    else
    begin
      if (Blobs.Exists) then
      begin
        Blobs.Close;
        Blobs.DeleteTable;
      end;
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
```

```
          begin
            Name := 'Field2';
            DataType := ftBlob;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
{
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
}
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Blobs.Active:= True;
        for i := 1 to 20 do
          Blobs.AppendRecord([i*111, IntToStr(i*222)]);
      end;
    end;
    Form1.BlobDS.DataSet:= Blobs;
    Blobs.Active:= True;
  end;

  procedure TForm1.ListBox1MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
  begin
    Form1Done := True;
  end;

  initialization

  Form1Done := False;
  {
  The following code sets the DataSource for a DBGrid
  component in Form2 to the currently selected data source in
  a list provided in Form1. This allows users to specify the
  desired information in Form1, and then move to Form2 to view
  the information. Select an item in the listbox and then
  activate Form2 by clicking in it.
  }
  procedure TForm2.FormActivate(Sender: TObject);
  begin
    if (Form1Done) then
    begin
      DBGrid2.DataSource :=
        Form1.ListBox1.Items.Objects[Form1.ListBox1.ItemIndex] as TDataSource;
      DBNavigator2.DataSource :=
        Form1.ListBox1.Items.Objects[Form1.ListBox1.ItemIndex] as TDataSource;
    end;
  end;
```

See Also

FormStyle (see page 13912)

OnDeactivate (see page 13953)

OnActivate (see page 13726)

72.1.7.4.2 TForm.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore: TAlignInsertBeforeEvent;
```

C++

```
__property TAlignInsertBeforeEvent OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore (see page 6137)

Align (see page 5798)

TAlignInsertBeforeEvent (see page 6293)

72.1.7.4.3 TForm.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition: TAlignPositionEvent;
```

C++

```
__property TAlignPositionEvent OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition (see page 6138)

Align (see page 5798)

Anchors (see page 5802)

TAlignInfo (see page 6269)

TAlignPositionEvent (see page 6293)

72.1.7.4.4 TForm.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize: TCanResizeEvent;
```

C++

```
__property TCanResizeEvent OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize (see page 5853)

SetBounds (see page 5783)

OnResize (see page 5870)

72.1.7.4.5 TForm.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
```

```
  edit control placed somewhere on the client area of the
  sheet.  Unchecking the checkbox will cause the client area
  (not the tab area) of the Tab Control to become invisible.
  }
  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    PageControl1.ActivePage.Visible := CheckBox1.Checked;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    CheckBox1.Checked := PageControl1.ActivePage.Visible;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

72.1.7.4.6 TForm.OnClose Event

Occurs when the form closes.

Pascal

```
property OnClose: TCloseEvent;
```

C++

```
__property TCloseEvent OnClose;
```

Description

Use OnClose to perform special processing when the form closes. The OnClose event specifies which event handler to call when a form is about to close. The handler specified by OnClose might, for example, test to make sure all fields in a data-entry form have valid contents before allowing the form to close.

A form is closed by the Close method or when the user chooses Close from the form's system menu.

The TCloseEvent type points to a method that handles the closing of a form. The value of the Action parameter determines if the form actually closes. These are the possible values of Action:

Value	Meaning
caNone	The form is not allowed to close, so nothing happens.
caHide	The form is not closed, but just hidden. Your application can still access a hidden form.
caFree	The form is closed and all allocated memory for the form is freed.
caMinimize	The form is minimized, rather than closed. This is the default action for MDI child forms.

If a form is an MDI child form, and its BorderIcons property is biMinimize, then the default Action is caMinimize. If a MDI child form does not have these settings, the default Action is caNone, meaning that nothing happens when the user attempts to close the form.

If a form is an SDI child form, Action defaults to caHide.

To close the form and free it in an OnClose event, set Action to caFree.

Note: When the application shuts down, the main form receives an OnClose event, but any child forms do not receive the OnClose event.

C++ Examples:

```
/*
This example displays a message dialog box when the user
attempts to close the form. If the user clicks the Yes
button, the form closes; otherwise, the form only minimizes.
TCustomForm.Close does not allow an action of caMinimize
for a form closing itself.  Close Form2 from Form1 (from
a button?) and implement the OnClose in Form2.  Place
"Application.CreateForm(TForm2, Form2);" in the project dpr
file to open Form2 and set the Form2 Visible property to
True.
*/
void __fastcall TForm2::FormClose(TObject *Sender, TCloseAction &Action)
{
  if (MessageDlg("Close application ?", mtConfirmation, TMsgDlgButtons() << mbYes << mbNo,0)
== mrYes)
    Action = caFree;
  else
    Action = caMinimize;
}
```

See Also

OnCreate (see page 13951)

OnCloseQuery (see page 13950)

OnDestroy (see page 13953)

72.1.7.4.7 TForm.OnCloseQuery Event

Occurs when close is attempted.

Pascal

```
property OnCloseQuery: TCloseQueryEvent;
```

C++

```
__property TCloseQueryEvent OnCloseQuery;
```

Description

Use OnCloseQuery to specify the conditions under which the form can close. An OnCloseQuery event handler returns a Boolean CanClose value that determines whether a form is allowed to close. Its default value is true.

You can use an OnCloseQuery event handler to ask users if they are sure they really want the form closed immediately. For example, you can use the handler to display a message box that prompts the user to save a file before closing the form.

The TCloseQueryEvent type points to the method that determines whether a form can be closed. The value of the CanClose parameter determines if the form can close or not.

C++ Examples:

```
/*
When the user attempts to close the form in this example, a
message dialog appears that asks the user if it is OK to
close the form. If the user chooses the OK button, the form
closes. If the user chooses Cancel, the form doesn't close.
Populate the OnCloseQuery event handler of the form.
*/
void __fastcall TForm1::FormCloseQuery(TObject *Sender, bool &CanClose)
{
  if (MessageDlg("Close the form?", mtConfirmation, TMsgDlgButtons() << mbOK << mbCancel,0) ==
mrCancel)
    CanClose = false;
}
```

Delphi Examples:

```
{
When the user attempts to close the form in this example, a
message dialog appears that asks the user if it is OK to
close the form. If the user chooses the OK button, the form
closes. If the user chooses Cancel, the form doesn't close.
Populate the OnCloseQuery event handler of the form.
}
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  if MessageDlg('Close the form?', mtConfirmation,
    [mbOk, mbCancel], 0) = mrCancel then
     CanClose := False;
end;
```

See Also

OnClose (see page 13949)

72.1.7.4.8 TForm.OnConstrainedResize Event

Adjust resize constraints.

Pascal

```
property OnConstrainedResize: TConstrainedResizeEvent;
```

C++

```
__property TConstrainedResizeEvent OnConstrainedResize;
```

Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's

Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

Note: The OnConstrainedResize handler is called immediately after the OnCanResize handler.

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

Constraints (see page 5812)

OnResize (see page 5870)

CanAutoSize (see page 5741)

72.1.7.4.9 TForm.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

72.1.7.4.10 TForm.OnCreate Event

Occurs when the form is created.

Pascal

```
property OnCreate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCreate;
```

Description

Use OnCreate to perform special processing when the form is created and is invoked by TCustomForm's constructor. Either implement this event or override the constructor of the form; do not do both. Any objects created in the OnCreate event should be freed by the OnDestroy event.

When a form is being created and its Visible property is true, the following events occur in the order listed:

1. OnCreate2. OnShow3. OnActivate4. OnPaint

Note: Use of the OnCreate event is discouraged in C++ code because it can interact badly with the form's constructor (see OldCreateOrder). It is recommended that you override the form constructor instead.

C++ Examples:

```
/*
This example requires a new TPageControl, as-is, with no new pages
created at design time, and a TUpDown control, also from the Win95
page of the component palette.  The form OnCreate event handler
adds several new TabSheet controls to the Page Control.  The
UpDown1Click event handler fires when the user clicks the up or
down button on the TUpDown control.  The SelectNextPage method of
the Page Control goes forward the comparison (Button = btNext)
evaluates as true.  If Button is not btNext, the previous Tab Page
is selected.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
const
  TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *page = new TTabSheet(PageControl1);
    page->PageControl = PageControl1;
    page->Caption = AnsiString("Page") + IntToStr(i);
    page->Brush->Color = colorPalette[i];
  }
}

void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  PageControl1->SelectNextPage(Button == btNext);
}
```

Delphi Examples:

```
{
This example requires a new TPageControl, as-is, with no new pages
created at design time, and a TUpDown control, also from the Win95
```

```
  page of the component palette.  The form OnCreate event handler
  adds several new TabSheet controls to the Page Control.  The
  UpDown1Click event handler fires when the user clicks the up or
  down button on the TUpDown control.  The SelectNextPage method of
  the Page Control goes forward the comparison (Button = btNext)
  evaluates as true.  If Button is not btNext, the previous Tab Page
  is selected.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  const
    colorPalette: Array[0..11] of TColor =
      (clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
      clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
  begin
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet #' + IntToStr(i);
        Brush.Color := colorPalette[i];
      end;
  end;

  procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
  begin
    PageControl1.SelectNextPage(Button = btNext);
  end;
```

See Also

OnDestroy (see page 13953)

OnActivate (see page 13943)

OnPaint (see page 13975)

OnShow (see page 13977)

OldCreateOrder (see page 13926)

72.1.7.4.11 TForm.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

72.1.7.4.12 TForm.OnDeactivate Event

Occurs when the form loses focus.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Use OnDeactivate to perform special processing when the form transitions from being the active form to another form in the same application becoming the active form. If activation goes to another application, this event is not triggered. To determine if another application has become active, Use the TApplication object's OnDeactivate event.

See Also

OnActivate (see page 13943)

OnDeactivate (see page 13727)

72.1.7.4.13 TForm.OnDestroy Event

Occurs when the form is destroyed.

Pascal

```
property OnDestroy: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDestroy;
```

Description

Use OnDestroy to perform special processing when the form is destroyed. Either implement this event or override the destructor of the class, but not both. This event should destroy any objects created in the OnCreate event.

C++ Examples:

```
/*
The following code allocates a TBitmap object that is used
by the application when the main form is first created. The
OnDestroy method frees the bitmap as the application shuts
down.
*/
Graphics::TBitmap *LogoBmp;
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  LogoBmp = new Graphics::TBitmap();
  LogoBmp->LoadFromFile("../FACTORY.BMP");
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete LogoBmp;
}

void __fastcall TForm1::FormPaint(TObject *Sender)
{
  Canvas->Brush->Bitmap = LogoBmp;
  Canvas->FillRect(Rect(0,0,100,100));
}
```

Delphi Examples:

```
{
The following code explicitly allocates memory for a pointer
in the OnCreate event of Form1, then releases the memory in
the OnDestroy event. Assume that MyPtr is a Pointer type
field of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  New(MyPtr);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Dispose(MyPtr);
end;
```

See Also

OnCreate (see page 13951)

Free (see page 22300)

Destroy (see page 13862)

OldCreateOrder (see page 13926)

72.1.7.4.14 TForm.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop: TDockDropEvent;
```

C++

```
__property TDockDropEvent OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
```

```
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
```

```
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockOver (see page 6216)

72.1.7.4.15 TForm.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver: TDockOverEvent;
```

C++

```
__property TDockOverEvent OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
```

```
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
```

```
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnGetSiteInfo (see page 6223)

TDragDockObject (see page 6009)

72.1.7.4.16 TForm.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}
```

```
/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
```

```
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

72.1.7.4.17 TForm.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;
```

```
procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

72.1.7.4.18 TForm.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

72.1.7.4.19 TForm.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo: TGetSiteInfoEvent;
```

C++

```
__property TGetSiteInfoEvent OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
```

```
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
```

```
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

TDragDockObject (see page 6009)

72.1.7.4.20 TForm.OnHelp Event

Occurs when the form receives a request for help.

Pascal

```
property OnHelp: THelpEvent;
```

C++

```
__property THelpEvent OnHelp;
```

Description

Write an OnHelp event handler to perform special processing when the user requests help. The HelpContext and the HelpJump methods automatically trigger the OnHelp event.

Set the CallHelp parameter to true if the application should still invoke the help system after the event. Set CallHelp to false to prevent the default response by the help system. The form calls the help system only if OnHelp's CallHelp parameter returns true or if no OnHelp event handler is assigned.

To find the possible values of the Command and Data parameters, search for WinHelp in the Win32 Developer's Reference Help (Win32.HLP) file, which explains the WinHelp API (application programming interface). The possible values for the Data parameter depend upon the value of the Command parameter.

The event handler returns true if it succeeds, false if it fails.

See Also

HelpCommand (see page 13678)

HelpContext (see page 13679)

HelpFile (see page 13916)

HelpJump (see page 13679)

HelpFile (see page 13711)

OnHelp (see page 13730)

72.1.7.4.21 TForm.OnHide Event

Occurs when the form is hidden (that is, when its Visible property is set to false).

Pascal

```
property OnHide: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnHide;
```

Description

Use OnHide to perform special processing when the form is hidden (that is, when the form's Visible property is set to false).

C++ Examples:

```
/*
This example uses two forms, each with a label and a button.
When the user clicks a button, the other form appears and
the current form disappears. Also, a message appears in the
label of the form that is showing, indicating that the other
form is hidden. Set the OnHide event handle for each form to
FormHide.This is the implementation section of Unit1:
*/
#include "TFormOnHide.h"
#include "Unit2.h"  // We must include unit2's header file too!

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
  Hide();
}

void __fastcall TForm1::FormHide(TObject *Sender)
{
  Form2->Label1->Caption = "Form1 is hiding";
}
/*
This is the implementation of Unit2:
*/
#include "Unit2.h"
```

```
#include "TFormOnHide.h"  // We must include unit1's header file too!

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm2 *Form2;

__fastcall TForm2::TForm2(TComponent* Owner)
    : TForm(Owner)
{
}

void __fastcall TForm2::Button1Click(TObject *Sender)
{
  Form1->Show();
  Hide();
}

void __fastcall TForm2::FormHide(TObject *Sender)
{
  Form1->Label1->Caption = "Form2 is hiding";
}
```

Delphi Examples:

```
{
This example uses two forms, each with a label and a button.
When the user clicks a button, the other form appears and
the current form disappears. Also, a message appears in the
label of the form that is showing, indicating that the other
form is hidden. Set the OnHide event handle for each form to
FormHide.This is the implementation section of Unit1:
}

implementation

{$R *.dfm}

uses Unit2;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
  Hide;
end;

procedure TForm1.FormHide(Sender: TObject);
begin
  Form2.Label1.Caption := 'Form1 is hiding';
end;
{
This is the implementation section of Unit2:
}
implementation

{$R *.dfm}

uses TFormOnHide;

procedure TForm2.Button1Click(Sender: TObject);
begin
  Form1.Show;
  Hide;
end;

procedure TForm2.FormHide(Sender: TObject);
begin
  Form1.Label1.Caption := 'Form2 is hiding';
```

```
end;
```

See Also

OnShow (see page 13977)

72.1.7.4.22 TForm.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
```

```
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
```

```
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

72.1.7.4.23 TForm.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

72.1.7.4.24 TForm.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
```

```
  event to FormKeyUp.  Set the KeyPreview on the form to True.
  }
  var
    FormColor: TColor;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    KeyPreview := True;
    FormColor := Form1.Color;
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

72.1.7.4.25 TForm.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

72.1.7.4.26 TForm.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
```

```
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
```

```
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

72.1.7.4.27 TForm.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

72.1.7.4.28 TForm.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

72.1.7.4.29 TForm.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
```

```
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
```

```
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

72.1.7.4.30 TForm.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
```

```
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

72.1.7.4.31 TForm.OnMouseWheel Event

Occurs when the mouse wheel is rotated.

Pascal

```
property OnMouseWheel: TMouseWheelEvent;
```

C++

```
__property TMouseWheelEvent OnMouseWheel;
```

Description

Write code in the OnMouseWheel event handler to respond to mouse wheel messages. If there is no OnMouseWheel event handler, or if the mouse wheel message is not handled in the OnMouseWheel event handler, then an OnMouseWheelUp or OnMouseWheelDown event occurs, depending on the direction the mouse wheel was rotated.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheelUp (see page 5870)

72.1.7.4.32 TForm.OnMouseWheelDown Event

Occurs when the mouse wheel is rotated downward.

Pascal

```
property OnMouseWheelDown: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelDown;
```

Description

Write code in the OnMouseWheelDown event handler to perform actions when the mouse wheel is rotated downward. OnMouseWheelDown occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheel (see page 5869)

OnMouseWheelUp (see page 5870)

72.1.7.4.33 TForm.OnMouseWheelUp Event

Occurs when the mouse wheel is rotated upward.

Pascal

```
property OnMouseWheelUp: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelUp;
```

Description

Write code in the OnMouseWheelUp event handler to perform actions when the mouse wheel is rotated upward. OnMouseWheelUp occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheel (see page 5869)

72.1.7.4.34 TForm.OnPaint Event

Occurs when the form is redrawn.

Pascal

```
property OnPaint: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPaint;
```

Description

Use OnPaint to perform special processing when the form is redrawn. Any special painting on the form should be done in this event. OnPaint occurs before any controls on the form are painted.

To determine which portions of the form's canvas need to be repainted, use the ClipRect property of the canvas.

If you use the form's Canvas property outside the OnPaint event, it will be erased and drawn-over by the next OnPaint event.

C++ Examples:

```
/*
The following code loads a background bitmap onto the Canvas
of the main form in the OnPaint event handler.
*/
Graphics::TBitmap *TheGraphic;

void __fastcall TForm1::FormPaint(TObject *Sender)// OnPaint event handler
{
  Form1->Canvas->Draw(0, 0, TheGraphic); // Draw the graphic on the Canvas
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TheGraphic = new Graphics::TBitmap(); // Create the bitmap object
  TheGraphic->LoadFromFile("..\\bigf.bmp"); // Load the bitmap from a file
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  delete TheGraphic;
}
```

Delphi Examples:

```
{
The following code loads a background bitmap onto the Canvas
of the main form in the OnPaint event handler.
}
procedure TForm1.FormPaint(Sender: TObject); { OnPaint event handler}
begin
  Form1.Canvas.Draw(0, 0, TheGraphic); { Draw the graphic on the Canvas }
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  TheGraphic := TBitmap.Create; { Create the bitmap object }
  TheGraphic.LoadFromFile('bigf.bmp'); { Load the bitmap from a file}
```

```
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  (TheGraphic as TObject).Free;
end;
```

See Also

Canvas (see page 13900)

ClipRect (see page 14486)

72.1.7.4.35 TForm.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

72.1.7.4.36 TForm.OnShortCut Event

Occurs when the user presses a key (before the OnKeyDown event).

Pascal

```
property OnShortCut: TShortCutEvent;
```

C++

```
__property TShortCutEvent OnShortCut;
```

Description

Use OnShortCut to dispatch shortcut keystrokes before the form handles them. When the user presses a key, the form can dispatch it as a shortcut key instead of allowing the standard keystroke processing (OnKeyDown, OnKeyPress, and OnKeyUp). Built-in shortcut processing is provided for menu shortcuts and actions associated with the form. OnShortCut allows the form to implement additional shortcuts.

If the OnShortCut implements a response to the keystroke, set the Handled parameter of the event handler to true. This prevents the keystroke from being passed on to menus or actions associated with the form. It also prevents the standard keystroke processing in the same way that a menu or action shortcut does.

See Also

OnKeyDown (see page 6225)

IsShortCut (see page 13880)

IsShortCut (see page 322)

IsShortCut (see page 16078)

72.1.7.4.37 TForm.OnShow Event

Occurs when the form is shown (that is, when its Visible property is set to true).

Pascal

```
property OnShow: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnShow;
```

Description

Use OnShow to perform special processing when the form is shown (that is, when the form's Visible property is set to true).

C++ Examples:

```
/*
This project requires a Page Control.  You will need to
populate the Page Control by right clicking and selecting
New Page from the context menu. As the tabs are selected by
the user, the Page Control OnChange event fires, which
causes the form caption to display the captions of each of
the TTabSheets attached to the Page Control.
Note the use of the Form’s OnShow event handler to set the
caption when the form first appears.  This occurs before
any changes trigger the OnChange event of the Page Control.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  Caption = AnsiString("Now working on tab: " + PageControl1->ActivePage->Caption);
}

void __fastcall TForm1::FormShow(TObject *Sender)
{
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This project requires a Page Control.  You will need to
populate the Page Control by right clicking and selecting
New Page from the context menu. As the tabs are selected by
```

```
the user, the Page Control OnChange event fires, which
causes the form caption to display the captions of each of
the TTabSheets attached to the Page Control.
Note the use of the Form's OnShow event handler to set the
caption when the form first appears.  This occurs before
any changes trigger the OnChange event of the Page Control.
}
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  Caption := ' Now working on tab:  ' + PageControl1.ActivePage.Caption;
end;

procedure TForm1.FormShow(Sender: TObject);
begin
  PageControl1Change(Sender);
end;
```

See Also

OnHide (see page 13960)

72.1.7.4.38 TForm.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

72.1.7.4.39 TForm.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock: TUnDockEvent;
```

C++

```
__property TUnDockEvent OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
```

```
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
```

```
    var Accept: Boolean);
  var
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

72.1.8 TFrame Class

TFrame is a container for components; it can be nested within forms or other frames.

Class Hierarchy

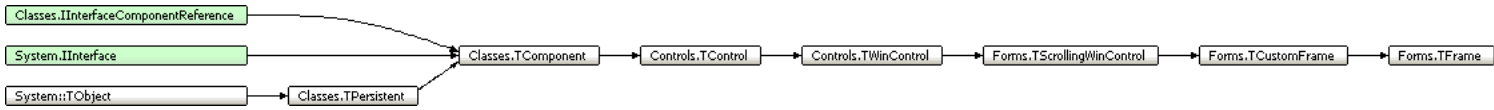

Pascal

```
TFrame = class(TCustomFrame);
```

C++

```
class TFrame : public TCustomFrame;
```

File

Forms

Description

When you create frames, they are implemented as descendants of TFrame.

A frame, like a form, is a container for other components. It uses the same ownership mechanism as forms for automatic instantiation and destruction of the components on it, and the same parent-child relationships for synchronization of component properties. But frames can be nested within forms or other frames, and they can be saved on the Component palette for easy reuse. After a frame is created and saved, it continues to function as a unit and to inherit changes from the components (including other frames) it contains. Moreover, an embedded frame continues to inherit changes made to the frame from which it is derived.

There is a nonstandard A-footnote (Frames_Object) in the main topic for TFrame. Please do not remove it, since it is necessary for making the Component palette Help work.

See Also

TForm (see page 13992)

72.1.8.1 TFrame Members

The following tables list the members exposed by TFrame.

Protected Events

	Name	Description
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 14324)	Adjusts the ClientRect property to exclude the scroll bars.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.

	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	AlignControls (see page 14325)	Aligns the controls in the scrolling windowed control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	AutoScrollEnabled (see page 14325)	Indicates whether automatic scrolling is enabled.
	AutoScrollInView (see page 14325)	Scrolls a control into the visible area of the scrolling windowed control.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	ChangeScale (see page 14326)	Resizes the scrolling windowed control by a specific ratio.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 13990)	Initializes frame creation parameters for efficient redrawing.
	CreateParams (see page 14326)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 14327)	Creates a Windows control corresponding to the TScrollingWinControl (see page 14312) object.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.

	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoFlipChildren (see page 14327)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetChildren (see page 13990)	Calls a specified method for each child of the frame.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.

	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
V	PaintWindow (see page 13991)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
V	Resizing (see page 14328)	This is Resizing, a member of class TScrollingWinControl.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.

	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParent (see page 13991)	Sets the Parent property.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 5731)	Creates an instance of TControl (see page 5724).
	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 13989)	Creates and initializes a TCustomFrame (see page 13978) object.

	Create (see page 14323)	Creates and initializes a new TScrollingWinControl (see page 14312) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 14323)	Removes the TScrollingWinControl (see page 14312) object from memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	DisableAutoRange (see page 14327)	Disables automatic scrolling.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EnableAutoRange (see page 14328)	Reenables automatic scrolling.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.

	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollInView (see page 14328)	Scrolls a control into the visible area of the scrolling windowed control.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.

V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.

R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 14150)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14151)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 14152)	Occurs when an attempt is made to resize the control.
	OnClick (see page 14152)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 14154)	Adjust resize constraints.
	OnContextPopup (see page 14155)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 14155)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 14156)	Occurs when another control is docked to the control.
	OnDockOver (see page 14158)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 14160)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 14162)	Occurs when the user drags an object over a control.
	OnEndDock (see page 14164)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 14165)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 14166)	Occurs when a control receives the input focus.
	OnExit (see page 14168)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 14170)	Returns the control's docking information.
	OnMouseActivate (see page 14172)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 14173)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 14175)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 14175)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 14175)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 14178)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 14180)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 14180)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 14181)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 14181)	Occurs immediately after the control is resized.
	OnStartDock (see page 14182)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 14182)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 14183)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 14126)	Determines how the control aligns within its container (parent control).

	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 14130)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 14130)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 14131)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BiDiMode (see page 14131)	Specifies the bi-directional mode for the control.
	Color (see page 14132)	Specifies the background color of the control.
	Constraints (see page 14132)	Specifies the size constraints for the control.
	Ctl3D (see page 14133)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DockSite (see page 14133)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 14134)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 14135)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 14135)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 14137)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 14137)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HorzScrollBar (see page 14329)	Represents the horizontal scroll bar for the scrolling windowed control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Padding (see page 14138)	Specifies the padding of a control.
	ParentBackground (see page 14139)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 14139)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 14139)	Determines where a control looks for its color information.
	ParentCtl3D (see page 14140)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 14140)	Determines where a control looks for its font information.
	ParentShowHint (see page 14141)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 14141)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 14146)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 14147)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 14148)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	VertScrollBar (see page 14330)	Represents the vertical scroll bar for the scrolling windowed control.
	Visible (see page 14148)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

72.1.8.2 TFrame Properties

The properties of the TFrame class are listed here.

Published Properties

	Name	Description
	Align (see page 14126)	Determines how the control aligns within its container (parent control).
	Anchors (see page 14130)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 14130)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 14131)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BiDiMode (see page 14131)	Specifies the bi-directional mode for the control.
	Color (see page 14132)	Specifies the background color of the control.
	Constraints (see page 14132)	Specifies the size constraints for the control.
	Ctl3D (see page 14133)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 14133)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 14134)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 14135)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 14135)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 14137)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 14137)	Controls the attributes of text written on or in the control.
	Padding (see page 14138)	Specifies the padding of a control.
	ParentBackground (see page 14139)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 14139)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 14139)	Determines where a control looks for its color information.
	ParentCtl3D (see page 14140)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 14140)	Determines where a control looks for its font information.
	ParentShowHint (see page 14141)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 14141)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 14146)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 14147)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 14148)	Determines if the user can tab to a control.
	Visible (see page 14148)	Determines whether the component appears onscreen.

72.1.8.2.1 TFrame.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on

it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
```

```
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form’s width }
  Split.MinSize := Form1.ClientWidth div 4;
```

```
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
```

```
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

72.1.8.2.2 TFrame.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

72.1.8.2.3 TFrame.AutoScroll Property

Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

If AutoScroll is true, the scroll bars appear automatically when necessary. For example, if the user resizes the control so that some of its controls are partially obscured, scroll bars appear.

If AutoScroll is false, scroll bars don't appear automatically. In this case, use the HorzScrollBar and VertScrollBar properties to make scroll bars appear.

See Also

AutoScrollInView (see page 14325)

HorzScrollBar (see page 14329)

VertScrollBar (see page 14330)

72.1.8.2.4 TFrame.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints (see page 5812)

CanAutoSize (see page 5741)

72.1.8.2.5 TFrame.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

72.1.8.2.6 TFrame.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

72.1.8.2.7 TFrame.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

72.1.8.2.8 TFrame.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

72.1.8.2.9 TFrame.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite: Boolean;
```

C++

```
__property Boolean DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

72.1.8.2.10 TFrame.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

72.1.8.2.11 TFrame.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

72.1.8.2.12 TFrame.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
```

```
  OnDragOver event handler must be implemented to accept the
  drop.
  */
  void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
        int X, int Y)
  {
    if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
      return;
    THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
    TNodeAttachMode AttachMode;
    TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
    if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
      AttachMode = naAddChild;
    else if (HT.Contains(htNowhere))
      AttachMode = naAdd;
    else if (HT.Contains(htOnIndent))
      AttachMode = naInsert;
    else
      return;
    TreeView1->Selected->MoveTo(Item, AttachMode);
  }

  void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
        int X, int Y, TDragState State, bool &Accept)
  {
    Accept = (Source == (TObject *)TreeView1);
  }
```

Delphi Examples:

```
  {
  The following code uses GetNodeAt to add a dragged node as a
  child of the node under the mouse when it is dropped.  This
  example requires a populated TreeView.  Also, the TTreeView
  DragMode property set to dmAutomatic, and the TTreeView
  OnDragOver event handler must be implemented to accept the
  drop.
  }
  procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
  var
    AnItem: TTreeNode;
    AttachMode: TNodeAttachMode;
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

72.1.8.2.13 TFrame.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

72.1.8.2.14 TFrame.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
```

```
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

72.1.8.2.15 TFrame.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding: TPadding;
```

C++

```
__property TPadding Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding (see page 6103)

72.1.8.2.16 TFrame.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground: Boolean;
```

C++

```
__property Boolean ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

72.1.8.2.17 TFrame.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

72.1.8.2.18 TFrame.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

72.1.8.2.19 TFrame.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

72.1.8.2.20 TFrame.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

72.1.8.2.21 TFrame.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

72.1.8.2.22 TFrame.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
```

```
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
```

```
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
```

```
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
```

```
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
```

```
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

72.1.8.2.23 TFrame.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
```

```
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

72.1.8.2.24 TFrame.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

72.1.8.2.25 TFrame.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

72.1.8.2.26 TFrame.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
```

```
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

72.1.8.3 TFrame Events

The events of the TFrame class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 14150)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14151)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 14152)	Occurs when an attempt is made to resize the control.
	OnClick (see page 14152)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 14154)	Adjust resize constraints.
	OnContextPopup (see page 14155)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 14155)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 14156)	Occurs when another control is docked to the control.
	OnDockOver (see page 14158)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 14160)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 14162)	Occurs when the user drags an object over a control.
	OnEndDock (see page 14164)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 14165)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 14166)	Occurs when a control receives the input focus.
	OnExit (see page 14168)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 14170)	Returns the control's docking information.
	OnMouseActivate (see page 14172)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 14173)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 14175)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 14175)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 14175)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 14178)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 14180)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 14180)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 14181)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 14181)	Occurs immediately after the control is resized.
	OnStartDock (see page 14182)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 14182)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 14183)	Occurs when the application tries to undock a control that is docked to the windowed control.

72.1.8.3.1 TFrame.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore: TAlignInsertBeforeEvent;
```

C++

```
__property TAlignInsertBeforeEvent OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore (see page 6137)

Align (see page 5798)

TAlignInsertBeforeEvent (see page 6293)

72.1.8.3.2 TFrame.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition: TAlignPositionEvent;
```

C++

```
__property TAlignPositionEvent OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition (see page 6138)

Align (see page 5798)

Anchors (see page 5802)

TAlignInfo (see page 6269)

TAlignPositionEvent (see page 6293)

72.1.8.3.3 TFrame.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize: TCanResizeEvent;
```

C++

```
__property TCanResizeEvent OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize (see page 5853)

SetBounds (see page 5783)

OnResize (see page 5870)

72.1.8.3.4 TFrame.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B.

However, focus does not move to the control in these instances.

- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
```

```
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

72.1.8.3.5 TFrame.OnConstrainedResize Event

Adjust resize constraints.

Pascal

```
property OnConstrainedResize: TConstrainedResizeEvent;
```

C++

```
__property TConstrainedResizeEvent OnConstrainedResize;
```

Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

Note: The OnConstrainedResize handler is called immediately after the OnCanResize handler.

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

Constraints (see page 5812)

OnResize (see page 5870)

CanAutoSize (see page 5741)

72.1.8.3.6 TFrame.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

72.1.8.3.7 TFrame.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

72.1.8.3.8 TFrame.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop: TDockDropEvent;
```

C++

```
__property TDockDropEvent OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
```

```
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock = DockClient->ClassNameIs("TDockableForm");
    if (!CanDock)
    {
       ShowMessage("GetSiteInfo CanDock went bad!");
    }
  }
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockOver (see page 6216)

72.1.8.3.9 TFrame.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver: TDockOverEvent;
```

C++

```
__property TDockOverEvent OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
```

```
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnGetSiteInfo (see page 6223)

TDragDockObject (see page 6009)

72.1.8.3.10 TFrame.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
```

```
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

72.1.8.3.11 TFrame.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
```

```
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
```

```
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

72.1.8.3.12 TFrame.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

72.1.8.3.13 TFrame.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

72.1.8.3.14 TFrame.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
```

```
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
```

```
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

72.1.8.3.15 TFrame.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK

button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

72.1.8.3.16 TFrame.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo: TGetSiteInfoEvent;
```

C++

```
__property TGetSiteInfoEvent OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
```

```
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

TDragDockObject (see page 6009)

72.1.8.3.17 TFrame.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

72.1.8.3.18 TFrame.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
```

```
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
```

```
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

72.1.8.3.19 TFrame.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

72.1.8.3.20 TFrame.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

72.1.8.3.21 TFrame.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
```

```
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
```

```
      end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

72.1.8.3.22 TFrame.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}
```

```
// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
```

```
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

72.1.8.3.23 TFrame.OnMouseWheel Event

Occurs when the mouse wheel is rotated.

Pascal

```
property OnMouseWheel: TMouseWheelEvent;
```

C++

```
__property TMouseWheelEvent OnMouseWheel;
```

Description

Write code in the OnMouseWheel event handler to respond to mouse wheel messages. If there is no OnMouseWheel event handler, or if the mouse wheel message is not handled in the OnMouseWheel event handler, then an OnMouseWheelUp or OnMouseWheelDown event occurs, depending on the direction the mouse wheel was rotated.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheelUp (see page 5870)

72.1.8.3.24 TFrame.OnMouseWheelDown Event

Occurs when the mouse wheel is rotated downward.

Pascal

```
property OnMouseWheelDown: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelDown;
```

Description

Write code in the OnMouseWheelDown event handler to perform actions when the mouse wheel is rotated downward.

OnMouseWheelDown occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheel (see page 5869)

OnMouseWheelUp (see page 5870)

72.1.8.3.25 TFrame.OnMouseWheelUp Event

Occurs when the mouse wheel is rotated upward.

Pascal

```
property OnMouseWheelUp: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelUp;
```

Description

Write code in the OnMouseWheelUp event handler to perform actions when the mouse wheel is rotated upward. OnMouseWheelUp occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheel (see page 5869)

72.1.8.3.26 TFrame.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

72.1.8.3.27 TFrame.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

72.1.8.3.28 TFrame.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

72.1.8.3.29 TFrame.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock: TUnDockEvent;
```

C++

```
__property TUnDockEvent OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
```

```
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
```

```
    // Only 2 dock clients means the host must be destroyed and
    // the remaining window undocked to its old position and size.
    // Recall that OnUnDock gets called before the undocking
    // actually occurs.
    if Client is TDockableForm then
      TDockableForm(Client).DockSite := True;
    if (DockClientCount = 2) and (NewTarget <> Self) then
      PostMessage(Self.Handle, WM_CLOSE, 0, 0);
    UpdateCaption(Client);
  end;
  {
  The following example is taken from the docking demo. It
  shows how set permissions to dock dockable objects onto a
  docking site.
  }

  procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer);
  begin
    //OnDockDrop gets called AFTER the client has actually docked,
    //so we check for DockClientCount = 1 before making the dock panel visible.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, True, nil);
    (Sender as TPanel).DockManager.ResetBounds(True);
    //Make DockManager repaints it's clients.
  end;

  procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer; State: TDragState;
    var Accept: Boolean);
  var
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

72.1.9 TGlassFrame Class

TGlassFrame controls Windows Vista Aero glass effects.

Class Hierarchy

Pascal

```
TGlassFrame = class(TPersistent);
```

C++

```
class TGlassFrame : public TPersistent;
```

File

Forms

Description

TGlassFrame uses the DwmExtendFrameIntoClientArea API to extend glass effects into the client area. Use the Top (see page 14190), Left (see page 14190), Bottom (see page 14190), and Right (see page 14190) properties to specify the amount to extend the glass effect into the client area, or use SheetOfGlass (see page 14190) to render the entire client area as a glass surface. This is only available on Windows Vista with Aero.

See Also

TForm (see page 13992)

TCustomForm (see page 13849)

72.1.9.1 TGlassFrame Members

The following tables list the members exposed by TGlassFrame.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Change (see page 14189)	This is Change, a member of class TGlassFrame.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Events

	Name	Description
	OnChange (see page 14191)	Occurs when the dimensions for glass effects change, or when they are enabled/disabled.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 14188)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 14187)	This is Create, a member of class TGlassFrame.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FrameExtended (see page 14189)	Returns True if glass effects are extended into the client area.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IntersectsControl (see page 14189)	Returns True if a control intersects the glass area.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Published Properties

	Name	Description
	Bottom (see page 14190)	Specifies the amount (in pixels) glass effects are extended into the client area.
	Enabled (see page 14190)	This property controls whether glass effects are extended into the client area or not.
	Left (see page 14190)	Specifies the amount (in pixels) glass effects are extended into the client area.
	Right (see page 14190)	Specifies the amount (in pixels) glass effects are extended into the client area.
	SheetOfGlass (see page 14190)	Extends glass effects into the entire client area.
	Top (see page 14190)	Specifies the amount (in pixels) glass effects are extended into the client area.

72.1.9.2 TGlassFrame.Create Constructor

Pascal

```
constructor Create(Client: TCustomForm);
```

C++

```
__fastcall TGlassFrame(TCustomForm Client);
```

Description

This is Create, a member of class TGlassFrame.

72.1.9.3 TGlassFrame Methods

The methods of the TGlassFrame class are listed here.

Protected Methods

	Name	Description
	Change (see page 14189)	This is Change, a member of class TGlassFrame.

Public Methods

	Name	Description
	Assign (see page 14188)	Copies the contents of another, similar object.
	FrameExtended (see page 14189)	Returns True if glass effects are extended into the client area.
	IntersectsControl (see page 14189)	Returns True if a control intersects the glass area.

72.1.9.3.1 TGlassFrame.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo (see page 2931)

EConvertError (see page 22724)

72.1.9.3.2 TGlassFrame.Change Method

Pascal

```
procedure Change; virtual;
```

C++

```
virtual __fastcall Change();
```

Description

This is Change, a member of class TGlassFrame.

72.1.9.3.3 TGlassFrame.FrameExtended Method

Returns True if glass effects are extended into the client area.

Pascal

```
function FrameExtended: Boolean;
```

C++

```
__fastcall Boolean FrameExtended();
```

Description

Use FrameExtended to determine if glass effects are supported and enabled. The return value is True if Desktop Window Manager (DWM) composition is enabled, Enabled (see page 14190) is True and either SheetOfGlass (see page 14190) is True or any of the properties Top (see page 14190), Left (see page 14190), Bottom (see page 14190) or Right (see page 14190) have a value other than zero.

72.1.9.3.4 TGlassFrame.IntersectsControl Method

Returns True if a control intersects the glass area.

Pascal

```
function IntersectsControl(Control: TControl): Boolean;
```

C++

```
__fastcall Boolean IntersectsControl(TControl * Control);
```

72.1.9.4 TGlassFrame Properties

The properties of the TGlassFrame class are listed here.

Published Properties

	Name	Description
	Bottom (see page 14190)	Specifies the amount (in pixels) glass effects are extended into the client area.
	Enabled (see page 14190)	This property controls whether glass effects are extended into the client area or not.
	Left (see page 14190)	Specifies the amount (in pixels) glass effects are extended into the client area.
	Right (see page 14190)	Specifies the amount (in pixels) glass effects are extended into the client area.
	SheetOfGlass (see page 14190)	Extends glass effects into the entire client area.
	Top (see page 14190)	Specifies the amount (in pixels) glass effects are extended into the client area.

72.1.9.4.1 TGlassFrame.Bottom Property

Specifies the amount (in pixels) glass effects are extended into the client area.

Pascal

```
property Bottom: Integer;
```

C++

```
__property int Bottom;
```

72.1.9.4.2 TGlassFrame.Enabled Property

This property controls whether glass effects are extended into the client area or not.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

72.1.9.4.3 TGlassFrame.Left Property

Specifies the amount (in pixels) glass effects are extended into the client area.

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

72.1.9.4.4 TGlassFrame.Right Property

Specifies the amount (in pixels) glass effects are extended into the client area.

Pascal

```
property Right: Integer;
```

C++

```
__property int Right;
```

72.1.9.4.5 TGlassFrame.SheetOfGlass Property

Extends glass effects into the entire client area.

Pascal

```
property SheetOfGlass: Boolean;
```

C++

```
__property Boolean SheetOfGlass;
```

72.1.9.4.6 TGlassFrame.Top Property

Specifies the amount (in pixels) glass effects are extended into the client area.

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

72.1.9.5 TGlassFrame Events

The events of the TGlassFrame class are listed here.

Public Events

	Name	Description
	OnChange (see page 14191)	Occurs when the dimensions for glass effects change, or when they are enabled/disabled.

72.1.9.5.1 TGlassFrame.OnChange Event

Occurs when the dimensions for glass effects change, or when they are enabled/disabled.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

72.1.10 TMonitor Class

TMonitor represents a monitor on which an application runs.

Class Hierarchy

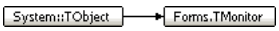

Pascal

```
TMonitor = class(TObject);
```

C++

```
class TMonitor : public TObject;
```

File

Forms

Description

TMonitor represents a monitor that displays the user interface of an application. It introduces properties to describe the dimensions of a single monitor. In addition, the Primary (see page 14195) property distinguishes the primary property from other monitors on the system.

When writing multi-monitor applications, use TMonitor to help lay out forms and dialogs so that they are not split over several monitors.

Applications do not create instances of TMonitor. Instead, monitor components for all available monitors are listed by the Monitors property of the global Screen (see page 14385) variable. The global Screen (see page 14385) variable also provides global information about the monitors on which the application runs, such as screen resolution, available fonts, and so

on.

See Also

TApplication (see page 13660)

TScreen (see page 14197)

TCustomForm (see page 13849)

Screen (see page 14385)

TCustomForm (see page 13849)

72.1.10.1 TMonitor Members

The following tables list the members exposed by TMonitor.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	BoundsRect (see page 14193)	Indicates the dimensions of the monitor in pixels.
R	Handle (see page 14194)	Indicates the Windows handle for the monitor.
R	Height (see page 14194)	Indicates the vertical size of the monitor in pixels.

R	Left (see page 14194)	Indicates the logical position of the left edge of the monitor.
R	MonitorNum (see page 14195)	Specifies the index of the monitor in the global screen object's Monitors list.
R	Primary (see page 14195)	Indicates whether this is the primary monitor.
R	Top (see page 14196)	Indicates the logical position of the top edge of the monitor.
R	Width (see page 14196)	Indicates the horizontal size of the monitor in pixels.
R	WorkareaRect (see page 14197)	Gives the application useable area of the monitor.

72.1.10.2 TMonitor Properties

The properties of the TMonitor class are listed here.

Public Properties

	Name	Description
R	BoundsRect (see page 14193)	Indicates the dimensions of the monitor in pixels.
R	Handle (see page 14194)	Indicates the Windows handle for the monitor.
R	Height (see page 14194)	Indicates the vertical size of the monitor in pixels.
R	Left (see page 14194)	Indicates the logical position of the left edge of the monitor.
R	MonitorNum (see page 14195)	Specifies the index of the monitor in the global screen object's Monitors list.
R	Primary (see page 14195)	Indicates whether this is the primary monitor.
R	Top (see page 14196)	Indicates the logical position of the top edge of the monitor.
R	Width (see page 14196)	Indicates the horizontal size of the monitor in pixels.
R	WorkareaRect (see page 14197)	Gives the application useable area of the monitor.

72.1.10.2.1 TMonitor.BoundsRect Property

Indicates the dimensions of the monitor in pixels.

Pascal

```
property BoundsRect: TRect;
```

C++

```
__property TRect BoundsRect;
```

Description

Read BoundsRect to learn the dimensions of the monitor. BoundsRect gives the dimensions of the monitor in pixels, where (0,0) represents the top-left corner of the primary monitor.

The top of BoundsRect is given by the Top (see page 14196) property, the left edge by the Left (see page 14194) property, and the height and width by the Height (see page 14194) and Width (see page 14196) properties respectively.

Note: The BoundsRect property does not take into account any task bars or tool bars docked on the monitor. To determine the area on the monitor that is free of such docked windows, use the WorkAreaRect property.

See Also

Width (see page 14196)

Top (see page 14196)

Left (see page 14194)

Height (see page 14194)

WorkAreaRect (see page 14197)

DesktopRect (see page 14218)

Primary (see page 14195)

72.1.10.2.2 TMonitor.Handle Property

Indicates the Windows handle for the monitor.

Pascal

```
property Handle: HMONITOR;
```

C++

```
__property HMONITOR Handle;
```

Description

Read Monitor to obtain a Windows handle for the monitor represented by this TMonitor (see page 14191) component. Use this handle for Windows API calls such as GetMonitorInfo.

See Also

MonitorNum (see page 14195)

72.1.10.2.3 TMonitor.Height Property

Indicates the vertical size of the monitor in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Read Height to learn the size of the monitor in pixels. Use this value along with the Top (see page 14196) property to calculate how to position objects vertically in multi-monitor applications.

Note: The Height property does not take into account any task bars or tool bars docked on the monitor. To determine the area on the monitor that is free of such docked windows, use the WorkAreaRect property.

See Also

Height (see page 14225)

DesktopHeight (see page 14217)

Width (see page 14196)

Top (see page 14196)

BoundsRect (see page 14193)

WorkAreaRect (see page 14197)

72.1.10.2.4 TMonitor.Left Property

Indicates the logical position of the left edge of the monitor.

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

In multi-monitor applications, the available screen space can span multiple monitors from left to right. Read Left to determine where, in the entire available screen space, this monitor appears. The value of Left is in pixels relative to the left edge of the primary monitor.

See Also

Width (see page 14196)

Top (see page 14196)

BoundsRect (see page 14193)

WorkAreaRect (see page 14197)

Primary (see page 14195)

DesktopLeft (see page 14218)

72.1.10.2.5 TMonitor.MonitorNum Property

Specifies the index of the monitor in the global screen object's Monitors list.

Pascal

```
property MonitorNum: Integer;
```

C++

```
__property int MonitorNum;
```

Description

The global screen variable maintains a list of all monitors in a multi-monitor application. MonitorNum is the index of this monitor in the list object that is the screen's Monitors property.

Use MonitorNum to identify a specific monitor when working with multiple monitors.

See Also

Monitors (see page 14228)

Screen (see page 14385)

72.1.10.2.6 TMonitor.Primary Property

Indicates whether this is the primary monitor.

Pascal

```
property Primary: Boolean;
```

C++

```
__property Boolean Primary;
```

Description

Read Primary to determine whether a monitor is the primary monitor in a multi-monitor application. When Primary is true, the monitor is the primary monitor. The coordinates of other monitors are given in terms of the primary monitor.

See Also

Monitors (see page 14228)

BoundsRect (see page 14193)

DesktopRect (see page 14218)

72.1.10.2.7 TMonitor.Top Property

Indicates the logical position of the top edge of the monitor.

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

In multi-monitor applications, the available screen space can span multiple monitors from top to bottom. Read Top to determine where, in the entire available screen space, this monitor appears. The value of Top is in pixels relative to the top edge of the primary monitor.

See Also

Height (see page 14194)

Left (see page 14194)

BoundsRect (see page 14193)

WorkAreaRect (see page 14197)

Primary (see page 14195)

DesktopTop (see page 14219)

72.1.10.2.8 TMonitor.Width Property

Indicates the horizontal size of the monitor in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Read Width to learn the size of the monitor in pixels. Use this value along with the Left (see page 14194) property to calculate vertical positions for objects in multi-monitor applications.

Note: The Width property does not take into account any task bars or tool bars docked on the monitor. To determine the area on the monitor that is free of such docked windows, use the WorkAreaRect property.

See Also

Height (see page 14194)

Left (see page 14194)

Width (see page 14229)

DesktopWidth (see page 14219)

BoundsRect (see page 14193)

WorkAreaRect (see page 14197)

72.1.10.2.9 TMonitor.WorkareaRect Property

Gives the application useable area of the monitor.

Pascal

```
property WorkareaRect: TRect;
```

C++

```
__property TRect WorkareaRect;
```

Description

WorkareaRect returns a TRect value furnished with the coordinates and dimensions of the work area of the Monitor. On Windows, for example, the application tabs at the screen mean that the Workarea is smaller than the monitor size.

Note: The TRect Right and Bottom values are one pixel beyond Workarea boundary. They are given these values to allow for easy calculation of Workarea width and height as (Right-Left (see page 14194)) and (Bottom-Top (see page 14196)) respectively.

See Also

TRect (see page 23302)

72.1.11 TScreen Class

TScreen represents the state of the screen in which an application runs.

Class Hierarchy

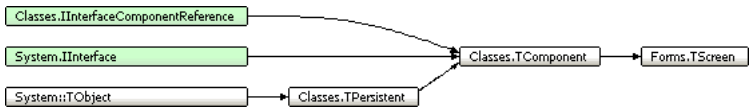

Pascal

```
TScreen = class(TComponent);
```

C++

```
class TScreen : public TComponent;
```

File

Forms

Description

TScreen introduces properties that keep track of

What forms and data modules have been instantiated by the application.

The active form, and the active control within that form.

The size and resolution of the screen.

The cursors and fonts available for the application to use.

The monitors on which the application's forms can appear, including their dimensions.

There is a global variable, Screen (see page 14385), of type TScreen, which is already instantiated for use by GUI applications. Use Screen (see page 14385) to obtain information about the current state of the screen in an application.

See Also

TApplication (see page 13660)

TDataModule (see page 2824)

TForm (see page 13992)

TFont (see page 14497)

Screen (see page 14385)

Creating Applications

MDI Applications

SDI Applications

User Interface Models

GUI Applications

Handling the Screen

Controlling Application Behavior

72.1.11.1 TScreen Members

The following tables list the members exposed by TScreen.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnActiveControlChange (see page 14234)	Occurs immediately after input focus changes to a new windowed control.
	OnActiveFormChange (see page 14236)	Occurs immediately after a new form becomes active in a multi-form application.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 14202)	Creates an instance of TScreen (see page 14197).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 14202)	Destroys an instance of TScreen (see page 14197).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 14203)	Prevents forms from being aligned in the screen.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.

	EnableAlign (see page 14203)	Allows forms to be aligned in the screen.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MonitorFromPoint (see page 14203)	Returns the monitor where a specified point is located.
	MonitorFromRect (see page 14204)	Returns the monitor where a specified rectangle is located.
	MonitorFromWindow (see page 14204)	Returns the monitor where a specified window is located.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 14205)	Realigns the screen's forms according to their Align properties.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	ResetFonts (see page 14205)	Reinitializes the fonts listed in the Fonts (see page 14220) property.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActiveControl (see page 14206)	Indicates which control currently has input focus on the screen.
R	ActiveCustomForm (see page 14207)	Indicates the descendant of TCustomForm (see page 13849) that currently has focus.
R	ActiveForm (see page 14207)	Indicates which form currently has focus.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.

R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Cursor (see page 14209)	Controls (see page 5711) the mouse cursor image on a global level.
R	CursorCount (see page 14212)	This is CursorCount, a member of class TScreen.
	Cursors (see page 14212)	Lists the cursors available to the application.
R	CustomFormCount (see page 14214)	Indicates the number of forms or property pages displayed on the screen.
R	CustomForms (see page 14215)	Lists all the forms and property pages that are currently displayed in the application.
R	DataModuleCount (see page 14216)	Indicates the number of data modules that are currently instantiated in the application.
R	DataModules (see page 14216)	Lists all the data modules that are currently instantiated in the application.
R	DefaultIme (see page 14216)	Indicates which input method editor (IME) is active when the TScreen (see page 14197) object is created.
R	DefaultKbLayout (see page 14217)	Indicates the Windows handle to the keyboard layout that was active when the application started running.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DesktopHeight (see page 14217)	Specifies the height of the entire virtual desktop.
R	DesktopLeft (see page 14218)	Specifies the x-coordinate of the desktop's left edge relative to the upper-left corner of the primary monitor.
R	DesktopRect (see page 14218)	Specifies the boundaries of the virtual desktop relative to the upper-left corner of the primary monitor.
R	DesktopTop (see page 14219)	Specifies the y-coordinate of the entire desktop's top edge relative to the upper-left corner of the primary monitor.
R	DesktopWidth (see page 14219)	Specifies the width of the entire virtual desktop.
	FocusedForm (see page 14220)	Provides a reference to the form that has focus.
R	Fonts (see page 14220)	Lists the face names for all fonts supported by the screen.
R	FormCount (see page 14221)	Indicates the number of forms displayed on the screen.
R	Forms (see page 14223)	Lists all the forms currently displayed in the application.
R	Height (see page 14225)	Indicates the vertical size of the screen in pixels.
	HintFont (see page 14226)	Specifies the font used to display help hints.
	IconFont (see page 14226)	Specifies the font used to display the labels on the icons in file selection dialogs.
R	Imes (see page 14227)	Lists all input method editors (IMEs) installed on the system.
	MenuFont (see page 14227)	Specifies the font used for menu commands.
R	MonitorCount (see page 14228)	Specifies the number of monitors used to comprise the desktop.
R	Monitors (see page 14228)	Provides access to an individual monitor used to comprise the desktop.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	PixelsPerInch (see page 14228)	Indicates the number of screen pixels that make up a logical inch in the vertical direction.
R	PrimaryMonitor (see page 14229)	Provides a reference to the primary monitor.
R	SaveFocusedList (see page 14229)	This is SaveFocusedList, a member of class TScreen.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Width (see page 14229)	Indicates the horizontal size of the screen in pixels.
R	WorkAreaHeight (see page 14231)	Specifies the height of the work area on the primary monitor.
R	WorkAreaLeft (see page 14231)	Specifies the left edge of the work area on the primary monitor.
R	WorkAreaRect (see page 14232)	Specifies the boundaries of the usable work area on the primary monitor.
R	WorkAreaTop (see page 14232)	Specifies the top edge of the work area on the primary monitor.
R	WorkAreaWidth (see page 14233)	Specifies the width of the work area on the primary monitor.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

72.1.11.2 TScreen.Create Constructor

Creates an instance of TScreen (see page 14197).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TScreen(TComponent * AOwner);
```

Description

Do not create separate instances of TScreen (see page 14197). There is a global variable, Screen (see page 14385), of type TScreen (see page 14197), which is already instantiated for use by GUI applications. The properties of Screen (see page 14385) are global to the entire application.

AOwner specifies the Owner of the screen, which is responsible for freeing it. For the global TScreen (see page 14197) object, this is the application.

See Also

Screen (see page 14385)

Owner (see page 2807)

TApplication (see page 13660)

72.1.11.3 TScreen.Destroy Destructor

Destroys an instance of TScreen (see page 14197).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScreen();
```

Description

Do not destroy the global object Screen (see page 14385) that is automatically instantiated in GUI applications.

See Also

Screen (see page 14385)

Free (see page 22300)

72.1.11.4 TScreen Methods

The methods of the TScreen class are listed here.

Public Methods

	Name	Description
	DisableAlign (see page 14203)	Prevents forms from being aligned in the screen.
	EnableAlign (see page 14203)	Allows forms to be aligned in the screen.
	MonitorFromPoint (see page 14203)	Returns the monitor where a specified point is located.

	MonitorFromRect (see page 14204)	Returns the monitor where a specified rectangle is located.
	MonitorFromWindow (see page 14204)	Returns the monitor where a specified window is located.
	Realign (see page 14205)	Realigns the screen's forms according to their Align properties.
	ResetFonts (see page 14205)	Reinitializes the fonts listed in the Fonts (see page 14220) property.

72.1.11.4.1 TScreen.DisableAlign Method

Prevents forms from being aligned in the screen.

Pascal

```
procedure DisableAlign;
```

C++

```
__fastcall DisableAlign();
```

Description

Call DisableAlign to prevent forms from being aligned in the screen. After calling DisableAlign, the Align properties of the screen's forms are ignored until the call is matched by a call to EnableAlign (see page 14203).

See Also

EnableAlign (see page 14203)

Realign (see page 14205)

Constraints (see page 5812)

72.1.11.4.2 TScreen.EnableAlign Method

Allows forms to be aligned in the screen.

Pascal

```
procedure EnableAlign;
```

C++

```
__fastcall EnableAlign();
```

Description

Call EnableAlign to allow forms to be aligned in the screen. After calling DisableAlign (see page 14203), the Align properties of the screen's forms are ignored until that call is matched by a call to EnableAlign.

See Also

DisableAlign (see page 14203)

Realign (see page 14205)

Constraints (see page 5812)

72.1.11.4.3 TScreen.MonitorFromPoint Method

Returns the monitor where a specified point is located.

Pascal

```
function MonitorFromPoint(const Point: TPoint; MonitorDefault: TMonitorDefaultTo = mdNearest):
TMonitor;
```

C++

```
__fastcall TMonitor MonitorFromPoint(const TPoint Point, TMonitorDefaultTo MonitorDefault =
mdNearest);
```

Description

Call MonitorFromPoint determine the monitor where a specified point is located.

Point is the coordinates of the point to locate. It is in pixels relative to the upper-left corner of the primary monitor.

MonitorDefault indicates how to select a monitor when the point does not fall clearly on a single monitor.

See Also

Monitors (see page 14228)

DesktopRect (see page 14218)

BoundsRect (see page 14193)

72.1.11.4.4 TScreen.MonitorFromRect Method

Returns the monitor where a specified rectangle is located.

Pascal

```
function MonitorFromRect(const Rect: TRect; MonitorDefault: TMonitorDefaultTo = mdNearest):
TMonitor;
```

C++

```
__fastcall TMonitor MonitorFromRect(const TRect Rect, TMonitorDefaultTo MonitorDefault =
mdNearest);
```

Description

Call MonitorFromRect determine the monitor where a specified boundary rectangle falls.

Rect is the coordinates of the rectangle to locate. It is in pixels relative to the upper-left corner of the primary monitor.

MonitorDefault indicates how to select a monitor when the rectangle does not fall entirely on a single monitor.

See Also

Monitors (see page 14228)

DesktopRect (see page 14218)

BoundsRect (see page 14193)

72.1.11.4.5 TScreen.MonitorFromWindow Method

Returns the monitor where a specified window is located.

Pascal

```
function MonitorFromWindow(const Handle: THandle; MonitorDefault: TMonitorDefaultTo =
mdNearest): TMonitor;
```

C++

```
__fastcall TMonitor MonitorFromWindow(const THandle Handle, TMonitorDefaultTo MonitorDefault =
mdNearest);
```

Description

Call MonitorFromWindow determine the monitor that displays a specified window.

Handle is the window handle for the window to be located.

MonitorDefault indicates how to select a monitor when the window does not fall entirely on a single monitor.

See Also

Monitors (see page 14228)

DesktopRect (see page 14218)

BoundsRect (see page 14193)

72.1.11.4.6 TScreen.Realign Method

Realigns the screen's forms according to their Align properties.

Pascal

```
procedure Realign;
```

C++

```
__fastcall Realign();
```

Description

Call Realign to realign the screen's forms according to their Align properties.

See Also

EnableAlign (see page 14203)

DisableAlign (see page 14203)

Constraints (see page 5812)

72.1.11.4.7 TScreen.ResetFonts Method

Reinitializes the fonts listed in the Fonts (see page 14220) property.

Pascal

```
procedure ResetFonts;
```

C++

```
__fastcall ResetFonts();
```

Description

Call ResetFonts to ensure that the fonts listed in the Fonts (see page 14220) property are up to date. ResetFonts reinitializes the Fonts (see page 14220) property so that its value is reread from the operating system the next time it is needed.

TScreen (see page 14197) calls ResetFonts internally in response to messages from Windows that the available fonts have changed.

See Also

Fonts (see page 14220)

72.1.11.5 TScreen Properties

The properties of the TScreen class are listed here.

Public Properties

	Name	Description
R	ActiveControl (see page 14206)	Indicates which control currently has input focus on the screen.
R	ActiveCustomForm (see page 14207)	Indicates the descendant of TCustomForm (see page 13849) that currently has focus.
R	ActiveForm (see page 14207)	Indicates which form currently has focus.
	Cursor (see page 14209)	Controls (see page 5711) the mouse cursor image on a global level.
R	CursorCount (see page 14212)	This is CursorCount, a member of class TScreen.
	Cursors (see page 14212)	Lists the cursors available to the application.
R	CustomFormCount (see page 14214)	Indicates the number of forms or property pages displayed on the screen.
R	CustomForms (see page 14215)	Lists all the forms and property pages that are currently displayed in the application.
R	DataModuleCount (see page 14216)	Indicates the number of data modules that are currently instantiated in the application.
R	DataModules (see page 14216)	Lists all the data modules that are currently instantiated in the application.
R	DefaultIme (see page 14216)	Indicates which input method editor (IME) is active when the TScreen (see page 14197) object is created.
R	DefaultKbLayout (see page 14217)	Indicates the Windows handle to the keyboard layout that was active when the application started running.
R	DesktopHeight (see page 14217)	Specifies the height of the entire virtual desktop.
R	DesktopLeft (see page 14218)	Specifies the x-coordinate of the desktop's left edge relative to the upper-left corner of the primary monitor.
R	DesktopRect (see page 14218)	Specifies the boundaries of the virtual desktop relative to the upper-left corner of the primary monitor.
R	DesktopTop (see page 14219)	Specifies the y-coordinate of the entire desktop's top edge relative to the upper-left corner of the primary monitor.
R	DesktopWidth (see page 14219)	Specifies the width of the entire virtual desktop.
	FocusedForm (see page 14220)	Provides a reference to the form that has focus.
R	Fonts (see page 14220)	Lists the face names for all fonts supported by the screen.
R	FormCount (see page 14221)	Indicates the number of forms displayed on the screen.
R	Forms (see page 14223)	Lists all the forms currently displayed in the application.
R	Height (see page 14225)	Indicates the vertical size of the screen in pixels.
	HintFont (see page 14226)	Specifies the font used to display help hints.
	IconFont (see page 14226)	Specifies the font used to display the labels on the icons in file selection dialogs.
R	Imes (see page 14227)	Lists all input method editors (IMEs) installed on the system.
	MenuFont (see page 14227)	Specifies the font used for menu commands.
R	MonitorCount (see page 14228)	Specifies the number of monitors used to comprise the desktop.
R	Monitors (see page 14228)	Provides access to an individual monitor used to comprise the desktop.
R	PixelsPerInch (see page 14228)	Indicates the number of screen pixels that make up a logical inch in the vertical direction.
R	PrimaryMonitor (see page 14229)	Provides a reference to the primary monitor.
R	SaveFocusedList (see page 14229)	This is SaveFocusedList, a member of class TScreen.
R	Width (see page 14229)	Indicates the horizontal size of the screen in pixels.
R	WorkAreaHeight (see page 14231)	Specifies the height of the work area on the primary monitor.
R	WorkAreaLeft (see page 14231)	Specifies the left edge of the work area on the primary monitor.
R	WorkAreaRect (see page 14232)	Specifies the boundaries of the usable work area on the primary monitor.
R	WorkAreaTop (see page 14232)	Specifies the top edge of the work area on the primary monitor.
R	WorkAreaWidth (see page 14233)	Specifies the width of the work area on the primary monitor.

72.1.11.5.1 TScreen.ActiveControl Property

Indicates which control currently has input focus on the screen.

Pascal

```
property ActiveControl: TWinControl;
```

C++

```
__property TWinControl * ActiveControl;
```

Description

Read ActiveControl to learn which windowed control object in the active form currently receives the input from the keyboard.

ActiveControl is a read-only property. To change the ActiveControl, use the SetFocusedControl method of the form that contains the control that should receive focus.

After focus shifts from one control to another, the screen receives an OnActiveControlChange (see page 14234) event.

See Also

OnActiveControlChange (see page 14234)

ActiveControl (see page 13896)

SetFocusedControl (see page 13887)

72.1.11.5.2 TScreen.ActiveCustomForm Property

Indicates the descendant of TCustomForm (see page 13849) that currently has focus.

Pascal

```
property ActiveCustomForm: TCustomForm;
```

C++

```
__property TCustomForm ActiveCustomForm;
```

Description

Read ActiveCustomForm to learn which form or property page has input focus. If the active custom form is a TForm (see page 13992) object, ActiveCustomForm is the same as ActiveForm (see page 14207).

See Also

ActiveForm (see page 14207)

OnActiveFormChange (see page 14236)

TForm (see page 13992)

TPropertyPage (see page 1603)

72.1.11.5.3 TScreen.ActiveForm Property

Indicates which form currently has focus.

Pascal

```
property ActiveForm: TForm;
```

C++

```
__property TForm * ActiveForm;
```

Description

Read ActiveForm to learn which form in the application has input focus. If the application is not currently active, ActiveForm is the form that will have focus when the application becomes active again.

ActiveForm is a read-only property. To change the ActiveForm, use the SetFocus method of the form that should receive focus. ActiveForm also changes if the SetFocusedControl method of an inactive form is called to set focus to a control on that inactive

form.

After focus shifts from one form to another, the screen receives an OnActiveFormChange (see page 14236) event.

Note: If the application window with focus is not a TForm (see page 13992) descendant (for example, if it is a property page), it will not appear as ActiveForm. To handle such cases, use the ActiveCustomForm (see page 14207) property instead.

C++ Examples:

```
/*
This example uses a two forms with a button on the first
form. When the user clicks the button, the second form
appears. As the user switches between forms, the form that
is active is colored aqua.
Note:   You must include the unit header for form 2 to the
unit file for form 1, and add the public ColorForm method
to the definition of the TForm1 class.
*/
#include "Unit2.h"

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::ColorForm(TObject *Sender)
{
  Color = clWhite;
  Form2->Color = clWhite;
  Screen->ActiveForm->Color = clAqua;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Screen->OnActiveFormChange = ColorForm;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  // This stops ColorForm from being called during termination.
  Screen->OnActiveFormChange = NULL;
}
```

Delphi Examples:

```
{
This example uses a two forms with a button on the first
form. When the user clicks the button, the second form
appears. As the user switches between forms, the form that
is active is colored aqua.
Note:   You must include the unit header for form 2 to the
unit file for form 1, and add the public ColorForm method
to the header for form 1's unit.
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure ColorForm(Sender: TObject);
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

uses Unit2;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.ColorForm(Sender: TObject);
begin
  Color := clWhite;
  Form2.Color := clWhite;
  Screen.ActiveForm.Color := clAqua;
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  Screen.OnActiveFormChange := ColorForm;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  // This stops ColorForm from being called during termination.
  Screen.OnActiveFormChange := nil;
end;
```

See Also

ActiveCustomForm (see page 14207)

OnActiveFormChange (see page 14236)

ActiveMDIChild (see page 13897)

SetFocus (see page 13886)

SetFocusedControl (see page 13887)

How Actions Find Their Targets

72.1.11.5.4 TScreen.Cursor Property

Controls (see page 5711) the mouse cursor image on a global level.

Pascal

```
property Cursor: TCursor;
```

C++

```
__property TCursor Cursor;
```

Description

Set Cursor to override the cursor image associated with individual control objects. When Cursor is crDefault, the individual objects determine the cursor image. Assigning any other value sets the mouse cursor image for all windows belonging to the application. The global mouse cursor image remains in effect until the screen's Cursor property is changed back to crDefault.

Cursor can be set to any of the cursor values available in the Cursors (see page 14212) property. This can be one of the built-in cursor values, or a custom cursor that has been added to the Cursors (see page 14212) property.

Note: Changes to the value of the Cursor property may not show up immediately. Call the ProcessMessages method of the global Application (see page 14383) object to allow the application to respond to a notification that the value has changed.

C++ Examples:

```
/*
This example shows how to add custom cursors to an
application.  The following code makes this cursor available
to the application via the constant crMyCursor, and sets it
as the global cursor to the application.
*/
Graphics::TBitmap *bmpMask, *bmpColor;
TIconInfo *iconInfo;

const crMyCursor = 5;

  void __fastcall TForm1::Button2Click(TObject *Sender)
{
  bmpMask = new Graphics::TBitmap();
  bmpColor = new Graphics::TBitmap();
  iconInfo = new TIconInfo();

  bmpMask->LoadFromFile("../SquareMask.bmp");
  bmpColor->LoadFromFile("../Square.bmp");

  iconInfo->fIcon = false;
  iconInfo->xHotspot = 15;
  iconInfo->yHotspot = 15;
  iconInfo->hbmMask = bmpMask->Handle;
  iconInfo->hbmColor = bmpColor->Handle;

  Screen->Cursors[crMyCursor] = CreateIconIndirect(iconInfo);

  Screen->Cursor = crMyCursor;

  delete bmpMask;
  delete bmpColor;
  delete iconInfo;
}
/*
Assignments to the Screen object's cursor property are
typically guarded by a try...finally statement to ensure
that normal cursor behavior is restored. Note that the
original cursor is saved and restored.  This allows cursor
changes to be nested without accidentally restoring the
cursor to a hard-coded value prematurely.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
TCursor Save_Cursor = Screen->Cursor;
Screen->Cursor = crHourGlass;    // Show hourglass cursor
try
{
  // Do some lengthy operation
  int I = 0;
  int J = 0;
  int X, Y;
  const magicnumber = 150000000;
  while (I < magicnumber)
  {
    Randomize;
    while (J < magicnumber)
    {
      Y = Random(J);
```

```
      J++;
    }
    X = Random(I);
    I++;
  }
}
__finally
{
  Screen->Cursor = Save_Cursor; // always restore the cursor
}
}
```

Delphi Examples:

```
{
This example shows how to add custom cursors to an
application.  The following code makes this cursor available
to the application via the constant crMyCursor, and sets it
as the global cursor to the application.
}
var
  bmpMask : TBitmap;
  bmpColor : TBitmap;
  iconInfo : TIconInfo;
const
  crMyCursor = 5;

procedure TForm1.Button2Click(Sender: TObject);
begin
  bmpMask := TBitmap.Create;
  bmpColor := TBitmap.Create;

  bmpMask.LoadFromFile('SquareMask.bmp');
  bmpColor.LoadFromFile('Square.bmp');

  with iconInfo do
  begin
    fIcon := false;
    xHotspot := 15;
    yHotspot := 15;
    hbmMask := bmpMask.Handle;
    hbmColor := bmpColor.Handle;
  end;

  Screen.Cursors[crMyCursor] := CreateIconIndirect(iconInfo);

  Screen.Cursor := crMyCursor;

  bmpMask.Free;
  bmpColor.Free;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  DestroyIcon(Screen.Cursors[crMyCursor]);
end;
{
Assignments to the Screen object's cursor property are
typically guarded by a try...finally statement to ensure
that normal cursor behavior is restored. Note that the
original cursor is saved and restored.  This allows cursor
changes to be nested without accidentally restoring the
cursor to a hard-coded value prematurely.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Save_Cursor : TCursor;
```

```
    I, J: Integer;
    X, Y: Word;
  const
    magicnumber = 3000000;
  begin
    Save_Cursor := Screen.Cursor;
    Screen.Cursor := crHourGlass;    { Show hourglass cursor }
    try
      { Do some lengthy operation }
    I := 0;
    J := 0;
    while I < magicnumber do
    begin
      Randomize;
      while J < magicnumber do
      begin
        Y := Random(J);
        Inc(J);
      end;
      X := Random(I);
      Inc(I);
    end;
    finally
      Screen.Cursor := Save_Cursor;  { Always restore to normal }
    end;
  end;
```

See Also

Cursors (see page 14212)

Cursor (see page 5815)

ProcessMessages (see page 13691)

72.1.11.5.5 TScreen.CursorCount Property

Pascal

```
property CursorCount: Integer;
```

C++

```
__property int CursorCount;
```

Description

This is CursorCount, a member of class TScreen.

72.1.11.5.6 TScreen.Cursors Property

Lists the cursors available to the application.

Pascal

```
property Cursors [Index: Integer]: HCURSOR;
```

C++

```
__property HCURSOR Cursors[int Index];
```

Description

Use Cursor (see page 14209) to access a particular cursor for use by the application or by a control within the application. TScreen (see page 14197) includes several built-in cursors that are indexed by symbolic cursor constants. The image associated with the built-in cursors constants can be changed by setting the Cursors property.

Custom cursors can be added to the Cursors property for use by the application or any of its controls. To add a custom cursor to an application:

1Create (see page 14202) the cursor resource using a resource editor.

2Declare a cursor constant with a value that does not conflict with an existing cursor constant.

3Use the Windows API function LoadCursor to obtain a handle to the new cursor.

4Set the Cursors property, indexed by the newly declared cursor constant, to the handle obtained from LoadCursor.

Note: Don't call the Windows API function DestroyCursor when finished with a custom cursor; it is handled automatically.

C++ Examples:

```
/*
This example shows how to add custom cursors to an
application.  The following code makes this cursor available
to the application via the constant crMyCursor, and sets it
as the global cursor to the application.
*/
Graphics::TBitmap *bmpMask, *bmpColor;
TIconInfo *iconInfo;

const crMyCursor = 5;

  void __fastcall TForm1::Button2Click(TObject *Sender)
{
  bmpMask = new Graphics::TBitmap();
  bmpColor = new Graphics::TBitmap();
  iconInfo = new TIconInfo();

  bmpMask->LoadFromFile("../SquareMask.bmp");
  bmpColor->LoadFromFile("../Square.bmp");

  iconInfo->fIcon = false;
  iconInfo->xHotspot = 15;
  iconInfo->yHotspot = 15;
  iconInfo->hbmMask = bmpMask->Handle;
  iconInfo->hbmColor = bmpColor->Handle;

  Screen->Cursors[crMyCursor] = CreateIconIndirect(iconInfo);

  Screen->Cursor = crMyCursor;

  delete bmpMask;
  delete bmpColor;
  delete iconInfo;
}
```

Delphi Examples:

```
{
This example shows how to add custom cursors to an
application.  The following code makes this cursor available
to the application via the constant crMyCursor, and sets it
as the global cursor to the application.
}
var
  bmpMask : TBitmap;
  bmpColor : TBitmap;
  iconInfo : TIconInfo;
const
  crMyCursor = 5;
```

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  bmpMask := TBitmap.Create;
  bmpColor := TBitmap.Create;

  bmpMask.LoadFromFile('SquareMask.bmp');
  bmpColor.LoadFromFile('Square.bmp');

  with iconInfo do
  begin
    fIcon := false;
    xHotspot := 15;
    yHotspot := 15;
    hbmMask := bmpMask.Handle;
    hbmColor := bmpColor.Handle;
  end;

  Screen.Cursors[crMyCursor] := CreateIconIndirect(iconInfo);

  Screen.Cursor := crMyCursor;

  bmpMask.Free;
  bmpColor.Free;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  DestroyIcon(Screen.Cursors[crMyCursor]);
end;
```

See Also

Cursor (see page 14209)

Cursor (see page 5815)

72.1.11.5.7 TScreen.CustomFormCount Property

Indicates the number of forms or property pages displayed on the screen.

Pascal

```
property CustomFormCount: Integer;
```

C++

```
__property int CustomFormCount;
```

Description

Read CustomFormCount to learn the number of forms or property pages currently displayed on the screen. These forms can be accessed by the CustomForms (see page 14215) property. CustomFormCount can be used with CustomForms (see page 14215) to iterate over all the forms and property pages in an application.

C++ Examples:

```
/*
The following code adds the name of all forms and dialogs on
the screen to ListBox1 when Button1 is clicked.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < Screen->CustomFormCount; i++)
    ListBox1->Items->Add(Screen->CustomForms[i]->Name);
}
```

Delphi Examples:

```
{
The following code adds the name of all forms and dialogs on
the screen to ListBox1 when Button1 is clicked.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to Screen.CustomFormCount-1 do
    ListBox1.Items.Add(Screen.CustomForms[I].Name);
end;
```

See Also

CustomForms (see page 14215)

FormCount (see page 14221)

72.1.11.5.8 TScreen.CustomForms Property

Lists all the forms and property pages that are currently displayed in the application.

Pascal

```
property CustomForms [Index: Integer]: TCustomForm;
```

C++

```
__property TCustomForm CustomForms[int Index];
```

Description

Use CustomForms to access a form or property page by index. The value of Index is a number between zero (the first custom form) and CustomFormCount (see page 14214) - 1. CustomForms can be used with CustomFormCount (see page 14214) when an application needs to iterate over all its forms and property pages.

Unlike the Forms (see page 14223) property, CustomForms lists TCustomForm (see page 13849) descendants that do not derive from TForm (see page 13992) (such as property pages).

Warning: The order in which CustomForms lists its forms is affected by the Z order of the forms. Do not change the Z order of forms when using CustomForms to iterate over the forms in an application.

C++ Examples:

```
/*
The following code adds the name of all forms and dialogs on
the screen to ListBox1 when Button1 is clicked.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < Screen->CustomFormCount; i++)
    ListBox1->Items->Add(Screen->CustomForms[i]->Name);
}
```

Delphi Examples:

```
{
The following code adds the name of all forms and dialogs on
the screen to ListBox1 when Button1 is clicked.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to Screen.CustomFormCount-1 do
```

```
    ListBox1.Items.Add(Screen.CustomForms[I].Name);
end;
```

See Also

CustomFormCount (see page 14214)

Forms (see page 14223)

72.1.11.5.9 TScreen.DataModuleCount Property

Indicates the number of data modules that are currently instantiated in the application.

Pascal

```
property DataModuleCount: Integer;
```

C++

```
__property int DataModuleCount;
```

Description

Read DataModuleCount to learn the number of data modules in the DataModules (see page 14216) property. DataModuleCount can be used with DataModules (see page 14216) when an application needs to iterate over all its data modules.

See Also

DataModules (see page 14216)

TDataModule (see page 2824)

72.1.11.5.10 TScreen.DataModules Property

Lists all the data modules that are currently instantiated in the application.

Pascal

```
property DataModules [Index: Integer]: TDataModule;
```

C++

```
__property TDataModule DataModules[int Index];
```

Description

Use DataModules to access an instantiated data module by index. The value of Index is a number between zero (the first data module) and DataModuleCount (see page 14216) - 1. DataModules can be used with DataModuleCount (see page 14216) when an application needs to iterate over all its data modules.

See Also

DataModuleCount (see page 14216)

72.1.11.5.11 TScreen.DefaultIme Property

Indicates which input method editor (IME) is active when the TScreen (see page 14197) object is created.

Pascal

```
property DefaultIme: string;
```

C++

```
__property AnsiString DefaultIme;
```

Description

In applications that support Asian language characters, read DefaultIme to determine the IME that is active when the application starts.

An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application. There can be multiple IMEs installed on a single system. They vary in user interface, conversion dictionaries, and input method.

Individual controls can override the default IME. When a control specifies a different IME should be used to enter information into the control, the IME that processes keyboard input changes when the control has focus. When the control loses focus, DefaultIme is restored as the system IME.

See Also

DefaultKbLayout (see page 14217)

Imes (see page 14227)

ImeName (see page 6208)

ResetIme (see page 6174)

72.1.11.5.12 TScreen.DefaultKbLayout Property

Indicates the Windows handle to the keyboard layout that was active when the application started running.

Pascal

```
property DefaultKbLayout: HKL;
```

C++

```
__property HKL DefaultKbLayout;
```

Description

Use DefaultKbLayout for making Windows API calls that require a handle to a keyboard layout. For example, if an application changes the keyboard layout to match the settings for another country, the value of DefaultKbLayout can be used as a parameter to the Windows ActivateKeyboardLayout function in order to restore the original layout.

For asian keyboards, the keyboard layout identifies the input method editor (IME) as well as the default language. Use DefaultKbLayout in Windows API calls that work with the IME. DefaultKbLayout is the keyboard layout handle associated with the IME named by the DefaultIme (see page 14216) property.

See Also

DefaultIme (see page 14216)

Imes (see page 14227)

72.1.11.5.13 TScreen.DesktopHeight Property

Specifies the height of the entire virtual desktop.

Pascal

```
property DesktopHeight: Integer;
```

C++

```
__property int DesktopHeight;
```

Description

Use DesktopHeight to determine the height of the entire desktop. The desktop is defined as the entire virtual desktop, which includes all monitors in the system.

On a single-monitor system, DesktopHeight corresponds to Height (see page 14225).

See Also

DesktopLeft (see page 14218)

DesktopRect (see page 14218)

DesktopTop (see page 14219)

DesktopWidth (see page 14219)

WorkAreaHeight (see page 14231)

Height (see page 14225)

72.1.11.5.14 TScreen.DesktopLeft Property

Specifies the x-coordinate of the desktop's left edge relative to the upper-left corner of the primary monitor.

Pascal

```
property DesktopLeft: Integer;
```

C++

```
__property int DesktopLeft;
```

Description

Use DesktopLeft to determine the x-coordinate of the entire desktop's left edge relative to the upper-left corner of the primary monitor. The primary monitor is the monitor with its Primary property set to true. The desktop is defined as the entire virtual desktop, which includes all monitors in the system.

On a single-monitor system, DesktopLeft is 0.

See Also

DesktopHeight (see page 14217)

DesktopRect (see page 14218)

DesktopTop (see page 14219)

DesktopWidth (see page 14219)

WorkAreaLeft (see page 14231)

Primary (see page 14195)

Monitors (see page 14228)

72.1.11.5.15 TScreen.DesktopRect Property

Specifies the boundaries of the virtual desktop relative to the upper-left corner of the primary monitor.

Pascal

```
property DesktopRect: TRect;
```

C++

```
__property TRect DesktopRect;
```

Description

Use DesktopRect to determine the coordinates of the entire virtual desktop, which includes all monitors in the system. DesktopRect is expressed in coordinates where (0,0) is the upper-left corner of the primary monitor. The primary monitor is the monitor with its Primary property set to true.

See Also

DesktopHeight (see page 14217)

DesktopLeft (see page 14218)

DesktopTop (see page 14219)

DesktopWidth (see page 14219)

WorkAreaRect (see page 14232)

Primary (see page 14195)

Monitors (see page 14228)

72.1.11.5.16 TScreen.DesktopTop Property

Specifies the y-coordinate of the entire desktop's top edge relative to the upper-left corner of the primary monitor.

Pascal

```
property DesktopTop: Integer;
```

C++

```
__property int DesktopTop;
```

Description

Use DesktopTop to determine the y-coordinate of the desktop's top edge relative to the upper-left corner of the primary monitor. The primary monitor is the monitor with its Primary property set to true. The desktop is defined as the entire virtual desktop, which includes all monitors in the system.

On a single-monitor system, DesktopTop is 0.

See Also

DesktopHeight (see page 14217)

DesktopRect (see page 14218)

DesktopLeft (see page 14218)

DesktopWidth (see page 14219)

WorkAreaTop (see page 14232)

Primary (see page 14195)

Monitors (see page 14228)

72.1.11.5.17 TScreen.DesktopWidth Property

Specifies the width of the entire virtual desktop.

Pascal

```
property DesktopWidth: Integer;
```

C++

```
__property int DesktopWidth;
```

Description

Use DesktopWidth to determine the width of the desktop. The desktop is defined as the entire virtual desktop, which includes all monitors in the system.

On a single-monitor system, DesktopWidth corresponds to Width (see page 14229).

See Also

DesktopHeight (see page 14217)

DesktopRect (see page 14218)

DesktopLeft (see page 14218)

DesktopTop (see page 14219)

WorkAreaWidth (see page 14233)

Width (see page 14229)

72.1.11.5.18 TScreen.FocusedForm Property

Provides a reference to the form that has focus.

Pascal

```
property FocusedForm: TCustomForm;
```

C++

```
__property TCustomForm FocusedForm;
```

Description

FocusedForm returns a reference to the form that currently has focus.

See Also

TCustomForm (see page 13849)

72.1.11.5.19 TScreen.Fonts Property

Lists the face names for all fonts supported by the screen.

Pascal

```
property Fonts: TStrings;
```

C++

```
__property TStrings Fonts;
```

Description

Read Fonts to learn what screen fonts are currently installed. Applications can use Fonts to ensure that they do not request a font that is not installed on the user's system. When an application uses a TFont object to request a font that is not installed, the system substitutes another font, which may not be a font appropriate to the application's needs.

Note: Fonts is a list of screen fonts. It does not give any indication of the printer fonts available to the application.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

TFont (see page 14497)

ResetFonts (see page 14205)

72.1.11.5.20 TScreen.FormCount Property

Indicates the number of forms displayed on the screen.

Pascal

```
property FormCount: Integer;
```

C++

```
__property int FormCount;
```

Description

Read FormCount to learn the number of forms currently displayed on the screen. These forms can be accessed by the Forms (see page 14223) property. FormCount can be used with Forms (see page 14223) to iterate over all the forms in an application.

C++ Examples:

```
/*
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *NewItem;
  // first create the separator
  NewItem = new TMenuItem(this); // If you change the AOwner (this) to MainMenu1->Items[1] and
then you add the new menu you will get "Menu inserted twice"?  Doesn't happen in Delphi.
  NewItem->Name = "Separator";
  NewItem->Caption = "-";
  // add the new item to the Windows menu
  MainMenu1->Items[1].Add(NewItem);
  // now create and add a menu item for each form
  for  (int I = 0; I < Screen->FormCount; I++)
  {
    NewItem = new TMenuItem(this);
    NewItem->Caption = Screen->Forms[I]->Name + "Item";
    NewItem->Name = Screen->Forms[I]->Name;
    MainMenu1->Items[1].Add(NewItem);
  }
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  // Count down, not up!
  for  (int I = MainMenu1->Items[1].Count - 1; I >= 0; I--)
  {
    delete MainMenu1->Items[1].Items[I];
  }
}
```

Delphi Examples:

```
{
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
}
procedure TForm1.Button1Click(Sender: TObject);
var
  NewItem: TMenuItem;
  I : integer;
begin
  { first create the separator }
  NewItem := TMenuItem.Create(MainMenu1.Items[1]);
  NewItem.Name:= 'Separator';
  NewItem.Caption := '-';
  { add the new item to the Windows menu }
  MainMenu1.Items[1].Add(NewItem);
  { now create and add a menu item for each form }
  for  I := 0 to Screen.FormCount-1 do
  begin
```

```
      NewItem := TMenuItem.Create(MainMenu1.Items[1]);
      NewItem.Caption := Screen.Forms[I].Name + 'Item';
      NewItem.Name := Screen.Forms[I].Name;
      MainMenu1.Items[1].Add(NewItem);
     end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I : Integer;
  begin
    // Count down, not up!
    for  I := MainMenu1.Items[1].Count - 1 downto 0 do
    begin
      MainMenu1.Items[1].Items[I].Free;
    end;
  end;
```

See Also

Forms (see page 14223)

TForm (see page 13992)

72.1.11.5.21 TScreen.Forms Property

Lists all the forms currently displayed in the application.

Pascal

```
property Forms [Index: Integer]: TForm;
```

C++

```
__property TForm * Forms[int Index];
```

Description

Use Forms to access a form by index. The value of Index is a number between zero (the first form) and FormCount (see page 14221) - 1. Forms can be used with FormCount (see page 14221) when an application needs to iterate over all its forms, including all dialogs.

Forms only lists the TForm (see page 13992) descendants in the application. This does not include, for example, property pages. To get a list that includes TCustomForm (see page 13849) descendants that do not descend from TForm (see page 13992), use CustomForms (see page 14215) instead.

Warning: The order in which Forms lists its forms is affected by the Z order of the forms. Do not change the Z order of forms when using Forms to iterate over the forms in an application.

C++ Examples:

```
/*
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *NewItem;
  // first create the separator
  NewItem = new TMenuItem(this); // If you change the AOwner (this) to MainMenu1->Items[1] and
then you add the new menu you will get "Menu inserted twice"?  Doesn't happen in Delphi.
```

```
    NewItem->Name = "Separator";
    NewItem->Caption = "-";
    // add the new item to the Windows menu
    MainMenu1->Items[1].Add(NewItem);
    // now create and add a menu item for each form
    for  (int I = 0; I < Screen->FormCount; I++)
    {
      NewItem = new TMenuItem(this);
      NewItem->Caption = Screen->Forms[I]->Name + "Item";
      NewItem->Name = Screen->Forms[I]->Name;
      MainMenu1->Items[1].Add(NewItem);
    }
  }

  void __fastcall TForm1::FormDestroy(TObject *Sender)
  {
    // Count down, not up!
    for  (int I = MainMenu1->Items[1].Count - 1; I >= 0; I--)
    {
      delete MainMenu1->Items[1].Items[I];
    }
  }
```

Delphi Examples:

```
{
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
}
procedure TForm1.Button1Click(Sender: TObject);
var
  NewItem: TMenuItem;
  I : integer;
begin
  { first create the separator }
  NewItem := TMenuItem.Create(MainMenu1.Items[1]);
  NewItem.Name:= 'Separator';
  NewItem.Caption := '-';
  { add the new item to the Windows menu }
  MainMenu1.Items[1].Add(NewItem);
  { now create and add a menu item for each form }
  for  I := 0 to Screen.FormCount-1 do
  begin
    NewItem := TMenuItem.Create(MainMenu1.Items[1]);
    NewItem.Caption := Screen.Forms[I].Name + 'Item';
    NewItem.Name := Screen.Forms[I].Name;
    MainMenu1.Items[1].Add(NewItem);
   end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I : Integer;
begin
  // Count down, not up!
  for  I := MainMenu1.Items[1].Count - 1 downto 0 do
  begin
    MainMenu1.Items[1].Items[I].Free;
  end;
end;
```

See Also

FormCount (see page 14221)

CustomForms (see page 14215)

72.1.11.5.22 TScreen.Height Property

Indicates the vertical size of the screen in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Read Height to learn the size of the user's screen in pixels. The size or position of objects placed on the screen, can then be set to ensure that the objects fit on screen and don't look crowded.

C++ Examples:

```
/*
The following code determines the height of all forms on the
screen and resizes any that are taller than the screen height.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < Screen->FormCount; i++)
  {
    if (Screen->Forms[i]->Top < 0)
    {
      Screen->Forms[i]->Height += Screen->Forms[i]->Top;
      Screen->Forms[i]->Top = 0;
    }
    if (Screen->Forms[i]->Left < 0)
    {
      Screen->Forms[i]->Width += Screen->Forms[i]->Left;
      Screen->Forms[i]->Left = 0;
    }
    if (Screen->Forms[i]->Height > Screen->Height)
      Screen->Forms[i]->Height = Screen->Height;
    if (Screen->Forms[i]->Width > Screen->Width)
      Screen->Forms[i]->Width = Screen->Width;
  }
}
```

Delphi Examples:

```
{
The following code determines the height of all forms on the
screen and resizes any that are taller than the screen height.
}
procedure TForm1.Button1Click(Sender: TObject);
var I: Integer;
begin
with Screen do
  for I := 0 to FormCount-1 do
    begin
    if Forms[I].Top < 0 then
    begin
      Forms[I].Height := Forms[I].Height + Forms[I].Top;
      Forms[I].Top := 0;
    end;
    if Forms[I].Left < 0 then
    begin
      Forms[I].Width := Forms[I].Width + Forms[I].Left;
      Forms[I].Left := 0;
```

```
    end;
    if Forms[I].Width > Width then
      Forms[I].Width := Width;
    if Forms[I].Height > Height then
      Forms[I].Height := Height;
    if Forms[I].Width > Width then
      Forms[I].Width := Width;
    end;
end;
```

See Also

PixelsPerInch (see page 14228)

Height (see page 5825)

Width (see page 14229)

WorkAreaHeight (see page 14231)

DesktopHeight (see page 14217)

Considerations When Dynamically Resizing Forms and Controls

72.1.11.5.23 TScreen.HintFont Property

Specifies the font used to display help hints.

Pascal

```
property HintFont: TFont;
```

C++

```
__property TFont HintFont;
```

Description

HintFont stores the Windows status font setting. This font is used by default for the help hints that appear when the user pauses the mouse over a control or menu item.

See Also

Fonts (see page 14220)

IconFont (see page 14226)

MenuFont (see page 14227)

Font (see page 5823)

Hint (see page 5827)

Hint (see page 13712)

72.1.11.5.24 TScreen.IconFont Property

Specifies the font used to display the labels on the icons in file selection dialogs.

Pascal

```
property IconFont: TFont;
```

C++

```
__property TFont IconFont;
```

Description

IconFont stores the Windows icon font setting. This font is used to label the icons on the desktop, and icons in the file selection dialogs.

See Also

Fonts (see page 14220)

HintFont (see page 14226)

MenuFont (see page 14227)

72.1.11.5.25 TScreen.Imes Property

Lists all input method editors (IMEs) installed on the system.

Pascal

```
property Imes: TStrings;
```

C++

```
__property TStrings Imes;
```

Description

Read Imes to obtain a list of all IMEs that are listed in the Windows registry. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application. IMEs vary in user interface, conversion dictionaries, and input method.

Individual controls can require a specific IME to process the keyboard input when they have focus. To specify an IME, the control sets its ImeName property to one of the strings in Imes.

Each IME name in Imes has an associated keyboard layout handle. A keyboard layout handle is a Windows handle that identifies the primary language and IME used to process keyboard input. Use the keyboard layout handle in Windows API calls for programming the IME. To get the keyboard layout handle for a particular IME, a control can use code such as the following:

```
HKeyboard := Screen.Imes.Objects(Screen.Imes.IndexOf(ImeName));
HKeyboard = Screen->Imes->Objects(Screen->Imes->IndexOf(ImeName));
```

See Also

DefaultIme (see page 14216)

DefaultKbLayout (see page 14217)

TStrings (see page 3029)

ImeName (see page 6208)

72.1.11.5.26 TScreen.MenuFont Property

Specifies the font used for menu commands.

Pascal

```
property MenuFont: TFont;
```

C++

```
__property TFont MenuFont;
```

Description

MenuFont stores the Windows menu font setting. This font is used for menu items in main menus or popup menus that do not

draw their own menu items in an OnDrawItem event handler.

See Also

Fonts (see page 14220)

IconFont (see page 14226)

HintFont (see page 14226)

OnDrawItem (see page 16159)

72.1.11.5.27 TScreen.MonitorCount Property

Specifies the number of monitors used to comprise the desktop.

Pascal

```
property MonitorCount: Integer;
```

C++

```
__property int MonitorCount;
```

Description

Use MonitorCount to determine the number of monitors used to comprise the desktop.

See Also

Monitors (see page 14228)

DesktopRect (see page 14218)

72.1.11.5.28 TScreen.Monitors Property

Provides access to an individual monitor used to comprise the desktop.

Pascal

```
property Monitors [Index: Integer]: TMonitor;
```

C++

```
__property TMonitor Monitors[int Index];
```

Description

Use Monitors to access an individual monitor used to comprise the desktop.

The Monitor with its Primary property set to true is the primary monitor.

See Also

MonitorCount (see page 14228)

DesktopRect (see page 14218)

Primary (see page 14195)

72.1.11.5.29 TScreen.PixelsPerInch Property

Indicates the number of screen pixels that make up a logical inch in the vertical direction.

Pascal

```
property PixelsPerInch: Integer;
```

C++

```
__property int PixelsPerInch;
```

Description

Read PixelsPerInch to convert between measurements in pixels and measurements in logical inches. PixelsPerInch is only accurate for vertical measurements. Most screens have a different scaling factor for horizontal measurements. The value in PixelsPerInch is set from system when the application loads.

See Also

Height (see page 14225)

PixelsPerInch (see page 13928)

72.1.11.5.30 TScreen.PrimaryMonitor Property

Provides a reference to the primary monitor.

Pascal

```
property PrimaryMonitor: TMonitor;
```

C++

```
__property TMonitor PrimaryMonitor;
```

Description

The coordinates of Monitors (see page 14228) in a multi-monitor configuration are related to the primary monitor. PrimaryMonitor returns a reference to the primary monitor. Use the Primary property to determine whether a specific monitor is the primary one.

See Also

TMonitor (see page 14191)

72.1.11.5.31 TScreen.SaveFocusedList Property

Pascal

```
property SaveFocusedList: TList;
```

C++

```
__property TList SaveFocusedList;
```

Description

This is SaveFocusedList, a member of class TScreen.

72.1.11.5.32 TScreen.Width Property

Indicates the horizontal size of the screen in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Read Width to learn the size of the user's screen in pixels. The size or position of objects placed on the screen, can then be set to ensure that the objects fit on screen and don't look crowded.

C++ Examples:

```
/*
The following code determines the height of all forms on the
screen and resizes any that are taller than the screen height.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < Screen->FormCount; i++)
  {
    if (Screen->Forms[i]->Top < 0)
    {
      Screen->Forms[i]->Height += Screen->Forms[i]->Top;
      Screen->Forms[i]->Top = 0;
    }
    if (Screen->Forms[i]->Left < 0)
    {
      Screen->Forms[i]->Width += Screen->Forms[i]->Left;
      Screen->Forms[i]->Left = 0;
    }
    if (Screen->Forms[i]->Height > Screen->Height)
      Screen->Forms[i]->Height = Screen->Height;
    if (Screen->Forms[i]->Width > Screen->Width)
      Screen->Forms[i]->Width = Screen->Width;
  }
}
```

Delphi Examples:

```
{
The following code determines the height of all forms on the
screen and resizes any that are taller than the screen height.
}
procedure TForm1.Button1Click(Sender: TObject);
var I: Integer;
begin
with Screen do
  for I := 0 to FormCount-1 do
    begin
    if Forms[I].Top < 0 then
    begin
      Forms[I].Height := Forms[I].Height + Forms[I].Top;
      Forms[I].Top := 0;
    end;
    if Forms[I].Left < 0 then
    begin
      Forms[I].Width := Forms[I].Width + Forms[I].Left;
      Forms[I].Left := 0;
    end;
    if Forms[I].Width > Width then
      Forms[I].Width := Width;
    if Forms[I].Height > Height then
      Forms[I].Height := Height;
    if Forms[I].Width > Width then
      Forms[I].Width := Width;
    end;
end;
```

See Also

Height (see page 14225)

Width (see page 5849)

DesktopWidth (see page 14219)

WorkAreaWidth (see page 14233)

Considerations When Dynamically Resizing Forms and Controls

72.1.11.5.33 TScreen.WorkAreaHeight Property

Specifies the height of the work area on the primary monitor.

Pascal

```
property WorkAreaHeight: Integer;
```

C++

```
__property int WorkAreaHeight;
```

Description

Use WorkAreaHeight to determine the height of the usable space on the primary monitor. WorkAreaHeight differs from the Height (see page 14225) property in that it adjusts the work area to subtract any space occupied by the task bar and any tool bars that sit on the desktop.

The primary monitor is the monitor whose Primary property is true.

See Also

WorkAreaLeft (see page 14231)

WorkAreaRect (see page 14232)

WorkAreaTop (see page 14232)

WorkAreaWidth (see page 14233)

DesktopHeight (see page 14217)

Height (see page 14225)

Monitors (see page 14228)

Primary (see page 14195)

72.1.11.5.34 TScreen.WorkAreaLeft Property

Specifies the left edge of the work area on the primary monitor.

Pascal

```
property WorkAreaLeft: Integer;
```

C++

```
__property int WorkAreaLeft;
```

Description

Use WorkAreaLeft to determine the left edge of the usable space on the primary monitor. This value indicates the number of pixels between the left edge of the primary monitor, and the space left after accounting for the task bar and any tool bars that sit

on the desktop.

The primary monitor is the monitor whose Primary property is true.

See Also

WorkAreaHeight (see page 14231)

WorkAreaRect (see page 14232)

WorkAreaTop (see page 14232)

WorkAreaWidth (see page 14233)

DesktopLeft (see page 14218)

Primary (see page 14195)

Monitors (see page 14228)

72.1.11.5.35 TScreen.WorkAreaRect Property

Specifies the boundaries of the usable work area on the primary monitor.

Pascal

```
property WorkAreaRect: TRect;
```

C++

```
__property TRect WorkAreaRect;
```

Description

Use WorkAreaRect to determine the coordinates of the usable space on the primary monitor. The usable work space is the surface of the primary monitor with the space occupied by the task bar and any tool bars subtracted off.

The primary monitor is the monitor with its Primary property set to true.

See Also

WorkAreaHeight (see page 14231)

WorkAreaLeft (see page 14231)

WorkAreaTop (see page 14232)

WorkAreaWidth (see page 14233)

DesktopRect (see page 14218)

Primary (see page 14195)

Monitors (see page 14228)

WorkAreaRect (see page 14197)

72.1.11.5.36 TScreen.WorkAreaTop Property

Specifies the top edge of the work area on the primary monitor.

Pascal

```
property WorkAreaTop: Integer;
```

C++

```
__property int WorkAreaTop;
```

Description

Use WorkAreaTop to determine the top edge of the usable space on the primary monitor. This value indicates the number of pixels between the top edge of the primary monitor, and the space left after accounting for the task bar and any tool bars that sit on the desktop.

The primary monitor is the monitor whose Primary property is true.

See Also

WorkAreaHeight (see page 14231)

WorkAreaRect (see page 14232)

WorkAreaLeft (see page 14231)

WorkAreaWidth (see page 14233)

DesktopTop (see page 14219)

Primary (see page 14195)

Monitors (see page 14228)

72.1.11.5.37 TScreen.WorkAreaWidth Property

Specifies the width of the work area on the primary monitor.

Pascal

```
property WorkAreaWidth: Integer;
```

C++

```
__property int WorkAreaWidth;
```

Description

Use WorkAreaWidth to determine the width of the usable space on the primary monitor. WorkAreaWidth differs from the Width (see page 14229) property in that it adjusts the work area to subtract any space occupied by the task bar and any tool bars that sit on the desktop.

The primary monitor is the monitor whose Primary property is true.

See Also

WorkAreaHeight (see page 14231)

WorkAreaRect (see page 14232)

WorkAreaLeft (see page 14231)

WorkAreaTop (see page 14232)

WorkAreaWidth

Width (see page 14229)

72.1.11.6 TScreen Events

The events of the TScreen class are listed here.

Public Events

	Name	Description
	OnActiveControlChange (see page 14234)	Occurs immediately after input focus changes to a new windowed control.
	OnActiveFormChange (see page 14236)	Occurs immediately after a new form becomes active in a multi-form application.

72.1.11.6.1 TScreen.OnActiveControlChange Event

Occurs immediately after input focus changes to a new windowed control.

Pascal

```
property OnActiveControlChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActiveControlChange;
```

Description

Write an OnActiveControlChange event handler to take specific action when input focus changes to a new control. The change in focus may be within the active form, or across forms to a new form that then becomes the active form.

When focus moves from one control to another, the following events occur.

1If the new focused control is in a different form, focus moves to the new form.

2Focus moves to the new active control.

3If the active form changed, an OnActiveFormChange (see page 14236) event occurs.

4An OnActiveControlChange event occurs.

C++ Examples:

```
/*
This example uses the OnActiveControlChange event to detect
when focus changes on the form. When focus changes, the hint
for the active control is displayed on the status bar. To use
this example, you must add a status bar to the form and set
its SimplePanel property to true.  Add a new public procedure,
ActiveControlChanged, to the TForm1 class declaration.
*/
void _fastcall TForm1::ActiveControlChanged(System::TObject *Sender)
{
  TWinControl *Active = NULL;
  for (int I = 0; I < Form1->ControlCount; I++)
  {
    TWinControl *Temp = dynamic_cast<TWinControl *>(Form1->Controls[I]);
    if (Temp && Temp->Focused())
      Active = Temp;
  }
  if ((Active != NULL) && (Active->Hint != ""))
    StatusBar2->SimpleText = GetLongHint(Active->Hint) + " focus";
}

void __fastcall TForm1::MouseOverChanged(TObject *Sender)
{
  TWinControl *Active = dynamic_cast<TWinControl *>(Sender);
  if (Active != NULL)
```

```
    StatusBar1->SimpleText = GetLongHint(Active->Hint) + " mouse over";
}

/*
Assign this method as the OnActiveControlChange event
handler by setting it from the form's OnCreate event handler:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Screen->OnActiveControlChange = ActiveControlChanged;
}

/*
Make sure you clean up the screen object when the form is
freed by adding this OnDestroy event handler to the form:
*/
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  Screen->OnActiveControlChange = 0;
}
```

Delphi Examples:

```
{
This example uses the OnActiveControlChange event to detect
when focus changes on the form. When focus changes, the hint
for the active control is displayed on the status bar. To use
this example, you must add a status bar to the form and set
its SimplePanel property to true.  Add a new public procedure,
ActiveControlChanged, to the TForm1 class declaration.
}

type
  TForm1 = class(TForm)
    StatusBar1, StatusBar2: TStatusBar;
    Button1: TButton;
    RadioButton1: TRadioButton;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure MouseOverChanged(Sender: TObject);
  private
    { Private declarations }
  public
    procedure ActiveControlChanged(Sender: TObject);
  end;

var
  Active : TWinControl;
  I: Integer;
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.ActiveControlChanged(Sender: TObject);
var
  Active : TWinControl;
  I: Integer;
begin
  Active := nil;
  for I:= 0 to ControlCount -1 do
  begin
    if (Controls[I] is TWinControl) then
      if (Controls[I] as TWinControl).Focused then
        Active := TWinControl(Controls[I]);
  end;
```

```
    if ((Active <> nil) and (Active.Hint <> '')) then
    begin
      StatusBar2.SimpleText := GetLongHint(Active.Hint) + ' focus';
    end
  end;

  procedure TForm1.MouseOverChanged(Sender: TObject);
  var
    Active : TWinControl;
    I: Integer;
  begin
    if (Sender is TWinControl) then
    begin
      Active := TWinControl(Sender);
      StatusBar1.SimpleText := GetLongHint(Active.Hint) + ' mouse over';
    end;
  end;

  {
  Assign this method as the OnActiveControlChange event
  handler by setting it from the form's OnCreate event handler:
  }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Screen.OnActiveControlChange := ActiveControlChanged;
  end;

  {
  Make sure you clean up the screen object when the form is
  freed by adding this OnDestroy event handler to the form:
  }
  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    Screen.OnActiveControlChange := nil;
  end;
```

See Also

ActiveControl (see page 14206)

OnActiveFormChange (see page 14236)

SetFocusedControl (see page 13887)

72.1.11.6.2 TScreen.OnActiveFormChange Event

Occurs immediately after a new form becomes active in a multi-form application.

Pascal

```
property OnActiveFormChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActiveFormChange;
```

Description

Write an OnActiveFormChange event handler to take specific action when a new form becomes active. OnActiveFormChange occurs when the active form for the application changes, not when a form becomes active because the application becomes active.

C++ Examples:

```
/*
This example uses a two forms with a button on the first
form. When the user clicks the button, the second form
```

```
appears. As the user switches between forms, the form that
is active is colored aqua.
Note:   You must include the unit header for form 2 to the
unit file for form 1, and add the public ColorForm method
to the definition of the TForm1 class.
*/
#include "Unit2.h"

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::ColorForm(TObject *Sender)
{
  Color = clWhite;
  Form2->Color = clWhite;
  Screen->ActiveForm->Color = clAqua;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Screen->OnActiveFormChange = ColorForm;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  // This stops ColorForm from being called during termination.
  Screen->OnActiveFormChange = NULL;
}
```

Delphi Examples:

```
{
This example uses a two forms with a button on the first
form. When the user clicks the button, the second form
appears. As the user switches between forms, the form that
is active is colored aqua.
Note:   You must include the unit header for form 2 to the
unit file for form 1, and add the public ColorForm method
to the header for form 1's unit.
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure ColorForm(Sender: TObject);
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

uses Unit2;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;
```

```
procedure TForm1.ColorForm(Sender: TObject);
begin
  Color := clWhite;
  Form2.Color := clWhite;
  Screen.ActiveForm.Color := clAqua;
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  Screen.OnActiveFormChange := ColorForm;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  // This stops ColorForm from being called during termination.
  Screen.OnActiveFormChange := nil;
end;
```

See Also

ActiveCustomForm (see page 14207)

OnActiveControlChange (see page 14234)

SetFocus (see page 13886)

72.1.12 TScrollBox Class

TScrollBox represents a scrolling area (a scroll box) in a window.

Class Hierarchy

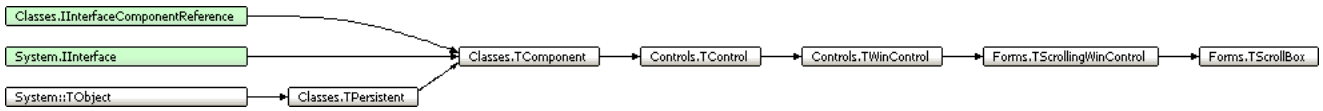

Pascal

```
TScrollBox = class(TScrollingWinControl);
```

C++

```
class TScrollBox : public TScrollingWinControl;
```

File

Forms

Description

Use TScrollBox to create a scroll box in a window.

One use of a scroll box is to prevent areas of a window, such as a toolbar or status bar built with TPanel components, from scrolling. To prevent a toolbar and status bar from scrolling, hide the window's scroll bars, and then position a scroll box in the client area of the window between the toolbar and status bar. The scroll bars associated with the scroll box will appear to belong to the window, but will scroll only the area inside the scroll box.

Another use of scroll boxes is to create multiple scrolling areas (views) in a window. Views are common in commercial word-processor, spreadsheet, and project management applications.

A scroll box can contain objects, such as TButton and TCheckBox objects.

See Also

TForm (see page 13992)

TFrame (see page 14114)

Grouping Controls

Scroll Boxes

72.1.12.1 TScrollBox Members

The following tables list the members exposed by TScrollBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 14324)	Adjusts the ClientRect property to exclude the scroll bars.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 14325)	Aligns the controls in the scrolling windowed control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.

	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	AutoScrollEnabled (see page 14325)	Indicates whether automatic scrolling is enabled.
	AutoScrollInView (see page 14325)	Scrolls a control into the visible area of the scrolling windowed control.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	ChangeScale (see page 14326)	Resizes the scrolling windowed control by a specific ratio.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 14250)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 14327)	Creates a Windows control corresponding to the TScrollingWinControl (see page 14312) object.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.

	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
V	DoFlipChildren (see page 14327)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 14250)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
V	Resizing (see page 14328)	This is Resizing, a member of class TScrollingWinControl.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.

	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).

	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 14249)	Creates and initializes a new TScrollBox (see page 14238) object.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 14323)	Removes the TScrollingWinControl (see page 14312) object from memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	DisableAutoRange (see page 14327)	Disables automatic scrolling.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EnableAutoRange (see page 14328)	Reenables automatic scrolling.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.

	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollInView (see page 14328)	Scrolls a control into the visible area of the scrolling windowed control.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.

R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 14278)	Occurs when an attempt is made to resize the control.
	OnClick (see page 14278)	Occurs when the user clicks the control.

	OnConstrainedResize (see page 14280)	Adjust resize constraints.
	OnContextPopup (see page 14281)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 14282)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 14282)	Occurs when another control is docked to the control.
	OnDockOver (see page 14284)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 14286)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 14288)	Occurs when the user drags an object over a control.
	OnEndDock (see page 14291)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 14291)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 14292)	Occurs when a control receives the input focus.
	OnExit (see page 14294)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 14296)	Returns the control's docking information.
	OnMouseActivate (see page 14298)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 14299)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 14301)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 14302)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 14302)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 14304)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 14306)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 14307)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 14307)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 14307)	Occurs immediately after the control is resized.
	OnStartDock (see page 14308)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 14309)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 14309)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 14251)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 14255)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 14255)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 14256)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 14256)	Specifies which edges of the control are beveled.
	BevelInner (see page 14257)	Specifies the cut of the inner bevel.
	BevelKind (see page 14257)	Specifies the control's bevel style.
	BevelOuter (see page 14258)	Specifies the cut of the outer bevel.
	BevelWidth (see page 14258)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 14258)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 14259)	Specifies whether the scroll box has a border.
	Color (see page 14259)	Specifies the background color of the control.
	Constraints (see page 14260)	Specifies the size constraints for the control.
	Ctl3D (see page 14261)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DockSite (see page 14261)	Specifies whether the control can be the target of drag-and-dock operations.

	DragCursor (see page 14262)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 14262)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 14263)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 14264)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 14265)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HorzScrollBar (see page 14329)	Represents the horizontal scroll bar for the scrolling windowed control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Padding (see page 14266)	Specifies the padding of a control.
	ParentBackground (see page 14266)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 14266)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 14267)	Determines where a control looks for its color information.
	ParentCtl3D (see page 14267)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 14268)	Determines where a control looks for its font information.
	ParentShowHint (see page 14268)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 14269)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 14273)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 14275)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 14275)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	VertScrollBar (see page 14330)	Represents the vertical scroll bar for the scrolling windowed control.
	Visible (see page 14276)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

72.1.12.2 TScrollBox.Create Constructor

Creates and initializes a new TScrollBox (see page 14238) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TScrollBox(TComponent * AOwner);
```

Description

Use Create to create and initialize a new TScrollBox (see page 14238) object.

AOwner is a component, typically the form, that is responsible for freeing the scroll box.

See Also

Owner (see page 2807)

72.1.12.3 TScrollBox Methods

The methods of the TScrollBox class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 14250)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	PaintWindow (see page 14250)	Renders the image of a windowed control.

72.1.12.3.1 TScrollBox.CreateParams Method

Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. CreateParams calls the inherited method and uses that Window class style, removing the CS_HREDRAW and CS_VREDRAW styles from it.

By removing these styles the scrolling windowed control will only redraw the invalidated regions, not the entire window.

See Also

CreateWnd (see page 14327)

CreateWindowHandle (see page 6137)

72.1.12.3.2 TScrollBox.PaintWindow Method

Renders the image of a windowed control.

Pascal

```
procedure PaintWindow(DC: HDC); override;
```

C++

```
virtual __fastcall PaintWindow(HDC DC);
```

Description

Call PaintWindow to repaint the control. PaintWindow sends a WM_PAINT message to the windowed control's DefaultHandler method, setting the message record's WParam field to the value passed in DC and the other parameter and result fields to zeros.

See Also

DefaultHandler (see page 6139)

PaintTo (see page 6167)

DoubleBuffered (see page 6205)

72.1.12.4 TScrollBox Properties

The properties of the TScrollBox class are listed here.

Published Properties

	Name	Description
	Align (see page 14251)	Determines how the control aligns within its container (parent control).
	Anchors (see page 14255)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 14255)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 14256)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 14256)	Specifies which edges of the control are beveled.
	BevelInner (see page 14257)	Specifies the cut of the inner bevel.
	BevelKind (see page 14257)	Specifies the control's bevel style.
	BevelOuter (see page 14258)	Specifies the cut of the outer bevel.
	BevelWidth (see page 14258)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 14258)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 14259)	Specifies whether the scroll box has a border.
	Color (see page 14259)	Specifies the background color of the control.
	Constraints (see page 14260)	Specifies the size constraints for the control.
	Ctl3D (see page 14261)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 14261)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 14262)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 14262)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 14263)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 14264)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 14265)	Controls the attributes of text written on or in the control.
	Padding (see page 14266)	Specifies the padding of a control.
	ParentBackground (see page 14266)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 14266)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 14267)	Determines where a control looks for its color information.
	ParentCtl3D (see page 14267)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 14268)	Determines where a control looks for its font information.
	ParentShowHint (see page 14268)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 14269)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 14273)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 14275)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 14275)	Determines if the user can tab to a control.
	Visible (see page 14276)	Determines whether the component appears onscreen.

72.1.12.4.1 TScrollBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
```

```
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
```

```
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
```

```
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
    end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

72.1.12.4.2 TScrollBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

72.1.12.4.3 TScrollBox.AutoScroll Property

Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

If AutoScroll is true, the scroll bars appear automatically when necessary. For example, if the user resizes the control so that some of its controls are partially obscured, scroll bars appear.

If AutoScroll is false, scroll bars don't appear automatically. In this case, use the HorzScrollBar and VertScrollBar properties to make scroll bars appear.

See Also

AutoScrollInView (see page 14325)

HorzScrollBar (see page 14329)

VertScrollBar (see page 14330)

72.1.12.4.4 TScrollBox.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints (see page 5812)

CanAutoSize (see page 5741)

72.1.12.4.5 TScrollBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

72.1.12.4.6 TScrollBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

72.1.12.4.7 TScrollBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

72.1.12.4.8 TScrollBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

72.1.12.4.9 TScrollBox.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

72.1.12.4.10 TScrollBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

72.1.12.4.11 TScrollBox.BorderStyle Property

Specifies whether the scroll box has a border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to get or set whether the scroll box has a border.

72.1.12.4.12 TScrollBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
```

```
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

72.1.12.4.13 TScrollBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

72.1.12.4.14 TScrollBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

72.1.12.4.15 TScrollBox.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite: Boolean;
```

C++

```
__property Boolean DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
```

```
    if ((DockClientCount == 2) && (NewTarget != this))        // Self?
      PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
    UpdateCaption(Client);
  }
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

72.1.12.4.16 TScrollBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

72.1.12.4.17 TScrollBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

72.1.12.4.18 TScrollBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
```

```
  example requires a populated TreeView.  Also, the TTreeView
  DragMode property set to dmAutomatic, and the TTreeView
  OnDragOver event handler must be implemented to accept the
  drop.
  }
  procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
  var
    AnItem: TTreeNode;
    AttachMode: TNodeAttachMode;
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

72.1.12.4.19 TScrollBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

72.1.12.4.20 TScrollBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
```

```
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

72.1.12.4.21 TScrollBox.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding: TPadding;
```

C++

```
__property TPadding Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding (see page 6103)

72.1.12.4.22 TScrollBox.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground: Boolean;
```

C++

```
__property Boolean ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

72.1.12.4.23 TScrollBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's

BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

72.1.12.4.24 TScrollBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

72.1.12.4.25 TScrollBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

72.1.12.4.26 TScrollBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

72.1.12.4.27 TScrollBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

72.1.12.4.28 TScrollBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
```

```
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
```

```
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
```

```
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

72.1.12.4.29 TScrollBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

72.1.12.4.30 TScrollBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

72.1.12.4.31 TScrollBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

72.1.12.4.32 TScrollBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
```

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

72.1.12.5 TScrollBox Events

The events of the TScrollBox class are listed here.

Published Events

	Name	Description
	OnCanResize (see page 14278)	Occurs when an attempt is made to resize the control.
	OnClick (see page 14278)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 14280)	Adjust resize constraints.
	OnContextPopup (see page 14281)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 14282)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 14282)	Occurs when another control is docked to the control.
	OnDockOver (see page 14284)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 14286)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 14288)	Occurs when the user drags an object over a control.
	OnEndDock (see page 14291)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 14291)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 14292)	Occurs when a control receives the input focus.
	OnExit (see page 14294)	Occurs when the input focus shifts away from one control to another.

	OnGetSiteInfo (see page 14296)	Returns the control's docking information.
	OnMouseActivate (see page 14298)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 14299)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 14301)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 14302)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 14302)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 14304)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 14306)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 14307)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 14307)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 14307)	Occurs immediately after the control is resized.
	OnStartDock (see page 14308)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 14309)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 14309)	Occurs when the application tries to undock a control that is docked to the windowed control.

72.1.12.5.1 TScrollBox.OnCanResize Event

Occurs when an attempt is made to resize the control.

Pascal

```
property OnCanResize: TCanResizeEvent;
```

C++

```
__property TCanResizeEvent OnCanResize;
```

Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

See Also

OnConstrainedResize (see page 5853)

SetBounds (see page 5783)

OnResize (see page 5870)

72.1.12.5.2 TScrollBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
```

```
  edit control placed somewhere on the client area of the
  sheet.  Unchecking the checkbox will cause the client area
  (not the tab area) of the Tab Control to become invisible.
  }
  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    PageControl1.ActivePage.Visible := CheckBox1.Checked;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    CheckBox1.Checked := PageControl1.ActivePage.Visible;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

72.1.12.5.3 TScrollBox.OnConstrainedResize Event

Adjust resize constraints.

Pascal

```
property OnConstrainedResize: TConstrainedResizeEvent;
```

C++

```
__property TConstrainedResizeEvent OnConstrainedResize;
```

Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being

applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

Note: The OnConstrainedResize handler is called immediately after the OnCanResize handler.

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

Constraints (see page 5812)

OnResize (see page 5870)

CanAutoSize (see page 5741)

72.1.12.5.4 TScrollBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

72.1.12.5.5 TScrollBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

72.1.12.5.6 TScrollBox.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop: TDockDropEvent;
```

C++

```
__property TDockDropEvent OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
```

```
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
```

```
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockOver (see page 6216)

72.1.12.5.7 TScrollBox.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver: TDockOverEvent;
```

C++

```
__property TDockOverEvent OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
```

```
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
```

```
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnGetSiteInfo (see page 6223)

TDragDockObject (see page 6009)

72.1.12.5.8 TScrollBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
```

```
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:
```

```
procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

72.1.12.5.9 TScrollBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler

is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;
```

```
procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

72.1.12.5.10 TScrollBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

72.1.12.5.11 TScrollBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

72.1.12.5.12 TScrollBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
```

```
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

72.1.12.5.13 TScrollBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

72.1.12.5.14 TScrollBox.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo: TGetSiteInfoEvent;
```

C++

```
__property TGetSiteInfoEvent OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
```

```
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
```

```
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

TDragDockObject (see page 6009)

72.1.12.5.15 TScrollBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of

whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

72.1.12.5.16 TScrollBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
```

```
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
```

```
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

72.1.12.5.17 TScrollBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

72.1.12.5.18 TScrollBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

72.1.12.5.19 TScrollBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;
```

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

72.1.12.5.20 TScrollBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
```

```
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
```

```
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

72.1.12.5.21 TScrollBox.OnMouseWheel Event

Occurs when the mouse wheel is rotated.

Pascal

```
property OnMouseWheel: TMouseWheelEvent;
```

C++

```
__property TMouseWheelEvent OnMouseWheel;
```

Description

Write code in the OnMouseWheel event handler to respond to mouse wheel messages. If there is no OnMouseWheel event handler, or if the mouse wheel message is not handled in the OnMouseWheel event handler, then an OnMouseWheelUp or

OnMouseWheelDown event occurs, depending on the direction the mouse wheel was rotated.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheelUp (see page 5870)

72.1.12.5.22 TScrollBox.OnMouseWheelDown Event

Occurs when the mouse wheel is rotated downward.

Pascal

```
property OnMouseWheelDown: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelDown;
```

Description

Write code in the OnMouseWheelDown event handler to perform actions when the mouse wheel is rotated downward. OnMouseWheelDown occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheel (see page 5869)

OnMouseWheelUp (see page 5870)

72.1.12.5.23 TScrollBox.OnMouseWheelUp Event

Occurs when the mouse wheel is rotated upward.

Pascal

```
property OnMouseWheelUp: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelUp;
```

Description

Write code in the OnMouseWheelUp event handler to perform actions when the mouse wheel is rotated upward. OnMouseWheelUp occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheel (see page 5869)

72.1.12.5.24 TScrollBox.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

72.1.12.5.25 TScrollBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

72.1.12.5.26 TScrollBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

72.1.12.5.27 TScrollBox.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock: TUnDockEvent;
```

C++

```
__property TUnDockEvent OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
```

```
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;
```

```
procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

72.1.13 TScrollingWinControl Class

TScrollingWinControl is the base class for controls that support scrolling.

Class Hierarchy

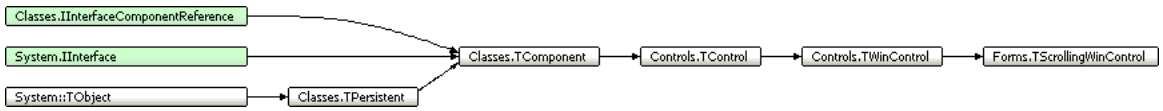

Pascal

```
TScrollingWinControl = class(TWinControl);
```

C++

```
class TScrollingWinControl : public TWinControl;
```

File

Forms

Description

Use TScrollingWinControl as a base class when defining a custom windowed control that supports scrolling.

Typically, a scrolling windowed control has horizontal and vertical scroll bars and scrolls a child control into view when the child control receives focus.

Examples of scrolling windowed controls include TForm (see page 13992) and TScrollBox (see page 14238) objects.

The scrolling window's BiDiMode determines whether the left or right side of the window's contents are displayed by default. The BiDiMode does not affect the window's coordinate system. The scroll bars must be adjusted programmatically according to the system locale.

See Also

TForm (see page 13992)

TScrollBox (see page 14238)

72.1.13.1 TScrollingWinControl Members

The following tables list the members exposed by TScrollingWinControl.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.

	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 14324)	Adjusts the ClientRect property to exclude the scroll bars.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 14325)	Aligns the controls in the scrolling windowed control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
V	AutoScrollEnabled (see page 14325)	Indicates whether automatic scrolling is enabled.
V	AutoScrollInView (see page 14325)	Scrolls a control into the visible area of the scrolling windowed control.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	ChangeScale (see page 14326)	Resizes the scrolling windowed control by a specific ratio.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 14326)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 14327)	Creates a Windows control corresponding to the TScrollingWinControl (see page 14312) object.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.

	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
V	DoFlipChildren (see page 14327)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.

	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.

	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	Resizing (see page 14328)	This is Resizing, a member of class TScrollingWinControl.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoScroll (see page 14329)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 14323)	Creates and initializes a new TScrollingWinControl (see page 14312) object.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 14323)	Removes the TScrollingWinControl (see page 14312) object from memory.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	DisableAutoRange (see page 14327)	Disables automatic scrolling.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EnableAutoRange (see page 14328)	Reenables automatic scrolling.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollInView (see page 14328)	Scrolls a control into the visible area of the scrolling windowed control.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14332)	Occurs when an object with custom alignment is aligned.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HorzScrollBar (see page 14329)	Represents the horizontal scroll bar for the scrolling windowed control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	VertScrollBar (see page 14330)	Represents the vertical scroll bar for the scrolling windowed control.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

72.1.13.2 TScrollingWinControl.Create Constructor

Creates and initializes a new TScrollingWinControl (see page 14312) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TScrollingWinControl(TComponent * AOwner);
```

Description

Use Create to create and initialize a new TScrollingWinControl (see page 14312) object.

AOwner is a TComponent object that owns the scrolling windowed control.

See Also

AutoScroll (see page 14329)

HorzScrollBar (see page 14329)

Create (see page 6123)

VertScrollBar (see page 14330)

72.1.13.3 TScrollingWinControl.Destroy Destructor

Removes the TScrollingWinControl (see page 14312) object from memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScrollingWinControl();
```

Description

Do not call Destroy.TScrollingWinControl (see page 14312) Instead, call Free, which calls DestroyWinControl if the TScrollingWinControl (see page 14312) object is not nil.

See Also

Destroy (see page 6124)

Free (see page 22300)

72.1.13.4 TScrollingWinControl Methods

The methods of the TScrollingWinControl class are listed here.

Protected Methods

	Name	Description
	AdjustClientRect (see page 14324)	Adjusts the ClientRect property to exclude the scroll bars.
	AlignControls (see page 14325)	Aligns the controls in the scrolling windowed control.
	AutoScrollEnabled (see page 14325)	Indicates whether automatic scrolling is enabled.
	AutoScrollInView (see page 14325)	Scrolls a control into the visible area of the scrolling windowed control.
	ChangeScale (see page 14326)	Resizes the scrolling windowed control by a specific ratio.
	CreateParams (see page 14326)	Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.
	CreateWnd (see page 14327)	Creates a Windows control corresponding to the TScrollingWinControl (see page 14312) object.
	DoFlipChildren (see page 14327)	Flips the control's children.
	Resizing (see page 14328)	This is Resizing, a member of class TScrollingWinControl.

Public Methods

	Name	Description
	DisableAutoRange (see page 14327)	Disables automatic scrolling.
	EnableAutoRange (see page 14328)	Reenables automatic scrolling.
	ScrollInView (see page 14328)	Scrolls a control into the visible area of the scrolling windowed control.

72.1.13.4.1 TScrollingWinControl.AdjustClientRect Method

Adjusts the ClientRect property to exclude the scroll bars.

Pascal

```
procedure AdjustClientRect(var Rect: TRect); override;
```

C++

```
virtual __fastcall AdjustClientRect(TRect Rect);
```

Description

AdjustClientRect is called internally when the control needs accurate information on where to place child controls within the client area. It indents Rect to exclude the area taken by the scroll bars, then calls the inherited AdjustClientRect method.

See Also

ClientRect

72.1.13.4.2 TScrollingWinControl.AlignControls Method

Aligns the controls in the scrolling windowed control.

Pascal

```
procedure AlignControls(AControl: TControl; var ARect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect ARect);
```

Description

Use AlignControls to align the controls in the scrolling control according to the values of their Align properties.

AControl specifies the control that takes precedence in alignment when other controls have the same Align value. AControl can be nil (Delphi) or NULL (C++).

ARect indicates the area in the scrolling control that contains the controls to align. ARect is automatically set to the entire area of the scrolling control.

If AutoScroll (see page 14329) is true and the scrolling control is not being resized, the ranges of the scroll bars on the scrolling control are automatically recalculated.

After recalculating the ranges of the scroll bars and setting the value of ARect, AlignControls calls the inherited AlignControls method, passing the AControl and ARect objects.

See Also

Align (see page 5798)

AutoScroll (see page 14329)

AlignControls (see page 6129)

72.1.13.4.3 TScrollingWinControl.AutoScrollEnabled Method

Indicates whether automatic scrolling is enabled.

Pascal

```
function AutoScrollEnabled: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean AutoScrollEnabled();
```

Description

Call AutoScrollEnabled to determine whether automatic scrolling is enabled.

AutoScrollEnabled returns true if the control is not a dock site using the dock manager and the AutoSize property is false.

See Also

AutoScroll

AutoSize (see page 5803)

UseDockManager (see page 6212)

72.1.13.4.4 TScrollingWinControl.AutoScrollInView Method

Scrolls a control into the visible area of the scrolling windowed control.

Pascal

```
procedure AutoScrollInView(AControl: TControl); virtual;
```

C++

```
virtual __fastcall AutoScrollInView(TControl * AControl);
```

Description

Use AutoScrollInView to scroll the control specified by the AControl parameter into the visible area of the scrolling windowed control.

Unlike ScrollInView (see page 14328), AutoScrollInView does not perform scrolling when the scrolling windowed control or the specified control are being loaded from a stream.

AutoScrollInView calls ScrollInView (see page 14328) to perform any needed scrolling.

See Also

ScrollInView (see page 14328)

72.1.13.4.5 TScrollingWinControl.ChangeScale Method

Resizes the scrolling windowed control by a specific ratio.

Pascal

```
procedure ChangeScale(M: Integer; D: Integer); override;
```

C++

```
virtual __fastcall ChangeScale(int M, int D);
```

Description

Use ChangeScale to resize the scrolling windowed control by a specific ratio.

M is the multiplier of the ratio; D is the divisor of the ratio. For example, to resize the scrolling windowed control to four-fifths of its current size, set M to four and D to five.

See Also

ChangeScale (see page 5743)

72.1.13.4.6 TScrollingWinControl.CreateParams Method

Initializes the window creation parameters so that the entire window is not redrawn when the width or height of the client area changes.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 14327) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. CreateParams calls the inherited method and uses that Window class style, removing the CS_HREDRAW and CS_VREDRAW styles from it.

By removing these styles the scrolling windowed control will only redraw the invalidated regions, not the entire window.

See Also

CreateWnd (see page 14327)

CreateWindowHandle (see page 6137)

72.1.13.4.7 TScrollingWinControl.CreateWnd Method

Creates a Windows control corresponding to the TScrollingWinControl (see page 14312) object.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the window must be destroyed and recreated to reflect property changes.

See Also

CreateWnd (see page 6137)

DestroyWnd (see page 6141)

72.1.13.4.8 TScrollingWinControl.DisableAutoRange Method

Disables automatic scrolling.

Pascal

```
procedure DisableAutoRange;
```

C++

```
__fastcall DisableAutoRange();
```

Description

Use DisableAutoRange to temporarily disable scrolling when rearranging controls in a scrolling windowed control. Use the EnableAutoRange (see page 14328) method to turn scrolling on again when finished rearranging the controls.

See Also

EnableAutoRange (see page 14328)

AutoScroll (see page 14329)

72.1.13.4.9 TScrollingWinControl.DoFlipChildren Method

Flips the control's children.

Pascal

```
procedure DoFlipChildren; override;
```

C++

```
virtual __fastcall DoFlipChildren();
```

Description

The FlipChildren method calls DoFlipChildren to reverse the positions of child controls. DoFlipChildren moves the control's children to opposite positions on the control; that is, children on the left side of the control are moved to the right side of the control and vice versa. In addition, DoFlipChildren moves the vertical scroll bars to the other side of the control.

Note: DoFlipChildren is a protected helper method for the public FlipChildren method. Unlike FlipChildren, DoFlipChildren does not adjust the alignment of controls that are aligned to one of the edges of the windowed control.

See Also

FlipChildren (see page 6150)

BiDiMode (see page 5803)

72.1.13.4.10 TScrollingWinControl.EnableAutoRange Method

Reenables automatic scrolling.

Pascal

```
procedure EnableAutoRange;
```

C++

```
__fastcall EnableAutoRange();
```

Description

Use EnableAutoRange to reenable scrolling after turning it off. This method can be used with the DisableAutoRange (see page 14327) method to turn scrolling on and off.

See Also

DisableAutoRange (see page 14327)

AutoScroll (see page 14329)

72.1.13.4.11 TScrollingWinControl.Resizing Method

Pascal

```
procedure Resizing(State: TWindowState); virtual;
```

C++

```
virtual __fastcall Resizing(TWindowState State);
```

Description

This is Resizing, a member of class TScrollingWinControl.

72.1.13.4.12 TScrollingWinControl.ScrollInView Method

Scrolls a control into the visible area of the scrolling windowed control.

Pascal

```
procedure ScrollInView(AControl: TControl);
```

C++

```
__fastcall ScrollInView(TControl * AControl);
```

Description

Use ScrollInView to ensure that a specified control is in the visible area of the scrolling windowed control. AControl specifies the TControl object to scroll into view.

Note: If the ComponentState property for AControl or the scrolling windowed control might be csLoading, use AutoScrollInView (see page 14325) instead.

See Also

AutoScrollInView (see page 14325)

ComponentState (see page 2804)

72.1.13.5 TScrollingWinControl Properties

The properties of the TScrollingWinControl class are listed here.

Protected Properties

	Name	Description
	AutoScroll (see page 14329)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Published Properties

	Name	Description
	HorzScrollBar (see page 14329)	Represents the horizontal scroll bar for the scrolling windowed control.
	VertScrollBar (see page 14330)	Represents the vertical scroll bar for the scrolling windowed control.

72.1.13.5.1 TScrollingWinControl.AutoScroll Property

Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

If AutoScroll is true, the scroll bars appear automatically when necessary. For example, if the user resizes the control so that some of its controls are partially obscured, scroll bars appear.

If AutoScroll is false, scroll bars don't appear automatically. In this case, use the HorzScrollBar (see page 14329) and VertScrollBar (see page 14330) properties to make scroll bars appear.

See Also

AutoScrollInView (see page 14325)

HorzScrollBar (see page 14329)

VertScrollBar (see page 14330)

72.1.13.5.2 TScrollingWinControl.HorzScrollBar Property

Represents the horizontal scroll bar for the scrolling windowed control.

Pascal

```
property HorzScrollBar: TControlScrollBar;
```

C++

```
__property TControlScrollBar HorzScrollBar;
```

Description

Use HorzScrollBar to hide, show, or manipulate the horizontal scroll bar for the scrolling windowed control.

C++ Examples:

```
/*
This example starts the form in a position scrolled so that
the right hand edge shows:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  /* set the width of the form window to display 300 pixels in the client area */
  ClientWidth = 300;
  /* now set the range to twice the client width.
   This means that the form has a logical size
   that is twice as big as the physical window.
   Note that Range must always be at least as big as ClientWidth! */
  HorzScrollBar->Range = 600;
  /* start the form fully scrolled */
  HorzScrollBar->Position = HorzScrollBar->Range - ClientWidth;
  /* clicking the scroll arrows moves the form 10 pixels */
  HorzScrollBar->Increment = 10;
  HorzScrollBar->Visible = true;  // Show the scrollbar
}
```

Delphi Examples:

```
{
This example starts the form in a position scrolled so that
the right hand edge shows:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ClientWidth := 300;
  with HorzScrollBar do
  begin
    { Set the range to twice the ClientWidth of the form }
    { This means that the form's logical size is twice as big }
    { as the physical window. }
    { Note that Range must always be larger than the ClientWidth }
    Range := 600;
    Position := Range - ClientWidth; { start form out fully scrolled }
    Increment := 10;  { clicking the scroll arrows moves the form 10 pixels }
    Visible := True;  { Show the scrollbar }
  end;
end;
```

See Also

AutoScroll (see page 14329)

VertScrollBar (see page 14330)

TControlScrollBar (see page 13744)

72.1.13.5.3 TScrollingWinControl.VertScrollBar Property

Represents the vertical scroll bar for the scrolling windowed control.

Pascal

```
property VertScrollBar: TControlScrollBar;
```

C++

```
__property TControlScrollBar VertScrollBar;
```

Description

Use VertScrollBar to hide, show, or manipulate the vertical scroll bar for the scrolling windowed control.

C++ Examples:

```
/*
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  const int iPageDelta = 10;

  if (Key == VK_NEXT)
    VertScrollBar->Position += iPageDelta;
  else if (Key == VK_PRIOR)
    VertScrollBar->Position -= iPageDelta;
}
```

Delphi Examples:

```
{
The following example demonstrates form scrolling with the
PgUp and PgDn keys.
}
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
const
  PageDelta = 10;
begin
  With VertScrollbar do
    if Key = VK_NEXT then
      Position := Position + PageDelta
    else if Key = VK_PRIOR then
      Position := Position - PageDelta;
end;
```

See Also

AutoScroll (see page 14329)

HorzScrollBar (see page 14329)

TControlScrollBar (see page 13744)

72.1.13.6 TScrollingWinControl Events

The events of the TScrollingWinControl class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14332)	Occurs when an object with custom alignment is aligned.

72.1.13.6.1 TScrollingWinControl.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore: TAlignInsertBeforeEvent;
```

C++

```
__property TAlignInsertBeforeEvent OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore (see page 6137)

Align (see page 5798)

TAlignInsertBeforeEvent (see page 6293)

72.1.13.6.2 TScrollingWinControl.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition: TAlignPositionEvent;
```

C++

```
__property TAlignPositionEvent OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition (see page 6138)

Align (see page 5798)

Anchors (see page 5802)

TAlignInfo (see page 6269)

TAlignPositionEvent (see page 6293)

72.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IDesignerHook (see page 14334)	IDesignerHook is an interface that allows component writers to interact with the form designer in the IDE.
	IOleForm (see page 14340)	IOleForm is designed to communicate with OLE containers. To embed a VCL form in an OLE container, CodeGear recommends using ActiveForms and TOleControl instead of IOleForm.

72.2.1 IDesignerHook Interface

IDesignerHook is an interface that allows component writers to interact with the form designer in the IDE.

Class Hierarchy

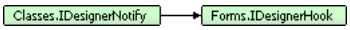

Pascal

```
IDesignerHook = interface(IDesignerNotify);
```

C++

```
__interface IDesignerHook : public IDesignerNotify;
```

File

Forms

Description

IDesignerHook provides access to the form currently under construction in the form designer, and influences the behavior of the designer by specifying whether the object under construction is a form (as opposed to a composite component), drawing the grid that assists in layout, and so on.

Component writers can access the current IDesignerHook interface using the Designer property of a form.

Note: C++ method declarations that use IDesignerHook use the _di_IDesignerHook type instead. This is a DelphiInterface wrapper around the IDesignerHook interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IDesignerHook > _di_IDesignerHook;
```

See Also

TCustomForm (see page 13849)

TComponentEditor (see page 10927)

TPropertyEditor (see page 11019)

IFormDesigner

72.2.1.1 IDesignerHook Members

The following tables list the members exposed by IDesignerHook.

Public Methods

	Name	Description
	GetCustomForm (see page 14335)	Specifies the value returned when the Form (see page 14339) property is read.
	GetIsControl (see page 14336)	Specifies the value returned when the IsControl (see page 14339) property is read.
	GetRoot (see page 14336)	Returns the current entity being edited by the form designer.
	IsDesignMsg (see page 14336)	Determines when the designer should handle a Windows message.
	Modified (see page 3196)	Notifies property and component editors when a change is made to a component.
	Notification (see page 3196)	Allows the designer to respond when a notification is sent to the form.
	PaintGrid (see page 14337)	Paints the alignment grid on the form's canvas.
	PaintMenu (see page 14337)	This is PaintMenu, a member of class IDesignerHook.
	SetCustomForm (see page 14337)	Specifies what happens when the Form (see page 14339) property is set.
	SetIsControl (see page 14338)	Specifies what happens when the IsControl (see page 14339) property is set.
	UniqueName (see page 14338)	Generates a unique name from a specified base string.
	ValidateRename (see page 14338)	Restricts the names that can be given to components on the form that is being designed.

Public Properties

	Name	Description
	Form (see page 14339)	Provides a reference to the form currently being designed.
	IsControl (see page 14339)	Specifies whether the form is in its form state or its control state.

72.2.1.2 IDesignerHook Methods

The methods of the IDesignerHook class are listed here.

Public Methods

	Name	Description
	GetCustomForm (see page 14335)	Specifies the value returned when the Form (see page 14339) property is read.
	GetIsControl (see page 14336)	Specifies the value returned when the IsControl (see page 14339) property is read.
	GetRoot (see page 14336)	Returns the current entity being edited by the form designer.
	IsDesignMsg (see page 14336)	Determines when the designer should handle a Windows message.
	PaintGrid (see page 14337)	Paints the alignment grid on the form's canvas.
	PaintMenu (see page 14337)	This is PaintMenu, a member of class IDesignerHook.
	SetCustomForm (see page 14337)	Specifies what happens when the Form (see page 14339) property is set.
	SetIsControl (see page 14338)	Specifies what happens when the IsControl (see page 14339) property is set.
	UniqueName (see page 14338)	Generates a unique name from a specified base string.
	ValidateRename (see page 14338)	Restricts the names that can be given to components on the form that is being designed.

72.2.1.2.1 IDesignerHook.GetCustomForm Method

Specifies the value returned when the Form (see page 14339) property is read.

Pascal

```
function GetCustomForm: TCustomForm;
```

C++

```
__fastcall TCustomForm GetCustomForm();
```

Description

Implement GetCustomForm to specify the value returned when the Form (see page 14339) property is read.

See Also

Form (see page 14339)

SetCustomForm (see page 14337)

72.2.1.2.2 IDesignerHook.GetIsControl Method

Specifies the value returned when the IsControl (see page 14339) property is read.

Pascal

```
function GetIsControl: Boolean;
```

C++

```
__fastcall Boolean GetIsControl();
```

Description

Implement GetIsControl to specify the value returned when the IsControl (see page 14339) property is read.

See Also

IsControl (see page 14339)

SetIsControl (see page 14338)

72.2.1.2.3 IDesignerHook.GetRoot Method

Returns the current entity being edited by the form designer.

Pascal

```
function GetRoot: TComponent;
```

C++

```
__fastcall TComponent * GetRoot();
```

Description

Call GetRoot to obtain the object the form designer is working on. If this object is a descendant of TCustomForm (see page 13849), GetRoot returns the value of the Form (see page 14339) property.

Use GetRoot rather than the Form (see page 14339) property, to ensure that code works even when editing a data module, remote data module, active form, or other entity that is not a descendant of TCustomForm (see page 13849).

See Also

Form (see page 14339)

ContainerWindow

72.2.1.2.4 IDesignerHook.IsDesignMsg Method

Determines when the designer should handle a Windows message.

Pascal

```
function IsDesignMsg(Sender: TControl; var Message: TMessage): Boolean;
```

C++

```
__fastcall Boolean IsDesignMsg(TControl * Sender, TMessage Message);
```

Description

IsDesignMsg is called for each message sent to a component in the designer. This method returns true if the message is a design message, meaning one the designer should handle for the component.

See Also

WindowProc (see page 5872)

WndProc (see page 6187)

72.2.1.2.5 IDesignerHook.PaintGrid Method

Paints the alignment grid on the form's canvas.

Pascal

```
procedure PaintGrid;
```

C++

```
__fastcall PaintGrid();
```

Description

Forms (see page 13657) that are associated with a designer call the PaintGrid method in response to Windows paint messages. If there is no designer associated with the form, the form's Paint method is called instead.

See Also

Paint (see page 13882)

72.2.1.2.6 IDesignerHook.PaintMenu Method

Pascal

```
procedure PaintMenu;
```

C++

```
__fastcall PaintMenu();
```

Description

This is PaintMenu, a member of class IDesignerHook.

72.2.1.2.7 IDesignerHook.SetCustomForm Method

Specifies what happens when the Form (see page 14339) property is set.

Pascal

```
procedure SetCustomForm(Value: TCustomForm);
```

C++

```
__fastcall SetCustomForm(TCustomForm Value);
```

Description

Implement SetCustomForm to specify what happens when the Form (see page 14339) property is set.

See Also

Form (see page 14339)

GetCustomForm (see page 14335)

72.2.1.2.8 IDesignerHook.SetIsControl Method

Specifies what happens when the IsControl (see page 14339) property is set.

Pascal

```
procedure SetIsControl(Value: Boolean);
```

C++

```
__fastcall SetIsControl(Boolean Value);
```

Description

Implement SetIsControl specify what happens when the IsControl (see page 14339) property is set.

See Also

IsControl (see page 14339)

GetIsControl (see page 14336)

72.2.1.2.9 IDesignerHook.UniqueName Method

Generates a unique name from a specified base string.

Pascal

```
function UniqueName(const BaseName: string): string;
```

C++

```
__fastcall AnsiString UniqueName(const AnsiString BaseName);
```

Description

Call UniqueName to automatically generate a unique name for a component. Specify the base string for the name by the BaseName parameter. UniqueName appends a number to BaseName to ensure that there are no name-space conflicts.

See Also

GetComponentName

72.2.1.2.10 IDesignerHook.ValidateRename Method

Restricts the names that can be given to components on the form that is being designed.

Pascal

```
procedure ValidateRename(AComponent: TComponent; const CurName: string; const NewName: string);
```

C++

```
__fastcall ValidateRename(TComponent * AComponent, const AnsiString CurName, const AnsiString
NewName);
```

Description

Call ValidateRename to determine if the component specified by AComponent, which currently has the name specified by CurName, can be renamed to the string specified by NewName. This method is usually called by the form's ValidateRename method to provide tighter restrictions on renaming, such as prohibiting reserved words.

See Also

ValidateRename (see page 13892)

72.2.1.3 IDesignerHook Properties

The properties of the IDesignerHook class are listed here.

Public Properties

	Name	Description
	Form (see page 14339)	Provides a reference to the form currently being designed.
	IsControl (see page 14339)	Specifies whether the form is in its form state or its control state.

72.2.1.3.1 IDesignerHook.Form Property

Provides a reference to the form currently being designed.

Pascal

```
property Form: TCustomForm;
```

C++

```
__property TCustomForm Form;
```

Description

Use Form to access the form, if any, that is being designed. When the user specifies changes in the form designer, the changes can be made to the form specified by the Form property.

Note: Not every designer has a form. For example, when the designer is used for a data module, remote data module, or active form, the Form property should not be used. Instead, use the object that the GetRoot (see page 14336) method returns.

See Also

Designer (see page 13910)

IsControl (see page 14339)

GetRoot (see page 14336)

72.2.1.3.2 IDesignerHook.IsControl Property

Specifies whether the form is in its form state or its control state.

Pascal

```
property IsControl: Boolean;
```

C++

```
__property Boolean IsControl;
```

Description

If the designer is used to create a control, rather than a form, set IsControl to true when the user finishes designing the control.

See Also

IsControl (see page 5829)

Form (see page 14339)

72.2.2 IOleForm Interface

IOleForm is designed to communicate with OLE containers. To embed a VCL form in an OLE container, CodeGear recommends using ActiveForms and TOleControl instead of IOleForm.

Class Hierarchy

Forms.IOleForm

Pascal

```
IOleForm = interface;
```

C++

```
__interface IOleForm;
```

File

Forms

See Also

TOleControl (see page 18030)

72.2.2.1 IOleForm Members

The following tables list the members exposed by IOleForm.

Public Methods

	Name	Description
	OnDestroy (see page 14340)	This is OnDestroy, a member of class IOleForm.
	OnResize (see page 14341)	This is OnResize, a member of class IOleForm.

72.2.2.2 IOleForm Methods

The methods of the IOleForm class are listed here.

Public Methods

	Name	Description
	OnDestroy (see page 14340)	This is OnDestroy, a member of class IOleForm.
	OnResize (see page 14341)	This is OnResize, a member of class IOleForm.

72.2.2.2.1 IOleForm.OnDestroy Method

Pascal

```
procedure OnDestroy;
```

C++

```
__fastcall OnDestroy();
```

Description

This is OnDestroy, a member of class IOleForm.

72.2.2.2.2 IOleForm.OnResize Method

Pascal

```
procedure OnResize;
```

C++

```
__fastcall OnResize();
```

Description

This is OnResize, a member of class IOleForm.

72.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AllocateHWnd (see page 14342)	Creates a window that implements a specified window procedure.
	DeallocateHWnd (see page 14343)	Frees a window that was created using AllocateHWnd (see page 14342).
	DisableTaskWindows (see page 14343)	This is function Forms.DisableTaskWindows.
	DoneCtl3D (see page 14344)	This is function Forms.DoneCtl3D.
	EnableTaskWindows (see page 14344)	This is function Forms.EnableTaskWindows.
	ForegroundTask (see page 14344)	Indicates whether the current thread is running in the foreground.
	FreeObjectInstance (see page 14345)	This is function Forms.FreeObjectInstance.
	GetApplicationMainFormOnTaskBar (see page 14345)	This is function Forms.GetApplicationMainFormOnTaskBar.
	GetCustomFormGlassFrame (see page 14345)	This is function Forms.GetCustomFormGlassFrame.
	GetParentForm (see page 14346)	Returns the form or property page that contains a specified control.
	InitCtl3D (see page 14346)	This is function Forms.InitCtl3D.
	IsAccel (see page 14346)	Indicates whether a particular character is an accelerator character (or hot key) within a given menu caption or other text string.
	KeyboardStateToShiftState (see page 14347)	Convert array of virtual keys to TShiftState.
	KeyboardStateToShiftState (see page 14347)	Convert array of virtual keys to TShiftState.
	KeyDataToShiftState (see page 14348)	Convert keyboard message data to TShiftState.
	KeysToShiftState (see page 14348)	Convert mouse event parameter to TShiftState.
	MakeObjectInstance (see page 14349)	This is function Forms.MakeObjectInstance.
	RestoreFocusState (see page 14349)	This is function Forms.RestoreFocusState.
	SaveFocusState (see page 14349)	This is function Forms.SaveFocusState.
	SetApplicationMainFormOnTaskBar (see page 14350)	This is function Forms.SetApplicationMainFormOnTaskBar.
	SetAutoSubClass (see page 14350)	This is function Forms.SetAutoSubClass.
	SetCustomFormGlassFrame (see page 14350)	This is function Forms.SetCustomFormGlassFrame.
	Subclass3DDlg (see page 14351)	This is function Forms.Subclass3DDlg.
	Subclass3DWnd (see page 14351)	This is function Forms.Subclass3DWnd.
	ValidParentForm (see page 14351)	Returns the form or property page that contains a specified control.

72.3.1 Forms.AllocateHWnd Function

Creates a window that implements a specified window procedure.

Pascal

```
function AllocateHWnd(Method: TWndMethod): HWND; deprecated;
```

C++

```
HWND AllocateHWnd(TWndMethod Method);
```

File

Forms

Description

Call AllocateHWnd to create a window that is not associated with a windowed control. Typically, this method is used to create non-visual windows that respond to messages but that do not appear in the user interface. For example, the TTimer component

uses this method to create a window that responds to timer messages from Windows.

The **Method** parameter specifies the window procedure that the generated window uses to respond to messages.

AllocateHWnd returns the handle of the newly created window.

Note: Use the DeallocateHWnd (see page 14343) procedure to free windows that are created using AllocateHWnd.

See Also

DeallocateHWnd (see page 14343)

TTimer

72.3.2 Forms.DeallocateHWnd Function

Frees a window that was created using AllocateHWnd (see page 14342).

Pascal

```
procedure DeallocateHWnd(Wnd: HWND); deprecated;
```

C++

```
DeallocateHWnd(HWND Wnd);
```

File

Forms

Description

Call DeallocateHWnd to free a window that was created using the AllocateHWnd (see page 14342) function. Windows must be freed to release the system resources that they consume.

The **Wnd** parameter is the window handle of the window to be freed. This value is the same as the handle returned by the call to AllocateHWnd (see page 14342).

Warning: Do not use DeallocateHWnd on the handle of a windowed control. Windowed controls free their handles using the protected DestroyHandle method, which ensures that the control is left in a consistent state.

See Also

AllocateHWnd (see page 14342)

72.3.3 Forms.DisableTaskWindows Function

Pascal

```
function DisableTaskWindows(ActiveWindow: HWnd): Pointer;
```

C++

```
void * DisableTaskWindows(HWnd ActiveWindow);
```

File

Forms

Description

This is function Forms.DisableTaskWindows.

72.3.4 Forms.DoneCtl3D Function

Pascal

```
procedure DoneCtl3D; deprecated;
```

C++

```
DoneCtl3D();
```

File

Forms

Description

This is function Forms.DoneCtl3D.

72.3.5 Forms.EnableTaskWindows Function

Pascal

```
procedure EnableTaskWindows(WindowList: Pointer);
```

C++

```
EnableTaskWindows(void * WindowList);
```

File

Forms

Description

This is function Forms.EnableTaskWindows.

72.3.6 Forms.ForegroundTask Function

Indicates whether the current thread is running in the foreground.

Pascal

```
function ForegroundTask: Boolean;
```

C++

```
Boolean ForegroundTask();
```

File

Forms

Description

Call ForegroundTask to determine whether the current thread is executing in the Foreground. This is useful if you want to provide feedback to users only if they are currently interacting with a window on the same thread, but do not want to interrupt other processes.

ForegroundTask checks whether the window that currently has focus belongs to the thread that executes the function. It returns

true if the focused window belongs to the current thread, false otherwise.

72.3.7 Forms.FreeObjectInstance Function

Pascal

```
procedure FreeObjectInstance(ObjectInstance: Pointer); deprecated;
```

C++

```
FreeObjectInstance(void * ObjectInstance);
```

File

Forms

Description

This is function Forms.FreeObjectInstance.

72.3.8 Forms.GetApplicationMainFormOnTaskBar Function

Pascal

```
function GetApplicationMainFormOnTaskBar(const Application: TApplication): Boolean; deprecated;
```

C++

```
Boolean GetApplicationMainFormOnTaskBar(const TApplication Application);
```

File

Forms

Description

This is function Forms.GetApplicationMainFormOnTaskBar.

72.3.9 Forms.GetCustomFormGlassFrame Function

Pascal

```
function GetCustomFormGlassFrame(const CustomForm: TCustomForm): TGlassFrame; deprecated;
```

C++

```
TGlassFrame GetCustomFormGlassFrame(const TCustomForm CustomForm);
```

File

Forms

Description

This is function Forms.GetCustomFormGlassFrame.

72.3.10 Forms.GetParentForm Function

Returns the form or property page that contains a specified control.

Pascal

```
function GetParentForm(Control: TControl; TopForm: Boolean = True): TCustomForm;
```

C++

```
TCustomForm GetParentForm(TControl * Control, Boolean TopForm = True);
```

File

Forms

Description

Call GetParentForm to obtain the descendant of TCustomForm (see page 13849) that contains the control specified by the **Control** parameter. If the **TopForm** parameter if true, this function goes all the way up the parent chain for **Control** until it finds a Form with no parent. If **TopForm** if false, this function goes up the parent chain until it finds a Form that is a TCustomForm (see page 13849).

If the control is not contained in a form (for example, an ActiveX control embedded in another type of window), GetParentForm returns **nil** (Delphi) or **NULL** (C++).

See Also

TControl (see page 5724)

TCustomForm (see page 13849)

ValidParentForm (see page 14351)

72.3.11 Forms.InitCtl3D Function

Pascal

```
procedure InitCtl3D; deprecated;
```

C++

```
InitCtl3D();
```

File

Forms

Description

This is function Forms.InitCtl3D.

72.3.12 Forms.IsAccel Function

Indicates whether a particular character is an accelerator character (or hot key) within a given menu caption or other text string.

Pascal

```
function IsAccel(VK: Word; const Str: string): Boolean;
```

C++

```
Boolean IsAccel(Word VK, const AnsiString Str);
```

File

Forms

Description

Call IsAccel to determine whether the character represented by the VK (virtual key) parameter appears as an accelerator in the string specified by the Str parameter. IsAccel only looks for accelerators that are identified by a preceding ampersand (&). Such accelerators appear underlined in the string when displayed as a control's caption.

Use IsAccel when processing key events to determine whether a key matches an accelerator character in a control's caption.

Note: IsAccel does not recognize hot key combinations in menu items that are identified by the HotKey property.

See Also

TCustomLabel

TCustomStaticText

TMenuItem

72.3.13 Forms.KeyboardStateToShiftState Function

Convert array of virtual keys to TShiftState.

Pascal

```
function KeyboardStateToShiftState(const KeyboardState: TKeyboardState): TShiftState; overload;
function KeyboardStateToShiftState: TShiftState; overload;
```

C++

```
TShiftState KeyboardStateToShiftState(const TKeyboardState KeyboardState);
TShiftState KeyboardStateToShiftState();
```

File

Forms

Description

Convert the given array of virtual keys **KeyboardState** to a TShiftState type. This array can be provided by the Windows function GetKeyboardState.

See Also

KeyDataToShiftState (see page 14348)

KeysToShiftState (see page 14348)

TShiftState (see page 3302)

72.3.14 Forms.KeyDataToShiftState Function

Convert keyboard message data to TShiftState.

Pascal

```
function KeyDataToShiftState(KeyData: Longint): TShiftState;
```

C++

```
TShiftState KeyDataToShiftState(Longint KeyData);
```

File

Forms

Description

Convert the current keyboard state and given **KeyData** to a TShiftState type.

See Also

KeyboardStateToShiftState (see page 14347)

KeysToShiftState (see page 14348)

TShiftState (see page 3302)

72.3.15 Forms.KeysToShiftState Function

Convert mouse event parameter to TShiftState.

Pascal

```
function KeysToShiftState(Keys: Word): TShiftState;
```

C++

```
TShiftState KeysToShiftState(Word Keys);
```

File

Forms

Description

Convert the parameter **Keys** to a TShiftState type. **Keys** contains flags that indicate the status of the mouse buttons and the CTRL and SHIFT keys at the time of the mouse event

See Also

KeyDataToShiftState (see page 14348)

KeyboardStateToShiftState (see page 14347)

TShiftState (see page 3302)

72.3.16 Forms.MakeObjectInstance Function

Pascal

```
function MakeObjectInstance(Method: TWndMethod): Pointer; deprecated;
```

C++

```
void * MakeObjectInstance(TWndMethod Method);
```

File

Forms

Description

This is function Forms.MakeObjectInstance.

72.3.17 Forms.RestoreFocusState Function

Pascal

```
procedure RestoreFocusState(FocusState: TFocusState);
```

C++

```
RestoreFocusState(TFocusState FocusState);
```

File

Forms

Description

This is function Forms.RestoreFocusState.

72.3.18 Forms.SaveFocusState Function

Pascal

```
function SaveFocusState: TFocusState;
```

C++

```
TFocusState SaveFocusState();
```

File

Forms

Description

This is function Forms.SaveFocusState.

72.3.19 Forms.SetApplicationMainFormOnTaskBar Function

Pascal

```
procedure SetApplicationMainFormOnTaskBar(const Application: TApplication; Value: Boolean);
deprecated;
```

C++

```
SetApplicationMainFormOnTaskBar(const TApplication Application, Boolean Value);
```

File

Forms

Description

This is function Forms.SetApplicationMainFormOnTaskBar.

72.3.20 Forms.SetAutoSubClass Function

Pascal

```
procedure SetAutoSubClass(Enable: Boolean); deprecated;
```

C++

```
SetAutoSubClass(Boolean Enable);
```

File

Forms

Description

This is function Forms.SetAutoSubClass.

72.3.21 Forms.SetCustomFormGlassFrame Function

Pascal

```
procedure SetCustomFormGlassFrame(const CustomForm: TCustomForm; const GlassFrame:
TGlassFrame); deprecated;
```

C++

```
SetCustomFormGlassFrame(const TCustomForm CustomForm, const TGlassFrame GlassFrame);
```

File

Forms

Description

This is function Forms.SetCustomFormGlassFrame.

72.3.22 Forms.Subclass3DDlg Function

Pascal

```
procedure Subclass3DDlg(Wnd: HWnd; Flags: Word); deprecated;
```

C++

```
Subclass3DDlg(HWnd Wnd, Word Flags);
```

File

Forms

Description

This is function Forms.Subclass3DDlg.

72.3.23 Forms.Subclass3DWnd Function

Pascal

```
function Subclass3DWnd(Wnd: HWnd): Boolean; deprecated;
```

C++

```
Boolean Subclass3DWnd(HWnd Wnd);
```

File

Forms

Description

This is function Forms.Subclass3DWnd.

72.3.24 Forms.ValidParentForm Function

Returns the form or property page that contains a specified control.

Pascal

```
function ValidParentForm(Control: TControl; TopForm: Boolean = True): TCustomForm;
```

C++

```
TCustomForm ValidParentForm(TControl * Control, Boolean TopForm = True);
```

File

Forms

Description

Call ValidParentForm to obtain the descendant of TCustomForm (see page 13849) that contains the control specified by the **Control** parameter. If the **TopForm** parameter if true, this function goes all the way up the parent chain for **Control** until it finds a Form with no parent. If **TopForm** if false, this function goes up the parent chain until it finds a Form that is a TCustomForm (see page 13849).

If the control is not contained in a form (for example, an ActiveX control embedded in another type of window), ValidParentForm raises an EInvalidOperation exception.

See Also

GetParentForm (see page 14346)

TControl (see page 5724)

TCustomForm (see page 13849)

72.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TActiveFormBorderStyle (see page 14353)	Specifies the style of an active form's border.
	TBorderIcon (see page 14354)	TBorderIcons (see page 14370) and TBorderIcons (see page 14370) specify the buttons that appear on the title bar of a form window.
	TCloseAction (see page 14355)	TCloseAction describes how a form should respond when it is closed.
	TDefaultMonitor (see page 14357)	Specifies the monitor on which the form appears.
	TFormBorderStyle (see page 14357)	TFormBorderStyle is the type of the BorderStyle property for windows-only forms and TBorderStyle (see page 14371) is the type of the BorderStyle property for other controls in windows-only applications.
	TFormStyle (see page 14358)	TFormStyle describes form style.
	TMonitorDefaultTo (see page 14361)	TMonitorDefautlTo specifies which monitor to select when there is no match to the specified criteria.
	TPopupMode (see page 14362)	TPopupMode enumerates the possible values of a Form's PopupMode property.
	TPosition (see page 14363)	TPosition describes the positioning of a form.
	TPrintScale (see page 14364)	Represents the proportions of a printed form.
	TScrollBarKind (see page 14365)	TScrollBarKind indicates the orientation of a scrollbar.
	TScrollBarStyle (see page 14366)	TScrollBarStyle specifies a scroll bar style.
	TShowAction (see page 14366)	This is record Forms.TShowAction.
	TTileMode (see page 14367)	Represents how MDI child forms are arranged when the Tile method is called.
	TTimerMode (see page 14367)	This is record Forms.TTimerMode.
	TWindowState (see page 14368)	TWindowState indicates whether a form is maximized, minimized, or normally sized.

Records

	Name	Description
	TCMHintShow (see page 14355)	This is record Forms.TCMHintShow.
	TCMHintShowPause (see page 14356)	This is record Forms.TCMHintShowPause.
	TCursorRec (see page 14356)	This is record Forms.TCursorRec.
	THintInfo (see page 14359)	THintInfo is used to define the appearance and behavior of the Help window.
	TPopupForm (see page 14362)	This is record Forms.TPopupForm.
	TPopupWnd (see page 14363)	This is record Forms.TPopupWnd.

72.4.1 Forms.TActiveFormBorderStyle Enumeration

Specifies the style of an active form's border.

Pascal

```
TActiveFormBorderStyle = (
  afbNone,
  afbSingle,
  afbSunken,
  afbRaised
);
```

C++

```
enum TActiveFormBorderStyle {
```

```
  afbNone,
  afbSingle,
  afbSunken,
  afbRaised
};
```

File

Forms

Description

TActiveFormBorderStyle is an enumeration of possible active form border styles, and may have on of the following are possible values :

Value	Meaning
afbNone	The form has no border.
afbSingle	The form has a regular single-line border.
afbSunken	The form has a 3D look—that is, a border with a sunken edge.
afbRaised	The form has a border with a raised edge.

72.4.2 Forms.TBorderIcon Enumeration

TBorderIcons (see page 14370) and TBorderIcons (see page 14370) specify the buttons that appear on the title bar of a form window.

Pascal

```
TBorderIcon = (
  biSystemMenu,
  biMinimize,
  biMaximize,
  biHelp
);
```

C++

```
enum TBorderIcon {
  biSystemMenu,
  biMinimize,
  biMaximize,
  biHelp
};
```

File

Forms

Description

TBorderIcons (see page 14370) can include any of the following TBorderIcon values:

Value	Meaning
biSystemMenu	The form has a Control menu (also known as a System (see page 22079) menu).
biMinimize	The form has a Minimize button
biMinimize	The form has a Maximize button

biHelp	If BorderStyle is bsDialog or biMinimize and biMaximize are excluded, a question mark appears in the form's title bar and when clicked, the cursor changes to crHelp; otherwise, no question mark appears.

72.4.3 Forms.TCloseAction Enumeration

TCloseAction describes how a form should respond when it is closed.

Pascal

```
TCloseAction = (
  caNone,
  caHide,
  caFree,
  caMinimize
);
```

C++

```
enum TCloseAction {
  caNone,
  caHide,
  caFree,
  caMinimize
};
```

File

Forms

Description

TCloseAction consists of the following values:

Value	Meaning
caNone	The form is not allowed to close, so nothing happens.
caHide	The form is not closed, but just hidden. Your application can still access a hidden form.
caFree	The form is closed and all allocated memory for the form is freed.
caMinimize	The form is minimized, rather than closed.

72.4.4 Forms.TCMHintShow Record

Pascal

```
TCMHintShow = record
  Msg: Cardinal;
  Reserved: Integer;
  HintInfo: PHintInfo;
  Result: Integer;
end;
```

C++

```
struct TCMHintShow {
  unsigned Msg;
  int Reserved;
```

```
    PHintInfo HintInfo;
    int Result;
  };
```

File

Forms

Description

This is record Forms.TCMHintShow.

72.4.5 Forms.TCMHintShowPause Record

Pascal

```
  TCMHintShowPause = record
    Msg: Cardinal;
    WasActive: Integer;
    Pause: PInteger;
    Result: Integer;
  end;
```

C++

```
  struct TCMHintShowPause {
    unsigned Msg;
    int WasActive;
    PInteger Pause;
    int Result;
  };
```

File

Forms

Description

This is record Forms.TCMHintShowPause.

72.4.6 Forms.TCursorRec Record

Pascal

```
  TCursorRec = record
    Next: PCursorRec;
    Index: Integer;
    Handle: HCURSOR;
  end;
```

C++

```
  struct TCursorRec {
    PCursorRec Next;
    int Index;
    HCURSOR Handle;
  };
```

File

Forms

Description

This is record Forms.TCursorRec.

72.4.7 Forms.TDefaultMonitor Enumeration

Specifies the monitor on which the form appears.

Pascal

```
TDefaultMonitor = (
  dmDesktop,
  dmPrimary,
  dmMainForm,
  dmActiveForm
);
```

C++

```
enum TDefaultMonitor {
  dmDesktop,
  dmPrimary,
  dmMainForm,
  dmActiveForm
};
```

File

Forms

Description

Use TDefaultMonitor to associate a form with a particular monitor in a multi-monitor application. The following table lists the possible values:

Value	Meaning
dmDesktop	No attempt is made to position the form on a specific monitor.
dmPrimary	The form is positioned on the first monitor listed in the global screen object's Monitors property.
dmMainForm	The form appears on the same monitor as the application's main form.
dmActiveForm	The form appears on the same monitor as the currently active form.

72.4.8 Forms.TFormBorderStyle Enumeration

TFormBorderStyle is the type of the BorderStyle property for windows-only forms and TBorderStyle (see page 14371) is the type of the BorderStyle property for other controls in windows-only applications.

Pascal

```
TFormBorderStyle = (
  bsNone,
  bsSingle,
  bsSizeable,
  bsDialog,
  bsToolWindow,
  bsSizeToolWin
```

```
);
```

C++

```
enum TFormBorderStyle {
  bsNone,
  bsSingle,
  bsSizeable,
  bsDialog,
  bsToolWindow,
  bsSizeToolWin
};
```

File

Forms

Description

These are the possible values of the TFormBorderStyle type:

Value	Meaning
bsNone	No visible border. (not resizable)
bsSingle	Single-line border. (not resizable)
bsDialog	Standard dialog box border. (not resizable)
bsSizeable	Standard resizable border.
bsToolWindow	Like bsSingle but with a smaller caption.
bsSizeToolWin	Like bsSizeable but with a smaller caption.

Only the bsNone and bsSingle values are possible values of TBorderStyle (see page 14371).

72.4.9 Forms.TFormStyle Enumeration

TFormStyle describes form style.

Pascal

```
TFormStyle = (
  fsNormal,
  fsMDIChild,
  fsMDIForm,
  fsStayOnTop
);
```

C++

```
enum TFormStyle {
  fsNormal,
  fsMDIChild,
  fsMDIForm,
  fsStayOnTop
};
```

File

Forms

Description

TFormStyle consists of the following values.

Value	Meaning
fsNormal	The form is neither an MDI parent window nor an MDI child window.
fsMDIChild	The form is an MDI child window.
fsMDIForm	The form is an MDI parent window.
fsStayOnTop	This form remains on top of the desktop and of other forms in the project, except any others that also have FormStyle set to fsStayOnTop. If one fsStayOnTop form launches another, neither form will consistently remain on top.

See Also

TCustomForm (see page 13849)

72.4.10 Forms.THintInfo Record

THintInfo is used to define the appearance and behavior of the Help window.

Pascal

```
THintInfo = record
  HintControl: TControl;
  HintWindowClass: THintWindowClass;
  HintPos: TPoint;
  HintMaxWidth: Integer;
  HintColor: TColor;
  CursorRect: TRect;
  CursorPos: TPoint;
  ReshowTimeout: Integer;
  HideTimeout: Integer;
  HintStr: string;
  HintData: Pointer;
end;
```

C++

```
struct THintInfo {
  TControl * HintControl;
  THintWindowClass HintWindowClass;
  TPoint HintPos;
  int HintMaxWidth;
  TColor HintColor;
  TRect CursorRect;
  TPoint CursorPos;
  int ReshowTimeout;
  int HideTimeout;
  AnsiString HintStr;
  void * HintData;
};
```

File

Forms

Description

The THintInfo type is used to define the appearance and behavior of the Help window in a TShowHintEvent (see page 14381) type OnShowHint event handler. The following table describes the meaning of its fields:

Field	Meaning
HintControl	The name of the control for which hint processing is occurring.
HintWindowClass	The class of the hint-window control. The default is THintWindow, but you can specify any class derived from THintWindow. Use this field if you want to substitute a custom hint window for THintWindow.
HintPos	The default position in screen coordinates of the top-left corner of the hint window. Change where the window appears by changing this value.
HintMaxWidth	The maximum width of the hint window before word wrapping begins. By default, the value is the width of the screen (the Width property of the global Screen (see page 14385) variable).
HintColor	The background color of the Hint window.
CursorRect	The rectangle the user's mouse pointer must be in for the hint window to appear. The default value for CursorRect is the client rectangle of the control. Change this value so that a single control can be divided into several hint regions. When the user moves the mouse pointer outside the rectangle, the hint window disappears.
CursorPos	The location of the mouse pointer within the control.
ReshowTimeout	How long the hint system should wait before asking about the hint status again. By default, this field is zero, indicating that the hint status will not change. Setting it to a non-zero value will cause the hint to act, after the requested milliseconds have elapsed, as if the user moved the mouse outside the hint rectangle and back in. This can be used to either put off hint processing for a period of time, or to allow the hint to be updated periodically.
HideTimeout	The number of milliseconds to show the hint. By default, it is set to the value of the Application (see page 14383) variable's HintHidePause property.
HintStr	The string to be displayed in the hint window. This allows an OnHint event handler to modify the contents of a hint before it is displayed. By default, it contains the value returned by the GetShortHint function when passed the value of the Application (see page 14383) variable's Hint property.
HintData	Additional data to be passed to the hint-window control. Use this field in conjunction with HintWindowClass.

C++ Examples:

```
/*
This example uses three speed buttons on a panel. The code
changes the color, width, and position of the text in the
Help Hint for the third speed button. Set the ShowHint
property of each speed button to true and give a value to
the Hint property of each.  You must declare the DoShow
method in the type declaration of the form. Once it is
declared, write the code for the DoShow method in the
implementation part of the unit. Finally, in the OnCreate
event handler for the form, assign the method to the
OnShowHint event of the application.
*/
void __fastcall TForm1::DoShowHint(AnsiString &HintStr, bool &CanShow, THintInfo &HintInfo)
{
  if (HintInfo.HintControl == SpeedButton3)
  {
    HintInfo.HintColor = clAqua;// Changes only for this hint
    HintInfo.HintMaxWidth = 120; // Hint text word wraps if width is greater than 120
    HintInfo.HintPos.x += SpeedButton3->Width; // Move hint to right edge
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->ShowHint = true;
  Application->OnShowHint = DoShowHint;
}
```

Delphi Examples:

```
{
This example uses three speed buttons on a panel. The code
changes the color, width, and position of the text in the
Help Hint for the third speed button. Set the ShowHint
property of each speed button to true and give a value to
the Hint property of each.  You must declare the DoShow
method in the type declaration of the form. Once it is
declared, write the code for the DoShow method in the
implementation part of the unit. Finally, in the OnCreate
event handler for the form, assign the method to the
OnShowHint event of the application.
}

procedure TForm1.DoShowHint(var HintStr: String; var CanShow: Boolean; var HintInfo:
THintInfo);
begin
  if HintInfo.HintControl = SpeedButton3 then
  begin
    with HintInfo do
    begin
      HintColor := clAqua;{ Changes only for this hint }
      HintMaxWidth := 120;{Hint text word wraps if width is greater than 120 }
      Inc(HintPos.X, SpeedButton3.Width); { Move hint to right edge }
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.ShowHint := True;
  Application.OnShowHint := DoShowHint;
end;
```

See Also

THintWindow

OnShowHint

72.4.11 Forms.TMonitorDefaultTo Enumeration

TMonitorDefautlTo specifies which monitor to select when there is no match to the specified criteria.

Pascal

```
TMonitorDefaultTo = (
  mdNearest,
  mdNull,
  mdPrimary
);
```

C++

```
enum TMonitorDefaultTo {
  mdNearest,
  mdNull,
  mdPrimary
};
```

File

Forms

Description

TMonitorDefaultTo indicates which monitor should be used when the selection criteria do not match any of the monitors in the system. It can have any of the following values:

Value	Meaning
mdNearest	Choose the monitor that comes closest to matching the criteria (the monitor closest to a point, on which most of a rectangle lies, etc).
mdNull	Do not select any monitor if the criteria can't be met. Use nil (Delphi) or NULL (C++) to indicate that no monitor could be found.
mdPrimary	Choose the primary monitor if no monitor matches the criteria.

72.4.12 Forms.TPopupForm Record

Pascal

```
TPopupForm = record
  FormID: Integer;
  Form: TCustomForm;
  WasPopup: Boolean;
end;
```

C++

```
struct TPopupForm {
  int FormID;
  TCustomForm Form;
  Boolean WasPopup;
};
```

File

Forms

Description

This is record Forms.TPopupForm.

72.4.13 Forms.TPopupMode Enumeration

TPopupMode enumerates the possible values of a Form's PopupMode property.

Pascal

```
TPopupMode = (
  pmNone,
  pmAuto,
  pmExplicit
);
```

C++

```
enum TPopupMode {
  pmNone,
  pmAuto,
  pmExplicit
};
```

File

Forms

Description

The PopupMode property controls how the top-level form behaves with respect to Window's WS_POPUP style. The TPopupMode can have one of these values :

Value	Meaning
pmNone	Set the PopupMode property to pmNone if you want the popup window to have the pre-Delphi 8 behavior.
pmAuto	Popup handling is automatic.
pmExplicit	Set the PopupMode property to pmExplicit for non-modal design windows such as tool palettes and other floating tool windows. This causes the windows to always remain on top of the main form.

See Also

PopupMode (see page 14054)

72.4.14 Forms.TPopupWnd Record

Pascal

```
TPopupWnd = record
  ID: Integer;
  ControlWnd: HWND;
end;
```

C++

```
struct TPopupWnd {
  int ID;
  HWND ControlWnd;
};
```

File

Forms

Description

This is record Forms.TPopupWnd.

72.4.15 Forms.TPosition Enumeration

TPosition describes the positioning of a form.

Pascal

```
TPosition = (
  poDesigned,
  poDefault,
  poDefaultPosOnly,
  poDefaultSizeOnly,
  poScreenCenter,
  poDesktopCenter,
  poMainFormCenter,
```

```
  poOwnerFormCenter
);
```

C++

```
enum TPosition {
  poDesigned,
  poDefault,
  poDefaultPosOnly,
  poDefaultSizeOnly,
  poScreenCenter,
  poDesktopCenter,
  poMainFormCenter,
  poOwnerFormCenter
};
```

File

Forms

Description

TPosition consists of the following values:

Value	Meaning
poDesigned	The form appears positioned on the screen and with the same height and width as it had at design time.
poDefault	The form appears in a position on the screen and with a height and width determined by the operating system.
poDefaultPosOnly	The form displays with the size you created it at design time, but the operating system chooses its position on the screen.
poDefaultSizeOnly	The form appears in the position you left it at design time, but the operating system chooses its size.
poScreenCenter	The form remains the size you left it at design time, but is positioned in the center of the screen. Note that in multi-monitor applications, this may result in a form that does not fall entirely on one monitor, because CLX does not include multi-monitor support.
poDesktopCenter	The form remains the size you left it at design time, but is positioned in the center of the screen.
poMainFormCenter	The form remains the size you left it at design time, but is positioned in the center of the application's main form. This position should only be used with secondary forms. If set for a main form, it acts like poScreenCenter.
poOwnerFormCenter	The form remains the size you left it at design time, but is positioned in the center of the form specified by the Owner property. If the Owner property does not specify a form, this position acts like poMainFormCenter.

See Also

TCustomForm (see page 13849)

72.4.16 Forms.TPrintScale Enumeration

Represents the proportions of a printed form.

Pascal

```
TPrintScale = (
  poNone,
  poProportional,
  poPrintToFit
);
```

C++

```
enum TPrintScale {
  poNone,
  poProportional,
  poPrintToFit
};
```

File

Forms

Description

Use TPrintScale to get or set the proportions of the printed form. TPrintScale can have one of the following values:

Value	Meaning
poNone	No special scaling occurs; therefore, the printed form and how the form appears onscreen may appear squished or stretched.
poProportional	The form is printed so that the printed image is approximately the same visible size as on the screen (WYSIWYG). The form image is scaled so that the printed image is approximately the same visible size as on the screen.
poPrintToFit	The form is printed using the same screen proportions, but in a size that just fits the printed page.

72.4.17 Forms.TScrollBarKind Enumeration

TScrollBarKind indicates the orientation of a scrollbar.

Pascal

```
TScrollBarKind = (
  sbHorizontal,
  sbVertical
);
```

C++

```
enum TScrollBarKind {
  sbHorizontal,
  sbVertical
};
```

File

Forms

Description

TScrollBarKind indicates whether a scrollbar is horizontal or vertical. It includes the following values:

Value	Meaning
sbHorizontal	Horizontal scrollbar.
sbVertical	Vertical scrollbar.

72.4.18 Forms.TScrollBarStyle Enumeration

TScrollBarStyle specifies a scroll bar style.

Pascal

```
TScrollBarStyle = (
  ssRegular,
  ssFlat,
  ssHotTrack
);
```

C++

```
enum TScrollBarStyle {
  ssRegular,
  ssFlat,
  ssHotTrack
};
```

File

Forms

Description

TScrollBarStyle can have one of the following values:

Value	Meaning
ssRegular	The scroll bar is normal.
ssFlat	The scroll bar is flat.
ssHotTrack	The scroll bar uses hot tracking.

72.4.19 Forms.TShowAction Enumeration

Pascal

```
TShowAction = (
  saIgnore,
  saRestore,
  saMinimize,
  saMaximize
);
```

C++

```
enum TShowAction {
  saIgnore,
  saRestore,
  saMinimize,
  saMaximize
};
```

File

Forms

Description

This is record Forms.TShowAction.

72.4.20 Forms.TTileMode Enumeration

Represents how MDI child forms are arranged when the Tile method is called.

Pascal

```
TTileMode = (
  tbHorizontal,
  tbVertical
);
```

C++

```
enum TTileMode {
  tbHorizontal,
  tbVertical
};
```

File

Forms

Description

TTileMode can have one of the following values:

Value	Meaning
tbHorizontal	Each form stretches across the width of the parent form.
tbVertical	Each form stretches along the height of the parent form.

Note: TTileMode is meaningful only if the form is an MDI child (that is, if the form's FormStyle property is set to fsMDIChild).

72.4.21 Forms.TTimerMode Enumeration

Pascal

```
TTimerMode = (
  tmShow,
  tmHide
);
```

C++

```
enum TTimerMode {
  tmShow,
  tmHide
};
```

File

Forms

Description

This is record Forms.TTimerMode.

72.4.22 Forms.TWindowState Enumeration

TWindowState indicates whether a form is maximized, minimized, or normally sized.

Pascal

```
TWindowState = (
  wsNormal,
  wsMinimized,
  wsMaximized
);
```

C++

```
enum TWindowState {
  wsNormal,
  wsMinimized,
  wsMaximized
};
```

File

Forms

Description

TWindowState describes the state of a form window. The following table lists the possible values:

Value	Meaning
wsNormal	The form is in its normal state (that is, neither minimized nor maximized).
wsMinimized	The form is minimized.
wsMaximized	The form is maximized.

See Also

TCustomForm (see page 13849)

72.5 Types

The following table lists types in this documentation.

Types

Name	Description
PCursorRec (see page 14369)	This is type Forms.PCursorRec.
PHintInfo (see page 14370)	PHintInfo declares a pointer to a Help Window appearance record.
TBorderIcons (see page 14370)	TBorderIcons specifies the buttons that appear on the title bar of a form window.
TBorderStyle (see page 14371)	TBorderStyle is the type of the BorderStyle property for framed controls.
TCloseEvent (see page 14371)	TCloseEvent is the type of a form's OnClose event handle.
TCloseQueryEvent (see page 14372)	TCloseQueryEvent is used for event handlers that are called when a window is about to close.
TCustomFormClass (see page 14372)	TCustomFormClass defines the metaclass for TCustomForm (see page 13849).
TCustomFrameClass (see page 14373)	TCustomFrameClass defines the metaclass for TCustomFrame (see page 13978).
TExceptionEvent (see page 14373)	TExceptionEvent is the type of event handlers that respond when the application traps an unhandled exception.
TFocusState (see page 14374)	This is type Forms.TFocusState.
TFormClass (see page 14374)	TFormClass defines the metaclass for TForm (see page 13992).
TFormState (see page 14374)	TFormState describes the possible state of a form.
TGetHandleEvent (see page 14375)	This is type Forms.TGetHandleEvent.
THelpEvent (see page 14375)	THelpEvent is the type of an OnHelp event handler.
TIdleEvent (see page 14376)	TIdleEvent is the type of event handler that perform actions when the application is idle.
TMessageEvent (see page 14376)	TMessageEvent is the type of an OnMessage event handler.
TPopupFormArray (see page 14377)	This is type Forms.TPopupFormArray.
TPopupWndArray (see page 14377)	This is type Forms.TPopupWndArray.
TScrollBarInc (see page 14377)	TScrollBarInc represents a number of logical units on a TControlScrollBar (see page 13744).
TSetLayeredWindowAttributes (see page 14378)	This is type Forms.TSetLayeredWindowAttributes.
TSettingChangeEvent (see page 14378)	TSettingChangeEvent is the type of an OnSettingChange event handler.
TShortCutEvent (see page 14381)	TShortCutEvent is the type of an OnShortCut event handler.
TShowHintEvent (see page 14381)	TShowHintEvent is the type of the OnShowHint event handler.
TWindowHook (see page 14382)	TWindowHook is the type for a dialog procedure.

72.5.1 Forms.PCursorRec Type

Pascal

```
PCursorRec = ^TCursorRec;
```

C++

```
^TCursorRec PCursorRec;
```

File

Forms

Description

This is type Forms.PCursorRec.

72.5.2 Forms.PHintInfo Type

PHintInfo declares a pointer to a Help Window appearance record.

Pascal

```
PHintInfo = ^THintInfo;
```

C++

```
^THintInfo PHintInfo;
```

File

Forms

Description

The PHintInfo type declares a pointer to a THintInfo (see page 14359) variable. The THintInfo (see page 14359) type is used to define the appearance and behavior of the Help window in a TShowHintEvent (see page 14381) type OnShowHint event handler.

See Also

OnShowHint (see page 13742)

TShowHintEvent (see page 14381)

THintInfo (see page 14359)

72.5.3 Forms.TBorderIcons Type

TBorderIcons specifies the buttons that appear on the title bar of a form window.

Pascal

```
TBorderIcons = set of TBorderIcon;
```

C++

```
TBorderIcon TBorderIcons;
```

File

Forms

Description

TBorderIcons type declares a pointer to a TBorderIcon (see page 14354) variable. TBorderIcons can include any of the following TBorderIcon (see page 14354) values:

Value	Meaning
biSystemMenu	The form has a Control menu (also known as a System (see page 22079) menu).
biMinimize	The form has a Minimize button
biMinimize	The form has a Maximize button
biHelp	If BorderStyle is bsDialog or biMinimize and biMaximize are excluded, a question mark appears in the form's title bar and when clicked, the cursor changes to crHelp; otherwise, no question mark appears.

See Also

TBorderIcon (see page 14354)

72.5.4 Forms.TBorderStyle Type

TBorderStyle is the type of the BorderStyle property for framed controls.

Pascal

```
TBorderStyle = bsNone..bsSingle;
```

C++

```
bsNone..bsSingle TBorderStyle;
```

File

Forms

Description

These are the possible values of the TBorderStyle type:

Value	Meaning
bsNone	No visible border.
bsSingle	Single-line border.

72.5.5 Forms.TCloseEvent Type

TCloseEvent is the type of a form's OnClose event handle.

Pascal

```
TCloseEvent = procedure (Sender: TObject; var Action: TCloseAction) of object;
```

C++

```
(Sender: TObject; var Action: TCloseAction) ( TCloseEvent)();
```

File

Forms

Description

The TCloseEvent type points to a method that handles the closing of a form. The **Sender** parameter is the form that the user tried to close. The **Action** parameter returns the action you want the form to take when it closes.

Setting **Action** controls how the form responds to the attempt to close it as shown below :

Value	Meaning
caNone	The form is not allowed to close, so nothing happens.
caHide	The form is not closed, but just hidden. Your application can still access a hidden form.
caFree	The form is closed and all allocated memory for the form is freed.

caMinimize	The form is minimized, rather than closed. This is the default action for MDI child forms.

See Also

OnClose (see page 14072)

72.5.6 Forms.TCloseQueryEvent Type

TCloseQueryEvent is used for event handlers that are called when a window is about to close.

Pascal

```
TCloseQueryEvent = procedure (Sender: TObject; var CanClose: Boolean) of object;
```

C++

```
(Sender: TObject; var CanClose: Boolean) ( TCloseQueryEvent)();
```

File

Forms

Description

The TCloseQueryEvent type is the type for event handlers that allow an application to block the closing of a window (form or dialog).

Sender is the object whose event handler is called. This is the form that is about to close or the dialog whose window is about to close.

CanClose is set by the event handler to indicate whether the form or dialog can close (true) or whether the user should be blocked from closing it (false).

See Also

OnClose (see page 14072)

TCloseQueryEvent

72.5.7 Forms.TCustomFormClass Type

TCustomFormClass defines the metaclass for TCustomForm (see page 13849).

Pascal

```
TCustomFormClass = class of TCustomForm;
```

C++

```
TCustomForm TCustomFormClass;
```

File

Forms

Description

TCustomFormClass is the metaclass for TCustomForm (see page 13849). Its value is the class reference for TCustomForm (see page 13849) or for one of its descendants.

See Also

TCustomForm (see page 13849)

72.5.8 Forms.TCustomFrameClass Type

TCustomFrameClass defines the metaclass for TCustomFrame (see page 13978).

Pascal

```
TCustomFrameClass = class of TCustomFrame;
```

C++

```
TCustomFrame TCustomFrameClass;
```

File

Forms

Description

TCustomFrameClass is the metaclass for TCustomFrame (see page 13978). Its value is the class reference for TCustomFrame (see page 13978) or for one of its descendants.

See Also

TCustomFrame (see page 13978)

72.5.9 Forms.TExceptionEvent Type

TExceptionEvent is the type of event handlers that respond when the application traps an unhandled exception.

Pascal

```
TExceptionEvent = procedure (Sender: TObject; E: Exception) of object;
```

C++

```
(Sender: TObject; E: Exception) ( TExceptionEvent)();
```

File

Forms

Description

The TExceptionEvent type is the type of the OnException event handler.

Sender is the object that receives the event notification. Typically, this is the application object.

E is the exception object that represents the exception.

See Also

OnException (see page 13728)

72.5.10 Forms.TFocusState Type

Pascal

```
TFocusState = type Pointer;
```

C++

```
type Pointer TFocusState;
```

File

Forms

Description

This is type Forms.TFocusState.

72.5.11 Forms.TFormClass Type

TFormClass defines the metaclass for TForm (see page 13992).

Pascal

```
TFormClass = class of TForm;
```

C++

```
TForm * TFormClass;
```

File

Forms

Description

TFormClass is the metaclass for TForm (see page 13992). Its value is the class reference for TForm (see page 13992) or for one of its descendants.

See Also

TForm (see page 13992)

72.5.12 Forms.TFormState Type

TFormState describes the possible state of a form.

Pascal

```
TFormState = set of (fsCreating, fsVisible, fsShowing, fsModal, fsCreatedMDIChild,
fsActivated);
```

C++

```
(fsCreating, fsVisible, fsShowing, fsModal, fsCreatedMDIChild, fsActivated) TFormState;
```

File

Forms

Description

The following table lists the values that can be included in a form's state:

Value	Meaning
fsCreating	The form's constructor is currently executing.
fsVisible	The form's window is visible. This state is used to update the Visible property.
fsShowing	The form's WindowState property is changing. This state is used to prevent WindowState changes from interfering with a transition that is in progress.
fsModal	The form was created as a modal window.
fsActivated	The form has received focus or the application became active but has not yet called the Activate method to generate an OnActivate event.

72.5.13 Forms.TGetHandleEvent Type

Pascal

```
TGetHandleEvent = procedure (var Handle: HWND) of object;
```

C++

```
(var Handle: HWND) ( TGetHandleEvent)();
```

File

Forms

Description

This is type Forms.TGetHandleEvent.

72.5.14 Forms.THelpEvent Type

THelpEvent is the type of an OnHelp event handler.

Pascal

```
THelpEvent = function (Command: Word; Data: Longint; var CallHelp: Boolean): Boolean of object;
```

C++

```
(Command: Word; Data: Longint; var CallHelp: Boolean): Boolean ( THelpEvent)();
```

File

Forms

Description

THelpEvent includes the following parameters:

Command indicates the type of help command that was issued. It's meaning depends on the current help system. For example, when using WinHelp, Command is the command used by the WinHelp API. For details on possible commands, see the documentation provided by the developer of the application's help system. For example, if the application uses WinHelp, the possible commands are listed in the WinHelp section of the Win32 Developer's Reference.

Data provides additional data to describe the desired action by the help system. Its value depends on the value of the Command property. Typically, Data specifies a keyword or context ID. For details, see the documentation provided by the application's help system.

CallHelp is set by the event handler. If CallHelp is true when the event handler exits, the application should forward the help command to the help system. If the event handler handles the command, it sets CallHelp to false so that the application does not forward the help command to the help system.

72.5.15 Forms.TIdleEvent Type

TIdleEvent is the type of event handler that perform actions when the application is idle.

Pascal

```
TIdleEvent = procedure (Sender: TObject; var Done: Boolean) of object;
```

C++

```
(Sender: TObject; var Done: Boolean) ( TIdleEvent)();
```

File

Forms

Description

The TIdleEvent type is the type of the OnIdle event.

Sender is the object that receives the event notification. Typically, this is the application object.

Done indicates whether the event handler has finished its background processing.

If the event handler exits when **Done** is true, all action components are then given a chance to update themselves and, on Windows, the application yields control to Windows until a new message appears in the application's message queue. When **Done** is false, any action components in the application are not allowed to update themselves when the event handler exits and, on Windows, the application does not explicitly yield control to the operating system.

See Also

OnIdle (see page 13733)

72.5.16 Forms.TMessageEvent Type

TMessageEvent is the type of an OnMessage event handler.

Pascal

```
TMessageEvent = procedure (var Msg: TMsg; var Handled: Boolean) of object;
```

C++

```
(var Msg: TMsg; var Handled: Boolean) ( TMessageEvent)();
```

File

Forms

Description

TMessageEvent includes the following parameters:

Msg identifies the Windows message that triggered the event.

Handled indicates whether the event handler responded to the message. If the event handler sets Handled to true, the application assumes that the message has been completely handled and halts any subsequent processing of the message.

72.5.17 Forms.TPopupFormArray Type

Pascal

```
TPopupFormArray = array of TPopupForm;
```

C++

```
array of TPopupForm TPopupFormArray;
```

File

Forms

Description

This is type Forms.TPopupFormArray.

72.5.18 Forms.TPopupWndArray Type

Pascal

```
TPopupWndArray = array of TPopupWnd;
```

C++

```
array of TPopupWnd TPopupWndArray;
```

File

Forms

Description

This is type Forms.TPopupWndArray.

72.5.19 Forms.TScrollBarInc Type

TScrollBarInc represents a number of logical units on a TControlScrollBar (see page 13744).

Pascal

```
TScrollBarInc = 1..32767;
```

C++

```
1..32767 TScrollBarInc;
```

File

Forms

Description

TScrollBarInc has a value between 1 and 32767 (inclusive). It indicates the number of positions to scroll.

See Also

TControlScrollBar

72.5.20 Forms.TSetLayeredWindowAttributes Type

Pascal

```
TSetLayeredWindowAttributes = function (Hwnd: THandle; crKey: COLORREF; bAlpha: Byte; dwFlags:
DWORD): Boolean;
```

C++

```
(Hwnd: THandle; crKey: COLORREF; bAlpha: Byte; dwFlags: DWORD): Boolean (
TSetLayeredWindowAttributes)();
```

File

Forms

Description

This is type Forms.TSetLayeredWindowAttributes.

72.5.21 Forms.TSettingChangeEvent Type

TSettingChangeEvent is the type of an OnSettingChange event handler.

Pascal

```
TSettingChangeEvent = procedure (Sender: TObject; Flag: Integer; const Section: string; var
Result: Longint) of object;
```

C++

```
(Sender: TObject; Flag: Integer; const Section: string; var Result: Longint) (
TSettingChangeEvent)();
```

File

Forms

Description

TSettingChangeEvent has the following parameters:

Sender is the application object.

Section indicates the area that contains the system parameter that changed. Typically, this is a registry key or the name of a section in Win.ini. If the notification indicates a policy change, Section is the string 'Policy'. If the notification indicates a change in locale settings, Section is the string 'intl'.

Result is the value returned to Windows. It should be 0 if the event handler responded to the setting change, nonzero otherwise.

Flag indicates what setting changed. The following table lists the possible values:

Value	Meaning
0	A user policy was applied or a user locale setting changed.
1	A computer policy was applied.
SPI_SETDESKWALLPAPER	The bitmap for the desktop wallpaper changed.
SPI_SETCURSORS	The system cursors changed.
SPI_SETDESKPATTERN	The desktop pattern was reloaded from Win.ini.
SPI_SETFONTSMOOTING	The font smoothing feature was enabled or disabled.
SPI_SETWORKAREA	The size of the work area (the portion of the screen not obscured by the system taskbar or desktop toolbars) changed.
SPI_ICONHORIZONTALSPACING	The width, in pixels, of an icon cell changed. This affects the arrangement of icons in large icon view.
SPI_ICONVERTICALSPACING	The height, in pixels, of an icon cell changed. This affects the arrangement of icons in large icon view.
SPI_SETICONMETRICS	The metrics associated with system icons changed.
SPI_SETICONS	The system icons were reloaded.
SPI_SETICONTITLELOGFONT	The font for icon titles changed.
SPI_SETICONTITLEWRAP	Icon title-wrapping was turned on or off.
SPI_SETBEEP	The warning beeper was turned on or off.
SPI_SETDEFAULTINPUTLANG	The default input language for the system shell changed
SPI_SETDOUBLECLICKTIME	The double-click time for the mouse changed.
SPI_SETDOUBLECLICKHEIGHT	The height of the double-click rectangle changed.
SPI_SETDOUBLECLICKWIDTH	The width of the double-click rectangle changed.
SPI_SETKEYBOARDCUES	The setting for whether Menu access key letters appear always or only when the menu is activated from the keyboard changed.
SPI_SETKEYBOARDDELAY	The keyboard repeat-delay setting changed.
SPI_SETKEYBOARDPREF	The keyboard preference changed. This indicates whether the user wants applications to display hidden keyboard interfaces.
SPI_SETKEYBOARDSPEED	The keyboard repeat-speed setting changed.
SPI_SETLANGTOGGLE	The hotkey for switching input languages changed.
SPI_SETMOUSE	The mouse threshold values or acceleration changed.
SPI_SETMOUSEBUTTONSWRAP	The meaning of left and right mouse buttons changed.
SPI_SETMOUSEHOVERHEIGHT	The height of the rectangle in which the mouse must stay for mouse hover messages changed.
SPI_SETMOUSEHOVERTIME	The time the mouse must stay in the hover rectangle to generate mouse hover messages changed.
SPI_SETMOUSEHOVERWIDTH	The width of the rectangle in which the mouse must stay for mouse hover messages changed.
SPI_SETMOUSESPEED	The mouse speed changed.
SPI_SETMOUSETRAILS	Mouse trails were turned on or off.
SPI_SETSNAPTODEFBUTTON	The automatic snapping of the mouse to the default button was turned on or off.
SPI_SETWHEELSCROLLLINES	The number of lines to scroll when the mouse wheel rotates changed.

SPI_SETMENUDROPALIGNMENT	Popup menus changed between left- and right-alignment.
SPI_SETMENUFADE	Menu fade animation was turned on or off (Windows 2K)
SPI_SETMENUSHOWDELAY	The delay before displaying a shortcut menu when the mouse is over a submenu item changed.
SPI_SETLOWPOWERACTIVE	The low-power phase of screen saving was activated or deactivated.
SPI_SETLOWPOWERTIMEOUT	The time period before the low-power phase of screen saving is activated changed.
SPI_SETPOWEROFFACTIVE	The power-off phase of screen saving came on or off.
SPI_SETPOWEROFFTIMEOUT	The delay before activating the power-off phase of screen saving changed.
SPI_SETSCREENSAVEACTIVE	The screen saver was activated or deactivated.
SPI_SETSCREENSAVETIMEOUT	The delay before the screen saver appears changed.
SPI_SETCOMBOBOXANIMATION	The slide-open effect for combo boxes was turned on or off.
SPI_SETCURSORSHADOW	The cursor shadow was turned on or off (Windows 2K).
SPI_SETGRADIENTCAPTIONS	The gradient effect on window title bars was turned on or off.
SPI_SETHOTTRACKING	Hot-tracking was enabled or disabled.
SPI_SETLISTBOXSMOOTHSCROLLING	Smooth scrolling of list boxes was enabled or disabled.
SPI_SETMENUANIMATION	Menu animation was enabled or disabled.
SPI_SETMENUUNDERLINES	Same as SPI_SETKEYBOARDCUES.
SPI_SETSELECTIONFADE	The menu selection fade feature was enabled or disabled. (Windows 2K)
SPI_SETTOOLTIPANIMATION	ToolTip animation was enabled or disabled. (Windows 2K)
SPI_SETTOOLTIPFADE	ToolTip animation changed between a fade and a slide effect. (Windows 2K)
SPI_SETUIEFFECTS	UI effects were enabled or disabled. (Windows 2K)
SPI_SETACTIVEWINDOWTRACKING	Active window tracking (where the active window follows the mouse) was enabled or disabled.
SPI_SETACTIVEWNDTRKZORDER	Active window tracking changed whether the active window was brought to the top.
SPI_SETANIMATION	Parameters to the animation effects for user actions changed.
SPI_SETBORDER	The width of a window's sizing border changed.
SPI_SETCARETWIDTH	The width of the caret in edit controls changed (Windows 2K)
SPI_SETDRAGFULLWINDOWS	Dragging of full windows was turned on or off.
SPI_SETDRAGHEIGHT	The height of the rectangle used to detect drag operations changed.
SPI_SETDRAGWIDTH	The width of the rectangle used to detect drag operations changed.
SPI_SETFOREGROUNDFLASHCOUNT	The number of times a taskbar button flashes when rejecting a foreground switch request changed.
SPI_SETFOREGROUNDLOCKTIMEOUT	The time after user input during which applications can't force themselves into the foreground changed.
SPI_SETMINIMIZEDMETRICS	The metrics for minimized windows changed.
SPI_SETNONCLIENTMETRICS	The metrics for the nonclient area of nonminimized windows changed.
SPI_SETSHOWIMEUI	The ability of the IME window to be visible on a per-user basis changed.
SPI_SETPENWINDOWS	Pen windows was loaded or unloaded. (Windows 9x)
SPI_SETACCESSTIMEOUT	The time-out period for accessibility features changed.
SPI_SETFILTERKEYS	The parameters to the FilterKeys accessibility feature changed.

SPI_SETHIGHCONTRAST	The parameters to the HighContrast accessibility feature changed.
SPI_SETMOUSECLICKLOCK	The mouse ClickLock state was turned on or off. (Windows Me)
SPI_SETMOUSECLICKLOCKTIME	The time delay for locking the primary mouse button changed. (Windows Me)
SPI_SETMOUSEKEYS	The parameters for the MouseKeys accessibility feature changed.
SPI_SETMOUSESONAR	The mouse Sonar feature was turned on or off. (Windows Me)
SPI_SETMOUSEVANISH	The mouse Vanish feature was turned on or off. (Windows Me)
SPI_SETSCREENREADER	A screen reviewer utility (such as a speech synthesizer or Braille display utility) started or stopped running.
SPI_SETSERIALKEYS	The parameters to the SerialKeys accessibility feature changed. (Windows 9x)
SPI_SETSHOWSOUNDS	The Show Sounds accessibility feature toggled.
SPI_SETSOUNDSENTRY	The parameters to the SoundSentry accessibility feature changed.
SPI_SETSTICKYKEYS	The parameters to the StickyKeys feature changed.
SPI_SETTOGGLEKEYS	The parameters to the ToggleKeys feature changed.

72.5.22 Forms.TShortCutEvent Type

TShortCutEvent is the type of an OnShortCut event handler.

Pascal

```
TShortCutEvent = procedure (var Msg: TWMKey; var Handled: Boolean) of object;
```

C++

```
(var Msg: TWMKey; var Handled: Boolean) ( TShortCutEvent)();
```

File

Forms

Description

The TShortCutEvent type represents event handlers that respond to keystrokes before they enter the standard keystroke processing. These event handlers allow an application to bypass the standard keystroke processing for special shortcut keys.

The Msg parameter represents the Windows keystroke message. The virtual key code is given by the CharCode field, and the KeyData field contains a set of flags that indicate the repeat count, scan code, extended-key flag, context code, previous key-state flag and transition-state flag. See the Windows documentation for details on these fields.

The Handled parameter indicates whether the event handler processed the key. If Handled is set to true, then the application stops processing the keystroke. Otherwise (if Handled remains false) the application proceeds with normal keystroke processing.

72.5.23 Forms.TShowHintEvent Type

TShowHintEvent is the type of the OnShowHint event handler.

Pascal

```
TShowHintEvent = procedure (var HintStr: string; var CanShow: Boolean; var HintInfo:
THintInfo) of object;
```

C++

```
(var HintStr: string; var CanShow: Boolean; var HintInfo: THintInfo) ( TShowHintEvent)();
```

File

Forms

Description

TShowHintEvent has the following parameters:

HintStr sets the text of the Help Hint. You can change this string in the event handler to change the hint text that appears to the user.

CanShow indicates whether the Help Hint should display. If the event handler sets CanShow to true, the Help Hint displays. If it sets CanShow to false, the Help Hint does not appear.

HintInfo contains information about the appearance and behavior of the Help window. Change its fields to customize the way the Help Hint is displayed.

72.5.24 Forms.TWindowHook Type

TWindowHook is the type for a dialog procedure.

Pascal

```
TWindowHook = function (var Message: TMessage): Boolean of object;
```

C++

```
(var Message: TMessage): Boolean ( TWindowHook)();
```

File

Forms

Description

TWindowHook is the type for the window procedure of a non-VCL dialog box. Like most window procedures, TWindowHook takes a TMessage value as an argument.

Except in response to the WM_INITDIALOG message, the window hook should return true if it processes the message, and false if it does not. When the Msg field of the Message parameter is WM_INITDIALOG, the window hook should return false if it explicitly sets the focus to one of the controls in the dialog box. Otherwise, it should return true, in which case the system sets the focus to the first control in the dialog box that can be given the focus.

See Also

TApplication (see page 13660)

72.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
Application (see page 14383)	Represents application-level information.
Ctl3DBtnWndProc (see page 14383)	This is variable Forms.Ctl3DBtnWndProc.
Ctl3DCtlColorEx (see page 14384)	This is variable Forms.Ctl3DCtlColorEx.
Ctl3DDlgFramePaint (see page 14384)	This is variable Forms.Ctl3DDlgFramePaint.
HintWindowClass (see page 14384)	Specifies the class for the window used to display help hints.
Screen (see page 14385)	Represents a screen device.
SetLayeredWindowAttributes (see page 14385)	This is variable Forms.SetLayeredWindowAttributes.

72.6.1 Forms.Application Variable

Represents application-level information.

Pascal

```
Application: TApplication;
```

C++

```
TApplication Application;
```

File

Forms

Description

By default, when a new project is created, the IDE constructs an application object and assigns it to the Application variable in the Forms (see page 13657) or QForms unit. Application has several properties that can be used to get information about an application while it runs.

See Also

TApplication (see page 13660)

TApplication (see page 13660)

TApplication (see page 13660)

72.6.2 Forms.Ctl3DBtnWndProc Variable

Pascal

```
Ctl3DBtnWndProc: Pointer = nil;
```

C++

```
void * Ctl3DBtnWndProc = nil;
```

File

Forms

Description

This is variable Forms.Ctl3DBtnWndProc.

72.6.3 Forms.Ctl3DCtlColorEx Variable

Pascal

```
Ctl3DCtlColorEx: function(Window: HWnd; Msg, wParam, lParam: Longint): Longint stdcall = nil;
```

C++

```
function(Window: HWnd; Msg, wParam, lParam: Longint): Longint stdcall Ctl3DCtlColorEx = nil;
```

File

Forms

Description

This is variable Forms.Ctl3DCtlColorEx.

72.6.4 Forms.Ctl3DDlgFramePaint Variable

Pascal

```
Ctl3DDlgFramePaint: function(Window: HWnd; Msg, wParam, lParam: Longint): Longint stdcall =
nil;
```

C++

```
function(Window: HWnd; Msg, wParam, lParam: Longint): Longint stdcall Ctl3DDlgFramePaint = nil;
```

File

Forms

Description

This is variable Forms.Ctl3DDlgFramePaint.

72.6.5 Forms.HintWindowClass Variable

Specifies the class for the window used to display help hints.

Pascal

```
HintWindowClass: THintWindowClass = THintWindow;
```

C++

```
THintWindowClass HintWindowClass = THintWindow;
```

File

Forms

Description

When the application displays a Help Hint, it creates an instance of HintWindowClass to represent the window used for displaying the hint. Applications can customize this window by creating a descendant of THintWindow and assigning it to the HintWindowClass variable at application startup.

See Also

THintWindow

TApplication (see page 13660)

72.6.6 Forms.Screen Variable

Represents a screen device.

Pascal

```
Screen: TScreen;
```

C++

```
TScreen Screen;
```

File

Forms

Description

The Screen variable is a TScreen (see page 14197) component that represents the screen of the system on which the application runs. By default, applications create a screen component based on information about the current screen device and assign it to Screen.

See Also

TApplication (see page 13660)

72.6.7 Forms.SetLayeredWindowAttributes Variable

Pascal

```
SetLayeredWindowAttributes: TSetLayeredWindowAttributes = nil;
```

C++

```
TSetLayeredWindowAttributes SetLayeredWindowAttributes = nil;
```

File

Forms

Description

This is variable Forms.SetLayeredWindowAttributes.

73 Graphics Namespace

This is namespace Graphics.

Topics

Name	Description
Classes (see page 14395)	The following table lists classes in this documentation.
Interfaces (see page 14606)	The following table lists interfaces in this documentation.
Functions (see page 14608)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 14630)	The following table lists structs, records, enums in this documentation.
Types (see page 14645)	The following table lists types in this documentation.
Variables (see page 14657)	The following table lists variables in this documentation.
Constants (see page 14659)	The following table lists constants in this documentation.

Classes

	Name	Description
	EInvalidGraphic (see page 14395)	EInvalidGraphic is the exception class for unrecognized graphic files.
	EInvalidGraphicOperation (see page 14396)	EInvalidGraphicOperation is the exception class for illegal operations on graphics.
	TBitmap (see page 14396)	TBitmap is an encapsulation of a Windows bitmap (HBITMAP), including its palette (HPALETTE).
	TBitmapImage (see page 14429)	TBitmapImage is the internal representation of the bitmap image for a TBitmap (see page 14396) object.
	TBrush (see page 14430)	TBrush represents the color and pattern used to fill solid shapes.
	TBrushRecall (see page 14439)	TBrushRecall saves and restores a TBrush (see page 14430) object.
	TCanvas (see page 14441)	TCanvas provides an abstract drawing space for objects that must render their own images.
	TFont (see page 14497)	TFont encapsulates a system font.
	TFontRecall (see page 14514)	TFontRecall saves and restores the properties of a TFont (see page 14497) object.
	TGraphic (see page 14516)	TGraphic is the abstract base class type for objects such as icons, bitmaps, and metafiles that can store and display visual images.
	TGraphicsObject (see page 14532)	TGraphicsObject is the abstract base class for objects which encapsulate a system graphics object: TBrush (see page 14430), TFont (see page 14497), and TPen (see page 14571).
	TIcon (see page 14537)	TIcon is an encapsulation of a Windows icon.
	TIconImage (see page 14547)	TIconImage is the internal representation of the bitmap image for a TIcon (see page 14537) object.
	TMetafile (see page 14549)	TMetafile is an encapsulation of the Win32 Enhanced (see page 14562) metafile.

	TMetafileCanvas (see page 14564)	TMetafileCanvas allows applications to create metafile images from scratch.
	TMetafileImage (see page 14569)	TMetafileImage is the internal representation of the metafile image for a TMetafile (see page 14549) object.
	TPen (see page 14571)	TPen is used to draw lines or outline shapes on a canvas.
	TPenRecall (see page 14582)	TPenRecall saves and restores the properties of a TPen (see page 14571) object.
	TPicture (see page 14584)	TPicture contains a bitmap, icon, metafile graphic, or user-defined graphic.
	TSharedImage (see page 14603)	TSharedImage is the base class for classes that contain images used for graphics objects.

Constants

Name	Description
cl3DDkShadow (see page 14663)	Represents the dark shadow of 3D components.
cl3DLight (see page 14663)	Represents the light part of 3D components.
clActiveBorder (see page 14664)	Represents the default border color of an active window.
clActiveCaption (see page 14664)	Represents the default color of the title bar of an active window.
clAppWorkSpace (see page 14665)	Represents the default Application WorkSpace color.
clAqua (see page 14665)	Represents the color Aqua.
clBackground (see page 14666)	Represents the default Windows background color.
clBlack (see page 14666)	Represents the color black.
clBlue (see page 14666)	Represents the color blue.
clBtnFace (see page 14667)	Represents the color of the face of a button.
clBtnHighlight (see page 14667)	Represents the color of the highlighting on a button.
clBtnShadow (see page 14668)	Represents the color of the shadow cast by of a button.
clBtnText (see page 14668)	Represents the default color of the text on a button.
clCaptionText (see page 14669)	Represents the default color of the text on the title bar of an active window.
clCream (see page 14669)	Represents the color cream.
clDefault (see page 14670)	Specifies the default color of a component.
clDkGray (see page 14670)	Represents the color dark gray.
clFuchsia (see page 14670)	Represents the color fuchsia.
clGradientActiveCaption (see page 14671)	Right side color in the color gradient of an active window's title bar.
clGradientInactiveCaption (see page 14671)	Right side color in the color gradient of an inactive window's title bar.
clGray (see page 14672)	Represents the color gray.
clGrayText (see page 14672)	Represents the color of text that is grayed out.
clGreen (see page 14673)	Represents the color green.
clHighlight (see page 14673)	Represents the color of the background of a block of selected text.
clHighlightText (see page 14674)	Represents the color of selected text.
clHotLight (see page 14674)	This is constant Graphics.clHotLight.
clInactiveBorder (see page 14675)	Represents the default border color of an inactive window.
clInactiveCaption (see page 14675)	Represents the default color of the title bar of an inactive window.
clInactiveCaptionText (see page 14675)	Represents the default color of the title text in an inactive window.
clInfoBk (see page 14676)	Represents the color of the background of a tool tip.
clInfoText (see page 14676)	Represents the color of tool tip text.
clLime (see page 14677)	Represents the color lime.
clLtGray (see page 14677)	Represents the color light gray.
clMaroon (see page 14678)	Represents the color maroon.
clMedGray (see page 14678)	Represents the color medium gray.
clMenu (see page 14679)	Represents the color of the background of an unselected menu sub item.
clMenuBar (see page 14679)	Represents the color of the background of an unselected main menu item.
clMenuHighlight (see page 14679)	Represents the color of the background of a selected menu item.
clMenuText (see page 14680)	Represents the default menu text color.
clMoneyGreen (see page 14680)	Represents the color money green.
clNavy (see page 14681)	Represents the color navy.
clNone (see page 14681)	Represents no selected color.
clOlive (see page 14682)	Represents the color olive.

clPurple (see page 14682)	Represents the color purple.
clRed (see page 14683)	Represents the color red.
clScrollBar (see page 14683)	This is constant Graphics.clScrollBar.
clSilver (see page 14683)	Represents the color silver.
clSkyBlue (see page 14684)	Represents the color sky blue.
clSystemColor (see page 14684)	Is a mask used to identify the set of system colors.
clTeal (see page 14685)	Represents the color teal.
clWebAliceBlue (see page 14685)	This is constant Graphics.clWebAliceBlue.
clWebAntiqueWhite (see page 14686)	This is constant Graphics.clWebAntiqueWhite.
clWebAqua (see page 14686)	This is constant Graphics.clWebAqua.
clWebAquamarine (see page 14686)	This is constant Graphics.clWebAquamarine.
clWebAzure (see page 14687)	This is constant Graphics.clWebAzure.
clWebBeige (see page 14687)	This is constant Graphics.clWebBeige.
clWebBisque (see page 14687)	This is constant Graphics.clWebBisque.
clWebBlack (see page 14688)	This is constant Graphics.clWebBlack.
clWebBlanchedAlmond (see page 14688)	This is constant Graphics.clWebBlanchedAlmond.
clWebBlue (see page 14688)	This is constant Graphics.clWebBlue.
clWebBlueViolet (see page 14689)	This is constant Graphics.clWebBlueViolet.
clWebBrown (see page 14689)	This is constant Graphics.clWebBrown.
clWebBurlywood (see page 14689)	This is constant Graphics.clWebBurlywood.
clWebCadetBlue (see page 14690)	This is constant Graphics.clWebCadetBlue.
clWebChartreuse (see page 14690)	This is constant Graphics.clWebChartreuse.
clWebChocolate (see page 14690)	This is constant Graphics.clWebChocolate.
clWebCoral (see page 14691)	This is constant Graphics.clWebCoral.
clWebCornFlowerBlue (see page 14691)	This is constant Graphics.clWebCornFlowerBlue.
clWebCornSilk (see page 14691)	This is constant Graphics.clWebCornSilk.
clWebCrimson (see page 14692)	This is constant Graphics.clWebCrimson.
clWebCyan (see page 14692)	This is constant Graphics.clWebCyan.
clWebDarkBlue (see page 14692)	This is constant Graphics.clWebDarkBlue.
clWebDarkCyan (see page 14693)	This is constant Graphics.clWebDarkCyan.
clWebDarkGoldenRod (see page 14693)	This is constant Graphics.clWebDarkGoldenRod.
clWebDarkGray (see page 14693)	This is constant Graphics.clWebDarkGray.
clWebDarkgreen (see page 14694)	This is constant Graphics.clWebDarkgreen.
clWebDarkKhaki (see page 14694)	This is constant Graphics.clWebDarkKhaki.
clWebDarkMagenta (see page 14694)	This is constant Graphics.clWebDarkMagenta.
clWebDarkOliveGreen (see page 14695)	This is constant Graphics.clWebDarkOliveGreen.
clWebDarkOrange (see page 14695)	This is constant Graphics.clWebDarkOrange.
clWebDarkOrchid (see page 14695)	This is constant Graphics.clWebDarkOrchid.
clWebDarkRed (see page 14696)	This is constant Graphics.clWebDarkRed.
clWebDarkSalmon (see page 14696)	This is constant Graphics.clWebDarkSalmon.
clWebDarkSeaGreen (see page 14696)	This is constant Graphics.clWebDarkSeaGreen.
clWebDarkSlateBlue (see page 14697)	This is constant Graphics.clWebDarkSlateBlue.
clWebDarkSlategray (see page 14697)	This is constant Graphics.clWebDarkSlategray.
clWebDarkTurquoise (see page 14697)	This is constant Graphics.clWebDarkTurquoise.
clWebDarkViolet (see page 14698)	This is constant Graphics.clWebDarkViolet.
clWebDeepPink (see page 14698)	This is constant Graphics.clWebDeepPink.
clWebDeepskyBlue (see page 14698)	This is constant Graphics.clWebDeepskyBlue.
clWebDimGray (see page 14699)	This is constant Graphics.clWebDimGray.
clWebDodgerBlue (see page 14699)	This is constant Graphics.clWebDodgerBlue.
clWebFirebrick (see page 14699)	This is constant Graphics.clWebFirebrick.
clWebFloralWhite (see page 14700)	This is constant Graphics.clWebFloralWhite.
clWebForestGreen (see page 14700)	This is constant Graphics.clWebForestGreen.
clWebFuchsia (see page 14700)	This is constant Graphics.clWebFuchsia.
clWebGainsboro (see page 14701)	This is constant Graphics.clWebGainsboro.

clWebGhostWhite (see page 14701)	This is constant Graphics.clWebGhostWhite.
clWebGold (see page 14701)	This is constant Graphics.clWebGold.
clWebGoldenRod (see page 14702)	This is constant Graphics.clWebGoldenRod.
clWebGray (see page 14702)	This is constant Graphics.clWebGray.
clWebGreen (see page 14702)	This is constant Graphics.clWebGreen.
clWebGreenYellow (see page 14703)	This is constant Graphics.clWebGreenYellow.
clWebHoneydew (see page 14703)	This is constant Graphics.clWebHoneydew.
clWebHotPink (see page 14703)	This is constant Graphics.clWebHotPink.
clWebIndianRed (see page 14704)	This is constant Graphics.clWebIndianRed.
clWebIndigo (see page 14704)	This is constant Graphics.clWebIndigo.
clWebIvory (see page 14704)	This is constant Graphics.clWebIvory.
clWebKhaki (see page 14705)	This is constant Graphics.clWebKhaki.
clWebLavender (see page 14705)	This is constant Graphics.clWebLavender.
clWebLavenderBlush (see page 14705)	This is constant Graphics.clWebLavenderBlush.
clWebLawnGreen (see page 14706)	This is constant Graphics.clWebLawnGreen.
clWebLemonChiffon (see page 14706)	This is constant Graphics.clWebLemonChiffon.
clWebLightBlue (see page 14706)	This is constant Graphics.clWebLightBlue.
clWebLightCoral (see page 14707)	This is constant Graphics.clWebLightCoral.
clWebLightCyan (see page 14707)	This is constant Graphics.clWebLightCyan.
clWebLightGoldenrodYellow (see page 14707)	This is constant Graphics.clWebLightGoldenrodYellow.
clWebLightGreen (see page 14708)	This is constant Graphics.clWebLightGreen.
clWebLightgrey (see page 14708)	This is constant Graphics.clWebLightgrey.
clWebLightPink (see page 14708)	This is constant Graphics.clWebLightPink.
clWebLightSalmon (see page 14709)	This is constant Graphics.clWebLightSalmon.
clWebLightSeaGreen (see page 14709)	This is constant Graphics.clWebLightSeaGreen.
clWebLightSkyBlue (see page 14709)	This is constant Graphics.clWebLightSkyBlue.
clWebLightSlateGray (see page 14710)	This is constant Graphics.clWebLightSlateGray.
clWebLightSteelBlue (see page 14710)	This is constant Graphics.clWebLightSteelBlue.
clWebLightYellow (see page 14710)	This is constant Graphics.clWebLightYellow.
clWebLime (see page 14711)	This is constant Graphics.clWebLime.
clWebLimeGreen (see page 14711)	This is constant Graphics.clWebLimeGreen.
clWebLinen (see page 14711)	This is constant Graphics.clWebLinen.
clWebMagenta (see page 14712)	This is constant Graphics.clWebMagenta.
clWebMaroon (see page 14712)	This is constant Graphics.clWebMaroon.
clWebMediumAquamarine (see page 14712)	This is constant Graphics.clWebMediumAquamarine.
clWebMediumBlue (see page 14713)	This is constant Graphics.clWebMediumBlue.
clWebMediumOrchid (see page 14713)	This is constant Graphics.clWebMediumOrchid.
clWebMediumPurple (see page 14713)	This is constant Graphics.clWebMediumPurple.
clWebMediumSeaGreen (see page 14714)	This is constant Graphics.clWebMediumSeaGreen.
clWebMediumSlateBlue (see page 14714)	This is constant Graphics.clWebMediumSlateBlue.
clWebMediumSpringGreen (see page 14714)	This is constant Graphics.clWebMediumSpringGreen.
clWebMediumTurquoise (see page 14715)	This is constant Graphics.clWebMediumTurquoise.
clWebMediumVioletRed (see page 14715)	This is constant Graphics.clWebMediumVioletRed.
clWebMidnightBlue (see page 14715)	This is constant Graphics.clWebMidnightBlue.
clWebMintcream (see page 14716)	This is constant Graphics.clWebMintcream.
clWebMistyRose (see page 14716)	This is constant Graphics.clWebMistyRose.
clWebMoccasin (see page 14716)	This is constant Graphics.clWebMoccasin.
clWebNavajoWhite (see page 14717)	This is constant Graphics.clWebNavajoWhite.
clWebNavy (see page 14717)	This is constant Graphics.clWebNavy.
clWebOldLace (see page 14717)	This is constant Graphics.clWebOldLace.
clWebOlive (see page 14718)	This is constant Graphics.clWebOlive.
clWebOliveDrab (see page 14718)	This is constant Graphics.clWebOliveDrab.
clWebOrange (see page 14718)	This is constant Graphics.clWebOrange.
clWebOrangeRed (see page 14719)	This is constant Graphics.clWebOrangeRed.

clWebOrchid (see page 14719)	This is constant Graphics.clWebOrchid.
clWebPaleGoldenrod (see page 14719)	This is constant Graphics.clWebPaleGoldenrod.
clWebPaleGreen (see page 14720)	This is constant Graphics.clWebPaleGreen.
clWebPaleTurquoise (see page 14720)	This is constant Graphics.clWebPaleTurquoise.
clWebPaleVioletRed (see page 14720)	This is constant Graphics.clWebPaleVioletRed.
clWebPapayaWhip (see page 14721)	This is constant Graphics.clWebPapayaWhip.
clWebPeachPuff (see page 14721)	This is constant Graphics.clWebPeachPuff.
clWebPeru (see page 14721)	This is constant Graphics.clWebPeru.
clWebPink (see page 14722)	This is constant Graphics.clWebPink.
clWebPlum (see page 14722)	This is constant Graphics.clWebPlum.
clWebPowderBlue (see page 14722)	This is constant Graphics.clWebPowderBlue.
clWebPurple (see page 14723)	This is constant Graphics.clWebPurple.
clWebRed (see page 14723)	This is constant Graphics.clWebRed.
clWebRosyBrown (see page 14723)	This is constant Graphics.clWebRosyBrown.
clWebRoyalBlue (see page 14724)	This is constant Graphics.clWebRoyalBlue.
clWebSaddleBrown (see page 14724)	This is constant Graphics.clWebSaddleBrown.
clWebSalmon (see page 14724)	This is constant Graphics.clWebSalmon.
clWebSandyBrown (see page 14725)	This is constant Graphics.clWebSandyBrown.
clWebSeaGreen (see page 14725)	This is constant Graphics.clWebSeaGreen.
clWebSeashell (see page 14725)	This is constant Graphics.clWebSeashell.
clWebSienna (see page 14726)	This is constant Graphics.clWebSienna.
clWebSilver (see page 14726)	This is constant Graphics.clWebSilver.
clWebSkyBlue (see page 14726)	This is constant Graphics.clWebSkyBlue.
clWebSlateBlue (see page 14727)	This is constant Graphics.clWebSlateBlue.
clWebSlateGray (see page 14727)	This is constant Graphics.clWebSlateGray.
clWebSnow (see page 14727)	This is constant Graphics.clWebSnow.
clWebSpringGreen (see page 14728)	This is constant Graphics.clWebSpringGreen.
clWebSteelBlue (see page 14728)	This is constant Graphics.clWebSteelBlue.
clWebTan (see page 14728)	This is constant Graphics.clWebTan.
clWebTeal (see page 14729)	This is constant Graphics.clWebTeal.
clWebThistle (see page 14729)	This is constant Graphics.clWebThistle.
clWebTomato (see page 14729)	This is constant Graphics.clWebTomato.
clWebTurquoise (see page 14730)	This is constant Graphics.clWebTurquoise.
clWebViolet (see page 14730)	This is constant Graphics.clWebViolet.
clWebWheat (see page 14730)	This is constant Graphics.clWebWheat.
clWebWhite (see page 14731)	This is constant Graphics.clWebWhite.
clWebWhiteSmoke (see page 14731)	This is constant Graphics.clWebWhiteSmoke.
clWebYellow (see page 14731)	This is constant Graphics.clWebYellow.
clWebYellowGreen (see page 14732)	This is constant Graphics.clWebYellowGreen.
clWhite (see page 14732)	Represents the color white.
clWindow (see page 14732)	Represents the default Window background color.
clWindowFrame (see page 14733)	Represents the default Window frame background color.
clWindowText (see page 14733)	Represents the default Window text color.
clYellow (see page 14734)	Represents the color yellow.
cmBlackness (see page 14734)	TCopyMode (see page 14650) value : fills the destination rectangle on the canvas with black.
cmDstInvert (see page 14735)	TCopyMode (see page 14650) value : inverts the image on the canvas and ignores the source.
cmMergeCopy (see page 14735)	TCopyMode (see page 14650) value : combines the image on the canvas and the source bitmap by using the Boolean AND operator.
cmMergePaint (see page 14736)	TCopyMode (see page 14650) value : combines the inverted source bitmap with the image on the canvas by using the Boolean OR operator.
cmNotSrcCopy (see page 14736)	TCopyMode (see page 14650) value : copies the inverted source bitmap to the canvas.
cmNotSrcErase (see page 14737)	TCopyMode (see page 14650) value : combines the image on the canvas and the source bitmap by using the Boolean OR operator, and inverts the result.

cmPatCopy (see page 14737)	TCopyMode (see page 14650) value : copies the source pattern to the canvas.
cmPatInvert (see page 14738)	TCopyMode (see page 14650) value : combines the source pattern with the image on the canvas using the Boolean XOR operator.
cmPatPaint (see page 14738)	TCopyMode (see page 14650) value : combines the inverted source bitmap with the source pattern by using the Boolean OR operator. Combines the result of this operation with the image on the canvas by using the Boolean OR operator.
cmSrcAnd (see page 14739)	TCopyMode (see page 14650) value : combines the image on the canvas and source bitmap by using the Boolean AND operator.
cmSrcCopy (see page 14739)	TCopyMode (see page 14650) value : copies the source bitmap to the canvas.
cmSrcErase (see page 14740)	TCopyMode (see page 14650) value : inverts the image on the canvas and combines the result with the source bitmap by using the Boolean AND operator.
cmSrcInvert (see page 14740)	TCopyMode (see page 14650) value : combines the image on the canvas and the source bitmap by using the Boolean XOR operator.
cmSrcPaint (see page 14741)	TCopyMode (see page 14650) value : combines the image on the canvas and the source bitmap by using the Boolean OR operator.
cmWhiteness (see page 14741)	TCopyMode (see page 14650) value : fills the destination rectangle on the canvas with white.
ExtendedColorsCount (see page 14742)	The total number of extended colors available
rc3_Cursor (see page 14743)	This is constant Graphics.rc3_Cursor.
rc3_Icon (see page 14743)	This is constant Graphics.rc3_Icon.
rc3_StockIcon (see page 14743)	This is constant Graphics.rc3_StockIcon.
StandardColorsCount (see page 14744)	The total number of standard colors available
WebColorsCount (see page 14745)	The total number of web colors available

Functions

	Name	Description
	AllocPatternBitmap (see page 14608)	Creates a TBitmap (see page 14396) filled with a pattern.
	BytesPerScanline (see page 14609)	This is function Graphics.BytesPerScanline.
	CharsetToIdent (see page 14609)	Obtains the name of a character set.
	ColorToIdent (see page 14610)	Provides the symbolic name of a TColor (see page 14648) value.
	ColorToRGB (see page 14610)	Converts a TColor (see page 14648) value into an RGB representation of the color.
	ColorToString (see page 14611)	Returns a string that represents the name of a TColor (see page 14648) value.
	CopyPalette (see page 14612)	Creates a new palette object that matches an existing palette.
	CreateGrayMappedBmp (see page 14612)	Remaps the standard gray colors in a bitmap with the system grays.
	CreateGrayMappedRes (see page 14613)	Remaps the standard gray colors in a bitmap resource with the system grays.
	CreateMappedBmp (see page 14614)	Changes the color table in a bitmap.
	CreateMappedRes (see page 14614)	Changes the color table in a bitmap resource.
	FreeMemoryContexts (see page 14615)	This is function Graphics.FreeMemoryContexts.
	GetCharsetValues (see page 14615)	Passes the name of every predefined character set string to a callback function.
	GetColorValues (see page 14619)	Passes the name of every predefined color constant to a callback function.
	GetDefFontCharSet (see page 14619)	Returns the character set of the default system font.
	GetDIB (see page 14620)	Returns the info header and image for a device independent bitmap.
	GetDIBSizes (see page 14620)	Returns the size of the info header and of the image for a device independent bitmap.
	GraphicExtension (see page 14621)	Returns the default file-name extension of a graphics object.
	GraphicFileMask (see page 14624)	Returns a file mask that specifies the valid extensions for a graphic class.
	GraphicFilter (see page 14624)	Returns a file filter compatible with the Filter property of an Open or Save dialog.
	IdentToCharset (see page 14627)	Translates the name of a character set constant to the corresponding character set.
	IdentToColor (see page 14627)	Translates the name of a color constant to the corresponding TColor (see page 14648) value.
	PaletteChanged (see page 14628)	This is function Graphics.PaletteChanged.
	StringToColor (see page 14628)	Translates a string representation of a color to the corresponding TColor (see page 14648) value.
	TransparentStretchBlt (see page 14629)	This is function Graphics.TransparentStretchBlt.

Interfaces

	Name	Description
	IChangeNotifier (see page 14606)	IChangeNotifier is the interface used for responding to status changes.

Structs, Records, Enums

	Name	Description
	TBitmapHandleType (see page 14630)	TBitmapHandleType indicates the format used to store the in-memory image of a bitmap.
	TBrushData (see page 14631)	This is record Graphics.TBrushData.
	TBrushStyle (see page 14631)	TBrushStyle specifies the pattern on a brush.
	TCanvasOrientation (see page 14632)	TCanvasOrientation specifies the pixel coordinate system that a canvas uses.
	TCanvasStates (see page 14633)	This is record Graphics.TCanvasStates.
	TCursorOrIcon (see page 14633)	This is record Graphics.TCursorOrIcon.
	TFillMode (see page 14634)	This is record Graphics.TFillMode.
	TFillStyle (see page 14634)	TFillStyle indicates the type of color change defines the boundary of a possibly non-rectangular region n a canvas.
	TFontData (see page 14635)	This is record Graphics.TFontData.
	TFontPitch (see page 14635)	TFontPitch indicates whether the characters in a font have the same width.
	TFontStyle (see page 14636)	TFontStyle and TFontStyles (see page 14653) specify style characteristics of a font.
	TIconRec (see page 14636)	This is record Graphics.TIconRec.
	TPenData (see page 14637)	This is record Graphics.TPenData.
	TPenMode (see page 14637)	TPenMode indicates how the color of a pen interacts with the color of a canvas.
	TPenStyle (see page 14639)	TPenStyle indicates the type of line a pen draws.
	TPixelFormat (see page 14640)	TPixelFormat specifies the format of a bitmap.
	TProgressStage (see page 14641)	TProgressStage indicates the stage of a lengthy operation.
	TResData (see page 14641)	This is record Graphics.TResData.
	TResource (see page 14642)	This is record Graphics.TResource.
	TTextFormats (see page 14642)	This is record Graphics.TTextFormats.
	TTransparentMode (see page 14643)	TTransparentMode indicates how the transparent color of a bitmap is determined.

Types

Name	Description
HENHMETAFILE (see page 14645)	Meta file handle type.
HMETAFILE (see page 14646)	This is type Graphics.HMETAFILE.
PColor (see page 14646)	PColor is a pointer to a TColor (see page 14648) type.
PCursorOrIcon (see page 14646)	This is type Graphics.PCursorOrIcon.
PIconRec (see page 14647)	This is type Graphics.PIconRec.
PResource (see page 14647)	This is type Graphics.PResource.
TCanvasState (see page 14647)	This is type Graphics.TCanvasState.
TColor (see page 14648)	TColor is used to specify the color of a Windows-only control.
TCopyMode (see page 14650)	TCopyMode specifies how a graphical image is copied onto a canvas.
TFontCharset (see page 14651)	TFontCharset represents the character set of a font.
TFontDataName (see page 14652)	This is type Graphics.TFontDataName.
TFontName (see page 14653)	TFontName is a string that holds the typeface name of a font.
TFontStyles (see page 14653)	TFontStyle (see page 14636) and TFontStyles specify style characteristics of a font.
TFontStylesBase (see page 14653)	This is type Graphics.TFontStylesBase.
TGraphicClass (see page 14654)	TGraphicClass defines the metaclass for TGraphic (see page 14516).
TProgressEvent (see page 14655)	TProgressEvent is the type of event handlers that provide feedback during lengthy operations.
TTextFormat (see page 14656)	TTextFormat indicates how a text viewer control displays its contents.

Variables

Name	Description
DDBsOnly (see page 14657)	Determines whether bitmap handles are created as device-dependent bitmaps (DDB).
DefFontData (see page 14658)	This is variable Graphics.DefFontData.
SystemPalette16 (see page 14658)	Contains the 16 VGA system colors.

73.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EInvalidGraphic (see page 14395)	EInvalidGraphic is the exception class for unrecognized graphic files.
	EInvalidGraphicOperation (see page 14396)	EInvalidGraphicOperation is the exception class for illegal operations on graphics.
	TBitmap (see page 14396)	TBitmap is an encapsulation of a Windows bitmap (HBITMAP), including its palette (HPALETTE).
	TBitmapImage (see page 14429)	TBitmapImage is the internal representation of the bitmap image for a TBitmap (see page 14396) object.
	TBrush (see page 14430)	TBrush represents the color and pattern used to fill solid shapes.
	TBrushRecall (see page 14439)	TBrushRecall saves and restores a TBrush (see page 14430) object.
	TCanvas (see page 14441)	TCanvas provides an abstract drawing space for objects that must render their own images.
	TFont (see page 14497)	TFont encapsulates a system font.
	TFontRecall (see page 14514)	TFontRecall saves and restores the properties of a TFont (see page 14497) object.
	TGraphic (see page 14516)	TGraphic is the abstract base class type for objects such as icons, bitmaps, and metafiles that can store and display visual images.
	TGraphicsObject (see page 14532)	TGraphicsObject is the abstract base class for objects which encapsulate a system graphics object: TBrush (see page 14430), TFont (see page 14497), and TPen (see page 14571).
	TIcon (see page 14537)	TIcon is an encapsulation of a Windows icon.
	TIconImage (see page 14547)	TIconImage is the internal representation of the bitmap image for a TIcon (see page 14537) object.
	TMetafile (see page 14549)	TMetafile is an encapsulation of the Win32 Enhanced (see page 14562) metafile.
	TMetafileCanvas (see page 14564)	TMetafileCanvas allows applications to create metafile images from scratch.
	TMetafileImage (see page 14569)	TMetafileImage is the internal representation of the metafile image for a TMetafile (see page 14549) object.
	TPen (see page 14571)	TPen is used to draw lines or outline shapes on a canvas.
	TPenRecall (see page 14582)	TPenRecall saves and restores the properties of a TPen (see page 14571) object.
	TPicture (see page 14584)	TPicture contains a bitmap, icon, metafile graphic, or user-defined graphic.
	TSharedImage (see page 14603)	TSharedImage is the base class for classes that contain images used for graphics objects.

73.1.1 EInvalidGraphic Class

EInvalidGraphic is the exception class for unrecognized graphic files.

Class Hierarchy

Pascal

```
EInvalidGraphic = class(Exception);
```

C++

```
class EInvalidGraphic : public Exception;
```

File

Graphics

Description

EInvalidGraphic is raised when

An application attempts to load a file that is not a valid bitmap, icon, metafile, or user-defined graphic type into a picture.

An application attempts to load a file with an unrecognized extension.

An image does not match the format specified in a call to LoadFromClipboardFormat or SaveToClipboardFormat.

An application attempts to set a bitmap's PixelFormat property to an unsupported value.

73.1.2 EInvalidGraphicOperation Class

EInvalidGraphicOperation is the exception class for illegal operations on graphics.

Class Hierarchy

Pascal

```
EInvalidGraphicOperation = class(Exception);
```

C++

```
class EInvalidGraphicOperation : public Exception;
```

File

Graphics

Description

EInvalidGraphicOperation is raised when an illegal operation is attempted on a graphic. For example, an EInvalidGraphicOperation exception is raised when an application attempts to resize an icon or copy an icon to the Clipboard.

See Also

TCanvas (see page 14441)

TIcon (see page 14537)

LoadFromClipboardFormat

SaveToClipboardFormat

TGraphic (see page 14516)

TGraphic (see page 14516)

73.1.3 TBitmap Class

TBitmap is an encapsulation of a Windows bitmap (HBITMAP), including its palette (HPALETTE).

Class Hierarchy

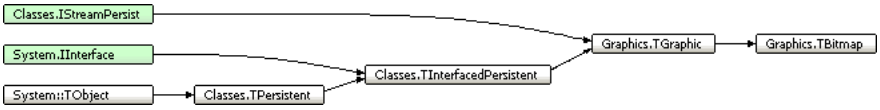

Pascal

```
TBitmap = class(TGraphic);
```

C++

```
class TBitmap : public TGraphic;
```

File

Graphics

Description

A bitmap is a powerful graphics object used to create, manipulate (scale, scroll, rotate, and paint), and store images in memory and as files on a disk.

TBitmap contains an internal image of the bitmap graphic and automatically manages realization of the palette when drawn.

To draw a bitmap on a canvas, call the Draw (see page 14405) or StretchDraw methods of a TCanvas (see page 14441) object, passing a TBitmap as a parameter.

Creating copies of a TBitmap is very fast since the handle is copied rather than the image. If the image is modified and the handle is shared by more than one TBitmap object, the image is copied before the modification is performed (that is, copy on write).

See Also

TCanvas (see page 14441)

TClipboard (see page 3328)

TGraphic (see page 14516)

TPicture (see page 14584)

Using the Main VCL

73.1.3.1 TBitmap Members

The following tables list the members exposed by TBitmap.

Protected Methods

	Name	Description
	_AddRef (see page 2853)	Increments the reference count for the interfaced object's interface.
	_Release (see page 2854)	Decrements the reference count for the interfaced object's interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Changed (see page 14403)	This is Changed, a member of class TBitmap.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 14520)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Draw (see page 14405)	This is Draw, a member of class TBitmap.
V	Equals (see page 14521)	This is Equals, a member of class TGraphic.

	GetEmpty (see page 14407)	This is GetEmpty, a member of class TBitmap.
	GetHeight (see page 14407)	This is GetHeight, a member of class TBitmap.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 14407)	This is GetPalette, a member of class TBitmap.
	GetTransparent (see page 14522)	This is GetTransparent, a member of class TGraphic.
	GetWidth (see page 14407)	This is GetWidth, a member of class TBitmap.
	HandleNeeded (see page 14408)	This is HandleNeeded, a member of class TBitmap.
	MaskHandleNeeded (see page 14412)	This is MaskHandleNeeded, a member of class TBitmap.
	PaletteNeeded (see page 14413)	This is PaletteNeeded, a member of class TBitmap.
	Progress (see page 14524)	This is Progress, a member of class TGraphic.
	ReadData (see page 14413)	This is ReadData, a member of class TBitmap.
	SetHeight (see page 14417)	This is SetHeight, a member of class TBitmap.
	SetPalette (see page 14418)	This is SetPalette, a member of class TBitmap.
	SetTransparent (see page 14527)	This is SetTransparent, a member of class TGraphic.
	SetWidth (see page 14418)	This is SetWidth, a member of class TBitmap.
	WriteData (see page 14418)	This is WriteData, a member of class TBitmap.

Public Events

	Name	Description
	OnChange (see page 14531)	Occurs whenever a graphical image changes.
	OnProgress (see page 14531)	Occurs when a graphical image is in the process of changing.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 2854)	Obtains a reference to the Owner's interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 14402)	Copies a new bitmap image to the bitmap object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 14400)	Instantiates a bitmap object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 14401)	Disposes of the instance of the bitmap object.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dormant (see page 14404)	Creates a memory bitmap image in order to release the bitmap handle, forcing the image into DIB format to save resources.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeImage (see page 14405)	Frees the cached file image stored in memory by the bitmap.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	HandleAllocated (see page 14407)	Indicates whether the TBitmap (see page 14396) object has acquired a handle to the underlying GDI object.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	LoadFromClipboardFormat (see page 14408)	Loads a bitmap from the Clipboard into the bitmap object.
V	LoadFromFile (see page 14523)	Loads a graphic image stored in a file.
	LoadFromResourceID (see page 14410)	Loads a bitmap from a resource into the bitmap object.
	LoadFromResourceName (see page 14410)	Loads a bitmap resource into the bitmap object.
V	LoadFromStream (see page 14412)	Loads the bitmap from a stream into the bitmap object.
	Mask (see page 14412)	Converts the current bitmap image into a monochrome mask, replacing TransparentColor (see page 14427) with white and every other color with black.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	QueryInterface (see page 2854)	Returns a reference to a specified interface if the object supports that interface.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReleaseHandle (see page 14413)	Returns the handle to the bitmap so that the TBitmap (see page 14396) object no longer knows about the handle.
	ReleaseMaskHandle (see page 14414)	Returns the handle to the bitmap's mask so that the TBitmap (see page 14396) object no longer knows about the mask handle.
	ReleasePalette (see page 14415)	Returns the handle to the bitmap's palette and disassociates the palette from the TBitmap (see page 14396) object.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SaveToClipboardFormat (see page 14415)	Creates a local copy of the bitmap in Clipboard format.
V	SaveToFile (see page 14525)	Saves a graphics image to a file.
V	SaveToStream (see page 14417)	Saves the bitmap to a stream.
V	SetSize (see page 14418)	Specifies the size of the bitmap.

Public Properties

	Name	Description
R	Canvas (see page 14419)	Provides access to a drawing surface that represents the bitmap.
R	Empty (see page 14528)	Indicates whether the graphics object contains a graphic.
	Handle (see page 14420)	Provides access to the Windows GDI bitmap handle for accessing the GDI bitmap object.
	HandleType (see page 14420)	Indicates whether the bitmap is a device-dependent bitmap, or a device-independent bitmap.
	Height (see page 14528)	Specifies the vertical size of the graphic in pixels.
	IgnorePalette (see page 14421)	Determines whether the bitmap realizes its palette when drawing its image.
	MaskHandle (see page 14421)	Provides access to the Windows GDI bitmap handle for accessing the GDI bitmap object.
	Modified (see page 14528)	Indicates whether the graphics object has been changed or edited.
	Monochrome (see page 14421)	Determines whether the bitmap displays its image in monochrome.
	Palette (see page 14529)	Indicates the color palette of the graphical image.
	PaletteModified (see page 14529)	Indicates whether the palette has changed.
	PixelFormat (see page 14423)	Indicates the bit format of the bitmap image, specifying how the image is displayed and how the pixels of the bitmap image are stored in memory.
R	ScanLine (see page 14425)	Provides indexed access to each line of pixels.
	Transparent (see page 14529)	Indicates whether the image covers its rectangular area.
	TransparentColor (see page 14427)	Determines which color of the bitmap is to be transparent when the bitmap is drawn.
	TransparentMode (see page 14428)	Determines whether the TransparentColor (see page 14427) property's value is automatically calculated or stored with the bitmap object.
	Width (see page 14530)	Determines the maximum width of the graphics object in pixels.

73.1.3.2 TBitmap.Create Constructor

Instantiates a bitmap object.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TBitmap();
```

Description

Call Create to instantiate a bitmap object at runtime. Create is also called automatically when a bitmap image is loaded into a TImage.

Create allocates memory for a bitmap object, and calls the inherited Create. Then it creates a TBitmapImage (see page 14429) as the internal image that represents the bitmap.

C++ Examples:

```
/*
This example requires a resource file in the sources directory.
*/

#pragma resource "extrares.res"

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
  try
  {
    Bitmap1->LoadFromResourceName((int)HInstance, "Live");
    Canvas->Draw(12,12,Bitmap1);
  }
  catch (...)
  {
    MessageBeep(0);
  }
  delete Bitmap1;
}
```

Delphi Examples:

```
{
LoadFromResourceName example
}

{$R extrares.res}

procedure TForm1.Button1Click(Sender: TObject);
var
 BitMap1 : TBitMap;
begin
  BitMap1 := TBitMap.Create;
  try
    BitMap1.LoadFromResourceName(HInstance,'Live');
    Canvas.Draw(12,12,BitMap1);
  finally
    BitMap1.Free;
  end;
end;
```

See Also

TBitmapImage (see page 14429)

73.1.3.3 TBitmap.Destroy Destructor

Disposes of the instance of the bitmap object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBitmap();
```

Description

Destroy is the destructor for a TBitmap (see page 14396) object.

Do not call the destructor directly in an application. Instead, call Free. Free verifies that the bitmap object is not nil before it calls Destroy.

Destroy releases the internal bitmap image (TBitmapImage (see page 14429) object) and frees the canvas before calling the inherited destructor.

See Also

Free (see page 22300)

delete

TBitmapImage (see page 14429)

73.1.3.4 TBitmap Methods

The methods of the TBitmap class are listed here.

Protected Methods

	Name	Description
	Changed (see page 14403)	This is Changed, a member of class TBitmap.
	Draw (see page 14405)	This is Draw, a member of class TBitmap.
	GetEmpty (see page 14407)	This is GetEmpty, a member of class TBitmap.
	GetHeight (see page 14407)	This is GetHeight, a member of class TBitmap.
	GetPalette (see page 14407)	This is GetPalette, a member of class TBitmap.
	GetWidth (see page 14407)	This is GetWidth, a member of class TBitmap.
	HandleNeeded (see page 14408)	This is HandleNeeded, a member of class TBitmap.
	MaskHandleNeeded (see page 14412)	This is MaskHandleNeeded, a member of class TBitmap.
	PaletteNeeded (see page 14413)	This is PaletteNeeded, a member of class TBitmap.
	ReadData (see page 14413)	This is ReadData, a member of class TBitmap.
	SetHeight (see page 14417)	This is SetHeight, a member of class TBitmap.
	SetPalette (see page 14418)	This is SetPalette, a member of class TBitmap.
	SetWidth (see page 14418)	This is SetWidth, a member of class TBitmap.
	WriteData (see page 14418)	This is WriteData, a member of class TBitmap.

Public Methods

	Name	Description
	Assign (see page 14402)	Copies a new bitmap image to the bitmap object.

	Dormant (see page 14404)	Creates a memory bitmap image in order to release the bitmap handle, forcing the image into DIB format to save resources.
	FreeImage (see page 14405)	Frees the cached file image stored in memory by the bitmap.
	HandleAllocated (see page 14407)	Indicates whether the TBitmap (see page 14396) object has acquired a handle to the underlying GDI object.
V	LoadFromClipboardFormat (see page 14408)	Loads a bitmap from the Clipboard into the bitmap object.
	LoadFromResourceID (see page 14410)	Loads a bitmap from a resource into the bitmap object.
	LoadFromResourceName (see page 14410)	Loads a bitmap resource into the bitmap object.
V	LoadFromStream (see page 14412)	Loads the bitmap from a stream into the bitmap object.
	Mask (see page 14412)	Converts the current bitmap image into a monochrome mask, replacing TransparentColor (see page 14427) with white and every other color with black.
	ReleaseHandle (see page 14413)	Returns the handle to the bitmap so that the TBitmap (see page 14396) object no longer knows about the handle.
	ReleaseMaskHandle (see page 14414)	Returns the handle to the bitmap's mask so that the TBitmap (see page 14396) object no longer knows about the mask handle.
	ReleasePalette (see page 14415)	Returns the handle to the bitmap's palette and disassociates the palette from the TBitmap (see page 14396) object.
V	SaveToClipboardFormat (see page 14415)	Creates a local copy of the bitmap in Clipboard format.
V	SaveToStream (see page 14417)	Saves the bitmap to a stream.
V	SetSize (see page 14418)	Specifies the size of the bitmap.

73.1.3.4.1 TBitmap.Assign Method

Copies a new bitmap image to the bitmap object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Assign copies the bitmap image contained in Source to the bitmap object. If Source is not a bitmap, Assign calls the inherited Assign method, which can copy an image from any class that knows how to copy to a TBitmap (see page 14396) object. If the bitmap needs to be changed, the actual bitmap image is copied before the changes are made (copy on write).

Note: An object of one type can always be assigned to another object of the same type. Also, the Source can be of type TPicture (see page 14584) if the Graphic property of the picture is a bitmap.

C++ Examples:

```
/*
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
Note: For the C++ Builder, capitalization matters for the file
name and path.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
  Graphics::TBitmap *Bitmap2 = new Graphics::TBitmap();
  try
  {
    Bitmap1->LoadFromFile("..\\FACTORY.BMP");
    Bitmap2->Assign(Bitmap1);     // Copy Bitmap1 into Bitmap2
    Bitmap2->Dormant();           // Free up GDI resources
```

```
    Bitmap2->FreeImage();          // Free up Memory.
    Canvas->Draw(20,20,Bitmap2);  // Note that previous calls don't lose the image
    Bitmap2->Monochrome = true;
    Canvas->Draw(80,80,Bitmap2);
    Bitmap2->ReleaseHandle();       // This will actually lose the bitmap;
  }
  catch (...)
  {
    MessageBeep(0);
  }
  delete Bitmap1;
  delete Bitmap2;
}
```

Delphi Examples:

```
{
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
}
procedure TForm1.Button1Click(Sender: TObject);
var
 BitMap1,BitMap2 : TBitMap;
 MyFormat : Word;
begin
   BitMap2 := TBitMap.Create;
   BitMap1 := TBitMap.Create;
try
   BitMap1.LoadFromFile('factory.bmp');
   BitMap2.Assign(BitMap1);     // Copy BitMap1 into BitMap2
   BitMap2.Dormant;             // Free up GDI resources
   BitMap2.FreeImage;           // Free up Memory.
   Canvas.Draw(20,20,BitMap2);  // Note that previous calls don't lose the image
   BitMap2.Monochrome := true;
   Canvas.Draw(80,80,BitMap2);
   BitMap2.ReleaseHandle;       // This will actually lose the bitmap;
 finally
   BitMap1.Free;
   BitMap2.Free;
 end;
end;
```

See Also

Assign (see page 2930)

TPicture (see page 14584)

Graphic (see page 14600)

73.1.3.4.2 TBitmap.Changed Method

Pascal

```
procedure Changed(Sender: TObject); override;
```

C++

```
virtual __fastcall Changed(TObject * Sender);
```

Description

This is Changed, a member of class TBitmap.

73.1.3.4.3 TBitmap.Dormant Method

Creates a memory bitmap image in order to release the bitmap handle, forcing the image into DIB format to save resources.

Pascal

```
procedure Dormant;
```

C++

```
__fastcall Dormant();
```

Description

Use Dormant to change the format of the bitmap in memory thereby reducing the amount of GDI resources used by the application.

Dormant creates a bitmap image in memory using a memory stream object. This preserves the image so that the bitmap can then free the HBITMAP (accessed through the Handle (see page 14420) property) that was assigned to it.

DIB handles may use fewer Win95 GDI resources than DDB, but DIBs may also use more memory than DDBs, depending on the current video driver and mode.

C++ Examples:

```
/*
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
Note: For the C++ Builder, capitalization matters for the file
name and path.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
  Graphics::TBitmap *Bitmap2 = new Graphics::TBitmap();
  try
  {
    Bitmap1->LoadFromFile("..\\FACTORY.BMP");
    Bitmap2->Assign(Bitmap1);     // Copy Bitmap1 into Bitmap2
    Bitmap2->Dormant();           // Free up GDI resources
    Bitmap2->FreeImage();         // Free up Memory.
    Canvas->Draw(20,20,Bitmap2);  // Note that previous calls don't lose the image
    Bitmap2->Monochrome = true;
    Canvas->Draw(80,80,Bitmap2);
    Bitmap2->ReleaseHandle();       // This will actually lose the bitmap;
  }
  catch (...)
  {
    MessageBeep(0);
  }
  delete Bitmap1;
  delete Bitmap2;
}
```

Delphi Examples:

```
{
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
}
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
 BitMap1,BitMap2 : TBitMap;
 MyFormat : Word;
begin
   BitMap2 := TBitMap.Create;
   BitMap1 := TBitMap.Create;
try
   BitMap1.LoadFromFile('factory.bmp');
   BitMap2.Assign(BitMap1);     // Copy BitMap1 into BitMap2
   BitMap2.Dormant;             // Free up GDI resources
   BitMap2.FreeImage;           // Free up Memory.
   Canvas.Draw(20,20,BitMap2);  // Note that previous calls don't lose the image
   BitMap2.Monochrome := true;
   Canvas.Draw(80,80,BitMap2);
   BitMap2.ReleaseHandle;       // This will actually lose the bitmap;
 finally
   BitMap1.Free;
   BitMap2.Free;
 end;
end;
```

See Also

Handle (see page 14420)

TBitmapImage (see page 14429)

TMemoryStream (see page 2902)

73.1.3.4.4 TBitmap.Draw Method

Pascal

```
procedure Draw(ACanvas: TCanvas; const Rect: TRect); override;
```

C++

```
virtual __fastcall Draw(TCanvas ACanvas, const TRect Rect);
```

Description

This is Draw, a member of class TBitmap.

73.1.3.4.5 TBitmap.FreeImage Method

Frees the cached file image stored in memory by the bitmap.

Pascal

```
procedure FreeImage;
```

C++

```
__fastcall FreeImage();
```

Description

Use FreeImage to reduce the memory requirements of an application when color depth and pixel format are not an issue. Freeing the image releases the memory allocated for the bitmap image when it was originally loaded to disk. Consequently, some of the original pixel format of the bitmap is lost (for example, if you changed its format to a DIB) as well as the color depth of the bitmap.

When a bitmap is loaded into a bitmap object, the bitmap object creates an image of the loaded bitmap in memory. If the bitmap isn't changed, the memory image is used when saving the bitmap, to verify that the bitmap has not lost color depth or changed the pixel format.

C++ Examples:

```
/*
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
Note: For the C++ Builder, capitalization matters for the file
name and path.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
  Graphics::TBitmap *Bitmap2 = new Graphics::TBitmap();
  try
  {
    Bitmap1->LoadFromFile("..\\FACTORY.BMP");
    Bitmap2->Assign(Bitmap1);     // Copy Bitmap1 into Bitmap2
    Bitmap2->Dormant();           // Free up GDI resources
    Bitmap2->FreeImage();         // Free up Memory.
    Canvas->Draw(20,20,Bitmap2);  // Note that previous calls don't lose the image
    Bitmap2->Monochrome = true;
    Canvas->Draw(80,80,Bitmap2);
    Bitmap2->ReleaseHandle();       // This will actually lose the bitmap;
  }
  catch (...)
  {
    MessageBeep(0);
  }
  delete Bitmap1;
  delete Bitmap2;
}
```

Delphi Examples:

```
{
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
}
procedure TForm1.Button1Click(Sender: TObject);
var
 BitMap1,BitMap2 : TBitMap;
 MyFormat : Word;
begin
   BitMap2 := TBitMap.Create;
   BitMap1 := TBitMap.Create;
try
   BitMap1.LoadFromFile('factory.bmp');
   BitMap2.Assign(BitMap1);     // Copy BitMap1 into BitMap2
   BitMap2.Dormant;             // Free up GDI resources
   BitMap2.FreeImage;           // Free up Memory.
   Canvas.Draw(20,20,BitMap2);  // Note that previous calls don't lose the image
   BitMap2.Monochrome := true;
   Canvas.Draw(80,80,BitMap2);
   BitMap2.ReleaseHandle;       // This will actually lose the bitmap;
 finally
   BitMap1.Free;
   BitMap2.Free;
 end;
end;
```

73.1.3.4.6 TBitmap.GetEmpty Method

Pascal

```
function GetEmpty: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetEmpty();
```

Description

This is GetEmpty, a member of class TBitmap.

73.1.3.4.7 TBitmap.GetHeight Method

Pascal

```
function GetHeight: Integer; override;
```

C++

```
virtual __fastcall int GetHeight();
```

Description

This is GetHeight, a member of class TBitmap.

73.1.3.4.8 TBitmap.GetPalette Method

Pascal

```
function GetPalette: HPALETTE; override;
```

C++

```
virtual __fastcall HPALETTE GetPalette();
```

Description

This is GetPalette, a member of class TBitmap.

73.1.3.4.9 TBitmap.GetWidth Method

Pascal

```
function GetWidth: Integer; override;
```

C++

```
virtual __fastcall int GetWidth();
```

Description

This is GetWidth, a member of class TBitmap.

73.1.3.4.10 TBitmap.HandleAllocated Method

Indicates whether the TBitmap (see page 14396) object has acquired a handle to the underlying GDI object.

Pascal

```
function HandleAllocated: Boolean;
```

C++

```
__fastcall Boolean HandleAllocated();
```

Description

Use HandleAllocated to determine whether the Handle (see page 14420) property is set to an HBITMAP value. HandleAllocated indicates whether the underlying GDI object has already been created. Use HandleAllocated rather than reading the Handle (see page 14420) property. When an application reads the Handle (see page 14420) property, it automatically creates a bitmap and returns its handle if the underlying GDI object does not already exist.

See Also

Handle (see page 14420)

73.1.3.4.11 TBitmap.HandleNeeded Method

Pascal

```
procedure HandleNeeded;
```

C++

```
__fastcall HandleNeeded();
```

Description

This is HandleNeeded, a member of class TBitmap.

73.1.3.4.12 TBitmap.LoadFromClipboardFormat Method

Loads a bitmap from the Clipboard into the bitmap object.

Pascal

```
procedure LoadFromClipboardFormat(AFormat: Word; AData: THandle; APalette: HPALETTE); override;
```

C++

```
virtual __fastcall LoadFromClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette);
```

Description

LoadFromClipboardFormat is called if the bitmap is registered with the TPicture (see page 14584) object using the RegisterClipboardFormat method.

LoadFromClipboardFormat replaces the current image with the data pointed to by the AData parameter. The palette for the bitmap is specified by the APalette parameter.

C++ Examples:

```
/*
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
*/
#include <Clipbrd.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClipboard *cb = Clipboard();
  if (cb->HasFormat(CF_BITMAP))
  {
    Graphics::TBitmap *Bitmap = new Graphics::TBitmap();
```

```
    try
    {
      Bitmap->LoadFromClipboardFormat(CF_BITMAP, cb->GetAsHandle(CF_BITMAP), 0);
      Canvas->Draw(5,5,Bitmap);
    }
    catch (...)
    {
      MessageBeep(0);
      ShowMessage("Error reading image from clipboard!");
    }
    delete Bitmap;
    cb->Clear();
  }
  else
  {
    MessageBeep(0);
    ShowMessage("Cannot find image in clipboard!");
  }
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  unsigned int DataHandle;
  HPALETTE APalette;
  unsigned short MyFormat;
  Graphics::TBitmap *Bitmap = new Graphics::TBitmap();
  try
  {
    Bitmap->LoadFromFile("C:\\Program Files\\Common Files\\CodeGear
Shared\\Images\\Splash\\256Color\\FACTORY.BMP");
    // generate a clipboard format, with data and palette
    Bitmap->SaveToClipboardFormat(
      MyFormat,
      DataHandle,
      APalette);
    // save the data to the clipboard using that format and
    // the generated data
    Clipboard()->SetAsHandle(MyFormat,DataHandle);
  }
  catch (...)
  {
     ShowMessage("Failed to copy image to clipboard!");
  }
  delete Bitmap;
}
```

Delphi Examples:

```
{
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitmap;
  s : string;
  Icon: TIcon;
begin
 Bitmap := TBitMap.Create;
 try
   Image1.Picture.RegisterClipboardFormat(cf_BitMap,TIcon);
   Bitmap.LoadFromClipBoardFormat(
     cf_BitMap,ClipBoard.GetAsHandle(cf_Bitmap),0);
   Canvas.draw(0,0,Bitmap);
```

```
  finally
    Bitmap.free;
    Clipboard.Clear;
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  MyFormat : Word;
  Bitmap : TBitMap;
  AData : THandle;
  APalette : HPALETTE;
begin
  Bitmap := TBitmap.Create;
  try
    Bitmap.LoadFromFile('c:\Program Files\Common Files\CodeGear
Shared\Images\Splash\256color\factory.bmp');
    Bitmap.SaveToClipBoardFormat(
      MyFormat,
      AData,
      APalette);
    ClipBoard.SetAsHandle(MyFormat,AData);
  finally
    Bitmap.Free;
  end;
end;
```

See Also

TClipboard (see page 3328)

TPicture (see page 14584)

RegisterClipboardFormat (see page 14592)

73.1.3.4.13 TBitmap.LoadFromResourceID Method

Loads a bitmap from a resource into the bitmap object.

Pascal

```
procedure LoadFromResourceID(Instance: THandle; ResID: Integer);
```

C++

```
__fastcall LoadFromResourceID(THandle Instance, int ResID);
```

Description

Use LoadFromResourceID to load a specified bitmap resource along with palette information from a module's executable file.

Instance is the handle of the module that contains the resource.

ResID is the resource ID for the bitmap.

See Also

LoadFromResourceName (see page 14410)

73.1.3.4.14 TBitmap.LoadFromResourceName Method

Loads a bitmap resource into the bitmap object.

Pascal

```
procedure LoadFromResourceName(Instance: THandle; const ResName: String);
```

C++

```
__fastcall LoadFromResourceName(THandle Instance, const AnsiString ResName);
```

Description

LoadFromResourceName loads the specified bitmap resource along with palette information from a module's executable file.

Instance is the handle of the module that contains the resource.

ResName is the name of the resource to load.

Note: Use this routine to load bitmaps from RES files instead of the LoadBitmap API. LoadBitmap does not support 256-color images.

C++ Examples:

```
/*
This example requires a resource file in the sources directory.
*/

#pragma resource "extrares.res"

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
  try
  {
    Bitmap1->LoadFromResourceName((int)HInstance, "Live");
    Canvas->Draw(12,12,Bitmap1);
  }
  catch (...)
  {
    MessageBeep(0);
  }
  delete Bitmap1;
}
```

Delphi Examples:

```
{
LoadFromResourceName example
}

{$R extrares.res}

procedure TForm1.Button1Click(Sender: TObject);
var
 BitMap1 : TBitMap;
begin
  BitMap1 := TBitMap.Create;
  try
    BitMap1.LoadFromResourceName(HInstance,'Live');
    Canvas.Draw(12,12,BitMap1);
  finally
    BitMap1.Free;
  end;
end;
```

See Also

LoadFromResourceID (see page 14410)

HInstance

73.1.3.4.15 TBitmap.LoadFromStream Method

Loads the bitmap from a stream into the bitmap object.

Pascal

```
procedure LoadFromStream(Stream: TStream); override;
```

C++

```
virtual __fastcall LoadFromStream(TStream Stream);
```

Description

Use LoadFromStream to load the bitmap image from a stream. The Stream parameter specifies the stream from which to read the bitmap image. For example, use LoadFromStream to load a bitmap image from a BLOB stream that reads from a graphic field in a dataset.

See Also

LoadFromClipboardFormat (see page 14408)

LoadFromResourceName (see page 14410)

LoadFromResourceID (see page 14410)

LoadFromFile (see page 14523)

SaveToStream (see page 14417)

73.1.3.4.16 TBitmap.Mask Method

Converts the current bitmap image into a monochrome mask, replacing TransparentColor (see page 14427) with white and every other color with black.

Pascal

```
procedure Mask(TransparentColor: TColor);
```

C++

```
__fastcall Mask(TColor TransparentColor);
```

Description

Use Mask to produce a monochrome mask bitmap based on a given transparent color. Mask produces the same image that MaskHandle (see page 14421) does, except that the resulting mask image replaces the image in the TBitmap (see page 14396) object, so it only makes sense to call Mask once. Use MaskHandle (see page 14421) when only a bitmap handle of the mask image is needed.

See Also

ReleaseMaskHandle (see page 14414)

73.1.3.4.17 TBitmap.MaskHandleNeeded Method

Pascal

```
procedure MaskHandleNeeded;
```

C++

```
__fastcall MaskHandleNeeded();
```

Description

This is MaskHandleNeeded, a member of class TBitmap.

73.1.3.4.18 TBitmap.PaletteNeeded Method

Pascal

```
procedure PaletteNeeded;
```

C++

```
__fastcall PaletteNeeded();
```

Description

This is PaletteNeeded, a member of class TBitmap.

73.1.3.4.19 TBitmap.ReadData Method

Pascal

```
procedure ReadData(Stream: TStream); override;
```

C++

```
virtual __fastcall ReadData(TStream Stream);
```

Description

This is ReadData, a member of class TBitmap.

73.1.3.4.20 TBitmap.ReleaseHandle Method

Returns the handle to the bitmap so that the TBitmap (see page 14396) object no longer knows about the handle.

Pascal

```
function ReleaseHandle: HBITMAP;
```

C++

```
__fastcall HBITMAP ReleaseHandle();
```

Description

Use ReleaseHandle to disassociate the bitmap from the bitmap handle. Use it when you need to give a bitmap handle to a routine or object that will assume ownership (or destroy) the bitmap handle.

C++ Examples:

```
/*
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
Note: For the C++ Builder, capitalization matters for the file
name and path.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
  Graphics::TBitmap *Bitmap2 = new Graphics::TBitmap();
  try
  {
```

```
      Bitmap1->LoadFromFile("..\\FACTORY.BMP");
      Bitmap2->Assign(Bitmap1);     // Copy Bitmap1 into Bitmap2
      Bitmap2->Dormant();           // Free up GDI resources
      Bitmap2->FreeImage();         // Free up Memory.
      Canvas->Draw(20,20,Bitmap2);  // Note that previous calls don't lose the image
      Bitmap2->Monochrome = true;
      Canvas->Draw(80,80,Bitmap2);
      Bitmap2->ReleaseHandle();       // This will actually lose the bitmap;
    }
    catch (...)
    {
      MessageBeep(0);
    }
    delete Bitmap1;
    delete Bitmap2;
  }
```

Delphi Examples:

```
{
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
}
procedure TForm1.Button1Click(Sender: TObject);
var
 BitMap1,BitMap2 : TBitMap;
 MyFormat : Word;
begin
   BitMap2 := TBitMap.Create;
   BitMap1 := TBitMap.Create;
try
   BitMap1.LoadFromFile('factory.bmp');
   BitMap2.Assign(BitMap1);     // Copy BitMap1 into BitMap2
   BitMap2.Dormant;             // Free up GDI resources
   BitMap2.FreeImage;           // Free up Memory.
   Canvas.Draw(20,20,BitMap2);  // Note that previous calls don't lose the image
   BitMap2.Monochrome := true;
   Canvas.Draw(80,80,BitMap2);
   BitMap2.ReleaseHandle;       // This will actually lose the bitmap;
 finally
   BitMap1.Free;
   BitMap2.Free;
 end;
end;
```

See Also

ReleaseMaskHandle (see page 14414)

Handle (see page 14420)

73.1.3.4.21 TBitmap.ReleaseMaskHandle Method

Returns the handle to the bitmap's mask so that the TBitmap (see page 14396) object no longer knows about the mask handle.

Pascal

```
function ReleaseMaskHandle: HBITMAP;
```

C++

```
__fastcall HBITMAP ReleaseMaskHandle();
```

Description

Use ReleaseMaskHandle to disassociate the bitmap object from the bitmap mask handle.

See Also

ReleaseHandle (see page 14413)

Handle (see page 14420)

Mask (see page 14412)

73.1.3.4.22 TBitmap.ReleasePalette Method

Returns the handle to the bitmap's palette and disassociates the palette from the TBitmap (see page 14396) object.

Pascal

```
function ReleasePalette: HPALETTE;
```

C++

```
__fastcall HPALETTE ReleasePalette();
```

Description

Use ReleasePalette to disassociate the palette from the bitmap image.

73.1.3.4.23 TBitmap.SaveToClipboardFormat Method

Creates a local copy of the bitmap in Clipboard format.

Pascal

```
procedure SaveToClipboardFormat(var Format: Word; var Data: THandle; var APalette: HPALETTE);
override;
```

C++

```
virtual __fastcall SaveToClipboardFormat(Word Format, THandle Data, HPALETTE APalette);
```

Description

Use SaveToClipboardFormat to copy the bitmap to a Clipboard format. The bitmap's palette is returned in the APalette parameter, the format in the AFormat parameter, and a handle to the data in the AData parameter. Before the bitmap can be saved, an application must have registered the format with the TBitmap (see page 14396) object using the RegisterClipboardFormat method.

C++ Examples:

```
/*
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
*/
#include <Clipbrd.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClipboard *cb = Clipboard();
  if (cb->HasFormat(CF_BITMAP))
  {
    Graphics::TBitmap *Bitmap = new Graphics::TBitmap();
    try
    {
```

```
      Bitmap->LoadFromClipboardFormat(CF_BITMAP, cb->GetAsHandle(CF_BITMAP), 0);
      Canvas->Draw(5,5,Bitmap);
    }
    catch (...)
    {
      MessageBeep(0);
      ShowMessage("Error reading image from clipboard!");
    }
    delete Bitmap;
    cb->Clear();
  }
  else
  {
    MessageBeep(0);
    ShowMessage("Cannot find image in clipboard!");
  }
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  unsigned int DataHandle;
  HPALETTE APalette;
  unsigned short MyFormat;
  Graphics::TBitmap *Bitmap = new Graphics::TBitmap();
  try
  {
    Bitmap->LoadFromFile("C:\\Program Files\\Common Files\\CodeGear
Shared\\Images\\Splash\\256Color\\FACTORY.BMP");
    // generate a clipboard format, with data and palette
    Bitmap->SaveToClipboardFormat(
      MyFormat,
      DataHandle,
      APalette);
    // save the data to the clipboard using that format and
    // the generated data
    Clipboard()->SetAsHandle(MyFormat,DataHandle);
  }
  catch (...)
  {
     ShowMessage("Failed to copy image to clipboard!");
  }
  delete Bitmap;
}
```

Delphi Examples:

```
{
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitmap;
  s : string;
  Icon: TIcon;
begin
 Bitmap := TBitMap.Create;
 try
   Image1.Picture.RegisterClipboardFormat(cf_BitMap,TIcon);
   Bitmap.LoadFromClipBoardFormat(
     cf_BitMap,ClipBoard.GetAsHandle(cf_Bitmap),0);
   Canvas.draw(0,0,Bitmap);
 finally
   Bitmap.free;
```

```
    Clipboard.Clear;
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  MyFormat : Word;
  Bitmap : TBitMap;
  AData : THandle;
  APalette : HPALETTE;
begin
  Bitmap := TBitmap.Create;
  try
    Bitmap.LoadFromFile('c:\Program Files\Common Files\CodeGear
Shared\Images\Splash\256color\factory.bmp');
    Bitmap.SaveToClipBoardFormat(
      MyFormat,
      AData,
      APalette);
    ClipBoard.SetAsHandle(MyFormat,AData);
  finally
    Bitmap.Free;
  end;
end;
```

See Also

RegisterClipboardFormat (see page 14592)

LoadFromClipboardFormat (see page 14408)

73.1.3.4.24 TBitmap.SaveToStream Method

Saves the bitmap to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream); override;
```

C++

```
virtual __fastcall SaveToStream(TStream Stream);
```

Description

Use SaveToStream to write the bitmap image to the stream specified by the Stream parameter. For example, specify a TBlobStream object as the Stream parameter to save the bitmap image to a graphic field in a dataset.

See Also

LoadFromStream (see page 14412)

73.1.3.4.25 TBitmap.SetHeight Method

Pascal

```
procedure SetHeight(Value: Integer); override;
```

C++

```
virtual __fastcall SetHeight(int Value);
```

Description

This is SetHeight, a member of class TBitmap.

73.1.3.4.26 TBitmap.SetPalette Method

Pascal

```
procedure SetPalette(Value: HPALETTE); override;
```

C++

```
virtual __fastcall SetPalette(HPALETTE Value);
```

Description

This is SetPalette, a member of class TBitmap.

73.1.3.4.27 TBitmap.SetSize Method

Specifies the size of the bitmap.

Pascal

```
procedure SetSize(AWidth: Integer; AHeight: Integer); override;
```

C++

```
virtual __fastcall SetSize(int AWidth, int AHeight);
```

Description

Use the SetSize method to set both the height and width of the bitmap. This results in better performance than setting the height and width separately.

73.1.3.4.28 TBitmap.SetWidth Method

Pascal

```
procedure SetWidth(Value: Integer); override;
```

C++

```
virtual __fastcall SetWidth(int Value);
```

Description

This is SetWidth, a member of class TBitmap.

73.1.3.4.29 TBitmap.WriteData Method

Pascal

```
procedure WriteData(Stream: TStream); override;
```

C++

```
virtual __fastcall WriteData(TStream Stream);
```

Description

This is WriteData, a member of class TBitmap.

73.1.3.5 TBitmap Properties

The properties of the TBitmap class are listed here.

Public Properties

	Name	Description
R	Canvas (see page 14419)	Provides access to a drawing surface that represents the bitmap.
	Handle (see page 14420)	Provides access to the Windows GDI bitmap handle for accessing the GDI bitmap object.
	HandleType (see page 14420)	Indicates whether the bitmap is a device-dependent bitmap, or a device-independent bitmap.
	IgnorePalette (see page 14421)	Determines whether the bitmap realizes its palette when drawing its image.
	MaskHandle (see page 14421)	Provides access to the Windows GDI bitmap handle for accessing the GDI bitmap object.
	Monochrome (see page 14421)	Determines whether the bitmap displays its image in monochrome.
	PixelFormat (see page 14423)	Indicates the bit format of the bitmap image, specifying how the image is displayed and how the pixels of the bitmap image are stored in memory.
R	ScanLine (see page 14425)	Provides indexed access to each line of pixels.
	TransparentColor (see page 14427)	Determines which color of the bitmap is to be transparent when the bitmap is drawn.
	TransparentMode (see page 14428)	Determines whether the TransparentColor (see page 14427) property's value is automatically calculated or stored with the bitmap object.

73.1.3.5.1 TBitmap.Canvas Property

Provides access to a drawing surface that represents the bitmap.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Canvas allows drawing on the bitmap by providing a T Canvas object for this purpose. Drawing on the canvas effectively modifies the underlying bitmap image pixels. Any canvas operation is valid on a bitmap (not just Draw (see page 14405) and StretchDraw) including line drawing, rectangles, and circles. The bitmap object is passed as a parameter to these methods.

A canvas object is created automatically for the bitmap and the property is read-only.

C++ Examples:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *bitmap = new Graphics::TBitmap();
  try
  {
    bitmap->LoadFromFile("..\\FACTORY.BMP ");
    bitmap->Transparent = true;
    bitmap->TransparentColor = bitmap->Canvas->Pixels[50][50];
    Form1->Canvas->Draw( 0, 0, bitmap);
    bitmap->TransparentMode = tmAuto;
    Form1->Canvas->Draw(50, 50, bitmap);
  }
  catch (...)
  {
    ShowMessage("Could not load or display bitmap");
  }
  delete bitmap;
```

Delphi Examples:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitMap;
```

```
begin
  Bitmap := TBitmap.Create;
  try
    with Bitmap do begin
      LoadFromFile('factory.bmp');
      Transparent := True;
      TransParentColor := BitMap.canvas.pixels[50,50];
      Form1.Canvas.Draw(0,0,BitMap);
      TransparentMode := tmAuto;
      Form1.Canvas.Draw(50,50,BitMap);
    end;
  finally
    Bitmap.Free;
  end;
end;
```

See Also

TCanvas (see page 14441)

Draw (see page 14454)

StretchDraw (see page 14472)

73.1.3.5.2 TBitmap.Handle Property

Provides access to the Windows GDI bitmap handle for accessing the GDI bitmap object.

Pascal

```
property Handle: HBITMAP;
```

C++

```
__property HBITMAP Handle;
```

Description

Use Handle to call a Windows API function that requires the handle of a bitmap object. Pass Handle as the bitmap handle parameter to these functions.

Handle is the HBITMAP encapsulated by the bitmap object. Avoid grabbing the handle directly since it causes the HBITMAP to be copied if more than one TBitmap (see page 14396) shares the handle.

Warning: Be careful when giving the handle to an object or routine. If the receiver takes ownership (and destroys) the bitmap handle, call TBitmap (see page 14396).::ReleaseHandle (see page 14413).

See Also

HandleType (see page 14420)

MaskHandle (see page 14421)

HandleAllocated (see page 14407)

73.1.3.5.3 TBitmap.HandleType Property

Indicates whether the bitmap is a device-dependent bitmap, or a device-independent bitmap.

Pascal

```
property HandleType: TBitmapHandleType;
```

C++

```
__property TBitmapHandleType HandleType;
```

Description

HandleType is used to find or modify the type of the bitmap. Device-dependent bitmaps (DDBs) may be faster to draw on the screen, but may be stored in memory in a device-specific format. Device-independent bitmaps (DIBs) are stored in memory in a standard format supported by all devices (including printers).

See Also

Handle (see page 14420)

73.1.3.5.4 TBitmap.IgnorePalette Property

Determines whether the bitmap realizes its palette when drawing its image.

Pascal

```
property IgnorePalette: Boolean;
```

C++

```
__property Boolean IgnorePalette;
```

Description

Use IgnorePalette when speed of drawing is a priority. When IgnorePalette is true, the bitmap does not realize its palette when drawing itself, resulting in lower picture quality on 256-color video drivers, but faster drawing of the bitmap image.

Note:

See Also

TBitmapImage (see page 14429)

TDBCtrlGrid (see page 8168)

TDBImage (see page 8813)

73.1.3.5.5 TBitmap.MaskHandle Property

Provides access to the Windows GDI bitmap handle for accessing the GDI bitmap object.

Pascal

```
property MaskHandle: HBITMAP;
```

C++

```
__property HBITMAP MaskHandle;
```

Description

Use MaskHandle to call a Windows API function that requires the handle of a bitmap object. Pass MaskHandle as the bitmap handle parameter to these functions. MaskHandle is the HBITMAP encapsulated by the bitmap object's mask.

See Also

Handle (see page 14420)

ReleaseMaskHandle (see page 14414)

73.1.3.5.6 TBitmap.Monochrome Property

Determines whether the bitmap displays its image in monochrome.

Pascal

```
property Monochrome: Boolean;
```

C++

```
__property Boolean Monochrome;
```

Description

If Monochrome is true if the bitmap displays as a monochrome bitmap. Monochrome is false if the bitmap displays in color, that is, if it contains more than one bit per pixel.

Note:

C++ Examples:

```
/*
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
Note: For the C++ Builder, capitalization matters for the file
name and path.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
  Graphics::TBitmap *Bitmap2 = new Graphics::TBitmap();
  try
  {
    Bitmap1->LoadFromFile("..\\FACTORY.BMP");
    Bitmap2->Assign(Bitmap1);     // Copy Bitmap1 into Bitmap2
    Bitmap2->Dormant();           // Free up GDI resources
    Bitmap2->FreeImage();         // Free up Memory.
    Canvas->Draw(20,20,Bitmap2);  // Note that previous calls don't lose the image
    Bitmap2->Monochrome = true;
    Canvas->Draw(80,80,Bitmap2);
    Bitmap2->ReleaseHandle();       // This will actually lose the bitmap;
  }
  catch (...)
  {
    MessageBeep(0);
  }
  delete Bitmap1;
  delete Bitmap2;
}
```

Delphi Examples:

```
{
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
}
procedure TForm1.Button1Click(Sender: TObject);
var
 BitMap1,BitMap2 : TBitMap;
 MyFormat : Word;
begin
   BitMap2 := TBitMap.Create;
   BitMap1 := TBitMap.Create;
try
   BitMap1.LoadFromFile('factory.bmp');
   BitMap2.Assign(BitMap1);     // Copy BitMap1 into BitMap2
```

```
    BitMap2.Dormant;             // Free up GDI resources
    BitMap2.FreeImage;           // Free up Memory.
    Canvas.Draw(20,20,BitMap2);  // Note that previous calls don't lose the image
    BitMap2.Monochrome := true;
    Canvas.Draw(80,80,BitMap2);
    BitMap2.ReleaseHandle;       // This will actually lose the bitmap;
  finally
    BitMap1.Free;
    BitMap2.Free;
  end;
end;
```

73.1.3.5.7 TBitmap.PixelFormat Property

Indicates the bit format of the bitmap image, specifying how the image is displayed and how the pixels of the bitmap image are stored in memory.

Pascal

```
property PixelFormat: TPixelFormat;
```

C++

```
__property TPixelFormat PixelFormat;
```

Description

Use PixelFormat to change a TBitmap (see page 14396)'s internal image to a particular memory format and color depth, or to find out what memory format and color depth a TBitmap (see page 14396) is using.

For example, PixelFormat can be used to set the pixel format of the bitmap image to 8-bit for video drivers that cannot display the native format of a bitmap image.

Note: The PixelFormat of a JPEG image object applies to the bitmap if the JPEG image is copied to it.

Changing the pixel format is most commonly used with ScanLine (see page 14425), because your code must decode the pixel data accessed by ScanLine (see page 14425). Image-editing tools usually use one pixel for all internal image operations and copy the results to the screen (in whatever format) as the last step.

C++ Examples:

```
/*
This example shows how to draw directly to a Bitmap.  It
loads a bitmap from a file and then copies it to another
bitmap twice it's size.  Then the two bitmaps are
displayed on the form canvas.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *Bitmap = new Graphics::TBitmap();
  Graphics::TBitmap *BigBitmap = new Graphics::TBitmap();
  TRGBTriple *ptr, *bigPtr;   // use a (byte *) for pf8bit color
  TPixelFormat pixForm, bigpixForm;
  try
  {
    Bitmap->LoadFromFile("../littlefac.bmp");
    pixForm = Bitmap->PixelFormat;
    bigpixForm  = BigBitmap->PixelFormat;
    Bitmap->PixelFormat = pf24bit;
    BigBitmap->PixelFormat = pf24bit;
    BigBitmap->Height = Bitmap->Height * 2;
    BigBitmap->Width = Bitmap->Width * 2;
    for (int y = 0; y < Bitmap->Height; y++)
    {
      ptr = (TRGBTriple *)Bitmap->ScanLine[y];
      for (int x = 0; x < Bitmap->Width; x++)
```

```
        {
          int bx = x * 2;
          int by = y * 2;
          bigPtr = (TRGBTriple *)BigBitmap->ScanLine[by];
          bigPtr[bx] = ptr[x];
          bigPtr[bx + 1] = ptr[x];
          bigPtr = (TRGBTriple *)BigBitmap->ScanLine[by + 1];
          bigPtr[bx] = ptr[x];
          bigPtr[bx + 1] = ptr[x];
        }
      }
      Canvas->Draw(0, 0, Bitmap);
      Canvas->Draw(200, 200, BigBitmap);
    }
    catch (...)
    {
      ShowMessage("Could not load or alter bitmap");
    }
    delete Bitmap;
    delete BigBitmap;
  }
```

Delphi Examples:

```
{
This example shows how to draw directly to a Bitmap.  It
loads a bitmap from a file and then copies it to another
bitmap twice it's size.  Then the two bitmaps are
displayed on the form canvas.
}
procedure TForm1.Button1Click(Sender: TObject);
type
  TRGBTripleArray = ARRAY[Word] of TRGBTriple;
  pRGBTripleArray = ^TRGBTripleArray; // use a PByteArray for pf8bit color
var
  x,y : Integer;
  bx, by : Integer;
  BitMap, BigBitMap : TBitMap;
  P, bigP : pRGBTripleArray;
  pixForm, bigpixForm : TPixelFormat;
begin
  BitMap := TBitMap.create;
  BigBitMap := TBitMap.create;
  try
    BitMap.LoadFromFile('littlefac.bmp');
    pixForm := BitMap.PixelFormat;
    bigpixForm := BigBitMap.PixelFormat;
    BitMap.PixelFormat := pf24bit;
    BigBitMap.PixelFormat := pf24bit;
    BigBitMap.Height := BitMap.Height * 2;
    BigBitMap.Width := BitMap.Width * 2;
    for y := 0 to BitMap.Height - 1 do
    begin
      P := BitMap.ScanLine[y];
      for x := 0 to BitMap.Width - 1 do
      begin
        bx := x * 2;
        by := y * 2;
        bigP := BigBitMap.ScanLine[by];
        bigP[bx] := P[x];
        bigP[bx + 1] := P[x];
        bigP := BigBitMap.ScanLine[by + 1];
        bigP[bx] := P[x];
        bigP[bx + 1] := P[x];
      end;
    end;
    Canvas.Draw(0, 0, BitMap);
```

```
    Canvas.Draw(200, 200, BigBitMap);
  finally
    BitMap.Free;
    BigBitMap.Free;
  end;
end;
```

See Also

PixelFormat (see page 15703)

ScanLine (see page 14425)

HandleType (see page 14420)

73.1.3.5.8 TBitmap.ScanLine Property

Provides indexed access to each line of pixels.

Pascal

```
property ScanLine [Row: Integer]: Pointer;
```

C++

```
__property void * ScanLine[int Row];
```

Description

ScanLine is used only with DIBs (Device Independent Bitmaps) for image editing tools that do low-level pixel work.

C++ Examples:

```
/*
This example shows how to draw directly to a Bitmap.  It
loads a bitmap from a file and then copies it to another
bitmap twice it's size.  Then the two bitmaps are
displayed on the form canvas.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *Bitmap = new Graphics::TBitmap();
  Graphics::TBitmap *BigBitmap = new Graphics::TBitmap();
  TRGBTriple *ptr, *bigPtr;   // use a (byte *) for pf8bit color
  TPixelFormat pixForm, bigpixForm;
  try
  {
    Bitmap->LoadFromFile("../littlefac.bmp");
    pixForm = Bitmap->PixelFormat;
    bigpixForm  = BigBitmap->PixelFormat;
    Bitmap->PixelFormat = pf24bit;
    BigBitmap->PixelFormat = pf24bit;
    BigBitmap->Height = Bitmap->Height * 2;
    BigBitmap->Width = Bitmap->Width * 2;
    for (int y = 0; y < Bitmap->Height; y++)
    {
      ptr = (TRGBTriple *)Bitmap->ScanLine[y];
      for (int x = 0; x < Bitmap->Width; x++)
      {
        int bx = x * 2;
        int by = y * 2;
        bigPtr = (TRGBTriple *)BigBitmap->ScanLine[by];
        bigPtr[bx] = ptr[x];
        bigPtr[bx + 1] = ptr[x];
        bigPtr = (TRGBTriple *)BigBitmap->ScanLine[by + 1];
        bigPtr[bx] = ptr[x];
        bigPtr[bx + 1] = ptr[x];
```

```
      }
    }
    Canvas->Draw(0, 0, Bitmap);
    Canvas->Draw(200, 200, BigBitmap);
  }
  catch (...)
  {
    ShowMessage("Could not load or alter bitmap");
  }
  delete Bitmap;
  delete BigBitmap;
}
```

Delphi Examples:

```
{
This example shows how to draw directly to a Bitmap.  It
loads a bitmap from a file and then copies it to another
bitmap twice it's size.  Then the two bitmaps are
displayed on the form canvas.
}
procedure TForm1.Button1Click(Sender: TObject);
type
  TRGBTripleArray = ARRAY[Word] of TRGBTriple;
  pRGBTripleArray = ^TRGBTripleArray; // use a PByteArray for pf8bit color
var
  x,y : Integer;
  bx, by : Integer;
  BitMap, BigBitMap : TBitMap;
  P, bigP : pRGBTripleArray;
  pixForm, bigpixForm : TPixelFormat;
begin
  BitMap := TBitMap.create;
  BigBitMap := TBitMap.create;
  try
    BitMap.LoadFromFile('littlefac.bmp');
    pixForm := BitMap.PixelFormat;
    bigpixForm := BigBitMap.PixelFormat;
    BitMap.PixelFormat := pf24bit;
    BigBitMap.PixelFormat := pf24bit;
    BigBitMap.Height := BitMap.Height * 2;
    BigBitMap.Width := BitMap.Width * 2;
    for y := 0 to BitMap.Height - 1 do
    begin
      P := BitMap.ScanLine[y];
      for x := 0 to BitMap.Width - 1 do
      begin
        bx := x * 2;
        by := y * 2;
        bigP := BigBitMap.ScanLine[by];
        bigP[bx] := P[x];
        bigP[bx + 1] := P[x];
        bigP := BigBitMap.ScanLine[by + 1];
        bigP[bx] := P[x];
        bigP[bx + 1] := P[x];
      end;
    end;
    Canvas.Draw(0, 0, BitMap);
    Canvas.Draw(200, 200, BigBitMap);
  finally
    BitMap.Free;
    BigBitMap.Free;
  end;
end;
```

See Also

PixelFormat (see page 14423)

73.1.3.5.9 TBitmap.TransparentColor Property

Determines which color of the bitmap is to be transparent when the bitmap is drawn.

Pascal

```
property TransparentColor: TColor;
```

C++

```
__property TColor TransparentColor;
```

Description

Use TransparentColor to determine how to draw the bitmap transparently. When the TransparentMode (see page 14428) property is set to tmAuto (default), TransparentColor returns the color of the first pixel in the bitmap image data. For "bottom-up" bitmaps, the first pixel is the bottom leftmost pixel shown onscreen. For "top-down" bitmaps (less common), the first pixel is in the top left corner shown onscreen.

If TransparentColor is assigned, the TransparentMode (see page 14428) is automatically set to tmFixed so that the new transparent color can be used later. If you want TransparentColor to disregard any assignments and return the bottom leftmost pixel color again, set TransparentMode (see page 14428) to tmAuto.

C++ Examples:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *bitmap = new Graphics::TBitmap();
  try
  {
    bitmap->LoadFromFile("..\\FACTORY.BMP ");
    bitmap->Transparent = true;
    bitmap->TransparentColor = bitmap->Canvas->Pixels[50][50];
    Form1->Canvas->Draw( 0, 0, bitmap);
    bitmap->TransparentMode = tmAuto;
    Form1->Canvas->Draw(50, 50, bitmap);
  }
  catch (...)
  {
    ShowMessage("Could not load or display bitmap");
  }
  delete bitmap;
```

Delphi Examples:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitMap;
begin
  Bitmap := TBitmap.Create;
  try
    with Bitmap do begin
      LoadFromFile('factory.bmp');
      Transparent := True;
      TransParentColor := BitMap.canvas.pixels[50,50];
      Form1.Canvas.Draw(0,0,BitMap);
      TransparentMode := tmAuto;
      Form1.Canvas.Draw(50,50,BitMap);
    end;
  finally
    Bitmap.Free;
```

```
    end;
  end;
```

See Also

TransparentMode (see page 14428)

73.1.3.5.10 TBitmap.TransparentMode Property

Determines whether the TransparentColor (see page 14427) property's value is automatically calculated or stored with the bitmap object.

Pascal

```
property TransparentMode: TTransparentMode;
```

C++

```
__property TTransparentMode TransparentMode;
```

Description

When TransparentMode is set to tmAuto (the default), the TransparentColor (see page 14427) property returns the color of the bottom-leftmost pixel of the bitmap image. When TransparentMode is set to tmFixed, the TransparentColor (see page 14427) property refers to the color stored in the bitmap object.

C++ Examples:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *bitmap = new Graphics::TBitmap();
  try
  {
    bitmap->LoadFromFile("..\\FACTORY.BMP ");
    bitmap->Transparent = true;
    bitmap->TransparentColor = bitmap->Canvas->Pixels[50][50];
    Form1->Canvas->Draw( 0, 0, bitmap);
    bitmap->TransparentMode = tmAuto;
    Form1->Canvas->Draw(50, 50, bitmap);
  }
  catch (...)
  {
    ShowMessage("Could not load or display bitmap");
  }
  delete bitmap;
```

Delphi Examples:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitMap;
begin
  Bitmap := TBitmap.Create;
  try
    with Bitmap do begin
      LoadFromFile('factory.bmp');
      Transparent := True;
      TransParentColor := BitMap.canvas.pixels[50,50];
      Form1.Canvas.Draw(0,0,BitMap);
      TransparentMode := tmAuto;
      Form1.Canvas.Draw(50,50,BitMap);
    end;
  finally
    Bitmap.Free;
  end;
end;
```

See Also

TransparentColor (see page 14427)

73.1.4 TBitmapImage Class

TBitmapImage is the internal representation of the bitmap image for a TBitmap (see page 14396) object.

Class Hierarchy

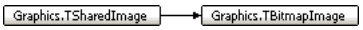

Pascal

```
TBitmapImage = class(TSharedImage);
```

C++

```
class TBitmapImage : public TSharedImage;
```

File

Graphics

Description

TBitmapImage is used only for internal implementation of the TBitmap (see page 14396) component that is defined in the Graphics (see page 14387) unit. It represents the internal image of the bitmap. All of the data and methods introduced in TBitmapImage are private. They contain information about the specific Windows bitmap resource, including pointers to the standard HBITMAP structures defined by Windows. TBitmapImage is referenced in the constructor of the TBitmap (see page 14396) object when the bitmap image is created.

Note: TBitmapImage is how TBitmap (see page 14396) does reference counting and handle sharing. Multiple instances can refer to the same TBitmap (see page 14396) image. TBitmapImage is the actual owner of the bitmap handle.

See Also

TBitmap (see page 14396)

73.1.4.1 TBitmapImage Members

The following tables list the members exposed by TBitmapImage.

Protected Methods

	Name	Description
V	FreeHandle (see page 14430)	This is FreeHandle, a member of class TBitmapImage.
	Reference (see page 14604)	This is Reference, a member of class TSharedImage.
	Release (see page 14604)	This is Release, a member of class TSharedImage.

Protected Properties

	Name	Description
R	RefCount (see page 14605)	This is RefCount, a member of class TSharedImage.

Public Methods

	Name	Description
V	Destroy (see page 14430)	Frees the memory associated with the TBitmapImage (see page 14429) object.

73.1.4.2 TBitmapImage.Destroy Destructor

Frees the memory associated with the TBitmapImage (see page 14429) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBitmapImage();
```

Description

Do not call Destroy. The bitmap object automatically destroys the bitmap image.

See Also

Free (see page 22300)

delete

73.1.4.3 TBitmapImage Methods

The methods of the TBitmapImage class are listed here.

Protected Methods

	Name	Description
	FreeHandle (see page 14430)	This is FreeHandle, a member of class TBitmapImage.

73.1.4.3.1 TBitmapImage.FreeHandle Method

Pascal

```
procedure FreeHandle; override;
```

C++

```
virtual __fastcall FreeHandle();
```

Description

This is FreeHandle, a member of class TBitmapImage.

73.1.5 TBrush Class

TBrush represents the color and pattern used to fill solid shapes.

Class Hierarchy

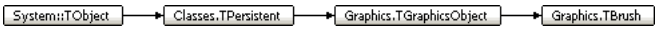

Pascal

```
TBrush = class(TGraphicsObject);
```

C++

```
class TBrush : public TGraphicsObject;
```

File

Graphics

Description

TBrush encapsulates the Windows brush object (HBRUSH) and is used to fill solid shapes, such as rectangles and ellipses, with a color or pattern.

See Also

TCanvas (see page 14441)

TPen (see page 14571)

Shapes

Using the Main VCL

73.1.5.1 TBrush Members

The following tables list the members exposed by TBrush.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 14534)	Generates an OnChange (see page 14536) event when the properties of the graphics object change.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetBitmap (see page 14434)	This is GetBitmap, a member of class TBrush.
	GetColor (see page 14434)	This is GetColor, a member of class TBrush.
	GetHandle (see page 14434)	This is GetHandle, a member of class TBrush.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetStyle (see page 14434)	This is GetStyle, a member of class TBrush.
	Lock (see page 14534)	Blocks other execution threads from using the associated canvas until the Unlock (see page 14535) method is called.
	SetBitmap (see page 14434)	This is SetBitmap, a member of class TBrush.
	SetColor (see page 14435)	This is SetColor, a member of class TBrush.
	SetHandle (see page 14435)	This is SetHandle, a member of class TBrush.
	SetStyle (see page 14435)	This is SetStyle, a member of class TBrush.
	Unlock (see page 14535)	Releases the lock on execution threads set by the Lock (see page 14534) method.

Public Events

	Name	Description
	OnChange (see page 14536)	Occurs immediately after the graphics object changes.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 14433)	Copies the contents of the brush to another brush.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 14432)	Creates and initializes a TBrush (see page 14430) instance.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 14433)	Destroys the TBrush (see page 14430) instance and frees its memory.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	HandleAllocated (see page 14534)	Indicates whether the TGraphicsObject (see page 14532) object has acquired a handle to the underlying GDI object.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Bitmap (see page 14436)	Specifies an external bitmap image that defines a pattern for the brush.
	Handle (see page 14438)	Provides access to the handle of a system brush.
	OwnerCriticalSection (see page 14536)	Points to information about the critical section that the canvas which draws with this graphics object uses to lock out other threads.

Published Properties

	Name	Description
	Color (see page 14437)	Indicates the color of the brush.
	Style (see page 14438)	Specifies the pattern for the brush.

73.1.5.2 TBrush.Create Constructor

Creates and initializes a TBrush (see page 14430) instance.

Pascal

```
constructor Create;
```

C++

```
__fastcall TBrush();
```

Description

Create generates a new TBrush (see page 14430) instance.

73.1.5.3 TBrush.Destroy Destructor

Destroys the TBrush (see page 14430) instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBrush();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free verifies that the TBrush (see page 14430) instance is not nil before calling Destroy.

See Also

Free (see page 22300)

73.1.5.4 TBrush Methods

The methods of the TBrush class are listed here.

Protected Methods

	Name	Description
	GetBitmap (see page 14434)	This is GetBitmap, a member of class TBrush.
	GetColor (see page 14434)	This is GetColor, a member of class TBrush.
	GetHandle (see page 14434)	This is GetHandle, a member of class TBrush.
	GetStyle (see page 14434)	This is GetStyle, a member of class TBrush.
	SetBitmap (see page 14434)	This is SetBitmap, a member of class TBrush.
	SetColor (see page 14435)	This is SetColor, a member of class TBrush.
	SetHandle (see page 14435)	This is SetHandle, a member of class TBrush.
	SetStyle (see page 14435)	This is SetStyle, a member of class TBrush.

Public Methods

	Name	Description
	Assign (see page 14433)	Copies the contents of the brush to another brush.

73.1.5.4.1 TBrush.Assign Method

Copies the contents of the brush to another brush.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Assign copies the brush's properties from another TBrush (see page 14430) instance.

See Also

Assign (see page 2930)

73.1.5.4.2 TBrush.GetBitmap Method

Pascal

```
function GetBitmap: TBitmap;
```

C++

```
__fastcall TBitmap GetBitmap();
```

Description

This is GetBitmap, a member of class TBrush.

73.1.5.4.3 TBrush.GetColor Method

Pascal

```
function GetColor: TColor;
```

C++

```
__fastcall TColor GetColor();
```

Description

This is GetColor, a member of class TBrush.

73.1.5.4.4 TBrush.GetHandle Method

Pascal

```
function GetHandle: HBrush;
```

C++

```
__fastcall HBrush GetHandle();
```

Description

This is GetHandle, a member of class TBrush.

73.1.5.4.5 TBrush.GetStyle Method

Pascal

```
function GetStyle: TBrushStyle;
```

C++

```
__fastcall TBrushStyle GetStyle();
```

Description

This is GetStyle, a member of class TBrush.

73.1.5.4.6 TBrush.SetBitmap Method

Pascal

```
procedure SetBitmap(Value: TBitmap);
```

C++

```
__fastcall SetBitmap(TBitmap Value);
```

Description

This is SetBitmap, a member of class TBrush.

73.1.5.4.7 TBrush.SetColor Method

Pascal

```
procedure SetColor(Value: TColor);
```

C++

```
__fastcall SetColor(TColor Value);
```

Description

This is SetColor, a member of class TBrush.

73.1.5.4.8 TBrush.SetHandle Method

Pascal

```
procedure SetHandle(Value: HBrush);
```

C++

```
__fastcall SetHandle(HBrush Value);
```

Description

This is SetHandle, a member of class TBrush.

73.1.5.4.9 TBrush.SetStyle Method

Pascal

```
procedure SetStyle(Value: TBrushStyle);
```

C++

```
__fastcall SetStyle(TBrushStyle Value);
```

Description

This is SetStyle, a member of class TBrush.

73.1.5.5 TBrush Properties

The properties of the TBrush class are listed here.

Public Properties

	Name	Description
	Bitmap (see page 14436)	Specifies an external bitmap image that defines a pattern for the brush.
	Handle (see page 14438)	Provides access to the handle of a system brush.

Published Properties

	Name	Description
	Color (see page 14437)	Indicates the color of the brush.
	Style (see page 14438)	Specifies the pattern for the brush.

73.1.5.5.1 TBrush.Bitmap Property

Specifies an external bitmap image that defines a pattern for the brush.

Pascal

```
property Bitmap: TBitmap;
```

C++

```
__property TBitmap Bitmap;
```

Description

Bitmap points to a TBitmap (see page 14396) object that holds a bitmap image. If Bitmap is nonempty, the bitmap image (rather than the Style (see page 14438) property) defines the brush's pattern. If the image is larger than eight pixels by eight pixels, only the top left eight-by-eight region is used.

Changing the image does not affect the brush until the TBitmap (see page 14396) is reassigned to the Bitmap property. Be sure to free the TBitmap (see page 14396) after finishing with the brush, since TBrush (see page 14430) will not free it.

C++ Examples:

```
/*
The following code loads a bitmap from a file and assigns it
to the Brush of the Canvas of Form1:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *BrushBmp = new Graphics::TBitmap;
  try
  {
    BrushBmp->LoadFromFile("../bm1.BMP");
    Form1->Canvas->Brush->Bitmap = BrushBmp;
    Form1->Canvas->FillRect(Rect(0,0,100,100));
  }
  __finally
  {
    Form1->Canvas->Brush->Bitmap = NULL;
    delete BrushBmp;
  }
}
```

Delphi Examples:

```
{
The following code loads a bitmap from a file and assigns it
to the Brush of the Canvas of Form1:
}
var
  Bitmap: TBitmap;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Bitmap := TBitmap.Create;
  try
    Bitmap.LoadFromFile('bm1.BMP');
    Form1.Canvas.Brush.Bitmap := Bitmap;
    Form1.Canvas.FillRect(Rect(0,0,100,100));
  finally
    Form1.Canvas.Brush.Bitmap := nil;
    Bitmap.Free;
  end;
end;
```

See Also

TBitmap (see page 14396)

Style (see page 14438)

73.1.5.5.2 TBrush.Color Property

Indicates the color of the brush.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

The Color property determines the color of the brush. This is the color that is used to draw the pattern represented by the Style (see page 14438) property, not the background color of the brush (unless Style (see page 14438) is bsSolid).

Note: If the value of the Style (see page 14438) property is bsClear, the Color property is ignored. Further, any value assigned to the Color property is lost when Style (see page 14438) is set to bsClear.

C++ Examples:

```
/*
The following example paints a cross-hatched ellipse onto
Image1 which is a TImage on the form when the user clicks on
Button1.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TCanvas *pCanvas = Image1->Canvas;
  pCanvas->Brush->Color = clRed;
  pCanvas->Brush->Style = bsDiagCross;
  pCanvas->Ellipse(0, 0, Image1->Width, Image1->Height);
}
```

Delphi Examples:

```
{
The following example paints a cross-hatched ellipse onto
Image1 which is a TImage on the form when the user clicks on
Button1.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Image1 do begin
    Canvas.Brush.Color := clRed;
    Canvas.Brush.Style := bsDiagCross;
    Canvas.Ellipse(0, 0, Image1.Width, Image1.Height);
  end;
end;
```

See Also

Style (see page 14438)

73.1.5.5.3 TBrush.Handle Property

Provides access to the handle of a system brush.

Pascal

```
property Handle: HBrush;
```

C++

```
__property HBrush Handle;
```

Description

The Handle property provides access to the Windows GDI object handle. TBrush (see page 14430) encapsulates the Windows brush object (HBRUSH) and is used to fill solid shapes, such as rectangles and ellipses, with a color or pattern.

73.1.5.5.4 TBrush.Style Property

Specifies the pattern for the brush.

Pascal

```
property Style: TBrushStyle;
```

C++

```
__property TBrushStyle Style;
```

Description

The Style property determines the pattern painted by the brush, unless a value is assigned to Bitmap (see page 14436).

Tip: Set the Style property to bsClear to eliminate flicker when the object repaints.

C++ Examples:

```
/*
The following example paints a cross-hatched ellipse onto
Image1 which is a TImage on the form when the user clicks on
Button1.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TCanvas *pCanvas = Image1->Canvas;
  pCanvas->Brush->Color = clRed;
  pCanvas->Brush->Style = bsDiagCross;
  pCanvas->Ellipse(0, 0, Image1->Width, Image1->Height);
}
```

Delphi Examples:

```
{
The following example paints a cross-hatched ellipse onto
Image1 which is a TImage on the form when the user clicks on
Button1.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Image1 do begin
    Canvas.Brush.Color := clRed;
    Canvas.Brush.Style := bsDiagCross;
    Canvas.Ellipse(0, 0, Image1.Width, Image1.Height);
  end;
end;
```

See Also

Bitmap (see page 14436)

Color (see page 14437)

73.1.6 TBrushRecall Class

TBrushRecall saves and restores a TBrush (see page 14430) object.

Class Hierarchy

Pascal

```
TBrushRecall = class(TRecall);
```

C++

```
class TBrushRecall : public TRecall;
```

File

Graphics

Description

Use TBrushRecall to store the current state of a TBrush (see page 14430) object. You are then free to change the current state of the brush object. To restore the brush object to its original state, all that is necessary is to free the TBrushRecall object and the brush object is automatically restored to the saved property values.

You can update the TBrushRecall instance to reflect the current properties of the TBrush (see page 14430) object by calling the Store method. You can prevent the TBrushRecall destructor from updating the TBrush (see page 14430) object by calling the Forget method.

See Also

TRecall (see page 2964)

TBrush (see page 14430)

Using Brushes

73.1.6.1 TBrushRecall Members

The following tables list the members exposed by TBrushRecall.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 14440)	Instantiates a TBrushRecall (see page 14439) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2966)	Destroys the TRecall (see page 2964) instance, assigning the stored properties to the reference object.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Forget (see page 2967)	Discards the saved property values and prevents the TRecall (see page 2964) object from updating the reference object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Store (see page 2967)	Saves the current property settings of the reference object.

Public Properties

	Name	Description
R	Reference (see page 2968)	Indicates the reference object that is updated when you destroy the TRecall (see page 2964) instance.

73.1.6.2 TBrushRecall.Create Constructor

Instantiates a TBrushRecall (see page 14439) object.

Pascal

```
constructor Create(ABrush: TBrush);
```

C++

```
__fastcall TBrushRecall(TBrush ABrush);
```

Description

Call Create to instantiate a brush recall object at runtime. Create saves the current properties of the TBrush (see page 14430) object specified by ABrush, which becomes the value of the Reference property.

Subsequent calls to the Store method take snapshots of the properties of ABrush and save them, replacing the values saved by this constructor. When you destroy the TBrushRecall (see page 14439) instance, the most recently saved properties are assigned to ABrush, restoring it to a previous state.

See Also

Destroy (see page 2966)

Reference (see page 2968)

Store (see page 2967)

73.1.7 TCanvas Class

TCanvas provides an abstract drawing space for objects that must render their own images.

Class Hierarchy

Pascal

```
TCanvas = class(TPersistent);
```

C++

```
class TCanvas : public TPersistent;
```

File

Graphics

Description

Use TCanvas as a drawing surface for objects that draw an image of themselves. Standard window controls such as edit controls or list boxes do not require a canvas, as they are drawn by the system.

TCanvas provides properties, events and methods that assist in creating an image by

Specifying the type of brush, pen and font to use.

Drawing and filling a variety of shapes and lines.

Writing text.

Rendering graphic images.

Enabling a response to changes in the current image.

TCanvas has two descendants, TControlCanvas and TMetafileCanvas (see page 14564), which assist in drawing images of controls and in creating metafile images for objects.

See Also

TControlCanvas (see page 5903)

TMetafileCanvas (see page 14564)

73.1.7.1 TCanvas Members

The following tables list the members exposed by TCanvas.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 14450)	This is Changed, a member of class TCanvas.
	Changing (see page 14450)	This is Changing, a member of class TCanvas.
	CreateHandle (see page 14454)	This is CreateHandle, a member of class TCanvas.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	GetOwner (see page 2932)	Returns the owner of an object.
	RequiredState (see page 14470)	This is RequiredState, a member of class TCanvas.

Public Events

	Name	Description
	OnChange (see page 14496)	Occurs when the image has just changed.
	OnChanging (see page 14496)	Occurs just before a change is made to the image.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Arc (see page 14448)	Draws an arc on the image along the perimeter of the ellipse bounded by the specified rectangle.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BrushCopy (see page 14449)	Copies a portion of a bitmap onto a rectangle on the canvas, replacing one of the colors of the bitmap with the brush of the canvas.
	Chord (see page 14451)	Draws a closed figure represented by the intersection of a line and an ellipse.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CopyRect (see page 14452)	Copies part of an image from another canvas into the canvas.
	Create (see page 14443)	Creates an instance of TCanvas (see page 14441).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 14446)	Destroys an instance of TCanvas (see page 14441).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Draw (see page 14454)	Renders the graphic specified by the Graphic parameter on the canvas at the location given by the coordinates (X, Y).
	DrawFocusRect (see page 14455)	Draws a rectangle in the style used to indicate that the object inside the rectangle has focus.
	Ellipse (see page 14457)	This is the overview for the Ellipse method overload.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FillRect (see page 14458)	Fills the specified rectangle on the canvas using the current brush.
	FloodFill (see page 14459)	Fills an area of the canvas using the current brush.
	FrameRect (see page 14460)	Draws a rectangle using the Brush (see page 14484) of the canvas to draw the border.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	HandleAllocated (see page 14461)	Indicates whether the TCanvas (see page 14441) object has acquired a handle to a device context.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LineTo (see page 14462)	Draws a line on the canvas from PenPos (see page 14494) to the point specified by X and Y, and sets the pen position to (X, Y).

	Lock (see page 14463)	Prevents other threads from drawing on the canvas.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveTo (see page 14463)	Changes the current drawing position to the point (X,Y).
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Pie (see page 14464)	Draws a pie-shaped the section of the ellipse bounded by the rectangle (X1, Y1) and (X2, Y2) on the canvas.
	PolyBezier (see page 14464)	Draws a set of Bezier curves.
	PolyBezierTo (see page 14465)	Draws a set of Bezier curves and updates the value of PenPos (see page 14494).
	Polygon (see page 14466)	Draws a series of lines on the canvas connecting the points passed in and closing the shape by drawing a line from the last point to the first point.
	Polyline (see page 14467)	Draws a series of lines on the canvas with the current pen, connecting each of the points passed to it in Points.
	Rectangle (see page 14468)	This is the overview for the Rectangle method overload.
	Refresh (see page 14470)	Deselects the Pen (see page 14492), Brush (see page 14484), and Font (see page 14488) from the device context.
	RoundRect (see page 14471)	Draws a rectangle with rounded corners on the canvas.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	StretchDraw (see page 14472)	Draws the graphic specified by the Graphic parameter in the rectangle specified by the Rect parameter.
	TextExtent (see page 14474)	Returns the width and height, in pixels, of a string rendered in the current font.
	TextHeight (see page 14474)	Returns the height, in pixels, of a string rendered in the current font.
	TextOut (see page 14475)	Writes a string on the canvas, starting at the point (X,Y), and then updates the PenPos (see page 14494) to the end of the string.
	TextRect (see page 14480)	This is the overview for the TextRect method overload.
	TextWidth (see page 14481)	Returns the width, in pixels, of a string rendered in the current font.
	TryLock (see page 14483)	Locks the canvas if it is currently unlocked.
	Unlock (see page 14483)	Decreases the LockCount (see page 14492) property, allowing other threads to execute when LockCount (see page 14492) reaches 0.

Public Properties

	Name	Description
R	CanvasOrientation (see page 14485)	Determines the orientation of the canvas as left-to-right or right-to-left.
R	ClipRect (see page 14486)	Read-only property that specifies the boundaries of the clipping rectangle.
	Handle (see page 14489)	Specifies the handle for this canvas.
R	LockCount (see page 14492)	Indicates the number of times the canvas has been locked to prevent interference from other threads.
	PenPos (see page 14494)	Specifies the current drawing position of the Pen (see page 14492).
	Pixels (see page 14494)	Specifies the color of the pixels within the current ClipRect (see page 14486).
	TextFlags (see page 14495)	Specifies how text is written to the canvas.

Published Properties

	Name	Description
	Brush (see page 14484)	Determines the color and pattern for filling graphical shapes and backgrounds.
	CopyMode (see page 14487)	Specifies how a graphical image is copied onto the canvas.
	Font (see page 14488)	Specifies the font to use when writing text on the image.
	Pen (see page 14492)	Specifies the kind of pen the canvas uses for drawing lines and outlining shapes.

73.1.7.2 TCanvas.Create Constructor

Creates an instance of TCanvas (see page 14441).

Pascal

```
constructor Create;
```

C++

```
__fastcall TCanvas();
```

Description

Call Create to instantiate a TCanvas (see page 14441) object at runtime. Create allocates memory for the instance of TCanvas (see page 14441) and creates the TFont (see page 14497), TBrush (see page 14430) and TPen (see page 14571) objects for the Font (see page 14488), Brush (see page 14484), and Pen (see page 14492) properties. Create also initializes the CopyMode (see page 14487) to cmSrcCopy (see page 14739).

C++ Examples:

```
/*
The GetFormImage allows you to easily obtain a bitmap of a
form. The following method can be added to a custom
TWinControl descendant to save itself as a bitmap.  Check the
produced BMP file to confirm.  Click the button and them
double click on the newly created foo.bmp file to view.
*/
class TMyControl : public TColorListBox
{
__published:    // IDE-managed Components
    void __fastcall SaveAsBmp(TFileName fileName);
private:    // User declarations
public:     // User declarations
    __fastcall TMyControl(TComponent* Owner);
};

TForm1 *Form1;
TMyControl *MyControl1;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
__fastcall TMyControl::TMyControl(TComponent* Owner)
    : TColorListBox(Owner)
{
}

typedef struct {
  TLogPalette lpal;
  TPaletteEntry dummy[256];
} LogPal;

void __fastcall TMyControl::SaveAsBmp(TFileName fileName)
{
  TCanvas *tempCanvas = new TCanvas;
  try
  {
    HWND notUsed;
    tempCanvas->Handle = GetDeviceContext(notUsed);
    TImage *image2save = new TImage(Form1);
    try
    {
      image2save->Height = Height;
      image2save->Width = Width;
      TRect destRect = Rect(0,0,Width,Height);
      TRect sourceRect = destRect;
      image2save->Canvas->CopyRect(destRect, tempCanvas, sourceRect);
      LogPal SysPal;
      SysPal.lpal.palVersion = 0x300;
      SysPal.lpal.palNumEntries = 256;
      GetSystemPaletteEntries(tempCanvas->Handle,0,256,SysPal.lpal.palPalEntry);
      image2save->Picture->Bitmap->Palette = CreatePalette((const tagLOGPALETTE
```

```
*)&SysPal.lpal);
      image2save->Picture->SaveToFile(fileName);
    }
    __finally
    {
     delete image2save;
    }
  }
  __finally
  {
    delete tempCanvas;
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  MyControl1->SaveAsBmp("../foo.bmp");
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyControl1 = new TMyControl(Form1);
  MyControl1->Parent = Form1;
  MyControl1->Visible  = true;
}
```

Delphi Examples:

```
{
The GetFormImage allows you to easily obtain a bitmap of a
form. The following method can be added to a custom
TWinControl descendant to save itself as a bitmap.  Check the
produced BMP file to confirm.  Click the button and them
double click on the newly created foo.bmp file to view.
}
type
  TMyControl = class(TColorListBox)
    procedure SaveAsBmp(fileName: TFileName);
  end;
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  MyControl1: TMyControl;
implementation

{$R *.dfm}

type
  LogPal = record
  lpal : TLogPalette;
  dummy:Array[0..255] of TPaletteEntry;
  end;

procedure TMyControl.SaveAsBmp(fileName: TFileName);
var
  Source: TComponent;
  SysPal : LogPal;
  tempCanvas: TCanvas;
```

```
    sourceRect, destRect: TRect;
    image2save: TImage;
    notUsed: HWND;
  begin
    tempCanvas := TCanvas.Create;
    try
      tempCanvas.Handle := GetDeviceContext(notUsed);
      image2save:=TImage.create(self);
      try
        with image2save do
        begin
          Height := Self.Height;
          Width :=  Self.Width;
          destRect := Rect(0,0,Width,Height);
          sourceRect := destRect;
          Canvas.CopyRect(destRect,tempCanvas,sourceRect);
          SysPal.lPal.palVersion:=$300;
          SysPal.lPal.palNumEntries:=256;
          GetSystemPaletteEntries(
            tempCanvas.Handle,0,256,SysPal.lpal.palPalEntry);
          Picture.Bitmap.Palette:= CreatePalette(Syspal.lpal);
        end;
        image2save.Picture.SaveToFile(fileName);
      finally
       image2save.Free;
      end;
    finally
      tempCanvas.Free;
    end;
  end;
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    MyControl1.SaveAsBmp('foo.bmp');
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    MyControl1:= TMyControl.Create(Form1);
    MyControl1.Parent:= Form1;
    MyControl1.visible := true;
  end;
```

See Also

Brush (see page 14484)

CopyMode (see page 14487)

Font (see page 14488)

Pen (see page 14492)

TBrush (see page 14430)

TFont (see page 14497)

TPen (see page 14571)

73.1.7.3 TCanvas.Destroy Destructor

Destroys an instance of TCanvas (see page 14441).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCanvas();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TCanvas (see page 14441) object is not nil and only then calls Destroy.

See Also

Brush (see page 14484)

Font (see page 14488)

Free (see page 22300)

Pen (see page 14492)

TBrush (see page 14430)

TFont (see page 14497)

TPen (see page 14571)

73.1.7.4 TCanvas Methods

The methods of the TCanvas class are listed here.

Protected Methods

	Name	Description
	Changed (see page 14450)	This is Changed, a member of class TCanvas.
	Changing (see page 14450)	This is Changing, a member of class TCanvas.
	CreateHandle (see page 14454)	This is CreateHandle, a member of class TCanvas.
	RequiredState (see page 14470)	This is RequiredState, a member of class TCanvas.

Public Methods

	Name	Description
	Arc (see page 14448)	Draws an arc on the image along the perimeter of the ellipse bounded by the specified rectangle.
	BrushCopy (see page 14449)	Copies a portion of a bitmap onto a rectangle on the canvas, replacing one of the colors of the bitmap with the brush of the canvas.
	Chord (see page 14451)	Draws a closed figure represented by the intersection of a line and an ellipse.
	CopyRect (see page 14452)	Copies part of an image from another canvas into the canvas.
	Draw (see page 14454)	Renders the graphic specified by the Graphic parameter on the canvas at the location given by the coordinates (X, Y).
	DrawFocusRect (see page 14455)	Draws a rectangle in the style used to indicate that the object inside the rectangle has focus.
	Ellipse (see page 14457)	This is the overview for the Ellipse method overload.
	FillRect (see page 14458)	Fills the specified rectangle on the canvas using the current brush.
	FloodFill (see page 14459)	Fills an area of the canvas using the current brush.
	FrameRect (see page 14460)	Draws a rectangle using the Brush (see page 14484) of the canvas to draw the border.
	HandleAllocated (see page 14461)	Indicates whether the TCanvas (see page 14441) object has acquired a handle to a device context.
	LineTo (see page 14462)	Draws a line on the canvas from PenPos (see page 14494) to the point specified by X and Y, and sets the pen position to (X, Y).
	Lock (see page 14463)	Prevents other threads from drawing on the canvas.
	MoveTo (see page 14463)	Changes the current drawing position to the point (X,Y).

	Pie (see page 14464)	Draws a pie-shaped the section of the ellipse bounded by the rectangle (X1, Y1) and (X2, Y2) on the canvas.
	PolyBezier (see page 14464)	Draws a set of Bezier curves.
	PolyBezierTo (see page 14465)	Draws a set of Bezier curves and updates the value of PenPos (see page 14494).
	Polygon (see page 14466)	Draws a series of lines on the canvas connecting the points passed in and closing the shape by drawing a line from the last point to the first point.
	Polyline (see page 14467)	Draws a series of lines on the canvas with the current pen, connecting each of the points passed to it in Points.
	Rectangle (see page 14468)	This is the overview for the Rectangle method overload.
	Refresh (see page 14470)	Deselects the Pen (see page 14492), Brush (see page 14484), and Font (see page 14488) from the device context.
	RoundRect (see page 14471)	Draws a rectangle with rounded corners on the canvas.
	StretchDraw (see page 14472)	Draws the graphic specified by the Graphic parameter in the rectangle specified by the Rect parameter.
	TextExtent (see page 14474)	Returns the width and height, in pixels, of a string rendered in the current font.
	TextHeight (see page 14474)	Returns the height, in pixels, of a string rendered in the current font.
	TextOut (see page 14475)	Writes a string on the canvas, starting at the point (X,Y), and then updates the PenPos (see page 14494) to the end of the string.
	TextRect (see page 14480)	This is the overview for the TextRect method overload.
	TextWidth (see page 14481)	Returns the width, in pixels, of a string rendered in the current font.
	TryLock (see page 14483)	Locks the canvas if it is currently unlocked.
	Unlock (see page 14483)	Decreases the LockCount (see page 14492) property, allowing other threads to execute when LockCount (see page 14492) reaches 0.

73.1.7.4.1 TCanvas.Arc Method

Draws an arc on the image along the perimeter of the ellipse bounded by the specified rectangle.

Pascal

```
procedure Arc(X1: Integer; Y1: Integer; X2: Integer; Y2: Integer; X3: Integer; Y3: Integer;
X4: Integer; Y4: Integer);
```

C++

```
__fastcall Arc(int X1, int Y1, int X2, int Y2, int X3, int Y3, int X4, int Y4);
```

Description

Use Arc to draw an elliptically curved line with the current Pen (see page 14492). The arc traverses the perimeter of an ellipse that is bounded by the points (X1,Y1) and (X2,Y2). The arc is drawn following the perimeter of the ellipse, counterclockwise, from the starting point to the ending point. The starting point is defined by the intersection of the ellipse and a line defined by the center of the ellipse and (X3,Y3). The ending point is defined by the intersection of the ellipse and a line defined by the center of the ellipse and (X4, Y4).

Note: On Windows 9x or Windows ME, the sums X1 + X2 and Y1 + Y2 cannot exceed 32768. Also, the sum X1 + X2 + Y1 + Y2 cannot exceed 32768.

On NT or Windows 2000, the drawing direction can be changed to clockwise using the Windows API call SetArcDirection.

C++ Examples:

```
/*
The following lines of code draw the top quarter of an arc
bounded by the current window:
*/
void __fastcall TForm1::FormPaint(TObject *Sender)
{
  TRect R = GetClientRect();  // Gets the rectangular
                              // coordinates of
                              // the current window.
  Canvas->Arc(R.Left, R.Top, R.Right, R.Bottom, R.Right, R.Top, R.Left, R.Top);
```

```
}
```

Delphi Examples:

```
{
The following lines of code draw the top quarter of an arc
bounded by the current window:
}
procedure TForm1.FormPaint(Sender: TObject);
var
  R: TRect;
begin
  R := GetClientRect;   { Gets the boundaries of the current window }
  Canvas.Arc(R.Left, R.Top, R.Right, R.Bottom, R.Right, R.Top, R.Left, R.Top);
end;
```

See Also

Ellipse (see page 14457)

Pen (see page 14492)

73.1.7.4.2 TCanvas.BrushCopy Method

Copies a portion of a bitmap onto a rectangle on the canvas, replacing one of the colors of the bitmap with the brush of the canvas.

Pascal

```
procedure BrushCopy(const Dest: TRect; Bitmap: TBitmap; const Source: TRect; Color: TColor);
```

C++

```
__fastcall BrushCopy(const TRect Dest, TBitmap Bitmap, const TRect Source, TColor Color);
```

Description

Use BrushCopy to achieve special effects such as making the copied image partially transparent. BrushCopy is provided mainly for backward compatibility. Use a TImageList instead of BrushCopy.

Dest specifies the rectangular portion of the canvas that will receive the copy of the bitmap. Bitmap specifies the graphic to copy from. Source specifies the rectangular area of Bitmap to copy. Color specifies the color in Bitmap to replace with the Brush (see page 14484) of the canvas.

To use BrushCopy to make the copied image partially transparent, specify the color of the surface of the canvas (clBackground (see page 14666) for example) as the Color of the Brush (see page 14484) property, then call BrushCopy.

C++ Examples:

```
/*
The following code illustrates the differences between
CopyRect and BrushCopy. The bitmap graphic 'FACTORY.BMP' is
loaded into Bitmap and displayed on the Canvas of Form1.
BrushCopy replaces the color black in the graphic with the
brush of the canvas, while CopyRect leaves the colors intact.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Graphics::TBitmap *Bitmap;
TRect   MyRect, MyOther;
MyRect = Rect(10,10,100,100);
MyOther = Rect(10,111,100, 201);
Bitmap = new Graphics::TBitmap;
Bitmap->LoadFromFile("c:/Program Files/Common Files/CodeGear
Shared/Images/Splash/256color/factory.bmp");
Form1->Canvas->BrushCopy(MyRect, Bitmap, MyRect, clBlack);
```

```
Form1->Canvas->CopyRect(MyOther, Bitmap->Canvas, MyRect);
delete Bitmap;
}
```

Delphi Examples:

```
{
The following code illustrates the differences between
CopyRect and BrushCopy. The bitmap graphic 'FACTORY.BMP' is
loaded into Bitmap and displayed on the Canvas of Form1.
BrushCopy replaces the color black in the graphic with the
brush of the canvas, while CopyRect leaves the colors intact.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap: TBitmap;
  MyRect, MyOther: TRect;
begin
  MyRect := Rect(10,10,150,150);
  MyOther := Rect(10,161,150,301);
  Bitmap := TBitmap.Create;
  Bitmap.LoadFromFile(
    'c:\Program Files\Common Files\CodeGear Shared\Images\Splash\256color\factory.bmp');
  Form1.Canvas.BrushCopy(MyRect, Bitmap, MyRect, clBlack);
  Form1.Canvas.CopyRect(MyOther,Bitmap.Canvas,MyRect);
  Bitmap.Free;
end;
```

See Also

Brush (see page 14484)

CopyMode (see page 14487)

StretchDraw (see page 14472)

TBitmap (see page 14396)

Color (see page 14437)

TImageList (see page 6078)

73.1.7.4.3 TCanvas.Changed Method

Pascal

```
procedure Changed; virtual;
```

C++

```
virtual __fastcall Changed();
```

Description

This is Changed, a member of class TCanvas.

73.1.7.4.4 TCanvas.Changing Method

Pascal

```
procedure Changing; virtual;
```

C++

```
virtual __fastcall Changing();
```

Description

This is Changing, a member of class TCanvas.

73.1.7.4.5 TCanvas.Chord Method

Draws a closed figure represented by the intersection of a line and an ellipse.

Pascal

```
procedure Chord(X1: Integer; Y1: Integer; X2: Integer; Y2: Integer; X3: Integer; Y3: Integer;
X4: Integer; Y4: Integer);
```

C++

```
__fastcall Chord(int X1, int Y1, int X2, int Y2, int X3, int Y3, int X4, int Y4);
```

Description

Use Chord to create a shape that is defined by an arc and a line that joins the endpoints of the arc. The chord consists of a portion of an ellipse that is bounded by the points (X1,Y1) and (X2,Y2). The ellipse is bisected by a line that runs between the points (X3,Y3) and (X4,Y4).

The perimeter of the chord runs counter clockwise from (X3, Y3), counterclockwise along the ellipse to (X4,Y4), and straight back to (X3,Y3). If (X3,Y3) and (X4,Y4) are not on the surface of the ellipse, the corresponding corners on the chord are the closest points on the perimeter that intersect the line. The outline of the chord is drawn using the value of Pen (see page 14492), and the shape is filled using the value of Brush (see page 14484).

Note: On Windows 9x or Windows ME, the sums X1 + X2 and Y1 + Y2 cannot exceed 32768. Also, the sum X1 + X2 + Y1 + Y2 cannot exceed 32768.

On NT, the drawing direction can be changed to clockwise using the Windows API call SetArcDirection.

C++ Examples:

```
/*
This code draws a chord on the top of an ellipse bounded by
the current window.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TRect R = GetClientRect();  //Get coordinates of the current window.
  Canvas->Chord(R.Left, R.Top, R.Right, R.Bottom, R.Right, R.Top, R.Left, R.Top);
}
```

Delphi Examples:

```
{
This code draws a chord on the top of an ellipse bounded by
the current window.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  R: TRect;
begin
  R := GetClientRect;  { Get coordinates of the current window. }
  Canvas.Chord(R.Left, R.Top, R.Right, R.Bottom, R.Right, R.Top, R.Left, R.Top);
end;
```

See Also

Arc (see page 14448)

Brush (see page 14484)

Ellipse (see page 14457)

Pen (see page 14492)

73.1.7.4.6 TCanvas.CopyRect Method

Copies part of an image from another canvas into the canvas.

Pascal

```
procedure CopyRect(const Dest: TRect; Canvas: TCanvas; const Source: TRect);
```

C++

```
__fastcall CopyRect(const TRect Dest, TCanvas Canvas, const TRect Source);
```

Description

Use CopyRect to transfer part of the image on another canvas to the image of the TCanvas (see page 14441) object. Dest specifies the rectangle on the canvas where the source image will be copied. The Canvas parameter specifies the canvas with the source image. Source specifies a rectangle bounding the portion of the source canvas that will be copied.

The portion of the source canvas is copied using the mode specified by CopyMode (see page 14487).

C++ Examples:

```
/*
The following code illustrates the differences between
CopyRect and BrushCopy. The bitmap graphic 'FACTORY.BMP' is
loaded into Bitmap and displayed on the Canvas of Form1.
BrushCopy replaces the color black in the graphic with the
brush of the canvas, while CopyRect leaves the colors intact.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Graphics::TBitmap *Bitmap;
TRect   MyRect, MyOther;
MyRect = Rect(10,10,100,100);
MyOther = Rect(10,111,100, 201);
Bitmap = new Graphics::TBitmap;
Bitmap->LoadFromFile("c:/Program Files/Common Files/CodeGear
Shared/Images/Splash/256color/factory.bmp");
Form1->Canvas->BrushCopy(MyRect, Bitmap, MyRect, clBlack);
Form1->Canvas->CopyRect(MyOther, Bitmap->Canvas, MyRect);
delete Bitmap;
}
/*
The following example uses the CopyMode of an image's canvas
to blank out the image when the user chooses the "Cut" menu
item.  This example requires a Main menu with a Copy and a
Cut menu item that have their OnClick error handlers populated.
Note: CopyMode and CopyRect are members of the destination
canvas.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
    TRect SrcRect, DstRect;
    Image2->Canvas->CopyMode = cmSrcCopy;
    DstRect = Rect(0, 0, Image2->Width, Image2->Height);
    SrcRect = Rect(0, 0, Image1->Width, Image1->Height);
    Image2->Canvas->CopyRect(DstRect, Image1->Canvas, SrcRect);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
 TRect r;
 Copy1Click(Sender); // Do the same thing as the copy menu item.
 Image1->Canvas->CopyMode = cmWhiteness;
```

```
 r = Rect(0, 0, Image1->Width, Image1->Height);
 Image1->Canvas->CopyRect(r, Image1->Canvas, r);
 Image1->Canvas->CopyMode = cmSrcCopy; // Restore the copy mode.
}
```

Delphi Examples:

```
{
The following code illustrates the differences between
CopyRect and BrushCopy. The bitmap graphic 'FACTORY.BMP' is
loaded into Bitmap and displayed on the Canvas of Form1.
BrushCopy replaces the color black in the graphic with the
brush of the canvas, while CopyRect leaves the colors intact.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap: TBitmap;
  MyRect, MyOther: TRect;
begin
  MyRect := Rect(10,10,150,150);
  MyOther := Rect(10,161,150,301);
  Bitmap := TBitmap.Create;
  Bitmap.LoadFromFile(
    'c:\Program Files\Common Files\CodeGear Shared\Images\Splash\256color\factory.bmp');
  Form1.Canvas.BrushCopy(MyRect, Bitmap, MyRect, clBlack);
  Form1.Canvas.CopyRect(MyOther,Bitmap.Canvas,MyRect);
  Bitmap.Free;
end;
{
The following example uses the CopyMode of an image's canvas
to blank out the image when the user chooses the "Cut" menu
item.  This example requires a Main menu with a Copy and a
Cut menu item that have their OnClick error handlers populated.
Note: CopyMode and CopyRect are members of the destination
canvas.
}
procedure TForm1.Copy1Click(Sender: TObject);
var
  DstRect, SrcRect: TRect;
begin
  with Image2.Canvas do
  begin
    CopyMode := cmSrcCopy;
    DstRect := Rect(0, 0, Image2.Width, Image2.Height);
    SrcRect := Rect(0, 0, Image1.Width, Image1.Height);
    CopyRect(DstRect, Image1.Canvas, SrcRect);
  end;
end;

procedure TForm1.Cut1Click(Sender: TObject);
var
  ARect: TRect;
begin
  Copy1Click(Sender); { do the same thing as the copy menu item }
  with Image1.Canvas do
  begin
    CopyMode := cmWhiteness;
    ARect := Rect(0, 0, Image1.Width, Image1.Height);
    CopyRect(ARect, Image1.Canvas, ARect);
    CopyMode := cmSrcCopy; { restore the copy mode }
  end;
end;
```

See Also

BrushCopy (see page 14449)

CopyMode (see page 14487)

StretchDraw (see page 14472)

73.1.7.4.7 TCanvas.CreateHandle Method

Pascal

```
procedure CreateHandle; virtual;
```

C++

```
virtual __fastcall CreateHandle();
```

Description

This is CreateHandle, a member of class TCanvas.

73.1.7.4.8 TCanvas.Draw Method

Renders the graphic specified by the Graphic parameter on the canvas at the location given by the coordinates (X, Y).

Pascal

```
procedure Draw(X: Integer; Y: Integer; Graphic: TGraphic);
```

C++

```
__fastcall Draw(int X, int Y, TGraphic Graphic);
```

Description

Call Draw to draw a graphic on the canvas. Draw calls the Draw method of the graphic. The image is rendered into a rectangle determined by the size of the graphic, with the upper left corner at the point (X, Y).

Graphics (see page 14387) can be bitmaps, icons, or metafiles. If the graphic is a TBitmap (see page 14396) object, the bitmap is rendered using the value of CopyMode (see page 14487).

C++ Examples:

```
/*
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PaintBox1->Canvas->Draw(0,0, Image1->Picture->Graphic);
}
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *bitmap = new Graphics::TBitmap();
  try
  {
    bitmap->LoadFromFile("..\\FACTORY.BMP ");
    bitmap->Transparent = true;
    bitmap->TransparentColor = bitmap->Canvas->Pixels[50][50];
    Form1->Canvas->Draw( 0, 0, bitmap);
    bitmap->TransparentMode = tmAuto;
    Form1->Canvas->Draw(50, 50, bitmap);
  }
  catch (...)
  {
    ShowMessage("Could not load or display bitmap");
  }
  delete bitmap;
```

Delphi Examples:

```
{
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  PaintBox1.Canvas.Draw(0,0, Image1.Picture.Graphic);
end;
procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitMap;
begin
  Bitmap := TBitmap.Create;
  try
    with Bitmap do begin
      LoadFromFile('factory.bmp');
      Transparent := True;
      TransParentColor := BitMap.canvas.pixels[50,50];
      Form1.Canvas.Draw(0,0,BitMap);
      TransparentMode := tmAuto;
      Form1.Canvas.Draw(50,50,BitMap);
    end;
  finally
    Bitmap.Free;
  end;
end;
```

See Also

BrushCopy (see page 14449)

CopyMode (see page 14487)

CopyRect (see page 14452)

StretchDraw (see page 14472)

73.1.7.4.9 TCanvas.DrawFocusRect Method

Draws a rectangle in the style used to indicate that the object inside the rectangle has focus.

Pascal

```
procedure DrawFocusRect(const Rect: TRect);
```

C++

```
__fastcall DrawFocusRect(const TRect Rect);
```

Description

Call DrawFocusRect for the perimeter of the image of a control when the control gets or loses input focus. Because DrawFocusRect uses an XOR function, calling it a second time while specifying the same rectangle removes the rectangle from the screen.

The rectangle this function draws cannot be scrolled. To scroll an area containing a rectangle drawn by DrawFocusRect, call DrawFocusRect to remove the rectangle from the screen, scroll the area, and then call DrawFocusRect to draw the rectangle in the new position.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
```

```
  component to draw the contents of each cell in a draw grid.
  It draws a focus rectangle around the cell that has focus.
  goDrawFocusSelect in the DrawGrid Options parameter must be
  True to set focus on a cell.  The ImageList Draw method must
  be called after DrawFocusRect. The OnSelectCell event handler
  must implemented to return true.
  */
  void __fastcall TForm1::DrawGrid1DrawCell(
    TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
  {
    long index = ARow * DrawGrid1->ColCount + ACol;
    DrawGrid1->Canvas->Brush->Color = clBackground;
    DrawGrid1->Canvas->FillRect(Rect);
    ImageList1->Draw(DrawGrid1->Canvas,Rect.Left,Rect.Top,index);
    if (State.Contains(gdFocused))
    {
      DrawGrid1->Canvas->DrawFocusRect(Rect);
      Label1->Caption = "Cell " + IntToStr(index) + " has the focus.";
    }
    ImageList1->Draw(
      DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
  }
  //---------------------------------------------------------------------------
  void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
        bool &CanSelect)
  {
    CanSelect= True;
  }
```

Delphi Examples:

```
  {
  The following code uses the bitmaps in an image list
  component to draw the contents of each cell in a draw grid.
  It draws a focus rectangle around the cell that has focus.
  goDrawFocusSelect in the DrawGrid Options parameter must be
  True to set focus on a cell.  The ImageList Draw method must
  be called after DrawFocusRect. The OnSelectCell event handler
  must implemented to return true.
  }
  procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
    Rect: TRect; State: TGridDrawState);
  var
    index: Integer;
  begin
    index := ARow * DrawGrid1.ColCount + ACol;
    DrawGrid1.Canvas.Brush.Color := clWhite;
    DrawGrid1.Canvas.FillRect(Rect);
    if (gdFocused in State) then
    begin
      DrawGrid1.Canvas.DrawFocusRect(Rect);
      Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
    end;
    ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  end;

  procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  begin
    CanSelect:= True;
  end;
```

See Also

ActiveControl (see page 13896)

ActiveControl (see page 14206)

73.1.7.4.10 Ellipse Method

This is the overview for the Ellipse method overload.

Overload List

	Name	Description
	TCanvas.Ellipse (Integer, Integer, Integer, Integer) (see page 14457)	Draws the ellipse defined by a bounding rectangle on the canvas.
	TCanvas.Ellipse (TRect) (see page 14457)	Draws the ellipse defined by a bounding rectangle on the canvas.

73.1.7.4.10.1 TCanvas.Ellipse Method (Integer, Integer, Integer, Integer)

Draws the ellipse defined by a bounding rectangle on the canvas.

Pascal

```
procedure Ellipse(X1: Integer; Y1: Integer; X2: Integer; Y2: Integer); overload;
procedure Ellipse(const Rect: TRect); overload;
```

C++

```
__fastcall Ellipse(int X1, int Y1, int X2, int Y2);
__fastcall Ellipse(const TRect Rect);
```

Description

Call Ellipse to draw a circle or ellipse on the canvas. Specify the bounding rectangle either by giving

The top left point at pixel coordinates (X1, Y1) and the bottom right point at (X2, Y2).

A TRect value.

If the bounding rectangle is a square, a circle is drawn.

The ellipse is outlined using the value of Pen (see page 14492), and filled using the value of Brush (see page 14484).

Note: On Windows 9x or Windows ME, the sums X1 + X2 and Y1 + Y2 cannot exceed 32768. Also, the sum X1 + X2 + Y1 + Y2 cannot exceed 32768.

C++ Examples:

```
/*
The following example paints a cross-hatched ellipse onto
Image1 which is a TImage on the form when the user clicks on
Button1.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TCanvas *pCanvas = Image1->Canvas;
  pCanvas->Brush->Color = clRed;
  pCanvas->Brush->Style = bsDiagCross;
  pCanvas->Ellipse(0, 0, Image1->Width, Image1->Height);
}
```

Delphi Examples:

```
{
The following example paints a cross-hatched ellipse onto
Image1 which is a TImage on the form when the user clicks on
Button1.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Image1 do begin
    Canvas.Brush.Color := clRed;
    Canvas.Brush.Style := bsDiagCross;
    Canvas.Ellipse(0, 0, Image1.Width, Image1.Height);
  end;
end;
```

See Also

Arc (see page 14448)

Brush (see page 14484)

Chord (see page 14451)

Pen (see page 14492)

73.1.7.4.11 TCanvas.FillRect Method

Fills the specified rectangle on the canvas using the current brush.

Pascal

```
procedure FillRect(const Rect: TRect);
```

C++

```
__fastcall FillRect(const TRect Rect);
```

Description

Use FillRect to fill a rectangular region using the current brush. The region is filled including the top and left sides of the rectangle, but excluding the bottom and right edges.

C++ Examples:

```
/*
The following code loads a bitmap from a file and assigns it
to the Brush of the Canvas of Form1:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *BrushBmp = new Graphics::TBitmap;
  try
  {
    BrushBmp->LoadFromFile("../bm1.BMP");
    Form1->Canvas->Brush->Bitmap = BrushBmp;
    Form1->Canvas->FillRect(Rect(0,0,100,100));
  }
  __finally
  {
    Form1->Canvas->Brush->Bitmap = NULL;
    delete BrushBmp;
  }
}
```

Delphi Examples:

```
{
The following code loads a bitmap from a file and assigns it
to the Brush of the Canvas of Form1:
}
var
  Bitmap: TBitmap;
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Bitmap := TBitmap.Create;
  try
    Bitmap.LoadFromFile('bm1.BMP');
    Form1.Canvas.Brush.Bitmap := Bitmap;
    Form1.Canvas.FillRect(Rect(0,0,100,100));
  finally
    Form1.Canvas.Brush.Bitmap := nil;
    Bitmap.Free;
  end;
end;
```

See Also

FloodFill (see page 14459)

73.1.7.4.12 TCanvas.FloodFill Method

Fills an area of the canvas using the current brush.

Pascal

```
procedure FloodFill(X: Integer; Y: Integer; Color: TColor; FillStyle: TFillStyle);
```

C++

```
__fastcall FloodFill(int X, int Y, TColor Color, TFillStyle FillStyle);
```

Description

Use FloodFill to fill a possibly non-rectangular region of the image with the value of Brush (see page 14484). The boundaries of the region to be filled are determined by moving outward from the point (X,Y) until a color boundary involving the Color parameter is encountered.

X and X are the coordinates on the canvas where filling starts.

Color is the color that defines the boundary of the region to fill. Its interpretation depends on the value of FillStyle.

FillStyle specifies whether the region is defined by all pixels with the same value as Color, or all points with a different value.

Tip: Use the Pixels (see page 14494) property to get the exact value of the color at the point (X,Y) when using a FillStyle of fsSurface. Similarly, when FillStyle is fsBorder, use Pixels (see page 14494) to get the exact value of the boundary color if a point on the boundary is known.

C++ Examples:

```
/*
The following code floodfills from the center point of
Form1's client area until the color black is encountered:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Canvas->Brush->Color = clBlue;
  Canvas->Brush->Style = bsDiagCross;
  Canvas->FloodFill(
    ClientWidth/2, ClientHeight/2, clBlack, fsBorder);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  int ewidth = 400;
  int eheight = 200;
```

```
  Canvas->Brush->Color = clRed;
  Canvas->Pen->Color = clBlack;
  Canvas->Ellipse(
    (ClientWidth - ewidth)/2,
    (ClientHeight - eheight)/2,
    ewidth, eheight);
}
```

Delphi Examples:

```
{
The following code floodfills from the center point of
Form1's client area until the color black is encountered:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Canvas.Brush.Color := clBlue;
  Canvas.Brush.Style := bsDiagCross;
  Canvas.FloodFill(
    ClientWidth div 2, ClientHeight div 2, clBlack, fsBorder);
end;

procedure TForm1.Button2Click(Sender: TObject);
var ewidth, eheight: integer;
begin
  ewidth := 400;
  eheight := 200;
  Canvas.Brush.Color := clRed;
  Canvas.Pen.Color := clBlack;
  Canvas.Ellipse(
    (ClientWidth - ewidth) div 2,
    (ClientHeight - eheight) div 2,
    ewidth, eheight);
end;
```

See Also

FillRect (see page 14458)

Pixels (see page 14494)

73.1.7.4.13 TCanvas.FrameRect Method

Draws a rectangle using the Brush (see page 14484) of the canvas to draw the border.

Pascal

```
procedure FrameRect(const Rect: TRect);
```

C++

```
__fastcall FrameRect(const TRect Rect);
```

Description

Use FrameRect to draw a 1 pixel wide border around a rectangular region. FrameRect does not fill the interior of the rectangle with the Brush (see page 14484) pattern.

To draw a boundary using the Pen (see page 14492) instead, use the Polygon (see page 14466) method.

C++ Examples:

```
/*
The following code displays the text "Hello, world!" in a
rectangle defined by the coordinates (10, 10) and (100, 100).
After displaying the text with the TextRect method, the code
draws a black, vertical line frame around the rectangle.
```

```
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TRect TheRect = Rect(10,10,100,100);
  Canvas->TextRect(TheRect, 10, 10, "Hello World");
  Canvas->Brush->Color = clBlack;
  Canvas->FrameRect(TheRect);
}
```

Delphi Examples:

```
{
The following code displays the text "Hello, world!" in a
rectangle defined by the coordinates (10, 10) and (100, 100).
After displaying the text with the TextRect method, the code
draws a black, vertical line frame around the rectangle.
}
procedure TForm1.Button1Click(Sender: TObject);
var TheRect: TRect;
begin
  TheRect := Rect(10,10,100,100);
  Form1.Canvas.TextRect(TheRect,10,10,'Hello, world!');
  Form1.Canvas.Brush.Color := clBlack;
  Form1.Canvas.FrameRect(TheRect);
end;
```

See Also

Brush (see page 14484)

FillRect (see page 14458)

Pen (see page 14492)

Polygon (see page 14466)

Rectangle (see page 14468)

RoundRect (see page 14471)

73.1.7.4.14 TCanvas.HandleAllocated Method

Indicates whether the TCanvas (see page 14441) object has acquired a handle to a device context.

Pascal

```
function HandleAllocated: Boolean;
```

C++

```
__fastcall Boolean HandleAllocated();
```

Description

Use HandleAllocated to determine whether the Handle (see page 14489) property is set to an HDC value. HandleAllocated indicates whether the canvas has acquired a device context. Use HandleAllocated rather than reading the Handle (see page 14489) property. When an application reads the Handle (see page 14489) property, TCanvas (see page 14441) automatically allocates a new device context and returns its handle if the property had not been previously set.

See Also

Handle (see page 14489)

73.1.7.4.15 TCanvas.LineTo Method

Draws a line on the canvas from PenPos (see page 14494) to the point specified by X and Y, and sets the pen position to (X, Y).

Pascal

```
procedure LineTo(X: Integer; Y: Integer);
```

C++

```
__fastcall LineTo(int X, int Y);
```

Description

Use LineTo to draw a line from PenPos (see page 14494) up to, but not including the point (X,Y). LineTo changes the value of PenPos (see page 14494) to (X,Y).

The line is drawn using Pen (see page 14492).

Note: If the current Pen (see page 14492) does not have a style of psSolid, the line is drawn with a background specified by the current brush.

C++ Examples:

```
/*
The following code draws a line from the upper left corner
of a form to the mouse position when the mouse is moved,
creating a "rubber band" effect.
*/
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  /* first call FillRect to paint the surface of the form.
  this removes any previously drawn lines (and anything else!) */
  Canvas->FillRect(ClientRect);
  Canvas->MoveTo(0, 0);
  Canvas->LineTo(X, Y);
}
```

Delphi Examples:

```
{
The following code draws a line from the upper left corner
of a form to the mouse position when the mouse is moved,
creating a "rubber band" effect.
}
procedure TForm1.FormMouseMove(
  Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  { first call FillRect to paint the surface of the form.
  this removes any previously drawn lines (and anything else!) }
  Canvas.FillRect(ClientRect);
  Canvas.MoveTo(0, 0);
  Canvas.LineTo(X, Y);
end;
```

See Also

Pen (see page 14492)

PenPos (see page 14494)

Polyline (see page 14467)

Style (see page 14579)

Brush (see page 14484)

73.1.7.4.16 TCanvas.Lock Method

Prevents other threads from drawing on the canvas.

Pascal

```
procedure Lock;
```

C++

```
__fastcall Lock();
```

Description

Call Lock in a multithreaded application to prevent other threads from drawing on the canvas. Lock prevents other cooperating threads in the application from executing until the Unlock (see page 14483) method is called. Nested calls to Lock increment the LockCount (see page 14492) property so that the canvas is not unlocked until the last lock is released.

In multi-threaded applications that use Lock to protect a canvas, all calls that use the canvas must be protected by a call to Lock. Any thread that does not lock the canvas before using it will introduce potential bugs.

Because Lock prevents other threads from executing, it can adversely affect performance. Do not call Lock unless there is a danger that another thread might interfere with drawing on the canvas.

See Also

LockCount (see page 14492)

TryLock (see page 14483)

Unlock (see page 14483)

Locking Objects

73.1.7.4.17 TCanvas.MoveTo Method

Changes the current drawing position to the point (X,Y).

Pascal

```
procedure MoveTo(X: Integer; Y: Integer);
```

C++

```
__fastcall MoveTo(int X, int Y);
```

Description

Use MoveTo to set the value of PenPos (see page 14494) before calling LineTo (see page 14462). Calling MoveTo is equivalent to setting the PenPos (see page 14494) property.

C++ Examples:

```
/*
The following code draws a line from the upper left corner
of a form to the mouse position when the mouse is moved,
creating a "rubber band" effect.
*/
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  /* first call FillRect to paint the surface of the form.
  this removes any previously drawn lines (and anything else!) */
  Canvas->FillRect(ClientRect);
  Canvas->MoveTo(0, 0);
```

```
  Canvas->LineTo(X, Y);
}
```

Delphi Examples:

```
{
The following code draws a line from the upper left corner
of a form to the mouse position when the mouse is moved,
creating a "rubber band" effect.
}
procedure TForm1.FormMouseMove(
  Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  { first call FillRect to paint the surface of the form.
  this removes any previously drawn lines (and anything else!) }
  Canvas.FillRect(ClientRect);
  Canvas.MoveTo(0, 0);
  Canvas.LineTo(X, Y);
end;
```

See Also

LineTo (see page 14462)

PenPos (see page 14494)

73.1.7.4.18 TCanvas.Pie Method

Draws a pie-shaped the section of the ellipse bounded by the rectangle (X1, Y1) and (X2, Y2) on the canvas.

Pascal

```
procedure Pie(X1: Integer; Y1: Integer; X2: Integer; Y2: Integer; X3: Integer; Y3: Integer;
X4: Integer; Y4: Integer);
```

C++

```
__fastcall Pie(int X1, int Y1, int X2, int Y2, int X3, int Y3, int X4, int Y4);
```

Description

Use Pie to draw a pie-shaped wedge on the image. The wedge is defined by the ellipse bounded by the rectangle determined by the points (X1, Y1) and X2, Y2). The section drawn is determined by two lines radiating from the center of the ellipse through the points (X3, Y3) and (X4, Y4).

The wedge is outlined using Pen (see page 14492), and filled using Brush (see page 14484).

Note: On Windows 9x or Windows ME, the sums X1 + X2 and Y1 + Y2 cannot exceed 32768. Also, the sum X1 + X2 + Y1 + Y2 cannot exceed 32768.

See Also

Arc (see page 14448)

Brush (see page 14484)

Chord (see page 14451)

Pen (see page 14492)

73.1.7.4.19 TCanvas.PolyBezier Method

Draws a set of Bezier curves.

Pascal

```
procedure PolyBezier(const Points: array of TPoint);
```

C++

```
__fastcall PolyBezier(const array of TPoint Points);
```

Description

Use PolyBezier to draw cubic Bezier curves using the endpoints and control points specified by the Points parameter. The first curve is drawn from the first point to the fourth point, using the second and third points as control points. Each subsequent curve in the sequence needs exactly three more points: the ending point of the previous curve is used as the starting point, the next two points in the sequence are control points, and the third is the ending point.

The Points parameter gives the endpoints to use when generating the Bezier curves.

Note: The Points_Size parameter is the index of the last point in the array (one less than the total number of points).

This procedure draws lines by using the current pen.

Note: In Delphi, you can use the Slice function to pass a portion of an array of points to the PolyBezier method. For example, to form a Bezier curve using the first ten points from an array of 100 points, use the Slice function as follows:

```
Canvas.PolyBezier(Slice(PointArray, 10));
```

Control points after a[Index+3] are ignored. Nothing happens if there aren't enough control points.

See Also

Polyline (see page 14467)

PolyBezierTo (see page 14465)

Pen (see page 14492)

73.1.7.4.20 TCanvas.PolyBezierTo Method

Draws a set of Bezier curves and updates the value of PenPos (see page 14494).

Pascal

```
procedure PolyBezierTo(const Points: array of TPoint);
```

C++

```
__fastcall PolyBezierTo(const array of TPoint Points);
```

Description

Use PolyBezierTo to draw cubic Bezier curves using the endpoints and control points specified by the Points parameter. The first curve is drawn from the first point to the fourth point, using the second and third points as control points. Each subsequent curve in the sequence needs exactly three more points: the ending point of the previous curve is used as the starting point, the next two points in the sequence are control points, and the third is the ending point.

The Points parameter gives the endpoints to use when generating the Bezier curves.

Note: The Points_Size parameter is the index of the last point in the array (one less than the total number of points).

This procedure draws lines by using the current pen and updates the value of PenPos (see page 14494) to the value of the last endpoint.

Note: In Delphi, you can use the Slice function to pass a portion of an array of points to the PolyBezierTo method. For example, to form a Bezier curve using the first ten points from an array of 100 points, use the Slice function as follows:

```
Canvas.PolyBezierTo(Slice(PointArray, 10));
```

See Also

Polyline (see page 14467)

PolyBezier (see page 14464)

Pen (see page 14492)

PenPos (see page 14494)

73.1.7.4.21 TCanvas.Polygon Method

Draws a series of lines on the canvas connecting the points passed in and closing the shape by drawing a line from the last point to the first point.

Pascal

```
procedure Polygon(const Points: array of TPoint);
```

C++

```
__fastcall Polygon(const array of TPoint Points);
```

Description

Use Polygon to draw a closed, many-sided shape on the canvas, using the value of Pen (see page 14492). After drawing the complete shape, Polygon fills the shape using the value of Brush (see page 14484).

The Points parameter is an array of points that give the vertices of the polygon.

Note: The Points_Size parameter is the index of the last point in the array (one less than the total number of points).

The first point is always connected to the last point.

Note: In Delphi, you can use the Slice function to pass a portion of an array of points to the Polygon method. For example, to form a polygon using the first ten points from an array of 100 points, use the Slice function as follows:

```
Canvas.Polygon(Slice(PointArray, 10));
```

To draw a polygon on the canvas, without filling it, use the Polyline (see page 14467) method, specifying the first point a second time at the end.

C++ Examples:

```
/*
This example draws a polygon in the specified shape, and
fills it with the color teal.
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
  TPoint points[4];
  points[0] = Point(10,10);
  points[1] = Point(30,10);
  points[2] = Point(130,30);
  points[3] = Point(240,120);
  ((TPaintBox *)Sender)->Canvas->Brush->Color = clTeal;
  ((TPaintBox *)Sender)->Canvas->Polygon(points, 3);
}
```

Delphi Examples:

```
{
This example draws a polygon in the specified shape, and
fills it with the color teal.
}
procedure TForm1.FormPaint(Sender: TObject);
begin
  Canvas.Brush.Color := clTeal;
  Canvas.Polygon([Point(10, 10), Point(30, 10),
    Point(130, 30), Point(240, 120)]);
end;
```

See Also

Brush (see page 14484)

Pen (see page 14492)

Polyline (see page 14467)

Slice (see page 22529)

73.1.7.4.22 TCanvas.Polyline Method

Draws a series of lines on the canvas with the current pen, connecting each of the points passed to it in Points.

Pascal

```
procedure Polyline(const Points: array of TPoint);
```

C++

```
__fastcall Polyline(const array of TPoint Points);
```

Description

Use Polyline to connect a set of points on the canvas. If you specify only two points, Polyline draws a single line.

The Points parameter is an array of points to be connected.

Note: The Points_Size parameter is the index of the last point in the array (one less than the total number of points).

Note: In Delphi, you can use the Slice function to pass a portion of an array of points to the Polyline method. For example, to form a line connecting the first ten points from an array of 100 points, use the Slice function as follows:

```
Canvas.Polyline(Slice(PointArray, 10));
```

Calling the MoveTo (see page 14463) function with the value of the first point, and then repeatedly calling LineTo (see page 14462) with all subsequent points will draw the same image on the canvas. However, unlike LineTo (see page 14462), Polyline does not change the value of PenPos (see page 14494).

C++ Examples:

```
/*
This example paints a white five-pointed star in a paintbox
control:
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
  TPaintBox *pPB = (TPaintBox *)Sender;
  TPoint points[6];
  pPB->Canvas->Pen->Color = clWhite;
  points[0].x = 40;
  points[0].y = 10;
```

```
  points[1].x = 20;
  points[1].y = 60;
  points[2].x = 70;
  points[2].y = 30;
  points[3].x = 10;
  points[3].y = 30;
  points[4].x = 60;
  points[4].y = 60;
  points[5].x = 40;
  points[5].y = 10;
  pPB->Canvas->Polyline(points,5);
}
```

Delphi Examples:

```
{
This example paints a white five-pointed star in a paintbox
control:
}
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
  with Sender as TPaintBox do
  begin
    Canvas.Pen.Color := clWhite;
    Canvas.Polyline([Point(40, 10), Point(20, 60), Point(70, 30),
    Point(10, 30), Point(60, 60), Point(40, 10)]);
  end;
end;
```

See Also

LineTo (see page 14462)

MoveTo (see page 14463)

Pen (see page 14492)

PenPos (see page 14494)

Polygon (see page 14466)

Slice (see page 22529)

73.1.7.4.23 Rectangle Method

This is the overview for the Rectangle method overload.

Overload List

	Name	Description
	TCanvas.Rectangle (Integer, Integer, Integer, Integer) (see page 14468)	Draws a rectangle on the canvas.
	TCanvas.Rectangle (TRect) (see page 14468)	Draws a rectangle on the canvas.

73.1.7.4.23.1 TCanvas.Rectangle Method (Integer, Integer, Integer, Integer)

Draws a rectangle on the canvas.

Pascal

```
procedure Rectangle(X1: Integer; Y1: Integer; X2: Integer; Y2: Integer); overload;
procedure Rectangle(const Rect: TRect); overload;
```

C++

```
__fastcall Rectangle(int X1, int Y1, int X2, int Y2);
__fastcall Rectangle(const TRect Rect);
```

Description

Use Rectangle to draw a rectangle using Pen (see page 14492) and fill it with Brush (see page 14484). Specify the rectangle's coordinates in one of two ways:

Giving four coordinates that define the upper left corner at the point (X1, Y1) and the lower right corner at the point (X2, Y2).

Using a TRect type.

To fill a rectangular region without drawing the boundary in the current pen, use FillRect (see page 14458). To outline a rectangular region without filling it, use FrameRect (see page 14460) or Polygon (see page 14466). To draw a rectangle with rounded corners, use RoundRect (see page 14471).

C++ Examples:

```
/*
This example draws many rectangles of various sizes and
colors on a form maximized to fill the entire screen. To run
the code, drop a TTimer component on the form and use the
object inspector to create the OnTimer and OnActivate event
handlers.
*/
int x, y;

void __fastcall TForm1::FormActivate(TObject *Sender)
{
  WindowState = wsMaximized;
  Timer1->Interval = 50;
  randomize();
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  x = random(Screen->Width - 10);
  y = random(Screen->Height - 10);
  Canvas->Pen->Color = (Graphics::TColor) random(65535);
  switch (random(5))
  {
    case 0: Canvas->Pen->Style = psSolid; break;
    case 1: Canvas->Pen->Style = psDash; break;
    case 2: Canvas->Pen->Style = psDot; break;
    case 3: Canvas->Pen->Style = psDashDot; break;
    case 4: Canvas->Pen->Style = psDashDotDot; break;
  }
  Canvas->Rectangle(x, y, x + random(400), y + random(400));
}
```

Delphi Examples:

```
{
This example draws many rectangles of various sizes and
colors on a form maximized to fill the entire screen. To run
the code, drop a TTimer component on the form and use the
object inspector to create the OnTimer and OnActivate event
handlers.
}
var
  X, Y: Integer;
procedure TForm1.FormActivate(Sender: TObject);
begin
  WindowState := wsMaximized;
  Timer1.Interval := 50;
```

```
    Randomize;
  end;

  procedure TForm1.Timer1Timer(Sender: TObject);
  begin
    X := Random(Screen.Width - 10);
    Y := Random(Screen.Height - 10);
    Canvas.Pen.Color := Random(65535);
    case Random(5) of
      0: Canvas.Pen.Style := psSolid;
      1: Canvas.Pen.Style := psDash;
      2: Canvas.Pen.Style := psDot;
      3: Canvas.Pen.Style := psDashDot;
      4: Canvas.Pen.Style := psDashDotDot;
    end;
    Canvas.Rectangle(X, Y, X + Random(400), Y + Random(400));
  end;
```

See Also

Brush (see page 14484)

FillRect (see page 14458)

FrameRect (see page 14460)

Pen (see page 14492)

Polygon (see page 14466)

RoundRect (see page 14471)

73.1.7.4.24 TCanvas.Refresh Method

Deselects the Pen (see page 14492), Brush (see page 14484), and Font (see page 14488) from the device context.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh to return the HDC used by the canvas to a default state. Refresh does not change the values of the Font (see page 14488), Brush (see page 14484), or Pen (see page 14492) properties. Calling any of the drawing methods of the canvas after calling Refresh will reselect the Font (see page 14488), Brush (see page 14484), or Pen (see page 14492) into the HDC if it is used by the drawing method.

See Also

Brush (see page 14484)

Font (see page 14488)

Handle (see page 14489)

Pen (see page 14492)

73.1.7.4.25 TCanvas.RequiredState Method

Pascal

```
procedure RequiredState(ReqState: TCanvasState);
```

C++

```
__fastcall RequiredState(TCanvasState ReqState);
```

Description

This is RequiredState, a member of class TCanvas.

73.1.7.4.26 TCanvas.RoundRect Method

Draws a rectangle with rounded corners on the canvas.

Pascal

```
procedure RoundRect(X1: Integer; Y1: Integer; X2: Integer; Y2: Integer; X3: Integer; Y3:
Integer);
```

C++

```
__fastcall RoundRect(int X1, int Y1, int X2, int Y2, int X3, int Y3);
```

Description

Use RoundRect to draw a rounded rectangle using Pen (see page 14492) and fill it with Brush (see page 14484). The rectangle will have edges defined by the points (X1,Y1), (X2,Y1), (X2,Y2), (X1,Y2), but the corners will be shaved to create a rounded appearance. The curve of the rounded corners matches the curvature of an ellipse with width X3 and height Y3.

To draw an ellipse instead, use Ellipse (see page 14457). To draw a true rectangle, use Rectangle (see page 14468).

C++ Examples:

```
/*
This example draws many rounded rectangles of various sizes
and colors on a form maximized to fill the entire screen:
*/
int x, y;

void __fastcall TForm1::FormActivate(TObject *Sender)
{
  WindowState = wsMaximized;
  Timer1->Interval = 50;
  randomize();
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  x = random(Screen->Width - 10);
  y = random(Screen->Height - 10);
  Canvas->Pen->Color = (Graphics::TColor) random(65535);
  Canvas->Pen->Width = random(7);
  int Dx = random(400);
  int Dy = random(400);
  Canvas->RoundRect(x, y, x + Dx, y + Dy, Dx/2, Dy/2);
}
```

Delphi Examples:

```
{
This example draws many rounded rectangles of various sizes
and colors on a form maximized to fill the entire screen:
}
var
  X, Y, DX, DY: Integer;
procedure TForm1.FormActivate(Sender: TObject);
begin
  WindowState := wsMaximized;
  Timer1.Interval := 50;
```

```
    Randomize;
  end;

  procedure TForm1.Timer1Timer(Sender: TObject);
  begin
    X := Random(Screen.Width - 10);
    Y := Random(Screen.Height - 10);
    Canvas.Pen.Color := Random(65535);
    Canvas.Pen.Width := Random(7);
    DX := Random(400);
    DY := Random(400);
    Canvas.RoundRect(X, Y, X + DX, Y + DY, DX div 2, DY div 2);
  end;
```

See Also

Brush (see page 14484)

Ellipse (see page 14457)

Pen (see page 14492)

Rectangle (see page 14468)

73.1.7.4.27 TCanvas.StretchDraw Method

Draws the graphic specified by the Graphic parameter in the rectangle specified by the Rect parameter.

Pascal

```
procedure StretchDraw(const Rect: TRect; Graphic: TGraphic);
```

C++

```
__fastcall StretchDraw(const TRect Rect, TGraphic Graphic);
```

Description

Call StretchDraw to draw a graphic on the canvas so that the image fits in the specified rectangle. StretchDraw calls the Draw (see page 14454) method of the graphic. The graphic object determines how to fit into the rectangle. This may involve changing magnification and/or aspect ratio.

To render the graphic in its natural size, use the Draw (see page 14454) method, instead.

If the graphic is a TBitmap (see page 14396) object, the bitmap is rendered using the value of CopyMode (see page 14487).

Note: If the graphic is an icon, it is not stretched.

C++ Examples:

```
/*
The following example is taken from the custom draw demo. It
shows how the OnCustomDraw event handler draws the
background for the tree view before the items and lines are
drawn.
*/
void __fastcall TCustomDrawForm::TVCustomDraw(TCustomTreeView *Sender,
      const TRect &ARect, bool &DefaultDraw)
{
/*
This event should be used to draw any background colors or
images.  ARect represents the entire client area of the TreeView.
Use the TreeView's canvas to do the drawing.
Note that drawing a background bitmap is not really
supported by CustomDraw, so scrolling can get messy. Best to
```

```
subclass the TreeView and handle scrolling messages.
*/
  if (None1 != NULL)
  {
    if (None1->Checked) //no picture
    {
      TV->Canvas->Brush->Color = BkgColorDialog->Color;
      TV->Canvas->Brush->Style = FBrushStyle;
      TV->Canvas->FillRect(ARect);
    }
    else if (Tile1->Checked) //tile bitmap
    {
      TV->Canvas->Brush->Bitmap = Image1->Picture->Bitmap;
      TV->Canvas->FillRect(ARect);
    }
    else //Stretch across the canvas.
      TV->Canvas->StretchDraw(ARect, Image1->Picture->Bitmap);
  };
  DefaultDraw = FDefaultDraw;
  // Setting DefaultDraw to false here prevents all calls to
  // OnCustomDrawItem.
}
```

Delphi Examples:

```
{
The following example is taken from the custom draw demo. It
shows how the OnCustomDraw event handler draws the
background for the tree view before the items and lines are
drawn.
}
procedure TCustomDrawForm.TVCustomDraw(Sender: TCustomTreeView; const ARect: TRect;
  var DefaultDraw: Boolean);
begin
{
This event should be used to draw any background colors or
images.  ARect represents the entire client area of the TreeView.
Use the TreeView's canvas to do the drawing.
Note that drawing a background bitmap is not really
supported by CustomDraw, so scrolling can get messy. Best to
subclass the TreeView and handle scrolling messages.
}
  with TV.Canvas do
  begin
    if None1 <> nil then
    begin
      if None1.Checked then //no picture
      begin
        Brush.Color := BkgColorDialog.Color;
        Brush.Style := FBrushStyle;
        FillRect(ARect);
      end else
        if Tile1.Checked then //tile bitmap
        begin
          Brush.Bitmap := Image1.Picture.Bitmap;
            FillRect(ARect);
          end else //Stretch across the canvas.
              StretchDraw(ARect, Image1.Picture.Bitmap);
    end;
  end;
  DefaultDraw := FDefaultDraw;
  // Setting DefaultDraw to false here prevents all calls to
  // OnCustomDrawItem.
end;
```

See Also

CopyMode (see page 14487)

CopyRect (see page 14452)

Draw (see page 14454)

73.1.7.4.28 TCanvas.TextExtent Method

Returns the width and height, in pixels, of a string rendered in the current font.

Pascal

```
function TextExtent(const Text: string): TSize;
```

C++

```
__fastcall TSize TextExtent(const AnsiString Text);
```

Description

Use TextExtent to determine the space a string will occupy in the image. Other elements in the image such as lines, boxes, or additional lines of text can be positioned to accommodate the size of the text. TextExtent returns the width and height of the rectangle that bounds the text on the canvas.

To check only the height, call TextHeight (see page 14474). To check only the width, call TextWidth (see page 14481).

See Also

Font (see page 14488)

TextHeight (see page 14474)

TextOut (see page 14475)

TextWidth (see page 14481)

73.1.7.4.29 TCanvas.TextHeight Method

Returns the height, in pixels, of a string rendered in the current font.

Pascal

```
function TextHeight(const Text: string): Integer;
```

C++

```
__fastcall int TextHeight(const AnsiString Text);
```

Description

Use TextHeight to determine the height a string will occupy in the image. Other elements in the image such as lines, boxes, or additional lines of text can be positioned to accommodate the height of the text. TextHeight returns the height of the font, the ascenders plus the descenders, not the actual height of the specified string. If Font.Height is negative, then TextHeight is equal to abs(Font.Height) plus the internal leading of the font. If Font.Height is positive, then TextHeight equals Font.Height.

TextHeight returns the same value as TextExtent (see page 14474)(Text).cy.

C++ Examples:

```
/*
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
```

```
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
     AnsiString s = PageControl1->Pages[i]->Caption;
     if (Canvas->TextWidth(s) * 2 > PageControl1->TabWidth)
        PageControl1->TabWidth = Canvas->TextWidth(s) * 2;
      if (Canvas->TextHeight(s) * 2 > PageControl1->TabHeight)
        PageControl1->TabHeight = Canvas->TextHeight(s) * 2;
  }
}
```

Delphi Examples:

```
{
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  with PageControl1 do
  begin
    for i := 0 to PageCount - 1 do
    begin
      if (Canvas.TextWidth(Pages[i].Caption) * 2) > TabWidth then
        TabWidth := Canvas.TextWidth(Pages[i].Caption) * 2;
      if (Canvas.TextHeight(Pages[i].Caption) * 2) > TabHeight then
        TabHeight := Canvas.TextHeight(Pages[i].Caption) * 2;
    end;
  end;
end;
```

See Also

Font (see page 14488)

TextExtent (see page 14474)

TextOut (see page 14475)

TextWidth (see page 14481)

73.1.7.4.30 TCanvas.TextOut Method

Writes a string on the canvas, starting at the point (X,Y), and then updates the PenPos (see page 14494) to the end of the string.

Pascal

```
procedure TextOut(X: Integer; Y: Integer; const Text: string);
```

C++

```
__fastcall TextOut(int X, int Y, const AnsiString Text);
```

Description

Use TextOut to write a string onto the canvas. The string will be written using the current value of Font (see page 14488). Use the TextExtent (see page 14474) method to determine the space occupied by the text in the image. To write only the text that fits within a clipping rectangle, use TextRect (see page 14480) instead.

After a call to TextOut, the PenPos (see page 14494) property indicates the point at the top right of the text on the canvas.

C++ Examples:

```
/*
Add a THeaderControl to the form and call
HeaderControl1DrawSection for the OnDrawSection event.
*/
void __fastcall TForm1::HeaderControl1DrawSection(THeaderControl *HeaderControl,
      THeaderSection *Section, const TRect &Rect, bool Pressed)
{
  if (Pressed)
    HeaderControl->Canvas->Font->Color = clRed;
  else
    HeaderControl->Canvas->Font->Color = clBlue;
  HeaderControl->Canvas->TextOut(Rect.Left + HeaderControl->Canvas->Font->Size, Rect.Top + 2,
"Custom " + IntToStr(Section->Index));
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  THeaderSection *Section;
  for (int i = 0; i <= 4; i++)
  {
    Section = HeaderControl1->Sections->Add();
    Section->Text = "Section " + IntToStr(i);
    Section->MinWidth =
      (Section->Text).Length() *
      HeaderControl1->Font->Size;
    // Owner draw every other section
    if (i % 2 == 0)
      Section->Style = hsOwnerDraw;
    else
      Section->Style = hsText;
  }
}
/*
The following code adds an object to MyList if it isn't
already in the list.
*/
typedef struct AList
{
  int I;
  char C;
} TAList;

typedef AList* PAList;

TList *MyList;
PAList ARecord1, ARecord2;

void __fastcall DisplayTList(TList *theList)
{
  PAList AStruct;
  // Go through the list, writing the elements to the
  // canvas of a paintbox component.
  int Y = 10; // position on canvas
  for (int i = 0; i < theList->Count; i++)
  {
    AStruct = (PAList) theList->Items[i];
```

```
    Form1->PaintBox1->Canvas->TextOut(10, Y, IntToStr(AStruct->I));
    Y += 30;  // Increment Y Value again
    Form1->PaintBox1->Canvas->TextOut(10, Y, AStruct->C);
    Y += 30;  //Increment Y Value
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (MyList->IndexOf(ARecord1) == -1) MyList->Add(ARecord1);
  DisplayTList(MyList);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  if (MyList->IndexOf(ARecord2) == -1) MyList->Add(ARecord2);
  DisplayTList(MyList);
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  if (MyList->Count != 0)
  {
    for (int B = MyList->Count - 1; B >= 0; B--)
    {
      PAList ARecord;
      ARecord = PAList(MyList->Items[B]);
      MyList->Remove(ARecord);
    }
  }
  Form1->PaintBox1->Repaint();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyList = new TList;
  ARecord1 = new TAList;
  ARecord1->I = 100;
  ARecord1->C = 'Z';
  ARecord2 = new TAList;
  ARecord2->I = 200;
  ARecord2->C = 'X';
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  // Clean up - must free memory for the items as well as the list
  for (int i = 0; i < MyList->Count; i++)
  {
    PAList ARecord = (PAList) MyList->Items[i];
    delete ARecord;
  }
  delete MyList;

}
```

Delphi Examples:

```
{
Add a THeaderControl to the form and call
HeaderControl1DrawSection for the OnDrawSection event.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  HeaderSection: THeaderSection;
  I: Integer;
begin
```

```
    for I := 0 to 4 do
    begin
      HeaderSection := HeaderControl1.Sections.Add;
      HeaderSection.Text := 'Text Section ' + IntToStr(I);
      HeaderSection.MinWidth := length(HeaderSection.Text) * Font.Size;
      // Owner draw every other section
      if (I mod 2 = 0) then
        HeaderSection.Style := hsOwnerDraw
      else
        HeaderSection.Style := hsText;
    end;
  end;

  procedure TForm1.HeaderControl1DrawSection(HeaderControl: THeaderControl;
    Section: THeaderSection; const Rect: TRect; Pressed: Boolean);
  begin
    with HeaderControl.Canvas do
    begin
      // highlight pressed sections
      if Pressed then
        Font.Color := clRed
      else
        Font.Color := clBlue;
      TextOut(Rect.Left + Font.Size, Rect.Top + 2, 'Owner Drawn text');
    end;
  end;
  {
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
{
The following code adds an object to MyList if it isn't
already in the list.
}
type
  PMyList = ^AList;
  AList = record
    I: Integer;
    C: Char;
  end;
var
  Form1: TForm1;
  MyList: TList;
  ARecord1, ARecord2: PMyList;

implementation

{$R *.dfm}

procedure DisplayTList(TheList: TList);
var
  ARecord: PMyList;
  B: Byte;
  Y: Word;
begin
    { Now paint the items onto the paintbox}
    Y := 10;             {Variable used in TextOut function}
    for B := 0 to (TheList.Count - 1) do
    begin
      ARecord := TheList.Items[B];
      Form1.PaintBox1.Canvas.TextOut(10, Y, IntToStr(ARecord^.I)); {Display I}
      Y := Y + 30;  {Increment Y Value again}
      Form1.PaintBox1.Canvas.TextOut(10, Y, ARecord^.C);  {Display C}
      Y := Y + 30;  {Increment Y Value}
    end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  if (MyList.IndexOf(ARecord1) = -1) then MyList.Add(ARecord1);
  DisplayTList(MyList);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  if (MyList.IndexOf(ARecord2) = -1) then MyList.Add(ARecord2);
  DisplayTList(MyList);
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  ARecord: PMyList;
```

```
    B: Integer;
  begin
    if (MyList.Count <> 0) then
    begin
      for B := (MyList.Count - 1) downto 0 do
      begin
        ARecord := MyList.Items[B];
        MyList.Remove(ARecord);
      end;
    end;
    Form1.PaintBox1.Repaint;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    MyList := TList.Create;
    New(ARecord1);
    ARecord1^.I := 100;
    ARecord1^.C := 'Z';
    New(ARecord2);
    ARecord2^.I := 200;
    ARecord2^.C := 'X';
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    Dispose(ARecord1);
    Dispose(ARecord2);
    MyList.Free;
  end;
```

See Also

Font (see page 14488)

TextExtent (see page 14474)

TextRect (see page 14480)

TextFlags (see page 14495)

PenPos (see page 14494)

73.1.7.4.31 TextRect Method

This is the overview for the TextRect method overload.

Overload List

	Name	Description
	TCanvas.TextRect (TRect, Integer, Integer, string) (see page 14480)	Writes a string inside a clipping rectangle.
	TCanvas.TextRect (TRect, string, TTextFormat) (see page 14480)	Writes a string inside a clipping rectangle.

73.1.7.4.31.1 TCanvas.TextRect Method (TRect, Integer, Integer, string)

Writes a string inside a clipping rectangle.

Pascal

```
procedure TextRect(Rect: TRect; X: Integer; Y: Integer; const Text: string); overload;
procedure TextRect(var Rect: TRect; var Text: string; TextFormat: TTextFormat = []); overload;
```

C++

```
__fastcall TextRect(TRect Rect, int X, int Y, const AnsiString Text);
__fastcall TextRect(TRect Rect, AnsiString Text, TTextFormat TextFormat = []);
```

Description

Use TextRect to write a string within a limited rectangular region. Any portions of the string that fall outside the rectangle passed in the Rect parameter are clipped and don't appear. The upper left corner of the text is placed at the point (X, Y).

C++ Examples:

```
/*
The following code displays the text "Hello, world!" in a
rectangle defined by the coordinates (10, 10) and (100, 100).
After displaying the text with the TextRect method, the code
draws a black, vertical line frame around the rectangle.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TRect TheRect = Rect(10,10,100,100);
  Canvas->TextRect(TheRect, 10, 10, "Hello World");
  Canvas->Brush->Color = clBlack;
  Canvas->FrameRect(TheRect);
}
```

Delphi Examples:

```
{
The following code displays the text "Hello, world!" in a
rectangle defined by the coordinates (10, 10) and (100, 100).
After displaying the text with the TextRect method, the code
draws a black, vertical line frame around the rectangle.
}
procedure TForm1.Button1Click(Sender: TObject);
var TheRect: TRect;
begin
  TheRect := Rect(10,10,100,100);
  Form1.Canvas.TextRect(TheRect,10,10,'Hello, world!');
  Form1.Canvas.Brush.Color := clBlack;
  Form1.Canvas.FrameRect(TheRect);
end;
```

See Also

ClipRect (see page 14486)

Font (see page 14488)

TextOut (see page 14475)

TextFlags (see page 14495)

73.1.7.4.32 TCanvas.TextWidth Method

Returns the width, in pixels, of a string rendered in the current font.

Pascal

```
function TextWidth(const Text: string): Integer;
```

C++

```
__fastcall int TextWidth(const AnsiString Text);
```

Description

Use TextWidth to determine the length a string will occupy in the image. TextWidth indicates whether a given string will fit in the

available space. Other graphical elements in the image such as lines, or additional strings can be positioned to accommodate the width of the text.

TextWidth returns the same value as TextExtent (see page 14474)(Text).cx.

C++ Examples:

```
/*
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
     AnsiString s = PageControl1->Pages[i]->Caption;
     if (Canvas->TextWidth(s) * 2 > PageControl1->TabWidth)
        PageControl1->TabWidth = Canvas->TextWidth(s) * 2;
      if (Canvas->TextHeight(s) * 2 > PageControl1->TabHeight)
        PageControl1->TabHeight = Canvas->TextHeight(s) * 2;
  }
}
```

Delphi Examples:

```
{
This project requires a TPageControl and several TTabPages.
To add new pages, right-click the TPageControl and select
New Page.  During the form creation the width and height of
all tabs on the Page Control becomes double the largest
previous value. So if there are three tabs with captions that
are 12, 23, and 19 characters long, the TabWidth would be the
number of pixels required to fit 46 characters.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  with PageControl1 do
  begin
    for i := 0 to PageCount - 1 do
    begin
      if (Canvas.TextWidth(Pages[i].Caption) * 2) > TabWidth then
        TabWidth := Canvas.TextWidth(Pages[i].Caption) * 2;
      if (Canvas.TextHeight(Pages[i].Caption) * 2) > TabHeight then
        TabHeight := Canvas.TextHeight(Pages[i].Caption) * 2;
    end;
  end;
end;
```

See Also

Font (see page 14488)

TextExtent (see page 14474)

TextHeight (see page 14474)

TextOut (see page 14475)

TextRect (see page 14480)

73.1.7.4.33 TCanvas.TryLock Method

Locks the canvas if it is currently unlocked.

Pascal

```
function TryLock: Boolean;
```

C++

```
__fastcall Boolean TryLock();
```

Description

Call TryLock to prevent other threads from writing on the surface of the canvas until the Unlock (see page 14483) method is called. Unlike calls to the Lock (see page 14463) method, which nest so that every call to Lock (see page 14463) must be matched with a subsequent call to Unlock (see page 14483), TryLock need only be matched by a call to Unlock (see page 14483) if the canvas was not already locked.

If the canvas is unlocked, TryLock locks the canvas, sets the LockCount (see page 14492) property to 1, and returns true. If the canvas is already locked, TryLock returns false without making any changes.

Use TryLock before attempting changes to the canvas that could interfere with other code that is protected by a Lock (see page 14463)/Unlock (see page 14483) pair of method calls.

See Also

Lock (see page 14463)

LockCount (see page 14492)

Unlock (see page 14483)

73.1.7.4.34 TCanvas.Unlock Method

Decreases the LockCount (see page 14492) property, allowing other threads to execute when LockCount (see page 14492) reaches 0.

Pascal

```
procedure Unlock;
```

C++

```
__fastcall Unlock();
```

Description

Call Unlock at the end of a critical section begun with a call to the Lock (see page 14463) method. Every call to Lock (see page 14463) must be matched by a call to Unlock.

See Also

Lock (see page 14463)

LockCount (see page 14492)

TryLock (see page 14483)

TThread (see page 3108)

Locking Objects

73.1.7.5 TCanvas Properties

The properties of the TCanvas class are listed here.

Public Properties

	Name	Description
R	CanvasOrientation (see page 14485)	Determines the orientation of the canvas as left-to-right or right-to-left.
R	ClipRect (see page 14486)	Read-only property that specifies the boundaries of the clipping rectangle.
	Handle (see page 14489)	Specifies the handle for this canvas.
R	LockCount (see page 14492)	Indicates the number of times the canvas has been locked to prevent interference from other threads.
	PenPos (see page 14494)	Specifies the current drawing position of the Pen (see page 14492).
	Pixels (see page 14494)	Specifies the color of the pixels within the current ClipRect (see page 14486).
	TextFlags (see page 14495)	Specifies how text is written to the canvas.

Published Properties

	Name	Description
	Brush (see page 14484)	Determines the color and pattern for filling graphical shapes and backgrounds.
	CopyMode (see page 14487)	Specifies how a graphical image is copied onto the canvas.
	Font (see page 14488)	Specifies the font to use when writing text on the image.
	Pen (see page 14492)	Specifies the kind of pen the canvas uses for drawing lines and outlining shapes.

73.1.7.5.1 TCanvas.Brush Property

Determines the color and pattern for filling graphical shapes and backgrounds.

Pascal

```
property Brush: TBrush;
```

C++

```
__property TBrush Brush;
```

Description

Set the Brush property to specify the color and pattern to use when drawing the background or filling in graphical shapes. The value of Brush is a TBrush (see page 14430) object. Set the properties of the TBrush (see page 14430) object to specify the color and pattern or bitmap to use when filling in spaces on the canvas.

Note: Setting the Brush property assigns the specified TBrush (see page 14430) object, rather than replacing the current TBrush (see page 14430) object.

C++ Examples:

```
/*
The following code loads a bitmap from a file and assigns it
to the Brush of the Canvas of Form1:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *BrushBmp = new Graphics::TBitmap;
  try
  {
    BrushBmp->LoadFromFile("../bm1.BMP");
    Form1->Canvas->Brush->Bitmap = BrushBmp;
    Form1->Canvas->FillRect(Rect(0,0,100,100));
  }
```

```
  __finally
  {
    Form1->Canvas->Brush->Bitmap = NULL;
    delete BrushBmp;
  }
}
```

Delphi Examples:

```
{
The following code loads a bitmap from a file and assigns it
to the Brush of the Canvas of Form1:
}
var
  Bitmap: TBitmap;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Bitmap := TBitmap.Create;
  try
    Bitmap.LoadFromFile('bm1.BMP');
    Form1.Canvas.Brush.Bitmap := Bitmap;
    Form1.Canvas.FillRect(Rect(0,0,100,100));
  finally
    Form1.Canvas.Brush.Bitmap := nil;
    Bitmap.Free;
  end;
end;
```

See Also

Chord (see page 14451)

Ellipse (see page 14457)

FillRect (see page 14458)

FloodFill (see page 14459)

FrameRect (see page 14460)

Pie (see page 14464)

Rectangle (see page 14468)

RoundRect (see page 14471)

73.1.7.5.2 TCanvas.CanvasOrientation Property

Determines the orientation of the canvas as left-to-right or right-to-left.

Pascal

```
property CanvasOrientation: TCanvasOrientation;
```

C++

```
__property TCanvasOrientation CanvasOrientation;
```

Description

CanvasOrientation is a read-only property used for bi-directional support. It indicates whether the orientation of the canvas is left-to-right (coLeftToRight) or right-to-left (coRightToLeft). The default is coLeftToRight, meaning that the 0,0 coordinate is at the top left corner of the canvas. In Middle eastern locales the orientation may be coRightToLeft, meaning that the 0,0 coordinate is at the top right corner of the canvas. This property is important for grid controls because it changes the grid's canvas orientation during drawing.

See Also

BiDiMode (see page 5803)

73.1.7.5.3 TCanvas.ClipRect Property

Read-only property that specifies the boundaries of the clipping rectangle.

Pascal

```
property ClipRect: TRect;
```

C++

```
__property TRect ClipRect;
```

Description

Use ClipRect to determine where the canvas needs painting. ClipRect limits the drawing region of the canvas so that any drawing that occurs at coordinates outside the ClipRect is clipped and does not appear in the image.

When handling a form's OnPaint event, the canvas' ClipRect property is set to the rectangle that needs to be painted. Portions of the image that do not overlap the ClipRect do not need to be drawn. Thus, OnPaint routines can use the value of ClipRect to optimize painting, speeding the overall performance of the application.

Note: ClipRect can be modified using the SelectClipRgn Win32 API.

C++ Examples:

```
/*
This example creates a region and selects this region as the
clipping rectangle for the Image component's canvas. It then
sets the canvas's brush color to red and calls FillRect
using the ClipRect as the area to fill. Lastly, the ClipRect
is reset to the original value that it contained by calling
SelectClipRect with nil as the second parameter and deletes
the region.  Place a TImage object in the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    HRGN MyRgn;

    MyRgn = ::CreateRectRgn(100,100,200,200);
    ::SelectClipRgn(Image1->Canvas->Handle,MyRgn);
    Image1->Canvas->Brush->Color = clRed;
    Image1->Canvas->FillRect(Image1->Canvas->ClipRect);
    Image1->Invalidate();
    ::SelectClipRgn(Image1->Canvas->Handle,NULL);
    ::DeleteObject(MyRgn);
}
```

Delphi Examples:

```
{
This example creates a region and selects this region as the
clipping rectangle for the Image component's canvas. It then
sets the canvas's brush color to red and calls FillRect
using the ClipRect as the area to fill. Lastly, the ClipRect
is reset to the original value that it contained by calling
SelectClipRect with nil as the second parameter and deletes
the region.  Place a TImage object in the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
    MyRgn: HRGN;
begin
    MyRgn := CreateRectRgn(100,100,200,200);
    SelectClipRgn(Image1.Canvas.Handle,MyRgn);
    Image1.Canvas.Brush.Color := clRed;
    Image1.Canvas.FillRect(Image1.Canvas.ClipRect);
    Image1.Invalidate;
    SelectClipRgn(Image1.Canvas.Handle, HRGN(nil));
    DeleteObject(MyRgn);
end;
```

See Also

OnPaint (see page 13975)

73.1.7.5.4 TCanvas.CopyMode Property

Specifies how a graphical image is copied onto the canvas.

Pascal

```
property CopyMode: TCopyMode;
```

C++

```
__property TCopyMode CopyMode;
```

Description

Set CopyMode to affect the way graphical images are drawn onto the canvas. The CopyMode is used when copying an image from another canvas using the CopyRect (see page 14452) method. CopyMode is also used by TBitmap (see page 14396) objects when they draw themselves to a canvas.

Use CopyMode to achieve a variety of affects when rendering an image. Achieve special effects like merged images and making parts of a bitmap transparent by combining multiple images with different CopyModes.

C++ Examples:

```
/*
The following example uses the CopyMode of an image's canvas
to blank out the image when the user chooses the "Cut" menu
item.  This example requires a Main menu with a Copy and a
Cut menu item that have their OnClick error handlers populated.
Note: CopyMode and CopyRect are members of the destination
canvas.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
    TRect SrcRect, DstRect;
    Image2->Canvas->CopyMode = cmSrcCopy;
    DstRect = Rect(0, 0, Image2->Width, Image2->Height);
    SrcRect = Rect(0, 0, Image1->Width, Image1->Height);
    Image2->Canvas->CopyRect(DstRect, Image1->Canvas, SrcRect);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
 TRect r;
 Copy1Click(Sender); // Do the same thing as the copy menu item.
 Image1->Canvas->CopyMode = cmWhiteness;
 r = Rect(0, 0, Image1->Width, Image1->Height);
 Image1->Canvas->CopyRect(r, Image1->Canvas, r);
 Image1->Canvas->CopyMode = cmSrcCopy; // Restore the copy mode.
}
```

Delphi Examples:

```
{
The following example uses the CopyMode of an image's canvas
to blank out the image when the user chooses the "Cut" menu
item.  This example requires a Main menu with a Copy and a
Cut menu item that have their OnClick error handlers populated.
Note: CopyMode and CopyRect are members of the destination
canvas.
}
procedure TForm1.Copy1Click(Sender: TObject);
var
  DstRect, SrcRect: TRect;
begin
  with Image2.Canvas do
  begin
    CopyMode := cmSrcCopy;
    DstRect := Rect(0, 0, Image2.Width, Image2.Height);
    SrcRect := Rect(0, 0, Image1.Width, Image1.Height);
    CopyRect(DstRect, Image1.Canvas, SrcRect);
  end;
end;

procedure TForm1.Cut1Click(Sender: TObject);
var
  ARect: TRect;
begin
  Copy1Click(Sender); { do the same thing as the copy menu item }
  with Image1.Canvas do
  begin
    CopyMode := cmWhiteness;
    ARect := Rect(0, 0, Image1.Width, Image1.Height);
    CopyRect(ARect, Image1.Canvas, ARect);
    CopyMode := cmSrcCopy; { restore the copy mode }
  end;
end;
```

See Also

BrushCopy (see page 14449)

CopyRect (see page 14452)

Draw (see page 14454)

StretchDraw (see page 14472)

73.1.7.5.5 TCanvas.Font Property

Specifies the font to use when writing text on the image.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

Set Font to specify the font to use for writing text on the image. The value of Font is a T Font object. Set the properties of the T Font object to specify the font face, color, size, style, and any other aspects of the font.

The Canvas.Font property is only guaranteed to equal the Font property if you have an owner-drawn listbox (where you are expected to use the canvas). Even in that case, it is only guaranteed to equal the Font property once the first paint message is underway. If you use a canvas inside an ownerdraw event, everything should work as expected. Using the canvas for a reason outside the scope of its intended usage may give unpredictable results.

Note: Setting the Font property assigns the specified T Font object, rather than replacing the current T Font object.

See Also

TextOut (see page 14475)

TextRect (see page 14480)

TFont (see page 14497)

Assign (see page 14500)

73.1.7.5.6 TCanvas.Handle Property

Specifies the handle for this canvas.

Pascal

```
property Handle: HDC;
```

C++

```
__property HDC Handle;
```

Description

The Handle property specifies the Windows GDI handle to the device context for this canvas.

Set Handle to the HDC for the device context the canvas must draw into. When a windowed control responds to a Windows paint message, the HDC for drawing is passed in to the PaintWindow method. In other cases, an HDC can be obtained for a window by calling the GetDeviceContext method of a control. Additionally, Windows provides API calls to obtain an HDC for a printer or for a memory image.

Read the Handle property to supplement the drawing services provided by the TCanvas (see page 14441) object with API calls that require a handle to a device context. Most of the Windows GDI calls require an HDC.

TCanvas (see page 14441) does not own the HDC. Applications must create an HDC and set the Handle property. Applications must release the HDC when the canvas no longer needs it. Setting the Handle property of a canvas that already has a valid HDC will not automatically release the initial HDC.

Note: Some descendants of TCanvas (see page 14441), such as TControlCanvas, do own the HDC. Do not set the Handle property for these objects. They fetch and free their own Handle.

C++ Examples:

```
/*
The GetFormImage allows you to easily obtain a bitmap of a
form. The following method can be added to a custom
TWinControl descendant to save itself as a bitmap.  Check the
produced BMP file to confirm.  Click the button and them
double click on the newly created foo.bmp file to view.
*/
class TMyControl : public TColorListBox
{
__published:    // IDE-managed Components
    void __fastcall SaveAsBmp(TFileName fileName);
private:    // User declarations
public:     // User declarations
    __fastcall TMyControl(TComponent* Owner);
};

TForm1 *Form1;
TMyControl *MyControl1;
```

```
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
__fastcall TMyControl::TMyControl(TComponent* Owner)
    : TColorListBox(Owner)
{
}

typedef struct {
  TLogPalette lpal;
  TPaletteEntry dummy[256];
} LogPal;

void __fastcall TMyControl::SaveAsBmp(TFileName fileName)
{
  TCanvas *tempCanvas = new TCanvas;
  try
  {
    HWND notUsed;
    tempCanvas->Handle = GetDeviceContext(notUsed);
    TImage *image2save = new TImage(Form1);
    try
    {
      image2save->Height = Height;
      image2save->Width = Width;
      TRect destRect = Rect(0,0,Width,Height);
      TRect sourceRect = destRect;
      image2save->Canvas->CopyRect(destRect, tempCanvas, sourceRect);
      LogPal SysPal;
      SysPal.lpal.palVersion = 0x300;
      SysPal.lpal.palNumEntries = 256;
      GetSystemPaletteEntries(tempCanvas->Handle,0,256,SysPal.lpal.palPalEntry);
      image2save->Picture->Bitmap->Palette = CreatePalette((const tagLOGPALETTE
*)&SysPal.lpal);
      image2save->Picture->SaveToFile(fileName);
    }
    __finally
    {
     delete image2save;
    }
  }
  __finally
  {
    delete tempCanvas;
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  MyControl1->SaveAsBmp("../foo.bmp");
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyControl1 = new TMyControl(Form1);
  MyControl1->Parent = Form1;
  MyControl1->Visible  = true;
}
```

Delphi Examples:

```
{
The GetFormImage allows you to easily obtain a bitmap of a
form. The following method can be added to a custom
TWinControl descendant to save itself as a bitmap.  Check the
```

```
  produced BMP file to confirm.  Click the button and them
  double click on the newly created foo.bmp file to view.
  }
  type
    TMyControl = class(TColorListBox)
      procedure SaveAsBmp(fileName: TFileName);
    end;
    TForm1 = class(TForm)
      Button1: TButton;
      procedure Button1Click(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      { Public declarations }
    end;

  var
    Form1: TForm1;
    MyControl1: TMyControl;
  implementation

  {$R *.dfm}

  type
    LogPal = record
    lpal : TLogPalette;
    dummy:Array[0..255] of TPaletteEntry;
    end;

  procedure TMyControl.SaveAsBmp(fileName: TFileName);
  var
    Source: TComponent;
    SysPal : LogPal;
    tempCanvas: TCanvas;
    sourceRect, destRect: TRect;
    image2save: TImage;
    notUsed: HWND;
  begin
    tempCanvas := TCanvas.Create;
    try
      tempCanvas.Handle := GetDeviceContext(notUsed);
      image2save:=TImage.create(self);
      try
        with image2save do
        begin
          Height := Self.Height;
          Width :=  Self.Width;
          destRect := Rect(0,0,Width,Height);
          sourceRect := destRect;
          Canvas.CopyRect(destRect,tempCanvas,sourceRect);
          SysPal.lPal.palVersion:=$300;
          SysPal.lPal.palNumEntries:=256;
          GetSystemPaletteEntries(
            tempCanvas.Handle,0,256,SysPal.lpal.palPalEntry);
          Picture.Bitmap.Palette:= CreatePalette(Syspal.lpal);
        end;
        image2save.Picture.SaveToFile(fileName);
      finally
       image2save.Free;
      end;
    finally
      tempCanvas.Free;
    end;
  end;
  procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  MyControl1.SaveAsBmp('foo.bmp');
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyControl1:= TMyControl.Create(Form1);
  MyControl1.Parent:= Form1;
  MyControl1.visible := true;
end;
```

See Also

GetDeviceContext (see page 5760)

PaintWindow (see page 6169)

73.1.7.5.7 TCanvas.LockCount Property

Indicates the number of times the canvas has been locked to prevent interference from other threads.

Pascal

```
property LockCount: Integer;
```

C++

```
__property int LockCount;
```

Description

Read LockCount to determine whether the canvas is locked. Whenever the Lock (see page 14463) method is called to block other threads from interfering with the canvas, LockCount increases. Whenever the Unlock (see page 14483) method is called at the end of a critical section, LockCount decreases. When LockCount is 0, other threads may draw on the canvas.

See Also

Lock (see page 14463)

TThread (see page 3108)

Unlock (see page 14483)

73.1.7.5.8 TCanvas.Pen Property

Specifies the kind of pen the canvas uses for drawing lines and outlining shapes.

Pascal

```
property Pen: TPen;
```

C++

```
__property TPen Pen;
```

Description

Set Pen to specify the pen to use for drawing lines and outlining shapes in the image. The value of Pen is a TPen (see page 14571) object. Set the properties of the TPen (see page 14571) object to specify the color, style, width, and mode of the pen.

Note: Setting the Pen property assigns the specified TPen (see page 14571) object, rather than replacing the current TPen (see page 14571) object.

C++ Examples:

```
/*
This example paints a white five-pointed star in a paintbox
control:
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
  TPaintBox *pPB = (TPaintBox *)Sender;
  TPoint points[6];
  pPB->Canvas->Pen->Color = clWhite;
  points[0].x = 40;
  points[0].y = 10;
  points[1].x = 20;
  points[1].y = 60;
  points[2].x = 70;
  points[2].y = 30;
  points[3].x = 10;
  points[3].y = 30;
  points[4].x = 60;
  points[4].y = 60;
  points[5].x = 40;
  points[5].y = 10;
  pPB->Canvas->Polyline(points,5);
}
```

Delphi Examples:

```
{
This example paints a white five-pointed star in a paintbox
control:
}
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
  with Sender as TPaintBox do
  begin
    Canvas.Pen.Color := clWhite;
    Canvas.Polyline([Point(40, 10), Point(20, 60), Point(70, 30),
    Point(10, 30), Point(60, 60), Point(40, 10)]);
  end;
end;
```

See Also

Arc (see page 14448)

Chord (see page 14451)

Ellipse (see page 14457)

LineTo (see page 14462)

MoveTo (see page 14463)

PenPos (see page 14494)

Pie (see page 14464)

Polygon (see page 14466)

Polyline (see page 14467)

Rectangle (see page 14468)

RoundRect (see page 14471)

TPen (see page 14571)

Assign (see page 14574)

73.1.7.5.9 TCanvas.PenPos Property

Specifies the current drawing position of the Pen (see page 14492).

Pascal

```
property PenPos: TPoint;
```

C++

```
__property TPoint PenPos;
```

Description

Read PenPos to learn the current drawing position of the Pen (see page 14492). This is the starting point of a line drawn by the LineTo (see page 14462) method. Setting the PenPos property is equivalent to calling the MoveTo (see page 14463) method.

See Also

LineTo (see page 14462)

MoveTo (see page 14463)

Pen (see page 14492)

73.1.7.5.10 TCanvas.Pixels Property

Specifies the color of the pixels within the current ClipRect (see page 14486).

Pascal

```
property Pixels [X, Y: Integer]: TColor;
```

C++

```
__property TColor Pixels[int X, Y];
```

Description

Read Pixels to learn the color on the drawing surface at a specific pixel position within the current clipping region. If the position is outside the clipping rectangle, reading the value of Pixels returns -1.

Write Pixels to change the color of individual pixels on the drawing surface. Use Pixels for detailed effects on an image. Pixels may also be used to determine the color that should be used for the FillRect (see page 14458) method.

Not every device context supports the Pixels property. Reading the Pixels property for such a device context will return a value of -1. Setting the Pixels property for such a device context does nothing.

C++ Examples:

```
/*
This example draws a red line (very slowly) when the user
presses a button. Attach the following code to the button's
OnClick event handler:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 10; i <= 200; i++)
    Canvas->Pixels[i][10] = clRed;
}
```

Delphi Examples:

```
{
This example draws a red line (very slowly) when the user
```

```
  presses a button. Attach the following code to the button's
  OnClick event handler:
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    W: Word;
  begin
      for W := 10 to 200 do
        Canvas.Pixels[W, 10] := clRed;
  end;
```

See Also

ClipRect (see page 14486)

Handle (see page 14489)

FillRect (see page 14458)

73.1.7.5.11 TCanvas.TextFlags Property

Specifies how text is written to the canvas.

Pascal

```
property TextFlags: Longint;
```

C++

```
__property Longint TextFlags;
```

Description

Use TextFlags to influence the way text is drawn when calling the TextOut (see page 14475) or TextRect (see page 14480) method. TextFlags is an integer made by combining any of the following constants:

Constant	Meaning
ETO_CLIPPED	Only the text that fits within a specified rectangle is displayed. This flag is automatically added when calling the TextRect (see page 14480) method. When writing text using TextOut (see page 14475) method, this flag has no effect because the drawing boundaries are determined by the TextExtent (see page 14474) property.
ETO_OPAQUE	The text is drawn with an opaque background color. Using this flag improves performance but does not allow images behind the text rectangle to show.
ETO_RTLREADING	Text strings are displayed from right to left. This is available only with Mideast versions of Windows, where text is written from right to left. Often this is used in combination with a CanvasOrientation (see page 14485) of coRightToLeft.
ETO_GLYPH_INDEX	The text is an array of character placement codes that are parsed directly by the Windows GDI. Glyph indexing only applies to TrueType fonts, but the flag can be used with other fonts to indicate that the GDI should process the text string directly without language processing. See the Windows GDI documentation for more details.
ETO_IGNORELANGUAGE	This is an undocumented Microsoft flag. Check updates of Microsoft documentation for when information is provided.
ETO_NUMERICSLOCAL	All numbers in the text are displayed using digits specific to the current locale.
ETO_NUMERICSLATIN	All numbers in the text are displayed using European digits.
ETO_PDY	This flag is only relevant when using the Windows API ExtTextOut. When this is set, the array pointed to by lpDx contains pairs of values. The first value of each pair is, as usual, the distance between origins of adjacent character cells, but the second value is the displacement along the vertical direction of the font.

See Also

CanvasOrientation (see page 14485)

TextExtent (see page 14474)

TextOut (see page 14475)

TextRect (see page 14480)

73.1.7.6 TCanvas Events

The events of the TCanvas class are listed here.

Public Events

	Name	Description
	OnChange (see page 14496)	Occurs when the image has just changed.
	OnChanging (see page 14496)	Occurs just before a change is made to the image.

73.1.7.6.1 TCanvas.OnChange Event

Occurs when the image has just changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the image on the canvas changes. When an application calls one of the drawing methods of TCanvas (see page 14441), the following steps occur.

1An OnChanging (see page 14496) event occurs.

2The TCanvas (see page 14441) method makes a change to the image.

3An OnChange event occurs.

OnChange events occur on changes to the image, not changes to the properties of the canvas. Applications can respond to changes in the Font (see page 14488), Brush (see page 14484), and Pen (see page 14492) properties by assigning OnChange event handlers to the corresponding TFont (see page 14497), TBrush (see page 14430), and TPen (see page 14571) objects. However, TCanvas (see page 14441) assigns its own OnChange event handlers for those objects. Before assigning an OnChange event handler to one of these objects, read the existing event handler. Use that value to call the TCanvas (see page 14441) event handler from the new event handler.

See Also

OnChanging (see page 14496)

OnChange (see page 14536)

73.1.7.6.2 TCanvas.OnChanging Event

Occurs just before a change is made to the image.

Pascal

```
property OnChanging: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChanging;
```

Description

Write an OnChanging event handler to take specific action before the image on the canvas changes. When an application calls one of the drawing methods of TCanvas (see page 14441), the following steps occur.

1An OnChanging event occurs.

2The TCanvas (see page 14441) method makes a change to the image.

3An OnChange (see page 14496) event occurs.

See Also

OnChange (see page 14496)

73.1.8 TFont Class

TFont encapsulates a system font.

Class Hierarchy

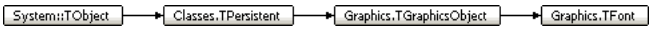

Pascal

```
TFont = class(TGraphicsObject);
```

C++

```
class TFont : public TGraphicsObject;
```

File

Graphics

Description

TFont describes font characteristics used when displaying text. TFont defines a set of characters by specifying the height, font family (typeface), attributes (such as bold or italic) and so on.

TFont encapsulates a windows logical font.

See Also

TTextAttributes (see page 4982)

Using the Main VCL

73.1.8.1 TFont Members

The following tables list the members exposed by TFont.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 14502)	Generates an OnChange event when the properties of the graphics object change.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetCharset (see page 14502)	This is GetCharset, a member of class TFont.
	GetHandle (see page 14503)	This is GetHandle, a member of class TFont.
	GetHeight (see page 14503)	This is GetHeight, a member of class TFont.
	GetName (see page 14503)	This is GetName, a member of class TFont.
	GetOrientation (see page 14503)	This is GetOrientation, a member of class TFont.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPitch (see page 14503)	This is GetPitch, a member of class TFont.
	GetSize (see page 14504)	This is GetSize, a member of class TFont.
	GetStyle (see page 14504)	This is GetStyle, a member of class TFont.
	Lock (see page 14534)	Blocks other execution threads from using the associated canvas until the Unlock (see page 14535) method is called.
	SetCharset (see page 14504)	This is SetCharset, a member of class TFont.
	SetColor (see page 14504)	This is SetColor, a member of class TFont.
	SetHandle (see page 14505)	This is SetHandle, a member of class TFont.
	SetHeight (see page 14505)	This is SetHeight, a member of class TFont.
	SetName (see page 14505)	This is SetName, a member of class TFont.
	SetOrientation (see page 14505)	This is SetOrientation, a member of class TFont.
	SetPitch (see page 14505)	This is SetPitch, a member of class TFont.
	SetSize (see page 14506)	This is SetSize, a member of class TFont.
	SetStyle (see page 14506)	This is SetStyle, a member of class TFont.
	Unlock (see page 14535)	Releases the lock on execution threads set by the Lock (see page 14534) method.

Public Events

	Name	Description
	OnChange (see page 14536)	Occurs immediately after the graphics object changes.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 14500)	Copies the properties of another TFont (see page 14497) object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 14499)	Creates and initializes an instance of TFont (see page 14497).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 14500)	Destroys an instance of TFont (see page 14497).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	HandleAllocated (see page 14534)	Indicates whether the TGraphicsObject (see page 14532) object has acquired a handle to the underlying GDI object.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	FontAdapter (see page 14508)	Designates an interface for providing the font information to ActiveX controls.
	Handle (see page 14509)	Provides access to the handle of a system font.
	OwnerCriticalSection (see page 14536)	Points to information about the critical section that the canvas which draws with this graphics object uses to lock out other threads.
	PixelsPerInch (see page 14512)	Indicates the conversion factor between logical inches and the pixels of the device (printer or screen) using the font.

Published Properties

	Name	Description
	Charset (see page 14506)	Specifies the character set of the font.
	Color (see page 14507)	Specifies the color of the text.
	Height (see page 14510)	Specifies the height of the font.
	Name (see page 14510)	Identifies the typeface of the font.
	Orientation (see page 14511)	Specifies text orientation.
	Pitch (see page 14512)	Specifies whether the characters in the font all have the same width.
	Size (see page 14513)	Specifies the height of the font in points.
	Style (see page 14513)	Determines whether the font is normal, italic, underlined, bold, and so on.

73.1.8.2 TFont.Create Constructor

Creates and initializes an instance of TFont (see page 14497).

Pascal

```
constructor Create;
```

C++

```
__fastcall TFont();
```

Description

Call Create to create a font object at runtime.

Create allocates memory for the font resource, and initializes the Color (see page 14507) property to clWindowText (see page 14733). Create also initializes the PixelsPerInch (see page 14512) property based on the screen resolution.

See Also

PixelsPerInch (see page 14512)

73.1.8.3 TFont.Destroy Destructor

Destroys an instance of TFont (see page 14497).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TFont();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TFont (see page 14497) reference is not nil, and only then calls Destroy. Destroy frees the resource identified by the Handle (see page 14509) property.

See Also

Handle (see page 14509)

Free (see page 22300)

73.1.8.4 TFont Methods

The methods of the TFont class are listed here.

Protected Methods

	Name	Description
	Changed (see page 14502)	Generates an OnChange event when the properties of the graphics object change.
	GetCharset (see page 14502)	This is GetCharset, a member of class TFont.
	GetHandle (see page 14503)	This is GetHandle, a member of class TFont.
	GetHeight (see page 14503)	This is GetHeight, a member of class TFont.
	GetName (see page 14503)	This is GetName, a member of class TFont.
	GetOrientation (see page 14503)	This is GetOrientation, a member of class TFont.
	GetPitch (see page 14503)	This is GetPitch, a member of class TFont.
	GetSize (see page 14504)	This is GetSize, a member of class TFont.
	GetStyle (see page 14504)	This is GetStyle, a member of class TFont.
	SetCharset (see page 14504)	This is SetCharset, a member of class TFont.
	SetColor (see page 14504)	This is SetColor, a member of class TFont.
	SetHandle (see page 14505)	This is SetHandle, a member of class TFont.
	SetHeight (see page 14505)	This is SetHeight, a member of class TFont.
	SetName (see page 14505)	This is SetName, a member of class TFont.
	SetOrientation (see page 14505)	This is SetOrientation, a member of class TFont.
	SetPitch (see page 14505)	This is SetPitch, a member of class TFont.
	SetSize (see page 14506)	This is SetSize, a member of class TFont.
	SetStyle (see page 14506)	This is SetStyle, a member of class TFont.

Public Methods

	Name	Description
	Assign (see page 14500)	Copies the properties of another TFont (see page 14497) object.

73.1.8.4.1 TFont.Assign Method

Copies the properties of another TFont (see page 14497) object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to set the properties of the font to match the characteristics of the Source object. If Source is another TFont (see page 14497) object, Assign matches the font characteristics of the source. Assign does not copy the PixelsPerInch (see page 14512) property, so this method can be used to copy a screen font to a printer font, or vice versa.

If the Source parameter is not another TFont (see page 14497) object, Assign calls the inherited method so that the font can be copied from any object that can assign font properties in its AssignTo method.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
```

```
    FontDialog1.Execute;
  end;

  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    myActiveControl := ActiveControl;
  end;

  procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
  begin
    if myActiveControl is TEdit then
      with myActiveControl as TEdit do
         Font.Assign(TFontDialog(Sender).Font)
    else if myActiveControl is TRichEdit then
      with myActiveControl as TRichEdit do
        SelAttributes.Assign(TFontDialog(Sender).Font)
    else
      Beep;
  end;
```

See Also

PixelsPerInch (see page 14512)

AssignTo (see page 2931)

73.1.8.4.2 TFont.Changed Method

Generates an OnChange event when the properties of the graphics object change.

Pascal

```
procedure Changed; override;
```

C++

```
virtual __fastcall Changed();
```

Description

Descendants of TGraphicsObject call Changed after making changes to the properties encapsulated by the TGraphicsObject object.

See Also

OnChange (see page 14536)

73.1.8.4.3 TFont.GetCharset Method

Pascal

```
function GetCharset: TFontCharset;
```

C++

```
__fastcall TFontCharset GetCharset();
```

Description

This is GetCharset, a member of class TFont.

73.1.8.4.4 TFont.GetHandle Method

Pascal

```
function GetHandle: HFont;
```

C++

```
__fastcall HFont GetHandle();
```

Description

This is GetHandle, a member of class TFont.

73.1.8.4.5 TFont.GetHeight Method

Pascal

```
function GetHeight: Integer;
```

C++

```
__fastcall int GetHeight();
```

Description

This is GetHeight, a member of class TFont.

73.1.8.4.6 TFont.GetName Method

Pascal

```
function GetName: TFontName;
```

C++

```
__fastcall TFontName GetName();
```

Description

This is GetName, a member of class TFont.

73.1.8.4.7 TFont.GetOrientation Method

Pascal

```
function GetOrientation: Integer;
```

C++

```
__fastcall int GetOrientation();
```

Description

This is GetOrientation, a member of class TFont.

73.1.8.4.8 TFont.GetPitch Method

Pascal

```
function GetPitch: TFontPitch;
```

C++

```
__fastcall TFontPitch GetPitch();
```

Description

This is GetPitch, a member of class TFont.

73.1.8.4.9 TFont.GetSize Method

Pascal

```
function GetSize: Integer;
```

C++

```
__fastcall int GetSize();
```

Description

This is GetSize, a member of class TFont.

73.1.8.4.10 TFont.GetStyle Method

Pascal

```
function GetStyle: TFontStyles;
```

C++

```
__fastcall TFontStyles GetStyle();
```

Description

This is GetStyle, a member of class TFont.

73.1.8.4.11 TFont.SetCharset Method

Pascal

```
procedure SetCharset(const Value: TFontCharset);
```

C++

```
__fastcall SetCharset(const TFontCharset Value);
```

Description

This is SetCharset, a member of class TFont.

73.1.8.4.12 TFont.SetColor Method

Pascal

```
procedure SetColor(const Value: TColor);
```

C++

```
__fastcall SetColor(const TColor Value);
```

Description

This is SetColor, a member of class TFont.

73.1.8.4.13 TFont.SetHandle Method

Pascal

```
procedure SetHandle(const Value: HFont);
```

C++

```
__fastcall SetHandle(const HFont Value);
```

Description

This is SetHandle, a member of class TFont.

73.1.8.4.14 TFont.SetHeight Method

Pascal

```
procedure SetHeight(const Value: Integer);
```

C++

```
__fastcall SetHeight(const int Value);
```

Description

This is SetHeight, a member of class TFont.

73.1.8.4.15 TFont.SetName Method

Pascal

```
procedure SetName(const Value: TFontName);
```

C++

```
__fastcall SetName(const TFontName Value);
```

Description

This is SetName, a member of class TFont.

73.1.8.4.16 TFont.SetOrientation Method

Pascal

```
procedure SetOrientation(const Value: Integer);
```

C++

```
__fastcall SetOrientation(const int Value);
```

Description

This is SetOrientation, a member of class TFont.

73.1.8.4.17 TFont.SetPitch Method

Pascal

```
procedure SetPitch(const Value: TFontPitch);
```

C++

```
__fastcall SetPitch(const TFontPitch Value);
```

Description

This is SetPitch, a member of class TFont.

73.1.8.4.18 TFont.SetSize Method

Pascal

```
procedure SetSize(const Value: Integer);
```

C++

```
__fastcall SetSize(const int Value);
```

Description

This is SetSize, a member of class TFont.

73.1.8.4.19 TFont.SetStyle Method

Pascal

```
procedure SetStyle(const Value: TFontStyles);
```

C++

```
__fastcall SetStyle(const TFontStyles Value);
```

Description

This is SetStyle, a member of class TFont.

73.1.8.5 TFont Properties

The properties of the TFont class are listed here.

Public Properties

	Name	Description
	FontAdapter (see page 14508)	Designates an interface for providing the font information to ActiveX controls.
	Handle (see page 14509)	Provides access to the handle of a system font.
	PixelsPerInch (see page 14512)	Indicates the conversion factor between logical inches and the pixels of the device (printer or screen) using the font.

Published Properties

	Name	Description
	Charset (see page 14506)	Specifies the character set of the font.
	Color (see page 14507)	Specifies the color of the text.
	Height (see page 14510)	Specifies the height of the font.
	Name (see page 14510)	Identifies the typeface of the font.
	Orientation (see page 14511)	Specifies text orientation.
	Pitch (see page 14512)	Specifies whether the characters in the font all have the same width.
	Size (see page 14513)	Specifies the height of the font in points.
	Style (see page 14513)	Determines whether the font is normal, italic, underlined, bold, and so on.

73.1.8.5.1 TFont.Charset Property

Specifies the character set of the font.

Pascal

```
property Charset: TFontCharset;
```

C++

```
__property TFontCharset Charset;
```

Description

Set Charset to identify the character set of the font. Each typeface (specified by the Name (see page 14510) property) supports one or more character sets. Check the information supplied by the font vendor to determine what values of Charset are valid.

See Also

Name (see page 14510)

73.1.8.5.2 TFont.Color Property

Specifies the color of the text.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to specify the color of the text characters (as opposed to the background color). If Color is set to a dithered color, the text appears in the closest non-dithered value.

C++ Examples:

```
/*
Add a THeaderControl to the form and call
HeaderControl1DrawSection for the OnDrawSection event.
*/
void __fastcall TForm1::HeaderControl1DrawSection(THeaderControl *HeaderControl,
      THeaderSection *Section, const TRect &Rect, bool Pressed)
{
  if (Pressed)
    HeaderControl->Canvas->Font->Color = clRed;
  else
    HeaderControl->Canvas->Font->Color = clBlue;
  HeaderControl->Canvas->TextOut(Rect.Left + HeaderControl->Canvas->Font->Size, Rect.Top + 2,
"Custom " + IntToStr(Section->Index));
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  THeaderSection *Section;
  for (int i = 0; i <= 4; i++)
  {
    Section = HeaderControl1->Sections->Add();
    Section->Text = "Section " + IntToStr(i);
    Section->MinWidth =
      (Section->Text).Length() *
      HeaderControl1->Font->Size;
    // Owner draw every other section
    if (i % 2 == 0)
      Section->Style = hsOwnerDraw;
    else
      Section->Style = hsText;
```

```
  }
}
```

Delphi Examples:

```
{
Add a THeaderControl to the form and call
HeaderControl1DrawSection for the OnDrawSection event.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  HeaderSection: THeaderSection;
  I: Integer;
begin
  for I := 0 to 4 do
  begin
    HeaderSection := HeaderControl1.Sections.Add;
    HeaderSection.Text := 'Text Section ' + IntToStr(I);
    HeaderSection.MinWidth := length(HeaderSection.Text) * Font.Size;
    // Owner draw every other section
    if (I mod 2 = 0) then
      HeaderSection.Style := hsOwnerDraw
    else
      HeaderSection.Style := hsText;
  end;
end;

procedure TForm1.HeaderControl1DrawSection(HeaderControl: THeaderControl;
  Section: THeaderSection; const Rect: TRect; Pressed: Boolean);
begin
  with HeaderControl.Canvas do
  begin
    // highlight pressed sections
    if Pressed then
      Font.Color := clRed
    else
      Font.Color := clBlue;
    TextOut(Rect.Left + Font.Size, Rect.Top + 2, 'Owner Drawn text');
  end;
end;
```

See Also

Color (see page 5812)

73.1.8.5.3 TFont.FontAdapter Property

Designates an interface for providing the font information to ActiveX controls.

Pascal

```
property FontAdapter: IChangeNotifier;
```

C++

```
__property IChangeNotifier FontAdapter;
```

Description

Do not use the FontAdapter property. FontAdapter is used internally by ActiveX controls to communicate font information through an OLE interface.

See Also

TFontAdapter (see page 1583)

73.1.8.5.4 TFont.Handle Property

Provides access to the handle of a system font.

Pascal

```
property Handle: HFont;
```

C++

```
__property HFont Handle;
```

Description

The Handle property indicates the Windows GDI font object handle.

Use Handle as a parameter when calling Windows API functions that require a font handle. Set Handle to make the TFont (see page 14497) object represent the Windows font indicated by the font handle. Setting Handle releases the current Windows font resource.

Note: When possible, change a font by setting the other properties of TFont (see page 14497) rather than using the Handle property. Setting Handle does not reset the properties that describe the font. However, an OnChange (see page 14536) event does occur when Handle is set directly by an application.

C++ Examples:

```
/*
This example creates a rotated font and displays it on the
form by assigning the handle of the rotated to the Form's
Canvas Font via the Font's Handle property.  Rotating fonts
is a straightforward process, so long as the Windows Font
Mapper can supply a rotated font based on the font you
request. If you are using a TrueType font there is virtually
no reason for failure.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    LOGFONT lf; // Windows native font structure

    Canvas->Brush->Style = bsClear; // set the brush style to transparent
    ZeroMemory(&lf, sizeof(LOGFONT));

    lf.lfHeight = 20;
    lf.lfEscapement = 10 * 45; // degrees to rotate
    lf.lfOrientation = 10 * 45;
    lf.lfCharSet = DEFAULT_CHARSET;
    strcpy(lf.lfFaceName, "Tahoma");

    Canvas->Font->Handle = CreateFontIndirect(&lf);
    Canvas->TextOut(10, 100, "Rotated text"); // output the rotated font
}
```

Delphi Examples:

```
{
This example creates a rotated font and displays it on the
form by assigning the handle of the rotated to the Form's
Canvas Font via the Font's Handle property.  Rotating fonts
is a straightforward process, so long as the Windows Font
Mapper can supply a rotated font based on the font you
request. If you are using a TrueType font there is virtually
no reason for failure.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  lf: LOGFONT; // Windows native font structure
begin
  Canvas.Brush.Style := bsClear; // set the brush style to transparent
  FillChar(lf, SizeOf(lf), Byte(0));
  lf.lfHeight := 20;
  lf.lfEscapement := 10 * 45; // degrees to rotate
  lf.lfOrientation := 10 * 45;
  lf.lfCharSet := DEFAULT_CHARSET;
  StrCopy(lf.lfFaceName, 'Tahoma');

  Canvas.Font.Handle := CreateFontIndirect(lf);
  Canvas.TextOut(10, 100, 'Rotated text'); // output the rotated font
end;
```

See Also

OnChange (see page 14536)

73.1.8.5.5 TFont.Height Property

Specifies the height of the font.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use Height to specify the height of the font in pixels. If the value is negative, the internal leading that appears at the top of each line of text is not measured. If the value is positive, Height represents the height of the characters plus the internal leading.

To specify the size of the font in points, use the Size (see page 14513) property instead. Users usually specify font size in points, while applications may be concerned with the actual size of the font in pixels when displaying the font on the screen.

The value of Height can be obtained from the point size using this formula:

```
Font.Height = -Font.Size * Font.PixelsPerInch / 72
```

When the Height property has a positive value, the Size (see page 14513) property has a negative value. When the Size (see page 14513) property has a positive value, the Height property has a negative value.

See Also

Size (see page 14513)

73.1.8.5.6 TFont.Name Property

Identifies the typeface of the font.

Pascal

```
property Name: TFontName;
```

C++

```
__property TFontName Name;
```

Description

Use Name to specify the typeface of the font. If the font family described by Name includes multiple character sets, be sure to set the Charset (see page 14506) property as well.

Note: If the combination of font family (typeface) and attributes (such as bold or italic) specifies a font that is not available on the system, the system substitutes a different font.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

Charset (see page 14506)

Pitch (see page 14512)

Size (see page 14513)

73.1.8.5.7 TFont.Orientation Property

Specifies text orientation.

Pascal

```
property Orientation: Integer;
```

C++

```
__property int Orientation;
```

Description

Use Orientation to specify the orientation of text characters. It is the angle in tenths of degrees between the character's baseline and device x-axis. This is similar to the **lfOrientation** field of the Windows GDI **LOGFONT** structure that defines font attributes.

See Also

Color (see page 14507)

73.1.8.5.8 TFont.Pitch Property

Specifies whether the characters in the font all have the same width.

Pascal

```
property Pitch: TFontPitch;
```

C++

```
__property TFontPitch Pitch;
```

Description

Each typeface has an associated pitch. The characters in fixed-pitch fonts are all the same width. The characters in variable-pitch fonts differ in width. Fixed pitch fonts are generally used for representing source code, because they allow the indentation to line up properly. Variable pitch fonts are generally used to give text a more natural appearance

Here are the possible values for Pitch:

Value	Meaning
fpDefault	The font pitch is set to the default value, which depends on the font specified in the Name (see page 14510) property.
fpFixed	The font pitch is set to Fixed. All characters in the font have the same width.
fpVariable	The font pitch is set to Variable. The characters in the font have varying widths.

Setting the Pitch of a fixed-pitch font to fpVariable or a variable-pitch font to fpFixed will cause window system to find what it considers the best match to all of the font characteristics that have been specified. This might have no effect on the appearance of a font, or might cause the window system to substitute a different font with the appropriate pitch. For example, setting the pitch of Serif (a variable-pitch font) to fpFixed causes Courier to be displayed.

See Also

Name (see page 14510)

73.1.8.5.9 TFont.PixelsPerInch Property

Indicates the conversion factor between logical inches and the pixels of the device (printer or screen) using the font.

Pascal

```
property PixelsPerInch: Integer;
```

C++

```
__property int PixelsPerInch;
```

Description

PixelsPerInch is used to ensure that when a font is copied from the form's canvas to the printer, the font is the same size in points. The PixelsPerInch property affects printer fonts only and should not be modified.

To modify the size of a font, use the Size (see page 14513) or Height (see page 14510) properties.

See Also

Height (see page 14510)

Size (see page 14513)

73.1.8.5.10 TFont.Size Property

Specifies the height of the font in points.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

Use Size to specify the point size of the font. If the value is negative, the internal leading that appears at the top of each line of text is included. If the value is positive, Size represents the height of the characters but not the internal leading.

To determine the size of the font in pixels, use the Height (see page 14510) property instead. The value of Size can be obtained from the height in pixels using this formula:

```
Font.Size = -Font.Height * 72 / Font.PixelsPerInch
```

When the Size property has a positive value, the Height (see page 14510) property has a negative value. When the Height (see page 14510) property has a positive value, the Size property has a negative value.

See Also

Height (see page 14510)

73.1.8.5.11 TFont.Style Property

Determines whether the font is normal, italic, underlined, bold, and so on.

Pascal

```
property Style: TFontStyles;
```

C++

```
__property TFontStyles Style;
```

Description

Use Style to add special characteristics to characters that use the font. Style is a set containing zero or more values from the following:

Value	Meaning
fsBold	The font is boldfaced.
fsItalic	The font is italicized.
fsUnderline	The font is underlined.

fsStrikeOut	The font is displayed with a horizontal line through it.

C++ Examples:

```
/*
This example demonstrates how to set and clear font styles.
This example requires a button and a label that has text
with some styles set.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   Label1->Font->Style = TFontStyles() << fsBold << fsUnderline;
}
/*
This example demonstrates how to clear any Styles that were previously set.
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
   Label1->Font->Style = TFontStyles();
}
```

Delphi Examples:

```
{
This example demonstrates how to set and clear font styles.
This example requires a button and a label that has text
with some styles set.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   Label1.Font.Style := [fsBold, fsUnderline];
end;
{
This code demonstrates how to clear any Styles that were
previously set.
}
procedure TForm1.Button2Click(Sender: TObject);
begin
   Label1.Font.Style := [];
end;
```

See Also

Color (see page 14507)

73.1.9 TFontRecall Class

TFontRecall saves and restores the properties of a TFont (see page 14497) object.

Class Hierarchy

Pascal

```
TFontRecall = class(TRecall);
```

C++

```
class TFontRecall : public TRecall;
```

File

Graphics

Description

Use TFontRecall to store the current state of a TFont (see page 14497) object. You are then free to change the current state of the font object. To restore the font object to its original state, all that is necessary is to free the TFontRecall object and the TFont (see page 14497) object is automatically restored to the saved property values.

You can update the TFontRecall instance to reflect the current properties of the TFont (see page 14497) object by calling the Store method. You can prevent the TFontRecall destructor from updating the TFont (see page 14497) object by calling the Forget method.

See Also

TRecall (see page 2964)

TFont (see page 14497)

73.1.9.1 TFontRecall Members

The following tables list the members exposed by TFontRecall.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 14516)	Instantiates a TFontRecall (see page 14514) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2966)	Destroys the TRecall (see page 2964) instance, assigning the stored properties to the reference object.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Forget (see page 2967)	Discards the saved property values and prevents the TRecall (see page 2964) object from updating the reference object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Store (see page 2967)	Saves the current property settings of the reference object.

Public Properties

	Name	Description
R	Reference (see page 2968)	Indicates the reference object that is updated when you destroy the TRecall (see page 2964) instance.

73.1.9.2 TFontRecall.Create Constructor

Instantiates a TFontRecall (see page 14514) object.

Pascal

```
constructor Create(AFont: TFont);
```

C++

```
__fastcall TFontRecall(TFont AFont);
```

Description

Call Create to instantiate a font recall object at runtime. Create saves the current properties of the TFont (see page 14497) object specified by AFont, which becomes the value of the Reference property.

Subsequent calls to the Store method take snapshots of the properties of AFont and save them, replacing the values saved by this constructor. When you destroy the TFontRecall (see page 14514) instance, the most recently-saved properties are assigned to AFont, restoring it to a previous state.

See Also

Destroy (see page 2966)

Reference (see page 2968)

Store (see page 2967)

73.1.10 TGraphic Class

TGraphic is the abstract base class type for objects such as icons, bitmaps, and metafiles that can store and display visual images.

Class Hierarchy

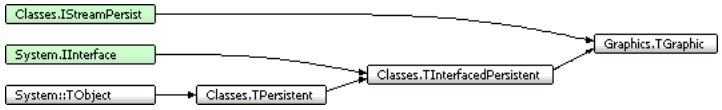

Pascal

```
TGraphic = class(TInterfacedPersistent, IStreamPersist);
```

C++

```
class TGraphic : public TInterfacedPersistent, public IStreamPersist;
```

File

Graphics

Description

TGraphic is an abstract class that cannot be instantiated. Descendant graphics objects override many of the methods of TGraphic to address the needs of their particular file format and graphical characteristics. TGraphic also introduces methods that work with TPicture (see page 14584) objects and the Clipboard. Properties of TGraphic provide information about the state and size of the graphic image.

When the type of graphic is known, store the graphic in its specific type object. Otherwise, use a TPicture (see page 14584) object that can hold any type of TGraphic.

See Also

TClipboard (see page 3328)

TBitmap (see page 14396)

TPicture (see page 14584)

TIcon (see page 14537)

Tmetafile

TJPEGImage (see page 15688)

TOleGraphic (see page 1587)

TImage (see page 12777)

Graphic Controls

73.1.10.1 TGraphic Members

The following tables list the members exposed by TGraphic.

Protected Methods

	Name	Description
	_AddRef (see page 2853)	Increments the reference count for the interfaced object's interface.
	_Release (see page 2854)	Decrements the reference count for the interfaced object's interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Changed (see page 14520)	This is Changed, a member of class TGraphic.
V	DefineProperties (see page 14520)	Provides an interface for a method that reads and writes otherwise unpublished data.
A	Draw (see page 14521)	This is Draw, a member of class TGraphic.
V	Equals (see page 14521)	This is Equals, a member of class TGraphic.
A	GetEmpty (see page 14521)	This is GetEmpty, a member of class TGraphic.
A	GetHeight (see page 14521)	This is GetHeight, a member of class TGraphic.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetPalette (see page 14522)	This is GetPalette, a member of class TGraphic.
V	GetTransparent (see page 14522)	This is GetTransparent, a member of class TGraphic.
A	GetWidth (see page 14522)	This is GetWidth, a member of class TGraphic.
	Progress (see page 14524)	This is Progress, a member of class TGraphic.
V	ReadData (see page 14524)	This is ReadData, a member of class TGraphic.
A	SetHeight (see page 14526)	This is SetHeight, a member of class TGraphic.
V	SetPalette (see page 14527)	This is SetPalette, a member of class TGraphic.
V	SetTransparent (see page 14527)	This is SetTransparent, a member of class TGraphic.

	SetWidth (see page 14527)	This is SetWidth, a member of class TGraphic.
	WriteData (see page 14527)	This is WriteData, a member of class TGraphic.

Public Events

	Name	Description
	OnChange (see page 14531)	Occurs whenever a graphical image changes.
	OnProgress (see page 14531)	Occurs when a graphical image is in the process of changing.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 2854)	Obtains a reference to the Owner's interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 14519)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LoadFromClipboardFormat (see page 14522)	Introduces an abstract method that loads the image from a variable in Clipboard format.
	LoadFromFile (see page 14523)	Loads a graphic image stored in a file.
	LoadFromStream (see page 14524)	Introduces an abstract method that loads the image from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 2854)	Returns a reference to a specified interface if the object supports that interface.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToClipboardFormat (see page 14525)	Introduces an abstract method that assigns the graphic to a variable in clipboard format.
	SaveToFile (see page 14525)	Saves a graphics image to a file.

	SaveToStream (see page 14526)	Introduces an abstract method that saves the graphic image to a stream.
	SetSize (see page 14527)	Specifies the size of the graphic.

Public Properties

	Name	Description
	Empty (see page 14528)	Indicates whether the graphics object contains a graphic.
	Height (see page 14528)	Specifies the vertical size of the graphic in pixels.
	Modified (see page 14528)	Indicates whether the graphics object has been changed or edited.
	Palette (see page 14529)	Indicates the color palette of the graphical image.
	PaletteModified (see page 14529)	Indicates whether the palette has changed.
	Transparent (see page 14529)	Indicates whether the image covers its rectangular area.
	Width (see page 14530)	Determines the maximum width of the graphics object in pixels.

73.1.10.2 TGraphic.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TGraphic();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

73.1.10.3 TGraphic Methods

The methods of the TGraphic class are listed here.

Protected Methods

	Name	Description
	Changed (see page 14520)	This is Changed, a member of class TGraphic.
	DefineProperties (see page 14520)	Provides an interface for a method that reads and writes otherwise unpublished data.
	Draw (see page 14521)	This is Draw, a member of class TGraphic.
	Equals (see page 14521)	This is Equals, a member of class TGraphic.
	GetEmpty (see page 14521)	This is GetEmpty, a member of class TGraphic.
	GetHeight (see page 14521)	This is GetHeight, a member of class TGraphic.
	GetPalette (see page 14522)	This is GetPalette, a member of class TGraphic.
	GetTransparent (see page 14522)	This is GetTransparent, a member of class TGraphic.
	GetWidth (see page 14522)	This is GetWidth, a member of class TGraphic.
	Progress (see page 14524)	This is Progress, a member of class TGraphic.
	ReadData (see page 14524)	This is ReadData, a member of class TGraphic.

	SetHeight (see page 14526)	This is SetHeight, a member of class TGraphic.
	SetPalette (see page 14527)	This is SetPalette, a member of class TGraphic.
	SetTransparent (see page 14527)	This is SetTransparent, a member of class TGraphic.
	SetWidth (see page 14527)	This is SetWidth, a member of class TGraphic.
	WriteData (see page 14527)	This is WriteData, a member of class TGraphic.

Public Methods

	Name	Description
	LoadFromClipboardFormat (see page 14522)	Introduces an abstract method that loads the image from a variable in Clipboard format.
	LoadFromFile (see page 14523)	Loads a graphic image stored in a file.
	LoadFromStream (see page 14524)	Introduces an abstract method that loads the image from a stream.
	SaveToClipboardFormat (see page 14525)	Introduces an abstract method that assigns the graphic to a variable in clipboard format.
	SaveToFile (see page 14525)	Saves a graphics image to a file.
	SaveToStream (see page 14526)	Introduces an abstract method that saves the graphic image to a stream.
	SetSize (see page 14527)	Specifies the size of the graphic.

73.1.10.3.1 TGraphic.Changed Method

Pascal

```
procedure Changed(Sender: TObject); virtual;
```

C++

```
virtual __fastcall Changed(TObject * Sender);
```

Description

This is Changed, a member of class TGraphic.

73.1.10.3.2 TGraphic.DefineProperties Method

Provides an interface for a method that reads and writes otherwise unpublished data.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

Descendants of TPersistent override DefineProperties to designate a method for storing the object's unpublished data to a stream such as a form file. By default, writing an object to a stream writes the values of all its published properties, and reading the object in reads those values and assigns them to the properties. Objects can also specify methods that read and write data other than published properties by overriding the DefineProperties method.

When overriding DefineProperties, consider including some or all of the following:

A call to the inherited method

Calls to the filer object's DefineProperty method

Calls to the filer object's DefineBinaryProperty method

DefineProperties is virtual, so descendant classes can override it as necessary but are not required to do so.

See Also

DefineProperties (see page 2771)

DefineBinaryProperty (see page 2837)

DefineProperty (see page 2838)

73.1.10.3.3 TGraphic.Draw Method

Pascal

```
procedure Draw(ACanvas: TCanvas; const Rect: TRect); virtual; abstract;
```

C++

```
virtual __fastcall Draw(TCanvas ACanvas, const TRect Rect) = 0;
```

Description

This is Draw, a member of class TGraphic.

73.1.10.3.4 TGraphic.Equals Method

Pascal

```
function Equals(Graphic: TGraphic): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean Equals(TGraphic Graphic);
```

Description

This is Equals, a member of class TGraphic.

73.1.10.3.5 TGraphic.GetEmpty Method

Pascal

```
function GetEmpty: Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean GetEmpty() = 0;
```

Description

This is GetEmpty, a member of class TGraphic.

73.1.10.3.6 TGraphic.GetHeight Method

Pascal

```
function GetHeight: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetHeight() = 0;
```

Description

This is GetHeight, a member of class TGraphic.

73.1.10.3.7 TGraphic.GetPalette Method

Pascal

```
function GetPalette: HPALETTE; virtual;
```

C++

```
virtual __fastcall HPALETTE GetPalette();
```

Description

This is GetPalette, a member of class TGraphic.

73.1.10.3.8 TGraphic.GetTransparent Method

Pascal

```
function GetTransparent: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetTransparent();
```

Description

This is GetTransparent, a member of class TGraphic.

73.1.10.3.9 TGraphic.GetWidth Method

Pascal

```
function GetWidth: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetWidth() = 0;
```

Description

This is GetWidth, a member of class TGraphic.

73.1.10.3.10 TGraphic.LoadFromClipboardFormat Method

Introduces an abstract method that loads the image from a variable in Clipboard format.

Pascal

```
procedure LoadFromClipboardFormat(AFormat: Word; AData: THandle; APalette: HPALETTE); virtual;
abstract;
```

C++

```
virtual __fastcall LoadFromClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette) = 0;
```

Description

Each descendant graphic object defines a LoadFromClipboardFormat method that replaces the current graphic image with the graphic indicated by AData, which it loads from the Clipboard.

The format for the new graphic object must be registered with the Clipboard in the RegisterClipboardFormat method, or an exception is raised.

See Also

TClipboard (see page 3328)

RegisterClipboardFormat (see page 14592)

73.1.10.3.11 TGraphic.LoadFromFile Method

Loads a graphic image stored in a file.

Pascal

```
procedure LoadFromFile(const Filename: string); virtual;
```

C++

```
virtual __fastcall LoadFromFile(const AnsiString Filename);
```

Description

LoadFromFile reads the file specified in FileName and loads the data into the graphics object.

C++ Examples:

```
/*
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
Note: For the C++ Builder, capitalization matters for the file
name and path.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *Bitmap1 = new Graphics::TBitmap();
  Graphics::TBitmap *Bitmap2 = new Graphics::TBitmap();
  try
  {
    Bitmap1->LoadFromFile("..\\FACTORY.BMP");
    Bitmap2->Assign(Bitmap1);     // Copy Bitmap1 into Bitmap2
    Bitmap2->Dormant();           // Free up GDI resources
    Bitmap2->FreeImage();         // Free up Memory.
    Canvas->Draw(20,20,Bitmap2);  // Note that previous calls don't lose the image
    Bitmap2->Monochrome = true;
    Canvas->Draw(80,80,Bitmap2);
    Bitmap2->ReleaseHandle();       // This will actually lose the bitmap;
  }
  catch (...)
  {
    MessageBeep(0);
  }
  delete Bitmap1;
  delete Bitmap2;
}
```

Delphi Examples:

```
{
This example uses a button on a form and creates two bitmaps
dynamically.  One bitmap is monchrome, which means all
non-white colors become black.  The bitmap file path is
relative to the Debug directory.
}
procedure TForm1.Button1Click(Sender: TObject);
var
 BitMap1,BitMap2 : TBitMap;
 MyFormat : Word;
```

```
begin
   BitMap2 := TBitMap.Create;
   BitMap1 := TBitMap.Create;
try
   BitMap1.LoadFromFile('factory.bmp');
   BitMap2.Assign(BitMap1);     // Copy BitMap1 into BitMap2
   BitMap2.Dormant;             // Free up GDI resources
   BitMap2.FreeImage;           // Free up Memory.
   Canvas.Draw(20,20,BitMap2);  // Note that previous calls don't lose the image
   BitMap2.Monochrome := true;
   Canvas.Draw(80,80,BitMap2);
   BitMap2.ReleaseHandle;       // This will actually lose the bitmap;
 finally
   BitMap1.Free;
   BitMap2.Free;
 end;
end;
```

See Also

LoadFromStream (see page 14524)

73.1.10.3.12 TGraphic.LoadFromStream Method

Introduces an abstract method that loads the image from a stream.

Pascal

```
procedure LoadFromStream(Stream: TStream); virtual; abstract;
```

C++

```
virtual __fastcall LoadFromStream(TStream Stream) = 0;
```

Description

Each descendant graphic object defines a LoadFromStream method that loads a graphics object from Stream.

See Also

LoadFromFile (see page 14523)

TStream (see page 2973)

73.1.10.3.13 TGraphic.Progress Method

Pascal

```
procedure Progress(Sender: TObject; Stage: TProgressStage; PercentDone: Byte; RedrawNow:
Boolean; const R: TRect; const Msg: string); dynamic;
```

C++

```
__fastcall Progress(TObject * Sender, TProgressStage Stage, Byte PercentDone, Boolean
RedrawNow, const TRect R, const AnsiString Msg);
```

Description

This is Progress, a member of class TGraphic.

73.1.10.3.14 TGraphic.ReadData Method

Pascal

```
procedure ReadData(Stream: TStream); virtual;
```

C++

```
virtual __fastcall ReadData(TStream Stream);
```

Description

This is ReadData, a member of class TGraphic.

73.1.10.3.15 TGraphic.SaveToClipboardFormat Method

Introduces an abstract method that assigns the graphic to a variable in clipboard format.

Pascal

```
procedure SaveToClipboardFormat(var AFormat: Word; var AData: THandle; var APalette:
HPALETTE); virtual; abstract;
```

C++

```
virtual __fastcall SaveToClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette) = 0;
```

Description

Each descendant graphic object defines a SaveToClipboardFormat method that saves that particular graphic image type to a Clipboard format. An application must have registered the format with the TGraphic (see page 14516) object using the RegisterClipboardFormat method.

See Also

TClipboard (see page 3328)

RegisterClipboardFormat (see page 14592)

73.1.10.3.16 TGraphic.SaveToFile Method

Saves a graphics image to a file.

Pascal

```
procedure SaveToFile(const Filename: string); virtual;
```

C++

```
virtual __fastcall SaveToFile(const AnsiString Filename);
```

Description

SaveToFile writes the graphic to a file, specified by Filename.

C++ Examples:

```
/*
This example converts a bitmap image to a jpeg file by using
the Assign method.  This example requires a button and two
images.  One image must have it's Picture property loaded
with a bitmap at design time.
*/

#include <Jpeg.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  //Requires "jpeg.hpp" to be included in the source file
  TJPEGImage *jp = new TJPEGImage();
  try
  {
    jp->Assign(Image1->Picture->Bitmap);
```

```
      jp->SaveToFile("..\\factory.jpg");
      Image2->Stretch = True;
      Image2->Picture->LoadFromFile("..\\factory.jpg");
    }
    __finally
    {
      delete jp;
    }
  }
```

Delphi Examples:

```
{
This example converts a bitmap image to a jpeg file by using
the Assign method.  This example requires a button and two
images.  One image must have it's Picture property loaded
with a bitmap at design time.
}
uses Jpeg;

procedure TForm1.Button1Click(Sender: TObject);
var
  jp: TJPEGImage;  //Requires the "jpeg" unit added to "uses" clause.
begin
  jp := TJPEGImage.Create;
  try
    with jp do
    begin
      Assign(Image1.Picture.Bitmap);
      SaveToFile('factory.jpg');
      Image2.Stretch := True;
      Image2.Picture.LoadFromFile('factory.jpg');
    end;
  finally
    jp.Free;
  end;
end;
```

73.1.10.3.17 TGraphic.SaveToStream Method

Introduces an abstract method that saves the graphic image to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream); virtual; abstract;
```

C++

```
virtual __fastcall SaveToStream(TStream Stream) = 0;
```

Description

Each descendant graphic object defines a SaveToStream method that saves the object to a Stream.

73.1.10.3.18 TGraphic.SetHeight Method

Pascal

```
procedure SetHeight(Value: Integer); virtual; abstract;
```

C++

```
virtual __fastcall SetHeight(int Value) = 0;
```

Description

This is SetHeight, a member of class TGraphic.

73.1.10.3.19 TGraphic.SetPalette Method

Pascal

```
procedure SetPalette(Value: HPALETTE); virtual;
```

C++

```
virtual __fastcall SetPalette(HPALETTE Value);
```

Description

This is SetPalette, a member of class TGraphic.

73.1.10.3.20 TGraphic.SetSize Method

Specifies the size of the graphic.

Pascal

```
procedure SetSize(AWidth: Integer; AHeight: Integer); virtual;
```

C++

```
virtual __fastcall SetSize(int AWidth, int AHeight);
```

Description

Use the SetSize method to set both the height and width of the graphic. This results in better performance than setting the height and width separately.

73.1.10.3.21 TGraphic.SetTransparent Method

Pascal

```
procedure SetTransparent(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetTransparent(Boolean Value);
```

Description

This is SetTransparent, a member of class TGraphic.

73.1.10.3.22 TGraphic.SetWidth Method

Pascal

```
procedure SetWidth(Value: Integer); virtual; abstract;
```

C++

```
virtual __fastcall SetWidth(int Value) = 0;
```

Description

This is SetWidth, a member of class TGraphic.

73.1.10.3.23 TGraphic.WriteData Method

Pascal

```
procedure WriteData(Stream: TStream); virtual;
```

C++

```
virtual __fastcall WriteData(TStream Stream);
```

Description

This is WriteData, a member of class TGraphic.

73.1.10.4 TGraphic Properties

The properties of the TGraphic class are listed here.

Public Properties

	Name	Description
R	Empty (see page 14528)	Indicates whether the graphics object contains a graphic.
	Height (see page 14528)	Specifies the vertical size of the graphic in pixels.
	Modified (see page 14528)	Indicates whether the graphics object has been changed or edited.
	Palette (see page 14529)	Indicates the color palette of the graphical image.
	PaletteModified (see page 14529)	Indicates whether the palette has changed.
	Transparent (see page 14529)	Indicates whether the image covers its rectangular area.
	Width (see page 14530)	Determines the maximum width of the graphics object in pixels.

73.1.10.4.1 TGraphic.Empty Property

Indicates whether the graphics object contains a graphic.

Pascal

```
property Empty: Boolean;
```

C++

```
__property Boolean Empty;
```

Description

Use Empty to determine whether the graphic is bound to an image. Each descendant graphic object defines its own Get method to access the Empty property.

73.1.10.4.2 TGraphic.Height Property

Specifies the vertical size of the graphic in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use Height to determine the height of the graphic image. Each descendant graphic object defines its own Get and Set methods to access the Height property.

73.1.10.4.3 TGraphic.Modified Property

Indicates whether the graphics object has been changed or edited.

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

If Modified is true, the graphic object has changed. If Modified is false, the graphics object is in the same state as when the object was loaded.

The Modified property indicates only if bitmap objects have been modified. Modified is not true if the graphics object contains an icon or metafile graphic, even if they have been modified.

If the graphics object was modified, save the changes to a file with the SaveToFile (see page 14525) method. The next time the application runs, it can load the graphic from the file with the LoadFromFile (see page 14523) method.

73.1.10.4.4 TGraphic.Palette Property

Indicates the color palette of the graphical image.

Pascal

```
property Palette: HPALETTE;
```

C++

```
__property HPALETTE Palette;
```

Description

Use Palette to get the color palette of a graphical image. If the graphic does not need or does not use a palette, the Palette property is zero.

73.1.10.4.5 TGraphic.PaletteModified Property

Indicates whether the palette has changed.

Pascal

```
property PaletteModified: Boolean;
```

C++

```
__property Boolean PaletteModified;
```

Description

Use PaletteModified to determine if the palette used for a graphical image has changed. PaletteModified is used in the OnChange (see page 14531) event. PaletteModified remains true until whoever is responsible for realizing this new palette (for example, TImage) sets it to false.

73.1.10.4.6 TGraphic.Transparent Property

Indicates whether the image covers its rectangular area.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Use Transparent to specify that the graphic be drawn transparently. Some descendants of TGraphic (see page 14516) such as TIcon (see page 14537) and TMetafile (see page 14549) are always transparent, so setting the property for those objects does not change their behavior. However, the TBitmap (see page 14396) graphic's drawing is affected by this property. The TImage component sets this property to be the same as its Transparent property to achieve transparent painting.

When the Transparent property is set to True, you can either specify a color as the transparent color or you can use the default color, which is the pixel in the lower left. The specified color is not displayed in the graphic, which lets the background show through. This can be useful in layering and for non-rectangular graphics.

C++ Examples:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *bitmap = new Graphics::TBitmap();
  try
  {
    bitmap->LoadFromFile("..\\FACTORY.BMP ");
    bitmap->Transparent = true;
    bitmap->TransparentColor = bitmap->Canvas->Pixels[50][50];
    Form1->Canvas->Draw( 0, 0, bitmap);
    bitmap->TransparentMode = tmAuto;
    Form1->Canvas->Draw(50, 50, bitmap);
  }
  catch (...)
  {
    ShowMessage("Could not load or display bitmap");
  }
  delete bitmap;
```

Delphi Examples:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitMap;
begin
  Bitmap := TBitmap.Create;
  try
    with Bitmap do begin
      LoadFromFile('factory.bmp');
      Transparent := True;
      TransParentColor := BitMap.canvas.pixels[50,50];
      Form1.Canvas.Draw(0,0,BitMap);
      TransparentMode := tmAuto;
      Form1.Canvas.Draw(50,50,BitMap);
    end;
  finally
    Bitmap.Free;
  end;
end;
```

73.1.10.4.7 TGraphic.Width Property

Determines the maximum width of the graphics object in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Each descendant graphic object defines its own Get and Set methods to access the Width property.

73.1.10.5 TGraphic Events

The events of the TGraphic class are listed here.

Public Events

	Name	Description
	OnChange (see page 14531)	Occurs whenever a graphical image changes.
	OnProgress (see page 14531)	Occurs when a graphical image is in the process of changing.

73.1.10.5.1 TGraphic.OnChange Event

Occurs whenever a graphical image changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Use OnChange to write a handler to perform an action then the graphical image changes.

73.1.10.5.2 TGraphic.OnProgress Event

Occurs when a graphical image is in the process of changing.

Pascal

```
property OnProgress: TProgressEvent;
```

C++

```
__property TProgressEvent OnProgress;
```

Description

For certain descendants of TGraphic (see page 14516), OnProgress occurs during slow processes such as loading, storing, or transforming image data. OnProgress allows applications to provide feedback to the user about the progress of the slow process.

Component writers can generate OnProgress events for new descendants of TGraphic (see page 14516) by calling the protected Progress (see page 14524) method. These events are propagated to the TPicture (see page 14584) and TImage objects.

See Also

OnProgress (see page 12831)

OnProgress (see page 14603)

73.1.11 TGraphicsObject Class

TGraphicsObject is the abstract base class for objects which encapsulate a system graphics object: TBrush (see page 14430), TFont (see page 14497), and TPen (see page 14571).

Class Hierarchy

Pascal

```
TGraphicsObject = class(TPersistent);
```

C++

```
class TGraphicsObject : public TPersistent;
```

File

Graphics

Description

Create descendants of TGraphicsObject when drawing an image to a canvas (TCanvas (see page 14441)). Each TGraphicsObject object describes a system graphics object that is selected into the device context of the canvas.

TGraphicsObject introduces a mechanism for responding to changes in the properties of the graphics object.

See Also

TBrush (see page 14430)

TCanvas (see page 14441)

TFont (see page 14497)

TPen (see page 14571)

73.1.11.1 TGraphicsObject Members

The following tables list the members exposed by TGraphicsObject.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 14534)	Generates an OnChange (see page 14536) event when the properties of the graphics object change.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.
	Lock (see page 14534)	Blocks other execution threads from using the associated canvas until the Unlock (see page 14535) method is called.
	Unlock (see page 14535)	Releases the lock on execution threads set by the Lock (see page 14534) method.

Public Events

	Name	Description
	OnChange (see page 14536)	Occurs immediately after the graphics object changes.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	HandleAllocated (see page 14534)	Indicates whether the TGraphicsObject (see page 14532) object has acquired a handle to the underlying GDI object.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	OwnerCriticalSection (see page 14536)	Points to information about the critical section that the canvas which draws with this graphics object uses to lock out other threads.

73.1.11.2 TGraphicsObject Methods

The methods of the TGraphicsObject class are listed here.

Protected Methods

	Name	Description
	Changed (see page 14534)	Generates an OnChange (see page 14536) event when the properties of the graphics object change.
	Lock (see page 14534)	Blocks other execution threads from using the associated canvas until the Unlock (see page 14535) method is called.

	Unlock (see page 14535)	Releases the lock on execution threads set by the Lock (see page 14534) method.

Public Methods

	Name	Description
	HandleAllocated (see page 14534)	Indicates whether the TGraphicsObject (see page 14532) object has acquired a handle to the underlying GDI object.

73.1.11.2.1 TGraphicsObject.Changed Method

Generates an OnChange (see page 14536) event when the properties of the graphics object change.

Pascal

```
procedure Changed; dynamic;
```

C++

```
__fastcall Changed();
```

Description

Descendants of TGraphicsObject (see page 14532) call Changed after making changes to the properties encapsulated by the TGraphicsObject (see page 14532) object.

See Also

OnChange (see page 14536)

73.1.11.2.2 TGraphicsObject.HandleAllocated Method

Indicates whether the TGraphicsObject (see page 14532) object has acquired a handle to the underlying GDI object.

Pascal

```
function HandleAllocated: Boolean;
```

C++

```
__fastcall Boolean HandleAllocated();
```

Description

Use HandleAllocated to determine whether the Handle property is set for the graphics object. HandleAllocated indicates whether the underlying GDI object has been acquired. Use HandleAllocated rather than reading the Handle property. When an application reads the Handle property, TGraphicsObject (see page 14532) descendants automatically create the underlying GDI object and return its handle if the property had not been previously set.

See Also

Handle (see page 14578)

Handle (see page 14509)

Handle (see page 14438)

73.1.11.2.3 TGraphicsObject.Lock Method

Blocks other execution threads from using the associated canvas until the Unlock (see page 14535) method is called.

Pascal

```
procedure Lock;
```

C++

```
__fastcall Lock();
```

Description

Call Lock in a multithreaded application before using the graphics object in a section of code that would not otherwise be thread-safe. Lock prevents other cooperating threads in the application from using the associated canvas until the Unlock (see page 14535) method is called. Calls to Lock can be nested so that the canvas is not unlocked until the last lock is released.

Because Lock prevents other threads from executing, it can adversely affect performance. Do not call Lock unless there is a danger that another thread might interfere with the drawing.

Note: Lock will do nothing unless the drawing surface sets the OwnerCriticalSection (see page 14536) property. TCanvas (see page 14441) objects automatically set the OwnerCriticalSection (see page 14536) property when they use graphics objects.

See Also

OwnerCriticalSection (see page 14536)

TCanvas (see page 14441)

TThread (see page 3108)

Unlock (see page 14535)

73.1.11.2.4 TGraphicsObject.Unlock Method

Releases the lock on execution threads set by the Lock (see page 14534) method.

Pascal

```
procedure Unlock;
```

C++

```
__fastcall Unlock();
```

Description

Call Unlock at the end of a thread-sensitive section of code that was started with a call to the Lock (see page 14534) method. When every call to Lock (see page 14534) is balanced by a call to Unlock, other threads are allowed to lock the associated canvas.

See Also

Lock (see page 14534)

TThread (see page 3108)

TCanvas (see page 14441)

73.1.11.3 TGraphicsObject Properties

The properties of the TGraphicsObject class are listed here.

Public Properties

	Name	Description
	OwnerCriticalSection (see page 14536)	Points to information about the critical section that the canvas which draws with this graphics object uses to lock out other threads.

73.1.11.3.1 TGraphicsObject.OwnerCriticalSection Property

Points to information about the critical section that the canvas which draws with this graphics object uses to lock out other threads.

Pascal

```
property OwnerCriticalSection: PRTLCriticalSection;
```

C++

```
__property PRTLCriticalSection OwnerCriticalSection;
```

Description

Use OwnerCriticalSection to obtain information about the thread locking state of the canvas that draws with this graphics object. The canvas sets OwnerCriticalSection when it creates the graphics object. Do not change this property while the canvas is using the graphics object.

If the graphics object is created on behalf of a drawing surface that is not a canvas, set OwnerCriticalSection to an initialized TRTLCriticalSection that is maintained by the drawing surface.

To use OwnerCriticalSection to block other execution threads, use the Lock (see page 14534) method.

See Also

Lock (see page 14534)

Unlock (see page 14535)

Lock (see page 14463)

73.1.11.4 TGraphicsObject Events

The events of the TGraphicsObject class are listed here.

Public Events

	Name	Description
	OnChange (see page 14536)	Occurs immediately after the graphics object changes.

73.1.11.4.1 TGraphicsObject.OnChange Event

Occurs immediately after the graphics object changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action when the graphics object changes. The properties of the graphics object reflect the new settings when OnChange occurs.

See Also

Changed (see page 14534)

73.1.12 TIcon Class

TIcon is an encapsulation of a Windows icon.

Class Hierarchy

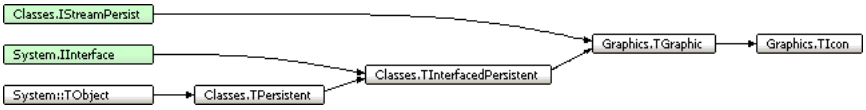

Pascal

```
TIcon = class(TGraphic);
```

C++

```
class TIcon : public TGraphic;
```

File

Graphics

Description

Use TIcon to represent one of the icons in an image list. Assign (see page 14541) an icon to objects that have an Icon property, such as TForm or TPicture (see page 14584).

Icon objects represent the value loaded from a Windows icon file (.ICO file). Draw (see page 14542) an icon on a canvas, using the Draw (see page 14542) method of the TCanvas (see page 14441) object. Icons do not stretch, so do not use StretchDraw (TCanvas (see page 14441)) with an icon.

Note: TIcon can be used to display the value of a multi-resolution (>16 color) icon. However, it can not support the creation of multi-resolution icons, nor the resizing of the image after it is loaded from a file or stream.

See Also

TCanvas (see page 14441)

Icon

TImageList (see page 6078)

TPicture (see page 14584)

73.1.12.1 TIcon Members

The following tables list the members exposed by TIcon.

Protected Methods

	Name	Description
	_AddRef (see page 2853)	Increments the reference count for the interfaced object's interface.
	_Release (see page 2854)	Decrements the reference count for the interfaced object's interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Changed (see page 14520)	This is Changed, a member of class TGraphic.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 14520)	Provides an interface for a method that reads and writes otherwise unpublished data.

	Draw (see page 14542)	This is Draw, a member of class TIcon.
	Equals (see page 14521)	This is Equals, a member of class TGraphic.
	GetEmpty (see page 14542)	This is GetEmpty, a member of class TIcon.
	GetHeight (see page 14542)	This is GetHeight, a member of class TIcon.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 14522)	This is GetPalette, a member of class TGraphic.
	GetTransparent (see page 14522)	This is GetTransparent, a member of class TGraphic.
	GetWidth (see page 14542)	This is GetWidth, a member of class TIcon.
	Progress (see page 14524)	This is Progress, a member of class TGraphic.
	ReadData (see page 14524)	This is ReadData, a member of class TGraphic.
	SetHeight (see page 14545)	This is SetHeight, a member of class TIcon.
	SetPalette (see page 14527)	This is SetPalette, a member of class TGraphic.
	SetTransparent (see page 14546)	This is SetTransparent, a member of class TIcon.
	SetWidth (see page 14546)	This is SetWidth, a member of class TIcon.
	WriteData (see page 14527)	This is WriteData, a member of class TGraphic.

Public Events

	Name	Description
	OnChange (see page 14531)	Occurs whenever a graphical image changes.
	OnProgress (see page 14531)	Occurs when a graphical image is in the process of changing.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 2854)	Obtains a reference to the Owner's interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 14541)	Copies an icon image from another TIcon (see page 14537) object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 14539)	Creates an instance of TIcon (see page 14537).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 14541)	Destroys an instance of TIcon (see page 14537).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	HandleAllocated (see page 14543)	Indicates whether the TIcon (see page 14537) object has acquired a handle to the underlying GDI object.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	LoadFromClipboardFormat (see page 14543)	Prevents applications from loading icons from the clipboard.
	LoadFromFile (see page 14523)	Loads a graphic image stored in a file.
	LoadFromResourceID (see page 14543)	This is LoadFromResourceID, a member of class TIcon.
	LoadFromResourceName (see page 14544)	Loads an icon resource into the icon object.
	LoadFromStream (see page 14544)	Loads the icon from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 2854)	Returns a reference to a specified interface if the object supports that interface.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReleaseHandle (see page 14544)	Releases the Windows GDI object represented by the icon.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToClipboardFormat (see page 14545)	Prevents applications from saving icons to the clipboard.
	SaveToFile (see page 14525)	Saves a graphics image to a file.
	SaveToStream (see page 14545)	Saves the icon to a stream.
	SetSize (see page 14546)	Specifies the size of the icon.

Public Properties

	Name	Description
	Empty (see page 14528)	Indicates whether the graphics object contains a graphic.
	Handle (see page 14546)	Provides access to the Windows GDI icon handle.
	Height (see page 14528)	Specifies the vertical size of the graphic in pixels.
	Modified (see page 14528)	Indicates whether the graphics object has been changed or edited.
	Palette (see page 14529)	Indicates the color palette of the graphical image.
	PaletteModified (see page 14529)	Indicates whether the palette has changed.
	Transparent (see page 14529)	Indicates whether the image covers its rectangular area.
	Width (see page 14530)	Determines the maximum width of the graphics object in pixels.

73.1.12.2 TIcon.Create Constructor

Creates an instance of TIcon (see page 14537).

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TIcon();
```

Description

Call Create to create an empty icon object. Once an image has been read into the icon from a file or stream, or by assigning another icon object, the icon can be drawn on a canvas or added to an image list. Do not create an icon object for setting the icon property of a form or picture. These objects create a TIcon (see page 14537) object in their own constructors. Instead, use the LoadFromFile (see page 14523), LoadFromStream (see page 14544), or LoadFromClipboardFormat (see page 14543) method on the TIcon (see page 14537) object that you can obtain by reading the Icon property.

C++ Examples:

```
/*
This example converts a specified icon to a bitmap. To run
this example, add an image, a button, and an Open dialog to
a form. Name the button ConvertIcon2Bitmap, and add the
following code as its OnClick event handler.
*/
void __fastcall TForm1::ConvertIcon2BitmapClick(TObject *Sender)
{
```

```
    OpenDialog1->DefaultExt = ".ICO";
    OpenDialog1->Filter = "icons (*.ico)|*.ICO";
    OpenDialog1->Options << ofOverwritePrompt << ofFileMustExist << ofHideReadOnly;
    if (OpenDialog1->Execute())
    {
      TIcon *pi = new TIcon();
      try
      {
        AnsiString as;
        pi->LoadFromFile(OpenDialog1->FileName);
        as = ChangeFileExt(OpenDialog1->FileName,".BMP");
        Image1->Width = pi->Width;
        Image1->Height = pi->Height;
        Image1->Canvas->Draw(0,0,pi);
        Image1->Picture->SaveToFile(as);
        ShowMessage(OpenDialog1->FileName + " Saved to " + as);
      }
      __finally
      {
        delete pi;
      }
    }
  }
```

Delphi Examples:

```
{
This example converts a specified icon to a bitmap. To run
this example, add an image, a button, and an Open dialog to
a form. Name the button ConvertIcon2Bitmap, and add the
following code as its OnClick event handler.
}
procedure TForm1.ConvertIcon2BitmapClick(Sender: TObject);
var
  s : string;
  Icon: TIcon;
begin
  OpenDialog1.DefaultExt := '.ICO';
  OpenDialog1.Filter := 'icons (*.ico)|*.ICO';
  OpenDialog1.Options := [ofOverwritePrompt, ofFileMustExist, ofHideReadOnly ];
  if OpenDialog1.Execute then
  begin
    Icon := TIcon.Create;
    try
      Icon.Loadfromfile(OpenDialog1.FileName);
      s:= ChangeFileExt(OpenDialog1.FileName,'.BMP');
      Image1.Width := Icon.Width;
      Image1.Height := Icon.Height;
      Image1.Canvas.Draw(0,0,Icon);
      Image1.Picture.SaveToFile(s);
      ShowMessage(OpenDialog1.FileName + ' Saved to ' + s);
    finally
      Icon.Free;
    end;
  end;
end;
```

See Also

Empty

LoadFromClipboardFormat (see page 14522)

LoadFromFile (see page 14523)

LoadFromStream (see page 14544)

73.1.12.3 TIcon.Destroy Destructor

Destroys an instance of TIcon (see page 14537).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIcon();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TIcon (see page 14537) reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

ReleaseHandle (see page 14544)

73.1.12.4 TIcon Methods

The methods of the TIcon class are listed here.

Protected Methods

	Name	Description
	Draw (see page 14542)	This is Draw, a member of class TIcon.
	GetEmpty (see page 14542)	This is GetEmpty, a member of class TIcon.
	GetHeight (see page 14542)	This is GetHeight, a member of class TIcon.
	GetWidth (see page 14542)	This is GetWidth, a member of class TIcon.
	SetHeight (see page 14545)	This is SetHeight, a member of class TIcon.
	SetTransparent (see page 14546)	This is SetTransparent, a member of class TIcon.
	SetWidth (see page 14546)	This is SetWidth, a member of class TIcon.

Public Methods

	Name	Description
	Assign (see page 14541)	Copies an icon image from another TIcon (see page 14537) object.
	HandleAllocated (see page 14543)	Indicates whether the TIcon (see page 14537) object has acquired a handle to the underlying GDI object.
	LoadFromClipboardFormat (see page 14543)	Prevents applications from loading icons from the clipboard.
	LoadFromResourceID (see page 14543)	This is LoadFromResourceID, a member of class TIcon.
	LoadFromResourceName (see page 14544)	Loads an icon resource into the icon object.
	LoadFromStream (see page 14544)	Loads the icon from a stream.
	ReleaseHandle (see page 14544)	Releases the Windows GDI object represented by the icon.
	SaveToClipboardFormat (see page 14545)	Prevents applications from saving icons to the clipboard.
	SaveToStream (see page 14545)	Saves the icon to a stream.
	SetSize (see page 14546)	Specifies the size of the icon.

73.1.12.4.1 TIcon.Assign Method

Copies an icon image from another TIcon (see page 14537) object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy another icon object. Assign copies the icon image from the Source parameter if it is another TIcon (see page 14537) object. Otherwise, Assign calls the inherited method, which copies the icon image from any source object that specifies how to copy to a TIcon (see page 14537) in its AssignTo method.

See Also

AssignTo (see page 2931)

73.1.12.4.2 TIcon.Draw Method

Pascal

```
procedure Draw(ACanvas: TCanvas; const Rect: TRect); override;
```

C++

```
virtual __fastcall Draw(TCanvas ACanvas, const TRect Rect);
```

Description

This is Draw, a member of class TIcon.

73.1.12.4.3 TIcon.GetEmpty Method

Pascal

```
function GetEmpty: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetEmpty();
```

Description

This is GetEmpty, a member of class TIcon.

73.1.12.4.4 TIcon.GetHeight Method

Pascal

```
function GetHeight: Integer; override;
```

C++

```
virtual __fastcall int GetHeight();
```

Description

This is GetHeight, a member of class TIcon.

73.1.12.4.5 TIcon.GetWidth Method

Pascal

```
function GetWidth: Integer; override;
```

C++

```
virtual __fastcall int GetWidth();
```

Description

This is GetWidth, a member of class TIcon.

73.1.12.4.6 TIcon.HandleAllocated Method

Indicates whether the TIcon (see page 14537) object has acquired a handle to the underlying GDI object.

Pascal

```
function HandleAllocated: Boolean;
```

C++

```
__fastcall Boolean HandleAllocated();
```

Description

Use HandleAllocated to determine whether the Handle (see page 14546) property is set to an HICON value. HandleAllocated indicates whether the underlying GDI object has already been created. Use HandleAllocated rather than reading the Handle (see page 14546) property. When an application reads the Handle (see page 14546) property, it automatically creates an icon and returns its handle if the underlying GDI object does not already exist.

See Also

Handle (see page 14546)

73.1.12.4.7 TIcon.LoadFromClipboardFormat Method

Prevents applications from loading icons from the clipboard.

Pascal

```
procedure LoadFromClipboardFormat(AFormat: Word; AData: THandle; APalette: HPALETTE); override;
```

C++

```
virtual __fastcall LoadFromClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette);
```

Description

LoadFromClipboardFormat is overridden in TIcon (see page 14537) because the clipboard format is not supported.

See Also

RegisterClipboardFormat (see page 14592)

73.1.12.4.8 TIcon.LoadFromResourceID Method

Pascal

```
procedure LoadFromResourceID(Instance: THandle; ResID: Integer);
```

C++

```
__fastcall LoadFromResourceID(THandle Instance, int ResID);
```

Description

This is LoadFromResourceID, a member of class TIcon.

73.1.12.4.9 TIcon.LoadFromResourceName Method

Loads an icon resource into the icon object.

Pascal

```
procedure LoadFromResourceName(Instance: THandle; const ResName: String);
```

C++

```
__fastcall LoadFromResourceName(THandle Instance, const AnsiString ResName);
```

Description

LoadFromResourceName loads the specified icon resource from a module's executable file.

Instance is the handle of the module that contains the resource.

ResName is the name of the resource to load.

See Also

LoadFromMimeSource

LoadFromStream (see page 14544)

73.1.12.4.10 TIcon.LoadFromStream Method

Loads the icon from a stream.

Pascal

```
procedure LoadFromStream(Stream: TStream); override;
```

C++

```
virtual __fastcall LoadFromStream(TStream Stream);
```

Description

Use LoadFromStream to read the icon image from a stream. Set the Stream parameter to a stream object that provides access to the memory image of the icon. To load the icon from a .ico file, use the LoadFromFile (see page 14523) method. To load the icon from the Clipboard, use the LoadFromClipboardFormat (see page 14543) method.

See Also

LoadFromClipboardFormat (see page 14522)

LoadFromFile (see page 14523)

SaveToStream (see page 14545)

73.1.12.4.11 TIcon.ReleaseHandle Method

Releases the Windows GDI object represented by the icon.

Pascal

```
function ReleaseHandle: HICON;
```

C++

```
__fastcall HICON ReleaseHandle();
```

Description

Call ReleaseHandle to release the resources used to represent the icon. ReleaseHandle sets the Handle (see page 14546)

property to nil (Delphi) or NULL (C++).

ReleaseHandle informs the TIcon (see page 14537) instance that it is no longer responsible for destroying the icon handle.

See Also

Handle (see page 14546)

73.1.12.4.12 TIcon.SaveToClipboardFormat Method

Prevents applications from saving icons to the clipboard.

Pascal

```
procedure SaveToClipboardFormat(var Format: Word; var Data: THandle; var APalette: HPALETTE);
override;
```

C++

```
virtual __fastcall SaveToClipboardFormat(Word Format, THandle Data, HPALETTE APalette);
```

Description

SaveToClipboardFormat is overridden in TIcon (see page 14537) because the format is not supported for the Clipboard.

See Also

LoadFromClipboardFormat (see page 14543)

73.1.12.4.13 TIcon.SaveToStream Method

Saves the icon to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream); override;
```

C++

```
virtual __fastcall SaveToStream(TStream Stream);
```

Description

Use SaveToStream to write the icon image to a stream. Specify the Stream parameter as the stream object that receives the memory image of the icon. To write the icon to a .ico file, use the SaveToFile (see page 14525) method. To write the icon to the Clipboard, use the SaveToClipboardFormat (see page 14545) method.

See Also

LoadFromStream (see page 14544)

SaveToClipboardFormat (see page 14525)

SaveToFile (see page 14525)

73.1.12.4.14 TIcon.SetHeight Method

Pascal

```
procedure SetHeight(Value: Integer); override;
```

C++

```
virtual __fastcall SetHeight(int Value);
```

Description

This is SetHeight, a member of class TIcon.

73.1.12.4.15 TIcon.SetSize Method

Specifies the size of the icon.

Pascal

```
procedure SetSize(AWidth: Integer; AHeight: Integer); override;
```

C++

```
virtual __fastcall SetSize(int AWidth, int AHeight);
```

Description

Use the SetSize method to set both the height and width of the icon. This results in better performance than setting the height and width separately.

73.1.12.4.16 TIcon.SetTransparent Method

Pascal

```
procedure SetTransparent(Value: Boolean); override;
```

C++

```
virtual __fastcall SetTransparent(Boolean Value);
```

Description

This is SetTransparent, a member of class TIcon.

73.1.12.4.17 TIcon.SetWidth Method

Pascal

```
procedure SetWidth(Value: Integer); override;
```

C++

```
virtual __fastcall SetWidth(int Value);
```

Description

This is SetWidth, a member of class TIcon.

73.1.12.5 TIcon Properties

The properties of the TIcon class are listed here.

Public Properties

	Name	Description
	Handle (see page 14546)	Provides access to the Windows GDI icon handle.

73.1.12.5.1 TIcon.Handle Property

Provides access to the Windows GDI icon handle.

Pascal

```
property Handle: HICON;
```

C++

```
__property HICON Handle;
```

Description

Use Handle to specify the icon when using a Windows API function that requires the handle of an icon object. The Handle property is nil (Delphi) or NULL (C++) if the icon has not been loaded.

Call the ReleaseHandle (see page 14544) method before changing the icon image by setting the Handle property. The icon image can also be loaded by assigning another icon object or using the LoadFromClipboardFormat (see page 14543), LoadFromFile (see page 14523), or LoadFromStream (see page 14544) method.

See Also

Assign (see page 14541)

Empty

LoadFromClipboardFormat (see page 14522)

LoadFromFile (see page 14523)

LoadFromStream (see page 14544)

ReleaseHandle (see page 14544)

73.1.13 TIconImage Class

TIconImage is the internal representation of the bitmap image for a TIcon (see page 14537) object.

Class Hierarchy

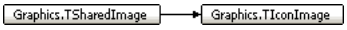

Pascal

```
TIconImage = class(TSharedImage);
```

C++

```
class TIconImage : public TSharedImage;
```

File

Graphics

Description

TIconImage is used for internal implementation only. It represents the internal image of the bitmap encapsulated by the TIcon (see page 14537) object. All of the data and methods introduced in TIconImage are private. They contain information about the specific icon resource, including pointers to the standard HICON structures defined by Windows. TIconImage is referenced in the constructor of the TIcon (see page 14537) object when the icon image is created.

See Also

TIcon (see page 14537)

73.1.13.1 TIconImage Members

The following tables list the members exposed by TIconImage.

Protected Methods

	Name	Description
V	FreeHandle (see page 14548)	This is FreeHandle, a member of class TIconImage.
	Reference (see page 14604)	This is Reference, a member of class TSharedImage.
	Release (see page 14604)	This is Release, a member of class TSharedImage.

Protected Properties

	Name	Description
R	RefCount (see page 14605)	This is RefCount, a member of class TSharedImage.

Public Methods

	Name	Description
V	Destroy (see page 14548)	Destroys the TIconImage (see page 14547) instance and frees its memory.

73.1.13.2 TIconImage.Destroy Destructor

Destroys the TIconImage (see page 14547) instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIconImage();
```

Description

You do not need to destroy the icon image. The TIconImage (see page 14547) object is automatically destroyed by the icon object.

See Also

Free (see page 22300)

73.1.13.3 TIconImage Methods

The methods of the TIconImage class are listed here.

Protected Methods

	Name	Description
V	FreeHandle (see page 14548)	This is FreeHandle, a member of class TIconImage.

73.1.13.3.1 TIconImage.FreeHandle Method

Pascal

```
procedure FreeHandle; override;
```

C++

```
virtual __fastcall FreeHandle();
```

Description

This is FreeHandle, a member of class TIconImage.

73.1.14 TMetafile Class

TMetafile is an encapsulation of the Win32 Enhanced (see page 14562) metafile.

Class Hierarchy

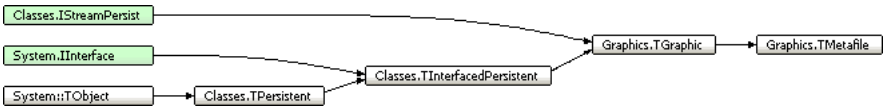

Pascal

```
TMetafile = class(TGraphic);
```

C++

```
class TMetafile : public TGraphic;
```

File

Graphics

Description

TMetafile contains a metafile graphic (EMF file format).

Properties of TMetafile indicate the size and characteristics of the metafile.

To draw a metafile on a canvas, call the Draw (see page 14555) or StretchDraw methods of a TCanvas (see page 14441) object, passing a TMetafile as a parameter. The Enhanced (see page 14562) property determines how the metafile will be stored on disk. If Enhanced (see page 14562) is true, the metafile is stored as an .EMF (Win32 Enhanced (see page 14562) Metafile) file. If Enhanced (see page 14562) is false, the metafile is stored as a .WMF (Windows 3.1 Metafile, with Aldus header).

See Also

TMetafileImage (see page 14569)

Using the Main VCL

73.1.14.1 TMetafile Members

The following tables list the members exposed by TMetafile.

Protected Methods

	Name	Description
	_AddRef (see page 2853)	Increments the reference count for the interfaced object's interface.
	_Release (see page 2854)	Decrements the reference count for the interfaced object's interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Changed (see page 14520)	This is Changed, a member of class TGraphic.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	DefineProperties (see page 14520)	Provides an interface for a method that reads and writes otherwise unpublished data.
	Draw (see page 14555)	This is Draw, a member of class TMetafile.
	Equals (see page 14521)	This is Equals, a member of class TGraphic.
	GetEmpty (see page 14555)	This is GetEmpty, a member of class TMetafile.
	GetHeight (see page 14555)	This is GetHeight, a member of class TMetafile.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 14556)	This is GetPalette, a member of class TMetafile.
	GetTransparent (see page 14522)	This is GetTransparent, a member of class TGraphic.
	GetWidth (see page 14556)	This is GetWidth, a member of class TMetafile.
	Progress (see page 14524)	This is Progress, a member of class TGraphic.
	ReadData (see page 14557)	This is ReadData, a member of class TMetafile.
	ReadEMFStream (see page 14557)	This is ReadEMFStream, a member of class TMetafile.
	ReadWMFStream (see page 14558)	This is ReadWMFStream, a member of class TMetafile.
	SetHeight (see page 14559)	This is SetHeight, a member of class TMetafile.
	SetPalette (see page 14527)	This is SetPalette, a member of class TGraphic.
	SetTransparent (see page 14560)	This is SetTransparent, a member of class TMetafile.
	SetWidth (see page 14560)	This is SetWidth, a member of class TMetafile.
	TestEMF (see page 14560)	This is TestEMF, a member of class TMetafile.
	WriteData (see page 14560)	This is WriteData, a member of class TMetafile.
	WriteEMFStream (see page 14561)	This is WriteEMFStream, a member of class TMetafile.
	WriteWMFStream (see page 14561)	This is WriteWMFStream, a member of class TMetafile.

Public Events

	Name	Description
	OnChange (see page 14531)	Occurs whenever a graphical image changes.
	OnProgress (see page 14531)	Occurs when a graphical image is in the process of changing.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 2854)	Obtains a reference to the Owner's interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 14554)	Copies an object to the metafile.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 14555)	Deletes the metafile image.
	Create (see page 14551)	Instantiates a metafile object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 14553)	Destroys an instance of a metafile object.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	HandleAllocated (see page 14556)	Indicates whether the TMetafile (see page 14549) object has acquired a handle to the underlying GDI object.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	LoadFromClipboardFormat (see page 14556)	Loads a metafile from the Clipboard.
V	LoadFromFile (see page 14523)	Loads a graphic image stored in a file.
V	LoadFromStream (see page 14557)	Loads the metafile from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	QueryInterface (see page 2854)	Returns a reference to a specified interface if the object supports that interface.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReleaseHandle (see page 14558)	Releases the Windows GDI object represented by the metafile.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SaveToClipboardFormat (see page 14558)	Saves a metafile to a Clipboard format.
V	SaveToFile (see page 14559)	Writes the metafile to disk.
V	SaveToStream (see page 14559)	Writes the metafile to a stream.
V	SetSize (see page 14560)	Specifies the size of the metafile.

Public Properties

	Name	Description
R	CreatedBy (see page 14561)	Specifies the name of the author or application used to create the metafile.
R	Description (see page 14562)	Provides an optional text description that is embedded in the metafile.
R	Empty (see page 14528)	Indicates whether the graphics object contains a graphic.
	Enhanced (see page 14562)	Determines how the metafile will be stored on disk.
	Handle (see page 14563)	Provides access to the Windows GDI metafile handle, for accessing the GDI metafile object.
	Height (see page 14528)	Specifies the vertical size of the graphic in pixels.
	Inch (see page 14563)	Returns the units per inch that are used for the metafile's coordinate mapping.
	MMHeight (see page 14563)	Contains the height of the metafile image in 0.01 millimeter units, the native scale used by enhanced metafiles.
	MMWidth (see page 14564)	Contains the width of the metafile image in 0.01 millimeter units, the native scale used by enhanced metafiles.
	Modified (see page 14528)	Indicates whether the graphics object has been changed or edited.
	Palette (see page 14529)	Indicates the color palette of the graphical image.
	PaletteModified (see page 14529)	Indicates whether the palette has changed.
	Transparent (see page 14529)	Indicates whether the image covers its rectangular area.
	Width (see page 14530)	Determines the maximum width of the graphics object in pixels.

73.1.14.2 TMetafile.Create Constructor

Instantiates a metafile object.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TMetafile();
```

Description

Call Create to instantiate a metafile at runtime.

Create allocates memory for a metafile object, and calls the inherited Create. Then sets the Enhanced (see page 14562) and Transparent (see page 14529) properties to true.

To create a metafile image from scratch, draw the image in a metafile canvas. When the metafile canvas is destroyed, it transfers the image into the metafile object provided to the metafile canvas constructor. After the image is drawn on the canvas and the canvas is destroyed, the image is 'playable' in the metafile object.

C++ Examples:

```
/*
This example shows how to create or augment a metafile using
a metafile canvas object.  This metafile can then be used to
draw on the canvas of another object such as a paintbox or a
printer.
*/
TMetafile *MyMetafile;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
MyMetafile = new TMetafile;
TMetafileCanvas *canvas = new TMetafileCanvas(MyMetafile, 0);
canvas->Brush->Color = clRed;
canvas->Ellipse(0, 0, 100, 200);
// ...
delete canvas;
Form1->Canvas->Draw(0, -50, MyMetafile); //1 red circle
PaintBox1->Canvas->Draw(0, 0, MyMetafile); //1 red circle
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  if (MyMetafile == NULL) return;   // click Button1 first!
  TMetafileCanvas *canvas = new TMetafileCanvas(MyMetafile, 0);
  canvas->Draw(-50, 0, MyMetafile);
  canvas->Brush->Color = clBlue;
  canvas->Ellipse(100, 100, 300, 200);
  // ...
  delete canvas;
  Form1->Canvas->Draw(0, 0, MyMetafile); // 1 red circle and 1 blue circle
  PaintBox1->Canvas->Draw(0, 0, MyMetafile); // 1 red circle and 1 blue circle
}
```

Delphi Examples:

```
{
This example shows how to create or augment a metafile using
a metafile canvas object.  This metafile can then be used to
draw on the canvas of another object such as a paintbox or a
printer.
}

var
  MyMetafile: TMetafile;

procedure TForm1.Button1Click(Sender: TObject);
begin
  MyMetafile := TMetafile.Create;
  with TMetafileCanvas.Create(MyMetafile, 0) do
  try
    Brush.Color := clRed;
    Ellipse(0, 0, 100, 200);
    //  ...
```

```
    finally
      Free;
    end;
    Form1.Canvas.Draw(0, 0, MyMetafile); {1 red circle }
    PaintBox1.Canvas.Draw(0, -50, MyMetafile); {1 red circle }
  end;

  {
  To add to an existing metafile image, create a metafile
  canvas and play the source metafile into the metafile canvas:
  }
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    if (MyMetafile = nil) then exit;  { click Button1 first! }

    with TMetafileCanvas.Create(MyMetafile, 0) do
    try
      Draw(-50, 0, MyMetafile);
      Brush.Color := clBlue;
      Ellipse(100, 100, 300, 200);
  // ...
    finally
      Free;
    end;
    Form1.Canvas.Draw(0, 0, MyMetafile); {1 red circle and 1 blue circle }
    PaintBox1.Canvas.Draw(0, 0, MyMetafile); {1 red circle and 1 blue circle }
  end;
```

See Also

Enhanced (see page 14562)

Create (see page 22287)

TMetafileCanvas (see page 14564)

TMetafileImage (see page 14569)

Transparent (see page 14529)

73.1.14.3 TMetafile.Destroy Destructor

Destroys an instance of a metafile object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMetafile();
```

Description

Do not call Destroy directly in an application. Instead, an application should call Free. Free verifies that the TMetafile (see page 14549) reference is not nil before it calls Destroy.

Destroy releases the image, which frees the image and the handle when the reference count on the metafile image is zero. Then Destroy calls the inherited Destroy.

See Also

Free (see page 22300)

TMetafileImage (see page 14569)

Destroy (see page 22287)

73.1.14.4 TMetafile Methods

The methods of the TMetafile class are listed here.

Protected Methods

	Name	Description
	Draw (see page 14555)	This is Draw, a member of class TMetafile.
	GetEmpty (see page 14555)	This is GetEmpty, a member of class TMetafile.
	GetHeight (see page 14555)	This is GetHeight, a member of class TMetafile.
	GetPalette (see page 14556)	This is GetPalette, a member of class TMetafile.
	GetWidth (see page 14556)	This is GetWidth, a member of class TMetafile.
	ReadData (see page 14557)	This is ReadData, a member of class TMetafile.
	ReadEMFStream (see page 14557)	This is ReadEMFStream, a member of class TMetafile.
	ReadWMFStream (see page 14558)	This is ReadWMFStream, a member of class TMetafile.
	SetHeight (see page 14559)	This is SetHeight, a member of class TMetafile.
	SetTransparent (see page 14560)	This is SetTransparent, a member of class TMetafile.
	SetWidth (see page 14560)	This is SetWidth, a member of class TMetafile.
	TestEMF (see page 14560)	This is TestEMF, a member of class TMetafile.
	WriteData (see page 14560)	This is WriteData, a member of class TMetafile.
	WriteEMFStream (see page 14561)	This is WriteEMFStream, a member of class TMetafile.
	WriteWMFStream (see page 14561)	This is WriteWMFStream, a member of class TMetafile.

Public Methods

	Name	Description
	Assign (see page 14554)	Copies an object to the metafile.
	Clear (see page 14555)	Deletes the metafile image.
	HandleAllocated (see page 14556)	Indicates whether the TMetafile (see page 14549) object has acquired a handle to the underlying GDI object.
	LoadFromClipboardFormat (see page 14556)	Loads a metafile from the Clipboard.
	LoadFromStream (see page 14557)	Loads the metafile from a stream.
	ReleaseHandle (see page 14558)	Releases the Windows GDI object represented by the metafile.
	SaveToClipboardFormat (see page 14558)	Saves a metafile to a Clipboard format.
	SaveToFile (see page 14559)	Writes the metafile to disk.
	SaveToStream (see page 14559)	Writes the metafile to a stream.
	SetSize (see page 14560)	Specifies the size of the metafile.

73.1.14.4.1 TMetafile.Assign Method

Copies an object to the metafile.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Assign copies the Enhanced (see page 14562) and Palette (see page 14529) properties of an object to the metafile. If the metafile is nil (Delphi) or NULL (C++), Assign creates one. Otherwise, Assign calls the inherited method. For assignment, metafiles are polymorphic with other TGraphic (see page 14516) objects, and with TPicture (see page 14584).

See Also

Enhanced (see page 14562)

Palette

Assign (see page 2930)

Assign (see page 14587)

Graphic (see page 14600)

73.1.14.4.2 TMetafile.Clear Method

Deletes the metafile image.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Use Clear to release the old metafile image, create a new image, and increment a reference count on it.

See Also

TMetafileImage (see page 14569)

73.1.14.4.3 TMetafile.Draw Method

Pascal

```
procedure Draw(ACanvas: TCanvas; const Rect: TRect); override;
```

C++

```
virtual __fastcall Draw(TCanvas ACanvas, const TRect Rect);
```

Description

This is Draw, a member of class TMetafile.

73.1.14.4.4 TMetafile.GetEmpty Method

Pascal

```
function GetEmpty: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetEmpty();
```

Description

This is GetEmpty, a member of class TMetafile.

73.1.14.4.5 TMetafile.GetHeight Method

Pascal

```
function GetHeight: Integer; override;
```

C++

```
virtual __fastcall int GetHeight();
```

Description

This is GetHeight, a member of class TMetafile.

73.1.14.4.6 TMetafile.GetPalette Method

Pascal

```
function GetPalette: HPALETTE; override;
```

C++

```
virtual __fastcall HPALETTE GetPalette();
```

Description

This is GetPalette, a member of class TMetafile.

73.1.14.4.7 TMetafile.GetWidth Method

Pascal

```
function GetWidth: Integer; override;
```

C++

```
virtual __fastcall int GetWidth();
```

Description

This is GetWidth, a member of class TMetafile.

73.1.14.4.8 TMetafile.HandleAllocated Method

Indicates whether the TMetafile (see page 14549) object has acquired a handle to the underlying GDI object.

Pascal

```
function HandleAllocated: Boolean;
```

C++

```
__fastcall Boolean HandleAllocated();
```

Description

Use HandleAllocated to determine whether the Handle (see page 14563) property is set to an HMETAFILE (see page 14646) value. HandleAllocated indicates whether the underlying GDI object has already been created. Use HandleAllocated rather than reading the Handle (see page 14563) property. When an application reads the Handle (see page 14563) property, it automatically creates a metafile and returns its handle if the underlying GDI object does not already exist.

See Also

Handle (see page 14563)

73.1.14.4.9 TMetafile.LoadFromClipboardFormat Method

Loads a metafile from the Clipboard.

Pascal

```
procedure LoadFromClipboardFormat(AFormat: Word; AData: THandle; APalette: HPALETTE); override;
```

C++

```
virtual __fastcall LoadFromClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette);
```

Description

LoadFromClipboardFormat replaces the current image with the data in the clipboard. LoadFromClipboardFormat ignores the AData and APalette parameters.

See Also

RegisterClipboardFormat (see page 14592)

73.1.14.4.10 TMetafile.LoadFromStream Method

Loads the metafile from a stream.

Pascal

```
procedure LoadFromStream(Stream: TStream); override;
```

C++

```
virtual __fastcall LoadFromStream(TStream Stream);
```

Description

Use LoadFromStream by specifying the stream from which the metafile is loaded as the value of Stream. LoadFromStream reads a metafile from a stream and replaces the current image with it.

See Also

SaveToStream (see page 14559)

73.1.14.4.11 TMetafile.ReadData Method

Pascal

```
procedure ReadData(Stream: TStream); override;
```

C++

```
virtual __fastcall ReadData(TStream Stream);
```

Description

This is ReadData, a member of class TMetafile.

73.1.14.4.12 TMetafile.ReadEMFStream Method

Pascal

```
procedure ReadEMFStream(Stream: TStream);
```

C++

```
__fastcall ReadEMFStream(TStream Stream);
```

Description

This is ReadEMFStream, a member of class TMetafile.

73.1.14.4.13 TMetafile.ReadWMFStream Method

Pascal

```
procedure ReadWMFStream(Stream: TStream; Length: Longint);
```

C++

```
__fastcall ReadWMFStream(TStream Stream, Longint Length);
```

Description

This is ReadWMFStream, a member of class TMetafile.

73.1.14.4.14 TMetafile.ReleaseHandle Method

Releases the Windows GDI object represented by the metafile.

Pascal

```
function ReleaseHandle: HENHMETAFILE;
```

C++

```
__fastcall HENHMETAFILE ReleaseHandle();
```

Description

Call ReleaseHandle to release the resources used to represent the metafile. ReleaseHandle sets the Handle (see page 14563) property to nil. NULL.

Call ReleaseHandle before setting the Handle (see page 14563) property, so that the resources associated with the metafile are not lost.

See Also

Handle (see page 14563)

73.1.14.4.15 TMetafile.SaveToClipboardFormat Method

Saves a metafile to a Clipboard format.

Pascal

```
procedure SaveToClipboardFormat(var AFormat: Word; var AData: THandle; var APalette:
HPALETTE); override;
```

C++

```
virtual __fastcall SaveToClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette);
```

Description

Use SaveToClipboardFormat to copy the metafile to a Clipboard format. The resulting values can then be copied to the Windows clipboard using the clipboard's SetAsHandle method.

The metafile's palette is returned in the APalette parameter, the format in the AFormat parameter, and a handle to the data in the AData parameter. Before the metafile can be saved, an application must have registered the format with the metafile object using the RegisterClipboardFormat method.

To save a metafile to a file, call SaveToFile (see page 14559).

See Also

LoadFromClipboardFormat (see page 14556)

SaveToFile (see page 14559)

SetAsHandle (see page 3343)

73.1.14.4.16 TMetafile.SaveToFile Method

Writes the metafile to disk.

Pascal

```
procedure SaveToFile(const Filename: String); override;
```

C++

```
virtual __fastcall SaveToFile(const AnsiString Filename);
```

Description

Use SaveToFile to save the metafile to a file provided by the Filename parameter. To load a metafile from a file, call the LoadFromFile (see page 14523) method.

See Also

SaveToStream (see page 14559)

LoadFromFile (see page 14523)

73.1.14.4.17 TMetafile.SaveToStream Method

Writes the metafile to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream); override;
```

C++

```
virtual __fastcall SaveToStream(TStream Stream);
```

Description

Use SaveToStream to save the metafile to the stream specified by the value of the Stream parameter.

See Also

LoadFromStream (see page 14557)

SaveToFile (see page 14559)

73.1.14.4.18 TMetafile.SetHeight Method

Pascal

```
procedure SetHeight(Value: Integer); override;
```

C++

```
virtual __fastcall SetHeight(int Value);
```

Description

This is SetHeight, a member of class TMetafile.

73.1.14.4.19 TMetafile.SetSize Method

Specifies the size of the metafile.

Pascal

```
procedure SetSize(AWidth: Integer; AHeight: Integer); override;
```

C++

```
virtual __fastcall SetSize(int AWidth, int AHeight);
```

Description

Use the SetSize method to set both the height and width of the metafile. This results in better performance than setting the height and width separately.

73.1.14.4.20 TMetafile.SetTransparent Method

Pascal

```
procedure SetTransparent(Value: Boolean); override;
```

C++

```
virtual __fastcall SetTransparent(Boolean Value);
```

Description

This is SetTransparent, a member of class TMetafile.

73.1.14.4.21 TMetafile.SetWidth Method

Pascal

```
procedure SetWidth(Value: Integer); override;
```

C++

```
virtual __fastcall SetWidth(int Value);
```

Description

This is SetWidth, a member of class TMetafile.

73.1.14.4.22 TMetafile.TestEMF Method

Pascal

```
function TestEMF(Stream: TStream): Boolean;
```

C++

```
__fastcall Boolean TestEMF(TStream Stream);
```

Description

This is TestEMF, a member of class TMetafile.

73.1.14.4.23 TMetafile.WriteData Method

Pascal

```
procedure WriteData(Stream: TStream); override;
```

C++

```
virtual __fastcall WriteData(TStream Stream);
```

Description

This is WriteData, a member of class TMetafile.

73.1.14.4.24 TMetafile.WriteEMFStream Method

Pascal

```
procedure WriteEMFStream(Stream: TStream);
```

C++

```
__fastcall WriteEMFStream(TStream Stream);
```

Description

This is WriteEMFStream, a member of class TMetafile.

73.1.14.4.25 TMetafile.WriteWMFStream Method

Pascal

```
procedure WriteWMFStream(Stream: TStream);
```

C++

```
__fastcall WriteWMFStream(TStream Stream);
```

Description

This is WriteWMFStream, a member of class TMetafile.

73.1.14.5 TMetafile Properties

The properties of the TMetafile class are listed here.

Public Properties

	Name	Description
R	CreatedBy (see page 14561)	Specifies the name of the author or application used to create the metafile.
R	Description (see page 14562)	Provides an optional text description that is embedded in the metafile.
	Enhanced (see page 14562)	Determines how the metafile will be stored on disk.
	Handle (see page 14563)	Provides access to the Windows GDI metafile handle, for accessing the GDI metafile object.
	Inch (see page 14563)	Returns the units per inch that are used for the metafile's coordinate mapping.
	MMHeight (see page 14563)	Contains the height of the metafile image in 0.01 millimeter units, the native scale used by enhanced metafiles.
	MMWidth (see page 14564)	Contains the width of the metafile image in 0.01 millimeter units, the native scale used by enhanced metafiles.

73.1.14.5.1 TMetafile.CreatedBy Property

Specifies the name of the author or application used to create the metafile.

Pascal

```
property CreatedBy: String;
```

C++

```
__property AnsiString CreatedBy;
```

Description

Use CreatedBy to set the string to an optional name of the author or application used to create the metafile. To set the CreatedBy string of a new metafile, call the TMetaFileCanvas CreateWithComment constructor (Delphi) or the TMetafileCanvas (see page 14564) constructor that takes a comment (C++).

See Also

TMetafileCanvas (see page 14564)

CreateWithComment (see page 14568)

Create_c

73.1.14.5.2 TMetafile.Description Property

Provides an optional text description that is embedded in the metafile.

Pascal

```
property Description: String;
```

C++

```
__property AnsiString Description;
```

Description

Set the Description string of a new metafile by calling the TMetaFileCanvas CreateWithComment constructor (Delphi) or the TMetafileCanvas (see page 14564) constructor that takes a comment (C++)..

See Also

TMetafileCanvas (see page 14564)

CreateWithComment (see page 14568)

Create_c

73.1.14.5.3 TMetafile.Enhanced Property

Determines how the metafile will be stored on disk.

Pascal

```
property Enhanced: Boolean;
```

C++

```
__property Boolean Enhanced;
```

Description

Use Enhanced to determine how the metafile is stored on disk. If Enhanced is true, the metafile is stored as an .EMF (Win32 Enhanced Metafile). If Enhanced is false, the metafile is stored as a .WMF (Windows 3.1 Metafile, with Aldus header).

The in-memory format is always EMF. WMF has very limited capabilities; storing as WMF will lose information that would be retained by EMF. This property is set to match the metafile type when loaded from a stream or file. If loaded as WMF, then save as WMF.

73.1.14.5.4 TMetafile.Handle Property

Provides access to the Windows GDI metafile handle, for accessing the GDI metafile object.

Pascal

```
property Handle: HENHMETAFILE;
```

C++

```
__property HENHMETAFILE Handle;
```

Description

Use Handle when calling a Windows API function that requires the handle of a metafile object. Pass the handle from the Handle property of the metafile object to the function.

See Also

DestroyMetaPict (see page 18011)

73.1.14.5.5 TMetafile.Inch Property

Returns the units per inch that are used for the metafile's coordinate mapping.

Pascal

```
property Inch: Word;
```

C++

```
__property Word Inch;
```

Description

Use Inch to find the units per inch assumed by a WMF metafile, and to alter scale when writing as WMF, but otherwise this property is obsolete. Enhanced (see page 14562) metafiles maintain complete scale information internally. For example, if the metafile was created in a Twips coordinate system (using MM_TWIPS mapping), the value of Inch is 1440.

73.1.14.5.6 TMetafile.MMHeight Property

Contains the height of the metafile image in 0.01 millimeter units, the native scale used by enhanced metafiles.

Pascal

```
property MMHeight: Integer;
```

C++

```
__property int MMHeight;
```

Description

MMHeight is used for a more accurate reading of the vertical size of the graphic. The Height (see page 14528) property, by contrast, is always in screen device pixel units; to avoid loss of precision in converting between device pixels and millimeters, set or read the dimensions in millimeters with MMHeight.

See Also

MMWidth (see page 14564)

73.1.14.5.7 TMetafile.MMWidth Property

Contains the width of the metafile image in 0.01 millimeter units, the native scale used by enhanced metafiles.

Pascal

```
property MMWidth: Integer;
```

C++

```
__property int MMWidth;
```

Description

MMWidth is used for a more accurate reading of the horizontal size of the graphic. The Width (see page 14530) property, by contrast, is always in screen device pixel units; to avoid loss of precision in converting between device pixels and millimeters, set or read the dimensions in millimeters with MMWidth.

The MMWidth property is always in screen device pixel units.

See Also

MMHeight (see page 14563)

73.1.15 TMetafileCanvas Class

TMetafileCanvas allows applications to create metafile images from scratch.

Class Hierarchy

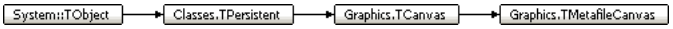

Pascal

```
TMetafileCanvas = class(TCanvas);
```

C++

```
class TMetafileCanvas : public TCanvas;
```

File

Graphics

Description

Use TMetafileCanvas as a surface on which to draw a metafile image. When the canvas is destroyed, it transfers the image into the metafile object provided to the canvas constructor. After the image is drawn on the canvas and the canvas is destroyed, the image is 'playable' in the metafile object.

See Also

TMetafile (see page 14549)

73.1.15.1 TMetafileCanvas Members

The following tables list the members exposed by TMetafileCanvas.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 14450)	This is Changed, a member of class TCanvas.
	Changing (see page 14450)	This is Changing, a member of class TCanvas.
	CreateHandle (see page 14454)	This is CreateHandle, a member of class TCanvas.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.
	RequiredState (see page 14470)	This is RequiredState, a member of class TCanvas.

Public Events

	Name	Description
	OnChange (see page 14496)	Occurs when the image has just changed.
	OnChanging (see page 14496)	Occurs just before a change is made to the image.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Arc (see page 14448)	Draws an arc on the image along the perimeter of the ellipse bounded by the specified rectangle.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BrushCopy (see page 14449)	Copies a portion of a bitmap onto a rectangle on the canvas, replacing one of the colors of the bitmap with the brush of the canvas.
	Chord (see page 14451)	Draws a closed figure represented by the intersection of a line and an ellipse.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CopyRect (see page 14452)	Copies part of an image from another canvas into the canvas.
	Create (see page 14567)	Creates an instance of TMetafileCanvas (see page 14564) and acquires a handle to a metafile device context.
	CreateWithComment (see page 14568)	Creates an instance of TMetafileCanvas (see page 14564) and acquires a handle to a metafile device context with a comment string.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 14569)	Destroys an instance of the TMetafileCanvas (see page 14564) after transferring the image to the TMetafile (see page 14549) object that was passed to the constructor.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Draw (see page 14454)	Renders the graphic specified by the Graphic parameter on the canvas at the location given by the coordinates (X, Y).
	DrawFocusRect (see page 14455)	Draws a rectangle in the style used to indicate that the object inside the rectangle has focus.
	Ellipse (see page 14457)	This is the overview for the Ellipse method overload.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FillRect (see page 14458)	Fills the specified rectangle on the canvas using the current brush.
	FloodFill (see page 14459)	Fills an area of the canvas using the current brush.
	FrameRect (see page 14460)	Draws a rectangle using the Brush (see page 14484) of the canvas to draw the border.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	HandleAllocated (see page 14461)	Indicates whether the TCanvas (see page 14441) object has acquired a handle to a device context.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LineTo (see page 14462)	Draws a line on the canvas from PenPos (see page 14494) to the point specified by X and Y, and sets the pen position to (X, Y).
	Lock (see page 14463)	Prevents other threads from drawing on the canvas.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveTo (see page 14463)	Changes the current drawing position to the point (X,Y).
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Pie (see page 14464)	Draws a pie-shaped the section of the ellipse bounded by the rectangle (X1, Y1) and (X2, Y2) on the canvas.
	PolyBezier (see page 14464)	Draws a set of Bezier curves.
	PolyBezierTo (see page 14465)	Draws a set of Bezier curves and updates the value of PenPos (see page 14494).
	Polygon (see page 14466)	Draws a series of lines on the canvas connecting the points passed in and closing the shape by drawing a line from the last point to the first point.
	Polyline (see page 14467)	Draws a series of lines on the canvas with the current pen, connecting each of the points passed to it in Points.
	Rectangle (see page 14468)	This is the overview for the Rectangle method overload.
	Refresh (see page 14470)	Deselects the Pen (see page 14492), Brush (see page 14484), and Font (see page 14488) from the device context.
	RoundRect (see page 14471)	Draws a rectangle with rounded corners on the canvas.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	StretchDraw (see page 14472)	Draws the graphic specified by the Graphic parameter in the rectangle specified by the Rect parameter.
	TextExtent (see page 14474)	Returns the width and height, in pixels, of a string rendered in the current font.
	TextHeight (see page 14474)	Returns the height, in pixels, of a string rendered in the current font.
	TextOut (see page 14475)	Writes a string on the canvas, starting at the point (X,Y), and then updates the PenPos (see page 14494) to the end of the string.
	TextRect (see page 14480)	This is the overview for the TextRect method overload.
	TextWidth (see page 14481)	Returns the width, in pixels, of a string rendered in the current font.
	TryLock (see page 14483)	Locks the canvas if it is currently unlocked.
	Unlock (see page 14483)	Decreases the LockCount (see page 14492) property, allowing other threads to execute when LockCount (see page 14492) reaches 0.

Public Properties

	Name	Description
R	CanvasOrientation (see page 14485)	Determines the orientation of the canvas as left-to-right or right-to-left.
R	ClipRect (see page 14486)	Read-only property that specifies the boundaries of the clipping rectangle.
	Handle (see page 14489)	Specifies the handle for this canvas.
R	LockCount (see page 14492)	Indicates the number of times the canvas has been locked to prevent interference from other threads.
	PenPos (see page 14494)	Specifies the current drawing position of the Pen (see page 14492).
	Pixels (see page 14494)	Specifies the color of the pixels within the current ClipRect (see page 14486).
	TextFlags (see page 14495)	Specifies how text is written to the canvas.

Published Properties

	Name	Description
	Brush (see page 14484)	Determines the color and pattern for filling graphical shapes and backgrounds.

	CopyMode (see page 14487)	Specifies how a graphical image is copied onto the canvas.
	Font (see page 14488)	Specifies the font to use when writing text on the image.
	Pen (see page 14492)	Specifies the kind of pen the canvas uses for drawing lines and outlining shapes.

73.1.15.2 TMetafileCanvas.Create Constructor

Creates an instance of TMetafileCanvas (see page 14564) and acquires a handle to a metafile device context.

Pascal

```
constructor Create(AMetafile: TMetafile; ReferenceDevice: HDC);
```

C++

```
__fastcall TMetafileCanvas(TMetafile AMetafile, HDC ReferenceDevice);
```

Description

Call Create to create an instantiate a TMetafileCanvas (see page 14564) object. Pass in the associated metafile whose image is about to be drawn as the AMetafile parameter. Pass in a handle to a device context on which to base the metafile's device context as the ReferenceDevice parameter. In most cases, this is the device context for a window on which the image is normally drawn.

Create sets the size of the TMetafile (see page 14549) object from ReferenceDevice if it does not already have the MMHeight and MMWidth properties set. Create then creates a metafile device context, and sets the Handle (see page 14489) property to its handle. All subsequent drawing methods will draw to the metafile device context.

When the TMetafileCanvas (see page 14564) is destroyed, The CreatedBy and Description strings will be used to set the properties of the same name in the TMetafile (see page 14549) object.

C++ Examples:

```
/*
This example shows how to create or augment a metafile using
a metafile canvas object.  This metafile can then be used to
draw on the canvas of another object such as a paintbox or a
printer.
*/
TMetafile *MyMetafile;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
MyMetafile = new TMetafile;
TMetafileCanvas *canvas = new TMetafileCanvas(MyMetafile, 0);
canvas->Brush->Color = clRed;
canvas->Ellipse(0, 0, 100, 200);
// ...
delete canvas;
Form1->Canvas->Draw(0, -50, MyMetafile); //1 red circle
PaintBox1->Canvas->Draw(0, 0, MyMetafile); //1 red circle
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  if (MyMetafile == NULL) return;   // click Button1 first!
  TMetafileCanvas *canvas = new TMetafileCanvas(MyMetafile, 0);
  canvas->Draw(-50, 0, MyMetafile);
  canvas->Brush->Color = clBlue;
  canvas->Ellipse(100, 100, 300, 200);
  // ...
  delete canvas;
  Form1->Canvas->Draw(0, 0, MyMetafile); // 1 red circle and 1 blue circle
  PaintBox1->Canvas->Draw(0, 0, MyMetafile); // 1 red circle and 1 blue circle
```

```
}
```

Delphi Examples:

```
{
This example shows how to create or augment a metafile using
a metafile canvas object.  This metafile can then be used to
draw on the canvas of another object such as a paintbox or a
printer.
}

var
  MyMetafile: TMetafile;

procedure TForm1.Button1Click(Sender: TObject);
begin
  MyMetafile := TMetafile.Create;
  with TMetafileCanvas.Create(MyMetafile, 0) do
  try
    Brush.Color := clRed;
    Ellipse(0, 0, 100, 200);
    //  ...
  finally
    Free;
  end;
  Form1.Canvas.Draw(0, 0, MyMetafile); {1 red circle }
  PaintBox1.Canvas.Draw(0, -50, MyMetafile); {1 red circle }
end;

{
To add to an existing metafile image, create a metafile
canvas and play the source metafile into the metafile canvas:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  if (MyMetafile = nil) then exit;  { click Button1 first! }

  with TMetafileCanvas.Create(MyMetafile, 0) do
  try
    Draw(-50, 0, MyMetafile);
    Brush.Color := clBlue;
    Ellipse(100, 100, 300, 200);
// ...
  finally
    Free;
  end;
  Form1.Canvas.Draw(0, 0, MyMetafile); {1 red circle and 1 blue circle }
  PaintBox1.Canvas.Draw(0, 0, MyMetafile); {1 red circle and 1 blue circle }
end;
```

See Also

CreateWithComment (see page 14568)

73.1.15.3 TMetafileCanvas.CreateWithComment Constructor

Creates an instance of TMetafileCanvas (see page 14564) and acquires a handle to a metafile device context with a comment string.

Pascal

```
constructor CreateWithComment(AMetafile: TMetafile; ReferenceDevice: HDC; const CreatedBy:
String; const Description: String);
```

C++

```
__fastcall TMetafileCanvas(TMetafile AMetafile, HDC ReferenceDevice, const AnsiString
CreatedBy, const AnsiString Description);
```

Description

Call CreateWithComment to create an instantiate a TMetafileCanvas (see page 14564) object for a metafile that will receive the CreatedBy and Description strings. CreateWithComment sets the size of the TMetafile (see page 14549) object from the ReferenceDevice if it does not already have the MMHeight and MMWidth properties set. Create (see page 14567) then creates a metafile device context using the CreatedBy and Description strings as a description, and sets the Handle (see page 14489) property to its handle. All subsequent drawing methods will draw to the metafile device context.

When the TMetafileCanvas (see page 14564) is destroyed, the CreatedBy and Description strings are used to set the properties of the same name in the TMetafile (see page 14549) object.

See Also

Description (see page 14562)

73.1.15.4 TMetafileCanvas.Destroy Destructor

Destroys an instance of the TMetafileCanvas (see page 14564) after transferring the image to the TMetafile (see page 14549) object that was passed to the constructor.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMetafileCanvas();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TMetafileCanvas (see page 14564) reference is not nil, and only then calls Destroy.

Destroy sets the Handle (see page 14489) property of the TMetafile (see page 14549) object that was passed to the constructor before freeing the memory associated with the object.

See Also

Free (see page 22300)

Handle (see page 14563)

73.1.16 TMetafileImage Class

TMetafileImage is the internal representation of the metafile image for a TMetafile (see page 14549) object.

Class Hierarchy

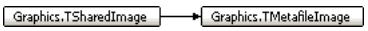

Pascal

```
TMetafileImage = class(TSharedImage);
```

C++

```
class TMetafileImage : public TSharedImage;
```

File

Graphics

Description

TMetafileImage is used for internal implementation only. It represents the internal image of the metafile encapsulated by the TMetafile (see page 14549) object. All of the data and methods introduced in TMetafileImage are private. They contain information about the specific metafile image, including pointers to the standard HENHMETAFILE (see page 14645) (enhanced metafile handle) and HPALETTE (handle to palette) structures defined by Windows.

See Also

TMetafile (see page 14549)

TMetafileCanvas (see page 14564)

73.1.16.1 TMetafileImage Members

The following tables list the members exposed by TMetafileImage.

Protected Methods

	Name	Description
V	FreeHandle (see page 14571)	This is FreeHandle, a member of class TMetafileImage.
	Reference (see page 14604)	This is Reference, a member of class TSharedImage.
	Release (see page 14604)	This is Release, a member of class TSharedImage.

Protected Properties

	Name	Description
R	RefCount (see page 14605)	This is RefCount, a member of class TSharedImage.

Public Methods

	Name	Description
V	Destroy (see page 14570)	Disposes of the instance of a metafile image.

73.1.16.2 TMetafileImage.Destroy Destructor

Disposes of the instance of a metafile image.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMetafileImage();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TMetafileImage (see page 14569) reference is not nil before it calls Destroy.

Destroy frees the metafile handle and palette before calling the inherited Destroy.

See Also

Destroy (see page 22287)

Free (see page 22300)

73.1.16.3 TMetafileImage Methods

The methods of the TMetafileImage class are listed here.

Protected Methods

	Name	Description
	FreeHandle (see page 14571)	This is FreeHandle, a member of class TMetafileImage.

73.1.16.3.1 TMetafileImage.FreeHandle Method

Pascal

```
procedure FreeHandle; override;
```

C++

```
virtual __fastcall FreeHandle();
```

Description

This is FreeHandle, a member of class TMetafileImage.

73.1.17 TPen Class

TPen is used to draw lines or outline shapes on a canvas.

Class Hierarchy

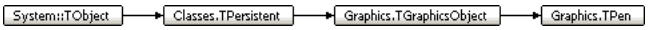

Pascal

```
TPen = class(TGraphicsObject);
```

C++

```
class TPen : public TGraphicsObject;
```

File

Graphics

Description

Use TPen to describe the attributes of a Windows pen when drawing an image to a canvas (TCanvas (see page 14441)). TPen encapsulates the pen properties that are selected into the device context of the canvas.

See Also

TPen

TCanvas (see page 14441)

Using the Main VCL

73.1.17.1 TPen Members

The following tables list the members exposed by TPen.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 14534)	Generates an OnChange (see page 14536) event when the properties of the graphics object change.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetColor (see page 14575)	This is GetColor, a member of class TPen.
	GetHandle (see page 14575)	This is GetHandle, a member of class TPen.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetStyle (see page 14575)	This is GetStyle, a member of class TPen.
	GetWidth (see page 14575)	This is GetWidth, a member of class TPen.
	Lock (see page 14534)	Blocks other execution threads from using the associated canvas until the Unlock (see page 14535) method is called.
	SetColor (see page 14575)	This is SetColor, a member of class TPen.
	SetHandle (see page 14576)	This is SetHandle, a member of class TPen.
	SetMode (see page 14576)	This is SetMode, a member of class TPen.
	SetStyle (see page 14576)	This is SetStyle, a member of class TPen.
	SetWidth (see page 14576)	This is SetWidth, a member of class TPen.
	Unlock (see page 14535)	Releases the lock on execution threads set by the Lock (see page 14534) method.

Public Events

	Name	Description
	OnChange (see page 14536)	Occurs immediately after the graphics object changes.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 14574)	Assigns the properties of the pen from another TPen (see page 14571) object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 14573)	Creates an instance of TPen (see page 14571).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 14573)	Destroys an instance of TPen (see page 14571).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	HandleAllocated (see page 14534)	Indicates whether the TGraphicsObject (see page 14532) object has acquired a handle to the underlying GDI object.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Handle (see page 14578)	Specifies the Windows pen object handle.
	OwnerCriticalSection (see page 14536)	Points to information about the critical section that the canvas which draws with this graphics object uses to lock out other threads.

Published Properties

	Name	Description
	Color (see page 14577)	Determines the color used to draw lines on the canvas.
	Mode (see page 14579)	Determines how the pen draws lines on the canvas.
	Style (see page 14579)	Determines the style in which the pen draws lines.
	Width (see page 14580)	Specifies the width of the pen in pixels.

73.1.17.2 TPen.Create Constructor

Creates an instance of TPen (see page 14571).

Pascal

```
constructor Create;
```

C++

```
__fastcall TPen();
```

Description

Call Create to create a TPen (see page 14571) object for assigning the pen property of a canvas. The pen for the canvas is created automatically in the constructor for the TCanvas (see page 14441) object.

Create allocates memory for the TPen (see page 14571) object and initializes the Mode (see page 14579) property to pmCopy.

See Also

Mode (see page 14579)

Pen (see page 14492)

73.1.17.3 TPen.Destroy Destructor

Destroys an instance of TPen (see page 14571).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPen();
```

Description

Do not call Destroy directly from an application. Instead, call Free. Free verifies that the TPen (see page 14571) reference is not nil, and only then calls Destroy.Destroy frees the resource identified by the Handle (see page 14578) property.

See Also

Free (see page 22300)

73.1.17.4 TPen Methods

The methods of the TPen class are listed here.

Protected Methods

	Name	Description
	GetColor (see page 14575)	This is GetColor, a member of class TPen.
	GetHandle (see page 14575)	This is GetHandle, a member of class TPen.
	GetStyle (see page 14575)	This is GetStyle, a member of class TPen.
	GetWidth (see page 14575)	This is GetWidth, a member of class TPen.
	SetColor (see page 14575)	This is SetColor, a member of class TPen.
	SetHandle (see page 14576)	This is SetHandle, a member of class TPen.
	SetMode (see page 14576)	This is SetMode, a member of class TPen.
	SetStyle (see page 14576)	This is SetStyle, a member of class TPen.
	SetWidth (see page 14576)	This is SetWidth, a member of class TPen.

Public Methods

	Name	Description
	Assign (see page 14574)	Assigns the properties of the pen from another TPen (see page 14571) object.

73.1.17.4.1 TPen.Assign Method

Assigns the properties of the pen from another TPen (see page 14571) object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

If the Source parameter identifies another TPen (see page 14571) object, Assign copies the Color (see page 14577), Width (see page 14580), Style (see page 14579), and Mode (see page 14579) properties of the source pen. Otherwise, the inherited method copies the properties from any object that can write to a TPen (see page 14571) object in its AssignTo method.

See Also

Color (see page 14577)

Mode (see page 14579)

Style (see page 14579)

AssignTo (see page 2931)

Width (see page 14580)

73.1.17.4.2 TPen.GetColor Method

Pascal

```
function GetColor: TColor;
```

C++

```
__fastcall TColor GetColor();
```

Description

This is GetColor, a member of class TPen.

73.1.17.4.3 TPen.GetHandle Method

Pascal

```
function GetHandle: HPen;
```

C++

```
__fastcall HPen GetHandle();
```

Description

This is GetHandle, a member of class TPen.

73.1.17.4.4 TPen.GetStyle Method

Pascal

```
function GetStyle: TPenStyle;
```

C++

```
__fastcall TPenStyle GetStyle();
```

Description

This is GetStyle, a member of class TPen.

73.1.17.4.5 TPen.GetWidth Method

Pascal

```
function GetWidth: Integer;
```

C++

```
__fastcall int GetWidth();
```

Description

This is GetWidth, a member of class TPen.

73.1.17.4.6 TPen.SetColor Method

Pascal

```
procedure SetColor(Value: TColor);
```

C++

```
__fastcall SetColor(TColor Value);
```

Description

This is SetColor, a member of class TPen.

73.1.17.4.7 TPen.SetHandle Method

Pascal

```
procedure SetHandle(Value: HPen);
```

C++

```
__fastcall SetHandle(HPen Value);
```

Description

This is SetHandle, a member of class TPen.

73.1.17.4.8 TPen.SetMode Method

Pascal

```
procedure SetMode(Value: TPenMode);
```

C++

```
__fastcall SetMode(TPenMode Value);
```

Description

This is SetMode, a member of class TPen.

73.1.17.4.9 TPen.SetStyle Method

Pascal

```
procedure SetStyle(Value: TPenStyle);
```

C++

```
__fastcall SetStyle(TPenStyle Value);
```

Description

This is SetStyle, a member of class TPen.

73.1.17.4.10 TPen.SetWidth Method

Pascal

```
procedure SetWidth(Value: Integer);
```

C++

```
__fastcall SetWidth(int Value);
```

Description

This is SetWidth, a member of class TPen.

73.1.17.5 TPen Properties

The properties of the TPen class are listed here.

Public Properties

	Name	Description
	Handle (see page 14578)	Specifies the Windows pen object handle.

Published Properties

	Name	Description
	Color (see page 14577)	Determines the color used to draw lines on the canvas.
	Mode (see page 14579)	Determines how the pen draws lines on the canvas.
	Style (see page 14579)	Determines the style in which the pen draws lines.
	Width (see page 14580)	Specifies the width of the pen in pixels.

73.1.17.5.1 TPen.Color Property

Determines the color used to draw lines on the canvas.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Set Color to change the color used to draw lines or outline shapes. The way the color is used by the pen depends on the Mode (see page 14579) and Style (see page 14579) properties.

C++ Examples:

```
/*
This example draws many rectangles of various sizes and
colors on a form maximized to fill the entire screen. To run
the code, drop a TTimer component on the form and use the
object inspector to create the OnTimer and OnActivate event
handlers.
*/
int x, y;

void __fastcall TForm1::FormActivate(TObject *Sender)
{
  WindowState = wsMaximized;
  Timer1->Interval = 50;
  randomize();
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  x = random(Screen->Width - 10);
  y = random(Screen->Height - 10);
  Canvas->Pen->Color = (Graphics::TColor) random(65535);
  switch (random(5))
  {
    case 0: Canvas->Pen->Style = psSolid; break;
    case 1: Canvas->Pen->Style = psDash; break;
    case 2: Canvas->Pen->Style = psDot; break;
    case 3: Canvas->Pen->Style = psDashDot; break;
```

```
    case 4: Canvas->Pen->Style = psDashDotDot; break;
  }
  Canvas->Rectangle(x, y, x + random(400), y + random(400));
}
```

Delphi Examples:

```
{
This example draws many rectangles of various sizes and
colors on a form maximized to fill the entire screen. To run
the code, drop a TTimer component on the form and use the
object inspector to create the OnTimer and OnActivate event
handlers.
}
var
  X, Y: Integer;
procedure TForm1.FormActivate(Sender: TObject);
begin
  WindowState := wsMaximized;
  Timer1.Interval := 50;
  Randomize;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  X := Random(Screen.Width - 10);
  Y := Random(Screen.Height - 10);
  Canvas.Pen.Color := Random(65535);
  case Random(5) of
    0: Canvas.Pen.Style := psSolid;
    1: Canvas.Pen.Style := psDash;
    2: Canvas.Pen.Style := psDot;
    3: Canvas.Pen.Style := psDashDot;
    4: Canvas.Pen.Style := psDashDotDot;
  end;
  Canvas.Rectangle(X, Y, X + Random(400), Y + Random(400));
end;
```

See Also

Mode (see page 14579)

Style (see page 14579)

73.1.17.5.2 TPen.Handle Property

Specifies the Windows pen object handle.

Pascal

```
property Handle: HPen;
```

C++

```
__property HPen Handle;
```

Description

The Handle property provides access to the Windows GDI object handle. TPen (see page 14571) encapsulates the Windows pen object (HPEN). Use Handle when making a Windows API function call that requires the handle of a pen object. Handle is the Windows handle currently selected into the device context of the canvas.

See Also

Handle (see page 14489)

73.1.17.5.3 TPen.Mode Property

Determines how the pen draws lines on the canvas.

Pascal

```
property Mode: TPenMode;
```

C++

```
__property TPenMode Mode;
```

Description

Set Mode to determine how the color of the pen interacts with the color on the canvas.

See Also

Color (see page 14577)

TCanvas (see page 14441)

73.1.17.5.4 TPen.Style Property

Determines the style in which the pen draws lines.

Pascal

```
property Style: TPenStyle;
```

C++

```
__property TPenStyle Style;
```

Description

Use Style to draw a dotted or dashed line, or to omit the line that appears as a frame around shapes.

Note: Only the psInsideFrame style will produce a dithered color to match a Color (see page 14577) property that is not in the color table. All others choose the closest color from the Windows color table.

Note: Dotted or dashed pen styles are not available when the Width (see page 14580) property is not 1.

C++ Examples:

```
/*
This example draws many rectangles of various sizes and
colors on a form maximized to fill the entire screen. To run
the code, drop a TTimer component on the form and use the
object inspector to create the OnTimer and OnActivate event
handlers.
*/
int x, y;

void __fastcall TForm1::FormActivate(TObject *Sender)
{
  WindowState = wsMaximized;
  Timer1->Interval = 50;
  randomize();
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
```

```
    x = random(Screen->Width - 10);
    y = random(Screen->Height - 10);
    Canvas->Pen->Color = (Graphics::TColor) random(65535);
    switch (random(5))
    {
      case 0: Canvas->Pen->Style = psSolid; break;
      case 1: Canvas->Pen->Style = psDash; break;
      case 2: Canvas->Pen->Style = psDot; break;
      case 3: Canvas->Pen->Style = psDashDot; break;
      case 4: Canvas->Pen->Style = psDashDotDot; break;
    }
    Canvas->Rectangle(x, y, x + random(400), y + random(400));
  }
```

Delphi Examples:

```
{
This example draws many rectangles of various sizes and
colors on a form maximized to fill the entire screen. To run
the code, drop a TTimer component on the form and use the
object inspector to create the OnTimer and OnActivate event
handlers.
}
var
  X, Y: Integer;
procedure TForm1.FormActivate(Sender: TObject);
begin
  WindowState := wsMaximized;
  Timer1.Interval := 50;
  Randomize;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  X := Random(Screen.Width - 10);
  Y := Random(Screen.Height - 10);
  Canvas.Pen.Color := Random(65535);
  case Random(5) of
    0: Canvas.Pen.Style := psSolid;
    1: Canvas.Pen.Style := psDash;
    2: Canvas.Pen.Style := psDot;
    3: Canvas.Pen.Style := psDashDot;
    4: Canvas.Pen.Style := psDashDotDot;
  end;
  Canvas.Rectangle(X, Y, X + Random(400), Y + Random(400));
end;
```

See Also

Color (see page 14577)

Width (see page 14580)

73.1.17.5.5 TPen.Width Property

Specifies the width of the pen in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use Width to give the line greater weight. If you attempt to set Width to a value less than 0, the new value is ignored.

Note: The value of Width influences which values of Style (see page 14579) are valid.

C++ Examples:

```
/*
This example draws many rounded rectangles of various sizes
and colors on a form maximized to fill the entire screen:
*/
int x, y;

void __fastcall TForm1::FormActivate(TObject *Sender)
{
  WindowState = wsMaximized;
  Timer1->Interval = 50;
  randomize();
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  x = random(Screen->Width - 10);
  y = random(Screen->Height - 10);
  Canvas->Pen->Color = (Graphics::TColor) random(65535);
  Canvas->Pen->Width = random(7);
  int Dx = random(400);
  int Dy = random(400);
  Canvas->RoundRect(x, y, x + Dx, y + Dy, Dx/2, Dy/2);
}
```

Delphi Examples:

```
{
This example draws many rounded rectangles of various sizes
and colors on a form maximized to fill the entire screen:
}
var
  X, Y, DX, DY: Integer;
procedure TForm1.FormActivate(Sender: TObject);
begin
  WindowState := wsMaximized;
  Timer1.Interval := 50;
  Randomize;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  X := Random(Screen.Width - 10);
  Y := Random(Screen.Height - 10);
  Canvas.Pen.Color := Random(65535);
  Canvas.Pen.Width := Random(7);
  DX := Random(400);
  DY := Random(400);
  Canvas.RoundRect(X, Y, X + DX, Y + DY, DX div 2, DY div 2);
end;
```

See Also

Style (see page 14579)

73.1.18 TPenRecall Class

TPenRecall saves and restores the properties of a TPen (see page 14571) object.

Class Hierarchy

Pascal

```
TPenRecall = class(TRecall);
```

C++

```
class TPenRecall : public TRecall;
```

File

Graphics

Description

Use TPenRecall to store the current state of a TPen (see page 14571) object. You are then free to change the current state of the pen object. To restore the pen object to its original state, all that is necessary is to destroy the TPenRecall object and the pen object is automatically restored to the saved property values.

You can update the TPenRecall instance to reflect the current properties of the TPen (see page 14571) object by calling the Store method. You can prevent the TPenRecall destructor from updating the TPen (see page 14571) object by calling the Forget method.

See Also

TRecall (see page 2964)

TPen (see page 14571)

Using Pens

73.1.18.1 TPenRecall Members

The following tables list the members exposed by TPenRecall.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 14583)	Instantiates a TPenRecall (see page 14582) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2966)	Destroys the TRecall (see page 2964) instance, assigning the stored properties to the reference object.

V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Forget (see page 2967)	Discards the saved property values and prevents the TRecall (see page 2964) object from updating the reference object.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Store (see page 2967)	Saves the current property settings of the reference object.

Public Properties

	Name	Description
R	Reference (see page 2968)	Indicates the reference object that is updated when you destroy the TRecall (see page 2964) instance.

73.1.18.2 TPenRecall.Create Constructor

Instantiates a TPenRecall (see page 14582) object.

Pascal

```
constructor Create(APen: TPen);
```

C++

```
__fastcall TPenRecall(TPen APen);
```

Description

Call Create to instantiate a pen recall object at runtime. Create saves the current properties of the TPen (see page 14571) object specified by APen, which becomes the value of the Reference property.

Subsequent calls to the Store method take snapshots of the properties of APen and save them, replacing the values saved by this constructor. When you destroy the TPenRecall (see page 14582) instance, the most recently-saved properties are assigned to APen, restoring it to a previous state.

See Also

Destroy (see page 2966)

Reference (see page 2968)

Store (see page 2967)

73.1.19 TPicture Class

TPicture contains a bitmap, icon, metafile graphic, or user-defined graphic.

Class Hierarchy

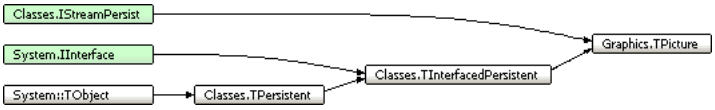

Pascal

```
TPicture = class(TInterfacedPersistent, IStreamPersist);
```

C++

```
class TPicture : public TInterfacedPersistent, public IStreamPersist;
```

File

Graphics

Description

TPicture is a TGraphic (see page 14516) container, used to hold a graphic, the type of which is specified in the Graphic (see page 14600) property. It is used in place of a TGraphic (see page 14516) if the graphic can be of any TGraphic (see page 14516) class. LoadFromFile (see page 14591) and SaveToFile (see page 14595) are polymorphic. For example, if the TPicture is holding an Icon (see page 14601), it is valid to LoadFromFile (see page 14591) a bitmap file, where the class TIcon (see page 14537) can only read .ICO files.

If the TPicture contains a bitmap graphic, the Bitmap (see page 14598) property specifies the graphic. If the TPicture contains an icon graphic, the Icon (see page 14601) property specifies the graphic. If the TPicture contains a metafile graphic, the Metafile (see page 14601) property specifies the graphic.

The properties of TPicture indicate the type of graphic that the picture object contains, and its size. The methods of TPicture are used to load, save, and manipulate graphics.

To load or save a picture to the Clipboard, use the Assign (see page 14587) method of a TClipboard object.

To draw a picture on a canvas, call the Draw or StretchDraw methods of a TCanvas (see page 14441) object, passing the Graphic (see page 14600) property of a TPicture as a parameter.

See Also

TCanvas (see page 14441)

TClipboard (see page 3328)

73.1.19.1 TPicture Members

The following tables list the members exposed by TPicture.

Protected Methods

	Name	Description
	_AddRef (see page 2853)	Increments the reference count for the interfaced object's interface.
	_Release (see page 2854)	Decrements the reference count for the interfaced object's interface.
	AssignTo (see page 14589)	Copies the properties of an object to a destination object.

	Changed (see page 14590)	This is Changed, a member of class TPicture.
	DefineProperties (see page 14590)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.
	LoadFromStream (see page 14591)	Reads the picture from a Stream.
	Progress (see page 14592)	This is Progress, a member of class TPicture.
	SaveToStream (see page 14597)	Saves the picture to a stream.

Public Events

	Name	Description
	OnChange (see page 14602)	Occurs when the specific graphics item encapsulated by the picture object changes.
	OnProgress (see page 14603)	Occurs periodically during slow operations that affect the graphic.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 2854)	Obtains a reference to the Owner's interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 14587)	Copies one object to another by copying the contents of that object to the other.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 14586)	Creates a TPicture (see page 14584) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 14586)	Destroys an instance of a picture object.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LoadFromClipboardFormat (see page 14590)	Reads the picture from the handle provided in the given Clipboard format.
	LoadFromFile (see page 14591)	Reads the file specified in Filename and loads the data into the TPicture (see page 14584) object.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 2854)	Returns a reference to a specified interface if the object supports that interface.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	RegisterClipboardFormat (see page 14592)	Registers a new TGraphic (see page 14516) class for use in the LoadFromClipboardFormat (see page 14590) method.
	RegisterFileFormat (see page 14594)	Registers a new TGraphic (see page 14516) class for use in LoadFromFile (see page 14591).
	RegisterFileFormatRes (see page 14594)	Registers a new TGraphic (see page 14516) class for use in the LoadFromFile (see page 14591) method.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToClipboardFormat (see page 14595)	Allocates a global handle and writes the picture in its native Clipboard format (CF_BITMAP for bitmaps, CF_METAFILE for metafiles, and so on).
	SaveToFile (see page 14595)	Writes the picture to disk.
	SupportsClipboardFormat (see page 14597)	Indicates if the given Clipboard format is supported by the LoadFromClipboardFormat (see page 14590) method.
	UnregisterGraphicClass (see page 14597)	This is UnregisterGraphicClass, a member of class TPicture.

Public Properties

	Name	Description
	Bitmap (see page 14598)	Specifies the contents of the picture object as a bitmap graphic (.BMP file format).
	Graphic (see page 14600)	Specifies the graphic that the picture contains.
R	Height (see page 14600)	Specifies the vertical size (in pixels) of the graphic.
	Icon (see page 14601)	Specifies the contents of the TPicture (see page 14584) object as an icon graphic (.ICO file format).
	Metafile (see page 14601)	Specifies the contents of the picture object as an Enhanced Windows metafile graphic (.EMF file format).
	PictureAdapter (see page 14601)	Represents an OLE interface for the picture.
R	Width (see page 14602)	Specifies the horizontal size (in pixels) of the picture.

73.1.19.2 TPicture.Create Constructor

Creates a TPicture (see page 14584) object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TPicture();
```

Description

Call Create to create an instance of TPicture (see page 14584) at runtime. Create allocates the memory for the picture object and initializes its internal formats. After creating the TPicture (see page 14584) object, use the LoadFromClipboardFormat (see page 14590) or LoadFromFile (see page 14591) method to assign a graphical image to the picture, so that it is available to the application.

See Also

LoadFromClipboardFormat (see page 14590)

LoadFromFile (see page 14591)

73.1.19.3 TPicture.Destroy Destructor

Destroys an instance of a picture object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPicture();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TPicture (see page 14584) reference is not nil before it calls Destroy.

Destroy frees the graphic image it contains before calling the inherited Destroy.

See Also

Free (see page 22300)

Destroy (see page 22287)

73.1.19.4 TPicture Methods

The methods of the TPicture class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 14589)	Copies the properties of an object to a destination object.
	Changed (see page 14590)	This is Changed, a member of class TPicture.
	DefineProperties (see page 14590)	Provides an interface for a method that reads and writes otherwise unpublished data.
	LoadFromStream (see page 14591)	Reads the picture from a Stream.
	Progress (see page 14592)	This is Progress, a member of class TPicture.
	SaveToStream (see page 14597)	Saves the picture to a stream.

Public Methods

	Name	Description
	Assign (see page 14587)	Copies one object to another by copying the contents of that object to the other.
	LoadFromClipboardFormat (see page 14590)	Reads the picture from the handle provided in the given Clipboard format.
	LoadFromFile (see page 14591)	Reads the file specified in Filename and loads the data into the TPicture (see page 14584) object.
	RegisterClipboardFormat (see page 14592)	Registers a new TGraphic (see page 14516) class for use in the LoadFromClipboardFormat (see page 14590) method.
	RegisterFileFormat (see page 14594)	Registers a new TGraphic (see page 14516) class for use in LoadFromFile (see page 14591).
	RegisterFileFormatRes (see page 14594)	Registers a new TGraphic (see page 14516) class for use in the LoadFromFile (see page 14591) method.
	SaveToClipboardFormat (see page 14595)	Allocates a global handle and writes the picture in its native Clipboard format (CF_BITMAP for bitmaps, CF_METAFILE for metafiles, and so on).
	SaveToFile (see page 14595)	Writes the picture to disk.
	SupportsClipboardFormat (see page 14597)	Indicates if the given Clipboard format is supported by the LoadFromClipboardFormat (see page 14590) method.
	UnregisterGraphicClass (see page 14597)	This is UnregisterGraphicClass, a member of class TPicture.

73.1.19.4.1 TPicture.Assign Method

Copies one object to another by copying the contents of that object to the other.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

When Source is a object type that is valid for the Graphic (see page 14600) property, Assign makes that graphic the value of the Graphic (see page 14600) property.

The actions performed by Assign depend on the actual types of the TPicture (see page 14584) Graphic (see page 14600) property and Source. For example, if the Graphic (see page 14600) property and Source are bitmaps (TBitmap (see page 14396)), the bitmap contained in Source is copied into the Graphic (see page 14600) property. Similar conversions are valid, for example, for TIcon (see page 14537) or TMetafile (see page 14549).

If the Source parameter is not a valid object for the Graphic (see page 14600) property, Assign calls the inherited method so that the picture can be copied from any object with which it is compatible.

C++ Examples:

```
/*
This example uses an image, a button, and a shape component
on a form. When the user clicks the button, an image of the
form is stored in the FormImage variable and copied to the
Clipboard. The image of the form is then copied back to
the image component, producing an interesting result,
especially if the button is clicked multiple times.  Add
ExtCtrls, StdCtrls, and Clipbrd to the uses clause.
}
*/

#include <Clipbrd.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *FormImage = GetFormImage();
  try
  {
    Clipboard()->Assign(FormImage);
    Image1->Picture->Assign(Clipboard());
  }
  __finally
  {
    delete FormImage;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Shape1->Shape = stEllipse;
  Shape1->Brush->Color = clLime;
  Image1->Stretch = true;
}
```

Delphi Examples:

```
{
This example uses an image, a button, and a shape component
on a form. When the user clicks the button, an image of the
form is stored in the FormImage variable and copied to the
Clipboard. The image of the form is then copied back to
the image component, producing an interesting result,
especially if the button is clicked multiple times.  Add
ExtCtrls, StdCtrls, and Clipbrd to the uses clause.
}
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
  FormImage: TBitmap;
begin
  FormImage := GetFormImage;
  try
    Clipboard.Assign(FormImage);
    Image1.Picture.Assign(Clipboard);
  finally
    FormImage.Free;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Shape1.Shape := stEllipse;
  Shape1.Brush.Color := clLime;
  Image1.Stretch := True;
end;
```

See Also

Graphic (see page 14600)

Assign (see page 2930)

73.1.19.4.2 TPicture.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign (see page 2930)

EConvertError (see page 22724)

73.1.19.4.3 TPicture.Changed Method

Pascal

```
procedure Changed(Sender: TObject); dynamic;
```

C++

```
__fastcall Changed(TObject * Sender);
```

Description

This is Changed, a member of class TPicture.

73.1.19.4.4 TPicture.DefineProperties Method

Provides an interface for a method that reads and writes otherwise unpublished data.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

Descendants of TPersistent override DefineProperties to designate a method for storing the object's unpublished data to a stream such as a form file. By default, writing an object to a stream writes the values of all its published properties, and reading the object in reads those values and assigns them to the properties. Objects can also specify methods that read and write data other than published properties by overriding the DefineProperties method.

When overriding DefineProperties, consider including some or all of the following:

A call to the inherited method

Calls to the filer object's DefineProperty method

Calls to the filer object's DefineBinaryProperty method

DefineProperties is virtual, so descendant classes can override it as necessary but are not required to do so.

See Also

DefineProperties (see page 2771)

DefineBinaryProperty (see page 2837)

DefineProperty (see page 2838)

73.1.19.4.5 TPicture.LoadFromClipboardFormat Method

Reads the picture from the handle provided in the given Clipboard format.

Pascal

```
procedure LoadFromClipboardFormat(AFormat: Word; AData: THandle; APalette: HPALETTE);
```

C++

```
__fastcall LoadFromClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette);
```

Description

Use LoadFromClipboardFormat to read in a graphic from the Clipboard. If the format is not supported, an EInvalidGraphic (see

page 14395) exception is raised.

See Also

EInvalidGraphic (see page 14395)

RegisterClipboardFormat (see page 14592)

TClipboard (see page 3328)

73.1.19.4.6 TPicture.LoadFromFile Method

Reads the file specified in Filename and loads the data into the TPicture (see page 14584) object.

Pascal

```
procedure LoadFromFile(const Filename: string);
```

C++

```
__fastcall LoadFromFile(const AnsiString Filename);
```

Description

Use LoadFromFile to read a picture from disk. The TGraphic (see page 14516) class created is determined by the file extension of the file. If the file extension is not recognized an EInvalidGraphic (see page 14395) exception is raised.

See Also

EInvalidGraphic (see page 14395)

RegisterFileFormat (see page 14594)

TGraphic (see page 14516)

73.1.19.4.7 TPicture.LoadFromStream Method

Reads the picture from a Stream.

Pascal

```
procedure LoadFromStream(Stream: TStream);
```

C++

```
__fastcall LoadFromStream(TStream Stream);
```

Description

Use LoadFromStream to read a picture from a TStream object. The TGraphic (see page 14516) class created is determined by the type of data contained in the stream.

The data in the stream should be previously written by a call to the SaveToStream (see page 14597) method of another TPicture (see page 14584) or of a TGraphic (see page 14516) descendant.

See Also

EInvalidGraphic (see page 14395)

LoadFromMimeSource

LoadFromFile (see page 14591)

SaveToStream (see page 14597)

SaveToStream (see page 14526)

73.1.19.4.8 TPicture.Progress Method

Pascal

```
procedure Progress(Sender: TObject; Stage: TProgressStage; PercentDone: Byte; RedrawNow:
Boolean; const R: TRect; const Msg: string); dynamic;
```

C++

```
__fastcall Progress(TObject * Sender, TProgressStage Stage, Byte PercentDone, Boolean
RedrawNow, const TRect R, const AnsiString Msg);
```

Description

This is Progress, a member of class TPicture.

73.1.19.4.9 TPicture.RegisterClipboardFormat Method

Registers a new TGraphic (see page 14516) class for use in the LoadFromClipboardFormat (see page 14590) method.

Pascal

```
class procedure RegisterClipboardFormat(AFormat: Word; AGraphicClass: TGraphicClass);
```

C++

```
__fastcall RegisterClipboardFormat(Word AFormat, TGraphicClass AGraphicClass);
```

Description

Use RegisterClipboardFormat register a new graphic format with TPicture (see page 14584) so that it can be used with the LoadFromClipboardFormat (see page 14590) method.

C++ Examples:

```
/*
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
*/
#include <Clipbrd.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClipboard *cb = Clipboard();
  if (cb->HasFormat(CF_BITMAP))
  {
    Graphics::TBitmap *Bitmap = new Graphics::TBitmap();
    try
    {
      Bitmap->LoadFromClipboardFormat(CF_BITMAP, cb->GetAsHandle(CF_BITMAP), 0);
      Canvas->Draw(5,5,Bitmap);
    }
    catch (...)
    {
      MessageBeep(0);
      ShowMessage("Error reading image from clipboard!");
    }
    delete Bitmap;
    cb->Clear();
  }
  else
  {
    MessageBeep(0);
    ShowMessage("Cannot find image in clipboard!");
```

```
  }
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  unsigned int DataHandle;
  HPALETTE APalette;
  unsigned short MyFormat;
  Graphics::TBitmap *Bitmap = new Graphics::TBitmap();
  try
  {
    Bitmap->LoadFromFile("C:\\Program Files\\Common Files\\CodeGear
Shared\\Images\\Splash\\256Color\\FACTORY.BMP");
    // generate a clipboard format, with data and palette
    Bitmap->SaveToClipboardFormat(
      MyFormat,
      DataHandle,
      APalette);
    // save the data to the clipboard using that format and
    // the generated data
    Clipboard()->SetAsHandle(MyFormat,DataHandle);
  }
  catch (...)
  {
     ShowMessage("Failed to copy image to clipboard!");
  }
  delete Bitmap;
}
```

Delphi Examples:

```
{
The following code draws a bitmap image from the Clipboard
when a button is pressed.  The SaveToClipboard button must
be selected before the LoadFromClipboard button in order to
place the icon on the clipboard.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitmap;
  s : string;
  Icon: TIcon;
begin
 Bitmap := TBitMap.Create;
 try
   Image1.Picture.RegisterClipboardFormat(cf_BitMap,TIcon);
   Bitmap.LoadFromClipBoardFormat(
     cf_BitMap,ClipBoard.GetAsHandle(cf_Bitmap),0);
   Canvas.draw(0,0,Bitmap);
 finally
   Bitmap.free;
   Clipboard.Clear;
 end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  MyFormat : Word;
  Bitmap : TBitMap;
  AData : THandle;
  APalette : HPALETTE;
begin
  Bitmap := TBitmap.Create;
  try
    Bitmap.LoadFromFile('c:\Program Files\Common Files\CodeGear
Shared\Images\Splash\256color\factory.bmp');
```

```
      Bitmap.SaveToClipBoardFormat(
        MyFormat,
        AData,
        APalette);
      ClipBoard.SetAsHandle(MyFormat,AData);
    finally
      Bitmap.Free;
    end;
  end;
```

See Also

LoadFromClipboardFormat (see page 14590)

TClipboard (see page 3328)

UnRegisterGraphicClass (see page 14597)

73.1.19.4.10 TPicture.RegisterFileFormat Method

Registers a new TGraphic (see page 14516) class for use in LoadFromFile (see page 14591).

Pascal

```
class procedure RegisterFileFormat(const AExtension: string; const ADescription: string;
AGraphicClass: TGraphicClass);
```

C++

```
__fastcall RegisterFileFormat(const AnsiString AExtension, const AnsiString ADescription,
TGraphicClass AGraphicClass);
```

Description

Use RegisterFileFormat to register a graphic file format with TPicture (see page 14584) so that it can be used with a Open or Save dialog box.

The AExtension parameter specifies the three-character system file extension to associate with the graphic class (for example, "bmp" is associated with TBitmap (see page 14396)). The ADescription parameter specifies the description of the graphic to appear in the drop down list of the dialog box (for example, "Bitmaps" is the description of TBitmap (see page 14396)). The AGraphicClass parameter registers the new graphic class to associate with the file format.

See Also

LoadFromFile (see page 14591)

UnRegisterGraphicClass (see page 14597)

73.1.19.4.11 TPicture.RegisterFileFormatRes Method

Registers a new TGraphic (see page 14516) class for use in the LoadFromFile (see page 14591) method.

Pascal

```
class procedure RegisterFileFormatRes(const AExtension: String; ADescriptionResID: Integer;
AGraphicClass: TGraphicClass);
```

C++

```
__fastcall RegisterFileFormatRes(const AnsiString AExtension, int ADescriptionResID,
TGraphicClass AGraphicClass);
```

Description

Use RegisterFileFormatRes by specifying a string resource.

The AExtension parameter specifies the three-character system file extension to associate with the graphic class (for example, "bmp" is associated with TBitmap (see page 14396)).

The ADescriptionResID parameter specifies the resource ID for a description of the graphic, which then appears in the drop down list of the dialog box (for example, "Bitmaps" is the description of TBitmap (see page 14396)).

The AGraphicClass parameter registers the new graphic class to associate with the file format.

See Also

LoadFromFile (see page 14591)

UnRegisterGraphicClass (see page 14597)

73.1.19.4.12 TPicture.SaveToClipboardFormat Method

Allocates a global handle and writes the picture in its native Clipboard format (CF_BITMAP for bitmaps, CF_METAFILE for metafiles, and so on).

Pascal

```
procedure SaveToClipboardFormat(var AFormat: Word; var AData: THandle; var APalette: HPALETTE);
```

C++

```
__fastcall SaveToClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette);
```

Description

Use SaveToClipboardFormat to copy the picture to a Clipboard format. The resulting values can then be copied to the Windows clipboard using the clipboard's SetAsHandle method.

The picture's palette is returned in the APalette parameter, the format in the AFormat parameter, and a handle to the data in the AData parameter. Before the picture can be saved, an application must have registered the format using the RegisterClipboardFormat (see page 14592) method.

See Also

Formats (see page 3348)

RegisterClipboardFormat (see page 14592)

73.1.19.4.13 TPicture.SaveToFile Method

Writes the picture to disk.

Pascal

```
procedure SaveToFile(const Filename: string);
```

C++

```
__fastcall SaveToFile(const AnsiString Filename);
```

Description

Use SaveToFile to save a TPicture (see page 14584) object to the file specified in Filename.

The saved picture can be loaded to another TPicture (see page 14584) or to an appropriate TGraphic (see page 14516) descendant by calling its LoadFromFile (see page 14591) method.

C++ Examples:

```
/*
This example converts a specified icon to a bitmap. To run
```

```
this example, add an image, a button, and an Open dialog to
a form. Name the button ConvertIcon2Bitmap, and add the
following code as its OnClick event handler.
*/
void __fastcall TForm1::ConvertIcon2BitmapClick(TObject *Sender)
{
  OpenDialog1->DefaultExt = ".ICO";
  OpenDialog1->Filter = "icons (*.ico)|*.ICO";
  OpenDialog1->Options << ofOverwritePrompt << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TIcon *pi = new TIcon();
    try
    {
      AnsiString as;
      pi->LoadFromFile(OpenDialog1->FileName);
      as = ChangeFileExt(OpenDialog1->FileName,".BMP");
      Image1->Width = pi->Width;
      Image1->Height = pi->Height;
      Image1->Canvas->Draw(0,0,pi);
      Image1->Picture->SaveToFile(as);
      ShowMessage(OpenDialog1->FileName + " Saved to " + as);
    }
    __finally
    {
      delete pi;
    }
  }
}
```

Delphi Examples:

```
{
This example converts a specified icon to a bitmap. To run
this example, add an image, a button, and an Open dialog to
a form. Name the button ConvertIcon2Bitmap, and add the
following code as its OnClick event handler.
}
procedure TForm1.ConvertIcon2BitmapClick(Sender: TObject);
var
  s : string;
  Icon: TIcon;
begin
  OpenDialog1.DefaultExt := '.ICO';
  OpenDialog1.Filter := 'icons (*.ico)|*.ICO';
  OpenDialog1.Options := [ofOverwritePrompt, ofFileMustExist, ofHideReadOnly ];
  if OpenDialog1.Execute then
  begin
    Icon := TIcon.Create;
    try
      Icon.Loadfromfile(OpenDialog1.FileName);
      s:= ChangeFileExt(OpenDialog1.FileName,'.BMP');
      Image1.Width := Icon.Width;
      Image1.Height := Icon.Height;
      Image1.Canvas.Draw(0,0,Icon);
      Image1.Picture.SaveToFile(s);
      ShowMessage(OpenDialog1.FileName + ' Saved to ' + s);
    finally
      Icon.Free;
    end;
  end;
end;
```

See Also

LoadFromFile (see page 14591)

LoadFromFile (see page 14523)

73.1.19.4.14 TPicture.SaveToStream Method

Saves the picture to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream);
```

C++

```
__fastcall SaveToStream(TStream Stream);
```

Description

Use SaveToStream to save a picture to the TStream object specified by the Stream parameter.

The saved picture can be loaded to another TPicture (see page 14584) or to an appropriate TGraphic (see page 14516) descendant by calling its LoadFromStream (see page 14591) method.

See Also

LoadFromStream (see page 14591)

LoadFromStream (see page 14524)

73.1.19.4.15 TPicture.SupportsClipboardFormat Method

Indicates if the given Clipboard format is supported by the LoadFromClipboardFormat (see page 14590) method.

Pascal

```
class function SupportsClipboardFormat(AFormat: Word): Boolean;
```

C++

```
__fastcall Boolean SupportsClipboardFormat(Word AFormat);
```

Description

If the LoadFromClipboardFormat (see page 14590) method supports the Clipboard format specified as the value of AFormat, SupportsClipboardFormat returns true. If the format is not supported, the method returns false.

See Also

LoadFromClipboardFormat (see page 14590)

73.1.19.4.16 TPicture.UnregisterGraphicClass Method

Pascal

```
class procedure UnregisterGraphicClass(AClass: TGraphicClass);
```

C++

```
__fastcall UnregisterGraphicClass(TGraphicClass AClass);
```

Description

This is UnregisterGraphicClass, a member of class TPicture.

73.1.19.5 TPicture Properties

The properties of the TPicture class are listed here.

Public Properties

	Name	Description
	Bitmap (see page 14598)	Specifies the contents of the picture object as a bitmap graphic (.BMP file format).
	Graphic (see page 14600)	Specifies the graphic that the picture contains.
R	Height (see page 14600)	Specifies the vertical size (in pixels) of the graphic.
	Icon (see page 14601)	Specifies the contents of the TPicture (see page 14584) object as an icon graphic (.ICO file format).
	Metafile (see page 14601)	Specifies the contents of the picture object as an Enhanced Windows metafile graphic (.EMF file format).
	PictureAdapter (see page 14601)	Represents an OLE interface for the picture.
R	Width (see page 14602)	Specifies the horizontal size (in pixels) of the picture.

73.1.19.5.1 TPicture.Bitmap Property

Specifies the contents of the picture object as a bitmap graphic (.BMP file format).

Pascal

```
property Bitmap: TBitmap;
```

C++

```
__property TBitmap Bitmap;
```

Description

Use Bitmap to reference the picture object when it contains a bitmap. If Bitmap is referenced when the picture contains a Metafile (see page 14601) or Icon (see page 14601) graphic, the graphic won't be converted. Instead, the original contents of the picture are discarded and Bitmap returns a new, blank bitmap.

Note: When assigning the Bitmap property, TPicture (see page 14584) assigns the properties of a another TBitmap (see page 14396) object. It does not take ownership of the specified value.

C++ Examples:

```
/*
The following example is taken from the custom draw demo. It
shows how the OnCustomDraw event handler draws the
background for the tree view before the items and lines are
drawn.
*/
void __fastcall TCustomDrawForm::TVCustomDraw(TCustomTreeView *Sender,
      const TRect &ARect, bool &DefaultDraw)
{
/*
This event should be used to draw any background colors or
images.  ARect represents the entire client area of the TreeView.
Use the TreeView's canvas to do the drawing.
Note that drawing a background bitmap is not really
supported by CustomDraw, so scrolling can get messy. Best to
subclass the TreeView and handle scrolling messages.
*/
  if (None1 != NULL)
  {
    if (None1->Checked) //no picture
```

```
      {
        TV->Canvas->Brush->Color = BkgColorDialog->Color;
        TV->Canvas->Brush->Style = FBrushStyle;
        TV->Canvas->FillRect(ARect);
      }
      else if (Tile1->Checked) //tile bitmap
      {
        TV->Canvas->Brush->Bitmap = Image1->Picture->Bitmap;
        TV->Canvas->FillRect(ARect);
      }
      else //Stretch across the canvas.
        TV->Canvas->StretchDraw(ARect, Image1->Picture->Bitmap);
    };
    DefaultDraw = FDefaultDraw;
    // Setting DefaultDraw to false here prevents all calls to
    // OnCustomDrawItem.
  }
```

Delphi Examples:

```
{
The following example is taken from the custom draw demo. It
shows how the OnCustomDraw event handler draws the
background for the tree view before the items and lines are
drawn.
}
procedure TCustomDrawForm.TVCustomDraw(Sender: TCustomTreeView; const ARect: TRect;
  var DefaultDraw: Boolean);
begin
{
This event should be used to draw any background colors or
images.  ARect represents the entire client area of the TreeView.
Use the TreeView's canvas to do the drawing.
Note that drawing a background bitmap is not really
supported by CustomDraw, so scrolling can get messy. Best to
subclass the TreeView and handle scrolling messages.
}
  with TV.Canvas do
  begin
    if None1 <> nil then
    begin
      if None1.Checked then //no picture
      begin
        Brush.Color := BkgColorDialog.Color;
        Brush.Style := FBrushStyle;
        FillRect(ARect);
      end else
        if Tile1.Checked then //tile bitmap
        begin
          Brush.Bitmap := Image1.Picture.Bitmap;
            FillRect(ARect);
          end else //Stretch across the canvas.
              StretchDraw(ARect, Image1.Picture.Bitmap);
    end;
  end;
  DefaultDraw := FDefaultDraw;
  // Setting DefaultDraw to false here prevents all calls to
  // OnCustomDrawItem.
end;
```

See Also

Graphic (see page 14600)

73.1.19.5.2 TPicture.Graphic Property

Specifies the graphic that the picture contains.

Pascal

```
property Graphic: TGraphic;
```

C++

```
__property TGraphic Graphic;
```

Description

Use Graphic to specify what graphic the TPicture (see page 14584) object contains. The graphic can be a bitmap, icon, metafile, or user-defined graphic class.

Note: When assigning the Graphic property, TPicture (see page 14584) assigns the properties of a another TGraphic (see page 14516) object. It does not take ownership of the specified object.

C++ Examples:

```
/*
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PaintBox1->Canvas->Draw(0,0, Image1->Picture->Graphic);
}
```

Delphi Examples:

```
{
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  PaintBox1.Canvas.Draw(0,0, Image1.Picture.Graphic);
end;
```

See Also

Bitmap (see page 14598)

Icon (see page 14601)

Metafile (see page 14601)

73.1.19.5.3 TPicture.Height Property

Specifies the vertical size (in pixels) of the graphic.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use Height to find the height of the graphic image contained in the picture object. Height is the native, unstretched, height of the picture.

See Also

Width (see page 14602)

73.1.19.5.4 TPicture.Icon Property

Specifies the contents of the TPicture (see page 14584) object as an icon graphic (.ICO file format).

Pascal

```
property Icon: TIcon;
```

C++

```
__property TIcon Icon;
```

Description

If Icon is referenced when the TPicture (see page 14584) contains a Bitmap (see page 14598) or Metafile (see page 14601) graphic, the graphic won't be converted. Instead, the original contents of the TPicture (see page 14584) are discarded and Icon returns a new, blank icon.

Note: When assigning the Icon property, TPicture (see page 14584) assigns the properties of a another Icon object. It does not take ownership of the specified value.

See Also

Graphic (see page 14600)

73.1.19.5.5 TPicture.Metafile Property

Specifies the contents of the picture object as an Enhanced Windows metafile graphic (.EMF file format).

Pascal

```
property Metafile: TMetafile;
```

C++

```
__property TMetafile Metafile;
```

Description

If Metafile is referenced when the TPicture (see page 14584) contains a Bitmap (see page 14598) or Icon (see page 14601) graphic, the graphic won't be converted. Instead, the original contents of the TPicture (see page 14584) are discarded and Metafile returns a new, blank metafile.

Note: When assigning the Metafile property, TPicture (see page 14584) assigns the properties of a another TMetafile (see page 14549) object. It does not take ownership of the specified value.

See Also

Graphic (see page 14600)

73.1.19.5.6 TPicture.PictureAdapter Property

Represents an OLE interface for the picture.

Pascal

```
property PictureAdapter: IChangeNotifier;
```

C++

```
__property IChangeNotifier PictureAdapter;
```

Description

PictureAdapter is for internal use only.

See Also

TPictureAdapter (see page 1599)

73.1.19.5.7 TPicture.Width Property

Specifies the horizontal size (in pixels) of the picture.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use Width to find the width of a graphic image. Width contains the native, unstretched, width of the picture.

See Also

Height (see page 14600)

73.1.19.6 TPicture Events

The events of the TPicture class are listed here.

Public Events

	Name	Description
	OnChange (see page 14602)	Occurs when the specific graphics item encapsulated by the picture object changes.
	OnProgress (see page 14603)	Occurs periodically during slow operations that affect the graphic.

73.1.19.6.1 TPicture.OnChange Event

Occurs when the specific graphics item encapsulated by the picture object changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Use OnChange to write an event handler to process an action when the graphic is modified.

73.1.19.6.2 TPicture.OnProgress Event

Occurs periodically during slow operations that affect the graphic.

Pascal

```
property OnProgress: TProgressEvent;
```

C++

```
__property TProgressEvent OnProgress;
```

Description

OnProgress is generated by the particular graphic that the picture object contains. Whether OnProgress occurs depends upon the type of graphic in the Graphic (see page 14600) property. Some graphics generate this event, others do not. Jpeg images, for example, generate an OnProgress event.

Write an OnProgress event handler to provide the user with feedback during slow operations such as loading large compressed images. TImage, for example, hooks the OnProgress event of TPicture (see page 14584).

Note: The PercentDone parameter on the event handler is only an approximation. With some image formats, the value of PercentDone may actually decrease from the value in previous events, as the graphic object discovers there is more work to do.

See Also

Graphic (see page 14600)

OnProgress (see page 12831)

73.1.20 TSharedImage Class

TSharedImage is the base class for classes that contain images used for graphics objects.

Class Hierarchy

Graphics.TSharedImage

Pascal

```
TSharedImage = class;
```

C++

```
class TSharedImage;
```

File

Graphics

Description

TSharedImage is used as a base class for classes that hold the internal representation of graphical images such as bitmaps and icons. TSharedImage is an abstract class. Its descendant classes are used internally by graphics objects, and are automatically created by their associated graphics objects.

See Also

TGraphic (see page 14516)

73.1.20.1 TSharedImage Members

The following tables list the members exposed by TSharedImage.

Protected Methods

	Name	Description
	FreeHandle (see page 14604)	This is FreeHandle, a member of class TSharedImage.
	Reference (see page 14604)	This is Reference, a member of class TSharedImage.
	Release (see page 14604)	This is Release, a member of class TSharedImage.

Protected Properties

	Name	Description
	RefCount (see page 14605)	This is RefCount, a member of class TSharedImage.

73.1.20.2 TSharedImage Methods

The methods of the TSharedImage class are listed here.

Protected Methods

	Name	Description
	FreeHandle (see page 14604)	This is FreeHandle, a member of class TSharedImage.
	Reference (see page 14604)	This is Reference, a member of class TSharedImage.
	Release (see page 14604)	This is Release, a member of class TSharedImage.

73.1.20.2.1 TSharedImage.FreeHandle Method

Pascal

```
procedure FreeHandle; virtual; abstract;
```

C++

```
virtual __fastcall FreeHandle() = 0;
```

Description

This is FreeHandle, a member of class TSharedImage.

73.1.20.2.2 TSharedImage.Reference Method

Pascal

```
procedure Reference;
```

C++

```
__fastcall Reference();
```

Description

This is Reference, a member of class TSharedImage.

73.1.20.2.3 TSharedImage.Release Method

Pascal

```
procedure Release;
```

C++

```
__fastcall Release();
```

Description

This is Release, a member of class TSharedImage.

73.1.20.3 TSharedImage Properties

The properties of the TSharedImage class are listed here.

Protected Properties

	Name	Description
	RefCount (see page 14605)	This is RefCount, a member of class TSharedImage.

73.1.20.3.1 TSharedImage.RefCount Property

Pascal

```
property RefCount: Integer;
```

C++

```
__property int RefCount;
```

Description

This is RefCount, a member of class TSharedImage.

73.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IChangeNotifier (see page 14606)	IChangeNotifier is the interface used for responding to status changes.

73.2.1 IChangeNotifier Interface

IChangeNotifier is the interface used for responding to status changes.

Class Hierarchy

Graphics.IChangeNotifier

Pascal

```
IChangeNotifier = interface;
```

C++

```
__interface IChangeNotifier;
```

File

Graphics

Description

IChangeNotifier introduces one method, Changed (see page 14607) that is invoked when the status of the implementing class changes.

See Also

Changed (see page 14607)

73.2.1.1 IChangeNotifier Members

The following tables list the members exposed by IChangeNotifier.

Public Methods

	Name	Description
	Changed (see page 14607)	Called when a change occurs in the implementing class.

73.2.1.2 IChangeNotifier Methods

The methods of the IChangeNotifier class are listed here.

Public Methods

	Name	Description
	Changed (see page 14607)	Called when a change occurs in the implementing class.

73.2.1.2.1 IChangeNotifier.Changed Method

Called when a change occurs in the implementing class.

Pascal

```
procedure Changed;
```

C++

```
__fastcall Changed();
```

Description

Changed is principally used internally by TFont (see page 14497) class TFontAdapter property and TPicture (see page 14584) class TPictureAdapter property.

See Also

TFontAdapter

TPictureAdapter

73.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AllocPatternBitmap (see page 14608)	Creates a TBitmap (see page 14396) filled with a pattern.
	BytesPerScanline (see page 14609)	This is function Graphics.BytesPerScanline.
	CharsetToIdent (see page 14609)	Obtains the name of a character set.
	ColorToIdent (see page 14610)	Provides the symbolic name of a TColor (see page 14648) value.
	ColorToRGB (see page 14610)	Converts a TColor (see page 14648) value into an RGB representation of the color.
	ColorToString (see page 14611)	Returns a string that represents the name of a TColor (see page 14648) value.
	CopyPalette (see page 14612)	Creates a new palette object that matches an existing palette.
	CreateGrayMappedBmp (see page 14612)	Remaps the standard gray colors in a bitmap with the system grays.
	CreateGrayMappedRes (see page 14613)	Remaps the standard gray colors in a bitmap resource with the system grays.
	CreateMappedBmp (see page 14614)	Changes the color table in a bitmap.
	CreateMappedRes (see page 14614)	Changes the color table in a bitmap resource.
	FreeMemoryContexts (see page 14615)	This is function Graphics.FreeMemoryContexts.
	GetCharsetValues (see page 14615)	Passes the name of every predefined character set string to a callback function.
	GetColorValues (see page 14619)	Passes the name of every predefined color constant to a callback function.
	GetDefFontCharSet (see page 14619)	Returns the character set of the default system font.
	GetDIB (see page 14620)	Returns the info header and image for a device independent bitmap.
	GetDIBSizes (see page 14620)	Returns the size of the info header and of the image for a device independent bitmap.
	GraphicExtension (see page 14621)	Returns the default file-name extension of a graphics object.
	GraphicFileMask (see page 14624)	Returns a file mask that specifies the valid extensions for a graphic class.
	GraphicFilter (see page 14624)	Returns a file filter compatible with the Filter property of an Open or Save dialog.
	IdentToCharset (see page 14627)	Translates the name of a character set constant to the corresponding character set.
	IdentToColor (see page 14627)	Translates the name of a color constant to the corresponding TColor (see page 14648) value.
	PaletteChanged (see page 14628)	This is function Graphics.PaletteChanged.
	StringToColor (see page 14628)	Translates a string representation of a color to the corresponding TColor (see page 14648) value.
	TransparentStretchBlt (see page 14629)	This is function Graphics.TransparentStretchBlt.

73.3.1 Graphics.AllocPatternBitmap Function

Creates a TBitmap (see page 14396) filled with a pattern.

Pascal

```
function AllocPatternBitmap(BkColor: TColor; FgColor: TColor): TBitmap;
```

C++

```
TBitmap AllocPatternBitmap(TColor BkColor, TColor FgColor);
```

File

Graphics

Description

AllocPatternBitmap returns a reference to an 8 by 8 pixel TBitmap (see page 14396) that is filled with a pattern. Pixels alternate between **BkColor** and **FgColor** colors horizontally and vertically in a quilt pattern.

See Also

TBitmap (see page 14396)

TColor (see page 14648)

73.3.2 Graphics.BytesPerScanline Function

Pascal

```
function BytesPerScanline(PixelsPerScanline: Longint; BitsPerPixel: Longint; Alignment:
Longint): Longint;
```

C++

```
Longint BytesPerScanline(Longint PixelsPerScanline, Longint BitsPerPixel, Longint Alignment);
```

File

Graphics

Description

This is function Graphics.BytesPerScanline.

73.3.3 Graphics.CharsetToIdent Function

Obtains the name of a character set.

Pascal

```
function CharsetToIdent(Charset: Longint; var Ident: string): Boolean;
```

C++

```
Boolean CharsetToIdent(Longint Charset, AnsiString Ident);
```

File

Graphics

Description

Call CharsetToIdent to obtain the symbolic name for the character set of a font. CharsetToIdent returns true if the Ident parameter is successfully given an identifying name for the character set.

Note: The return value of CharsetToIdent does not indicate whether the Charset parameter is valid for a given font. Many fonts support only a limited number of character sets. Some fonts may support less common character sets that can't be mapped to identifiers by CharsetToIdent.

See Also

TFont (see page 14497)

73.3.4 Graphics.ColorToIdent Function

Provides the symbolic name of a TColor (see page 14648) value.

Pascal

```
function ColorToIdent(Color: Longint; var Ident: string): Boolean;
```

C++

```
Boolean ColorToIdent(Longint Color, AnsiString Ident);
```

File

Graphics

Description

Call ColorToIdent to obtain the name of the symbolic constant that represents a TColor (see page 14648) value, such as clBlack (see page 14666) or clWindow (see page 14732). The Ident parameter is set to the symbolic name. If there is no symbolic constant defined for the color, ColorToIdent returns false.

See Also

ColorToString (see page 14611)

TColor (see page 14648)

IdentToColor (see page 14627)

73.3.5 Graphics.ColorToRGB Function

Converts a TColor (see page 14648) value into an RGB representation of the color.

Pascal

```
function ColorToRGB(Color: TColor): Longint;
```

C++

```
Longint ColorToRGB(TColor Color);
```

File

Graphics

Description

Call ColorToRGB to obtain an RGB representation of a color for using with Windows API calls. ColorToRGB strips away the information that is stored in the highest order bits about which palette to use for colors that are not always available.

Delphi Examples:

```
{
The following code converts the color of the current form,
Form1, to a Windows RGB value:
}
const
  NumPaletteEntries = 20;
var
  FPaletteEntries: array[0..NumPaletteEntries - 1] of TPaletteEntry;
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
  Result: Integer;
  RealColor, paletteColor: TColor;
begin
  Result := 0;
  I := 0;
  RealColor := Graphics.ColorToRGB(Form1.Color);
  while I < NumPaletteEntries do
  begin
    with FPaletteEntries[I] do
    begin
      paletteColor:= TColor(RGB(peRed, peGreen, peBlue));
      if RealColor = paletteColor then
      begin
        Label1.Caption := IntToStr(I);
        RedEdit.Text:= IntToStr(peRed);
        GreenEdit.Text:= IntToStr(peGreen);
        BlueEdit.Text:= IntToStr(peBlue);
        Exit;
      end;
    end;
    Inc(I)
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form1.Color:=
    StrToInt(RedEdit.Text) shl 16 +
    StrToInt(GreenEdit.Text) shl 8 +
    StrToInt(BlueEdit.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  GetPaletteEntries(GetStockObject(DEFAULT_PALETTE), 0, NumPaletteEntries,
    FPaletteEntries);
  Form1.Color:= clMoneyGreen;
end;
```

See Also

ColorToIdent (see page 14610)

73.3.6 Graphics.ColorToString Function

Returns a string that represents the name of a TColor (see page 14648) value.

Pascal

```
function ColorToString(Color: TColor): string;
```

C++

```
AnsiString ColorToString(TColor Color);
```

File

Graphics

Description

Call ColorToString to obtain a string that represents a TColor (see page 14648) value. If there is a symbolic constant defined for the color (such as clBlack (see page 14666) or clHighlightText (see page 14674)), ColorToString returns the name of the constant. Otherwise, ColorToString returns the hex value of the color, formatted as a string such as "$02FF8800" (Delphi) or "0x02FF8800" (C++).

See Also

ColorToIdent (see page 14610)

StringToColor (see page 14628)

73.3.7 Graphics.CopyPalette Function

Creates a new palette object that matches an existing palette.

Pascal

```
function CopyPalette(Palette: HPALETTE): HPALETTE;
```

C++

```
HPALETTE CopyPalette(HPALETTE Palette);
```

File

Graphics

Description

Call CopyPalette to create a new Windows palette object that exactly matches an existing palette. If the Palette parameter does not refer to a valid Windows palette, or if a problem occurs while copying the palette, CopyPalette returns 0.

73.3.8 Graphics.CreateGrayMappedBmp Function

Remaps the standard gray colors in a bitmap with the system grays.

Pascal

```
function CreateGrayMappedBmp(Handle: HBITMAP): HBITMAP;
```

C++

```
HBITMAP CreateGrayMappedBmp(HBITMAP Handle);
```

File

Graphics

Description

Call CreateGrayMappedBmp to create a bitmap that uses the system gray colors (clBtnHightlight, clBtnFace (see page 14667), clBtnShadow (see page 14668) and clBtnText (see page 14668)) from a bitmap that uses standard gray scale colors (clWhite (see page 14732), clSilver (see page 14683), clGray (see page 14672), and clBlack (see page 14666)). This allows an application to dynamically create images that match the user's control panel settings, given a base image.

CreateGrayMappedBmp creates a new bitmap. It does not alter the original bitmap, which has the handle specified by the Handle parameter.

Note: CreateGrayMappedBmp does not work with bitmaps that have more than 256 colors. For bitmaps with more than 256 colors (8bpp), this function returns the original bitmap.

See Also

TBitmap (see page 14396)

CreateMappedBmp (see page 14614)

CreateGrayMappedRes (see page 14613)

Graphics

73.3.9 Graphics.CreateGrayMappedRes Function

Remaps the standard gray colors in a bitmap resource with the system grays.

Pascal

```
function CreateGrayMappedRes(Instance: THandle; ResName: PChar): HBITMAP;
```

C++

```
HBITMAP CreateGrayMappedRes(THandle Instance, const char * ResName);
```

File

Graphics

Description

Call CreateGrayMappedRes to create a bitmap that uses the system gray colors (clBtnHightlight, clBtnFace (see page 14667), clBtnShadow (see page 14668) and clBtnText (see page 14668)) from a resourced bitmap that uses standard gray scale colors (clWhite (see page 14732), clSilver (see page 14683), clGray (see page 14672), and clBlack (see page 14666)). This allows an application to dynamically create images that match the user's control panel settings, by creating a base image as a bitmap resource.

Use the Instance and ResName parameters to identify the bitmap resource whose color table should be changed

Note: CreateMappedRes (see page 14614) does not work with bitmaps that have more than 256 colors. For bitmaps with more than 256 colors (8bpp), this function returns an instance of the original bitmap without color modifications.

See Also

TBitmap (see page 14396)

CreateGrayMappedBmp (see page 14612)

FindResourceHInstance

Hinstance

CreateMappedRes (see page 14614)

Graphics

73.3.10 Graphics.CreateMappedBmp Function

Changes the color table in a bitmap.

Pascal

```
function CreateMappedBmp(Handle: HBITMAP; const OldColors: array of TColor; const NewColors:
array of TColor): HBITMAP;
```

C++

```
HBITMAP CreateMappedBmp(HBITMAP Handle, const array of TColor OldColors, const array of TColor
NewColors);
```

File

Graphics

Description

Call CreateMappedBmp to make selective changes to the color map of a bitmap. This function replaces the colors specified by the OldColors parameter with the colors specified by the NewColors parameter and returns the handle for a new device dependent bitmap (DDB) that uses the new color table. Handle is the handle to the bitmap whose color table should be changed. In C++, specify the size of the color arrays in OldColors_Size and NewColors_Size. This value should be the index of the last entry in the corresponding array. The OldColors and NewColors arrays should contain the same number of elements.

CreateMappedBmp creates a new bitmap. It does not alter the original bitmap, which has the handle specified by the Handle parameter.

Note: CreateMappedBmp does not work with bitmaps that have more than 256 colors. For bitmaps with more than 256 colors (8bpp), this function returns the original bitmap.

See Also

TBitmap (see page 14396)

CreateMappedRes (see page 14614)

CreateGrayMappedBmp (see page 14612)

73.3.11 Graphics.CreateMappedRes Function

Changes the color table in a bitmap resource.

Pascal

```
function CreateMappedRes(Instance: THandle; ResName: PChar; const OldColors: array of TColor;
const NewColors: array of TColor): HBITMAP;
```

C++

```
HBITMAP CreateMappedRes(THandle Instance, const char * ResName, const array of TColor
OldColors, const array of TColor NewColors);
```

File

Graphics

Description

Call CreateMappedRes to make selective changes to the color map of a resourced bitmap. This function replaces the colors specified by the OldColors parameter with the colors specified by the NewColors parameter and returns the handle for a new device dependent bitmap (DDB) that uses the new color table. Use the Instance and ResName parameters to identify the bitmap resource whose color table should be changed. In C++, indicate the number of elements in the color arrays by the OldColors_Size and NewColors_Size parameters. These values should be the index of the last element in the corresponding color array, not the number of entries. The OldColors and NewColors arrays should contain the same number of elements.

Note: CreateMappedRes does not work with bitmaps that have more than 256 colors. For bitmaps with more than 256 colors (8bpp), this function returns an instance of the original bitmap without color modifications.

See Also

TBitmap (see page 14396)

CreateMappedBmp (see page 14614)

FindResourceHInstance

Hinstance

CreateGrayMappedRes (see page 14613)

73.3.12 Graphics.FreeMemoryContexts Function

Pascal

```
procedure FreeMemoryContexts;
```

C++

```
FreeMemoryContexts();
```

File

Graphics

Description

This is function Graphics.FreeMemoryContexts.

73.3.13 Graphics.GetCharsetValues Function

Passes the name of every predefined character set string to a callback function.

Pascal

```
procedure GetCharsetValues(Proc: TGetStrProc);
```

C++

```
GetCharsetValues(TGetStrProc Proc);
```

File

Graphics

Description

Call GetCharsetValues to execute a callback for every character set name that has a corresponding constant defined in the application. The Proc parameter is the callback function that is called for every character set constant name.

Note: To convert the strings that are passed to the callback into the TFontCharset (see page 14651) type, use the IdentToCharset (see page 14627) function.

C++ Examples:

```
/*
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
*/
void __fastcall TForm1::ComboBox1KeyPress(TObject *Sender, char &Key)
{
  // first, process the keystroke to obtain the current string
  // This code requires all items in list to be uppercase
  if (Key >= 'a' && Key <= 'z')
    Key -= 32; // Force Uppercase only!
  TComboBox *pCB = (TComboBox *)Sender;
  AnsiString TmpStr;
  bool BackSpace = (Key == (char)VK_BACK);
  if (BackSpace && pCB->SelLength)
    TmpStr = pCB->Text.SubString(1,pCB->SelStart)+
             pCB->Text.SubString(pCB->SelLength+pCB->SelStart+1,255);
  else if (BackSpace) // SelLength == 0
    TmpStr = pCB->Text.SubString(1,pCB->SelStart-1)+
             pCB->Text.SubString(pCB->SelStart+1,255);
  else //Key is a visible character
    TmpStr = pCB->Text.SubString(1,pCB->SelStart)+ Key +
             pCB->Text.SubString(pCB->SelLength+pCB->SelStart+1,255);
  if (TmpStr.IsEmpty())
    return;
  // set SelSt to the current insertion point
  int SelSt = pCB->SelStart;
  if (BackSpace && SelSt > 0)
    SelSt--;
  else if (!BackSpace)
    SelSt++;
  Key = 0; // indicate that key was handled
  if (SelSt == 0)
  {
    pCB->Text = "";
    return;
  }
  // Now that TmpStr is the currently typed string, see if we can locate a match
  bool Found = false;
  for (int i = 1; i < pCB->Items->Count; i++)
    if (TmpStr == pCB->Items->Strings[i-1].SubString(1,TmpStr.Length()))
    {
      pCB->Text = pCB->Items->Strings[i-1]; // update to the match that was found
      pCB->ItemIndex = i-1;
      Found = true;
      break;
    }
  if (Found) // select the untyped end of the string
  {
```

```
    pCB->SelStart = SelSt;
    pCB->SelLength = pCB->Text.Length() - SelSt;
  }
  else Beep();
}
//---------------------------------------------------------------------------
TStringList *CharSetList;

void __fastcall TForm1::AddCharacterSet(AnsiString S)
{
  CharSetList->Add(S);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
CharSetList = new TStringList();
GetCharsetValues(AddCharacterSet);
CharSetList->Sort();
ComboBox1->Items = CharSetList;
}
```

Delphi Examples:

```
{
The following uses the GetCharsetValues procedure to fill a
string list with the names of all the available character
set constants. This string list can then be used to allow
the user to select a character set.  Here is the callback
that adds the character sets to the string list:
}
var
  CharSetList : TStringList;

procedure TForm1.AddCharacterSet(const S: string);
begin
  CharSetList.Add(S);
end;
{
Here is the code that creates the string list, fills it, and sorts the character set names.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  CharSetList := TStringList.Create;
  Graphics.GetCharsetValues(AddCharacterSet);
  CharSetList.Sort;
  ListBox1.Items := CharSetList;
  CharSetList.Destroy;
end;
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
Note: Set the TComboBox AutoComplete to false to turn off
default auto completion.
}
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
var
  Found: boolean;
  i, ct, SelSt: Integer;
  TmpStr: string;
begin
  { first, process the keystroke to obtain the current string }
```

```
    { This code requires all items in list to be uppercase}
    if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
    ct := (Sender as TComboBox).Items.Count;
    with (Sender as TComboBox) do
    begin
      SelSt := SelStart;
      if (Key = Chr(vk_Back)) and (SelLength <> 0) then
       TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
      else if Key = Chr(vk_Back) then {SelLength = 0}
       TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
      else {Key in ['A'..'Z', etc]}
       TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
      if TmpStr = '' then Exit;
      { update SelSt to the current insertion point }
      if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
      else if Key <> Chr(vk_Back) then Inc(SelSt);
      Key := #0; { indicate that key was handled }
      if SelSt = 0 then
      begin
        Text:= '';
        Exit;
      end;
     {Now that TmpStr is the currently typed string, see if we can locate a match }
      Found := False;
      for i := 1 to ct do
      begin
        if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
        begin
          Text := Items[i-1]; { update to the match that was found }
          ItemIndex := i-1;
          Found := True;
          Break;
        end;
      end;
      if Found then { select the untyped end of the string }
      begin
        SelStart := SelSt;
        SelLength := Length(Text) - SelSt;
      end
      else Beep;
    end;
  end;

  var
    CharSetList : TStringList;

  procedure TForm1.AddCharacterSet(const S: string);
  begin
    CharSetList.Add(S);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    CharSetList := TStringList.Create;
    Graphics.GetCharsetValues(AddCharacterSet);
    ComboBox1.Items := CharSetList;
  end;
```

See Also

TFont (see page 14497)

IdentToCharset (see page 14627)

TFontCharset (see page 14651)

73.3.14 Graphics.GetColorValues Function

Passes the name of every predefined color constant to a callback function.

Pascal

```
procedure GetColorValues(Proc: TGetStrProc);
```

C++

```
GetColorValues(TGetStrProc Proc);
```

File

Graphics

Description

Call GetColorValues to execute a callback for every TColor (see page 14648) constant defined in the application. The Proc parameter is the callback function that is called for every color constant name.

Note: To convert the strings that are passed to the callback into the TColor (see page 14648) type, use the IdentToColor (see page 14627) function.

See Also

TColor (see page 14648)

IdentToColor (see page 14627)

73.3.15 Graphics.GetDefFontCharSet Function

Returns the character set of the default system font.

Pascal

```
function GetDefFontCharSet: TFontCharSet;
```

C++

```
TFontCharSet GetDefFontCharSet();
```

File

Graphics

Description

Call GetDefFontCharSet to determine the character set that is used when a font is not explicitly assigned to a control. This character set can be used to initialize a font that is specified by font name only, to best match the characters for the locale of the machine on which the application is running.

See Also

TFont (see page 14497)

73.3.16 Graphics.GetDIB Function

Returns the info header and image for a device independent bitmap.

Pascal

```
function GetDIB(Bitmap: HBITMAP; Palette: HPALETTE; var BitmapInfo; var Bits): Boolean;
```

C++

```
Boolean GetDIB(HBITMAP Bitmap, HPALETTE Palette,  BitmapInfo,  Bits);
```

File

Graphics

Description

Call GetDIB to convert a device dependent bitmap into a device independent bitmap (DIB). Pass the handle to a device dependent bitmap and a handle to its palette as the Bitmap and Palette parameters. GetDIB fills in the BitmapInfo and Bits parameters with the info header and image of the corresponding DIB. GetDIB returns true if the bitmap was successfully converted into a DIB, false otherwise.

Before calling GetDIB, the memory for the info header and image must be allocated. Call GetDIBSizes (see page 14620) to determine the amount of memory that must be allocated for the info header and image.

See Also

GetDIBSizes (see page 14620)

73.3.17 Graphics.GetDIBSizes Function

Returns the size of the info header and of the image for a device independent bitmap.

Pascal

```
procedure GetDIBSizes(Bitmap: HBITMAP; var InfoHeaderSize: DWORD; var ImageSize: DWORD);
```

C++

```
GetDIBSizes(HBITMAP Bitmap, DWORD InfoHeaderSize, DWORD ImageSize);
```

File

Graphics

Description

Call GetDIBSizes to determine the amount of memory needed to convert a device dependent bitmap into a device independent bitmap (DIB). Pass the handle to a device dependent bitmap as the Bitmap parameter. GetDIBSizes returns the sizes of the info header and image of the corresponding DIB as the InfoHeaderSize parameter and the ImageSize parameter, respectively.

Call GetDIBSizes to determine the amount of memory that must be allocated before calling GetDIB (see page 14620) to convert a device dependent bitmap into a device independent bitmap.

See Also

GetDIB (see page 14620)

73.3.18 Graphics.GraphicExtension Function

Returns the default file-name extension of a graphics object.

Pascal

```
function GraphicExtension(GraphicClass: TGraphicClass): string;
```

C++

```
AnsiString GraphicExtension(TGraphicClass GraphicClass);
```

File

Graphics

Description

Call GraphicExtension to obtain a default extension value for the DefaultExt property of an Open, Open Picture, Save Picture or Save dialog. GraphicExtension returns the default extension for the type of graphics object specified by the GraphicClass parameter. In Delphi, the TGraphicClass (see page 14654) type is the class type for descendants TGraphic (see page 14516).

These are the file extensions returned for each built-in graphics class:

Graphic class	File extension returned
TBitmap (see page 14396)	bmp
TIcon (see page 14537)	ico
TMetafile (see page 14549)	emf

If custom graphics classes register a file extension, GraphicExtension will return that extension when passed the custom graphics class as an argument.

C++ Examples:

```
/*
The following example allows users to type the name of a
graphics class into an edit control and when a button is
pressed, the name of the default extension for that class is
displayed in another edit control.  In the form's OnCreate
event handler (or a similar place) you must register the
graphics classes so that GetClass can find them.
*/
TMetaClass *MetaClass;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClassFinder *myClassFinder = new TClassFinder(MetaClass, False);
  TGraphicClass mygraphclass = (TGraphicClass) (myClassFinder->GetClass(Edit2->Text));
//  TGraphicClass mygraphclass = (TGraphicClass) GetClass(Edit2->Text);
  Edit1->Text = GraphicExtension(mygraphclass);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MetaClass = __classid(TIcon); // ico
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(Graphics::TBitmap); // bmp
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TMetafile); // emf
```

```
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TShape);  // nographic extension
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TImage); // nographic extension
  RegisterClasses(&MetaClass, 0);
}
/*
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString NewFileName, OldFileName;
  TFileStream *NewFile, *OldFile;
  SavePictureDialog1->DefaultExt = GraphicExtension(__classid(Graphics::TBitmap));
  SavePictureDialog1->Filter = GraphicFilter(__classid(Graphics::TBitmap));
  OldFileName = "factory.bmp";
  if (SavePictureDialog1->Execute())
  {
    NewFileName = SavePictureDialog1->FileName;
    OldFile = new TFileStream(OldFileName, (fmOpenRead | fmShareDenyWrite));
    try
    {
      NewFile = new TFileStream(NewFileName, (fmCreate | fmShareDenyRead));
      try
      {
        NewFile->CopyFrom(OldFile, OldFile->Size);
      }
      __finally
      {
        FreeAndNil(&NewFile);
      };
    }
    __finally
    {
      FreeAndNil(&OldFile);
    };
  };
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Image1->Picture->LoadFromFile("../factory.bmp");
}
```

Delphi Examples:

```
{
The following example allows users to type the name of a
graphics class into an edit control and when a button is
pressed, the name of the default extension for that class is
displayed in another edit control.  In the form’s OnCreate
event handler (or a similar place) you must register the
graphics classes so that GetClass can find them.
}

uses ExtCtrls;

procedure TForm1.Button1Click(Sender: TObject);
var
  myClassFinder: TClassFinder;
  mygraphclass: TGraphicClass;
begin
  { make sure the classes are registered so that GetClass will work -- }
```

```
    { Usually, this goes in the initialization section where it is only executed once }
    RegisterClasses([TIcon, TBitmap, TButton, TShape, TImage, TMetafile]);
    try
      myClassFinder := TClassFinder.Create(TIcon, False);
      mygraphclass := TGraphicClass(myClassFinder.GetClass(Edit2.Text));
//      mygraphclass := TGraphicClass(GetClass(Edit2.Text));  // works as well
      Edit1.Text := GraphicExtension(mygraphclass);
    except
      on EAccessViolation do
        MessageDlg('Invalid class name.', mtError, [mbOK], 0)
    end;
  end;
  {
  This code displays a Save Picture dialog box with the
  TBitmap default extension that is added automatically to
  files that are given no extension. Confirm that the
  bitmap file has been saved under the filename and path
  specified.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var NewFileName, OldFileName: string;
      NewFile, OldFile: TFileStream;
  begin
    SavePictureDialog1.DefaultExt := GraphicExtension(TBitmap);
    SavePictureDialog1.Filter := GraphicFilter(TBitmap);
    OldFileName := 'factory.bmp';
    if SavePictureDialog1.Execute then
    begin
      NewFileName := SavePictureDialog1.FileName;
      OldFile := TFileStream.Create(
        OldFileName, fmOpenRead or fmShareDenyWrite);
      try
        NewFile := TFileStream.Create(
          NewFileName, fmCreate or fmShareDenyRead);
        try
          NewFile.CopyFrom(OldFile, OldFile.Size);
        finally
          FreeAndNil(NewFile);
        end;
      finally
        FreeAndNil(OldFile);
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Image1.Picture.LoadFromFile('factory.bmp');
  end;
```

See Also

GraphicFileMask (see page 14624)

GraphicFilter (see page 14624)

TPicture (see page 14584)

TOpenDialog

TOpenPictureDialog

TSavePictureDialog

73.3.19 Graphics.GraphicFileMask Function

Returns a file mask that specifies the valid extensions for a graphic class.

Pascal

```
function GraphicFileMask(GraphicClass: TGraphicClass): string;
```

C++

```
AnsiString GraphicFileMask(TGraphicClass GraphicClass);
```

File

Graphics

Description

Call GraphicFileMask to obtain a mask to determine the file extensions associated with a graphic class. The GraphicClass parameter can specify one of these values: TBitmap (see page 14396), TGraphic (see page 14516), TIcon (see page 14537), TMetafile (see page 14549), or a user-defined descendant of TGraphic (see page 14516). These are the strings that are returned for each built-in graphics class:

Graphic class	Filter string returned
TBitmap (see page 14396)	*.bmp
TIcon (see page 14537)	*.ico
TMetafile (see page 14549)	*.emf;*.wmf
TGraphic (see page 14516)	*.bmp;*.ico;*.emf;*.wmf
TGraphic (see page 14516)	;*.bmp;*.png;*.xpm*.ico

See Also

GraphicExtension (see page 14621)

GraphicFilter (see page 14624)

TFilterComboBox

TPicture (see page 14584)

73.3.20 Graphics.GraphicFilter Function

Returns a file filter compatible with the Filter property of an Open or Save dialog.

Pascal

```
function GraphicFilter(GraphicClass: TGraphicClass): string;
```

C++

```
AnsiString GraphicFilter(TGraphicClass GraphicClass);
```

File

Graphics

Description

Call GraphicFilter to obtain a value for the Filter property of an Open, Open Picture, Save Picture or Save dialog box. The GraphicClass parameter can specify one of these values: TBitmap (see page 14396), TGraphic (see page 14516), TIcon (see page 14537), TMetafile (see page 14549), or a user-defined descendant of TGraphic (see page 14516).

These are the strings that are returned for each built-in graphics class when the AllFilter parameter is true. (When AllFilter is false, the section labelled "All" is omitted):

Graphic class	Filter string returned
TBitmap (see page 14396)	Bitmaps (*.bmp;*.png;*.xpm)|*.bmp;*.png;*.xpm
TIcon (see page 14537)	Icons (*.ico)|*.ico
TGraphic (see page 14516)	All (*.bmp;*.png;*.xpm*.ico)|*.bmp;*.png;*.xpm*.ico|Bitmaps (*.bmp;*.png;*.xpm)|*.bmp;*.png;*.xpm|Icons (*.ico)|*.ico
TBitmap (see page 14396)	Bitmaps (*.bmp)|*.bmp
TIcon (see page 14537)	Icons (*.ico)|*.ico
TMetafile (see page 14549)	All (*.emf;*.wmf)|*.emf;*.wmf|Enhanced Metafiles (*.emf)|*.emf|Metafiles(*.wmf)|*.wmf
TGraphic (see page 14516)	All (*.bmp;*.ico;*.emf;*.wmf)|*.bmp;*.ico;*.emf; *.wmf|Bitmaps (*.bmp)|*.bmp|Icons (*.ico)|*.ico|Enhanced Metafiles (*.emf)|*.emf|Metafiles(*.wmf)|*.wmf

Graphic class	Filter string returned
TBitmap (see page 14396)	Bitmaps (*.bmp)|*.bmp
TIcon (see page 14537)	Icons (*.ico)|*.ico
TMetafile (see page 14549)	All (*.emf;*.wmf)|*.emf;*.wmf|Enhanced Metafiles (*.emf)|*.emf|Metafiles(*.wmf)|*.wmf
TGraphic (see page 14516)	All (*.bmp;*.ico;*.emf;*.wmf)|*.bmp;*.ico;*.emf; *.wmf|Bitmaps (*.bmp)|*.bmp|Icons (*.ico)|*.ico|Enhanced Metafiles (*.emf)|*.emf|Metafiles(*.wmf)|*.wmf

C++ Examples:

```
/*
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString NewFileName, OldFileName;
  TFileStream *NewFile, *OldFile;
  SavePictureDialog1->DefaultExt = GraphicExtension(__classid(Graphics::TBitmap));
  SavePictureDialog1->Filter = GraphicFilter(__classid(Graphics::TBitmap));
  OldFileName = "factory.bmp";
  if (SavePictureDialog1->Execute())
  {
```

```
    NewFileName = SavePictureDialog1->FileName;
    OldFile = new TFileStream(OldFileName, (fmOpenRead | fmShareDenyWrite));
    try
    {
      NewFile = new TFileStream(NewFileName, (fmCreate | fmShareDenyRead));
      try
      {
        NewFile->CopyFrom(OldFile, OldFile->Size);
      }
      __finally
      {
        FreeAndNil(&NewFile);
      };
    }
    __finally
    {
      FreeAndNil(&OldFile);
    };
  };
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Image1->Picture->LoadFromFile("../factory.bmp");
}
```

Delphi Examples:

```
{
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
}
procedure TForm1.Button1Click(Sender: TObject);
var NewFileName, OldFileName: string;
    NewFile, OldFile: TFileStream;
begin
  SavePictureDialog1.DefaultExt := GraphicExtension(TBitmap);
  SavePictureDialog1.Filter := GraphicFilter(TBitmap);
  OldFileName := 'factory.bmp';
  if SavePictureDialog1.Execute then
  begin
    NewFileName := SavePictureDialog1.FileName;
    OldFile := TFileStream.Create(
      OldFileName, fmOpenRead or fmShareDenyWrite);
    try
      NewFile := TFileStream.Create(
        NewFileName, fmCreate or fmShareDenyRead);
      try
        NewFile.CopyFrom(OldFile, OldFile.Size);
      finally
        FreeAndNil(NewFile);
      end;
    finally
      FreeAndNil(OldFile);
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Picture.LoadFromFile('factory.bmp');
end;
```

See Also

GraphicExtension (see page 14621)

GraphicFileMask (see page 14624)

TOpenDialog

TOpenPictureDialog

TPicture (see page 14584)

TSavePictureDialog

73.3.21 Graphics.IdentToCharset Function

Translates the name of a character set constant to the corresponding character set.

Pascal

```
function IdentToCharset(const Ident: string; var Charset: Longint): Boolean;
```

C++

```
Boolean IdentToCharset(const AnsiString Ident, Longint Charset);
```

File

Graphics

Description

Call IdentToCharset to reverse the translation performed by CharsetToIdent (see page 14609). This method is useful for converting the strings generated by GetCharsetValues (see page 14615) into useable character sets. IdentToCharset returns true if the Ident parameter is successfully converted into a character set that is returned by the Charset parameter. IdentToCharset returns false if the Ident parameter is not the name of one of the TFontCharset (see page 14651) constants.

See Also

TFont (see page 14497)

GetCharsetValues (see page 14615)

CharSetToIdent

TFontCharset (see page 14651)

73.3.22 Graphics.IdentToColor Function

Translates the name of a color constant to the corresponding TColor (see page 14648) value.

Pascal

```
function IdentToColor(const Ident: string; var Color: Longint): Boolean;
```

C++

```
Boolean IdentToColor(const AnsiString Ident, Longint Color);
```

File

Graphics

Description

Call IdentToColor to reverse the translation performed by ColorToIdent (see page 14610). This method is useful for converting the strings generated by GetColorValues (see page 14619) into useable TColor (see page 14648) values. IdentToColor returns true if the Ident parameter is successfully converted into a color that is returned by the Color parameter. IdentToColor returns false if the Ident parameter is not the name of one of the TColor (see page 14648) constants.

Note: If the string identifier for a color may represent either a hex value or a color constant, use the StringToColor (see page 14628) function instead.

See Also

TColor (see page 14648)

GetColorValues (see page 14619)

ColorToIdent (see page 14610)

StringToColor (see page 14628)

73.3.23 Graphics.PaletteChanged Function

Pascal

```
procedure PaletteChanged;
```

C++

```
PaletteChanged();
```

File

Graphics

Description

This is function Graphics.PaletteChanged.

73.3.24 Graphics.StringToColor Function

Translates a string representation of a color to the corresponding TColor (see page 14648) value.

Pascal

```
function StringToColor(const S: string): TColor;
```

C++

```
TColor StringToColor(const AnsiString S);
```

File

Graphics

Description

Call StringToColor to reverse the translation performed by ColorToString (see page 14611). This method is useful for

converting the strings entered by the user into useable TColor (see page 14648) values.

The S parameter can be either the name of a built-in color constant, such as "clBtnFace (see page 14667)" (Windows) or "clBtnText (see page 14668)" (cross-platform), or the string representation of a valid TColor (see page 14648) value such as "$02FF8800" (Delphi) or "0x02FF8800" (C++). StringToColor returns the TColor (see page 14648) value that corresponds to the S parameter.

Note: If the string identifier can only be the name of a color constant, use IdentToColor (see page 14627) instead.

See Also

ColorToString (see page 14611)

IdentToColor (see page 14627)

73.3.25 Graphics.TransparentStretchBlt Function

Pascal

```
function TransparentStretchBlt(DstDC: HDC; DstX: Integer; DstY: Integer; DstW: Integer; DstH:
Integer; SrcDC: HDC; SrcX: Integer; SrcY: Integer; SrcW: Integer; SrcH: Integer; MaskDC: HDC;
MaskX: Integer; MaskY: Integer): Boolean;
```

C++

```
Boolean TransparentStretchBlt(HDC DstDC, int DstX, int DstY, int DstW, int DstH, HDC SrcDC,
int SrcX, int SrcY, int SrcW, int SrcH, HDC MaskDC, int MaskX, int MaskY);
```

File

Graphics

Description

This is function Graphics.TransparentStretchBlt.

73.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TBitmapHandleType (see page 14630)	TBitmapHandleType indicates the format used to store the in-memory image of a bitmap.
	TBrushStyle (see page 14631)	TBrushStyle specifies the pattern on a brush.
	TCanvasOrientation (see page 14632)	TCanvasOrientation specifies the pixel coordinate system that a canvas uses.
	TCanvasStates (see page 14633)	This is record Graphics.TCanvasStates.
	TFillMode (see page 14634)	This is record Graphics.TFillMode.
	TFillStyle (see page 14634)	TFillStyle indicates the type of color change defines the boundary of a possibly non-rectangular region n a canvas.
	TFontPitch (see page 14635)	TFontPitch indicates whether the characters in a font have the same width.
	TFontStyle (see page 14636)	TFontStyle and TFontStyles (see page 14653) specify style characteristics of a font.
	TPenMode (see page 14637)	TPenMode indicates how the color of a pen interacts with the color of a canvas.
	TPenStyle (see page 14639)	TPenStyle indicates the type of line a pen draws.
	TPixelFormat (see page 14640)	TPixelFormat specifies the format of a bitmap.
	TProgressStage (see page 14641)	TProgressStage indicates the stage of a lengthy operation.
	TTextFormats (see page 14642)	This is record Graphics.TTextFormats.
	TTransparentMode (see page 14643)	TTransparentMode indicates how the transparent color of a bitmap is determined.

Records

	Name	Description
	TBrushData (see page 14631)	This is record Graphics.TBrushData.
	TCursorOrIcon (see page 14633)	This is record Graphics.TCursorOrIcon.
	TFontData (see page 14635)	This is record Graphics.TFontData.
	TIconRec (see page 14636)	This is record Graphics.TIconRec.
	TPenData (see page 14637)	This is record Graphics.TPenData.
	TResData (see page 14641)	This is record Graphics.TResData.
	TResource (see page 14642)	This is record Graphics.TResource.

73.4.1 Graphics.TBitmapHandleType Enumeration

TBitmapHandleType indicates the format used to store the in-memory image of a bitmap.

Pascal

```
TBitmapHandleType = (
  bmDIB,
  bmDDB
);
```

C++

```
enum TBitmapHandleType {
  bmDIB,
  bmDDB
};
```

File

Graphics

Description

TBitmapHandleType can be any of the following values:

Value	Description
bmDIB	device-independent bitmap (DIB)
bmDDB	device-dependent bitmap (DDB)

73.4.2 Graphics.TBrushData Record

Pascal

```
TBrushData = record
  Handle: HBrush;
  Color: TColor;
  Bitmap: TBitmap;
  Style: TBrushStyle;
  OwnsBitmap: Boolean;
end;
```

C++

```
struct TBrushData {
  HBrush Handle;
  TColor Color;
  TBitmap Bitmap;
  TBrushStyle Style;
  Boolean OwnsBitmap;
};
```

File

Graphics

Description

This is record Graphics.TBrushData.

73.4.3 Graphics.TBrushStyle Enumeration

TBrushStyle specifies the pattern on a brush.

Pascal

```
TBrushStyle = (
  bsSolid,
  bsClear,
  bsHorizontal,
  bsVertical,
  bsFDiagonal,
  bsBDiagonal,
  bsCross,
  bsDiagCross
);
```

C++

```
enum TBrushStyle {
  bsSolid,
  bsClear,
  bsHorizontal,
  bsVertical,
  bsFDiagonal,
  bsBDiagonal,
  bsCross,
  bsDiagCross
};
```

File

Graphics

Description

The possible values of TBrushStyle are

ValuePatternValuePattern

Value	Pattern
bsSolid	bsCross
bsClear	bsDiagCross
bsBDiagonal	bsHorizontal
bsFDiagonal	bsVertical

See Also

TBrush (see page 14430)

73.4.4 Graphics.TCanvasOrientation Enumeration

TCanvasOrientation specifies the pixel coordinate system that a canvas uses.

Pascal

```
TCanvasOrientation = (
  coLeftToRight,
  coRightToLeft
);
```

C++

```
enum TCanvasOrientation {
  coLeftToRight,
  coRightToLeft
};
```

File

Graphics

Description

TCanvasOrientation determines where the (0,0) point falls on a canvas, and whether pixel coordinates are interpreted in a right-to-left or left-to-right manner.

The following table describes the possible values:

Value	Meaning
coLeftToRight	The origin (0,0) appears in the top-left corner. Pixel coordinates get larger as you move left or down.
coRightToLeft	The origin (0,0) appears in the top-right corner. Pixel coordinates get larger as you move right or down.

73.4.5 Graphics.TCanvasStates Enumeration

Pascal

```
TCanvasStates = (
  csHandleValid,
  csFontValid,
  csPenValid,
  csBrushValid
);
```

C++

```
enum TCanvasStates {
  csHandleValid,
  csFontValid,
  csPenValid,
  csBrushValid
};
```

File

Graphics

Description

This is record Graphics.TCanvasStates.

73.4.6 Graphics.TCursorOrIcon Record

Pascal

```
TCursorOrIcon = packed record
  Reserved: Word;
  wType: Word;
  Count: Word;
end;
```

C++

```
struct TCursorOrIcon {
  Word Reserved;
  Word wType;
  Word Count;
};
```

File

Graphics

Description

This is record Graphics.TCursorOrIcon.

73.4.7 Graphics.TFillMode Enumeration

Pascal

```
TFillMode = (
  fmAlternate,
  fmWinding
);
```

C++

```
enum TFillMode {
  fmAlternate,
  fmWinding
};
```

File

Graphics

Description

This is record Graphics.TFillMode.

73.4.8 Graphics.TFillStyle Enumeration

TFillStyle indicates the type of color change defines the boundary of a possibly non-rectangular region n a canvas.

Pascal

```
TFillStyle = (
  fsSurface,
  fsBorder
);
```

C++

```
enum TFillStyle {
  fsSurface,
  fsBorder
};
```

File

Graphics

Description

TFillStyle values determine what type of color changes define the boundaries of a (possibly) non-rectangular region of a canvas, as indicated in the following table.

Value	Meaning
fsSurface	The region is defined as all area that has a specified color. When another color is encountered, it is taken as the boundary of the region.
fsBorder	The region is defined as all area that does not have a specified color. When the specified color is encountered, it is taken as the boundary of the region.

73.4.9 Graphics.TFontData Record

Pascal

```
TFontData = record
  Handle: HFont;
  Height: Integer;
  Orientation: Integer;
  Pitch: TFontPitch;
  Style: TFontStylesBase;
  Charset: TFontCharset;
  Name: TFontDataName;
end;
```

C++

```
struct TFontData {
  HFont Handle;
  int Height;
  int Orientation;
  TFontPitch Pitch;
  TFontStylesBase Style;
  TFontCharset Charset;
  TFontDataName Name;
};
```

File

Graphics

Description

This is record Graphics.TFontData.

73.4.10 Graphics.TFontPitch Enumeration

TFontPitch indicates whether the characters in a font have the same width.

Pascal

```
TFontPitch = (
  fpDefault,
  fpVariable,
  fpFixed
);
```

C++

```
enum TFontPitch {
  fpDefault,
  fpVariable,
  fpFixed
};
```

File

Graphics

Description

The following table lists the possible values of the TFontPitch type:

Value	Meaning
fpDefault	The font pitch is set to the default value, which depends on the font specified in the font object's Name property.
fpFixed	The font pitch is set to fixed. All characters in the font have the same width.
fpVariable	The font pitch is set to variable. The characters in the font have different widths.

73.4.11 Graphics.TFontStyle Enumeration

TFontStyle and TFontStyles (see page 14653) specify style characteristics of a font.

Pascal

```
TFontStyle = (
  fsBold,
  fsItalic,
  fsUnderline,
  fsStrikeOut
);
```

C++

```
enum TFontStyle {
  fsBold,
  fsItalic,
  fsUnderline,
  fsStrikeOut
};
```

File

Graphics

Description

TFontStyles (see page 14653) is a set of TFontStyle values. The following table lists the possible values of the TFontStyle type:

Value	Meaning
fsBold	The font is boldfaced.
fsItalic	The font is italicized.
fsUnderline	The font is underlined.
fsStrikeOut	The font is displayed with a horizontal line through it.

73.4.12 Graphics.TIconRec Record

Pascal

```
TIconRec = packed record
  Width: Byte;
  Height: Byte;
  Colors: Word;
  Reserved1: Word;
```

```
    Reserved2: Word;
    DIBSize: Longint;
    DIBOffset: Longint;
  end;
```

C++

```
  struct TIconRec {
    Byte Width;
    Byte Height;
    Word Colors;
    Word Reserved1;
    Word Reserved2;
    Longint DIBSize;
    Longint DIBOffset;
  };
```

File

Graphics

Description

This is record Graphics.TIconRec.

73.4.13 Graphics.TPenData Record

Pascal

```
  TPenData = record
    Handle: HPen;
    Color: TColor;
    Width: Integer;
    Style: TPenStyle;
  end;
```

C++

```
  struct TPenData {
    HPen Handle;
    TColor Color;
    int Width;
    TPenStyle Style;
  };
```

File

Graphics

Description

This is record Graphics.TPenData.

73.4.14 Graphics.TPenMode Enumeration

TPenMode indicates how the color of a pen interacts with the color of a canvas.

Pascal

```
  TPenMode = (
    pmBlack,
    pmWhite,
```

```
    pmNop,
    pmNot,
    pmCopy,
    pmNotCopy,
    pmMergePenNot,
    pmMaskPenNot,
    pmMergeNotPen,
    pmMaskNotPen,
    pmMerge,
    pmNotMerge,
    pmMask,
    pmNotMask,
    pmXor,
    pmNotXor
  );
```

C++

```
  enum TPenMode {
    pmBlack,
    pmWhite,
    pmNop,
    pmNot,
    pmCopy,
    pmNotCopy,
    pmMergePenNot,
    pmMaskPenNot,
    pmMergeNotPen,
    pmMaskNotPen,
    pmMerge,
    pmNotMerge,
    pmMask,
    pmNotMask,
    pmXor,
    pmNotXor
  };
```

File

Graphics

Description

TPenMode indicates how a pen color interacts with the color of the canvas it is writing on. The following table lists the possible values:

Mode	Pixel color
pmBlack	Always black
pmWhite	Always white
pmNop	Unchanged
pmNot	Inverse of canvas background color
pmCopy	Pen color specified in Color property
pmNotCopy	Inverse of pen color
pmMergePenNot	Combination of pen color and inverse of canvas background
pmMaskPenNot	Combination of colors common to both pen and inverse of canvas background.
pmMergeNotPen	Combination of canvas background color and inverse of pen color
pmMaskNotPen	Combination of colors common to both canvas background and inverse of pen
pmMerge	Combination of pen color and canvas background color
pmNotMerge	Inverse of pmMerge: combination of pen color and canvas background color

pmMask	Combination of colors common to both pen and canvas background
pmNotMask	Inverse of pmMask: combination of colors common to both pen and canvas background
pmXor	Combination of colors in either pen or canvas background, but not both
pmNotXor	Inverse of pmXor: combination of colors in either pen or canvas background, but not both

73.4.15 Graphics.TPenStyle Enumeration

TPenStyle indicates the type of line a pen draws.

Pascal

```
TPenStyle = (
  psSolid,
  psDash,
  psDot,
  psDashDot,
  psDashDotDot,
  psClear,
  psInsideFrame,
  psUserStyle,
  psAlternate
);
```

C++

```
enum TPenStyle {
  psSolid,
  psDash,
  psDot,
  psDashDot,
  psDashDotDot,
  psClear,
  psInsideFrame,
  psUserStyle,
  psAlternate
};
```

File

Graphics

Description

TPenStyle can have one of the following values:

Value	Meaning
psSolid	A solid line.
psDash	A line made up of a series of dashes.
psDot	A line made up of a series of dots.
psDashDot	A line made up of alternating dashes and dots.
psDashDotDot	A line made up of a series of dash-dot-dot combinations.
psClear	No line is drawn (used to omit the line around shapes that draw an outline using the current pen).
psInsideFrame	A solid line, but one that may use a dithered color if Width is greater than 1.

psAlternate	The pen sets every other pixel. (This style is applicable only for cosmetic pens.) This style is only valid for pens created with the `ExtCreatePen` API function. (See MS Windows SDK docs.) This applies to both VCL and VCL.NET.
psUserStyle	The pen uses a styling array supplied by the user. Thus style is only valid for pens created with the `ExtCreatePen` API function. (See MS Windows SDK docs.) This applies to both VCL and VCL.NET.

73.4.16 Graphics.TPixelFormat Enumeration

TPixelFormat specifies the format of a bitmap.

Pascal

```
TPixelFormat = (
  pfDevice,
  pf1bit,
  pf4bit,
  pf8bit,
  pf15bit,
  pf16bit,
  pf24bit,
  pf32bit,
  pfCustom
);
```

C++

```
enum TPixelFormat {
  pfDevice,
  pf1bit,
  pf4bit,
  pf8bit,
  pf15bit,
  pf16bit,
  pf24bit,
  pf32bit,
  pfCustom
};
```

File

Graphics

Description

TPixelFormat can have one of the following values:

Value	Meaning
pfDevice	The bitmap is stored as a device-dependent bitmap.
pf1bit	The bitmap is a device-independent bitmap with one bit per pixel (black and white palette)
pf4bit	The bitmap is a device-independent bitmap that uses a 16-color palette.
pf8bit	The bitmap is a device-independent bitmap that uses a 256color palette.
pf15bit	The bitmap is a device-independent true-color bitmap that uses 15 bits per pixel (RGB compression).
pf16bit	The bitmap is a device-independent true-color bitmap that uses 16 bits per pixel (bitfield compression).
pf24bit	The bitmap is a device-independent true-color bitmap that uses 24 bits per pixel.

pf32bit	The bitmap is a device-independent true-color bitmap that uses 32 bits per pixel (RGB compression).
pfCustom	The bitmap uses some other format. TBitmap (see page 14396) does not support pfCustom.

73.4.17 Graphics.TProgressStage Enumeration

TProgressStage indicates the stage of a lengthy operation.

Pascal

```
TProgressStage = (
  psStarting,
  psRunning,
  psEnding
);
```

C++

```
enum TProgressStage {
  psStarting,
  psRunning,
  psEnding
};
```

File

Graphics

Description

The following table lists the possible values:

Value	Meaning
psStarting	The operation is about to begin.
psRunning	The operation is underway and has not yet completed.
psEnding	The operation has just completed.

73.4.18 Graphics.TResData Record

Pascal

```
TResData = record
  Handle: THandle;
end;
```

C++

```
struct TResData {
  THandle Handle;
};
```

File

Graphics

Description

This is record Graphics.TResData.

73.4.19 Graphics.TResource Record

Pascal

```
TResource = record
  Next: PResource;
  RefCount: Integer;
  Handle: THandle;
  HashCode: Word;
  case Integer of
    0: (Data: TResData;);
    1: (Font: TFontData;);
    2: (Pen: TPenData;);
    3: (Brush: TBrushData;);
end;
```

C++

```
struct TResource {
  PResource Next;
  int RefCount;
  THandle Handle;
  Word HashCode;
  TResData Data;
  TFontData Font;
  TPenData Pen;
  TBrushData Brush;
};
```

File

Graphics

Description

This is record Graphics.TResource.

73.4.20 Graphics.TTextFormats Enumeration

Pascal

```
TTextFormats = (
  tfBottom,
  tfCalcRect,
  tfCenter,
  tfEditControl,
  tfEndEllipsis,
  tfPathEllipsis,
  tfExpandTabs,
  tfExternalLeading,
  tfLeft,
  tfModifyString,
  tfNoClip,
  tfNoPrefix,
  tfRight,
  tfRtlReading,
```

```
    tfSingleLine,
    tfTop,
    tfVerticalCenter,
    tfWordBreak
  );
```

C++

```
  enum TTextFormats {
    tfBottom,
    tfCalcRect,
    tfCenter,
    tfEditControl,
    tfEndEllipsis,
    tfPathEllipsis,
    tfExpandTabs,
    tfExternalLeading,
    tfLeft,
    tfModifyString,
    tfNoClip,
    tfNoPrefix,
    tfRight,
    tfRtlReading,
    tfSingleLine,
    tfTop,
    tfVerticalCenter,
    tfWordBreak
  };
```

File

Graphics

Description

This is record Graphics.TTextFormats.

73.4.21 Graphics.TTransparentMode Enumeration

TTransparentMode indicates how the transparent color of a bitmap is determined.

Pascal

```
  TTransparentMode = (
    tmAuto,
    tmFixed
  );
```

C++

```
  enum TTransparentMode {
    tmAuto,
    tmFixed
  };
```

File

Graphics

Description

The following table lists the possible values:

Value	Meaning
tmAuto	The transparent color is determined by the bottom-leftmost pixel of the bitmap image.
tmFixed	The transparent color is explicitly assigned and stored in the bitmap object.

73.5 Types

The following table lists types in this documentation.

Types

Name	Description
HENHMETAFILE (see page 14645)	Meta file handle type.
HMETAFILE (see page 14646)	This is type Graphics.HMETAFILE.
PColor (see page 14646)	PColor is a pointer to a TColor (see page 14648) type.
PCursorOrIcon (see page 14646)	This is type Graphics.PCursorOrIcon.
PIconRec (see page 14647)	This is type Graphics.PIconRec.
PResource (see page 14647)	This is type Graphics.PResource.
TCanvasState (see page 14647)	This is type Graphics.TCanvasState.
TColor (see page 14648)	TColor is used to specify the color of a Windows-only control.
TCopyMode (see page 14650)	TCopyMode specifies how a graphical image is copied onto a canvas.
TFontCharset (see page 14651)	TFontCharset represents the character set of a font.
TFontDataName (see page 14652)	This is type Graphics.TFontDataName.
TFontName (see page 14653)	TFontName is a string that holds the typeface name of a font.
TFontStyles (see page 14653)	TFontStyle (see page 14636) and TFontStyles specify style characteristics of a font.
TFontStylesBase (see page 14653)	This is type Graphics.TFontStylesBase.
TGraphicClass (see page 14654)	TGraphicClass defines the metaclass for TGraphic (see page 14516).
TProgressEvent (see page 14655)	TProgressEvent is the type of event handlers that provide feedback during lengthy operations.
TTextFormat (see page 14656)	TTextFormat indicates how a text viewer control displays its contents.

73.5.1 Graphics.HENHMETAFILE Type

Meta file handle type.

Pascal

```
HENHMETAFILE = THandle;
```

C++

```
THandle HENHMETAFILE;
```

File

Graphics

Description

ReleaseHandle is called to release the resources used to represent the metafile. HENHMETAFILE is the data type passed to that method.

See Also

ReleaseHandle

73.5.2 Graphics.HMETAFILE Type

Pascal

```
HMETAFILE = THandle;
```

C++

```
THandle HMETAFILE;
```

File

Graphics

Description

This is type Graphics.HMETAFILE.

73.5.3 Graphics.PColor Type

PColor is a pointer to a TColor (see page 14648) type.

Pascal

```
PColor = ^TColor;
```

C++

```
^TColor PColor;
```

File

Graphics

Description

The TColor (see page 14648) type represents a color as an integer.

Note: Static arrays are declared with an immutable size.

See Also

TColor (see page 14648)

73.5.4 Graphics.PCursorOrIcon Type

Pascal

```
PCursorOrIcon = ^TCursorOrIcon;
```

C++

```
^TCursorOrIcon PCursorOrIcon;
```

File

Graphics

Description

This is type Graphics.PCursorOrIcon.

73.5.5 Graphics.PIconRec Type

Pascal

```
PIconRec = ^TIconRec;
```

C++

```
^TIconRec PIconRec;
```

File

Graphics

Description

This is type Graphics.PIconRec.

73.5.6 Graphics.PResource Type

Pascal

```
PResource = ^TResource;
```

C++

```
^TResource PResource;
```

File

Graphics

Description

This is type Graphics.PResource.

73.5.7 Graphics.TCanvasState Type

Pascal

```
TCanvasState = set of TCanvasStates;
```

C++

```
TCanvasStates TCanvasState;
```

File

Graphics

Description

This is type Graphics.TCanvasState.

73.5.8 Graphics.TColor Type

TColor is used to specify the color of a Windows-only control.

Pascal

```
TColor = -$7FFFFFFF-1..$7FFFFFFF;
```

C++

```
-$7FFFFFFF-1..$7FFFFFFF TColor;
```

File

Graphics

Description

TColor is used to specify the color of a Windows-only control. It is used by the Color property of many components and by a number of other properties that specify color values.

The Graphics (see page 14387) unit contains definitions of useful constants for TColor. These constants map either directly to the closest matching color in the system palette (for example, clBlue (see page 14666) maps to blue) or to the corresponding system screen element color defined in the Color section of the Windows Control panel (for example, clBtnFace (see page 14667) maps to the system color for button faces).

If you specify TColor as a specific 4-byte hexadecimal number instead of using the constants defined in the Graphics (see page 14387) unit, the low three bytes represent RGB color intensities for blue, green, and red, respectively. The value $00FF0000 (Delphi) or 0x00FF0000 (C++) represents full-intensity, pure blue, $0000FF00 (Delphi) or 0x0000FF00 (C++) is pure green, and $000000FF (Delphi) or 0x000000FF (C++) is pure red. $00000000 (Delphi) or 0x00000000 (C++) is black and $00FFFFFF (Delphi) or 0x00FFFFFF (C++) is white.

If the highest-order byte is zero, the color obtained is the closest matching color in the system palette. If the highest-order byte is one ($01 or 0x01), the color obtained is the closest matching color in the currently realized palette. If the highest-order byte is two ($02 or 0x02), the value is matched with the nearest color in the logical palette of the current device context.

The following tables lists the color constants from the Graphics (see page 14387) unit. The first two columns list the colors that map to the closest matching color in the system palette, while the last two columns list the colors that are defined in the Windows Control panel.

Value	Meaning	Value	Meaning
clNone (see page 14681)	White on Windows 9x, Black on NT.	clScrollBar (see page 14683)	Current color for the of scroll bar track.
clAqua (see page 14665)	Aqua	clBackground (see page 14666)	Current background color of the Windows desktop
clBlack (see page 14666)	Black	clActiveCaption (see page 14664)	Current color of the title bar of the active window
clBlue (see page 14666)	Blue	clInactiveCaption (see page 14675)	Current color of the title bar of inactive windows

clCream (see page 14669)	Cream	clMenu (see page 14679)	Current background color of menus
clDkGray (see page 14670)	Dark Gray	clWindow (see page 14732)	Current background color of windows
clFuchsia (see page 14670)	Fuchsia	clWindowFrame (see page 14733)	Current color of window frames
clGray (see page 14672)	Gray	clMenuText (see page 14680)	Current color of text on menus
clGreen (see page 14673)	Green	clWindowText (see page 14733)	Current color of text in windows
clLime (see page 14677)	Lime green	clCaptionText (see page 14669)	Current color of the text on the title bar of the active window
clLtGray (see page 14677)	Light Gray	clActiveBorder (see page 14664)	Current border color of the active window
clMaroon (see page 14678)	Maroon	clInactiveBorder (see page 14675)	Current border color of inactive windows
clMedGray (see page 14678)	Medium Gray	clAppWorkSpace (see page 14665)	Current color of the application workspace
clMoneyGreen (see page 14680)	Mint green	clHighlight (see page 14673)	Current background color of selected text
clNavy (see page 14681)	Navy blue	clHightlightText	Current color of selected text
clOlive (see page 14682)	Olive green	clBtnFace (see page 14667)	Current color of a button face
clPurple (see page 14682)	Purple	clBtnShadow (see page 14668)	Current color of a shadow cast by a button
clRed (see page 14683)	Red	clGrayText (see page 14672)	Current color of text that is dimmed
clSilver (see page 14683)	Silver	clBtnText (see page 14668)	Current color of text on a button
clSkyBlue (see page 14684)	Sky blue	clInactiveCaptionText (see page 14675)	Current color of the text on the title bar of an inactive window
clTeal (see page 14685)	Teal	clBtnHighlight (see page 14667)	Current color of the highlighting on a button

clWhite (see page 14732)	White	cl3DDkShadow (see page 14663)	Windows 95 or NT 4.0 only: Dark shadow for three-dimensional display elements
clYellow (see page 14734)	Yellow	cl3DLight (see page 14663)	Windows 95 or NT 4.0 only: Light color for three-dimensional display elements (for edges facing the light source)
		clInfoText (see page 14676)	Windows 95 or NT 4.0 only: Text color for tool tip controls
		clInfoBk (see page 14676)	Windows 95 or NT 4.0 only: Background color for tool tip controls
		clGradientActiveCaption (see page 14671)	Windows 98 or Windows 2000: Right side color in the color gradient of an active window's title bar. clActiveCaption (see page 14664) specifies the left side color.
		clGradientInactiveCaption (see page 14671)	Windows 98 or Windows 2000: Right side color in the color gradient of an inactive window's title bar. clInactiveCaption (see page 14675) specifies the left side color.
		clDefault (see page 14670)	The default color for the control to which the color is assigned.

73.5.9 Graphics.TCopyMode Type

TCopyMode specifies how a graphical image is copied onto a canvas.

Pascal

```
TCopyMode = Longint;
```

C++

```
Longint TCopyMode;
```

File

Graphics

Description

TCopyMode values describe how to combine the colors of a source bitmap and a destination bitmap.

The Windows unit defines the following constants for TCopyMode values:

Value	Meaning
cmBlackness (see page 14734)	Fills the destination rectangle on the canvas with black.
cmDstInvert (see page 14735)	Inverts the image on the canvas and ignores the source.
cmMergeCopy (see page 14735)	Combines the image on the canvas and the source bitmap by using the Boolean AND operator.
cmMergePaint (see page 14736)	Combines the inverted source bitmap with the image on the canvas by using the Boolean OR operator.

cmNotSrcCopy (see page 14736)	Copies the inverted source bitmap to the canvas.
cmNotSrcErase (see page 14737)	Combines the image on the canvas and the source bitmap by using the Boolean OR operator, and inverts the result.
cmPatCopy (see page 14737)	Copies the source pattern to the canvas.
cmPatInvert (see page 14738)	Combines the source pattern with the image on the canvas using the Boolean XOR operator
cmPatPaint (see page 14738)	Combines the inverted source bitmap with the source pattern by using the Boolean OR operator. Combines the result of this operation with the image on the canvas by using the Boolean OR operator.
cmSrcAnd (see page 14739)	Combines the image on the canvas and source bitmap by using the Boolean AND operator.
cmSrcCopy (see page 14739)	Copies the source bitmap to the canvas.
cmSrcErase (see page 14740)	Inverts the image on the canvas and combines the result with the source bitmap by using the Boolean AND operator.
cmSrcInvert (see page 14740)	Combines the image on the canvas and the source bitmap by using the Boolean XOR operator.
cmSrcPaint (see page 14741)	Combines the image on the canvas and the source bitmap by using the Boolean OR operator.
cmWhiteness (see page 14741)	Fills the destination rectangle on the canvas with white.

73.5.10 Graphics.TFontCharset Type

TFontCharset represents the character set of a font.

Pascal

```
TFontCharset = 0..255;
```

C++

```
0..255 TFontCharset;
```

File

Graphics

Description

Each Windows typeface supports one or more character sets, which indicate what characters can be written using the font. For

example, most fonts that can represent japanese characters use a particular multi-byte character set.

Each font supports its own, unique set of character sets. Check with the font vendor to determine which character sets are supported.

The following table lists the predefined constants provided for standard character sets:

Constant	Value	Description
ANSI_CHARSET	0	ANSI characters.
DEFAULT_CHARSET	1	Font is chosen based solely on Name and Size. If the described font is not available on the system, Windows will substitute another font.
SYMBOL_CHARSET	2	Standard symbol set.
MAC_CHARSET	77	Macintosh characters. Not available on NT 3.51.
SHIFTJIS_CHARSET	128	Japanese shift-JIS characters.
HANGEUL_CHARSET	129	Korean characters (Wansung).
JOHAB_CHARSET	130	Korean characters (Johab). Not available on NT 3.51
GB2312_CHARSET	134	Simplified Chinese characters (mainland china).
CHINESEBIG5_CHARSET	136	Traditional Chinese characters (Taiwanese).
GREEK_CHARSET	161	Greek characters. Not available on NT 3.51.
TURKISH_CHARSET	162	Turkish characters. Not available on NT 3.51
VIETNAMESE_CHARSET	163	Vietnamese characters. Not available on NT 3.51.
HEBREW_CHARSET	177	Hebrew characters. Not available on NT 3.51
ARABIC_CHARSET	178	Arabic characters. Not available on NT 3.51
BALTIC_CHARSET	186	Baltic characters. Not available on NT 3.51.
RUSSIAN_CHARSET	204	Cyrillic characters. Not available on NT 3.51.
THAI_CHARSET	222	Thai characters. Not available on NT 3.51
EASTEUROPE_CHARSET	238	Includes diacritical marks for eastern european countries. Not available on NT 3.51.
OEM_CHARSET	255	Depends on the codepage of the operating system.

73.5.11 Graphics.TFontDataName Type

Pascal

```
TFontDataName = string[LF_FACESIZE - 1];
```

C++

```
string[LF_FACESIZE - 1] TFontDataName;
```

File

Graphics

Description

This is type Graphics.TFontDataName.

73.5.12 Graphics.TFontName Type

TFontName is a string that holds the typeface name of a font.

Pascal

```
TFontName = type string;
```

C++

```
type string TFontName;
```

File

Graphics

Description

TFontName maps to a string. It is the type for representing the typeface name of a font.

73.5.13 Graphics.TFontStyles Type

TFontStyle (see page 14636) and TFontStyles specify style characteristics of a font.

Pascal

```
TFontStyles = set of TFontStyle;
```

C++

```
TFontStyle TFontStyles;
```

File

Graphics

Description

TFontStyles is a set of TFontStyle (see page 14636) values. The following table lists the possible values of the TFontStyle (see page 14636) type:

Value	Meaning
fsBold	The font is boldfaced.
fsItalic	The font is italicized.
fsUnderline	The font is underlined.
fsStrikeOut	The font is displayed with a horizontal line through it.

73.5.14 Graphics.TFontStylesBase Type

Pascal

```
TFontStylesBase = set of TFontStyle;
```

C++

```
TFontStyle TFontStylesBase;
```

File

Graphics

Description

This is type Graphics.TFontStylesBase.

73.5.15 Graphics.TGraphicClass Type

TGraphicClass defines the metaclass for TGraphic (see page 14516).

Pascal

```
TGraphicClass = class of TGraphic;
```

C++

```
TGraphic TGraphicClass;
```

File

Graphics

Description

TGraphicClass is the metaclass for TGraphic (see page 14516). Its value is the class reference for TGraphic (see page 14516) or for one of its descendants.

C++ Examples:

```
/*
The following example allows users to type the name of a
graphics class into an edit control and when a button is
pressed, the name of the default extension for that class is
displayed in another edit control.  In the form's OnCreate
event handler (or a similar place) you must register the
graphics classes so that GetClass can find them.
*/
TMetaClass *MetaClass;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClassFinder *myClassFinder = new TClassFinder(MetaClass, False);
  TGraphicClass mygraphclass = (TGraphicClass) (myClassFinder->GetClass(Edit2->Text));
//  TGraphicClass mygraphclass = (TGraphicClass) GetClass(Edit2->Text);
  Edit1->Text = GraphicExtension(mygraphclass);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MetaClass = __classid(TIcon); // ico
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(Graphics::TBitmap); // bmp
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TMetafile); // emf
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TShape);  // nographic extension
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TImage); // nographic extension
  RegisterClasses(&MetaClass, 0);
```

```
}
```

Delphi Examples:

```
{
The following example allows users to type the name of a
graphics class into an edit control and when a button is
pressed, the name of the default extension for that class is
displayed in another edit control.  In the form's OnCreate
event handler (or a similar place) you must register the
graphics classes so that GetClass can find them.
}

uses ExtCtrls;

procedure TForm1.Button1Click(Sender: TObject);
var
  myClassFinder: TClassFinder;
  mygraphclass: TGraphicClass;
begin
  { make sure the classes are registered so that GetClass will work -- }
  { Usually, this goes in the initialization section where it is only executed once }
  RegisterClasses([TIcon, TBitmap, TButton, TShape, TImage, TMetafile]);
  try
    myClassFinder := TClassFinder.Create(TIcon, False);
    mygraphclass := TGraphicClass(myClassFinder.GetClass(Edit2.Text));
//    mygraphclass := TGraphicClass(GetClass(Edit2.Text));  // works as well
    Edit1.Text := GraphicExtension(mygraphclass);
  except
    on EAccessViolation do
      MessageDlg('Invalid class name.', mtError, [mbOK], 0)
  end;
end;
```

73.5.16 Graphics.TProgressEvent Type

TProgressEvent is the type of event handlers that provide feedback during lengthy operations.

Pascal

```
TProgressEvent = procedure (Sender: TObject; Stage: TProgressStage; PercentDone: Byte;
RedrawNow: Boolean; const R: TRect; const Msg: string) of object;
```

C++

```
(Sender: TObject; Stage: TProgressStage; PercentDone: Byte; RedrawNow: Boolean; const R:
TRect; const Msg: string) ( TProgressEvent)();
```

File

Graphics

Description

Sender is the object that receives the event notification. This is typically the component that initiated the lengthy operation.

Stage indicates whether the operation is beginning, continuing, or ending. If the event handler displays an indicator such as a progress bar, the indicator can be created when Stage is psStarting, updated while Stage is psRunning, and removed when Stage is psEnding.

PercentDone is an approximation of how much of the operation has completed. Use PercentDone to update the position of a progress bar or other indicator.

RedrawNow indicates whether the partial results of the operation can safely be drawn on screen at this point.

R gives the dimensions of the portion of the display that has changed and needs to be redrawn.

Msg contains one or two words that describe what operation is occurring. For example, the value of Msg could be a string such as Loading, Storing, or Reducing colors. The Msg string can also be empty.

73.5.17 Graphics.TTextFormat Type

TTextFormat indicates how a text viewer control displays its contents.

Pascal

```
TTextFormat = set of TTextFormats;
```

C++

```
TTextFormats TTextFormat;
```

File

Graphics

Description

TTextFormat has the following values:

Value	Meaning
tfAutoText	The control attempts to determine whether Text contains rich text. If a valid tag is seen on the first line, Text is assumed to be a rich text document. Otherwise, Text is displayed as plain text.
tfText	The control assumes that Text contains rich text, even if no tags are present. Note that in rich text, any sequence consisting entirely of spaces, tabs, and line breaks counts as a single space.
tfPlainText	The control assumes that Text contains plain text.

73.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
DDBsOnly (see page 14657)	Determines whether bitmap handles are created as device-dependent bitmaps (DDB).
DefFontData (see page 14658)	This is variable Graphics.DefFontData.
SystemPalette16 (see page 14658)	Contains the 16 VGA system colors.

73.6.1 Graphics.DDBsOnly Variable

Determines whether bitmap handles are created as device-dependent bitmaps (DDB).

Pascal

```
DDBsOnly: Boolean = False;
```

C++

```
Boolean DDBsOnly = False;
```

File

Graphics

Description

Set DDBsOnly to force all bitmap handles to be created as device-dependent bitmaps (DDB). The default bitmap handle format is DIB (device-independent bitmap).

Setting DDBsOnly to true undercuts much of the graphics architecture; it should not be used unless it is absolutely required. Examples of when to set it to true are:

A graphics-intensive application does bitmap operations that Windows does not support with DIBs; you can either set each bitmap object to DDB before you access its handle property, or you can set this global variable to Truetrue.

An application uses hundreds of very small bitmaps. Each DIB consumes a separate 4k memory page. DIBs are less efficient for storing many small images than DDBs. Before resorting to changing the DDBsOnly flag to address this, however, consider switching your code to use image lists instead of individual tiny bitmaps. Image lists are more efficient than DIBs and DDBs for storing many same-size small images.

Note: Setting DDBsOnly to true introduces the issue of image alteration. When a BMP file or resource is loaded into a DDB, the color information in the original image is changed to fit the pixel format of the current video device and mode. If you read an image, touch its handle, and write it out, you will not get the same file you read in. DIBs solve this problem. Do not change DDBsOnly unless you have a specific need for device-dependent bitmaps and are willing to accept the image-altering DDB side effects.

See Also

GetDIB (see page 14620)

TImageList

73.6.2 Graphics.DefFontData Variable

Pascal

```
DefFontData: TFontData = ( Handle: 0; Height: 0; Orientation: 0; Pitch: fpDefault; Style: [];
Charset : DEFAULT_CHARSET; Name: 'MS Sans Serif');
```

C++

```
TFontData DefFontData = ( Handle: 0; Height: 0; Orientation: 0; Pitch: fpDefault; Style: [];
Charset : DEFAULT_CHARSET; Name: 'MS Sans Serif');
```

File

Graphics

Description

This is variable Graphics.DefFontData.

73.6.3 Graphics.SystemPalette16 Variable

Contains the 16 VGA system colors.

Pascal

```
SystemPalette16: HPalette;
```

C++

```
HPalette SystemPalette16;
```

File

Graphics

Description

SystemPalette16 holds a palette handle that contains the 16 VGA system colors. This palette is built at runtime from the system colors defined by the current video card and driver. The actual color values of the stock VGA palette can vary from one video card vendor to the next. Therefore it is not safe to assume, for example, that clGray (see page 14672) will be RGB(128,128,128) on all systems.

This handle is available as a convenience. This is the palette that VCL forces on all 16 color bitmaps that are loaded from files or resources.

Note: Using custom 16-color palettes does not work on 16-color video systems. It is problematic on many 256 color drivers because the variation in colors interferes with transparency operations.

See Also

TGraphic (see page 14516)

Graphics

73.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
cl3DDkShadow (see page 14663)	Represents the dark shadow of 3D components.
cl3DLight (see page 14663)	Represents the light part of 3D components.
clActiveBorder (see page 14664)	Represents the default border color of an active window.
clActiveCaption (see page 14664)	Represents the default color of the title bar of an active window.
clAppWorkSpace (see page 14665)	Represents the default Application WorkSpace color.
clAqua (see page 14665)	Represents the color Aqua.
clBackground (see page 14666)	Represents the default Windows background color.
clBlack (see page 14666)	Represents the color black.
clBlue (see page 14666)	Represents the color blue.
clBtnFace (see page 14667)	Represents the color of the face of a button.
clBtnHighlight (see page 14667)	Represents the color of the highlighting on a button.
clBtnShadow (see page 14668)	Represents the color of the shadow cast by of a button.
clBtnText (see page 14668)	Represents the default color of the text on a button.
clCaptionText (see page 14669)	Represents the default color of the text on the title bar of an active window.
clCream (see page 14669)	Represents the color cream.
clDefault (see page 14670)	Specifies the default color of a component.
clDkGray (see page 14670)	Represents the color dark gray.
clFuchsia (see page 14670)	Represents the color fuchsia.
clGradientActiveCaption (see page 14671)	Right side color in the color gradient of an active window's title bar.
clGradientInactiveCaption (see page 14671)	Right side color in the color gradient of an inactive window's title bar.
clGray (see page 14672)	Represents the color gray.
clGrayText (see page 14672)	Represents the color of text that is grayed out.
clGreen (see page 14673)	Represents the color green.
clHighlight (see page 14673)	Represents the color of the background of a block of selected text.
clHighlightText (see page 14674)	Represents the color of selected text.
clHotLight (see page 14674)	This is constant Graphics.clHotLight.
clInactiveBorder (see page 14675)	Represents the default border color of an inactive window.
clInactiveCaption (see page 14675)	Represents the default color of the title bar of an inactive window.
clInactiveCaptionText (see page 14675)	Represents the default color of the title text in an inactive window.
clInfoBk (see page 14676)	Represents the color of the background of a tool tip.
clInfoText (see page 14676)	Represents the color of tool tip text.
clLime (see page 14677)	Represents the color lime.
clLtGray (see page 14677)	Represents the color light gray.
clMaroon (see page 14678)	Represents the color maroon.
clMedGray (see page 14678)	Represents the color medium gray.
clMenu (see page 14679)	Represents the color of the background of an unselected menu sub item.
clMenuBar (see page 14679)	Represents the color of the background of an unselected main menu item.
clMenuHighlight (see page 14679)	Represents the color of the background of a selected menu item.
clMenuText (see page 14680)	Represents the default menu text color.
clMoneyGreen (see page 14680)	Represents the color money green.
clNavy (see page 14681)	Represents the color navy.
clNone (see page 14681)	Represents no selected color.
clOlive (see page 14682)	Represents the color olive.
clPurple (see page 14682)	Represents the color purple.

clRed (see page 14683)	Represents the color red.
clScrollBar (see page 14683)	This is constant Graphics.clScrollBar.
clSilver (see page 14683)	Represents the color silver.
clSkyBlue (see page 14684)	Represents the color sky blue.
clSystemColor (see page 14684)	Is a mask used to identify the set of system colors.
clTeal (see page 14685)	Represents the color teal.
clWebAliceBlue (see page 14685)	This is constant Graphics.clWebAliceBlue.
clWebAntiqueWhite (see page 14686)	This is constant Graphics.clWebAntiqueWhite.
clWebAqua (see page 14686)	This is constant Graphics.clWebAqua.
clWebAquamarine (see page 14686)	This is constant Graphics.clWebAquamarine.
clWebAzure (see page 14687)	This is constant Graphics.clWebAzure.
clWebBeige (see page 14687)	This is constant Graphics.clWebBeige.
clWebBisque (see page 14687)	This is constant Graphics.clWebBisque.
clWebBlack (see page 14688)	This is constant Graphics.clWebBlack.
clWebBlanchedAlmond (see page 14688)	This is constant Graphics.clWebBlanchedAlmond.
clWebBlue (see page 14688)	This is constant Graphics.clWebBlue.
clWebBlueViolet (see page 14689)	This is constant Graphics.clWebBlueViolet.
clWebBrown (see page 14689)	This is constant Graphics.clWebBrown.
clWebBurlywood (see page 14689)	This is constant Graphics.clWebBurlywood.
clWebCadetBlue (see page 14690)	This is constant Graphics.clWebCadetBlue.
clWebChartreuse (see page 14690)	This is constant Graphics.clWebChartreuse.
clWebChocolate (see page 14690)	This is constant Graphics.clWebChocolate.
clWebCoral (see page 14691)	This is constant Graphics.clWebCoral.
clWebCornFlowerBlue (see page 14691)	This is constant Graphics.clWebCornFlowerBlue.
clWebCornSilk (see page 14691)	This is constant Graphics.clWebCornSilk.
clWebCrimson (see page 14692)	This is constant Graphics.clWebCrimson.
clWebCyan (see page 14692)	This is constant Graphics.clWebCyan.
clWebDarkBlue (see page 14692)	This is constant Graphics.clWebDarkBlue.
clWebDarkCyan (see page 14693)	This is constant Graphics.clWebDarkCyan.
clWebDarkGoldenRod (see page 14693)	This is constant Graphics.clWebDarkGoldenRod.
clWebDarkGray (see page 14693)	This is constant Graphics.clWebDarkGray.
clWebDarkgreen (see page 14694)	This is constant Graphics.clWebDarkgreen.
clWebDarkKhaki (see page 14694)	This is constant Graphics.clWebDarkKhaki.
clWebDarkMagenta (see page 14694)	This is constant Graphics.clWebDarkMagenta.
clWebDarkOliveGreen (see page 14695)	This is constant Graphics.clWebDarkOliveGreen.
clWebDarkOrange (see page 14695)	This is constant Graphics.clWebDarkOrange.
clWebDarkOrchid (see page 14695)	This is constant Graphics.clWebDarkOrchid.
clWebDarkRed (see page 14696)	This is constant Graphics.clWebDarkRed.
clWebDarkSalmon (see page 14696)	This is constant Graphics.clWebDarkSalmon.
clWebDarkSeaGreen (see page 14696)	This is constant Graphics.clWebDarkSeaGreen.
clWebDarkSlateBlue (see page 14697)	This is constant Graphics.clWebDarkSlateBlue.
clWebDarkSlategray (see page 14697)	This is constant Graphics.clWebDarkSlategray.
clWebDarkTurquoise (see page 14697)	This is constant Graphics.clWebDarkTurquoise.
clWebDarkViolet (see page 14698)	This is constant Graphics.clWebDarkViolet.
clWebDeepPink (see page 14698)	This is constant Graphics.clWebDeepPink.
clWebDeepskyBlue (see page 14698)	This is constant Graphics.clWebDeepskyBlue.
clWebDimGray (see page 14699)	This is constant Graphics.clWebDimGray.
clWebDodgerBlue (see page 14699)	This is constant Graphics.clWebDodgerBlue.
clWebFirebrick (see page 14699)	This is constant Graphics.clWebFirebrick.
clWebFloralWhite (see page 14700)	This is constant Graphics.clWebFloralWhite.
clWebForestGreen (see page 14700)	This is constant Graphics.clWebForestGreen.
clWebFuchsia (see page 14700)	This is constant Graphics.clWebFuchsia.
clWebGainsboro (see page 14701)	This is constant Graphics.clWebGainsboro.
clWebGhostWhite (see page 14701)	This is constant Graphics.clWebGhostWhite.

clWebGold (see page 14701)	This is constant Graphics.clWebGold.
clWebGoldenRod (see page 14702)	This is constant Graphics.clWebGoldenRod.
clWebGray (see page 14702)	This is constant Graphics.clWebGray.
clWebGreen (see page 14702)	This is constant Graphics.clWebGreen.
clWebGreenYellow (see page 14703)	This is constant Graphics.clWebGreenYellow.
clWebHoneydew (see page 14703)	This is constant Graphics.clWebHoneydew.
clWebHotPink (see page 14703)	This is constant Graphics.clWebHotPink.
clWebIndianRed (see page 14704)	This is constant Graphics.clWebIndianRed.
clWebIndigo (see page 14704)	This is constant Graphics.clWebIndigo.
clWebIvory (see page 14704)	This is constant Graphics.clWebIvory.
clWebKhaki (see page 14705)	This is constant Graphics.clWebKhaki.
clWebLavender (see page 14705)	This is constant Graphics.clWebLavender.
clWebLavenderBlush (see page 14705)	This is constant Graphics.clWebLavenderBlush.
clWebLawnGreen (see page 14706)	This is constant Graphics.clWebLawnGreen.
clWebLemonChiffon (see page 14706)	This is constant Graphics.clWebLemonChiffon.
clWebLightBlue (see page 14706)	This is constant Graphics.clWebLightBlue.
clWebLightCoral (see page 14707)	This is constant Graphics.clWebLightCoral.
clWebLightCyan (see page 14707)	This is constant Graphics.clWebLightCyan.
clWebLightGoldenrodYellow (see page 14707)	This is constant Graphics.clWebLightGoldenrodYellow.
clWebLightGreen (see page 14708)	This is constant Graphics.clWebLightGreen.
clWebLightgrey (see page 14708)	This is constant Graphics.clWebLightgrey.
clWebLightPink (see page 14708)	This is constant Graphics.clWebLightPink.
clWebLightSalmon (see page 14709)	This is constant Graphics.clWebLightSalmon.
clWebLightSeaGreen (see page 14709)	This is constant Graphics.clWebLightSeaGreen.
clWebLightSkyBlue (see page 14709)	This is constant Graphics.clWebLightSkyBlue.
clWebLightSlateGray (see page 14710)	This is constant Graphics.clWebLightSlateGray.
clWebLightSteelBlue (see page 14710)	This is constant Graphics.clWebLightSteelBlue.
clWebLightYellow (see page 14710)	This is constant Graphics.clWebLightYellow.
clWebLime (see page 14711)	This is constant Graphics.clWebLime.
clWebLimeGreen (see page 14711)	This is constant Graphics.clWebLimeGreen.
clWebLinen (see page 14711)	This is constant Graphics.clWebLinen.
clWebMagenta (see page 14712)	This is constant Graphics.clWebMagenta.
clWebMaroon (see page 14712)	This is constant Graphics.clWebMaroon.
clWebMediumAquamarine (see page 14712)	This is constant Graphics.clWebMediumAquamarine.
clWebMediumBlue (see page 14713)	This is constant Graphics.clWebMediumBlue.
clWebMediumOrchid (see page 14713)	This is constant Graphics.clWebMediumOrchid.
clWebMediumPurple (see page 14713)	This is constant Graphics.clWebMediumPurple.
clWebMediumSeaGreen (see page 14714)	This is constant Graphics.clWebMediumSeaGreen.
clWebMediumSlateBlue (see page 14714)	This is constant Graphics.clWebMediumSlateBlue.
clWebMediumSpringGreen (see page 14714)	This is constant Graphics.clWebMediumSpringGreen.
clWebMediumTurquoise (see page 14715)	This is constant Graphics.clWebMediumTurquoise.
clWebMediumVioletRed (see page 14715)	This is constant Graphics.clWebMediumVioletRed.
clWebMidnightBlue (see page 14715)	This is constant Graphics.clWebMidnightBlue.
clWebMintcream (see page 14716)	This is constant Graphics.clWebMintcream.
clWebMistyRose (see page 14716)	This is constant Graphics.clWebMistyRose.
clWebMoccasin (see page 14716)	This is constant Graphics.clWebMoccasin.
clWebNavajoWhite (see page 14717)	This is constant Graphics.clWebNavajoWhite.
clWebNavy (see page 14717)	This is constant Graphics.clWebNavy.
clWebOldLace (see page 14717)	This is constant Graphics.clWebOldLace.
clWebOlive (see page 14718)	This is constant Graphics.clWebOlive.
clWebOliveDrab (see page 14718)	This is constant Graphics.clWebOliveDrab.
clWebOrange (see page 14718)	This is constant Graphics.clWebOrange.
clWebOrangeRed (see page 14719)	This is constant Graphics.clWebOrangeRed.
clWebOrchid (see page 14719)	This is constant Graphics.clWebOrchid.

clWebPaleGoldenrod (see page 14719)	This is constant Graphics.clWebPaleGoldenrod.
clWebPaleGreen (see page 14720)	This is constant Graphics.clWebPaleGreen.
clWebPaleTurquoise (see page 14720)	This is constant Graphics.clWebPaleTurquoise.
clWebPaleVioletRed (see page 14720)	This is constant Graphics.clWebPaleVioletRed.
clWebPapayaWhip (see page 14721)	This is constant Graphics.clWebPapayaWhip.
clWebPeachPuff (see page 14721)	This is constant Graphics.clWebPeachPuff.
clWebPeru (see page 14721)	This is constant Graphics.clWebPeru.
clWebPink (see page 14722)	This is constant Graphics.clWebPink.
clWebPlum (see page 14722)	This is constant Graphics.clWebPlum.
clWebPowderBlue (see page 14722)	This is constant Graphics.clWebPowderBlue.
clWebPurple (see page 14723)	This is constant Graphics.clWebPurple.
clWebRed (see page 14723)	This is constant Graphics.clWebRed.
clWebRosyBrown (see page 14723)	This is constant Graphics.clWebRosyBrown.
clWebRoyalBlue (see page 14724)	This is constant Graphics.clWebRoyalBlue.
clWebSaddleBrown (see page 14724)	This is constant Graphics.clWebSaddleBrown.
clWebSalmon (see page 14724)	This is constant Graphics.clWebSalmon.
clWebSandyBrown (see page 14725)	This is constant Graphics.clWebSandyBrown.
clWebSeaGreen (see page 14725)	This is constant Graphics.clWebSeaGreen.
clWebSeashell (see page 14725)	This is constant Graphics.clWebSeashell.
clWebSienna (see page 14726)	This is constant Graphics.clWebSienna.
clWebSilver (see page 14726)	This is constant Graphics.clWebSilver.
clWebSkyBlue (see page 14726)	This is constant Graphics.clWebSkyBlue.
clWebSlateBlue (see page 14727)	This is constant Graphics.clWebSlateBlue.
clWebSlateGray (see page 14727)	This is constant Graphics.clWebSlateGray.
clWebSnow (see page 14727)	This is constant Graphics.clWebSnow.
clWebSpringGreen (see page 14728)	This is constant Graphics.clWebSpringGreen.
clWebSteelBlue (see page 14728)	This is constant Graphics.clWebSteelBlue.
clWebTan (see page 14728)	This is constant Graphics.clWebTan.
clWebTeal (see page 14729)	This is constant Graphics.clWebTeal.
clWebThistle (see page 14729)	This is constant Graphics.clWebThistle.
clWebTomato (see page 14729)	This is constant Graphics.clWebTomato.
clWebTurquoise (see page 14730)	This is constant Graphics.clWebTurquoise.
clWebViolet (see page 14730)	This is constant Graphics.clWebViolet.
clWebWheat (see page 14730)	This is constant Graphics.clWebWheat.
clWebWhite (see page 14731)	This is constant Graphics.clWebWhite.
clWebWhiteSmoke (see page 14731)	This is constant Graphics.clWebWhiteSmoke.
clWebYellow (see page 14731)	This is constant Graphics.clWebYellow.
clWebYellowGreen (see page 14732)	This is constant Graphics.clWebYellowGreen.
clWhite (see page 14732)	Represents the color white.
clWindow (see page 14732)	Represents the default Window background color.
clWindowFrame (see page 14733)	Represents the default Window frame background color.
clWindowText (see page 14733)	Represents the default Window text color.
clYellow (see page 14734)	Represents the color yellow.
cmBlackness (see page 14734)	TCopyMode (see page 14650) value : fills the destination rectangle on the canvas with black.
cmDstInvert (see page 14735)	TCopyMode (see page 14650) value : inverts the image on the canvas and ignores the source.
cmMergeCopy (see page 14735)	TCopyMode (see page 14650) value : combines the image on the canvas and the source bitmap by using the Boolean AND operator.
cmMergePaint (see page 14736)	TCopyMode (see page 14650) value : combines the inverted source bitmap with the image on the canvas by using the Boolean OR operator.
cmNotSrcCopy (see page 14736)	TCopyMode (see page 14650) value : copies the inverted source bitmap to the canvas.
cmNotSrcErase (see page 14737)	TCopyMode (see page 14650) value : combines the image on the canvas and the source bitmap by using the Boolean OR operator, and inverts the result.
cmPatCopy (see page 14737)	TCopyMode (see page 14650) value : copies the source pattern to the canvas.

cmPatInvert (see page 14738)	TCopyMode (see page 14650) value : combines the source pattern with the image on the canvas using the Boolean XOR operator.
cmPatPaint (see page 14738)	TCopyMode (see page 14650) value : combines the inverted source bitmap with the source pattern by using the Boolean OR operator. Combines the result of this operation with the image on the canvas by using the Boolean OR operator.
cmSrcAnd (see page 14739)	TCopyMode (see page 14650) value : combines the image on the canvas and source bitmap by using the Boolean AND operator.
cmSrcCopy (see page 14739)	TCopyMode (see page 14650) value : copies the source bitmap to the canvas.
cmSrcErase (see page 14740)	TCopyMode (see page 14650) value : inverts the image on the canvas and combines the result with the source bitmap by using the Boolean AND operator.
cmSrcInvert (see page 14740)	TCopyMode (see page 14650) value : combines the image on the canvas and the source bitmap by using the Boolean XOR operator.
cmSrcPaint (see page 14741)	TCopyMode (see page 14650) value : combines the image on the canvas and the source bitmap by using the Boolean OR operator.
cmWhiteness (see page 14741)	TCopyMode (see page 14650) value : fills the destination rectangle on the canvas with white.
ExtendedColorsCount (see page 14742)	The total number of extended colors available
rc3_Cursor (see page 14743)	This is constant Graphics.rc3_Cursor.
rc3_Icon (see page 14743)	This is constant Graphics.rc3_Icon.
rc3_StockIcon (see page 14743)	This is constant Graphics.rc3_StockIcon.
StandardColorsCount (see page 14744)	The total number of standard colors available
WebColorsCount (see page 14745)	The total number of web colors available

73.7.1 Graphics.cl3DDkShadow Constant

Represents the dark shadow of 3D components.

Pascal

```
cl3DDkShadow = TColor(clSystemColor or COLOR_3DDKSHADOW);
```

C++

```
cl3DDkShadow = TColor(clSystemColor or COLOR_3DDKSHADOW);
```

File

Graphics

Description

For Windows 95 or NT 4.0 only, cl3DDkShadow represents the dark shadow for three-dimensional display elements.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.2 Graphics.cl3DLight Constant

Represents the light part of 3D components.

Pascal

```
cl3DLight = TColor(clSystemColor or COLOR_3DLIGHT);
```

C++

```
cl3DLight = TColor(clSystemColor or COLOR_3DLIGHT);
```

File

Graphics

Description

For Windows 95 or NT 4.0 only, cl3DLight represents the light color for three-dimensional display elements (for edges facing the light source).

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.3 Graphics.clActiveBorder Constant

Represents the default border color of an active window.

Pascal

```
clActiveBorder = TColor(clSystemColor or COLOR_ACTIVEBORDER);
```

C++

```
clActiveBorder = TColor(clSystemColor or COLOR_ACTIVEBORDER);
```

File

Graphics

Description

clActiveBorder is the default color used to paint the border of an active window. clActiveBorder is a System (see page 22079) color - used for painting a component.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.4 Graphics.clActiveCaption Constant

Represents the default color of the title bar of an active window.

Pascal

```
clActiveCaption = TColor(clSystemColor or COLOR_ACTIVECAPTION);
```

C++

```
clActiveCaption = TColor(clSystemColor or COLOR_ACTIVECAPTION);
```

File

Graphics

Description

clActiveCaption is the default color used to paint the border of an active window. clActiveCaption is a System (see page 22079) color - used for painting a component.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.5 Graphics.clAppWorkSpace Constant

Represents the default Application WorkSpace color.

Pascal

```
clAppWorkSpace = TColor(clSystemColor or COLOR_APPWORKSPACE);
```

C++

```
clAppWorkSpace = TColor(clSystemColor or COLOR_APPWORKSPACE);
```

File

Graphics

Description

clAppWorkSpace is the default Application WorkSpace color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.6 Graphics.clAqua Constant

Represents the color Aqua.

Pascal

```
clAqua = TColor($FFFF00);
```

C++

```
clAqua = TColor($FFFF00);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to Aqua. Aqua is the Delphi name for cyan - 100% blue and green with no red. Aqua is a standard color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.7 Graphics.clBackground Constant

Represents the default Windows background color.

Pascal

```
clBackground = TColor(clSystemColor or COLOR_BACKGROUND);
```

C++

```
clBackground = TColor(clSystemColor or COLOR_BACKGROUND);
```

File

Graphics

Description

clBackground is the default Windows background color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.8 Graphics.clBlack Constant

Represents the color black.

Pascal

```
clBlack = TColor($000000);
```

C++

```
clBlack = TColor($000000);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to black. Black is the complete absence of blue, green and red. Black is a standard color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.9 Graphics.clBlue Constant

Represents the color blue.

Pascal

```
clBlue = TColor($FF0000);
```

C++

```
clBlue = TColor($FF0000);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to blue. Blue is 100% blue with no green or red. Blue is a standard color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.10 Graphics.clBtnFace Constant

Represents the color of the face of a button.

Pascal

```
clBtnFace = TColor(clSystemColor or COLOR_BTNFACE);
```

C++

```
clBtnFace = TColor(clSystemColor or COLOR_BTNFACE);
```

File

Graphics

Description

clBtnFace is the color used to paint the face of a button.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.11 Graphics.clBtnHighlight Constant

Represents the color of the highlighting on a button.

Pascal

```
clBtnHighlight = TColor(clSystemColor or COLOR_BTNHIGHLIGHT);
```

C++

```
clBtnHighlight = TColor(clSystemColor or COLOR_BTNHIGHLIGHT);
```

File

Graphics

Description

clBtnHighlight is the color used to paint the highlighting on a button.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.12 Graphics.clBtnShadow Constant

Represents the color of the shadow cast by of a button.

Pascal

```
clBtnShadow = TColor(clSystemColor or COLOR_BTNSHADOW);
```

C++

```
clBtnShadow = TColor(clSystemColor or COLOR_BTNSHADOW);
```

File

Graphics

Description

clBtnShadow is the color used to paint the shadow cast by a button.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.13 Graphics.clBtnText Constant

Represents the default color of the text on a button.

Pascal

```
clBtnText = TColor(clSystemColor or COLOR_BTNTEXT);
```

C++

```
clBtnText = TColor(clSystemColor or COLOR_BTNTEXT);
```

File

Graphics

Description

clBtnText is the default color used to paint the text on a button.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.14 Graphics.clCaptionText Constant

Represents the default color of the text on the title bar of an active window.

Pascal

```
clCaptionText = TColor(clSystemColor or COLOR_CAPTIONTEXT);
```

C++

```
clCaptionText = TColor(clSystemColor or COLOR_CAPTIONTEXT);
```

File

Graphics

Description

clCaptionText is the default color used for the text on the title bar of an active window. clCaptionText is a System (see page 22079) color - used for painting a component.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.15 Graphics.clCream Constant

Represents the color cream.

Pascal

```
clCream = TColor($F0FBFF);
```

C++

```
clCream = TColor($F0FBFF);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to cream. Cream is an off white color. Cream is an extended color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.16 Graphics.clDefault Constant

Specifies the default color of a component.

Pascal

```
clDefault = TColor($20000000);
```

C++

```
clDefault = TColor($20000000);
```

File

Graphics

Description

clDefault is used to determine or set the default color of a component.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.17 Graphics.clDkGray Constant

Represents the color dark gray.

Pascal

```
clDkGray = TColor($808080);
```

C++

```
clDkGray = TColor($808080);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to dark gray. Dark gray comprises 50% intensity of each of blue, green and red - identical to clGray (see page 14672). Dark Gray is a standard color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.18 Graphics.clFuchsia Constant

Represents the color fuchsia.

Pascal

```
clFuchsia = TColor($FF00FF);
```

C++

```
clFuchsia = TColor($FF00FF);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to fuchsia. Fuchsia is 100% intensity red, 100% intensity blue and 0% green. Fuchsia is a standard color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.19 Graphics.clGradientActiveCaption Constant

Right side color in the color gradient of an active window's title bar.

Pascal

```
clGradientActiveCaption = TColor(clSystemColor or COLOR_GRADIENTACTIVECAPTION);
```

C++

```
clGradientActiveCaption = TColor(clSystemColor or COLOR_GRADIENTACTIVECAPTION);
```

File

Graphics

Description

For a Windows 98 and Windows 2000 color gradient in an active caption, clActiveCaption (see page 14664) defines the left side color, and clGradientActiveCaption defines the right side color. The left side color blends to the right side color.

See Also

clActiveCaption (see page 14664)

clInactiveCaption (see page 14675)

clGradientInactiveCaption (see page 14671)

Color (see page 14437)

TColor (see page 14648)

73.7.20 Graphics.clGradientInactiveCaption Constant

Right side color in the color gradient of an inactive window's title bar.

Pascal

```
clGradientInactiveCaption = TColor(clSystemColor or COLOR_GRADIENTINACTIVECAPTION);
```

C++

```
clGradientInactiveCaption = TColor(clSystemColor or COLOR_GRADIENTINACTIVECAPTION);
```

File

Graphics

Description

For a Windows 98 and Windows 2000 color gradient in an inactive caption, clInactiveCaption (see page 14675) defines the left side color, and clGradientInactiveCaption defines the right side color. The left side color blends to the right side color.

See Also

clActiveCaption (see page 14664)

clGradientActiveCaption (see page 14671)

clInactiveCaption (see page 14675)

Color (see page 14437)

TColor (see page 14648)

73.7.21 Graphics.clGray Constant

Represents the color gray.

Pascal

```
clGray = TColor($808080);
```

C++

```
clGray = TColor($808080);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to gray. Gray comprises 50% intensity of each of blue, green and red. Gray is a standard color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.22 Graphics.clGrayText Constant

Represents the color of text that is grayed out.

Pascal

```
clGrayText = TColor(clSystemColor or COLOR_GRAYTEXT);
```

C++

```
clGrayText = TColor(clSystemColor or COLOR_GRAYTEXT);
```

File

Graphics

Description

clGrayText is the color used to paint text that has been set inactive, such as in an inactive menu item. clGrayText is a System (see page 22079) color - used for painting a component.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.23 Graphics.clGreen Constant

Represents the color green.

Pascal

```
clGreen = TColor($008000);
```

C++

```
clGreen = TColor($008000);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to green. Green is 100% green with no blue or red. Green is a standard color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.24 Graphics.clHighlight Constant

Represents the color of the background of a block of selected text.

Pascal

```
clHighlight = TColor(clSystemColor or COLOR_HIGHLIGHT);
```

C++

```
clHighlight = TColor(clSystemColor or COLOR_HIGHLIGHT);
```

File

Graphics

Description

clHighlight is the default color used to paint the background of a selected block of text. clHighlight is a System (see page 22079) color - used for painting a component.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.25 Graphics.clHighlightText Constant

Represents the color of selected text.

Pascal

```
clHighlightText = TColor(clSystemColor or COLOR_HIGHLIGHTTEXT);
```

C++

```
clHighlightText = TColor(clSystemColor or COLOR_HIGHLIGHTTEXT);
```

File

Graphics

Description

clHighlightText is the default color used for a selected block of text. clHighlightText is a System (see page 22079) color - used for painting a component.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.26 Graphics.clHotLight Constant

Pascal

```
clHotLight = TColor(clSystemColor or COLOR_HOTLIGHT);
```

C++

```
clHotLight = TColor(clSystemColor or COLOR_HOTLIGHT);
```

File

Graphics

Description

This is constant Graphics.clHotLight.

73.7.27 Graphics.clInactiveBorder Constant

Represents the default border color of an inactive window.

Pascal

```
clInactiveBorder = TColor(clSystemColor or COLOR_INACTIVEBORDER);
```

C++

```
clInactiveBorder = TColor(clSystemColor or COLOR_INACTIVEBORDER);
```

File

Graphics

Description

clInactiveBorder is the default color used to paint the border of an inactive window. clInactiveBorder is a System (see page 22079) color - used for painting a component.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.28 Graphics.clInactiveCaption Constant

Represents the default color of the title bar of an inactive window.

Pascal

```
clInactiveCaption = TColor(clSystemColor or COLOR_INACTIVECAPTION);
```

C++

```
clInactiveCaption = TColor(clSystemColor or COLOR_INACTIVECAPTION);
```

File

Graphics

Description

clInactiveCaption is the default color used to paint the border of an inactive window. clInactiveCaption is a System (see page 22079) color - used for painting a component.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.29 Graphics.clInactiveCaptionText Constant

Represents the default color of the title text in an inactive window.

Pascal

```
clInactiveCaptionText = TColor(clSystemColor or COLOR_INACTIVECAPTIONTEXT);
```

C++

```
clInactiveCaptionText = TColor(clSystemColor or COLOR_INACTIVECAPTIONTEXT);
```

File

Graphics

Description

clInactiveCaptionText is the default color used for the title font an inactive window. clInactiveCaptionText is a System (see page 22079) color - used for painting a component.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.30 Graphics.clInfoBk Constant

Represents the color of the background of a tool tip.

Pascal

```
clInfoBk = TColor(clSystemColor or COLOR_INFOBK);
```

C++

```
clInfoBk = TColor(clSystemColor or COLOR_INFOBK);
```

File

Graphics

Description

clInfoBk is the default color used for the background of a tool tip. clInfoBk is a System (see page 22079) color - used for painting a component.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.31 Graphics.clInfoText Constant

Represents the color of tool tip text.

Pascal

```
clInfoText = TColor(clSystemColor or COLOR_INFOTEXT);
```

C++

```
clInfoText = TColor(clSystemColor or COLOR_INFOTEXT);
```

File

Graphics

Description

clInfoText is the default color used for tool tip text. clInfoText is a System (see page 22079) color - used for painting a component.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.32 Graphics.clLime Constant

Represents the color lime.

Pascal

```
clLime = TColor($00FF00);
```

C++

```
clLime = TColor($00FF00);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to lime. Lime is 100% green with no blue or red. It is identical to clGreen (see page 14673). Lime is a standard color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.33 Graphics.clLtGray Constant

Represents the color light gray.

Pascal

```
clLtGray = TColor($C0C0C0);
```

C++

```
clLtGray = TColor($C0C0C0);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to light gray. Light gray comprises 75% intensity of each of blue, green and red. It is identical to clSilver (see page 14683). Light Gray is a standard color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.34 Graphics.clMaroon Constant

Represents the color maroon.

Pascal

```
clMaroon = TColor($000080);
```

C++

```
clMaroon = TColor($000080);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to maroon. Maroon gray comprises 50% intensity red, with no blue or green. Maroon is a standard color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.35 Graphics.clMedGray Constant

Represents the color medium gray.

Pascal

```
clMedGray = TColor($A4A0A0);
```

C++

```
clMedGray = TColor($A4A0A0);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to medium gray. Medium gray lies between clGray (see page 14672) and clLtGray (see page 14677), but, unusually, with a blue tint. Medium Gray is an extended color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.36 Graphics.clMenu Constant

Represents the color of the background of an unselected menu sub item.

Pascal

```
clMenu = TColor(clSystemColor or COLOR_MENU);
```

C++

```
clMenu = TColor(clSystemColor or COLOR_MENU);
```

File

Graphics

Description

clMenu is the default color used for the background of a sub menu item. clMenuBar (see page 14679) defines the color used for the background of the main menu items. clMenu is a System (see page 22079) color - used for painting a component.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.37 Graphics.clMenuBar Constant

Represents the color of the background of an unselected main menu item.

Pascal

```
clMenuBar = TColor(clSystemColor or COLOR_MENUBAR);
```

C++

```
clMenuBar = TColor(clSystemColor or COLOR_MENUBAR);
```

File

Graphics

Description

clMenuBar is the default color used for the background of a main menu item. clMenu (see page 14679) defines the color used for the background of menu sub items. clMenuBar is a System (see page 22079) color - used for painting a component.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.38 Graphics.clMenuHighlight Constant

Represents the color of the background of a selected menu item.

Pascal

```
clMenuHighlight = TColor(clSystemColor or COLOR_MENUHILIGHT);
```

C++

```
clMenuHighlight = TColor(clSystemColor or COLOR_MENUHILIGHT);
```

File

Graphics

Description

clMenuHighlight is the default color used for the background of a selcetd main or sub menu item. clMenuHighlight is a System (see page 22079) color - used for painting a component.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.39 Graphics.clMenuText Constant

Represents the default menu text color.

Pascal

```
clMenuText = TColor(clSystemColor or COLOR_MENUTEXT);
```

C++

```
clMenuText = TColor(clSystemColor or COLOR_MENUTEXT);
```

File

Graphics

Description

clMenuText is the default color for menu text.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.40 Graphics.clMoneyGreen Constant

Represents the color money green.

Pascal

```
clMoneyGreen = TColor($C0DCC0);
```

C++

```
clMoneyGreen = TColor($C0DCC0);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to money green. Money green is a green tinted light gray. Money green is an extended color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.41 Graphics.clNavy Constant

Represents the color navy.

Pascal

```
clNavy = TColor($800000);
```

C++

```
clNavy = TColor($800000);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to navy. Navy gray comprises 50% intensity blue, with no red or green. Navy is a standard color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.42 Graphics.clNone Constant

Represents no selected color.

Pascal

```
clNone = TColor($1FFFFFFF);
```

C++

```
clNone = TColor($1FFFFFFF);
```

File

Graphics

Description

clNone represents no selected color.

Note: clNone is white on Windows 9.x systems, and black on Windows NT systems.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.43 Graphics.clOlive Constant

Represents the color olive.

Pascal

```
clOlive = TColor($008080);
```

C++

```
clOlive = TColor($008080);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to olive. Olive is 50% intensity green, 50% intensity red with no blue. Olive is a standard color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.44 Graphics.clPurple Constant

Represents the color purple.

Pascal

```
clPurple = TColor($800080);
```

C++

```
clPurple = TColor($800080);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to purple. Purple is 50% intensity red, 50% intensity blue and 0% green. Purple is a standard color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.45 Graphics.clRed Constant

Represents the color red.

Pascal

```
clRed = TColor($0000FF);
```

C++

```
clRed = TColor($0000FF);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to red. Red is 100% red with no green or blue. Red is a standard color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.46 Graphics.clScrollBar Constant

Pascal

```
clScrollBar = TColor(clSystemColor or COLOR_SCROLLBAR);
```

C++

```
clScrollBar = TColor(clSystemColor or COLOR_SCROLLBAR);
```

File

Graphics

Description

This is constant Graphics.clScrollBar.

73.7.47 Graphics.clSilver Constant

Represents the color silver.

Pascal

```
clSilver = TColor($C0C0C0);
```

C++

```
clSilver = TColor($C0C0C0);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to silver. Silver comprises 75% intensity of each of blue, green and red. It is identical to clLtGray (see page 14677). Silver is a standard color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.48 Graphics.clSkyBlue Constant

Represents the color sky blue.

Pascal

```
clSkyBlue = TColor($F0CAA6);
```

C++

```
clSkyBlue = TColor($F0CAA6);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to sky blue. Sky blue light blue that is similar to the sky color. Sky blue is an extended color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.49 Graphics.clSystemColor Constant

Is a mask used to identify the set of system colors.

Pascal

```
clSystemColor = $FF000000;
```

C++

```
clSystemColor = 0xFF000000;
```

File

Graphics

Description

clSystemColor has the 1st byte bits set on as ameans of identifying a system color.

For example, clActiveBorder (see page 14664) is system color, used as the default color to paint the active border of a component - it has the high order byte set to $FF. clRed (see page 14683) is not a system color, since the high order byte is 0.

See Also

clActiveBorder (see page 14664)

clRed (see page 14683)

Color (see page 14437)

TColor (see page 14648)

73.7.50 Graphics.clTeal Constant

Represents the color teal.

Pascal

```
clTeal = TColor($808000);
```

C++

```
clTeal = TColor($808000);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to teal. Teal is dark cyan : 50% blue, 50% green, 0% red. Teal is a standard color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.51 Graphics.clWebAliceBlue Constant

Pascal

```
clWebAliceBlue = $FFF8F0;
```

C++

```
clWebAliceBlue = 0xFFF8F0;
```

File

Graphics

Description

This is constant Graphics.clWebAliceBlue.

73.7.52 Graphics.clWebAntiqueWhite Constant

Pascal

```
clWebAntiqueWhite = $D7EBFA;
```

C++

```
clWebAntiqueWhite = 0xD7EBFA;
```

File

Graphics

Description

This is constant Graphics.clWebAntiqueWhite.

73.7.53 Graphics.clWebAqua Constant

Pascal

```
clWebAqua = $FFFF00;
```

C++

```
clWebAqua = 0xFFFF00;
```

File

Graphics

Description

This is constant Graphics.clWebAqua.

73.7.54 Graphics.clWebAquamarine Constant

Pascal

```
clWebAquamarine = $D4FF7F;
```

C++

```
clWebAquamarine = 0xD4FF7F;
```

File

Graphics

Description

This is constant Graphics.clWebAquamarine.

73.7.55 Graphics.clWebAzure Constant

Pascal

```
clWebAzure = $FFFFF0;
```

C++

```
clWebAzure = 0xFFFFF0;
```

File

Graphics

Description

This is constant Graphics.clWebAzure.

73.7.56 Graphics.clWebBeige Constant

Pascal

```
clWebBeige = $DCF5F5;
```

C++

```
clWebBeige = 0xDCF5F5;
```

File

Graphics

Description

This is constant Graphics.clWebBeige.

73.7.57 Graphics.clWebBisque Constant

Pascal

```
clWebBisque = $C4E4FF;
```

C++

```
clWebBisque = 0xC4E4FF;
```

File

Graphics

Description

This is constant Graphics.clWebBisque.

73.7.58 Graphics.clWebBlack Constant

Pascal

```
clWebBlack = $000000;
```

C++

```
clWebBlack = 0x000000;
```

File

Graphics

Description

This is constant Graphics.clWebBlack.

73.7.59 Graphics.clWebBlanchedAlmond Constant

Pascal

```
clWebBlanchedAlmond = $CDEBFF;
```

C++

```
clWebBlanchedAlmond = 0xCDEBFF;
```

File

Graphics

Description

This is constant Graphics.clWebBlanchedAlmond.

73.7.60 Graphics.clWebBlue Constant

Pascal

```
clWebBlue = $FF0000;
```

C++

```
clWebBlue = 0xFF0000;
```

File

Graphics

Description

This is constant Graphics.clWebBlue.

73.7.61 Graphics.clWebBlueViolet Constant

Pascal

```
clWebBlueViolet = $E22B8A;
```

C++

```
clWebBlueViolet = 0xE22B8A;
```

File

Graphics

Description

This is constant Graphics.clWebBlueViolet.

73.7.62 Graphics.clWebBrown Constant

Pascal

```
clWebBrown = $2A2AA5;
```

C++

```
clWebBrown = 0x2A2AA5;
```

File

Graphics

Description

This is constant Graphics.clWebBrown.

73.7.63 Graphics.clWebBurlywood Constant

Pascal

```
clWebBurlywood = $87B8DE;
```

C++

```
clWebBurlywood = 0x87B8DE;
```

File

Graphics

Description

This is constant Graphics.clWebBurlywood.

73.7.64 Graphics.clWebCadetBlue Constant

Pascal

```
clWebCadetBlue = $A09E5F;
```

C++

```
clWebCadetBlue = 0xA09E5F;
```

File

Graphics

Description

This is constant Graphics.clWebCadetBlue.

73.7.65 Graphics.clWebChartreuse Constant

Pascal

```
clWebChartreuse = $00FF7F;
```

C++

```
clWebChartreuse = 0x00FF7F;
```

File

Graphics

Description

This is constant Graphics.clWebChartreuse.

73.7.66 Graphics.clWebChocolate Constant

Pascal

```
clWebChocolate = $1E69D2;
```

C++

```
clWebChocolate = 0x1E69D2;
```

File

Graphics

Description

This is constant Graphics.clWebChocolate.

73.7.67 Graphics.clWebCoral Constant

Pascal

```
clWebCoral = $507FFF;
```

C++

```
clWebCoral = 0x507FFF;
```

File

Graphics

Description

This is constant Graphics.clWebCoral.

73.7.68 Graphics.clWebCornFlowerBlue Constant

Pascal

```
clWebCornFlowerBlue = $ED9564;
```

C++

```
clWebCornFlowerBlue = 0xED9564;
```

File

Graphics

Description

This is constant Graphics.clWebCornFlowerBlue.

73.7.69 Graphics.clWebCornSilk Constant

Pascal

```
clWebCornSilk = $DCF8FF;
```

C++

```
clWebCornSilk = 0xDCF8FF;
```

File

Graphics

Description

This is constant Graphics.clWebCornSilk.

73.7.70 Graphics.clWebCrimson Constant

Pascal

```
clWebCrimson = $3C14DC;
```

C++

```
clWebCrimson = 0x3C14DC;
```

File

Graphics

Description

This is constant Graphics.clWebCrimson.

73.7.71 Graphics.clWebCyan Constant

Pascal

```
clWebCyan = $FFFF00;
```

C++

```
clWebCyan = 0xFFFF00;
```

File

Graphics

Description

This is constant Graphics.clWebCyan.

73.7.72 Graphics.clWebDarkBlue Constant

Pascal

```
clWebDarkBlue = $8B0000;
```

C++

```
clWebDarkBlue = 0x8B0000;
```

File

Graphics

Description

This is constant Graphics.clWebDarkBlue.

73.7.73 Graphics.clWebDarkCyan Constant

Pascal

```
clWebDarkCyan = $8B8B00;
```

C++

```
clWebDarkCyan = 0x8B8B00;
```

File

Graphics

Description

This is constant Graphics.clWebDarkCyan.

73.7.74 Graphics.clWebDarkGoldenRod Constant

Pascal

```
clWebDarkGoldenRod = $0B86B8;
```

C++

```
clWebDarkGoldenRod = 0x0B86B8;
```

File

Graphics

Description

This is constant Graphics.clWebDarkGoldenRod.

73.7.75 Graphics.clWebDarkGray Constant

Pascal

```
clWebDarkGray = $A9A9A9;
```

C++

```
clWebDarkGray = 0xA9A9A9;
```

File

Graphics

Description

This is constant Graphics.clWebDarkGray.

73.7.76 Graphics.clWebDarkgreen Constant

Pascal

```
clWebDarkgreen = $006400;
```

C++

```
clWebDarkgreen = 0x006400;
```

File

Graphics

Description

This is constant Graphics.clWebDarkgreen.

73.7.77 Graphics.clWebDarkKhaki Constant

Pascal

```
clWebDarkKhaki = $6BB7BD;
```

C++

```
clWebDarkKhaki = 0x6BB7BD;
```

File

Graphics

Description

This is constant Graphics.clWebDarkKhaki.

73.7.78 Graphics.clWebDarkMagenta Constant

Pascal

```
clWebDarkMagenta = $8B008B;
```

C++

```
clWebDarkMagenta = 0x8B008B;
```

File

Graphics

Description

This is constant Graphics.clWebDarkMagenta.

73.7.79 Graphics.clWebDarkOliveGreen Constant

Pascal

```
clWebDarkOliveGreen = $2F6B55;
```

C++

```
clWebDarkOliveGreen = 0x2F6B55;
```

File

Graphics

Description

This is constant Graphics.clWebDarkOliveGreen.

73.7.80 Graphics.clWebDarkOrange Constant

Pascal

```
clWebDarkOrange = $008CFF;
```

C++

```
clWebDarkOrange = 0x008CFF;
```

File

Graphics

Description

This is constant Graphics.clWebDarkOrange.

73.7.81 Graphics.clWebDarkOrchid Constant

Pascal

```
clWebDarkOrchid = $CC3299;
```

C++

```
clWebDarkOrchid = 0xCC3299;
```

File

Graphics

Description

This is constant Graphics.clWebDarkOrchid.

73.7.82 Graphics.clWebDarkRed Constant

Pascal

```
clWebDarkRed = $00008B;
```

C++

```
clWebDarkRed = 0x00008B;
```

File

Graphics

Description

This is constant Graphics.clWebDarkRed.

73.7.83 Graphics.clWebDarkSalmon Constant

Pascal

```
clWebDarkSalmon = $7A96E9;
```

C++

```
clWebDarkSalmon = 0x7A96E9;
```

File

Graphics

Description

This is constant Graphics.clWebDarkSalmon.

73.7.84 Graphics.clWebDarkSeaGreen Constant

Pascal

```
clWebDarkSeaGreen = $8FBC8F;
```

C++

```
clWebDarkSeaGreen = 0x8FBC8F;
```

File

Graphics

Description

This is constant Graphics.clWebDarkSeaGreen.

73.7.85 Graphics.clWebDarkSlateBlue Constant

Pascal

```
clWebDarkSlateBlue = $8B3D48;
```

C++

```
clWebDarkSlateBlue = 0x8B3D48;
```

File

Graphics

Description

This is constant Graphics.clWebDarkSlateBlue.

73.7.86 Graphics.clWebDarkSlategray Constant

Pascal

```
clWebDarkSlategray = $4F4F2F;
```

C++

```
clWebDarkSlategray = 0x4F4F2F;
```

File

Graphics

Description

This is constant Graphics.clWebDarkSlategray.

73.7.87 Graphics.clWebDarkTurquoise Constant

Pascal

```
clWebDarkTurquoise = $D1CE00;
```

C++

```
clWebDarkTurquoise = 0xD1CE00;
```

File

Graphics

Description

This is constant Graphics.clWebDarkTurquoise.

73.7.88 Graphics.clWebDarkViolet Constant

Pascal

```
clWebDarkViolet = $D30094;
```

C++

```
clWebDarkViolet = 0xD30094;
```

File

Graphics

Description

This is constant Graphics.clWebDarkViolet.

73.7.89 Graphics.clWebDeepPink Constant

Pascal

```
clWebDeepPink = $9314FF;
```

C++

```
clWebDeepPink = 0x9314FF;
```

File

Graphics

Description

This is constant Graphics.clWebDeepPink.

73.7.90 Graphics.clWebDeepskyBlue Constant

Pascal

```
clWebDeepskyBlue = $FFBF00;
```

C++

```
clWebDeepskyBlue = 0xFFBF00;
```

File

Graphics

Description

This is constant Graphics.clWebDeepskyBlue.

73.7.91 Graphics.clWebDimGray Constant

Pascal

```
clWebDimGray = $696969;
```

C++

```
clWebDimGray = 0x696969;
```

File

Graphics

Description

This is constant Graphics.clWebDimGray.

73.7.92 Graphics.clWebDodgerBlue Constant

Pascal

```
clWebDodgerBlue = $FF901E;
```

C++

```
clWebDodgerBlue = 0xFF901E;
```

File

Graphics

Description

This is constant Graphics.clWebDodgerBlue.

73.7.93 Graphics.clWebFirebrick Constant

Pascal

```
clWebFirebrick = $2222B2;
```

C++

```
clWebFirebrick = 0x2222B2;
```

File

Graphics

Description

This is constant Graphics.clWebFirebrick.

73.7.94 Graphics.clWebFloralWhite Constant

Pascal

```
clWebFloralWhite = $F0FAFF;
```

C++

```
clWebFloralWhite = 0xF0FAFF;
```

File

Graphics

Description

This is constant Graphics.clWebFloralWhite.

73.7.95 Graphics.clWebForestGreen Constant

Pascal

```
clWebForestGreen = $228B22;
```

C++

```
clWebForestGreen = 0x228B22;
```

File

Graphics

Description

This is constant Graphics.clWebForestGreen.

73.7.96 Graphics.clWebFuchsia Constant

Pascal

```
clWebFuchsia = $FF00FF;
```

C++

```
clWebFuchsia = 0xFF00FF;
```

File

Graphics

Description

This is constant Graphics.clWebFuchsia.

73.7.97 Graphics.clWebGainsboro Constant

Pascal

```
clWebGainsboro = $DCDCDC;
```

C++

```
clWebGainsboro = 0xDCDCDC;
```

File

Graphics

Description

This is constant Graphics.clWebGainsboro.

73.7.98 Graphics.clWebGhostWhite Constant

Pascal

```
clWebGhostWhite = $FFF8F8;
```

C++

```
clWebGhostWhite = 0xFFF8F8;
```

File

Graphics

Description

This is constant Graphics.clWebGhostWhite.

73.7.99 Graphics.clWebGold Constant

Pascal

```
clWebGold = $00D7FF;
```

C++

```
clWebGold = 0x00D7FF;
```

File

Graphics

Description

This is constant Graphics.clWebGold.

73.7.100 Graphics.clWebGoldenRod Constant

Pascal

```
clWebGoldenRod = $20A5DA;
```

C++

```
clWebGoldenRod = 0x20A5DA;
```

File

Graphics

Description

This is constant Graphics.clWebGoldenRod.

73.7.101 Graphics.clWebGray Constant

Pascal

```
clWebGray = $808080;
```

C++

```
clWebGray = 0x808080;
```

File

Graphics

Description

This is constant Graphics.clWebGray.

73.7.102 Graphics.clWebGreen Constant

Pascal

```
clWebGreen = $008000;
```

C++

```
clWebGreen = 0x008000;
```

File

Graphics

Description

This is constant Graphics.clWebGreen.

73.7.103 Graphics.clWebGreenYellow Constant

Pascal

```
clWebGreenYellow = $2FFFAD;
```

C++

```
clWebGreenYellow = 0x2FFFAD;
```

File

Graphics

Description

This is constant Graphics.clWebGreenYellow.

73.7.104 Graphics.clWebHoneydew Constant

Pascal

```
clWebHoneydew = $F0FFF0;
```

C++

```
clWebHoneydew = 0xF0FFF0;
```

File

Graphics

Description

This is constant Graphics.clWebHoneydew.

73.7.105 Graphics.clWebHotPink Constant

Pascal

```
clWebHotPink = $B469FF;
```

C++

```
clWebHotPink = 0xB469FF;
```

File

Graphics

Description

This is constant Graphics.clWebHotPink.

73.7.106 Graphics.clWebIndianRed Constant

Pascal

```
clWebIndianRed = $5C5CCD;
```

C++

```
clWebIndianRed = 0x5C5CCD;
```

File

Graphics

Description

This is constant Graphics.clWebIndianRed.

73.7.107 Graphics.clWebIndigo Constant

Pascal

```
clWebIndigo = $82004B;
```

C++

```
clWebIndigo = 0x82004B;
```

File

Graphics

Description

This is constant Graphics.clWebIndigo.

73.7.108 Graphics.clWebIvory Constant

Pascal

```
clWebIvory = $F0FFFF;
```

C++

```
clWebIvory = 0xF0FFFF;
```

File

Graphics

Description

This is constant Graphics.clWebIvory.

73.7.109 Graphics.clWebKhaki Constant

Pascal

```
clWebKhaki = $8CE6F0;
```

C++

```
clWebKhaki = 0x8CE6F0;
```

File

Graphics

Description

This is constant Graphics.clWebKhaki.

73.7.110 Graphics.clWebLavender Constant

Pascal

```
clWebLavender = $FAE6E6;
```

C++

```
clWebLavender = 0xFAE6E6;
```

File

Graphics

Description

This is constant Graphics.clWebLavender.

73.7.111 Graphics.clWebLavenderBlush Constant

Pascal

```
clWebLavenderBlush = $F5F0FF;
```

C++

```
clWebLavenderBlush = 0xF5F0FF;
```

File

Graphics

Description

This is constant Graphics.clWebLavenderBlush.

73.7.112 Graphics.clWebLawnGreen Constant

Pascal

```
clWebLawnGreen = $00FC7C;
```

C++

```
clWebLawnGreen = 0x00FC7C;
```

File

Graphics

Description

This is constant Graphics.clWebLawnGreen.

73.7.113 Graphics.clWebLemonChiffon Constant

Pascal

```
clWebLemonChiffon = $CDFAFF;
```

C++

```
clWebLemonChiffon = 0xCDFAFF;
```

File

Graphics

Description

This is constant Graphics.clWebLemonChiffon.

73.7.114 Graphics.clWebLightBlue Constant

Pascal

```
clWebLightBlue = $E6D8AD;
```

C++

```
clWebLightBlue = 0xE6D8AD;
```

File

Graphics

Description

This is constant Graphics.clWebLightBlue.

73.7.115 Graphics.clWebLightCoral Constant

Pascal

```
clWebLightCoral = $8080F0;
```

C++

```
clWebLightCoral = 0x8080F0;
```

File

Graphics

Description

This is constant Graphics.clWebLightCoral.

73.7.116 Graphics.clWebLightCyan Constant

Pascal

```
clWebLightCyan = $FFFFE0;
```

C++

```
clWebLightCyan = 0xFFFFE0;
```

File

Graphics

Description

This is constant Graphics.clWebLightCyan.

73.7.117 Graphics.clWebLightGoldenrodYellow Constant

Pascal

```
clWebLightGoldenrodYellow = $D2FAFA;
```

C++

```
clWebLightGoldenrodYellow = 0xD2FAFA;
```

File

Graphics

Description

This is constant Graphics.clWebLightGoldenrodYellow.

73.7.118 Graphics.clWebLightGreen Constant

Pascal

```
clWebLightGreen = $90EE90;
```

C++

```
clWebLightGreen = 0x90EE90;
```

File

Graphics

Description

This is constant Graphics.clWebLightGreen.

73.7.119 Graphics.clWebLightgrey Constant

Pascal

```
clWebLightgrey = $D3D3D3;
```

C++

```
clWebLightgrey = 0xD3D3D3;
```

File

Graphics

Description

This is constant Graphics.clWebLightgrey.

73.7.120 Graphics.clWebLightPink Constant

Pascal

```
clWebLightPink = $C1B6FF;
```

C++

```
clWebLightPink = 0xC1B6FF;
```

File

Graphics

Description

This is constant Graphics.clWebLightPink.

73.7.121 Graphics.clWebLightSalmon Constant

Pascal

```
clWebLightSalmon = $7AA0FF;
```

C++

```
clWebLightSalmon = 0x7AA0FF;
```

File

Graphics

Description

This is constant Graphics.clWebLightSalmon.

73.7.122 Graphics.clWebLightSeaGreen Constant

Pascal

```
clWebLightSeaGreen = $AAB220;
```

C++

```
clWebLightSeaGreen = 0xAAB220;
```

File

Graphics

Description

This is constant Graphics.clWebLightSeaGreen.

73.7.123 Graphics.clWebLightSkyBlue Constant

Pascal

```
clWebLightSkyBlue = $FACE87;
```

C++

```
clWebLightSkyBlue = 0xFACE87;
```

File

Graphics

Description

This is constant Graphics.clWebLightSkyBlue.

73.7.124 Graphics.clWebLightSlateGray Constant

Pascal

```
clWebLightSlateGray = $998877;
```

C++

```
clWebLightSlateGray = 0x998877;
```

File

Graphics

Description

This is constant Graphics.clWebLightSlateGray.

73.7.125 Graphics.clWebLightSteelBlue Constant

Pascal

```
clWebLightSteelBlue = $DEC4B0;
```

C++

```
clWebLightSteelBlue = 0xDEC4B0;
```

File

Graphics

Description

This is constant Graphics.clWebLightSteelBlue.

73.7.126 Graphics.clWebLightYellow Constant

Pascal

```
clWebLightYellow = $E0FFFF;
```

C++

```
clWebLightYellow = 0xE0FFFF;
```

File

Graphics

Description

This is constant Graphics.clWebLightYellow.

73.7.127 Graphics.clWebLime Constant

Pascal

```
clWebLime = $00FF00;
```

C++

```
clWebLime = 0x00FF00;
```

File

Graphics

Description

This is constant Graphics.clWebLime.

73.7.128 Graphics.clWebLimeGreen Constant

Pascal

```
clWebLimeGreen = $32CD32;
```

C++

```
clWebLimeGreen = 0x32CD32;
```

File

Graphics

Description

This is constant Graphics.clWebLimeGreen.

73.7.129 Graphics.clWebLinen Constant

Pascal

```
clWebLinen = $E6F0FA;
```

C++

```
clWebLinen = 0xE6F0FA;
```

File

Graphics

Description

This is constant Graphics.clWebLinen.

73.7.130 Graphics.clWebMagenta Constant

Pascal

```
clWebMagenta = $FF00FF;
```

C++

```
clWebMagenta = 0xFF00FF;
```

File

Graphics

Description

This is constant Graphics.clWebMagenta.

73.7.131 Graphics.clWebMaroon Constant

Pascal

```
clWebMaroon = $000080;
```

C++

```
clWebMaroon = 0x000080;
```

File

Graphics

Description

This is constant Graphics.clWebMaroon.

73.7.132 Graphics.clWebMediumAquamarine Constant

Pascal

```
clWebMediumAquamarine = $AACD66;
```

C++

```
clWebMediumAquamarine = 0xAACD66;
```

File

Graphics

Description

This is constant Graphics.clWebMediumAquamarine.

73.7.133 Graphics.clWebMediumBlue Constant

Pascal

```
clWebMediumBlue = $CD0000;
```

C++

```
clWebMediumBlue = 0xCD0000;
```

File

Graphics

Description

This is constant Graphics.clWebMediumBlue.

73.7.134 Graphics.clWebMediumOrchid Constant

Pascal

```
clWebMediumOrchid = $D355BA;
```

C++

```
clWebMediumOrchid = 0xD355BA;
```

File

Graphics

Description

This is constant Graphics.clWebMediumOrchid.

73.7.135 Graphics.clWebMediumPurple Constant

Pascal

```
clWebMediumPurple = $DB7093;
```

C++

```
clWebMediumPurple = 0xDB7093;
```

File

Graphics

Description

This is constant Graphics.clWebMediumPurple.

73.7.136 Graphics.clWebMediumSeaGreen Constant

Pascal

```
clWebMediumSeaGreen = $71B33C;
```

C++

```
clWebMediumSeaGreen = 0x71B33C;
```

File

Graphics

Description

This is constant Graphics.clWebMediumSeaGreen.

73.7.137 Graphics.clWebMediumSlateBlue Constant

Pascal

```
clWebMediumSlateBlue = $EE687B;
```

C++

```
clWebMediumSlateBlue = 0xEE687B;
```

File

Graphics

Description

This is constant Graphics.clWebMediumSlateBlue.

73.7.138 Graphics.clWebMediumSpringGreen Constant

Pascal

```
clWebMediumSpringGreen = $9AFA00;
```

C++

```
clWebMediumSpringGreen = 0x9AFA00;
```

File

Graphics

Description

This is constant Graphics.clWebMediumSpringGreen.

73.7.139 Graphics.clWebMediumTurquoise Constant

Pascal

```
clWebMediumTurquoise = $CCD148;
```

C++

```
clWebMediumTurquoise = 0xCCD148;
```

File

Graphics

Description

This is constant Graphics.clWebMediumTurquoise.

73.7.140 Graphics.clWebMediumVioletRed Constant

Pascal

```
clWebMediumVioletRed = $8515C7;
```

C++

```
clWebMediumVioletRed = 0x8515C7;
```

File

Graphics

Description

This is constant Graphics.clWebMediumVioletRed.

73.7.141 Graphics.clWebMidnightBlue Constant

Pascal

```
clWebMidnightBlue = $701919;
```

C++

```
clWebMidnightBlue = 0x701919;
```

File

Graphics

Description

This is constant Graphics.clWebMidnightBlue.

73.7.142 Graphics.clWebMintcream Constant

Pascal

```
clWebMintcream = $FAFFF5;
```

C++

```
clWebMintcream = 0xFAFFF5;
```

File

Graphics

Description

This is constant Graphics.clWebMintcream.

73.7.143 Graphics.clWebMistyRose Constant

Pascal

```
clWebMistyRose = $E1E4FF;
```

C++

```
clWebMistyRose = 0xE1E4FF;
```

File

Graphics

Description

This is constant Graphics.clWebMistyRose.

73.7.144 Graphics.clWebMoccasin Constant

Pascal

```
clWebMoccasin = $B5E4FF;
```

C++

```
clWebMoccasin = 0xB5E4FF;
```

File

Graphics

Description

This is constant Graphics.clWebMoccasin.

73.7.145 Graphics.clWebNavajoWhite Constant

Pascal

```
clWebNavajoWhite = $ADDEFF;
```

C++

```
clWebNavajoWhite = 0xADDEFF;
```

File

Graphics

Description

This is constant Graphics.clWebNavajoWhite.

73.7.146 Graphics.clWebNavy Constant

Pascal

```
clWebNavy = $800000;
```

C++

```
clWebNavy = 0x800000;
```

File

Graphics

Description

This is constant Graphics.clWebNavy.

73.7.147 Graphics.clWebOldLace Constant

Pascal

```
clWebOldLace = $E6F5FD;
```

C++

```
clWebOldLace = 0xE6F5FD;
```

File

Graphics

Description

This is constant Graphics.clWebOldLace.

73.7.148 Graphics.clWebOlive Constant

Pascal

```
clWebOlive = $008080;
```

C++

```
clWebOlive = 0x008080;
```

File

Graphics

Description

This is constant Graphics.clWebOlive.

73.7.149 Graphics.clWebOliveDrab Constant

Pascal

```
clWebOliveDrab = $238E6B;
```

C++

```
clWebOliveDrab = 0x238E6B;
```

File

Graphics

Description

This is constant Graphics.clWebOliveDrab.

73.7.150 Graphics.clWebOrange Constant

Pascal

```
clWebOrange = $00A5FF;
```

C++

```
clWebOrange = 0x00A5FF;
```

File

Graphics

Description

This is constant Graphics.clWebOrange.

73.7.151 Graphics.clWebOrangeRed Constant

Pascal

```
clWebOrangeRed = $0045FF;
```

C++

```
clWebOrangeRed = 0x0045FF;
```

File

Graphics

Description

This is constant Graphics.clWebOrangeRed.

73.7.152 Graphics.clWebOrchid Constant

Pascal

```
clWebOrchid = $D670DA;
```

C++

```
clWebOrchid = 0xD670DA;
```

File

Graphics

Description

This is constant Graphics.clWebOrchid.

73.7.153 Graphics.clWebPaleGoldenrod Constant

Pascal

```
clWebPaleGoldenrod = $AAE8EE;
```

C++

```
clWebPaleGoldenrod = 0xAAE8EE;
```

File

Graphics

Description

This is constant Graphics.clWebPaleGoldenrod.

73.7.154 Graphics.clWebPaleGreen Constant

Pascal

```
clWebPaleGreen = $98FB98;
```

C++

```
clWebPaleGreen = 0x98FB98;
```

File

Graphics

Description

This is constant Graphics.clWebPaleGreen.

73.7.155 Graphics.clWebPaleTurquoise Constant

Pascal

```
clWebPaleTurquoise = $EEEEAF;
```

C++

```
clWebPaleTurquoise = 0xEEEEAF;
```

File

Graphics

Description

This is constant Graphics.clWebPaleTurquoise.

73.7.156 Graphics.clWebPaleVioletRed Constant

Pascal

```
clWebPaleVioletRed = $9370DB;
```

C++

```
clWebPaleVioletRed = 0x9370DB;
```

File

Graphics

Description

This is constant Graphics.clWebPaleVioletRed.

73.7.157 Graphics.clWebPapayaWhip Constant

Pascal

```
clWebPapayaWhip = $D5EFFF;
```

C++

```
clWebPapayaWhip = 0xD5EFFF;
```

File

Graphics

Description

This is constant Graphics.clWebPapayaWhip.

73.7.158 Graphics.clWebPeachPuff Constant

Pascal

```
clWebPeachPuff = $B9DAFF;
```

C++

```
clWebPeachPuff = 0xB9DAFF;
```

File

Graphics

Description

This is constant Graphics.clWebPeachPuff.

73.7.159 Graphics.clWebPeru Constant

Pascal

```
clWebPeru = $3F85CD;
```

C++

```
clWebPeru = 0x3F85CD;
```

File

Graphics

Description

This is constant Graphics.clWebPeru.

73.7.160 Graphics.clWebPink Constant

Pascal

```
clWebPink = $CBC0FF;
```

C++

```
clWebPink = 0xCBC0FF;
```

File

Graphics

Description

This is constant Graphics.clWebPink.

73.7.161 Graphics.clWebPlum Constant

Pascal

```
clWebPlum = $DDA0DD;
```

C++

```
clWebPlum = 0xDDA0DD;
```

File

Graphics

Description

This is constant Graphics.clWebPlum.

73.7.162 Graphics.clWebPowderBlue Constant

Pascal

```
clWebPowderBlue = $E6E0B0;
```

C++

```
clWebPowderBlue = 0xE6E0B0;
```

File

Graphics

Description

This is constant Graphics.clWebPowderBlue.

73.7.163 Graphics.clWebPurple Constant

Pascal

```
clWebPurple = $800080;
```

C++

```
clWebPurple = 0x800080;
```

File

Graphics

Description

This is constant Graphics.clWebPurple.

73.7.164 Graphics.clWebRed Constant

Pascal

```
clWebRed = $0000FF;
```

C++

```
clWebRed = 0x0000FF;
```

File

Graphics

Description

This is constant Graphics.clWebRed.

73.7.165 Graphics.clWebRosyBrown Constant

Pascal

```
clWebRosyBrown = $8F8FBC;
```

C++

```
clWebRosyBrown = 0x8F8FBC;
```

File

Graphics

Description

This is constant Graphics.clWebRosyBrown.

73.7.166 Graphics.clWebRoyalBlue Constant

Pascal

```
clWebRoyalBlue = $E16941;
```

C++

```
clWebRoyalBlue = 0xE16941;
```

File

Graphics

Description

This is constant Graphics.clWebRoyalBlue.

73.7.167 Graphics.clWebSaddleBrown Constant

Pascal

```
clWebSaddleBrown = $13458B;
```

C++

```
clWebSaddleBrown = 0x13458B;
```

File

Graphics

Description

This is constant Graphics.clWebSaddleBrown.

73.7.168 Graphics.clWebSalmon Constant

Pascal

```
clWebSalmon = $7280FA;
```

C++

```
clWebSalmon = 0x7280FA;
```

File

Graphics

Description

This is constant Graphics.clWebSalmon.

73.7.169 Graphics.clWebSandyBrown Constant

Pascal

```
clWebSandyBrown = $60A4F4;
```

C++

```
clWebSandyBrown = 0x60A4F4;
```

File

Graphics

Description

This is constant Graphics.clWebSandyBrown.

73.7.170 Graphics.clWebSeaGreen Constant

Pascal

```
clWebSeaGreen = $578B2E;
```

C++

```
clWebSeaGreen = 0x578B2E;
```

File

Graphics

Description

This is constant Graphics.clWebSeaGreen.

73.7.171 Graphics.clWebSeashell Constant

Pascal

```
clWebSeashell = $EEF5FF;
```

C++

```
clWebSeashell = 0xEEF5FF;
```

File

Graphics

Description

This is constant Graphics.clWebSeashell.

73.7.172 Graphics.clWebSienna Constant

Pascal

```
clWebSienna = $2D52A0;
```

C++

```
clWebSienna = 0x2D52A0;
```

File

Graphics

Description

This is constant Graphics.clWebSienna.

73.7.173 Graphics.clWebSilver Constant

Pascal

```
clWebSilver = $C0C0C0;
```

C++

```
clWebSilver = 0xC0C0C0;
```

File

Graphics

Description

This is constant Graphics.clWebSilver.

73.7.174 Graphics.clWebSkyBlue Constant

Pascal

```
clWebSkyBlue = $EBCE87;
```

C++

```
clWebSkyBlue = 0xEBCE87;
```

File

Graphics

Description

This is constant Graphics.clWebSkyBlue.

73.7.175 Graphics.clWebSlateBlue Constant

Pascal

```
clWebSlateBlue = $CD5A6A;
```

C++

```
clWebSlateBlue = 0xCD5A6A;
```

File

Graphics

Description

This is constant Graphics.clWebSlateBlue.

73.7.176 Graphics.clWebSlateGray Constant

Pascal

```
clWebSlateGray = $908070;
```

C++

```
clWebSlateGray = 0x908070;
```

File

Graphics

Description

This is constant Graphics.clWebSlateGray.

73.7.177 Graphics.clWebSnow Constant

Pascal

```
clWebSnow = $FAFAFF;
```

C++

```
clWebSnow = 0xFAFAFF;
```

File

Graphics

Description

This is constant Graphics.clWebSnow.

73.7.178 Graphics.clWebSpringGreen Constant

Pascal

```
clWebSpringGreen = $7FFF00;
```

C++

```
clWebSpringGreen = 0x7FFF00;
```

File

Graphics

Description

This is constant Graphics.clWebSpringGreen.

73.7.179 Graphics.clWebSteelBlue Constant

Pascal

```
clWebSteelBlue = $B48246;
```

C++

```
clWebSteelBlue = 0xB48246;
```

File

Graphics

Description

This is constant Graphics.clWebSteelBlue.

73.7.180 Graphics.clWebTan Constant

Pascal

```
clWebTan = $8CB4D2;
```

C++

```
clWebTan = 0x8CB4D2;
```

File

Graphics

Description

This is constant Graphics.clWebTan.

73.7.181 Graphics.clWebTeal Constant

Pascal

```
clWebTeal = $808000;
```

C++

```
clWebTeal = 0x808000;
```

File

Graphics

Description

This is constant Graphics.clWebTeal.

73.7.182 Graphics.clWebThistle Constant

Pascal

```
clWebThistle = $D8BFD8;
```

C++

```
clWebThistle = 0xD8BFD8;
```

File

Graphics

Description

This is constant Graphics.clWebThistle.

73.7.183 Graphics.clWebTomato Constant

Pascal

```
clWebTomato = $4763FF;
```

C++

```
clWebTomato = 0x4763FF;
```

File

Graphics

Description

This is constant Graphics.clWebTomato.

73.7.184 Graphics.clWebTurquoise Constant

Pascal

```
clWebTurquoise = $D0E040;
```

C++

```
clWebTurquoise = 0xD0E040;
```

File

Graphics

Description

This is constant Graphics.clWebTurquoise.

73.7.185 Graphics.clWebViolet Constant

Pascal

```
clWebViolet = $EE82EE;
```

C++

```
clWebViolet = 0xEE82EE;
```

File

Graphics

Description

This is constant Graphics.clWebViolet.

73.7.186 Graphics.clWebWheat Constant

Pascal

```
clWebWheat = $B3DEF5;
```

C++

```
clWebWheat = 0xB3DEF5;
```

File

Graphics

Description

This is constant Graphics.clWebWheat.

73.7.187 Graphics.clWebWhite Constant

Pascal

```
clWebWhite = $FFFFFF;
```

C++

```
clWebWhite = 0xFFFFFF;
```

File

Graphics

Description

This is constant Graphics.clWebWhite.

73.7.188 Graphics.clWebWhiteSmoke Constant

Pascal

```
clWebWhiteSmoke = $F5F5F5;
```

C++

```
clWebWhiteSmoke = 0xF5F5F5;
```

File

Graphics

Description

This is constant Graphics.clWebWhiteSmoke.

73.7.189 Graphics.clWebYellow Constant

Pascal

```
clWebYellow = $00FFFF;
```

C++

```
clWebYellow = 0x00FFFF;
```

File

Graphics

Description

This is constant Graphics.clWebYellow.

73.7.190 Graphics.clWebYellowGreen Constant

Pascal

```
clWebYellowGreen = $32CD9A;
```

C++

```
clWebYellowGreen = 0x32CD9A;
```

File

Graphics

Description

This is constant Graphics.clWebYellowGreen.

73.7.191 Graphics.clWhite Constant

Represents the color white.

Pascal

```
clWhite = TColor($FFFFFF);
```

C++

```
clWhite = TColor($FFFFFF);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to white. White is 100% intensity blue, green and red. White is a standard color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.192 Graphics.clWindow Constant

Represents the default Window background color.

Pascal

```
clWindow = TColor(clSystemColor or COLOR_WINDOW);
```

C++

```
clWindow = TColor(clSystemColor or COLOR_WINDOW);
```

File

Graphics

Description

clWindow is the default background color for a window.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.193 Graphics.clWindowFrame Constant

Represents the default Window frame background color.

Pascal

```
clWindowFrame = TColor(clSystemColor or COLOR_WINDOWFRAME);
```

C++

```
clWindowFrame = TColor(clSystemColor or COLOR_WINDOWFRAME);
```

File

Graphics

Description

clWindowFrame is the default background color for a window frame.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.194 Graphics.clWindowText Constant

Represents the default Window text color.

Pascal

```
clWindowText = TColor(clSystemColor or COLOR_WINDOWTEXT);
```

C++

```
clWindowText = TColor(clSystemColor or COLOR_WINDOWTEXT);
```

File

Graphics

Description

clWindowText is the default color for window text.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.195 Graphics.clYellow Constant

Represents the color yellow.

Pascal

```
clYellow = TColor($00FFFF);
```

C++

```
clYellow = TColor($00FFFF);
```

File

Graphics

Description

Use the Color property of a visual component to set its principle color to yellow. Yellow is 100% green, 100% red and 0% blue. Yellow is a standard color.

See Also

Color (see page 14437)

TColor (see page 14648)

73.7.196 Graphics.cmBlackness Constant

TCopyMode (see page 14650) value : fills the destination rectangle on the canvas with black.

Pascal

```
cmBlackness = BLACKNESS;
```

C++

```
cmBlackness = BLACKNESS;
```

File

Graphics

Description

cmBlackness is a TCopyMode (see page 14650) value used in the TCanvas (see page 14441) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

cmBlackness results in the destination rectangle on the canvas simply being filled with the clBlack (see page 14666) color.

See Also

CopyMode (see page 14487)

CopyRect (see page 14452)

TCopyMode (see page 14650)

73.7.197 Graphics.cmDstInvert Constant

TCopyMode (see page 14650) value : inverts the image on the canvas and ignores the source.

Pascal

```
cmDstInvert = DSTINVERT;
```

C++

```
cmDstInvert = DSTINVERT;
```

File

Graphics

Description

cmDstInvert is a TCopyMode (see page 14650) value used in the TCanvas (see page 14441) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

cmDstInvert results in the the bits of each pixel in destination rectangle on the canvas simply being inverted. The source rectangle is ignored.

See Also

CopyMode (see page 14487)

CopyRect (see page 14452)

TCopyMode (see page 14650)

73.7.198 Graphics.cmMergeCopy Constant

TCopyMode (see page 14650) value : combines the image on the canvas and the source bitmap by using the Boolean AND operator.

Pascal

```
cmMergeCopy = MERGECOPY;
```

C++

```
cmMergeCopy = MERGECOPY;
```

File

Graphics

Description

cmMergeCopy is a TCopyMode (see page 14650) value used in the TCanvas (see page 14441) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

cmMergeCopy results in the pixels in the source and destination rectangles being logically ANDed. For example, a green pixel in the source will merge with a blue destination pixel to result in a cyan pixel.

See Also

CopyMode (see page 14487)

CopyRect (see page 14452)

TCopyMode (see page 14650)

73.7.199 Graphics.cmMergePaint Constant

TCopyMode (see page 14650) value : combines the inverted source bitmap with the image on the canvas by using the Boolean OR operator.

Pascal

```
cmMergePaint = MERGEPAINT;
```

C++

```
cmMergePaint = MERGEPAINT;
```

File

Graphics

Description

cmMergePaint is a TCopyMode (see page 14650) value used in the TCanvas (see page 14441) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmMergePaint, the source pixels are inverted before being logically ORed with the destination pixels.

See Also

CopyMode (see page 14487)

CopyRect (see page 14452)

TCopyMode (see page 14650)

73.7.200 Graphics.cmNotSrcCopy Constant

TCopyMode (see page 14650) value : copies the inverted source bitmap to the canvas.

Pascal

```
cmNotSrcCopy = NOTSRCCOPY;
```

C++

```
cmNotSrcCopy = NOTSRCCOPY;
```

File

Graphics

Description

cmNotSrcCopy is a TCopyMode (see page 14650) value used in the TCanvas (see page 14441) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmNotSrcCopy, the source pixels are inverted before being written to the target rectangle. The existing target rectangle contents are ignored and overwritten.

See Also

CopyMode (see page 14487)

CopyRect (see page 14452)

TCopyMode (see page 14650)

73.7.201 Graphics.cmNotSrcErase Constant

TCopyMode (see page 14650) value : combines the image on the canvas and the source bitmap by using the Boolean OR operator, and inverts the result.

Pascal

```
cmNotSrcErase = NOTSRCERASE;
```

C++

```
cmNotSrcErase = NOTSRCERASE;
```

File

Graphics

Description

cmNotSrcErase is a TCopyMode (see page 14650) value used in the TCanvas (see page 14441) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmNotSrcErase, the source and destination pixels are ORed together and the result inverted and stored in the target rectangle.

See Also

CopyMode (see page 14487)

CopyRect (see page 14452)

TCopyMode (see page 14650)

73.7.202 Graphics.cmPatCopy Constant

TCopyMode (see page 14650) value : copies the source pattern to the canvas.

Pascal

```
cmPatCopy = PATCOPY;
```

C++

```
cmPatCopy = PATCOPY;
```

File

Graphics

Description

cmPatCopy is a TCopyMode (see page 14650) value used in the TCanvas (see page 14441) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmPatCopy, the source pattern is written to the target rectangle.

See Also

CopyMode (see page 14487)

CopyRect (see page 14452)

TCopyMode (see page 14650)

73.7.203 Graphics.cmPatInvert Constant

TCopyMode (see page 14650) value : combines the source pattern with the image on the canvas using the Boolean XOR operator.

Pascal

```
cmPatInvert = PATINVERT;
```

C++

```
cmPatInvert = PATINVERT;
```

File

Graphics

Description

cmPatInvert is a TCopyMode (see page 14650) value used in the TCanvas (see page 14441) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

cmPatInvert, XORs each source pattern pixel with the target image pixels.

See Also

CopyMode (see page 14487)

CopyRect (see page 14452)

TCopyMode (see page 14650)

73.7.204 Graphics.cmPatPaint Constant

TCopyMode (see page 14650) value : combines the inverted source bitmap with the source pattern by using the Boolean OR operator. Combines the result of this operation with the image on the canvas by using the Boolean OR operator.

Pascal

```
cmPatPaint = PATPAINT;
```

C++

```
cmPatPaint = PATPAINT;
```

File

Graphics

Description

cmPatPaint is a TCopyMode (see page 14650) value used in the TCanvas (see page 14441) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmPatPaint, the source pattern and inverted bitmap pixels are ORed before ORing the resulting pixels with the target image pixels.

See Also

CopyMode (see page 14487)

CopyRect (see page 14452)

TCopyMode (see page 14650)

73.7.205 Graphics.cmSrcAnd Constant

TCopyMode (see page 14650) value : combines the image on the canvas and source bitmap by using the Boolean AND operator.

Pascal

```
cmSrcAnd = SRCAND;
```

C++

```
cmSrcAnd = SRCAND;
```

File

Graphics

Description

cmSrcAnd is a TCopyMode (see page 14650) value used in the TCanvas (see page 14441) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmSrcAnd, the source pixels are ANDed with the target target pixels.

See Also

CopyMode (see page 14487)

CopyRect (see page 14452)

TCopyMode (see page 14650)

73.7.206 Graphics.cmSrcCopy Constant

TCopyMode (see page 14650) value : copies the source bitmap to the canvas.

Pascal

```
cmSrcCopy = SRCCOPY;
```

C++

```
cmSrcCopy = SRCCOPY;
```

File

Graphics

Description

cmSrcCopy is a TCopyMode (see page 14650) value used in the TCanvas (see page 14441) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmSrcCopy, the source pixels are written unchanged to the target rectangle.

See Also

CopyMode (see page 14487)

CopyRect (see page 14452)

TCopyMode (see page 14650)

73.7.207 Graphics.cmSrcErase Constant

TCopyMode (see page 14650) value : inverts the image on the canvas and combines the result with the source bitmap by using the Boolean AND operator.

Pascal

```
cmSrcErase = SRCERASE;
```

C++

```
cmSrcErase = SRCERASE;
```

File

Graphics

Description

cmSrcErase is a TCopyMode (see page 14650) value used in the TCanvas (see page 14441) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmSrcErase, the source pixels are ANDed with the inverted target pixels.

See Also

CopyMode (see page 14487)

CopyRect (see page 14452)

TCopyMode (see page 14650)

73.7.208 Graphics.cmSrcInvert Constant

TCopyMode (see page 14650) value : combines the image on the canvas and the source bitmap by using the Boolean XOR operator.

Pascal

```
cmSrcInvert = SRCINVERT;
```

C++

```
cmSrcInvert = SRCINVERT;
```

File

Graphics

Description

cmSrcInvert is a TCopyMode (see page 14650) value used in the TCanvas (see page 14441) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmSrcInvert, the source pixels are XORed with the target pixels.

See Also

CopyMode (see page 14487)

CopyRect (see page 14452)

TCopyMode (see page 14650)

73.7.209 Graphics.cmSrcPaint Constant

TCopyMode (see page 14650) value : combines the image on the canvas and the source bitmap by using the Boolean OR operator.

Pascal

```
cmSrcPaint = SRCPAINT;
```

C++

```
cmSrcPaint = SRCPAINT;
```

File

Graphics

Description

cmSrcPaint is a TCopyMode (see page 14650) value used in the TCanvas (see page 14441) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

With cmSrcPaint, the source pixels are ORed with the target pixels.

See Also

CopyMode (see page 14487)

CopyRect (see page 14452)

TCopyMode (see page 14650)

73.7.210 Graphics.cmWhiteness Constant

TCopyMode (see page 14650) value : fills the destination rectangle on the canvas with white.

Pascal

```
cmWhiteness = WHITENESS;
```

C++

```
cmWhiteness = WHITENESS;
```

File

Graphics

Description

cmWhiteness is a TCopyMode (see page 14650) value used in the TCanvas (see page 14441) CopyMode property.

The CopyMode property is used by the CopyRect routine when copying an image from another canvas onto the current canvas.

cmWhiteness results in the destination rectangle on the canvas simply being filled with the clWhite (see page 14732) color.

See Also

CopyMode (see page 14487)

CopyRect (see page 14452)

TCopyMode (see page 14650)

73.7.211 Graphics.ExtendedColorsCount Constant

The total number of extended colors available

Pascal

```
ExtendedColorsCount = 4;
```

C++

```
ExtendedColorsCount = 4;
```

File

Graphics

Description

TColor (see page 14648) can be assigned standard, extended, web and system color values. The system colors are default values for various visual component colors. Extended colors are :

Name	Meaning
clMoneyGreen (see page 14680)	Money Green.
clSkyBlue (see page 14684)	Sky blue.
clCream (see page 14669)	Cream.
clMedGray (see page 14678)	Medium Gray.

See Also

Color (see page 14437)

StandardColorsCount (see page 14744)

WebColorsCount (see page 14745)

TColor (see page 14648)

73.7.212 Graphics.rc3_Cursor Constant

Pascal

```
rc3_Cursor = 2;
```

C++

```
rc3_Cursor = 2;
```

File

Graphics

Description

This is constant Graphics.rc3_Cursor.

73.7.213 Graphics.rc3_Icon Constant

Pascal

```
rc3_Icon = 1;
```

C++

```
rc3_Icon = 1;
```

File

Graphics

Description

This is constant Graphics.rc3_Icon.

73.7.214 Graphics.rc3_StockIcon Constant

Pascal

```
rc3_StockIcon = 0;
```

C++

```
rc3_StockIcon = 0;
```

File

Graphics

Description

This is constant Graphics.rc3_StockIcon.

73.7.215 Graphics.StandardColorsCount Constant

The total number of standard colors available

Pascal

```
StandardColorsCount = 16;
```

C++

```
StandardColorsCount = 16;
```

File

Graphics

Description

TColor (see page 14648) can be assigned standard, extended, web and system color values. The system colors are default values for various visual component colors. Extended colors include clCream (see page 14669). The standard colors are :

Name	Meaning
clBlack (see page 14666)	Black.
clMaroon (see page 14678)	Maroon.
clOlive (see page 14682)	Olive.
clNavy (see page 14681)	Navy.
clPurple (see page 14682)	Purple.
clTeal (see page 14685)	Teal.
clGray (see page 14672)	Gray.
clSilver (see page 14683)	Silver.
clRed (see page 14683)	Red.
clLime (see page 14677)	Lime.
clYellow (see page 14734)	Yellow.
clBlue (see page 14666)	Blue.
clFuchsia (see page 14670)	Fuchsia.
clAqua (see page 14665)	Aqua (cyan).
clLtGray (see page 14677)	Light Gray.
clDkGray (see page 14670)	Dark Gray.
clWhite (see page 14732)	White.

See Also

Color (see page 14437)

ExtendedColorsCount (see page 14742)

WebColorsCount (see page 14745)

TColor (see page 14648)

73.7.216 Graphics.WebColorsCount Constant

The total number of web colors available

Pascal

```
WebColorsCount = 140;
```

C++

```
WebColorsCount = 140;
```

File

Graphics

Description

TColor (see page 14648) can be assigned standard, extended, web and system color values. The system colors are default values for various visual component colors. Web colors are those defined as uniformly displayable across the Internet - they represent the Internet standard colors.

See Also

Color (see page 14437)

ExtendedColorsCount (see page 14742)

StandardColorsCount (see page 14744)

TColor (see page 14648)

74 GraphUtil Namespace

This is namespace GraphUtil.

Topics

Name	Description
Functions (see page 14749)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 14756)	The following table lists structs, records, enums in this documentation.
Types (see page 14759)	The following table lists types in this documentation.
Constants (see page 14760)	The following table lists constants in this documentation.

Constants

Name	Description
WebNamedColors (see page 14760)	This is constant GraphUtil.WebNamedColors.
WebNamedColorsCount (see page 14762)	This is constant GraphUtil.WebNamedColorsCount.

Functions

	Name	Description
	ColorAdjustLuma (see page 14749)	This is function GraphUtil.ColorAdjustLuma.
	ColorHLSToRGB (see page 14749)	This is function GraphUtil.ColorHLSToRGB.
	ColorRGBToHLS (see page 14750)	This is function GraphUtil.ColorRGBToHLS.
	ColorToWebColorName (see page 14750)	This is function GraphUtil.ColorToWebColorName.
	ColorToWebColorStr (see page 14750)	This is function GraphUtil.ColorToWebColorStr.
	DrawArrow (see page 14751)	This is function GraphUtil.DrawArrow.
	DrawCheck (see page 14751)	This is function GraphUtil.DrawCheck.
	DrawChevron (see page 14751)	This is function GraphUtil.DrawChevron.
	GetHighLightColor (see page 14752)	This is function GraphUtil.GetHighLightColor.
	GetShadowColor (see page 14752)	This is function GraphUtil.GetShadowColor.
	GradientFillCanvas (see page 14752)	This is function GraphUtil.GradientFillCanvas.
	RGBToWebColorName (see page 14753)	This is function GraphUtil.RGBToWebColorName.
	RGBToWebColorStr (see page 14753)	This is function GraphUtil.RGBToWebColorStr.
	ScaleImage (see page 14753)	This is function GraphUtil.ScaleImage.
	SortColorArray (see page 14754)	This is function GraphUtil.SortColorArray.
	WebColorNameToColor (see page 14754)	This is function GraphUtil.WebColorNameToColor.
	WebColorStrToColor (see page 14754)	This is function GraphUtil.WebColorStrToColor.
	WebColorToRGB (see page 14755)	This is function GraphUtil.WebColorToRGB.

Structs, Records, Enums

	Name	Description
	TArrowType (see page 14756)	This is record GraphUtil.TArrowType.
	TColorArraySortType (see page 14756)	This is record GraphUtil.TColorArraySortType.
	TGradientDirection (see page 14757)	This is record GraphUtil.TGradientDirection.
	TScrollDirection (see page 14757)	This is record GraphUtil.TScrollDirection.

Types

Name	Description
TColorArray (see page 14759)	This is type GraphUtil.TColorArray.

74.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	ColorAdjustLuma (see page 14749)	This is function GraphUtil.ColorAdjustLuma.
	ColorHLSToRGB (see page 14749)	This is function GraphUtil.ColorHLSToRGB.
	ColorRGBToHLS (see page 14750)	This is function GraphUtil.ColorRGBToHLS.
	ColorToWebColorName (see page 14750)	This is function GraphUtil.ColorToWebColorName.
	ColorToWebColorStr (see page 14750)	This is function GraphUtil.ColorToWebColorStr.
	DrawArrow (see page 14751)	This is function GraphUtil.DrawArrow.
	DrawCheck (see page 14751)	This is function GraphUtil.DrawCheck.
	DrawChevron (see page 14751)	This is function GraphUtil.DrawChevron.
	GetHighLightColor (see page 14752)	This is function GraphUtil.GetHighLightColor.
	GetShadowColor (see page 14752)	This is function GraphUtil.GetShadowColor.
	GradientFillCanvas (see page 14752)	This is function GraphUtil.GradientFillCanvas.
	RGBToWebColorName (see page 14753)	This is function GraphUtil.RGBToWebColorName.
	RGBToWebColorStr (see page 14753)	This is function GraphUtil.RGBToWebColorStr.
	ScaleImage (see page 14753)	This is function GraphUtil.ScaleImage.
	SortColorArray (see page 14754)	This is function GraphUtil.SortColorArray.
	WebColorNameToColor (see page 14754)	This is function GraphUtil.WebColorNameToColor.
	WebColorStrToColor (see page 14754)	This is function GraphUtil.WebColorStrToColor.
	WebColorToRGB (see page 14755)	This is function GraphUtil.WebColorToRGB.

74.1.1 GraphUtil.ColorAdjustLuma Function

Pascal

```
function ColorAdjustLuma(clrRGB: TColor; n: Integer; fScale: BOOL): TColor;
```

C++

```
TColor ColorAdjustLuma(TColor clrRGB, int n, BOOL fScale);
```

File

GraphUtil

Description

This is function GraphUtil.ColorAdjustLuma.

74.1.2 GraphUtil.ColorHLSToRGB Function

Pascal

```
function ColorHLSToRGB(Hue: Word; Luminance: Word; Saturation: Word): TColorRef;
```

C++

```
TColorRef ColorHLSToRGB(Word Hue, Word Luminance, Word Saturation);
```

File

GraphUtil

Description

This is function GraphUtil.ColorHLSToRGB.

74.1.3 GraphUtil.ColorRGBToHLS Function

Pascal

```
procedure ColorRGBToHLS(clrRGB: TColorRef; var Hue: Word; var Luminance: Word; var Saturation:
Word);
```

C++

```
ColorRGBToHLS(TColorRef clrRGB, Word Hue, Word Luminance, Word Saturation);
```

File

GraphUtil

Description

This is function GraphUtil.ColorRGBToHLS.

74.1.4 GraphUtil.ColorToWebColorName Function

Pascal

```
function ColorToWebColorName(Color: TColor): string;
```

C++

```
AnsiString ColorToWebColorName(TColor Color);
```

File

GraphUtil

Description

This is function GraphUtil.ColorToWebColorName.

74.1.5 GraphUtil.ColorToWebColorStr Function

Pascal

```
function ColorToWebColorStr(Color: TColor): string;
```

C++

```
AnsiString ColorToWebColorStr(TColor Color);
```

File

GraphUtil

Description

This is function GraphUtil.ColorToWebColorStr.

74.1.6 GraphUtil.DrawArrow Function

Pascal

```
procedure DrawArrow(ACanvas: TCanvas; Direction: TScrollDirection; Location: TPoint; Size:
Integer);
```

C++

```
DrawArrow(TCanvas ACanvas, TScrollDirection Direction, TPoint Location, int Size);
```

File

GraphUtil

Description

This is function GraphUtil.DrawArrow.

74.1.7 GraphUtil.DrawCheck Function

Pascal

```
procedure DrawCheck(ACanvas: TCanvas; Location: TPoint; Size: Integer; Shadow: Boolean = True);
```

C++

```
DrawCheck(TCanvas ACanvas, TPoint Location, int Size, Boolean Shadow = True);
```

File

GraphUtil

Description

This is function GraphUtil.DrawCheck.

74.1.8 GraphUtil.DrawChevron Function

Pascal

```
procedure DrawChevron(ACanvas: TCanvas; Direction: TScrollDirection; Location: TPoint; Size:
Integer);
```

C++

```
DrawChevron(TCanvas ACanvas, TScrollDirection Direction, TPoint Location, int Size);
```

File

GraphUtil

Description

This is function GraphUtil.DrawChevron.

74.1.9 GraphUtil.GetHighLightColor Function

Pascal

```
function GetHighLightColor(const Color: TColor; Luminance: Integer = 19): TColor;
```

C++

```
TColor GetHighLightColor(const TColor Color, int Luminance = 19);
```

File

GraphUtil

Description

This is function GraphUtil.GetHighLightColor.

74.1.10 GraphUtil.GetShadowColor Function

Pascal

```
function GetShadowColor(const Color: TColor; Luminance: Integer = -50): TColor;
```

C++

```
TColor GetShadowColor(const TColor Color, int Luminance = -50);
```

File

GraphUtil

Description

This is function GraphUtil.GetShadowColor.

74.1.11 GraphUtil.GradientFillCanvas Function

Pascal

```
procedure GradientFillCanvas(const ACanvas: TCanvas; const AStartColor: TColor; const
AEndColor: TColor; const ARect: TRect; const Direction: TGradientDirection);
```

C++

```
GradientFillCanvas(const TCanvas ACanvas, const TColor AStartColor, const TColor AEndColor,
const TRect ARect, const TGradientDirection Direction);
```

File

GraphUtil

Description

This is function GraphUtil.GradientFillCanvas.

74.1.12 GraphUtil.RGBToWebColorName Function

Pascal

```
function RGBToWebColorName(RGB: Integer): string;
```

C++

```
AnsiString RGBToWebColorName(int RGB);
```

File

GraphUtil

Description

This is function GraphUtil.RGBToWebColorName.

74.1.13 GraphUtil.RGBToWebColorStr Function

Pascal

```
function RGBToWebColorStr(RGB: Integer): string;
```

C++

```
AnsiString RGBToWebColorStr(int RGB);
```

File

GraphUtil

Description

This is function GraphUtil.RGBToWebColorStr.

74.1.14 GraphUtil.ScaleImage Function

Pascal

```
procedure ScaleImage(const SourceBitmap: TBitmap; const ResizedBitmap: TBitmap; const
ScaleAmount: Double);
```

C++

```
ScaleImage(const TBitmap SourceBitmap, const TBitmap ResizedBitmap, const Double ScaleAmount);
```

File

GraphUtil

Description

This is function GraphUtil.ScaleImage.

74.1.15 GraphUtil.SortColorArray Function

Pascal

```
procedure SortColorArray(ColorArray: TColorArray; L: Integer; R: Integer; SortType:
TColorArraySortType; Reverse: Boolean = False);
```

C++

```
SortColorArray(TColorArray ColorArray, int L, int R, TColorArraySortType SortType, Boolean
Reverse = False);
```

File

GraphUtil

Description

This is function GraphUtil.SortColorArray.

74.1.16 GraphUtil.WebColorNameToColor Function

Pascal

```
function WebColorNameToColor(WebColorName: string): TColor;
```

C++

```
TColor WebColorNameToColor(AnsiString WebColorName);
```

File

GraphUtil

Description

This is function GraphUtil.WebColorNameToColor.

74.1.17 GraphUtil.WebColorStrToColor Function

Pascal

```
function WebColorStrToColor(WebColor: string): TColor;
```

C++

```
TColor WebColorStrToColor(AnsiString WebColor);
```

File

GraphUtil

Description

This is function GraphUtil.WebColorStrToColor.

74.1.18 GraphUtil.WebColorToRGB Function

Pascal

```
function WebColorToRGB(WebColor: Integer): Integer;
```

C++

```
int WebColorToRGB(int WebColor);
```

File

GraphUtil

Description

This is function GraphUtil.WebColorToRGB.

74.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TArrowType (see page 14756)	This is record GraphUtil.TArrowType.
	TColorArraySortType (see page 14756)	This is record GraphUtil.TColorArraySortType.
	TGradientDirection (see page 14757)	This is record GraphUtil.TGradientDirection.
	TScrollDirection (see page 14757)	This is record GraphUtil.TScrollDirection.

74.2.1 GraphUtil.TArrowType Enumeration

Pascal

```
TArrowType = (
  atSolid,
  atArrows
);
```

C++

```
enum TArrowType {
  atSolid,
  atArrows
};
```

File

GraphUtil

Description

This is record GraphUtil.TArrowType.

74.2.2 GraphUtil.TColorArraySortType Enumeration

Pascal

```
TColorArraySortType = (
  stHue,
  stSaturation,
  stLuminance,
  stRed,
  stGreen,
  stBlue,
  stCombo
);
```

C++

```
enum TColorArraySortType {
  stHue,
  stSaturation,
```

```
    stLuminance,
    stRed,
    stGreen,
    stBlue,
    stCombo
  };
```

File

GraphUtil

Description

This is record GraphUtil.TColorArraySortType.

74.2.3 GraphUtil.TGradientDirection Enumeration

Pascal

```
  TGradientDirection = (
    gdHorizontal,
    gdVertical
  );
```

C++

```
  enum TGradientDirection {
    gdHorizontal,
    gdVertical
  };
```

File

GraphUtil

Description

This is record GraphUtil.TGradientDirection.

74.2.4 GraphUtil.TScrollDirection Enumeration

Pascal

```
  TScrollDirection = (
    sdLeft,
    sdRight,
    sdUp,
    sdDown
  );
```

C++

```
  enum TScrollDirection {
    sdLeft,
    sdRight,
    sdUp,
    sdDown
  };
```

File

GraphUtil

Description

This is record GraphUtil.TScrollDirection.

74.3 Types

The following table lists types in this documentation.

Types

Name	Description
TColorArray (see page 14759)	This is type GraphUtil.TColorArray.

74.3.1 GraphUtil.TColorArray Type

Pascal

```
TColorArray = array of TIdentMapEntry;
```

C++

```
array of TIdentMapEntry TColorArray;
```

File

GraphUtil

Description

This is type GraphUtil.TColorArray.

74.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
WebNamedColors (see page 14760)	This is constant GraphUtil.WebNamedColors.
WebNamedColorsCount (see page 14762)	This is constant GraphUtil.WebNamedColorsCount.

74.4.1 GraphUtil.WebNamedColors Constant

Pascal

```
WebNamedColors: array[0..WebNamedColorsCount - 1] of TIdentMapEntry = ( (Value: clWebSnow;
Name: 'clWebSnow'), (Value: clWebFloralWhite; Name: 'clWebFloralWhite'), (Value:
clWebLavenderBlush; Name: 'clWebLavenderBlush'), (Value: clWebOldLace; Name: 'clWebOldLace'),
(Value: clWebIvory; Name: 'clWebIvory'), (Value: clWebCornSilk; Name: 'clWebCornSilk'),
(Value: clWebBeige; Name: 'clWebBeige'), (Value: clWebAntiqueWhite; Name:
'clWebAntiqueWhite'), (Value: clWebWheat; Name: 'clWebWheat'), (Value: clWebAliceBlue; Name:
'clWebAliceBlue'), (Value: clWebGhostWhite; Name: 'clWebGhostWhite'), (Value: clWebLavender;
Name: 'clWebLavender'), (Value: clWebSeashell; Name: 'clWebSeashell'), (Value:
clWebLightYellow; Name: 'clWebLightYellow'), (Value: clWebPapayaWhip; Name:
'clWebPapayaWhip'), (Value: clWebNavajoWhite; Name: 'clWebNavajoWhite'), (Value:
clWebMoccasin; Name: 'clWebMoccasin'), (Value: clWebBurlywood; Name: 'clWebBurlywood'),
(Value: clWebAzure; Name: 'clWebAzure'), (Value: clWebMintcream; Name: 'clWebMintcream'),
(Value: clWebHoneydew; Name: 'clWebHoneydew'), (Value: clWebLinen; Name: 'clWebLinen'),
(Value: clWebLemonChiffon; Name: 'clWebLemonChiffon'), (Value: clWebBlanchedAlmond; Name:
'clWebBlanchedAlmond'), (Value: clWebBisque; Name: 'clWebBisque'), (Value: clWebPeachPuff;
Name: 'clWebPeachPuff'), (Value: clWebTan; Name: 'clWebTan'), (Value: clWebYellow; Name:
'clWebYellow'), (Value: clWebDarkOrange; Name: 'clWebDarkOrange'), (Value: clWebRed; Name:
'clWebRed'), (Value: clWebDarkRed; Name: 'clWebDarkRed'), (Value: clWebMaroon; Name:
'clWebMaroon'), (Value: clWebIndianRed; Name: 'clWebIndianRed'), (Value: clWebSalmon; Name:
'clWebSalmon'), (Value: clWebCoral; Name: 'clWebCoral'), (Value: clWebGold; Name:
'clWebGold'), (Value: clWebTomato; Name: 'clWebTomato'), (Value: clWebCrimson; Name:
'clWebCrimson'), (Value: clWebBrown; Name: 'clWebBrown'), (Value: clWebChocolate; Name:
'clWebChocolate'), (Value: clWebSandyBrown; Name: 'clWebSandyBrown'), (Value:
clWebLightSalmon; Name: 'clWebLightSalmon'), (Value: clWebLightCoral; Name:
'clWebLightCoral'), (Value: clWebOrange; Name: 'clWebOrange'), (Value: clWebOrangeRed; Name:
'clWebOrangeRed'), (Value: clWebFirebrick; Name: 'clWebFirebrick'), (Value: clWebSaddleBrown;
Name: 'clWebSaddleBrown'), (Value: clWebSienna; Name: 'clWebSienna'), (Value: clWebPeru; Name:
'clWebPeru'), (Value: clWebDarkSalmon; Name: 'clWebDarkSalmon'), (Value: clWebRosyBrown; Name:
'clWebRosyBrown'), (Value: clWebPaleGoldenrod; Name: 'clWebPaleGoldenrod'), (Value:
clWebLightGoldenrodYellow; Name: 'clWebLightGoldenrodYellow'), (Value: clWebOlive; Name:
'clWebOlive'), (Value: clWebForestGreen; Name: 'clWebForestGreen'), (Value: clWebGreenYellow;
Name: 'clWebGreenYellow'), (Value: clWebChartreuse; Name: 'clWebChartreuse'), (Value:
clWebLightGreen; Name: 'clWebLightGreen'), (Value: clWebAquamarine; Name: 'clWebAquamarine'),
(Value: clWebSeaGreen; Name: 'clWebSeaGreen'), (Value: clWebGoldenRod; Name:
'clWebGoldenRod'), (Value: clWebKhaki; Name: 'clWebKhaki'), (Value: clWebOliveDrab; Name:
'clWebOliveDrab'), (Value: clWebGreen; Name: 'clWebGreen'), (Value: clWebYellowGreen; Name:
'clWebYellowGreen'), (Value: clWebLawnGreen; Name: 'clWebLawnGreen'), (Value: clWebPaleGreen;
Name: 'clWebPaleGreen'), (Value: clWebMediumAquamarine; Name: 'clWebMediumAquamarine'),
(Value: clWebMediumSeaGreen; Name: 'clWebMediumSeaGreen'), (Value: clWebDarkGoldenRod; Name:
'clWebDarkGoldenRod'), (Value: clWebDarkKhaki; Name: 'clWebDarkKhaki'), (Value:
clWebDarkOliveGreen; Name: 'clWebDarkOliveGreen'), (Value: clWebDarkgreen; Name:
'clWebDarkgreen'), (Value: clWebLimeGreen; Name: 'clWebLimeGreen'), (Value: clWebLime; Name:
'clWebLime'), (Value: clWebSpringGreen; Name: 'clWebSpringGreen'), (Value:
```

```
clWebMediumSpringGreen; Name: 'clWebMediumSpringGreen'), (Value: clWebDarkSeaGreen; Name:
'clWebDarkSeaGreen'), (Value: clWebLightSeaGreen; Name: 'clWebLightSeaGreen'), (Value:
clWebPaleTurquoise; Name: 'clWebPaleTurquoise'), (Value: clWebLightCyan; Name:
'clWebLightCyan'), (Value: clWebLightBlue; Name: 'clWebLightBlue'), (Value: clWebLightSkyBlue;
Name: 'clWebLightSkyBlue'), (Value: clWebCornFlowerBlue; Name: 'clWebCornFlowerBlue'), (Value:
clWebDarkBlue; Name: 'clWebDarkBlue'), (Value: clWebIndigo; Name: 'clWebIndigo'), (Value:
clWebMediumTurquoise; Name: 'clWebMediumTurquoise'), (Value: clWebTurquoise; Name:
'clWebTurquoise'), (Value: clWebCyan; Name: 'clWebCyan'), (Value: clWebPowderBlue; Name:
'clWebPowderBlue'), (Value: clWebSkyBlue; Name: 'clWebSkyBlue'), (Value: clWebRoyalBlue; Name:
'clWebRoyalBlue'), (Value: clWebMediumBlue; Name: 'clWebMediumBlue'), (Value:
clWebMidnightBlue; Name: 'clWebMidnightBlue'), (Value: clWebDarkTurquoise; Name:
'clWebDarkTurquoise'), (Value: clWebCadetBlue; Name: 'clWebCadetBlue'), (Value: clWebDarkCyan;
Name: 'clWebDarkCyan'), (Value: clWebTeal; Name: 'clWebTeal'), (Value: clWebDeepSkyBlue; Name:
'clWebDeepskyBlue'), (Value: clWebDodgerBlue; Name: 'clWebDodgerBlue'), (Value: clWebBlue;
Name: 'clWebBlue'), (Value: clWebNavy; Name: 'clWebNavy'), (Value: clWebDarkViolet; Name:
'clWebDarkViolet'), (Value: clWebDarkOrchid; Name: 'clWebDarkOrchid'), (Value: clWebMagenta;
Name: 'clWebMagenta'), (Value: clWebDarkMagenta; Name: 'clWebDarkMagenta'), (Value:
clWebMediumVioletRed; Name: 'clWebMediumVioletRed'), (Value: clWebPaleVioletRed; Name:
'clWebPaleVioletRed'), (Value: clWebBlueViolet; Name: 'clWebBlueViolet'), (Value:
clWebMediumOrchid; Name: 'clWebMediumOrchid'), (Value: clWebMediumPurple; Name:
'clWebMediumPurple'), (Value: clWebPurple; Name: 'clWebPurple'), (Value: clWebDeepPink; Name:
'clWebDeepPink'), (Value: clWebLightPink; Name: 'clWebLightPink'), (Value: clWebViolet; Name:
'clWebViolet'), (Value: clWebOrchid; Name: 'clWebOrchid'), (Value: clWebPlum; Name:
'clWebPlum'), (Value: clWebThistle; Name: 'clWebThistle'), (Value: clWebHotPink; Name:
'clWebHotPink'), (Value: clWebPink; Name: 'clWebPink'), (Value: clWebLightSteelBlue; Name:
'clWebLightSteelBlue'), (Value: clWebMediumSlateBlue; Name: 'clWebMediumSlateBlue'), (Value:
clWebLightSlateGray; Name: 'clWebLightSlateGray'), (Value: clWebWhite; Name: 'clWebWhite'),
(Value: clWebLightgrey; Name: 'clWebLightgrey'), (Value: clWebGray; Name: 'clWebGray'),
(Value: clWebSteelBlue; Name: 'clWebSteelBlue'), (Value: clWebSlateBlue; Name:
'clWebSlateBlue'), (Value: clWebSlateGray; Name: 'clWebSlateGray'), (Value: clWebWhiteSmoke;
Name: 'clWebWhiteSmoke'), (Value: clWebSilver; Name: 'clWebSilver'), (Value: clWebDimGray;
Name: 'clWebDimGray'), (Value: clWebMistyRose; Name: 'clWebMistyRose'), (Value:
clWebDarkSlateBlue; Name: 'clWebDarkSlateBlue'), (Value: clWebDarkSlategray; Name:
'clWebDarkSlategray'), (Value: clWebGainsboro; Name: 'clWebGainsboro'), (Value: clWebDarkGray;
Name: 'clWebDarkGray'), (Value: clWebBlack; Name: 'clWebBlack'));
```

C++

```
array[0..WebNamedColorsCount - 1] of TIdentMapEntry WebNamedColors = ( (Value: clWebSnow;
Name: 'clWebSnow'), (Value: clWebFloralWhite; Name: 'clWebFloralWhite'), (Value:
clWebLavenderBlush; Name: 'clWebLavenderBlush'), (Value: clWebOldLace; Name: 'clWebOldLace'),
(Value: clWebIvory; Name: 'clWebIvory'), (Value: clWebCornSilk; Name: 'clWebCornSilk'),
(Value: clWebBeige; Name: 'clWebBeige'), (Value: clWebAntiqueWhite; Name:
'clWebAntiqueWhite'), (Value: clWebWheat; Name: 'clWebWheat'), (Value: clWebAliceBlue; Name:
'clWebAliceBlue'), (Value: clWebGhostWhite; Name: 'clWebGhostWhite'), (Value: clWebLavender;
Name: 'clWebLavender'), (Value: clWebSeashell; Name: 'clWebSeashell'), (Value:
clWebLightYellow; Name: 'clWebLightYellow'), (Value: clWebPapayaWhip; Name:
'clWebPapayaWhip'), (Value: clWebNavajoWhite; Name: 'clWebNavajoWhite'), (Value:
clWebMoccasin; Name: 'clWebMoccasin'), (Value: clWebBurlywood; Name: 'clWebBurlywood'),
(Value: clWebAzure; Name: 'clWebAzure'), (Value: clWebMintcream; Name: 'clWebMintcream'),
(Value: clWebHoneydew; Name: 'clWebHoneydew'), (Value: clWebLinen; Name: 'clWebLinen'),
(Value: clWebLemonChiffon; Name: 'clWebLemonChiffon'), (Value: clWebBlanchedAlmond; Name:
'clWebBlanchedAlmond'), (Value: clWebBisque; Name: 'clWebBisque'), (Value: clWebPeachPuff;
Name: 'clWebPeachPuff'), (Value: clWebTan; Name: 'clWebTan'), (Value: clWebYellow; Name:
'clWebYellow'), (Value: clWebDarkOrange; Name: 'clWebDarkOrange'), (Value: clWebRed; Name:
'clWebRed'), (Value: clWebDarkRed; Name: 'clWebDarkRed'), (Value: clWebMaroon; Name:
'clWebMaroon'), (Value: clWebIndianRed; Name: 'clWebIndianRed'), (Value: clWebSalmon; Name:
'clWebSalmon'), (Value: clWebCoral; Name: 'clWebCoral'), (Value: clWebGold; Name:
'clWebGold'), (Value: clWebTomato; Name: 'clWebTomato'), (Value: clWebCrimson; Name:
'clWebCrimson'), (Value: clWebBrown; Name: 'clWebBrown'), (Value: clWebChocolate; Name:
'clWebChocolate'), (Value: clWebSandyBrown; Name: 'clWebSandyBrown'), (Value:
clWebLightSalmon; Name: 'clWebLightSalmon'), (Value: clWebLightCoral; Name:
'clWebLightCoral'), (Value: clWebOrange; Name: 'clWebOrange'), (Value: clWebOrangeRed; Name:
'clWebOrangeRed'), (Value: clWebFirebrick; Name: 'clWebFirebrick'), (Value: clWebSaddleBrown;
Name: 'clWebSaddleBrown'), (Value: clWebSienna; Name: 'clWebSienna'), (Value: clWebPeru; Name:
'clWebPeru'), (Value: clWebDarkSalmon; Name: 'clWebDarkSalmon'), (Value: clWebRosyBrown; Name:
'clWebRosyBrown'), (Value: clWebPaleGoldenrod; Name: 'clWebPaleGoldenrod'), (Value:
```

```
clWebLightGoldenrodYellow; Name: 'clWebLightGoldenrodYellow'), (Value: clWebOlive; Name:
'clWebOlive'), (Value: clWebForestGreen; Name: 'clWebForestGreen'), (Value: clWebGreenYellow;
Name: 'clWebGreenYellow'), (Value: clWebChartreuse; Name: 'clWebChartreuse'), (Value:
clWebLightGreen; Name: 'clWebLightGreen'), (Value: clWebAquamarine; Name: 'clWebAquamarine'),
(Value: clWebSeaGreen; Name: 'clWebSeaGreen'), (Value: clWebGoldenRod; Name:
'clWebGoldenRod'), (Value: clWebKhaki; Name: 'clWebKhaki'), (Value: clWebOliveDrab; Name:
'clWebOliveDrab'), (Value: clWebGreen; Name: 'clWebGreen'), (Value: clWebYellowGreen; Name:
'clWebYellowGreen'), (Value: clWebLawnGreen; Name: 'clWebLawnGreen'), (Value: clWebPaleGreen;
Name: 'clWebPaleGreen'), (Value: clWebMediumAquamarine; Name: 'clWebMediumAquamarine'),
(Value: clWebMediumSeaGreen; Name: 'clWebMediumSeaGreen'), (Value: clWebDarkGoldenRod; Name:
'clWebDarkGoldenRod'), (Value: clWebDarkKhaki; Name: 'clWebDarkKhaki'), (Value:
clWebDarkOliveGreen; Name: 'clWebDarkOliveGreen'), (Value: clWebDarkgreen; Name:
'clWebDarkgreen'), (Value: clWebLimeGreen; Name: 'clWebLimeGreen'), (Value: clWebLime; Name:
'clWebLime'), (Value: clWebSpringGreen; Name: 'clWebSpringGreen'), (Value:
clWebMediumSpringGreen; Name: 'clWebMediumSpringGreen'), (Value: clWebDarkSeaGreen; Name:
'clWebDarkSeaGreen'), (Value: clWebLightSeaGreen; Name: 'clWebLightSeaGreen'), (Value:
clWebPaleTurquoise; Name: 'clWebPaleTurquoise'), (Value: clWebLightCyan; Name:
'clWebLightCyan'), (Value: clWebLightBlue; Name: 'clWebLightBlue'), (Value: clWebLightSkyBlue;
Name: 'clWebLightSkyBlue'), (Value: clWebCornFlowerBlue; Name: 'clWebCornFlowerBlue'), (Value:
clWebDarkBlue; Name: 'clWebDarkBlue'), (Value: clWebIndigo; Name: 'clWebIndigo'), (Value:
clWebMediumTurquoise; Name: 'clWebMediumTurquoise'), (Value: clWebTurquoise; Name:
'clWebTurquoise'), (Value: clWebCyan; Name: 'clWebCyan'), (Value: clWebPowderBlue; Name:
'clWebPowderBlue'), (Value: clWebSkyBlue; Name: 'clWebSkyBlue'), (Value: clWebRoyalBlue; Name:
'clWebRoyalBlue'), (Value: clWebMediumBlue; Name: 'clWebMediumBlue'), (Value:
clWebMidnightBlue; Name: 'clWebMidnightBlue'), (Value: clWebDarkTurquoise; Name:
'clWebDarkTurquoise'), (Value: clWebCadetBlue; Name: 'clWebCadetBlue'), (Value: clWebDarkCyan;
Name: 'clWebDarkCyan'), (Value: clWebTeal; Name: 'clWebTeal'), (Value: clWebDeepSkyBlue; Name:
'clWebDeepskyBlue'), (Value: clWebDodgerBlue; Name: 'clWebDodgerBlue'), (Value: clWebBlue;
Name: 'clWebBlue'), (Value: clWebNavy; Name: 'clWebNavy'), (Value: clWebDarkViolet; Name:
'clWebDarkViolet'), (Value: clWebDarkOrchid; Name: 'clWebDarkOrchid'), (Value: clWebMagenta;
Name: 'clWebMagenta'), (Value: clWebDarkMagenta; Name: 'clWebDarkMagenta'), (Value:
clWebMediumVioletRed; Name: 'clWebMediumVioletRed'), (Value: clWebPaleVioletRed; Name:
'clWebPaleVioletRed'), (Value: clWebBlueViolet; Name: 'clWebBlueViolet'), (Value:
clWebMediumOrchid; Name: 'clWebMediumOrchid'), (Value: clWebMediumPurple; Name:
'clWebMediumPurple'), (Value: clWebPurple; Name: 'clWebPurple'), (Value: clWebDeepPink; Name:
'clWebDeepPink'), (Value: clWebLightPink; Name: 'clWebLightPink'), (Value: clWebViolet; Name:
'clWebViolet'), (Value: clWebOrchid; Name: 'clWebOrchid'), (Value: clWebPlum; Name:
'clWebPlum'), (Value: clWebThistle; Name: 'clWebThistle'), (Value: clWebHotPink; Name:
'clWebHotPink'), (Value: clWebPink; Name: 'clWebPink'), (Value: clWebLightSteelBlue; Name:
'clWebLightSteelBlue'), (Value: clWebMediumSlateBlue; Name: 'clWebMediumSlateBlue'), (Value:
clWebLightSlateGray; Name: 'clWebLightSlateGray'), (Value: clWebWhite; Name: 'clWebWhite'),
(Value: clWebLightgrey; Name: 'clWebLightgrey'), (Value: clWebGray; Name: 'clWebGray'),
(Value: clWebSteelBlue; Name: 'clWebSteelBlue'), (Value: clWebSlateBlue; Name:
'clWebSlateBlue'), (Value: clWebSlateGray; Name: 'clWebSlateGray'), (Value: clWebWhiteSmoke;
Name: 'clWebWhiteSmoke'), (Value: clWebSilver; Name: 'clWebSilver'), (Value: clWebDimGray;
Name: 'clWebDimGray'), (Value: clWebMistyRose; Name: 'clWebMistyRose'), (Value:
clWebDarkSlateBlue; Name: 'clWebDarkSlateBlue'), (Value: clWebDarkSlategray; Name:
'clWebDarkSlategray'), (Value: clWebGainsboro; Name: 'clWebGainsboro'), (Value: clWebDarkGray;
Name: 'clWebDarkGray'), (Value: clWebBlack; Name: 'clWebBlack'));
```

File

GraphUtil

Description

This is constant GraphUtil.WebNamedColors.

74.4.2 GraphUtil.WebNamedColorsCount Constant

Pascal

```
WebNamedColorsCount = 138;
```

C++

```
WebNamedColorsCount = 138;
```

File

GraphUtil

Description

This is constant GraphUtil.WebNamedColorsCount.

75 Grids Namespace

This is namespace Grids.

Topics

Name	Description
Classes (see page 14767)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 14997)	The following table lists structs, records, enums in this documentation.
Types (see page 15003)	The following table lists types in this documentation.
Constants (see page 15009)	The following table lists constants in this documentation.

Classes

	Name	Description
	EInvalidGridOperation (see page 14767)	EInvalidGridOperation is the exception class for illegal grid operations.
	TCustomDrawGrid (see page 14768)	TCustomDrawGrid is the base class for grids that publish events for painting and manipulating the cells in the grid.
	TCustomGrid (see page 14800)	TCustomGrid is the base type for all components that present information in a two-dimensional grid.
	TDrawGrid (see page 14860)	TDrawGrid represents a grid control that displays information in column and row format.
	TInplaceEdit (see page 14938)	TInplaceEdit provides text editing capabilities for grid controls.
	TInplaceEditList (see page 14959)	This is class Grids.TInplaceEditList.
	TStringGrid (see page 14968)	TStringGrid represents a grid control designed to simplify the handling of strings and associated objects.
	TStringGridStrings (see page 14988)	TStringGridStrings maintains a list of the strings that appear in the cells of a string grid.

Constants

Name	Description
MaxCustomExtents (see page 15009)	This is constant Grids.MaxCustomExtents.
MaxShortInt (see page 15009)	This is constant Grids.MaxShortInt.

Structs, Records, Enums

	Name	Description
	TEditStyle (see page 14997)	TEditStyle indicates whether a cell in a grid contains a drop-down arrow or ellipsis button.
	TGridAxisDrawInfo (see page 14998)	TGridAxisDrawInfo and TGridDrawInfo (see page 14999) represent information that is useful when rendering a grid.
	TGridCoord (see page 14998)	TGridCoord specifies a cell in a grid control.

	TGridDrawInfo (see page 14999)	This is record Grids.TGridDrawInfo.
	TGridOption (see page 14999)	TGridOption and TGridOptions (see page 15005) customize the appearance and behavior of a grid control.
	TGridRect (see page 15001)	TGridRect specifies a set of cells that occupy a rectangular region in a grid control.
	TGridState (see page 15001)	TGridState indicates any user actions that are currently changing the layout of the grid.

Types

Name	Description
TDrawCellEvent (see page 15003)	Function type for OnDrawCell event.
TGetEditEvent (see page 15004)	TGetEditEvent is the type of the event handlers that provide strings for the inplace editor of a grid.
TGetExtentsFunc (see page 15004)	This is type Grids.TGetExtentsFunc.
TGridDrawState (see page 15004)	TGridDrawState represents information that is useful when rendering the cell of a grid.
TGridMovement (see page 15005)	This is type Grids.TGridMovement.
TGridOptions (see page 15005)	TGridOptions and TGridOption (see page 14999) customize the appearance and behavior of a grid control.
TGridScrollDirection (see page 15006)	TGridScrollDirection indicates the direction in which to scroll the cells of a grid control.
TMovedEvent (see page 15007)	This is type Grids.TMovedEvent.
TOnGetPickListItems (see page 15007)	This is type Grids.TOnGetPickListItems.
TSelectCellEvent (see page 15007)	This is type Grids.TSelectCellEvent.
TSetEditEvent (see page 15008)	TSetEditEvent is the type of the event handlers that assign strings to a cell in a grid.

75.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EInvalidGridOperation (see page 14767)	EInvalidGridOperation is the exception class for illegal grid operations.
	TCustomDrawGrid (see page 14768)	TCustomDrawGrid is the base class for grids that publish events for painting and manipulating the cells in the grid.
	TCustomGrid (see page 14800)	TCustomGrid is the base type for all components that present information in a two-dimensional grid.
	TDrawGrid (see page 14860)	TDrawGrid represents a grid control that displays information in column and row format.
	TInplaceEdit (see page 14938)	TInplaceEdit provides text editing capabilities for grid controls.
	TInplaceEditList (see page 14959)	This is class Grids.TInplaceEditList.
	TStringGrid (see page 14968)	TStringGrid represents a grid control designed to simplify the handling of strings and associated objects.
	TStringGridStrings (see page 14988)	TStringGridStrings maintains a list of the strings that appear in the cells of a string grid.

75.1.1 EInvalidGridOperation Class

EInvalidGridOperation is the exception class for illegal grid operations.

Class Hierarchy

Pascal

```
EInvalidGridOperation = class(Exception);
```

C++

```
class EInvalidGridOperation : public Exception;
```

File

Grids

Description

EInvalidGridOperation is raised when an application attempts an illegal operation on a grid. For example, EInvalidGridOperation is raised if an application tries to access a cell or column that does not exist within the grid.

See Also

TStringGrid (see page 14968)

TCustomGrid (see page 14800)

75.1.2 TCustomDrawGrid Class

TCustomDrawGrid is the base class for grids that publish events for painting and manipulating the cells in the grid.

Class Hierarchy

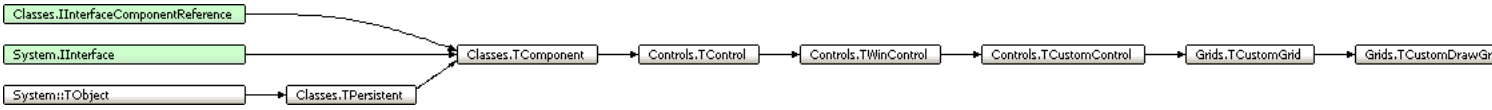

Pascal

```
TCustomDrawGrid = class(TCustomGrid);
```

C++

```
class TCustomDrawGrid : public TCustomGrid;
```

File

Grids

Description

Use TCustomDrawGrid as a base class when defining grid components that publish events for painting or manipulating the cells in the grid. TCustomDrawGrid introduces a number of events and methods that take advantage of the tabular organization of the grid in responding to user actions.

Do not create instances of TCustomDrawGrid. Instead, put a grid control in a form by instantiating a descendant of TCustomDrawGrid. Use TDrawGrid (see page 14860) to represent non-textual material in a tabular grid. For text, use a TStringGrid (see page 14968) object. If the text represents field values from a dataset, use TDBGrid. If you are displaying Name/Value pairs, use TValueListEditor.

See Also

TDrawGrid (see page 14860)

TStringGrid (see page 14968)

75.1.2.1 TCustomDrawGrid Members

The following tables list the members exposed by TCustomDrawGrid.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnColumnMoved (see page 14794)	Occurs immediately after the position of a column changes.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.

	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDrawCell (see page 14795)	Occurs when a cell in the grid needs to be drawn.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetEditMask (see page 14796)	Occurs when the in-place editor requests an edit mask.
	OnGetEditText (see page 14797)	Occurs when the in-place editor requests the value of a cell.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnRowMoved (see page 14798)	Occurs immediately after the position of a row changes.
	OnSelectCell (see page 14798)	Occurs before a cell in the grid is selected.
	OnSetEditText (see page 14800)	Occurs when the user edits the value of a cell.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnTopLeftChanged (see page 14800)	Occurs immediately after the TopRow (see page 14793) property or the LeftCol (see page 14790) property changes.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 14814)	This is DesignOptionsBoost, a member of class TCustomGrid.
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FGridState (see page 14814)	This is FGridState, a member of class TCustomGrid.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.

	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveCellExtents (see page 14814)	This is FSaveCellExtents, a member of class TCustomGrid.
	VirtualView (see page 14814)	This is VirtualView, a member of class TCustomGrid.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AdjustSize (see page 14816)	Adds or deletes rows or columns from the grid.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	BeginColumnDrag (see page 14817)	Starts the dragging of a column in the grid.
	BeginRowDrag (see page 14817)	Starts the dragging of a row in the grid.
	BoxRect (see page 14818)	Returns the screen coordinates of a specified set of cells.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcDrawInfo (see page 14818)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 14818)	Calculates the pixel locations of the fixed cells.
V	CalcSizingState (see page 14819)	Determines the appearance of the mouse cursor during resizing.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanEditAcceptKey (see page 14819)	Indicates whether the Key parameter can be entered into the current cell.
	CanEditModify (see page 14819)	Indicates whether the current cell in the grid can be edited.
V	CanEditShow (see page 14820)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 14820)	Indicates whether the grid should respond to a given key combination.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeGridOrientation (see page 14821)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CheckColumnDrag (see page 14821)	CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 14822)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
	Click (see page 5743)	Respond to user click.
V	ColumnMoved (see page 14781)	Generates an OnColumnMoved (see page 14794) event.
	ColWidthsChanged (see page 14822)	Responds when the column widths change.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.

	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateEditor (see page 14823)	Creates the inplace editor control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 14823)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
V	DefineProperties (see page 14824)	Adds the ColWidths (see page 14845) and RowHeights (see page 14855) properties to the values saved with the grid.
V	DeleteColumn (see page 14824)	Deletes a specified column from the grid.
V	DeleteRow (see page 14825)	Deletes a specified from the grid.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
V	DoExit (see page 14825)	Hides the inplace editor when selection moves to another control in a form.
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
V	DoMouseWheelDown (see page 14825)	Performs tasks when the mouse wheel is rotated downward.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
V	DoMouseWheelUp (see page 14826)	Performs tasks when the mouse wheel is rotated upward.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.

	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawCell (see page 14782)	Generates an OnDrawCell (see page 14795) event.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EndColumnDrag (see page 14827)	Ends the dragging of a column in the grid.
	EndRowDrag (see page 14827)	Ends the dragging of a row in the grid.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	FocusCell (see page 14827)	Changes the current cell to a specified cell.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEditLimit (see page 14828)	Provides the interface for a method that limits the number of characters the user can type into cells.
V	GetEditMask (see page 14782)	Generates an OnGetEditMask (see page 14796) event.
	GetEditStyle (see page 14828)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
V	GetEditText (see page 14783)	Generates an OnGetEditText (see page 14797) event.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetGridHeight (see page 14829)	Returns the value of the GridHeight (see page 14850) property.
	GetGridWidth (see page 14830)	Returns the value of the GridWidth (see page 14851) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HideEdit (see page 14830)	Hides the inplace editor.
	HideEditor (see page 14830)	Hides the inplace editor when it is not in use.
	InvalidateCell (see page 14831)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.

	InvalidateCol (see page 14831)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	InvalidateEditor (see page 14832)	Causes the inplace editor to repaint itself, moving to a new location if necessary.
	InvalidateGrid (see page 14832)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 14832)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
V	KeyDown (see page 14833)	Provides special processing when the user presses a key.
	KeyPress (see page 6163)	Respond to keyboard input.
V	KeyPress (see page 14833)	Provides special processing when the user presses a key.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
V	MouseDown (see page 14834)	Provides special processing when the user presses a mouse button.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
V	MouseMove (see page 14834)	Provides special processing when the user moves the mouse.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	MouseUp (see page 14835)	Provides special processing when the user releases the mouse button.
	MoveColRow (see page 14835)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 14836)	Moves a column to a new position.
	MoveRow (see page 14836)	Moves a row to a new position.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 5920)	Renders the image of a custom control.
V	Paint (see page 14836)	Draws the image of the grid control on the screen.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.

	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	RowHeightsChanged (see page 14837)	Responds when the row heights change.
V	RowMoved (see page 14783)	Generates an OnRowMoved (see page 14798) event.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	ScrollData (see page 14838)	Scrolls the visual image of the cells in the grid.
V	SelectCell (see page 14784)	Determines whether a particular cell in the grid can be selected.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEditText (see page 14784)	Generates an OnSetEditText (see page 14800) event.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	ShowEditor (see page 14839)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 14839)	Displays the inplace editor, with its text set to the specified character.
	SizeChanged (see page 14840)	Responds to changes in the size of the grid.
	Sizing (see page 14840)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	TimedScroll (see page 14840)	Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.
V	TopLeftChanged (see page 14784)	Generates an OnTopLeftChanged (see page 14800) event.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateDesigner (see page 14841)	Notifies the designer when rows and columns of the grid are moved or resized.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 14843)	Determines whether a single line border is drawn around the grid.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	ColCount (see page 14844)	Specifies the number of columns in the grid.
	Color (see page 5812)	Specifies the background color of the control.
	Color (see page 14844)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColWidth (see page 14846)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 14847)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 14847)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	FixedColor (see page 14848)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 14848)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 14849)	Specifies the number of rows on the top of the grid that cannot be scrolled.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GridLineWidth (see page 14850)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
R	HitTest (see page 14851)	Indicates the current position of the mouse cursor.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

R	InplaceEditor (see page 14852)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	Options (see page 14853)	Specifies various display and behavioral properties of the grid.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 14853)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	RowCount (see page 14854)	Specifies the number of rows in the grid.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ScrollBars (see page 14856)	Specifies whether the grid includes horizontal and vertical scroll bars.
	Text (see page 5842)	Contains a text string associated with the control.
R	VisibleColCount (see page 14859)	Indicates the number of scrollable columns visible in the grid.
R	VisibleRowCount (see page 14859)	Indicates the number of scrollable rows visible in the grid.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	CellRect (see page 14781)	Returns the screen coordinates of a cell in the grid.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).

	Create (see page 5918)	Creates an instance of TCustomControl (see page 5908).
	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 14813)	Creates an instance of TCustomGrid (see page 14800).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 14813)	Destroys an instance of TCustomGrid (see page 14800).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.

	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseCoord (see page 14834)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.
	MouseToCell (see page 14783)	Returns the column and row of the cell at the position with screen coordinates (X,Y).
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.

V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	Canvas (see page 14785)	Specifies the TCanvas object that presents a drawing surface for the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Col (see page 14787)	Specifies the index of the column that contains the selected cell.
	ColWidths (see page 14788)	Indicates the width (in pixels) of all the columns in the grid.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

	EditorMode (see page 14789)	Determines whether the current cell can be edited.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GridHeight (see page 14789)	Specifies the height of the grid in pixels.
	GridWidth (see page 14789)	Specifies the width of the grid in pixels.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LeftCol (see page 14790)	Specifies the index of the first visible scrollable column in the grid.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	Row (see page 14790)	Specifies the index of the row that contains the selected cell.
	RowHeights (see page 14791)	Gives the height (in pixels) of all rows in the grid.
	Selection (see page 14791)	Indicates the boundaries of the current selection.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TabStops (see page 14793)	Indicates whether the user can tab to specified columns in the grid.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TopRow (see page 14793)	Specifies the index of the first visible scrollable row in the grid.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.

	Name (see page 2805)	Specifies the name of the component as referenced in code.
	TabStop (see page 14857)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

75.1.2.2 TCustomDrawGrid Methods

The methods of the TCustomDrawGrid class are listed here.

Protected Methods

	Name	Description
	ColumnMoved (see page 14781)	Generates an OnColumnMoved (see page 14794) event.
	DrawCell (see page 14782)	Generates an OnDrawCell (see page 14795) event.
	GetEditMask (see page 14782)	Generates an OnGetEditMask (see page 14796) event.
	GetEditText (see page 14783)	Generates an OnGetEditText (see page 14797) event.
	RowMoved (see page 14783)	Generates an OnRowMoved (see page 14798) event.
	SelectCell (see page 14784)	Determines whether a particular cell in the grid can be selected.
	SetEditText (see page 14784)	Generates an OnSetEditText (see page 14800) event.
	TopLeftChanged (see page 14784)	Generates an OnTopLeftChanged (see page 14800) event.

Public Methods

	Name	Description
	CellRect (see page 14781)	Returns the screen coordinates of a cell in the grid.
	MouseToCell (see page 14783)	Returns the column and row of the cell at the position with screen coordinates (X,Y).

75.1.2.2.1 TCustomDrawGrid.CellRect Method

Returns the screen coordinates of a cell in the grid.

Pascal

```
function CellRect(ACol: Longint; ARow: Longint): TRect;
```

C++

```
__fastcall TRect CellRect(Longint ACol, Longint ARow);
```

Description

Call CellRect to obtain the boundaries, in pixels, of the cell in the column and row indicated by the ACol and ARow parameters. The top left cell is specified when ACol and ARow are set to zero.

If the indicated cell is not visible, CellRect returns an empty rectangle.

See Also

MouseToCell (see page 14783)

75.1.2.2.2 TCustomDrawGrid.ColumnMoved Method

Generates an OnColumnMoved (see page 14794) event.

Pascal

```
procedure ColumnMoved(FromIndex: Longint; ToIndex: Longint); override;
```

C++

```
virtual __fastcall ColumnMoved(Longint FromIndex, Longint ToIndex);
```

Description

ColumnMoved is called immediately after a column in the grid has moved. It overrides the inherited method to generate an OnColumnMoved (see page 14794) event. Descendants of TCustomDrawGrid (see page 14768) can override this method to make internal adjustments or to block the OnColumnMoved (see page 14794) event.

See Also

OnColumnMoved (see page 14794)

75.1.2.2.3 TCustomDrawGrid.DrawCell Method

Generates an OnDrawCell (see page 14795) event.

Pascal

```
procedure DrawCell(ACol: Longint; ARow: Longint; ARect: TRect; AState: TGridDrawState);
override;
```

C++

```
virtual __fastcall DrawCell(Longint ACol, Longint ARow, TRect ARect, TGridDrawState AState);
```

Description

DrawCell is called automatically whenever a cell in the grid is drawn. If the DefaultDrawing (see page 14847) property is true, the background of the cell is painted before DrawCell is called, and the 3D effects of the fixed cells or the focus rectangle around the focused cell are drawn after DrawCell. Descendants of TCustomDrawGrid (see page 14768) can override this method to draw the contents of the cell or to block the OnDrawCell (see page 14795) event.

See Also

DefaultDrawing (see page 14847)

FixedCols (see page 14848)

FixedRows (see page 14849)

OnDrawCell (see page 14795)

75.1.2.2.4 TCustomDrawGrid.GetEditMask Method

Generates an OnGetEditMask (see page 14796) event.

Pascal

```
function GetEditMask(ACol: Longint; ARow: Longint): string; override;
```

C++

```
virtual __fastcall AnsiString GetEditMask(Longint ACol, Longint ARow);
```

Description

The inplace editor calls GetEditMask to obtain any input mask associated with the cell specified by the ACol and ARow parameters. GetEditMask returns the Value parameter set by the OnGetEditMask (see page 14796) event handler. If there is no OnGetEditMask (see page 14796) event handler, GetEditMask returns an empty string, indicating that there is no edit mask.

See Also

OnGetEditMask (see page 14796)

75.1.2.2.5 TCustomDrawGrid.GetEditText Method

Generates an OnGetEditText (see page 14797) event.

Pascal

```
function GetEditText(ACol: Longint; ARow: Longint): string; override;
```

C++

```
virtual __fastcall AnsiString GetEditText(Longint ACol, Longint ARow);
```

Description

The inplace editor calls GetEditText to obtain the text for the indicated cell. GetEditText returns the Value parameter set by the OnGetEditText (see page 14797) event handler. If there is no OnGetEditText (see page 14797) event handler, GetEditText returns an empty string.

See Also

OnGetEditText (see page 14797)

75.1.2.2.6 TCustomDrawGrid.MouseToCell Method

Returns the column and row of the cell at the position with screen coordinates (X,Y).

Pascal

```
procedure MouseToCell(X: Integer; Y: Integer; var ACol: Longint; var ARow: Longint);
```

C++

```
__fastcall MouseToCell(int X, int Y, Longint ACol, Longint ARow);
```

Description

Call MouseToCell to convert from grid-relative screen coordinates to row and column indexes. The X and Y parameters are the screen coordinates of the point to be converted. MouseToCell returns the ACol parameter as the number of the column over the point (X,Y), and the ARow parameter as the number of the row.

Usually the MouseToCell method is used in a mouse event handler, which supplies the mouse coordinates as the X and Y parameters of the method call.

See Also

CellRect (see page 14781)

75.1.2.2.7 TCustomDrawGrid.RowMoved Method

Generates an OnRowMoved (see page 14798) event.

Pascal

```
procedure RowMoved(FromIndex: Longint; ToIndex: Longint); override;
```

C++

```
virtual __fastcall RowMoved(Longint FromIndex, Longint ToIndex);
```

Description

RowMoved is called immediately after a row in the grid changes position. It overrides the inherited method to generate an OnRowMoved (see page 14798) event. Descendants of TCustomDrawGrid (see page 14768) can override this method to make internal adjustments or to block the OnRowMoved (see page 14798) event.

See Also

OnRowMoved (see page 14798)

75.1.2.2.8 TCustomDrawGrid.SelectCell Method

Determines whether a particular cell in the grid can be selected.

Pascal

```
function SelectCell(ACol: Longint; ARow: Longint): Boolean; override;
```

C++

```
virtual __fastcall Boolean SelectCell(Longint ACol, Longint ARow);
```

Description

Grid controls call SelectCell internally before attempting to move selection to a the cell identified by the ACol and ARow parameters. ACol and ARow are the column and row indexes for the cell, where the first column has index 0, and the top row has index 0.

SelectCell generates an OnSelectCell (see page 14798) event. This method then returns true, unless the OnSelectCell (see page 14798) event handler indicates that the cell should not be selected. Descendants of TCustomDrawGrid (see page 14768) can override this method to check the properties of the information displayed in the cell or block the OnSelectCell (see page 14798) event.

See Also

OnSelectCell (see page 14798)

75.1.2.2.9 TCustomDrawGrid.SetEditText Method

Generates an OnSetEditText (see page 14800) event.

Pascal

```
procedure SetEditText(ACol: Longint; ARow: Longint; const Value: string); override;
```

C++

```
virtual __fastcall SetEditText(Longint ACol, Longint ARow, const AnsiString Value);
```

Description

SetEditText is called by the inplace editor so that the grid can store the new value for the current cell. SetEditText overrides the inherited parent class's method to generate an OnSetEditText (see page 14800) event. Descendants of TCustomDrawGrid (see page 14768) can override this method to update an internal representation of the cell's value, or to block the OnSetEditText (see page 14800) event.

See Also

OnSetEditText (see page 14800)

75.1.2.2.10 TCustomDrawGrid.TopLeftChanged Method

Generates an OnTopLeftChanged (see page 14800) event.

Pascal

```
procedure TopLeftChanged; override;
```

C++

```
virtual __fastcall TopLeftChanged();
```

Description

Grid controls call TopLeftChanged internally, whenever the LeftCol (see page 14790) or TopRow (see page 14793) property changes. After calling the inherited parent class's method to update the position of the inplace editor, TopLeftChanged generates an OnTopLeftChanged (see page 14800) event.

See Also

LeftCol (see page 14852)

OnTopLeftChanged (see page 14800)

TopRow (see page 14858)

75.1.2.3 TCustomDrawGrid Properties

The properties of the TCustomDrawGrid class are listed here.

Public Properties

	Name	Description
	Canvas (see page 14785)	Specifies the TCanvas object that presents a drawing surface for the control.
	Col (see page 14787)	Specifies the index of the column that contains the selected cell.
	ColWidths (see page 14788)	Indicates the width (in pixels) of all the columns in the grid.
	EditorMode (see page 14789)	Determines whether the current cell can be edited.
	GridHeight (see page 14789)	Specifies the height of the grid in pixels.
	GridWidth (see page 14789)	Specifies the width of the grid in pixels.
	LeftCol (see page 14790)	Specifies the index of the first visible scrollable column in the grid.
	Row (see page 14790)	Specifies the index of the row that contains the selected cell.
	RowHeights (see page 14791)	Gives the height (in pixels) of all rows in the grid.
	Selection (see page 14791)	Indicates the boundaries of the current selection.
	TabStops (see page 14793)	Indicates whether the user can tab to specified columns in the grid.
	TopRow (see page 14793)	Specifies the index of the first visible scrollable row in the grid.

75.1.2.3.1 TCustomDrawGrid.Canvas Property

Specifies the TCanvas object that presents a drawing surface for the control.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control. Canvas encapsulates a Windows device context, providing all the tools and methods needed for drawing and painting.

Canvas is a protected property that is usually redeclared as public in descendants of TCustomControl.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
```

```
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas,Rect.Left,Rect.Top,index);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(index) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TStringGrid *myStringGrid;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TStringGrid")
  {
    myStringGrid = (TStringGrid *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myStringGrid->Canvas->MoveTo(myStringGrid->ClientRect.Left, myStringGrid->ClientRect.Top);
    myStringGrid->Canvas->LineTo(myStringGrid->ClientRect.Right,
myStringGrid->ClientRect.Bottom);
    myStringGrid = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
```

```
    DrawGrid1.Canvas.Brush.Color := clWhite;
    DrawGrid1.Canvas.FillRect(Rect);
    if (gdFocused in State) then
    begin
      DrawGrid1.Canvas.DrawFocusRect(Rect);
      Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
    end;
    ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  end;

  procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  begin
    CanSelect:= True;
  end;
  {
  The following code uses ClientRect to find and draw a line
  from the top left to the bottom right of the current
  control.  Select a TControl in the ListBox list.  This
  example assumes that the current object is a descendent of
  TControl and has a public Canvas property.
  }
    if (ListBox1.Items.Objects[Index] is TStringGrid) then
    begin
      myStringGrid:= (ListBox1.Items.Objects[Index] as TStringGrid);
      with myStringGrid.ClientRect do
      begin // make sure we are using the TControl's canvas and not the form's!
        myStringGrid.Canvas.MoveTo(Left, Top);
        myStringGrid.Canvas.LineTo(Right, Bottom);
      end;
    myStringGrid:= nil;
    end;
```

See Also

Paint (see page 5920)

75.1.2.3.2 TCustomDrawGrid.Col Property

Specifies the index of the column that contains the selected cell.

Pascal

```
property Col: Longint;
```

C++

```
__property Longint Col;
```

Description

Use Col at runtime to determine the current column in the grid. Setting Col moves focus to the cell in the current row that is in the new column. The first column has an index of 0, the second column an index of 1, and so on.

The selected cell in the grid can be located by reading the Col property and the Row property to obtain the indexes of its column and row. When focus moves to the grid, the selected cell gets input focus.

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It uses a draw grid with a label above it on a form. When
the user clicks a cell in the grid, the location of the
cursor is displayed in the label caption. This example
requires a populated image list, or can just display an
empty grid.
```

```
}
procedure TForm1.DrawGrid1Click(Sender: TObject);
begin
  Label1.Caption := 'From OnClick: The cursor is in column ' +
                    IntToStr(DrawGrid1.Col + 1) +
                    ', row ' +
                    IntToStr(DrawGrid1.Row + 1);
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label2.Caption:= 'From OnDrawCell: Cell ' + InttoStr(index) + ' has the focus.';
  end;
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

Row (see page 14854)

75.1.2.3.3 TCustomDrawGrid.ColWidths Property

Indicates the width (in pixels) of all the columns in the grid.

Pascal

```
property ColWidths [Index: Longint]: Integer;
```

C++

```
__property int ColWidths[Longint Index];
```

Description

Read ColWidths to determine the width of the cells in the column specified by the Index parameter. An index of 0 refers to the first column. Set ColWidths at runtime to change the width of an individual column.

If the width of a column has not been set explicitly by resizing with the mouse, or by using the ColWidths property, its width is DefaultColWidth.

Use the ColCount property to determine the number of entries in the ColWidths array.

See Also

ColCount (see page 14844)

DefaultColWidth (see page 14846)

RowHeights (see page 14855)

75.1.2.3.4 TCustomDrawGrid.EditorMode Property

Determines whether the current cell can be edited.

Pascal

```
property EditorMode: Boolean;
```

C++

```
__property Boolean EditorMode;
```

Description

Set EditorMode to true, at runtime, to put the grid in edit mode. When EditorMode is true, the user can edit cells in the grid. When the user presses F2, EditorMode is set to true. When the user presses Enter, the value of EditorMode is toggled.

Note: EditorMode has no effect if the Options property does not include goEditing or if the Options property includes goAlwaysShowEditor.

See Also

HideEditor (see page 14830)

Options (see page 14853)

ShowEditor (see page 14839)

75.1.2.3.5 TCustomDrawGrid.GridHeight Property

Specifies the height of the grid in pixels.

Pascal

```
property GridHeight: Integer;
```

C++

```
__property int GridHeight;
```

Description

Read GridHeight to determine the size of the grid. If GridHeight is less than the value of ClientHeight, all of the rows of the grid appear in the control with an empty region below the grid. If the underlying grid is too tall to appear in the control, GridHeight is the same as ClientHeight, and the user must scroll to see the entire contents of the grid.

See Also

ClientHeight (see page 5808)

GridWidth (see page 14851)

75.1.2.3.6 TCustomDrawGrid.GridWidth Property

Specifies the width of the grid in pixels.

Pascal

```
property GridWidth: Integer;
```

C++

```
__property int GridWidth;
```

Description

Read GridWidth to determine the size of the grid. If GridWidth is less than the value of ClientWidth, all of the columns of the grid appear in the control with an empty region to the right of the grid. If the underlying grid is too wide to appear in the control, GridWidth is the same as ClientWidth, and the user must scroll to see the entire contents of the grid.

See Also

ClientWidth (see page 5811)

GridHeight (see page 14850)

75.1.2.3.7 TCustomDrawGrid.LeftCol Property

Specifies the index of the first visible scrollable column in the grid.

Pascal

```
property LeftCol: Longint;
```

C++

```
__property Longint LeftCol;
```

Description

Read LeftCol to determine the index of the first column in the scrollable region that is visible. Set LeftCol to scroll the columns in the grid so that the column with index LeftCol is the first column after the fixed columns.

See Also

FixedCols (see page 14848)

ScrollData (see page 14838)

VisibleColCount (see page 14859)

75.1.2.3.8 TCustomDrawGrid.Row Property

Specifies the index of the row that contains the selected cell.

Pascal

```
property Row: Longint;
```

C++

```
__property Longint Row;
```

Description

Use Row at runtime to determine the current row in the grid. Setting Row moves focus to the cell in the current column that is in the new row. The first row has an index of 0, the second row an index of 1, and so on.

The selected cell in the grid can be located by reading the Row property and the Col property to obtain the indexes of its row and column. When focus moves to the grid, the selected cell gets input focus.

Delphi Examples:

```
{
This example displays a string grid, a list box, and a
button on a form. When the application runs, strings are put
in the cells of row 1 of the string grid. When the user
clicks the button, The strings in the current row of the
grid are displayed in the list box.
Note that when the grid contains an empty cell, the list box
```

```
has a corresponding empty string.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.Items := StringGrid1.Rows[StringGrid1.Row];
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
    Cells[1,1] := 'Object';
    Cells[2,1] := 'Pascal';
    Cells[3,1] := 'is';
    Cells[4,1] := 'excellent';
    Cells[0,2] := 'Row 2';
    Cells[1,2] := 'Delphi';
    Cells[2,2] := 'is';
    Cells[4,2] := 'RAD';
  end;
end;
```

See Also

Col (see page 14843)

75.1.2.3.9 TCustomDrawGrid.RowHeights Property

Gives the height (in pixels) of all rows in the grid.

Pascal

```
property RowHeights [Index: Longint]: Integer;
```

C++

```
__property int RowHeights[Longint Index];
```

Description

Read RowHeights to determine the height of the cells in the row specified by the Index parameter. An index of 0 refers to the first row. Set RowWidths at runtime to change the height of an individual row.

If the height of a row has not been set explicitly by resizing with the mouse, or by using the RowHeights property, it's height is DefaultRowHeight.

Use the RowCount property to determine the number of entries in the RowHeights array.

See Also

ColWidths (see page 14845)

DefaultRowHeight (see page 14847)

RowCount (see page 14854)

75.1.2.3.10 TCustomDrawGrid.Selection Property

Indicates the boundaries of the current selection.

Pascal

```
property Selection: TGridRect;
```

C++

```
__property TGridRect Selection;
```

Description

Set Selection to select a range of cells in the grid. Selection can either represent the first column, first row, last column and last row in the grid, or the row and column coordinates of the upper left and bottom right cells in the selected region.

Selection can only indicate more than one cell when Options includes goRangeSelect.

Delphi Examples:

```
{
The following code selects the rectangle containing rows 1
through 4, and columns 2 and 3.  goRangeSelect in the
DrawGrid Options parameter must be True to select a range of
cells.  This example requires a button, a DrawGrid and a
populated ImageList.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  myRect: TGridRect;
begin
  myRect.Left := 2;
  myRect.Top := 1;
  myRect.Right := 3;
  myRect.Bottom := 4;
  DrawGrid1.Selection := myRect;
  DrawGrid1.Repaint;
end;

procedure TForm1.DrawGrid1Click(Sender: TObject);
begin
  Label1.Caption := 'From OnClick: The cursor is in column ' +
                    IntToStr(DrawGrid1.Col + 1) +
                    ', row ' +
                    IntToStr(DrawGrid1.Row + 1);
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := (ARow * DrawGrid1.ColCount + ACol) mod (DrawGrid1.ColCount + 1);
  if (gdSelected in State) then
    DrawGrid1.Canvas.Brush.Color := clYellow
  else
    DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label2.Caption:= 'From OnDrawCell: Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
```

```
end;
```

See Also

Col (see page 14843)

Options (see page 14853)

Row (see page 14854)

75.1.2.3.11 TCustomDrawGrid.TabStops Property

Indicates whether the user can tab to specified columns in the grid.

Pascal

```
property TabStops [Index: Longint]: Boolean;
```

C++

```
__property Boolean TabStops[Longint Index];
```

Description

Set TabStops to false to remove the column identified by Index from the tab order. The first column in the grid is identified by an Index of 0.

Note: When the Options property does not include goTabs, the user can't tab to any column, even if TabStops is true.

See Also

Options (see page 14853)

75.1.2.3.12 TCustomDrawGrid.TopRow Property

Specifies the index of the first visible scrollable row in the grid.

Pascal

```
property TopRow: Longint;
```

C++

```
__property Longint TopRow;
```

Description

Read TopRow to determine the index of the first row in the scrollable region that is visible. Set TopRow to scroll the rows in the grid so that the row with index TopRow is the first row after the fixed rows.

Delphi Examples:

```
{
This code uses a draw grid and a button on a form. When
the user clicks the button, the last row of the draw
grid becomes the top row.  This example requires a
button, a DrawGrid and a populated ImageList.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  DrawGrid1.TopRow := DrawGrid1.RowCount - 1;
end;

procedure TForm1.DrawGrid1Click(Sender: TObject);
begin
  Label1.Caption := 'From OnClick: The cursor is in column ' +
                    IntToStr(DrawGrid1.Col + 1) +
```

```
                        ', row ' +
                        IntToStr(DrawGrid1.Row + 1);
  end;

  procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
    Rect: TRect; State: TGridDrawState);
  var
    index: Integer;
  begin
    index := (ARow * DrawGrid1.ColCount + ACol) mod (DrawGrid1.ColCount + 1);
    if (gdSelected in State) then
      DrawGrid1.Canvas.Brush.Color := clYellow
    else
      DrawGrid1.Canvas.Brush.Color := clBackground;
    DrawGrid1.Canvas.FillRect(Rect);
    if (gdFocused in State) then
    begin
      DrawGrid1.Canvas.DrawFocusRect(Rect);
      Label2.Caption:= 'From OnDrawCell: Cell ' + InttoStr(index) + ' has the focus.';
    end;
    ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  end;

  procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  begin
    CanSelect:= True;
  end;
```

See Also

FixedRows (see page 14849)

ScrollData (see page 14838)

VisibleRowCount (see page 14859)

75.1.2.4 TCustomDrawGrid Events

The events of the TCustomDrawGrid class are listed here.

Protected Events

	Name	Description
	OnColumnMoved (see page 14794)	Occurs immediately after the position of a column changes.
	OnDrawCell (see page 14795)	Occurs when a cell in the grid needs to be drawn.
	OnGetEditMask (see page 14796)	Occurs when the in-place editor requests an edit mask.
	OnGetEditText (see page 14797)	Occurs when the in-place editor requests the value of a cell.
	OnRowMoved (see page 14798)	Occurs immediately after the position of a row changes.
	OnSelectCell (see page 14798)	Occurs before a cell in the grid is selected.
	OnSetEditText (see page 14800)	Occurs when the user edits the value of a cell.
	OnTopLeftChanged (see page 14800)	Occurs immediately after the TopRow (see page 14793) property or the LeftCol (see page 14790) property changes.

75.1.2.4.1 TCustomDrawGrid.OnColumnMoved Event

Occurs immediately after the position of a column changes.

Pascal

```
property OnColumnMoved: TMovedEvent;
```

C++

```
__property TMovedEvent OnColumnMoved;
```

Description

Use OnColumnMoved to perform special processing when the position of a grid column changes. Columns can be moved programmatically or by user manipulation. OnColumnMoved does not occur unless the Options (see page 14853) property includes goColMoving.

See Also

OnRowMoved (see page 14798)

Options (see page 14853)

75.1.2.4.2 TCustomDrawGrid.OnDrawCell Event

Occurs when a cell in the grid needs to be drawn.

Pascal

```
property OnDrawCell: TDrawCellEvent;
```

C++

```
__property TDrawCellEvent OnDrawCell;
```

Description

Write an OnDrawCell event handler to draw the contents of all the cells in the grid. Draw on the cell using the methods of the Canvas (see page 14785) property. The Rect parameter indicates the location of the cell on the canvas. The Col (see page 14787) and Row (see page 14790) parameters indicate the column and row indexes of the cell that should be drawn. The State parameter indicates whether the cell has input focus, whether the cell is selected, and whether the cell is a fixed (nonscrolling) cell.

If the OnDrawCell event handler is not assigned, all cells in the draw grid will appear empty. If the DefaultDrawing (see page 14847) property is true, the draw grid paints the background color of the cell before the OnDrawCell event, and draws a focus rectangle around the selected cell after the OnDrawCell event handler finishes drawing the contents of the cell. If the DefaultDrawing (see page 14847) property is false, the OnDrawCell event handler should paint the background of the cell and provide all visual indication of selection and focus.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas,Rect.Left,Rect.Top,index);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(index) + " has the focus.";
```

```
    }
    ImageList1->Draw(
      DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
  }
  //---------------------------------------------------------------------------
  void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
        bool &CanSelect)
  {
    CanSelect= True;
  }
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

Canvas (see page 5921)

DefaultDrawing (see page 14847)

75.1.2.4.3 TCustomDrawGrid.OnGetEditMask Event

Occurs when the in-place editor requests an edit mask.

Pascal

```
property OnGetEditMask: TGetEditEvent;
```

C++

```
__property TGetEditEvent OnGetEditMask;
```

Description

Write an OnGetEditMask event handler to provide the in-place editor with an edit mask that describes the valid input strings a user can type into a cell. Set the Value parameter to the string that describes the edit mask for the cell specified by the ACol and ARow parameters. If the cell does not have an edit mask, set the Value parameter to an empty string.

OnGetEditMask does not occur unless the Options (see page 14853) property includes goEditing.

Delphi Examples:

```
{
This example specifies an edit mask in a string grid to
limit the cells that contain telephone numbers, as indicated
by a label cell in the first row.  Include goEditing in the
TStringGrid Options property.  Set the title of one of the
columns to 'Phone Number'.  Set the DefaultColWidth large
enough to see a phone number.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  StringGrid1.Cells[2, 0]:= 'Phone Number';
end;

procedure TForm1.StringGrid1GetEditMask(Sender: TObject; ACol, ARow: Longint; var Value:
string);
begin
  if StringGrid1.Cells[ACol, 0] = 'Phone Number' then
    Value :=  '!\(999\)000-0000;1';
end;
```

See Also

Options (see page 14853)

EditMask (see page 15821)

75.1.2.4.4 TCustomDrawGrid.OnGetEditText Event

Occurs when the in-place editor requests the value of a cell.

Pascal

```
property OnGetEditText: TGetEditEvent;
```

C++

```
__property TGetEditEvent OnGetEditText;
```

Description

Write an OnGetEditText event handler to provide the in-place editor with a string representing the contents of the cell. Set the Value parameter to the string for the cell specified by the ACol and ARow parameters. When the cell enters edit mode, the contents of the cell are drawn as the Value string returned by the OnGetEditText event handler. This image need not match the appearance of the cell when it is not in edit mode, which is drawn using the OnDrawCell (see page 14795) event handler.

OnGetEditText does not occur unless the Options (see page 14853) property includes goEditing.

Delphi Examples:

```
{
The following code lowercases the text for a string grid
cell when the user begins to edit it.
}
procedure TForm1.StringGrid1GetEditText(Sender: TObject; ACol, ARow: Longint; var Value:
string);
begin
  Value := AnsiLowerCase(Value);
end;
```

See Also

OnDrawCell (see page 14795)

OnSetEditText (see page 14800)

75.1.2.4.5 TCustomDrawGrid.OnRowMoved Event

Occurs immediately after the position of a row changes.

Pascal

```
property OnRowMoved: TMovedEvent;
```

C++

```
__property TMovedEvent OnRowMoved;
```

Description

Write an OnRowMoved event handler to provide special processing whenever a row in the grid changes position. Rows can be moved programmatically or by user manipulation. OnRowMoved does not occur unless the Options (see page 14853) property includes goRowMoving.

Delphi Examples:

```
{
The following code displays the number of rows a row was
moved in a label.  To move rows in a TStringGrid, goRowMoving
must be included in the Options property.
}
procedure TForm1.StringGrid1RowMoved(Sender: TObject; FromIndex, ToIndex: Longint);
begin
  Label1.Caption := IntToStr(Abs(FromIndex-ToIndex));
end;
```

See Also

OnColumnMoved (see page 14794)

Options (see page 14853)

75.1.2.4.6 TCustomDrawGrid.OnSelectCell Event

Occurs before a cell in the grid is selected.

Pascal

```
property OnSelectCell: TSelectCellEvent;
```

C++

```
__property TSelectCellEvent OnSelectCell;
```

Description

Write an OnSelectCell event handler to specify whether any particular cell in the grid can be selected. The Col (see page 14787) and Row (see page 14790) parameters indicate the column and row indexes of the cell that is about to be selected. Set the CanSelect parameter to false to prevent the cell being selected.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
```

```
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas,Rect.Left,Rect.Top,index);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(index) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
{
The following code ensures that the user cannot select a cell unless it is empty.
}
procedure TForm1.StringGrid1SelectCell(Sender: TObject; ACol, ARow: Longint; var CanSelect:
Boolean);
begin
  CanSelect := (StringGrid1.Cells[ACol, ARow]='')
end;
```

See Also

Col (see page 14843)

Row (see page 14854)

TabStops (see page 14858)

75.1.2.4.7 TCustomDrawGrid.OnSetEditText Event

Occurs when the user edits the value of a cell.

Pascal

```
property OnSetEditText: TSetEditEvent;
```

C++

```
__property TSetEditEvent OnSetEditText;
```

Description

Write an OnSetEditText event handler to perform any special processing of the text edited by the user in an in-place editor. For example, use the OnSetEditText event to retrieve and store the value of a cell so that it can be displayed in an OnDrawCell (see page 14795) event handler. OnSetEditText occurs every time the user changes the text.

The Value parameter is the new value for the cell specified by the ACol and ARow parameters.

OnSetEditText does not occur unless the Options (see page 14853) property includes goEditing.

See Also

OnDrawCell (see page 14795)

OnGetEditText (see page 14797)

Options (see page 14853)

75.1.2.4.8 TCustomDrawGrid.OnTopLeftChanged Event

Occurs immediately after the TopRow (see page 14793) property or the LeftCol (see page 14790) property changes.

Pascal

```
property OnTopLeftChanged: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnTopLeftChanged;
```

Description

Use OnTopLeftChanged to perform special processing when the non-fixed cells in the grid are scrolled.

See Also

LeftCol (see page 14852)

TopRow (see page 14858)

75.1.3 TCustomGrid Class

TCustomGrid is the base type for all components that present information in a two-dimensional grid.

Class Hierarchy

Pascal

```
TCustomGrid = class(TCustomControl);
```

C++

```
class TCustomGrid : public TCustomControl;
```

File

Grids

Description

Use TCustomGrid as a base class when defining components that represent information in a tabular format. TCustomGrid introduces properties and methods to control the appearance and behavior of the grid, including responses to mouse actions and user editing.

Do not create instances of TCustomGrid. Instead, put a grid control in a form by instantiating a descendant of TCustomGrid. Use TDrawGrid (see page 14860) to represent non-textual material in a tabular grid. For text, use a TStringGrid (see page 14968) object. If the text represents field values from a dataset, use TDBGrid.

See Also

TDBGrid (see page 9668)

TDrawGrid (see page 14860)

TStringGrid (see page 14968)

75.1.3.1 TCustomGrid Members

The following tables list the members exposed by TCustomGrid.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 14814)	This is DesignOptionsBoost, a member of class TCustomGrid.
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FGridState (see page 14814)	This is FGridState, a member of class TCustomGrid.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveCellExtents (see page 14814)	This is FSaveCellExtents, a member of class TCustomGrid.
	VirtualView (see page 14814)	This is VirtualView, a member of class TCustomGrid.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 14816)	Adds or deletes rows or columns from the grid.

	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	BeginColumnDrag (see page 14817)	Starts the dragging of a column in the grid.
	BeginRowDrag (see page 14817)	Starts the dragging of a row in the grid.
	BoxRect (see page 14818)	Returns the screen coordinates of a specified set of cells.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcDrawInfo (see page 14818)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 14818)	Calculates the pixel locations of the fixed cells.
	CalcSizingState (see page 14819)	Determines the appearance of the mouse cursor during resizing.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanEditAcceptKey (see page 14819)	Indicates whether the Key parameter can be entered into the current cell.
	CanEditModify (see page 14819)	Indicates whether the current cell in the grid can be edited.
	CanEditShow (see page 14820)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 14820)	Indicates whether the grid should respond to a given key combination.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	CellRect (see page 14821)	Returns the screen coordinates of a cell in the grid.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeGridOrientation (see page 14821)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CheckColumnDrag (see page 14821)	CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 14822)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
	Click (see page 5743)	Respond to user click.
	ColumnMoved (see page 14822)	Provides the interface for a method that responds when the position of a column changes.
	ColWidthsChanged (see page 14822)	Responds when the column widths change.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateEditor (see page 14823)	Creates the inplace editor control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 14823)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 14824)	Adds the ColWidths (see page 14845) and RowHeights (see page 14855) properties to the values saved with the grid.
	DeleteColumn (see page 14824)	Deletes a specified column from the grid.

	DeleteRow (see page 14825)	Deletes a specified from the grid.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 14825)	Hides the inplace editor when selection moves to another control in a form.
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 14825)	Performs tasks when the mouse wheel is rotated downward.
	DoMouseWheelUp (see page 14826)	Performs tasks when the mouse wheel is rotated upward.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawCell (see page 14826)	Provides the prototype for a method that draws the contents of a cell in the grid.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EndColumnDrag (see page 14827)	Ends the dragging of a column in the grid.
	EndRowDrag (see page 14827)	Ends the dragging of a row in the grid.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	FocusCell (see page 14827)	Changes the current cell to a specified cell.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.

	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEditLimit (see page 14828)	Provides the interface for a method that limits the number of characters the user can type into cells.
	GetEditMask (see page 14828)	Provides the interface for a method that returns an input mask for the specified cell.
	GetEditStyle (see page 14828)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
	GetEditText (see page 14829)	Returns the value of the indicated cell formatted for editing.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetGridHeight (see page 14829)	Returns the value of the GridHeight (see page 14850) property.
	GetGridWidth (see page 14830)	Returns the value of the GridWidth (see page 14851) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HideEdit (see page 14830)	Hides the inplace editor.
	HideEditor (see page 14830)	Hides the inplace editor when it is not in use.
	InvalidateCell (see page 14831)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCol (see page 14831)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	InvalidateEditor (see page 14832)	Causes the inplace editor to repaint itself, moving to a new location if necessary.
	InvalidateGrid (see page 14832)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 14832)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
V	KeyDown (see page 14833)	Provides special processing when the user presses a key.
V	KeyPress (see page 14833)	Provides special processing when the user presses a key.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 14834)	Provides special processing when the user presses a mouse button.
V	MouseMove (see page 14834)	Provides special processing when the user moves the mouse.
V	MouseUp (see page 14835)	Provides special processing when the user releases the mouse button.
	MoveColRow (see page 14835)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 14836)	Moves a column to a new position.

	MoveRow (see page 14836)	Moves a row to a new position.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 14836)	Draws the image of the grid control on the screen.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	RowHeightsChanged (see page 14837)	Responds when the row heights change.
	RowMoved (see page 14837)	Provides the interface for a method that responds when the position of a row changes.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	ScrollData (see page 14838)	Scrolls the visual image of the cells in the grid.
V	SelectCell (see page 14838)	Determines whether a particular cell in the grid can be selected.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEditText (see page 14839)	Provides the interface for a method that updates the text associated with a cell.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	ShowEditor (see page 14839)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 14839)	Displays the inplace editor, with its text set to the specified character.
	SizeChanged (see page 14840)	Responds to changes in the size of the grid.
	Sizing (see page 14840)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	TimedScroll (see page 14840)	Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.
	TopLeftChanged (see page 14841)	Updates the position of the inplace editor when the grid scrolls.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateDesigner (see page 14841)	Notifies the designer when rows and columns of the grid are moved or resized.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.

	BorderStyle (see page 14843)	Determines whether a single line border is drawn around the grid.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Col (see page 14843)	Specifies the index of the column that contains the selected cell.
	ColCount (see page 14844)	Specifies the number of columns in the grid.
	Color (see page 14844)	Specifies the background color of the control.
	ColWidths (see page 14845)	Indicates the width (in pixels) of all the columns in the grid.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColWidth (see page 14846)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 14847)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 14847)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EditorMode (see page 14848)	Determines whether the current cell can be edited.
	FixedColor (see page 14848)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 14848)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 14849)	Specifies the number of rows on the top of the grid that cannot be scrolled.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
R	GridHeight (see page 14850)	Specifies the height of the grid in pixels.
	GridLineWidth (see page 14850)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
R	GridWidth (see page 14851)	Specifies the width of the grid in pixels.
R	HitTest (see page 14851)	Indicates the current position of the mouse cursor.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
R	InplaceEditor (see page 14852)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	LeftCol (see page 14852)	Specifies the index of the first visible scrollable column in the grid.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	Options (see page 14853)	Specifies various display and behavioral properties of the grid.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 14853)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	Row (see page 14854)	Specifies the index of the row that contains the selected cell.
	RowCount (see page 14854)	Specifies the number of rows in the grid.
	RowHeights (see page 14855)	Gives the height (in pixels) of all rows in the grid.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ScrollBars (see page 14856)	Specifies whether the grid includes horizontal and vertical scroll bars.
	Selection (see page 14856)	Indicates the boundaries of the current selection.
	TabStops (see page 14858)	Indicates whether the user can tab to specified columns in the grid.
	Text (see page 5842)	Contains a text string associated with the control.
	TopRow (see page 14858)	Specifies the index of the first visible scrollable row in the grid.

R	VisibleColCount (see page 14859)	Indicates the number of scrollable columns visible in the grid.
R	VisibleRowCount (see page 14859)	Indicates the number of scrollable rows visible in the grid.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 14813)	Creates an instance of TCustomGrid (see page 14800).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 14813)	Destroys an instance of TCustomGrid (see page 14800).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.

	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseCoord (see page 14834)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.

	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.

R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.

	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	TabStop (see page 14857)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

75.1.3.2 TCustomGrid.Create Constructor

Creates an instance of TCustomGrid (see page 14800).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomGrid(TComponent * AOwner);
```

Description

Call Create to create a grid control at runtime. Grid controls placed on a form at design time are created automatically.

Owner is a component that is responsible for freeing the grid and for initiating the streaming process when it is saved to a form file.

See Also

Owner (see page 2807)

75.1.3.3 TCustomGrid.Destroy Destructor

Destroys an instance of TCustomGrid (see page 14800).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomGrid();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the grid reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

75.1.3.4 TCustomGrid Fields

The fields of the TCustomGrid class are listed here.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 14814)	This is DesignOptionsBoost, a member of class TCustomGrid.
	FGridState (see page 14814)	This is FGridState, a member of class TCustomGrid.
	FSaveCellExtents (see page 14814)	This is FSaveCellExtents, a member of class TCustomGrid.
	VirtualView (see page 14814)	This is VirtualView, a member of class TCustomGrid.

75.1.3.4.1 TCustomGrid.DesignOptionsBoost Field

Pascal

```
DesignOptionsBoost: TGridOptions;
```

C++

```
TGridOptions DesignOptionsBoost;
```

Description

This is DesignOptionsBoost, a member of class TCustomGrid.

75.1.3.4.2 TCustomGrid.FGridState Field

Pascal

```
FGridState: TGridState;
```

C++

```
TGridState FGridState;
```

Description

This is FGridState, a member of class TCustomGrid.

75.1.3.4.3 TCustomGrid.FSaveCellExtents Field

Pascal

```
FSaveCellExtents: Boolean;
```

C++

```
Boolean FSaveCellExtents;
```

Description

This is FSaveCellExtents, a member of class TCustomGrid.

75.1.3.4.4 TCustomGrid.VirtualView Field

Pascal

```
VirtualView: Boolean;
```

C++

```
Boolean VirtualView;
```

Description

This is VirtualView, a member of class TCustomGrid.

75.1.3.5 TCustomGrid Methods

The methods of the TCustomGrid class are listed here.

Protected Methods

	Name	Description
	AdjustSize (see page 14816)	Adds or deletes rows or columns from the grid.
	BeginColumnDrag (see page 14817)	Starts the dragging of a column in the grid.
	BeginRowDrag (see page 14817)	Starts the dragging of a row in the grid.
	BoxRect (see page 14818)	Returns the screen coordinates of a specified set of cells.
	CalcDrawInfo (see page 14818)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 14818)	Calculates the pixel locations of the fixed cells.
V	CalcSizingState (see page 14819)	Determines the appearance of the mouse cursor during resizing.
	CanEditAcceptKey (see page 14819)	Indicates whether the Key parameter can be entered into the current cell.
	CanEditModify (see page 14819)	Indicates whether the current cell in the grid can be edited.
V	CanEditShow (see page 14820)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 14820)	Indicates whether the grid should respond to a given key combination.
	CellRect (see page 14821)	Returns the screen coordinates of a cell in the grid.
	ChangeGridOrientation (see page 14821)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	CheckColumnDrag (see page 14821)	CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 14822)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
	ColumnMoved (see page 14822)	Provides the interface for a method that responds when the position of a column changes.
	ColWidthsChanged (see page 14822)	Responds when the column widths change.
V	CreateEditor (see page 14823)	Creates the inplace editor control.
V	CreateParams (see page 14823)	Initializes the window-creation parameter record.
V	DefineProperties (see page 14824)	Adds the ColWidths (see page 14845) and RowHeights (see page 14855) properties to the values saved with the grid.
V	DeleteColumn (see page 14824)	Deletes a specified column from the grid.
V	DeleteRow (see page 14825)	Deletes a specified from the grid.
V	DoExit (see page 14825)	Hides the inplace editor when selection moves to another control in a form.
V	DoMouseWheelDown (see page 14825)	Performs tasks when the mouse wheel is rotated downward.
V	DoMouseWheelUp (see page 14826)	Performs tasks when the mouse wheel is rotated upward.
A	DrawCell (see page 14826)	Provides the prototype for a method that draws the contents of a cell in the grid.
	EndColumnDrag (see page 14827)	Ends the dragging of a column in the grid.
	EndRowDrag (see page 14827)	Ends the dragging of a row in the grid.
	FocusCell (see page 14827)	Changes the current cell to a specified cell.
	GetEditLimit (see page 14828)	Provides the interface for a method that limits the number of characters the user can type into cells.
	GetEditMask (see page 14828)	Provides the interface for a method that returns an input mask for the specified cell.
	GetEditStyle (see page 14828)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
	GetEditText (see page 14829)	Returns the value of the indicated cell formatted for editing.
	GetGridHeight (see page 14829)	Returns the value of the GridHeight (see page 14850) property.
	GetGridWidth (see page 14830)	Returns the value of the GridWidth (see page 14851) property.
	HideEdit (see page 14830)	Hides the inplace editor.

	HideEditor (see page 14830)	Hides the inplace editor when it is not in use.
	InvalidateCell (see page 14831)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCol (see page 14831)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateEditor (see page 14832)	Causes the inplace editor to repaint itself, moving to a new location if necessary.
	InvalidateGrid (see page 14832)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 14832)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
V	KeyDown (see page 14833)	Provides special processing when the user presses a key.
V	KeyPress (see page 14833)	Provides special processing when the user presses a key.
V	MouseDown (see page 14834)	Provides special processing when the user presses a mouse button.
V	MouseMove (see page 14834)	Provides special processing when the user moves the mouse.
V	MouseUp (see page 14835)	Provides special processing when the user releases the mouse button.
	MoveColRow (see page 14835)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 14836)	Moves a column to a new position.
	MoveRow (see page 14836)	Moves a row to a new position.
V	Paint (see page 14836)	Draws the image of the grid control on the screen.
	RowHeightsChanged (see page 14837)	Responds when the row heights change.
	RowMoved (see page 14837)	Provides the interface for a method that responds when the position of a row changes.
	ScrollData (see page 14838)	Scrolls the visual image of the cells in the grid.
V	SelectCell (see page 14838)	Determines whether a particular cell in the grid can be selected.
	SetEditText (see page 14839)	Provides the interface for a method that updates the text associated with a cell.
	ShowEditor (see page 14839)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 14839)	Displays the inplace editor, with its text set to the specified character.
	SizeChanged (see page 14840)	Responds to changes in the size of the grid.
	Sizing (see page 14840)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	TimedScroll (see page 14840)	Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.
	TopLeftChanged (see page 14841)	Updates the position of the inplace editor when the grid scrolls.
	UpdateDesigner (see page 14841)	Notifies the designer when rows and columns of the grid are moved or resized.

Public Methods

	Name	Description
	MouseCoord (see page 14834)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.

75.1.3.5.1 TCustomGrid.AdjustSize Method

Adds or deletes rows or columns from the grid.

Pascal

```
procedure AdjustSize(Index: Longint; Amount: Longint; Rows: Boolean); dynamic; reintroduce;
```

C++

```
__fastcall AdjustSize(Longint Index, Longint Amount, Boolean Rows);
```

Description

Grid components call AdjustSize internally to fix up the number of rows or columns when the user resizes them using the mouse. Index is the index of the row or column where the insertion or deletion should begin. Amount is the number of rows or columns to insert (Amount > 0) or delete (Amount < 0). Rows is a boolean that indicates whether the rows or the columns are changing.

See Also

DefaultColWidth (see page 14846)

DefaultRowHeight (see page 14847)

75.1.3.5.2 TCustomGrid.BeginColumnDrag Method

Starts the dragging of a column in the grid.

Pascal

```
function BeginColumnDrag(var Origin: Integer; var Destination: Integer; const MousePt:
TPoint): Boolean; dynamic;
```

C++

```
__fastcall Boolean BeginColumnDrag(int Origin, int Destination, const TPoint MousePt);
```

Description

BeginColumnDrag is called automatically when the user clicks on a column to start dragging. It enables the grid to control whether the column can be repositioned and if so, which column.

Origin is the index of the column to be moved. When BeginColumnDrag is called, this is the index of the column in which the mouse was clicked. TCustomGrid (see page 14800) does not change this value. However, descendant classes can override this method to adjust which column can be moved.

Destination is the index of where the column should move. When BeginColumnDrag is called, this is the same as Origin. TCustomGrid (see page 14800) does not change this value because the mouse has not yet moved. Descendant classes can override this method to adjust the default destination of the column.

MousePt is the coordinates of the mouse button when the user clicked.

BeginColumnDrag returns true if the drag operation is permitted, false if the drag should be disallowed.

See Also

BeginRowDrag (see page 14817)

EndColumnDrag (see page 14827)

75.1.3.5.3 TCustomGrid.BeginRowDrag Method

Starts the dragging of a row in the grid.

Pascal

```
function BeginRowDrag(var Origin: Integer; var Destination: Integer; const MousePt: TPoint):
Boolean; dynamic;
```

C++

```
__fastcall Boolean BeginRowDrag(int Origin, int Destination, const TPoint MousePt);
```

Description

Call BeginRowDrag to start a drag operation. BeginRowDrag enables the grid to control where the row can be dropped. It is exposed as a protected method so that descendants of TCustomGrid (see page 14800) can customize row dragging in a grid.

See Also

BeginColumnDrag (see page 14817)

EndRowDrag (see page 14827)

75.1.3.5.4 TCustomGrid.BoxRect Method

Returns the screen coordinates of a specified set of cells.

Pascal

```
function BoxRect(ALeft: Longint; ATop: Longint; ARight: Longint; ABottom: Longint): TRect;
```

C++

```
__fastcall TRect BoxRect(Longint ALeft, Longint ATop, Longint ARight, Longint ABottom);
```

Description

Call BoxRect to obtain the screen coordinates, in pixels, of the cells that run from the column with index ALeft to the column with index ARight and the row with index ATop to the row with index ABottom. The coordinates of cells that are not visible onscreen are not included in the final rectangle.

See Also

BoundsRect (see page 5803)

CellRect (see page 14821)

75.1.3.5.5 TCustomGrid.CalcDrawInfo Method

Calculates the pixel locations of the grid boundaries.

Pascal

```
procedure CalcDrawInfo(var DrawInfo: TGridDrawInfo);
```

C++

```
__fastcall CalcDrawInfo(TGridDrawInfo DrawInfo);
```

Description

Use CalcDrawInfo to determine the pixel locations of the major grid boundaries in the X and Y directions, including things like the first visible column/row, the last visible column/row, the pixel extent of the fixed column/row, etc.

See Also

CalcFixedInfo (see page 14818)

75.1.3.5.6 TCustomGrid.CalcFixedInfo Method

Calculates the pixel locations of the fixed cells.

Pascal

```
procedure CalcFixedInfo(var DrawInfo: TGridDrawInfo);
```

C++

```
__fastcall CalcFixedInfo(TGridDrawInfo DrawInfo);
```

Description

Use CalcDrawInfo (see page 14818) to determine the pixel locations of the fixed cells in the X and Y directions.

See Also

CalcDrawInfo (see page 14818)

75.1.3.5.7 TCustomGrid.CalcSizingState Method

Determines the appearance of the mouse cursor during resizing.

Pascal

```
procedure CalcSizingState(X: Integer; Y: Integer; var State: TGridState; var Index: Longint;
var SizingPos: Integer; var SizingOfs: Integer; var FixedInfo: TGridDrawInfo); virtual;
```

C++

```
virtual __fastcall CalcSizingState(int X, int Y, TGridState State, Longint Index, int
SizingPos, int SizingOfs, TGridDrawInfo FixedInfo);
```

Description

Use CalcSizingState to determine if the mouse cursor should be changed to a resize or drag cursor based on where the mouse is in the grid. For example, cursor over a grid line, in the fixed cell region, and column resize is enabled = resize cursor.

See Also

CalcFixedInfo (see page 14818)

75.1.3.5.8 TCustomGrid.CanEditAcceptKey Method

Indicates whether the Key parameter can be entered into the current cell.

Pascal

```
function CanEditAcceptKey(Key: Char): Boolean; dynamic;
```

C++

```
__fastcall Boolean CanEditAcceptKey(Char Key);
```

Description

CanEditAcceptKey provides a simple screening of keyboard input while the user is editing the contents of a cell. The CanEditAcceptKey method for TCustomGrid (see page 14800) always returns true. Descendants of TCustomGrid (see page 14800) override this method to screen out illegal characters.

More refined validation can be obtained by using an edit mask.

See Also

CanGridAcceptKey (see page 14820)

GetEditMask (see page 14828)

KeyPress (see page 14833)

75.1.3.5.9 TCustomGrid.CanEditModify Method

Indicates whether the current cell in the grid can be edited.

Pascal

```
function CanEditModify: Boolean; dynamic;
```

C++

```
__fastcall Boolean CanEditModify();
```

Description

Grid components use CanEditModify internally to determine whether to allow the user to edit the current cell. CanEditModify

returns true unless the inplace editor has disallowed editing for the cell.

See Also

CanEditShow (see page 14820)

InplaceEditor (see page 14852)

75.1.3.5.10 TCustomGrid.CanEditShow Method

Indicates whether the inplace edit control can be created to allow editing.

Pascal

```
function CanEditShow: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanEditShow();
```

Description

Grid controls call CanEditShow internally before creating the editor for a cell. CanEditShow is called after the grid has determined that the field can be edited by calling CanEditModify (see page 14819).

CanEditShow returns false if the Options (see page 14853) property does not indicate that the grid allows editing, or if it indicates that the grid permits row selection. Otherwise, if the Options (see page 14853) property indicates that the grid is locked in edit mode, or if the grid has input focus, CanEditShow returns true. If the grid is not locked in edit mode and the grid does not have input focus, CanEditShow returns false.

See Also

CanEditModify (see page 14819)

InplaceEditor (see page 14852)

Options (see page 14853)

75.1.3.5.11 TCustomGrid.CanGridAcceptKey Method

Indicates whether the grid should respond to a given key combination.

Pascal

```
function CanGridAcceptKey(Key: Word; Shift: TShiftState): Boolean; dynamic;
```

C++

```
__fastcall Boolean CanGridAcceptKey(Word Key, TShiftState Shift);
```

Description

Grid controls call CanGridAcceptKey internally to screen key combinations when the user first presses a key. The CanGridAcceptKey method for TCustomGrid (see page 14800) always returns true. Descendants of TCustomGrid (see page 14800) override this method to screen out irrelevant key combinations, or key combinations that indicate actions that would violate the Options (see page 14853) or mode of the grid.

See Also

CanEditAcceptKey (see page 14819)

KeyDown (see page 14833)

Options (see page 14853)

75.1.3.5.12 TCustomGrid.CellRect Method

Returns the screen coordinates of a cell in the grid.

Pascal

```
function CellRect(ACol: Longint; ARow: Longint): TRect;
```

C++

```
__fastcall TRect CellRect(Longint ACol, Longint ARow);
```

Description

Call CellRect to obtain the boundaries, in pixels, of the cell in the column and row indicated by the ACol and ARow parameters. The top left cell is specified when ACol and ARow are set to zero.

If the indicated cell is not visible, CellRect returns an empty rectangle.

See Also

BoxRect (see page 14818)

75.1.3.5.13 TCustomGrid.ChangeGridOrientation Method

Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.

Pascal

```
procedure ChangeGridOrientation(RightToLeftOrientation: Boolean);
```

C++

```
__fastcall ChangeGridOrientation(Boolean RightToLeftOrientation);
```

Description

TCustomGrid (see page 14800) uses ChangeGridOrientation internally when painting. ChangeGridOrientation sets the origin and dimensions of the grid's canvas so that painting occurs from right-to-left or from left-to-right based on the value of the RightToLeftOrientation parameter.

RightToLeftOrientation is true if the grid should prepare for painting from right to left. RightToLeftOrientation is false if the grid should restore the canvas to normal.

See Also

UseRightToLeftAlignment (see page 5790)

BiDiMode (see page 5803)

75.1.3.5.14 TCustomGrid.CheckColumnDrag Method

CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.

Pascal

```
function CheckColumnDrag(var Origin: Integer; var Destination: Integer; const MousePt:
TPoint): Boolean; dynamic;
```

C++

```
__fastcall Boolean CheckColumnDrag(int Origin, int Destination, const TPoint MousePt);
```

Description

Use CheckColumnDrag to validate that the column currently selected for dragging can be dropped at the current location.

CheckColumnDrag can change the Origin (the column being dragged) and/or the Destination (the drop target), as needed, to keep the grid's columns in order.

See Also

BeginColumnDrag (see page 14817)

EndColumnDrag (see page 14827)

75.1.3.5.15 TCustomGrid.CheckRowDrag Method

CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.

Pascal

```
function CheckRowDrag(var Origin: Integer; var Destination: Integer; const MousePt: TPoint):
Boolean; dynamic;
```

C++

```
__fastcall Boolean CheckRowDrag(int Origin, int Destination, const TPoint MousePt);
```

Description

Use CheckRowDrag to validate that the row currently selected for dragging can be dropped at the current location. CheckRowDrag can change the Origin (the row being dragged) and/or the Destination (the drop target), as needed, to keep the grid's rows in order.

See Also

BeginRowDrag (see page 14817)

EndRowDrag (see page 14827)

75.1.3.5.16 TCustomGrid.ColumnMoved Method

Provides the interface for a method that responds when the position of a column changes.

Pascal

```
procedure ColumnMoved(FromIndex: Longint; ToIndex: Longint); dynamic;
```

C++

```
__fastcall ColumnMoved(Longint FromIndex, Longint ToIndex);
```

Description

ColumnMoved is called immediately after a column in the grid has moved. The FromIndex parameter is the old index of the column, and the ToIndex parameter is the new index. The ColumnMoved method of TCustomGrid (see page 14800) does nothing. Descendants of TCustomGrid (see page 14800) override this method to make internal adjustments or to generate an OnColumnMoved event.

See Also

MoveColumn (see page 14836)

75.1.3.5.17 TCustomGrid.ColWidthsChanged Method

Responds when the column widths change.

Pascal

```
procedure ColWidthsChanged; dynamic;
```

C++

```
__fastcall ColWidthsChanged();
```

Description

ColWidthsChanged is called immediately after the column widths change. The change can result from setting the ColWidths (see page 14845) property, from setting the DefaultColWidth (see page 14846) property, from moving one of the columns, or from resizing the columns with the mouse.

ColWidthsChanged adjusts the scroll bar and the position of the inplace editor to take into account the new column size and positions of the columns.

See Also

ColWidths (see page 14845)

DefaultColWidth (see page 14846)

InplaceEditor (see page 14852)

ScrollBars (see page 14856)

75.1.3.5.18 TCustomGrid.CreateEditor Method

Creates the inplace editor control.

Pascal

```
function CreateEditor: TInplaceEdit; virtual;
```

C++

```
virtual __fastcall TInplaceEdit CreateEditor();
```

Description

Grid controls call CreateEditor internally the first time the grid goes into edit mode. CreateEditor is exposed as a protected method so that descendants of TCustomGrid (see page 14800) can substitute a specialized descendant of TInplaceEdit (see page 14938) to handle cell editing.

See Also

EditorMode (see page 14848)

InplaceEditor (see page 14852)

ShowEditor (see page 14839)

75.1.3.5.19 TCustomGrid.CreateParams Method

Initializes the window-creation parameter record.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method is called internally to specify the properties of the window that implements this control. These properties are set in the window-creation parameter record passed as Params. The fields of this record become the parameters to a call to

the CreateWindowEx API function.

After calling the inherited method, CreateParams adjusts the value of Params to request that the grid receive double click messages, and to implement the values of the ScrollBars (see page 14856) and BorderStyle (see page 14843) properties.

See Also

BorderStyle (see page 14843)

ScrollBars (see page 14856)

CreateWnd (see page 6137)

75.1.3.5.20 TCustomGrid.DefineProperties Method

Adds the ColWidths (see page 14845) and RowHeights (see page 14855) properties to the values saved with the grid.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

DefineProperties overrides the inherited method to save the ColWidths (see page 14845) and RowHeights (see page 14855) properties with the grid when they contain values other than the default widths and heights.

See Also

ColWidths (see page 14845)

DefaultColWidth (see page 14846)

DefaultRowHeight (see page 14847)

RowHeights (see page 14855)

TFiler (see page 2834)

75.1.3.5.21 TCustomGrid.DeleteColumn Method

Deletes a specified column from the grid.

Pascal

```
procedure DeleteColumn(ACol: Longint); virtual;
```

C++

```
virtual __fastcall DeleteColumn(Longint ACol);
```

Description

Call DeleteColumn to remove a column from the middle of the grid. Provide the index of the column to remove as ACol. To remove the last column from the grid, set the ColCount (see page 14844) property instead.

See Also

ColCount (see page 14844)

75.1.3.5.22 TCustomGrid.DeleteRow Method

Deletes a specified from the grid.

Pascal

```
procedure DeleteRow(ARow: Longint); virtual;
```

C++

```
virtual __fastcall DeleteRow(Longint ARow);
```

Description

Call DeleteRow to remove a row from the middle of the grid. Provide the index of the row to remove as ARow. To remove the last row from the grid, set the RowCount (see page 14854) property instead.

See Also

RowCount (see page 14854)

75.1.3.5.23 TCustomGrid.DoExit Method

Hides the inplace editor when selection moves to another control in a form.

Pascal

```
procedure DoExit; override;
```

C++

```
virtual __fastcall DoExit();
```

Description

After calling the inherited method, which generates an OnExit event, DoExit hides the inplace editor unless the Options (see page 14853) property includes goAlwaysShowEditor.

See Also

HideEditor (see page 14830)

Options (see page 14853)

OnExit (see page 6221)

75.1.3.5.24 TCustomGrid.DoMouseWheelDown Method

Performs tasks when the mouse wheel is rotated downward.

Pascal

```
function DoMouseWheelDown(Shift: TShiftState; MousePos: TPoint): Boolean; override;
```

C++

```
virtual __fastcall Boolean DoMouseWheelDown(TShiftState Shift, TPoint MousePos);
```

Description

Override DoMouseWheelDown to perform tasks when the mouse wheel is rotated downward.

DoMouseWheelDown returns false if the control's parent should handle the message.

In TCustomGrid (see page 14800), DoMouseWheelDown increments the current row and returns true.

See Also

DoMouseWheelUp (see page 14826)

Row (see page 14854)

75.1.3.5.25 TCustomGrid.DoMouseWheelUp Method

Performs tasks when the mouse wheel is rotated upward.

Pascal

```
function DoMouseWheelUp(Shift: TShiftState; MousePos: TPoint): Boolean; override;
```

C++

```
virtual __fastcall Boolean DoMouseWheelUp(TShiftState Shift, TPoint MousePos);
```

Description

Override DoMouseWheelUp to perform tasks when the mouse wheel is rotated downward.

DoMouseWheelUp should return false if the control's parent should handle the message.

In TCustomGrid (see page 14800), DoMouseWheelUp decrements the current row and returns true.

See Also

DoMouseWheelDown (see page 14825)

Row (see page 14854)

75.1.3.5.26 TCustomGrid.DrawCell Method

Provides the prototype for a method that draws the contents of a cell in the grid.

Pascal

```
procedure DrawCell(ACol: Longint; ARow: Longint; ARect: TRect; AState: TGridDrawState);
virtual; abstract;
```

C++

```
virtual __fastcall DrawCell(Longint ACol, Longint ARow, TRect ARect, TGridDrawState AState) =
0;
```

Description

Descendants of TCustomGrid (see page 14800) implement the DrawCell method to draw the contents of the cell or to generate an OnDrawCell event. If the DefaultDrawing (see page 14847) property is true, the background of the cell is painted before DrawCell is called, and the 3D effects of the fixed cells or the focus rectangle around the focused cell are drawn after DrawCell.

The ACol and ARow parameters give the column and row indexes of the cell that should be drawn. The ARect parameter gives the boundaries of the cell in the coordinates of the canvas. The AState parameter indicates whether the cell is selected, whether the cell has input focus, and whether the cell is in the fixed, non-scrolling portion of the grid.

See Also

DefaultDrawing (see page 14847)

FixedCols (see page 14848)

FixedRows (see page 14849)

Options (see page 14853)

75.1.3.5.27 TCustomGrid.EndColumnDrag Method

Ends the dragging of a column in the grid.

Pascal

```
function EndColumnDrag(var Origin: Integer; var Destination: Integer; const MousePt: TPoint):
Boolean; dynamic;
```

C++

```
__fastcall Boolean EndColumnDrag(int Origin, int Destination, const TPoint MousePt);
```

Description

Call EndColumnDrag to end a drag operation. EndColumnDrag enables the grid to control where the column can be dropped. It is exposed as a protected method so that descendants of TCustomGrid (see page 14800) can customize column dragging in a grid.

See Also

BeginColumnDrag (see page 14817)

75.1.3.5.28 TCustomGrid.EndRowDrag Method

Ends the dragging of a row in the grid.

Pascal

```
function EndRowDrag(var Origin: Integer; var Destination: Integer; const MousePt: TPoint):
Boolean; dynamic;
```

C++

```
__fastcall Boolean EndRowDrag(int Origin, int Destination, const TPoint MousePt);
```

Description

Call EndRowDrag to end a drag operation. EndRowDrag enables the grid to control where the row can be dropped. It is exposed as a protected method so that descendants of TCustomGrid (see page 14800) can customize row dragging in a grid.

See Also

BeginRowDrag (see page 14817)

75.1.3.5.29 TCustomGrid.FocusCell Method

Changes the current cell to a specified cell.

Pascal

```
procedure FocusCell(ACol: Longint; ARow: Longint; MoveAnchor: Boolean);
```

C++

```
__fastcall FocusCell(Longint ACol, Longint ARow, Boolean MoveAnchor);
```

Description

TCustomGrid (see page 14800) calls FocusCell internally to change the currently selected cell. FocusCell changes the current cell and updates the selection appropriately.

ACol is the column of the cell to select.

ARow is the row of the cell to select.

MoveAnchor determines how the selection changes. If the grid does not allow multiple cells to be selected (Options (see page 14853) does not include goRangeSelect), MoveAnchor is ignored and the specified cell becomes the selected cell. If the grid allows multiple cells to be selected, MoveAnchor determines whether the new cell becomes the starting cell (anchor) of a new selection, or whether the current selection is updated to range from the current anchor to the specified cell.

See Also

Col (see page 14843)

Row (see page 14854)

Selection (see page 14856)

Options (see page 14853)

75.1.3.5.30 TCustomGrid.GetEditLimit Method

Provides the interface for a method that limits the number of characters the user can type into cells.

Pascal

```
function GetEditLimit: Integer; dynamic;
```

C++

```
__fastcall int GetEditLimit();
```

Description

The inplace editor calls GetEditLimit to determine if a limit should be placed on the number of characters a user can type into a cell. GetEditLimit sets no limits. Descendants of TCustomGrid (see page 14800) override this method to limit the text for certain cells.

See Also

InplaceEditor (see page 14852)

75.1.3.5.31 TCustomGrid.GetEditMask Method

Provides the interface for a method that returns an input mask for the specified cell.

Pascal

```
function GetEditMask(ACol: Longint; ARow: Longint): string; dynamic;
```

C++

```
__fastcall AnsiString GetEditMask(Longint ACol, Longint ARow);
```

Description

The inplace editor calls GetEditMask to obtain any input mask associated with the cell specified by the ACol and ARow parameters. GetEditMask returns an empty string, indicating that the cell has no input mask. Descendants of TCustomGrid (see page 14800) override this method to provide an input mask or generate an OnGetEditMask event.

See Also

InplaceEditor (see page 14852)

EditMask (see page 15821)

75.1.3.5.32 TCustomGrid.GetEditStyle Method

Indicates whether a specified cell has an ellipsis button or drop-down arrow.

Pascal

```
function GetEditStyle(ACol: Longint; ARow: Longint): TEditStyle; dynamic;
```

C++

```
__fastcall TEditStyle GetEditStyle(Longint ACol, Longint ARow);
```

Description

GetEditStyle indicates whether the specified cell can only be edited directly (esSimple), whether it has an ellipsis button that users can click to launch an editor for modifying field values (esEllipsis), or whether it has a drop-down arrow that users can click to display a pick list of possible values (esPickList).

ACol is the column that contains the cell. ARow is the row that contains the cell.

In TCustomGrid (see page 14800), GetEditStyle always returns esSimple, indicating that there is no ellipsis button or drop-down arrow. Descendant classes can override this methods if they provide a way to add pick lists or ellipsis buttons to cells.

75.1.3.5.33 TCustomGrid.GetEditText Method

Returns the value of the indicated cell formatted for editing.

Pascal

```
function GetEditText(ACol: Longint; ARow: Longint): string; dynamic;
```

C++

```
__fastcall AnsiString GetEditText(Longint ACol, Longint ARow);
```

Description

The inplace editor calls GetEditText to obtain the text for the indicated cell. GetEditText returns an empty string. Descendants of TCustomGrid (see page 14800) override the GetEditText method to return a string that represents the contents of the cell indicated by the ACol and ARow parameters.

See Also

InplaceEditor (see page 14852)

75.1.3.5.34 TCustomGrid.GetGridHeight Method

Returns the value of the GridHeight (see page 14850) property.

Pascal

```
function GetGridHeight: Integer;
```

C++

```
__fastcall int GetGridHeight();
```

Description

GetGridHeight is the protected property access method for the GridHeight (see page 14850) property. It computes the value of the GridHeight (see page 14850) property based on the row heights, separator width, and the size of the control.

See Also

ClientHeight (see page 5808)

GetGridWidth (see page 14830)

GridHeight (see page 14850)

GridLineWidth (see page 14850)

RowHeights (see page 14855)

75.1.3.5.35 TCustomGrid.GetGridWidth Method

Returns the value of the GridWidth (see page 14851) property.

Pascal

```
function GetGridWidth: Integer;
```

C++

```
__fastcall int GetGridWidth();
```

Description

GetGridWidth is the protected property access method for the GridWidth (see page 14851) property. It computes the value of the GridWidth (see page 14851) property based on the column widths, separator width, and the size of the control.

See Also

ClientWidth (see page 5811)

ColWidths (see page 14845)

GetGridHeight (see page 14829)

GridWidth (see page 14851)

GridLineWidth (see page 14850)

75.1.3.5.36 TCustomGrid.HideEdit Method

Hides the inplace editor.

Pascal

```
procedure HideEdit;
```

C++

```
__fastcall HideEdit();
```

Description

The HideEditor (see page 14830) method calls the protected HideEdit method to hide the editor. This method hides the editor window after updating the text to reflect the text in the editor.

See Also

HideEditor (see page 14830)

InplaceEditor (see page 14852)

75.1.3.5.37 TCustomGrid.HideEditor Method

Hides the inplace editor when it is not in use.

Pascal

```
procedure HideEditor;
```

C++

```
__fastcall HideEditor();
```

Description

Grid controls call HideEditor when the grid goes out of edit mode. HideEditor hides the window of the inplace editor.

See Also

EditorMode (see page 14848)

InplaceEditor (see page 14852)

ShowEditor (see page 14839)

75.1.3.5.38 TCustomGrid.InvalidateCell Method

Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.

Pascal

```
procedure InvalidateCell(ACol: Longint; ARow: Longint);
```

C++

```
__fastcall InvalidateCell(Longint ACol, Longint ARow);
```

Description

Call InvalidateCell when the contents of a cell change so that the cell will need to be repainted. The ACol parameter is the column index of the cell, and the ARow parameter is the row index.

See Also

CellRect (see page 14821)

InvalidateCol (see page 14831)

InvalidateEditor (see page 14832)

InvalidateRow (see page 14832)

75.1.3.5.39 TCustomGrid.InvalidateCol Method

Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.

Pascal

```
procedure InvalidateCol(ACol: Longint);
```

C++

```
__fastcall InvalidateCol(Longint ACol);
```

Description

Call InvalidateCol when a column changes so that the area it occupies will need to be repainted. The ACol parameter is the column index.

See Also

BoxRect (see page 14818)

InvalidateCell (see page 14831)

InvalidateRow (see page 14832)

75.1.3.5.40 TCustomGrid.InvalidateEditor Method

Causes the inplace editor to repaint itself, moving to a new location if necessary.

Pascal

```
procedure InvalidateEditor;
```

C++

```
__fastcall InvalidateEditor();
```

Description

InvalidateEditor is called when the selected cell changes while the grid is in edit mode.

See Also

EditorMode (see page 14848)

HideEditor (see page 14830)

InplaceEditor (see page 14852)

InvalidateCell (see page 14831)

ShowEditor (see page 14839)

75.1.3.5.41 TCustomGrid.InvalidateGrid Method

Tells Windows to repaint the grid after other important Windows messages are handled.

Pascal

```
procedure InvalidateGrid;
```

C++

```
__fastcall InvalidateGrid();
```

Description

InvalidateGrid is used internally to ensure that the grid is repainted when its properties change. It calls the Invalidate method to indicate that the grid needs repainting.

See Also

Invalidate (see page 6161)

75.1.3.5.42 TCustomGrid.InvalidateRow Method

Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.

Pascal

```
procedure InvalidateRow(ARow: Longint);
```

C++

```
__fastcall InvalidateRow(Longint ARow);
```

Description

Call InvalidateRow when a row changes so that the area it occupies will need to be repainted. The ARow parameter is the row index.

See Also

BoxRect (see page 14818)

InvalidateCell (see page 14831)

InvalidateCol (see page 14831)

75.1.3.5.43 TCustomGrid.KeyDown Method

Provides special processing when the user presses a key.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

After generating an OnKeyDown event, KeyDown calls CanGridAcceptKey (see page 14820) to screen out problematic key combinations. KeyDown moves to the appropriate cell in response to navigational key combinations, or sets the EditorMode (see page 14848) property to true when the user presses F2.

See Also

CanGridAcceptKey (see page 14820)

EditorMode (see page 14848)

KeyPress (see page 14833)

OnKeyDown (see page 6225)

75.1.3.5.44 TCustomGrid.KeyPress Method

Provides special processing when the user presses a key.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress is called automatically after the KeyDown (see page 14833) method, but before the user releases the key.

After generating an OnKeyPress event, KeyPress toggles the EditorMode (see page 14848) property when the user presses the Enter key.

See Also

EditorMode (see page 14848)

KeyDown (see page 14833)

OnKeyPress (see page 6227)

75.1.3.5.45 TCustomGrid.MouseCoord Method

Returns the row and column indexes of the cell that contains a point specified in screen coordinates.

Pascal

```
function MouseCoord(X: Integer; Y: Integer): TGridCoord;
```

C++

```
__fastcall TGridCoord MouseCoord(int X, int Y);
```

Description

Call MouseCoord to locate the column and row of the cell which contains a given screen coordinate. Usually, MouseCoord is used to locate the cell that is under the mouse.

See Also

CellRect (see page 14821)

MouseDown (see page 14834)

MouseMove (see page 14834)

MouseUp (see page 14835)

75.1.3.5.46 TCustomGrid.MouseDown Method

Provides special processing when the user presses a mouse button.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

MouseDown hides the editor window so that it can move selection to the cell that receives the mouse click. If the mouse lands in the hot region for beginning a move or resize of a row or column, MouseDown starts the move or resize, and provides visual feedback by changing the cursor. If the editor window was hidden, it is brought back up in its new location. Finally, MouseDown generates an OnMouseDown event.

See Also

HideEditor (see page 14830)

InplaceEditor (see page 14852)

MouseMove (see page 14834)

MouseUp (see page 14835)

ShowEditor (see page 14839)

OnMouseDown (see page 5862)

75.1.3.5.47 TCustomGrid.MouseMove Method

Provides special processing when the user moves the mouse.

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

If the grid is in the middle of moving or resizing a row or column, or in the middle of selecting a range of cells using the mouse, MouseMove calculates the changes implied by the mouse move and provides visual feedback to the user. Finally, MouseMove generates an OnMouseMove event.

See Also

MouseDown (see page 14834)

MouseUp (see page 14835)

OnMouseMove (see page 5865)

75.1.3.5.48 TCustomGrid.MouseUp Method

Provides special processing when the user releases the mouse button.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

MouseUp finishes a move or resize operation on a row or column, or if the user is selecting a range of cells using the mouse, MouseUp finishes the select operation. Finally, MouseUp generates an OnMouseUp event.

See Also

MouseDown (see page 14834)

MouseMove (see page 14834)

MoveColumn (see page 14836)

MoveRow (see page 14836)

OnMouseUp (see page 5867)

75.1.3.5.49 TCustomGrid.MoveColRow Method

Selects the cell indicated by the ACol and ARow parameters.

Pascal

```
procedure MoveColRow(ACol: Longint; ARow: Longint; MoveAnchor: Boolean; Show: Boolean);
```

C++

```
__fastcall MoveColRow(Longint ACol, Longint ARow, Boolean MoveAnchor, Boolean Show);
```

Description

Grid controls call MoveColRow internally when changing the selected cell or cells. MoveColRow selects the indicated cell, but does not take care of removing any previous selection.

The ACol and ARow parameters are the column index and the row index of the cell to be selected. The MoveAnchor parameter indicates whether the new cell becomes the upper left corner of the selected region. MoveAnchor is true unless MoveColRow is called when extending a multicell selected region. The Show parameter indicates whether MoveColRow should ensure that the newly selected cell is visible.

See Also

Col (see page 14843)

Row (see page 14854)

SelectCell (see page 14838)

Selection (see page 14856)

75.1.3.5.50 TCustomGrid.MoveColumn Method

Moves a column to a new position.

Pascal

```
procedure MoveColumn(FromIndex: Longint; ToIndex: Longint);
```

C++

```
__fastcall MoveColumn(Longint FromIndex, Longint ToIndex);
```

Description

Call MoveColumn to move the column identified by the FromIndex parameter to the position specified by the ToIndex parameter.

See Also

ColumnMoved (see page 14822)

ColWidthsChanged (see page 14822)

75.1.3.5.51 TCustomGrid.MoveRow Method

Moves a row to a new position.

Pascal

```
procedure MoveRow(FromIndex: Longint; ToIndex: Longint);
```

C++

```
__fastcall MoveRow(Longint FromIndex, Longint ToIndex);
```

Description

Call MoveRow to move the row identified by the FromIndex parameter to the position specified by the ToIndex parameter.

See Also

RowHeightsChanged (see page 14837)

RowMoved (see page 14837)

75.1.3.5.52 TCustomGrid.Paint Method

Draws the image of the grid control on the screen.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Paint is called when the grid receives a paint message. Paint draws the grid, calling the DrawCell (see page 14826) method when it is time to paint the contents of each cell. If DefaultDrawing (see page 14847) is false, Paint does not draw the background of the cells, provide the raised 3D effects on the fixed cells, or draw the focus rectangle around the selected cell when it has focus.

See Also

DefaultDrawing (see page 14847)

DrawCell (see page 14826)

InvalidateCell (see page 14831)

InvalidateCol (see page 14831)

InvalidateRow (see page 14832)

75.1.3.5.53 TCustomGrid.RowHeightsChanged Method

Responds when the row heights change.

Pascal

```
procedure RowHeightsChanged; dynamic;
```

C++

```
__fastcall RowHeightsChanged();
```

Description

RowHeightsChanged is called immediately after the row heights change. The change can result from setting the RowHeights (see page 14855) property, from setting the DefaultRowHeight (see page 14847) property, from moving one of the rows, or from resizing the rows with the mouse.

RowHeightsChanged adjusts the scroll bar and the position of the inplace editor to take into account the new row size and positions of the rows.

See Also

DefaultRowHeight (see page 14847)

InplaceEditor (see page 14852)

RowHeights (see page 14855)

ScrollBars (see page 14856)

75.1.3.5.54 TCustomGrid.RowMoved Method

Provides the interface for a method that responds when the position of a row changes.

Pascal

```
procedure RowMoved(FromIndex: Longint; ToIndex: Longint); dynamic;
```

C++

```
__fastcall RowMoved(Longint FromIndex, Longint ToIndex);
```

Description

RowMoved is called immediately after a row in the grid changes position. The FromIndex parameter is the old index of the row, and the ToIndex parameter is the new index. The RowMoved method of TCustomGrid (see page 14800) does nothing. Descendants of TCustomGrid (see page 14800) override this method to make internal adjustments or to generate an OnRowMoved event.

See Also

MoveRow (see page 14836)

75.1.3.5.55 TCustomGrid.ScrollData Method

Scrolls the visual image of the cells in the grid.

Pascal

```
procedure ScrollData(DX: Integer; DY: Integer);
```

C++

```
__fastcall ScrollData(int DX, int DY);
```

Description

Grid controls call ScrollData internally when the nonfixed rows or columns scroll. ScrollData updates the image of the grid.

See Also

LeftCol (see page 14852)

TopRow (see page 14858)

75.1.3.5.56 TCustomGrid.SelectCell Method

Determines whether a particular cell in the grid can be selected.

Pascal

```
function SelectCell(ACol: Longint; ARow: Longint): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean SelectCell(Longint ACol, Longint ARow);
```

Description

Grid controls call SelectCell internally before attempting to move selection to a the cell identified by the ACol and ARow parameters. ACol and ARow are the column and row indexes for the cell, where the first column has index 0, and the top row has index 0.

The SelectCell method for TCustomGrid (see page 14800) always returns true, allowing the cell to be selected. Descendants of TCustomGrid (see page 14800) override this method to check the properties of the information displayed in the cell or generating an OnSelectCell event.

See Also

MoveColRow (see page 14835)

TabStops (see page 14858)

75.1.3.5.57 TCustomGrid.SetEditText Method

Provides the interface for a method that updates the text associated with a cell.

Pascal

```
procedure SetEditText(ACol: Longint; ARow: Longint; const Value: string); dynamic;
```

C++

```
__fastcall SetEditText(Longint ACol, Longint ARow, const AnsiString Value);
```

Description

SetEditText is called by the inplace editor so that the grid can store the new value for the current cell. The SetEditText method for TCustomGrid (see page 14800) does nothing. Descendants of TCustomGrid (see page 14800) override this method to update an internal representation of the cell's value, or to generate an OnSetEditText event.

See Also

GetEditText (see page 14829)

75.1.3.5.58 TCustomGrid.ShowEditor Method

Displays the inplace editor when the grid enters edit mode.

Pascal

```
procedure ShowEditor;
```

C++

```
__fastcall ShowEditor();
```

Description

Grid controls call ShowEditor when the grid enters edit mode. ShowEditor displays the window of the inplace editor over the currently selected cell. If the inplace editor does not yet exist, ShowEditor creates it.

See Also

CreateEditor (see page 14823)

EditorMode (see page 14848)

HideEditor (see page 14830)

InplaceEditor (see page 14852)

75.1.3.5.59 TCustomGrid.ShowEditorChar Method

Displays the inplace editor, with its text set to the specified character.

Pascal

```
procedure ShowEditorChar(Ch: Char);
```

C++

```
__fastcall ShowEditorChar(Char Ch);
```

Description

Grid controls call ShowEditorChar when the user types a character into a cell and the editor is not already showing. ShowEditorChar displays the window of the inplace editor over the currently selected cell, and sets its text property to the

character Ch. If the inplace editor does not yet exist, ShowEditorChar creates it.

See Also

CreateEditor (see page 14823)

EditorMode (see page 14848)

HideEditor (see page 14830)

InplaceEditor (see page 14852)

ShowEditor (see page 14839)

75.1.3.5.60 TCustomGrid.SizeChanged Method

Responds to changes in the size of the grid.

Pascal

```
procedure SizeChanged(OldColCount: Longint; OldRowCount: Longint); dynamic;
```

C++

```
__fastcall SizeChanged(Longint OldColCount, Longint OldRowCount);
```

Description

SizeChanged provides the interface for a method that responds to changes in the number of rows or columns in the grid. The SizeChanged method of TCustomGrid (see page 14800) does nothing. Descendants of TCustomGrid (see page 14800) override this method to make internal adjustments or generate events.

See Also

ColCount (see page 14844)

RowCount (see page 14854)

75.1.3.5.61 TCustomGrid.Sizing Method

Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.

Pascal

```
function Sizing(X: Integer; Y: Integer): Boolean;
```

C++

```
__fastcall Boolean Sizing(int X, int Y);
```

Description

Grid controls call Sizing internally with the coordinates of a mouse click to determine whether the user is about to resize a row or column.

See Also

MouseDown (see page 14834)

75.1.3.5.62 TCustomGrid.TimedScroll Method

Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.

Pascal

```
procedure TimedScroll(Direction: TGridScrollDirection); dynamic;
```

C++

```
__fastcall TimedScroll(TGridScrollDirection Direction);
```

Description

TimedScroll is called periodically as the user continues to drag with the mouse outside of the scrollable cells in the grid. TimedScroll does nothing unless the user is moving a row or column, resizing a row or column, or selecting a multiple cell region using the mouse. TimedScroll scrolls the cells in the grid to allow the operation to continue into the portion of the grid that was hidden due to scrolling.

See Also

LeftCol (see page 14852)

MouseMove (see page 14834)

TopRow (see page 14858)

VisibleColCount (see page 14859)

VisibleRowCount (see page 14859)

75.1.3.5.63 TCustomGrid.TopLeftChanged Method

Updates the position of the inplace editor when the grid scrolls.

Pascal

```
procedure TopLeftChanged; dynamic;
```

C++

```
__fastcall TopLeftChanged();
```

Description

Grid controls call TopLeftChanged internally, whenever the LeftCol (see page 14852) or TopRow (see page 14858) property changes.

Descendants of TCustomGrid (see page 14800) override this method to make additional adjustments or to generate an OnTopLeftChanged event.

See Also

InplaceEditor (see page 14852)

LeftCol (see page 14852)

TopRow (see page 14858)

75.1.3.5.64 TCustomGrid.UpdateDesigner Method

Notifies the designer when rows and columns of the grid are moved or resized.

Pascal

```
procedure UpdateDesigner;
```

C++

```
__fastcall UpdateDesigner();
```

Description

At design time, the designer automatically keeps track of property changes so that it can prompt the user to save any changes. Moving or resizing the rows and columns of the grid using the mouse bypasses this system. When such changes occur, UpdateDesigner notifies the designer that the properties of the grid have changed and that the changes should be saved.

See Also

MouseUp (see page 14835)

75.1.3.6 TCustomGrid Properties

The properties of the TCustomGrid class are listed here.

Protected Properties

	Name	Description
	BorderStyle (see page 14843)	Determines whether a single line border is drawn around the grid.
	Col (see page 14843)	Specifies the index of the column that contains the selected cell.
	ColCount (see page 14844)	Specifies the number of columns in the grid.
	Color (see page 14844)	Specifies the background color of the control.
	ColWidths (see page 14845)	Indicates the width (in pixels) of all the columns in the grid.
	DefaultColWidth (see page 14846)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 14847)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 14847)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	EditorMode (see page 14848)	Determines whether the current cell can be edited.
	FixedColor (see page 14848)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 14848)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 14849)	Specifies the number of rows on the top of the grid that cannot be scrolled.
R	GridHeight (see page 14850)	Specifies the height of the grid in pixels.
	GridLineWidth (see page 14850)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
R	GridWidth (see page 14851)	Specifies the width of the grid in pixels.
R	HitTest (see page 14851)	Indicates the current position of the mouse cursor.
R	InplaceEditor (see page 14852)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	LeftCol (see page 14852)	Specifies the index of the first visible scrollable column in the grid.
	Options (see page 14853)	Specifies various display and behavioral properties of the grid.
	ParentColor (see page 14853)	Determines where a control looks for its color information.
	Row (see page 14854)	Specifies the index of the row that contains the selected cell.
	RowCount (see page 14854)	Specifies the number of rows in the grid.
	RowHeights (see page 14855)	Gives the height (in pixels) of all rows in the grid.
	ScrollBars (see page 14856)	Specifies whether the grid includes horizontal and vertical scroll bars.
	Selection (see page 14856)	Indicates the boundaries of the current selection.
	TabStops (see page 14858)	Indicates whether the user can tab to specified columns in the grid.
	TopRow (see page 14858)	Specifies the index of the first visible scrollable row in the grid.
R	VisibleColCount (see page 14859)	Indicates the number of scrollable columns visible in the grid.
R	VisibleRowCount (see page 14859)	Indicates the number of scrollable rows visible in the grid.

Published Properties

	Name	Description
	TabStop (see page 14857)	Determines if the user can tab to a control.

75.1.3.6.1 TCustomGrid.BorderStyle Property

Determines whether a single line border is drawn around the grid.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Set BorderStyle to bsSingle to add a single line border around the grid's image. Set BorderStyle to bsNone to omit the border.

75.1.3.6.2 TCustomGrid.Col Property

Specifies the index of the column that contains the selected cell.

Pascal

```
property Col: Longint;
```

C++

```
__property Longint Col;
```

Description

Use Col at runtime to determine the current column in the grid. Setting Col moves focus to the cell in the current row that is in the new column. The first column has an index of 0, the second column an index of 1, and so on.

The selected cell in the grid can be located by reading the Col property and the Row (see page 14854) property to obtain the indexes of its column and row. When focus moves to the grid, the selected cell gets input focus.

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It uses a draw grid with a label above it on a form. When
the user clicks a cell in the grid, the location of the
cursor is displayed in the label caption. This example
requires a populated image list, or can just display an
empty grid.
}
procedure TForm1.DrawGrid1Click(Sender: TObject);
begin
  Label1.Caption := 'From OnClick: The cursor is in column ' +
                    IntToStr(DrawGrid1.Col + 1) +
                    ', row ' +
                    IntToStr(DrawGrid1.Row + 1);
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  if (gdFocused in State) then
  begin
```

```
      DrawGrid1.Canvas.DrawFocusRect(Rect);
      Label2.Caption:= 'From OnDrawCell: Cell ' + InttoStr(index) + ' has the focus.';
    end;
  end;

  procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  begin
    CanSelect:= True;
  end;
```

See Also

Row (see page 14854)

75.1.3.6.3 TCustomGrid.ColCount Property

Specifies the number of columns in the grid.

Pascal

```
property ColCount: Longint;
```

C++

```
__property Longint ColCount;
```

Description

Read ColCount to determine the number entries in the ColWidths (see page 14845) array. Set ColCount to add or delete columns at the righthand side of the grid. The value of ColCount includes any fixed columns at the left of the grid as well as the scrollable columns in the body of the grid.

Delphi Examples:

```
{
This code fills each cell with a number. The numbers are
arranged in consecutive order by rows.  Notice that even
cells in the fixed rows and columns are altered.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K : Integer;
begin
  K := 0;
  with StringGrid1 do
    for I := 0 to ColCount - 1 do
      for J:= 0 to RowCount - 1 do
        begin
          K := K + 1;
          Cells[I,J] := IntToStr(K);
        end;
end;
```

See Also

ColWidths (see page 14845)

RowCount (see page 14854)

VisibleColCount (see page 14859)

75.1.3.6.4 TCustomGrid.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

75.1.3.6.5 TCustomGrid.ColWidths Property

Indicates the width (in pixels) of all the columns in the grid.

Pascal

```
property ColWidths [Index: Longint]: Integer;
```

C++

```
__property int ColWidths[Longint Index];
```

Description

Read ColWidths to determine the width of the cells in the column specified by the Index parameter. An index of 0 refers to the first column. Set ColWidths at runtime to change the width of an individual column.

If the width of a column has not been set explicitly by resizing with the mouse, or by using the ColWidths property, its width is DefaultColWidth (see page 14846).

Use the ColCount (see page 14844) property to determine the number of entries in the ColWidths array.

See Also

ColCount (see page 14844)

DefaultColWidth (see page 14846)

RowHeights (see page 14855)

75.1.3.6.6 TCustomGrid.DefaultColWidth Property

Determines the width (in pixels) of all columns that have not been explicitly resized.

Pascal

```
property DefaultColWidth: Integer;
```

C++

```
__property int DefaultColWidth;
```

Description

Set DefaultColWidth to change the size of all columns in the grid. When DefaultColWidth is set, columns that have been resized using the mouse or by setting the ColWidths (see page 14845) property are given the DefaultColWidth as well.

When new columns are added to the grid using the ColCount (see page 14844) property, they are created with a width of DefaultColWidth.

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if DrawGrid1.DefaultColWidth > 90 then
    DrawGrid1.GridLineWidth := 2
  else
    DrawGrid1.GridLineWidth := 1;
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
```

```
  end;
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

ColCount (see page 14844)

ColWidths (see page 14845)

DefaultRowHeight (see page 14847)

75.1.3.6.7 TCustomGrid.DefaultDrawing Property

Determines whether the Cells are painted when the grid is drawn.

Pascal

```
property DefaultDrawing: Boolean;
```

C++

```
__property Boolean DefaultDrawing;
```

Description

Set DefaultDrawing to true to have the grid automatically paint the cells in the grid, including adding the background color, the 3-D effects on the fixed cells, and the focus rectangle around the selected cell. Set DefaultDrawing to false to skip the painting of the cells when the grid is drawn.

When DefaultDrawing is false, the cells should be painted using the DrawCell (see page 14826) method. For TCustomGrid (see page 14800), DrawCell (see page 14826) is an abstract method. Descendants of TCustomGrid (see page 14800) use DrawCell (see page 14826) to provide their own customized drawing code, or to generate an OnDrawCell event.

See Also

DrawCell (see page 14826)

OnDrawCell (see page 14914)

75.1.3.6.8 TCustomGrid.DefaultRowHeight Property

Specifies the height (in pixels) of all rows that have not been explicitly resized.

Pascal

```
property DefaultRowHeight: Integer;
```

C++

```
__property int DefaultRowHeight;
```

Description

Set DefaultRowHeight to change the size of all rows in the grid. When DefaultRowHeight is set, rows that have been resized using the mouse or by setting the RowHeights (see page 14855) property are given the DefaultRowHeight as well.

When new rows are added to the grid using the RowCount (see page 14854) property, they are created with a height of DefaultRowHeight.

See Also

DefaultColWidth (see page 14846)

RowCount (see page 14854)

RowHeights (see page 14855)

75.1.3.6.9 TCustomGrid.EditorMode Property

Determines whether the current cell can be edited.

Pascal

```
property EditorMode: Boolean;
```

C++

```
__property Boolean EditorMode;
```

Description

Set EditorMode to true, at runtime, to put the grid in edit mode. When EditorMode is true, the user can edit cells in the grid. When the user presses F2, EditorMode is set to true. When the user presses Enter, the value of EditorMode is toggled.

Note: EditorMode has no effect if the Options (see page 14853) property does not include goEditing or if the Options (see page 14853) property includes goAlwaysShowEditor.

See Also

HideEditor (see page 14830)

Options (see page 14853)

ShowEditor (see page 14839)

75.1.3.6.10 TCustomGrid.FixedColor Property

Specifies the background color of the fixed rows and columns in the grid.

Pascal

```
property FixedColor: TColor;
```

C++

```
__property TColor FixedColor;
```

Description

Set FixedColor to specify the color for the nonscrolling cells in the grid. The scrolling region of the grid is drawn using the Color (see page 14844) property.

See Also

Color (see page 5812)

FixedCols (see page 14848)

FixedRows (see page 14849)

75.1.3.6.11 TCustomGrid.FixedCols Property

Specifies the number of columns on the left of the grid that cannot be scrolled.

Pascal

```
property FixedCols: Integer;
```

C++

```
__property int FixedCols;
```

Description

Set FixedCols to create or get rid of nonscrolling columns. Nonscrolling columns appear at the left of the grid, and are always visible, even when the user scrolls the other columns in the grid. Use nonscrolling columns for displaying row titles or row numbers, or to implement a scroll lock that the user can set.

Nonscrolling columns are visually distinct from the scrollable columns in the grid. They are painted with a separate background color, and their cells have a raised, 3-D look.

Note: Grids (see page 14765) must include at least one scrolling column. Do not set FixedCols to a value greater than ColCount (see page 14844) - 1.

See Also

ColCount (see page 14844)

DefaultDrawing (see page 14847)

FixedColor (see page 14848)

FixedRows (see page 14849)

75.1.3.6.12 TCustomGrid.FixedRows Property

Specifies the number of rows on the top of the grid that cannot be scrolled.

Pascal

```
property FixedRows: Integer;
```

C++

```
__property int FixedRows;
```

Description

Set FixedRows to create or get rid of nonscrolling rows. Nonscrolling rows appear at the top of the grid, and are always visible, even when the user scrolls the other rows in the grid. Use nonscrolling rows for displaying column titles or column numbers.

Nonscrolling rows are visually distinct from the scrollable rows in the grid. They are painted with a separate background color, and their cells have a raised, 3-D look.

Note: Grids (see page 14765) must include at least one scrolling row. Do not set FixedRows to a value greater than RowCount (see page 14854) - 1.

See Also

DefaultDrawing (see page 14847)

FixedColor (see page 14848)

FixedCols (see page 14848)

RowCount (see page 14854)

75.1.3.6.13 TCustomGrid.GridHeight Property

Specifies the height of the grid in pixels.

Pascal

```
property GridHeight: Integer;
```

C++

```
__property int GridHeight;
```

Description

Read GridHeight to determine the size of the grid. If GridHeight is less than the value of ClientHeight, all of the rows of the grid appear in the control with an empty region below the grid. If the underlying grid is too tall to appear in the control, GridHeight is the same as ClientHeight, and the user must scroll to see the entire contents of the grid.

See Also

ClientHeight (see page 5808)

GridWidth (see page 14851)

75.1.3.6.14 TCustomGrid.GridLineWidth Property

Specifies the width (in pixels) of the lines that separate the cells of the grid.

Pascal

```
property GridLineWidth: Integer;
```

C++

```
__property int GridLineWidth;
```

Description

Set GridLineWidth to make the lines that separate the cells in the grid heavier or lighter. When GridLineWidth is zero, no separators are drawn between the cells of the grid.

GridLineWidth will have no effect if the Options (see page 14853) property does not include goFixedHorzLine, goFixedVertLine, goHorzLine, or goVertLine.

Note: Values greater than 3 pixels are not recommended for applications that run on Windows 9x or Windows ME because of distortions that can appear.

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if DrawGrid1.DefaultColWidth > 90 then
```

```
    DrawGrid1.GridLineWidth := 2
  else
    DrawGrid1.GridLineWidth := 1;
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
  end;
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

FixedCols (see page 14848)

FixedRows (see page 14849)

Options (see page 14853)

75.1.3.6.15 TCustomGrid.GridWidth Property

Specifies the width of the grid in pixels.

Pascal

```
property GridWidth: Integer;
```

C++

```
__property int GridWidth;
```

Description

Read GridWidth to determine the size of the grid. If GridWidth is less than the value of ClientWidth, all of the columns of the grid appear in the control with an empty region to the right of the grid. If the underlying grid is too wide to appear in the control, GridWidth is the same as ClientWidth, and the user must scroll to see the entire contents of the grid.

See Also

ClientWidth (see page 5811)

GridHeight (see page 14850)

75.1.3.6.16 TCustomGrid.HitTest Property

Indicates the current position of the mouse cursor.

Pascal

```
property HitTest: TPoint;
```

C++

```
__property TPoint HitTest;
```

Description

Read HitTest to determine the position of the mouse when it is over the grid or when the grid has captured the mouse. HitTest may represent a position that is not within the client area of the grid control if the user is moving or resizing the rows or columns, or if the user is selecting a range of cells using the mouse. HitTest is the position of the mouse cursor in the coordinate system where (0,0) is the upper left corner of the grid control's client area.

See Also

ClientOrigin

MouseCoord (see page 14834)

75.1.3.6.17 TCustomGrid.InplaceEditor Property

Designates the edit control the grid uses to allow users to edit the contents of the selected cell.

Pascal

```
property InplaceEditor: TInplaceEdit;
```

C++

```
__property TInplaceEdit InplaceEditor;
```

Description

Read InplaceEditor to gain direct access to the edit control that sits on top of the selected cell when the grid is in edit mode. The inplace editor is created the first time the grid is put in edit mode. It is not destroyed until the grid is destroyed. The grid moves the editor when the selected cell changes. The grid hides the editor's window when the grid goes out of edit mode.

See Also

CreateEditor (see page 14823)

EditorMode (see page 14848)

HideEditor (see page 14830)

ShowEditor (see page 14839)

75.1.3.6.18 TCustomGrid.LeftCol Property

Specifies the index of the first visible scrollable column in the grid.

Pascal

```
property LeftCol: Longint;
```

C++

```
__property Longint LeftCol;
```

Description

Read LeftCol to determine the index of the first column in the scrollable region that is visible. Set LeftCol to scroll the columns in the grid so that the column with index LeftCol is the first column after the fixed columns.

See Also

FixedCols (see page 14848)

ScrollData (see page 14838)

VisibleColCount (see page 14859)

75.1.3.6.19 TCustomGrid.Options Property

Specifies various display and behavioral properties of the grid.

Pascal

```
property Options: TGridOptions;
```

C++

```
__property TGridOptions Options;
```

Description

Set Options to include the desired properties for the grid. See the type Grids.TGridOptions for definitions of these properties.

See Also

ColWidths (see page 14788)

EditorMode (see page 14848)

GridLineWidth (see page 14850)

RowHeights (see page 14855)

Selection (see page 14856)

TabStops (see page 14858)

75.1.3.6.20 TCustomGrid.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

75.1.3.6.21 TCustomGrid.Row Property

Specifies the index of the row that contains the selected cell.

Pascal

```
property Row: Longint;
```

C++

```
__property Longint Row;
```

Description

Use Row at runtime to determine the current row in the grid. Setting Row moves focus to the cell in the current column that is in the new row. The first row has an index of 0, the second row an index of 1, and so on.

The selected cell in the grid can be located by reading the Row property and the Col (see page 14843) property to obtain the indexes of its row and column. When focus moves to the grid, the selected cell gets input focus.

Delphi Examples:

```
{
This example displays a string grid, a list box, and a
button on a form. When the application runs, strings are put
in the cells of row 1 of the string grid. When the user
clicks the button, The strings in the current row of the
grid are displayed in the list box.
Note that when the grid contains an empty cell, the list box
has a corresponding empty string.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.Items := StringGrid1.Rows[StringGrid1.Row];
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
    Cells[1,1] := 'Object';
    Cells[2,1] := 'Pascal';
    Cells[3,1] := 'is';
    Cells[4,1] := 'excellent';
    Cells[0,2] := 'Row 2';
    Cells[1,2] := 'Delphi';
    Cells[2,2] := 'is';
    Cells[4,2] := 'RAD';
  end;
end;
```

See Also

Col (see page 14843)

75.1.3.6.22 TCustomGrid.RowCount Property

Specifies the number of rows in the grid.

Pascal

```
property RowCount: Longint;
```

C++

```
__property Longint RowCount;
```

Description

Read RowCount to determine the number entries in the RowHeights (see page 14855) array. Set RowCount to add or delete rows at the bottom of the grid. The value of RowCount includes any fixed rows at the top of the grid as well as the scrollable rows in the body of the grid.

Delphi Examples:

```
{
This code fills each cell with a number. The numbers are
arranged in consecutive order by rows.  Notice that even
cells in the fixed rows and columns are altered.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K : Integer;
begin
  K := 0;
  with StringGrid1 do
    for I := 0 to ColCount - 1 do
      for J:= 0 to RowCount - 1 do
        begin
          K := K + 1;
          Cells[I,J] := IntToStr(K);
        end;
end;
```

See Also

ColCount (see page 14844)

RowHeights (see page 14855)

VisibleRowCount (see page 14859)

75.1.3.6.23 TCustomGrid.RowHeights Property

Gives the height (in pixels) of all rows in the grid.

Pascal

```
property RowHeights [Index: Longint]: Integer;
```

C++

```
__property int RowHeights[Longint Index];
```

Description

Read RowHeights to determine the height of the cells in the row specified by the Index parameter. An index of 0 refers to the first row. Set RowWidths at runtime to change the height of an individual row.

If the height of a row has not been set explicitly by resizing with the mouse, or by using the RowHeights property, it's height is DefaultRowHeight (see page 14847).

Use the RowCount (see page 14854) property to determine the number of entries in the RowHeights array.

See Also

ColWidths (see page 14845)

DefaultRowHeight (see page 14847)

RowCount (see page 14854)

75.1.3.6.24 TCustomGrid.ScrollBars Property

Specifies whether the grid includes horizontal and vertical scroll bars.

Pascal

```
property ScrollBars: TScrollStyle;
```

C++

```
__property TScrollStyle ScrollBars;
```

Description

Use ScrollBars to give the grid horizontal or vertical scroll bars.

Note: If all the cells in the grid fit in the ClientWidth, no horizontal scroll bar appears, even if ScrollBars is ssHorizontal or ssBoth. If all the cells fit in the ClientHeight, no vertical scroll bar appears, even if ScrollBars is ssVertical or ssBoth.

See Also

ClientHeight (see page 5808)

ClientWidth (see page 5811)

75.1.3.6.25 TCustomGrid.Selection Property

Indicates the boundaries of the current selection.

Pascal

```
property Selection: TGridRect;
```

C++

```
__property TGridRect Selection;
```

Description

Set Selection to select a range of cells in the grid. Selection can either represent the first column, first row, last column and last row in the grid, or the row and column coordinates of the upper left and bottom right cells in the selected region.

Selection can only indicate more than one cell when Options (see page 14853) includes goRangeSelect.

Delphi Examples:

```
{
The following code selects the rectangle containing rows 1
through 4, and columns 2 and 3.  goRangeSelect in the
DrawGrid Options parameter must be True to select a range of
cells.  This example requires a button, a DrawGrid and a
populated ImageList.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  myRect: TGridRect;
begin
  myRect.Left := 2;
```

```
    myRect.Top := 1;
    myRect.Right := 3;
    myRect.Bottom := 4;
    DrawGrid1.Selection := myRect;
    DrawGrid1.Repaint;
  end;

  procedure TForm1.DrawGrid1Click(Sender: TObject);
  begin
    Label1.Caption := 'From OnClick: The cursor is in column ' +
                      IntToStr(DrawGrid1.Col + 1) +
                      ', row ' +
                      IntToStr(DrawGrid1.Row + 1);
  end;

  procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
    Rect: TRect; State: TGridDrawState);
  var
    index: Integer;
  begin
    index := (ARow * DrawGrid1.ColCount + ACol) mod (DrawGrid1.ColCount + 1);
    if (gdSelected in State) then
      DrawGrid1.Canvas.Brush.Color := clYellow
    else
      DrawGrid1.Canvas.Brush.Color := clBackground;
    DrawGrid1.Canvas.FillRect(Rect);
    if (gdFocused in State) then
    begin
      DrawGrid1.Canvas.DrawFocusRect(Rect);
      Label2.Caption:= 'From OnDrawCell: Cell ' + InttoStr(index) + ' has the focus.';
    end;
    ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  end;

  procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  begin
    CanSelect:= True;
  end;
```

See Also

Col (see page 14843)

Options (see page 14853)

Row (see page 14854)

75.1.3.6.26 TCustomGrid.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

75.1.3.6.27 TCustomGrid.TabStops Property

Indicates whether the user can tab to specified columns in the grid.

Pascal

```
property TabStops [Index: Longint]: Boolean;
```

C++

```
__property Boolean TabStops[Longint Index];
```

Description

Set TabStops to false to remove the column identified by Index from the tab order. The first column in the grid is identified by an Index of 0.

Note: When the Options (see page 14853) property does not include goTabs, the user can't tab to any column, even if TabStops is true.

See Also

Options (see page 14853)

75.1.3.6.28 TCustomGrid.TopRow Property

Specifies the index of the first visible scrollable row in the grid.

Pascal

```
property TopRow: Longint;
```

C++

```
__property Longint TopRow;
```

Description

Read TopRow to determine the index of the first row in the scrollable region that is visible. Set TopRow to scroll the rows in the grid so that the row with index TopRow is the first row after the fixed rows.

Delphi Examples:

```
{
This code uses a draw grid and a button on a form. When
the user clicks the button, the last row of the draw
grid becomes the top row.  This example requires a
button, a DrawGrid and a populated ImageList.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  DrawGrid1.TopRow := DrawGrid1.RowCount - 1;
end;

procedure TForm1.DrawGrid1Click(Sender: TObject);
begin
  Label1.Caption := 'From OnClick: The cursor is in column ' +
                    IntToStr(DrawGrid1.Col + 1) +
                    ', row ' +
                    IntToStr(DrawGrid1.Row + 1);
```

```
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := (ARow * DrawGrid1.ColCount + ACol) mod (DrawGrid1.ColCount + 1);
  if (gdSelected in State) then
    DrawGrid1.Canvas.Brush.Color := clYellow
  else
    DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label2.Caption:= 'From OnDrawCell: Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

FixedRows (see page 14849)

ScrollData (see page 14838)

VisibleRowCount (see page 14859)

75.1.3.6.29 TCustomGrid.VisibleColCount Property

Indicates the number of scrollable columns visible in the grid.

Pascal

```
property VisibleColCount: Integer;
```

C++

```
__property int VisibleColCount;
```

Description

Use VisibleColCount to determine the number of scrollable columns fully visible in the grid. VisibleColCount does not include the fixed columns counted by the FixedCols (see page 14848) property. It does not include any partially visible columns on the right edge of the grid.

See Also

ColCount (see page 14844)

FixedCols (see page 14848)

VisibleRowCount (see page 14859)

75.1.3.6.30 TCustomGrid.VisibleRowCount Property

Indicates the number of scrollable rows visible in the grid.

Pascal

```
property VisibleRowCount: Integer;
```

C++

```
__property int VisibleRowCount;
```

Description

Use VisibleRowCount to determine the number of scrollable rows fully visible in the grid. VisibleRowCount does not include the fixed rows counted by the FixedRows (see page 14849) property. It does not include any partially visible rows on the bottom of the grid.

See Also

FixedRows (see page 14849)

RowCount (see page 14854)

VisibleColCount (see page 14859)

75.1.4 TDrawGrid Class

TDrawGrid represents a grid control that displays information in column and row format.

Class Hierarchy

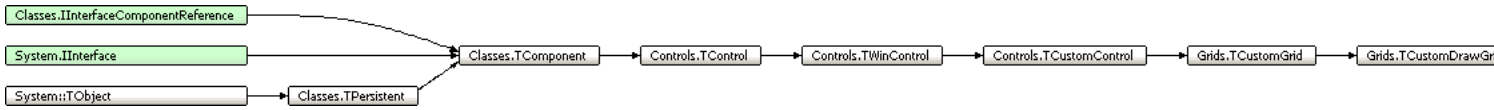

Pascal

```
TDrawGrid = class(TCustomDrawGrid);
```

C++

```
class TDrawGrid : public TCustomDrawGrid;
```

File

Grids

Description

Add a TDrawGrid object to a form to present arbitrary information in a tabular format. TDrawGrid provides many properties to control the appearance of the grid, as well as events and methods that take advantage of the tabular organization of the grid in responding to user actions.

TDrawGrid introduces several events and methods to allow applications to customize the appearance and behavior of the grid.

If the cells of the grid contain text strings that represent objects, consider using a TStringGrid (see page 14968) object instead. If the grid displays field values from the records in a dataset, use TDBGrid instead.

See Also

TDBGrid (see page 9668)

TStringGrid (see page 14968)

Draw Grids

Grids

TGridOption (see page 14999)

75.1.4.1 TDrawGrid Members

The following tables list the members exposed by TDrawGrid.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 14814)	This is DesignOptionsBoost, a member of class TCustomGrid.
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FGridState (see page 14814)	This is FGridState, a member of class TCustomGrid.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveCellExtents (see page 14814)	This is FSaveCellExtents, a member of class TCustomGrid.
	VirtualView (see page 14814)	This is VirtualView, a member of class TCustomGrid.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AdjustSize (see page 14816)	Adds or deletes rows or columns from the grid.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.

	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	BeginColumnDrag (see page 14817)	Starts the dragging of a column in the grid.
	BeginRowDrag (see page 14817)	Starts the dragging of a row in the grid.
	BoxRect (see page 14818)	Returns the screen coordinates of a specified set of cells.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcDrawInfo (see page 14818)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 14818)	Calculates the pixel locations of the fixed cells.
	CalcSizingState (see page 14819)	Determines the appearance of the mouse cursor during resizing.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanEditAcceptKey (see page 14819)	Indicates whether the Key parameter can be entered into the current cell.
	CanEditModify (see page 14819)	Indicates whether the current cell in the grid can be edited.
	CanEditShow (see page 14820)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 14820)	Indicates whether the grid should respond to a given key combination.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	CellRect (see page 14821)	Returns the screen coordinates of a cell in the grid.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeGridOrientation (see page 14821)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CheckColumnDrag (see page 14821)	CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 14822)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
	Click (see page 5743)	Respond to user click.
	ColumnMoved (see page 14781)	Generates an OnColumnMoved (see page 14794) event.
	ColumnMoved (see page 14822)	Provides the interface for a method that responds when the position of a column changes.
	ColWidthsChanged (see page 14822)	Responds when the column widths change.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateEditor (see page 14823)	Creates the inplace editor control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 14823)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
V	DefineProperties (see page 14824)	Adds the ColWidths (see page 14845) and RowHeights (see page 14855) properties to the values saved with the grid.
V	DeleteColumn (see page 14824)	Deletes a specified column from the grid.
V	DeleteRow (see page 14825)	Deletes a specified from the grid.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
V	DoExit (see page 14825)	Hides the inplace editor when selection moves to another control in a form.
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
V	DoMouseWheelDown (see page 14825)	Performs tasks when the mouse wheel is rotated downward.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
V	DoMouseWheelUp (see page 14826)	Performs tasks when the mouse wheel is rotated upward.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawCell (see page 14782)	Generates an OnDrawCell (see page 14795) event.
A	DrawCell (see page 14826)	Provides the prototype for a method that draws the contents of a cell in the grid.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EndColumnDrag (see page 14827)	Ends the dragging of a column in the grid.
	EndRowDrag (see page 14827)	Ends the dragging of a row in the grid.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	FocusCell (see page 14827)	Changes the current cell to a specified cell.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.

	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEditLimit (see page 14828)	Provides the interface for a method that limits the number of characters the user can type into cells.
V	GetEditMask (see page 14782)	Generates an OnGetEditMask (see page 14796) event.
	GetEditMask (see page 14828)	Provides the interface for a method that returns an input mask for the specified cell.
	GetEditStyle (see page 14828)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
V	GetEditText (see page 14783)	Generates an OnGetEditText (see page 14797) event.
	GetEditText (see page 14829)	Returns the value of the indicated cell formatted for editing.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetGridHeight (see page 14829)	Returns the value of the GridHeight (see page 14850) property.
	GetGridWidth (see page 14830)	Returns the value of the GridWidth (see page 14851) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HideEdit (see page 14830)	Hides the inplace editor.
	HideEditor (see page 14830)	Hides the inplace editor when it is not in use.
	InvalidateCell (see page 14831)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCol (see page 14831)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	InvalidateEditor (see page 14832)	Causes the inplace editor to repaint itself, moving to a new location if necessary.
	InvalidateGrid (see page 14832)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 14832)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.

	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
V	KeyDown (see page 14833)	Provides special processing when the user presses a key.
	KeyPress (see page 6163)	Respond to keyboard input.
V	KeyPress (see page 14833)	Provides special processing when the user presses a key.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
V	MouseDown (see page 14834)	Provides special processing when the user presses a mouse button.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
V	MouseMove (see page 14834)	Provides special processing when the user moves the mouse.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	MouseUp (see page 14835)	Provides special processing when the user releases the mouse button.
	MoveColRow (see page 14835)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 14836)	Moves a column to a new position.
	MoveRow (see page 14836)	Moves a row to a new position.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 5920)	Renders the image of a custom control.
V	Paint (see page 14836)	Draws the image of the grid control on the screen.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	RowHeightsChanged (see page 14837)	Responds when the row heights change.
V	RowMoved (see page 14783)	Generates an OnRowMoved (see page 14798) event.
	RowMoved (see page 14837)	Provides the interface for a method that responds when the position of a row changes.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	ScrollData (see page 14838)	Scrolls the visual image of the cells in the grid.

	SelectCell (see page 14784)	Determines whether a particular cell in the grid can be selected.
	SelectCell (see page 14838)	Determines whether a particular cell in the grid can be selected.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEditText (see page 14784)	Generates an OnSetEditText (see page 14800) event.
	SetEditText (see page 14839)	Provides the interface for a method that updates the text associated with a cell.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	ShowEditor (see page 14839)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 14839)	Displays the inplace editor, with its text set to the specified character.
	SizeChanged (see page 14840)	Responds to changes in the size of the grid.
	Sizing (see page 14840)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	TimedScroll (see page 14840)	Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.
	TopLeftChanged (see page 14784)	Generates an OnTopLeftChanged (see page 14800) event.
	TopLeftChanged (see page 14841)	Updates the position of the inplace editor when the grid scrolls.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateDesigner (see page 14841)	Notifies the designer when rows and columns of the grid are moved or resized.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.

V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Col (see page 14843)	Specifies the index of the column that contains the selected cell.
	ColWidths (see page 14845)	Indicates the width (in pixels) of all the columns in the grid.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	EditorMode (see page 14848)	Determines whether the current cell can be edited.
R	GridHeight (see page 14850)	Specifies the height of the grid in pixels.
R	GridWidth (see page 14851)	Specifies the width of the grid in pixels.
R	HitTest (see page 14851)	Indicates the current position of the mouse cursor.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
R	InplaceEditor (see page 14852)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	LeftCol (see page 14852)	Specifies the index of the first visible scrollable column in the grid.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	Row (see page 14854)	Specifies the index of the row that contains the selected cell.
	RowHeights (see page 14855)	Gives the height (in pixels) of all rows in the grid.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Selection (see page 14856)	Indicates the boundaries of the current selection.
	TabStops (see page 14858)	Indicates whether the user can tab to specified columns in the grid.
	Text (see page 5842)	Contains a text string associated with the control.
	TopRow (see page 14858)	Specifies the index of the first visible scrollable row in the grid.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	CellRect (see page 14781)	Returns the screen coordinates of a cell in the grid.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 5918)	Creates an instance of TCustomControl (see page 5908).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 14813)	Creates an instance of TCustomGrid (see page 14800).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 14813)	Destroys an instance of TCustomGrid (see page 14800).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.

	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseCoord (see page 14834)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.

	MouseToCell (see page 14783)	Returns the column and row of the cell at the position with screen coordinates (X,Y).
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	Canvas (see page 14785)	Specifies the TCanvas object that presents a drawing surface for the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.

R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Col (see page 14787)	Specifies the index of the column that contains the selected cell.
	ColWidths (see page 14788)	Indicates the width (in pixels) of all the columns in the grid.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EditorMode (see page 14789)	Determines whether the current cell can be edited.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GridHeight (see page 14789)	Specifies the height of the grid in pixels.
	GridWidth (see page 14789)	Specifies the width of the grid in pixels.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LeftCol (see page 14790)	Specifies the index of the first visible scrollable column in the grid.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	Row (see page 14790)	Specifies the index of the row that contains the selected cell.
	RowHeights (see page 14791)	Gives the height (in pixels) of all rows in the grid.
	Selection (see page 14791)	Indicates the boundaries of the current selection.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TabStops (see page 14793)	Indicates whether the user can tab to specified columns in the grid.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TopRow (see page 14793)	Specifies the index of the first visible scrollable row in the grid.

	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 14906)	Occurs when the user clicks the control.
	OnColumnMoved (see page 14908)	Occurs immediately after the position of a column changes.
	OnContextPopup (see page 14908)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 14909)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 14909)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 14911)	Occurs when the user drags an object over a control.
	OnDrawCell (see page 14914)	Occurs when a cell in the grid needs to be drawn.
	OnEndDock (see page 14914)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 14915)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 14916)	Occurs when a control receives the input focus.
	OnExit (see page 14918)	Occurs when the input focus shifts away from one control to another.
	OnGetEditMask (see page 14920)	Occurs when the in-place editor requests an edit mask.
	OnGetEditText (see page 14920)	Occurs when the in-place editor requests the value of a cell.
	OnKeyDown (see page 14920)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 14923)	Occurs when a key is pressed.
	OnKeyUp (see page 14924)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 14926)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 14927)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 14929)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 14929)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 14930)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 14932)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheelDown (see page 14934)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 14935)	Occurs when the mouse wheel is rotated upward.
	OnRowMoved (see page 14935)	Occurs immediately after the position of a row changes.
	OnSelectCell (see page 14935)	Occurs before a cell in the grid is selected.
	OnSetEditText (see page 14936)	Occurs when the user edits the value of a cell.
	OnStartDock (see page 14936)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 14937)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnTopLeftChanged (see page 14937)	Occurs immediately after the TopRow (see page 14793) property or the LeftCol (see page 14790) property changes.

Published Properties

	Name	Description
	Align (see page 14874)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 14878)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 14879)	Specifies which edges of the control are beveled.
	BevelInner (see page 14879)	Specifies the cut of the inner bevel.
	BevelKind (see page 14879)	Specifies the control's bevel style.

	BevelOuter (see page 14880)	Specifies the cut of the outer bevel.
	BevelWidth (see page 14880)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 14881)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 14881)	Determines whether a single line border is drawn around the grid.
	ColCount (see page 14881)	Specifies the number of columns in the grid.
	Color (see page 14882)	Specifies the background color of the control.
	Constraints (see page 14883)	Specifies the size constraints for the control.
	Ctl3D (see page 14884)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DefaultColWidth (see page 14884)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 14885)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 14885)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DragCursor (see page 14886)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 14886)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 14887)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 14888)	Controls whether the control responds to mouse, keyboard, and timer events.
	FixedColor (see page 14889)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 14889)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 14889)	Specifies the number of rows on the top of the grid that cannot be scrolled.
	Font (see page 14890)	Controls the attributes of text written on or in the control.
	GridLineWidth (see page 14891)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 14892)	Specifies various display and behavioral properties of the grid.
	ParentBiDiMode (see page 14893)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 14893)	Determines where a control looks for its color information.
	ParentCtl3D (see page 14893)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 14894)	Determines where a control looks for its font information.
	ParentShowHint (see page 14894)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 14895)	Identifies the pop-up menu associated with the control.
	RowCount (see page 14899)	Specifies the number of rows in the grid.
	ScrollBars (see page 14900)	Specifies whether the grid includes horizontal and vertical scroll bars.
	ShowHint (see page 14901)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 14902)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 14857)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 14903)	Determines whether the component appears onscreen.
	VisibleColCount (see page 14904)	Indicates the number of scrollable columns visible in the grid.
	VisibleRowCount (see page 14905)	Indicates the number of scrollable rows visible in the grid.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

75.1.4.2 TDrawGrid Properties

The properties of the TDrawGrid class are listed here.

Published Properties

	Name	Description
	Align (see page 14874)	Determines how the control aligns within its container (parent control).
	Anchors (see page 14878)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 14879)	Specifies which edges of the control are beveled.
	BevelInner (see page 14879)	Specifies the cut of the inner bevel.
	BevelKind (see page 14879)	Specifies the control's bevel style.
	BevelOuter (see page 14880)	Specifies the cut of the outer bevel.
	BevelWidth (see page 14880)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 14881)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 14881)	Determines whether a single line border is drawn around the grid.
	ColCount (see page 14881)	Specifies the number of columns in the grid.
	Color (see page 14882)	Specifies the background color of the control.
	Constraints (see page 14883)	Specifies the size constraints for the control.
	Ctl3D (see page 14884)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColWidth (see page 14884)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 14885)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 14885)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DragCursor (see page 14886)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 14886)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 14887)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 14888)	Controls whether the control responds to mouse, keyboard, and timer events.
	FixedColor (see page 14889)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 14889)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 14889)	Specifies the number of rows on the top of the grid that cannot be scrolled.
	Font (see page 14890)	Controls the attributes of text written on or in the control.
	GridLineWidth (see page 14891)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
	Options (see page 14892)	Specifies various display and behavioral properties of the grid.
	ParentBiDiMode (see page 14893)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 14893)	Determines where a control looks for its color information.
	ParentCtl3D (see page 14893)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 14894)	Determines where a control looks for its font information.
	ParentShowHint (see page 14894)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 14895)	Identifies the pop-up menu associated with the control.
	RowCount (see page 14899)	Specifies the number of rows in the grid.
	ScrollBars (see page 14900)	Specifies whether the grid includes horizontal and vertical scroll bars.
	ShowHint (see page 14901)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 14902)	Indicates the position of the control in its parent's tab order.
	Visible (see page 14903)	Determines whether the component appears onscreen.
	VisibleColCount (see page 14904)	Indicates the number of scrollable columns visible in the grid.
	VisibleRowCount (see page 14905)	Indicates the number of scrollable rows visible in the grid.

75.1.4.2.1 TDrawGrid.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;
```

```
  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
```

```
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
```

```
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

75.1.4.2.2 TDrawGrid.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

75.1.4.2.3 TDrawGrid.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

75.1.4.2.4 TDrawGrid.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

75.1.4.2.5 TDrawGrid.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

75.1.4.2.6 TDrawGrid.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

75.1.4.2.7 TDrawGrid.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

75.1.4.2.8 TDrawGrid.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

75.1.4.2.9 TDrawGrid.BorderStyle Property

Determines whether a single line border is drawn around the grid.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Set BorderStyle to bsSingle to add a single line border around the grid's image. Set BorderStyle to bsNone to omit the border.

75.1.4.2.10 TDrawGrid.ColCount Property

Specifies the number of columns in the grid.

Pascal

```
property ColCount: Longint;
```

C++

```
__property Longint ColCount;
```

Description

Read ColCount to determine the number entries in the ColWidths array. Set ColCount to add or delete columns at the righthand side of the grid. The value of ColCount includes any fixed columns at the left of the grid as well as the scrollable columns in the body of the grid.

Delphi Examples:

```
{
This code fills each cell with a number. The numbers are
arranged in consecutive order by rows.  Notice that even
cells in the fixed rows and columns are altered.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K : Integer;
begin
  K := 0;
  with StringGrid1 do
    for I := 0 to ColCount - 1 do
      for J:= 0 to RowCount - 1 do
        begin
          K := K + 1;
          Cells[I,J] := IntToStr(K);
        end;
end;
```

See Also

ColWidths (see page 14845)

RowCount (see page 14854)

VisibleColCount (see page 14859)

75.1.4.2.11 TDrawGrid.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

75.1.4.2.12 TDrawGrid.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

75.1.4.2.13 TDrawGrid.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

75.1.4.2.14 TDrawGrid.DefaultColWidth Property

Determines the width (in pixels) of all columns that have not been explicitly resized.

Pascal

```
property DefaultColWidth: Integer;
```

C++

```
__property int DefaultColWidth;
```

Description

Set DefaultColWidth to change the size of all columns in the grid. When DefaultColWidth is set, columns that have been resized using the mouse or by setting the ColWidths property are given the DefaultColWidth as well.

When new columns are added to the grid using the ColCount property, they are created with a width of DefaultColWidth.

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if DrawGrid1.DefaultColWidth > 90 then
    DrawGrid1.GridLineWidth := 2
```

```
    else
      DrawGrid1.GridLineWidth := 1;
  end;

  procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
    Rect: TRect; State: TGridDrawState);
  var
    index: Integer;
  begin
    index := ARow * DrawGrid1.ColCount + ACol;
    DrawGrid1.Canvas.Brush.Color := clBackground;
    DrawGrid1.Canvas.FillRect(Rect);
    ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
    if (gdFocused in State) then
    begin
      DrawGrid1.Canvas.DrawFocusRect(Rect);
    end;
  end;

  procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  begin
    CanSelect:= True;
  end;
```

See Also

ColCount (see page 14844)

ColWidths (see page 14845)

DefaultRowHeight (see page 14847)

75.1.4.2.15 TDrawGrid.DefaultDrawing Property

Determines whether the Cells are painted when the grid is drawn.

Pascal

```
property DefaultDrawing: Boolean;
```

C++

```
__property Boolean DefaultDrawing;
```

Description

Set DefaultDrawing to true to have the grid automatically paint the cells in the grid, including adding the background color, the 3-D effects on the fixed cells, and the focus rectangle around the selected cell. Set DefaultDrawing to false to skip the painting of the cells when the grid is drawn.

When DefaultDrawing is false, the cells should be painted using the DrawCell method. For TCustomGrid, DrawCell is an abstract method. Descendants of TCustomGrid use DrawCell to provide their own customized drawing code, or to generate an OnDrawCell event.

See Also

DrawCell (see page 14826)

OnDrawCell (see page 14914)

75.1.4.2.16 TDrawGrid.DefaultRowHeight Property

Specifies the height (in pixels) of all rows that have not been explicitly resized.

Pascal

```
property DefaultRowHeight: Integer;
```

C++

```
__property int DefaultRowHeight;
```

Description

Set DefaultRowHeight to change the size of all rows in the grid. When DefaultRowHeight is set, rows that have been resized using the mouse or by setting the RowHeights property are given the DefaultRowHeight as well.

When new rows are added to the grid using the RowCount property, they are created with a height of DefaultRowHeight.

See Also

DefaultColWidth (see page 14846)

RowCount (see page 14854)

RowHeights (see page 14855)

75.1.4.2.17 TDrawGrid.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

75.1.4.2.18 TDrawGrid.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

75.1.4.2.19 TDrawGrid.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
```

```
    AttachMode: TNodeAttachMode;
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

75.1.4.2.20 TDrawGrid.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

75.1.4.2.21 TDrawGrid.FixedColor Property

Specifies the background color of the fixed rows and columns in the grid.

Pascal

```
property FixedColor: TColor;
```

C++

```
__property TColor FixedColor;
```

Description

Set FixedColor to specify the color for the nonscrolling cells in the grid. The scrolling region of the grid is drawn using the Color property.

See Also

Color (see page 5812)

FixedCols (see page 14848)

FixedRows (see page 14849)

75.1.4.2.22 TDrawGrid.FixedCols Property

Specifies the number of columns on the left of the grid that cannot be scrolled.

Pascal

```
property FixedCols: Integer;
```

C++

```
__property int FixedCols;
```

Description

Set FixedCols to create or get rid of nonscrolling columns. Nonscrolling columns appear at the left of the grid, and are always visible, even when the user scrolls the other columns in the grid. Use nonscrolling columns for displaying row titles or row numbers, or to implement a scroll lock that the user can set.

Nonscrolling columns are visually distinct from the scrollable columns in the grid. They are painted with a separate background color, and their cells have a raised, 3-D look.

Note: Grids must include at least one scrolling column. Do not set FixedCols to a value greater than ColCount - 1.

See Also

ColCount (see page 14844)

DefaultDrawing (see page 14847)

FixedColor (see page 14848)

FixedRows (see page 14849)

75.1.4.2.23 TDrawGrid.FixedRows Property

Specifies the number of rows on the top of the grid that cannot be scrolled.

Pascal

```
property FixedRows: Integer;
```

C++

```
__property int FixedRows;
```

Description

Set FixedRows to create or get rid of nonscrolling rows. Nonscrolling rows appear at the top of the grid, and are always visible, even when the user scrolls the other rows in the grid. Use nonscrolling rows for displaying column titles or column numbers.

Nonscrolling rows are visually distinct from the scrollable rows in the grid. They are painted with a separate background color, and their cells have a raised, 3-D look.

Note: Grids must include at least one scrolling row. Do not set FixedRows to a value greater than RowCount - 1.

See Also

DefaultDrawing (see page 14847)

FixedColor (see page 14848)

FixedCols (see page 14848)

RowCount (see page 14854)

75.1.4.2.24 TDrawGrid.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

75.1.4.2.25 TDrawGrid.GridLineWidth Property

Specifies the width (in pixels) of the lines that separate the cells of the grid.

Pascal

```
property GridLineWidth: Integer;
```

C++

```
__property int GridLineWidth;
```

Description

Set GridLineWidth to make the lines that separate the cells in the grid heavier or lighter. When GridLineWidth is zero, no separators are drawn between the cells of the grid.

GridLineWidth will have no effect if the Options property does not include goFixedHorzLine, goFixedVertLine, goHorzLine, or goVertLine.

Note: Values greater than 3 pixels are not recommended for applications that run on Windows 9x or Windows ME because of distortions that can appear.

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if DrawGrid1.DefaultColWidth > 90 then
    DrawGrid1.GridLineWidth := 2
  else
    DrawGrid1.GridLineWidth := 1;
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
  end;
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

FixedCols (see page 14848)

FixedRows (see page 14849)

Options (see page 14853)

75.1.4.2.26 TDrawGrid.Options Property

Specifies various display and behavioral properties of the grid.

Pascal

```
property Options: TGridOptions;
```

C++

```
__property TGridOptions Options;
```

Description

Set Options to include the desired properties for the grid. See the type Grids.TGridOptions for definitions of these properties.

See Also

ColWidths (see page 14788)

EditorMode (see page 14848)

GridLineWidth (see page 14850)

RowHeights (see page 14855)

Selection (see page 14856)

TabStops (see page 14858)

75.1.4.2.27 TDrawGrid.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

75.1.4.2.28 TDrawGrid.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

75.1.4.2.29 TDrawGrid.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

75.1.4.2.30 TDrawGrid.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

75.1.4.2.31 TDrawGrid.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

75.1.4.2.32 TDrawGrid.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
```

```
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Cut1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      const int num_items = 4;
      for (int index = 0; index < num_items; ++index)
      {
          TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
          PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
          NewItem->Caption = "Menu Item " + IntToStr(index);
          NewItem->Tag = index;
          NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
          TNotifyEvent();
      }
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
          {
```

```
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
```

```
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
```

```
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

75.1.4.2.33 TDrawGrid.RowCount Property

Specifies the number of rows in the grid.

Pascal

```
property RowCount: Longint;
```

C++

```
__property Longint RowCount;
```

Description

Read RowCount to determine the number entries in the RowHeights array. Set RowCount to add or delete rows at the bottom of the grid. The value of RowCount includes any fixed rows at the top of the grid as well as the scrollable rows in the body of the grid.

Delphi Examples:

```
{
This code fills each cell with a number. The numbers are
arranged in consecutive order by rows.  Notice that even
cells in the fixed rows and columns are altered.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K : Integer;
begin
  K := 0;
  with StringGrid1 do
    for I := 0 to ColCount - 1 do
      for J:= 0 to RowCount - 1 do
        begin
          K := K + 1;
          Cells[I,J] := IntToStr(K);
        end;
end;
```

See Also

ColCount (see page 14844)

RowHeights (see page 14855)

VisibleRowCount (see page 14859)

75.1.4.2.34 TDrawGrid.ScrollBars Property

Specifies whether the grid includes horizontal and vertical scroll bars.

Pascal

```
property ScrollBars: TScrollStyle;
```

C++

```
__property TScrollStyle ScrollBars;
```

Description

Use ScrollBars to give the grid horizontal or vertical scroll bars.

Note: If all the cells in the grid fit in the ClientWidth, no horizontal scroll bar appears, even if ScrollBars is ssHorizontal or ssBoth. If all the cells fit in the ClientHeight, no vertical scroll bar appears, even if ScrollBars is ssVertical or ssBoth.

See Also

ClientHeight (see page 5808)

ClientWidth (see page 5811)

75.1.4.2.35 TDrawGrid.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

75.1.4.2.36 TDrawGrid.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

75.1.4.2.37 TDrawGrid.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
```

```
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

75.1.4.2.38 TDrawGrid.VisibleColCount Property

Indicates the number of scrollable columns visible in the grid.

Pascal

```
property VisibleColCount: Integer;
```

C++

```
__property int VisibleColCount;
```

Description

Use VisibleColCount to determine the number of scrollable columns fully visible in the grid. VisibleColCount does not include the fixed columns counted by the FixedCols property. It does not include any partially visible columns on the right edge of the grid.

See Also

ColCount (see page 14844)

FixedCols (see page 14848)

VisibleRowCount (see page 14859)

75.1.4.2.39 TDrawGrid.VisibleRowCount Property

Indicates the number of scrollable rows visible in the grid.

Pascal

```
property VisibleRowCount: Integer;
```

C++

```
__property int VisibleRowCount;
```

Description

Use VisibleRowCount to determine the number of scrollable rows fully visible in the grid. VisibleRowCount does not include the fixed rows counted by the FixedRows property. It does not include any partially visible rows on the bottom of the grid.

See Also

FixedRows (see page 14849)

RowCount (see page 14854)

VisibleColCount (see page 14859)

75.1.4.3 TDrawGrid Events

The events of the TDrawGrid class are listed here.

Published Events

	Name	Description
	OnClick (see page 14906)	Occurs when the user clicks the control.
	OnColumnMoved (see page 14908)	Occurs immediately after the position of a column changes.
	OnContextPopup (see page 14908)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 14909)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 14909)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 14911)	Occurs when the user drags an object over a control.
	OnDrawCell (see page 14914)	Occurs when a cell in the grid needs to be drawn.
	OnEndDock (see page 14914)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 14915)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 14916)	Occurs when a control receives the input focus.
	OnExit (see page 14918)	Occurs when the input focus shifts away from one control to another.
	OnGetEditMask (see page 14920)	Occurs when the in-place editor requests an edit mask.
	OnGetEditText (see page 14920)	Occurs when the in-place editor requests the value of a cell.
	OnKeyDown (see page 14920)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 14923)	Occurs when a key is pressed.
	OnKeyUp (see page 14924)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 14926)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 14927)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 14929)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 14929)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 14930)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

	OnMouseUp (see page 14932)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheelDown (see page 14934)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 14935)	Occurs when the mouse wheel is rotated upward.
	OnRowMoved (see page 14935)	Occurs immediately after the position of a row changes.
	OnSelectCell (see page 14935)	Occurs before a cell in the grid is selected.
	OnSetEditText (see page 14936)	Occurs when the user edits the value of a cell.
	OnStartDock (see page 14936)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 14937)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnTopLeftChanged (see page 14937)	Occurs immediately after the TopRow (see page 14793) property or the LeftCol (see page 14790) property changes.

75.1.4.3.1 TDrawGrid.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}
```

```
void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

75.1.4.3.2 TDrawGrid.OnColumnMoved Event

Occurs immediately after the position of a column changes.

Pascal

```
property OnColumnMoved: TMovedEvent;
```

C++

```
__property TMovedEvent OnColumnMoved;
```

Description

Use OnColumnMoved to perform special processing when the position of a grid column changes. Columns can be moved programmatically or by user manipulation. The FromIndex parameter is the old index of the column, and the ToIndex parameter is the new index. OnColumnMoved does not occur unless the Options (see page 14892) property includes goColMoving.

See Also

OnRowMoved (see page 14935)

Options (see page 14853)

75.1.4.3.3 TDrawGrid.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

75.1.4.3.4 TDrawGrid.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

75.1.4.3.5 TDrawGrid.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}
```

```
/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
```

```
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

75.1.4.3.6 TDrawGrid.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;
```

```
procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

75.1.4.3.7 TDrawGrid.OnDrawCell Event

Occurs when a cell in the grid needs to be drawn.

Pascal

```
property OnDrawCell: TDrawCellEvent;
```

C++

```
__property TDrawCellEvent OnDrawCell;
```

Description

Write an OnDrawCell event handler to draw the contents of all the cells in the grid. Draw on the cell using the methods of the Canvas (see page 14785) property.

If the OnDrawCell event handler is not assigned, all cells in the draw grid will appear empty. If the DefaultDrawing (see page 14885) property is true, the draw grid paints the background color of the cell before the OnDrawCell event, and draws a focus rectangle around the selected cell after the OnDrawCell event handler finishes drawing the contents of the cell. If the DefaultDrawing (see page 14885) property is false, the OnDrawCell event handler should paint the background of the cell and provide all visual indication of selection and focus.

See Also

Canvas (see page 5921)

DefaultDrawing (see page 14847)

75.1.4.3.8 TDrawGrid.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

75.1.4.3.9 TDrawGrid.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

75.1.4.3.10 TDrawGrid.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
```

```
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

75.1.4.3.11 TDrawGrid.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

75.1.4.3.12 TDrawGrid.OnGetEditMask Event

Occurs when the in-place editor requests an edit mask.

Pascal

```
property OnGetEditMask: TGetEditEvent;
```

C++

```
__property TGetEditEvent OnGetEditMask;
```

Description

Write an OnGetEditMask event handler to provide the in-place editor with an edit mask that describes the valid input strings a user can type into a cell. Set the Value parameter to the string that describes the edit mask for the specified cell. If the cell does not have an edit mask, set the Value parameter to an empty string.

OnGetEditMask does not occur unless the Options (see page 14892) property includes goEditing.

See Also

Options (see page 14853)

EditMask (see page 15821)

75.1.4.3.13 TDrawGrid.OnGetEditText Event

Occurs when the in-place editor requests the value of a cell.

Pascal

```
property OnGetEditText: TGetEditEvent;
```

C++

```
__property TGetEditEvent OnGetEditText;
```

Description

Write an OnGetEditText event handler to provide the in-place editor with a string representing the contents of the cell. Set the Value parameter to the string for the specified cell. When the cell enters edit mode, the contents of the cell are drawn as the Value string returned by the OnGetEditText event handler. This image need not match the appearance of the cell when it is not in edit mode, which is drawn using the OnDrawCell (see page 14914) event handler.

OnGetEditText does not occur unless the Options (see page 14892) property includes goEditing.

See Also

OnDrawCell (see page 14914)

OnSetEditText (see page 14936)

75.1.4.3.14 TDrawGrid.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
```

```
      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

75.1.4.3.15 TDrawGrid.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys

that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

75.1.4.3.16 TDrawGrid.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
```

```
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

75.1.4.3.17 TDrawGrid.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the

OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

75.1.4.3.18 TDrawGrid.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
```

```
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
```

```
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

75.1.4.3.19 TDrawGrid.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

75.1.4.3.20 TDrawGrid.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

75.1.4.3.21 TDrawGrid.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
```

```
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
```

```
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

75.1.4.3.22 TDrawGrid.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
```

```
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
```

```
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

75.1.4.3.23 TDrawGrid.OnMouseWheelDown Event

Occurs when the mouse wheel is rotated downward.

Pascal

```
property OnMouseWheelDown: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelDown;
```

Description

Write code in the OnMouseWheelDown event handler to perform actions when the mouse wheel is rotated downward. OnMouseWheelDown occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheel (see page 5869)

OnMouseWheelUp (see page 5870)

75.1.4.3.24 TDrawGrid.OnMouseWheelUp Event

Occurs when the mouse wheel is rotated upward.

Pascal

```
property OnMouseWheelUp: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelUp;
```

Description

Write code in the OnMouseWheelUp event handler to perform actions when the mouse wheel is rotated upward. OnMouseWheelUp occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheel (see page 5869)

75.1.4.3.25 TDrawGrid.OnRowMoved Event

Occurs immediately after the position of a row changes.

Pascal

```
property OnRowMoved: TMovedEvent;
```

C++

```
__property TMovedEvent OnRowMoved;
```

Description

Write an OnRowMoved event handler to provide special processing whenever a row in the grid changes position. Rows can be moved programmatically or by user manipulation. The FromIndex parameter is the old index of the row, and the ToIndex parameter is the new index. OnRowMoved does not occur unless the Options (see page 14892) property includes goRowMoving.

See Also

OnColumnMoved (see page 14908)

Options (see page 14853)

75.1.4.3.26 TDrawGrid.OnSelectCell Event

Occurs before a cell in the grid is selected.

Pascal

```
property OnSelectCell: TSelectCellEvent;
```

C++

```
__property TSelectCellEvent OnSelectCell;
```

Description

Write an OnSelectCell event handler to specify whether any particular cell in the grid can be selected.

See Also

Col (see page 14843)

Row (see page 14854)

TabStops (see page 14858)

75.1.4.3.27 TDrawGrid.OnSetEditText Event

Occurs when the user edits the value of a cell.

Pascal

```
property OnSetEditText: TSetEditEvent;
```

C++

```
__property TSetEditEvent OnSetEditText;
```

Description

Write an OnSetEditText event handler to perform any special processing of the text edited by the user in an in-place editor. For example, use the OnSetEditText event to retrieve and store the value of a cell so that it can be displayed in an OnDrawCell (see page 14914) event handler. OnSetEditText occurs every time the user changes the text.

OnSetEditText does not occur unless the Options (see page 14892) property includes goEditing.

See Also

OnDrawCell (see page 14914)

OnGetEditText (see page 14920)

Options (see page 14853)

75.1.4.3.28 TDrawGrid.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

75.1.4.3.29 TDrawGrid.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

75.1.4.3.30 TDrawGrid.OnTopLeftChanged Event

Occurs immediately after the TopRow (see page 14793) property or the LeftCol (see page 14790) property changes.

Pascal

```
property OnTopLeftChanged: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnTopLeftChanged;
```

Description

Use OnTopLeftChanged to perform special processing when the non-fixed cells in the grid are scrolled.

See Also

LeftCol (see page 14852)

TopRow (see page 14858)

75.1.5 TInplaceEdit Class

TInplaceEdit provides text editing capabilities for grid controls.

Class Hierarchy

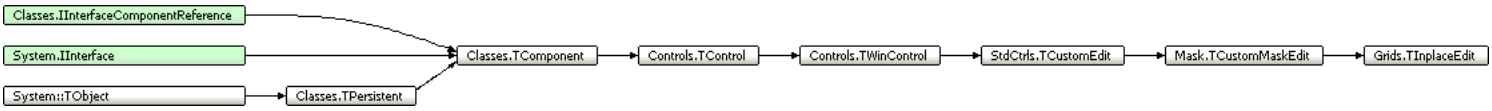

Pascal

```
TInplaceEdit = class(TCustomMaskEdit);
```

C++

```
class TInplaceEdit : public TCustomMaskEdit;
```

File

Grids

Description

TInplaceEdit is a helper object tailored to work with grid controls. It provides a multiline editing window that appears in the appropriate cell of a grid control. TInplaceEdit provides services so that the grid can influence the response of the control to user actions, and can move or resize the editing window as cells are moved or resized.

The TInplaceEdit window has no border, so it does not stand out as a separate control. TInplaceEdit is not a data-aware control, but can work with a data-aware grid such as TDBGrid, allowing the grid to fetch or post the text, and allowing the grid to impose a mask to limit the text of the edit control to data that is valid.

See Also

TCustomGrid (see page 14800)

TDBEdit (see page 8735)

TDBGrid (see page 9668)

75.1.5.1 TInplaceEdit Members

The following tables list the members exposed by TInplaceEdit.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnChange (see page 21275)	Occurs when the text for the edit control may have changed.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.

	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
V	BoundsChanged (see page 14951)	Indents the text area of the window slightly after the grid repositions the edit control.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21245)	Generates an OnChange (see page 21275) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CheckCursor (see page 15811)	Moves the cursor to the closest editable character and selects it if nothing is currently selected.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 14952)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWindowHandle (see page 21248)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CreateWnd (see page 21248)	Creates a Windows control corresponding to the edit control.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
V	DblClick (see page 14952)	Informs the grid when the user double-clicks in the edit window.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.

	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21251)	Destroys the Windows control that corresponds to the edit control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 14953)	Delegates mouse wheel messages to the associated grid.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoSetMaxLength (see page 21251)	Sets the MaxLength (see page 21263) property.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EditCanModify (see page 14953)	Determines whether the user can edit the Text.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.

	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFirstEditChar (see page 15813)	Returns the index in the EditMask (see page 15821) of the first nonliteral character.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetLastEditChar (see page 15813)	Returns the index in the EditMask (see page 15821) of the last nonliteral character.
	GetMaxChars (see page 15814)	Returns the length of the EditText (see page 15821).
	GetNextEditChar (see page 15814)	Returns the first index of a nonliteral character equal to or greater than an offset.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetPriorEditChar (see page 15815)	Returns the first index of a nonliteral character equal to or less than an offset.
	GetSel (see page 15815)	Returns the endpoints of the current selection in the EditText (see page 15821).
	GetSelLength (see page 21252)	Returns the SelLength (see page 21265) property.
	GetSelStart (see page 21252)	Returns the SelStart (see page 21269) property.
	GetSelText (see page 21252)	Returns the SelText (see page 21272) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 14954)	Responds when the user presses a key.
	KeyPress (see page 14955)	Responds when the user types a key.
	KeyUp (see page 14955)	Responds when the user releases a key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.

	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReformatText (see page 15817)	Applies the mask NewMask to the underlying text, and updates the internal representation of the mask.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
V	Reset (see page 15818)	Returns the EditText (see page 15821) to its value at the time when the edit control last lost focus.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetAutoSize (see page 21257)	Sets the AutoSize (see page 21260) property.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetCursor (see page 15818)	Moves the cursor to the first editable character at or before Pos, and selects that character.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.

	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSel (see page 15819)	Selects character range.
	SetSelLength (see page 21257)	Sets the SelLength (see page 21265) property.
	SetSelStart (see page 21257)	Sets the SelStart (see page 21269) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateContents (see page 14957)	Sets the EditMask and Text properties to values specified by the grid.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	Validate (see page 15819)	Validates a specified string against the current mask.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateError (see page 15820)	Beeps and raises an EDBEditError (see page 15798) exception.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 14958)	Supplements the inherited method to add special processing to some Windows messages.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSelect (see page 21260)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AutoSize (see page 21260)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 21261)	Determines whether the edit control has a single line border around the client area.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	CharCase (see page 21262)	Determines the case of the text within the edit control.
	Color (see page 5812)	Specifies the background color of the control.

	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EditMask (see page 15821)	Specifies the mask that represents what text is valid for the masked edit control.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
R	Grid (see page 14958)	Specifies the grid object that owns and controls this TInplaceEdit (see page 14938).
	HideSelection (see page 21262)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MaskState (see page 15822)	Indicates state information about how the mask is being applied.
	MaxLength (see page 15822)	Specifies the maximum number of characters that can appear in the edit control.
	MaxLength (see page 21263)	Specifies the maximum number of characters the user can enter into the edit control.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	OEMConvert (see page 21264)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 21264)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PasswordChar (see page 21265)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.

	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 15812)	Sets the Text (see page 15823) property to an empty string.
V	Clear (see page 21245)	Deletes all text from the edit control.
	ClearSelection (see page 21245)	Removes the selected text from the edit control.
	ClearUndo (see page 21246)	Clears the undo buffer so that no changes to the text can be backed out.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopyToClipboard (see page 21246)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
V	Create (see page 14950)	Creates an instance of TInplaceEdit (see page 14938).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CutToClipboard (see page 21249)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 21250)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Deselect (see page 14952)	Removes the selection from the edit window.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.

	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
V	GetSelTextBuf (see page 21253)	Copies the selected text into a buffer and returns the number of characters copied.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	GetTextLen (see page 15816)	Returns the length of the underlying text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 14953)	Hides the edit control window and moves input focus to the grid.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 14954)	Invalidates the edit window and the corresponding portion of the grid window.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	Move (see page 14956)	Changes the position of the edit window and repaints the contents.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	PasteFromClipboard (see page 21254)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PosEqual (see page 14956)	Indicates whether the edit window has the size and position indicated by Rect.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectAll (see page 21255)	Selects all text in the edit control.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
	SetFocus (see page 14956)	Gives the editor window input focus if it is visible.
	SetSelTextBuf (see page 21258)	Replaces the selected text with a null-terminated string.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Undo (see page 21259)	Backs out all changes in the undo buffer.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UpdateLoc (see page 14957)	Changes the position of the edit window but does not repaint the contents.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.
V	ValidateEdit (see page 15819)	Validates the EditText (see page 15821) against the current mask.
	Visible (see page 14958)	This is Visible, a member of class TInplaceEdit.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	CanUndo (see page 21262)	Indicates whether the edit control contains changes that can be backed out.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.

R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EditText (see page 15821)	Specifies the text that appears in the window of the masked edit control.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
R	IsMasked (see page 15822)	Indicates whether a mask exists for the edit control.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Modified (see page 21263)	Indicates whether the user edited the text of the edit control.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ReadOnly (see page 21265)	Determines whether the user can change the text of the edit control.
	SelLength (see page 21265)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 21269)	Specifies the position of the first selected character in the text.
	SelText (see page 21272)	Specifies the selected portion of the edit control's text.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.

	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	Text (see page 15823)	Denotes the underlying text for the masked edit control before the mask has been applied.
	Text (see page 21273)	Contains a text string associated with the control.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	TabStop (see page 21273)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

75.1.5.2 TInplaceEdit.Create Constructor

Creates an instance of TInplaceEdit (see page 14938).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TInplaceEdit(TComponent * AOwner);
```

Description

Do not create instances of TInplaceEdit (see page 14938). These are created automatically by the grid that uses the editor.

AOwner specifies the Owner for the editor. This is the component that is responsible for freeing the TInplaceEdit (see page 14938) instance.

Create overrides the default constructor to make sure that the control has no border to provide a visual indication that the control exists.

See Also

Hide (see page 14953)

BoundsChanged (see page 14951)

Owner (see page 2807)

75.1.5.3 TInplaceEdit Methods

The methods of the TInplaceEdit class are listed here.

Protected Methods

	Name	Description
	BoundsChanged (see page 14951)	Indents the text area of the window slightly after the grid repositions the edit control.
	CreateParams (see page 14952)	Initializes a window-creation parameter record passed in the Params parameter.
	DblClick (see page 14952)	Informs the grid when the user double-clicks in the edit window.
	DoMouseWheel (see page 14953)	Delegates mouse wheel messages to the associated grid.
	EditCanModify (see page 14953)	Determines whether the user can edit the Text.
	KeyDown (see page 14954)	Responds when the user presses a key.
	KeyPress (see page 14955)	Responds when the user types a key.
	KeyUp (see page 14955)	Responds when the user releases a key.
	UpdateContents (see page 14957)	Sets the EditMask and Text properties to values specified by the grid.
	WndProc (see page 14958)	Supplements the inherited method to add special processing to some Windows messages.

Public Methods

	Name	Description
	Deselect (see page 14952)	Removes the selection from the edit window.
	Hide (see page 14953)	Hides the edit control window and moves input focus to the grid.
	Invalidate (see page 14954)	Invalidates the edit window and the corresponding portion of the grid window.
	Move (see page 14956)	Changes the position of the edit window and repaints the contents.
	PosEqual (see page 14956)	Indicates whether the edit window has the size and position indicated by Rect.
	SetFocus (see page 14956)	Gives the editor window input focus if it is visible.
	UpdateLoc (see page 14957)	Changes the position of the edit window but does not repaint the contents.
	Visible (see page 14958)	This is Visible, a member of class TInplaceEdit.

75.1.5.3.1 TInplaceEdit.BoundsChanged Method

Indents the text area of the window slightly after the grid repositions the edit control.

Pascal

```
procedure BoundsChanged; virtual;
```

C++

```
virtual __fastcall BoundsChanged();
```

Description

BoundsChanged reinitializes the editing window after the control has been moved or resized. It creates a slight indentation in the text formatting area so that the contents of the editor do not appear to crowd the edges of the cell.

BoundsChanged is exposed as a protected method so that derived objects can add further enhancements when the editing window changes size or location.

See Also

Move (see page 14956)

UpdateLoc (see page 14957)

PosEqual (see page 14956)

75.1.5.3.2 TInplaceEdit.CreateParams Method

Initializes a window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. TInplaceEdit (see page 14938) overrides the inherited method to specify a multiline edit control.

See Also

CreateWnd (see page 21248)

CreateWindowHandle (see page 6137)

75.1.5.3.3 TInplaceEdit.DblClick Method

Informs the grid when the user double-clicks in the edit window.

Pascal

```
procedure DblClick; override;
```

C++

```
virtual __fastcall DblClick();
```

Description

DblClick is called when the user double-clicks in the edit window. DblClick simply passes the event on to the grid. The grid must interpret and respond to the double-click. If the edit window needs to respond to the double-click, the grid must trigger the response by calling a method or setting a property of the TInplaceEdit (see page 14938).

See Also

Grid (see page 14958)

75.1.5.3.4 TInplaceEdit.Deselect Method

Removes the selection from the edit window.

Pascal

```
procedure Deselect;
```

C++

```
__fastcall Deselect();
```

Description

Use Deselect to cause all text in the edit window to be unselected. Deselect simply changes the selection, it does not change the actual text.

See Also

ClearSelection (see page 21245)

SelectAll (see page 21255)

KeyPress (see page 14955)

75.1.5.3.5 TInplaceEdit.DoMouseWheel Method

Delegates mouse wheel messages to the associated grid.

Pascal

```
function DoMouseWheel(Shift: TShiftState; WheelDelta: Integer; MousePos: TPoint): Boolean;
override;
```

C++

```
virtual __fastcall Boolean DoMouseWheel(TShiftState Shift, int WheelDelta, TPoint MousePos);
```

Description

TInplaceEdit (see page 14938) overrides the inherited DoMouseWheel method so that it can pass any mouse wheel messages on to the grid specified by the Grid (see page 14958) property. This causes the grid's OnMouseWheel event handler to execute even though the inplace edit control has focus.

See Also

OnMouseWheel (see page 5869)

Grid (see page 14958)

75.1.5.3.6 TInplaceEdit.EditCanModify Method

Determines whether the user can edit the Text.

Pascal

```
function EditCanModify: Boolean; override;
```

C++

```
virtual __fastcall Boolean EditCanModify();
```

Description

EditCanModify is called automatically when the user tries to edit the value of Text. EditCanModify calls the CanEditModify method of the grid to determine whether the current cell of the grid should be edited. When EditCanModify returns true, the user is permitted to edit the text. When EditCanModify returns false, the user is prevented from editing the text.

See Also

Grid (see page 14958)

CanEditModify (see page 14819)

Text (see page 5842)

75.1.5.3.7 TInplaceEdit.Hide Method

Hides the edit control window and moves input focus to the grid.

Pascal

```
procedure Hide;
```

C++

```
__fastcall Hide();
```

Description

Hide makes the edit control window invisible without destroying it. Hide also moves the input focus from the editor window to the grid, so that the invisible window does not swallow keyboard events.

The grid calls Hide to hide the edit window when changing out of edit mode. It also calls Hide when scrolling the cell that is being edited out of the grid window.

See Also

Grid (see page 14958)

75.1.5.3.8 TInplaceEdit.Invalidate Method

Invalidates the edit window and the corresponding portion of the grid window.

Pascal

```
procedure Invalidate; reintroduce;
```

C++

```
__fastcall Invalidate();
```

Description

Invalidate ensures that the portion of the grid window under the edit control is invalidated at the same time as the edit window. This is important when the edit window is hidden or destroyed so that the grid window can repaint the cell that was covered by the edit window.

See Also

Grid (see page 14958)

Hide (see page 14953)

75.1.5.3.9 TInplaceEdit.KeyDown Method

Responds when the user presses a key.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When the user presses a key, KeyDown checks whether the edit control or grid should process the KeyDown event. The Enter key is always passed on to the grid. The F2 key calls Deselect (see page 14952) to remove the current selection. Navigation keystrokes are passed on to the grid when the movement cannot be accommodated within the text window. For example, if the user types Left when the cursor is already at the left edge of the edit window, the keystroke is passed to the grid which moves focus to the previous cell. If the cursor is in the middle of a line, the keystroke is processed by the edit window, which moves left a single character.

KeyDown calls the OnKeyDown event handler for the grid, if there is one, before calling the inherited KeyDown for the edit control.

See Also

Grid (see page 14958)

KeyPress (see page 14955)

KeyUp (see page 14955)

KeyDown (see page 15816)

75.1.5.3.10 TInplaceEdit.KeyPress Method

Responds when the user types a key.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress allows the grid to handle the keystroke, and only processes keystrokes that the grid does not fully handle. If the grid is not in edit mode after handling the keystroke, KeyPress throws away the keystroke. Keystroke processes the Return Key and calls the inherited method to handle all other unprocessed keystrokes.

Pressing the Return key causes KeyPress to select all the text, unless it is already selected, in which case it causes KeyPress to deselect all the text.

See Also

Grid (see page 14958)

SelectAll (see page 21255)

Deselect (see page 14952)

KeyPress (see page 15816)

KeyDown (see page 14954)

KeyUp (see page 14955)

75.1.5.3.11 TInplaceEdit.KeyUp Method

Responds when the user releases a key.

Pascal

```
procedure KeyUp(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyUp(Word Key, TShiftState Shift);
```

Description

The inplace edit control does not process the KeyUp event at all. It simply passes this event on to the grid, which may or may not process the event.

See Also

Grid (see page 14958)

KeyDown (see page 14954)

KeyPress (see page 14955)

75.1.5.3.12 TInplaceEdit.Move Method

Changes the position of the edit window and repaints the contents.

Pascal

```
procedure Move(const Loc: TRect);
```

C++

```
__fastcall Move(const TRect Loc);
```

Description

The grid calls Move to move the edit window to the indicated rectangle when the corresponding cell moves or resizes. The position of Loc is relative to the grid's client area.

See Also

Grid (see page 14958)

UpdateLoc (see page 14957)

75.1.5.3.13 TInplaceEdit.PosEqual Method

Indicates whether the edit window has the size and position indicated by Rect.

Pascal

```
function PosEqual(const Rect: TRect): Boolean;
```

C++

```
__fastcall Boolean PosEqual(const TRect Rect);
```

Description

Use PosEqual to determine if the edit window occupies a particular rectangle, where Rect is given in global coordinates. PosEqual can be used to determine whether the inplace edit control represents a particular cell in the grid, by checking the bounding rectangle of the cell.

See Also

BoundsRect (see page 5803)

TCustomGrid (see page 14800)

75.1.5.3.14 TInplaceEdit.SetFocus Method

Gives the editor window input focus if it is visible.

Pascal

```
procedure SetFocus; reintroduce;
```

C++

```
__fastcall SetFocus();
```

Description

Use SetFocus to cause keyboard events to go to the editor window. Focus will not move to the window if it is hidden by a call to

the Hide (see page 14953) method.

See Also

KeyPress (see page 14955)

KeyDown (see page 14954)

KeyUp (see page 14955)

Hide (see page 14953)

75.1.5.3.15 TInplaceEdit.UpdateContents Method

Sets the EditMask and Text properties to values specified by the grid.

Pascal

```
procedure UpdateContents; virtual;
```

C++

```
virtual __fastcall UpdateContents();
```

Description

UpdateContents calls the GetEditMask and GetEditText methods of the grid, and sets the EditMask and Text properties accordingly. UpdateContents allows the grid to bring up the inplace edit control with the contents of a particular cell, and to impose a mask to restrict the editor to valid input for the particular cell.

See Also

Grid (see page 14958)

EditMask (see page 15821)

Text (see page 5842)

GetEditMask (see page 14828)

GetEditText (see page 14829)

75.1.5.3.16 TInplaceEdit.UpdateLoc Method

Changes the position of the edit window but does not repaint the contents.

Pascal

```
procedure UpdateLoc(const Loc: TRect);
```

C++

```
__fastcall UpdateLoc(const TRect Loc);
```

Description

The grid calls UpdateLoc to move the edit window when the grid is moved. Unlike the Move (see page 14956) method, UpdateLoc does not invalidate the edit window to cause a repaint.

See Also

Grid (see page 14958)

Move (see page 14956)

Invalidate (see page 14954)

75.1.5.3.17 TInplaceEdit.Visible Method

Pascal

```
function Visible: Boolean;
```

C++

```
__fastcall Boolean Visible();
```

Description

This is Visible, a member of class TInplaceEdit.

75.1.5.3.18 TInplaceEdit.WndProc Method

Supplements the inherited method to add special processing to some Windows messages.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc is the first method of a control that receives Windows messages.

WndProc for TInplaceEdit (see page 14938) passes focus events on to the grid. It calls the inherited method for all other Windows messages. WndProc converts mouse clicks to double clicks when the grid receives the first click within the double-click time period.

See Also

Grid (see page 14958)

WndProc (see page 6187)

75.1.5.4 TInplaceEdit Properties

The properties of the TInplaceEdit class are listed here.

Protected Properties

	Name	Description
R	Grid (see page 14958)	Specifies the grid object that owns and controls this TInplaceEdit (see page 14938).

75.1.5.4.1 TInplaceEdit.Grid Property

Specifies the grid object that owns and controls this TInplaceEdit (see page 14938).

Pascal

```
property Grid: TCustomGrid;
```

C++

```
__property TCustomGrid Grid;
```

Description

Read Grid to obtain the grid object for which TInplaceEdit (see page 14938) provides editing services. TInplaceEdit (see page 14938) uses the Grid property so that it can receive information about the text to be edited and so that it can pass on relevant events.

See Also

UpdateContents (see page 14957)

UpdateLoc (see page 14957)

75.1.6 TInplaceEditList Class

Class Hierarchy

Pascal

```
TInplaceEditList = class(TInPlaceEdit);
```

C++

```
class TInplaceEditList : public TInPlaceEdit;
```

File

Grids

Description

This is class Grids.TInplaceEditList.

75.1.6.1 TInplaceEditList Members

The following tables list the members exposed by TInplaceEditList.

Protected Methods

	Name	Description
	BoundsChanged (see page 14961)	This is BoundsChanged, a member of class TInplaceEditList.
	ButtonRect (see page 14961)	This is ButtonRect, a member of class TInplaceEditList.
	CloseUp (see page 14961)	This is CloseUp, a member of class TInplaceEditList.
	DblClick (see page 14961)	This is DblClick, a member of class TInplaceEditList.
	DoDropDownKeys (see page 14962)	This is DoDropDownKeys, a member of class TInplaceEditList.
	DoEditButtonClick (see page 14962)	This is DoEditButtonClick, a member of class TInplaceEditList.
	DoGetPickListItems (see page 14962)	This is DoGetPickListItems, a member of class TInplaceEditList.
	DropDown (see page 14962)	This is DropDown, a member of class TInplaceEditList.
	KeyDown (see page 14963)	This is KeyDown, a member of class TInplaceEditList.
	ListMouseUp (see page 14963)	This is ListMouseUp, a member of class TInplaceEditList.
	MouseDown (see page 14963)	This is MouseDown, a member of class TInplaceEditList.
	MouseMove (see page 14963)	This is MouseMove, a member of class TInplaceEditList.
	MouseUp (see page 14963)	This is MouseUp, a member of class TInplaceEditList.
	OverButton (see page 14964)	This is OverButton, a member of class TInplaceEditList.
	PaintWindow (see page 14964)	This is PaintWindow, a member of class TInplaceEditList.
	StopTracking (see page 14964)	This is StopTracking, a member of class TInplaceEditList.

	TrackButton (see page 14965)	This is TrackButton, a member of class TInplaceEditList.
V	UpdateContents (see page 14965)	This is UpdateContents, a member of class TInplaceEditList.
V	WndProc (see page 14965)	This is WndProc, a member of class TInplaceEditList.

Public Events

	Name	Description
	OnEditButtonClick (see page 14967)	This is OnEditButtonClick, a member of class TInplaceEditList.
	OnGetPickListitems (see page 14968)	This is OnGetPickListitems, a member of class TInplaceEditList.

Public Methods

	Name	Description
V	Create (see page 14960)	This is Create, a member of class TInplaceEditList.
	RestoreContents (see page 14964)	This is RestoreContents, a member of class TInplaceEditList.

Public Properties

	Name	Description
	ActiveList (see page 14966)	This is ActiveList, a member of class TInplaceEditList.
	ButtonWidth (see page 14966)	This is ButtonWidth, a member of class TInplaceEditList.
	DropDownRows (see page 14966)	This is DropDownRows, a member of class TInplaceEditList.
R	EditStyle (see page 14966)	This is EditStyle, a member of class TInplaceEditList.
	ListVisible (see page 14966)	This is ListVisible, a member of class TInplaceEditList.
R	PickList (see page 14967)	This is PickList, a member of class TInplaceEditList.
	PickListLoaded (see page 14967)	This is PickListLoaded, a member of class TInplaceEditList.
R	Pressed (see page 14967)	This is Pressed, a member of class TInplaceEditList.

75.1.6.2 TInplaceEditList.Create Constructor

Pascal

```
constructor Create(Owner: TComponent); override;
```

C++

```
virtual __fastcall TInplaceEditList(TComponent * Owner);
```

Description

This is Create, a member of class TInplaceEditList.

75.1.6.3 TInplaceEditList Methods

The methods of the TInplaceEditList class are listed here.

Protected Methods

	Name	Description
V	BoundsChanged (see page 14961)	This is BoundsChanged, a member of class TInplaceEditList.
	ButtonRect (see page 14961)	This is ButtonRect, a member of class TInplaceEditList.
	CloseUp (see page 14961)	This is CloseUp, a member of class TInplaceEditList.
V	DblClick (see page 14961)	This is DblClick, a member of class TInplaceEditList.
V	DoDropDownKeys (see page 14962)	This is DoDropDownKeys, a member of class TInplaceEditList.
V	DoEditButtonClick (see page 14962)	This is DoEditButtonClick, a member of class TInplaceEditList.
	DoGetPickListItems (see page 14962)	This is DoGetPickListItems, a member of class TInplaceEditList.
	DropDown (see page 14962)	This is DropDown, a member of class TInplaceEditList.
V	KeyDown (see page 14963)	This is KeyDown, a member of class TInplaceEditList.
	ListMouseUp (see page 14963)	This is ListMouseUp, a member of class TInplaceEditList.

	MouseDown (see page 14963)	This is MouseDown, a member of class TInplaceEditList.
	MouseMove (see page 14963)	This is MouseMove, a member of class TInplaceEditList.
	MouseUp (see page 14963)	This is MouseUp, a member of class TInplaceEditList.
	OverButton (see page 14964)	This is OverButton, a member of class TInplaceEditList.
	PaintWindow (see page 14964)	This is PaintWindow, a member of class TInplaceEditList.
	StopTracking (see page 14964)	This is StopTracking, a member of class TInplaceEditList.
	TrackButton (see page 14965)	This is TrackButton, a member of class TInplaceEditList.
	UpdateContents (see page 14965)	This is UpdateContents, a member of class TInplaceEditList.
	WndProc (see page 14965)	This is WndProc, a member of class TInplaceEditList.

Public Methods

	Name	Description
	RestoreContents (see page 14964)	This is RestoreContents, a member of class TInplaceEditList.

75.1.6.3.1 TInplaceEditList.BoundsChanged Method

Pascal

```
procedure BoundsChanged; override;
```

C++

```
virtual __fastcall BoundsChanged();
```

Description

This is BoundsChanged, a member of class TInplaceEditList.

75.1.6.3.2 TInplaceEditList.ButtonRect Method

Pascal

```
function ButtonRect: TRect;
```

C++

```
__fastcall TRect ButtonRect();
```

Description

This is ButtonRect, a member of class TInplaceEditList.

75.1.6.3.3 TInplaceEditList.CloseUp Method

Pascal

```
procedure CloseUp(Accept: Boolean); dynamic;
```

C++

```
__fastcall CloseUp(Boolean Accept);
```

Description

This is CloseUp, a member of class TInplaceEditList.

75.1.6.3.4 TInplaceEditList.DblClick Method

Pascal

```
procedure DblClick; override;
```

C++

```
virtual __fastcall DblClick();
```

Description

This is DblClick, a member of class TInplaceEditList.

75.1.6.3.5 TInplaceEditList.DoDropDownKeys Method

Pascal

```
procedure DoDropDownKeys(var Key: Word; Shift: TShiftState); virtual;
```

C++

```
virtual __fastcall DoDropDownKeys(Word Key, TShiftState Shift);
```

Description

This is DoDropDownKeys, a member of class TInplaceEditList.

75.1.6.3.6 TInplaceEditList.DoEditButtonClick Method

Pascal

```
procedure DoEditButtonClick; virtual;
```

C++

```
virtual __fastcall DoEditButtonClick();
```

Description

This is DoEditButtonClick, a member of class TInplaceEditList.

75.1.6.3.7 TInplaceEditList.DoGetPickListItems Method

Pascal

```
procedure DoGetPickListItems; dynamic;
```

C++

```
__fastcall DoGetPickListItems();
```

Description

This is DoGetPickListItems, a member of class TInplaceEditList.

75.1.6.3.8 TInplaceEditList.DropDown Method

Pascal

```
procedure DropDown; dynamic;
```

C++

```
__fastcall DropDown();
```

Description

This is DropDown, a member of class TInplaceEditList.

75.1.6.3.9 TInplaceEditList.KeyDown Method

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

This is KeyDown, a member of class TInplaceEditList.

75.1.6.3.10 TInplaceEditList.ListMouseUp Method

Pascal

```
procedure ListMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X: Integer;
Y: Integer);
```

C++

```
__fastcall ListMouseUp(TObject * Sender, TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

This is ListMouseUp, a member of class TInplaceEditList.

75.1.6.3.11 TInplaceEditList.MouseDown Method

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

This is MouseDown, a member of class TInplaceEditList.

75.1.6.3.12 TInplaceEditList.MouseMove Method

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

This is MouseMove, a member of class TInplaceEditList.

75.1.6.3.13 TInplaceEditList.MouseUp Method

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

This is MouseUp, a member of class TInplaceEditList.

75.1.6.3.14 TInplaceEditList.OverButton Method

Pascal

```
function OverButton(const P: TPoint): Boolean;
```

C++

```
__fastcall Boolean OverButton(const TPoint P);
```

Description

This is OverButton, a member of class TInplaceEditList.

75.1.6.3.15 TInplaceEditList.PaintWindow Method

Pascal

```
procedure PaintWindow(DC: HDC); override;
```

C++

```
virtual __fastcall PaintWindow(HDC DC);
```

Description

This is PaintWindow, a member of class TInplaceEditList.

75.1.6.3.16 TInplaceEditList.RestoreContents Method

Pascal

```
procedure RestoreContents;
```

C++

```
__fastcall RestoreContents();
```

Description

This is RestoreContents, a member of class TInplaceEditList.

75.1.6.3.17 TInplaceEditList.StopTracking Method

Pascal

```
procedure StopTracking;
```

C++

```
__fastcall StopTracking();
```

Description

This is StopTracking, a member of class TInplaceEditList.

75.1.6.3.18 TInplaceEditList.TrackButton Method

Pascal

```
procedure TrackButton(X: Integer; Y: Integer);
```

C++

```
__fastcall TrackButton(int X, int Y);
```

Description

This is TrackButton, a member of class TInplaceEditList.

75.1.6.3.19 TInplaceEditList.UpdateContents Method

Pascal

```
procedure UpdateContents; override;
```

C++

```
virtual __fastcall UpdateContents();
```

Description

This is UpdateContents, a member of class TInplaceEditList.

75.1.6.3.20 TInplaceEditList.WndProc Method

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

This is WndProc, a member of class TInplaceEditList.

75.1.6.4 TInplaceEditList Properties

The properties of the TInplaceEditList class are listed here.

Public Properties

	Name	Description
	ActiveList (see page 14966)	This is ActiveList, a member of class TInplaceEditList.
	ButtonWidth (see page 14966)	This is ButtonWidth, a member of class TInplaceEditList.
	DropDownRows (see page 14966)	This is DropDownRows, a member of class TInplaceEditList.
R	EditStyle (see page 14966)	This is EditStyle, a member of class TInplaceEditList.
	ListVisible (see page 14966)	This is ListVisible, a member of class TInplaceEditList.
R	PickList (see page 14967)	This is PickList, a member of class TInplaceEditList.
	PickListLoaded (see page 14967)	This is PickListLoaded, a member of class TInplaceEditList.
R	Pressed (see page 14967)	This is Pressed, a member of class TInplaceEditList.

75.1.6.4.1 TInplaceEditList.ActiveList Property

Pascal

```
property ActiveList: TWinControl;
```

C++

```
__property TWinControl * ActiveList;
```

Description

This is ActiveList, a member of class TInplaceEditList.

75.1.6.4.2 TInplaceEditList.ButtonWidth Property

Pascal

```
property ButtonWidth: Integer;
```

C++

```
__property int ButtonWidth;
```

Description

This is ButtonWidth, a member of class TInplaceEditList.

75.1.6.4.3 TInplaceEditList.DropDownRows Property

Pascal

```
property DropDownRows: Integer;
```

C++

```
__property int DropDownRows;
```

Description

This is DropDownRows, a member of class TInplaceEditList.

75.1.6.4.4 TInplaceEditList.EditStyle Property

Pascal

```
property EditStyle: TEditStyle;
```

C++

```
__property TEditStyle EditStyle;
```

Description

This is EditStyle, a member of class TInplaceEditList.

75.1.6.4.5 TInplaceEditList.ListVisible Property

Pascal

```
property ListVisible: Boolean;
```

C++

```
__property Boolean ListVisible;
```

Description

This is ListVisible, a member of class TInplaceEditList.

75.1.6.4.6 TInplaceEditList.PickList Property

Pascal

```
property PickList: TCustomListbox;
```

C++

```
__property TCustomListbox PickList;
```

Description

This is PickList, a member of class TInplaceEditList.

75.1.6.4.7 TInplaceEditList.PickListLoaded Property

Pascal

```
property PickListLoaded: Boolean;
```

C++

```
__property Boolean PickListLoaded;
```

Description

This is PickListLoaded, a member of class TInplaceEditList.

75.1.6.4.8 TInplaceEditList.Pressed Property

Pascal

```
property Pressed: Boolean;
```

C++

```
__property Boolean Pressed;
```

Description

This is Pressed, a member of class TInplaceEditList.

75.1.6.5 TInplaceEditList Events

The events of the TInplaceEditList class are listed here.

Public Events

	Name	Description
	OnEditButtonClick (see page 14967)	This is OnEditButtonClick, a member of class TInplaceEditList.
	OnGetPickListitems (see page 14968)	This is OnGetPickListitems, a member of class TInplaceEditList.

75.1.6.5.1 TInplaceEditList.OnEditButtonClick Event

Pascal

```
property OnEditButtonClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEditButtonClick;
```

Description

This is OnEditButtonClick, a member of class TInplaceEditList.

75.1.6.5.2 TInplaceEditList.OnGetPickListitems Event

Pascal

```
property OnGetPickListitems: TOnGetPickListItems;
```

C++

```
__property TOnGetPickListItems OnGetPickListitems;
```

Description

This is OnGetPickListitems, a member of class TInplaceEditList.

75.1.7 TStringGrid Class

TStringGrid represents a grid control designed to simplify the handling of strings and associated objects.

Class Hierarchy

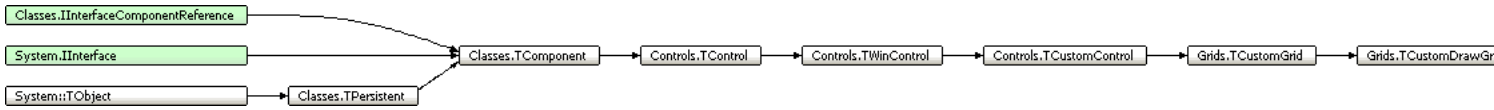

Pascal

```
TStringGrid = class(TDrawGrid);
```

C++

```
class TStringGrid : public TDrawGrid;
```

File

Grids

Description

Add a TStringGrid object to a form to present textual data in a tabular format. TStringGrid provides many properties to control the appearance of the grid, as well as events and methods that take advantage of the tabular organization of the grid in responding to user actions.

TStringGrid introduces the ability to associate an object with each string in the grid. These objects can encapsulate any information or behavior represented by the strings that are presented to the user.

If the strings to be presented in a grid represent field values from the records in a dataset, use TDBGrid instead.

See Also

TDBGrid (see page 9668)

TValueListEditor (see page 23401)

String Grids

Grids

75.1.7.1 TStringGrid Members

The following tables list the members exposed by TStringGrid.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnColumnMoved (see page 14794)	Occurs immediately after the position of a column changes.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDrawCell (see page 14795)	Occurs when a cell in the grid needs to be drawn.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetEditMask (see page 14796)	Occurs when the in-place editor requests an edit mask.
	OnGetEditText (see page 14797)	Occurs when the in-place editor requests the value of a cell.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnRowMoved (see page 14798)	Occurs immediately after the position of a row changes.
	OnSelectCell (see page 14798)	Occurs before a cell in the grid is selected.
	OnSetEditText (see page 14800)	Occurs when the user edits the value of a cell.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

	OnTopLeftChanged (see page 14800)	Occurs immediately after the TopRow (see page 14793) property or the LeftCol (see page 14790) property changes.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 14814)	This is DesignOptionsBoost, a member of class TCustomGrid.
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FGridState (see page 14814)	This is FGridState, a member of class TCustomGrid.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveCellExtents (see page 14814)	This is FSaveCellExtents, a member of class TCustomGrid.
	VirtualView (see page 14814)	This is VirtualView, a member of class TCustomGrid.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AdjustSize (see page 14816)	Adds or deletes rows or columns from the grid.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	BeginColumnDrag (see page 14817)	Starts the dragging of a column in the grid.
	BeginRowDrag (see page 14817)	Starts the dragging of a row in the grid.
	BoxRect (see page 14818)	Returns the screen coordinates of a specified set of cells.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcDrawInfo (see page 14818)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 14818)	Calculates the pixel locations of the fixed cells.
	CalcSizingState (see page 14819)	Determines the appearance of the mouse cursor during resizing.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanEditAcceptKey (see page 14819)	Indicates whether the Key parameter can be entered into the current cell.
	CanEditModify (see page 14819)	Indicates whether the current cell in the grid can be edited.

	CanEditShow (see page 14820)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 14820)	Indicates whether the grid should respond to a given key combination.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	CellRect (see page 14821)	Returns the screen coordinates of a cell in the grid.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeGridOrientation (see page 14821)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CheckColumnDrag (see page 14821)	CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 14822)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
	Click (see page 5743)	Respond to user click.
	ColumnMoved (see page 14984)	Generates an OnColumnMoved event.
	ColWidthsChanged (see page 14822)	Responds when the column widths change.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateEditor (see page 14823)	Creates the inplace editor control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 14823)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DefineProperties (see page 14824)	Adds the ColWidths (see page 14845) and RowHeights (see page 14855) properties to the values saved with the grid.
	DeleteColumn (see page 14824)	Deletes a specified column from the grid.
	DeleteRow (see page 14825)	Deletes a specified from the grid.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.

	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
V	DoExit (see page 14825)	Hides the inplace editor when selection moves to another control in a form.
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
V	DoMouseWheelDown (see page 14825)	Performs tasks when the mouse wheel is rotated downward.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
V	DoMouseWheelUp (see page 14826)	Performs tasks when the mouse wheel is rotated upward.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawCell (see page 14984)	Generates an OnDrawCell event.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EndColumnDrag (see page 14827)	Ends the dragging of a column in the grid.
	EndRowDrag (see page 14827)	Ends the dragging of a row in the grid.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	FocusCell (see page 14827)	Changes the current cell to a specified cell.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEditLimit (see page 14828)	Provides the interface for a method that limits the number of characters the user can type into cells.
V	GetEditMask (see page 14782)	Generates an OnGetEditMask (see page 14796) event.
	GetEditMask (see page 14828)	Provides the interface for a method that returns an input mask for the specified cell.
	GetEditStyle (see page 14828)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.

	GetEditText (see page 14985)	Generates an OnGetEditText event.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetGridHeight (see page 14829)	Returns the value of the GridHeight (see page 14850) property.
	GetGridWidth (see page 14830)	Returns the value of the GridWidth (see page 14851) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HideEdit (see page 14830)	Hides the inplace editor.
	HideEditor (see page 14830)	Hides the inplace editor when it is not in use.
	InvalidateCell (see page 14831)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCol (see page 14831)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	InvalidateEditor (see page 14832)	Causes the inplace editor to repaint itself, moving to a new location if necessary.
	InvalidateGrid (see page 14832)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 14832)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyDown (see page 14833)	Provides special processing when the user presses a key.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyPress (see page 14833)	Provides special processing when the user presses a key.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseDown (see page 14834)	Provides special processing when the user presses a mouse button.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseMove (see page 14834)	Provides special processing when the user moves the mouse.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	MouseUp (see page 14835)	Provides special processing when the user releases the mouse button.
	MoveColRow (see page 14835)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 14836)	Moves a column to a new position.
	MoveRow (see page 14836)	Moves a row to a new position.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.

	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 5920)	Renders the image of a custom control.
	Paint (see page 14836)	Draws the image of the grid control on the screen.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	RowHeightsChanged (see page 14837)	Responds when the row heights change.
	RowMoved (see page 14985)	Generates an OnRowMoved event.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	ScrollData (see page 14838)	Scrolls the visual image of the cells in the grid.
	SelectCell (see page 14784)	Determines whether a particular cell in the grid can be selected.
	SelectCell (see page 14838)	Determines whether a particular cell in the grid can be selected.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEditText (see page 14985)	Generates an OnSetEditText event.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.

	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	ShowEditor (see page 14839)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 14839)	Displays the inplace editor, with its text set to the specified character.
	SizeChanged (see page 14840)	Responds to changes in the size of the grid.
	Sizing (see page 14840)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	TimedScroll (see page 14840)	Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.
	TopLeftChanged (see page 14784)	Generates an OnTopLeftChanged (see page 14800) event.
	TopLeftChanged (see page 14841)	Updates the position of the inplace editor when the grid scrolls.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateDesigner (see page 14841)	Notifies the designer when rows and columns of the grid are moved or resized.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 14843)	Determines whether a single line border is drawn around the grid.

	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Col (see page 14843)	Specifies the index of the column that contains the selected cell.
	ColCount (see page 14844)	Specifies the number of columns in the grid.
	Color (see page 5812)	Specifies the background color of the control.
	Color (see page 14844)	Specifies the background color of the control.
	ColWidths (see page 14845)	Indicates the width (in pixels) of all the columns in the grid.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColWidth (see page 14846)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 14847)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 14847)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EditorMode (see page 14848)	Determines whether the current cell can be edited.
	FixedColor (see page 14848)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 14848)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 14849)	Specifies the number of rows on the top of the grid that cannot be scrolled.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
R	GridHeight (see page 14850)	Specifies the height of the grid in pixels.
	GridLineWidth (see page 14850)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
R	GridWidth (see page 14851)	Specifies the width of the grid in pixels.
R	HitTest (see page 14851)	Indicates the current position of the mouse cursor.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
R	InplaceEditor (see page 14852)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	LeftCol (see page 14852)	Specifies the index of the first visible scrollable column in the grid.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	Options (see page 14853)	Specifies various display and behavioral properties of the grid.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 14853)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	Row (see page 14854)	Specifies the index of the row that contains the selected cell.
	RowCount (see page 14854)	Specifies the number of rows in the grid.
	RowHeights (see page 14855)	Gives the height (in pixels) of all rows in the grid.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ScrollBars (see page 14856)	Specifies whether the grid includes horizontal and vertical scroll bars.
	Selection (see page 14856)	Indicates the boundaries of the current selection.
	TabStops (see page 14858)	Indicates whether the user can tab to specified columns in the grid.
	Text (see page 5842)	Contains a text string associated with the control.

	TopRow (see page 14858)	Specifies the index of the first visible scrollable row in the grid.
R	VisibleColCount (see page 14859)	Indicates the number of scrollable columns visible in the grid.
R	VisibleRowCount (see page 14859)	Indicates the number of scrollable rows visible in the grid.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	CellRect (see page 14781)	Returns the screen coordinates of a cell in the grid.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 14983)	Creates an instance of TStringGrid (see page 14968).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 14983)	Destroys an instance of TStringGrid (see page 14968).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.

	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseCoord (see page 14834)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.
	MouseToCell (see page 14783)	Returns the column and row of the cell at the position with screen coordinates (X,Y).

	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	Canvas (see page 14785)	Specifies the TCanvas object that presents a drawing surface for the control.
	Cells (see page 14986)	Lists the strings for each cell in the grid.

	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Col (see page 14787)	Specifies the index of the column that contains the selected cell.
	Cols (see page 14987)	Lists strings with their associated objects for each column.
	ColWidths (see page 14788)	Indicates the width (in pixels) of all the columns in the grid.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EditorMode (see page 14789)	Determines whether the current cell can be edited.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GridHeight (see page 14789)	Specifies the height of the grid in pixels.
	GridWidth (see page 14789)	Specifies the width of the grid in pixels.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LeftCol (see page 14790)	Specifies the index of the first visible scrollable column in the grid.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	Objects (see page 14987)	Lists the objects for each cell in the grid.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	Row (see page 14790)	Specifies the index of the row that contains the selected cell.
	RowHeights (see page 14791)	Gives the height (in pixels) of all rows in the grid.

	Rows (see page 14987)	Lists the strings and their associated objects for each row.
	Selection (see page 14791)	Indicates the boundaries of the current selection.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TabStops (see page 14793)	Indicates whether the user can tab to specified columns in the grid.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TopRow (see page 14793)	Specifies the index of the first visible scrollable row in the grid.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 14906)	Occurs when the user clicks the control.
	OnColumnMoved (see page 14908)	Occurs immediately after the position of a column changes.
	OnContextPopup (see page 14908)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 14909)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 14909)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 14911)	Occurs when the user drags an object over a control.
	OnDrawCell (see page 14914)	Occurs when a cell in the grid needs to be drawn.
	OnEndDock (see page 14914)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 14915)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 14916)	Occurs when a control receives the input focus.
	OnExit (see page 14918)	Occurs when the input focus shifts away from one control to another.
	OnGetEditMask (see page 14920)	Occurs when the in-place editor requests an edit mask.
	OnGetEditText (see page 14920)	Occurs when the in-place editor requests the value of a cell.
	OnKeyDown (see page 14920)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 14923)	Occurs when a key is pressed.
	OnKeyUp (see page 14924)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 14926)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 14927)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 14929)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 14929)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 14930)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 14932)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheelDown (see page 14934)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 14935)	Occurs when the mouse wheel is rotated upward.
	OnRowMoved (see page 14935)	Occurs immediately after the position of a row changes.
	OnSelectCell (see page 14935)	Occurs before a cell in the grid is selected.
	OnSetEditText (see page 14936)	Occurs when the user edits the value of a cell.
	OnStartDock (see page 14936)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 14937)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

	OnTopLeftChanged (see page 14937)	Occurs immediately after the TopRow (see page 14793) property or the LeftCol (see page 14790) property changes.

Published Properties

	Name	Description
	Align (see page 14874)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 14878)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 14879)	Specifies which edges of the control are beveled.
	BevelInner (see page 14879)	Specifies the cut of the inner bevel.
	BevelKind (see page 14879)	Specifies the control's bevel style.
	BevelOuter (see page 14880)	Specifies the cut of the outer bevel.
	BevelWidth (see page 14880)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 14881)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 14881)	Determines whether a single line border is drawn around the grid.
	ColCount (see page 14881)	Specifies the number of columns in the grid.
	Color (see page 14882)	Specifies the background color of the control.
	Constraints (see page 14883)	Specifies the size constraints for the control.
	Ctl3D (see page 14884)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DefaultColWidth (see page 14884)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 14885)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 14885)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DragCursor (see page 14886)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 14886)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 14887)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 14888)	Controls whether the control responds to mouse, keyboard, and timer events.
	FixedColor (see page 14889)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 14889)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 14889)	Specifies the number of rows on the top of the grid that cannot be scrolled.
	Font (see page 14890)	Controls the attributes of text written on or in the control.
	GridLineWidth (see page 14891)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 14892)	Specifies various display and behavioral properties of the grid.
	ParentBiDiMode (see page 14893)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 14893)	Determines where a control looks for its color information.
	ParentCtl3D (see page 14893)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 14894)	Determines where a control looks for its font information.
	ParentShowHint (see page 14894)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 14895)	Identifies the pop-up menu associated with the control.
	RowCount (see page 14899)	Specifies the number of rows in the grid.
	ScrollBars (see page 14900)	Specifies whether the grid includes horizontal and vertical scroll bars.

	ShowHint (see page 14901)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 14902)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 14857)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 14903)	Determines whether the component appears onscreen.
	VisibleColCount (see page 14904)	Indicates the number of scrollable columns visible in the grid.
	VisibleRowCount (see page 14905)	Indicates the number of scrollable rows visible in the grid.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

75.1.7.2 TStringGrid.Create Constructor

Creates an instance of TStringGrid (see page 14968).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TStringGrid(TComponent * AOwner);
```

Description

Call Create to create an instance of TStringGrid (see page 14968) at runtime. For string grids placed on forms at design time, Create is called automatically.

AOwner is another component, typically a form, that becomes the grid's Owner. The Owner is responsible for freeing the grid.

See Also

Create (see page 14813)

75.1.7.3 TStringGrid.Destroy Destructor

Destroys an instance of TStringGrid (see page 14968).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TStringGrid();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the string grid is not nil, and only then calls Destroy.

Destroy frees the helper objects used to manage the array of strings and their associated objects.

See Also

Free (see page 22300)

75.1.7.4 TStringGrid Methods

The methods of the TStringGrid class are listed here.

Protected Methods

	Name	Description
	ColumnMoved (see page 14984)	Generates an OnColumnMoved event.
	DrawCell (see page 14984)	Generates an OnDrawCell event.
	GetEditText (see page 14985)	Generates an OnGetEditText event.
	RowMoved (see page 14985)	Generates an OnRowMoved event.
	SetEditText (see page 14985)	Generates an OnSetEditText event.

75.1.7.4.1 TStringGrid.ColumnMoved Method

Generates an OnColumnMoved event.

Pascal

```
procedure ColumnMoved(FromIndex: Longint; ToIndex: Longint); override;
```

C++

```
virtual __fastcall ColumnMoved(Longint FromIndex, Longint ToIndex);
```

Description

ColumnMoved is called immediately after a column in the grid has moved. It overrides the inherited method to generate an OnColumnMoved event. Descendants of TCustomDrawGrid can override this method to make internal adjustments or to block the OnColumnMoved event.

See Also

OnColumnMoved (see page 14794)

75.1.7.4.2 TStringGrid.DrawCell Method

Generates an OnDrawCell event.

Pascal

```
procedure DrawCell(ACol: Longint; ARow: Longint; ARect: TRect; AState: TGridDrawState);
override;
```

C++

```
virtual __fastcall DrawCell(Longint ACol, Longint ARow, TRect ARect, TGridDrawState AState);
```

Description

DrawCell is called automatically whenever a cell in the grid is drawn. If the DefaultDrawing property is true, the background of the cell is painted before DrawCell is called, and the 3D effects of the fixed cells or the focus rectangle around the focused cell are drawn after DrawCell. Descendants of TCustomDrawGrid can override this method to draw the contents of the cell or to block the OnDrawCell event.

See Also

DefaultDrawing (see page 14847)

FixedCols (see page 14848)

FixedRows (see page 14849)

OnDrawCell (see page 14795)

75.1.7.4.3 TStringGrid.GetEditText Method

Generates an OnGetEditText event.

Pascal

```
function GetEditText(ACol: Longint; ARow: Longint): string; override;
```

C++

```
virtual __fastcall AnsiString GetEditText(Longint ACol, Longint ARow);
```

Description

The inplace editor calls GetEditText to obtain the text for the indicated cell. GetEditText returns the Value parameter set by the OnGetEditText event handler. If there is no OnGetEditText event handler, GetEditText returns an empty string.

See Also

OnGetEditText (see page 14797)

75.1.7.4.4 TStringGrid.RowMoved Method

Generates an OnRowMoved event.

Pascal

```
procedure RowMoved(FromIndex: Longint; ToIndex: Longint); override;
```

C++

```
virtual __fastcall RowMoved(Longint FromIndex, Longint ToIndex);
```

Description

RowMoved is called immediately after a row in the grid changes position. It overrides the inherited method to generate an OnRowMoved event. Descendants of TCustomDrawGrid can override this method to make internal adjustments or to block the OnRowMoved event.

See Also

OnRowMoved (see page 14798)

75.1.7.4.5 TStringGrid.SetEditText Method

Generates an OnSetEditText event.

Pascal

```
procedure SetEditText(ACol: Longint; ARow: Longint; const Value: string); override;
```

C++

```
virtual __fastcall SetEditText(Longint ACol, Longint ARow, const AnsiString Value);
```

Description

SetEditText is called by the inplace editor so that the grid can store the new value for the current cell. SetEditText overrides the inherited parent class's method to generate an OnSetEditText event. Descendants of TCustomDrawGrid can override this method to update an internal representation of the cell's value, or to block the OnSetEditText event.

See Also

OnSetEditText (see page 14800)

75.1.7.5 TStringGrid Properties

The properties of the TStringGrid class are listed here.

Public Properties

	Name	Description
	Cells (see page 14986)	Lists the strings for each cell in the grid.
	Cols (see page 14987)	Lists strings with their associated objects for each column.
	Objects (see page 14987)	Lists the objects for each cell in the grid.
	Rows (see page 14987)	Lists the strings and their associated objects for each row.

75.1.7.5.1 TStringGrid.Cells Property

Lists the strings for each cell in the grid.

Pascal

```
property Cells [ACol, ARow: Integer]: string;
```

C++

```
__property AnsiString Cells[int ACol, ARow];
```

Description

Use Cells to access the string within a particular cell. ACol is the column coordinate of the cell, and ARow is the row coordinate of the cell. The first row is row zero, and the first column is column zero.

The ColCount (see page 14881) and RowCount (see page 14899) property values define the size of the array of strings.

To access the objects associated with the strings in the Cells array, use the Objects (see page 14987) property.

Delphi Examples:

```
{
This code fills each cell with a number. The numbers are
arranged in consecutive order by rows.  Notice that even
cells in the fixed rows and columns are altered.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K : Integer;
begin
  K := 0;
  with StringGrid1 do
    for I := 0 to ColCount - 1 do
      for J:= 0 to RowCount - 1 do
        begin
          K := K + 1;
          Cells[I,J] := IntToStr(K);
        end;
end;
```

See Also

ColCount (see page 14844)

Objects (see page 14987)

RowCount (see page 14854)

75.1.7.5.2 TStringGrid.Cols Property

Lists strings with their associated objects for each column.

Pascal

```
property Cols [Index: Integer]: TStrings;
```

C++

```
__property TStrings Cols[int Index];
```

Description

Use Cols to access all the strings for a single column, along with their associated objects. The number of strings and associated objects is always equal to the value of the ColCount (see page 14881) property, the number of columns in the grid. The Index parameter is the number of the column, where the value of the first column in the grid is zero.

To get all the strings and objects for a row, rather than for a column, use the Rows (see page 14987) property.

See Also

ColCount (see page 14844)

Objects (see page 14987)

Rows (see page 14987)

75.1.7.5.3 TStringGrid.Objects Property

Lists the objects for each cell in the grid.

Pascal

```
property Objects [ACol, ARow: Integer]: TObject;
```

C++

```
__property TObject * Objects[int ACol, ARow];
```

Description

Use the Objects property to access the object associated with a particular cell. The ColCount (see page 14881) and RowCount (see page 14899) values define the size of the array of objects. ACol is the column coordinate of the cell, and ARow is the row coordinate of the cell, where the first column is column 0, and the first row is row 0.

Note: The string grid does not own the objects in the Objects array. Objects added to the Objects array still exist even if the string grid is destroyed. They must be explicitly destroyed by the application.

See Also

Cells (see page 14986)

ColCount (see page 14844)

RowCount (see page 14854)

75.1.7.5.4 TStringGrid.Rows Property

Lists the strings and their associated objects for each row.

Pascal

```
property Rows [Index: Integer]: TStrings;
```

C++

```
__property TStrings Rows[int Index];
```

Description

Use Rows to access all the strings for a single row, along with their associated objects. The number of strings and associated objects is always equal to the value of the RowCount (see page 14899) property, the number of rows in the grid. The Index parameter is the number of the row, where the value of the first row in the grid is zero.

To get all the strings and objects for a column, rather than for a row, use the Cols (see page 14987) property.

Delphi Examples:

```
{
This example displays a string grid, a list box, and a
button on a form. When the application runs, strings are put
in the cells of row 1 of the string grid. When the user
clicks the button, The strings in the current row of the
grid are displayed in the list box.
Note that when the grid contains an empty cell, the list box
has a corresponding empty string.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.Items := StringGrid1.Rows[StringGrid1.Row];
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
    Cells[1,1] := 'Object';
    Cells[2,1] := 'Pascal';
    Cells[3,1] := 'is';
    Cells[4,1] := 'excellent';
    Cells[0,2] := 'Row 2';
    Cells[1,2] := 'Delphi';
    Cells[2,2] := 'is';
    Cells[4,2] := 'RAD';
  end;
end;
```

See Also

ColCount (see page 14844)

Cols (see page 14987)

Objects (see page 14987)

75.1.8 TStringGridStrings Class

TStringGridStrings maintains a list of the strings that appear in the cells of a string grid.

Class Hierarchy

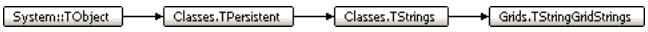

Pascal

```
TStringGridStrings = class(TStrings);
```

C++

```
class TStringGridStrings : public TStrings;
```

File

Grids

Description

Do not use TStringGridStrings directly in an application. It is tailored to work with a TStringGrid (see page 14968) object, and is created by the string grid to represent the strings (and associated objects) in a row or column of the grid.

See Also

TStringGrid (see page 14968)

TStringList (see page 3006)

75.1.8.1 TStringGridStrings Members

The following tables list the members exposed by TStringGridStrings.

Protected Methods

	Name	Description
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	CompareStrings (see page 3048)	Compares two strings.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 3049)	Reads and writes the Strings (see page 3080) property as if it were published.
	Error (see page 3052)	This is the overview for the Error method overload.
	ExtractName (see page 3053)	Returns the name portion of a string that is a name value pair.
V	Get (see page 14994)	Returns a string given its index.
V	GetCapacity (see page 3054)	Returns the currently allocated size of the list of strings.
V	GetCount (see page 14994)	Returns the number of strings in the list
V	GetObject (see page 14994)	Returns the object associated with the string at a specified index.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetTextStr (see page 3056)	Returns the value of the Text (see page 3084) property.
V	Put (see page 14995)	Changes the value of the string with a specified index.
V	PutObject (see page 14996)	Changes the object associated with the string at a specified index.
V	SetCapacity (see page 3068)	Changes the amount of memory allocated to hold strings in the list.
V	SetTextStr (see page 3069)	Sets the Text (see page 3084) property.
V	SetUpdateState (see page 14996)	Performs internal adjustments before or after a series of updates.

Protected Properties

	Name	Description
R	UpdateCount (see page 3085)	Indicates the number of calls to BeginUpdate (see page 3045) that have not been matched by a call to EndUpdate (see page 3050).

Public Methods

	Name	Description
V	Add (see page 14992)	Replaces the first empty string with a specified string.
V	AddObject (see page 3036)	Adds a string to the list, and associates an object with the string.
V	AddStrings (see page 3039)	Adds a group of strings to the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	Append (see page 3041)	Adds a string to the list.
V	Assign (see page 14992)	Copies the list of strings and objects from another TStrings object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 3045)	Enables the TStrings (see page 3029) object to track when the list of strings is changing.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 14993)	Replaces all strings in the list with empty strings.
	Create (see page 14991)	Creates an instance of TStringGridStrings (see page 14988).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Delete (see page 14993)	Prevents strings from being deleted.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 3032)	Destroys the TPersistent instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 3050)	Enables the TStrings (see page 3029) object to keep track of when the list of strings has finished changing.
	Equals (see page 3051)	Compares the list of strings to the list from another TStrings (see page 3029) object and returns true if the two lists match.
V	Exchange (see page 3053)	Swaps the position of two strings in the list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 3054)	Returns a TStrings (see page 3029) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetText (see page 3055)	Allocates a text buffer and fills it with the value of the Text (see page 3084) property.
V	IndexOf (see page 3056)	Returns the position of a string in the list.
V	IndexOfName (see page 3057)	Returns the position of the first name-value pair with the specified name.
V	IndexOfObject (see page 3059)	Returns the index of the first string in the list associated with a given object.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Insert (see page 14995)	Prevents strings from being inserted.
V	InsertObject (see page 3060)	Inserts a string into the list at the specified position, and associates it with an object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	LoadFromFile (see page 3061)	Fills the list with the lines of text in a specified file.
V	LoadFromStream (see page 3062)	Fills the list with lines of text read from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	Move (see page 3063)	Changes the position of a string in the list.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SaveToFile (see page 3065)	Saves the strings in the list to the specified file.

V	SaveToStream (see page 3067)	Writes the value of the Text (see page 3084) property to a stream object.
V	SetText (see page 3068)	Sets the Text (see page 3084) property.

Public Properties

	Name	Description
	Capacity (see page 3070)	Indicates the number of strings the TStrings (see page 3029) object can hold.
	CommaText (see page 3070)	Lists the strings in the TStrings (see page 3029) object in system data format (SDF).
R	Count (see page 3071)	Introduces an abstract property to represent the number of strings in the list.
	DelimitedText (see page 3074)	Represents all the strings in the TStrings (see page 3029) object as a single delimited string.
	Delimiter (see page 3074)	Specifies the delimiter used by the DelimitedText (see page 3074) property.
	LineBreak (see page 3075)	Defines the line break character(s).
R	Names (see page 3075)	Indicates the name part of strings that are name-value pairs.
	NameValueSeparator (see page 3077)	Indicates the character used to separate names from values.
	Objects (see page 3077)	Represents a set of objects that are associated one with each of the strings in the Strings (see page 3080) property.
	QuoteChar (see page 3080)	Specifies the quote character used by the DelimitedText (see page 3074) property.
	StrictDelimiter (see page 3080)	Determines how the Delimiter (see page 3074) property is used.
	Strings (see page 3080)	References the strings in the list by their positions.
	StringsAdapter (see page 3084)	Implements an IStringsAdapter (see page 3208) interface for the TStrings (see page 3029) object.
	Text (see page 3084)	Lists the strings in the TStrings (see page 3029) object as a single string with the individual strings delimited by carriage returns and line feeds.
	ValueFromIndex (see page 3085)	Represents the value part of a string with a given index, on strings that are name-value pairs.
	Values (see page 3086)	Represents the value part of a string associated with a given name, on strings that are name-value pairs.

75.1.8.2 TStringGridStrings.Create Constructor

Creates an instance of TStringGridStrings (see page 14988).

Pascal

```
constructor Create(AGrid: TStringGrid; AIndex: Longint);
```

C++

```
__fastcall TStringGridStrings(TStringGrid AGrid, Longint AIndex);
```

Description

Do not create instances of TStringGridStrings (see page 14988). String grids create TStringGridStrings (see page 14988) objects for every row and column, to allow quick access of the strings and objects associated with the cells in the grid.

The AGrid parameter specifies the string grid that will use the TStringGridStrings (see page 14988) object. The AIndex parameter specifies the index of the row or column associated with the TStringGridStrings (see page 14988) object. If AIndex is greater than 0, the TStringGridStrings (see page 14988) object represents the row indexed by AIndex. If AIndex is less than 0, the TStringGridStrings (see page 14988) object represents the column indexed by the absolute value of AIndex. If AIndex is 0, the TStringGridStrings (see page 14988) object represents an empty row or column.

See Also

TStringGrid (see page 14968)

75.1.8.3 TStringGridStrings Methods

The methods of the TStringGridStrings class are listed here.

Protected Methods

	Name	Description
	Get (see page 14994)	Returns a string given its index.
	GetCount (see page 14994)	Returns the number of strings in the list
	GetObject (see page 14994)	Returns the object associated with the string at a specified index.
	Put (see page 14995)	Changes the value of the string with a specified index.
	PutObject (see page 14996)	Changes the object associated with the string at a specified index.
	SetUpdateState (see page 14996)	Performs internal adjustments before or after a series of updates.

Public Methods

	Name	Description
	Add (see page 14992)	Replaces the first empty string with a specified string.
	Assign (see page 14992)	Copies the list of strings and objects from another TStrings object.
	Clear (see page 14993)	Replaces all strings in the list with empty strings.
	Delete (see page 14993)	Prevents strings from being deleted.
	Insert (see page 14995)	Prevents strings from being inserted.

75.1.8.3.1 TStringGridStrings.Add Method

Replaces the first empty string with a specified string.

Pascal

```
function Add(const S: string): Integer; override;
```

C++

```
virtual __fastcall int Add(const AnsiString S);
```

Description

Call Add to add a string to the list. Add searches the list for an empty string and replaces it with S. It then returns the index where it added S to the list. If there are no empty strings in the list, Add does not add S to the list and returns –1.

See Also

AddStrings (see page 3039)

Append (see page 3041)

Count

IndexOf (see page 3056)

Insert (see page 14995)

75.1.8.3.2 TStringGridStrings.Assign Method

Copies the list of strings and objects from another TStrings object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the strings and objects from a source object of type TStrings. If the source object has more strings than the number of cells in the row or column represented by the TStringGridStrings (see page 14988) object, Assign only copies the first Count strings and objects.

If the Source parameter is not a TStrings object, Assign calls the inherited method.

See Also

Count

Objects

Strings

75.1.8.3.3 TStringGridStrings.Clear Method

Replaces all strings in the list with empty strings.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call Clear to remove all strings from the list, including any references to associated objects. Clear replaces every string with an empty string and every object reference with nil (Delphi) or NULL (C++).

Note that Clear does not free the objects that are associated with strings in the list.

See Also

Delete (see page 14993)

Objects

Strings

75.1.8.3.4 TStringGridStrings.Delete Method

Prevents strings from being deleted.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Because TStringGridStrings (see page 14988) objects represent the rows or columns of a string grid, the number of entries in the list must correspond exactly to the number of cells in the corresponding row or column. Rather than allow entries to be deleted, destroying this correspondence, Delete raises an EInvalidGridOperation (see page 14767) exception.

See Also

EInvalidGridOperation (see page 14767)

Clear (see page 14993)

75.1.8.3.5 TStringGridStrings.Get Method

Returns a string given its index.

Pascal

```
function Get(Index: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString Get(int Index);
```

Description

Get is the protected read implementation of the Strings property.

In TStrings Get is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to return the string with the specified index.

See Also

Strings (see page 3080)

75.1.8.3.6 TStringGridStrings.GetCount Method

Returns the number of strings in the list

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCount is the protected read implementation of the Count property.

In TStrings GetCount is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to return the number of strings that have been added to the list.

See Also

Count (see page 3071)

75.1.8.3.7 TStringGridStrings.GetObject Method

Returns the object associated with the string at a specified index.

Pascal

```
function GetObject(Index: Integer): TObject; override;
```

C++

```
virtual __fastcall TObject * GetObject(int Index);
```

Description

GetObject is the protected read implementation of the Objects property.

Index is the index of the string with which the object is associated.

In TStrings, GetObject always returns nil (Delphi) or NULL (C++). This provides a default implementation for descendants that do not support associating objects with the strings in the list. Descendants that support this feature override GetObject to return the specified object.

See Also

Objects (see page 3077)

75.1.8.3.8 TStringGridStrings.Insert Method

Prevents strings from being inserted.

Pascal

```
procedure Insert(Index: Integer; const S: string); override;
```

C++

```
virtual __fastcall Insert(int Index, const AnsiString S);
```

Description

Because TStringGridStrings (see page 14988) objects represent the rows or columns of a string grid, the number of entries in the list must correspond exactly to the number of cells in the corresponding row or column. Rather than allow entries to be inserted, destroying this correspondence, Insert raises an EInvalidGridOperation (see page 14767) exception.

See Also

EInvalidGridOperation (see page 14767)

Add (see page 14992)

75.1.8.3.9 TStringGridStrings.Put Method

Changes the value of the string with a specified index.

Pascal

```
procedure Put(Index: Integer; const S: string); override;
```

C++

```
virtual __fastcall Put(int Index, const AnsiString S);
```

Description

Put is the protected write implementation of the Strings property.

Put changes the value of the string with the index specified by Index to S. Put does not change the object at the specified position. That is, any object associated with the previous string becomes associated with the new string.

See Also

Strings (see page 3080)

Objects (see page 3077)

75.1.8.3.10 TStringGridStrings.PutObject Method

Changes the object associated with the string at a specified index.

Pascal

```
procedure PutObject(Index: Integer; AObject: TObject); override;
```

C++

```
virtual __fastcall PutObject(int Index, TObject * AObject);
```

Description

PutObject is the protected write implementation of the Objects property.

As implemented in TStrings, PutObject does nothing. This provides a default implementation for descendants that do not support associating objects with the strings in the list. Descendants that support this feature override PutObject to change the specified object.

See Also

Objects (see page 3077)

75.1.8.3.11 TStringGridStrings.SetUpdateState Method

Performs internal adjustments before or after a series of updates.

Pascal

```
procedure SetUpdateState(Updating: Boolean); override;
```

C++

```
virtual __fastcall SetUpdateState(Boolean Updating);
```

Description

SetUpdateState is called at the beginning or end of a series of updates. When the BeginUpdate method is first called and the TStrings object is not already in the middle of an update, TStrings calls SetUpdateState internally, with Updating set to true. When the EndUpdate method is called and it cancels out the last unmatched call to BeginUpdate, TStrings calls SetUpdateState internally, with Updating set to false.

As implemented in TStrings, SetUpdateState does nothing. Descendant classes can override this method to optimize the response to updates.

See Also

BeginUpdate (see page 3045)

EndUpdate (see page 3050)

75.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TEditStyle (see page 14997)	TEditStyle indicates whether a cell in a grid contains a drop-down arrow or ellipsis button.
	TGridOption (see page 14999)	TGridOption and TGridOptions (see page 15005) customize the appearance and behavior of a grid control.
	TGridState (see page 15001)	TGridState indicates any user actions that are currently changing the layout of the grid.

Records

	Name	Description
	TGridAxisDrawInfo (see page 14998)	TGridAxisDrawInfo and TGridDrawInfo (see page 14999) represent information that is useful when rendering a grid.
	TGridCoord (see page 14998)	TGridCoord specifies a cell in a grid control.
	TGridDrawInfo (see page 14999)	This is record Grids.TGridDrawInfo.
	TGridRect (see page 15001)	TGridRect specifies a set of cells that occupy a rectangular region in a grid control.

75.2.1 Grids.TEditStyle Enumeration

TEditStyle indicates whether a cell in a grid contains a drop-down arrow or ellipsis button.

Pascal

```
TEditStyle = (
  esSimple,
  esEllipsis,
  esPickList
);
```

C++

```
enum TEditStyle {
  esSimple,
  esEllipsis,
  esPickList
};
```

File

Grids

Description

TEditStyle describes the buttons, if any, that can appear in the cell of a grid. It can be any of the following values:

Value	Meaning
esSimple	The cell acts as a simple edit control where users can type a new value.
esEllipsis	The cell contains an ellipsis button that users can click to launch a dialog.
esPickList	The cell contains a drop-down arrow button that users can click to display a pick list of possible values.

75.2.2 Grids.TGridAxisDrawInfo Record

TGridAxisDrawInfo and TGridDrawInfo (see page 14999) represent information that is useful when rendering a grid.

Pascal

```
TGridAxisDrawInfo = record
  EffectiveLineWidth: Integer;
  FixedBoundary: Integer;
  GridBoundary: Integer;
  GridExtent: Integer;
  LastFullVisibleCell: Longint;
  FullVisBoundary: Integer;
  FixedCellCount: Integer;
  FirstGridCell: Integer;
  GridCellCount: Integer;
  GetExtent: TGetExtentsFunc;
end;
```

C++

```
struct TGridAxisDrawInfo {
  int EffectiveLineWidth;
  int FixedBoundary;
  int GridBoundary;
  int GridExtent;
  Longint LastFullVisibleCell;
  int FullVisBoundary;
  int FixedCellCount;
  int FirstGridCell;
  int GridCellCount;
  TGetExtentsFunc GetExtent;
};
```

File

Grids

Description

TGridDrawInfo (see page 14999) encapsulates information that reflects the layout of the grid axes. Each axis is described by a TGridAxisDrawInfo value, which encodes information about the size of the axis and the type and number of cells along that axis.

75.2.3 Grids.TGridCoord Record

TGridCoord specifies a cell in a grid control.

Pascal

```
TGridCoord = record
  X: Longint;
  Y: Longint;
end;
```

C++

```
struct TGridCoord {
  Longint X;
  Longint Y;
};
```

File

Grids

Description

TGridCoord indicates a specific cell in a grid.

X indicates the column of the grid, where 0 specifies the first column, 1 specifies the second column, and so on.

Y indicates the row of the grid, where 0 specifies the first row, 1 specifies the second row, and so on.

75.2.4 Grids.TGridDrawInfo Record

Pascal

```
TGridDrawInfo = record
  Horz: TGridAxisDrawInfo;
  Vert: TGridAxisDrawInfo;
end;
```

C++

```
struct TGridDrawInfo {
  TGridAxisDrawInfo Horz;
  TGridAxisDrawInfo Vert;
};
```

File

Grids

Description

This is record Grids.TGridDrawInfo.

75.2.5 Grids.TGridOption Enumeration

TGridOption and TGridOptions (see page 15005) customize the appearance and behavior of a grid control.

Pascal

```
TGridOption = (
  goFixedVertLine,
  goFixedHorzLine,
  goVertLine,
  goHorzLine,
  goRangeSelect,
  goDrawFocusSelected,
  goRowSizing,
  goColSizing,
  goRowMoving,
  goColMoving,
  goEditing,
  goTabs,
  goRowSelect,
  goAlwaysShowEditor,
  goThumbTracking
);
```

C++

```
enum TGridOption {
  goFixedVertLine,
  goFixedHorzLine,
  goVertLine,
  goHorzLine,
  goRangeSelect,
  goDrawFocusSelected,
  goRowSizing,
  goColSizing,
  goRowMoving,
  goColMoving,
  goEditing,
  goTabs,
  goRowSelect,
  goAlwaysShowEditor,
  goThumbTracking
};
```

File

Grids

Description

TGridOptions (see page 15005) is a set of TGridOption values. The following table lists the possible TGridOption values:

Value	Meaning
goFixedVertLine	Vertical lines are drawn to separate the fixed (nonscrolling) columns in the grid.
goFixedHorzLine	Horizontal lines are drawn to separate the fixed (nonscrolling) rows in the grid.
goVertLine	Vertical lines are drawn to separate the scrollable columns in the grid.
goHorzLine	Horizontal lines are drawn to separate the scrollable rows in the grid.
goRangeSelect	Users can select ranges of cells at one time. goRangeSelect is ignored if Options includes goEditing.
goDrawFocusSelected	Cells with input focus are drawn with a special highlight color, just like selected cells without input focus. If goDrawFocusSelected is not included, the cell with input focus is distinguished by a focus rectangle, not by a special background color.
goRowSizing	Scrollable rows can be individually resized.
goColSizing	Scrollable columns can be individually resized.
goRowMoving	Scrollable rows can be moved using the mouse.
goColMoving	Scrollable columns can be moved using the mouse.
goEditing	Users can edit the contents of cells. When goEditing is included in Options, goRangeSelect has no effect.
goTabs	Users can navigate through the cells in the grid using Tab and Shift+Tab.
goRowSelect	Entire rows are selected rather than individual cells. If goRowSelect is included in Options, goAlwaysShowEditor has no effect.
goAlwaysShowEditor	The grid is locked into edit mode. The user does not need to use Enter or F2 to turn on EditorMode. If Options does not include goEditing, goAlwaysShowEditor has no effect. If Options includes goRowSelect, goAlwaysShowEditor has no effect.
goThumbTracking	The grid image updates while the user is dragging the thumb of the scroll bar. If goThumbTracking is not included, the image does not update until the user releases the thumb in a new position.

75.2.6 Grids.TGridRect Record

TGridRect specifies a set of cells that occupy a rectangular region in a grid control.

Pascal

```
TGridRect = record
  case Integer of
    0: (Bottom: Longint;
        Left: Longint;
        Right: Longint;
        Top: Longint;);
    1: (BottomRight: TGridCoord;
        TopLeft: TGridCoord;);
end;
```

C++

```
struct TGridRect {
  Longint Bottom;
  Longint Left;
  Longint Right;
  Longint Top;
  TGridCoord BottomRight;
  TGridCoord TopLeft;
};
```

File

Grids

Description

TGridRect indicates a rectangular set of cells from the grid in one of two ways:

Using the first method, TopLeft indicates the coordinates of the top-left cell in the set and BottomRight indicates the coordinates of the bottom-right cell.

Using the second method, Left indicates the first column in the set and Right the last column in the set, where 0 is the first column of the grid, 1 is the second column, and so on. Top indicates the first row in the set and Bottom the last row, where 0 is the first row of the grid, 1 the second column, and so on.

75.2.7 Grids.TGridState Enumeration

TGridState indicates any user actions that are currently changing the layout of the grid.

Pascal

```
TGridState = (
  gsNormal,
  gsSelecting,
  gsRowSizing,
  gsColSizing,
  gsRowMoving,
  gsColMoving
);
```

C++

```
enum TGridState {
```

```
    gsNormal,
    gsSelecting,
    gsRowSizing,
    gsColSizing,
    gsRowMoving,
    gsColMoving
  };
```

File

Grids

Description

TGridState represents state information about how the grid layout is changing. The following table lists the possible values:

Value	Meaning
gsNormal	The grid layout is not changing.
gsSelecting	The user is selecting a cell or row.
gsRowSizing	The user is changing the height of a row.
gsColSizing	The user is changing the width of a column.
gsRowMoving	The user is dragging a row to a new position.
gsColMoving	The user is dragging a column to a new position.

75.3 Types

The following table lists types in this documentation.

Types

Name	Description
TDrawCellEvent (see page 15003)	Function type for OnDrawCell event.
TGetEditEvent (see page 15004)	TGetEditEvent is the type of the event handlers that provide strings for the inplace editor of a grid.
TGetExtentsFunc (see page 15004)	This is type Grids.TGetExtentsFunc.
TGridDrawState (see page 15004)	TGridDrawState represents information that is useful when rendering the cell of a grid.
TGridMovement (see page 15005)	This is type Grids.TGridMovement.
TGridOptions (see page 15005)	TGridOptions and TGridOption (see page 14999) customize the appearance and behavior of a grid control.
TGridScrollDirection (see page 15006)	TGridScrollDirection indicates the direction in which to scroll the cells of a grid control.
TMovedEvent (see page 15007)	This is type Grids.TMovedEvent.
TOnGetPickListItems (see page 15007)	This is type Grids.TOnGetPickListItems.
TSelectCellEvent (see page 15007)	This is type Grids.TSelectCellEvent.
TSetEditEvent (see page 15008)	TSetEditEvent is the type of the event handlers that assign strings to a cell in a grid.

75.3.1 Grids.TDrawCellEvent Type

Function type for OnDrawCell event.

Pascal

```
TDrawCellEvent = procedure (Sender: TObject; ACol, ARow: Longint; Rect: TRect; State:
TGridDrawState) of object;
```

C++

```
(Sender: TObject; ACol, ARow: Longint; Rect: TRect; State: TGridDrawState) ( TDrawCellEvent)();
```

File

Grids

Description

TDrawCellEvent is the function type for a OnDrawCell event. Its parameters are described in this table.

Parameter	Meaning
Sender	Grid in which cell is being drawn.
ACol, ARow	Column and row indexes of the cell being drawn.
Rect	Location of cell on canvas.
State	TGridDrawState (see page 15004) that indicates whether the cell has input focus, whether the cell is selected, and whether the cell is a fixed (nonscrolling) cell.

See Also

OnDrawCell (see page 14795)

TGridDrawState (see page 15004)

75.3.2 Grids.TGetEditEvent Type

TGetEditEvent is the type of the event handlers that provide strings for the inplace editor of a grid.

Pascal

```
TGetEditEvent = procedure (Sender: TObject; ACol, ARow: Longint; var Value: string) of object;
```

C++

```
(Sender: TObject; ACol, ARow: Longint; var Value: string) ( TGetEditEvent)();
```

File

Grids

Description

Sender is the grid whose cell is being edited.

ACol and ARow indicate the column and row indexes of the cell that is being edited.

Value is the string requested by the inplace editor for that cell. The event handler supplies this string.

75.3.3 Grids.TGetExtentsFunc Type

Pascal

```
TGetExtentsFunc = function (Index: Longint): Integer of object;
```

C++

```
(Index: Longint): Integer ( TGetExtentsFunc)();
```

File

Grids

Description

This is type Grids.TGetExtentsFunc.

75.3.4 Grids.TGridDrawState Type

TGridDrawState represents information that is useful when rendering the cell of a grid.

Pascal

```
TGridDrawState = set of (gdSelected, gdFocused, gdFixed);
```

C++

```
(gdSelected, gdFocused, gdFixed) TGridDrawState;
```

File

Grids

Description

TGridDrawState represents state information about the cell of a grid that is being rendered. It is a set that includes 0 or more of the following:

Value	Meaning
gdSelected	The cell is currently selected.
gdFocused	The cell has input focus.
gdFixed	The cell is in the fixed region of the grid.

75.3.5 Grids.TGridMovement Type

Pascal

```
TGridMovement = gsRowMoving..gsColMoving;
```

C++

```
gsRowMoving..gsColMoving TGridMovement;
```

File

Grids

Description

This is type Grids.TGridMovement.

75.3.6 Grids.TGridOptions Type

TGridOptions and TGridOption (see page 14999) customize the appearance and behavior of a grid control.

Pascal

```
TGridOptions = set of TGridOption;
```

C++

```
TGridOption TGridOptions;
```

File

Grids

Description

TGridOptions is a set of TGridOption (see page 14999) values. The following table lists the possible TGridOption (see page 14999) values:

Value	Meaning
goFixedVertLine	Vertical lines are drawn to separate the fixed (nonscrolling) columns in the grid.
goFixedHorzLine	Horizontal lines are drawn to separate the fixed (nonscrolling) rows in the grid.
goVertLine	Vertical lines are drawn to separate the scrollable columns in the grid.
goHorzLine	Horizontal lines are drawn to separate the scrollable rows in the grid.
goRangeSelect	Users can select ranges of cells at one time. goRangeSelect is ignored if Options includes goEditing.
goDrawFocusSelected	Cells with input focus are drawn with a special highlight color, just like selected cells without input focus. If goDrawFocusSelected is not included, the cell with input focus is distinguished by a focus rectangle, not by a special background color.
goRowSizing	Scrollable rows can be individually resized.
goColSizing	Scrollable columns can be individually resized.
goRowMoving	Scrollable rows can be moved using the mouse.
goColMoving	Scrollable columns can be moved using the mouse.
goEditing	Users can edit the contents of cells. When goEditing is included in Options, goRangeSelect has no effect.
goTabs	Users can navigate through the cells in the grid using Tab and Shift+Tab.
goRowSelect	Entire rows are selected rather than individual cells. If goRowSelect is included in Options, goAlwaysShowEditor has no effect.
goAlwaysShowEditor	The grid is locked into edit mode. The user does not need to use Enter or F2 to turn on EditorMode. If Options does not include goEditing, goAlwaysShowEditor has no effect. If Options includes goRowSelect, goAlwaysShowEditor has no effect.
goThumbTracking	The grid image updates while the user is dragging the thumb of the scroll bar. If goThumbTracking is not included, the image does not update until the user releases the thumb in a new position.

75.3.7 Grids.TGridScrollDirection Type

TGridScrollDirection indicates the direction in which to scroll the cells of a grid control.

Pascal

```
TGridScrollDirection = set of (sdLeft, sdRight, sdUp, sdDown);
```

C++

```
(sdLeft, sdRight, sdUp, sdDown) TGridScrollDirection;
```

File

Grids

Description

TGridScrollDirection is a set of one or two values that indicate the direction in which to scroll a grid. The set contains at least one value and at most one horizontal and one vertical value. The following table lists the possible values:

Value	Meaning
sdLeft	Scroll cells horizontally to the left.
sdRight	Scroll cells horizontally to the right.

sdUp	Scroll cells vertically up.
sdDown	Scroll cells vertically down.

See Also

TCustomGrid

75.3.8 Grids.TMovedEvent Type

Pascal

```
TMovedEvent = procedure (Sender: TObject; FromIndex, ToIndex: Longint) of object;
```

C++

```
(Sender: TObject; FromIndex, ToIndex: Longint) ( TMovedEvent)();
```

File

Grids

Description

This is type Grids.TMovedEvent.

75.3.9 Grids.TOnGetPickListItems Type

Pascal

```
TOnGetPickListItems = procedure (ACol, ARow: Integer; Items: TStrings) of object;
```

C++

```
(ACol, ARow: Integer; Items: TStrings) ( TOnGetPickListItems)();
```

File

Grids

Description

This is type Grids.TOnGetPickListItems.

75.3.10 Grids.TSelectCellEvent Type

Pascal

```
TSelectCellEvent = procedure (Sender: TObject; ACol, ARow: Longint; var CanSelect: Boolean) of
object;
```

C++

```
(Sender: TObject; ACol, ARow: Longint; var CanSelect: Boolean) ( TSelectCellEvent)();
```

File

Grids

Description

This is type Grids.TSelectCellEvent.

75.3.11 Grids.TSetEditEvent Type

TSetEditEvent is the type of the event handlers that assign strings to a cell in a grid.

Pascal

```
TSetEditEvent = procedure (Sender: TObject; ACol, ARow: Longint; const Value: string) of
object;
```

C++

```
(Sender: TObject; ACol, ARow: Longint; const Value: string) ( TSetEditEvent)();
```

File

Grids

Description

Sender is the grid whose cell was just edited.

ACol and ARow indicate the column and row indexes of the cell that was edited.

Value is the string to assign to the cell.

75.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
MaxCustomExtents (see page 15009)	This is constant Grids.MaxCustomExtents.
MaxShortInt (see page 15009)	This is constant Grids.MaxShortInt.

75.4.1 Grids.MaxCustomExtents Constant

Pascal

```
MaxCustomExtents = MaxListSize;
```

C++

```
MaxCustomExtents = MaxListSize;
```

File

Grids

Description

This is constant Grids.MaxCustomExtents.

75.4.2 Grids.MaxShortInt Constant

Pascal

```
MaxShortInt = High(ShortInt);
```

C++

```
MaxShortInt = High(ShortInt);
```

File

Grids

Description

This is constant Grids.MaxShortInt.

76 HelpIntfs Namespace

This is namespace HelpIntfs.

Topics

Name	Description
Classes (see page 15012)	The following table lists classes in this documentation.
Interfaces (see page 15013)	The following table lists interfaces in this documentation.
Functions (see page 15033)	The following table lists functions in this documentation.

Classes

	Name	Description
	EHelpSystemException (see page 15012)	EHelpSystemException is the class for errors raised by the RTL Help System (see page 22079).

Functions

	Name	Description
	GetHelpSystem (see page 15033)	Returns Help System (see page 22079) object.
	GetHelpSystem (see page 15033)	Returns Help System (see page 22079) object.
	RegisterViewer (see page 15033)	Register Help Viewer to receive requests.

Interfaces

	Name	Description
	ICustomHelpViewer (see page 15013)	Help Viewer access for Help System (see page 22079)..
	IExtendedHelpViewer (see page 15017)	Help Viewer with context help support.
	IHelpManager (see page 15020)	IHelpManager provides access to the Help System (see page 22079) for a Help Viewer.
	IHelpSelector (see page 15022)	Interface for Help Selector object.
	IHelpSelector2 (see page 15024)	This is class HelpIntfs.IHelpSelector2.
	IHelpSystem (see page 15025)	Application Help System (see page 22079) that supports multiple Help Viewers.
	IHelpSystem2 (see page 15028)	This is class HelpIntfs.IHelpSystem2.
	IHelpSystemFlags (see page 15029)	This is class HelpIntfs.IHelpSystemFlags.
	ISpecialWinHelpViewer (see page 15031)	Help Viewer that supports special Windows Help requests.

76.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EHelpSystemException (see page 15012)	EHelpSystemException is the class for errors raised by the RTL Help System (see page 22079).

76.1.1 EHelpSystemException Class

EHelpSystemException is the class for errors raised by the RTL Help System (see page 22079).

Class Hierarchy

System::TObject → SysUtils.Exception → HelpIntfs.EHelpSystemException

Pascal

```
EHelpSystemException = class(Exception);
```

C++

```
class EHelpSystemException : public Exception;
```

File

HelpIntfs

Description

The Run-Time Library Help System (see page 22079) raises EHelpSystemException when an error occurs while registering or invoking a help viewer.

76.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	ICustomHelpViewer (see page 15013)	Help Viewer access for Help System (see page 22079)..
	IExtendedHelpViewer (see page 15017)	Help Viewer with context help support.
	IHelpManager (see page 15020)	IHelpManager provides access to the Help System (see page 22079) for a Help Viewer.
	IHelpSelector (see page 15022)	Interface for Help Selector object.
	IHelpSelector2 (see page 15024)	This is class HelpIntfs.IHelpSelector2.
	IHelpSystem (see page 15025)	Application Help System (see page 22079) that supports multiple Help Viewers.
	IHelpSystem2 (see page 15028)	This is class HelpIntfs.IHelpSystem2.
	IHelpSystemFlags (see page 15029)	This is class HelpIntfs.IHelpSystemFlags.
	ISpecialWinHelpViewer (see page 15031)	Help Viewer that supports special Windows Help requests.

76.2.1 ICustomHelpViewer Interface

Help Viewer access for Help System (see page 22079)..

Class Hierarchy

Pascal

```
ICustomHelpViewer = interface(IInterface);
```

C++

```
__interface ICustomHelpViewer : public IInterface;
```

File

HelpIntfs

Description

A Help Viewer is an object which handles application Help requests passed to it by the Help System (see page 22079) on behalf of an application. A Help Viewer is typically a wrapper for an external application, such as a help engine or web browser.

The Help Viewer class must implement the ICustomHelpViewer interface, or one of its descendant interfaces, IExtendedHelpViewer (see page 15017) and ISpecialWinHelpViewer (see page 15031). The Help System (see page 22079) uses methods from these interfaces to pass Help requests to the Help Viewer.

To use a Help Viewer, an application must call the global RegisterViewer (see page 15033) function.

Note: ISpecialWinHelpViewer (see page 15031) is only provided with Windows products. Do not use this type in cross-platform applications.

See Also

RegisterViewer (see page 15033)

Communicating with the Help Manager

Help System Interfaces

Implementing ICustomHelpViewer

76.2.1.1 ICustomHelpViewer Members

The following tables list the members exposed by ICustomHelpViewer.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	CanShowTableOfContents (see page 15014)	Query support for Table of Contents.
	GetHelpStrings (see page 15015)	Return Help Strings that match a keyword.
	GetViewerName (see page 15015)	Return string identifying Help Viewer.
	NotifyID (see page 15016)	Save Viewer's numeric ID.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ShowHelp (see page 15016)	Show a specific Help topic.
	ShowTableOfContents (see page 15016)	Show a Table of Contents.
	ShutDown (see page 15017)	Permanent Viewer shut down.
	SoftShutDown (see page 15017)	This is SoftShutDown, a member of class ICustomHelpViewer.
	UnderstandsKeyword (see page 15017)	Would respond to a keyword-based Help request.

76.2.1.2 ICustomHelpViewer Methods

The methods of the ICustomHelpViewer class are listed here.

Public Methods

	Name	Description
	CanShowTableOfContents (see page 15014)	Query support for Table of Contents.
	GetHelpStrings (see page 15015)	Return Help Strings that match a keyword.
	GetViewerName (see page 15015)	Return string identifying Help Viewer.
	NotifyID (see page 15016)	Save Viewer's numeric ID.
	ShowHelp (see page 15016)	Show a specific Help topic.
	ShowTableOfContents (see page 15016)	Show a Table of Contents.
	ShutDown (see page 15017)	Permanent Viewer shut down.
	SoftShutDown (see page 15017)	This is SoftShutDown, a member of class ICustomHelpViewer.
	UnderstandsKeyword (see page 15017)	Would respond to a keyword-based Help request.

76.2.1.2.1 ICustomHelpViewer.CanShowTableOfContents Method

Query support for Table of Contents.

Pascal

```
function CanShowTableOfContents: Boolean;
```

C++

```
__fastcall Boolean CanShowTableOfContents();
```

Description

The Help System (see page 22079) uses CanShowTableOfContents to determine which Help Viewers know how to display a

Table of Contents. CanShowTableOfContents should return true if the Viewer is ready for a subsequent call to ShowTableOfContents (see page 15016). The Help System (see page 22079) should not call ShowTableOfContents (see page 15016) for a Viewer after CanShowTableOfContents returns false; if this occurs, the Viewer should raise an EHelpSystemException (see page 15012).

See Also

ShowTableOfContents (see page 15027)

ShowTableOfContents (see page 15016)

EHelpSystemException (see page 15012)

76.2.1.2.2 ICustomHelpViewer.GetHelpStrings Method

Return Help Strings that match a keyword.

Pascal

```
function GetHelpStrings(const HelpString: string): TStringList;
```

C++

```
__fastcall TStringList GetHelpStrings(const AnsiString HelpString);
```

Description

The Help System (see page 22079) calls GetHelpStrings to obtain a list of Help Strings for topics that match a keyword-based Help request. GetHelpStrings should expect the keyword in the parameter and return the Help Strings in a TStringList object.

A Help String is an identifying string associated with a specific topic. The Help System (see page 22079) requires that each of the topics provided by a given Viewer have a different Help String. The Help System (see page 22079) does not require that two Viewers never use the same Help String, but an application should still avoid doing this. One strategy is to include the Viewer Name in the Help String.

The Help System (see page 22079) uses Help Strings to provide the Help Selector with a merged list of Help topics that match a request. Typically, Help Strings are human-readable topic descriptions, and the Help Selector simply displays them to the user so the user can choose which topic will be displayed.

See Also

ShowHelp (see page 15027)

UnderstandsKeyword (see page 15017)

ShowHelp (see page 15016)

GetViewerName (see page 15015)

76.2.1.2.3 ICustomHelpViewer.GetViewerName Method

Return string identifying Help Viewer.

Pascal

```
function GetViewerName: string;
```

C++

```
__fastcall AnsiString GetViewerName();
```

Description

The Help System (see page 22079) calls GetViewerName to obtain a string identifying the Help Viewer. Every Help Viewer

within an application must return a unique Viewer Name.

76.2.1.2.4 ICustomHelpViewer.NotifyID Method

Save Viewer's numeric ID.

Pascal

```
procedure NotifyID(const ViewerID: Integer);
```

C++

```
__fastcall NotifyID(const int ViewerID);
```

Description

The Help System (see page 22079) calls NotifyID to pass the application a unique numeric ID associated with a Viewer. The application uses this ID to identify a specific viewer in calls to Release (IHelpManager (see page 15020)).

See Also

Release (see page 15021)

76.2.1.2.5 ICustomHelpViewer.ShowHelp Method

Show a specific Help topic.

Pascal

```
procedure ShowHelp(const HelpString: string);
```

C++

```
__fastcall ShowHelp(const AnsiString HelpString);
```

Description

The Help System (see page 22079) calls ShowHelp when it needs the Help Viewer to display the Help Topic identified by HelpString. The HelpString parameter is a value previously provided by the Help Viewer in response to a GetHelpStrings (see page 15015) call.

76.2.1.2.6 ICustomHelpViewer.ShowTableOfContents Method

Show a Table of Contents.

Pascal

```
procedure ShowTableOfContents;
```

C++

```
__fastcall ShowTableOfContents();
```

Description

The Help System (see page 22079) calls ShowTableOfContents when it needs the Help Viewer to display a Table of Contents. The Help System (see page 22079) does not call ShowTableOfContents for a Viewer unless that Viewer has previously returned true for a call to CanShowTableOfContents (see page 15014).

See Also

CanShowTableOfContents (see page 15014)

76.2.1.2.7 ICustomHelpViewer.ShutDown Method

Permanent Viewer shut down.

Pascal

```
procedure ShutDown;
```

C++

```
__fastcall ShutDown();
```

Description

The Help System (see page 22079) calls ShutDown when it is notified that the application is shutting down. A Viewer should respond to this call by terminating any external applications used to display Help topics. After a call to ShutDown, a Viewer can assume that no more Help requests will arrive.

76.2.1.2.8 ICustomHelpViewer.SoftShutDown Method

Pascal

```
procedure SoftShutDown;
```

C++

```
__fastcall SoftShutDown();
```

Description

This is SoftShutDown, a member of class ICustomHelpViewer.

76.2.1.2.9 ICustomHelpViewer.UnderstandsKeyword Method

Would respond to a keyword-based Help request.

Pascal

```
function UnderstandsKeyword(const HelpString: string): Integer;
```

C++

```
__fastcall int UnderstandsKeyword(const AnsiString HelpString);
```

Description

The Help System (see page 22079) calls UnderstandsKeyword to determine to determine if the Help Viewer can provide topics for a keyword-based Help request based on HelpString. UnderstandsKeyword should return the number of Help topics that would match such a request.

76.2.2 IExtendedHelpViewer Interface

Help Viewer with context help support.

Class Hierarchy

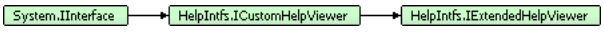

Pascal

```
IExtendedHelpViewer = interface(ICustomHelpViewer);
```

C++

```
__interface IExtendedHelpViewer : public ICustomHelpViewer;
```

File

HelpIntfs

Description

IExtendedHelpViewer extends ICustomHelpViewer (see page 15013) by adding support for Topic IDs and Context IDs.

A topic ID is a short string that uniquely identifies each topic provided by a Help Viewer. A topic ID provides a simple way for an application to display a specific Help topic without searching for it.

A context ID is an integer value that uniquely identifies each Help topic. Context IDs support context-sensitive Help by providing a value that associates forms and controls with their help topics. A form or control can be associated a topic simply by setting its HelpContext property.

See Also

RegisterViewer (see page 15033)

tcontrol

Help System Interfaces

76.2.2.1 IExtendedHelpViewer Members

The following tables list the members exposed by IExtendedHelpViewer.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	CanShowTableOfContents (see page 15014)	Query support for Table of Contents.
	DisplayHelpByContext (see page 15019)	Use Context ID to display topic.
	DisplayTopic (see page 15019)	Use Topic ID to display topic.
	GetHelpStrings (see page 15015)	Return Help Strings that match a keyword.
	GetViewerName (see page 15015)	Return string identifying Help Viewer.
	NotifyID (see page 15016)	Save Viewer's numeric ID.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ShowHelp (see page 15016)	Show a specific Help topic.
	ShowTableOfContents (see page 15016)	Show a Table of Contents.
	ShutDown (see page 15017)	Permanent Viewer shut down.
	SoftShutDown (see page 15017)	This is SoftShutDown, a member of class ICustomHelpViewer.
	UnderstandsContext (see page 15019)	Can Help Viewer respond to Context-based. Help request?
	UnderstandsKeyword (see page 15017)	Would respond to a keyword-based Help request.
	UnderstandsTopic (see page 15020)	Can Help Viewer respond to Topic-based. Help request?

76.2.2.2 IExtendedHelpViewer Methods

The methods of the IExtendedHelpViewer class are listed here.

Public Methods

	Name	Description
	DisplayHelpByContext (see page 15019)	Use Context ID to display topic.

	DisplayTopic (see page 15019)	Use Topic ID to display topic.
	UnderstandsContext (see page 15019)	Can Help Viewer respond to Context-based. Help request?
	UnderstandsTopic (see page 15020)	Can Help Viewer respond to Topic-based. Help request?

76.2.2.2.1 IExtendedHelpViewer.DisplayHelpByContext Method

Use Context ID to display topic.

Pascal

```
procedure DisplayHelpByContext(const ContextID: Integer; const HelpFileName: string);
```

C++

```
__fastcall DisplayHelpByContext(const int ContextID, const AnsiString HelpFileName);
```

Description

The RTL Help System (see page 22079) calls DisplayHelpByContext to have the Viewer handle a context-based Help request. ContextID is the topic's Context ID. HelpFileName is the name of the file containing the topic. Note that some Help engines require the file name to be fully qualified.

See Also

UnderstandsTopic (see page 15020)

76.2.2.2.2 IExtendedHelpViewer.DisplayTopic Method

Use Topic ID to display topic.

Pascal

```
procedure DisplayTopic(const Topic: string);
```

C++

```
__fastcall DisplayTopic(const AnsiString Topic);
```

Description

The RTL Help System (see page 22079) calls DisplayTopic to have the Viewer handle a topic-based Help request. Topic is the Topic ID, a short string that must be unique for all the topics maintained by a particular Viewer.

See Also

UnderstandsTopic (see page 15020)

76.2.2.2.3 IExtendedHelpViewer.UnderstandsContext Method

Can Help Viewer respond to Context-based. Help request?

Pascal

```
function UnderstandsContext(const ContextID: Integer; const HelpFileName: string): Boolean;
```

C++

```
__fastcall Boolean UnderstandsContext(const int ContextID, const AnsiString HelpFileName);
```

Description

UnderstandsContext should return true if the Viewer is able to display the topic identified by ContextID in the file HelpFileName. When the RTL Help System (see page 22079) receives a Context-based Help request, it calls UnderstandsContext for each Viewer that implements it until the method returns true. (Viewers are polled in the order they were registered.) The Help System (see page 22079) than calls that Viewer's DisplayHelpByContext (see page 15019) method.

See Also

DisplayHelpByContext (see page 15019)

76.2.2.2.4 IExtendedHelpViewer.UnderstandsTopic Method

Can Help Viewer respond to Topic-based. Help request?

Pascal

```
function UnderstandsTopic(const Topic: string): Boolean;
```

C++

```
__fastcall Boolean UnderstandsTopic(const AnsiString Topic);
```

Description

UnderstandsTopic should return true if the Viewer is able to display the topic identified by the Topic parameter. When the RTL Help System (see page 22079) receives a Topic-based Help request, it calls UnderstandsTopic for each Viewer that implements it until the method returns true. (Viewers are polled in the order they were registered.) The Help System (see page 22079) than calls that Viewer's DisplayTopic (see page 15019) method.

See Also

DisplayTopic (see page 15019)

76.2.3 IHelpManager Interface

IHelpManager provides access to the Help System (see page 22079) for a Help Viewer.

Class Hierarchy

HelpIntfs.IHelpManager

Pascal

```
IHelpManager = interface;
```

C++

```
__interface IHelpManager;
```

File

HelpIntfs

Description

IHelpManager defines methods that a Help Viewer uses to interact with the Help System (see page 22079). A Viewer accesses these methods using the object that the global RegisterViewer (see page 15033) function returns.

See Also

RegisterViewer (see page 15033)

Help System Interfaces

76.2.3.1 IHelpManager Members

The following tables list the members exposed by IHelpManager.

Public Methods

	Name	Description
	GetHandle (see page 15021)	Get application window handle.
	GetHelpFile (see page 15021)	Get Help File name.
	Release (see page 15021)	Notify Help System (see page 22079) of Help Viewer shutdown.

76.2.3.2 IHelpManager Methods

The methods of the IHelpManager class are listed here.

Public Methods

	Name	Description
	GetHandle (see page 15021)	Get application window handle.
	GetHelpFile (see page 15021)	Get Help File name.
	Release (see page 15021)	Notify Help System (see page 22079) of Help Viewer shutdown.

76.2.3.2.1 IHelpManager.GetHandle Method

Get application window handle.

Pascal

```
function GetHandle: LongInt;
```

C++

```
__fastcall LongInt GetHandle();
```

Description

GetHandle returns the main window handle of the application responsible for the current Help request.

76.2.3.2.2 IHelpManager.GetHelpFile Method

Get Help File name.

Pascal

```
function GetHelpFile: string;
```

C++

```
__fastcall AnsiString GetHelpFile();
```

Description

GetHelpFile returns the name of the Help File associated with the current Help request.

76.2.3.2.3 IHelpManager.Release Method

Notify Help System (see page 22079) of Help Viewer shutdown.

Pascal

```
procedure Release(const ViewerID: Integer);
```

C++

```
__fastcall Release(const int ViewerID);
```

Description

Release disconnects the Help Viewer from the RTL Help System (see page 22079). The ViewerID parameter is the value the Help System (see page 22079) previously passed to the Viewer by calling NotifyID.

A Help Viewer should call Release when it is about to shut down, unless the Viewer is shutting down in response to a call to ShutDown.

See Also

NotifyID (see page 15016)

ShutDown (see page 15017)

76.2.4 IHelpSelector Interface

Interface for Help Selector object.

Class Hierarchy

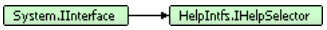

Pascal

```
IHelpSelector = interface(IInterface);
```

C++

```
__interface IHelpSelector : public IInterface;
```

File

HelpIntfs

Description

When an application uses more than one Help Viewer, there may be multiple responses to a Help request. By default, the Help System (see page 22079) polls the Viewers in the order they were registered, and gives the request to the first viewer that claims to be able to handle it.

An application can optionally provide a Help Selector, which determines the Viewer for keyword-based Help requests and for requests to display a Table of Contents. To define a Help Selector, create an object that implements IHelpSelector and pass it to the Help System (see page 22079), using AssignHelpSelector.

The Help Selector cannot affect context-based or topic-based Help requests, which always go to the first Viewer that claims to be able to handle the request.

See Also

RegisterViewer (see page 15033)

IHelpSystem (see page 15025)

Help System Interfaces

76.2.4.1 IHelpSelector Members

The following tables list the members exposed by IHelpSelector.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	SelectKeyword (see page 15023)	Keyword-based Help support for multiple Viewers.
	TableOfContents (see page 15024)	Choose Viewer to display Table of Contents.

76.2.4.2 IHelpSelector Methods

The methods of the IHelpSelector class are listed here.

Public Methods

	Name	Description
	SelectKeyword (see page 15023)	Keyword-based Help support for multiple Viewers.
	TableOfContents (see page 15024)	Choose Viewer to display Table of Contents.

76.2.4.2.1 IHelpSelector.SelectKeyword Method

Keyword-based Help support for multiple Viewers.

Pascal

```
function SelectKeyword(Keywords: TStrings): Integer;
```

C++

```
__fastcall int SelectKeyword(TStrings Keywords);
```

Description

The Help System (see page 22079) calls SelectKeyword when more than one Help Viewer has topics that match a keyword-based Help request. The Keywords parameter contains a merged list of Help Strings, obtained by calling GetHelpStrings for each Viewer. SelectKeyword indicates a specific topic by returning the index of a specific Help String. The Help System (see page 22079) then passes this Help String to the ShowHelp method of the appropriate Viewer. If the return value is negative, the Help System (see page 22079) assumes that the Help request has been aborted and does nothing.

Typically, SelectKeyword displays a dialog box so the user can choose one of the Help Strings, and returns the index of the user's choice, or a negative value if the user cancels the dialog. This duplicates a feature of most Viewers, but allows the user to see all matching topics in a single list.

Note: SelectKeyword must not modify or rearrange the Help Strings list.

See Also

UnderstandsKeyword (see page 15017)

GetHelpStrings (see page 15015)

ShowHelp (see page 15016)

76.2.4.2.2 IHelpSelector.TableOfContents Method

Choose Viewer to display Table of Contents.

Pascal

```
function TableOfContents(Contents: TStrings): Integer;
```

C++

```
__fastcall int TableOfContents(TStrings Contents);
```

Description

The Help System (see page 22079) calls TableOfContents when more than one Help Viewer responds to a request to show a Help Table of Contents. The Help System (see page 22079) passes a list of Viewer names in the Contents parameter. TableOfContents should return the index of the chosen Viewer's name.

Note: TableOfContents must not modify or rearrange the Viewer names list.

See Also

CanShowTableOfContents (see page 15014)

76.2.5 IHelpSelector2 Interface

Class Hierarchy

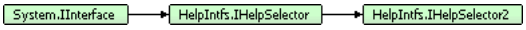

Pascal

```
IHelpSelector2 = interface(IHelpSelector);
```

C++

```
__interface IHelpSelector2 : public IHelpSelector;
```

File

HelpIntfs

Description

This is class HelpIntfs.IHelpSelector2.

76.2.5.1 IHelpSelector2 Members

The following tables list the members exposed by IHelpSelector2.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	SelectContext (see page 15025)	This is SelectContext, a member of class IHelpSelector2.
	SelectKeyword (see page 15023)	Keyword-based Help support for multiple Viewers.
	TableOfContents (see page 15024)	Choose Viewer to display Table of Contents.

76.2.5.2 IHelpSelector2 Methods

The methods of the IHelpSelector2 class are listed here.

Public Methods

	Name	Description
	SelectContext (see page 15025)	This is SelectContext, a member of class IHelpSelector2.

76.2.5.2.1 IHelpSelector2.SelectContext Method

Pascal

```
function SelectContext(Viewers: TStrings): Integer;
```

C++

```
__fastcall int SelectContext(TStrings Viewers);
```

Description

This is SelectContext, a member of class IHelpSelector2.

76.2.6 IHelpSystem Interface

Application Help System (see page 22079) that supports multiple Help Viewers.

Class Hierarchy

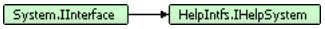

Pascal

```
IHelpSystem = interface(IInterface);
```

C++

```
__interface IHelpSystem : public IInterface;
```

File

HelpIntfs

Description

The Run-Time Library Help System (see page 22079) allows an application to access one or more Help Viewers through a uniform interface, IHelpSystem. To obtain an object that implements IHelpSystem, use the HelpSystem property of the Application global variable, or call the global GetHelpSystem (see page 15033) function.

See Also

Help System Interfaces

76.2.6.1 IHelpSystem Members

The following tables list the members exposed by IHelpSystem.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AssignHelpSelector (see page 15026)	Assign a Help Selector object.
	Hook (see page 15026)	Do a WinHelp-style Help request.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ShowContextHelp (see page 15027)	Do Context-based Help request.
	ShowHelp (see page 15027)	Do Keyword-based Help request.
	ShowTableOfContents (see page 15027)	Show Table of Contents.
	ShowTopicHelp (see page 15028)	Show Help topic.

76.2.6.2 IHelpSystem Methods

The methods of the IHelpSystem class are listed here.

Public Methods

	Name	Description
	AssignHelpSelector (see page 15026)	Assign a Help Selector object.
	Hook (see page 15026)	Do a WinHelp-style Help request.
	ShowContextHelp (see page 15027)	Do Context-based Help request.
	ShowHelp (see page 15027)	Do Keyword-based Help request.
	ShowTableOfContents (see page 15027)	Show Table of Contents.
	ShowTopicHelp (see page 15028)	Show Help topic.

76.2.6.2.1 IHelpSystem.AssignHelpSelector Method

Assign a Help Selector object.

Pascal

```
procedure AssignHelpSelector(Selector: IHelpSelector);
```

C++

```
__fastcall AssignHelpSelector(IHelpSelector Selector);
```

Description

Call AssignHelpSelector to provide a Help Selector object. Use of a Help Selector allows an application to define the way a Help Viewer is chosen to handle keyword-based Help requests and requests to display a Table of Contents.

76.2.6.2.2 IHelpSystem.Hook Method

Do a WinHelp-style Help request.

Pascal

```
function Hook(Handle: Longint; HelpFile: string; Comand: Word; Data: Longint): Boolean;
```

C++

```
__fastcall Boolean Hook(Longint Handle, AnsiString HelpFile, Word Comand, Longint Data);
```

Description

Hook accepts Help requests that uses the same conventions as the Microsoft Windows WinHelp API. Hook should only be called for requests that cannot be translated into calls to ShowContextHelp (see page 15027), ShowHelp (see page 15027),

ShowTableOfContents (see page 15027), or ShowTopicHelp (see page 15028).

Note: If the Help System (see page 22079) is unable to decipher the Help request, it looks for a Viewer that implements ISpecialWinHelpViewer (see page 15031). If the application has registered such a viewer, the request succeeds if the Viewer is able to handle it. Otherwise the request fails.

76.2.6.2.3 IHelpSystem.ShowContextHelp Method

Do Context-based Help request.

Pascal

```
procedure ShowContextHelp(const ContextID: Longint; const HelpFileName: string);
```

C++

```
__fastcall ShowContextHelp(const Longint ContextID, const AnsiString HelpFileName);
```

Description

Call ShowContextHelp to display a single Help topic, identified by ContextID and HelpFileName. For context-based Help requests to work, the application must have previously registered at least one Help Viewer that implements IExtendedHelpViewer (see page 15017). If no such Viewer is found, an EHelpSystemException (see page 15012) is raised.

76.2.6.2.4 IHelpSystem.ShowHelp Method

Do Keyword-based Help request.

Pascal

```
procedure ShowHelp(const HelpKeyword: string; const HelpFileName: string);
```

C++

```
__fastcall ShowHelp(const AnsiString HelpKeyword, const AnsiString HelpFileName);
```

Description

Call ShowHelp to submit a keyword-based Help request. By default, the RTL Help System (see page 22079) gives the request to the first Help Viewer that claims to be able to handle it. Typically, the Viewer passes the Help request to an external Help engine, which displays a list of matching topics (if more than one topic matches) or goes directly to a specific topic (if a single topic matches).

To support keyword-based Help requests using multiple Help Viewers, assign a Help Selector, using the AssignHelpSelector (see page 15026) method.

76.2.6.2.5 IHelpSystem.ShowTableOfContents Method

Show Table of Contents.

Pascal

```
procedure ShowTableOfContents;
```

C++

```
__fastcall ShowTableOfContents();
```

Description

Call ShowTableOfContents to request that the Help System (see page 22079) display a Table of Contents.

Note: Tapplication.HelpShowTableOfContents is the preferred method of opening the Table of Contents.

76.2.6.2.6 IHelpSystem.ShowTopicHelp Method

Show Help topic.

Pascal

```
procedure ShowTopicHelp(const Topic: string; const HelpFileName: string);
```

C++

```
__fastcall ShowTopicHelp(const AnsiString Topic, const AnsiString HelpFileName);
```

Description

Call ShowTopicHelp to request that the Help System (see page 22079) display a specific Help topic.

The Topic parameter is a short string identifying the specific topic.

The HelpFileName parameter is the name of the file containing the topic. If this parameter is a zero-length string, the HelpFile property of the global Application variable is used.

76.2.7 IHelpSystem2 Interface

Class Hierarchy

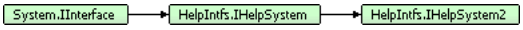

Pascal

```
IHelpSystem2 = interface(IHelpSystem);
```

C++

```
__interface IHelpSystem2 : public IHelpSystem;
```

File

HelpIntfs

Description

This is class HelpIntfs.IHelpSystem2.

76.2.7.1 IHelpSystem2 Members

The following tables list the members exposed by IHelpSystem2.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AssignHelpSelector (see page 15026)	Assign a Help Selector object.
	Hook (see page 15026)	Do a WinHelp-style Help request.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ShowContextHelp (see page 15027)	Do Context-based Help request.
	ShowHelp (see page 15027)	Do Keyword-based Help request.
	ShowTableOfContents (see page 15027)	Show Table of Contents.
	ShowTopicHelp (see page 15028)	Show Help topic.

	UnderstandsKeyword (see page 15029)	This is UnderstandsKeyword, a member of class IHelpSystem2.

76.2.7.2 IHelpSystem2 Methods

The methods of the IHelpSystem2 class are listed here.

Public Methods

	Name	Description
	UnderstandsKeyword (see page 15029)	This is UnderstandsKeyword, a member of class IHelpSystem2.

76.2.7.2.1 IHelpSystem2.UnderstandsKeyword Method

Pascal

```
function UnderstandsKeyword(const HelpKeyword: string; const HelpFileName: string): Boolean;
```

C++

```
__fastcall Boolean UnderstandsKeyword(const AnsiString HelpKeyword, const AnsiString
HelpFileName);
```

Description

This is UnderstandsKeyword, a member of class IHelpSystem2.

76.2.8 IHelpSystemFlags Interface

Class Hierarchy

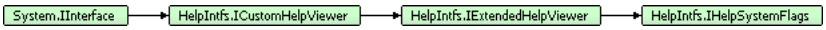

Pascal

```
IHelpSystemFlags = interface(IExtendedHelpViewer);
```

C++

```
__interface IHelpSystemFlags : public IExtendedHelpViewer;
```

File

HelpIntfs

Description

This is class HelpIntfs.IHelpSystemFlags.

76.2.8.1 IHelpSystemFlags Members

The following tables list the members exposed by IHelpSystemFlags.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	CanShowTableOfContents (see page 15014)	Query support for Table of Contents.

	DisplayHelpByContext (see page 15019)	Use Context ID to display topic.
	DisplayTopic (see page 15019)	Use Topic ID to display topic.
	GetHelpStrings (see page 15015)	Return Help Strings that match a keyword.
	GetUseDefaultTopic (see page 15030)	This is GetUseDefaultTopic, a member of class IHelpSystemFlags.
	GetViewerName (see page 15015)	Return string identifying Help Viewer.
	NotifyID (see page 15016)	Save Viewer's numeric ID.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	SetUseDefaultTopic (see page 15030)	This is SetUseDefaultTopic, a member of class IHelpSystemFlags.
	ShowHelp (see page 15016)	Show a specific Help topic.
	ShowTableOfContents (see page 15016)	Show a Table of Contents.
	ShutDown (see page 15017)	Permanent Viewer shut down.
	SoftShutDown (see page 15017)	This is SoftShutDown, a member of class ICustomHelpViewer.
	UnderstandsContext (see page 15019)	Can Help Viewer respond to Context-based. Help request?
	UnderstandsKeyword (see page 15017)	Would respond to a keyword-based Help request.
	UnderstandsTopic (see page 15020)	Can Help Viewer respond to Topic-based. Help request?

76.2.8.2 IHelpSystemFlags Methods

The methods of the IHelpSystemFlags class are listed here.

Public Methods

	Name	Description
	GetUseDefaultTopic (see page 15030)	This is GetUseDefaultTopic, a member of class IHelpSystemFlags.
	SetUseDefaultTopic (see page 15030)	This is SetUseDefaultTopic, a member of class IHelpSystemFlags.

76.2.8.2.1 IHelpSystemFlags.GetUseDefaultTopic Method

Pascal

```
function GetUseDefaultTopic: Boolean;
```

C++

```
__fastcall Boolean GetUseDefaultTopic();
```

Description

This is GetUseDefaultTopic, a member of class IHelpSystemFlags.

76.2.8.2.2 IHelpSystemFlags.SetUseDefaultTopic Method

Pascal

```
procedure SetUseDefaultTopic(AValue: Boolean);
```

C++

```
__fastcall SetUseDefaultTopic(Boolean AValue);
```

Description

This is SetUseDefaultTopic, a member of class IHelpSystemFlags.

76.2.9 ISpecialWinHelpViewer Interface

Help Viewer that supports special Windows Help requests.

Class Hierarchy

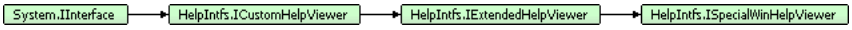

Pascal

```
ISpecialWinHelpViewer = interface(IExtendedHelpViewer);
```

C++

```
__interface ISpecialWinHelpViewer : public IExtendedHelpViewer;
```

File

HelpIntfs

Description

ISpecialWinHelpViewer supports Help requests that cannot be easily generalized, and should only be handled by the Microsoft Windows Help engine or a closely compatible application.

ISpecialWinHelpViewer consists of a single method, CallWinHelp (see page 15032), which should pass the Help request to the Help engine with as little manipulation as possible. An application needs to provide only one Viewer that implements ISpecialWinHelpViewer. If more than one Viewer is registered that implements ISpecialWinHelpViewer, only the first Viewer ever receives calls to CallWinHelp (see page 15032).

See Also

IhelpManager

IhelpSystem

ISpecialWinHelpViewer

RegisterViewer (see page 15033)

Help System Interfaces

76.2.9.1 ISpecialWinHelpViewer Members

The following tables list the members exposed by ISpecialWinHelpViewer.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	CallWinHelp (see page 15032)	Pass Help request directly to engine.
	CanShowTableOfContents (see page 15014)	Query support for Table of Contents.
	DisplayHelpByContext (see page 15019)	Use Context ID to display topic.
	DisplayTopic (see page 15019)	Use Topic ID to display topic.
	GetHelpStrings (see page 15015)	Return Help Strings that match a keyword.
	GetViewerName (see page 15015)	Return string identifying Help Viewer.
	NotifyID (see page 15016)	Save Viewer's numeric ID.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	ShowHelp (see page 15016)	Show a specific Help topic.
	ShowTableOfContents (see page 15016)	Show a Table of Contents.
	ShutDown (see page 15017)	Permanent Viewer shut down.
	SoftShutDown (see page 15017)	This is SoftShutDown, a member of class ICustomHelpViewer.
	UnderstandsContext (see page 15019)	Can Help Viewer respond to Context-based. Help request?
	UnderstandsKeyword (see page 15017)	Would respond to a keyword-based Help request.
	UnderstandsTopic (see page 15020)	Can Help Viewer respond to Topic-based. Help request?

76.2.9.2 ISpecialWinHelpViewer Methods

The methods of the ISpecialWinHelpViewer class are listed here.

Public Methods

	Name	Description
	CallWinHelp (see page 15032)	Pass Help request directly to engine.

76.2.9.2.1 ISpecialWinHelpViewer.CallWinHelp Method

Pass Help request directly to engine.

Pascal

```
function CallWinHelp(Handle: LongInt; const HelpFile: string; Command: Word; Data: LongInt):
Boolean;
```

C++

```
__fastcall Boolean CallWinHelp(LongInt Handle, const AnsiString HelpFile, Word Command,
LongInt Data);
```

Description

The RTL Help System (see page 22079) calls CallWinHelp when it receives a Windows-style Help request that cannot be generalized as a Topic-based or Context-based request. CallWinHelp should pass the request directly to the Windows winhelp() API or to a functional equivalent.

CallWinHelp should return true if it was able to handle the request.

76.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetHelpSystem (see page 15033)	Returns Help System (see page 22079) object.
	GetHelpSystem (see page 15033)	Returns Help System (see page 22079) object.
	RegisterViewer (see page 15033)	Register Help Viewer to receive requests.

76.3.1 HelpIntfs.GetHelpSystem Function

Returns Help System (see page 22079) object.

Pascal

```
function GetHelpSystem(out System: IHelpSystem): Boolean; overload;
function GetHelpSystem(out System: IHelpSystem2): Boolean; overload;
```

C++

```
Boolean GetHelpSystem(IHelpSystem System);
Boolean GetHelpSystem(IHelpSystem2 System);
```

File

HelpIntfs

Description

GetHelpSystem returns an object that implements IHelpSystem (see page 15025). An application calls IHelpSystem (see page 15025) methods to interact with the Help System (see page 22079).

GetHelpSystem returns 1 if the Help System (see page 22079) object was successfully obtained, 0 otherwise.

See Also

TApplication

76.3.2 HelpIntfs.RegisterViewer Function

Register Help Viewer to receive requests.

Pascal

```
function RegisterViewer(const newViewer: ICustomHelpViewer; out Manager: IHelpManager):
Integer;
```

C++

```
int RegisterViewer(const ICustomHelpViewer newViewer, IHelpManager Manager);
```

File

HelpIntfs

Description

Call RegisterViewer to register a Help Viewer with the Help System (see page 22079). An application can register any number of Viewers. The order in which Help Viewers are registered is significant, because each Help request causes the RTL Help System (see page 22079) to poll the Viewers in the order they were registered, and give the Help request to the first viewer that claims to be able to handle it. An application can override this behavior for keyword-based Help requests and for requests to display a Table of Contents by providing a Help Selector object.

Pass the Viewer object in the newViewer parameter.

RegisterViewer returns a Help Manager object in the Manager parameter; the Viewer uses this object to interact with the Help System (see page 22079).

The return value of RegisterViewer is 1 is the Viewer was successfully registered, 0 otherwise.

77 HTTPApp Namespace

This is namespace HTTPApp.

Topics

Name	Description
Classes (see page 15038)	The following table lists classes in this documentation.
Interfaces (see page 15219)	The following table lists interfaces in this documentation.
Functions (see page 15239)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 15246)	The following table lists structs, records, enums in this documentation.
Types (see page 15248)	The following table lists types in this documentation.
Constants (see page 15250)	The following table lists constants in this documentation.

Classes

	Name	Description
	EWebBrokerException (see page 15038)	EWebBrokerException is an exception raised by WebBroker (see page 24617) objects.
	TAbstractContentParser (see page 15039)	TAbstractContentParser is the base class for objects that parse the content of an HTTP request.
	TAbstractWebModuleFactory (see page 15043)	TAbstractWebModuleFactory is the base class for all Web module factory objects.
	TAbstractWebModuleList (see page 15051)	TAbstractWebModuleList lists the WebModules in a WebSnap application and provides access to their factory objects.
	TAbstractWebPageInfo (see page 15060)	TAbstractWebPageInfo represents the attributes of a Web page module.
	TAbstractWebPageModuleFactory (see page 15070)	TAbstractWebPageModuleFactory is the base class for Web module factory objects whose generated Web modules create Web pages.
	TAbstractWebRequestFile (see page 15073)	TAbstractWebRequestFile describes an uploaded file from an HTTP request.
	TAbstractWebRequestFiles (see page 15078)	TAbstractWebRequestFiles is a collection of TAbstractWebRequestFile (see page 15073) objects.
	TAbstractWebSession (see page 15082)	TAbstractWebSession is the base class for objects that represent a session in a WebSnap application.
	TContentParser (see page 15088)	TContentParser is the default parser for extracting named value strings and uploaded files from an HTTP request.
	TCookie (see page 15092)	TCookie represents a Cookie header in an HTTP response message.
	TCookieCollection (see page 15098)	TCookieCollection is a collection of TCookie (see page 15092) objects.
	TCustomContentProducer (see page 15102)	TCustomContentProducer is the base class for all objects that produce the string that is sent as the contents of an HTTP response message.
	TCustomWebDispatcher (see page 15110)	TCustomWebDispatcher is the abstract base class for Web dispatcher objects.
	TDefaultWebAppServices (see page 15125)	This is class HTTPApp.TDefaultWebAppServices.

	TWebActionItem (see page 15130)	TWebActionItem allows a Web server application to create and send responses to HTTP request messages.
	TWebActionItems (see page 15140)	TWebActionItems is a collection of TWebActionItem (see page 15130) objects.
	TWebDispatcher (see page 15147)	TWebDispatcher passes HTTP request messages on to the appropriate action items that assemble a response.
	TWebModule (see page 15154)	TWebModule is the automatically generated Web module for a Web server application.
	TWebRequest (see page 15161)	TWebRequest is the base class for all objects that represent client request information that is passed to a Web server application.
	TWebResponse (see page 15191)	TWebResponse is the base class for all objects that represent HTTP messages sent in response to an HTTP request message.

Constants

Name	Description
MAX_DATETIMES (see page 15250)	This is constant HTTPApp.MAX_DATETIMES.
MAX_INTEGERS (see page 15250)	This is constant HTTPApp.MAX_INTEGERS.
MAX_STRINGS (see page 15251)	This is constant HTTPApp.MAX_STRINGS.
sDateFormat (see page 15251)	This is constant HTTPApp.sDateFormat.

Functions

	Name	Description
	DayOfWeekStr (see page 15239)	Converts a TDateTime into a string representing the day of the week.
	DosPathToUnixPath (see page 15240)	Converts a DOS-compatible path specification to a Unix-compatible path specification.
	ExtractHeaderFields (see page 15240)	Parses a multi-valued string into its constituent fields.
	ExtractHTTPFields (see page 15241)	Parses a multi-valued string into its constituent fields.
	HTMLDecode (see page 15241)	This is function HTTPApp.HTMLDecode.
	HTMLEncode (see page 15242)	This is function HTTPApp.HTMLEncode.
	HTTPDecode (see page 15242)	Decodes a string that includes HTTP escape characters.
	HTTPEncode (see page 15243)	Converts a string into a form that contains no values that are illegal in an HTTP message header.
	MonthStr (see page 15243)	Converts a TDateTime into a string representing the month.
	ParseDate (see page 15244)	Converts a date value from an HTTP message header into a TDateTime.
	RegisterContentParser (see page 15244)	This is function HTTPApp.RegisterContentParser.
	StatusString (see page 15244)	Returns the explanatory string associated with one of the predefined HTTP response status codes.
	UnixPathToDosPath (see page 15245)	Converts a Unix-compatible path specification to a Dos-compatible path specification.

Interfaces

	Name	Description
	IGetAppDispatcher (see page 15219)	This is class HTTPApp.IGetAppDispatcher.
	IGetWebAppServices (see page 15220)	This is class HTTPApp.IGetWebAppServices.
	IGetWebRequestHandler (see page 15221)	This is class HTTPApp.IGetWebRequestHandler.
	IMultiModuleSupport (see page 15222)	This is class HTTPApp.IMultiModuleSupport.
	IProduceContent (see page 15223)	IProduceContent is the interface for generating the content of a Web page.
	IProduceContentFrom (see page 15225)	This is class HTTPApp.IProduceContentFrom.
	ISetAppDispatcher (see page 15226)	This is class HTTPApp.ISetAppDispatcher.
	IWebAppServices (see page 15227)	IWebAppServices is the interface for setting up and finishing the processing of HTTP request messages.
	IWebDispatch (see page 15230)	
	IWebDispatchActions (see page 15233)	IWebDispatchActions is the interface for components that dispatch HTTP request messages using action items.
	IWebDispatcherAccess (see page 15234)	IWebDispatcherAccess provides access to the current HTTP request and response object.
	IWebExceptionHandler (see page 15236)	This is class HTTPApp.IWebExceptionHandler.
	IWebRequestHandler (see page 15237)	IWebRequestHandler is the interface for objects that dispatch HTTP request messages.

Structs, Records, Enums

	Name	Description
	TMethodType (see page 15246)	This is record HTTPApp.TMethodType.
	TWebModuleCacheMode (see page 15246)	This is record HTTPApp.TWebModuleCacheMode.
	TWebModuleCreateMode (see page 15247)	This is record HTTPApp.TWebModuleCreateMode.

Types

Name	Description
TContentParserClass (see page 15248)	This is type HTTPApp.TContentParserClass.
THTTPMethodEvent (see page 15248)	This is type HTTPApp.THTTPMethodEvent.
TModuleAddedProc (see page 15249)	This is type HTTPApp.TModuleAddedProc.
TWebExceptionEvent (see page 15249)	This is type HTTPApp.TWebExceptionEvent.
TWebModuleContext (see page 15249)	This is type HTTPApp.TWebModuleContext.

77.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EWebBrokerException (see page 15038)	EWebBrokerException is an exception raised by WebBroker (see page 24617) objects.
	TAbstractContentParser (see page 15039)	TAbstractContentParser is the base class for objects that parse the content of an HTTP request.
	TAbstractWebModuleFactory (see page 15043)	TAbstractWebModuleFactory is the base class for all Web module factory objects.
	TAbstractWebModuleList (see page 15051)	TAbstractWebModuleList lists the WebModules in a WebSnap application and provides access to their factory objects.
	TAbstractWebPageInfo (see page 15060)	TAbstractWebPageInfo represents the attributes of a Web page module.
	TAbstractWebPageModuleFactory (see page 15070)	TAbstractWebPageModuleFactory is the base class for Web module factory objects whose generated Web modules create Web pages.
	TAbstractWebRequestFile (see page 15073)	TAbstractWebRequestFile describes an uploaded file from an HTTP request.
	TAbstractWebRequestFiles (see page 15078)	TAbstractWebRequestFiles is a collection of TAbstractWebRequestFile (see page 15073) objects.
	TAbstractWebSession (see page 15082)	TAbstractWebSession is the base class for objects that represent a session in a WebSnap application.
	TContentParser (see page 15088)	TContentParser is the default parser for extracting named value strings and uploaded files from an HTTP request.
	TCookie (see page 15092)	TCookie represents a Cookie header in an HTTP response message.
	TCookieCollection (see page 15098)	TCookieCollection is a collection of TCookie (see page 15092) objects.
	TCustomContentProducer (see page 15102)	TCustomContentProducer is the base class for all objects that produce the string that is sent as the contents of an HTTP response message.
	TCustomWebDispatcher (see page 15110)	TCustomWebDispatcher is the abstract base class for Web dispatcher objects.
	TDefaultWebAppServices (see page 15125)	This is class HTTPApp.TDefaultWebAppServices.
	TWebActionItem (see page 15130)	TWebActionItem allows a Web server application to create and send responses to HTTP request messages.
	TWebActionItems (see page 15140)	TWebActionItems is a collection of TWebActionItem (see page 15130) objects.
	TWebDispatcher (see page 15147)	TWebDispatcher passes HTTP request messages on to the appropriate action items that assemble a response.
	TWebModule (see page 15154)	TWebModule is the automatically generated Web module for a Web server application.
	TWebRequest (see page 15161)	TWebRequest is the base class for all objects that represent client request information that is passed to a Web server application.
	TWebResponse (see page 15191)	TWebResponse is the base class for all objects that represent HTTP messages sent in response to an HTTP request message.

77.1.1 EWebBrokerException Class

EWebBrokerException is an exception raised by WebBroker (see page 24617) objects.

Class Hierarchy

Pascal

```
EWebBrokerException = class(Exception);
```

C++

```
class EWebBrokerException : public Exception;
```

File

HTTPApp

Description

EWebBrokerException is raised when a WebBroker (see page 24617) object encounters a problem. It functions identically to its parent class, TException.

See Also

Exception (see page 22750)

77.1.2 TAbstractContentParser Class

TAbstractContentParser is the base class for objects that parse the content of an HTTP request.

Class Hierarchy

Pascal

```
TAbstractContentParser = class(TObject);
```

C++

```
class TAbstractContentParser : public TObject;
```

File

HTTPApp

Description

Web request objects use a descendant of TAbstractContentParser to retrieve information about the named values in the content of the HTTP request they represent. TAbstractContentParser defines two protected methods, GetContentFields (see page 15042) and GetFiles (see page 15042), which the Web request uses to obtain values for its ContentFields and Files properties, respectively.

TAbstractContentParser should not be instantiated. The GetContentFields (see page 15042) and GetFiles (see page 15042) methods are abstract or, in C++ terminology, pure virtual, which means that TAbstractContentParser does not supply an implementation. Each TAbstractContentParser descendant overrides these methods to support its own method of extracting named value strings and uploaded files from an HTTP request. The TAbstractContentParser descendant also overrides the public CanParse (see page 15041) method to check whether the request message contains information encoded in a way that the content parser understands.

Each TAbstractContentParser descendant registers itself with the Web server application. Then, when a Web request object needs to parse its content to set the ContentFields and Files properties, it searches through the registered content parsers to locate the most recently registered parser that can parse the current content.

See Also

TWebRequest (see page 15161)

TWebRequest (see page 15161)

TWebRequest (see page 15161)

TContentParser (see page 15088)

TMultipartContentParser (see page 18710)

77.1.2.1 TAbstractContentParser Members

The following tables list the members exposed by TAbstractContentParser.

Protected Methods

	Name	Description
	GetContentFields (see page 15042)	Returns the named value strings that are included in the content of the current Web request.
	GetFiles (see page 15042)	Returns a set of uploaded files that the parser extracts from the current Web request.

Protected Properties

	Name	Description
	WebRequest (see page 15043)	Indicates the Web request object that uses this parser to parse its content.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanParse (see page 15041)	Indicates whether a specified request message contains information encoded in a way that this parser understands.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 15041)	Creates and initializes a new TAbstractContentParser (see page 15039) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

77.1.2.2 TAbstractContentParser.Create Constructor

Creates and initializes a new TAbstractContentParser (see page 15039) object.

Pascal

```
constructor Create(AWebRequest: TWebRequest); virtual;
```

C++

```
virtual __fastcall TAbstractContentParser(TWebRequest AWebRequest);
```

Description

Do not create instances of TAbstractContentParser (see page 15039). This class and its descendants are intended only as helper objects to the Web request. The Web request creates its own instance of a TAbstractContentParser (see page 15039) descendant that can parse its content.

AWebRequest is the Web request object that creates this content parser to extract information from its content. This becomes the value of the WebRequest (see page 15043) property.

See Also

WebRequest (see page 15043)

77.1.2.3 TAbstractContentParser Methods

The methods of the TAbstractContentParser class are listed here.

Protected Methods

	Name	Description
	GetContentFields (see page 15042)	Returns the named value strings that are included in the content of the current Web request.
	GetFiles (see page 15042)	Returns a set of uploaded files that the parser extracts from the current Web request.

Public Methods

	Name	Description
	CanParse (see page 15041)	Indicates whether a specified request message contains information encoded in a way that this parser understands.

77.1.2.3.1 TAbstractContentParser.CanParse Method

Indicates whether a specified request message contains information encoded in a way that this parser understands.

Pascal

```
class function CanParse(AWebRequest: TWebRequest): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanParse(TWebRequest AWebRequest);
```

Description

CanParse indicates whether a content parser class can handle parsing a specified request message. CanParse is a class method (Delphi) or static method (C++), which means the Web request does not need an instantiated object to call this method. The Web request object calls the CanParse method of every registered TAbstractContentParser (see page 15039) descendant until it finds one that can parse its content.

AWebRequest is the current HTTP request message.

CanParse returns true if this class can parse AWebRequest, false if AWebRequest does not indicate an encoding that this class understands.

In TAbstractContentParser (see page 15039), CanParse always returns false. Descendant classes that implement the GetContentFields (see page 15042) and GetFiles (see page 15042) methods override CanParse to indicate when a Web request has a format they can parse.

See Also

GetContentFields (see page 15042)

GetFiles (see page 15042)

77.1.2.3.2 TAbstractContentParser.GetContentFields Method

Returns the named value strings that are included in the content of the current Web request.

Pascal

```
function GetContentFields: TStrings; virtual; abstract;
```

C++

```
virtual __fastcall TStrings GetContentFields() = 0;
```

Description

GetContentFields returns a list of strings with the form "Name=Value" that represent any name/value pairs that the parser identifies in the content of the current Web request.

In TAbstractContentParser (see page 15039), GetContentFields is abstract or, in C++ terminology, pure virtual, meaning that this class does not supply an implementation. Descendant classes override this method to extract name/value pairs from the current request and add them to the returned string list.

Note: The content parser owns the string list it returns. The caller should not free the string list.

See Also

ContentFields (see page 15177)

WebRequest (see page 15043)

ExtractContentFields (see page 15166)

77.1.2.3.3 TAbstractContentParser.GetFiles Method

Returns a set of uploaded files that the parser extracts from the current Web request.

Pascal

```
function GetFiles: TAbstractWebRequestFiles; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractWebRequestFiles GetFiles() = 0;
```

Description

GetFiles returns a list of Web request file objects, each of which represents an uploaded file from the current Web request. Each Web request file object has properties to describe an uploaded file, including a stream object from which to read the content of the uploaded file.

In TAbstractContentParser (see page 15039), GetFiles is abstract or, in C++ terminology, pure virtual, meaning that this class

does not supply an implementation. Descendant classes override this method to extract uploaded files from the current request and add them to the Web request files object.

Note: The content parser owns the list it returns. The caller should not free this list.

See Also

Files (see page 15182)

WebRequest (see page 15043)

TAbstractWebRequestFile (see page 15073)

77.1.2.4 TAbstractContentParser Properties

The properties of the TAbstractContentParser class are listed here.

Protected Properties

	Name	Description
	WebRequest (see page 15043)	Indicates the Web request object that uses this parser to parse its content.

77.1.2.4.1 TAbstractContentParser.WebRequest Property

Indicates the Web request object that uses this parser to parse its content.

Pascal

```
property WebRequest: TWebRequest;
```

C++

```
__property TWebRequest WebRequest;
```

Description

Content parsers use the protected WebRequest property internally to access the content they parse. The value of WebRequest is a Web request object whose properties represent the headers and content of the current HTTP request message.

See Also

Content (see page 15176)

77.1.3 TAbstractWebModuleFactory Class

TAbstractWebModuleFactory is the base class for all Web module factory objects.

Class Hierarchy

HTTPApp.TAbstractWebModuleFactory

Pascal

```
TAbstractWebModuleFactory = class;
```

C++

```
class TAbstractWebModuleFactory;
```

File

HTTPApp

Description

All WebSnap modules are registered with the WebSnap application using a factory object. In Delphi, the factory object is typically created in the initialization section of the module source file. The factory object is responsible for creating Web module instance. Applications can set flags for each factory object to indicate when it should create instances of its Web module and whether it can cache Web module instance for reuse with subsequent request messages.

Do not create instances of TAbstractWebModuleFactory. Most of its methods are abstract or, in C++ terminology, pure virtual, meaning they have no implementation. Instead, create instances of a TAbstractWebModuleFactory descendant such as TWebDataModuleFactory, TWebAppDataModuleFactory, TWebPageModuleFactory, or TWebAppPageModuleFactory. Each of these classes includes an implementation of the abstract or pure virtual methods, and represents group of Web module factories. These groups are based on whether the Web module factory generates the single application Web module (which contains the components to dispatch HTTP request messages), and it generates a Web module that creates a Web page.

See Also

TWebAppPageModuleFactory (see page 25012)

TWebAppDataModuleFactory (see page 25009)

TWebPageModuleFactory (see page 25019)

TWebRequestHandler (see page 25448)

TWebDataModuleFactory (see page 25014)

77.1.3.1 TAbstractWebModuleFactory Members

The following tables list the members exposed by TAbstractWebModuleFactory.

Protected Methods

	Name	Description
A	GetCacheMode (see page 15045)	Returns the value of the CacheMode (see page 15048) property.
A	GetComponentClass (see page 15045)	Returns the value of the ComponentClass (see page 15049) property.
A	GetCreateMode (see page 15046)	Returns the value of the CreateMode (see page 15049) property.
A	GetIsAppModule (see page 15046)	Returns the value of the IsAppModule (see page 15050) property.
A	GetModuleName (see page 15047)	Returns the value of the ModuleName (see page 15050) property.
V	GetWebPageInfo (see page 15047)	Returns the value of the WebPageInfo (see page 15050) property.

Public Methods

	Name	Description
A	GetModule (see page 15046)	Creates an instance of the associated Web module.
A	PreventDestruction (see page 15048)	Prevents the WebSnap application from freeing the Web module instance.

Public Properties

	Name	Description
R	CacheMode (see page 15048)	Indicates whether the factory can cache Web modules for reuse.
R	ComponentClass (see page 15049)	Specifies the class type of the Web module that this factory creates.
R	CreateMode (see page 15049)	Indicates when the factory creates instances of its Web module.
R	IsAppModule (see page 15050)	Indicates whether the corresponding Web module is the main Web application module.
R	ModuleName (see page 15050)	Specifies the name of the Web module class that this factory instantiates.
R	WebPageInfo (see page 15050)	Describes the attributes of the Web page that the corresponding Web module generates, if any.

77.1.3.2 TAbstractWebModuleFactory Methods

The methods of the TAbstractWebModuleFactory class are listed here.

Protected Methods

	Name	Description
	GetCacheMode (see page 15045)	Returns the value of the CacheMode (see page 15048) property.
	GetComponentClass (see page 15045)	Returns the value of the ComponentClass (see page 15049) property.
	GetCreateMode (see page 15046)	Returns the value of the CreateMode (see page 15049) property.
	GetIsAppModule (see page 15046)	Returns the value of the IsAppModule (see page 15050) property.
	GetModuleName (see page 15047)	Returns the value of the ModuleName (see page 15050) property.
	GetWebPageInfo (see page 15047)	Returns the value of the WebPageInfo (see page 15050) property.

Public Methods

	Name	Description
	GetModule (see page 15046)	Creates an instance of the associated Web module.
	PreventDestruction (see page 15048)	Prevents the WebSnap application from freeing the Web module instance.

77.1.3.2.1 TAbstractWebModuleFactory.GetCacheMode Method

Returns the value of the CacheMode (see page 15048) property.

Pascal

```
function GetCacheMode: TWebModuleCacheMode; virtual; abstract;
```

C++

```
virtual __fastcall TWebModuleCacheMode GetCacheMode() = 0;
```

Description

GetCacheMode is the protected read implementation of the CacheMode (see page 15048) property.

In TAbstractWebModuleFactory (see page 15043), GetCacheMode is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the CacheMode (see page 15048) property.

See Also

CacheMode (see page 15048)

77.1.3.2.2 TAbstractWebModuleFactory.GetComponentClass Method

Returns the value of the ComponentClass (see page 15049) property.

Pascal

```
function GetComponentClass: TComponentClass; virtual; abstract;
```

C++

```
virtual __fastcall TComponentClass GetComponentClass() = 0;
```

Description

GetComponentClass is the protected read implementation of the ComponentClass (see page 15049) property.

In TAbstractWebModuleFactory (see page 15043), GetComponentClass is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the

ComponentClass (see page 15049) property.

See Also

ComponentClass (see page 15049)

77.1.3.2.3 TAbstractWebModuleFactory.GetCreateMode Method

Returns the value of the CreateMode (see page 15049) property.

Pascal

```
function GetCreateMode: TWebModuleCreateMode; virtual; abstract;
```

C++

```
virtual __fastcall TWebModuleCreateMode GetCreateMode() = 0;
```

Description

GetCreateMode is the protected read implementation of the CreateMode (see page 15049) property.

In TAbstractWebModuleFactory (see page 15043), GetCreateMode is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the CreateMode (see page 15049) property.

See Also

CreateMode (see page 15049)

77.1.3.2.4 TAbstractWebModuleFactory.GetIsAppModule Method

Returns the value of the IsAppModule (see page 15050) property.

Pascal

```
function GetIsAppModule: Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean GetIsAppModule() = 0;
```

Description

GetIsAppModule is the protected read implementation of the IsAppModule (see page 15050) property.

In TAbstractWebModuleFactory (see page 15043), GetIsAppModule is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the IsAppModule (see page 15050) property.

See Also

IsAppModule (see page 15050)

77.1.3.2.5 TAbstractWebModuleFactory.GetModule Method

Creates an instance of the associated Web module.

Pascal

```
function GetModule: TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * GetModule() = 0;
```

Description

Call GetModule to obtain a new instance of the Web module that the factory creates. This method should only be called once for each request message that requires the use of the Web module.

GetModule returns the Web module instance.

In TAbstractWebModuleFactory (see page 15043), GetModule is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation that creates a new Web module instance.

See Also

ComponentClass (see page 15049)

CreateMode (see page 15049)

CacheMode (see page 15048)

77.1.3.2.6 TAbstractWebModuleFactory.GetModuleName Method

Returns the value of the ModuleName (see page 15050) property.

Pascal

```
function GetModuleName: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetModuleName() = 0;
```

Description

GetModuleName is the protected read implementation of the ModuleName (see page 15050) property.

In TAbstractWebModuleFactory (see page 15043), GetModuleName is abstract pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the ModuleName (see page 15050) property.

See Also

ModuleName (see page 15050)

77.1.3.2.7 TAbstractWebModuleFactory.GetWebPageInfo Method

Returns the value of the WebPageInfo (see page 15050) property.

Pascal

```
function GetWebPageInfo: TAbstractWebPageInfo; virtual;
```

C++

```
virtual __fastcall TAbstractWebPageInfo GetWebPageInfo();
```

Description

GetWebPageInfo is the protected read implementation of the WebPageInfo (see page 15050) property.

In TAbstractWebModuleFactory (see page 15043), GetWebPageInfo always returns nil (Delphi) or NULL (C++). Descendant classes that represent the factories for Web page modules override this method to return the Web page information object that describes the module's Web page.

See Also

WebPageInfo (see page 15050)

77.1.3.2.8 TAbstractWebModuleFactory.PreventDestruction Method

Prevents the WebSnap application from freeing the Web module instance.

Pascal

```
procedure PreventDestruction; virtual; abstract;
```

C++

```
virtual __fastcall PreventDestruction() = 0;
```

Description

Call PreventDestruction if the WebSnap application should not free the Web module instance when the current HTTP request message has been fully processed. For example, if another Web module contains a reference to the Web module that this factory creates, you may not want to allow the WebSnap application to free the Web module that this factory creates while the other module exists.

In TAbstractWebModuleFactory (see page 15043), PreventDestruction is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation that prevents the Web module from being freed.

See Also

CacheMode (see page 15048)

77.1.3.3 TAbstractWebModuleFactory Properties

The properties of the TAbstractWebModuleFactory class are listed here.

Public Properties

	Name	Description
R	CacheMode (see page 15048)	Indicates whether the factory can cache Web modules for reuse.
R	ComponentClass (see page 15049)	Specifies the class type of the Web module that this factory creates.
R	CreateMode (see page 15049)	Indicates when the factory creates instances of its Web module.
R	IsAppModule (see page 15050)	Indicates whether the corresponding Web module is the main Web application module.
R	ModuleName (see page 15050)	Specifies the name of the Web module class that this factory instantiates.
R	WebPageInfo (see page 15050)	Describes the attributes of the Web page that the corresponding Web module generates, if any.

77.1.3.3.1 TAbstractWebModuleFactory.CacheMode Property

Indicates whether the factory can cache Web modules for reuse.

Pascal

```
property CacheMode: TWebModuleCacheMode;
```

C++

```
__property TWebModuleCacheMode CacheMode;
```

Description

The Web module factory creates an instance of its Web module to handle a specific HTTP request message. After the WebSnap

application has finished with that request, it can free the Web module instance, or cache it so that the next time it needs to create an instance of its Web module, it can simply reuse the cached instance.

When CacheMode is caCache, the WebSnap application caches Web module instances for reuse. This can improve performance.

When CacheMode is caDestroy, the WebSnap application frees Web module instances when they are no longer needed for the current request message. This is the safest approach if changes to the Web module or any of the components it contains that are made while processing a request message can interfere with the correct handling of subsequent request messages.

See Also

CreateMode (see page 15049)

PreventDestruction (see page 15048)

77.1.3.3.2 TAbstractWebModuleFactory.ComponentClass Property

Specifies the class type of the Web module that this factory creates.

Pascal

```
property ComponentClass: TComponentClass;
```

C++

```
__property TComponentClass ComponentClass;
```

Description

ComponentClass is the class type of the Web module that the factory object creates. This is a descendant of TWebDataModule, TWebAppDataModule, TWebPageModule, or TWebAppPageModule. The WebSnap application wizard or Web module wizard generates the class declaration for the specific Web module class associated with a Web module factory.

See Also

ModuleName (see page 15050)

WebPageInfo (see page 15050)

IsAppModule (see page 15050)

77.1.3.3.3 TAbstractWebModuleFactory.CreateMode Property

Indicates when the factory creates instances of its Web module.

Pascal

```
property CreateMode: TWebModuleCreateMode;
```

C++

```
__property TWebModuleCreateMode CreateMode;
```

Description

Read CreateMode to determine whether the factory creates its Web module for every request message (crAlways), or when the WebSnap application specifically needs the Web module for the current request (crOnDemand).

Note: When the Web module factory creates the Web application module, CreateMode is always crAlways. The Web application module is always required because it contains the components that dispatch HTTP messages. For Web module factories that create any other Web modules in the application, CreateMode reflects a flag passed to the constructor.

See Also

CacheMode (see page 15048)

IsAppModule (see page 15050)

77.1.3.3.4 TAbstractWebModuleFactory.IsAppModule Property

Indicates whether the corresponding Web module is the main Web application module.

Pascal

```
property IsAppModule: Boolean;
```

C++

```
__property Boolean IsAppModule;
```

Description

WebSnap applications read IsAppModule to determine whether a Web module factory creates the single application module. WebSnap applications include one and only one application module, which holds the components responsible for dispatching HTTP requests. The Web application module is an instance of TWebAppPageModule or TWebAppDataModule.

See Also

TWebAppPageModule (see page 25394)

TWebAppDataModule (see page 25388)

ComponentClass (see page 15049)

TPageDispatcher (see page 24949)

77.1.3.3.5 TAbstractWebModuleFactory.ModuleName Property

Specifies the name of the Web module class that this factory instantiates.

Pascal

```
property ModuleName: string;
```

C++

```
__property AnsiString ModuleName;
```

Description

ModuleName is the name of the Web module class that the factory object instantiates. That is, this is the name of the class for which the ComponentClass (see page 15049) property is a class reference.

See Also

ComponentClass (see page 15049)

WebPageInfo (see page 15050)

IsAppModule (see page 15050)

FindModuleName (see page 15055)

77.1.3.3.6 TAbstractWebModuleFactory.WebPageInfo Property

Describes the attributes of the Web page that the corresponding Web module generates, if any.

Pascal

```
property WebPageInfo: TAbstractWebPageInfo;
```

C++

```
__property TAbstractWebPageInfo WebPageInfo;
```

Description

When the Web module that the factory object creates can generate a Web page, WebPageInfo describes the attributes of the Web page, such as its name, title, access rights, and so on. This information is stored in the Web module factory so that it is available even when there is no current instance of the Web module.

When the Web module that the factory object creates does not generate a Web page, WebPageInfo is nil (Delphi) or NULL (C++).

See Also

ComponentClass (see page 15049)

77.1.4 TAbstractWebModuleList Class

TAbstractWebModuleList lists the WebModules in a WebSnap application and provides access to their factory objects.

Class Hierarchy

Pascal

```
TAbstractWebModuleList = class(TObject);
```

C++

```
class TAbstractWebModuleList : public TObject;
```

File

HTTPApp

Description

TAbstractWebModuleList is the base class for TWebModuleList, which lists every Web module that has been instantiated to serve the current request. It introduces properties and methods for accessing the Web modules in the application, as well as for the Web module factory objects that create those Web module instances. All the methods in TAbstractWebModuleList are abstract or, in C++ terminology, pure virtual, meaning they have no implementation. TWebModuleList, a TAbstractWebModuleList descendant, overrides those abstract or pure virtual methods to provide a default implementation.

When a WebSnap application handles multiple requests concurrently, it includes multiple web modules lists, each on a separate thread and each associated with a different request.

The Web context, which can be accessed using the global WebContext function, provides access to the TAbstractWebModuleList object that is associated with the current request. Use the Web context's WebModules property to obtain a reference to the Web module list.

See Also

TWebModuleList (see page 25440)

TWebModuleFactoryList (see page 25436)

TAbstractWebModuleFactory (see page 15043)

WebContext

TAbstractWebContext (see page 24644)

77.1.4.1 TAbstractWebModuleList Members

The following tables list the members exposed by TAbstractWebModuleList.

Protected Methods

	Name	Description
	GetFactory (see page 15055)	This is GetFactory, a member of class TAbstractWebModuleList.
	GetFactoryCount (see page 15056)	
	GetItem (see page 15056)	This is GetItem, a member of class TAbstractWebModuleList.
	GetItemCount (see page 15056)	
	GetOnModuleAdded (see page 15056)	
	SetOnModuleAdded (see page 15057)	

Public Events

	Name	Description
	OnModuleAdded (see page 15059)	Occurs when a new Web module is added to the Items (see page 15058) property array.

Public Methods

	Name	Description
	AddModuleClass (see page 15053)	Creates a new Web module, given its class, and adds it to the Items (see page 15058) property array.
	AddModuleName (see page 15054)	Creates a new Web module, given its name, and adds it to the Items (see page 15058) property array.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindModuleClass (see page 15054)	Locates a Web module given its class reference.
	FindModuleName (see page 15055)	Locates a Web module given its name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Factory (see page 15057)	Lists the Web module factories for all the Web modules available in the application.
R	FactoryCount (see page 15058)	Indicates the number of Web module factories listed in the Factory (see page 15057) property array.
R	ItemCount (see page 15058)	Indicates the number of Web modules that have been instantiated in the current thread.
R	Items (see page 15058)	Lists the instantiated Web modules in the current thread.

77.1.4.2 TAbstractWebModuleList Methods

The methods of the TAbstractWebModuleList class are listed here.

Protected Methods

	Name	Description
A	GetFactory (see page 15055)	This is GetFactory, a member of class TAbstractWebModuleList.
A	GetFactoryCount (see page 15056)	
A	GetItem (see page 15056)	This is GetItem, a member of class TAbstractWebModuleList.
A	GetItemCount (see page 15056)	
A	GetOnModuleAdded (see page 15056)	
A	SetOnModuleAdded (see page 15057)	

Public Methods

	Name	Description
A	AddModuleClass (see page 15053)	Creates a new Web module, given its class, and adds it to the Items (see page 15058) property array.
A	AddModuleName (see page 15054)	Creates a new Web module, given its name, and adds it to the Items (see page 15058) property array.
A	FindModuleClass (see page 15054)	Locates a Web module given its class reference.
A	FindModuleName (see page 15055)	Locates a Web module given its name.

77.1.4.2.1 TAbstractWebModuleList.AddModuleClass Method

Creates a new Web module, given its class, and adds it to the Items (see page 15058) property array.

Pascal

```
function AddModuleClass(AClass: TComponentClass): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * AddModuleClass(TComponentClass AClass) = 0;
```

Description

AddModuleClass searches for a Web module factory that can create a Web module of the specified class. If it finds a match, it instantiates the Web module and adds it to the Items (see page 15058) property array.

AClass is a class reference to the Web module you want to instantiate. It should not match the class type of any Web module already listed in the Items (see page 15058) property array.

AddModuleClass returns the new Web module instance. If the Factory (see page 15057) property array does not include a Web module factory that can create a Web module of the specified class, AddModuleClass returns nil (Delphi) or NULL (C++).

Note: In TAbstractWebModuleList (see page 15051), AddModuleClass is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes override this method to provide an implementation.

See Also

AddModuleName (see page 15054)

Items (see page 15058)

Factory (see page 15057)

FindModuleClass (see page 15054)

77.1.4.2.2 TAbstractWebModuleList.AddModuleName Method

Creates a new Web module, given its name, and adds it to the Items (see page 15058) property array.

Pascal

```
function AddModuleName(const AClass: string): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * AddModuleName(const AnsiString AClass) = 0;
```

Description

AddModuleName searches for a Web module factory that can create a Web module of the specified name. If it finds a match, it instantiates the Web module and adds it to the Items (see page 15058) property array.

AName is the name of the Web module you want to instantiate. It should not match the name of any Web module already listed in the Items (see page 15058) property array.

AddModuleClass (see page 15053) returns the new Web module instance. If the Factory (see page 15057) property array does not include a Web module factory that can create a Web module of the specified name, AddModuleClass (see page 15053) returns nil (Delphi) or NULL (C++).

Note: In TAbstractWebModuleList (see page 15051), AddModuleName is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes override this method to provide an implementation.

See Also

AddModuleClass (see page 15053)

Items (see page 15058)

Factory (see page 15057)

FindModuleName (see page 15055)

77.1.4.2.3 TAbstractWebModuleList.FindModuleClass Method

Locates a Web module given its class reference.

Pascal

```
function FindModuleClass(AClass: TComponentClass): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * FindModuleClass(TComponentClass AClass) = 0;
```

Description

Call FindModuleClass to locate the Web module instance of a given class type. FindModuleClass searches the Items (see page 15058) property array for a Web module instance of the specified class.

AClass is a class reference to the class type of the desired Web module.

FindModuleClass returns the requested Web module instance if it exists. If there is no Web module of the specified class, FindModuleClass returns nil (Delphi) or NULL (C++).

Note: In TAbstractWebModuleList (see page 15051), FindModuleClass is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes override this method to provide an implementation.

See Also

FindModuleName (see page 15055)

Items (see page 15058)

77.1.4.2.4 TAbstractWebModuleList.FindModuleName Method

Locates a Web module given its name.

Pascal

```
function FindModuleName(const AClass: string): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * FindModuleName(const AnsiString AClass) = 0;
```

Description

Call FindModuleName to locate the Web module instance of a given name. FindModuleClass (see page 15054) searches the Items (see page 15058) property array for a Web module instance of the specified name.

AName is the name of the desired Web module.

FindModuleName returns the requested Web module instance if it exists. If there is no Web module with the specified name, FindModuleClass (see page 15054) returns nil (Delphi) or NULL (C++).

Note: In TAbstractWebModuleList (see page 15051), FindModuleClass (see page 15054) is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes override this method to provide an implementation.

See Also

FindModuleClass (see page 15054)

Items (see page 15058)

77.1.4.2.5 TAbstractWebModuleList.GetFactory Method

Pascal

```
function GetFactory(I: Integer): TAbstractWebModuleFactory; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractWebModuleFactory GetFactory(int I) = 0;
```

Description

This is GetFactory, a member of class TAbstractWebModuleList.

77.1.4.2.6 TAbstractWebModuleList.GetFactoryCount Method

Pascal

```
function GetFactoryCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetFactoryCount() = 0;
```

See Also

GetFactory (see page 15055)

FactoryCount (see page 15058)

77.1.4.2.7 TAbstractWebModuleList.GetItem Method

Pascal

```
function GetItem(I: Integer): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * GetItem(int I) = 0;
```

Description

This is GetItem, a member of class TAbstractWebModuleList.

77.1.4.2.8 TAbstractWebModuleList.GetItemCount Method

Pascal

```
function GetItemCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetItemCount() = 0;
```

See Also

GetItem (see page 15056)

ItemCount (see page 15058)

77.1.4.2.9 TAbstractWebModuleList.GetOnModuleAdded Method

Pascal

```
function GetOnModuleAdded: TModuleAddedProc; virtual; abstract;
```

C++

```
virtual __fastcall TModuleAddedProc GetOnModuleAdded() = 0;
```

See Also

OnModuleAdded (see page 15059)

SetOnModuleAdded (see page 15057)

77.1.4.2.10 TAbstractWebModuleList.SetOnModuleAdded Method

Pascal

```
procedure SetOnModuleAdded(AProc: TModuleAddedProc); virtual; abstract;
```

C++

```
virtual __fastcall SetOnModuleAdded(TModuleAddedProc AProc) = 0;
```

See Also

GetOnModuleAdded (see page 15056)

OnModuleAdded (see page 15059)

77.1.4.3 TAbstractWebModuleList Properties

The properties of the TAbstractWebModuleList class are listed here.

Public Properties

	Name	Description
R	Factory (see page 15057)	Lists the Web module factories for all the Web modules available in the application.
R	FactoryCount (see page 15058)	Indicates the number of Web module factories listed in the Factory (see page 15057) property array.
R	ItemCount (see page 15058)	Indicates the number of Web modules that have been instantiated in the current thread.
R	Items (see page 15058)	Lists the instantiated Web modules in the current thread.

77.1.4.3.1 TAbstractWebModuleList.Factory Property

Lists the Web module factories for all the Web modules available in the application.

Pascal

```
property Factory [I: Integer]: TAbstractWebModuleFactory;
```

C++

```
__property TAbstractWebModuleFactory Factory[int I];
```

Description

Read Factory to access the factory objects that instantiate Web modules for the WebSnap application. Each factory object includes methods to create or (if an instance exists) access the corresponding Web module, as well as properties that provide information about the Web module it creates.

Use the FactoryCount (see page 15058) property in conjunction with Factory to iterate over all the Web module factory objects in the WebSnap application.

I is the index of the factory object to read, where 0 identifies the first factory, 1 identifies the second factory, and so on, until FactoryCount (see page 15058) – 1 identifies the last factory.

See Also

FactoryCount (see page 15058)

Items (see page 15058)

77.1.4.3.2 TAbstractWebModuleList.FactoryCount Property

Indicates the number of Web module factories listed in the Factory (see page 15057) property array.

Pascal

```
property FactoryCount: Integer;
```

C++

```
__property int FactoryCount;
```

Description

Read FactoryCount to determine the number of Web module factory object (and hence the potential number of Web modules) in the WebSnap application. FactoryCount provides an upper bound to the index of the Factory (see page 15057) property array.

See Also

Factory (see page 15057)

ItemCount (see page 15058)

77.1.4.3.3 TAbstractWebModuleList.ItemCount Property

Indicates the number of Web modules that have been instantiated in the current thread.

Pascal

```
property ItemCount: Integer;
```

C++

```
__property int ItemCount;
```

Description

Read ItemCount to determine the number of instantiated Web modules in the current thread of the WebSnap application. ItemCount provides an upper bound to the index of the Items (see page 15058) property array.

Because ItemCount counts only the instantiated Web modules, its value is always less than or equal to the value of FactoryCount (see page 15058), which gives the number of Web module factories.

See Also

Items (see page 15058)

FactoryCount (see page 15058)

77.1.4.3.4 TAbstractWebModuleList.Items Property

Lists the instantiated Web modules in the current thread.

Pascal

```
property Items [I: Integer]: TComponent;
```

C++

```
__property TComponent * Items[int I];
```

Description

Read Items to access the Web modules that have been created to process the current HTTP request.

Use the ItemCount (see page 15058) property in conjunction with Items to iterate over all the Web modules in the current

thread.

I is the index of the object to read, where 0 identifies the first Web module, 1 identifies the second Web module, and so on, until ItemCount (see page 15058) – 1 identifies the last Web module.

Note: In Delphi, Items is the default property of TAbstractWebModuleList (see page 15051). This means that the property name may be omitted. Thus the line

Note: MyModule := WebContext.WebModules.Items[I];

Note: can be written

Note: MyModule := WebContext.WebModules[I];

Note: In C++, a similar effect to the use of Items as a default property can be achieved using the [] operator.

See Also

ItemCount (see page 15058)

Factory (see page 15057)

operator_sb

77.1.4.4 TAbstractWebModuleList Events

The events of the TAbstractWebModuleList class are listed here.

Public Events

	Name	Description
	OnModuleAdded (see page 15059)	Occurs when a new Web module is added to the Items (see page 15058) property array.

77.1.4.4.1 TAbstractWebModuleList.OnModuleAdded Event

Occurs when a new Web module is added to the Items (see page 15058) property array.

Pascal

```
property OnModuleAdded: TModuleAddedProc;
```

C++

```
__property TModuleAddedProc OnModuleAdded;
```

Description

Write an OnModuleAdded event handler to take specific action when a Web module is instantiated in the current Web context. When the WebSnap application creates a new Web module instance, it generates an OnModuleAdded event immediately after adding the instance to the Items (see page 15058) property array.

Note: Do not write an OnModuleAdded event handler if the application contains a TWebAppComponents object. TWebAppComponents uses this event to initialize the Web modules when they are added.

See Also

Items (see page 15058)

AddModuleClass (see page 15053)

AddModuleName (see page 15054)

TWebAppComponents (see page 24958)

77.1.5 TAbstractWebPageInfo Class

TAbstractWebPageInfo represents the attributes of a Web page module.

Class Hierarchy

HTTPApp.TAbstractWebPageInfo

Pascal

```
TAbstractWebPageInfo = class;
```

C++

```
class TAbstractWebPageInfo;
```

File

HTTPApp

Description

TAbstractWebPageInfo is the abstract base class for Web page information objects. It introduces a number of properties that describe the attributes of a Web page module, including

PageName (see page 15068), which is the logical name for the Web page the module generates.

IsPublished (see page 15067), which indicates whether the WebSnap application automatically displays the page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 15068).

IsLoginRequired (see page 15066), which indicates whether the user must log in to access the Web page.

ViewAccess (see page 15069), which can be used to restrict access to the Web page.

PageTitle (see page 15069), which is the title that appears on the Web page.

PageFile (see page 15068), which identifies an associated HTML template that the Web page module uses to generate its Web page.

PageHREF (see page 15068), which identifies the HREF to use for references to the page that appear in hypertext links.

PageDescription (see page 15067), which describes the Web page.

Do not create instances of TAbstractWebPageInfo. It includes abstract or, in C++ terminology, pure virtual methods (methods that have no implementation) that must be overridden in a descendant class.

Web page information objects are used to set the attributes of a Web page module. An instance of a TAbstractWebPageInfo descendant, such as TWebPageInfo, is supplied to the constructor of the factory object that creates instances of the Web page module. Typically, the code to create this object and supply it to the Web page factory's constructor is generated by a Web page module wizard. You can change this generated code, which appears in the generated unit for the Web page module, to alter the properties of the Web page information object. For example, you may want to add a ViewAccess (see page 15069) string to allow your application to limit access to the Web page.

In a WebSnap application, you can access the Web page information object of any Web page module by calling the FindPageInfo method of the application's global Web context.

See Also

TWebPageInfo (see page 25016)

TAbstractWebContext (see page 24644)

TWebAppPageModuleFactory (see page 25012)

TWebPageModuleFactory (see page 25019)

WebContext

77.1.5.1 TAbstractWebPageInfo Members

The following tables list the members exposed by TAbstractWebPageInfo.

Protected Methods

	Name	Description
	GetFactory (see page 15062)	Returns the value of the Factory (see page 15066) property.
V	GetIsLoginRequired (see page 15062)	Returns the value of the IsLoginRequired (see page 15066) property.
V	GetIsPublished (see page 15062)	Returns the value of the IsPublished (see page 15067) property.
V	GetPageDescription (see page 15063)	Returns the value of the PageDescription (see page 15067) property.
A	GetPageFile (see page 15063)	Returns the value of the PageFile (see page 15068) property.
V	GetPageHREF (see page 15064)	Returns the value of the PageHREF (see page 15068) property.
A	GetPageName (see page 15064)	Returns the value of the PageName (see page 15068) property.
V	GetPageTitle (see page 15064)	Returns the value of the PageTitle (see page 15069) property.
V	GetViewAccess (see page 15065)	Returns the value of the ViewAccess (see page 15069) property.
V	SetFactory (see page 15065)	Sets the value of the Factory (see page 15066) property.

Public Properties

	Name	Description
	Factory (see page 15066)	Specifies the factory object that creates instances of the Web page module this object describes.
R	IsLoginRequired (see page 15066)	Indicates whether callers must log in before accessing the associated Web page.
R	IsPublished (see page 15067)	Indicates whether the WebSnap application automatically displays the associated page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 15068).
R	PageDescription (see page 15067)	Describes the Web page.
R	PageFile (see page 15068)	Identifies an associated file that the Web page module uses to generate its Web page.
R	PageHREF (see page 15068)	Identifies the HREF to use for references to the associated Web page that appear in hypertext links.
R	PageName (see page 15068)	Denotes the logical name for the Web page that the associated module generates
R	PageTitle (see page 15069)	Indicates the title that appears on the associated Web page.
R	ViewAccess (see page 15069)	Indicates the view access string that controls access to the associated Web page.

77.1.5.2 TAbstractWebPageInfo Methods

The methods of the TAbstractWebPageInfo class are listed here.

Protected Methods

	Name	Description
	GetFactory (see page 15062)	Returns the value of the Factory (see page 15066) property.

	GetIsLoginRequired (see page 15062)	Returns the value of the IsLoginRequired (see page 15066) property.
	GetIsPublished (see page 15062)	Returns the value of the IsPublished (see page 15067) property.
	GetPageDescription (see page 15063)	Returns the value of the PageDescription (see page 15067) property.
	GetPageFile (see page 15063)	Returns the value of the PageFile (see page 15068) property.
	GetPageHREF (see page 15064)	Returns the value of the PageHREF (see page 15068) property.
	GetPageName (see page 15064)	Returns the value of the PageName (see page 15068) property.
	GetPageTitle (see page 15064)	Returns the value of the PageTitle (see page 15069) property.
	GetViewAccess (see page 15065)	Returns the value of the ViewAccess (see page 15069) property.
	SetFactory (see page 15065)	Sets the value of the Factory (see page 15066) property.

77.1.5.2.1 TAbstractWebPageInfo.GetFactory Method

Returns the value of the Factory (see page 15066) property.

Pascal

```
function GetFactory: TAbstractWebPageModuleFactory;
```

C++

```
__fastcall TAbstractWebPageModuleFactory GetFactory();
```

Description

GetFactory is the protected read implementation of the Factory (see page 15066) property.

See Also

Factory (see page 15066)

SetFactory (see page 15065)

77.1.5.2.2 TAbstractWebPageInfo.GetIsLoginRequired Method

Returns the value of the IsLoginRequired (see page 15066) property.

Pascal

```
function GetIsLoginRequired: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetIsLoginRequired();
```

Description

GetIsLoginRequired is the protected read implementation of the IsLoginRequired (see page 15066) property.

In TAbstractWebPageInfo (see page 15060), GetIsLoginRequired always returns false. Descendant classes override this method to indicate when users must log in to access the associated Web page.

See Also

IsLoginRequired (see page 15066)

77.1.5.2.3 TAbstractWebPageInfo.GetIsPublished Method

Returns the value of the IsPublished (see page 15067) property.

Pascal

```
function GetIsPublished: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetIsPublished();
```

Description

GetIsPublished is the protected read implementation of the IsPublished (see page 15067) property.

In TAbstractWebPageInfo (see page 15060), GetIsPublished always returns false. Descendant classes override this method to indicate when the page dispatcher can automatically forward HTTP request messages that address the associated Web module.

See Also

IsPublished (see page 15067)

77.1.5.2.4 TAbstractWebPageInfo.GetPageDescription Method

Returns the value of the PageDescription (see page 15067) property.

Pascal

```
function GetPageDescription: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetPageDescription();
```

Description

GetPageDescription is the protected read implementation of the PageDescription (see page 15067) property.

In TAbstractWebPageInfo (see page 15060), GetPageDescription returns the value of the PageName (see page 15068) property. Descendant classes override this method to allow page descriptions that differ from the Web page's logical name.

See Also

PageDescription (see page 15067)

PageName (see page 15068)

77.1.5.2.5 TAbstractWebPageInfo.GetPageFile Method

Returns the value of the PageFile (see page 15068) property.

Pascal

```
function GetPageFile: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetPageFile() = 0;
```

Description

GetPageFile is the protected read implementation of the PageFile (see page 15068) property.

In TAbstractWebPageInfo (see page 15060), GetPageFile is abstract or, in C++ terminology, pure virtual. Descendant classes must override this method to provide an implementation for the PageFile (see page 15068) property.

See Also

PageFile (see page 15068)

77.1.5.2.6 TAbstractWebPageInfo.GetPageHREF Method

Returns the value of the PageHREF (see page 15068) property.

Pascal

```
function GetPageHREF: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetPageHREF();
```

Description

GetPageHREF is the protected read implementation of the PageHREF (see page 15068) property.

In TAbstractWebPageInfo (see page 15060), GetPageHREF returns the value of the PageName (see page 15068) property. Descendant classes override this method to allow HREF values that differ from the Web page's logical name.

See Also

PageHREF (see page 15068)

PageName (see page 15068)

77.1.5.2.7 TAbstractWebPageInfo.GetPageName Method

Returns the value of the PageName (see page 15068) property.

Pascal

```
function GetPageName: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetPageName() = 0;
```

Description

GetPageName is the protected read implementation of the PageName (see page 15068) property.

In TAbstractWebPageInfo (see page 15060), GetPageName is abstract or, in C++ terminology, pure virtual. Descendant classes must override this method to provide an implementation for the PageName (see page 15068) property.

See Also

PageName (see page 15068)

77.1.5.2.8 TAbstractWebPageInfo.GetPageTitle Method

Returns the value of the PageTitle (see page 15069) property.

Pascal

```
function GetPageTitle: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetPageTitle();
```

Description

GetPageTitle is the protected read implementation of the PageTitle (see page 15069) property.

In TAbstractWebPageInfo (see page 15060), GetPageTitle returns the value of the PageName (see page 15068) property. Descendant classes override this method to allow Web pages to have titles that differ from their logical name.

See Also

PageTitle (see page 15069)

PageName (see page 15068)

77.1.5.2.9 TAbstractWebPageInfo.GetViewAccess Method

Returns the value of the ViewAccess (see page 15069) property.

Pascal

```
function GetViewAccess: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetViewAccess();
```

Description

GetViewAccess is the protected read implementation of the ViewAccess (see page 15069) property.

In TAbstractWebPageInfo (see page 15060), GetViewAccess returns an empty string. Descendant classes override this method to allow Web pages to have view access strings.

See Also

ViewAccess (see page 15069)

77.1.5.2.10 TAbstractWebPageInfo.SetFactory Method

Sets the value of the Factory (see page 15066) property.

Pascal

```
procedure SetFactory(AFactory: TAbstractWebPageModuleFactory); virtual;
```

C++

```
virtual __fastcall SetFactory(TAbstractWebPageModuleFactory AFactory);
```

Description

SetFactory is the protected write implementation of the Factory (see page 15066) property.

AFactory is the Web page module factory object that instantiates the Web page module that this object describes.

See Also

Factory (see page 15066)

GetFactory (see page 15062)

77.1.5.3 TAbstractWebPageInfo Properties

The properties of the TAbstractWebPageInfo class are listed here.

Public Properties

	Name	Description
	Factory (see page 15066)	Specifies the factory object that creates instances of the Web page module this object describes.
R	IsLoginRequired (see page 15066)	Indicates whether callers must log in before accessing the associated Web page.

R	IsPublished (see page 15067)	Indicates whether the WebSnap application automatically displays the associated page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 15068).
R	PageDescription (see page 15067)	Describes the Web page.
R	PageFile (see page 15068)	Identifies an associated file that the Web page module uses to generate its Web page.
R	PageHREF (see page 15068)	Identifies the HREF to use for references to the associated Web page that appear in hypertext links.
R	PageName (see page 15068)	Denotes the logical name for the Web page that the associated module generates
R	PageTitle (see page 15069)	Indicates the title that appears on the associated Web page.
R	ViewAccess (see page 15069)	Indicates the view access string that controls access to the associated Web page.

77.1.5.3.1 TAbstractWebPageInfo.Factory Property

Specifies the factory object that creates instances of the Web page module this object describes.

Pascal

```
property Factory: TAbstractWebPageModuleFactory;
```

C++

```
__property TAbstractWebPageModuleFactory Factory;
```

Description

Factory indicates the factory object that creates instances of the associated Web page module. If the associated Web page module is also the Web application module, Factory is an instance of TWebAppPageModuleFactory. Otherwise, it is an instance of TWebPageModuleFactory.

This property is set by the factory object's constructor.

See Also

TWebAppPageModuleFactory (see page 25012)

TWebPageModuleFactory (see page 25019)

Create_d

Create_c

77.1.5.3.2 TAbstractWebPageInfo.IsLoginRequired Property

Indicates whether callers must log in before accessing the associated Web page.

Pascal

```
property IsLoginRequired: Boolean;
```

C++

```
__property Boolean IsLoginRequired;
```

Description

Read IsLoginRequired to determine whether users must log in before accessing the associated Web page. The page dispatcher component (TPageDispatcher) checks this value to determine whether it can forward requests to the associated Web page when the caller has not yet logged in. The application can overrule the IsLoginRequired property by supplying an OnIsLoginRequired event handler to the page dispatcher.

In TAbstractWebPageInfo (see page 15060), IsLoginRequired is always false. Descendant classes override this property, initializing it to a value that is passed to the constructor.

See Also

Create_c

Create_d

IsPublished (see page 15067)

ViewAccess (see page 15069)

OnIsLoginRequired (see page 24904)

TPageDispatcher (see page 24949)

77.1.5.3.3 TAbstractWebPageInfo.IsPublished Property

Indicates whether the WebSnap application automatically displays the associated page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 15068).

Pascal

```
property IsPublished: Boolean;
```

C++

```
__property Boolean IsPublished;
```

Description

Read IsPublished to determine whether the page dispatcher (TPageDispatcher) automatically calls on the associated Web page module when it receives HTTP request messages that address it by name. HTTP request messages address a Web page module when the pathinfo portion of their target URL matches the value of the PageName (see page 15068) property.

When IsPublished is false, the WebSnap application must explicitly retrieve the content that the Web page module generates in order to use the Web page module.

In TAbstractWebPageInfo (see page 15060), IsPublished is always false. Descendant classes override this property, initializing it to a value that is passed to the constructor.

See Also

PageName (see page 15068)

Create_d

Create_c

TPageDispatcher (see page 24949)

77.1.5.3.4 TAbstractWebPageInfo.PageDescription Property

Describes the Web page.

Pascal

```
property PageDescription: string;
```

C++

```
__property AnsiString PageDescription;
```

Description

Read PageDescription to obtain a brief description of the Web page that the associated Web page module generates. This property can be used, for example, to provide descriptions for a dynamically generated catalog of available Web pages.

In TAbstractWebPageInfo (see page 15060), PageDescription is the same as the value of the PageName (see page 15068) property.

See Also

PageName (see page 15068)

PageTitle (see page 15069)

77.1.5.3.5 TAbstractWebPageInfo.PageFile Property

Identifies an associated file that the Web page module uses to generate its Web page.

Pascal

```
property PageFile: string;
```

C++

```
__property AnsiString PageFile;
```

Description

Read PageFile to identify the file, if any, that the associated Web page module uses when generating its Web page. This can be the Web page that the Web page module returns, or it can be a template that the Web page module modifies to generate its final product.

In TAbstractWebPageInfo (see page 15060), the property read access method for PageFile is abstract pure virtual. Descendant classes must override the protected GetPageFile (see page 15063) method to provide an implementation for this property.

77.1.5.3.6 TAbstractWebPageInfo.PageHREF Property

Identifies the HREF to use for references to the associated Web page that appear in hypertext links.

Pascal

```
property PageHREF: string;
```

C++

```
__property AnsiString PageHREF;
```

Description

Read PageHREF to obtain the string that should be inserted in HTML code to create a hypertext link to the associated Web page.

In TAbstractWebPageInfo (see page 15060), PageHREF is the same as the value of the PageName (see page 15068) property.

See Also

PageName (see page 15068)

77.1.5.3.7 TAbstractWebPageInfo.PageName Property

Denotes the logical name for the Web page that the associated module generates

Pascal

```
property PageName: string;
```

C++

```
__property AnsiString PageName;
```

Description

Read PageName to determine the name of the associated Web page. This name is used to identify that page in server-side scripts, and as the pathinfo portion of the URL for the page when the IsPublished (see page 15067) property is true.

In TAbstractWebPageInfo (see page 15060), the property read access method for PageName is abstract pure virtual. Descendant classes must override the protected GetPageName (see page 15064) method to provide an implementation for this property.

See Also

PageTitle (see page 15069)

PageHREF (see page 15068)

IsPublished (see page 15067)

77.1.5.3.8 TAbstractWebPageInfo.PageTitle Property

Indicates the title that appears on the associated Web page.

Pascal

```
property PageTitle: string;
```

C++

```
__property AnsiString PageTitle;
```

Description

Read PageTitle to determine the title that appears on the associated Web page when viewed from a user's browser.

In TAbstractWebPageInfo (see page 15060), PageTitle is the same as the value of the PageName (see page 15068) property.

See Also

PageName (see page 15068)

77.1.5.3.9 TAbstractWebPageInfo.ViewAccess Property

Indicates the view access string that controls access to the associated Web page.

Pascal

```
property ViewAccess: string;
```

C++

```
__property AnsiString ViewAccess;
```

Description

Read ViewAccess to obtain the view access string that controls which users have access to the associated Web page. When a user tries to access the associated Web page, the page dispatcher checks the user's access rights against this string by calling the HasViewRights method of the end user adapter. Adapters in the Web page module check execute and modify rights against this string by calling the HasExecuteRights and HasModifyRights methods of the end user adapter.

In TAbstractWebPageInfo (see page 15060), ViewAccess is always an empty string. Descendant classes override this property, initializing it to a value that is passed to the constructor.

See Also

HasExecuteRights (see page 24675)

HasModifyRights (see page 24676)

HasViewRights (see page 24676)

77.1.6 TAbstractWebPageModuleFactory Class

TAbstractWebPageModuleFactory is the base class for Web module factory objects whose generated Web modules create Web pages.

Class Hierarchy

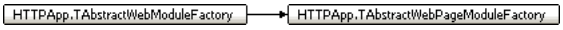

Pascal

```
TAbstractWebPageModuleFactory = class(TAbstractWebModuleFactory);
```

C++

```
class TAbstractWebPageModuleFactory : public TAbstractWebModuleFactory;
```

File

HTTPApp

Description

All WebSnap modules are registered with the WebSnap application using a factory object. In Delphi, the factory object is typically created in the initialization section of the module source file. The factory object is responsible for creating Web module instances. Applications can set flags for each factory object to indicate when it should create instances of its Web module and whether it can cache Web module instances for reuse with subsequent request messages.

TAbstractWebPageModuleFactory is the base class for all Web module factory objects that generate Web page modules. In addition to holding components that perform tasks for the WebSnap application, Web page modules can generate a Web page. The Web page is described by a Web page information object, which the Web page module factory makes available as its WebPageInfo (see page 15050) property.

Do not create instances of TAbstractWebPageModuleFactory. Most of its methods are abstract or, in C++ terminology, pure virtual, meaning they have no implementation. Instead, create instances of a TAbstractWebPageModuleFactory descendant such as TWebPageModuleFactory or TWebAppPageModuleFactory.

See Also

TWebAppPageModuleFactory (see page 25012)

TWebPageModuleFactory (see page 25019)

TWebRequestHandler (see page 25448)

TWebPageInfo (see page 25016)

77.1.6.1 TAbstractWebPageModuleFactory Members

The following tables list the members exposed by TAbstractWebPageModuleFactory.

Protected Methods

	Name	Description
	GetCacheMode (see page 15045)	Returns the value of the CacheMode (see page 15048) property.

	GetComponentClass (see page 15045)	Returns the value of the ComponentClass (see page 15049) property.
	GetCreateMode (see page 15046)	Returns the value of the CreateMode (see page 15049) property.
	GetIsAppModule (see page 15046)	Returns the value of the IsAppModule (see page 15050) property.
	GetModuleName (see page 15047)	Returns the value of the ModuleName (see page 15050) property.
	GetWebPageInfo (see page 15072)	Returns the value of the WebPageInfo (see page 15050) property.

Public Methods

	Name	Description
	Create (see page 15071)	Constructs an object and initializes its data before the object is first used.
	Destroy (see page 15071)	Disposes of an object instance.
	GetModule (see page 15046)	Creates an instance of the associated Web module.
	PreventDestruction (see page 15048)	Prevents the WebSnap application from freeing the Web module instance.

Public Properties

	Name	Description
	CacheMode (see page 15048)	Indicates whether the factory can cache Web modules for reuse.
	ComponentClass (see page 15049)	Specifies the class type of the Web module that this factory creates.
	CreateMode (see page 15049)	Indicates when the factory creates instances of its Web module.
	IsAppModule (see page 15050)	Indicates whether the corresponding Web module is the main Web application module.
	ModuleName (see page 15050)	Specifies the name of the Web module class that this factory instantiates.
	WebPageInfo (see page 15050)	Describes the attributes of the Web page that the corresponding Web module generates, if any.

77.1.6.2 TAbstractWebPageModuleFactory.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create(AWebPageInfo: TAbstractWebPageInfo);
```

C++

```
__fastcall TAbstractWebPageModuleFactory(TAbstractWebPageInfo AWebPageInfo);
```

Description

Do not create instances of TAbstractWebPageModuleFactory (see page 15070). Instead, construct descendants of TAbstractWebPageModuleFactory (see page 15070) using the new keyword, passing any arguments for the descendant object's constructor.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

Destroy (see page 22287)

new

77.1.6.3 TAbstractWebPageModuleFactory.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAbstractWebPageModuleFactory();
```

Description

Do not call ~TAbstractWebPageModuleFactory (see page 15070) directly. Instead, use the delete keyword when destroying VCL objects.

~TAbstractWebPageModuleFactory (see page 15070) deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

Warning: Never destroy a component from one of its own event handlers or from the event handler of a component it owns or contains. For example, don't destroy a button, or the form that owns the button, in the button's OnClick event handler.

To destroy a form, call its Release method, which destroys the form and releases the memory allocated for it after all its event handlers and those of the components it contains have executed.

See Also

delete

77.1.6.4 TAbstractWebPageModuleFactory Methods

The methods of the TAbstractWebPageModuleFactory class are listed here.

Protected Methods

	Name	Description
	GetWebPageInfo (see page 15072)	Returns the value of the WebPageInfo (see page 15050) property.

77.1.6.4.1 TAbstractWebPageModuleFactory.GetWebPageInfo Method

Returns the value of the WebPageInfo (see page 15050) property.

Pascal

```
function GetWebPageInfo: TAbstractWebPageInfo; override;
```

C++

```
virtual __fastcall TAbstractWebPageInfo GetWebPageInfo();
```

Description

GetWebPageInfo is the protected read implementation of the WebPageInfo (see page 15050) property. It returns the Web page information object that is supplied to the TAbstractWebPageModuleFactory (see page 15070) constructor.

See Also

WebPageInfo

Create_c

77.1.7 TAbstractWebRequestFile Class

TAbstractWebRequestFile describes an uploaded file from an HTTP request.

Class Hierarchy

Pascal

```
TAbstractWebRequestFile = class(TObject);
```

C++

```
class TAbstractWebRequestFile : public TObject;
```

File

HTTPApp

Description

TAbstractWebRequestFile introduces properties to describe and provide access to an uploaded file from an HTTP request. These include

FileName (see page 15077), which gives the name of the uploaded file.

FieldName (see page 15077), which gives the name of the field whose value is the uploaded file.

ContentType (see page 15076), which describes how the file data is encoded.

Stream (see page 15077), which is a stream object from which you can read the file content.

The methods that implement these properties are all abstract or, in C++ terminology, pure virtual. This means that the Web server application does not instantiate TAbstractWebRequestFile. Instead, it uses a TAbstractWebRequestFile descendant such as TWebRequestFile (which is included in WebSnap).

Each TAbstractWebRequestFile descendant describes a single uploaded file. A collection of all the uploaded files from the current request is available as the Files property of TWebRequest (see page 15161). In addition, in WebSnap applications, adapter fields that work with uploaded files surface those files using a TUpdateFileList object or IActionFieldValues interface.

Note: Support for uploaded files is only included in WebSnap. For other Web server applications, this class exists only as a placeholder for the Files property on TWebRequest (see page 15161).

See Also

TWebRequestFile (see page 18696)

TAbstractWebRequestFiles (see page 15078)

IActionFieldValues (see page 925)

TUpdateFileList (see page 24550)

TAdapterFileField (see page 23837)

77.1.7.1 TAbstractWebRequestFile Members

The following tables list the members exposed by TAbstractWebRequestFile.

Protected Methods

	Name	Description
	GetContentType (see page 15075)	Returns the value of the ContentType (see page 15076) property.
	GetFieldName (see page 15075)	Returns the value of the FieldName (see page 15077) property.
	GetFileName (see page 15075)	Returns the value of the FileName (see page 15077) property.
	GetStream (see page 15076)	Returns the value of the Stream (see page 15077) property.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	ContentType (see page 15076)	Describes how the file data is encoded.
	FieldName (see page 15077)	Indicates the name of the adapter field whose value is the contents of the file.
	FileName (see page 15077)	Indicates the name of the uploaded file.
	Stream (see page 15077)	Provides access to the contents of the uploaded file.

77.1.7.2 TAbstractWebRequestFile Methods

The methods of the TAbstractWebRequestFile class are listed here.

Protected Methods

	Name	Description
	GetContentType (see page 15075)	Returns the value of the ContentType (see page 15076) property.
	GetFieldName (see page 15075)	Returns the value of the FieldName (see page 15077) property.
	GetFileName (see page 15075)	Returns the value of the FileName (see page 15077) property.
	GetStream (see page 15076)	Returns the value of the Stream (see page 15077) property.

77.1.7.2.1 TAbstractWebRequestFile.GetContentType Method

Returns the value of the ContentType (see page 15076) property.

Pascal

```
function GetContentType: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetContentType() = 0;
```

Description

GetContentType is the protected read implementation for the ContentType (see page 15076) property.

In TAbstractWebRequestFile (see page 15073), GetContentType is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the ContentType (see page 15076) property.

See Also

ContentType (see page 15076)

77.1.7.2.2 TAbstractWebRequestFile.GetFieldName Method

Returns the value of the FieldName (see page 15077) property.

Pascal

```
function GetFieldName: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetFieldName() = 0;
```

Description

GetFieldName is the protected read implementation for the FieldName (see page 15077) property.

In TAbstractWebRequestFile (see page 15073), GetFieldName is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the FieldName (see page 15077) property.

See Also

FieldName (see page 15077)

77.1.7.2.3 TAbstractWebRequestFile.GetFileName Method

Returns the value of the FileName (see page 15077) property.

Pascal

```
function GetFileName: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetFileName() = 0;
```

Description

GetFileName is the protected read implementation for the FileName (see page 15077) property.

In TAbstractWebRequestFile (see page 15073), GetFileName is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the FileName (see page 15077) property.

See Also

FileName (see page 15077)

77.1.7.2.4 TAbstractWebRequestFile.GetStream Method

Returns the value of the Stream (see page 15077) property.

Pascal

```
function GetStream: TStream; virtual; abstract;
```

C++

```
virtual __fastcall TStream GetStream() = 0;
```

Description

GetStream is the protected read implementation for the Stream (see page 15077) property.

In TAbstractWebRequestFile (see page 15073), GetStream is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Stream (see page 15077) property.

See Also

Stream (see page 15077)

77.1.7.3 TAbstractWebRequestFile Properties

The properties of the TAbstractWebRequestFile class are listed here.

Public Properties

	Name	Description
R	ContentType (see page 15076)	Describes how the file data is encoded.
R	FieldName (see page 15077)	Indicates the name of the adapter field whose value is the contents of the file.
R	FileName (see page 15077)	Indicates the name of the uploaded file.
R	Stream (see page 15077)	Provides access to the contents of the uploaded file.

77.1.7.3.1 TAbstractWebRequestFile.ContentType Property

Describes how the file data is encoded.

Pascal

```
property ContentType: string;
```

C++

```
__property AnsiString ContentType;
```

Description

Read ContentType to determine the format of the data you can read from the Stream (see page 15077) property. ContentType is a string of the format Type/SubType, which names the MIME type and subtype used to encode the file contents.

See Also

Stream (see page 15077)

77.1.7.3.2 TAbstractWebRequestFile.FieldName Property

Indicates the name of the adapter field whose value is the contents of the file.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;
```

Description

Read FieldName to identify the adapter field for which this file was uploaded. This is the same as the FieldName property of the adapter field object.

See Also

FieldName (see page 23944)

FileName (see page 15077)

77.1.7.3.3 TAbstractWebRequestFile.FileName Property

Indicates the name of the uploaded file.

Pascal

```
property FileName: string;
```

C++

```
__property AnsiString FileName;
```

Description

Read FileName to obtain the name of the uploaded file.

See Also

Stream (see page 15077)

77.1.7.3.4 TAbstractWebRequestFile.Stream Property

Provides access to the contents of the uploaded file.

Pascal

```
property Stream: TStream;
```

C++

```
__property TStream Stream;
```

Description

Use Stream to read the content of the uploaded file. Call the Read method of this stream to read the file contents.

Note: The ContentType (see page 15076) property indicates the format used to encode the file contents that you read from Stream.

See Also

FileName (see page 15077)

ContentType (see page 15076)

Read (see page 2982)

77.1.8 TAbstractWebRequestFiles Class

TAbstractWebRequestFiles is a collection of TAbstractWebRequestFile (see page 15073) objects.

Class Hierarchy

Pascal

```
TAbstractWebRequestFiles = class(TObject);
```

C++

```
class TAbstractWebRequestFiles : public TObject;
```

File

HTTPApp

Description

TAbstractWebRequestFiles is the base class for objects that manage a set of Web request files objects (TAbstractWebRequestFile (see page 15073) descendants). Each Web request file object in the collection describes an uploaded file that was extracted from the current HTTP request object.

TAbstractWebRequestFiles should not be instantiated. The methods that implement its properties are abstract or, in C++ terminology, pure virtual. Instead, applications use a TAbstractWebRequestFiles descendant such as TWebRequestFiles (which is included in WebSnap).

Note: Support for uploaded files is only included in WebSnap. For other Web server applications, this class exists only as a placeholder for the Files property on TWebRequest (see page 15161).

See Also

TAbstractWebRequestFile (see page 15073)

TWebRequest (see page 15161)

TUpdateFileList (see page 24550)

TWebRequestFiles (see page 18700)

77.1.8.1 TAbstractWebRequestFiles Members

The following tables list the members exposed by TAbstractWebRequestFiles.

Protected Methods

	Name	Description
	GetCount (see page 15080)	Returns the value of the Count (see page 15081) property.
	GetItem (see page 15080)	Returns the value of the Items (see page 15081) property.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Count (see page 15081)	Indicates the number of uploaded files in the collection.
	Items (see page 15081)	Lists the uploaded files in the collection.

77.1.8.2 TAbstractWebRequestFiles Methods

The methods of the TAbstractWebRequestFiles class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 15080)	Returns the value of the Count (see page 15081) property.
	GetItem (see page 15080)	Returns the value of the Items (see page 15081) property.

77.1.8.2.1 TAbstractWebRequestFiles.GetCount Method

Returns the value of the Count (see page 15081) property.

Pascal

```
function GetCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetCount() = 0;
```

Description

GetCount is the protected read implementation of the Count (see page 15081) property.

In TAbstractWebRequestFile (see page 15073), GetCount is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Count (see page 15081) property.

See Also

Count (see page 15081)

77.1.8.2.2 TAbstractWebRequestFiles.GetItem Method

Returns the value of the Items (see page 15081) property.

Pascal

```
function GetItem(I: Integer): TAbstractWebRequestFile; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractWebRequestFile GetItem(int I) = 0;
```

Description

GetItem is the protected read implementation of the Items (see page 15081) property.

I is the index of the Web request file object to return.

In TAbstractWebRequestFile (see page 15073), GetCount (see page 15080) is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Count (see page 15081) property.

77.1.8.3 TAbstractWebRequestFiles Properties

The properties of the TAbstractWebRequestFiles class are listed here.

Public Properties

	Name	Description
	Count (see page 15081)	Indicates the number of uploaded files in the collection.
	Items (see page 15081)	Lists the uploaded files in the collection.

77.1.8.3.1 TAbstractWebRequestFiles.Count Property

Indicates the number of uploaded files in the collection.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read Count to determine the number of files in this collection. Count provides an upper bound to the index of the Items (see page 15081) property.

See Also

Items (see page 15081)

77.1.8.3.2 TAbstractWebRequestFiles.Items Property

Lists the uploaded files in the collection.

Pascal

```
property Items [I: Integer]: TAbstractWebRequestFile;
```

C++

```
__property TAbstractWebRequestFile Items[int I];
```

Description

Read Items to access the descriptions of the individual uploaded files in the collection.

I is the index of the file to access, where 0 indicates the first file, 1 indicates the second file, and so on. The value of I must be less than the value of the Count (see page 15081) property.

Note: Items is the default property of TAbstractWebRequestFiles (see page 15078). This means that the property name can be omitted when using this property. Thus, the line

Note: Upload := WebContext.Request.Files.Items[0];

Note: can also be written

Note: Upload := WebContext.Request.Files[0];

See Also

Count (see page 15081)

Files (see page 15182)

operator_sb

77.1.9 TAbstractWebSession Class

TAbstractWebSession is the base class for objects that represent a session in a WebSnap application.

Class Hierarchy

Pascal

```
TAbstractWebSession = class(TObject);
```

C++

```
class TAbstractWebSession : public TObject;
```

File

HTTPApp

Description

TAbstractWebSession is the base class for all Web session objects. These include active sessions, which are represented by TWebSession objects, and invalid sessions (TInvalidWebSession), which are used to represent sessions that can no longer be used because they have expired.

A Web session has an ID and a list of name/value pairs. The ID is a random number that uniquely identifies the session. The name/value pairs are used to associate information with the ID. The values are variants. You can use these named Variants (see page 23567) to store application-defined information that is associated with a single user. In this way, the information can persist over multiple calls from the same user.

The session ID is stored on the end user's computer using a cookie. When a request is received, the WebSnap application examines the cookie. If a session associated with this cookie exists, then it becomes the current session.

There is no need to instantiate session objects. A Session is created automatically when the WebSnap application receives an HTTP request message.

When the current session is terminated, its session ID and name/value pairs are discarded. For example, the current session is typically terminated when the user logs out. Sessions can also be terminated when they are not accessed within a specified time-out period, or when the application explicitly calls the session's Terminate (see page 15086) method. A new session is created after a session has been terminated.

See Also

TSessionsService (see page 25582)

TAbstractWebContext (see page 24644)

TWebSession (see page 24968)

77.1.9.1 TAbstractWebSession Members

The following tables list the members exposed by TAbstractWebSession.

Protected Methods

	Name	Description
	GetSessionID (see page 15084)	Returns the value of the SessionID (see page 15087) property.
	GetTimeoutMinutes (see page 15084)	Returns the value of the TimeoutMinutes (see page 15087) property.

	GetValue (see page 15084)	Returns the value associated with a specified name.
	SetTimeoutMinutes (see page 15085)	Sets the value of the TimoutMinutes property.
	SetValue (see page 15085)	Assigns a value to a specified name.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Terminate (see page 15086)	Terminates the session.
	UpdateResponse (see page 15086)	Adds the session ID to a cookie in a response message.

Public Properties

	Name	Description
	SessionID (see page 15087)	Uniquely identifies the Session.
	TimeoutMinutes (see page 15087)	Indicates the time, in minutes, that a session can be idle before it terminates itself.
	Values (see page 15088)	Lists the named values that the session stores.

77.1.9.2 TAbstractWebSession Methods

The methods of the TAbstractWebSession class are listed here.

Protected Methods

	Name	Description
	GetSessionID (see page 15084)	Returns the value of the SessionID (see page 15087) property.
	GetTimeoutMinutes (see page 15084)	Returns the value of the TimeoutMinutes (see page 15087) property.

	GetValue (see page 15084)	Returns the value associated with a specified name.
	SetTimeoutMinutes (see page 15085)	Sets the value of the TimoutMinutes property.
	SetValue (see page 15085)	Assigns a value to a specified name.

Public Methods

	Name	Description
	Terminate (see page 15086)	Terminates the session.
	UpdateResponse (see page 15086)	Adds the session ID to a cookie in a response message.

77.1.9.2.1 TAbstractWebSession.GetSessionID Method

Returns the value of the SessionID (see page 15087) property.

Pascal

```
function GetSessionID: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetSessionID() = 0;
```

Description

GetSessionID is the protected read implementation of the SessionID (see page 15087) property.

In TAbstractWebSession (see page 15082), GetSessionID is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the SessionID (see page 15087) property.

See Also

SessionID (see page 15087)

77.1.9.2.2 TAbstractWebSession.GetTimeoutMinutes Method

Returns the value of the TimeoutMinutes (see page 15087) property.

Pascal

```
function GetTimeoutMinutes: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetTimeoutMinutes() = 0;
```

Description

GetTimeoutMinutes is the protected read implementation of the TimeoutMinutes (see page 15087) property.

In TAbstractWebSession (see page 15082), GetTimeoutMinutes is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the TimeoutMinutes (see page 15087) property.

See Also

TimeoutMinutes (see page 15087)

SetTimeoutMinutes (see page 15085)

77.1.9.2.3 TAbstractWebSession.GetValue Method

Returns the value associated with a specified name.

Pascal

```
function GetValue(const AName: string): Variant; virtual; abstract;
```

C++

```
virtual __fastcall Variant GetValue(const AnsiString AName) = 0;
```

Description

GetValue is the protected read implementation of the Values (see page 15088) property.

AName is the name of the value to return.

In TAbstractWebSession (see page 15082), GetValue is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Values (see page 15088) property.

See Also

Values (see page 15088)

SetValue (see page 15085)

77.1.9.2.4 TAbstractWebSession.SetTimeoutMinutes Method

Sets the value of the TimoutMinutes property.

Pascal

```
procedure SetTimeoutMinutes(AValue: Integer); virtual; abstract;
```

C++

```
virtual __fastcall SetTimeoutMinutes(int AValue) = 0;
```

Description

SetTimeoutMinutes is the protected write implementation of the TimeoutMinutes (see page 15087) property.

AValue is the number of minutes that can elapse before the session terminates itself.

In TAbstractWebSession (see page 15082), SetTimeoutMinutes is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the TimeoutMinutes (see page 15087) property.

See Also

GetTimeoutMinutes (see page 15084)

TimeoutMinutes (see page 15087)

77.1.9.2.5 TAbstractWebSession.SetValue Method

Assigns a value to a specified name.

Pascal

```
procedure SetValue(const AName: string; const AValue: Variant); virtual; abstract;
```

C++

```
virtual __fastcall SetValue(const AnsiString AName, const Variant AValue) = 0;
```

Description

SetValue is the protected write implementation of the Value property.

AName is the name of the value to store.

AValue is the value to store under the specified name.

In TAbstractWebSession (see page 15082), SetValue is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Values (see page 15088) property.

See Also

GetValue (see page 15084)

Values (see page 15088)

77.1.9.2.6 TAbstractWebSession.Terminate Method

Terminates the session.

Pascal

```
procedure Terminate; virtual; abstract;
```

C++

```
virtual __fastcall Terminate() = 0;
```

Description

Call Terminate to explicitly terminate a session. Once a session terminates, the application can no longer access its Values (see page 15088).

Sessions are terminated automatically when the application does not receive any requests for the session for TimoutMinutes minutes.

In TAbstractWebSession (see page 15082), Terminate is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation.

See Also

TimeoutMinutes (see page 15087)

Values (see page 15088)

77.1.9.2.7 TAbstractWebSession.UpdateResponse Method

Adds the session ID to a cookie in a response message.

Pascal

```
procedure UpdateResponse(AResponse: TWebResponse); virtual; abstract;
```

C++

```
virtual __fastcall UpdateResponse(TWebResponse AResponse) = 0;
```

Description

Call UpdateResponse to add the value of the SessionID (see page 15087) property to one of the response message's cookies. This allows the user's next request message to use the Values (see page 15088) stored in this session.

AResponse is the Web response message to which a cookie with the session ID should be added.

See Also

SessionID (see page 15087)

Values (see page 15088)

77.1.9.3 TAbstractWebSession Properties

The properties of the TAbstractWebSession class are listed here.

Public Properties

	Name	Description
R	SessionID (see page 15087)	Uniquely identifies the Session.
	TimeoutMinutes (see page 15087)	Indicates the time, in minutes, that a session can be idle before it terminates itself.
	Values (see page 15088)	Lists the named values that the session stores.

77.1.9.3.1 TAbstractWebSession.SessionID Property

Uniquely identifies the Session.

Pascal

```
property SessionID: string;
```

C++

```
__property AnsiString SessionID;
```

Description

SessionID is an automatically-generated identifier that uniquely identifies the session. The UpdateResponse (see page 15086) method adds this identifier to response messages, using a cookie. When the WebSnap application receives a request containing a cookie with a session ID, the WebSnap application uses the ID to recreate a copy of the previous session object.

See Also

UpdateResponse (see page 15086)

77.1.9.3.2 TAbstractWebSession.TimeoutMinutes Property

Indicates the time, in minutes, that a session can be idle before it terminates itself.

Pascal

```
property TimeoutMinutes: Integer;
```

C++

```
__property int TimeoutMinutes;
```

Description

Set TimeoutMinutes to specify the number of minutes that can elapse before a session times out. If this number of minutes elapses with no request messages that call for the session, the session is terminated.

If you do not specify a value for TimeoutMinutes, the session times out after the default time-out period associated with the application's sessions service (TSessionsService).

See Also

Terminate (see page 15086)

DefaultTimeout (see page 25580)

77.1.9.3.3 TAbstractWebSession.Values Property

Lists the named values that the session stores.

Pascal

```
property Values [const AName: string]: Variant;
```

C++

```
__property Variant Values[AnsiString const AName];
```

Description

Use Values to get or set the information that the session stores. Because sessions are associated with individual users, this information typically represents information about the user or choices the user makes. The Values property can be used to implement application-specific tasks, such as saving user preferences or maintaining a list user selections.

Each value is indexed by a name that indicates its meaning. The names can be any application-defined string.

See Also

TEndUserSessionAdapter (see page 24458)

77.1.10 TContentParser Class

TContentParser is the default parser for extracting named value strings and uploaded files from an HTTP request.

Class Hierarchy

Pascal

```
TContentParser = class(TAbstractContentParser);
```

C++

```
class TContentParser : public TAbstractContentParser;
```

File

HTTPApp

Description

Web request objects use a TContentParser object to parse the content of an HTTP request message when there is no other registered content parser that knows how to interpret that content. The Web request uses the content parser to assign values for its ContentFields and Files properties.

TContentParser does not know how to parse special request formats. It simply calls the Web request object's own ExtractContentFields method to extract named value strings for the request's ContentFields property. It does not parse uploaded files, but merely returns an empty list for the request's Files property.

See Also

TWebRequest (see page 15161)

TWebRequest (see page 15161)

TWebRequest (see page 15161)

TMultipartContentParser (see page 18710)

TWebRequest (see page 15161)

77.1.10.1 TContentParser Members

The following tables list the members exposed by TContentParser.

Protected Properties

	Name	Description
R	WebRequest (see page 15043)	Indicates the Web request object that uses this parser to parse its content.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	CanParse (see page 15090)	Indicates whether a specified request message contains information encoded in a way that this parser understands.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 15041)	Creates and initializes a new TAbstractContentParser (see page 15039) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 15090)	Frees the memory associated with the TContentParser (see page 15088) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
V	GetContentFields (see page 15091)	Returns the named value strings that are included in the content of the current Web request.
V	GetFiles (see page 15091)	Returns a set of uploaded files that the parser extracts from the current Web request.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

77.1.10.2 TContentParser.Destroy Destructor

Frees the memory associated with the TContentParser (see page 15088) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TContentParser();
```

Description

Do not free content parser instances. The Web request object frees the content parser when it no longer needs it.

See Also

Free (see page 22300)

77.1.10.3 TContentParser Methods

The methods of the TContentParser class are listed here.

Public Methods

	Name	Description
	CanParse (see page 15090)	Indicates whether a specified request message contains information encoded in a way that this parser understands.
	GetContentFields (see page 15091)	Returns the named value strings that are included in the content of the current Web request.
	GetFiles (see page 15091)	Returns a set of uploaded files that the parser extracts from the current Web request.

77.1.10.3.1 TContentParser.CanParse Method

Indicates whether a specified request message contains information encoded in a way that this parser understands.

Pascal

```
class function CanParse(AWebRequest: TWebRequest): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanParse(TWebRequest AWebRequest);
```

Description

CanParse indicates whether a content parser class can handle parsing a specified request message. CanParse is a static class method, which means the Web request does not need an instantiated object to call this method. The Web request object calls the CanParse method of every registered content parser until it finds one that can parse its content.

AWebRequest is the current HTTP request message.

CanParse returns true if this class can parse AWebRequest, false if AWebRequest does not indicate an encoding that this class understands.

In TContentParser (see page 15088), CanParse always returns true. TContentParser (see page 15088) is always the last content parser that the Web request object checks, and provides the default parsing when no other parser is available.

See Also

GetContentFields (see page 15091)

GetFiles (see page 15091)

77.1.10.3.2 TContentParser.GetContentFields Method

Returns the named value strings that are included in the content of the current Web request.

Pascal

```
function GetContentFields: TStrings; override;
```

C++

```
virtual __fastcall TStrings GetContentFields();
```

Description

GetContentFields returns a list of strings with the form "Name=Value" that represent any name/value pairs that the parser identifies in the content of the current Web request.

In TContentParser (see page 15088), GetContentFields simply calls the ExtractContentFields method of the current Web request. This method assumes that fields are strings with the form "Name=Value", separated by ampersands (&).

Note: The content parser owns the string list it returns. The caller should not free the string list.

See Also

ContentFields (see page 15177)

ExtractContentFields (see page 15166)

77.1.10.3.3 TContentParser.GetFiles Method

Returns a set of uploaded files that the parser extracts from the current Web request.

Pascal

```
function GetFiles: TAbstractWebRequestFiles; override;
```

C++

```
virtual __fastcall TAbstractWebRequestFiles GetFiles();
```

Description

GetFiles returns a list of Web request file objects, each of which represents an uploaded file from the current Web request. Each Web request file object has properties to describe an uploaded file, including a stream object from which to read the content of the uploaded file.

In TContentParser (see page 15088), GetFiles always returns an empty list. That is, TContentParser (see page 15088) does not implement support for uploaded files in HTTP requests.

Note: The content parser owns the list it returns. The caller should not free this list.

See Also

Files (see page 15182)

TAbstractWebRequestFile (see page 15073)

77.1.11 TCookie Class

TCookie represents a Cookie header in an HTTP response message.

Class Hierarchy

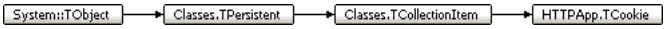

Pascal

```
TCookie = class(TCollectionItem);
```

C++

```
class TCookie : public TCollectionItem;
```

File

HTTPApp

Description

Use TCookie to specify the cookie header that should be sent with a response message when responding to an HTTP request message. Cookie headers contain state information that the Web client passes on to other locations on the Internet or Intranet.

See Also

TCookieCollection (see page 15098)

TWebResponse (see page 15191)

77.1.11.1 TCookie Members

The following tables list the members exposed by TCookie.

Protected Methods

	Name	Description
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
	GetHeaderValue (see page 15095)	This is GetHeaderValue, a member of class TCookie.
V	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
V	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
V	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
V	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	AssignTo (see page 15094)	Copies the properties of the TCookie (see page 15092) object to another TCookie (see page 15092) object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 15093)	Creates an instance of TCookie (see page 15092).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
	Domain (see page 15095)	Specifies the Domain field of the cookie header.
	Expires (see page 15095)	Specifies when the cookie is no longer valid.
R	HeaderValue (see page 15096)	Specifies the Cookie header string that represents this TCookie (see page 15092) object.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
	Name (see page 15096)	Specifies the name of the custom field represented by this Cookie header.
	Path (see page 15097)	Identifies the paths of URLs to which the Cookie should be sent.
	Secure (see page 15097)	Determines whether the Web client must use a secure connection to pass on the Cookie header.
	Value (see page 15097)	Specifies the value of the Cookie header field named by the Name (see page 15096) property.

77.1.11.2 TCookie.Create Constructor

Creates an instance of TCookie (see page 15092).

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TCookie(TCollection Collection);
```

Description

It is seldom necessary to call the constructor for TCookie (see page 15092) directly. Instead, create TCookie (see page 15092) objects indirectly by calling the Add method of TCookieCollection (see page 15098).

After calling the inherited constructor, Create initializes the Expires (see page 15095) property to indicate no expiration date.

See Also

Add (see page 15100)

Expires (see page 15095)

77.1.11.3 TCookie Methods

The methods of the TCookie class are listed here.

Protected Methods

	Name	Description
	GetHeaderValue (see page 15095)	This is GetHeaderValue, a member of class TCookie.

Public Methods

	Name	Description
	AssignTo (see page 15094)	Copies the properties of the TCookie (see page 15092) object to another TCookie (see page 15092) object.

77.1.11.3.1 TCookie.AssignTo Method

Copies the properties of the TCookie (see page 15092) object to another TCookie (see page 15092) object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Do not call the AssignTo method for TCookie (see page 15092). Instead, call the Assign method of the destination object. The Assign method of the destination object calls the AssignTo method if it does not contain code for copying the properties of a TCookie (see page 15092) object. If the destination object is a TCookie (see page 15092) object, AssignTo copies the Name (see page 15096), Value (see page 15097), Domain (see page 15095), Path (see page 15097), Expires (see page 15095), and Secure (see page 15097) properties so the destination cookie generates the same Cookie header string.

See Also

Assign (see page 2930)

77.1.11.3.2 TCookie.GetHeaderValue Method

Pascal

```
function GetHeaderValue: string;
```

C++

```
__fastcall AnsiString GetHeaderValue();
```

Description

This is GetHeaderValue, a member of class TCookie.

77.1.11.4 TCookie Properties

The properties of the TCookie class are listed here.

Public Properties

	Name	Description
	Domain (see page 15095)	Specifies the Domain field of the cookie header.
	Expires (see page 15095)	Specifies when the cookie is no longer valid.
R	HeaderValue (see page 15096)	Specifies the Cookie header string that represents this TCookie (see page 15092) object.
	Name (see page 15096)	Specifies the name of the custom field represented by this Cookie header.
	Path (see page 15097)	Identifies the paths of URLs to which the Cookie should be sent.
	Secure (see page 15097)	Determines whether the Web client must use a secure connection to pass on the Cookie header.
	Value (see page 15097)	Specifies the value of the Cookie header field named by the Name (see page 15096) property.

77.1.11.4.1 TCookie.Domain Property

Specifies the Domain field of the cookie header.

Pascal

```
property Domain: string;
```

C++

```
__property AnsiString Domain;
```

Description

Use Domain to limit the recipients of the cookie header to certain domains. The Web client will only send this cookie on to URIs in the specified domain. To limit the recipients to certain paths within the specified domain, use the Path (see page 15097) property.

See Also

Path (see page 15097)

77.1.11.4.2 TCookie.Expires Property

Specifies when the cookie is no longer valid.

Pascal

```
property Expires: TDateTime;
```

C++

```
__property TDateTime Expires;
```

Description

Set Expires to the date and time after which the Web client should not send the cookie on with HTTP messages.

See Also

Expires (see page 15212)

77.1.11.4.3 TCookie.HeaderValue Property

Specifies the Cookie header string that represents this TCookie (see page 15092) object.

Pascal

```
property HeaderValue: string;
```

C++

```
__property AnsiString HeaderValue;
```

Description

Read HeaderValue to determine the Cookie header that is added to the HTTP response message for this TCookie (see page 15092) object. HeaderValue combines the information specified by the Name (see page 15096) and Value (see page 15097) properties into a single Name (see page 15096)=Value (see page 15097) field, and then adds the Domain (see page 15095), Path (see page 15097), Expires (see page 15095), and Secure (see page 15097) fields based on the values of the properties with the corresponding names.

See Also

Name (see page 15096)

Value (see page 15097)

Domain (see page 15095)

Path (see page 15097)

Expires (see page 15095)

Secure (see page 15097)

77.1.11.4.4 TCookie.Name Property

Specifies the name of the custom field represented by this Cookie header.

Pascal

```
property Name: string;
```

C++

```
__property AnsiString Name;
```

Description

Set Name to the name that identifies the use of the Value (see page 15097) property in the Cookie header. The Value (see page 15097) property is combined with the Name property into a string of the form Name=Value (see page 15097). This string appears as one of the fields in the Cookie header represented by the HeaderValue (see page 15096) property.

See Also

HeaderValue (see page 15096)

Value (see page 15097)

77.1.11.4.5 TCookie.Path Property

Identifies the paths of URLs to which the Cookie should be sent.

Pascal

```
property Path: string;
```

C++

```
__property AnsiString Path;
```

Description

Use Path to limit the recipients of the cookie header to certain paths. The Web client will only send this cookie header on to URLs with the specified path. To limit the recipients to certain domains, use the Domain (see page 15095) property.

See Also

Domain (see page 15095)

PathInfo (see page 15185)

77.1.11.4.6 TCookie.Secure Property

Determines whether the Web client must use a secure connection to pass on the Cookie header.

Pascal

```
property Secure: Boolean;
```

C++

```
__property Boolean Secure;
```

Description

Set Secure to true if the Web client must use a secure connection when sending the Cookie header. Secure connections are connections where messages may not be stored on any permanent medium such as a back-up tape. If the message is stored temporarily in a file, the file must be deleted.

Set Secure to false if it does not matter whether the Cookie header is sent using a secure connection.

77.1.11.4.7 TCookie.Value Property

Specifies the value of the Cookie header field named by the Name (see page 15096) property.

Pascal

```
property Value: string;
```

C++

```
__property AnsiString Value;
```

Description

Set Value to the value associated with the field named by the Name (see page 15096) property. The Value property is combined with the Name (see page 15096) property into a string of the form Name (see page 15096)=Value. This string

appears as one of the fields in the Cookie header represented by the HeaderValue (see page 15096) property.

See Also

HeaderValue (see page 15096)

Name (see page 15096)

77.1.12 TCookieCollection Class

TCookieCollection is a collection of TCookie (see page 15092) objects.

Class Hierarchy

Pascal

```
TCookieCollection = class(TCollection);
```

C++

```
class TCookieCollection : public TCollection;
```

File

HTTPApp

Description

Use TCookieCollection to represent all the Cookie headers of an HTTP response message. Each Cookie header is represented by a single TCookie (see page 15092) object. A Cookie header includes a named field with an associated value, of the form Name=Value, plus a number of other, standard fields that specify the destination of the cookie, when the cookie is valid, and so on. Web clients pass Cookie headers on to the destinations specified by the fields of the Cookie header.

See Also

TCookie (see page 15092)

TWebResponse (see page 15191)

77.1.12.1 TCookieCollection Members

The following tables list the members exposed by TCookieCollection.

Protected Methods

	Name	Description
	Added (see page 2736)	Responds when items are added to the collection.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetCookie (see page 15101)	This is GetCookie, a member of class TCookieCollection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.

	GetOwner (see page 2932)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetCookie (see page 15101)	This is SetCookie, a member of class TCookieCollection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 15100)	Creates a new TCookie (see page 15092) object and adds it to the end of the Items (see page 15101) property array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 15100)	Creates and initializes a TCookieCollection (see page 15098) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 15101)	Provides indexed access to the TCookie (see page 15092) objects in the collection.
R	WebResponse (see page 15102)	Indicates the TWebResponse (see page 15191) object that uses the cookie collection to represent all of its Cookie headers.

77.1.12.2 TCookieCollection.Create Constructor

Creates and initializes a TCookieCollection (see page 15098) object.

Pascal

```
constructor Create(WebResponse: TWebResponse; ItemClass: TCollectionItemClass);
```

C++

```
__fastcall TCookieCollection(TWebResponse WebResponse, TCollectionItemClass ItemClass);
```

Description

Applications should not need to call the Create method directly. The Web response object that uses the cookie collection to represent its Cookie headers creates the TCookieCollection (see page 15098) object from its own constructor.

77.1.12.3 TCookieCollection Methods

The methods of the TCookieCollection class are listed here.

Protected Methods

	Name	Description
	GetCookie (see page 15101)	This is GetCookie, a member of class TCookieCollection.
	SetCookie (see page 15101)	This is SetCookie, a member of class TCookieCollection.

Public Methods

	Name	Description
	Add (see page 15100)	Creates a new TCookie (see page 15092) object and adds it to the end of the Items (see page 15101) property array.

77.1.12.3.1 TCookieCollection.Add Method

Creates a new TCookie (see page 15092) object and adds it to the end of the Items (see page 15101) property array.

Pascal

```
function Add: TCookie;
```

C++

```
__fastcall TCookie Add();
```

Description

Call Add to add a new Cookie header to the set of cookie headers sent with the Web response message. Use Add when adding a Cookie header whose fields will be set at a later time. If the values of the TCookie (see page 15092) properties are known at the time of creation, use the SetCookieField method of the Web response object instead.

See Also

WebResponse (see page 15102)

SetCookieField (see page 15203)

Items (see page 15101)

77.1.12.3.2 TCookieCollection.GetCookie Method

Pascal

```
function GetCookie(Index: Integer): TCookie;
```

C++

```
__fastcall TCookie GetCookie(int Index);
```

Description

This is GetCookie, a member of class TCookieCollection.

77.1.12.3.3 TCookieCollection.SetCookie Method

Pascal

```
procedure SetCookie(Index: Integer; Cookie: TCookie);
```

C++

```
__fastcall SetCookie(int Index, TCookie Cookie);
```

Description

This is SetCookie, a member of class TCookieCollection.

77.1.12.4 TCookieCollection Properties

The properties of the TCookieCollection class are listed here.

Public Properties

	Name	Description
	Items (see page 15101)	Provides indexed access to the TCookie (see page 15092) objects in the collection.
R	WebResponse (see page 15102)	Indicates the TWebResponse (see page 15191) object that uses the cookie collection to represent all of its Cookie headers.

77.1.12.4.1 TCookieCollection.Items Property

Provides indexed access to the TCookie (see page 15092) objects in the collection.

Pascal

```
property Items [Index: Integer]: TCookie;
```

C++

```
__property TCookie Items[int Index];
```

Description

Use Items to access the properties of the TCookie (see page 15092) objects maintained by TCookieCollection (see page 15098).

Note: In Delphi, items is the default property of TCookieCollection (see page 15098), so the name of the Items property can be left off when using this property. Thus the line

Note: FirstCookie := TApacheResponse1.Cookies.Items[0];

Note: can be written

Note: FirstCookie := TApacheResponse1.Cookies[0];

See Also

Cookies (see page 15210)

operator_sb

77.1.12.4.2 TCookieCollection.WebResponse Property

Indicates the TWebResponse (see page 15191) object that uses the cookie collection to represent all of its Cookie headers.

Pascal

```
property WebResponse: TWebResponse;
```

C++

```
__property TWebResponse WebResponse;
```

Description

Read WebResponse to access the response object that is using the cookie collection. The response object is a descendant of TWebResponse (see page 15191), either TISAPIResponse, TWinCGIResponse,TCGIResponse, or TApacheResponse, depending on the type of Web server application.

Use the properties of the response object to obtain information about the other headers in the response message or the content of the response message. Many applications use this information to determine what field values to specify as Cookie headers.

77.1.13 TCustomContentProducer Class

TCustomContentProducer is the base class for all objects that produce the string that is sent as the contents of an HTTP response message.

Class Hierarchy

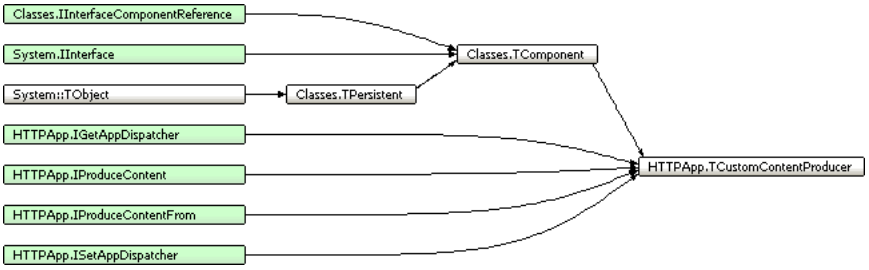

Pascal

```
TCustomContentProducer = class(TComponent, ISetAppDispatcher, IGetAppDispatcher,
IProduceContent, IProduceContentFrom);
```

C++

```
class TCustomContentProducer : public TComponent, public ISetAppDispatcher, public
IGetAppDispatcher, public IProduceContent, public IProduceContentFrom;
```

File

HTTPApp

Description

Use TCustomContentProducer as a base class when defining a class to produce the contents of an HTTP response message. In most cases, the content producer will assemble a string of HTML commands. However, descendants of TCustomContentProducer can be defined to produce any valid MIME type.

See Also

TDSTableProducer (see page 10290)

TPageProducer (see page 15309)

TWebResponse (see page 15191)

Generating the Content of Response Messages

The Web Module

77.1.13.1 TCustomContentProducer Members

The following tables list the members exposed by TCustomContentProducer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.

	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 15108)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Content (see page 15106)	Returns the content produced by the content producer.
V	ContentFromStream (see page 15107)	Returns the content produced in response to input read from a stream.
V	ContentFromString (see page 15107)	Returns the content produced in response to an input string.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.

V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

77.1.13.2 TCustomContentProducer Methods

The methods of the TCustomContentProducer class are listed here.

Protected Methods

	Name	Description
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
V	Notification (see page 15108)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.

Public Methods

	Name	Description
V	Content (see page 15106)	Returns the content produced by the content producer.
V	ContentFromStream (see page 15107)	Returns the content produced in response to input read from a stream.
V	ContentFromString (see page 15107)	Returns the content produced in response to an input string.

77.1.13.2.1 TCustomContentProducer.Content Method

Returns the content produced by the content producer.

Pascal

```
function Content: string; virtual;
```

C++

```
virtual __fastcall AnsiString Content();
```

Description

Call Content to retrieve the content for using in an HTTP response message. TCustomContentProducer (see page 15102) returns an empty string. Descendants of TCustomContentProducer (see page 15102) return the value of the content that they produce. In most cases, this content is a string of HTML commands, although it can be any valid MIME type.

See Also

ContentFromStream (see page 15107)

ContentFromString (see page 15107)

Content (see page 15207)

Responding to Request Messages with Action Items

77.1.13.2.2 TCustomContentProducer.ContentFromStream Method

Returns the content produced in response to input read from a stream.

Pascal

```
function ContentFromStream(Stream: TStream): string; virtual;
```

C++

```
virtual __fastcall AnsiString ContentFromStream(TStream Stream);
```

Description

Call ContentFromStream to retrieve the content produced by the content producer based on the information provided by the Stream parameter. TCustomContentProducer (see page 15102) ignores the Stream parameter and returns the value created by the Content (see page 15106) method.

Descendants of TCustomContentProducer (see page 15102) override this method to interpret streamed input in their own way.

See Also

Content (see page 15106)

ContentFromString (see page 15107)

Content (see page 15207)

77.1.13.2.3 TCustomContentProducer.ContentFromString Method

Returns the content produced in response to an input string.

Pascal

```
function ContentFromString(const S: string): string; virtual;
```

C++

```
virtual __fastcall AnsiString ContentFromString(const AnsiString S);
```

Description

Call ContentFromString to retrieve the content produced by the content producer based on the information provided by the S parameter. TCustomContentProducer (see page 15102) ignores the string and returns the value created by the Content (see page 15106) method.

Descendants of TCustomContentProducer (see page 15102) override this method to interpret the input string in their own way.

See Also

Content (see page 15106)

ContentFromStream (see page 15107)

Content (see page 15207)

77.1.13.2.4 TCustomContentProducer.GetAppDispatcher Method

Pascal

```
function GetAppDispatcher: TComponent;
```

C++

```
__fastcall TComponent * GetAppDispatcher();
```

Description

This is GetAppDispatcher, a member of class TCustomContentProducer.

77.1.13.2.5 TCustomContentProducer.Notification Method

Overrides the inherited method to respond to messages that the dispatcher is going away.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

After calling the inherited method, Notification checks whether this call was triggered by the removal of the dispatcher. If so, the Dispatcher (see page 15109) property is set to nil. NULL.

See Also

DispatcherComponent (see page 15110)

Dispatcher (see page 15109)

77.1.13.2.6 TCustomContentProducer.ProduceContent Method

Returns the content that the producer creates.

Pascal

```
function ProduceContent: string;
```

C++

```
__fastcall AnsiString ProduceContent();
```

Description

Call ProduceContent to retrieve the content for using in an HTTP response message. In most cases, this content is a string of HTML commands defining a Web page, although it can be any valid MIME type.

See Also

Content (see page 15106)

77.1.13.2.7 TCustomContentProducer.ProduceContentFromStream Method

Pascal

```
function ProduceContentFromStream(Stream: TStream): string;
```

C++

```
__fastcall AnsiString ProduceContentFromStream(TStream Stream);
```

Description

This is ProduceContentFromStream, a member of class TCustomContentProducer.

77.1.13.2.8 TCustomContentProducer.ProduceContentFromString Method

Pascal

```
function ProduceContentFromString(const S: string): string;
```

C++

```
__fastcall AnsiString ProduceContentFromString(const AnsiString S);
```

Description

This is ProduceContentFromString, a member of class TCustomContentProducer.

77.1.13.2.9 TCustomContentProducer.SetAppDispatcher Method

Pascal

```
procedure SetAppDispatcher(const ADispatcher: TComponent);
```

C++

```
__fastcall SetAppDispatcher(const TComponent * ADispatcher);
```

Description

This is SetAppDispatcher, a member of class TCustomContentProducer.

77.1.13.3 TCustomContentProducer Properties

The properties of the TCustomContentProducer class are listed here.

Public Properties

	Name	Description
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.

77.1.13.3.1 TCustomContentProducer.Dispatcher Property

Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.

Pascal

```
property Dispatcher: IWebDispatcherAccess;
```

C++

```
__property IWebDispatcherAccess Dispatcher;
```

Description

Read Dispatcher to access the interface of the dispatcher. In Web Broker applications, this is the interface of a TWebDispatcher (see page 15147) or TWebModule (see page 15154) component. In WebSnap applications, this is the interface of a TWebAppComponents component.

The IWebDispatcherAccess (see page 15234) interface has two methods:

Request, which returns the HTTP request message to which this component responds.

Response, which returns the HTTP response message for which this component generates content.

See Also

TWebModule (see page 15154)

TWebDispatcher (see page 15147)

TWebAppComponents (see page 24958)

TWebRequest (see page 15161)

TWebResponse (see page 15191)

DispatcherComponent (see page 15110)

77.1.13.3.2 TCustomContentProducer.DispatcherComponent Property

Provides access to the dispatcher component.

Pascal

```
property DispatcherComponent: TComponent;
```

C++

```
__property TComponent * DispatcherComponent;
```

Description

Dispatcher (see page 15109) component is the component that implements the interface that is the value of the Dispatcher (see page 15109) property. The content producer uses this property so that it can respond to notifications when the component is freed. To use the dispatcher, use the Dispatcher (see page 15109) property instead.

See Also

Dispatcher (see page 15109)

77.1.14 TCustomWebDispatcher Class

TCustomWebDispatcher is the abstract base class for Web dispatcher objects.

Class Hierarchy

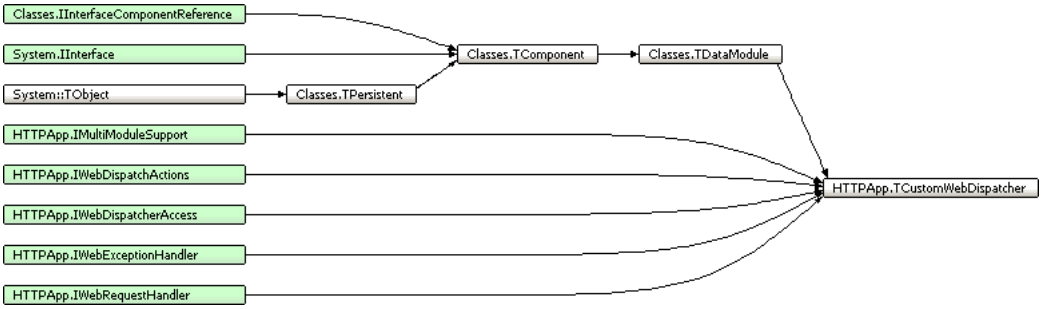

Pascal

```
TCustomWebDispatcher = class(TDataModule, IWebRequestHandler, IWebDispatchActions,
IWebDispatcherAccess, IMultiModuleSupport, IWebExceptionHandler);
```

C++

```
class TCustomWebDispatcher : public TDataModule, public IWebRequestHandler, public
IWebDispatchActions, public IWebDispatcherAccess, public IMultiModuleSupport, public
IWebExceptionHandler;
```

File

HTTPApp

Description

Add a descendant of TCustomWebDispatcher to a Web application to enable it to respond to HTTP request messages. When a Web application is created, it automatically includes a TWebModule (see page 15154) object, which is a descendant of TCustomWebDispatcher. Keep the default Web module, or replace it with another data module that contains a descendant of TCustomWebDispatcher, such as TWebDispatcher (see page 15147).

TCustomWebDispatcher manages the action items that assemble responses to HTTP request messages. When the TWebApplication object receives an HTTP request message, TCustomWebDispatcher passes the request on to the appropriate action item or action items to service the message.

TCustomWebDispatcher also keeps track of information about the current HTTP message for the benefit of TCustomContentProducer (see page 15102) objects.

See Also

TCustomContentProducer (see page 15102)

TWebActionItem (see page 15130)

TWebActionItems (see page 15140)

TWebApplication (see page 24618)

TWebDispatcher (see page 15147)

TWebModule (see page 15154)

TWebRequest (see page 15161)

TWebResponse (see page 15191)

The Web Module

77.1.14.1 TCustomWebDispatcher Members

The following tables list the members exposed by TCustomWebDispatcher.

Protected Events

	Name	Description
	AfterDispatch (see page 15123)	Occurs after the HTTP response message has been successfully filled out by the action items but not yet sent.
	BeforeDispatch (see page 15124)	Occurs before the dispatcher tries to match the HTTP request message with any of its action items.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	Access_Request (see page 15116)	This is Access_Request, a member of class TCustomWebDispatcher.
	Access_Response (see page 15117)	This is Access_Response, a member of class TCustomWebDispatcher.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DispatchAction (see page 15117)	Passes an HTTP request message to the action items in Actions (see page 15122) until it is handled.
	DoAfterDispatch (see page 15118)	Generates an AfterDispatch (see page 15123) event and returns a value that indicates whether the Web application can send the response.
	DoBeforeDispatch (see page 15119)	Generates a BeforeDispatch (see page 15124) event and returns a value that indicates whether the dispatcher should look for action items to handle the request.
V	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
V	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
	FinishContext (see page 15119)	This is FinishContext, a member of class TCustomWebDispatcher.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.
	HandleException (see page 15119)	Provides default handling of exceptions for the web application.
	HandleRequest (see page 15120)	Processes an HTTP request message.
	InitContext (see page 15120)	This is InitContext, a member of class TCustomWebDispatcher.
	InitModule (see page 15120)	This is InitModule, a member of class TCustomWebDispatcher.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 15121)	Responds to notifications when objects are added to the module.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnException (see page 15124)	Occurs when an unhandled exception occurs in the application.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActionByName (see page 15117)	Returns the action item identified by the AName parameter.
V	AfterConstruction (see page 2829)	Generates an OnCreate (see page 2833) event.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 2830)	Generates an OnDestroy (see page 2834) event.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 15115)	Creates an instance of TCustomWebDispatcher (see page 15110).
V	CreateNew (see page 2828)	Creates an instance of a data module and registers with the global screen object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 15115)	Frees an instance of TCustomWebDispatcher (see page 15110).
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	Action (see page 15121)	Provides indexed access to the action items managed by the TCustomWebDispatcher (see page 15110) object.
	Actions (see page 15122)	Represents the collection of action items managed by the TCustomWebDispatcher (see page 15110) object.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.

	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Request (see page 15122)	Provides access to the TWebRequest (see page 15161) object that the dispatcher is currently handling.
R	Response (see page 15122)	Provides access to the TWebResponse (see page 15191) object for the message the dispatcher is currently handling.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.
	Tag (see page 2808)	Stores an integer value as part of a component.

77.1.14.2 TCustomWebDispatcher.Create Constructor

Creates an instance of TCustomWebDispatcher (see page 15110).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomWebDispatcher(TComponent * AOwner);
```

Description

Do not call the constructor for TCustomWebDispatcher (see page 15110). Web applications automatically include a Web module. If the Web module is replaced by another data model, the Web dispatcher should be added to the new data module at design time. Objects placed at design time are created automatically. If an application tries to create a dispatcher object in a Web module, or in a data module that already has a Web dispatcher, an exception is raised.

Create creates the TWebActionItems (see page 15140) object that holds the action items managed by the dispatcher. If the Web dispatcher is included in another data module, it adds a reference to itself to any content producer in that data module.

After the Web dispatcher is created automatically at runtime, an OnCreate event occurs.

See Also

Actions (see page 15122)

OnCreate (see page 2833)

Dispatcher (see page 15109)

77.1.14.3 TCustomWebDispatcher.Destroy Destructor

Frees an instance of TCustomWebDispatcher (see page 15110).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWebDispatcher();
```

Description

Do not destroy the TCustomWebDispatcher (see page 15110) directly in an application. The dispatcher is freed automatically when the server application shuts down.

Destroy frees the TWebActionItems (see page 15140) object used to hold the collection of action items.

See Also

Actions (see page 15122)

Free (see page 22300)

77.1.14.4 TCustomWebDispatcher Methods

The methods of the TCustomWebDispatcher class are listed here.

Protected Methods

	Name	Description
	Access_Request (see page 15116)	This is Access_Request, a member of class TCustomWebDispatcher.
	Access_Response (see page 15117)	This is Access_Response, a member of class TCustomWebDispatcher.
	DispatchAction (see page 15117)	Passes an HTTP request message to the action items in Actions (see page 15122) until it is handled.
	DoAfterDispatch (see page 15118)	Generates an AfterDispatch (see page 15123) event and returns a value that indicates whether the Web application can send the response.
	DoBeforeDispatch (see page 15119)	Generates a BeforeDispatch (see page 15124) event and returns a value that indicates whether the dispatcher should look for action items to handle the request.
	FinishContext (see page 15119)	This is FinishContext, a member of class TCustomWebDispatcher.
	HandleException (see page 15119)	Provides default handling of exceptions for the web application.
	HandleRequest (see page 15120)	Processes an HTTP request message.
	InitContext (see page 15120)	This is InitContext, a member of class TCustomWebDispatcher.
	InitModule (see page 15120)	This is InitModule, a member of class TCustomWebDispatcher.
V	Notification (see page 15121)	Responds to notifications when objects are added to the module.

Public Methods

	Name	Description
	ActionByName (see page 15117)	Returns the action item identified by the AName parameter.

77.1.14.4.1 TCustomWebDispatcher.Access_Request Method

Pascal

```
function Access_Request: TWebRequest;
```

C++

```
__fastcall TWebRequest Access_Request();
```

Description

This is Access_Request, a member of class TCustomWebDispatcher.

77.1.14.4.2 TCustomWebDispatcher.Access_Response Method

Pascal

```
function Access_Response: TWebResponse;
```

C++

```
__fastcall TWebResponse Access_Response();
```

Description

This is Access_Response, a member of class TCustomWebDispatcher.

77.1.14.4.3 TCustomWebDispatcher.ActionByName Method

Returns the action item identified by the AName parameter.

Pascal

```
function ActionByName(const AName: string): TWebActionItem;
```

C++

```
__fastcall TWebActionItem ActionByName(const AnsiString AName);
```

Description

Call ActionByName to gain access to an individual action item when its name is known. Change the properties of the action item so that it handles different HTTP request messages, or invoke its OnAction event handler to respond to an HTTP request message.

See Also

Action (see page 15121)

Actions (see page 15122)

TWebActionItem (see page 15130)

Name (see page 15137)

77.1.14.4.4 TCustomWebDispatcher.DispatchAction Method

Passes an HTTP request message to the action items in Actions (see page 15122) until it is handled.

Pascal

```
function DispatchAction(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean DispatchAction(TWebRequest Request, TWebResponse Response);
```

Description

The Web application calls DispatchAction when it receives an HTTP request message so that the dispatcher can dispatch the request to any action items that can generate the response.

Before DispatchAction looks for action items to handle the HTTP request message represented by the Request (see page 15122) parameter, it generates a BeforeDispatch (see page 15124) event. If the BeforeDispatch (see page 15124) event handler handles the action, the dispatcher does not pass the request message to any of the action items.

If there is no BeforeDispatch (see page 15124) event handler, or of the BeforeDispatch (see page 15124) event handler does not handle the request, DispatchAction attempts to match the MethodType and PathInfo of the request with the

MethodType and PathInfo properties of each action item in the Actions (see page 15122) property. When an action item matches the MethodType and PathInfo of the request, DispatchAction calls its OnAction event handler.

The OnAction event handler can complete the response to the request, or indicate that the request is not fully handled and allow another action item to finish processing the request. If the OnAction event handler does not complete the request, DispatchAction continues checking the action items in the Actions (see page 15122) property and calling the OnAction event handler when a match is found.

If all of the action items are checked, and the request is still not fully handled, DispatchAction calls the OnAction event handler of the default action item.

Finally, if the request has been handled by the BeforeDispatch (see page 15124) event handler or any of the action items, DispatchAction generates an AfterDispatch (see page 15123) event. The AfterDispatch (see page 15123) event does not occur if the BeforeDispatch (see page 15124) event or any of the action items has already sent the response.

DispatchAction returns a Boolean value indicating whether the Web application should send the response message if it is not already sent. DispatchAction returns true if the action was handled by the action items and the AfterDispatch (see page 15123) event handler did not change its Handled parameter to false.

See Also

Actions (see page 15122)

AfterDispatch (see page 15123)

BeforeDispatch (see page 15124)

TWebActionItem (see page 15130)

OnAction (see page 15139)

HandleRequest (see page 25454)

TWebRequest (see page 15161)

TWebResponse (see page 15191)

77.1.14.4.5 TCustomWebDispatcher.DoAfterDispatch Method

Generates an AfterDispatch (see page 15123) event and returns a value that indicates whether the Web application can send the response.

Pascal

```
function DoAfterDispatch(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean DoAfterDispatch(TWebRequest Request, TWebResponse Response);
```

Description

Applications cannot call DoAfterDispatch. The DispatchAction (see page 15117) method calls DoAfterDispatch after sending the HTTP request message to any matching action items. The DispatchAction (see page 15117) method does not call DoAfterDispatch if the HTTP request message was not handled by the action items or by a BeforeDispatch (see page 15124) event handler. DoAfterDispatch is also not called if the message was already sent back to the Web client.

If the AfterDispatch (see page 15123) event handler indicates that the response should not be sent, DoAfterDispatch returns false, and the Web application drops the connection without sending the response message. Otherwise, DoAfterDispatch returns true, and the Web application sends the response message if it was not sent from within the AfterDispatch (see page 15123) event handler.

See Also

AfterDispatch (see page 15123)

BeforeDispatch (see page 15124)

DispatchAction (see page 15117)

77.1.14.4.6 TCustomWebDispatcher.DoBeforeDispatch Method

Generates a BeforeDispatch (see page 15124) event and returns a value that indicates whether the dispatcher should look for action items to handle the request.

Pascal

```
function DoBeforeDispatch(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean DoBeforeDispatch(TWebRequest Request, TWebResponse Response);
```

Description

Applications cannot call DoBeforeDispatch. The DispatchAction (see page 15117) method calls DoBeforeDispatch before trying to match the HTTP request message with any of its action items.

If the BeforeDispatch (see page 15124) event handler indicates that it has handled the request message, DoBeforeDispatch returns true, and the DispatchAction (see page 15117) method does not try to pass the request message to any of the action items. Otherwise, DoBeforeDispatch returns false, and unless the response message was sent from within the BeforeDispatch (see page 15124) event handler, the DispatchAction (see page 15117) method proceeds with looking for action items to handle the request message.

If DoBeforeDispatch returns true but the response has not been sent, DispatchAction (see page 15117) generates an AfterDispatch (see page 15123) event.

See Also

AfterDispatch (see page 15123)

BeforeDispatch (see page 15124)

DispatchAction (see page 15117)

77.1.14.4.7 TCustomWebDispatcher.FinishContext Method

Pascal

```
procedure FinishContext;
```

C++

```
__fastcall FinishContext();
```

Description

This is FinishContext, a member of class TCustomWebDispatcher.

77.1.14.4.8 TCustomWebDispatcher.HandleException Method

Provides default handling of exceptions for the web application.

Pascal

```
procedure HandleException(E: Exception; var Handled: Boolean);
```

C++

```
__fastcall HandleException(Exception E, Boolean Handled);
```

Description

If an exception passes through all the try blocks in the web application code, the application automatically calls HandleException, which displays a dialog box indicating an error occurred. Unless the exception object is EAbort, HandleException calls the OnException (see page 15124) event handler.

To assign other exception handling code for the web application, use the OnException (see page 15124) event handler.

See Also

OnException (see page 15124)

77.1.14.4.9 TCustomWebDispatcher.HandleRequest Method

Processes an HTTP request message.

Pascal

```
function HandleRequest(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(TWebRequest Request, TWebResponse Response);
```

Description

Call HandleRequest to process an HTTP request message and obtain the response that should be sent back to the caller.

Request is the request object that describes the request to process.

Response is the response object that should be filled out when the request is processed.

77.1.14.4.10 TCustomWebDispatcher.InitContext Method

Pascal

```
procedure InitContext(Request: TWebRequest; Response: TWebResponse);
```

C++

```
__fastcall InitContext(TWebRequest Request, TWebResponse Response);
```

Description

This is InitContext, a member of class TCustomWebDispatcher.

77.1.14.4.11 TCustomWebDispatcher.InitModule Method

Pascal

```
procedure InitModule(AModule: TComponent);
```

C++

```
__fastcall InitModule(TComponent * AModule);
```

Description

This is InitModule, a member of class TCustomWebDispatcher.

77.1.14.4.12 TCustomWebDispatcher.Notification Method

Responds to notifications when objects are added to the module.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification ensures that all TCustomContentProducer (see page 15102) objects have access to the dispatcher, by setting their Dispatcher property as they are added to the application.

See Also

Dispatcher (see page 15109)

77.1.14.5 TCustomWebDispatcher Properties

The properties of the TCustomWebDispatcher class are listed here.

Public Properties

	Name	Description
R	Action (see page 15121)	Provides indexed access to the action items managed by the TCustomWebDispatcher (see page 15110) object.
	Actions (see page 15122)	Represents the collection of action items managed by the TCustomWebDispatcher (see page 15110) object.
R	Request (see page 15122)	Provides access to the TWebRequest (see page 15161) object that the dispatcher is currently handling.
R	Response (see page 15122)	Provides access to the TWebResponse (see page 15191) object for the message the dispatcher is currently handling.

77.1.14.5.1 TCustomWebDispatcher.Action Property

Provides indexed access to the action items managed by the TCustomWebDispatcher (see page 15110) object.

Pascal

```
property Action [Index: Integer]: TWebActionItem;
```

C++

```
__property TWebActionItem Action[int Index];
```

Description

Use Action to iterate over all of the actions in the Actions (see page 15122) collection. Zero is the index of the first action item, one is the index of the second action item, and so on. To access individual actions by name, use the ActionByName (see page 15117) method.

See Also

Actions (see page 15122)

TWebActionItem (see page 15130)

77.1.14.5.2 TCustomWebDispatcher.Actions Property

Represents the collection of action items managed by the TCustomWebDispatcher (see page 15110) object.

Pascal

```
property Actions: TWebActionItems;
```

C++

```
__property TWebActionItems Actions;
```

Description

Use the methods of Actions to add action items for use by the dispatcher, to clear the set of action items managed by the dispatcher, or to access the attributes of the action items.

See Also

Action (see page 15121)

ActionByName (see page 15117)

TWebActionItems (see page 15140)

77.1.14.5.3 TCustomWebDispatcher.Request Property

Provides access to the TWebRequest (see page 15161) object that the dispatcher is currently handling.

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

Use Request to obtain information about the current HTTP request message. The dispatcher matches this TWebRequest (see page 15161) object to an action item, and invokes the OnAction event handler of the action item to generate and send a response to the HTTP request message. OnAction event handlers often use descendants of TCustomContentProducer (see page 15102) to help generate the response to the request message. These content producers can use the Request property to correctly produce the content of the HTTP response.

See Also

DispatchAction (see page 15117)

Response (see page 15122)

TCustomContentProducer (see page 15102)

OnAction (see page 15139)

77.1.14.5.4 TCustomWebDispatcher.Response Property

Provides access to the TWebResponse (see page 15191) object for the message the dispatcher is currently handling.

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

Use Response to obtain information about the emerging response to the current HTTP request message. The dispatcher matches the request object specified by the Request (see page 15122) property to an action item. It then invokes the OnAction event handler of the action item to fill in the TWebResponse (see page 15191) object specified by the Response property. OnAction event handlers often use descendants of TCustomContentProducer (see page 15102) to help generate the response. These content producers can use Response to determine the state of the emerging response that is being produced.

See Also

DispatchAction (see page 15117)

Request (see page 15122)

TCustomContentProducer (see page 15102)

OnAction (see page 15139)

77.1.14.6 TCustomWebDispatcher Events

The events of the TCustomWebDispatcher class are listed here.

Protected Events

	Name	Description
	AfterDispatch (see page 15123)	Occurs after the HTTP response message has been successfully filled out by the action items but not yet sent.
	BeforeDispatch (see page 15124)	Occurs before the dispatcher tries to match the HTTP request message with any of its action items.

Public Events

	Name	Description
	OnException (see page 15124)	Occurs when an unhandled exception occurs in the application.

77.1.14.6.1 TCustomWebDispatcher.AfterDispatch Event

Occurs after the HTTP response message has been successfully filled out by the action items but not yet sent.

Pascal

```
property AfterDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent AfterDispatch;
```

Description

Write an AfterDispatch event handler to provide a final check of the HTTP response message before it is sent to the Web client. An AfterDispatch event handler can let a response stand as is, change the response, send the response, or change the Handled parameter to False to prevent the Web application from sending the response.

The AfterDispatch event does not occur if all of the action items indicate that the response was not handled, or if the response is sent by any of the action items.

See Also

Actions (see page 15122)

BeforeDispatch (see page 15124)

TWebResponse (see page 15191)

Dispatching Request Messages

77.1.14.6.2 TCustomWebDispatcher.BeforeDispatch Event

Occurs before the dispatcher tries to match the HTTP request message with any of its action items.

Pascal

```
property BeforeDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent BeforeDispatch;
```

Description

Write a BeforeDispatch event handler to preprocess all HTTP request messages. A BeforeDispatch event handler can enable or disable the action items appropriate to the response, begin filling out the response object, or provide any other necessary preprocessing.

If the BeforeDispatch event handler finishes filling out the response object, it should change the Handled parameter to True so the dispatcher will not send the request on to any of the action items. If the BeforeDispatch event handler sends the response message, the dispatcher will not pass the request on to any of the action items, even if the Handled parameter is left as False.

If the BeforeDispatch event handler sets the Handled parameter to True but does not send the response, the Web dispatcher will generate an AfterDispatch (see page 15123) event.

See Also

Actions (see page 15122)

AfterDispatch (see page 15123)

TWebResponse (see page 15191)

Dispatching Request Messages

Enabling and Disabling Action Items

77.1.14.6.3 TCustomWebDispatcher.OnException Event

Occurs when an unhandled exception occurs in the application.

Pascal

```
property OnException: TWebExceptionEvent;
```

C++

```
__property TWebExceptionEvent OnException;
```

Description

Use OnException to change the default behavior that occurs when an exception is not handled by application code. The OnException event handler is called automatically in the HandleException (see page 15119) method.

OnException only handles exceptions that occur during message processing. Exceptions that occur before or after the execution of the application's Run method do not generate OnException events.

If an exception passes through the try blocks in the application code, the application automatically calls the HandleException (see page 15119) method. Unless the exception object is EAbort, HandleException (see page 15119) calls the OnException

handler, if one exists. Otherwise, it calls ShowException to display a message box indicating an error occurred.

See Also

Actions (see page 15122)

AfterDispatch (see page 15123)

TWebResponse (see page 15191)

77.1.15 TDefaultWebAppServices Class

Class Hierarchy

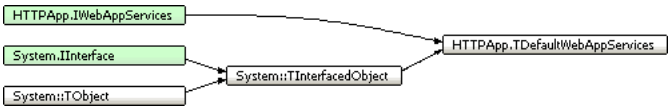

Pascal

```
TDefaultWebAppServices = class(TInterfacedObject, IWebAppServices);
```

C++

```
class TDefaultWebAppServices : public TInterfacedObject, public IWebAppServices;
```

File

HTTPApp

Description

This is class HTTPApp.TDefaultWebAppServices.

77.1.15.1 TDefaultWebAppServices Members

The following tables list the members exposed by TDefaultWebAppServices.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
V	FindRequestHandler (see page 15127)	This is FindRequestHandler, a member of class TDefaultWebAppServices.
	FinishContext (see page 15127)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetExceptionHandler (see page 15127)	Returns the object that handles exceptions that arise in the WebSnap application.
	GetRequestHandler (see page 15128)	This is GetRequestHandler, a member of class TDefaultWebAppServices.
	HandleRequest (see page 15128)	Directs the processing of the current request message.
	InitContext (see page 15128)	Creates and initializes the Web context for the current Web request.
	InvokeDispatcher (see page 15129)	This is InvokeDispatcher, a member of class TDefaultWebAppServices.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Protected Properties

	Name	Description
R	Request (see page 15129)	This is Request, a member of class TDefaultWebAppServices.
R	RequestHandler (see page 15130)	This is RequestHandler, a member of class TDefaultWebAppServices.
R	Response (see page 15130)	This is Response, a member of class TDefaultWebAppServices.
R	WebModules (see page 15130)	This is WebModules, a member of class TDefaultWebAppServices.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ExceptionHandler (see page 15230)	Indicates the object that handles exceptions that arise when processing a request message.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

77.1.15.2 TDefaultWebAppServices Methods

The methods of the TDefaultWebAppServices class are listed here.

Protected Methods

	Name	Description
V	FindRequestHandler (see page 15127)	This is FindRequestHandler, a member of class TDefaultWebAppServices.
	FinishContext (see page 15127)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetExceptionHandler (see page 15127)	Returns the object that handles exceptions that arise in the WebSnap application.
	GetRequestHandler (see page 15128)	This is GetRequestHandler, a member of class TDefaultWebAppServices.
	HandleRequest (see page 15128)	Directs the processing of the current request message.
	InitContext (see page 15128)	Creates and initializes the Web context for the current Web request.
	InvokeDispatcher (see page 15129)	This is InvokeDispatcher, a member of class TDefaultWebAppServices.

77.1.15.2.1 TDefaultWebAppServices.FindRequestHandler Method

Pascal

```
function FindRequestHandler: IWebRequestHandler; virtual;
```

C++

```
virtual __fastcall IWebRequestHandler FindRequestHandler();
```

Description

This is FindRequestHandler, a member of class TDefaultWebAppServices.

77.1.15.2.2 TDefaultWebAppServices.FinishContext Method

Performs any final tasks after the WebSnap application has finished processing a request.

Pascal

```
procedure FinishContext;
```

C++

```
__fastcall FinishContext();
```

Description

FinishContext is called as the last step before a WebSnap application sends a response message back to the caller. FinishContext performs any final tasks, and frees all the objects and components that were created to handle the request message (including the Web context).

See Also

InitContext (see page 15229)

WebContext (see page 24687)

77.1.15.2.3 TDefaultWebAppServices.GetExceptionHandler Method

Returns the object that handles exceptions that arise in the WebSnap application.

Pascal

```
function GetExceptionHandler: TObject;
```

C++

```
__fastcall TObject * GetExceptionHandler();
```

Description

GetExceptionHandler is the read implementation of the ExceptionHandler property.

See Also

ExceptionHandler (see page 15230)

77.1.15.2.4 TDefaultWebAppServices.GetRequestHandler Method

Pascal

```
function GetRequestHandler: IWebRequestHandler;
```

C++

```
__fastcall IWebRequestHandler GetRequestHandler();
```

Description

This is GetRequestHandler, a member of class TDefaultWebAppServices.

77.1.15.2.5 TDefaultWebAppServices.HandleRequest Method

Directs the processing of the current request message.

Pascal

```
function HandleRequest: Boolean;
```

C++

```
__fastcall Boolean HandleRequest();
```

Description

When the WebSnap application receives a request message, it makes the following calls:

1. The InitContext method is called to generate a Web context for the request.
2. HandleRequest is called to generate a response to the request.
3. FinishContext is called to perform any final cleanup.

HandleRequest locates the dispatcher components that handle the request and forwards the request to a dispatcher for processing.

HandleRequest returns true if the request is handled and the response should be sent back to the caller (if it was not already sent). It returns false if the WebSnap application could not handle the request.

See Also

InitContext (see page 15229)

FinishContext (see page 15228)

77.1.15.2.6 TDefaultWebAppServices.InitContext Method

Creates and initializes the Web context for the current Web request.

Pascal

```
procedure InitContext(WebModuleList: TAbstractWebModuleList; Request: TWebRequest; Response:
TWebResponse);
```

C++

```
__fastcall InitContext(TAbstractWebModuleList WebModuleList, TWebRequest Request, TWebResponse
Response);
```

Description

InitContext is called when the WebSnap application first receives a request message to set up the Web context before the application attempts to handle the request. It creates and sets up the global Web context, which can be accessed by calling the global WebContext function. It also performs any other initializations that ready the application for handling a new request message.

WebModuleList lists the currently active Web modules in the WebSnap application and provides access to all Web module factories so that the application can create any new modules it needs for processing the request message.

Request describes the new request message.

Response is the response object that the WebSnap application should fill out when handling the current request.

See Also

FinishContext (see page 15228)

WebContext (see page 24687)

77.1.15.2.7 TDefaultWebAppServices.InvokeDispatcher Method

Pascal

```
function InvokeDispatcher: Boolean;
```

C++

```
__fastcall Boolean InvokeDispatcher();
```

Description

This is InvokeDispatcher, a member of class TDefaultWebAppServices.

77.1.15.3 TDefaultWebAppServices Properties

The properties of the TDefaultWebAppServices class are listed here.

Protected Properties

	Name	Description
R	Request (see page 15129)	This is Request, a member of class TDefaultWebAppServices.
R	RequestHandler (see page 15130)	This is RequestHandler, a member of class TDefaultWebAppServices.
R	Response (see page 15130)	This is Response, a member of class TDefaultWebAppServices.
R	WebModules (see page 15130)	This is WebModules, a member of class TDefaultWebAppServices.

77.1.15.3.1 TDefaultWebAppServices.Request Property

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

This is Request, a member of class TDefaultWebAppServices.

77.1.15.3.2 TDefaultWebAppServices.RequestHandler Property

Pascal

```
property RequestHandler: IWebRequestHandler;
```

C++

```
__property IWebRequestHandler RequestHandler;
```

Description

This is RequestHandler, a member of class TDefaultWebAppServices.

77.1.15.3.3 TDefaultWebAppServices.Response Property

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

This is Response, a member of class TDefaultWebAppServices.

77.1.15.3.4 TDefaultWebAppServices.WebModules Property

Pascal

```
property WebModules: TAbstractWebModuleList;
```

C++

```
__property TAbstractWebModuleList WebModules;
```

Description

This is WebModules, a member of class TDefaultWebAppServices.

77.1.16 TWebActionItem Class

TWebActionItem allows a Web server application to create and send responses to HTTP request messages.

Class Hierarchy

Pascal

```
TWebActionItem = class(TCollectionItem);
```

C++

```
class TWebActionItem : public TCollectionItem;
```

File

HTTPApp

Description

Use TWebActionItem to assemble the response to a particular type of HTTP request message in a Web server application. Each action item is specific to a particular URI path and HTTP request method.

The web dispatcher chooses an appropriate action item to handle an HTTP request message by

matching the path portion of the URI requested by the Web client with the PathInfo (see page 15138) property in the TWebActionItem object

matching the method requested by the Web client with the method type the action item knows how to handle.

An action item handles an HTTP request message automatically using an associated content producer or using its OnAction (see page 15139) event handler. The OnAction (see page 15139) event handler can read the request message and assemble a response by setting the properties of a TWebResponse (see page 15191) object. It can tell the TWebResponse (see page 15191) object to send the response message back to the Web client, or allow the TWebApplication object to send the message after other action items have had a chance to add to the response.

See Also

TWebActionItems (see page 15140)

TWebDispatcher (see page 15147)

TWebRequest (see page 15161)

TWebResponse (see page 15191)

Action Items

77.1.16.1 TWebActionItem Members

The following tables list the members exposed by TWebActionItem.

Protected Methods

	Name	Description
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetDisplayName (see page 15134)	Returns the name of the collection item as it appears in the collection editor.
V	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPathInfo (see page 15135)	This is GetPathInfo, a member of class TWebActionItem.
V	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
V	SetDisplayName (see page 15135)	Sets the name of the collection item as it appears in the collection editor.
V	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	AssignTo (see page 15134)	Copies the properties of the action item to another action item.

V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 15133)	Creates an instance of TWebActionItem (see page 15130).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 15133)	Destroys an instance of TWebActionItem (see page 15130).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Events

	Name	Description
	OnAction (see page 15139)	Occurs when TWebDispatcher (see page 15147) matches the action item to a newly-arrived HTTP request message.

Published Properties

	Name	Description
	Default (see page 15135)	Indicates whether the action item should handle any request that is not handled by another available action item.
	Enabled (see page 15136)	Indicates whether the action item can respond to HTTP request messages that match its MethodType (see page 15137) and PathInfo (see page 15138) properties.
	MethodType (see page 15137)	This is MethodType, a member of class TWebActionItem.
	Name (see page 15137)	Identifies an individual action item when it is referenced from other objects.
	PathInfo (see page 15138)	Indicates the path portion of the Uniform Resource Identifier (URI) for which the action item handles requests.

	Producer (see page 15138)	Specifies a content producer that generates the content of response messages when the action item executes.
	ProducerContent (see page 15139)	Specifies a content producer that generates the content of response messages when the action item executes.

77.1.16.2 TWebActionItem.Create Constructor

Creates an instance of TWebActionItem (see page 15130).

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TWebActionItem(TCollection Collection);
```

Description

Call Create to create a TWebActionItem (see page 15130) object at runtime. Action items created in the action editor at design time are instantiated automatically.

After calling the inherited constructor, Create initializes the Enabled (see page 15136) property to True, and sets the action item so that it will match any HTTP request targeted to a URI with an empty PathInfo (see page 15138).

See Also

Enabled (see page 15136)

PathInfo (see page 15138)

Create (see page 2757)

77.1.16.3 TWebActionItem.Destroy Destructor

Destroys an instance of TWebActionItem (see page 15130).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebActionItem();
```

Description

Do not call Destroy directly in an application. Action items are destroyed when the TWebActionItems (see page 15140) object that manages them calls the Free method.

See Also

Free (see page 22300)

TWebActionItems (see page 15140)

77.1.16.4 TWebActionItem Methods

The methods of the TWebActionItem class are listed here.

Protected Methods

	Name	Description
	GetDisplayName (see page 15134)	Returns the name of the collection item as it appears in the collection editor.
	GetPathInfo (see page 15135)	This is GetPathInfo, a member of class TWebActionItem.
	SetDisplayName (see page 15135)	Sets the name of the collection item as it appears in the collection editor.

Public Methods

	Name	Description
	AssignTo (see page 15134)	Copies the properties of the action item to another action item.

77.1.16.4.1 TWebActionItem.AssignTo Method

Copies the properties of the action item to another action item.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

To assign the value of one action item to another, use the Assign method of the target action item.

AssignTo enables the Assign method of a target action item to copy the properties of a source action item. AssignTo copies the values of the Default (see page 15135), Enabled (see page 15136), MethodType (see page 15137) and PathInfo (see page 15138) properties to the Dest parameter if it is an action item. If Dest is not an action item, AssignTo calls the inherited method.

See Also

Assign (see page 2930)

Default (see page 15135)

Enabled (see page 15136)

MethodType (see page 15137)

PathInfo (see page 15138)

AssignTo (see page 2931)

77.1.16.4.2 TWebActionItem.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName (see page 2761)

GetNamePath (see page 2759)

77.1.16.4.3 TWebActionItem.GetPathInfo Method

Pascal

```
function GetPathInfo: string;
```

C++

```
__fastcall AnsiString GetPathInfo();
```

Description

This is GetPathInfo, a member of class TWebActionItem.

77.1.16.4.4 TWebActionItem.SetDisplayName Method

Sets the name of the collection item as it appears in the collection editor.

Pascal

```
procedure SetDisplayName(const Value: string); override;
```

C++

```
virtual __fastcall SetDisplayName(const AnsiString Value);
```

Description

SetDisplayName is the protected write implementation of the DisplayName property.

Value specifies the name for the item that should appear in the collection editor.

See Also

DisplayName (see page 2761)

77.1.16.5 TWebActionItem Properties

The properties of the TWebActionItem class are listed here.

Published Properties

	Name	Description
	Default (see page 15135)	Indicates whether the action item should handle any request that is not handled by another available action item.
	Enabled (see page 15136)	Indicates whether the action item can respond to HTTP request messages that match its MethodType (see page 15137) and PathInfo (see page 15138) properties.
	MethodType (see page 15137)	This is MethodType, a member of class TWebActionItem.
	Name (see page 15137)	Identifies an individual action item when it is referenced from other objects.
	PathInfo (see page 15138)	Indicates the path portion of the Uniform Resource Identifier (URI) for which the action item handles requests.
	Producer (see page 15138)	Specifies a content producer that generates the content of response messages when the action item executes.
	ProducerContent (see page 15139)	Specifies a content producer that generates the content of response messages when the action item executes.

77.1.16.5.1 TWebActionItem.Default Property

Indicates whether the action item should handle any request that is not handled by another available action item.

Pascal

```
property Default: Boolean;
```

C++

```
__property Boolean Default;
```

Description

Set Default to true to make this action item the default action in the list of action items maintained by the TWebDispatcher (see page 15147) object. TWebDispatcher (see page 15147) attempts to match HTTP request messages to action items by matching the MethodType (see page 15137) and PathInfo (see page 15138) of the target URI. If no match can be found, the TWebDispatcher (see page 15147) invokes the default action item, regardless of its MethodType (see page 15137) or PathInfo (see page 15138) properties.

When an action item is chosen as the default action item, the value of its Enabled (see page 15136) property is ignored. Thus, if an action item sets Enabled (see page 15136) to false and Default to true, TWebDispatcher (see page 15147) will first look for a match with any other action item, even if the PathInfo (see page 15138) and MethodType (see page 15137) of the default action item match the request perfectly. Then, only after all other action items have been rejected, TWebDispatcher (see page 15147) selects the default action item. If the default action item is enabled, it may be selected before all other action items have been considered because the MethodType (see page 15137) and PathInfo (see page 15138) match the request.

The default action item must be prepared to generate a response to any HTTP request, for any MethodType (see page 15137) and PathInfo (see page 15138). This may consist of simply setting a response status code of 404 (the URI could not be matched) or 405 (the requested method is not supported).

Setting Default to true sets the Default property of the previous default action item to false.

See Also

Enabled (see page 15136)

OnAction (see page 15139)

MethodType (see page 15137)

PathInfo (see page 15138)

TWebDispatcher (see page 15147)

StatusCode (see page 15215)

Choosing a Default Action Item

77.1.16.5.2 TWebActionItem.Enabled Property

Indicates whether the action item can respond to HTTP request messages that match its MethodType (see page 15137) and PathInfo (see page 15138) properties.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Set Enabled to false to disable an action item. TWebDispatcher (see page 15147) will ignore disabled action items when it looks for an action item to handle a request.

When an action item is chosen as the default action item, the value of its Enabled property is ignored. Thus, if an action item sets

Enabled to false and Default (see page 15135) to true, TWebDispatcher (see page 15147) will first look for a match with any other action item, even if the PathInfo (see page 15138) and MethodType (see page 15137) of the default action item match the request perfectly. Then, only after all other action items have been rejected, the TWebDispatcher (see page 15147) selects the default action item. If the default action item is enabled, it may be selected before all other action items have been considered because the MethodType (see page 15137) and PathInfo (see page 15138) match the request.

See Also

Default (see page 15135)

MethodType (see page 15137)

OnAction (see page 15139)

PathInfo (see page 15138)

TWebDispatcher (see page 15147)

Choosing a Default Action Item

Enabling and Disabling Action Items

77.1.16.5.3 TWebActionItem.MethodType Property

Pascal

```
property MethodType: TMethodType;
```

C++

```
__property TMethodType MethodType;
```

Description

This is MethodType, a member of class TWebActionItem.

77.1.16.5.4 TWebActionItem.Name Property

Identifies an individual action item when it is referenced from other objects.

Pascal

```
property Name: string;
```

C++

```
__property AnsiString Name;
```

Description

Use Name to uniquely specify an individual action item in code anywhere in the application. For action items, Name is equivalent to the DisplayName property. Therefore, when setting the Name property, the new value must not match the DisplayName of any other action item in the list maintained by TCustomWebDispatcher (see page 15110). Trying to set the Name to a DisplayName that is already in use raises an exception.

See Also

DisplayName

Name (see page 2805)

77.1.16.5.5 TWebActionItem.PathInfo Property

Indicates the path portion of the Uniform Resource Identifier (URI) for which the action item handles requests.

Pascal

```
property PathInfo: string;
```

C++

```
__property AnsiString PathInfo;
```

Description

Set PathInfo to the path portion of the URI for the target of any request messages serviced by the action item. TWebDispatcher (see page 15147) matches action items to request messages by comparing PathInfo with the path portion of the URI specified as the target of the request. If the PathInfo properties match, and the MethodType (see page 15137) matches, TWebDispatcher (see page 15147) invokes the action item to respond to the request message.

If the action item has an associated content producer, the value of PathInfo is the name of the producer unless it is explicitly changed.

See Also

OnAction (see page 15139)

MethodType (see page 15137)

TWebDispatcher (see page 15147)

PathInfo (see page 15185)

Producer (see page 15138)

The Target URL

77.1.16.5.6 TWebActionItem.Producer Property

Specifies a content producer that generates the content of response messages when the action item executes.

Pascal

```
property Producer: TCustomContentProducer;
```

C++

```
__property TCustomContentProducer Producer;
```

Description

Set Producer to associate the action item with a content producer of the type TCustomContentProducer (see page 15102). (If a content producer of type IProduceContent (see page 15223) is used instead of TCustomContentProducer (see page 15102), then the property ProducerContent (see page 15139) should be set instead.) If Producer is not nil (Delphi) or NULL (C++), then when the web dispatcher passes a request message to the action item, the action item sets the Content property of the response message to the value returned by the producer's Content method. After the content producer has assigned the response's Content property, the action item receives an OnAction (see page 15139) event, where it can optionally set other response properties or send the response message.

If Producer is nil (Delphi) or NULL (C++), the action item must use the OnAction (see page 15139) event handler to influence the response message.

The properties Producer and ProducerContent (see page 15139) are mutually exclusive. If one is set, the other must not be.

Note: Each content producer can be associated with only one action item at a time. Setting the Producer property of one action

item causes the Web dispatcher that manages that item to check whether any other action item has its Producer property set to the same component. If another action item was associated with the same producer, its Producer property is changed to nil (Delphi) or NULL (C++).

See Also

Content (see page 15106)

OnAction (see page 15139)

ProducerContent (see page 15139)

Content (see page 15207)

Creating Web Pages with an InternetExpress Page Producer

Responding to Request Messages with Action Items

77.1.16.5.7 TWebActionItem.ProducerContent Property

Specifies a content producer that generates the content of response messages when the action item executes.

Pascal

```
property ProducerContent: TComponent;
```

C++

```
__property TComponent * ProducerContent;
```

Description

This property is identical to Producer (see page 15138) except for the type of content producer. ProducerContent is used when the producer is of type IProduceContent (see page 15223) instead of TCustomContentProducer (see page 15102). In the latter case, the property Producer (see page 15138) is set instead. Note that Producer (see page 15138) and ProducerContent are mutually exclusive; if one is set, the other must not be.

See Also

Producer (see page 15138)

77.1.16.6 TWebActionItem Events

The events of the TWebActionItem class are listed here.

Published Events

	Name	Description
	OnAction (see page 15139)	Occurs when TWebDispatcher (see page 15147) matches the action item to a newly-arrived HTTP request message.

77.1.16.6.1 TWebActionItem.OnAction Event

Occurs when TWebDispatcher (see page 15147) matches the action item to a newly-arrived HTTP request message.

Pascal

```
property OnAction: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent OnAction;
```

Description

Write an OnAction event handler to respond to HTTP request messages that match the MethodType (see page 15137) and PathInfo (see page 15138) properties. OnAction occurs immediately after the response message's Content property has been initialized by the action item's Producer (see page 15138) property (if Producer (see page 15138) is set).

In an OnAction event handler, read Request and perform the requested Method if possible. Set the properties of the Response parameter as appropriate. If the OnAction event handler does not service the response, or if it only partially handles the request and must let another action item finish the response, set the Handled parameter to false.

An OnAction event handler can call the SendRedirect or SendResponse method of Response to send the HTTP response message to the Web client. If the OnAction event handler does not send the response, the Web application will send it after all action items have finished filling in the Response.

Tip: The OnAction event handler can use descendants of TCustomContentProducer (see page 15102) to help assemble the contents of the response message.

See Also

MethodType (see page 15137)

PathInfo (see page 15138)

TCustomContentProducer (see page 15102)

TWebRequest (see page 15161)

TWebResponse (see page 15191)

SendRedirect (see page 15201)

SendResponse (see page 15201)

Producer (see page 15138)

Creating Web Pages with an InternetExpress Page Producer

Responding to Request Messages with Action Items

Sending the Response

77.1.17 TWebActionItems Class

TWebActionItems is a collection of TWebActionItem (see page 15130) objects.

Class Hierarchy

Pascal

```
TWebActionItems = class(TCollection);
```

C++

```
class TWebActionItems : public TCollection;
```

File

HTTPApp

Description

TCustomWebDispatcher (see page 15110) uses TWebActionItems to store and manipulate the individual action items it manages. TWebActionItems provides properties and events to

Add (see page 15143) new action items to the collection.

Copy the list of action items from another TWebActionItems.

Get the attributes of specified action items in the collection.

Empty the collection of action items.

See Also

TCustomWebDispatcher (see page 15110)

TWebActionItem (see page 15130)

77.1.17.1 TWebActionItems Members

The following tables list the members exposed by TWebActionItems.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
V	GetAttr (see page 15144)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
V	GetAttrCount (see page 15144)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
V	GetItemAttr (see page 15144)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 15145)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 15145)	Initializes the name of a newly inserted collection item.
V	Update (see page 15146)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 15143)	Adds a new action item to the end of the collection.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 15142)	Creates an instance of TWebActionItems (see page 15140).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 15146)	Lists the TWebActionItem (see page 15130) objects stored by this TWebActionItems (see page 15140) object.
R	WebDispatcher (see page 15147)	Provides access to the dispatcher that uses TWebActionItems (see page 15140) to store and manipulate its action items.

77.1.17.2 TWebActionItems.Create Constructor

Creates an instance of TWebActionItems (see page 15140).

Pascal

```
constructor Create(WebDispatcher: TCustomWebDispatcher; ItemClass: TCollectionItemClass);
```

C++

```
__fastcall TWebActionItems(TCustomWebDispatcher WebDispatcher, TCollectionItemClass ItemClass);
```

Description

Do not call the constructor for TWebActionItems (see page 15140). TWebActionItems (see page 15140) is tailored to work with TCustomWebDispatcher (see page 15110). The TCustomWebDispatcher (see page 15110) object creates TWebActionItems (see page 15140) in its own constructor.

See Also

Create (see page 2734)

77.1.17.3 TWebActionItems Methods

The methods of the TWebActionItems class are listed here.

Protected Methods

	Name	Description
	GetAttr (see page 15144)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
	GetAttrCount (see page 15144)	Returns the number of custom attributes associated with items in the collection.
	GetItemAttr (see page 15144)	Returns the value of a custom attribute assigned to one of the collection's items.
	GetOwner (see page 15145)	Returns the owner of an object.
	SetItemName (see page 15145)	Initializes the name of a newly inserted collection item.
	Update (see page 15146)	Updates the collection to reflect changes to its items.

Public Methods

	Name	Description
	Add (see page 15143)	Adds a new action item to the end of the collection.

77.1.17.3.1 TWebActionItems.Add Method

Adds a new action item to the end of the collection.

Pascal

```
function Add: TWebActionItem;
```

C++

```
__fastcall TWebActionItem Add();
```

Description

Call Add to add an action item to the collection of action items managed by the dispatcher. The new item is added to the end of the Items (see page 15146) array. Add returns the action item so that its properties can be set to specify its use.

Because the dispatcher looks through the action items in order when it tries to find an action item to handle an HTTP request message, add any action items that preprocess HTTP requests first. Then add the most specific action items. This way, more general action items will not be matched to request messages before a closer match can be found.

See Also

Items (see page 15146)

77.1.17.3.2 TWebActionItems.GetAttr Method

Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.

Pascal

```
function GetAttr(Index: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetAttr(int Index);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. The GetAttr method returns the name of an attribute.

Index identifies the attribute whose name is requested. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

As implemented by TCollection, GetAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttrCount (see page 2744)

GetItemAttr (see page 2745)

77.1.17.3.3 TWebActionItems.GetAttrCount Method

Returns the number of custom attributes associated with items in the collection.

Pascal

```
function GetAttrCount: Integer; override;
```

C++

```
virtual __fastcall int GetAttrCount();
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetAttrCount returns the number of distinct attributes assigned to each item in the collection.

As implemented in TCollection, GetAttrCount always returns 0, because TCollection defines no custom attributes.

See Also

GetAttr (see page 2743)

GetItemAttr (see page 2745)

77.1.17.3.4 TWebActionItems.GetItemAttr Method

Returns the value of a custom attribute assigned to one of the collection's items.

Pascal

```
function GetItemAttr(Index: Integer; ItemIndex: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetItemAttr(int Index, int ItemIndex);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetItemAttr returns the value of one of these attributes for a specified item in the collection.

Index identifies which attribute's value is desired. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

ItemIndex identifies the item whose attribute value is desired. This is an index into the Items property array.

As implemented in TCollection, GetItemAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttr (see page 2743)

GetAttrCount (see page 2744)

77.1.17.3.5 TWebActionItems.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

77.1.17.3.6 TWebActionItems.SetItemName Method

Initializes the name of a newly inserted collection item.

Pascal

```
procedure SetItemName(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall SetItemName(TCollectionItem Item);
```

Description

The Insert method calls SetItemName to initialize the Name property of items when it inserts them into the collection.

As implemented in TCollection, SetItemName does nothing. Some TCollection descendants override this method to provide collection items with default names.

See Also

Insert (see page 2746)

Name (see page 2805)

77.1.17.3.7 TWebActionItems.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed (see page 2740)

Changed (see page 2758)

77.1.17.4 TWebActionItems Properties

The properties of the TWebActionItems class are listed here.

Public Properties

	Name	Description
	Items (see page 15146)	Lists the TWebActionItem (see page 15130) objects stored by this TWebActionItems (see page 15140) object.
R	WebDispatcher (see page 15147)	Provides access to the dispatcher that uses TWebActionItems (see page 15140) to store and manipulate its action items.

77.1.17.4.1 TWebActionItems.Items Property

Lists the TWebActionItem (see page 15130) objects stored by this TWebActionItems (see page 15140) object.

Pascal

```
property Items [Index: Integer]: TWebActionItem;
```

C++

```
__property TWebActionItem Items[int Index];
```

Description

Use Items to get access to the individual action items stored by the TWebActionItems (see page 15140) object. Items is an indexed array, where the first action item has index 0, the second action item has index 1, and so on. Use Items with the Count property to iterate over all the action items in the collection.

See Also

Count (see page 2749)

TWebActionItem (see page 15130)

operator_sb

77.1.17.4.2 TWebActionItems.WebDispatcher Property

Provides access to the dispatcher that uses TWebActionItems (see page 15140) to store and manipulate its action items.

Pascal

```
property WebDispatcher: TCustomWebDispatcher;
```

C++

```
__property TCustomWebDispatcher WebDispatcher;
```

Description

Read WebDispatcher to get access to the dispatcher object that uses the collection of action items to respond to HTTP request messages.

77.1.18 TWebDispatcher Class

TWebDispatcher passes HTTP request messages on to the appropriate action items that assemble a response.

Class Hierarchy

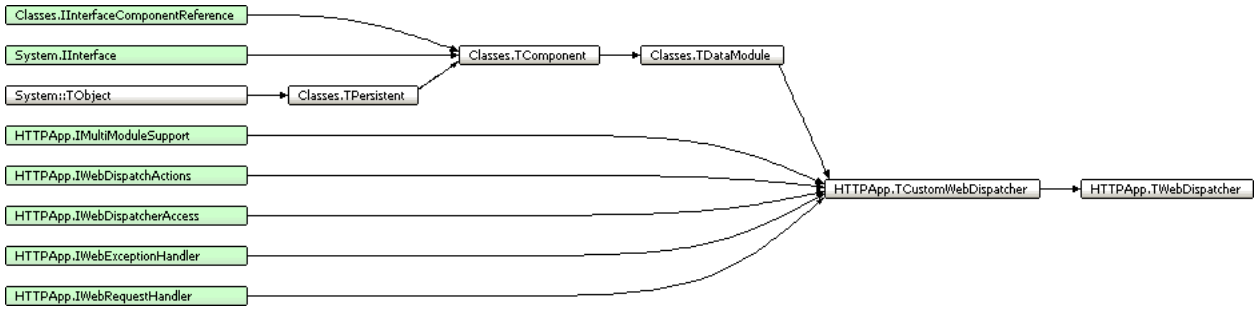

Pascal

```
TWebDispatcher = class(TCustomWebDispatcher);
```

C++

```
class TWebDispatcher : public TCustomWebDispatcher;
```

File

HTTPApp

Description

Use TWebDispatcher to convert an ordinary data module to a Web module. Add a TWebDispatcher object to an existing data module that contains other nonvisual components and business rules for the Web application. Then delete the automatically generated Web module from the Web application and replace it with the data module that now contains the Web dispatcher.

TWebDispatcher enables the Web server application to respond to HTTP request messages. Only one Web dispatcher can be added to an application. This is why the automatically generated Web module, which is a type of Web dispatcher, must be deleted from the Web application before the data module containing TWebDispatcher can be added.

The TWebDispatcher object manages a collection of action items which know how to respond to HTTP request messages. View, add, and delete the action items managed by the dispatcher using the Action (see page 15121) editor which is opened by double-clicking on the TWebDispatcher icon.

TWebDispatcher publishes some of the properties and events introduced by TCustomWebDispatcher (see page 15110), but introduces no new properties, events, or methods of its own.

See Also

The Web Dispatcher

The Web Module

77.1.18.1 TWebDispatcher Members

The following tables list the members exposed by TWebDispatcher.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	Access_Request (see page 15116)	This is Access_Request, a member of class TCustomWebDispatcher.
	Access_Response (see page 15117)	This is Access_Response, a member of class TCustomWebDispatcher.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DispatchAction (see page 15117)	Passes an HTTP request message to the action items in Actions (see page 15122) until it is handled.
	DoAfterDispatch (see page 15118)	Generates an AfterDispatch (see page 15123) event and returns a value that indicates whether the Web application can send the response.
	DoBeforeDispatch (see page 15119)	Generates a BeforeDispatch (see page 15124) event and returns a value that indicates whether the dispatcher should look for action items to handle the request.
V	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
V	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
	FinishContext (see page 15119)	This is FinishContext, a member of class TCustomWebDispatcher.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.

	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.
	HandleException (see page 15119)	Provides default handling of exceptions for the web application.
	HandleRequest (see page 15120)	Processes an HTTP request message.
	InitContext (see page 15120)	This is InitContext, a member of class TCustomWebDispatcher.
	InitModule (see page 15120)	This is InitModule, a member of class TCustomWebDispatcher.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 15121)	Responds to notifications when objects are added to the module.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActionByName (see page 15117)	Returns the action item identified by the AName parameter.
V	AfterConstruction (see page 2829)	Generates an OnCreate (see page 2833) event.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 2830)	Generates an OnDestroy (see page 2834) event.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 2827)	Creates an instance of a data module.
V	Create (see page 15115)	Creates an instance of TCustomWebDispatcher (see page 15110).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	CreateNew (see page 2828)	Creates an instance of a data module and registers with the global screen object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2829)	Destroys the instance of the data module.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 15115)	Frees an instance of TCustomWebDispatcher (see page 15110).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FinishContext (see page 15223)	This is FinishContext, a member of class IMultiModuleSupport.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandleException (see page 15236)	This is HandleException, a member of class IWebExceptionHandler.
	HandleRequest (see page 15238)	Processes an HTTP request message.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitContext (see page 15223)	This is InitContext, a member of class IMultiModuleSupport.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InitModule (see page 15223)	This is InitModule, a member of class IMultiModuleSupport.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Request (see page 15235)	Returns the Web request object for the HTTP request message to which the application is responding.
	Response (see page 15235)	Returns the Web response object for the HTTP response message that the application is assembling.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	Action (see page 15121)	Provides indexed access to the action items managed by the TCustomWebDispatcher (see page 15110) object.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Request (see page 15122)	Provides access to the TWebRequest (see page 15161) object that the dispatcher is currently handling.
R	Response (see page 15122)	Provides access to the TWebResponse (see page 15191) object for the message the dispatcher is currently handling.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterDispatch (see page 15152)	Occurs after the HTTP response message has been successfully filled out by the action items but not yet sent.
	BeforeDispatch (see page 15153)	Occurs before the dispatcher tries to match the HTTP request message with any of its action items.
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.
	OnException (see page 15153)	Occurs when an unhandled exception occurs in the application.

Published Properties

	Name	Description
	Actions (see page 15152)	Represents the collection of action items managed by the TCustomWebDispatcher object.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.
	Tag (see page 2808)	Stores an integer value as part of a component.

77.1.18.2 TWebDispatcher Properties

The properties of the TWebDispatcher class are listed here.

Published Properties

	Name	Description
	Actions (see page 15152)	Represents the collection of action items managed by the TCustomWebDispatcher object.

77.1.18.2.1 TWebDispatcher.Actions Property

Represents the collection of action items managed by the TCustomWebDispatcher object.

Pascal

```
property Actions: TWebActionItems;
```

C++

```
__property TWebActionItems Actions;
```

Description

Use the methods of Actions to add action items for use by the dispatcher, to clear the set of action items managed by the dispatcher, or to access the attributes of the action items.

See Also

Action (see page 15121)

ActionByName (see page 15117)

TWebActionItems (see page 15140)

77.1.18.3 TWebDispatcher Events

The events of the TWebDispatcher class are listed here.

Published Events

	Name	Description
	AfterDispatch (see page 15152)	Occurs after the HTTP response message has been successfully filled out by the action items but not yet sent.
	BeforeDispatch (see page 15153)	Occurs before the dispatcher tries to match the HTTP request message with any of its action items.
	OnException (see page 15153)	Occurs when an unhandled exception occurs in the application.

77.1.18.3.1 TWebDispatcher.AfterDispatch Event

Occurs after the HTTP response message has been successfully filled out by the action items but not yet sent.

Pascal

```
property AfterDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent AfterDispatch;
```

Description

Write an AfterDispatch event handler to provide a final check of the HTTP response message before it is sent to the Web client. An AfterDispatch event handler can let a response stand as is, change the response, send the response, or change the Handled parameter to False to prevent the Web application from sending the response.

The AfterDispatch event does not occur if all of the action items indicate that the response was not handled, or if the response is sent by any of the action items.

See Also

Actions (see page 15122)

BeforeDispatch (see page 15124)

TWebResponse (see page 15191)

Dispatching Request Messages

77.1.18.3.2 TWebDispatcher.BeforeDispatch Event

Occurs before the dispatcher tries to match the HTTP request message with any of its action items.

Pascal

```
property BeforeDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent BeforeDispatch;
```

Description

Write a BeforeDispatch event handler to preprocess all HTTP request messages. A BeforeDispatch event handler can enable or disable the action items appropriate to the response, begin filling out the response object, or provide any other necessary preprocessing.

If the BeforeDispatch event handler finishes filling out the response object, it should change the Handled parameter to True so the dispatcher will not send the request on to any of the action items. If the BeforeDispatch event handler sends the response message, the dispatcher will not pass the request on to any of the action items, even if the Handled parameter is left as False.

If the BeforeDispatch event handler sets the Handled parameter to True but does not send the response, the Web dispatcher will generate an AfterDispatch event.

See Also

Actions (see page 15122)

AfterDispatch (see page 15123)

TWebResponse (see page 15191)

Dispatching Request Messages

Enabling and Disabling Action Items

77.1.18.3.3 TWebDispatcher.OnException Event

Occurs when an unhandled exception occurs in the application.

Pascal

```
property OnException: TWebExceptionEvent;
```

C++

```
__property TWebExceptionEvent OnException;
```

Description

Use OnException to change the default behavior that occurs when an exception is not handled by application code. The OnException event handler is called automatically in the HandleException method.

OnException only handles exceptions that occur during message processing. Exceptions that occur before or after the execution of the application's Run method do not generate OnException events.

If an exception passes through the try blocks in the application code, the application automatically calls the HandleException method. Unless the exception object is EAbort, HandleException calls the OnException handler, if one exists. Otherwise, it calls ShowException to display a message box indicating an error occurred.

See Also

Actions (see page 15122)

AfterDispatch (see page 15123)

TWebResponse (see page 15191)

77.1.19 TWebModule Class

TWebModule is the automatically generated Web module for a Web server application.

Class Hierarchy

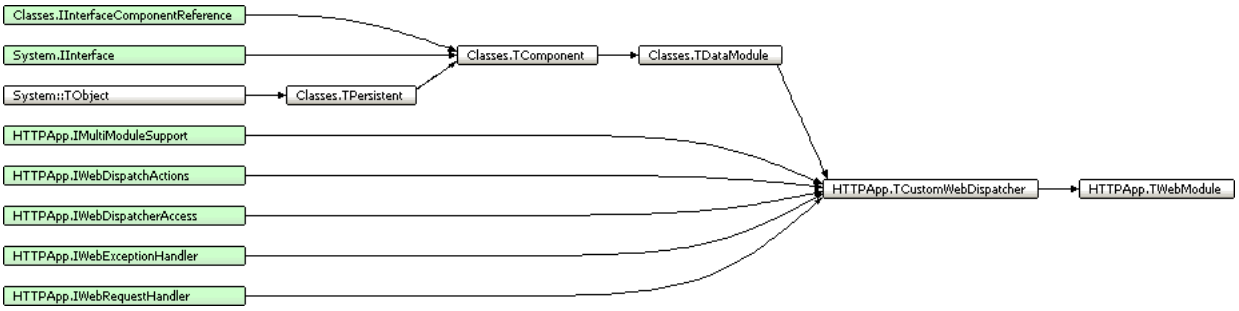

Pascal

```
TWebModule = class(TCustomWebDispatcher);
```

C++

```
class TWebModule : public TCustomWebDispatcher;
```

File

HTTPApp

Description

When a new Web application is created, it automatically contains a Web module. The Web module serves as a repository for nonvisual components such as TPageProducer, TSQLDataSet, TDataSetTableProducer, etc. It also enables the Web server application to respond to HTTP request messages by passing the request and response objects to the appropriate action items. An application can have only one Web module.

The TWebModule object manages a collection of action items which know how to respond to HTTP request messages. View,

add and delete the action items managed by the Web module using the Action (see page 15121) editor which is opened by double clicking on the TWebModule icon.

If a data module already exists which contains many of the non-visual components and business rules for the application, the Web module can be replaced by that data module. Simply delete the Web module, add the new data module in its place, and add a TWebDispatcher (see page 15147) object to the new data module.

See the WebServ demo for an example of how to use TWebModule.

Note: When you create a new Apache module project and save it with a new name, it may be necessary to change the name of the apache_module (in the exports section of the project source code) by hand.

See Also

TDataSetTableProducer (see page 10277)

TPageProducer (see page 15309)

TSQLDataSet (see page 20265)

TWebActionItems (see page 15140)

TWebDispatcher (see page 15147)

The Web Module

77.1.19.1 TWebModule Members

The following tables list the members exposed by TWebModule.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	Access_Request (see page 15116)	This is Access_Request, a member of class TCustomWebDispatcher.
	Access_Response (see page 15117)	This is Access_Response, a member of class TCustomWebDispatcher.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DispatchAction (see page 15117)	Passes an HTTP request message to the action items in Actions (see page 15122) until it is handled.
	DoAfterDispatch (see page 15118)	Generates an AfterDispatch (see page 15123) event and returns a value that indicates whether the Web application can send the response.
	DoBeforeDispatch (see page 15119)	Generates a BeforeDispatch (see page 15124) event and returns a value that indicates whether the dispatcher should look for action items to handle the request.
V	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
V	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
	FinishContext (see page 15119)	This is FinishContext, a member of class TCustomWebDispatcher.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.

	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.
	HandleException (see page 15119)	Provides default handling of exceptions for the web application.
	HandleRequest (see page 15120)	Processes an HTTP request message.
	InitContext (see page 15120)	This is InitContext, a member of class TCustomWebDispatcher.
	InitModule (see page 15120)	This is InitModule, a member of class TCustomWebDispatcher.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 15121)	Responds to notifications when objects are added to the module.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActionByName (see page 15117)	Returns the action item identified by the AName parameter.
V	AfterConstruction (see page 2829)	Generates an OnCreate (see page 2833) event.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 2830)	Generates an OnDestroy (see page 2834) event.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 15159)	Creates an instance of TWebModule (see page 15154).
V	CreateNew (see page 2828)	Creates an instance of a data module and registers with the global screen object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2829)	Destroys the instance of the data module.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 15115)	Frees an instance of TCustomWebDispatcher (see page 15110).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FinishContext (see page 15223)	This is FinishContext, a member of class IMultiModuleSupport.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandleException (see page 15236)	This is HandleException, a member of class IWebExceptionHandler.
	HandleRequest (see page 15238)	Processes an HTTP request message.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitContext (see page 15223)	This is InitContext, a member of class IMultiModuleSupport.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InitModule (see page 15223)	This is InitModule, a member of class IMultiModuleSupport.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Request (see page 15235)	Returns the Web request object for the HTTP request message to which the application is responding.
	Response (see page 15235)	Returns the Web response object for the HTTP response message that the application is assembling.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	Action (see page 15121)	Provides indexed access to the action items managed by the TCustomWebDispatcher (see page 15110) object.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Request (see page 15122)	Provides access to the TWebRequest (see page 15161) object that the dispatcher is currently handling.
R	Response (see page 15122)	Provides access to the TWebResponse (see page 15191) object for the message the dispatcher is currently handling.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterDispatch (see page 15160)	Occurs after the HTTP response message has been successfully filled out by the action items but not yet sent.
	BeforeDispatch (see page 15160)	Occurs before the dispatcher tries to match the HTTP request message with any of its action items.
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.
	OnException (see page 15161)	Occurs when an unhandled exception occurs in the application.

Published Properties

	Name	Description
	Actions (see page 15159)	Represents the collection of action items managed by the TCustomWebDispatcher object.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.
	Tag (see page 2808)	Stores an integer value as part of a component.

77.1.19.2 TWebModule.Create Constructor

Creates an instance of TWebModule (see page 15154).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebModule(TComponent * AOwner);
```

Description

Do not create instances at runtime. The Web module is created automatically when a Web application project is created at design time. Every time a Web server application is run in response to an HTTP request message, a new TWebModule (see page 15154) is created based on the Web module that was created at design time.

Create generates an OnCreate event, so that Web applications can initialize the Web module before the new HTTP request message is processed.

See Also

OnCreate (see page 2833)

Create_d

77.1.19.3 TWebModule Properties

The properties of the TWebModule class are listed here.

Published Properties

	Name	Description
	Actions (see page 15159)	Represents the collection of action items managed by the TCustomWebDispatcher object.

77.1.19.3.1 TWebModule.Actions Property

Represents the collection of action items managed by the TCustomWebDispatcher object.

Pascal

```
property Actions: TWebActionItems;
```

C++

```
__property TWebActionItems Actions;
```

Description

Use the methods of Actions to add action items for use by the dispatcher, to clear the set of action items managed by the dispatcher, or to access the attributes of the action items.

See Also

Action (see page 15121)

ActionByName (see page 15117)

TWebActionItems (see page 15140)

77.1.19.4 TWebModule Events

The events of the TWebModule class are listed here.

Published Events

	Name	Description
	AfterDispatch (see page 15160)	Occurs after the HTTP response message has been successfully filled out by the action items but not yet sent.
	BeforeDispatch (see page 15160)	Occurs before the dispatcher tries to match the HTTP request message with any of its action items.
	OnException (see page 15161)	Occurs when an unhandled exception occurs in the application.

77.1.19.4.1 TWebModule.AfterDispatch Event

Occurs after the HTTP response message has been successfully filled out by the action items but not yet sent.

Pascal

```
property AfterDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent AfterDispatch;
```

Description

Write an AfterDispatch event handler to provide a final check of the HTTP response message before it is sent to the Web client. An AfterDispatch event handler can let a response stand as is, change the response, send the response, or change the Handled parameter to False to prevent the Web application from sending the response.

The AfterDispatch event does not occur if all of the action items indicate that the response was not handled, or if the response is sent by any of the action items.

See Also

Actions (see page 15122)

BeforeDispatch (see page 15124)

TWebResponse (see page 15191)

Dispatching Request Messages

77.1.19.4.2 TWebModule.BeforeDispatch Event

Occurs before the dispatcher tries to match the HTTP request message with any of its action items.

Pascal

```
property BeforeDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent BeforeDispatch;
```

Description

Write a BeforeDispatch event handler to preprocess all HTTP request messages. A BeforeDispatch event handler can enable or disable the action items appropriate to the response, begin filling out the response object, or provide any other necessary preprocessing.

If the BeforeDispatch event handler finishes filling out the response object, it should change the Handled parameter to True so

the dispatcher will not send the request on to any of the action items. If the BeforeDispatch event handler sends the response message, the dispatcher will not pass the request on to any of the action items, even if the Handled parameter is left as False.

If the BeforeDispatch event handler sets the Handled parameter to True but does not send the response, the Web dispatcher will generate an AfterDispatch event.

See Also

Actions (see page 15122)

AfterDispatch (see page 15123)

TWebResponse (see page 15191)

Dispatching Request Messages

Enabling and Disabling Action Items

77.1.19.4.3 TWebModule.OnException Event

Occurs when an unhandled exception occurs in the application.

Pascal

```
property OnException: TWebExceptionEvent;
```

C++

```
__property TWebExceptionEvent OnException;
```

Description

Use OnException to change the default behavior that occurs when an exception is not handled by application code. The OnException event handler is called automatically in the HandleException method.

OnException only handles exceptions that occur during message processing. Exceptions that occur before or after the execution of the application's Run method do not generate OnException events.

If an exception passes through the try blocks in the application code, the application automatically calls the HandleException method. Unless the exception object is EAbort, HandleException calls the OnException handler, if one exists. Otherwise, it calls ShowException to display a message box indicating an error occurred.

See Also

Actions (see page 15122)

AfterDispatch (see page 15123)

TWebResponse (see page 15191)

77.1.20 TWebRequest Class

TWebRequest is the base class for all objects that represent client request information that is passed to a Web server application.

Class Hierarchy

Pascal

```
TWebRequest = class(TObject);
```

C++

```
class TWebRequest : public TObject;
```

File

HTTPApp

Description

Use a descendant of TWebRequest to interpret an HTTP request message.

Web application objects create TWebRequest objects to capture client requests. The properties of the TWebRequest objects are obtained from the HTTP request message. The Web application uses TWebRequest objects to create corresponding TWebResponse (see page 15191) objects. The TWebDispatcher (see page 15147) for the application then invokes the TWebActionItem (see page 15130) that is associated with the PathInfo (see page 15185) of the TWebRequest object.

TWebRequest has several descendants which represent client requests for different types of Web server applications. These descendants are

TApacheRequest objects for Apache server applications.

TCGIRequest objects for CGI server applications.

TISAPIRequest objects for DLLs that respond to the ISAPI or NSAPI servers (Windows only).

TWinCGIRequest objects for Windows-based CGI servers (Windows only).

See Also

TApacheRequest

TCGIRequest (see page 2564)

TISAPIRequest (see page 15662)

TWebActionItem (see page 15130)

TWebApplication (see page 24618)

TWebDispatcher (see page 15147)

TWebResponse (see page 15191)

TWinCGIRequest (see page 2583)

Creating HTTP Response Messages

Responding to Request Messages with Action Items

The Structure of a Web Broker Application

Dispatching Requests and Responses

77.1.20.1 TWebRequest Members

The following tables list the members exposed by TWebRequest.

Protected Methods

	Name	Description
	GetDateVariable (see page 15168)	Provides the abstract (pure virtual) prototype for a method to extract a date value from an indexed table of HTTP header variables.
	GetIntegerVariable (see page 15169)	Provides the abstract (pure virtual) prototype for a method to extract an integer value from an indexed table of HTTP header variables.

	GetInternalPathInfo (see page 15169)	This is GetInternalPathInfo, a member of class TWebRequest.
	GetInternalScriptName (see page 15169)	This is GetInternalScriptName, a member of class TWebRequest.
	GetStringVariable (see page 15169)	Provides the abstract (pure virtual) prototype for a method to extract a string value from an indexed table of HTTP header variables.
	UpdateMethodType (see page 15172)	Provides the abstract prototype for a method to write a pure virtual method for updating the client method type.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 15165)	Creates an instance of TWebRequest (see page 15161).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 15165)	Destroys an instance of TWebRequest (see page 15161).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExtractContentFields (see page 15166)	Parses the Content (see page 15176) property into its constituent fields.
	ExtractCookieFields (see page 15166)	Parses the Cookie (see page 15180) property into its constituent fields.
	ExtractFields (see page 15167)	Parses a multi-valued string into its constituent fields.
	ExtractQueryFields (see page 15167)	Parses the Query (see page 15186) property into its constituent fields.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetFieldByName (see page 15168)	Provides the abstract (pure virtual) prototype for a method that returns the value of the HTTP header variable specified by the Name parameter.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ReadClient (see page 15170)	Provides the abstract (pure virtual) prototype for a method to read from the HTTP request into a buffer.
	ReadString (see page 15171)	Provides the abstract (pure virtual) prototype for a method to read Count bytes from the HTTP request into a string.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	TranslateURI (see page 15171)	Provides the abstract (pure virtual) prototype for a method to translate a Uniform Resource Identifier (URI) into an internal representation of a location on the server.
	WriteClient (see page 15172)	Provides the abstract (pure virtual) prototype for a method to write Count bytes from Buffer back to the HTTP client.
	WriteHeaders (see page 15172)	Provides the abstract (pure virtual) prototype for a method to write an HTTP header string.
	WriteString (see page 15173)	Provides the abstract (pure virtual) prototype for a method to write the string AString back to the HTTP client.

Public Properties

	Name	Description
R	Accept (see page 15174)	Reports the value of the Accept header of the HTTP request message.
R	Authorization (see page 15175)	Reports the value of the Authorization header of the HTTP request message.
R	CacheControl (see page 15175)	Reports the value of the Cache-Control header of the HTTP request message.
R	Connection (see page 15176)	Reports the value of the Connection header of the HTTP request message.
R	Content (see page 15176)	Represents the unparsed contents of the HTTP request message.
R	ContentEncoding (see page 15177)	Reports the value of the Content (see page 15176)-Encoding header of the HTTP request message.
R	ContentFields (see page 15177)	Reports the parsed content of a POST request message.
R	ContentLength (see page 15178)	Reports the value of the Content (see page 15176)-Length header of the HTTP request message.
R	ContentParser (see page 15178)	Refers to the object's content parser object.
R	ContentType (see page 15179)	Reports the value of the Content (see page 15176)-Type header of the HTTP request message.
R	ContentVersion (see page 15179)	Reports the value of the Content (see page 15176)-Version header of the HTTP request message.
R	Cookie (see page 15180)	Reports the value of the Cookie header of the HTTP request message.
R	CookieFields (see page 15180)	Represents the parsed contents of the Cookie (see page 15180) header of an HTTP request message.
R	Date (see page 15181)	Reports the value of the Date header of the HTTP request message.
R	DerivedFrom (see page 15181)	Reports the value of the Derived-From (see page 15182) header of the HTTP request message.
R	Expires (see page 15182)	Reports the value of the Expires header of the HTTP request message.
R	Files (see page 15182)	Provides access to the files uploaded in the HTTP request.
R	From (see page 15182)	Reports the value of the From header of the HTTP request message.
R	Host (see page 15183)	Reports the value of the Host header of the HTTP request message.
R	IfModifiedSince (see page 15183)	Reports the value of the If-Modified-Since header of the HTTP request message.
R	InternalPathInfo (see page 15184)	Returns the path info portion of the URL (see page 15190).
R	InternalScriptName (see page 15184)	Returns the ScriptName (see page 15189) part of the URL (see page 15190).
R	Method (see page 15185)	This is Method, a member of class TWebRequest.
R	MethodType (see page 15185)	This is MethodType, a member of class TWebRequest.
R	PathInfo (see page 15185)	Reports the value of the path information (if any) of the URL (see page 15190) specified in the HTTP request message.
R	PathTranslated (see page 15186)	Represents a translation of the PathInfo (see page 15185) property to a fully qualified path on the Web server.
R	ProtocolVersion (see page 15186)	Reports the version number of the HTTP protocol specified in the initial header of the HTTP request message.
R	Query (see page 15186)	Indicates the value of the query information (if any) of the URL (see page 15190) specified in the HTTP request message.
R	QueryFields (see page 15187)	Represents the parsed contents of the Query (see page 15186) in the URL (see page 15190) of the HTTP request message.
R	Referer (see page 15188)	Reports the value of the Referer header of the HTTP request message.
R	RemoteAddr (see page 15188)	Indicates the remote IP address of the client associated with the HTTP request message.
R	RemoteHost (see page 15188)	Reports the fully qualified reverse domain lookup of the client associated with the HTTP request message.
R	ScriptName (see page 15189)	Indicates the value of the script information (if any) of the URL (see page 15190) specified in the HTTP request message.
R	ServerPort (see page 15189)	Indicates the port number of the port on the Web server that receives the HTTP request message.
R	Title (see page 15190)	Reports the value of the Title header of the HTTP request message.
R	URL (see page 15190)	Reports the Uniform Resource Identifier (URI) specified by the HTTP request message to identify the target of the request.
R	UserAgent (see page 15191)	Reports the value of the User-Agent header of the HTTP request message.

77.1.20.2 TWebRequest.Create Constructor

Creates an instance of TWebRequest (see page 15161).

Pascal

```
constructor Create;
```

C++

```
__fastcall TWebRequest();
```

Description

Applications shouldn't need to call Create to instantiate a request object. The TWebApplication object creates a TWebRequest (see page 15161) object whenever it receives a client request.

See Also

TWebApplication (see page 24618)

77.1.20.3 TWebRequest.Destroy Destructor

Destroys an instance of TWebRequest (see page 15161).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebRequest();
```

Description

Do not call Destroy. When the request message is fully handled, TWebApplication destroys the request object by calling its Free method.

Destroy frees the TStrings objects used to represent the ContentFields (see page 15177), CookieFields (see page 15180), and QueryFields (see page 15187) properties.

See Also

ContentFields (see page 15177)

CookieFields (see page 15180)

Free (see page 22300)

QueryFields (see page 15187)

TWebApplication (see page 24618)

77.1.20.4 TWebRequest Methods

The methods of the TWebRequest class are listed here.

Protected Methods

	Name	Description
	GetDateVariable (see page 15168)	Provides the abstract (pure virtual) prototype for a method to extract a date value from an indexed table of HTTP header variables.

	GetIntegerVariable (see page 15169)	Provides the abstract (pure virtual) prototype for a method to extract an integer value from an indexed table of HTTP header variables.
	GetInternalPathInfo (see page 15169)	This is GetInternalPathInfo, a member of class TWebRequest.
	GetInternalScriptName (see page 15169)	This is GetInternalScriptName, a member of class TWebRequest.
	GetStringVariable (see page 15169)	Provides the abstract (pure virtual) prototype for a method to extract a string value from an indexed table of HTTP header variables.
	UpdateMethodType (see page 15172)	Provides the abstract prototype for a method to write a pure virtual method for updating the client method type.

Public Methods

	Name	Description
	ExtractContentFields (see page 15166)	Parses the Content (see page 15176) property into its constituent fields.
	ExtractCookieFields (see page 15166)	Parses the Cookie (see page 15180) property into its constituent fields.
	ExtractFields (see page 15167)	Parses a multi-valued string into its constituent fields.
	ExtractQueryFields (see page 15167)	Parses the Query (see page 15186) property into its constituent fields.
	GetFieldByName (see page 15168)	Provides the abstract (pure virtual) prototype for a method that returns the value of the HTTP header variable specified by the Name parameter.
	ReadClient (see page 15170)	Provides the abstract (pure virtual) prototype for a method to read from the HTTP request into a buffer.
	ReadString (see page 15171)	Provides the abstract (pure virtual) prototype for a method to read Count bytes from the HTTP request into a string.
	TranslateURI (see page 15171)	Provides the abstract (pure virtual) prototype for a method to translate a Uniform Resource Identifier (URI) into an internal representation of a location on the server.
	WriteClient (see page 15172)	Provides the abstract (pure virtual) prototype for a method to write Count bytes from Buffer back to the HTTP client.
	WriteHeaders (see page 15172)	Provides the abstract (pure virtual) prototype for a method to write an HTTP header string.
	WriteString (see page 15173)	Provides the abstract (pure virtual) prototype for a method to write the string AString back to the HTTP client.

77.1.20.4.1 TWebRequest.ExtractContentFields Method

Parses the Content (see page 15176) property into its constituent fields.

Pascal

```
procedure ExtractContentFields(Strings: TStrings);
```

C++

```
__fastcall ExtractContentFields(TStrings Strings);
```

Description

Call ExtractContentFields to fill the Strings parameter with the individual fields in the Content (see page 15176) property. ExtractContentFields assumes that fields are separated by ampersands (&). This is the format used when the MethodType (see page 15185) is mtPost. If the Content (see page 15176) property is not assigned, ExtractContentFields does nothing.

See Also

Content (see page 15176)

ContentFields (see page 15177)

MethodType (see page 15185)

TStrings (see page 3029)

77.1.20.4.2 TWebRequest.ExtractCookieFields Method

Parses the Cookie (see page 15180) property into its constituent fields.

Pascal

```
procedure ExtractCookieFields(Strings: TStrings);
```

C++

```
__fastcall ExtractCookieFields(TStrings Strings);
```

Description

Call ExtractCookieFields to fill the Strings parameter with the individual fields in the Cookie (see page 15180) property. ExtractCookieFields assumes that fields are separated by semicolons. If the Cookie (see page 15180) property is not assigned, ExtractCookieFields does nothing.

See Also

Cookie (see page 15180)

CookieFields (see page 15180)

TStrings (see page 3029)

77.1.20.4.3 TWebRequest.ExtractFields Method

Parses a multi-valued string into its constituent fields.

Pascal

```
procedure ExtractFields(Separators: TSysCharSet; WhiteSpace: TSysCharSet; Content: PChar;
Strings: TStrings);
```

C++

```
__fastcall ExtractFields(TSysCharSet Separators, TSysCharSet WhiteSpace, const char * Content,
TStrings Strings);
```

Description

Call ExtractFields to parse a multi-valued HTTP header into separate strings for each value. For example, if the separator character of the UserAgent (see page 15191) property is known, ExtractFields can parse this property into separate UserAgent (see page 15191) values.

Separators is a set of characters that are used to separate individual values within the multi-valued string. WhiteSpace is a set of characters that are to be ignored when parsing the string. Content (see page 15176) is the multi-valued string to be parsed. Strings is the TStrings object that receives the individual values that are parsed from Content (see page 15176).

See Also

ExtractContentFields (see page 15166)

ExtractCookieFields (see page 15166)

ExtractQueryFields (see page 15167)

TStrings (see page 3029)

77.1.20.4.4 TWebRequest.ExtractQueryFields Method

Parses the Query (see page 15186) property into its constituent fields.

Pascal

```
procedure ExtractQueryFields(Strings: TStrings);
```

C++

```
__fastcall ExtractQueryFields(TStrings Strings);
```

Description

Call ExtractQueryFields to fill the Strings parameter with the individual fields in the Query (see page 15186) property. ExtractQueryFields assumes that fields are separated by ampersands (&). If the Query (see page 15186) property is not assigned, ExtractQueryFields does nothing.

See Also

Query (see page 15186)

QueryFields (see page 15187)

TStrings (see page 3029)

77.1.20.4.5 TWebRequest.GetDateVariable Method

Provides the abstract (pure virtual) prototype for a method to extract a date value from an indexed table of HTTP header variables.

Pascal

```
function GetDateVariable(Index: Integer): TDateTime; virtual; abstract;
```

C++

```
virtual __fastcall TDateTime GetDateVariable(int Index) = 0;
```

Description

Descendants of TWebRequest (see page 15161) implement GetDateVariable to provide a method by which the date-time values are read from the header strings of the HTTP request message. The request object uses GetDateVariable to access the properties that represent dates from the HTTP request headers.

See Also

Date (see page 15181)

Expires (see page 15182)

IfModifiedSince (see page 15183)

77.1.20.4.6 TWebRequest.GetFieldByName Method

Provides the abstract (pure virtual) prototype for a method that returns the value of the HTTP header variable specified by the Name parameter.

Pascal

```
function GetFieldByName(const Name: string): string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetFieldByName(const AnsiString Name) = 0;
```

Description

Descendants of TWebRequest (see page 15161) implement GetFieldByName to provide access to arbitrary header fields in the HTTP request message. Use GetFieldByName to read the value of an HTTP header variable that is not surfaced as one of the properties of TWebRequest (see page 15161). For information on possible HTTP request header variables, see the current Internet standards documents.

See Also

GetStringVariable (see page 15169)

Properties That Contain Request Header Information

77.1.20.4.7 TWebRequest.GetIntegerVariable Method

Provides the abstract (pure virtual) prototype for a method to extract an integer value from an indexed table of HTTP header variables.

Pascal

```
function GetIntegerVariable(Index: Integer): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetIntegerVariable(int Index) = 0;
```

Description

Descendants of TWebRequest (see page 15161) implement GetIntegerVariable to provide a method by which the integer values are read from the header strings of the HTTP request message. The request object uses GetIntegerVariable to access the properties that represent integers from the HTTP request headers.

See Also

ContentLength (see page 15178)

ServerPort (see page 15189)

77.1.20.4.8 TWebRequest.GetInternalPathInfo Method

Pascal

```
function GetInternalPathInfo: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetInternalPathInfo();
```

Description

This is GetInternalPathInfo, a member of class TWebRequest.

77.1.20.4.9 TWebRequest.GetInternalScriptName Method

Pascal

```
function GetInternalScriptName: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetInternalScriptName();
```

Description

This is GetInternalScriptName, a member of class TWebRequest.

77.1.20.4.10 TWebRequest.GetStringVariable Method

Provides the abstract (pure virtual) prototype for a method to extract a string value from an indexed table of HTTP header variables.

Pascal

```
function GetStringVariable(Index: Integer): string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetStringVariable(int Index) = 0;
```

Description

Descendants of TWebRequest (see page 15161) implement GetStringVariable to provide a method by which the string values are read from the header strings of the HTTP request message. The request object uses GetStringVariable to access the properties that represent strings from the HTTP request headers.

See Also

Accept (see page 15174)

CacheControl (see page 15175)

Connection (see page 15176)

Content (see page 15176)

ContentEncoding (see page 15177)

ContentType (see page 15179)

ContentVersion (see page 15179)

Cookie (see page 15180)

DerivedFrom (see page 15181)

From (see page 15182)

Host (see page 15183)

TWebRequest (see page 15161)

PathInfo (see page 15185)

PathTranslated (see page 15186)

ProtocolVersion (see page 15186)

Query (see page 15186)

Referer (see page 15188)

RemoteAddr (see page 15188)

RemoteHost (see page 15188)

ScriptName (see page 15189)

Title (see page 15190)

URL (see page 15190)

UserAgent (see page 15191)

77.1.20.4.11 TWebRequest.ReadClient Method

Provides the abstract (pure virtual) prototype for a method to read from the HTTP request into a buffer.

Pascal

```
function ReadClient(var Buffer; Count: Integer): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int ReadClient( Buffer, int Count) = 0;
```

Description

Descendants of TWebRequest (see page 15161) implement ReadClient. ReadClient is used to read content information from the HTTP request. Use ReadClient to get the next chunk of information when the content of the request message is too large for the Content (see page 15176) property.

Use Buffer to provide a buffer to contain the content information. Use Count to specify the number of bytes to read. ReadClient returns the number of bytes actually read.

See Also

Content (see page 15176)

ReadString (see page 15171)

WriteClient (see page 15172)

77.1.20.4.12 TWebRequest.ReadString Method

Provides the abstract (pure virtual) prototype for a method to read Count bytes from the HTTP request into a string.

Pascal

```
function ReadString(Count: Integer): string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString ReadString(int Count) = 0;
```

Description

Descendants of TWebRequest (see page 15161) implement ReadString. ReadString is used to load content information from the HTTP request into a string. Use ReadString to get the next chunk of information when the content of the request message is too large for the Content (see page 15176) property.

See Also

Content (see page 15176)

ReadClient (see page 15170)

WriteString (see page 15173)

77.1.20.4.13 TWebRequest.TranslateURI Method

Provides the abstract (pure virtual) prototype for a method to translate a Uniform Resource Identifier (URI) into an internal representation of a location on the server.

Pascal

```
function TranslateURI(const URI: string): string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString TranslateURI(const AnsiString URI) = 0;
```

Description

Descendants of TWebRequest (see page 15161) implement TranslateURI. TranslateURI is used to translate the URI specified in a client request into a format that can be used by the server application. TranslateURI is not relevant to CGI applications.

Note: On Windows, the internal format varies, depending on whether the server application is an ISAPI or NSAPI application.

See Also

PathTranslated (see page 15186)

URL (see page 15190)

77.1.20.4.14 TWebRequest.UpdateMethodType Method

Provides the abstract prototype for a method to write a pure virtual method for updating the client method type.

Pascal

```
procedure UpdateMethodType;
```

C++

```
__fastcall UpdateMethodType();
```

Description

Use UpdateMethodType to update the type of response that is requested by the Web client.

See Also

TWebRequest (see page 15161)

77.1.20.4.15 TWebRequest.WriteClient Method

Provides the abstract (pure virtual) prototype for a method to write Count bytes from Buffer back to the HTTP client.

Pascal

```
function WriteClient(var Buffer; Count: Integer): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int WriteClient( Buffer, int Count) = 0;
```

Description

Descendants of TWebRequest (see page 15161) implement WriteClient. WriteClient is used by TWebResponse (see page 15191) to send the contents of a response to the Web client.

See Also

ReadClient (see page 15170)

TWebResponse (see page 15191)

WriteString (see page 15173)

77.1.20.4.16 TWebRequest.WriteHeaders Method

Provides the abstract (pure virtual) prototype for a method to write an HTTP header string.

Pascal

```
function WriteHeaders(StatusCode: Integer; const ReasonString: string; const Headers: string):
Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean WriteHeaders(int StatusCode, const AnsiString ReasonString, const
AnsiString Headers) = 0;
```

Description

Use WriteHeaders to write an HTTP header string.

See Also

WriteClient (see page 15172)

WriteString (see page 15173)

77.1.20.4.17 TWebRequest.WriteString Method

Provides the abstract (pure virtual) prototype for a method to write the string AString back to the HTTP client.

Pascal

```
function WriteString(const AString: string): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean WriteString(const AnsiString AString) = 0;
```

Description

Descendants of TWebRequest (see page 15161) implement WriteString. WriteString is used by TWebResponse (see page 15191) to send the contents of a response to the Web client.

See Also

ReadString (see page 15171)

TWebResponse (see page 15191)

WriteClient (see page 15172)

77.1.20.5 TWebRequest Properties

The properties of the TWebRequest class are listed here.

Public Properties

	Name	Description
R	Accept (see page 15174)	Reports the value of the Accept header of the HTTP request message.
R	Authorization (see page 15175)	Reports the value of the Authorization header of the HTTP request message.
R	CacheControl (see page 15175)	Reports the value of the Cache-Control header of the HTTP request message.
R	Connection (see page 15176)	Reports the value of the Connection header of the HTTP request message.
R	Content (see page 15176)	Represents the unparsed contents of the HTTP request message.
R	ContentEncoding (see page 15177)	Reports the value of the Content (see page 15176)-Encoding header of the HTTP request message.
R	ContentFields (see page 15177)	Reports the parsed content of a POST request message.
R	ContentLength (see page 15178)	Reports the value of the Content (see page 15176)-Length header of the HTTP request message.
R	ContentParser (see page 15178)	Refers to the object's content parser object.

R	ContentType (see page 15179)	Reports the value of the Content (see page 15176)-Type header of the HTTP request message.
R	ContentVersion (see page 15179)	Reports the value of the Content (see page 15176)-Version header of the HTTP request message.
R	Cookie (see page 15180)	Reports the value of the Cookie header of the HTTP request message.
R	CookieFields (see page 15180)	Represents the parsed contents of the Cookie (see page 15180) header of an HTTP request message.
R	Date (see page 15181)	Reports the value of the Date header of the HTTP request message.
R	DerivedFrom (see page 15181)	Reports the value of the Derived-From (see page 15182) header of the HTTP request message.
R	Expires (see page 15182)	Reports the value of the Expires header of the HTTP request message.
R	Files (see page 15182)	Provides access to the files uploaded in the HTTP request.
R	From (see page 15182)	Reports the value of the From header of the HTTP request message.
R	Host (see page 15183)	Reports the value of the Host header of the HTTP request message.
R	IfModifiedSince (see page 15183)	Reports the value of the If-Modified-Since header of the HTTP request message.
R	InternalPathInfo (see page 15184)	Returns the path info portion of the URL (see page 15190).
R	InternalScriptName (see page 15184)	Returns the ScriptName (see page 15189) part of the URL (see page 15190).
R	Method (see page 15185)	This is Method, a member of class TWebRequest.
R	MethodType (see page 15185)	This is MethodType, a member of class TWebRequest.
R	PathInfo (see page 15185)	Reports the value of the path information (if any) of the URL (see page 15190) specified in the HTTP request message.
R	PathTranslated (see page 15186)	Represents a translation of the PathInfo (see page 15185) property to a fully qualified path on the Web server.
R	ProtocolVersion (see page 15186)	Reports the version number of the HTTP protocol specified in the initial header of the HTTP request message.
R	Query (see page 15186)	Indicates the value of the query information (if any) of the URL (see page 15190) specified in the HTTP request message.
R	QueryFields (see page 15187)	Represents the parsed contents of the Query (see page 15186) in the URL (see page 15190) of the HTTP request message.
R	Referer (see page 15188)	Reports the value of the Referer header of the HTTP request message.
R	RemoteAddr (see page 15188)	Indicates the remote IP address of the client associated with the HTTP request message.
R	RemoteHost (see page 15188)	Reports the fully qualified reverse domain lookup of the client associated with the HTTP request message.
R	ScriptName (see page 15189)	Indicates the value of the script information (if any) of the URL (see page 15190) specified in the HTTP request message.
R	ServerPort (see page 15189)	Indicates the port number of the port on the Web server that receives the HTTP request message.
R	Title (see page 15190)	Reports the value of the Title header of the HTTP request message.
R	URL (see page 15190)	Reports the Uniform Resource Identifier (URI) specified by the HTTP request message to identify the target of the request.
R	UserAgent (see page 15191)	Reports the value of the User-Agent header of the HTTP request message.

77.1.20.5.1 TWebRequest.Accept Property

Reports the value of the Accept header of the HTTP request message.

Pascal

```
property Accept: string;
```

C++

```
__property AnsiString Accept;
```

Description

Read Accept to determine the media types the Web client can accept in response to the request. Not every HTTP request message has an Accept header. If the Accept property is not set, then any media type is potentially acceptable in the response.

Accept headers usually indicate that the response should be limited to a small set of desired types. For example, a Web client

requesting an audio response might produce a TWebRequest (see page 15161) object with Accept set to the following string:

```
audio/*; q=0.2, audio/basic
```

This request indicates that the preferred response media type is audio/basic, but that any audio type is acceptable if it is the best available after an 80% markdown in quality.

For information on the format of Accept headers, see the current Internet standards documentation.

See Also

ContentType (see page 15209)

Properties That Describe the Expected Response

77.1.20.5.2 TWebRequest.Authorization Property

Reports the value of the Authorization header of the HTTP request message.

Pascal

```
property Authorization: string;
```

C++

```
__property AnsiString Authorization;
```

Description

Read Authorization to check whether the client that sent the request message has provided the correct authentication information. Authorization allows the Web application to provided limited access to the resource specified by the URL (see page 15190) property.

For more information about HTTP authentication schemes, see the Internet standards documentation.

See Also

WWWAuthenticate (see page 15218)

URL (see page 15190)

77.1.20.5.3 TWebRequest.CacheControl Property

Reports the value of the Cache-Control header of the HTTP request message.

Pascal

```
property CacheControl: string;
```

C++

```
__property AnsiString CacheControl;
```

Description

Read CacheControl to determine any Web client directives that must be obeyed by all caching mechanisms along the request/response chain. The CacheControl directives of the HTTP request need not apply to the corresponding HTTP response.

The HTTP 1.1 standard defines the following directives for CacheControl:

Value	Meaning
no-cache	The request may not be cached. The Web server validates the request before the TWebRequest (see page 15161) object is created.

no-store	The request contains sensitive information and may not be stored on any permanent medium such as a backup tape. If the request is stored temporarily in a file, the file must be deleted.
max-age=ss	The request is considered invalid if it is older than ss seconds. This directive takes precedence over the Expires (see page 15182) property, even if Expires (see page 15182) is more restrictive. Invalid requests are validated by the Web server before the TWebRequest (see page 15161) object is created.
max-stale or max-stale=ss	The client is willing to accept a response that has exceeded its expiration time. The max-stale directive can optionally specify the number of seconds by which the response can exceed its expiration time.
min-fresh=ss	The client is not willing to accept a response that expires in less than ss seconds.

See Also

Date (see page 15181)

Expires (see page 15182)

77.1.20.5.4 TWebRequest.Connection Property

Reports the value of the Connection header of the HTTP request message.

Pascal

```
property Connection: string;
```

C++

```
__property AnsiString Connection;
```

Description

Read Connection to determine the options requested by the Web client for its connection to the server application. The HTTP 1.1 standard specifies only one value for the Connection header. If the value of Connection is close, the connection should be closed after the completion of the response.

See Also

Host (see page 15183)

RemoteAddr (see page 15188)

RemoteHost (see page 15188)

ServerPort (see page 15189)

Connecting to a Data Store Using TADOConnection

77.1.20.5.5 TWebRequest.Content Property

Represents the unparsed contents of the HTTP request message.

Pascal

```
property Content: string;
```

C++

```
__property AnsiString Content;
```

Description

Use Content to read the contents of an HTTP request message that is sending information to the server. The value of Content depends on the Method (see page 15185) of the HTTP request. When the Method (see page 15185) is POST, Content is

the unparsed string giving the values to be posted with their associated fields. When the Method (see page 15185) is PUT, Content is the value that should replace the URL (see page 15190) given by the URL (see page 15190) property.

Tip: When the Method (see page 15185) is POST, the parsed version of the post request is available in the ContentFields (see page 15177) property.

Note: If the content of a message is particularly large, it is transferred in chunks. In this case, the Content property does not contain the entire message content. To read the rest of the content, use the ReadClient (see page 15170) method.

See Also

ContentEncoding (see page 15177)

ContentFields (see page 15177)

ContentLength (see page 15178)

ContentType (see page 15179)

ContentVersion (see page 15179)

ExtractContentFields (see page 15166)

TWebRequest (see page 15161)

The Content of HTTP Request Messages

77.1.20.5.6 TWebRequest.ContentEncoding Property

Reports the value of the Content (see page 15176)-Encoding header of the HTTP request message.

Pascal

```
property ContentEncoding: string;
```

C++

```
__property AnsiString ContentEncoding;
```

Description

Read ContentEncoding to determine what encoding mechanisms, if any, have been applied to the Content (see page 15176) of the HTTP request. The encoding mechanisms are listed in the order in which the Web client applied them, separated by commas. Typically, ContentEncoding indicates a compression method or an encoding scheme used to transfer extended ASCII characters over the net.

Reverse the ContentEncoding mechanisms to decode the Content (see page 15176) into the form indicated by the ContentType (see page 15179) property.

See Also

Content (see page 15176)

ContentType (see page 15179)

Properties That Describe the Content

77.1.20.5.7 TWebRequest.ContentFields Property

Reports the parsed content of a POST request message.

Pascal

```
property ContentFields: TStrings;
```

C++

```
__property TStrings ContentFields;
```

Description

Use ContentFields to read the fields of an HTTP request message when the MethodType (see page 15185) is mtPost. The Content (see page 15176) of the HTTP request is parsed, separating into fields at the ampersand character.

Each field appears as a separate string in the ContentFields list with the form Name=Value. As a TStrings object, ContentFields allows applications to manipulate the individual fields by name or value.

See Also

Content (see page 15176)

ExtractContentFields (see page 15166)

TWebRequest (see page 15161)

TStrings (see page 3029)

Names (see page 3075)

Values (see page 3086)

The Content of HTTP Request Messages

77.1.20.5.8 TWebRequest.ContentLength Property

Reports the value of the Content (see page 15176)-Length header of the HTTP request message.

Pascal

```
property ContentLength: Integer;
```

C++

```
__property int ContentLength;
```

Description

Read the ContentLength property to determine the length of the Content (see page 15176) property. ContentLength allows an application to determine the end of the Content (see page 15176) property when the ContentType (see page 15179) is not text/html. A ContentLength of 0 indicates that the length of the Content (see page 15176) property is not determined, either because the request does not include any Content (see page 15176), or because the Web client did not know the length.

Note: The length of the Content (see page 15176) may be less than ContentLength when the Content (see page 15176) is passed using chunked transfer coding.

See Also

Content (see page 15176)

ContentType (see page 15179)

TWebRequest (see page 15161)

Properties That Describe the Content

77.1.20.5.9 TWebRequest.ContentParser Property

Refers to the object's content parser object.

Pascal

```
property ContentParser: TAbstractContentParser;
```

C++

```
__property TAbstractContentParser ContentParser;
```

Description

Read ContentParser to locate the TWebRequest (see page 15161) instance's TAbstractContentParser (see page 15039) object. The TAbstractContentParser (see page 15039) is responsible for parsing the HTTP request.

77.1.20.5.10 TWebRequest.ContentType Property

Reports the value of the Content (see page 15176)-Type header of the HTTP request message.

Pascal

```
property ContentType: string;
```

C++

```
__property AnsiString ContentType;
```

Description

Read ContentType to determine the media type of the Content (see page 15176) of the HTTP request message. The ContentType is a string of the form Type/SubType.

Most HTTP requests do not require any Content (see page 15176) at all. For most requests with a Method (see page 15185) that requires a value for Content (see page 15176), the ContentType is text/html. However, HTTP requests for server-specific values of Method (see page 15185) can use any media type appropriate to the request.

See Also

Accept (see page 15174)

Content (see page 15176)

TWebRequest (see page 15161)

Properties That Describe the Content

77.1.20.5.11 TWebRequest.ContentVersion Property

Reports the value of the Content (see page 15176)-Version header of the HTTP request message.

Pascal

```
property ContentVersion: string;
```

C++

```
__property AnsiString ContentVersion;
```

Description

Read ContentVersion to determine the version of the Content (see page 15176) of the HTTP request message. Most HTTP requests do not require any Content (see page 15176) at all. When the Method (see page 15185) requires a value for Content (see page 15176), the ContentVersion may be set to indicate the version of the source of that content. The format of the ContentVersion depends on the ContentType (see page 15179) of the request.

See Also

Content (see page 15176)

ContentType (see page 15179)

Properties That Describe the Content

77.1.20.5.12 TWebRequest.Cookie Property

Reports the value of the Cookie header of the HTTP request message.

Pascal

```
property Cookie: string;
```

C++

```
__property AnsiString Cookie;
```

Description

Use Cookie to read the "cookie" strings provided in the Web client request. Cookie strings are used to track state information in a Web application. Cookie is the unparsed value of the Cookie header. The Cookie header can include multiple fields separated by semicolons. Use the CookieFields (see page 15180) property to work with the Cookie header on a field-by-field basis.

See Also

CookieFields (see page 15180)

ExtractCookieFields (see page 15166)

Cookies (see page 15210)

Properties That Describe the Expected Response

77.1.20.5.13 TWebRequest.CookieFields Property

Represents the parsed contents of the Cookie (see page 15180) header of an HTTP request message.

Pascal

```
property CookieFields: TStrings;
```

C++

```
__property TStrings CookieFields;
```

Description

Use CookieFields to work with individual "cookie" strings provided by the Web client request. The Cookie (see page 15180) property is parsed into separate fields by separating into substrings whenever a semicolon is encountered.

Cookie (see page 15180) strings are used to track state information in a Web application. Each field appears as a separate string in the CookieFields list with the form Name=Value. As a TStrings object, CookieFields allows applications to manipulate the individual fields by name or value.

Note: Applications can give state variables any name except one of the reserved cookie field names. See the Netscape documentation for a list of reserved cookie field names and their meanings.

See Also

Cookie (see page 15180)

ExtractCookieFields (see page 15166)

TStrings (see page 3029)

Names (see page 3075)

Values (see page 3086)

77.1.20.5.14 TWebRequest.Date Property

Reports the value of the Date header of the HTTP request message.

Pascal

```
property Date: TDateTime;
```

C++

```
__property TDateTime Date;
```

Description

Read Date to determine the date and time at which the HTTP request message originated. Date can be used to determine whether the Content (see page 15176) of the request is still valid. The value of Date is translated into local time as specified by the system clock from the time units set by the Web client.

For example, if an HTTP request message specifies a MethodType (see page 15185) of mtPost, and Date is older than the most recent values of the fields that are being posted, the server application may return a status code that indicates an error rather than overwriting more recent values.

Date is represented using the TDateTime type. The integral part of a TDateTime value is the number of days that have passed since Dec. 30 1899. The fractional part of a TDateTime value is the fraction of a 24-hour day that has elapsed.

See Also

Expires (see page 15182)

IfModifiedSince (see page 15183)

MethodType (see page 15185)

StatusCode (see page 15215)

77.1.20.5.15 TWebRequest.DerivedFrom Property

Reports the value of the Derived-From (see page 15182) header of the HTTP request message.

Pascal

```
property DerivedFrom: string;
```

C++

```
__property AnsiString DerivedFrom;
```

Description

Read DerivedFrom to determine the URI from which the value of Content (see page 15176) originated. DerivedFrom is only meaningful when the request includes a Content (see page 15176) property and the Content (see page 15176) did not originate with the Web client.

See Also

Content (see page 15176)

Referer (see page 15188)

Properties That Describe the Web Client

77.1.20.5.16 TWebRequest.Expires Property

Reports the value of the Expires header of the HTTP request message.

Pascal

```
property Expires: TDateTime;
```

C++

```
__property TDateTime Expires;
```

Description

Use Expires to determine whether the request is "stale." Expires is the date and time at which the request becomes invalid. The value of Expires is translated into local time as specified by the system clock from the time units set by the Web client.

If the request has a CacheControl (see page 15175) value with a max-age directive, the max-age directive overrides the value of Expires.

Expires is represented using the TDateTime type. The integral part of a TDateTime value is the number of days that have passed since Dec. 30, 1899. The fractional part of a TDateTime value is the fraction of a 24-hour day that has elapsed.

See Also

Date (see page 15181)

IfModifiedSince (see page 15183)

CacheControl (see page 15175)

77.1.20.5.17 TWebRequest.Files Property

Provides access to the files uploaded in the HTTP request.

Pascal

```
property Files: TAbstractWebRequestFiles;
```

C++

```
__property TAbstractWebRequestFiles Files;
```

Description

Use this property to find the specific instance of the object's TAbstractRequestFiles object. The TAbstractRequestFiles object contains uploaded files extracted from the HTTP request.

77.1.20.5.18 TWebRequest.From Property

Reports the value of the From header of the HTTP request message.

Pascal

```
property From: string;
```

C++

```
__property AnsiString From;
```

Description

Read From to determine the Internet e-mail address of the human user for whom the Web client sent the HTTP request message. This value can be used for logging purposes, or to contact the originator of the request if problems occur.

See Also

RemoteAddr (see page 15188)

RemoteHost (see page 15188)

UserAgent (see page 15191)

Properties That Describe the Web Client

77.1.20.5.19 TWebRequest.Host Property

Reports the value of the Host header of the HTTP request message.

Pascal

```
property Host: string;
```

C++

```
__property AnsiString Host;
```

Description

Read Host to determine the Internet host and port number of resource being requested. For example, if the client request specifies the following URL (see page 15190):

`http://www.TSite.com/art/gallery.cgi/mammals?animal=dog&color=black`

The value of Host would be

`www.TSite.com`

The value of Host can be used to differentiate between internally ambiguous URLs, such as the root "/" URL (see page 15190) of a server for multiple host names on a single IP address.

See Also

PathInfo (see page 15185)

ScriptName (see page 15189)

ServerPort (see page 15189)

URL (see page 15190)

Properties That Identify the Target

77.1.20.5.20 TWebRequest.IfModifiedSince Property

Reports the value of the If-Modified-Since header of the HTTP request message.

Pascal

```
property IfModifiedSince: TDateTime;
```

C++

```
__property TDateTime IfModifiedSince;
```

Description

Read IfModifiedSince on a request with a MethodType (see page 15185) of mtGet to determine whether the request is conditional. If the information requested by the Web client was not modified since the time indicated by IfModifiedSince, the response should be a not modified error rather than the information indicated by the URL (see page 15190) property.

The value of IfModifiedSince is given according to the local clock, not the clock of the Web client.

Note: On Windows, the value of the local clock is specified in the Windows Control Panel.

IfModifiedSince is represented using the TDateTime type. The integral part of a TDateTime value is the number of days that have passed since Dec. 30, 1899. The fractional part of a TDateTime value is the fraction of a 24-hour day that has elapsed.

See Also

Expires (see page 15182)

Date (see page 15181)

Properties That Describe the Expected Response

77.1.20.5.21 TWebRequest.InternalPathInfo Property

Returns the path info portion of the URL (see page 15190).

Pascal

```
property InternalPathInfo: string;
```

C++

```
__property AnsiString InternalPathInfo;
```

Description

InternalPathInfo contains the path information portion of the URL (see page 15190), which identifies the destination of the message. This property generally contains the same information as PathInfo.To ensure cross-platform code compatibility, use InternalPathInfo instead of PathInfo (see page 15185).

Note: On Windows, this property varies slightly from PathInfo (see page 15185) if the application has an embedded test server. In that case, information specific to the test server is contained in the InternalScriptName (see page 15184) instead of the InternalPathInfo.

See Also

PathInfo (see page 15185)

77.1.20.5.22 TWebRequest.InternalScriptName Property

Returns the ScriptName (see page 15189) part of the URL (see page 15190).

Pascal

```
property InternalScriptName: string;
```

C++

```
__property AnsiString InternalScriptName;
```

Description

InternalScriptName contains the script name portion of the URL (see page 15190), which specifies the name of a Web server application. This property generally contains the same information as ScriptName (see page 15189). To ensure cross platform code compatibility, use InternalScriptName instead of ScriptName (see page 15189).

Note: On Windows, this property varies slightly from ScriptName (see page 15189) if the application has an embedded test server. In that case, information specific to the test server is contained in the InternalScriptName instead of the InternalPathInfo (see page 15184).

See Also

ScriptName (see page 15189)

77.1.20.5.23 TWebRequest.Method Property

Pascal

```
property Method: string;
```

C++

```
__property AnsiString Method;
```

Description

This is Method, a member of class TWebRequest.

77.1.20.5.24 TWebRequest.MethodType Property

Pascal

```
property MethodType: TMethodType;
```

C++

```
__property TMethodType MethodType;
```

Description

This is MethodType, a member of class TWebRequest.

77.1.20.5.25 TWebRequest.PathInfo Property

Reports the value of the path information (if any) of the URL (see page 15190) specified in the HTTP request message.

Pascal

```
property PathInfo: string;
```

C++

```
__property AnsiString PathInfo;
```

Description

Read PathInfo to determine the path on the server machine where the target of the request is found. For example, given this URL (see page 15190):

`http://www.TSite.com/art/gallery.cgi/mammals?animal=dog&color=black`

the PathInfo portion is

`/mammals`

The TWebDispatcher (see page 15147) object uses the PathInfo to identify the appropriate TWebActionItem (see page 15130) for servicing the request message.

See Also

Host (see page 15183)

PathTranslated (see page 15186)

Query (see page 15186)

ScriptName (see page 15189)

TWebActionItem (see page 15130)

TWebDispatcher (see page 15147)

InternalPathInfo (see page 15184)

URL (see page 15190)

Dispatching Request Messages

Properties That Identify the Target

77.1.20.5.26 TWebRequest.PathTranslated Property

Represents a translation of the PathInfo (see page 15185) property to a fully qualified path on the Web server.

Pascal

```
property PathTranslated: string;
```

C++

```
__property AnsiString PathTranslated;
```

Description

Use PathTranslated to refer to the location of the target of the request in an HTTP response.

See Also

Host (see page 15183)

PathInfo (see page 15185)

TWebActionItem (see page 15130)

TWebResponse (see page 15191)

URL (see page 15190)

77.1.20.5.27 TWebRequest.ProtocolVersion Property

Reports the version number of the HTTP protocol specified in the initial header of the HTTP request message.

Pascal

```
property ProtocolVersion: string;
```

C++

```
__property AnsiString ProtocolVersion;
```

Description

Use ProtocolVersion to determine the protocol used by the Web client when generating the HTTP request message.

Some properties of the HTTP request may be interpreted differently depending on the ProtocolVersion. For example, if ProtocolVersion is 1.0, then the connection should not be considered persistent, even if the Connection (see page 15176) property does not specify the close option.

See Also

Connection (see page 15176)

77.1.20.5.28 TWebRequest.Query Property

Indicates the value of the query information (if any) of the URL (see page 15190) specified in the HTTP request message.

Pascal

```
property Query: string;
```

C++

```
__property AnsiString Query;
```

Description

Read Query to determine the details of a query specified in the URL (see page 15190) of the Web client request. For example, given this URL (see page 15190):

`http://www.TSite.com/art/gallery.cgi/mammals?animal=dog&color=black`

the Query portion is

`animal=dog&color=black`

Query is the unparsed query portion of the URL (see page 15190). The query may include multiple fields separated by ampersands. Use the QueryFields (see page 15187) property to work with the query on a field-by-field basis.

See Also

ExtractQueryFields (see page 15167)

Host (see page 15183)

PathInfo (see page 15185)

QueryFields (see page 15187)

ScriptName (see page 15189)

Properties That Identify the Target

77.1.20.5.29 TWebRequest.QueryFields Property

Represents the parsed contents of the Query (see page 15186) in the URL (see page 15190) of the HTTP request message.

Pascal

```
property QueryFields: TStrings;
```

C++

```
__property TStrings QueryFields;
```

Description

Use QueryFields to work with individual query values provided by the Web client request. The Query (see page 15186) property is parsed into separate fields by separating into substrings whenever an ampersand is encountered.

Each query field appears as a separate string in the QueryFields list with the form Name=Value. As a TStrings object, QueryFields allows applications to manipulate the individual fields by name or value.

See Also

ExtractQueryFields (see page 15167)

Query (see page 15186)

TStrings (see page 3029)

Names (see page 3075)

Values (see page 3086)

Properties That Identify the Target

77.1.20.5.30 TWebRequest.Referer Property

Reports the value of the Referer header of the HTTP request message.

Pascal

```
property Referer: string;
```

C++

```
__property AnsiString Referer;
```

Description

Read Referer to determine the URI of the resource where the HTTP request message originated. Referer allows the server application to generate back-links to the source of the request. If Referer is a partial URI, it should be interpreted as a relative path to the URL (see page 15190) property.

The Referer property is not set when the URL (see page 15190) of the HTTP request was obtained from a source that does not have its own URI, such as input from the user keyboard.

See Also

DerivedFrom (see page 15181)

From (see page 15182)

RemoteAddr (see page 15188)

RemoteHost (see page 15188)

UserAgent (see page 15191)

Properties That Describe the Web Client

77.1.20.5.31 TWebRequest.RemoteAddr Property

Indicates the remote IP address of the client associated with the HTTP request message.

Pascal

```
property RemoteAddr: string;
```

C++

```
__property AnsiString RemoteAddr;
```

Description

Read RemoteAddr to obtain the IP address of the source of the Web client request.

See Also

From (see page 15182)

RemoteHost (see page 15188)

UserAgent (see page 15191)

Properties That Describe the Web Client

77.1.20.5.32 TWebRequest.RemoteHost Property

Reports the fully qualified reverse domain lookup of the client associated with the HTTP request message.

Pascal

```
property RemoteHost: string;
```

C++

```
__property AnsiString RemoteHost;
```

Description

Use RemoteHost to compose URI values for back-links to the source of the request when the Referer (see page 15188) property is not specified. RemoteHost does not include any path information.

See Also

Referer (see page 15188)

RemoteAddr (see page 15188)

UserAgent (see page 15191)

Properties That Describe the Web Client

77.1.20.5.33 TWebRequest.ScriptName Property

Indicates the value of the script information (if any) of the URL (see page 15190) specified in the HTTP request message.

Pascal

```
property ScriptName: string;
```

C++

```
__property AnsiString ScriptName;
```

Description

Read ScriptName to determine the full path to the Web server application specified by the client request. For example, given this URL (see page 15190):

`http://www.TSite.com/art/gallery.cgi/mammals?animal=dog&color=black`

the ScriptName portion is:

`/art/gallery.cgi`

See Also

Host (see page 15183)

PathInfo (see page 15185)

Query (see page 15186)

URL (see page 15190)

Properties That Identify the Target

77.1.20.5.34 TWebRequest.ServerPort Property

Indicates the port number of the port on the Web server that receives the HTTP request message.

Pascal

```
property ServerPort: Integer;
```

C++

```
__property int ServerPort;
```

Description

Read ServerPort to determine the number of the port used in the connection to the Web server. The Host (see page 15183) property will include the ServerPort if it is specified as part of the URL (see page 15190) for the client request.

See Also

Host (see page 15183)

URL (see page 15190)

77.1.20.5.35 TWebRequest.Title Property

Reports the value of the Title header of the HTTP request message.

Pascal

```
property Title: string;
```

C++

```
__property AnsiString Title;
```

Description

Read Title to determine the Title of the information requested in the Web client request.

See Also

Cookie (see page 15180)

Title (see page 15217)

Properties That Describe the Content

77.1.20.5.36 TWebRequest.URL Property

Reports the Uniform Resource Identifier (URI) specified by the HTTP request message to identify the target of the request.

Pascal

```
property URL: string;
```

C++

```
__property AnsiString URL;
```

Description

Use the URL property to identify the resource the Web client identifies as the target of the request message. URL may be an HTML page to return, a resource to be created or deleted, or an object that provides some other service for the Web client.

The TWebRequest (see page 15161) object parses the URL into its constituent parts, yielding the Host (see page 15183), ScriptName (see page 15189), PathInfo (see page 15185), and Query (see page 15186) properties.

Note: Usually the URL property is a Uniform Resource Locator (URL) that can be parsed into host, script, path, and query portions. However, it is possible for HTTP request messages to specify the target in more general terms, as a Uniform Resource Identifier (URI).

See Also

Host (see page 15183)

PathInfo (see page 15185)

Query (see page 15186)

ScriptName (see page 15189)

Properties That Identify the Target

77.1.20.5.37 TWebRequest.UserAgent Property

Reports the value of the User-Agent header of the HTTP request message.

Pascal

```
property UserAgent: string;
```

C++

```
__property AnsiString UserAgent;
```

Description

Read UserAgent to obtain information about the user agent originating the client request. Use the UserAgent information for statistical purposes, for tracing protocol violations, and for tailoring responses to known limitations of specific applications that generate client requests. The UserAgent string contains one or more descriptors, in order of significance. Each descriptor gives the name of the product or application that generated the request, or a comment about the user agent.

See Also

From (see page 15182)

Referer (see page 15188)

Properties That Describe the Web Client

77.1.21 TWebResponse Class

TWebResponse is the base class for all objects that represent HTTP messages sent in response to an HTTP request message.

Class Hierarchy

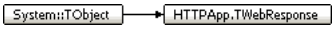

Pascal

```
TWebResponse = class(TObject);
```

C++

```
class TWebResponse : public TObject;
```

File

HTTPApp

Description

Use a descendant of TWebResponse to specify the response to an HTTP request message.

A web application creates a TWebResponse object based on the TWebRequest (see page 15161) object for an incoming HTTP request message. The Web dispatcher for the application then passes the TWebResponse object to the TWebActionItem (see page 15130) associated with the TWebRequest (see page 15161) object, so that the response can be formulated.

TWebResponse has descendants that represent HTTP response messages in different types of web server applications. These descendants are

TApacheResponse objects in Apache server applications.

TCGIResponse objects in CGI server applications.

TISAPIResponse objects in server DLLs (Windows only).

TWinCGIResponse objects in Windows-based CGI servers (Windows only).

See Also

TApacheResponse

TCGIResponse (see page 2572)

TISAPIResponse (see page 15670)

TWebActionItem (see page 15130)

TWebApplication (see page 24618)

TWebDispatcher (see page 15147)

TWebRequest (see page 15161)

TWinCGIResponse (see page 2590)

Filling in the Response Header

Responding to Request Messages with Action Items

The Structure of a Web Broker Application

Dispatching Requests and Responses

77.1.21.1 TWebResponse Members

The following tables list the members exposed by TWebResponse.

Protected Fields

	Name	Description
	FCustomHeaders (see page 15195)	This is FCustomHeaders, a member of class TWebResponse.
	FHTTPRequest (see page 15196)	This is FHTTPRequest, a member of class TWebResponse.

Protected Methods

	Name	Description
	AddCustomHeaders (see page 15197)	Appends the custom headers specified by the CustomHeaders (see page 15211) property to the Headers parameter.
	FormatAuthenticate (see page 15197)	Lists of one or more comma-separated challenges provided when the request fails with StatusCode (see page 15215) 401 (Unauthorized).
A	GetContent (see page 15198)	Provides the abstract (pure virtual) prototype of a method to access the Content (see page 15207) property.
A	GetDateVariable (see page 15198)	Provides the abstract (pure virtual) prototype of a method to extract a date value from an indexed table of HTTP header variables.
A	GetIntegerVariable (see page 15199)	Provides the abstract (in C++ terminology, pure vitrual) prototype of a method to extract an integer value from an indexed table of HTTP header variables.
A	GetLogMessage (see page 15199)	Provides the abstract (pure virtual) prototype of a method to access the LogMessage (see page 15214) property.
A	GetStatusCode (see page 15199)	Provides the abstract (pure virtual) prototype of a method to access the StatusCode (see page 15215) property.

	GetStringVariable (see page 15200)	Provides the prototype of a method to extract a string value from an indexed table of HTTP header variables.
	SetContent (see page 15202)	Provides the abstract (pure virtual) prototype of a method to set the Content (see page 15207) property.
	SetContentStream (see page 15203)	Sets the value of the ContentStream (see page 15209) property to the Value parameter.
	SetDateVariable (see page 15204)	Provides the abstract (pure virtual) prototype of a method to set a date-time valued HTTP header property.
	SetIntegerVariable (see page 15204)	Provides the abstract (pure virtual) prototype of a method to set an integer valued HTTP header property.
	SetLogMessage (see page 15205)	Provides the abstract (pure virtual) prototype of a method to set the value of the LogMessage (see page 15214) property.
	SetStatusCode (see page 15205)	Provides the abstract (pure virtual) prototype of a method to set the value of the StatusCode (see page 15215) property.
	SetStringVariable (see page 15205)	Provides the abstract (pure virtual) prototype of a method to set a string valued HTTP header property.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 15194)	Creates an instance of Create to represent the response to the HTTPRequest (see page 15213) parameter.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 15195)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetCustomHeader (see page 15198)	Returns the value of the custom header with the specified name.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SendRedirect (see page 15201)	Provides the abstract (pure virtual) prototype of a method that sends the response with a StatusCode (see page 15215) of 301 (Moved Permanently) and a Location (see page 15214) of URI.
	SendResponse (see page 15201)	Provides the abstract (pure virtual) prototype of a method that sends the HTTP response message to the web client.
	SendStream (see page 15201)	Provides the abstract (pure virtual) prototype of a method that sends the HTTP response message with the contents set to the contents of AStream.

V	Sent (see page 15202)	Indicates whether the HTTP response message has already been sent.
	SetCookieField (see page 15203)	Adds a cookie header to the response message.
	SetCustomHeader (see page 15203)	Replaces the value of a specified custom header.

Public Properties

	Name	Description
	Allow (see page 15207)	Lists the HTTP request methods that the application can service.
	Content (see page 15207)	Contains the information sent to the web client in response to an HTTP request message.
	ContentEncoding (see page 15208)	Lists the mechanisms, if any, that are used to encode the content of the HTTP response message.
	ContentLength (see page 15208)	Specifies the length, in bytes, of the Content (see page 15207) property.
	ContentStream (see page 15209)	Specifies the stream object for writing the contents of the HTTP response message to the web client.
	ContentType (see page 15209)	Indicates the media type of the Content (see page 15207) of the HTTP response message.
	ContentVersion (see page 15210)	Represents the version associated with the Content (see page 15207) of the HTTP response message.
R	Cookies (see page 15210)	Lists all cookie headers that should be sent with the response.
	CustomHeaders (see page 15211)	Lists message header strings that are added to the response message before it is sent.
	Date (see page 15211)	Indicates the time and date when the HTTP response message is sent to the web client.
	DerivedFrom (see page 15212)	Specifies the URI of the resource where the Content (see page 15207) of the HTTP response originated.
	Expires (see page 15212)	Specifies the date and time after which the Content (see page 15207) of the HTTP response message should be considered invalid.
R	HTTPRequest (see page 15213)	Represents the HTTP request message for which this is a response.
	LastModified (see page 15213)	Indicates the date and time that the value of the Content (see page 15207) property was last updated.
	Location (see page 15214)	Specifies the URI of a resource the web client should point to for the completion of its request.
	LogMessage (see page 15214)	Describes the HTTP response message for event logging purposes.
	Realm (see page 15214)	Specifies the set of protected-access URIs that includes the URL specified by the HTTP request message.
	ReasonString (see page 15215)	Indicates the status of the response to the web client request as a human-readable string.
	Server (see page 15215)	Provides information about the software used by the server application to handle the request.
	StatusCode (see page 15215)	Indicates the status of the response to the HTTP request message as a 3-digit integer.
	Title (see page 15217)	Specifies a title for the Content (see page 15207) of the HTTP response message.
	Version (see page 15217)	Specifies the HTTP protocol version number that the web client is using.
	WWWAuthenticate (see page 15218)	Lists of one or more comma-separated challenges provided when the request fails with StatusCode (see page 15215) 401 (Unauthorized).

77.1.21.2 TWebResponse.Create Constructor

Creates an instance of Create to represent the response to the HTTPRequest (see page 15213) parameter.

Pascal

```
constructor Create(HTTPRequest: TWebRequest);
```

C++

```
__fastcall TWebResponse(TWebRequest HTTPRequest);
```

Description

Applications shouldn't need to call Create to instantiate a response object. The TWebApplication object creates a Create for

each TWebRequest (see page 15161) it creates.

See Also

TWebApplication (see page 24618)

TWebRequest (see page 15161)

77.1.21.3 TWebResponse.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebResponse();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

77.1.21.4 TWebResponse Fields

The fields of the TWebResponse class are listed here.

Protected Fields

	Name	Description
	FCustomHeaders (see page 15195)	This is FCustomHeaders, a member of class TWebResponse.
	FHTTPRequest (see page 15196)	This is FHTTPRequest, a member of class TWebResponse.

77.1.21.4.1 TWebResponse.FCustomHeaders Field

Pascal

```
FCustomHeaders: TStrings;
```

C++

```
TStrings FCustomHeaders;
```

Description

This is FCustomHeaders, a member of class TWebResponse.

77.1.21.4.2 TWebResponse.FHTTPRequest Field

Pascal

```
FHTTPRequest: TWebRequest;
```

C++

```
TWebRequest FHTTPRequest;
```

Description

This is FHTTPRequest, a member of class TWebResponse.

77.1.21.5 TWebResponse Methods

The methods of the TWebResponse class are listed here.

Protected Methods

	Name	Description
	AddCustomHeaders (see page 15197)	Appends the custom headers specified by the CustomHeaders (see page 15211) property to the Headers parameter.
	FormatAuthenticate (see page 15197)	Lists of one or more comma-separated challenges provided when the request fails with StatusCode (see page 15215) 401 (Unauthorized).
A	GetContent (see page 15198)	Provides the abstract (pure virtual) prototype of a method to access the Content (see page 15207) property.
A	GetDateVariable (see page 15198)	Provides the abstract (pure virtual) prototype of a method to extract a date value from an indexed table of HTTP header variables.
A	GetIntegerVariable (see page 15199)	Provides the abstract (in C++ terminology, pure vitrual) prototype of a method to extract an integer value from an indexed table of HTTP header variables.
A	GetLogMessage (see page 15199)	Provides the abstract (pure virtual) prototype of a method to access the LogMessage (see page 15214) property.
A	GetStatusCode (see page 15199)	Provides the abstract (pure virtual) prototype of a method to access the StatusCode (see page 15215) property.
A	GetStringVariable (see page 15200)	Provides the prototype of a method to extract a string value from an indexed table of HTTP header variables.
A	SetContent (see page 15202)	Provides the abstract (pure virtual) prototype of a method to set the Content (see page 15207) property.
V	SetContentStream (see page 15203)	Sets the value of the ContentStream (see page 15209) property to the Value parameter.
A	SetDateVariable (see page 15204)	Provides the abstract (pure virtual) prototype of a method to set a date-time valued HTTP header property.
A	SetIntegerVariable (see page 15204)	Provides the abstract (pure virtual) prototype of a method to set an integer valued HTTP header property.
A	SetLogMessage (see page 15205)	Provides the abstract (pure virtual) prototype of a method to set the value of the LogMessage (see page 15214) property.
A	SetStatusCode (see page 15205)	Provides the abstract (pure virtual) prototype of a method to set the value of the StatusCode (see page 15215) property.
A	SetStringVariable (see page 15205)	Provides the abstract (pure virtual) prototype of a method to set a string valued HTTP header property.

Public Methods

	Name	Description
	GetCustomHeader (see page 15198)	Returns the value of the custom header with the specified name.
A	SendRedirect (see page 15201)	Provides the abstract (pure virtual) prototype of a method that sends the response with a StatusCode (see page 15215) of 301 (Moved Permanently) and a Location (see page 15214) of URI.
A	SendResponse (see page 15201)	Provides the abstract (pure virtual) prototype of a method that sends the HTTP response message to the web client.
A	SendStream (see page 15201)	Provides the abstract (pure virtual) prototype of a method that sends the HTTP response message with the contents set to the contents of AStream.
V	Sent (see page 15202)	Indicates whether the HTTP response message has already been sent.
	SetCookieField (see page 15203)	Adds a cookie header to the response message.
	SetCustomHeader (see page 15203)	Replaces the value of a specified custom header.

77.1.21.5.1 TWebResponse.AddCustomHeaders Method

Appends the custom headers specified by the CustomHeaders (see page 15211) property to the Headers parameter.

Pascal

```
procedure AddCustomHeaders(var Headers: string);
```

C++

```
__fastcall AddCustomHeaders(AnsiString Headers);
```

Description

Call AddCustomHeaders to add the message headers described by the CustomHeaders (see page 15211) property to a string of message headers. Each header is added on its own line (separated by carriage return and line feed), in the format HeaderName: Value.

See Also

CustomHeaders (see page 15211)

GetCustomHeader (see page 15198)

77.1.21.5.2 TWebResponse.FormatAuthenticate Method

Lists of one or more comma-separated challenges provided when the request fails with StatusCode (see page 15215) 401 (Unauthorized).

Pascal

```
function FormatAuthenticate: string;
```

C++

```
__fastcall AnsiString FormatAuthenticate();
```

Description

FormatAuthenticate returns the value of WWWAuthenticate (see page 15218) if Realm (see page 15214) is an empty string. If it is not empty, it returns a name/value pair describing the realm of the WWWAuthenticate (see page 15218) property.

See Also

WWWAuthenticate (see page 15218)

Realm (see page 15214)

77.1.21.5.3 TWebResponse.GetContent Method

Provides the abstract (pure virtual) prototype of a method to access the Content (see page 15207) property.

Pascal

```
function GetContent: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetContent() = 0;
```

Description

Descendants of TWebResponse (see page 15191) implement GetContent to provide an access method for their internal representation of the Content (see page 15207) property. This method is abstract (pure virtual) because response objects for different types of HTTP applications store the value of the Content (see page 15207) property differently.

See Also

Content (see page 15207)

SetContent (see page 15202)

77.1.21.5.4 TWebResponse.GetCustomHeader Method

Returns the value of the custom header with the specified name.

Pascal

```
function GetCustomHeader(const Name: string): String;
```

C++

```
__fastcall AnsiString GetCustomHeader(const AnsiString Name);
```

Description

Call GetCustomHeader to obtain the value of one of the custom headers specified in the CustomHeaders (see page 15211) property. The Name parameter specifies the name of the HTTP message header.

See Also

CustomHeaders (see page 15211)

SetCustomHeader (see page 15203)

77.1.21.5.5 TWebResponse.GetDateVariable Method

Provides the abstract (pure virtual) prototype of a method to extract a date value from an indexed table of HTTP header variables.

Pascal

```
function GetDateVariable(Index: Integer): TDateTime; virtual; abstract;
```

C++

```
virtual __fastcall TDateTime GetDateVariable(int Index) = 0;
```

Description

Descendants of TWebResponse (see page 15191) implement GetDateVariable to provide a method by which date-time values are read from the internal table used to store HTTP header variables. The response object uses GetDateVariable to access the properties that represent dates.

See Also

Date (see page 15211)

Expires (see page 15212)

LastModified (see page 15213)

SetDateVariable (see page 15204)

77.1.21.5.6 TWebResponse.GetIntegerVariable Method

Provides the abstract (in C++ terminology, pure vitrual) prototype of a method to extract an integer value from an indexed table of HTTP header variables.

Pascal

```
function GetIntegerVariable(Index: Integer): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetIntegerVariable(int Index) = 0;
```

Description

Descendants of TWebResponse (see page 15191) implement GetIntegerVariable to provide a method by which integer values are read from the internal table used to store HTTP header variables. The response object uses GetIntegerVariable to access the properties that represent integers.

See Also

ContentLength (see page 15208)

SetIntegerVariable (see page 15204)

77.1.21.5.7 TWebResponse.GetLogMessage Method

Provides the abstract (pure virtual) prototype of a method to access the LogMessage (see page 15214) property.

Pascal

```
function GetLogMessage: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetLogMessage() = 0;
```

Description

Descendants of TWebResponse (see page 15191) implement GetLogMessage to provide an access method for internal representation of the LogMessage (see page 15214) property. This method is abstract (pure virtual) because response objects for different types of HTTP applications store the value of the LogMessage (see page 15214) property differently.

See Also

LogMessage (see page 15214)

SetLogMessage (see page 15205)

77.1.21.5.8 TWebResponse.GetStatusCode Method

Provides the abstract (pure virtual) prototype of a method to access the StatusCode (see page 15215) property.

Pascal

```
function GetStatusCode: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetStatusCode() = 0;
```

Description

Descendants of TWebResponse (see page 15191) implement GetStatusCode to provide an access method for internal representation of the StatusCode (see page 15215) property. This method is abstract (pure virtual) because response objects for different types of HTTP applications store the value of the StatusCode (see page 15215) property differently.

See Also

StatusCode (see page 15215)

SetStatusCode (see page 15205)

77.1.21.5.9 TWebResponse.GetStringVariable Method

Provides the prototype of a method to extract a string value from an indexed table of HTTP header variables.

Pascal

```
function GetStringVariable(Index: Integer): string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetStringVariable(int Index) = 0;
```

Description

GetStringVariable provides the abstract (pure virtual) prototype of a method to extract a string value from an indexed table of HTTP header variables. Descendants of TWebResponse (see page 15191) implement GetStringVariable to provide a method by which the string values are read from the internal table used to store HTTP header variables. The response object uses GetStringVariable to access the properties that represent strings.

See Also

Allow (see page 15207)

ContentEncoding (see page 15208)

ContentType (see page 15209)

ContentVersion (see page 15210)

DerivedFrom (see page 15212)

Location (see page 15214)

Realm (see page 15214)

ReasonString (see page 15215)

Server (see page 15215)

SetStringVariable (see page 15205)

Title (see page 15217)

Version (see page 15217)

WWWAuthenticate (see page 15218)

77.1.21.5.10 TWebResponse.SendRedirect Method

Provides the abstract (pure virtual) prototype of a method that sends the response with a StatusCode (see page 15215) of 301 (Moved Permanently) and a Location (see page 15214) of URI.

Pascal

```
procedure SendRedirect(const URI: string); virtual; abstract;
```

C++

```
virtual __fastcall SendRedirect(const AnsiString URI) = 0;
```

Description

Call SendRedirect to redirect the web client request to another URI that is the proper target of the request. SendRedirect does the same thing as setting the StatusCode (see page 15215) to 301, the Location (see page 15214) to URI, and calling SendResponse (see page 15201).

See Also

Location (see page 15214)

SendResponse (see page 15201)

StatusCode (see page 15215)

Sending the Response

77.1.21.5.11 TWebResponse.SendResponse Method

Provides the abstract (pure virtual) prototype of a method that sends the HTTP response message to the web client.

Pascal

```
procedure SendResponse; virtual; abstract;
```

C++

```
virtual __fastcall SendResponse() = 0;
```

Description

Call SendResponse after setting the properties of the HTTP response object to specify the response that should be made to the web client. If the request should be redirected to another URI, it is more efficient to call SendRedirect (see page 15201) instead.

See Also

SendRedirect (see page 15201)

Sending the Response

77.1.21.5.12 TWebResponse.SendStream Method

Provides the abstract (pure virtual) prototype of a method that sends the HTTP response message with the contents set to the contents of AStream.

Pascal

```
procedure SendStream(AStream: TStream); virtual; abstract;
```

C++

```
virtual __fastcall SendStream(TStream AStream) = 0;
```

Description

Only call SendStream after first calling SendResponse (see page 15201). Unlike SendStream, SendResponse (see page 15201) fills the header of the HTTP response message, based on the properties of the TWebResponse (see page 15191) object. If SendResponse (see page 15201) has not been called first to set the header of the HTTP response message, set the ContentStream (see page 15209) property to AStream, and call SendResponse (see page 15201).

See Also

ContentStream (see page 15209)

SendResponse (see page 15201)

Sent (see page 15202)

77.1.21.5.13 TWebResponse.Sent Method

Indicates whether the HTTP response message has already been sent.

Pascal

```
function Sent: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean Sent();
```

Description

Because SendRedirect (see page 15201) and SendStream (see page 15201) are abstract (pure virtual) methods for TWebResponse (see page 15191), Sent always returns false. Descendants of TWebResponse (see page 15191) override Sent to check whether the message was sent and return true or false as appropriate.

TWebApplication objects read Sent and send unsent responses to the web client.

See Also

SendRedirect (see page 15201)

SendStream (see page 15201)

TWebApplication (see page 24618)

77.1.21.5.14 TWebResponse.SetContent Method

Provides the abstract (pure virtual) prototype of a method to set the Content (see page 15207) property.

Pascal

```
procedure SetContent(const Value: string); virtual; abstract;
```

C++

```
virtual __fastcall SetContent(const AnsiString Value) = 0;
```

Description

Descendants of TWebResponse (see page 15191) implement SetContent to provide a method for setting the Content (see page 15207) property. This method is abstract (pure virtual) because response objects for different types of HTTP applications can store the value of the Content (see page 15207) property differently.

See Also

Content (see page 15207)

GetContent (see page 15198)

77.1.21.5.15 TWebResponse.SetContentStream Method

Sets the value of the ContentStream (see page 15209) property to the Value parameter.

Pascal

```
procedure SetContentStream(Value: TStream); virtual;
```

C++

```
virtual __fastcall SetContentStream(TStream Value);
```

Description

SetContentStream frees the previous value of ContentStream (see page 15209) and replaces the property with the Value parameter.

See Also

ContentStream (see page 15209)

77.1.21.5.16 TWebResponse.SetCookieField Method

Adds a cookie header to the response message.

Pascal

```
procedure SetCookieField(Values: TStrings; const ADomain: string; const APath: string;
AExpires: TDateTime; ASecure: Boolean);
```

C++

```
__fastcall SetCookieField(TStrings Values, const AnsiString ADomain, const AnsiString APath,
TDateTime AExpires, Boolean ASecure);
```

Description

Call SetCookieField to add a cookie header to the Cookies (see page 15210) property. Each cookie header contains a set of name/value pairs that can be passed on by the client application. Use the Values parameter to specify the name/value pairs as strings of the form Name=Value. Use the ADomain and APath parameters to indicate which URLs the cookie should be sent to. Use the AExpires parameter to indicate when the cookie is no longer valid. Use the ASecure property to indicate whether the cookie should only be passed on by the client if a secure connection is used.

See the documentation on the Netscape Web site for more information about cookies.

See Also

Cookies (see page 15210)

77.1.21.5.17 TWebResponse.SetCustomHeader Method

Replaces the value of a specified custom header.

Pascal

```
procedure SetCustomHeader(const Name: string; const Value: string);
```

C++

```
__fastcall SetCustomHeader(const AnsiString Name, const AnsiString Value);
```

Description

Call SetCustomHeader to add a new custom header to the list maintained by the CustomHeaders (see page 15211) property, or to change the value of one of the headers already in the list. SetCustomHeader locates the header in the list with the name specified by the Name parameter. If a header with that name is in the list, SetCustomHeader replaces the value of that header with the value specified as the Value parameter. If the name was not found, SetCustomHeader adds a new header to the list with the given name and value.

See Also

CustomHeaders (see page 15211)

GetCustomHeader (see page 15198)

77.1.21.5.18 TWebResponse.SetDateVariable Method

Provides the abstract (pure virtual) prototype of a method to set a date-time valued HTTP header property.

Pascal

```
procedure SetDateVariable(Index: Integer; const Value: TDateTime); virtual; abstract;
```

C++

```
virtual __fastcall SetDateVariable(int Index, const TDateTime Value) = 0;
```

Description

Descendants of TWebResponse (see page 15191) implement SetDateVariable to provide a method by which date-time values are written to the internal table used to store HTTP header variables. The response object uses SetDateVariable to set the properties that represent dates.

See Also

Date (see page 15211)

Expires (see page 15212)

GetDateVariable (see page 15198)

LastModified (see page 15213)

77.1.21.5.19 TWebResponse.SetIntegerVariable Method

Provides the abstract (pure virtual) prototype of a method to set an integer valued HTTP header property.

Pascal

```
procedure SetIntegerVariable(Index: Integer; Value: Integer); virtual; abstract;
```

C++

```
virtual __fastcall SetIntegerVariable(int Index, int Value) = 0;
```

Description

Descendants of TWebResponse (see page 15191) implement SetIntegerVariable to provide a method by which integer values are written to the internal table used to store HTTP header variables. The response object uses SetIntegerVariable to set the properties that represent integers.

See Also

ContentLength (see page 15208)

GetIntegerVariable (see page 15199)

77.1.21.5.20 TWebResponse.SetLogMessage Method

Provides the abstract (pure virtual) prototype of a method to set the value of the LogMessage (see page 15214) property.

Pascal

```
procedure SetLogMessage(const Value: string); virtual; abstract;
```

C++

```
virtual __fastcall SetLogMessage(const AnsiString Value) = 0;
```

Description

Descendants of TWebResponse (see page 15191) implement SetLogMessage to provide a method to set the internal representation of the LogMessage (see page 15214) property. This method is abstract (pure virtual) because response objects for different types of HTTP applications store the value of the LogMessage (see page 15214) property differently.

See Also

GetLogMessage (see page 15199)

LogMessage (see page 15214)

77.1.21.5.21 TWebResponse.SetStatusCode Method

Provides the abstract (pure virtual) prototype of a method to set the value of the StatusCode (see page 15215) property.

Pascal

```
procedure SetStatusCode(Value: Integer); virtual; abstract;
```

C++

```
virtual __fastcall SetStatusCode(int Value) = 0;
```

Description

Descendants of TWebResponse (see page 15191) implement SetStatusCode to provide a method to set the internal representation of the StatusCode (see page 15215) property. This method is abstract (pure virtual) because response objects for different types of HTTP applications store the value of the StatusCode (see page 15215) property differently.

See Also

GetStatusCode (see page 15199)

StatusCode (see page 15215)

77.1.21.5.22 TWebResponse.SetStringVariable Method

Provides the abstract (pure virtual) prototype of a method to set a string valued HTTP header property.

Pascal

```
procedure SetStringVariable(Index: Integer; const Value: string); virtual; abstract;
```

C++

```
virtual __fastcall SetStringVariable(int Index, const AnsiString Value) = 0;
```

Description

Descendants of TWebResponse (see page 15191) implement SetStringVariable to provide a method by which string values are written to the internal table used to store HTTP header variables. The response object uses SetStringVariable to set the properties that represent strings.

See Also

Allow (see page 15207)

ContentEncoding (see page 15208)

ContentType (see page 15209)

ContentVersion (see page 15210)

DerivedFrom (see page 15212)

GetStringVariable (see page 15200)

Location (see page 15214)

Realm (see page 15214)

ReasonString (see page 15215)

Server (see page 15215)

Title (see page 15217)

Version (see page 15217)

WWWAuthenticate (see page 15218)

77.1.21.6 TWebResponse Properties

The properties of the TWebResponse class are listed here.

Public Properties

	Name	Description
	Allow (see page 15207)	Lists the HTTP request methods that the application can service.
	Content (see page 15207)	Contains the information sent to the web client in response to an HTTP request message.
	ContentEncoding (see page 15208)	Lists the mechanisms, if any, that are used to encode the content of the HTTP response message.
	ContentLength (see page 15208)	Specifies the length, in bytes, of the Content (see page 15207) property.
	ContentStream (see page 15209)	Specifies the stream object for writing the contents of the HTTP response message to the web client.
	ContentType (see page 15209)	Indicates the media type of the Content (see page 15207) of the HTTP response message.
	ContentVersion (see page 15210)	Represents the version associated with the Content (see page 15207) of the HTTP response message.
R	Cookies (see page 15210)	Lists all cookie headers that should be sent with the response.
	CustomHeaders (see page 15211)	Lists message header strings that are added to the response message before it is sent.
	Date (see page 15211)	Indicates the time and date when the HTTP response message is sent to the web client.
	DerivedFrom (see page 15212)	Specifies the URI of the resource where the Content (see page 15207) of the HTTP response originated.
	Expires (see page 15212)	Specifies the date and time after which the Content (see page 15207) of the HTTP response message should be considered invalid.
R	HTTPRequest (see page 15213)	Represents the HTTP request message for which this is a response.
	LastModified (see page 15213)	Indicates the date and time that the value of the Content (see page 15207) property was last updated.
	Location (see page 15214)	Specifies the URI of a resource the web client should point to for the completion of its request.
	LogMessage (see page 15214)	Describes the HTTP response message for event logging purposes.

	Realm (see page 15214)	Specifies the set of protected-access URIs that includes the URL specified by the HTTP request message.
	ReasonString (see page 15215)	Indicates the status of the response to the web client request as a human-readable string.
	Server (see page 15215)	Provides information about the software used by the server application to handle the request.
	StatusCode (see page 15215)	Indicates the status of the response to the HTTP request message as a 3-digit integer.
	Title (see page 15217)	Specifies a title for the Content (see page 15207) of the HTTP response message.
	Version (see page 15217)	Specifies the HTTP protocol version number that the web client is using.
	WWWAuthenticate (see page 15218)	Lists of one or more comma-separated challenges provided when the request fails with StatusCode (see page 15215) 401 (Unauthorized).

77.1.21.6.1 TWebResponse.Allow Property

Lists the HTTP request methods that the application can service.

Pascal

```
property Allow: string;
```

C++

```
__property AnsiString Allow;
```

Description

Use Allow to indicate what type of request methods the server application can respond to for the URI specified by HTTPRequest (see page 15213). Allow is a string of method names, separated by commas. The method names can include any pre-defined methods from the HTTP standard, or any additional methods implemented by the server application. At a minimum, the TWebResponse (see page 15191) object must allow GET and HEAD requests.

See Also

HTTPRequest (see page 15213)

TWebRequest (see page 15161)

URL (see page 15190)

Describing the Server Application

77.1.21.6.2 TWebResponse.Content Property

Contains the information sent to the web client in response to an HTTP request message.

Pascal

```
property Content: string;
```

C++

```
__property AnsiString Content;
```

Description

Use Content to send requested information to the web client. Not every HTTP request message requires content to be returned. For example, responses to requests with a Method of OPTIONS, HEAD, POST, PUT, DELETE, or TRACE should not include any Content. In all these cases, the response information is contained entirely in the response header, which is built from the other properties of the TWebResponse (see page 15191) object.

The response content can be a string of HTML commands, the name of an HTML file, graphic content such as a bit-stream, or any other MIME content type.

Note: If the ContentStream (see page 15209) property is set, it supersedes the value of the Content property. If both Content and ContentStream (see page 15209) are set, the web client is sent the value of the ContentStream (see page 15209), not the value of the Content property.

See Also

ContentEncoding (see page 15208)

ContentLength (see page 15208)

ContentStream (see page 15209)

ContentType (see page 15209)

ContentVersion (see page 15210)

DerivedFrom (see page 15212)

Expires (see page 15212)

TWebRequest (see page 15161)

Responding to Request Messages with Action Items

Setting the Response Content

77.1.21.6.3 TWebResponse.ContentEncoding Property

Lists the mechanisms, if any, that are used to encode the content of the HTTP response message.

Pascal

```
property ContentEncoding: string;
```

C++

```
__property AnsiString ContentEncoding;
```

Description

Set ContentEncoding when the value of Content (see page 15207) has been encoded before it is sent to the web client. The encoding mechanisms are listed in the order in which they are applied, separated by commas. Typically, ContentEncoding indicates a compression method or an encoding method used to transfer extended-ASCII characters over the net.

See Also

Content (see page 15207)

ContentType (see page 15209)

Describing the Content

77.1.21.6.4 TWebResponse.ContentLength Property

Specifies the length, in bytes, of the Content (see page 15207) property.

Pascal

```
property ContentLength: Integer;
```

C++

```
__property int ContentLength;
```

Description

Set ContentLength to the length of the Content (see page 15207) property, if it is known. If the length of the Content (see page 15207) property is unknown, or if the response message does not include any content, set ContentLength to 0. Web clients use the ContentLength when reading the contents of the HTTP response message.

If the contents of the response message are taken from a stream, the value of ContentLength is set from the stream.

See Also

Content (see page 15207)

ContentStream (see page 15209)

Describing the Content

77.1.21.6.5 TWebResponse.ContentStream Property

Specifies the stream object for writing the contents of the HTTP response message to the web client.

Pascal

```
property ContentStream: TStream;
```

C++

```
__property TStream ContentStream;
```

Description

Use ContentStream to set the contents of the HTTP response message to a value that can be written from a stream. ContentStream allows the response object to write its contents directly from a stream.

The stream specified by ContentStream can access a string of HTML commands, the contents of a BLOB database field, or any MIME content type.

Note: ContentStream supersedes the Content (see page 15207) property. If both Content (see page 15207) and ContentStream are set, the web client is sent the value of the ContentStream, not the value of the Content (see page 15207) property.

See Also

Content (see page 15207)

ContentType (see page 15209)

ContentVersion (see page 15210)

DerivedFrom (see page 15212)

Expires (see page 15212)

Setting the Response Content

77.1.21.6.6 TWebResponse.ContentType Property

Indicates the media type of the Content (see page 15207) of the HTTP response message.

Pascal

```
property ContentType: string;
```

C++

```
__property AnsiString ContentType;
```

Description

Set ContentType to indicate the media type of the Content (see page 15207) of the HTTP response message. The value of ContentType must specify a valid MIME content type. The ContentType string has the form Type/SubType.

For example, if the HTTP response message sends a string of HTML commands to the web client in response to a GET request, the ContentType is text/html.

Note: If the HTTP request message includes an Accept header, ContentType should be one of the content types listed in the Accept property of TWebRequest (see page 15161).

See Also

Content (see page 15207)

Accept (see page 15174)

Describing the Content

77.1.21.6.7 TWebResponse.ContentVersion Property

Represents the version associated with the Content (see page 15207) of the HTTP response message.

Pascal

```
property ContentVersion: string;
```

C++

```
__property AnsiString ContentVersion;
```

Description

Use ContentVersion to specify a version associated with the Content (see page 15207) of the HTTP response. The meaning of ContentVersion depends on the type of information that is sent in the HTTP response. For example, if the Content (see page 15207) of the HTTP response has a ContentType (see page 15209) of text/html, ContentVersion would be the version of HTML used.

Server (see page 15215) applications that introduce meanings for the ContentVersion property should document the interpretation they assign to ContentVersion.

See Also

Content (see page 15207)

ContentType (see page 15209)

Describing the Content

77.1.21.6.8 TWebResponse.Cookies Property

Lists all cookie headers that should be sent with the response.

Pascal

```
property Cookies: TCookieCollection;
```

C++

```
__property TCookieCollection Cookies;
```

Description

Use Cookies to obtain direct access to the individual TCookie (see page 15092) objects that represent the fields of the cookie headers in the response message. Each TCookie (see page 15092) object represents in individual cookie header that is

included with the response message.

Use the Add method of the cookie collection to add a single cookie to the collection, or use the SetCookieField (see page 15203) method to add multiple related cookie headers to this collection. SetCookieField (see page 15203) creates and adds new cookies that share a common set of values for the domain, path, expires, and secure fields. Use the HeaderValue property of the individual entries in Cookies to determine the actual response header string that is sent for each cookie header.

Note: For more information about the use of cookies, see the documentation on Netscape's Web site.

See Also

SetCookieField (see page 15203)

Cookie (see page 15180)

TCookie (see page 15092)

Describing the Server Application

77.1.21.6.9 TWebResponse.CustomHeaders Property

Lists message header strings that are added to the response message before it is sent.

Pascal

```
property CustomHeaders: TStrings;
```

C++

```
__property TStrings CustomHeaders;
```

Description

Use CustomHeaders to add header information to the response that can not be specified using the other properties of TWebResponse (see page 15191). Each string in the CustomHeaders property has the form HeaderName=Value, where HeaderName specifies the name of the header, and Value is the value for that header, formatted as a string.

Note: Setting the CustomHeaders property assigns a set of strings to the custom headers list. It does not replace the TStrings object that is the value of CustomHeaders.

Warning: CustomHeaders includes at most one value for each HeaderName. If the HTTP message includes multiple headers with the same name, only the first header's value will appear in CustomHeaders.

See Also

GetCustomHeader (see page 15198)

SetCustomHeader (see page 15203)

TStrings (see page 3029)

Assign (see page 3043)

77.1.21.6.10 TWebResponse.Date Property

Indicates the time and date when the HTTP response message is sent to the web client.

Pascal

```
property Date: TDateTime;
```

C++

```
__property TDateTime Date;
```

Description

Set Date to the current date and time immediately before sending the HTTP response message to the web client. The Web client uses Date to determine the age of the information returned.

Date is represented using the TDateTime type. The integral part of a TDateTime value is the number of days that have passed since December 30, 1899. The fractional part of a TDateTime value is the fraction of a 24-hour day that has elapsed.

See Also

Expires (see page 15212)

LastModified (see page 15213)

77.1.21.6.11 TWebResponse.DerivedFrom Property

Specifies the URI of the resource where the Content (see page 15207) of the HTTP response originated.

Pascal

```
property DerivedFrom: string;
```

C++

```
__property AnsiString DerivedFrom;
```

Description

Set DerivedFrom to the URI where the value of Content (see page 15207) originates. It is especially important to set DerivedFrom when the Content (see page 15207) of the HTTP response did not originate from the resource indicated by the URL property of the request object.

See Also

Content (see page 15207)

URL (see page 15190)

Describing the Content

77.1.21.6.12 TWebResponse.Expires Property

Specifies the date and time after which the Content (see page 15207) of the HTTP response message should be considered invalid.

Pascal

```
property Expires: TDateTime;
```

C++

```
__property TDateTime Expires;
```

Description

Set Expires when the Content (see page 15207) of the HTTP response message contains information that may become invalid with the passage of time. Setting an Expires value does not imply that the information or its source will necessarily change or cease to exist when the Expires date and time is reached. Web clients are simply warned to check the freshness of the information after the Expires date and time.

The value of Expires is specified in Greenwich Mean Time.

Expires is represented using the TDateTime type. The integral part of a TDateTime value is the number of days that have passed since December 30, 1899. The fractional part of a TDateTime value is the fraction of a 24-hour day that has elapsed.

See Also

Date (see page 15211)

LastModified (see page 15213)

Describing the Content

77.1.21.6.13 TWebResponse.HTTPRequest Property

Represents the HTTP request message for which this is a response.

Pascal

```
property HTTPRequest: TWebRequest;
```

C++

```
__property TWebRequest HTTPRequest;
```

Description

Read HTTPRequest to determine the header and contents of the HTTP request message that spawned this response message. The properties of HTTPRequest give access to the information sent in the web client request message.

See Also

TWebRequest (see page 15161)

77.1.21.6.14 TWebResponse.LastModified Property

Indicates the date and time that the value of the Content (see page 15207) property was last updated.

Pascal

```
property LastModified: TDateTime;
```

C++

```
__property TDateTime LastModified;
```

Description

Set LastModified to indicate the last time the Content (see page 15207) of the response was modified. This allows web clients to determine whether their local version of that information is up-to-date. When the HTTP request is a HEAD message, this may allow the web client and server to skip a potentially expensive transfer of information.

The value of LastModified is specified in the local time of the machine. The HTTP response message translates this value to the date encoding used by HTTP.

LastModified is represented using the TDateTime type. The integral part of a TDateTime value is the number of days that have passed since December 30, 1899. The fractional part of a TDateTime value is the fraction of a 24-hour day that has elapsed.

See Also

Expires (see page 15212)

Date (see page 15211)

IfModifiedSince (see page 15183)

Describing the Content

77.1.21.6.15 TWebResponse.Location Property

Specifies the URI of a resource the web client should point to for the completion of its request.

Pascal

```
property Location: string;
```

C++

```
__property AnsiString Location;
```

Description

Set Location when the TWebResponse (see page 15191) created a new resource in response to the HTTP request message, or when servicing of the request must be delegated to another URI than the one specified in the HTTP request message.

See Also

URL (see page 15190)

Indicating the Need for Client Action

77.1.21.6.16 TWebResponse.LogMessage Property

Describes the HTTP response message for event logging purposes.

Pascal

```
property LogMessage: string;
```

C++

```
__property AnsiString LogMessage;
```

Description

Set LogMessage to a description of the action carried out by the TWebActionItem (see page 15130) object that services this HTTP response. Usually, LogMessage includes the URI that handled the request message, along with date and time information.

See Also

TWebActionItem (see page 15130)

TWebRequest (see page 15161)

77.1.21.6.17 TWebResponse.Realm Property

Specifies the set of protected-access URIs that includes the URL specified by the HTTP request message.

Pascal

```
property Realm: string;
```

C++

```
__property AnsiString Realm;
```

Description

Set Realm when the response message returns 401 (Unauthorized) or 402 (Payment required). Realm is the name of a set of URIs that require a password or payment before the web client can access them. All the URIs in the realm, including the URL specified in the request message, are made available as soon as the Web client provides the password or payment for the realm. The server application should keep track, internally, of the passwords or payments the client has provided, using the client information provided by the TWebRequest (see page 15161) object.

See Also

StatusCode (see page 15215)

TWebRequest (see page 15161)

URL (see page 15190)

WWWAuthenticate (see page 15218)

77.1.21.6.18 TWebResponse.ReasonString Property

Indicates the status of the response to the web client request as a human-readable string.

Pascal

```
property ReasonString: string;
```

C++

```
__property AnsiString ReasonString;
```

Description

Set ReasonString to inform the human web client of the success or failure of the HTTP request message. Each reason string is associated with a 3-digit status code whose value indicates the status of the response. The HTTP standard defines a number of stock status codes and corresponding reason strings. In addition, server applications can define new reason strings and associate them with unused status codes.

See Also

StatusCode (see page 15215)

Indicating the Response Status

77.1.21.6.19 TWebResponse.Server Property

Provides information about the software used by the server application to handle the request.

Pascal

```
property Server: string;
```

C++

```
__property AnsiString Server;
```

Description

Set Server to a string giving the name and version number of the application that provides the Content (see page 15207) of the response message. The value of Server can contain multiple product identifiers and comments, listed in order of significance and separated by spaces. Product identifiers take the form ProductName/VersionNumber.

See Also

Content (see page 15207)

UserAgent (see page 15191)

Describing the Server Application

77.1.21.6.20 TWebResponse.StatusCode Property

Indicates the status of the response to the HTTP request message as a 3-digit integer.

Pascal

```
property StatusCode: Integer;
```

C++

```
__property int StatusCode;
```

Description

Set StatusCode to inform the web client the status of the HTTP request message. Status codes are standard return values that can be interpreted by all web clients. Each status code has a corresponding reason string that provides a human web client with a more understandable indication of the status.

The following tables lists the standard status codes and their corresponding reason strings.

StatusCode	ReasonString (see page 15215)
100	Continue
101	Switching Protocols
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content (see page 15207)
205	Reset Content (see page 15207)
206	Partial Content (see page 15207)
300	Multiple Choices
301	Moved Permanently
302	Moved Temporarily
303	See Other
304	Not Modified
305	Use Proxy
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Time-out
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large

414	Request-URI Too Large
415	Unsupported Media Type
500	Internal Server (see page 15215) Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Time-out
505	HTTP Version (see page 15217) not supported

Status codes not in the table can be created by the server application. When creating a new status code, the most significant digit should indicate the class of response, as follows:

1xx: Informational (Request received, continuing process)

2xx: Success (Request received, understood, and accepted)

3xx: Redirection (Further action needed to complete the request)

4xx: Client Error (Request contains bad syntax or cannot be serviced)

5xx: Server (see page 15215) Error (Server (see page 15215) failed to handle a valid request)

See Also

ReasonString (see page 15215)

Indicating the Response Status

77.1.21.6.21 TWebResponse.Title Property

Specifies a title for the Content (see page 15207) of the HTTP response message.

Pascal

```
property Title: string;
```

C++

```
__property AnsiString Title;
```

Description

Set the Title property to add a Title header to the HTTP response message. Title is an optional application-specific title associated with the information included in the message.

See Also

Title (see page 15190)

Describing the Content

77.1.21.6.22 TWebResponse.Version Property

Specifies the HTTP protocol version number that the web client is using.

Pascal

```
property Version: string;
```

C++

```
__property AnsiString Version;
```

Description

Read Version to ensure that the properties set in the TWebResponse (see page 15191) object will be understood by the Web client. The value of Version is taken from the ProtocolVersion property of the TWebRequest (see page 15161) object.

For more information on the different HTTP protocol versions, see the Internet standards documentation.

See Also

ProtocolVersion (see page 15186)

77.1.21.6.23 TWebResponse.WWWAuthenticate Property

Lists of one or more comma-separated challenges provided when the request fails with StatusCode (see page 15215) 401 (Unauthorized).

Pascal

```
property WWWAuthenticate: string;
```

C++

```
__property AnsiString WWWAuthenticate;
```

Description

Set WWWAuthenticate when the HTTP request fails because it requires access to a URI in a realm for which the Web client has not provided a password. WWWAuthenticate consists of an authentication scheme name, followed by a comma-separated list of parameter/value pairs that specify the parameters to the authentication scheme. The first parameter/value pair indicates the realm.

For more information about HTTP authentication schemes, see the Internet standards documents.

See Also

Realm (see page 15214)

StatusCode (see page 15215)

Indicating the Need for Client Action

77.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IGetAppDispatcher (see page 15219)	This is class HTTPApp.IGetAppDispatcher.
	IGetWebAppServices (see page 15220)	This is class HTTPApp.IGetWebAppServices.
	IGetWebRequestHandler (see page 15221)	This is class HTTPApp.IGetWebRequestHandler.
	IMultiModuleSupport (see page 15222)	This is class HTTPApp.IMultiModuleSupport.
	IProduceContent (see page 15223)	IProduceContent is the interface for generating the content of a Web page.
	IProduceContentFrom (see page 15225)	This is class HTTPApp.IProduceContentFrom.
	ISetAppDispatcher (see page 15226)	This is class HTTPApp.ISetAppDispatcher.
	IWebAppServices (see page 15227)	IWebAppServices is the interface for setting up and finishing the processing of HTTP request messages.
	IWebDispatch (see page 15230)	
	IWebDispatchActions (see page 15233)	IWebDispatchActions is the interface for components that dispatch HTTP request messages using action items.
	IWebDispatcherAccess (see page 15234)	IWebDispatcherAccess provides access to the current HTTP request and response object.
	IWebExceptionHandler (see page 15236)	This is class HTTPApp.IWebExceptionHandler.
	IWebRequestHandler (see page 15237)	IWebRequestHandler is the interface for objects that dispatch HTTP request messages.

77.2.1 IGetAppDispatcher Interface

Class Hierarchy

HTTPApp.IGetAppDispatcher

Pascal

```
IGetAppDispatcher = interface;
```

C++

```
__interface IGetAppDispatcher;
```

File

HTTPApp

Description

This is class HTTPApp.IGetAppDispatcher.

77.2.1.1 IGetAppDispatcher Members

The following tables list the members exposed by IGetAppDispatcher.

Public Methods

	Name	Description
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.

77.2.1.2 IGetAppDispatcher Methods

The methods of the IGetAppDispatcher class are listed here.

Public Methods

	Name	Description
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.

77.2.1.2.1 IGetAppDispatcher.GetAppDispatcher Method

Pascal

```
function GetAppDispatcher: TComponent;
```

C++

```
__fastcall TComponent * GetAppDispatcher();
```

Description

This is GetAppDispatcher, a member of class IGetAppDispatcher.

77.2.2 IGetWebAppServices Interface

Class Hierarchy

HTTPApp.IGetWebAppServices

Pascal

```
IGetWebAppServices = interface;
```

C++

```
__interface IGetWebAppServices;
```

File

HTTPApp

Description

This is class HTTPApp.IGetWebAppServices.

77.2.2.1 IGetWebAppServices Members

The following tables list the members exposed by IGetWebAppServices.

Public Methods

	Name	Description
	GetWebAppServices (see page 15221)	This is GetWebAppServices, a member of class IGetWebAppServices.

77.2.2.2 IGetWebAppServices Methods

The methods of the IGetWebAppServices class are listed here.

Public Methods

	Name	Description
	GetWebAppServices (see page 15221)	This is GetWebAppServices, a member of class IGetWebAppServices.

77.2.2.2.1 IGetWebAppServices.GetWebAppServices Method

Pascal

```
function GetWebAppServices: IWebAppServices;
```

C++

```
__fastcall IWebAppServices GetWebAppServices();
```

Description

This is GetWebAppServices, a member of class IGetWebAppServices.

77.2.3 IGetWebRequestHandler Interface

Class Hierarchy

HTTPApp.IGetWebRequestHandler

Pascal

```
IGetWebRequestHandler = interface;
```

C++

```
__interface IGetWebRequestHandler;
```

File

HTTPApp

Description

This is class HTTPApp.IGetWebRequestHandler.

77.2.3.1 IGetWebRequestHandler Members

The following tables list the members exposed by IGetWebRequestHandler.

Public Methods

	Name	Description
	GetWebRequestHandler (see page 15222)	This is GetWebRequestHandler, a member of class IGetWebRequestHandler.

77.2.3.2 IGetWebRequestHandler Methods

The methods of the IGetWebRequestHandler class are listed here.

Public Methods

	Name	Description
	GetWebRequestHandler (see page 15222)	This is GetWebRequestHandler, a member of class IGetWebRequestHandler.

77.2.3.2.1 IGetWebRequestHandler.GetWebRequestHandler Method

Pascal

```
function GetWebRequestHandler: IWebRequestHandler;
```

C++

```
__fastcall IWebRequestHandler GetWebRequestHandler();
```

Description

This is GetWebRequestHandler, a member of class IGetWebRequestHandler.

77.2.4 IMultiModuleSupport Interface

Class Hierarchy

HTTPApp.IMultiModuleSupport

Pascal

```
IMultiModuleSupport = interface;
```

C++

```
__interface IMultiModuleSupport;
```

File

HTTPApp

Description

This is class HTTPApp.IMultiModuleSupport.

77.2.4.1 IMultiModuleSupport Members

The following tables list the members exposed by IMultiModuleSupport.

Public Methods

	Name	Description
	FinishContext (see page 15223)	This is FinishContext, a member of class IMultiModuleSupport.
	InitContext (see page 15223)	This is InitContext, a member of class IMultiModuleSupport.
	InitModule (see page 15223)	This is InitModule, a member of class IMultiModuleSupport.

77.2.4.2 IMultiModuleSupport Methods

The methods of the IMultiModuleSupport class are listed here.

Public Methods

	Name	Description
	FinishContext (see page 15223)	This is FinishContext, a member of class IMultiModuleSupport.
	InitContext (see page 15223)	This is InitContext, a member of class IMultiModuleSupport.
	InitModule (see page 15223)	This is InitModule, a member of class IMultiModuleSupport.

77.2.4.2.1 IMultiModuleSupport.FinishContext Method

Pascal

```
procedure FinishContext;
```

C++

```
__fastcall FinishContext();
```

Description

This is FinishContext, a member of class IMultiModuleSupport.

77.2.4.2.2 IMultiModuleSupport.InitContext Method

Pascal

```
procedure InitContext(Request: TWebRequest; Response: TWebResponse);
```

C++

```
__fastcall InitContext(TWebRequest Request, TWebResponse Response);
```

Description

This is InitContext, a member of class IMultiModuleSupport.

77.2.4.2.3 IMultiModuleSupport.InitModule Method

Pascal

```
procedure InitModule(AModule: TComponent);
```

C++

```
__fastcall InitModule(TComponent * AModule);
```

Description

This is InitModule, a member of class IMultiModuleSupport.

77.2.5 IProduceContent Interface

IProduceContent is the interface for generating the content of a Web page.

Class Hierarchy

HTTPApp.IProduceContent

Pascal

```
IProduceContent = interface;
```

C++

```
__interface IProduceContent;
```

File

HTTPApp

Description

Content producers (such as TAdapaterPageProducer) implement the IProduceContent interface. This interface defines a single method, ProduceContent (see page 15224), which returns the content to assign to an HTTP response message in a WebSnap application.

Note: In C++, method declarations that use IProduceContent use the _di_IProduceContent type instead. This type is a DelphiInterface wrapper around the IProduceContent interface:

Note: typedef System::DelphiInterface (see page 22167)< IProduceContent > _di_IProduceContent;

See Also

TCustomContentProducer (see page 15102)

TCustomWebPageModule (see page 25357)

TAdapterPagePRoducer

TWebResponse (see page 15191)

77.2.5.1 IProduceContent Members

The following tables list the members exposed by IProduceContent.

Public Methods

	Name	Description
	ProduceContent (see page 15224)	Returns the content that the producer creates.

77.2.5.2 IProduceContent Methods

The methods of the IProduceContent class are listed here.

Public Methods

	Name	Description
	ProduceContent (see page 15224)	Returns the content that the producer creates.

77.2.5.2.1 IProduceContent.ProduceContent Method

Returns the content that the producer creates.

Pascal

```
function ProduceContent: string;
```

C++

```
__fastcall AnsiString ProduceContent();
```

Description

Call ProduceContent to retrieve the content for using in an HTTP response message. In most cases, this content is a string of HTML commands defining a Web page, although it can be any valid MIME type.

See Also

Content (see page 15106)

77.2.6 IProduceContentFrom Interface

Class Hierarchy

HTTPApp.IProduceContentFrom

Pascal

```
IProduceContentFrom = interface;
```

C++

```
__interface IProduceContentFrom;
```

File

HTTPApp

Description

This is class HTTPApp.IProduceContentFrom.

77.2.6.1 IProduceContentFrom Members

The following tables list the members exposed by IProduceContentFrom.

Public Methods

	Name	Description
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.

77.2.6.2 IProduceContentFrom Methods

The methods of the IProduceContentFrom class are listed here.

Public Methods

	Name	Description
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.

77.2.6.2.1 IProduceContentFrom.ProduceContentFromStream Method

Pascal

```
function ProduceContentFromStream(Stream: TStream): string;
```

C++

```
__fastcall AnsiString ProduceContentFromStream(TStream Stream);
```

Description

This is ProduceContentFromStream, a member of class IProduceContentFrom.

77.2.6.2.2 IProduceContentFrom.ProduceContentFromString Method

Pascal

```
function ProduceContentFromString(const S: string): string;
```

C++

```
__fastcall AnsiString ProduceContentFromString(const AnsiString S);
```

Description

This is ProduceContentFromString, a member of class IProduceContentFrom.

77.2.7 ISetAppDispatcher Interface

Class Hierarchy

HTTPApp.ISetAppDispatcher

Pascal

```
ISetAppDispatcher = interface;
```

C++

```
__interface ISetAppDispatcher;
```

File

HTTPApp

Description

This is class HTTPApp.ISetAppDispatcher.

77.2.7.1 ISetAppDispatcher Members

The following tables list the members exposed by ISetAppDispatcher.

Public Methods

	Name	Description
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.

77.2.7.2 ISetAppDispatcher Methods

The methods of the ISetAppDispatcher class are listed here.

Public Methods

	Name	Description
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.

77.2.7.2.1 ISetAppDispatcher.SetAppDispatcher Method

Pascal

```
procedure SetAppDispatcher(const ADispatcher: TComponent);
```

C++

```
__fastcall SetAppDispatcher(const TComponent * ADispatcher);
```

Description

This is SetAppDispatcher, a member of class ISetAppDispatcher.

77.2.8 IWebAppServices Interface

IWebAppServices is the interface for setting up and finishing the processing of HTTP request messages.

Class Hierarchy

HTTPApp.IWebAppServices

Pascal

```
IWebAppServices = interface;
```

C++

```
__interface IWebAppServices;
```

File

HTTPApp

Description

WebSnap applications use the IWebAppServices interface to drive the processing of incoming request messages. When the WebSnap application receives an HTTP request message it

1. Calls InitContext (see page 15229) to set up a Web context for the request.
2. Calls HandleRequest (see page 15229) to forward the request to the components that process it.
3. Calls FinishContext (see page 15228) to shut down the Web context in an orderly fashion.

In addition to providing access to these core methods of the application, IWebAppServices introduces the ExceptionHandler (see page 15230) property. The ExceptionHandler (see page 15230) property allows the application access to the component that handles any exceptions that escape all of the try blocks in the application while processing the message.

Note: Method declarations that use IWebAppServices use the _di_IWebAppServices type instead. This type is a DelphiInterface wrapper around the IWebAppServices interface:

Note: typedef System::DelphiInterface (see page 22167)< IWebAppServices > _di_IWebAppServices;

See Also

TMultiModuleWebAppServices (see page 24933)

AppServices

WebContext

77.2.8.1 IWebAppServices Members

The following tables list the members exposed by IWebAppServices.

Public Methods

	Name	Description
	FinishContext (see page 15228)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetExceptionHandler (see page 15229)	Returns the object that handles exceptions that arise in the WebSnap application.
	HandleRequest (see page 15229)	Directs the processing of the current request message.
	InitContext (see page 15229)	Creates and initializes the Web context for the current Web request.

Public Properties

	Name	Description
R	ExceptionHandler (see page 15230)	Indicates the object that handles exceptions that arise when processing a request message.

77.2.8.2 IWebAppServices Methods

The methods of the IWebAppServices class are listed here.

Public Methods

	Name	Description
	FinishContext (see page 15228)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetExceptionHandler (see page 15229)	Returns the object that handles exceptions that arise in the WebSnap application.
	HandleRequest (see page 15229)	Directs the processing of the current request message.
	InitContext (see page 15229)	Creates and initializes the Web context for the current Web request.

77.2.8.2.1 IWebAppServices.FinishContext Method

Performs any final tasks after the WebSnap application has finished processing a request.

Pascal

```
procedure FinishContext;
```

C++

```
__fastcall FinishContext();
```

Description

FinishContext is called as the last step before a WebSnap application sends a response message back to the caller. FinishContext performs any final tasks, and frees all the objects and components that were created to handle the request message (including the Web context).

See Also

InitContext (see page 15229)

WebContext (see page 24687)

77.2.8.2.2 IWebAppServices.GetExceptionHandler Method

Returns the object that handles exceptions that arise in the WebSnap application.

Pascal

```
function GetExceptionHandler: TObject;
```

C++

```
__fastcall TObject * GetExceptionHandler();
```

Description

GetExceptionHandler is the read implementation of the ExceptionHandler (see page 15230) property.

See Also

ExceptionHandler (see page 15230)

77.2.8.2.3 IWebAppServices.HandleRequest Method

Directs the processing of the current request message.

Pascal

```
function HandleRequest: Boolean;
```

C++

```
__fastcall Boolean HandleRequest();
```

Description

When the WebSnap application receives a request message, it makes the following calls:

1. The InitContext (see page 15229) method is called to generate a Web context for the request.
2. HandleRequest is called to generate a response to the request.
3. FinishContext (see page 15228) is called to perform any final cleanup.

HandleRequest locates the dispatcher components that handle the request and forwards the request to a dispatcher for processing.

HandleRequest returns true if the request is handled and the response should be sent back to the caller (if it was not already sent). It returns false if the WebSnap application could not handle the request.

See Also

InitContext (see page 15229)

FinishContext (see page 15228)

77.2.8.2.4 IWebAppServices.InitContext Method

Creates and initializes the Web context for the current Web request.

Pascal

```
procedure InitContext(WebModuleList: TAbstractWebModuleList; Request: TWebRequest; Response:
TWebResponse);
```

C++

```
__fastcall InitContext(TAbstractWebModuleList WebModuleList, TWebRequest Request, TWebResponse Response);
```

Description

InitContext is called when the WebSnap application first receives a request message to set up the Web context before the application attempts to handle the request. It creates and sets up the global Web context, which can be accessed by calling the global WebContext function. It also performs any other initializations that ready the application for handling a new request message.

WebModuleList lists the currently active Web modules in the WebSnap application and provides access to all Web module factories so that the application can create any new modules it needs for processing the request message.

Request describes the new request message.

Response is the response object that the WebSnap application should fill out when handling the current request.

See Also

FinishContext (see page 15228)

WebContext (see page 24687)

77.2.8.3 IWebAppServices Properties

The properties of the IWebAppServices class are listed here.

Public Properties

	Name	Description
R	ExceptionHandler (see page 15230)	Indicates the object that handles exceptions that arise when processing a request message.

77.2.8.3.1 IWebAppServices.ExceptionHandler Property

Indicates the object that handles exceptions that arise when processing a request message.

Pascal

```
property ExceptionHandler: TObject;
```

C++

```
__property TObject * ExceptionHandler;
```

Description

ExceptionHandler is the object that the WebSnap application calls when an exception is raised while trying to process a request message. The request handler must implement the IWebExceptionHandler (see page 15236) interface, which includes only a single method: HandleException.

77.2.9 IWebDispatch Interface

Class Hierarchy

HTTPApp.IWebDispatch

Pascal

```
IWebDispatch = interface;
```

C++

```
__interface IWebDispatch;
```

File

HTTPApp

See Also

TCustomWebDispatcher (see page 15110)

77.2.9.1 IWebDispatch Members

The following tables list the members exposed by IWebDispatch.

Public Methods

	Name	Description
	DispatchEnabled (see page 15231)	
	DispatchMask (see page 15232)	
	DispatchMethodType (see page 15232)	
	DispatchRequest (see page 15232)	

Public Properties

	Name	Description
R	Enabled (see page 15232)	
R	Mask (see page 15233)	
R	MethodType (see page 15233)	This is MethodType, a member of class IWebDispatch.

77.2.9.2 IWebDispatch Methods

The methods of the IWebDispatch class are listed here.

Public Methods

	Name	Description
	DispatchEnabled (see page 15231)	
	DispatchMask (see page 15232)	
	DispatchMethodType (see page 15232)	
	DispatchRequest (see page 15232)	

77.2.9.2.1 IWebDispatch.DispatchEnabled Method

Pascal

```
function DispatchEnabled: Boolean;
```

C++

```
__fastcall Boolean DispatchEnabled();
```

See Also

Enabled (see page 15232)

77.2.9.2.2 IWebDispatch.DispatchMask Method

Pascal

```
function DispatchMask: TMask;
```

C++

```
__fastcall TMask DispatchMask();
```

See Also

Mask (see page 15233)

77.2.9.2.3 IWebDispatch.DispatchMethodType Method

Pascal

```
function DispatchMethodType: TMethodType;
```

C++

```
__fastcall TMethodType DispatchMethodType();
```

See Also

MethodType (see page 15233)

77.2.9.2.4 IWebDispatch.DispatchRequest Method

Pascal

```
function DispatchRequest(Sender: TObject; Request: TWebRequest; Response: TWebResponse):
Boolean;
```

C++

```
__fastcall Boolean DispatchRequest(TObject * Sender, TWebRequest Request, TWebResponse
Response);
```

77.2.9.3 IWebDispatch Properties

The properties of the IWebDispatch class are listed here.

Public Properties

	Name	Description
R	Enabled (see page 15232)	
R	Mask (see page 15233)	
R	MethodType (see page 15233)	This is MethodType, a member of class IWebDispatch.

77.2.9.3.1 IWebDispatch.Enabled Property

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

See Also

DispatchEnabled (see page 15231)

77.2.9.3.2 IWebDispatch.Mask Property

Pascal

```
property Mask: TMask;
```

C++

```
__property TMask Mask;
```

See Also

DispatchMask (see page 15232)

77.2.9.3.3 IWebDispatch.MethodType Property

Pascal

```
property MethodType: TMethodType;
```

C++

```
__property TMethodType MethodType;
```

Description

This is MethodType, a member of class IWebDispatch.

77.2.10 IWebDispatchActions Interface

IWebDispatchActions is the interface for components that dispatch HTTP request messages using action items.

Class Hierarchy

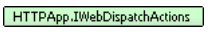

Pascal

```
IWebDispatchActions = interface;
```

C++

```
__interface IWebDispatchActions;
```

File

HTTPApp

Description

Web dispatcher components (TWebDispatcher (see page 15147) and TWebModule (see page 15154)) implement the IWebDispatchActions interface. This interface serves primarily to identify components that use action items to handle HTTP request messages.

To obtain an IWebDispatchActions interface, use the DispatchActions property of a TWebAppComponents object.

Note: In C++, method declarations that use IWebDispatchActions use the _di_IWebDispatchActions type instead. This type is a DelphiInterface wrapper around the IWebDispatchActions interface:

Note: typedef System::DelphiInterface (see page 22167)< IWebDispatchActions > _di_IWebDispatchActions;

See Also

TWebDispatcher (see page 15147)

TWEbModule

TWebAppComponents (see page 24958)

TCustomWebAppComponents (see page 24906)

77.2.11 IWebDispatcherAccess Interface

IWebDispatcherAccess provides access to the current HTTP request and response object.

Class Hierarchy

HTTPApp.IWebDispatcherAccess

Pascal

```
IWebDispatcherAccess = interface;
```

C++

```
__interface IWebDispatcherAccess;
```

File

HTTPApp

Description

The IWebDispatcherAccess interface declares two methods, Request (see page 15235) and Response (see page 15235). These return the objects that represent the current HTTP request message that is being processed and the emerging HTTP response message that will be sent back to the caller.

In Web Broker applications, a TWebDispatcher (see page 15147) or TWebModule (see page 15154) component implements this interface. In WebSnap applications, a TWebAppComponents component implements this interface.

Note: In C++, method declarations that use IWebDispatcherAccess use the _di_IWebDispatcherAccess type instead. This type is a DelphiInterface wrapper around the IWebDispatcherAccess interface:

Note: typedef System::DelphiInterface (see page 22167)< IWebDispatcherAccess > _di_IWebDispatcherAccess;

See Also

TCustomContentProducer (see page 15102)

TWebDispatcher (see page 15147)

TWebAppComponents (see page 24958)

TWebModule (see page 15154)

77.2.11.1 IWebDispatcherAccess Members

The following tables list the members exposed by IWebDispatcherAccess.

Public Methods

	Name	Description
	Request (see page 15235)	Returns the Web request object for the HTTP request message to which the application is responding.
	Response (see page 15235)	Returns the Web response object for the HTTP response message that the application is assembling.

77.2.11.2 IWebDispatcherAccess Methods

The methods of the IWebDispatcherAccess class are listed here.

Public Methods

	Name	Description
	Request (see page 15235)	Returns the Web request object for the HTTP request message to which the application is responding.
	Response (see page 15235)	Returns the Web response object for the HTTP response message that the application is assembling.

77.2.11.2.1 IWebDispatcherAccess.Request Method

Returns the Web request object for the HTTP request message to which the application is responding.

Pascal

```
function Request: TWebRequest;
```

C++

```
__fastcall TWebRequest Request();
```

Description

Call Request to obtain access to the current Web request object. The request object provides access to the headers of the request message, as well as the content of the message.

See Also

Response (see page 15235)

77.2.11.2.2 IWebDispatcherAccess.Response Method

Returns the Web response object for the HTTP response message that the application is assembling.

Pascal

```
function Response: TWebResponse;
```

C++

```
__fastcall TWebResponse Response();
```

Description

Call Response to obtain access to the current Web response object. The response object provides access to the headers of the response message, as well as the content of the message.

See Also

Request (see page 15235)

77.2.12 IWebExceptionHandler Interface

Class Hierarchy

HTTPApp.IWebExceptionHandler

Pascal

```
IWebExceptionHandler = interface;
```

C++

```
__interface IWebExceptionHandler;
```

File

HTTPApp

Description

This is class HTTPApp.IWebExceptionHandler.

77.2.12.1 IWebExceptionHandler Members

The following tables list the members exposed by IWebExceptionHandler.

Public Methods

	Name	Description
	HandleException (see page 15236)	This is HandleException, a member of class IWebExceptionHandler.

77.2.12.2 IWebExceptionHandler Methods

The methods of the IWebExceptionHandler class are listed here.

Public Methods

	Name	Description
	HandleException (see page 15236)	This is HandleException, a member of class IWebExceptionHandler.

77.2.12.2.1 IWebExceptionHandler.HandleException Method

Pascal

```
procedure HandleException(E: Exception; var Handled: Boolean);
```

C++

```
__fastcall HandleException(Exception E, Boolean Handled);
```

Description

This is HandleException, a member of class IWebExceptionHandler.

77.2.13 IWebRequestHandler Interface

IWebRequestHandler is the interface for objects that dispatch HTTP request messages.

Class Hierarchy

HTTPApp.IWebRequestHandler

Pascal

```
IWebRequestHandler = interface;
```

C++

```
__interface IWebRequestHandler;
```

File

HTTPApp

Description

Dispatcher components such as TPageDispatcher , TAdapterDispatcher, and TWebDispatcher (see page 15147) implement the IWebRequestHandler interface. This interface includes only a single method, HandleRequest (see page 15238), which processes an HTTP request message, filling out the corresponding response, which can then be returned to the caller.

Note: In C++, method declarations that use IWebRequestHandler use the _di_IWebRequestHandler type instead. This type is a DelphiInterface wrapper around the IWebRequestHandler interface:

Note: typedef System::DelphiInterface (see page 22167)< IWebRequestHandler > _di_IWebRequestHandler;

See Also

TWebRequest (see page 15161)

TWebResponse (see page 15191)

TMultiModuleWebAppServices (see page 24933)

TPageDispatcher (see page 24949)

TAdapterDispatcher (see page 24840)

TWebDispatcher (see page 15147)

77.2.13.1 IWebRequestHandler Members

The following tables list the members exposed by IWebRequestHandler.

Public Methods

	Name	Description
	HandleRequest (see page 15238)	Processes an HTTP request message.

77.2.13.2 IWebRequestHandler Methods

The methods of the IWebRequestHandler class are listed here.

Public Methods

	Name	Description
	HandleRequest (see page 15238)	Processes an HTTP request message.

77.2.13.2.1 IWebRequestHandler.HandleRequest Method

Processes an HTTP request message.

Pascal

```
function HandleRequest(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(TWebRequest Request, TWebResponse Response);
```

Description

Call HandleRequest to process an HTTP request message and obtain the response that should be sent back to the caller.

Request is the request object that describes the request to process.

Response is the response object that should be filled out when the request is processed.

77.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DayOfWeekStr (see page 15239)	Converts a TDateTime into a string representing the day of the week.
	DosPathToUnixPath (see page 15240)	Converts a DOS-compatible path specification to a Unix-compatible path specification.
	ExtractHeaderFields (see page 15240)	Parses a multi-valued string into its constituent fields.
	ExtractHTTPFields (see page 15241)	Parses a multi-valued string into its constituent fields.
	HTMLDecode (see page 15241)	This is function HTTPApp.HTMLDecode.
	HTMLEncode (see page 15242)	This is function HTTPApp.HTMLEncode.
	HTTPDecode (see page 15242)	Decodes a string that includes HTTP escape characters.
	HTTPEncode (see page 15243)	Converts a string into a form that contains no values that are illegal in an HTTP message header.
	MonthStr (see page 15243)	Converts a TDateTime into a string representing the month.
	ParseDate (see page 15244)	Converts a date value from an HTTP message header into a TDateTime.
	RegisterContentParser (see page 15244)	This is function HTTPApp.RegisterContentParser.
	StatusString (see page 15244)	Returns the explanatory string associated with one of the predefined HTTP response status codes.
	UnixPathToDosPath (see page 15245)	Converts a Unix-compatible path specification to a Dos-compatible path specification.

77.3.1 HTTPApp.DayOfWeekStr Function

Converts a TDateTime into a string representing the day of the week.

Pascal

```
function DayOfWeekStr(DateTime: TDateTime): string;
```

C++

```
AnsiString DayOfWeekStr(TDateTime DateTime);
```

File

HTTPApp

Description

HTTP message headers permit several formats for the representation of date and time values. DayOfWeekStr converts a TDateTime into a string representing the day of the week. DayOfWeekStr allows server applications to work with date values taken from HTTP request messages, without worrying about the details of how they are formatted.

See Also

MonthStr (see page 15243)

77.3.2 HTTPApp.DosPathToUnixPath Function

Converts a DOS-compatible path specification to a Unix-compatible path specification.

Pascal

```
function DosPathToUnixPath(const Path: string): string;
```

C++

```
AnsiString DosPathToUnixPath(const AnsiString Path);
```

File

HTTPApp

Description

Use DosPathToUnixPath to convert a path name on the local machine to a path name that can be sent in an HTTP message, or that will be used by a Unix-based application. DosPathToUnixPath converts all backslash characters (\) to forward slashes (/).

The Path parameter is the Dos-compatible path. The return value is the corresponding Unix-compatible path.

See Also

UnixPathToDosPath (see page 15245)

77.3.3 HTTPApp.ExtractHeaderFields Function

Parses a multi-valued string into its constituent fields.

Pascal

```
procedure ExtractHeaderFields(Separators: TSysCharSet; WhiteSpace: TSysCharSet; Content:
PChar; Strings: TStrings; Decode: Boolean; StripQuotes: Boolean = False);
```

C++

```
ExtractHeaderFields(TSysCharSet Separators, TSysCharSet WhiteSpace, const char * Content,
TStrings Strings, Boolean Decode, Boolean StripQuotes = False);
```

File

HTTPApp

Description

ExtractHeaderFields is a general utility to parse multi-valued HTTP header strings into separate substrings.

Separators is a set of characters that are used to separate individual values within the multi-valued string.

WhiteSpace is a set of characters that are to be ignored when parsing the string.

Content is the multi-valued string to be parsed.

Strings is the TStrings object that receives the individual values that are parsed from Content.

StripQuotes determines whether the surrounding quotes are removed from the resulting items. When StripQuotes is true, surrounding quotes are removed before substrings are added to Strings.

Note: Characters contained in Separators or WhiteSpace are treated as part of a value substring if the substring is surrounded by single or double quote marks. HTTP escape characters are converted using the HTTPDecode (see page 15242) function.

See Also

HTTPDecode (see page 15242)

TStrings

77.3.4 HTTPApp.ExtractHTTPFields Function

Parses a multi-valued string into its constituent fields.

Pascal

```
procedure ExtractHTTPFields(Separators: TSysCharSet; WhiteSpace: TSysCharSet; Content: PChar;
Strings: TStrings; StripQuotes: Boolean = False);
```

C++

```
ExtractHTTPFields(TSysCharSet Separators, TSysCharSet WhiteSpace, const char * Content,
TStrings Strings, Boolean StripQuotes = False);
```

File

HTTPApp

Description

ExtractHTTPFields is a general utility to parse multi-valued HTTP header strings into separate substrings.

Separators is a set of characters that are used to separate individual values within the multi-valued string.

WhiteSpace is a set of characters that are to be ignored when parsing the string.

Content is the multi-valued string to be parsed.

Strings is the TStrings object that receives the individual values that are parsed from Content.

StripQuotes determines whether the surrounding quotes are removed from the resulting items. When StripQuotes is true, surrounding quotes are removed before substrings are added to Strings.

Note: Characters contained in Separators or WhiteSpace are treated as part of a value substring if the substring is surrounded by single or double quote marks. HTTP escape characters are converted using the HTTPDecode (see page 15242) function.

See Also

HTTPDecode (see page 15242)

TStrings

77.3.5 HTTPApp.HTMLDecode Function

Pascal

```
function HTMLDecode(const AStr: String): String;
```

C++

```
AnsiString HTMLDecode(const AnsiString AStr);
```

File

HTTPApp

Description

This is function HTTPApp.HTMLDecode.

77.3.6 HTTPApp.HTMLEncode Function

Pascal

```
function HTMLEncode(const AStr: String): String;
```

C++

```
AnsiString HTMLEncode(const AnsiString AStr);
```

File

HTTPApp

Description

This is function HTTPApp.HTMLEncode.

77.3.7 HTTPApp.HTTPDecode Function

Decodes a string that includes HTTP escape characters.

Pascal

```
function HTTPDecode(const AStr: String): string;
```

C++

```
AnsiString HTTPDecode(const AnsiString AStr);
```

File

HTTPApp

Description

HTTPDecode decodes the escape characters (+ and %) in a string that comes from an HTTP message header. Plus characters (+) are converted into spaces. Percent signs (%) indicate that the next two characters are the hex representation of a character. If a percent sign is followed by another percent sign (%%), it is converted into a single percent sign.

For example, the string

%%+%3f

is converted into

% ?

The AStr parameter is the undecoded string. The return value is the converted version of the string.

See Also

ExtractHTTPFields (see page 15241)

HTTPEncode (see page 15243)

77.3.8 HTTPApp.HTTPEncode Function

Converts a string into a form that contains no values that are illegal in an HTTP message header.

Pascal

```
function HTTPEncode(const AStr: String): string;
```

C++

```
AnsiString HTTPEncode(const AnsiString AStr);
```

File

HTTPApp

Description

HTTPEncode converts all characters in the AStr parameter except for the letters A through Z (and a through z), numerals 0 through 9, the asterisk (*), dollar sign ($), exclamation point (!), at sign (@), period (.), underscore (_), single quote ('), comma (,) parentheses, and hyphen (-). Spaces are converted to plus characters (+), and all other characters are converted into hex values preceded by the percent sign (%).

For example, the string

% ?

is converted into

%%+%3f

HTTPEncode returns the converted string.

See Also

HTTPDecode (see page 15242)

77.3.9 HTTPApp.MonthStr Function

Converts a TDateTime into a string representing the month.

Pascal

```
function MonthStr(DateTime: TDateTime): string;
```

C++

```
AnsiString MonthStr(TDateTime DateTime);
```

File

HTTPApp

Description

HTTP message headers permit several formats for the representation of date and time values. MonthStr converts a TDateTime into a string representing the month. MonthStr allows server applications to work with date values taken from HTTP request messages, without worrying about the details of how they are formatted.

See Also

DayOfWeekStr (see page 15239)

77.3.10 HTTPApp.ParseDate Function

Converts a date value from an HTTP message header into a TDateTime.

Pascal

```
function ParseDate(const DateStr: string): TDateTime;
```

C++

```
TDateTime ParseDate(const AnsiString DateStr);
```

File

HTTPApp

Description

HTTP message headers permit several formats for the representation of date and time values. ParseDate converts from any of these formats into a TDateTime. ParseDate allows server applications to work with date values taken from HTTP request messages, without worrying about the details of how they are formatted.

Use ParseDate when interpreting temporal header values in HTTP request messages when they are not already parsed as a property of TWebRequest (see page 15161). TWebRequest (see page 15161) parses the Date header and Expires header automatically.

See Also

TWebRequest (see page 15161)

77.3.11 HTTPApp.RegisterContentParser Function

Pascal

```
procedure RegisterContentParser(AClass: TContentParserClass);
```

C++

```
RegisterContentParser(TContentParserClass AClass);
```

File

HTTPApp

Description

This is function HTTPApp.RegisterContentParser.

77.3.12 HTTPApp.StatusString Function

Returns the explanatory string associated with one of the predefined HTTP response status codes.

Pascal

```
function StatusString(StatusCode: Integer): string;
```

C++

```
AnsiString StatusString(int StatusCode);
```

File

HTTPApp

Description

The HTTP protocol specification defines a number of standard status codes that indicate the status of an HTTP request message. HTTP applications interpret these status codes, but for human beings, each of these codes has a corresponding string that provides a more understandable indication of the status of the HTTP request.

Call StatusString to obtain the string that corresponds to a standard status code. If the StatusCode parameter is not one of the codes defined in the HTTP standard, StatusString returns an empty string.

See Also

TWebResponse (see page 15191)

77.3.13 HTTPApp.UnixPathToDosPath Function

Converts a Unix-compatible path specification to a Dos-compatible path specification.

Pascal

```
function UnixPathToDosPath(const Path: string): string;
```

C++

```
AnsiString UnixPathToDosPath(const AnsiString Path);
```

File

HTTPApp

Description

Use UnixPathToDosPath to convert a path name obtained from an HTTP message or other Unix-based application to a path name that can be used by Windows. UnixPathToDosPath converts all forward slashes (/) to backslash characters (\).

The Path parameter is the Unix-compatible path. The return value is the corresponding Dos-compatible path.

See Also

DosPathToUnixPath (see page 15240)

77.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TMethodType (see page 15246)	This is record HTTPApp.TMethodType.
	TWebModuleCacheMode (see page 15246)	This is record HTTPApp.TWebModuleCacheMode.
	TWebModuleCreateMode (see page 15247)	This is record HTTPApp.TWebModuleCreateMode.

77.4.1 HTTPApp.TMethodType Enumeration

Pascal

```
TMethodType = (
  mtAny,
  mtGet,
  mtPut,
  mtPost,
  mtHead
);
```

C++

```
enum TMethodType {
  mtAny,
  mtGet,
  mtPut,
  mtPost,
  mtHead
};
```

File

HTTPApp

Description

This is record HTTPApp.TMethodType.

77.4.2 HTTPApp.TWebModuleCacheMode Enumeration

Pascal

```
TWebModuleCacheMode = (
  caCache,
  caDestroy
);
```

C++

```
enum TWebModuleCacheMode {
  caCache,
  caDestroy
```

```
};
```

File

HTTPApp

Description

This is record HTTPApp.TWebModuleCacheMode.

77.4.3 HTTPApp.TWebModuleCreateMode Enumeration

Pascal

```
TWebModuleCreateMode = (
  crOnDemand,
  crAlways
);
```

C++

```
enum TWebModuleCreateMode {
  crOnDemand,
  crAlways
};
```

File

HTTPApp

Description

This is record HTTPApp.TWebModuleCreateMode.

77.5 Types

The following table lists types in this documentation.

Types

Name	Description
TContentParserClass (see page 15248)	This is type HTTPApp.TContentParserClass.
THTTPMethodEvent (see page 15248)	This is type HTTPApp.THTTPMethodEvent.
TModuleAddedProc (see page 15249)	This is type HTTPApp.TModuleAddedProc.
TWebExceptionEvent (see page 15249)	This is type HTTPApp.TWebExceptionEvent.
TWebModuleContext (see page 15249)	This is type HTTPApp.TWebModuleContext.

77.5.1 HTTPApp.TContentParserClass Type

Pascal

```
TContentParserClass = class of TAbstractContentParser;
```

C++

```
TAbstractContentParser TContentParserClass;
```

File

HTTPApp

Description

This is type HTTPApp.TContentParserClass.

77.5.2 HTTPApp.THTTPMethodEvent Type

Pascal

```
THTTPMethodEvent = procedure (Sender: TObject; Request: TWebRequest; Response: TWebResponse;
var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; Request: TWebRequest; Response: TWebResponse; var Handled: Boolean) (
THTTPMethodEvent)();
```

File

HTTPApp

Description

This is type HTTPApp.THTTPMethodEvent.

77.5.3 HTTPApp.TModuleAddedProc Type

Pascal

```
TModuleAddedProc = procedure (AWebModule: TComponent) of object;
```

C++

```
(AWebModule: TComponent) ( TModuleAddedProc)();
```

File

HTTPApp

Description

This is type HTTPApp.TModuleAddedProc.

77.5.4 HTTPApp.TWebExceptionEvent Type

Pascal

```
TWebExceptionEvent = procedure (Sender: TObject; E: Exception; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; E: Exception; var Handled: Boolean) ( TWebExceptionEvent)();
```

File

HTTPApp

Description

This is type HTTPApp.TWebExceptionEvent.

77.5.5 HTTPApp.TWebModuleContext Type

Pascal

```
TWebModuleContext = TObject;
```

C++

```
TObject * TWebModuleContext;
```

File

HTTPApp

Description

This is type HTTPApp.TWebModuleContext.

77.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
MAX_DATETIMES (see page 15250)	This is constant HTTPApp.MAX_DATETIMES.
MAX_INTEGERS (see page 15250)	This is constant HTTPApp.MAX_INTEGERS.
MAX_STRINGS (see page 15251)	This is constant HTTPApp.MAX_STRINGS.
sDateFormat (see page 15251)	This is constant HTTPApp.sDateFormat.

77.6.1 HTTPApp.MAX_DATETIMES Constant

Pascal

```
MAX_DATETIMES = 3;
```

C++

```
MAX_DATETIMES = 3;
```

File

HTTPApp

Description

This is constant HTTPApp.MAX_DATETIMES.

77.6.2 HTTPApp.MAX_INTEGERS Constant

Pascal

```
MAX_INTEGERS = 1;
```

C++

```
MAX_INTEGERS = 1;
```

File

HTTPApp

Description

This is constant HTTPApp.MAX_INTEGERS.

77.6.3 HTTPApp.MAX_STRINGS Constant

Pascal

```
MAX_STRINGS = 12;
```

C++

```
MAX_STRINGS = 12;
```

File

HTTPApp

Description

This is constant HTTPApp.MAX_STRINGS.

77.6.4 HTTPApp.sDateFormat Constant

Pascal

```
sDateFormat = '"%s", dd "%s" yyyy hh:nn:ss';
```

C++

```
sDateFormat = '"%s", dd "%s" yyyy hh:nn:ss';
```

File

HTTPApp

Description

This is constant HTTPApp.sDateFormat.

78 HTTPProd Namespace

This is namespace HTTPProd.

Topics

Name	Description
Classes (see page 15255)	The following table lists classes in this documentation.
Interfaces (see page 15317)	The following table lists interfaces in this documentation.
Functions (see page 15333)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 15336)	The following table lists structs, records, enums in this documentation.
Types (see page 15338)	The following table lists types in this documentation.
Variables (see page 15340)	The following table lists variables in this documentation.
Constants (see page 15341)	The following table lists constants in this documentation.

Classes

	Name	Description
	EScriptError (see page 15255)	This is class HTTPProd.EScriptError.
	TAbstractScriptEnginesList (see page 15258)	This is class HTTPProd.TAbstractScriptEnginesList.
	TAbstractScriptError (see page 15260)	This is class HTTPProd.TAbstractScriptError.
	TAbstractScriptErrors (see page 15264)	This is class HTTPProd.TAbstractScriptErrors.
	TAbstractScriptProducer (see page 15267)	
	TBasePageProducer (see page 15269)	TBasePageProducer is a base class for objects that produce a string of HTML commands based on an input template.
	TCustomPageProducer (see page 15281)	TCustomPageProducer is the base class for objects that produce a string of HTML commands based on an input template.
	THTMLTableAttributes (see page 15288)	THTMLTableAttributes represents the display attributes of an HTML table tag.
	THTMLTableCellAttributes (see page 15295)	THTMLTableCellAttributes represents the display attributes of an HTML table cell tag.
	THTMLTableElementAttributes (see page 15295)	THTMLTableElementAttributes is the common base class for objects that describe the display attributes of parts of an HTML table.
	THTMLTableHeaderAttributes (see page 15300)	THTMLTableHeaderAttributes represents the display attributes of the header tag in an HTML table.
	THTMLTableRowAttributes (see page 15304)	THTMLTableRowAttributes represents the display attributes of the row tag in an HTML table.
	THTMLTagAttributes (see page 15304)	THTMLTagAttributes is the base class for all objects that represent the attributes of an HTML tag.
	TPageProducer (see page 15309)	TPageProducer produces a string of HTML commands based on an input template.

Constants

Name	Description
HTMLAlign (see page 15341)	This is constant HTTPProd.HTMLAlign.
HTMLVAlign (see page 15341)	This is constant HTTPProd.HTMLVAlign.

Functions

	Name	Description
	ContentFromScriptFile (see page 15333)	This is function HTTPProd.ContentFromScriptFile.
	ContentFromScriptStream (see page 15333)	This is function HTTPProd.ContentFromScriptStream.
	ContentFromStream (see page 15334)	This is function HTTPProd.ContentFromStream.
	ContentFromString (see page 15334)	This is function HTTPProd.ContentFromString.
	FindComponentWebModuleContext (see page 15334)	This is function HTTPProd.FindComponentWebModuleContext.
	GetTagID (see page 15335)	This is function HTTPProd.GetTagID.

Interfaces

	Name	Description
	IDesignerFileManager (see page 15317)	This is class HTTPProd.IDesignerFileManager.
	IGetLocateFileService (see page 15318)	This is class HTTPProd.IGetLocateFileService.
	IGetProducerTemplate (see page 15319)	This is class HTTPProd.IGetProducerTemplate.
	ILocateFileService (see page 15320)	This is class HTTPProd.ILocateFileService.
	IScriptContext (see page 15321)	This is class HTTPProd.IScriptContext.
	IScriptProducer (see page 15323)	This is class HTTPProd.IScriptProducer.
	IWebVariableName (see page 15328)	This is class HTTPProd.IWebVariableName.
	IWebVariablesContainer (see page 15329)	This is class HTTPProd.IWebVariablesContainer.

Structs, Records, Enums

	Name	Description
	THTMLAlign (see page 15336)	This is record HTTPProd.THTMLAlign.
	THTMLVAlign (see page 15336)	This is record HTTPProd.THTMLVAlign.
	TTag (see page 15337)	This is record HTTPProd.TTag.

Types

Name	Description
THandledTagProc (see page 15338)	This is type HTTPProd.THandledTagProc.
THandleTagProc (see page 15338)	This is type HTTPProd.THandleTagProc.
THTMLBgColor (see page 15339)	This is type HTTPProd.THTMLBgColor.
THTMLTagEvent (see page 15339)	This is type HTTPProd.THTMLTagEvent.
TScriptProducerClass (see page 15339)	This is type HTTPProd.TScriptProducerClass.

Variables

Name	Description
DesignerFileManager (see page 15340)	This is variable HTTPProd.DesignerFileManager.
ScriptEnginesList (see page 15340)	This is variable HTTPProd.ScriptEnginesList.

78.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EScriptError (see page 15255)	This is class HTTPProd.EScriptError.
	TAbstractScriptEnginesList (see page 15258)	This is class HTTPProd.TAbstractScriptEnginesList.
	TAbstractScriptError (see page 15260)	This is class HTTPProd.TAbstractScriptError.
	TAbstractScriptErrors (see page 15264)	This is class HTTPProd.TAbstractScriptErrors.
	TAbstractScriptProducer (see page 15267)	
	TBasePageProducer (see page 15269)	TBasePageProducer is a base class for objects that produce a string of HTML commands based on an input template.
	TCustomPageProducer (see page 15281)	TCustomPageProducer is the base class for objects that produce a string of HTML commands based on an input template.
	THTMLTableAttributes (see page 15288)	THTMLTableAttributes represents the display attributes of an HTML table tag.
	THTMLTableCellAttributes (see page 15295)	THTMLTableCellAttributes represents the display attributes of an HTML table cell tag.
	THTMLTableElementAttributes (see page 15295)	THTMLTableElementAttributes is the common base class for objects that describe the display attributes of parts of an HTML table.
	THTMLTableHeaderAttributes (see page 15300)	THTMLTableHeaderAttributes represents the display attributes of the header tag in an HTML table.
	THTMLTableRowAttributes (see page 15304)	THTMLTableRowAttributes represents the display attributes of the row tag in an HTML table.
	THTMLTagAttributes (see page 15304)	THTMLTagAttributes is the base class for all objects that represent the attributes of an HTML tag.
	TPageProducer (see page 15309)	TPageProducer produces a string of HTML commands based on an input template.

78.1.1 EScriptError Class

Class Hierarchy

Pascal

```
EScriptError = class(EWebBrokerException);
```

C++

```
class EScriptError : public EWebBrokerException;
```

File

HTTPProd

Description

This is class HTTPProd.EScriptError.

78.1.1.1 EScriptError Members

The following tables list the members exposed by EScriptError.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 15257)	This is Create, a member of class EScriptError.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 15257)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Content (see page 15258)	This is Content, a member of class EScriptError.
R	Errors (see page 15258)	This is Errors, a member of class EScriptError.
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.

78.1.1.2 EScriptError.Create Constructor

Pascal

```
constructor Create(const AErrors: TAbstractScriptErrors; const AContent: string);
```

C++

```
__fastcall EScriptError(const TAbstractScriptErrors AErrors, const AnsiString AContent);
```

Description

This is Create, a member of class EScriptError.

78.1.1.3 EScriptError.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~EScriptError();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

78.1.1.4 EScriptError Properties

The properties of the EScriptError class are listed here.

Public Properties

	Name	Description
R	Content (see page 15258)	This is Content, a member of class EScriptError.
R	Errors (see page 15258)	This is Errors, a member of class EScriptError.

78.1.1.4.1 EScriptError.Content Property

Pascal

```
property Content: string;
```

C++

```
__property AnsiString Content;
```

Description

This is Content, a member of class EScriptError.

78.1.1.4.2 EScriptError.Errors Property

Pascal

```
property Errors: TAbstractScriptErrors;
```

C++

```
__property TAbstractScriptErrors Errors;
```

Description

This is Errors, a member of class EScriptError.

78.1.2 TAbstractScriptEnginesList Class

Class Hierarchy

Pascal

```
TAbstractScriptEnginesList = class(TObject);
```

C++

```
class TAbstractScriptEnginesList : public TObject;
```

File

HTTPProd

Description

This is class HTTPProd.TAbstractScriptEnginesList.

78.1.2.1 TAbstractScriptEnginesList Members

The following tables list the members exposed by TAbstractScriptEnginesList.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 15259)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
A	FindScriptProducerClass (see page 15260)	This is FindScriptProducerClass, a member of class TAbstractScriptEnginesList.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

78.1.2.2 TAbstractScriptEnginesList.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TAbstractScriptEnginesList();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

78.1.2.3 TAbstractScriptEnginesList Methods

The methods of the TAbstractScriptEnginesList class are listed here.

Public Methods

	Name	Description
	FindScriptProducerClass (see page 15260)	This is FindScriptProducerClass, a member of class TAbstractScriptEnginesList.

78.1.2.3.1 TAbstractScriptEnginesList.FindScriptProducerClass Method

Pascal

```
function FindScriptProducerClass(const ALanguageName: string): TScriptProducerClass; virtual;
abstract;
```

C++

```
virtual __fastcall TScriptProducerClass FindScriptProducerClass(const AnsiString
ALanguageName) = 0;
```

Description

This is FindScriptProducerClass, a member of class TAbstractScriptEnginesList.

78.1.3 TAbstractScriptError Class

Class Hierarchy

Pascal

```
TAbstractScriptError = class(TObject);
```

C++

```
class TAbstractScriptError : public TObject;
```

File

HTTPProd

Description

This is class HTTPProd.TAbstractScriptError.

78.1.3.1 TAbstractScriptError Members

The following tables list the members exposed by TAbstractScriptError.

Protected Methods

	Name	Description
	GetCharPos (see page 15262)	This is GetCharPos, a member of class TAbstractScriptError.
	GetDescription (see page 15262)	This is GetDescription, a member of class TAbstractScriptError.
	GetFileName (see page 15262)	This is GetFileName, a member of class TAbstractScriptError.
	GetLine (see page 15262)	This is GetLine, a member of class TAbstractScriptError.

	GetSourceLine (see page 15263)	This is GetSourceLine, a member of class TAbstractScriptError.
	SetDescription (see page 15263)	This is SetDescription, a member of class TAbstractScriptError.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	CharPos (see page 15263)	This is CharPos, a member of class TAbstractScriptError.
	Description (see page 15263)	This is Description, a member of class TAbstractScriptError.
	FileName (see page 15264)	This is FileName, a member of class TAbstractScriptError.
	Line (see page 15264)	This is Line, a member of class TAbstractScriptError.
	SourceLine (see page 15264)	This is SourceLine, a member of class TAbstractScriptError.

78.1.3.2 TAbstractScriptError Methods

The methods of the TAbstractScriptError class are listed here.

Protected Methods

	Name	Description
	GetCharPos (see page 15262)	This is GetCharPos, a member of class TAbstractScriptError.
	GetDescription (see page 15262)	This is GetDescription, a member of class TAbstractScriptError.
	GetFileName (see page 15262)	This is GetFileName, a member of class TAbstractScriptError.

	GetLine (see page 15262)	This is GetLine, a member of class TAbstractScriptError.
	GetSourceLine (see page 15263)	This is GetSourceLine, a member of class TAbstractScriptError.
	SetDescription (see page 15263)	This is SetDescription, a member of class TAbstractScriptError.

78.1.3.2.1 TAbstractScriptError.GetCharPos Method

Pascal

```
function GetCharPos: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetCharPos() = 0;
```

Description

This is GetCharPos, a member of class TAbstractScriptError.

78.1.3.2.2 TAbstractScriptError.GetDescription Method

Pascal

```
function GetDescription: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetDescription() = 0;
```

Description

This is GetDescription, a member of class TAbstractScriptError.

78.1.3.2.3 TAbstractScriptError.GetFileName Method

Pascal

```
function GetFileName: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetFileName() = 0;
```

Description

This is GetFileName, a member of class TAbstractScriptError.

78.1.3.2.4 TAbstractScriptError.GetLine Method

Pascal

```
function GetLine: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetLine() = 0;
```

Description

This is GetLine, a member of class TAbstractScriptError.

78.1.3.2.5 TAbstractScriptError.GetSourceLine Method

Pascal

```
function GetSourceLine: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetSourceLine() = 0;
```

Description

This is GetSourceLine, a member of class TAbstractScriptError.

78.1.3.2.6 TAbstractScriptError.SetDescription Method

Pascal

```
procedure SetDescription(const AValue: string); virtual; abstract;
```

C++

```
virtual __fastcall SetDescription(const AnsiString AValue) = 0;
```

Description

This is SetDescription, a member of class TAbstractScriptError.

78.1.3.3 TAbstractScriptError Properties

The properties of the TAbstractScriptError class are listed here.

Public Properties

	Name	Description
R	CharPos (see page 15263)	This is CharPos, a member of class TAbstractScriptError.
	Description (see page 15263)	This is Description, a member of class TAbstractScriptError.
R	FileName (see page 15264)	This is FileName, a member of class TAbstractScriptError.
R	Line (see page 15264)	This is Line, a member of class TAbstractScriptError.
R	SourceLine (see page 15264)	This is SourceLine, a member of class TAbstractScriptError.

78.1.3.3.1 TAbstractScriptError.CharPos Property

Pascal

```
property CharPos: Integer;
```

C++

```
__property int CharPos;
```

Description

This is CharPos, a member of class TAbstractScriptError.

78.1.3.3.2 TAbstractScriptError.Description Property

Pascal

```
property Description: string;
```

C++

```
__property AnsiString Description;
```

Description

This is Description, a member of class TAbstractScriptError.

78.1.3.3.3 TAbstractScriptError.FileName Property

Pascal

```
property FileName: string;
```

C++

```
__property AnsiString FileName;
```

Description

This is FileName, a member of class TAbstractScriptError.

78.1.3.3.4 TAbstractScriptError.Line Property

Pascal

```
property Line: Integer;
```

C++

```
__property int Line;
```

Description

This is Line, a member of class TAbstractScriptError.

78.1.3.3.5 TAbstractScriptError.SourceLine Property

Pascal

```
property SourceLine: string;
```

C++

```
__property AnsiString SourceLine;
```

Description

This is SourceLine, a member of class TAbstractScriptError.

78.1.4 TAbstractScriptErrors Class

Class Hierarchy

Pascal

```
TAbstractScriptErrors = class(TObject);
```

C++

```
class TAbstractScriptErrors : public TObject;
```

File

HTTPProd

Description

This is class HTTPProd.TAbstractScriptErrors.

78.1.4.1 TAbstractScriptErrors Members

The following tables list the members exposed by TAbstractScriptErrors.

Protected Methods

	Name	Description
	GetError (see page 15266)	This is GetError, a member of class TAbstractScriptErrors.
	GetErrorCount (see page 15266)	This is GetErrorCount, a member of class TAbstractScriptErrors.

Public Methods

	Name	Description
	Add (see page 15266)	This is Add, a member of class TAbstractScriptErrors.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Count (see page 15267)	This is Count, a member of class TAbstractScriptErrors.

	Errors (see page 15267)	This is Errors, a member of class TAbstractScriptErrors.

78.1.4.2 TAbstractScriptErrors Methods

The methods of the TAbstractScriptErrors class are listed here.

Protected Methods

	Name	Description
	GetError (see page 15266)	This is GetError, a member of class TAbstractScriptErrors.
	GetErrorCount (see page 15266)	This is GetErrorCount, a member of class TAbstractScriptErrors.

Public Methods

	Name	Description
	Add (see page 15266)	This is Add, a member of class TAbstractScriptErrors.

78.1.4.2.1 TAbstractScriptErrors.Add Method

Pascal

```
procedure Add(const AError: TAbstractScriptError); virtual; abstract;
```

C++

```
virtual __fastcall Add(const TAbstractScriptError AError) = 0;
```

Description

This is Add, a member of class TAbstractScriptErrors.

78.1.4.2.2 TAbstractScriptErrors.GetError Method

Pascal

```
function GetError(I: Integer): TAbstractScriptError; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractScriptError GetError(int I) = 0;
```

Description

This is GetError, a member of class TAbstractScriptErrors.

78.1.4.2.3 TAbstractScriptErrors.GetErrorCount Method

Pascal

```
function GetErrorCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetErrorCount() = 0;
```

Description

This is GetErrorCount, a member of class TAbstractScriptErrors.

78.1.4.3 TAbstractScriptErrors Properties

The properties of the TAbstractScriptErrors class are listed here.

Public Properties

	Name	Description
R	Count (see page 15267)	This is Count, a member of class TAbstractScriptErrors.
R	Errors (see page 15267)	This is Errors, a member of class TAbstractScriptErrors.

78.1.4.3.1 TAbstractScriptErrors.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TAbstractScriptErrors.

78.1.4.3.2 TAbstractScriptErrors.Errors Property

Pascal

```
property Errors [I: Integer]: TAbstractScriptError;
```

C++

```
__property TAbstractScriptError Errors[int I];
```

Description

This is Errors, a member of class TAbstractScriptErrors.

78.1.5 TAbstractScriptProducer Class

Class Hierarchy

Pascal

```
TAbstractScriptProducer = class(TInterfacedObject);
```

C++

```
class TAbstractScriptProducer : public TInterfacedObject;
```

File

HTTPProd

78.1.5.1 TAbstractScriptProducer Members

The following tables list the members exposed by TAbstractScriptProducer.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 15269)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

78.1.5.2 TAbstractScriptProducer.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create(AWebModuleContext: TWebModuleContext; AStripParamQuotes: Boolean;
AHandleTag: THandleTagProc; const AScriptEngine: string; ALocateFileService:
ILocateFileService); virtual;
```

C++

```
virtual __fastcall TAbstractScriptProducer(TWebModuleContext AWebModuleContext, Boolean
AStripParamQuotes, THandleTagProc AHandleTag, const AnsiString AScriptEngine,
ILocateFileService ALocateFileService);
```

Description

Do not create instances of Create. Instead, construct descendants of Create using the new keyword, passing any arguments for the descendant object's constructor.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

Destroy (see page 22287)

new

78.1.6 TBasePageProducer Class

TBasePageProducer is a base class for objects that produce a string of HTML commands based on an input template.

Class Hierarchy

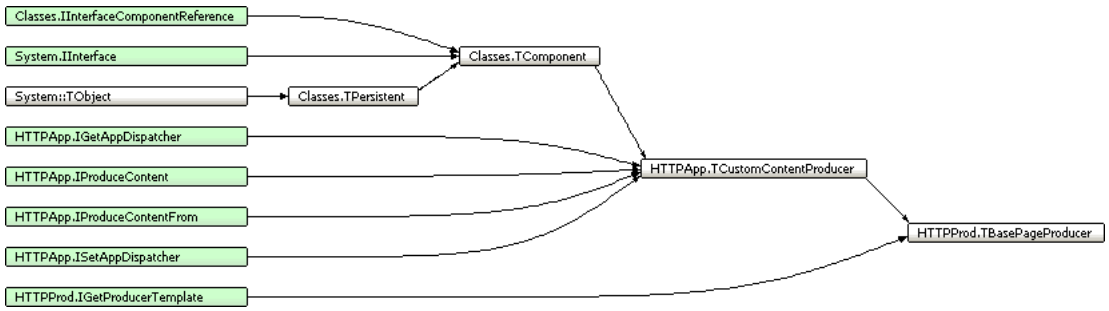

Pascal

```
TBasePageProducer = class(TCustomContentProducer, IGetProducerTemplate);
```

C++

```
class TBasePageProducer : public TCustomContentProducer, public IGetProducerTemplate;
```

File

HTTPProd

Description

Do not create instances of TBasePageProducer. Instead, use TBasePageProducer as a base class when defining a page producer. TBasePageProducer has two descendants,

TCustomPageProducer (see page 15281), which is the base class for page producers that produce content based primarily on the content of HTTP messages and an OnHTML event handler. Examples include TPageProducer (see page 15309) and TDataSetPageProducer.

TComponentsPageProducer, which is the base class for page producers that assemble content from components that generate HTML. Examples include TAdapterPageProducer.

A page producer is an object that converts an HTML template into a string of HTML commands which can be interpreted by a client application such as a Web browser. HTML templates include HTML commands and HTML-transparent tags that are replaced with customized content by the page producer.

See Also

TDSTableProducer (see page 10290)

TCustomPageProducer (see page 15281)

TPageProducer (see page 15309)

TWebResponse (see page 15191)

TDataSetPageProducer (see page 11807)

TPageItemsProducer (see page 18383)

TInetXPageProducer (see page 17145)

TAdapterPageProducer (see page 19802)

78.1.6.1 TBasePageProducer Members

The following tables list the members exposed by TBasePageProducer.

Protected Events

	Name	Description
	OnHTMLTag (see page 15280)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.

	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoTagEvent (see page 15276)	This is DoTagEvent, a member of class TBasePageProducer.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetProducerTemplateFileName (see page 15276)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.
	GetProducerTemplateStream (see page 15276)	This is GetProducerTemplateStream, a member of class TBasePageProducer.
V	GetScriptEngine (see page 15277)	This is GetScriptEngine, a member of class TBasePageProducer.
	GetTagID (see page 15277)	This is GetTagID, a member of class TBasePageProducer.
V	GetTemplateFileName (see page 15277)	This is GetTemplateFileName, a member of class TBasePageProducer.
V	GetTemplateStream (see page 15277)	This is GetTemplateStream, a member of class TBasePageProducer.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleScriptTag (see page 15277)	This is HandleScriptTag, a member of class TBasePageProducer.
V	HandleTag (see page 15278)	This is HandleTag, a member of class TBasePageProducer.
	ImplHandleTag (see page 15278)	This is ImplHandleTag, a member of class TBasePageProducer.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 15108)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ServerScriptFromStream (see page 15278)	This is ServerScriptFromStream, a member of class TBasePageProducer.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UseScriptEngine (see page 15278)	This is UseScriptEngine, a member of class TBasePageProducer.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 15274)	Returns the content produced by the content producer.
	ContentFromStream (see page 15275)	Returns the content produced in response to input read from a stream.
	ContentFromString (see page 15275)	Returns the content produced in response to an input string.
	Create (see page 15274)	Creates an instance of TBasePageProducer (see page 15269).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.

	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ScriptEngine (see page 15279)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 15279)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebModuleContext (see page 15279)	Provides access to the Web module context object.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

78.1.6.2 TBasePageProducer.Create Constructor

Creates an instance of TBasePageProducer (see page 15269).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBasePageProducer(TComponent * AOwner);
```

Description

Call Create to create and initialize an instance of TBasePageProducer (see page 15269).

AOwner is a component, typically a Web module, that is responsible for freeing the page producer.

See Also

Owner (see page 2807)

78.1.6.3 TBasePageProducer Methods

The methods of the TBasePageProducer class are listed here.

Protected Methods

	Name	Description
	DoTagEvent (see page 15276)	This is DoTagEvent, a member of class TBasePageProducer.
	GetProducerTemplateFileName (see page 15276)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.
	GetProducerTemplateStream (see page 15276)	This is GetProducerTemplateStream, a member of class TBasePageProducer.
V	GetScriptEngine (see page 15277)	This is GetScriptEngine, a member of class TBasePageProducer.
	GetTagID (see page 15277)	This is GetTagID, a member of class TBasePageProducer.
V	GetTemplateFileName (see page 15277)	This is GetTemplateFileName, a member of class TBasePageProducer.
V	GetTemplateStream (see page 15277)	This is GetTemplateStream, a member of class TBasePageProducer.
V	HandleScriptTag (see page 15277)	This is HandleScriptTag, a member of class TBasePageProducer.
V	HandleTag (see page 15278)	This is HandleTag, a member of class TBasePageProducer.
	ImplHandleTag (see page 15278)	This is ImplHandleTag, a member of class TBasePageProducer.
	ServerScriptFromStream (see page 15278)	This is ServerScriptFromStream, a member of class TBasePageProducer.
V	UseScriptEngine (see page 15278)	This is UseScriptEngine, a member of class TBasePageProducer.

Public Methods

	Name	Description
V	Content (see page 15274)	Returns the content produced by the content producer.
V	ContentFromStream (see page 15275)	Returns the content produced in response to input read from a stream.
V	ContentFromString (see page 15275)	Returns the content produced in response to an input string.

78.1.6.3.1 TBasePageProducer.Content Method

Returns the content produced by the content producer.

Pascal

```
function Content: string; override;
```

C++

```
virtual __fastcall AnsiString Content();
```

Description

Call Content to retrieve the content for using in an HTTP response message. TBasePageProducer (see page 15269) attempts to locate a default template and uses that to generate the resulting content.

See Also

ContentFromStream (see page 15275)

ContentFromString (see page 15275)

Content (see page 15207)

Specifying the HTML Template

78.1.6.3.2 TBasePageProducer.ContentFromStream Method

Returns the content produced in response to input read from a stream.

Pascal

```
function ContentFromStream(Stream: TStream): string; override;
```

C++

```
virtual __fastcall AnsiString ContentFromStream(TStream Stream);
```

Description

Call ContentFromStream to retrieve the content produced by the content producer based on the information provided by the Stream parameter.

When the ScriptEngine (see page 15279) property is assigned, the page producer generates its content using a special script-producer component. When ScriptEngine (see page 15279) is not assigned, the page producer uses only the OnHTMLTag (see page 15280) event, plus any other built-in translations provided by a descendant class.

See Also

Content (see page 15274)

ContentFromString (see page 15275)

Content (see page 15207)

ScriptEngine (see page 15279)

OnHTMLTag (see page 15280)

Specifying the HTML Template

78.1.6.3.3 TBasePageProducer.ContentFromString Method

Returns the content produced in response to an input string.

Pascal

```
function ContentFromString(const S: string): string; override;
```

C++

```
virtual __fastcall AnsiString ContentFromString(const AnsiString S);
```

Description

Call ContentFromString to retrieve the content produced by the content producer based on the information provided by the S parameter.

When the ScriptEngine (see page 15279) property is assigned, the page producer generates its content using a special script-producer component. When ScriptEngine (see page 15279) is not assigned, the page producer uses only the OnHTMLTag (see page 15280) event, plus any other built-in translations provided by a descendant class.

See Also

Content (see page 15274)

ContentFromStream (see page 15275)

Content (see page 15207)

Chaining Page Producers Together

Specifying the HTML Template

78.1.6.3.4 TBasePageProducer.DoTagEvent Method

Pascal

```
procedure DoTagEvent(Tag: TTag; const TagString: string; TagParams: TStrings; var ReplaceText:
string); dynamic;
```

C++

```
__fastcall DoTagEvent(TTag Tag, const AnsiString TagString, TStrings TagParams, AnsiString
ReplaceText);
```

Description

This is DoTagEvent, a member of class TBasePageProducer.

78.1.6.3.5 TBasePageProducer.GetProducerTemplateFileName Method

Pascal

```
function GetProducerTemplateFileName: string;
```

C++

```
__fastcall AnsiString GetProducerTemplateFileName();
```

Description

This is GetProducerTemplateFileName, a member of class TBasePageProducer.

78.1.6.3.6 TBasePageProducer.GetProducerTemplateStream Method

Pascal

```
function GetProducerTemplateStream(out AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream GetProducerTemplateStream(Boolean AOwned);
```

Description

This is GetProducerTemplateStream, a member of class TBasePageProducer.

78.1.6.3.7 TBasePageProducer.GetScriptEngine Method

Pascal

```
function GetScriptEngine: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetScriptEngine();
```

Description

This is GetScriptEngine, a member of class TBasePageProducer.

78.1.6.3.8 TBasePageProducer.GetTagID Method

Pascal

```
function GetTagID(const TagString: string): TTag;
```

C++

```
__fastcall TTag GetTagID(const AnsiString TagString);
```

Description

This is GetTagID, a member of class TBasePageProducer.

78.1.6.3.9 TBasePageProducer.GetTemplateFileName Method

Pascal

```
function GetTemplateFileName: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetTemplateFileName();
```

Description

This is GetTemplateFileName, a member of class TBasePageProducer.

78.1.6.3.10 TBasePageProducer.GetTemplateStream Method

Pascal

```
function GetTemplateStream(out AOwned: Boolean): TStream; virtual;
```

C++

```
virtual __fastcall TStream GetTemplateStream(Boolean AOwned);
```

Description

This is GetTemplateStream, a member of class TBasePageProducer.

78.1.6.3.11 TBasePageProducer.HandleScriptTag Method

Pascal

```
function HandleScriptTag(const TagString: string; TagParams: TStrings; var ReplaceString:
string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean HandleScriptTag(const AnsiString TagString, TStrings TagParams,
AnsiString ReplaceString);
```

Description

This is HandleScriptTag, a member of class TBasePageProducer.

78.1.6.3.12 TBasePageProducer.HandleTag Method

Pascal

```
function HandleTag(const TagString: string; TagParams: TStrings): string; virtual;
```

C++

```
virtual __fastcall AnsiString HandleTag(const AnsiString TagString, TStrings TagParams);
```

Description

This is HandleTag, a member of class TBasePageProducer.

78.1.6.3.13 TBasePageProducer.ImplHandleTag Method

Pascal

```
function ImplHandleTag(const TagString: string; TagParams: TStrings): string;
```

C++

```
__fastcall AnsiString ImplHandleTag(const AnsiString TagString, TStrings TagParams);
```

Description

This is ImplHandleTag, a member of class TBasePageProducer.

78.1.6.3.14 TBasePageProducer.ServerScriptFromStream Method

Pascal

```
function ServerScriptFromStream(Stream: TStream): string;
```

C++

```
__fastcall AnsiString ServerScriptFromStream(TStream Stream);
```

Description

This is ServerScriptFromStream, a member of class TBasePageProducer.

78.1.6.3.15 TBasePageProducer.UseScriptEngine Method

Pascal

```
function UseScriptEngine: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean UseScriptEngine();
```

Description

This is UseScriptEngine, a member of class TBasePageProducer.

78.1.6.4 TBasePageProducer Properties

The properties of the TBasePageProducer class are listed here.

Public Properties

	Name	Description
	ScriptEngine (see page 15279)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 15279)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.
R	WebModuleContext (see page 15279)	Provides access to the Web module context object.

78.1.6.4.1 TBasePageProducer.ScriptEngine Property

Names the script engine that generates or interprets script embedded in the documents that this producer creates.

Pascal

```
property ScriptEngine: string;
```

C++

```
__property AnsiString ScriptEngine;
```

Description

ScriptEngine indicates the scripting language that the producer uses in its generated documents. For example, if the producer generates code that includes jscript, the value of ScriptEngine is 'JSCRIPT'.

When ScriptEngine is defined, the page producer generates its content using a special script-producer component. When ScriptEngine is not defined, the page producer uses only the OnHTMLTag (see page 15280) event, plus any other built-in translations provided by a descendant class.

See Also

OnHTMLTag (see page 15280)

78.1.6.4.2 TBasePageProducer.StripParamQuotes Property

Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.

Pascal

```
property StripParamQuotes: Boolean;
```

C++

```
__property Boolean StripParamQuotes;
```

Description

Set StripParamQuotes to indicate whether the page producer should strip quotation marks from the parameters of HTML-transparent tags before handling the tags. By setting StripParamQuotes to true, the page producer compensates for HTML editors that automatically insert quotation marks around the values of tag options.

78.1.6.4.3 TBasePageProducer.WebModuleContext Property

Provides access to the Web module context object.

Pascal

```
property WebModuleContext: TWebModuleContext;
```

C++

```
__property TWebModuleContext WebModuleContext;
```

Description

Use WebModuleContext to get access to the TWebModuleContext object. The TWebModuleContext object maintains a list of named variables that the producer uses when translating HTML-transparent tags.

78.1.6.5 TBasePageProducer Events

The events of the TBasePageProducer class are listed here.

Protected Events

	Name	Description
	OnHTMLTag (see page 15280)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

78.1.6.5.1 TBasePageProducer.OnHTMLTag Event

Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Pascal

```
property OnHTMLTag: THTMLTagEvent;
```

C++

```
__property THTMLTagEvent OnHTMLTag;
```

Description

Write an OnHTMLTag event handler to convert an HTML-transparent tag into the corresponding HTML commands that can be interpreted by a Web browser. If an OnHTMLTag event is not supplied, the HTML-transparent tags in the HTML template will be replaced by empty strings.

Note: The sequence of HTML that the event handler produces for an HTML-transparent tag can contain HTML-transparent tags for another page producer to convert. For example, one page producer might have a template that represents the format of the final HTTP response. It could read the HTTP request and for each section in the request assemble a set of HTML-transparent tags with parameters based on the request. Another page producer might take the content of the first page producer, and interpret those parameters.

See Also

Names (see page 3075)

Values (see page 3086)

Customizing the InternetExpress Page Producer Template

Chaining Page Producers Together

Converting HTML-transparent Tags

78.1.7 TCustomPageProducer Class

TCustomPageProducer is the base class for objects that produce a string of HTML commands based on an input template.

Class Hierarchy

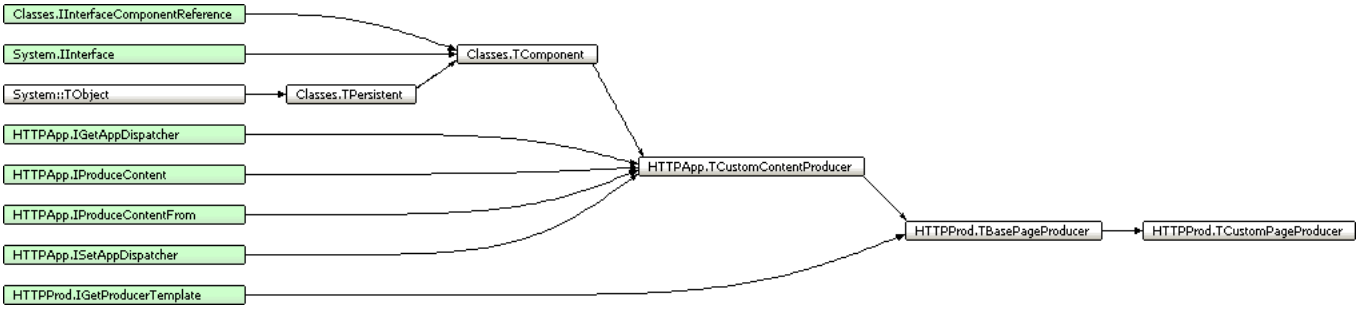

Pascal

```
TCustomPageProducer = class(TBasePageProducer);
```

C++

```
class TCustomPageProducer : public TBasePageProducer;
```

File

HTTPProd

Description

Use TCustomPageProducer as a base class when defining an object that converts an HTML template into a string of HTML commands which can be interpreted by a client application such as a Web browser. HTML templates include HTML commands and HTML-transparent tags that are replaced with customized content by the page producer.

See Also

TDSTableProducer (see page 10290)

TPageProducer (see page 15309)

78.1.7.1 TCustomPageProducer Members

The following tables list the members exposed by TCustomPageProducer.

Protected Events

	Name	Description
	OnHTMLTag (see page 15280)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.

	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoTagEvent (see page 15276)	This is DoTagEvent, a member of class TBasePageProducer.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetProducerTemplateFileName (see page 15276)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.
	GetProducerTemplateStream (see page 15276)	This is GetProducerTemplateStream, a member of class TBasePageProducer.
V	GetScriptEngine (see page 15277)	This is GetScriptEngine, a member of class TBasePageProducer.
	GetTagID (see page 15277)	This is GetTagID, a member of class TBasePageProducer.
V	GetTemplateFileName (see page 15286)	This is GetTemplateFileName, a member of class TCustomPageProducer.
V	GetTemplateStream (see page 15286)	This is GetTemplateStream, a member of class TCustomPageProducer.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleScriptTag (see page 15277)	This is HandleScriptTag, a member of class TBasePageProducer.
V	HandleTag (see page 15286)	This is HandleTag, a member of class TCustomPageProducer.
	ImplHandleTag (see page 15278)	This is ImplHandleTag, a member of class TBasePageProducer.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 15108)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ServerScriptFromStream (see page 15278)	This is ServerScriptFromStream, a member of class TBasePageProducer.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UseScriptEngine (see page 15278)	This is UseScriptEngine, a member of class TBasePageProducer.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 15106)	Returns the content produced by the content producer.
	Content (see page 15274)	Returns the content produced by the content producer.
	ContentFromStream (see page 15107)	Returns the content produced in response to input read from a stream.
	ContentFromStream (see page 15275)	Returns the content produced in response to input read from a stream.
	ContentFromString (see page 15107)	Returns the content produced in response to an input string.
	ContentFromString (see page 15275)	Returns the content produced in response to an input string.
	Create (see page 15285)	Creates an instance of TCustomPageProducer (see page 15281).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 15285)	Destroys an instance of TCustomPageProducer (see page 15281).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.

	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProducerTemplateFileName (see page 15320)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.
	GetProducerTemplateStream (see page 15320)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.
	HTMLDoc (see page 15287)	Specifies the HTML template used to generate the value returned by the Content (see page 15274) method.
	HTMLFile (see page 15287)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content (see page 15274) method.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ScriptEngine (see page 15279)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 15279)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebModuleContext (see page 15279)	Provides access to the Web module context object.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

78.1.7.2 TCustomPageProducer.Create Constructor

Creates an instance of TCustomPageProducer (see page 15281).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomPageProducer(TComponent * AOwner);
```

Description

Call Create to create and initialize an instance of TCustomPageProducer (see page 15281).

After calling the inherited constructor, Create allocates the TStrings object that implements the HTMLDoc (see page 15287) property.

See Also

HTMLDoc (see page 15287)

Create (see page 2766)

78.1.7.3 TCustomPageProducer.Destroy Destructor

Destroys an instance of TCustomPageProducer (see page 15281).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomPageProducer();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the page producer reference is not nil, and only then calls Destroy.

Destroy frees the TStrings object implements the HTMLDoc (see page 15287) property before calling the inherited destructor.

See Also

Free (see page 22300)

HTMLDoc (see page 15287)

78.1.7.4 TCustomPageProducer Methods

The methods of the TCustomPageProducer class are listed here.

Protected Methods

	Name	Description
	GetTemplateFileName (see page 15286)	This is GetTemplateFileName, a member of class TCustomPageProducer.
	GetTemplateStream (see page 15286)	This is GetTemplateStream, a member of class TCustomPageProducer.
	HandleTag (see page 15286)	This is HandleTag, a member of class TCustomPageProducer.

78.1.7.4.1 TCustomPageProducer.GetTemplateFileName Method

Pascal

```
function GetTemplateFileName: string; override;
```

C++

```
virtual __fastcall AnsiString GetTemplateFileName();
```

Description

This is GetTemplateFileName, a member of class TCustomPageProducer.

78.1.7.4.2 TCustomPageProducer.GetTemplateStream Method

Pascal

```
function GetTemplateStream(out AOwned: Boolean): TStream; override;
```

C++

```
virtual __fastcall TStream GetTemplateStream(Boolean AOwned);
```

Description

This is GetTemplateStream, a member of class TCustomPageProducer.

78.1.7.4.3 TCustomPageProducer.HandleTag Method

Pascal

```
function HandleTag(const TagString: string; TagParams: TStrings): string; override;
```

C++

```
virtual __fastcall AnsiString HandleTag(const AnsiString TagString, TStrings TagParams);
```

Description

This is HandleTag, a member of class TCustomPageProducer.

78.1.7.5 TCustomPageProducer Properties

The properties of the TCustomPageProducer class are listed here.

Public Properties

	Name	Description
	HTMLDoc (see page 15287)	Specifies the HTML template used to generate the value returned by the Content (see page 15274) method.
	HTMLFile (see page 15287)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content (see page 15274) method.

78.1.7.5.1 TCustomPageProducer.HTMLDoc Property

Specifies the HTML template used to generate the value returned by the Content (see page 15274) method.

Pascal

```
property HTMLDoc: TStrings;
```

C++

```
__property TStrings HTMLDoc;
```

Description

Set HTMLDoc to a set of strings that make up the HTML template. Each string is a sequence of one or more HTML commands or HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLDoc clears the HTMLFile (see page 15287) property.

See Also

Content (see page 15274)

HTMLFile (see page 15287)

Specifying the HTML Template

78.1.7.5.2 TCustomPageProducer.HTMLFile Property

Specifies the name of a file that contains the HTML template used to generate the value returned by the Content (see page 15274) method.

Pascal

```
property HTMLFile: TFileName;
```

C++

```
__property TFileName HTMLFile;
```

Description

Set HTMLFile to the name of the file that contains an HTML template. The HTML template is a sequence of HTML commands and HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where

there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLFile clears the HTMLDoc (see page 15287) property.

Note: When distributing applications that contain page producers which use the HTMLFile property, be sure to distribute the file along with the application. Also, be sure that the value of HTMLFile does not include a hard-coded path name that will not be available on the user's disk.

See Also

Content (see page 15274)

HTMLDoc (see page 15287)

Specifying the HTML Template

78.1.8 THTMLTableAttributes Class

THTMLTableAttributes represents the display attributes of an HTML table tag.

Class Hierarchy

Pascal

```
THTMLTableAttributes = class(THTMLTagAttributes);
```

C++

```
class THTMLTableAttributes : public THTMLTagAttributes;
```

File

HTTPProd

Description

Use THTMLTableAttributes to describe the desired appearance of an HTML table. The properties of the THTMLTableAttributes object correspond to the options that can be added to the HTML table tag. When the attributes of all the parts of the HTML table have been specified, an HTML producer object creates the HTML commands to specify the table as described.

See Also

TCustomContentProducer (see page 15102)

TDSTableProducer (see page 10290)

THTMLTableCellAttributes (see page 15295)

THTMLTableColumn (see page 10318)

THTMLTableHeaderAttributes (see page 15300)

THTMLTableRowAttributes (see page 15304)

Specifying the Table Attributes

78.1.8.1 THTMLTableAttributes Members

The following tables list the members exposed by THTMLTableAttributes.

Protected Methods

	Name	Description
	AssignTo (see page 15291)	Copies the properties of a THTMLTableAttributes (see page 15288) object to another THTMLTableAttributes (see page 15288) object.
	Changed (see page 15307)	Generates an OnChange (see page 15308) event.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Events

	Name	Description
	OnChange (see page 15308)	Occurs immediately after a tag attributes property changes.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 15290)	Creates an instance of THTMLTableAttributes (see page 15288).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RestoreDefaults (see page 15291)	Changes all properties of the table attributes object back to their initial settings.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Producer (see page 15308)	Indicates the content producer that uses the THTMLTagAttributes (see page 15304) object to represent the options for one of the HTML tags in the content it produces.

Published Properties

	Name	Description
	Align (see page 15292)	Specifies the alignment of the table within the HTML document.
	BgColor (see page 15293)	Specifies the background color of the HTML table.
	Border (see page 15293)	Specifies the thickness of the border drawn around the HTML table.
	CellPadding (see page 15294)	Specifies the amount of space to leave around the contents of each cell in the HTML table.
	CellSpacing (see page 15294)	Specifies the amount of space to leave between cells in the HTML table.
	Custom (see page 15307)	Represents a tag option and its value in an HTML tag.
	Width (see page 15294)	Specifies the width of the entire HTML table.

78.1.8.2 THTMLTableAttributes.Create Constructor

Creates an instance of THTMLTableAttributes (see page 15288).

Pascal

```
constructor Create(Producer: TCustomContentProducer);
```

C++

```
__fastcall THTMLTableAttributes(TCustomContentProducer Producer);
```

Description

Call Create to create and initialize a THTMLTableAttributes (see page 15288) object. The Producer (see page 15308) parameter specifies the HTML producer that is using the THTMLTableAttributes (see page 15288) object to specify the attributes of an HTML table that makes up part of its output.

After calling the inherited constructor, Create initializes

Width (see page 15294) to a value of 100.

Border (see page 15293) to a value of -1.

CellPadding (see page 15294) to a value of -1.

CellSpacing (see page 15294) to a value of -1.

See Also

Border (see page 15293)

CellPadding (see page 15294)

CellSpacing (see page 15294)

Create (see page 15306)

Width (see page 15294)

78.1.8.3 THTMLTableAttributes Methods

The methods of the THTMLTableAttributes class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 15291)	Copies the properties of a THTMLTableAttributes (see page 15288) object to another THTMLTableAttributes (see page 15288) object.

Public Methods

	Name	Description
	RestoreDefaults (see page 15291)	Changes all properties of the table attributes object back to their initial settings.

78.1.8.3.1 THTMLTableAttributes.AssignTo Method

Copies the properties of a THTMLTableAttributes (see page 15288) object to another THTMLTableAttributes (see page 15288) object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Applications cannot call the protected AssignTo method. To assign the properties of a THTMLTableAttributes (see page 15288) object to another object, use the Assign method of the destination object.

When the Assign method of another object specifies a THTMLTableAttributes (see page 15288) object as the source of the assignment, the AssignTo method of the THTMLTableAttributes (see page 15288) object is called. AssignTo checks whether the Dest parameter is another THTMLTableAttributes (see page 15288) object, and if so, copies the values of the Align (see page 15292), Border (see page 15293), CellSpacing (see page 15294), CellPadding (see page 15294), and Width (see page 15294) properties. If Dest is any other class of object, the inherited method is called.

Note: AssignTo does not copy the Custom (see page 15307) property.

See Also

Align (see page 15292)

Border (see page 15293)

CellPadding (see page 15294)

CellSpacing (see page 15294)

Assign (see page 2930)

Width (see page 15294)

78.1.8.3.2 THTMLTableAttributes.RestoreDefaults Method

Changes all properties of the table attributes object back to their initial settings.

Pascal

```
procedure RestoreDefaults; override;
```

C++

```
virtual __fastcall RestoreDefaults();
```

Description

Call RestoreDefaults to undo any changes that were made to the table attributes either programmatically, or using the response editor at design time. RestoreAttributes sets

Align (see page 15292) to haDefault. (Delphi) or haDefault; (C++)

Border (see page 15293), CellPadding (see page 15294), and CellSpacing (see page 15294) to –1. (Delphi) or -1; (C++)

Custom (see page 15307) to an empty string.

Width (see page 15294) to 100. (Delphi) or 100; (C++)

See Also

Align (see page 15292)

Border (see page 15293)

CellPadding (see page 15294)

CellSpacing (see page 15294)

Custom (see page 15307)

Width (see page 15294)

78.1.8.4 THTMLTableAttributes Properties

The properties of the THTMLTableAttributes class are listed here.

Published Properties

	Name	Description
	Align (see page 15292)	Specifies the alignment of the table within the HTML document.
	BgColor (see page 15293)	Specifies the background color of the HTML table.
	Border (see page 15293)	Specifies the thickness of the border drawn around the HTML table.
	CellPadding (see page 15294)	Specifies the amount of space to leave around the contents of each cell in the HTML table.
	CellSpacing (see page 15294)	Specifies the amount of space to leave between cells in the HTML table.
	Width (see page 15294)	Specifies the width of the entire HTML table.

78.1.8.4.1 THTMLTableAttributes.Align Property

Specifies the alignment of the table within the HTML document.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Set Align to indicate the desired alignment of the table within the HTML document, or within the cell of a surrounding HTML table object. Align indicates the desired value of the Align option of the HTML table tag.

The default value of Align is haDefault, which means the web browser that reads the HTML document positions the HTML table where it wants.

See Also

CellPadding (see page 15294)

Width (see page 15294)

78.1.8.4.2 THTMLTableAttributes.BgColor Property

Specifies the background color of the HTML table.

Pascal

```
property BgColor: THTMLBgColor;
```

C++

```
__property THTMLBgColor BgColor;
```

Description

Use BgColor to indicate the desired background color of the HTML table. The value of BgColor is the name of color. This can be one of the following strings:

Aqua	Black	Blue	Fuschia
Gray	Green	Lime	Maroon
Navy	Olive	Purple	Red
Silver	Teal	White	Yellow

Use the BgColor property of the HTML attributes objects for the pieces of the table for more involved color effects.

See Also

BgColor (see page 10325)

BgColor (see page 15299)

78.1.8.4.3 THTMLTableAttributes.Border Property

Specifies the thickness of the border drawn around the HTML table.

Pascal

```
property Border: Integer;
```

C++

```
__property int Border;
```

Description

Set Border to specify whether a border should be drawn around the table, and if so, how thick it should be. The thickness specified by any particular value of border depends on the web browser that views the HTML table. For any browser, however, larger numbers indicate thicker borders. A value of -1 indicates that no border should be drawn around the table.

See Also

CellSpacing (see page 15294)

78.1.8.4.4 THTMLTableAttributes.CellPadding Property

Specifies the amount of space to leave around the contents of each cell in the HTML table.

Pascal

```
property CellPadding: Integer;
```

C++

```
__property int CellPadding;
```

Description

Set CellPadding to specify how many pixels of whitespace should appear between a cell's contents and the edges of the cell. A value of -1 indicates that the web browser should decide how to pad the cells of the HTML table.

See Also

CellSpacing (see page 15294)

78.1.8.4.5 THTMLTableAttributes.CellSpacing Property

Specifies the amount of space to leave between cells in the HTML table.

Pascal

```
property CellSpacing: Integer;
```

C++

```
__property int CellSpacing;
```

Description

Set CellSpacing to specify how many pixels of whitespace should appear between the cells in the table. A value of -1 indicates that the web browser should decide how to separate the cells of the HTML table.

Note: CellSpacing refers to the whitespace between cells in an HTML table, not the thickness of the lines drawn to separate rows and columns.

See Also

Border (see page 15293)

CellPadding (see page 15294)

78.1.8.4.6 THTMLTableAttributes.Width Property

Specifies the width of the entire HTML table.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Set Width to indicate the desired width of the HTML table as a percentage of the width of the web browser window. If the value of Width is too small for the contents of the table, the web browser will ignore this property.

The default value of Width is 100, meaning the table will span the entire browser window. In this case, the value of the Align (

see page 15292) property is meaningless.

See Also

Align (see page 15292)

CellSpacing (see page 15294)

78.1.9 THTMLTableCellAttributes Class

THTMLTableCellAttributes represents the display attributes of an HTML table cell tag.

Class Hierarchy

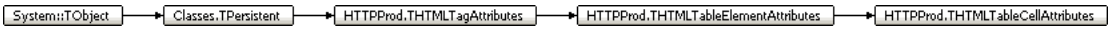

Pascal

```
THTMLTableCellAttributes = class(THTMLTableElementAttributes);
```

C++

```
class THTMLTableCellAttributes : public THTMLTableElementAttributes;
```

File

HTTPProd

Description

Use THTMLTableCellAttributes to describe the desired appearance of a cell in an HTML table. The properties of the THTMLTableCellAttributes object correspond to the options that can be added to the HTML table data tag (TD). When the attributes of all the parts of the HTML table have been specified, an HTML producer object creates the HTML commands to specify the table as described.

See Also

TCustomContentProducer (see page 15102)

THTMLTableAttributes (see page 15288)

THTMLTableColumn (see page 10318)

THTMLTableHeaderAttributes (see page 15300)

THTMLTableRowAttributes (see page 15304)

78.1.10 THTMLTableElementAttributes Class

THTMLTableElementAttributes is the common base class for objects that describe the display attributes of parts of an HTML table.

Class Hierarchy

Pascal

```
THTMLTableElementAttributes = class(THTMLTagAttributes);
```

C++

```
class THTMLTableElementAttributes : public THTMLTagAttributes;
```

File

HTTPProd

Description

Use the properties of THTMLTableElementAttributes to describe the desired appearance of a piece of an HTML table. The properties of THTMLTableElementAttributes correspond to the common options that can be added to the tags for the parts of the HTML table. When the attributes of all the parts of the HTML table have been specified, an HTML producer object creates the HTML commands to specify the table as described.

See Also

TCustomContentProducer (see page 15102)

TDSTableProducer (see page 10290)

THTMLTableCellAttributes (see page 15295)

THTMLTableHeaderAttributes (see page 15300)

THTMLTableRowAttributes (see page 15304)

78.1.10.1 THTMLTableElementAttributes Members

The following tables list the members exposed by THTMLTableElementAttributes.

Protected Methods

	Name	Description
	AssignTo (see page 15297)	Copies the properties of a THTMLTableElementAttributes (see page 15295) object to another THTMLTableElementAttributes (see page 15295) object.
	Changed (see page 15307)	Generates an OnChange (see page 15308) event.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Events

	Name	Description
	OnChange (see page 15308)	Occurs immediately after a tag attributes property changes.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 15306)	Creates an instance of THTMLTagAttributes (see page 15304).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.

	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RestoreDefaults (see page 15298)	Changes all properties of the attributes object back to their initial settings.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Producer (see page 15308)	Indicates the content producer that uses the THTMLTagAttributes (see page 15304) object to represent the options for one of the HTML tags in the content it produces.

Published Properties

	Name	Description
	Align (see page 15299)	Specifies the horizontal alignment of the contents of the HTML table element.
	BgColor (see page 15299)	Specifies the background color of the HTML table element.
	Custom (see page 15307)	Represents a tag option and its value in an HTML tag.
	VAlign (see page 15300)	Specifies the vertical alignment of the contents of the HTML table element.

78.1.10.2 THTMLTableElementAttributes Methods

The methods of the THTMLTableElementAttributes class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 15297)	Copies the properties of a THTMLTableElementAttributes (see page 15295) object to another THTMLTableElementAttributes (see page 15295) object.

Public Methods

	Name	Description
	RestoreDefaults (see page 15298)	Changes all properties of the attributes object back to their initial settings.

78.1.10.2.1 THTMLTableElementAttributes.AssignTo Method

Copies the properties of a THTMLTableElementAttributes (see page 15295) object to another THTMLTableElementAttributes (see page 15295) object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Applications can not call the protected AssignTo method. To assign the properties of a THTMLTableElementAttributes (see page 15295) object to another object, use the Assign method of the destination object.

When the Assign method of another object specifies a THTMLTableElementAttributes (see page 15295) object as the source of the assignment, the AssignTo method of the THTMLTableElementAttributes (see page 15295) object is called. AssignTo checks whether the Dest parameter is another THTMLTableElementAttributes (see page 15295) object, and if so, copies the values of the BgColor (see page 15299), Align (see page 15299), and VAlign (see page 15300) properties. If Dest is any other class of object, the inherited method is called.

Note: AssignTo does not copy the Custom (see page 15307) property.

See Also

Align (see page 15299)

Assign (see page 2930)

VAlign (see page 15300)

78.1.10.2.2 THTMLTableElementAttributes.RestoreDefaults Method

Changes all properties of the attributes object back to their initial settings.

Pascal

```
procedure RestoreDefaults; override;
```

C++

```
virtual __fastcall RestoreDefaults();
```

Description

Call RestoreDefaults to undo any changes that were made to the table element attributes either programmatically, or using the Object Inspector at design time. RestoreAttributes sets

Align (see page 15299) to haDefault, and VAlign (see page 15300) to haVDefault.

BgColor (see page 15299) to an empty string.

Custom (see page 15307) to an empty string.

See Also

Align (see page 15299)

BgColor (see page 15299)

Custom (see page 15307)

VAlign (see page 15300)

78.1.10.3 THTMLTableElementAttributes Properties

The properties of the THTMLTableElementAttributes class are listed here.

Published Properties

	Name	Description
	Align (see page 15299)	Specifies the horizontal alignment of the contents of the HTML table element.
	BgColor (see page 15299)	Specifies the background color of the HTML table element.
	VAlign (see page 15300)	Specifies the vertical alignment of the contents of the HTML table element.

78.1.10.3.1 THTMLTableElementAttributes.Align Property

Specifies the horizontal alignment of the contents of the HTML table element.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Set Align to indicate the desired horizontal alignment of the contents of the cell or cells indicated by the table element. Align indicates the desired value of the Align option of the HTML tag for the table element.

The default value of Align is haDefault, which means the web browser that reads the HTML document horizontally positions the contents of the table element wherever it wishes.

The Align property of table data cells or header cells overrides the Align property of the table row in which they appear.

See Also

VAlign (see page 15300)

78.1.10.3.2 THTMLTableElementAttributes.BgColor Property

Specifies the background color of the HTML table element.

Pascal

```
property BgColor: THTMLBgColor;
```

C++

```
__property THTMLBgColor BgColor;
```

Description

Use BgColor to indicate the desired background color of the HTML table element. The value of BgColor is the name of color. This can be one of the following strings:

Use the BgColor property of the THTMLTableAttributes (see page 15288) object to set a background color for the entire table. Use the BgColor property of the table element attributes object to make a table element stand out from the rest of the table.

See Also

BgColor (see page 15293)

78.1.10.3.3 THTMLTableElementAttributes.VAlign Property

Specifies the vertical alignment of the contents of the HTML table element.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

Set VAlign to indicate the desired vertical alignment of the contents of the HTML table element. VAlign indicates the desired value of the VAlign option of the HTML tag for the table element.

The default value of VAlign is haVDefault, which means the web browser that reads the HTML document positions the contents of cells in the table element wherever it wishes vertically.

The VAlign property of table data cells or header cells overrides the VAlign property of the table row in which they appear.

See Also

Align (see page 15299)

78.1.11 THTMLTableHeaderAttributes Class

THTMLTableHeaderAttributes represents the display attributes of the header tag in an HTML table.

Class Hierarchy

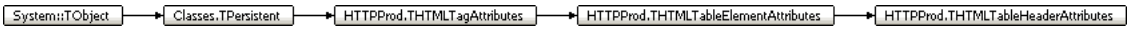

Pascal

```
THTMLTableHeaderAttributes = class(THTMLTableElementAttributes);
```

C++

```
class THTMLTableHeaderAttributes : public THTMLTableElementAttributes;
```

File

HTTPProd

Description

Use THTMLTableHeaderAttributes to describe the desired appearance of a header cell in an HTML table. The properties of the THTMLTableHeaderAttributes object correspond to the options that can be added to the HTML header tag (TH). When the attributes of all the parts of the HTML table have been specified, the HTML content producer creates the HTML commands to specify the table as described.

See Also

THTMLTableCellAttributes (see page 15295)

78.1.11.1 THTMLTableHeaderAttributes Members

The following tables list the members exposed by THTMLTableHeaderAttributes.

Protected Methods

	Name	Description
	AssignTo (see page 15302)	Copies the properties of a THTMLTableElementAttributes object to another THTMLTableElementAttributes object.
	Changed (see page 15307)	Generates an OnChange (see page 15308) event.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Events

	Name	Description
	OnChange (see page 15308)	Occurs immediately after a tag attributes property changes.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 15306)	Creates an instance of THTMLTagAttributes (see page 15304).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RestoreDefaults (see page 15303)	Changes all properties of the attributes object back to their initial settings.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Producer (see page 15308)	Indicates the content producer that uses the THTMLTagAttributes (see page 15304) object to represent the options for one of the HTML tags in the content it produces.

Published Properties

	Name	Description
	Align (see page 15299)	Specifies the horizontal alignment of the contents of the HTML table element.
	BgColor (see page 15299)	Specifies the background color of the HTML table element.
	Caption (see page 15303)	Represents the text that appears in the header cell.
	Custom (see page 15307)	Represents a tag option and its value in an HTML tag.
	VAlign (see page 15300)	Specifies the vertical alignment of the contents of the HTML table element.

78.1.11.2 THTMLTableHeaderAttributes Methods

The methods of the THTMLTableHeaderAttributes class are listed here.

Protected Methods

	Name	Description
V	AssignTo (see page 15302)	Copies the properties of a THTMLTableElementAttributes object to another THTMLTableElementAttributes object.

Public Methods

	Name	Description
V	RestoreDefaults (see page 15303)	Changes all properties of the attributes object back to their initial settings.

78.1.11.2.1 THTMLTableHeaderAttributes.AssignTo Method

Copies the properties of a THTMLTableElementAttributes object to another THTMLTableElementAttributes object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Applications can not call the protected AssignTo method. To assign the properties of a THTMLTableElementAttributes object to another object, use the Assign method of the destination object.

When the Assign method of another object specifies a THTMLTableElementAttributes object as the source of the assignment, the AssignTo method of the THTMLTableElementAttributes object is called. AssignTo checks whether the Dest parameter is another THTMLTableElementAttributes object, and if so, copies the values of the BgColor, Align, and VAlign properties. If Dest is any other class of object, the inherited method is called.

Note: AssignTo does not copy the Custom property.

See Also

Align (see page 15299)

Assign (see page 2930)

VAlign (see page 15300)

78.1.11.2.2 THTMLTableHeaderAttributes.RestoreDefaults Method

Changes all properties of the attributes object back to their initial settings.

Pascal

```
procedure RestoreDefaults; override;
```

C++

```
virtual __fastcall RestoreDefaults();
```

Description

Call RestoreDefaults to undo any changes that were made to the table element attributes either programmatically, or using the Object Inspector at design time. RestoreAttributes sets

Align (see page 15299) to haDefault, and VAlign (see page 15300) to haVDefault.

BgColor (see page 15299) to an empty string.

Caption (see page 15303) to an empty string.

Custom (see page 15307) to an empty string.

See Also

Align (see page 15299)

BgColor (see page 15299)

Caption (see page 15303)

Custom (see page 15307)

VAlign (see page 15300)

78.1.11.3 THTMLTableHeaderAttributes Properties

The properties of the THTMLTableHeaderAttributes class are listed here.

Published Properties

	Name	Description
	Caption (see page 15303)	Represents the text that appears in the header cell.

78.1.11.3.1 THTMLTableHeaderAttributes.Caption Property

Represents the text that appears in the header cell.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Set the Caption property to provide a caption for the column that contains the header cell. If the header cell is used by a dataset table producer or a query table producer, the Caption property overrides the field name that would label the column by default.

See Also

TDataSetTableProducer (see page 10277)

TQueryTableProducer (see page 8152)

TSQLQueryTableProducer (see page 10842)

78.1.12 THTMLTableRowAttributes Class

THTMLTableRowAttributes represents the display attributes of the row tag in an HTML table.

Class Hierarchy

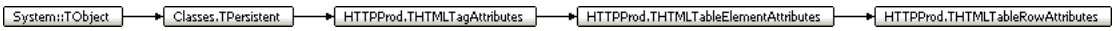

Pascal

```
THTMLTableRowAttributes = class(THTMLTableElementAttributes);
```

C++

```
class THTMLTableRowAttributes : public THTMLTableElementAttributes;
```

File

HTTPProd

Description

Use THTMLTableRowAttributes to describe the desired appearance of a row of an HTML table. The properties of the THTMLTableRowAttributes object correspond to the options that can be added to the HTML row tag (TR). When the attributes of all the parts of the HTML table have been specified, the content producer creates the HTML commands to specify the table as described.

See Also

TDSTableProducer (see page 10290)

THTMLTableHeaderAttributes (see page 15300)

Specifying the Row Attributes

78.1.13 THTMLTagAttributes Class

THTMLTagAttributes is the base class for all objects that represent the attributes of an HTML tag.

Class Hierarchy

Pascal

```
THTMLTagAttributes = class(TPersistent);
```

C++

```
class THTMLTagAttributes : public TPersistent;
```

File

HTTPProd

Description

Descendants of TCustomContentProducer use THTMLTagAttributes objects to encapsulate the options added to the HTML tags in the HTML commands that they produce. When the TCustomContentProducer object generates HTML commands, it looks up the tag attributes represented by the properties of its tag attributes objects, and generates the appropriate tag options.

See Also

TCustomContentProducer (see page 15102)

78.1.13.1 THTMLTagAttributes Members

The following tables list the members exposed by THTMLTagAttributes.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 15307)	Generates an OnChange (see page 15308) event.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Events

	Name	Description
	OnChange (see page 15308)	Occurs immediately after a tag attributes property changes.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 15306)	Creates an instance of THTMLTagAttributes (see page 15304).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	RestoreDefaults (see page 15307)	Changes all properties of the tag attributes object back to their initial settings.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Producer (see page 15308)	Indicates the content producer that uses the THTMLTagAttributes (see page 15304) object to represent the options for one of the HTML tags in the content it produces.

Published Properties

	Name	Description
	Custom (see page 15307)	Represents a tag option and its value in an HTML tag.

78.1.13.2 THTMLTagAttributes.Create Constructor

Creates an instance of THTMLTagAttributes (see page 15304).

Pascal

```
constructor Create(Producer: TCustomContentProducer);
```

C++

```
__fastcall THTMLTagAttributes(TCustomContentProducer Producer);
```

Description

Call Create to create and initialize an instance of THTMLTagAttributes (see page 15304) at runtime. After calling the inherited constructor, Create sets the Producer (see page 15308) property to the value of the Producer (see page 15308) parameter.

See Also

Producer (see page 15308)

Create (see page 22287)

78.1.13.3 THTMLTagAttributes Methods

The methods of the THTMLTagAttributes class are listed here.

Protected Methods

	Name	Description
	Changed (see page 15307)	Generates an OnChange (see page 15308) event.

Public Methods

	Name	Description
V	RestoreDefaults (see page 15307)	Changes all properties of the tag attributes object back to their initial settings.

78.1.13.3.1 THTMLTagAttributes.Changed Method

Generates an OnChange (see page 15308) event.

Pascal

```
procedure Changed;
```

C++

```
__fastcall Changed();
```

Description

Call Changed from a descendant of THTMLTagAttributes (see page 15304) after a tag option introduced by that descendant changes. Changed is exposed as a protected method so that component writers can create descendants of THTMLTagAttributes (see page 15304) which correctly trigger the OnChange (see page 15308) event handler.

See Also

OnChange (see page 15308)

78.1.13.3.2 THTMLTagAttributes.RestoreDefaults Method

Changes all properties of the tag attributes object back to their initial settings.

Pascal

```
procedure RestoreDefaults; virtual;
```

C++

```
virtual __fastcall RestoreDefaults();
```

Description

Call RestoreDefaults to undo any changes that were made to the tag attributes either programmatically, or at design time. RestoreAttributes sets the Custom (see page 15307) property to an empty string.

See Also

Custom (see page 15307)

78.1.13.4 THTMLTagAttributes Properties

The properties of the THTMLTagAttributes class are listed here.

Public Properties

	Name	Description
R	Producer (see page 15308)	Indicates the content producer that uses the THTMLTagAttributes (see page 15304) object to represent the options for one of the HTML tags in the content it produces.

Published Properties

	Name	Description
	Custom (see page 15307)	Represents a tag option and its value in an HTML tag.

78.1.13.4.1 THTMLTagAttributes.Custom Property

Represents a tag option and its value in an HTML tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to represent one or more options to be added to an HTML tag. Each option and its value should have the form

`OPTIONNAME=Value`

The options specified by Custom should be separated by spaces. For example, to merge a cell in an HTML table with the three adjacent cells to the left and below, the table data tag for the cell requires the ROWSPAN and COLSPAN options to be set. The value of Custom to specify these values would be the string

`ROWSPAN=2 COLSPAN=2`

The HTML producer inserts the Custom string into the HTML tag when it generates its Content.

See Also

Producer (see page 15308)

Content (see page 15106)

78.1.13.4.2 THTMLTagAttributes.Producer Property

Indicates the content producer that uses the THTMLTagAttributes (see page 15304) object to represent the options for one of the HTML tags in the content it produces.

Pascal

```
property Producer: TCustomContentProducer;
```

C++

```
__property TCustomContentProducer Producer;
```

Description

Use Producer to gain access to the HTML producer object that is using the THTMLTagAttributes (see page 15304) object. The THTMLTagAttributes (see page 15304) object can use the Producer property in an OnChange (see page 15308) event handler to notify the TCustomContentProducer object of changes to the attributes it represents.

See Also

OnChange (see page 15308)

78.1.13.5 THTMLTagAttributes Events

The events of the THTMLTagAttributes class are listed here.

Public Events

	Name	Description
	OnChange (see page 15308)	Occurs immediately after a tag attributes property changes.

78.1.13.5.1 THTMLTagAttributes.OnChange Event

Occurs immediately after a tag attributes property changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever any of the properties of the tag attributes object changes. When OnChange occurs, the tag attributes object already has the new property setting. Use an OnChange event handler to inform the HTML content producer whenever the attributes of a tag change so that the content producer can update the content it is producing.

See Also

Producer (see page 15308)

78.1.14 TPageProducer Class

TPageProducer produces a string of HTML commands based on an input template.

Class Hierarchy

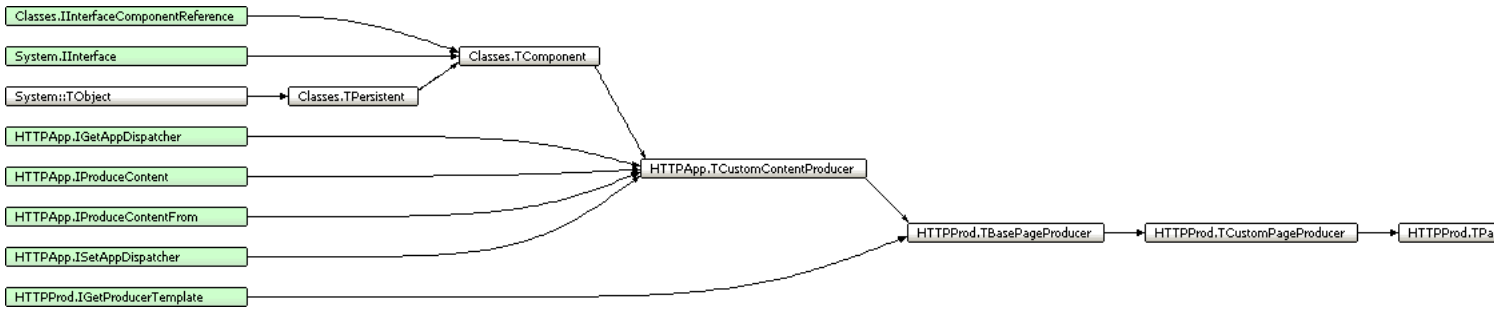

Pascal

```
TPageProducer = class(TCustomPageProducer);
```

C++

```
class TPageProducer : public TCustomPageProducer;
```

File

HTTPProd

Description

Use TPageProducer to convert an HTML template into a string of HTML commands that can be interpreted by a client application such as a Web browser. In Web Broker applications, HTML templates include HTML commands and HTML-transparent tags that are replaced with customized content by the OnHTMLTag (see page 15316) event. WebSnap application page templates often include server-side scripting (such as JScript or JavaScript) to dynamically generate content.

See Also

TDataSetTableProducer (see page 10277)

TSQLQueryTableProducer (see page 10842)

TQueryTableProducer (see page 8152)

TDataSetPageProducer (see page 11807)

TAdapterPageProducer (see page 19802)

The Web Module

Using Page Producer Components

78.1.14.1 TPageProducer Members

The following tables list the members exposed by TPageProducer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoTagEvent (see page 15276)	This is DoTagEvent, a member of class TBasePageProducer.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetProducerTemplateFileName (see page 15276)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.
	GetProducerTemplateStream (see page 15276)	This is GetProducerTemplateStream, a member of class TBasePageProducer.
V	GetScriptEngine (see page 15277)	This is GetScriptEngine, a member of class TBasePageProducer.
	GetTagID (see page 15277)	This is GetTagID, a member of class TBasePageProducer.
V	GetTemplateFileName (see page 15277)	This is GetTemplateFileName, a member of class TBasePageProducer.
V	GetTemplateFileName (see page 15286)	This is GetTemplateFileName, a member of class TCustomPageProducer.
V	GetTemplateStream (see page 15277)	This is GetTemplateStream, a member of class TBasePageProducer.
V	GetTemplateStream (see page 15286)	This is GetTemplateStream, a member of class TCustomPageProducer.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleScriptTag (see page 15277)	This is HandleScriptTag, a member of class TBasePageProducer.
V	HandleTag (see page 15278)	This is HandleTag, a member of class TBasePageProducer.
V	HandleTag (see page 15286)	This is HandleTag, a member of class TCustomPageProducer.
	ImplHandleTag (see page 15278)	This is ImplHandleTag, a member of class TBasePageProducer.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.

	Notification (see page 15108)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ServerScriptFromStream (see page 15278)	This is ServerScriptFromStream, a member of class TBasePageProducer.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UseScriptEngine (see page 15278)	This is UseScriptEngine, a member of class TBasePageProducer.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 15106)	Returns the content produced by the content producer.
	Content (see page 15274)	Returns the content produced by the content producer.
	ContentFromStream (see page 15107)	Returns the content produced in response to input read from a stream.
	ContentFromStream (see page 15275)	Returns the content produced in response to input read from a stream.
	ContentFromString (see page 15107)	Returns the content produced in response to an input string.
	ContentFromString (see page 15275)	Returns the content produced in response to an input string.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 15274)	Creates an instance of TBasePageProducer (see page 15269).

	Create (see page 15285)	Creates an instance of TCustomPageProducer (see page 15281).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 15285)	Destroys an instance of TCustomPageProducer (see page 15281).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProducerTemplateFileName (see page 15320)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.
	GetProducerTemplateStream (see page 15320)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.

	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebModuleContext (see page 15279)	Provides access to the Web module context object.

Published Events

	Name	Description
	OnHTMLTag (see page 15316)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Published Properties

	Name	Description
	HTMLDoc (see page 15314)	Specifies the HTML template used to generate the value returned by the Content method.
	HTMLFile (see page 15314)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ScriptEngine (see page 15315)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 15315)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.
	Tag (see page 2808)	Stores an integer value as part of a component.

78.1.14.2 TPageProducer Properties

The properties of the TPageProducer class are listed here.

Published Properties

	Name	Description
	HTMLDoc (see page 15314)	Specifies the HTML template used to generate the value returned by the Content method.
	HTMLFile (see page 15314)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	ScriptEngine (see page 15315)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 15315)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.

78.1.14.2.1 TPageProducer.HTMLDoc Property

Specifies the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLDoc: TStrings;
```

C++

```
__property TStrings HTMLDoc;
```

Description

Set HTMLDoc to a set of strings that make up the HTML template. Each string is a sequence of one or more HTML commands or HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLDoc clears the HTMLFile property.

See Also

Content (see page 15274)

HTMLFile (see page 15287)

Specifying the HTML Template

78.1.14.2.2 TPageProducer.HTMLFile Property

Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLFile: TFileName;
```

C++

```
__property TFileName HTMLFile;
```

Description

Set HTMLFile to the name of the file that contains an HTML template. The HTML template is a sequence of HTML commands and HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white

space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLFile clears the HTMLDoc property.

Note: When distributing applications that contain page producers which use the HTMLFile property, be sure to distribute the file along with the application. Also, be sure that the value of HTMLFile does not include a hard-coded path name that will not be available on the user's disk.

See Also

Content (see page 15274)

HTMLDoc (see page 15287)

Specifying the HTML Template

78.1.14.2.3 TPageProducer.ScriptEngine Property

Names the script engine that generates or interprets script embedded in the documents that this producer creates.

Pascal

```
property ScriptEngine: string;
```

C++

```
__property AnsiString ScriptEngine;
```

Description

ScriptEngine indicates the scripting language that the producer uses in its generated documents. For example, if the producer generates code that includes jscript, the value of ScriptEngine is 'JSCRIPT'.

When ScriptEngine is defined, the page producer generates its content using a special script-producer component. When ScriptEngine is not defined, the page producer uses only the OnHTMLTag event, plus any other built-in translations provided by a descendant class.

See Also

OnHTMLTag (see page 15280)

78.1.14.2.4 TPageProducer.StripParamQuotes Property

Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.

Pascal

```
property StripParamQuotes: Boolean;
```

C++

```
__property Boolean StripParamQuotes;
```

Description

Set StripParamQuotes to indicate whether the page producer should strip quotation marks from the parameters of HTML-transparent tags before handling the tags. By setting StripParamQuotes to true, the page producer compensates for HTML editors that automatically insert quotation marks around the values of tag options.

78.1.14.3 TPageProducer Events

The events of the TPageProducer class are listed here.

Published Events

	Name	Description
	OnHTMLTag (see page 15316)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

78.1.14.3.1 TPageProducer.OnHTMLTag Event

Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Pascal

```
property OnHTMLTag: THTMLTagEvent;
```

C++

```
__property THTMLTagEvent OnHTMLTag;
```

Description

Write an OnHTMLTag event handler to convert an HTML-transparent tag into the corresponding HTML commands that can be interpreted by a Web browser. If an OnHTMLTag event is not supplied, the HTML-transparent tags in the HTML template will be replaced by empty strings.

Note: The sequence of HTML that the event handler produces for an HTML-transparent tag can contain HTML-transparent tags for another page producer to convert. For example, one page producer might have a template that represents the format of the final HTTP response. It could read the HTTP request and for each section in the request assemble a set of HTML-transparent tags with parameters based on the request. Another page producer might take the content of the first page producer, and interpret those parameters.

See Also

Names (see page 3075)

Values (see page 3086)

Customizing the InternetExpress Page Producer Template

Chaining Page Producers Together

Converting HTML-transparent Tags

78.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IDesignerFileManager (see page 15317)	This is class HTTPProd.IDesignerFileManager.
	IGetLocateFileService (see page 15318)	This is class HTTPProd.IGetLocateFileService.
	IGetProducerTemplate (see page 15319)	This is class HTTPProd.IGetProducerTemplate.
	ILocateFileService (see page 15320)	This is class HTTPProd.ILocateFileService.
	IScriptContext (see page 15321)	This is class HTTPProd.IScriptContext.
	IScriptProducer (see page 15323)	This is class HTTPProd.IScriptProducer.
	IWebVariableName (see page 15328)	This is class HTTPProd.IWebVariableName.
	IWebVariablesContainer (see page 15329)	This is class HTTPProd.IWebVariablesContainer.

78.2.1 IDesignerFileManager Interface

Class Hierarchy

HTTPProd.IDesignerFileManager

Pascal

```
IDesignerFileManager = interface;
```

C++

```
__interface IDesignerFileManager;
```

File

HTTPProd

Description

This is class HTTPProd.IDesignerFileManager.

78.2.1.1 IDesignerFileManager Members

The following tables list the members exposed by IDesignerFileManager.

Public Methods

	Name	Description
	GetStream (see page 15318)	This is GetStream, a member of class IDesignerFileManager.
	QualifyFileName (see page 15318)	This is QualifyFileName, a member of class IDesignerFileManager.

78.2.1.2 IDesignerFileManager Methods

The methods of the IDesignerFileManager class are listed here.

Public Methods

	Name	Description
	GetStream (see page 15318)	This is GetStream, a member of class IDesignerFileManager.
	QualifyFileName (see page 15318)	This is QualifyFileName, a member of class IDesignerFileManager.

78.2.1.2.1 IDesignerFileManager.GetStream Method

Pascal

```
function GetStream(const AFileName: string; var AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream GetStream(const AnsiString AFileName, Boolean AOwned);
```

Description

This is GetStream, a member of class IDesignerFileManager.

78.2.1.2.2 IDesignerFileManager.QualifyFileName Method

Pascal

```
function QualifyFileName(const AFileName: string): string;
```

C++

```
__fastcall AnsiString QualifyFileName(const AnsiString AFileName);
```

Description

This is QualifyFileName, a member of class IDesignerFileManager.

78.2.2 IGetLocateFileService Interface

Class Hierarchy

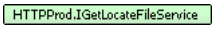

Pascal

```
IGetLocateFileService = interface;
```

C++

```
__interface IGetLocateFileService;
```

File

HTTPProd

Description

This is class HTTPProd.IGetLocateFileService.

78.2.2.1 IGetLocateFileService Members

The following tables list the members exposed by IGetLocateFileService.

Public Methods

	Name	Description
	GetLocateFileService (see page 15319)	This is GetLocateFileService, a member of class IGetLocateFileService.

78.2.2.2 IGetLocateFileService Methods

The methods of the IGetLocateFileService class are listed here.

Public Methods

	Name	Description
	GetLocateFileService (see page 15319)	This is GetLocateFileService, a member of class IGetLocateFileService.

78.2.2.2.1 IGetLocateFileService.GetLocateFileService Method

Pascal

```
function GetLocateFileService: ILocateFileService;
```

C++

```
__fastcall ILocateFileService GetLocateFileService();
```

Description

This is GetLocateFileService, a member of class IGetLocateFileService.

78.2.3 IGetProducerTemplate Interface

Class Hierarchy

HTTPProd.IGetProducerTemplate

Pascal

```
IGetProducerTemplate = interface;
```

C++

```
__interface IGetProducerTemplate;
```

File

HTTPProd

Description

This is class HTTPProd.IGetProducerTemplate.

78.2.3.1 IGetProducerTemplate Members

The following tables list the members exposed by IGetProducerTemplate.

Public Methods

	Name	Description
	GetProducerTemplateFileName (see page 15320)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.

	GetProducerTemplateStream (see page 15320)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.

78.2.3.2 IGetProducerTemplate Methods

The methods of the IGetProducerTemplate class are listed here.

Public Methods

	Name	Description
	GetProducerTemplateFileName (see page 15320)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.
	GetProducerTemplateStream (see page 15320)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.

78.2.3.2.1 IGetProducerTemplate.GetProducerTemplateFileName Method

Pascal

```
function GetProducerTemplateFileName: string;
```

C++

```
__fastcall AnsiString GetProducerTemplateFileName();
```

Description

This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.

78.2.3.2.2 IGetProducerTemplate.GetProducerTemplateStream Method

Pascal

```
function GetProducerTemplateStream(out AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream GetProducerTemplateStream(Boolean AOwned);
```

Description

This is GetProducerTemplateStream, a member of class IGetProducerTemplate.

78.2.4 ILocateFileService Interface

Class Hierarchy

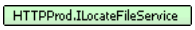

Pascal

```
ILocateFileService = interface;
```

C++

```
__interface ILocateFileService;
```

File

HTTPProd

Description

This is class HTTPProd.ILocateFileService.

78.2.4.1 ILocateFileService Members

The following tables list the members exposed by ILocateFileService.

Public Methods

	Name	Description
	GetTemplateStream (see page 15321)	This is GetTemplateStream, a member of class ILocateFileService.

78.2.4.2 ILocateFileService Methods

The methods of the ILocateFileService class are listed here.

Public Methods

	Name	Description
	GetTemplateStream (see page 15321)	This is GetTemplateStream, a member of class ILocateFileService.

78.2.4.2.1 ILocateFileService.GetTemplateStream Method

Pascal

```
function GetTemplateStream(AComponent: TComponent; AFileName: string; out AOwned: Boolean):
TStream;
```

C++

```
__fastcall TStream GetTemplateStream(TComponent * AComponent, AnsiString AFileName, Boolean
AOwned);
```

Description

This is GetTemplateStream, a member of class ILocateFileService.

78.2.5 IScriptContext Interface

Class Hierarchy

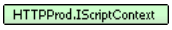

Pascal

```
IScriptContext = interface;
```

C++

```
__interface IScriptContext;
```

File

HTTPProd

Description

This is class HTTPProd.IScriptContext.

78.2.5.1 IScriptContext Members

The following tables list the members exposed by IScriptContext.

Public Methods

	Name	Description
	GetWebModuleContext (see page 15322)	This is GetWebModuleContext, a member of class IScriptContext.

Public Properties

	Name	Description
R	WebModuleContext (see page 15322)	This is WebModuleContext, a member of class IScriptContext.

78.2.5.2 IScriptContext Methods

The methods of the IScriptContext class are listed here.

Public Methods

	Name	Description
	GetWebModuleContext (see page 15322)	This is GetWebModuleContext, a member of class IScriptContext.

78.2.5.2.1 IScriptContext.GetWebModuleContext Method

Pascal

```
function GetWebModuleContext: TWebModuleContext;
```

C++

```
__fastcall TWebModuleContext GetWebModuleContext();
```

Description

This is GetWebModuleContext, a member of class IScriptContext.

78.2.5.3 IScriptContext Properties

The properties of the IScriptContext class are listed here.

Public Properties

	Name	Description
R	WebModuleContext (see page 15322)	This is WebModuleContext, a member of class IScriptContext.

78.2.5.3.1 IScriptContext.WebModuleContext Property

Pascal

```
property WebModuleContext: TWebModuleContext;
```

C++

```
__property TWebModuleContext WebModuleContext;
```

Description

This is WebModuleContext, a member of class IScriptContext.

78.2.6 IScriptProducer Interface

Class Hierarchy

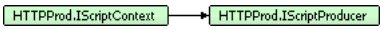

Pascal

```
IScriptProducer = interface(IScriptContext);
```

C++

```
__interface IScriptProducer : public IScriptContext;
```

File

HTTPProd

Description

This is class HTTPProd.IScriptProducer.

78.2.6.1 IScriptProducer Members

The following tables list the members exposed by IScriptProducer.

Public Methods

	Name	Description
	Evaluate (see page 15324)	This is Evaluate, a member of class IScriptProducer.
	GetContent (see page 15324)	This is GetContent, a member of class IScriptProducer.
	GetErrors (see page 15324)	This is GetErrors, a member of class IScriptProducer.
	GetHTMLBlock (see page 15325)	This is GetHTMLBlock, a member of class IScriptProducer.
	GetHTMLBlockCount (see page 15325)	This is GetHTMLBlockCount, a member of class IScriptProducer.
	GetWebModuleContext (see page 15322)	This is GetWebModuleContext, a member of class IScriptContext.
	HandleScriptError (see page 15325)	This is HandleScriptError, a member of class IScriptProducer.
	ParseStream (see page 15325)	This is ParseStream, a member of class IScriptProducer.
	ParseString (see page 15325)	This is ParseString, a member of class IScriptProducer.
	ReplaceTags (see page 15326)	This is ReplaceTags, a member of class IScriptProducer.
	SetContent (see page 15326)	This is SetContent, a member of class IScriptProducer.
	Write (see page 15326)	This is the overview for the Write method overload.
	WriteItem (see page 15327)	This is WriteItem, a member of class IScriptProducer.

Public Properties

	Name	Description
	Content (see page 15327)	This is Content, a member of class IScriptProducer.
R	Errors (see page 15327)	This is Errors, a member of class IScriptProducer.
R	HTMLBlockCount (see page 15328)	This is HTMLBlockCount, a member of class IScriptProducer.
R	HTMLBlocks (see page 15328)	This is HTMLBlocks, a member of class IScriptProducer.
R	WebModuleContext (see page 15322)	This is WebModuleContext, a member of class IScriptContext.

78.2.6.2 IScriptProducer Methods

The methods of the IScriptProducer class are listed here.

Public Methods

	Name	Description
	Evaluate (see page 15324)	This is Evaluate, a member of class IScriptProducer.
	GetContent (see page 15324)	This is GetContent, a member of class IScriptProducer.
	GetErrors (see page 15324)	This is GetErrors, a member of class IScriptProducer.
	GetHTMLBlock (see page 15325)	This is GetHTMLBlock, a member of class IScriptProducer.
	GetHTMLBlockCount (see page 15325)	This is GetHTMLBlockCount, a member of class IScriptProducer.
	HandleScriptError (see page 15325)	This is HandleScriptError, a member of class IScriptProducer.
	ParseStream (see page 15325)	This is ParseStream, a member of class IScriptProducer.
	ParseString (see page 15325)	This is ParseString, a member of class IScriptProducer.
	ReplaceTags (see page 15326)	This is ReplaceTags, a member of class IScriptProducer.
	SetContent (see page 15326)	This is SetContent, a member of class IScriptProducer.
	Write (see page 15326)	This is the overview for the Write method overload.
	WriteItem (see page 15327)	This is WriteItem, a member of class IScriptProducer.

78.2.6.2.1 IScriptProducer.Evaluate Method

Pascal

```
function Evaluate: string;
```

C++

```
__fastcall AnsiString Evaluate();
```

Description

This is Evaluate, a member of class IScriptProducer.

78.2.6.2.2 IScriptProducer.GetContent Method

Pascal

```
function GetContent: string;
```

C++

```
__fastcall AnsiString GetContent();
```

Description

This is GetContent, a member of class IScriptProducer.

78.2.6.2.3 IScriptProducer.GetErrors Method

Pascal

```
function GetErrors: TAbstractScriptErrors;
```

C++

```
__fastcall TAbstractScriptErrors GetErrors();
```

Description

This is GetErrors, a member of class IScriptProducer.

78.2.6.2.4 IScriptProducer.GetHTMLBlock Method

Pascal

```
function GetHTMLBlock(I: Integer): string;
```

C++

```
__fastcall AnsiString GetHTMLBlock(int I);
```

Description

This is GetHTMLBlock, a member of class IScriptProducer.

78.2.6.2.5 IScriptProducer.GetHTMLBlockCount Method

Pascal

```
function GetHTMLBlockCount: Integer;
```

C++

```
__fastcall int GetHTMLBlockCount();
```

Description

This is GetHTMLBlockCount, a member of class IScriptProducer.

78.2.6.2.6 IScriptProducer.HandleScriptError Method

Pascal

```
function HandleScriptError(const ScriptError: IUnknown): HRESULT;
```

C++

```
__fastcall HRESULT HandleScriptError(const IUnknown ScriptError);
```

Description

This is HandleScriptError, a member of class IScriptProducer.

78.2.6.2.7 IScriptProducer.ParseStream Method

Pascal

```
procedure ParseStream(Stream: TStream; Owned: Boolean = False);
```

C++

```
__fastcall ParseStream(TStream Stream, Boolean Owned = False);
```

Description

This is ParseStream, a member of class IScriptProducer.

78.2.6.2.8 IScriptProducer.ParseString Method

Pascal

```
procedure ParseString(const S: string);
```

C++

```
__fastcall ParseString(const AnsiString S);
```

Description

This is ParseString, a member of class IScriptProducer.

78.2.6.2.9 IScriptProducer.ReplaceTags Method

Pascal

```
function ReplaceTags(const S: string): string;
```

C++

```
__fastcall AnsiString ReplaceTags(const AnsiString S);
```

Description

This is ReplaceTags, a member of class IScriptProducer.

78.2.6.2.10 IScriptProducer.SetContent Method

Pascal

```
procedure SetContent(const Value: string);
```

C++

```
__fastcall SetContent(const AnsiString Value);
```

Description

This is SetContent, a member of class IScriptProducer.

78.2.6.2.11 Write Method

This is the overview for the Write method overload.

Overload List

	Name	Description
	IScriptProducer.Write (PWideChar, Integer) (see page 15326)	This is Write, a member of class IScriptProducer.
	IScriptProducer.Write (string) (see page 15326)	This is Write, a member of class IScriptProducer.

78.2.6.2.11.1 IScriptProducer.Write Method (PWideChar, Integer)

Pascal

```
procedure Write(const Value: PWideChar; ALength: Integer); overload;
```

C++

```
__fastcall Write(const PWideChar Value, int ALength);
```

Description

This is Write, a member of class IScriptProducer.

78.2.6.2.11.2 IScriptProducer.Write Method (string)

Pascal

```
procedure Write(const Value: string); overload;
```

C++

```
__fastcall Write(const AnsiString Value);
```

Description

This is Write, a member of class IScriptProducer.

78.2.6.2.12 IScriptProducer.WriteItem Method

Pascal

```
procedure WriteItem(Index: Integer);
```

C++

```
__fastcall WriteItem(int Index);
```

Description

This is WriteItem, a member of class IScriptProducer.

78.2.6.3 IScriptProducer Properties

The properties of the IScriptProducer class are listed here.

Public Properties

	Name	Description
	Content (see page 15327)	This is Content, a member of class IScriptProducer.
R	Errors (see page 15327)	This is Errors, a member of class IScriptProducer.
R	HTMLBlockCount (see page 15328)	This is HTMLBlockCount, a member of class IScriptProducer.
R	HTMLBlocks (see page 15328)	This is HTMLBlocks, a member of class IScriptProducer.

78.2.6.3.1 IScriptProducer.Content Property

Pascal

```
property Content: string;
```

C++

```
__property AnsiString Content;
```

Description

This is Content, a member of class IScriptProducer.

78.2.6.3.2 IScriptProducer.Errors Property

Pascal

```
property Errors: TAbstractScriptErrors;
```

C++

```
__property TAbstractScriptErrors Errors;
```

Description

This is Errors, a member of class IScriptProducer.

78.2.6.3.3 IScriptProducer.HTMLBlockCount Property

Pascal

```
property HTMLBlockCount: Integer;
```

C++

```
__property int HTMLBlockCount;
```

Description

This is HTMLBlockCount, a member of class IScriptProducer.

78.2.6.3.4 IScriptProducer.HTMLBlocks Property

Pascal

```
property HTMLBlocks [I: Integer]: string;
```

C++

```
__property AnsiString HTMLBlocks[int I];
```

Description

This is HTMLBlocks, a member of class IScriptProducer.

78.2.7 IWebVariableName Interface

Class Hierarchy

HTTPProd.IWebVariableName

Pascal

```
IWebVariableName = interface;
```

C++

```
__interface IWebVariableName;
```

File

HTTPProd

Description

This is class HTTPProd.IWebVariableName.

78.2.7.1 IWebVariableName Members

The following tables list the members exposed by IWebVariableName.

Public Methods

	Name	Description
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.

Public Properties

	Name	Description
	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.

78.2.7.2 IWebVariableName Methods

The methods of the IWebVariableName class are listed here.

Public Methods

	Name	Description
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.

78.2.7.2.1 IWebVariableName.GetVariableName Method

Pascal

```
function GetVariableName: string;
```

C++

```
__fastcall AnsiString GetVariableName();
```

Description

This is GetVariableName, a member of class IWebVariableName.

78.2.7.3 IWebVariableName Properties

The properties of the IWebVariableName class are listed here.

Public Properties

	Name	Description
	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.

78.2.7.3.1 IWebVariableName.VariableName Property

Pascal

```
property VariableName: string;
```

C++

```
__property AnsiString VariableName;
```

Description

This is VariableName, a member of class IWebVariableName.

78.2.8 IWebVariablesContainer Interface

Class Hierarchy

HTTPProd.IWebVariablesContainer

Pascal

```
IWebVariablesContainer = interface;
```

C++

```
__interface IWebVariablesContainer;
```

File

HTTPProd

Description

This is class HTTPProd.IWebVariablesContainer.

78.2.8.1 IWebVariablesContainer Members

The following tables list the members exposed by IWebVariablesContainer.

Public Methods

	Name	Description
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.

Public Properties

	Name	Description
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.

78.2.8.2 IWebVariablesContainer Methods

The methods of the IWebVariablesContainer class are listed here.

Public Methods

	Name	Description
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.

78.2.8.2.1 IWebVariablesContainer.FindVariable Method

Pascal

```
function FindVariable(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariable(const AnsiString AName);
```

Description

This is FindVariable, a member of class IWebVariablesContainer.

78.2.8.2.2 IWebVariablesContainer.FindVariableContainer Method

Pascal

```
function FindVariableContainer(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariableContainer(const AnsiString AName);
```

Description

This is FindVariableContainer, a member of class IWebVariablesContainer.

78.2.8.2.3 IWebVariablesContainer.GetVariable Method

Pascal

```
function GetVariable(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetVariable(int I);
```

Description

This is GetVariable, a member of class IWebVariablesContainer.

78.2.8.2.4 IWebVariablesContainer.GetVariableCount Method

Pascal

```
function GetVariableCount: Integer;
```

C++

```
__fastcall int GetVariableCount();
```

Description

This is GetVariableCount, a member of class IWebVariablesContainer.

78.2.8.3 IWebVariablesContainer Properties

The properties of the IWebVariablesContainer class are listed here.

Public Properties

	Name	Description
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.

78.2.8.3.1 IWebVariablesContainer.VariableCount Property

Pascal

```
property VariableCount: Integer;
```

C++

```
__property int VariableCount;
```

Description

This is VariableCount, a member of class IWebVariablesContainer.

78.2.8.3.2 IWebVariablesContainer.Variables Property

Pascal

```
property Variables [I: Integer]: TComponent;
```

C++

```
__property TComponent * Variables[int I];
```

Description

This is Variables, a member of class IWebVariablesContainer.

78.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	ContentFromScriptFile (see page 15333)	This is function HTTPProd.ContentFromScriptFile.
	ContentFromScriptStream (see page 15333)	This is function HTTPProd.ContentFromScriptStream.
	ContentFromStream (see page 15334)	This is function HTTPProd.ContentFromStream.
	ContentFromString (see page 15334)	This is function HTTPProd.ContentFromString.
	FindComponentWebModuleContext (see page 15334)	This is function HTTPProd.FindComponentWebModuleContext.
	GetTagID (see page 15335)	This is function HTTPProd.GetTagID.

78.3.1 HTTPProd.ContentFromScriptFile Function

Pascal

```
function ContentFromScriptFile(const AFileName: TFileName; AWebModuleContext:
TWebModuleContext; AStripParamQuotes: Boolean; AHandleTag: THandleTagProc; AHandleScriptTag:
THandledTagProc; const AScriptEngine: string; ALocateFileService: ILocateFileService): string;
```

C++

```
AnsiString ContentFromScriptFile(const TFileName AFileName, TWebModuleContext
AWebModuleContext, Boolean AStripParamQuotes, THandleTagProc AHandleTag, THandledTagProc
AHandleScriptTag, const AnsiString AScriptEngine, ILocateFileService ALocateFileService);
```

File

HTTPProd

Description

This is function HTTPProd.ContentFromScriptFile.

78.3.2 HTTPProd.ContentFromScriptStream Function

Pascal

```
function ContentFromScriptStream(AStream: TStream; AWebModuleContext: TWebModuleContext;
AStripParamQuotes: Boolean; AHandleTag: THandleTagProc; AHandleScriptTag: THandledTagProc;
const AScriptEngine: string; ALocateFileService: ILocateFileService): string;
```

C++

```
AnsiString ContentFromScriptStream(TStream AStream, TWebModuleContext AWebModuleContext,
Boolean AStripParamQuotes, THandleTagProc AHandleTag, THandledTagProc AHandleScriptTag, const
AnsiString AScriptEngine, ILocateFileService ALocateFileService);
```

File

HTTPProd

Description

This is function HTTPProd.ContentFromScriptStream.

78.3.3 HTTPProd.ContentFromStream Function

Pascal

```
function ContentFromStream(AStream: TStream; AStripParamQuotes: Boolean; AHandleTag:
THandleTagProc; AHandledTag: THandledTagProc): string;
```

C++

```
AnsiString ContentFromStream(TStream AStream, Boolean AStripParamQuotes, THandleTagProc
AHandleTag, THandledTagProc AHandledTag);
```

File

HTTPProd

Description

This is function HTTPProd.ContentFromStream.

78.3.4 HTTPProd.ContentFromString Function

Pascal

```
function ContentFromString(const AValue: string; AStripParamQuotes: Boolean; AHandleTag:
THandleTagProc; AHandledTag: THandledTagProc): string;
```

C++

```
AnsiString ContentFromString(const AnsiString AValue, Boolean AStripParamQuotes,
THandleTagProc AHandleTag, THandledTagProc AHandledTag);
```

File

HTTPProd

Description

This is function HTTPProd.ContentFromString.

78.3.5 HTTPProd.FindComponentWebModuleContext Function

Pascal

```
function FindComponentWebModuleContext(AComponent: TComponent): TWebModuleContext;
```

C++

```
TWebModuleContext FindComponentWebModuleContext(TComponent * AComponent);
```

File

HTTPProd

Description

This is function HTTPProd.FindComponentWebModuleContext.

78.3.6 HTTPProd.GetTagID Function

Pascal

```
function GetTagID(const TagString: string): TTag;
```

C++

```
TTag GetTagID(const AnsiString TagString);
```

File

HTTPProd

Description

This is function HTTPProd.GetTagID.

78.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	THTMLAlign (see page 15336)	This is record HTTPProd.THTMLAlign.
	THTMLVAlign (see page 15336)	This is record HTTPProd.THTMLVAlign.
	TTag (see page 15337)	This is record HTTPProd.TTag.

78.4.1 HTTPProd.THTMLAlign Enumeration

Pascal

```
THTMLAlign = (
  haDefault,
  haLeft,
  haRight,
  haCenter
);
```

C++

```
enum THTMLAlign {
  haDefault,
  haLeft,
  haRight,
  haCenter
};
```

File

HTTPProd

Description

This is record HTTPProd.THTMLAlign.

78.4.2 HTTPProd.THTMLVAlign Enumeration

Pascal

```
THTMLVAlign = (
  haVDefault,
  haTop,
  haMiddle,
  haBottom,
  haBaseline
);
```

C++

```
enum THTMLVAlign {
  haVDefault,
```

```
    haTop,
    haMiddle,
    haBottom,
    haBaseline
  };
```

File

HTTPProd

Description

This is record HTTPProd.THTMLVAlign.

78.4.3 HTTPProd.TTag Enumeration

Pascal

```
  TTag = (
    tgCustom,
    tgLink,
    tgImage,
    tgTable,
    tgImageMap,
    tgObject,
    tgEmbed
  );
```

C++

```
  enum TTag {
    tgCustom,
    tgLink,
    tgImage,
    tgTable,
    tgImageMap,
    tgObject,
    tgEmbed
  };
```

File

HTTPProd

Description

This is record HTTPProd.TTag.

78.5 Types

The following table lists types in this documentation.

Types

Name	Description
THandledTagProc (see page 15338)	This is type HTTPProd.THandledTagProc.
THandleTagProc (see page 15338)	This is type HTTPProd.THandleTagProc.
THTMLBgColor (see page 15339)	This is type HTTPProd.THTMLBgColor.
THTMLTagEvent (see page 15339)	This is type HTTPProd.THTMLTagEvent.
TScriptProducerClass (see page 15339)	This is type HTTPProd.TScriptProducerClass.

78.5.1 HTTPProd.THandledTagProc Type

Pascal

```
THandledTagProc = function (const TagString: string; TagParams: TStrings; var ReplaceString:
string): Boolean of object;
```

C++

```
(const TagString: string; TagParams: TStrings; var ReplaceString: string): Boolean (
THandledTagProc)();
```

File

HTTPProd

Description

This is type HTTPProd.THandledTagProc.

78.5.2 HTTPProd.THandleTagProc Type

Pascal

```
THandleTagProc = function (const TagString: string; TagParams: TStrings): string of object;
```

C++

```
(const TagString: string; TagParams: TStrings): string ( THandleTagProc)();
```

File

HTTPProd

Description

This is type HTTPProd.THandleTagProc.

78.5.3 HTTPProd.THTMLBgColor Type

Pascal

```
THTMLBgColor = type string;
```

C++

```
type string THTMLBgColor;
```

File

HTTPProd

Description

This is type HTTPProd.THTMLBgColor.

78.5.4 HTTPProd.THTMLTagEvent Type

Pascal

```
THTMLTagEvent = procedure (Sender: TObject; Tag: TTag; const TagString: string; TagParams:
TStrings; var ReplaceText: string) of object;
```

C++

```
(Sender: TObject; Tag: TTag; const TagString: string; TagParams: TStrings; var ReplaceText:
string) ( THTMLTagEvent)();
```

File

HTTPProd

Description

This is type HTTPProd.THTMLTagEvent.

78.5.5 HTTPProd.TScriptProducerClass Type

Pascal

```
TScriptProducerClass = class of TAbstractScriptProducer;
```

C++

```
TAbstractScriptProducer TScriptProducerClass;
```

File

HTTPProd

Description

This is type HTTPProd.TScriptProducerClass.

78.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
DesignerFileManager (see page 15340)	This is variable HTTPProd.DesignerFileManager.
ScriptEnginesList (see page 15340)	This is variable HTTPProd.ScriptEnginesList.

78.6.1 HTTPProd.DesignerFileManager Variable

Pascal

```
DesignerFileManager: IDesignerFileManager = nil;
```

C++

```
IDesignerFileManager DesignerFileManager = nil;
```

File

HTTPProd

Description

This is variable HTTPProd.DesignerFileManager.

78.6.2 HTTPProd.ScriptEnginesList Variable

Pascal

```
ScriptEnginesList: TAbstractScriptEnginesList = nil;
```

C++

```
TAbstractScriptEnginesList ScriptEnginesList = nil;
```

File

HTTPProd

Description

This is variable HTTPProd.ScriptEnginesList.

78.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
HTMLAlign (see page 15341)	This is constant HTTPProd.HTMLAlign.
HTMLVAlign (see page 15341)	This is constant HTTPProd.HTMLVAlign.

78.7.1 HTTPProd.HTMLAlign Constant

Pascal

```
HTMLAlign: array[THTMLAlign] of string = ('', ' Align="left"', ' Align="right"', '
Align="center"');
```

C++

```
array[THTMLAlign] of string HTMLAlign = ('', ' Align="left"', ' Align="right"', '
Align="center"');
```

File

HTTPProd

Description

This is constant HTTPProd.HTMLAlign.

78.7.2 HTTPProd.HTMLVAlign Constant

Pascal

```
HTMLVAlign: array[THTMLVAlign] of string = ('', ' VAlign="top"', ' VAlign="middle"', '
VAlign="bottom"', ' VAlign="baseline"');
```

C++

```
array[THTMLVAlign] of string HTMLVAlign = ('', ' VAlign="top"', ' VAlign="middle"', '
VAlign="bottom"', ' VAlign="baseline"');
```

File

HTTPProd

Description

This is constant HTTPProd.HTMLVAlign.

79 HTTPUtil Namespace

This is namespace HTTPUtil.

Topics

Name	Description
Interfaces (see page 15344)	The following table lists interfaces in this documentation.
Functions (see page 15353)	The following table lists functions in this documentation.

Functions

	Name	Description
	FirstDelimiter (see page 15353)	This is function HTTPUtil.FirstDelimiter.
	FirstDelimiter (see page 15353)	This is function HTTPUtil.FirstDelimiter.
	GetDefaultStreamLoader (see page 15354)	This is function HTTPUtil.GetDefaultStreamLoader.
	HTMLEscape (see page 15354)	This is function HTTPUtil.HTMLEscape.
	StartsWith (see page 15354)	This is function HTTPUtil.StartsWith.
	StringTokenizer (see page 15355)	This is function HTTPUtil.StringTokenizer.
	StringToStringArray (see page 15355)	This is function HTTPUtil.StringToStringArray.

Interfaces

	Name	Description
	IStreamLoader (see page 15344)	IStreamLoader is the interface for loading a WSDL document into a memory stream.
	IStringTokenizer (see page 15350)	A string tokenizer breaks a string into tokens.

79.1 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IStreamLoader (see page 15344)	IStreamLoader is the interface for loading a WSDL document into a memory stream.
	IStringTokenizer (see page 15350)	A string tokenizer breaks a string into tokens.

79.1.1 IStreamLoader Interface

IStreamLoader is the interface for loading a WSDL document into a memory stream.

Class Hierarchy

HTTPUtil.IStreamLoader

Pascal

```
IStreamLoader = interface;
```

C++

```
__interface IStreamLoader;
```

File

HTTPUtil

Description

IStreamLoader is an interface for objects that load WSDL documents. The properties specify information needed to connect to the server that hosts the WSDL document. In addition, IStreamLoader defines a Load (see page 15346) method, which loads a WSDL document into a memory stream. TWSDLItems uses the IStreamLoader interface to load its WSDL document.

Note: C++ method declarations that use IStreamLoader use the _di_IStreamLoader type instead. This type is a DelphiInterface wrapper around the IStreamLoader interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IStreamLoaders >  _di_IStreamLoader;
```

See Also

TWSDLItems (see page 25951)

TMemoryStream (see page 2902)

79.1.1.1 IStreamLoader Members

The following tables list the members exposed by IStreamLoader.

Public Methods

	Name	Description
	GetPassword (see page 15345)	Returns the value of the Password (see page 15348) property.
	GetProxy (see page 15346)	Returns the value of the Proxy (see page 15349) property.
	GetTimeout (see page 15346)	This is GetTimeout, a member of class IStreamLoader.
	GetUserName (see page 15346)	Returns the value of the UserName (see page 15349) property.
	Load (see page 15346)	Loads a WSDL document into a memory stream.
	SetPassword (see page 15347)	Sets the value of the Password (see page 15348) property.
	SetProxy (see page 15347)	Sets the value of the Proxy (see page 15349) property.
	SetTimeout (see page 15348)	This is SetTimeout, a member of class IStreamLoader.
	SetUserName (see page 15348)	Sets the value of the UserName (see page 15349) property.

Public Properties

	Name	Description
	Password (see page 15348)	Specifies the password to use when logging on if the WSDL document is on a server that requires authentication.
	Proxy (see page 15349)	Lists any proxy servers that are required to connect to a server that hosts the WSDL document.
	Timeout (see page 15349)	This is Timeout, a member of class IStreamLoader.
	UserName (see page 15349)	Specifies the user name to use when logging on if the WSDL document is on a server that requires authentication.

79.1.1.2 IStreamLoader Methods

The methods of the IStreamLoader class are listed here.

Public Methods

	Name	Description
	GetPassword (see page 15345)	Returns the value of the Password (see page 15348) property.
	GetProxy (see page 15346)	Returns the value of the Proxy (see page 15349) property.
	GetTimeout (see page 15346)	This is GetTimeout, a member of class IStreamLoader.
	GetUserName (see page 15346)	Returns the value of the UserName (see page 15349) property.
	Load (see page 15346)	Loads a WSDL document into a memory stream.
	SetPassword (see page 15347)	Sets the value of the Password (see page 15348) property.
	SetProxy (see page 15347)	Sets the value of the Proxy (see page 15349) property.
	SetTimeout (see page 15348)	This is SetTimeout, a member of class IStreamLoader.
	SetUserName (see page 15348)	Sets the value of the UserName (see page 15349) property.

79.1.1.2.1 IStreamLoader.GetPassword Method

Returns the value of the Password (see page 15348) property.

Pascal

```
function GetPassword: string;
```

C++

```
__fastcall AnsiString GetPassword();
```

Description

GetPassword is the read implementation for the Password (see page 15348) property.

See Also

Password (see page 15348)

79.1.1.2.2 IStreamLoader.GetProxy Method

Returns the value of the Proxy (see page 15349) property.

Pascal

```
function GetProxy: string;
```

C++

```
__fastcall AnsiString GetProxy();
```

Description

GetProxy is the read implementation for the Proxy (see page 15349) property.

See Also

Proxy (see page 15349)

79.1.1.2.3 IStreamLoader.GetTimeout Method

Pascal

```
function GetTimeout: Integer;
```

C++

```
__fastcall int GetTimeout();
```

Description

This is GetTimeout, a member of class IStreamLoader.

79.1.1.2.4 IStreamLoader.GetUserName Method

Returns the value of the UserName (see page 15349) property.

Pascal

```
function GetUserName: string;
```

C++

```
__fastcall AnsiString GetUserName();
```

Description

GetUserName is the read implementation for the UserName (see page 15349) property.

See Also

UserName (see page 15349)

79.1.1.2.5 IStreamLoader.Load Method

Loads a WSDL document into a memory stream.

Pascal

```
procedure Load(const WSDLFileName: WideString; Stream: TMemoryStream);
```

C++

```
__fastcall Load(const BSTR WSDLFileName, TMemoryStream Stream);
```

Description

Load reads the specified WSDL document into the specified memory stream.

WSDLFileName is the name of a WSDL document file, an XML schema file, or the URL where such a document can be found. If WSDLFileName is an URL, you may need to use the UserName (see page 15349), Password (see page 15348), and Proxy (see page 15349) properties to indicate how the stream loader can connect to the host identified in that URL.

Stream is a memory stream to which the stream loader writes the contents of the WSDL document.

See Also

UserName (see page 15349)

Password (see page 15348)

Proxy (see page 15349)

79.1.1.2.6 IStreamLoader.SetPassword Method

Sets the value of the Password (see page 15348) property.

Pascal

```
procedure SetPassword(const Password: string);
```

C++

```
__fastcall SetPassword(const AnsiString Password);
```

Description

SetPassword is the write implementation for the Password (see page 15348) property.

Password (see page 15348) is the password to use when connecting to the server that hosts the WSDL document.

See Also

Password (see page 15348)

79.1.1.2.7 IStreamLoader.SetProxy Method

Sets the value of the Proxy (see page 15349) property.

Pascal

```
procedure SetProxy(const Proxy: string);
```

C++

```
__fastcall SetProxy(const AnsiString Proxy);
```

Description

SetProxy is the write implementation for the Proxy (see page 15349) property.

Proxy (see page 15349) lists any proxy servers to use when connecting to the server that hosts the WSDL document.

See Also

Proxy (see page 15349)

79.1.1.2.8 IStreamLoader.SetTimeout Method

Pascal

```
procedure SetTimeout(ATimeOut: Integer);
```

C++

```
__fastcall SetTimeout(int ATimeOut);
```

Description

This is SetTimeout, a member of class IStreamLoader.

79.1.1.2.9 IStreamLoader.SetUserName Method

Sets the value of the UserName (see page 15349) property.

Pascal

```
procedure SetUserName(const UserName: string);
```

C++

```
__fastcall SetUserName(const AnsiString UserName);
```

Description

SetUserName is the write implementation for the UserName (see page 15349) property.

UserName (see page 15349) is the user name to use when connecting to the server that hosts the WSDL document.

See Also

UserName (see page 15349)

79.1.1.3 IStreamLoader Properties

The properties of the IStreamLoader class are listed here.

Public Properties

	Name	Description
	Password (see page 15348)	Specifies the password to use when logging on if the WSDL document is on a server that requires authentication.
	Proxy (see page 15349)	Lists any proxy servers that are required to connect to a server that hosts the WSDL document.
	Timeout (see page 15349)	This is Timeout, a member of class IStreamLoader.
	UserName (see page 15349)	Specifies the user name to use when logging on if the WSDL document is on a server that requires authentication.

79.1.1.3.1 IStreamLoader.Password Property

Specifies the password to use when logging on if the WSDL document is on a server that requires authentication.

Pascal

```
property Password: string;
```

C++

```
__property AnsiString Password;
```

Description

Use Password to provide a password that the stream loader can use if the WSDL document is on a server that requires authentication or if it needs to use a proxy server that requires authentication. Password is the password for the user specified by the UserName (see page 15349) property.

See Also

UserName (see page 15349)

Proxy (see page 15349)

79.1.1.3.2 IStreamLoader.Proxy Property

Lists any proxy servers that are required to connect to a server that hosts the WSDL document.

Pascal

```
property Proxy: string;
```

C++

```
__property AnsiString Proxy;
```

Description

Use Proxy if the stream loader can't resolve the host portion of an URL for the WSDL document locally. If the server that hosts the WSDL document is registered locally or if the WSDL document is a file on the local machine, Proxy can be a null string. Otherwise, Proxy lists the host names of any proxy servers that can forward request messages on to the intended server machine.

See Also

Password (see page 15348)

UserName (see page 15349)

Load (see page 15346)

79.1.1.3.3 IStreamLoader.Timeout Property

Pascal

```
property Timeout: integer;
```

C++

```
__property int Timeout;
```

Description

This is Timeout, a member of class IStreamLoader.

79.1.1.3.4 IStreamLoader.UserName Property

Specifies the user name to use when logging on if the WSDL document is on a server that requires authentication.

Pascal

```
property UserName: string;
```

C++

```
__property AnsiString UserName;
```

Description

Use UserName to specify the user name that the stream loader can use if the WSDL document is on a server that requires authentication or if it needs to use a proxy server that requires authentication. The password for this user name is specified by the Password (see page 15348) property.

See Also

Password (see page 15348)

Proxy (see page 15349)

79.1.2 IStringTokenizer Interface

A string tokenizer breaks a string into tokens.

Class Hierarchy

HTTPUtil.IStringTokenizer

Pascal

```
IStringTokenizer = interface;
```

C++

```
__interface IStringTokenizer;
```

File

HTTPUtil

Description

IStringTokenizer is implemented by string tokenizers, objects which iterate through the tokens in a string. To obtain a simple string tokenizer, call StringTokenizer (see page 15355).

A tokenizer is created from two strings: a source string and a delimiter string. In the source string, any characters in the delimiter string separate tokens, while any other characters are part of the tokens themselves. Thus in the following calls:

StringTokenizer (see page 15355)('This is a token/So+is+this', '/'); // Delphi example

StringTokenizer (see page 15355)("This is a token/So+is+this', '/'); // C++ example

StringTokenizer (see page 15355) creates a tokenizer that produces two tokens, consisting of the strings "This is a token" and "So+is+this".

Use the nextToken (see page 15351) method to iterate through the tokens.

See Also

StringTokenizer (see page 15355)

79.1.2.1 IStringTokenizer Members

The following tables list the members exposed by IStringTokenizer.

Public Methods

	Name	Description
	getTokenCounts (see page 15351)	Implements countTokens (see page 15352) property.

	hasMoreTokens (see page 15351)	Returns true if more tokens available.
	nextToken (see page 15351)	Returns next available token.

Public Properties

	Name	Description
R	countTokens (see page 15352)	Contains number of tokens in string.

79.1.2.2 IStringTokenizer Methods

The methods of the IStringTokenizer class are listed here.

Public Methods

	Name	Description
	getTokenCounts (see page 15351)	Implements countTokens (see page 15352) property.
	hasMoreTokens (see page 15351)	Returns true if more tokens available.
	nextToken (see page 15351)	Returns next available token.

79.1.2.2.1 IStringTokenizer.getTokenCounts Method

Implements countTokens (see page 15352) property.

Pascal

```
function getTokenCounts: integer;
```

C++

```
__fastcall int getTokenCounts();
```

Description

The getTokenCounts method returns the value of the countTokens (see page 15352) property.

See Also

counttokens (see page 15352)

79.1.2.2.2 IStringTokenizer.hasMoreTokens Method

Returns true if more tokens available.

Pascal

```
function hasMoreTokens: boolean;
```

C++

```
__fastcall boolean hasMoreTokens();
```

Description

The hasMoreTokens method returns true if unretrieved tokens remain, false otherwise.

See Also

nextToken

79.1.2.2.3 IStringTokenizer.nextToken Method

Returns next available token.

Pascal

```
function nextToken: WideString;
```

C++

```
__fastcall BSTR nextToken();
```

Description

The nextToken method returns the next available token. If no more tokens are available (hasMoreTokens (see page 15351) returns false), nextToken returns an empty string.

See Also

nextToken

79.1.2.3 IStringTokenizer Properties

The properties of the IStringTokenizer class are listed here.

Public Properties

	Name	Description
R	countTokens (see page 15352)	Contains number of tokens in string.

79.1.2.3.1 IStringTokenizer.countTokens Property

Contains number of tokens in string.

Pascal

```
property countTokens: integer;
```

C++

```
__property int countTokens;
```

Description

The countTokens property contains the number of tokens in the string.

See Also

AbsoluteIndex (see page 5252)

operator_sb

79.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FirstDelimiter (see page 15353)	This is function HTTPUtil.FirstDelimiter.
	FirstDelimiter (see page 15353)	This is function HTTPUtil.FirstDelimiter.
	GetDefaultStreamLoader (see page 15354)	This is function HTTPUtil.GetDefaultStreamLoader.
	HTMLEscape (see page 15354)	This is function HTTPUtil.HTMLEscape.
	StartsWith (see page 15354)	This is function HTTPUtil.StartsWith.
	StringTokenizer (see page 15355)	This is function HTTPUtil.StringTokenizer.
	StringToStringArray (see page 15355)	This is function HTTPUtil.StringToStringArray.

79.2.1 HTTPUtil.FirstDelimiter Function

Pascal

```
function FirstDelimiter(const delimiters: WideString; const Str: WideString): integer;
overload;
```

C++

```
int FirstDelimiter(const BSTR delimiters, const BSTR Str);
```

File

HTTPUtil

Description

This is function HTTPUtil.FirstDelimiter.

79.2.2 HTTPUtil.FirstDelimiter Function

Pascal

```
function FirstDelimiter(const delimiters: string; const Str: String): integer; overload;
```

C++

```
int FirstDelimiter(const AnsiString delimiters, const AnsiString Str);
```

File

HTTPUtil

Description

This is function HTTPUtil.FirstDelimiter.

79.2.3 HTTPUtil.GetDefaultStreamLoader Function

Pascal

```
function GetDefaultStreamLoader: IStreamLoader;
```

C++

```
IStreamLoader GetDefaultStreamLoader();
```

File

HTTPUtil

Description

This is function HTTPUtil.GetDefaultStreamLoader.

79.2.4 HTTPUtil.HTMLEscape Function

Pascal

```
function HTMLEscape(const Str: string): string;
```

C++

```
AnsiString HTMLEscape(const AnsiString Str);
```

File

HTTPUtil

Description

This is function HTTPUtil.HTMLEscape.

79.2.5 HTTPUtil.StartsWith Function

Pascal

```
function StartsWith(const str: string; const sub: string): Boolean;
```

C++

```
Boolean StartsWith(const AnsiString str, const AnsiString sub);
```

File

HTTPUtil

Description

This is function HTTPUtil.StartsWith.

79.2.6 HTTPUtil.StringTokenizer Function

Pascal

```
function StringTokenizer(const str: string; const delim: string): IStringTokenizer;
```

C++

```
IStringTokenizer StringTokenizer(const AnsiString str, const AnsiString delim);
```

File

HTTPUtil

Description

This is function HTTPUtil.StringTokenizer.

79.2.7 HTTPUtil.StringToStringArray Function

Pascal

```
function StringToStringArray(const str: string; const delim: string): TStringDynArray;
```

C++

```
TStringDynArray StringToStringArray(const AnsiString str, const AnsiString delim);
```

File

HTTPUtil

Description

This is function HTTPUtil.StringToStringArray.

80 ImgList Namespace

This is namespace ImgList.

Topics

Name	Description
Classes (see page 15358)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 15394)	The following table lists structs, records, enums in this documentation.
Types (see page 15397)	The following table lists types in this documentation.

Classes

	Name	Description
	TChangeLink (see page 15358)	The TImageList component uses TChangeLink internally to notify other objects when changes are made to the image list.
	TCustomImageList (see page 15361)	TCustomImageList is the base class for collections of same-sized images, each of which can be referred to by its index.

Structs, Records, Enums

	Name	Description
	TDrawingStyle (see page 15394)	TDrawingStyle indicates the style to use when rendering an image from an image list.
	TImageType (see page 15395)	TImageType indicates whether to use a mask when rendering an image from an image list.
	TLoadResource (see page 15395)	TLoadResource and TLoadResources (see page 15397) determine how a resource is loaded into an image list.
	TResType (see page 15396)	TResType identifies a type of graphical resource to an image list.

Types

Name	Description
TImageIndex (see page 15397)	This is type ImgList.TImageIndex.
TLoadResources (see page 15397)	This is type ImgList.TLoadResources.
TOverlay (see page 15397)	This is type ImgList.TOverlay.

80.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TChangeLink (see page 15358)	The TImageList component uses TChangeLink internally to notify other objects when changes are made to the image list.
	TCustomImageList (see page 15361)	TCustomImageList is the base class for collections of same-sized images, each of which can be referred to by its index.

80.1.1 TChangeLink Class

The TImageList component uses TChangeLink internally to notify other objects when changes are made to the image list.

Class Hierarchy

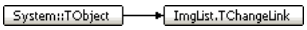

Pascal

```
TChangeLink = class(TObject);
```

C++

```
class TChangeLink : public TObject;
```

File

ImgList

Description

A TImageList keeps a list of its TChangeLink objects. Each is associated with a particular component, such as a tree view or list view control. A component can register its change link with an image list by calling the image list's RegisterChanges method. Before destruction, the component should then call the UnRegisterChanges method.

Component writers can use the change link object to receive notification when changes are made to an image list that a control uses. Use the OnChange (see page 15361) event of the change link to execute some code when the image list changes. For example, the tree view control uses a change link object to know when its image list has changed.

See Also

TImageList (see page 6078)

TTreeView (see page 5293)

TListView (see page 4285)

TCustomImageList (see page 15361)

80.1.1.1 TChangeLink Members

The following tables list the members exposed by TChangeLink.

Public Events

	Name	Description
	OnChange (see page 15361)	Occurs when there's a change in any of the images stored in the associated image list.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	Change (see page 15360)	Generates an OnChange (see page 15361) event.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 15359)	Destroys the change link object and releases the memory allocated to it.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Sender (see page 15360)	Specifies which image list component notifies the change link object of changes.

80.1.1.2 TChangeLink.Destroy Destructor

Destroys the change link object and releases the memory allocated to it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TChangeLink();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the change link is not nil before calling Destroy.

Before freeing the memory for the change link, Destroy first calls the image list's UnRegisterChanges method to unregister itself from receiving notification when the image list changes.

See Also

Free (see page 22300)

UnRegisterChanges (see page 15386)

80.1.1.3 TChangeLink Methods

The methods of the TChangeLink class are listed here.

Public Methods

	Name	Description
	Change (see page 15360)	Generates an OnChange (see page 15361) event.

80.1.1.3.1 TChangeLink.Change Method

Generates an OnChange (see page 15361) event.

Pascal

```
procedure Change; dynamic;
```

C++

```
__fastcall Change();
```

Description

Use the Change method to trigger the OnChange (see page 15361) event handler of the change link object.

See Also

OnChange (see page 15361)

80.1.1.4 TChangeLink Properties

The properties of the TChangeLink class are listed here.

Public Properties

	Name	Description
	Sender (see page 15360)	Specifies which image list component notifies the change link object of changes.

80.1.1.4.1 TChangeLink.Sender Property

Specifies which image list component notifies the change link object of changes.

Pascal

```
property Sender: TCustomImageList;
```

C++

```
__property TCustomImageList Sender;
```

Description

Use Sender to access the image list that has changed. The TImageList automatically sets this property when a call to RegisterChanges is made.

See Also

RegisterChanges (see page 15383)

80.1.1.5 TChangeLink Events

The events of the TChangeLink class are listed here.

Public Events

	Name	Description
	OnChange (see page 15361)	Occurs when there's a change in any of the images stored in the associated image list.

80.1.1.5.1 TChangeLink.OnChange Event

Occurs when there's a change in any of the images stored in the associated image list.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Use the OnChange event to notify a control when the image list that it's using changes. Specify any special processing to occur at that time in the OnChange event handler.

See Also

Change (see page 15360)

80.1.2 TCustomImageList Class

TCustomImageList is the base class for collections of same-sized images, each of which can be referred to by its index.

Class Hierarchy

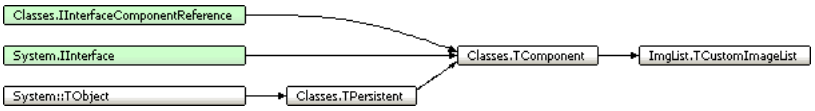

Pascal

```
TCustomImageList = class(TComponent);
```

C++

```
class TCustomImageList : public TComponent;
```

File

ImgList

Description

Image lists manage large sets of icons or bitmaps. All images in an image list are contained in a single, wide bitmap in screen device format. An image list may also include a monochrome bitmap that contains masks that are combined with the images to produce a transparent effect such as is common in icons.

See Also

TPicture (see page 14584)

TImage (see page 12777)

80.1.2.1 TCustomImageList Members

The following tables list the members exposed by TCustomImageList.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 15370)	Copies the image list's properties to another image list.
	Change (see page 15371)	Generates an OnChange (see page 15393) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 15371)	Allows bitmap data to be saved and loaded.
V	DoDraw (see page 15372)	Draws the specified image onto the provided Canvas in the indicated style.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetImages (see page 15378)	Returns a specified image and its mask as bitmaps.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleNeeded (see page 15381)	This is HandleNeeded, a member of class TCustomImageList.
V	Initialize (see page 15381)	Initializes a newly allocated image list.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadData (see page 15383)	This is ReadData, a member of class TCustomImageList.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.

	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteData (see page 15386)	This is WriteData, a member of class TCustomImageList.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnChange (see page 15393)	Occurs when the image list changes.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Add (see page 15368)	Adds an Image and its Mask (see page 15797) to the image list object.
	AddIcon (see page 15368)	Adds an icon to the image list object.
	AddImage (see page 15368)	Adds single image from another image list.
	AddImages (see page 15369)	Adds all images from another image list.
	AddMasked (see page 15369)	Adds an image to the image list along with a color which is used to generate the transparent mask.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 15370)	Copies the information from one image list to another.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 15371)	Clears all images from the image list.
V	Create (see page 15365)	Creates an empty image list.
	CreateSize (see page 15366)	Creates an empty image list of a specified size.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 15372)	Deletes a particular image from the image list.
V	Destroy (see page 15366)	Destroys the image list and releases the memory allocated to it.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.

	Draw (see page 15373)	This is the overview for the Draw method overload.
	DrawOverlay (see page 15374)	This is the overview for the DrawOverlay method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FileLoad (see page 15375)	Loads a resourced image into the image list.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetBitmap (see page 15376)	Retrieves a specified image as a bitmap.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetHotSpot (see page 15376)	Returns the hot spot location associated with the current drag image.
	GetIcon (see page 15377)	This is the overview for the GetIcon method overload.
	GetImageBitmap (see page 15378)	Returns a handle to a bitmap containing all of the images in the image list.
	GetInstRes (see page 15378)	This is the overview for the GetInstRes method overload.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaskBitmap (see page 15379)	Returns a handle to a bitmap containing all masks for images in the image list.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetResource (see page 15380)	Loads the specified bitmap, cursor, or icon resource into the image list.
	HandleAllocated (see page 15380)	Indicates whether a window handle exists for the image list.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 15381)	Inserts a bitmap and its mask into the image list following a specified image.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertIcon (see page 15382)	Inserts an icon at a specified position in the image list.
	InsertMasked (see page 15382)	Inserts a bitmap at a specified position, creating the mask from a specified transparent color.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	Move (see page 15382)	Moves a specified image to another position.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Overlay (see page 15383)	Identifies an image in the list as an image that can be used as an overlay mask.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 15383)	Registers an object to be notified when changes occur in the image list.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Replace (see page 15384)	Replaces an image in the image list with a new image and mask.
	ReplaceIcon (see page 15384)	Replaces an image in the image list with an icon.

	ReplaceMasked (see page 15384)	Replaces an image in the image list with a new image and automatically-generated mask.
	ResInstLoad (see page 15385)	Loads a resource into the image list.
	ResourceLoad (see page 15385)	Loads a resource into the image list.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 15386)	Removes an object from the list of objects to be notified when the image list changes.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	AllocBy (see page 15387)	Sets the number of images by which the image list grows when it needs to make room for new images.
	BkColor (see page 15387)	Determines which background color to use when drawing an image.
	BlendColor (see page 15387)	Determines which foreground color to use when drawing an image.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Count (see page 15388)	Indicates the number of images in the image list.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DrawingStyle (see page 15390)	Specifies the style to be used when the image list is drawing an image.
	Handle (see page 15390)	Designates the handle for the image list.
	Height (see page 15391)	Specifies the height of images within the image list.
	ImageType (see page 15391)	Determines whether the image list uses the associated mask when drawing.
	Masked (see page 15392)	Specifies whether the image list includes masks that can be combined with the images.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ShareImages (see page 15392)	Specifies whether the image list destroys its handle when the image list is destroyed.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Width (see page 15392)	Specifies the width of all images within the image list.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

80.1.2.2 TCustomImageList.Create Constructor

Creates an empty image list.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomImageList(TComponent * AOwner);
```

Description

Call Create to instantiate an image list at runtime. ImageLists placed on forms or data modules at design time are created automatically. Specify the owner of the new imagelist using the AOwner parameter.

The new image list contains no images and has a default size of 16 by 16 pixels.

See Also

CreateSize (see page 15366)

80.1.2.3 TCustomImageList.CreateSize Constructor

Creates an empty image list of a specified size.

Pascal

```
constructor CreateSize(AWidth: Integer; AHeight: Integer);
```

C++

```
__fastcall TCustomImageList(int AWidth, int AHeight);
```

Description

Use CreateSize to create an image list object at runtime with an image size other than the default of 16 x 16 pixels. The initial size of the image list is determined by the AHeight and AWidth parameters. The height and width can be altered after the image list has been created using the Width (see page 15392) and Height (see page 15391) properties.

See Also

Create (see page 15365)

Height (see page 15391)

Width (see page 15392)

80.1.2.4 TCustomImageList.Destroy Destructor

Destroys the image list and releases the memory allocated to it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomImageList();
```

Description

Destroy first calls UnRegisterChanges (see page 15386) to unregister any objects registered to receive notification when the image list changed, and frees all helper objects associated with the image list.

Destroy seldom needs to be called. However, when an image list is constructed without an owner by the Create (see page 15365) or CreateSize (see page 15366) method, call Free to release memory and dispose of the object. Free checks to see if the pointer is nil before calling Destroy.

See Also

Free (see page 22300)

UnRegisterChanges (see page 15386)

80.1.2.5 TCustomImageList Methods

The methods of the TCustomImageList class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 15370)	Copies the image list's properties to another image list.
	Change (see page 15371)	Generates an OnChange (see page 15393) event.
	DefineProperties (see page 15371)	Allows bitmap data to be saved and loaded.
	DoDraw (see page 15372)	Draws the specified image onto the provided Canvas in the indicated style.
	GetImages (see page 15378)	Returns a specified image and its mask as bitmaps.
	HandleNeeded (see page 15381)	This is HandleNeeded, a member of class TCustomImageList.
	Initialize (see page 15381)	Initializes a newly allocated image list.
	ReadData (see page 15383)	This is ReadData, a member of class TCustomImageList.
	WriteData (see page 15386)	This is WriteData, a member of class TCustomImageList.

Public Methods

	Name	Description
	Add (see page 15368)	Adds an Image and its Mask (see page 15797) to the image list object.
	AddIcon (see page 15368)	Adds an icon to the image list object.
	AddImage (see page 15368)	Adds single image from another image list.
	AddImages (see page 15369)	Adds all images from another image list.
	AddMasked (see page 15369)	Adds an image to the image list along with a color which is used to generate the transparent mask.
	Assign (see page 15370)	Copies the information from one image list to another.
	Clear (see page 15371)	Clears all images from the image list.
	Delete (see page 15372)	Deletes a particular image from the image list.
	Draw (see page 15373)	This is the overview for the Draw method overload.
	DrawOverlay (see page 15374)	This is the overview for the DrawOverlay method overload.
	FileLoad (see page 15375)	Loads a resourced image into the image list.
	GetBitmap (see page 15376)	Retrieves a specified image as a bitmap.
	GetHotSpot (see page 15376)	Returns the hot spot location associated with the current drag image.
	GetIcon (see page 15377)	This is the overview for the GetIcon method overload.
	GetImageBitmap (see page 15378)	Returns a handle to a bitmap containing all of the images in the image list.
	GetInstRes (see page 15378)	This is the overview for the GetInstRes method overload.
	GetMaskBitmap (see page 15379)	Returns a handle to a bitmap containing all masks for images in the image list.
	GetResource (see page 15380)	Loads the specified bitmap, cursor, or icon resource into the image list.
	HandleAllocated (see page 15380)	Indicates whether a window handle exists for the image list.
	Insert (see page 15381)	Inserts a bitmap and its mask into the image list following a specified image.
	InsertIcon (see page 15382)	Inserts an icon at a specified position in the image list.
	InsertMasked (see page 15382)	Inserts a bitmap at a specified position, creating the mask from a specified transparent color.
	Move (see page 15382)	Moves a specified image to another position.
	Overlay (see page 15383)	Identifies an image in the list as an image that can be used as an overlay mask.
	RegisterChanges (see page 15383)	Registers an object to be notified when changes occur in the image list.
	Replace (see page 15384)	Replaces an image in the image list with a new image and mask.
	ReplaceIcon (see page 15384)	Replaces an image in the image list with an icon.
	ReplaceMasked (see page 15384)	Replaces an image in the image list with a new image and automatically-generated mask.
	ResInstLoad (see page 15385)	Loads a resource into the image list.
	ResourceLoad (see page 15385)	Loads a resource into the image list.
	UnRegisterChanges (see page 15386)	Removes an object from the list of objects to be notified when the image list changes.

80.1.2.5.1 TCustomImageList.Add Method

Adds an Image and its Mask (see page 15797) to the image list object.

Pascal

```
function Add(Image: TBitmap; Mask: TBitmap): Integer;
```

C++

```
__fastcall int Add(TBitmap Image, TBitmap Mask);
```

Description

Use the Add method to add images with their masks to an image list. It returns the index of the added image, or -1 if it fails. When the Masked (see page 15392) property of the image list is false, the second bitmap is ignored; it can be set to nil (Delphi) or NULL (C++).

See Also

AddIcon (see page 15368)

AddImages (see page 15369)

AddMasked (see page 15369)

Masked (see page 15392)

80.1.2.5.2 TCustomImageList.AddIcon Method

Adds an icon to the image list object.

Pascal

```
function AddIcon(Image: TIcon): Integer;
```

C++

```
__fastcall int AddIcon(TIcon Image);
```

Description

Use the AddIcon method to add the icon specified by the Image parameter to an image list. AddIcon returns the index of the added image, or -1 if it fails.

See Also

Add (see page 15368)

AddImages (see page 15369)

AddMasked (see page 15369)

80.1.2.5.3 TCustomImageList.AddImage Method

Adds single image from another image list.

Pascal

```
function AddImage(Value: TCustomImageList; Index: Integer): Integer;
```

C++

```
__fastcall int AddImage(TCustomImageList Value, int Index);
```

Description

Use AddImage to add a single image from another image list. Values specifies the source image list. Index indicates the specific image, where images are numbered from 0.

AddImage returns the new size of the image list.

See Also

Add (see page 15368)

AddIcon (see page 15368)

AddMasked (see page 15369)

AddImages (see page 15369)

80.1.2.5.4 TCustomImageList.AddImages Method

Adds all images from another image list.

Pascal

```
procedure AddImages(Value: TCustomImageList);
```

C++

```
__fastcall AddImages(TCustomImageList Value);
```

Description

Use AddImages to add all the images from another image list. Value specifies the source image list.

See Also

Add (see page 15368)

AddIcon (see page 15368)

AddMasked (see page 15369)

AddImage (see page 15368)

80.1.2.5.5 TCustomImageList.AddMasked Method

Adds an image to the image list along with a color which is used to generate the transparent mask.

Pascal

```
function AddMasked(Image: TBitmap; MaskColor: TColor): Integer;
```

C++

```
__fastcall int AddMasked(TBitmap Image, TColor MaskColor);
```

Description

AddMasked is similar to Add (see page 15368) except that a mask bitmap is not specified. Instead, a color that can be used to automatically generate the masks is specified. Each pixel of the specified color in the image bitmap is changed to black, and the corresponding bit in the mask is set to one. As a result, any pixel in the image that matches the specified color is transparent when the image is drawn. AddMasked returns -1 if it fails, 0 if it's successful.

See Also

Add (see page 15368)

AddIcon (see page 15368)

AddImages (see page 15369)

Masked (see page 15392)

80.1.2.5.6 TCustomImageList.Assign Method

Copies the information from one image list to another.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use this method to copy another object's information to the image list.

If Source is another image list (TCustomImageList (see page 15361) descendant), Assign discards any current image information and replaces it with the information from Source.

If Source is nil (Delphi) or NULL (C++), Assign clears the image list.

If Source is any other type of object, Assign calls its inherited method, which can copy from any source object that implements a way to copy to an image list in its AssignTo (see page 15370) method.

See Also

AddImages (see page 15369)

Add (see page 15368)

AssignTo (see page 2931)

80.1.2.5.7 TCustomImageList.AssignTo Method

Copies the image list's properties to another image list.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Applications do not call the protected AssignTo method. Instead, call the public Assign (see page 15370) method of the destination image list. AssignTo is called when the image list is passed as the Source parameter of another object's Assign (see page 15370) method and that Assign (see page 15370) method can't copy information to the image list. The object whose Assign (see page 15370) method was called is passed as the Dest parameter. AssignTo gives the image list an opportunity to copy properties to objects that were not created with the image list in mind.

See Also

AddImages (see page 15369)

Add (see page 15368)

Assign (see page 15370)

80.1.2.5.8 TCustomImageList.Change Method

Generates an OnChange (see page 15393) event.

Pascal

```
procedure Change; dynamic;
```

C++

```
__fastcall Change();
```

Description

Change calls the OnChange (see page 15393) event handler if it is assigned. Override Change to block this event or provide additional responses to changes in the image list.

See Also

RegisterChanges (see page 15383)

OnChange (see page 15393)

80.1.2.5.9 TCustomImageList.Clear Method

Clears all images from the image list.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Use the Clear method to remove all images from the image list. Use the Delete (see page 15372) method to delete one image from the image list.

See Also

Delete (see page 15372)

80.1.2.5.10 TCustomImageList.DefineProperties Method

Allows bitmap data to be saved and loaded.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

DefineProperties allows the image list to create pseudo properties that are loaded and saved with the image list. TCustomImageList (see page 15361) overrides the inherited method to save or load the bitmap image in addition to the other published properties.

See Also

DefineBinaryProperty (see page 2837)

DefineProperties (see page 2931)

80.1.2.5.11 TCustomImageList.Delete Method

Deletes a particular image from the image list.

Pascal

```
procedure Delete(Index: Integer);
```

C++

```
__fastcall Delete(int Index);
```

Description

Use the Delete method to delete an image specified by Index in the image list. Use the Clear (see page 15371) method to delete all images in the image list.

See Also

Clear (see page 15371)

80.1.2.5.12 TCustomImageList.DoDraw Method

Draws the specified image onto the provided Canvas in the indicated style.

Pascal

```
procedure DoDraw(Index: Integer; Canvas: TCanvas; X: Integer; Y: Integer; Style: Cardinal;
Enabled: Boolean = True); virtual;
```

C++

```
virtual __fastcall DoDraw(int Index, TCanvas Canvas, int X, int Y, unsigned Style, Boolean
Enabled = True);
```

Description

Applications can't call the protected DoDraw method. It provides the underlying implementation of the public Draw (see page 15373) method.

Index specifies an image to draw.

Canvas indicates the drawing surface on which the image is rendered.

X and Y specify the top-left corner of the image on Canvas.

Style encodes how the image blends color or uses a mask (the DrawingStyle (see page 15390) and ImageType (see page 15391)).

Enabled indicates whether the image appears as a gray-mapped image. If Enabled is false, DoDraw renders a gray-mapped version of the image.

See Also

Draw (see page 15373)

DrawingStyle (see page 15390)

ImageType (see page 15391)

80.1.2.5.13 Draw Method

This is the overview for the Draw method overload.

Overload List

	Name	Description
	TCustomImageList.Draw (TCanvas, Integer, Integer, Integer, Boolean) (see page 15373)	Draws the image specified by the Index parameter onto the provided Canvas.
	TCustomImageList.Draw (TCanvas, Integer, Integer, Integer, TDrawingStyle, TImageType, Boolean) (see page 15373)	Draws the image specified by the Index parameter onto the provided Canvas.

80.1.2.5.13.1 TCustomImageList.Draw Method (TCanvas, Integer, Integer, Integer, Boolean)

Draws the image specified by the Index parameter onto the provided Canvas.

Pascal

```
procedure Draw(Canvas: TCanvas; X: Integer; Y: Integer; Index: Integer; Enabled: Boolean =
True); overload;
procedure Draw(Canvas: TCanvas; X: Integer; Y: Integer; Index: Integer; ADrawingStyle:
TDrawingStyle; AImageType: TImageType; Enabled: Boolean = True); overload;
```

C++

```
__fastcall Draw(TCanvas Canvas, int X, int Y, int Index, Boolean Enabled = True);
__fastcall Draw(TCanvas Canvas, int X, int Y, int Index, TDrawingStyle ADrawingStyle,
TImageType AImageType, Boolean Enabled = True);
```

Description

Use the Draw method to draw one of the images in the image list onto a specified canvas.

Canvas is the drawing surface on which to render the image.

X and Y specify the location where the top-left corner should appear on Canvas.

Index indicates which image to draw, where 0 specifies the first image, 1 specifies the second image, and so on.

ADrawingStyle indicates how the color of the image may be altered. If this parameter is not specified, Draw uses the value of the DrawingStyle (see page 15390) property.

AImageType indicates whether to draw the image or its associated mask. If this parameter is not specified, Draw uses the value of the ImageType (see page 15391) property.

Enabled indicates whether the image appears as a gray-mapped image. If Enabled is false, Draw renders a gray-mapped version of the image. Because Enabled has a default value of true, this parameter can be omitted if the image should not be grayed.

C++ Examples:

```
/*
The following code is the OnDrawPanel event handler of a
status bar.  It draws each panel of the status bar, adding
text and icons stored in ImageList1. The image list contains
as many icons as there are header sections.  This example
requires a populated image list and a statusbar with several
panels added to the Panels property.  Select each panel and
set the Style property to psOwnerDraw.
*/
void __fastcall TForm1::StatusBar1DrawPanel(TStatusBar *StatusBar,
```

```
        TStatusPanel *Panel, const TRect &Rect)
{
  TCanvas *canvas = StatusBar->Canvas;
  canvas->Brush->Color = clRed;
  canvas->FillRect(Rect);
  canvas->Font->Color = clYellow;
  ImageList1->Draw(canvas,Rect.Left,Rect.Top, Panel->Index, true);
  canvas->TextOut(Rect.left + 30, Rect.top + 2, "Panel" + IntToStr(Panel->Index));
}
```

Delphi Examples:

```
{
The following code is the OnDrawPanel event handler of a
status bar.  It draws each panel of the status bar, adding
text and icons stored in ImageList1. The image list contains
as many icons as there are header sections.  This example
requires a populated image list and a statusbar with several
panels added to the Panels property.  Select each panel and
set the Style property to psOwnerDraw.
}
procedure TForm1.StatusBar1DrawPanel(StatusBar: TStatusBar;
  Panel: TStatusPanel; const Rect: TRect);
begin
with StatusBar1.Canvas do
  begin
    Brush.Color := clRed;
    FillRect(Rect);
    Font.Color := clYellow;
    ImageList1.Draw(StatusBar1.Canvas,Rect.Left,Rect.Top,Panel.Index);
    TextOut(Rect.left + 30, Rect.top + 2, 'Panel' + IntToStr(Panel.Index));
  end;
end;
```

See Also

DrawingStyle (see page 15390)

DrawOverlay (see page 15375)

GetImages (see page 15378)

ImageType (see page 15391)

80.1.2.5.14 DrawOverlay Method

This is the overview for the DrawOverlay method overload.

Overload List

	Name	Description
	TCustomImageList.DrawOverlay (TCanvas, Integer, Integer, Integer, TOverlay, Boolean) (see page 15375)	Draws an image and an overlay onto the provided Canvas.
	TCustomImageList.DrawOverlay (TCanvas, Integer, Integer, Integer, TOverlay, TDrawingStyle, TImageType, Boolean) (see page 15375)	Draws an image and an overlay onto the provided Canvas.

80.1.2.5.14.1 TCustomImageList.DrawOverlay Method (TCanvas, Integer, Integer, Integer, TOverlay, Boolean)

Draws an image and an overlay onto the provided Canvas.

Pascal

```
procedure DrawOverlay(Canvas: TCanvas; X: Integer; Y: Integer; ImageIndex: Integer; Overlay:
TOverlay; Enabled: Boolean = True); overload;
procedure DrawOverlay(Canvas: TCanvas; X: Integer; Y: Integer; ImageIndex: Integer; Overlay:
TOverlay; ADrawingStyle: TDrawingStyle; AImageType: TImageType; Enabled: Boolean = True);
overload;
```

C++

```
__fastcall DrawOverlay(TCanvas Canvas, int X, int Y, int ImageIndex, TOverlay Overlay, Boolean
Enabled = True);
__fastcall DrawOverlay(TCanvas Canvas, int X, int Y, int ImageIndex, TOverlay Overlay,
TDrawingStyle ADrawingStyle, TImageType AImageType, Boolean Enabled = True);
```

Description

An overlay mask is an image drawn transparently over another image. Any image can be used as an overlay mask. To draw an overlay mask over an image use the DrawOverlay method with the index of the overlay mask.

Canvas is the drawing surface on which to draw the overlay.

X and Y specify the location where the top-left corner should appear on Canvas.

ImageIndex indicates which image to draw, where 0 specifies the first image, 1 specifies the second image, and so on.

Overlay (see page 15383) specifies the index of the overlay mask to use. This index is assigned to the image that is used as an overlay mask by calling the Overlay (see page 15383) method.

ADrawingStyle indicates how the color of the image may be altered. If this parameter is not specified, Draw (see page 15373) uses the value of the DrawingStyle (see page 15390) property.

AImageType indicates whether to draw the image or its associated mask. If this parameter is not specified, Draw (see page 15373) uses the value of the ImageType (see page 15391) property.

Enabled indicates whether the image appears as a gray-mapped image. If Enabled is false, DoDraw (see page 15372) renders a gray-mapped version of the image. Because Enabled has a default value of true, this parameter can be omitted if the image should not be grayed.

See Also

Overlay (see page 15383)

Draw (see page 15373)

SetDragImage (see page 6029)

80.1.2.5.15 TCustomImageList.FileLoad Method

Loads a resourced image into the image list.

Pascal

```
function FileLoad(ResType: TResType; const Name: string; MaskColor: TColor): Boolean;
```

C++

```
__fastcall Boolean FileLoad(TResType ResType, const AnsiString Name, TColor MaskColor);
```

Description

Use the FileLoad method to load a bitmap, cursor, or icon from a resource file. Use ResType to specify the type of resource. Use Name to identify the resource by name. Use MaskColor to set the transparent color for the images. FileLoad returns true if successful.

Note: Within a package, FileLoad returns an error. GetInstRes (see page 15379) must be used to load a resource from within a package.

See Also

GetResource (see page 15380)

ResourceLoad (see page 15385)

GetInstRes (see page 15379)

80.1.2.5.16 TCustomImageList.GetBitmap Method

Retrieves a specified image as a bitmap.

Pascal

```
function GetBitmap(Index: Integer; Image: TBitmap): Boolean;
```

C++

```
__fastcall Boolean GetBitmap(int Index, TBitmap Image);
```

Description

Use the GetBitmap method to obtain a particular image in the image list as a bitmap object. Index denotes the index of the image. GetBitmap assigns the image to the TBitmap object passed in as the Image parameter.

Delphi Examples:

```
{
This example assigns a bitmap contained in an ImageList to
an Image component's Picture's Bitmap property and displays
the bitmap.  TImage Stretch will fit the bitmap to the image
size.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   ImageList1.GetBitmap(2,Image1.Picture.Bitmap);
   Image1.Stretch := True;
end;
```

See Also

GetMaskBitmap (see page 15379)

80.1.2.5.17 TCustomImageList.GetHotSpot Method

Returns the hot spot location associated with the current drag image.

Pascal

```
function GetHotSpot: TPoint; virtual;
```

C++

```
virtual __fastcall TPoint GetHotSpot();
```

Description

Use the GetHotSpot method to obtain the position of the hot spot of the current drag image.

See Also

DragMove (see page 6027)

SetDragImage (see page 6029)

80.1.2.5.18 GetIcon Method

This is the overview for the GetIcon method overload.

Overload List

	Name	Description
	TCustomImageList.GetIcon (Integer, TIcon) (see page 15377)	Returns the image specified by Index as an icon into the Image parameter.
	TCustomImageList.GetIcon (Integer, TIcon, TDrawingStyle, TImageType) (see page 15377)	Returns the image specified by Index as an icon into the Image parameter.

80.1.2.5.18.1 TCustomImageList.GetIcon Method (Integer, TIcon)

Returns the image specified by Index as an icon into the Image parameter.

Pascal

```
procedure GetIcon(Index: Integer; Image: TIcon); overload;
procedure GetIcon(Index: Integer; Image: TIcon; ADrawingStyle: TDrawingStyle; AImageType:
TImageType); overload;
```

C++

```
__fastcall GetIcon(int Index, TIcon Image);
__fastcall GetIcon(int Index, TIcon Image, TDrawingStyle ADrawingStyle, TImageType AImageType);
```

Description

Call GetIcon to obtain a particular image in the image list as an icon.

Index identifies the desired image, where 0 specifies the first image, 1 specifies the second image, and so on.

Image is a TIcon object that receives the new image.

ADrawingStyle specifies the amount of color that is blended with the image to achieve various affects. When GetIcon is called using the version that does not take an ADrawingStyle parameter, it uses the value of the DrawingStyle (see page 15390) property.

AImageType specifies whether the image is rendered using a mask. When GetIcon is called using the version that does not take an AImageType parameter, it uses the value of the ImageType (see page 15391) property.

See Also

GetImageBitmap (see page 15378)

DrawingStyle (see page 15390)

ImageType (see page 15391)

80.1.2.5.19 TCustomImageList.GetImageBitmap Method

Returns a handle to a bitmap containing all of the images in the image list.

Pascal

```
function GetImageBitmap: HBITMAP;
```

C++

```
__fastcall HBITMAP GetImageBitmap();
```

Description

Use the GetImageBitmap method to obtain a Windows handle for the bitmap that contains all images in the image list. GetImageBitmap returns 0 if unsuccessful.

See Also

GetImages (see page 15378)

GetBitmap (see page 15376)

80.1.2.5.20 TCustomImageList.GetImages Method

Returns a specified image and its mask as bitmaps.

Pascal

```
procedure GetImages(Index: Integer; Image: TBitmap; Mask: TBitmap);
```

C++

```
__fastcall GetImages(int Index, TBitmap Image, TBitmap Mask);
```

Description

Use GetImages to obtain the image (and its mask) identified by the Index parameter.

Index specifies the image (and mask) to return, where 0 identifies the first image, 1 the second image, and so on.

Image is a TBitmap object that receives the image of the bitmap.

Mask (see page 15797) is a TBitmap object that receives the image of the mask.

See Also

GetImageBitmap (see page 15378)

GetMaskBitmap (see page 15379)

GetBitmap (see page 15376)

80.1.2.5.21 GetInstRes Method

This is the overview for the GetInstRes method overload.

Overload List

	Name	Description
	TCustomImageList.GetInstRes (THandle, TResType, DWORD, Integer, TLoadResources, TColor) (see page 15379)	Loads a specified bitmap, cursor, or icon resource into the image list.

	TCustomImageList.GetInstRes (THandle, TResType, string, Integer, TLoadResources, TColor) (see page 15379)	Loads a specified bitmap, cursor, or icon resource into the image list.

80.1.2.5.21.1 TCustomImageList.GetInstRes Method (THandle, TResType, DWORD, Integer, TLoadResources, TColor)

Loads a specified bitmap, cursor, or icon resource into the image list.

Pascal

```
function GetInstRes(Instance: THandle; ResType: TResType; const Name: string; Width: Integer;
LoadFlags: TLoadResources; MaskColor: TColor): Boolean; overload;
function GetInstRes(Instance: THandle; ResType: TResType; ResID: DWORD; Width: Integer;
LoadFlags: TLoadResources; MaskColor: TColor): Boolean; overload;
```

C++

```
__fastcall Boolean GetInstRes(THandle Instance, TResType ResType, const AnsiString Name, int
Width, TLoadResources LoadFlags, TColor MaskColor);
__fastcall Boolean GetInstRes(THandle Instance, TResType ResType, DWORD ResID, int Width,
TLoadResources LoadFlags, TColor MaskColor);
```

Description

Use the GetInstRes method to load bitmap, cursor, or icon resource into an image list.

Instance is the instance handle of the resource module.

ResType indicates the type of resource (rtBitmap, rtCursor or rtIcon).

Name or ResID identifies the resource by name or ID, respectively.

Width (see page 15392) specifies the width, in pixels, of the image.

LoadFlags specifies how to load the image.

MaskColor sets the transparent color for the image.

Note: GetInstRes must be used to load a resource from within a package. Within a package, GetResource (see page 15380) or FileLoad (see page 15375) returns an error.

See Also

ResInstLoad (see page 15385)

ResourceLoad (see page 15385)

FileLoad (see page 15375)

GetResource (see page 15380)

80.1.2.5.22 TCustomImageList.GetMaskBitmap Method

Returns a handle to a bitmap containing all masks for images in the image list.

Pascal

```
function GetMaskBitmap: HBITMAP;
```

C++

```
__fastcall HBITMAP GetMaskBitmap();
```

Description

Use the GetMaskBitmap to obtain a bitmap that contains all masks in the image list. GetMaskBitmap returns 0 if unsuccessful.

See Also

GetImageBitmap (see page 15378)

GetImages (see page 15378)

80.1.2.5.23 TCustomImageList.GetResource Method

Loads the specified bitmap, cursor, or icon resource into the image list.

Pascal

```
function GetResource(ResType: TResType; const Name: string; Width: Integer; LoadFlags:
TLoadResources; MaskColor: TColor): Boolean;
```

C++

```
__fastcall Boolean GetResource(TResType ResType, const AnsiString Name, int Width,
TLoadResources LoadFlags, TColor MaskColor);
```

Description

Use the GetResource method to load bitmap, cursor, or icon resource into an image list.

Note: Within a package, GetResource returns an error. GetInstRes (see page 15379) must be used to load a resource from within a package.

ResType specifies the type of graphical resource that is to be loaded into the image list and can be rtBitmap, rtCursor or rtIcon.

Name identifies the resource by name.

Width (see page 15392) specifies the width, in pixels, of the image.

LoadFlags specifies how to load the image.

MaskColor sets the transparent color for the images in the file.

See Also

ResourceLoad (see page 15385)

FileLoad (see page 15375)

GetInstRes (see page 15379)

80.1.2.5.24 TCustomImageList.HandleAllocated Method

Indicates whether a window handle exists for the image list.

Pascal

```
function HandleAllocated: Boolean;
```

C++

```
__fastcall Boolean HandleAllocated();
```

Description

Use HandleAllocated to determine if a handle exists for the image list. HandleAllocated returns true if the Windows image list has been created and a handle is available, false otherwise. HandleAllocated doesn't create a handle if it doesn't already exist.

Note: Getting the Handle (see page 15390) property of creates the Windows image list automatically if it didn't previously

exist, and returns the resulting handle.

See Also

Handle (see page 15390)

80.1.2.5.25 TCustomImageList.HandleNeeded Method

Pascal

```
procedure HandleNeeded;
```

C++

```
__fastcall HandleNeeded();
```

Description

This is HandleNeeded, a member of class TCustomImageList.

80.1.2.5.26 TCustomImageList.Initialize Method

Initializes a newly allocated image list.

Pascal

```
procedure Initialize; virtual;
```

C++

```
virtual __fastcall Initialize();
```

Description

Initialize is called from the constructor of the image list. It sets numerous image list properties to their default values and creates instances of helper objects used internally.

See Also

Create (see page 15365)

CreateSize (see page 15366)

80.1.2.5.27 TCustomImageList.Insert Method

Inserts a bitmap and its mask into the image list following a specified image.

Pascal

```
procedure Insert(Index: Integer; Image: TBitmap; Mask: TBitmap);
```

C++

```
__fastcall Insert(int Index, TBitmap Image, TBitmap Mask);
```

Description

Use the Insert method to insert a bitmap and its mask into the image list. The image is inserted following the image specified by Index.

See Also

Add (see page 15368)

AddMasked (see page 15369)

80.1.2.5.28 TCustomImageList.InsertIcon Method

Inserts an icon at a specified position in the image list.

Pascal

```
procedure InsertIcon(Index: Integer; Image: TIcon);
```

C++

```
__fastcall InsertIcon(int Index, TIcon Image);
```

Description

Use the InsertIcon method to insert an icon into the image list. The icon is inserted following the image specified by Index.

See Also

AddIcon (see page 15368)

80.1.2.5.29 TCustomImageList.InsertMasked Method

Inserts a bitmap at a specified position, creating the mask from a specified transparent color.

Pascal

```
procedure InsertMasked(Index: Integer; Image: TBitmap; MaskColor: TColor);
```

C++

```
__fastcall InsertMasked(int Index, TBitmap Image, TColor MaskColor);
```

Description

Use the InsertMasked method to insert a masked image into the image list. The new image is inserted following the image specified by Index. Use MaskColor to identify the color in the image used to generate the Mask (see page 15797).

See Also

AddMasked (see page 15369)

Insert (see page 15381)

80.1.2.5.30 TCustomImageList.Move Method

Moves a specified image to another position.

Pascal

```
procedure Move(CurIndex: Integer; NewIndex: Integer);
```

C++

```
__fastcall Move(int CurIndex, int NewIndex);
```

Description

Use the Move method to move an image to another position in the image list. Specify the current position using CurIndex, and the new position using NewIndex.

See Also

Insert (see page 15381)

80.1.2.5.31 TCustomImageList.Overlay Method

Identifies an image in the list as an image that can be used as an overlay mask.

Pascal

```
function Overlay(ImageIndex: Integer; Overlay: TOverlay): Boolean;
```

C++

```
__fastcall Boolean Overlay(int ImageIndex, TOverlay Overlay);
```

Description

Use Overlay to assign an overlay index to one of the images in the list. Once an image has an overlay index, it can be used as an overlay mask. An overlay mask is an image drawn transparently over another image.

ImageIndex identifies the image that can act as an overlay mask.

Overlay is the overlay index to assign to that image. It must be a value between 0 and 3 (inclusive). Note that this means at most four images can be used as overlay masks.

If successful, Overlay returns true.

To draw an overlay mask over an image use the DrawOverlaymethodwith the index of the overlay mask.

See Also

DrawOverlay (see page 15375)

80.1.2.5.32 TCustomImageList.ReadData Method

Pascal

```
procedure ReadData(Stream: TStream); virtual;
```

C++

```
virtual __fastcall ReadData(TStream Stream);
```

Description

This is ReadData, a member of class TCustomImageList.

80.1.2.5.33 TCustomImageList.RegisterChanges Method

Registers an object to be notified when changes occur in the image list.

Pascal

```
procedure RegisterChanges(Value: TChangeLink);
```

C++

```
__fastcall RegisterChanges(TChangeLink Value);
```

Description

Use the RegisterChanges method to have an object notified whenever changes occur in the image list. The TChangeLink (see page 15358)'s OnChange (see page 15393) event is called whenever a change in the image list occurs.

Create (see page 15365) a change link object for each object that should be notified of changes. Register each object by calling RegisterChanges and passing its change link object as the Value parameter. The OnChange (see page 15393) event of all registered change link objects is then triggered when a change occurs in the image list. Any code assigned to the OnChange (see page 15393) event handlers will execute.

See Also

UnRegisterChanges (see page 15386)

OnChange (see page 15361)

80.1.2.5.34 TCustomImageList.Replace Method

Replaces an image in the image list with a new image and mask.

Pascal

```
procedure Replace(Index: Integer; Image: TBitmap; Mask: TBitmap);
```

C++

```
__fastcall Replace(int Index, TBitmap Image, TBitmap Mask);
```

Description

Use the Replace method to replace the image at the Index position in the image list with a new Image and overlay Mask (see page 15797).

See Also

Insert (see page 15381)

ReplaceIcon (see page 15384)

ReplaceMasked (see page 15384)

80.1.2.5.35 TCustomImageList.ReplaceIcon Method

Replaces an image in the image list with an icon.

Pascal

```
procedure ReplaceIcon(Index: Integer; Image: TIcon);
```

C++

```
__fastcall ReplaceIcon(int Index, TIcon Image);
```

Description

Use the ReplaceIcon method to replace the Image at the Index position in the image list with a new icon.

See Also

Replace (see page 15384)

InsertIcon (see page 15382)

80.1.2.5.36 TCustomImageList.ReplaceMasked Method

Replaces an image in the image list with a new image and automatically-generated mask.

Pascal

```
procedure ReplaceMasked(Index: Integer; NewImage: TBitmap; MaskColor: TColor);
```

C++

```
__fastcall ReplaceMasked(int Index, TBitmap NewImage, TColor MaskColor);
```

Description

Use the ReplaceMasked method to replace the image at Index position within the image list specified by NewImage and the mask generated from NewImage by masking out all pixels of the color specified by MaskColor.

See Also

Replace (see page 15384)

InsertMasked (see page 15382)

80.1.2.5.37 TCustomImageList.ResInstLoad Method

Loads a resource into the image list.

Pascal

```
function ResInstLoad(Instance: THandle; ResType: TResType; const Name: string; MaskColor:
TColor): Boolean;
```

C++

```
__fastcall Boolean ResInstLoad(THandle Instance, TResType ResType, const AnsiString Name,
TColor MaskColor);
```

Description

Use the ResoInstLoad method to load a resource of type TResType (see page 15396) into the image list. Use Instance to specify the instance handle of the resource module, and Name to identify the resource within that module. MaskColor is used to create the mask for the resulting image. ResInstLoad should be used rather than ResourceLoad (see page 15385) to load a resource from within a package.

ResInstLoad returns true if it is successful, false otherwise.

See Also

GetInstRes (see page 15379)

GetResource (see page 15380)

80.1.2.5.38 TCustomImageList.ResourceLoad Method

Loads a resource into the image list.

Pascal

```
function ResourceLoad(ResType: TResType; const Name: string; MaskColor: TColor): Boolean;
```

C++

```
__fastcall Boolean ResourceLoad(TResType ResType, const AnsiString Name, TColor MaskColor);
```

Description

Use the ResourceLoad method to load a resource of type TResType (see page 15396) into the image list. Use Name to identify the resource. MaskColor is used to create the mask for the resulting image. ResourceLoad returns true if successful, false otherwise.

Note: Within a package, use ResInstLoad (see page 15385) instead of ResourceLoad. ResourceLoad returns an error when called from within a package.

See Also

GetResource (see page 15380)

ResInstLoad (see page 15385)

80.1.2.5.39 TCustomImageList.UnRegisterChanges Method

Removes an object from the list of objects to be notified when the image list changes.

Pascal

```
procedure UnRegisterChanges(Value: TChangeLink);
```

C++

```
__fastcall UnRegisterChanges(TChangeLink Value);
```

Description

Use the UnRegisterChanges method to unregister a TChangeLink (see page 15358) object. The object will no longer receive notification when the image list changes.

See Also

RegisterChanges (see page 15383)

80.1.2.5.40 TCustomImageList.WriteData Method

Pascal

```
procedure WriteData(Stream: TStream); virtual;
```

C++

```
virtual __fastcall WriteData(TStream Stream);
```

Description

This is WriteData, a member of class TCustomImageList.

80.1.2.6 TCustomImageList Properties

The properties of the TCustomImageList class are listed here.

Public Properties

	Name	Description
	AllocBy (see page 15387)	Sets the number of images by which the image list grows when it needs to make room for new images.
	BkColor (see page 15387)	Determines which background color to use when drawing an image.
	BlendColor (see page 15387)	Determines which foreground color to use when drawing an image.
R	Count (see page 15388)	Indicates the number of images in the image list.
	DrawingStyle (see page 15390)	Specifies the style to be used when the image list is drawing an image.
	Handle (see page 15390)	Designates the handle for the image list.
	Height (see page 15391)	Specifies the height of images within the image list.
	ImageType (see page 15391)	Determines whether the image list uses the associated mask when drawing.
	Masked (see page 15392)	Specifies whether the image list includes masks that can be combined with the images.
	ShareImages (see page 15392)	Specifies whether the image list destroys its handle when the image list is destroyed.
	Width (see page 15392)	Specifies the width of all images within the image list.

80.1.2.6.1 TCustomImageList.AllocBy Property

Sets the number of images by which the image list grows when it needs to make room for new images.

Pascal

```
property AllocBy: Integer;
```

C++

```
__property int AllocBy;
```

Description

Use the AllocBy method to specify the number of new images that can be added to the resized image list.

See Also

Count (see page 15388)

Add (see page 15368)

AddIcon (see page 15368)

AddImages (see page 15369)

AddMasked (see page 15369)

80.1.2.6.2 TCustomImageList.BkColor Property

Determines which background color to use when drawing an image.

Pascal

```
property BkColor: TColor;
```

C++

```
__property TColor BkColor;
```

Description

Use BkColor to specify the color used for the masked regions of the images in the list. When BkColor is clNone, the image is drawn transparently using the mask to define the transparent areas. Otherwise, the value of BkColor appears in the masked out portions.

Note: BkColor has no effect when the Masked (see page 15392) property is false.

See Also

BlendColor (see page 15387)

Masked (see page 15392)

ImageType (see page 15391)

80.1.2.6.3 TCustomImageList.BlendColor Property

Determines which foreground color to use when drawing an image.

Pascal

```
property BlendColor: TColor;
```

C++

```
__property TColor BlendColor;
```

Description

Set BlendColor to cause the images in the list to be combined with the specified color when the DrawingStyle (see page 15390) is dsFocus or dsSelected. More of BlendColor is combined with the images when DrawingStyle (see page 15390) is dsSelected. clNone specifies no blend color while clDefault specifies the system highlight .

See Also

BkColor (see page 15387)

DrawingStyle (see page 15390)

80.1.2.6.4 TCustomImageList.Count Property

Indicates the number of images in the image list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use Count when iterating through the images in the image list. Retrieve the individual images using the GetBitmap (see page 15376) method. Draw (see page 15373) individual images using the Draw (see page 15373) method.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
```

```
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", (TObject *)vsIcon);
  ComboBox1->Items->AddObject("vsList", (TObject *)vsList);
  ComboBox1->Items->AddObject("vsReport", (TObject *)vsReport);
  ComboBox1->Items->AddObject("vsSmallIcon", (TObject *)vsSmallIcon);
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := 'Image' + IntToStr(I);
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
```

```
      NewColumn.Caption := 'Column 2';
      NewColumn.Width := 200;
      // Add View styles and constants to the Combo Box
      ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
      ComboBox1.Items.AddObject('vsList', TObject(vsList));
      ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
      ComboBox1.Items.AddObject(
        'vsSmallIcon', TObject(vsSmallIcon));
      // Display first item in the Combo Box
      ComboBox1.ItemIndex := 0;
    end;
  end;

  procedure TComboForm.ComboBox1Click(Sender: TObject);
  begin
    with ComboBox1 do
      ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

  end;
```

See Also

AllocBy (see page 15387)

GetBitmap (see page 15376)

Draw (see page 15373)

80.1.2.6.5 TCustomImageList.DrawingStyle Property

Specifies the style to be used when the image list is drawing an image.

Pascal

```
property DrawingStyle: TDrawingStyle;
```

C++

```
__property TDrawingStyle DrawingStyle;
```

Description

Use the DrawingStyle property to control how the image is to appear. DrawingStyle controls how much of the blend color specified by BlendColor (see page 15387) is combined with the foreground, or what background color to use.

See Also

Draw (see page 15373)

DrawOverlay (see page 15375)

ImageType (see page 15391)

BlendColor (see page 15387)

BkColor (see page 15387)

80.1.2.6.6 TCustomImageList.Handle Property

Designates the handle for the image list.

Pascal

```
property Handle: HImageList;
```

C++

```
__property HImageList Handle;
```

Description

Handle is the Windows image list object handle. Use this handle when calling Windows API functions that require it.

See Also

HandleAllocated (see page 15380)

80.1.2.6.7 TCustomImageList.Height Property

Specifies the height of images within the image list.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use the Height property to specify the height of images before any are added to the image list. The image list is cleared when this value changes.

See Also

Width (see page 15392)

CreateSize (see page 15366)

Create (see page 15365)

80.1.2.6.8 TCustomImageList.ImageType Property

Determines whether the image list uses the associated mask when drawing.

Pascal

```
property ImageType: TImageType;
```

C++

```
__property TImageType ImageType;
```

Description

Use the ImageType property to control whether the images are drawn using the image mask (itMask), or without the mask (itImage).

See Also

DrawingStyle (see page 15390)

Draw (see page 15373)

Masked (see page 15392)

DrawOverlay (see page 15375)

80.1.2.6.9 TCustomImageList.Masked Property

Specifies whether the image list includes masks that can be combined with the images.

Pascal

```
property Masked: Boolean;
```

C++

```
__property Boolean Masked;
```

Description

Set the Masked property to true to combine the images with masks. The masked portion of the bitmap is drawn transparently, or replaced by the color specified by the BkColor (see page 15387) property.

Masked indicates whether the image list stores masks to combine with the images. To use those masks, the ImageType (see page 15391) property indicates whether to draw the image or the mask.

See Also

AddMasked (see page 15369)

InsertMasked (see page 15382)

BkColor (see page 15387)

ImageType (see page 15391)

80.1.2.6.10 TCustomImageList.ShareImages Property

Specifies whether the image list destroys its handle when the image list is destroyed.

Pascal

```
property ShareImages: Boolean;
```

C++

```
__property Boolean ShareImages;
```

Description

Set the ShareImages property to false to ensure that the handle to the image list is freed when the image list is destroyed.

See Also

Destroy (see page 15366)

Handle (see page 15390)

80.1.2.6.11 TCustomImageList.Width Property

Specifies the width of all images within the image list.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use the Width property to specify the width of images before any are added to the image list. The image list is cleared when this

value changes.

See Also

Height (see page 15391)

Create (see page 15365)

CreateSize (see page 15366)

80.1.2.7 TCustomImageList Events

The events of the TCustomImageList class are listed here.

Public Events

	Name	Description
	OnChange (see page 15393)	Occurs when the image list changes.

80.1.2.7.1 TCustomImageList.OnChange Event

Occurs when the image list changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to execute code when the image list changes. Any objects registered with the RegisterChanges (see page 15383) method will also be notified.

See Also

RegisterChanges (see page 15383)

80.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TDrawingStyle (see page 15394)	TDrawingStyle indicates the style to use when rendering an image from an image list.
	TImageType (see page 15395)	TImageType indicates whether to use a mask when rendering an image from an image list.
	TLoadResource (see page 15395)	TLoadResource and TLoadResources (see page 15397) determine how a resource is loaded into an image list.
	TResType (see page 15396)	TResType identifies a type of graphical resource to an image list.

80.2.1 ImgList.TDrawingStyle Enumeration

TDrawingStyle indicates the style to use when rendering an image from an image list.

Pascal

```
TDrawingStyle = (
  dsFocus,
  dsSelected,
  dsNormal,
  dsTransparent
);
```

C++

```
enum TDrawingStyle {
  dsFocus,
  dsSelected,
  dsNormal,
  dsTransparent
};
```

File

ImgList

Description

TDrawingStyle describes how to render an image from an image list component. The following table describes the possible values:

Value	Meaning
dsFocus	Draws the image blending 25% with the color specified by the BlendColor property. This only affects image lists that contain masks.
dsSelected	Draws the image blending 50% with the color specified by the BlendColor property. This only affects image lists that contain masks.
dsNormal	Draws the image using the color specified in the BkColor property. If the BkColor is clNone, the image is drawn transparently using the mask.
dsTransparent	Draws using the mask regardless of the BkColor setting.

80.2.2 ImgList.TImageType Enumeration

TImageType indicates whether to use a mask when rendering an image from an image list.

Pascal

```
TImageType = (
  itImage,
  itMask
);
```

C++

```
enum TImageType {
  itImage,
  itMask
};
```

File

ImgList

Description

TImageType indicates whether to use a mask when rendering an image in an image list. It includes the following values:

Value	Meaning
itImage	Image does not include a mask.
itMask	Image uses a mask.

80.2.3 ImgList.TLoadResource Enumeration

TLoadResource and TLoadResources (see page 15397) determine how a resource is loaded into an image list.

Pascal

```
TLoadResource = (
  lrDefaultColor,
  lrDefaultSize,
  lrFromFile,
  lrMap3DColors,
  lrTransparent,
  lrMonoChrome
);
```

C++

```
enum TLoadResource {
  lrDefaultColor,
  lrDefaultSize,
  lrFromFile,
  lrMap3DColors,
  lrTransparent,
  lrMonoChrome
};
```

File

ImgList

Description

TLoadResources (see page 15397) is a set of flags (TLoadResource values) the govern the way an image list loads image resources. The following table lists the possible TLoadResource values:

Value	Meaning
lrDefaultColor	Use the color format of the display.
lrDefaultSize	Use system metrics for Height and Width if the Width parameter to GetResource is 0. If the Width parameter is 0 and this option is not specified for cursors and icons, the resources width and height will be used.
lrFromFile	Loads the resource from a specified file.
lrMap3DColors	Replaces the images' gray shades with the 3D windows colors.
lrTransparent	Replaces the color table entry for the first pixel of the resource with the default window color (COLOR_WINDOW). This applies only to images with color tables.
lrMonoChrome	Loads the resource as black and white.

See Also

SelectDirectory (see page 13619)

80.2.4 ImgList.TResType Enumeration

TResType identifies a type of graphical resource to an image list.

Pascal

```
TResType = (
  rtBitmap,
  rtCursor,
  rtIcon
);
```

C++

```
enum TResType {
  rtBitmap,
  rtCursor,
  rtIcon
};
```

File

ImgList

Description

TResTypes is used by the image list methods ResourceLoad, GetResource and FileLoad to specify the type of graphical resource that is to be loaded or retrieved from an image list.

Value	Meaning
rtBitmap	Loads or retrieves a bitmap
rtCursor	Loads or retrieves a cursor
rtIcon	Loads or retrieves an icon

80.3 Types

The following table lists types in this documentation.

Types

Name	Description
TImageIndex (see page 15397)	This is type ImgList.TImageIndex.
TLoadResources (see page 15397)	This is type ImgList.TLoadResources.
TOverlay (see page 15397)	This is type ImgList.TOverlay.

80.3.1 ImgList.TImageIndex Type

Pascal

```
TImageIndex = type Integer;
```

C++

```
type Integer TImageIndex;
```

File

ImgList

Description

This is type ImgList.TImageIndex.

80.3.2 ImgList.TLoadResources Type

Pascal

```
TLoadResources = set of TLoadResource;
```

C++

```
TLoadResource TLoadResources;
```

File

ImgList

Description

This is type ImgList.TLoadResources.

80.3.3 ImgList.TOverlay Type

Pascal

```
TOverlay = 0..3;
```

C++

```
0..3 TOverlay;
```

File

ImgList

Description

This is type ImgList.TOverlay.

81 IMouse Namespace

This is namespace IMouse.

Topics

Name	Description
Classes (see page 15401)	The following table lists classes in this documentation.
Functions (see page 15423)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 15424)	The following table lists structs, records, enums in this documentation.
Types (see page 15426)	The following table lists types in this documentation.
Constants (see page 15428)	The following table lists constants in this documentation.

Classes

	Name	Description
	TPanningWindow (see page 15401)	This is a class of TCustomPanningWindow.

Constants

Name	Description
crPanAll (see page 15428)	This is constant IMouse.crPanAll.
crPanDown (see page 15428)	This is constant IMouse.crPanDown.
crPanDownLeft (see page 15429)	This is constant IMouse.crPanDownLeft.
crPanDownRight (see page 15429)	This is constant IMouse.crPanDownRight.
crPanLeft (see page 15429)	This is constant IMouse.crPanLeft.
crPanLeftRight (see page 15430)	This is constant IMouse.crPanLeftRight.
crPanRight (see page 15430)	This is constant IMouse.crPanRight.
crPanUp (see page 15430)	This is constant IMouse.crPanUp.
crPanUpDown (see page 15431)	This is constant IMouse.crPanUpDown.
crPanUpLeft (see page 15431)	This is constant IMouse.crPanUpLeft.
crPanUpRight (see page 15431)	This is constant IMouse.crPanUpRight.

Functions

	Name	Description
	StartPan (see page 15423)	This is function IMouse.StartPan.

Structs, Records, Enums

	Name	Description
	TPanDirection (see page 15424)	The type describes the panning direction.

	TPanOption (see page 15424)	This is record IMouse.TPanOption.

Types

Name	Description
TPanDirections (see page 15426)	Set of TPanDirection (see page 15424). .
TPanOptions (see page 15426)	This is type IMouse.TPanOptions.
TUpdateEvent (see page 15426)	Initiates the cursor update.

81.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TPanningWindow (see page 15401)	This is a class of TCustomPanningWindow.

81.1.1 TPanningWindow Class

This is a class of TCustomPanningWindow.

Class Hierarchy

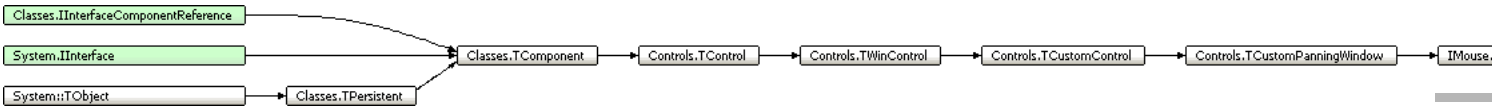

Pascal

```
TPanningWindow = class(TCustomPanningWindow);
```

C++

```
class TPanningWindow : public TCustomPanningWindow;
```

File

imouse

Description

Use this type if you are creating a specialized version of TPanningWindowClass.

81.1.1.1 TPanningWindow Members

The following tables list the members exposed by TPanningWindow.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.

	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.

	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 15413)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.

	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DoUpdate (see page 15414)	This is DoUpdate, a member of class TPanningWindow.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
V	GetPanOptions (see page 15414)	This is GetPanOptions, a member of class TPanningWindow.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
V	MouseMove (see page 15414)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 15415)	Renders the image of a custom control.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.

	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetPanCursor (see page 15415)	This is SetPanCursor, a member of class TPanningWindow.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateScroll (see page 15416)	This is UpdateScroll, a member of class TPanningWindow.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 15416)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.

	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnUpdate (see page 15422)	This is OnUpdate, a member of class TPanningWindow.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 15412)	Creates an instance of TCustomControl.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 15412)	Destroys an instance of TCustomControl.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetIsPanning (see page 15414)	Returns whether or not the mouse is actively panning.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.

	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.

	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	StartPanning (see page 15416)	Tells the panning window to start the panning operation.
V	StopPanning (see page 15416)	Tells the panning window to stop the panning operation.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.

R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
R	IsPanning (see page 15421)	This is IsPanning, a member of class TPanningWindow.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
R	PanControl (see page 15421)	This is PanControl, a member of class TPanningWindow.
	PanInterval (see page 15421)	This is PanInterval, a member of class TPanningWindow.
R	PanOptions (see page 15422)	This is PanOptions, a member of class TPanningWindow.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

81.1.1.2 TPanningWindow.Create Constructor

Creates an instance of TCustomControl.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TPanningWindow(TComponent * AOwner);
```

Description

Calling Create constructs and initializes an instance of TCustomControl. However, you should never attempt to instantiate a TCustomControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants. Controls placed on forms at design time are created automatically.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes a TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create (see page 6123)

Destroy (see page 5919)

Canvas (see page 5921)

Owner (see page 2807)

81.1.1.3 TPanningWindow.Destroy Destructor

Destroys an instance of TCustomControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPanningWindow();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the control is not nil, and only then calls Destroy.

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Destroys the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

Free (see page 22300)

delete

Create (see page 5918)

81.1.1.4 TPanningWindow Methods

The methods of the TPanningWindow class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 15413)	Initializes a window-creation parameter data structure.
	DoUpdate (see page 15414)	This is DoUpdate, a member of class TPanningWindow.
	GetPanOptions (see page 15414)	This is GetPanOptions, a member of class TPanningWindow.
	MouseMove (see page 15414)	Respond to mouse moving over control area..
	Paint (see page 15415)	Renders the image of a custom control.
	SetPanCursor (see page 15415)	This is SetPanCursor, a member of class TPanningWindow.
	UpdateScroll (see page 15416)	This is UpdateScroll, a member of class TPanningWindow.
	WndProc (see page 15416)	Provides specific message responses for the control.

Public Methods

	Name	Description
	GetIsPanning (see page 15414)	Returns whether or not the mouse is actively panning.
	StartPanning (see page 15416)	Tells the panning window to start the panning operation.
	StopPanning (see page 15416)	Tells the panning window to stop the panning operation.

81.1.1.4.1 TPanningWindow.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

81.1.1.4.2 TPanningWindow.DoUpdate Method

Pascal

```
procedure DoUpdate(var Delta: TPoint; var CursorDirection: TPanDirections); dynamic;
```

C++

```
__fastcall DoUpdate(TPoint Delta, TPanDirections CursorDirection);
```

Description

This is DoUpdate, a member of class TPanningWindow.

81.1.1.4.3 TPanningWindow.GetIsPanning Method

Returns whether or not the mouse is actively panning.

Pascal

```
function GetIsPanning: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsPanning();
```

Description

This method brings back a boolean value. True means that the mouse is actively panning; False indicates that no panning is occurring.

81.1.1.4.4 TPanningWindow.GetPanOptions Method

Pascal

```
function GetPanOptions: TPanOptions; virtual;
```

C++

```
virtual __fastcall TPanOptions GetPanOptions();
```

Description

This is GetPanOptions, a member of class TPanningWindow.

81.1.1.4.5 TPanningWindow.MouseMove Method

Respond to mouse moving over control area..

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation.

See Also

MouseDown (see page 5773)

MouseUp (see page 5774)

OnMouseMove (see page 5865)

81.1.1.4.6 TPanningWindow.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas (see page 5921)

PaintWindow (see page 5920)

GetDeviceContext (see page 6155)

ClipRect (see page 14486)

81.1.1.4.7 TPanningWindow.SetPanCursor Method

Pascal

```
procedure SetPanCursor(ADirection: TPanDirections); virtual;
```

C++

```
virtual __fastcall SetPanCursor(TPanDirections ADirection);
```

Description

This is SetPanCursor, a member of class TPanningWindow.

81.1.1.4.8 TPanningWindow.StartPanning Method

Tells the panning window to start the panning operation.

Pascal

```
function StartPanning(AHandle: THandle; AControl: TControl): Boolean; override;
```

C++

```
virtual __fastcall Boolean StartPanning(THandle AHandle, TControl * AControl);
```

Description

Include this method in your code where you want the panning window to start the panning operation.

81.1.1.4.9 TPanningWindow.StopPanning Method

Tells the panning window to stop the panning operation.

Pascal

```
procedure StopPanning; override;
```

C++

```
virtual __fastcall StopPanning();
```

Description

Add this method to your code where you want the panning window to stop the panning operation.

81.1.1.4.10 TPanningWindow.UpdateScroll Method

Pascal

```
procedure UpdateScroll(Pt: TPoint); virtual;
```

C++

```
virtual __fastcall UpdateScroll(TPoint Pt);
```

Description

This is UpdateScroll, a member of class TPanningWindow.

81.1.1.4.11 TPanningWindow.WndProc Method

Provides specific message responses for the control.

Pascal

```
procedure WndProc(var Msg: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Msg);
```

Description

Override WndProc to change the initial Windows message handler for the control. The WindowProc property is initialized to point to the WndProc method.

WndProc for TWinControl overrides the inherited method to define responses for focus, mouse, and keyboard messages. It sends all others to its inherited WndProc.

When overriding WndProc to provide specialized responses to messages, call the inherited WndProc at the end to dispatch any unhandled messages.

C++ Examples:

```
/*
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
*/
class TMyListBoxDescendant : public TListBox
{
__published:    // IDE-managed Components
    void __fastcall SubClassWndProc(Messages::TMessage &Message);
    void __fastcall ToggleSubClass(bool On);
    void __fastcall OnDrawItemProc(
      TWinControl *Control, int Index, const TRect &Rect,
      TOwnerDrawState State);
private:    // User declarations
public:     // User declarations
    __fastcall TMyListBoxDescendant(TComponent* Owner);
};

TForm1 *Form1;
TMyListBoxDescendant *MyListBoxDescendant1;
Graphics::TBitmap *bitmap0;

const WM_STYLEMESSAGE = WM_USER + 2000;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

__fastcall TMyListBoxDescendant::TMyListBoxDescendant(TComponent* Owner)
    : TListBox(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
}

void __fastcall TForm1::SubClassRadioGroup1Click(TObject *Sender)
{
  MyListBoxDescendant1->ToggleSubClass(SubClassRadioGroup1->ItemIndex == 0);
}

void __fastcall TMyListBoxDescendant::SubClassWndProc(Messages::TMessage &Message)
```

```
{
  if (Message.Msg == WM_STYLEMESSAGE)
    Style = (TListBoxStyle)Message.WParam;
  else
    WndProc(Message);
}

void __fastcall TMyListBoxDescendant::ToggleSubClass(bool On)
{
  if (On)
    WindowProc = SubClassWndProc;
  else
    WindowProc = WndProc;
}

void __fastcall TMyListBoxDescendant::OnDrawItemProc(TWinControl *Control, int Index,
      const TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *bitmap; // temporary variable for item's bitmap
  int Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *canvas = ((TListBox *)Control)->Canvas;
  canvas->FillRect(Rect); // clear the rectangle
  bitmap = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
  if (bitmap)
  {
    canvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, bitmap->Width, bitmap->Height),
      bitmap, Bounds(0, 0, bitmap->Width, bitmap->Height), clRed); // render bitmap
    Offset += bitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  canvas->TextOut(Rect.Left + Offset, Rect.Top, ((TListBox *)Control)->Items->Strings[Index]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyListBoxDescendant1 = new TMyListBoxDescendant(Form1);
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Parent = Form1;
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Left =
    SubClassRadioGroup1->Left + SubClassRadioGroup1->Width + 30;;
  MyListBoxDescendant1->Top = SubClassRadioGroup1->Top;
  MyListBoxDescendant1->Height = SubClassRadioGroup1->Height;
  MyListBoxDescendant1->OnDrawItem =
    MyListBoxDescendant1->OnDrawItemProc;

  bitmap0 = new Graphics::TBitmap;
  ImageList1->GetBitmap(0, bitmap0);
  MyListBoxDescendant1->Items->AddObject("Butterfly", bitmap0);

  SubClassRadioGroup1->Items->Add("SubClassWndProc");
  SubClassRadioGroup1->Items->Add("WndProc");
  SubClassRadioGroup1->ItemIndex = 2;
}
```

Delphi Examples:

```
{
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
}
```

```
type
  TMyListBoxDescendant = class(TlistBox)
    procedure SubClassWndProc(var Message: TMessage);
    procedure ToggleSubClass(On: Boolean);
    procedure OnDrawItemProc(
      Control: TWinControl;
      Index: Integer;
      Rect:TRect;
      State: TOwnerDrawState);
  end;
  TForm1 = class(TForm)
    SubClassRadioGroup1: TRadioGroup;
    Button1: TButton;
    ImageList1: TImageList;
    Button2: TButton;
    procedure SubClassRadioGroup1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  MyListBoxDescendant1: TMyListBoxDescendant;
  bitmap0: TBitmap;

implementation

{$R *.dfm}

const WM_STYLEMESSAGE = WM_USER + 2000;

procedure TForm1.Button1Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
end;

procedure TMyListBoxDescendant.SubClassWndProc(var Message: TMessage);
begin
  if (Message.Msg = WM_STYLEMESSAGE) then
    Style:= TListBoxStyle(Message.WParam)
  else
    WndProc(Message);
end;

procedure TMyListBoxDescendant.ToggleSubClass(On: Boolean);
begin
```

```
    if On then
      WindowProc := SubClassWndProc
    else
      WindowProc := WndProc;
  end;

  procedure TForm1.SubClassRadioGroup1Click(Sender: TObject);
  begin
    MyListBoxDescendant1.ToggleSubClass(
      SubClassRadioGroup1.ItemIndex = 0);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    MyListBoxDescendant1:= TMyListBoxDescendant.Create(self);
    MyListBoxDescendant1.Visible:= True;
    MyListBoxDescendant1.Parent:= Form1;
    MyListBoxDescendant1.Visible:= True;
    MyListBoxDescendant1.Left:=
      SubClassRadioGroup1.Left + SubClassRadioGroup1.Width + 30;;
    MyListBoxDescendant1.Top:= SubClassRadioGroup1.Top;
    MyListBoxDescendant1.Height:= SubClassRadioGroup1.Height;
    MyListBoxDescendant1.OnDrawItem:=
      MyListBoxDescendant1.OnDrawItemProc;

    bitmap0 := TBitmap.Create;
    ImageList1.GetBitmap(0, bitmap0);
    MyListBoxDescendant1.Items.AddObject('Butterfly', bitmap0);

    SubClassRadioGroup1.Items.Add('SubClassWndProc');
    SubClassRadioGroup1.Items.Add('WndProc');
    SubClassRadioGroup1.ItemIndex := 2;
  end;

  procedure TMyListBoxDescendant.OnDrawItemProc(
    Control: TWinControl;
    Index: Integer;
    Rect:TRect;
    State: TOwnerDrawState);
  var
    Bitmap: TBitmap;      { temporary variable for the item's bitmap }
    Offset: Integer;      { text offset width }
  begin
    { draw on control canvas, not on the form }
    with (Control as TListBox).Canvas do
    begin
      FillRect(Rect);       { clear the rectangle }
      Offset := 2;          { provide default offset }
      Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
      if Bitmap <> nil then
      begin
        BrushCopy(
          Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
          Bitmap,
          Bounds(0, 0, Bitmap.Width, Bitmap.Height),
          clRed);  {render bitmap}
        Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
      end;
      TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
  text }
    end;
  end;
```

See Also

DefWndProc (see page 6199)

MainWndProc (see page 6165)

WndProc (see page 5791)

WindowProc (see page 5872)

81.1.1.5 TPanningWindow Properties

The properties of the TPanningWindow class are listed here.

Public Properties

	Name	Description
R	IsPanning (see page 15421)	This is IsPanning, a member of class TPanningWindow.
R	PanControl (see page 15421)	This is PanControl, a member of class TPanningWindow.
	PanInterval (see page 15421)	This is PanInterval, a member of class TPanningWindow.
R	PanOptions (see page 15422)	This is PanOptions, a member of class TPanningWindow.

81.1.1.5.1 TPanningWindow.IsPanning Property

Pascal

```
property IsPanning: Boolean;
```

C++

```
__property Boolean IsPanning;
```

Description

This is IsPanning, a member of class TPanningWindow.

81.1.1.5.2 TPanningWindow.PanControl Property

Pascal

```
property PanControl: TControl;
```

C++

```
__property TControl * PanControl;
```

Description

This is PanControl, a member of class TPanningWindow.

81.1.1.5.3 TPanningWindow.PanInterval Property

Pascal

```
property PanInterval: Integer;
```

C++

```
__property int PanInterval;
```

Description

This is PanInterval, a member of class TPanningWindow.

81.1.1.5.4 TPanningWindow.PanOptions Property

Pascal

```
property PanOptions: TPanOptions;
```

C++

```
__property TPanOptions PanOptions;
```

Description

This is PanOptions, a member of class TPanningWindow.

81.1.1.6 TPanningWindow Events

The events of the TPanningWindow class are listed here.

Public Events

	Name	Description
	OnUpdate (see page 15422)	This is OnUpdate, a member of class TPanningWindow.

81.1.1.6.1 TPanningWindow.OnUpdate Event

Pascal

```
property OnUpdate: TUpdateEvent;
```

C++

```
__property TUpdateEvent OnUpdate;
```

Description

This is OnUpdate, a member of class TPanningWindow.

81.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	StartPan (see page 15423)	This is function IMouse.StartPan.

81.2.1 IMouse.StartPan Function

Pascal

```
function StartPan(WndHandle: THandle; AControl: TControl): Boolean;
```

C++

```
Boolean StartPan(THandle WndHandle, TControl * AControl);
```

File

imouse

Description

This is function IMouse.StartPan.

81.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TPanDirection (see page 15424)	The type describes the panning direction.
	TPanOption (see page 15424)	This is record IMouse.TPanOption.

81.3.1 IMouse.TPanDirection Enumeration

The type describes the panning direction.

Pascal

```
TPanDirection = (
  pdUp,
  pdDown,
  pdLeft,
  pdRight
);
```

C++

```
enum TPanDirection {
  pdUp,
  pdDown,
  pdLeft,
  pdRight
};
```

File

imouse

Description

The possible panning values are: pdUp, pdDown, pdLeft, pdRight.

81.3.2 IMouse.TPanOption Enumeration

Pascal

```
TPanOption = (
  poVertical,
  poHorizontal
);
```

C++

```
enum TPanOption {
  poVertical,
  poHorizontal
};
```

File

imouse

Description

This is record IMouse.TPanOption.

81.4 Types

The following table lists types in this documentation.

Types

Name	Description
TPanDirections (see page 15426)	Set of TPanDirection (see page 15424). .
TPanOptions (see page 15426)	This is type IMouse.TPanOptions.
TUpdateEvent (see page 15426)	Initiates the cursor update.

81.4.1 IMouse.TPanDirections Type

Set of TPanDirection (see page 15424). .

Pascal

```
TPanDirections = set of TPanDirection;
```

C++

```
TPanDirection TPanDirections;
```

File

imouse

Description

Set of TPanDirection (see page 15424).

81.4.2 IMouse.TPanOptions Type

Pascal

```
TPanOptions = set of TPanOption;
```

C++

```
TPanOption TPanOptions;
```

File

imouse

Description

This is type IMouse.TPanOptions.

81.4.3 IMouse.TUpdateEvent Type

Initiates the cursor update.

Pascal

```
TUpdateEvent = procedure (Sender: TObject; var Delta: TPoint; var CursorDirection:
TPanDirections) of object;
```

C++

```
(Sender: TObject; var Delta: TPoint; var CursorDirection: TPanDirections) ( TUpdateEvent)();
```

File

imouse

Description

This event is called when the cursor is to be updated.

81.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
crPanAll (see page 15428)	This is constant IMouse.crPanAll.
crPanDown (see page 15428)	This is constant IMouse.crPanDown.
crPanDownLeft (see page 15429)	This is constant IMouse.crPanDownLeft.
crPanDownRight (see page 15429)	This is constant IMouse.crPanDownRight.
crPanLeft (see page 15429)	This is constant IMouse.crPanLeft.
crPanLeftRight (see page 15430)	This is constant IMouse.crPanLeftRight.
crPanRight (see page 15430)	This is constant IMouse.crPanRight.
crPanUp (see page 15430)	This is constant IMouse.crPanUp.
crPanUpDown (see page 15431)	This is constant IMouse.crPanUpDown.
crPanUpLeft (see page 15431)	This is constant IMouse.crPanUpLeft.
crPanUpRight (see page 15431)	This is constant IMouse.crPanUpRight.

81.5.1 IMouse.crPanAll Constant

Pascal

```
crPanAll = TCursor(-50);
```

C++

```
crPanAll = TCursor(-50);
```

File

imouse

Description

This is constant IMouse.crPanAll.

81.5.2 IMouse.crPanDown Constant

Pascal

```
crPanDown = TCursor(-51);
```

C++

```
crPanDown = TCursor(-51);
```

File

imouse

Description

This is constant IMouse.crPanDown.

81.5.3 IMouse.crPanDownLeft Constant

Pascal

```
crPanDownLeft = TCursor(-52);
```

C++

```
crPanDownLeft = TCursor(-52);
```

File

imouse

Description

This is constant IMouse.crPanDownLeft.

81.5.4 IMouse.crPanDownRight Constant

Pascal

```
crPanDownRight = TCursor(-53);
```

C++

```
crPanDownRight = TCursor(-53);
```

File

imouse

Description

This is constant IMouse.crPanDownRight.

81.5.5 IMouse.crPanLeft Constant

Pascal

```
crPanLeft = TCursor(-54);
```

C++

```
crPanLeft = TCursor(-54);
```

File

imouse

Description

This is constant IMouse.crPanLeft.

81.5.6 IMouse.crPanLeftRight Constant

Pascal

```
crPanLeftRight = TCursor(-55);
```

C++

```
crPanLeftRight = TCursor(-55);
```

File

imouse

Description

This is constant IMouse.crPanLeftRight.

81.5.7 IMouse.crPanRight Constant

Pascal

```
crPanRight = TCursor(-56);
```

C++

```
crPanRight = TCursor(-56);
```

File

imouse

Description

This is constant IMouse.crPanRight.

81.5.8 IMouse.crPanUp Constant

Pascal

```
crPanUp = TCursor(-57);
```

C++

```
crPanUp = TCursor(-57);
```

File

imouse

Description

This is constant IMouse.crPanUp.

81.5.9 IMouse.crPanUpDown Constant

Pascal

```
crPanUpDown = TCursor(-58);
```

C++

```
crPanUpDown = TCursor(-58);
```

File

imouse

Description

This is constant IMouse.crPanUpDown.

81.5.10 IMouse.crPanUpLeft Constant

Pascal

```
crPanUpLeft = TCursor(-59);
```

C++

```
crPanUpLeft = TCursor(-59);
```

File

imouse

Description

This is constant IMouse.crPanUpLeft.

81.5.11 IMouse.crPanUpRight Constant

Pascal

```
crPanUpRight = TCursor(-60);
```

C++

```
crPanUpRight = TCursor(-60);
```

File

imouse

Description

This is constant IMouse.crPanUpRight.

82 IniFiles Namespace

This is namespace IniFiles.

Topics

Name	Description
Classes (see page 15434)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 15475)	The following table lists structs, records, enums in this documentation.
Types (see page 15476)	The following table lists types in this documentation.

Classes

	Name	Description
	EIniFileException (see page 15434)	EIniFileException is the exception class for errors that occur when working with INI files using TIniFile (see page 15455).
	TCustomIniFile (see page 15434)	TCustomIniFile is the base class for components that encapsulate access to .ini files and the Windows system registry.
	THashedStringList (see page 15450)	THashedStringList maintains a list of strings using an internal hash table.
	TIniFile (see page 15455)	TIniFile stores and retrieves application-specific information and settings from INI files.
	TMemIniFile (see page 15462)	TMemIniFile enables buffered storage and retrieval of application-specific information and settings in an INI file.
	TStringHash (see page 15471)	This is class IniFiles.TStringHash.

Structs, Records, Enums

	Name	Description
	THashItem (see page 15475)	This is record IniFiles.THashItem.

Types

Name	Description
PHashItem (see page 15476)	This is type IniFiles.PHashItem.
PPHashItem (see page 15476)	This is type IniFiles.PPHashItem.

82.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EIniFileException (see page 15434)	EIniFileException is the exception class for errors that occur when working with INI files using TIniFile (see page 15455).
	TCustomIniFile (see page 15434)	TCustomIniFile is the base class for components that encapsulate access to .ini files and the Windows system registry.
	THashedStringList (see page 15450)	THashedStringList maintains a list of strings using an internal hash table.
	TIniFile (see page 15455)	TIniFile stores and retrieves application-specific information and settings from INI files.
	TMemIniFile (see page 15462)	TMemIniFile enables buffered storage and retrieval of application-specific information and settings in an INI file.
	TStringHash (see page 15471)	This is class IniFiles.TStringHash.

82.1.1 EIniFileException Class

EIniFileException is the exception class for errors that occur when working with INI files using TIniFile (see page 15455).

Class Hierarchy

Pascal

```
EIniFileException = class(Exception);
```

C++

```
class EIniFileException : public Exception;
```

File

IniFiles

Description

EIniFileException is raised when TIniFile (see page 15455) fails in an attempt to modify its associated INI file.

See Also

TIniFile (see page 15455)

82.1.2 TCustomIniFile Class

TCustomIniFile is the base class for components that encapsulate access to .ini files and the Windows system registry.

Class Hierarchy

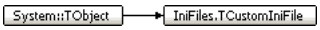

Pascal

```
TCustomIniFile = class(TObject);
```

C++

```
class TCustomIniFile : public TObject;
```

File

IniFiles

Description

Note: Use TCustomIniFile as a base class when defining components that access ini files. Ini files are text files that are divided into sections with variable assignments in each section. TCustomIniFile introduces properties and methods to handle storage and retrieval of application-specific information and settings in a standard ini file.

Note: On Windows, TCustomIniFile is also the base class for TRegistryIniFile, which allows applications that use both ini files and the system registry to handle both media in a unified manner. This differs from the features of TRegistry, which corresponds more directly to Windows system registry native semantics.

Do not create instances of TCustomIniFile. Instead, use or create descendants of TCustomIniFile, such as TIniFile (see page 15455), TRegistryIniFile, and TMemIniFile (see page 15462).

See Also

TIniFile (see page 15455)

TMemIniFile (see page 15462)

TRegistry (see page 18645)

TRegistryIniFile (see page 18675)

82.1.2.1 TCustomIniFile Members

The following tables list the members exposed by TCustomIniFile.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 15436)	Creates a TCustomIniFile (see page 15434) object for an application.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	DeleteKey (see page 15437)	Erases the data value associated with an ini file entry.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
A	EraseSection (see page 15438)	Erases an entire section of an ini file.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	ReadBinaryStream (see page 15438)	Retrieves a binary value from the ini file.
V	ReadBool (see page 15439)	Retrieves a Boolean value from an ini file.
V	ReadDate (see page 15439)	Retrieves a date value from an ini file.
V	ReadDateTime (see page 15440)	Retrieves a date-time value from an ini file.
V	ReadFloat (see page 15440)	Retrieves a float value from an ini file.
V	ReadInteger (see page 15441)	Retrieves an integer value from an ini file.
A	ReadSection (see page 15442)	Reads all the key names from a specified section of an ini file into a string list.
A	ReadSections (see page 15442)	This is the overview for the ReadSections method overload.
A	ReadSectionValues (see page 15443)	Reads the values from all keys in a section of an ini file into a string list.
A	ReadString (see page 15443)	Retrieves a string value from an ini file.
V	ReadTime (see page 15444)	Retrieves a time value from an ini file.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SectionExists (see page 15444)	Indicates whether a section exists in the ini file.
A	UpdateFile (see page 15445)	Flushes buffered ini file data to disk.
V	ValueExists (see page 15445)	Indicates whether a key exists in the ini file.
V	WriteBinaryStream (see page 15446)	This is WriteBinaryStream, a member of class TCustomIniFile.
V	WriteBool (see page 15446)	Writes a Boolean value to an ini file.
V	WriteDate (see page 15446)	Writes a date value to an ini file.
V	WriteDateTime (see page 15447)	Writes a date-time value to an ini file.
V	WriteFloat (see page 15447)	Writes a float value to an ini file.
V	WriteInteger (see page 15448)	Writes an integer value to an ini file.
A	WriteString (see page 15448)	Writes a string value to an ini file.
V	WriteTime (see page 15449)	Writes a time value to an ini file.

Public Properties

	Name	Description
R	FileName (see page 15450)	Contains the name of the ini file from which to read and to which to write information.

82.1.2.2 TCustomIniFile.Create Constructor

Creates a TCustomIniFile (see page 15434) object for an application.

Pascal

```
constructor Create(const FileName: string);
```

C++

```
__fastcall TCustomIniFile(const AnsiString FileName);
```

Description

Create constructs a TCustomIniFile (see page 15434) object for an application. Create assigns the FileName (see page

15450) parameter to the FileName (see page 15450) property, which is used to specify the name of the ini file to use.

See Also

FileName (see page 15450)

Create (see page 22287)

82.1.2.3 TCustomIniFile Methods

The methods of the TCustomIniFile class are listed here.

Public Methods

	Name	Description
	DeleteKey (see page 15437)	Erases the data value associated with an ini file entry.
	EraseSection (see page 15438)	Erases an entire section of an ini file.
	ReadBinaryStream (see page 15438)	Retrieves a binary value from the ini file.
	ReadBool (see page 15439)	Retrieves a Boolean value from an ini file.
	ReadDate (see page 15439)	Retrieves a date value from an ini file.
	ReadDateTime (see page 15440)	Retrieves a date-time value from an ini file.
	ReadFloat (see page 15440)	Retrieves a float value from an ini file.
	ReadInteger (see page 15441)	Retrieves an integer value from an ini file.
	ReadSection (see page 15442)	Reads all the key names from a specified section of an ini file into a string list.
	ReadSections (see page 15442)	This is the overview for the ReadSections method overload.
	ReadSectionValues (see page 15443)	Reads the values from all keys in a section of an ini file into a string list.
	ReadString (see page 15443)	Retrieves a string value from an ini file.
	ReadTime (see page 15444)	Retrieves a time value from an ini file.
	SectionExists (see page 15444)	Indicates whether a section exists in the ini file.
	UpdateFile (see page 15445)	Flushes buffered ini file data to disk.
	ValueExists (see page 15445)	Indicates whether a key exists in the ini file.
	WriteBinaryStream (see page 15446)	This is WriteBinaryStream, a member of class TCustomIniFile.
	WriteBool (see page 15446)	Writes a Boolean value to an ini file.
	WriteDate (see page 15446)	Writes a date value to an ini file.
	WriteDateTime (see page 15447)	Writes a date-time value to an ini file.
	WriteFloat (see page 15447)	Writes a float value to an ini file.
	WriteInteger (see page 15448)	Writes an integer value to an ini file.
	WriteString (see page 15448)	Writes a string value to an ini file.
	WriteTime (see page 15449)	Writes a time value to an ini file.

82.1.2.3.1 TCustomIniFile.DeleteKey Method

Erases the data value associated with an ini file entry.

Pascal

```
procedure DeleteKey(const Section: String; const Ident: String); virtual; abstract;
```

C++

```
virtual __fastcall DeleteKey(const AnsiString Section, const AnsiString Ident) = 0;
```

Description

Call DeleteKey to erase a data value associated with an ini file entry. Section is string containing the name of an ini file section, and Ident is a string containing the name of the key for which to set a nil (Delphi) or NULL (C++) value.

Note: Attempting to erase a data value in a nonexistent section or attempting to erase data from a nonexistent key are not errors. In these cases, DeleteKey creates the section and key and sets its initial value to an empty value.

See Also

EraseSection (see page 15438)

82.1.2.3.2 TCustomIniFile.EraseSection Method

Erases an entire section of an ini file.

Pascal

```
procedure EraseSection(const Section: string); virtual; abstract;
```

C++

```
virtual __fastcall EraseSection(const AnsiString Section) = 0;
```

Description

Call EraseSection to remove a section, all its keys, and their data values from an ini file. Section identifies the ini file section to remove. If a section cannot be removed, an exception is raised.

See Also

SectionExists (see page 15444)

DeleteKey (see page 15437)

82.1.2.3.3 TCustomIniFile.ReadBinaryStream Method

Retrieves a binary value from the ini file.

Pascal

```
function ReadBinaryStream(const Section: string; const Name: string; Value: TStream): Integer;
virtual;
```

C++

```
virtual __fastcall int ReadBinaryStream(const AnsiString Section, const AnsiString Name,
TStream Value);
```

Description

Call ReadBinaryStream to read a binary value into a specified stream. After reading the binary value, which it converts from a series of hexadecimal characters, from the ini file into the current position of the stream, ReadBinaryStream returns the number of bytes written to the stream. If the stream is a memory stream (TMemoryStream or one of its descendants), the stream is left positioned to the same byte as it was before the call to ReadBinaryStream (immediately before the new binary value).

Section is the section that contains the desired key.

Name is the key under which the binary data is stored.

Value is a stream to which the binary value is written.

Note: When used with a Windows .ini file, ReadBinaryStream is limited to 1023 characters,

See Also

ReadString (see page 15443)

writebinarystream (see page 15446)

82.1.2.3.4 TCustomIniFile.ReadBool Method

Retrieves a Boolean value from an ini file.

Pascal

```
function ReadBool(const Section: string; const Ident: string; Default: Boolean): Boolean;
virtual;
```

C++

```
virtual __fastcall Boolean ReadBool(const AnsiString Section, const AnsiString Ident, Boolean
Default);
```

Description

Call ReadBool to read a Boolean value from an ini file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the Boolean value. Default is the Boolean value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadDate (see page 15439)

ReadDateTime (see page 15440)

ReadFloat (see page 15440)

ReadInteger (see page 15441)

ReadString (see page 15443)

ReadTime (see page 15444)

82.1.2.3.5 TCustomIniFile.ReadDate Method

Retrieves a date value from an ini file.

Pascal

```
function ReadDate(const Section: string; const Name: string; Default: TDateTime): TDateTime;
virtual;
```

C++

```
virtual __fastcall TDateTime ReadDate(const AnsiString Section, const AnsiString Name,
TDateTime Default);
```

Description

Call ReadDate to read a date value from an ini file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the date value. Default is the date value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 15439)

ReadDateTime (see page 15440)

ReadFloat (see page 15440)

ReadInteger (see page 15441)

ReadString (see page 15443)

ReadTime (see page 15444)

82.1.2.3.6 TCustomIniFile.ReadDateTime Method

Retrieves a date-time value from an ini file.

Pascal

```
function ReadDateTime(const Section: string; const Name: string; Default: TDateTime):
TDateTime; virtual;
```

C++

```
virtual __fastcall TDateTime ReadDateTime(const AnsiString Section, const AnsiString Name,
TDateTime Default);
```

Description

Call ReadDateTime to read a date-time value from an ini file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the date-time value. Default is the date-time value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 15439)

ReadDate (see page 15439)

ReadDateTime

ReadFloat (see page 15440)

ReadInteger (see page 15441)

ReadString (see page 15443)

ReadTime (see page 15444)

82.1.2.3.7 TCustomIniFile.ReadFloat Method

Retrieves a float value from an ini file.

Pascal

```
function ReadFloat(const Section: string; const Name: string; Default: Double): Double;
virtual;
```

C++

```
virtual __fastcall Double ReadFloat(const AnsiString Section, const AnsiString Name, Double
Default);
```

Description

Call ReadFloat to read a float value from an ini file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the float value. Default is the float value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 15439)

ReadDate (see page 15439)

ReadDateTime (see page 15440)

ReadInteger (see page 15441)

ReadString (see page 15443)

ReadTime (see page 15444)

82.1.2.3.8 TCustomIniFile.ReadInteger Method

Retrieves an integer value from an ini file.

Pascal

```
function ReadInteger(const Section: string; const Ident: string; Default: Longint): Longint;
virtual;
```

C++

```
virtual __fastcall Longint ReadInteger(const AnsiString Section, const AnsiString Ident,
Longint Default);
```

Description

Call ReadInteger to read an integer value from an ini file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the value. Default is the integer value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 15439)

ReadDate (see page 15439)

ReadDateTime (see page 15440)

ReadFloat (see page 15440)

ReadString (see page 15443)

ReadTime (see page 15444)

82.1.2.3.9 TCustomIniFile.ReadSection Method

Reads all the key names from a specified section of an ini file into a string list.

Pascal

```
procedure ReadSection(const Section: string; Strings: TStrings); virtual; abstract;
```

C++

```
virtual __fastcall ReadSection(const AnsiString Section, TStrings Strings) = 0;
```

Description

Call ReadSection to retrieve the names of all keys within a specified section of an ini file into a string list object.

Section identifies the section from which to retrieve a list of key names. Strings specifies the string object to hold the retrieved names. Strings can point to a TStrings descendant such as a string list, or to a component property, such as Items for a TListBox component.

See Also

SectionExists (see page 15444)

ReadSections (see page 15442)

ReadSectionValues (see page 15443)

82.1.2.3.10 ReadSections Method

This is the overview for the ReadSections method overload.

Overload List

	Name	Description
	TCustomIniFile.ReadSections (string, TStrings) (see page 15442)	Reads the names of all sections in an ini file into a string list.
	TCustomIniFile.ReadSections (TStrings) (see page 15442)	Reads the names of all sections in an ini file into a string list.

82.1.2.3.10.1 TCustomIniFile.ReadSections Method (TStrings)

Reads the names of all sections in an ini file into a string list.

Pascal

```
procedure ReadSections(Strings: TStrings); virtual; abstract; overload;
procedure ReadSections(const Section: string; Strings: TStrings); virtual; overload;
```

C++

```
virtual __fastcall ReadSections(TStrings Strings) = 0;
virtual __fastcall ReadSections(const AnsiString Section, TStrings Strings);
```

Description

Call ReadSections to retrieve the names of all sections in an ini file into a string list.

Strings specifies the string object to hold the retrieved names. Strings can point to a TStrings descendant, or to a component property, such as Items for a TListBox component.

Section identifies the section from which to retrieve a list of key names.

See Also

ReadSection (see page 15442)

ReadSectionValues (see page 15443)

82.1.2.3.11 TCustomIniFile.ReadSectionValues Method

Reads the values from all keys in a section of an ini file into a string list.

Pascal

```
procedure ReadSectionValues(const Section: string; Strings: TStrings); virtual; abstract;
```

C++

```
virtual __fastcall ReadSectionValues(const AnsiString Section, TStrings Strings) = 0;
```

Description

Call ReadSectionValues to read the values from all keys in a specified section of an ini file into a string list object. The values are added to the string list as strings of the form KeyName=ValueName.

Section identifies the section in the file from which to read key values.

Strings is the string list object into which to read the values. Strings can point to a TStrings object, or to a component property, such as Items for a TListBox component.

See Also

SectionExists (see page 15444)

ReadSection (see page 15442)

ReadSections (see page 15442)

82.1.2.3.12 TCustomIniFile.ReadString Method

Retrieves a string value from an ini file.

Pascal

```
function ReadString(const Section: string; const Ident: string; const Default: string):
string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString ReadString(const AnsiString Section, const AnsiString Ident,
const AnsiString Default) = 0;
```

Description

Call ReadString to read a string value from a .ini file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the value. Default is the string value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

Note: On Windows, if the string in the .ini file is surrounded by quotation marks, the return value has those marks stripped away. On Linux, the quotation marks are preserved.

See Also

ReadBool (see page 15439)

ReadDate (see page 15439)

ReadDateTime (see page 15440)

ReadFloat (see page 15440)

ReadInteger (see page 15441)

ReadTime (see page 15444)

ReadBinaryStream (see page 15438)

82.1.2.3.13 TCustomIniFile.ReadTime Method

Retrieves a time value from an ini file.

Pascal

```
function ReadTime(const Section: string; const Name: string; Default: TDateTime): TDateTime;
virtual;
```

C++

```
virtual __fastcall TDateTime ReadTime(const AnsiString Section, const AnsiString Name,
TDateTime Default);
```

Description

Call ReadTime to read a time value from an ini file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the time value. Default is the time value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 15439)

ReadDate (see page 15439)

ReadDateTime (see page 15440)

ReadFloat (see page 15440)

ReadInteger (see page 15441)

ReadString (see page 15443)

82.1.2.3.14 TCustomIniFile.SectionExists Method

Indicates whether a section exists in the ini file.

Pascal

```
function SectionExists(const Section: string): Boolean;
```

C++

```
__fastcall Boolean SectionExists(const AnsiString Section);
```

Description

Use SectionExists to determine whether a section exists within the ini file specified in FileName (see page 15450).

Section is the ini file section SectionExists determines the existence of.

SectionExists returns a Boolean value that indicates whether the section in question exists.

See Also

FileName (see page 15450)

ValueExists (see page 15445)

82.1.2.3.15 TCustomIniFile.UpdateFile Method

Flushes buffered ini file data to disk.

Pascal

```
procedure UpdateFile; virtual; abstract;
```

C++

```
virtual __fastcall UpdateFile() = 0;
```

Description

Call UpdateFile to flush buffered reads from and writes to the ini file to disk. [SK comment -- UpdateFile is useful under Windows95but has no effect under Windows NT as NT does not buffer ini file reads and writes. do not delete!]

See Also

FileName (see page 15450)

82.1.2.3.16 TCustomIniFile.ValueExists Method

Indicates whether a key exists in the ini file.

Pascal

```
function ValueExists(const Section: string; const Ident: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ValueExists(const AnsiString Section, const AnsiString Ident);
```

Description

Use ValueExists to determine whether a key exists in the ini file specified in FileName (see page 15450).

Section is the section in the ini file in which to search for the key.

Ident is the name of the key to search for.

ValueExists returns a boolean value that indicates whether the key exists in the specified section.

See Also

FileName (see page 15450)

SectionExists (see page 15444)

82.1.2.3.17 TCustomIniFile.WriteBinaryStream Method

Pascal

```
procedure WriteBinaryStream(const Section: string; const Name: string; Value: TStream);
virtual;
```

C++

```
virtual __fastcall WriteBinaryStream(const AnsiString Section, const AnsiString Name, TStream
Value);
```

Description

This is WriteBinaryStream, a member of class TCustomIniFile.

82.1.2.3.18 TCustomIniFile.WriteBool Method

Writes a Boolean value to an ini file.

Pascal

```
procedure WriteBool(const Section: string; const Ident: string; Value: Boolean); virtual;
```

C++

```
virtual __fastcall WriteBool(const AnsiString Section, const AnsiString Ident, Boolean Value);
```

Description

Call WriteBool to write a Boolean value to an ini file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the Boolean value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteBool creates the section and key and sets its initial value to Value.

See Also

WriteDate (see page 15446)

WriteDateTime (see page 15447)

WriteFloat (see page 15447)

WriteInteger (see page 15448)

WriteString (see page 15448)

WriteTime (see page 15449)

82.1.2.3.19 TCustomIniFile.WriteDate Method

Writes a date value to an ini file.

Pascal

```
procedure WriteDate(const Section: string; const Name: string; Value: TDateTime); virtual;
```

C++

```
virtual __fastcall WriteDate(const AnsiString Section, const AnsiString Name, TDateTime Value);
```

Description

Call WriteDate to write a date value to an ini file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the date value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteDate creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 15446)

WriteDateTime (see page 15447)

WriteFloat (see page 15447)

WriteInteger (see page 15448)

WriteString (see page 15448)

WriteTime (see page 15449)

82.1.2.3.20 TCustomIniFile.WriteDateTime Method

Writes a date-time value to an ini file.

Pascal

```
procedure WriteDateTime(const Section: string; const Name: string; Value: TDateTime); virtual;
```

C++

```
virtual __fastcall WriteDateTime(const AnsiString Section, const AnsiString Name, TDateTime
Value);
```

Description

Call WriteDateTime to write a date-time value to an ini file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the date-time value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteDateTime creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 15446)

WriteDate (see page 15446)

WriteFloat (see page 15447)

WriteInteger (see page 15448)

WriteString (see page 15448)

WriteTime (see page 15449)

82.1.2.3.21 TCustomIniFile.WriteFloat Method

Writes a float value to an ini file.

Pascal

```
procedure WriteFloat(const Section: string; const Name: string; Value: Double); virtual;
```

C++

```
virtual __fastcall WriteFloat(const AnsiString Section, const AnsiString Name, Double Value);
```

Description

Call WriteFloat to write a float value to an ini file. Section identifies the section in the file that contain the key to which to write.

Ident is the name of the key for which to set a value. Value is the float value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteFloat creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 15446)

WriteDate (see page 15446)

WriteDateTime (see page 15447)

WriteInteger (see page 15448)

WriteString (see page 15448)

WriteTime (see page 15449)

82.1.2.3.22 TCustomIniFile.WriteInteger Method

Writes an integer value to an ini file.

Pascal

```
procedure WriteInteger(const Section: string; const Ident: string; Value: Longint); virtual;
```

C++

```
virtual __fastcall WriteInteger(const AnsiString Section, const AnsiString Ident, Longint
Value);
```

Description

Call WriteInteger to write an integer value to an ini file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the integer value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteInteger creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 15446)

WriteDate (see page 15446)

WriteDateTime (see page 15447)

WriteFloat (see page 15447)

WriteString (see page 15448)

WriteTime (see page 15449)

82.1.2.3.23 TCustomIniFile.WriteString Method

Writes a string value to an ini file.

Pascal

```
procedure WriteString(const Section: String; const Ident: String; const Value: String);
virtual; abstract;
```

C++

```
virtual __fastcall WriteString(const AnsiString Section, const AnsiString Ident, const
```

```
AnsiString Value) = 0;
```

Description

Call WriteString to write a string value to an ini file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the string value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteString creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 15446)

WriteDate (see page 15446)

WriteDateTime (see page 15447)

WriteFloat (see page 15447)

WriteInteger (see page 15448)

WriteTime (see page 15449)

82.1.2.3.24 TCustomIniFile.WriteTime Method

Writes a time value to an ini file.

Pascal

```
procedure WriteTime(const Section: string; const Name: string; Value: TDateTime); virtual;
```

C++

```
virtual __fastcall WriteTime(const AnsiString Section, const AnsiString Name, TDateTime Value);
```

Description

Call WriteTime to write a time value to an ini file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the time value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteTime creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 15446)

WriteDate (see page 15446)

WriteDateTime (see page 15447)

WriteFloat (see page 15447)

WriteInteger (see page 15448)

WriteString (see page 15448)

82.1.2.4 TCustomIniFile Properties

The properties of the TCustomIniFile class are listed here.

Public Properties

	Name	Description
R	FileName (see page 15450)	Contains the name of the ini file from which to read and to which to write information.

82.1.2.4.1 TCustomIniFile.FileName Property

Contains the name of the ini file from which to read and to which to write information.

Pascal

```
property FileName: string;
```

C++

```
__property AnsiString FileName;
```

Description

Read FileName to retrieve the name of the ini file that this object can be used to access. The methods that read and write ini file information automatically handle opening and closing the file specified by FileName.

FileName is read-only. The name of the ini file the object uses is supplied as the parameter for the constructor.

See Also

Create (see page 15436)

82.1.3 THashedStringList Class

THashedStringList maintains a list of strings using an internal hash table.

Class Hierarchy

Pascal

```
THashedStringList = class(TStringList);
```

C++

```
class THashedStringList : public TStringList;
```

File

IniFiles

Description

THashedStringList is a string list that uses a hash table internally to speed the process of locating strings. It is used internally by TMemIniFile (see page 15462) to manage the strings from an INI file, but can be used in the same way as any other string list. By using THashedStringList instead of TStringList, you can improve performance when the list contains a large number of strings.

See Also

TStringList (see page 3006)

TMemIniFile (see page 15462)

Working with Lists

82.1.3.1 THashedStringList Members

The following tables list the members exposed by THashedStringList.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 15454)	This is Changed, a member of class THashedStringList.
	Changing (see page 3013)	This is Changing, a member of class TStringList.
	CompareStrings (see page 3014)	Compares two strings.
	CompareStrings (see page 3048)	Compares two strings.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 3049)	Reads and writes the Strings (see page 3080) property as if it were published.
	Error (see page 3052)	This is the overview for the Error method overload.
	ExtractName (see page 3053)	Returns the name portion of a string that is a name value pair.
	Get (see page 3017)	Returns a string given its index.
	Get (see page 3053)	Returns a string given its index.
	GetCapacity (see page 3017)	Returns the currently allocated size of the list of strings.
	GetCapacity (see page 3054)	Returns the currently allocated size of the list of strings.
	GetCount (see page 3018)	Returns the number of strings in the list
	GetCount (see page 3054)	Returns the number of strings in the list
	GetObject (see page 3018)	Returns the object associated with the string at a specified index.
	GetObject (see page 3055)	Returns the object associated with the string at a specified index.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTextStr (see page 3056)	Returns the value of the Text (see page 3084) property.
	InsertItem (see page 3020)	This is InsertItem, a member of class TStringList.
	Put (see page 3022)	Changes the value of the string with a specified index.
	Put (see page 3064)	Changes the value of the string with a specified index.
	PutObject (see page 3022)	Changes the object associated with the string at a specified index.
	PutObject (see page 3065)	Changes the object associated with the string at a specified index.
	SetCapacity (see page 3022)	Changes the amount of memory allocated to hold strings in the list.
	SetCapacity (see page 3068)	Changes the amount of memory allocated to hold strings in the list.
	SetTextStr (see page 3069)	Sets the Text (see page 3084) property.
	SetUpdateState (see page 3023)	Performs internal adjustments before or after a series of updates.
	SetUpdateState (see page 3069)	Performs internal adjustments before or after a series of updates.

Protected Properties

	Name	Description
	UpdateCount (see page 3085)	Indicates the number of calls to BeginUpdate (see page 3045) that have not been matched by a call to EndUpdate (see page 3050).

Public Events

	Name	Description
	OnChange (see page 3028)	Occurs immediately after the list of strings changes.
	OnChanging (see page 3028)	Occurs immediately before the list of strings changes.

Public Methods

	Name	Description
	Add (see page 3010)	Adds a new string to the list.
	Add (see page 3033)	Adds a string at the end of the list.
	AddObject (see page 3012)	Adds a string to the list, and associates an object with the string.
	AddObject (see page 3036)	Adds a string to the list, and associates an object with the string.
	AddStrings (see page 3039)	Adds a group of strings to the list.

V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 3041)	Adds a string to the list.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	Assign (see page 3043)	Sets the strings in the list, and possibly associated objects, from a source object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 3045)	Enables the TStrings (see page 3029) object to track when the list of strings is changing.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 3013)	Deletes all the strings from the list.
A	Clear (see page 3047)	Introduces an abstract (Delphi) or pure virtual (C++) method to empty the list and any associated objects.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	CustomSort (see page 3014)	Sorts the strings in the list in a customized order.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Delete (see page 3015)	Removes the string specified by the Index parameter.
A	Delete (see page 3049)	Introduces an abstract (Delphi) or pure virtual (C++) method to delete a specified string from the list.
V	Destroy (see page 15453)	Destroys an instance of THashedStringList (see page 15450).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 3050)	Enables the TStrings (see page 3029) object to keep track of when the list of strings has finished changing.
	Equals (see page 3051)	Compares the list of strings to the list from another TStrings (see page 3029) object and returns true if the two lists match.
V	Exchange (see page 3015)	Swaps the position of two strings in the list.
V	Exchange (see page 3053)	Swaps the position of two strings in the list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
V	Find (see page 3016)	Locates the index for a string in a sorted list and indicates whether a string with that value already exists in the list.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 3054)	Returns a TStrings (see page 3029) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetText (see page 3055)	Allocates a text buffer and fills it with the value of the Text (see page 3084) property.
V	IndexOf (see page 15454)	Returns the position of a string in the list.
V	IndexOfName (see page 15455)	Returns the position of the first string with the form Name=Value with the specified name part.
V	IndexOfObject (see page 3059)	Returns the index of the first string in the list associated with a given object.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Insert (see page 3019)	Inserts a string to the list at the position specified by Index.
A	Insert (see page 3059)	Introduces abstract (Delphi) or pure virtual (C++) method to insert a string at a specified position.
V	InsertObject (see page 3020)	Inserts a string into the list at the specified position, and associates it with an object.

V	InsertObject (see page 3060)	Inserts a string into the list at the specified position, and associates it with an object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	LoadFromFile (see page 3061)	Fills the list with the lines of text in a specified file.
V	LoadFromStream (see page 3062)	Fills the list with lines of text read from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	Move (see page 3063)	Changes the position of a string in the list.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SaveToFile (see page 3065)	Saves the strings in the list to the specified file.
V	SaveToStream (see page 3067)	Writes the value of the Text (see page 3084) property to a stream object.
V	SetText (see page 3068)	Sets the Text (see page 3084) property.
V	Sort (see page 3023)	Sorts the strings in the list in ascending order.

Public Properties

	Name	Description
	Capacity (see page 3070)	Indicates the number of strings the TStrings (see page 3029) object can hold.
	CaseSensitive (see page 3025)	Controls (see page 5711) whether strings are located, sorted, and identified as duplicates in a case-sensitive or case-insensitive manner.
	CommaText (see page 3070)	Lists the strings in the TStrings (see page 3029) object in system data format (SDF).
R	Count (see page 3071)	Introduces an abstract property to represent the number of strings in the list.
	DelimitedText (see page 3074)	Represents all the strings in the TStrings (see page 3029) object as a single delimited string.
	Delimiter (see page 3074)	Specifies the delimiter used by the DelimitedText (see page 3074) property.
	Duplicates (see page 3025)	Specifies whether duplicate strings can be added to sorted lists.
	LineBreak (see page 3075)	Defines the line break character(s).
R	Names (see page 3075)	Indicates the name part of strings that are name-value pairs.
	NameValueSeparator (see page 3077)	Indicates the character used to separate names from values.
	Objects (see page 3077)	Represents a set of objects that are associated one with each of the strings in the Strings (see page 3080) property.
	QuoteChar (see page 3080)	Specifies the quote character used by the DelimitedText (see page 3074) property.
	Sorted (see page 3026)	Specifies whether the strings in the list should be automatically sorted.
	StrictDelimiter (see page 3080)	Determines how the Delimiter (see page 3074) property is used.
	Strings (see page 3080)	References the strings in the list by their positions.
	StringsAdapter (see page 3084)	Implements an IStringsAdapter (see page 3208) interface for the TStrings (see page 3029) object.
	Text (see page 3084)	Lists the strings in the TStrings (see page 3029) object as a single string with the individual strings delimited by carriage returns and line feeds.
	ValueFromIndex (see page 3085)	Represents the value part of a string with a given index, on strings that are name-value pairs.
	Values (see page 3086)	Represents the value part of a string associated with a given name, on strings that are name-value pairs.

82.1.3.2 THashedStringList.Destroy Destructor

Destroys an instance of THashedStringList (see page 15450).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THashedStringList();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the THashedStringList (see page 15450) reference is not nil and only then calls Destroy.

See Also

Free (see page 22300)

82.1.3.3 THashedStringList Methods

The methods of the THashedStringList class are listed here.

Protected Methods

	Name	Description
	Changed (see page 15454)	This is Changed, a member of class THashedStringList.

Public Methods

	Name	Description
	IndexOf (see page 15454)	Returns the position of a string in the list.
	IndexOfName (see page 15455)	Returns the position of the first string with the form Name=Value with the specified name part.

82.1.3.3.1 THashedStringList.Changed Method

Pascal

```
procedure Changed; override;
```

C++

```
virtual __fastcall Changed();
```

Description

This is Changed, a member of class THashedStringList.

82.1.3.3.2 THashedStringList.IndexOf Method

Returns the position of a string in the list.

Pascal

```
function IndexOf(const S: string): Integer; override;
```

C++

```
virtual __fastcall int IndexOf(const AnsiString S);
```

Description

Call IndexOf to obtain the position of the first occurrence of a string that matches S. A string matches S if it is identical to S or differs only in case.

IndexOf returns the 0-based index of the string. Thus, if S matches the first string in the list, IndexOf returns 0, if S is the second string, IndexOf returns 1, and so on. If the string does not have a match in the string list, IndexOf returns -1.

See Also

Find (see page 3016)

IndexOfName (see page 15455)

IndexOfObject (see page 3059)

Strings

82.1.3.3.3 THashedStringList.IndexOfName Method

Returns the position of the first string with the form Name=Value with the specified name part.

Pascal

```
function IndexOfName(const Name: string): Integer; override;
```

C++

```
virtual __fastcall int IndexOfName(const AnsiString Name);
```

Description

Call IndexOfName to locate the first occurrence of a string with the form Name=Value where the name part is equal to the Name parameter or differs only in case. IndexOfName returns the 0-based index of the string. If no string in the list has the indicated name, IndexOfName returns -1.

The Name that identifies the string is to the left of the equal sign (=), and the current Value of the Name identifier is on the right side. There should be no spaces present before or after the equal sign.

Note: If there is more than one string with a name portion matching the Name parameter, IndexOfName returns the position of the first such string.

See Also

IndexOf (see page 15454)

Names (see page 3075)

Values (see page 3086)

IndexOfObject (see page 3059)

82.1.4 TIniFile Class

TIniFile stores and retrieves application-specific information and settings from INI files.

Class Hierarchy

Pascal

```
TIniFile = class(TCustomIniFile);
```

C++

```
class TIniFile : public TCustomIniFile;
```

File

IniFiles

Description

TIniFile enables handling the storage and retrieval of application-specific information and settings in a standard INI file. An INI file stores information in logical groupings, called "sections." Within each section, actual data values are stored in named keys. Keys take the form:

```
<keyname>=<value>
```

A FileName (see page 15450) is passed to the TIniFile constructor and identifies the INI file that the object accesses.

Note: On Windows, a related object, TMemIniFile (see page 15462), works the same way as TIniFile, but buffers writes in memory to minimize disk access.

Note: On Linux, TIniFile is the same as TMemIniFile (see page 15462).

See Also

TRegistry (see page 18645)

TRegistryIniFile (see page 18675)

82.1.4.1 TIniFile Members

The following tables list the members exposed by TIniFile.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 15436)	Creates a TCustomIniFile (see page 15434) object for an application.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DeleteKey (see page 15458)	Removes an INI file entry for a specified key value.
	Destroy (see page 15457)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EraseSection (see page 15458)	Erases an entire section of an INI file.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ReadBinaryStream (see page 15438)	Retrieves a binary value from the ini file.

	ReadBool (see page 15439)	Retrieves a Boolean value from an ini file.
	ReadDate (see page 15439)	Retrieves a date value from an ini file.
	ReadDateTime (see page 15440)	Retrieves a date-time value from an ini file.
	ReadFloat (see page 15440)	Retrieves a float value from an ini file.
	ReadInteger (see page 15441)	Retrieves an integer value from an ini file.
	ReadSection (see page 15459)	Reads all the key names from a specified section of an INI file into a string list.
	ReadSections (see page 15459)	Reads the names of all sections in an INI file into a string list.
	ReadSections (see page 15442)	This is the overview for the ReadSections method overload.
	ReadSectionValues (see page 15460)	Reads the values from all keys within a section of an INI file into a string list.
	ReadString (see page 15460)	Retrieves a string value from an INI file.
	ReadTime (see page 15444)	Retrieves a time value from an ini file.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SectionExists (see page 15444)	Indicates whether a section exists in the ini file.
	UpdateFile (see page 15461)	Flushes buffered INI file data to disk.
	ValueExists (see page 15445)	Indicates whether a key exists in the ini file.
	WriteBinaryStream (see page 15446)	This is WriteBinaryStream, a member of class TCustomIniFile.
	WriteBool (see page 15446)	Writes a Boolean value to an ini file.
	WriteDate (see page 15446)	Writes a date value to an ini file.
	WriteDateTime (see page 15447)	Writes a date-time value to an ini file.
	WriteFloat (see page 15447)	Writes a float value to an ini file.
	WriteInteger (see page 15448)	Writes an integer value to an ini file.
	WriteString (see page 15461)	Writes a string value to an INI file.
	WriteTime (see page 15449)	Writes a time value to an ini file.

Public Properties

	Name	Description
	FileName (see page 15450)	Contains the name of the ini file from which to read and to which to write information.

82.1.4.2 TIniFile.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIniFile();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the

object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

82.1.4.3 TIniFile Methods

The methods of the TIniFile class are listed here.

Public Methods

	Name	Description
	DeleteKey (see page 15458)	Removes an INI file entry for a specified key value.
	EraseSection (see page 15458)	Erases an entire section of an INI file.
	ReadSection (see page 15459)	Reads all the key names from a specified section of an INI file into a string list.
	ReadSections (see page 15459)	Reads the names of all sections in an INI file into a string list.
	ReadSectionValues (see page 15460)	Reads the values from all keys within a section of an INI file into a string list.
	ReadString (see page 15460)	Retrieves a string value from an INI file.
	UpdateFile (see page 15461)	Flushes buffered INI file data to disk.
	WriteString (see page 15461)	Writes a string value to an INI file.

82.1.4.3.1 TIniFile.DeleteKey Method

Removes an INI file entry for a specified key value.

Pascal

```
procedure DeleteKey(const Section: String; const Ident: String); override;
```

C++

```
virtual __fastcall DeleteKey(const AnsiString Section, const AnsiString Ident);
```

Description

Call DeleteKey to erase an INI file entry. Section is string containing the name of an INI file section, and Ident is a string containing the name of the key from for which to set a nil (Delphi) or NULL (C++) value.

This version of DeleteKey is only available if the application is compiled for Windows. On Linux, DeleteKey is inherited from TMemIniFile (see page 15462).

Note: Attempting to delete a key in a nonexistent section or attempting to delete a nonexistent key are not errors. In these cases, DeleteKey does nothing.

See Also

EraseSection (see page 15458)

82.1.4.3.2 TIniFile.EraseSection Method

Erases an entire section of an INI file.

Pascal

```
procedure EraseSection(const Section: string); override;
```

C++

```
virtual __fastcall EraseSection(const AnsiString Section);
```

Description

Call EraseSection to remove a section, all its keys, and their data values from an INI file. Section identifies the INI file section to remove. If a section cannot be removed, an exception is raised.

This version of EraseSection is only available if the application is compiled for Windows. On Linux, EraseSection is inherited from TMemIniFile (see page 15462).

See Also

DeleteKey (see page 15458)

82.1.4.3.3 TIniFile.ReadSection Method

Reads all the key names from a specified section of an INI file into a string list.

Pascal

```
procedure ReadSection(const Section: string; Strings: TStrings); override;
```

C++

```
virtual __fastcall ReadSection(const AnsiString Section, TStrings Strings);
```

Description

Call ReadSection to retrieve the names of all keys within a specified section of an INI file into a string list object.

Section identifies the section from which to retrieve a list of key names. Strings specifies the TStrings descendant to hold the retrieved names. Strings can point to a TStrings descendant such as a string list (TStringList), or to a component property that is of type TStrings, such as the Items property of TListBox.

This version of ReadSection is only available if the application is compiled for Windows. On Linux, ReadSection is inherited from TMemIniFile (see page 15462).

See Also

ReadSections (see page 15459)

ReadSectionValues (see page 15460)

82.1.4.3.4 TIniFile.ReadSections Method

Reads the names of all sections in an INI file into a string list.

Pascal

```
procedure ReadSections(Strings: TStrings); override;
```

C++

```
virtual __fastcall ReadSections(TStrings Strings);
```

Description

Call ReadSections to retrieve the names of all sections in an INI file into a string list object.

Strings specifies the string object to hold the retrieved names. Strings can point to a TStrings object, or to a component property, such as Items for a TListBox component.

This version of ReadSections is only available if the application is compiled for Windows. On Linux, ReadSections is inherited from TMemIniFile (see page 15462).

See Also

ReadSection (see page 15459)

ReadSectionValues (see page 15460)

82.1.4.3.5 TIniFile.ReadSectionValues Method

Reads the values from all keys within a section of an INI file into a string list.

Pascal

```
procedure ReadSectionValues(const Section: string; Strings: TStrings); override;
```

C++

```
virtual __fastcall ReadSectionValues(const AnsiString Section, TStrings Strings);
```

Description

Call ReadSectionValues to read the keys, and the values from all keys, within a specified section of an INI file into a string list object (TStrings descendant). The key names and values are added to the string list as strings of the form KeyName=ValueName.

Section identifies the section in the file from which to read key values.

Strings is the string list object into which to read the values. Strings can point to a TStrings object, or to a component property, such as Items for a TListBox component, that is a TStrings object itself.

This version of ReadSectionValues is only available if the application is compiled for Windows. On Linux, ReadSectionValues is inherited from TMemIniFile (see page 15462).

See Also

ReadSection (see page 15459)

ReadSections (see page 15459)

82.1.4.3.6 TIniFile.ReadString Method

Retrieves a string value from an INI file.

Pascal

```
function ReadString(const Section: string; const Ident: string; const Default: string):
string; override;
```

C++

```
virtual __fastcall AnsiString ReadString(const AnsiString Section, const AnsiString Ident,
const AnsiString Default);
```

Description

Call ReadString to read a string value from an INI file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the value. Default is the string value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

This version of ReadString is only available if the application is compiled for Windows. On Linux, ReadString is inherited from TMemIniFile (see page 15462).

See Also

ReadBool (see page 15439)

ReadInteger (see page 15441)

82.1.4.3.7 TIniFile.UpdateFile Method

Flushes buffered INI file data to disk.

Pascal

```
procedure UpdateFile; override;
```

C++

```
virtual __fastcall UpdateFile();
```

Description

Call UpdateFile to flush buffered reads from and writes to the INI file to disk. UpdateFile is useful under Windows9x, but has no effect under Windows NT because NT does not buffer INI file reads and writes.

This version of UpdateFile is only available if the application is compiled for Windows. On Linux, Update is inherited from TMemIniFile (see page 15462).

See Also

FileName (see page 15450)

TMemIniFile (see page 15462)

82.1.4.3.8 TIniFile.WriteString Method

Writes a string value to an INI file.

Pascal

```
procedure WriteString(const Section: String; const Ident: String; const Value: String);
override;
```

C++

```
virtual __fastcall WriteString(const AnsiString Section, const AnsiString Ident, const
AnsiString Value);
```

Description

Call WriteString to write a string value to an INI file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the string value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteString creates the section and key and sets its initial value to Value.

This version of WriteString is only available if the application is compiled for Windows. On Linux, WriteString is inherited from TMemIniFile (see page 15462).

See Also

WriteBool (see page 15446)

WriteInteger (see page 15448)

82.1.5 TMemIniFile Class

TMemIniFile enables buffered storage and retrieval of application-specific information and settings in an INI file.

Class Hierarchy

Pascal

```
TMemIniFile = class(TCustomIniFile);
```

C++

```
class TMemIniFile : public TCustomIniFile;
```

File

IniFiles

Description

Use TMemIniFile to store and retrieve application-specific information and settings in a Windows INI file. An INI file stores information in logical groupings, called "sections." Within each section, actual data values are stored in named keys. Keys take the form:

```
<keyname>=<value>
```

TMemIniFile buffers all changes to the INI file. The INI file is read once, when the object is first created. Data from the INI file is stored in nested string lists. Each section in the INI file occupies one element in the top-most string list, and each element in this may itself contain a string list. Each element in each of the contained string list represents a key within the section. After the data is read, any changes to the data are stored in memory. To write the data from memory back to the associated INI file, call the UpdateFile (see page 15470) method.

All TMemIniFile methods to read, write, and erase sections, keys, and values operate on the in-memory copy of the INI file.

See Also

TStrings (see page 3029)

TIniFile (see page 15455)

TCustomIniFile (see page 15434)

TRegistry (see page 18645)

TRegistryIniFile (see page 18675)

82.1.5.1 TMemIniFile Members

The following tables list the members exposed by TMemIniFile.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 15465)	Erases all data from the INI file in memory.
	Create (see page 15464)	Creates a TMemIniFile (see page 15462) object for an application.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	DeleteKey (see page 15465)	Deletes a specified entry from the .ini file.
V	Destroy (see page 15464)	Frees the memory associated with the TMemIniFile (see page 15462) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EraseSection (see page 15466)	Erases an entire section of the INI file.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetStrings (see page 15466)	Populates a string list with INI file data.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	ReadBinaryStream (see page 15438)	Retrieves a binary value from the ini file.
V	ReadBool (see page 15439)	Retrieves a Boolean value from an ini file.
V	ReadDate (see page 15439)	Retrieves a date value from an ini file.
V	ReadDateTime (see page 15440)	Retrieves a date-time value from an ini file.
V	ReadFloat (see page 15440)	Retrieves a float value from an ini file.
V	ReadInteger (see page 15441)	Retrieves an integer value from an ini file.
V	ReadSection (see page 15467)	Reads all the key names from a specified section of an INI file into a string list.
V	ReadSections (see page 15467)	Reads the names of all sections in an INI file into a string list.
A	ReadSections (see page 15442)	This is the overview for the ReadSections method overload.
V	ReadSectionValues (see page 15468)	Reads the values from all keys within a section of an INI file into a string list.
V	ReadString (see page 15468)	Retrieves a string value from the INI file.
V	ReadTime (see page 15444)	Retrieves a time value from an ini file.
	Rename (see page 15469)	Associates the TMemIniFile (see page 15462) with a different INI file.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SectionExists (see page 15444)	Indicates whether a section exists in the ini file.
	SetStrings (see page 15469)	Populates the memory buffer with contents of a string list.
V	UpdateFile (see page 15470)	Flushes buffered INI file data to disk.
V	ValueExists (see page 15445)	Indicates whether a key exists in the ini file.
V	WriteBinaryStream (see page 15446)	This is WriteBinaryStream, a member of class TCustomIniFile.
V	WriteBool (see page 15446)	Writes a Boolean value to an ini file.
V	WriteDate (see page 15446)	Writes a date value to an ini file.
V	WriteDateTime (see page 15447)	Writes a date-time value to an ini file.
V	WriteFloat (see page 15447)	Writes a float value to an ini file.
V	WriteInteger (see page 15448)	Writes an integer value to an ini file.
V	WriteString (see page 15470)	Writes a string value to an INI file.
V	WriteTime (see page 15449)	Writes a time value to an ini file.

Public Properties

	Name	Description
	CaseSensitive (see page 15471)	Controls (see page 5711) whether section and key names are handled in a case-sensitive manner.
R	FileName (see page 15450)	Contains the name of the ini file from which to read and to which to write information.

82.1.5.2 TMemIniFile.Create Constructor

Creates a TMemIniFile (see page 15462) object for an application.

Pascal

```
constructor Create(const FileName: string);
```

C++

```
__fastcall TMemIniFile(const AnsiString FileName);
```

Description

Call Create to instantiate a TMemIniFile (see page 15462) object for an application. Create sets the FileName (see page 15450) property, and loads a copy of the INI file into memory, in an internal string list.

FileName (see page 15450) is the name of the INI file the TMemIniFile (see page 15462) uses.

Note: On Windows, INI files are meant to be stored in the \WINDOWS directory or the application's directory. To work with an INI file in another location, specify the full path name of the file in FileName (see page 15450).

See Also

FileName (see page 15450)

Create (see page 22287)

82.1.5.3 TMemIniFile.Destroy Destructor

Frees the memory associated with the TMemIniFile (see page 15462) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMemIniFile();
```

Description

Do not call Destroy directly; instead call Free.

Warning: The destructor does not write the current memory image to disk. Be sure to save changes made to TMemIniFile (see page 15462) by calling UpdateFile (see page 15470) before destroying a TMemIniFile (see page 15462) instance.

See Also

Free (see page 22300)

UpdateFile (see page 15470)

Create (see page 15464)

82.1.5.4 TMemIniFile Methods

The methods of the TMemIniFile class are listed here.

Public Methods

	Name	Description
	Clear (see page 15465)	Erases all data from the INI file in memory.
V	DeleteKey (see page 15465)	Deletes a specified entry from the .ini file.
V	EraseSection (see page 15466)	Erases an entire section of the INI file.
	GetStrings (see page 15466)	Populates a string list with INI file data.
V	ReadSection (see page 15467)	Reads all the key names from a specified section of an INI file into a string list.
V	ReadSections (see page 15467)	Reads the names of all sections in an INI file into a string list.
V	ReadSectionValues (see page 15468)	Reads the values from all keys within a section of an INI file into a string list.
V	ReadString (see page 15468)	Retrieves a string value from the INI file.
	Rename (see page 15469)	Associates the TMemIniFile (see page 15462) with a different INI file.
	SetStrings (see page 15469)	Populates the memory buffer with contents of a string list.
V	UpdateFile (see page 15470)	Flushes buffered INI file data to disk.
V	WriteString (see page 15470)	Writes a string value to an INI file.

82.1.5.4.1 TMemIniFile.Clear Method

Erases all data from the INI file in memory.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to erase all data from the INI file that is currently buffered in memory. All sections, keys, and values are erased.

Warning: No exception is generated when using Clear and the data has not been saved to the INI file with the UpdateFile (see page 15470) method.

See Also

UpdateFile (see page 15470)

SetStrings (see page 15469)

82.1.5.4.2 TMemIniFile.DeleteKey Method

Deletes a specified entry from the .ini file.

Pascal

```
procedure DeleteKey(const Section: String; const Ident: String); override;
```

C++

```
virtual __fastcall DeleteKey(const AnsiString Section, const AnsiString Ident);
```

Description

Call DeleteKey to erase .ini file entry. Section is string containing the name of a .ini file section, and Ident is a string containing the name of the key to remove.

Note: Attempting to erase a key in a nonexistent section or attempting to erase a nonexistent key are not errors. In these cases, DeleteKey does nothing.

DeleteKey only affects the in-memory copy of the .ini file, not the copy on disk. The change is not written to disk until you call UpdateFile (see page 15470).

See Also

SectionExists (see page 15444)

ValueExists (see page 15445)

EraseSection (see page 15466)

UpdateFile (see page 15445)

CaseSensitive (see page 15471)

82.1.5.4.3 TMemIniFile.EraseSection Method

Erases an entire section of the INI file.

Pascal

```
procedure EraseSection(const Section: string); override;
```

C++

```
virtual __fastcall EraseSection(const AnsiString Section);
```

Description

Call EraseSection to remove a section, all its keys, and their data values from an INI file. Section identifies the INI file section to remove. If a section cannot be removed, an exception is raised.

EraseSection only affects the in-memory copy of the INI file, not the copy on disk. The change is not written to disk until you call UpdateFile (see page 15470).

See Also

SectionExists (see page 15444)

DeleteKey (see page 15465)

UpdateFile (see page 15470)

CaseSensitive (see page 15471)

82.1.5.4.4 TMemIniFile.GetStrings Method

Populates a string list with INI file data.

Pascal

```
procedure GetStrings(List: TStrings);
```

C++

```
__fastcall GetStrings(TStrings List);
```

Description

Call GetStrings to copy the INI file data currently stored in memory into a pre-existing string list.

List is a string list object, created and maintained by the application, into which the INI file data is copied.

See Also

SetStrings (see page 15469)

82.1.5.4.5 TMemIniFile.ReadSection Method

Reads all the key names from a specified section of an INI file into a string list.

Pascal

```
procedure ReadSection(const Section: string; Strings: TStrings); override;
```

C++

```
virtual __fastcall ReadSection(const AnsiString Section, TStrings Strings);
```

Description

Call ReadSection to retrieve the names of all keys within a specified section of an INI file into a string list.

Section identifies the section from which to retrieve a list of key names. Strings specifies the string to hold the retrieved names. Strings can point to a TStrings descendant such as a string list, or to a component property, such as Items for a TListBox component.

Data read by ReadSection comes from the in-memory copy of the INI file, not the copy on disk.

See Also

SectionExists (see page 15444)

ReadSections (see page 15467)

ReadSectionValues (see page 15468)

CaseSensitive (see page 15471)

82.1.5.4.6 TMemIniFile.ReadSections Method

Reads the names of all sections in an INI file into a string list.

Pascal

```
procedure ReadSections(Strings: TStrings); override;
```

C++

```
virtual __fastcall ReadSections(TStrings Strings);
```

Description

Call ReadSections to retrieve the names of all sections in an INI file into a string list.

Strings specifies the string object to hold the retrieved names. Strings can point to a TStrings descendant, or to a component property, such as Items for a TListBox component.

Data read by ReadSections comes from the in-memory copy of the INI file, not the copy on disk.

See Also

ReadSection (see page 15467)

ReadSectionValues (see page 15468)

82.1.5.4.7 TMemIniFile.ReadSectionValues Method

Reads the values from all keys within a section of an INI file into a string list.

Pascal

```
procedure ReadSectionValues(const Section: string; Strings: TStrings); override;
```

C++

```
virtual __fastcall ReadSectionValues(const AnsiString Section, TStrings Strings);
```

Description

Call ReadSectionValues to read the values from all keys within a specified section of an INI file into a string list object. The values are added to the string list as strings of the form KeyName=ValueName.

Section identifies the section in the file from which to read key values.

Strings is the string list object into which to read the values. Strings can point to a TStrings descendant, or to a component property, such as Items for a TListBox component.

Data read by ReadSectionValues comes from the in-memory copy of the INI file, not the copy on disk.

See Also

SectionExists (see page 15444)

ReadSection (see page 15467)

ReadSections (see page 15467)

CaseSensitive (see page 15471)

82.1.5.4.8 TMemIniFile.ReadString Method

Retrieves a string value from the INI file.

Pascal

```
function ReadString(const Section: string; const Ident: string; const Default: string):
string; override;
```

C++

```
virtual __fastcall AnsiString ReadString(const AnsiString Section, const AnsiString Ident,
const AnsiString Default);
```

Description

Call ReadString to read a string value from an INI file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the value. Default is the string value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

The ReadString method inherited from TCustomIniFile (see page 15434) is modified to operate on the in-memory copy of the INI file data. The read methods for all other data types use ReadString internally and then translate to the appropriate data type.

See Also

ReadBool (see page 15439)

ReadDate (see page 15439)

ReadDateTime (see page 15440)

ReadFloat (see page 15440)

ReadInteger (see page 15441)

ReadTime (see page 15444)

CaseSensitive (see page 15471)

82.1.5.4.9 TMemIniFile.Rename Method

Associates the TMemIniFile (see page 15462) with a different INI file.

Pascal

```
procedure Rename(const FileName: string; Reload: Boolean);
```

C++

```
__fastcall Rename(const AnsiString FileName, Boolean Reload);
```

Description

Call Rename to change the INI file used by the TMemIniFile (see page 15462) object to a file other than the one specified when the object was created.

FileName (see page 15450) is name of the new INI file to use.

Reload determines whether data is immediately loaded into memory from the INI file specified in FileName (see page 15450). When Reload is true, the TMemIniFile (see page 15462) object is repopulated with the data from the specified file. When Reload is false, the in-memory image is unchanged. This is useful, for example, when copying the data from one INI file to another file.

See Also

FileName (see page 15450)

Create (see page 15464)

82.1.5.4.10 TMemIniFile.SetStrings Method

Populates the memory buffer with contents of a string list.

Pascal

```
procedure SetStrings(List: TStrings);
```

C++

```
__fastcall SetStrings(TStrings List);
```

Description

Call SetStrings to populate the in-memory copy of the INI file data with the contents of a string list.

List is a string list object, created and maintained by the application, from which the INI file data is copied.

SetStrings uses the Clear (see page 15465) method to erase any existing INI file data prior to copying the new data.

SetStrings can be used to reread data from the INI file from disk after it is erased with the Clear (see page 15465) method. The LoadFromFile method of a string list can be used to retrieve the contents of the INI file and then SetStrings can copy the data into the memory buffer.

See Also

LoadFromFile (see page 3061)

Clear (see page 15465)

GetStrings (see page 15466)

82.1.5.4.11 TMemIniFile.UpdateFile Method

Flushes buffered INI file data to disk.

Pascal

```
procedure UpdateFile; override;
```

C++

```
virtual __fastcall UpdateFile();
```

Description

Call UpdateFile to copy INI file data stored in memory to the copy of the INI file on disk UpdateFile overwrites all data in the disk copy of the INI file with the INI file data stored in memory.

If the Rename (see page 15469) method is used to change the INI file the TMemIniFile (see page 15462) uses, UpdateFile copies the INI file data in memory to a file of the new name. If the file does not already exist, it is created. If the new file already exists, it is overwritten.

See Also

FileName (see page 15450)

82.1.5.4.12 TMemIniFile.WriteString Method

Writes a string value to an INI file.

Pascal

```
procedure WriteString(const Section: String; const Ident: String; const Value: String);
override;
```

C++

```
virtual __fastcall WriteString(const AnsiString Section, const AnsiString Ident, const
AnsiString Value);
```

Description

Call WriteString to write a string value to an INI file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the string value to write.

The WriteString method inherited from TCustomIniFile (see page 15434) is modified to operate on the in-memory copy of the INI file data. The write methods for all other data types use WriteString internally after data values are translated from the original data type.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteString creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 15446)

WriteDate (see page 15446)

WriteDateTime (see page 15447)

WriteFloat (see page 15447)

WriteInteger (see page 15448)

WriteTime (see page 15449)

CaseSensitive (see page 15471)

82.1.5.5 TMemIniFile Properties

The properties of the TMemIniFile class are listed here.

Public Properties

	Name	Description
	CaseSensitive (see page 15471)	Controls (see page 5711) whether section and key names are handled in a case-sensitive manner.

82.1.5.5.1 TMemIniFile.CaseSensitive Property

Controls (see page 5711) whether section and key names are handled in a case-sensitive manner.

Pascal

```
property CaseSensitive: Boolean;
```

C++

```
__property Boolean CaseSensitive;
```

Description

Use CaseSensitive to indicate whether the names of sections and keys should be treated in a case-sensitive or case-insensitive manner. Set CaseSensitive to true to allow multiple sections or keys that differ only based on case. Set CaseSensitive to false to treat names that differ only by case as the same.

See Also

ReadSection (see page 15467)

ReadString (see page 15468)

SectionExists (see page 15444)

ValueExists (see page 15445)

82.1.6 TStringHash Class

Class Hierarchy

IniFiles.TStringHash

Pascal

```
TStringHash = class;
```

C++

```
class TStringHash;
```

File

IniFiles

Description

This is class IniFiles.TStringHash.

82.1.6.1 TStringHash Members

The following tables list the members exposed by TStringHash.

Protected Methods

	Name	Description
	Find (see page 15473)	This is Find, a member of class TStringHash.
	HashOf (see page 15474)	This is HashOf, a member of class TStringHash.

Public Methods

	Name	Description
	Add (see page 15473)	This is Add, a member of class TStringHash.
	Clear (see page 15473)	This is Clear, a member of class TStringHash.
	Create (see page 15472)	This is Create, a member of class TStringHash.
	Destroy (see page 15472)	This is Destroy, a member of class TStringHash.
	Modify (see page 15474)	This is Modify, a member of class TStringHash.
	Remove (see page 15474)	This is Remove, a member of class TStringHash.
	ValueOf (see page 15474)	This is ValueOf, a member of class TStringHash.

82.1.6.2 TStringHash.Create Constructor

Pascal

```
constructor Create(Size: Cardinal = 256);
```

C++

```
__fastcall TStringHash(unsigned Size = 256);
```

Description

This is Create, a member of class TStringHash.

82.1.6.3 TStringHash.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TStringHash();
```

Description

This is Destroy, a member of class TStringHash.

82.1.6.4 TStringHash Methods

The methods of the TStringHash class are listed here.

Protected Methods

	Name	Description
	Find (see page 15473)	This is Find, a member of class TStringHash.
	HashOf (see page 15474)	This is HashOf, a member of class TStringHash.

Public Methods

	Name	Description
	Add (see page 15473)	This is Add, a member of class TStringHash.
	Clear (see page 15473)	This is Clear, a member of class TStringHash.
	Modify (see page 15474)	This is Modify, a member of class TStringHash.
	Remove (see page 15474)	This is Remove, a member of class TStringHash.
	ValueOf (see page 15474)	This is ValueOf, a member of class TStringHash.

82.1.6.4.1 TStringHash.Add Method

Pascal

```
procedure Add(const Key: string; Value: Integer);
```

C++

```
__fastcall Add(const AnsiString Key, int Value);
```

Description

This is Add, a member of class TStringHash.

82.1.6.4.2 TStringHash.Clear Method

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

This is Clear, a member of class TStringHash.

82.1.6.4.3 TStringHash.Find Method

Pascal

```
function Find(const Key: string): PPHashItem;
```

C++

```
__fastcall PPHashItem Find(const AnsiString Key);
```

Description

This is Find, a member of class TStringHash.

82.1.6.4.4 TStringHash.HashOf Method

Pascal

```
function HashOf(const Key: string): Cardinal; virtual;
```

C++

```
virtual __fastcall unsigned HashOf(const AnsiString Key);
```

Description

This is HashOf, a member of class TStringHash.

82.1.6.4.5 TStringHash.Modify Method

Pascal

```
function Modify(const Key: string; Value: Integer): Boolean;
```

C++

```
__fastcall Boolean Modify(const AnsiString Key, int Value);
```

Description

This is Modify, a member of class TStringHash.

82.1.6.4.6 TStringHash.Remove Method

Pascal

```
procedure Remove(const Key: string);
```

C++

```
__fastcall Remove(const AnsiString Key);
```

Description

This is Remove, a member of class TStringHash.

82.1.6.4.7 TStringHash.ValueOf Method

Pascal

```
function ValueOf(const Key: string): Integer;
```

C++

```
__fastcall int ValueOf(const AnsiString Key);
```

Description

This is ValueOf, a member of class TStringHash.

82.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	THashItem (see page 15475)	This is record IniFiles.THashItem.

82.2.1 IniFiles.THashItem Record

Pascal

```
THashItem = record
  Next: PHashItem;
  Key: string;
  Value: Integer;
end;
```

C++

```
struct THashItem {
  PHashItem Next;
  AnsiString Key;
  int Value;
};
```

File

IniFiles

Description

This is record IniFiles.THashItem.

82.3 Types

The following table lists types in this documentation.

Types

Name	Description
PHashItem (see page 15476)	This is type IniFiles.PHashItem.
PPHashItem (see page 15476)	This is type IniFiles.PPHashItem.

82.3.1 IniFiles.PHashItem Type

Pascal

```
PHashItem = ^THashItem;
```

C++

```
^THashItem PHashItem;
```

File

IniFiles

Description

This is type IniFiles.PHashItem.

82.3.2 IniFiles.PPHashItem Type

Pascal

```
PPHashItem = ^PHashItem;
```

C++

```
^PHashItem PPHashItem;
```

File

IniFiles

Description

This is type IniFiles.PPHashItem.

83 IntfInfo Namespace

This is namespace IntfInfo.

Topics

Name	Description
Classes (see page 15479)	The following table lists classes in this documentation.
Functions (see page 15480)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 15485)	The following table lists structs, records, enums in this documentation.
Types (see page 15488)	The following table lists types in this documentation.
Constants (see page 15491)	The following table lists constants in this documentation.

Classes

	Name	Description
	EInterfaceRTTIException (see page 15479)	EInterfaceRTTIException is the exception raised when an interface does not have RTTI.

Constants

Name	Description
CallingConventionName (see page 15491)	This is constant IntfInfo.CallingConventionName.
TypeInfoNames (see page 15491)	This is constant IntfInfo.TypeInfoNames.

Functions

	Name	Description
	GetClsMemberTypeInfo (see page 15480)	This is function IntfInfo.GetClsMemberTypeInfo.
	GetDynArrayElTypeInfo (see page 15480)	This is function IntfInfo.GetDynArrayElTypeInfo.
	GetDynArrayNextInfo (see page 15481)	This is function IntfInfo.GetDynArrayNextInfo.
	GetDynArrayNextInfo2 (see page 15481)	This is function IntfInfo.GetDynArrayNextInfo2.
	GetIntfMetaData (see page 15481)	This is function IntfInfo.GetIntfMetaData.
	GetIntfMetaData (see page 15482)	This is function IntfInfo.GetIntfMetaData.
	GetMethNum (see page 15482)	This is function IntfInfo.GetMethNum.
	GetPropListFlat (see page 15482)	This is function IntfInfo.GetPropListFlat.
	IsStoredPropConst (see page 15483)	This is function IntfInfo.IsStoredPropConst.
	OtherTypeName (see page 15483)	This is function IntfInfo.OtherTypeName.
	SameTypeInfo (see page 15483)	This is function IntfInfo.SameTypeInfo.
	TypeNamesMatch (see page 15484)	This is function IntfInfo.TypeNamesMatch.

Structs, Records, Enums

	Name	Description
	TCallConv (see page 15485)	This is record IntfInfo.TCallConv.
	TFillMethodArrayOpt (see page 15485)	This is record IntfInfo.TFillMethodArrayOpt.
	TIntfMetaData (see page 15486)	This is record IntfInfo.TIntfMetaData.
	TIntfMethEntry (see page 15486)	This is record IntfInfo.TIntfMethEntry.
	TIntfParamEntry (see page 15487)	This is record IntfInfo.TIntfParamEntry.

Types

Name	Description
PIntfMetaData (see page 15488)	This is type IntfInfo.PIntfMetaData.
PIntfMethEntry (see page 15488)	This is type IntfInfo.PIntfMethEntry.
PIntfParamEntry (see page 15489)	This is type IntfInfo.PIntfParamEntry.
TIntfMethEntryArray (see page 15489)	This is type IntfInfo.TIntfMethEntryArray.
TIntfParamEntryArray (see page 15489)	This is type IntfInfo.TIntfParamEntryArray.
TPIntfMethEntryArray (see page 15490)	This is type IntfInfo.TPIntfMethEntryArray.

83.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EInterfaceRTTIException (see page 15479)	EInterfaceRTTIException is the exception raised when an interface does not have RTTI.

83.1.1 EInterfaceRTTIException Class

EInterfaceRTTIException is the exception raised when an interface does not have RTTI.

Class Hierarchy

System::TObject → SysUtils.Exception → IntfInfo.EInterfaceRTTIException

Pascal

```
EInterfaceRTTIException = class(Exception);
```

C++

```
class EInterfaceRTTIException : public Exception;
```

File

IntfInfo

Description

EInterfaceRTTIException is thrown when an application expects an interface to be invokable but discovers that it does not include any runtime type information (RTTI). For example, the invocation registry raises an EInterfaceRTTIException exception when an application tries to register an interface that is not compiled with RTTI. Invoker components raise an EInterfaceRTTIException exception when an incoming request includes a call to an interface that does not include RTTI.

See Also

TInvokableClassRegistry (see page 15526)

TSoapPascalInvoker (see page 20020)

IInvokable (see page 22398)

83.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetClsMemberTypeInfo (see page 15480)	This is function IntfInfo.GetClsMemberTypeInfo.
	GetDynArrayElTypeInfo (see page 15480)	This is function IntfInfo.GetDynArrayElTypeInfo.
	GetDynArrayNextInfo (see page 15481)	This is function IntfInfo.GetDynArrayNextInfo.
	GetDynArrayNextInfo2 (see page 15481)	This is function IntfInfo.GetDynArrayNextInfo2.
	GetIntfMetaData (see page 15481)	This is function IntfInfo.GetIntfMetaData.
	GetIntfMetaData (see page 15482)	This is function IntfInfo.GetIntfMetaData.
	GetMethNum (see page 15482)	This is function IntfInfo.GetMethNum.
	GetPropListFlat (see page 15482)	This is function IntfInfo.GetPropListFlat.
	IsStoredPropConst (see page 15483)	This is function IntfInfo.IsStoredPropConst.
	OtherTypeName (see page 15483)	This is function IntfInfo.OtherTypeName.
	SameTypeInfo (see page 15483)	This is function IntfInfo.SameTypeInfo.
	TypeNamesMatch (see page 15484)	This is function IntfInfo.TypeNamesMatch.

83.2.1 IntfInfo.GetClsMemberTypeInfo Function

Pascal

```
function GetClsMemberTypeInfo(const ObjectTypeInfo: PTypeInfo; const MemberName: string = ''):
PTypeInfo;
```

C++

```
PTypeInfo GetClsMemberTypeInfo(const PTypeInfo ObjectTypeInfo, const AnsiString MemberName =
'');
```

File

IntfInfo

Description

This is function IntfInfo.GetClsMemberTypeInfo.

83.2.2 IntfInfo.GetDynArrayElTypeInfo Function

Pascal

```
procedure GetDynArrayElTypeInfo(typeInfo: PTypeInfo; var EltInfo: PTypeInfo; var Dims:
Integer);
```

C++

```
GetDynArrayElTypeInfo(PTypeInfo typeInfo, PTypeInfo EltInfo, int Dims);
```

File

IntfInfo

Description

This is function IntfInfo.GetDynArrayElTypeInfo.

83.2.3 IntfInfo.GetDynArrayNextInfo Function

Pascal

```
function GetDynArrayNextInfo(typeInfo: PTypeInfo): PTypeInfo;
```

C++

```
PTypeInfo GetDynArrayNextInfo(PTypeInfo typeInfo);
```

File

IntfInfo

Description

This is function IntfInfo.GetDynArrayNextInfo.

83.2.4 IntfInfo.GetDynArrayNextInfo2 Function

Pascal

```
function GetDynArrayNextInfo2(typeInfo: PTypeInfo; var Name: string): PTypeInfo;
```

C++

```
PTypeInfo GetDynArrayNextInfo2(PTypeInfo typeInfo, AnsiString Name);
```

File

IntfInfo

Description

This is function IntfInfo.GetDynArrayNextInfo2.

83.2.5 IntfInfo.GetIntfMetaData Function

Pascal

```
procedure GetIntfMetaData(Info: PTypeInfo; var IntfMD: TIntfMetaData; IncludeAllAncMethods:
Boolean = False); overload;
```

C++

```
GetIntfMetaData(PTypeInfo Info, TIntfMetaData IntfMD, Boolean IncludeAllAncMethods = False);
```

File

IntfInfo

Description

This is function IntfInfo.GetIntfMetaData.

83.2.6 IntfInfo.GetIntfMetaData Function

Pascal

```
procedure GetIntfMetaData(Info: PTypeInfo; var IntfMD: TIntfMetaData; MethodArrayOpt:
TFillMethodArrayOpt); overload;
```

C++

```
GetIntfMetaData(PTypeInfo Info, TIntfMetaData IntfMD, TFillMethodArrayOpt MethodArrayOpt);
```

File

IntfInfo

Description

This is function IntfInfo.GetIntfMetaData.

83.2.7 IntfInfo.GetMethNum Function

Pascal

```
function GetMethNum(const IntfMD: TIntfMetaData; const MethName: string; ParamCount: Integer =
-1): Integer;
```

C++

```
int GetMethNum(const TIntfMetaData IntfMD, const AnsiString MethName, int ParamCount = -1);
```

File

IntfInfo

Description

This is function IntfInfo.GetMethNum.

83.2.8 IntfInfo.GetPropListFlat Function

Pascal

```
function GetPropListFlat(TypeInfo: PTypeInfo; out PropList: PPropList): Integer;
```

C++

```
int GetPropListFlat(PTypeInfo TypeInfo, PPropList PropList);
```

File

IntfInfo

Description

This is function IntfInfo.GetPropListFlat.

83.2.9 IntfInfo.IsStoredPropConst Function

Pascal

```
function IsStoredPropConst(Instance: TObject; PropInfo: PPropInfo): Boolean;
```

C++

```
Boolean IsStoredPropConst(TObject * Instance, PPropInfo PropInfo);
```

File

IntfInfo

Description

This is function IntfInfo.IsStoredPropConst.

83.2.10 IntfInfo.OtherTypeName Function

Pascal

```
function OtherTypeName(const TypeName: string): string;
```

C++

```
AnsiString OtherTypeName(const AnsiString TypeName);
```

File

IntfInfo

Description

This is function IntfInfo.OtherTypeName.

83.2.11 IntfInfo.SameTypeInfo Function

Pascal

```
function SameTypeInfo(const RegInfo: PTypeInfo; const OtherInfo: PTypeInfo): boolean;
```

C++

```
boolean SameTypeInfo(const PTypeInfo RegInfo, const PTypeInfo OtherInfo);
```

File

IntfInfo

Description

This is function IntfInfo.SameTypeInfo.

83.2.12 IntfInfo.TypeNamesMatch Function

Pascal

```
function TypeNamesMatch(const RegName: string; const OtherName: string): boolean;
```

C++

```
boolean TypeNamesMatch(const AnsiString RegName, const AnsiString OtherName);
```

File

IntfInfo

Description

This is function IntfInfo.TypeNamesMatch.

83.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCallConv (see page 15485)	This is record IntfInfo.TCallConv.
	TFillMethodArrayOpt (see page 15485)	This is record IntfInfo.TFillMethodArrayOpt.

Records

	Name	Description
	TIntfMetaData (see page 15486)	This is record IntfInfo.TIntfMetaData.
	TIntfMethEntry (see page 15486)	This is record IntfInfo.TIntfMethEntry.
	TIntfParamEntry (see page 15487)	This is record IntfInfo.TIntfParamEntry.

83.3.1 IntfInfo.TCallConv Enumeration

Pascal

```
TCallConv = (
  ccReg,
  ccCdecl,
  ccPascal,
  ccStdCall,
  ccSafeCall
);
```

C++

```
enum TCallConv {
  ccReg,
  ccCdecl,
  ccPascal,
  ccStdCall,
  ccSafeCall
};
```

File

IntfInfo

Description

This is record IntfInfo.TCallConv.

83.3.2 IntfInfo.TFillMethodArrayOpt Enumeration

Pascal

```
TFillMethodArrayOpt = (
  fmoAllBaseMethods,
  fmoRTTIBaseMethods,
```

```
    fmoNoBaseMethods
  );
```

C++

```
  enum TFillMethodArrayOpt {
    fmoAllBaseMethods,
    fmoRTTIBaseMethods,
    fmoNoBaseMethods
  };
```

File

IntfInfo

Description

This is record IntfInfo.TFillMethodArrayOpt.

83.3.3 IntfInfo.TIntfMetaData Record

Pascal

```
  TIntfMetaData = record
    Name: string;
    UnitName: string;
    MDA: TIntfMethEntryArray;
    IID: TGUID;
    Info: PTypeInfo;
    AncInfo: PTypeInfo;
    NumAnc: Integer;
  end;
```

C++

```
  struct TIntfMetaData {
    AnsiString Name;
    AnsiString UnitName;
    TIntfMethEntryArray MDA;
    TGUID IID;
    PTypeInfo Info;
    PTypeInfo AncInfo;
    int NumAnc;
  };
```

File

IntfInfo

Description

This is record IntfInfo.TIntfMetaData.

83.3.4 IntfInfo.TIntfMethEntry Record

Pascal

```
  TIntfMethEntry = record
    Name: string;
    CC: TCallConv;
    Pos: Integer;
    ParamCount: Integer;
```

```
    ResultInfo: PTypeInfo;
    SelfInfo: PTypeInfo;
    Params: TIntfParamEntryArray;
    HasRTTI: Boolean;
  end;
```

C++

```
  struct TIntfMethEntry {
    AnsiString Name;
    TCallConv CC;
    int Pos;
    int ParamCount;
    PTypeInfo ResultInfo;
    PTypeInfo SelfInfo;
    TIntfParamEntryArray Params;
    Boolean HasRTTI;
  };
```

File

IntfInfo

Description

This is record IntfInfo.TIntfMethEntry.

83.3.5 IntfInfo.TIntfParamEntry Record

Pascal

```
  TIntfParamEntry = record
    Flags: TParamFlags;
    Name: string;
    Info: PTypeInfo;
  end;
```

C++

```
  struct TIntfParamEntry {
    TParamFlags Flags;
    AnsiString Name;
    PTypeInfo Info;
  };
```

File

IntfInfo

Description

This is record IntfInfo.TIntfParamEntry.

83.4 Types

The following table lists types in this documentation.

Types

Name	Description
PIntfMetaData (see page 15488)	This is type IntfInfo.PIntfMetaData.
PIntfMethEntry (see page 15488)	This is type IntfInfo.PIntfMethEntry.
PIntfParamEntry (see page 15489)	This is type IntfInfo.PIntfParamEntry.
TIntfMethEntryArray (see page 15489)	This is type IntfInfo.TIntfMethEntryArray.
TIntfParamEntryArray (see page 15489)	This is type IntfInfo.TIntfParamEntryArray.
TPIntfMethEntryArray (see page 15490)	This is type IntfInfo.TPIntfMethEntryArray.

83.4.1 IntfInfo.PIntfMetaData Type

Pascal

```
PIntfMetaData = ^TIntfMetaData;
```

C++

```
^TIntfMetaData PIntfMetaData;
```

File

IntfInfo

Description

This is type IntfInfo.PIntfMetaData.

83.4.2 IntfInfo.PIntfMethEntry Type

Pascal

```
PIntfMethEntry = ^TIntfMethEntry;
```

C++

```
^TIntfMethEntry PIntfMethEntry;
```

File

IntfInfo

Description

This is type IntfInfo.PIntfMethEntry.

83.4.3 IntfInfo.PIntfParamEntry Type

Pascal

```
PIntfParamEntry = ^TIntfParamEntry;
```

C++

```
^TIntfParamEntry PIntfParamEntry;
```

File

IntfInfo

Description

This is type IntfInfo.PIntfParamEntry.

83.4.4 IntfInfo.TIntfMethEntryArray Type

Pascal

```
TIntfMethEntryArray = array of TIntfMethEntry;
```

C++

```
array of TIntfMethEntry TIntfMethEntryArray;
```

File

IntfInfo

Description

This is type IntfInfo.TIntfMethEntryArray.

83.4.5 IntfInfo.TIntfParamEntryArray Type

Pascal

```
TIntfParamEntryArray = array of TIntfParamEntry;
```

C++

```
array of TIntfParamEntry TIntfParamEntryArray;
```

File

IntfInfo

Description

This is type IntfInfo.TIntfParamEntryArray.

83.4.6 IntfInfo.TPIntfMethEntryArray Type

Pascal

```
TPIntfMethEntryArray = array of PIntfMethEntry;
```

C++

```
array of PIntfMethEntry TPIntfMethEntryArray;
```

File

IntfInfo

Description

This is type IntfInfo.TPIntfMethEntryArray.

83.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CallingConventionName (see page 15491)	This is constant IntfInfo.CallingConventionName.
TypeInfoNames (see page 15491)	This is constant IntfInfo.TypeInfoNames.

83.5.1 IntfInfo.CallingConventionName Constant

Pascal

```
CallingConventionName: array[ccReg..ccSafeCall] of string = ('REGISTER', 'CDECL', 'PASCAL',
'STDCALL', 'SAFECALL');
```

C++

```
array[ccReg..ccSafeCall] of string CallingConventionName = ('REGISTER', 'CDECL', 'PASCAL',
'STDCALL', 'SAFECALL');
```

File

IntfInfo

Description

This is constant IntfInfo.CallingConventionName.

83.5.2 IntfInfo.TypeInfoNames Constant

Pascal

```
TypeInfoNames: array[0..33] of string = ( 'Boolean', 'bool', 'Char', 'char', 'Char', 'signed
char', 'Byte', 'unsigned char', 'SmallInt', 'short', 'Word', 'unsigned short', 'Integer',
'int', 'Cardinal', 'unsigned int', 'Integer', 'long', 'Cardinal', 'unsigned long', 'Int64',
'__int64', 'Int64', 'unsigned __int64', 'Single', 'float', 'Double', 'double', 'Extended',
'long double', 'String', 'AnsiString', 'WideString', 'WideString');
```

C++

```
array[0..33] of string TypeInfoNames = ( 'Boolean', 'bool', 'Char', 'char', 'Char', 'signed
char', 'Byte', 'unsigned char', 'SmallInt', 'short', 'Word', 'unsigned short', 'Integer',
'int', 'Cardinal', 'unsigned int', 'Integer', 'long', 'Cardinal', 'unsigned long', 'Int64',
'__int64', 'Int64', 'unsigned __int64', 'Single', 'float', 'Double', 'double', 'Extended',
'long double', 'String', 'AnsiString', 'WideString', 'WideString');
```

File

IntfInfo

Description

This is constant IntfInfo.TypeInfoNames.

84 InvokeRegistry Namespace

This is namespace InvokeRegistry.

Topics

Name	Description
Classes (see page 15496)	The following table lists classes in this documentation.
Interfaces (see page 15609)	The following table lists interfaces in this documentation.
Functions (see page 15619)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 15624)	The following table lists structs, records, enums in this documentation.
Types (see page 15636)	The following table lists types in this documentation.
Variables (see page 15644)	The following table lists variables in this documentation.
Constants (see page 15645)	The following table lists constants in this documentation.

Classes

	Name	Description
	ERemotableException (see page 15496)	ERemotableException is the class for exceptions that can be communicated from a Web Service application to a client application.
	ETypeRegistryException (see page 15500)	ETypeRegistryException is the exception raised when the invocation registry or the invokable type registry encounters an invalid type.
	TDataContext (see page 15501)	TDataContext stores temporary information about a set of objects, Variants (see page 23567), and other related values.
	THeaderList (see page 15509)	THeaderList maintains a list of SOAP headers.
	TInvContext (see page 15516)	TInvContext stores information about a method call on an invokeable interface.
	TInvokableClass (see page 15520)	TInvokableClass is the base class for the implementation classes of invokable interfaces.
	TInvokableClassRegistry (see page 15526)	TInvokableClassRegistry is the type of the invocation registry.
	TRemotable (see page 15556)	TRemotable is the base class for classes that can be passed as parameters or return values in a Web Service application.
	TRemotableTypeRegistry (see page 15562)	TRemotableTypeRegistry is the type of the remotable type registry.
	TRemotableXS (see page 15580)	TRemotableXS is the base class for classes that represent a non-native scalar type on an invokable interface in a Web Service application.
	TSOAPAttachment (see page 15583)	TSOAPAttachment represents an attachment that is included with SOAP messages that represent method calls in a Web Service application.
	TSOAPHeader (see page 15596)	TSOAPHeader represents a header on the SOAP envelope that contains an encoded message in a Web Service application.
	TSOAPHeaders (see page 15600)	TSOAPHeaders is a helper object for implementing the ISOAPHeaders (see page 15613) interface.
	TSOAPHeadersBase (see page 15605)	TSOAPHeadersBase is the base class for classes that maintain lists of SOAP headers.

Constants

Name	Description
AS_ATTRIBUTE (see page 15645)	Identifies a property of a remotable class as one that corresponds to an attribute in the SOAP encoding.
XMLBase64Types (see page 15645)	This is constant InvokeRegistry.XMLBase64Types.
XMLSchemaInstNamepspaces (see page 15646)	This is constant InvokeRegistry.XMLSchemaInstNamepspaces.
XMLSchemaNamepspaces (see page 15646)	This is constant InvokeRegistry.XMLSchemaNamepspaces.

Functions

	Name	Description
	GetRemotableDataContext (see page 15619)	Returns the TDataContext (see page 15501) object that provides temporary storage during a call to an invokable interface.
	InvRegistry (see page 15620)	Provides access to the invocation registry.
	RemClassRegistry (see page 15620)	Provides access to the remotable class registry.
	RemTypeRegistry (see page 15621)	Provides access to the remotable type registry.
	SetRemotableDataContext (see page 15622)	Sets the data context that provides temporary storage during a call to an invokable interface.
	SubstituteStrings (see page 15623)	This is function InvokeRegistry.SubstituteStrings.

Interfaces

	Name	Description
	IHeadersSetter (see page 15609)	This is class InvokeRegistry.IHeadersSetter.
	IObjConverter (see page 15610)	IObjConverter is the interface for converting between a remotable class and its SOAP representation.
	ISOAPHeaders (see page 15613)	ISOAPHeaders is the interface for accessing the headers that accompany a message encoded using SOAP.

Structs, Records, Enums

	Name	Description
	eHeaderMethodType (see page 15624)	eHeaderMethodType and THeaderMethodTypeArray (see page 15638) indicate the types of messages that should include a given header type.
	ExtNameMapItem (see page 15625)	This is record InvokeRegistry.ExtNameMapItem.
	IntfExceptionItem (see page 15625)	IntfExceptionItem and TExceptionItemArray (see page 15638) describe the association between a remotable exception and an interface that can raise it.
	IntfHeaderItem (see page 15626)	IntfHeaderItem and THeaderItemArray (see page 15638) describe the association between a header and the SOAP messages that include it.
	InvRegClassEntry (see page 15627)	This is record InvokeRegistry.InvRegClassEntry.
	InvRegIntfEntry (see page 15627)	InvRegIntfEntry stores information about an invokable interface.
	MethParamNameMapItem (see page 15629)	This is record InvokeRegistry.MethParamNameMapItem.
	ObjectConvertOptions (see page 15629)	ObjectConvertOptions and TObjectConvertOptions (see page 15640) are flags that customize how to convert a remotable object to or from its SOAP representation.
	SerializationOptions (see page 15630)	SerializationOptions and TSerializationOptions (see page 15642) are flags that customize how to convert a remotable object to or from its SOAP representation.
	TDynToClear (see page 15631)	This is record InvokeRegistry.TDynToClear.
	TIntfInvokeOption (see page 15632)	TIntfInvokeOption and TIntfInvokeOptions (see page 15639) represent flags that customize how to handle calls to an invokable interface.
	TObjMultiOptions (see page 15633)	TObjMultiOptions indicates how multiple references to the same remotable class should be translated into a SOAP representation.
	TRemHeaderEntry (see page 15634)	This is record InvokeRegistry.TRemHeaderEntry.
	TRemRegEntry (see page 15634)	TRemRegEntry stores information for the remotable type registry.

Types

Name	Description
ERemotableExceptionClass (see page 15636)	This is type InvokeRegistry.ERemotableExceptionClass.
InvString (see page 15637)	This is type InvokeRegistry.InvString.
PTRemotable (see page 15637)	This is type InvokeRegistry.PTRemotable.

TCreateInstanceProc (see page 15637)	TCreateInstanceProc is the type of the factory routine that returns an instance of an invokable class.
TExceptionItemArray (see page 15638)	This is type InvokeRegistry.TExceptionItemArray.
THeaderItemArray (see page 15638)	This is type InvokeRegistry.THeaderItemArray.
THeaderMethodTypeArray (see page 15638)	This is type InvokeRegistry.THeaderMethodTypeArray.
TIntfInvokeOptions (see page 15639)	This is type InvokeRegistry.TIntfInvokeOptions.
TInvokableClassClass (see page 15639)	This is type InvokeRegistry.TInvokableClassClass.
TInvRegIntfEntryArray (see page 15639)	This is type InvokeRegistry.TInvRegIntfEntryArray.
TObjectConvertOptions (see page 15640)	This is type InvokeRegistry.TObjectConvertOptions.
TPascalRemotableClassRegistry (see page 15640)	TPascalRemotableClassRegistry is the type of the remotable class registry.
TPascalRemotableTypeRegistry (see page 15640)	TPascalRemotableTypeRegistry is the type of the remotable type registry.
TRemotableClass (see page 15641)	This is type InvokeRegistry.TRemotableClass.
TRemotableClassRegistry (see page 15641)	This is type InvokeRegistry.TRemotableClassRegistry.
TRemotableXSClass (see page 15641)	This is type InvokeRegistry.TRemotableXSClass.
TRemRegEntryArray (see page 15642)	This is type InvokeRegistry.TRemRegEntryArray.
TRequiredArray (see page 15642)	TRequiredArray is a dynamic array of boolean values.
TSerializationOptions (see page 15642)	This is type InvokeRegistry.TSerializationOptions.
TSOAPAttachmentClass (see page 15643)	This is type InvokeRegistry.TSOAPAttachmentClass.
TSOAPHeaderClass (see page 15643)	This is type InvokeRegistry.TSOAPHeaderClass.

Variables

Name	Description
AppNameSpacePrefix (see page 15644)	Supplies a unique prefix to use for automatically generated namespaces given to invokable interfaces and remotable types.

84.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ERemotableException (see page 15496)	ERemotableException is the class for exceptions that can be communicated from a Web Service application to a client application.
	ETypeRegistryException (see page 15500)	ETypeRegistryException is the exception raised when the invocation registry or the invokable type registry encounters an invalid type.
	TDataContext (see page 15501)	TDataContext stores temporary information about a set of objects, Variants (see page 23567), and other related values.
	THeaderList (see page 15509)	THeaderList maintains a list of SOAP headers.
	TInvContext (see page 15516)	TInvContext stores information about a method call on an invokeable interface.
	TInvokableClass (see page 15520)	TInvokableClass is the base class for the implementation classes of invokable interfaces.
	TInvokableClassRegistry (see page 15526)	TInvokableClassRegistry is the type of the invocation registry.
	TRemotable (see page 15556)	TRemotable is the base class for classes that can be passed as parameters or return values in a Web Service application.
	TRemotableTypeRegistry (see page 15562)	TRemotableTypeRegistry is the type of the remotable type registry.
	TRemotableXS (see page 15580)	TRemotableXS is the base class for classes that represent a non-native scalar type on an invokable interface in a Web Service application.
	TSOAPAttachment (see page 15583)	TSOAPAttachment represents an attachment that is included with SOAP messages that represent method calls in a Web Service application.
	TSOAPHeader (see page 15596)	TSOAPHeader represents a header on the SOAP envelope that contains an encoded message in a Web Service application.
	TSOAPHeaders (see page 15600)	TSOAPHeaders is a helper object for implementing the ISOAPHeaders (see page 15613) interface.
	TSOAPHeadersBase (see page 15605)	TSOAPHeadersBase is the base class for classes that maintain lists of SOAP headers.

84.1.1 ERemotableException Class

ERemotableException is the class for exceptions that can be communicated from a Web Service application to a client application.

Class Hierarchy

Pascal

```
ERemotableException = class(Exception);
```

C++

```
class ERemotableException : public Exception;
```

File

InvokeRegistry

Description

ERemotableException is the exception class for exceptions that a client raises based on a SOAP fault packet that it receives from a Web Service application. The ERemotableException properties represent standard elements of the SOAP fault packet.

Create (see page 15498) an ERemotableException descendant to define a custom exception class for problems encountered when executing the methods of an invokable interface in a Web Service application. When a Web Service application raises an ERemotableException descendant in the course of executing a request from a client application, it adds the values of any published properties to the fault packet, in addition to the value of the Message property. Note that these published properties must be remotable types: that is, the value must be a scalar type, or it must be a registered descendant of TRemotable (see page 15556).

Before the properties of an ERemotableException descendant can be transmitted in a fault packet, it must be registered with the remotable type registry (RemTypeRegistry (see page 15621)). Call the RegisterXSClass method of the remotable type registry, on both the client and the server application, to enable your exception class to be transmitted.

See Also

RemTypeRegistry (see page 15621)

TRemotableTypeRegistry (see page 15562)

Creating Custom Exception Classes for Web Services

84.1.1.1 ERemotableException Members

The following tables list the members exposed by ERemotableException.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 15498)	This is the overview for the Create constructor overload.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	FaultActor (see page 15499)	Indicates the agent that raised the fault.
	FaultCode (see page 15499)	Indicates where the fault was encountered.
	FaultDetail (see page 15500)	Provides application-specific details about the exception that occurred.
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.

84.1.1.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	ERemotableException.Create () (see page 15498)	Creates and initializes an instance of ERemotableException (see page 15496).
	ERemotableException.Create (string, WideString, WideString) (see page 15498)	Creates and initializes an instance of ERemotableException (see page 15496).

84.1.1.2.1 ERemotableException.Create Constructor ()

Creates and initializes an instance of ERemotableException (see page 15496).

Pascal

```
constructor Create; virtual; overload;
constructor Create(const Msg: string; const AFaultCode: WideString = ''; const AFaultActor:
WideString = ''); virtual; overload;
```

C++

```
virtual __fastcall ERemotableException();
virtual __fastcall ERemotableException(const AnsiString Msg, const BSTR AFaultCode = '', const
BSTR AFaultActor = '');
```

Description

Create generates an ERemotableException (see page 15496) object.

Msg is the error message associated with the exception. It becomes the value of the Message property.

AFaultCode indicates where the fault that gave rise to the exception was encountered. It becomes the value of the FaultCode (see page 15499) property.

AFaultActor indicates the agent that gave rise to the fault. It becomes the value of the FaultActor (see page 15499) property.

See Also

Message (see page 22760)

FaultCode (see page 15499)

FaultActor (see page 15499)

84.1.1.3 ERemotableException Properties

The properties of the ERemotableException class are listed here.

Public Properties

	Name	Description
	FaultActor (see page 15499)	Indicates the agent that raised the fault.
	FaultCode (see page 15499)	Indicates where the fault was encountered.
	FaultDetail (see page 15500)	Provides application-specific details about the exception that occurred.

84.1.1.3.1 ERemotableException.FaultActor Property

Indicates the agent that raised the fault.

Pascal

```
property FaultActor: WideString;
```

C++

```
__property BSTR FaultActor;
```

Description

FaultActor is the URI of the agent that caused the fault. This is used when the SOAP request passes through multiple agents when processed. The FaultActor can be a proxy server, the Web Service application to which the request was addressed, or another agent that the target Web Service application calls on when processing the request.

This property is only set if the Web Service provider specifies a FaultActor in its SOAP fault packet.

See Also

FaultCode (see page 15499)

84.1.1.3.2 ERemotableException.FaultCode Property

Indicates where the fault was encountered.

Pascal

```
property FaultCode: WideString;
```

C++

```
__property BSTR FaultCode;
```

Description

FaultCode is one of the following strings:

Value	Meaning
Server	The Web Service application raised the exception before it was able to begin processing the content of the SOAP request packet. For example, if the server can't process a request because it is out of memory, it specifies a FaultCode of 'Server'.
Client	The Web Service application raised the exception because of an error in the SOAP request packet that the client provided. For example, if the SOAP request packet specified a nonexistent method or invalid parameter, the server specifies a FaultCode of 'Client'.
VersionMismatch	The Web Service application is a different version than the one that the SOAP request packet specifies.
MustUnderstand	The Web Service application did not understand one of the SOAP headers in the request packet and the request packet set its MustUnderstand header to indicate that the server must handle all of its SOAP headers.

See Also

FaultActor (see page 15499)

84.1.1.3.3 ERemotableException.FaultDetail Property

Provides application-specific details about the exception that occurred.

Pascal

```
property FaultDetail: WideString;
```

C++

```
__property BSTR FaultDetail;
```

Description

FaultDetail is a fragment of XML that the server generates to provide detailed information about the exception that occurred. FaultDetail provides context information in addition to the fault message that becomes the value of the Message property.

Each server defines the structure and meaning of FaultDetail in an application-specific way. When an ERemotableException (see page 15496) descendant includes published properties, their values are added to FaultDetail.

See Also

Message (see page 22760)

84.1.2 ETypeRegistryException Class

ETypeRegistryException is the exception raised when the invocation registry or the invokable type registry encounters an invalid type.

Class Hierarchy

Pascal

```
ETypeRegistryException = class(Exception);
```

C++

```
class ETypeRegistryException : public Exception;
```

File

InvokeRegistry

Description

ETypeRegistryException is thrown when

An application tries to register an implementation class with the invocation registry and that class does not implement any interfaces.

An application tries to look up a type in the remotable type registry that is not remotable.

See Also

TInvokableClassRegistry (see page 15526)

TRemotableTypeRegistry (see page 15562)

84.1.3 TDataContext Class

TDataContext stores temporary information about a set of objects, Variants (see page 23567), and other related values.

Class Hierarchy

Pascal

```
TDataContext = class;
```

C++

```
class TDataContext;
```

File

InvokeRegistry

Description

TDataContext is a repository for information that belongs together. It introduces a number of methods for storing different types of information, including objects, Variants (see page 23567), pointers, and raw data (untyped bytes). When the TDataContext object is freed, it performs all necessary cleanup of the information it stores, freeing the objects, clearing the Variants (see page 23567), and so on.

Do not use TDataContext directly. It is intended as a base class for classes that need a temporary repository that serves a specific purpose. TDataContext does not allocate the internal dynamic arrays that store pointers and raw data: This task is left to descendant classes, such as TInvContext (see page 15516).

See Also

TInvContext (see page 15516)

84.1.3.1 TDataContext Members

The following tables list the members exposed by TDataContext.

Protected Fields

	Name	Description
	Data (see page 15503)	This is Data, a member of class TDataContext.
	DataOffset (see page 15503)	This is DataOffset, a member of class TDataContext.
	DataP (see page 15503)	This is DataP, a member of class TDataContext.
	DynArrayToClear (see page 15504)	This is DynArrayToClear, a member of class TDataContext.
	FObjsToDestroy (see page 15504)	This is FObjsToDestroy, a member of class TDataContext.
	StrToClear (see page 15504)	This is StrToClear, a member of class TDataContext.
	VarToClear (see page 15504)	This is VarToClear, a member of class TDataContext.
	WStrToClear (see page 15504)	This is WStrToClear, a member of class TDataContext.

Public Methods

	Name	Description
	AddDynArrayToClear (see page 15505)	Stores a dynamic array that can be cleared when the data context is finished.
	AddObjectToDestroy (see page 15506)	Stores an object that can be freed when the data context is finished.
	AddStrToClear (see page 15506)	Stores a string that can be cleared when the data context is finished.
	AddVariantToClear (see page 15507)	Stores the value of a Variant that can be cleared when the data context is finished.
	AddWStrToClear (see page 15507)	Stores a wide string that can be cleared when the data context is finished.
	AllocData (see page 15508)	Returns a pointer to the data context's cache of raw memory.
	Create (see page 15502)	Creates an instance of TDataContext (see page 15501).
V	Destroy (see page 15502)	Destroys an instance of TDataContext (see page 15501).
	GetDataPointer (see page 15508)	Returns a pointer from the internal cache of pointers.
	RemoveObjectToDestroy (see page 15508)	Remove an object from the list of objects to free when the data context is freed.
	SetDataPointer (see page 15509)	Stores a pointer in the internal cache of pointers.

84.1.3.2 TDataContext.Create Constructor

Creates an instance of TDataContext (see page 15501).

Pascal

```
constructor Create;
```

C++

```
__fastcall TDataContext();
```

Description

Call Create to instantiate a TDataContext (see page 15501) object.

84.1.3.3 TDataContext.Destroy Destructor

Destroys an instance of TDataContext (see page 15501).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataContext();
```

Description

Destroy frees all the temporary data stored in the TDataContext (see page 15501) instance.

See Also

delete

84.1.3.4 TDataContext Fields

The fields of the TDataContext class are listed here.

Protected Fields

	Name	Description
	Data (see page 15503)	This is Data, a member of class TDataContext.
	DataOffset (see page 15503)	This is DataOffset, a member of class TDataContext.
	DataP (see page 15503)	This is DataP, a member of class TDataContext.
	DynArrayToClear (see page 15504)	This is DynArrayToClear, a member of class TDataContext.
	FObjsToDestroy (see page 15504)	This is FObjsToDestroy, a member of class TDataContext.
	StrToClear (see page 15504)	This is StrToClear, a member of class TDataContext.
	VarToClear (see page 15504)	This is VarToClear, a member of class TDataContext.
	WStrToClear (see page 15504)	This is WStrToClear, a member of class TDataContext.

84.1.3.4.1 TDataContext.Data Field

Pascal

```
Data: array of Byte;
```

C++

```
array of Byte Data;
```

Description

This is Data, a member of class TDataContext.

84.1.3.4.2 TDataContext.DataOffset Field

Pascal

```
DataOffset: Integer;
```

C++

```
int DataOffset;
```

Description

This is DataOffset, a member of class TDataContext.

84.1.3.4.3 TDataContext.DataP Field

Pascal

```
DataP: array of Pointer;
```

C++

```
array of Pointer DataP;
```

Description

This is DataP, a member of class TDataContext.

84.1.3.4.4 TDataContext.DynArrayToClear Field

Pascal

```
DynArrayToClear: array of TDynToClear;
```

C++

```
array of TDynToClear DynArrayToClear;
```

Description

This is DynArrayToClear, a member of class TDataContext.

84.1.3.4.5 TDataContext.FObjsToDestroy Field

Pascal

```
FObjsToDestroy: array of TObject;
```

C++

```
array of TObject FObjsToDestroy;
```

Description

This is FObjsToDestroy, a member of class TDataContext.

84.1.3.4.6 TDataContext.StrToClear Field

Pascal

```
StrToClear: array of Pointer;
```

C++

```
array of Pointer StrToClear;
```

Description

This is StrToClear, a member of class TDataContext.

84.1.3.4.7 TDataContext.VarToClear Field

Pascal

```
VarToClear: array of Pointer;
```

C++

```
array of Pointer VarToClear;
```

Description

This is VarToClear, a member of class TDataContext.

84.1.3.4.8 TDataContext.WStrToClear Field

Pascal

```
WStrToClear: array of Pointer;
```

C++

```
array of Pointer WStrToClear;
```

Description

This is WStrToClear, a member of class TDataContext.

84.1.3.5 TDataContext Methods

The methods of the TDataContext class are listed here.

Public Methods

	Name	Description
	AddDynArrayToClear (see page 15505)	Stores a dynamic array that can be cleared when the data context is finished.
	AddObjectToDestroy (see page 15506)	Stores an object that can be freed when the data context is finished.
	AddStrToClear (see page 15506)	Stores a string that can be cleared when the data context is finished.
	AddVariantToClear (see page 15507)	Stores the value of a Variant that can be cleared when the data context is finished.
	AddWStrToClear (see page 15507)	Stores a wide string that can be cleared when the data context is finished.
	AllocData (see page 15508)	Returns a pointer to the data context's cache of raw memory.
	GetDataPointer (see page 15508)	Returns a pointer from the internal cache of pointers.
	RemoveObjectToDestroy (see page 15508)	Remove an object from the list of objects to free when the data context is freed.
	SetDataPointer (see page 15509)	Stores a pointer in the internal cache of pointers.

84.1.3.5.1 TDataContext.AddDynArrayToClear Method

Stores a dynamic array that can be cleared when the data context is finished.

Pascal

```
procedure AddDynArrayToClear(P: Pointer; Info: PTypeInfo);
```

C++

```
__fastcall AddDynArrayToClear(void * P, PTypeInfo Info);
```

Description

Call AddDynArrayToClear to add a dynamic array to the list of temporary dynamic arrays that TDataContext (see page 15501) maintains. When the TDataContext (see page 15501) instance is freed, it clears all dynamic arrays in this list of temporary arrays.

P is a pointer to the dynamic array.

Info is the TypeInfo pointer for the dynamic array.

Note: TDataContext (see page 15501) does not provide access to this list. The caller must keep a reference to the dynamic arrays for use while the TDataContext (see page 15501) instance exists.

See Also

Destroy (see page 15502)

AddObjectToDestroy (see page 15506)

AddVariantToClear (see page 15507)

AddStrToClear (see page 15506)

AddWStrToClear (see page 15507)

84.1.3.5.2 TDataContext.AddObjectToDestroy Method

Stores an object that can be freed when the data context is finished.

Pascal

```
procedure AddObjectToDestroy(Obj: TObject);
```

C++

```
__fastcall AddObjectToDestroy(TObject * Obj);
```

Description

Call AddObjectToDestroy to add an object instance to the list of temporary objects that TDataContext (see page 15501) maintains. When the TDataContext (see page 15501) instance is freed, it frees all the objects in this list of temporary objects.

Note: TDataContext (see page 15501) does not provide access to this list. The caller must keep a reference to the object for use while the TDataContext (see page 15501) instance exists.

See Also

Destroy (see page 15502)

AddDynArrayToClear (see page 15505)

AddVariantToClear (see page 15507)

RemoveObjectToDestroy (see page 15508)

AddStrToClear (see page 15506)

84.1.3.5.3 TDataContext.AddStrToClear Method

Stores a string that can be cleared when the data context is finished.

Pascal

```
procedure AddStrToClear(P: Pointer);
```

C++

```
__fastcall AddStrToClear(void * P);
```

Description

Call AddStrToClear to add a string to the list of temporary strings that TDataContext (see page 15501) maintains. When the TDataContext (see page 15501) instance is freed, it clears all strings in this list.

P is a pointer to the string.

Note: TDataContext (see page 15501) does not provide access to this list. The caller must keep a reference to the string for use while the TDataContext (see page 15501) instance exists.

See Also

Destroy (see page 15502)

AddObjectToDestroy (see page 15506)

AddVariantToClear (see page 15507)

AddDynArrayToClear (see page 15505)

AddWStrToClear (see page 15507)

84.1.3.5.4 TDataContext.AddVariantToClear Method

Stores the value of a Variant that can be cleared when the data context is finished.

Pascal

```
procedure AddVariantToClear(P: PVarData);
```

C++

```
__fastcall AddVariantToClear(PVarData P);
```

Description

Call AddVariantToClear to add a Variant to the list of temporary Variant values that TDataContext (see page 15501) maintains. When the TDataContext (see page 15501) instance is freed, it clears all the Variants (see page 23567) in this list of temporary Variant values.

Note: TDataContext (see page 15501) does not provide access to this list. The caller must keep a reference to the Variant for use while the TDataContext (see page 15501) instance exists.

See Also

Destroy (see page 15502)

AddStrToClear (see page 15506)

AddWStrToClear (see page 15507)

84.1.3.5.5 TDataContext.AddWStrToClear Method

Stores a wide string that can be cleared when the data context is finished.

Pascal

```
procedure AddWStrToClear(P: Pointer);
```

C++

```
__fastcall AddWStrToClear(void * P);
```

Description

Call AddWStrToClear to add a wide string to the list of temporary wide strings that TDataContext (see page 15501) maintains. When the TDataContext (see page 15501) instance is freed, it clears all strings in this list.

P is a pointer to the WideString.

Note: TDataContext (see page 15501) does not provide access to this list. The caller must keep a reference to the string for use while the TDataContext (see page 15501) instance exists.

See Also

Destroy (see page 15502)

AddObjectToDestroy (see page 15506)

AddVariantToClear (see page 15507)

AddDynArrayToClear (see page 15505)

AddStrToClear (see page 15506)

84.1.3.5.6 TDataContext.AllocData Method

Returns a pointer to the data context's cache of raw memory.

Pascal

```
function AllocData(Size: Integer): Pointer;
```

C++

```
__fastcall void * AllocData(int Size);
```

Description

Use AllocData to obtain a pointer to a chunk of temporary memory.

Size is the number of bytes desired.

AllocData returns a pointer to Size bytes.

Note: TDataContext (see page 15501) does not actually allocate the memory that AllocData returns. Descendant classes must initialize the internal cache that the returned pointer addresses.

See Also

GetDataPointer (see page 15508)

84.1.3.5.7 TDataContext.GetDataPointer Method

Returns a pointer from the internal cache of pointers.

Pascal

```
function GetDataPointer(Index: Integer): Pointer;
```

C++

```
__fastcall void * GetDataPointer(int Index);
```

Description

GetDataPointer returns a pointer from the internal cache of pointers. This value is assigned by a call to the SetDataPointer (see page 15509) method.

Index is the index of the desired pointer, where 0 is the first pointer, 1 is the second pointer, and so on.

Note: TDataContext (see page 15501) does not actually allocate the memory to store these pointers. Descendant classes must initialize the internal dynamic array that TDataContext (see page 15501) uses to store pointers.

See Also

SetDataPointer (see page 15509)

84.1.3.5.8 TDataContext.RemoveObjectToDestroy Method

Remove an object from the list of objects to free when the data context is freed.

Pascal

```
procedure RemoveObjectToDestroy(Obj: TObject);
```

C++

```
__fastcall RemoveObjectToDestroy(TObject * Obj);
```

Description

Call RemoveObjectToDestroy to remove an object instance from the list of temporary objects that TDataContext (see page 15501) maintains. This prevents TDataContext (see page 15501) from freeing the object when it is itself freed.

For example, TRemotable (see page 15556) descendants that are created by a Web Service application when it executes an invokable interface call are automatically added to a data context that persists for the length of the method call. When an application changes the remotable object's DataContext property, it automatically calls RemoveObjectToDestroy so that the object is not automatically freed when the method call is complete.

See Also

DataContext (see page 15561)

AddObjectToDestroy (see page 15506)

84.1.3.5.9 TDataContext.SetDataPointer Method

Stores a pointer in the internal cache of pointers.

Pascal

```
procedure SetDataPointer(Index: Integer; P: Pointer);
```

C++

```
__fastcall SetDataPointer(int Index, void * P);
```

Description

SetDataPointer stores a pointer that can later be retrieved by a call to GetDataPointer (see page 15508). TDataContext (see page 15501) owns only the pointer, not the data to which it points. When you free the TDataContext (see page 15501) instance, it frees its list of pointers, but not the memory to which they point, unless they are first obtained by a call to the AllocData (see page 15508) method.

Index is the index of the pointer in the internal cache of pointers, where 0 specifies the first pointer, 1 specifies the second pointer, and so on.

P is the pointer to store.

Note: TDataContext (see page 15501) does not actually allocate the memory to store these pointers. Descendant classes must initialize the internal dynamic array that TDataContext (see page 15501) uses to store pointers.

See Also

GetDataPointer (see page 15508)

AllocData (see page 15508)

84.1.4 THeaderList Class

THeaderList maintains a list of SOAP headers.

Class Hierarchy

Pascal

```
THeaderList = class;
```

C++

```
class THeaderList;
```

File

InvokeRegistry

Description

Use THeaderList to store and maintain a list of SOAP headers (TSOAPHeader (see page 15596) descendants).

See Also

TSoapHeader

84.1.4.1 THeaderList Members

The following tables list the members exposed by THeaderList.

Protected Methods

	Name	Description
	GetCount (see page 15513)	This is GetCount, a member of class THeaderList.
	GetHeader (see page 15513)	This is GetHeader, a member of class THeaderList.
	GetOwnsObjects (see page 15513)	This is GetOwnsObjects, a member of class THeaderList.
	SetOwnsObjects (see page 15514)	This is SetOwnsObjects, a member of class THeaderList.

Public Methods

	Name	Description
	Add (see page 15511)	This is the overview for the Add method overload.
	Clear (see page 15512)	Empties the list and frees the TSOAPHeader (see page 15596) objects.
	Create (see page 15510)	Creates and initializes a THeaderList (see page 15509) instance.
V	Destroy (see page 15511)	Destroys an instance of THeaderList (see page 15509).
	Extract (see page 15512)	Removes a specified header from the list without freeing it.
	IndexOf (see page 15513)	This is IndexOf, a member of class THeaderList.

Public Properties

	Name	Description
R	Count (see page 15514)	Indicates the number of headers in the list.
R	Headers (see page 15515)	Lists the headers.
	OwnsObjects (see page 15515)	Allows THeaderList (see page 15509) to free objects when they the list is cleared or destroyed.

84.1.4.2 THeaderList.Create Constructor

Creates and initializes a THeaderList (see page 15509) instance.

Pascal

```
constructor Create;
```

C++

```
__fastcall THeaderList();
```

Description

Create creates a THeaderList (see page 15509) instance.

84.1.4.3 THeaderList.Destroy Destructor

Destroys an instance of THeaderList (see page 15509).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THeaderList();
```

Description

Do not call Destroy directly. Instead, call Free. Free verifies that the THeaderList (see page 15509) reference is not nil, and only then calls Destroy.

Destroy frees the THeaderList (see page 15509) instance. If the OwnsObjects (see page 15515) property is true, it also frees the header objects in the list.

See Also

Free (see page 22300)

OwnsObjects (see page 15515)

84.1.4.4 THeaderList Methods

The methods of the THeaderList class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 15513)	This is GetCount, a member of class THeaderList.
	GetHeader (see page 15513)	This is GetHeader, a member of class THeaderList.
	GetOwnsObjects (see page 15513)	This is GetOwnsObjects, a member of class THeaderList.
	SetOwnsObjects (see page 15514)	This is SetOwnsObjects, a member of class THeaderList.

Public Methods

	Name	Description
	Add (see page 15511)	This is the overview for the Add method overload.
	Clear (see page 15512)	Empties the list and frees the TSOAPHeader (see page 15596) objects.
	Extract (see page 15512)	Removes a specified header from the list without freeing it.
	IndexOf (see page 15513)	This is IndexOf, a member of class THeaderList.

84.1.4.4.1 Add Method

This is the overview for the Add method overload.

Overload List

	Name	Description
	THeaderList.Add (TObject) (see page 15512)	Adds a new header to the end of the list.
	THeaderList.Add (TSOAPHeader) (see page 15512)	Adds a new header to the end of the list.

84.1.4.4.1.1 THeaderList.Add Method (TObject)

Adds a new header to the end of the list.

Pascal

```
procedure Add(Header: TSOAPHeader); overload;
procedure Add(Header: TObject); overload;
```

C++

```
__fastcall Add(TSOAPHeader Header);
__fastcall Add(TObject * Header);
```

Description

Call Add to add a header to the end of the Headers (see page 15515) property array. When you call Add, the new Header is added to the end of the array and the value of the Count (see page 15514) property is incremented.

Header is the TSOAPHeader (see page 15596) descendants that is added to the list.

See Also

Headers (see page 15515)

Count (see page 15514)

Clear (see page 15512)

84.1.4.4.2 THeaderList.Clear Method

Empties the list and frees the TSOAPHeader (see page 15596) objects.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Clear removes all of the headers from the list and, if the OwnsObjects (see page 15515) property is true, frees each one . After a call to Clear, the value of the Count (see page 15514) property is 0.

See Also

Headers (see page 15515)

Count (see page 15514)

OwnsObjects (see page 15515)

84.1.4.4.3 THeaderList.Extract Method

Removes a specified header from the list without freeing it.

Pascal

```
function Extract(Obj: TObject): TObject;
```

C++

```
__fastcall TObject * Extract(TObject * Obj);
```

Description

Call Extract to remove a header from the list without freeing the header object. After a header is removed, all the headers that follow it are moved up in index position and Count (see page 15514) is decremented.

See Also

Add (see page 15512)

Count (see page 15514)

Headers (see page 15515)

OwnsObjects (see page 15515)

84.1.4.4.4 THeaderList.GetCount Method

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

This is GetCount, a member of class THeaderList.

84.1.4.4.5 THeaderList.GetHeader Method

Pascal

```
function GetHeader(Index: Integer): TObject;
```

C++

```
__fastcall TObject * GetHeader(int Index);
```

Description

This is GetHeader, a member of class THeaderList.

84.1.4.4.6 THeaderList.GetOwnsObjects Method

Pascal

```
function GetOwnsObjects: Boolean;
```

C++

```
__fastcall Boolean GetOwnsObjects();
```

Description

This is GetOwnsObjects, a member of class THeaderList.

84.1.4.4.7 THeaderList.IndexOf Method

Pascal

```
function IndexOf(Obj: TObject): Integer;
```

C++

```
__fastcall int IndexOf(TObject * Obj);
```

Description

This is IndexOf, a member of class THeaderList.

84.1.4.4.8 THeaderList.SetOwnsObjects Method

Pascal

```
procedure SetOwnsObjects(Val: Boolean);
```

C++

```
__fastcall SetOwnsObjects(Boolean Val);
```

Description

This is SetOwnsObjects, a member of class THeaderList.

84.1.4.5 THeaderList Properties

The properties of the THeaderList class are listed here.

Public Properties

	Name	Description
R	Count (see page 15514)	Indicates the number of headers in the list.
R	Headers (see page 15515)	Lists the headers.
	OwnsObjects (see page 15515)	Allows THeaderList (see page 15509) to free objects when they the list is cleared or destroyed.

84.1.4.5.1 THeaderList.Count Property

Indicates the number of headers in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Count indicates the number of headers that the Headers (see page 15515) property lists. To change the value of Count, you can add headers using the Add (see page 15512) method, remove them using the Extract (see page 15512) method, or remove all headers using the Clear (see page 15512) method.

See Also

Headers (see page 15515)

Add (see page 15512)

Clear (see page 15512)

Extract (see page 15512)

84.1.4.5.2 THeaderList.Headers Property

Lists the headers.

Pascal

```
property Headers [Index: Integer]: TObject;
```

C++

```
__property TObject * Headers[int Index];
```

Description

Use Headers to access headers in the list. Headers is a zero-based array of TSOAPHeader (see page 15596) descendants: The first header is indexed as 0, the second header is indexed as 1, and so forth. You can read the value at a specific index, or use Headers with the Count (see page 15514) property to iterate through the list.

Note: In Delphi, Headers is the default property of THeaderList (see page 15509). This means you can omit the property name.

Note: In C++, you can use the subscript operator to achieve a similar effect.

See Also

OwnsObjects (see page 15515)

Add (see page 15512)

Extract (see page 15512)

Count (see page 15514)

Clear (see page 15512)

operator_sb

84.1.4.5.3 THeaderList.OwnsObjects Property

Allows THeaderList (see page 15509) to free objects when they the list is cleared or destroyed.

Pascal

```
property OwnsObjects: Boolean;
```

C++

```
__property Boolean OwnsObjects;
```

Description

OwnsObjects allows THeaderList (see page 15509) to control the memory of its TSOAPHeader (see page 15596) objects. If OwnsObjects is true,

calling Clear (see page 15512) frees all the headers in the list in addition to emptying the list.

freeing the THeaderList (see page 15509) object frees all the objects in the list in addition to destroying the THeaderList (see page 15509) itself.

Note: Even if OwnsObjects is true, the Extract (see page 15512) method removes objects from the list without freeing them.

See Also

Headers (see page 15515)

Clear (see page 2874)

Destroy (see page 15511)

Extract (see page 15512)

84.1.5 TInvContext Class

TInvContext stores information about a method call on an invokeable interface.

Class Hierarchy

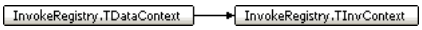

Pascal

```
TInvContext = class(TDataContext);
```

C++

```
class TInvContext : public TDataContext;
```

File

InvokeRegistry

Description

TInvContext is a helper object created and used by TRIO and TSoapPascalInvoker for generating calls to the methods on an invokeable interface. It stores temporary data needed when making the method call. TInvContext introduces methods for retrieving and setting information about the method being called, including its type signature, parameter values, and so on.

Component writers implementing their own classes to implement the IOPConvert interface must translate between the information stored in a TInvContext object and their class-specific encoding of a method call.

See Also

IOPConvert (see page 18122)

TRio

TSOAPPascalInvoker

84.1.5.1 TInvContext Members

The following tables list the members exposed by TInvContext.

Protected Fields

	Name	Description
	Data (see page 15503)	This is Data, a member of class TDataContext.
	DataOffset (see page 15503)	This is DataOffset, a member of class TDataContext.
	DataP (see page 15503)	This is DataP, a member of class TDataContext.
	DynArrayToClear (see page 15504)	This is DynArrayToClear, a member of class TDataContext.
	FObjsToDestroy (see page 15504)	This is FObjsToDestroy, a member of class TDataContext.
	StrToClear (see page 15504)	This is StrToClear, a member of class TDataContext.
	VarToClear (see page 15504)	This is VarToClear, a member of class TDataContext.
	WStrToClear (see page 15504)	This is WStrToClear, a member of class TDataContext.

Public Methods

	Name	Description
	AddDynArrayToClear (see page 15505)	Stores a dynamic array that can be cleared when the data context is finished.
	AddObjectToDestroy (see page 15506)	Stores an object that can be freed when the data context is finished.
	AddStrToClear (see page 15506)	Stores a string that can be cleared when the data context is finished.
	AddVariantToClear (see page 15507)	Stores the value of a Variant that can be cleared when the data context is finished.
	AddWStrToClear (see page 15507)	Stores a wide string that can be cleared when the data context is finished.
	AllocData (see page 15508)	Returns a pointer to the data context's cache of raw memory.
	AllocServerData (see page 15517)	Allocates memory for the parameters and result values of a specified method and sets up the parameter and result pointers.
	Create (see page 15502)	Creates an instance of TDataContext (see page 15501).
V	Destroy (see page 15502)	Destroys an instance of TDataContext (see page 15501).
	GetDataPointer (see page 15508)	Returns a pointer from the internal cache of pointers.
	GetParamPointer (see page 15518)	Returns a pointer to the value of a parameter on the called method.
	GetResultPointer (see page 15518)	Returns a pointer to the return value of the method call.
	RemoveObjectToDestroy (see page 15508)	Remove an object from the list of objects to free when the data context is freed.
	SetDataPointer (see page 15509)	Stores a pointer in the internal cache of pointers.
	SetMethodInfo (see page 15518)	Initializes the TInvContext (see page 15516) for a particular method on the invokeable interface.
	SetParamPointer (see page 15519)	Sets the value of a specified parameter.
	SetResultPointer (see page 15519)	This is SetResultPointer, a member of class TInvContext.

84.1.5.2 TInvContext Methods

The methods of the TInvContext class are listed here.

Public Methods

	Name	Description
	AllocServerData (see page 15517)	Allocates memory for the parameters and result values of a specified method and sets up the parameter and result pointers.
	GetParamPointer (see page 15518)	Returns a pointer to the value of a parameter on the called method.
	GetResultPointer (see page 15518)	Returns a pointer to the return value of the method call.
	SetMethodInfo (see page 15518)	Initializes the TInvContext (see page 15516) for a particular method on the invokeable interface.
	SetParamPointer (see page 15519)	Sets the value of a specified parameter.
	SetResultPointer (see page 15519)	This is SetResultPointer, a member of class TInvContext.

84.1.5.2.1 TInvContext.AllocServerData Method

Allocates memory for the parameters and result values of a specified method and sets up the parameter and result pointers.

Pascal

```
procedure AllocServerData(const MD: TIntfMethEntry);
```

C++

```
__fastcall AllocServerData(const TIntfMethEntry MD);
```

Description

AllocServerData suballocates the memory that the SetMethodInfo (see page 15518) method allocates so that it can represent the parameters and return value of a method. It looks up the types of each parameter and of the return type for the method, allocates memory for their values using the AllocData (see page 15508) method, and stores the pointers to that memory by calling the SetParamPointer (see page 15519) and SetResultPointer (see page 15519) methods.

MD is a description of the method, including its type, calling convention, and parameter signature.

Note: AllocServerData must come after a call to SetMethodInfo (see page 15518) so that the pool of memory is available. After calling AllocServerData, an application can call GetParamPointer (see page 15518) and GetResultPointer (see page 15518) to obtain pointers to the memory allocated for each parameter and for the return value of the method.

See Also

AllocData (see page 15508)

SetParamPointer (see page 15519)

setresultpointer (see page 15519)

SetMethodInfo (see page 15518)

84.1.5.2.2 TInvContext.GetParamPointer Method

Returns a pointer to the value of a parameter on the called method.

Pascal

```
function GetParamPointer(Param: Integer): Pointer;
```

C++

```
__fastcall void * GetParamPointer(int Param);
```

Description

GetParamPointer returns a pointer to the value of a specified parameter.

Param is the index of the desired parameter, where 0 is the first parameter, 1 is the second parameter, and so on.

This value is assigned by a call to the SetParamPointer (see page 15519) method.

See Also

SetParamPointer (see page 15519)

84.1.5.2.3 TInvContext.GetResultPointer Method

Returns a pointer to the return value of the method call.

Pascal

```
function GetResultPointer: Pointer;
```

C++

```
__fastcall void * GetResultPointer();
```

Description

GetResultPointer returns a pointer to the return value of the method call. This value is assigned by a call to the SetResultPointer (see page 15519) method.

See Also

setresultpointer (see page 15519)

84.1.5.2.4 TInvContext.SetMethodInfo Method

Initializes the TInvContext (see page 15516) for a particular method on the invokeable interface.

Pascal

```
procedure SetMethodInfo(const MD: TIntfMethEntry);
```

C++

```
__fastcall SetMethodInfo(const TIntfMethEntry MD);
```

Description

SetMethodInfo initializes the invocation context so that it represents a particular method. Objects that implement the IOPConvert interface call this method to initialize an invocation context when implementing their MsgToInvContext method.

MD is a description of the method, including its type, calling convention, and parameter signature.

Note: SetMethodInfo allocates the memory used internally to store pointers and raw data. An application must call this method before any calls to the AllocData (see page 15508), SetDataPointer (see page 15509), or SetParamPointer (see page 15519) methods.

See Also

IOPConvert (see page 18122)

MsgToInvContext (see page 18127)

AllocData (see page 15508)

SetDataPointer (see page 15509)

SetParamPointer (see page 15519)

84.1.5.2.5 TInvContext.SetParamPointer Method

Sets the value of a specified parameter.

Pascal

```
procedure SetParamPointer(Param: Integer; P: Pointer);
```

C++

```
__fastcall SetParamPointer(int Param, void * P);
```

Description

SetParamPointer associates a pointer with a specified parameter.

Param identifies the parameter whose value is to be set, where 0 specifies the first parameter, 1 specifies the second parameter, and so on.

P is a pointer to the value of the parameter. Memory to contain the value of P can be allocated using the AllocData (see page 15508) method.

See Also

GetParamPointer (see page 15518)

AllocData (see page 15508)

84.1.5.2.6 TInvContext.SetResultPointer Method

Pascal

```
procedure SetResultPointer(P: Pointer);
```

C++

```
__fastcall SetResultPointer(void * P);
```

Description

This is SetResultPointer, a member of class TInvContext.

84.1.6 TInvokableClass Class

TInvokableClass is the base class for the implementation classes of invokable interfaces.

Class Hierarchy

Pascal

```
TInvokableClass = class(TObject, IInterface);
```

C++

```
class TInvokableClass : public TObject, public IInterface;
```

File

InvokeRegistry

Description

Use TInvokableClass as a base class when writing the implementation class of an invokable interface. TInvokableClass has the following important attributes:

The invocation registry (InvRegistry (see page 15620)) knows how to create instances of TInvokableClass and (because it has a virtual constructor) its descendants. This allows the registry to supply an invoker in a Web Service application with an instance of the invokable class that can handle an incoming request.

TInvokableClass is an interfaced object that frees itself when the reference count on its interface drops to zero. Invoker components do not know when to free the implementation classes of the interfaces they call. Because TInvokableClass knows when to free itself, you do not need to supply your own lifetime management for this object.

TInvokableClass supports the ISOAPHeaders (see page 15613) interface so that your server can process the headers that accompany requests and add headers to outgoing responses.

To use a TInvokableClass descendant in a Web Service application, you must register it with the invocation registry. Use the RegisterInvokableClass method of the invocation registry to register your TInvokableClass descendant.

Note: If you do not use TInvokableClass as a base class when writing the implementation class for an invokable interface, you must supply the invocation registry with a method to retrieve instances of your class and you must handle the lifetime management of your class. If you are sending or receiving SOAP headers, you must use the global GetSOAPHeaders function to obtain an ISOAPHeaders (see page 15613) interface instead of obtaining one from the implementation class.

See Also

InvRegistry (see page 15620)

IInvokable (see page 22398)

TInvokableClassRegistry (see page 15526)

THTTPSoapPascalInvoker (see page 19951)

ISOAPHeaders (see page 15613)

Adding New Web Services

84.1.6.1 TInvokableClass Members

The following tables list the members exposed by TInvokableClass.

Protected Fields

	Name	Description
	FRefCount (see page 15522)	This is FRefCount, a member of class TInvokableClass.
	FSOAPHeaders (see page 15523)	This is FSOAPHeaders, a member of class TInvokableClass.

Protected Methods

	Name	Description
	_AddRef (see page 15523)	Increments the reference count for this interface.
	_Release (see page 15524)	Decrements the reference count for this interface.
	QueryInterface (see page 15525)	Implements the IInterface QueryInterface method.

Public Methods

	Name	Description
V	AfterConstruction (see page 15524)	Responds after the last constructor executes.
V	BeforeDestruction (see page 15524)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 15522)	Creates and initializes a TInvokableClass (see page 15520) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 15525)	Allocates memory for each instance of an invokable object.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	RefCount (see page 15526)	Indicates the number of interface pointers currently dependent upon the object.

84.1.6.2 TInvokableClass.Create Constructor

Creates and initializes a TInvokableClass (see page 15520) object.

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TInvokableClass();
```

Description

The TInvokableClass (see page 15520) constructor does nothing special. It exists primarily because

The TInvokableClass (see page 15520) constructor exists primarily because

The invocation registry knows how to call it.

It is virtual, so that the invocation registry can also instantiate descendant classes.

In addition, the constructor initializes the invokable class so that it can support the ISOAPHeaders (see page 15613) interface.

See Also

InvRegistry (see page 15620)

ISOAPHeaders (see page 15613)

84.1.6.3 TInvokableClass Fields

The fields of the TInvokableClass class are listed here.

Protected Fields

	Name	Description
	FRefCount (see page 15522)	This is FRefCount, a member of class TInvokableClass.
	FSOAPHeaders (see page 15523)	This is FSOAPHeaders, a member of class TInvokableClass.

84.1.6.3.1 TInvokableClass.FRefCount Field

Pascal

```
FRefCount: Integer;
```

C++

```
int FRefCount;
```

Description

This is FRefCount, a member of class TInvokableClass.

84.1.6.3.2 TInvokableClass.FSOAPHeaders Field

Pascal

```
FSOAPHeaders: TSOAPHeaders;
```

C++

```
TSOAPHeaders FSOAPHeaders;
```

Description

This is FSOAPHeaders, a member of class TInvokableClass.

84.1.6.4 TInvokableClass Methods

The methods of the TInvokableClass class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 15523)	Increments the reference count for this interface.
	_Release (see page 15524)	Decrements the reference count for this interface.
	QueryInterface (see page 15525)	Implements the IInterface QueryInterface method.

Public Methods

	Name	Description
	AfterConstruction (see page 15524)	Responds after the last constructor executes.
	BeforeDestruction (see page 15524)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	NewInstance (see page 15525)	Allocates memory for each instance of an invokable object.

84.1.6.4.1 TInvokableClass._AddRef Method

Increments the reference count for this interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

The Delphi compiler automatically generates a call to __AddRef for every reference to this interface. For most implementations, __AddRef increments the reference count for the interface and returns the new reference count. When the caller is through with the interface, the compiler generates a call to the _Release method, which decrements the reference count. When the reference count drops to zero, the object is automatically freed.

Note: Implementation objects can implement __AddRef and _Release so that they do not perform any reference counting. For such objects, the reference count never drops to zero and the object is not automatically freed. In these cases, it is the application's responsibility to free the object.

See Also

_Release (see page 22397)

84.1.6.4.2 TInvokableClass._Release Method

Decrements the reference count for this interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

The Delphi compiler automatically generates a call to __Release whenever a reference to this interface goes out of scope. For most implementations, __Release decrements the reference count for the interface and returns the new reference count. When the reference count drops to zero, the object is automatically freed.

Note: Implementation objects can implement __Release so that it does not perform any reference counting. For such objects, the reference count never drops to zero and the object is not automatically freed. In these cases, it is the application's responsibility to free the object.

See Also

_AddRef (see page 22397)

84.1.6.4.3 TInvokableClass.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of TInvokableClass (see page 15520) is allocated, RefCount (see page 15526) is incremented to prevent its destruction while processing the constructor. AfterConstruction decrements that RefCount (see page 15526) to re-establish the correct reference count after all constructors have executed.

See Also

BeforeDestruction (see page 15524)

NewInstance (see page 15525)

RefCount (see page 15526)

84.1.6.4.4 TInvokableClass.BeforeDestruction Method

Checks that the reference count is zero before allowing the interfaced object to be destroyed.

Pascal

```
procedure BeforeDestruction; override;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

BeforeDestruction is called automatically immediately before the interfaced object's destructor. It checks that the reference count is zero, and, if it is not, raises an exception.

See Also

RefCount (see page 15526)

Free (see page 22300)

84.1.6.4.5 TInvokableClass.NewInstance Method

Allocates memory for each instance of an invokable object.

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
virtual __fastcall TObject * NewInstance();
```

Description

Constructors call NewInstance automatically. NewInstance calls InstanceSize to determine how much memory to allocate from the heap to contain a particular instance. Do not call NewInstance directly.

TInvokableClass (see page 15520) overrides NewInstance to increment the reference count when a new instance is created. This prevents any constructor from accidentally deleting the instance due to a RefCount (see page 15526) of zero. The AfterConstruction (see page 15524) method decrements the value of RefCount (see page 15526) after all constructors have executed.

See Also

RefCount (see page 15526)

NewInstance (see page 22306)

InstanceSize (see page 22306)

AfterConstruction (see page 15524)

84.1.6.4.6 TInvokableClass.QueryInterface Method

Implements the IInterface QueryInterface method.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

Do not call the protected QueryInterface method directly. QueryInterface is called through the IInterface interface to obtain an interface pointer for the interface identified by the IID parameter. If the object supports the requested interface, it is returned in the Obj parameter and QueryInterface returns S_OK. If the object does not support the interface, QueryInterface returns E_NOINTERFACE.

If QueryInterface successfully returns an interface pointer, it automatically increments the reference count.

See Also

_addref (see page 15523)

84.1.6.5 TInvokableClass Properties

The properties of the TInvokableClass class are listed here.

Public Properties

	Name	Description
R	RefCount (see page 15526)	Indicates the number of interface pointers currently dependent upon the object.

84.1.6.5.1 TInvokableClass.RefCount Property

Indicates the number of interface pointers currently dependent upon the object.

Pascal

```
property RefCount: Integer;
```

C++

```
__property int RefCount;
```

Description

RefCount defines the lifetime of the object. When RefCount is zero, the object is destroyed.

RefCount is incremented by calls to the _AddRef (see page 15523) method and decremented by the _Release (see page 15524) method.

Note:

See Also

_addref (see page 15523)

_release (see page 15524)

84.1.7 TInvokableClassRegistry Class

TInvokableClassRegistry is the type of the invocation registry.

Class Hierarchy

Pascal

```
TInvokableClassRegistry = class(TInterfacedObject);
```

C++

```
class TInvokableClassRegistry : public TInterfacedObject;
```

File

InvokeRegistry

Description

TInvokableClassRegistry manages information about invokable interfaces and the classes that implement them. Invokable interfaces are used in applications that work with Web Services.

Server applications that implement Web Services need to register both the invokable interfaces and the implementation classes. The invoker component (such as THTTPSoapPascalInvoker or THTTPSoapCppInvoker) can then use the invocation registry to locate invokable interfaces and execute calls made to them. In addition, server applications should register the headers that are used with calls to the methods of the invokable interface and the exceptions that such calls can raise.

Client applications that call on Web Services register the invokable interfaces that they call. The remote interfaced object component (THTTPRio) can then use the invocation registry to generate a dynamic vtable for calling the invokable interfaces.

Do not create instances of TInvokableClassRegistry. Instead, use the global InvRegistry (see page 15620) function to access the invocation registry. Most applications need only call the RegisterInterface (see page 15552) and (if a server) the RegisterInvokableClass (see page 15553) methods. The other TInvokableClassRegistry methods are used by invoker components, remote interfaced object components, and the WSDL document importer. The allow these components to look up information about invokable interfaces and their implementation classes, access invokable interfaces, and so on.

See Also

InvRegistry (see page 15620)

IInvokable (see page 22398)

THTTPSoapCppInvoker (see page 19950)

THTTPSoapPascalInvoker (see page 19951)

THTTPRio

84.1.7.1 TInvokableClassRegistry Members

The following tables list the members exposed by TInvokableClassRegistry.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 15529)	Creates and initializes a TInvokableClassRegistry (see page 15526) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 15530)	Frees the memory associated with the TInvokableClassRegistry (see page 15526) object.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetActionURIOfIID (see page 15531)	Returns the SOAP Action header associated with the interface with a specified GUID.
	GetActionURIOfInfo (see page 15532)	Returns the SOAP Action header associated with the type info of a registered interface.
	GetClassFromIntfInfo (see page 15532)	Retrieves the implementation class that corresponds to the type info of an invokable interface.
	GetExceptionInfoForInterface (see page 15533)	Retrieves an array of exceptions that are registered as associated with an invokable interface.
	GetHeaderClass (see page 15533)	Returns the class reference for a header type, given its name and namespace.
	GetHeaderInfoForInterface (see page 15534)	Retrieves an array of headers that are registered as associated with an invokable interface.
	GetHeaderName (see page 15534)	Returns the name of a header that is identified by a specified class reference.
	GetHeaderNamespace (see page 15535)	This is the overview for the GetHeaderNamespace method overload.
	GetInfoForURI (see page 15536)	Retrieves information about a registered invokable interface that is identified by a SOAP action header or the path of an HTTP message.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceCount (see page 15536)	Returns the number of registered invokable interfaces.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceExternalName (see page 15537)	This is the overview for the GetInterfaceExternalName method overload.
	GetInterfaceInfoFromName (see page 15538)	Returns information about a registered invokable interface given its name.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetInterfaceTypeInfo (see page 15538)	Returns the runtime type information (RTTI) of an invokable interface.
	GetIntfInvokeOptions (see page 15538)	This is the overview for the GetIntfInvokeOptions method overload.
	GetInvokableObjectFromClass (see page 15539)	Returns an instance of the implementation class for a registered invokable interface.
	GetMethExternalName (see page 15540)	Returns the name of the operation that corresponds to a method on an invokable interface.
	GetMethInternalName (see page 15540)	Returns the name of the method that corresponds to an operation in a WSDL document.
	GetNamespaceByGUID (see page 15541)	Returns a namespace that is uniquely associated with a registered interface.
	GetParamExternalName (see page 15541)	Returns the name of the part that corresponds to a parameter on a method of an invokable interface.
	GetParamInternalName (see page 15541)	Returns the name of the parameter that corresponds to a part on a message of an operation in a WSDL document.
	GetRegInterfaceEntry (see page 15542)	Returns the internal data that TInvokableClassRegistry (see page 15526) maintains for one of its registered interfaces.
	GetRequestHeaderInfoForInterface (see page 15542)	Retrieves an array of headers that are registered as associated with requests to an invokable interface.

	GetResponseHeaderInfoForInterface (see page 15543)	Retrieves an array of headers that are registered as associated with responses to invocations of an invokable interface.
	GetReturnParamNames (see page 15544)	Returns the list of parameter names that map to return values of interface methods.
	GetUDDIInfo (see page 15544)	This is the overview for the GetUDDIInfo method overload.
	GetWSDLEncoding (see page 15545)	This is the overview for the GetWSDLEncoding method overload.
	HasRegInterfaceImpl (see page 15546)	Indicates whether there is a registered implementation class for an invokable interface.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterAllSOAPActions (see page 15546)	Supplies a set of SOAP Action headers to use for an already-registered interface.
	RegisterDefaultSOAPAction (see page 15547)	Supplies a SOAP Action header to use for an already-registered interface.
	RegisterException (see page 15547)	Registers the remotable exception classes that calls to an invokable interface can raise.
	RegisterExceptionMethod (see page 15548)	Registers methods of an invokable interface that can raise a registered exception class.
	RegisterExternalMethName (see page 15548)	Registers the correspondence between a method name on an invokable interface and the corresponding operation name in a WSDL document.
	RegisterExternalParamName (see page 15549)	Registers the correspondence between a parameter name on an invokable interface and the corresponding part name in a WSDL document.
	RegisterHeaderClass (see page 15549)	This is the overview for the RegisterHeaderClass method overload.
	RegisterHeaderMethod (see page 15551)	Registers methods of an invokable interface that make use of a registered header.
	RegisterInterface (see page 15552)	Registers an invokable interface.
	RegisterInvokableClass (see page 15553)	This is the overview for the RegisterInvokableClass method overload.
	RegisterInvokeOptions (see page 15553)	This is the overview for the RegisterInvokeOptions method overload.
	RegisterReturnParamNames (see page 15554)	Registers a list of parameter names that map to return values of interface methods.
	RegisterUDDIInfo (see page 15555)	Registers the location of the UDDI description that can be used to import the location of a Web Service.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	UnRegisterInterface (see page 15555)	Unregisters an invokable interface.
	UnRegisterInvokableClass (see page 15556)	Unregisters the implementation class for an invokable interface.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

84.1.7.2 TInvokableClassRegistry.Create Constructor

Creates and initializes a TInvokableClassRegistry (see page 15526) object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TInvokableClassRegistry();
```

Description

Do not create your own instances of TInvokableClassRegistry (see page 15526). Instead, use the global InvRegistry (see

page 15620) function, which provides access to a single global instance of TInvokableClassRegistry (see page 15526).

See Also

InvRegistry (see page 15620)

84.1.7.3 TInvokableClassRegistry.Destroy Destructor

Frees the memory associated with the TInvokableClassRegistry (see page 15526) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TInvokableClassRegistry();
```

Description

Do not destroy the global TInvokableClassRegistry (see page 15526) instance that is available through the InvRegistry (see page 15620) function. This object is freed automatically when the application unloads the InvokeRegistry (see page 15493) unit.

See Also

InvRegistry (see page 15620)

84.1.7.4 TInvokableClassRegistry Methods

The methods of the TInvokableClassRegistry class are listed here.

Public Methods

	Name	Description
	GetActionURIOfIID (see page 15531)	Returns the SOAP Action header associated with the interface with a specified GUID.
	GetActionURIOfInfo (see page 15532)	Returns the SOAP Action header associated with the type info of a registered interface.
	GetClassFromIntfInfo (see page 15532)	Retrieves the implementation class that corresponds to the type info of an invokable interface.
	GetExceptionInfoForInterface (see page 15533)	Retrieves an array of exceptions that are registered as associated with an invokable interface.
	GetHeaderClass (see page 15533)	Returns the class reference for a header type, given its name and namespace.
	GetHeaderInfoForInterface (see page 15534)	Retrieves an array of headers that are registered as associated with an invokable interface.
	GetHeaderName (see page 15534)	Returns the name of a header that is identified by a specified class reference.
	GetHeaderNamespace (see page 15535)	This is the overview for the GetHeaderNamespace method overload.
	GetInfoForURI (see page 15536)	Retrieves information about a registered invokable interface that is identified by a SOAP action header or the path of an HTTP message.
	GetInterfaceCount (see page 15536)	Returns the number of registered invokable interfaces.
	GetInterfaceExternalName (see page 15537)	This is the overview for the GetInterfaceExternalName method overload.
	GetInterfaceInfoFromName (see page 15538)	Returns information about a registered invokable interface given its name.
	GetInterfaceTypeInfo (see page 15538)	Returns the runtime type information (RTTI) of an invokable interface.
	GetIntfInvokeOptions (see page 15538)	This is the overview for the GetIntfInvokeOptions method overload.
	GetInvokableObjectFromClass (see page 15539)	Returns an instance of the implementation class for a registered invokable interface.
	GetMethExternalName (see page 15540)	Returns the name of the operation that corresponds to a method on an invokable interface.
	GetMethInternalName (see page 15540)	Returns the name of the method that corresponds to an operation in a WSDL document.

	GetNamespaceByGUID (see page 15541)	Returns a namespace that is uniquely associated with a registered interface.
	GetParamExternalName (see page 15541)	Returns the name of the part that corresponds to a parameter on a method of an invokable interface.
	GetParamInternalName (see page 15541)	Returns the name of the parameter that corresponds to a part on a message of an operation in a WSDL document.
	GetRegInterfaceEntry (see page 15542)	Returns the internal data that TInvokableClassRegistry (see page 15526) maintains for one of its registered interfaces.
	GetRequestHeaderInfoForInterface (see page 15542)	Retrieves an array of headers that are registered as associated with requests to an invokable interface.
	GetResponseHeaderInfoForInterface (see page 15543)	Retrieves an array of headers that are registered as associated with responses to invocations of an invokable interface.
	GetReturnParamNames (see page 15544)	Returns the list of parameter names that map to return values of interface methods.
	GetUDDIInfo (see page 15544)	This is the overview for the GetUDDIInfo method overload.
	GetWSDLEncoding (see page 15545)	This is the overview for the GetWSDLEncoding method overload.
	HasRegInterfaceImpl (see page 15546)	Indicates whether there is a registered implementation class for an invokable interface.
	RegisterAllSOAPActions (see page 15546)	Supplies a set of SOAP Action headers to use for an already-registered interface.
	RegisterDefaultSOAPAction (see page 15547)	Supplies a SOAP Action header to use for an already-registered interface.
	RegisterException (see page 15547)	Registers the remotable exception classes that calls to an invokable interface can raise.
	RegisterExceptionMethod (see page 15548)	Registers methods of an invokable interface that can raise a registered exception class.
	RegisterExternalMethName (see page 15548)	Registers the correspondence between a method name on an invokable interface and the corresponding operation name in a WSDL document.
	RegisterExternalParamName (see page 15549)	Registers the correspondence between a parameter name on an invokable interface and the corresponding part name in a WSDL document.
	RegisterHeaderClass (see page 15549)	This is the overview for the RegisterHeaderClass method overload.
	RegisterHeaderMethod (see page 15551)	Registers methods of an invokable interface that make use of a registered header.
	RegisterInterface (see page 15552)	Registers an invokable interface.
	RegisterInvokableClass (see page 15553)	This is the overview for the RegisterInvokableClass method overload.
	RegisterInvokeOptions (see page 15553)	This is the overview for the RegisterInvokeOptions method overload.
	RegisterReturnParamNames (see page 15554)	Registers a list of parameter names that map to return values of interface methods.
	RegisterUDDIInfo (see page 15555)	Registers the location of the UDDI description that can be used to import the location of a Web Service.
	UnRegisterInterface (see page 15555)	Unregisters an invokable interface.
	UnRegisterInvokableClass (see page 15556)	Unregisters the implementation class for an invokable interface.

84.1.7.4.1 TInvokableClassRegistry.GetActionURIOfIID Method

Returns the SOAP Action header associated with the interface with a specified GUID.

Pascal

```
function GetActionURIOfIID(const AGUID: TGUID): string;
```

C++

```
__fastcall AnsiString GetActionURIOfIID(const TGUID AGUID);
```

Description

Typically, applications do not need to call GetActionURIOfIID. This method is called by remote interfaced objects (THTTPRIO) when generating the SOAP Action header of messages that call a method on a registered interface.

AGUID is the globally unique identifier (GUID) of the interface to call. The interface must be previously registered by a call to the RegisterInterface (see page 15552) method.

GetActionURIOfIID returns the string that should appear as the SOAP Action header on request messages to that interface.

See Also

GetActionURIOfInfo (see page 15532)

RegisterDefaultSOAPAction (see page 15547)

RegisterInterface (see page 15552)

THTTPRIO (see page 19928)

84.1.7.4.2 TInvokableClassRegistry.GetActionURIOfInfo Method

Returns the SOAP Action header associated with the type info of a registered interface.

Pascal

```
function GetActionURIOfInfo(const IntfInfo: PTypeInfo; const MethodName: WideString;
MethodIndex: Integer): string;
```

C++

```
__fastcall AnsiString GetActionURIOfInfo(const PTypeInfo IntfInfo, const BSTR MethodName, int
MethodIndex);
```

Description

GetActionURIOfInfo looks up or generates the SOAP Action header that should be used with messages that call a specified method on an invokable interface.

IntfInfo (see page 15477) is the runtime type information (RTTI) that is compiled for the invokable interface. The interface must have been previously registered by a call to the RegisterInterface (see page 15552) method.

MethodName is the name of the method to call on the specified interface.

MethodIndex indicates which method to use when the interface includes an overload of the method that MethodName specifies. 0 indicates the overload that is declared first, 1 the overload that is declared second, and so on.

GetActionURIOfInfo returns the string that should appear as the SOAP Action header on request messages to the specified method of the specified interface.

See Also

GetActionURIOfIID (see page 15531)

RegisterDefaultSOAPAction (see page 15547)

RegisterInterface (see page 15552)

THTTPRIO (see page 19928)

84.1.7.4.3 TInvokableClassRegistry.GetClassFromIntfInfo Method

Retrieves the implementation class that corresponds to the type info of an invokable interface.

Pascal

```
procedure GetClassFromIntfInfo(Info: PTypeInfo; var AClass: TClass);
```

C++

```
__fastcall GetClassFromIntfInfo(PTypeInfo Info, TClass AClass);
```

Description

Typically, applications do not need to call GetClassFromIntfInfo. This method allows the invoker component

(THTTPSoapPascalInvoker or THTTPSoapCppInvoker) to look up the implementation class for an interface it is invoking.

Info is the runtime type information (RTTI) from a registered invokable interface. The interface must have been previously registered by a call to the RegisterInterface (see page 15552) method.

AClass returns a class reference for the class that implements the invokable interface identified by Info. It can be passed to the GetInvokableObjectFromClass (see page 15539) method to obtain an instance of the implementation class.

See Also

GetInvokableObjectFromClass (see page 15539)

RegisterInvokableClass (see page 15553)

RegisterInterface (see page 15552)

THTTPSoapPascalInvoker (see page 19951)

THTTPSoapCppInvoker (see page 19950)

84.1.7.4.4 TInvokableClassRegistry.GetExceptionInfoForInterface Method

Retrieves an array of exceptions that are registered as associated with an invokable interface.

Pascal

```
function GetExceptionInfoForInterface(Info: PTypeInfo): TExceptionItemArray;
```

C++

```
__fastcall TExceptionItemArray GetExceptionInfoForInterface(PTypeInfo Info);
```

Description

Typically, applications do not need to call GetExceptionInfoForInterface. This method is used internally by the WSDL publisher to generate the WSDL that describes the exceptions that an invokable interface can raise.

Info is the runtime type information (RTTI) from a registered invokable interface. The interface must have been previously registered by a call to the RegisterInterface (see page 15552) method.

GetExceptionInfoForInterface returns an array of IntfExceptionItem (see page 15625) values, each of which describes a remotable exception that was registered for the invokable interface by a call to the RegisterException (see page 15547) method.

See Also

RegisterException (see page 15547)

TWSDLHTMLPublish (see page 26024)

RegisterInterface (see page 15552)

84.1.7.4.5 TInvokableClassRegistry.GetHeaderClass Method

Returns the class reference for a header type, given its name and namespace.

Pascal

```
function GetHeaderClass(Name: WideString; Namespace: WideString): TClass;
```

C++

```
__fastcall TClass GetHeaderClass(BSTR Name, BSTR Namespace);
```

Description

Typically, applications do not need to call GetHeaderClass. This method is used internally to look up the class of a header that is identified by name.

Name is the name of the header node.

Namespace is the URI of the namespace in which Name is defined.

GetHeaderClass returns the class reference for the TSOAPHeader (see page 15596) descendant that represents the identified header.

See Also

RegisterHeaderClass (see page 15550)

TSOAPHeader (see page 15596)

GetHeaderName (see page 15534)

GetHeaderNamespace (see page 15535)

84.1.7.4.6 TInvokableClassRegistry.GetHeaderInfoForInterface Method

Retrieves an array of headers that are registered as associated with an invokable interface.

Pascal

```
function GetHeaderInfoForInterface(Info: PTypeInfo; MethodType: eHeaderMethodType = hmtAll):
THeaderItemArray;
```

C++

```
__fastcall THeaderItemArray GetHeaderInfoForInterface(PTypeInfo Info, eHeaderMethodType
MethodType = hmtAll);
```

Description

Typically, applications do not need to call GetHeaderInfoForInterface. This method is used internally by the WSDL publisher to generate the WSDL that describes the headers that are used in messages about an invokable interface.

Info is the runtime type information (RTTI) from a registered invokable interface. The interface must have been previously registered by a call to the RegisterInterface (see page 15552) method.

MethodType indicates whether to retrieve information about the headers to use on request messages, the headers to use on response messages, or both.

GetHeaderInfoForInterface returns an array of IntfHeaderItem (see page 15626) values, each of which describes a header that was registered for the invokable interface by a call to the RegisterHeaderClass (see page 15550) method.

See Also

RegisterHeaderClass (see page 15550)

TWSDLHTMLPublish (see page 26024)

RegisterInterface (see page 15552)

84.1.7.4.7 TInvokableClassRegistry.GetHeaderName Method

Returns the name of a header that is identified by a specified class reference.

Pascal

```
function GetHeaderName(Info: PTypeInfo; AClass: TClass): WideString;
```

C++

```
__fastcall BSTR GetHeaderName(PTypeInfo Info, TClass AClass);
```

Description

GetHeaderName returns the name of a header given the class that represents it. This is the name supplied to a call to RegisterHeaderClass (see page 15550) when the header was registered. If the header was registered without supplying a name, this is the same name as the type name supplied for the header when it is registered with the remotable type registry.

Info is the runtime type information for the interface associated with the header.

AClass is a class reference to the TSOAPHeader (see page 15596) descendant that represents the header.

GetHeaderName returns the name of the header. This name is defined within the namespace that the GetHeaderNamespace (see page 15535) method returns.

See Also

RegisterHeaderClass (see page 15550)

TSOAPHeader (see page 15596)

GetHeaderNamespace (see page 15535)

GetHeaderClass (see page 15533)

84.1.7.4.8 GetHeaderNamespace Method

This is the overview for the GetHeaderNamespace method overload.

Overload List

	Name	Description
	TInvokableClassRegistry.GetHeaderNamespace (PTypeInfo, TClass) (see page 15535)	Returns the namespace URI for a header that is identified by a specified class reference.
	TInvokableClassRegistry.GetHeaderNamespace (TClass) (see page 15535)	Returns the namespace URI for a header that is identified by a specified class reference.

84.1.7.4.8.1 TInvokableClassRegistry.GetHeaderNamespace Method (PTypeInfo, TClass)

Returns the namespace URI for a header that is identified by a specified class reference.

Pascal

```
function GetHeaderNamespace(Info: PTypeInfo; AClass: TClass): WideString; overload;
function GetHeaderNamespace(AClass: TClass): WideString; overload;
```

C++

```
__fastcall BSTR GetHeaderNamespace(PTypeInfo Info, TClass AClass);
__fastcall BSTR GetHeaderNamespace(TClass AClass);
```

Description

GetHeaderNamespace returns the namespace in which a header's name is defined. This is the namespace URI supplied to a call to RegisterHeaderClass (see page 15550) when the header was registered. If the header was registered without supplying a namespace URI, this is the same namespace as the one associated with the interface that uses the header.

Info is the runtime type information for the interface associated with the header.

AClass is a class reference to the TSOAPHeader (see page 15596) descendant that represents the header.

GetHeaderNamespace returns the namespace URI to use for resolving the name of the header.

See Also

RegisterHeaderClass (see page 15550)

TSOAPHeader (see page 15596)

GetHeaderName (see page 15534)

GetHeaderClass (see page 15533)

84.1.7.4.9 TInvokableClassRegistry.GetInfoForURI Method

Retrieves information about a registered invokable interface that is identified by a SOAP action header or the path of an HTTP message.

Pascal

```
function GetInfoForURI(const PathURI: string; const ActionURI: string; var AClass: TClass; var
IntfInfo: PTypeInfo; var AMeth: string): Boolean;
```

C++

```
__fastcall Boolean GetInfoForURI(const AnsiString PathURI, const AnsiString ActionURI, TClass
AClass, PTypeInfo IntfInfo, AnsiString AMeth);
```

Description

Typically, applications do not need to call GetInfoForURI. This method is called by the invoker component (THTTPSoapPascalInvoker or THTTPSoapCppInvoker) to look up the target of a SOAP message that it is invoking.

PathURI is the path portion of the target URI on the SOAP message. When using a path to identify an invokable interface, the path portion must end in the name of the unit followed by the name of the interface. It is only used if ActionURI is an empty string.

ActionURI is the SOAP action header of the SOAP message. It identifies an interface based on the namespace in the SOAP action header. This namespace must match the namespace supplied to the RegisterInterface (see page 15552) method (or generated by that method if the caller did not supply a namespace). If ActionURI is an empty string, GetInfoForURI uses PathURI instead to identify the target interface.

AClass returns a class reference for the class that implements the invokable interface identified by PathURI or ActionURI.

IntfInfo (see page 15477) returns the runtime type information (RTTI) of the invokable interface identified by PathURI or ActionURI.

AMeth is not currently used.

See Also

DispatchSOAP (see page 19947)

RegisterInterface (see page 15552)

84.1.7.4.10 TInvokableClassRegistry.GetInterfaceCount Method

Returns the number of registered invokable interfaces.

Pascal

```
function GetInterfaceCount: Integer;
```

C++

```
__fastcall int GetInterfaceCount();
```

Description

GetInterfaceCount returns the number of invokable interfaces that have been registered. Each invokable interface is registered by a call to the RegisterInterface (see page 15552) method.

See Also

RegisterInterface (see page 15552)

84.1.7.4.11 GetInterfaceExternalName Method

This is the overview for the GetInterfaceExternalName method overload.

Overload List

	Name	Description
	TInvokableClassRegistry.GetInterfaceExternalName (PTypeInfo) (see page 15537)	Returns the name of the port type that corresponds to an invokable interface.
	TInvokableClassRegistry.GetInterfaceExternalName (PTypeInfo, string) (see page 15537)	Returns the name of the port type that corresponds to an invokable interface.
	TInvokableClassRegistry.GetInterfaceExternalName (PTypeInfo, string, string) (see page 15537)	Returns the name of the port type that corresponds to an invokable interface.

84.1.7.4.11.1 TInvokableClassRegistry.GetInterfaceExternalName Method (PTypeInfo)

Returns the name of the port type that corresponds to an invokable interface.

Pascal

```
function GetInterfaceExternalName(Info: PTypeInfo; const Namespace: string; const
InternalIntfName: string): InvString; overload;
function GetInterfaceExternalName(Info: PTypeInfo; const Namespace: string): InvString;
overload;
function GetInterfaceExternalName(Info: PTypeInfo): InvString; overload;
```

C++

```
__fastcall InvString GetInterfaceExternalName(PTypeInfo Info, const AnsiString Namespace,
const AnsiString InternalIntfName);
__fastcall InvString GetInterfaceExternalName(PTypeInfo Info, const AnsiString Namespace);
__fastcall InvString GetInterfaceExternalName(PTypeInfo Info);
```

Description

GetInterfaceExternalName returns the name of the port type that corresponds to a specified invokable interface. This is the name given to the corresponding port type in a WSDL document. The association between the external name and the interface name is established by the RegisterInterface (see page 15552) method.

Info is the type information pointer for the invokable interface.

Namespace is the namespace assigned to the invokable interface when it was registered by a call to the RegisterInterface (see page 15552) method. If the interface was registered using the automatically-generated namespace, this parameter can be omitted or you can supply an empty string.

InternalIntfName is the name of the invokable interface.

See Also

RegisterInterface (see page 15552)

84.1.7.4.12 TInvokableClassRegistry.GetInterfaceInfoFromName Method

Returns information about a registered invokable interface given its name.

Pascal

```
procedure GetInterfaceInfoFromName(const UnitName: string; const IntfName: string; var Info:
PTypeInfo; var IID: TGUID);
```

C++

```
__fastcall GetInterfaceInfoFromName(const AnsiString UnitName, const AnsiString IntfName,
PTypeInfo Info, TGUID IID);
```

Description

GetInterfaceInfoFromName looks up a registered interface given its name and the name of the unit in which it is declared.

UnitName is the name of the unit that contains the interface definition.

IntfName is the name of the interface.

Info returns the type information that is compiled with the specified interface.

IID returns the globally unique identifier for the specified interface.

See Also

GetInterfaceTypeInfo (see page 15538)

GetInfoForURI (see page 15536)

84.1.7.4.13 TInvokableClassRegistry.GetInterfaceTypeInfo Method

Returns the runtime type information (RTTI) of an invokable interface.

Pascal

```
function GetInterfaceTypeInfo(const AGUID: TGUID): Pointer;
```

C++

```
__fastcall void * GetInterfaceTypeInfo(const TGUID AGUID);
```

Description

GetInterfaceTypeInfo returns a pointer to the RTTI that is compiled with an invokable interface.

AGUID is the globally unique identifier that specifies the desired interface.

See Also

GetInterfaceInfoFromName (see page 15538)

GetInfoForURI (see page 15536)

84.1.7.4.14 GetIntfInvokeOptions Method

This is the overview for the GetIntfInvokeOptions method overload.

Overload List

	Name	Description
	TInvokableClassRegistry.GetIntfInvokeOptions (PTypeInfo) (see page 15539)	Retrieves any flags that customize the way calls to a registered interface are made.
	TInvokableClassRegistry.GetIntfInvokeOptions (TGUID) (see page 15539)	Retrieves any flags that customize the way calls to a registered interface are made.

84.1.7.4.14.1 TInvokableClassRegistry.GetIntfInvokeOptions Method (PTypeInfo)

Retrieves any flags that customize the way calls to a registered interface are made.

Pascal

```
function GetIntfInvokeOptions(const IntfInfo: PTypeInfo): TIntfInvokeOptions; overload;
function GetIntfInvokeOptions(const AGUID: TGUID): TIntfInvokeOptions; overload;
```

C++

```
__fastcall TIntfInvokeOptions GetIntfInvokeOptions(const PTypeInfo IntfInfo);
__fastcall TIntfInvokeOptions GetIntfInvokeOptions(const TGUID AGUID);
```

Description

GetIntfInvokeOptions retrieves any invocation options that have been registered for an interface using the RegisterInvokeOptions (see page 15554) method. Invocation options customize the way a registered interface is called, especially the way such calls are marshaled or unmarshaled. Remote interfaced objects (THTTPRIO) call this method to determine what options they need to set on the component that is the value of their Converter property.

AGUID identifies the registered interface using its GUID.

IntfInfo (see page 15477) identifies the registered interface using its runtime type information (RTTI). You can obtain the type information from an invokable interface by calling the TypeInfo function (delphi) or the __delphirtti operator (C++).

GetIntfInvokeOptions returns a set of flags that indicate how to make calls to the specified interface.

See Also

RegisterInvokeOptions (see page 15554)

Converter (see page 19935)

Options (see page 18176)

RegisterInterface (see page 15552)

84.1.7.4.15 TInvokableClassRegistry.GetInvokableObjectFromClass Method

Returns an instance of the implementation class for a registered invokable interface.

Pascal

```
function GetInvokableObjectFromClass(AClass: TClass): TObject;
```

C++

```
__fastcall TObject * GetInvokableObjectFromClass(TClass AClass);
```

Description

Typically, applications do not need to call GetInvokableObjectFromClass. This method is called by the invoker component (THTTPSoapPascalInvoker or THTTPSoapCppInvoker) to obtain an instance of the class that implements a registered invokable interface. When the invoker is finished with the object, it calls GetWSDLEncoding (see page 15545) to free the instance.

AClass identifies the class whose instance is desired. It must be registered by a call to RegisterInvokableClass (see page 15553). The invoker obtains this class reference by calling the GetInfoForURI (see page 15536) method.

See Also

GetInfoForURI (see page 15536)

RegisterInvokableClass (see page 15553)

GetWSDLEncoding (see page 15545)

84.1.7.4.16 TInvokableClassRegistry.GetMethExternalName Method

Returns the name of the operation that corresponds to a method on an invokable interface.

Pascal

```
function GetMethExternalName(Info: PTypeInfo; const MethodIntName: string): InvString;
```

C++

```
__fastcall InvString GetMethExternalName(PTypeInfo Info, const AnsiString MethodIntName);
```

Description

GetMethExternalName returns the name of the operation that corresponds to a specified method. This is the name used to define the operation for that method in a WSDL document. The association between the external name and the method name is established by the RegisterExternalMethName (see page 15548) method.

Info is the type information pointer for the invokable interface that includes the method.

MethodIntName is the name of the method on that invokable interface.

See Also

RegisterExternalMethName (see page 15548)

GetMethInternalName (see page 15540)

84.1.7.4.17 TInvokableClassRegistry.GetMethInternalName Method

Returns the name of the method that corresponds to an operation in a WSDL document.

Pascal

```
function GetMethInternalName(Info: PTypeInfo; const MethodExtName: InvString): string;
```

C++

```
__fastcall AnsiString GetMethInternalName(PTypeInfo Info, const InvString MethodExtName);
```

Description

GetMethInternalName returns the name of a method on a registered invokable interface that corresponds to a specified operation. The operation name comes from a WSDL document that describes the invokable interface and its methods. The association between the operation name and the method name is established by the RegisterExternalMethName (see page 15548) method.

Info is the type information pointer for the invokable interface that includes the method.

MethodExtName is the name of the operation for which you want the corresponding method name.

See Also

RegisterExternalMethName (see page 15548)

GetMethExternalName (see page 15540)

84.1.7.4.18 TInvokableClassRegistry.GetNamespaceByGUID Method

Returns a namespace that is uniquely associated with a registered interface.

Pascal

```
function GetNamespaceByGUID(const AGUID: TGUID): string;
```

C++

```
__fastcall AnsiString GetNamespaceByGUID(const TGUID AGUID);
```

Description

GetNamespaceByGUID returns the namespace URI that identifies the specified interface. The namespace is originally associated with the interface when it is registered by a call to the RegisterInterface (see page 15552) method. This namespace can be supplied when the interface is registered, or automatically generated by the RegisterInterface (see page 15552) method.

AGUID is the globally unique identifier that identifies the interface whose namespace is desired..

See Also

GetInfoForURI (see page 15536)

RegisterInterface (see page 15552)

84.1.7.4.19 TInvokableClassRegistry.GetParamExternalName Method

Returns the name of the part that corresponds to a parameter on a method of an invokable interface.

Pascal

```
function GetParamExternalName(Info: PTypeInfo; const MethodName: string; const
InternalParamName: string): InvString;
```

C++

```
__fastcall InvString GetParamExternalName(PTypeInfo Info, const AnsiString MethodName, const
AnsiString InternalParamName);
```

Description

GetParamExternalName returns the name of the part that corresponds to a specified method parameter. This is the name used to define the part for that parameter in a WSDL document. The association between the external name and the parameter name is established by the RegisterExternalParamName (see page 15549) method.

Info is the type information pointer for the invokable interface that includes the method.

MethodName is the name of the method on that invokable interface.

InternalParamName is the name of the parameter on the method of the invokable interface.

See Also

RegisterExternalParamName (see page 15549)

GetParamInternalName (see page 15541)

84.1.7.4.20 TInvokableClassRegistry.GetParamInternalName Method

Returns the name of the parameter that corresponds to a part on a message of an operation in a WSDL document.

Pascal

```
function GetParamInternalName(Info: PTypeInfo; const MethodName: string; const
ExternalParamName: InvString): string;
```

C++

```
__fastcall AnsiString GetParamInternalName(PTypeInfo Info, const AnsiString MethodName, const
InvString ExternalParamName);
```

Description

GetParamInternalName returns the name of the method parameter that corresponds to a specified part. This is the parameter of a method on an invokable interface that is registered to correspond to a part that is defined in a WSDL document. The association between the parameter name and the part name is established by the RegisterExternalParamName (see page 15549) method.

Info is the type information pointer for the invokable interface that includes the method.

MethodName is the name of the method on that invokable interface.

ExternalParamName is the name of a part on one of the messages in the operation that corresponds to MethodName. GetParamInternalName returns the corresponding parameter name.

See Also

RegisterExternalParamName (see page 15549)

GetParamExternalName (see page 15541)

84.1.7.4.21 TInvokableClassRegistry.GetRegInterfaceEntry Method

Returns the internal data that TInvokableClassRegistry (see page 15526) maintains for one of its registered interfaces.

Pascal

```
function GetRegInterfaceEntry(Index: Integer): InvRegIntfEntry;
```

C++

```
__fastcall InvRegIntfEntry GetRegInterfaceEntry(int Index);
```

Description

TInvokableClassRegistry (see page 15526) maintains an InvRegIntfEntry (see page 15627) value for every registered invokable interface. GetRegInterfaceEntry provides indexed access to this information.

The InvRegIntfEntry (see page 15627) value for a registered interface is intended for internal use by the invocation registry and the components that use it.

See Also

GetInfoForURI (see page 15536)

GetInterfaceCount (see page 15536)

RegisterDefaultSOAPAction (see page 15547)

84.1.7.4.22 TInvokableClassRegistry.GetRequestHeaderInfoForInterface Method

Retrieves an array of headers that are registered as associated with requests to an invokable interface.

Pascal

```
function GetRequestHeaderInfoForInterface(Info: PTypeInfo): THeaderItemArray;
```

C++

```
__fastcall THeaderItemArray GetRequestHeaderInfoForInterface(PTypeInfo Info);
```

Description

Typically, applications do not need to call GetRequestHeaderInfoForInterface. This method is used internally by the WSDL publisher to generate the WSDL that describes the headers that are used in messages that invoke a method on an invokable interface.

Info is the runtime type information (RTTI) from a registered invokable interface. The interface must have been previously registered by a call to the RegisterInterface (see page 15552) method.

GetRequestHeaderInfoForInterface returns an array of IntfHeaderItem (see page 15626) values, each of which describes a header that was registered for request messages on the invokable interface by a call to the RegisterHeaderClass (see page 15550) method.

See Also

RegisterHeaderClass (see page 15550)

TWSDLHTMLPublish (see page 26024)

RegisterInterface (see page 15552)

GetHeaderInfoForInterface (see page 15534)

84.1.7.4.23 TInvokableClassRegistry.GetResponseHeaderInfoForInterface Method

Retrieves an array of headers that are registered as associated with responses to invocations of an invokable interface.

Pascal

```
function GetResponseHeaderInfoForInterface(Info: PTypeInfo): THeaderItemArray;
```

C++

```
__fastcall THeaderItemArray GetResponseHeaderInfoForInterface(PTypeInfo Info);
```

Description

Typically, applications do not need to call GetResponseHeaderInfoForInterface. This method is used internally by the WSDL publisher to generate the WSDL that describes the headers that are used in response messages that return the results of executing a method on an invokable interface.

Info is the runtime type information (RTTI) from a registered invokable interface. The interface must have been previously registered by a call to the RegisterInterface (see page 15552) method.

GetResponseHeaderInfoForInterface returns an array of IntfHeaderItem (see page 15626) values, each of which describes a header that was registered for response messages on the invokable interface by a call to the RegisterHeaderClass (see page 15550) method.

See Also

RegisterHeaderClass (see page 15550)

TWSDLHTMLPublish (see page 26024)

RegisterInterface (see page 15552)

GetHeaderInfoForInterface (see page 15534)

84.1.7.4.24 TInvokableClassRegistry.GetReturnParamNames Method

Returns the list of parameter names that map to return values of interface methods.

Pascal

```
function GetReturnParamNames(const IntfInfo: PTypeInfo): string;
```

C++

```
__fastcall AnsiString GetReturnParamNames(const PTypeInfo IntfInfo);
```

Description

GetReturnParamNames returns the list of parameter names that have been registered as mapping to return values of methods on a specified interface. Any out messages in a port type with these names are identified as the return value of a method on the corresponding interface. If the application does not explicitly register output parameter names by a call to RegisterReturnParamNames (see page 15554), GetReturnParamNames returns the default list, 'Result;Return'.

IntfInfo (see page 15477) is the type information pointer for the invokable interface for which you want the registered parameter names.

GetReturnParamNames returns the list of parameter names as a semicolon delimited list.

See Also

RegisterReturnParamNames (see page 15554)

84.1.7.4.25 GetUDDIInfo Method

This is the overview for the GetUDDIInfo method overload.

Overload List

	Name	Description
	TInvokableClassRegistry.GetUDDIInfo (PTypeInfo, string, string) (see page 15544)	Retrieves the description of the UDDI location that describes a registered interface.
	TInvokableClassRegistry.GetUDDIInfo (TGUID, string, string) (see page 15544)	Retrieves the description of the UDDI location that describes a registered interface.

84.1.7.4.25.1 TInvokableClassRegistry.GetUDDIInfo Method (PTypeInfo, string, string)

Retrieves the description of the UDDI location that describes a registered interface.

Pascal

```
function GetUDDIInfo(const IntfInfo: PTypeInfo; var Operator: string; var BindingKey: string):
Boolean; overload;
function GetUDDIInfo(const AGUID: TGUID; var Operator: string; var BindingKey: string):
Boolean; overload;
```

C++

```
__fastcall Boolean GetUDDIInfo(const PTypeInfo IntfInfo, AnsiString Operator, AnsiString
BindingKey);
__fastcall Boolean GetUDDIInfo(const TGUID AGUID, AnsiString Operator, AnsiString BindingKey);
```

Description

Typically, applications do not need to call GetUDDIInfo. This method is used internally to provide fail-over support when a

registered interface was imported using the UDDI browser. It retrieves the information that identifies the UDDI entry for a Web Service, so that the application can re-fetch the target URL for a registered interface. The UDDI information is previously registered by a call to the RegisterUDDIInfo (see page 15555) method.

IntfInfo (see page 15477) is the runtime type information (RTTI) from a registered invokable interface. The interface must have been previously registered by a call to the RegisterInterface (see page 15552) method.

AGUID is a GUID that identifies the registered interface. The interface must have been previously registered by a call to the RegisterInterface (see page 15552) method.

Operator returns the URL of the UDDI registry that was used to import the interface definition.

BindingKey is the unique identifier for the TModel that describes the Web Service to which the interface belongs.

GetUDDIInfo returns true if it successfully locates and returns values for the Operator and BindingKey parameters. It returns false if there is no registered UDDI information for the specified interface.

See Also

RegisterUDDIInfo (see page 15555)

RegisterInterface (see page 15552)

84.1.7.4.26 GetWSDLEncoding Method

This is the overview for the GetWSDLEncoding method overload.

Overload List

	Name	Description
	TInvokableClassRegistry.GetWSDLEncoding (PTypeInfo) (see page 15545)	Returns the encoding attribute of the xml header of a document that defines the port type to which the interface corresponds.
	TInvokableClassRegistry.GetWSDLEncoding (PTypeInfo, string) (see page 15545)	Returns the encoding attribute of the xml header of a document that defines the port type to which the interface corresponds.
	TInvokableClassRegistry.GetWSDLEncoding (PTypeInfo, string, string) (see page 15545)	Returns the encoding attribute of the xml header of a document that defines the port type to which the interface corresponds.

84.1.7.4.26.1 TInvokableClassRegistry.GetWSDLEncoding Method (PTypeInfo)

Returns the encoding attribute of the xml header of a document that defines the port type to which the interface corresponds.

Pascal

```
function GetWSDLEncoding(Info: PTypeInfo; const Namespace: string; const InternalIntfName:
string): InvString; overload;
function GetWSDLEncoding(Info: PTypeInfo; const Namespace: string): InvString; overload;
function GetWSDLEncoding(Info: PTypeInfo): InvString; overload;
```

C++

```
__fastcall InvString GetWSDLEncoding(PTypeInfo Info, const AnsiString Namespace, const
AnsiString InternalIntfName);
__fastcall InvString GetWSDLEncoding(PTypeInfo Info, const AnsiString Namespace);
__fastcall InvString GetWSDLEncoding(PTypeInfo Info);
```

Description

GetWSDLEncoding returns the encoding attribute, if any, that is registered with the interface by a call to the RegisterInterface (

see page 15552) method. The encoding attribute is an attribute of the xml header of a WSDL document, and describes the character set that should be used with all the elements that the document defines.

Info points to the type information of the registered interface.

Namespace specifies the namespace that was registered for the interface by the RegisterInterface (see page 15552) method. It can be an empty string.

InternalIntfName is the name of the interface (which may differ from the port type to which it corresponds).

See Also

RegisterInterface (see page 15552)

84.1.7.4.27 TInvokableClassRegistry.HasRegInterfaceImpl Method

Indicates whether there is a registered implementation class for an invokable interface.

Pascal

```
function HasRegInterfaceImpl(Index: Integer): Boolean;
```

C++

```
__fastcall Boolean HasRegInterfaceImpl(int Index);
```

Description

Typically, applications do not need to call HasRegInterfaceImpl. This method is used internally to determine whether an implementation class has been registered for an invokable interface. The RegisterInvokableClass (see page 15553) method registers these implementation classes.

Index identifies the invokable interface for which an implementation class is sought. This index is the same as the index used in the GetRegInterfaceEntry (see page 15542) method, and reflects the order in which interfaces are registered.

HasRegInterfaceImpl returns true if there is a registered implementation class for the specified interface, false otherwise.

See Also

GetInterfaceCount (see page 15536)

GetRegInterfaceEntry (see page 15542)

RegisterInterface (see page 15552)

RegisterInvokableClass (see page 15553)

84.1.7.4.28 TInvokableClassRegistry.RegisterAllSOAPActions Method

Supplies a set of SOAP Action headers to use for an already-registered interface.

Pascal

```
procedure RegisterAllSOAPActions(Info: PTypeInfo; const AllSOAPActions: InvString);
```

C++

```
__fastcall RegisterAllSOAPActions(PTypeInfo Info, const InvString AllSOAPActions);
```

Description

RegisterDefaultSOAPAction (see page 15547) associates a registered invokable interface with a set of SOAP Action headers. A default SOAP Action header is generated (based on the namespace) when you register the interface by a call to the RegisterInterface (see page 15552) method. RegisterAllSOAPActions lets you supply or change the SOAP Action header to a set of headers after the interface is already registered.

Info is the runtime type information (RTTI) for the interface for which you are supplying a set of SOAP Action headers.

AllSOAPActions is a string that includes the SOAP headers to use for calls to the interface that Info identifies. The individual header strings are delimited by a character that is given by the first character in the string.

See Also

RegisterInterface (see page 15552)

RegisterDefaultSOAPAction (see page 15547)

84.1.7.4.29 TInvokableClassRegistry.RegisterDefaultSOAPAction Method

Supplies a SOAP Action header to use for an already-registered interface.

Pascal

```
procedure RegisterDefaultSOAPAction(Info: PTypeInfo; const DefSOAPAction: InvString);
```

C++

```
__fastcall RegisterDefaultSOAPAction(PTypeInfo Info, const InvString DefSOAPAction);
```

Description

RegisterDefaultSOAPAction associates a registered invokable interface with a SOAP Action header. A default SOAP Action header is generated (based on the namespace) when you register the interface by a call to the RegisterInterface (see page 15552) method. RegisterDefaultSOAPAction lets you supply or change the SOAP Action header after the interface is already registered.

Info is the runtime type information (RTTI) for the interface for which you are supplying a default SOAP Action header.

DefSOAPAction is the SOAP Action header to use for calls to the interface that Info identifies. If this is an empty string, the server automatically generates a SOAP Action header based on the namespace in which the corresponding port type is defined.

See Also

RegisterInterface (see page 15552)

THTTPRIO (see page 19928)

RegisterAllSOAPActions (see page 15546)

84.1.7.4.30 TInvokableClassRegistry.RegisterException Method

Registers the remotable exception classes that calls to an invokable interface can raise.

Pascal

```
procedure RegisterException(Info: PTypeInfo; AClass: TClass);
```

C++

```
__fastcall RegisterException(PTypeInfo Info, TClass AClass);
```

Description

Use RegisterException to register a remotable exception class. When an exception class is registered, its definition is exported to WSDL documents so that it can be used to communicate exception information between clients and servers.

Info points to the type information of the invokable interface that can raise the exception. If more than one interface can raise the exception, you must call this method once for each interface type.

AClass describes the exception class. This must be the class reference for an ERemotableException (see page 15496) descendant.

Note: After a call to RegisterException, the exception class is registered so that it is associated with all methods of the specified interface. If only a subset of the interface's methods raise the exception, you can indicate this by calling the RegisterExceptionMethod (see page 15548) method for each method that can raise the exception.

See Also

RegisterInterface (see page 15552)

ERemotableException (see page 15496)

RegisterExceptionMethod (see page 15548)

Creating Custom Exception Classes for Web Services

84.1.7.4.31 TInvokableClassRegistry.RegisterExceptionMethod Method

Registers methods of an invokable interface that can raise a registered exception class.

Pascal

```
procedure RegisterExceptionMethod(Info: PTypeInfo; AClass: TClass; const MethodName: string);
```

C++

```
__fastcall RegisterExceptionMethod(PTypeInfo Info, TClass AClass, const AnsiString MethodName);
```

Description

Call RegisterExceptionMethod for each method of an invokable interface that can raise a registered exception class. The exception class must be previously registered by a call to the RegisterException (see page 15547) method. If you do not call RegisterExceptionMethod, then the WSDL document for the interface treats the exception as associated with all methods of the interface. If you want to limit the use of the exception to a subset of the methods, call RegisterExceptionMethod for each method that can raise the exception. After the first method is registered this way, the exception becomes limited to that method alone (and to any method subsequently registered by another call to RegisterExceptionMethod).

Info points to the type information of the invokable interface that can raise the exception.

AClass describes the exception class. This must be the class reference used to register the exception previously by a call to the RegisterException (see page 15547) method.

MethodName is the name of a method that can raise the exception.

See Also

RegisterInterface (see page 15552)

ERemotableException (see page 15496)

RegisterException (see page 15547)

84.1.7.4.32 TInvokableClassRegistry.RegisterExternalMethName Method

Registers the correspondence between a method name on an invokable interface and the corresponding operation name in a WSDL document.

Pascal

```
procedure RegisterExternalMethName(Info: PTypeInfo; const InternalName: string; const
ExternalName: InvString);
```

C++

```
__fastcall RegisterExternalMethName(PTypeInfo Info, const AnsiString InternalName, const
InvString ExternalName);
```

Description

Use RegisterExternalMethName to create an association between the name of an operation in a WSDL document and the corresponding method of the invokable interface that corresponds to that operation's port type. This allows the invokable interface to represent a port type that includes operations with names that are not legal in Object Pascal (for example, keywords).

Info points to the type information of the invokable interface.

InternalName is the name used for the method of the invokable interface that corresponds to the operation.

ExternalName is the name of the operation that is defined in the WSDL document.

See Also

RegisterExternalParamName (see page 15549)

RegisterInterface (see page 15552)

84.1.7.4.33 TInvokableClassRegistry.RegisterExternalParamName Method

Registers the correspondence between a parameter name on an invokable interface and the corresponding part name in a WSDL document.

Pascal

```
procedure RegisterExternalParamName(Info: PTypeInfo; const MethodName: string; const
InternalName: string; const ExternalName: InvString);
```

C++

```
__fastcall RegisterExternalParamName(PTypeInfo Info, const AnsiString MethodName, const
AnsiString InternalName, const InvString ExternalName);
```

Description

Use RegisterExternalParamName to create an association between the name of a part in a WSDL document and the corresponding parameter on a method of an invokable interface. This allows the invokable interface to handle definitions that have names that are not legal in Object Pascal (for example, keywords).

Info points to the type information of the invokable interface.

MethodName is the name of the method on the invokable interface.

InternalName is the name used for the parameter of the specified method that corresponds to the part with the name specified by ExternalName.

ExternalName is the name of the part that is defined in the WSDL document.

See Also

RegisterExternalMethName (see page 15548)

RegisterInterface (see page 15552)

84.1.7.4.34 RegisterHeaderClass Method

This is the overview for the RegisterHeaderClass method overload.

Overload List

	Name	Description
	TInvokableClassRegistry.RegisterHeaderClass (PTypeInfo, TClass, eHeaderMethodType, Boolean) (see page 15550)	Registers a class that represents a headers which accompanies calls to an invokable interface.
	TInvokableClassRegistry.RegisterHeaderClass (PTypeInfo, TClass, WideString, WideString, eHeaderMethodType, Boolean) (see page 15550)	Registers a class that represents a headers which accompanies calls to an invokable interface.

84.1.7.4.34.1 TInvokableClassRegistry.RegisterHeaderClass Method (PTypeInfo, TClass, WideString, WideString, eHeaderMethodType, Boolean)

Registers a class that represents a headers which accompanies calls to an invokable interface.

Pascal

```
procedure RegisterHeaderClass(Info: PTypeInfo; AClass: TClass; const HeaderName: WideString;
const HeaderNamespace: WideString; DefaultMethodType: eHeaderMethodType = hmtAll; Required:
Boolean = False); overload;
procedure RegisterHeaderClass(Info: PTypeInfo; AClass: TClass; DefaultMethodType:
eHeaderMethodType = hmtAll; Required: Boolean = False); overload;
```

C++

```
__fastcall RegisterHeaderClass(PTypeInfo Info, TClass AClass, const BSTR HeaderName, const
BSTR HeaderNamespace, eHeaderMethodType DefaultMethodType = hmtAll, Boolean Required = False);
__fastcall RegisterHeaderClass(PTypeInfo Info, TClass AClass, eHeaderMethodType
DefaultMethodType = hmtAll, Boolean Required = False);
```

Description

Use RegisterHeaderClass to register a TSOAPHeader (see page 15596) descendant. Registering a TSOAPHeader (see page 15596) descendant serves two purposes:

On servers, when a TSOAPHeader (see page 15596) descendant is registered, its definition is exported to WSDL documents so that applications using that WSDL document can recognize and process the header.

On both servers and clients, registering associates the TSOAPHeader (see page 15596) descendant with a header name and the namespace in which that name is defined.

Info points to the type information of the invokable interface that the header should accompany. If the header should be included in messages invoking more than one interface or in the responses to more than one interface, you must call this method once for each interface type.

AClass describes the TSOAPHeader (see page 15596) descendant that represents the header.

HeaderName is the name of the header, as defined within the header's namespace. If HeaderName is not supplied, the header's name is assumed to be the same as the type name under which it is registered in the remotable type registry.

HeaderNamespace is the URI for the namespace in which the header's name is defined. If HeaderName and HeaderNamespace are not supplied, the header is assumed to be defined in the same namespace as the interface with which it is associated.

DefaultMethodType indicates whether the header is to be used on request messages, response messages, or both. This default value is associated with the interface as a whole. It can be changed for individual methods of the interface by subsequent calls to the RegisterHeaderMethod (see page 15551) method.

Required indicates whether the header must be included in messages about the interface. When Required is true, the header must be included and understood by the recipient of the message. When Required is false, the header is optional. Note that if

the header is required by some but not all messages, you must set this Required parameter to false, and then add the requirement to individual methods by subsequent calls to the RegisterHeaderMethod (see page 15551) method.

Note: After a call to RegisterHeaderClass, the header class is registered so that it is associated with all methods of the specified interface. If only a subset of the interface's methods use the header, or if the header should have a different method type for some of the methods on the interface, or if the header is required by a subset of the methods on the interface, you can indicate this by calling the RegisterHeaderMethod (see page 15551) method for each method that the header accompanies.

See Also

RegisterInterface (see page 15552)

TSOAPHeader (see page 15596)

RegisterHeaderMethod (see page 15551)

84.1.7.4.35 TInvokableClassRegistry.RegisterHeaderMethod Method

Registers methods of an invokable interface that make use of a registered header.

Pascal

```
procedure RegisterHeaderMethod(Info: PTypeInfo; AClass: TClass; const MethodName: string;
MethodType: eHeaderMethodType = hmtAll; Required: Boolean = False); overload;
```

C++

```
__fastcall RegisterHeaderMethod(PTypeInfo Info, TClass AClass, const AnsiString MethodName,
eHeaderMethodType MethodType = hmtAll, Boolean Required = False);
```

Description

Call RegisterHeaderMethod for each method of an invokable interface that uses a registered header type. The header class must be previously registered by a call to the RegisterHeaderClass (see page 15550) method. If you do not call RegisterHeaderMethod, then the WSDL document for the interface treats the header as associated with all methods of the interface. Use RegisterHeaderMethod if you want to

Limit the use of the header to a subset of the methods.

Specify whether the header is used for request messages, response messages, or both in a way that varies from method to method.

Indicate that the header is required for some methods but not others.

Call RegisterHeaderMethod for each method that uses the header. After the first method is registered this way, the header becomes limited to that method alone (and to any method subsequently registered by another call to RegisterHeaderMethod).

Info points to the type information of the invokable interface that the header should accompany.

AClass describes the TSOAPHeader (see page 15596) descendant that represents the header. This class must be previously registered by a call to RegisterHeaderClass (see page 15550).

MethodName is the name of a method that uses the header.

MethodType indicates whether the header is to be used on request messages, response messages, or both.

Required indicates whether the header must be included in messages about the specified method. When Required is true, the header must be included and understood by the recipient of the message. When Required is false, the header is optional.

See Also

RegisterInterface (see page 15552)

TSOAPHeader (see page 15596)

RegisterHeaderClass (see page 15550)

84.1.7.4.36 TInvokableClassRegistry.RegisterInterface Method

Registers an invokable interface.

Pascal

```
procedure RegisterInterface(Info: PTypeInfo; const Namespace: InvString = ''; const
WSDLEncoding: InvString = ''; const Doc: string = ''; const ExtName: InvString = '');
```

C++

```
__fastcall RegisterInterface(PTypeInfo Info, const InvString Namespace = '', const InvString
WSDLEncoding = '', const AnsiString Doc = '', const InvString ExtName = '');
```

Description

Call RegisterInterface to register an invokable interface with the invocation registry. Typically, the call to RegisterInterface is added by the WSDL importer or Web Service wizard. In Delphi, this call goes in the initialization section of the unit that defines the invokable interface. In C++, it is called within a RegTypes method that uses the pragma startup directive.

Once the invokable interface is registered, client applications can statically bind to it using THTTPRio. On the server side, you must also register the implementation class (using the RegisterInvokableClass (see page 15553) method) before the server can respond to incoming SOAP requests.

PInfo is a pointer to the type information for the interface. You can obtain the type information from an invokable interface by calling the TypeInfo function (Delphi) or __delphirtti (C++).

Namespace specifies a namespace URI that uniquely identifies the interface. If Namespace is set to an empty string, RegisterInterface automatically generates a unique string to use as a namespace URI. Note that RegisterInterface generates a SOAP Action header to use based on the Namespace of the interface. You can change this SOAP Action header using the RegisterDefaultSoapAction method.

WSDLEncoding specifies a encoding attribute that appears on the xml header of any WSDL document that includes the definition of this interface (or rather, the port type to which it corresponds). The encoding attribute describes the character set to use in the WSDL document.

Doc is a descriptive string that describes the purpose or use of the interface. It is used if the application publishes a WSDL document that defines this interface.

ExtName is the name of the port type that corresponds to this interface in a WSDL document. This can be an empty string if the port type and interface have the same name. It is useful when a port type in a WSDL document has a name that is not a valid Object Pascal identifier (for example, a keyword).

DefSOAPAction is the SOAP Action header that should be used for messages that represent calls to the methods on this interface. If you do not supply a default SOAP Action header, your application uses an automatically-generated header that is based on the namespace of the interface.

See Also

RegisterInvokableClass (see page 15553)

IInvokable (see page 22398)

THTTPRio

UnRegisterInterface (see page 15555)

RegisterDefaultSOAPAction (see page 15547)

84.1.7.4.37 RegisterInvokableClass Method

This is the overview for the RegisterInvokableClass method overload.

Overload List

	Name	Description
	TInvokableClassRegistry.RegisterInvokableClass (TClass) (see page 15553)	Registers the implementation class for an invokable interface.
	TInvokableClassRegistry.RegisterInvokableClass (TClass, TCreateInstanceProc) (see page 15553)	Registers the implementation class for an invokable interface.

84.1.7.4.37.1 TInvokableClassRegistry.RegisterInvokableClass Method (TClass)

Registers the implementation class for an invokable interface.

Pascal

```
procedure RegisterInvokableClass(AClass: TClass; CreateProc: TCreateInstanceProc); overload;
procedure RegisterInvokableClass(AClass: TClass); overload;
```

C++

```
__fastcall RegisterInvokableClass(TClass AClass, TCreateInstanceProc CreateProc);
__fastcall RegisterInvokableClass(TClass AClass);
```

Description

RegisterInvokableClass registers the class that implements an already-registered invokable interface. It is only needed on server applications that implement Web Services. The interface itself is registered by the RegisterInterface (see page 15552) method.

The invocation registry must be able to instantiate the implementation class. If the class has a virtual constructor that takes no arguments, you do not need to supply a method to create the class instance. If the constructor takes any arguments, of if it is not virtual, you must supply a method to instantiate the class as the value of the CreateProc argument.

AClass is a class reference to the class being registered. In Delphi code, you need only supply the class name. In C++, you can obtain a class reference for a class using the __classid operator.

CreateProc is a procedure that returns an instance of AClass. If CreateProc is nil (Delphi) or NULL (C++), then AClass must be a descendant of TInvokableClass (see page 15520).

See Also

RegisterInterface (see page 15552)

UnRegisterInvokableClass (see page 15556)

TInvokableClass (see page 15520)

84.1.7.4.38 RegisterInvokeOptions Method

This is the overview for the RegisterInvokeOptions method overload.

Overload List

	Name	Description
	TInvokableClassRegistry.RegisterInvokeOptions (PTypeInfo, TIntfInvokeOption) (see page 15554)	Registers flags that customize the way calls to a registered interface are made.
	TInvokableClassRegistry.RegisterInvokeOptions (PTypeInfo, TIntfInvokeOptions) (see page 15554)	Registers flags that customize the way calls to a registered interface are made.

84.1.7.4.38.1 TInvokableClassRegistry.RegisterInvokeOptions Method (PTypeInfo, TIntfInvokeOption)

Registers flags that customize the way calls to a registered interface are made.

Pascal

```
procedure RegisterInvokeOptions(Info: PTypeInfo; const InvokeOption: TIntfInvokeOption);
overload;
procedure RegisterInvokeOptions(Info: PTypeInfo; const InvokeOptions: TIntfInvokeOptions);
overload;
```

C++

```
__fastcall RegisterInvokeOptions(PTypeInfo Info, const TIntfInvokeOption InvokeOption);
__fastcall RegisterInvokeOptions(PTypeInfo Info, const TIntfInvokeOptions InvokeOptions);
```

Description

RegisterInvokeOptions associates the specified invocation flag or flags with a registered interface. The interface must be previously registered by a call to the RegisterInterface (see page 15552) method. Invocation flags customize the way a registered interface is called, especially the way such calls are marshaled or unmarshaled.

Info identifies the registered interface using its runtime type information (RTTI). You can obtain the type information from an invokable interface by calling the TypeInfo function (delphi) or __delphirtti (C++).

InvokeOption specifies a single invocation option to associate with the interface.

InvokeOptions specifies a set of invocation options to associate with the interface.

Note: RegisterInvokeOptions adds the specified option or options to the set of invocation flags already registered for the interface. It does not remove any previously registered options.

See Also

GetIntfInvokeOptions (see page 15539)

Converter (see page 19935)

Options (see page 18176)

RegisterInterface (see page 15552)

84.1.7.4.39 TInvokableClassRegistry.RegisterReturnParamNames Method

Registers a list of parameter names that map to return values of interface methods.

Pascal

```
procedure RegisterReturnParamNames(Info: PTypeInfo; const RetParamNames: string);
```

C++

```
__fastcall RegisterReturnParamNames(PTypeInfo Info, const AnsiString RetParamNames);
```

Description

RegisterReturnParamNames registers a list of parameter names that correspond to the return value of methods on a specified interface. Typically, this method is called by the WSDL importer. It is important that these names correspond exactly to those that the WSDL importer uses if the interface was imported from a WSDL document.

Info is the type information pointer for the invokable interface whose method return values map to the names in the list.

RetParamNames is a semicolon delimited list of parameter names that map to return values of methods on the specified interface. If RetParamNames is an empty string, the methods of the interface never have a return value (only output parameters). If no return parameter names are registered for an interface, the default list, 'Result;Return', is used.

See Also

GetReturnParamNames (see page 15544)

84.1.7.4.40 TInvokableClassRegistry.RegisterUDDIInfo Method

Registers the location of the UDDI description that can be used to import the location of a Web Service.

Pascal

```
procedure RegisterUDDIInfo(Info: PTypeInfo; const Operator: String; const BindingKey: string);
```

C++

```
__fastcall RegisterUDDIInfo(PTypeInfo Info, const AnsiString Operator, const AnsiString
BindingKey);
```

Description

Typically, applications do not need to call RegisterUDDIInfo. This method is called internally when a registered interface is imported using the UDDI browser. It associates the location of the imported UDDI entry with a registered interface so that the application can later provide fail-over support by re-fetching the location of the Web Service.

Info is the runtime type information (RTTI) from a registered invokable interface. The interface must have been previously registered by a call to the RegisterInterface (see page 15552) method.

Operator is the URL of the UDDI registry that was used to import the interface definition.

BindingKey is the unique identifier for the TModel that describes the Web Service to which the interface belongs.

See Also

GetUDDIInfo (see page 15544)

RegisterInterface (see page 15552)

84.1.7.4.41 TInvokableClassRegistry.UnRegisterInterface Method

Unregisters an invokable interface.

Pascal

```
procedure UnRegisterInterface(Info: PTypeInfo);
```

C++

```
__fastcall UnRegisterInterface(PTypeInfo Info);
```

Description

Call UnRegisterInterface to unregister an invokable interface from the invocation registry. In Delphi, this call goes in the finalization section of the unit that defines the invokable interface. In C++, it is called from a method that uses the pragma exit directive.

Once an invokable interface is unregistered, client applications can no longer call it and server applications can't respond to calls directed at the interface methods.

Info is a pointer to the runtime type information for the interface. You can obtain the type information from an invokable interface by calling the TypeInfo function in Delphi or __delphirtti in C++.

See Also

UnRegisterInvokableClass (see page 15556)

IInvokable (see page 22398)

RegisterInterface (see page 15552)

84.1.7.4.42 TInvokableClassRegistry.UnRegisterInvokableClass Method

Unregisters the implementation class for an invokable interface.

Pascal

```
procedure UnRegisterInvokableClass(AClass: TClass);
```

C++

```
__fastcall UnRegisterInvokableClass(TClass AClass);
```

Description

UnRegisterInvokableClass unregisters an implementation class that was previously registered by a call to RegisterInvokableClass (see page 15553). Once a class is unregistered, the invocation registry can no longer create and use instances of the specified class to handle incoming method calls to its corresponding invokable interface.

AClass is a class reference to the class being unregistered.

See Also

UnRegisterInterface (see page 15555)

RegisterInvokableClass (see page 15553)

84.1.8 TRemotable Class

TRemotable is the base class for classes that can be passed as parameters or return values in a Web Service application.

Class Hierarchy

Pascal

```
TRemotable = class;
```

C++

```
class TRemotable;
```

File

InvokeRegistry

Description

Use TRemotable as a base class when defining a class that can be used for parameters or return values on an invokable interface. TRemotable descendants can represent non-scalar types, or scalar types that do not correspond to Object Pascal types. TRemotable is compiled with runtime type information (RTTI) and has a virtual constructor that the remotable class registry recognizes and uses to supply class instances.

Note: To create a class that represents scalar values that do not correspond directly to an Object Pascal type, use TRemotableXS (see page 15580) as a base class instead. TRemotableXS (see page 15580) is a descendant of the TRemotable class.

Note: When writing a TRemotable descendant, you can control whether a property is encoded as an element node or as an attribute node by using the stored directive. Properties declared with a stored directive set to AS_ATTRIBUTE (see page 15645) are encoded as attribute nodes. Properties that are declared without a stored directive, or with a stored directive set to anything other than AS_ATTRIBUTE (see page 15645) are encoded as element nodes.

Both the client application and server application must register a TRemotable descendant with the remotable class registry before they can use it. To register a remotable class, use the RegisterXSClass method of the object that the global RemTypeRegistry (see page 15621) function returns.

On server applications, instances of TRemotable descendants that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TRemotable descendant is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TRemotable instances that it passes as input parameters, and for freeing any TRemotable instances it creates or that are returned as output parameters or method results.

See Also

RemClassRegistry (see page 15620)

RemTypeRegistry (see page 15621)

TRemotableTypeRegistry (see page 15562)

IInvokable (see page 22398)

Using Remotable Objects

84.1.8.1 TRemotable Members

The following tables list the members exposed by TRemotable.

Protected Fields

	Name	Description
	FSerializationOptions (see page 15559)	This is FSerializationOptions, a member of class TRemotable.

Public Methods

	Name	Description
	Create (see page 15558)	Creates and initializes a TRemotable (see page 15556) object.

	Destroy (see page 15558)	Frees the memory associated with the TRemotable (see page 15556) object.
	ObjectToSOAP (see page 15559)	Converts this remotable object to its SOAP representation.
	SOAPToObject (see page 15560)	Initializes this TRemotable (see page 15556) instance to represent the value of its SOAP representation.

Public Properties

	Name	Description
	DataContext (see page 15561)	Specifies the object that controls the lifetime of a TRemotable (see page 15556) instance.
	SerializationOptions (see page 15561)	Indicates how the value of the TRemotable (see page 15556) descendant encodes its value.

84.1.8.2 TRemotable.Create Constructor

Creates and initializes a TRemotable (see page 15556) object.

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TRemotable();
```

Description

The TRemotable (see page 15556) constructor does nothing special. It exists primarily because

The remotable type registry knows how to call it.

It is virtual, so that the remotable type registry can also instantiate descendant classes.

See Also

RemTypeRegistry (see page 15621)

84.1.8.3 TRemotable.Destroy Destructor

Frees the memory associated with the TRemotable (see page 15556) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRemotable();
```

Description

Use the Free method to free a TRemotable (see page 15556) instance. Free checks that the object is not nil before calling Destroy.

On server applications, there is typically no need to explicitly free a TRemotable (see page 15556) instance (for example, one created as a return value or output parameter). By default, when a TRemotable (see page 15556) descendant is created in a method that was called remotely using an invokable interface, it is added to a data context (the value of the DataContext (see page 15561) property). As long as the remotable object belongs to the data context, the data context handles freeing the object. Similarly, TRemotable (see page 15556) instances that are passed in as a parameter belong to the data context.

On client applications, TRemotable (see page 15556) instances created and passed to an invokable interface or returned as a parameter or result of an invokable interface must be freed by the calling application.

See Also

Free (see page 22300)

DataContext (see page 15561)

84.1.8.4 TRemotable Fields

The fields of the TRemotable class are listed here.

Protected Fields

	Name	Description
	FSerializationOptions (see page 15559)	This is FSerializationOptions, a member of class TRemotable.

84.1.8.4.1 TRemotable.FSerializationOptions Field

Pascal

```
FSerializationOptions: TSerializationOptions;
```

C++

```
TSerializationOptions FSerializationOptions;
```

Description

This is FSerializationOptions, a member of class TRemotable.

84.1.8.5 TRemotable Methods

The methods of the TRemotable class are listed here.

Public Methods

	Name	Description
	ObjectToSOAP (see page 15559)	Converts this remotable object to its SOAP representation.
	SOAPToObject (see page 15560)	Initializes this TRemotable (see page 15556) instance to represent the value of its SOAP representation.

84.1.8.5.1 TRemotable.ObjectToSOAP Method

Converts this remotable object to its SOAP representation.

Pascal

```
function ObjectToSOAP(RootNode: IXMLNode; ParentNode: IXMLNode; const ObjConverter:
IObjConverter; const Name: InvString; const URI: InvString; ObjConvOpts:
TObjectConvertOptions; out RefID: InvString): IXMLNode; virtual;
```

C++

```
virtual __fastcall IXMLNode ObjectToSOAP(IXMLNode RootNode, IXMLNode ParentNode, const
IObjConverter ObjConverter, const InvString Name, const InvString URI, TObjectConvertOptions
ObjConvOpts, InvString RefID);
```

Description

Override ObjectToSOAP to customize the way your application converts a TRemotable (see page 15556) descendant to its SOAP representation. In TRemotable (see page 15556), ObjectToSOAP simply calls the ObjInstanceToSOAP method of the converter that is supplied by the ObjConverter parameter.

RootNode is an ancestor of the generated node to which information about the generated node can be added as attributes.

ParentNode is the node in the SOAP representation that should act as a parent to the node that ObjectToSOAP generates.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

Name is the name of the node that is generated.

URI is the URI for the namespace in which Name is defined.

ObjConvOpts are flags that customize the way the node is serialized.

RefID returns a reference ID for the node. This is used if the node is multi-referenced.

ObjectToSOAP returns the XML for the newly generated node.

See Also

ObjInstanceToSOAP (see page 15612)

SOAPToObject (see page 15560)

84.1.8.5.2 TRemotable.SOAPToObject Method

Initializes this TRemotable (see page 15556) instance to represent the value of its SOAP representation.

Pascal

```
procedure SOAPToObject(const RootNode: IXMLNode; const Node: IXMLNode; const ObjConverter:
IObjConverter); virtual;
```

C++

```
virtual __fastcall SOAPToObject(const IXMLNode RootNode, const IXMLNode Node, const
IObjConverter ObjConverter);
```

Description

SOAPToObject initializes the published properties of this object instance so that they reflect the header's SOAP representation. Override this method to customize the way your application creates a specific remotable type from its XML representation.

RootNode is an ancestor node of the node for this object that can contain information about the remotable type's value.

Node is the node in the SOAP envelope that represents the remotable type's value.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

In TRemotable (see page 15556), SOAPToObject simply calls the InitObjectFromSOAP method of the converter supplied by the ObjConverter parameter.

See Also

ObjectToSOAP (see page 15559)

InitObjectFromSOAP (see page 15612)

84.1.8.6 TRemotable Properties

The properties of the TRemotable class are listed here.

Public Properties

	Name	Description
	DataContext (see page 15561)	Specifies the object that controls the lifetime of a TRemotable (see page 15556) instance.
R	SerializationOptions (see page 15561)	Indicates how the value of the TRemotable (see page 15556) descendant encodes its value.

84.1.8.6.1 TRemotable.DataContext Property

Specifies the object that controls the lifetime of a TRemotable (see page 15556) instance.

Pascal

```
property DataContext: TDataContext;
```

C++

```
__property TDataContext DataContext;
```

Description

By default, when a Web Service application creates an instance of a remotable object, that object exists only for the time it takes to marshal the information it represents or to unmarshal that information into the TRemotable (see page 15556) instance and then make a method call that uses the instance. The object specified by DataContext handles the freeing of the TRemotable (see page 15556) instance.

There are times when a server may not want the TRemotable (see page 15556) instance to be so transient. For example, if the object contains state information, it may be more efficient to have a single instance that is used for every message call. If you change DataContext, the TRemotable (see page 15556) instance is removed from the data context, and is not freed when the data context goes away at the end of the method call.

Note: If DataContext is nil (Delphi) or NULL (C++), the application is responsible for freeing the TRemotable (see page 15556) instance. For example, TRemotable (see page 15556) objects created on client applications are created with a nil (Delphi) or NULL (C++) value for DataContext and must be freed by the client application.

See Also

Using Remotable Objects

84.1.8.6.2 TRemotable.SerializationOptions Property

Indicates how the value of the TRemotable (see page 15556) descendant encodes its value.

Pascal

```
property SerializationOptions: TSerializationOptions;
```

C++

```
__property TSerializationOptions SerializationOptions;
```

Description

SerializationOptions is used internally when an instance of TRemotable (see page 15556) is converted to or from its SOAP encoding. The value of this property is a set of flags that configure the mapping between the published properties of the TRemotable (see page 15556) descendant and the corresponding SOAP encoding.

Typically, the value of this read-only property is set by the constructor of a TRemotable (see page 15556) descendant or by the WSDL importer when it registers a remotable class.

See Also

SerializeOptions (see page 15575)

RegisterSerializeOptions (see page 15572)

84.1.9 TRemotableTypeRegistry Class

TRemotableTypeRegistry is the type of the remotable type registry.

Class Hierarchy

InvokeRegistry.TRemotableTypeRegistry

Pascal

```
TRemotableTypeRegistry = class;
```

C++

```
class TRemotableTypeRegistry;
```

File

InvokeRegistry

Description

TRemotableTypeRegistry manages information about remotable classes and types that can be communicated between client applications and a Web Service provider. Remotable classes can be used to represent non-scalar types for parameters and return values, or they can represent exceptions that occur when the Web Service application is trying to execute a request. All remotable classes are compiled with runtime type information (RTTI) and have a virtual constructor that the remotable type registry recognizes and uses to supply class instances. Remotable types include dynamic arrays, enumerated types, and boolean types, which can be used in an invokable interface without resorting to a remotable class representation, but which must be registered because their values require special marshaling.

Both the client application and server application must register remotable classes and types with the remotable type registry before they can use them. Use the RegisterXSClass (see page 15573) method to register a remotable class. Use the RegiterXSInfo method to register a dynamic array type or enumerated type.

Do not create instances of TRemotableTypeRegistry. Instead, use the global RemTypeRegistry (see page 15621) function to access the remotable type registry. Most applications need only call the RegisterXSClass (see page 15573) or RegisterXSInfo (see page 15574) methods. The other TRemotableTypeRegistry methods are used internally to create or look up information about remotable classes and types.

See Also

RemTypeRegistry (see page 15621)

TRemotable (see page 15556)

IInvokable (see page 22398)

ERemotableException (see page 15496)

TInvokableClassRegistry (see page 15526)

84.1.9.1 TRemotableTypeRegistry Members

The following tables list the members exposed by TRemotableTypeRegistry.

Protected Methods

	Name	Description
	Lock (see page 15570)	Prevents other threads from accessing the type registry.
	UnLock (see page 15576)	Allows other threads to access the type registry after it was locked.

Public Methods

	Name	Description
	ClassOptions (see page 15565)	Indicates how to handle multiple references to the same object when converting a specified remotable class to its SOAP representation.
	ClassToURI (see page 15566)	This is the overview for the ClassToURI method overload.
	Create (see page 15563)	Creates and initializes a TRemotableTypeRegistry (see page 15562) object.
	Destroy (see page 15564)	Frees the memory associated with the TRemotableTypeRegistry (see page 15562) object.
	GetExternalPropName (see page 15566)	Returns the name of the XML element that corresponds to a property of a registered remotable class.
	GetInternalPropName (see page 15567)	Returns the name of the property on a remotable class that corresponds to a specified XML element.
	GetURICount (see page 15567)	Returns the number of entries in the remotable type registry.
	GetURIMap (see page 15568)	This is the overview for the GetURIMap method overload.
	GetVarTypeFromXSD (see page 15569)	Returns the variant type that corresponds to a registered remotable type.
	GetXSDInfoForClass (see page 15569)	Returns type name and namespace URI that correspond to a remotable class.
	InfoToURI (see page 15569)	Returns the namespace URI and type name for a registered type.
	IsClassScalar (see page 15570)	Indicates whether a registered class represents a scalar type.
	RegisterExternalPropName (see page 15571)	Registers the correspondence between a property name on a remotable class and the corresponding element name in an XML schema.
	RegisterHolderClsMember (see page 15571)	Registers a remotable type whose type information must be extracted from a holder class.
	RegisterSerializeOptions (see page 15572)	This is the overview for the RegisterSerializeOptions method overload.
	RegisterXSClass (see page 15573)	Registers a remotable class.
	RegisterXSInfo (see page 15574)	Registers a remotable type.
	SerializeOptions (see page 15574)	This is the overview for the SerializeOptions method overload.
	TypeInfoToXSD (see page 15575)	Returns the type name and namespace URI registered with a remotable class or type.
	UnRegisterXSClass (see page 15576)	Unregisters a remotable class.
	UnRegisterXSInfo (see page 15577)	Unregisters a remotable type.
	URIToClass (see page 15577)	This is the overview for the URIToClass method overload.
	URIToInfo (see page 15578)	Returns the type information pointer for a registered remotable class or type.
	VariantToInfo (see page 15578)	Returns the TypeInfo pointer for the type of a specified Variant's data.
	XSDToTypeInfo (see page 15579)	Returns the type information pointer associated with a specified type name.

Public Properties

	Name	Description
	AutoRegisterNativeTypes (see page 15579)	Specifies whether types that are used in an invokable interface or by a remotable class are automatically registered.

84.1.9.2 TRemotableTypeRegistry.Create Constructor

Creates and initializes a TRemotableTypeRegistry (see page 15562) object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TRemotableTypeRegistry();
```

Description

Do not create your own instances of TRemotableTypeRegistry (see page 15562). Instead, use the global RemTypeRegistry (see page 15621) function, which provides access to a single global instance of TRemotableTypeRegistry (see page 15562).

See Also

RemTypeRegistry (see page 15621)

84.1.9.3 TRemotableTypeRegistry.Destroy Destructor

Frees the memory associated with the TRemotableTypeRegistry (see page 15562) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRemotableTypeRegistry();
```

Description

Do not destroy the global TRemotableTypeRegistry (see page 15562) instance that is available through the RemClassRegistry (see page 15620) function. This object is freed automatically when the application unloads the InvokeRegistry (see page 15493) unit.

See Also

RemClassRegistry (see page 15620)

84.1.9.4 TRemotableTypeRegistry Methods

The methods of the TRemotableTypeRegistry class are listed here.

Protected Methods

	Name	Description
	Lock (see page 15570)	Prevents other threads from accessing the type registry.
	UnLock (see page 15576)	Allows other threads to access the type registry after it was locked.

Public Methods

	Name	Description
	ClassOptions (see page 15565)	Indicates how to handle multiple references to the same object when converting a specified remotable class to its SOAP representation.
	ClassToURI (see page 15566)	This is the overview for the ClassToURI method overload.
	GetExternalPropName (see page 15566)	Returns the name of the XML element that corresponds to a property of a registered remotable class.
	GetInternalPropName (see page 15567)	Returns the name of the property on a remotable class that corresponds to a specified XML element.
	GetURICount (see page 15567)	Returns the number of entries in the remotable type registry.
	GetURIMap (see page 15568)	This is the overview for the GetURIMap method overload.
	GetVarTypeFromXSD (see page 15569)	Returns the variant type that corresponds to a registered remotable type.
	GetXSDInfoForClass (see page 15569)	Returns type name and namespace URI that correspond to a remotable class.
	InfoToURI (see page 15569)	Returns the namespace URI and type name for a registered type.
	IsClassScalar (see page 15570)	Indicates whether a registered class represents a scalar type.
	RegisterExternalPropName (see page 15571)	Registers the correspondence between a property name on a remotable class and the corresponding element name in an XML schema.
	RegisterHolderClsMember (see page 15571)	Registers a remotable type whose type information must be extracted from a holder class.

	RegisterSerializeOptions (see page 15572)	This is the overview for the RegisterSerializeOptions method overload.
	RegisterXSClass (see page 15573)	Registers a remotable class.
	RegisterXSInfo (see page 15574)	Registers a remotable type.
	SerializeOptions (see page 15574)	This is the overview for the SerializeOptions method overload.
	TypeInfoToXSD (see page 15575)	Returns the type name and namespace URI registered with a remotable class or type.
	UnRegisterXSClass (see page 15576)	Unregisters a remotable class.
	UnRegisterXSInfo (see page 15577)	Unregisters a remotable type.
	URIToClass (see page 15577)	This is the overview for the URIToClass method overload.
	URIToInfo (see page 15578)	Returns the type information pointer for a registered remotable class or type.
	VariantToInfo (see page 15578)	Returns the TypeInfo pointer for the type of a specified Variant's data.
	XSDToTypeInfo (see page 15579)	Returns the type information pointer associated with a specified type name.

84.1.9.4.1 TRemotableTypeRegistry.ClassOptions Method

Indicates how to handle multiple references to the same object when converting a specified remotable class to its SOAP representation.

Pascal

```
function ClassOptions(AClass: TClass): TObjMultiOptions;
```

C++

```
__fastcall TObjMultiOptions ClassOptions(TClass AClass);
```

Description

When an application converts a remotable class to its SOAP representation, it can convert multiple references to the same object in one of two ways. It can treat the object as if its properties have meaning, but not the reference itself. (That is, two references to the same object are equivalent values, but not the same thing.) Alternately, if the object reference itself has meaning, it can encode the fact that these are multiple references to the same entity in the SOAP packet.

Individual remotable classes can specify how they want this conversion to be handled, overriding any defaults that would otherwise be used. For example, THTTPRIO, THTTPSoapPascalInvoker, and THTTPSoapCppInvoker use TOPToSoapDomConvert to convert between remotable classes and their SOAP representation. TOPToSoapDomConvert has an Options property that specifies, among other things, whether to encode multiple references to the same object by value or as references. Individual remotable classes can supply an override to this default strategy when they are registered using the RegisterXSClass (see page 15573) method.

ClassOptions returns how to handle multiple references of a specified class.

AClass is a class reference for the registered class that you want to know how to convert.

See Also

RegisterXSClass (see page 15573)

Options (see page 18176)

IOPConvert (see page 18122)

THTTPSoapCppInvoker (see page 19950)

THTTPSoapPascalInvoker (see page 19951)

THTTPRIO (see page 19928)

TOPToSoapDomConvert (see page 18137)

84.1.9.4.2 ClassToURI Method

This is the overview for the ClassToURI method overload.

Overload List

	Name	Description
	TRemotableTypeRegistry.ClassToURI (TClass, WideString, WideString) (see page 15566)	Returns the namespace URI associated with a specified remotable class.
	TRemotableTypeRegistry.ClassToURI (TClass, WideString, WideString, Boolean, Boolean) (see page 15566)	Returns the namespace URI associated with a specified remotable class.

84.1.9.4.2.1 TRemotableTypeRegistry.ClassToURI Method (TClass, WideString, WideString)

Returns the namespace URI associated with a specified remotable class.

Pascal

```
function ClassToURI(AClass: TClass; var URI: WideString; var Name: WideString; var IsScalar:
Boolean; tryToRegister: Boolean = True): Boolean; overload;
function ClassToURI(AClass: TClass; var URI: WideString; var Name: WideString): Boolean;
overload;
```

C++

```
__fastcall Boolean ClassToURI(TClass AClass, BSTR URI, BSTR Name, Boolean IsScalar, Boolean
tryToRegister = True);
__fastcall Boolean ClassToURI(TClass AClass, BSTR URI, BSTR Name);
```

Description

ClassToURI looks up a registered remotable class and retrieves its namespace URI and typename. The association of the class and its namespace URI and typename is established when you call RegisterXSClass (see page 15573) to register the remotable class.

AClass is a class reference for the remotable class you want to look up.

URI returns the namespace URI that is associated with the class.

Name returns the registered name of the type the class represents.

IsScalar indicates whether the class corresponds to a scalar type in a WSDL document.

tryToRegister indicates whether the remotable type registry should auto-register the class that AClass specifies if it is not already registered. This parameter is only used if the AutoRegisterNativeTypes (see page 15579) property is true.

See Also

RegisterXSClass (see page 15573)

URIToClass (see page 15577)

AutoRegisterNativeTypes (see page 15579)

84.1.9.4.3 TRemotableTypeRegistry.GetExternalPropName Method

Returns the name of the XML element that corresponds to a property of a registered remotable class.

Pascal

```
function GetExternalPropName(Info: PTypeInfo; InternalName: string): InvString;
```

C++

```
__fastcall InvString GetExternalPropName(PTypeInfo Info, AnsiString InternalName);
```

Description

Use GetExternalPropName to obtain the name of the XML element that was registered as the XML schema analog of a property on a registered remotable class. The association between the element and the property must be previously registered by the RegisterExternalPropname method.

Info points to the type information of the remotable class that has the property.

InternalName is the name of the property from which you want to obtain the corresponding XML element name.

See Also

GetInternalPropName (see page 15567)

RegisterExternalPropName (see page 15571)

84.1.9.4.4 TRemotableTypeRegistry.GetInternalPropName Method

Returns the name of the property on a remotable class that corresponds to a specified XML element.

Pascal

```
function GetInternalPropName(Info: PTypeInfo; ExternalName: InvString): string;
```

C++

```
__fastcall AnsiString GetInternalPropName(PTypeInfo Info, InvString ExternalName);
```

Description

Use GetInternalPropName to obtain the name of the property on a registered remotable class that corresponds to a specified XML element. The association between the element and the property must be previously registered by the RegisterExternalPropname method.

Info points to the type information of the remotable class that has the property.

ExternalName is the name of an XML element that is part of a complex type defined in an XML schema. The remotable class identified by Info must represent the complex type that includes ExternalName.

See Also

GetExternalPropName (see page 15566)

RegisterExternalPropName (see page 15571)

84.1.9.4.5 TRemotableTypeRegistry.GetURICount Method

Returns the number of entries in the remotable type registry.

Pascal

```
function GetURICount: Integer;
```

C++

```
__fastcall int GetURICount();
```

Description

GetURICount returns the number of entries in the remotable type registry. This value can be used as an upper bound for the index parameter of the GetURIMap (see page 15568) method.

Each entry in the remotable type registry represents a registered class or a registered type. The RegisterXSClass (see page 15573) method adds entries for classes, and the The RegisterXSInfo (see page 15574) method adds entries for types. Each entry associates the class or type with a unique namespace URI.

See Also

GetURIMap (see page 15568)

RegisterXSClass (see page 15573)

RegisterXSInfo (see page 15574)

84.1.9.4.6 GetURIMap Method

This is the overview for the GetURIMap method overload.

Overload List

	Name	Description
	TRemotableTypeRegistry.GetURIMap (Integer) (see page 15568)	Returns an entry from the remotable type registry.
	TRemotableTypeRegistry.GetURIMap (Integer, TRemRegEntry) (see page 15568)	Returns an entry from the remotable type registry.

84.1.9.4.6.1 TRemotableTypeRegistry.GetURIMap Method (Integer)

Returns an entry from the remotable type registry.

Pascal

```
function GetURIMap(Index: Integer): TRemRegEntry; overload;
function GetURIMap(Index: Integer; out RegEntry: TRemRegEntry): boolean; overload;
```

C++

```
__fastcall TRemRegEntry GetURIMap(int Index);
__fastcall boolean GetURIMap(int Index, TRemRegEntry RegEntry);
```

Description

GetURIMap returns an entry from the remotable type registry that is identified by its index. It is useful for operations that iterate over all the registered entries. Use the GetURICount (see page 15567) method to obtain an upper bound for the index.

Index is the index of the entry you want to see.

Using the first syntax, GetURIMap returns a TRemRegEntry (see page 15634) value that describes a registered class or type.

Using the second syntax, the RegEntry parameter returns the TRemRegEntry (see page 15634) at the specified index. Using this syntax, GetURIMap returns true if Index is a valid index, false if Index is out of range.

The TRemRegEntry (see page 15634) value that this method returns is used to represent both remotable classes, which are added using the RegisterXSClass (see page 15573) method, and remotable types, which are added using the RegisterXSInfo (see page 15574) method.

See Also

GetURICount (see page 15567)

RegisterXSClass (see page 15573)

84.1.9.4.7 TRemotableTypeRegistry.GetVarTypeFromXSD Method

Returns the variant type that corresponds to a registered remotable type.

Pascal

```
function GetVarTypeFromXSD(const URI: InvString; const TypeName: InvString): TVarType;
```

C++

```
__fastcall TVarType GetVarTypeFromXSD(const InvString URI, const InvString TypeName);
```

Description

GetVarTypeFromXSD returns the built-in Variant type that corresponds most naturally to a remotable type that was registered by a call to the RegisterXSInfo (see page 15574) method.

URI is the namespace URI associated with the type when it was registered.

TypeName is the type name associated with the type when it was registered.

See Also

VariantToInfo (see page 15578)

XSDToTypeInfo (see page 15579)

URIToInfo (see page 15578)

RegisterXSInfo (see page 15574)

84.1.9.4.8 TRemotableTypeRegistry.GetXSDInfoForClass Method

Returns type name and namespace URI that correspond to a remotable class.

Pascal

```
procedure GetXSDInfoForClass(Info: PTypeInfo; var URI: WideString; var TypeName: WideString);
```

C++

```
__fastcall GetXSDInfoForClass(PTypeInfo Info, BSTR URI, BSTR TypeName);
```

Description

GetXSDInfoForClass returns the type name under which a remotable class is registered, along with the namespace URI in which that type name is defined. The class must be previously registered by a call to the RegisterXSClass (see page 15573) method.

Info is the type info pointer for the registered remotable class.

URI returns the namespace URI that was associated with the class when it was registered.

TypeName returns the type name that was associated with the class when it was registered.

See Also

RegisterXSClass (see page 15573)

XSDToTypeInfo (see page 15579)

84.1.9.4.9 TRemotableTypeRegistry.InfoToURI Method

Returns the namespace URI and type name for a registered type.

Pascal

```
function InfoToURI(Info: PTypeInfo; var URI: WideString; var Name: WideString; var IsScalar:
Boolean; tryToRegister: Boolean = True): Boolean;
```

C++

```
__fastcall Boolean InfoToURI(PTypeInfo Info, BSTR URI, BSTR Name, Boolean IsScalar, Boolean
tryToRegister = True);
```

Description

InfoToURI returns the namespace URI and type name that is associated with the specified type description. This association is established when you register a type by calling the RegisterXSInfo (see page 15574) method.

Info is a pointer to the runtime type information (RTTI) for the registered type.

URI returns the namespace URI that is associated with the type.

Name returns the registered name of the type.

IsScalar indicates whether the type corresponds to a scalar type in a WSDL document.

tryToRegister indicates whether the remotable type registry should auto-register the type that Info specifies if it is not already registered. This parameter is only used if the AutoRegisterNativeTypes (see page 15579) property is true.

See Also

RegisterXSInfo (see page 15574)

AutoRegisterNativeTypes (see page 15579)

84.1.9.4.10 TRemotableTypeRegistry.IsClassScalar Method

Indicates whether a registered class represents a scalar type.

Pascal

```
function IsClassScalar(AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean IsClassScalar(TClass AClass);
```

Description

IsClassScalar returns true if the specified class represents a scalar type. Classes (see page 2685) that represent scalar types descend from the TRemotableXS (see page 15580) class, and can convert their value to or from a string.

AClass identifes the class to check. It must be previously registered by a call to RegisterXSClass (see page 15573).

See Also

RegisterXSClass (see page 15573)

TRemotableXS (see page 15580)

84.1.9.4.11 TRemotableTypeRegistry.Lock Method

Prevents other threads from accessing the type registry.

Pascal

```
procedure Lock; virtual;
```

C++

```
virtual __fastcall Lock();
```

Description

Lock uses a critical section to prevent simultaneous use of the type registry by multiple threads in the same application. Once one thread has called Lock to obtain access to the type registry, no other thread can return from the Lock method until the first thread has called the UnLock (see page 15576) method to release its lock.

Lock is called automatically by the methods of TRemotableTypeRegistry (see page 15562).

See Also

UnLock (see page 15576)

84.1.9.4.12 TRemotableTypeRegistry.RegisterExternalPropName Method

Registers the correspondence between a property name on a remotable class and the corresponding element name in an XML schema.

Pascal

```
procedure RegisterExternalPropName(Info: PTypeInfo; const InternalName: string; const
ExternalName: InvString);
```

C++

```
__fastcall RegisterExternalPropName(PTypeInfo Info, const AnsiString InternalName, const
InvString ExternalName);
```

Description

Use RegisterExternalPropName to create an association between the name of an element in an XML schema complex type and the corresponding property of the remotable class that represents that type. This allows the remotable class to represent complex types that include elements with names that are not valid (for example, keywords).

Info points to the type information of the remotable class that corresponds to a complex type defined in an XML schema.

InternalName is the name used for the property of the remotable class that corresponds to the element.

ExternalName is the name of the element that is part of the complex type that the XML schema defines.

84.1.9.4.13 TRemotableTypeRegistry.RegisterHolderClsMember Method

Registers a remotable type whose type information must be extracted from a holder class.

Pascal

```
procedure RegisterHolderClsMember(ClsTypeInfo: PTypeInfo; const URI: WideString = ''; const
Name: WideString = ''; const ExtName: WideString = '');
```

C++

```
__fastcall RegisterHolderClsMember(PTypeInfo ClsTypeInfo, const BSTR URI = '', const BSTR Name
= '', const BSTR ExtName = '');
```

Description

Use RegisterHolderClsMember to register a type that is used as the type of the only published property of a holder class. This method is used primarily for C++ applications, which can contain types that require a holder class so that the compiler can extract their runtime type information (RTTI). If the holder class contains more than one published property, use the RegisterXSInfo (see page 15574) method after extracting the type information from the property using the global GetClsMemberTypeInfo function.

Info points to the type information of the holder class that has a single published member whose type you want to register.

URI is the namespace URI to associate with the type that is registered. This should uniquely identify the type. If you omit this parameter or supply an empty string, RegisterXSInfo (see page 15574) generates a namespace URI for you, based on the name of the type, the name of the unit in which it is declared, and the value of the global AppNameSpacePrefix (see page 15644) variable. Note that the namespace URI must be the same in both client and server applications. In Delphi, if you do not explicitly supply a namespace URI, the class must be declared in a unit with the same name, and the value of AppNameSpacePrefix (see page 15644) must be the same in both applications. In C++, the client application must always explicitly supply a namespace.

Name is the name of the type as it appears in native code.

ExtName is the name of the type as it appears in encoded method calls and WSDL documents.

See Also

GetClsMemberTypeInfo

RegisterXSInfo (see page 15574)

84.1.9.4.14 RegisterSerializeOptions Method

This is the overview for the RegisterSerializeOptions method overload.

Overload List

	Name	Description
	TRemotableTypeRegistry.RegisterSerializeOptions (PTypeInfo, TSerializationOptions) (see page 15572)	Registers a set of serialization options that should be used with a registered remotable class or type.
	TRemotableTypeRegistry.RegisterSerializeOptions (TClass, TSerializationOptions) (see page 15572)	Registers a set of serialization options that should be used with a registered remotable class or type.

84.1.9.4.14.1 TRemotableTypeRegistry.RegisterSerializeOptions Method (PTypeInfo, TSerializationOptions)

Registers a set of serialization options that should be used with a registered remotable class or type.

Pascal

```
procedure RegisterSerializeOptions(Info: PTypeInfo; SerialOpt: TSerializationOptions);
overload;
procedure RegisterSerializeOptions(AClass: TClass; SerialOpt: TSerializationOptions); overload;
```

C++

```
__fastcall RegisterSerializeOptions(PTypeInfo Info, TSerializationOptions SerialOpt);
__fastcall RegisterSerializeOptions(TClass AClass, TSerializationOptions SerialOpt);
```

Description

RegisterSerializeOptions associates a registered remotable class or type with a set of flags that indicate how to encode or decode its value. Typically, this method is called by the WSDL importer.

AClass identifes the class for which you are supplying serialization options. It must be previously registered by a call to RegisterXSClass (see page 15573).

Info is the runtime type information for a type when you are supplying serialization options for a type rather than a class. It must be previously registered by a call to RegisterXSInfo (see page 15574).

SerialOpt is the set of flags that customize how the class is encoded or decoded.

See Also

SerializationOptions (see page 15561)

SerializeOptions (see page 15575)

RegisterXSClass (see page 15573)

RegisterXSInfo (see page 15574)

Defining and Using SOAP Headers

84.1.9.4.15 TRemotableTypeRegistry.RegisterXSClass Method

Registers a remotable class.

Pascal

```
procedure RegisterXSClass(AClass: TClass; const URI: WideString = ''; const Name: WideString =
''; const ExtName: WideString = ''; IsScalar: Boolean = False; MultiRefOpt: TObjMultiOptions =
ocDefault);
```

C++

```
__fastcall RegisterXSClass(TClass AClass, const BSTR URI = '', const BSTR Name = '', const
BSTR ExtName = '', Boolean IsScalar = False, TObjMultiOptions MultiRefOpt = ocDefault);
```

Description

Call RegisterXSClass to register a remotable class so that it can be communicated between client applications and a Web Service provider. All remotable classes are compiled with runtime type information (RTTI) and have a virtual constructor that the remotable type registry recognizes and uses to supply class instances.

When the remotable class is a TRemotable (see page 15556) descendant, it can be used to represent non-scalar types for parameters and return values on an invokable interface. When it is an ERemotableException (see page 15496) descendant, it allows the Web Service application to send clients a fault packet that includes all the published properties of the exception class.

AClass is the class reference for the remotable class. In Delphi, you specify this in code using the class name. In C++, you can obtain this class reference for a class using the __classid keyword.

URI is the namespace URI to associate with that class. This should uniquely identify the class. If you omit this parameter (or supply an empty string), RegisterXSClass generates a namespace URI for you, based on the name of the class, the name of the unit in which it is declared, and the value of the global AppNameSpacePrefix (see page 15644) variable. Note that the namespace URI must be the same in both client and server applications. Thus, if you do not explicitly supply a namespace URI, the class must be declared in a unit with the same name, and the value of AppNameSpacePrefix (see page 15644) must be the same in both applications.

Name is the name of a type that the class represents. This is used for the mapping between the class and its definition in a WSDL document.

ExtName is the same as Name.

IsScalar indicates whether the class represents a scalar type. Remotable classes that represent scalar types must implement the XSToNative and NativeToXS methods.

MultiRefOpt indicates how multiple references to the same instance of the class should be handled when converting them to their SOAP representation. For more information, see the ClassOptions (see page 15565) method.

See Also

TRemotable (see page 15556)

ERemotableException (see page 15496)

AppNameSPacePrefix

UnRegisterXSClass (see page 15576)

ClassOptions (see page 15565)

Creating Custom Exception Classes for Web Services

Registering Nonscalar Types

84.1.9.4.16 TRemotableTypeRegistry.RegisterXSInfo Method

Registers a remotable type.

Pascal

```
procedure RegisterXSInfo(Info: PTypeInfo; const URI: WideString = ''; const Name: WideString =
''; const ExtName: WideString = '');
```

C++

```
__fastcall RegisterXSInfo(PTypeInfo Info, const BSTR URI = '', const BSTR Name = '', const
BSTR ExtName = '');
```

Description

Call RegisterXSInfo to register a type that requires special marshaling but does not need to be represented by a TRemotable (see page 15556) descendant before it can be marshaled for transport to or from a Web Services provider. Such types include dynamic arrays such as the dynamic array types defined in the Types (see page 23293) unit (Delphi) or sysdyn.h (C++), enumerated types, and boolean types (Delphi) or types defined using a typedef statement (C++).

Info is the TypeInfo pointer for the type. It can be obtained using the global TypeInfo function (Delphi) or the __delphirtti keyword (C++). In C++, it is sometimes obtained from a global function such as GetClsMemberTypeInfo, which extracts the type information from a holder class.

URI is the namespace URI to associate with that type. This should uniquely identify the type. If you omit this parameter or supply an empty string, RegisterXSInfo generates a namespace URI for you, based on the name of the type, the name of the unit in which it is declared, and the value of the global AppNameSpacePrefix (see page 15644) variable. Note that the namespace URI must be the same in both client and server applications. In Delphi, if you do not explicitly supply a namespace URI, the class must be declared in a unit with the same name, and the value of AppNameSpacePrefix (see page 15644) must be the same in both applications. In C++, the client application must always explicitly supply a namespace.

Name is the name of the type as it appears in native code.

ExtName is the name of the type as it appears in encoded method calls and WSDL documents.

See Also

AppNameSPacePrefix

UnRegisterXSInfo (see page 15577)

GetClsMemberTypeInfo

Registering Nonscalar Types

84.1.9.4.17 SerializeOptions Method

This is the overview for the SerializeOptions method overload.

Overload List

	Name	Description
	TRemotableTypeRegistry.SerializeOptions (PTypeInfo) (see page 15575)	Returns the serialization options for a registered remotable class or type.
	TRemotableTypeRegistry.SerializeOptions (TClass) (see page 15575)	Returns the serialization options for a registered remotable class or type.

84.1.9.4.17.1 TRemotableTypeRegistry.SerializeOptions Method (PTypeInfo)

Returns the serialization options for a registered remotable class or type.

Pascal

```
function SerializeOptions(Info: PTypeInfo): TSerializationOptions; overload;
function SerializeOptions(AClass: TClass): TSerializationOptions; overload;
```

C++

```
__fastcall TSerializationOptions SerializeOptions(PTypeInfo Info);
__fastcall TSerializationOptions SerializeOptions(TClass AClass);
```

Description

SerializeOptions is used internally to retrieve the flags that describe how a remotable class or type is encoded and decoded. These flags are associated with the remotable class or type by calling the RegisterSerializeOptions (see page 15572) method.

AClass is the registered class for which you want to retrieve serialization options.

Info is the runtime type information for the registered type for which you want to retrieve serialization options.

SerializeOptions returns the value of SerializationOptions (see page 15630) for the class that AClass specifies or the registered options for the type that Info specifies..

See Also

SerializationOptions (see page 15561)

RegisterSerializeOptions (see page 15572)

84.1.9.4.18 TRemotableTypeRegistry.TypeInfoToXSD Method

Returns the type name and namespace URI registered with a remotable class or type.

Pascal

```
function TypeInfoToXSD(Info: PTypeInfo; var URI: WideString; var TypeName: WideString):
Boolean;
```

C++

```
__fastcall Boolean TypeInfoToXSD(PTypeInfo Info, BSTR URI, BSTR TypeName);
```

Description

TypInfoToXSD returns a type name and namespace URI from a type information pointer. The type information can describe either a remotable class that was previously registered by a call to RegisterXSClass (see page 15573), or a remotable type that was previously registered by a call to RegisterXSInfo (see page 15574).

Info is the type information pointer that describes the class or type.

URI returns the namespace URI that was associated with the class or type when it was registered.

TypeName returns the type name that was associated with the class or type when it was registered.

See Also

XSDToTypeInfo (see page 15579)

RegisterXSInfo (see page 15574)

RegisterXSClass (see page 15573)

InfoToURI (see page 15569)

84.1.9.4.19 TRemotableTypeRegistry.UnLock Method

Allows other threads to access the type registry after it was locked.

Pascal

```
procedure UnLock; virtual;
```

C++

```
virtual __fastcall UnLock();
```

Description

UnLock exits the critical section that the Lock (see page 15570) method entered. It reenables other threads access to the type registry after a call to the Lock (see page 15570) method.

Every call to the Lock (see page 15570) method must be paired by a call to UnLock when the locking thread is finished with the type registry. The methods of TRemotableTypeRegistry (see page 15562) that call the Lock (see page 15570) method also call the UnLock method when they are finished.

See Also

Lock (see page 15570)

84.1.9.4.20 TRemotableTypeRegistry.UnRegisterXSClass Method

Unregisters a remotable class.

Pascal

```
procedure UnRegisterXSClass(AClass: TClass);
```

C++

```
__fastcall UnRegisterXSClass(TClass AClass);
```

Description

Call UnRegisterXSClass to Unregister a remotable class that was previously registered by a call to the RegisterXSClass (see page 15573) method. In Delphi, this call is made in the finalization section of the unit that implements the remotable class. In C++, it is made from a method that uses the pragma exit directive in the unit that implements the remotable class.

AClass is the class reference for the remotable class. In Delphi code, you specify this using the class name. In C++, you can obtain this class reference for a class using the __classid keyword.

See Also

TRemotable (see page 15556)

ERemotableException (see page 15496)

RegisterXSClass (see page 15573)

84.1.9.4.21 TRemotableTypeRegistry.UnRegisterXSInfo Method

Unregisters a remotable type.

Pascal

```
procedure UnRegisterXSInfo(Info: PTypeInfo);
```

C++

```
__fastcall UnRegisterXSInfo(PTypeInfo Info);
```

Description

Call UnRegisterXSInfo to unregister a type that was previously registered (directly or indirectly) by a call to the RegisterXSInfo (see page 15574) method.

Info is the TypeInfo pointer for the type. It is the same as the Info parameter of the RegisterXSInfo (see page 15574) method.

See Also

RegisterXSInfo (see page 15574)

84.1.9.4.22 URIToClass Method

This is the overview for the URIToClass method overload.

Overload List

	Name	Description
	TRemotableTypeRegistry.URIToClass (WideString, WideString) (see page 15577)	Returns the class associated with a specified namespace URI.
	TRemotableTypeRegistry.URIToClass (WideString, WideString, Boolean) (see page 15577)	Returns the class associated with a specified namespace URI.

84.1.9.4.22.1 TRemotableTypeRegistry.URIToClass Method (WideString, WideString)

Returns the class associated with a specified namespace URI.

Pascal

```
function URIToClass(const URI: WideString; const Name: WideString; var IsScalar: Boolean):
TClass; overload;
function URIToClass(const URI: WideString; const Name: WideString): TClass; overload;
```

C++

```
__fastcall TClass URIToClass(const BSTR URI, const BSTR Name, Boolean IsScalar);
__fastcall TClass URIToClass(const BSTR URI, const BSTR Name);
```

Description

URIToClass looks up a registered remotable class given its namespace URI and typename.

URI is the namespace URI associated with the class. This association is established by the RegisterXSClass (see page 15573) method.

Name is the name of the type that the class represents. If the class does not have a corresponding type name, Name should be an empty string.

IsScalar returns true if the class represents a scalar type, false otherwise.

URIToClass returns the class reference for the class identified by URI (and Name).

See Also

RegisterXSClass (see page 15573)

ClassToURI (see page 15566)

84.1.9.4.23 TRemotableTypeRegistry.URIToInfo Method

Returns the type information pointer for a registered remotable class or type.

Pascal

```
function URIToInfo(const URI: WideString; const Name: WideString): PTypeInfo;
```

C++

```
__fastcall PTypeInfo URIToInfo(const BSTR URI, const BSTR Name);
```

Description

URIToInfo looks up a registered remotable class or type given its namespace URI and typename. It returns the type information pointer that describes the type or class.

URI is the namespace URI associated with the class or type. This association is established by the RegisterXSClass (see page 15573) or RegisterXSInfo (see page 15574) method.

Name is the type name that was associated with the class or type when it was registered.

Tip: To look up the type information for a namespace and type that may possibly represent a built-in XML schema type, use the XSDToTypeInfo (see page 15579) method instead.

See Also

RegisterXSClass (see page 15573)

InfoToURI (see page 15569)

RegisterXSInfo (see page 15574)

84.1.9.4.24 TRemotableTypeRegistry.VariantToInfo Method

Returns the TypeInfo pointer for the type of a specified Variant's data.

Pascal

```
function VariantToInfo(const V: Variant; TryAllSchema: Boolean): PTypeInfo;
```

C++

```
__fastcall PTypeInfo VariantToInfo(const Variant V, Boolean TryAllSchema);
```

Description

VariantToInfo returns the TypeInfo pointer of the scalar type that represents the value of a Variant. If the Variant's value is not a scalar type that can be marshaled without using a registered object or type, VariantToInfo raises an ETypeRegistryException (see page 15500) exception.

V is the Variant for which you want the corresponding TypeInfo pointer.

TryAllSchema specifies whether VariantToInfo should check for corresponding scalar type definitions in the XML schema associated with this application, or whether it should check all the possible sets of XML schema types that are defined in the xmlschema unit.

See Also

VarType (see page 23667)

ETypeRegistryException (see page 15500)

84.1.9.4.25 TRemotableTypeRegistry.XSDToTypeInfo Method

Returns the type information pointer associated with a specified type name.

Pascal

```
function XSDToTypeInfo(const URI: WideString; const TypeName: WideString): PTypeInfo;
```

C++

```
__fastcall PTypeInfo XSDToTypeInfo(const BSTR URI, const BSTR TypeName);
```

Description

XSDToTypeInfo looks up a namespace URI and typename, and returns the type information pointer that describes the corresponding Object Pascal type or class. Unlike the URIToInfo (see page 15578) method, XSDToTypeInfo can look up built-in type correspondences that do not need to be registered.

URI is the namespace URI for the type.

TypeName is the type name to look up.

See Also

RegisterXSClass (see page 15573)

URIToInfo (see page 15578)

RegisterXSInfo (see page 15574)

84.1.9.5 TRemotableTypeRegistry Properties

The properties of the TRemotableTypeRegistry class are listed here.

Public Properties

	Name	Description
	AutoRegisterNativeTypes (see page 15579)	Specifies whether types that are used in an invokable interface or by a remotable class are automatically registered.

84.1.9.5.1 TRemotableTypeRegistry.AutoRegisterNativeTypes Property

Specifies whether types that are used in an invokable interface or by a remotable class are automatically registered.

Pascal

```
property AutoRegisterNativeTypes: Boolean;
```

C++

```
__property Boolean AutoRegisterNativeTypes;
```

Description

When AutoRegisterNativeTypes is true (the default), type definitions are automatically registered when the invokable interface or remotable class that uses them is registered. That is, when you register an invokable interface with the invocation registry or when you register a remotable class with the remotable type registry, the registry checks whether it makes use of types that are

not yet registered, and if so, registers them as well. Note that you can still provide your own registration (for example, to control the external name or namespace) by registering the type before the call that registers the interface or remotable class.

See Also

RegisterXSInfo (see page 15574)

RegisterXSClass (see page 15573)

84.1.10 TRemotableXS Class

TRemotableXS is the base class for classes that represent a non-native scalar type on an invokable interface in a Web Service application.

Class Hierarchy

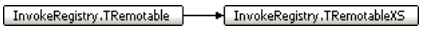

Pascal

```
TRemotableXS = class(TRemotable);
```

C++

```
class TRemotableXS : public TRemotable;
```

File

InvokeRegistry

Description

Use TRemotableXS as a base class when defining a class to represent scalar values from a WSDL document that have no corresponding Object Pascal type. When creating a TRemotableXS descendant, override the XSToNative (see page 15583) and NativeToXS (see page 15581) methods to indicate how the value of the class can be converted to and from the string representation that is used to encode the value for transport.

Both the client application and server application must register a TRemotableXS descendant with the remotable class registry before they can use it. To register a remotable class, use the RegisterXSClass method of the object that the global RemTypeRegistry (see page 15621) function returns.

On server applications, instances of TRemotableXS descendants that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TRemotableXS descendant is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable (see page 15556) descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TRemotableXS instances that it passes as input parameters, and for freeing any TRemotableXS instances it creates or that are returned as output parameters or method results.

See Also

RemTypeRegistry (see page 15621)

TRemotableTypeRegistry (see page 15562)

IInvokable (see page 22398)

Using Remotable Objects

84.1.10.1 TRemotableXS Members

The following tables list the members exposed by TRemotableXS.

Protected Fields

	Name	Description
	FSerializationOptions (see page 15559)	This is FSerializationOptions, a member of class TRemotable.

Public Methods

	Name	Description
V	Create (see page 15558)	Creates and initializes a TRemotable (see page 15556) object.
V	Destroy (see page 15558)	Frees the memory associated with the TRemotable (see page 15556) object.
A	NativeToXS (see page 15581)	Returns a string that represents the value of a TRemotableXS (see page 15580) instance.
V	ObjectToSOAP (see page 15582)	Converts this remotable object to its SOAP representation.
V	SOAPToObject (see page 15582)	Initializes this TRemotableXS (see page 15580) instance to represent the value of its SOAP representation.
A	XSToNative (see page 15583)	Assigns the value of a string to the TRemotableXS (see page 15580) instance.

Public Properties

	Name	Description
	DataContext (see page 15561)	Specifies the object that controls the lifetime of a TRemotable (see page 15556) instance.
R	SerializationOptions (see page 15561)	Indicates how the value of the TRemotable (see page 15556) descendant encodes its value.

84.1.10.2 TRemotableXS Methods

The methods of the TRemotableXS class are listed here.

Public Methods

	Name	Description
A	NativeToXS (see page 15581)	Returns a string that represents the value of a TRemotableXS (see page 15580) instance.
V	ObjectToSOAP (see page 15582)	Converts this remotable object to its SOAP representation.
V	SOAPToObject (see page 15582)	Initializes this TRemotableXS (see page 15580) instance to represent the value of its SOAP representation.
A	XSToNative (see page 15583)	Assigns the value of a string to the TRemotableXS (see page 15580) instance.

84.1.10.2.1 TRemotableXS.NativeToXS Method

Returns a string that represents the value of a TRemotableXS (see page 15580) instance.

Pascal

```
function NativeToXS: WideString; virtual; abstract;
```

C++

```
virtual __fastcall BSTR NativeToXS() = 0;
```

Description

Override NativeToXS when implementing a TRemotableXS (see page 15580) descendant that represents a non-native scalar type. NativeToXS returns a string that represents the value of the TRemotableXS (see page 15580) instance. This string is used when marshaling the value of the TRemotableXS (see page 15580) descendant.

Classes (see page 2685) that implement NativeToXS must also override the XSToNative (see page 15583) method so that it performs the converse transformation.

See Also

XSToNative (see page 15583)

Using Remotable Objects

84.1.10.2.2 TRemotableXS.ObjectToSOAP Method

Converts this remotable object to its SOAP representation.

Pascal

```
function ObjectToSOAP(RootNode: IXMLNode; ParentNode: IXMLNode; const ObjConverter:
IObjConverter; const Name: InvString; const URI: InvString; ObjConvOpts:
TObjectConvertOptions; out RefID: InvString): IXMLNode; override;
```

C++

```
virtual __fastcall IXMLNode ObjectToSOAP(IXMLNode RootNode, IXMLNode ParentNode, const
IObjConverter ObjConverter, const InvString Name, const InvString URI, TObjectConvertOptions
ObjConvOpts, InvString RefID);
```

Description

Override ObjectToSOAP to customize the way your application converts a TRemotableXS (see page 15580) descendant to its SOAP representation. In TRemotableXS (see page 15580), ObjectToSOAP uses the default conversion, replacing the text of the node with the value that the NativeToXS (see page 15581) method supplies.

RootNode is an ancestor of the generated node to which information about the generated node can be added as attributes.

ParentNode is the node in the SOAP representation that should act as a parent to the node that ObjectToSOAP generates.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

Name is the name of the node that is generated.

URI is the URI for the namespace in which Name is defined.

ObjConvOpts are flags that customize the way the node is serialized.

RefID returns a reference ID for the node. This is used if the node is multi-referenced.

ObjectToSOAP returns the XML for the newly generated node.

See Also

SOAPToObject (see page 15582)

NativeToXS (see page 15581)

84.1.10.2.3 TRemotableXS.SOAPToObject Method

Initializes this TRemotableXS (see page 15580) instance to represent the value of its SOAP representation.

Pascal

```
procedure SOAPToObject(const RootNode: IXMLNode; const Node: IXMLNode; const ObjConverter:
IObjConverter); override;
```

C++

```
virtual __fastcall SOAPToObject(const IXMLNode RootNode, const IXMLNode Node, const
```

```
IObjConverter ObjConverter);
```

Description

SOAPToObject initializes the published properties of this object instance so that they reflect the header's SOAP representation. Override this method to customize the way your application creates a scalar value from its XML representation.

RootNode is an ancestor node of the node for this object that can contain information about the remotable type's value.

Node is the node in the SOAP envelope that represents the remotable type's value.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

In TRemotableXS (see page 15580), SOAPToObject calls the XSToNative (see page 15583) method, passing in the text of the specified node.

See Also

ObjectToSOAP (see page 15582)

XSToNative (see page 15583)

84.1.10.2.4 TRemotableXS.XSToNative Method

Assigns the value of a string to the TRemotableXS (see page 15580) instance.

Pascal

```
procedure XSToNative(Data: WideString); virtual; abstract;
```

C++

```
virtual __fastcall XSToNative(BSTR Data) = 0;
```

Description

Override XSToNative when implementing a TRemotableXS (see page 15580) descendant that represents a non-native scalar type. NativeToXS (see page 15581) interprets a string that represents the value of the TRemotableXS (see page 15580) instance and uses it to make any necessary changes so that the TRemotableXS (see page 15580) object reflects that value. It reverses the transformation that the NativeToXS (see page 15581) method performs.

Data is a string that represents the value of the TRemotableXS (see page 15580) instance.

See Also

NativeToXS (see page 15581)

Using Remotable Objects

84.1.11 TSOAPAttachment Class

TSOAPAttachment represents an attachment that is included with SOAP messages that represent method calls in a Web Service application.

Class Hierarchy

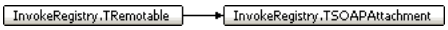

Pascal

```
TSOAPAttachment = class(TRemotable);
```

C++

```
class TSOAPAttachment : public TRemotable;
```

File

InvokeRegistry

Description

Use TSOAPAttachment to receive or send an attachment as a parameter or return value of a method on an invokable interface. When a SOAP method call requires an attachment, it is marshaled as a MIME multipart form, where any attachments are included as additional parts.

Warning: Some Web Services (such as .Net) use DIME multipart forms rather than MIME multipart forms for attachments. TSOAPAttachment will not work with such services.

Note: Only use TSOAPAttachment and its descendants as the type for parameters or the return value of a method of an invokable interface. TSOAPAttachment can't be used as part of a more complex type, such as the type of a published property on a remotable object.

When sending an attachment, use the SourceString (see page 15596) property or the SetSourceFile (see page 15590) or SetSourceStream (see page 15591) method to supply the attachment data. Use the ContentType (see page 15593) property to indicate the type (encoding) of the attachment data.

When receiving an attachment, the attachment is automatically stored in a temporary file, whose name is given by the CacheFile (see page 15592) property. You can copy the contents of this temporary file using the SaveToFile (see page 15589) or SaveToStream (see page 15589) method. To determine the format of the data, read the ContentType (see page 15593) property. You can read any of the other headers for the part of the multipart form that contained the attachment using the Headers (see page 15594) property. By default, when the TSOAPAttachment object is freed, it deletes the temporary file that contains the attachment data that was received. You can use the CacheFilePersist (see page 15593) property to prevent this file from being deleted.

Note: If you use TSOAPAttachment as a base class for a customized attachment class, it is marshaled as a TSOAPAttachment value. That is, the application that receives the attachment sees it only as a TSOAPAttachment instance, not as an instance of the descendant class.

The application must register TSOAPAttachment with the remotable class registry before it can use this class. To register TSOAPAttachment or one of its descendants, use the RegisterXSClass method of the object that the global RemTypeRegistry (see page 15621) function returns. When you import a WSDL document, the call to register TSOAPAttachment is automatically added if any of the interfaces use attachments.

The caller of an invokable interface (client) is responsible for creating any TSOAPAttachment instances that it passes as input parameters, and for freeing any TSOAPAttachment instances it creates or that are returned as output parameters or method results.

In Web Service applications (servers), any TSOAPAttachment instances that represent parameters or return values are added to the data context, which frees them as necessary.

See Also

RemTypeRegistry (see page 15621)

TRemotableTypeRegistry (see page 15562)

IInvokable (see page 22398)

Using Remotable Objects

84.1.11.1 TSOAPAttachment Members

The following tables list the members exposed by TSOAPAttachment.

Protected Fields

	Name	Description
	FSerializationOptions (see page 15559)	This is FSerializationOptions, a member of class TRemotable.

Protected Methods

	Name	Description
V	GetSourceStream (see page 15587)	This is GetSourceStream, a member of class TSOAPAttachment.
V	InternalSetCacheFile (see page 15588)	This is InternalSetCacheFile, a member of class TSOAPAttachment.
	InternalSetSourceStream (see page 15588)	This is InternalSetSourceStream, a member of class TSOAPAttachment.
V	SetOwnership (see page 15590)	This is SetOwnership, a member of class TSOAPAttachment.
V	SetSourceString (see page 15591)	This is SetSourceString, a member of class TSOAPAttachment.

Public Methods

	Name	Description
V	Create (see page 15585)	Creates and initializes a TSOAPAttachment (see page 15583) object.
V	Destroy (see page 15586)	Frees the memory associated with the TSOAPAttachment (see page 15583) object.
	Init (see page 15587)	Initializes the properties of a TSOAPAttachment (see page 15583) instance.
V	ObjectToSOAP (see page 15588)	Converts this attachment object to its SOAP representation.
	SaveToFile (see page 15589)	Saves the contents of the attachment to a specified file.
	SaveToStream (see page 15589)	Saves the contents of the attachment to a specified file.
V	SetSourceFile (see page 15590)	Specifies a file from which TSOAPAttachment (see page 15583) can read the content of the attachment.
V	SetSourceStream (see page 15591)	Specifies a stream from which TSOAPAttachment (see page 15583) can read the content of the attachment.
V	SOAPToObject (see page 15591)	Initializes this TSOAPAttachment (see page 15583) instance to represent the value of its SOAP representation.

Public Properties

	Name	Description
R	CacheFile (see page 15592)	Identifies the temporary file where the attachment is saved.
	CacheFilePersist (see page 15593)	Indicates whether the temporary file that contains the attachment data is deleted when TSOAPAttachment (see page 15583) is freed.
	ContentType (see page 15593)	Specifies the MIME type of the attachment data.
	DataContext (see page 15561)	Specifies the object that controls the lifetime of a TRemotable (see page 15556) instance.
	Encoding (see page 15594)	Indicates the encoding (character set) of the body of the attachment.
R	Headers (see page 15594)	Indicates the values of the headers received with the attachment.
	Ownership (see page 15595)	Indicates whether the TSOAPAttachment (see page 15583) object should free its source stream.
R	SerializationOptions (see page 15561)	Indicates how the value of the TRemotable (see page 15556) descendant encodes its value.
R	SourceStream (see page 15595)	Indicates a stream from which to read the content of an attachment that the application sends.
	SourceString (see page 15596)	Specifies a string that is sent as the value of the attachment.

84.1.11.2 TSOAPAttachment.Create Constructor

Creates and initializes a TSOAPAttachment (see page 15583) object.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TSOAPAttachment();
```

Description

Call Create to instantiate a TSOAPAttachment (see page 15583) object for an attachment you wish to send. When an application receives a SOAP attachment, an instance of TSOAPAttachment (see page 15583) is created automatically to represent that attachment.

84.1.11.3 TSOAPAttachment.Destroy Destructor

Frees the memory associated with the TSOAPAttachment (see page 15583) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSOAPAttachment();
```

Description

Do not call Destroy to free a TSOAPAttachment (see page 15583) object. Instead, use the Free method, which checks for a nil reference before calling Destroy.

On server applications, there is typically no need to explicitly free a TSOAPAttachment (see page 15583) instance (for example, one created as a return value or output parameter). By default, when a TSOAPAttachment (see page 15583) descendant is created in a method that was called remotely using an invokable interface, it is added to a data context (the value of the DataContext (see page 15561) property). As long as the remotable object belongs to the data context, the data context handles freeing the object. Similarly, TSOAPAttachment (see page 15583) instances that are passed in as a parameter belong to the data context.

On client applications, TSOAPAttachment (see page 15583) instances created and passed to an invokable interface or returned as a parameter or result of an invokable interface must be freed by the calling application.

See Also

DataContext (see page 15561)

Free (see page 22300)

84.1.11.4 TSOAPAttachment Methods

The methods of the TSOAPAttachment class are listed here.

Protected Methods

	Name	Description
	GetSourceStream (see page 15587)	This is GetSourceStream, a member of class TSOAPAttachment.
	InternalSetCacheFile (see page 15588)	This is InternalSetCacheFile, a member of class TSOAPAttachment.
	InternalSetSourceStream (see page 15588)	This is InternalSetSourceStream, a member of class TSOAPAttachment.
	SetOwnership (see page 15590)	This is SetOwnership, a member of class TSOAPAttachment.
	SetSourceString (see page 15591)	This is SetSourceString, a member of class TSOAPAttachment.

Public Methods

	Name	Description
	Init (see page 15587)	Initializes the properties of a TSOAPAttachment (see page 15583) instance.
V	ObjectToSOAP (see page 15588)	Converts this attachment object to its SOAP representation.
	SaveToFile (see page 15589)	Saves the contents of the attachment to a specified file.
	SaveToStream (see page 15589)	Saves the contents of the attachment to a specified file.
V	SetSourceFile (see page 15590)	Specifies a file from which TSOAPAttachment (see page 15583) can read the content of the attachment.
V	SetSourceStream (see page 15591)	Specifies a stream from which TSOAPAttachment (see page 15583) can read the content of the attachment.
V	SOAPToObject (see page 15591)	Initializes this TSOAPAttachment (see page 15583) instance to represent the value of its SOAP representation.

84.1.11.4.1 TSOAPAttachment.GetSourceStream Method

Pascal

```
function GetSourceStream: TStream; virtual;
```

C++

```
virtual __fastcall TStream GetSourceStream();
```

Description

This is GetSourceStream, a member of class TSOAPAttachment.

84.1.11.4.2 TSOAPAttachment.Init Method

Initializes the properties of a TSOAPAttachment (see page 15583) instance.

Pascal

```
procedure Init(ATempLocation: string; AHeaders: TStrings; AContentType: String; AEncoding:
WideString);
```

C++

```
__fastcall Init(AnsiString ATempLocation, TStrings AHeaders, AnsiString AContentType, BSTR
AEncoding);
```

Description

Init sets the values of the CacheFile (see page 15592), Headers (see page 15594), ContentType (see page 15593), and Encoding (see page 15594) properties to the values supplied as parameters. It is used internally to initialize a TSOAPAttachment (see page 15583) instance to describe an attachment that the application received.

ATempLocation is the name of the temporary file where the attachment has been stored. It becomes the value of the CacheFile (see page 15592) property.

AHeaders lists the headers from the portion of the multipart form that represented this attachment. It becomes the value of the Headers (see page 15594) property.

AContentType is the content-type of the attachment. It becomes the value of the ContentType (see page 15593) property.

AEncoding is the character set encoding used for the attachment. It becomes the value of the Encoding (see page 15594) property.

See Also

CacheFile (see page 15592)

Headers (see page 15594)

ContentType (see page 15593)

Encoding (see page 15594)

84.1.11.4.3 TSOAPAttachment.InternalSetCacheFile Method

Pascal

```
procedure InternalSetCacheFile(const Value: string); virtual;
```

C++

```
virtual __fastcall InternalSetCacheFile(const AnsiString Value);
```

Description

This is InternalSetCacheFile, a member of class TSOAPAttachment.

84.1.11.4.4 TSOAPAttachment.InternalSetSourceStream Method

Pascal

```
procedure InternalSetSourceStream(const Value: TStream; const Ownership: TStreamOwnership =
soReference);
```

C++

```
__fastcall InternalSetSourceStream(const TStream Value, const TStreamOwnership Ownership =
soReference);
```

Description

This is InternalSetSourceStream, a member of class TSOAPAttachment.

84.1.11.4.5 TSOAPAttachment.ObjectToSOAP Method

Converts this attachment object to its SOAP representation.

Pascal

```
function ObjectToSOAP(RootNode: IXMLNode; ParentNode: IXMLNode; const ObjConverter:
IObjConverter; const Name: InvString; const URI: InvString; ObjConvOpts:
TObjectConvertOptions; out RefID: InvString): IXMLNode; override;
```

C++

```
virtual __fastcall IXMLNode ObjectToSOAP(IXMLNode RootNode, IXMLNode ParentNode, const
IObjConverter ObjConverter, const InvString Name, const InvString URI, TObjectConvertOptions
ObjConvOpts, InvString RefID);
```

Description

ObjectToSOAP customizes the way an application converts a TSOAPAttachment (see page 15583) object to its SOAP representation. In TSOAPAttachment (see page 15583), ObjectToSOAP calls the inherited method, adding ocoDontSerializeProps and ocoDontPutTypeAttr to the ObjConvOpts flags so that the converter ignores the published properties and does not add type information to the generated node. It then generates a unique identifier for the attachment and calls the converter's AddAttachment method to properly generate the SOAP representation of the attachment.

RootNode is an ancestor of the generated node to which information about the node can be added as attributes.

ParentNode is the node in the SOAP representation that should act as a parent to the node that ObjectToSOAP generates.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

Name is the name of the node that is generated for the attachment.

URI is the URI for the namespace in which Name is defined.

ObjConvOpts are flags that customize the way the node is serialized.

RefID returns a reference ID for the node. This is used if the node is multi-referenced.

ObjectToSOAP returns the XML for the newly generated node.

See Also

ObjInstanceToSOAP (see page 15612)

AddAttachment (see page 15611)

SOAPToObject (see page 15591)

84.1.11.4.6 TSOAPAttachment.SaveToFile Method

Saves the contents of the attachment to a specified file.

Pascal

```
procedure SaveToFile(AFileName: string);
```

C++

```
__fastcall SaveToFile(AnsiString AFileName);
```

Description

Call SaveToFile to save the contents of the attachment to a specified file. This content can come from

The temporary file that the CacheFile (see page 15592) property indicates.

The stream that the SourceStream (see page 15595) property specifies.

The file that the SetSourceFile (see page 15590) method supplies.

The string that the SourceString (see page 15596) property specifies.

AFileName is the name of the file to which SaveToFile writes the content of the attachment.

See Also

CacheFile (see page 15592)

SourceStream (see page 15595)

SourceString (see page 15596)

SaveToStream (see page 15589)

84.1.11.4.7 TSOAPAttachment.SaveToStream Method

Saves the contents of the attachment to a specified file.

Pascal

```
procedure SaveToStream(AStream: TStream);
```

C++

```
__fastcall SaveToStream(TStream AStream);
```

Description

Call SaveToStream to write the contents of the attachment to a specified stream. This content can come from

The temporary file that the CacheFile (see page 15592) property indicates.

The stream that the SourceStream (see page 15595) property specifies.

The file that the SetSourceFile (see page 15590) method supplies.

The string that the SourceString (see page 15596) property specifies.

AStream is the stream to which SaveToStream writes the content of the attachment.

See Also

CacheFile (see page 15592)

SourceStream (see page 15595)

SourceString (see page 15596)

SaveToStream

SetSourceFile (see page 15590)

84.1.11.4.8 TSOAPAttachment.SetOwnership Method

Pascal

```
procedure SetOwnership(const Value: TStreamOwnership); virtual;
```

C++

```
virtual __fastcall SetOwnership(const TStreamOwnership Value);
```

Description

This is SetOwnership, a member of class TSOAPAttachment.

84.1.11.4.9 TSOAPAttachment.SetSourceFile Method

Specifies a file from which TSOAPAttachment (see page 15583) can read the content of the attachment.

Pascal

```
procedure SetSourceFile(const Value: string); virtual;
```

C++

```
virtual __fastcall SetSourceFile(const AnsiString Value);
```

Description

Call SetSourceFile to supply the content of an attachment that the application sends from a file. SetSourceFile reads the content from the specified file, clears the SourceStream (see page 15595) and SourceString (see page 15596) properties, and sets the ContentType (see page 15593) property to 'Application/binary'. If another MIME type string better describes the data, reset the ContentType (see page 15593) property after calling SetSourceFile.

Value is the fully-qualified file name for the file whose content should be sent as an attachment.

See Also

ContentType (see page 15593)

SetSourceStream (see page 15591)

SourceString (see page 15596)

84.1.11.4.10 TSOAPAttachment.SetSourceStream Method

Specifies a stream from which TSOAPAttachment (see page 15583) can read the content of the attachment.

Pascal

```
procedure SetSourceStream(const Value: TStream; const Ownership: TStreamOwnership =
soReference); virtual;
```

C++

```
virtual __fastcall SetSourceStream(const TStream Value, const TStreamOwnership Ownership =
soReference);
```

Description

Call SetSourceStream to supply the content of an attachment that the application sends from a stream object. SetSourceStream assigns the specified stream as the value of the SourceStream (see page 15595) property, clears the FileName and SourceString (see page 15596) properties, and sets the ContentType (see page 15593) property to 'Application/binary'. If another MIME type string better describes the data, reset the ContentType (see page 15593) property after calling SetSourceStream.

Value is a stream from which TSOAPAttachment (see page 15583) can read the content of the attachment.

Ownership (see page 15595) indicates whether the TSOAPAttachment (see page 15583) object should free the stream (Value) when it no longer needs it. It becomes the value of the Ownership (see page 15595) property.

See Also

SourceStream (see page 15595)

ContentType (see page 15593)

Ownership (see page 15595)

SourceString (see page 15596)

SetSourceFile (see page 15590)

84.1.11.4.11 TSOAPAttachment.SetSourceString Method

Pascal

```
procedure SetSourceString(const Value: string); virtual;
```

C++

```
virtual __fastcall SetSourceString(const AnsiString Value);
```

Description

This is SetSourceString, a member of class TSOAPAttachment.

84.1.11.4.12 TSOAPAttachment.SOAPToObject Method

Initializes this TSOAPAttachment (see page 15583) instance to represent the value of its SOAP representation.

Pascal

```
procedure SOAPToObject(const RootNode: IXMLNode; const Node: IXMLNode; const ObjConverter:
IObjConverter); override;
```

C++

```
virtual __fastcall SOAPToObject(const IXMLNode RootNode, const IXMLNode Node, const
IObjConverter ObjConverter);
```

Description

SOAPToObject initializes the published properties of this object instance so that they reflect the header's SOAP representation. In TSOAPAttachment (see page 15583), this method calls the converter's FindAttachment method to locate information about the attachment, and then calls the Init (see page 15587) method to initialize this object from the located information.

RootNode is an ancestor node of the attachment node that can contain information about the attachment node.

Node is the node in the SOAP envelope that represents a reference to the attachment.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

In TSOAPAttachment (see page 15583), SOAPToObject simply calls the InitObjectFromSOAP method of the converter supplied by the ObjConverter parameter.

See Also

ObjectToSOAP (see page 15588)

FindAttachment (see page 15612)

Init (see page 15587)

84.1.11.5 TSOAPAttachment Properties

The properties of the TSOAPAttachment class are listed here.

Public Properties

	Name	Description
R	CacheFile (see page 15592)	Identifies the temporary file where the attachment is saved.
	CacheFilePersist (see page 15593)	Indicates whether the temporary file that contains the attachment data is deleted when TSOAPAttachment (see page 15583) is freed.
	ContentType (see page 15593)	Specifies the MIME type of the attachment data.
	Encoding (see page 15594)	Indicates the encoding (character set) of the body of the attachment.
R	Headers (see page 15594)	Indicates the values of the headers received with the attachment.
	Ownership (see page 15595)	Indicates whether the TSOAPAttachment (see page 15583) object should free its source stream.
R	SourceStream (see page 15595)	Indicates a stream from which to read the content of an attachment that the application sends.
	SourceString (see page 15596)	Specifies a string that is sent as the value of the attachment.

84.1.11.5.1 TSOAPAttachment.CacheFile Property

Identifies the temporary file where the attachment is saved.

Pascal

```
property CacheFile: string;
```

C++

```
__property AnsiString CacheFile;
```

Description

CacheFile is set automatically when the application receives an attachment along with a SOAP message. This is the name of the

temporary file where the application stores the contents of the attachment.

To save this temporary file to another file of your choosing, use the SaveToFile (see page 15589) method. You can also save its contents to a stream, using the SaveToStream (see page 15589) method.

The ContentType (see page 15593) property indicates the format of the data in this temporary file.

By default, when the TSOAPAttachment (see page 15583) object is freed, it deletes the temporary file that CacheFile indicates. Use the CacheFilePersist (see page 15593) property to prevent this file from being deleted.

See Also

CacheFilePersist (see page 15593)

ContentType (see page 15593)

Data

SaveToFile (see page 15589)

SaveToStream (see page 15589)

84.1.11.5.2 TSOAPAttachment.CacheFilePersist Property

Indicates whether the temporary file that contains the attachment data is deleted when TSOAPAttachment (see page 15583) is freed.

Pascal

```
property CacheFilePersist: Boolean;
```

C++

```
__property Boolean CacheFilePersist;
```

Description

When CacheFilePersist is false (the default), the temporary file that the CacheFile (see page 15592) property identifies is automatically deleted when the TSOAPAttachment (see page 15583) instance is freed. You can save the information from the cache file using the SaveToFile (see page 15589) or SaveToStream (see page 15589) method before freeing the TSOAPAttachment (see page 15583) instance.

When CacheFilePersist is true, the temporary file is not deleted when the TSOAPAttachment (see page 15583) instance is freed.

See Also

CacheFile (see page 15592)

SaveToFile (see page 15589)

SaveToStream (see page 15589)

84.1.11.5.3 TSOAPAttachment.ContentType Property

Specifies the MIME type of the attachment data.

Pascal

```
property ContentType: string;
```

C++

```
__property AnsiString ContentType;
```

Description

Use ContentType to determine the MIME type (format) used for the data in the attachment.

When receiving an attachment, the ContentType property is automatically set to reflect the value of the content-type header of the part of the multipart form that holds the attachment. You can read this property to determine how to interpret the contents of the file that the CacheFile (see page 15592) property indicates.

When sending an attachment, set ContentType to indicate the MIME type for the data you are sending. This property should not be set until after setting the SourceString (see page 15596) property calling the SetSourceFile (see page 15590) or SetSourceStream (see page 15591) method. That is because calling SetSourceStream (see page 15591) or SetSourceFile (see page 15590) automatically sets ContentType to 'Application/binary', and setting SourceString (see page 15596) automatically sets ContentType to 'Text/simple'. There is no need to reset ContentType if these values are correct.

See Also

CacheFile (see page 15592)

SourceString (see page 15596)

SetSourceFile (see page 15590)

SetSourceStream (see page 15591)

Data

84.1.11.5.4 TSOAPAttachment.Encoding Property

Indicates the encoding (character set) of the body of the attachment.

Pascal

```
property Encoding: WideString;
```

C++

```
__property BSTR Encoding;
```

Description

Use Encoding to look up or set the encoding type of the attachment's content. When sending an attachment, set this property to indicate the encoding that was used for the source you supply.

If you supply the attachment data as a string by setting the SourceString (see page 15596) property, TSOAPAttachment (see page 15583) assumes an encoding of UTF-8 if you do not set the Encoding property. If you supply the attachment data by calling SetSourceFile (see page 15590) or SetSourceStream (see page 15591), you must set the Encoding property, as TSOAPAttachment (see page 15583) does not have a meaningful default encoding.

Note: Encoding indicates the encoding of the data that the SOAP attachment contains. The application does not automatically use this information to encode or decode that data.

See Also

ContentType (see page 15593)

84.1.11.5.5 TSOAPAttachment.Headers Property

Indicates the values of the headers received with the attachment.

Pascal

```
property Headers: TStrings;
```

C++

```
__property TStrings Headers;
```

Description

Read Headers to determine the headers that appeared in the part of the multipart form that contained the attachment. Headers is only intended for attachments that the application receives. When sending attachments, the application automatically generates the headers that accompany the attachment.

Headers is a string list where each string has the form HeaderName=Value. HeaderName is the name of the header, such as Content-Id, Content-Location, Content-Size, and so on. Value is the string that is the header value. Note that if you are only interested in the content type, you can use the ContentType (see page 15593) property instead.

Note: Although the MIME specification states that the Content-Length header is optional, TSoapAttachment requires this header and raises an exception if it is not present.

See Also

ContentType (see page 15593)

CacheFile (see page 15592)

84.1.11.5.6 TSOAPAttachment.Ownership Property

Indicates whether the TSOAPAttachment (see page 15583) object should free its source stream.

Pascal

```
property Ownership: TStreamOwnership;
```

C++

```
__property TStreamOwnership Ownership;
```

Description

Ownership indicates whether TSOAPAttachment (see page 15583) needs to free the stream that supplied the content of the attachment when the application supplied this data using the SetSourceStream (see page 15591) method.

See Also

SourceStream (see page 15595)

SetSourceStream (see page 15591)

84.1.11.5.7 TSOAPAttachment.SourceStream Property

Indicates a stream from which to read the content of an attachment that the application sends.

Pascal

```
property SourceStream: TStream;
```

C++

```
__property TStream SourceStream;
```

Description

Read SourceStream to access the content of an attachment that the application is sending. When SourceStream is not nil (Delphi) or NULL (C++), the content of the attachment is read from this stream and the SourceString (see page 15596) property is not set. Depending on the value of the Ownership (see page 15595) property, SourceStream can be owned by the TSOAPAttachment (see page 15583) object (in which case TSOAPAttachment (see page 15583) frees the stream object

when it is no longer needed) or can exist as an independent object that TSOAPAttachment (see page 15583) does not free.

SourceStream is a read-only property. To specify a stream from which TSOAPAttachment (see page 15583) reads its data, use the SetSourceStream (see page 15591) method.

See Also

SourceString (see page 15596)

SetSourceStream (see page 15591)

Ownership (see page 15595)

ContentType (see page 15593)

84.1.11.5.8 TSOAPAttachment.SourceString Property

Specifies a string that is sent as the value of the attachment.

Pascal

```
property SourceString: string;
```

C++

```
__property AnsiString SourceString;
```

Description

Use SourceString to supply the content of an attachment as a string.

Setting the SourceString property causes TSOAPAttachment (see page 15583) to clear the SourceStream (see page 15595) property and to set the ContentType (see page 15593) property to 'Text/simple'. If another MIME type string better describes the data, reset the ContentType (see page 15593) property after setting SourceString.

See Also

Data

SourceStream (see page 15595)

SetSourceStream (see page 15591)

ContentType (see page 15593)

SetSourceFile (see page 15590)

84.1.12 TSOAPHeader Class

TSOAPHeader represents a header on the SOAP envelope that contains an encoded message in a Web Service application.

Class Hierarchy

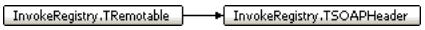

Pascal

```
TSOAPHeader = class(TRemotable);
```

C++

```
class TSOAPHeader : public TRemotable;
```

File

InvokeRegistry

Description

TSOAPHeader is the base class for classes that represent SOAP headers. Each type of header has an associated TSOAPHeader descendant, which is registered with the remotable type registry. Register a TSOAPHeader descendant like any other remotable type, using the name of the header as the type name. If you import a WSDL document, the code to define and register the TSOAPHeader descendant is generated automtically.

You can also register the TSOAPHeader descendant with the invocation registry. The invocation registry's RegisterHeaderClass method associates the header class and the invokable interfaces with which it should be included. Registering the header with the invocation registry serves two purposes:

When a server publishes WSDL information about the Web Service, information about the header type is included in the resulting WSDL document.

On both servers and clients, the invocation registry associates the header class with a header name and a namespace in which that name is defined. This is important if the namespace for the header type is different than the namespace for the invokable interface that uses the header.

A SOAP header is a special remotable type that represents a node in the header section of a SOAP envelope. As with any other remotable type, the published properties of the TSOAPHeader descendant correspond to the child nodes and attributes of the corresponding header node. TSOAPHeader introduces two properties for common attributes of SOAP headers: MustUnderstand (see page 15600) and Actor (see page 15599). Typically, SOAP headers represent information that is associated with an entire invokable interface (port type) rather than just the method that is the focus of a single message.

When an application receives a SOAP message (that is, when a server receives a request to execute a method on an invokable interface or when a client receives a response after sending such a request), the headers on that message are automatically converted to the corresponding TSOAPHeader descendants that are registered with the remotable type registry. You can access these TSOAPHeader descendants using the ISOAPHeaders (see page 15613) interface, which can be obtained from the implementation class of the invokable interface or from the invokable interface of a remote interfaced object.

When an application sends a SOAP message (either a client sending a request or a server sending a response), supply the headers that should appear on that outgoing message by creating TSOAPHeader descendants and calling the Send method of an ISOAPHeaders (see page 15613) interface.

See Also

ISOAPHeaders (see page 15613)

TRemotableTypeRegistry (see page 15562)

TInvokableClassRegistry (see page 15526)

84.1.12.1 TSOAPHeader Members

The following tables list the members exposed by TSOAPHeader.

Protected Fields

	Name	Description
	FSerializationOptions (see page 15559)	This is FSerializationOptions, a member of class TRemotable.

Public Methods

	Name	Description
	Create (see page 15558)	Creates and initializes a TRemotable (see page 15556) object.

V	Destroy (see page 15558)	Frees the memory associated with the TRemotable (see page 15556) object.
V	ObjectToSOAP (see page 15598)	Converts this TSOAPHeader (see page 15596) object to its SOAP representation.
V	SOAPToObject (see page 15599)	Initializes this TSOAPHeader (see page 15596) instance to represent the value of its SOAP representation.

Public Properties

	Name	Description
	Actor (see page 15599)	Identifies the agent to which the header is directed.
	DataContext (see page 15561)	Specifies the object that controls the lifetime of a TRemotable (see page 15556) instance.
	MustUnderstand (see page 15600)	Indicates whether an application is required to recognize and handle the header on received messages.
R	SerializationOptions (see page 15561)	Indicates how the value of the TRemotable (see page 15556) descendant encodes its value.

84.1.12.2 TSOAPHeader Methods

The methods of the TSOAPHeader class are listed here.

Public Methods

	Name	Description
V	ObjectToSOAP (see page 15598)	Converts this TSOAPHeader (see page 15596) object to its SOAP representation.
V	SOAPToObject (see page 15599)	Initializes this TSOAPHeader (see page 15596) instance to represent the value of its SOAP representation.

84.1.12.2.1 TSOAPHeader.ObjectToSOAP Method

Converts this TSOAPHeader (see page 15596) object to its SOAP representation.

Pascal

```
function ObjectToSOAP(RootNode: IXMLNode; ParentNode: IXMLNode; const ObjConverter:
IObjConverter; const Name: InvString; const URI: InvString; ObjConvOpts:
TObjectConvertOptions; out RefID: InvString): IXMLNode; override;
```

C++

```
virtual __fastcall IXMLNode ObjectToSOAP(IXMLNode RootNode, IXMLNode ParentNode, const
IObjConverter ObjConverter, const InvString Name, const InvString URI, TObjectConvertOptions
ObjConvOpts, InvString RefID);
```

Description

Override ObjectToSOAP to customize the way your application converts a TSOAPHeader (see page 15596) descendant to its SOAP representation. In TSOAPHeader (see page 15596), ObjectToSOAP uses the default conversion (supplied by the ObjConverter parameter), adding a MustUnderstand (see page 15600) attribute whose value reflects the value of the MustUnderstand (see page 15600) property.

RootNode is an ancestor of the generated node to which information about the generated node can be added as attributes.

ParentNode is the node in the SOAP representation that should act as a parent to the node that ObjectToSOAP generates.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

Name is the name of the node that is generated.

URI is the URI for the namespace in which Name is defined.

ObjConvOpts are flags that customize the way the node is serialized.

RefID returns a reference ID for the node. This is used if the node is multi-referenced.

ObjectToSOAP returns the XML for the newly generated node.

See Also

MustUnderstand (see page 15600)

ObjInstanceToSOAP (see page 15612)

SOAPToObject (see page 15599)

84.1.12.2.2 TSOAPHeader.SOAPToObject Method

Initializes this TSOAPHeader (see page 15596) instance to represent the value of its SOAP representation.

Pascal

```
procedure SOAPToObject(const RootNode: IXMLNode; const Node: IXMLNode; const ObjConverter:
IObjConverter); override;
```

C++

```
virtual __fastcall SOAPToObject(const IXMLNode RootNode, const IXMLNode Node, const
IObjConverter ObjConverter);
```

Description

SOAPToObject initializes the published properties of this object instance so that they reflect the header's SOAP representation. Override this method to customize the way your application creates a specific type of header from its XML representation.

RootNode is an ancestor node of the header node that can contain information about the header node.

Node is the node in the SOAP envelope that represents the header.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

See Also

ObjectToSOAP (see page 15598)

InitObjectFromSOAP (see page 15612)

MustUnderstand (see page 15600)

84.1.12.3 TSOAPHeader Properties

The properties of the TSOAPHeader class are listed here.

Public Properties

	Name	Description
	Actor (see page 15599)	Identifies the agent to which the header is directed.
	MustUnderstand (see page 15600)	Indicates whether an application is required to recognize and handle the header on received messages.

84.1.12.3.1 TSOAPHeader.Actor Property

Identifies the agent to which the header is directed.

Pascal

```
property Actor: WideString;
```

C++

```
__property BSTR Actor;
```

Description

Actor is the URI to which the header is directed. Set Actor to indicate the ultimate target of a header you are sending. Read Actor to determine whether a header the application receives is supposed to be handled by the application rather than merely passed on to the next processor.

Note: When the Actor property of a received header identifies this application and the application raises an exception concerning that header (such as a MustUnderstand (see page 15600) fault for a header that it does not know how to handle), the value of Actor should become the value of the FaultActor property on the ERemotableException (see page 15496) object.

See Also

MustUnderstand (see page 15600)

FaultActor (see page 15499)

84.1.12.3.2 TSOAPHeader.MustUnderstand Property

Indicates whether an application is required to recognize and handle the header on received messages.

Pascal

```
property MustUnderstand: Boolean;
```

C++

```
__property Boolean MustUnderstand;
```

Description

When MustUnderstand is false, applications can ignore this header if they do not recognize it and know how to handle it.

When MustUnderstand is true, applications that receive messages with this header but that do not recognize or handle it must throw a MustUnderstand fault.

See Also

Actor (see page 15599)

84.1.13 TSOAPHeaders Class

TSOAPHeaders is a helper object for implementing the ISOAPHeaders (see page 15613) interface.

Class Hierarchy

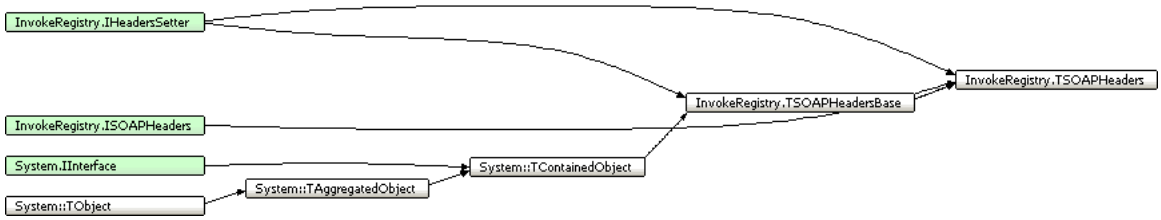

Pascal

```
TSOAPHeaders = class(TSOAPHeadersBase, ISOAPHeaders, IHeadersSetter);
```

C++

```
class TSOAPHeaders : public TSOAPHeadersBase, public ISOAPHeaders, public IHeadersSetter;
```

File

InvokeRegistry

Description

TSOAPHeaders implements the ISOAPHeaders (see page 15613) interface for a class that needs to provide access to the headers of SOAP messages that the application sends and receives Typically, an application does not work directly with instances of TSOAPHeaders. Rather, it uses the ISOAPHeaders (see page 15613) interface that this object implements for another class, such as TRIO or TInvokableClass (see page 15520).

See Also

ISOAPHeaders (see page 15613)

TSOAPHeader (see page 15596)

TRIO (see page 18718)

TInvokableClass (see page 15520)

84.1.13.1 TSOAPHeaders Members

The following tables list the members exposed by TSOAPHeaders.

Protected Fields

	Name	Description
	FHeadersInbound (see page 15607)	This is FHeadersInbound, a member of class TSOAPHeadersBase.
	FHeadersOutBound (see page 15608)	This is FHeadersOutBound, a member of class TSOAPHeadersBase.

Protected Methods

	Name	Description
	_AddRef (see page 22250)	This is _AddRef, a member of class TAggregatedObject.
	_Release (see page 22250)	This is _Release, a member of class TAggregatedObject.
	QueryInterface (see page 22250)	This is QueryInterface, a member of class TAggregatedObject.
V	QueryInterface (see page 22255)	Returns a reference to a specified interface if the object supports that interface.
	SetHeadersInOut (see page 15608)	Sets the lists of incoming and outgoing headers.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22249)	Instantiates an aggregated object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	Get (see page 15614)	This is the overview for the Get method overload.
	Get (see page 15603)	This is the overview for the Get method overload.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetOwnsSentHeaders (see page 15603)	Indicates whether the TSOAPHeaders (see page 15600) object frees headers that are added using the Send (see page 15604) method
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Send (see page 15604)	Adds a header to an outgoing message.
	SendAt (see page 15604)	Retrieves a header from an outgoing message, given its position.
	SendCount (see page 15605)	Indicates the number of headers that have been added to an outgoing message.
	SetHeadersInOut (see page 15610)	This is SetHeadersInOut, a member of class IHeadersSetter.
	SetOwnsSentHeaders (see page 15605)	Sets a flag that indicates who should free outgoing header objects.

Public Properties

	Name	Description
R	Controller (see page 22251)	Provides access to the controlling interface of the aggregate.
	OwnsSentHeaders (see page 15617)	Indicates who is responsible for freeing header objects that are included in outbound messages.

84.1.13.2 TSOAPHeaders Methods

The methods of the TSOAPHeaders class are listed here.

Public Methods

	Name	Description
	Get (see page 15603)	This is the overview for the Get method overload.
	GetOwnsSentHeaders (see page 15603)	Indicates whether the TSOAPHeaders (see page 15600) object frees headers that are added using the Send (see page 15604) method
	Send (see page 15604)	Adds a header to an outgoing message.
	SendAt (see page 15604)	Retrieves a header from an outgoing message, given its position.
	SendCount (see page 15605)	Indicates the number of headers that have been added to an outgoing message.
	SetOwnsSentHeaders (see page 15605)	Sets a flag that indicates who should free outgoing header objects.

84.1.13.2.1 Get Method

This is the overview for the Get method overload.

Overload List

	Name	Description
	TSOAPHeaders.Get (TClass) (see page 15603)	Retrieves a header from an incoming message.
	TSOAPHeaders.Get (TClass, TSOAPHeader) (see page 15603)	Retrieves a header from an incoming message.
	TSOAPHeaders.Get (WideString, WideString) (see page 15603)	Retrieves a header from an incoming message.

84.1.13.2.1.1 TSOAPHeaders.Get Method (TClass)

Retrieves a header from an incoming message.

Pascal

```
procedure Get(Cls: TClass; out Hdr: TSOAPHeader); overload;
function Get(Cls: TClass): TSOAPHeader; overload;
function Get(const Name: WideString; const URI: WideString): TSOAPHeader; overload;
```

C++

```
__fastcall Get(TClass Cls, TSOAPHeader Hdr);
__fastcall TSOAPHeader Get(TClass Cls);
__fastcall TSOAPHeader Get(const BSTR Name, const BSTR URI);
```

Description

Call Get to access the header object that is automatically created to represent one of the headers on an incoming messages. On the server, this is a header on the request message. On the client, this is a header on the response message.

Cls is the class reference that indicates the type of header class you want to retrieve. Each header class corresponds to a type of header, where the type name under which the class is registered corresponds to the name of the header node.

Hdr returns a TSOAPHeader (see page 15596) descendant that represents the header of the incoming message. If this syntax is not used, Get returns the header instance.

Name is the name of the header (the node name for the header node) that you want to retrieve.

URI is the URI for the namespace in which Name is defined.

See Also

Send (see page 15604)

DataContext (see page 15561)

84.1.13.2.2 TSOAPHeaders.GetOwnsSentHeaders Method

Indicates whether the TSOAPHeaders (see page 15600) object frees headers that are added using the Send (see page 15604) method

Pascal

```
function GetOwnsSentHeaders: Boolean;
```

C++

```
__fastcall Boolean GetOwnsSentHeaders();
```

Description

GetOwnsSentHeaders returns true if the TSOAPHeaders (see page 15600) instance frees any header objects added using the Send (see page 15604) method. It returns false if the caller is responsible for freeing these header objects.

See Also

Send (see page 15604)

84.1.13.2.3 TSOAPHeaders.Send Method

Adds a header to an outgoing message.

Pascal

```
procedure Send(const Hdr: TSOAPHeader);
```

C++

```
__fastcall Send(const TSOAPHeader Hdr);
```

Description

Call Send to add a header to an outgoing message. On the client, call Send before executing a method on an invokable interface. On the server, call Send before returning from the method that the client invoked.

Hdr is the header object instance that represents the header you want to add to the outgoing message.

Note: The SetOwnsSentHeaders (see page 15605) method controls whether you must free the object that is supplied as the Hdr parameter after sending the message.

See Also

SendAt (see page 15604)

SendCount (see page 15605)

SetOwnsSentHeaders (see page 15605)

84.1.13.2.4 TSOAPHeaders.SendAt Method

Retrieves a header from an outgoing message, given its position.

Pascal

```
function SendAt(Index: Integer): TSOAPHeader;
```

C++

```
__fastcall TSOAPHeader SendAt(int Index);
```

Description

Call SendAt to retrieve the header object for a header you have already added to an outgoing message using the Send (see page 15604) method.

Index is the position of the header you want to retrieve, where 0 specifies the first header, 1 specifies the second header, and so on. The SendCount (see page 15605) property provides an upper bound for this index.

SendAt returns the header at the specified position.

See Also

Send (see page 15604)

SendCount (see page 15605)

84.1.13.2.5 TSOAPHeaders.SendCount Method

Indicates the number of headers that have been added to an outgoing message.

Pascal

```
function SendCount: Integer;
```

C++

```
__fastcall int SendCount();
```

Description

Call SendCount to determine the number of headers that have been added to an outgoing message. These headers are added by previous calls to the Send (see page 15604) method.

See Also

Send (see page 15604)

SendAt (see page 15604)

84.1.13.2.6 TSOAPHeaders.SetOwnsSentHeaders Method

Sets a flag that indicates who should free outgoing header objects.

Pascal

```
procedure SetOwnsSentHeaders(Flag: Boolean);
```

C++

```
__fastcall SetOwnsSentHeaders(Boolean Flag);
```

Description

Call SetOwnsSentHeaders to specify who should free header objects that are added using the Send (see page 15604) method.

Flag indicates whether this TSOAPHeaders (see page 15600) frees the outgoing headers. If Flag is true, TSOAPHeaders (see page 15600) frees the headers that are added by calling Send (see page 15604). If Flag is false, the caller is responsible for freeing these header objects.

See Also

Send (see page 15604)

84.1.14 TSOAPHeadersBase Class

TSOAPHeadersBase is the base class for classes that maintain lists of SOAP headers.

Class Hierarchy

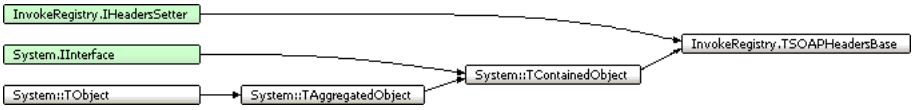

Pascal

```
TSOAPHeadersBase = class(TContainedObject, IHeadersSetter);
```

C++

```
class TSOAPHeadersBase : public TContainedObject, public IHeadersSetter;
```

File

InvokeRegistry

Description

TSOAPHeadersBase provides support for maintaining two lists of TSOAPHeader (see page 15596) descendants. One list is for the headers that an application sends and the other is for the headers that are it receives with an incoming message.

TSOAPHeadersBase is a descendant of TContainedObject. As such, it is designed to be used as a helper object in supporting an interface for working with SOAP headers. TSOAPHeadersBase is intended only as a base class. For working with SOAP headers, use a TSOAPHeadersBase descendant such as TSOAPHeaders (see page 15600).

See Also

TSOAPHeaders (see page 15600)

TSOAPHeader (see page 15596)

84.1.14.1 TSOAPHeadersBase Members

The following tables list the members exposed by TSOAPHeadersBase.

Protected Fields

	Name	Description
	FHeadersInbound (see page 15607)	This is FHeadersInbound, a member of class TSOAPHeadersBase.
	FHeadersOutBound (see page 15608)	This is FHeadersOutBound, a member of class TSOAPHeadersBase.

Protected Methods

	Name	Description
	_AddRef (see page 22250)	This is _AddRef, a member of class TAggregatedObject.
	_Release (see page 22250)	This is _Release, a member of class TAggregatedObject.
	QueryInterface (see page 22250)	This is QueryInterface, a member of class TAggregatedObject.
V	QueryInterface (see page 22255)	Returns a reference to a specified interface if the object supports that interface.
	SetHeadersInOut (see page 15608)	Sets the lists of incoming and outgoing headers.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22249)	Instantiates an aggregated object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Controller (see page 22251)	Provides access to the controlling interface of the aggregate.

84.1.14.2 TSOAPHeadersBase Fields

The fields of the TSOAPHeadersBase class are listed here.

Protected Fields

	Name	Description
	FHeadersInbound (see page 15607)	This is FHeadersInbound, a member of class TSOAPHeadersBase.
	FHeadersOutBound (see page 15608)	This is FHeadersOutBound, a member of class TSOAPHeadersBase.

84.1.14.2.1 TSOAPHeadersBase.FHeadersInbound Field

Pascal

```
FHeadersInbound: THeaderList;
```

C++

```
THeaderList FHeadersInbound;
```

Description

This is FHeadersInbound, a member of class TSOAPHeadersBase.

84.1.14.2.2 TSOAPHeadersBase.FHeadersOutBound Field

Pascal

```
FHeadersOutBound: THeaderList;
```

C++

```
THeaderList FHeadersOutBound;
```

Description

This is FHeadersOutBound, a member of class TSOAPHeadersBase.

84.1.14.3 TSOAPHeadersBase Methods

The methods of the TSOAPHeadersBase class are listed here.

Protected Methods

	Name	Description
	SetHeadersInOut (see page 15608)	Sets the lists of incoming and outgoing headers.

84.1.14.3.1 TSOAPHeadersBase.SetHeadersInOut Method

Sets the lists of incoming and outgoing headers.

Pascal

```
procedure SetHeadersInOut(var InHdrs: THeaderList; var OutHdrs: THeaderList);
```

C++

```
__fastcall SetHeadersInOut(THeaderList InHdrs, THeaderList OutHdrs);
```

Description

Applications can't call the protected SetHeadersInOut method. It is used by descendant classes to initialize the lists of incoming and outgoing header objects.

InHdrs is a list of TSoapHeader descendants that represent the headers received in a SOAP message.

OutHdrs is a list of TSoapHeader descendants that represent the headers sent in a SOAP message.

See Also

TSoapHeader

84.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IHeadersSetter (see page 15609)	This is class InvokeRegistry.IHeadersSetter.
	IObjConverter (see page 15610)	IObjConverter is the interface for converting between a remotable class and its SOAP representation.
	ISOAPHeaders (see page 15613)	ISOAPHeaders is the interface for accessing the headers that accompany a message encoded using SOAP.

84.2.1 IHeadersSetter Interface

Class Hierarchy

InvokeRegistry.IHeadersSetter

Pascal

```
IHeadersSetter = interface;
```

C++

```
__interface IHeadersSetter;
```

File

InvokeRegistry

Description

This is class InvokeRegistry.IHeadersSetter.

84.2.1.1 IHeadersSetter Members

The following tables list the members exposed by IHeadersSetter.

Public Methods

	Name	Description
	SetHeadersInOut (see page 15610)	This is SetHeadersInOut, a member of class IHeadersSetter.

84.2.1.2 IHeadersSetter Methods

The methods of the IHeadersSetter class are listed here.

Public Methods

	Name	Description
	SetHeadersInOut (see page 15610)	This is SetHeadersInOut, a member of class IHeadersSetter.

84.2.1.2.1 IHeadersSetter.SetHeadersInOut Method

Pascal

```
procedure SetHeadersInOut(var InHdrs: THeaderList; var OutHdrs: THeaderList);
```

C++

```
__fastcall SetHeadersInOut(THeaderList InHdrs, THeaderList OutHdrs);
```

Description

This is SetHeadersInOut, a member of class IHeadersSetter.

84.2.2 IObjConverter Interface

IObjConverter is the interface for converting between a remotable class and its SOAP representation.

Class Hierarchy

Pascal

```
IObjConverter = interface;
```

C++

```
__interface IObjConverter;
```

File

InvokeRegistry

Description

IObjConverter defines methods that aid in the conversion between remotable classes (classes that descend from TRemotable (see page 15556)) and their representation in the messages that pass between clients and Web Service providers. That is, this interface provides methods that are used when converting between a remotable class, whether it represents a complex type in an invokable interface, an attachment, or a SOAP header, and its SOAP representation. The TSOAPDomConv class implements this interface.

Note: C++ method declarations that use IObjConverter use the _di_IObjConverter type instead. This type is a DelphiInterface wrapper around the IObjConverter interface:

Note: typedef System::DelphiInterface (see page 22167)< IObjConverter > _di_IObjConverter;

See Also

TSOAPDomCOnv

TRemotable (see page 15556)

TSoapHeader

TSoapAttachment

84.2.2.1 IObjConverter Members

The following tables list the members exposed by IObjConverter.

Public Methods

	Name	Description
	AddAttachment (see page 15611)	Stores information about a SOAP attachment so that it is included in the final message that is created.
	FindAttachment (see page 15612)	Returns a TSOAPAttachment (see page 15583) instance that represent the attachment with a specified ID.
	InitObjectFromSOAP (see page 15612)	Initializes an instance of a remotable object to reflect the values in its SOAP representation.
	ObjInstanceToSOAP (see page 15612)	Returns the SOAP representation of a remotable object.

84.2.2.2 IObjConverter Methods

The methods of the IObjConverter class are listed here.

Public Methods

	Name	Description
	AddAttachment (see page 15611)	Stores information about a SOAP attachment so that it is included in the final message that is created.
	FindAttachment (see page 15612)	Returns a TSOAPAttachment (see page 15583) instance that represent the attachment with a specified ID.
	InitObjectFromSOAP (see page 15612)	Initializes an instance of a remotable object to reflect the values in its SOAP representation.
	ObjInstanceToSOAP (see page 15612)	Returns the SOAP representation of a remotable object.

84.2.2.2.1 IObjConverter.AddAttachment Method

Stores information about a SOAP attachment so that it is included in the final message that is created.

Pascal

```
procedure AddAttachment(Attachment: TSOAPAttachment; const AContentId: string);
```

C++

```
__fastcall AddAttachment(TSOAPAttachment Attachment, const AnsiString AContentId);
```

Description

AddAttachment stores a SOAP attachment so that it can be used to generate part of the multipart form that includes the SOAP envelope as another of its parts.

Attachment is the attachment that should be added to the multipart form.

AContentId is a string that uniquely identifies the attachment. It is added as a header to the part of the form that contains the attachment so that XML in the SOAP envelope can refer to the attachment. When interpreting the encoded SOAP envelope, this value is passed to the FindAttachment (see page 15612) method so that the information in the attachment part can be used to recreate the TSOAPAttachment (see page 15583) instance.

See Also

FindAttachment (see page 15612)

84.2.2.2.2 IObjConverter.FindAttachment Method

Returns a TSOAPAttachment (see page 15583) instance that represent the attachment with a specified ID.

Pascal

```
function FindAttachment(const AContentId: string): TSOAPAttachment;
```

C++

```
__fastcall TSOAPAttachment FindAttachment(const AnsiString AContentId);
```

Description

FindAttachment locates the attachment that is identified by a specified content ID (HREF).

AContentId is the unique identifier for the attachment that was supplied to the AddAttachment (see page 15611) method when the attachment was first added to the multipart form.

FindAttachment returns a TSOAPAttachment (see page 15583) instance that represents the attachment.

See Also

AddAttachment (see page 15611)

84.2.2.2.3 IObjConverter.InitObjectFromSOAP Method

Initializes an instance of a remotable object to reflect the values in its SOAP representation.

Pascal

```
procedure InitObjectFromSOAP(Instance: TObject; RootNode: IXMLNode; Node: IXMLNode);
```

C++

```
__fastcall InitObjectFromSOAP(TObject * Instance, IXMLNode RootNode, IXMLNode Node);
```

Description

InitObjectFromSOAP assigns values to the published properties of a remotable object that represents a node from the SOAP envelope.

Instance is the remotable object to initialize from its SOAP representation.

RootNode is an ancestor node of the node for the remotable object that can contain information needed for initializing the remotable object.

Node is the node in the SOAP envelope that represents the remotable object.

See Also

ObjInstanceToSOAP (see page 15612)

84.2.2.2.4 IObjConverter.ObjInstanceToSOAP Method

Returns the SOAP representation of a remotable object.

Pascal

```
function ObjInstanceToSOAP(Instance: TObject; RootNode: IXMLNode; ParentNode: IXMLNode; const
NodeName: InvString; const NodeNamespace: InvString; ObjConvOpts: TObjectConvertOptions; out
RefID: InvString): IXMLNode;
```

C++

```
__fastcall IXMLNode ObjInstanceToSOAP(TObject * Instance, IXMLNode RootNode, IXMLNode
```

```
ParentNode, const InvString NodeName, const InvString NodeNamespace, TObjectConvertOptions
ObjConvOpts, InvString RefID);
```

Description

ObjInstanceToSOAP adds a node to represent the value of a remotable object.

Instance is the remotable object to encode as its SOAP representation.

RootNode is an ancestor of the node to which information about the node can be added as attributes.

ParentNode is the node in the SOAP representation that should act as a parent to the node that ObjInstanceToSOAP generates.

NodeName is the name of the node that is generated.

NodeNamespace is the URI for the namespace in which NodeName is defined.

ObjConvOpts are flags that customize the way the node is serialized.

RefID returns a reference ID for the node. This is used if the node is multi-referenced.

ObjInstanceToSOAP returns the XML for the newly generated node.

See Also

InitObjectFromSOAP (see page 15612)

84.2.3 ISOAPHeaders Interface

ISOAPHeaders is the interface for accessing the headers that accompany a message encoded using SOAP.

Class Hierarchy

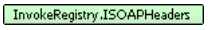

Pascal

```
ISOAPHeaders = interface;
```

C++

```
__interface ISOAPHeaders;
```

File

InvokeRegistry

Description

ISOAPHeaders is the interface for accessing the headers of a SOAP envelope. It defines methods for adding headers that are sent with an outgoing message and for retrieving the headers that were included in an incoming message.

On the server, the implementation class for in invokable interface supports this interface. You can obtain a reference to the ISOAPHeaders interface from the implementation class (a TInvokableClass (see page 15520) descendant). You can then use this interface to retrieve the headers that were included in an incoming request or to add headers to an outgoing response message. If your implementation class does not descend from TInvokableClass (see page 15520), you can obtain a reference to this interface using the global GetSOAPHeaders function.

For client-side header support, TRIO implements the ISOAPHeaders interface. However, you can't obtain an ISOAPHeaders interface directly from a remote interfaced object, because when you try to obtain an interface from TRIO or one of its descendants, it assumes that you are trying to obtain an invokable interface for a server object and generates an in-memory vtable. Instead, you must access the ISOAPHeaders interface from the invokable interface that the remote interface object supports.

Note: C++ method declarations that use ISOAPHeaders use the _di_ISOAPHeaders type instead. This type is a DelphiInterface wrapper around the ISOAPHeaders interface:

Note: typedef System::DelphiInterface (see page 22167)< ISOAPHeaders > _di_ISOAPHeaders;

See Also

TRIO (see page 18718)

TInvokableClass (see page 15520)

TSOAPHeaders (see page 15600)

GetSOAPHeaders

Defining and Using SOAP Headers

Processing Headers in Client Applications

84.2.3.1 ISOAPHeaders Members

The following tables list the members exposed by ISOAPHeaders.

Public Methods

	Name	Description
	Get (see page 15614)	This is the overview for the Get method overload.
	GetOwnsSentHeaders (see page 15615)	Returns the value of the OwnsSentHeaders (see page 15617) property.
	Send (see page 15616)	Adds a header to an outgoing message.
	SendAt (see page 15616)	Retrieves a header from an outgoing message, given its position.
	SendCount (see page 15617)	Indicates the number of headers that have been added to an outgoing message.
	SetOwnsSentHeaders (see page 15617)	Sets the value of the OwnsSentHeaders (see page 15617) property.

Public Properties

	Name	Description
	OwnsSentHeaders (see page 15617)	Indicates who is responsible for freeing header objects that are included in outbound messages.

84.2.3.2 ISOAPHeaders Methods

The methods of the ISOAPHeaders class are listed here.

Public Methods

	Name	Description
	Get (see page 15614)	This is the overview for the Get method overload.
	GetOwnsSentHeaders (see page 15615)	Returns the value of the OwnsSentHeaders (see page 15617) property.
	Send (see page 15616)	Adds a header to an outgoing message.
	SendAt (see page 15616)	Retrieves a header from an outgoing message, given its position.
	SendCount (see page 15617)	Indicates the number of headers that have been added to an outgoing message.
	SetOwnsSentHeaders (see page 15617)	Sets the value of the OwnsSentHeaders (see page 15617) property.

84.2.3.2.1 Get Method

This is the overview for the Get method overload.

Overload List

	Name	Description
	ISOAPHeaders.Get (TClass) (see page 15615)	Retrieves a header from an incoming message.
	ISOAPHeaders.Get (TClass, TSOAPHeader) (see page 15615)	Retrieves a header from an incoming message.
	ISOAPHeaders.Get (WideString, WideString) (see page 15615)	Retrieves a header from an incoming message.

84.2.3.2.1.1 ISOAPHeaders.Get Method (TClass)

Retrieves a header from an incoming message.

Pascal

```
procedure Get(Cls: TClass; out Hdr: TSOAPHeader); overload;
function Get(Cls: TClass): TSOAPHeader; overload;
function Get(const Name: WideString; const URI: WideString): TSOAPHeader; overload;
```

C++

```
__fastcall Get(TClass Cls, TSOAPHeader Hdr);
__fastcall TSOAPHeader Get(TClass Cls);
__fastcall TSOAPHeader Get(const BSTR Name, const BSTR URI);
```

Description

Call Get to access the header object that is automatically created to represent one of the headers on an incoming messages. On the server, this is a header on the request message. On the client, this is a header on the response message.

Cls is the class reference that indicates the type of header class you want to retrieve. Each header class corresponds to a type of header, where the type name under which the class is registered corresponds to the name of the header node.

Hdr returns a TSOAPHeader (see page 15596) descendant that represents the header of the incoming message. If this syntax is not used, Get returns the header instance.

Name is the name of the header (the node name for the header node) that you want to retrieve.

URI is the URI for the namespace in which Name is defined.

See Also

Send (see page 15616)

DataContext (see page 15561)

84.2.3.2.2 ISOAPHeaders.GetOwnsSentHeaders Method

Returns the value of the OwnsSentHeaders (see page 15617) property.

Pascal

```
function GetOwnsSentHeaders: Boolean;
```

C++

```
__fastcall Boolean GetOwnsSentHeaders();
```

Description

GetOwnsSentHeaders is the read implementation for the OwnsSentHeaders (see page 15617) property.

See Also

OwnsSentHeaders (see page 15617)

84.2.3.2.3 ISOAPHeaders.Send Method

Adds a header to an outgoing message.

Pascal

```
procedure Send(const Hdr: TSOAPHeader);
```

C++

```
__fastcall Send(const TSOAPHeader Hdr);
```

Description

Call Send to add a header to an outgoing message. On the client, call Send before executing a method on an invokable interface. On the server, call Send before returning from the method that the client invoked.

Hdr is the header object instance that represents the header you want to add to the outgoing message.

Note: The OwnsSentHeaders (see page 15617) property controls whether you must free the object that is supplied as the Hdr parameter after sending the message.

See Also

SendAt (see page 15616)

SendCount (see page 15617)

OwnsSentHeaders (see page 15617)

84.2.3.2.4 ISOAPHeaders.SendAt Method

Retrieves a header from an outgoing message, given its position.

Pascal

```
function SendAt(Index: Integer): TSOAPHeader;
```

C++

```
__fastcall TSOAPHeader SendAt(int Index);
```

Description

Call SendAt to retrieve the header object for a header you have already added to an outgoing message using the Send (see page 15616) method.

Index is the position of the header you want to retrieve, where 0 specifies the first header, 1 specifies the second header, and so on. The SendCount (see page 15617) property provides an upper bound for this index.

SendAt returns the header at the specified position.

See Also

Send (see page 15616)

SendCount (see page 15617)

84.2.3.2.5 ISOAPHeaders.SendCount Method

Indicates the number of headers that have been added to an outgoing message.

Pascal

```
function SendCount: Integer;
```

C++

```
__fastcall int SendCount();
```

Description

Call SendCount to determine the number of headers that have been added to an outgoing message. These headers are added by previous calls to the Send (see page 15616) method.

See Also

Send (see page 15616)

SendAt (see page 15616)

84.2.3.2.6 ISOAPHeaders.SetOwnsSentHeaders Method

Sets the value of the OwnsSentHeaders (see page 15617) property.

Pascal

```
procedure SetOwnsSentHeaders(Flag: Boolean);
```

C++

```
__fastcall SetOwnsSentHeaders(Boolean Flag);
```

Description

SetOwnsSentHeaders is the write implementation for the OwnsSentHeaders (see page 15617) property.

Flag is the new value of the OwnsSentHeaders (see page 15617) property.

See Also

OwnsSentHeaders (see page 15617)

84.2.3.3 ISOAPHeaders Properties

The properties of the ISOAPHeaders class are listed here.

Public Properties

	Name	Description
	OwnsSentHeaders (see page 15617)	Indicates who is responsible for freeing header objects that are included in outbound messages.

84.2.3.3.1 ISOAPHeaders.OwnsSentHeaders Property

Indicates who is responsible for freeing header objects that are included in outbound messages.

Pascal

```
property OwnsSentHeaders: Boolean;
```

C++

```
__property Boolean OwnsSentHeaders;
```

Description

Use OwnsSentHeaders to indicate whether the header object instances you create and add to messages by calling the Send (see page 15616) or SendAt (see page 15616) method should be freed by the object that implements this interface. When OwnsSentHeaders is true, the object implementing ISOAPHeaders (see page 15613) frees the header objects after sending them. When OwnsSentHeaders is false, the caller is responsible for freeing the header objects.

Note: For inbound headers (headers retrieved from incoming messages), the object that implements always frees them unless you call the Get (see page 15615) method. After a call to Get (see page 15615), the caller is responsible for freeing the header, except on the server, where the header object is freed by the header object's DataContext.

See Also

Send (see page 15616)

SendAt (see page 15616)

DataContext (see page 15561)

84.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetRemotableDataContext (see page 15619)	Returns the TDataContext (see page 15501) object that provides temporary storage during a call to an invokable interface.
	InvRegistry (see page 15620)	Provides access to the invocation registry.
	RemClassRegistry (see page 15620)	Provides access to the remotable class registry.
	RemTypeRegistry (see page 15621)	Provides access to the remotable type registry.
	SetRemotableDataContext (see page 15622)	Sets the data context that provides temporary storage during a call to an invokable interface.
	SubstituteStrings (see page 15623)	This is function InvokeRegistry.SubstituteStrings.

84.3.1 InvokeRegistry.GetRemotableDataContext Function

Returns the TDataContext (see page 15501) object that provides temporary storage during a call to an invokable interface.

Pascal

```
function GetRemotableDataContext: Pointer;
```

C++

```
void * GetRemotableDataContext();
```

File

InvokeRegistry

Description

GetRemotableDataContext returns a pointer to the data context object that stores all the temporary objects and values needed for implementing a call to an invokable interface. When a client application calls an invokable interface on a Web Service application, the Web Service application automatically creates a data context for the call. Any objects or variables that are needed for marshaling the interface call are automatically added to the data context for the duration of the call. In addition, any remotable objects (TRemotable (see page 15556) descendants) that are created while there is a remotable data context available are automatically added to the data context. When the interface call is complete, the data context is freed, and it automatically frees the stored objects and variables.

See Also

SetRemotableDataContext (see page 15622)

TRemotable (see page 15556)

TDataContext (see page 15501)

84.3.2 InvokeRegistry.InvRegistry Function

Provides access to the invocation registry.

Pascal

```
function InvRegistry: TInvokableClassRegistry;
```

C++

```
TInvokableClassRegistry InvRegistry();
```

File

InvokeRegistry

Description

Call InvRegistry to access the invocation registry. The returned object has two methods, RegisterInterface and RegisterInvokableClass, that you can use for registering invokable interfaces (interfaces compiled with RTTI) and the classes that implement them.

Server applications that implement Web Services need to register both the invokable interfaces and the implementation classes. The invoker component can then use the invocation registry to locate invokable interfaces and identify classes that implement them.

Client applications that call on Web Services register the invokable interfaces that they call. The remote interfaced object component can then use the invocation registry to generate a dynamic method table for calling the invokable interfaces.

Note: Both client and server applications must register invokable interfaces using the same namespace. You can either supply this namespace explicitly when you register an interface, or you can use the automatically generated namespace. If you use the automatically generated interface, the interface must be defined in the same unit on both client and server, and both client and server must have the same value set for the global AppNameSpacePrefix (see page 15644) variable.

See Also

IInvokable

THTTPSoapPascalInvoker

THTTPRio

AppNamespacePrefix

RemTypeRegistry (see page 15621)

84.3.3 InvokeRegistry.RemClassRegistry Function

Provides access to the remotable class registry.

Pascal

```
function RemClassRegistry: TRemotableClassRegistry;
```

C++

```
TRemotableClassRegistry RemClassRegistry();
```

File

InvokeRegistry

Description

RemClassRegistry returns the object that implements the remotable class registry. Use its RegisterXSClass method to register a remotable class in an application that publishes or uses Web Services.

Remotable classes are classes that can be transmitted between the client of a Web Services application and the server that implements the service. They can be used to represent non-scalar types for parameters and return values, or they can represent exceptions that occur when trying to execute a request to a Web Service application. All remotable classes are compiled with runtime type information (RTTI) and have a virtual constructor that the remotable class registry recognizes and uses to obtain class instances.

Both the client application and server application must register remotable classes before they can use them. They must register them to use the same namespace. You can either supply this namespace explicitly when you register the class, or you can use the automatically generated namespace. If you use the automatically generated interface, the class must be defined in the same unit on both client and server, and both client and server must have the same value set for the global AppNameSpacePrefix (see page 15644) variable.

Note: The remotable class registry is the same thing as the remotable type registry. You can use RemClassRegistry interchangeably with the RemTypeRegistry (see page 15621) function.

See Also

TRemotable (see page 15556)

ERemotableException (see page 15496)

AppNameSpacePrefix (see page 15644)

InvRegistry (see page 15620)

RemTypeRegistry (see page 15621)

84.3.4 InvokeRegistry.RemTypeRegistry Function

Provides access to the remotable type registry.

Pascal

```
function RemTypeRegistry: TRemotableTypeRegistry;
```

C++

```
TRemotableTypeRegistry RemTypeRegistry();
```

File

InvokeRegistry

Description

Call RemTypeRegistry to access to the remotable type registry. Use the RegisterXSClass method of the returned object to register a remotable class in an application that publishes or uses Web Services. Use its RegisterXSInfo method to register a remotable type.

Remotable classes are classes that can be transmitted between the client of a Web Services application and the server that implements the service. They can be used to represent non-scalar types for parameters and return values, or they can represent exceptions that occur when trying to execute a request to a Web Service application. All remotable classes are compiled with

runtime type information (RTTI) and have a virtual constructor that the remotable class registry recognizes and uses to obtain class instances.

Remotable types include dynamic arrays, enumerated types, and boolean types, which can be used in an invokable interface without resorting to a remotable class representation, but which must be registered because their values require special marshaling.

Both the client application and server application must register remotable classes and types before they can use them. They must register them to use the same namespace. You can either supply this namespace explicitly when you register the class or type, or you can use the automatically generated namespace. If you use the automatically generated interface, the class must be defined in the same unit on both client and server, and both client and server must have the same value set for the global AppNameSpacePrefix (see page 15644) variable.

Note: The remotable type registry is the same thing as the remotable class registry. You can use RemTypeRegistry interchangeably with the RemClassRegistry (see page 15620) function.

See Also

TRemotable (see page 15556)

ERemotableException (see page 15496)

AppNameSpacePrefix (see page 15644)

InvRegistry (see page 15620)

84.3.5 InvokeRegistry.SetRemotableDataContext Function

Sets the data context that provides temporary storage during a call to an invokable interface.

Pascal

```
procedure SetRemotableDataContext(Value: Pointer);
```

C++

```
SetRemotableDataContext(void * Value);
```

File

InvokeRegistry

Description

SetRemotableDataContext assigns a data context for a remote call to an invokable interface. Once a data context is assigned, it is available to store the temporary objects and values needed for implementing a call to an invokable interface. If the application creates a TRemotable (see page 15556) descendant while there is a remotable data context, that remotable object is automatically added to the data context and freed when the application frees the data context.

Note: The invoker component in a Web Services application automatically calls SetRemotableDataContext to set up a data context for incoming SOAP requests. Thus, there is typically no need to call this method in server applications.

Value is a pointer to a TDataContext (see page 15501) descendant that temporarily stores information for remote interface calls.

See Also

GetRemotableDataContext (see page 15619)

TRemotable (see page 15556)

TDataContext (see page 15501)

84.3.6 InvokeRegistry.SubstituteStrings Function

Pascal

```
function SubstituteStrings(const InputString: WideString; const SubString: WideString; const
Replacement: WideString): WideString;
```

C++

```
BSTR SubstituteStrings(const BSTR InputString, const BSTR SubString, const BSTR Replacement);
```

File

InvokeRegistry

Description

This is function InvokeRegistry.SubstituteStrings.

84.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	eHeaderMethodType (see page 15624)	eHeaderMethodType and THeaderMethodTypeArray (see page 15638) indicate the types of messages that should include a given header type.
	ObjectConvertOptions (see page 15629)	ObjectConvertOptions and TObjectConvertOptions (see page 15640) are flags that customize how to convert a remotable object to or from its SOAP representation.
	SerializationOptions (see page 15630)	SerializationOptions and TSerializationOptions (see page 15642) are flags that customize how to convert a remotable object to or from its SOAP representation.
	TIntfInvokeOption (see page 15632)	TIntfInvokeOption and TIntfInvokeOptions (see page 15639) represent flags that customize how to handle calls to an invokable interface.
	TObjMultiOptions (see page 15633)	TObjMultiOptions indicates how multiple references to the same remotable class should be translated into a SOAP representation.

Records

	Name	Description
	ExtNameMapItem (see page 15625)	This is record InvokeRegistry.ExtNameMapItem.
	IntfExceptionItem (see page 15625)	IntfExceptionItem and TExceptionItemArray (see page 15638) describe the association between a remotable exception and an interface that can raise it.
	IntfHeaderItem (see page 15626)	IntfHeaderItem and THeaderItemArray (see page 15638) describe the association between a header and the SOAP messages that include it.
	InvRegClassEntry (see page 15627)	This is record InvokeRegistry.InvRegClassEntry.
	InvRegIntfEntry (see page 15627)	InvRegIntfEntry stores information about an invokable interface.
	MethParamNameMapItem (see page 15629)	This is record InvokeRegistry.MethParamNameMapItem.
	TDynToClear (see page 15631)	This is record InvokeRegistry.TDynToClear.
	TRemHeaderEntry (see page 15634)	This is record InvokeRegistry.TRemHeaderEntry.
	TRemRegEntry (see page 15634)	TRemRegEntry stores information for the remotable type registry.

84.4.1 InvokeRegistry.eHeaderMethodType Enumeration

eHeaderMethodType and THeaderMethodTypeArray (see page 15638) indicate the types of messages that should include a given header type.

Pascal

```
eHeaderMethodType = (
  hmtAll,
  hmtRequest,
  hmtResponse
);
```

C++

```
enum eHeaderMethodType {
  hmtAll,
  hmtRequest,
  hmtResponse
};
```

File

InvokeRegistry

Description

THeaderMethodTypeArray (see page 15638) is a dynamic array of eHeaderMethodType values. The following table describes the possible eHeaderMethodType values:

Value	Meaning
hmtAll	The header should be included in all messages pertaining to the invokable interface.
hmtRequest	The header should be included in request messages in which the client attempts to invoke the invokable interface.
hmtResponse	The header should be included in response message that the server returns after executing a method on the invokable interface.

84.4.2 InvokeRegistry.ExtNameMapItem Record

Pascal

```
ExtNameMapItem = record
  Name: string;
  ExtName: WideString;
end;
```

C++

```
struct ExtNameMapItem {
  AnsiString Name;
  BSTR ExtName;
};
```

File

InvokeRegistry

Description

This is record InvokeRegistry.ExtNameMapItem.

84.4.3 InvokeRegistry.IntfExceptionItem Record

IntfExceptionItem and TExceptionItemArray (see page 15638) describe the association between a remotable exception and an interface that can raise it.

Pascal

```
IntfExceptionItem = record
  ClassType: TClass;
  MethodNames: string;
end;
```

C++

```
struct IntfExceptionItem {
  TClass ClassType;
  AnsiString MethodNames;
```

```
};
```

File

InvokeRegistry

Description

IntfExceptionItem describes a single remotable exception type. TExceptionItemArray (see page 15638) is an array of IntfExceptionItem descriptors. Typically, this array describes the set of exceptions associated with an invokable interface.

ClassType is the class reference for the remotable exception class.

MethodNames is a comma-delimited list of method names. If MethodNames is an empty string, the exception is associated with all the methods on an interface. If MethodNames is nonempty, the exception is associated only with those methods of the interface that the MethodNames field lists.

84.4.4 InvokeRegistry.IntfHeaderItem Record

IntfHeaderItem and THeaderItemArray (see page 15638) describe the association between a header and the SOAP messages that include it.

Pascal

```
IntfHeaderItem = record
  Info: PTypeInfo;
  ClassType: TClass;
  Namespace: WideString;
  Name: WideString;
  DefaultMethodType: eHeaderMethodType;
  DefaultRequired: Boolean;
  MethodNames: string;
  MethodTypes: THeaderMethodTypeArray;
  HeaderRequired: TRequiredArray;
end;
```

C++

```
struct IntfHeaderItem {
  PTypeInfo Info;
  TClass ClassType;
  BSTR Namespace;
  BSTR Name;
  eHeaderMethodType DefaultMethodType;
  Boolean DefaultRequired;
  AnsiString MethodNames;
  THeaderMethodTypeArray MethodTypes;
  TRequiredArray HeaderRequired;
};
```

File

InvokeRegistry

Description

IntfHeaderItem describes a single header type. THeaderItemArray (see page 15638) is an array of IntfHeaderItem descriptors. Typically, this array describes the set of headers associated with an invokable interface.

Info is a pointer to the type description for the interface that the header is associated with.

ClassType is the class reference for the header class (a TSOAPHeader (see page 15596) descendant).

NameSpace is the namespace in which the header's name is defined.

Name is the name of the header node.

DefaultMethodType indicates whether the header is included in request messages, response messages, or both. This value is only used if the MethodNames field is an empty string. If MethodNames is nonempty, use the MethodTypes field instead.

DefaultRequired indicates whether the header must appear in messages about the interface that Info describes. If DefaultRequired is false, the header is optional. This value is only used if the MethodNames field is an empty string. If MethodNames is nonempty, use the HeaderRequired field instead.

MethodNames is a comma-delimited list of method names. If empty, the header is associated with all methods on the interface. If MethodNames includes one or more names, the header is only associated with those methods of the interface that are contained in the list.

MethodTypes is a dynamic array of values that indicate whether the header is included in request messages, response messages, or both. Each value is associated with a method listed in the MethodNames field in the same position. If MethodNames is an empty string, MethodTypes is not used.

HeaderRequired is a dynamic array of values that indicate whether the header is required or optional. Each value is associated with a method listed in the MethodNames field in the same position. If MethodNames is an empty string, HeaderRequired is not used.

84.4.5 InvokeRegistry.InvRegClassEntry Record

Pascal

```
InvRegClassEntry = record
  ClassType: TClass;
  Proc: TCreateInstanceProc;
  URI: string;
end;
```

C++

```
struct InvRegClassEntry {
  TClass ClassType;
  TCreateInstanceProc Proc;
  AnsiString URI;
};
```

File

InvokeRegistry

Description

This is record InvokeRegistry.InvRegClassEntry.

84.4.6 InvokeRegistry.InvRegIntfEntry Record

InvRegIntfEntry stores information about an invokable interface.

Pascal

```
InvRegIntfEntry = record
  Name: string;
  ExtName: Widestring;
  UnitName: string;
  GUID: TGUID;
```

```
    Info: PTypeInfo;
    DefImpl: TClass;
    Namespace: Widestring;
    WSDLEncoding: WideString;
    Documentation: string;
    SOAPAction: string;
    ReturnParamNames: string;
    InvokeOptions: TIntfInvokeOptions;
    MethNameMap: array of ExtNameMapItem;
    MethParamNameMap: array of MethParamNameMapItem;
    IntfHeaders: array of IntfHeaderItem;
    IntfExceptions: array of IntfExceptionItem;
    UDDIOperator: String;
    UDDIBindingKey: String;
  end;
```

C++

```
  struct InvRegIntfEntry {
    AnsiString Name;
    BSTR ExtName;
    AnsiString UnitName;
    TGUID GUID;
    PTypeInfo Info;
    TClass DefImpl;
    BSTR Namespace;
    BSTR WSDLEncoding;
    AnsiString Documentation;
    AnsiString SOAPAction;
    AnsiString ReturnParamNames;
    TIntfInvokeOptions InvokeOptions;
    array of ExtNameMapItem MethNameMap;
    array of MethParamNameMapItem MethParamNameMap;
    array of IntfHeaderItem IntfHeaders;
    array of IntfExceptionItem IntfExceptions;
    AnsiString UDDIOperator;
    AnsiString UDDIBindingKey;
  };
```

File

InvokeRegistry

Description

InvRegIntfEntry is used by the invocation registry to store information about a registered invokable interface.

Name is the name of the registered invokable interface.

ExtName is the name of the corresponding port type that appears in the WSDL document describing this Web Service. ExtName can be an empty string if the interface and port type have the same name.

UnitName is the name of the unit in which the interface is defined.

GUID is the globally unique identifier for the interface.

Info points to the runtime type information (RTTI) compiled with the interface.

DefImpl is the default implementation class for the interface.

Namespace is the namespace that is uniquely associated with the interface.

WSDLEncoding is the encoding attribute, if any, of the xml header on a WSDL document that defines the port type to which the invokable interface corresponds.

Documentation is a brief description that documents the interface.

DefaultSOAPAction is the SOAP Action header (or set of SOAP Action headers) to use on messages directed to this interface. If

this is an empty string, the application generates a SOAP Action header that is based on the Namespace field. Assign SOAP Action headers using the RegisterDefaultSOAPAction or RegisterAllSOAPActions method. Note that if you have more than one interface with the same namespace URI, you must register them with different SOAP Action headers so that the server can distinguish them.

ReturnParamNames is a semicolon-delimited list of parameter names that map to the return value of a function.

InvokeOptions are the options for the interface that have been registered using the RegisterInvokeOptions method. These options customize the way the interface is called.

MethNameMap is an array of entries that describe the relationship between methods on the invokable interface and the operation names to which they correspond (if different).

MethParamNameMap is an array of entries that describe the relationship between parameters on a method of the invokable interface and the part names to which they correspond (if different).

IntfHeaders is an array of entries that describe the SOAP headers that should accompany requests directed at the invokable interface.

IntfExceptions is an array of entries that describe the exceptions that the interface can raise.

UDDIOperator is the URL of the UDDI registry, if any, that was used to import the interface definition.

UDDIBindingKey is the unique identifier for the TModel from UDDIOperator that describes the Web Service to which the interface belongs.

84.4.7 InvokeRegistry.MethParamNameMapItem Record

Pascal

```
MethParamNameMapItem = record
  MethName: string;
  ParamNameMap: array of ExtNameMapItem;
end;
```

C++

```
struct MethParamNameMapItem {
  AnsiString MethName;
  array of ExtNameMapItem ParamNameMap;
};
```

File

InvokeRegistry

Description

This is record InvokeRegistry.MethParamNameMapItem.

84.4.8 InvokeRegistry.ObjectConvertOptions Enumeration

ObjectConvertOptions and TObjectConvertOptions (see page 15640) are flags that customize how to convert a remotable object to or from its SOAP representation.

Pascal

```
ObjectConvertOptions = (
```

```
    ocoDontPrefixNode,
    ocoDontSerializeProps,
    ocoDontPutTypeAttr
  );
```

C++

```
  enum ObjectConvertOptions {
    ocoDontPrefixNode,
    ocoDontSerializeProps,
    ocoDontPutTypeAttr
  };
```

File

InvokeRegistry

Description

TObjectConvertOptions (see page 15640) is a set of ObjectConvertOptions values. ObjectConvertOptions values are flags that can be supplied when customizing the conversion between an object and its SOAP representation.

Value	Meaning
ocoDontPrefixNode	When set, the SOAP representation does not include a namespace prefix on node names.
ocoDontSerializeProps	When set, the SOAP representation does not include any representation for the published properties of the remotable object.
ocoDontPutTypeAttr	When set, nodes do not include an xsi:type attribute that indicates their type.

84.4.9 InvokeRegistry.SerializationOptions Enumeration

SerializationOptions and TSerializationOptions (see page 15642) are flags that customize how to convert a remotable object to or from its SOAP representation.

Pascal

```
  SerializationOptions = (
    xoHolderClass,
    xoAttributeOnLastMember,
    xoInlineArrays,
    xoLiteralParam,
    xoSimpleTypeWrapper,
    xoOption6,
    xoOption7,
    xoOption8,
    xoOption9,
    xoOptionA,
    xoOptionB,
    xoOptionC,
    xoOptionD,
    xoOptionE,
    xoOptionF,
    xoOptionG,
    xoOptionH,
    xoOptionI
  );
```

C++

```
  enum SerializationOptions {
    xoHolderClass,
    xoAttributeOnLastMember,
```

```
    xoInlineArrays,
    xoLiteralParam,
    xoSimpleTypeWrapper,
    xoOption6,
    xoOption7,
    xoOption8,
    xoOption9,
    xoOptionA,
    xoOptionB,
    xoOptionC,
    xoOptionD,
    xoOptionE,
    xoOptionF,
    xoOptionG,
    xoOptionH,
    xoOptionI
  };
```

File

InvokeRegistry

Description

TSerializationOptions (see page 15642) is a set of SerializationOptions values. SerializationOptions values are flags that can be registered with an invokable interface to indicate how it should be called. The following table lists the possible values.

Value	Meaning
xoHolderClass	The remotable object corresponds to a "holder" class. That is, the SOAP representation does not include a node for the class itself, just for its members. This is used when a type that would otherwise not require a remotable class uses a feature only available on remotable classes (such as attributes).
xoAttributeOnLastMember	Attributes of the remotable class are stored on the last member of the class that maps to an element node rather than on the node that corresponds to the class itself. This is necessary if a holder class has attributes.
xoInlineArrays	If the remotable class contains class members that are array types, the encoding of those members omits the type name of the array types. This results in more efficient encoding of holder classes that represent array types.
xoLiteralParam	The remotable class represents one of the two top-level nodes (the input or output value) of a document-literal encoding. The class is treated as a holder class if it is used as the direct input or output parameter of a function.
xoSimpleTypeWrapper	The remotable class is a wrapper for a simple type such as a string or integer. The value that the class represents is represented as a single published property of the remotable class.
xoOption5 - xoOptionI	The remaining options are reserved for future use.

84.4.10 InvokeRegistry.TDynToClear Record

Pascal

```
  TDynToClear = record
    P: Pointer;
    Info: PTypeInfo;
  end;
```

C++

```
  struct TDynToClear {
    void * P;
```

```
  PTypeInfo Info;
};
```

File

InvokeRegistry

Description

This is record InvokeRegistry.TDynToClear.

84.4.11 InvokeRegistry.TIntfInvokeOption Enumeration

TIntfInvokeOption and TIntfInvokeOptions (see page 15639) represent flags that customize how to handle calls to an invokable interface.

Pascal

```
TIntfInvokeOption = (
  ioDefault,
  ioDocument,
  ioLiteral,
  ioHasDefaultSOAPAction,
  ioHasReturnParamNames,
  ioHasNamespace,
  ioIsAppServerSOAP,
  ioHasUDDIInfo,
  ioHasAllSOAPActions
);
```

C++

```
enum TIntfInvokeOption {
  ioDefault,
  ioDocument,
  ioLiteral,
  ioHasDefaultSOAPAction,
  ioHasReturnParamNames,
  ioHasNamespace,
  ioIsAppServerSOAP,
  ioHasUDDIInfo,
  ioHasAllSOAPActions
};
```

File

InvokeRegistry

Description

TIntfInvokeOptions (see page 15639) is a set of TIntfInvokeOption values. TIntfInvokeOption values are flags that can be registered with an invokable interface to indicate how it should be called. The following table lists the possible values.

Value	Meaning
ioDefault	Calls to the interface use the default options unless overridden by other options. (RPC-based SOAP encoding and HTTP as a transport protocol).
ioDocument	The Web Services server uses document-style encoding.
ioLiteral	The Web Services server uses document-literal style and the types for input and output values have not been unwound to create a method call.
ioHasDefaultSOAPAction	A default SOAP action header has been registered for use with the interface.

ioHasReturnParamNames	A set of parameter names have been registered for use with the interface that are reserved for the return value of function calls. If this option is not set, the strings 'Return' and 'Result' are used.
ioHasNamespace	A namespace was explicitly provided when the interface was registered, rather than using an automatically-generated namespace.
ioIsAppServerSOAP	The interface descends from IAppServerSOAP. Such interfaces are used in multi-tier database applications to interact with the application server.
ioHasUDDIInfo	UDDI information has been registered for the interface. This option is set when the Web Service description is imported using the UDDI browser.
ioHasAllSOAPActions	The SOAPAction headers could not be collapsed into a single string. When representing the SOAP action headers, a single string is created where the first character represents a delimiter that marks the boundary between the various SOAP action headers.

84.4.12 InvokeRegistry.TObjMultiOptions Enumeration

TObjMultiOptions indicates how multiple references to the same remotable class should be translated into a SOAP representation.

Pascal

```
TObjMultiOptions = (
  ocDefault,
  ocMultiRef,
  ocNoMultiRef
);
```

C++

```
enum TObjMultiOptions {
  ocDefault,
  ocMultiRef,
  ocNoMultiRef
};
```

File

InvokeRegistry

Description

TObjMultiOptions indicates a class-specific override to the default strategy for converting multiple references to the same object instance into the corresponding SOAP representation. It indicates whether a remotable object should be treated by value (like a structure) or by reference (where object identity is meaningful).

Value	Meaning
ocDefault	Use the default strategy for handling multiple references.
ocMultiRef	Always indicate in the SOAP packet when multiple references refer to the same instance of the specified class.
ocNoMultiRef	Never indicate in the SOAP packet when multiple references refer to the same instance of the specified class. That is, always treat it "by value".

See Also

TRemotableTypeRegistry (see page 15562)

84.4.13 InvokeRegistry.TRemHeaderEntry Record

Pascal

```
TRemHeaderEntry = record
  Name: WideString;
  ExtName: WideString;
  URI: WideString;
end;
```

C++

```
struct TRemHeaderEntry {
  BSTR Name;
  BSTR ExtName;
  BSTR URI;
};
```

File

InvokeRegistry

Description

This is record InvokeRegistry.TRemHeaderEntry.

84.4.14 InvokeRegistry.TRemRegEntry Record

TRemRegEntry stores information for the remotable type registry.

Pascal

```
TRemRegEntry = record
  ClassType: TClass;
  Info: PTypeInfo;
  URI: WideString;
  Name: WideString;
  ExtName: WideString;
  IsScalar: Boolean;
  MultiRefOpt: TObjMultiOptions;
  SerializationOpt: TSerializationOptions;
  PropNameMap: array of ExtNameMapItem;
end;
```

C++

```
struct TRemRegEntry {
  TClass ClassType;
  PTypeInfo Info;
  BSTR URI;
  BSTR Name;
  BSTR ExtName;
  Boolean IsScalar;
  TObjMultiOptions MultiRefOpt;
  TSerializationOptions SerializationOpt;
  array of ExtNameMapItem PropNameMap;
};
```

File

InvokeRegistry

Description

TRemRegEntry is the type that the remotable type registry uses to store information about a registered type. The TRemRegEntry type is used to represent both remotable classes and remotable types. It contains the following members:

ClassType is the class reference for the class. It is not used for entries that represent types that are not classes.

Info is the type info pointer for a registered class or type.

URI is the namespace URI that the registry associates with the class or type.

Name is the type name of the class or type, as it appears in encoded method calls or WSDL documents.

ExtName is the same as Name.

IsScalar indicates whether the entry describes a class or type that is marshaled as a scalar type.

MultiRefOpt indicates how to marshal multiple instances of a registered class. It is not used for entries that represent types.

SerializationOpt indicates how to convert between a registered class and its encoded representation.

PropNameMap is an array of entries that describe the relationship between properties on the invokable interface and the external names to which they correspond (if different).

84.5 Types

The following table lists types in this documentation.

Types

Name	Description
ERemotableExceptionClass (see page 15636)	This is type InvokeRegistry.ERemotableExceptionClass.
InvString (see page 15637)	This is type InvokeRegistry.InvString.
PTRemotable (see page 15637)	This is type InvokeRegistry.PTRemotable.
TCreateInstanceProc (see page 15637)	TCreateInstanceProc is the type of the factory routine that returns an instance of an invokable class.
TExceptionItemArray (see page 15638)	This is type InvokeRegistry.TExceptionItemArray.
THeaderItemArray (see page 15638)	This is type InvokeRegistry.THeaderItemArray.
THeaderMethodTypeArray (see page 15638)	This is type InvokeRegistry.THeaderMethodTypeArray.
TIntfInvokeOptions (see page 15639)	This is type InvokeRegistry.TIntfInvokeOptions.
TInvokableClassClass (see page 15639)	This is type InvokeRegistry.TInvokableClassClass.
TInvRegIntfEntryArray (see page 15639)	This is type InvokeRegistry.TInvRegIntfEntryArray.
TObjectConvertOptions (see page 15640)	This is type InvokeRegistry.TObjectConvertOptions.
TPascalRemotableClassRegistry (see page 15640)	TPascalRemotableClassRegistry is the type of the remotable class registry.
TPascalRemotableTypeRegistry (see page 15640)	TPascalRemotableTypeRegistry is the type of the remotable type registry.
TRemotableClass (see page 15641)	This is type InvokeRegistry.TRemotableClass.
TRemotableClassRegistry (see page 15641)	This is type InvokeRegistry.TRemotableClassRegistry.
TRemotableXSClass (see page 15641)	This is type InvokeRegistry.TRemotableXSClass.
TRemRegEntryArray (see page 15642)	This is type InvokeRegistry.TRemRegEntryArray.
TRequiredArray (see page 15642)	TRequiredArray is a dynamic array of boolean values.
TSerializationOptions (see page 15642)	This is type InvokeRegistry.TSerializationOptions.
TSOAPAttachmentClass (see page 15643)	This is type InvokeRegistry.TSOAPAttachmentClass.
TSOAPHeaderClass (see page 15643)	This is type InvokeRegistry.TSOAPHeaderClass.

84.5.1 InvokeRegistry.ERemotableExceptionClass Type

Pascal

```
ERemotableExceptionClass = class of ERemotableException;
```

C++

```
ERemotableException ERemotableExceptionClass;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.ERemotableExceptionClass.

84.5.2 InvokeRegistry.InvString Type

Pascal

```
InvString = WideString;
```

C++

```
BSTR InvString;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.InvString.

84.5.3 InvokeRegistry.PTRemotable Type

Pascal

```
PTRemotable = ^TRemotable;
```

C++

```
^TRemotable PTRemotable;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.PTRemotable.

84.5.4 InvokeRegistry.TCreateInstanceProc Type

TCreateInstanceProc is the type of the factory routine that returns an instance of an invokable class.

Pascal

```
TCreateInstanceProc = procedure (out obj: TObject);
```

C++

```
(out obj: TObject) ( TCreateInstanceProc)();
```

File

InvokeRegistry

Description

TCreateInstanceProc is the type for procedures that return an instance of an invokable class on behalf of the invocation registry. When the class that implements an invokable interface is not a descendant of TInvokableClass (see page 15520), or when you want to control the instantiation of the implementation class (for example to use a single shared instance for all incoming requests), supply a procedure of this type when registering the implementation class.

obj returns an instance of the invokable class, which can handle the current incoming request.

84.5.5 InvokeRegistry.TExceptionItemArray Type

Pascal

```
TExceptionItemArray = array of IntfExceptionItem;
```

C++

```
array of IntfExceptionItem TExceptionItemArray;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TExceptionItemArray.

84.5.6 InvokeRegistry.THeaderItemArray Type

Pascal

```
THeaderItemArray = array of IntfHeaderItem;
```

C++

```
array of IntfHeaderItem THeaderItemArray;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.THeaderItemArray.

84.5.7 InvokeRegistry.THeaderMethodTypeArray Type

Pascal

```
THeaderMethodTypeArray = array of eHeaderMethodType;
```

C++

```
array of eHeaderMethodType THeaderMethodTypeArray;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.THeaderMethodTypeArray.

84.5.8 InvokeRegistry.TIntfInvokeOptions Type

Pascal

```
TIntfInvokeOptions = set of TIntfInvokeOption;
```

C++

```
TIntfInvokeOption TIntfInvokeOptions;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TIntfInvokeOptions.

84.5.9 InvokeRegistry.TInvokableClassClass Type

Pascal

```
TInvokableClassClass = class of TInvokableClass;
```

C++

```
TInvokableClass TInvokableClassClass;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TInvokableClassClass.

84.5.10 InvokeRegistry.TInvRegIntfEntryArray Type

Pascal

```
TInvRegIntfEntryArray = array of InvRegIntfEntry;
```

C++

```
array of InvRegIntfEntry TInvRegIntfEntryArray;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TInvRegIntfEntryArray.

84.5.11 InvokeRegistry.TObjectConvertOptions Type

Pascal

```
TObjectConvertOptions = set of ObjectConvertOptions;
```

C++

```
ObjectConvertOptions TObjectConvertOptions;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TObjectConvertOptions.

84.5.12 InvokeRegistry.TPascalRemotableClassRegistry Type

TPascalRemotableClassRegistry is the type of the remotable class registry.

Pascal

```
TPascalRemotableClassRegistry = TRemotableTypeRegistry;
```

C++

```
TRemotableTypeRegistry TPascalRemotableClassRegistry;
```

File

InvokeRegistry

Description

TPascalRemotableClassRegistry is a synonym for TRemotableTypeRegistry (see page 15562). It is defined for compatibility with Delphi 6.

See Also

RemClassRegistry (see page 15620)

TRemotable (see page 15556)

84.5.13 InvokeRegistry.TPascalRemotableTypeRegistry Type

TPascalRemotableTypeRegistry is the type of the remotable type registry.

Pascal

```
TPascalRemotableTypeRegistry = TRemotableTypeRegistry;
```

C++

```
TRemotableTypeRegistry TPascalRemotableTypeRegistry;
```

File

InvokeRegistry

Description

TPascalRemotableTypeRegistry is a synonym for TRemotableTypeRegistry (see page 15562). It is defined for compatibility with Delphi 6.

See Also

RemTypeRegistry (see page 15621)

TRemotable (see page 15556)

84.5.14 InvokeRegistry.TRemotableClass Type

Pascal

```
TRemotableClass = class of TRemotable;
```

C++

```
TRemotable TRemotableClass;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TRemotableClass.

84.5.15 InvokeRegistry.TRemotableClassRegistry Type

Pascal

```
TRemotableClassRegistry = TRemotableTypeRegistry;
```

C++

```
TRemotableTypeRegistry TRemotableClassRegistry;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TRemotableClassRegistry.

84.5.16 InvokeRegistry.TRemotableXSClass Type

Pascal

```
TRemotableXSClass = class of TRemotableXS;
```

C++

```
TRemotableXS TRemotableXSClass;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TRemotableXSClass.

84.5.17 InvokeRegistry.TRemRegEntryArray Type

Pascal

```
TRemRegEntryArray = array of TRemRegEntry;
```

C++

```
array of TRemRegEntry TRemRegEntryArray;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TRemRegEntryArray.

84.5.18 InvokeRegistry.TRequiredArray Type

TRequiredArray is a dynamic array of boolean values.

Pascal

```
TRequiredArray = array of Boolean;
```

C++

```
array of Boolean TRequiredArray;
```

File

InvokeRegistry

Description

TRequiredArray is a dynamic array of boolean values. Each value indicates whether a corresponding item is required (true) or optional (false).

84.5.19 InvokeRegistry.TSerializationOptions Type

Pascal

```
TSerializationOptions = set of SerializationOptions;
```

C++

```
SerializationOptions TSerializationOptions;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TSerializationOptions.

84.5.20 InvokeRegistry.TSOAPAttachmentClass Type

Pascal

```
TSOAPAttachmentClass = class of TSOAPAttachment;
```

C++

```
TSOAPAttachment TSOAPAttachmentClass;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TSOAPAttachmentClass.

84.5.21 InvokeRegistry.TSOAPHeaderClass Type

Pascal

```
TSOAPHeaderClass = class of TSOAPHeader;
```

C++

```
TSOAPHeader TSOAPHeaderClass;
```

File

InvokeRegistry

Description

This is type InvokeRegistry.TSOAPHeaderClass.

84.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
AppNameSpacePrefix (see page 15644)	Supplies a unique prefix to use for automatically generated namespaces given to invokable interfaces and remotable types.

84.6.1 InvokeRegistry.AppNameSpacePrefix Variable

Supplies a unique prefix to use for automatically generated namespaces given to invokable interfaces and remotable types.

Pascal

```
AppNameSpacePrefix: string;
```

C++

```
AnsiString AppNameSpacePrefix;
```

File

InvokeRegistry

Description

Set AppNameSpacePrefix to a unique string that the invocation registry and remotable type registry use when generating namespaces. These generated namespaces must be unique strings that identify invokable interfaces and remotable classes in a Web Service application. They are generated by the RegisterInterface method of the invocation registry and the RegisterXSClass and RegisterXSInfo methods of the remotable class registry if the application does not explicitly supply a namespace.

The invocation registry and remotable type registry generate namespaces using the value of AppNameSpacePrefix, the name of the unit in which the interface or class is defined, and the name of the class, type, or interface. If your unit and interface or class names are at all generic, it is a good idea to supply a unique value to AppNameSpacePrefix to guard against the possibility of another interface, type, or class with the same name that is registered in another unit with the same name.

Both client and server applications must register invokable interfaces and remotable types and classes using the same namespace. If you rely on automatically-generated namespaces, be sure to use the same value of AppNameSpacePrefix in both the client and the server applications.

See Also

InvRegistry (see page 15620)

TInvokableClassRegistry (see page 15526)

TRemotableTypeRegistry (see page 15562)

RemTypeRegistry (see page 15621)

84.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
AS_ATTRIBUTE (see page 15645)	Identifies a property of a remotable class as one that corresponds to an attribute in the SOAP encoding.
XMLBase64Types (see page 15645)	This is constant InvokeRegistry.XMLBase64Types.
XMLSchemaInstNamepspaces (see page 15646)	This is constant InvokeRegistry.XMLSchemaInstNamepspaces.
XMLSchemaNamepspaces (see page 15646)	This is constant InvokeRegistry.XMLSchemaNamepspaces.

84.7.1 InvokeRegistry.AS_ATTRIBUTE Constant

Identifies a property of a remotable class as one that corresponds to an attribute in the SOAP encoding.

Pascal

```
AS_ATTRIBUTE = False;
```

C++

```
AS_ATTRIBUTE = False;
```

File

InvokeRegistry

Description

Assign AS_ATTRIBUTE as the value of the stored directive on a property declaration in a TRemotable (see page 15556) descendant to indicate that the property is encoded as an attribute node rather than as an element node. If a property declaration does not include a stored directive with a value of AS_ATTRIBUTE, it is encoded as an element node instead.

Note: Note:AS_ATTRIBUTE must be used directly. If you set the stored directive to a function that returns AS_ATTRIBUTE (false), the property is still encoded as an element node.

See Also

TRemotable (see page 15556)

TRemotableXS (see page 15580)

84.7.2 InvokeRegistry.XMLBase64Types Constant

Pascal

```
XMLBase64Types: array[0..1] of InvString = ('base64Binary', 'bin.base64');
```

C++

```
array[0..1] of InvString XMLBase64Types = ('base64Binary', 'bin.base64');
```

File

InvokeRegistry

Description

This is constant InvokeRegistry.XMLBase64Types.

84.7.3 InvokeRegistry.XMLSchemaInstNamepspaces Constant

Pascal

```
XMLSchemaInstNamepspaces: array[0..2] of InvString = (SXMLSchemaInstURI_1999,
SXMLSchemaInstURI_2000_10, SXMLSchemaInstURI);
```

C++

```
array[0..2] of InvString XMLSchemaInstNamepspaces = (SXMLSchemaInstURI_1999,
SXMLSchemaInstURI_2000_10, SXMLSchemaInstURI);
```

File

InvokeRegistry

Description

This is constant InvokeRegistry.XMLSchemaInstNamepspaces.

84.7.4 InvokeRegistry.XMLSchemaNamepspaces Constant

Pascal

```
XMLSchemaNamepspaces: array[0..2] of InvString = (SXMLSchemaURI_1999, SXMLSchemaURI_2000_10,
SXMLSchemaURI_2001);
```

C++

```
array[0..2] of InvString XMLSchemaNamepspaces = (SXMLSchemaURI_1999, SXMLSchemaURI_2000_10,
SXMLSchemaURI_2001);
```

File

InvokeRegistry

Description

This is constant InvokeRegistry.XMLSchemaNamepspaces.

85 ISAPIApp Namespace

This is namespace ISAPIApp.

Topics

Name	Description
Classes (see page 15648)	The following table lists classes in this documentation.
Functions (see page 15657)	The following table lists functions in this documentation.
Variables (see page 15659)	The following table lists variables in this documentation.

Classes

	Name	Description
	TISAPIApplication (see page 15648)	TISAPIApplication encapsulates an ISAPI or NSAPI server application.
	TISAPIFactory (see page 15654)	

Functions

	Name	Description
	GetExtensionVersion (see page 15657)	Returns the name and version number of an ISAPI or NSAPI application.
	HttpExtensionProc (see page 15657)	Receives an HTTP request message for an ISAPI or NSAPI application.
	TerminateExtension (see page 15658)	Indicates whether an ISAPI or NSAPI application is ready to be shut down.

Variables

Name	Description
DispatchThread (see page 15659)	This is variable ISAPIApp.DispatchThread.

85.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TISAPIApplication (see page 15648)	TISAPIApplication encapsulates an ISAPI or NSAPI server application.
	TISAPIFactory (see page 15654)	

85.1.1 TISAPIApplication Class

TISAPIApplication encapsulates an ISAPI or NSAPI server application.

Class Hierarchy

Pascal

```
TISAPIApplication = class(TWebApplication);
```

C++

```
class TISAPIApplication : public TWebApplication;
```

File

ISAPIApp

Description

TISAPIApplication provides the fundamental behavior of an ISAPI or NSAPI server application. The HTTP server calls the methods of TISAPIApplication when it receives an HTTP request message. TISAPIApplication creates a thread to manage the request message, and a creates a new instance of the Web module for that thread, or reuses an old instance that was deactivated after a previous response was sent.

TISAPIApplication creates TISAPIRequest and TISAPIResponse objects to represent the request message and any response that should be sent in return, passes these to the dispatcher so that the response can be filled in, and sends the response if it was not already sent by a TWebActionItem.

Each ISAPI server project automatically declares an Application variable of type TISAPIApplication as the instance of the application. This declaration overrides the Application variable declared in the Forms (see page 13657) unit if the ISAPIApp (see page 15647) unit is declared last in the uses list of the project file. Every ISAPI server application must include the ISAPIApp (see page 15647) unit.

To work with an Apache server, use the TApacheApplication object. To work with a CGI server, use the TCGIApplication object instead.

See Also

TCGIApplication (see page 2554)

TISAPIRequest (see page 15662)

TISAPIResponse (see page 15670)

TWebActionItem (see page 15130)

TWebDispatcher (see page 15147)

85.1.1.1 TISAPIApplication Members

The following tables list the members exposed by TISAPIApplication.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActivateWebModules (see page 25453)	Activates a Web module for the application and returns it.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DeactivateWebModules (see page 25453)	Deactivates the Web module of the application.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleRequest (see page 25454)	Responds to an HTTP request message received by the server.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddWebModuleFactory (see page 25453)	Adds a Web module factory to the request handler.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 15652)	Creates an instance of TWebApplication.
V	CreateForm (see page 24623)	Registers the single Web module with the Web application.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 15652)	Destroys an instance of TWebApplication.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetExtensionVersion (see page 15652)	Returns the application name and version number.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.

	HandleException (see page 25454)	Responds to exceptions related to TWebRequestHandler (see page 25448) objects.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HttpExtensionProc (see page 15653)	Responds to an HTTP request message.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 24623)	Introduces an interface for a method that initializes the application.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Run (see page 24623)	Executes the Web application.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	TerminateExtension (see page 15653)	Returns true.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActiveCount (see page 25455)	Indicates the number of active connections currently being serviced by the application.
	CacheConnections (see page 25455)	Determines whether the Web modules used by the application are cached for re-use.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	InactiveCount (see page 25456)	Indicates the number of Web modules cached for later reuse.
	MaxConnections (see page 25457)	Sets or returns the MaxConnections used by the socket.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Title (see page 24624)	Specifies a title for the Web application.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	WebModuleClass (see page 25457)	Contains a reference to a Web module.
R	WebModuleFactory (see page 25457)	A list of Web module factory objects.
R	WebModuleFactoryCount (see page 25458)	The number of Web module factories in use by the request handler.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

85.1.1.2 TISAPIApplication.Create Constructor

Creates an instance of TWebApplication.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TISAPIApplication(TComponent * AOwner);
```

Description

Do not call Create. The single Web application object named Application is instantiated automatically when the CGIApp unit, the ISAPIApp unit, or the ApacheApp unit is included in the uses list of the project.

Create allocates the helper objects used by TWebApplication.

See Also

Create (see page 2766)

85.1.1.3 TISAPIApplication.Destroy Destructor

Destroys an instance of TWebApplication.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TISAPIApplication();
```

Description

Never call the Destroy or Free method. The application is freed automatically when the application stops running.

Destroy frees the helper objects that were created at startup, as well as all Web modules created by the application.

See Also

Free (see page 22300)

Run (see page 24623)

85.1.1.4 TISAPIApplication Methods

The methods of the TISAPIApplication class are listed here.

Public Methods

	Name	Description
	GetExtensionVersion (see page 15652)	Returns the application name and version number.
	HttpExtensionProc (see page 15653)	Responds to an HTTP request message.
	TerminateExtension (see page 15653)	Returns true.

85.1.1.4.1 TISAPIApplication.GetExtensionVersion Method

Returns the application name and version number.

Pascal

```
function GetExtensionVersion(var Ver: THSE_VERSION_INFO): BOOL;
```

C++

```
__fastcall BOOL GetExtensionVersion(THSE_VERSION_INFO Ver);
```

Description

The HTTP server calls GetExtensionVersion to obtain the application name and version number. GetExtensionVersion returns the version number for the HTTP functions in the runtime library, and the Title property of the ISAPI application object.

See Also

Title (see page 24624)

85.1.1.4.2 TISAPIApplication.HttpExtensionProc Method

Responds to an HTTP request message.

Pascal

```
function HttpExtensionProc(var ECB: TEXTENSION_CONTROL_BLOCK): DWORD;
```

C++

```
__fastcall DWORD HttpExtensionProc(TEXTENSION_CONTROL_BLOCK ECB);
```

Description

The HTTP server calls HttpExtensionProc when it receives an HTTP request message. HttpExtensionProc creates the TISAPIRequest and TISAPIResponse objects for the HTTP request message, and calls the HandleRequest message so that the response can be filled out and sent. Finally, HttpExtensionProc frees the request and response objects.

If the response is filled out and sent, HttpExtensionProc returns HSE_STATUS_SUCCESS. Otherwise it returns HSE_STATUS_ERROR.

See Also

TISAPIRequest (see page 15662)

TISAPIResponse (see page 15670)

HandleRequest (see page 25454)

TWebDispatcher (see page 15147)

85.1.1.4.3 TISAPIApplication.TerminateExtension Method

Returns true.

Pascal

```
function TerminateExtension(dwFlags: DWORD): BOOL;
```

C++

```
__fastcall BOOL TerminateExtension(DWORD dwFlags);
```

Description

The HTTP server calls TerminateExtension when it wants to shut down the ISAPI application. TerminateExtension always returns true, allowing the server to shut down.

85.1.2 TISAPIFactory Class

Class Hierarchy

System::TObject → ISAPIApp.TISAPIFactory

Pascal

```
TISAPIFactory = class(TObject);
```

C++

```
class TISAPIFactory : public TObject;
```

File

ISAPIApp

See Also

TCGIApplication (see page 2554)

85.1.2.1 TISAPIFactory Members

The following tables list the members exposed by TISAPIFactory.

Protected Methods

	Name	Description
	NewRequest (see page 15655)	This is NewRequest, a member of class TISAPIFactory.
	NewResponse (see page 15655)	This is NewResponse, a member of class TISAPIFactory.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 15655)	Creates an instance of Create. {Placeholder—Update for BCB}
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

85.1.2.2 TISAPIFactory.Create Constructor

Creates an instance of Create. {Placeholder—Update for BCB}

Pascal

```
constructor Create;
```

C++

```
__fastcall TISAPIFactory();
```

Description

Do not create instances of Create. Instead, use the new keyword, passing any arguments for the Create's constructor.

See Also

Destroy (see page 22287)

new

85.1.2.3 TISAPIFactory Methods

The methods of the TISAPIFactory class are listed here.

Protected Methods

	Name	Description
V	NewRequest (see page 15655)	This is NewRequest, a member of class TISAPIFactory.
V	NewResponse (see page 15655)	This is NewResponse, a member of class TISAPIFactory.

85.1.2.3.1 TISAPIFactory.NewRequest Method

Pascal

```
function NewRequest(var AECB: TEXTENSION_CONTROL_BLOCK): TISAPIRequest; virtual;
```

C++

```
virtual __fastcall TISAPIRequest NewRequest(TEXTENSION_CONTROL_BLOCK AECB);
```

Description

This is NewRequest, a member of class TISAPIFactory.

85.1.2.3.2 TISAPIFactory.NewResponse Method

Pascal

```
function NewResponse(ISAPIRequest: TISAPIRequest): TISAPIResponse; virtual;
```

C++

```
virtual __fastcall TISAPIResponse NewResponse(TISAPIRequest ISAPIRequest);
```

Description

This is NewResponse, a member of class TISAPIFactory.

85.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetExtensionVersion (see page 15657)	Returns the name and version number of an ISAPI or NSAPI application.
	HttpExtensionProc (see page 15657)	Receives an HTTP request message for an ISAPI or NSAPI application.
	TerminateExtension (see page 15658)	Indicates whether an ISAPI or NSAPI application is ready to be shut down.

85.2.1 ISAPIApp.GetExtensionVersion Function

Returns the name and version number of an ISAPI or NSAPI application.

Pascal

```
function GetExtensionVersion(var Ver: THSE_VERSION_INFO): BOOL; stdcall;
```

C++

```
__stdcall BOOL GetExtensionVersion(THSE_VERSION_INFO Ver);
```

File

ISAPIApp

Description

Do not call GetExtensionVersion. ISAPI and NSAPI DLLs export GetExtension to provide an entry point that is called by ISAPI or NSAPI Web servers. This allows the Web server to obtain version information about the web server application.

See Also

HttpExtensionProc (see page 15657)

TerminateExtension (see page 15658)

TIsapiapplication

85.2.2 ISAPIApp.HttpExtensionProc Function

Receives an HTTP request message for an ISAPI or NSAPI application.

Pascal

```
function HttpExtensionProc(var ECB: TEXTENSION_CONTROL_BLOCK): DWORD; stdcall;
```

C++

```
__stdcall DWORD HttpExtensionProc(TEXTENSION_CONTROL_BLOCK ECB);
```

File

ISAPIApp

Description

Do not call HttpExtensionProc. ISAPI and NSAPI DLLs export HttpExtensionProc to provide an entry point that is called by ISAPI or NSAPI Web servers when they receive HTTP request messages.

See Also

GetExtensionVersion (see page 15657)

TerminateExtension (see page 15658)

85.2.3 ISAPIApp.TerminateExtension Function

Indicates whether an ISAPI or NSAPI application is ready to be shut down.

Pascal

```
function TerminateExtension(dwFlags: DWORD): BOOL; stdcall;
```

C++

```
__stdcall BOOL TerminateExtension(DWORD dwFlags);
```

File

ISAPIApp

Description

Do not call TerminateExtension. TerminateExtension is exported by ISAPI and NSAPI DLLs. It provides an entry point that is called by ISAPI or NSAPI Web servers, when they are ready to shut down the ISAPI or NSAPI application, to check whether it would cause problems to do so.

See Also

HttpExtensionProc (see page 15657)

85.3 Variables

The following table lists variables in this documentation.

Variables

Name	Description
DispatchThread (see page 15659)	This is variable ISAPIApp.DispatchThread.

85.3.1 ISAPIApp.DispatchThread Variable

Pascal

```
DispatchThread: function (AECB: PEXTENSION_CONTROL_BLOCK): Boolean of object;
```

C++

```
function (AECB: PEXTENSION_CONTROL_BLOCK): Boolean of object DispatchThread;
```

File

ISAPIApp

Description

This is variable ISAPIApp.DispatchThread.

86 IsapiHTTP Namespace

This is namespace IsapiHTTP.

Topics

Name	Description
Classes (see page 15662)	The following table lists classes in this documentation.

Classes

	Name	Description
	TISAPIRequest (see page 15662)	TISAPIRequest represents client request information that is passed to a server application DLL.
	TISAPIResponse (see page 15670)	TISAPIResponse represents HTTP messages sent by an ISAPI application in response to an HTTP request message.

86.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TISAPIRequest (see page 15662)	TISAPIRequest represents client request information that is passed to a server application DLL.
	TISAPIResponse (see page 15670)	TISAPIResponse represents HTTP messages sent by an ISAPI application in response to an HTTP request message.

86.1.1 TISAPIRequest Class

TISAPIRequest represents client request information that is passed to a server application DLL.

Class Hierarchy

Pascal

```
TISAPIRequest = class(TWebRequest);
```

C++

```
class TISAPIRequest : public TWebRequest;
```

File

IsapiHTTP

Description

Use TISAPIRequest to interpret a client request to a server application DLL.

ISAPI application objects create TISAPIRequest objects to capture client requests. The properties of the TISAPIRequest objects are obtained from the Web server. The ISAPI application uses TISAPIRequest objects to create corresponding TISAPIResponse (see page 15670) objects. The TWebDispatcher for the application then invokes the TWebActionItem that is associated with the path portion of the URL of the TISAPIRequest object.

See Also

TISAPIApplication (see page 15648)

TISAPIResponse (see page 15670)

TWebActionItem (see page 15130)

TWebDispatcher (see page 15147)

Accessing Client Request Information

86.1.1.1 TISAPIRequest Members

The following tables list the members exposed by TISAPIRequest.

Protected Methods

	Name	Description
	GetDateVariable (see page 15665)	Extracts a date value from the extension control block.
	GetIntegerVariable (see page 15666)	Extracts an integer value from the extension control block.
	GetInternalPathInfo (see page 15169)	This is GetInternalPathInfo, a member of class TWebRequest.
	GetInternalScriptName (see page 15169)	This is GetInternalScriptName, a member of class TWebRequest.
	GetStringVariable (see page 15666)	Extracts a string value from the extension control block.
	UpdateMethodType (see page 15172)	Provides the abstract prototype for a method to write a pure virtual method for updating the client method type.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 15665)	Creates an instance of TISAPIRequest (see page 15662) to represent the request message information contained in the AECB parameter.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 15165)	Destroys an instance of TWebRequest (see page 15161).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExtractContentFields (see page 15166)	Parses the Content (see page 15176) property into its constituent fields.
	ExtractCookieFields (see page 15166)	Parses the Cookie (see page 15180) property into its constituent fields.
	ExtractFields (see page 15167)	Parses a multi-valued string into its constituent fields.
	ExtractQueryFields (see page 15167)	Parses the Query (see page 15186) property into its constituent fields.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetFieldByName (see page 15666)	Returns the value of the HTTP header variable specified by the Name parameter.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ReadClient (see page 15667)	Reads Count bytes from the HTTP request into Buffer.
	ReadString (see page 15668)	Reads Count bytes from the HTTP request into a string.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	TranslateURI (see page 15668)	Translates a Uniform Resource Identifier (URI) into the internal representation of a location on the server.
	WriteClient (see page 15669)	Writes Count bytes from Buffer back to the HTTP client.
	WriteHeaders (see page 15669)	Writes the header contents back to the HTTP client.

V	WriteString (see page 15670)	Writes the string AString back to the HTTP client.

Public Properties

	Name	Description
R	Accept (see page 15174)	Reports the value of the Accept header of the HTTP request message.
R	Authorization (see page 15175)	Reports the value of the Authorization header of the HTTP request message.
R	CacheControl (see page 15175)	Reports the value of the Cache-Control header of the HTTP request message.
R	Connection (see page 15176)	Reports the value of the Connection header of the HTTP request message.
R	Content (see page 15176)	Represents the unparsed contents of the HTTP request message.
R	ContentEncoding (see page 15177)	Reports the value of the Content (see page 15176)-Encoding header of the HTTP request message.
R	ContentFields (see page 15177)	Reports the parsed content of a POST request message.
R	ContentLength (see page 15178)	Reports the value of the Content (see page 15176)-Length header of the HTTP request message.
R	ContentParser (see page 15178)	Refers to the object's content parser object.
R	ContentType (see page 15179)	Reports the value of the Content (see page 15176)-Type header of the HTTP request message.
R	ContentVersion (see page 15179)	Reports the value of the Content (see page 15176)-Version header of the HTTP request message.
R	Cookie (see page 15180)	Reports the value of the Cookie header of the HTTP request message.
R	CookieFields (see page 15180)	Represents the parsed contents of the Cookie (see page 15180) header of an HTTP request message.
R	Date (see page 15181)	Reports the value of the Date header of the HTTP request message.
R	DerivedFrom (see page 15181)	Reports the value of the Derived-From (see page 15182) header of the HTTP request message.
R	ECB (see page 15670)	Indicates the extension control block structure used by the server application DLL to communicate with the Web server.
R	Expires (see page 15182)	Reports the value of the Expires header of the HTTP request message.
R	Files (see page 15182)	Provides access to the files uploaded in the HTTP request.
R	From (see page 15182)	Reports the value of the From header of the HTTP request message.
R	Host (see page 15183)	Reports the value of the Host header of the HTTP request message.
R	IfModifiedSince (see page 15183)	Reports the value of the If-Modified-Since header of the HTTP request message.
R	InternalPathInfo (see page 15184)	Returns the path info portion of the URL (see page 15190).
R	InternalScriptName (see page 15184)	Returns the ScriptName (see page 15189) part of the URL (see page 15190).
R	Method (see page 15185)	This is Method, a member of class TWebRequest.
R	MethodType (see page 15185)	This is MethodType, a member of class TWebRequest.
R	PathInfo (see page 15185)	Reports the value of the path information (if any) of the URL (see page 15190) specified in the HTTP request message.
R	PathTranslated (see page 15186)	Represents a translation of the PathInfo (see page 15185) property to a fully qualified path on the Web server.
R	ProtocolVersion (see page 15186)	Reports the version number of the HTTP protocol specified in the initial header of the HTTP request message.
R	Query (see page 15186)	Indicates the value of the query information (if any) of the URL (see page 15190) specified in the HTTP request message.
R	QueryFields (see page 15187)	Represents the parsed contents of the Query (see page 15186) in the URL (see page 15190) of the HTTP request message.
R	Referer (see page 15188)	Reports the value of the Referer header of the HTTP request message.
R	RemoteAddr (see page 15188)	Indicates the remote IP address of the client associated with the HTTP request message.
R	RemoteHost (see page 15188)	Reports the fully qualified reverse domain lookup of the client associated with the HTTP request message.
R	ScriptName (see page 15189)	Indicates the value of the script information (if any) of the URL (see page 15190) specified in the HTTP request message.
R	ServerPort (see page 15189)	Indicates the port number of the port on the Web server that receives the HTTP request message.
R	Title (see page 15190)	Reports the value of the Title header of the HTTP request message.
R	URL (see page 15190)	Reports the Uniform Resource Identifier (URI) specified by the HTTP request message to identify the target of the request.
R	UserAgent (see page 15191)	Reports the value of the User-Agent header of the HTTP request message.

86.1.1.2 TISAPIRequest.Create Constructor

Creates an instance of TISAPIRequest (see page 15662) to represent the request message information contained in the AECB parameter.

Pascal

```
constructor Create(AECB: PEXTENSION_CONTROL_BLOCK);
```

C++

```
__fastcall TISAPIRequest(PEXTENSION_CONTROL_BLOCK AECB);
```

Description

Applications don't need to call Create to instantiate an ISAPI request object. TISAPIApplication creates a TISAPIRequest (see page 15662) object whenever it receives a client request.

See Also

ECB (see page 15670)

TISAPIApplication (see page 15648)

86.1.1.3 TISAPIRequest Methods

The methods of the TISAPIRequest class are listed here.

Protected Methods

	Name	Description
	GetDateVariable (see page 15665)	Extracts a date value from the extension control block.
	GetIntegerVariable (see page 15666)	Extracts an integer value from the extension control block.
	GetStringVariable (see page 15666)	Extracts a string value from the extension control block.

Public Methods

	Name	Description
	GetFieldByName (see page 15666)	Returns the value of the HTTP header variable specified by the Name parameter.
	ReadClient (see page 15667)	Reads Count bytes from the HTTP request into Buffer.
	ReadString (see page 15668)	Reads Count bytes from the HTTP request into a string.
	TranslateURI (see page 15668)	Translates a Uniform Resource Identifier (URI) into the internal representation of a location on the server.
	WriteClient (see page 15669)	Writes Count bytes from Buffer back to the HTTP client.
	WriteHeaders (see page 15669)	Writes the header contents back to the HTTP client.
	WriteString (see page 15670)	Writes the string AString back to the HTTP client.

86.1.1.3.1 TISAPIRequest.GetDateVariable Method

Extracts a date value from the extension control block.

Pascal

```
function GetDateVariable(Index: Integer): TDateTime; override;
```

C++

```
virtual __fastcall TDateTime GetDateVariable(int Index);
```

Description

GetDateVariable reads date-time values from the extension control block specified by the ECB (see page 15670) property.

The ISAPI request object uses GetDateVariable to access the properties that represent dates from the HTTP request headers.

See Also

Date (see page 15181)

ECB (see page 15670)

Expires (see page 15182)

IfModifiedSince (see page 15183)

86.1.1.3.2 TISAPIRequest.GetFieldByName Method

Returns the value of the HTTP header variable specified by the Name parameter.

Pascal

```
function GetFieldByName(const Name: string): string; override;
```

C++

```
virtual __fastcall AnsiString GetFieldByName(const AnsiString Name);
```

Description

Use GetFieldByName to read the value of an HTTP header variable that is not surfaced as one of the properties of TISAPIRequest (see page 15662). For information on possible HTTP request header variables, see the current Internet standards documents.

See Also

ECB (see page 15670)

GetStringVariable (see page 15666)

86.1.1.3.3 TISAPIRequest.GetIntegerVariable Method

Extracts an integer value from the extension control block.

Pascal

```
function GetIntegerVariable(Index: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetIntegerVariable(int Index);
```

Description

GetIntegerVariable reads integer values from the extension control block specified by the ECB (see page 15670) property. The ISAPI request object uses GetIntegerVariable to access the properties that represent integers from the HTTP request headers.

See Also

ContentLength (see page 15178)

ECB (see page 15670)

ServerPort (see page 15189)

86.1.1.3.4 TISAPIRequest.GetStringVariable Method

Extracts a string value from the extension control block.

Pascal

```
function GetStringVariable(Index: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetStringVariable(int Index);
```

Description

GetStringVariable reads string values from the extension control block specified by the ECB (see page 15670) property. The ISAPI request object uses GetStringVariable to access the properties that represent strings from the HTTP request headers.

See Also

Accept (see page 15174)

CacheControl (see page 15175)

Connection (see page 15176)

Content (see page 15176)

ContentEncoding (see page 15177)

ContentType (see page 15179)

ContentVersion (see page 15179)

Cookie (see page 15180)

DerivedFrom (see page 15181)

ECB (see page 15670)

From (see page 15182)

Host (see page 15183)

TISAPIRequest (see page 15662)

PathInfo (see page 15185)

PathTranslated (see page 15186)

ProtocolVersion (see page 15186)

Query (see page 15186)

Referer (see page 15188)

RemoteAddr (see page 15188)

RemoteHost (see page 15188)

ScriptName (see page 15189)

Title (see page 15190)

URL (see page 15190)

UserAgent (see page 15191)

86.1.1.3.5 TISAPIRequest.ReadClient Method

Reads Count bytes from the HTTP request into Buffer.

Pascal

```
function ReadClient(var Buffer; Count: Integer): Integer; override;
```

C++

```
virtual __fastcall int ReadClient( Buffer, int Count);
```

Description

Use ReadClient to load a chunk of information from the contents of an HTTP request into a buffer. ReadClient allows the server application to get the next chunk of information when the content of the request message is too large for the Content property. ReadClient returns the number of bytes actually read. If no more content is available, ReadClient returns -1.

See Also

ReadString (see page 15668)

WriteClient (see page 15669)

86.1.1.3.6 TISAPIRequest.ReadString Method

Reads Count bytes from the HTTP request into a string.

Pascal

```
function ReadString(Count: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString ReadString(int Count);
```

Description

Use ReadString to load the next chunk of content information form the HTTP request into a string. ReadString allows the server application to get the next chunk of information when the content of the request message is too large for the Content property.

See Also

GetStringVariable (see page 15666)

ReadClient (see page 15667)

WriteString (see page 15670)

86.1.1.3.7 TISAPIRequest.TranslateURI Method

Translates a Uniform Resource Identifier (URI) into the internal representation of a location on the server.

Pascal

```
function TranslateURI(const URI: string): string; override;
```

C++

```
virtual __fastcall AnsiString TranslateURI(const AnsiString URI);
```

Description

Use TranslateURI is used to translate the URI specified in a client request into a format that can be used by the server application.

See Also

PathTranslated (see page 15186)

URL (see page 15190)

86.1.1.3.8 TISAPIRequest.WriteClient Method

Writes Count bytes from Buffer back to the HTTP client.

Pascal

```
function WriteClient(var Buffer; Count: Integer): Integer; override;
```

C++

```
virtual __fastcall int WriteClient( Buffer, int Count);
```

Description

Use WriteClient to write the contents of a response back to the Web client.

See Also

ReadClient (see page 15667)

TISAPIResponse (see page 15670)

WriteString (see page 15670)

WriteHeaders (see page 15669)

86.1.1.3.9 TISAPIRequest.WriteHeaders Method

Writes the header contents back to the HTTP client.

Pascal

```
function WriteHeaders(StatusCode: Integer; const StatusString: string; const Headers: string):
Boolean; override;
```

C++

```
virtual __fastcall Boolean WriteHeaders(int StatusCode, const AnsiString StatusString, const
AnsiString Headers);
```

Description

WriteHeaders writes the HTTP headers of a response message before a call to the WriteClient (see page 15669) method. These headers are used on the HTTP message that the WriteClient (see page 15669) method sends in response to this request.

StatusCode is the status code of the response. This informs the client of the status of the request. For a list of status code values, see the StatusCode property of TWebResponse.

StatusString is a text string that describes the status that StatusCode indicates.

Headers is a formatted string that includes any other HTTP headers to add. It represents each header as a string of the form 'Headername: Value', where name/value pairs are separated by line breaks.

WriteHeaders returns true if it succeeds in writing the specified headers.

See Also

WriteClient (see page 15669)

StatusCode (see page 15215)

86.1.1.3.10 TISAPIRequest.WriteString Method

Writes the string AString back to the HTTP client.

Pascal

```
function WriteString(const AString: string): Boolean; override;
```

C++

```
virtual __fastcall Boolean WriteString(const AnsiString AString);
```

Description

Use WriteString to write the contents of a response back to the Web client.

See Also

ReadString (see page 15668)

TISAPIResponse (see page 15670)

WriteClient (see page 15669)

86.1.1.4 TISAPIRequest Properties

The properties of the TISAPIRequest class are listed here.

Public Properties

	Name	Description
	ECB (see page 15670)	Indicates the extension control block structure used by the server application DLL to communicate with the Web server.

86.1.1.4.1 TISAPIRequest.ECB Property

Indicates the extension control block structure used by the server application DLL to communicate with the Web server.

Pascal

```
property ECB: PEXTENSION_CONTROL_BLOCK;
```

C++

```
__property PEXTENSION_CONTROL_BLOCK ECB;
```

Description

Read ECB to gain direct access to the extension control block that the server application DLL uses to represent the client request message. Most applications will not need to use the ECB property because the header fields values can be read using the GetFieldByName (see page 15666) method.

See Also

GetFieldByName (see page 15666)

86.1.2 TISAPIResponse Class

TISAPIResponse represents HTTP messages sent by an ISAPI application in response to an HTTP request message.

Class Hierarchy

System::TObject → HTTPApp.TWebResponse → IsapiHTTP.TISAPIResponse

Pascal

```
TISAPIResponse = class(TWebResponse);
```

C++

```
class TISAPIResponse : public TWebResponse;
```

File

IsapiHTTP

Description

Use TISAPIResponse to specify the response to an HTTP request message.

An ISAPI application creates a TISAPIResponse object based on the TISAPIRequest (see page 15662) object for an incoming HTTP request message. TWebDispatcher then passes the TISAPIResponse object to the TWebActionItem associated with the TISAPIRequest (see page 15662) object. The TWebActionItem sets the properties of the TISAPIResponse object.

See Also

TISAPIApplication (see page 15648)

TISAPIRequest (see page 15662)

TWebActionItem (see page 15130)

TWebDispatcher (see page 15147)

Creating HTTP Response Messages

86.1.2.1 TISAPIResponse Members

The following tables list the members exposed by TISAPIResponse.

Protected Fields

	Name	Description
	FCustomHeaders (see page 15195)	This is FCustomHeaders, a member of class TWebResponse.
	FHTTPRequest (see page 15196)	This is FHTTPRequest, a member of class TWebResponse.

Protected Methods

	Name	Description
	AddCustomHeaders (see page 15197)	Appends the custom headers specified by the CustomHeaders (see page 15211) property to the Headers parameter.
	FormatAuthenticate (see page 15197)	Lists of one or more comma-separated challenges provided when the request fails with StatusCode (see page 15215) 401 (Unauthorized).
V	GetContent (see page 15674)	Reads the value of the Content property.
V	GetDateVariable (see page 15674)	Extracts a date from the internal table of HTTP header variables.
V	GetIntegerVariable (see page 15675)	Extracts an integer value from the internal table of HTTP header variables.
V	GetLogMessage (see page 15675)	Reads the value of the LogMessage property.
V	GetStatusCode (see page 15675)	Reads the value of the StatusCode property.
V	GetStringVariable (see page 15676)	Extracts a string value from the internal table of HTTP header variables.
V	InitResponse (see page 15676)	This is InitResponse, a member of class TISAPIResponse.
V	SetContent (see page 15678)	Sets the Content property.
V	SetContentStream (see page 15203)	Sets the value of the ContentStream (see page 15209) property to the Value parameter.

	SetDateVariable (see page 15678)	Sets a date-time valued HTTP header property.
	SetIntegerVariable (see page 15679)	Sets an integer valued HTTP header property.
	SetLogMessage (see page 15679)	Sets the value of the LogMessage property.
	SetStatusCode (see page 15680)	Sets the StatusCode property.
	SetStringVariable (see page 15680)	Sets a string valued HTTP header property.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 15673)	Creates an instance of TISAPIResponse (see page 15670) to represent the response to the HTTPRequest parameter.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 15195)	Disposes of an object instance.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetCustomHeader (see page 15198)	Returns the value of the custom header with the specified name.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SendRedirect (see page 15677)	Sends the response with a StatusCode of 301 (Moved Permanently) and with Location set to the URI parameter.
	SendResponse (see page 15677)	Sends the ISAPI response message to the web client.
	SendStream (see page 15677)	Sends the ISAPI response message with the contents set to the contents of AStream.
	Sent (see page 15678)	Indicates whether the ISAPI response message has been sent.
	SetCookieField (see page 15203)	Adds a cookie header to the response message.
	SetCustomHeader (see page 15203)	Replaces the value of a specified custom header.

Public Properties

	Name	Description
	Allow (see page 15207)	Lists the HTTP request methods that the application can service.
	Content (see page 15207)	Contains the information sent to the web client in response to an HTTP request message.

	ContentEncoding (see page 15208)	Lists the mechanisms, if any, that are used to encode the content of the HTTP response message.
	ContentLength (see page 15208)	Specifies the length, in bytes, of the Content (see page 15207) property.
	ContentStream (see page 15209)	Specifies the stream object for writing the contents of the HTTP response message to the web client.
	ContentType (see page 15209)	Indicates the media type of the Content (see page 15207) of the HTTP response message.
	ContentVersion (see page 15210)	Represents the version associated with the Content (see page 15207) of the HTTP response message.
R	Cookies (see page 15210)	Lists all cookie headers that should be sent with the response.
	CustomHeaders (see page 15211)	Lists message header strings that are added to the response message before it is sent.
	Date (see page 15211)	Indicates the time and date when the HTTP response message is sent to the web client.
	DerivedFrom (see page 15212)	Specifies the URI of the resource where the Content (see page 15207) of the HTTP response originated.
	Expires (see page 15212)	Specifies the date and time after which the Content (see page 15207) of the HTTP response message should be considered invalid.
R	HTTPRequest (see page 15213)	Represents the HTTP request message for which this is a response.
	LastModified (see page 15213)	Indicates the date and time that the value of the Content (see page 15207) property was last updated.
	Location (see page 15214)	Specifies the URI of a resource the web client should point to for the completion of its request.
	LogMessage (see page 15214)	Describes the HTTP response message for event logging purposes.
	Realm (see page 15214)	Specifies the set of protected-access URIs that includes the URL specified by the HTTP request message.
	ReasonString (see page 15215)	Indicates the status of the response to the web client request as a human-readable string.
	Server (see page 15215)	Provides information about the software used by the server application to handle the request.
	StatusCode (see page 15215)	Indicates the status of the response to the HTTP request message as a 3-digit integer.
	Title (see page 15217)	Specifies a title for the Content (see page 15207) of the HTTP response message.
	Version (see page 15217)	Specifies the HTTP protocol version number that the web client is using.
	WWWAuthenticate (see page 15218)	Lists of one or more comma-separated challenges provided when the request fails with StatusCode (see page 15215) 401 (Unauthorized).

86.1.2.2 TISAPIResponse.Create Constructor

Creates an instance of TISAPIResponse (see page 15670) to represent the response to the HTTPRequest parameter.

Pascal

```
constructor Create(HTTPRequest: TWebRequest);
```

C++

```
__fastcall TISAPIResponse(TWebRequest HTTPRequest);
```

Description

Applications don't need to call Create to instantiate a ISAPI response object. TISAPIApplication creates a TISAPIResponse (see page 15670) for each TISAPIRequest (see page 15662) it creates.

See Also

TISAPIApplication (see page 15648)

TISAPIRequest (see page 15662)

86.1.2.3 TISAPIResponse Methods

The methods of the TISAPIResponse class are listed here.

Protected Methods

	Name	Description
	GetContent (see page 15674)	Reads the value of the Content property.
	GetDateVariable (see page 15674)	Extracts a date from the internal table of HTTP header variables.
	GetIntegerVariable (see page 15675)	Extracts an integer value from the internal table of HTTP header variables.
	GetLogMessage (see page 15675)	Reads the value of the LogMessage property.
	GetStatusCode (see page 15675)	Reads the value of the StatusCode property.
	GetStringVariable (see page 15676)	Extracts a string value from the internal table of HTTP header variables.
	InitResponse (see page 15676)	This is InitResponse, a member of class TISAPIResponse.
	SetContent (see page 15678)	Sets the Content property.
	SetDateVariable (see page 15678)	Sets a date-time valued HTTP header property.
	SetIntegerVariable (see page 15679)	Sets an integer valued HTTP header property.
	SetLogMessage (see page 15679)	Sets the value of the LogMessage property.
	SetStatusCode (see page 15680)	Sets the StatusCode property.
	SetStringVariable (see page 15680)	Sets a string valued HTTP header property.

Public Methods

	Name	Description
	SendRedirect (see page 15677)	Sends the response with a StatusCode of 301 (Moved Permanently) and with Location set to the URI parameter.
	SendResponse (see page 15677)	Sends the ISAPI response message to the web client.
	SendStream (see page 15677)	Sends the ISAPI response message with the contents set to the contents of AStream.
	Sent (see page 15678)	Indicates whether the ISAPI response message has been sent.

86.1.2.3.1 TISAPIResponse.GetContent Method

Reads the value of the Content property.

Pascal

```
function GetContent: string; override;
```

C++

```
virtual __fastcall AnsiString GetContent();
```

Description

GetContent is the protected access function for the Content property.

See Also

Content (see page 15207)

SetContent (see page 15678)

86.1.2.3.2 TISAPIResponse.GetDateVariable Method

Extracts a date from the internal table of HTTP header variables.

Pascal

```
function GetDateVariable(Index: Integer): TDateTime; override;
```

C++

```
virtual __fastcall TDateTime GetDateVariable(int Index);
```

Description

The ISAPI response object uses GetDateVariable to access the properties that represent dates.

See Also

Date (see page 15211)

Expires (see page 15212)

LastModified (see page 15213)

SetDateVariable (see page 15678)

86.1.2.3.3 TISAPIResponse.GetIntegerVariable Method

Extracts an integer value from the internal table of HTTP header variables.

Pascal

```
function GetIntegerVariable(Index: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetIntegerVariable(int Index);
```

Description

The ISAPI response object uses GetIntegerVariable to access the properties that represent integers.

See Also

ContentLength (see page 15208)

SetIntegerVariable (see page 15679)

86.1.2.3.4 TISAPIResponse.GetLogMessage Method

Reads the value of the LogMessage property.

Pascal

```
function GetLogMessage: string; override;
```

C++

```
virtual __fastcall AnsiString GetLogMessage();
```

Description

GetLogMessage is the protected access function for LogMessage property.

See Also

LogMessage (see page 15214)

SetLogMessage (see page 15679)

86.1.2.3.5 TISAPIResponse.GetStatusCode Method

Reads the value of the StatusCode property.

Pascal

```
function GetStatusCode: Integer; override;
```

C++

```
virtual __fastcall int GetStatusCode();
```

Description

GetStatusCode is the protected access function for the StatusCode property.

See Also

StatusCode (see page 15215)

SetStatusCode (see page 15205)

86.1.2.3.6 TISAPIResponse.GetStringVariable Method

Extracts a string value from the internal table of HTTP header variables.

Pascal

```
function GetStringVariable(Index: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetStringVariable(int Index);
```

Description

The ISAPI response object uses GetStringVariable to access the properties that represent strings.

See Also

Allow (see page 15207)

ContentEncoding (see page 15208)

ContentType (see page 15209)

ContentVersion (see page 15210)

DerivedFrom (see page 15212)

Location (see page 15214)

Realm (see page 15214)

ReasonString (see page 15215)

Server (see page 15215)

SetStringVariable (see page 15680)

Title (see page 15217)

Version (see page 15217)

WWWAuthenticate (see page 15218)

86.1.2.3.7 TISAPIResponse.InitResponse Method

Pascal

```
procedure InitResponse; virtual;
```

C++

```
virtual __fastcall InitResponse();
```

Description

This is InitResponse, a member of class TISAPIResponse.

86.1.2.3.8 TISAPIResponse.SendRedirect Method

Sends the response with a StatusCode of 301 (Moved Permanently) and with Location set to the URI parameter.

Pascal

```
procedure SendRedirect(const URI: string); override;
```

C++

```
virtual __fastcall SendRedirect(const AnsiString URI);
```

Description

Call SendRedirect to redirect the web client request to another URI that is the proper target of the request. SendRedirect does the same thing as setting the StatusCode to 301, the Location to URI, and calling SendResponse (see page 15677).

See Also

Location (see page 15214)

SendResponse (see page 15677)

StatusCode (see page 15215)

86.1.2.3.9 TISAPIResponse.SendResponse Method

Sends the ISAPI response message to the web client.

Pascal

```
procedure SendResponse; override;
```

C++

```
virtual __fastcall SendResponse();
```

Description

Call SendResponse after setting the properties of the HTTP response object to specify the response that should be made to the web client. If the request should be redirected to another URI, it is more efficient to call SendRedirect (see page 15677).

See Also

SendRedirect (see page 15677)

86.1.2.3.10 TISAPIResponse.SendStream Method

Sends the ISAPI response message with the contents set to the contents of AStream.

Pascal

```
procedure SendStream(AStream: TStream); override;
```

C++

```
virtual __fastcall SendStream(TStream AStream);
```

Description

Only call SendStream after first calling SendResponse (see page 15677). Unlike SendStream, SendResponse (see page 15677) fills the header of the HTTP response message, based on the properties of the TISAPIResponse (see page 15670) object. If SendResponse (see page 15677) has not been called first to set the header of the HTTP response message, set the ContentStream property to AStream, and call SendResponse (see page 15677).

See Also

ContentStream (see page 15209)

SendResponse (see page 15677)

Sent (see page 15678)

86.1.2.3.11 TISAPIResponse.Sent Method

Indicates whether the ISAPI response message has been sent.

Pascal

```
function Sent: Boolean; override;
```

C++

```
virtual __fastcall Boolean Sent();
```

Description

Sent returns true if the SendRedirect (see page 15677) or SendStream (see page 15677) method has been called. TISAPIApplication objects read Sent to determine whether they should send a response to the web client.

See Also

SendRedirect (see page 15677)

SendStream (see page 15677)

86.1.2.3.12 TISAPIResponse.SetContent Method

Sets the Content property.

Pascal

```
procedure SetContent(const Value: string); override;
```

C++

```
virtual __fastcall SetContent(const AnsiString Value);
```

Description

SetContent is the protected method used by ISAPI response objects to set the Content property.

See Also

Content (see page 15207)

GetContent (see page 15674)

86.1.2.3.13 TISAPIResponse.SetDateVariable Method

Sets a date-time valued HTTP header property.

Pascal

```
procedure SetDateVariable(Index: Integer; const Value: TDateTime); override;
```

C++

```
virtual __fastcall SetDateVariable(int Index, const TDateTime Value);
```

Description

SetDateVariable writes date-time to the internal table used to store HTTP header variables. The ISAPI response object uses SetDateVariable to set the properties that represent dates.

See Also

Date (see page 15211)

Expires (see page 15212)

GetDateVariable (see page 15674)

LastModified (see page 15213)

86.1.2.3.14 TISAPIResponse.SetIntegerVariable Method

Sets an integer valued HTTP header property.

Pascal

```
procedure SetIntegerVariable(Index: Integer; Value: Integer); override;
```

C++

```
virtual __fastcall SetIntegerVariable(int Index, int Value);
```

Description

SetIntegerVariable writes integer values to the internal table used to store HTTP header variables. The ISAPI response object uses SetIntegerVariable to set the properties that represent integers.

See Also

ContentLength (see page 15208)

GetIntegerVariable (see page 15675)

86.1.2.3.15 TISAPIResponse.SetLogMessage Method

Sets the value of the LogMessage property.

Pascal

```
procedure SetLogMessage(const Value: string); override;
```

C++

```
virtual __fastcall SetLogMessage(const AnsiString Value);
```

Description

SetLogMessage is the protected method used by ISAPI response objects to set the LogMessage property.

See Also

GetLogMessage (see page 15675)

LogMessage (see page 15214)

86.1.2.3.16 TISAPIResponse.SetStatusCode Method

Sets the StatusCode property.

Pascal

```
procedure SetStatusCode(Value: Integer); override;
```

C++

```
virtual __fastcall SetStatusCode(int Value);
```

Description

SetStatusCode is the protected method used by ISAPI response objects to set the StatusCode property.

See Also

StatusCode (see page 15215)

GetStatusCode (see page 15199)

86.1.2.3.17 TISAPIResponse.SetStringVariable Method

Sets a string valued HTTP header property.

Pascal

```
procedure SetStringVariable(Index: Integer; const Value: string); override;
```

C++

```
virtual __fastcall SetStringVariable(int Index, const AnsiString Value);
```

Description

SetStringVariable writes string values to the internal table used to store HTTP header variables. The ISAPI response object uses SetStringVariable to set the properties that represent strings.

See Also

Allow (see page 15207)

ContentEncoding (see page 15208)

ContentType (see page 15209)

ContentVersion (see page 15210)

DerivedFrom (see page 15212)

GetStringVariable (see page 15676)

Location (see page 15214)

Realm (see page 15214)

ReasonString (see page 15215)

Server (see page 15215)

Title (see page 15217)

Version (see page 15217)

WWWAuthenticate (see page 15218)

87 Java Namespace

This is namespace Java.

Namespaces

Name	Description
Sql (see page 15682)	This is namespace Java.Sql.

87.1 Java.Sql Namespace

This is namespace Java.Sql.

Topics

Name	Description
Classes (see page 15682)	The following table lists classes in this documentation.

Classes

	Name	Description
	SQLException (see page 15682)	This is class Java.Sql.SQLException.

87.1.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	SQLException (see page 15682)	This is class Java.Sql.SQLException.

87.1.1.1 SQLException Class

Class Hierarchy

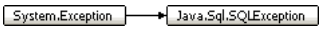

Pascal

```
SQLException = class(System.Exception);
```

C++

```
class SQLException : public System.Exception;
```

File

SQLException.cs

Description

This is class Java.Sql.SQLException.

87.1.1.1.1 SQLException Members

The following tables list the members exposed by SQLException.

Internal Methods

	Name	Description
	getErrorCode (see page 15684)	This is getErrorCode, a member of class SQLException.
	getSQLState (see page 15684)	This is getSQLState, a member of class SQLException.

Public Methods

	Name	Description
	SQLException (see page 15683)	This is the overview for the SQLException constructor overload.

Public Properties

	Name	Description
	ErrorCode (see page 15684)	This is ErrorCode, a member of class SQLException.
	SQLState (see page 15684)	This is SQLState, a member of class SQLException.

87.1.1.1.2 SQLException Constructor

This is the overview for the SQLException constructor overload.

Overload List

	Name	Description
	SQLException.SQLException (string) (see page 15683)	This is SQLException, a member of class SQLException.
	SQLException.SQLException (string, string, int, Exception) (see page 15683)	This is SQLException, a member of class SQLException.

87.1.1.1.2.1 SQLException.SQLException Constructor (string)

Pascal

```
constructor Create(reason: string);
```

C++

```
SQLException(string reason);
```

Description

This is SQLException, a member of class SQLException.

87.1.1.1.2.2 SQLException.SQLException Constructor (string, string, int, Exception)

Pascal

```
constructor Create(reason: string; SQLState: string; vendorCode: Integer; cause: Exception);
```

C++

```
SQLException(string reason, string SQLState, int vendorCode, Exception cause);
```

Description

This is SQLException, a member of class SQLException.

87.1.1.1.3 SQLException Methods

The methods of the SQLException class are listed here.

Internal Methods

	Name	Description
	getErrorCode (see page 15684)	This is getErrorCode, a member of class SQLException.
	getSQLState (see page 15684)	This is getSQLState, a member of class SQLException.

87.1.1.1.3.1 SQLException.getErrorCode Method

Pascal

```
function getErrorCode: Integer;
```

C++

```
int getErrorCode();
```

Description

This is getErrorCode, a member of class SQLException.

87.1.1.1.3.2 SQLException.getSQLState Method

Pascal

```
function getSQLState: string;
```

C++

```
string getSQLState();
```

Description

This is getSQLState, a member of class SQLException.

87.1.1.1.4 SQLException Properties

The properties of the SQLException class are listed here.

Public Properties

	Name	Description
	ErrorCode (see page 15684)	This is ErrorCode, a member of class SQLException.
	SQLState (see page 15684)	This is SQLState, a member of class SQLException.

87.1.1.1.4.1 SQLException.ErrorCode Property

Pascal

```
property ErrorCode: Integer;
```

C++

```
__property int ErrorCode;
```

Description

This is ErrorCode, a member of class SQLException.

87.1.1.1.4.2 SQLException.SQLState Property

Pascal

```
property SQLState: string;
```

C++

```
__property string SQLState;
```

Description

This is SQLState, a member of class SQLException.

88 jpeg Namespace

This is namespace jpeg.

Topics

Name	Description
Classes (see page 15686)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 15706)	The following table lists structs, records, enums in this documentation.
Types (see page 15709)	The following table lists types in this documentation.
Variables (see page 15710)	The following table lists variables in this documentation.

Classes

	Name	Description
	TJPEGData (see page 15686)	TJPEGData encapsulates the compressed file that contains the data source for the original compressed jpeg image used by a TJPEGImage (see page 15688) object.
	TJPEGImage (see page 15688)	TJPEGImage encapsulates a jpeg graphic.

Structs, Records, Enums

	Name	Description
	TJPEGDefaults (see page 15706)	This is record jpeg.TJPEGDefaults.
	TJPEGPerformance (see page 15707)	TJPEGPerformance represents the type of optimization to use when displaying a JPEG image.
	TJPEGPixelFormat (see page 15707)	TJPEGPixelFormat indicates the format used to display a JPEG image.
	TJPEGScale (see page 15708)	TJPEGScale represents the display size of a JPEG image.

Types

Name	Description
TJPEGQualityRange (see page 15709)	TJPEGQualityRange represents the trade-off ratio between image quality and file size in a JPEG image.

Variables

Name	Description
JPEGDefaults (see page 15710)	This is variable jpeg.JPEGDefaults.

88.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TJPEGData (see page 15686)	TJPEGData encapsulates the compressed file that contains the data source for the original compressed jpeg (see page 15685) image used by a TJPEGImage (see page 15688) object.
	TJPEGImage (see page 15688)	TJPEGImage encapsulates a jpeg (see page 15685) graphic.

88.1.1 TJPEGData Class

TJPEGData encapsulates the compressed file that contains the data source for the original compressed jpeg (see page 15685) image used by a TJPEGImage (see page 15688) object.

Class Hierarchy

Pascal

```
TJPEGData = class(TSharedImage);
```

C++

```
class TJPEGData : public TSharedImage;
```

File

jpeg

Description

TJPEGData is used for the internal implementation of TJPEGImage (see page 15688). TJPEGData is created automatically by the TJPEGImage (see page 15688) object when that image object is created.

TJPEGData owns the file handle to the jpeg (see page 15685) data source. Thus, TJPEGData implements how TJPEGImage (see page 15688) does reference counting and handle sharing. When a jpeg (see page 15685) image object is copied using the Assign method, those multiple instances of the image object refer to the same TJPEGData object.

The TJPEGData file itself is never modified. Each TJPEGImage (see page 15688) instance that shares it has its own properties that determine how this data is decompressed into its own internal, local bitmap representation. However, if a TJPEGImage (see page 15688) needs to modify the actual jpeg (see page 15685) data, it breaks its link to that particular TJPEGData instance and creates a new one.

See Also

TJPEGImage (see page 15688)

TJPEGImage (see page 15688)

88.1.1.1 TJPEGData Members

The following tables list the members exposed by TJPEGData.

Protected Methods

	Name	Description
	FreeHandle (see page 15687)	This is FreeHandle, a member of class TJPEGData.
	Reference (see page 14604)	This is Reference, a member of class TSharedImage.
	Release (see page 14604)	This is Release, a member of class TSharedImage.

Protected Properties

	Name	Description
	RefCount (see page 14605)	This is RefCount, a member of class TSharedImage.

Public Methods

	Name	Description
	Destroy (see page 15687)	Destroys of an instance of a jpeg (see page 15685) data object.

88.1.1.2 TJPEGData.Destroy Destructor

Destroys of an instance of a jpeg (see page 15685) data object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TJPEGData();
```

Description

The jpeg (see page 15685) data object is automatically destroyed by the jpeg (see page 15685) image object that uses it. It is not necessary to call Destroy or to directly access the jpeg (see page 15685) data object.

The Destroy method of a jpeg (see page 15685) data object handles the freeing of the data.

See Also

Free (see page 22300)

Destroy (see page 22287)

88.1.1.3 TJPEGData Methods

The methods of the TJPEGData class are listed here.

Protected Methods

	Name	Description
	FreeHandle (see page 15687)	This is FreeHandle, a member of class TJPEGData.

88.1.1.3.1 TJPEGData.FreeHandle Method

Pascal

```
procedure FreeHandle; override;
```

C++

```
virtual __fastcall FreeHandle();
```

Description

This is FreeHandle, a member of class TJPEGData.

88.1.2 TJPEGImage Class

TJPEGImage encapsulates a jpeg (see page 15685) graphic.

Class Hierarchy

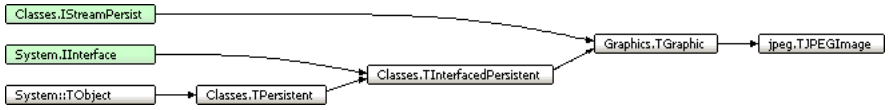

Pascal

```
TJPEGImage = class(TGraphic);
```

C++

```
class TJPEGImage : public TGraphic;
```

File

jpeg

Description

Use TJPEGImage to read and write jpeg (see page 15685) compressed image data. TJPEGImage handles the digital compression and decompression of still images for use in computer systems. It uses the data from an instance of TJPEGData (see page 15686), which contains the actual jpeg (see page 15685) data source and is never modified. Each jpeg (see page 15685) image object may share its TJPEGData (see page 15686) object with other instances of a jpeg (see page 15685) image by creating copies using the Assign (see page 15693) method. The jpeg (see page 15685) data source handles reference counting for the jpeg (see page 15685) image objects that are linked to it.

TJPEGImage has an internal bitmap that represents the jpeg (see page 15685) image. This internal image and the original source of the jpeg (see page 15685) image are read only. TJPEGImage has properties that determine how each instance will handle color conversion, compression, decompression, performance, and so on.

The following are characteristics of this object. A TJPEGImage object:

Has no canvas (so it cannot draw onto a canvas). However, TJPEGImage implements the protected Draw (see page 15696) method introduced in TGraphic, so it can draw itself on the canvas of another object.

Provides no access to the internal bitmap image that it creates for the JPEG image.

Performs reference counting and handle sharing by means of the TJPEGData (see page 15686) object. Multiple instances can refer to the same TJPEGData (see page 15686) image. TJPEGData (see page 15686) is the actual owner of the file handle to the jpeg (see page 15685) data source.

See Also

TJPEGData (see page 15686)

TBitmap (see page 14396)

88.1.2.1 TJPEGImage Members

The following tables list the members exposed by TJPEGImage.

Protected Methods

	Name	Description
	_AddRef (see page 2853)	Increments the reference count for the interfaced object's interface.
	_Release (see page 2854)	Decrements the reference count for the interfaced object's interface.
V	AssignTo (see page 15694)	Copies the properties of an object to a destination object.
V	Changed (see page 15695)	This is Changed, a member of class TJPEGImage.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 14520)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Draw (see page 15696)	This is Draw, a member of class TJPEGImage.
V	Equals (see page 15696)	This is Equals, a member of class TJPEGImage.
	FreeBitmap (see page 15696)	This is FreeBitmap, a member of class TJPEGImage.
V	GetEmpty (see page 15697)	This is GetEmpty, a member of class TJPEGImage.
V	GetHeight (see page 15697)	This is GetHeight, a member of class TJPEGImage.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetPalette (see page 15697)	This is GetPalette, a member of class TJPEGImage.
V	GetTransparent (see page 14522)	This is GetTransparent, a member of class TGraphic.
V	GetWidth (see page 15697)	This is GetWidth, a member of class TJPEGImage.
	NewBitmap (see page 15699)	This is NewBitmap, a member of class TJPEGImage.
	NewImage (see page 15699)	This is NewImage, a member of class TJPEGImage.
	Progress (see page 14524)	This is Progress, a member of class TGraphic.
V	ReadData (see page 15699)	This is ReadData, a member of class TJPEGImage.
	ReadStream (see page 15699)	This is ReadStream, a member of class TJPEGImage.
V	SetHeight (see page 15700)	This is SetHeight, a member of class TJPEGImage.
V	SetPalette (see page 15701)	This is SetPalette, a member of class TJPEGImage.
V	SetTransparent (see page 14527)	This is SetTransparent, a member of class TGraphic.
V	SetWidth (see page 15701)	This is SetWidth, a member of class TJPEGImage.
V	WriteData (see page 15701)	This is WriteData, a member of class TJPEGImage.

Protected Properties

	Name	Description
R	Bitmap (see page 15702)	This is Bitmap, a member of class TJPEGImage.

Public Events

	Name	Description
	OnChange (see page 14531)	Occurs whenever a graphical image changes.
	OnProgress (see page 14531)	Occurs when a graphical image is in the process of changing.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 2854)	Obtains a reference to the Owner's interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 15693)	Copies the jpeg (see page 15685) image object and creates a new reference to the internal data source object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Compress (see page 15695)	Forces the action of compression.
V	Create (see page 15691)	Instantiates a JPEG image object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 15692)	Destroys of an instance of a jpeg (see page 15685) image object.
	DIBNeeded (see page 15696)	Decompress the jpeg (see page 15685) image into a bitmap.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	JPEGNeeded (see page 15697)	Creates a new TJPEGData (see page 15686) object if the jpeg (see page 15685) image only has a bitmap and a jpeg (see page 15685) is needed.
V	LoadFromClipboardFormat (see page 15698)	Reads in a jpeg (see page 15685) image from the Clipboard into the jpeg (see page 15685) image object.
V	LoadFromFile (see page 14523)	Loads a graphic image stored in a file.
V	LoadFromStream (see page 15698)	Loads the jpeg (see page 15685) image from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	QueryInterface (see page 2854)	Returns a reference to a specified interface if the object supports that interface.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SaveToClipboardFormat (see page 15700)	Writes a jpeg (see page 15685) image to a variable in Clipboard format.
V	SaveToFile (see page 14525)	Saves a graphics image to a file.
V	SaveToStream (see page 15700)	Writes the jpeg (see page 15685) data to a stream.
V	SetSize (see page 14527)	Specifies the size of the graphic.

Public Properties

	Name	Description
	CompressionQuality (see page 15702)	Indicates the trade-off ratio between the image quality and the file size.
R	Empty (see page 14528)	Indicates whether the graphics object contains a graphic.
	Grayscale (see page 15702)	Determines whether the image output of a JPEG image is black and white, or color.
	Height (see page 14528)	Specifies the vertical size of the graphic in pixels.
	Modified (see page 14528)	Indicates whether the graphics object has been changed or edited.
	Palette (see page 14529)	Indicates the color palette of the graphical image.
	PaletteModified (see page 14529)	Indicates whether the palette has changed.
	Performance (see page 15703)	Controls (see page 5711) the trade-off between color quality and speed of decompression.
	PixelFormat (see page 15703)	Determines whether the JPEG image is displayed in 24 or 8-bit format.

	ProgressiveDisplay (see page 15703)	Controls (see page 5711) how the image is displayed when decompressing files.
	ProgressiveEncoding (see page 15704)	Determines whether an image can be progressively displayed when it is decompressed.
	Scale (see page 15704)	Determines the size of the JPEG image when it is displayed.
	Smoothing (see page 15704)	Determines whether the jpeg (see page 15685) image displays in blocks or with blurred edges.
	Transparent (see page 14529)	Indicates whether the image covers its rectangular area.
	Width (see page 14530)	Determines the maximum width of the graphics object in pixels.

88.1.2.2 TJPEGImage.Create Constructor

Instantiates a JPEG image object.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TJPEGImage();
```

Description

Call Create to instantiate a jpeg (see page 15685) image object at runtime.

Create allocates memory for a jpeg (see page 15685) image object, and then initializes its properties.

After the jpeg (see page 15685) is loaded from a file, an internal bitmap is created of the image.

C++ Examples:

```
/*
This example converts a bitmap image to a jpeg file by using
the Assign method.  This example requires a button and two
images.  One image must have it's Picture property loaded
with a bitmap at design time.
*/

#include <Jpeg.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  //Requires "jpeg.hpp" to be included in the source file
  TJPEGImage *jp = new TJPEGImage();
  try
  {
    jp->Assign(Image1->Picture->Bitmap);
    jp->SaveToFile("..\\factory.jpg");
    Image2->Stretch = True;
    Image2->Picture->LoadFromFile("..\\factory.jpg");
  }
  __finally
  {
    delete jp;
  }
}
```

Delphi Examples:

```
{
This example converts a bitmap image to a jpeg file by using
the Assign method.  This example requires a button and two
images.  One image must have it's Picture property loaded
with a bitmap at design time.
}
```

```
uses Jpeg;

procedure TForm1.Button1Click(Sender: TObject);
var
  jp: TJPEGImage;  //Requires the "jpeg" unit added to "uses" clause.
begin
  jp := TJPEGImage.Create;
  try
    with jp do
    begin
      Assign(Image1.Picture.Bitmap);
      SaveToFile('factory.jpg');
      Image2.Stretch := True;
      Image2.Picture.LoadFromFile('factory.jpg');
    end;
  finally
    jp.Free;
  end;
end;
```

See Also

Create (see page 22287)

88.1.2.3 TJPEGImage.Destroy Destructor

Destroys of an instance of a jpeg (see page 15685) image object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TJPEGImage();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TJPEGImage (see page 15688) reference is not nil before it calls Destroy.

Destroy handles the freeing of the internal bitmap image and decrements the reference count on the jpeg (see page 15685) data source to which it was linked.

See Also

Free (see page 22300)

TJPEGData (see page 15686)

Destroy (see page 22287)

88.1.2.4 TJPEGImage Methods

The methods of the TJPEGImage class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 15694)	Copies the properties of an object to a destination object.
	Changed (see page 15695)	This is Changed, a member of class TJPEGImage.
	Draw (see page 15696)	This is Draw, a member of class TJPEGImage.
	Equals (see page 15696)	This is Equals, a member of class TJPEGImage.

	FreeBitmap (see page 15696)	This is FreeBitmap, a member of class TJPEGImage.
V	GetEmpty (see page 15697)	This is GetEmpty, a member of class TJPEGImage.
V	GetHeight (see page 15697)	This is GetHeight, a member of class TJPEGImage.
V	GetPalette (see page 15697)	This is GetPalette, a member of class TJPEGImage.
V	GetWidth (see page 15697)	This is GetWidth, a member of class TJPEGImage.
	NewBitmap (see page 15699)	This is NewBitmap, a member of class TJPEGImage.
	NewImage (see page 15699)	This is NewImage, a member of class TJPEGImage.
V	ReadData (see page 15699)	This is ReadData, a member of class TJPEGImage.
	ReadStream (see page 15699)	This is ReadStream, a member of class TJPEGImage.
V	SetHeight (see page 15700)	This is SetHeight, a member of class TJPEGImage.
V	SetPalette (see page 15701)	This is SetPalette, a member of class TJPEGImage.
V	SetWidth (see page 15701)	This is SetWidth, a member of class TJPEGImage.
V	WriteData (see page 15701)	This is WriteData, a member of class TJPEGImage.

Public Methods

	Name	Description
V	Assign (see page 15693)	Copies the jpeg (see page 15685) image object and creates a new reference to the internal data source object.
	Compress (see page 15695)	Forces the action of compression.
	DIBNeeded (see page 15696)	Decompress the jpeg (see page 15685) image into a bitmap.
	JPEGNeeded (see page 15697)	Creates a new TJPEGData (see page 15686) object if the jpeg (see page 15685) image only has a bitmap and a jpeg (see page 15685) is needed.
V	LoadFromClipboardFormat (see page 15698)	Reads in a jpeg (see page 15685) image from the Clipboard into the jpeg (see page 15685) image object.
V	LoadFromStream (see page 15698)	Loads the jpeg (see page 15685) image from a stream.
V	SaveToClipboardFormat (see page 15700)	Writes a jpeg (see page 15685) image to a variable in Clipboard format.
V	SaveToStream (see page 15700)	Writes the jpeg (see page 15685) data to a stream.

88.1.2.4.1 TJPEGImage.Assign Method

Copies the jpeg (see page 15685) image object and creates a new reference to the internal data source object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Assign copies the jpeg (see page 15685) image contained in Source to the jpeg (see page 15685) image object. Assign then calls the inherited Assign. Assign creates a new link to the internal jpeg (see page 15685) data source by which the jpeg (see page 15685) image objects do reference counting and handle sharing.

Tip: To draw on a jpeg (see page 15685) image, Assign the jpeg (see page 15685) image to a bitmap, and then draw on that.

Note: An object of one type can always be assigned to another object of the same type. Also, the Source can be of type TPicture if the Graphic property of the picture is a jpeg (see page 15685) image or a bitmap.

C++ Examples:

```
/*
This example converts a bitmap image to a jpeg file by using
the Assign method.  This example requires a button and two
```

```
images.  One image must have it's Picture property loaded
with a bitmap at design time.
*/

#include <Jpeg.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  //Requires "jpeg.hpp" to be included in the source file
  TJPEGImage *jp = new TJPEGImage();
  try
  {
    jp->Assign(Image1->Picture->Bitmap);
    jp->SaveToFile("..\\factory.jpg");
    Image2->Stretch = True;
    Image2->Picture->LoadFromFile("..\\factory.jpg");
  }
  __finally
  {
    delete jp;
  }
}
```

Delphi Examples:

```
{
This example converts a bitmap image to a jpeg file by using
the Assign method.  This example requires a button and two
images.  One image must have it's Picture property loaded
with a bitmap at design time.
}
uses Jpeg;

procedure TForm1.Button1Click(Sender: TObject);
var
  jp: TJPEGImage;  //Requires the "jpeg" unit added to "uses" clause.
begin
  jp := TJPEGImage.Create;
  try
    with jp do
    begin
      Assign(Image1.Picture.Bitmap);
      SaveToFile('factory.jpg');
      Image2.Stretch := True;
      Image2.Picture.LoadFromFile('factory.jpg');
    end;
  finally
    jp.Free;
  end;
end;
```

See Also

TJPEGData (see page 15686)

Assign (see page 2930)

Graphic (see page 14600)

88.1.2.4.2 TJPEGImage.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign (see page 2930)

EConvertError (see page 22724)

88.1.2.4.3 TJPEGImage.Changed Method

Pascal

```
procedure Changed(Sender: TObject); override;
```

C++

```
virtual __fastcall Changed(TObject * Sender);
```

Description

This is Changed, a member of class TJPEGImage.

88.1.2.4.4 TJPEGImage.Compress Method

Forces the action of compression.

Pascal

```
procedure Compress;
```

C++

```
__fastcall Compress();
```

Description

Call Compress to compress for optimization, such as before streaming out when assigning a bitmap to a jpeg (see page 15685). Changing properties like CompressionQuality (see page 15702) does not force compression. To force compression, call Compress.

See Also

CompressionQuality (see page 15702)

88.1.2.4.5 TJPEGImage.DIBNeeded Method

Decompress the jpeg (see page 15685) image into a bitmap.

Pascal

```
procedure DIBNeeded;
```

C++

```
__fastcall DIBNeeded();
```

Description

DIBNeeded is used when the jpeg (see page 15685) image needs a bitmap representation of its image. Call DIBNeeded just before drawing rather than while drawing for optimization.

See Also

JPEGNeeded (see page 15697)

88.1.2.4.6 TJPEGImage.Draw Method

Pascal

```
procedure Draw(ACanvas: TCanvas; const Rect: TRect); override;
```

C++

```
virtual __fastcall Draw(TCanvas ACanvas, const TRect Rect);
```

Description

This is Draw, a member of class TJPEGImage.

88.1.2.4.7 TJPEGImage.Equals Method

Pascal

```
function Equals(Graphic: TGraphic): Boolean; override;
```

C++

```
virtual __fastcall Boolean Equals(TGraphic Graphic);
```

Description

This is Equals, a member of class TJPEGImage.

88.1.2.4.8 TJPEGImage.FreeBitmap Method

Pascal

```
procedure FreeBitmap;
```

C++

```
__fastcall FreeBitmap();
```

Description

This is FreeBitmap, a member of class TJPEGImage.

88.1.2.4.9 TJPEGImage.GetEmpty Method

Pascal

```
function GetEmpty: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetEmpty();
```

Description

This is GetEmpty, a member of class TJPEGImage.

88.1.2.4.10 TJPEGImage.GetHeight Method

Pascal

```
function GetHeight: Integer; override;
```

C++

```
virtual __fastcall int GetHeight();
```

Description

This is GetHeight, a member of class TJPEGImage.

88.1.2.4.11 TJPEGImage.GetPalette Method

Pascal

```
function GetPalette: HPALETTE; override;
```

C++

```
virtual __fastcall HPALETTE GetPalette();
```

Description

This is GetPalette, a member of class TJPEGImage.

88.1.2.4.12 TJPEGImage.GetWidth Method

Pascal

```
function GetWidth: Integer; override;
```

C++

```
virtual __fastcall int GetWidth();
```

Description

This is GetWidth, a member of class TJPEGImage.

88.1.2.4.13 TJPEGImage.JPEGNeeded Method

Creates a new TJPEGData (see page 15686) object if the jpeg (see page 15685) image only has a bitmap and a jpeg (

see page 15685) is needed.

Pascal

```
procedure JPEGNeeded;
```

C++

```
__fastcall JPEGNeeded();
```

Description

Call JPEGNeeded to create a new jpeg (see page 15685) data source when the jpeg (see page 15685) image has thrown away the jpeg (see page 15685) data and only has an internal bitmap. JPEGNeeded creates a new jpeg (see page 15685) data source. The SaveToStream (see page 15700) method uses JPEGNeeded.

See Also

SaveToStream (see page 15700)

TJPEGData (see page 15686)

88.1.2.4.14 TJPEGImage.LoadFromClipboardFormat Method

Reads in a jpeg (see page 15685) image from the Clipboard into the jpeg (see page 15685) image object.

Pascal

```
procedure LoadFromClipboardFormat(AFormat: Word; AData: THandle; APalette: HPALETTE); override;
```

C++

```
virtual __fastcall LoadFromClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette);
```

Description

LoadFromClipboardFormat is called if the jpeg (see page 15685) image is registered with the TPicture object using the RegisterClipboardFormat method. LoadFromClipboardFormat can also recognize, and supports, a bitmap format on the Clipboard.

LoadFromClipboardFormat replaces the current image with the data pointed to by the AData parameter. The palette for the jpeg (see page 15685) image is specified by the APalette parameter.

See Also

TBitmap (see page 14396)

TClipboard (see page 3328)

TPicture (see page 14584)

RegisterClipboardFormat (see page 14592)

88.1.2.4.15 TJPEGImage.LoadFromStream Method

Loads the jpeg (see page 15685) image from a stream.

Pascal

```
procedure LoadFromStream(Stream: TStream); override;
```

C++

```
virtual __fastcall LoadFromStream(TStream Stream);
```

Description

Use LoadFromStream by specifying the stream from which the jpeg (see page 15685) image is loaded as the value of Stream. LoadFromStream reads a jpeg (see page 15685) data source from a stream and replaces the current image with it.

See Also

SaveToStream (see page 15700)

TJPEGData (see page 15686)

88.1.2.4.16 TJPEGImage.NewBitmap Method

Pascal

```
procedure NewBitmap;
```

C++

```
__fastcall NewBitmap();
```

Description

This is NewBitmap, a member of class TJPEGImage.

88.1.2.4.17 TJPEGImage.NewImage Method

Pascal

```
procedure NewImage;
```

C++

```
__fastcall NewImage();
```

Description

This is NewImage, a member of class TJPEGImage.

88.1.2.4.18 TJPEGImage.ReadData Method

Pascal

```
procedure ReadData(Stream: TStream); override;
```

C++

```
virtual __fastcall ReadData(TStream Stream);
```

Description

This is ReadData, a member of class TJPEGImage.

88.1.2.4.19 TJPEGImage.ReadStream Method

Pascal

```
procedure ReadStream(Size: Longint; Stream: TStream);
```

C++

```
__fastcall ReadStream(Longint Size, TStream Stream);
```

Description

This is ReadStream, a member of class TJPEGImage.

88.1.2.4.20 TJPEGImage.SaveToClipboardFormat Method

Writes a jpeg (see page 15685) image to a variable in Clipboard format.

Pascal

```
procedure SaveToClipboardFormat(var AFormat: Word; var AData: THandle; var APalette:
HPALETTE); override;
```

C++

```
virtual __fastcall SaveToClipboardFormat(Word AFormat, THandle AData, HPALETTE APalette);
```

Description

Use SaveToClipboardFormat to save the jpeg (see page 15685) image to a variable. The palette is returned as the APalette parameter, the format as the Aformat parameter, and the data as the AData parameter. Before the jpeg (see page 15685) image can be saved, an application must have registered the format with the TJPEGImage (see page 15688) object using the RegisterClipboardFormat method.

See Also

RegisterClipboardFormat (see page 14592)

88.1.2.4.21 TJPEGImage.SaveToStream Method

Writes the jpeg (see page 15685) data to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream); override;
```

C++

```
virtual __fastcall SaveToStream(TStream Stream);
```

Description

Use SaveToStream to save the jpeg (see page 15685) data to the stream specified by the Stream parameter. SaveToStream expects jpeg (see page 15685) image data and may incur overhead of a compression cycle if the source is a bitmap.

See Also

LoadFromStream (see page 15698)

88.1.2.4.22 TJPEGImage.SetHeight Method

Pascal

```
procedure SetHeight(Value: Integer); override;
```

C++

```
virtual __fastcall SetHeight(int Value);
```

Description

This is SetHeight, a member of class TJPEGImage.

88.1.2.4.23 TJPEGImage.SetPalette Method

Pascal

```
procedure SetPalette(Value: HPalette); override;
```

C++

```
virtual __fastcall SetPalette(HPalette Value);
```

Description

This is SetPalette, a member of class TJPEGImage.

88.1.2.4.24 TJPEGImage.SetWidth Method

Pascal

```
procedure SetWidth(Value: Integer); override;
```

C++

```
virtual __fastcall SetWidth(int Value);
```

Description

This is SetWidth, a member of class TJPEGImage.

88.1.2.4.25 TJPEGImage.WriteData Method

Pascal

```
procedure WriteData(Stream: TStream); override;
```

C++

```
virtual __fastcall WriteData(TStream Stream);
```

Description

This is WriteData, a member of class TJPEGImage.

88.1.2.5 TJPEGImage Properties

The properties of the TJPEGImage class are listed here.

Protected Properties

	Name	Description
	Bitmap (see page 15702)	This is Bitmap, a member of class TJPEGImage.

Public Properties

	Name	Description
	CompressionQuality (see page 15702)	Indicates the trade-off ratio between the image quality and the file size.
	Grayscale (see page 15702)	Determines whether the image output of a JPEG image is black and white, or color.
	Performance (see page 15703)	Controls (see page 5711) the trade-off between color quality and speed of decompression.
	PixelFormat (see page 15703)	Determines whether the JPEG image is displayed in 24 or 8-bit format.
	ProgressiveDisplay (see page 15703)	Controls (see page 5711) how the image is displayed when decompressing files.

	ProgressiveEncoding (see page 15704)	Determines whether an image can be progressively displayed when it is decompressed.
	Scale (see page 15704)	Determines the size of the JPEG image when it is displayed.
	Smoothing (see page 15704)	Determines whether the jpeg (see page 15685) image displays in blocks or with blurred edges.

88.1.2.5.1 TJPEGImage.Bitmap Property

Pascal

```
property Bitmap: TBitmap;
```

C++

```
__property TBitmap Bitmap;
```

Description

This is Bitmap, a member of class TJPEGImage.

88.1.2.5.2 TJPEGImage.CompressionQuality Property

Indicates the trade-off ratio between the image quality and the file size.

Pascal

```
property CompressionQuality: TJPEGQualityRange;
```

C++

```
__property TJPEGQualityRange CompressionQuality;
```

Description

Use CompressionQuality to set the compression quality of the JPEG image when writing out a jpeg (see page 15685) image. Higher compression results in a poorer picture quality, but a smaller file size. This property is not used for reading in files.

TJPEGQualityRange (see page 15709) is the type of the CompressionQuality property. The higher the TJPEGQualityRange (see page 15709) value (up to a maximum of 100), the better the image quality, but the larger the file size. The lower the TJPEGQualityRange (see page 15709) value (to a minimum of 1), the smaller the resulting file size, but at the expense of picture quality.

88.1.2.5.3 TJPEGImage.Grayscale Property

Determines whether the image output of a JPEG image is black and white, or color.

Pascal

```
property Grayscale: Boolean;
```

C++

```
__property Boolean Grayscale;
```

Description

Use GrayScale for reading in and writing out, that is, for decompressing and compressing, a jpeg (see page 15685) image. This property affects how the image is displayed.

GrayScale is used for speed in output optimization. When GrayScale is set to true, the color is separated from luminosity. Color takes the most time to decompress, so for previewing the image, GrayScale can be set to true for speed. The output then contains 255 shades of gray.

88.1.2.5.4 TJPEGImage.Performance Property

Controls (see page 5711) the trade-off between color quality and speed of decompression.

Pascal

```
property Performance: TJPEGPerformance;
```

C++

```
__property TJPEGPerformance Performance;
```

Description

Use Performance to set the performance of the jpeg (see page 15685) image when reading, that is, for decompressing files. This property is not used for writing out files.

Performance affects the display of the internal bitmap image. This image is a 256-color bitmap. Setting Performance to jpBestSpeed may cause some dithering in the bitmap image.

88.1.2.5.5 TJPEGImage.PixelFormat Property

Determines whether the JPEG image is displayed in 24 or 8-bit format.

Pascal

```
property PixelFormat: TJPEGPixelFormat;
```

C++

```
__property TJPEGPixelFormat PixelFormat;
```

Description

Use PixelFormat to set the pixel format of the jpeg (see page 15685) image to 8-bit for video drivers that cannot display 24-bit. The native format of a jpeg (see page 15685) image is 24-bit. PixelFormat is used for decompression, that is, for reading in files. PixelFormat will also apply to a bitmap if the image is copied to it.

88.1.2.5.6 TJPEGImage.ProgressiveDisplay Property

Controls (see page 5711) how the image is displayed when decompressing files.

Pascal

```
property ProgressiveDisplay: Boolean;
```

C++

```
__property Boolean ProgressiveDisplay;
```

Description

Use ProgressiveDisplay to turn on or off the incremental display of an image when reading in files. For example, if the jpeg (see page 15685) data source file is on the hard drive, progressively displaying it slows down the display time: in this case setting ProgressiveDisplay to false speeds up the displaying of the image. If the file must be downloaded, set ProgressiveDisplay to true if the downloading process will take as long as the progressive displaying of the image.

ProgressiveDisplay is used only for decompression, but is affected by the manner in which the image was compressed. Images that were compressed with ProgressiveEncoding (see page 15704) set to false cannot take advantage of ProgressiveDisplay. Setting ProgressiveDisplay to true for these images has no effect.

See Also

ProgressiveEncoding (see page 15704)

Smoothing (see page 15704)

88.1.2.5.7 TJPEGImage.ProgressiveEncoding Property

Determines whether an image can be progressively displayed when it is decompressed.

Pascal

```
property ProgressiveEncoding: Boolean;
```

C++

```
__property Boolean ProgressiveEncoding;
```

Description

Use ProgressiveEncoding to compress an image when writing out files, so that it can be progressively displayed when it is decompressed. Setting the ProgressiveDisplay (see page 15703) property to true has no effect if ProgressiveEncoding was not set to true when the image was compressed.

See Also

ProgressiveDisplay (see page 15703)

88.1.2.5.8 TJPEGImage.Scale Property

Determines the size of the JPEG image when it is displayed.

Pascal

```
property Scale: TJPEGScale;
```

C++

```
__property TJPEGScale Scale;
```

Description

Use Scale to optimize performance for previewing an image. jsFullSize displays the image as maximum size, whereas jsEighth displays an eighth of the image size, but 8 times faster.

TJPEGScale (see page 15708) is the type of the Scale property.

Scale it used for reading in files, that is, for decompression.

88.1.2.5.9 TJPEGImage.Smoothing Property

Determines whether the jpeg (see page 15685) image displays in blocks or with blurred edges.

Pascal

```
property Smoothing: Boolean;
```

C++

```
__property Boolean Smoothing;
```

Description

Smoothing is used if ProgressiveDisplay (see page 15703) is true, to determine whether the jpeg (see page 15685) image is displayed in small blocks or with blurred edges. If Smoothing is true, the edges are blurred during progressive displaying of the image. If Smoothing is false, raw blocks of color are used to display the image.

See Also

ProgressiveDisplay (see page 15703)

88.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TJPEGPerformance (see page 15707)	TJPEGPerformance represents the type of optimization to use when displaying a JPEG image.
	TJPEGPixelFormat (see page 15707)	TJPEGPixelFormat indicates the format used to display a JPEG image.
	TJPEGScale (see page 15708)	TJPEGScale represents the display size of a JPEG image.

Records

	Name	Description
	TJPEGDefaults (see page 15706)	This is record jpeg.TJPEGDefaults.

88.2.1 jpeg.TJPEGDefaults Record

Pascal

```
TJPEGDefaults = record
  CompressionQuality: TJPEGQualityRange;
  Grayscale: Boolean;
  Performance: TJPEGPerformance;
  PixelFormat: TJPEGPixelFormat;
  ProgressiveDisplay: Boolean;
  ProgressiveEncoding: Boolean;
  Scale: TJPEGScale;
  Smoothing: Boolean;
end;
```

C++

```
struct TJPEGDefaults {
  TJPEGQualityRange CompressionQuality;
  Boolean Grayscale;
  TJPEGPerformance Performance;
  TJPEGPixelFormat PixelFormat;
  Boolean ProgressiveDisplay;
  Boolean ProgressiveEncoding;
  TJPEGScale Scale;
  Boolean Smoothing;
};
```

File

jpeg

Description

This is record jpeg.TJPEGDefaults.

88.2.2 jpeg.TJPEGPerformance Enumeration

TJPEGPerformance represents the type of optimization to use when displaying a JPEG image.

Pascal

```
TJPEGPerformance = (
  jpBestQuality,
  jpBestSpeed
);
```

C++

```
enum TJPEGPerformance {
  jpBestQuality,
  jpBestSpeed
};
```

File

jpeg

Description

The following table lists the possible values:

Value	Meaning
jpBestQuality	Optimize for image quality. This results in slower rendering of the image.
jpBestSpeed	Optimize for speed of display. This may result in some dithering in the image.

88.2.3 jpeg.TJPEGPixelFormat Enumeration

TJPEGPixelFormat indicates the format used to display a JPEG image.

Pascal

```
TJPEGPixelFormat = (
  jf24Bit,
  jf8Bit
);
```

C++

```
enum TJPEGPixelFormat {
  jf24Bit,
  jf8Bit
};
```

File

jpeg

Description

The following table lists the possible values:

Value	Meaning
jf24Bit	Use the native 24-bit jpeg (see page 15685) image format.

jf8Bit	Use an 8-bit format (for video drivers that do not support the 24-bit format)

88.2.4 jpeg.TJPEGScale Enumeration

TJPEGScale represents the display size of a JPEG image.

Pascal

```
TJPEGScale = (
  jsFullSize,
  jsHalf,
  jsQuarter,
  jsEighth
);
```

C++

```
enum TJPEGScale {
  jsFullSize,
  jsHalf,
  jsQuarter,
  jsEighth
};
```

File

jpeg

Description

The following table lists the values of TJPEGScale:

Value	Meaning
jsFullSize	Displays a full-sized image.
jsHalf	Displays a half-size image in half the time it takes to display a full-sized image.
jsQuarter	Displays a quarter-size image in a quarter of the time it takes to display a full-sized image.
jsEighth	Displays an eighth-size image in an eighth of the time it takes to display a full-sized image.

88.3 Types

The following table lists types in this documentation.

Types

Name	Description
TJPEGQualityRange (see page 15709)	TJPEGQualityRange represents the trade-off ratio between image quality and file size in a JPEG image.

88.3.1 jpeg.TJPEGQualityRange Type

TJPEGQualityRange represents the trade-off ratio between image quality and file size in a JPEG image.

Pascal

```
TJPEGQualityRange = 1..100;
```

C++

```
1..100 TJPEGQualityRange;
```

File

jpeg

Description

TJPEGQualityRange represents a scale of 1 to 100, where higher values represent better image quality (but larger file size) and lower values represent smaller file sizes but with a loss of picture quality.

88.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
JPEGDefaults (see page 15710)	This is variable jpeg.JPEGDefaults.

88.4.1 jpeg.JPEGDefaults Variable

Pascal

```
JPEGDefaults: TJPEGDefaults = ( CompressionQuality: 90; Grayscale: False; Performance:
jpBestQuality; PixelFormat: jf24Bit; ProgressiveDisplay: False; ProgressiveEncoding: False;
Scale: jsFullSize; Smoothing: True; );
```

C++

```
TJPEGDefaults JPEGDefaults = ( CompressionQuality: 90; Grayscale: False; Performance:
jpBestQuality; PixelFormat: jf24Bit; ProgressiveDisplay: False; ProgressiveEncoding: False;
Scale: jsFullSize; Smoothing: True; );
```

File

jpeg

Description

This is variable jpeg.JPEGDefaults.

89 ListActns Namespace

This is namespace ListActns.

Topics

Name	Description
Classes (see page 15713)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 15792)	The following table lists structs, records, enums in this documentation.
Types (see page 15793)	The following table lists types in this documentation.

Classes

	Name	Description
	TCustomListAction (see page 15713)	TCustomListAction is the base class for action objects that provide items to client controls.
	TCustomStaticListAction (see page 15724)	This is class ListActns.TCustomStaticListAction.
	TCustomVirtualListAction (see page 15738)	This is class ListActns.TCustomVirtualListAction.
	TListActionLink (see page 15744)	This is class ListActns.TListActionLink.
	TListControlItem (see page 15750)	TListControlItem represents an item with a text value, an image, and a data pointer.
	TListControlItems (see page 15757)	TListControlItems manages a collection of TListControlItem (see page 15750) objects.
	TStaticListAction (see page 15765)	TStaticListAction provides items to client controls.
	TStaticListItems (see page 15777)	TStaticListItems represents the items that TStaticListAction (see page 15765) supplies to client controls.
	TVirtualListAction (see page 15780)	TVirtualListAction provides items to a client controls.

Structs, Records, Enums

	Name	Description
	TListItemsSortType (see page 15792)	This is record ListActns.TListItemsSortType.

Types

Name	Description
TGetItemCountEvent (see page 15793)	This is type ListActns.TGetItemCountEvent.
TGetItemEvent (see page 15793)	This is type ListActns.TGetItemEvent.
TGetVirtualItemEvent (see page 15794)	This is type ListActns.TGetVirtualItemEvent.
TItemSelectedEvent (see page 15794)	This is type ListActns.TItemSelectedEvent.
TListCompareEvent (see page 15794)	This is type ListActns.TListCompareEvent.
TListControlItemClass (see page 15795)	This is type ListActns.TListControlItemClass.

TListItemsCompare (see page 15795)	This is type ListActns.TListItemsCompare.

89.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCustomListAction (see page 15713)	TCustomListAction is the base class for action objects that provide items to client controls.
	TCustomStaticListAction (see page 15724)	This is class ListActns.TCustomStaticListAction.
	TCustomVirtualListAction (see page 15738)	This is class ListActns.TCustomVirtualListAction.
	TListActionLink (see page 15744)	This is class ListActns.TListActionLink.
	TListControlItem (see page 15750)	TListControlItem represents an item with a text value, an image, and a data pointer.
	TListControlItems (see page 15757)	TListControlItems manages a collection of TListControlItem (see page 15750) objects.
	TStaticListAction (see page 15765)	TStaticListAction provides items to client controls.
	TStaticListItems (see page 15777)	TStaticListItems represents the items that TStaticListAction (see page 15765) supplies to client controls.
	TVirtualListAction (see page 15780)	TVirtualListAction provides items to a client controls.

89.1.1 TCustomListAction Class

TCustomListAction is the base class for action objects that provide items to client controls.

Class Hierarchy

Pascal

```
TCustomListAction = class(TCustomAction);
```

C++

```
class TCustomListAction : public TCustomAction;
```

File

ListActns

Description

TCustomListAction implements the common behavior for actions that supply the items to a list view, list box or combo box.

When an application sets the Active (see page 15721) property of a list action to true, the action fills all client list boxes and combo boxes with items. Each item may include information such as a text caption, an image, and a pointer to data.

When the user selects an item in one of the client controls, the client "executes" the list action, which causes it to generate an OnItemSelected (see page 15724) event.

When a TCustomListAction is added to an action manager, it uses the action to generate a drop-down list on a tool bar.

See Also

TStaticListAction (see page 15765)

TVirtualListAction (see page 15780)

89.1.1.1 TCustomListAction Members

The following tables list the members exposed by TCustomListAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.
	OnGetItemCount (see page 15723)	Occurs when the list action needs to determine the number of items it supplies to client controls.
	OnItemSelected (see page 15724)	Occurs when a client control executes the action because a user selected one of the items.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetCount (see page 15719)	This is GetCount, a member of class TCustomListAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetString (see page 15719)	Returns the value of the Strings (see page 15723) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 15720)	Initializes the action after it is loaded into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.

V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetString (see page 15720)	Sets the value of the Strings (see page 15723) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Images (see page 15722)	Lists the images that the list action supplies along with items.
R	Loading (see page 15723)	Indicates whether the action is currently adding items to client controls.
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.

	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 15718)	Creates an instance of TCustomListAction (see page 15713).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 15718)	Generates an OnItemSelected (see page 15724) event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 15719)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.

	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	Active (see page 15721)	Indicates whether the list action has supplied items to its client controls.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Count (see page 15722)	This is Count, a member of class TCustomListAction.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
	ItemIndex (see page 15722)	Specifies the index of the selected item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	Strings (see page 15723)	This is Strings, a member of class TCustomListAction.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Properties

	Name	Description
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

89.1.1.2 TCustomListAction.Create Constructor

Creates an instance of TCustomListAction (see page 15713).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomListAction(TComponent * AOwner);
```

Description

Most applications do not need to call the TCustomListAction (see page 15713) constructor because actions are added at design time to an action list or action manager.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TCustomListAction (see page 15713) instance.

See Also

Owner (see page 2807)

89.1.1.3 TCustomListAction Methods

The methods of the TCustomListAction class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 15719)	This is GetCount, a member of class TCustomListAction.
	GetString (see page 15719)	Returns the value of the Strings (see page 15723) property.
	Loaded (see page 15720)	Initializes the action after it is loaded into memory.
	SetString (see page 15720)	Sets the value of the Strings (see page 15723) property.

Public Methods

	Name	Description
	ExecuteTarget (see page 15718)	Generates an OnItemSelected (see page 15724) event.
	HandlesTarget (see page 15719)	Indicates whether the action can execute its function.

89.1.1.3.1 TCustomListAction.ExecuteTarget Method

Generates an OnItemSelected (see page 15724) event.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

ExecuteTarget is called automatically when the user selects one of the items that the list action supplies to a client list box or combo box.

Target is the control whose item was selected.

ExecuteTarget updates the ItemIndex (see page 15722) property of all client controls that descend from TCustomListControl, and then generates an OnItemSelected (see page 15724) event, where your application can respond to the user's selection.

See Also

OnItemSelected (see page 15724)

ItemIndex (see page 15722)

89.1.1.3.2 TCustomListAction.GetCount Method

Pascal

```
function GetCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetCount();
```

Description

This is GetCount, a member of class TCustomListAction.

89.1.1.3.3 TCustomListAction.GetString Method

Returns the value of the Strings (see page 15723) property.

Pascal

```
function GetString(Index: Integer): String; virtual;
```

C++

```
virtual __fastcall AnsiString GetString(int Index);
```

Description

GetString is the protected read implementation of the Strings (see page 15723) property.

In TCustomListAction (see page 15713), GetString always returns an empty string.

Override GetString to provide a more meaningful value to the Strings (see page 15723) property. For example, TStaticListAction (see page 15765) overrides GetString to return the caption of the specified item.

See Also

strings (see page 15723)

89.1.1.3.4 TCustomListAction.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the currently active object.

HandlesTarget always returns true because the list action's ability to perform its function is not dependant on any particular target.

See Also

ExecuteTarget (see page 15718)

UpdateTarget (see page 2715)

89.1.1.3.5 TCustomListAction.Loaded Method

Initializes the action after it is loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Loaded is called automatically when the form or data module containing the action has been completely loaded into memory. TCustomListAction (see page 15713) overrides the inherited method to set Active (see page 15721) and Images (see page 15722) properties now that the other components involved (the client controls whose items are set when Active (see page 15721) becomes true and the image list component that is the value of Images (see page 15722)) are known to be loaded into memory.

See Also

Loaded (see page 2782)

Images (see page 15722)

Active (see page 15721)

89.1.1.3.6 TCustomListAction.SetString Method

Sets the value of the Strings (see page 15723) property.

Pascal

```
procedure SetString(Index: Integer; const Value: String); virtual;
```

C++

```
virtual __fastcall SetString(int Index, const AnsiString Value);
```

Description

SetString is the protected write implementation of the Strings (see page 15723) property.

In TCustomListAction (see page 15713), SetString does nothing.

Override SetString to provide a more meaningful value to the Strings (see page 15723) property. For example, TStaticListAction (see page 15765) overrides SetString to set the caption of the specified item.

See Also

strings (see page 15723)

89.1.1.4 TCustomListAction Properties

The properties of the TCustomListAction class are listed here.

Protected Properties

	Name	Description
	Images (see page 15722)	Lists the images that the list action supplies along with items.
R	Loading (see page 15723)	Indicates whether the action is currently adding items to client controls.

Public Properties

	Name	Description
	Active (see page 15721)	Indicates whether the list action has supplied items to its client controls.
R	Count (see page 15722)	This is Count, a member of class TCustomListAction.
	ItemIndex (see page 15722)	Specifies the index of the selected item.
	Strings (see page 15723)	This is Strings, a member of class TCustomListAction.

89.1.1.4.1 TCustomListAction.Active Property

Indicates whether the list action has supplied items to its client controls.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Set Active to true to cause the list action to fill all client controls with the items. These Items are supplied by calling the GetItem method, iterating over every index from 0 to Count (see page 15722).

See Also

GetItem

getitem

Items (see page 21348)

items

Items (see page 21187)

count (see page 15722)

Items (see page 3814)

strings (see page 15723)

89.1.1.4.2 TCustomListAction.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TCustomListAction.

89.1.1.4.3 TCustomListAction.Images Property

Lists the images that the list action supplies along with items.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

For each item that the list action supplies to clients, it can supply an image index along with other information such as the text value. The supplied image index refers to an image in the Images property.

Note: List actions do not supply images to all clients. The Images property controls the SmallImages property of a list view or the Images property of a TComboBoxEx descendant.

See Also

SmallImages (see page 3825)

Images (see page 3700)

89.1.1.4.4 TCustomListAction.ItemIndex Property

Specifies the index of the selected item.

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

Description

ItemIndex represents the index of the selected item in all client controls that descend from TCustomListControl. Setting ItemIndex causes these clients to change their selection. When the user updates the selection using one of these clients, TCustomListAction (see page 15713) automatically updates the others so that they select the same item.

Note: List actions do not supply ItemIndex to all clients. The ItemIndex property controls only the ItemIndex property of clients that descend from TCustomListControl.

See Also

OnItemSelected (see page 15724)

ItemIndex (see page 5944)

89.1.1.4.5 TCustomListAction.Loading Property

Indicates whether the action is currently adding items to client controls.

Pascal

```
property Loading: Boolean;
```

C++

```
__property Boolean Loading;
```

Description

Loading is used internally to detect when the list action is in the process of supplying client controls with items.

See Also

Active (see page 15721)

89.1.1.4.6 TCustomListAction.Strings Property

Pascal

```
property Strings [Index: Integer]: String;
```

C++

```
__property AnsiString Strings[int Index];
```

Description

This is Strings, a member of class TCustomListAction.

89.1.1.5 TCustomListAction Events

The events of the TCustomListAction class are listed here.

Protected Events

	Name	Description
	OnGetItemCount (see page 15723)	Occurs when the list action needs to determine the number of items it supplies to client controls.
	OnItemSelected (see page 15724)	Occurs when a client control executes the action because a user selected one of the items.

89.1.1.5.1 TCustomListAction.OnGetItemCount Event

Occurs when the list action needs to determine the number of items it supplies to client controls.

Pascal

```
property OnGetItemCount: TGetItemCountEvent;
```

C++

```
__property TGetItemCountEvent OnGetItemCount;
```

Description

The GetCount (see page 15719) method generates an OnGetItemCount event to supply the value of the Count (see page 15722) property. Write an OnGetItemCount event handler to indicate the number of items for which you are supplying values in

the OnGetItem event handler.

Sender is the list action that is supplying data to client controls.

Count (see page 15722) returns the number of items this list action supplies to client controls.

See Also

getcount (see page 15719)

count (see page 15722)

89.1.1.5.2 TCustomListAction.OnItemSelected Event

Occurs when a client control executes the action because a user selected one of the items.

Pascal

```
property OnItemSelected: TItemSelectedEvent;
```

C++

```
__property TItemSelectedEvent OnItemSelected;
```

Description

Write an OnItemSelected event handler to respond when the user selects one of the items supplied to client controls. Before this event is called, the list action first updates the ItemIndex (see page 15722) property of any client controls that descend from TCustomListControl.

Sender is the list action that supplies items to the control in which an item was selected.

Control is the control in which the user selected an item.

See Also

ExecuteTarget (see page 15718)

ItemIndex (see page 15722)

89.1.2 TCustomStaticListAction Class

Class Hierarchy

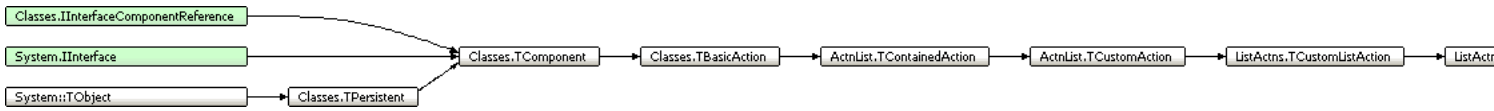

Pascal

```
TCustomStaticListAction = class(TCustomListAction);
```

C++

```
class TCustomStaticListAction : public TCustomListAction;
```

File

ListActns

Description

This is class ListActns.TCustomStaticListAction.

89.1.2.1 TCustomStaticListAction Members

The following tables list the members exposed by TCustomStaticListAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.
	OnGetItemCount (see page 15723)	Occurs when the list action needs to determine the number of items it supplies to client controls.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetCount (see page 15730)	This is GetCount, a member of class TCustomStaticListAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetItem (see page 15730)	This is GetItem, a member of class TCustomStaticListAction.
V	GetItemClass (see page 15730)	This is GetItemClass, a member of class TCustomStaticListAction.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetString (see page 15730)	Returns the value of the Strings property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 15720)	Initializes the action after it is loaded into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.

	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetString (see page 15731)	Sets the value of the Strings property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Loading (see page 15723)	Indicates whether the action is currently adding items to client controls.
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 15729)	Creates an instance of TCustomListAction.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 15729)	Frees an instance of TCustomAction.

	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
V	ExecuteTarget (see page 15718)	Generates an OnItemSelected (see page 15724) event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 15719)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.

V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Count (see page 15732)	This is Count, a member of class TCustomStaticListAction.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Strings (see page 15723)	This is Strings, a member of class TCustomListAction.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnGetItem (see page 15736)	This is OnGetItem, a member of class TCustomStaticListAction.
	OnHint (see page 15736)	Occurs when the mouse pauses over a client control or menu item.
	OnItemSelected (see page 15736)	Occurs when a client control executes the action because a user selected one of the items.
	OnUpdate (see page 15737)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Active (see page 15731)	Indicates whether the list action has supplied items to its client controls.
	Caption (see page 15732)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Enabled (see page 15733)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 15733)	Indicates the help context ID for client controls and menu items.
	Hint (see page 15733)	Indicates the Help hint for client controls and menu items.
	Images (see page 15734)	Lists the images that the list action supplies along with items.
	ItemIndex (see page 15734)	Specifies the index of the selected item.
	Items (see page 15734)	This is Items, a member of class TCustomStaticListAction.
	Name (see page 2805)	Specifies the name of the component as referenced in code.

	SecondaryShortCuts (see page 15735)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 15735)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 15735)	Specifies the Visible property for client controls and menu items.

89.1.2.2 TCustomStaticListAction.Create Constructor

Creates an instance of TCustomListAction.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomStaticListAction(TComponent * AOwner);
```

Description

Most applications do not need to call the TCustomListAction constructor because actions are added at design time to an action list or action manager.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TCustomListAction instance.

See Also

Owner (see page 2807)

89.1.2.3 TCustomStaticListAction.Destroy Destructor

Frees an instance of TCustomAction.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomStaticListAction();
```

Description

Do not call the destructor directly in an application. Instead, call Free. Free ensures that the action is not nil before calling the destructor.

If the action is contained in an action list or action manager, Destroy removes the action from that list.

See Also

Create (see page 301)

Actions (see page 325)

89.1.2.4 TCustomStaticListAction Methods

The methods of the TCustomStaticListAction class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 15730)	This is GetCount, a member of class TCustomStaticListAction.

	GetItem (see page 15730)	This is GetItem, a member of class TCustomStaticListAction.
V	GetItemClass (see page 15730)	This is GetItemClass, a member of class TCustomStaticListAction.
V	GetString (see page 15730)	Returns the value of the Strings property.
V	SetString (see page 15731)	Sets the value of the Strings property.

89.1.2.4.1 TCustomStaticListAction.GetCount Method

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

This is GetCount, a member of class TCustomStaticListAction.

89.1.2.4.2 TCustomStaticListAction.GetItem Method

Pascal

```
function GetItem(const Index: Integer; AnItem: TListControlItem): Boolean;
```

C++

```
__fastcall Boolean GetItem(const int Index, TListControlItem AnItem);
```

Description

This is GetItem, a member of class TCustomStaticListAction.

89.1.2.4.3 TCustomStaticListAction.GetItemClass Method

Pascal

```
function GetItemClass: TListControlItemClass; virtual;
```

C++

```
virtual __fastcall TListControlItemClass GetItemClass();
```

Description

This is GetItemClass, a member of class TCustomStaticListAction.

89.1.2.4.4 TCustomStaticListAction.GetString Method

Returns the value of the Strings property.

Pascal

```
function GetString(Index: Integer): String; override;
```

C++

```
virtual __fastcall AnsiString GetString(int Index);
```

Description

GetString is the protected read implementation of the Strings property.

In TCustomListAction, GetString always returns an empty string.

Override GetString to provide a more meaningful value to the Strings property. For example, TStaticListAction overrides GetString to return the caption of the specified item.

See Also

strings (see page 15723)

89.1.2.4.5 TCustomStaticListAction.SetString Method

Sets the value of the Strings property.

Pascal

```
procedure SetString(Index: Integer; const Value: String); override;
```

C++

```
virtual __fastcall SetString(int Index, const AnsiString Value);
```

Description

SetString is the protected write implementation of the Strings property.

In TCustomListAction, SetString does nothing.

Override SetString to provide a more meaningful value to the Strings property. For example, TStaticListAction overrides SetString to set the caption of the specified item.

See Also

strings (see page 15723)

89.1.2.5 TCustomStaticListAction Properties

The properties of the TCustomStaticListAction class are listed here.

Public Properties

	Name	Description
	Count (see page 15732)	This is Count, a member of class TCustomStaticListAction.

Published Properties

	Name	Description
	Active (see page 15731)	Indicates whether the list action has supplied items to its client controls.
	Caption (see page 15732)	Represents the caption of client controls and menu items.
	Enabled (see page 15733)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 15733)	Indicates the help context ID for client controls and menu items.
	Hint (see page 15733)	Indicates the Help hint for client controls and menu items.
	Images (see page 15734)	Lists the images that the list action supplies along with items.
	ItemIndex (see page 15734)	Specifies the index of the selected item.
	Items (see page 15734)	This is Items, a member of class TCustomStaticListAction.
	SecondaryShortCuts (see page 15735)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 15735)	Specifies the ShortCut property for client menu items.
	Visible (see page 15735)	Specifies the Visible property for client controls and menu items.

89.1.2.5.1 TCustomStaticListAction.Active Property

Indicates whether the list action has supplied items to its client controls.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Set Active to true to cause the list action to fill all client controls with the items. These Items are supplied by calling the GetItem method, iterating over every index from 0 to Count.

See Also

GetItem

getitem

Items (see page 21348)

items

Items (see page 21187)

count (see page 15722)

Items (see page 3814)

strings (see page 15723)

89.1.2.5.2 TCustomStaticListAction.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

89.1.2.5.3 TCustomStaticListAction.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TCustomStaticListAction.

89.1.2.5.4 TCustomStaticListAction.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

89.1.2.5.5 TCustomStaticListAction.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

89.1.2.5.6 TCustomStaticListAction.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

89.1.2.5.7 TCustomStaticListAction.Images Property

Lists the images that the list action supplies along with items.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

For each item that the list action supplies to clients, it can supply an image index along with other information such as the text value. The supplied image index refers to an image in the Images property.

Note: List actions do not supply images to all clients. The Images property controls the SmallImages property of a list view or the Images property of a TComboBoxEx descendant.

See Also

SmallImages (see page 3825)

Images (see page 3700)

89.1.2.5.8 TCustomStaticListAction.ItemIndex Property

Specifies the index of the selected item.

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

Description

ItemIndex represents the index of the selected item in all client controls that descend from TCustomListControl. Setting ItemIndex causes these clients to change their selection. When the user updates the selection using one of these clients, TCustomListAction automatically updates the others so that they select the same item.

Note: List actions do not supply ItemIndex to all clients. The ItemIndex property controls only the ItemIndex property of clients that descend from TCustomListControl.

See Also

OnItemSelected (see page 15724)

ItemIndex (see page 5944)

89.1.2.5.9 TCustomStaticListAction.Items Property

Pascal

```
property Items: TStaticListItems;
```

C++

```
__property TStaticListItems Items;
```

Description

This is Items, a member of class TCustomStaticListAction.

89.1.2.5.10 TCustomStaticListAction.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

89.1.2.5.11 TCustomStaticListAction.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

89.1.2.5.12 TCustomStaticListAction.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

89.1.2.6 TCustomStaticListAction Events

The events of the TCustomStaticListAction class are listed here.

Published Events

	Name	Description
	OnGetItem (see page 15736)	This is OnGetItem, a member of class TCustomStaticListAction.
	OnHint (see page 15736)	Occurs when the mouse pauses over a client control or menu item.
	OnItemSelected (see page 15736)	Occurs when a client control executes the action because a user selected one of the items.
	OnUpdate (see page 15737)	Occurs when the application is idle or when the action list updates.

89.1.2.6.1 TCustomStaticListAction.OnGetItem Event

Pascal

```
property OnGetItem: TGetItemEvent;
```

C++

```
__property TGetItemEvent OnGetItem;
```

Description

This is OnGetItem, a member of class TCustomStaticListAction.

89.1.2.6.2 TCustomStaticListAction.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

89.1.2.6.3 TCustomStaticListAction.OnItemSelected Event

Occurs when a client control executes the action because a user selected one of the items.

Pascal

```
property OnItemSelected: TItemSelectedEvent;
```

C++

```
__property TItemSelectedEvent OnItemSelected;
```

Description

Write an OnItemSelected event handler to respond when the user selects one of the items supplied to client controls. Before this event is called, the list action first updates the ItemIndex property of any client controls that descend from TCustomListControl.

Sender is the list action that supplies items to the control in which an item was selected.

Control is the control in which the user selected an item.

See Also

ExecuteTarget (see page 15718)

ItemIndex (see page 15722)

89.1.2.6.4 TCustomStaticListAction.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
```

```
      }
      pa->Checked = ToolBar1->Visible;
    }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

89.1.3 TCustomVirtualListAction Class

Class Hierarchy

Pascal

```
TCustomVirtualListAction = class(TCustomListAction);
```

C++

```
class TCustomVirtualListAction : public TCustomListAction;
```

File

ListActns

Description

This is class ListActns.TCustomVirtualListAction.

89.1.3.1 TCustomVirtualListAction Members

The following tables list the members exposed by TCustomVirtualListAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.
	OnGetItemCount (see page 15723)	Occurs when the list action needs to determine the number of items it supplies to client controls.
	OnItemSelected (see page 15724)	Occurs when a client control executes the action because a user selected one of the items.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetCount (see page 15719)	This is GetCount, a member of class TCustomListAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetItem (see page 15743)	This is GetItem, a member of class TCustomVirtualListAction.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetString (see page 15743)	Returns the value of the Strings property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 15720)	Initializes the action after it is loaded into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.

	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetString (see page 15720)	Sets the value of the Strings (see page 15723) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Images (see page 15722)	Lists the images that the list action supplies along with items.
R	Loading (see page 15723)	Indicates whether the action is currently adding items to client controls.
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnGetItem (see page 15744)	This is OnGetItem, a member of class TCustomVirtualListAction.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 15718)	Creates an instance of TCustomListAction (see page 15713).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
V	ExecuteTarget (see page 15718)	Generates an OnItemSelected (see page 15724) event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 15719)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	Active (see page 15721)	Indicates whether the list action has supplied items to its client controls.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Count (see page 15744)	This is Count, a member of class TCustomVirtualListAction.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
	ItemIndex (see page 15722)	Specifies the index of the selected item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	Strings (see page 15723)	This is Strings, a member of class TCustomListAction.

	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Properties

	Name	Description
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

89.1.3.2 TCustomVirtualListAction Methods

The methods of the TCustomVirtualListAction class are listed here.

Protected Methods

	Name	Description
	GetItem (see page 15743)	This is GetItem, a member of class TCustomVirtualListAction.
	GetString (see page 15743)	Returns the value of the Strings property.

89.1.3.2.1 TCustomVirtualListAction.GetItem Method

Pascal

```
function GetItem(const Index: Integer; var Value: String; var ImageIndex: Integer; var Data:
Pointer): Boolean;
```

C++

```
__fastcall Boolean GetItem(const int Index, AnsiString Value, int ImageIndex, void * Data);
```

Description

This is GetItem, a member of class TCustomVirtualListAction.

89.1.3.2.2 TCustomVirtualListAction.GetString Method

Returns the value of the Strings property.

Pascal

```
function GetString(Index: Integer): String; override;
```

C++

```
virtual __fastcall AnsiString GetString(int Index);
```

Description

GetString is the protected read implementation of the Strings property.

In TCustomListAction, GetString always returns an empty string.

Override GetString to provide a more meaningful value to the Strings property. For example, TStaticListAction overrides GetString to return the caption of the specified item.

See Also

strings (see page 15723)

89.1.3.3 TCustomVirtualListAction Properties

The properties of the TCustomVirtualListAction class are listed here.

Public Properties

	Name	Description
	Count (see page 15744)	This is Count, a member of class TCustomVirtualListAction.

89.1.3.3.1 TCustomVirtualListAction.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TCustomVirtualListAction.

89.1.3.4 TCustomVirtualListAction Events

The events of the TCustomVirtualListAction class are listed here.

Public Events

	Name	Description
	OnGetItem (see page 15744)	This is OnGetItem, a member of class TCustomVirtualListAction.

89.1.3.4.1 TCustomVirtualListAction.OnGetItem Event

Pascal

```
property OnGetItem: TGetVirtualItemEvent;
```

C++

```
__property TGetVirtualItemEvent OnGetItem;
```

Description

This is OnGetItem, a member of class TCustomVirtualListAction.

89.1.4 TListActionLink Class

Class Hierarchy

System::TObject → Classes.TBasicActionLink → ActnList.TActionLink → Controls.TControlActionLink → Controls.TWinControlActionLink → ListActns.TListActionLink

Pascal

```
TListActionLink = class(TWinControlActionLink);
```

C++

```
class TListActionLink : public TWinControlActionLink;
```

File

ListActns

Description

This is class ListActns.TListActionLink.

89.1.4.1 TListActionLink Members

The following tables list the members exposed by TListActionLink.

Protected Fields

	Name	Description
	FAction (see page 2722)	This is FAction, a member of class TBasicActionLink.
	FClient (see page 5894)	This is FClient, a member of class TControlActionLink.
	FClient (see page 6236)	This is FClient, a member of class TWinControlActionLink.

Protected Methods

	Name	Description
	AddItem (see page 15747)	This is the overview for the AddItem method overload.
	AssignClient (see page 2723)	This is AssignClient, a member of class TBasicActionLink.
	AssignClient (see page 5895)	Assigns a client control that is linked to the associated action.
	AssignClient (see page 6236)	Specifies the client control.
	Change (see page 2723)	This is Change, a member of class TBasicActionLink.
	DoShowHint (see page 5896)	Indicates whether the hint is shown, and modified.
	IsActiveLinked (see page 15748)	This is IsActiveLinked, a member of class TListActionLink.
	IsCaptionLinked (see page 268)	Indicates whether the Caption properties for the action and client are linked.
	IsCaptionLinked (see page 5896)	Indicates whether the caption for the action and the client are linked.
	IsCheckedLinked (see page 269)	Indicates whether the Checked properties for the action and client are linked.
	IsDropdownMenuLinked (see page 5896)	This is IsDropdownMenuLinked, a member of class TControlActionLink.
	IsEnabledLinked (see page 269)	Indicates whether the Enabled properties for the action and client are linked.
	IsEnabledLinked (see page 5897)	Indicates whether the enabled state for the action and the client are linked.
	IsEnableDropdownLinked (see page 5897)	This is IsEnableDropdownLinked, a member of class TControlActionLink.
	IsGroupIndexLinked (see page 270)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsHelpContextLinked (see page 270)	Indicates whether the HelpContext properties for the action and client controls are linked.
	IsHelpContextLinked (see page 6236)	Indicates whether the Help context for the action and the client are linked.
	IsHelpLinked (see page 271)	Indicates whether the help properties for the action and client are linked.
	IsHelpLinked (see page 5897)	Indicates whether the help properties for the action and the client are linked.
	IsHintLinked (see page 271)	Indicates whether the Hint properties for the action and client are linked.
	IsHintLinked (see page 5898)	Indicates whether the Hint property for the action and the client are linked.
	IsImageIndexLinked (see page 272)	Indicates whether the image index for the action and the client are linked.
	IsImagesLinked (see page 15748)	This is IsImagesLinked, a member of class TListActionLink.
	IsOnExecuteLinked (see page 2723)	This is IsOnExecuteLinked, a member of class TBasicActionLink.
	IsOnExecuteLinked (see page 5898)	Indicates whether the OnExecute event handler for the action and the client are linked.
	IsPopupMenuLinked (see page 5898)	This is IsPopupMenuLinked, a member of class TControlActionLink.
	IsShortCutLinked (see page 272)	Indicates whether the ShortCut properties for the action and client are linked.
	IsVisibleLinked (see page 273)	Indicates whether the Visible properties for the action and client are linked.
	IsVisibleLinked (see page 5899)	Indicates whether the visible state for the action and the client are linked.
	RefreshControl (see page 15748)	This is RefreshControl, a member of class TListActionLink.

	SetAction (see page 15749)	This is SetAction, a member of class TListActionLink.
	SetActive (see page 15749)	This is SetActive, a member of class TListActionLink.
	SetAutoCheck (see page 274)	Introduces an interface for setting the AutoCheck property of a client.
	SetCaption (see page 274)	Introduces an interface for setting the caption on a client.
	SetCaption (see page 5899)	Sets the caption of a control if the client is linked to the action for this event.
	SetChecked (see page 274)	Introduces an interface for setting the checked state of a client.
	SetDropdownMenu (see page 5899)	This is SetDropdownMenu, a member of class TControlActionLink.
	SetEnabled (see page 275)	Introduces an interface for setting the enabled state of a client.
	SetEnabled (see page 5900)	Sets the enabled state of a control if the client is linked to the action for this event.
	SetEnableDropdown (see page 5900)	This is SetEnableDropdown, a member of class TControlActionLink.
	SetGroupIndex (see page 275)	Introduces an interface for setting the GroupIndex property of a client.
	SetHelpContext (see page 276)	Introduces an interface for setting the help context of a client.
	SetHelpContext (see page 5900)	Sets the HelpContext of a control if the client is linked to the action help properties.
	SetHelpContext (see page 6237)	Sets the Help context of a windowed control if the client is linked to the action for this event.
	SetHelpKeyword (see page 276)	Introduces an interface for setting the help keyword of a client.
	SetHelpKeyword (see page 5901)	Sets the HelpKeyword of a control if the client is linked to the action help properties.
	SetHelpType (see page 277)	Introduces an interface for setting the HelpType property of a client.
	SetHelpType (see page 5901)	Sets the HelpType of a control if the client is linked to the action help properties.
	SetHint (see page 278)	Introduces an interface for setting the hint for a client.
	SetHint (see page 5901)	Sets the hint of a control if the client is linked to the action for this event.
	SetImageIndex (see page 278)	Introduces an interface for setting the image index for a client.
	SetImages (see page 15749)	This is SetImages, a member of class TListActionLink.
	SetItemIndex (see page 15749)	This is SetItemIndex, a member of class TListActionLink.
	SetOnExecute (see page 2724)	This is SetOnExecute, a member of class TBasicActionLink.
	SetOnExecute (see page 5902)	Sets the execution event handler for a control if the client is linked to the action for this event.
	SetPopupMenu (see page 5902)	This is SetPopupMenu, a member of class TControlActionLink.
	SetShortCut (see page 279)	Introduces an interface for setting the shortcut for a client.
	SetVisible (see page 279)	Introduces an interface for setting the visible state of a client.
	SetVisible (see page 5902)	Sets the visible state of a control if the client is linked to the action for this event.

Public Events

	Name	Description
	OnChange (see page 2725)	Occurs when an OnChange event is triggered on the action.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2721)	Instantiates and initializes a TBasicActionLink (see page 2720) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2722)	Destroys an instance of an action link object.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.

	Execute (see page 2723)	Invokes the action's Execute method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Update (see page 2724)	Invokes the action's Update method.

Public Properties

	Name	Description
	Action (see page 2725)	Indicates which action the link connects to its client.

89.1.4.2 TListActionLink Methods

The methods of the TListActionLink class are listed here.

Protected Methods

	Name	Description
	AddItem (see page 15747)	This is the overview for the AddItem method overload.
	IsActiveLinked (see page 15748)	This is IsActiveLinked, a member of class TListActionLink.
	IsImagesLinked (see page 15748)	This is IsImagesLinked, a member of class TListActionLink.
	RefreshControl (see page 15748)	This is RefreshControl, a member of class TListActionLink.
	SetAction (see page 15749)	This is SetAction, a member of class TListActionLink.
	SetActive (see page 15749)	This is SetActive, a member of class TListActionLink.
	SetImages (see page 15749)	This is SetImages, a member of class TListActionLink.
	SetItemIndex (see page 15749)	This is SetItemIndex, a member of class TListActionLink.

89.1.4.2.1 AddItem Method

This is the overview for the AddItem method overload.

Overload List

	Name	Description
	TListActionLink.AddItem (String, Integer, Pointer) (see page 15748)	This is AddItem, a member of class TListActionLink.
	TListActionLink.AddItem (TListControlItem) (see page 15748)	This is AddItem, a member of class TListActionLink.

89.1.4.2.1.1 TListActionLink.AddItem Method (String, Integer, Pointer)

Pascal

```
procedure AddItem(ACaption: String; AImageIndex: Integer; DataPtr: Pointer); virtual; overload;
```

C++

```
virtual __fastcall AddItem(AnsiString ACaption, int AImageIndex, void * DataPtr);
```

Description

This is AddItem, a member of class TListActionLink.

89.1.4.2.1.2 TListActionLink.AddItem Method (TListControlItem)

Pascal

```
procedure AddItem(AnItem: TListControlItem); virtual; overload;
```

C++

```
virtual __fastcall AddItem(TListControlItem AnItem);
```

Description

This is AddItem, a member of class TListActionLink.

89.1.4.2.2 TListActionLink.IsActiveLinked Method

Pascal

```
function IsActiveLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsActiveLinked();
```

Description

This is IsActiveLinked, a member of class TListActionLink.

89.1.4.2.3 TListActionLink.IsImagesLinked Method

Pascal

```
function IsImagesLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsImagesLinked();
```

Description

This is IsImagesLinked, a member of class TListActionLink.

89.1.4.2.4 TListActionLink.RefreshControl Method

Pascal

```
procedure RefreshControl;
```

C++

```
__fastcall RefreshControl();
```

Description

This is RefreshControl, a member of class TListActionLink.

89.1.4.2.5 TListActionLink.SetAction Method

Pascal

```
procedure SetAction(Value: TBasicAction); override;
```

C++

```
virtual __fastcall SetAction(TBasicAction Value);
```

Description

This is SetAction, a member of class TListActionLink.

89.1.4.2.6 TListActionLink.SetActive Method

Pascal

```
procedure SetActive(const Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetActive(const Boolean Value);
```

Description

This is SetActive, a member of class TListActionLink.

89.1.4.2.7 TListActionLink.SetImages Method

Pascal

```
procedure SetImages(Value: TCustomImageList); virtual;
```

C++

```
virtual __fastcall SetImages(TCustomImageList Value);
```

Description

This is SetImages, a member of class TListActionLink.

89.1.4.2.8 TListActionLink.SetItemIndex Method

Pascal

```
procedure SetItemIndex(const Value: Integer); virtual;
```

C++

```
virtual __fastcall SetItemIndex(const int Value);
```

Description

This is SetItemIndex, a member of class TListActionLink.

89.1.5 TListControlItem Class

TListControlItem represents an item with a text value, an image, and a data pointer.

Class Hierarchy

Pascal

```
TListControlItem = class(TCollectionItem);
```

C++

```
class TListControlItem : public TCollectionItem;
```

File

ListActns

Description

TListControlItem represents a single item in a control that lists items. TListControlItem objects are used by TStaticListAction (see page 15765) to represent the items in a variety of client controls such as list views, list boxes, and combo boxes. In addition, TListControlItem is the base class for TComboExItem, which is specialized to work with the TComboBoxEx control.

Each TListControlItem has

A text value, which it stores as the Caption (see page 15756) property.

An associated image, which is identified by the ImageIndex (see page 15757) property.

A data pointer, which it stores as the Data (see page 15756) property.

See Also

TListControlItems (see page 15757)

TComboExItem (see page 3546)

89.1.5.1 TListControlItem Members

The following tables list the members exposed by TListControlItem.

Protected Fields

	Name	Description
	FCaption (see page 15752)	This is FCaption, a member of class TListControlItem.
	FData (see page 15752)	This is FData, a member of class TListControlItem.
	FImageIndex (see page 15753)	This is FImageIndex, a member of class TListControlItem.

Protected Methods

	Name	Description
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 15754)	Responds when any of the item's properties changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetDisplayName (see page 15754)	Specifies the name of the item as it appears in the Object Inspector.
V	GetOwner (see page 2759)	Returns the collection to which this item belongs.

	GetOwner (see page 2932)	Returns the owner of an object.
V	SetCaption (see page 15755)	Sets the value of the Caption (see page 15756) property.
V	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
V	SetData (see page 15755)	Sets the value of the Data (see page 15756) property.
V	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
V	SetImageIndex (see page 15755)	Sets the value of the ImageIndex (see page 15757) property.
V	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 15753)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 15752)	Creates and initializes a TListControlItem (see page 15750) instance.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	Data (see page 15756)	Points to user-defined data associated with the control.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	Caption (see page 15756)	Specifies the text value of the item.
	ImageIndex (see page 15757)	Identifies the image associated with this item.

89.1.5.2 TListControlItem.Create Constructor

Creates and initializes a TListControlItem (see page 15750) instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TListControlItem(TCollection Collection);
```

Description

Do not call the constructor to create a TListControlItem (see page 15750) instance. Instead, use the collection's Add method to create a new list control item.

Collection is the collection object that owns and manages this list control item.

See Also

Add (see page 2736)

89.1.5.3 TListControlItem Fields

The fields of the TListControlItem class are listed here.

Protected Fields

	Name	Description
	FCaption (see page 15752)	This is FCaption, a member of class TListControlItem.
	FData (see page 15752)	This is FData, a member of class TListControlItem.
	FImageIndex (see page 15753)	This is FImageIndex, a member of class TListControlItem.

89.1.5.3.1 TListControlItem.FCaption Field

Pascal

```
FCaption: String;
```

C++

```
AnsiString FCaption;
```

Description

This is FCaption, a member of class TListControlItem.

89.1.5.3.2 TListControlItem.FData Field

Pascal

```
FData: Pointer;
```

C++

```
void * FData;
```

Description

This is FData, a member of class TListControlItem.

89.1.5.3.3 TListControlItem.FImageIndex Field

Pascal

```
FImageIndex: TImageIndex;
```

C++

```
TImageIndex FImageIndex;
```

Description

This is FImageIndex, a member of class TListControlItem.

89.1.5.4 TListControlItem Methods

The methods of the TListControlItem class are listed here.

Protected Methods

	Name	Description
	Changed (see page 15754)	Responds when any of the item's properties changes.
V	GetDisplayName (see page 15754)	Specifies the name of the item as it appears in the Object Inspector.
V	SetCaption (see page 15755)	Sets the value of the Caption (see page 15756) property.
V	SetData (see page 15755)	Sets the value of the Data (see page 15756) property.
V	SetImageIndex (see page 15755)	Sets the value of the ImageIndex (see page 15757) property.

Public Methods

	Name	Description
V	Assign (see page 15753)	Copies the contents of another, similar object.

89.1.5.4.1 TListControlItem.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo (see page 2931)

EConvertError (see page 22724)

89.1.5.4.2 TListControlItem.Changed Method

Responds when any of the item's properties changes.

Pascal

```
procedure Changed;
```

C++

```
__fastcall Changed();
```

Description

Changed is called automatically when the item's properties change. It calls the Update method of the corresponding TListControlItems (see page 15757) object, so that the collection can perform any necessary adjustments.

See Also

Update (see page 2747)

89.1.5.4.3 TListControlItem.GetDisplayName Method

Specifies the name of the item as it appears in the Object Inspector.

Pascal

```
function GetDisplayName: String; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName returns the string that represents the item in the Collection editor and the Object Inspector.

GetDisplayName returns the value of the Caption (see page 15756) if it is set. Otherwise, it returns the class name of the TListControlItem (see page 15750) descendant for which this is an instance, with a number added to ensure uniqueness.

See Also

Caption (see page 15756)

89.1.5.4.4 TListControlItem.SetCaption Method

Sets the value of the Caption (see page 15756) property.

Pascal

```
procedure SetCaption(const Value: String); virtual;
```

C++

```
virtual __fastcall SetCaption(const AnsiString Value);
```

Description

SetCaption is the protected write implementation of the Caption (see page 15756) property. It sets the Caption (see page 15756) property to Value, and then calls the Changed (see page 15754) method so that the collection can update itself.

See Also

Caption (see page 15756)

Changed (see page 15754)

89.1.5.4.5 TListControlItem.SetData Method

Sets the value of the Data (see page 15756) property.

Pascal

```
procedure SetData(const Value: Pointer); virtual;
```

C++

```
virtual __fastcall SetData(const void * Value);
```

Description

SetData is the protected write implementation of the Data (see page 15756) property. It sets the Data (see page 15756) property to Value, and then calls the Changed (see page 15754) method so that the collection can update itself.

See Also

Data (see page 15756)

Changed (see page 15754)

89.1.5.4.6 TListControlItem.SetImageIndex Method

Sets the value of the ImageIndex (see page 15757) property.

Pascal

```
procedure SetImageIndex(const Value: TImageIndex); virtual;
```

C++

```
virtual __fastcall SetImageIndex(const TImageIndex Value);
```

Description

SetImageIndex is the protected write implementation of the ImageIndex (see page 15757) property. It sets the ImageIndex (see page 15757) property to Value, and then calls the Changed (see page 15754) method so that the collection can update

itself.

See Also

ImageIndex (see page 15757)

Changed (see page 15754)

89.1.5.5 TListControlItem Properties

The properties of the TListControlItem class are listed here.

Public Properties

	Name	Description
	Data (see page 15756)	Points to user-defined data associated with the control.

Published Properties

	Name	Description
	Caption (see page 15756)	Specifies the text value of the item.
	ImageIndex (see page 15757)	Identifies the image associated with this item.

89.1.5.5.1 TListControlItem.Caption Property

Specifies the text value of the item.

Pascal

```
property Caption: String;
```

C++

```
__property AnsiString Caption;
```

Description

Set Caption to provide the item with a text value. This is the value that appears in list boxes or combo boxes, or which labels items in list view controls.

See Also

ImageIndex (see page 15757)

Data (see page 15756)

89.1.5.5.2 TListControlItem.Data Property

Points to user-defined data associated with the control.

Pascal

```
property Data: Pointer;
```

C++

```
__property void * Data;
```

Description

Set Data to associate this item with application-specific data. Data has no predefined meaning, although it is used for sorting items when the managing collection's SortType property is stData.

See Also

SortType (see page 15764)

ImageIndex (see page 15757)

Caption (see page 15756)

89.1.5.5.3 TListControlItem.ImageIndex Property

Identifies the image associated with this item.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

Set ImageIndex to associate this item with an image. ImageIndex is an index into the Images property of the component that uses this TListControlItem (see page 15750) object to represent one of its items.

See Also

Images (see page 15722)

Images (see page 3700)

89.1.6 TListControlItems Class

TListControlItems manages a collection of TListControlItem (see page 15750) objects.

Class Hierarchy

Pascal

```
TListControlItems = class(TOwnedCollection);
```

C++

```
class TListControlItems : public TOwnedCollection;
```

File

ListActns

Description

TListControlItems manages a collection of items that each include a text value, associated image, and a data pointer.

Use the properties and methods of TListControlItems to:

Access a specific item.

Add (see page 15760) or delete items from the collection.

Find out how many items are in the collection.

Sort (see page 15762) the items in the collection.

Although TListControlItems can be used on its own, it also acts as a base class to classes that manage the items for a specific component. TListControlItems descendants include TComboExItems, which manages the items of a TComboBoxEx control, and TStaticListItems (see page 15777), which manages items for a list action component.

See Also

TListControlItem (see page 15750)

TStaticListItems (see page 15777)

TComboExItems (see page 3552)

89.1.6.1 TListControlItems Members

The following tables list the members exposed by TListControlItems.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	CompareItems (see page 15761)	Returns the relative order of two items in the collection.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2915)	Returns the owner of an object.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 15760)	Adds a new item to the collection.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 15760)	Creates and initializes a TOwnedCollection instance.
	CustomSort (see page 15761)	Sorts the items in the collection using a specified comparison function.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Sort (see page 15762)	Sorts the items in the collection.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
R	Items (see page 15763)	Lists the items in the collection.

Published Events

	Name	Description
	OnCompare (see page 15764)	Occurs to when two items need to be compared for sorting the collection.

Published Properties

	Name	Description
	CaseSensitive (see page 15762)	Controls (see page 5711) whether items are sorted in a case-sensitive or case-insensitive manner.

	SortType (see page 15764)	Indicates how the items should be sorted.

89.1.6.2 TListControlItems.Create Constructor

Creates and initializes a TOwnedCollection instance.

Pascal

```
constructor Create(AOwner: TPersistent; ItemClass: TCollectionItemClass);
```

C++

```
__fastcall TListControlItems(TPersistent * AOwner, TCollectionItemClass ItemClass);
```

Description

The Create method takes two parameters: the name of an instance object descended from TPersistent and the name of a TCollectionItem descendant class. The first parameter is the owner of the TOwnedCollection instance. The second parameter determines the class of the items created by the Add method.

See Also

Add (see page 2736)

89.1.6.3 TListControlItems Methods

The methods of the TListControlItems class are listed here.

Protected Methods

	Name	Description
	CompareItems (see page 15761)	Returns the relative order of two items in the collection.

Public Methods

	Name	Description
	Add (see page 15760)	Adds a new item to the collection.
	CustomSort (see page 15761)	Sorts the items in the collection using a specified comparison function.
	Sort (see page 15762)	Sorts the items in the collection.

89.1.6.3.1 TListControlItems.Add Method

Adds a new item to the collection.

Pascal

```
function Add: TListControlItem;
```

C++

```
__fastcall TListControlItem Add();
```

Description

Call Add to add a new item to the collection. Add instantiates an object of the type that was specified when the TListControlItems (see page 15757) instance was created. It returns the newly created object.

See Also

Items (see page 15763)

Delete (see page 2741)

89.1.6.3.2 TListControlItems.CompareItems Method

Returns the relative order of two items in the collection.

Pascal

```
function CompareItems(I1: TListControlItem; I2: TListControlItem): Integer; virtual;
```

C++

```
virtual __fastcall int CompareItems(TListControlItem I1, TListControlItem I2);
```

Description

CompareItems is used internally to provide the default sort order for items in the list. It is used when the SortType (see page 15764) property is not stNone. The Sort (see page 15762) method also uses this as a comparison function.

If I1 is greater than I2, CompareItems returns a value greater than 0. If I1 is the same as I2, CompareItems returns 0. If I1 is less than I2, CompareItems returns a value less than 0.

CompareItems always uses an OnCompare (see page 15764) event handler if it is assigned. Otherwise, it compares the captions of the items, taking into account the CaseSensitive (see page 15762) property. Descendant classes can override CompareItems to provide a different default sort order.

See Also

SortType (see page 15764)

CustomSort (see page 15761)

Sort (see page 15762)

CaseSensitive (see page 15762)

OnCompare (see page 15764)

89.1.6.3.3 TListControlItems.CustomSort Method

Sorts the items in the collection using a specified comparison function.

Pascal

```
procedure CustomSort(Compare: TListItemsCompare);
```

C++

```
__fastcall CustomSort(TListItemsCompare Compare);
```

Description

Call Sort (see page 15762) to sort the items in the collection based on the comparison function that is the value of the Compare parameter.

Compare is a function that compares two items in the list. Its List parameter is this TListControlItems (see page 15757) object. The Index1 and Index2 parameters are the indexes of the items to compare in the Items (see page 15763) property array. This comparison function should return a value less than 0 if the first item is less than the second, a value greater than 0 if it is greater than the second, and 0 if the two items are equivalent.

Note: CustomSort sorts the items in the list, but does not maintain the sort order when new items are added. To provide a custom sort order that persists when new items are added, set the SortType (see page 15764) property to stData and supply an OnCompare (see page 15764) event handler.

See Also

SortType (see page 15764)

Sort (see page 15762)

OnCompare (see page 15764)

Items (see page 15763)

89.1.6.3.4 TListControlItems.Sort Method

Sorts the items in the collection.

Pascal

```
procedure Sort;
```

C++

```
__fastcall Sort();
```

Description

Call Sort to sort the items in the collection. Sort uses the comparison function specified by the protected CompareItems (see page 15761) method to determine the sort order.

Note: Sort sorts the items in the list, but does not maintain the sort order when new items are added. To impose a sort order that persists when new items are added, set the SortType (see page 15764) property to a value other than stNone and, if desired, supply an OnCompare (see page 15764) event handler.

See Also

SortType (see page 15764)

CustomSort (see page 15761)

OnCompare (see page 15764)

Items (see page 15763)

CompareItems (see page 15761)

89.1.6.4 TListControlItems Properties

The properties of the TListControlItems class are listed here.

Public Properties

	Name	Description
R	Items (see page 15763)	Lists the items in the collection.

Published Properties

	Name	Description
	CaseSensitive (see page 15762)	Controls (see page 5711) whether items are sorted in a case-sensitive or case-insensitive manner.
	SortType (see page 15764)	Indicates how the items should be sorted.

89.1.6.4.1 TListControlItems.CaseSensitive Property

Controls (see page 5711) whether items are sorted in a case-sensitive or case-insensitive manner.

Pascal

```
property CaseSensitive: Boolean;
```

C++

```
__property Boolean CaseSensitive;
```

Description

Use CaseSensitive to indicate whether the case of item captions should be considered when sorting the items in the list.

See Also

Items (see page 15763)

SortType (see page 15764)

Caption (see page 15756)

89.1.6.4.2 TListControlItems.Items Property

Lists the items in the collection.

Pascal

```
property Items [const Index: Integer]: TListControlItem;
```

C++

```
__property TListControlItem Items[int const Index];
```

Description

Use Items to access the items in this collection.

Index indicates the item to access, where 0 is the index of the first item, 1 is the index of the second item, and so on. The Count property indicates the maximum number of items.

Note: In Delphi code, Items is the default property for TListControlItems (see page 15757). This means that the property name can be omitted when using the Items property. For example,

```
StaticListAction.Items.Items[0].Caption := Edit1.Text;
```

Note: can also be written

```
StaticListAction.Items[0].Caption := Edit1.Text;
```

Note: Items is a read-only property. To add or delete items from the collection, use the Add (see page 15760) and Delete methods.

See Also

Add (see page 15760)

Delete (see page 2741)

Count (see page 2749)

operator_[]

89.1.6.4.3 TListControlItems.SortType Property

Indicates how the items should be sorted.

Pascal

```
property SortType: TListItemsSortType;
```

C++

```
__property TListItemsSortType SortType;
```

Description

Use SortType to indicate how you want the items sorted, if at all. The following table lists the possible values:

Value	Meaning
stNone	Items (see page 15763) are not automatically sorted. You can still sort the items explicitly by calling the Sort (see page 15762) or CustomSort (see page 15761) method.
stData	Items (see page 15763) are sorted based on the value of their Data property. You must supply an OnCompare (see page 15764) event handler to specify the sort order.
stText	Items (see page 15763) are sorted base on the value of their Caption property. The CaseSensitive (see page 15762) property specifies whether comparisons are case-sensitive.
stBoth	Items (see page 15763) are sorted based on an OnCompare (see page 15764) event handler if one is assigned, otherwise based on the value of their Caption property.

See Also

Items (see page 15763)

OnCompare (see page 15764)

Data (see page 15756)

CaseSensitive (see page 15762)

CustomSort (see page 15761)

Sort (see page 15762)

89.1.6.5 TListControlItems Events

The events of the TListControlItems class are listed here.

Published Events

	Name	Description
	OnCompare (see page 15764)	Occurs to when two items need to be compared for sorting the collection.

89.1.6.5.1 TListControlItems.OnCompare Event

Occurs to when two items need to be compared for sorting the collection.

Pascal

```
property OnCompare: TListCompareEvent;
```

C++

```
__property TListCompareEvent OnCompare;
```

Description

Write an OnCompare event handler to indicate the sort order when the SortType (see page 15764) property is stData or stBoth.

The OnCompare event handler compares the items passed as the Item1 and Item2 parameters. If Item1 is the same as Item2 in the sort order, the event handler should return 0. If Item1 is less than Item2, the event handler should return a value less than 0. If Item1 is greater than Item2, the event handler should return value greater than 0.

See Also

SortType (see page 15764)

89.1.7 TStaticListAction Class

TStaticListAction provides items to client controls.

Class Hierarchy

Pascal

```
TStaticListAction = class(TCustomStaticListAction);
```

C++

```
class TStaticListAction : public TCustomStaticListAction;
```

File

ListActns

Description

TStaticListAction supplies items to client controls such as list views, list boxes, or combo boxes. TStaticListAction maintains a list of items in its Items (see page 15773) property. This lets you specify, at design time, the items it supplies to clients.

When an application sets the Active (see page 15770) property of a list action to true, the action fills all client controls with items. Each item may include text, an image, and a pointer to data. Each client control determines which of these features of the item it accepts.

When the user selects an item in one of the client controls, the client "executes" the list action, which causes it to generate an OnItemSelected (see page 15775) event.

When a TStaticListAction is added to an action manager, it uses the action to generate a drop-down list on a tool bar.

See Also

TVirtualListAction (see page 15780)

TCustomCombo (see page 21154)

TCustomListView (see page 3757)

TCustomListBox (see page 21308)

TCustomComboBox (see page 21201)

89.1.7.1 TStaticListAction Members

The following tables list the members exposed by TStaticListAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.
	OnGetItemCount (see page 15723)	Occurs when the list action needs to determine the number of items it supplies to client controls.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetCount (see page 15719)	This is GetCount, a member of class TCustomListAction.
V	GetCount (see page 15730)	This is GetCount, a member of class TCustomStaticListAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetItem (see page 15730)	This is GetItem, a member of class TCustomStaticListAction.
V	GetItemClass (see page 15730)	This is GetItemClass, a member of class TCustomStaticListAction.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetString (see page 15719)	Returns the value of the Strings (see page 15723) property.
V	GetString (see page 15730)	Returns the value of the Strings property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 15720)	Initializes the action after it is loaded into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.

	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetString (see page 15720)	Sets the value of the Strings (see page 15723) property.
	SetString (see page 15731)	Sets the value of the Strings property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Loading (see page 15723)	Indicates whether the action is currently adding items to client controls.
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 15718)	Creates an instance of TCustomListAction (see page 15713).
	Create (see page 15729)	Creates an instance of TCustomListAction.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 15729)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	ExecuteTarget (see page 15718)	Generates an OnItemSelected (see page 15724) event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 15719)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Count (see page 15722)	This is Count, a member of class TCustomListAction.
	Count (see page 15732)	This is Count, a member of class TCustomStaticListAction.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Strings (see page 15723)	This is Strings, a member of class TCustomListAction.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnGetItem (see page 15775)	This is OnGetItem, a member of class TStaticListAction.
	OnHint (see page 15775)	Occurs when the mouse pauses over a client control or menu item.
	OnItemSelected (see page 15775)	Occurs when a client control executes the action because a user selected one of the items.

	OnUpdate (see page 15776)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Active (see page 15770)	Indicates whether the list action has supplied items to its client controls.
	Caption (see page 15771)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Enabled (see page 15771)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 15771)	Indicates the help context ID for client controls and menu items.
	Hint (see page 15772)	Indicates the Help hint for client controls and menu items.
	Images (see page 15772)	Lists the images that the list action supplies along with items.
	ItemIndex (see page 15773)	Specifies the index of the selected item.
	Items (see page 15773)	Lists the items that this list action supplies to clients.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 15773)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 15774)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 15774)	Specifies the Visible property for client controls and menu items.

89.1.7.2 TStaticListAction Properties

The properties of the TStaticListAction class are listed here.

Published Properties

	Name	Description
	Active (see page 15770)	Indicates whether the list action has supplied items to its client controls.
	Caption (see page 15771)	Represents the caption of client controls and menu items.
	Enabled (see page 15771)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 15771)	Indicates the help context ID for client controls and menu items.
	Hint (see page 15772)	Indicates the Help hint for client controls and menu items.
	Images (see page 15772)	Lists the images that the list action supplies along with items.
	ItemIndex (see page 15773)	Specifies the index of the selected item.
	Items (see page 15773)	Lists the items that this list action supplies to clients.
	SecondaryShortCuts (see page 15773)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 15774)	Specifies the ShortCut property for client menu items.
	Visible (see page 15774)	Specifies the Visible property for client controls and menu items.

89.1.7.2.1 TStaticListAction.Active Property

Indicates whether the list action has supplied items to its client controls.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Set Active to true to cause the list action to fill all client controls with the items. These Items are supplied by calling the GetItem method, iterating over every index from 0 to Count.

See Also

GetItem

getitem

Items (see page 21348)

items

Items (see page 21187)

count (see page 15722)

Items (see page 3814)

strings (see page 15723)

89.1.7.2.2 TStaticListAction.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

89.1.7.2.3 TStaticListAction.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

89.1.7.2.4 TStaticListAction.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

89.1.7.2.5 TStaticListAction.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

89.1.7.2.6 TStaticListAction.Images Property

Lists the images that the list action supplies along with items.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

For each item that the list action supplies to clients, it can supply an image index along with other information such as the text value. The supplied image index refers to an image in the Images property.

Note: List actions do not supply images to all clients. The Images property controls the SmallImages property of a list view or the Images property of a TComboBoxEx descendant.

See Also

SmallImages (see page 3825)

Images (see page 3700)

89.1.7.2.7 TStaticListAction.ItemIndex Property

Specifies the index of the selected item.

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

Description

ItemIndex represents the index of the selected item in all client controls that descend from TCustomListControl. Setting ItemIndex causes these clients to change their selection. When the user updates the selection using one of these clients, TCustomListAction automatically updates the others so that they select the same item.

Note: List actions do not supply ItemIndex to all clients. The ItemIndex property controls only the ItemIndex property of clients that descend from TCustomListControl.

See Also

OnItemSelected (see page 15724)

ItemIndex (see page 5944)

89.1.7.2.8 TStaticListAction.Items Property

Lists the items that this list action supplies to clients.

Pascal

```
property Items: TStaticListItems;
```

C++

```
__property TStaticListItems Items;
```

Description

Set Items to indicate the items that this action supplies to clients. At design time, you can use the collection editor to add items to the list and set their properties.

See Also

Count

strings

89.1.7.2.9 TStaticListAction.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

89.1.7.2.10 TStaticListAction.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

89.1.7.2.11 TStaticListAction.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

89.1.7.3 TStaticListAction Events

The events of the TStaticListAction class are listed here.

Published Events

	Name	Description
	OnGetItem (see page 15775)	This is OnGetItem, a member of class TStaticListAction.
	OnHint (see page 15775)	Occurs when the mouse pauses over a client control or menu item.
	OnItemSelected (see page 15775)	Occurs when a client control executes the action because a user selected one of the items.
	OnUpdate (see page 15776)	Occurs when the application is idle or when the action list updates.

89.1.7.3.1 TStaticListAction.OnGetItem Event

Pascal

```
property OnGetItem: TGetItemEvent;
```

C++

```
__property TGetItemEvent OnGetItem;
```

Description

This is OnGetItem, a member of class TStaticListAction.

89.1.7.3.2 TStaticListAction.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

89.1.7.3.3 TStaticListAction.OnItemSelected Event

Occurs when a client control executes the action because a user selected one of the items.

Pascal

```
property OnItemSelected: TItemSelectedEvent;
```

C++

```
__property TItemSelectedEvent OnItemSelected;
```

Description

Write an OnItemSelected event handler to respond when the user selects one of the items supplied to client controls. Before this event is called, the list action first updates the ItemIndex property of any client controls that descend from TCustomListControl.

Sender is the list action that supplies items to the control in which an item was selected.

Control is the control in which the user selected an item.

See Also

ExecuteTarget (see page 15718)

ItemIndex (see page 15722)

89.1.7.3.4 TStaticListAction.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
```

```
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

89.1.8 TStaticListItems Class

TStaticListItems represents the items that TStaticListAction (see page 15765) supplies to client controls.

Class Hierarchy

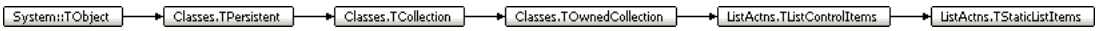

Pascal

```
TStaticListItems = class(TListControlItems);
```

C++

```
class TStaticListItems : public TListControlItems;
```

File

ListActns

Description

TStaticListItems manages a collection of items for a TStaticListAction (see page 15765) component. Each item in the collection can include a text value, associated image, and a data pointer. If the associated TStaticListAction (see page 15765) is active, then when items are added to or deleted from TStaticListItems, the list action's clients are updated so that they reflect the current set of items.

See Also

TStaticListAction (see page 15765)

89.1.8.1 TStaticListItems Members

The following tables list the members exposed by TStaticListItems.

Protected Methods

	Name	Description
	Added (see page 2736)	Responds when items are added to the collection.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
	CompareItems (see page 15761)	Returns the relative order of two items in the collection.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	Deleting (see page 2741)	Responds when items are deleted from the collection.

	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2915)	Returns the owner of an object.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Notify (see page 15779)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 15780)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 2736)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	Add (see page 15760)	Adds a new item to the collection.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 2734)	Creates and initializes a collection.
	Create (see page 2914)	Creates and initializes a TOwnedCollection (see page 2912) instance.
	Create (see page 15760)	Creates and initializes a TOwnedCollection instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CustomSort (see page 15761)	Sorts the items in the collection using a specified comparison function.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Sort (see page 15762)	Sorts the items in the collection.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 2751)	Lists the items in the collection.
R	Items (see page 15763)	Lists the items in the collection.

Published Events

	Name	Description
	OnCompare (see page 15764)	Occurs to when two items need to be compared for sorting the collection.

Published Properties

	Name	Description
	CaseSensitive (see page 15762)	Controls (see page 5711) whether items are sorted in a case-sensitive or case-insensitive manner.
	SortType (see page 15764)	Indicates how the items should be sorted.

89.1.8.2 TStaticListItems Methods

The methods of the TStaticListItems class are listed here.

Protected Methods

	Name	Description
V	Notify (see page 15779)	Responds when items are added to or removed from the collection.
V	Update (see page 15780)	Updates the collection to reflect changes to its items.

89.1.8.2.1 TStaticListItems.Notify Method

Responds when items are added to or removed from the collection.

Pascal

```
procedure Notify(Item: TCollectionItem; Action: TCollectionNotification); override;
```

C++

```
virtual __fastcall Notify(TCollectionItem Item, TCollectionNotification Action);
```

Description

Notify is called automatically when the items in the collection change.

Item is the item that was just added to or that is about to be removed from the collection.

Action indicates whether item was added, is about to be removed, or is about to be deleted:

As implemented in TCollection, Notify calls Added when Action is cnAdded and calls Deleting when Action is cnDeleting. TCollection ignores the cnExtracting action. Descendant classes can override Notify to modify this behavior.

See Also

Added (see page 2736)

Deleting (see page 2741)

89.1.8.2.2 TStaticListItems.Update Method

Updates the collection to reflect changes to its items.

Pascal

```
procedure Update(Item: TCollectionItem); override;
```

C++

```
virtual __fastcall Update(TCollectionItem Item);
```

Description

Override Update in a descendant class to make any necessary changes when the items in the collection change. This method is called automatically when an update is complete.

Item identifies the item that changed. If the Item parameter is nil (Delphi) or NULL (C++), then the change affects more than one item in the collection.

As implemented in TCollection, Update does nothing. Descendant classes override this method to make any necessary adjustments.

See Also

Changed (see page 2740)

Changed (see page 2758)

89.1.9 TVirtualListAction Class

TVirtualListAction provides items to a client controls.

Class Hierarchy

Pascal

```
TVirtualListAction = class(TCustomVirtualListAction);
```

C++

```
class TVirtualListAction : public TCustomVirtualListAction;
```

File

ListActns

Description

TVirtualListAction supplies items dynamically at runtime to client controls such as list views, list boxes, or combo boxes. These items are supplied by the code you supply to an OnGetItem (see page 15790) event handler.

When an application sets the Active (see page 15786) property of a list action to true, the action fills all client controls with items. Each item may include text, an image, and a pointer to data. Each client control determines which of these features of the item it accepts.

When the user selects an item in one of the client controls, the client "executes" the list action, which causes it to generate an OnItemSelected (see page 15791) event.

When a TVirtualListAction is added to an action manager, it uses the action to generate a drop-down list on a tool bar.

See Also

TStaticListAction (see page 15765)

TCustomCombo (see page 21154)

TCustomListView (see page 3757)

TCustomListBox (see page 21308)

TCustomComboBox (see page 21201)

89.1.9.1 TVirtualListAction Members

The following tables list the members exposed by TVirtualListAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetCount (see page 15719)	This is GetCount, a member of class TCustomListAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetItem (see page 15743)	This is GetItem, a member of class TCustomVirtualListAction.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetString (see page 15719)	Returns the value of the Strings (see page 15723) property.
V	GetString (see page 15743)	Returns the value of the Strings property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 15720)	Initializes the action after it is loaded into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetString (see page 15720)	Sets the value of the Strings (see page 15723) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Loading (see page 15723)	Indicates whether the action is currently adding items to client controls.
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 15718)	Creates an instance of TCustomListAction (see page 15713).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".

	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	ExecuteTarget (see page 15718)	Generates an OnItemSelected (see page 15724) event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 15719)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
	Update (see page 295)	Indicates whether the action updates itself.
	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Count (see page 15722)	This is Count, a member of class TCustomListAction.
	Count (see page 15744)	This is Count, a member of class TCustomVirtualListAction.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Strings (see page 15723)	This is Strings, a member of class TCustomListAction.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnGetItem (see page 15790)	This is OnGetItem, a member of class TVirtualListAction.
	OnGetItemCount (see page 15790)	Occurs when the list action needs to determine the number of items it supplies to client controls.
	OnHint (see page 15791)	Occurs when the mouse pauses over a client control or menu item.
	OnItemSelected (see page 15791)	Occurs when a client control executes the action because a user selected one of the items.

Published Properties

	Name	Description
	Active (see page 15786)	Indicates whether the list action has supplied items to its client controls.
	Caption (see page 15786)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Enabled (see page 15787)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 15787)	Indicates the help context ID for client controls and menu items.
	Hint (see page 15788)	Indicates the Help hint for client controls and menu items.
	Images (see page 15788)	Lists the images that the list action supplies along with items.
	ItemIndex (see page 15788)	Specifies the index of the selected item.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 15789)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 15789)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 15789)	Specifies the Visible property for client controls and menu items.

89.1.9.2 TVirtualListAction Properties

The properties of the TVirtualListAction class are listed here.

Published Properties

	Name	Description
	Active (see page 15786)	Indicates whether the list action has supplied items to its client controls.
	Caption (see page 15786)	Represents the caption of client controls and menu items.
	Enabled (see page 15787)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 15787)	Indicates the help context ID for client controls and menu items.
	Hint (see page 15788)	Indicates the Help hint for client controls and menu items.
	Images (see page 15788)	Lists the images that the list action supplies along with items.
	ItemIndex (see page 15788)	Specifies the index of the selected item.
	SecondaryShortCuts (see page 15789)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 15789)	Specifies the ShortCut property for client menu items.
	Visible (see page 15789)	Specifies the Visible property for client controls and menu items.

89.1.9.2.1 TVirtualListAction.Active Property

Indicates whether the list action has supplied items to its client controls.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Set Active to true to cause the list action to fill all client controls with the items. These Items are supplied by calling the GetItem method, iterating over every index from 0 to Count.

See Also

GetItem

getitem

Items (see page 21348)

items

Items (see page 21187)

count (see page 15722)

Items (see page 3814)

strings (see page 15723)

89.1.9.2.2 TVirtualListAction.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

89.1.9.2.3 TVirtualListAction.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

89.1.9.2.4 TVirtualListAction.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

89.1.9.2.5 TVirtualListAction.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

89.1.9.2.6 TVirtualListAction.Images Property

Lists the images that the list action supplies along with items.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

For each item that the list action supplies to clients, it can supply an image index along with other information such as the text value. The supplied image index refers to an image in the Images property.

Note: List actions do not supply images to all clients. The Images property controls the SmallImages property of a list view or the Images property of a TComboBoxEx descendant.

See Also

SmallImages (see page 3825)

Images (see page 3700)

89.1.9.2.7 TVirtualListAction.ItemIndex Property

Specifies the index of the selected item.

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

Description

ItemIndex represents the index of the selected item in all client controls that descend from TCustomListControl. Setting ItemIndex causes these clients to change their selection. When the user updates the selection using one of these clients, TCustomListAction automatically updates the others so that they select the same item.

Note: List actions do not supply ItemIndex to all clients. The ItemIndex property controls only the ItemIndex property of clients that descend from TCustomListControl.

See Also

OnItemSelected (see page 15724)

ItemIndex (see page 5944)

89.1.9.2.8 TVirtualListAction.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

89.1.9.2.9 TVirtualListAction.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

89.1.9.2.10 TVirtualListAction.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

89.1.9.3 TVirtualListAction Events

The events of the TVirtualListAction class are listed here.

Published Events

	Name	Description
	OnGetItem (see page 15790)	This is OnGetItem, a member of class TVirtualListAction.
	OnGetItemCount (see page 15790)	Occurs when the list action needs to determine the number of items it supplies to client controls.
	OnHint (see page 15791)	Occurs when the mouse pauses over a client control or menu item.
	OnItemSelected (see page 15791)	Occurs when a client control executes the action because a user selected one of the items.

89.1.9.3.1 TVirtualListAction.OnGetItem Event

Pascal

```
property OnGetItem: TGetVirtualItemEvent;
```

C++

```
__property TGetVirtualItemEvent OnGetItem;
```

Description

This is OnGetItem, a member of class TVirtualListAction.

89.1.9.3.2 TVirtualListAction.OnGetItemCount Event

Occurs when the list action needs to determine the number of items it supplies to client controls.

Pascal

```
property OnGetItemCount: TGetItemCountEvent;
```

C++

```
__property TGetItemCountEvent OnGetItemCount;
```

Description

The GetCount method generates an OnGetItemCount event to supply the value of the Count property. Write an OnGetItemCount event handler to indicate the number of items for which you are supplying values in the OnGetItem event handler.

Sender is the list action that is supplying data to client controls.

Count returns the number of items this list action supplies to client controls.

See Also

getcount (see page 15719)

count (see page 15722)

89.1.9.3.3 TVirtualListAction.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

89.1.9.3.4 TVirtualListAction.OnItemSelected Event

Occurs when a client control executes the action because a user selected one of the items.

Pascal

```
property OnItemSelected: TItemSelectedEvent;
```

C++

```
__property TItemSelectedEvent OnItemSelected;
```

Description

Write an OnItemSelected event handler to respond when the user selects one of the items supplied to client controls. Before this event is called, the list action first updates the ItemIndex property of any client controls that descend from TCustomListControl.

Sender is the list action that supplies items to the control in which an item was selected.

Control is the control in which the user selected an item.

See Also

ExecuteTarget (see page 15718)

ItemIndex (see page 15722)

89.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TListItemsSortType (see page 15792)	This is record ListActns.TListItemsSortType.

89.2.1 ListActns.TListItemsSortType Enumeration

Pascal

```
TListItemsSortType = (
  stNone,
  stData,
  stText,
  stBoth
);
```

C++

```
enum TListItemsSortType {
  stNone,
  stData,
  stText,
  stBoth
};
```

File

ListActns

Description

This is record ListActns.TListItemsSortType.

89.3 Types

The following table lists types in this documentation.

Types

Name	Description
TGetItemCountEvent (see page 15793)	This is type ListActns.TGetItemCountEvent.
TGetItemEvent (see page 15793)	This is type ListActns.TGetItemEvent.
TGetVirtualItemEvent (see page 15794)	This is type ListActns.TGetVirtualItemEvent.
TItemSelectedEvent (see page 15794)	This is type ListActns.TItemSelectedEvent.
TListCompareEvent (see page 15794)	This is type ListActns.TListCompareEvent.
TListControlItemClass (see page 15795)	This is type ListActns.TListControlItemClass.
TListItemsCompare (see page 15795)	This is type ListActns.TListItemsCompare.

89.3.1 ListActns.TGetItemCountEvent Type

Pascal

```
TGetItemCountEvent = procedure (Sender: TCustomListAction; var Count: Integer) of object;
```

C++

```
(Sender: TCustomListAction; var Count: Integer) ( TGetItemCountEvent)();
```

File

ListActns

Description

This is type ListActns.TGetItemCountEvent.

89.3.2 ListActns.TGetItemEvent Type

Pascal

```
TGetItemEvent = procedure (Sender: TCustomListAction; const Index: Integer; var Item:
TListControlItem) of object;
```

C++

```
(Sender: TCustomListAction; const Index: Integer; var Item: TListControlItem) (
TGetItemEvent)();
```

File

ListActns

Description

This is type ListActns.TGetItemEvent.

89.3.3 ListActns.TGetVirtualItemEvent Type

Pascal

```
TGetVirtualItemEvent = procedure (Sender: TCustomListAction; const Index: Integer; var Value:
String; var ImageIndex: Integer; var Data: Pointer) of object;
```

C++

```
(Sender: TCustomListAction; const Index: Integer; var Value: String; var ImageIndex: Integer;
var Data: Pointer) ( TGetVirtualItemEvent)();
```

File

ListActns

Description

This is type ListActns.TGetVirtualItemEvent.

89.3.4 ListActns.TItemSelectedEvent Type

Pascal

```
TItemSelectedEvent = procedure (Sender: TCustomListAction; Control: TControl) of object;
```

C++

```
(Sender: TCustomListAction; Control: TControl) ( TItemSelectedEvent)();
```

File

ListActns

Description

This is type ListActns.TItemSelectedEvent.

89.3.5 ListActns.TListCompareEvent Type

Pascal

```
TListCompareEvent = function (List: TListControlItems; Item1, Item2: TListControlItem):
Integer of object;
```

C++

```
(List: TListControlItems; Item1, Item2: TListControlItem): Integer ( TListCompareEvent)();
```

File

ListActns

Description

This is type ListActns.TListCompareEvent.

89.3.6 ListActns.TListControlItemClass Type

Pascal

```
TListControlItemClass = class of TListControlItem;
```

C++

```
TListControlItem TListControlItemClass;
```

File

ListActns

Description

This is type ListActns.TListControlItemClass.

89.3.7 ListActns.TListItemsCompare Type

Pascal

```
TListItemsCompare = function (List: TListControlItems; Index1, Index2: Integer): Integer;
```

C++

```
(List: TListControlItems; Index1, Index2: Integer): Integer ( TListItemsCompare)();
```

File

ListActns

Description

This is type ListActns.TListItemsCompare.

90 Mask Namespace

This is namespace Mask.

Topics

Name	Description
Classes (see page 15798)	The following table lists classes in this documentation.
Types (see page 15895)	The following table lists types in this documentation.

Classes

	Name	Description
	EDBEditError (see page 15798)	EDBEditError is the exception class for data that are incompatible with a specified mask.
	TCustomMaskEdit (see page 15798)	TCustomMaskEdit is the base type for edit controls that provide a mask for checking the validity of text typed into the control.
	TMaskEdit (see page 15824)	TMaskEdit implements a generic masked edit control.

Types

Name	Description
TMaskedState (see page 15895)	TMaskedState is a set of flags that describe the current state of a masked edit control.

90.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EDBEditError (see page 15798)	EDBEditError is the exception class for data that are incompatible with a specified mask.
	TCustomMaskEdit (see page 15798)	TCustomMaskEdit is the base type for edit controls that provide a mask for checking the validity of text typed into the control.
	TMaskEdit (see page 15824)	TMaskEdit implements a generic masked edit control.

90.1.1 EDBEditError Class

EDBEditError is the exception class for data that are incompatible with a specified mask.

Class Hierarchy

Pascal

```
EDBEditError = class(Exception);
```

C++

```
class EDBEditError : public Exception;
```

File

Mask

Description

EDBEditError is raised when an application attempts to use data that are incompatible with the mask specified for a field.

A mask restricts the data that can be put into an edit box or data field. Use the EditMask property to specify a mask.

See Also

TCustomMaskEdit (see page 15798)

EditMask

90.1.2 TCustomMaskEdit Class

TCustomMaskEdit is the base type for edit controls that provide a mask for checking the validity of text typed into the control.

Class Hierarchy

Pascal

```
TCustomMaskEdit = class(TCustomEdit);
```

C++

```
class TCustomMaskEdit : public TCustomEdit;
```

File

Mask

Description

TCustomMaskEdit introduces new properties and methods to expand the capabilities of TCustomEdit for validity checking using an edit mask. To create an edit control that takes advantage of the masking capabilities of TCustomMaskEdit, use a TMaskEdit (see page 15824) object. To create an edit control that does not validate the input, use the TEdit object instead.

Do not create instances of TCustomMaskEdit. Use TCustomMaskEdit as a base class when declaring control objects that implement a masked edit control. Properties and methods of TCustomMaskEdit provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

EDBEditError (see page 15798)

TDBEdit (see page 8735)

TEdit (see page 21402)

TMaskEdit (see page 15824)

90.1.2.1 TCustomMaskEdit Members

The following tables list the members exposed by TCustomMaskEdit.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnChange (see page 21275)	Occurs when the text for the edit control may have changed.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.

	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.

	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21245)	Generates an OnChange (see page 21275) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CheckCursor (see page 15811)	Moves the cursor to the closest editable character and selects it if nothing is currently selected.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 21248)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWindowHandle (see page 21248)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 21248)	Creates a Windows control corresponding to the edit control.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21251)	Destroys the Windows control that corresponds to the edit control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.

	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoSetMaxLength (see page 21251)	Sets the MaxLength (see page 21263) property.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EditCanModify (see page 15812)	Permits the user to edit control contents.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFirstEditChar (see page 15813)	Returns the index in the EditMask (see page 15821) of the first nonliteral character.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetLastEditChar (see page 15813)	Returns the index in the EditMask (see page 15821) of the last nonliteral character.
	GetMaxChars (see page 15814)	Returns the length of the EditText (see page 15821).
	GetNextEditChar (see page 15814)	Returns the first index of a nonliteral character equal to or greater than an offset.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetPriorEditChar (see page 15815)	Returns the first index of a nonliteral character equal to or less than an offset.

	GetSel (see page 15815)	Returns the endpoints of the current selection in the EditText (see page 15821).
V	GetSelLength (see page 21252)	Returns the SelLength (see page 21265) property.
V	GetSelStart (see page 21252)	Returns the SelStart (see page 21269) property.
V	GetSelText (see page 21252)	Returns the SelText (see page 21272) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
V	KeyDown (see page 15816)	Supplements the inherited KeyDown to process navigation keys and the Delete key.
V	KeyPress (see page 15816)	Supplements the inherited method to process the Esc and Enter keys.
V	KeyUp (see page 15817)	Key up event dispatcher.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReformatText (see page 15817)	Applies the mask NewMask to the underlying text, and updates the internal representation of the mask.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
V	Reset (see page 15818)	Returns the EditText (see page 15821) to its value at the time when the edit control last lost focus.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.

	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetAutoSize (see page 21257)	Sets the AutoSize (see page 21260) property.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetCursor (see page 15818)	Moves the cursor to the first editable character at or before Pos, and selects that character.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSel (see page 15819)	Selects character range.
V	SetSelLength (see page 21257)	Sets the SelLength (see page 21265) property.
V	SetSelStart (see page 21257)	Sets the SelStart (see page 21269) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	Validate (see page 15819)	Validates a specified string against the current mask.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateError (see page 15820)	Beeps and raises an EDBEditError (see page 15798) exception.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSelect (see page 21260)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AutoSize (see page 21260)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 21261)	Determines whether the edit control has a single line border around the client area.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	CharCase (see page 21262)	Determines the case of the text within the edit control.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EditMask (see page 15821)	Specifies the mask that represents what text is valid for the masked edit control.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	HideSelection (see page 21262)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MaskState (see page 15822)	Indicates state information about how the mask is being applied.
	MaxLength (see page 15822)	Specifies the maximum number of characters that can appear in the edit control.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	OEMConvert (see page 21264)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 21264)	Determines where a control looks for its color information.

	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PasswordChar (see page 21265)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 15812)	Sets the Text (see page 15823) property to an empty string.
	ClearSelection (see page 21245)	Removes the selected text from the edit control.
	ClearUndo (see page 21246)	Clears the undo buffer so that no changes to the text can be backed out.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopyToClipboard (see page 21246)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
V	Create (see page 15810)	Creates an instance of TCustomMaskEdit (see page 15798).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CutToClipboard (see page 21249)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 21250)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.

	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetSelTextBuf (see page 21253)	Copies the selected text into a buffer and returns the number of characters copied.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 15816)	Returns the length of the underlying text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	PasteFromClipboard (see page 21254)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectAll (see page 21255)	Selects all text in the edit control.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSelTextBuf (see page 21258)	Replaces the selected text with a null-terminated string.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Undo (see page 21259)	Backs out all changes in the undo buffer.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.
V	ValidateEdit (see page 15819)	Validates the EditText (see page 15821) against the current mask.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	CanUndo (see page 21262)	Indicates whether the edit control contains changes that can be backed out.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EditText (see page 15821)	Specifies the text that appears in the window of the masked edit control.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.

R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
R	IsMasked (see page 15822)	Indicates whether a mask exists for the edit control.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Modified (see page 21263)	Indicates whether the user edited the text of the edit control.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ReadOnly (see page 21265)	Determines whether the user can change the text of the edit control.
	SelLength (see page 21265)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 21269)	Specifies the position of the first selected character in the text.
	SelText (see page 21272)	Specifies the selected portion of the edit control's text.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	Text (see page 15823)	Denotes the underlying text for the masked edit control before the mask has been applied.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	TabStop (see page 21273)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

90.1.2.2 TCustomMaskEdit.Create Constructor

Creates an instance of TCustomMaskEdit (see page 15798).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomMaskEdit(TComponent * AOwner);
```

Description

Call Create to instantiate a masked edit control at runtime. AOwner is the component that is responsible for freeing the masked edit control (typically the form). It becomes the value of the Owner property.

See Also

Owner (see page 2807)

90.1.2.3 TCustomMaskEdit Methods

The methods of the TCustomMaskEdit class are listed here.

Protected Methods

	Name	Description
	CheckCursor (see page 15811)	Moves the cursor to the closest editable character and selects it if nothing is currently selected.
V	EditCanModify (see page 15812)	Permits the user to edit control contents.
	GetFirstEditChar (see page 15813)	Returns the index in the EditMask (see page 15821) of the first nonliteral character.
	GetLastEditChar (see page 15813)	Returns the index in the EditMask (see page 15821) of the last nonliteral character.
	GetMaxChars (see page 15814)	Returns the length of the EditText (see page 15821).
	GetNextEditChar (see page 15814)	Returns the first index of a nonliteral character equal to or greater than an offset.
	GetPriorEditChar (see page 15815)	Returns the first index of a nonliteral character equal to or less than an offset.
	GetSel (see page 15815)	Returns the endpoints of the current selection in the EditText (see page 15821).
V	KeyDown (see page 15816)	Supplements the inherited KeyDown to process navigation keys and the Delete key.
V	KeyPress (see page 15816)	Supplements the inherited method to process the Esc and Enter keys.
V	KeyUp (see page 15817)	Key up event dispatcher.
	ReformatText (see page 15817)	Applies the mask NewMask to the underlying text, and updates the internal representation of the mask.
V	Reset (see page 15818)	Returns the EditText (see page 15821) to its value at the time when the edit control last lost focus.
	SetCursor (see page 15818)	Moves the cursor to the first editable character at or before Pos, and selects that character.
	SetSel (see page 15819)	Selects character range.
V	Validate (see page 15819)	Validates a specified string against the current mask.
V	ValidateError (see page 15820)	Beeps and raises an EDBEditError (see page 15798) exception.

Public Methods

	Name	Description
V	Clear (see page 15812)	Sets the Text (see page 15823) property to an empty string.
	GetTextLen (see page 15816)	Returns the length of the underlying text.
V	ValidateEdit (see page 15819)	Validates the EditText (see page 15821) against the current mask.

90.1.2.3.1 TCustomMaskEdit.CheckCursor Method

Moves the cursor to the closest editable character and selects it if nothing is currently selected.

Pascal

```
procedure CheckCursor;
```

C++

```
__fastcall CheckCursor();
```

Description

CheckCursor is called whenever the position of the cursor may have changed. It ensures that at least one character is always selected. Because editing a masked edit control consists of replacing the blank characters in the mask, there must always be a selection to replace. The selection corresponds to one or more nonliteral characters in the mask, which are either blank or contain characters for the user to reenter.

See Also

GetSel (see page 15815)

SetCursor (see page 15818)

SetSel (see page 15819)

90.1.2.3.2 TCustomMaskEdit.Clear Method

Sets the Text (see page 15823) property to an empty string.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Use Clear to remove all text that was entered into the masked edit control. Clear sets the value of Text (see page 15823) to an empty string. It does not clear the EditMask (see page 15821). If the edit control has a mask, EditText (see page 15821) is not empty, but has all non-literal characters replaced by blanks.

See Also

EditMask (see page 15821)

EditText (see page 15821)

IsMasked (see page 15822)

Reset (see page 15818)

Text (see page 15823)

90.1.2.3.3 TCustomMaskEdit.EditCanModify Method

Permits the user to edit control contents.

Pascal

```
function EditCanModify: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean EditCanModify();
```

Description

The EditCanModify method defined by TCustomMaskEdit (see page 15798) always returns true, allowing the Text (see page 15823) to be edited. EditCanModify exists primarily to provide an interface for a method that can conditionally disallow changes to the Text (see page 15823) by returning false.

See Also

Text (see page 15823)

90.1.2.3.4 TCustomMaskEdit.GetFirstEditChar Method

Returns the index in the EditMask (see page 15821) of the first nonliteral character.

Pascal

```
function GetFirstEditChar: Integer;
```

C++

```
__fastcall int GetFirstEditChar();
```

Description

Use GetFirstEditChar to find the first position of the EditMask (see page 15821) where the user can modify the EditText (see page 15821). 0 specifies the first character, 1 the second character, and so on. If there is no mask, GetFirstEditChar returns zero.

Use GetFirstEditChar as a starting point for programmatically processing all the nonliteral characters in the EditText (see page 15821). Use GetNextEditChar (see page 15814) to continue to loop through the editable characters in the EditText (see page 15821), and GetLastEditChar (see page 15813) to determine when all nonliteral characters have been processed.

See Also

EditMask (see page 15821)

EditText (see page 15821)

GetLastEditChar (see page 15813)

GetNextEditChar (see page 15814)

GetPriorEditChar (see page 15815)

90.1.2.3.5 TCustomMaskEdit.GetLastEditChar Method

Returns the index in the EditMask (see page 15821) of the last nonliteral character.

Pascal

```
function GetLastEditChar: Integer;
```

C++

```
__fastcall int GetLastEditChar();
```

Description

Use GetLastEditChar to find the last position of the EditMask (see page 15821) where the user can modify the EditText (see page 15821). 0 specifies the first character, 1 the second character, and so on. If there is no mask, GetLastEditChar returns the length of the text.

See Also

EditMask (see page 15821)

EditText (see page 15821)

GetFirstEditChar (see page 15813)

GetMaxChars (see page 15814)

GetNextEditChar (see page 15814)

GetPriorEditChar (see page 15815)

90.1.2.3.6 TCustomMaskEdit.GetMaxChars Method

Returns the length of the EditText (see page 15821).

Pascal

```
function GetMaxChars: Integer;
```

C++

```
__fastcall int GetMaxChars();
```

Description

If there is a mask, GetMaxChars returns the length of the EditText (see page 15821), including all blanks. This is the same as MaxLength (see page 15822), because editing in a masked edit control consists of replacing nonliteral characters. If there is no mask, GetMaxChars returns the length of the text in the window. In this case, the value returned by GetMaxChars may be less than MaxLength (see page 15822), which is the maximum number of characters that can be entered into the edit control.

See Also

EditText (see page 15821)

GetLastEditChar (see page 15813)

MaxLength (see page 15822)

Text (see page 15823)

90.1.2.3.7 TCustomMaskEdit.GetNextEditChar Method

Returns the first index of a nonliteral character equal to or greater than an offset.

Pascal

```
function GetNextEditChar(Offset: Integer): Integer;
```

C++

```
__fastcall int GetNextEditChar(int Offset);
```

Description

Call GetNextEditChar to find the first position of the EditMask (see page 15821) at or following Offset where the user can modify the EditText (see page 15821). 0 specifies the first character, 1 the second character, and so on. If there is no nonliteral character at or following Offset, GetNextEditChar returns MaxLength (see page 15822).

Use GetNextEditChar to loop through the editable characters in the EditText (see page 15821), starting with the value returned by GetFirstEditChar (see page 15813) and using GetLastEditChar (see page 15813) to determine when all nonliteral characters have been processed.

See Also

EditMask (see page 15821)

EditText (see page 15821)

GetFirstEditChar (see page 15813)

GetLastEditChar (see page 15813)

GetPriorEditChar (see page 15815)

90.1.2.3.8 TCustomMaskEdit.GetPriorEditChar Method

Returns the first index of a nonliteral character equal to or less than an offset.

Pascal

```
function GetPriorEditChar(Offset: Integer): Integer;
```

C++

```
__fastcall int GetPriorEditChar(int Offset);
```

Description

Call GetPriorEditChar to find the first position of the EditMask (see page 15821) at or preceding Offset, where the user can modify the EditText (see page 15821). 0 specifies the first character, 1 the second character, and so on. If there is no nonliteral character at or preceding Offset, GetPriorEditChar returns the index of the first editable character following Offset.

See Also

EditMask (see page 15821)

EditText (see page 15821)

GetFirstEditChar (see page 15813)

GetLastEditChar (see page 15813)

GetNextEditChar (see page 15814)

90.1.2.3.9 TCustomMaskEdit.GetSel Method

Returns the endpoints of the current selection in the EditText (see page 15821).

Pascal

```
procedure GetSel(var SelStart: Integer; var SelStop: Integer);
```

C++

```
__fastcall GetSel(int SelStart, int SelStop);
```

Description

Use GetSel to obtain the indexes of the first and last selected characters of the EditText (see page 15821), where 0 is the index of the first character. Not all the characters in the selection will necessarily be editable, if the mask contains literal characters. Do not use the resulting indexes on the Text (see page 15823) property if there is a mask, because the EditText (see page 15821) may contain blanks or nonliteral characters not found in the Text (see page 15823).

To programmatically change the selected text, use GetNextEditChar (see page 15814) to cycle through the editable characters of the selection from SelStart to SelStop.

See Also

EditText (see page 15821)

GetNextEditChar (see page 15814)

SelText (see page 21272)

SetSel (see page 15819)

Text (see page 15823)

90.1.2.3.10 TCustomMaskEdit.GetTextLen Method

Returns the length of the underlying text.

Pascal

```
function GetTextLen: Integer;
```

C++

```
__fastcall int GetTextLen();
```

Description

Use GetTextLen to determine the length of the underlying text for the masked edit control. The underlying text has not been formatted for display by the mask, and may differ from the EditText (see page 15821) that appears in the window of the control.

See Also

EditText (see page 15821)

GetTextBuf (see page 5766)

MaxLength (see page 15822)

Text (see page 15823)

90.1.2.3.11 TCustomMaskEdit.KeyDown Method

Supplements the inherited KeyDown to process navigation keys and the Delete key.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

After calling the inherited KeyDown, KeyDown ensures that moving the cursor within the masked edit control places the cursor on an editable character, and that deleting the current selection replaces nonliteral characters with blanks. Derived objects can override this method to further supplement the processing that takes place when the user presses a key down.

See Also

CheckCursor (see page 15811)

KeyPress (see page 15816)

KeyUp (see page 15817)

SetSel (see page 15819)

90.1.2.3.12 TCustomMaskEdit.KeyPress Method

Supplements the inherited method to process the Esc and Enter keys.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

After calling the inherited method, KeyPress calls the Reset (see page 15818) method in response to the Esc key, or calls the ValidateEdit (see page 15819) method in response to the Enter key. Derived classes can override this method to further supplement the processing of specific keystrokes.

See Also

KeyDown (see page 15816)

Reset (see page 15818)

ValidateEdit (see page 15819)

90.1.2.3.13 TCustomMaskEdit.KeyUp Method

Key up event dispatcher.

Pascal

```
procedure KeyUp(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyUp(Word Key, TShiftState Shift);
```

Description

After calling the inherited method, KeyUp makes sure that selection using the Ctrl+Left and Ctrl+Right key combinations ends with an editable selection in the edit window. Derived classes can override this method to further supplement the processing that takes place when the user releases a key.

See Also

CheckCursor (see page 15811)

KeyDown (see page 15816)

KeyPress (see page 15816)

SetSel (see page 15819)

90.1.2.3.14 TCustomMaskEdit.ReformatText Method

Applies the mask NewMask to the underlying text, and updates the internal representation of the mask.

Pascal

```
procedure ReformatText(const NewMask: string);
```

C++

```
__fastcall ReformatText(const AnsiString NewMask);
```

Description

ReformatText does the bulk of the work of setting the EditMask (see page 15821) property at runtime. ReformatText should not be used in place of the EditMask (see page 15821) property, however, as it does not deal with the MaskState (see page

15822) property or the positioning of the cursor.

See Also

CheckCursor (see page 15811)

EditMask (see page 15821)

EditText (see page 15821)

MaskState (see page 15822)

Text (see page 15823)

90.1.2.3.15 TCustomMaskEdit.Reset Method

Returns the EditText (see page 15821) to its value at the time when the edit control last lost focus.

Pascal

```
procedure Reset; virtual;
```

C++

```
virtual __fastcall Reset();
```

Description

Reset cancels the most recent edits to the masked edit control. The EditText (see page 15821) returns to the value at the time the control last lost focus, not the time when it first regained focus. Thus, any changes to the EditText (see page 15821) that occurred programmatically while the edit control did not have focus are also discarded.

See Also

Clear (see page 15812)

EditText (see page 15821)

KeyPress (see page 15816)

90.1.2.3.16 TCustomMaskEdit.SetCursor Method

Moves the cursor to the first editable character at or before Pos, and selects that character.

Pascal

```
procedure SetCursor(Pos: Integer);
```

C++

```
__fastcall SetCursor(int Pos);
```

Description

SetCursor handles positioning of the cursor within the EditText (see page 15821) so that it is always on an editable character. It also ensures that at least one character is always selected. Because editing a masked edit control consists of replacing the blank characters in the mask, there must always be a selection to replace. The selection corresponds to one or more nonliteral characters in the mask, which are either blank or contain characters for the user to reenter.

See Also

CheckCursor (see page 15811)

GetPriorEditChar (see page 15815)

SetSel (see page 15819)

90.1.2.3.17 TCustomMaskEdit.SetSel Method

Selects character range.

Pascal

```
procedure SetSel(SelStart: Integer; SelStop: Integer);
```

C++

```
__fastcall SetSel(int SelStart, int SelStop);
```

Description

SetSel selects the indicated range of characters in the EditText (see page 15821), regardless of whether any of them are editable. If SelStart is the same as SelStop, SetSel positions the cursor at the SelStart character. When using SetSel, leave the cursor in an editable position if the intent is to create a selection for the user to replace.

SetSel can be used to select non-editable characters that can then be captured by using the SelText property or by copying to the Clipboard. Once the selection has been captured, return the cursor to an editable position by calling CheckCursor (see page 15811).

See Also

CheckCursor (see page 15811)

EditText (see page 15821)

GetSel (see page 15815)

SelText (see page 21272)

SetCursor (see page 15818)

90.1.2.3.18 TCustomMaskEdit.Validate Method

Validates a specified string against the current mask.

Pascal

```
function Validate(const Value: string; var Pos: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean Validate(const AnsiString Value, int Pos);
```

Description

The ValidateEdit (see page 15819) method calls Validate to perform the actual validation of the current text against the edit mask. It returns true if the string specified by Value is consistent with the edit mask, false if it detects an invalid character. If ValidateEdit (see page 15819) returns false, it returns the position of the invalid character as the Pos parameter.

See Also

EditMask (see page 15821)

ValidateEdit (see page 15819)

90.1.2.3.19 TCustomMaskEdit.ValidateEdit Method

Validates the EditText (see page 15821) against the current mask.

Pascal

```
procedure ValidateEdit; virtual;
```

C++

```
virtual __fastcall ValidateEdit();
```

Description

ValidateEdit attempts to validate the EditText (see page 15821) after each keystroke, and when the user attempts to leave the masked edit field. If the EditText (see page 15821) does not match the specifications of the EditMask (see page 15821), ValidateEdit calls ValidateError (see page 15820) to raise an exception, and returns the cursor to the first position where the EditText (see page 15821) did not comply with the EditMask (see page 15821).

See Also

EditMask (see page 15821)

EditText (see page 15821)

KeyPress (see page 15816)

ValidateError (see page 15820)

90.1.2.3.20 TCustomMaskEdit.ValidateError Method

Beeps and raises an EDBEditError (see page 15798) exception.

Pascal

```
procedure ValidateError; virtual;
```

C++

```
virtual __fastcall ValidateError();
```

Description

ValidateEdit (see page 15819) calls ValidateError when the EditText (see page 15821) fails to comply with the EditMask (see page 15821).

See Also

EditMask (see page 15821)

EditText (see page 15821)

ValidateEdit (see page 15819)

90.1.2.4 TCustomMaskEdit Properties

The properties of the TCustomMaskEdit class are listed here.

Protected Properties

	Name	Description
	EditMask (see page 15821)	Specifies the mask that represents what text is valid for the masked edit control.
	MaskState (see page 15822)	Indicates state information about how the mask is being applied.
	MaxLength (see page 15822)	Specifies the maximum number of characters that can appear in the edit control.

Public Properties

	Name	Description
	EditText (see page 15821)	Specifies the text that appears in the window of the masked edit control.
R	IsMasked (see page 15822)	Indicates whether a mask exists for the edit control.
	Text (see page 15823)	Denotes the underlying text for the masked edit control before the mask has been applied.

90.1.2.4.1 TCustomMaskEdit.EditMask Property

Specifies the mask that represents what text is valid for the masked edit control.

Pascal

```
property EditMask: TEditMask;
```

C++

```
__property TEditMask EditMask;
```

Description

Use EditMask to restrict the characters a user can enter into the masked edit control to valid characters and formats. If the user attempts to enter an invalid character, the edit control does not accept the character. Validation is performed on a character-by-character basis by the ValidateEdit (see page 15819) method.

Setting EditMask to an empty string removes the mask.

See Also

EditText (see page 15821)

IsMasked (see page 15822)

MaskState (see page 15822)

Text (see page 15823)

ValidateEdit (see page 15819)

ValidateError (see page 15820)

90.1.2.4.2 TCustomMaskEdit.EditText Property

Specifies the text that appears in the window of the masked edit control.

Pascal

```
property EditText: string;
```

C++

```
__property AnsiString EditText;
```

Description

EditText is the value of the text for the edit control after it is formatted using the EditMask (see page 15821). Before the text is fully entered into a masked edit control, the EditText includes a blank character for each unentered character. As characters are entered, the blank characters in the EditText are replaced by the values that the user types.

If literal mask characters are included in the text, and no character is substituted for blanks, the values of EditText and Text (see page 15823) are the same.

Note: EditText only applies at runtime.

See Also

EditMask (see page 15821)

Text (see page 15823)

90.1.2.4.3 TCustomMaskEdit.IsMasked Property

Indicates whether a mask exists for the edit control.

Pascal

```
property IsMasked: Boolean;
```

C++

```
__property Boolean IsMasked;
```

Description

Read IsMasked to determine whether the EditMask (see page 15821) property has been set. If IsMasked is true, the control has a mask, and input is validated at runtime. If IsMasked is false, the control does not restrict editing at runtime.

See Also

EditMask (see page 15821)

90.1.2.4.4 TCustomMaskEdit.MaskState Property

Indicates state information about how the mask is being applied.

Pascal

```
property MaskState: TMaskedState;
```

C++

```
__property TMaskedState MaskState;
```

Description

MaskState is used internally to determine appropriate actions based on how the mask is currently being applied. MaskState is a set drawn from among the following values:

Value	Meaning
msMasked	A mask has been assigned. MaskState includes msMasked when IsMasked (see page 15822) is true.
msReEnter	An invalid character has been rejected and the next edit should be a valid replacement character.
msDBSetText	The text comes from a database field and should not be validated.

MaskState is exposed as a protected property so that derived classes can maintain and use this state information in the implementation of properties and methods.

See Also

IsMasked (see page 15822)

ValidateEdit (see page 15819)

90.1.2.4.5 TCustomMaskEdit.MaxLength Property

Specifies the maximum number of characters that can appear in the edit control.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

MaxLength is the length of the EditText (see page 15821). Set MaxLength to limit the number of characters that can appear in the edit control when there is no EditMask (see page 15821). If there is an EditMask (see page 15821), MaxLength is implied by the mask itself, and cannot be changed. The EditText (see page 15821) contains blank characters for each character that has not been entered, padding the end or beginning of every optional section, so it remains constant in length.

See Also

EditMask (see page 15821)

EditText (see page 15821)

IsMasked (see page 15822)

Text (see page 15823)

90.1.2.4.6 TCustomMaskEdit.Text Property

Denotes the underlying text for the masked edit control before the mask has been applied.

Pascal

```
property Text: TMaskedText;
```

C++

```
__property TMaskedText Text;
```

Description

Use Text to determine the underlying value of the text before it has been formatted by the mask. Text may differ from the EditText (see page 15821) visible in the masked edit control if the mask specifies that literal characters should be removed, if the mask includes spaces, or if the mask includes characters that have not yet been filled in by the user. Text will not contain the blank characters, spaces will not be replaced by _, and literal characters in the mask will be removed if the mask indicates they should be.

When setting Text, the value of the text is formatted using the mask, and the resulting EditText (see page 15821) is used to update the window.

If there is no mask, Text is the text that appears in the edit control.

See Also

EditMask (see page 15821)

EditText (see page 15821)

GetTextLen (see page 15816)

IsMasked (see page 15822)

ReformatText (see page 15817)

90.1.3 TMaskEdit Class

TMaskEdit implements a generic masked edit control.

Class Hierarchy

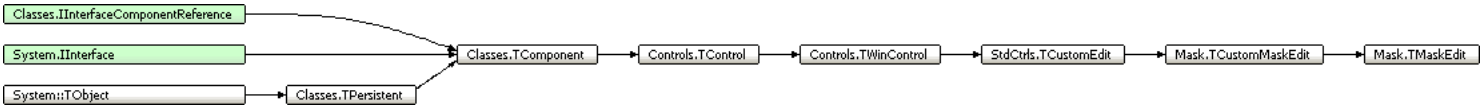

Pascal

```
TMaskEdit = class(TCustomMaskEdit);
```

C++

```
class TMaskEdit : public TCustomMaskEdit;
```

File

Mask

Description

Use a TMaskEdit object to put a masked edit control on your form. Masked edit controls validate the text the user enters against a mask that encodes the valid forms the text can take. The mask can also format text that is displayed to the user.

TMaskEdit implements the generic behavior introduced in TCustomMaskEdit (see page 15798). TMaskEdit publishes many of the properties and methods inherited from TCustomMaskEdit (see page 15798), but does not introduce any new behavior.

See Also

TDBEdit (see page 8735)

TEdit (see page 21402)

TCustomMaskEdit (see page 15798)

Edit Controls

90.1.3.1 TMaskEdit Members

The following tables list the members exposed by TMaskEdit.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.

	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21245)	Generates an OnChange (see page 21275) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CheckCursor (see page 15811)	Moves the cursor to the closest editable character and selects it if nothing is currently selected.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.

	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 21248)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWindowHandle (see page 21248)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CreateWnd (see page 21248)	Creates a Windows control corresponding to the edit control.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
V	DestroyWnd (see page 21251)	Destroys the Windows control that corresponds to the edit control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
V	DoSetMaxLength (see page 21251)	Sets the MaxLength (see page 21263) property.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.

	EditCanModify (see page 15812)	Permits the user to edit control contents.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFirstEditChar (see page 15813)	Returns the index in the EditMask (see page 15821) of the first nonliteral character.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetLastEditChar (see page 15813)	Returns the index in the EditMask (see page 15821) of the last nonliteral character.
	GetMaxChars (see page 15814)	Returns the length of the EditText (see page 15821).
	GetNextEditChar (see page 15814)	Returns the first index of a nonliteral character equal to or greater than an offset.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetPriorEditChar (see page 15815)	Returns the first index of a nonliteral character equal to or less than an offset.
	GetSel (see page 15815)	Returns the endpoints of the current selection in the EditText (see page 15821).
	GetSelLength (see page 21252)	Returns the SelLength (see page 21265) property.
	GetSelStart (see page 21252)	Returns the SelStart (see page 21269) property.
	GetSelText (see page 21252)	Returns the SelText (see page 21272) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.

	KeyDown (see page 6163)	Respond to key press events.
	KeyDown (see page 15816)	Supplements the inherited KeyDown to process navigation keys and the Delete key.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyPress (see page 15816)	Supplements the inherited method to process the Esc and Enter keys.
	KeyUp (see page 6164)	Respond to released key.
	KeyUp (see page 15817)	Key up event dispatcher.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReformatText (see page 15817)	Applies the mask NewMask to the underlying text, and updates the internal representation of the mask.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Reset (see page 15818)	Returns the EditText (see page 15821) to its value at the time when the edit control last lost focus.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetAutoSize (see page 21257)	Sets the AutoSize (see page 21260) property.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetCursor (see page 15818)	Moves the cursor to the first editable character at or before Pos, and selects that character.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSel (see page 15819)	Selects character range.
	SetSelLength (see page 21257)	Sets the SelLength (see page 21265) property.
	SetSelStart (see page 21257)	Sets the SelStart (see page 21269) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	Validate (see page 15819)	Validates a specified string against the current mask.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateError (see page 15820)	Beeps and raises an EDBEditError (see page 15798) exception.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.

	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	HideSelection (see page 21262)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MaskState (see page 15822)	Indicates state information about how the mask is being applied.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	OEMConvert (see page 21264)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 15812)	Sets the Text (see page 15823) property to an empty string.
V	Clear (see page 21245)	Deletes all text from the edit control.
	ClearSelection (see page 21245)	Removes the selected text from the edit control.
	ClearUndo (see page 21246)	Clears the undo buffer so that no changes to the text can be backed out.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopyToClipboard (see page 21246)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 15810)	Creates an instance of TCustomMaskEdit (see page 15798).
V	Create (see page 21244)	Creates an instance of TCustomEdit (see page 21232).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.

	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CutToClipboard (see page 21249)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 21250)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
V	GetSelTextBuf (see page 21253)	Copies the selected text into a buffer and returns the number of characters copied.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.

	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	GetTextLen (see page 15816)	Returns the length of the underlying text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	PasteFromClipboard (see page 21254)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectAll (see page 21255)	Selects all text in the edit control.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.

V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSelTextBuf (see page 21258)	Replaces the selected text with a null-terminated string.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Undo (see page 21259)	Backs out all changes in the undo buffer.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.
V	ValidateEdit (see page 15819)	Validates the EditText (see page 15821) against the current mask.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	CanUndo (see page 21262)	Indicates whether the edit control contains changes that can be backed out.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EditText (see page 15821)	Specifies the text that appears in the window of the masked edit control.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.

R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
R	IsMasked (see page 15822)	Indicates whether a mask exists for the edit control.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Modified (see page 21263)	Indicates whether the user edited the text of the edit control.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelLength (see page 21265)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 21269)	Specifies the position of the first selected character in the text.
	SelText (see page 21272)	Specifies the selected portion of the edit control's text.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 15866)	Occurs when the text for the edit control may have changed.
	OnClick (see page 15866)	Occurs when the user clicks the control.
	OnDblClick (see page 15868)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 15869)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 15871)	Occurs when the user drags an object over a control.
	OnEndDock (see page 15873)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 15874)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 15875)	Occurs when a control receives the input focus.
	OnExit (see page 15877)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 15879)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 15882)	Occurs when a key is pressed.
	OnKeyUp (see page 15883)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 15884)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 15885)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 15888)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 15888)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 15888)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 15890)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 15893)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 15893)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 15837)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 15840)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 15841)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 15841)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 15842)	Specifies which edges of the control are beveled.
	BevelInner (see page 15843)	Specifies the cut of the inner bevel.
	BevelKind (see page 15843)	Specifies the control's bevel style.
	BevelOuter (see page 15843)	Specifies the cut of the outer bevel.
	BevelWidth (see page 15844)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 15844)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 15845)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 15845)	Determines the case of the text within the edit control.
	Color (see page 15845)	Specifies the background color of the control.
	Constraints (see page 15846)	Specifies the size constraints for the control.
	Ctl3D (see page 15847)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 15847)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 15848)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 15848)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EditMask (see page 15850)	Specifies the mask that represents what text is valid for the masked edit control.
	Enabled (see page 15850)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 15851)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 15852)	Determines the behavior of the input method editor (IME).
	ImeName (see page 15852)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	MaxLength (see page 15853)	Specifies the maximum number of characters that can appear in the edit control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 15853)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 15854)	Determines where a control looks for its color information.
	ParentCtl3D (see page 15854)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 15854)	Determines where a control looks for its font information.
	ParentShowHint (see page 15855)	Determines where a control looks to find out if its Help Hint should be shown.
	PasswordChar (see page 15856)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 15856)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 15861)	Determines whether the user can change the text of the edit control.
	ShowHint (see page 15861)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

	TabOrder (see page 15862)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 15863)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Text (see page 15863)	Denotes the underlying text for the masked edit control before the mask has been applied.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 15864)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

90.1.3.2 TMaskEdit Properties

The properties of the TMaskEdit class are listed here.

Published Properties

	Name	Description
	Align (see page 15837)	Determines how the control aligns within its container (parent control).
	Anchors (see page 15840)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 15841)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 15841)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 15842)	Specifies which edges of the control are beveled.
	BevelInner (see page 15843)	Specifies the cut of the inner bevel.
	BevelKind (see page 15843)	Specifies the control's bevel style.
	BevelOuter (see page 15843)	Specifies the cut of the outer bevel.
	BevelWidth (see page 15844)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 15844)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 15845)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 15845)	Determines the case of the text within the edit control.
	Color (see page 15845)	Specifies the background color of the control.
	Constraints (see page 15846)	Specifies the size constraints for the control.
	Ctl3D (see page 15847)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 15847)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 15848)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 15848)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EditMask (see page 15850)	Specifies the mask that represents what text is valid for the masked edit control.
	Enabled (see page 15850)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 15851)	Controls the attributes of text written on or in the control.
	ImeMode (see page 15852)	Determines the behavior of the input method editor (IME).
	ImeName (see page 15852)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	MaxLength (see page 15853)	Specifies the maximum number of characters that can appear in the edit control.
	ParentBiDiMode (see page 15853)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 15854)	Determines where a control looks for its color information.
	ParentCtl3D (see page 15854)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 15854)	Determines where a control looks for its font information.
	ParentShowHint (see page 15855)	Determines where a control looks to find out if its Help Hint should be shown.
	PasswordChar (see page 15856)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 15856)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 15861)	Determines whether the user can change the text of the edit control.

	ShowHint (see page 15861)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 15862)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 15863)	Determines if the user can tab to a control.
	Text (see page 15863)	Denotes the underlying text for the masked edit control before the mask has been applied.
	Visible (see page 15864)	Determines whether the component appears onscreen.

90.1.3.2.1 TMaskEdit.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);
```

```
  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
```

```
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
```

```
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

90.1.3.2.2 TMaskEdit.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at

design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

90.1.3.2.3 TMaskEdit.AutoSelect Property

Determines whether all the text in the edit control is automatically selected when the control gets focus.

Pascal

```
property AutoSelect: Boolean;
```

C++

```
__property Boolean AutoSelect;
```

Description

Set AutoSelect to select all the text when the edit control gets focus. AutoSelect only applies to single-line edit controls.

Use AutoSelect when the user is more likely to replace the text in the edit control than to append to it.

See Also

SelectAll (see page 21255)

SelLength (see page 21265)

SelStart (see page 21269)

SelText (see page 21272)

SetSelTextBuf (see page 21258)

90.1.3.2.4 TMaskEdit.AutoSize Property

Determines whether the height of the edit control automatically resizes to accommodate the text.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;__property Boolean AutoSize;
```

Description

Use AutoSize to make the edit control adjust its size automatically so the client area accommodates the height of the text. When AutoSize is false, the edit control has a fixed height. When AutoSize is true, the size of the control is readjusted whenever a change occurs that could affect the height of the control, such as a change to the font or border style.

Note: AutoSize only adjusts the height of the edit control. Even with AutoSize set to true, text in the edit control may appear truncated if it extends beyond the width of the control.

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit1->AutoSize = true;
  Edit1->Font->Size = 20;
  Edit1->Text = "The edit box is taller now";
}
```

Delphi Examples:

```
{
{
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.AutoSize := True;
  Edit1.Font.Size := 20;
  Edit1.Text := 'The edit box is taller now';
end;
```

See Also

BorderStyle (see page 21261)

ClientHeight (see page 5808)

ControlStyle (see page 5814)

Height (see page 5825)

90.1.3.2.5 TMaskEdit.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

90.1.3.2.6 TMaskEdit.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

90.1.3.2.7 TMaskEdit.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

90.1.3.2.8 TMaskEdit.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

90.1.3.2.9 TMaskEdit.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

90.1.3.2.10 TMaskEdit.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

90.1.3.2.11 TMaskEdit.BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the edit control stands out. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the edit control has a single-line border around the client area. If BorderStyle is bsNone, there will be no border.

90.1.3.2.12 TMaskEdit.CharCase Property

Determines the case of the text within the edit control.

Pascal

```
property CharCase: TEditCharCase;
```

C++

```
__property TEditCharCase CharCase;
```

Description

Use CharCase to force the contents of the edit control to assume a particular case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types them into the edit control. Changing the CharCase property to ecLowerCase or ecUpperCase changes the actual contents of the text, not just the appearance. Any case information is lost and can't be recaptured by changing CharCase to ecNormal.

See Also

Text (see page 5842)

90.1.3.2.13 TMaskEdit.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

90.1.3.2.14 TMaskEdit.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the

ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

90.1.3.2.15 TMaskEdit.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

90.1.3.2.16 TMaskEdit.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

90.1.3.2.17 TMaskEdit.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

90.1.3.2.18 TMaskEdit.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
```

```
    if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
      AttachMode = naAddChild;
    else if (HT.Contains(htNowhere))
      AttachMode = naAdd;
    else if (HT.Contains(htOnIndent))
      AttachMode = naInsert;
    else
      return;
    TreeView1->Selected->MoveTo(Item, AttachMode);
  }

  void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
        int X, int Y, TDragState State, bool &Accept)
  {
    Accept = (Source == (TObject *)TreeView1);
  }
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

90.1.3.2.19 TMaskEdit.EditMask Property

Specifies the mask that represents what text is valid for the masked edit control.

Pascal

```
property EditMask: TEditMask;
```

C++

```
__property TEditMask EditMask;
```

Description

Use EditMask to restrict the characters a user can enter into the masked edit control to valid characters and formats. If the user attempts to enter an invalid character, the edit control does not accept the character. Validation is performed on a character-by-character basis by the ValidateEdit method.

Setting EditMask to an empty string removes the mask.

See Also

EditText (see page 15821)

IsMasked (see page 15822)

MaskState (see page 15822)

Text (see page 15823)

ValidateEdit (see page 15819)

ValidateError (see page 15820)

90.1.3.2.20 TMaskEdit.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

90.1.3.2.21 TMaskEdit.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

90.1.3.2.22 TMaskEdit.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

90.1.3.2.23 TMaskEdit.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

90.1.3.2.24 TMaskEdit.MaxLength Property

Specifies the maximum number of characters that can appear in the edit control.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;__property int MaxLength;
```

Description

MaxLength is the length of the EditText. Set MaxLength to limit the number of characters that can appear in the edit control when there is no EditMask. If there is an EditMask, MaxLength is implied by the mask itself, and cannot be changed. The EditText contains blank characters for each character that has not been entered, padding the end or beginning of every optional section, so it remains constant in length.

See Also

EditMask (see page 15821)

EditText (see page 15821)

IsMasked (see page 15822)

Text (see page 15823)

90.1.3.2.25 TMaskEdit.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

90.1.3.2.26 TMaskEdit.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

90.1.3.2.27 TMaskEdit.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

90.1.3.2.28 TMaskEdit.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

90.1.3.2.29 TMaskEdit.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

90.1.3.2.30 TMaskEdit.PasswordChar Property

Indicates the character, if any, to display in place of the actual characters typed in the control.

Pascal

```
property PasswordChar: Char;
```

C++

```
__property Char PasswordChar;
```

Description

Use the PasswordChar property to create an edit control that displays a special character in place of any entered text. If PasswordChar is set to the null character (ANSI character zero), the edit control displays its text normally. If PasswordChar is any other character, the edit control displays PasswordChar in place of each character typed. PasswordChar affects the appearance of the edit control only. The value of the Text property reflects the actual characters that are typed.

See Also

Text (see page 5842)

90.1.3.2.31 TMaskEdit.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
```

```
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

```
void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
```

```
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;
```

```
implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

90.1.3.2.32 TMaskEdit.ReadOnly Property

Determines whether the user can change the text of the edit control.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

Setting ReadOnly to true ensures that the text is not altered, while still allowing the user to select text. The selected text can then be manipulated by the application, or copied to the Clipboard.

See Also

CopyToClipboard (see page 21246)

GetSelTextBuf (see page 21253)

90.1.3.2.33 TMaskEdit.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;
```

```
    char string[10];
    char index[3];
    for(int i = 1; i <= 10; i++)
    {
      strcpy(string, "Item");
      itoa(i, index, 10);
      strcat(string, index);
      ListBox1->Items->Add(string);
    }
    ListBox1->Hint = "Select an item";
    ListBox1->ShowHint = true;
    Application->ShowHint = true;
  }
  /*
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  */
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

90.1.3.2.34 TMaskEdit.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

90.1.3.2.35 TMaskEdit.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

90.1.3.2.36 TMaskEdit.Text Property

Denotes the underlying text for the masked edit control before the mask has been applied.

Pascal

```
property Text: TCaption;
```

C++

```
__property TMaskedText Text;__property TCaption Text;
```

Description

Use Text to determine the underlying value of the text before it has been formatted by the mask. Text may differ from the EditText visible in the masked edit control if the mask specifies that literal characters should be removed, if the mask includes spaces, or if the mask includes characters that have not yet been filled in by the user. Text will not contain the blank characters, spaces will not be replaced by _, and literal characters in the mask will be removed if the mask indicates they should be.

When setting Text, the value of the text is formatted using the mask, and the resulting EditText is used to update the window.

If there is no mask, Text is the text that appears in the edit control.

See Also

EditMask (see page 15821)

EditText (see page 15821)

GetTextLen (see page 15816)

IsMasked (see page 15822)

ReformatText (see page 15817)

90.1.3.2.37 TMaskEdit.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

90.1.3.3 TMaskEdit Events

The events of the TMaskEdit class are listed here.

Published Events

	Name	Description
	OnChange (see page 15866)	Occurs when the text for the edit control may have changed.
	OnClick (see page 15866)	Occurs when the user clicks the control.
	OnDblClick (see page 15868)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 15869)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 15871)	Occurs when the user drags an object over a control.
	OnEndDock (see page 15873)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 15874)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 15875)	Occurs when a control receives the input focus.
	OnExit (see page 15877)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 15879)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 15882)	Occurs when a key is pressed.
	OnKeyUp (see page 15883)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 15884)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 15885)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 15888)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 15888)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 15888)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 15890)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 15893)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 15893)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

90.1.3.3.1 TMaskEdit.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change (see page 21245)

Text (see page 5842)

90.1.3.3.2 TMaskEdit.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
```

```
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

90.1.3.3.3 TMaskEdit.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

90.1.3.3.4 TMaskEdit.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
```

```
    ListBox1->Items->Add("Alphabetical");
    ListBox1->Items->Add("Order");

  }
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
```

```
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

90.1.3.3.5 TMaskEdit.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/
```

```
// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
```

```
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

90.1.3.3.6 TMaskEdit.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

90.1.3.3.7 TMaskEdit.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
```

```
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

90.1.3.3.8 TMaskEdit.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}
```

```
void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
```

```
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

90.1.3.3.9 TMaskEdit.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit

event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
```

```
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

90.1.3.3.10 TMaskEdit.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
```

```
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
```

```
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
  {
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

90.1.3.3.11 TMaskEdit.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

90.1.3.3.12 TMaskEdit.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

90.1.3.3.13 TMaskEdit.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

90.1.3.3.14 TMaskEdit.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
```

```
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

90.1.3.3.15 TMaskEdit.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

90.1.3.3.16 TMaskEdit.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

90.1.3.3.17 TMaskEdit.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
```

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

90.1.3.3.18 TMaskEdit.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

90.1.3.3.19 TMaskEdit.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

90.1.3.3.20 TMaskEdit.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

90.2 Types

The following table lists types in this documentation.

Types

Name	Description
TMaskedState (see page 15895)	TMaskedState is a set of flags that describe the current state of a masked edit control.

90.2.1 Mask.TMaskedState Type

TMaskedState is a set of flags that describe the current state of a masked edit control.

Pascal

```
TMaskedState = set of (msMasked, msReEnter, msDBSetText);
```

C++

```
(msMasked, msReEnter, msDBSetText) TMaskedState;
```

File

Mask

Description

The following table indicates the meaning of the various TMaskedState flags:

Value	Meaning
msMasked	A mask has been assigned. MaskState includes msMasked when IsMasked is true.
msReEnter	An invalid character has been rejected and the next edit should be a valid replacement character.
msDBSetText	The text comes from a database field and should not be validated.

91 Masks Namespace

This is namespace Masks.

Topics

Name	Description
Classes (see page 15898)	The following table lists classes in this documentation.
Functions (see page 15902)	The following table lists functions in this documentation.

Classes

	Name	Description
	EMaskException (see page 15898)	EMaskException is the exception class for errors that occur when comparing a file name to a mask.
	TMask (see page 15898)	TMask allows the comparison of strings containing wildcards to a file mask.

Functions

	Name	Description
	MatchesMask (see page 15902)	Indicates whether a file name conforms to the format specified by a filter string.

91.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EMaskException (see page 15898)	EMaskException is the exception class for errors that occur when comparing a file name to a mask.
	TMask (see page 15898)	TMask allows the comparison of strings containing wildcards to a file mask.

91.1.1 EMaskException Class

EMaskException is the exception class for errors that occur when comparing a file name to a mask.

Class Hierarchy

System::TObject → SysUtils.Exception → Masks.EMaskException

Pascal

```
EMaskException = class(Exception);
```

C++

```
class EMaskException : public Exception;
```

File

Masks

Description

EMaskException is raised when the TMask (see page 15898) object or the global MatchesMask (see page 15902) function encounters an improperly formed mask string.

See Also

TMask (see page 15898)

MatchesMask (see page 15902)

91.1.2 TMask Class

TMask allows the comparison of strings containing wildcards to a file mask.

Class Hierarchy

Masks.TMask

Pascal

```
TMask = class;
```

C++

```
class TMask;
```

File

Masks

Description

Use TMask to determine if a string matches the format described by a mask. The mask contains literal characters, wildcards that can match any character, and sets that match a single character that is included in the set.

Note: Do not confuse TMask with the EditMask of a field or masked edit object. While both are used for comparing strings to a symbolic description of valid values, the special mask symbols and matching rules are completely different.

See Also

TCustomMaskEdit (see page 15798)

TField (see page 7115)

91.1.2.1 TMask Members

The following tables list the members exposed by TMask.

Public Methods

	Name	Description
	Create (see page 15899)	Creates an instance of TMask (see page 15898) for a given mask.
	Destroy (see page 15900)	Destroys an instance of TMask (see page 15898).
	Matches (see page 15900)	Indicates whether a specified string matches the mask.

91.1.2.2 TMask.Create Constructor

Creates an instance of TMask (see page 15898) for a given mask.

Pascal

```
constructor Create(const MaskValue: string);
```

C++

```
__fastcall TMask(const AnsiString MaskValue);
```

Description

Call Create to create a mask object for comparing strings to the mask given by the MaskValue parameter. A valid mask consists of literal characters, sets, and wildcards.

Each literal character must match a single character in the string. The comparison to literal characters is case-insensitive.

Each set begins with an opening bracket ([) and ends with a closing bracket (]). Between the brackets are the elements of the set. Each element is a literal character or a range. Ranges are specified by an initial value, a dash (-), and a final value. Do not use spaces or commas to separate the elements of the set. A set must match a single character in the string. The character matches the set if it is the same as one of the literal characters in the set, or if it is in one of the ranges in the set. A character is in a range if it matches the initial value, the final value, or falls between the two values. All comparisons are case-insensitive. If the first character after the opening bracket of a set is an exclamation point (!), then the set matches any character that is not in the set.

Wildcards are asterisks (*) or question marks (?). An asterisk matches any number of characters. A question mark matches a

single arbitrary character.

See Also

Matches (see page 15900)

91.1.2.3 TMask.Destroy Destructor

Destroys an instance of TMask (see page 15898).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMask();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TMask (see page 15898) reference is not nil, and only then calls Destroy.

Destroy frees the memory used to store a parsed image of the mask.

See Also

Free (see page 22300)

91.1.2.4 TMask Methods

The methods of the TMask class are listed here.

Public Methods

	Name	Description
	Matches (see page 15900)	Indicates whether a specified string matches the mask.

91.1.2.4.1 TMask.Matches Method

Indicates whether a specified string matches the mask.

Pascal

```
function Matches(const Filename: string): Boolean;
```

C++

```
__fastcall Boolean Matches(const AnsiString Filename);
```

Description

Call Matches to compare the Filename parameter to the mask that describes valid values. Matches returns true if the string matches the mask. Matches returns false if the string does not match the mask. Matches raises an exception if the mask is syntactically invalid.

The mask is set when the TMask (see page 15898) object is created.

Note: Filename parameter does not need to be a file name. TMask (see page 15898) can be used to check strings against any syntactically correct mask.

See Also

Create (see page 15899)

91.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	MatchesMask (see page 15902)	Indicates whether a file name conforms to the format specified by a filter string.

91.2.1 Masks.MatchesMask Function

Indicates whether a file name conforms to the format specified by a filter string.

Pascal

```
function MatchesMask(const Filename: string; const Mask: string): Boolean;
```

C++

```
Boolean MatchesMask(const AnsiString Filename, const AnsiString Mask);
```

File

Masks

Description

Call MatchesMask to check the Filename parameter using the Mask (see page 15797) parameter to describe valid values. A valid mask consists of literal characters, sets, and wildcards.

Each literal character must match a single character in the string. The comparison to literal characters is case-insensitive.

Each set begins with an opening bracket ([) and ends with a closing bracket (]). Between the brackets are the elements of the set. Each element is a literal character or a range. Ranges are specified by an initial value, a dash (-), and a final value. Do not use spaces or commas to separate the elements of the set. A set must match a single character in the string. The character matches the set if it is the same as one of the literal characters in the set, or if it is in one of the ranges in the set. A character is in a range if it matches the initial value, the final value, or falls between the two values. All comparisons are case-insensitive. If the first character after the opening bracket of a set is an exclamation point (!), then the set matches any character that is not in the set.

Wildcards are asterisks (*) or question marks (?). An asterisk matches any number of characters. A question mark matches a single arbitrary character.

MatchesMask returns true if the string matches the mask. MatchesMask returns false if the string does not match the mask. MatchesMask raises an exception if the mask is syntactically invalid.

Note: The Filename parameter does not need to be a file name. MatchesMask can be used to check strings against any syntactically correct mask.

See Also

TMask (see page 15898)

92 MaskUtils Namespace

This is namespace MaskUtils.

Topics

Name	Description
Functions (see page 15905)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 15911)	The following table lists structs, records, enums in this documentation.
Types (see page 15912)	The following table lists types in this documentation.
Variables (see page 15915)	The following table lists variables in this documentation.
Constants (see page 15917)	The following table lists constants in this documentation.

Constants

Name	Description
mDirLiteral (see page 15917)	This is constant MaskUtils.mDirLiteral.
mDirLowerCase (see page 15917)	This is constant MaskUtils.mDirLowerCase.
mDirReverse (see page 15918)	This is constant MaskUtils.mDirReverse.
mDirUpperCase (see page 15918)	This is constant MaskUtils.mDirUpperCase.
mMskAlpha (see page 15918)	This is constant MaskUtils.mMskAlpha.
mMskAlphaNum (see page 15919)	This is constant MaskUtils.mMskAlphaNum.
mMskAlphaNumOpt (see page 15919)	This is constant MaskUtils.mMskAlphaNumOpt.
mMskAlphaOpt (see page 15919)	This is constant MaskUtils.mMskAlphaOpt.
mMskAscii (see page 15920)	This is constant MaskUtils.mMskAscii.
mMskAsciiOpt (see page 15920)	This is constant MaskUtils.mMskAsciiOpt.
mMskDateSeparator (see page 15920)	This is constant MaskUtils.mMskDateSeparator.
mMskNumeric (see page 15921)	This is constant MaskUtils.mMskNumeric.
mMskNumericOpt (see page 15921)	This is constant MaskUtils.mMskNumericOpt.
mMskNumSymOpt (see page 15921)	This is constant MaskUtils.mMskNumSymOpt.
mMskTimeSeparator (see page 15922)	This is constant MaskUtils.mMskTimeSeparator.

Functions

	Name	Description
	FormatMaskText (see page 15905)	Returns a string formatted using an edit mask.
	IsLiteralChar (see page 15907)	This is function MaskUtils.IsLiteralChar.
	MaskDoFormatText (see page 15907)	This is function MaskUtils.MaskDoFormatText.
	MaskGetCharType (see page 15907)	This is function MaskUtils.MaskGetCharType.
	MaskGetCurrentDirectives (see page 15908)	This is function MaskUtils.MaskGetCurrentDirectives.

	MaskGetFldSeparator (see page 15908)	This is function MaskUtils.MaskGetFldSeparator.
	MaskGetMaskBlank (see page 15908)	This is function MaskUtils.MaskGetMaskBlank.
	MaskGetMaskSave (see page 15909)	This is function MaskUtils.MaskGetMaskSave.
	MaskIntlLiteralToChar (see page 15909)	This is function MaskUtils.MaskIntlLiteralToChar.
	MaskOffsetToOffset (see page 15909)	This is function MaskUtils.MaskOffsetToOffset.
	MaskOffsetToWideOffset (see page 15910)	This is function MaskUtils.MaskOffsetToWideOffset.
	OffsetToMaskOffset (see page 15910)	This is function MaskUtils.OffsetToMaskOffset.
	PadInputLiterals (see page 15910)	This is function MaskUtils.PadInputLiterals.

Structs, Records, Enums

	Name	Description
	TMaskCharType (see page 15911)	This is record MaskUtils.TMaskCharType.

Types

Name	Description
TEditMask (see page 15912)	Represents the mask that validates and formats user input.
TMaskDirectives (see page 15913)	This is type MaskUtils.TMaskDirectives.
TMaskedText (see page 15914)	TMaskedText is the type for the text that is typed into a masked edit control.

Variables

Name	Description
DefaultBlank (see page 15915)	This is variable MaskUtils.DefaultBlank.
MaskFieldSeparator (see page 15915)	This is variable MaskUtils.MaskFieldSeparator.
MaskNoSave (see page 15915)	This is variable MaskUtils.MaskNoSave.

92.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FormatMaskText (see page 15905)	Returns a string formatted using an edit mask.
	IsLiteralChar (see page 15907)	This is function MaskUtils.IsLiteralChar.
	MaskDoFormatText (see page 15907)	This is function MaskUtils.MaskDoFormatText.
	MaskGetCharType (see page 15907)	This is function MaskUtils.MaskGetCharType.
	MaskGetCurrentDirectives (see page 15908)	This is function MaskUtils.MaskGetCurrentDirectives.
	MaskGetFldSeparator (see page 15908)	This is function MaskUtils.MaskGetFldSeparator.
	MaskGetMaskBlank (see page 15908)	This is function MaskUtils.MaskGetMaskBlank.
	MaskGetMaskSave (see page 15909)	This is function MaskUtils.MaskGetMaskSave.
	MaskIntlLiteralToChar (see page 15909)	This is function MaskUtils.MaskIntlLiteralToChar.
	MaskOffsetToOffset (see page 15909)	This is function MaskUtils.MaskOffsetToOffset.
	MaskOffsetToWideOffset (see page 15910)	This is function MaskUtils.MaskOffsetToWideOffset.
	OffsetToMaskOffset (see page 15910)	This is function MaskUtils.OffsetToMaskOffset.
	PadInputLiterals (see page 15910)	This is function MaskUtils.PadInputLiterals.

92.1.1 MaskUtils.FormatMaskText Function

Returns a string formatted using an edit mask.

Pascal

```
function FormatMaskText(const EditMask: string; const Value: string): string;
```

C++

```
AnsiString FormatMaskText(const AnsiString EditMask, const AnsiString Value);
```

File

MaskUtils

Description

Call FormatMaskText to apply the mask specified by the EditMask parameter to the text string specified by the Value parameter. The edit mask string consists of three fields with semicolons separating them. The first part of the mask is the mask itself. The second part is the character that determines whether the literal characters of the mask are matched to characters in the Value parameter or are inserted into the Value string. The third part of the mask is the character used to represent missing characters in the mask.

These are the special characters used in the first field of the mask:

Character	Meaning in mask
!	If a ! character appears in the mask, optional characters are represented in the returned string as leading blanks. If a ! character is not present, optional characters are represented in the returned string as trailing blanks.
>	If a > character appears in the mask, all characters that follow are in uppercase until the end of the mask or until a < character is encountered.

<	If a < character appears in the mask, all characters that follow are in lowercase until the end of the mask or until a > character is encountered.
<>	If these two characters appear together in a mask, no case checking is done and the data is formatted with the case present in the Value parameter.
\	The character that follows a \ character is a literal character. Use this character to use any of the mask special characters as a literal.
L	The L character requires an alphabetic character only in this position. For the US, this is A-Z, a-z.
l	The l character permits only an alphabetic character in this position, but doesn't require it.
A	The A character requires an alphanumeric character only in this position. For the US, this is A-Z, a-z, 0-9.
a	The a character permits an alphanumeric character in this position, but doesn't require it.
C	The C character requires an arbitrary character in this position.
c	The c character permits an arbitrary character in this position, but doesn't require it.
0	The 0 character requires a numeric character only in this position.
9	The 9 character permits a numeric character in this position, but doesn't require it.
#	The # character permits a numeric character or a plus or minus sign in this position, but doesn't require it.
:	The : character is used to separate hours, minutes, and seconds in times. If the character that separates hours, minutes, and seconds is different in the regional settings of the Control Panel, that character is substituted in the returned string.
/	The / character is used to separate months, days, and years in dates. If the character that separates months, days, and years is different in the regional settings of the Control Panel, that character is substituted in the returned string.
;	The ; character is used to separate the three fields of the mask.
_	The _ character automatically inserts spaces into the returned string.

Any character that does not appear in the preceding table can appear in the first part of the mask as a literal character. Literal characters are inserted automatically if the second field of the mask is 0, or matched to characters in the Value parameter if the second field is any other value. The special mask characters can also appear as literal characters if preceded by a backslash character (\).

The second field of the mask is a single character that indicates whether literal characters from the mask are included in the Value parameter. For example, the mask for a telephone number with area code could be the following string:

(000)_000-0000;0;*

The 0 in the second field indicates that the Value parameter should consist of the 10 digits of the phone number, rather than the 14 characters that make up the final formatted string.

A 0 in the second field indicates that literals are inserted into the Value string, any other character indicates that they should be included. The character to indicate whether literals should be included can be changed by changing the MaskNoSave (see page 15915) constant that is declared in the MaskUtils (see page 15903) unit.

The third field of the mask is the character that appears in the returned string for blanks (characters that do not appear in Value). By default, this is the same as the character that stands for literal spaces. The two characters appear the same in the returned string.

Note: When working with multibyte character sets, each special mask character represents a single byte. To specify double-byte characters using the L, l, A, a, C, or c specifiers, the mask characters must be doubled as well. For example, LL would represent two single-byte alphabetic characters or a one double-byte character. Only single-byte literal characters are supported.

See Also

TCustomMaskEdit

TField

92.1.2 MaskUtils.IsLiteralChar Function

Pascal

```
function IsLiteralChar(const EditMask: string; Offset: Integer): Boolean;
```

C++

```
Boolean IsLiteralChar(const AnsiString EditMask, int Offset);
```

File

MaskUtils

Description

This is function MaskUtils.IsLiteralChar.

92.1.3 MaskUtils.MaskDoFormatText Function

Pascal

```
function MaskDoFormatText(const EditMask: string; const Value: string; Blank: Char): string;
```

C++

```
AnsiString MaskDoFormatText(const AnsiString EditMask, const AnsiString Value, Char Blank);
```

File

MaskUtils

Description

This is function MaskUtils.MaskDoFormatText.

92.1.4 MaskUtils.MaskGetCharType Function

Pascal

```
function MaskGetCharType(const EditMask: string; MaskOffset: Integer): TMaskCharType;
```

C++

```
TMaskCharType MaskGetCharType(const AnsiString EditMask, int MaskOffset);
```

File

MaskUtils

Description

This is function MaskUtils.MaskGetCharType.

92.1.5 MaskUtils.MaskGetCurrentDirectives Function

Pascal

```
function MaskGetCurrentDirectives(const EditMask: string; MaskOffset: Integer):
TMaskDirectives;
```

C++

```
TMaskDirectives MaskGetCurrentDirectives(const AnsiString EditMask, int MaskOffset);
```

File

MaskUtils

Description

This is function MaskUtils.MaskGetCurrentDirectives.

92.1.6 MaskUtils.MaskGetFldSeparator Function

Pascal

```
function MaskGetFldSeparator(const EditMask: string): Integer;
```

C++

```
int MaskGetFldSeparator(const AnsiString EditMask);
```

File

MaskUtils

Description

This is function MaskUtils.MaskGetFldSeparator.

92.1.7 MaskUtils.MaskGetMaskBlank Function

Pascal

```
function MaskGetMaskBlank(const EditMask: string): Char;
```

C++

```
Char MaskGetMaskBlank(const AnsiString EditMask);
```

File

MaskUtils

Description

This is function MaskUtils.MaskGetMaskBlank.

92.1.8 MaskUtils.MaskGetMaskSave Function

Pascal

```
function MaskGetMaskSave(const EditMask: string): Boolean;
```

C++

```
Boolean MaskGetMaskSave(const AnsiString EditMask);
```

File

MaskUtils

Description

This is function MaskUtils.MaskGetMaskSave.

92.1.9 MaskUtils.MaskIntlLiteralToChar Function

Pascal

```
function MaskIntlLiteralToChar(IChar: Char): Char;
```

C++

```
Char MaskIntlLiteralToChar(Char IChar);
```

File

MaskUtils

Description

This is function MaskUtils.MaskIntlLiteralToChar.

92.1.10 MaskUtils.MaskOffsetToOffset Function

Pascal

```
function MaskOffsetToOffset(const EditMask: String; MaskOffset: Integer): Integer;
```

C++

```
int MaskOffsetToOffset(const AnsiString EditMask, int MaskOffset);
```

File

MaskUtils

Description

This is function MaskUtils.MaskOffsetToOffset.

92.1.11 MaskUtils.MaskOffsetToWideOffset Function

Pascal

```
function MaskOffsetToWideOffset(const EditMask: String; MaskOffset: Integer): Integer;
```

C++

```
int MaskOffsetToWideOffset(const AnsiString EditMask, int MaskOffset);
```

File

MaskUtils

Description

This is function MaskUtils.MaskOffsetToWideOffset.

92.1.12 MaskUtils.OffsetToMaskOffset Function

Pascal

```
function OffsetToMaskOffset(const EditMask: string; Offset: Integer): Integer;
```

C++

```
int OffsetToMaskOffset(const AnsiString EditMask, int Offset);
```

File

MaskUtils

Description

This is function MaskUtils.OffsetToMaskOffset.

92.1.13 MaskUtils.PadInputLiterals Function

Pascal

```
function PadInputLiterals(const EditMask: String; const Value: string; Blank: Char): string;
```

C++

```
AnsiString PadInputLiterals(const AnsiString EditMask, const AnsiString Value, Char Blank);
```

File

MaskUtils

Description

This is function MaskUtils.PadInputLiterals.

92.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TMaskCharType (see page 15911)	This is record MaskUtils.TMaskCharType.

92.2.1 MaskUtils.TMaskCharType Enumeration

Pascal

```
TMaskCharType = (
  mcNone,
  mcLiteral,
  mcIntlLiteral,
  mcDirective,
  mcMask,
  mcMaskOpt,
  mcFieldSeparator,
  mcField
);
```

C++

```
enum TMaskCharType {
  mcNone,
  mcLiteral,
  mcIntlLiteral,
  mcDirective,
  mcMask,
  mcMaskOpt,
  mcFieldSeparator,
  mcField
};
```

File

MaskUtils

Description

This is record MaskUtils.TMaskCharType.

92.3 Types

The following table lists types in this documentation.

Types

Name	Description
TEditMask (see page 15912)	Represents the mask that validates and formats user input.
TMaskDirectives (see page 15913)	This is type MaskUtils.TMaskDirectives.
TMaskedText (see page 15914)	TMaskedText is the type for the text that is typed into a masked edit control.

92.3.1 MaskUtils.TEditMask Type

Represents the mask that validates and formats user input.

Pascal

```
TEditMask = type string;
```

C++

```
type string TEditMask;
```

File

MaskUtils

Description

TEditMask is a string that consists of three fields with semicolons separating them. The first part of the mask is the mask itself. The second part is the character that determines whether the literal characters of a mask are saved as part of the data. The third part of the mask is the character used to represent unentered characters in the mask.

These are the special characters used in the first field of the mask:

Character	Meaning in mask
!	If a ! character appears in the mask, optional characters are represented in the text as leading blanks. If a ! character is not present, optional characters are represented in the text as trailing blanks.
>	If a > character appears in the mask, all characters that follow are in uppercase until the end of the mask or until a < character is encountered.
<	If a < character appears in the mask, all characters that follow are in lowercase until the end of the mask or until a > character is encountered.
<>	If these two characters appear together in a mask, no case checking is done and the data is formatted with the case the user uses to enter the data.
\	The character that follows a \ character is a literal character. Use this character to use any of the mask special characters as a literal in the data.
L	The L character requires an alphabetic character only in this position. For the US, this is A-Z, a-z.
l	The l character permits only an alphabetic character in this position, but doesn't require it.
A	The A character requires an alphanumeric character only in this position. For the US, this is A-Z, a-z, 0-9.
a	The a character permits an alphanumeric character in this position, but doesn't require it.

C	The C character requires an arbitrary character in this position.
c	The c character permits an arbitrary character in this position, but doesn't require it.
0	The 0 character requires a numeric character only in this position.
9	The 9 character permits a numeric character in this position, but doesn't require it.
#	The # character permits a numeric character or a plus or minus sign in this position, but doesn't require it.
:	The : character is used to separate hours, minutes, and seconds in times. If the character that separates hours, minutes, and seconds is different in the regional settings of the Control Panel utility on your computer system, that character is used instead.
/	The / character is used to separate months, days, and years in dates. If the character that separates months, days, and years is different in the regional settings of the Control Panel utility on your computer system, that character is used instead.
;	The ; character is used to separate the three fields of the mask.
_	The _ character automatically inserts spaces into the text. When the user enters characters in the field, the cursor skips the _ character.

Any character that does not appear in the preceding table can appear in the first part of the mask as a literal character. Literal characters must be matched exactly in the edit control. They are inserted automatically, and the cursor skips over them during editing. The special mask characters can also appear as literal characters if preceded by a backslash character (\).

The second field of the mask is a single character that indicates whether literal characters from the mask should be included as part of the text for the edit control. For example, the mask for a telephone number with area code could be the following string:

(000)_000-0000;0;*

The 0 in the second field indicates that the Text property for the edit control would consist of the 10 digits that were entered, rather than the 14 characters that make up the telephone number as it appears in the edit control.

A 0 in the second field indicates that literals should not be included, any other character indicates that they should be included. The character that indicates whether literals should be included can be changed in the Edit Mask (see page 15797) property editor, or programmatically by changing the MaskNoSave (see page 15915) typed constant.

The third field of the mask is the character that appears in the edit control for blanks (characters that have not been entered). By default, this is the same as the character that stands for literal spaces. The two characters appear the same in an edit window. However, when a user edits the text in a masked edit control, the cursor selects each blank character in turn, and skips over the space character.

Note: When working with multibyte character sets, each special mask character represents a single byte. To specify multi-byte characters using the L, l, A, a, C, or c specifiers, the mask characters must be duplicated as well. For example, LL would represent two single-byte alphabetic characters or a one double-byte character. Only single-byte literal characters are supported.

92.3.2 MaskUtils.TMaskDirectives Type

Pascal

```
TMaskDirectives = set of (mdReverseDir, mdUpperCase, mdLowerCase, mdLiteralChar);
```

C++

```
(mdReverseDir, mdUpperCase, mdLowerCase, mdLiteralChar) TMaskDirectives;
```

File

MaskUtils

Description

This is type MaskUtils.TMaskDirectives.

92.3.3 MaskUtils.TMaskedText Type

TMaskedText is the type for the text that is typed into a masked edit control.

Pascal

```
TMaskedText = type string;
```

C++

```
type string TMaskedText;
```

File

MaskUtils

Description

TMaskedText is the value of a masked edit control's text before the mask is applied.

92.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
DefaultBlank (see page 15915)	This is variable MaskUtils.DefaultBlank.
MaskFieldSeparator (see page 15915)	This is variable MaskUtils.MaskFieldSeparator.
MaskNoSave (see page 15915)	This is variable MaskUtils.MaskNoSave.

92.4.1 MaskUtils.DefaultBlank Variable

Pascal

```
DefaultBlank: Char = '_';
```

C++

```
Char DefaultBlank = '_';
```

File

MaskUtils

Description

This is variable MaskUtils.DefaultBlank.

92.4.2 MaskUtils.MaskFieldSeparator Variable

Pascal

```
MaskFieldSeparator: Char = ';';
```

C++

```
Char MaskFieldSeparator = ';';
```

File

MaskUtils

Description

This is variable MaskUtils.MaskFieldSeparator.

92.4.3 MaskUtils.MaskNoSave Variable

Pascal

```
MaskNoSave: Char = '0';
```

C++

```
Char MaskNoSave = '0';
```

File

MaskUtils

Description

This is variable MaskUtils.MaskNoSave.

92.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
mDirLiteral (see page 15917)	This is constant MaskUtils.mDirLiteral.
mDirLowerCase (see page 15917)	This is constant MaskUtils.mDirLowerCase.
mDirReverse (see page 15918)	This is constant MaskUtils.mDirReverse.
mDirUpperCase (see page 15918)	This is constant MaskUtils.mDirUpperCase.
mMskAlpha (see page 15918)	This is constant MaskUtils.mMskAlpha.
mMskAlphaNum (see page 15919)	This is constant MaskUtils.mMskAlphaNum.
mMskAlphaNumOpt (see page 15919)	This is constant MaskUtils.mMskAlphaNumOpt.
mMskAlphaOpt (see page 15919)	This is constant MaskUtils.mMskAlphaOpt.
mMskAscii (see page 15920)	This is constant MaskUtils.mMskAscii.
mMskAsciiOpt (see page 15920)	This is constant MaskUtils.mMskAsciiOpt.
mMskDateSeparator (see page 15920)	This is constant MaskUtils.mMskDateSeparator.
mMskNumeric (see page 15921)	This is constant MaskUtils.mMskNumeric.
mMskNumericOpt (see page 15921)	This is constant MaskUtils.mMskNumericOpt.
mMskNumSymOpt (see page 15921)	This is constant MaskUtils.mMskNumSymOpt.
mMskTimeSeparator (see page 15922)	This is constant MaskUtils.mMskTimeSeparator.

92.5.1 MaskUtils.mDirLiteral Constant

Pascal

```
mDirLiteral = '\';
```

C++

```
mDirLiteral = '\';
```

File

MaskUtils

Description

This is constant MaskUtils.mDirLiteral.

92.5.2 MaskUtils.mDirLowerCase Constant

Pascal

```
mDirLowerCase = '<';
```

C++

```
mDirLowerCase = '<';
```

File

MaskUtils

Description

This is constant MaskUtils.mDirLowerCase.

92.5.3 MaskUtils.mDirReverse Constant

Pascal

```
mDirReverse = '!';
```

C++

```
mDirReverse = '!';
```

File

MaskUtils

Description

This is constant MaskUtils.mDirReverse.

92.5.4 MaskUtils.mDirUpperCase Constant

Pascal

```
mDirUpperCase = '>';
```

C++

```
mDirUpperCase = '>';
```

File

MaskUtils

Description

This is constant MaskUtils.mDirUpperCase.

92.5.5 MaskUtils.mMskAlpha Constant

Pascal

```
mMskAlpha = 'L';
```

C++

```
mMskAlpha = 'L';
```

File

MaskUtils

Description

This is constant MaskUtils.mMskAlpha.

92.5.6 MaskUtils.mMskAlphaNum Constant

Pascal

```
mMskAlphaNum = 'A';
```

C++

```
mMskAlphaNum = 'A';
```

File

MaskUtils

Description

This is constant MaskUtils.mMskAlphaNum.

92.5.7 MaskUtils.mMskAlphaNumOpt Constant

Pascal

```
mMskAlphaNumOpt = 'a';
```

C++

```
mMskAlphaNumOpt = 'a';
```

File

MaskUtils

Description

This is constant MaskUtils.mMskAlphaNumOpt.

92.5.8 MaskUtils.mMskAlphaOpt Constant

Pascal

```
mMskAlphaOpt = 'l';
```

C++

```
mMskAlphaOpt = 'l';
```

File

MaskUtils

Description

This is constant MaskUtils.mMskAlphaOpt.

92.5.9 MaskUtils.mMskAscii Constant

Pascal

```
mMskAscii = 'C';
```

C++

```
mMskAscii = 'C';
```

File

MaskUtils

Description

This is constant MaskUtils.mMskAscii.

92.5.10 MaskUtils.mMskAsciiOpt Constant

Pascal

```
mMskAsciiOpt = 'c';
```

C++

```
mMskAsciiOpt = 'c';
```

File

MaskUtils

Description

This is constant MaskUtils.mMskAsciiOpt.

92.5.11 MaskUtils.mMskDateSeparator Constant

Pascal

```
mMskDateSeparator = '/';
```

C++

```
mMskDateSeparator = '/';
```

File

MaskUtils

Description

This is constant MaskUtils.mMskDateSeparator.

92.5.12 MaskUtils.mMskNumeric Constant

Pascal

```
mMskNumeric = '0';
```

C++

```
mMskNumeric = '0';
```

File

MaskUtils

Description

This is constant MaskUtils.mMskNumeric.

92.5.13 MaskUtils.mMskNumericOpt Constant

Pascal

```
mMskNumericOpt = '9';
```

C++

```
mMskNumericOpt = '9';
```

File

MaskUtils

Description

This is constant MaskUtils.mMskNumericOpt.

92.5.14 MaskUtils.mMskNumSymOpt Constant

Pascal

```
mMskNumSymOpt = '#';
```

C++

```
mMskNumSymOpt = '#';
```

File

MaskUtils

Description

This is constant MaskUtils.mMskNumSymOpt.

92.5.15 MaskUtils.mMskTimeSeparator Constant

Pascal

```
mMskTimeSeparator = ':';
```

C++

```
mMskTimeSeparator = ':';
```

File

MaskUtils

Description

This is constant MaskUtils.mMskTimeSeparator.

93 Math Namespace

This is namespace Math.

Topics

Name	Description
Classes (see page 15927)	The following table lists classes in this documentation.
Functions (see page 15929)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 15981)	The following table lists structs, records, enums in this documentation.
Types (see page 15985)	The following table lists types in this documentation.
Constants (see page 15988)	The following table lists constants in this documentation.

Classes

	Name	Description
	EInvalidArgument (see page 15927)	EInvalidArgument is the exception class for out-of-range parameters in specialized math and accounting functions.

Constants

Name	Description
Infinity (see page 15988)	Infinity represents positive infinity.
MaxComp (see page 15989)	MaxComp defines the maximum absolute value for a Comp variable.
MaxDouble (see page 15989)	MaxDouble defines the maximum absolute value for a Double variable.
MaxExtended (see page 15989)	MaxExtended defines the maximum absolute value for an Extended variable.
MaxSingle (see page 15990)	MaxSingle defines the maximum absolute value for a Single variable.
MinComp (see page 15990)	MinComp defines the smallest absolute value for a Comp variable.
MinDouble (see page 15991)	MinDouble defines the smallest absolute value for a Double variable.
MinExtended (see page 15991)	MinExtended defines the smallest absolute value for an Extended variable.
MinSingle (see page 15991)	MinSingle defines the smallest absolute value for a Single variable.
NaN (see page 15992)	NaN represents a value that cannot be represented as a valid number.
NegativeValue (see page 15992)	NegativeValue defines a negative number result from using Sign (see page 15973).
NegInfinity (see page 15993)	NegInfinity represents negative infinity.
PositiveValue (see page 15993)	PositiveValue defines a positive number result from using Sign (see page 15973).
ZeroValue (see page 15994)	ZeroValue defines a zero value number result from using Sign (see page 15973).

Functions

	Name	Description
	ArcCos (see page 15931)	Calculates the inverse cosine of a given number.
	ArcCos (see page 15931)	Calculates the inverse cosine of a given number.
	ArcCos (see page 15931)	Calculates the inverse cosine of a given number.
	ArcCosh (see page 15932)	Calculates the inverse hyperbolic cosine of a given number.
	ArcCot (see page 15932)	Calculates the inverse cotangent of a given number.
	ArcCotH (see page 15933)	Calculates the inverse hyperbolic cotangent of a given number.
	ArcCsc (see page 15933)	Calculates the inverse cosecant of a given number.
	ArcCscH (see page 15934)	Calculates the inverse hyperbolic cosecant of a given number.
	ArcSec (see page 15934)	Calculates the inverse secant of a given number.
	ArcSecH (see page 15934)	Calculates the inverse hyperbolic secant of a given number.
	ArcSin (see page 15935)	Calculates the inverse sine of a given number.
	ArcSin (see page 15935)	Calculates the inverse sine of a given number.
	ArcSin (see page 15935)	Calculates the inverse sine of a given number.
	ArcSinh (see page 15935)	Calculates the inverse hyperbolic sine of a given number.
	ArcTan2 (see page 15936)	Calculates the arctangent angle and quadrant of a given number.
	ArcTanh (see page 15936)	Calculates the inverse hyperbolic tangent of a given number.
	Ceil (see page 15937)	Rounds variables up toward positive infinity.
	ClearExceptions (see page 15937)	Clears exception bits in FPU status word.
	CompareValue (see page 15938)	Returns the relationship between two numeric values.
	CompareValue (see page 15938)	Returns the relationship between two numeric values.
	CompareValue (see page 15938)	Returns the relationship between two numeric values.
	CompareValue (see page 15938)	Returns the relationship between two numeric values.
	CompareValue (see page 15938)	Returns the relationship between two numeric values.
	Cosecant (see page 15938)	Returns the cosecant of an angle.
	Cosh (see page 15939)	Calculates the hyperbolic cosine of an angle.
	Cot (see page 15939)	Calculates the cotangent of an angle.
	Cotan (see page 15940)	Calculates the cotangent of an angle.
	CotH (see page 15940)	Calculates the hyperbolic cotangent of an angle.
	Csc (see page 15941)	Returns the cosecant of an angle.
	CscH (see page 15941)	Returns the hyperbolic cosecant of an angle.
	CycleToDeg (see page 15942)	Converts an angle measurement from cycles to degrees.
	CycleToGrad (see page 15942)	Converts an angle measurement from cycles to grads.
	CycleToRad (see page 15942)	Converts an angle measurement from cycles to radians.
	DegToCycle (see page 15943)	Returns the value of a degree measurement expressed in cycles.
	DegToGrad (see page 15943)	Returns the value of a degree measurement expressed in grads.
	DegToRad (see page 15944)	Returns the value of a degree measurement expressed in radians.
	DivMod (see page 15944)	Returns the result of an integer division, including the remainder.
	DoubleDecliningBalance (see page 15944)	Calculates the depreciation of an asset using the double-declining balance method.
	EnsureRange (see page 15945)	Returns the closest value to a specified value within a specified range.
	EnsureRange (see page 15945)	Returns the closest value to a specified value within a specified range.
	EnsureRange (see page 15945)	Returns the closest value to a specified value within a specified range.
	Floor (see page 15945)	Rounds variables toward negative infinity.
	Frexp (see page 15946)	Separates the Mantissa and Exponent of X
	FutureValue (see page 15946)	Calculates the future value of an investment.
	GetExceptionMask (see page 15947)	Returns the exception mask from the FPU control word.
	GetPrecisionMode (see page 15947)	Returns the FPU precision control mode.
	GetRoundMode (see page 15948)	Returns the FPU rounding mode.
	GradToCycle (see page 15949)	Converts grad measurements to cycles.
	GradToDeg (see page 15949)	Converts grad measurements to degrees.
	GradToRad (see page 15949)	Converts grad measurements to radians.
	Hypot (see page 15950)	Calculates the length of the hypotenuse.

	IfThen (see page 15950)	Conditionally returns one of two specified values.
	IfThen (see page 15950)	Conditionally returns one of two specified values.
	IfThen (see page 15950)	Conditionally returns one of two specified values.
	InRange (see page 15950)	Indicates whether a value falls within a specified range.
	InRange (see page 15950)	Indicates whether a value falls within a specified range.
	InRange (see page 15950)	Indicates whether a value falls within a specified range.
	InterestPayment (see page 15951)	Calculates the interest portion of a loan payment.
	InterestRate (see page 15952)	Returns the interest rate required to increase PresentValue (see page 15965) to FutureValue (see page 15946).
	InternalRateOfReturn (see page 15952)	Calculates the internal rate of return for an investment.
	IntPower (see page 15953)	Calculates the integral power of a base value.
	IsInfinite (see page 15953)	Indicates when a variable or expression represents an infinite value.
	IsNan (see page 15954)	Indicates when a variable or expression does not evaluate to a numeric value.
	IsNan (see page 15954)	Indicates when a variable or expression does not evaluate to a numeric value.
	IsNan (see page 15954)	Indicates when a variable or expression does not evaluate to a numeric value.
	IsZero (see page 15954)	Indicates when a floating-point variable or expression evaluates to zero, or very close to zero.
	IsZero (see page 15954)	Indicates when a floating-point variable or expression evaluates to zero, or very close to zero.
	IsZero (see page 15954)	Indicates when a floating-point variable or expression evaluates to zero, or very close to zero.
	Ldexp (see page 15955)	Calculates X times (2 to the power of P).
	LnXP1 (see page 15955)	Returns the natural log of (X+1)
	Log10 (see page 15955)	Calculates log base 10.
	Log2 (see page 15956)	Calculates log base 2.
	LogN (see page 15956)	Calculates the log of X for a specified base.
	Max (see page 15957)	Returns the greater of two numeric values (Delphi only).
	Max (see page 15957)	Returns the greater of two numeric values (Delphi only).
	Max (see page 15957)	Returns the greater of two numeric values (Delphi only).
	Max (see page 15957)	Returns the greater of two numeric values (Delphi only).
	Max (see page 15957)	Returns the greater of two numeric values (Delphi only).
	MaxIntValue (see page 15957)	Returns the largest signed value in an integer array.
	MaxValue (see page 15958)	Returns the largest signed value in an array.
	Mean (see page 15958)	Returns the average of all values in an array.
	MeanAndStdDev (see page 15959)	Calculates the mean and standard deviation of array elements.
	Min (see page 15959)	Returns the lesser of two numeric values (Delphi only).
	Min (see page 15959)	Returns the lesser of two numeric values (Delphi only).
	Min (see page 15959)	Returns the lesser of two numeric values (Delphi only).
	Min (see page 15959)	Returns the lesser of two numeric values (Delphi only).
	Min (see page 15959)	Returns the lesser of two numeric values (Delphi only).
	MinIntValue (see page 15960)	Returns the smallest signed value in an integer array.
	MinValue (see page 15960)	Returns smallest signed value in an array.
	MomentSkewKurtosis (see page 15961)	Calculates the mean, variance, skew, and kurtosis.
	NetPresentValue (see page 15961)	Calculates the current value from an array of estimated cash flow values.
	Norm (see page 15962)	Returns the Euclidean 'L-2' norm.
	NumberOfPeriods (see page 15962)	Returns the number of payment periods for an investment.
	Payment (see page 15963)	Calculates a fully amortized payment.
	PeriodPayment (see page 15963)	Returns the principal amount from a full payment.
	Poly (see page 15963)	Evaluates a uniform polynomial of one variable at the value X.
	PopnStdDev (see page 15964)	Calculates the population standard deviation.
	PopnVariance (see page 15964)	Calculates the population variance.
	Power (see page 15965)	Raises Base to any power.
	Power (see page 15965)	Raises Base to any power.
	Power (see page 15965)	Raises Base to any power.
	PresentValue (see page 15965)	Calculates the present value of an investment.
	RadToCycle (see page 15966)	Converts radians to cycles.

	RadToDeg (see page 15966)	Converts radians to degrees.
	RadToGrad (see page 15966)	Converts radians to grads.
	RandG (see page 15967)	Generates random numbers with Gaussian distribution.
	RandomFrom (see page 15967)	Returns a randomly selected element from an array.
	RandomFrom (see page 15967)	Returns a randomly selected element from an array.
	RandomFrom (see page 15967)	Returns a randomly selected element from an array.
	RandomRange (see page 15968)	Returns a random integer from a specified range.
	RoundTo (see page 15968)	Rounds a floating-point value to a specified digit or power of ten using "Banker's rounding".
	SameValue (see page 15969)	Indicates whether two floating-point values are (approximately) equal.
	SameValue (see page 15969)	Indicates whether two floating-point values are (approximately) equal.
	SameValue (see page 15969)	Indicates whether two floating-point values are (approximately) equal.
	Sec (see page 15970)	Calculates the secant of an angle.
	Secant (see page 15970)	Calculates the secant of an angle.
	SecH (see page 15971)	Calculates the hyperbolic secant of an angle.
	SetExceptionMask (see page 15971)	Sets the exception mask on the FPU control word.
	SetPrecisionMode (see page 15972)	Sets the FPU precision control mode.
	SetRoundMode (see page 15972)	Sets the FPU rounding mode.
	Sign (see page 15973)	Indicates whether a numeric value is positive, negative, or zero.
	Sign (see page 15973)	Indicates whether a numeric value is positive, negative, or zero.
	Sign (see page 15973)	Indicates whether a numeric value is positive, negative, or zero.
	SimpleRoundTo (see page 15973)	Rounds a floating-point value to a specified digit or power of ten using asymmetric arithmetic rounding.
	SinCos (see page 15974)	Returns sine and cosine of an angle.
	Sinh (see page 15974)	Returns the hyperbolic sine of an angle.
	SLNDepreciation (see page 15975)	Returns the straight-line depreciation allowance of an asset.
	StdDev (see page 15975)	Returns the sample standard deviation for elements in an array.
	Sum (see page 15976)	Returns the sum of the elements in an array.
	SumInt (see page 15976)	Returns the sum of the elements in an integer array.
	SumOfSquares (see page 15977)	Returns the sum of the squared values from a data array.
	SumsAndSquares (see page 15977)	Returns the sum of the values and the sum of the squared values in an array.
	SYDDepreciation (see page 15978)	Calculates depreciation for an asset.
	Tan (see page 15978)	Returns the tangent of X.
	Tanh (see page 15979)	Returns the hyperbolic tangent of X.
	TotalVariance (see page 15979)	Returns the statistical variance from an array of values.
	Variance (see page 15980)	Calculates statistical sample variance from an array of data.

Structs, Records, Enums

	Name	Description
	TFPUException (see page 15981)	TFPUException defines TFPUExceptionMask (see page 15985) values.
	TFPUPrecisionMode (see page 15982)	TFPUPrecisionMode defines how the FPU handles precision.
	TFPURoundingMode (see page 15983)	TFPURoundingMode defines how the FPU handles rounding issues.
	TPaymentTime (see page 15984)	TPaymentTime is an enumerated type used as a parameter in financial functions.

Types

Name	Description
TFPUExceptionMask (see page 15985)	TFPUExceptionMask defines a set of TFPUException (see page 15981) values.
TRoundToRange (see page 15986)	TRoundToRange defines the RoundTo (see page 15968) rounding limits.
TValueSign (see page 15986)	TValueSign defines the Sign (see page 15973) returned value.

93.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EInvalidArgument (see page 15927)	EInvalidArgument is the exception class for out-of-range parameters in specialized math and accounting functions.

93.1.1 EInvalidArgument Class

EInvalidArgument is the exception class for out-of-range parameters in specialized math and accounting functions.

Class Hierarchy

System::TObject → SysUtils.Exception → SysUtils.EExternal → SysUtils.EMathError → Math.EInvalidArgument

Pascal

```
EInvalidArgument = class(EMathError);
```

C++

```
class EInvalidArgument : public EMathError;
```

File

Math

Description

EInvalidArgument is raised by some functions in the Math (see page 15923) unit (mostly financial functions) when one of the parameters is out of range or makes the calculation impossible.

Note: On Windows, because EInvalidArgument descends (indirectly) from EExternal, it saves the Windows exception record (if there is one) in its ExceptionRecord data structure.

See Also

EExternal (see page 22727)

DoubleDecliningBalance (see page 15944)

FutureValue (see page 15946)

InterestPayment (see page 15951)

NetPresentValue (see page 15961)

PeriodPayment (see page 15963)

Power (see page 15965)

SLNDepreciation (see page 15975)

SYDDepreciation (see page 15978)

EInvalidOp (see page 22738)

EOverflow (see page 22743)

EUnderflow

EZeroDivide (see page 22761)

93.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	ArcCos (see page 15931)	Calculates the inverse cosine of a given number.
	ArcCos (see page 15931)	Calculates the inverse cosine of a given number.
	ArcCos (see page 15931)	Calculates the inverse cosine of a given number.
	ArcCosh (see page 15932)	Calculates the inverse hyperbolic cosine of a given number.
	ArcCot (see page 15932)	Calculates the inverse cotangent of a given number.
	ArcCotH (see page 15933)	Calculates the inverse hyperbolic cotangent of a given number.
	ArcCsc (see page 15933)	Calculates the inverse cosecant of a given number.
	ArcCscH (see page 15934)	Calculates the inverse hyperbolic cosecant of a given number.
	ArcSec (see page 15934)	Calculates the inverse secant of a given number.
	ArcSecH (see page 15934)	Calculates the inverse hyperbolic secant of a given number.
	ArcSin (see page 15935)	Calculates the inverse sine of a given number.
	ArcSin (see page 15935)	Calculates the inverse sine of a given number.
	ArcSin (see page 15935)	Calculates the inverse sine of a given number.
	ArcSinh (see page 15935)	Calculates the inverse hyperbolic sine of a given number.
	ArcTan2 (see page 15936)	Calculates the arctangent angle and quadrant of a given number.
	ArcTanh (see page 15936)	Calculates the inverse hyperbolic tangent of a given number.
	Ceil (see page 15937)	Rounds variables up toward positive infinity.
	ClearExceptions (see page 15937)	Clears exception bits in FPU status word.
	CompareValue (see page 15938)	Returns the relationship between two numeric values.
	CompareValue (see page 15938)	Returns the relationship between two numeric values.
	CompareValue (see page 15938)	Returns the relationship between two numeric values.
	CompareValue (see page 15938)	Returns the relationship between two numeric values.
	CompareValue (see page 15938)	Returns the relationship between two numeric values.
	Cosecant (see page 15938)	Returns the cosecant of an angle.
	Cosh (see page 15939)	Calculates the hyperbolic cosine of an angle.
	Cot (see page 15939)	Calculates the cotangent of an angle.
	Cotan (see page 15940)	Calculates the cotangent of an angle.
	CotH (see page 15940)	Calculates the hyperbolic cotangent of an angle.
	Csc (see page 15941)	Returns the cosecant of an angle.
	CscH (see page 15941)	Returns the hyperbolic cosecant of an angle.
	CycleToDeg (see page 15942)	Converts an angle measurement from cycles to degrees.
	CycleToGrad (see page 15942)	Converts an angle measurement from cycles to grads.
	CycleToRad (see page 15942)	Converts an angle measurement from cycles to radians.
	DegToCycle (see page 15943)	Returns the value of a degree measurement expressed in cycles.
	DegToGrad (see page 15943)	Returns the value of a degree measurement expressed in grads.
	DegToRad (see page 15944)	Returns the value of a degree measurement expressed in radians.
	DivMod (see page 15944)	Returns the result of an integer division, including the remainder.
	DoubleDecliningBalance (see page 15944)	Calculates the depreciation of an asset using the double-declining balance method.
	EnsureRange (see page 15945)	Returns the closest value to a specified value within a specified range.
	EnsureRange (see page 15945)	Returns the closest value to a specified value within a specified range.
	EnsureRange (see page 15945)	Returns the closest value to a specified value within a specified range.
	Floor (see page 15945)	Rounds variables toward negative infinity.
	Frexp (see page 15946)	Separates the Mantissa and Exponent of X
	FutureValue (see page 15946)	Calculates the future value of an investment.

	GetExceptionMask (see page 15947)	Returns the exception mask from the FPU control word.
	GetPrecisionMode (see page 15947)	Returns the FPU precision control mode.
	GetRoundMode (see page 15948)	Returns the FPU rounding mode.
	GradToCycle (see page 15949)	Converts grad measurements to cycles.
	GradToDeg (see page 15949)	Converts grad measurements to degrees.
	GradToRad (see page 15949)	Converts grad measurements to radians.
	Hypot (see page 15950)	Calculates the length of the hypotenuse.
	IfThen (see page 15950)	Conditionally returns one of two specified values.
	IfThen (see page 15950)	Conditionally returns one of two specified values.
	IfThen (see page 15950)	Conditionally returns one of two specified values.
	InRange (see page 15950)	Indicates whether a value falls within a specified range.
	InRange (see page 15950)	Indicates whether a value falls within a specified range.
	InRange (see page 15950)	Indicates whether a value falls within a specified range.
	InterestPayment (see page 15951)	Calculates the interest portion of a loan payment.
	InterestRate (see page 15952)	Returns the interest rate required to increase PresentValue (see page 15965) to FutureValue (see page 15946).
	InternalRateOfReturn (see page 15952)	Calculates the internal rate of return for an investment.
	IntPower (see page 15953)	Calculates the integral power of a base value.
	IsInfinite (see page 15953)	Indicates when a variable or expression represents an infinite value.
	IsNan (see page 15954)	Indicates when a variable or expression does not evaluate to a numeric value.
	IsNan (see page 15954)	Indicates when a variable or expression does not evaluate to a numeric value.
	IsNan (see page 15954)	Indicates when a variable or expression does not evaluate to a numeric value.
	IsZero (see page 15954)	Indicates when a floating-point variable or expression evaluates to zero, or very close to zero.
	IsZero (see page 15954)	Indicates when a floating-point variable or expression evaluates to zero, or very close to zero.
	IsZero (see page 15954)	Indicates when a floating-point variable or expression evaluates to zero, or very close to zero.
	Ldexp (see page 15955)	Calculates X times (2 to the power of P).
	LnXP1 (see page 15955)	Returns the natural log of (X+1)
	Log10 (see page 15955)	Calculates log base 10.
	Log2 (see page 15956)	Calculates log base 2.
	LogN (see page 15956)	Calculates the log of X for a specified base.
	Max (see page 15957)	Returns the greater of two numeric values (Delphi only).
	Max (see page 15957)	Returns the greater of two numeric values (Delphi only).
	Max (see page 15957)	Returns the greater of two numeric values (Delphi only).
	Max (see page 15957)	Returns the greater of two numeric values (Delphi only).
	Max (see page 15957)	Returns the greater of two numeric values (Delphi only).
	MaxIntValue (see page 15957)	Returns the largest signed value in an integer array.
	MaxValue (see page 15958)	Returns the largest signed value in an array.
	Mean (see page 15958)	Returns the average of all values in an array.
	MeanAndStdDev (see page 15959)	Calculates the mean and standard deviation of array elements.
	Min (see page 15959)	Returns the lesser of two numeric values (Delphi only).
	Min (see page 15959)	Returns the lesser of two numeric values (Delphi only).
	Min (see page 15959)	Returns the lesser of two numeric values (Delphi only).
	Min (see page 15959)	Returns the lesser of two numeric values (Delphi only).
	Min (see page 15959)	Returns the lesser of two numeric values (Delphi only).
	MinIntValue (see page 15960)	Returns the smallest signed value in an integer array.
	MinValue (see page 15960)	Returns smallest signed value in an array.
	MomentSkewKurtosis (see page 15961)	Calculates the mean, variance, skew, and kurtosis.
	NetPresentValue (see page 15961)	Calculates the current value from an array of estimated cash flow values.
	Norm (see page 15962)	Returns the Euclidean 'L-2' norm.
	NumberOfPeriods (see page 15962)	Returns the number of payment periods for an investment.
	Payment (see page 15963)	Calculates a fully amortized payment.
	PeriodPayment (see page 15963)	Returns the principal amount from a full payment.
	Poly (see page 15963)	Evaluates a uniform polynomial of one variable at the value X.

	PopnStdDev (see page 15964)	Calculates the population standard deviation.
	PopnVariance (see page 15964)	Calculates the population variance.
	Power (see page 15965)	Raises Base to any power.
	Power (see page 15965)	Raises Base to any power.
	Power (see page 15965)	Raises Base to any power.
	PresentValue (see page 15965)	Calculates the present value of an investment.
	RadToCycle (see page 15966)	Converts radians to cycles.
	RadToDeg (see page 15966)	Converts radians to degrees.
	RadToGrad (see page 15966)	Converts radians to grads.
	RandG (see page 15967)	Generates random numbers with Gaussian distribution.
	RandomFrom (see page 15967)	Returns a randomly selected element from an array.
	RandomFrom (see page 15967)	Returns a randomly selected element from an array.
	RandomFrom (see page 15967)	Returns a randomly selected element from an array.
	RandomRange (see page 15968)	Returns a random integer from a specified range.
	RoundTo (see page 15968)	Rounds a floating-point value to a specified digit or power of ten using "Banker's rounding".
	SameValue (see page 15969)	Indicates whether two floating-point values are (approximately) equal.
	SameValue (see page 15969)	Indicates whether two floating-point values are (approximately) equal.
	SameValue (see page 15969)	Indicates whether two floating-point values are (approximately) equal.
	Sec (see page 15970)	Calculates the secant of an angle.
	Secant (see page 15970)	Calculates the secant of an angle.
	SecH (see page 15971)	Calculates the hyperbolic secant of an angle.
	SetExceptionMask (see page 15971)	Sets the exception mask on the FPU control word.
	SetPrecisionMode (see page 15972)	Sets the FPU precision control mode.
	SetRoundMode (see page 15972)	Sets the FPU rounding mode.
	Sign (see page 15973)	Indicates whether a numeric value is positive, negative, or zero.
	Sign (see page 15973)	Indicates whether a numeric value is positive, negative, or zero.
	Sign (see page 15973)	Indicates whether a numeric value is positive, negative, or zero.
	SimpleRoundTo (see page 15973)	Rounds a floating-point value to a specified digit or power of ten using asymmetric arithmetic rounding.
	SinCos (see page 15974)	Returns sine and cosine of an angle.
	Sinh (see page 15974)	Returns the hyperbolic sine of an angle.
	SLNDepreciation (see page 15975)	Returns the straight-line depreciation allowance of an asset.
	StdDev (see page 15975)	Returns the sample standard deviation for elements in an array.
	Sum (see page 15976)	Returns the sum of the elements in an array.
	SumInt (see page 15976)	Returns the sum of the elements in an integer array.
	SumOfSquares (see page 15977)	Returns the sum of the squared values from a data array.
	SumsAndSquares (see page 15977)	Returns the sum of the values and the sum of the squared values in an array.
	SYDDepreciation (see page 15978)	Calculates depreciation for an asset.
	Tan (see page 15978)	Returns the tangent of X.
	Tanh (see page 15979)	Returns the hyperbolic tangent of X.
	TotalVariance (see page 15979)	Returns the statistical variance from an array of values.
	Variance (see page 15980)	Calculates statistical sample variance from an array of data.

93.2.1 Math.ArcCos Function

Calculates the inverse cosine of a given number.

Pascal

```
function ArcCos(const X: Extended): Extended; overload;
function ArcCos(const X: Double): Double; overload;
function ArcCos(const X: Single): Single; overload;
```

C++

```
Extended ArcCos(const Extended X);
Double ArcCos(const Double X);
Single ArcCos(const Single X);
```

File

Math

Description

ArcCos returns the inverse cosine of X.

X must be between -1 and 1.

The return value is in the range [0..Pi], in radians.

See Also

Cos

ArcCosh (see page 15932)

ArcSin (see page 15935)

93.2.2 Math.ArcCosh Function

Calculates the inverse hyperbolic cosine of a given number.

Pascal

```
function ArcCosh(const X: Extended): Extended;
```

C++

```
Extended ArcCosh(const Extended X);
```

File

Math

Description

ArcCosh returns the inverse hyperbolic cosine of X. The value of X must be greater than or equal to 1.

See Also

Cosh (see page 15939)

ArcCos (see page 15931)

93.2.3 Math.ArcCot Function

Calculates the inverse cotangent of a given number.

Pascal

```
function ArcCot(const X: Extended): Extended;
```

C++

```
Extended ArcCot(const Extended X);
```

File

Math

Description

ArcCot returns the inverse cotangent of X.

See Also

Cot (see page 15939)

ArcCoth

93.2.4 Math.ArcCotH Function

Calculates the inverse hyperbolic cotangent of a given number.

Pascal

```
function ArcCotH(const X: Extended): Extended;
```

C++

```
Extended ArcCotH(const Extended X);
```

File

Math

Description

ArcCot (see page 15932) returns the inverse hyperbolic cotangent of X.

See Also

Cot (see page 15939)

ArcCoth

93.2.5 Math.ArcCsc Function

Calculates the inverse cosecant of a given number.

Pascal

```
function ArcCsc(const X: Extended): Extended;
```

C++

```
Extended ArcCsc(const Extended X);
```

File

Math

Description

ArcCsc returns the inverse cosecant of X.

See Also

Csc (see page 15941)

ArcCsch

93.2.6 Math.ArcCscH Function

Calculates the inverse hyperbolic cosecant of a given number.

Pascal

```
function ArcCscH(const X: Extended): Extended;
```

C++

```
Extended ArcCscH(const Extended X);
```

File

Math

Description

ArcCsc (see page 15933) returns the inverse hyperbolic cosecant of X.

See Also

Csc (see page 15941)

Csch

93.2.7 Math.ArcSec Function

Calculates the inverse secant of a given number.

Pascal

```
function ArcSec(const X: Extended): Extended;
```

C++

```
Extended ArcSec(const Extended X);
```

File

Math

Description

ArcSec returns the inverse secant of X.

See Also

Sec (see page 15970)

ArcSech

93.2.8 Math.ArcSecH Function

Calculates the inverse hyperbolic secant of a given number.

Pascal

```
function ArcSecH(const X: Extended): Extended;
```

C++

```
Extended ArcSecH(const Extended X);
```

File

Math

Description

ArcSec (see page 15934) returns the inverse hyperbolic secant of X.

See Also

Sec (see page 15970)

Sech

93.2.9 Math.ArcSin Function

Calculates the inverse sine of a given number.

Pascal

```
function ArcSin(const X: Extended): Extended; overload;
function ArcSin(const X: Double): Double; overload;
function ArcSin(const X: Single): Single; overload;
```

C++

```
Extended ArcSin(const Extended X);
Double ArcSin(const Double X);
Single ArcSin(const Single X);
```

File

Math

Description

ArcSin returns the inverse sine of X. X must be between -1 and 1. The return value will be in the range [-Pi/2..Pi/2], in radians.

See Also

Sin

ArcSinh (see page 15935)

ArcCos (see page 15931)

93.2.10 Math.ArcSinh Function

Calculates the inverse hyperbolic sine of a given number.

Pascal

```
function ArcSinh(const X: Extended): Extended;
```

C++

```
Extended ArcSinh(const Extended X);
```

File

Math

Description

ArcSinh returns the inverse hyperbolic sine of X.

See Also

Sinh (see page 15974)

93.2.11 Math.ArcTan2 Function

Calculates the arctangent angle and quadrant of a given number.

Pascal

```
function ArcTan2(const Y: Extended; const X: Extended): Extended;
```

C++

```
Extended ArcTan2(const Extended Y, const Extended X);
```

File

Math

Description

ArcTan2 calculates ArcTan(Y/X), and returns an angle in the correct quadrant. The values of X and Y must be between –2^64 and 2^64. In addition, the value of X can't be 0. The return value will fall in the range from -Pi to Pi radians.

93.2.12 Math.ArcTanh Function

Calculates the inverse hyperbolic tangent of a given number.

Pascal

```
function ArcTanh(const X: Extended): Extended;
```

C++

```
Extended ArcTanh(const Extended X);
```

File

Math

Description

ArcTanh returns the inverse hyperbolic tangent of X. The value of X must be between -1 and 1 (inclusive).

See Also

Tanh (see page 15979)

93.2.13 Math.Ceil Function

Rounds variables up toward positive infinity.

Pascal

```
function Ceil(const X: Extended): Integer;
```

C++

```
int Ceil(const Extended X);
```

File

Math

Description

Call Ceil to obtain the lowest integer greater than or equal to X. The absolute value of X must be less than MaxInt. For example:

Ceil(-2.8) = -2

Ceil(2.8) = 3

Ceil(-1.0) = -1

See Also

Trunc (see page 22537)

Floor (see page 15945)

Ceil

93.2.14 Math.ClearExceptions Function

Clears exception bits in FPU status word.

Pascal

```
procedure ClearExceptions(RaisePending: Boolean = True);
```

C++

```
ClearExceptions(Boolean RaisePending = True);
```

File

Math

Description

ClearExceptions clears the exception bits in the FPU status word. If RaisePending is true, any pending floating point exceptions cause an error to be raised.

See Also

GetExceptionMask (see page 15947)

SetExceptionMask (see page 15971)

Set8087CW

93.2.15 Math.CompareValue Function

Returns the relationship between two numeric values.

Pascal

```
function CompareValue(const A: Extended; const B: Extended; Epsilon: Extended = 0):
TValueRelationship; overload;
function CompareValue(const A: Double; const B: Double; Epsilon: Double = 0):
TValueRelationship; overload;
function CompareValue(const A: Single; const B: Single; Epsilon: Single = 0):
TValueRelationship; overload;
function CompareValue(const A: Integer; const B: Integer): TValueRelationship; overload;
function CompareValue(const A: Int64; const B: Int64): TValueRelationship; overload;
```

C++

```
TValueRelationship CompareValue(const Extended A, const Extended B, Extended Epsilon = 0);
TValueRelationship CompareValue(const Double A, const Double B, Double Epsilon = 0);
TValueRelationship CompareValue(const Single A, const Single B, Single Epsilon = 0);
TValueRelationship CompareValue(const int A, const int B);
TValueRelationship CompareValue(const Int64 A, const Int64 B);
```

File

Math

Description

Call CompareValue to determine the relationship between two numeric values. When comparing floating-point values, CompareValue lets you specify a maximum difference to use when comparing values, so that they are considered the same if they are within that amount.

A and B are the values to compare.

Epsilon is the maximum amount by which A and B can differ and still be considered the same value.

CompareValue returns

LessThanValue if A is less than B (by more than Epsilon if A and B are floating-point numbers).

EqualsValue if A is equivalent to B (the same, or within Epsilon if A and B are floating-point numbers).

GreaterThanValue if A is larger than B (by more than Epsilon if A and B are floating-point numbers).

See Also

SameValue (see page 15969)

VarCompareValue

93.2.16 Math.Cosecant Function

Returns the cosecant of an angle.

Pascal

```
function Cosecant(const X: Extended): Extended;
```

C++

```
Extended Cosecant(const Extended X);
```

File

Math

Description

Use the Cosecant to calculate the cosecant of X, where X is an angle in radians. The cosecant is calculated as 1/ Sin(X).

See Also

ArcCsc (see page 15933)

CscH (see page 15941)

Secant (see page 15970)

Sin

93.2.17 Math.Cosh Function

Calculates the hyperbolic cosine of an angle.

Pascal

```
function Cosh(const X: Extended): Extended;
```

C++

```
Extended Cosh(const Extended X);
```

File

Math

Description

Use the Cosh to calculate the hyperbolic cosine of X.

See Also

Cos

ArcCosh (see page 15932)

Sinh (see page 15974)

93.2.18 Math.Cot Function

Calculates the cotangent of an angle.

Pascal

```
function Cot(const X: Extended): Extended;
```

C++

```
Extended Cot(const Extended X);
```

File

Math

Description

Call Cot to obtain the cotangent of X. The cotangent is calculated using the formula 1 / Tan (see page 15978)(X).

See Also

Tan (see page 15978)

Sin

Cos

ArcTan

93.2.19 Math.Cotan Function

Calculates the cotangent of an angle.

Pascal

```
function Cotan(const X: Extended): Extended;
```

C++

```
Extended Cotan(const Extended X);
```

File

Math

Description

Call Cotan to obtain the cotangent of X. The cotangent is calculated using the formula

1 / Tan (see page 15978)(X)

Note: Do not call Cotan with X = 0.

See Also

Tan (see page 15978)

Sin

Cos

ArcTan

93.2.20 Math.CotH Function

Calculates the hyperbolic cotangent of an angle.

Pascal

```
function CotH(const X: Extended): Extended;
```

C++

```
Extended CotH(const Extended X);
```

File

Math

Description

Call CotH to obtain the hyperbolic cotangent of X, where X is an angle in Radians.

See Also

Cot (see page 15939)

ArcCot (see page 15932)

93.2.21 Math.Csc Function

Returns the cosecant of an angle.

Pascal

```
function Csc(const X: Extended): Extended;
```

C++

```
Extended Csc(const Extended X);
```

File

Math

Description

Use the Csc to calculate the cosecant of X, where X is an angle in radians.

See Also

ArcCsc (see page 15933)

CscH (see page 15941)

Sec (see page 15970)

Sin

93.2.22 Math.CscH Function

Returns the hyperbolic cosecant of an angle.

Pascal

```
function CscH(const X: Extended): Extended;
```

C++

```
Extended CscH(const Extended X);
```

File

Math

Description

Use the CscH to calculate the hyperbolic cosecant of X, where X is an angle in radians.

See Also

ArcCscH (see page 15934)

Csc (see page 15941)

SecH (see page 15971)

93.2.23 Math.CycleToDeg Function

Converts an angle measurement from cycles to degrees.

Pascal

```
function CycleToDeg(const Cycles: Extended): Extended;
```

C++

```
Extended CycleToDeg(const Extended Cycles);
```

File

Math

Description

CycleToDeg converts angles measured in cycles into degrees, where

degrees = cycles * 360.

93.2.24 Math.CycleToGrad Function

Converts an angle measurement from cycles to grads.

Pascal

```
function CycleToGrad(const Cycles: Extended): Extended;
```

C++

```
Extended CycleToGrad(const Extended Cycles);
```

File

Math

Description

CycleToGrad converts angles measured in cycles into grads.

93.2.25 Math.CycleToRad Function

Converts an angle measurement from cycles to radians.

Pascal

```
function CycleToRad(const Cycles: Extended): Extended;
```

C++

```
Extended CycleToRad(const Extended Cycles);
```

File

Math

Description

CycleToRad converts angles measured in cycles into radians, where

radians = 2pi * cycles.

93.2.26 Math.DegToCycle Function

Returns the value of a degree measurement expressed in cycles.

Pascal

```
function DegToCycle(const Degrees: Extended): Extended;
```

C++

```
Extended DegToCycle(const Extended Degrees);
```

File

Math

Description

Use DegToCycle to convert angles expressed in degrees to the corresponding value in cycles.

93.2.27 Math.DegToGrad Function

Returns the value of a degree measurement expressed in grads.

Pascal

```
function DegToGrad(const Degrees: Extended): Extended;
```

C++

```
Extended DegToGrad(const Extended Degrees);
```

File

Math

Description

Use DegToGrad to convert angles expressed in degrees to the corresponding value in grads.

93.2.28 Math.DegToRad Function

Returns the value of a degree measurement expressed in radians.

Pascal

```
function DegToRad(const Degrees: Extended): Extended;
```

C++

```
Extended DegToRad(const Extended Degrees);
```

File

Math

Description

Use DegToRad to convert angles expressed in degrees to the corresponding value in radians, where

radians = degrees(pi/180).

93.2.29 Math.DivMod Function

Returns the result of an integer division, including the remainder.

Pascal

```
procedure DivMod(Dividend: Cardinal; Divisor: Word; var Result: Word; var Remainder: Word);
```

C++

```
DivMod(unsigned Dividend, Word Divisor, Word Result, Word Remainder);
```

File

Math

Description

Call DivMod to perform 16-bit integer division and fetch the remainder as well, all in one operation.

Dividend is the integer into which you are dividing.

Divisor is the value by which to divide Dividend.

Result returns the result of the integer division.

Remainder returns the remainder (the difference between Result * Divisor and Dividend).

93.2.30 Math.DoubleDecliningBalance Function

Calculates the depreciation of an asset using the double-declining balance method.

Pascal

```
function DoubleDecliningBalance(const Cost: Extended; const Salvage: Extended; Life: Integer;
Period: Integer): Extended;
```

C++

```
Extended DoubleDecliningBalance(const Extended Cost, const Extended Salvage, int Life, int
Period);
```

File

Math

Description

DoubleDecliningBalance determines accelerated depreciation values for an asset, given the initial cost, life expectancy, end value, and depreciation period.

93.2.31 Math.EnsureRange Function

Returns the closest value to a specified value within a specified range.

Pascal

```
function EnsureRange(const AValue: Integer; const AMin: Integer; const AMax: Integer):
Integer; overload;
function EnsureRange(const AValue: Int64; const AMin: Int64; const AMax: Int64): Int64;
overload;
function EnsureRange(const AValue: Double; const AMin: Double; const AMax: Double): Double;
overload;
```

C++

```
int EnsureRange(const int AValue, const int AMin, const int AMax);
Int64 EnsureRange(const Int64 AValue, const Int64 AMin, const Int64 AMax);
Double EnsureRange(const Double AValue, const Double AMin, const Double AMax);
```

File

Math

Description

EnsureRange returns a value that is guaranteed to be within a specified range.

AValue is the value you want as long as it is in range. If AValue is in range, EnsureRange returns AValue.

AMin is the minimum acceptable value. If AValue is less than AMin, EnsureRange returns AMin.

AMax is the maximum acceptable value. If AValue is greater than AMax, EnsureRange returns AMax.

See Also

VarEnsureRange

InRange (see page 15950)

93.2.32 Math.Floor Function

Rounds variables toward negative infinity.

Pascal

```
function Floor(const X: Extended): Integer;
```

C++

```
int Floor(const Extended X);
```

File

Math

Description

Call Floor to obtain the highest integer less than or equal to X. For example:

Floor(-2.8) = -3

Floor(2.8) = 2

Floor(-1.0) = -1

Note: The absolute value of X must be less than MaxInt.

See Also

Trunc (see page 22537)

Ceil (see page 15937)

93.2.33 Math.Frexp Function

Separates the Mantissa and Exponent of X

Pascal

```
procedure Frexp(const X: Extended; var Mantissa: Extended; var Exponent: Integer);
```

C++

```
Frexp(const Extended X, Extended Mantissa, int Exponent);
```

File

Math

Description

Frexp returns the mantissa of X as Mantissa and the exponent as Exponent.

93.2.34 Math.FutureValue Function

Calculates the future value of an investment.

Pascal

```
function FutureValue(const Rate: Extended; NPeriods: Integer; const Payment: Extended; const
PresentValue: Extended; PaymentTime: TPaymentTime): Extended;
```

C++

```
Extended FutureValue(const Extended Rate, int NPeriods, const Extended Payment, const Extended
PresentValue, TPaymentTime PaymentTime);
```

File

Math

Description

FutureValue returns the future value of an investment of PresentValue (see page 15965) where Payment (see page 15963) is invested for NPeriods at the rate of Rate per period. The PaymentTime parameter indicates whether the investment is an ordinary annuity or an annuity due (enter ptEndOfPeriod if payments are at the end of each period, ptStartOfPeriod if they are at the beginning).

93.2.35 Math.GetExceptionMask Function

Returns the exception mask from the FPU control word.

Pascal

```
function GetExceptionMask: TFPUExceptionMask;
```

C++

```
TFPUExceptionMask GetExceptionMask();
```

File

Math

Description

Call GetExceptionMask to ascertain the types of exceptions the FPU (floating-point unit) generates. Any element set in this mask prevents the FPU from raising the corresponding type of exception. Instead, the FPU returns its best attempt at a value – often NaN (see page 15992), Infinity (see page 15988), or NegInfinity (see page 15993). The value depends on the operation and the current rounding mode.

See Also

Set8087CW

SetExceptionMask (see page 15971)

NaN (see page 15992)

Infinity (see page 15988)

NegInfinity (see page 15993)

IsInfinite (see page 15953)

IsNaN

93.2.36 Math.GetPrecisionMode Function

Returns the FPU precision control mode.

Pascal

```
function GetPrecisionMode: TFPUPrecisionMode;
```

C++

```
TFPUPrecisionMode GetPrecisionMode();
```

File

Math

Description

Call GetPrecisionMode to ascertain the level of precision that the FPU (floating-point unit) uses for floating-point calculations. The precision control mode can be any of the following values:

Value	Meaning
pmSingle	single precision
pmReserved	not used
pmDouble	double precision
pmExtended	extended precision

See Also

Set8087CW

SetPrecisionMode (see page 15972)

93.2.37 Math.GetRoundMode Function

Returns the FPU rounding mode.

Pascal

```
function GetRoundMode: TFPURoundingMode;
```

C++

```
TFPURoundingMode GetRoundMode();
```

File

Math

Description

Call GetRoundingMode to ascertain how the FPU handles rounding issues. The rounding mode can be any of the following values:

Value	Meaning
rmNearest	Rounds to the closest value.
rmDown	Rounds toward negative infinity.
rmUp	Rounds toward positive infinity.
rmTruncate	Truncates the value, rounding positive numbers down and negative numbers up.

See Also

Set8087CW

SetRoundMode (see page 15972)

93.2.38 Math.GradToCycle Function

Converts grad measurements to cycles.

Pascal

```
function GradToCycle(const Grads: Extended): Extended;
```

C++

```
Extended GradToCycle(const Extended Grads);
```

File

Math

Description

GradToCycle converts angles measured in grads into cycles.

93.2.39 Math.GradToDeg Function

Converts grad measurements to degrees.

Pascal

```
function GradToDeg(const Grads: Extended): Extended;
```

C++

```
Extended GradToDeg(const Extended Grads);
```

File

Math

Description

GradToDeg converts angles measured in grads into degrees.

93.2.40 Math.GradToRad Function

Converts grad measurements to radians.

Pascal

```
function GradToRad(const Grads: Extended): Extended;
```

C++

```
Extended GradToRad(const Extended Grads);
```

File

Math

Description

GradToRad converts angles measured in grads into radians, where

radians = grads(pi/200).

93.2.41 Math.Hypot Function

Calculates the length of the hypotenuse.

Pascal

```
function Hypot(const X: Extended; const Y: Extended): Extended;
```

C++

```
Extended Hypot(const Extended X, const Extended Y);
```

File

Math

Description

Hypot returns the length of the hypotenuse of a right triangle. Specify the lengths of the sides adjacent to the right angle in X and Y. Hypot uses the formula

Sqrt(X**2 + Y**2)

93.2.42 Math.IfThen Function

Conditionally returns one of two specified values.

Pascal

```
function IfThen(AValue: Boolean; const ATrue: Integer; const AFalse: Integer = 0): Integer;
overload;
function IfThen(AValue: Boolean; const ATrue: Int64; const AFalse: Int64 = 0): Int64; overload;
function IfThen(AValue: Boolean; const ATrue: Double; const AFalse: Double = 0.0): Double;
overload;
```

C++

```
int IfThen(Boolean AValue, const int ATrue, const int AFalse = 0);
Int64 IfThen(Boolean AValue, const Int64 ATrue, const Int64 AFalse = 0);
Double IfThen(Boolean AValue, const Double ATrue, const Double AFalse = 0.0);
```

File

Math

Description

IfThen checks the expression passed as AValue and returns ATrue if it evaluates to true, or AFalse if it evaluates to false. In Delphi, if the AFalse parameter is omitted, IfThen returns 0 or an empty string when AValue evaluates to False.

93.2.43 Math.InRange Function

Indicates whether a value falls within a specified range.

Pascal

```
function InRange(const AValue: Integer; const AMin: Integer; const AMax: Integer): Boolean;
overload;
function InRange(const AValue: Int64; const AMin: Int64; const AMax: Int64): Boolean; overload;
function InRange(const AValue: Double; const AMin: Double; const AMax: Double): Boolean;
overload;
```

C++

```
Boolean InRange(const int AValue, const int AMin, const int AMax);
Boolean InRange(const Int64 AValue, const Int64 AMin, const Int64 AMax);
Boolean InRange(const Double AValue, const Double AMin, const Double AMax);
```

File

Math

Description

InRange returns true if AValue is greater than or equal to AMin and less than or Equal to AMax. It returns false if AValue is less than AMin or greater than AMax.

See Also

VarInRange

EnsureRange (see page 15945)

93.2.44 Math.InterestPayment Function

Calculates the interest portion of a loan payment.

Pascal

```
function InterestPayment(const Rate: Extended; Period: Integer; NPeriods: Integer; const
PresentValue: Extended; const FutureValue: Extended; PaymentTime: TPaymentTime): Extended;
```

C++

```
Extended InterestPayment(const Extended Rate, int Period, int NPeriods, const Extended
PresentValue, const Extended FutureValue, TPaymentTime PaymentTime);
```

File

Math

Description

Call InterestPayment to calculate the portion of a loan payment that reflects the interest.

Rate represents the fixed periodic interest rate.

Period identifies the payment period.

NPeriods is the number of periods of the loan.

PresentValue (see page 15965) represents the amount borrowed (the principal).

FutureValue (see page 15946) is the future value of the investment.

PaymentTime indicates whether the cash flows occur at the beginning or end of the period.

93.2.45 Math.InterestRate Function

Returns the interest rate required to increase PresentValue (see page 15965) to FutureValue (see page 15946).

Pascal

```
function InterestRate(NPeriods: Integer; const Payment: Extended; const PresentValue:
Extended; const FutureValue: Extended; PaymentTime: TPaymentTime): Extended;
```

C++

```
Extended InterestRate(int NPeriods, const Extended Payment, const Extended PresentValue, const
Extended FutureValue, TPaymentTime PaymentTime);
```

File

Math

Description

InterestRate calculates the interest rate required in order for an investment of PresentValue (see page 15965), with periodic payments of Payment (see page 15963), to be worth FutureValue (see page 15946) within NPeriods compounding periods. If NPeriods represents years, an annual interest rate results; if NPeriods represents months, a monthly interest rate results, and so on. The PaymentTime parameter indicates whether the cash flows occur at the beginning or end of the period.

93.2.46 Math.InternalRateOfReturn Function

Calculates the internal rate of return for an investment.

Pascal

```
function InternalRateOfReturn(const Guess: Extended; const CashFlows: array of Double):
Extended;
```

C++

```
Extended InternalRateOfReturn(const Extended Guess, const array of Double CashFlows);
```

File

Math

Description

InternalRateOfReturn determines the internal rate of return on an investment. It references an array that contains cash flow information and uses the supplied internal rate of return estimate to calculate results.

Before using this function, define an array containing expected cash flow amounts over a period of time. It is assumed that the amounts are received at regular intervals. Negative amounts are interpreted as cash outflows, and positive amounts as inflows. The first amount must be a negative number, to reflect the initial investment. The following amounts can all be the same for each time period, or they can be different (including a mixture of negatives, positives, or zeros).

Specify the estimated rate of return as the Guess parameter. Pass the array of expected cash flow amounts as the CashFlows parameter. In C++, the CashFlows_Size parameter gives the index of the last entry in CashFlows (one less than the number of entries).

93.2.47 Math.IntPower Function

Calculates the integral power of a base value.

Pascal

```
function IntPower(const Base: Extended; const Exponent: Integer): Extended;
```

C++

```
Extended IntPower(const Extended Base, const int Exponent);
```

File

Math

Description

IntPower raises Base to the power specified by Exponent.

See Also

Power (see page 15965)

93.2.48 Math.IsInfinite Function

Indicates when a variable or expression represents an infinite value.

Pascal

```
function IsInfinite(const AValue: Double): Boolean;
```

C++

```
Boolean IsInfinite(const Double AValue);
```

File

Math

Description

Use IsInfinite to test whether the value specified by AValue represents positive or negative infinity. Use the Sign (see page 15973) function to distinguish positive infinity from negative infinity.

See Also

GetExceptionMask (see page 15947)

NegInfinity (see page 15993)

IsNan (see page 15954)

Infinity (see page 15988)

IsZero (see page 15954)

Sign (see page 15973)

93.2.49 Math.IsNan Function

Indicates when a variable or expression does not evaluate to a numeric value.

Pascal

```
function IsNan(const AValue: Double): Boolean; overload;
function IsNan(const AValue: Single): Boolean; overload;
function IsNan(const AValue: Extended): Boolean; overload;
```

C++

```
Boolean IsNan(const Double AValue);
Boolean IsNan(const Single AValue);
Boolean IsNan(const Extended AValue);
```

File

Math

Description

Use IsNan to test whether the value specified by AValue represents NaN (see page 15992) (not a number). Note that NaN (see page 15992) is distinct from positive and negative infinity, which can be detected using the IsInfinite (see page 15953) function.

See Also

GetExceptionMask (see page 15947)

Nan

IsInfinite (see page 15953)

93.2.50 Math.IsZero Function

Indicates when a floating-point variable or expression evaluates to zero, or very close to zero.

Pascal

```
function IsZero(const A: Extended; Epsilon: Extended = 0): Boolean; overload;
function IsZero(const A: Double; Epsilon: Double = 0): Boolean; overload;
function IsZero(const A: Single; Epsilon: Single = 0): Boolean; overload;
```

C++

```
Boolean IsZero(const Extended A, Extended Epsilon = 0);
Boolean IsZero(const Double A, Double Epsilon = 0);
Boolean IsZero(const Single A, Single Epsilon = 0);
```

File

Math

Description

Use IsZero to test whether the value specified by AValue is zero, or differs from zero by at most Epsilon.

See Also

SameValue (see page 15969)

IsInfinite (see page 15953)

Sign (see page 15973)

93.2.51 Math.Ldexp Function

Calculates X times (2 to the power of P).

Pascal

```
function Ldexp(const X: Extended; const P: Integer): Extended;
```

C++

```
Extended Ldexp(const Extended X, const int P);
```

File

Math

Description

Ldexp returns X times (2 to the power of P).

93.2.52 Math.LnXP1 Function

Returns the natural log of (X+1)

Pascal

```
function LnXP1(const X: Extended): Extended;
```

C++

```
Extended LnXP1(const Extended X);
```

File

Math

Description

LnXP1 returns the natural logarithm of (X+1). Use LnXP1 when X is a value near 0.

93.2.53 Math.Log10 Function

Calculates log base 10.

Pascal

```
function Log10(const X: Extended): Extended;
```

C++

```
Extended Log10(const Extended X);
```

File

Math

Description

Log10 returns the log base 10 of X.

See Also

Log2 (see page 15956)

Ln

LogN (see page 15956)

93.2.54 Math.Log2 Function

Calculates log base 2.

Pascal

```
function Log2(const X: Extended): Extended;
```

C++

```
Extended Log2(const Extended X);
```

File

Math

Description

Log2 returns the log base 2 of X.

See Also

Log10 (see page 15955)

Ln

LogN (see page 15956)

93.2.55 Math.LogN Function

Calculates the log of X for a specified base.

Pascal

```
function LogN(const Base: Extended; const X: Extended): Extended;
```

C++

```
Extended LogN(const Extended Base, const Extended X);
```

File

Math

Description

LogN returns the log base Base of X.

See Also

Log2 (see page 15956)

Log10 (see page 15955)

Ln

93.2.56 Math.Max Function

Returns the greater of two numeric values (Delphi only).

Pascal

```
function Max(const A: Integer; const B: Integer): Integer; overload;
function Max(const A: Int64; const B: Int64): Int64; overload;
function Max(const A: Single; const B: Single): Single; overload;
function Max(const A: Double; const B: Double): Double; overload;
function Max(const A: Extended; const B: Extended): Extended; overload;
```

C++

```
int Max(const int A, const int B);
Int64 Max(const Int64 A, const Int64 B);
Single Max(const Single A, const Single B);
Double Max(const Double A, const Double B);
Extended Max(const Extended A, const Extended B);
```

File

Math

Description

In Delphi, call Max to compare two numeric values. Max returns the greater value of the two.

See Also

MaxValue (see page 15958)

Min (see page 15959)

MaxIntValue (see page 15957)

93.2.57 Math.MaxIntValue Function

Returns the largest signed value in an integer array.

Pascal

```
function MaxIntValue(const Data: array of Integer): Integer;
```

C++

```
int MaxIntValue(const array of Integer Data);
```

File

Math

Description

Call MaxIntValue to obtain the largest integer from the Data array. In C++, Data_Size is the index of the last element in the array (one less than the number of elements).

See Also

MaxValue (see page 15958)

MinIntValue (see page 15960)

93.2.58 Math.MaxValue Function

Returns the largest signed value in an array.

Pascal

```
function MaxValue(const Data: array of Double): Double;
```

C++

```
Double MaxValue(const array of Double Data);
```

File

Math

Description

MaxValue returns the largest signed value in the Data array. In C++, the Data_Size parameter gives the index of the last element of the array (one less than the number of elements).

See Also

MaxIntValue (see page 15957)

MinValue (see page 15960)

93.2.59 Math.Mean Function

Returns the average of all values in an array.

Pascal

```
function Mean(const Data: array of Double): Extended;
```

C++

```
Extended Mean(const array of Double Data);
```

File

Math

Description

Mean calculates the arithmetic average of all the values in the Data array parameter. In C++, the Data_Size parameter gives the index of the last element of the array (one less than the number of elements).

See Also

MeanAndStdDev (see page 15959)

93.2.60 Math.MeanAndStdDev Function

Calculates the mean and standard deviation of array elements.

Pascal

```
procedure MeanAndStdDev(const Data: array of Double; var Mean: Extended; var StdDev: Extended);
```

C++

```
MeanAndStdDev(const array of Double Data, Extended Mean, Extended StdDev);
```

File

Math

Description

MeanAndStdDev calculates Mean (see page 15958) and standard deviation in one pass. Using this is twice as fast as calculating them separately. Precision may be lost when the Mean (see page 15958) is very large (> 10e7) or the variance is very small.

Data specifies the data to be summarized. In C++, Data_Size is the index of the last element in the Data array (one less than the number of elements). The mean is returned as Mean (see page 15958) and the standard deviation is returned as StdDev (see page 15975).

Note: If the sample size is 1 (that is, if Data contains only one entry), StdDev (see page 15975) returns the value of Data[0] and not zero.

See Also

Mean (see page 15958)

StdDev (see page 15975)

93.2.61 Math.Min Function

Returns the lesser of two numeric values (Delphi only).

Pascal

```
function Min(const A: Integer; const B: Integer): Integer; overload;
function Min(const A: Int64; const B: Int64): Int64; overload;
function Min(const A: Single; const B: Single): Single; overload;
function Min(const A: Double; const B: Double): Double; overload;
function Min(const A: Extended; const B: Extended): Extended; overload;
```

C++

```
int Min(const int A, const int B);
Int64 Min(const Int64 A, const Int64 B);
Single Min(const Single A, const Single B);
Double Min(const Double A, const Double B);
Extended Min(const Extended A, const Extended B);
```

File

Math

Description

Call Min to compare two numeric values in Delphi. Min returns the smaller value of the two.

See Also

MinValue (see page 15960)

Max (see page 15957)

MinIntValue (see page 15960)

93.2.62 Math.MinIntValue Function

Returns the smallest signed value in an integer array.

Pascal

```
function MinIntValue(const Data: array of Integer): Integer;
```

C++

```
int MinIntValue(const array of Integer Data);
```

File

Math

Description

Call MinIntValue to obtain the smallest integer in the Data array. In C++, the Data_Size parameter specifies the index of the last element in the array (one less than the total number of elements).

See Also

MaxIntValue (see page 15957)

MinValue (see page 15960)

93.2.63 Math.MinValue Function

Returns smallest signed value in an array.

Pascal

```
function MinValue(const Data: array of Double): Double;
```

C++

```
Double MinValue(const array of Double Data);
```

File

Math

Description

MinValue returns the smallest signed value in the Data array. The Data_Size parameter specifies the index of the last element in the array (one less than the total number of elements).

See Also

MaxValue (see page 15958)

MinIntValue (see page 15960)

93.2.64 Math.MomentSkewKurtosis Function

Calculates the mean, variance, skew, and kurtosis.

Pascal

```
procedure MomentSkewKurtosis(const Data: array of Double; var M1: Extended; var M2: Extended;
var M3: Extended; var M4: Extended; var Skew: Extended; var Kurtosis: Extended);
```

C++

```
MomentSkewKurtosis(const array of Double Data, Extended M1, Extended M2, Extended M3, Extended
M4, Extended Skew, Extended Kurtosis);
```

File

Math

Description

MomentSkewKurtosis calculates the core factors of statistical analysis: the first four moments plus the coefficients of skew and kurtosis.

Data specifies the data to be analyzed. In C++, Data_Size is the index of the last element in the Data array (one less than the number of elements). M1, M2, M3 and M4 return the first four moments (M1 is the mean, M2 is the variance, and so on). Skew reflects symmetry of distribution. Kurtosis reflects flatness of distribution.

93.2.65 Math.NetPresentValue Function

Calculates the current value from an array of estimated cash flow values.

Pascal

```
function NetPresentValue(const Rate: Extended; const CashFlows: array of Double; PaymentTime:
TPaymentTime): Extended;
```

C++

```
Extended NetPresentValue(const Extended Rate, const array of Double CashFlows, TPaymentTime
PaymentTime);
```

File

Math

Description

NetPresentValue calculates the current value of an array of estimated cash flow values, discounted at the given interest rate of Rate.

CashFlows is the array of estimated cash flow values. In C++, CashFlows_Size is the index of the last element in that array (one less than the number of elements).

PaymentTime indicates whether the cash flows occur at the beginning or end of the period.

NetPresentValue helps determine how much an investment is currently worth, based on expected earnings, although its accuracy depends on the accuracy of the cash flows in the array.

93.2.66 Math.Norm Function

Returns the Euclidean 'L-2' norm.

Pascal

```
function Norm(const Data: array of Double): Extended;
```

C++

```
Extended Norm(const array of Double Data);
```

File

Math

Description

Norm returns the Euclidean 'L-2' norm for the values in the Data array. The 'L-2' norm is the square root of the sum of squares.

Note: In C++, Data is an array of values, and Data_Size is the index of the last value in the array (one less than the number of values).

See Also

SumOfSquares (see page 15977)

93.2.67 Math.NumberOfPeriods Function

Returns the number of payment periods for an investment.

Pascal

```
function NumberOfPeriods(const Rate: Extended; Payment: Extended; const PresentValue:
Extended; const FutureValue: Extended; PaymentTime: TPaymentTime): Extended;
```

C++

```
Extended NumberOfPeriods(const Extended Rate, Extended Payment, const Extended PresentValue,
const Extended FutureValue, TPaymentTime PaymentTime);
```

File

Math

Description

NumberOfPeriods computes the number of payment periods required for an investment of PresentValue (see page 15965) to reach a value of FutureValue (see page 15946), while making regular payments of Payment (see page 15963) and accruing interest at the rate of Rate per compounding period. PaymentTime indicates whether the cash flows occur at the beginning or end of the period.

93.2.68 Math.Payment Function

Calculates a fully amortized payment.

Pascal

```
function Payment(Rate: Extended; NPeriods: Integer; const PresentValue: Extended; const
FutureValue: Extended; PaymentTime: TPaymentTime): Extended;
```

C++

```
Extended Payment(Extended Rate, int NPeriods, const Extended PresentValue, const Extended
FutureValue, TPaymentTime PaymentTime);
```

File

Math

Description

Payment calculates the fully amortized payment of borrowing PresentValue (see page 15965) at Rate percent per period over NPeriods. It assumes that interest is paid at the end of each period.

FutureValue (see page 15946) is the value that the investment will reach at some point. PaymentTime indicates whether the cash flows occur at the beginning or end of the period.

93.2.69 Math.PeriodPayment Function

Returns the principal amount from a full payment.

Pascal

```
function PeriodPayment(const Rate: Extended; Period: Integer; NPeriods: Integer; const
PresentValue: Extended; const FutureValue: Extended; PaymentTime: TPaymentTime): Extended;
```

C++

```
Extended PeriodPayment(const Extended Rate, int Period, int NPeriods, const Extended
PresentValue, const Extended FutureValue, TPaymentTime PaymentTime);
```

File

Math

Description

PeriodPayment gives the part of the payment that is principal. The InterestPayment (see page 15951) function gives the part of the payment that is interest.

The Rate parameter is the interest rate. Period is the number of periods into the loan for which the principal is desired, and NPeriods is the number of periods of the loan. FutureValue (see page 15946) is the value the investment will reach at some point. PaymentTime indicates whether the cash flows occur at the beginning or end of the period.

93.2.70 Math.Poly Function

Evaluates a uniform polynomial of one variable at the value X.

Pascal

```
function Poly(const X: Extended; const Coefficients: array of Double): Extended;
```

C++

```
Extended Poly(const Extended X, const array of Double Coefficients);
```

File

Math

Description

Call Poly to evaluate the polynomial represented by the Coefficients parameter at the point where the variable equals the value of the X parameter. The coefficients are ordered in increasing powers of X, as shown in the following Delphi-like example:

Coefficients[0] + Coefficients[1]*X + ... + Coefficients[N]*(X**N)

Note: In C++, the Coefficients_Size parameter is N, the highest power to which X is raised.

93.2.71 Math.PopnStdDev Function

Calculates the population standard deviation.

Pascal

```
function PopnStdDev(const Data: array of Double): Extended;
```

C++

```
Extended PopnStdDev(const array of Double Data);
```

File

Math

Description

PopnStdDev calculates the population standard deviation (the square root of the population variance) of all values in the Data array parameter. In C++, the Data_Size parameter indicates the index of the last element in Data (one less than the number of data points).

This function is used in some business and financial calculations.

See Also

StdDev (see page 15975)

PopnVariance (see page 15964)

93.2.72 Math.PopnVariance Function

Calculates the population variance.

Pascal

```
function PopnVariance(const Data: array of Double): Extended;
```

C++

```
Extended PopnVariance(const array of Double Data);
```

File

Math

Description

PopnVariance calculates the population variance of all values in the Data array parameter, using the n method (biased): TotalVariance (see page 15979) / n. In C++,. the Data_Size parameter is the index of the last value in the Data array (one less than the number of data items).

See Also

Variance (see page 15980)

TotalVariance (see page 15979)

PopnStdDev (see page 15964)

93.2.73 Math.Power Function

Raises Base to any power.

Pascal

```
function Power(const Base: Extended; const Exponent: Extended): Extended; overload;
function Power(const Base: Double; const Exponent: Double): Double; overload;
function Power(const Base: Single; const Exponent: Single): Single; overload;
```

C++

```
Extended Power(const Extended Base, const Extended Exponent);
Double Power(const Double Base, const Double Exponent);
Single Power(const Single Base, const Single Exponent);
```

File

Math

Description

Power raises Base to any power. For fractional exponents or exponents greater than MaxInt, Base must be greater than 0.

See Also

IntPower (see page 15953)

Sqrt

93.2.74 Math.PresentValue Function

Calculates the present value of an investment.

Pascal

```
function PresentValue(const Rate: Extended; NPeriods: Integer; const Payment: Extended; const
FutureValue: Extended; PaymentTime: TPaymentTime): Extended;
```

C++

```
Extended PresentValue(const Extended Rate, int NPeriods, const Extended Payment, const
Extended FutureValue, TPaymentTime PaymentTime);
```

File

Math

Description

PresentValue calculates the present value of an investment where Payment (see page 15963) is received for NPeriods and is discounted at the rate of Rate per period. FutureValue (see page 15946) is the value the investment may reach at some point. PaymentTime indicates whether the cash flows occur at the beginning or end of the period.

93.2.75 Math.RadToCycle Function

Converts radians to cycles.

Pascal

```
function RadToCycle(const Radians: Extended): Extended;
```

C++

```
Extended RadToCycle(const Extended Radians);
```

File

Math

Description

Use RadToCycle to convert angles measured in radians into cycles, where

cycles = radians/(2pi).

93.2.76 Math.RadToDeg Function

Converts radians to degrees.

Pascal

```
function RadToDeg(const Radians: Extended): Extended;
```

C++

```
Extended RadToDeg(const Extended Radians);
```

File

Math

Description

Use RadToDeg to convert angles measured in radians to degrees, where

degrees = radians(180/pi).

93.2.77 Math.RadToGrad Function

Converts radians to grads.

Pascal

```
function RadToGrad(const Radians: Extended): Extended;
```

C++

```
Extended RadToGrad(const Extended Radians);
```

File

Math

Description

Use RadToGrad to convert angles measured in radians to grads, where

grads = radians(200/pi).

93.2.78 Math.RandG Function

Generates random numbers with Gaussian distribution.

Pascal

```
function RandG(Mean: Extended; StdDev: Extended): Extended;
```

C++

```
Extended RandG(Extended Mean, Extended StdDev);
```

File

Math

Description

RandG produces random numbers with Gaussian distribution about the Mean (see page 15958). This is useful for simulating data with sampling errors and expected deviations from the Mean (see page 15958).

93.2.79 Math.RandomFrom Function

Returns a randomly selected element from an array.

Pascal

```
function RandomFrom(const AValues: array of Integer): Integer; overload;
function RandomFrom(const AValues: array of Int64): Int64; overload;
function RandomFrom(const AValues: array of Double): Double; overload;
```

C++

```
int RandomFrom(const array of Integer AValues);
Int64 RandomFrom(const array of Int64 AValues);
Double RandomFrom(const array of Double AValues);
```

File

Math

Description

RandomFrom returns a randomly selected element from the array passed as the AValues argument. In C++, AValues_Size is

the index of the last array element (one less than the number of elements.)

To initialize the random number generator, add a single call Randomize or assign a value to the RandSeed variable before making any calls to RandomFrom.

See Also

Randomize

RandSeed

93.2.80 Math.RandomRange Function

Returns a random integer from a specified range.

Pascal

```
function RandomRange(const AFrom: Integer; const ATo: Integer): Integer;
```

C++

```
int RandomRange(const int AFrom, const int ATo);
```

File

Math

Description

RandomRange returns a random integer from the range that extends between AFrom and ATo (non-inclusive). RandomRange can handle negative ranges (where AFrom is greater than ATo).

To initialize the random number generator, add a single call Randomize or assign a value to the RandSeed variable before making any calls to RandomRange.

See Also

Randomize

RandSeed

RandomFrom (see page 15967)

Random

93.2.81 Math.RoundTo Function

Rounds a floating-point value to a specified digit or power of ten using "Banker's rounding".

Pascal

```
function RoundTo(const AValue: Double; const ADigit: TRoundToRange): Double;
```

C++

```
Double RoundTo(const Double AValue, const TRoundToRange ADigit);
```

File

Math

Description

Call RoundTo to round AValue to a specified power of ten.

AValue is the value to round.

ADigit indicates the power of ten to which you want AValue rounded. It can be any value from –37 to 37 (inclusive).

RoundTo uses "Banker's Rounding" to determine how to round values that are exactly midway between the two values that have the desired number of significant digits. This method rounds to an even number in the case that AValue is not nearer to either value.

The following examples illustrate the use of RoundTo:

Expression	Value
RoundTo(1234567, 3)	1234000
RoundTo(1.234, -2)	1.23
RoundTo(1.235, -2)	1.24
RoundTo(1.245, -2)	1.24

Note: The behavior of RoundTo can be affected by the Set8087CW procedure or SetRoundMode (see page 15972) function.

See Also

Round

SetRoundMode (see page 15972)

Set8087CW

SimpleRoundTo (see page 15973)

93.2.82 Math.SameValue Function

Indicates whether two floating-point values are (approximately) equal.

Pascal

```
function SameValue(const A: Extended; const B: Extended; Epsilon: Extended = 0): Boolean;
overload;
function SameValue(const A: Double; const B: Double; Epsilon: Double = 0): Boolean; overload;
function SameValue(const A: Single; const B: Single; Epsilon: Single = 0): Boolean; overload;
```

C++

```
Boolean SameValue(const Extended A, const Extended B, Extended Epsilon = 0);
Boolean SameValue(const Double A, const Double B, Double Epsilon = 0);
Boolean SameValue(const Single A, const Single B, Single Epsilon = 0);
```

File

Math

Description

Call SameValue to determine when two floating-point values are equal, or approximately equal.

A and B are the values to compare.

Epsilon is the maximum amount by which A and B can differ and still be considered the same value.

See Also

CompareValue (see page 15938)

IsZero (see page 15954)

93.2.83 Math.Sec Function

Calculates the secant of an angle.

Pascal

```
function Sec(const X: Extended): Extended;
```

C++

```
Extended Sec(const Extended X);
```

File

Math

Description

Call Sec to obtain the secant of X, where X is an angle in radians. The secant is calculated using the formula 1 / Cos(X).

Note: Sec is identical to the Secant (see page 15970) function.

See Also

ArcSec (see page 15934)

Secant (see page 15970)

SecH (see page 15971)

Cos

93.2.84 Math.Secant Function

Calculates the secant of an angle.

Pascal

```
function Secant(const X: Extended): Extended;
```

C++

```
Extended Secant(const Extended X);
```

File

Math

Description

Call Secant to obtain the secant of X, where X is an angle in radians. The secant is calculated using the formula 1 / Cos(X).

See Also

Sec (see page 15970)

SecH (see page 15971)

Cos

93.2.85 Math.SecH Function

Calculates the hyperbolic secant of an angle.

Pascal

```
function SecH(const X: Extended): Extended;
```

C++

```
Extended SecH(const Extended X);
```

File

Math

Description

Call SecH to obtain the hyperbolic secant of X, where X is an angle in Radians.

See Also

Sec (see page 15970)

ArcSec (see page 15934)

93.2.86 Math.SetExceptionMask Function

Sets the exception mask on the FPU control word.

Pascal

```
function SetExceptionMask(const Mask: TFPUExceptionMask): TFPUExceptionMask;
```

C++

```
TFPUExceptionMask SetExceptionMask(const TFPUExceptionMask Mask);
```

File

Math

Description

Call SetExceptionMask to specify the types of exceptions the FPU (floating-point unit) generates. Any element set in this mask prevents the FPU from raising the corresponding type of exception. Instead, the FPU returns its best attempt at a value – often NaN (see page 15992), Infinity (see page 15988), or NegInfinity (see page 15993). The value depends on the operation and the current rounding mode.

SetExceptionMask returns the previous exception mask.

See Also

Set8087CW

GetExceptionMask (see page 15947)

NaN (see page 15992)

Infinity (see page 15988)

NegInfinity (see page 15993)

93.2.87 Math.SetPrecisionMode Function

Sets the FPU precision control mode.

Pascal

```
function SetPrecisionMode(const Precision: TFPUPrecisionMode): TFPUPrecisionMode;
```

C++

```
TFPUPrecisionMode SetPrecisionMode(const TFPUPrecisionMode Precision);
```

File

Math

Description

Call SetPrecisionMode to specify the level of precision that the FPU (floating-point unit) uses for floating-point calculations. The precision control mode can be any of the following values:

Value	Meaning
pmSingle	single precision
pmReserved	not used
pmDouble	double precision
pmExtended	extended precision

SetPrecisionMode returns the previous precision control mode.

See Also

Set8087CW

GetPrecisionMode (see page 15947)

93.2.88 Math.SetRoundMode Function

Sets the FPU rounding mode.

Pascal

```
function SetRoundMode(const RoundMode: TFPURoundingMode): TFPURoundingMode;
```

C++

```
TFPURoundingMode SetRoundMode(const TFPURoundingMode RoundMode);
```

File

Math

Description

Call SetRoundingMode to specify how the FPU handles rounding issues. The rounding mode can be any of the following values:

Value	Meaning
rmNearest	Rounds to the closest value.
rmDown	Rounds toward negative infinity.
rmUp	Rounds toward positive infinity.
rmTruncate	Truncates the value, rounding positive numbers down and negative numbers up.

See Also

Set8087CW

GetRoundMode (see page 15948)

93.2.89 Math.Sign Function

Indicates whether a numeric value is positive, negative, or zero.

Pascal

```
function Sign(const AValue: Integer): TValueSign; overload;
function Sign(const AValue: Int64): TValueSign; overload;
function Sign(const AValue: Double): TValueSign; overload;
```

C++

```
TValueSign Sign(const int AValue);
TValueSign Sign(const Int64 AValue);
TValueSign Sign(const Double AValue);
```

File

Math

Description

Use Sign to test the sign of a numeric value. Sign returns

0 if AValue is zero.

1 if AValue is greater than zero.

-1 if AValue is less than zero.

See Also

IsZero (see page 15954)

93.2.90 Math.SimpleRoundTo Function

Rounds a floating-point value to a specified digit or power of ten using asymmetric arithmetic rounding.

Pascal

```
function SimpleRoundTo(const AValue: Double; const ADigit: TRoundToRange = -2): Double;
```

C++

```
Double SimpleRoundTo(const Double AValue, const TRoundToRange ADigit = -2);
```

File

Math

Description

Call SimpleRoundTo to round AValue to a specified power of ten.

AValue is the value to round.

ADigit indicates the power of ten to which you want AValue rounded. It can be any value from –37 to 37 (inclusive).

SimpleRoundTo uses asymmetric arithmetic rounding to determine how to round values that are exactly midway between the two values that have the desired number of significant digits. This method always rounds to the larger value.

The following examples illustrate the use of SimpleRoundTo:

Expression	Value
SimpleRoundTo(1234567, 3)	1234000
SimpleRoundTo(1.234, -2)	1.23
SimpleRoundTo(1.235, -2)	1.24
SimpleRoundTo(-1.235, -2)	-1.23

See Also

Round

RoundTo (see page 15968)

93.2.91 Math.SinCos Function

Returns sine and cosine of an angle.

Pascal

```
procedure SinCos(const Theta: Extended; var Sin: Extended; var Cos: Extended);
```

C++

```
SinCos(const Extended Theta, Extended Sin, Extended Cos);
```

File

Math

Description

Call SinCos to obtain the sine and cosine of an angle expressed in radians. Theta specifies the angle. The sine and cosine are returned in the Sin and Cos parameters, respectively.

SinCos is twice as fast as calling Sin and Cos separately for the same angle.

93.2.92 Math.Sinh Function

Returns the hyperbolic sine of an angle.

Pascal

```
function Sinh(const X: Extended): Extended;
```

C++

```
Extended Sinh(const Extended X);
```

File

Math

Description

Sinh calculates the hyperbolic sine of X.

93.2.93 Math.SLNDepreciation Function

Returns the straight-line depreciation allowance of an asset.

Pascal

```
function SLNDepreciation(const Cost: Extended; const Salvage: Extended; Life: Integer):
Extended;
```

C++

```
Extended SLNDepreciation(const Extended Cost, const Extended Salvage, int Life);
```

File

Math

Description

SLNDepreciation calculates the straight-line depreciation allowance for an asset over one period of its life. The function divides the Cost minus the Salvage by the number of years of useful Life of the asset. Cost is the amount initially paid for the asset. Salvage is the value of the asset at the end of its useful life.

To compute accelerated depreciation (allowing higher depreciation values in the first years of the assets life), use the SYDDepreciation (see page 15978) function.

See Also

SYDDepreciation (see page 15978)

93.2.94 Math.StdDev Function

Returns the sample standard deviation for elements in an array.

Pascal

```
function StdDev(const Data: array of Double): Extended;
```

C++

```
Extended StdDev(const array of Double Data);
```

File

Math

Description

StdDev calculates the sample standard deviation (the square root of the sample variance) of all values in the Data array parameter. In C++, Data_Size indicates the index of the last element in the array (one less than the number of elements).

See Also

TotalVariance (see page 15979)

Variance (see page 15980)

93.2.95 Math.Sum Function

Returns the sum of the elements in an array.

Pascal

```
function Sum(const Data: array of Double): Extended;
```

C++

```
Extended Sum(const array of Double Data);
```

File

Math

Description

Sum returns the sum of all the values in the Data array parameter. In C++, the Data_Size parameter specifies the index of the last value in Data (one less than the number of values).

See Also

SumInt (see page 15976)

93.2.96 Math.SumInt Function

Returns the sum of the elements in an integer array.

Pascal

```
function SumInt(const Data: array of Integer): Integer;
```

C++

```
int SumInt(const array of Integer Data);
```

File

Math

Description

Sum (see page 15976) returns the sum of all the values in the Data array parameter. In C++, the Data_Size parameter specifies the index of the last value in Data (one less than the number of values).

See Also

Sum (see page 15976)

93.2.97 Math.SumOfSquares Function

Returns the sum of the squared values from a data array.

Pascal

```
function SumOfSquares(const Data: array of Double): Extended;
```

C++

```
Extended SumOfSquares(const array of Double Data);
```

File

Math

Description

SumOfSquares returns the sum of the squares of the Data values: Data[0]^2 + Data[1]^2 + Data[2]^2... In C++, the Data_Size parameter specifies the index of the last value in Data (one less than the number of values).

See Also

Sum (see page 15976)

SumsAndSquares (see page 15977)

93.2.98 Math.SumsAndSquares Function

Returns the sum of the values and the sum of the squared values in an array.

Pascal

```
procedure SumsAndSquares(const Data: array of Double; var Sum: Extended; var SumOfSquares:
Extended);
```

C++

```
SumsAndSquares(const array of Double Data, Extended Sum, Extended SumOfSquares);
```

File

Math

Description

SumsAndSquares returns the sum of the Data values in Sum (see page 15976) and the sum of the squares in SumOfSquares (see page 15977). The data to sum is specified in the Data array parameter. In C++, Data_Size indicates the index of the last value (one less than the number of values in Data).

This function is faster than calculating the sums separately.

See Also

Sum (see page 15976)

SumOfSquares (see page 15977)

TotalVariance (see page 15979)

93.2.99 Math.SYDDepreciation Function

Calculates depreciation for an asset.

Pascal

```
function SYDDepreciation(const Cost: Extended; const Salvage: Extended; Life: Integer; Period:
Integer): Extended;
```

C++

```
Extended SYDDepreciation(const Extended Cost, const Extended Salvage, int Life, int Period);
```

File

Math

Description

SYDDepreciation (for "sum-of-years-digits depreciation") calculates depreciation amounts for an asset using an accelerated depreciation method. This allows for higher depreciation in the earlier years of an asset's life.

Cost is the initial cost of the asset. Salvage is the value of the asset at the end of its life expectancy. Life is the length of the asset's life expectancy. Period is the period for which to calculate the depreciation.

See Also

SLNDepreciation (see page 15975)

93.2.100 Math.Tan Function

Returns the tangent of X.

Pascal

```
function Tan(const X: Extended): Extended;
```

C++

```
Extended Tan(const Extended X);
```

File

Math

Description

Tan returns the tangent of X. Tan(X) = Sin(X) / Cos(X).

See Also

Sin

Cos

ArcTan

93.2.101 Math.Tanh Function

Returns the hyperbolic tangent of X.

Pascal

```
function Tanh(const X: Extended): Extended;
```

C++

```
Extended Tanh(const Extended X);
```

File

Math

Description

Tanh calculates the hyperbolic tangent of X.

See Also

Cosh (see page 15939)

Sinh (see page 15974)

93.2.102 Math.TotalVariance Function

Returns the statistical variance from an array of values.

Pascal

```
function TotalVariance(const Data: array of Double): Extended;
```

C++

```
Extended TotalVariance(const array of Double Data);
```

File

Math

Description

TotalVariance returns the sum of the squares of the difference between each Data value and the mean of the data values.

Total Variance (see page 15980): SUM(i=1,N)[(X(i) - Mean (see page 15958))**2]

See Also

SumOfSquares (see page 15977)

Variance (see page 15980)

StdDev (see page 15975)

Mean (see page 15958)

93.2.103 Math.Variance Function

Calculates statistical sample variance from an array of data.

Pascal

```
function Variance(const Data: array of Double): Extended;
```

C++

```
Extended Variance(const array of Double Data);
```

File

Math

Description

Variance calculates the sample variance of all values in the Data array parameter, using the n-1 method (unbiased). In C++, Data_Size specifies the index of the last value in Data (on less than the number of data values).

See Also

TotalVariance (see page 15979)

StdDev (see page 15975)

Mean (see page 15958)

PopnVariance (see page 15964)

93.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TFPUException (see page 15981)	TFPUException defines TFPUExceptionMask (see page 15985) values.
	TFPUPrecisionMode (see page 15982)	TFPUPrecisionMode defines how the FPU handles precision.
	TFPURoundingMode (see page 15983)	TFPURoundingMode defines how the FPU handles rounding issues.
	TPaymentTime (see page 15984)	TPaymentTime is an enumerated type used as a parameter in financial functions.

93.3.1 Math.TFPUException Enumeration

TFPUException defines TFPUExceptionMask (see page 15985) values.

Pascal

```
TFPUException = (
  exInvalidOp,
  exDenormalized,
  exZeroDivide,
  exOverflow,
  exUnderflow,
  exPrecision
);
```

C++

```
enum TFPUException {
  exInvalidOp,
  exDenormalized,
  exZeroDivide,
  exOverflow,
  exUnderflow,
  exPrecision
};
```

File

Math

Description

The FPU (Floating Point Unit) handles floating point calculations. You can prevent exceptions from occuring by using SetExceptionMask (see page 15971) to mask some or all of them.

When an exception is masked, the FPU returns its best attempt at a value – often NaN (see page 15992), Infinity (see page 15988), or NegInfinity (see page 15993). The value depends on the operation and the current rounding mode (see GetRoundMode (see page 15948) and SetRoundMode (see page 15972)).

TFPUExceptionMask (see page 15985) can have none, one, some or all of the following TFPUException values set :

Value	Meaning
exInvalidOp	An invalid operation was attempted.
exDenormalized	A number was reduced in size smaller than can be stored as non-zero. It has been denormalized.

exZeroDivide	An attempt was made to divide by zero.
exOverflow	A number has exceeded the highest positive value supported.
exUnderflow	A number has exceeded the highest negative value supported.
exPrecision	A number has exceeded the number of digits of precision.

See Also

GetExceptionMask (see page 15947)

GetRoundMode (see page 15948)

SetExceptionMask (see page 15971)

SetRoundMode (see page 15972)

TFPUExceptionMask (see page 15985)

93.3.2 Math.TFPUPrecisionMode Enumeration

TFPUPrecisionMode defines how the FPU handles precision.

Pascal

```
TFPUPrecisionMode = (
  pmSingle,
  pmReserved,
  pmDouble,
  pmExtended
);
```

C++

```
enum TFPUPrecisionMode {
  pmSingle,
  pmReserved,
  pmDouble,
  pmExtended
};
```

File

Math

Description

The FPU (Floating Point Unit) handles floating point calculations. GetPrecisionMode (see page 15947) is used to read, and SetPrecisionMode (see page 15972) is used to set the precision control mode, with TFPUPrecisionMode defining this mode.

TFPUPrecisionMode can be one of the following values :

Value	Meaning
pmSingle	Single precision : 7 significant digits, exponent from -38 to +38.
pmReserved	Reserved for future use.
pmDouble	Double precision : 15 significant digits, exponent from -308 to +308.
pmExtended	Extended precision : 19 significant digits, exponent from -4932 to +4932.

See Also

GetPrecisionMode (see page 15947)

GetRoundMode (see page 15948)

SetPrecisionMode (see page 15972)

SetRoundMode (see page 15972)

93.3.3 Math.TFPURoundingMode Enumeration

TFPURoundingMode defines how the FPU handles rounding issues.

Pascal

```
TFPURoundingMode = (
  rmNearest,
  rmDown,
  rmUp,
  rmTruncate
);
```

C++

```
enum TFPURoundingMode {
  rmNearest,
  rmDown,
  rmUp,
  rmTruncate
};
```

File

Math

Description

The FPU (Floating Point Unit) handles floating point calculations. GetRoundMode (see page 15948) is used to read, and SetRoundMode (see page 15972) is used to set the rounding mode, with TFPURoundingMode defining this mode.

TFPURoundingMode can be one of the following values :

Value	Meaning
rmNearest	Rounds to the closest value.
rmDown	Rounds toward negative infinity.
rmUp	Rounds toward positive infinity.
rmTruncate	Truncates the value, rounding positive numbers down and negative numbers up.

See Also

GetPrecisionMode (see page 15947)

GetRoundMode (see page 15948)

SetPrecisionMode (see page 15972)

SetRoundMode (see page 15972)

93.3.4 Math.TPaymentTime Enumeration

TPaymentTime is an enumerated type used as a parameter in financial functions.

Pascal

```
TPaymentTime = (
  ptEndOfPeriod,
  ptStartOfPeriod
);
```

C++

```
enum TPaymentTime {
  ptEndOfPeriod,
  ptStartOfPeriod
};
```

File

Math

Description

TPaymentTime indicates when interest is calculated. The following table lists the possible values:

Value	Meaning
ptStartOfPeriod	Interest is calculated on the principle balance at the beginning of the payment period
ptEndOfPeriod	Interest is calculated at the end of the payment period

93.4 Types

The following table lists types in this documentation.

Types

Name	Description
TFPUExceptionMask (see page 15985)	TFPUExceptionMask defines a set of TFPUException (see page 15981) values.
TRoundToRange (see page 15986)	TRoundToRange defines the RoundTo (see page 15968) rounding limits.
TValueSign (see page 15986)	TValueSign defines the Sign (see page 15973) returned value.

93.4.1 Math.TFPUExceptionMask Type

TFPUExceptionMask defines a set of TFPUException (see page 15981) values.

Pascal

```
TFPUExceptionMask = set of TFPUException;
```

C++

```
TFPUException TFPUExceptionMask;
```

File

Math

Description

The FPU (Floating Point Unit) handles floating point calculations. You can prevent exceptions from occuring by using SetExceptionMask (see page 15971) to mask some or all of them.

When an exception is masked, the FPU returns its best attempt at a value – often NaN (see page 15992), Infinity (see page 15988), or NegInfinity (see page 15993). The value depends on the operation and the current rounding mode (see GetRoundMode (see page 15948) and SetRoundMode (see page 15972)).

TFPUExceptionMask can have none, one, some or all of the following TFPUException (see page 15981) values set :

Value	Meaning
exInvalidOp	An invalid operation was attempted.
exDenormalized	A number was reduced in size smaller than can be stored as non-zero. It has been denormalized.
exZeroDivide	An attempt was made to divide by zero.
exOverflow	A number has exceeded the highest positive value supported.
exUnderflow	A number has exceeded the highest negative value supported.
exPrecision	A number has exceeded the number of digits of precision.

See Also

GetExceptionMask (see page 15947)

GetRoundMode (see page 15948)

SetExceptionMask (see page 15971)

SetRoundMode (see page 15972)

TFPUException (see page 15981)

93.4.2 Math.TRoundToRange Type

TRoundToRange defines the RoundTo (see page 15968) rounding limits.

Pascal

```
TRoundToRange = -37..37;
```

C++

```
-37..37 TRoundToRange;
```

File

Math

Description

The RoundTo (see page 15968) routine rounds a floating-point value to a specified digit or power of ten using "Banker's rounding". A negative TRoundToRange value rounds to this number of decimal places. A positive value removes all decimal places, and zeroes this number of significant digits to the left of the decimal point. Banker's rounding always rounds down to an even value last digit when it has a choice.

See Also

RoundTo (see page 15968)

93.4.3 Math.TValueSign Type

TValueSign defines the Sign (see page 15973) returned value.

Pascal

```
TValueSign = -1..1;
```

C++

```
-1..1 TValueSign;
```

File

Math

Description

The Sign (see page 15973) routine determines whether a number is negative, zero or positive. It returns the result in a TValueSign value as follows :

Value	Associated constant	Meaning
-1	NegativeValue (see page 15992)	Negative number.
0	ZeroValue (see page 15994)	Zero.
+1	PositiveValue (see page 15993)	Positive number.

See Also

NegativeValue (see page 15992)

PositiveValue (see page 15993)

ZeroValue (see page 15994)

RoundTo (see page 15968)

93.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
Infinity (see page 15988)	Infinity represents positive infinity.
MaxComp (see page 15989)	MaxComp defines the maximum absolute value for a Comp variable.
MaxDouble (see page 15989)	MaxDouble defines the maximum absolute value for a Double variable.
MaxExtended (see page 15989)	MaxExtended defines the maximum absolute value for an Extended variable.
MaxSingle (see page 15990)	MaxSingle defines the maximum absolute value for a Single variable.
MinComp (see page 15990)	MinComp defines the smallest absolute value for a Comp variable.
MinDouble (see page 15991)	MinDouble defines the smallest absolute value for a Double variable.
MinExtended (see page 15991)	MinExtended defines the smallest absolute value for an Extended variable.
MinSingle (see page 15991)	MinSingle defines the smallest absolute value for a Single variable.
NaN (see page 15992)	NaN represents a value that cannot be represented as a valid number.
NegativeValue (see page 15992)	NegativeValue defines a negative number result from using Sign (see page 15973).
NegInfinity (see page 15993)	NegInfinity represents negative infinity.
PositiveValue (see page 15993)	PositiveValue defines a positive number result from using Sign (see page 15973).
ZeroValue (see page 15994)	ZeroValue defines a zero value number result from using Sign (see page 15973).

93.5.1 Math.Infinity Constant

Infinity represents positive infinity.

Pascal

```
Infinity = 1.0 / 0.0;
```

C++

```
Infinity = 1.0 / 0.0;
```

File

Math

Description

Infinity is literally defined as 1.0/0.0 : positive infinity.

Note: Only use Infinity in assignments : use IsInfinite (see page 15953).

See Also

IsInfinite (see page 15953)

NegInfinity (see page 15993)

93.5.2 Math.MaxComp Constant

MaxComp defines the maximum absolute value for a Comp variable.

Pascal

```
MaxComp = 9.223372036854775807e+18;
```

C++

```
MaxComp = 9.223372036854775807e+18;
```

File

Math

Description

MaxComp defines the maximum absolute value for a Comp variable.

See Also

MinComp (see page 15990)

93.5.3 Math.MaxDouble Constant

MaxDouble defines the maximum absolute value for a Double variable.

Pascal

```
MaxDouble = 1.7e+308;
```

C++

```
MaxDouble = 1.7e+308;
```

File

Math

Description

MaxDouble defines the maximum absolute value for a Double variable.

See Also

MinDouble (see page 15991)

93.5.4 Math.MaxExtended Constant

MaxExtended defines the maximum absolute value for an Extended variable.

Pascal

```
MaxExtended = 1.1e+4932;
```

C++

```
MaxExtended = 1.1e+4932;
```

File

Math

Description

MaxExtended defines the maximum absolute value for an Extended variable.

See Also

MinExtended (see page 15991)

93.5.5 Math.MaxSingle Constant

MaxSingle defines the maximum absolute value for a Single variable.

Pascal

```
MaxSingle = 3.4e+38;
```

C++

```
MaxSingle = 3.4e+38;
```

File

Math

Description

MaxSingle defines the maximum absolute value for a Single variable.

See Also

MinSingle (see page 15991)

93.5.6 Math.MinComp Constant

MinComp defines the smallest absolute value for a Comp variable.

Pascal

```
MinComp = -9.223372036854775807e+18;
```

C++

```
MinComp = -9.223372036854775807e+18;
```

File

Math

Description

MinComp defines the smallest absolute value for a Comp variable.

See Also

MaxComp (see page 15989)

93.5.7 Math.MinDouble Constant

MinDouble defines the smallest absolute value for a Double variable.

Pascal

```
MinDouble = 5.0e-324;
```

C++

```
MinDouble = 5.0e-324;
```

File

Math

Description

MinDouble defines the smallest absolute value for a Double variable.

See Also

MaxDouble (see page 15989)

93.5.8 Math.MinExtended Constant

MinExtended defines the smallest absolute value for an Extended variable.

Pascal

```
MinExtended = 3.4e-4932;
```

C++

```
MinExtended = 3.4e-4932;
```

File

Math

Description

MinExtended defines the smallest absolute value for an Extended variable.

See Also

MaxExtended (see page 15989)

93.5.9 Math.MinSingle Constant

MinSingle defines the smallest absolute value for a Single variable.

Pascal

```
MinSingle = 1.5e-45;
```

C++

```
MinSingle = 1.5e-45;
```

File

Math

Description

MinSingle defines the smallest absolute value for a Single variable.

See Also

MaxSingle (see page 15990)

93.5.10 Math.NaN Constant

NaN represents a value that cannot be represented as a valid number.

Pascal

```
NaN = 0.0 / 0.0;
```

C++

```
NaN = 0.0 / 0.0;
```

File

Math

Description

NaN is literally defined as 0.0/0.0 : an undefined value.

Note: Only use NaN in assignments : use IsNaN.

See Also

IsNaN

93.5.11 Math.NegativeValue Constant

NegativeValue defines a negative number result from using Sign (see page 15973).

Pascal

```
NegativeValue = Low(TValueSign);
```

C++

```
NegativeValue = Low(TValueSign);
```

File

Math

Description

Sign (see page 15973) determines the sign of a number, returning the result in a TValueSign (see page 15986) value. TValueSign (see page 15986) will be NegativeValue if the number is negative.

See Also

PositiveValue (see page 15993)

Sign (see page 15973)

TValueSign (see page 15986)

ZeroValue (see page 15994)

93.5.12 Math.NegInfinity Constant

NegInfinity represents negative infinity.

Pascal

```
NegInfinity = -1.0 / 0.0;
```

C++

```
NegInfinity = -1.0 / 0.0;
```

File

Math

Description

NegInfinity is literally defined as -1.0/0.0 : negative infinity.

Note: Only use NegInfinity in assignments : use IsInfinite (see page 15953).

See Also

Infinity (see page 15988)

IsInfinity

IsInfinite (see page 15953)

93.5.13 Math.PositiveValue Constant

PositiveValue defines a positive number result from using Sign (see page 15973).

Pascal

```
PositiveValue = High(TValueSign);
```

C++

```
PositiveValue = High(TValueSign);
```

File

Math

Description

Sign (see page 15973) determines the sign of a number, returning the result in a TValueSign (see page 15986) value. TValueSign (see page 15986) will be PositiveValue if the number is positive.

See Also

NegativeValue (see page 15992)

Sign (see page 15973)

TValueSign (see page 15986)

ZeroValue (see page 15994)

93.5.14 Math.ZeroValue Constant

ZeroValue defines a zero value number result from using Sign (see page 15973).

Pascal

```
ZeroValue = 0;
```

C++

```
ZeroValue = 0;
```

File

Math

Description

Sign (see page 15973) determines the sign of a number, returning the result in a TValueSign (see page 15986) value. TValueSign (see page 15986) will be ZeroValue if the number is zero.

See Also

NegativeValue (see page 15992)

PositiveValue (see page 15993)

Sign (see page 15973)

TValueSign (see page 15986)

94 MConnect Namespace

This is namespace MConnect.

Topics

Name	Description
Classes (see page 15996)	The following table lists classes in this documentation.
Functions (see page 16051)	The following table lists functions in this documentation.

Classes

	Name	Description
	TCOMConnection (see page 15996)	TCOMConnection is the base class for components that connect to application servers using DCOM or OLE.
	TCustomObjectBroker (see page 16002)	TCustomObjectBroker is the base class for components that locate available application servers for a DataSnap connection component.
	TDCOMConnection (see page 16008)	TDCOMConnection handles a client application's DCOM connection to the remote server in a in a multi-tiered application.
	TDispatchAppServer (see page 16015)	This is class MConnect.TDispatchAppServer.
	TDispatchConnection (see page 16017)	TDispatchConnection is the base class for objects that connect client applications to an IDispatch-based application server.
	TOLEnterpriseConnection (see page 16033)	TOLEnterpriseConnection is provided for backward compatibility with applications that connect to an application server using OLEnterprise.
	TSharedConnection (see page 16039)	TSharedConnection connects to a child remote data module that is accessible only through the properties of a main remote data module on the application server.

Functions

	Name	Description
	GetMIDASAppServerList (see page 16051)	This is function MConnect.GetMIDASAppServerList.

94.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCOMConnection (see page 15996)	TCOMConnection is the base class for components that connect to application servers using DCOM or OLE.
	TCustomObjectBroker (see page 16002)	TCustomObjectBroker is the base class for components that locate available application servers for a DataSnap connection component.
	TDCOMConnection (see page 16008)	TDCOMConnection handles a client application's DCOM connection to the remote server in a in a multi-tiered application.
	TDispatchAppServer (see page 16015)	This is class MConnect.TDispatchAppServer.
	TDispatchConnection (see page 16017)	TDispatchConnection is the base class for objects that connect client applications to an IDispatch-based application server.
	TOLEnterpriseConnection (see page 16033)	TOLEnterpriseConnection is provided for backward compatibility with applications that connect to an application server using OLEnterprise.
	TSharedConnection (see page 16039)	TSharedConnection connects to a child remote data module that is accessible only through the properties of a main remote data module on the application server.

94.1.1 TCOMConnection Class

TCOMConnection is the base class for components that connect to application servers using DCOM or OLE.

Class Hierarchy

Pascal

```
TCOMConnection = class(TDispatchConnection);
```

C++

```
class TCOMConnection : public TDispatchConnection;
```

File

MConnect

Description

Do not create instances of TCOMConnection. Instead, use TCOMConnection as a base class when created a custom component that forms a COM-based connection to an application server.

TCOMConnection adds to the ability to work with COM objects to the generic connection capabilities inherited from TDispatchConnection (see page 16017). It has two descendants: TDCOMConnection (see page 16008), which creates and manages connections using DCOM, and TOLEnterpriseConnection (see page 16033), which is provided for backward compatibility and handles connections using OLEnterprise.

See Also

TClientDataSet (see page 8289)

TDCOMConnection (see page 16008)

TOLEnterpriseConnection (see page 16033)

TDataSetProvider (see page 18508)

IAppServer (see page 16450)

Writing Applications Using COM

94.1.1.1 TCOMConnection Members

The following tables list the members exposed by TCOMConnection.

Protected Events

	Name	Description
	OnGetUsername (see page 8533)	This is OnGetUsername, a member of class TCustomRemoteServer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoConnect (see page 16001)	Connects to an application server.
V	DoDisconnect (see page 6864)	Provides the interface for a method that terminates the connection.
V	DoDisconnect (see page 16023)	Drops a connection to an application server.
V	GetAppServer (see page 8530)	Returns the value of the AppServer (see page 8532) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetConnected (see page 6865)	Returns the value of the Connected (see page 6868) property.
V	GetConnected (see page 16023)	Returns the value of the Connected (see page 16026) property.
V	GetDataSet (see page 6865)	Returns a dataset, given its index in the DataSets (see page 6870) property array.
V	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetServerCLSID (see page 16024)	Returns the GUID for the application server.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.

	Loaded (see page 6866)	Initializes the connection component after all objects in the form or data module have been loaded.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 16025)	Responds when other objects are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RegisterClient (see page 6866)	Registers an object that uses the connection component.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppServer (see page 8531)	Sets the AppServer (see page 8532) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnected (see page 16001)	Sets the Connected (see page 16026) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	UnRegisterClient (see page 6868)	Removes a registered client so that it is no longer associated with the connection component.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ObjectBroker (see page 16028)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.

Public Events

	Name	Description
	AfterConnect (see page 6874)	Occurs after a connection is established.
	AfterDisconnect (see page 6875)	Occurs after the connection closes.
	BeforeConnect (see page 6875)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 6875)	Occurs immediately before the connection closes.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 6863)	Closes the connection.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 6862)	Creates an instance of TCustomConnection (see page 6858).
V	Create (see page 8529)	Creates an instance of a remote server component.
V	Create (see page 16022)	Creates an instance of a dispatch connection component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 6863)	Destroys the instance of a connection component.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetProviderNames (see page 8530)	Executes a callback for all provider objects on the application server.
V	GetServer (see page 8530)	Returns the IAppServer interface of the application server.
V	GetServer (see page 16023)	Returns the IAppServer interface of the application server.
V	GetServerList (see page 8531)	Returns a list of application servers (remote data modules) to which the component can connect.
V	GetServerList (see page 16024)	Returns a list of remote data modules to which the component can connect.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 6866)	Opens the connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AppServer (see page 8532)	Contains the interface for communicating with the application server.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Connected (see page 6868)	Determines whether a connection has been established to the remote source of data.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 6870)	Lists the datasets currently using this connection component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	LoginPrompt (see page 6872)	Specifies whether a login dialog appears immediately before opening a new connection.
	OnLogin (see page 6873)	Occurs when a communication channel opens to the server.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterConnect (see page 16031)	Occurs after a connection is established.
	AfterDisconnect (see page 16031)	Occurs after the connection closes.
	BeforeConnect (see page 16032)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 16032)	Occurs immediately before the connection closes.
	OnGetUsername (see page 16033)	This is OnGetUsername, a member of class TDispatchConnection.

Published Properties

	Name	Description
	Connected (see page 16026)	Indicates whether the TDispatchConnection (see page 16017) object is connected to an application server.
	LoginPrompt (see page 16027)	Specifies whether a login dialog appears immediately before opening a new connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OnLogin (see page 16028)	Occurs when a communication channel opens to the server.
	ServerGUID (see page 16030)	Specifies the GUID of the application server with which to connect.
	ServerName (see page 16030)	Specifies the name of the application server with which to connect.
	Tag (see page 2808)	Stores an integer value as part of a component.

94.1.1.2 TCOMConnection Methods

The methods of the TCOMConnection class are listed here.

Protected Methods

	Name	Description
	DoConnect (see page 16001)	Connects to an application server.
	SetConnected (see page 16001)	Sets the Connected (see page 16026) property.

94.1.1.2.1 TCOMConnection.DoConnect Method

Connects to an application server.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

DoConnect is called internally when the Connected (see page 16026) property is set to true. DoConnect creates a COM object for the application server (based on the ServerName (see page 16030) or ServerGUID (see page 16030) property), and sets the value of the AppServer property to this COM object.

Descendants of TCOMConnection (see page 15996) override DoConnect to change the way the client application forms a connection to an application server.

See Also

AfterConnect (see page 6874)

BeforeConnect (see page 6875)

Connected

AppServer (see page 8532)

ServerName (see page 16030)

ServerGUID (see page 16030)

94.1.1.2.2 TCOMConnection.SetConnected Method

Sets the Connected (see page 16026) property.

Pascal

```
procedure SetConnected(Value: Boolean); override;
```

C++

```
virtual __fastcall SetConnected(Boolean Value);
```

Description

SetConnected is the protected implementation of the Connected (see page 16026) property.

If Value is false, SetConnected generates a BeforeDisconnect (see page 16032) event, closes the connection, and generates an AfterDisconnect (see page 16031) event.

If Value is true, SetConnected performs the following actions:

1. It checks the value of the ServerGUID (see page 16030) or ServerName (see page 16030) property and raises an exception if these do not specify a valid application server.
2. If LoginPrompt (see page 16027) is true, and the connection is not already open, SetConnected generates an OnGetUsername (see page 16033) event, and uses that to initialize a login dialog. It displays the login dialog, and aborts the connection if the user cancels.
3. It generates a BeforeConnect (see page 16032) event.
4. It opens the connection.
5. It generates an AfterConnect (see page 16031) event.
6. If LoginPrompt (see page 16027) is true, and the connection opened successfully, SetConnected generates an OnLogin (see page 16028) event.

See Also

Connected

ServerGUID (see page 16030)

ServerName (see page 16030)

BeforeConnect (see page 6875)

AfterConnect (see page 6874)

BeforeDisconnect (see page 6875)

AfterDisconnect (see page 6875)

LoginPrompt (see page 6872)

OnGetUserName (see page 8533)

OnLogin (see page 6873)

94.1.2 TCustomObjectBroker Class

TCustomObjectBroker is the base class for components that locate available application servers for a DataSnap connection component.

Class Hierarchy

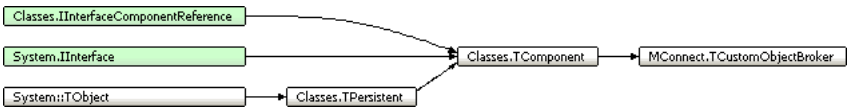

Pascal

```
TCustomObjectBroker = class(TComponent);
```

C++

```
class TCustomObjectBroker : public TComponent;
```

File

MConnect

Description

TCustomObjectBroker defines the methods by which a connection component obtains information about available application servers. When a connection component uses an object broker, it does not save any properties that identify a particular server machine. Instead, when it needs to open a connection, it uses the methods of the object broker to get that identifying information. This process allows client applications to locate server systems dynamically at runtime. If there are redundant server systems, this allows client applications to continue functioning when a server goes down.

TCustomObjectBroker does not implement any of the methods that provide information to connection components. Instead, descendants such as TSimpleObjectBroker implement a mechanism for choosing servers and identifying them to connection components. Application developers can derive more elaborate object brokers that provide additional benefits such as intelligent load balancing.

See Also

TDispatchConnection (see page 16017)

TSimpleObjectBroker (see page 17877)

94.1.2.1 TCustomObjectBroker Members

The following tables list the members exposed by TCustomObjectBroker.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.

	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.

A	GetComputerForGUID (see page 16006)	Returns a string that identifies a server machine for an interface identified by its GUID.
A	GetComputerForProgID (see page 16006)	Returns the machine name, URL, host name, or IP address of a server machine that supports an interface identified by its class name.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
A	GetPortForComputer (see page 16007)	Returns a port number for connecting to a specified remote server.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
A	SetConnectStatus (see page 16008)	Records whether a connection was established to a specified server.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

94.1.2.2 TCustomObjectBroker Methods

The methods of the TCustomObjectBroker class are listed here.

Public Methods

	Name	Description
	GetComputerForGUID (see page 16006)	Returns a string that identifies a server machine for an interface identified by its GUID.
	GetComputerForProgID (see page 16006)	Returns the machine name, URL, host name, or IP address of a server machine that supports an interface identified by its class name.
	GetPortForComputer (see page 16007)	Returns a port number for connecting to a specified remote server.
	SetConnectStatus (see page 16008)	Records whether a connection was established to a specified server.

94.1.2.2.1 TCustomObjectBroker.GetComputerForGUID Method

Returns a string that identifies a server machine for an interface identified by its GUID.

Pascal

```
function GetComputerForGUID(GUID: TGUID): string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetComputerForGUID(TGUID GUID) = 0;
```

Description

Call GetComputerForGUID to obtain a string that identifies a server machine on which a given interface is registered. The GUID parameter specifies the global unique identifier of the desired interface.

TCustomObjectBroker (see page 16002) declares GetComputerForGUID as an abstract (pure virtual) method. It is implemented in descendant objects that return a string designed to work with a particular connection component.

Connection components call this method, passing in the value of their ServerGUID property. They use the return value to set the ComputerName, URL, Host, or Address property, as appropriate.

See Also

GetComputerForProgID (see page 16006)

ServerGUID (see page 16030)

computername (see page 16014)

Address (see page 18935)

Host (see page 18936)

URL (see page 19878)

94.1.2.2.2 TCustomObjectBroker.GetComputerForProgID Method

Returns the machine name, URL, host name, or IP address of a server machine that supports an interface identified by its class name.

Pascal

```
function GetComputerForProgID(const ProgID): string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetComputerForProgID(const  ProgID) = 0;
```

Description

Call GetComputerForProgID to obtain the name or address of a server machine on which a given interface is registered. The ProgID parameter specifies the name of the desired interface. If the server was created using one of the remote data module wizards, this is the value that was entered as the class name for the wizard.

TCustomObjectBroker (see page 16002) declares GetComputerForProgID as an abstract (pure virtual) method. It is implemented in descendant objects that return a string designed to work with a particular connection component.

Connection components can call this method, passing in the value of their ServerName property. They can use the return value to set the ComputerName, URL, Host, or Address property, as appropriate.

See Also

GetComputerForGUID (see page 16006)

ServerName (see page 16030)

computername (see page 16014)

Address (see page 18935)

Host (see page 18936)

URL (see page 19878)

94.1.2.2.3 TCustomObjectBroker.GetPortForComputer Method

Returns a port number for connecting to a specified remote server.

Pascal

```
function GetPortForComputer(const ComputerName: string): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetPortForComputer(const AnsiString ComputerName) = 0;
```

Description

Socket connection components call GetPortForComputer to get a value for their Port property. The ComputerName parameter specifies a Host name or IP address that was returned by GetComputerForGUID (see page 16006) or GetComputerForProgID (see page 16006).

None of the connection components that work on Linux explicitly use a Port number. GetPortForComputer is included in TCustomObjectBroker (see page 16002) to allow for the possibility of connection components that use an explicit Port number. The ComputerName parameter is a Host name or IP address that is returned by the GetComputerForGUID (see page 16006) or GetComputerForProgID (see page 16006) method.

TCustomObjectBroker (see page 16002) declares GetPortForComputer as an abstract (pure virtual) method. It is implemented in descendant objects that return a string designed to work with a particular connection component.

Note: This interface assumes that a single socket dispatcher on each server system handles all client requests, even when they are destined for different application servers on the same host.

See Also

GetComputerForProgID (see page 16006)

GetComputerForGUID (see page 16006)

Port (see page 18938)

94.1.2.2.4 TCustomObjectBroker.SetConnectStatus Method

Records whether a connection was established to a specified server.

Pascal

```
procedure SetConnectStatus(ComputerName: string; Success: Boolean); virtual; abstract;
```

C++

```
virtual __fastcall SetConnectStatus(AnsiString ComputerName, Boolean Success) = 0;
```

Description

After calling GetComputerForGUID (see page 16006) or GetComputerForProgID (see page 16006), connection components use the returned value to open a connection to the identified server. They then call SetConnectStatus to tell the broker whether the connection was successfully opened.

TCustomObjectBroker (see page 16002) declares SetConnectStatus as an abstract (pure virtual) method. Descendants override this method to keep track of when server machines are down or to monitor the number of connections created for each server.

See Also

GetComputerForGUID (see page 16006)

GetComputerForProgID (see page 16006)

94.1.3 TDCOMConnection Class

TDCOMConnection handles a client application's DCOM connection to the remote server in a in a multi-tiered application.

Class Hierarchy

Pascal

```
TDCOMConnection = class(TCOMConnection);
```

C++

```
class TDCOMConnection : public TCOMConnection;
```

File

MConnect

Description

TDCOMConnection is used in the client portion of a multi-tiered database application to establish and maintain the connection between the client and a remote application server. The DCOM connection object can

Establish an initial connection to a remote application server.

Obtain an IAppServer for the application server.

Obtain a list of providers on the application server.

Drop the connection to the remote application server.

TDCOMConnection uses DCOM to establish the initial connection between the client application and a remote application server. In order to do so, the application server must be registered on the client machine.

Once the connection is established, the client registers any or all of its client datasets with TDCOMConnection, and these client datasets use the IAppServer interface from the DCOM connection component to communicate with providers on the application server, or to otherwise call the application server's data module interface.

To connect client applications to remote application servers using TCP/IP sockets, use TSocketConnection instead. To connect to remote application servers using HTTP, use the TWebConnection component. To connect to application servers using SOAP, use TSoapConnection.

See Also

TClientDataSet (see page 8289)

TSocketConnection (see page 18928)

TRemoteDataModule (see page 6585)

TDataSetProvider (see page 18508)

IAppServer (see page 16450)

TWebConnection (see page 18968)

TSoapConnection (see page 19864)

Specifying a Connection Using DCOM

94.1.3.1 TDCOMConnection Members

The following tables list the members exposed by TDCOMConnection.

Protected Events

	Name	Description
	OnGetUsername (see page 8533)	This is OnGetUsername, a member of class TCustomRemoteServer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoConnect (see page 16013)	Connects to an application server.
V	DoDisconnect (see page 6864)	Provides the interface for a method that terminates the connection.
V	DoDisconnect (see page 16023)	Drops a connection to an application server.
V	GetAppServer (see page 8530)	Returns the value of the AppServer (see page 8532) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.

	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetConnected (see page 6865)	Returns the value of the Connected (see page 6868) property.
	GetConnected (see page 16023)	Returns the value of the Connected (see page 16026) property.
	GetDataSet (see page 6865)	Returns a dataset, given its index in the DataSets (see page 6870) property array.
	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetServerCLSID (see page 16024)	Returns the GUID for the application server.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6866)	Initializes the connection component after all objects in the form or data module have been loaded.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 16025)	Responds when other objects are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RegisterClient (see page 6866)	Registers an object that uses the connection component.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppServer (see page 8531)	Sets the AppServer (see page 8532) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnected (see page 6867)	Changes the value of the Connected (see page 6868) property.
	SetConnected (see page 16001)	Sets the Connected (see page 16026) property.
	SetConnected (see page 16025)	Sets the Connected (see page 16026) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	UnRegisterClient (see page 6868)	Removes a registered client so that it is no longer associated with the connection component.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.

Public Events

	Name	Description
	AfterConnect (see page 6874)	Occurs after a connection is established.
	AfterDisconnect (see page 6875)	Occurs after the connection closes.
	BeforeConnect (see page 6875)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 6875)	Occurs immediately before the connection closes.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 6863)	Closes the connection.
V	Create (see page 16013)	Creates and initializes a DCOM connection component.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 6863)	Destroys the instance of a connection component.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetProviderNames (see page 8530)	Executes a callback for all provider objects on the application server.
V	GetServer (see page 8530)	Returns the IAppServer interface of the application server.
V	GetServer (see page 16023)	Returns the IAppServer interface of the application server.

V	GetServerList (see page 8531)	Returns a list of application servers (remote data modules) to which the component can connect.
V	GetServerList (see page 16024)	Returns a list of remote data modules to which the component can connect.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 6866)	Opens the connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AppServer (see page 8532)	Contains the interface for communicating with the application server.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Connected (see page 6868)	Determines whether a connection has been established to the remote source of data.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 6870)	Lists the datasets currently using this connection component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	LoginPrompt (see page 6872)	Specifies whether a login dialog appears immediately before opening a new connection.
	OnLogin (see page 6873)	Occurs when a communication channel opens to the server.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterConnect (see page 16031)	Occurs after a connection is established.
	AfterDisconnect (see page 16031)	Occurs after the connection closes.
	BeforeConnect (see page 16032)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 16032)	Occurs immediately before the connection closes.
	OnGetUsername (see page 16033)	This is OnGetUsername, a member of class TDispatchConnection.

Published Properties

	Name	Description
	ComputerName (see page 16014)	This is ComputerName, a member of class TDCOMConnection.
	Connected (see page 16026)	Indicates whether the TDispatchConnection (see page 16017) object is connected to an application server.
	LoginPrompt (see page 16027)	Specifies whether a login dialog appears immediately before opening a new connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectBroker (see page 16014)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	OnLogin (see page 16028)	Occurs when a communication channel opens to the server.
	ServerGUID (see page 16030)	Specifies the GUID of the application server with which to connect.
	ServerName (see page 16030)	Specifies the name of the application server with which to connect.
	Tag (see page 2808)	Stores an integer value as part of a component.

94.1.3.2 TDCOMConnection.Create Constructor

Creates and initializes a DCOM connection component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDCOMConnection(TComponent * AOwner);
```

Description

Call Create to instantiate TDCOMConnection (see page 16008) at runtime. DCOM connection components placed in forms or data modules at design time are created automatically.

The AOwner parameter specifies the component (usually the form or data module) that is responsible for freeing the DCOM connection component.

94.1.3.3 TDCOMConnection Methods

The methods of the TDCOMConnection class are listed here.

Protected Methods

	Name	Description
	DoConnect (see page 16013)	Connects to an application server.

94.1.3.3.1 TDCOMConnection.DoConnect Method

Connects to an application server.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

DoConnect is called when the Connected (see page 16026) property is set to true. DoConnect establishes a dual interface with the remote data module on the application server. This interface is used to locate providers on the application server.

To locate the server system, DoConnect uses the ComputerName (see page 16014) property. If the ObjectBroker (see page 16014) property is set, DoConnect gets this value from the object broker. Otherwise, the application must assign this property a value directly.

See Also

AfterConnect (see page 6874)

BeforeConnect (see page 6875)

Connected

AppServer (see page 8532)

computername (see page 16014)

ObjectBroker (see page 16028)

94.1.3.4 TDCOMConnection Properties

The properties of the TDCOMConnection class are listed here.

Published Properties

	Name	Description
	ComputerName (see page 16014)	This is ComputerName, a member of class TDCOMConnection.
	ObjectBroker (see page 16014)	Specifies an Object Broker that allows the connection component to choose from several application servers.

94.1.3.4.1 TDCOMConnection.ComputerName Property

Pascal

```
property ComputerName: string;
```

C++

```
__property AnsiString ComputerName;
```

Description

This is ComputerName, a member of class TDCOMConnection.

94.1.3.4.2 TDCOMConnection.ObjectBroker Property

Specifies an Object Broker that allows the connection component to choose from several application servers.

Pascal

```
property ObjectBroker: TCustomObjectBroker;
```

C++

```
__property TCustomObjectBroker ObjectBroker;
```

Description

Set ObjectBroker to use a broker that maintains a list of servers that support the application specified by ServerName or ServerGUID. When the connection component needs to open a connection, it requests information to identify a server machine from the Object Broker.

Descendants of TDispatchConnection define properties that specify how to locate server machines. These properties are not saved with the connection component if the ObjectBroker property is set. Instead, the Object Broker provides this information

dynamically at runtime from its list of available servers.

Applications can use TSimpleObjectBroker for an object broker, or derive their own object brokers from TCustomObjectBroker.

See Also

DoConnect (see page 6864)

ServerName (see page 16030)

ServerGUID (see page 16030)

TSimpleObjectBroker (see page 17877)

Brokering Connections

Specifying a Connection Using HTTP

Specifying a Connection Using DCOM

Specifying a Connection Using Sockets

Using TSimpleObjectBroker

94.1.4 TDispatchAppServer Class

Class Hierarchy

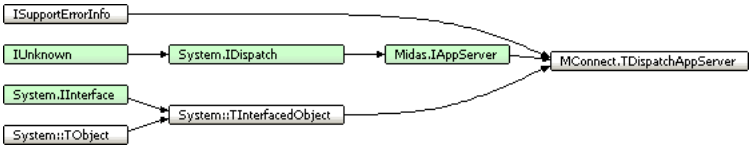

Pascal

```
TDispatchAppServer = class(TInterfacedObject, IAppServer, ISupportErrorInfo);
```

C++

```
class TDispatchAppServer : public TInterfacedObject, public IAppServer, public
ISupportErrorInfo;
```

File

MConnect

Description

This is class MConnect.TDispatchAppServer.

94.1.4.1 TDispatchAppServer Members

The following tables list the members exposed by TDispatchAppServer.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AS_ApplyUpdates (see page 16451)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 16452)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 16453)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 16454)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 16454)	Returns a list of all the available providers on a remote data module.
	AS_GetRecords (see page 16455)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 16456)	Returns information from a specified record of the provider's dataset.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 16017)	This is Create, a member of class TDispatchAppServer.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.

	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 16017)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

94.1.4.2 TDispatchAppServer.Create Constructor

Pascal

```
constructor Create(const AppServer: IAppServerDisp);
```

C++

```
__fastcall TDispatchAppServer(const IAppServerDisp AppServer);
```

Description

This is Create, a member of class TDispatchAppServer.

94.1.4.3 TDispatchAppServer Methods

The methods of the TDispatchAppServer class are listed here.

Public Methods

	Name	Description
	SafeCallException (see page 16017)	Handles exceptions in methods declared using the safecall calling convention.

94.1.4.3.1 TDispatchAppServer.SafeCallException Method

Handles exceptions in methods declared using the safecall calling convention.

Pascal

```
function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): HResult; override;
```

C++

```
virtual __fastcall HRESULT SafeCallException(TObject * ExceptObject, void * ExceptAddr);
```

Description

System::TObject::SafeCallException handles exceptions in methods that use the safecall calling convention. Some classes that implement interfaces override this method to handle errors that might occur.

As implemented in System::TObject::TObject, System::TObject::SafeCallException simply returns E_UNEXPECTED. This is the appropriate response for classes that do no support interfaces

94.1.5 TDispatchConnection Class

TDispatchConnection is the base class for objects that connect client applications to an IDispatch-based application server.

Class Hierarchy

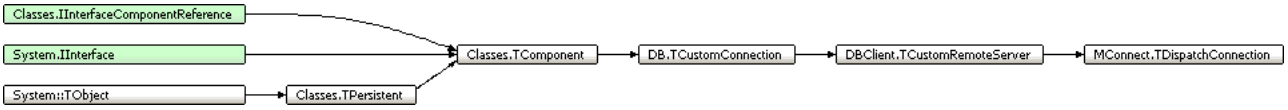

Pascal

```
TDispatchConnection = class(TCustomRemoteServer);
```

C++

```
class TDispatchConnection : public TCustomRemoteServer;
```

File

MConnect

Description

Do not create instances of TDispatchConnection. Instead, use TDispatchConnection as a base class when creating a custom component that forms a connection to a COM-based application server in a multi-tiered database application.

TDispatchConnection introduces the infrastructure for logging on to an application server, obtaining an IAppServer interface, calling the application server's interface, and managing a remote connection. Additionally, TCustomRemoteServer inherits the ability to work with one or more client datasets.

TDispatchConnection is not capable of connecting to an application server. This ability is added by descendants, which are specialized for different connection types. Once a connection is formed, however, TDispatchConnection makes the application server's IAppServer interface available to TClientDataSet components.

To enable a client application to connect to a COM-based application server, use a descendant of TDispatchConnection such as TSocketConnection, TWebConnection, or TDCOMConnection (see page 16008).

See Also

TDataSetProvider (see page 18508)

TClientDataSet (see page 8289)

TDCOMConnection (see page 16008)

TSocketConnection (see page 18928)

IAppServer (see page 16450)

TWebConnection (see page 18968)

TSOAPConnection

94.1.5.1 TDispatchConnection Members

The following tables list the members exposed by TDispatchConnection.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoConnect (see page 6864)	Provides the interface for a method that opens a connection.
	DoDisconnect (see page 16023)	Drops a connection to an application server.
	GetAppServer (see page 8530)	Returns the value of the AppServer (see page 8532) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetConnected (see page 16023)	Returns the value of the Connected (see page 16026) property.
	GetDataSet (see page 6865)	Returns a dataset, given its index in the DataSets (see page 6870) property array.
	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetServerCLSID (see page 16024)	Returns the GUID for the application server.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6866)	Initializes the connection component after all objects in the form or data module have been loaded.
	Notification (see page 16025)	Responds when other objects are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RegisterClient (see page 6866)	Registers an object that uses the connection component.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppServer (see page 8531)	Sets the AppServer (see page 8532) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnected (see page 16025)	Sets the Connected (see page 16026) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	UnRegisterClient (see page 6868)	Removes a registered client so that it is no longer associated with the connection component.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ObjectBroker (see page 16028)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 6863)	Closes the connection.
	Create (see page 16022)	Creates an instance of a dispatch connection component.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 6863)	Destroys the instance of a connection component.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetProviderNames (see page 8530)	Executes a callback for all provider objects on the application server.
V	GetServer (see page 16023)	Returns the IAppServer interface of the application server.
V	GetServerList (see page 16024)	Returns a list of remote data modules to which the component can connect.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 6866)	Opens the connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AppServer (see page 8532)	Contains the interface for communicating with the application server.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 6870)	Lists the datasets currently using this connection component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterConnect (see page 16031)	Occurs after a connection is established.
	AfterDisconnect (see page 16031)	Occurs after the connection closes.
	BeforeConnect (see page 16032)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 16032)	Occurs immediately before the connection closes.

	OnGetUsername (see page 16033)	This is OnGetUsername, a member of class TDispatchConnection.

Published Properties

	Name	Description
	Connected (see page 16026)	Indicates whether the TDispatchConnection (see page 16017) object is connected to an application server.
	LoginPrompt (see page 16027)	Specifies whether a login dialog appears immediately before opening a new connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OnLogin (see page 16028)	Occurs when a communication channel opens to the server.
	ServerGUID (see page 16030)	Specifies the GUID of the application server with which to connect.
	ServerName (see page 16030)	Specifies the name of the application server with which to connect.
	Tag (see page 2808)	Stores an integer value as part of a component.

94.1.5.2 TDispatchConnection.Create Constructor

Creates an instance of a dispatch connection component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDispatchConnection(TComponent * AOwner);
```

Description

Call Create to instantiate a dispatch connection component at runtime. Connection components placed in forms or data modules at design time are created automatically.

After calling the inherited constructor, Create initializes the LoginPrompt (see page 16027) property to false.

See Also

LoginPrompt (see page 6872)

Create (see page 8529)

94.1.5.3 TDispatchConnection Methods

The methods of the TDispatchConnection class are listed here.

Protected Methods

	Name	Description
	DoDisconnect (see page 16023)	Drops a connection to an application server.
	GetConnected (see page 16023)	Returns the value of the Connected (see page 16026) property.
	GetServerCLSID (see page 16024)	Returns the GUID for the application server.
	Notification (see page 16025)	Responds when other objects are created or destroyed.
	SetConnected (see page 16025)	Sets the Connected (see page 16026) property.

Public Methods

	Name	Description
	GetServer (see page 16023)	Returns the IAppServer interface of the application server.
	GetServerList (see page 16024)	Returns a list of remote data modules to which the component can connect.

94.1.5.3.1 TDispatchConnection.DoDisconnect Method

Drops a connection to an application server.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

DoDisconnect is called internally when the Connected (see page 16026) property is set to false. It drops the connection and changes the AppServer property to a NULL Variant.

See Also

AfterDisconnect (see page 6875)

BeforeDisconnect (see page 6875)

Connected (see page 16026)

AppServer (see page 8532)

94.1.5.3.2 TDispatchConnection.GetConnected Method

Returns the value of the Connected (see page 16026) property.

Pascal

```
function GetConnected: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected implementation of the Connected (see page 16026) property. It uses the AppServer property (set by the DoConnect method of a TDispatchConnection (see page 16017) descendant) to determine whether a connection is established.

If a connection is established, GetConnected returns true. Otherwise, GetConnected returns false.

See Also

Connected (see page 16026)

DoConnect (see page 16001)

94.1.5.3.3 TDispatchConnection.GetServer Method

Returns the IAppServer interface of the application server.

Pascal

```
function GetServer: IAppServer; override;
```

C++

```
virtual __fastcall IAppServer GetServer();
```

Description

Client datasets call GetServer to set the value of their AppServer property. This allows them to access providers on the remote application server.

GetServer ensures that there is a connection and then returns an IAppServer interface that can be used to communicate with providers on the application server. To use other methods of the application server's interface, use the AppServer property instead.

See Also

Connected (see page 16026)

AppServer (see page 8532)

AppServer (see page 8462)

94.1.5.3.4 TDispatchConnection.GetServerCLSID Method

Returns the GUID for the application server.

Pascal

```
function GetServerCLSID: TGUID;
```

C++

```
__fastcall TGUID GetServerCLSID();
```

Description

GetServerCLSID is used internally to obtain the GUID for the application server.

In most cases, GetServerCLSID returns the value of the ServerGUID (see page 16030) property. However, if the application server was specified using the ServerName (see page 16030) property, and the application server was not registered when the ServerName (see page 16030) property was set, ServerGUID (see page 16030) will not represent the application server's GUID. GetServerCLSID can return a GUID that only becomes available when a connection is opened.

If neither the ServerName (see page 16030) nor the ServerGUID (see page 16030) property is set, GetServerCLSID raises an exception.

See Also

ServerGUID (see page 16030)

ServerName (see page 16030)

94.1.5.3.5 TDispatchConnection.GetServerList Method

Returns a list of remote data modules to which the component can connect.

Pascal

```
function GetServerList: OleVariant; override;
```

C++

```
virtual __fastcall OleVariant GetServerList();
```

Description

Connection components use GetServerList internally to fill the drop down list in the Object Inspector for the ServerName (see page 16030) property. GetServerList returns a variant array that lists all application servers that are on the local machine, registered in the system registry, or on the remote machine identified by the connection component.

See Also

ServerName (see page 16030)

94.1.5.3.6 TDispatchConnection.Notification Method

Responds when other objects are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically so objects can make internal adjustments when components are created or destroyed. TDispatchConnection (see page 16017) overrides the inherited method to set the ObjectBroker (see page 16028) property to nil (Delphi) or NULL (C++) when it receives notification that that object is about to be destroyed.

See Also

ObjectBroker (see page 16028)

94.1.5.3.7 TDispatchConnection.SetConnected Method

Sets the Connected (see page 16026) property.

Pascal

```
procedure SetConnected(Value: Boolean); override;
```

C++

```
virtual __fastcall SetConnected(Boolean Value);
```

Description

SetConnected is the protected implementation of the Connected (see page 16026) property. It establishes a connection to the Application server if Value is true, and disconnects from the application server if Value is false.

If Value is false, SetConnected generates a BeforeDisconnect (see page 16032) event, closes the connection, and generates an AfterDisconnect (see page 16031) event.

If Value is true, SetConnected performs the following actions:

1. If LoginPrompt (see page 16027) is true, and the connection is not already open, SetConnected generates an OnGetUsername (see page 16033) event, and uses that to initialize a login dialog. It displays the login dialog, and aborts the connection if the user cancels.
2. It generates a BeforeConnect (see page 16032) event.
3. It opens the connection.
4. It generates an AfterConnect (see page 16031) event.
5. If LoginPrompt (see page 16027) is true, and the connection opened successfully, SetConnected generates an OnLogin (see page 16028) event.

See Also

Connected (see page 16026)

BeforeConnect (see page 6875)

AfterConnect (see page 6874)

BeforeDisconnect (see page 6875)

AfterDisconnect (see page 6875)

LoginPrompt (see page 6872)

OnGetUserName (see page 8533)

OnLogin (see page 6873)

94.1.5.4 TDispatchConnection Properties

The properties of the TDispatchConnection class are listed here.

Protected Properties

	Name	Description
	ObjectBroker (see page 16028)	Specifies an Object Broker that allows the connection component to choose from several application servers.

Published Properties

	Name	Description
	Connected (see page 16026)	Indicates whether the TDispatchConnection (see page 16017) object is connected to an application server.
	LoginPrompt (see page 16027)	Specifies whether a login dialog appears immediately before opening a new connection.
	OnLogin (see page 16028)	Occurs when a communication channel opens to the server.
	ServerGUID (see page 16030)	Specifies the GUID of the application server with which to connect.
	ServerName (see page 16030)	Specifies the name of the application server with which to connect.

94.1.5.4.1 TDispatchConnection.Connected Property

Indicates whether the TDispatchConnection (see page 16017) object is connected to an application server.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Set Connected to true to establish a connection to the application server. Set Connected to false to terminate the connection.

When changing the value of Connected, TDispatchConnection (see page 16017) generates a BeforeConnect (see page 16032) and an AfterConnect (see page 16031) event, or a BeforeDisconnect (see page 16032) and an AfterDisconnect (see page 16031) event. In addition, when setting Connected to true, TDispatchConnection (see page 16017) may display a login dialog, depending on the value of LoginPrompt (see page 16027).

The Connected property uses the protected DoConnect and DoDisconnect (see page 16023) methods to connect to or disconnect from an application server. Descendants of TDispatchConnection (see page 16017) override DoConnect and DoDisconnect (see page 16023) to change the way the connection is formed or terminated.

See Also

DoDisconnect (see page 16023)

LoginPrompt (see page 6872)

AfterConnect (see page 6874)

AfterDisconnect (see page 6875)

BeforeConnect (see page 6875)

BeforeDisconnect (see page 6875)

DoConnect (see page 16001)

94.1.5.4.2 TDispatchConnection.LoginPrompt Property

Specifies whether a login dialog appears immediately before opening a new connection.

Pascal

```
property LoginPrompt: Boolean;
```

C++

```
__property Boolean LoginPrompt;
```

Description

Set LoginPrompt to true to provide login support when establishing a connection. LoginPrompt controls two things: the occurrence of the OnLogin event, and the appearance of a default login dialog that prompts users for a name and password when you include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi). When the OnLogin event occurs and when the default login dialog appears depend on the type of connection component:

For TDatabase, TIBDatabase, and TSQLConnection, the dialog appears after the BeforeConnect event and before the AfterConnect event, unless you supply an OnLogin event handler. If there is an OnLogin event handler, that event occurs in place of the login dialog, and there is no need to include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi). If correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler, the connection fails. The OnLogin event does not fire unless LoginPrompt is set to true.

For TADOConnection components, the dialog appears after the OnWillConnect event and before the BeforeConnect event. If there is an OnLogin event handler, that event occurs after the login dialog. If you do not include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi), the OnLogin event occurs but there is no default login dialog. If correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler, the connection fails. The OnLogin event does not fire unless LoginPrompt is set to true.

For DataSnap SOAP connection components, the dialog appears after the OnGetUsername event and before the BeforeConnect, AfterConnect, and OnLogin events. If the user cancels from the login dialog, no attempt is made to open a connection.

When LoginPrompt is false, the application must supply user name and password values programmatically:

For TDatabase, the user name and password can be supplied as USER NAME and PASSWORD parameters in the Params property.

For TADOConnection, the user name and password can be supplied as the ConnectionString property.

For TSQLConnection, the user name, password, and database can be supplied as UserName, Password, and Database parameters in the Params property, or provided as connection parameters associated with the connection name.

For DataSnap SOAP connection components, there is no built-in use for the user name and password supplied by the login dialog. (The UserName and Password properties of TWebConnection are unrelated).

Warning: Storing hard-coded user name and password entries as property values or in code for an OnLogin event handler can compromise server security.

See Also

OnLogin (see page 6873)

BeforeConnect (see page 6875)

AfterConnect (see page 6874)

OnGetUserName (see page 8533)

OnLogin (see page 20259)

OnLogin (see page 9994)

Controlling Server Login

94.1.5.4.3 TDispatchConnection.ObjectBroker Property

Specifies an Object Broker that allows the connection component to choose from several application servers.

Pascal

```
property ObjectBroker: TCustomObjectBroker;
```

C++

```
__property TCustomObjectBroker ObjectBroker;
```

Description

Set ObjectBroker to use a broker that maintains a list of servers that support the application specified by ServerName (see page 16030) or ServerGUID (see page 16030). When the connection component needs to open a connection, it requests information to identify a server machine from the Object Broker.

Descendants of TDispatchConnection (see page 16017) define properties that specify how to locate server machines. These properties are not saved with the connection component if the ObjectBroker property is set. Instead, the Object Broker provides this information dynamically at runtime from its list of available servers.

Applications can use TSimpleObjectBroker for an object broker, or derive their own object brokers from TCustomObjectBroker (see page 16002).

See Also

DoConnect (see page 6864)

ServerName (see page 16030)

ServerGUID (see page 16030)

TSimpleObjectBroker (see page 17877)

Brokering Connections

Specifying a Connection Using HTTP

Specifying a Connection Using DCOM

Specifying a Connection Using Sockets

Using TSimpleObjectBroker

94.1.5.4.4 TDispatchConnection.OnLogin Property

Occurs when a communication channel opens to the server.

Pascal

```
property OnLogin: TLoginEvent;
```

C++

```
__property TLoginEvent OnLogin;
```

Description

OnLogin occurs when connections are opened if the LoginPrompt property is true.

For DataSnap SOAP connection components, the following sequence of events occurs when opening a connection to an application server:

1. An OnGetUsername event occurs, where the application can supply an initial user name for the login dialog.
2. The login dialog appears, where the user can supply a user name and password.
3. A BeforeConnect event occurs, the connection is opened, and an AfterConnect event occurs.
4. The OnLogin event occurs, where you can write code to log in to the application server using the username and password that were provided in the remote login dialog.

   Note: Do not use the OnLogin event if the connection component provides a different scheme for logging in. For example, TWebConnection uses the Username and Password properties instead.

   For ADO connection components, the following sequence of events occurs when opening a connection to an application server:

1. A BeforeConnect event occurs.

2. An OnWillConnect event occurs, where the application can supply an initial user name and password for the login dialog, as well as the connection options and status.

3. The login dialog appears, where the user can supply a user name and password.

4. The OnLogin event occurs, where you can write code to respond to the user-supplied values. (For example, you could convert an application-specific user name and password to database user name and password).

5. The connection is opened, and an AfterConnect event occurs.

Note: Some

C connection components that do not connect to an application server in a multitiered database application replace the OnLogin event defined in TCustomConnection with a different event that occurs before a connection is established.

See Also

Connected (see page 6868)

Open (see page 6866)

OnGetUserName (see page 8533)

BeforeConnect (see page 6875)

AfterConnect (see page 6874)

LoginPrompt (see page 6872)

OnWillConnect (see page 1039)

94.1.5.4.5 TDispatchConnection.ServerGUID Property

Specifies the GUID of the application server with which to connect.

Pascal

```
property ServerGUID: string;
```

C++

```
__property AnsiString ServerGUID;
```

Description

Use ServerGUID to specify the application server to which the client application should connect by its GUID (global unique identifier). If the value specifies a valid GUID for a registered application server, setting ServerGUID changes the value of ServerName (see page 16030) to the class name of the corresponding COM object.

Using ServerGUID rather than ServerName (see page 16030) to identify the application server is more robust because it does not require the application server to be registered on the client system.

Note: Either ServerName (see page 16030) or ServerGUID must be provided so that the dispatch connection can create and communicate with the appropriate server COM object. This is true even for connections that are not formed using DCOM, because the remote data module on the application server is always implemented as a COM object.

See Also

ServerName (see page 16030)

Connecting to the Application Server

Connecting to the Application Server using DataSnap Components

94.1.5.4.6 TDispatchConnection.ServerName Property

Specifies the name of the application server with which to connect.

Pascal

```
property ServerName: string;
```

C++

```
__property AnsiString ServerName;
```

Description

Use ServerName to specify the name of the application server to which the client application should connect. At design time, the Object Inspector provides a drop-down list of all available servers if one of the following conditions is met:

For DCOM and OLEnterprise connections, the servers must be registered in the system Registry (see page 18633).

For Socket connections, ScktSrvr.exe must be running on the server machine.

For Web connections, the connection component must have an URL for locating httpsrvr.dll.

If ServerName specifies a valid name for a registered application server, setting ServerName sets the ServerGUID (see page 16030) property to the GUID that corresponds to the named application server.

Note: Either ServerName or ServerGUID (see page 16030) must be provided so that the dispatch connection can create and communicate with the appropriate server COM object. This is true even for connections that are not formed using DCOM, because the remote data module on the application server is always implemented as a COM object. Using ServerGUID (see page 16030) rather than ServerName to identify the application server is more robust because it does not require the application server to be registered on the client system when using a COM-based connection.

See Also

ServerGUID (see page 16030)

Connecting to the Application Server

94.1.5.5 TDispatchConnection Events

The events of the TDispatchConnection class are listed here.

Published Events

	Name	Description
	AfterConnect (see page 16031)	Occurs after a connection is established.
	AfterDisconnect (see page 16031)	Occurs after the connection closes.
	BeforeConnect (see page 16032)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 16032)	Occurs immediately before the connection closes.
	OnGetUsername (see page 16033)	This is OnGetUsername, a member of class TDispatchConnection.

94.1.5.5.1 TDispatchConnection.AfterConnect Event

Occurs after a connection is established.

Pascal

```
property AfterConnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterConnect;
```

Description

Write an AfterConnect event handler to take application-specific actions immediately after the connection component opens a connection to the remote source of database information.

See Also

BeforeConnect (see page 6875)

Connected (see page 6868)

Connecting to a Database Server

94.1.5.5.2 TDispatchConnection.AfterDisconnect Event

Occurs after the connection closes.

Pascal

```
property AfterDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterDisconnect;
```

Description

Write an AfterDisconnect event handler to take application-specific actions after the connection component drops a connection.

See Also

BeforeDisconnect (see page 6875)

Connected (see page 6868)

Disconnecting from a Database Server

94.1.5.5.3 TDispatchConnection.BeforeConnect Event

Occurs immediately before establishing a connection.

Pascal

```
property BeforeConnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeConnect;
```

Description

Write a BeforeConnect event handler to take application-specific actions before the connection component opens a connection to the remote source of database information.

See Also

AfterConnect (see page 6874)

Connected (see page 6868)

Connecting to a Database Server

94.1.5.5.4 TDispatchConnection.BeforeDisconnect Event

Occurs immediately before the connection closes.

Pascal

```
property BeforeDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeDisconnect;
```

Description

Write a BeforeDisconnect event handler to take application-specific actions before dropping a connection.

Delphi Examples:

```
{
This BeforeDisconnect event handler makes sure that the
client dataset has applied all its pending updates before
terminating the connection to the server.
}
procedure TForm1.RemoteServer1BeforeDisconnect(Sender: TObject);
begin
  if (ClientDataSet1.ChangeCount > 0) then
    ClientDataSet1.ApplyUpdates(-1);
end;
```

See Also

AfterDisconnect (see page 6875)

Connected (see page 6868)

Disconnecting from a Database Server

94.1.5.5.5 TDispatchConnection.OnGetUsername Event

Pascal

```
property OnGetUsername: TGetUsernameEvent;
```

C++

```
__property TGetUsernameEvent OnGetUsername;
```

Description

This is OnGetUsername, a member of class TDispatchConnection.

94.1.6 TOLEnterpriseConnection Class

TOLEnterpriseConnection is provided for backward compatibility with applications that connect to an application server using OLEnterprise.

Class Hierarchy

Pascal

```
TOLEnterpriseConnection = class(TCOMConnection);
```

C++

```
class TOLEnterpriseConnection : public TCOMConnection;
```

File

MConnect

Description

TOLEnterpriseConnection is used in the client portion of a multi-tiered database application to establish and maintain the connection between the client and a remote application server using OLEnterprise. This component is provided for backward compatibility.

TOLEnterpriseConnection establishes the initial connection between the client application and a remote application server using OLEnterprise. Once the connection is established, the client registers any or all of its client datasets with the remote server, and these client datasets use the IAppServer interface from the OLEnterprise connection component to communicate with providers on the application server, or to otherwise call the application server's data module interface.

To connect to remote application servers using DCOM, use the TDCOMConnection (see page 16008) component instead. To connect client applications to remote application servers using TCP/IP sockets, use TSocketConnection. To connect client applications to remote application servers using HTTP, use TWebConnection. To connect to remote application servers using SOAP, use TSOAPConnection.

See Also

TClientDataSet (see page 8289)

TSocketConnection (see page 18928)

TDCOMConnection (see page 16008)

TRemoteDataModule (see page 6585)

IAppServer (see page 16450)

TWebConnection (see page 18968)

94.1.6.1 TOLEnterpriseConnection Members

The following tables list the members exposed by TOLEnterpriseConnection.

Protected Events

	Name	Description
	OnGetUsername (see page 8533)	This is OnGetUsername, a member of class TCustomRemoteServer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoConnect (see page 16038)	Establishes a connection to the application server.
V	DoDisconnect (see page 6864)	Provides the interface for a method that terminates the connection.
V	DoDisconnect (see page 16023)	Drops a connection to an application server.
V	GetAppServer (see page 8530)	Returns the value of the AppServer (see page 8532) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetConnected (see page 6865)	Returns the value of the Connected (see page 6868) property.
V	GetConnected (see page 16023)	Returns the value of the Connected (see page 16026) property.
V	GetDataSet (see page 6865)	Returns a dataset, given its index in the DataSets (see page 6870) property array.
V	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetServerCLSID (see page 16024)	Returns the GUID for the application server.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 6866)	Initializes the connection component after all objects in the form or data module have been loaded.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16025)	Responds when other objects are created or destroyed.

	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	RegisterClient (see page 6866)	Registers an object that uses the connection component.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAppServer (see page 8531)	Sets the AppServer (see page 8532) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetConnected (see page 6867)	Changes the value of the Connected (see page 6868) property.
V	SetConnected (see page 16001)	Sets the Connected (see page 16026) property.
V	SetConnected (see page 16025)	Sets the Connected (see page 16026) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	UnRegisterClient (see page 6868)	Removes a registered client so that it is no longer associated with the connection component.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ObjectBroker (see page 16028)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.

Public Events

	Name	Description
	AfterConnect (see page 6874)	Occurs after a connection is established.
	AfterDisconnect (see page 6875)	Occurs after the connection closes.
	BeforeConnect (see page 6875)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 6875)	Occurs immediately before the connection closes.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 6863)	Closes the connection.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 6862)	Creates an instance of TCustomConnection (see page 6858).
V	Create (see page 8529)	Creates an instance of a remote server component.
V	Create (see page 16022)	Creates an instance of a dispatch connection component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 6863)	Destroys the instance of a connection component.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetProviderNames (see page 8530)	Executes a callback for all provider objects on the application server.
V	GetServer (see page 8530)	Returns the IAppServer interface of the application server.
V	GetServer (see page 16023)	Returns the IAppServer interface of the application server.
V	GetServerList (see page 8531)	Returns a list of application servers (remote data modules) to which the component can connect.
V	GetServerList (see page 16024)	Returns a list of remote data modules to which the component can connect.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 6866)	Opens the connection.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AppServer (see page 8532)	Contains the interface for communicating with the application server.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Connected (see page 6868)	Determines whether a connection has been established to the remote source of data.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 6870)	Lists the datasets currently using this connection component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	LoginPrompt (see page 6872)	Specifies whether a login dialog appears immediately before opening a new connection.
	OnLogin (see page 6873)	Occurs when a communication channel opens to the server.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterConnect (see page 16031)	Occurs after a connection is established.
	AfterDisconnect (see page 16031)	Occurs after the connection closes.
	BeforeConnect (see page 16032)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 16032)	Occurs immediately before the connection closes.
	OnGetUsername (see page 16033)	This is OnGetUsername, a member of class TDispatchConnection.

Published Properties

	Name	Description
	BrokerName (see page 16038)	Specifies the name of the Business Object Broker that locates the application server.
	ComputerName (see page 16039)	Specifies the name of the machine on which the application server is located.
	Connected (see page 16026)	Indicates whether the TDispatchConnection (see page 16017) object is connected to an application server.
	LoginPrompt (see page 16027)	Specifies whether a login dialog appears immediately before opening a new connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OnLogin (see page 16028)	Occurs when a communication channel opens to the server.
	ServerGUID (see page 16030)	Specifies the GUID of the application server with which to connect.
	ServerName (see page 16030)	Specifies the name of the application server with which to connect.
	Tag (see page 2808)	Stores an integer value as part of a component.

94.1.6.2 TOLEnterpriseConnection Methods

The methods of the TOLEnterpriseConnection class are listed here.

Protected Methods

	Name	Description
	DoConnect (see page 16038)	Establishes a connection to the application server.

94.1.6.2.1 TOLEnterpriseConnection.DoConnect Method

Establishes a connection to the application server.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

DoConnect is called when the Connected (see page 16026) property is set to true. DoConnect obtains the interface of the remote data module on the application server using OLEnterprise runtime. This interface can then be used to obtain interfaces for the providers on the application server or to make interface calls to the remote data module.

See Also

AfterConnect (see page 6874)

BeforeConnect (see page 6875)

Connected

AppServer (see page 8532)

94.1.6.3 TOLEnterpriseConnection Properties

The properties of the TOLEnterpriseConnection class are listed here.

Published Properties

	Name	Description
	BrokerName (see page 16038)	Specifies the name of the Business Object Broker that locates the application server.
	ComputerName (see page 16039)	Specifies the name of the machine on which the application server is located.

94.1.6.3.1 TOLEnterpriseConnection.BrokerName Property

Specifies the name of the Business Object Broker that locates the application server.

Pascal

```
property BrokerName: string;
```

C++

```
__property AnsiString BrokerName;
```

Description

Use BrokerName to specify the OLEnterprise Business Object Broker with which the application server is registered. The Business Object Broker locates an available application server with the name specified by the ServerName (see page 16030) property.

If you are not using the Business Object Broker, use the ComputerName (see page 16039) property instead. You must specify a value for either BrokerName or ComputerName (see page 16039):

Note: Setting the value of BrokerName changes the ComputerName (see page 16039) property to blank, and vice versa.

See Also

ServerName (see page 16030)

ComputerName (see page 16039)

94.1.6.3.2 TOLEnterpriseConnection.ComputerName Property

Specifies the name of the machine on which the application server is located.

Pascal

```
property ComputerName: string;
```

C++

```
__property AnsiString ComputerName;
```

Description

Use ComputerName to specify the machine where the application server resides if you are not using the Business Object Broker. If you are not using the Business Object Broker and ComputerName is blank (the default), TOLEnterpriseConnection (see page 16033) assumes the application server can be found on the user's machine or that the computer name can be derived from the system registry based on the DCOM configuration. If ComputerName is not blank, it overrides any system registry entries.

If you are using the Business Object Broker, use the BrokerName (see page 16038) property instead. BrokerName (see page 16038) and ComputerName are mutually exclusive: Setting the value of ComputerName changes the BrokerName (see page 16038) property to blank, and vice versa.

See Also

ServerName (see page 16030)

BrokerName (see page 16038)

94.1.7 TSharedConnection Class

TSharedConnection connects to a child remote data module that is accessible only through the properties of a main remote data module on the application server.

Class Hierarchy

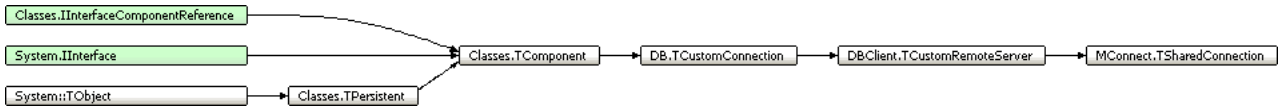

Pascal

```
TSharedConnection = class(TCustomRemoteServer);
```

C++

```
class TSharedConnection : public TCustomRemoteServer;
```

File

MConnect

Description

TSharedConnection manages a connection to a "child" remote data module that is managed by a single main remote data module on an application server. The client application does not connect directly to the "child" remote data module. Rather, it uses a single connection to the main remote data module on the server machine, and that main remote data module dispatches calls to the IAppServer interface of other "child" data modules. This allows the client application to use a single connection rather than one connection for each remote data module on the application server.

On the server-side, the interface for the child remote data module is surfaced as a property of the main remote data module's interface. The TSharedConnection component uses this property name to indicate which child remote data module is the actual target of its connection.

TSharedConnection uses a TDispatchConnection (see page 16017) descendant (TDCOMConnection (see page 16008), TWebConnection, or TSocketConnection) to form the connection to the main remote data module on the application server. When you use TSharedConnection to open a connection, it causes the associated dispatch connection to connect to the main remote data module if it has not already done so. When you close the connection maintained by TSharedConnection, however, it severs only the connection to the child remote data module, leaving the connection to the main remote data module open in case other components on the client application are still using it.

See Also

TCustomClientDataSet (see page 8373)

TSocketConnection (see page 18928)

TWebConnection (see page 18968)

TDCOMConnection (see page 16008)

Connecting to an Application Server That Uses Multiple Data Modules

94.1.7.1 TSharedConnection Members

The following tables list the members exposed by TSharedConnection.

Protected Events

	Name	Description
	OnGetUsername (see page 8533)	This is OnGetUsername, a member of class TCustomRemoteServer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.

	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ConnectEvent (see page 16045)	This is ConnectEvent, a member of class TSharedConnection.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoConnect (see page 16045)	Provides the interface for a method that opens a connection.
	DoDisconnect (see page 16045)	Provides the interface for a method that terminates the connection.
	GetAppServer (see page 8530)	Returns the value of the AppServer (see page 8532) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetConnected (see page 16046)	Returns the value of the Connected property.
	GetDataSet (see page 6865)	Returns a dataset, given its index in the DataSets (see page 6870) property array.
	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6866)	Initializes the connection component after all objects in the form or data module have been loaded.
	Notification (see page 16046)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RegisterClient (see page 6866)	Registers an object that uses the connection component.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppServer (see page 8531)	Sets the AppServer (see page 8532) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnected (see page 6867)	Changes the value of the Connected (see page 6868) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	UnRegisterClient (see page 6868)	Removes a registered client so that it is no longer associated with the connection component.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.

	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 6863)	Closes the connection.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 16044)	Destroys the TSharedConnection (see page 16039) instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProviderNames (see page 8530)	Executes a callback for all provider objects on the application server.
	GetServer (see page 16046)	Returns the IAppServer interface of the child remote data module.
	GetServerList (see page 8531)	Returns a list of application servers (remote data modules) to which the component can connect.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 6866)	Opens the connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AppServer (see page 8532)	Contains the interface for communicating with the application server.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Connected (see page 6868)	Determines whether a connection has been established to the remote source of data.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 6870)	Lists the datasets currently using this connection component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	LoginPrompt (see page 6872)	Specifies whether a login dialog appears immediately before opening a new connection.
	OnLogin (see page 6873)	Occurs when a communication channel opens to the server.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterConnect (see page 16048)	Occurs after a connection is established.
	AfterDisconnect (see page 16049)	Occurs after the connection closes.
	BeforeConnect (see page 16049)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 16049)	Occurs immediately before the connection closes.

Published Methods

	Name	Description
V	Create (see page 16044)	Creates an instance of a remote server component.

Published Properties

	Name	Description
	ChildName (see page 16047)	Identifies the child remote data module on the application server with which you want to communicate.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentConnection (see page 16048)	Specifies the connection component that establishes a connection to the main remote data module on the application server.
	Tag (see page 2808)	Stores an integer value as part of a component.

94.1.7.2 TSharedConnection.Create Constructor

Creates an instance of a remote server component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSharedConnection(TComponent * AOwner);
```

Description

Create instantiates an instance of a remote server component and initializes its properties.

AOwner is the component that is responsible for freeing the TCustomRemoteServer instance. It becomes the value of the Owner property.

See Also

Create (see page 6862)

Owner (see page 2807)

94.1.7.3 TSharedConnection.Destroy Destructor

Destroys the TSharedConnection (see page 16039) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSharedConnection();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which verifies that the reference to the shared connection component is not nil before calling Destroy

See Also

Free (see page 22300)

94.1.7.4 TSharedConnection Methods

The methods of the TSharedConnection class are listed here.

Protected Methods

	Name	Description
	ConnectEvent (see page 16045)	This is ConnectEvent, a member of class TSharedConnection.
V	DoConnect (see page 16045)	Provides the interface for a method that opens a connection.
V	DoDisconnect (see page 16045)	Provides the interface for a method that terminates the connection.
V	GetConnected (see page 16046)	Returns the value of the Connected property.
V	Notification (see page 16046)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
V	GetServer (see page 16046)	Returns the IAppServer interface of the child remote data module.

94.1.7.4.1 TSharedConnection.ConnectEvent Method

Pascal

```
procedure ConnectEvent(Sender: TObject; Connecting: Boolean);
```

C++

```
__fastcall ConnectEvent(TObject * Sender, Boolean Connecting);
```

Description

This is ConnectEvent, a member of class TSharedConnection.

94.1.7.4.2 TSharedConnection.DoConnect Method

Provides the interface for a method that opens a connection.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

The Connected property uses DoConnect to establish a connection. Descendant classes override the DoConnect method to establish their connection as appropriate. As implemented in TCustomConnection, DoConnect does nothing.

See Also

Connected (see page 6868)

DoDisconnect (see page 6864)

94.1.7.4.3 TSharedConnection.DoDisconnect Method

Provides the interface for a method that terminates the connection.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

The Connected property uses DoDisconnect to close a connection. Descendant classes override the DoDisconnect method to drop a connection. As implemented in TCustomConnection, DoDisconnect does nothing.

See Also

Connected (see page 6868)

DoConnect (see page 6864)

94.1.7.4.4 TSharedConnection.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected read implementation of the Connected property. Descendant classes override this method to return true if a connection is open and false otherwise. As defined in TCustomConnection, GetConnected always returns false.

See Also

Connected (see page 6868)

SetConnected (see page 6867)

94.1.7.4.5 TSharedConnection.GetServer Method

Returns the IAppServer interface of the child remote data module.

Pascal

```
function GetServer: IAppServer; override;
```

C++

```
virtual __fastcall IAppServer GetServer();
```

Description

Call GetServer to obtain the interface of the remote data module on the application server. GetServer returns the IAppServer interface of the same remote data module as the one whose interface is the value of the AppServer property. Client applications can communicate directly with the child remote data module through this interface.

See Also

AppServer (see page 8532)

94.1.7.4.6 TSharedConnection.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

94.1.7.5 TSharedConnection Properties

The properties of the TSharedConnection class are listed here.

Published Properties

	Name	Description
	ChildName (see page 16047)	Identifies the child remote data module on the application server with which you want to communicate.
	ParentConnection (see page 16048)	Specifies the connection component that establishes a connection to the main remote data module on the application server.

94.1.7.5.1 TSharedConnection.ChildName Property

Identifies the child remote data module on the application server with which you want to communicate.

Pascal

```
property ChildName: String;
```

C++

```
__property AnsiString ChildName;
```

Description

Set ChildName to the name of the property on the application server's main remote data module that exposes the interface for the desired child remote data module. TSharedConnection (see page 16039) uses this property to obtain the interface for the child remote data module, which it sets as the value of the AppServer property.

See Also

AppServer (see page 8532)

ParentConnection (see page 16048)

94.1.7.5.2 TSharedConnection.ParentConnection Property

Specifies the connection component that establishes a connection to the main remote data module on the application server.

Pascal

```
property ParentConnection: TDispatchConnection;
```

C++

```
__property TDispatchConnection ParentConnection;
```

Description

Set ParentConnection to a connection component that connects to the main remote data module on the application server. TSharedConnection (see page 16039) uses this component's connection for communicating to the child remote data module.

The main remote data module with which ParentConnection communicates must expose the interface for the child remote data module as a property on its IAppServer interface.

See Also

AppServer (see page 8532)

ChildName (see page 16047)

AppServer (see page 8532)

94.1.7.6 TSharedConnection Events

The events of the TSharedConnection class are listed here.

Published Events

	Name	Description
	AfterConnect (see page 16048)	Occurs after a connection is established.
	AfterDisconnect (see page 16049)	Occurs after the connection closes.
	BeforeConnect (see page 16049)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 16049)	Occurs immediately before the connection closes.

94.1.7.6.1 TSharedConnection.AfterConnect Event

Occurs after a connection is established.

Pascal

```
property AfterConnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterConnect;
```

Description

Write an AfterConnect event handler to take application-specific actions immediately after the connection component opens a connection to the remote source of database information.

See Also

BeforeConnect (see page 6875)

Connected (see page 6868)

Connecting to a Database Server

94.1.7.6.2 TSharedConnection.AfterDisconnect Event

Occurs after the connection closes.

Pascal

```
property AfterDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterDisconnect;
```

Description

Write an AfterDisconnect event handler to take application-specific actions after the connection component drops a connection.

See Also

BeforeDisconnect (see page 6875)

Connected (see page 6868)

Disconnecting from a Database Server

94.1.7.6.3 TSharedConnection.BeforeConnect Event

Occurs immediately before establishing a connection.

Pascal

```
property BeforeConnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeConnect;
```

Description

Write a BeforeConnect event handler to take application-specific actions before the connection component opens a connection to the remote source of database information.

See Also

AfterConnect (see page 6874)

Connected (see page 6868)

Connecting to a Database Server

94.1.7.6.4 TSharedConnection.BeforeDisconnect Event

Occurs immediately before the connection closes.

Pascal

```
property BeforeDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeDisconnect;
```

Description

Write a BeforeDisconnect event handler to take application-specific actions before dropping a connection.

Delphi Examples:

```
{
This BeforeDisconnect event handler makes sure that the
client dataset has applied all its pending updates before
terminating the connection to the server.
}
procedure TForm1.RemoteServer1BeforeDisconnect(Sender: TObject);
begin
  if (ClientDataSet1.ChangeCount > 0) then
    ClientDataSet1.ApplyUpdates(-1);
end;
```

See Also

AfterDisconnect (see page 6875)

Connected (see page 6868)

Disconnecting from a Database Server

94.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetMIDASAppServerList (see page 16051)	This is function MConnect.GetMIDASAppServerList.

94.2.1 MConnect.GetMIDASAppServerList Function

Pascal

```
procedure GetMIDASAppServerList(List: TStringList; const RegCheck: string);
```

C++

```
GetMIDASAppServerList(TStringList List, const AnsiString RegCheck);
```

File

MConnect

Description

This is function MConnect.GetMIDASAppServerList.

95 Menus Namespace

This is namespace Menus.

Topics

Name	Description
Classes (see page 16055)	The following table lists classes in this documentation.
Functions (see page 16186)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 16198)	The following table lists structs, records, enums in this documentation.
Types (see page 16203)	The following table lists types in this documentation.
Variables (see page 16207)	The following table lists variables in this documentation.
Constants (see page 16209)	The following table lists constants in this documentation.

Classes

	Name	Description
	EMenuError (see page 16055)	EMenuError is the exception class for errors involving menu items.
	TMainMenu (see page 16056)	TMainMenu encapsulates a menu bar and its accompanying drop-down menus for a form.
	TMenu (see page 16067)	TMenu is the base type for menu components such as TMainMenu (see page 16056) and TPopupMenu (see page 16169).
	TMenuActionLink (see page 16090)	TMenuActionLink links menu item clients to the corresponding menu action.
	TMenuItem (see page 16102)	TMenuItem describes the properties of an item in a menu.
	TMenuItemEnumerator (see page 16160)	This is class Menus.TMenuItemEnumerator.
	TMenuItemStack (see page 16162)	This is class Menus.TMenuItemStack.
	TPopupList (see page 16164)	TPopupList manages the window that handles popup menu messages.
	TPopupMenu (see page 16169)	TPopupMenu encapsulates the properties, methods, and events of a pop-up menu.

Constants

Name	Description
cDialogSuffix (see page 16209)	This is constant Menus.cDialogSuffix.
cHotkeyPrefix (see page 16209)	This is constant Menus.cHotkeyPrefix.
cLineCaption (see page 16209)	This is constant Menus.cLineCaption.

Functions

	Name	Description
	AnsiSameCaption (see page 16186)	Indicates whether two strings are the same based on the current locale, ignoring case information and accelerator characters.

	DrawMenuItem (see page 16187)	This is function Menus.DrawMenuItem.
	GetHotkey (see page 16187)	Returns the last character in a string marked with an ampersand (&).
	IsAltGRPressed (see page 16187)	This is function Menus.IsAltGRPressed.
	NewItem (see page 16188)	Creates and initializes a menu item.
	NewLine (see page 16189)	Creates a menu item for a separator.
	NewMenu (see page 16190)	Creates and initializes a main menu.
	NewPopupMenu (see page 16191)	Creates and initializes a popup menu.
	NewSubMenu (see page 16192)	Creates and initializes a menu item that represents a submenu.
	ShortCut (see page 16193)	Creates a menu shortcut for a specified key combination.
	ShortCutFromMessage (see page 16194)	This is function Menus.ShortCutFromMessage.
	ShortCutToKey (see page 16194)	Returns the key code and shift state of a menu shortcut.
	ShortCutToText (see page 16195)	Converts a menu shortcut into a string that describes the shortcut.
	StripHotkey (see page 16196)	This is function Menus.StripHotkey.
	TextToShortCut (see page 16196)	Creates a menu shortcut from a text string.

Structs, Records, Enums

	Name	Description
	TFindItemKind (see page 16198)	TFindItemKind indicates the criterion to use when locating menu items.
	TMenuAnimations (see page 16199)	TMenuAnimation (see page 16204) and TMenuAnimations indicate how a popup menu appears on screen.
	TMenuBreak (see page 16199)	TMenuBreak indicates the criterion to use when locating menu items.
	TMenuItemAutoFlag (see page 16200)	TMenuItemAutoFlag and TMenuAutoFlag (see page 16205) Indicate when menu items are automatically maintained.
	TPopupAlignment (see page 16201)	TPopupAlignment represents the position of a popup menu relative to the mouse.
	TTrackButton (see page 16201)	TTrackButton specifies which mouse button activates a tool button menu.

Types

Name	Description
PMenuItem (see page 16203)	This is type Menus.PMenuItem.
TAdvancedMenuDrawItemEvent (see page 16203)	TAdvancedMenuDrawItemEvent is the type for event handlers that render menu items in owner-drawn menus.
TMenuActionLinkClass (see page 16204)	This is type Menus.TMenuActionLinkClass.
TMenuAnimation (see page 16204)	TMenuAnimation and TMenuAnimations (see page 16199) indicate how a popup menu appears on screen.
TMenuAutoFlag (see page 16205)	TMenuItemAutoFlag (see page 16200) and TMenuAutoFlag Indicate when menu items are automatically maintained.
TMenuChangeEvent (see page 16205)	This is type Menus.TMenuChangeEvent.
TMenuDrawItemEvent (see page 16206)	TMenuDrawItemEvent is the type for event handlers that render menu items in owner-drawn menus.
TMenuMeasureItemEvent (see page 16206)	TMenuMeasureItemEvent is the type for event handlers that render menu items in owner-Measuren menus.

Variables

Name	Description
PopupList (see page 16207)	This is variable Menus.PopupList.
ShortCutItems (see page 16207)	This is variable Menus.ShortCutItems.
ValidMenuHotkeys (see page 16207)	This is variable Menus.ValidMenuHotkeys.

95.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EMenuError (see page 16055)	EMenuError is the exception class for errors involving menu items.
	TMainMenu (see page 16056)	TMainMenu encapsulates a menu bar and its accompanying drop-down menus for a form.
	TMenu (see page 16067)	TMenu is the base type for menu components such as TMainMenu (see page 16056) and TPopupMenu (see page 16169).
	TMenuActionLink (see page 16090)	TMenuActionLink links menu item clients to the corresponding menu action.
	TMenuItem (see page 16102)	TMenuItem describes the properties of an item in a menu.
	TMenuItemEnumerator (see page 16160)	This is class Menus.TMenuItemEnumerator.
	TMenuItemStack (see page 16162)	This is class Menus.TMenuItemStack.
	TPopupList (see page 16164)	TPopupList manages the window that handles popup menu messages.
	TPopupMenu (see page 16169)	TPopupMenu encapsulates the properties, methods, and events of a pop-up menu.

95.1.1 EMenuError Class

EMenuError is the exception class for errors involving menu items.

Class Hierarchy

Pascal

```
EMenuError = class(Exception);
```

C++

```
class EMenuError : public Exception;
```

File

Menus

Description

EMenuError is raised if an error occurs while an application is working with menu items. For example, an EMenuError exception occurs if

An application tries to delete an item from an empty menu

A menu item attempts to insert itself into it's own list of items

An application attempts to reference a menu that does not exist

An application specifies an invalid GroupIndex for a menu item

See Also

TMenuItem (see page 16102)

TMenu (see page 16067)

95.1.2 TMainMenu Class

TMainMenu encapsulates a menu bar and its accompanying drop-down menus for a form.

Class Hierarchy

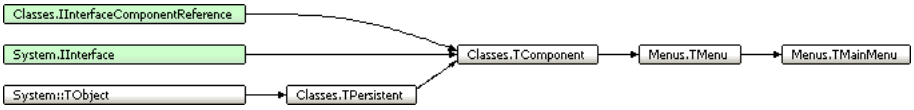

Pascal

```
TMainMenu = class(TMenu);
```

C++

```
class TMainMenu : public TMenu;
```

File

Menus

Description

Use TMainMenu to provide the main menu for a form. To begin designing a menu, add a main menu to a form, and double-click the component.

TMainMenu introduces properties and methods for merging the drop-down menus of the main menu with the main menu of another form.

TMainMenu introduces properties and methods for assisting in the menu negotiation process of an OLE container.

Tip: To create a menu that users can configure at runtime, use the TActionMainMenuBar component with an action manager instead.

See Also

TMenuItem (see page 16102)

TPopupMenu (see page 16169)

TActionMainMenuBar (see page 585)

TActionManager (see page 432)

95.1.2.1 TMainMenu Members

The following tables list the members exposed by TMainMenu.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.

	AdjustBiDiBehavior (see page 16073)	Applies the BiDiMode (see page 16083) property to the menu when the Windows control is created.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoBiDiModeChanged (see page 16074)	Updates the menu when the BiDiMode (see page 16083) property changes.
V	DoChange (see page 16075)	Generates an OnChange (see page 16090) event.
	DoGetMenuString (see page 16075)	Returns the caption for an item in the menu.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 16076)	Executes a callback for child menu items.
V	GetHandle (see page 16060)	Returns the value of the Handle property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsOwnerDraw (see page 16077)	Indicates whether menu items can contain more than simple captions.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 16078)	Calls the DoChange (see page 16075) method when the menu is first loaded into memory.
V	MenuChanged (see page 16061)	Responds when the menu changes.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16079)	Responds when components are added or deleted.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 16081)	Ensures that the menu item indexes reflect the order in which they are loaded and saved.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateItems (see page 16081)	Causes all menu items to update themselves.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.

	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 16072)	Creates an instance of TMenu (see page 16067).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateMenuItem (see page 16073)	This is CreateMenuItem, a member of class TMenu.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 16072)	Destroys the menu component.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DispatchCommand (see page 16074)	Calls the Click method of the menu item with the specified value as its Command property.
	DispatchPopup (see page 16074)	Dispatches a pop-up menu command.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindItem (see page 16076)	Locates and returns a menu item.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHelpContext (see page 16077)	Returns the help context ID associated with a menu item.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetOle2AcceleratorTable (see page 16060)	Returns the main menu's accelerator table.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 16077)	Indicates whether menu items read from right to left.
	IsShortCut (see page 16078)	Dispatches a key message if it represents a shortcut key for an item in the menu.
	Merge (see page 16061)	Combines the main menu of one form with the main menu of another in non-MDI applications.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentBiDiModeChanged (see page 16079)	This is the overview for the ParentBiDiModeChanged method overload.
	PopulateOle2Menu (see page 16062)	Populates an OLE menu with items from the main menu.
	ProcessMenuChar (see page 16080)	Responds to keyboard input for the menu.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetOle2MenuHandle (see page 16062)	Associates the handle for an OLE menu with the main menu.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	Unmerge (see page 16063)	Reverses the merging of two menus in a non-MDI application.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Handle (see page 16083)	Provides access to the Windows menu handle for the menu.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	WindowHandle (see page 16089)	Provides access to the handle of the window that uses the menu.

Published Events

	Name	Description
	OnChange (see page 16067)	Occurs when the menu changes.

Published Properties

	Name	Description
	AutoHotkeys (see page 16063)	Determines whether the accelerator keys for menu items can be reset automatically.
	AutoLineReduction (see page 16064)	Determines whether redundant separator bars are automatically removed from the menu.

	AutoMerge (see page 16065)	Determines if the main menus of secondary forms merge with the main menu of the main form in non-MDI applications at runtime.
	BiDiMode (see page 16065)	Determines the layout of the menu in Middle East locales.
	Images (see page 16065)	Lists the images that can appear beside individual menu items.
R	Items (see page 16084)	Describes the elements of the menu.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OwnerDraw (see page 16066)	Specifies whether the menu items are drawn by the application.
	ParentBiDiMode (see page 16066)	Indicates whether the BiDiMode property is inherited from the control that uses the menu.
	Tag (see page 2808)	Stores an integer value as part of a component.

95.1.2.2 TMainMenu Methods

The methods of the TMainMenu class are listed here.

Protected Methods

	Name	Description
V	GetHandle (see page 16060)	Returns the value of the Handle property.
V	MenuChanged (see page 16061)	Responds when the menu changes.

Public Methods

	Name	Description
	GetOle2AcceleratorTable (see page 16060)	Returns the main menu's accelerator table.
	Merge (see page 16061)	Combines the main menu of one form with the main menu of another in non-MDI applications.
	PopulateOle2Menu (see page 16062)	Populates an OLE menu with items from the main menu.
	SetOle2MenuHandle (see page 16062)	Associates the handle for an OLE menu with the main menu.
	Unmerge (see page 16063)	Reverses the merging of two menus in a non-MDI application.

95.1.2.2.1 TMainMenu.GetHandle Method

Returns the value of the Handle property.

Pascal

```
function GetHandle: HMENU; override;
```

C++

```
virtual __fastcall HMENU GetHandle();
```

Description

GetHandle is the protected implementation of the Handle property. Override this method to change the implementation of the handle property. For example, TMainMenu overrides GetHandle to adjust for menus added by embedded OLE objects.

See Also

Handle (see page 16083)

95.1.2.2.2 TMainMenu.GetOle2AcceleratorTable Method

Returns the main menu's accelerator table.

Pascal

```
procedure GetOle2AcceleratorTable(var AccelTable: HAccel; var AccelCount: Integer; const
Groups: array of Integer);
```

C++

```
__fastcall GetOle2AcceleratorTable(HAccel AccelTable, int AccelCount, const array of Integer
Groups);
```

Description

Call GetOle2AcceleratorTable to obtain a handle to an accelerator table for all menu items with a GroupIndex value that appears in the Groups parameter. The AccelTable parameter is set to the accelerator table handle, and the AccelCount is set to the number of entries in the table. GetOle2AcceleratorTable is used by OLE containers when merging menus.

See Also

GroupIndex (see page 16146)

ShortCut (see page 16152)

95.1.2.2.3 TMainMenu.MenuChanged Method

Responds when the menu changes.

Pascal

```
procedure MenuChanged(Sender: TObject; Source: TMenuItem; Rebuild: Boolean); override;
```

C++

```
virtual __fastcall MenuChanged(TObject * Sender, TMenuItem Source, Boolean Rebuild);
```

Description

MenuChanged is called automatically when changes occur that affect the structure of the menu. The Sender parameter indicates the object that originates the change. It's value can be nil (Delphi) or NULL (C++). Source indicates which menu item is affected. Rebuild indicates whether the menu must be rebuilt to reflect the change (for example when items are added or deleted).

MenuChanged calls the DoChange method, which generates an OnChange event.

Note: Changes that affect individual items rather than the menu as a whole trigger the UpdateItems method instead of MenuChanged. For example, when the Images property changes, UpdateItems is called rather than MenuChanged.

See Also

Items (see page 16084)

UpdateItems (see page 16081)

DoChange (see page 16075)

95.1.2.2.4 TMainMenu.Merge Method

Combines the main menu of one form with the main menu of another in non-MDI applications.

Pascal

```
procedure Merge(Menu: TMainMenu);
```

C++

```
__fastcall Merge(TMainMenu Menu);
```

Description

Call Merge to merge another main menu with this one. Specify the menu you want merged with this menu as the Menu parameter.

For example, to use the main menu of the first form in an application as the main menu for the application, call Merge to merge

the main menu of any additional forms in the application with the main menu of the first form.

Depending on the value of the GroupIndex property of menu items on the main menu, the merged menu items can replace menu items on the menu bar, or insert menu items into the menu bar. See GroupIndex for information on how to do these things.

To merge and unmerge main menus automatically when other forms are displayed, change the value of the AutoMerge (see page 16065) property to true on all forms whose menus should be merged in.

See Also

AutoMerge (see page 16065)

GroupIndex (see page 16146)

Unmerge (see page 16063)

95.1.2.2.5 TMainMenu.PopulateOle2Menu Method

Populates an OLE menu with items from the main menu.

Pascal

```
procedure PopulateOle2Menu(SharedMenu: HMenu; const Groups: array of Integer; var Widths:
array of Longint);
```

C++

```
__fastcall PopulateOle2Menu(HMenu SharedMenu, const array of Integer Groups, array of Longint
Widths);
```

Description

Call PopulateOle2Menu to merge the menu items with GroupIndex values specified by the Groups parameter into the menu that uses the SharedMenu handle. PopulateOle2Menu appends all menu items in the specified groups to the menu, and increments the corresponding entries in the Widths array for every menu item appended. PopulateOle2Menu is used by OLE containers when merging menus.

Note: PopulateOle2Menu populates the menu specified by the SharedMenu parameter but does not set this as the OLE menu associated with the menu.

See Also

GetOle2AcceleratorTable (see page 16060)

SetOle2MenuHandle (see page 16062)

GroupIndex (see page 16146)

95.1.2.2.6 TMainMenu.SetOle2MenuHandle Method

Associates the handle for an OLE menu with the main menu.

Pascal

```
procedure SetOle2MenuHandle(Handle: HMENU);
```

C++

```
__fastcall SetOle2MenuHandle(HMENU Handle);
```

Description

Call SetOle2MenuHandle to temporarily replace the menu of a form with the menu represented by the Handle (see page 16083) parameter. To restore the original menu, call SetOle2MenuHandle with the Handle (see page 16083) parameter set to 0. SetOle2MenuHandle allows an OLE container to use menu items provided by an OLE server.

See Also

GetOle2AcceleratorTable (see page 16060)

PopulateOle2Menu (see page 16062)

95.1.2.2.7 TMainMenu.Unmerge Method

Reverses the merging of two menus in a non-MDI application.

Pascal

```
procedure Unmerge(Menu: TMainMenu);
```

C++

```
__fastcall Unmerge(TMainMenu Menu);
```

Description

Call Unmerge to remove items from another main menu that were added using the Merge (see page 16061) method. The Menu parameter is the merged menu whose items should be removed from the main menu.

See Also

AutoMerge (see page 16065)

Merge (see page 16061)

95.1.2.3 TMainMenu Properties

The properties of the TMainMenu class are listed here.

Published Properties

	Name	Description
	AutoHotkeys (see page 16063)	Determines whether the accelerator keys for menu items can be reset automatically.
	AutoLineReduction (see page 16064)	Determines whether redundant separator bars are automatically removed from the menu.
	AutoMerge (see page 16065)	Determines if the main menus of secondary forms merge with the main menu of the main form in non-MDI applications at runtime.
	BiDiMode (see page 16065)	Determines the layout of the menu in Middle East locales.
	Images (see page 16065)	Lists the images that can appear beside individual menu items.
	OwnerDraw (see page 16066)	Specifies whether the menu items are drawn by the application.
	ParentBiDiMode (see page 16066)	Indicates whether the BiDiMode property is inherited from the control that uses the menu.

95.1.2.3.1 TMainMenu.AutoHotkeys Property

Determines whether the accelerator keys for menu items can be reset automatically.

Pascal

```
property AutoHotkeys: TMenuAutoFlag;
```

C++

```
__property TMenuAutoFlag AutoHotkeys;
```

Description

Set AutoHotkeys to indicate whether the menu should automatically ensure that accelerator keys for its items are reset if

necessary so that every menu item has an accelerator key and no two items have the same accelerator key. This ensures that menu items added dynamically at runtime can have accelerator keys that do not introduce conflicts. TMenu tries to preserve all existing accelerator key mappings when adjusting the accelerator keys, but may be forced to change some in order to ensure that every item has a unique accelerator.

Before the menu is displayed, the menu checks and adjusts the accelerator key mappings if AutoHotkeys is maAutomatic. When AutoHotkeys is maManual, the menu does not alter the top-level menu items, but may adjust the accelerator keys for submenus, depending on the AutoHotkeys property of the menu item that is the parent of the submenu.

When AutoHotkeys is maManual, an application can still adjust the menu items by calling the Items property's RethinkHotkeys method. AutoHotkeys merely controls whether this adjustment occurs automatically.

Note: The AutoHotkeys property of a menu is linked to the AutoHotkeys property of the TMenuItems object that implements its Items property. Setting one sets the other.

See Also

AutoHotkeys (see page 16134)

Items (see page 16084)

RethinkHotkeys (see page 16128)

95.1.2.3.2 TMainMenu.AutoLineReduction Property

Determines whether redundant separator bars are automatically removed from the menu.

Pascal

```
property AutoLineReduction: TMenuAutoFlag;
```

C++

```
__property TMenuAutoFlag AutoLineReduction;
```

Description

Set AutoLineReduction to indicate whether the menu should automatically ensure that the menu does not start or end with a separator bar and that two separator bars do not appear next to each other. This ensures that menus built dynamically at runtime do not have separator bars in undesired positions.

Before the menu is displayed, it removes superfluous separator bars when AutoLineReduction is maAutomatic. When AutoLineReduction is maManual, the menu does not remove separators from the top-level menu items, but may remove superfluous separators from submenus, depending on the AutoLineReduction property of the menu item that is the parent of the submenu.

When AutoLineReduction is maManual, an application can still remove superfluous separators by calling the Items property's RethinkLines method. AutoLineReduction merely determines whether this adjustment occurs automatically.

Note: The AutoLineReduction property of a menu is linked to the AutoLineReduction property of the TMenuItems object that implements its Items property. Setting one sets the other.

See Also

AutoLineReduction (see page 16135)

Items (see page 16084)

RethinkLines (see page 16129)

95.1.2.3.3 TMainMenu.AutoMerge Property

Determines if the main menus of secondary forms merge with the main menu of the main form in non-MDI applications at runtime.

Pascal

```
property AutoMerge: Boolean;
```

C++

```
__property Boolean AutoMerge;
```

Description

To merge a form's menus with the main menu in the main form, set the AutoMerge property of each main menu you want merged to true. Make sure that the AutoMerge property of the main menu of the main form (the one other menus are merged into) remains false. How menus merge depends on the value of the GroupIndex property for each menu item.

If the application is an MDI application (the FormStyle properties are set so the main form is a parent form and subsequent forms are child forms), menu merging occurs automatically without using the AutoMerge property.

See Also

FormStyle (see page 13912)

GroupIndex (see page 16146)

Merge (see page 16061)

95.1.2.3.4 TMainMenu.BiDiMode Property

Determines the layout of the menu in Middle East locales.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Set BiDiMode to control the layout of menu items. BiDiMode controls whether the menu items read from left to right or right to left.

See Also

ParentBiDiMode (see page 16089)

IsRightToLeft (see page 16077)

95.1.2.3.5 TMainMenu.Images Property

Lists the images that can appear beside individual menu items.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Use Images to hold the set of images that can appear beside menu items in the menu. Individual menu items set their ImageIndex property to indicate which image in the list appears to the left of the menu item caption.

Images in the list should be 16x16. To accommodate every system on which your application runs, it is recommended that you do not use 256 colors.

Tip: If the application uses action lists, this property should be the value of the action list's Images property. This way, menu items and buttons that do the same thing can display the same image.

See Also

ImageIndex (see page 16148)

Images (see page 325)

Items (see page 16084)

95.1.2.3.6 TMainMenu.OwnerDraw Property

Specifies whether the menu items are drawn by the application.

Pascal

```
property OwnerDraw: Boolean;
```

C++

```
__property Boolean OwnerDraw;
```

Description

Set OwnerDraw to true to indicate that the application will draw menu items in an OnDrawItem event handler. When OwnerDraw is true, menu items receive an OnMeasureItem and an OnDrawItem event when they need to be rendered on screen.

Set OwnerDraw to false to accept the default drawing. When OwnerDraw is false, menu items display the caption with an optional image to the left (If the Images and ImageIndex properties are set).

Owner-draw menus are useful for displaying the value that appears when selecting a menu item. For example, in a menu that allows users to choose a color, an owner-draw menu could display rectangles drawn in the designated color, rather than color names.

See Also

OnMeasureItem (see page 16160)

OnDrawItem (see page 16159)

Images (see page 16083)

95.1.2.3.7 TMainMenu.ParentBiDiMode Property

Indicates whether the BiDiMode property is inherited from the control that uses the menu.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

By default, ParentBiDiMode is true, indicating that the layout of menu items follows the BiDiMode setting of the associated form or control. Change ParentBiDiMode to false to override the default BiDiMode setting and control the menu's layout independently.

See Also

BiDiMode (see page 16083)

95.1.2.4 TMainMenu Events

The events of the TMainMenu class are listed here.

Published Events

	Name	Description
	OnChange (see page 16067)	Occurs when the menu changes.

95.1.2.4.1 TMainMenu.OnChange Event

Occurs when the menu changes.

Pascal

```
property OnChange: TMenuChangeEvent;
```

C++

```
__property TMenuChangeEvent OnChange;
```

Description

Write an OnChange event handler to respond to changes in the menu's structure. OnChange occurs when the menu is loaded into memory, and in response to property changes that affect the menu's structure.

See Also

Items (see page 16084)

95.1.3 TMenu Class

TMenu is the base type for menu components such as TMainMenu (see page 16056) and TPopupMenu (see page 16169).

Class Hierarchy

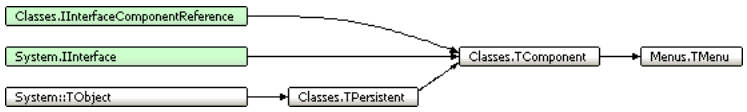

Pascal

```
TMenu = class(TComponent);
```

C++

```
class TMenu : public TComponent;
```

File

Menus

Description

Use TMenu as a base class when defining a component that represents a collection of menu items. TMenu introduces properties and methods to create a Windows menu that represents a set of menu items, dispatch commands associated with those menu items, and locate the help context id for any of those menu items.

See Also

TMainMenu (see page 16056)

TMenuItem (see page 16102)

TPopupMenu (see page 16169)

Disabling Menu Items

Adding Images to Menu Items

Adding

Building Menus

Determining the Order of Merged Menu Items: GroupIndex Property

Editing Menu Items in the Object Inspector

Importing Resource Files

Manipulating Menu Items at Runtime

Merging Menus

Moving Menu Items

Naming Conventions for Template Menu Items and Event Handlers

Naming Menus

Naming the Menu Items

Opening the Menu Designer

Saving a Menu as a Template

Specifying Accelerator Keys and Keyboard Shortcuts

Specifying the Active Menu: Menu Property

Switching Between Menus at Design Time

Using Menu Templates

Using the Menu Designer Context Menu

Viewing the Menu

95.1.3.1 TMenu Members

The following tables list the members exposed by TMenu.

Protected Events

	Name	Description
	OnChange (see page 16090)	Occurs when the menu changes.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AdjustBiDiBehavior (see page 16073)	Applies the BiDiMode (see page 16083) property to the menu when the Windows control is created.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoBiDiModeChanged (see page 16074)	Updates the menu when the BiDiMode (see page 16083) property changes.
V	DoChange (see page 16075)	Generates an OnChange (see page 16090) event.
	DoGetMenuString (see page 16075)	Returns the caption for an item in the menu.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
V	GetChildren (see page 16076)	Executes a callback for child menu items.
V	GetHandle (see page 16076)	Returns the value of the Handle (see page 16083) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsOwnerDraw (see page 16077)	Indicates whether menu items can contain more than simple captions.
V	Loaded (see page 16078)	Calls the DoChange (see page 16075) method when the menu is first loaded into memory.
V	MenuChanged (see page 16079)	Responds when the menu changes.
V	Notification (see page 16079)	Responds when components are added or deleted.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetChildOrder (see page 16081)	Ensures that the menu item indexes reflect the order in which they are loaded and saved.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateItems (see page 16081)	Causes all menu items to update themselves.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 16072)	Creates an instance of TMenu (see page 16067).
	CreateMenuItem (see page 16073)	This is CreateMenuItem, a member of class TMenu.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 16072)	Destroys the menu component.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DispatchCommand (see page 16074)	Calls the Click method of the menu item with the specified value as its Command property.
	DispatchPopup (see page 16074)	Dispatches a pop-up menu command.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindItem (see page 16076)	Locates and returns a menu item.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHelpContext (see page 16077)	Returns the help context ID associated with a menu item.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 16077)	Indicates whether menu items read from right to left.
	IsShortCut (see page 16078)	Dispatches a key message if it represents a shortcut key for an item in the menu.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentBiDiModeChanged (see page 16079)	This is the overview for the ParentBiDiModeChanged method overload.
	ProcessMenuChar (see page 16080)	Responds to keyboard input for the menu.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	AutoHotkeys (see page 16082)	Determines whether the accelerator keys for menu items can be reset automatically.
	AutoLineReduction (see page 16082)	Determines whether redundant separator bars are automatically removed from the menu.
	BiDiMode (see page 16083)	Determines the layout of the menu in Middle East locales.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Handle (see page 16083)	Provides access to the Windows menu handle for the menu.
	Images (see page 16083)	Lists the images that can appear beside individual menu items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	OwnerDraw (see page 16088)	Specifies whether the menu items are drawn by the application.
	ParentBiDiMode (see page 16089)	Indicates whether the BiDiMode (see page 16083) property is inherited from the control that uses the menu.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	WindowHandle (see page 16089)	Provides access to the handle of the window that uses the menu.

Published Properties

	Name	Description
R	Items (see page 16084)	Describes the elements of the menu.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

95.1.3.2 TMenu.Create Constructor

Creates an instance of TMenu (see page 16067).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TMenu(TComponent * AOwner);
```

Description

Call Create to create a menu at runtime. Menus (see page 16053) placed on forms at design time are created automatically. Create calls the inherited Create method, then sets the initial values for the menu component, including creating an Items (see page 16084) property.

The AOwner parameter indicates the component (usually a form) that is responsible for managing the memory associated with the menu.

See Also

Items (see page 16084)

95.1.3.3 TMenu.Destroy Destructor

Destroys the menu component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMenu();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TMenu (see page 16067) reference is not nil, and only then calls Destroy. Destroy frees the Items (see page 16084) property, and then calls the inherited Destroy method.

See Also

Free (see page 22300)

Items (see page 16084)

95.1.3.4 TMenu Methods

The methods of the TMenu class are listed here.

Protected Methods

	Name	Description
	AdjustBiDiBehavior (see page 16073)	Applies the BiDiMode (see page 16083) property to the menu when the Windows control is created.
	DoBiDiModeChanged (see page 16074)	Updates the menu when the BiDiMode (see page 16083) property changes.
V	DoChange (see page 16075)	Generates an OnChange (see page 16090) event.
	DoGetMenuString (see page 16075)	Returns the caption for an item in the menu.

	GetChildren (see page 16076)	Executes a callback for child menu items.
	GetHandle (see page 16076)	Returns the value of the Handle (see page 16083) property.
	IsOwnerDraw (see page 16077)	Indicates whether menu items can contain more than simple captions.
	Loaded (see page 16078)	Calls the DoChange (see page 16075) method when the menu is first loaded into memory.
	MenuChanged (see page 16079)	Responds when the menu changes.
	Notification (see page 16079)	Responds when components are added or deleted.
	SetChildOrder (see page 16081)	Ensures that the menu item indexes reflect the order in which they are loaded and saved.
	UpdateItems (see page 16081)	Causes all menu items to update themselves.

Public Methods

	Name	Description
	CreateMenuItem (see page 16073)	This is CreateMenuItem, a member of class TMenu.
	DispatchCommand (see page 16074)	Calls the Click method of the menu item with the specified value as its Command property.
	DispatchPopup (see page 16074)	Dispatches a pop-up menu command.
	FindItem (see page 16076)	Locates and returns a menu item.
	GetHelpContext (see page 16077)	Returns the help context ID associated with a menu item.
	IsRightToLeft (see page 16077)	Indicates whether menu items read from right to left.
	IsShortCut (see page 16078)	Dispatches a key message if it represents a shortcut key for an item in the menu.
	ParentBiDiModeChanged (see page 16079)	This is the overview for the ParentBiDiModeChanged method overload.
	ProcessMenuChar (see page 16080)	Responds to keyboard input for the menu.

95.1.3.4.1 TMenu.AdjustBiDiBehavior Method

Applies the BiDiMode (see page 16083) property to the menu when the Windows control is created.

Pascal

```
procedure AdjustBiDiBehavior;
```

C++

```
__fastcall AdjustBiDiBehavior();
```

Description

AdjustBiDiBehavior is called automatically when the Windows control for the menu is created. It ensures that the BiDiMode (see page 16083) property is properly reflected in the menu if the application is running in a Middle Eastern locale.

See Also

BiDiMode (see page 16083)

95.1.3.4.2 TMenu.CreateMenuItem Method

Pascal

```
function CreateMenuItem: TMenuItem; dynamic;
```

C++

```
__fastcall TMenuItem CreateMenuItem();
```

Description

This is CreateMenuItem, a member of class TMenu.

95.1.3.4.3 TMenu.DispatchCommand Method

Calls the Click method of the menu item with the specified value as its Command property.

Pascal

```
function DispatchCommand(ACommand: Word): Boolean;
```

C++

```
__fastcall Boolean DispatchCommand(Word ACommand);
```

Description

Call DispatchCommand to simulate a click on the menu item associated with a command ID. If DispatchCommand locates a menu item that matches the command ID, it calls the Click method for that menu item and returns true. If there is no such item, DispatchCommand returns false.

See Also

DispatchPopup (see page 16074)

IsShortCut (see page 16078)

Items (see page 16084)

Click (see page 16116)

Command (see page 16141)

95.1.3.4.4 TMenu.DispatchPopup Method

Dispatches a pop-up menu command.

Pascal

```
function DispatchPopup(AHandle: HMENU): Boolean;
```

C++

```
__fastcall Boolean DispatchPopup(HMENU AHandle);
```

Description

Call DispatchPopup to simulate a click on the menu item associated with the pop-up menu handle. If DispatchPopup locates a menu item that matches the pop-up menu handle, it calls the Click method for that menu item and returns true. If there is no such item, DispatchPopup returns false.

See Also

DispatchCommand (see page 16074)

IsShortCut (see page 16078)

Items (see page 16084)

Click (see page 16116)

95.1.3.4.5 TMenu.DoBiDiModeChanged Method

Updates the menu when the BiDiMode (see page 16083) property changes.

Pascal

```
procedure DoBiDiModeChanged;
```

C++

```
__fastcall DoBiDiModeChanged();
```

Description

Do not call DoBiDiModeChanged. It is called automatically when the menu's BiDiMode (see page 16083) property changes to update the menu to reflect the current setting.

See Also

BiDiMode (see page 16083)

95.1.3.4.6 TMenu.DoChange Method

Generates an OnChange (see page 16090) event.

Pascal

```
procedure DoChange(Source: TMenuItem; Rebuild: Boolean); virtual;
```

C++

```
virtual __fastcall DoChange(TMenuItem Source, Boolean Rebuild);
```

Description

DoChange is called automatically when the properties of any menu items change. The Source parameter specifies the menu item (if any) that changed. The Rebuild parameter indicates whether the change requires the menu to regenerate its items.

See Also

OnChange (see page 16090)

MenuChanged (see page 16079)

95.1.3.4.7 TMenu.DoGetMenuString Method

Returns the caption for an item in the menu.

Pascal

```
function DoGetMenuString(Menu: HMENU; ItemID: UINT; Str: PChar; MaxCount: Integer; Flag:
UINT): Integer;
```

C++

```
__fastcall int DoGetMenuString(HMENU Menu, UINT ItemID, const char * Str, int MaxCount, UINT
Flag);
```

Description

DoGetMenuString is used internally to obtain menu item captions, given the Windows menu handle. The Menu parameter indicates the menu's handle. The ItemID parameter specifies the menu item whose caption is desired. If the Flag parameter is MF_BYCOMMAND, ItemID is the Command property of the menu item. If Flag is MF_BYPOSITION, ItemID is the 0-based index of the menu item. The caption is returned in a buffer pointed to by the Str parameter. MaxCount indicates the size of that buffer.

Unlike the Windows API function GetMenuString, DoGetMenuString returns the correct caption for owner-draw menu items.

See Also

Caption (see page 16136)

Command (see page 16141)

95.1.3.4.8 TMenu.FindItem Method

Locates and returns a menu item.

Pascal

```
function FindItem(Value: Integer; Kind: TFindItemKind): TMenuItem;
```

C++

```
__fastcall TMenuItem FindItem(int Value, TFindItemKind Kind);
```

Description

Call FindItem to locate a menu item in the menu or one of its submenus. The menu item is identified by either a pop-up menu handle, command ID, or menu shortcut value. The Kind parameter specifies how the menu item is to be identified.

See Also

TMenuItem (see page 16102)

Command (see page 16141)

Handle (see page 16147)

ShortCut (see page 16152)

95.1.3.4.9 TMenu.GetChildren Method

Executes a callback for child menu items.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called automatically by the streaming system so that child menu items are saved and read with their parent menu.

See Also

GetParentComponent

SetChildOrder

95.1.3.4.10 TMenu.GetHandle Method

Returns the value of the Handle (see page 16083) property.

Pascal

```
function GetHandle: HMENU; virtual;
```

C++

```
virtual __fastcall HMENU GetHandle();
```

Description

GetHandle is the protected implementation of the Handle (see page 16083) property. Override this method to change the implementation of the handle property. For example, TMainMenu (see page 16056) overrides GetHandle to adjust for menus added by embedded OLE objects.

See Also

Handle (see page 16083)

95.1.3.4.11 TMenu.GetHelpContext Method

Returns the help context ID associated with a menu item.

Pascal

```
function GetHelpContext(Value: Integer; ByCommand: Boolean): THelpContext;
```

C++

```
__fastcall THelpContext GetHelpContext(int Value, Boolean ByCommand);
```

Description

Call GetHelpContext to obtain the ID for context-sensitive help on a particular menu item. Identify the menu item as the item that matches the Value property. If ByCommand is true, Value is the Command property of the TMenuItem (see page 16102). If ByCommand is false, Value is the Handle (see page 16083) property.

See Also

FindItem (see page 16076)

Command (see page 16141)

Handle (see page 16147)

HelpContext (see page 16148)

HelpContext (see page 5826)

95.1.3.4.12 TMenu.IsOwnerDraw Method

Indicates whether menu items can contain more than simple captions.

Pascal

```
function IsOwnerDraw: Boolean;
```

C++

```
__fastcall Boolean IsOwnerDraw();
```

Description

IsOwnerDraw returns true when the OwnerDraw (see page 16088) property is true or the Images (see page 16083) property is not nil (Delphi) or NULL (C++). This is used internally to determine how the menu should be painted.

See Also

OwnerDraw (see page 16088)

Images (see page 16083)

95.1.3.4.13 TMenu.IsRightToLeft Method

Indicates whether menu items read from right to left.

Pascal

```
function IsRightToLeft: Boolean;
```

C++

```
__fastcall Boolean IsRightToLeft();
```

Description

Call IsRightToLeft to determine the layout of main menu items. IsRightToLeft returns true if the application is running on a system with a Middle Eastern locale and the BiDiMode (see page 16083) property is not bdLeftToRight.

See Also

BiDiMode (see page 16083)

95.1.3.4.14 TMenu.IsShortCut Method

Dispatches a key message if it represents a shortcut key for an item in the menu.

Pascal

```
function IsShortCut(var Message: TWMKey): Boolean; dynamic;
```

C++

```
__fastcall Boolean IsShortCut(TWMKey Message);
```

Description

Call IsShortCut when processing a KeyDown message to identify shortcut keys and trigger the appropriate menu action. IsShortCut decodes the key message passed in the Message parameter. It then encodes the state of the Shift, Control, and Alt keys with the value of the character that was typed to obtain the ShortCut (see page 16193) property of a menu item. If IsShortCut locates a menu item associated with the shortcut key combination, it calls the item's Click method and returns true. If no item in the menu has a shortcut corresponding to the message, IsShortCut returns false.

See Also

DispatchCommand (see page 16074)

DispatchPopup (see page 16074)

FindItem (see page 16076)

Click (see page 16116)

95.1.3.4.15 TMenu.Loaded Method

Calls the DoChange (see page 16075) method when the menu is first loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Loaded is called automatically immediately after the menu is loaded into memory. It calls the DoChange (see page 16075) method to allow any necessary fixup.

See Also

DoChange (see page 16075)

95.1.3.4.16 TMenu.MenuChanged Method

Responds when the menu changes.

Pascal

```
procedure MenuChanged(Sender: TObject; Source: TMenuItem; Rebuild: Boolean); virtual;
```

C++

```
virtual __fastcall MenuChanged(TObject * Sender, TMenuItem Source, Boolean Rebuild);
```

Description

MenuChanged is called automatically when changes occur that affect the structure of the menu. The Sender parameter indicates the object that originates the change. It's value can be nil (Delphi) or NULL (C++). Source indicates which menu item is affected. Rebuild indicates whether the menu must be rebuilt to reflect the change (for example when items are added or deleted).

MenuChanged calls the DoChange (see page 16075) method, which generates an OnChange (see page 16090) event.

Note: Changes that affect individual items rather than the menu as a whole trigger the UpdateItems (see page 16081) method instead of MenuChanged. For example, when the Images (see page 16083) property changes, UpdateItems (see page 16081) is called rather than MenuChanged.

See Also

Items (see page 16084)

UpdateItems (see page 16081)

DoChange (see page 16075)

95.1.3.4.17 TMenu.Notification Method

Responds when components are added or deleted.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

TMenu (see page 16067) overrides the inherited method to set the Images (see page 16083) property to nil (Delphi) or NULL (C++) if the ImageList is deleted.

See Also

Notification (see page 2783)

Images (see page 16083)

95.1.3.4.18 ParentBiDiModeChanged Method

This is the overview for the ParentBiDiModeChanged method overload.

Overload List

	Name	Description
	TMenu.ParentBiDiModeChanged () (see page 16080)	Ensures that the menu's layout follows the current settings of the BiDiMode (see page 16083) and ParentBiDiMode (see page 16089) properties.
	TMenu.ParentBiDiModeChanged (TObject) (see page 16080)	Ensures that the menu's layout follows the current settings of the BiDiMode (see page 16083) and ParentBiDiMode (see page 16089) properties.

95.1.3.4.18.1 TMenu.ParentBiDiModeChanged Method ()

Ensures that the menu's layout follows the current settings of the BiDiMode (see page 16083) and ParentBiDiMode (see page 16089) properties.

Pascal

```
procedure ParentBiDiModeChanged; overload;
procedure ParentBiDiModeChanged(AControl: TObject); overload;
```

C++

```
__fastcall ParentBiDiModeChanged();
__fastcall ParentBiDiModeChanged(TObject * AControl);
```

Description

ParentBiDiMode (see page 16089) changed is called when the menu's BiDiMode (see page 16083) property must be updated to reflect the bi-directional mode of its associated control.

See Also

BiDiMode (see page 16083)

ParentBiDiMode (see page 16089)

DoBiDiModeChanged (see page 16074)

95.1.3.4.19 TMenu.ProcessMenuChar Method

Responds to keyboard input for the menu.

Pascal

```
procedure ProcessMenuChar(var Message: TWMMenuChar);
```

C++

```
__fastcall ProcessMenuChar(TWMMenuChar Message);
```

Description

ProcessMenuChar handles keyboard messages from Windows. It checks whether the character typed is an accelerator character from one of the menu's items. If so, ProcessMenuChar executes the menu item (or, if the item is disabled, selects it). If the key is not an accelerator, ProcessMenuChar checks for captions that begin with the character. If there is only one, it is treated like an accelerator. Otherwise, the first item beginning with that character is selected.

When calling ProcessMenuChar programmatically, the User, MenuFlag, and Menu fields of Message parameter must be supplied.

See Also

Caption (see page 16136)

95.1.3.4.20 TMenu.SetChildOrder Method

Ensures that the menu item indexes reflect the order in which they are loaded and saved.

Pascal

```
procedure SetChildOrder(Child: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Child, int Order);
```

Description

SetChildOrder is called automatically when menus are loaded into memory. The Child parameter specifies the child object that was just loaded. The Order parameter specifies the order in which it was loaded. SetChildOrder assigns the MenuIndex for each child to match the order in which the items are loaded.

See Also

MenuIndex (see page 16150)

95.1.3.4.21 TMenu.UpdateItems Method

Causes all menu items to update themselves.

Pascal

```
procedure UpdateItems;
```

C++

```
__fastcall UpdateItems();
```

Description

UpdateItems is called automatically when changes occur that must be reflected by individual menu items. Changes that affect the menu as a whole trigger the MenuChanged (see page 16079) method instead.

See Also

MenuChanged (see page 16079)

95.1.3.5 TMenu Properties

The properties of the TMenu class are listed here.

Public Properties

	Name	Description
	AutoHotkeys (see page 16082)	Determines whether the accelerator keys for menu items can be reset automatically.
	AutoLineReduction (see page 16082)	Determines whether redundant separator bars are automatically removed from the menu.
	BiDiMode (see page 16083)	Determines the layout of the menu in Middle East locales.
R	Handle (see page 16083)	Provides access to the Windows menu handle for the menu.
	Images (see page 16083)	Lists the images that can appear beside individual menu items.
	OwnerDraw (see page 16088)	Specifies whether the menu items are drawn by the application.
	ParentBiDiMode (see page 16089)	Indicates whether the BiDiMode (see page 16083) property is inherited from the control that uses the menu.
	WindowHandle (see page 16089)	Provides access to the handle of the window that uses the menu.

Published Properties

	Name	Description
	Items (see page 16084)	Describes the elements of the menu.

95.1.3.5.1 TMenu.AutoHotkeys Property

Determines whether the accelerator keys for menu items can be reset automatically.

Pascal

```
property AutoHotkeys: TMenuAutoFlag;
```

C++

```
__property TMenuAutoFlag AutoHotkeys;
```

Description

Set AutoHotkeys to indicate whether the menu should automatically ensure that accelerator keys for its items are reset if necessary so that every menu item has an accelerator key and no two items have the same accelerator key. This ensures that menu items added dynamically at runtime can have accelerator keys that do not introduce conflicts. TMenu (see page 16067) tries to preserve all existing accelerator key mappings when adjusting the accelerator keys, but may be forced to change some in order to ensure that every item has a unique accelerator.

Before the menu is displayed, the menu checks and adjusts the accelerator key mappings if AutoHotkeys is maAutomatic. When AutoHotkeys is maManual, the menu does not alter the top-level menu items, but may adjust the accelerator keys for submenus, depending on the AutoHotkeys property of the menu item that is the parent of the submenu.

When AutoHotkeys is maManual, an application can still adjust the menu items by calling the Items (see page 16084) property's RethinkHotkeys method. AutoHotkeys merely controls whether this adjustment occurs automatically.

Note: The AutoHotkeys property of a menu is linked to the AutoHotkeys property of the TMenuItems object that implements its Items (see page 16084) property. Setting one sets the other.

See Also

AutoHotkeys (see page 16134)

Items (see page 16084)

RethinkHotkeys (see page 16128)

95.1.3.5.2 TMenu.AutoLineReduction Property

Determines whether redundant separator bars are automatically removed from the menu.

Pascal

```
property AutoLineReduction: TMenuAutoFlag;
```

C++

```
__property TMenuAutoFlag AutoLineReduction;
```

Description

Set AutoLineReduction to indicate whether the menu should automatically ensure that the menu does not start or end with a separator bar and that two separator bars do not appear next to each other. This ensures that menus built dynamically at runtime do not have separator bars in undesired positions.

Before the menu is displayed, it removes superfluous separator bars when AutoLineReduction is maAutomatic. When AutoLineReduction is maManual, the menu does not remove separators from the top-level menu items, but may remove

superfluous separators from submenus, depending on the AutoLineReduction property of the menu item that is the parent of the submenu.

When AutoLineReduction is maManual, an application can still remove superfluous separators by calling the Items (see page 16084) property's RethinkLines method. AutoLineReduction merely determines whether this adjustment occurs automatically.

Note: The AutoLineReduction property of a menu is linked to the AutoLineReduction property of the TMenuItems object that implements its Items (see page 16084) property. Setting one sets the other.

See Also

AutoLineReduction (see page 16135)

Items (see page 16084)

RethinkLines (see page 16129)

95.1.3.5.3 TMenu.BiDiMode Property

Determines the layout of the menu in Middle East locales.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Set BiDiMode to control the layout of menu items. BiDiMode controls whether the menu items read from left to right or right to left.

See Also

ParentBiDiMode (see page 16089)

IsRightToLeft (see page 16077)

95.1.3.5.4 TMenu.Handle Property

Provides access to the Windows menu handle for the menu.

Pascal

```
property Handle: HMENU;
```

C++

```
__property HMENU Handle;
```

Description

Use Handle to call a Windows API function that requires a menu handle. Handle is the handle for the menu that contains only the items described by the Items (see page 16084) property. If no other menus have been merged with this menu, this is the same as the WindowHandle (see page 16089) property.

See Also

WindowHandle (see page 16089)

95.1.3.5.5 TMenu.Images Property

Lists the images that can appear beside individual menu items.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Use Images to hold the set of images that can appear beside menu items in the menu. Individual menu items set their ImageIndex property to indicate which image in the list appears to the left of the menu item caption.

Images in the list should be 16x16. To accommodate every system on which your application runs, it is recommended that you do not use 256 colors.

Tip: If the application uses action lists, this property should be the value of the action list's Images property. This way, menu items and buttons that do the same thing can display the same image.

See Also

ImageIndex (see page 16148)

Images (see page 325)

Items (see page 16084)

95.1.3.5.6 TMenu.Items Property

Describes the elements of the menu.

Pascal

```
property Items: TMenuItem;
```

C++

```
__property TMenuItem Items;
```

Description

Use Items to access information about the elements in the menu. Items is a single TMenuItem (see page 16102) object that describes the elements of the menu in its own Items property.

Note: Because Items is the default property of TMenuItem (see page 16102), the Items property of TMenu (see page 16067) can be treated as an indexed array of menu items that describe the individual items in the menu. That is, instead of writing

Note: FirstItem := Menu1.Items.Items[0];

Note: you can use

Note: FirstItem := Menu1.Items[0];

Set the Items property at design time by clicking on the Items property in the Property Inspector. The Menu Designer automatically inserts a menu item. As each menu item is entered, outlined regions appear at the possible locations of additional menu items. Click in those regions to specify additional menu items. At runtime, create a TMenuItem (see page 16102) that describes the elements of the menu programmatically, and set the Items property of the menu.

C++ Examples:

```
/*
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *NewItem;
  // first create the separator
  NewItem = new TMenuItem(this); // If you change the AOwner (this) to MainMenu1->Items[1] and
then you add the new menu you will get "Menu inserted twice"?  Doesn't happen in Delphi.
  NewItem->Name = "Separator";
  NewItem->Caption = "-";
  // add the new item to the Windows menu
  MainMenu1->Items[1].Add(NewItem);
  // now create and add a menu item for each form
  for  (int I = 0; I < Screen->FormCount; I++)
  {
    NewItem = new TMenuItem(this);
    NewItem->Caption = Screen->Forms[I]->Name + "Item";
    NewItem->Name = Screen->Forms[I]->Name;
    MainMenu1->Items[1].Add(NewItem);
  }
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  // Count down, not up!
  for  (int I = MainMenu1->Items[1].Count - 1; I >= 0; I--)
  {
    delete MainMenu1->Items[1].Items[I];
  }
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
```

```
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
}
procedure TForm1.Button1Click(Sender: TObject);
var
  NewItem: TMenuItem;
  I : integer;
begin
  { first create the separator }
  NewItem := TMenuItem.Create(MainMenu1.Items[1]);
  NewItem.Name:= 'Separator';
  NewItem.Caption := '-';
  { add the new item to the Windows menu }
  MainMenu1.Items[1].Add(NewItem);
  { now create and add a menu item for each form }
  for  I := 0 to Screen.FormCount-1 do
  begin
    NewItem := TMenuItem.Create(MainMenu1.Items[1]);
```

```
    NewItem.Caption := Screen.Forms[I].Name + 'Item';
    NewItem.Name := Screen.Forms[I].Name;
    MainMenu1.Items[1].Add(NewItem);
   end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I : Integer;
begin
  // Count down, not up!
  for  I := MainMenu1.Items[1].Count - 1 downto 0 do
  begin
    MainMenu1.Items[1].Items[I].Free;
  end;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
```

```
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

TMenuItem (see page 16102)

Items (see page 16149)

95.1.3.5.7 TMenu.OwnerDraw Property

Specifies whether the menu items are drawn by the application.

Pascal

```
property OwnerDraw: Boolean;
```

C++

```
__property Boolean OwnerDraw;
```

Description

Set OwnerDraw to true to indicate that the application will draw menu items in an OnDrawItem event handler. When OwnerDraw is true, menu items receive an OnMeasureItem and an OnDrawItem event when they need to be rendered on screen.

Set OwnerDraw to false to accept the default drawing. When OwnerDraw is false, menu items display the caption with an optional image to the left (If the Images (see page 16083) and ImageIndex properties are set).

Owner-draw menus are useful for displaying the value that appears when selecting a menu item. For example, in a menu that allows users to choose a color, an owner-draw menu could display rectangles drawn in the designated color, rather than color names.

See Also

OnMeasureItem (see page 16160)

OnDrawItem (see page 16159)

Images (see page 16083)

95.1.3.5.8 TMenu.ParentBiDiMode Property

Indicates whether the BiDiMode (see page 16083) property is inherited from the control that uses the menu.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

By default, ParentBiDiMode is true, indicating that the layout of menu items follows the BiDiMode (see page 16083) setting of the associated form or control. Change ParentBiDiMode to false to override the default BiDiMode (see page 16083) setting and control the menu's layout independently.

See Also

BiDiMode (see page 16083)

95.1.3.5.9 TMenu.WindowHandle Property

Provides access to the handle of the window that uses the menu.

Pascal

```
property WindowHandle: HWND;
```

C++

```
__property HWND WindowHandle;
```

Description

Read WindowHandle to get the window handle of the control that receives WM_COMMAND messages when the user clicks on an item in the menu. When menu items are selected, use this handle in an OnClick event handler to send messages to the control that uses the menu or to make Windows API calls that affect the control.

Set WindowHandle to associate the menu with a control.

See Also

DispatchCommand (see page 16074)

Command (see page 16141)

OnClick (see page 16154)

95.1.3.6 TMenu Events

The events of the TMenu class are listed here.

Protected Events

	Name	Description
	OnChange (see page 16090)	Occurs when the menu changes.

95.1.3.6.1 TMenu.OnChange Event

Occurs when the menu changes.

Pascal

```
property OnChange: TMenuChangeEvent;
```

C++

```
__property TMenuChangeEvent OnChange;
```

Description

Write an OnChange event handler to respond to changes in the menu's structure. OnChange occurs when the menu is loaded into memory, and in response to property changes that affect the menu's structure.

See Also

Items (see page 16084)

95.1.4 TMenuActionLink Class

TMenuActionLink links menu item clients to the corresponding menu action.

Class Hierarchy

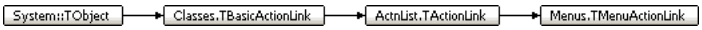

Pascal

```
TMenuActionLink = class(TActionLink);
```

C++

```
class TMenuActionLink : public TActionLink;
```

File

Menus

Description

TMenuActionLink is designed to link actions to menu item clients. TMenuActionLink overrides methods to determine whether the action and client are linked, and to ensure that action events are appropriate for properties of and events of menu item clients.

TMenuActionLink overrides methods to support linking the following properties and events with menu item client properties or events, most of which have the same name:

Caption

Checked

Enabled

HelpContext

Hint

ImageIndex

ShortCut (see page 16193)

Visible

OnExecute

95.1.4.1 TMenuActionLink Members

The following tables list the members exposed by TMenuActionLink.

Protected Fields

	Name	Description
	FAction (see page 2722)	This is FAction, a member of class TBasicActionLink.
	FClient (see page 16093)	This is FClient, a member of class TMenuActionLink.

Protected Methods

	Name	Description
	AssignClient (see page 16093)	Assigns a client menu item that is linked to the associated action.
	Change (see page 2723)	This is Change, a member of class TBasicActionLink.
	IsAutoCheckLinked (see page 16094)	Indicates whether the AutoCheck state for the action and the client are linked.
	IsCaptionLinked (see page 16094)	Indicates whether the caption for the action and the client are linked.
	IsCheckedLinked (see page 16095)	Indicates whether the checked state for the action and the client are linked.
	IsEnabledLinked (see page 16095)	Indicates whether the enabled state for the action and the client are linked.
	IsGroupIndexLinked (see page 16095)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsHelpContextLinked (see page 16096)	Indicates whether the HelpContext properties for the action and client controls are linked.
	IsHelpLinked (see page 271)	Indicates whether the help properties for the action and client are linked.
	IsHintLinked (see page 16096)	Indicates whether the Hint property for the action and the client are linked.
	IsImageIndexLinked (see page 16097)	Indicates whether the image index for the action and the client are linked.
	IsOnExecuteLinked (see page 16097)	Indicates whether the OnExecute event handler for the action and the client are linked.
	IsShortCutLinked (see page 16097)	Indicates whether the ShortCut (see page 16193) property for the action and the client are linked.
	IsVisibleLinked (see page 16098)	Indicates whether the visible state for the action and the client are linked.
	SetAction (see page 2724)	This is SetAction, a member of class TBasicActionLink.
	SetAutoCheck (see page 16098)	Sets the AutoCheck property of a menu item if the client is linked to the action for this property.
	SetCaption (see page 16099)	Sets the caption of a menu item if the client is linked to the action for this property.
	SetChecked (see page 16099)	Sets the checked state of a menu item if the client is linked to the action for this property.
	SetEnabled (see page 16099)	Sets the enabled state of a menu item if the client is linked to the action for this property.
	SetGroupIndex (see page 275)	Introduces an interface for setting the GroupIndex property of a client.
	SetHelpContext (see page 16100)	Sets the Help context of a menu item that is the client of this action link.
	SetHelpKeyword (see page 276)	Introduces an interface for setting the help keyword of a client.
	SetHelpType (see page 277)	Introduces an interface for setting the HelpType property of a client.
	SetHint (see page 16100)	Sets the hint of a menu item if the client is linked to the action for this property.
	SetImageIndex (see page 16100)	Sets the image index of a menu item if the client is linked to the action for this property.
	SetOnExecute (see page 16101)	Sets the execution event handler for a menu item if the client is linked to the action for this property.
	SetShortCut (see page 16101)	Sets the shortcut of a menu item if the client is linked to the action for this property.
	SetVisible (see page 16102)	Sets the visible state of a menu item if the client is linked to the action for this property.

Public Events

	Name	Description
	OnChange (see page 2725)	Occurs when an OnChange event is triggered on the action.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2721)	Instantiates and initializes a TBasicActionLink (see page 2720) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2722)	Destroys an instance of an action link object.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 2723)	Invokes the action's Execute method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Update (see page 2724)	Invokes the action's Update method.

Public Properties

	Name	Description
	Action (see page 2725)	Indicates which action the link connects to its client.

95.1.4.2 TMenuActionLink Fields

The fields of the TMenuActionLink class are listed here.

Protected Fields

	Name	Description
	FClient (see page 16093)	This is FClient, a member of class TMenuActionLink.

95.1.4.2.1 TMenuActionLink.FClient Field

Pascal

```
FClient: TMenuItem;
```

C++

```
TMenuItem FClient;
```

Description

This is FClient, a member of class TMenuActionLink.

95.1.4.3 TMenuActionLink Methods

The methods of the TMenuActionLink class are listed here.

Protected Methods

	Name	Description
	AssignClient (see page 16093)	Assigns a client menu item that is linked to the associated action.
	IsAutoCheckLinked (see page 16094)	Indicates whether the AutoCheck state for the action and the client are linked.
	IsCaptionLinked (see page 16094)	Indicates whether the caption for the action and the client are linked.
	IsCheckedLinked (see page 16095)	Indicates whether the checked state for the action and the client are linked.
	IsEnabledLinked (see page 16095)	Indicates whether the enabled state for the action and the client are linked.
	IsGroupIndexLinked (see page 16095)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsHelpContextLinked (see page 16096)	Indicates whether the HelpContext properties for the action and client controls are linked.
	IsHintLinked (see page 16096)	Indicates whether the Hint property for the action and the client are linked.
	IsImageIndexLinked (see page 16097)	Indicates whether the image index for the action and the client are linked.
	IsOnExecuteLinked (see page 16097)	Indicates whether the OnExecute event handler for the action and the client are linked.
	IsShortCutLinked (see page 16097)	Indicates whether the ShortCut (see page 16193) property for the action and the client are linked.
	IsVisibleLinked (see page 16098)	Indicates whether the visible state for the action and the client are linked.
	SetAutoCheck (see page 16098)	Sets the AutoCheck property of a menu item if the client is linked to the action for this property.
	SetCaption (see page 16099)	Sets the caption of a menu item if the client is linked to the action for this property.
	SetChecked (see page 16099)	Sets the checked state of a menu item if the client is linked to the action for this property.
	SetEnabled (see page 16099)	Sets the enabled state of a menu item if the client is linked to the action for this property.
	SetHelpContext (see page 16100)	Sets the Help context of a menu item that is the client of this action link.
	SetHint (see page 16100)	Sets the hint of a menu item if the client is linked to the action for this property.
	SetImageIndex (see page 16100)	Sets the image index of a menu item if the client is linked to the action for this property.
	SetOnExecute (see page 16101)	Sets the execution event handler for a menu item if the client is linked to the action for this property.
	SetShortCut (see page 16101)	Sets the shortcut of a menu item if the client is linked to the action for this property.
	SetVisible (see page 16102)	Sets the visible state of a menu item if the client is linked to the action for this property.

95.1.4.3.1 TMenuActionLink.AssignClient Method

Assigns a client menu item that is linked to the associated action.

Pascal

```
procedure AssignClient(AClient: TObject); override;
```

C++

```
virtual __fastcall AssignClient(TObject * AClient);
```

Description

TMenuActionLink (see page 16090) overrides AssignClient to ensure that the client for the link is a menu item.

See Also

Action (see page 16133)

Create (see page 2721)

95.1.4.3.2 TMenuActionLink.IsAutoCheckLinked Method

Indicates whether the AutoCheck state for the action and the client are linked.

Pascal

```
function IsAutoCheckLinked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsAutoCheckLinked();
```

Description

IsAutoCheckLinked indicates whether the AutoCheck property is linked to the appropriate client property. IsAutoCheckLinked returns true if the client requests that the action control this property, false otherwise.

See Also

AutoCheck (see page 308)

AutoCheck (see page 16134)

SetAutoCheck (see page 16098)

95.1.4.3.3 TMenuActionLink.IsCaptionLinked Method

Indicates whether the caption for the action and the client are linked.

Pascal

```
function IsCaptionLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCaptionLinked();
```

Description

IsCaptionLinked indicates whether the Caption property is linked to the appropriate client property. IsCaptionLinked returns true if the client requests that the action control this property, false otherwise.

See Also

Caption (see page 308)

Caption (see page 16136)

SetCaption (see page 16099)

95.1.4.3.4 TMenuActionLink.IsCheckedLinked Method

Indicates whether the checked state for the action and the client are linked.

Pascal

```
function IsCheckedLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCheckedLinked();
```

Description

IsCheckedLinked indicates whether the Checked property is linked to the appropriate client property. IsCheckedLinked returns true if the client requests that the action control this property, false otherwise.

See Also

Checked (see page 308)

Checked (see page 16140)

SetChecked (see page 16099)

95.1.4.3.5 TMenuActionLink.IsEnabledLinked Method

Indicates whether the enabled state for the action and the client are linked.

Pascal

```
function IsEnabledLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsEnabledLinked();
```

Description

IsEnabledLinked indicates whether the Enabled property is linked to the appropriate client property. IsEnabledLinked returns true if the client requests that the action control this property, false otherwise.

See Also

Enabled (see page 310)

Enabled (see page 16145)

SetEnabled (see page 16099)

95.1.4.3.6 TMenuActionLink.IsGroupIndexLinked Method

Indicates whether the GroupIndex property for the action controls client behavior.

Pascal

```
function IsGroupIndexLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsGroupIndexLinked();
```

Description

TActionLink descendants can call IsGroupIndexLinked to determine whether they need to propagate changes to the GroupIndex property of the action so that they are reflected by the client object. When IsGroupIndexLinked returns true, the action controls

the GroupIndex property: Changes to the GroupIndex property of the action are reflected in the client object. When IsGroupIndexLinked returns false, the client object is not influenced by the GroupIndex property of its action.

As implemented in TActionLink, IsGroupIndexLinked merely checks that the Action property specifies a descendant of TCustomAction, and if so, returns true.

See Also

GroupIndex (see page 311)

SetGroupIndex (see page 275)

Action (see page 2725)

95.1.4.3.7 TMenuActionLink.IsHelpContextLinked Method

Indicates whether the HelpContext properties for the action and client controls are linked.

Pascal

```
function IsHelpContextLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsHelpContextLinked();
```

Description

TActionLink descendants can call IsHelpContextLinked to determine whether they need to propagate changes to the HelpContext property of the action so that they are reflected by the client object. When IsHelpContextLinked returns true, the action controls the HelpContext property: Changes to the HelpContext property of the action are reflected in the client object. When IsHelpContextLinked returns false, the client object is not influenced by the HelpContext property of its action.

As implemented in TActionLink, IsHelpContextLinked merely checks that the Action property specifies a descendant of TCustomAction, and if so, returns true.

See Also

HelpContext (see page 311)

SetHelpContext (see page 276)

Action (see page 2725)

IsHelpLinked (see page 271)

95.1.4.3.8 TMenuActionLink.IsHintLinked Method

Indicates whether the Hint property for the action and the client are linked.

Pascal

```
function IsHintLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsHintLinked();
```

Description

IsHintLinked indicates whether the Hint property is linked to the appropriate client property. IsHintLinked returns true if the client requests that the action control this property, false otherwise.

See Also

Hint (see page 312)

Hint (see page 16148)

SetHint (see page 16100)

95.1.4.3.9 TMenuActionLink.IsImageIndexLinked Method

Indicates whether the image index for the action and the client are linked.

Pascal

```
function IsImageIndexLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsImageIndexLinked();
```

Description

IsImageIndexLinked indicates whether the ImageIndex property is linked to the appropriate client property. IsImageIndexLinked returns true if the client requests that the action control this property, false otherwise.

See Also

ImageIndex (see page 313)

ImageIndex (see page 16148)

SetImageIndex (see page 16100)

95.1.4.3.10 TMenuActionLink.IsOnExecuteLinked Method

Indicates whether the OnExecute event handler for the action and the client are linked.

Pascal

```
function IsOnExecuteLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsOnExecuteLinked();
```

Description

IsOnExecuteLinked indicates whether the OnExecute event handler is linked to the appropriate client event. IsOnExecuteLinked returns true if the client requests that the action control this event, false otherwise.

See Also

OnExecute (see page 2716)

OnClick (see page 16154)

SetOnExecute (see page 16101)

95.1.4.3.11 TMenuActionLink.IsShortCutLinked Method

Indicates whether the ShortCut (see page 16193) property for the action and the client are linked.

Pascal

```
function IsShortCutLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsShortCutLinked();
```

Description

IsShortCutLinked indicates whether the ShortCut (see page 16193) property is linked to the appropriate client property. IsVisibleLinked (see page 16098) returns true if the client requests that the action control this property, false otherwise.

See Also

ShortCut (see page 314)

ShortCut (see page 16152)

SetShortCut (see page 16101)

95.1.4.3.12 TMenuActionLink.IsVisibleLinked Method

Indicates whether the visible state for the action and the client are linked.

Pascal

```
function IsVisibleLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsVisibleLinked();
```

Description

IsVisibleLinked indicates whether the Visible property is linked to the appropriate client property. IsVisibleLinked returns true if the client requests that the action control this property, false otherwise.

See Also

Visible (see page 314)

Visible (see page 16153)

SetVisible (see page 16102)

95.1.4.3.13 TMenuActionLink.SetAutoCheck Method

Sets the AutoCheck property of a menu item if the client is linked to the action for this property.

Pascal

```
procedure SetAutoCheck(Value: Boolean); override;
```

C++

```
virtual __fastcall SetAutoCheck(Boolean Value);
```

Description

SetAutoCheck sets the AutoCheck property of a menu item, and is called only if the client is linked for this event. Value specifies the AutoCheck new value of AutoCheck.

See Also

AutoCheck (see page 308)

AutoCheck (see page 16134)

IsAutoCheckLinked (see page 16094)

95.1.4.3.14 TMenuActionLink.SetCaption Method

Sets the caption of a menu item if the client is linked to the action for this property.

Pascal

```
procedure SetCaption(const Value: string); override;
```

C++

```
virtual __fastcall SetCaption(const AnsiString Value);
```

Description

SetCaption sets the caption of a menu item, and is called only if the client is linked for this event. Value is the string to which the caption should be set.

See Also

Caption (see page 308)

Caption (see page 16136)

IsCaptionLinked (see page 16094)

95.1.4.3.15 TMenuActionLink.SetChecked Method

Sets the checked state of a menu item if the client is linked to the action for this property.

Pascal

```
procedure SetChecked(Value: Boolean); override;
```

C++

```
virtual __fastcall SetChecked(Boolean Value);
```

Description

SetChecked sets the checked state of a menu item, and is called only if the client is linked for this event. Value specifies whether the checked state should be set.

See Also

Checked (see page 308)

Checked (see page 16140)

IsCheckedLinked (see page 16095)

95.1.4.3.16 TMenuActionLink.SetEnabled Method

Sets the enabled state of a menu item if the client is linked to the action for this property.

Pascal

```
procedure SetEnabled(Value: Boolean); override;
```

C++

```
virtual __fastcall SetEnabled(Boolean Value);
```

Description

SetEnabled sets the enabled state of a menu item, and is called only if the client is linked for this event. Value specifies whether the enabled state should be set.

See Also

Enabled (see page 310)

Enabled (see page 16145)

IsEnabledLinked (see page 16095)

95.1.4.3.17 TMenuActionLink.SetHelpContext Method

Sets the Help context of a menu item that is the client of this action link.

Pascal

```
procedure SetHelpContext(Value: THelpContext); override;
```

C++

```
virtual __fastcall SetHelpContext(THelpContext Value);
```

Description

SetHelpContext sets the help context of a client menu item, and is called only if the client is linked for this property. Value specifies what the help context should be.

See Also

HelpContext (see page 311)

HelpContext (see page 16148)

IsHelpLinked

95.1.4.3.18 TMenuActionLink.SetHint Method

Sets the hint of a menu item if the client is linked to the action for this property.

Pascal

```
procedure SetHint(const Value: string); override;
```

C++

```
virtual __fastcall SetHint(const AnsiString Value);
```

Description

SetHint sets the hint of a menu item, and is called only if the client is linked for this event. Value is the string that to which the hint should be set.

See Also

Hint (see page 312)

Hint (see page 16148)

IsHintLinked (see page 16096)

95.1.4.3.19 TMenuActionLink.SetImageIndex Method

Sets the image index of a menu item if the client is linked to the action for this property.

Pascal

```
procedure SetImageIndex(Value: Integer); override;
```

C++

```
virtual __fastcall SetImageIndex(int Value);
```

Description

SetImageIndex sets the image index of a menu item, and is called only if the client is linked for this event. Value specifies what the image index should be.

See Also

ImageIndex (see page 313)

ImageIndex (see page 16148)

IsImageIndexLinked (see page 16097)

95.1.4.3.20 TMenuActionLink.SetOnExecute Method

Sets the execution event handler for a menu item if the client is linked to the action for this property.

Pascal

```
procedure SetOnExecute(Value: TNotifyEvent); override;
```

C++

```
virtual __fastcall SetOnExecute(TNotifyEvent Value);
```

Description

SetOnExecute sets the appropriate execution event handler of a menu item, and is called only if the client is linked for this event. Value specifies the procedure that should be used for this event.

See Also

OnExecute (see page 2716)

OnClick (see page 16154)

IsOnExecuteLinked (see page 16097)

95.1.4.3.21 TMenuActionLink.SetShortCut Method

Sets the shortcut of a menu item if the client is linked to the action for this property.

Pascal

```
procedure SetShortCut(Value: TShortCut); override;
```

C++

```
virtual __fastcall SetShortCut(TShortCut Value);
```

Description

SetShortCut sets the shortcut of a menu item, and is called only if the client is linked for this event. Value specifies what the shortcut should be.

See Also

ShortCut (see page 314)

ShortCut (see page 16152)

IsShortCutLinked (see page 16097)

95.1.4.3.22 TMenuActionLink.SetVisible Method

Sets the visible state of a menu item if the client is linked to the action for this property.

Pascal

```
procedure SetVisible(Value: Boolean); override;
```

C++

```
virtual __fastcall SetVisible(Boolean Value);
```

Description

SetVisible sets the visible state of a menu item, and is called only if the client is linked for this event. Value specifies whether the visible state should be set.

See Also

Visible (see page 314)

Visible (see page 16153)

IsVisibleLinked (see page 16098)

95.1.5 TMenuItem Class

TMenuItem describes the properties of an item in a menu.

Class Hierarchy

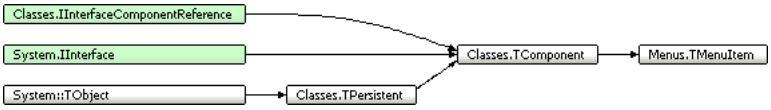

Pascal

```
TMenuItem = class(TComponent);
```

C++

```
class TMenuItem : public TComponent;
```

File

Menus

Description

Use TMenuItem to specify the appearance and behavior of an item in a menu. Each TMainMenu (see page 16056) or TPopupMenu (see page 16169) component can contain multiple menu items. When a menu is defined at design time, the Menu Designer automatically creates menu item objects for each command on the menu.

See Also

TMainMenu (see page 16056)

TPopupMenu (see page 16169)

NewItem (see page 16188)

NewLine (see page 16189)

NewSubMenu (see page 16192)

95.1.5.1 TMenuItem Members

The following tables list the members exposed by TMenuItem.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 16110)	This is ActionChange, a member of class TMenuItem.
V	AdvancedDrawItem (see page 16115)	This is AdvancedDrawItem, a member of class TMenuItem.
V	AssignTo (see page 16115)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 16116)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoDrawText (see page 16117)	This is DoDrawText, a member of class TMenuItem.
V	DrawItem (see page 16118)	This is DrawItem, a member of class TMenuItem.
	GetActionLinkClass (see page 16118)	This is GetActionLinkClass, a member of class TMenuItem.
	GetAutoHotkeys (see page 16118)	This is GetAutoHotkeys, a member of class TMenuItem.
	GetAutoLineReduction (see page 16119)	This is GetAutoLineReduction, a member of class TMenuItem.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
V	GetChildren (see page 16119)	This is GetChildren, a member of class TMenuItem.
	GetCount (see page 16119)	This is GetCount, a member of class TMenuItem.
	GetHandle (see page 16120)	This is GetHandle, a member of class TMenuItem.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetItem (see page 16120)	This is GetItem, a member of class TMenuItem.
	GetMenuIndex (see page 16121)	This is GetMenuIndex, a member of class TMenuItem.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InsertNewLine (see page 16124)	This is InsertNewLine, a member of class TMenuItem.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 16125)	Initializes the component after the form file has been read into memory.
V	MeasureItem (see page 16126)	This is MeasureItem, a member of class TMenuItem.
V	MenuChanged (see page 16126)	This is MenuChanged, a member of class TMenuItem.
V	Notification (see page 16127)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBreak (see page 16129)	This is SetBreak, a member of class TMenuItem.
	SetCaption (see page 16130)	This is SetCaption, a member of class TMenuItem.
	SetChecked (see page 16130)	This is SetChecked, a member of class TMenuItem.

	SetChildOrder (see page 16130)	This is SetChildOrder, a member of class TMenuItem.
	SetDefault (see page 16130)	This is SetDefault, a member of class TMenuItem.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEnabled (see page 16131)	This is SetEnabled, a member of class TMenuItem.
	SetGroupIndex (see page 16131)	This is SetGroupIndex, a member of class TMenuItem.
	SetImageIndex (see page 16131)	This is SetImageIndex, a member of class TMenuItem.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetMenuIndex (see page 16131)	This is SetMenuIndex, a member of class TMenuItem.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 16131)	This is SetParentComponent, a member of class TMenuItem.
	SetRadioItem (see page 16132)	This is SetRadioItem, a member of class TMenuItem.
	SetShortCut (see page 16132)	This is SetShortCut, a member of class TMenuItem.
	SetVisible (see page 16132)	This is SetVisible, a member of class TMenuItem.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateItems (see page 16132)	This is UpdateItems, a member of class TMenuItem.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 16134)	This is ActionLink, a member of class TMenuItem.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Add (see page 16110)	This is the overview for the Add method overload.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 16115)	Removes and frees all menu items listed in the Items (see page 16149) property.
	Click (see page 16116)	Simulates a mouse click.
	Create (see page 16107)	Creates an instance of TMenuItem (see page 16102).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 16117)	Removes a menu item from the Items (see page 16149) property array.
	Destroy (see page 16108)	Destroys an instance of TMenuItem (see page 16102).

	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Find (see page 16118)	Locates a menu item in the Items (see page 16149) property array given its caption.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 16119)	Returns a TComponent enumerator.
	GetImageList (see page 16120)	Returns the Image list that supplies an image for the menu item.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 16121)	Returns the parent component of the menu item.
	GetParentMenu (see page 16121)	Returns the main menu of which the menu item is part.
V	HasParent (see page 16121)	Indicates that the menu item has a parent that is responsible for loading and saving its data.
	IndexOf (see page 16122)	Returns the position of a menu item within the Items (see page 16149) array.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 16122)	Calls the action link's Update method if the menu item is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 16123)	Inserts a menu item into a specified position in the Items (see page 16149) array.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertNewLineAfter (see page 16124)	Inserts a separator bar after a specified item from the Items (see page 16149) property array.
	InsertNewLineBefore (see page 16124)	Inserts a separator bar before a specified item from the Items (see page 16149) property array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLine (see page 16125)	Indicates whether the menu item represents a separator bar.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewBottomLine (see page 16126)	Inserts a separator bar at the end of the Items (see page 16149) property array.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NewTopLine (see page 16127)	Inserts a separator bar at the beginning of the Items (see page 16149) property array.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Remove (see page 16128)	Removes a menu item from the Items (see page 16149) property array
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RethinkHotkeys (see page 16128)	Adjusts the captions of subitems so that every item has an accelerator and there are no duplicate accelerator keys.
	RethinkLines (see page 16129)	Removes superfluous separator lines from the Items (see page 16149) property array.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	Command (see page 16141)	Specifies the Windows Command ID associated with the menu item.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Count (see page 16141)	Indicates the number of subitems of the menu item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Handle (see page 16147)	Indicates the Windows menu handle of the drop-down menu associated with the menu item.
R	Items (see page 16149)	Lists the menu items in the submenu of the menu item.
	MenuIndex (see page 16150)	Indicates the index of the menu item within its parent menu.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Parent (see page 16151)	Identifies the parent menu item of this menu item.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnAdvancedDrawItem (see page 16154)	Occurs when an owner-draw menu needs to be drawn.
	OnClick (see page 16154)	Occurs when the user clicks menu item.
	OnDrawItem (see page 16159)	Occurs when an owner-draw menu needs to be drawn.
	OnMeasureItem (see page 16160)	Occurs on owner-draw menu items to determine menu item size before the menu is drawn.

Published Properties

	Name	Description
	Action (see page 16133)	Designates the action associated with the menu item.
	AutoCheck (see page 16134)	Indicates whether the menu item's checked state toggles automatically when the item is clicked.
	AutoHotkeys (see page 16134)	Determines whether the accelerator keys for submenu items can be reset automatically.
	AutoLineReduction (see page 16135)	Determines whether redundant separator bars are automatically removed from the submenu.
	Bitmap (see page 16135)	Designates a bitmap that is drawn for the menu item.
	Break (see page 16136)	Determines whether the menu item starts a new column in the menu.
	Caption (see page 16136)	Specifies the text of the menu item.
	Checked (see page 16140)	Specifies whether a check mark should appear beside the Caption (see page 16136).
	Default (see page 16145)	Specifies whether the menu item is invoked when the parent item is double clicked.
	Enabled (see page 16145)	Specifies whether the menu item is enabled.

	GroupIndex (see page 16146)	Identifies the logical group to which the menu item belongs.
	HelpContext (see page 16148)	Specifies the help context ID associated with the menu item.
	Hint (see page 16148)	Specifies the text string that can appear when the user moves the mouse pointer over a menu item.
	ImageIndex (see page 16148)	Indicates which image maintained by the parent menu appears next to the menu item.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RadioItem (see page 16151)	Specifies whether the menu item is mutually exclusive with other menu items in its group.
	ShortCut (see page 16152)	Specifies the key combination users can type to quickly access the menu item.
	SubMenuImages (see page 16152)	Lists the images that can appear beside individual items in a submenu of this menu item.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 16153)	Specifies whether the menu item appears in the menu.

95.1.5.2 TMenuItem.Create Constructor

Creates an instance of TMenuItem (see page 16102).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TMenuItem(TComponent * AOwner);
```

Description

Use Create to create a menu item at runtime. Menu items defined at design time using the Menu designer are created automatically.

AOwner is the component that is responsible for freeing the menu item. It becomes the value of the Owner property.

C++ Examples:

```
/*
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *NewItem;
  // first create the separator
  NewItem = new TMenuItem(this); // If you change the AOwner (this) to MainMenu1->Items[1] and
then you add the new menu you will get "Menu inserted twice"?  Doesn't happen in Delphi.
  NewItem->Name = "Separator";
  NewItem->Caption = "-";
  // add the new item to the Windows menu
  MainMenu1->Items[1].Add(NewItem);
  // now create and add a menu item for each form
  for  (int I = 0; I < Screen->FormCount; I++)
  {
    NewItem = new TMenuItem(this);
    NewItem->Caption = Screen->Forms[I]->Name + "Item";
    NewItem->Name = Screen->Forms[I]->Name;
    MainMenu1->Items[1].Add(NewItem);
  }
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
```

```
{
  // Count down, not up!
  for  (int I = MainMenu1->Items[1].Count - 1; I >= 0; I--)
  {
    delete MainMenu1->Items[1].Items[I];
  }
}
```

Delphi Examples:

```
{
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
}
procedure TForm1.Button1Click(Sender: TObject);
var
  NewItem: TMenuItem;
  I : integer;
begin
  { first create the separator }
  NewItem := TMenuItem.Create(MainMenu1.Items[1]);
  NewItem.Name:= 'Separator';
  NewItem.Caption := '-';
  { add the new item to the Windows menu }
  MainMenu1.Items[1].Add(NewItem);
  { now create and add a menu item for each form }
  for  I := 0 to Screen.FormCount-1 do
  begin
    NewItem := TMenuItem.Create(MainMenu1.Items[1]);
    NewItem.Caption := Screen.Forms[I].Name + 'Item';
    NewItem.Name := Screen.Forms[I].Name;
    MainMenu1.Items[1].Add(NewItem);
   end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I : Integer;
begin
  // Count down, not up!
  for  I := MainMenu1.Items[1].Count - 1 downto 0 do
  begin
    MainMenu1.Items[1].Items[I].Free;
  end;
end;
```

See Also

Owner (see page 2807)

NewItem (see page 16188)

95.1.5.3 TMenuItem.Destroy Destructor

Destroys an instance of TMenuItem (see page 16102).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMenuItem();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TMenuItem (see page 16102) reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

95.1.5.4 TMenuItem Methods

The methods of the TMenuItem class are listed here.

Protected Methods

	Name	Description
	ActionChange (see page 16110)	This is ActionChange, a member of class TMenuItem.
V	AdvancedDrawItem (see page 16115)	This is AdvancedDrawItem, a member of class TMenuItem.
V	AssignTo (see page 16115)	Copies the properties of an object to a destination object.
V	DefineProperties (see page 16116)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoDrawText (see page 16117)	This is DoDrawText, a member of class TMenuItem.
V	DrawItem (see page 16118)	This is DrawItem, a member of class TMenuItem.
	GetActionLinkClass (see page 16118)	This is GetActionLinkClass, a member of class TMenuItem.
	GetAutoHotkeys (see page 16118)	This is GetAutoHotkeys, a member of class TMenuItem.
	GetAutoLineReduction (see page 16119)	This is GetAutoLineReduction, a member of class TMenuItem.
V	GetChildren (see page 16119)	This is GetChildren, a member of class TMenuItem.
	GetCount (see page 16119)	This is GetCount, a member of class TMenuItem.
	GetHandle (see page 16120)	This is GetHandle, a member of class TMenuItem.
	GetItem (see page 16120)	This is GetItem, a member of class TMenuItem.
	GetMenuIndex (see page 16121)	This is GetMenuIndex, a member of class TMenuItem.
	InsertNewLine (see page 16124)	This is InsertNewLine, a member of class TMenuItem.
V	Loaded (see page 16125)	Initializes the component after the form file has been read into memory.
V	MeasureItem (see page 16126)	This is MeasureItem, a member of class TMenuItem.
V	MenuChanged (see page 16126)	This is MenuChanged, a member of class TMenuItem.
V	Notification (see page 16127)	Forwards notification messages to all owned components.
	SetBreak (see page 16129)	This is SetBreak, a member of class TMenuItem.
	SetCaption (see page 16130)	This is SetCaption, a member of class TMenuItem.
	SetChecked (see page 16130)	This is SetChecked, a member of class TMenuItem.
V	SetChildOrder (see page 16130)	This is SetChildOrder, a member of class TMenuItem.
	SetDefault (see page 16130)	This is SetDefault, a member of class TMenuItem.
	SetEnabled (see page 16131)	This is SetEnabled, a member of class TMenuItem.
	SetGroupIndex (see page 16131)	This is SetGroupIndex, a member of class TMenuItem.
	SetImageIndex (see page 16131)	This is SetImageIndex, a member of class TMenuItem.
	SetMenuIndex (see page 16131)	This is SetMenuIndex, a member of class TMenuItem.
V	SetParentComponent (see page 16131)	This is SetParentComponent, a member of class TMenuItem.
	SetRadioItem (see page 16132)	This is SetRadioItem, a member of class TMenuItem.
	SetShortCut (see page 16132)	This is SetShortCut, a member of class TMenuItem.
	SetVisible (see page 16132)	This is SetVisible, a member of class TMenuItem.
	UpdateItems (see page 16132)	This is UpdateItems, a member of class TMenuItem.

Public Methods

	Name	Description
	Add (see page 16110)	This is the overview for the Add method overload.
	Clear (see page 16115)	Removes and frees all menu items listed in the Items (see page 16149) property.

	Click (see page 16116)	Simulates a mouse click.
	Delete (see page 16117)	Removes a menu item from the Items (see page 16149) property array.
	Find (see page 16118)	Locates a menu item in the Items (see page 16149) property array given its caption.
	GetEnumerator (see page 16119)	Returns a TComponent enumerator.
	GetImageList (see page 16120)	Returns the Image list that supplies an image for the menu item.
	GetParentComponent (see page 16121)	Returns the parent component of the menu item.
	GetParentMenu (see page 16121)	Returns the main menu of which the menu item is part.
	HasParent (see page 16121)	Indicates that the menu item has a parent that is responsible for loading and saving its data.
	IndexOf (see page 16122)	Returns the position of a menu item within the Items (see page 16149) array.
	InitiateAction (see page 16122)	Calls the action link's Update method if the menu item is associated with an action link.
	Insert (see page 16123)	Inserts a menu item into a specified position in the Items (see page 16149) array.
	InsertNewLineAfter (see page 16124)	Inserts a separator bar after a specified item from the Items (see page 16149) property array.
	InsertNewLineBefore (see page 16124)	Inserts a separator bar before a specified item from the Items (see page 16149) property array.
	IsLine (see page 16125)	Indicates whether the menu item represents a separator bar.
	NewBottomLine (see page 16126)	Inserts a separator bar at the end of the Items (see page 16149) property array.
	NewTopLine (see page 16127)	Inserts a separator bar at the beginning of the Items (see page 16149) property array.
	Remove (see page 16128)	Removes a menu item from the Items (see page 16149) property array
	RethinkHotkeys (see page 16128)	Adjusts the captions of subitems so that every item has an accelerator and there are no duplicate accelerator keys.
	RethinkLines (see page 16129)	Removes superfluous separator lines from the Items (see page 16149) property array.

95.1.5.4.1 TMenuItem.ActionChange Method

Pascal

```
procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); dynamic;
```

C++

```
__fastcall ActionChange(TObject * Sender, Boolean CheckDefaults);
```

Description

This is ActionChange, a member of class TMenuItem.

95.1.5.4.2 Add Method

This is the overview for the Add method overload.

Overload List

	Name	Description
	TMenuItem.Add (array of TMenuItem) (see page 16110)	Adds one or more menu items to the end of the Items (see page 16149) property array.
	TMenuItem.Add (TMenuItem) (see page 16110)	Adds one or more menu items to the end of the Items (see page 16149) property array.

95.1.5.4.2.1 TMenuItem.Add Method (TMenuItem)

Adds one or more menu items to the end of the Items (see page 16149) property array.

Pascal

```
procedure Add(Item: TMenuItem); overload;
procedure Add(const AItems: array of TMenuItem); overload;
```

C++

```
__fastcall Add(TMenuItem Item);
__fastcall Add(const array of TMenuItem AItems);
```

Description

Use Add to add new menu items to the dropdown menu for this menu item. If this menu item is the value of the Items (see page 16149) property of a main menu or pop-up menu, Add adds the items to the end of the menu.

Specify a single new menu item to add as the value of the Item parameter, or an array of new menu items as the value of the AItems parameter. AItems_Size is the index of the last item in the AItems array (one less than the number of items).

C++ Examples:

```
/*
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *NewItem;
  // first create the separator
  NewItem = new TMenuItem(this); // If you change the AOwner (this) to MainMenu1->Items[1] and
then you add the new menu you will get "Menu inserted twice"?  Doesn't happen in Delphi.
  NewItem->Name = "Separator";
  NewItem->Caption = "-";
  // add the new item to the Windows menu
  MainMenu1->Items[1].Add(NewItem);
  // now create and add a menu item for each form
  for  (int I = 0; I < Screen->FormCount; I++)
  {
    NewItem = new TMenuItem(this);
    NewItem->Caption = Screen->Forms[I]->Name + "Item";
    NewItem->Name = Screen->Forms[I]->Name;
    MainMenu1->Items[1].Add(NewItem);
  }
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  // Count down, not up!
  for  (int I = MainMenu1->Items[1].Count - 1; I >= 0; I--)
  {
    delete MainMenu1->Items[1].Items[I];
  }
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
```

```
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
```

```
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
}
procedure TForm1.Button1Click(Sender: TObject);
var
  NewItem: TMenuItem;
  I : integer;
begin
  { first create the separator }
  NewItem := TMenuItem.Create(MainMenu1.Items[1]);
  NewItem.Name:= 'Separator';
  NewItem.Caption := '-';
  { add the new item to the Windows menu }
  MainMenu1.Items[1].Add(NewItem);
  { now create and add a menu item for each form }
  for  I := 0 to Screen.FormCount-1 do
  begin
    NewItem := TMenuItem.Create(MainMenu1.Items[1]);
    NewItem.Caption := Screen.Forms[I].Name + 'Item';
    NewItem.Name := Screen.Forms[I].Name;
    MainMenu1.Items[1].Add(NewItem);
   end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I : Integer;
begin
  // Count down, not up!
  for  I := MainMenu1.Items[1].Count - 1 downto 0 do
  begin
    MainMenu1.Items[1].Items[I].Free;
  end;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;
```

```
implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

Delete (see page 16117)

Insert (see page 16123)

Items (see page 16149)

Remove (see page 16128)

Items (see page 16084)

NewItem (see page 16188)

NewSubMenu (see page 16192)

95.1.5.4.3 TMenuItem.AdvancedDrawItem Method

Pascal

```
procedure AdvancedDrawItem(ACanvas: TCanvas; ARect: TRect; State: TOwnerDrawState; TopLevel:
Boolean); virtual;
```

C++

```
virtual __fastcall AdvancedDrawItem(TCanvas ACanvas, TRect ARect, TOwnerDrawState State,
Boolean TopLevel);
```

Description

This is AdvancedDrawItem, a member of class TMenuItem.

95.1.5.4.4 TMenuItem.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign (see page 2930)

EConvertError (see page 22724)

95.1.5.4.5 TMenuItem.Clear Method

Removes and frees all menu items listed in the Items (see page 16149) property.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Use Clear to free all the items listed in the Items (see page 16149) property array.

See Also

Items (see page 16149)

Delete (see page 16117)

95.1.5.4.6 TMenuItem.Click Method

Simulates a mouse click.

Pascal

```
procedure Click; virtual;
```

C++

```
virtual __fastcall Click();
```

Description

Click generates an OnClick (see page 16154) event, as if the user had clicked the menu item. Unlike a real mouse click, Click does not generate a WM_COMMAND message to the window associated with the menu.

Component writers can override click to provide a different response when the user selects the menu item.

See Also

Command (see page 16141)

OnClick (see page 16154)

95.1.5.4.7 TMenuItem.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor (see page 2839)

DefineBinaryProperty (see page 2837)

DefineProperties (see page 2931)

DefineProperty (see page 2838)

95.1.5.4.8 TMenuItem.Delete Method

Removes a menu item from the Items (see page 16149) property array.

Pascal

```
procedure Delete(Index: Integer);
```

C++

```
__fastcall Delete(int Index);
```

Description

Call Delete to remove a menu item and all its submenus from a menu. The Index parameter is the index of the item to be deleted, where 0 specifies the first item in the menu, 1 specifies the second item, and so on.

Delete does not free the removed item's memory. To remove an item from the menu and free its memory, call the Free method in the item you want to remove.

To temporarily hide a menu item, without deleting it, use the Visible (see page 16153) property.

See Also

Add (see page 16110)

Insert (see page 16123)

Items (see page 16149)

Remove (see page 16128)

Visible (see page 16153)

Free (see page 22300)

95.1.5.4.9 TMenuItem.DoDrawText Method

Pascal

```
procedure DoDrawText(ACanvas: TCanvas; const ACaption: string; var Rect: TRect; Selected:
Boolean; Flags: Longint);
```

C++

```
__fastcall DoDrawText(TCanvas ACanvas, const AnsiString ACaption, TRect Rect, Boolean
Selected, Longint Flags);
```

Description

This is DoDrawText, a member of class TMenuItem.

95.1.5.4.10 TMenuItem.DrawItem Method

Pascal

```
procedure DrawItem(ACanvas: TCanvas; ARect: TRect; Selected: Boolean); virtual;
```

C++

```
virtual __fastcall DrawItem(TCanvas ACanvas, TRect ARect, Boolean Selected);
```

Description

This is DrawItem, a member of class TMenuItem.

95.1.5.4.11 TMenuItem.Find Method

Locates a menu item in the Items (see page 16149) property array given its caption.

Pascal

```
function Find(ACaption: string): TMenuItem;
```

C++

```
__fastcall TMenuItem Find(AnsiString ACaption);
```

Description

Use Find to locate the menu item in the Items (see page 16149) property array with the caption specified by the ACaption parameter. Find ignores accelerator characters when comparing item captions to the ACaption parameter.

Find returns the first menu item in the Items (see page 16149) property array with the specified caption. If no item is found, Find returns nil (Delphi) or NULL (C++).

See Also

Items (see page 16149)

Caption (see page 16136)

95.1.5.4.12 TMenuItem.GetActionLinkClass Method

Pascal

```
function GetActionLinkClass: TMenuActionLinkClass; dynamic;
```

C++

```
__fastcall TMenuActionLinkClass GetActionLinkClass();
```

Description

This is GetActionLinkClass, a member of class TMenuItem.

95.1.5.4.13 TMenuItem.GetAutoHotkeys Method

Pascal

```
function GetAutoHotkeys: Boolean;
```

C++

```
__fastcall Boolean GetAutoHotkeys();
```

Description

This is GetAutoHotkeys, a member of class TMenuItem.

95.1.5.4.14 TMenuItem.GetAutoLineReduction Method

Pascal

```
function GetAutoLineReduction: Boolean;
```

C++

```
__fastcall Boolean GetAutoLineReduction();
```

Description

This is GetAutoLineReduction, a member of class TMenuItem.

95.1.5.4.15 TMenuItem.GetChildren Method

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

This is GetChildren, a member of class TMenuItem.

95.1.5.4.16 TMenuItem.GetCount Method

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

This is GetCount, a member of class TMenuItem.

95.1.5.4.17 TMenuItem.GetEnumerator Method

Returns a TComponent enumerator.

Pascal

```
function GetEnumerator: TMenuItemEnumerator;
```

C++

```
__fastcall TMenuItemEnumerator GetEnumerator();
```

Description

GetEnumerator returns a TComponentEnumerator reference, which enumerates the components contained within a specified containing component.

To process all these sub components, call the TComponentEnumerator GetCurrent method within a While MoveNext do loop.

See Also

TComponentEnumerator (see page 2812)

95.1.5.4.18 TMenuItem.GetHandle Method

Pascal

```
function GetHandle: HMENU;
```

C++

```
__fastcall HMENU GetHandle();
```

Description

This is GetHandle, a member of class TMenuItem.

95.1.5.4.19 TMenuItem.GetImageList Method

Returns the Image list that supplies an image for the menu item.

Pascal

```
function GetImageList: TCustomImageList;
```

C++

```
__fastcall TCustomImageList GetImageList();
```

Description

Use GetImageList to access the list of images that contains the image for this menu item. The ImageIndex (see page 16148) property identifies a specific image from the list that GetImageList returns.

GetImageList first checks the Parent (see page 16151) of this menu item for a SubMenuImages (see page 16152) property. If the parent menu item does not maintain a list of images, it checks that menu item's parent, and so on until reaching a top-level menu item (a menu item where Parent (see page 16151) is not set). If none of the parent menu items has a SubMenuImages (see page 16152) property, GetImageList checks the top-level menu for an Images property.

GetImageList returns the first image list it finds. If it doesn't find an image list, GetImageList returns nil (Delphi) or NULL (C++).

See Also

ImageIndex (see page 16148)

SubMenuImages (see page 16152)

Images (see page 16083)

Parent (see page 16151)

95.1.5.4.20 TMenuItem.GetItem Method

Pascal

```
function GetItem(Index: Integer): TMenuItem;
```

C++

```
__fastcall TMenuItem GetItem(int Index);
```

Description

This is GetItem, a member of class TMenuItem.

95.1.5.4.21 TMenuItem.GetMenuIndex Method

Pascal

```
function GetMenuIndex: Integer;
```

C++

```
__fastcall int GetMenuIndex();
```

Description

This is GetMenuIndex, a member of class TMenuItem.

95.1.5.4.22 TMenuItem.GetParentComponent Method

Returns the parent component of the menu item.

Pascal

```
function GetParentComponent: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetParentComponent();
```

Description

GetParentComponent is called by the streaming system that loads and saves components. It ensures that objects are loaded and saved with their parent objects. GetParentComponent is overridden in descendant classes to return the appropriate parent for their particular type of component. TMenuItem (see page 16102) overrides it to identify its associated menu, if it has one, as its parent.

95.1.5.4.23 TMenuItem.GetParentMenu Method

Returns the main menu of which the menu item is part.

Pascal

```
function GetParentMenu: TMenu;
```

C++

```
__fastcall TMenu GetParentMenu();
```

Description

GetParentMenu returns the menu that contains the menu item.

See Also

Free (see page 22300)

95.1.5.4.24 TMenuItem.HasParent Method

Indicates that the menu item has a parent that is responsible for loading and saving its data.

Pascal

```
function HasParent: Boolean; override;
```

C++

```
virtual __fastcall Boolean HasParent();
```

Description

TMenuItem (see page 16102) overrides HasParent to return true, indicating that menu items have a parent. Applications seldom call the HasParent method. It is used by the streaming system that loads and saves forms to determine when another object is responsible for writing a component to a stream.

See Also

Parent (see page 16151)

HasParent

95.1.5.4.25 TMenuItem.IndexOf Method

Returns the position of a menu item within the Items (see page 16149) array.

Pascal

```
function IndexOf(Item: TMenuItem): Integer;
```

C++

```
__fastcall int IndexOf(TMenuItem Item);
```

Description

Call IndexOf to locate a menu item in the submenu headed by this menu item. IndexOf returns 0 for the first menu item, 1 for the second menu item, and so on. If the menu item is not in the submenu, IndexOf returns -1.

Note: IndexOf may not correspond to the position of the item in the menu the user sees if the menu includes some items that are not visible.

See Also

Items (see page 16149)

MenuIndex (see page 16150)

Visible (see page 16153)

95.1.5.4.26 TMenuItem.InitiateAction Method

Calls the action link's Update method if the menu item is associated with an action link.

Pascal

```
procedure InitiateAction; virtual;
```

C++

```
virtual __fastcall InitiateAction();
```

Description

InitiateAction calls the action link, if there is one for the menu item, to update the association between the action and the menu item.

See Also

Items (see page 16149)

MenuIndex (see page 16150)

Visible (see page 16153)

95.1.5.4.27 TMenuItem.Insert Method

Inserts a menu item into a specified position in the Items (see page 16149) array.

Pascal

```
procedure Insert(Index: Integer; Item: TMenuItem);
```

C++

```
__fastcall Insert(int Index, TMenuItem Item);
```

Description

Use Insert to insert a new menu item to the dropdown menu for this menu item. If this menu item is the value of the Items (see page 16149) property of a main menu or pop-up menu, Insert inserts an item into that menu. Specify the new menu item to add as the value of the Item parameter, and the position in the Items (see page 16149) array as the value of the Index parameter.

Delphi Examples:

```
{
This code inserts a menu item after the first item in a menu
named FileMenu:
}
procedure TForm1.InsertMenuItemClick(Sender: TObject);
var
  NewItem: TMenuItem;
begin
  NewItem := TMenuItem.Create(FileMenu);
  try
    NewItem.Caption := 'Do this';
    FileMenu.Insert(1, NewItem);
  except
    NewItem.Free; // Throw away the menu item if there is an error.
    raise; // reraise the exception
  end;
  // Don't free the menu item on success.  Need it for the menu!
end;

// Free all the menu items that we have created dynamically.
procedure TForm1.FormDestroy(Sender: TObject);
var
  I : Integer;
begin
  for I := FileMenu.Count - 1 downto 0 do  // Index backwards because Count changes as you
remove items.
    if FileMenu.Items[I].Name = '' then  // Items created dynamically have not name.
      FileMenu.Items[I].Free;
end;
```

See Also

Add (see page 16110)

Delete (see page 16117)

Items (see page 16149)

Remove (see page 16128)

Items (see page 16084)

95.1.5.4.28 TMenuItem.InsertNewLine Method

Pascal

```
function InsertNewLine(ABefore: Boolean; AItem: TMenuItem): Integer;
```

C++

```
__fastcall int InsertNewLine(Boolean ABefore, TMenuItem AItem);
```

Description

This is InsertNewLine, a member of class TMenuItem.

95.1.5.4.29 TMenuItem.InsertNewLineAfter Method

Inserts a separator bar after a specified item from the Items (see page 16149) property array.

Pascal

```
function InsertNewLineAfter(AItem: TMenuItem): Integer;
```

C++

```
__fastcall int InsertNewLineAfter(TMenuItem AItem);
```

Description

Use InsertNewLineAfter to insert a separator bar after the menu item specified by AItem. If AItem is not an item in the Items (see page 16149) property array, InsertNewLineAfter raises an exception.

InsertNewLineAfter returns the index of the newly inserted separator item in the Items (see page 16149) property array.

Tip: When inserting separator bars at runtime, it is a good idea to set AutoLineReduction (see page 16135) to true so that superfluous separators can be automatically removed.

See Also

Insert (see page 16123)

InsertNewLineBefore (see page 16124)

Items (see page 16149)

NewBottomLine (see page 16126)

NewTopLine (see page 16127)

AutoLineReduction (see page 16135)

95.1.5.4.30 TMenuItem.InsertNewLineBefore Method

Inserts a separator bar before a specified item from the Items (see page 16149) property array.

Pascal

```
function InsertNewLineBefore(AItem: TMenuItem): Integer;
```

C++

```
__fastcall int InsertNewLineBefore(TMenuItem AItem);
```

Description

Use InsertNewLineBefore to insert a separator bar before the menu item specified by AItem. If AItem is not an item in the Items (see page 16149) property array, InsertNewLineBefore raises an exception.

InsertNewLineBefore returns the index of the newly inserted separator item in the Items (see page 16149) property array.

Tip: When inserting separator bars at runtime, it is a good idea to set AutoLineReduction (see page 16135) to true so that superfluous separators can be automatically removed.

See Also

Insert (see page 16123)

InsertNewLineAfter (see page 16124)

Items (see page 16149)

NewBottomLine (see page 16126)

NewTopLine (see page 16127)

AutoLineReduction (see page 16135)

95.1.5.4.31 TMenuItem.IsLine Method

Indicates whether the menu item represents a separator bar.

Pascal

```
function IsLine: Boolean;
```

C++

```
__fastcall Boolean IsLine();
```

Description

Call IsLine to determine whether the menu item is a separator bar. IsLine checks the Caption (see page 16136) of the menu item and returns true if the value of Caption (see page 16136) is '-'. Menu items with this Caption (see page 16136) appear as separator bars.

See Also

Caption (see page 16136)

95.1.5.4.32 TMenuItem.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState (see page 2804)

Create (see page 2766)

TReader (see page 2933)

95.1.5.4.33 TMenuItem.MeasureItem Method

Pascal

```
procedure MeasureItem(ACanvas: TCanvas; var Width: Integer; var Height: Integer); virtual;
```

C++

```
virtual __fastcall MeasureItem(TCanvas ACanvas, int Width, int Height);
```

Description

This is MeasureItem, a member of class TMenuItem.

95.1.5.4.34 TMenuItem.MenuChanged Method

Pascal

```
procedure MenuChanged(Rebuild: Boolean); virtual;
```

C++

```
virtual __fastcall MenuChanged(Boolean Rebuild);
```

Description

This is MenuChanged, a member of class TMenuItem.

95.1.5.4.35 TMenuItem.NewBottomLine Method

Inserts a separator bar at the end of the Items (see page 16149) property array.

Pascal

```
function NewBottomLine: Integer;
```

C++

```
__fastcall int NewBottomLine();
```

Description

Use NewBottomLine to insert a separator bar at the end of the items property array. This is useful when building up submenus dynamically at runtime.

NewBottomLine returns the index of the newly inserted separator item in the Items (see page 16149) property array.

Tip: When inserting separator bars at runtime, it is a good idea to set AutoLineReduction (see page 16135) to true so that superfluous separators can be automatically removed.

See Also

Add (see page 16110)

InsertNewLineBefore (see page 16124)

Items (see page 16149)

InsertNewLineAfter (see page 16124)

NewTopLine (see page 16127)

AutoLineReduction (see page 16135)

95.1.5.4.36 TMenuItem.NewTopLine Method

Inserts a separator bar at the beginning of the Items (see page 16149) property array.

Pascal

```
function NewTopLine: Integer;
```

C++

```
__fastcall int NewTopLine();
```

Description

Use NewTopLine to insert a separator bar at the beginning of the items property array. This is useful when building up submenus dynamically at runtime.

NewTopLine returns the index of the newly inserted separator item in the Items (see page 16149) property array.

Tip: When inserting separator bars at runtime, it is a good idea to set AutoLineReduction (see page 16135) to true so that superfluous separators can be automatically removed.

See Also

Add (see page 16110)

InsertNewLineBefore (see page 16124)

Items (see page 16149)

InsertNewLineAfter (see page 16124)

NewBottomLine (see page 16126)

AutoLineReduction (see page 16135)

95.1.5.4.37 TMenuItem.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification

to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

95.1.5.4.38 TMenuItem.Remove Method

Removes a menu item from the Items (see page 16149) property array

Pascal

```
procedure Remove(Item: TMenuItem);
```

C++

```
__fastcall Remove(TMenuItem Item);
```

Description

Call Remove to remove a menu item and all its submenus from a menu. The Item parameter is the item to be removed. If Item is not a menu item in the Items (see page 16149) array, Remove raises an exception.

Remove does not free the item's memory. To remove an item from the menu and free its memory, call the Free method in the item you want to remove.

To temporarily hide a menu item, without deleting it, use the Visible (see page 16153) property.

See Also

Add (see page 16110)

Delete (see page 16117)

EMenuError (see page 16055)

Insert (see page 16123)

Items (see page 16149)

Visible (see page 16153)

95.1.5.4.39 TMenuItem.RethinkHotkeys Method

Adjusts the captions of subitems so that every item has an accelerator and there are no duplicate accelerator keys.

Pascal

```
function RethinkHotkeys: Boolean;
```

C++

```
__fastcall Boolean RethinkHotkeys();
```

Description

Call RethinkHotkeys to adjust the accelerator keys for the items in the Items (see page 16149) property and in any submenu of those items. RethinkHotkeys checks for items that do not have an accelerator key or that have duplicate accelerator keys. It then replaces any duplicate accelerators with unique characters and adds accelerator keys to items that don't have them. RethinkHotkeys gives first priority to existing accelerator key mappings, but may change them if necessary to ensure uniqueness.

RethinkHotkeys lets a menu adjust the accelerator keys when the AutoHotkeys (see page 16134) property is false. For any menu or submenu that has AutoHotkeys (see page 16134) set to true, this adjustment occurs automatically.

RethinkHotkeys returns true if RethinkHotkeys changed the Caption (see page 16136) property of any menu items, false otherwise.

See Also

AutoHotkeys (see page 16134)

AutoHotkeys (see page 16082)

Caption (see page 16136)

Items (see page 16149)

95.1.5.4.40 TMenuItem.RethinkLines Method

Removes superfluous separator lines from the Items (see page 16149) property array.

Pascal

```
function RethinkLines: Boolean;
```

C++

```
__fastcall Boolean RethinkLines();
```

Description

Call RethinkLines to remove superfluous separators from the items in the Items (see page 16149) property and in any submenu of those items. RethinkLines removes separators that appear at the very beginning or end of a submenu and separators that immediately follow another separator.

RethinkLines lets a menu remove superfluous separators when the AutoLineReduction (see page 16135) property is false. For any menu or submenu that has AutoLineReduction (see page 16135) set to true, this adjustment occurs automatically.

RethinkLines returns true if RethinkLines removed any menu items, false otherwise.

See Also

AutoLineReduction (see page 16135)

AutoLineReduction (see page 16082)

IsLine (see page 16125)

Items (see page 16149)

95.1.5.4.41 TMenuItem.SetBreak Method

Pascal

```
procedure SetBreak(Value: TMenuBreak);
```

C++

```
__fastcall SetBreak(TMenuBreak Value);
```

Description

This is SetBreak, a member of class TMenuItem.

95.1.5.4.42 TMenuItem.SetCaption Method

Pascal

```
procedure SetCaption(const Value: string);
```

C++

```
__fastcall SetCaption(const AnsiString Value);
```

Description

This is SetCaption, a member of class TMenuItem.

95.1.5.4.43 TMenuItem.SetChecked Method

Pascal

```
procedure SetChecked(Value: Boolean);
```

C++

```
__fastcall SetChecked(Boolean Value);
```

Description

This is SetChecked, a member of class TMenuItem.

95.1.5.4.44 TMenuItem.SetChildOrder Method

Pascal

```
procedure SetChildOrder(Child: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Child, int Order);
```

Description

This is SetChildOrder, a member of class TMenuItem.

95.1.5.4.45 TMenuItem.SetDefault Method

Pascal

```
procedure SetDefault(Value: Boolean);
```

C++

```
__fastcall SetDefault(Boolean Value);
```

Description

This is SetDefault, a member of class TMenuItem.

95.1.5.4.46 TMenuItem.SetEnabled Method

Pascal

```
procedure SetEnabled(Value: Boolean);
```

C++

```
__fastcall SetEnabled(Boolean Value);
```

Description

This is SetEnabled, a member of class TMenuItem.

95.1.5.4.47 TMenuItem.SetGroupIndex Method

Pascal

```
procedure SetGroupIndex(Value: Byte);
```

C++

```
__fastcall SetGroupIndex(Byte Value);
```

Description

This is SetGroupIndex, a member of class TMenuItem.

95.1.5.4.48 TMenuItem.SetImageIndex Method

Pascal

```
procedure SetImageIndex(Value: TImageIndex);
```

C++

```
__fastcall SetImageIndex(TImageIndex Value);
```

Description

This is SetImageIndex, a member of class TMenuItem.

95.1.5.4.49 TMenuItem.SetMenuIndex Method

Pascal

```
procedure SetMenuIndex(Value: Integer);
```

C++

```
__fastcall SetMenuIndex(int Value);
```

Description

This is SetMenuIndex, a member of class TMenuItem.

95.1.5.4.50 TMenuItem.SetParentComponent Method

Pascal

```
procedure SetParentComponent(Value: TComponent); override;
```

C++

```
virtual __fastcall SetParentComponent(TComponent * Value);
```

Description

This is SetParentComponent, a member of class TMenuItem.

95.1.5.4.51 TMenuItem.SetRadioItem Method

Pascal

```
procedure SetRadioItem(Value: Boolean);
```

C++

```
__fastcall SetRadioItem(Boolean Value);
```

Description

This is SetRadioItem, a member of class TMenuItem.

95.1.5.4.52 TMenuItem.SetShortCut Method

Pascal

```
procedure SetShortCut(Value: TShortCut);
```

C++

```
__fastcall SetShortCut(TShortCut Value);
```

Description

This is SetShortCut, a member of class TMenuItem.

95.1.5.4.53 TMenuItem.SetVisible Method

Pascal

```
procedure SetVisible(Value: Boolean);
```

C++

```
__fastcall SetVisible(Boolean Value);
```

Description

This is SetVisible, a member of class TMenuItem.

95.1.5.4.54 TMenuItem.UpdateItems Method

Pascal

```
procedure UpdateItems;
```

C++

```
__fastcall UpdateItems();
```

Description

This is UpdateItems, a member of class TMenuItem.

95.1.5.5 TMenuItem Properties

The properties of the TMenuItem class are listed here.

Protected Properties

	Name	Description
	ActionLink (see page 16134)	This is ActionLink, a member of class TMenuItem.

Public Properties

	Name	Description
R	Command (see page 16141)	Specifies the Windows Command ID associated with the menu item.
R	Count (see page 16141)	Indicates the number of subitems of the menu item.
R	Handle (see page 16147)	Indicates the Windows menu handle of the drop-down menu associated with the menu item.
R	Items (see page 16149)	Lists the menu items in the submenu of the menu item.
	MenuIndex (see page 16150)	Indicates the index of the menu item within its parent menu.
R	Parent (see page 16151)	Identifies the parent menu item of this menu item.

Published Properties

	Name	Description
	Action (see page 16133)	Designates the action associated with the menu item.
	AutoCheck (see page 16134)	Indicates whether the menu item's checked state toggles automatically when the item is clicked.
	AutoHotkeys (see page 16134)	Determines whether the accelerator keys for submenu items can be reset automatically.
	AutoLineReduction (see page 16135)	Determines whether redundant separator bars are automatically removed from the submenu.
	Bitmap (see page 16135)	Designates a bitmap that is drawn for the menu item.
	Break (see page 16136)	Determines whether the menu item starts a new column in the menu.
	Caption (see page 16136)	Specifies the text of the menu item.
	Checked (see page 16140)	Specifies whether a check mark should appear beside the Caption (see page 16136).
	Default (see page 16145)	Specifies whether the menu item is invoked when the parent item is double clicked.
	Enabled (see page 16145)	Specifies whether the menu item is enabled.
	GroupIndex (see page 16146)	Identifies the logical group to which the menu item belongs.
	HelpContext (see page 16148)	Specifies the help context ID associated with the menu item.
	Hint (see page 16148)	Specifies the text string that can appear when the user moves the mouse pointer over a menu item.
	ImageIndex (see page 16148)	Indicates which image maintained by the parent menu appears next to the menu item.
	RadioItem (see page 16151)	Specifies whether the menu item is mutually exclusive with other menu items in its group.
	ShortCut (see page 16152)	Specifies the key combination users can type to quickly access the menu item.
	SubMenuImages (see page 16152)	Lists the images that can appear beside individual items in a submenu of this menu item.
	Visible (see page 16153)	Specifies whether the menu item appears in the menu.

95.1.5.5.1 TMenuItem.Action Property

Designates the action associated with the menu item.

Pascal

```
property Action: TBasicAction;
```

C++

```
__property TBasicAction Action;
```

Description

Action designates the action object that is associated with the menu item. Actions can be associated with a menu item by using the action list editor at design-time. Actions are a way of centralizing user responses to commands.

95.1.5.5.2 TMenuItem.ActionLink Property

Pascal

```
property ActionLink: TMenuActionLink;
```

C++

```
__property TMenuActionLink ActionLink;
```

Description

This is ActionLink, a member of class TMenuItem.

95.1.5.5.3 TMenuItem.AutoCheck Property

Indicates whether the menu item's checked state toggles automatically when the item is clicked.

Pascal

```
property AutoCheck: Boolean;
```

C++

```
__property Boolean AutoCheck;
```

Description

When AutoCheck is true, then every time the menu item is clicked, the value of the Checked (see page 16140) property toggles automatically before the OnClick (see page 16154) event occurs. When AutoCheck is false, the application must explicitly set the value of the Checked (see page 16140) property (for example, in an OnClick (see page 16154) event handler).

See Also

Checked (see page 16140)

OnClick (see page 16154)

95.1.5.5.4 TMenuItem.AutoHotkeys Property

Determines whether the accelerator keys for submenu items can be reset automatically.

Pascal

```
property AutoHotkeys: TMenuItemAutoFlag;
```

C++

```
__property TMenuItemAutoFlag AutoHotkeys;
```

Description

Set AutoHotkeys to indicate whether the accelerator keys for the items in the Items (see page 16149) property can be reset if necessary to ensure that every menu item has an accelerator key and no two items have the same accelerator key. This ensures that menu items added dynamically at runtime can have accelerator keys that do not introduce conflicts. TMenuItem (see page 16102) tries to preserve all existing accelerator key mappings when adjusting the accelerator keys, but may be forced to change some in order to ensure that every item has a unique accelerator.

Before the menu is displayed, it checks and adjusts the accelerator key mappings if AutoHotkeys is maAutomatic or if AutoHotkeys is maParent and the parent menu item adjusts accelerator key mappings. Otherwise, the menu does not alter the items listed in the Items (see page 16149) property, but may adjust the accelerator keys for submenus, depending on the AutoHotkeys property of the menu item that is the parent of the submenu.

Note: When AutoHotkeys disallows automatic adjustment of accelerator keys, an application can still adjust the menu items by calling the RethinkHotkeys (see page 16128) method. AutoHotkeys merely controls whether this adjustment occurs automatically.

See Also

AutoHotkeys (see page 16082)

Items (see page 16149)

RethinkHotkeys (see page 16128)

95.1.5.5.5 TMenuItem.AutoLineReduction Property

Determines whether redundant separator bars are automatically removed from the submenu.

Pascal

```
property AutoLineReduction: TMenuItemAutoFlag;
```

C++

```
__property TMenuItemAutoFlag AutoLineReduction;
```

Description

Set AutoLineReduction to indicate whether the menu item should automatically ensure that the submenu defined by its Items (see page 16149) property does not start or end with a separator bar and that two separator bars do not appear next to each other. This ensures that menus built dynamically at runtime do not have separator bars in undesired positions.

Before the menu is displayed, the menu item removes superfluous separator bars if AutoLineReduction is maAutomatic or AutoLineReduction is maParent and the parent menu item removes superfluous separator bars. Otherwise, the menu item does not remove separators from the Items (see page 16149) property, but may remove superfluous separators from submenus, depending on the AutoLineReduction property of the menu item that is the parent of the submenu.

Note: When AutoLineReduction disallows automatic removal of separator bars, an application can still remove superfluous separators by calling the RethinkLines (see page 16129) method. AutoLineReduction merely controls whether this adjustment occurs automatically.

See Also

AutoLineReduction (see page 16082)

Items (see page 16149)

RethinkLines (see page 16129)

95.1.5.5.6 TMenuItem.Bitmap Property

Designates a bitmap that is drawn for the menu item.

Pascal

```
property Bitmap: TBitmap;
```

C++

```
__property TBitmap Bitmap;
```

Description

Use Bitmap to provide a custom image that appears next to the caption in the menu item. On owner-draw menus, Bitmap can be used from an OnDrawItem (see page 16159) event handler to represent the entire menu item or a portion of the menu item.

Note: If the ImageIndex (see page 16148) property is set and the parent menu has a non-nil (Delphi) or NULL (C++) Images property, the image identified by ImageIndex (see page 16148) is used instead of Bitmap.

See Also

ImageIndex (see page 16148)

Images (see page 16083)

Caption (see page 16136)

OwnerDraw (see page 16088)

OnDrawItem (see page 16159)

95.1.5.5.7 TMenuItem.Break Property

Determines whether the menu item starts a new column in the menu.

Pascal

```
property Break: TMenuBreak;
```

C++

```
__property TMenuBreak Break;
```

Description

Set Break to break a long menu into columns at the item described by this menu item. The value of Break indicates whether the menu is broken into columns, and if so, whether a bar appears between the columns of the parent menu.

The default is mbNone (no breaking into columns).

See Also

Caption (see page 16136)

Parent (see page 16151)

95.1.5.5.8 TMenuItem.Caption Property

Specifies the text of the menu item.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Set Caption to the string the user sees for this item in the menu. To underline a character in a string, include an ampersand (&) before the character. This type of character is called an accelerator character. If Caption includes an accelerator character, the user can select the menu item by pressing Alt while typing the underlined character.

Tip: To automatically assign or adjust the accelerator characters in menu item captions, use the RethinkHotkeys (see page 16128) method.

Use the Caption property to identify the menu item as a line that separates the menu into parts. Specify a hyphen character (-) as the value of Caption for the menu item to indicate that the menu item is a separator. To separate the menu into multiple columns, use the Break (see page 16136) property instead.

Note: This property is not stored with the menu item if it is supplied by a TAction object.

C++ Examples:

```
/*
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *NewItem;
  // first create the separator
  NewItem = new TMenuItem(this); // If you change the AOwner (this) to MainMenu1->Items[1] and
then you add the new menu you will get "Menu inserted twice"?  Doesn't happen in Delphi.
  NewItem->Name = "Separator";
  NewItem->Caption = "-";
  // add the new item to the Windows menu
  MainMenu1->Items[1].Add(NewItem);
  // now create and add a menu item for each form
  for  (int I = 0; I < Screen->FormCount; I++)
  {
    NewItem = new TMenuItem(this);
    NewItem->Caption = Screen->Forms[I]->Name + "Item";
    NewItem->Name = Screen->Forms[I]->Name;
    MainMenu1->Items[1].Add(NewItem);
  }
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  // Count down, not up!
  for  (int I = MainMenu1->Items[1].Count - 1; I >= 0; I--)
  {
    delete MainMenu1->Items[1].Items[I];
  }
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

```
void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example assumes that the main form of the application
has a main menu with a menu item. The following code adds a
separator, and the name of all forms to the menu. Do not
access FormCount in the mainform's FormCreate.  The other
forms do not exist yet!
}
procedure TForm1.Button1Click(Sender: TObject);
var
  NewItem: TMenuItem;
  I : integer;
begin
  { first create the separator }
  NewItem := TMenuItem.Create(MainMenu1.Items[1]);
  NewItem.Name:= 'Separator';
  NewItem.Caption := '-';
```

```
    { add the new item to the Windows menu }
    MainMenu1.Items[1].Add(NewItem);
    { now create and add a menu item for each form }
    for  I := 0 to Screen.FormCount-1 do
    begin
      NewItem := TMenuItem.Create(MainMenu1.Items[1]);
      NewItem.Caption := Screen.Forms[I].Name + 'Item';
      NewItem.Name := Screen.Forms[I].Name;
      MainMenu1.Items[1].Add(NewItem);
     end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I : Integer;
  begin
    // Count down, not up!
    for  I := MainMenu1.Items[1].Count - 1 downto 0 do
    begin
      MainMenu1.Items[1].Items[I].Free;
    end;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
```

```
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

Break (see page 16136)

Checked (see page 16140)

Caption (see page 308)

RethinkHotkeys (see page 16128)

95.1.5.5.9 TMenuItem.Checked Property

Specifies whether a check mark should appear beside the Caption (see page 16136).

Pascal

```
property Checked: Boolean;
```

C++

```
__property Boolean Checked;
```

Description

Use Checked to allow a menu item to work like a check box. If Checked is true, the menu item appears checked. If Checked is

false, the menu item remains unchecked.

Note: This property is not stored with the menu item if it is supplied by a TAction object.

See Also

Caption (see page 16136)

RadioItem (see page 16151)

Checked (see page 308)

AutoCheck (see page 16134)

95.1.5.5.10 TMenuItem.Command Property

Specifies the Windows Command ID associated with the menu item.

Pascal

```
property Command: Word;
```

C++

```
__property Word Command;
```

Description

Use Command in an application that directly handles WM_COMMAND messages. When the user selects the menu item, Windows sends a WM_COMMAND to the window associated with the menu. The ItemID of the message contains the value of the Command property of the menu item that was selected.

See Also

OnClick (see page 16154)

DispatchCommand (see page 16074)

WindowHandle (see page 16089)

95.1.5.5.11 TMenuItem.Count Property

Indicates the number of subitems of the menu item.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read Count to determine the number of subitems listed in the Items (see page 16149) property array. When the user clicks on a menu item that has subitems, a dropdown menu appears which displays those subitems. Each subitem can, in turn, contain additional subitems. The Count property counts only the immediate subitems of the menu item.

C++ Examples:

```
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
```

```
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
```

```
}
```

Delphi Examples:

```
{
The following code disables all the subitems of of the menu
item File1.  This example requires a button, a TMainMenu
with several menu items added, one of them named File1.
}
procedure TForm1.DisableSubItemsClick(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to File1.Count - 1 do
    File1.Items[I].Enabled := False;
end;

procedure TForm1.Edit1Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the Edit command');
end;

procedure TForm1.File1Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the File command');
end;

procedure TForm1.FileSub11Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the FileSub1 command');
end;

procedure TForm1.FileSub21Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the FileSub2 command');
end;

procedure TForm1.FileSub31Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the FileSub3 command');
end;

procedure TForm1.Options1Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the Options command');
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
```

```
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

Items (see page 16149)

MenuIndex (see page 16150)

95.1.5.5.12 TMenuItem.Default Property

Specifies whether the menu item is invoked when the parent item is double clicked.

Pascal

```
property Default: Boolean;
```

C++

```
__property Boolean Default;
```

Description

Set Default to specify whether a menu item is the default item in a submenu. Default menu items are executed when the parent item is double-clicked, allowing users to avoid navigating the submenus that contain them.

The default item appears in boldface. A submenu can have only one default item. If the Items (see page 16149) list of the parent menu item already contains a default menu item, setting the Default property to true sets the Default property of the previous default item to false.

See Also

Click (see page 16116)

Parent (see page 16151)

95.1.5.5.13 TMenuItem.Enabled Property

Specifies whether the menu item is enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to enable or disable a menu item. If Enabled is true, the Click (see page 16116) method is called when the user selects the item with the mouse. If Enabled is false, the menu item appears dimmed and the user cannot select it. However, calling the Click (see page 16116) method works even when Enabled is false.

Note: This property is not stored with the menu item if it is supplied by a TAction object.

Delphi Examples:

```
{
The following code disables all the subitems of of the menu
item File1.  This example requires a button, a TMainMenu
with several menu items added, one of them named File1.
}
procedure TForm1.DisableSubItemsClick(Sender: TObject);
var
  I: Integer;
```

```
begin
  for I := 0 to File1.Count - 1 do
    File1.Items[I].Enabled := False;
end;

procedure TForm1.Edit1Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the Edit command');
end;

procedure TForm1.File1Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the File command');
end;

procedure TForm1.FileSub11Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the FileSub1 command');
end;

procedure TForm1.FileSub21Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the FileSub2 command');
end;

procedure TForm1.FileSub31Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the FileSub3 command');
end;

procedure TForm1.Options1Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the Options command');
end;
```

See Also

Click (see page 16116)

Visible (see page 16153)

Enabled (see page 310)

95.1.5.5.14 TMenuItem.GroupIndex Property

Identifies the logical group to which the menu item belongs.

Pascal

```
property GroupIndex: Byte;
```

C++

```
__property Byte GroupIndex;
```

Description

Use GroupIndex to control how menus are merged. In applications that contain multiple forms, items from the menus of secondary forms can be merged into the application's main menu as those forms become active. MDI applications always merge the menus of child windows with the main menu of the parent window.

By default, all menu items in a menu bar (main menu) have the same GroupIndex value. To use GroupIndex to merge menus, set the GroupIndex value of each successive menu item to a value equal to or greater than that of the previous menu item.

When a menu item in a menu bar that is merged into the application's main menu has the same GroupIndex value as a menu item in the main menu, the menu item in the application's main menu is replaced by the corresponding menu item. If multiple

menu items in the application's main menu have the same GroupIndex value, they can be replaced only by separate menu items from the other form's menu that also have repeated GroupIndex values. Thus, the first main menu item with a given value is replaced by the first item in the second form with that value, the second item in the main form is replaced by the second item in the second form, and so on.

When a menu item in a menu bar that is merged into the application's main menu has a GroupIndex that falls between the GroupIndex values of two adjacent items on the application's main menu, that menu item is inserted between the two adjacent items. If the item's GroupIndex value is greater than all other GroupIndex values in the application's main menu, the item appears at the end of the main menu. If the GroupIndex value is less than all other GroupIndex values in the application's main menu, the menu item appears at the beginning of the main menu bar.

When an object created by an OLE 2.0 server application is activated, the server can try to merge its menus with the menus of the container application. The GroupIndex property is used to replace up to three main menu items with items from the server, in the manner described above. The server application uses the following predefined GroupIndex values to locate menu items to replace:

Group	Index	Description
Edit	1	Server menu items for editing the active OLE object
View	3	Server menu items for modifying the view of the OLE object.
Help	5	Server menu items for accessing the server's online Help.

Note: See the documentation for the OLE server for information about whether it attempts menu merge during in-place activation.

GroupIndex can also be used to define groups of menu items that act like radio buttons. To allow menu items to behave like radio buttons, set the RadioItem (see page 16151) property to all menu items in the group to true, and assign each menu item the same value of GroupIndex. Checking any menu item in the group will then uncheck the others.

See Also

MenuIndex (see page 16150)

RadioItem (see page 16151)

AutoMerge (see page 16065)

Merge (see page 16061)

PopulateOle2Menu (see page 16062)

95.1.5.5.15 TMenuItem.Handle Property

Indicates the Windows menu handle of the drop-down menu associated with the menu item.

Pascal

```
property Handle: HMENU;
```

C++

```
__property HMENU Handle;
```

Description

Use Handle when calling a Windows API function that requires a menu handle. Handle is only valid if the Count (see page 16141) property is greater than 0.

See Also

Count (see page 16141)

Items (see page 16149)

95.1.5.5.16 TMenuItem.HelpContext Property

Specifies the help context ID associated with the menu item.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

Use HelpContext to associate the menu item with a screen in the Help system of the application. Each screen in the Help system should have a unique context ID. When a menu item is selected using the keyboard, pressing F1 displays a Help screen. Which Help screen appears depends on the value of the HelpContext property.

Note: This property is not stored with the menu item if it is supplied by a TAction object.

See Also

HelpFile (see page 13711)

HelpContext (see page 5826)

HelpContext (see page 311)

95.1.5.5.17 TMenuItem.Hint Property

Specifies the text string that can appear when the user moves the mouse pointer over a menu item.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Set Hint to a string that provides more information about the meaning of the menu item than the Caption (see page 16136). The hint text appears in the **Status Bar** when the user pauses with the mouse over the menu item if Help Hints are enabled (that is, if the Form's and the Application's Show Hint properties are true). It is also available for the code in the application's On Hint event handler.

The value of Hint can specify both a short value for the Help Hint window and a longer string to be used by the OnHint event handler. To provide both a short and a long hint, set Hint to the short string, followed by a vertical bar (|), followed by the long string.

Note: This property is not stored with the menu item if it is supplied by a TAction object.

95.1.5.5.18 TMenuItem.ImageIndex Property

Indicates which image maintained by the parent menu appears next to the menu item.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

Set ImageIndex to designate an image that should appear next to the caption when the menu item is drawn. On top-level menu items, ImageIndex specifies a zero-offset index into the Images property of the parent menu (TMenu (see page 16067) or TPopupMenu (see page 16169)). On submenu items, ImageIndex is the zero-offset index into the parent item's SubMenuImages (see page 16152) property.

If the parent menu or item does not maintain a list of images, use the Bitmap (see page 16135) property to provide an image for a single menu item. However, any image specified by ImageIndex takes precedence over the Bitmap (see page 16135) property. Bitmap (see page 16135) is only used when ImageIndex has a negative value or the parent's Images or SubMenuImages (see page 16152) property is nil (Delphi) or NULL (C++).

Note: This property is not stored with the menu item if it is supplied by a TAction object.

See Also

Images (see page 16083)

Bitmap (see page 16135)

ImageIndex (see page 313)

SubMenuImages (see page 16152)

95.1.5.5.19 TMenuItem.Items Property

Lists the menu items in the submenu of the menu item.

Pascal

```
property Items [Index: Integer]: TMenuItem;
```

C++

```
__property TMenuItem Items[int Index];
```

Description

Use Items to access to a subitem by its position in the list of subitems. The value of Index is the position of the subitem within the Items array. For example, if an application has a File drop-down menu that contains the menu items New, Open, and Save, in that order, the expression

```
FileMenu.Items[2]
FileMenu->Items[2]
```

refers to the Save command.

Note: In Delphi, Items is the default property for TMenuItem (see page 16102). This means you can leave off the property name. Thus, instead of writing FileMenu.Items[2], you can write FileMenu[2].

Note: In C++, Items can be accessed using the [] operator, to achieve an affect similar to the default property in Delphi.

Delphi Examples:

```
{
The following code disables all the subitems of of the menu
item File1.  This example requires a button, a TMainMenu
```

```
with several menu items added, one of them named File1.
}
procedure TForm1.DisableSubItemsClick(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to File1.Count - 1 do
    File1.Items[I].Enabled := False;
end;

procedure TForm1.Edit1Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the Edit command');
end;

procedure TForm1.File1Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the File command');
end;

procedure TForm1.FileSub11Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the FileSub1 command');
end;

procedure TForm1.FileSub21Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the FileSub2 command');
end;

procedure TForm1.FileSub31Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the FileSub3 command');
end;

procedure TForm1.Options1Click(Sender: TObject);
begin
  ListBox1.Items.Add('This is the Options command');
end;
```

See Also

Count (see page 16141)

MenuIndex (see page 16150)

Parent (see page 16151)

operator_[]

95.1.5.5.20 TMenuItem.MenuIndex Property

Indicates the index of the menu item within its parent menu.

Pascal

```
property MenuIndex: Integer;
```

C++

```
__property int MenuIndex;
```

Description

Use MenuIndex to locate the menu item in the Items (see page 16149) property of its parent menu item. Changing this value moves the menu item within its menu.

Note: MenuIndex may not correspond to the position of the item in the menu the user sees. The menu can include items that are not visible.

See Also

Count (see page 16141)

GroupIndex (see page 16146)

Items (see page 16149)

Parent (see page 16151)

Visible (see page 16153)

95.1.5.5.21 TMenuItem.Parent Property

Identifies the parent menu item of this menu item.

Pascal

```
property Parent: TMenuItem;
```

C++

```
__property TMenuItem Parent;
```

Description

Read Parent to determine the menu item that represents the dropdown menu that contains this menu item. If the menu item is a top level menu item in a main menu or pop-up menu, Parent is the value of that menu's Items (see page 16149) property.

See Also

Items (see page 16149)

MenuIndex (see page 16150)

Items (see page 16084)

95.1.5.5.22 TMenuItem.RadioItem Property

Specifies whether the menu item is mutually exclusive with other menu items in its group.

Pascal

```
property RadioItem: Boolean;
```

C++

```
__property Boolean RadioItem;
```

Description

Use RadioItem to make the menu item behave like a radio button with a group of other menu items. The group of menu items is the set of all menu items in a pop-up or drop-down menu that have the same value of GroupIndex (see page 16146). When RadioItem is true, only one menu item in the group that contains the menu item can be checked at a time. The single selected menu item in the group is indicated by a round dot next to the Caption (see page 16136).

Note: RadioItem does not automatically select the item when the user clicks it at runtime: you must still explicitly set the Checked (see page 16140) property to true in response to the OnClick (see page 16154) event. RadioItem simply changes the appearance of the Checked (see page 16140) state (from a check mark to a round dot) and automatically unchecks other menu items in the same group when you set the Checked (see page 16140) property to true.

RadioItem does not work for the top-level items in a main menu.

Note: Under Windows NT 3.51, the round dot indicator only appears if the "new shell" has been installed.

See Also

Caption (see page 16136)

Checked (see page 16140)

GroupIndex (see page 16146)

TMainMenu (see page 16056)

OnClick (see page 16154)

95.1.5.5.23 TMenuItem.ShortCut Property

Specifies the key combination users can type to quickly access the menu item.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

Use ShortCut to allow the user to type a key combination instead of selecting the item using the menu. This key combination appears to the right of the menu item in the menu.

When setting menu shortcuts at design time, the Object Inspector provides a long list of key combinations to choose from. To create menu shortcuts at runtime, use the global ShortCut or TextToShortCut (see page 16196) function. For example, the following line creates a shortcut, Ctrl+C and assigns it to a menu item named CloseCommand.

```
CloseCommand.ShortCut := ShortCut(Word('C'), [ssCtrl]);
CloseCommand->ShortCut = ShortCut(Word('C'), TShiftState() << ssCtrl);
```

Note: This property is not stored with the menu item if it is supplied by a TAction object.

See Also

ShortCut (see page 16193)

ShortCutToKey (see page 16194)

ShortCutToText (see page 16195)

TextToShortCut (see page 16196)

IsShortCut (see page 16078)

ShortCut (see page 314)

95.1.5.5.24 TMenuItem.SubMenuImages Property

Lists the images that can appear beside individual items in a submenu of this menu item.

Pascal

```
property SubMenuImages: TCustomImageList;
```

C++

```
__property TCustomImageList SubMenuImages;
```

Description

Use SubMenuImages to supply a set of images that can appear beside items listed in the Items (see page 16149) property. Individual menu items set their ImageIndex (see page 16148) property to indicate which image in the list appears to the left of the menu item caption.

Tip: If the application uses action lists, this property can be the value of the command list's Images property. This way, menu items and buttons that do the same thing can display the same image.

See Also

ImageIndex (see page 16148)

Images (see page 325)

Items (see page 16149)

95.1.5.5.25 TMenuItem.Visible Property

Specifies whether the menu item appears in the menu.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use Visible to prevent the user from seeing an item in the menu. For example, remove menu items that do not apply to the current component of a pop-up menu in an OnPopup event handler.

If Visible is true, the menu item appears. If Visible is false, the menu item is hidden.

Note: This property is not stored with the menu item if it is supplied by a TAction object.

See Also

Enabled (see page 16145)

OnPopup (see page 16184)

PopupComponent (see page 16181)

Visible (see page 314)

95.1.5.6 TMenuItem Events

The events of the TMenuItem class are listed here.

Published Events

	Name	Description
	OnAdvancedDrawItem (see page 16154)	Occurs when an owner-draw menu needs to be drawn.
	OnClick (see page 16154)	Occurs when the user clicks menu item.
	OnDrawItem (see page 16159)	Occurs when an owner-draw menu needs to be drawn.
	OnMeasureItem (see page 16160)	Occurs on owner-draw menu items to determine menu item size before the menu is drawn.

95.1.5.6.1 TMenuItem.OnAdvancedDrawItem Event

Occurs when an owner-draw menu needs to be drawn.

Pascal

```
property OnAdvancedDrawItem: TAdvancedMenuDrawItemEvent;
```

C++

```
__property TAdvancedMenuDrawItemEvent OnAdvancedDrawItem;
```

Description

Write an OnAdvancedDrawItem event handler to render the image of a menu item in an owner-draw menu.

In the event handler, use the Bitmap (see page 16135) property or the ImageIndex (see page 16148) property to access a bitmap that represents the menu item.

Note: OnAdvancedDrawItem will only occur if the parent menu's OwnerDraw property is true or the parent menu's Images property has been set.

See Also

OnMeasureItem (see page 16160)

Bitmap (see page 16135)

ImageIndex (see page 16148)

Images (see page 16083)

OwnerDraw (see page 16088)

OnDrawItem (see page 16159)

95.1.5.6.2 TMenuItem.OnClick Event

Occurs when the user clicks menu item.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Write an OnClick event handler to implement the desired behavior for when the user selects the menu item. An advantage of using the OnClick event handler over directly handling WM_COMMAND messages sent when the user selects the menu item, is that only the OnClick event occurs when the Click (see page 16116) method is called.

Note: This event handler is not stored with the menu item if it is supplied by a TAction object.

C++ Examples:

```
/*
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
```

```
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
*/
void __fastcall TForm1::MyArrangeIconsClick(TObject *Sender)
{
  Form1->ArrangeIcons();
}
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
```

```
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
```

```
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses several forms. The first form has its
FormStyle property set to MDIForm. The others have their
FormStyle properties set to MDIChild and their Visible
properties set to true. Add a main menu component and name
one of the menu items MyArrangeIcons. This is code for the
MyArrangeIconsClick handler.  When the user chooses the
ArrangeIcons command, minimized child forms are arranged
so that they are evenly spaced and don't overlap.  DO NOT
name the menu items "ArrangeIcons" as that will override
the TForm method.
}
procedure TForm1.MyArrangeIconsClick(Sender: TObject);
begin
  Form1.ArrangeIcons;
end;
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
```

```
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
```

```
      PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

Click (see page 16116)

Command (see page 16141)

OnExecute (see page 2716)

95.1.5.6.3 TMenuItem.OnDrawItem Event

Occurs when an owner-draw menu needs to be drawn.

Pascal

```
property OnDrawItem: TMenuDrawItemEvent;
```

C++

```
__property TMenuDrawItemEvent OnDrawItem;
```

Description

Write an OnDrawItem event handler to render the image of a menu item in an owner-draw menu. In the event handler, use the Bitmap (see page 16135) property or the ImageIndex (see page 16148) property to access a bitmap that represents the menu item.

Note: OnDrawItem will only occur if the parent menu's OwnerDraw property is true or the parent menu's Images property has been set.

Tip: For more precise information about the menu item's state, use the OnAdvancedDrawItem (see page 16154) event instead.

See Also

OnMeasureItem (see page 16160)

Bitmap (see page 16135)

ImageIndex (see page 16148)

Images (see page 16083)

OwnerDraw (see page 16088)

OnAdvancedDrawItem (see page 16154)

95.1.5.6.4 TMenuItem.OnMeasureItem Event

Occurs on owner-draw menu items to determine menu item size before the menu is drawn.

Pascal

```
property OnMeasureItem: TMenuMeasureItemEvent;
```

C++

```
__property TMenuMeasureItemEvent OnMeasureItem;
```

Description

Write an OnMeasureItem event handler to indicate the size of owner-draw menu items. If the menu item is using a bitmap image, the event handler can use the Bitmap (see page 16135) property or the ImageIndex (see page 16148) property to access the bitmap to determine its size.

Note: OnMeasureItem will only occur if the parent menu's OwnerDraw property is true or the parent menu's Images property has been set.

See Also

OnDrawItem (see page 16159)

Bitmap (see page 16135)

ImageIndex (see page 16148)

Images (see page 16083)

OwnerDraw (see page 16088)

TextExtent (see page 14474)

95.1.6 TMenuItemEnumerator Class

Class Hierarchy

Menus.TMenuItemEnumerator

Pascal

```
TMenuItemEnumerator = class;
```

C++

```
class TMenuItemEnumerator;
```

File

Menus

Description

This is class Menus.TMenuItemEnumerator.

95.1.6.1 TMenuItemEnumerator Members

The following tables list the members exposed by TMenuItemEnumerator.

Public Methods

	Name	Description
	Create (see page 16161)	This is Create, a member of class TMenuItemEnumerator.
	GetCurrent (see page 16161)	This is GetCurrent, a member of class TMenuItemEnumerator.
	MoveNext (see page 16162)	This is MoveNext, a member of class TMenuItemEnumerator.

Public Properties

	Name	Description
R	Current (see page 16162)	This is Current, a member of class TMenuItemEnumerator.

95.1.6.2 TMenuItemEnumerator.Create Constructor

Pascal

```
constructor Create(AMenuItem: TMenuItem);
```

C++

```
__fastcall TMenuItemEnumerator(TMenuItem AMenuItem);
```

Description

This is Create, a member of class TMenuItemEnumerator.

95.1.6.3 TMenuItemEnumerator Methods

The methods of the TMenuItemEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 16161)	This is GetCurrent, a member of class TMenuItemEnumerator.
	MoveNext (see page 16162)	This is MoveNext, a member of class TMenuItemEnumerator.

95.1.6.3.1 TMenuItemEnumerator.GetCurrent Method

Pascal

```
function GetCurrent: TMenuItem;
```

C++

```
__fastcall TMenuItem GetCurrent();
```

Description

This is GetCurrent, a member of class TMenuItemEnumerator.

95.1.6.3.2 TMenuItemEnumerator.MoveNext Method

Pascal

```
function MoveNext: Boolean;
```

C++

```
__fastcall Boolean MoveNext();
```

Description

This is MoveNext, a member of class TMenuItemEnumerator.

95.1.6.4 TMenuItemEnumerator Properties

The properties of the TMenuItemEnumerator class are listed here.

Public Properties

	Name	Description
	Current (see page 16162)	This is Current, a member of class TMenuItemEnumerator.

95.1.6.4.1 TMenuItemEnumerator.Current Property

Pascal

```
property Current: TMenuItem;
```

C++

```
__property TMenuItem Current;
```

Description

This is Current, a member of class TMenuItemEnumerator.

95.1.7 TMenuItemStack Class

Class Hierarchy

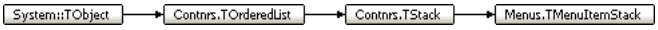

Pascal

```
TMenuItemStack = class(TStack);
```

C++

```
class TMenuItemStack : public TStack;
```

File

Menus

Description

This is class Menus.TMenuItemStack.

95.1.7.1 TMenuItemStack Members

The following tables list the members exposed by TMenuItemStack.

Protected Methods

	Name	Description
V	PeekItem (see page 5697)	Returns a pointer to the next item in the list.
V	PopItem (see page 5697)	Removes and returns a pointer to the next item in the list.
A	PushItem (see page 5698)	Adds an item to the list.
V	PushItem (see page 5703)	Adds an item to the list.

Protected Properties

	Name	Description
R	List (see page 5699)	Identifies the internal list used to store and manage the items in the ordered list.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AtLeast (see page 5696)	Checks the size of the list.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearItem (see page 16164)	This is ClearItem, a member of class TMenuItemStack.
	Count (see page 5696)	Returns the size of the list.
	Create (see page 5695)	Creates a TOrderedList (see page 5693) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5695)	Destroys the TOrderedList (see page 5693) instance.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Peek (see page 5696)	Returns a pointer to the next item in the list.
	Pop (see page 5697)	Removes and returns the next item in the list.
	Push (see page 5698)	Adds an item to the list.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

95.1.7.2 TMenuItemStack Methods

The methods of the TMenuItemStack class are listed here.

Public Methods

	Name	Description
	ClearItem (see page 16164)	This is ClearItem, a member of class TMenuItemStack.

95.1.7.2.1 TMenuItemStack.ClearItem Method

Pascal

```
procedure ClearItem(AItem: TMenuItem);
```

C++

```
__fastcall ClearItem(TMenuItem AItem);
```

Description

This is ClearItem, a member of class TMenuItemStack.

95.1.8 TPopupList Class

TPopupList manages the window that handles popup menu messages.

Class Hierarchy

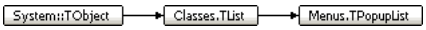

Pascal

```
TPopupList = class(TList);
```

C++

```
class TPopupList : public TList;
```

File

Menus

Description

TPopupList is the type of the global PopupList (see page 16207) variable, which provides centralized message handling for all popup menus in an application.

See Also

TPopupMenu (see page 16169)

PopupList (see page 16207)

95.1.8.1 TPopupList Members

The following tables list the members exposed by TPopupList.

Protected Fields

	Name	Description
	FWindow (see page 16166)	This is FWindow, a member of class TPopupList.

Protected Methods

	Name	Description
	Get (see page 2878)	This is Get, a member of class TList.
V	Grow (see page 2879)	This is Grow, a member of class TList.
	MainWndProc (see page 16167)	Wraps the WndProc (see page 16168) method with an exception-handling layer.
V	Notify (see page 2884)	Responds when items are added, removed, or extracted from the list.
	Put (see page 2887)	This is Put, a member of class TList.
	SetCapacity (see page 2891)	This is SetCapacity, a member of class TList.
	SetCount (see page 2892)	This is SetCount, a member of class TList.
V	WndProc (see page 16168)	Provides centralized message handling for all popup menus in the application.

Public Methods

	Name	Description
	Add (see page 16167)	Inserts a new popup menu at the end of the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2874)	Copies elements of one list to another.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 2874)	Deletes all items from the list.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2874)	Removes the item at the position given by the Index parameter.
V	Destroy (see page 2870)	Destroys an instance of TList (see page 2868).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Error (see page 2875)	This is the overview for the Error method overload.
	Exchange (see page 2876)	Swaps the position of two items in the Items (see page 2898) array.
	Expand (see page 2877)	Increases the Capacity (see page 2895) of the list.
	Extract (see page 2877)	Removes a specified item from the list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	First (see page 2878)	Returns Items (see page 2898)[0].
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2878)	Returns a TList (see page 2868) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	IndexOf (see page 2879)	Returns the index of the first entry in the Items (see page 2898) array with a specified value.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2882)	Adds an object to the Items (see page 2898) array at the position specified by Index.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Last (see page 2882)	Returns Items (see page 2898)[Count (see page 2896) - 1].
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	Move (see page 2884)	Changes the position of an item in the Items (see page 2898) array.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Pack (see page 2885)	Deletes all nil (Delphi) or NULL (C++) items from the Items (see page 2898) array.
	Remove (see page 16167)	Removes a popup menu from the list.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Sort (see page 2892)	Performs a QuickSort on the list based on the comparison function Compare.

Public Properties

	Name	Description
	Capacity (see page 2895)	Specifies the allocated size of the array of pointers maintained by the TList (see page 2868) object.
	Count (see page 2896)	Indicates the number of entries in the list that are in use.
	Items (see page 2898)	Lists the object references.
R	List (see page 2900)	Specifies the array of pointers that make up the Items (see page 2898) array.
R	Window (see page 16168)	Provides access to the Window handle of the hidden window that processes popup menu messages.

95.1.8.2 TPopupList Fields

The fields of the TPopupList class are listed here.

Protected Fields

	Name	Description
	FWindow (see page 16166)	This is FWindow, a member of class TPopupList.

95.1.8.2.1 TPopupList.FWindow Field

Pascal

```
FWindow: HWND;
```

C++

```
HWND FWindow;
```

Description

This is FWindow, a member of class TPopupList.

95.1.8.3 TPopupList Methods

The methods of the TPopupList class are listed here.

Protected Methods

	Name	Description
	MainWndProc (see page 16167)	Wraps the WndProc (see page 16168) method with an exception-handling layer.
V	WndProc (see page 16168)	Provides centralized message handling for all popup menus in the application.

Public Methods

	Name	Description
	Add (see page 16167)	Inserts a new popup menu at the end of the list.
	Remove (see page 16167)	Removes a popup menu from the list.

95.1.8.3.1 TPopupList.Add Method

Inserts a new popup menu at the end of the list.

Pascal

```
procedure Add(Popup: TPopupMenu);
```

C++

```
__fastcall Add(TPopupMenu Popup);
```

Description

Do not call the Add method. Popup menus automatically add themselves to the list when they are created.

Add increments Count and, if necessary, allocates memory by increasing the value of Capacity. When the first popup menu is added to the list, Add creates the hidden window whose handle is available as the Window (see page 16168) property.

See Also

Remove (see page 16167)

Window (see page 16168)

95.1.8.3.2 TPopupList.MainWndProc Method

Wraps the WndProc (see page 16168) method with an exception-handling layer.

Pascal

```
procedure MainWndProc(var Message: TMessage);
```

C++

```
__fastcall MainWndProc(TMessage Message);
```

Description

Do not use the protected MainWndProc method. It is used internally when creating the hidden window that responds to popup menu messages. To subclass the window procedure that handles messages sent to popup menus, override the WndProc (see page 16168) method instead.

See Also

WndProc (see page 16168)

95.1.8.3.3 TPopupList.Remove Method

Removes a popup menu from the list.

Pascal

```
procedure Remove(Popup: TPopupMenu);
```

C++

```
__fastcall Remove(TPopupMenu Popup);
```

Description

Do not call the Remove method. It is called automatically by popup menus from their destructors, to remove themselves from the list.

See Also

Add (see page 16167)

95.1.8.3.4 TPopupList.WndProc Method

Provides centralized message handling for all popup menus in the application.

Pascal

```
procedure WndProc(var Message: TMessage); virtual;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

Override WndProc to subclass the window procedure that handles messages directed at the popup menus in an application. WndProc handles messages by dispatching commands, displaying help, painting menu item, and so on.

95.1.8.4 TPopupList Properties

The properties of the TPopupList class are listed here.

Public Properties

	Name	Description
R	Window (see page 16168)	Provides access to the Window handle of the hidden window that processes popup menu messages.

95.1.8.4.1 TPopupList.Window Property

Provides access to the Window handle of the hidden window that processes popup menu messages.

Pascal

```
property Window: HWND;
```

C++

```
__property HWND Window;
```

Description

Use Window for Windows API calls that require a window handle. The hidden window that processes messages for the popup menus in an application is not created until the first popup menu is added to the list. If the list contains no popup menus (that is, if no popup menus are created by the application), Window is 0.

95.1.9 TPopupMenu Class

TPopupMenu encapsulates the properties, methods, and events of a pop-up menu.

Class Hierarchy

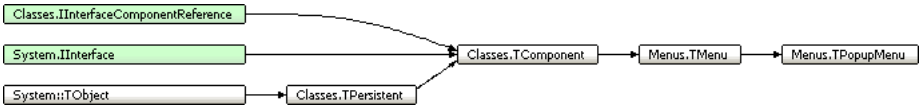

Pascal

```
TPopupMenu = class(TMenu);
```

C++

```
class TPopupMenu : public TMenu;
```

File

Menus

Description

Use TPopupMenu to define the pop-up menu that appears when the user clicks on a control with the right mouse button. To make a pop-up menu available, assign the TPopupMenu object to the control's PopupMenu property.

Tip: To create a popup menu that users can configure at runtime, use the TStandardMenuPopup component with an action manager instead.

Note: If the popup menu's ParentBiDiMode (see page 16180) is true, the popup menu's BiDiMode (see page 16178) is set to the BiDiMode (see page 16178) of the control that activates it. If a control cannot be found, the BiDiMode (see page 16178) is set to the value of the Application object's BiDiMode (see page 16178). The popup menu's BiDiMode (see page 16178) affects all of its menu items.

See Also

TMainMenu (see page 16056)

TMenuItem (see page 16102)

NewPopupMenu (see page 16191)

TStandardMenuPopup (see page 20478)

TActionManager (see page 432)

Providing a Pop-up Menu

Setting Up Action Bands

95.1.9.1 TPopupMenu Members

The following tables list the members exposed by TPopupMenu.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AdjustBiDiBehavior (see page 16073)	Applies the BiDiMode (see page 16083) property to the menu when the Windows control is created.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoBiDiModeChanged (see page 16074)	Updates the menu when the BiDiMode (see page 16083) property changes.
V	DoChange (see page 16075)	Generates an OnChange (see page 16090) event.
	DoGetMenuString (see page 16075)	Returns the caption for an item in the menu.
V	DoPopup (see page 16174)	Generates an OnPopup (see page 16184) event.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 16076)	Executes a callback for child menu items.
V	GetHandle (see page 16076)	Returns the value of the Handle (see page 16083) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsOwnerDraw (see page 16077)	Indicates whether menu items can contain more than simple captions.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 16078)	Calls the DoChange (see page 16075) method when the menu is first loaded into memory.
V	MenuChanged (see page 16079)	Responds when the menu changes.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16079)	Responds when components are added or deleted.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 16081)	Ensures that the menu item indexes reflect the order in which they are loaded and saved.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetPopupPoint (see page 16175)	This is SetPopupPoint, a member of class TPopupMenu.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateItems (see page 16081)	Causes all menu items to update themselves.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UseRightToLeftAlignment (see page 16175)	Specifies whether the control's alignment is in a right-to-left mode.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 16173)	Creates an instance of TPopupMenu (see page 16169).
	CreateMenuItem (see page 16073)	This is CreateMenuItem, a member of class TMenu.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 16173)	Destroys an instance of TPopupMenu (see page 16169).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DispatchCommand (see page 16074)	Calls the Click method of the menu item with the specified value as its Command property.
	DispatchPopup (see page 16074)	Dispatches a pop-up menu command.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindItem (see page 16076)	Locates and returns a menu item.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHelpContext (see page 16077)	Returns the help context ID associated with a menu item.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 16077)	Indicates whether menu items read from right to left.
	IsShortCut (see page 16078)	Dispatches a key message if it represents a shortcut key for an item in the menu.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentBiDiModeChanged (see page 16079)	This is the overview for the ParentBiDiModeChanged method overload.
V	Popup (see page 16174)	Displays the pop-up menu onscreen.
	ProcessMenuChar (see page 16080)	Responds to keyboard input for the menu.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Handle (see page 16083)	Provides access to the Windows menu handle for the menu.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	PopupComponent (see page 16181)	Indicates the component that last displayed the popup menu in response to a right mouse click.
R	PopupPoint (see page 16183)	Indicates the location where the pop-up menu appears.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	WindowHandle (see page 16089)	Provides access to the handle of the window that uses the menu.

Published Events

	Name	Description
	OnChange (see page 16184)	Occurs when the menu changes.
	OnPopup (see page 16184)	Occurs just before the pop-up menu appears.

Published Properties

	Name	Description
	Alignment (see page 16176)	Determines where the pop-up menu appears when the user clicks the right mouse button.
	AutoHotkeys (see page 16176)	Determines whether the accelerator keys for menu items can be reset automatically.
	AutoLineReduction (see page 16177)	Determines whether redundant separator bars are automatically removed from the menu.
	AutoPopup (see page 16178)	Determines whether the pop-up menu appears automatically when the user clicks the right mouse button.
	BiDiMode (see page 16178)	Determines the layout of the menu in Middle East locales.
	HelpContext (see page 16179)	Specifies a context ID for the entire pop-up menu.
	Images (see page 16179)	Lists the images that can appear beside individual menu items.
R	Items (see page 16084)	Describes the elements of the menu.
	MenuAnimation (see page 16180)	Specifies how the menu appears when it is first displayed.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OwnerDraw (see page 16180)	Specifies whether the menu items are drawn by the application.
	ParentBiDiMode (see page 16180)	Indicates whether the BiDiMode property is inherited from the control that uses the menu.
	Tag (see page 2808)	Stores an integer value as part of a component.
	TrackButton (see page 16183)	Specifies which mouse button activates the popup menu when it is associated with a toolbar button.

95.1.9.2 TPopupMenu.Create Constructor

Creates an instance of TPopupMenu (see page 16169).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TPopupMenu(TComponent * AOwner);
```

Description

Call Create to create to create a pop-up menu at runtime. Pop-up menus placed in forms or data modules at design time are created automatically.

See Also

Create (see page 16072)

95.1.9.3 TPopupMenu.Destroy Destructor

Destroys an instance of TPopupMenu (see page 16169).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPopupMenu();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TPopupMenu (see page 16169) reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

95.1.9.4 TPopupMenu Methods

The methods of the TPopupMenu class are listed here.

Protected Methods

	Name	Description
	DoPopup (see page 16174)	Generates an OnPopup (see page 16184) event.
	SetPopupPoint (see page 16175)	This is SetPopupPoint, a member of class TPopupMenu.
	UseRightToLeftAlignment (see page 16175)	Specifies whether the control's alignment is in a right-to-left mode.

Public Methods

	Name	Description
	Popup (see page 16174)	Displays the pop-up menu onscreen.

95.1.9.4.1 TPopupMenu.DoPopup Method

Generates an OnPopup (see page 16184) event.

Pascal

```
procedure DoPopup(Sender: TObject); virtual;
```

C++

```
virtual __fastcall DoPopup(TObject * Sender);
```

Description

DoPopup is called automatically just before the pop-up menu appears to generate an OnPopup (see page 16184) event. Override DoPopup in a descendant class to change the event.

See Also

OnPopup (see page 16184)

Popup (see page 16174)

95.1.9.4.2 TPopupMenu.Popup Method

Displays the pop-up menu onscreen.

Pascal

```
procedure Popup(X: Integer; Y: Integer); virtual;
```

C++

```
virtual __fastcall Popup(int X, int Y);
```

Description

Call Popup to bring up the pop-up menu. The menu appears at the screen coordinates indicated by the values (in pixels) of X and Y.

Delphi Examples:

```
{
This example uses a pop-up menu. When the user presses the
```

```
mouse button, the pop-up menu appears where the user
clicked the mouse:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := False;
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  PopupMenu1.Popup(Left + X, Top + Y);
end;
```

See Also

Alignment (see page 16176)

AutoPopup (see page 16178)

OnPopup (see page 16184)

95.1.9.4.3 TPopupMenu.SetPopupPoint Method

Pascal

```
procedure SetPopupPoint(APopupPoint: TPoint);
```

C++

```
__fastcall SetPopupPoint(TPoint APopupPoint);
```

Description

This is SetPopupPoint, a member of class TPopupMenu.

95.1.9.4.4 TPopupMenu.UseRightToLeftAlignment Method

Specifies whether the control's alignment is in a right-to-left mode.

Pascal

```
function UseRightToLeftAlignment: Boolean;
```

C++

```
__fastcall Boolean UseRightToLeftAlignment();
```

Description

Call UseRightToLeftAlignment to determine whether the control's alignment is in a right-to-left mode. This information can be used when painting the control or when responding to mouse events.

UseRightToLeftAlignment returns the setting from the control that activated the popup. If the popup was not activated by a control, UseRightToLeftAlignment returns the setting used in the global Application object.

95.1.9.5 TPopupMenu Properties

The properties of the TPopupMenu class are listed here.

Public Properties

	Name	Description
	PopupComponent (see page 16181)	Indicates the component that last displayed the popup menu in response to a right mouse click.
R	PopupPoint (see page 16183)	Indicates the location where the pop-up menu appears.

Published Properties

	Name	Description
	Alignment (see page 16176)	Determines where the pop-up menu appears when the user clicks the right mouse button.
	AutoHotkeys (see page 16176)	Determines whether the accelerator keys for menu items can be reset automatically.
	AutoLineReduction (see page 16177)	Determines whether redundant separator bars are automatically removed from the menu.
	AutoPopup (see page 16178)	Determines whether the pop-up menu appears automatically when the user clicks the right mouse button.
	BiDiMode (see page 16178)	Determines the layout of the menu in Middle East locales.
	HelpContext (see page 16179)	Specifies a context ID for the entire pop-up menu.
	Images (see page 16179)	Lists the images that can appear beside individual menu items.
	MenuAnimation (see page 16180)	Specifies how the menu appears when it is first displayed.
	OwnerDraw (see page 16180)	Specifies whether the menu items are drawn by the application.
	ParentBiDiMode (see page 16180)	Indicates whether the BiDiMode property is inherited from the control that uses the menu.
	TrackButton (see page 16183)	Specifies which mouse button activates the popup menu when it is associated with a toolbar button.

95.1.9.5.1 TPopupMenu.Alignment Property

Determines where the pop-up menu appears when the user clicks the right mouse button.

Pascal

```
property Alignment: TPopupAlignment;
```

C++

```
__property TPopupAlignment Alignment;
```

Description

Set Alignment to control the placement of the popup menu. Alignment controls whether the menu appears to the left of the mouse, to the right of the mouse, or centered over the mouse.

By default, the menu appears with its top left corner under the mouse (paLeft).

See Also

AutoPopup (see page 16178)

OnPopup (see page 16184)

95.1.9.5.2 TPopupMenu.AutoHotkeys Property

Determines whether the accelerator keys for menu items can be reset automatically.

Pascal

```
property AutoHotkeys: TMenuAutoFlag;
```

C++

```
__property TMenuAutoFlag AutoHotkeys;
```

Description

Set AutoHotkeys to indicate whether the menu should automatically ensure that accelerator keys for its items are reset if necessary so that every menu item has an accelerator key and no two items have the same accelerator key. This ensures that menu items added dynamically at runtime can have accelerator keys that do not introduce conflicts. TMenu tries to preserve all existing accelerator key mappings when adjusting the accelerator keys, but may be forced to change some in order to ensure that every item has a unique accelerator.

Before the menu is displayed, the menu checks and adjusts the accelerator key mappings if AutoHotkeys is maAutomatic. When AutoHotkeys is maManual, the menu does not alter the top-level menu items, but may adjust the accelerator keys for submenus, depending on the AutoHotkeys property of the menu item that is the parent of the submenu.

When AutoHotkeys is maManual, an application can still adjust the menu items by calling the Items property's RethinkHotkeys method. AutoHotkeys merely controls whether this adjustment occurs automatically.

Note: The AutoHotkeys property of a menu is linked to the AutoHotkeys property of the TMenuItems object that implements its Items property. Setting one sets the other.

See Also

AutoHotkeys (see page 16134)

Items (see page 16084)

RethinkHotkeys (see page 16128)

95.1.9.5.3 TPopupMenu.AutoLineReduction Property

Determines whether redundant separator bars are automatically removed from the menu.

Pascal

```
property AutoLineReduction: TMenuAutoFlag;
```

C++

```
__property TMenuAutoFlag AutoLineReduction;
```

Description

Set AutoLineReduction to indicate whether the menu should automatically ensure that the menu does not start or end with a separator bar and that two separator bars do not appear next to each other. This ensures that menus built dynamically at runtime do not have separator bars in undesired positions.

Before the menu is displayed, it removes superfluous separator bars when AutoLineReduction is maAutomatic. When AutoLineReduction is maManual, the menu does not remove separators from the top-level menu items, but may remove superfluous separators from submenus, depending on the AutoLineReduction property of the menu item that is the parent of the submenu.

When AutoLineReduction is maManual, an application can still remove superfluous separators by calling the Items property's RethinkLines method. AutoLineReduction merely determines whether this adjustment occurs automatically.

Note: The AutoLineReduction property of a menu is linked to the AutoLineReduction property of the TMenuItems object that implements its Items property. Setting one sets the other.

See Also

AutoLineReduction (see page 16135)

Items (see page 16084)

RethinkLines (see page 16129)

95.1.9.5.4 TPopupMenu.AutoPopup Property

Determines whether the pop-up menu appears automatically when the user clicks the right mouse button.

Pascal

```
property AutoPopup: Boolean;
```

C++

```
__property Boolean AutoPopup;
```

Description

Set AutoPopup to true to cause the pop-up menu to appear automatically when the user clicks the right mouse button on a control that has the pop-up menu specified as its PopupMenu property. The application key present on keyboards specifically designed for 32-bit Windows will also show this menu when pressed.

Set AutoPopup to false if the appearance of the pop-up menu should be controlled programmatically. To display a pop-up menu when AutoPopup is false, use the Popup (see page 16174) method.

Delphi Examples:

```
{
This example uses a pop-up menu. When the user presses the
mouse button, the pop-up menu appears where the user
clicked the mouse:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := False;
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  PopupMenu1.Popup(Left + X, Top + Y);
end;
```

See Also

Popup (see page 16174)

OnPopup (see page 16184)

PopupMenu (see page 5835)

95.1.9.5.5 TPopupMenu.BiDiMode Property

Determines the layout of the menu in Middle East locales.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Set BiDiMode to control the layout of menu items. BiDiMode controls whether the menu items read from left to right or right to left.

See Also

ParentBiDiMode (see page 16089)

IsRightToLeft (see page 16077)

95.1.9.5.6 TPopupMenu.HelpContext Property

Specifies a context ID for the entire pop-up menu.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

Use HelpContext to associate a help screen with the entire pop-up menu. Each screen in the Help system should have a unique context ID. When a component is selected in the application, pressing F1 displays the Help screen associated with the value of HelpContext.

To associate a help screen with the individual items in the pop-up menu, use the HelpContext property of the associated TMenuItem (see page 16102) objects.

See Also

GetHelpContext (see page 16077)

HelpFile (see page 13711)

HelpContext (see page 16148)

95.1.9.5.7 TPopupMenu.Images Property

Lists the images that can appear beside individual menu items.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Use Images to hold the set of images that can appear beside menu items in the menu. Individual menu items set their ImageIndex property to indicate which image in the list appears to the left of the menu item caption.

Images in the list should be 16x16. To accommodate every system on which your application runs, it is recommended that you do not use 256 colors.

Tip: If the application uses action lists, this property should be the value of the action list's Images property. This way, menu items and buttons that do the same thing can display the same image.

See Also

ImageIndex (see page 16148)

Images (see page 325)

Items (see page 16084)

95.1.9.5.8 TPopupMenu.MenuAnimation Property

Specifies how the menu appears when it is first displayed.

Pascal

```
property MenuAnimation: TMenuAnimation;
```

C++

```
__property TMenuAnimation MenuAnimation;
```

Description

Use MenuAnimation to specify how the popup menu appears on the screen. The menu can simply appear as a whole, or can slide into view like a window shade being drawn from any edge.

Note: MenuAnimation only has an effect when running on Windows 98, NT 5.0, or later.

See Also

Popup (see page 16174)

95.1.9.5.9 TPopupMenu.OwnerDraw Property

Specifies whether the menu items are drawn by the application.

Pascal

```
property OwnerDraw: Boolean;
```

C++

```
__property Boolean OwnerDraw;
```

Description

Set OwnerDraw to true to indicate that the application will draw menu items in an OnDrawItem event handler. When OwnerDraw is true, menu items receive an OnMeasureItem and an OnDrawItem event when they need to be rendered on screen.

Set OwnerDraw to false to accept the default drawing. When OwnerDraw is false, menu items display the caption with an optional image to the left (If the Images and ImageIndex properties are set).

Owner-draw menus are useful for displaying the value that appears when selecting a menu item. For example, in a menu that allows users to choose a color, an owner-draw menu could display rectangles drawn in the designated color, rather than color names.

See Also

OnMeasureItem (see page 16160)

OnDrawItem (see page 16159)

Images (see page 16083)

95.1.9.5.10 TPopupMenu.ParentBiDiMode Property

Indicates whether the BiDiMode property is inherited from the control that uses the menu.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

By default, ParentBiDiMode is true, indicating that the layout of menu items follows the BiDiMode setting of the associated form or control. Change ParentBiDiMode to false to override the default BiDiMode setting and control the menu's layout independently.

See Also

BiDiMode (see page 16083)

95.1.9.5.11 TPopupMenu.PopupComponent Property

Indicates the component that last displayed the popup menu in response to a right mouse click.

Pascal

```
property PopupComponent: TComponent;
```

C++

```
__property TComponent * PopupComponent;
```

Description

Read PopupComponent to determine which control is currently using the popup menu. In applications where multiple controls share the same pop-up menu, use PopupComponent to determine which of them displayed the menu.

Set PopupComponent to associate a control with the menu before calling the Popup (see page 16174) method programmatically to bring up the pop-up menu.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
```

```
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;
```

```
procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
```

See Also

AutoPopup (see page 16178)

OnPopup (see page 16184)

Popup (see page 16174)

PopupMenu (see page 5835)

95.1.9.5.12 TPopupMenu.PopupPoint Property

Indicates the location where the pop-up menu appears.

Pascal

```
property PopupPoint: TPoint;
```

C++

```
__property TPoint PopupPoint;
```

Description

This public property is used internally to locate the popup menu. Its value is set by the Popup (see page 16174) method.

See Also

Popup (see page 16174)

95.1.9.5.13 TPopupMenu.TrackButton Property

Specifies which mouse button activates the popup menu when it is associated with a toolbar button.

Pascal

```
property TrackButton: TTrackButton;
```

C++

```
__property TTrackButton TrackButton;
```

Description

Use TrackButton to specify which mouse button activates the popup menu when it is associated with a toolbar button.

95.1.9.6 TPopupMenu Events

The events of the TPopupMenu class are listed here.

Published Events

	Name	Description
	OnChange (see page 16184)	Occurs when the menu changes.
	OnPopup (see page 16184)	Occurs just before the pop-up menu appears.

95.1.9.6.1 TPopupMenu.OnChange Event

Occurs when the menu changes.

Pascal

```
property OnChange: TMenuChangeEvent;
```

C++

```
__property TMenuChangeEvent OnChange;
```

Description

Write an OnChange event handler to respond to changes in the menu's structure. OnChange occurs when the menu is loaded into memory, and in response to property changes that affect the menu's structure.

See Also

Items (see page 16084)

95.1.9.6.2 TPopupMenu.OnPopup Event

Occurs just before the pop-up menu appears.

Pascal

```
property OnPopup: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPopup;
```

Description

Write an OnPopup event handler to take specific action just before the pop-up menu appears. For example, use an OnPopup event handler to set the Checked, Enabled, or Visible property of individual items in the menu so that they are appropriate to the PopupComponent (see page 16181).

Delphi Examples:

```
{
The following code enables the Copy item from the pop-up
menu if the edit control has a nonempty selection.
Note: A portion of the edit control text must be highlighted.
}
procedure TForm1.PopupMenu1Popup(Sender: TObject);
begin
  Copy1.Enabled := Edit1.SelLength > 0;
end;

procedure TForm1.Copy1Click(Sender: TObject);
begin
```

```
    ListBox1.Items.Add('"Copy" menu');
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  PopupMenu1.Popup(Left + X, Top + Y);
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

PopupComponent (see page 16181)

PopupMenu (see page 5835)

95.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AnsiSameCaption (see page 16186)	Indicates whether two strings are the same based on the current locale, ignoring case information and accelerator characters.
	DrawMenuItem (see page 16187)	This is function Menus.DrawMenuItem.
	GetHotkey (see page 16187)	Returns the last character in a string marked with an ampersand (&).
	IsAltGRPressed (see page 16187)	This is function Menus.IsAltGRPressed.
	NewItem (see page 16188)	Creates and initializes a menu item.
	NewLine (see page 16189)	Creates a menu item for a separator.
	NewMenu (see page 16190)	Creates and initializes a main menu.
	NewPopupMenu (see page 16191)	Creates and initializes a popup menu.
	NewSubMenu (see page 16192)	Creates and initializes a menu item that represents a submenu.
	ShortCut (see page 16193)	Creates a menu shortcut for a specified key combination.
	ShortCutFromMessage (see page 16194)	This is function Menus.ShortCutFromMessage.
	ShortCutToKey (see page 16194)	Returns the key code and shift state of a menu shortcut.
	ShortCutToText (see page 16195)	Converts a menu shortcut into a string that describes the shortcut.
	StripHotkey (see page 16196)	This is function Menus.StripHotkey.
	TextToShortCut (see page 16196)	Creates a menu shortcut from a text string.

95.2.1 Menus.AnsiSameCaption Function

Indicates whether two strings are the same based on the current locale, ignoring case information and accelerator characters.

Pascal

```
function AnsiSameCaption(const Text1: string; const Text2: string): Boolean;
```

C++

```
Boolean AnsiSameCaption(const AnsiString Text1, const AnsiString Text2);
```

File

Menus

Description

AnsiSameCaption compares Text1 to Text2, without case sensitivity and ignoring any ampersands (&) that are included to create accelerator characters. The compare operation is controlled by the current locale. The return value is true if the two strings have the same value, false otherwise.

See Also

StringReplace

IsAccel

AnsiSameText

WideSameCaption

95.2.2 Menus.DrawMenuItem Function

Pascal

```
procedure DrawMenuItem(MenuItem: TMenuItem; ACanvas: TCanvas; ARect: TRect; State:
TOwnerDrawState);
```

C++

```
DrawMenuItem(TMenuItem MenuItem, TCanvas ACanvas, TRect ARect, TOwnerDrawState State);
```

File

Menus

Description

This is function Menus.DrawMenuItem.

95.2.3 Menus.GetHotkey Function

Returns the last character in a string marked with an ampersand (&).

Pascal

```
function GetHotkey(const Text: string): string;
```

C++

```
AnsiString GetHotkey(const AnsiString Text);
```

File

Menus

Description

GetHotKey searches Text, starting at the end, for a character that is identified as an accelerator key by a preceding ampersand (&). If it locates a character marked with an ampersand, GetHotKey returns that character. Otherwise, it returns an empty string.

See Also

AnsiSameCaption (see page 16186)

StripHotKey

IsAccel

95.2.4 Menus.IsAltGRPressed Function

Pascal

```
function IsAltGRPressed: boolean;
```

C++

```
boolean IsAltGRPressed();
```

File

Menus

Description

This is function Menus.IsAltGRPressed.

95.2.5 Menus.NewItem Function

Creates and initializes a menu item.

Pascal

```
function NewItem(const ACaption: string; AShortCut: TShortCut; AChecked: Boolean; AEnabled:
Boolean; AOnClick: TNotifyEvent; hCtx: THelpContext; const AName: string): TMenuItem;
```

C++

```
TMenuItem NewItem(const AnsiString ACaption, TShortCut AShortCut, Boolean AChecked, Boolean
AEnabled, TNotifyEvent AOnClick, THelpContext hCtx, const AnsiString AName);
```

File

Menus

Description

Call NewItem to create and initialize a menu item when constructing a menu programmatically.

The ACaption parameter specifies the string that should appear for the menu item.

The AShortCut parameter specifies any associated shortcut.

The AChecked parameter indicates whether the menu item appears with a check mark.

The AEnabled parameter specifies whether the menu item is enabled or grayed.

The AOnClick parameter specifies an event handler that executes when the user clicks the menu item.

The hCtx parameter specifies the help context ID for the menu item.

The AName parameter specifies the name of the menu item, which can be used to refer to the menu item in code.

The menu item returned by NewItem can be added to a menu using the target parent's Add method. If the entire menu is being created dynamically at runtime, the new menu item can be added to an array that is passed as a parameter to the NewMenu (see page 16190), NewPopupMenu (see page 16191), or NewSubMenu (see page 16192) function.

Warning: The item returned by NewItem does not have an owner. You are responsible for freeing its memory when it is no longer needed. The Delete and Remove methods of TMenuItem (see page 16102) do not free memory.

C++ Examples:

```
/*
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *Item1 = NewItem("New item", TextToShortCut("Ctrl+N"), false, true, MyOnExecute,
0, "Item1");
```

```
  Menu = NewMenu(this, "MyMenu", &Item1, 0);
}

void __fastcall TForm1::MyOnExecute(TObject *Sender)
{
  MessageDlg("Item1 has been executed.", mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Menu := NewMenu(Self, 'MyMenu',
            [NewItem('New item', TextToShortCut('Ctrl+N'),
            False, True, MyOnExecute, 0, 'Item1')]);
end;

procedure TForm1.MyOnExecute(Sender: TObject);
begin
  MessageDlg('Item1 has been executed.', mtInformation, [mbOK], 0);
end;
```

See Also

NewSubMenu (see page 16192)

NewLine (see page 16189)

NewMenu (see page 16190)

NewPopupMenu (see page 16191)

TMenuItem (see page 16102)

TMenuItem (see page 16102)

TMenuItem (see page 16102)

95.2.6 Menus.NewLine Function

Creates a menu item for a separator.

Pascal

```
function NewLine: TMenuItem;
```

C++

```
TMenuItem NewLine();
```

File

Menus

Description

Call NewLine to create a separator that divides the items in a menu into logical groups. Once the menu item for the separator has been created using NewLine, it can be added to a menu using the target parent's Add method. If the entire menu is being created dynamically at runtime, the new menu item can be added to an array that is passed as a parameter to the NewMenu (see page 16190), NewPopupMenu (see page 16191), or NewSubMenu (see page 16192) function.

Warning: The item returned by NewLine does not have an owner. You are responsible for freeing its memory when it is no longer needed. The Delete and Remove methods of TMenuItem (see page 16102) do not free memory.

See Also

NewItem (see page 16188)

NewPopupMenu (see page 16191)

NewSubMenu (see page 16192)

TMenuItem (see page 16102)

TMenuItem (see page 16102)

TMenuItem (see page 16102)

95.2.7 Menus.NewMenu Function

Creates and initializes a main menu.

Pascal

```
function NewMenu(Owner: TComponent; const AName: string; const Items: array of TMenuItem):
TMainMenu;
```

C++

```
TMainMenu NewMenu(TComponent * Owner, const AnsiString AName, const array of TMenuItem Items);
```

File

Menus

Description

Call NewMenu to create and initialize a main menu programmatically. NewMenu returns the TMainMenu (see page 16056) component that represents the menu.

The Owner parameter specifies the component that is responsible for freeing the menu (typically the form).

The AName parameter specifies the name of the menu, which is used to refer to it in code.

The Items parameter is an array of menu items that make up the top level menu items. To create the menu items for the Items parameter, use the NewItem (see page 16188) function.

Note: The Items_Size parameter specifies the index of the last menu item in Items (one less than the number of menu items).

C++ Examples:

```
/*
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *Item1 = NewItem("New item", TextToShortCut("Ctrl+N"), false, true, MyOnExecute,
0, "Item1");
  Menu = NewMenu(this, "MyMenu", &Item1, 0);
}
```

```
void __fastcall TForm1::MyOnExecute(TObject *Sender)
{
  MessageDlg("Item1 has been executed.", mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Menu := NewMenu(Self, 'MyMenu',
            [NewItem('New item', TextToShortCut('Ctrl+N'),
            False, True, MyOnExecute, 0, 'Item1')]);
end;

procedure TForm1.MyOnExecute(Sender: TObject);
begin
  MessageDlg('Item1 has been executed.', mtInformation, [mbOK], 0);
end;
```

See Also

NewPopupMenu (see page 16191)

NewSubMenu (see page 16192)

NewItem (see page 16188)

95.2.8 Menus.NewPopupMenu Function

Creates and initializes a popup menu.

Pascal

```
function NewPopupMenu(Owner: TComponent; const AName: string; Alignment: TPopupAlignment;
AutoPopup: Boolean; const Items: array of TMenuItem): TPopupMenu;
```

C++

```
TPopupMenu NewPopupMenu(TComponent * Owner, const AnsiString AName, TPopupAlignment Alignment,
Boolean AutoPopup, const array of TMenuItem Items);
```

File

Menus

Description

Call NewPopupMenu to create and initialize a popup menu programmatically. NewPopupMenu returns the TPopupMenu (see page 16169) component that represents the popup menu, but does not associate that popup menu with any control.

The Owner parameter specifies the component that is responsible for freeing the menu.

The AName parameter specifies the name of the popup menu, which is used to refer to it in code.

The Alignment parameter determines where the menu appears relative to the mouse when the user right clicks on the control that uses the menu.

The AutoPopup parameter specifies whether the menu should appear automatically when the user right clicks the associated control.

The Items parameter is an array of menu items that make up the top level menu items. To create the menu items for the Items parameter, use the NewItem (see page 16188) function.

Note: The Items_Size parameter specifies the index of the last menu item in Items (one less than the number of menu items).

See Also

NewMenu (see page 16190)

NewSubMenu (see page 16192)

NewItem (see page 16188)

95.2.9 Menus.NewSubMenu Function

Creates and initializes a menu item that represents a submenu.

Pascal

```
function NewSubMenu(const ACaption: string; hCtx: THelpContext; const AName: string; const
Items: array of TMenuItem; AEnabled: Boolean = True): TMenuItem;
```

C++

```
TMenuItem NewSubMenu(const AnsiString ACaption, THelpContext hCtx, const AnsiString AName,
const array of TMenuItem Items, Boolean AEnabled = True);
```

File

Menus

Description

Call NewSubMenu to create a menu item that represents a submenu when constructing a menu programmatically at runtime.

The ACaption parameter indicates the string that appears in the parent menu for the submenu.

The hCtx parameter specifies the help context ID for the menu item.

The AName parameter specifies the name of the menu item, which can be used to refer to the menu item in code.

The Items parameter specifies an array of menu items that make up the submenu and that appear when the user clicks the submenu item in the parent menu.

Note: The Items_Size parameter specifies the index of the last menu item in Items (one less than the number of menu items).

The AEnabled parameter specifies whether the menu item that appears in the parent menu is enabled.

Once a submenu has been created using NewSubMenu, it can be added to a menu using the target parent's Add method. If the entire menu is being created dynamically at runtime, the new menu item can be added to an array that is passed as a parameter to the NewMenu (see page 16190), NewPopupMenu (see page 16191), or NewSubMenu function.

Warning: The item returned by NewSubMenu does not have an owner. You are responsible for freeing its memory when it is no longer needed. The Delete and Remove methods of TMenuItem (see page 16102) do not free memory.

See Also

NewItem (see page 16188)

TMenuItem (see page 16102)

NewLine (see page 16189)

NewMenu (see page 16190)

NewPopupMenu (see page 16191)

TMenuItem (see page 16102)

TMenuItem (see page 16102)

TMenuItem (see page 16102)

95.2.10 Menus.ShortCut Function

Creates a menu shortcut for a specified key combination.

Pascal

```
function ShortCut(Key: Word; Shift: TShiftState): TShortCut;
```

C++

```
TShortCut ShortCut(Word Key, TShiftState Shift);
```

File

Menus

Description

Call ShortCut to create a menu shortcut at runtime.

Specify a Key value using a virtual key code (Menus (see page 16053) version) or Qt key code (QMenus version).

Specify the values for the Shift, Control, and Alt keys by using a set of type TShiftState. For example, to specify the Shift key, use the set [ssShift]. To specify a Shift and Ctrl key combination, use the set [ssShift, ssCtrl].

Assign the value returned by ShortCut as the value of the ShortCut property of a menu item.

Note: TextToShortCut (see page 16196) can also create a shortcut. TextToShortCut (see page 16196) converts a string to a shortcut, so it can be used to let the user specify the shortcut. However, TextToShortCut (see page 16196) executes much more slowly than ShortCut. Use ShortCut whenever possible.

Delphi Examples:

```
{
This line creates a shortcut, Ctrl+O, at run time and
assigns it to the Open command on a File menu.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  OpenCommand.ShortCut := Menus.ShortCut(Word('O'), [ssCtrl]);
end;

procedure TForm1.OpenCommandClick(Sender: TObject);
begin
  MessageDlg('The Open command has been selected.', mtInformation, [mbOk], 0);
end;
```

See Also

ShortCutToKey (see page 16194)

ShortCutToText (see page 16195)

TextToShortCut (see page 16196)

TMenuItem (see page 16102)

95.2.11 Menus.ShortCutFromMessage Function

Pascal

```
function ShortCutFromMessage(Message: TWMKey): TShortCut;
```

C++

```
TShortCut ShortCutFromMessage(TWMKey Message);
```

File

Menus

Description

This is function Menus.ShortCutFromMessage.

95.2.12 Menus.ShortCutToKey Function

Returns the key code and shift state of a menu shortcut.

Pascal

```
procedure ShortCutToKey(ShortCut: TShortCut; var Key: Word; var Shift: TShiftState);
```

C++

```
ShortCutToKey(TShortCut ShortCut, Word Key, TShiftState Shift);
```

File

Menus

Description

Call ShortCutToKey to parse a menu shortcut into its virtual key code and shift state parts.

ShortCut (see page 16193) is the value of the menu shortcut.

Key returns the virtual key code (Menus (see page 16053) version) or Qt key code (QMenus version) for the key.

Shirt returns the associated combination of Shift, Ctrl, and Alt keys.

Delphi Examples:

```
{
The following code redefines the ShortCut of CloseCommand if
the original short cut used the [ssCtrl] shift state.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  TheKey: Word;
  TheShiftState: TShiftState;
begin
  Menus.ShortCutToKey(CloseCommand.ShortCut, TheKey, TheShiftState);
  if TheShiftState = [ssCtrl] then
   CloseCommand.ShortCut := ShortCut(Word('C'), [ssShift]);
end;
```

```
procedure TForm1.CloseCommandClick(Sender: TObject);
begin
  MessageDlg('The Close command has been selected.', mtInformation, [mbOk], 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  CloseCommand.ShortCut := Menus.ShortCut(Word('C'), [ssCtrl]);
end;
```

See Also

ShortCut (see page 16193)

ShortCutToText (see page 16195)

TextToShortCut (see page 16196)

TMenuItem (see page 16102)

95.2.13 Menus.ShortCutToText Function

Converts a menu shortcut into a string that describes the shortcut.

Pascal

```
function ShortCutToText(ShortCut: TShortCut): string;
```

C++

```
AnsiString ShortCutToText(TShortCut ShortCut);
```

File

Menus

Description

Call ShortCutToText to obtain a text description of the shortcut associated with a menu item. Use this function when the application needs to display a description of a shortcut to the user.

Delphi Examples:

```
{
This line creates a shortcut, Ctrl+O, at run time and
assigns it to the Open command on a File menu.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.Text := Menus.ShortCutToText(OpenCommand.ShortCut);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  OpenCommand.ShortCut := Menus.ShortCut(Word('O'), [ssCtrl]);
end;

procedure TForm1.OpenCommandClick(Sender: TObject);
begin
  MessageDlg('The Open command has been selected.', mtInformation, [mbOk], 0);
end;
```

See Also

ShortCut (see page 16193)

ShortCutToKey (see page 16194)

TextToShortCut (see page 16196)

TMenuItem (see page 16102)

95.2.14 Menus.StripHotkey Function

Pascal

```
function StripHotkey(const Text: string): string;
```

C++

```
AnsiString StripHotkey(const AnsiString Text);
```

File

Menus

Description

This is function Menus.StripHotkey.

95.2.15 Menus.TextToShortCut Function

Creates a menu shortcut from a text string.

Pascal

```
function TextToShortCut(Text: string): TShortCut;
```

C++

```
TShortCut TextToShortCut(AnsiString Text);
```

File

Menus

Description

Call TextToShortCut to create a menu shortcut from a descriptive string. For example, an application can allow users to specify the value of a shortcut in an edit box control, and then call TextToShortCut to create a menu shortcut for the value entered by the user.

Note: TextToShortCut executes slowly. When possible, use the ShortCut (see page 16193) function to create a menu shortcut instead.

C++ Examples:

```
/*
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMenuItem *Item1 = NewItem("New item", TextToShortCut("Ctrl+N"), false, true, MyOnExecute,
```

```
0, "Item1");
  Menu = NewMenu(this, "MyMenu", &Item1, 0);
}

void __fastcall TForm1::MyOnExecute(TObject *Sender)
{
  MessageDlg("Item1 has been executed.", mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This code displays a new menu named MyMenu when the user
clicks the button. MyOnExecute must be included in the
definition of the TForm1 class.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Menu := NewMenu(Self, 'MyMenu',
            [NewItem('New item', TextToShortCut('Ctrl+N'),
            False, True, MyOnExecute, 0, 'Item1')]);
end;

procedure TForm1.MyOnExecute(Sender: TObject);
begin
  MessageDlg('Item1 has been executed.', mtInformation, [mbOK], 0);
end;
```

See Also

ShortCut (see page 16193)

ShortCutToText (see page 16195)

TMenuItem (see page 16102)

95.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TFindItemKind (see page 16198)	TFindItemKind indicates the criterion to use when locating menu items.
	TMenuAnimations (see page 16199)	TMenuAnimation (see page 16204) and TMenuAnimations indicate how a popup menu appears on screen.
	TMenuBreak (see page 16199)	TMenuBreak indicates the criterion to use when locating menu items.
	TMenuItemAutoFlag (see page 16200)	TMenuItemAutoFlag and TMenuAutoFlag (see page 16205) Indicate when menu items are automatically maintained.
	TPopupAlignment (see page 16201)	TPopupAlignment represents the position of a popup menu relative to the mouse.
	TTrackButton (see page 16201)	TTrackButton specifies which mouse button activates a tool button menu.

95.3.1 Menus.TFindItemKind Enumeration

TFindItemKind indicates the criterion to use when locating menu items.

Pascal

```
TFindItemKind = (
  fkCommand,
  fkHandle,
  fkShortCut
);
```

C++

```
enum TFindItemKind {
  fkCommand,
  fkHandle,
  fkShortCut
};
```

File

Menus

Description

TFindItemKind values indicate the criterion to use when locating a menu item. The following table gives the possible values:

Value	Meaning
fkID	Identify the item by its Windows command ID (parameter to WM_COMMAND message).
fkHandle	Identify the item by its pop-up menu handle.
fkShortCut	Identify the item by its menu shortcut key code.

95.3.2 Menus.TMenuAnimations Enumeration

TMenuAnimation (see page 16204) and TMenuAnimations indicate how a popup menu appears on screen.

Pascal

```
TMenuAnimations = (
  maLeftToRight,
  maRightToLeft,
  maTopToBottom,
  maBottomToTop,
  maNone
);
```

C++

```
enum TMenuAnimations {
  maLeftToRight,
  maRightToLeft,
  maTopToBottom,
  maBottomToTop,
  maNone
};
```

File

Menus

Description

TMenuAnimations indicates the direction in which a menu can "slide" into view. TMenuAnimation (see page 16204) is a set of TMenuAnimations values, which can be used to specify more than one direction (for a popup menu that slides in at a diagonal). Note that some TMenuAnimations values are mutually exclusive (for example maLeftToRight and maRightToLeft).

Value	Meaning
maLeftToRight	The popup menu appears like a window shade that pulls from left to right.
maRightToLeft	The popup menu appears like a window shade that pulls from right to left.
maTopToBottom	The popup menu appears like a window shade that pulls from top to bottom.
maBottomToTop	The popup menu appears like a window shade that pulls from bottom to top.
maNone	The popup menu appears as a whole with no animation effects.

See Also

TPopupMenu

95.3.3 Menus.TMenuBreak Enumeration

TMenuBreak indicates the criterion to use when locating menu items.

Pascal

```
TMenuBreak = (
  mbNone,
  mbBreak,
  mbBarBreak
```

```
);
```

C++

```
enum TMenuBreak {
  mbNone,
  mbBreak,
  mbBarBreak
};
```

File

Menus

Description

TMenuBreak values indicate the criterion to use when locating a menu item. The following table gives the possible values:

Value	Meaning
mbNone	No menu breaking occurs.
mbBarBreak	The Parent menu breaks into another column with the menu item appearing at the top of the new column. A bar separates the new and the old columns.
mbBreak	The Parent menu breaks into another column with the menu item appearing at the top of the new column. Only space separates the new and the old columns.

95.3.4 Menus.TMenuItemAutoFlag Enumeration

TMenuItemAutoFlag and TMenuAutoFlag (see page 16205) Indicate when menu items are automatically maintained.

Pascal

```
TMenuItemAutoFlag = (
  maAutomatic,
  maManual,
  maParent
);
```

C++

```
enum TMenuItemAutoFlag {
  maAutomatic,
  maManual,
  maParent
};
```

File

Menus

Description

TMenuItemAutoFlag indicates when a submenu is checked for problems and updated. TMenuAutoFlag (see page 16205) provides the same information for top-level menu items. The following table lists the possible values:

Values	Meaning
maAutomatic	The immediate submenu of the menu item or the top-level menu items of the menu are checked for problems and automatically corrected when the menu is displayed at runtime.
maManual	The immediate submenu of the menu item or the top-level menu items of the menu are checked for problems and automatically corrected only when the application specifically calls a method of the menu item to do so.

maParent	Automatic checking and correction of the immediate submenu follows the same rule as the menu item's parent. This value is not applicable to menus, which have no parent menu item.

See Also

AutoHotkeys

AutoLineReduction

95.3.5 Menus.TPopupAlignment Enumeration

TPopupAlignment represents the position of a popup menu relative to the mouse.

Pascal

```
TPopupAlignment = (
  paLeft,
  paRight,
  paCenter
);
```

C++

```
enum TPopupAlignment {
  paLeft,
  paRight,
  paCenter
};
```

File

Menus

Description

These are the possible values:

Value	Meaning
paLeft	The pop-up menu appears with its top left corner under the mouse.
paCenter	The pop-up menu appears with the top center of the menu under the mouse.
paRight	The pop-up menu appears with its top right corner under the mouse.

95.3.6 Menus.TTrackButton Enumeration

TTrackButton specifies which mouse button activates a tool button menu.

Pascal

```
TTrackButton = (
  tbRightButton,
  tbLeftButton
);
```

C++

```
enum TTrackButton {
```

```
    tbRightButton,
    tbLeftButton
  };
```

File

Menus

Description

TTrackButton can have one of the following values:

Value	Meaning
tbRightButton	The right mouse button activates the tool button menu.
tbLeftButton	The left mouse button activates the tool button menu.

95.4 Types

The following table lists types in this documentation.

Types

Name	Description
PMenuItem (see page 16203)	This is type Menus.PMenuItem.
TAdvancedMenuDrawItemEvent (see page 16203)	TAdvancedMenuDrawItemEvent is the type for event handlers that render menu items in owner-drawn menus.
TMenuActionLinkClass (see page 16204)	This is type Menus.TMenuActionLinkClass.
TMenuAnimation (see page 16204)	TMenuAnimation and TMenuAnimations (see page 16199) indicate how a popup menu appears on screen.
TMenuAutoFlag (see page 16205)	TMenuItemAutoFlag (see page 16200) and TMenuAutoFlag Indicate when menu items are automatically maintained.
TMenuChangeEvent (see page 16205)	This is type Menus.TMenuChangeEvent.
TMenuDrawItemEvent (see page 16206)	TMenuDrawItemEvent is the type for event handlers that render menu items in owner-drawn menus.
TMenuMeasureItemEvent (see page 16206)	TMenuMeasureItemEvent is the type for event handlers that render menu items in owner-Measuren menus.

95.4.1 Menus.PMenuItem Type

Pascal

```
PMenuItem = ^TMenuItem;
```

C++

```
^TMenuItem PMenuItem;
```

File

Menus

Description

This is type Menus.PMenuItem.

95.4.2 Menus.TAdvancedMenuDrawItemEvent Type

TAdvancedMenuDrawItemEvent is the type for event handlers that render menu items in owner-drawn menus.

Pascal

```
TAdvancedMenuDrawItemEvent = procedure (Sender: TObject; ACanvas: TCanvas; ARect: TRect;
State: TOwnerDrawState) of object;
```

C++

```
(Sender: TObject; ACanvas: TCanvas; ARect: TRect; State: TOwnerDrawState) (
TAdvancedMenuDrawItemEvent)();
```

File

Menus

Description

Sender is the TMenuItem (see page 16102) instance that represents the menu item to be drawn.

ACanvas provides a drawing surface on which to draw the menu item.

ARect indicates the boundaries (in pixels) of the menu item on ACanvas.

State is a set indicating various aspects of the menu items state such as whether it is selected, checked, grayed, and so on.

95.4.3 Menus.TMenuActionLinkClass Type

Pascal

```
TMenuActionLinkClass = class of TMenuActionLink;
```

C++

```
TMenuActionLink TMenuActionLinkClass;
```

File

Menus

Description

This is type Menus.TMenuActionLinkClass.

95.4.4 Menus.TMenuAnimation Type

TMenuAnimation and TMenuAnimations (see page 16199) indicate how a popup menu appears on screen.

Pascal

```
TMenuAnimation = set of TMenuAnimations;
```

C++

```
TMenuAnimations TMenuAnimation;
```

File

Menus

Description

TMenuAnimations (see page 16199) indicates the direction in which a menu can "slide" into view. TMenuAnimation is a set of TMenuAnimations (see page 16199) values, which can be used to specify more than one direction (for a popup menu that slides in at a diagonal). Note that some TMenuAnimations (see page 16199) values are mutually exclusive (for example maLeftToRight and maRightToLeft).

Value	Meaning
maLeftToRight	The popup menu appears like a window shade that pulls from left to right.
maRightToLeft	The popup menu appears like a window shade that pulls from right to left.

maTopToBottom	The popup menu appears like a window shade that pulls from top to bottom.
maBottomToTop	The popup menu appears like a window shade that pulls from bottom to top.
maNone	The popup menu appears as a whole with no animation effects.

See Also

TPopupMenu

95.4.5 Menus.TMenuAutoFlag Type

TMenuItemAutoFlag (see page 16200) and TMenuAutoFlag Indicate when menu items are automatically maintained.

Pascal

```
TMenuAutoFlag = maAutomatic..maManual;
```

C++

```
maAutomatic..maManual TMenuAutoFlag;
```

File

Menus

Description

TMenuItemAutoFlag (see page 16200) indicates when a submenu is checked for problems and updated. TMenuAutoFlag provides the same information for top-level menu items. The following table lists the possible values:

Values	Meaning
maAutomatic	The immediate submenu of the menu item or the top-level menu items of the menu are checked for problems and automatically corrected when the menu is displayed at runtime.
maManual	The immediate submenu of the menu item or the top-level menu items of the menu are checked for problems and automatically corrected only when the application specifically calls a method of the menu item to do so.
maParent	Automatic checking and correction of the immediate submenu follows the same rule as the menu item's parent. This value is not applicable to menus, which have no parent menu item.

See Also

AutoHotkeys

AutoLineReduction

95.4.6 Menus.TMenuChangeEvent Type

Pascal

```
TMenuChangeEvent = procedure (Sender: TObject; Source: TMenuItem; Rebuild: Boolean) of object;
```

C++

```
(Sender: TObject; Source: TMenuItem; Rebuild: Boolean) ( TMenuChangeEvent)();
```

File

Menus

Description

This is type Menus.TMenuChangeEvent.

95.4.7 Menus.TMenuDrawItemEvent Type

TMenuDrawItemEvent is the type for event handlers that render menu items in owner-drawn menus.

Pascal

```
TMenuDrawItemEvent = procedure (Sender: TObject; ACanvas: TCanvas; ARect: TRect; Selected:
Boolean) of object;
```

C++

```
(Sender: TObject; ACanvas: TCanvas; ARect: TRect; Selected: Boolean) ( TMenuDrawItemEvent)();
```

File

Menus

Description

Sender is the TMenuItem (see page 16102) instance that represents the menu item to be drawn.

ACanvas provides a drawing surface on which to draw the menu item.

ARect indicates the boundaries (in pixels) of the menu item on ACanvas.

Selected indicates whether the menu item is selected (that is, if it will be clicked if the user presses return)

95.4.8 Menus.TMenuMeasureItemEvent Type

TMenuMeasureItemEvent is the type for event handlers that render menu items in owner-Measuren menus.

Pascal

```
TMenuMeasureItemEvent = procedure (Sender: TObject; ACanvas: TCanvas; var Width, Height:
Integer) of object;
```

C++

```
(Sender: TObject; ACanvas: TCanvas; var Width, Height: Integer) ( TMenuMeasureItemEvent)();
```

File

Menus

Description

Sender is the TMenuItem (see page 16102) instance that represents the menu item to be measured.

ACanvas represents the canvas where the menu item will be drawn. Use its methods, if necessary, to determine the size of the menu item.

Width and Height return the dimensions, in pixels, that should be used to draw the menu item.

95.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
PopupList (see page 16207)	This is variable Menus.PopupList.
ShortCutItems (see page 16207)	This is variable Menus.ShortCutItems.
ValidMenuHotkeys (see page 16207)	This is variable Menus.ValidMenuHotkeys.

95.5.1 Menus.PopupList Variable

Pascal

```
PopupList: TPopupList;
```

C++

```
TPopupList PopupList;
```

File

Menus

Description

This is variable Menus.PopupList.

95.5.2 Menus.ShortCutItems Variable

Pascal

```
ShortCutItems: TMenuItemStack;
```

C++

```
TMenuItemStack ShortCutItems;
```

File

Menus

Description

This is variable Menus.ShortCutItems.

95.5.3 Menus.ValidMenuHotkeys Variable

Pascal

```
ValidMenuHotkeys: string = '1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ';
```

C++

```
AnsiString ValidMenuHotkeys = '1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ';
```

File

Menus

Description

This is variable Menus.ValidMenuHotkeys.

95.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
cDialogSuffix (see page 16209)	This is constant Menus.cDialogSuffix.
cHotkeyPrefix (see page 16209)	This is constant Menus.cHotkeyPrefix.
cLineCaption (see page 16209)	This is constant Menus.cLineCaption.

95.6.1 Menus.cDialogSuffix Constant

Pascal

```
cDialogSuffix = '...';
```

C++

```
cDialogSuffix = '...';
```

File

Menus

Description

This is constant Menus.cDialogSuffix.

95.6.2 Menus.cHotkeyPrefix Constant

Pascal

```
cHotkeyPrefix = '&';
```

C++

```
cHotkeyPrefix = '&';
```

File

Menus

Description

This is constant Menus.cHotkeyPrefix.

95.6.3 Menus.cLineCaption Constant

Pascal

```
cLineCaption = '-';
```

C++

```
cLineCaption = '-';
```

File

Menus

Description

This is constant Menus.cLineCaption.

96 Messages Namespace

This is namespace Messages.

Topics

Name	Description
Structs, Records, Enums (see page 16223)	The following table lists structs, records, enums in this documentation.
Types (see page 16278)	The following table lists types in this documentation.
Constants (see page 16305)	The following table lists constants in this documentation.

Constants

Name	Description
BM_CLICK (see page 16312)	This is constant Messages.BM_CLICK.
BM_GETCHECK (see page 16313)	This is constant Messages.BM_GETCHECK.
BM_GETIMAGE (see page 16313)	This is constant Messages.BM_GETIMAGE.
BM_GETSTATE (see page 16313)	This is constant Messages.BM_GETSTATE.
BM_SETCHECK (see page 16314)	This is constant Messages.BM_SETCHECK.
BM_SETIMAGE (see page 16314)	This is constant Messages.BM_SETIMAGE.
BM_SETSTATE (see page 16314)	This is constant Messages.BM_SETSTATE.
BM_SETSTYLE (see page 16315)	This is constant Messages.BM_SETSTYLE.
BN_CLICKED (see page 16315)	This is constant Messages.BN_CLICKED.
BN_DBLCLK (see page 16315)	This is constant Messages.BN_DBLCLK.
BN_DISABLE (see page 16316)	This is constant Messages.BN_DISABLE.
BN_DOUBLECLICKED (see page 16316)	This is constant Messages.BN_DOUBLECLICKED.
BN_HILITE (see page 16316)	This is constant Messages.BN_HILITE.
BN_KILLFOCUS (see page 16317)	This is constant Messages.BN_KILLFOCUS.
BN_PAINT (see page 16317)	This is constant Messages.BN_PAINT.
BN_PUSHED (see page 16317)	This is constant Messages.BN_PUSHED.
BN_SETFOCUS (see page 16318)	This is constant Messages.BN_SETFOCUS.
BN_UNHILITE (see page 16318)	This is constant Messages.BN_UNHILITE.
BN_UNPUSHED (see page 16318)	This is constant Messages.BN_UNPUSHED.
CB_ADDSTRING (see page 16319)	This is constant Messages.CB_ADDSTRING.
CB_DELETESTRING (see page 16319)	This is constant Messages.CB_DELETESTRING.
CB_DIR (see page 16319)	This is constant Messages.CB_DIR.
CB_FINDSTRING (see page 16320)	This is constant Messages.CB_FINDSTRING.
CB_FINDSTRINGEXACT (see page 16320)	This is constant Messages.CB_FINDSTRINGEXACT.
CB_GETCOUNT (see page 16320)	This is constant Messages.CB_GETCOUNT.

CB_GETCURSEL (see page 16321)	This is constant Messages.CB_GETCURSEL.
CB_GETDROPPEDCONTROLRECT (see page 16321)	This is constant Messages.CB_GETDROPPEDCONTROLRECT.
CB_GETDROPPEDSTATE (see page 16321)	This is constant Messages.CB_GETDROPPEDSTATE.
CB_GETDROPPEDWIDTH (see page 16322)	This is constant Messages.CB_GETDROPPEDWIDTH.
CB_GETEDITSEL (see page 16322)	This is constant Messages.CB_GETEDITSEL.
CB_GETEXTENDEDUI (see page 16322)	This is constant Messages.CB_GETEXTENDEDUI.
CB_GETHORIZONTALEXTENT (see page 16323)	This is constant Messages.CB_GETHORIZONTALEXTENT.
CB_GETITEMDATA (see page 16323)	This is constant Messages.CB_GETITEMDATA.
CB_GETITEMHEIGHT (see page 16323)	This is constant Messages.CB_GETITEMHEIGHT.
CB_GETLBTEXT (see page 16324)	This is constant Messages.CB_GETLBTEXT.
CB_GETLBTEXTLEN (see page 16324)	This is constant Messages.CB_GETLBTEXTLEN.
CB_GETLOCALE (see page 16324)	This is constant Messages.CB_GETLOCALE.
CB_GETTOPINDEX (see page 16325)	This is constant Messages.CB_GETTOPINDEX.
CB_INITSTORAGE (see page 16325)	This is constant Messages.CB_INITSTORAGE.
CB_INSERTSTRING (see page 16325)	This is constant Messages.CB_INSERTSTRING.
CB_LIMITTEXT (see page 16326)	This is constant Messages.CB_LIMITTEXT.
CB_MSGMAX (see page 16326)	This is constant Messages.CB_MSGMAX.
CB_RESETCONTENT (see page 16326)	This is constant Messages.CB_RESETCONTENT.
CB_SELECTSTRING (see page 16327)	This is constant Messages.CB_SELECTSTRING.
CB_SETCURSEL (see page 16327)	This is constant Messages.CB_SETCURSEL.
CB_SETDROPPEDWIDTH (see page 16327)	This is constant Messages.CB_SETDROPPEDWIDTH.
CB_SETEDITSEL (see page 16328)	This is constant Messages.CB_SETEDITSEL.
CB_SETEXTENDEDUI (see page 16328)	This is constant Messages.CB_SETEXTENDEDUI.
CB_SETHORIZONTALEXTENT (see page 16328)	This is constant Messages.CB_SETHORIZONTALEXTENT.
CB_SETITEMDATA (see page 16329)	This is constant Messages.CB_SETITEMDATA.
CB_SETITEMHEIGHT (see page 16329)	This is constant Messages.CB_SETITEMHEIGHT.
CB_SETLOCALE (see page 16329)	This is constant Messages.CB_SETLOCALE.
CB_SETTOPINDEX (see page 16330)	This is constant Messages.CB_SETTOPINDEX.
CB_SHOWDROPDOWN (see page 16330)	This is constant Messages.CB_SHOWDROPDOWN.
CBN_CLOSEUP (see page 16330)	This is constant Messages.CBN_CLOSEUP.
CBN_DBLCLK (see page 16331)	This is constant Messages.CBN_DBLCLK.
CBN_DROPDOWN (see page 16331)	This is constant Messages.CBN_DROPDOWN.
CBN_EDITCHANGE (see page 16331)	This is constant Messages.CBN_EDITCHANGE.
CBN_EDITUPDATE (see page 16332)	This is constant Messages.CBN_EDITUPDATE.
CBN_ERRSPACE (see page 16332)	This is constant Messages.CBN_ERRSPACE.
CBN_KILLFOCUS (see page 16332)	This is constant Messages.CBN_KILLFOCUS.
CBN_SELCHANGE (see page 16333)	This is constant Messages.CBN_SELCHANGE.
CBN_SELENDCANCEL (see page 16333)	This is constant Messages.CBN_SELENDCANCEL.
CBN_SELENDOK (see page 16333)	This is constant Messages.CBN_SELENDOK.
CBN_SETFOCUS (see page 16334)	This is constant Messages.CBN_SETFOCUS.
DM_GETDEFID (see page 16334)	This is constant Messages.DM_GETDEFID.
DM_REPOSITION (see page 16334)	This is constant Messages.DM_REPOSITION.
DM_SETDEFID (see page 16335)	This is constant Messages.DM_SETDEFID.
EM_CANUNDO (see page 16335)	This is constant Messages.EM_CANUNDO.
EM_CHARFROMPOS (see page 16335)	This is constant Messages.EM_CHARFROMPOS.
EM_EMPTYUNDOBUFFER (see page 16336)	This is constant Messages.EM_EMPTYUNDOBUFFER.
EM_FMTLINES (see page 16336)	This is constant Messages.EM_FMTLINES.
EM_GETFIRSTVISIBLELINE (see page 16336)	This is constant Messages.EM_GETFIRSTVISIBLELINE.
EM_GETHANDLE (see page 16337)	This is constant Messages.EM_GETHANDLE.
EM_GETIMESTATUS (see page 16337)	This is constant Messages.EM_GETIMESTATUS.
EM_GETLIMITTEXT (see page 16337)	This is constant Messages.EM_GETLIMITTEXT.
EM_GETLINE (see page 16338)	This is constant Messages.EM_GETLINE.
EM_GETLINECOUNT (see page 16338)	This is constant Messages.EM_GETLINECOUNT.
EM_GETMARGINS (see page 16338)	This is constant Messages.EM_GETMARGINS.

EM_GETMODIFY (see page 16339)	This is constant Messages.EM_GETMODIFY.
EM_GETPASSWORDCHAR (see page 16339)	This is constant Messages.EM_GETPASSWORDCHAR.
EM_GETRECT (see page 16339)	This is constant Messages.EM_GETRECT.
EM_GETSEL (see page 16340)	This is constant Messages.EM_GETSEL.
EM_GETTHUMB (see page 16340)	This is constant Messages.EM_GETTHUMB.
EM_GETWORDBREAKPROC (see page 16340)	This is constant Messages.EM_GETWORDBREAKPROC.
EM_LIMITTEXT (see page 16341)	This is constant Messages.EM_LIMITTEXT.
EM_LINEFROMCHAR (see page 16341)	This is constant Messages.EM_LINEFROMCHAR.
EM_LINEINDEX (see page 16341)	This is constant Messages.EM_LINEINDEX.
EM_LINELENGTH (see page 16342)	This is constant Messages.EM_LINELENGTH.
EM_LINESCROLL (see page 16342)	This is constant Messages.EM_LINESCROLL.
EM_POSFROMCHAR (see page 16342)	This is constant Messages.EM_POSFROMCHAR.
EM_REPLACESEL (see page 16343)	This is constant Messages.EM_REPLACESEL.
EM_SCROLL (see page 16343)	This is constant Messages.EM_SCROLL.
EM_SCROLLCARET (see page 16343)	This is constant Messages.EM_SCROLLCARET.
EM_SETHANDLE (see page 16344)	This is constant Messages.EM_SETHANDLE.
EM_SETIMESTATUS (see page 16344)	This is constant Messages.EM_SETIMESTATUS.
EM_SETLIMITTEXT (see page 16344)	This is constant Messages.EM_SETLIMITTEXT.
EM_SETMARGINS (see page 16345)	This is constant Messages.EM_SETMARGINS.
EM_SETMODIFY (see page 16345)	This is constant Messages.EM_SETMODIFY.
EM_SETPASSWORDCHAR (see page 16345)	This is constant Messages.EM_SETPASSWORDCHAR.
EM_SETREADONLY (see page 16346)	This is constant Messages.EM_SETREADONLY.
EM_SETRECT (see page 16346)	This is constant Messages.EM_SETRECT.
EM_SETRECTNP (see page 16346)	This is constant Messages.EM_SETRECTNP.
EM_SETSEL (see page 16347)	This is constant Messages.EM_SETSEL.
EM_SETTABSTOPS (see page 16347)	This is constant Messages.EM_SETTABSTOPS.
EM_SETWORDBREAKPROC (see page 16347)	This is constant Messages.EM_SETWORDBREAKPROC.
EM_UNDO (see page 16348)	This is constant Messages.EM_UNDO.
EN_CHANGE (see page 16348)	This is constant Messages.EN_CHANGE.
EN_ERRSPACE (see page 16348)	This is constant Messages.EN_ERRSPACE.
EN_HSCROLL (see page 16349)	This is constant Messages.EN_HSCROLL.
EN_KILLFOCUS (see page 16349)	This is constant Messages.EN_KILLFOCUS.
EN_MAXTEXT (see page 16349)	This is constant Messages.EN_MAXTEXT.
EN_SETFOCUS (see page 16350)	This is constant Messages.EN_SETFOCUS.
EN_UPDATE (see page 16350)	This is constant Messages.EN_UPDATE.
EN_VSCROLL (see page 16350)	This is constant Messages.EN_VSCROLL.
LB_ADDFILE (see page 16351)	This is constant Messages.LB_ADDFILE.
LB_ADDSTRING (see page 16351)	This is constant Messages.LB_ADDSTRING.
LB_DELETESTRING (see page 16351)	This is constant Messages.LB_DELETESTRING.
LB_DIR (see page 16352)	This is constant Messages.LB_DIR.
LB_FINDSTRING (see page 16352)	This is constant Messages.LB_FINDSTRING.
LB_FINDSTRINGEXACT (see page 16352)	This is constant Messages.LB_FINDSTRINGEXACT.
LB_GETANCHORINDEX (see page 16353)	This is constant Messages.LB_GETANCHORINDEX.
LB_GETCARETINDEX (see page 16353)	This is constant Messages.LB_GETCARETINDEX.
LB_GETCOUNT (see page 16353)	This is constant Messages.LB_GETCOUNT.
LB_GETCURSEL (see page 16354)	This is constant Messages.LB_GETCURSEL.
LB_GETHORIZONTALEXTENT (see page 16354)	This is constant Messages.LB_GETHORIZONTALEXTENT.
LB_GETITEMDATA (see page 16354)	This is constant Messages.LB_GETITEMDATA.
LB_GETITEMHEIGHT (see page 16355)	This is constant Messages.LB_GETITEMHEIGHT.
LB_GETITEMRECT (see page 16355)	This is constant Messages.LB_GETITEMRECT.
LB_GETLOCALE (see page 16355)	This is constant Messages.LB_GETLOCALE.
LB_GETSEL (see page 16356)	This is constant Messages.LB_GETSEL.
LB_GETSELCOUNT (see page 16356)	This is constant Messages.LB_GETSELCOUNT.
LB_GETSELITEMS (see page 16356)	This is constant Messages.LB_GETSELITEMS.

LB_GETTEXT (see page 16357)	This is constant Messages.LB_GETTEXT.
LB_GETTEXTLEN (see page 16357)	This is constant Messages.LB_GETTEXTLEN.
LB_GETTOPINDEX (see page 16357)	This is constant Messages.LB_GETTOPINDEX.
LB_INITSTORAGE (see page 16358)	This is constant Messages.LB_INITSTORAGE.
LB_INSERTSTRING (see page 16358)	This is constant Messages.LB_INSERTSTRING.
LB_ITEMFROMPOINT (see page 16358)	This is constant Messages.LB_ITEMFROMPOINT.
LB_MSGMAX (see page 16359)	This is constant Messages.LB_MSGMAX.
LB_RESETCONTENT (see page 16359)	This is constant Messages.LB_RESETCONTENT.
LB_SELECTSTRING (see page 16359)	This is constant Messages.LB_SELECTSTRING.
LB_SELITEMRANGE (see page 16360)	This is constant Messages.LB_SELITEMRANGE.
LB_SELITEMRANGEEX (see page 16360)	This is constant Messages.LB_SELITEMRANGEEX.
LB_SETANCHORINDEX (see page 16360)	This is constant Messages.LB_SETANCHORINDEX.
LB_SETCARETINDEX (see page 16361)	This is constant Messages.LB_SETCARETINDEX.
LB_SETCOLUMNWIDTH (see page 16361)	This is constant Messages.LB_SETCOLUMNWIDTH.
LB_SETCOUNT (see page 16361)	This is constant Messages.LB_SETCOUNT.
LB_SETCURSEL (see page 16362)	This is constant Messages.LB_SETCURSEL.
LB_SETHORIZONTALEXTENT (see page 16362)	This is constant Messages.LB_SETHORIZONTALEXTENT.
LB_SETITEMDATA (see page 16362)	This is constant Messages.LB_SETITEMDATA.
LB_SETITEMHEIGHT (see page 16363)	This is constant Messages.LB_SETITEMHEIGHT.
LB_SETLOCALE (see page 16363)	This is constant Messages.LB_SETLOCALE.
LB_SETSEL (see page 16363)	This is constant Messages.LB_SETSEL.
LB_SETTABSTOPS (see page 16364)	This is constant Messages.LB_SETTABSTOPS.
LB_SETTOPINDEX (see page 16364)	This is constant Messages.LB_SETTOPINDEX.
LBN_DBLCLK (see page 16364)	This is constant Messages.LBN_DBLCLK.
LBN_ERRSPACE (see page 16365)	This is constant Messages.LBN_ERRSPACE.
LBN_KILLFOCUS (see page 16365)	This is constant Messages.LBN_KILLFOCUS.
LBN_SELCANCEL (see page 16365)	This is constant Messages.LBN_SELCANCEL.
LBN_SELCHANGE (see page 16366)	This is constant Messages.LBN_SELCHANGE.
LBN_SETFOCUS (see page 16366)	This is constant Messages.LBN_SETFOCUS.
PSM_PAGEINFO (see page 16366)	This is constant Messages.PSM_PAGEINFO.
PSM_SHEETINFO (see page 16367)	This is constant Messages.PSM_SHEETINFO.
SBM_ENABLE_ARROWS (see page 16367)	This is constant Messages.SBM_ENABLE_ARROWS.
SBM_GETPOS (see page 16367)	This is constant Messages.SBM_GETPOS.
SBM_GETRANGE (see page 16368)	This is constant Messages.SBM_GETRANGE.
SBM_GETSCROLLINFO (see page 16368)	This is constant Messages.SBM_GETSCROLLINFO.
SBM_SETPOS (see page 16368)	This is constant Messages.SBM_SETPOS.
SBM_SETRANGE (see page 16369)	This is constant Messages.SBM_SETRANGE.
SBM_SETRANGEREDRAW (see page 16369)	This is constant Messages.SBM_SETRANGEREDRAW.
SBM_SETSCROLLINFO (see page 16369)	This is constant Messages.SBM_SETSCROLLINFO.
WM_ACTIVATE (see page 16370)	This is constant Messages.WM_ACTIVATE.
WM_ACTIVATEAPP (see page 16370)	This is constant Messages.WM_ACTIVATEAPP.
WM_APP (see page 16370)	This is constant Messages.WM_APP.
WM_APPCOMMAND (see page 16371)	This is constant Messages.WM_APPCOMMAND.
WM_ASKCBFORMATNAME (see page 16371)	This is constant Messages.WM_ASKCBFORMATNAME.
WM_CANCELJOURNAL (see page 16371)	This is constant Messages.WM_CANCELJOURNAL.
WM_CANCELMODE (see page 16372)	This is constant Messages.WM_CANCELMODE.
WM_CAPTURECHANGED (see page 16372)	This is constant Messages.WM_CAPTURECHANGED.
WM_CHANGECBCHAIN (see page 16372)	This is constant Messages.WM_CHANGECBCHAIN.
WM_CHANGEUISTATE (see page 16373)	This is constant Messages.WM_CHANGEUISTATE.
WM_CHAR (see page 16373)	This is constant Messages.WM_CHAR.
WM_CHARTOITEM (see page 16373)	This is constant Messages.WM_CHARTOITEM.
WM_CHILDACTIVATE (see page 16374)	This is constant Messages.WM_CHILDACTIVATE.
WM_CLEAR (see page 16374)	This is constant Messages.WM_CLEAR.
WM_CLOSE (see page 16374)	This is constant Messages.WM_CLOSE.

WM_COALESCE_FIRST (see page 16375)	This is constant Messages.WM_COALESCE_FIRST.
WM_COALESCE_LAST (see page 16375)	This is constant Messages.WM_COALESCE_LAST.
WM_COMMAND (see page 16375)	This is constant Messages.WM_COMMAND.
WM_COMMNOTIFY (see page 16376)	This is constant Messages.WM_COMMNOTIFY.
WM_COMPACTING (see page 16376)	This is constant Messages.WM_COMPACTING.
WM_COMPAREITEM (see page 16376)	This is constant Messages.WM_COMPAREITEM.
WM_CONTEXTMENU (see page 16377)	This is constant Messages.WM_CONTEXTMENU.
WM_COPY (see page 16377)	This is constant Messages.WM_COPY.
WM_COPYDATA (see page 16377)	This is constant Messages.WM_COPYDATA.
WM_CREATE (see page 16378)	This is constant Messages.WM_CREATE.
WM_CTLCOLOR (see page 16378)	This is constant Messages.WM_CTLCOLOR.
WM_CTLCOLORBTN (see page 16378)	This is constant Messages.WM_CTLCOLORBTN.
WM_CTLCOLORDLG (see page 16379)	This is constant Messages.WM_CTLCOLORDLG.
WM_CTLCOLOREDIT (see page 16379)	This is constant Messages.WM_CTLCOLOREDIT.
WM_CTLCOLORLISTBOX (see page 16379)	This is constant Messages.WM_CTLCOLORLISTBOX.
WM_CTLCOLORMSGBOX (see page 16380)	This is constant Messages.WM_CTLCOLORMSGBOX.
WM_CTLCOLORSCROLLBAR (see page 16380)	This is constant Messages.WM_CTLCOLORSCROLLBAR.
WM_CTLCOLORSTATIC (see page 16380)	This is constant Messages.WM_CTLCOLORSTATIC.
WM_CUT (see page 16381)	This is constant Messages.WM_CUT.
WM_DDE_ACK (see page 16381)	This is constant Messages.WM_DDE_ACK.
WM_DDE_ADVISE (see page 16381)	This is constant Messages.WM_DDE_ADVISE.
WM_DDE_DATA (see page 16382)	This is constant Messages.WM_DDE_DATA.
WM_DDE_EXECUTE (see page 16382)	This is constant Messages.WM_DDE_EXECUTE.
WM_DDE_FIRST (see page 16382)	This is constant Messages.WM_DDE_FIRST.
WM_DDE_INITIATE (see page 16383)	This is constant Messages.WM_DDE_INITIATE.
WM_DDE_LAST (see page 16383)	This is constant Messages.WM_DDE_LAST.
WM_DDE_POKE (see page 16383)	This is constant Messages.WM_DDE_POKE.
WM_DDE_REQUEST (see page 16384)	This is constant Messages.WM_DDE_REQUEST.
WM_DDE_TERMINATE (see page 16384)	This is constant Messages.WM_DDE_TERMINATE.
WM_DDE_UNADVISE (see page 16384)	This is constant Messages.WM_DDE_UNADVISE.
WM_DEADCHAR (see page 16385)	This is constant Messages.WM_DEADCHAR.
WM_DELETEITEM (see page 16385)	This is constant Messages.WM_DELETEITEM.
WM_DESTROY (see page 16385)	This is constant Messages.WM_DESTROY.
WM_DESTROYCLIPBOARD (see page 16386)	This is constant Messages.WM_DESTROYCLIPBOARD.
WM_DEVICECHANGE (see page 16386)	This is constant Messages.WM_DEVICECHANGE.
WM_DEVMODECHANGE (see page 16386)	This is constant Messages.WM_DEVMODECHANGE.
WM_DISPLAYCHANGE (see page 16387)	This is constant Messages.WM_DISPLAYCHANGE.
WM_DRAWCLIPBOARD (see page 16387)	This is constant Messages.WM_DRAWCLIPBOARD.
WM_DRAWITEM (see page 16387)	This is constant Messages.WM_DRAWITEM.
WM_DROPFILES (see page 16388)	This is constant Messages.WM_DROPFILES.
WM_DWMCOLORIZATIONCOLORCHANGED (see page 16388)	This is constant Messages.WM_DWMCOLORIZATIONCOLORCHANGED.
WM_DWMCOMPOSITIONCHANGED (see page 16388)	This is constant Messages.WM_DWMCOMPOSITIONCHANGED.
WM_DWMNCRENDERINGCHANGED (see page 16389)	This is constant Messages.WM_DWMNCRENDERINGCHANGED.
WM_DWMWINDOWMAXIMIZEDCHANGE (see page 16389)	This is constant Messages.WM_DWMWINDOWMAXIMIZEDCHANGE.
WM_ENABLE (see page 16389)	This is constant Messages.WM_ENABLE.
WM_ENDSESSION (see page 16390)	This is constant Messages.WM_ENDSESSION.
WM_ENTERIDLE (see page 16390)	This is constant Messages.WM_ENTERIDLE.
WM_ENTERMENULOOP (see page 16390)	This is constant Messages.WM_ENTERMENULOOP.
WM_ENTERSIZEMOVE (see page 16391)	This is constant Messages.WM_ENTERSIZEMOVE.
WM_ERASEBKGND (see page 16391)	This is constant Messages.WM_ERASEBKGND.
WM_EXITMENULOOP (see page 16391)	This is constant Messages.WM_EXITMENULOOP.
WM_EXITSIZEMOVE (see page 16392)	This is constant Messages.WM_EXITSIZEMOVE.
WM_FONTCHANGE (see page 16392)	This is constant Messages.WM_FONTCHANGE.
WM_GETDLGCODE (see page 16392)	This is constant Messages.WM_GETDLGCODE.

WM_GETFONT (see page 16393)	This is constant Messages.WM_GETFONT.
WM_GETHOTKEY (see page 16393)	This is constant Messages.WM_GETHOTKEY.
WM_GETICON (see page 16393)	This is constant Messages.WM_GETICON.
WM_GETMINMAXINFO (see page 16394)	This is constant Messages.WM_GETMINMAXINFO.
WM_GETOBJECT (see page 16394)	This is constant Messages.WM_GETOBJECT.
WM_GETTEXT (see page 16394)	This is constant Messages.WM_GETTEXT.
WM_GETTEXTLENGTH (see page 16395)	This is constant Messages.WM_GETTEXTLENGTH.
WM_HANDHELDFIRST (see page 16395)	This is constant Messages.WM_HANDHELDFIRST.
WM_HANDHELDLAST (see page 16395)	This is constant Messages.WM_HANDHELDLAST.
WM_HELP (see page 16396)	This is constant Messages.WM_HELP.
WM_HOTKEY (see page 16396)	This is constant Messages.WM_HOTKEY.
WM_HSCROLL (see page 16396)	This is constant Messages.WM_HSCROLL.
WM_HSCROLLCLIPBOARD (see page 16397)	This is constant Messages.WM_HSCROLLCLIPBOARD.
WM_ICONERASEBKGND (see page 16397)	This is constant Messages.WM_ICONERASEBKGND.
WM_IME_CHAR (see page 16397)	This is constant Messages.WM_IME_CHAR.
WM_IME_COMPOSITION (see page 16398)	This is constant Messages.WM_IME_COMPOSITION.
WM_IME_COMPOSITIONFULL (see page 16398)	This is constant Messages.WM_IME_COMPOSITIONFULL.
WM_IME_CONTROL (see page 16398)	This is constant Messages.WM_IME_CONTROL.
WM_IME_ENDCOMPOSITION (see page 16399)	This is constant Messages.WM_IME_ENDCOMPOSITION.
WM_IME_KEYDOWN (see page 16399)	This is constant Messages.WM_IME_KEYDOWN.
WM_IME_KEYLAST (see page 16399)	This is constant Messages.WM_IME_KEYLAST.
WM_IME_KEYUP (see page 16400)	This is constant Messages.WM_IME_KEYUP.
WM_IME_NOTIFY (see page 16400)	This is constant Messages.WM_IME_NOTIFY.
WM_IME_REQUEST (see page 16400)	This is constant Messages.WM_IME_REQUEST.
WM_IME_SELECT (see page 16401)	This is constant Messages.WM_IME_SELECT.
WM_IME_SETCONTEXT (see page 16401)	This is constant Messages.WM_IME_SETCONTEXT.
WM_IME_STARTCOMPOSITION (see page 16401)	This is constant Messages.WM_IME_STARTCOMPOSITION.
WM_INITDIALOG (see page 16402)	This is constant Messages.WM_INITDIALOG.
WM_INITMENU (see page 16402)	This is constant Messages.WM_INITMENU.
WM_INITMENUPOPUP (see page 16402)	This is constant Messages.WM_INITMENUPOPUP.
WM_INPUT (see page 16403)	This is constant Messages.WM_INPUT.
WM_INPUTLANGCHANGE (see page 16403)	This is constant Messages.WM_INPUTLANGCHANGE.
WM_INPUTLANGCHANGEREQUEST (see page 16403)	This is constant Messages.WM_INPUTLANGCHANGEREQUEST.
WM_KEYDOWN (see page 16404)	This is constant Messages.WM_KEYDOWN.
WM_KEYFIRST (see page 16404)	This is constant Messages.WM_KEYFIRST.
WM_KEYLAST (see page 16404)	This is constant Messages.WM_KEYLAST.
WM_KEYUP (see page 16405)	This is constant Messages.WM_KEYUP.
WM_KILLFOCUS (see page 16405)	This is constant Messages.WM_KILLFOCUS.
WM_LBUTTONDBLCLK (see page 16405)	This is constant Messages.WM_LBUTTONDBLCLK.
WM_LBUTTONDOWN (see page 16406)	This is constant Messages.WM_LBUTTONDOWN.
WM_LBUTTONUP (see page 16406)	This is constant Messages.WM_LBUTTONUP.
WM_MBUTTONDBLCLK (see page 16406)	This is constant Messages.WM_MBUTTONDBLCLK.
WM_MBUTTONDOWN (see page 16407)	This is constant Messages.WM_MBUTTONDOWN.
WM_MBUTTONUP (see page 16407)	This is constant Messages.WM_MBUTTONUP.
WM_MDIACTIVATE (see page 16407)	This is constant Messages.WM_MDIACTIVATE.
WM_MDICASCADE (see page 16408)	This is constant Messages.WM_MDICASCADE.
WM_MDICREATE (see page 16408)	This is constant Messages.WM_MDICREATE.
WM_MDIDESTROY (see page 16408)	This is constant Messages.WM_MDIDESTROY.
WM_MDIGETACTIVE (see page 16409)	This is constant Messages.WM_MDIGETACTIVE.
WM_MDIICONARRANGE (see page 16409)	This is constant Messages.WM_MDIICONARRANGE.
WM_MDIMAXIMIZE (see page 16409)	This is constant Messages.WM_MDIMAXIMIZE.
WM_MDINEXT (see page 16410)	This is constant Messages.WM_MDINEXT.
WM_MDIREFRESHMENU (see page 16410)	This is constant Messages.WM_MDIREFRESHMENU.
WM_MDIRESTORE (see page 16410)	This is constant Messages.WM_MDIRESTORE.

WM_MDISETMENU (see page 16411)	This is constant Messages.WM_MDISETMENU.
WM_MDITILE (see page 16411)	This is constant Messages.WM_MDITILE.
WM_MEASUREITEM (see page 16411)	This is constant Messages.WM_MEASUREITEM.
WM_MENUCHAR (see page 16412)	This is constant Messages.WM_MENUCHAR.
WM_MENUCOMMAND (see page 16412)	This is constant Messages.WM_MENUCOMMAND.
WM_MENUDRAG (see page 16412)	This is constant Messages.WM_MENUDRAG.
WM_MENUGETOBJECT (see page 16413)	This is constant Messages.WM_MENUGETOBJECT.
WM_MENURBUTTONUP (see page 16413)	This is constant Messages.WM_MENURBUTTONUP.
WM_MENUSELECT (see page 16413)	This is constant Messages.WM_MENUSELECT.
WM_MOUSEACTIVATE (see page 16414)	This is constant Messages.WM_MOUSEACTIVATE.
WM_MOUSEFIRST (see page 16414)	This is constant Messages.WM_MOUSEFIRST.
WM_MOUSEHOVER (see page 16414)	This is constant Messages.WM_MOUSEHOVER.
WM_MOUSELAST (see page 16415)	This is constant Messages.WM_MOUSELAST.
WM_MOUSELEAVE (see page 16415)	This is constant Messages.WM_MOUSELEAVE.
WM_MOUSEMOVE (see page 16415)	This is constant Messages.WM_MOUSEMOVE.
WM_MOUSEWHEEL (see page 16416)	This is constant Messages.WM_MOUSEWHEEL.
WM_MOVE (see page 16416)	This is constant Messages.WM_MOVE.
WM_MOVING (see page 16416)	This is constant Messages.WM_MOVING.
WM_NCACTIVATE (see page 16417)	This is constant Messages.WM_NCACTIVATE.
WM_NCCALCSIZE (see page 16417)	This is constant Messages.WM_NCCALCSIZE.
WM_NCCREATE (see page 16417)	This is constant Messages.WM_NCCREATE.
WM_NCDESTROY (see page 16418)	This is constant Messages.WM_NCDESTROY.
WM_NCHITTEST (see page 16418)	This is constant Messages.WM_NCHITTEST.
WM_NCLBUTTONDBLCLK (see page 16418)	This is constant Messages.WM_NCLBUTTONDBLCLK.
WM_NCLBUTTONDOWN (see page 16419)	This is constant Messages.WM_NCLBUTTONDOWN.
WM_NCLBUTTONUP (see page 16419)	This is constant Messages.WM_NCLBUTTONUP.
WM_NCMBUTTONDBLCLK (see page 16419)	This is constant Messages.WM_NCMBUTTONDBLCLK.
WM_NCMBUTTONDOWN (see page 16420)	This is constant Messages.WM_NCMBUTTONDOWN.
WM_NCMBUTTONUP (see page 16420)	This is constant Messages.WM_NCMBUTTONUP.
WM_NCMOUSEHOVER (see page 16420)	This is constant Messages.WM_NCMOUSEHOVER.
WM_NCMOUSELEAVE (see page 16421)	This is constant Messages.WM_NCMOUSELEAVE.
WM_NCMOUSEMOVE (see page 16421)	This is constant Messages.WM_NCMOUSEMOVE.
WM_NCPAINT (see page 16421)	This is constant Messages.WM_NCPAINT.
WM_NCRBUTTONDBLCLK (see page 16422)	This is constant Messages.WM_NCRBUTTONDBLCLK.
WM_NCRBUTTONDOWN (see page 16422)	This is constant Messages.WM_NCRBUTTONDOWN.
WM_NCRBUTTONUP (see page 16422)	This is constant Messages.WM_NCRBUTTONUP.
WM_NCXBUTTONDBLCLK (see page 16423)	This is constant Messages.WM_NCXBUTTONDBLCLK.
WM_NCXBUTTONDOWN (see page 16423)	This is constant Messages.WM_NCXBUTTONDOWN.
WM_NCXBUTTONUP (see page 16423)	This is constant Messages.WM_NCXBUTTONUP.
WM_NEXTDLGCTL (see page 16424)	This is constant Messages.WM_NEXTDLGCTL.
WM_NEXTMENU (see page 16424)	This is constant Messages.WM_NEXTMENU.
WM_NOTIFY (see page 16424)	This is constant Messages.WM_NOTIFY.
WM_NOTIFYFORMAT (see page 16425)	This is constant Messages.WM_NOTIFYFORMAT.
WM_NULL (see page 16425)	This is constant Messages.WM_NULL.
WM_PAINT (see page 16425)	This is constant Messages.WM_PAINT.
WM_PAINTCLIPBOARD (see page 16426)	This is constant Messages.WM_PAINTCLIPBOARD.
WM_PAINTICON (see page 16426)	This is constant Messages.WM_PAINTICON.
WM_PALETTECHANGED (see page 16426)	This is constant Messages.WM_PALETTECHANGED.
WM_PALETTEISCHANGING (see page 16427)	This is constant Messages.WM_PALETTEISCHANGING.
WM_PARENTNOTIFY (see page 16427)	This is constant Messages.WM_PARENTNOTIFY.
WM_PASTE (see page 16427)	This is constant Messages.WM_PASTE.
WM_PENWINFIRST (see page 16428)	This is constant Messages.WM_PENWINFIRST.
WM_PENWINLAST (see page 16428)	This is constant Messages.WM_PENWINLAST.
WM_POWER (see page 16428)	This is constant Messages.WM_POWER.

WM_POWERBROADCAST (see page 16429)	This is constant Messages.WM_POWERBROADCAST.
WM_PRINT (see page 16429)	This is constant Messages.WM_PRINT.
WM_PRINTCLIENT (see page 16429)	This is constant Messages.WM_PRINTCLIENT.
WM_QUERYDRAGICON (see page 16430)	This is constant Messages.WM_QUERYDRAGICON.
WM_QUERYENDSESSION (see page 16430)	This is constant Messages.WM_QUERYENDSESSION.
WM_QUERYNEWPALETTE (see page 16430)	This is constant Messages.WM_QUERYNEWPALETTE.
WM_QUERYOPEN (see page 16431)	This is constant Messages.WM_QUERYOPEN.
WM_QUERYUISTATE (see page 16431)	This is constant Messages.WM_QUERYUISTATE.
WM_QUEUESYNC (see page 16431)	This is constant Messages.WM_QUEUESYNC.
WM_QUIT (see page 16432)	This is constant Messages.WM_QUIT.
WM_RBUTTONDBLCLK (see page 16432)	This is constant Messages.WM_RBUTTONDBLCLK.
WM_RBUTTONDOWN (see page 16432)	This is constant Messages.WM_RBUTTONDOWN.
WM_RBUTTONUP (see page 16433)	This is constant Messages.WM_RBUTTONUP.
WM_RENDERALLFORMATS (see page 16433)	This is constant Messages.WM_RENDERALLFORMATS.
WM_RENDERFORMAT (see page 16433)	This is constant Messages.WM_RENDERFORMAT.
WM_SETCURSOR (see page 16434)	This is constant Messages.WM_SETCURSOR.
WM_SETFOCUS (see page 16434)	This is constant Messages.WM_SETFOCUS.
WM_SETFONT (see page 16434)	This is constant Messages.WM_SETFONT.
WM_SETHOTKEY (see page 16435)	This is constant Messages.WM_SETHOTKEY.
WM_SETICON (see page 16435)	This is constant Messages.WM_SETICON.
WM_SETREDRAW (see page 16435)	This is constant Messages.WM_SETREDRAW.
WM_SETTEXT (see page 16436)	This is constant Messages.WM_SETTEXT.
WM_SETTINGCHANGE (see page 16436)	This is constant Messages.WM_SETTINGCHANGE.
WM_SHOWWINDOW (see page 16436)	This is constant Messages.WM_SHOWWINDOW.
WM_SIZE (see page 16437)	This is constant Messages.WM_SIZE.
WM_SIZECLIPBOARD (see page 16437)	This is constant Messages.WM_SIZECLIPBOARD.
WM_SIZING (see page 16437)	This is constant Messages.WM_SIZING.
WM_SPOOLERSTATUS (see page 16438)	This is constant Messages.WM_SPOOLERSTATUS.
WM_STYLECHANGED (see page 16438)	This is constant Messages.WM_STYLECHANGED.
WM_STYLECHANGING (see page 16438)	This is constant Messages.WM_STYLECHANGING.
WM_SYSCHAR (see page 16439)	This is constant Messages.WM_SYSCHAR.
WM_SYSCOLORCHANGE (see page 16439)	This is constant Messages.WM_SYSCOLORCHANGE.
WM_SYSCOMMAND (see page 16439)	This is constant Messages.WM_SYSCOMMAND.
WM_SYSDEADCHAR (see page 16440)	This is constant Messages.WM_SYSDEADCHAR.
WM_SYSKEYDOWN (see page 16440)	This is constant Messages.WM_SYSKEYDOWN.
WM_SYSKEYUP (see page 16440)	This is constant Messages.WM_SYSKEYUP.
WM_SYSTEMERROR (see page 16441)	This is constant Messages.WM_SYSTEMERROR.
WM_TABLET_FIRST (see page 16441)	This is constant Messages.WM_TABLET_FIRST.
WM_TABLET_LAST (see page 16441)	This is constant Messages.WM_TABLET_LAST.
WM_TCARD (see page 16442)	This is constant Messages.WM_TCARD.
WM_THEMECHANGED (see page 16442)	This is constant Messages.WM_THEMECHANGED.
WM_TIMECHANGE (see page 16442)	This is constant Messages.WM_TIMECHANGE.
WM_TIMER (see page 16443)	This is constant Messages.WM_TIMER.
WM_UNDO (see page 16443)	This is constant Messages.WM_UNDO.
WM_UNICHAR (see page 16443)	This is constant Messages.WM_UNICHAR.
WM_UNINITMENUPOPUP (see page 16444)	This is constant Messages.WM_UNINITMENUPOPUP.
WM_UPDATEUISTATE (see page 16444)	This is constant Messages.WM_UPDATEUISTATE.
WM_USER (see page 16444)	This is constant Messages.WM_USER.
WM_USERCHANGED (see page 16445)	This is constant Messages.WM_USERCHANGED.
WM_VKEYTOITEM (see page 16445)	This is constant Messages.WM_VKEYTOITEM.
WM_VSCROLL (see page 16445)	This is constant Messages.WM_VSCROLL.
WM_VSCROLLCLIPBOARD (see page 16446)	This is constant Messages.WM_VSCROLLCLIPBOARD.
WM_WINDOWPOSCHANGED (see page 16446)	This is constant Messages.WM_WINDOWPOSCHANGED.
WM_WINDOWPOSCHANGING (see page 16446)	This is constant Messages.WM_WINDOWPOSCHANGING.

WM_WININICHANGE (see page 16447)	This is constant Messages.WM_WININICHANGE.
WM_WTSSESSION_CHANGE (see page 16447)	This is constant Messages.WM_WTSSESSION_CHANGE.

Structs, Records, Enums

	Name	Description
	TMessage (see page 16225)	TMessage represents a Windows message.
	TMSHMouseWheel (see page 16226)	This is record Messages.TMSHMouseWheel.
	TWMActivate (see page 16226)	This is record Messages.TWMActivate.
	TWMActivateApp (see page 16227)	This is record Messages.TWMActivateApp.
	TWMAskCBFormatName (see page 16227)	This is record Messages.TWMAskCBFormatName.
	TWMChangeCBChain (see page 16228)	This is record Messages.TWMChangeCBChain.
	TWMCharToItem (see page 16228)	This is record Messages.TWMCharToItem.
	TWMChooseFont_GetLogFont (see page 16229)	This is record Messages.TWMChooseFont_GetLogFont.
	TWMCommand (see page 16229)	This is record Messages.TWMCommand.
	TWMCompacting (see page 16230)	This is record Messages.TWMCompacting.
	TWMCompareItem (see page 16230)	This is record Messages.TWMCompareItem.
	TWMContextMenu (see page 16231)	This is record Messages.TWMContextMenu.
	TWMCopyData (see page 16231)	This is record Messages.TWMCopyData.
	TWMCreate (see page 16232)	This is record Messages.TWMCreate.
	TWMCtlColor (see page 16232)	This is record Messages.TWMCtlColor.
	TWMDDE_Ack (see page 16233)	This is record Messages.TWMDDE_Ack.
	TWMDDE_Advise (see page 16233)	This is record Messages.TWMDDE_Advise.
	TWMDDE_Data (see page 16234)	This is record Messages.TWMDDE_Data.
	TWMDDE_Execute (see page 16234)	This is record Messages.TWMDDE_Execute.
	TWMDDE_Initiate (see page 16235)	This is record Messages.TWMDDE_Initiate.
	TWMDDE_Poke (see page 16235)	This is record Messages.TWMDDE_Poke.
	TWMDDE_Request (see page 16236)	This is record Messages.TWMDDE_Request.
	TWMDDE_Terminate (see page 16236)	This is record Messages.TWMDDE_Terminate.
	TWMDDE_Unadvise (see page 16237)	This is record Messages.TWMDDE_Unadvise.
	TWMDeleteItem (see page 16237)	This is record Messages.TWMDeleteItem.
	TWMDevModeChange (see page 16238)	This is record Messages.TWMDevModeChange.
	TWMDisplayChange (see page 16238)	This is record Messages.TWMDisplayChange.
	TWMDrawItem (see page 16239)	This is record Messages.TWMDrawItem.
	TWMDropFiles (see page 16239)	This is record Messages.TWMDropFiles.
	TWMEnable (see page 16240)	This is record Messages.TWMEnable.
	TWMEndSession (see page 16240)	This is record Messages.TWMEndSession.
	TWMEnterIdle (see page 16241)	This is record Messages.TWMEnterIdle.
	TWMEnterMenuLoop (see page 16241)	This is record Messages.TWMEnterMenuLoop.
	TWMEraseBkgnd (see page 16242)	This is record Messages.TWMEraseBkgnd.
	TWMGetIcon (see page 16242)	This is record Messages.TWMGetIcon.
	TWMGetMinMaxInfo (see page 16243)	This is record Messages.TWMGetMinMaxInfo.
	TWMGetText (see page 16243)	This is record Messages.TWMGetText.
	TWMHelp (see page 16244)	This is record Messages.TWMHelp.
	TWMHotKey (see page 16244)	This is record Messages.TWMHotKey.
	TWMHScrollClipboard (see page 16245)	This is record Messages.TWMHScrollClipboard.
	TWMInitDialog (see page 16245)	This is record Messages.TWMInitDialog.
	TWMInitMenu (see page 16246)	This is record Messages.TWMInitMenu.
	TWMInitMenuPopup (see page 16246)	This is record Messages.TWMInitMenuPopup.
	TWMKey (see page 16247)	This is record Messages.TWMKey.
	TWMKillFocus (see page 16247)	This is record Messages.TWMKillFocus.
	TWMMDIActivate (see page 16248)	This is record Messages.TWMMDIActivate.
	TWMMDICascade (see page 16248)	This is record Messages.TWMMDICascade.
	TWMMDICreate (see page 16249)	This is record Messages.TWMMDICreate.
	TWMMDIDestroy (see page 16249)	This is record Messages.TWMMDIDestroy.

	TWMMDIMaximize (see page 16250)	This is record Messages.TWMMDIMaximize.
	TWMMDINext (see page 16250)	This is record Messages.TWMMDINext.
	TWMMDIRestore (see page 16251)	This is record Messages.TWMMDIRestore.
	TWMMDISetMenu (see page 16251)	This is record Messages.TWMMDISetMenu.
	TWMMDITile (see page 16252)	This is record Messages.TWMMDITile.
	TWMMeasureItem (see page 16252)	This is record Messages.TWMMeasureItem.
	TWMMenuChar (see page 16253)	This is record Messages.TWMMenuChar.
	TWMMenuSelect (see page 16253)	This is record Messages.TWMMenuSelect.
	TWMMouse (see page 16254)	This is record Messages.TWMMouse.
	TWMMouseActivate (see page 16254)	This is record Messages.TWMMouseActivate.
	TWMMouseWheel (see page 16255)	This is record Messages.TWMMouseWheel.
	TWMMove (see page 16255)	This is record Messages.TWMMove.
	TWMMoving (see page 16256)	This is record Messages.TWMMoving.
	TWMNCActivate (see page 16256)	This is record Messages.TWMNCActivate.
	TWMNCCalcSize (see page 16257)	This is record Messages.TWMNCCalcSize.
	TWMNCCreate (see page 16257)	This is record Messages.TWMNCCreate.
	TWMNCHitMessage (see page 16258)	This is record Messages.TWMNCHitMessage.
	TWMNCHitTest (see page 16258)	This is record Messages.TWMNCHitTest.
	TWMNCPaint (see page 16259)	This is record Messages.TWMNCPaint.
	TWMNextDlgCtl (see page 16259)	This is record Messages.TWMNextDlgCtl.
	TWMNoParams (see page 16260)	This is record Messages.TWMNoParams.
	TWMNotify (see page 16260)	This is record Messages.TWMNotify.
	TWMNotifyFormat (see page 16261)	This is record Messages.TWMNotifyFormat.
	TWMPaint (see page 16261)	This is record Messages.TWMPaint.
	TWMPaintClipboard (see page 16262)	This is record Messages.TWMPaintClipboard.
	TWMPaletteChanged (see page 16262)	This is record Messages.TWMPaletteChanged.
	TWMPaletteIsChanging (see page 16263)	This is record Messages.TWMPaletteIsChanging.
	TWMParentNotify (see page 16263)	This is record Messages.TWMParentNotify.
	TWMPower (see page 16264)	This is record Messages.TWMPower.
	TWMPrint (see page 16264)	This is record Messages.TWMPrint.
	TWMQueryEndSession (see page 16265)	This is record Messages.TWMQueryEndSession.
	TWMQuit (see page 16265)	This is record Messages.TWMQuit.
	TWMRenderFormat (see page 16266)	This is record Messages.TWMRenderFormat.
	TWMScroll (see page 16266)	This is record Messages.TWMScroll.
	TWMSetCursor (see page 16267)	This is record Messages.TWMSetCursor.
	TWMSetFocus (see page 16267)	This is record Messages.TWMSetFocus.
	TWMSetFont (see page 16268)	This is record Messages.TWMSetFont.
	TWMSetHotKey (see page 16268)	This is record Messages.TWMSetHotKey.
	TWMSetIcon (see page 16269)	This is record Messages.TWMSetIcon.
	TWMSetRedraw (see page 16269)	This is record Messages.TWMSetRedraw.
	TWMSetText (see page 16270)	This is record Messages.TWMSetText.
	TWMSettingChange (see page 16270)	This is record Messages.TWMSettingChange.
	TWMShowWindow (see page 16271)	This is record Messages.TWMShowWindow.
	TWMSize (see page 16271)	This is record Messages.TWMSize.
	TWMSizeClipboard (see page 16272)	This is record Messages.TWMSizeClipboard.
	TWMSpoolerStatus (see page 16272)	This is record Messages.TWMSpoolerStatus.
	TWMStyleChange (see page 16273)	This is record Messages.TWMStyleChange.
	TWMSysCommand (see page 16273)	This is record Messages.TWMSysCommand.
	TWMSysDeadChar (see page 16274)	This is record Messages.TWMSysDeadChar.
	TWMSystemError (see page 16274)	This is record Messages.TWMSystemError.
	TWMTimer (see page 16275)	This is record Messages.TWMTimer.
	TWMUIState (see page 16275)	This is record Messages.TWMUIState.
	TWMVScrollClipboard (see page 16276)	This is record Messages.TWMVScrollClipboard.
	TWMWindowPosMsg (see page 16276)	This is record Messages.TWMWindowPosMsg.

	TWMWinIniChange (see page 16277)	This is record Messages.TWMWinIniChange.

Types

Name	Description
PMessage (see page 16279)	This is type Messages.PMessage.
TWMCancelMode (see page 16280)	This is type Messages.TWMCancelMode.
TWMChangeUIState (see page 16280)	This is type Messages.TWMChangeUIState.
TWMChar (see page 16280)	This is type Messages.TWMChar.
TWMChildActivate (see page 16281)	This is type Messages.TWMChildActivate.
TWMClear (see page 16281)	This is type Messages.TWMClear.
TWMClose (see page 16281)	This is type Messages.TWMClose.
TWMCopy (see page 16282)	This is type Messages.TWMCopy.
TWMCtlColorBtn (see page 16282)	This is type Messages.TWMCtlColorBtn.
TWMCtlColorDlg (see page 16282)	This is type Messages.TWMCtlColorDlg.
TWMCtlColorEdit (see page 16283)	This is type Messages.TWMCtlColorEdit.
TWMCtlColorListbox (see page 16283)	This is type Messages.TWMCtlColorListbox.
TWMCtlColorMsgbox (see page 16283)	This is type Messages.TWMCtlColorMsgbox.
TWMCtlColorScrollbar (see page 16284)	This is type Messages.TWMCtlColorScrollbar.
TWMCtlColorStatic (see page 16284)	This is type Messages.TWMCtlColorStatic.
TWMCut (see page 16284)	This is type Messages.TWMCut.
TWMDeadChar (see page 16285)	This is type Messages.TWMDeadChar.
TWMDestroy (see page 16285)	This is type Messages.TWMDestroy.
TWMDestroyClipboard (see page 16285)	This is type Messages.TWMDestroyClipboard.
TWMDrawClipboard (see page 16286)	This is type Messages.TWMDrawClipboard.
TWMExitMenuLoop (see page 16286)	This is type Messages.TWMExitMenuLoop.
TWMFontChange (see page 16286)	This is type Messages.TWMFontChange.
TWMGetDlgCode (see page 16287)	This is type Messages.TWMGetDlgCode.
TWMGetFont (see page 16287)	This is type Messages.TWMGetFont.
TWMGetHotKey (see page 16287)	This is type Messages.TWMGetHotKey.
TWMGetTextLength (see page 16288)	This is type Messages.TWMGetTextLength.
TWMHScroll (see page 16288)	This is type Messages.TWMHScroll.
TWMIconEraseBkgnd (see page 16288)	This is type Messages.TWMIconEraseBkgnd.
TWMKeyDown (see page 16289)	This is type Messages.TWMKeyDown.
TWMKeyUp (see page 16289)	This is type Messages.TWMKeyUp.
TWMLButtonDblClk (see page 16289)	This is type Messages.TWMLButtonDblClk.
TWMLButtonDown (see page 16290)	TWMLButtonDown represents the Windows message that occurs when the user clicks the left mouse button.
TWMLButtonUp (see page 16290)	This is type Messages.TWMLButtonUp.
TWMMButtonDblClk (see page 16290)	This is type Messages.TWMMButtonDblClk.
TWMMButtonDown (see page 16291)	This is type Messages.TWMMButtonDown.
TWMMButtonUp (see page 16291)	This is type Messages.TWMMButtonUp.
TWMMDIGetActive (see page 16291)	This is type Messages.TWMMDIGetActive.
TWMMDIIconArrange (see page 16292)	This is type Messages.TWMMDIIconArrange.
TWMMDIRefreshMenu (see page 16292)	This is type Messages.TWMMDIRefreshMenu.
TWMMouseMove (see page 16292)	This is type Messages.TWMMouseMove.
TWMNCDestroy (see page 16293)	This is type Messages.TWMNCDestroy.
TWMNCLButtonDblClk (see page 16293)	This is type Messages.TWMNCLButtonDblClk.
TWMNCLButtonDown (see page 16293)	This is type Messages.TWMNCLButtonDown.
TWMNCLButtonUp (see page 16294)	This is type Messages.TWMNCLButtonUp.
TWMNCMButtonDblClk (see page 16294)	This is type Messages.TWMNCMButtonDblClk.
TWMNCMButtonDown (see page 16294)	This is type Messages.TWMNCMButtonDown.
TWMNCMButtonUp (see page 16295)	This is type Messages.TWMNCMButtonUp.
TWMNCMouseMove (see page 16295)	This is type Messages.TWMNCMouseMove.
TWMNCRButtonDblClk (see page 16295)	This is type Messages.TWMNCRButtonDblClk.
TWMNCRButtonDown (see page 16296)	This is type Messages.TWMNCRButtonDown.

TWMNCRButtonUp (see page 16296)	This is type Messages.TWMNCRButtonUp.
TWMPaintIcon (see page 16296)	This is type Messages.TWMPaintIcon.
TWMPaste (see page 16297)	This is type Messages.TWMPaste.
TWMPrintClient (see page 16297)	This is type Messages.TWMPrintClient.
TWMQueryDragIcon (see page 16297)	This is type Messages.TWMQueryDragIcon.
TWMQueryNewPalette (see page 16298)	This is type Messages.TWMQueryNewPalette.
TWMQueryOpen (see page 16298)	This is type Messages.TWMQueryOpen.
TWMQueryUIState (see page 16298)	This is type Messages.TWMQueryUIState.
TWMQueueSync (see page 16299)	This is type Messages.TWMQueueSync.
TWMRButtonDblClk (see page 16299)	This is type Messages.TWMRButtonDblClk.
TWMRButtonDown (see page 16299)	This is type Messages.TWMRButtonDown.
TWMRButtonUp (see page 16300)	This is type Messages.TWMRButtonUp.
TWMRenderAllFormats (see page 16300)	This is type Messages.TWMRenderAllFormats.
TWMStyleChanged (see page 16300)	This is type Messages.TWMStyleChanged.
TWMStyleChanging (see page 16301)	This is type Messages.TWMStyleChanging.
TWMSysChar (see page 16301)	This is type Messages.TWMSysChar.
TWMSysColorChange (see page 16301)	This is type Messages.TWMSysColorChange.
TWMSysKeyDown (see page 16302)	This is type Messages.TWMSysKeyDown.
TWMSysKeyUp (see page 16302)	This is type Messages.TWMSysKeyUp.
TWMTimeChange (see page 16302)	This is type Messages.TWMTimeChange.
TWMUndo (see page 16303)	This is type Messages.TWMUndo.
TWMUpdateUIState (see page 16303)	This is type Messages.TWMUpdateUIState.
TWMVKeyToItem (see page 16303)	This is type Messages.TWMVKeyToItem.
TWMVScroll (see page 16304)	This is type Messages.TWMVScroll.
TWMWindowPosChanged (see page 16304)	This is type Messages.TWMWindowPosChanged.
TWMWindowPosChanging (see page 16304)	This is type Messages.TWMWindowPosChanging.

96.1 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TMessage (see page 16225)	TMessage represents a Windows message.
	TMSHMouseWheel (see page 16226)	This is record Messages.TMSHMouseWheel.
	TWMActivate (see page 16226)	This is record Messages.TWMActivate.
	TWMActivateApp (see page 16227)	This is record Messages.TWMActivateApp.
	TWMAskCBFormatName (see page 16227)	This is record Messages.TWMAskCBFormatName.
	TWMChangeCBChain (see page 16228)	This is record Messages.TWMChangeCBChain.
	TWMCharToItem (see page 16228)	This is record Messages.TWMCharToItem.
	TWMChooseFont_GetLogFont (see page 16229)	This is record Messages.TWMChooseFont_GetLogFont.
	TWMCommand (see page 16229)	This is record Messages.TWMCommand.
	TWMCompacting (see page 16230)	This is record Messages.TWMCompacting.
	TWMCompareItem (see page 16230)	This is record Messages.TWMCompareItem.
	TWMContextMenu (see page 16231)	This is record Messages.TWMContextMenu.
	TWMCopyData (see page 16231)	This is record Messages.TWMCopyData.
	TWMCreate (see page 16232)	This is record Messages.TWMCreate.
	TWMCtlColor (see page 16232)	This is record Messages.TWMCtlColor.
	TWMDDE_Ack (see page 16233)	This is record Messages.TWMDDE_Ack.
	TWMDDE_Advise (see page 16233)	This is record Messages.TWMDDE_Advise.
	TWMDDE_Data (see page 16234)	This is record Messages.TWMDDE_Data.
	TWMDDE_Execute (see page 16234)	This is record Messages.TWMDDE_Execute.
	TWMDDE_Initiate (see page 16235)	This is record Messages.TWMDDE_Initiate.
	TWMDDE_Poke (see page 16235)	This is record Messages.TWMDDE_Poke.
	TWMDDE_Request (see page 16236)	This is record Messages.TWMDDE_Request.
	TWMDDE_Terminate (see page 16236)	This is record Messages.TWMDDE_Terminate.
	TWMDDE_Unadvise (see page 16237)	This is record Messages.TWMDDE_Unadvise.
	TWMDeleteItem (see page 16237)	This is record Messages.TWMDeleteItem.
	TWMDevModeChange (see page 16238)	This is record Messages.TWMDevModeChange.
	TWMDisplayChange (see page 16238)	This is record Messages.TWMDisplayChange.
	TWMDrawItem (see page 16239)	This is record Messages.TWMDrawItem.
	TWMDropFiles (see page 16239)	This is record Messages.TWMDropFiles.
	TWMEnable (see page 16240)	This is record Messages.TWMEnable.
	TWMEndSession (see page 16240)	This is record Messages.TWMEndSession.
	TWMEnterIdle (see page 16241)	This is record Messages.TWMEnterIdle.
	TWMEnterMenuLoop (see page 16241)	This is record Messages.TWMEnterMenuLoop.
	TWMEraseBkgnd (see page 16242)	This is record Messages.TWMEraseBkgnd.
	TWMGetIcon (see page 16242)	This is record Messages.TWMGetIcon.
	TWMGetMinMaxInfo (see page 16243)	This is record Messages.TWMGetMinMaxInfo.
	TWMGetText (see page 16243)	This is record Messages.TWMGetText.
	TWMHelp (see page 16244)	This is record Messages.TWMHelp.
	TWMHotKey (see page 16244)	This is record Messages.TWMHotKey.
	TWMHScrollClipboard (see page 16245)	This is record Messages.TWMHScrollClipboard.
	TWMInitDialog (see page 16245)	This is record Messages.TWMInitDialog.
	TWMInitMenu (see page 16246)	This is record Messages.TWMInitMenu.
	TWMInitMenuPopup (see page 16246)	This is record Messages.TWMInitMenuPopup.
	TWMKey (see page 16247)	This is record Messages.TWMKey.

	TWMKillFocus (see page 16247)	This is record Messages.TWMKillFocus.
	TWMMDIActivate (see page 16248)	This is record Messages.TWMMDIActivate.
	TWMMDICascade (see page 16248)	This is record Messages.TWMMDICascade.
	TWMMDICreate (see page 16249)	This is record Messages.TWMMDICreate.
	TWMMDIDestroy (see page 16249)	This is record Messages.TWMMDIDestroy.
	TWMMDIMaximize (see page 16250)	This is record Messages.TWMMDIMaximize.
	TWMMDINext (see page 16250)	This is record Messages.TWMMDINext.
	TWMMDIRestore (see page 16251)	This is record Messages.TWMMDIRestore.
	TWMMDISetMenu (see page 16251)	This is record Messages.TWMMDISetMenu.
	TWMMDITile (see page 16252)	This is record Messages.TWMMDITile.
	TWMMeasureItem (see page 16252)	This is record Messages.TWMMeasureItem.
	TWMMenuChar (see page 16253)	This is record Messages.TWMMenuChar.
	TWMMenuSelect (see page 16253)	This is record Messages.TWMMenuSelect.
	TWMMouse (see page 16254)	This is record Messages.TWMMouse.
	TWMMouseActivate (see page 16254)	This is record Messages.TWMMouseActivate.
	TWMMouseWheel (see page 16255)	This is record Messages.TWMMouseWheel.
	TWMMove (see page 16255)	This is record Messages.TWMMove.
	TWMMoving (see page 16256)	This is record Messages.TWMMoving.
	TWMNCActivate (see page 16256)	This is record Messages.TWMNCActivate.
	TWMNCCalcSize (see page 16257)	This is record Messages.TWMNCCalcSize.
	TWMNCCreate (see page 16257)	This is record Messages.TWMNCCreate.
	TWMNCHitMessage (see page 16258)	This is record Messages.TWMNCHitMessage.
	TWMNCHitTest (see page 16258)	This is record Messages.TWMNCHitTest.
	TWMNCPaint (see page 16259)	This is record Messages.TWMNCPaint.
	TWMNextDlgCtl (see page 16259)	This is record Messages.TWMNextDlgCtl.
	TWMNoParams (see page 16260)	This is record Messages.TWMNoParams.
	TWMNotify (see page 16260)	This is record Messages.TWMNotify.
	TWMNotifyFormat (see page 16261)	This is record Messages.TWMNotifyFormat.
	TWMPaint (see page 16261)	This is record Messages.TWMPaint.
	TWMPaintClipboard (see page 16262)	This is record Messages.TWMPaintClipboard.
	TWMPaletteChanged (see page 16262)	This is record Messages.TWMPaletteChanged.
	TWMPaletteIsChanging (see page 16263)	This is record Messages.TWMPaletteIsChanging.
	TWMParentNotify (see page 16263)	This is record Messages.TWMParentNotify.
	TWMPower (see page 16264)	This is record Messages.TWMPower.
	TWMPrint (see page 16264)	This is record Messages.TWMPrint.
	TWMQueryEndSession (see page 16265)	This is record Messages.TWMQueryEndSession.
	TWMQuit (see page 16265)	This is record Messages.TWMQuit.
	TWMRenderFormat (see page 16266)	This is record Messages.TWMRenderFormat.
	TWMScroll (see page 16266)	This is record Messages.TWMScroll.
	TWMSetCursor (see page 16267)	This is record Messages.TWMSetCursor.
	TWMSetFocus (see page 16267)	This is record Messages.TWMSetFocus.
	TWMSetFont (see page 16268)	This is record Messages.TWMSetFont.
	TWMSetHotKey (see page 16268)	This is record Messages.TWMSetHotKey.
	TWMSetIcon (see page 16269)	This is record Messages.TWMSetIcon.
	TWMSetRedraw (see page 16269)	This is record Messages.TWMSetRedraw.
	TWMSetText (see page 16270)	This is record Messages.TWMSetText.
	TWMSettingChange (see page 16270)	This is record Messages.TWMSettingChange.
	TWMShowWindow (see page 16271)	This is record Messages.TWMShowWindow.
	TWMSize (see page 16271)	This is record Messages.TWMSize.
	TWMSizeClipboard (see page 16272)	This is record Messages.TWMSizeClipboard.
	TWMSpoolerStatus (see page 16272)	This is record Messages.TWMSpoolerStatus.
	TWMStyleChange (see page 16273)	This is record Messages.TWMStyleChange.
	TWMSysCommand (see page 16273)	This is record Messages.TWMSysCommand.
	TWMSysDeadChar (see page 16274)	This is record Messages.TWMSysDeadChar.

	TWMSystemError (see page 16274)	This is record Messages.TWMSystemError.
	TWMTimer (see page 16275)	This is record Messages.TWMTimer.
	TWMUIState (see page 16275)	This is record Messages.TWMUIState.
	TWMVScrollClipboard (see page 16276)	This is record Messages.TWMVScrollClipboard.
	TWMWindowPosMsg (see page 16276)	This is record Messages.TWMWindowPosMsg.
	TWMWinIniChange (see page 16277)	This is record Messages.TWMWinIniChange.

96.1.1 Messages.TMessage Record

TMessage represents a Windows message.

Pascal

```
TMessage = packed record
  Msg: Cardinal;
  case Integer of
    0: (WParam: Longint;
        LParam: Longint;
        Result: Longint;);
    1: (WParamLo: Word;
        WParamHi: Word;
        LParamLo: Word;
        LParamHi: Word;
        ResultLo: Word;
        ResultHi: Word;);
end;
```

C++

```
struct TMessage {
  unsigned Msg;
  Longint WParam;
  Longint LParam;
  Longint Result;
  Word WParamLo;
  Word WParamHi;
  Word LParamLo;
  Word LParamHi;
  Word ResultLo;
  Word ResultHi;
};
```

File

Messages

Description

The TMessage type represents a Windows message in WndProc and other methods.

The Msg field represents the Windows Message code.

The WParam field represents the WParam of the message. To access the low and high words of this field, use the WParamLo and WParamHi fields instead.

The LParam field represents the LParam of the message. To access the low and high words of this field, use the LParamLo and LParamHi fields instead.

The Result field holds the return value. To access the low and high words of this field, use the ResultLo and ResultHi fields instead.

See Also

TWMKey (see page 16247)

TWMMouse (see page 16254)

TWMPaint (see page 16261)

TWMMenuChar (see page 16253)

TWMNotify (see page 16260)

TWMLButtonDown (see page 16290)

TMsg

96.1.2 Messages.TMSHMouseWheel Record

Pascal

```
TMSHMouseWheel = packed record
  Msg: Cardinal;
  WheelDelta: Integer;
  case Integer of
    0: (XPos: Smallint;
        YPos: Smallint;);
    1: (Pos: TSmallPoint;
        Result: Longint;);
end;
```

C++

```
struct TMSHMouseWheel {
  unsigned Msg;
  int WheelDelta;
  Smallint XPos;
  Smallint YPos;
  TSmallPoint Pos;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TMSHMouseWheel.

96.1.3 Messages.TWMActivate Record

Pascal

```
TWMActivate = packed record
  Msg: Cardinal;
  Active: Word;
  Minimized: WordBool;
  ActiveWindow: HWND;
  Result: Longint;
end;
```

C++

```
struct TWMActivate {
  unsigned Msg;
  Word Active;
  WordBool Minimized;
  HWND ActiveWindow;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMActivate.

96.1.4 Messages.TWMActivateApp Record

Pascal

```
TWMActivateApp = packed record
  Msg: Cardinal;
  Active: BOOL;
  ThreadId: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMActivateApp {
  unsigned Msg;
  BOOL Active;
  Longint ThreadId;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMActivateApp.

96.1.5 Messages.TWMAskCBFormatName Record

Pascal

```
TWMAskCBFormatName = packed record
  Msg: Cardinal;
  NameLen: Word;
  Unused: Word;
  FormatName: PChar;
  Result: Longint;
end;
```

C++

```
struct TWMAskCBFormatName {
```

```
    unsigned Msg;
    Word NameLen;
    Word Unused;
    const char * FormatName;
    Longint Result;
  };
```

File

Messages

Description

This is record Messages.TWMAskCBFormatName.

96.1.6 Messages.TWMChangeCBChain Record

Pascal

```
  TWMChangeCBChain = packed record
    Msg: Cardinal;
    Remove: HWND;
    Next: HWND;
    Result: Longint;
  end;
```

C++

```
  struct TWMChangeCBChain {
    unsigned Msg;
    HWND Remove;
    HWND Next;
    Longint Result;
  };
```

File

Messages

Description

This is record Messages.TWMChangeCBChain.

96.1.7 Messages.TWMCharToItem Record

Pascal

```
  TWMCharToItem = packed record
    Msg: Cardinal;
    Key: Word;
    CaretPos: Word;
    ListBox: HWND;
    Result: Longint;
  end;
```

C++

```
  struct TWMCharToItem {
    unsigned Msg;
    Word Key;
    Word CaretPos;
    HWND ListBox;
```

```
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMCharToItem.

96.1.8 Messages.TWMChooseFont_GetLogFont Record

Pascal

```
TWMChooseFont_GetLogFont = packed record
  Msg: Cardinal;
  Unused: Longint;
  LogFont: PLogFont;
  Result: Longint;
end;
```

C++

```
struct TWMChooseFont_GetLogFont {
  unsigned Msg;
  Longint Unused;
  PLogFont LogFont;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMChooseFont_GetLogFont.

96.1.9 Messages.TWMCommand Record

Pascal

```
TWMCommand = packed record
  Msg: Cardinal;
  ItemID: Word;
  NotifyCode: Word;
  Ctl: HWND;
  Result: Longint;
end;
```

C++

```
struct TWMCommand {
  unsigned Msg;
  Word ItemID;
  Word NotifyCode;
  HWND Ctl;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMCommand.

96.1.10 Messages.TWMCompacting Record

Pascal

```
TWMCompacting = packed record
  Msg: Cardinal;
  CompactRatio: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMCompacting {
  unsigned Msg;
  Longint CompactRatio;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMCompacting.

96.1.11 Messages.TWMCompareItem Record

Pascal

```
TWMCompareItem = packed record
  Msg: Cardinal;
  Ctl: HWnd;
  CompareItemStruct: PCompareItemStruct;
  Result: Longint;
end;
```

C++

```
struct TWMCompareItem {
  unsigned Msg;
  HWnd Ctl;
  PCompareItemStruct CompareItemStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMCompareItem.

96.1.12 Messages.TWMContextMenu Record

Pascal

```
TWMContextMenu = packed record
  Msg: Cardinal;
  hWnd: HWND;
  case Integer of
    0: (XPos: Smallint;
        YPos: Smallint;);
    1: (Pos: TSmallPoint;
        Result: Longint;);
end;
```

C++

```
struct TWMContextMenu {
  unsigned Msg;
  HWND hWnd;
  Smallint XPos;
  Smallint YPos;
  TSmallPoint Pos;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMContextMenu.

96.1.13 Messages.TWMCopyData Record

Pascal

```
TWMCopyData = packed record
  Msg: Cardinal;
  From: HWND;
  CopyDataStruct: PCopyDataStruct;
  Result: Longint;
end;
```

C++

```
struct TWMCopyData {
  unsigned Msg;
  HWND From;
  PCopyDataStruct CopyDataStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMCopyData.

96.1.14 Messages.TWMCreate Record

Pascal

```
TWMCreate = packed record
  Msg: Cardinal;
  Unused: Integer;
  CreateStruct: PCreateStruct;
  Result: Longint;
end;
```

C++

```
struct TWMCreate {
  unsigned Msg;
  int Unused;
  PCreateStruct CreateStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMCreate.

96.1.15 Messages.TWMCtlColor Record

Pascal

```
TWMCtlColor = packed record
  Msg: Cardinal;
  ChildDC: HDC;
  ChildWnd: HWND;
  Result: Longint;
end;
```

C++

```
struct TWMCtlColor {
  unsigned Msg;
  HDC ChildDC;
  HWND ChildWnd;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMCtlColor.

96.1.16 Messages.TWMDDE_Ack Record

Pascal

```
TWMDDE_Ack = packed record
  Msg: Cardinal;
  PostingApp: HWND;
  case Word of
    WM_DDE_EXECUTE: (PackedVal: Longint;);
    WM_DDE_INITIATE: (App: Word;
                      Topic: Word;
                      Result: Longint;);
end;
```

C++

```
struct TWMDDE_Ack {
  unsigned Msg;
  HWND PostingApp;
  Word App;
  Word Topic;
  Longint Result;
  Longint PackedVal;
};
```

File

Messages

Description

This is record Messages.TWMDDE_Ack.

96.1.17 Messages.TWMDDE_Advise Record

Pascal

```
TWMDDE_Advise = packed record
  Msg: Cardinal;
  PostingApp: HWND;
  PackedVal: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMDDE_Advise {
  unsigned Msg;
  HWND PostingApp;
  Longint PackedVal;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDDE_Advise.

96.1.18 Messages.TWMDDE_Data Record

Pascal

```
TWMDDE_Data = packed record
  Msg: Cardinal;
  PostingApp: HWND;
  PackedVal: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMDDE_Data {
  unsigned Msg;
  HWND PostingApp;
  Longint PackedVal;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDDE_Data.

96.1.19 Messages.TWMDDE_Execute Record

Pascal

```
TWMDDE_Execute = packed record
  Msg: Cardinal;
  PostingApp: HWND;
  Commands: THandle;
  Result: Longint;
end;
```

C++

```
struct TWMDDE_Execute {
  unsigned Msg;
  HWND PostingApp;
  THandle Commands;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDDE_Execute.

96.1.20 Messages.TWMDDE_Initiate Record

Pascal

```
TWMDDE_Initiate = packed record
  Msg: Cardinal;
  PostingApp: HWND;
  App: Word;
  Topic: Word;
  Result: Longint;
end;
```

C++

```
struct TWMDDE_Initiate {
  unsigned Msg;
  HWND PostingApp;
  Word App;
  Word Topic;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDDE_Initiate.

96.1.21 Messages.TWMDDE_Poke Record

Pascal

```
TWMDDE_Poke = packed record
  Msg: Cardinal;
  PostingApp: HWND;
  PackedVal: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMDDE_Poke {
  unsigned Msg;
  HWND PostingApp;
  Longint PackedVal;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDDE_Poke.

96.1.22 Messages.TWMDDE_Request Record

Pascal

```
TWMDDE_Request = packed record
  Msg: Cardinal;
  PostingApp: HWND;
  Format: Word;
  Item: Word;
  Result: Longint;
end;
```

C++

```
struct TWMDDE_Request {
  unsigned Msg;
  HWND PostingApp;
  Word Format;
  Word Item;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDDE_Request.

96.1.23 Messages.TWMDDE_Terminate Record

Pascal

```
TWMDDE_Terminate = packed record
  Msg: Cardinal;
  PostingApp: HWND;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMDDE_Terminate {
  unsigned Msg;
  HWND PostingApp;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDDE_Terminate.

96.1.24 Messages.TWMDDE_Unadvise Record

Pascal

```
TWMDDE_Unadvise = packed record
  Msg: Cardinal;
  PostingApp: HWND;
  Format: Word;
  Item: Word;
  Result: Longint;
end;
```

C++

```
struct TWMDDE_Unadvise {
  unsigned Msg;
  HWND PostingApp;
  Word Format;
  Word Item;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDDE_Unadvise.

96.1.25 Messages.TWMDeleteItem Record

Pascal

```
TWMDeleteItem = packed record
  Msg: Cardinal;
  Ctl: HWND;
  DeleteItemStruct: PDeleteItemStruct;
  Result: Longint;
end;
```

C++

```
struct TWMDeleteItem {
  unsigned Msg;
  HWND Ctl;
  PDeleteItemStruct DeleteItemStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDeleteItem.

96.1.26 Messages.TWMDevModeChange Record

Pascal

```
TWMDevModeChange = packed record
  Msg: Cardinal;
  Unused: Integer;
  Device: PChar;
  Result: Longint;
end;
```

C++

```
struct TWMDevModeChange {
  unsigned Msg;
  int Unused;
  const char * Device;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDevModeChange.

96.1.27 Messages.TWMDisplayChange Record

Pascal

```
TWMDisplayChange = packed record
  Msg: Cardinal;
  BitsPerPixel: Integer;
  Width: Word;
  Height: Word;
  Result: Longint;
end;
```

C++

```
struct TWMDisplayChange {
  unsigned Msg;
  int BitsPerPixel;
  Word Width;
  Word Height;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDisplayChange.

96.1.28 Messages.TWMDrawItem Record

Pascal

```
TWMDrawItem = packed record
  Msg: Cardinal;
  Ctl: HWND;
  DrawItemStruct: PDrawItemStruct;
  Result: Longint;
end;
```

C++

```
struct TWMDrawItem {
  unsigned Msg;
  HWND Ctl;
  PDrawItemStruct DrawItemStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDrawItem.

96.1.29 Messages.TWMDropFiles Record

Pascal

```
TWMDropFiles = packed record
  Msg: Cardinal;
  Drop: THANDLE;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMDropFiles {
  unsigned Msg;
  THANDLE Drop;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMDropFiles.

96.1.30 Messages.TWMEnable Record

Pascal

```
TWMEnable = packed record
  Msg: Cardinal;
  Enabled: LongBool;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMEnable {
  unsigned Msg;
  LongBool Enabled;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMEnable.

96.1.31 Messages.TWMEndSession Record

Pascal

```
TWMEndSession = packed record
  Msg: Cardinal;
  EndSession: LongBool;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMEndSession {
  unsigned Msg;
  LongBool EndSession;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMEndSession.

96.1.32 Messages.TWMEnterIdle Record

Pascal

```
TWMEnterIdle = packed record
  Msg: Cardinal;
  Source: Longint;
  IdleWnd: HWND;
  Result: Longint;
end;
```

C++

```
struct TWMEnterIdle {
  unsigned Msg;
  Longint Source;
  HWND IdleWnd;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMEnterIdle.

96.1.33 Messages.TWMEnterMenuLoop Record

Pascal

```
TWMEnterMenuLoop = packed record
  Msg: Cardinal;
  IsTrackPopupMenu: LongBool;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMEnterMenuLoop {
  unsigned Msg;
  LongBool IsTrackPopupMenu;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMEnterMenuLoop.

96.1.34 Messages.TWMEraseBkgnd Record

Pascal

```
TWMEraseBkgnd = packed record
  Msg: Cardinal;
  DC: HDC;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMEraseBkgnd {
  unsigned Msg;
  HDC DC;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMEraseBkgnd.

96.1.35 Messages.TWMGetIcon Record

Pascal

```
TWMGetIcon = packed record
  Msg: Cardinal;
  BigIcon: Longbool;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMGetIcon {
  unsigned Msg;
  Longbool BigIcon;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMGetIcon.

96.1.36 Messages.TWMGetMinMaxInfo Record

Pascal

```
TWMGetMinMaxInfo = packed record
  Msg: Cardinal;
  Unused: Integer;
  MinMaxInfo: PMinMaxInfo;
  Result: Longint;
end;
```

C++

```
struct TWMGetMinMaxInfo {
  unsigned Msg;
  int Unused;
  PMinMaxInfo MinMaxInfo;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMGetMinMaxInfo.

96.1.37 Messages.TWMGetText Record

Pascal

```
TWMGetText = packed record
  Msg: Cardinal;
  TextMax: Integer;
  Text: PChar;
  Result: Longint;
end;
```

C++

```
struct TWMGetText {
  unsigned Msg;
  int TextMax;
  const char * Text;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMGetText.

96.1.38 Messages.TWMHelp Record

Pascal

```
TWMHelp = packed record
  Msg: Cardinal;
  Unused: Integer;
  HelpInfo: PHelpInfo;
  Result: Longint;
end;
```

C++

```
struct TWMHelp {
  unsigned Msg;
  int Unused;
  PHelpInfo HelpInfo;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMHelp.

96.1.39 Messages.TWMHotKey Record

Pascal

```
TWMHotKey = packed record
  Msg: Cardinal;
  HotKey: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMHotKey {
  unsigned Msg;
  Longint HotKey;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMHotKey.

96.1.40 Messages.TWMHScrollClipboard Record

Pascal

```
TWMHScrollClipboard = packed record
  Msg: Cardinal;
  Viewer: HWND;
  ScrollCode: Word;
  Pos: Word;
  Result: Longint;
end;
```

C++

```
struct TWMHScrollClipboard {
  unsigned Msg;
  HWND Viewer;
  Word ScrollCode;
  Word Pos;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMHScrollClipboard.

96.1.41 Messages.TWMInitDialog Record

Pascal

```
TWMInitDialog = packed record
  Msg: Cardinal;
  Focus: HWND;
  InitParam: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMInitDialog {
  unsigned Msg;
  HWND Focus;
  Longint InitParam;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMInitDialog.

96.1.42 Messages.TWMInitMenu Record

Pascal

```
TWMInitMenu = packed record
  Msg: Cardinal;
  Menu: HMENU;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMInitMenu {
  unsigned Msg;
  HMENU Menu;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMInitMenu.

96.1.43 Messages.TWMInitMenuPopup Record

Pascal

```
TWMInitMenuPopup = packed record
  Msg: Cardinal;
  MenuPopup: HMENU;
  Pos: Smallint;
  SystemMenu: WordBool;
  Result: Longint;
end;
```

C++

```
struct TWMInitMenuPopup {
  unsigned Msg;
  HMENU MenuPopup;
  Smallint Pos;
  WordBool SystemMenu;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMInitMenuPopup.

96.1.44 Messages.TWMKey Record

Pascal

```
TWMKey = packed record
  Msg: Cardinal;
  CharCode: Word;
  Unused: Word;
  KeyData: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMKey {
  unsigned Msg;
  Word CharCode;
  Word Unused;
  Longint KeyData;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMKey.

96.1.45 Messages.TWMKillFocus Record

Pascal

```
TWMKillFocus = packed record
  Msg: Cardinal;
  FocusedWnd: HWND;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMKillFocus {
  unsigned Msg;
  HWND FocusedWnd;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMKillFocus.

96.1.46 Messages.TWMMDIActivate Record

Pascal

```
TWMMDIActivate = packed record
  Msg: Cardinal;
  case Integer of
    0: (ChildWnd: HWND;);
    1: (DeactiveWnd: HWND;
        ActiveWnd: HWND;
        Result: Longint;);
end;
```

C++

```
struct TWMMDIActivate {
  unsigned Msg;
  HWND ChildWnd;
  HWND DeactiveWnd;
  HWND ActiveWnd;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMDIActivate.

96.1.47 Messages.TWMMDICascade Record

Pascal

```
TWMMDICascade = packed record
  Msg: Cardinal;
  Cascade: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMMDICascade {
  unsigned Msg;
  Longint Cascade;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMDICascade.

96.1.48 Messages.TWMMDICreate Record

Pascal

```
TWMMDICreate = packed record
  Msg: Cardinal;
  Unused: Integer;
  MDICreateStruct: PMDICreateStruct;
  Result: Longint;
end;
```

C++

```
struct TWMMDICreate {
  unsigned Msg;
  int Unused;
  PMDICreateStruct MDICreateStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMDICreate.

96.1.49 Messages.TWMMDIDestroy Record

Pascal

```
TWMMDIDestroy = packed record
  Msg: Cardinal;
  Child: HWND;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMMDIDestroy {
  unsigned Msg;
  HWND Child;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMDIDestroy.

96.1.50 Messages.TWMMDIMaximize Record

Pascal

```
TWMMDIMaximize = packed record
  Msg: Cardinal;
  Maximize: HWND;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMMDIMaximize {
  unsigned Msg;
  HWND Maximize;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMDIMaximize.

96.1.51 Messages.TWMMDINext Record

Pascal

```
TWMMDINext = packed record
  Msg: Cardinal;
  Child: HWND;
  Next: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMMDINext {
  unsigned Msg;
  HWND Child;
  Longint Next;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMDINext.

96.1.52 Messages.TWMMDIRestore Record

Pascal

```
TWMMDIRestore = packed record
  Msg: Cardinal;
  IDChild: HWND;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMMDIRestore {
  unsigned Msg;
  HWND IDChild;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMDIRestore.

96.1.53 Messages.TWMMDISetMenu Record

Pascal

```
TWMMDISetMenu = packed record
  Msg: Cardinal;
  MenuFrame: HMENU;
  MenuWindow: HMENU;
  Result: Longint;
end;
```

C++

```
struct TWMMDISetMenu {
  unsigned Msg;
  HMENU MenuFrame;
  HMENU MenuWindow;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMDISetMenu.

96.1.54 Messages.TWMMDITile Record

Pascal

```
TWMMDITile = packed record
  Msg: Cardinal;
  Tile: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMMDITile {
  unsigned Msg;
  Longint Tile;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMDITile.

96.1.55 Messages.TWMMeasureItem Record

Pascal

```
TWMMeasureItem = packed record
  Msg: Cardinal;
  IDCtl: HWnd;
  MeasureItemStruct: PMeasureItemStruct;
  Result: Longint;
end;
```

C++

```
struct TWMMeasureItem {
  unsigned Msg;
  HWnd IDCtl;
  PMeasureItemStruct MeasureItemStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMeasureItem.

96.1.56 Messages.TWMMenuChar Record

Pascal

```
TWMMenuChar = packed record
  Msg: Cardinal;
  User: Char;
  Unused: Byte;
  MenuFlag: Word;
  Menu: HMENU;
  Result: Longint;
end;
```

C++

```
struct TWMMenuChar {
  unsigned Msg;
  Char User;
  Byte Unused;
  Word MenuFlag;
  HMENU Menu;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMenuChar.

96.1.57 Messages.TWMMenuSelect Record

Pascal

```
TWMMenuSelect = packed record
  Msg: Cardinal;
  IDItem: Word;
  MenuFlag: Word;
  Menu: HMENU;
  Result: Longint;
end;
```

C++

```
struct TWMMenuSelect {
  unsigned Msg;
  Word IDItem;
  Word MenuFlag;
  HMENU Menu;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMenuSelect.

96.1.58 Messages.TWMMouse Record

Pascal

```
TWMMouse = packed record
  Msg: Cardinal;
  Keys: Longint;
  case Integer of
    0: (XPos: Smallint;
        YPos: Smallint;);
    1: (Pos: TSmallPoint;
        Result: Longint;);
end;
```

C++

```
struct TWMMouse {
  unsigned Msg;
  Longint Keys;
  Smallint XPos;
  Smallint YPos;
  TSmallPoint Pos;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMouse.

96.1.59 Messages.TWMMouseActivate Record

Pascal

```
TWMMouseActivate = packed record
  Msg: Cardinal;
  TopLevel: HWND;
  HitTestCode: Word;
  MouseMsg: Word;
  Result: Longint;
end;
```

C++

```
struct TWMMouseActivate {
  unsigned Msg;
  HWND TopLevel;
  Word HitTestCode;
  Word MouseMsg;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMouseActivate.

96.1.60 Messages.TWMMouseWheel Record

Pascal

```
TWMMouseWheel = packed record
  Msg: Cardinal;
  Keys: SmallInt;
  WheelDelta: SmallInt;
  case Integer of
    0: (XPos: Smallint;
        YPos: Smallint;);
    1: (Pos: TSmallPoint;
        Result: Longint;);
end;
```

C++

```
struct TWMMouseWheel {
  unsigned Msg;
  SmallInt Keys;
  SmallInt WheelDelta;
  Smallint XPos;
  Smallint YPos;
  TSmallPoint Pos;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMMouseWheel.

96.1.61 Messages.TWMMove Record

Pascal

```
TWMMove = packed record
  Msg: Cardinal;
  Unused: Integer;
  case Integer of
    0: (XPos: Smallint;
        YPos: Smallint;);
    1: (Pos: TSmallPoint;
        Result: Longint;);
end;
```

C++

```
struct TWMMove {
  unsigned Msg;
  int Unused;
  Smallint XPos;
  Smallint YPos;
```

```
    TSmallPoint Pos;
    Longint Result;
  };
```

File

Messages

Description

This is record Messages.TWMMove.

96.1.62 Messages.TWMMoving Record

Pascal

```
  TWMMoving = packed record
    Msg: Cardinal;
    Edge: Integer;
    DragRect: PRect;
  end;
```

C++

```
  struct TWMMoving {
    unsigned Msg;
    int Edge;
    PRect DragRect;
  };
```

File

Messages

Description

This is record Messages.TWMMoving.

96.1.63 Messages.TWMNCActivate Record

Pascal

```
  TWMNCActivate = packed record
    Msg: Cardinal;
    Active: BOOL;
    Unused: Longint;
    Result: Longint;
  end;
```

C++

```
  struct TWMNCActivate {
    unsigned Msg;
    BOOL Active;
    Longint Unused;
    Longint Result;
  };
```

File

Messages

Description

This is record Messages.TWMNCActivate.

96.1.64 Messages.TWMNCCalcSize Record

Pascal

```
TWMNCCalcSize = packed record
  Msg: Cardinal;
  CalcValidRects: BOOL;
  CalcSize_Params: PNCCalcSizeParams;
  Result: Longint;
end;
```

C++

```
struct TWMNCCalcSize {
  unsigned Msg;
  BOOL CalcValidRects;
  PNCCalcSizeParams CalcSize_Params;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMNCCalcSize.

96.1.65 Messages.TWMNCCreate Record

Pascal

```
TWMNCCreate = packed record
  Msg: Cardinal;
  Unused: Integer;
  CreateStruct: PCreateStruct;
  Result: Longint;
end;
```

C++

```
struct TWMNCCreate {
  unsigned Msg;
  int Unused;
  PCreateStruct CreateStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMNCCreate.

96.1.66 Messages.TWMNCHitMessage Record

Pascal

```
TWMNCHitMessage = packed record
  Msg: Cardinal;
  HitTest: Longint;
  XCursor: Smallint;
  YCursor: Smallint;
  Result: Longint;
end;
```

C++

```
struct TWMNCHitMessage {
  unsigned Msg;
  Longint HitTest;
  Smallint XCursor;
  Smallint YCursor;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMNCHitMessage.

96.1.67 Messages.TWMNCHitTest Record

Pascal

```
TWMNCHitTest = packed record
  Msg: Cardinal;
  Unused: Longint;
  case Integer of
    0: (XPos: Smallint;
        YPos: Smallint;);
    1: (Pos: TSmallPoint;
        Result: Longint;);
end;
```

C++

```
struct TWMNCHitTest {
  unsigned Msg;
  Longint Unused;
  Smallint XPos;
  Smallint YPos;
  TSmallPoint Pos;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMNCHitTest.

96.1.68 Messages.TWMNCPaint Record

Pascal

```
TWMNCPaint = packed record
  Msg: Cardinal;
  RGN: HRGN;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMNCPaint {
  unsigned Msg;
  HRGN RGN;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMNCPaint.

96.1.69 Messages.TWMNextDlgCtl Record

Pascal

```
TWMNextDlgCtl = packed record
  Msg: Cardinal;
  CtlFocus: Longint;
  Handle: WordBool;
  Unused: Word;
  Result: Longint;
end;
```

C++

```
struct TWMNextDlgCtl {
  unsigned Msg;
  Longint CtlFocus;
  WordBool Handle;
  Word Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMNextDlgCtl.

96.1.70 Messages.TWMNoParams Record

Pascal

```
TWMNoParams = packed record
  Msg: Cardinal;
  Unused: array[0..3] of Word;
  Result: Longint;
end;
```

C++

```
struct TWMNoParams {
  unsigned Msg;
  array[0..3] of Word Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMNoParams.

96.1.71 Messages.TWMNotify Record

Pascal

```
TWMNotify = packed record
  Msg: Cardinal;
  IDCtrl: Longint;
  NMHdr: PNMHdr;
  Result: Longint;
end;
```

C++

```
struct TWMNotify {
  unsigned Msg;
  Longint IDCtrl;
  PNMHdr NMHdr;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMNotify.

96.1.72 Messages.TWMNotifyFormat Record

Pascal

```
TWMNotifyFormat = packed record
  Msg: Cardinal;
  From: HWND;
  Command: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMNotifyFormat {
  unsigned Msg;
  HWND From;
  Longint Command;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMNotifyFormat.

96.1.73 Messages.TWMPaint Record

Pascal

```
TWMPaint = packed record
  Msg: Cardinal;
  DC: HDC;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMPaint {
  unsigned Msg;
  HDC DC;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMPaint.

96.1.74 Messages.TWMPaintClipboard Record

Pascal

```
TWMPaintClipboard = packed record
  Msg: Cardinal;
  Viewer: HWND;
  PaintStruct: THandle;
  Result: Longint;
end;
```

C++

```
struct TWMPaintClipboard {
  unsigned Msg;
  HWND Viewer;
  THandle PaintStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMPaintClipboard.

96.1.75 Messages.TWMPaletteChanged Record

Pascal

```
TWMPaletteChanged = packed record
  Msg: Cardinal;
  PalChg: HWND;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMPaletteChanged {
  unsigned Msg;
  HWND PalChg;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMPaletteChanged.

96.1.76 Messages.TWMPaletteIsChanging Record

Pascal

```
TWMPaletteIsChanging = packed record
  Msg: Cardinal;
  Realize: HWND;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMPaletteIsChanging {
  unsigned Msg;
  HWND Realize;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMPaletteIsChanging.

96.1.77 Messages.TWMParentNotify Record

Pascal

```
TWMParentNotify = packed record
  Msg: Cardinal;
  Event: Word;
  case Word of
    0: (Value1: Word;
        Value2: Longint;
        Result: Longint;);
    WM_DESTROY: (ChildID: Word;
                 ChildWnd: HWnd;);
    WM_RBUTTONDOWN: (Value: Word;
                     XPos: Smallint;
                     YPos: Smallint;);
end;
```

C++

```
struct TWMParentNotify {
  unsigned Msg;
  Word Event;
  Word ChildID;
  HWnd ChildWnd;
  Word Value;
  Smallint XPos;
  Smallint YPos;
  Word Value1;
  Longint Value2;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMParentNotify.

96.1.78 Messages.TWMPower Record

Pascal

```
TWMPower = packed record
  Msg: Cardinal;
  PowerEvt: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMPower {
  unsigned Msg;
  Longint PowerEvt;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMPower.

96.1.79 Messages.TWMPrint Record

Pascal

```
TWMPrint = packed record
  Msg: Cardinal;
  DC: HDC;
  Flags: Cardinal;
  Result: Integer;
end;
```

C++

```
struct TWMPrint {
  unsigned Msg;
  HDC DC;
  unsigned Flags;
  int Result;
};
```

File

Messages

Description

This is record Messages.TWMPrint.

96.1.80 Messages.TWMQueryEndSession Record

Pascal

```
TWMQueryEndSession = packed record
  Msg: Cardinal;
  Source: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMQueryEndSession {
  unsigned Msg;
  Longint Source;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMQueryEndSession.

96.1.81 Messages.TWMQuit Record

Pascal

```
TWMQuit = packed record
  Msg: Cardinal;
  ExitCode: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMQuit {
  unsigned Msg;
  Longint ExitCode;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMQuit.

96.1.82 Messages.TWMRenderFormat Record

Pascal

```
TWMRenderFormat = packed record
  Msg: Cardinal;
  Format: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMRenderFormat {
  unsigned Msg;
  Longint Format;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMRenderFormat.

96.1.83 Messages.TWMScroll Record

Pascal

```
TWMScroll = packed record
  Msg: Cardinal;
  ScrollCode: Smallint;
  Pos: Smallint;
  ScrollBar: HWND;
  Result: Longint;
end;
```

C++

```
struct TWMScroll {
  unsigned Msg;
  Smallint ScrollCode;
  Smallint Pos;
  HWND ScrollBar;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMScroll.

96.1.84 Messages.TWMSetCursor Record

Pascal

```
TWMSetCursor = packed record
  Msg: Cardinal;
  CursorWnd: HWND;
  HitTest: Word;
  MouseMsg: Word;
  Result: Longint;
end;
```

C++

```
struct TWMSetCursor {
  unsigned Msg;
  HWND CursorWnd;
  Word HitTest;
  Word MouseMsg;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSetCursor.

96.1.85 Messages.TWMSetFocus Record

Pascal

```
TWMSetFocus = packed record
  Msg: Cardinal;
  FocusedWnd: HWND;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMSetFocus {
  unsigned Msg;
  HWND FocusedWnd;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSetFocus.

96.1.86 Messages.TWMSetFont Record

Pascal

```
TWMSetFont = packed record
  Msg: Cardinal;
  Font: HFONT;
  Redraw: WordBool;
  Unused: Word;
  Result: Longint;
end;
```

C++

```
struct TWMSetFont {
  unsigned Msg;
  HFONT Font;
  WordBool Redraw;
  Word Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSetFont.

96.1.87 Messages.TWMSetHotKey Record

Pascal

```
TWMSetHotKey = packed record
  Msg: Cardinal;
  Key: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMSetHotKey {
  unsigned Msg;
  Longint Key;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSetHotKey.

96.1.88 Messages.TWMSetIcon Record

Pascal

```
TWMSetIcon = packed record
  Msg: Cardinal;
  BigIcon: Longbool;
  Icon: HICON;
  Result: Longint;
end;
```

C++

```
struct TWMSetIcon {
  unsigned Msg;
  Longbool BigIcon;
  HICON Icon;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSetIcon.

96.1.89 Messages.TWMSetRedraw Record

Pascal

```
TWMSetRedraw = packed record
  Msg: Cardinal;
  Redraw: Longint;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMSetRedraw {
  unsigned Msg;
  Longint Redraw;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSetRedraw.

96.1.90 Messages.TWMSetText Record

Pascal

```
TWMSetText = packed record
  Msg: Cardinal;
  Unused: Longint;
  Text: PChar;
  Result: Longint;
end;
```

C++

```
struct TWMSetText {
  unsigned Msg;
  Longint Unused;
  const char * Text;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSetText.

96.1.91 Messages.TWMSettingChange Record

Pascal

```
TWMSettingChange = packed record
  Msg: Cardinal;
  Flag: Integer;
  Section: PChar;
  Result: Longint;
end;
```

C++

```
struct TWMSettingChange {
  unsigned Msg;
  int Flag;
  const char * Section;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSettingChange.

96.1.92 Messages.TWMShowWindow Record

Pascal

```
TWMShowWindow = packed record
  Msg: Cardinal;
  Show: BOOL;
  Status: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMShowWindow {
  unsigned Msg;
  BOOL Show;
  Longint Status;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMShowWindow.

96.1.93 Messages.TWMSize Record

Pascal

```
TWMSize = packed record
  Msg: Cardinal;
  SizeType: Longint;
  Width: Word;
  Height: Word;
  Result: Longint;
end;
```

C++

```
struct TWMSize {
  unsigned Msg;
  Longint SizeType;
  Word Width;
  Word Height;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSize.

96.1.94 Messages.TWMSizeClipboard Record

Pascal

```
TWMSizeClipboard = packed record
  Msg: Cardinal;
  Viewer: HWND;
  RC: THandle;
  Result: Longint;
end;
```

C++

```
struct TWMSizeClipboard {
  unsigned Msg;
  HWND Viewer;
  THandle RC;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSizeClipboard.

96.1.95 Messages.TWMSpoolerStatus Record

Pascal

```
TWMSpoolerStatus = packed record
  Msg: Cardinal;
  JobStatus: Longint;
  JobsLeft: Word;
  Unused: Word;
  Result: Longint;
end;
```

C++

```
struct TWMSpoolerStatus {
  unsigned Msg;
  Longint JobStatus;
  Word JobsLeft;
  Word Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSpoolerStatus.

96.1.96 Messages.TWMStyleChange Record

Pascal

```
TWMStyleChange = packed record
  Msg: Cardinal;
  StyleType: Longint;
  StyleStruct: PStyleStruct;
  Result: Longint;
end;
```

C++

```
struct TWMStyleChange {
  unsigned Msg;
  Longint StyleType;
  PStyleStruct StyleStruct;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMStyleChange.

96.1.97 Messages.TWMSysCommand Record

Pascal

```
TWMSysCommand = packed record
  Msg: Cardinal;
  CmdType: Longint;
  case Longint of
    SC_HOTKEY: (ActivateWnd: HWND;);
    SC_KEYMENU: (Key: Word;);
    SC_VSCROLL: (XPos: Smallint;
                 YPos: Smallint;
                 Result: Longint;);
end;
```

C++

```
struct TWMSysCommand {
  unsigned Msg;
  Longint CmdType;
  HWND ActivateWnd;
  Word Key;
  Smallint XPos;
  Smallint YPos;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSysCommand.

96.1.98 Messages.TWMSysDeadChar Record

Pascal

```
TWMSysDeadChar = packed record
  Msg: Cardinal;
  CharCode: Word;
  Unused: Word;
  KeyData: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMSysDeadChar {
  unsigned Msg;
  Word CharCode;
  Word Unused;
  Longint KeyData;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSysDeadChar.

96.1.99 Messages.TWMSystemError Record

Pascal

```
TWMSystemError = packed record
  Msg: Cardinal;
  ErrSpec: Word;
  Unused: Longint;
  Result: Longint;
end;
```

C++

```
struct TWMSystemError {
  unsigned Msg;
  Word ErrSpec;
  Longint Unused;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMSystemError.

96.1.100 Messages.TWMTimer Record

Pascal

```
TWMTimer = packed record
  Msg: Cardinal;
  TimerID: Longint;
  TimerProc: TFarProc;
  Result: Longint;
end;
```

C++

```
struct TWMTimer {
  unsigned Msg;
  Longint TimerID;
  TFarProc TimerProc;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMTimer.

96.1.101 Messages.TWMUIState Record

Pascal

```
TWMUIState = packed record
  Msg: Cardinal;
  Action: Word;
  Flags: Word;
  Unused: Longint;
end;
```

C++

```
struct TWMUIState {
  unsigned Msg;
  Word Action;
  Word Flags;
  Longint Unused;
};
```

File

Messages

Description

This is record Messages.TWMUIState.

96.1.102 Messages.TWMVScrollClipboard Record

Pascal

```
TWMVScrollClipboard = packed record
  Msg: Cardinal;
  Viewer: HWND;
  ScollCode: Word;
  ThumbPos: Word;
  Result: Longint;
end;
```

C++

```
struct TWMVScrollClipboard {
  unsigned Msg;
  HWND Viewer;
  Word ScollCode;
  Word ThumbPos;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMVScrollClipboard.

96.1.103 Messages.TWMWindowPosMsg Record

Pascal

```
TWMWindowPosMsg = packed record
  Msg: Cardinal;
  Unused: Integer;
  WindowPos: PWindowPos;
  Result: Longint;
end;
```

C++

```
struct TWMWindowPosMsg {
  unsigned Msg;
  int Unused;
  PWindowPos WindowPos;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMWindowPosMsg.

96.1.104 Messages.TWMWinIniChange Record

Pascal

```
TWMWinIniChange = packed record
  Msg: Cardinal;
  Unused: Integer;
  Section: PChar;
  Result: Longint;
end;
```

C++

```
struct TWMWinIniChange {
  unsigned Msg;
  int Unused;
  const char * Section;
  Longint Result;
};
```

File

Messages

Description

This is record Messages.TWMWinIniChange.

96.2 Types

The following table lists types in this documentation.

Types

Name	Description
PMessage (see page 16279)	This is type Messages.PMessage.
TWMCancelMode (see page 16280)	This is type Messages.TWMCancelMode.
TWMChangeUIState (see page 16280)	This is type Messages.TWMChangeUIState.
TWMChar (see page 16280)	This is type Messages.TWMChar.
TWMChildActivate (see page 16281)	This is type Messages.TWMChildActivate.
TWMClear (see page 16281)	This is type Messages.TWMClear.
TWMClose (see page 16281)	This is type Messages.TWMClose.
TWMCopy (see page 16282)	This is type Messages.TWMCopy.
TWMCtlColorBtn (see page 16282)	This is type Messages.TWMCtlColorBtn.
TWMCtlColorDlg (see page 16282)	This is type Messages.TWMCtlColorDlg.
TWMCtlColorEdit (see page 16283)	This is type Messages.TWMCtlColorEdit.
TWMCtlColorListbox (see page 16283)	This is type Messages.TWMCtlColorListbox.
TWMCtlColorMsgbox (see page 16283)	This is type Messages.TWMCtlColorMsgbox.
TWMCtlColorScrollbar (see page 16284)	This is type Messages.TWMCtlColorScrollbar.
TWMCtlColorStatic (see page 16284)	This is type Messages.TWMCtlColorStatic.
TWMCut (see page 16284)	This is type Messages.TWMCut.
TWMDeadChar (see page 16285)	This is type Messages.TWMDeadChar.
TWMDestroy (see page 16285)	This is type Messages.TWMDestroy.
TWMDestroyClipboard (see page 16285)	This is type Messages.TWMDestroyClipboard.
TWMDrawClipboard (see page 16286)	This is type Messages.TWMDrawClipboard.
TWMExitMenuLoop (see page 16286)	This is type Messages.TWMExitMenuLoop.
TWMFontChange (see page 16286)	This is type Messages.TWMFontChange.
TWMGetDlgCode (see page 16287)	This is type Messages.TWMGetDlgCode.
TWMGetFont (see page 16287)	This is type Messages.TWMGetFont.
TWMGetHotKey (see page 16287)	This is type Messages.TWMGetHotKey.
TWMGetTextLength (see page 16288)	This is type Messages.TWMGetTextLength.
TWMHScroll (see page 16288)	This is type Messages.TWMHScroll.
TWMIconEraseBkgnd (see page 16288)	This is type Messages.TWMIconEraseBkgnd.
TWMKeyDown (see page 16289)	This is type Messages.TWMKeyDown.
TWMKeyUp (see page 16289)	This is type Messages.TWMKeyUp.
TWMLButtonDblClk (see page 16289)	This is type Messages.TWMLButtonDblClk.
TWMLButtonDown (see page 16290)	TWMLButtonDown represents the Windows message that occurs when the user clicks the left mouse button.
TWMLButtonUp (see page 16290)	This is type Messages.TWMLButtonUp.
TWMMButtonDblClk (see page 16290)	This is type Messages.TWMMButtonDblClk.
TWMMButtonDown (see page 16291)	This is type Messages.TWMMButtonDown.
TWMMButtonUp (see page 16291)	This is type Messages.TWMMButtonUp.
TWMMDIGetActive (see page 16291)	This is type Messages.TWMMDIGetActive.
TWMMDIIconArrange (see page 16292)	This is type Messages.TWMMDIIconArrange.
TWMMDIRefreshMenu (see page 16292)	This is type Messages.TWMMDIRefreshMenu.
TWMMouseMove (see page 16292)	This is type Messages.TWMMouseMove.
TWMNCDestroy (see page 16293)	This is type Messages.TWMNCDestroy.
TWMNCLButtonDblClk (see page 16293)	This is type Messages.TWMNCLButtonDblClk.
TWMNCLButtonDown (see page 16293)	This is type Messages.TWMNCLButtonDown.
TWMNCLButtonUp (see page 16294)	This is type Messages.TWMNCLButtonUp.

TWMNCMButtonDblClk (see page 16294)	This is type Messages.TWMNCMButtonDblClk.
TWMNCMButtonDown (see page 16294)	This is type Messages.TWMNCMButtonDown.
TWMNCMButtonUp (see page 16295)	This is type Messages.TWMNCMButtonUp.
TWMNCMouseMove (see page 16295)	This is type Messages.TWMNCMouseMove.
TWMNCRButtonDblClk (see page 16295)	This is type Messages.TWMNCRButtonDblClk.
TWMNCRButtonDown (see page 16296)	This is type Messages.TWMNCRButtonDown.
TWMNCRButtonUp (see page 16296)	This is type Messages.TWMNCRButtonUp.
TWMPaintIcon (see page 16296)	This is type Messages.TWMPaintIcon.
TWMPaste (see page 16297)	This is type Messages.TWMPaste.
TWMPrintClient (see page 16297)	This is type Messages.TWMPrintClient.
TWMQueryDragIcon (see page 16297)	This is type Messages.TWMQueryDragIcon.
TWMQueryNewPalette (see page 16298)	This is type Messages.TWMQueryNewPalette.
TWMQueryOpen (see page 16298)	This is type Messages.TWMQueryOpen.
TWMQueryUIState (see page 16298)	This is type Messages.TWMQueryUIState.
TWMQueueSync (see page 16299)	This is type Messages.TWMQueueSync.
TWMRButtonDblClk (see page 16299)	This is type Messages.TWMRButtonDblClk.
TWMRButtonDown (see page 16299)	This is type Messages.TWMRButtonDown.
TWMRButtonUp (see page 16300)	This is type Messages.TWMRButtonUp.
TWMRenderAllFormats (see page 16300)	This is type Messages.TWMRenderAllFormats.
TWMStyleChanged (see page 16300)	This is type Messages.TWMStyleChanged.
TWMStyleChanging (see page 16301)	This is type Messages.TWMStyleChanging.
TWMSysChar (see page 16301)	This is type Messages.TWMSysChar.
TWMSysColorChange (see page 16301)	This is type Messages.TWMSysColorChange.
TWMSysKeyDown (see page 16302)	This is type Messages.TWMSysKeyDown.
TWMSysKeyUp (see page 16302)	This is type Messages.TWMSysKeyUp.
TWMTimeChange (see page 16302)	This is type Messages.TWMTimeChange.
TWMUndo (see page 16303)	This is type Messages.TWMUndo.
TWMUpdateUIState (see page 16303)	This is type Messages.TWMUpdateUIState.
TWMVKeyToItem (see page 16303)	This is type Messages.TWMVKeyToItem.
TWMVScroll (see page 16304)	This is type Messages.TWMVScroll.
TWMWindowPosChanged (see page 16304)	This is type Messages.TWMWindowPosChanged.
TWMWindowPosChanging (see page 16304)	This is type Messages.TWMWindowPosChanging.

96.2.1 Messages.PMessage Type

Pascal

```
PMessage = ^TMessage;
```

C++

```
^TMessage PMessage;
```

File

Messages

Description

This is type Messages.PMessage.

96.2.2 Messages.TWMCancelMode Type

Pascal

```
TWMCancelMode = TWMNoParams;
```

C++

```
TWMNoParams TWMCancelMode;
```

File

Messages

Description

This is type Messages.TWMCancelMode.

96.2.3 Messages.TWMChangeUIState Type

Pascal

```
TWMChangeUIState = TWMUIState;
```

C++

```
TWMUIState TWMChangeUIState;
```

File

Messages

Description

This is type Messages.TWMChangeUIState.

96.2.4 Messages.TWMChar Type

Pascal

```
TWMChar = TWMKey;
```

C++

```
TWMKey TWMChar;
```

File

Messages

Description

This is type Messages.TWMChar.

96.2.5 Messages.TWMChildActivate Type

Pascal

```
TWMChildActivate = TWMNoParams;
```

C++

```
TWMNoParams TWMChildActivate;
```

File

Messages

Description

This is type Messages.TWMChildActivate.

96.2.6 Messages.TWMClear Type

Pascal

```
TWMClear = TWMNoParams;
```

C++

```
TWMNoParams TWMClear;
```

File

Messages

Description

This is type Messages.TWMClear.

96.2.7 Messages.TWMClose Type

Pascal

```
TWMClose = TWMNoParams;
```

C++

```
TWMNoParams TWMClose;
```

File

Messages

Description

This is type Messages.TWMClose.

96.2.8 Messages.TWMCopy Type

Pascal

```
TWMCopy = TWMNoParams;
```

C++

```
TWMNoParams TWMCopy;
```

File

Messages

Description

This is type Messages.TWMCopy.

96.2.9 Messages.TWMCtlColorBtn Type

Pascal

```
TWMCtlColorBtn = TWMCtlColor;
```

C++

```
TWMCtlColor TWMCtlColorBtn;
```

File

Messages

Description

This is type Messages.TWMCtlColorBtn.

96.2.10 Messages.TWMCtlColorDlg Type

Pascal

```
TWMCtlColorDlg = TWMCtlColor;
```

C++

```
TWMCtlColor TWMCtlColorDlg;
```

File

Messages

Description

This is type Messages.TWMCtlColorDlg.

96.2.11 Messages.TWMCtlColorEdit Type

Pascal

```
TWMCtlColorEdit = TWMCtlColor;
```

C++

```
TWMCtlColor TWMCtlColorEdit;
```

File

Messages

Description

This is type Messages.TWMCtlColorEdit.

96.2.12 Messages.TWMCtlColorListbox Type

Pascal

```
TWMCtlColorListbox = TWMCtlColor;
```

C++

```
TWMCtlColor TWMCtlColorListbox;
```

File

Messages

Description

This is type Messages.TWMCtlColorListbox.

96.2.13 Messages.TWMCtlColorMsgbox Type

Pascal

```
TWMCtlColorMsgbox = TWMCtlColor;
```

C++

```
TWMCtlColor TWMCtlColorMsgbox;
```

File

Messages

Description

This is type Messages.TWMCtlColorMsgbox.

96.2.14 Messages.TWMCtlColorScrollbar Type

Pascal

```
TWMCtlColorScrollbar = TWMCtlColor;
```

C++

```
TWMCtlColor TWMCtlColorScrollbar;
```

File

Messages

Description

This is type Messages.TWMCtlColorScrollbar.

96.2.15 Messages.TWMCtlColorStatic Type

Pascal

```
TWMCtlColorStatic = TWMCtlColor;
```

C++

```
TWMCtlColor TWMCtlColorStatic;
```

File

Messages

Description

This is type Messages.TWMCtlColorStatic.

96.2.16 Messages.TWMCut Type

Pascal

```
TWMCut = TWMNoParams;
```

C++

```
TWMNoParams TWMCut;
```

File

Messages

Description

This is type Messages.TWMCut.

96.2.17 Messages.TWMDeadChar Type

Pascal

```
TWMDeadChar = TWMChar;
```

C++

```
TWMChar TWMDeadChar;
```

File

Messages

Description

This is type Messages.TWMDeadChar.

96.2.18 Messages.TWMDestroy Type

Pascal

```
TWMDestroy = TWMNoParams;
```

C++

```
TWMNoParams TWMDestroy;
```

File

Messages

Description

This is type Messages.TWMDestroy.

96.2.19 Messages.TWMDestroyClipboard Type

Pascal

```
TWMDestroyClipboard = TWMNoParams;
```

C++

```
TWMNoParams TWMDestroyClipboard;
```

File

Messages

Description

This is type Messages.TWMDestroyClipboard.

96.2.20 Messages.TWMDrawClipboard Type

Pascal

```
TWMDrawClipboard = TWMNoParams;
```

C++

```
TWMNoParams TWMDrawClipboard;
```

File

Messages

Description

This is type Messages.TWMDrawClipboard.

96.2.21 Messages.TWMExitMenuLoop Type

Pascal

```
TWMExitMenuLoop = TWMEnterMenuLoop;
```

C++

```
TWMEnterMenuLoop TWMExitMenuLoop;
```

File

Messages

Description

This is type Messages.TWMExitMenuLoop.

96.2.22 Messages.TWMFontChange Type

Pascal

```
TWMFontChange = TWMNoParams;
```

C++

```
TWMNoParams TWMFontChange;
```

File

Messages

Description

This is type Messages.TWMFontChange.

96.2.23 Messages.TWMGetDlgCode Type

Pascal

```
TWMGetDlgCode = TWMNoParams;
```

C++

```
TWMNoParams TWMGetDlgCode;
```

File

Messages

Description

This is type Messages.TWMGetDlgCode.

96.2.24 Messages.TWMGetFont Type

Pascal

```
TWMGetFont = TWMNoParams;
```

C++

```
TWMNoParams TWMGetFont;
```

File

Messages

Description

This is type Messages.TWMGetFont.

96.2.25 Messages.TWMGetHotKey Type

Pascal

```
TWMGetHotKey = TWMNoParams;
```

C++

```
TWMNoParams TWMGetHotKey;
```

File

Messages

Description

This is type Messages.TWMGetHotKey.

96.2.26 Messages.TWMGetTextLength Type

Pascal

```
TWMGetTextLength = TWMNoParams;
```

C++

```
TWMNoParams TWMGetTextLength;
```

File

Messages

Description

This is type Messages.TWMGetTextLength.

96.2.27 Messages.TWMHScroll Type

Pascal

```
TWMHScroll = TWMScroll;
```

C++

```
TWMScroll TWMHScroll;
```

File

Messages

Description

This is type Messages.TWMHScroll.

96.2.28 Messages.TWMIconEraseBkgnd Type

Pascal

```
TWMIconEraseBkgnd = TWMEraseBkgnd;
```

C++

```
TWMEraseBkgnd TWMIconEraseBkgnd;
```

File

Messages

Description

This is type Messages.TWMIconEraseBkgnd.

96.2.29 Messages.TWMKeyDown Type

Pascal

```
TWMKeyDown = TWMKey;
```

C++

```
TWMKey TWMKeyDown;
```

File

Messages

Description

This is type Messages.TWMKeyDown.

96.2.30 Messages.TWMKeyUp Type

Pascal

```
TWMKeyUp = TWMKey;
```

C++

```
TWMKey TWMKeyUp;
```

File

Messages

Description

This is type Messages.TWMKeyUp.

96.2.31 Messages.TWMLButtonDblClk Type

Pascal

```
TWMLButtonDblClk = TWMMouse;
```

C++

```
TWMMouse TWMLButtonDblClk;
```

File

Messages

Description

This is type Messages.TWMLButtonDblClk.

96.2.32 Messages.TWMLButtonDown Type

TWMLButtonDown represents the Windows message that occurs when the user clicks the left mouse button.

Pascal

```
TWMLButtonDown = TWMMouse;
```

C++

```
TWMMouse TWMLButtonDown;
```

File

Messages

Description

TWMLButtonDown is the data structure that CLX uses for Windows messages informing an application that the user just clicked the left mouse button. It is the same as all mouse messages.

See Also

TMessage

TWMMouse

TWMPaint

TWMMenuChar

TWMNotify

TWMKey

96.2.33 Messages.TWMLButtonUp Type

Pascal

```
TWMLButtonUp = TWMMouse;
```

C++

```
TWMMouse TWMLButtonUp;
```

File

Messages

Description

This is type Messages.TWMLButtonUp.

96.2.34 Messages.TWMMButtonDblClk Type

Pascal

```
TWMMButtonDblClk = TWMMouse;
```

C++

```
TWMMouse TWMMButtonDblClk;
```

File

Messages

Description

This is type Messages.TWMMButtonDblClk.

96.2.35 Messages.TWMMButtonDown Type

Pascal

```
TWMMButtonDown = TWMMouse;
```

C++

```
TWMMouse TWMMButtonDown;
```

File

Messages

Description

This is type Messages.TWMMButtonDown.

96.2.36 Messages.TWMMButtonUp Type

Pascal

```
TWMMButtonUp = TWMMouse;
```

C++

```
TWMMouse TWMMButtonUp;
```

File

Messages

Description

This is type Messages.TWMMButtonUp.

96.2.37 Messages.TWMMDIGetActive Type

Pascal

```
TWMMDIGetActive = TWMNoParams;
```

C++

```
TWMNoParams TWMMDIGetActive;
```

File

Messages

Description

This is type Messages.TWMMDIGetActive.

96.2.38 Messages.TWMMDIIconArrange Type

Pascal

```
TWMMDIIconArrange = TWMNoParams;
```

C++

```
TWMNoParams TWMMDIIconArrange;
```

File

Messages

Description

This is type Messages.TWMMDIIconArrange.

96.2.39 Messages.TWMMDIRefreshMenu Type

Pascal

```
TWMMDIRefreshMenu = TWMNoParams;
```

C++

```
TWMNoParams TWMMDIRefreshMenu;
```

File

Messages

Description

This is type Messages.TWMMDIRefreshMenu.

96.2.40 Messages.TWMMouseMove Type

Pascal

```
TWMMouseMove = TWMMouse;
```

C++

```
TWMMouse TWMMouseMove;
```

File

Messages

Description

This is type Messages.TWMMouseMove.

96.2.41 Messages.TWMNCDestroy Type

Pascal

```
TWMNCDestroy = TWMNoParams;
```

C++

```
TWMNoParams TWMNCDestroy;
```

File

Messages

Description

This is type Messages.TWMNCDestroy.

96.2.42 Messages.TWMNCLButtonDblClk Type

Pascal

```
TWMNCLButtonDblClk = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCLButtonDblClk;
```

File

Messages

Description

This is type Messages.TWMNCLButtonDblClk.

96.2.43 Messages.TWMNCLButtonDown Type

Pascal

```
TWMNCLButtonDown = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCLButtonDown;
```

File

Messages

Description

This is type Messages.TWMNCLButtonDown.

96.2.44 Messages.TWMNCLButtonUp Type

Pascal

```
TWMNCLButtonUp = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCLButtonUp;
```

File

Messages

Description

This is type Messages.TWMNCLButtonUp.

96.2.45 Messages.TWMNCMButtonDblClk Type

Pascal

```
TWMNCMButtonDblClk = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCMButtonDblClk;
```

File

Messages

Description

This is type Messages.TWMNCMButtonDblClk.

96.2.46 Messages.TWMNCMButtonDown Type

Pascal

```
TWMNCMButtonDown = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCMButtonDown;
```

File

Messages

Description

This is type Messages.TWMNCMButtonDown.

96.2.47 Messages.TWMNCMButtonUp Type

Pascal

```
TWMNCMButtonUp = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCMButtonUp;
```

File

Messages

Description

This is type Messages.TWMNCMButtonUp.

96.2.48 Messages.TWMNCMouseMove Type

Pascal

```
TWMNCMouseMove = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCMouseMove;
```

File

Messages

Description

This is type Messages.TWMNCMouseMove.

96.2.49 Messages.TWMNCRButtonDblClk Type

Pascal

```
TWMNCRButtonDblClk = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCRButtonDblClk;
```

File

Messages

Description

This is type Messages.TWMNCRButtonDblClk.

96.2.50 Messages.TWMNCRButtonDown Type

Pascal

```
TWMNCRButtonDown = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCRButtonDown;
```

File

Messages

Description

This is type Messages.TWMNCRButtonDown.

96.2.51 Messages.TWMNCRButtonUp Type

Pascal

```
TWMNCRButtonUp = TWMNCHitMessage;
```

C++

```
TWMNCHitMessage TWMNCRButtonUp;
```

File

Messages

Description

This is type Messages.TWMNCRButtonUp.

96.2.52 Messages.TWMPaintIcon Type

Pascal

```
TWMPaintIcon = TWMNoParams;
```

C++

```
TWMNoParams TWMPaintIcon;
```

File

Messages

Description

This is type Messages.TWMPaintIcon.

96.2.53 Messages.TWMPaste Type

Pascal

```
TWMPaste = TWMNoParams;
```

C++

```
TWMNoParams TWMPaste;
```

File

Messages

Description

This is type Messages.TWMPaste.

96.2.54 Messages.TWMPrintClient Type

Pascal

```
TWMPrintClient = TWMPrint;
```

C++

```
TWMPrint TWMPrintClient;
```

File

Messages

Description

This is type Messages.TWMPrintClient.

96.2.55 Messages.TWMQueryDragIcon Type

Pascal

```
TWMQueryDragIcon = TWMNoParams;
```

C++

```
TWMNoParams TWMQueryDragIcon;
```

File

Messages

Description

This is type Messages.TWMQueryDragIcon.

96.2.56 Messages.TWMQueryNewPalette Type

Pascal

```
TWMQueryNewPalette = TWMNoParams;
```

C++

```
TWMNoParams TWMQueryNewPalette;
```

File

Messages

Description

This is type Messages.TWMQueryNewPalette.

96.2.57 Messages.TWMQueryOpen Type

Pascal

```
TWMQueryOpen = TWMNoParams;
```

C++

```
TWMNoParams TWMQueryOpen;
```

File

Messages

Description

This is type Messages.TWMQueryOpen.

96.2.58 Messages.TWMQueryUIState Type

Pascal

```
TWMQueryUIState = TWMNoParams;
```

C++

```
TWMNoParams TWMQueryUIState;
```

File

Messages

Description

This is type Messages.TWMQueryUIState.

96.2.59 Messages.TWMQueueSync Type

Pascal

```
TWMQueueSync = TWMNoParams;
```

C++

```
TWMNoParams TWMQueueSync;
```

File

Messages

Description

This is type Messages.TWMQueueSync.

96.2.60 Messages.TWMRButtonDblClk Type

Pascal

```
TWMRButtonDblClk = TWMMouse;
```

C++

```
TWMMouse TWMRButtonDblClk;
```

File

Messages

Description

This is type Messages.TWMRButtonDblClk.

96.2.61 Messages.TWMRButtonDown Type

Pascal

```
TWMRButtonDown = TWMMouse;
```

C++

```
TWMMouse TWMRButtonDown;
```

File

Messages

Description

This is type Messages.TWMRButtonDown.

96.2.62 Messages.TWMRButtonUp Type

Pascal

```
TWMRButtonUp = TWMMouse;
```

C++

```
TWMMouse TWMRButtonUp;
```

File

Messages

Description

This is type Messages.TWMRButtonUp.

96.2.63 Messages.TWMRenderAllFormats Type

Pascal

```
TWMRenderAllFormats = TWMNoParams;
```

C++

```
TWMNoParams TWMRenderAllFormats;
```

File

Messages

Description

This is type Messages.TWMRenderAllFormats.

96.2.64 Messages.TWMStyleChanged Type

Pascal

```
TWMStyleChanged = TWMStyleChange;
```

C++

```
TWMStyleChange TWMStyleChanged;
```

File

Messages

Description

This is type Messages.TWMStyleChanged.

96.2.65 Messages.TWMStyleChanging Type

Pascal

```
TWMStyleChanging = TWMStyleChange;
```

C++

```
TWMStyleChange TWMStyleChanging;
```

File

Messages

Description

This is type Messages.TWMStyleChanging.

96.2.66 Messages.TWMSysChar Type

Pascal

```
TWMSysChar = TWMKey;
```

C++

```
TWMKey TWMSysChar;
```

File

Messages

Description

This is type Messages.TWMSysChar.

96.2.67 Messages.TWMSysColorChange Type

Pascal

```
TWMSysColorChange = TWMNoParams;
```

C++

```
TWMNoParams TWMSysColorChange;
```

File

Messages

Description

This is type Messages.TWMSysColorChange.

96.2.68 Messages.TWMSysKeyDown Type

Pascal

```
TWMSysKeyDown = TWMKey;
```

C++

```
TWMKey TWMSysKeyDown;
```

File

Messages

Description

This is type Messages.TWMSysKeyDown.

96.2.69 Messages.TWMSysKeyUp Type

Pascal

```
TWMSysKeyUp = TWMKey;
```

C++

```
TWMKey TWMSysKeyUp;
```

File

Messages

Description

This is type Messages.TWMSysKeyUp.

96.2.70 Messages.TWMTimeChange Type

Pascal

```
TWMTimeChange = TWMNoParams;
```

C++

```
TWMNoParams TWMTimeChange;
```

File

Messages

Description

This is type Messages.TWMTimeChange.

96.2.71 Messages.TWMUndo Type

Pascal

```
TWMUndo = TWMNoParams;
```

C++

```
TWMNoParams TWMUndo;
```

File

Messages

Description

This is type Messages.TWMUndo.

96.2.72 Messages.TWMUpdateUIState Type

Pascal

```
TWMUpdateUIState = TWMUIState;
```

C++

```
TWMUIState TWMUpdateUIState;
```

File

Messages

Description

This is type Messages.TWMUpdateUIState.

96.2.73 Messages.TWMVKeyToItem Type

Pascal

```
TWMVKeyToItem = TWMCharToItem;
```

C++

```
TWMCharToItem TWMVKeyToItem;
```

File

Messages

Description

This is type Messages.TWMVKeyToItem.

96.2.74 Messages.TWMVScroll Type

Pascal

```
TWMVScroll = TWMScroll;
```

C++

```
TWMScroll TWMVScroll;
```

File

Messages

Description

This is type Messages.TWMVScroll.

96.2.75 Messages.TWMWindowPosChanged Type

Pascal

```
TWMWindowPosChanged = TWMWindowPosMsg;
```

C++

```
TWMWindowPosMsg TWMWindowPosChanged;
```

File

Messages

Description

This is type Messages.TWMWindowPosChanged.

96.2.76 Messages.TWMWindowPosChanging Type

Pascal

```
TWMWindowPosChanging = TWMWindowPosMsg;
```

C++

```
TWMWindowPosMsg TWMWindowPosChanging;
```

File

Messages

Description

This is type Messages.TWMWindowPosChanging.

96.3 Constants

The following table lists constants in this documentation.

Constants

Name	Description
BM_CLICK (see page 16312)	This is constant Messages.BM_CLICK.
BM_GETCHECK (see page 16313)	This is constant Messages.BM_GETCHECK.
BM_GETIMAGE (see page 16313)	This is constant Messages.BM_GETIMAGE.
BM_GETSTATE (see page 16313)	This is constant Messages.BM_GETSTATE.
BM_SETCHECK (see page 16314)	This is constant Messages.BM_SETCHECK.
BM_SETIMAGE (see page 16314)	This is constant Messages.BM_SETIMAGE.
BM_SETSTATE (see page 16314)	This is constant Messages.BM_SETSTATE.
BM_SETSTYLE (see page 16315)	This is constant Messages.BM_SETSTYLE.
BN_CLICKED (see page 16315)	This is constant Messages.BN_CLICKED.
BN_DBLCLK (see page 16315)	This is constant Messages.BN_DBLCLK.
BN_DISABLE (see page 16316)	This is constant Messages.BN_DISABLE.
BN_DOUBLECLICKED (see page 16316)	This is constant Messages.BN_DOUBLECLICKED.
BN_HILITE (see page 16316)	This is constant Messages.BN_HILITE.
BN_KILLFOCUS (see page 16317)	This is constant Messages.BN_KILLFOCUS.
BN_PAINT (see page 16317)	This is constant Messages.BN_PAINT.
BN_PUSHED (see page 16317)	This is constant Messages.BN_PUSHED.
BN_SETFOCUS (see page 16318)	This is constant Messages.BN_SETFOCUS.
BN_UNHILITE (see page 16318)	This is constant Messages.BN_UNHILITE.
BN_UNPUSHED (see page 16318)	This is constant Messages.BN_UNPUSHED.
CB_ADDSTRING (see page 16319)	This is constant Messages.CB_ADDSTRING.
CB_DELETESTRING (see page 16319)	This is constant Messages.CB_DELETESTRING.
CB_DIR (see page 16319)	This is constant Messages.CB_DIR.
CB_FINDSTRING (see page 16320)	This is constant Messages.CB_FINDSTRING.
CB_FINDSTRINGEXACT (see page 16320)	This is constant Messages.CB_FINDSTRINGEXACT.
CB_GETCOUNT (see page 16320)	This is constant Messages.CB_GETCOUNT.
CB_GETCURSEL (see page 16321)	This is constant Messages.CB_GETCURSEL.
CB_GETDROPPEDCONTROLRECT (see page 16321)	This is constant Messages.CB_GETDROPPEDCONTROLRECT.
CB_GETDROPPEDSTATE (see page 16321)	This is constant Messages.CB_GETDROPPEDSTATE.
CB_GETDROPPEDWIDTH (see page 16322)	This is constant Messages.CB_GETDROPPEDWIDTH.
CB_GETEDITSEL (see page 16322)	This is constant Messages.CB_GETEDITSEL.
CB_GETEXTENDEDUI (see page 16322)	This is constant Messages.CB_GETEXTENDEDUI.
CB_GETHORIZONTALEXTENT (see page 16323)	This is constant Messages.CB_GETHORIZONTALEXTENT.
CB_GETITEMDATA (see page 16323)	This is constant Messages.CB_GETITEMDATA.
CB_GETITEMHEIGHT (see page 16323)	This is constant Messages.CB_GETITEMHEIGHT.
CB_GETLBTEXT (see page 16324)	This is constant Messages.CB_GETLBTEXT.
CB_GETLBTEXTLEN (see page 16324)	This is constant Messages.CB_GETLBTEXTLEN.
CB_GETLOCALE (see page 16324)	This is constant Messages.CB_GETLOCALE.
CB_GETTOPINDEX (see page 16325)	This is constant Messages.CB_GETTOPINDEX.
CB_INITSTORAGE (see page 16325)	This is constant Messages.CB_INITSTORAGE.
CB_INSERTSTRING (see page 16325)	This is constant Messages.CB_INSERTSTRING.
CB_LIMITTEXT (see page 16326)	This is constant Messages.CB_LIMITTEXT.
CB_MSGMAX (see page 16326)	This is constant Messages.CB_MSGMAX.
CB_RESETCONTENT (see page 16326)	This is constant Messages.CB_RESETCONTENT.
CB_SELECTSTRING (see page 16327)	This is constant Messages.CB_SELECTSTRING.

CB_SETCURSEL (see page 16327)	This is constant Messages.CB_SETCURSEL.
CB_SETDROPPEDWIDTH (see page 16327)	This is constant Messages.CB_SETDROPPEDWIDTH.
CB_SETEDITSEL (see page 16328)	This is constant Messages.CB_SETEDITSEL.
CB_SETEXTENDEDUI (see page 16328)	This is constant Messages.CB_SETEXTENDEDUI.
CB_SETHORIZONTALEXTENT (see page 16328)	This is constant Messages.CB_SETHORIZONTALEXTENT.
CB_SETITEMDATA (see page 16329)	This is constant Messages.CB_SETITEMDATA.
CB_SETITEMHEIGHT (see page 16329)	This is constant Messages.CB_SETITEMHEIGHT.
CB_SETLOCALE (see page 16329)	This is constant Messages.CB_SETLOCALE.
CB_SETTOPINDEX (see page 16330)	This is constant Messages.CB_SETTOPINDEX.
CB_SHOWDROPDOWN (see page 16330)	This is constant Messages.CB_SHOWDROPDOWN.
CBN_CLOSEUP (see page 16330)	This is constant Messages.CBN_CLOSEUP.
CBN_DBLCLK (see page 16331)	This is constant Messages.CBN_DBLCLK.
CBN_DROPDOWN (see page 16331)	This is constant Messages.CBN_DROPDOWN.
CBN_EDITCHANGE (see page 16331)	This is constant Messages.CBN_EDITCHANGE.
CBN_EDITUPDATE (see page 16332)	This is constant Messages.CBN_EDITUPDATE.
CBN_ERRSPACE (see page 16332)	This is constant Messages.CBN_ERRSPACE.
CBN_KILLFOCUS (see page 16332)	This is constant Messages.CBN_KILLFOCUS.
CBN_SELCHANGE (see page 16333)	This is constant Messages.CBN_SELCHANGE.
CBN_SELENDCANCEL (see page 16333)	This is constant Messages.CBN_SELENDCANCEL.
CBN_SELENDOK (see page 16333)	This is constant Messages.CBN_SELENDOK.
CBN_SETFOCUS (see page 16334)	This is constant Messages.CBN_SETFOCUS.
DM_GETDEFID (see page 16334)	This is constant Messages.DM_GETDEFID.
DM_REPOSITION (see page 16334)	This is constant Messages.DM_REPOSITION.
DM_SETDEFID (see page 16335)	This is constant Messages.DM_SETDEFID.
EM_CANUNDO (see page 16335)	This is constant Messages.EM_CANUNDO.
EM_CHARFROMPOS (see page 16335)	This is constant Messages.EM_CHARFROMPOS.
EM_EMPTYUNDOBUFFER (see page 16336)	This is constant Messages.EM_EMPTYUNDOBUFFER.
EM_FMTLINES (see page 16336)	This is constant Messages.EM_FMTLINES.
EM_GETFIRSTVISIBLELINE (see page 16336)	This is constant Messages.EM_GETFIRSTVISIBLELINE.
EM_GETHANDLE (see page 16337)	This is constant Messages.EM_GETHANDLE.
EM_GETIMESTATUS (see page 16337)	This is constant Messages.EM_GETIMESTATUS.
EM_GETLIMITTEXT (see page 16337)	This is constant Messages.EM_GETLIMITTEXT.
EM_GETLINE (see page 16338)	This is constant Messages.EM_GETLINE.
EM_GETLINECOUNT (see page 16338)	This is constant Messages.EM_GETLINECOUNT.
EM_GETMARGINS (see page 16338)	This is constant Messages.EM_GETMARGINS.
EM_GETMODIFY (see page 16339)	This is constant Messages.EM_GETMODIFY.
EM_GETPASSWORDCHAR (see page 16339)	This is constant Messages.EM_GETPASSWORDCHAR.
EM_GETRECT (see page 16339)	This is constant Messages.EM_GETRECT.
EM_GETSEL (see page 16340)	This is constant Messages.EM_GETSEL.
EM_GETTHUMB (see page 16340)	This is constant Messages.EM_GETTHUMB.
EM_GETWORDBREAKPROC (see page 16340)	This is constant Messages.EM_GETWORDBREAKPROC.
EM_LIMITTEXT (see page 16341)	This is constant Messages.EM_LIMITTEXT.
EM_LINEFROMCHAR (see page 16341)	This is constant Messages.EM_LINEFROMCHAR.
EM_LINEINDEX (see page 16341)	This is constant Messages.EM_LINEINDEX.
EM_LINELENGTH (see page 16342)	This is constant Messages.EM_LINELENGTH.
EM_LINESCROLL (see page 16342)	This is constant Messages.EM_LINESCROLL.
EM_POSFROMCHAR (see page 16342)	This is constant Messages.EM_POSFROMCHAR.
EM_REPLACESEL (see page 16343)	This is constant Messages.EM_REPLACESEL.
EM_SCROLL (see page 16343)	This is constant Messages.EM_SCROLL.
EM_SCROLLCARET (see page 16343)	This is constant Messages.EM_SCROLLCARET.
EM_SETHANDLE (see page 16344)	This is constant Messages.EM_SETHANDLE.
EM_SETIMESTATUS (see page 16344)	This is constant Messages.EM_SETIMESTATUS.
EM_SETLIMITTEXT (see page 16344)	This is constant Messages.EM_SETLIMITTEXT.
EM_SETMARGINS (see page 16345)	This is constant Messages.EM_SETMARGINS.

EM_SETMODIFY (see page 16345)	This is constant Messages.EM_SETMODIFY.
EM_SETPASSWORDCHAR (see page 16345)	This is constant Messages.EM_SETPASSWORDCHAR.
EM_SETREADONLY (see page 16346)	This is constant Messages.EM_SETREADONLY.
EM_SETRECT (see page 16346)	This is constant Messages.EM_SETRECT.
EM_SETRECTNP (see page 16346)	This is constant Messages.EM_SETRECTNP.
EM_SETSEL (see page 16347)	This is constant Messages.EM_SETSEL.
EM_SETTABSTOPS (see page 16347)	This is constant Messages.EM_SETTABSTOPS.
EM_SETWORDBREAKPROC (see page 16347)	This is constant Messages.EM_SETWORDBREAKPROC.
EM_UNDO (see page 16348)	This is constant Messages.EM_UNDO.
EN_CHANGE (see page 16348)	This is constant Messages.EN_CHANGE.
EN_ERRSPACE (see page 16348)	This is constant Messages.EN_ERRSPACE.
EN_HSCROLL (see page 16349)	This is constant Messages.EN_HSCROLL.
EN_KILLFOCUS (see page 16349)	This is constant Messages.EN_KILLFOCUS.
EN_MAXTEXT (see page 16349)	This is constant Messages.EN_MAXTEXT.
EN_SETFOCUS (see page 16350)	This is constant Messages.EN_SETFOCUS.
EN_UPDATE (see page 16350)	This is constant Messages.EN_UPDATE.
EN_VSCROLL (see page 16350)	This is constant Messages.EN_VSCROLL.
LB_ADDFILE (see page 16351)	This is constant Messages.LB_ADDFILE.
LB_ADDSTRING (see page 16351)	This is constant Messages.LB_ADDSTRING.
LB_DELETESTRING (see page 16351)	This is constant Messages.LB_DELETESTRING.
LB_DIR (see page 16352)	This is constant Messages.LB_DIR.
LB_FINDSTRING (see page 16352)	This is constant Messages.LB_FINDSTRING.
LB_FINDSTRINGEXACT (see page 16352)	This is constant Messages.LB_FINDSTRINGEXACT.
LB_GETANCHORINDEX (see page 16353)	This is constant Messages.LB_GETANCHORINDEX.
LB_GETCARETINDEX (see page 16353)	This is constant Messages.LB_GETCARETINDEX.
LB_GETCOUNT (see page 16353)	This is constant Messages.LB_GETCOUNT.
LB_GETCURSEL (see page 16354)	This is constant Messages.LB_GETCURSEL.
LB_GETHORIZONTALEXTENT (see page 16354)	This is constant Messages.LB_GETHORIZONTALEXTENT.
LB_GETITEMDATA (see page 16354)	This is constant Messages.LB_GETITEMDATA.
LB_GETITEMHEIGHT (see page 16355)	This is constant Messages.LB_GETITEMHEIGHT.
LB_GETITEMRECT (see page 16355)	This is constant Messages.LB_GETITEMRECT.
LB_GETLOCALE (see page 16355)	This is constant Messages.LB_GETLOCALE.
LB_GETSEL (see page 16356)	This is constant Messages.LB_GETSEL.
LB_GETSELCOUNT (see page 16356)	This is constant Messages.LB_GETSELCOUNT.
LB_GETSELITEMS (see page 16356)	This is constant Messages.LB_GETSELITEMS.
LB_GETTEXT (see page 16357)	This is constant Messages.LB_GETTEXT.
LB_GETTEXTLEN (see page 16357)	This is constant Messages.LB_GETTEXTLEN.
LB_GETTOPINDEX (see page 16357)	This is constant Messages.LB_GETTOPINDEX.
LB_INITSTORAGE (see page 16358)	This is constant Messages.LB_INITSTORAGE.
LB_INSERTSTRING (see page 16358)	This is constant Messages.LB_INSERTSTRING.
LB_ITEMFROMPOINT (see page 16358)	This is constant Messages.LB_ITEMFROMPOINT.
LB_MSGMAX (see page 16359)	This is constant Messages.LB_MSGMAX.
LB_RESETCONTENT (see page 16359)	This is constant Messages.LB_RESETCONTENT.
LB_SELECTSTRING (see page 16359)	This is constant Messages.LB_SELECTSTRING.
LB_SELITEMRANGE (see page 16360)	This is constant Messages.LB_SELITEMRANGE.
LB_SELITEMRANGEEX (see page 16360)	This is constant Messages.LB_SELITEMRANGEEX.
LB_SETANCHORINDEX (see page 16360)	This is constant Messages.LB_SETANCHORINDEX.
LB_SETCARETINDEX (see page 16361)	This is constant Messages.LB_SETCARETINDEX.
LB_SETCOLUMNWIDTH (see page 16361)	This is constant Messages.LB_SETCOLUMNWIDTH.
LB_SETCOUNT (see page 16361)	This is constant Messages.LB_SETCOUNT.
LB_SETCURSEL (see page 16362)	This is constant Messages.LB_SETCURSEL.
LB_SETHORIZONTALEXTENT (see page 16362)	This is constant Messages.LB_SETHORIZONTALEXTENT.
LB_SETITEMDATA (see page 16362)	This is constant Messages.LB_SETITEMDATA.
LB_SETITEMHEIGHT (see page 16363)	This is constant Messages.LB_SETITEMHEIGHT.

LB_SETLOCALE (see page 16363)	This is constant Messages.LB_SETLOCALE.
LB_SETSEL (see page 16363)	This is constant Messages.LB_SETSEL.
LB_SETTABSTOPS (see page 16364)	This is constant Messages.LB_SETTABSTOPS.
LB_SETTOPINDEX (see page 16364)	This is constant Messages.LB_SETTOPINDEX.
LBN_DBLCLK (see page 16364)	This is constant Messages.LBN_DBLCLK.
LBN_ERRSPACE (see page 16365)	This is constant Messages.LBN_ERRSPACE.
LBN_KILLFOCUS (see page 16365)	This is constant Messages.LBN_KILLFOCUS.
LBN_SELCANCEL (see page 16365)	This is constant Messages.LBN_SELCANCEL.
LBN_SELCHANGE (see page 16366)	This is constant Messages.LBN_SELCHANGE.
LBN_SETFOCUS (see page 16366)	This is constant Messages.LBN_SETFOCUS.
PSM_PAGEINFO (see page 16366)	This is constant Messages.PSM_PAGEINFO.
PSM_SHEETINFO (see page 16367)	This is constant Messages.PSM_SHEETINFO.
SBM_ENABLE_ARROWS (see page 16367)	This is constant Messages.SBM_ENABLE_ARROWS.
SBM_GETPOS (see page 16367)	This is constant Messages.SBM_GETPOS.
SBM_GETRANGE (see page 16368)	This is constant Messages.SBM_GETRANGE.
SBM_GETSCROLLINFO (see page 16368)	This is constant Messages.SBM_GETSCROLLINFO.
SBM_SETPOS (see page 16368)	This is constant Messages.SBM_SETPOS.
SBM_SETRANGE (see page 16369)	This is constant Messages.SBM_SETRANGE.
SBM_SETRANGEREDRAW (see page 16369)	This is constant Messages.SBM_SETRANGEREDRAW.
SBM_SETSCROLLINFO (see page 16369)	This is constant Messages.SBM_SETSCROLLINFO.
WM_ACTIVATE (see page 16370)	This is constant Messages.WM_ACTIVATE.
WM_ACTIVATEAPP (see page 16370)	This is constant Messages.WM_ACTIVATEAPP.
WM_APP (see page 16370)	This is constant Messages.WM_APP.
WM_APPCOMMAND (see page 16371)	This is constant Messages.WM_APPCOMMAND.
WM_ASKCBFORMATNAME (see page 16371)	This is constant Messages.WM_ASKCBFORMATNAME.
WM_CANCELJOURNAL (see page 16371)	This is constant Messages.WM_CANCELJOURNAL.
WM_CANCELMODE (see page 16372)	This is constant Messages.WM_CANCELMODE.
WM_CAPTURECHANGED (see page 16372)	This is constant Messages.WM_CAPTURECHANGED.
WM_CHANGECBCHAIN (see page 16372)	This is constant Messages.WM_CHANGECBCHAIN.
WM_CHANGEUISTATE (see page 16373)	This is constant Messages.WM_CHANGEUISTATE.
WM_CHAR (see page 16373)	This is constant Messages.WM_CHAR.
WM_CHARTOITEM (see page 16373)	This is constant Messages.WM_CHARTOITEM.
WM_CHILDACTIVATE (see page 16374)	This is constant Messages.WM_CHILDACTIVATE.
WM_CLEAR (see page 16374)	This is constant Messages.WM_CLEAR.
WM_CLOSE (see page 16374)	This is constant Messages.WM_CLOSE.
WM_COALESCE_FIRST (see page 16375)	This is constant Messages.WM_COALESCE_FIRST.
WM_COALESCE_LAST (see page 16375)	This is constant Messages.WM_COALESCE_LAST.
WM_COMMAND (see page 16375)	This is constant Messages.WM_COMMAND.
WM_COMMNOTIFY (see page 16376)	This is constant Messages.WM_COMMNOTIFY.
WM_COMPACTING (see page 16376)	This is constant Messages.WM_COMPACTING.
WM_COMPAREITEM (see page 16376)	This is constant Messages.WM_COMPAREITEM.
WM_CONTEXTMENU (see page 16377)	This is constant Messages.WM_CONTEXTMENU.
WM_COPY (see page 16377)	This is constant Messages.WM_COPY.
WM_COPYDATA (see page 16377)	This is constant Messages.WM_COPYDATA.
WM_CREATE (see page 16378)	This is constant Messages.WM_CREATE.
WM_CTLCOLOR (see page 16378)	This is constant Messages.WM_CTLCOLOR.
WM_CTLCOLORBTN (see page 16378)	This is constant Messages.WM_CTLCOLORBTN.
WM_CTLCOLORDLG (see page 16379)	This is constant Messages.WM_CTLCOLORDLG.
WM_CTLCOLOREDIT (see page 16379)	This is constant Messages.WM_CTLCOLOREDIT.
WM_CTLCOLORLISTBOX (see page 16379)	This is constant Messages.WM_CTLCOLORLISTBOX.
WM_CTLCOLORMSGBOX (see page 16380)	This is constant Messages.WM_CTLCOLORMSGBOX.
WM_CTLCOLORSCROLLBAR (see page 16380)	This is constant Messages.WM_CTLCOLORSCROLLBAR.
WM_CTLCOLORSTATIC (see page 16380)	This is constant Messages.WM_CTLCOLORSTATIC.
WM_CUT (see page 16381)	This is constant Messages.WM_CUT.

WM_DDE_ACK (see page 16381)	This is constant Messages.WM_DDE_ACK.
WM_DDE_ADVISE (see page 16381)	This is constant Messages.WM_DDE_ADVISE.
WM_DDE_DATA (see page 16382)	This is constant Messages.WM_DDE_DATA.
WM_DDE_EXECUTE (see page 16382)	This is constant Messages.WM_DDE_EXECUTE.
WM_DDE_FIRST (see page 16382)	This is constant Messages.WM_DDE_FIRST.
WM_DDE_INITIATE (see page 16383)	This is constant Messages.WM_DDE_INITIATE.
WM_DDE_LAST (see page 16383)	This is constant Messages.WM_DDE_LAST.
WM_DDE_POKE (see page 16383)	This is constant Messages.WM_DDE_POKE.
WM_DDE_REQUEST (see page 16384)	This is constant Messages.WM_DDE_REQUEST.
WM_DDE_TERMINATE (see page 16384)	This is constant Messages.WM_DDE_TERMINATE.
WM_DDE_UNADVISE (see page 16384)	This is constant Messages.WM_DDE_UNADVISE.
WM_DEADCHAR (see page 16385)	This is constant Messages.WM_DEADCHAR.
WM_DELETEITEM (see page 16385)	This is constant Messages.WM_DELETEITEM.
WM_DESTROY (see page 16385)	This is constant Messages.WM_DESTROY.
WM_DESTROYCLIPBOARD (see page 16386)	This is constant Messages.WM_DESTROYCLIPBOARD.
WM_DEVICECHANGE (see page 16386)	This is constant Messages.WM_DEVICECHANGE.
WM_DEVMODECHANGE (see page 16386)	This is constant Messages.WM_DEVMODECHANGE.
WM_DISPLAYCHANGE (see page 16387)	This is constant Messages.WM_DISPLAYCHANGE.
WM_DRAWCLIPBOARD (see page 16387)	This is constant Messages.WM_DRAWCLIPBOARD.
WM_DRAWITEM (see page 16387)	This is constant Messages.WM_DRAWITEM.
WM_DROPFILES (see page 16388)	This is constant Messages.WM_DROPFILES.
WM_DWMCOLORIZATIONCOLORCHANGED (see page 16388)	This is constant Messages.WM_DWMCOLORIZATIONCOLORCHANGED.
WM_DWMCOMPOSITIONCHANGED (see page 16388)	This is constant Messages.WM_DWMCOMPOSITIONCHANGED.
WM_DWMNCRENDERINGCHANGED (see page 16389)	This is constant Messages.WM_DWMNCRENDERINGCHANGED.
WM_DWMWINDOWMAXIMIZEDCHANGE (see page 16389)	This is constant Messages.WM_DWMWINDOWMAXIMIZEDCHANGE.
WM_ENABLE (see page 16389)	This is constant Messages.WM_ENABLE.
WM_ENDSESSION (see page 16390)	This is constant Messages.WM_ENDSESSION.
WM_ENTERIDLE (see page 16390)	This is constant Messages.WM_ENTERIDLE.
WM_ENTERMENULOOP (see page 16390)	This is constant Messages.WM_ENTERMENULOOP.
WM_ENTERSIZEMOVE (see page 16391)	This is constant Messages.WM_ENTERSIZEMOVE.
WM_ERASEBKGND (see page 16391)	This is constant Messages.WM_ERASEBKGND.
WM_EXITMENULOOP (see page 16391)	This is constant Messages.WM_EXITMENULOOP.
WM_EXITSIZEMOVE (see page 16392)	This is constant Messages.WM_EXITSIZEMOVE.
WM_FONTCHANGE (see page 16392)	This is constant Messages.WM_FONTCHANGE.
WM_GETDLGCODE (see page 16392)	This is constant Messages.WM_GETDLGCODE.
WM_GETFONT (see page 16393)	This is constant Messages.WM_GETFONT.
WM_GETHOTKEY (see page 16393)	This is constant Messages.WM_GETHOTKEY.
WM_GETICON (see page 16393)	This is constant Messages.WM_GETICON.
WM_GETMINMAXINFO (see page 16394)	This is constant Messages.WM_GETMINMAXINFO.
WM_GETOBJECT (see page 16394)	This is constant Messages.WM_GETOBJECT.
WM_GETTEXT (see page 16394)	This is constant Messages.WM_GETTEXT.
WM_GETTEXTLENGTH (see page 16395)	This is constant Messages.WM_GETTEXTLENGTH.
WM_HANDHELDFIRST (see page 16395)	This is constant Messages.WM_HANDHELDFIRST.
WM_HANDHELDLAST (see page 16395)	This is constant Messages.WM_HANDHELDLAST.
WM_HELP (see page 16396)	This is constant Messages.WM_HELP.
WM_HOTKEY (see page 16396)	This is constant Messages.WM_HOTKEY.
WM_HSCROLL (see page 16396)	This is constant Messages.WM_HSCROLL.
WM_HSCROLLCLIPBOARD (see page 16397)	This is constant Messages.WM_HSCROLLCLIPBOARD.
WM_ICONERASEBKGND (see page 16397)	This is constant Messages.WM_ICONERASEBKGND.
WM_IME_CHAR (see page 16397)	This is constant Messages.WM_IME_CHAR.
WM_IME_COMPOSITION (see page 16398)	This is constant Messages.WM_IME_COMPOSITION.
WM_IME_COMPOSITIONFULL (see page 16398)	This is constant Messages.WM_IME_COMPOSITIONFULL.
WM_IME_CONTROL (see page 16398)	This is constant Messages.WM_IME_CONTROL.
WM_IME_ENDCOMPOSITION (see page 16399)	This is constant Messages.WM_IME_ENDCOMPOSITION.

WM_IME_KEYDOWN (see page 16399)	This is constant Messages.WM_IME_KEYDOWN.
WM_IME_KEYLAST (see page 16399)	This is constant Messages.WM_IME_KEYLAST.
WM_IME_KEYUP (see page 16400)	This is constant Messages.WM_IME_KEYUP.
WM_IME_NOTIFY (see page 16400)	This is constant Messages.WM_IME_NOTIFY.
WM_IME_REQUEST (see page 16400)	This is constant Messages.WM_IME_REQUEST.
WM_IME_SELECT (see page 16401)	This is constant Messages.WM_IME_SELECT.
WM_IME_SETCONTEXT (see page 16401)	This is constant Messages.WM_IME_SETCONTEXT.
WM_IME_STARTCOMPOSITION (see page 16401)	This is constant Messages.WM_IME_STARTCOMPOSITION.
WM_INITDIALOG (see page 16402)	This is constant Messages.WM_INITDIALOG.
WM_INITMENU (see page 16402)	This is constant Messages.WM_INITMENU.
WM_INITMENUPOPUP (see page 16402)	This is constant Messages.WM_INITMENUPOPUP.
WM_INPUT (see page 16403)	This is constant Messages.WM_INPUT.
WM_INPUTLANGCHANGE (see page 16403)	This is constant Messages.WM_INPUTLANGCHANGE.
WM_INPUTLANGCHANGEREQUEST (see page 16403)	This is constant Messages.WM_INPUTLANGCHANGEREQUEST.
WM_KEYDOWN (see page 16404)	This is constant Messages.WM_KEYDOWN.
WM_KEYFIRST (see page 16404)	This is constant Messages.WM_KEYFIRST.
WM_KEYLAST (see page 16404)	This is constant Messages.WM_KEYLAST.
WM_KEYUP (see page 16405)	This is constant Messages.WM_KEYUP.
WM_KILLFOCUS (see page 16405)	This is constant Messages.WM_KILLFOCUS.
WM_LBUTTONDBLCLK (see page 16405)	This is constant Messages.WM_LBUTTONDBLCLK.
WM_LBUTTONDOWN (see page 16406)	This is constant Messages.WM_LBUTTONDOWN.
WM_LBUTTONUP (see page 16406)	This is constant Messages.WM_LBUTTONUP.
WM_MBUTTONDBLCLK (see page 16406)	This is constant Messages.WM_MBUTTONDBLCLK.
WM_MBUTTONDOWN (see page 16407)	This is constant Messages.WM_MBUTTONDOWN.
WM_MBUTTONUP (see page 16407)	This is constant Messages.WM_MBUTTONUP.
WM_MDIACTIVATE (see page 16407)	This is constant Messages.WM_MDIACTIVATE.
WM_MDICASCADE (see page 16408)	This is constant Messages.WM_MDICASCADE.
WM_MDICREATE (see page 16408)	This is constant Messages.WM_MDICREATE.
WM_MDIDESTROY (see page 16408)	This is constant Messages.WM_MDIDESTROY.
WM_MDIGETACTIVE (see page 16409)	This is constant Messages.WM_MDIGETACTIVE.
WM_MDIICONARRANGE (see page 16409)	This is constant Messages.WM_MDIICONARRANGE.
WM_MDIMAXIMIZE (see page 16409)	This is constant Messages.WM_MDIMAXIMIZE.
WM_MDINEXT (see page 16410)	This is constant Messages.WM_MDINEXT.
WM_MDIREFRESHMENU (see page 16410)	This is constant Messages.WM_MDIREFRESHMENU.
WM_MDIRESTORE (see page 16410)	This is constant Messages.WM_MDIRESTORE.
WM_MDISETMENU (see page 16411)	This is constant Messages.WM_MDISETMENU.
WM_MDITILE (see page 16411)	This is constant Messages.WM_MDITILE.
WM_MEASUREITEM (see page 16411)	This is constant Messages.WM_MEASUREITEM.
WM_MENUCHAR (see page 16412)	This is constant Messages.WM_MENUCHAR.
WM_MENUCOMMAND (see page 16412)	This is constant Messages.WM_MENUCOMMAND.
WM_MENUDRAG (see page 16412)	This is constant Messages.WM_MENUDRAG.
WM_MENUGETOBJECT (see page 16413)	This is constant Messages.WM_MENUGETOBJECT.
WM_MENURBUTTONUP (see page 16413)	This is constant Messages.WM_MENURBUTTONUP.
WM_MENUSELECT (see page 16413)	This is constant Messages.WM_MENUSELECT.
WM_MOUSEACTIVATE (see page 16414)	This is constant Messages.WM_MOUSEACTIVATE.
WM_MOUSEFIRST (see page 16414)	This is constant Messages.WM_MOUSEFIRST.
WM_MOUSEHOVER (see page 16414)	This is constant Messages.WM_MOUSEHOVER.
WM_MOUSELAST (see page 16415)	This is constant Messages.WM_MOUSELAST.
WM_MOUSELEAVE (see page 16415)	This is constant Messages.WM_MOUSELEAVE.
WM_MOUSEMOVE (see page 16415)	This is constant Messages.WM_MOUSEMOVE.
WM_MOUSEWHEEL (see page 16416)	This is constant Messages.WM_MOUSEWHEEL.
WM_MOVE (see page 16416)	This is constant Messages.WM_MOVE.
WM_MOVING (see page 16416)	This is constant Messages.WM_MOVING.
WM_NCACTIVATE (see page 16417)	This is constant Messages.WM_NCACTIVATE.

WM_NCCALCSIZE (see page 16417)	This is constant Messages.WM_NCCALCSIZE.
WM_NCCREATE (see page 16417)	This is constant Messages.WM_NCCREATE.
WM_NCDESTROY (see page 16418)	This is constant Messages.WM_NCDESTROY.
WM_NCHITTEST (see page 16418)	This is constant Messages.WM_NCHITTEST.
WM_NCLBUTTONDBLCLK (see page 16418)	This is constant Messages.WM_NCLBUTTONDBLCLK.
WM_NCLBUTTONDOWN (see page 16419)	This is constant Messages.WM_NCLBUTTONDOWN.
WM_NCLBUTTONUP (see page 16419)	This is constant Messages.WM_NCLBUTTONUP.
WM_NCMBUTTONDBLCLK (see page 16419)	This is constant Messages.WM_NCMBUTTONDBLCLK.
WM_NCMBUTTONDOWN (see page 16420)	This is constant Messages.WM_NCMBUTTONDOWN.
WM_NCMBUTTONUP (see page 16420)	This is constant Messages.WM_NCMBUTTONUP.
WM_NCMOUSEHOVER (see page 16420)	This is constant Messages.WM_NCMOUSEHOVER.
WM_NCMOUSELEAVE (see page 16421)	This is constant Messages.WM_NCMOUSELEAVE.
WM_NCMOUSEMOVE (see page 16421)	This is constant Messages.WM_NCMOUSEMOVE.
WM_NCPAINT (see page 16421)	This is constant Messages.WM_NCPAINT.
WM_NCRBUTTONDBLCLK (see page 16422)	This is constant Messages.WM_NCRBUTTONDBLCLK.
WM_NCRBUTTONDOWN (see page 16422)	This is constant Messages.WM_NCRBUTTONDOWN.
WM_NCRBUTTONUP (see page 16422)	This is constant Messages.WM_NCRBUTTONUP.
WM_NCXBUTTONDBLCLK (see page 16423)	This is constant Messages.WM_NCXBUTTONDBLCLK.
WM_NCXBUTTONDOWN (see page 16423)	This is constant Messages.WM_NCXBUTTONDOWN.
WM_NCXBUTTONUP (see page 16423)	This is constant Messages.WM_NCXBUTTONUP.
WM_NEXTDLGCTL (see page 16424)	This is constant Messages.WM_NEXTDLGCTL.
WM_NEXTMENU (see page 16424)	This is constant Messages.WM_NEXTMENU.
WM_NOTIFY (see page 16424)	This is constant Messages.WM_NOTIFY.
WM_NOTIFYFORMAT (see page 16425)	This is constant Messages.WM_NOTIFYFORMAT.
WM_NULL (see page 16425)	This is constant Messages.WM_NULL.
WM_PAINT (see page 16425)	This is constant Messages.WM_PAINT.
WM_PAINTCLIPBOARD (see page 16426)	This is constant Messages.WM_PAINTCLIPBOARD.
WM_PAINTICON (see page 16426)	This is constant Messages.WM_PAINTICON.
WM_PALETTECHANGED (see page 16426)	This is constant Messages.WM_PALETTECHANGED.
WM_PALETTEISCHANGING (see page 16427)	This is constant Messages.WM_PALETTEISCHANGING.
WM_PARENTNOTIFY (see page 16427)	This is constant Messages.WM_PARENTNOTIFY.
WM_PASTE (see page 16427)	This is constant Messages.WM_PASTE.
WM_PENWINFIRST (see page 16428)	This is constant Messages.WM_PENWINFIRST.
WM_PENWINLAST (see page 16428)	This is constant Messages.WM_PENWINLAST.
WM_POWER (see page 16428)	This is constant Messages.WM_POWER.
WM_POWERBROADCAST (see page 16429)	This is constant Messages.WM_POWERBROADCAST.
WM_PRINT (see page 16429)	This is constant Messages.WM_PRINT.
WM_PRINTCLIENT (see page 16429)	This is constant Messages.WM_PRINTCLIENT.
WM_QUERYDRAGICON (see page 16430)	This is constant Messages.WM_QUERYDRAGICON.
WM_QUERYENDSESSION (see page 16430)	This is constant Messages.WM_QUERYENDSESSION.
WM_QUERYNEWPALETTE (see page 16430)	This is constant Messages.WM_QUERYNEWPALETTE.
WM_QUERYOPEN (see page 16431)	This is constant Messages.WM_QUERYOPEN.
WM_QUERYUISTATE (see page 16431)	This is constant Messages.WM_QUERYUISTATE.
WM_QUEUESYNC (see page 16431)	This is constant Messages.WM_QUEUESYNC.
WM_QUIT (see page 16432)	This is constant Messages.WM_QUIT.
WM_RBUTTONDBLCLK (see page 16432)	This is constant Messages.WM_RBUTTONDBLCLK.
WM_RBUTTONDOWN (see page 16432)	This is constant Messages.WM_RBUTTONDOWN.
WM_RBUTTONUP (see page 16433)	This is constant Messages.WM_RBUTTONUP.
WM_RENDERALLFORMATS (see page 16433)	This is constant Messages.WM_RENDERALLFORMATS.
WM_RENDERFORMAT (see page 16433)	This is constant Messages.WM_RENDERFORMAT.
WM_SETCURSOR (see page 16434)	This is constant Messages.WM_SETCURSOR.
WM_SETFOCUS (see page 16434)	This is constant Messages.WM_SETFOCUS.
WM_SETFONT (see page 16434)	This is constant Messages.WM_SETFONT.
WM_SETHOTKEY (see page 16435)	This is constant Messages.WM_SETHOTKEY.

WM_SETICON (see page 16435)	This is constant Messages.WM_SETICON.
WM_SETREDRAW (see page 16435)	This is constant Messages.WM_SETREDRAW.
WM_SETTEXT (see page 16436)	This is constant Messages.WM_SETTEXT.
WM_SETTINGCHANGE (see page 16436)	This is constant Messages.WM_SETTINGCHANGE.
WM_SHOWWINDOW (see page 16436)	This is constant Messages.WM_SHOWWINDOW.
WM_SIZE (see page 16437)	This is constant Messages.WM_SIZE.
WM_SIZECLIPBOARD (see page 16437)	This is constant Messages.WM_SIZECLIPBOARD.
WM_SIZING (see page 16437)	This is constant Messages.WM_SIZING.
WM_SPOOLERSTATUS (see page 16438)	This is constant Messages.WM_SPOOLERSTATUS.
WM_STYLECHANGED (see page 16438)	This is constant Messages.WM_STYLECHANGED.
WM_STYLECHANGING (see page 16438)	This is constant Messages.WM_STYLECHANGING.
WM_SYSCHAR (see page 16439)	This is constant Messages.WM_SYSCHAR.
WM_SYSCOLORCHANGE (see page 16439)	This is constant Messages.WM_SYSCOLORCHANGE.
WM_SYSCOMMAND (see page 16439)	This is constant Messages.WM_SYSCOMMAND.
WM_SYSDEADCHAR (see page 16440)	This is constant Messages.WM_SYSDEADCHAR.
WM_SYSKEYDOWN (see page 16440)	This is constant Messages.WM_SYSKEYDOWN.
WM_SYSKEYUP (see page 16440)	This is constant Messages.WM_SYSKEYUP.
WM_SYSTEMERROR (see page 16441)	This is constant Messages.WM_SYSTEMERROR.
WM_TABLET_FIRST (see page 16441)	This is constant Messages.WM_TABLET_FIRST.
WM_TABLET_LAST (see page 16441)	This is constant Messages.WM_TABLET_LAST.
WM_TCARD (see page 16442)	This is constant Messages.WM_TCARD.
WM_THEMECHANGED (see page 16442)	This is constant Messages.WM_THEMECHANGED.
WM_TIMECHANGE (see page 16442)	This is constant Messages.WM_TIMECHANGE.
WM_TIMER (see page 16443)	This is constant Messages.WM_TIMER.
WM_UNDO (see page 16443)	This is constant Messages.WM_UNDO.
WM_UNICHAR (see page 16443)	This is constant Messages.WM_UNICHAR.
WM_UNINITMENUPOPUP (see page 16444)	This is constant Messages.WM_UNINITMENUPOPUP.
WM_UPDATEUISTATE (see page 16444)	This is constant Messages.WM_UPDATEUISTATE.
WM_USER (see page 16444)	This is constant Messages.WM_USER.
WM_USERCHANGED (see page 16445)	This is constant Messages.WM_USERCHANGED.
WM_VKEYTOITEM (see page 16445)	This is constant Messages.WM_VKEYTOITEM.
WM_VSCROLL (see page 16445)	This is constant Messages.WM_VSCROLL.
WM_VSCROLLCLIPBOARD (see page 16446)	This is constant Messages.WM_VSCROLLCLIPBOARD.
WM_WINDOWPOSCHANGED (see page 16446)	This is constant Messages.WM_WINDOWPOSCHANGED.
WM_WINDOWPOSCHANGING (see page 16446)	This is constant Messages.WM_WINDOWPOSCHANGING.
WM_WININICHANGE (see page 16447)	This is constant Messages.WM_WININICHANGE.
WM_WTSSESSION_CHANGE (see page 16447)	This is constant Messages.WM_WTSSESSION_CHANGE.

96.3.1 Messages.BM_CLICK Constant

Pascal

```
BM_CLICK = $00F5;
```

C++

```
BM_CLICK = 0x00F5;
```

File

Messages

Description

This is constant Messages.BM_CLICK.

96.3.2 Messages.BM_GETCHECK Constant

Pascal

```
BM_GETCHECK = $00F0;
```

C++

```
BM_GETCHECK = 0x00F0;
```

File

Messages

Description

This is constant Messages.BM_GETCHECK.

96.3.3 Messages.BM_GETIMAGE Constant

Pascal

```
BM_GETIMAGE = $00F6;
```

C++

```
BM_GETIMAGE = 0x00F6;
```

File

Messages

Description

This is constant Messages.BM_GETIMAGE.

96.3.4 Messages.BM_GETSTATE Constant

Pascal

```
BM_GETSTATE = $00F2;
```

C++

```
BM_GETSTATE = 0x00F2;
```

File

Messages

Description

This is constant Messages.BM_GETSTATE.

96.3.5 Messages.BM_SETCHECK Constant

Pascal

```
BM_SETCHECK = $00F1;
```

C++

```
BM_SETCHECK = 0x00F1;
```

File

Messages

Description

This is constant Messages.BM_SETCHECK.

96.3.6 Messages.BM_SETIMAGE Constant

Pascal

```
BM_SETIMAGE = $00F7;
```

C++

```
BM_SETIMAGE = 0x00F7;
```

File

Messages

Description

This is constant Messages.BM_SETIMAGE.

96.3.7 Messages.BM_SETSTATE Constant

Pascal

```
BM_SETSTATE = $00F3;
```

C++

```
BM_SETSTATE = 0x00F3;
```

File

Messages

Description

This is constant Messages.BM_SETSTATE.

96.3.8 Messages.BM_SETSTYLE Constant

Pascal

```
BM_SETSTYLE = $00F4;
```

C++

```
BM_SETSTYLE = 0x00F4;
```

File

Messages

Description

This is constant Messages.BM_SETSTYLE.

96.3.9 Messages.BN_CLICKED Constant

Pascal

```
BN_CLICKED = 0;
```

C++

```
BN_CLICKED = 0;
```

File

Messages

Description

This is constant Messages.BN_CLICKED.

96.3.10 Messages.BN_DBLCLK Constant

Pascal

```
BN_DBLCLK = BN_DOUBLECLICKED;
```

C++

```
BN_DBLCLK = BN_DOUBLECLICKED;
```

File

Messages

Description

This is constant Messages.BN_DBLCLK.

96.3.11 Messages.BN_DISABLE Constant

Pascal

```
BN_DISABLE = 4;
```

C++

```
BN_DISABLE = 4;
```

File

Messages

Description

This is constant Messages.BN_DISABLE.

96.3.12 Messages.BN_DOUBLECLICKED Constant

Pascal

```
BN_DOUBLECLICKED = 5;
```

C++

```
BN_DOUBLECLICKED = 5;
```

File

Messages

Description

This is constant Messages.BN_DOUBLECLICKED.

96.3.13 Messages.BN_HILITE Constant

Pascal

```
BN_HILITE = 2;
```

C++

```
BN_HILITE = 2;
```

File

Messages

Description

This is constant Messages.BN_HILITE.

96.3.14 Messages.BN_KILLFOCUS Constant

Pascal

```
BN_KILLFOCUS = 7;
```

C++

```
BN_KILLFOCUS = 7;
```

File

Messages

Description

This is constant Messages.BN_KILLFOCUS.

96.3.15 Messages.BN_PAINT Constant

Pascal

```
BN_PAINT = 1;
```

C++

```
BN_PAINT = 1;
```

File

Messages

Description

This is constant Messages.BN_PAINT.

96.3.16 Messages.BN_PUSHED Constant

Pascal

```
BN_PUSHED = BN_HILITE;
```

C++

```
BN_PUSHED = BN_HILITE;
```

File

Messages

Description

This is constant Messages.BN_PUSHED.

96.3.17 Messages.BN_SETFOCUS Constant

Pascal

```
BN_SETFOCUS = 6;
```

C++

```
BN_SETFOCUS = 6;
```

File

Messages

Description

This is constant Messages.BN_SETFOCUS.

96.3.18 Messages.BN_UNHILITE Constant

Pascal

```
BN_UNHILITE = 3;
```

C++

```
BN_UNHILITE = 3;
```

File

Messages

Description

This is constant Messages.BN_UNHILITE.

96.3.19 Messages.BN_UNPUSHED Constant

Pascal

```
BN_UNPUSHED = BN_UNHILITE;
```

C++

```
BN_UNPUSHED = BN_UNHILITE;
```

File

Messages

Description

This is constant Messages.BN_UNPUSHED.

96.3.20 Messages.CB_ADDSTRING Constant

Pascal

```
CB_ADDSTRING = $0143;
```

C++

```
CB_ADDSTRING = 0x0143;
```

File

Messages

Description

This is constant Messages.CB_ADDSTRING.

96.3.21 Messages.CB_DELETESTRING Constant

Pascal

```
CB_DELETESTRING = $0144;
```

C++

```
CB_DELETESTRING = 0x0144;
```

File

Messages

Description

This is constant Messages.CB_DELETESTRING.

96.3.22 Messages.CB_DIR Constant

Pascal

```
CB_DIR = $0145;
```

C++

```
CB_DIR = 0x0145;
```

File

Messages

Description

This is constant Messages.CB_DIR.

96.3.23 Messages.CB_FINDSTRING Constant

Pascal

```
CB_FINDSTRING = $014C;
```

C++

```
CB_FINDSTRING = 0x014C;
```

File

Messages

Description

This is constant Messages.CB_FINDSTRING.

96.3.24 Messages.CB_FINDSTRINGEXACT Constant

Pascal

```
CB_FINDSTRINGEXACT = $0158;
```

C++

```
CB_FINDSTRINGEXACT = 0x0158;
```

File

Messages

Description

This is constant Messages.CB_FINDSTRINGEXACT.

96.3.25 Messages.CB_GETCOUNT Constant

Pascal

```
CB_GETCOUNT = $0146;
```

C++

```
CB_GETCOUNT = 0x0146;
```

File

Messages

Description

This is constant Messages.CB_GETCOUNT.

96.3.26 Messages.CB_GETCURSEL Constant

Pascal

```
CB_GETCURSEL = $0147;
```

C++

```
CB_GETCURSEL = 0x0147;
```

File

Messages

Description

This is constant Messages.CB_GETCURSEL.

96.3.27 Messages.CB_GETDROPPEDCONTROLRECT Constant

Pascal

```
CB_GETDROPPEDCONTROLRECT = $0152;
```

C++

```
CB_GETDROPPEDCONTROLRECT = 0x0152;
```

File

Messages

Description

This is constant Messages.CB_GETDROPPEDCONTROLRECT.

96.3.28 Messages.CB_GETDROPPEDSTATE Constant

Pascal

```
CB_GETDROPPEDSTATE = $0157;
```

C++

```
CB_GETDROPPEDSTATE = 0x0157;
```

File

Messages

Description

This is constant Messages.CB_GETDROPPEDSTATE.

96.3.29 Messages.CB_GETDROPPEDWIDTH Constant

Pascal

```
CB_GETDROPPEDWIDTH = 351;
```

C++

```
CB_GETDROPPEDWIDTH = 351;
```

File

Messages

Description

This is constant Messages.CB_GETDROPPEDWIDTH.

96.3.30 Messages.CB_GETEDITSEL Constant

Pascal

```
CB_GETEDITSEL = $0140;
```

C++

```
CB_GETEDITSEL = 0x0140;
```

File

Messages

Description

This is constant Messages.CB_GETEDITSEL.

96.3.31 Messages.CB_GETEXTENDEDUI Constant

Pascal

```
CB_GETEXTENDEDUI = $0156;
```

C++

```
CB_GETEXTENDEDUI = 0x0156;
```

File

Messages

Description

This is constant Messages.CB_GETEXTENDEDUI.

96.3.32 Messages.CB_GETHORIZONTALEXTENT Constant

Pascal

```
CB_GETHORIZONTALEXTENT = 349;
```

C++

```
CB_GETHORIZONTALEXTENT = 349;
```

File

Messages

Description

This is constant Messages.CB_GETHORIZONTALEXTENT.

96.3.33 Messages.CB_GETITEMDATA Constant

Pascal

```
CB_GETITEMDATA = $0150;
```

C++

```
CB_GETITEMDATA = 0x0150;
```

File

Messages

Description

This is constant Messages.CB_GETITEMDATA.

96.3.34 Messages.CB_GETITEMHEIGHT Constant

Pascal

```
CB_GETITEMHEIGHT = $0154;
```

C++

```
CB_GETITEMHEIGHT = 0x0154;
```

File

Messages

Description

This is constant Messages.CB_GETITEMHEIGHT.

96.3.35 Messages.CB_GETLBTEXT Constant

Pascal

```
CB_GETLBTEXT = $0148;
```

C++

```
CB_GETLBTEXT = 0x0148;
```

File

Messages

Description

This is constant Messages.CB_GETLBTEXT.

96.3.36 Messages.CB_GETLBTEXTLEN Constant

Pascal

```
CB_GETLBTEXTLEN = $0149;
```

C++

```
CB_GETLBTEXTLEN = 0x0149;
```

File

Messages

Description

This is constant Messages.CB_GETLBTEXTLEN.

96.3.37 Messages.CB_GETLOCALE Constant

Pascal

```
CB_GETLOCALE = 346;
```

C++

```
CB_GETLOCALE = 346;
```

File

Messages

Description

This is constant Messages.CB_GETLOCALE.

96.3.38 Messages.CB_GETTOPINDEX Constant

Pascal

```
CB_GETTOPINDEX = 347;
```

C++

```
CB_GETTOPINDEX = 347;
```

File

Messages

Description

This is constant Messages.CB_GETTOPINDEX.

96.3.39 Messages.CB_INITSTORAGE Constant

Pascal

```
CB_INITSTORAGE = 353;
```

C++

```
CB_INITSTORAGE = 353;
```

File

Messages

Description

This is constant Messages.CB_INITSTORAGE.

96.3.40 Messages.CB_INSERTSTRING Constant

Pascal

```
CB_INSERTSTRING = $014A;
```

C++

```
CB_INSERTSTRING = 0x014A;
```

File

Messages

Description

This is constant Messages.CB_INSERTSTRING.

96.3.41 Messages.CB_LIMITTEXT Constant

Pascal

```
CB_LIMITTEXT = $0141;
```

C++

```
CB_LIMITTEXT = 0x0141;
```

File

Messages

Description

This is constant Messages.CB_LIMITTEXT.

96.3.42 Messages.CB_MSGMAX Constant

Pascal

```
CB_MSGMAX = 354;
```

C++

```
CB_MSGMAX = 354;
```

File

Messages

Description

This is constant Messages.CB_MSGMAX.

96.3.43 Messages.CB_RESETCONTENT Constant

Pascal

```
CB_RESETCONTENT = $014B;
```

C++

```
CB_RESETCONTENT = 0x014B;
```

File

Messages

Description

This is constant Messages.CB_RESETCONTENT.

96.3.44 Messages.CB_SELECTSTRING Constant

Pascal

```
CB_SELECTSTRING = $014D;
```

C++

```
CB_SELECTSTRING = 0x014D;
```

File

Messages

Description

This is constant Messages.CB_SELECTSTRING.

96.3.45 Messages.CB_SETCURSEL Constant

Pascal

```
CB_SETCURSEL = $014E;
```

C++

```
CB_SETCURSEL = 0x014E;
```

File

Messages

Description

This is constant Messages.CB_SETCURSEL.

96.3.46 Messages.CB_SETDROPPEDWIDTH Constant

Pascal

```
CB_SETDROPPEDWIDTH = 352;
```

C++

```
CB_SETDROPPEDWIDTH = 352;
```

File

Messages

Description

This is constant Messages.CB_SETDROPPEDWIDTH.

96.3.47 Messages.CB_SETEDITSEL Constant

Pascal

```
CB_SETEDITSEL = $0142;
```

C++

```
CB_SETEDITSEL = 0x0142;
```

File

Messages

Description

This is constant Messages.CB_SETEDITSEL.

96.3.48 Messages.CB_SETEXTENDEDUI Constant

Pascal

```
CB_SETEXTENDEDUI = $0155;
```

C++

```
CB_SETEXTENDEDUI = 0x0155;
```

File

Messages

Description

This is constant Messages.CB_SETEXTENDEDUI.

96.3.49 Messages.CB_SETHORIZONTALEXTENT Constant

Pascal

```
CB_SETHORIZONTALEXTENT = 350;
```

C++

```
CB_SETHORIZONTALEXTENT = 350;
```

File

Messages

Description

This is constant Messages.CB_SETHORIZONTALEXTENT.

96.3.50 Messages.CB_SETITEMDATA Constant

Pascal

```
CB_SETITEMDATA = $0151;
```

C++

```
CB_SETITEMDATA = 0x0151;
```

File

Messages

Description

This is constant Messages.CB_SETITEMDATA.

96.3.51 Messages.CB_SETITEMHEIGHT Constant

Pascal

```
CB_SETITEMHEIGHT = $0153;
```

C++

```
CB_SETITEMHEIGHT = 0x0153;
```

File

Messages

Description

This is constant Messages.CB_SETITEMHEIGHT.

96.3.52 Messages.CB_SETLOCALE Constant

Pascal

```
CB_SETLOCALE = 345;
```

C++

```
CB_SETLOCALE = 345;
```

File

Messages

Description

This is constant Messages.CB_SETLOCALE.

96.3.53 Messages.CB_SETTOPINDEX Constant

Pascal

```
CB_SETTOPINDEX = 348;
```

C++

```
CB_SETTOPINDEX = 348;
```

File

Messages

Description

This is constant Messages.CB_SETTOPINDEX.

96.3.54 Messages.CB_SHOWDROPDOWN Constant

Pascal

```
CB_SHOWDROPDOWN = $014F;
```

C++

```
CB_SHOWDROPDOWN = 0x014F;
```

File

Messages

Description

This is constant Messages.CB_SHOWDROPDOWN.

96.3.55 Messages.CBN_CLOSEUP Constant

Pascal

```
CBN_CLOSEUP = 8;
```

C++

```
CBN_CLOSEUP = 8;
```

File

Messages

Description

This is constant Messages.CBN_CLOSEUP.

96.3.56 Messages.CBN_DBLCLK Constant

Pascal

```
CBN_DBLCLK = 2;
```

C++

```
CBN_DBLCLK = 2;
```

File

Messages

Description

This is constant Messages.CBN_DBLCLK.

96.3.57 Messages.CBN_DROPDOWN Constant

Pascal

```
CBN_DROPDOWN = 7;
```

C++

```
CBN_DROPDOWN = 7;
```

File

Messages

Description

This is constant Messages.CBN_DROPDOWN.

96.3.58 Messages.CBN_EDITCHANGE Constant

Pascal

```
CBN_EDITCHANGE = 5;
```

C++

```
CBN_EDITCHANGE = 5;
```

File

Messages

Description

This is constant Messages.CBN_EDITCHANGE.

96.3.59 Messages.CBN_EDITUPDATE Constant

Pascal

```
CBN_EDITUPDATE = 6;
```

C++

```
CBN_EDITUPDATE = 6;
```

File

Messages

Description

This is constant Messages.CBN_EDITUPDATE.

96.3.60 Messages.CBN_ERRSPACE Constant

Pascal

```
CBN_ERRSPACE = (-1);
```

C++

```
CBN_ERRSPACE = (-1);
```

File

Messages

Description

This is constant Messages.CBN_ERRSPACE.

96.3.61 Messages.CBN_KILLFOCUS Constant

Pascal

```
CBN_KILLFOCUS = 4;
```

C++

```
CBN_KILLFOCUS = 4;
```

File

Messages

Description

This is constant Messages.CBN_KILLFOCUS.

96.3.62 Messages.CBN_SELCHANGE Constant

Pascal

```
CBN_SELCHANGE = 1;
```

C++

```
CBN_SELCHANGE = 1;
```

File

Messages

Description

This is constant Messages.CBN_SELCHANGE.

96.3.63 Messages.CBN_SELENDCANCEL Constant

Pascal

```
CBN_SELENDCANCEL = 10;
```

C++

```
CBN_SELENDCANCEL = 10;
```

File

Messages

Description

This is constant Messages.CBN_SELENDCANCEL.

96.3.64 Messages.CBN_SELENDOK Constant

Pascal

```
CBN_SELENDOK = 9;
```

C++

```
CBN_SELENDOK = 9;
```

File

Messages

Description

This is constant Messages.CBN_SELENDOK.

96.3.65 Messages.CBN_SETFOCUS Constant

Pascal

```
CBN_SETFOCUS = 3;
```

C++

```
CBN_SETFOCUS = 3;
```

File

Messages

Description

This is constant Messages.CBN_SETFOCUS.

96.3.66 Messages.DM_GETDEFID Constant

Pascal

```
DM_GETDEFID = (WM_USER+0);
```

C++

```
DM_GETDEFID = (WM_USER+0);
```

File

Messages

Description

This is constant Messages.DM_GETDEFID.

96.3.67 Messages.DM_REPOSITION Constant

Pascal

```
DM_REPOSITION = (WM_USER+2);
```

C++

```
DM_REPOSITION = (WM_USER+2);
```

File

Messages

Description

This is constant Messages.DM_REPOSITION.

96.3.68 Messages.DM_SETDEFID Constant

Pascal

```
DM_SETDEFID = (WM_USER+1);
```

C++

```
DM_SETDEFID = (WM_USER+1);
```

File

Messages

Description

This is constant Messages.DM_SETDEFID.

96.3.69 Messages.EM_CANUNDO Constant

Pascal

```
EM_CANUNDO = $00C6;
```

C++

```
EM_CANUNDO = 0x00C6;
```

File

Messages

Description

This is constant Messages.EM_CANUNDO.

96.3.70 Messages.EM_CHARFROMPOS Constant

Pascal

```
EM_CHARFROMPOS = 215;
```

C++

```
EM_CHARFROMPOS = 215;
```

File

Messages

Description

This is constant Messages.EM_CHARFROMPOS.

96.3.71 Messages.EM_EMPTYUNDOBUFFER Constant

Pascal

```
EM_EMPTYUNDOBUFFER = $00CD;
```

C++

```
EM_EMPTYUNDOBUFFER = 0x00CD;
```

File

Messages

Description

This is constant Messages.EM_EMPTYUNDOBUFFER.

96.3.72 Messages.EM_FMTLINES Constant

Pascal

```
EM_FMTLINES = $00C8;
```

C++

```
EM_FMTLINES = 0x00C8;
```

File

Messages

Description

This is constant Messages.EM_FMTLINES.

96.3.73 Messages.EM_GETFIRSTVISIBLELINE Constant

Pascal

```
EM_GETFIRSTVISIBLELINE = $00CE;
```

C++

```
EM_GETFIRSTVISIBLELINE = 0x00CE;
```

File

Messages

Description

This is constant Messages.EM_GETFIRSTVISIBLELINE.

96.3.74 Messages.EM_GETHANDLE Constant

Pascal

```
EM_GETHANDLE = $00BD;
```

C++

```
EM_GETHANDLE = 0x00BD;
```

File

Messages

Description

This is constant Messages.EM_GETHANDLE.

96.3.75 Messages.EM_GETIMESTATUS Constant

Pascal

```
EM_GETIMESTATUS = 217;
```

C++

```
EM_GETIMESTATUS = 217;
```

File

Messages

Description

This is constant Messages.EM_GETIMESTATUS.

96.3.76 Messages.EM_GETLIMITTEXT Constant

Pascal

```
EM_GETLIMITTEXT = 213;
```

C++

```
EM_GETLIMITTEXT = 213;
```

File

Messages

Description

This is constant Messages.EM_GETLIMITTEXT.

96.3.77 Messages.EM_GETLINE Constant

Pascal

```
EM_GETLINE = $00C4;
```

C++

```
EM_GETLINE = 0x00C4;
```

File

Messages

Description

This is constant Messages.EM_GETLINE.

96.3.78 Messages.EM_GETLINECOUNT Constant

Pascal

```
EM_GETLINECOUNT = $00BA;
```

C++

```
EM_GETLINECOUNT = 0x00BA;
```

File

Messages

Description

This is constant Messages.EM_GETLINECOUNT.

96.3.79 Messages.EM_GETMARGINS Constant

Pascal

```
EM_GETMARGINS = 212;
```

C++

```
EM_GETMARGINS = 212;
```

File

Messages

Description

This is constant Messages.EM_GETMARGINS.

96.3.80 Messages.EM_GETMODIFY Constant

Pascal

```
EM_GETMODIFY = $00B8;
```

C++

```
EM_GETMODIFY = 0x00B8;
```

File

Messages

Description

This is constant Messages.EM_GETMODIFY.

96.3.81 Messages.EM_GETPASSWORDCHAR Constant

Pascal

```
EM_GETPASSWORDCHAR = $00D2;
```

C++

```
EM_GETPASSWORDCHAR = 0x00D2;
```

File

Messages

Description

This is constant Messages.EM_GETPASSWORDCHAR.

96.3.82 Messages.EM_GETRECT Constant

Pascal

```
EM_GETRECT = $00B2;
```

C++

```
EM_GETRECT = 0x00B2;
```

File

Messages

Description

This is constant Messages.EM_GETRECT.

96.3.83 Messages.EM_GETSEL Constant

Pascal

```
EM_GETSEL = $00B0;
```

C++

```
EM_GETSEL = 0x00B0;
```

File

Messages

Description

This is constant Messages.EM_GETSEL.

96.3.84 Messages.EM_GETTHUMB Constant

Pascal

```
EM_GETTHUMB = $00BE;
```

C++

```
EM_GETTHUMB = 0x00BE;
```

File

Messages

Description

This is constant Messages.EM_GETTHUMB.

96.3.85 Messages.EM_GETWORDBREAKPROC Constant

Pascal

```
EM_GETWORDBREAKPROC = $00D1;
```

C++

```
EM_GETWORDBREAKPROC = 0x00D1;
```

File

Messages

Description

This is constant Messages.EM_GETWORDBREAKPROC.

96.3.86 Messages.EM_LIMITTEXT Constant

Pascal

```
EM_LIMITTEXT = $00C5;
```

C++

```
EM_LIMITTEXT = 0x00C5;
```

File

Messages

Description

This is constant Messages.EM_LIMITTEXT.

96.3.87 Messages.EM_LINEFROMCHAR Constant

Pascal

```
EM_LINEFROMCHAR = $00C9;
```

C++

```
EM_LINEFROMCHAR = 0x00C9;
```

File

Messages

Description

This is constant Messages.EM_LINEFROMCHAR.

96.3.88 Messages.EM_LINEINDEX Constant

Pascal

```
EM_LINEINDEX = $00BB;
```

C++

```
EM_LINEINDEX = 0x00BB;
```

File

Messages

Description

This is constant Messages.EM_LINEINDEX.

96.3.89 Messages.EM_LINELENGTH Constant

Pascal

```
EM_LINELENGTH = $00C1;
```

C++

```
EM_LINELENGTH = 0x00C1;
```

File

Messages

Description

This is constant Messages.EM_LINELENGTH.

96.3.90 Messages.EM_LINESCROLL Constant

Pascal

```
EM_LINESCROLL = $00B6;
```

C++

```
EM_LINESCROLL = 0x00B6;
```

File

Messages

Description

This is constant Messages.EM_LINESCROLL.

96.3.91 Messages.EM_POSFROMCHAR Constant

Pascal

```
EM_POSFROMCHAR = 214;
```

C++

```
EM_POSFROMCHAR = 214;
```

File

Messages

Description

This is constant Messages.EM_POSFROMCHAR.

96.3.92 Messages.EM_REPLACESEL Constant

Pascal

```
EM_REPLACESEL = $00C2;
```

C++

```
EM_REPLACESEL = 0x00C2;
```

File

Messages

Description

This is constant Messages.EM_REPLACESEL.

96.3.93 Messages.EM_SCROLL Constant

Pascal

```
EM_SCROLL = $00B5;
```

C++

```
EM_SCROLL = 0x00B5;
```

File

Messages

Description

This is constant Messages.EM_SCROLL.

96.3.94 Messages.EM_SCROLLCARET Constant

Pascal

```
EM_SCROLLCARET = $00B7;
```

C++

```
EM_SCROLLCARET = 0x00B7;
```

File

Messages

Description

This is constant Messages.EM_SCROLLCARET.

96.3.95 Messages.EM_SETHANDLE Constant

Pascal

```
EM_SETHANDLE = $00BC;
```

C++

```
EM_SETHANDLE = 0x00BC;
```

File

Messages

Description

This is constant Messages.EM_SETHANDLE.

96.3.96 Messages.EM_SETIMESTATUS Constant

Pascal

```
EM_SETIMESTATUS = 216;
```

C++

```
EM_SETIMESTATUS = 216;
```

File

Messages

Description

This is constant Messages.EM_SETIMESTATUS.

96.3.97 Messages.EM_SETLIMITTEXT Constant

Pascal

```
EM_SETLIMITTEXT = EM_LIMITTEXT;
```

C++

```
EM_SETLIMITTEXT = EM_LIMITTEXT;
```

File

Messages

Description

This is constant Messages.EM_SETLIMITTEXT.

96.3.98 Messages.EM_SETMARGINS Constant

Pascal

```
EM_SETMARGINS = 211;
```

C++

```
EM_SETMARGINS = 211;
```

File

Messages

Description

This is constant Messages.EM_SETMARGINS.

96.3.99 Messages.EM_SETMODIFY Constant

Pascal

```
EM_SETMODIFY = $00B9;
```

C++

```
EM_SETMODIFY = 0x00B9;
```

File

Messages

Description

This is constant Messages.EM_SETMODIFY.

96.3.100 Messages.EM_SETPASSWORDCHAR Constant

Pascal

```
EM_SETPASSWORDCHAR = $00CC;
```

C++

```
EM_SETPASSWORDCHAR = 0x00CC;
```

File

Messages

Description

This is constant Messages.EM_SETPASSWORDCHAR.

96.3.101 Messages.EM_SETREADONLY Constant

Pascal

```
EM_SETREADONLY = $00CF;
```

C++

```
EM_SETREADONLY = 0x00CF;
```

File

Messages

Description

This is constant Messages.EM_SETREADONLY.

96.3.102 Messages.EM_SETRECT Constant

Pascal

```
EM_SETRECT = $00B3;
```

C++

```
EM_SETRECT = 0x00B3;
```

File

Messages

Description

This is constant Messages.EM_SETRECT.

96.3.103 Messages.EM_SETRECTNP Constant

Pascal

```
EM_SETRECTNP = $00B4;
```

C++

```
EM_SETRECTNP = 0x00B4;
```

File

Messages

Description

This is constant Messages.EM_SETRECTNP.

96.3.104 Messages.EM_SETSEL Constant

Pascal

```
EM_SETSEL = $00B1;
```

C++

```
EM_SETSEL = 0x00B1;
```

File

Messages

Description

This is constant Messages.EM_SETSEL.

96.3.105 Messages.EM_SETTABSTOPS Constant

Pascal

```
EM_SETTABSTOPS = $00CB;
```

C++

```
EM_SETTABSTOPS = 0x00CB;
```

File

Messages

Description

This is constant Messages.EM_SETTABSTOPS.

96.3.106 Messages.EM_SETWORDBREAKPROC Constant

Pascal

```
EM_SETWORDBREAKPROC = $00D0;
```

C++

```
EM_SETWORDBREAKPROC = 0x00D0;
```

File

Messages

Description

This is constant Messages.EM_SETWORDBREAKPROC.

96.3.107 Messages.EM_UNDO Constant

Pascal

```
EM_UNDO = $00C7;
```

C++

```
EM_UNDO = 0x00C7;
```

File

Messages

Description

This is constant Messages.EM_UNDO.

96.3.108 Messages.EN_CHANGE Constant

Pascal

```
EN_CHANGE = $0300;
```

C++

```
EN_CHANGE = 0x0300;
```

File

Messages

Description

This is constant Messages.EN_CHANGE.

96.3.109 Messages.EN_ERRSPACE Constant

Pascal

```
EN_ERRSPACE = $0500;
```

C++

```
EN_ERRSPACE = 0x0500;
```

File

Messages

Description

This is constant Messages.EN_ERRSPACE.

96.3.110 Messages.EN_HSCROLL Constant

Pascal

```
EN_HSCROLL = $0601;
```

C++

```
EN_HSCROLL = 0x0601;
```

File

Messages

Description

This is constant Messages.EN_HSCROLL.

96.3.111 Messages.EN_KILLFOCUS Constant

Pascal

```
EN_KILLFOCUS = $0200;
```

C++

```
EN_KILLFOCUS = 0x0200;
```

File

Messages

Description

This is constant Messages.EN_KILLFOCUS.

96.3.112 Messages.EN_MAXTEXT Constant

Pascal

```
EN_MAXTEXT = $0501;
```

C++

```
EN_MAXTEXT = 0x0501;
```

File

Messages

Description

This is constant Messages.EN_MAXTEXT.

96.3.113 Messages.EN_SETFOCUS Constant

Pascal

```
EN_SETFOCUS = $0100;
```

C++

```
EN_SETFOCUS = 0x0100;
```

File

Messages

Description

This is constant Messages.EN_SETFOCUS.

96.3.114 Messages.EN_UPDATE Constant

Pascal

```
EN_UPDATE = $0400;
```

C++

```
EN_UPDATE = 0x0400;
```

File

Messages

Description

This is constant Messages.EN_UPDATE.

96.3.115 Messages.EN_VSCROLL Constant

Pascal

```
EN_VSCROLL = $0602;
```

C++

```
EN_VSCROLL = 0x0602;
```

File

Messages

Description

This is constant Messages.EN_VSCROLL.

96.3.116 Messages.LB_ADDFILE Constant

Pascal

```
LB_ADDFILE = $0196;
```

C++

```
LB_ADDFILE = 0x0196;
```

File

Messages

Description

This is constant Messages.LB_ADDFILE.

96.3.117 Messages.LB_ADDSTRING Constant

Pascal

```
LB_ADDSTRING = $0180;
```

C++

```
LB_ADDSTRING = 0x0180;
```

File

Messages

Description

This is constant Messages.LB_ADDSTRING.

96.3.118 Messages.LB_DELETESTRING Constant

Pascal

```
LB_DELETESTRING = $0182;
```

C++

```
LB_DELETESTRING = 0x0182;
```

File

Messages

Description

This is constant Messages.LB_DELETESTRING.

96.3.119 Messages.LB_DIR Constant

Pascal

```
LB_DIR = $018D;
```

C++

```
LB_DIR = 0x018D;
```

File

Messages

Description

This is constant Messages.LB_DIR.

96.3.120 Messages.LB_FINDSTRING Constant

Pascal

```
LB_FINDSTRING = $018F;
```

C++

```
LB_FINDSTRING = 0x018F;
```

File

Messages

Description

This is constant Messages.LB_FINDSTRING.

96.3.121 Messages.LB_FINDSTRINGEXACT Constant

Pascal

```
LB_FINDSTRINGEXACT = $01A2;
```

C++

```
LB_FINDSTRINGEXACT = 0x01A2;
```

File

Messages

Description

This is constant Messages.LB_FINDSTRINGEXACT.

96.3.122 Messages.LB_GETANCHORINDEX Constant

Pascal

```
LB_GETANCHORINDEX = $019D;
```

C++

```
LB_GETANCHORINDEX = 0x019D;
```

File

Messages

Description

This is constant Messages.LB_GETANCHORINDEX.

96.3.123 Messages.LB_GETCARETINDEX Constant

Pascal

```
LB_GETCARETINDEX = $019F;
```

C++

```
LB_GETCARETINDEX = 0x019F;
```

File

Messages

Description

This is constant Messages.LB_GETCARETINDEX.

96.3.124 Messages.LB_GETCOUNT Constant

Pascal

```
LB_GETCOUNT = $018B;
```

C++

```
LB_GETCOUNT = 0x018B;
```

File

Messages

Description

This is constant Messages.LB_GETCOUNT.

96.3.125 Messages.LB_GETCURSEL Constant

Pascal

```
LB_GETCURSEL = $0188;
```

C++

```
LB_GETCURSEL = 0x0188;
```

File

Messages

Description

This is constant Messages.LB_GETCURSEL.

96.3.126 Messages.LB_GETHORIZONTALEXTENT Constant

Pascal

```
LB_GETHORIZONTALEXTENT = $0193;
```

C++

```
LB_GETHORIZONTALEXTENT = 0x0193;
```

File

Messages

Description

This is constant Messages.LB_GETHORIZONTALEXTENT.

96.3.127 Messages.LB_GETITEMDATA Constant

Pascal

```
LB_GETITEMDATA = $0199;
```

C++

```
LB_GETITEMDATA = 0x0199;
```

File

Messages

Description

This is constant Messages.LB_GETITEMDATA.

96.3.128 Messages.LB_GETITEMHEIGHT Constant

Pascal

```
LB_GETITEMHEIGHT = $01A1;
```

C++

```
LB_GETITEMHEIGHT = 0x01A1;
```

File

Messages

Description

This is constant Messages.LB_GETITEMHEIGHT.

96.3.129 Messages.LB_GETITEMRECT Constant

Pascal

```
LB_GETITEMRECT = $0198;
```

C++

```
LB_GETITEMRECT = 0x0198;
```

File

Messages

Description

This is constant Messages.LB_GETITEMRECT.

96.3.130 Messages.LB_GETLOCALE Constant

Pascal

```
LB_GETLOCALE = $01A6;
```

C++

```
LB_GETLOCALE = 0x01A6;
```

File

Messages

Description

This is constant Messages.LB_GETLOCALE.

96.3.131 Messages.LB_GETSEL Constant

Pascal

```
LB_GETSEL = $0187;
```

C++

```
LB_GETSEL = 0x0187;
```

File

Messages

Description

This is constant Messages.LB_GETSEL.

96.3.132 Messages.LB_GETSELCOUNT Constant

Pascal

```
LB_GETSELCOUNT = $0190;
```

C++

```
LB_GETSELCOUNT = 0x0190;
```

File

Messages

Description

This is constant Messages.LB_GETSELCOUNT.

96.3.133 Messages.LB_GETSELITEMS Constant

Pascal

```
LB_GETSELITEMS = $0191;
```

C++

```
LB_GETSELITEMS = 0x0191;
```

File

Messages

Description

This is constant Messages.LB_GETSELITEMS.

96.3.134 Messages.LB_GETTEXT Constant

Pascal

```
LB_GETTEXT = $0189;
```

C++

```
LB_GETTEXT = 0x0189;
```

File

Messages

Description

This is constant Messages.LB_GETTEXT.

96.3.135 Messages.LB_GETTEXTLEN Constant

Pascal

```
LB_GETTEXTLEN = $018A;
```

C++

```
LB_GETTEXTLEN = 0x018A;
```

File

Messages

Description

This is constant Messages.LB_GETTEXTLEN.

96.3.136 Messages.LB_GETTOPINDEX Constant

Pascal

```
LB_GETTOPINDEX = $018E;
```

C++

```
LB_GETTOPINDEX = 0x018E;
```

File

Messages

Description

This is constant Messages.LB_GETTOPINDEX.

96.3.137 Messages.LB_INITSTORAGE Constant

Pascal

```
LB_INITSTORAGE = $01A8;
```

C++

```
LB_INITSTORAGE = 0x01A8;
```

File

Messages

Description

This is constant Messages.LB_INITSTORAGE.

96.3.138 Messages.LB_INSERTSTRING Constant

Pascal

```
LB_INSERTSTRING = $0181;
```

C++

```
LB_INSERTSTRING = 0x0181;
```

File

Messages

Description

This is constant Messages.LB_INSERTSTRING.

96.3.139 Messages.LB_ITEMFROMPOINT Constant

Pascal

```
LB_ITEMFROMPOINT = $01A9;
```

C++

```
LB_ITEMFROMPOINT = 0x01A9;
```

File

Messages

Description

This is constant Messages.LB_ITEMFROMPOINT.

96.3.140 Messages.LB_MSGMAX Constant

Pascal

```
LB_MSGMAX = 432;
```

C++

```
LB_MSGMAX = 432;
```

File

Messages

Description

This is constant Messages.LB_MSGMAX.

96.3.141 Messages.LB_RESETCONTENT Constant

Pascal

```
LB_RESETCONTENT = $0184;
```

C++

```
LB_RESETCONTENT = 0x0184;
```

File

Messages

Description

This is constant Messages.LB_RESETCONTENT.

96.3.142 Messages.LB_SELECTSTRING Constant

Pascal

```
LB_SELECTSTRING = $018C;
```

C++

```
LB_SELECTSTRING = 0x018C;
```

File

Messages

Description

This is constant Messages.LB_SELECTSTRING.

96.3.143 Messages.LB_SELITEMRANGE Constant

Pascal

```
LB_SELITEMRANGE = $019B;
```

C++

```
LB_SELITEMRANGE = 0x019B;
```

File

Messages

Description

This is constant Messages.LB_SELITEMRANGE.

96.3.144 Messages.LB_SELITEMRANGEEX Constant

Pascal

```
LB_SELITEMRANGEEX = $0183;
```

C++

```
LB_SELITEMRANGEEX = 0x0183;
```

File

Messages

Description

This is constant Messages.LB_SELITEMRANGEEX.

96.3.145 Messages.LB_SETANCHORINDEX Constant

Pascal

```
LB_SETANCHORINDEX = $019C;
```

C++

```
LB_SETANCHORINDEX = 0x019C;
```

File

Messages

Description

This is constant Messages.LB_SETANCHORINDEX.

96.3.146 Messages.LB_SETCARETINDEX Constant

Pascal

```
LB_SETCARETINDEX = $019E;
```

C++

```
LB_SETCARETINDEX = 0x019E;
```

File

Messages

Description

This is constant Messages.LB_SETCARETINDEX.

96.3.147 Messages.LB_SETCOLUMNWIDTH Constant

Pascal

```
LB_SETCOLUMNWIDTH = $0195;
```

C++

```
LB_SETCOLUMNWIDTH = 0x0195;
```

File

Messages

Description

This is constant Messages.LB_SETCOLUMNWIDTH.

96.3.148 Messages.LB_SETCOUNT Constant

Pascal

```
LB_SETCOUNT = $01A7;
```

C++

```
LB_SETCOUNT = 0x01A7;
```

File

Messages

Description

This is constant Messages.LB_SETCOUNT.

96.3.149 Messages.LB_SETCURSEL Constant

Pascal

```
LB_SETCURSEL = $0186;
```

C++

```
LB_SETCURSEL = 0x0186;
```

File

Messages

Description

This is constant Messages.LB_SETCURSEL.

96.3.150 Messages.LB_SETHORIZONTALEXTENT Constant

Pascal

```
LB_SETHORIZONTALEXTENT = $0194;
```

C++

```
LB_SETHORIZONTALEXTENT = 0x0194;
```

File

Messages

Description

This is constant Messages.LB_SETHORIZONTALEXTENT.

96.3.151 Messages.LB_SETITEMDATA Constant

Pascal

```
LB_SETITEMDATA = $019A;
```

C++

```
LB_SETITEMDATA = 0x019A;
```

File

Messages

Description

This is constant Messages.LB_SETITEMDATA.

96.3.152 Messages.LB_SETITEMHEIGHT Constant

Pascal

```
LB_SETITEMHEIGHT = $01A0;
```

C++

```
LB_SETITEMHEIGHT = 0x01A0;
```

File

Messages

Description

This is constant Messages.LB_SETITEMHEIGHT.

96.3.153 Messages.LB_SETLOCALE Constant

Pascal

```
LB_SETLOCALE = $01A5;
```

C++

```
LB_SETLOCALE = 0x01A5;
```

File

Messages

Description

This is constant Messages.LB_SETLOCALE.

96.3.154 Messages.LB_SETSEL Constant

Pascal

```
LB_SETSEL = $0185;
```

C++

```
LB_SETSEL = 0x0185;
```

File

Messages

Description

This is constant Messages.LB_SETSEL.

96.3.155 Messages.LB_SETTABSTOPS Constant

Pascal

```
LB_SETTABSTOPS = $0192;
```

C++

```
LB_SETTABSTOPS = 0x0192;
```

File

Messages

Description

This is constant Messages.LB_SETTABSTOPS.

96.3.156 Messages.LB_SETTOPINDEX Constant

Pascal

```
LB_SETTOPINDEX = $0197;
```

C++

```
LB_SETTOPINDEX = 0x0197;
```

File

Messages

Description

This is constant Messages.LB_SETTOPINDEX.

96.3.157 Messages.LBN_DBLCLK Constant

Pascal

```
LBN_DBLCLK = 2;
```

C++

```
LBN_DBLCLK = 2;
```

File

Messages

Description

This is constant Messages.LBN_DBLCLK.

96.3.158 Messages.LBN_ERRSPACE Constant

Pascal

```
LBN_ERRSPACE = (-2);
```

C++

```
LBN_ERRSPACE = (-2);
```

File

Messages

Description

This is constant Messages.LBN_ERRSPACE.

96.3.159 Messages.LBN_KILLFOCUS Constant

Pascal

```
LBN_KILLFOCUS = 5;
```

C++

```
LBN_KILLFOCUS = 5;
```

File

Messages

Description

This is constant Messages.LBN_KILLFOCUS.

96.3.160 Messages.LBN_SELCANCEL Constant

Pascal

```
LBN_SELCANCEL = 3;
```

C++

```
LBN_SELCANCEL = 3;
```

File

Messages

Description

This is constant Messages.LBN_SELCANCEL.

96.3.161 Messages.LBN_SELCHANGE Constant

Pascal

```
LBN_SELCHANGE = 1;
```

C++

```
LBN_SELCHANGE = 1;
```

File

Messages

Description

This is constant Messages.LBN_SELCHANGE.

96.3.162 Messages.LBN_SETFOCUS Constant

Pascal

```
LBN_SETFOCUS = 4;
```

C++

```
LBN_SETFOCUS = 4;
```

File

Messages

Description

This is constant Messages.LBN_SETFOCUS.

96.3.163 Messages.PSM_PAGEINFO Constant

Pascal

```
PSM_PAGEINFO = (WM_USER+100);
```

C++

```
PSM_PAGEINFO = (WM_USER+100);
```

File

Messages

Description

This is constant Messages.PSM_PAGEINFO.

96.3.164 Messages.PSM_SHEETINFO Constant

Pascal

```
PSM_SHEETINFO = (WM_USER+101);
```

C++

```
PSM_SHEETINFO = (WM_USER+101);
```

File

Messages

Description

This is constant Messages.PSM_SHEETINFO.

96.3.165 Messages.SBM_ENABLE_ARROWS Constant

Pascal

```
SBM_ENABLE_ARROWS = 228;
```

C++

```
SBM_ENABLE_ARROWS = 228;
```

File

Messages

Description

This is constant Messages.SBM_ENABLE_ARROWS.

96.3.166 Messages.SBM_GETPOS Constant

Pascal

```
SBM_GETPOS = 225;
```

C++

```
SBM_GETPOS = 225;
```

File

Messages

Description

This is constant Messages.SBM_GETPOS.

96.3.167 Messages.SBM_GETRANGE Constant

Pascal

```
SBM_GETRANGE = 227;
```

C++

```
SBM_GETRANGE = 227;
```

File

Messages

Description

This is constant Messages.SBM_GETRANGE.

96.3.168 Messages.SBM_GETSCROLLINFO Constant

Pascal

```
SBM_GETSCROLLINFO = 234;
```

C++

```
SBM_GETSCROLLINFO = 234;
```

File

Messages

Description

This is constant Messages.SBM_GETSCROLLINFO.

96.3.169 Messages.SBM_SETPOS Constant

Pascal

```
SBM_SETPOS = 224;
```

C++

```
SBM_SETPOS = 224;
```

File

Messages

Description

This is constant Messages.SBM_SETPOS.

96.3.170 Messages.SBM_SETRANGE Constant

Pascal

```
SBM_SETRANGE = 226;
```

C++

```
SBM_SETRANGE = 226;
```

File

Messages

Description

This is constant Messages.SBM_SETRANGE.

96.3.171 Messages.SBM_SETRANGEREDRAW Constant

Pascal

```
SBM_SETRANGEREDRAW = 230;
```

C++

```
SBM_SETRANGEREDRAW = 230;
```

File

Messages

Description

This is constant Messages.SBM_SETRANGEREDRAW.

96.3.172 Messages.SBM_SETSCROLLINFO Constant

Pascal

```
SBM_SETSCROLLINFO = 233;
```

C++

```
SBM_SETSCROLLINFO = 233;
```

File

Messages

Description

This is constant Messages.SBM_SETSCROLLINFO.

96.3.173 Messages.WM_ACTIVATE Constant

Pascal

```
WM_ACTIVATE = $0006;
```

C++

```
WM_ACTIVATE = 0x0006;
```

File

Messages

Description

This is constant Messages.WM_ACTIVATE.

96.3.174 Messages.WM_ACTIVATEAPP Constant

Pascal

```
WM_ACTIVATEAPP = $001C;
```

C++

```
WM_ACTIVATEAPP = 0x001C;
```

File

Messages

Description

This is constant Messages.WM_ACTIVATEAPP.

96.3.175 Messages.WM_APP Constant

Pascal

```
WM_APP = $8000;
```

C++

```
WM_APP = 0x8000;
```

File

Messages

Description

This is constant Messages.WM_APP.

96.3.176 Messages.WM_APPCOMMAND Constant

Pascal

```
WM_APPCOMMAND = $0319;
```

C++

```
WM_APPCOMMAND = 0x0319;
```

File

Messages

Description

This is constant Messages.WM_APPCOMMAND.

96.3.177 Messages.WM_ASKCBFORMATNAME Constant

Pascal

```
WM_ASKCBFORMATNAME = $030C;
```

C++

```
WM_ASKCBFORMATNAME = 0x030C;
```

File

Messages

Description

This is constant Messages.WM_ASKCBFORMATNAME.

96.3.178 Messages.WM_CANCELJOURNAL Constant

Pascal

```
WM_CANCELJOURNAL = $004B;
```

C++

```
WM_CANCELJOURNAL = 0x004B;
```

File

Messages

Description

This is constant Messages.WM_CANCELJOURNAL.

96.3.179 Messages.WM_CANCELMODE Constant

Pascal

```
WM_CANCELMODE = $001F;
```

C++

```
WM_CANCELMODE = 0x001F;
```

File

Messages

Description

This is constant Messages.WM_CANCELMODE.

96.3.180 Messages.WM_CAPTURECHANGED Constant

Pascal

```
WM_CAPTURECHANGED = 533;
```

C++

```
WM_CAPTURECHANGED = 533;
```

File

Messages

Description

This is constant Messages.WM_CAPTURECHANGED.

96.3.181 Messages.WM_CHANGECBCHAIN Constant

Pascal

```
WM_CHANGECBCHAIN = $030D;
```

C++

```
WM_CHANGECBCHAIN = 0x030D;
```

File

Messages

Description

This is constant Messages.WM_CHANGECBCHAIN.

96.3.182 Messages.WM_CHANGEUISTATE Constant

Pascal

```
WM_CHANGEUISTATE = $0127;
```

C++

```
WM_CHANGEUISTATE = 0x0127;
```

File

Messages

Description

This is constant Messages.WM_CHANGEUISTATE.

96.3.183 Messages.WM_CHAR Constant

Pascal

```
WM_CHAR = $0102;
```

C++

```
WM_CHAR = 0x0102;
```

File

Messages

Description

This is constant Messages.WM_CHAR.

96.3.184 Messages.WM_CHARTOITEM Constant

Pascal

```
WM_CHARTOITEM = $002F;
```

C++

```
WM_CHARTOITEM = 0x002F;
```

File

Messages

Description

This is constant Messages.WM_CHARTOITEM.

96.3.185 Messages.WM_CHILDACTIVATE Constant

Pascal

```
WM_CHILDACTIVATE = $0022;
```

C++

```
WM_CHILDACTIVATE = 0x0022;
```

File

Messages

Description

This is constant Messages.WM_CHILDACTIVATE.

96.3.186 Messages.WM_CLEAR Constant

Pascal

```
WM_CLEAR = $0303;
```

C++

```
WM_CLEAR = 0x0303;
```

File

Messages

Description

This is constant Messages.WM_CLEAR.

96.3.187 Messages.WM_CLOSE Constant

Pascal

```
WM_CLOSE = $0010;
```

C++

```
WM_CLOSE = 0x0010;
```

File

Messages

Description

This is constant Messages.WM_CLOSE.

96.3.188 Messages.WM_COALESCE_FIRST Constant

Pascal

```
WM_COALESCE_FIRST = $0390;
```

C++

```
WM_COALESCE_FIRST = 0x0390;
```

File

Messages

Description

This is constant Messages.WM_COALESCE_FIRST.

96.3.189 Messages.WM_COALESCE_LAST Constant

Pascal

```
WM_COALESCE_LAST = $039F;
```

C++

```
WM_COALESCE_LAST = 0x039F;
```

File

Messages

Description

This is constant Messages.WM_COALESCE_LAST.

96.3.190 Messages.WM_COMMAND Constant

Pascal

```
WM_COMMAND = $0111;
```

C++

```
WM_COMMAND = 0x0111;
```

File

Messages

Description

This is constant Messages.WM_COMMAND.

96.3.191 Messages.WM_COMMNOTIFY Constant

Pascal

```
WM_COMMNOTIFY = $0044;
```

C++

```
WM_COMMNOTIFY = 0x0044;
```

File

Messages

Description

This is constant Messages.WM_COMMNOTIFY.

96.3.192 Messages.WM_COMPACTING Constant

Pascal

```
WM_COMPACTING = $0041;
```

C++

```
WM_COMPACTING = 0x0041;
```

File

Messages

Description

This is constant Messages.WM_COMPACTING.

96.3.193 Messages.WM_COMPAREITEM Constant

Pascal

```
WM_COMPAREITEM = $0039;
```

C++

```
WM_COMPAREITEM = 0x0039;
```

File

Messages

Description

This is constant Messages.WM_COMPAREITEM.

96.3.194 Messages.WM_CONTEXTMENU Constant

Pascal

```
WM_CONTEXTMENU = $007B;
```

C++

```
WM_CONTEXTMENU = 0x007B;
```

File

Messages

Description

This is constant Messages.WM_CONTEXTMENU.

96.3.195 Messages.WM_COPY Constant

Pascal

```
WM_COPY = $0301;
```

C++

```
WM_COPY = 0x0301;
```

File

Messages

Description

This is constant Messages.WM_COPY.

96.3.196 Messages.WM_COPYDATA Constant

Pascal

```
WM_COPYDATA = $004A;
```

C++

```
WM_COPYDATA = 0x004A;
```

File

Messages

Description

This is constant Messages.WM_COPYDATA.

96.3.197 Messages.WM_CREATE Constant

Pascal

```
WM_CREATE = $0001;
```

C++

```
WM_CREATE = 0x0001;
```

File

Messages

Description

This is constant Messages.WM_CREATE.

96.3.198 Messages.WM_CTLCOLOR Constant

Pascal

```
WM_CTLCOLOR = $0019;
```

C++

```
WM_CTLCOLOR = 0x0019;
```

File

Messages

Description

This is constant Messages.WM_CTLCOLOR.

96.3.199 Messages.WM_CTLCOLORBTN Constant

Pascal

```
WM_CTLCOLORBTN = $0135;
```

C++

```
WM_CTLCOLORBTN = 0x0135;
```

File

Messages

Description

This is constant Messages.WM_CTLCOLORBTN.

96.3.200 Messages.WM_CTLCOLORDLG Constant

Pascal

```
WM_CTLCOLORDLG = $0136;
```

C++

```
WM_CTLCOLORDLG = 0x0136;
```

File

Messages

Description

This is constant Messages.WM_CTLCOLORDLG.

96.3.201 Messages.WM_CTLCOLOREDIT Constant

Pascal

```
WM_CTLCOLOREDIT = $0133;
```

C++

```
WM_CTLCOLOREDIT = 0x0133;
```

File

Messages

Description

This is constant Messages.WM_CTLCOLOREDIT.

96.3.202 Messages.WM_CTLCOLORLISTBOX Constant

Pascal

```
WM_CTLCOLORLISTBOX = $0134;
```

C++

```
WM_CTLCOLORLISTBOX = 0x0134;
```

File

Messages

Description

This is constant Messages.WM_CTLCOLORLISTBOX.

96.3.203 Messages.WM_CTLCOLORMSGBOX Constant

Pascal

```
WM_CTLCOLORMSGBOX = $0132;
```

C++

```
WM_CTLCOLORMSGBOX = 0x0132;
```

File

Messages

Description

This is constant Messages.WM_CTLCOLORMSGBOX.

96.3.204 Messages.WM_CTLCOLORSCROLLBAR Constant

Pascal

```
WM_CTLCOLORSCROLLBAR = $0137;
```

C++

```
WM_CTLCOLORSCROLLBAR = 0x0137;
```

File

Messages

Description

This is constant Messages.WM_CTLCOLORSCROLLBAR.

96.3.205 Messages.WM_CTLCOLORSTATIC Constant

Pascal

```
WM_CTLCOLORSTATIC = $0138;
```

C++

```
WM_CTLCOLORSTATIC = 0x0138;
```

File

Messages

Description

This is constant Messages.WM_CTLCOLORSTATIC.

96.3.206 Messages.WM_CUT Constant

Pascal

```
WM_CUT = $0300;
```

C++

```
WM_CUT = 0x0300;
```

File

Messages

Description

This is constant Messages.WM_CUT.

96.3.207 Messages.WM_DDE_ACK Constant

Pascal

```
WM_DDE_ACK = WM_DDE_FIRST + 4;
```

C++

```
WM_DDE_ACK = WM_DDE_FIRST + 4;
```

File

Messages

Description

This is constant Messages.WM_DDE_ACK.

96.3.208 Messages.WM_DDE_ADVISE Constant

Pascal

```
WM_DDE_ADVISE = WM_DDE_FIRST + 2;
```

C++

```
WM_DDE_ADVISE = WM_DDE_FIRST + 2;
```

File

Messages

Description

This is constant Messages.WM_DDE_ADVISE.

96.3.209 Messages.WM_DDE_DATA Constant

Pascal

```
WM_DDE_DATA = WM_DDE_FIRST + 5;
```

C++

```
WM_DDE_DATA = WM_DDE_FIRST + 5;
```

File

Messages

Description

This is constant Messages.WM_DDE_DATA.

96.3.210 Messages.WM_DDE_EXECUTE Constant

Pascal

```
WM_DDE_EXECUTE = WM_DDE_FIRST + 8;
```

C++

```
WM_DDE_EXECUTE = WM_DDE_FIRST + 8;
```

File

Messages

Description

This is constant Messages.WM_DDE_EXECUTE.

96.3.211 Messages.WM_DDE_FIRST Constant

Pascal

```
WM_DDE_FIRST = $03E0;
```

C++

```
WM_DDE_FIRST = 0x03E0;
```

File

Messages

Description

This is constant Messages.WM_DDE_FIRST.

96.3.212 Messages.WM_DDE_INITIATE Constant

Pascal

```
WM_DDE_INITIATE = WM_DDE_FIRST + 0;
```

C++

```
WM_DDE_INITIATE = WM_DDE_FIRST + 0;
```

File

Messages

Description

This is constant Messages.WM_DDE_INITIATE.

96.3.213 Messages.WM_DDE_LAST Constant

Pascal

```
WM_DDE_LAST = WM_DDE_FIRST + 8;
```

C++

```
WM_DDE_LAST = WM_DDE_FIRST + 8;
```

File

Messages

Description

This is constant Messages.WM_DDE_LAST.

96.3.214 Messages.WM_DDE_POKE Constant

Pascal

```
WM_DDE_POKE = WM_DDE_FIRST + 7;
```

C++

```
WM_DDE_POKE = WM_DDE_FIRST + 7;
```

File

Messages

Description

This is constant Messages.WM_DDE_POKE.

96.3.215 Messages.WM_DDE_REQUEST Constant

Pascal

```
WM_DDE_REQUEST = WM_DDE_FIRST + 6;
```

C++

```
WM_DDE_REQUEST = WM_DDE_FIRST + 6;
```

File

Messages

Description

This is constant Messages.WM_DDE_REQUEST.

96.3.216 Messages.WM_DDE_TERMINATE Constant

Pascal

```
WM_DDE_TERMINATE = WM_DDE_FIRST + 1;
```

C++

```
WM_DDE_TERMINATE = WM_DDE_FIRST + 1;
```

File

Messages

Description

This is constant Messages.WM_DDE_TERMINATE.

96.3.217 Messages.WM_DDE_UNADVISE Constant

Pascal

```
WM_DDE_UNADVISE = WM_DDE_FIRST + 3;
```

C++

```
WM_DDE_UNADVISE = WM_DDE_FIRST + 3;
```

File

Messages

Description

This is constant Messages.WM_DDE_UNADVISE.

96.3.218 Messages.WM_DEADCHAR Constant

Pascal

```
WM_DEADCHAR = $0103;
```

C++

```
WM_DEADCHAR = 0x0103;
```

File

Messages

Description

This is constant Messages.WM_DEADCHAR.

96.3.219 Messages.WM_DELETEITEM Constant

Pascal

```
WM_DELETEITEM = $002D;
```

C++

```
WM_DELETEITEM = 0x002D;
```

File

Messages

Description

This is constant Messages.WM_DELETEITEM.

96.3.220 Messages.WM_DESTROY Constant

Pascal

```
WM_DESTROY = $0002;
```

C++

```
WM_DESTROY = 0x0002;
```

File

Messages

Description

This is constant Messages.WM_DESTROY.

96.3.221 Messages.WM_DESTROYCLIPBOARD Constant

Pascal

```
WM_DESTROYCLIPBOARD = $0307;
```

C++

```
WM_DESTROYCLIPBOARD = 0x0307;
```

File

Messages

Description

This is constant Messages.WM_DESTROYCLIPBOARD.

96.3.222 Messages.WM_DEVICECHANGE Constant

Pascal

```
WM_DEVICECHANGE = 537;
```

C++

```
WM_DEVICECHANGE = 537;
```

File

Messages

Description

This is constant Messages.WM_DEVICECHANGE.

96.3.223 Messages.WM_DEVMODECHANGE Constant

Pascal

```
WM_DEVMODECHANGE = $001B;
```

C++

```
WM_DEVMODECHANGE = 0x001B;
```

File

Messages

Description

This is constant Messages.WM_DEVMODECHANGE.

96.3.224 Messages.WM_DISPLAYCHANGE Constant

Pascal

```
WM_DISPLAYCHANGE = $007E;
```

C++

```
WM_DISPLAYCHANGE = 0x007E;
```

File

Messages

Description

This is constant Messages.WM_DISPLAYCHANGE.

96.3.225 Messages.WM_DRAWCLIPBOARD Constant

Pascal

```
WM_DRAWCLIPBOARD = $0308;
```

C++

```
WM_DRAWCLIPBOARD = 0x0308;
```

File

Messages

Description

This is constant Messages.WM_DRAWCLIPBOARD.

96.3.226 Messages.WM_DRAWITEM Constant

Pascal

```
WM_DRAWITEM = $002B;
```

C++

```
WM_DRAWITEM = 0x002B;
```

File

Messages

Description

This is constant Messages.WM_DRAWITEM.

96.3.227 Messages.WM_DROPFILES Constant

Pascal

```
WM_DROPFILES = $0233;
```

C++

```
WM_DROPFILES = 0x0233;
```

File

Messages

Description

This is constant Messages.WM_DROPFILES.

96.3.228 Messages.WM_DWMCOLORIZATIONCOLORCHANGED Constant

Pascal

```
WM_DWMCOLORIZATIONCOLORCHANGED = $0320;
```

C++

```
WM_DWMCOLORIZATIONCOLORCHANGED = 0x0320;
```

File

Messages

Description

This is constant Messages.WM_DWMCOLORIZATIONCOLORCHANGED.

96.3.229 Messages.WM_DWMCOMPOSITIONCHANGED Constant

Pascal

```
WM_DWMCOMPOSITIONCHANGED = $031E;
```

C++

```
WM_DWMCOMPOSITIONCHANGED = 0x031E;
```

File

Messages

Description

This is constant Messages.WM_DWMCOMPOSITIONCHANGED.

96.3.230 Messages.WM_DWMNCRENDERINGCHANGED Constant

Pascal

```
WM_DWMNCRENDERINGCHANGED = $031F;
```

C++

```
WM_DWMNCRENDERINGCHANGED = 0x031F;
```

File

Messages

Description

This is constant Messages.WM_DWMNCRENDERINGCHANGED.

96.3.231 Messages.WM_DWMWINDOWMAXIMIZEDCHANGE Constant

Pascal

```
WM_DWMWINDOWMAXIMIZEDCHANGE = $0321;
```

C++

```
WM_DWMWINDOWMAXIMIZEDCHANGE = 0x0321;
```

File

Messages

Description

This is constant Messages.WM_DWMWINDOWMAXIMIZEDCHANGE.

96.3.232 Messages.WM_ENABLE Constant

Pascal

```
WM_ENABLE = $000A;
```

C++

```
WM_ENABLE = 0x000A;
```

File

Messages

Description

This is constant Messages.WM_ENABLE.

96.3.233 Messages.WM_ENDSESSION Constant

Pascal

```
WM_ENDSESSION = $0016;
```

C++

```
WM_ENDSESSION = 0x0016;
```

File

Messages

Description

This is constant Messages.WM_ENDSESSION.

96.3.234 Messages.WM_ENTERIDLE Constant

Pascal

```
WM_ENTERIDLE = $0121;
```

C++

```
WM_ENTERIDLE = 0x0121;
```

File

Messages

Description

This is constant Messages.WM_ENTERIDLE.

96.3.235 Messages.WM_ENTERMENULOOP Constant

Pascal

```
WM_ENTERMENULOOP = $0211;
```

C++

```
WM_ENTERMENULOOP = 0x0211;
```

File

Messages

Description

This is constant Messages.WM_ENTERMENULOOP.

96.3.236 Messages.WM_ENTERSIZEMOVE Constant

Pascal

```
WM_ENTERSIZEMOVE = $0231;
```

C++

```
WM_ENTERSIZEMOVE = 0x0231;
```

File

Messages

Description

This is constant Messages.WM_ENTERSIZEMOVE.

96.3.237 Messages.WM_ERASEBKGND Constant

Pascal

```
WM_ERASEBKGND = $0014;
```

C++

```
WM_ERASEBKGND = 0x0014;
```

File

Messages

Description

This is constant Messages.WM_ERASEBKGND.

96.3.238 Messages.WM_EXITMENULOOP Constant

Pascal

```
WM_EXITMENULOOP = $0212;
```

C++

```
WM_EXITMENULOOP = 0x0212;
```

File

Messages

Description

This is constant Messages.WM_EXITMENULOOP.

96.3.239 Messages.WM_EXITSIZEMOVE Constant

Pascal

```
WM_EXITSIZEMOVE = $0232;
```

C++

```
WM_EXITSIZEMOVE = 0x0232;
```

File

Messages

Description

This is constant Messages.WM_EXITSIZEMOVE.

96.3.240 Messages.WM_FONTCHANGE Constant

Pascal

```
WM_FONTCHANGE = $001D;
```

C++

```
WM_FONTCHANGE = 0x001D;
```

File

Messages

Description

This is constant Messages.WM_FONTCHANGE.

96.3.241 Messages.WM_GETDLGCODE Constant

Pascal

```
WM_GETDLGCODE = $0087;
```

C++

```
WM_GETDLGCODE = 0x0087;
```

File

Messages

Description

This is constant Messages.WM_GETDLGCODE.

96.3.242 Messages.WM_GETFONT Constant

Pascal

```
WM_GETFONT = $0031;
```

C++

```
WM_GETFONT = 0x0031;
```

File

Messages

Description

This is constant Messages.WM_GETFONT.

96.3.243 Messages.WM_GETHOTKEY Constant

Pascal

```
WM_GETHOTKEY = $0033;
```

C++

```
WM_GETHOTKEY = 0x0033;
```

File

Messages

Description

This is constant Messages.WM_GETHOTKEY.

96.3.244 Messages.WM_GETICON Constant

Pascal

```
WM_GETICON = $007F;
```

C++

```
WM_GETICON = 0x007F;
```

File

Messages

Description

This is constant Messages.WM_GETICON.

96.3.245 Messages.WM_GETMINMAXINFO Constant

Pascal

```
WM_GETMINMAXINFO = $0024;
```

C++

```
WM_GETMINMAXINFO = 0x0024;
```

File

Messages

Description

This is constant Messages.WM_GETMINMAXINFO.

96.3.246 Messages.WM_GETOBJECT Constant

Pascal

```
WM_GETOBJECT = $003D;
```

C++

```
WM_GETOBJECT = 0x003D;
```

File

Messages

Description

This is constant Messages.WM_GETOBJECT.

96.3.247 Messages.WM_GETTEXT Constant

Pascal

```
WM_GETTEXT = $000D;
```

C++

```
WM_GETTEXT = 0x000D;
```

File

Messages

Description

This is constant Messages.WM_GETTEXT.

96.3.248 Messages.WM_GETTEXTLENGTH Constant

Pascal

```
WM_GETTEXTLENGTH = $000E;
```

C++

```
WM_GETTEXTLENGTH = 0x000E;
```

File

Messages

Description

This is constant Messages.WM_GETTEXTLENGTH.

96.3.249 Messages.WM_HANDHELDFIRST Constant

Pascal

```
WM_HANDHELDFIRST = 856;
```

C++

```
WM_HANDHELDFIRST = 856;
```

File

Messages

Description

This is constant Messages.WM_HANDHELDFIRST.

96.3.250 Messages.WM_HANDHELDLAST Constant

Pascal

```
WM_HANDHELDLAST = 863;
```

C++

```
WM_HANDHELDLAST = 863;
```

File

Messages

Description

This is constant Messages.WM_HANDHELDLAST.

96.3.251 Messages.WM_HELP Constant

Pascal

```
WM_HELP = $0053;
```

C++

```
WM_HELP = 0x0053;
```

File

Messages

Description

This is constant Messages.WM_HELP.

96.3.252 Messages.WM_HOTKEY Constant

Pascal

```
WM_HOTKEY = $0312;
```

C++

```
WM_HOTKEY = 0x0312;
```

File

Messages

Description

This is constant Messages.WM_HOTKEY.

96.3.253 Messages.WM_HSCROLL Constant

Pascal

```
WM_HSCROLL = $0114;
```

C++

```
WM_HSCROLL = 0x0114;
```

File

Messages

Description

This is constant Messages.WM_HSCROLL.

96.3.254 Messages.WM_HSCROLLCLIPBOARD Constant

Pascal

```
WM_HSCROLLCLIPBOARD = $030E;
```

C++

```
WM_HSCROLLCLIPBOARD = 0x030E;
```

File

Messages

Description

This is constant Messages.WM_HSCROLLCLIPBOARD.

96.3.255 Messages.WM_ICONERASEBKGND Constant

Pascal

```
WM_ICONERASEBKGND = $0027;
```

C++

```
WM_ICONERASEBKGND = 0x0027;
```

File

Messages

Description

This is constant Messages.WM_ICONERASEBKGND.

96.3.256 Messages.WM_IME_CHAR Constant

Pascal

```
WM_IME_CHAR = $0286;
```

C++

```
WM_IME_CHAR = 0x0286;
```

File

Messages

Description

This is constant Messages.WM_IME_CHAR.

96.3.257 Messages.WM_IME_COMPOSITION Constant

Pascal

```
WM_IME_COMPOSITION = $010F;
```

C++

```
WM_IME_COMPOSITION = 0x010F;
```

File

Messages

Description

This is constant Messages.WM_IME_COMPOSITION.

96.3.258 Messages.WM_IME_COMPOSITIONFULL Constant

Pascal

```
WM_IME_COMPOSITIONFULL = $0284;
```

C++

```
WM_IME_COMPOSITIONFULL = 0x0284;
```

File

Messages

Description

This is constant Messages.WM_IME_COMPOSITIONFULL.

96.3.259 Messages.WM_IME_CONTROL Constant

Pascal

```
WM_IME_CONTROL = $0283;
```

C++

```
WM_IME_CONTROL = 0x0283;
```

File

Messages

Description

This is constant Messages.WM_IME_CONTROL.

96.3.260 Messages.WM_IME_ENDCOMPOSITION Constant

Pascal

```
WM_IME_ENDCOMPOSITION = $010E;
```

C++

```
WM_IME_ENDCOMPOSITION = 0x010E;
```

File

Messages

Description

This is constant Messages.WM_IME_ENDCOMPOSITION.

96.3.261 Messages.WM_IME_KEYDOWN Constant

Pascal

```
WM_IME_KEYDOWN = $0290;
```

C++

```
WM_IME_KEYDOWN = 0x0290;
```

File

Messages

Description

This is constant Messages.WM_IME_KEYDOWN.

96.3.262 Messages.WM_IME_KEYLAST Constant

Pascal

```
WM_IME_KEYLAST = $010F;
```

C++

```
WM_IME_KEYLAST = 0x010F;
```

File

Messages

Description

This is constant Messages.WM_IME_KEYLAST.

96.3.263 Messages.WM_IME_KEYUP Constant

Pascal

```
WM_IME_KEYUP = $0291;
```

C++

```
WM_IME_KEYUP = 0x0291;
```

File

Messages

Description

This is constant Messages.WM_IME_KEYUP.

96.3.264 Messages.WM_IME_NOTIFY Constant

Pascal

```
WM_IME_NOTIFY = $0282;
```

C++

```
WM_IME_NOTIFY = 0x0282;
```

File

Messages

Description

This is constant Messages.WM_IME_NOTIFY.

96.3.265 Messages.WM_IME_REQUEST Constant

Pascal

```
WM_IME_REQUEST = $0288;
```

C++

```
WM_IME_REQUEST = 0x0288;
```

File

Messages

Description

This is constant Messages.WM_IME_REQUEST.

96.3.266 Messages.WM_IME_SELECT Constant

Pascal

```
WM_IME_SELECT = $0285;
```

C++

```
WM_IME_SELECT = 0x0285;
```

File

Messages

Description

This is constant Messages.WM_IME_SELECT.

96.3.267 Messages.WM_IME_SETCONTEXT Constant

Pascal

```
WM_IME_SETCONTEXT = $0281;
```

C++

```
WM_IME_SETCONTEXT = 0x0281;
```

File

Messages

Description

This is constant Messages.WM_IME_SETCONTEXT.

96.3.268 Messages.WM_IME_STARTCOMPOSITION Constant

Pascal

```
WM_IME_STARTCOMPOSITION = $010D;
```

C++

```
WM_IME_STARTCOMPOSITION = 0x010D;
```

File

Messages

Description

This is constant Messages.WM_IME_STARTCOMPOSITION.

96.3.269 Messages.WM_INITDIALOG Constant

Pascal

```
WM_INITDIALOG = $0110;
```

C++

```
WM_INITDIALOG = 0x0110;
```

File

Messages

Description

This is constant Messages.WM_INITDIALOG.

96.3.270 Messages.WM_INITMENU Constant

Pascal

```
WM_INITMENU = $0116;
```

C++

```
WM_INITMENU = 0x0116;
```

File

Messages

Description

This is constant Messages.WM_INITMENU.

96.3.271 Messages.WM_INITMENUPOPUP Constant

Pascal

```
WM_INITMENUPOPUP = $0117;
```

C++

```
WM_INITMENUPOPUP = 0x0117;
```

File

Messages

Description

This is constant Messages.WM_INITMENUPOPUP.

96.3.272 Messages.WM_INPUT Constant

Pascal

```
WM_INPUT = $00FF;
```

C++

```
WM_INPUT = 0x00FF;
```

File

Messages

Description

This is constant Messages.WM_INPUT.

96.3.273 Messages.WM_INPUTLANGCHANGE Constant

Pascal

```
WM_INPUTLANGCHANGE = $0051;
```

C++

```
WM_INPUTLANGCHANGE = 0x0051;
```

File

Messages

Description

This is constant Messages.WM_INPUTLANGCHANGE.

96.3.274 Messages.WM_INPUTLANGCHANGEREQUEST Constant

Pascal

```
WM_INPUTLANGCHANGEREQUEST = $0050;
```

C++

```
WM_INPUTLANGCHANGEREQUEST = 0x0050;
```

File

Messages

Description

This is constant Messages.WM_INPUTLANGCHANGEREQUEST.

96.3.275 Messages.WM_KEYDOWN Constant

Pascal

```
WM_KEYDOWN = $0100;
```

C++

```
WM_KEYDOWN = 0x0100;
```

File

Messages

Description

This is constant Messages.WM_KEYDOWN.

96.3.276 Messages.WM_KEYFIRST Constant

Pascal

```
WM_KEYFIRST = $0100;
```

C++

```
WM_KEYFIRST = 0x0100;
```

File

Messages

Description

This is constant Messages.WM_KEYFIRST.

96.3.277 Messages.WM_KEYLAST Constant

Pascal

```
WM_KEYLAST = $0109;
```

C++

```
WM_KEYLAST = 0x0109;
```

File

Messages

Description

This is constant Messages.WM_KEYLAST.

96.3.278 Messages.WM_KEYUP Constant

Pascal

```
WM_KEYUP = $0101;
```

C++

```
WM_KEYUP = 0x0101;
```

File

Messages

Description

This is constant Messages.WM_KEYUP.

96.3.279 Messages.WM_KILLFOCUS Constant

Pascal

```
WM_KILLFOCUS = $0008;
```

C++

```
WM_KILLFOCUS = 0x0008;
```

File

Messages

Description

This is constant Messages.WM_KILLFOCUS.

96.3.280 Messages.WM_LBUTTONDBLCLK Constant

Pascal

```
WM_LBUTTONDBLCLK = $0203;
```

C++

```
WM_LBUTTONDBLCLK = 0x0203;
```

File

Messages

Description

This is constant Messages.WM_LBUTTONDBLCLK.

96.3.281 Messages.WM_LBUTTONDOWN Constant

Pascal

```
WM_LBUTTONDOWN = $0201;
```

C++

```
WM_LBUTTONDOWN = 0x0201;
```

File

Messages

Description

This is constant Messages.WM_LBUTTONDOWN.

96.3.282 Messages.WM_LBUTTONUP Constant

Pascal

```
WM_LBUTTONUP = $0202;
```

C++

```
WM_LBUTTONUP = 0x0202;
```

File

Messages

Description

This is constant Messages.WM_LBUTTONUP.

96.3.283 Messages.WM_MBUTTONDBLCLK Constant

Pascal

```
WM_MBUTTONDBLCLK = $0209;
```

C++

```
WM_MBUTTONDBLCLK = 0x0209;
```

File

Messages

Description

This is constant Messages.WM_MBUTTONDBLCLK.

96.3.284 Messages.WM_MBUTTONDOWN Constant

Pascal

```
WM_MBUTTONDOWN = $0207;
```

C++

```
WM_MBUTTONDOWN = 0x0207;
```

File

Messages

Description

This is constant Messages.WM_MBUTTONDOWN.

96.3.285 Messages.WM_MBUTTONUP Constant

Pascal

```
WM_MBUTTONUP = $0208;
```

C++

```
WM_MBUTTONUP = 0x0208;
```

File

Messages

Description

This is constant Messages.WM_MBUTTONUP.

96.3.286 Messages.WM_MDIACTIVATE Constant

Pascal

```
WM_MDIACTIVATE = $0222;
```

C++

```
WM_MDIACTIVATE = 0x0222;
```

File

Messages

Description

This is constant Messages.WM_MDIACTIVATE.

96.3.287 Messages.WM_MDICASCADE Constant

Pascal

```
WM_MDICASCADE = $0227;
```

C++

```
WM_MDICASCADE = 0x0227;
```

File

Messages

Description

This is constant Messages.WM_MDICASCADE.

96.3.288 Messages.WM_MDICREATE Constant

Pascal

```
WM_MDICREATE = $0220;
```

C++

```
WM_MDICREATE = 0x0220;
```

File

Messages

Description

This is constant Messages.WM_MDICREATE.

96.3.289 Messages.WM_MDIDESTROY Constant

Pascal

```
WM_MDIDESTROY = $0221;
```

C++

```
WM_MDIDESTROY = 0x0221;
```

File

Messages

Description

This is constant Messages.WM_MDIDESTROY.

96.3.290 Messages.WM_MDIGETACTIVE Constant

Pascal

```
WM_MDIGETACTIVE = $0229;
```

C++

```
WM_MDIGETACTIVE = 0x0229;
```

File

Messages

Description

This is constant Messages.WM_MDIGETACTIVE.

96.3.291 Messages.WM_MDIICONARRANGE Constant

Pascal

```
WM_MDIICONARRANGE = $0228;
```

C++

```
WM_MDIICONARRANGE = 0x0228;
```

File

Messages

Description

This is constant Messages.WM_MDIICONARRANGE.

96.3.292 Messages.WM_MDIMAXIMIZE Constant

Pascal

```
WM_MDIMAXIMIZE = $0225;
```

C++

```
WM_MDIMAXIMIZE = 0x0225;
```

File

Messages

Description

This is constant Messages.WM_MDIMAXIMIZE.

96.3.293 Messages.WM_MDINEXT Constant

Pascal

```
WM_MDINEXT = $0224;
```

C++

```
WM_MDINEXT = 0x0224;
```

File

Messages

Description

This is constant Messages.WM_MDINEXT.

96.3.294 Messages.WM_MDIREFRESHMENU Constant

Pascal

```
WM_MDIREFRESHMENU = $0234;
```

C++

```
WM_MDIREFRESHMENU = 0x0234;
```

File

Messages

Description

This is constant Messages.WM_MDIREFRESHMENU.

96.3.295 Messages.WM_MDIRESTORE Constant

Pascal

```
WM_MDIRESTORE = $0223;
```

C++

```
WM_MDIRESTORE = 0x0223;
```

File

Messages

Description

This is constant Messages.WM_MDIRESTORE.

96.3.296 Messages.WM_MDISETMENU Constant

Pascal

```
WM_MDISETMENU = $0230;
```

C++

```
WM_MDISETMENU = 0x0230;
```

File

Messages

Description

This is constant Messages.WM_MDISETMENU.

96.3.297 Messages.WM_MDITILE Constant

Pascal

```
WM_MDITILE = $0226;
```

C++

```
WM_MDITILE = 0x0226;
```

File

Messages

Description

This is constant Messages.WM_MDITILE.

96.3.298 Messages.WM_MEASUREITEM Constant

Pascal

```
WM_MEASUREITEM = $002C;
```

C++

```
WM_MEASUREITEM = 0x002C;
```

File

Messages

Description

This is constant Messages.WM_MEASUREITEM.

96.3.299 Messages.WM_MENUCHAR Constant

Pascal

```
WM_MENUCHAR = $0120;
```

C++

```
WM_MENUCHAR = 0x0120;
```

File

Messages

Description

This is constant Messages.WM_MENUCHAR.

96.3.300 Messages.WM_MENUCOMMAND Constant

Pascal

```
WM_MENUCOMMAND = $0126;
```

C++

```
WM_MENUCOMMAND = 0x0126;
```

File

Messages

Description

This is constant Messages.WM_MENUCOMMAND.

96.3.301 Messages.WM_MENUDRAG Constant

Pascal

```
WM_MENUDRAG = $0123;
```

C++

```
WM_MENUDRAG = 0x0123;
```

File

Messages

Description

This is constant Messages.WM_MENUDRAG.

96.3.302 Messages.WM_MENUGETOBJECT Constant

Pascal

```
WM_MENUGETOBJECT = $0124;
```

C++

```
WM_MENUGETOBJECT = 0x0124;
```

File

Messages

Description

This is constant Messages.WM_MENUGETOBJECT.

96.3.303 Messages.WM_MENURBUTTONUP Constant

Pascal

```
WM_MENURBUTTONUP = $0122;
```

C++

```
WM_MENURBUTTONUP = 0x0122;
```

File

Messages

Description

This is constant Messages.WM_MENURBUTTONUP.

96.3.304 Messages.WM_MENUSELECT Constant

Pascal

```
WM_MENUSELECT = $011F;
```

C++

```
WM_MENUSELECT = 0x011F;
```

File

Messages

Description

This is constant Messages.WM_MENUSELECT.

96.3.305 Messages.WM_MOUSEACTIVATE Constant

Pascal

```
WM_MOUSEACTIVATE = $0021;
```

C++

```
WM_MOUSEACTIVATE = 0x0021;
```

File

Messages

Description

This is constant Messages.WM_MOUSEACTIVATE.

96.3.306 Messages.WM_MOUSEFIRST Constant

Pascal

```
WM_MOUSEFIRST = $0200;
```

C++

```
WM_MOUSEFIRST = 0x0200;
```

File

Messages

Description

This is constant Messages.WM_MOUSEFIRST.

96.3.307 Messages.WM_MOUSEHOVER Constant

Pascal

```
WM_MOUSEHOVER = $02A1;
```

C++

```
WM_MOUSEHOVER = 0x02A1;
```

File

Messages

Description

This is constant Messages.WM_MOUSEHOVER.

96.3.308 Messages.WM_MOUSELAST Constant

Pascal

```
WM_MOUSELAST = $020A;
```

C++

```
WM_MOUSELAST = 0x020A;
```

File

Messages

Description

This is constant Messages.WM_MOUSELAST.

96.3.309 Messages.WM_MOUSELEAVE Constant

Pascal

```
WM_MOUSELEAVE = $02A3;
```

C++

```
WM_MOUSELEAVE = 0x02A3;
```

File

Messages

Description

This is constant Messages.WM_MOUSELEAVE.

96.3.310 Messages.WM_MOUSEMOVE Constant

Pascal

```
WM_MOUSEMOVE = $0200;
```

C++

```
WM_MOUSEMOVE = 0x0200;
```

File

Messages

Description

This is constant Messages.WM_MOUSEMOVE.

96.3.311 Messages.WM_MOUSEWHEEL Constant

Pascal

```
WM_MOUSEWHEEL = $020A;
```

C++

```
WM_MOUSEWHEEL = 0x020A;
```

File

Messages

Description

This is constant Messages.WM_MOUSEWHEEL.

96.3.312 Messages.WM_MOVE Constant

Pascal

```
WM_MOVE = $0003;
```

C++

```
WM_MOVE = 0x0003;
```

File

Messages

Description

This is constant Messages.WM_MOVE.

96.3.313 Messages.WM_MOVING Constant

Pascal

```
WM_MOVING = 534;
```

C++

```
WM_MOVING = 534;
```

File

Messages

Description

This is constant Messages.WM_MOVING.

96.3.314 Messages.WM_NCACTIVATE Constant

Pascal

```
WM_NCACTIVATE = $0086;
```

C++

```
WM_NCACTIVATE = 0x0086;
```

File

Messages

Description

This is constant Messages.WM_NCACTIVATE.

96.3.315 Messages.WM_NCCALCSIZE Constant

Pascal

```
WM_NCCALCSIZE = $0083;
```

C++

```
WM_NCCALCSIZE = 0x0083;
```

File

Messages

Description

This is constant Messages.WM_NCCALCSIZE.

96.3.316 Messages.WM_NCCREATE Constant

Pascal

```
WM_NCCREATE = $0081;
```

C++

```
WM_NCCREATE = 0x0081;
```

File

Messages

Description

This is constant Messages.WM_NCCREATE.

96.3.317 Messages.WM_NCDESTROY Constant

Pascal

```
WM_NCDESTROY = $0082;
```

C++

```
WM_NCDESTROY = 0x0082;
```

File

Messages

Description

This is constant Messages.WM_NCDESTROY.

96.3.318 Messages.WM_NCHITTEST Constant

Pascal

```
WM_NCHITTEST = $0084;
```

C++

```
WM_NCHITTEST = 0x0084;
```

File

Messages

Description

This is constant Messages.WM_NCHITTEST.

96.3.319 Messages.WM_NCLBUTTONDBLCLK Constant

Pascal

```
WM_NCLBUTTONDBLCLK = $00A3;
```

C++

```
WM_NCLBUTTONDBLCLK = 0x00A3;
```

File

Messages

Description

This is constant Messages.WM_NCLBUTTONDBLCLK.

96.3.320 Messages.WM_NCLBUTTONDOWN Constant

Pascal

```
WM_NCLBUTTONDOWN = $00A1;
```

C++

```
WM_NCLBUTTONDOWN = 0x00A1;
```

File

Messages

Description

This is constant Messages.WM_NCLBUTTONDOWN.

96.3.321 Messages.WM_NCLBUTTONUP Constant

Pascal

```
WM_NCLBUTTONUP = $00A2;
```

C++

```
WM_NCLBUTTONUP = 0x00A2;
```

File

Messages

Description

This is constant Messages.WM_NCLBUTTONUP.

96.3.322 Messages.WM_NCMBUTTONDBLCLK Constant

Pascal

```
WM_NCMBUTTONDBLCLK = $00A9;
```

C++

```
WM_NCMBUTTONDBLCLK = 0x00A9;
```

File

Messages

Description

This is constant Messages.WM_NCMBUTTONDBLCLK.

96.3.323 Messages.WM_NCMBUTTONDOWN Constant

Pascal

```
WM_NCMBUTTONDOWN = $00A7;
```

C++

```
WM_NCMBUTTONDOWN = 0x00A7;
```

File

Messages

Description

This is constant Messages.WM_NCMBUTTONDOWN.

96.3.324 Messages.WM_NCMBUTTONUP Constant

Pascal

```
WM_NCMBUTTONUP = $00A8;
```

C++

```
WM_NCMBUTTONUP = 0x00A8;
```

File

Messages

Description

This is constant Messages.WM_NCMBUTTONUP.

96.3.325 Messages.WM_NCMOUSEHOVER Constant

Pascal

```
WM_NCMOUSEHOVER = $02A0;
```

C++

```
WM_NCMOUSEHOVER = 0x02A0;
```

File

Messages

Description

This is constant Messages.WM_NCMOUSEHOVER.

96.3.326 Messages.WM_NCMOUSELEAVE Constant

Pascal

```
WM_NCMOUSELEAVE = $02A2;
```

C++

```
WM_NCMOUSELEAVE = 0x02A2;
```

File

Messages

Description

This is constant Messages.WM_NCMOUSELEAVE.

96.3.327 Messages.WM_NCMOUSEMOVE Constant

Pascal

```
WM_NCMOUSEMOVE = $00A0;
```

C++

```
WM_NCMOUSEMOVE = 0x00A0;
```

File

Messages

Description

This is constant Messages.WM_NCMOUSEMOVE.

96.3.328 Messages.WM_NCPAINT Constant

Pascal

```
WM_NCPAINT = $0085;
```

C++

```
WM_NCPAINT = 0x0085;
```

File

Messages

Description

This is constant Messages.WM_NCPAINT.

96.3.329 Messages.WM_NCRBUTTONDBLCLK Constant

Pascal

```
WM_NCRBUTTONDBLCLK = $00A6;
```

C++

```
WM_NCRBUTTONDBLCLK = 0x00A6;
```

File

Messages

Description

This is constant Messages.WM_NCRBUTTONDBLCLK.

96.3.330 Messages.WM_NCRBUTTONDOWN Constant

Pascal

```
WM_NCRBUTTONDOWN = $00A4;
```

C++

```
WM_NCRBUTTONDOWN = 0x00A4;
```

File

Messages

Description

This is constant Messages.WM_NCRBUTTONDOWN.

96.3.331 Messages.WM_NCRBUTTONUP Constant

Pascal

```
WM_NCRBUTTONUP = $00A5;
```

C++

```
WM_NCRBUTTONUP = 0x00A5;
```

File

Messages

Description

This is constant Messages.WM_NCRBUTTONUP.

96.3.332 Messages.WM_NCXBUTTONDBLCLK Constant

Pascal

```
WM_NCXBUTTONDBLCLK = $00AD;
```

C++

```
WM_NCXBUTTONDBLCLK = 0x00AD;
```

File

Messages

Description

This is constant Messages.WM_NCXBUTTONDBLCLK.

96.3.333 Messages.WM_NCXBUTTONDOWN Constant

Pascal

```
WM_NCXBUTTONDOWN = $00AB;
```

C++

```
WM_NCXBUTTONDOWN = 0x00AB;
```

File

Messages

Description

This is constant Messages.WM_NCXBUTTONDOWN.

96.3.334 Messages.WM_NCXBUTTONUP Constant

Pascal

```
WM_NCXBUTTONUP = $00AC;
```

C++

```
WM_NCXBUTTONUP = 0x00AC;
```

File

Messages

Description

This is constant Messages.WM_NCXBUTTONUP.

96.3.335 Messages.WM_NEXTDLGCTL Constant

Pascal

```
WM_NEXTDLGCTL = $0028;
```

C++

```
WM_NEXTDLGCTL = 0x0028;
```

File

Messages

Description

This is constant Messages.WM_NEXTDLGCTL.

96.3.336 Messages.WM_NEXTMENU Constant

Pascal

```
WM_NEXTMENU = $0213;
```

C++

```
WM_NEXTMENU = 0x0213;
```

File

Messages

Description

This is constant Messages.WM_NEXTMENU.

96.3.337 Messages.WM_NOTIFY Constant

Pascal

```
WM_NOTIFY = $004E;
```

C++

```
WM_NOTIFY = 0x004E;
```

File

Messages

Description

This is constant Messages.WM_NOTIFY.

96.3.338 Messages.WM_NOTIFYFORMAT Constant

Pascal

```
WM_NOTIFYFORMAT = $0055;
```

C++

```
WM_NOTIFYFORMAT = 0x0055;
```

File

Messages

Description

This is constant Messages.WM_NOTIFYFORMAT.

96.3.339 Messages.WM_NULL Constant

Pascal

```
WM_NULL = $0000;
```

C++

```
WM_NULL = 0x0000;
```

File

Messages

Description

This is constant Messages.WM_NULL.

96.3.340 Messages.WM_PAINT Constant

Pascal

```
WM_PAINT = $000F;
```

C++

```
WM_PAINT = 0x000F;
```

File

Messages

Description

This is constant Messages.WM_PAINT.

96.3.341 Messages.WM_PAINTCLIPBOARD Constant

Pascal

```
WM_PAINTCLIPBOARD = $0309;
```

C++

```
WM_PAINTCLIPBOARD = 0x0309;
```

File

Messages

Description

This is constant Messages.WM_PAINTCLIPBOARD.

96.3.342 Messages.WM_PAINTICON Constant

Pascal

```
WM_PAINTICON = $0026;
```

C++

```
WM_PAINTICON = 0x0026;
```

File

Messages

Description

This is constant Messages.WM_PAINTICON.

96.3.343 Messages.WM_PALETTECHANGED Constant

Pascal

```
WM_PALETTECHANGED = $0311;
```

C++

```
WM_PALETTECHANGED = 0x0311;
```

File

Messages

Description

This is constant Messages.WM_PALETTECHANGED.

96.3.344 Messages.WM_PALETTEISCHANGING Constant

Pascal

```
WM_PALETTEISCHANGING = $0310;
```

C++

```
WM_PALETTEISCHANGING = 0x0310;
```

File

Messages

Description

This is constant Messages.WM_PALETTEISCHANGING.

96.3.345 Messages.WM_PARENTNOTIFY Constant

Pascal

```
WM_PARENTNOTIFY = $0210;
```

C++

```
WM_PARENTNOTIFY = 0x0210;
```

File

Messages

Description

This is constant Messages.WM_PARENTNOTIFY.

96.3.346 Messages.WM_PASTE Constant

Pascal

```
WM_PASTE = $0302;
```

C++

```
WM_PASTE = 0x0302;
```

File

Messages

Description

This is constant Messages.WM_PASTE.

96.3.347 Messages.WM_PENWINFIRST Constant

Pascal

```
WM_PENWINFIRST = $0380;
```

C++

```
WM_PENWINFIRST = 0x0380;
```

File

Messages

Description

This is constant Messages.WM_PENWINFIRST.

96.3.348 Messages.WM_PENWINLAST Constant

Pascal

```
WM_PENWINLAST = $038F;
```

C++

```
WM_PENWINLAST = 0x038F;
```

File

Messages

Description

This is constant Messages.WM_PENWINLAST.

96.3.349 Messages.WM_POWER Constant

Pascal

```
WM_POWER = $0048;
```

C++

```
WM_POWER = 0x0048;
```

File

Messages

Description

This is constant Messages.WM_POWER.

96.3.350 Messages.WM_POWERBROADCAST Constant

Pascal

```
WM_POWERBROADCAST = 536;
```

C++

```
WM_POWERBROADCAST = 536;
```

File

Messages

Description

This is constant Messages.WM_POWERBROADCAST.

96.3.351 Messages.WM_PRINT Constant

Pascal

```
WM_PRINT = 791;
```

C++

```
WM_PRINT = 791;
```

File

Messages

Description

This is constant Messages.WM_PRINT.

96.3.352 Messages.WM_PRINTCLIENT Constant

Pascal

```
WM_PRINTCLIENT = 792;
```

C++

```
WM_PRINTCLIENT = 792;
```

File

Messages

Description

This is constant Messages.WM_PRINTCLIENT.

96.3.353 Messages.WM_QUERYDRAGICON Constant

Pascal

```
WM_QUERYDRAGICON = $0037;
```

C++

```
WM_QUERYDRAGICON = 0x0037;
```

File

Messages

Description

This is constant Messages.WM_QUERYDRAGICON.

96.3.354 Messages.WM_QUERYENDSESSION Constant

Pascal

```
WM_QUERYENDSESSION = $0011;
```

C++

```
WM_QUERYENDSESSION = 0x0011;
```

File

Messages

Description

This is constant Messages.WM_QUERYENDSESSION.

96.3.355 Messages.WM_QUERYNEWPALETTE Constant

Pascal

```
WM_QUERYNEWPALETTE = $030F;
```

C++

```
WM_QUERYNEWPALETTE = 0x030F;
```

File

Messages

Description

This is constant Messages.WM_QUERYNEWPALETTE.

96.3.356 Messages.WM_QUERYOPEN Constant

Pascal

```
WM_QUERYOPEN = $0013;
```

C++

```
WM_QUERYOPEN = 0x0013;
```

File

Messages

Description

This is constant Messages.WM_QUERYOPEN.

96.3.357 Messages.WM_QUERYUISTATE Constant

Pascal

```
WM_QUERYUISTATE = $0129;
```

C++

```
WM_QUERYUISTATE = 0x0129;
```

File

Messages

Description

This is constant Messages.WM_QUERYUISTATE.

96.3.358 Messages.WM_QUEUESYNC Constant

Pascal

```
WM_QUEUESYNC = $0023;
```

C++

```
WM_QUEUESYNC = 0x0023;
```

File

Messages

Description

This is constant Messages.WM_QUEUESYNC.

96.3.359 Messages.WM_QUIT Constant

Pascal

```
WM_QUIT = $0012;
```

C++

```
WM_QUIT = 0x0012;
```

File

Messages

Description

This is constant Messages.WM_QUIT.

96.3.360 Messages.WM_RBUTTONDBLCLK Constant

Pascal

```
WM_RBUTTONDBLCLK = $0206;
```

C++

```
WM_RBUTTONDBLCLK = 0x0206;
```

File

Messages

Description

This is constant Messages.WM_RBUTTONDBLCLK.

96.3.361 Messages.WM_RBUTTONDOWN Constant

Pascal

```
WM_RBUTTONDOWN = $0204;
```

C++

```
WM_RBUTTONDOWN = 0x0204;
```

File

Messages

Description

This is constant Messages.WM_RBUTTONDOWN.

96.3.362 Messages.WM_RBUTTONUP Constant

Pascal

```
WM_RBUTTONUP = $0205;
```

C++

```
WM_RBUTTONUP = 0x0205;
```

File

Messages

Description

This is constant Messages.WM_RBUTTONUP.

96.3.363 Messages.WM_RENDERALLFORMATS Constant

Pascal

```
WM_RENDERALLFORMATS = $0306;
```

C++

```
WM_RENDERALLFORMATS = 0x0306;
```

File

Messages

Description

This is constant Messages.WM_RENDERALLFORMATS.

96.3.364 Messages.WM_RENDERFORMAT Constant

Pascal

```
WM_RENDERFORMAT = $0305;
```

C++

```
WM_RENDERFORMAT = 0x0305;
```

File

Messages

Description

This is constant Messages.WM_RENDERFORMAT.

96.3.365 Messages.WM_SETCURSOR Constant

Pascal

```
WM_SETCURSOR = $0020;
```

C++

```
WM_SETCURSOR = 0x0020;
```

File

Messages

Description

This is constant Messages.WM_SETCURSOR.

96.3.366 Messages.WM_SETFOCUS Constant

Pascal

```
WM_SETFOCUS = $0007;
```

C++

```
WM_SETFOCUS = 0x0007;
```

File

Messages

Description

This is constant Messages.WM_SETFOCUS.

96.3.367 Messages.WM_SETFONT Constant

Pascal

```
WM_SETFONT = $0030;
```

C++

```
WM_SETFONT = 0x0030;
```

File

Messages

Description

This is constant Messages.WM_SETFONT.

96.3.368 Messages.WM_SETHOTKEY Constant

Pascal

```
WM_SETHOTKEY = $0032;
```

C++

```
WM_SETHOTKEY = 0x0032;
```

File

Messages

Description

This is constant Messages.WM_SETHOTKEY.

96.3.369 Messages.WM_SETICON Constant

Pascal

```
WM_SETICON = $0080;
```

C++

```
WM_SETICON = 0x0080;
```

File

Messages

Description

This is constant Messages.WM_SETICON.

96.3.370 Messages.WM_SETREDRAW Constant

Pascal

```
WM_SETREDRAW = $000B;
```

C++

```
WM_SETREDRAW = 0x000B;
```

File

Messages

Description

This is constant Messages.WM_SETREDRAW.

96.3.371 Messages.WM_SETTEXT Constant

Pascal

```
WM_SETTEXT = $000C;
```

C++

```
WM_SETTEXT = 0x000C;
```

File

Messages

Description

This is constant Messages.WM_SETTEXT.

96.3.372 Messages.WM_SETTINGCHANGE Constant

Pascal

```
WM_SETTINGCHANGE = WM_WININICHANGE;
```

C++

```
WM_SETTINGCHANGE = WM_WININICHANGE;
```

File

Messages

Description

This is constant Messages.WM_SETTINGCHANGE.

96.3.373 Messages.WM_SHOWWINDOW Constant

Pascal

```
WM_SHOWWINDOW = $0018;
```

C++

```
WM_SHOWWINDOW = 0x0018;
```

File

Messages

Description

This is constant Messages.WM_SHOWWINDOW.

96.3.374 Messages.WM_SIZE Constant

Pascal

```
WM_SIZE = $0005;
```

C++

```
WM_SIZE = 0x0005;
```

File

Messages

Description

This is constant Messages.WM_SIZE.

96.3.375 Messages.WM_SIZECLIPBOARD Constant

Pascal

```
WM_SIZECLIPBOARD = $030B;
```

C++

```
WM_SIZECLIPBOARD = 0x030B;
```

File

Messages

Description

This is constant Messages.WM_SIZECLIPBOARD.

96.3.376 Messages.WM_SIZING Constant

Pascal

```
WM_SIZING = 532;
```

C++

```
WM_SIZING = 532;
```

File

Messages

Description

This is constant Messages.WM_SIZING.

96.3.377 Messages.WM_SPOOLERSTATUS Constant

Pascal

```
WM_SPOOLERSTATUS = $002A;
```

C++

```
WM_SPOOLERSTATUS = 0x002A;
```

File

Messages

Description

This is constant Messages.WM_SPOOLERSTATUS.

96.3.378 Messages.WM_STYLECHANGED Constant

Pascal

```
WM_STYLECHANGED = $007D;
```

C++

```
WM_STYLECHANGED = 0x007D;
```

File

Messages

Description

This is constant Messages.WM_STYLECHANGED.

96.3.379 Messages.WM_STYLECHANGING Constant

Pascal

```
WM_STYLECHANGING = $007C;
```

C++

```
WM_STYLECHANGING = 0x007C;
```

File

Messages

Description

This is constant Messages.WM_STYLECHANGING.

96.3.380 Messages.WM_SYSCHAR Constant

Pascal

```
WM_SYSCHAR = $0106;
```

C++

```
WM_SYSCHAR = 0x0106;
```

File

Messages

Description

This is constant Messages.WM_SYSCHAR.

96.3.381 Messages.WM_SYSCOLORCHANGE Constant

Pascal

```
WM_SYSCOLORCHANGE = $0015;
```

C++

```
WM_SYSCOLORCHANGE = 0x0015;
```

File

Messages

Description

This is constant Messages.WM_SYSCOLORCHANGE.

96.3.382 Messages.WM_SYSCOMMAND Constant

Pascal

```
WM_SYSCOMMAND = $0112;
```

C++

```
WM_SYSCOMMAND = 0x0112;
```

File

Messages

Description

This is constant Messages.WM_SYSCOMMAND.

96.3.383 Messages.WM_SYSDEADCHAR Constant

Pascal

```
WM_SYSDEADCHAR = $0107;
```

C++

```
WM_SYSDEADCHAR = 0x0107;
```

File

Messages

Description

This is constant Messages.WM_SYSDEADCHAR.

96.3.384 Messages.WM_SYSKEYDOWN Constant

Pascal

```
WM_SYSKEYDOWN = $0104;
```

C++

```
WM_SYSKEYDOWN = 0x0104;
```

File

Messages

Description

This is constant Messages.WM_SYSKEYDOWN.

96.3.385 Messages.WM_SYSKEYUP Constant

Pascal

```
WM_SYSKEYUP = $0105;
```

C++

```
WM_SYSKEYUP = 0x0105;
```

File

Messages

Description

This is constant Messages.WM_SYSKEYUP.

96.3.386 Messages.WM_SYSTEMERROR Constant

Pascal

```
WM_SYSTEMERROR = $0017;
```

C++

```
WM_SYSTEMERROR = 0x0017;
```

File

Messages

Description

This is constant Messages.WM_SYSTEMERROR.

96.3.387 Messages.WM_TABLET_FIRST Constant

Pascal

```
WM_TABLET_FIRST = $02C0;
```

C++

```
WM_TABLET_FIRST = 0x02C0;
```

File

Messages

Description

This is constant Messages.WM_TABLET_FIRST.

96.3.388 Messages.WM_TABLET_LAST Constant

Pascal

```
WM_TABLET_LAST = $02DF;
```

C++

```
WM_TABLET_LAST = 0x02DF;
```

File

Messages

Description

This is constant Messages.WM_TABLET_LAST.

96.3.389 Messages.WM_TCARD Constant

Pascal

```
WM_TCARD = $0052;
```

C++

```
WM_TCARD = 0x0052;
```

File

Messages

Description

This is constant Messages.WM_TCARD.

96.3.390 Messages.WM_THEMECHANGED Constant

Pascal

```
WM_THEMECHANGED = $031A;
```

C++

```
WM_THEMECHANGED = 0x031A;
```

File

Messages

Description

This is constant Messages.WM_THEMECHANGED.

96.3.391 Messages.WM_TIMECHANGE Constant

Pascal

```
WM_TIMECHANGE = $001E;
```

C++

```
WM_TIMECHANGE = 0x001E;
```

File

Messages

Description

This is constant Messages.WM_TIMECHANGE.

96.3.392 Messages.WM_TIMER Constant

Pascal

```
WM_TIMER = $0113;
```

C++

```
WM_TIMER = 0x0113;
```

File

Messages

Description

This is constant Messages.WM_TIMER.

96.3.393 Messages.WM_UNDO Constant

Pascal

```
WM_UNDO = $0304;
```

C++

```
WM_UNDO = 0x0304;
```

File

Messages

Description

This is constant Messages.WM_UNDO.

96.3.394 Messages.WM_UNICHAR Constant

Pascal

```
WM_UNICHAR = $0109;
```

C++

```
WM_UNICHAR = 0x0109;
```

File

Messages

Description

This is constant Messages.WM_UNICHAR.

96.3.395 Messages.WM_UNINITMENUPOPUP Constant

Pascal

```
WM_UNINITMENUPOPUP = $0125;
```

C++

```
WM_UNINITMENUPOPUP = 0x0125;
```

File

Messages

Description

This is constant Messages.WM_UNINITMENUPOPUP.

96.3.396 Messages.WM_UPDATEUISTATE Constant

Pascal

```
WM_UPDATEUISTATE = $0128;
```

C++

```
WM_UPDATEUISTATE = 0x0128;
```

File

Messages

Description

This is constant Messages.WM_UPDATEUISTATE.

96.3.397 Messages.WM_USER Constant

Pascal

```
WM_USER = $0400;
```

C++

```
WM_USER = 0x0400;
```

File

Messages

Description

This is constant Messages.WM_USER.

96.3.398 Messages.WM_USERCHANGED Constant

Pascal

```
WM_USERCHANGED = $0054;
```

C++

```
WM_USERCHANGED = 0x0054;
```

File

Messages

Description

This is constant Messages.WM_USERCHANGED.

96.3.399 Messages.WM_VKEYTOITEM Constant

Pascal

```
WM_VKEYTOITEM = $002E;
```

C++

```
WM_VKEYTOITEM = 0x002E;
```

File

Messages

Description

This is constant Messages.WM_VKEYTOITEM.

96.3.400 Messages.WM_VSCROLL Constant

Pascal

```
WM_VSCROLL = $0115;
```

C++

```
WM_VSCROLL = 0x0115;
```

File

Messages

Description

This is constant Messages.WM_VSCROLL.

96.3.401 Messages.WM_VSCROLLCLIPBOARD Constant

Pascal

```
WM_VSCROLLCLIPBOARD = $030A;
```

C++

```
WM_VSCROLLCLIPBOARD = 0x030A;
```

File

Messages

Description

This is constant Messages.WM_VSCROLLCLIPBOARD.

96.3.402 Messages.WM_WINDOWPOSCHANGED Constant

Pascal

```
WM_WINDOWPOSCHANGED = $0047;
```

C++

```
WM_WINDOWPOSCHANGED = 0x0047;
```

File

Messages

Description

This is constant Messages.WM_WINDOWPOSCHANGED.

96.3.403 Messages.WM_WINDOWPOSCHANGING Constant

Pascal

```
WM_WINDOWPOSCHANGING = $0046;
```

C++

```
WM_WINDOWPOSCHANGING = 0x0046;
```

File

Messages

Description

This is constant Messages.WM_WINDOWPOSCHANGING.

96.3.404 Messages.WM_WININICHANGE Constant

Pascal

```
WM_WININICHANGE = $001A;
```

C++

```
WM_WININICHANGE = 0x001A;
```

File

Messages

Description

This is constant Messages.WM_WININICHANGE.

96.3.405 Messages.WM_WTSSESSION_CHANGE Constant

Pascal

```
WM_WTSSESSION_CHANGE = $02B1;
```

C++

```
WM_WTSSESSION_CHANGE = 0x02B1;
```

File

Messages

Description

This is constant Messages.WM_WTSSESSION_CHANGE.

97 Midas Namespace

This is namespace Midas.

Topics

Name	Description
Interfaces (see page 16450)	The following table lists interfaces in this documentation.
Constants (see page 16460)	The following table lists constants in this documentation.

Constants

Name	Description
IID_IAppServer (see page 16460)	This is constant Midas.IID_IAppServer.
LIBID_Midas (see page 16460)	This is constant Midas.LIBID_Midas.

Interfaces

	Name	Description
	IAppServer (see page 16450)	IAppServer is the interface that client datasets use to communicate with a dataset provider.
	IAppServerDisp (see page 16457)	This is class Midas.IAppServerDisp.

97.1 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAppServer (see page 16450)	IAppServer is the interface that client datasets use to communicate with a dataset provider.
	IAppServerDisp (see page 16457)	This is class Midas.IAppServerDisp.

97.1.1 IAppServer Interface

IAppServer is the interface that client datasets use to communicate with a dataset provider.

Class Hierarchy

Pascal

```
IAppServer = interface(IDispatch);
```

C++

```
__interface IAppServer : public IDispatch;
```

File

Midas

Description

Client datasets use an IAppServer interface to implement many of their properties and methods. This interface provides the basis of communication between the client dataset and a provider component from which it receives data and to which it applies updates.

If the provider is in the same application as the client dataset, the IAppServer interface is implemented by a hidden object that is created to manage the communication between the client dataset and the provider. In multi-tiered applications, client datasets obtain an IAppServer instance from a connection component that resides in the client application.

When client datasets use IAppServer to talk to a provider component, they pass the name of the desired provider to all method calls. By default, the interface is stateless, meaning that each method call functions independently, with no reliance on previous method calls. Because of this, IAppServer has no properties, which would represent persistent state information. However, there are cases where the client dataset relies on state information, such as the current database cursor position when performing incremental fetching using the AS_GetRecords (see page 16455) method. This state information is preserved only if multiple clients do not share an application server that implements IAppServer.

Note: Component Method declarations that use IAppServer use the _di_IAppServer type instead. This type is a DelphiInterface wrapper around the IAppServer interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IAppServer >  _di_IAppServer;
```

See Also

TClientDataSet (see page 8289)

TDataSetProvider (see page 18508)

TCustomClientDataSet (see page 8373)

Communicating with Providers Using Custom Events

Communicating with the Client Dataset

97.1.1.1 IAppServer Members

The following tables list the members exposed by IAppServer.

Public Methods

	Name	Description
	AS_ApplyUpdates (see page 16451)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 16452)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 16453)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 16454)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 16454)	Returns a list of all the available providers on a remote data module.
	AS_GetRecords (see page 16455)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 16456)	Returns information from a specified record of the provider's dataset.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.

97.1.1.2 IAppServer Methods

The methods of the IAppServer class are listed here.

Public Methods

	Name	Description
	AS_ApplyUpdates (see page 16451)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 16452)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 16453)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 16454)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 16454)	Returns a list of all the available providers on a remote data module.
	AS_GetRecords (see page 16455)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 16456)	Returns information from a specified record of the provider's dataset.

97.1.1.2.1 IAppServer.AS_ApplyUpdates Method

Applies updates received from a client dataset using a specified provider.

Pascal

```
function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_ApplyUpdates(const BSTR ProviderName, OleVariant Delta, int
```

```
MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

AS_AS_ApplyUpdates writes client updates to a dataset or database server.

ProviderName specifies the name of the provider component that manages the update operation.

Delta is a Variant containing the updated, inserted, and deleted records to write to the database.

MaxErrors indicates the maximum number of errors to permit before stopping the update operation.

ErrorCount returns the actual number of errors encountered during the update operation.

OwnerData is information supplied by the client application in a BeforeAS_ApplyUpdates event handler and returned to the client application in an AfterAS_ApplyUpdates event handler.

AS_AS_ApplyUpdates (Delphi) or AS_ApplyUpdates_result parameter (C++) returns a Variant that is a data packet containing all records that could not be applied to the database.

See Also

ApplyUpdates (see page 8394)

applyupdates (see page 18459)

BeforeApplyUpdates (see page 18479)

AfterApplyUpdates (see page 18475)

BeforeApplyUpdates (see page 8511)

AfterApplyUpdates (see page 8497)

AS_ApplyUpdates (see page 6590)

as_applyupdates (see page 19892)

Communicating with the Client Dataset

97.1.1.2.2 IAppServer.AS_DataRequest Method

Generates an OnAS_DataRequest event on the specified provider component.

Pascal

```
function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

Use AS_AS_DataRequest to invoke an OnAS_DataRequest event handler of the provider named by ProviderName. This method is provided to allow application developers to customize the communication between a client application and a provider component. There is no predefined meaning for the Data parameter or the return value (Delphi) or the AS_DataRequest_result parameter (C++), which is returned to the caller.

See Also

OnDataRequest (see page 18482)

DataRequest (see page 8404)

Communicating with the Client Dataset

97.1.1.2.3 IAppServer.AS_Execute Method

AS_Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); safecall;
```

C++

```
__fastcall AS_Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant Params,
OleVariant OwnerData);
```

Description

Use AS_AS_Execute to pass an execute command to a remote provider, using a specified set of parameters. AS_AS_Execute calls the AS_Execute method of the specified provider after assigning any parameters. Output parameters that result from executing a query or stored procedure are returned in the Params parameter.

The provider's AS_Execute method generates a BeforeAS_Execute event, passes the command on to its associated dataset, and generates an AfterAS_Execute event. The dataset must represent a query or stored procedure that does not return a cursor.

The ProviderName parameter specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the query, or the name of a stored procedure that replaces the dataset's stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

The Params parameter encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams routine. To decode Params into a TParams object, use the Unpackparams routine.

The OwnerData parameter contains custom information that appears as an argument to the provider's BeforeAS_Execute and AfterAS_Execute event handlers. This information originates in a client dataset's BeforeAS_Execute event handler. The value returned in OwnerData is passed to the client dataset's AfterAS_Execute event handler.

See Also

DataSet (see page 18529)

beforeexecute (see page 18479)

AfterExecute (see page 18476)

Params (see page 8489)

Execute (see page 8413)

Execute (see page 18466)

BeforeExecute (see page 8514)

AfterExecute (see page 8504)

Options (see page 18440)

PackageParams (see page 8534)

UnpackParams (see page 8534)

Communicating with the Client Dataset

97.1.1.2.4 IAppServer.AS_GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

Use AS_AS_GetParams to retrieve the current parameter values from the dataset associated with a provider.

The ProviderName parameter specifies the name of the provider component whose dataset has the parameters.

The OwnerData represents custom information that originates in a client dataset's BeforeAS_GetParams event handler and returns information that is passed to the client dataset's AfterAS_GetParams event handler.

AS_GetParams_result (C++) or AS_AS_GetParams (Delphi) returns the parameters, encoded as a Variant array, where each element is a 2-element Variant array containing a parameter name and a parameter value. These parameters can be converted to a TParams object using the UnpackParams procedure. If the dataset has no parameters, or if the provider does not support parameter fetching, AS_GetParams_result (C++) or AS_AS_GetParams (Delphi) returns a Null Variant.

Note: AS_AS_GetParams should not be used to retrieve output parameters from stateless application servers, because parameter values may be changed by other applications. When writing a stateless application server, retrieve parameter values using the AS_Execute (see page 16453) method instead.

See Also

DataSet (see page 18529)

Params (see page 8489)

FetchParams (see page 8415)

GetParams (see page 18467)

BeforeGetParams (see page 18480)

aftergetparams (see page 18477)

AS_Execute (see page 16453)

UnpackParams (see page 8534)

Communicating with the Client Dataset

97.1.1.2.5 IAppServer.AS_GetProviderNames Method

Returns a list of all the available providers on a remote data module.

Pascal

```
function AS_GetProviderNames: OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetProviderNames();
```

Description

Call AS_AS_GetProviderNames for a list of all providers available from a remote data module. These values indicate the

possible values for the ProviderName property of a client dataset that uses the IAppServer (see page 16450) interface of an application server. When calling any other IAppServer (see page 16450) method, pass one of these names to indicate the provider that is the target of the method call.

AS_AS_GetProviderNames returns the names of the providers in a Variant array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

See Also

providers (see page 6599)

ProviderName (see page 8492)

Providers (see page 19908)

Communicating with the Client Dataset

97.1.1.2.6 IAppServer.AS_GetRecords Method

Returns a data packet that contains the specified data.

Pascal

```
function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetRecords(const BSTR ProviderName, int Count, int RecsOut, int
Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

Call AS_AS_GetRecords to retrieve a specified number of records. AS_GetRecords returns the requested records, starting with the current record of the provider's dataset. When working with a stateless remote data module, you may need to reposition the cursor or re-execute a query or stored procedure in the provider's BeforeAS_GetRecords event handler.

ProviderName gives the name of the provider component that provides the records.

Count indicates the number or type of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only Count records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants. For example, the following value requests metadata along with a data packet encoded in XML:

```
MetaDataOption + XMLOption
MetaDataOption || XMLOption
```

CommandText specifies an optional SQL statement that replaces the SQL of a query, or the name of a table or stored procedure that replaces the provider's table or stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

Params is a Variant containing any parameters that should be passed to the provider's dataset before it executes to generate the requested data. It returns any output parameters. You can convert between the Variant format for parameters used by this argument and a TParams object using the global PackageParams and UnpackParams routines.

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_GetRecords event handler. This information is passed to the provider's BeforeAS_GetRecords event handler. OwnerData returns information supplied by the provider's AfterAS_GetRecords event handler.

Records are returned as a data packet in a Variantparameter, AS_GetRecords_result (C++).

See Also

Data (see page 8467)

Params (see page 8489)

GetNextPacket (see page 8423)

GetRecords (see page 18468)

BeforeGetRecords (see page 8515)

BeforeGetRecords (see page 18480)

AfterGetRecords (see page 18477)

Options (see page 18440)

PackageParams (see page 8534)

UnpackParams (see page 8534)

Communicating with the Client Dataset

97.1.1.2.7 IAppServer.AS_RowRequest Method

Returns information from a specified record of the provider's dataset.

Pascal

```
function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_RowRequest(const BSTR ProviderName, OleVariant Row, int RequestType,
OleVariant OwnerData);
```

Description

Client datasets call AS_AS_RowRequest automatically to implement their FetchBlobs, FetchDetails , or RefreshRecord method.

The ProviderName parameter indicates the provider associated with the dataset from which information should be fetched.

The Row parameter is an OleVariant that describes the current record on the client dataset.

The RequestType parameter indicates the type of information required. It is an integer version of the TFetchOptions type. (An integer because the value may be marshaled to a remote application server). To create a value for RequestType, take the corresponding TFetchOptions value, cast it to a Byte, and cast the result to an Integer:

```
Integer(Byte([foBlobs, foDetails]));
(int) (Byte) (TFetchOptions() << foBlobs << foDetails);
```

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_RowRequest event handler. This information is passed to the provider's BeforeAS_RowRequest event handler. OwnerData returns custom information supplied by the provider's AfterAS_RowRequest event handler.

The requested data is returned as a delta packet in AS_RowRequest_result (C++).

See Also

RowRequest (see page 18472)

BeforeRowRequest (see page 18482)

AfterRowRequest (see page 18478)

FetchBlobs (see page 8414)

FetchDetails (see page 8414)

RefreshRecord (see page 8442)

Communicating with the Client Dataset

97.1.2 IAppServerDisp Interface

Class Hierarchy

Midas.IAppServerDisp

Pascal

```
IAppServerDisp = dispinterface;
```

C++

```
class IAppServerDisp;
```

File

Midas

Description

This is class Midas.IAppServerDisp.

97.1.2.1 IAppServerDisp Members

The following tables list the members exposed by IAppServerDisp.

Public Methods

	Name	Description
	AS_ApplyUpdates (see page 16458)	This is AS_ApplyUpdates, a member of class IAppServerDisp.
	AS_DataRequest (see page 16458)	This is AS_DataRequest, a member of class IAppServerDisp.
	AS_Execute (see page 16458)	This is AS_Execute, a member of class IAppServerDisp.
	AS_GetParams (see page 16458)	This is AS_GetParams, a member of class IAppServerDisp.
	AS_GetProviderNames (see page 16459)	This is AS_GetProviderNames, a member of class IAppServerDisp.
	AS_GetRecords (see page 16459)	This is AS_GetRecords, a member of class IAppServerDisp.
	AS_RowRequest (see page 16459)	This is AS_RowRequest, a member of class IAppServerDisp.

97.1.2.2 IAppServerDisp Methods

The methods of the IAppServerDisp class are listed here.

Public Methods

	Name	Description
	AS_ApplyUpdates (see page 16458)	This is AS_ApplyUpdates, a member of class IAppServerDisp.
	AS_DataRequest (see page 16458)	This is AS_DataRequest, a member of class IAppServerDisp.
	AS_Execute (see page 16458)	This is AS_Execute, a member of class IAppServerDisp.
	AS_GetParams (see page 16458)	This is AS_GetParams, a member of class IAppServerDisp.
	AS_GetProviderNames (see page 16459)	This is AS_GetProviderNames, a member of class IAppServerDisp.

	AS_GetRecords (see page 16459)	This is AS_GetRecords, a member of class IAppServerDisp.
	AS_RowRequest (see page 16459)	This is AS_RowRequest, a member of class IAppServerDisp.

97.1.2.2.1 IAppServerDisp.AS_ApplyUpdates Method

Pascal

```
function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant AS_ApplyUpdates(const BSTR ProviderName, OleVariant Delta, int
MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

This is AS_ApplyUpdates, a member of class IAppServerDisp.

97.1.2.2.2 IAppServerDisp.AS_DataRequest Method

Pascal

```
function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant AS_DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

This is AS_DataRequest, a member of class IAppServerDisp.

97.1.2.2.3 IAppServerDisp.AS_Execute Method

Pascal

```
procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant);
```

C++

```
__fastcall AS_Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant Params,
OleVariant OwnerData);
```

Description

This is AS_Execute, a member of class IAppServerDisp.

97.1.2.2.4 IAppServerDisp.AS_GetParams Method

Pascal

```
function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant AS_GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

This is AS_GetParams, a member of class IAppServerDisp.

97.1.2.2.5 IAppServerDisp.AS_GetProviderNames Method

Pascal

```
function AS_GetProviderNames: OleVariant;
```

C++

```
__fastcall OleVariant AS_GetProviderNames();
```

Description

This is AS_GetProviderNames, a member of class IAppServerDisp.

97.1.2.2.6 IAppServerDisp.AS_GetRecords Method

Pascal

```
function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant AS_GetRecords(const BSTR ProviderName, int Count, int RecsOut, int
Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

This is AS_GetRecords, a member of class IAppServerDisp.

97.1.2.2.7 IAppServerDisp.AS_RowRequest Method

Pascal

```
function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant AS_RowRequest(const BSTR ProviderName, OleVariant Row, int RequestType,
OleVariant OwnerData);
```

Description

This is AS_RowRequest, a member of class IAppServerDisp.

97.2 Constants

The following table lists constants in this documentation.

Constants

Name	Description
IID_IAppServer (see page 16460)	This is constant Midas.IID_IAppServer.
LIBID_Midas (see page 16460)	This is constant Midas.LIBID_Midas.

97.2.1 Midas.IID_IAppServer Constant

Pascal

```
IID_IAppServer: TGUID = '{1AEFCC20-7A24-11D2-98B0-C69BEB4B5B6D}';
```

C++

```
TGUID IID_IAppServer = '{1AEFCC20-7A24-11D2-98B0-C69BEB4B5B6D}';
```

File

Midas

Description

This is constant Midas.IID_IAppServer.

97.2.2 Midas.LIBID_Midas Constant

Pascal

```
LIBID_Midas: TGUID = '{83F57D68-CA9A-11D2-9088-00C04FA35CFA}';
```

C++

```
TGUID LIBID_Midas = '{83F57D68-CA9A-11D2-9088-00C04FA35CFA}';
```

File

Midas

Description

This is constant Midas.LIBID_Midas.

98 MidasCon Namespace

This is namespace MidasCon.

Topics

Name	Description
Classes (see page 16462)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 16472)	The following table lists structs, records, enums in this documentation.

Classes

	Name	Description
	TMidasConnection (see page 16462)	TMidasConnection handles a server connection for the client of a multi-tiered application.
	TRemoteServer (see page 16470)	TRemoteServer handles a DCOM connection to a remote server in a client application that is part of a multi-tiered application.

Structs, Records, Enums

	Name	Description
	TConnectType (see page 16472)	This is record MidasCon.TConnectType.

98.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TMidasConnection (see page 16462)	TMidasConnection handles a server connection for the client of a multi-tiered application.
	TRemoteServer (see page 16470)	TRemoteServer handles a DCOM connection to a remote server in a client application that is part of a multi-tiered application.

98.1.1 TMidasConnection Class

TMidasConnection handles a server connection for the client of a multi-tiered application.

Class Hierarchy

Pascal

```
TMidasConnection = class(TDCOMConnection);
```

C++

```
class TMidasConnection : public TDCOMConnection;
```

File

MidasCon

Description

TMidasConnection is provided for backward compatibility. When writing client applications that connect to application servers using DCOM, use TDCOMConnection instead. When writing client applications that connect to application servers using sockets, use TSocketConnection instead. When writing client applications that connect to application servers using HTTP, use TWebConnection instead.

See Also

TClientDataSet (see page 8289)

TDCOMConnection (see page 16008)

TSocketConnection (see page 18928)

TWebConnection (see page 18968)

98.1.1.1 TMidasConnection Members

The following tables list the members exposed by TMidasConnection.

Protected Events

	Name	Description
	OnGetUsername (see page 8533)	This is OnGetUsername, a member of class TCustomRemoteServer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoConnect (see page 16467)	Connects to an application server.
V	DoDisconnect (see page 16468)	Drops a connection to an application server.
V	GetAppServer (see page 8530)	Returns the value of the AppServer (see page 8532) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetConnected (see page 6865)	Returns the value of the Connected (see page 6868) property.
V	GetConnected (see page 16023)	Returns the value of the Connected (see page 16026) property.
V	GetDataSet (see page 6865)	Returns a dataset, given its index in the DataSets (see page 6870) property array.
V	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetServerCLSID (see page 16024)	Returns the GUID for the application server.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 6866)	Initializes the connection component after all objects in the form or data module have been loaded.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16025)	Responds when other objects are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	RegisterClient (see page 6866)	Registers an object that uses the connection component.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAppServer (see page 8531)	Sets the AppServer (see page 8532) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetConnected (see page 6867)	Changes the value of the Connected (see page 6868) property.
V	SetConnected (see page 16001)	Sets the Connected (see page 16026) property.

	SetConnected (see page 16025)	Sets the Connected (see page 16026) property.
	SetConnectType (see page 16468)	This is SetConnectType, a member of class TMidasConnection.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetUseBroker (see page 16468)	This is SetUseBroker, a member of class TMidasConnection.
	UnRegisterClient (see page 6868)	Removes a registered client so that it is no longer associated with the connection component.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ObjectBroker (see page 16028)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.

Public Events

	Name	Description
	AfterConnect (see page 6874)	Occurs after a connection is established.
	AfterDisconnect (see page 6875)	Occurs after the connection closes.
	BeforeConnect (see page 6875)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 6875)	Occurs immediately before the connection closes.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 6863)	Closes the connection.
	Create (see page 16466)	Creates and initializes an instance of TMidasConnection (see page 16462).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.

	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 6863)	Destroys the instance of a connection component.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProviderNames (see page 8530)	Executes a callback for all provider objects on the application server.
	GetServer (see page 8530)	Returns the IAppServer interface of the application server.
	GetServer (see page 16023)	Returns the IAppServer interface of the application server.
	GetServerList (see page 8531)	Returns a list of application servers (remote data modules) to which the component can connect.
	GetServerList (see page 16024)	Returns a list of remote data modules to which the component can connect.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 6866)	Opens the connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AppServer (see page 8532)	Contains the interface for communicating with the application server.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Connected (see page 6868)	Determines whether a connection has been established to the remote source of data.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 6870)	Lists the datasets currently using this connection component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	LoginPrompt (see page 6872)	Specifies whether a login dialog appears immediately before opening a new connection.
	OnLogin (see page 6873)	Occurs when a communication channel opens to the server.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterConnect (see page 16031)	Occurs after a connection is established.
	AfterDisconnect (see page 16031)	Occurs after the connection closes.
	BeforeConnect (see page 16032)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 16032)	Occurs immediately before the connection closes.
	OnGetUsername (see page 16033)	This is OnGetUsername, a member of class TDispatchConnection.

Published Properties

	Name	Description
	ComputerName (see page 16014)	This is ComputerName, a member of class TDCOMConnection.
	Connected (see page 16026)	Indicates whether the TDispatchConnection (see page 16017) object is connected to an application server.
	ConnectType (see page 16469)	Specifies the protocol to use to establish the connection to the application server.
	LoginPrompt (see page 16027)	Specifies whether a login dialog appears immediately before opening a new connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectBroker (see page 16014)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	OnLogin (see page 16028)	Occurs when a communication channel opens to the server.
	ServerGUID (see page 16030)	Specifies the GUID of the application server with which to connect.
	ServerName (see page 16030)	Specifies the name of the application server with which to connect.
	ServerPort (see page 16469)	Specifies the port used by the listening socket on the server machine.
	Tag (see page 2808)	Stores an integer value as part of a component.
	UseBroker (see page 16470)	Indicates whether an OLEnterprise connection should use the OLEnterprise Business Object Broker.

98.1.1.2 TMidasConnection.Create Constructor

Creates and initializes an instance of TMidasConnection (see page 16462).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TMidasConnection(TComponent * AOwner);
```

Description

Call Create to instantiate a TMidasConnection (see page 16462) component at runtime. TMidasConnection (see page 16462) components placed in forms or data modules at design time are created automatically.

Create initializes the value of the ServerPort (see page 16469) property to 211. This is also the default port number for ScktSrvr.exe and ScktSrvc.exe.

See Also

ServerPort (see page 16469)

Create (see page 16022)

Port (see page 18751)

98.1.1.3 TMidasConnection Methods

The methods of the TMidasConnection class are listed here.

Protected Methods

	Name	Description
	DoConnect (see page 16467)	Connects to an application server.
	DoDisconnect (see page 16468)	Drops a connection to an application server.
	SetConnectType (see page 16468)	This is SetConnectType, a member of class TMidasConnection.
	SetUseBroker (see page 16468)	This is SetUseBroker, a member of class TMidasConnection.

98.1.1.3.1 TMidasConnection.DoConnect Method

Connects to an application server.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

DoConnect is called when the Connected property is set to true. DoConnect establishes a dual interface with the remote data module on the application server. This interface is used to locate providers on the application server.

To locate the server system, DoConnect uses the ComputerName property. If the ObjectBroker property is set, DoConnect gets this value from the object broker. Otherwise, the application must assign this property a value directly.

See Also

AfterConnect (see page 6874)

BeforeConnect (see page 6875)

Connected

AppServer (see page 8532)

computername (see page 16014)

ObjectBroker (see page 16028)

98.1.1.3.2 TMidasConnection.DoDisconnect Method

Drops a connection to an application server.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

DoDisconnect is called internally when the Connected property is set to false. It drops the connection and changes the AppServer property to a NULL Variant.

See Also

AfterDisconnect (see page 6875)

BeforeDisconnect (see page 6875)

Connected (see page 16026)

AppServer (see page 8532)

98.1.1.3.3 TMidasConnection.SetConnectType Method

Pascal

```
procedure SetConnectType(Value: TConnectType); virtual;
```

C++

```
virtual __fastcall SetConnectType(TConnectType Value);
```

Description

This is SetConnectType, a member of class TMidasConnection.

98.1.1.3.4 TMidasConnection.SetUseBroker Method

Pascal

```
procedure SetUseBroker(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetUseBroker(Boolean Value);
```

Description

This is SetUseBroker, a member of class TMidasConnection.

98.1.1.4 TMidasConnection Properties

The properties of the TMidasConnection class are listed here.

Published Properties

	Name	Description
	ConnectType (see page 16469)	Specifies the protocol to use to establish the connection to the application server.
	ServerPort (see page 16469)	Specifies the port used by the listening socket on the server machine.
	UseBroker (see page 16470)	Indicates whether an OLEnterprise connection should use the OLEnterprise Business Object Broker.

98.1.1.4.1 TMidasConnection.ConnectType Property

Specifies the protocol to use to establish the connection to the application server.

Pascal

```
property ConnectType: TConnectType;
```

C++

```
__property TConnectType ConnectType;
```

Description

Use ConnectType to specify the desired protocol for connecting with the application server.

See Also

ServerPort (see page 16469)

computername (see page 16014)

UseBroker (see page 16470)

ServerName (see page 16030)

98.1.1.4.2 TMidasConnection.ServerPort Property

Specifies the port used by the listening socket on the server machine.

Pascal

```
property ServerPort: Integer;
```

C++

```
__property int ServerPort;
```

Description

When establishing connections to the application server using TCP/IP sockets, the application server must have a socket dispatcher listening on a predefined port for requests from client applications. If the value of ConnectType (see page 16469) is ctSockets, set ServerPort to the port number used by Scktsrvr.exe or ScktSrvc.exe on the server system. TMidasConnection (see page 16462) attempts to connect to the application server by sending a message to the system specified by the ComputerName property at the port specified by ServerPort.

Note: ServerPort has no effect if ConnectType (see page 16469) is not ctSockets.

See Also

ConnectType (see page 16469)

computername (see page 16014)

Port (see page 18751)

98.1.1.4.3 TMidasConnection.UseBroker Property

Indicates whether an OLEnterprise connection should use the OLEnterprise Business Object Broker.

Pascal

```
property UseBroker: Boolean;
```

C++

```
__property Boolean UseBroker;
```

Description

When establishing connections to the application server using OLEnterprise, UseBroker specifies whether the connection should use the Business Object Broker. If UseBroker is true, TMidasConnection (see page 16462) uses the Business Object Broker, and the ComputerName property specifies the name of the Business Object Broker. If UseBroker is false, TMidasConnection (see page 16462) uses OLEnterprise, but does not go through the Business Object Broker. When UseBroker is false, ComputerName specifies the name of the OLEnterprise server.

For more information about using OLEnterprise, see the OLEnterprise documentation.

Note: The value of UseBroker has no effect unless ConnectType (see page 16469) is ctOLEnterprise.

See Also

ConnectType (see page 16469)

computername (see page 16014)

98.1.2 TRemoteServer Class

TRemoteServer handles a DCOM connection to a remote server in a client application that is part of a multi-tiered application.

Class Hierarchy

Pascal

```
TRemoteServer = class(TDCOMConnection);
```

C++

```
class TRemoteServer : public TDCOMConnection;
```

File

MidasCon

Description

TRemoteServer is provided for backward compatibility. When writing client applications that connect to application servers using DCOM, use TDCOMConnection instead.

See Also

TClientDataSet (see page 8289)

TDCOMConnection (see page 16008)

TSocketConnection (see page 18928)

TWebConnection (see page 18968)

TDatasetprovider

98.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TConnectType (see page 16472)	This is record MidasCon.TConnectType.

98.2.1 MidasCon.TConnectType Enumeration

Pascal

```
TConnectType = (
  ctDCOM,
  ctSockets,
  ctOLEnterprise
);
```

C++

```
enum TConnectType {
  ctDCOM,
  ctSockets,
  ctOLEnterprise
};
```

File

MidasCon

Description

This is record MidasCon.TConnectType.

99 MidItems Namespace

This is namespace MidItems.

Topics

Name	Description
Classes (see page 16478)	The following table lists classes in this documentation.
Interfaces (see page 17053)	The following table lists interfaces in this documentation.
Functions (see page 17085)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 17090)	The following table lists structs, records, enums in this documentation.
Types (see page 17093)	The following table lists types in this documentation.
Constants (see page 17095)	The following table lists constants in this documentation.

Classes

	Name	Description
	TApplyUpdatesButton (see page 16480)	TApplyUpdatesButton generates an HTML control that sends an Update message to the XML broker on an InternetExpress application.
	TCaptionAttributes (see page 16489)	TCaptionAttributes customizes the appearance of the label that identifies a display control in a WebSnap or InternetExpress application.
	TCustomDataForm (see page 16493)	TCustomDataForm is the base class for components that generate an HTML form in an InternetExpress application.
	TCustomDataGrid (see page 16499)	TCustomDataGrid is the base class for Web items that generate an HTML table to display a dataset in an XML data packet.
	TCustomDataNavigator (see page 16511)	TCustomDataNavigator assembles the HTML generated by a set of TDataSetButton (see page 16576) and TXMLButton (see page 16989) descendants.
	TCustomFieldGroup (see page 16519)	TCustomFieldGroup is the base class for Web items that arrange information on an HTML form as a set of labeled fields.
	TCustomLayoutGroup (see page 16528)	TCustomLayoutGroup is the base class for Web items that arrange content produced by arbitrary Web items.
	TCustomQueryButtons (see page 16537)	TCustomQueryButtons assembles the HTML generated by a set of TQueryButton (see page 16718) descendants.
	TCustomQueryForm (see page 16545)	TCustomQueryForm is the base class for Web items that generate HTML forms for letting users enter application-defined values.
	TDataForm (see page 16554)	TDataForm generates an HTML form that displays database information and lets users apply updates.
	TDataGrid (see page 16560)	TDataGrid generates an HTML table to display a dataset in an XML data packet.
	TDataNavigator (see page 16569)	TDataNavigator assembles the HTML generated by a set of TDataSetButton (see page 16576) and TXMLButton (see page 16989) descendants.
	TDataSetButton (see page 16576)	TDataSetButton is the base class for Web items that create HTML buttons for navigating or editing data.

	TDeleteButton (see page 16586)	TDeleteButton generates an HTML control that deletes a record from an XML data packet.
	TFieldGroup (see page 16591)	TFieldGroup arranges database information on an HTML form as a set of labeled fields.
	TFieldRadioGroup (see page 16599)	TFieldRadioGroup creates an HTML radio group control to represent a field in an XML data packet.
	TFieldSelectOptions (see page 16612)	TFieldSelectOptions creates an HTML list control to represent a field in an XML data packet.
	TFieldStatus (see page 16624)	TFieldStatus generates an HTML control that indicates whether a record in an XML data packet has been modified by the end user.
	TFieldText (see page 16631)	TFieldText creates a single-line HTML input control to represent a field in an XML data packet.
	TFieldTextArea (see page 16640)	TFieldTextArea creates a multi-line HTML input control to represent a field in an XML data packet.
	TFirstButton (see page 16651)	TFirstButton generates an HTML control that moves to the first record in an XML data packet.
	TFormLayout (see page 16656)	TFormLayout lets child Web items adhere to layout rules imposed by their parent.
	TGridAttributes (see page 16664)	TGridAttributes describes the options the TDataGrid (see page 16560) component adds to an HTML <TABLE> tag.
	TGridRowAttributes (see page 16671)	TGridRowAttributes describes the options the TDataGrid (see page 16560) component adds to its HTML row tags (<TR>).
	TInsertButton (see page 16677)	TInsertButton generates an HTML control that inserts a new record in an XML data packet.
	TLastButton (see page 16682)	TLastButton generates an HTML control that moves to the last record in an XML data packet.
	TLayoutGroup (see page 16687)	TLayoutGroup generates the HTML to arrange content produced by other Web items.
	TNextButton (see page 16694)	TNextButton generates an HTML control that moves to the next record in an XML data packet.
	TNextPageButton (see page 16699)	TNextPageButton generates an HTML control that moves to the next set of records in an XML data packet.
	TPostButton (see page 16704)	TPostButton generates an HTML control that posts changes to an XML delta packet.
	TPriorButton (see page 16709)	TPriorButton generates an HTML control that moves to the previous record in an XML data packet.
	TPriorPageButton (see page 16713)	TPriorPageButton generates an HTML control that moves to the previous set of records in an XML data packet.
	TQueryButton (see page 16718)	TQueryButton is the base class for Web items that create HTML buttons that manage application-defined values.
	TQueryButtons (see page 16730)	TQueryButtons assembles the HTML generated by a set of TQueryButton (see page 16718) descendants.
	TQueryFieldGroup (see page 16736)	TQueryFieldGroup arranges information on an HTML form as a set of labeled fields.
	TQueryForm (see page 16745)	TQueryForm generates HTML forms for letting users enter application-defined values.
	TQueryRadioGroup (see page 16753)	TQueryRadioGroup creates an HTML radio group control to represent named parameter.
	TQuerySelectOptions (see page 16768)	TQuerySelectOptions creates an HTML list control to represent a named parameter.
	TQueryText (see page 16783)	TQueryText creates a single-line HTML input control to represent a named parameter.
	TQueryTextArea (see page 16795)	TQueryTextArea creates a multi-line HTML input control to represent named parameter.
	TResetQueryButton (see page 16809)	TResetQueryButton generates an HTML control that backs out changes to the values on a query form.
	TSelectOptionsColumn (see page 16814)	TSelectOptionsColumn creates an HTML list control to represent a field value in a data grid.
	TStatusColumn (see page 16826)	TStatusColumn generates an HTML control that indicates whether a record in a data grid has been modified by the end user.
	TSubmitQueryButton (see page 16833)	TSubmitQueryButton generates an HTML control that backs out changes to the values on a query form.
	TTextAreaColumn (see page 16838)	TTextAreaColumn creates a multi-line HTML input control to represent a field value in a data grid.

	TTextColumn (see page 16849)	TTextColumn creates a single-line HTML input control to represent a field value in a data grid.
	TUndoButton (see page 16858)	TUndoButton generates an HTML control that backs out changes to the current record.
	TWebButton (see page 16863)	TWebButton is the base class for Web items that create HTML button controls.
	TWebControlGroup (see page 16872)	TWebControlGroup is the base class for Web items that arrange HTML controls.
	TWebDataDisplay (see page 16884)	TWebDataDisplay is the base class for Web items that create HTML display controls.
	TWebDataInput (see page 16900)	TWebDataInput is the base class for Web items that create HTML display controls for entering information.
	TWebDisplay (see page 16911)	This is class MidItems.TWebDisplay.
	TWebDisplayReferenceGroup (see page 16915)	TWebDisplayReferenceGroup is the base class for Web items that arrange a set of controls that influence a control that displays data.
	TWebForm (see page 16922)	TWebForm is the base class for Web items that create HTML forms in InternetExpress applications.
	TWebListInput (see page 16935)	TWebListInput is the base class for Web items that create HTML display controls which present a fixed set of possible values.
	TWebRadioGroupInput (see page 16947)	TWebRadioGroupInput is the base class for Web items that create HTML radio group controls.
	TWebSelectOptionsInput (see page 16956)	TWebSelectOptionsInput is the base class for Web items that create HTML list controls.
	TWebStatus (see page 16963)	TWebStatus is the base class for Web items that indicate the status of records in an XML data packet.
	TWebTextAreaInput (see page 16971)	TWebTextAreaInput is the base class for Web items that create HTML multi-line text controls.
	TWebTextInput (see page 16980)	TWebTextInput is the base class for Web items that create HTML single-line text controls.
	TXMLButton (see page 16989)	TXMLButton is the base class for Web items that create HTML buttons that perform some action with the XML data packet.
	TXMLData (see page 16998)	TXMLData manages the relationship between an XML broker and a Web item that represents a dataset in that XML broker's data packet.
	TXMLDataParent (see page 17003)	This is class MidItems.TXMLDataParent.
	TXMLDataSet (see page 17006)	TXMLDataSet manages the relationship between a Web item and the XML dataset to which it is bound.
	TXMLDataSetParent (see page 17014)	This is class MidItems.TXMLDataSetParent.
	TXMLDisplayGroup (see page 17019)	TXMLDisplayGroup is the base class for Web items that arrange a set of special-purpose controls in an HTML form.
	TXMLDisplayParent (see page 17034)	This is class MidItems.TXMLDisplayParent.
	TXMLDisplayReferenceButton (see page 17037)	TXMLDisplayReferenceButton is the base class for Web items that generate HTML buttons for influencing controls that display data.
	TXMLDisplayReferenceGroup (see page 17046)	TXMLDisplayReferenceGroup is the base class for Web items that arrange a set of button controls that influence a control that displays data.

Constants

Name	Description
FormMethodNames (see page 17095)	This is constant MidItems.FormMethodNames.
HTMLAlignValues (see page 17096)	This is constant MidItems.HTMLAlignValues.
HTMLVAlignValues (see page 17096)	This is constant MidItems.HTMLVAlignValues.
sActionAttr (see page 17096)	This is constant MidItems.sActionAttr.
sAlignAttr (see page 17097)	This is constant MidItems.sAlignAttr.
sBgColorAttr (see page 17097)	This is constant MidItems.sBgColorAttr.
sBorderAttr (see page 17097)	This is constant MidItems.sBorderAttr.
sCellPaddingAttr (see page 17098)	This is constant MidItems.sCellPaddingAttr.
sCellSpacingAttr (see page 17098)	This is constant MidItems.sCellSpacingAttr.
sCheckedAttr (see page 17098)	This is constant MidItems.sCheckedAttr.
sClassAttr (see page 17099)	This is constant MidItems.sClassAttr.
sColsAttr (see page 17099)	This is constant MidItems.sColsAttr.
sColSpanAttr (see page 17099)	This is constant MidItems.sColSpanAttr.
sEncTypeAttr (see page 17100)	This is constant MidItems.sEncTypeAttr.
sImageHeightAttr (see page 17100)	This is constant MidItems.sImageHeightAttr.

sImageWidthAttr (see page 17100)	This is constant MidItems.sImageWidthAttr.
sMaxLengthAttr (see page 17101)	This is constant MidItems.sMaxLengthAttr.
sMethodAttr (see page 17101)	This is constant MidItems.sMethodAttr.
sMultipleAttr (see page 17101)	This is constant MidItems.sMultipleAttr.
sNameAttr (see page 17102)	This is constant MidItems.sNameAttr.
sOnClickAttr (see page 17102)	This is constant MidItems.sOnClickAttr.
sReadOnlyAttr (see page 17102)	This is constant MidItems.sReadOnlyAttr.
sRowsAttr (see page 17103)	This is constant MidItems.sRowsAttr.
sSelectedAttr (see page 17103)	This is constant MidItems.sSelectedAttr.
sSizeAttr (see page 17103)	This is constant MidItems.sSizeAttr.
sStyleAttr (see page 17104)	This is constant MidItems.sStyleAttr.
sTabIndexAttr (see page 17104)	This is constant MidItems.sTabIndexAttr.
sTypeAttr (see page 17104)	This is constant MidItems.sTypeAttr.
sVAlignAttr (see page 17105)	This is constant MidItems.sVAlignAttr.
sValueAttr (see page 17105)	This is constant MidItems.sValueAttr.
sWidthAttr (see page 17105)	This is constant MidItems.sWidthAttr.
sWrapAttr (see page 17106)	This is constant MidItems.sWrapAttr.
TextAreaWrap (see page 17106)	This is constant MidItems.TextAreaWrap.

Functions

	Name	Description
	AddBoolAttrib (see page 17085)	This is function MidItems.AddBoolAttrib.
	AddCustomAttrib (see page 17085)	This is function MidItems.AddCustomAttrib.
	AddIntAttrib (see page 17086)	This is function MidItems.AddIntAttrib.
	AddQuotedAttrib (see page 17086)	This is function MidItems.AddQuotedAttrib.
	AddStringAttrib (see page 17086)	This is function MidItems.AddStringAttrib.
	CanAddClassHelper (see page 17087)	This is function MidItems.CanAddClassHelper.
	CreateDefaultButtonClasses (see page 17087)	This is function MidItems.CreateDefaultButtonClasses.
	DeclareSubmitForm (see page 17087)	This is function MidItems.DeclareSubmitForm.
	FindDispatcher (see page 17088)	This is function MidItems.FindDispatcher.
	FindProducer (see page 17088)	This is function MidItems.FindProducer.
	FormatColumnData (see page 17088)	This is function MidItems.FormatColumnData.
	FormatColumnHeading (see page 17089)	This is function MidItems.FormatColumnHeading.
	GetItemValuesFromDataSet (see page 17089)	This is function MidItems.GetItemValuesFromDataSet.

Interfaces

	Name	Description
	IDataSetComponent (see page 17053)	IDataSetComponent defines the methods that bind a Web item to an XML broker.
	IDataSetField (see page 17057)	This is class MidItems.IDataSetField.
	IDataSetFields (see page 17059)	This is class MidItems.IDataSetFields.
	IFormatColumn (see page 17061)	This is class MidItems.IFormatColumn.
	IHTMLField (see page 17062)	IHTMLField is an interface that must be implemented by Web items that generate <INPUT> elements.
	IHTMLForm (see page 17064)	IHTMLForm is an interface that must be implemented by Web items that generate forms.
	IInetXWebDataSet (see page 17066)	IInetXWebDataSet lets a Web item interact with information about a dataset in an XML broker's data packet.
	IQueryField (see page 17070)	This is class MidItems.IQueryField.
	IQueryFields (see page 17073)	This is class MidItems.IQueryFields.
	IQueryForm (see page 17074)	This is class MidItems.IQueryForm.
	IRestoreDefaults (see page 17074)	This is class MidItems.IRestoreDefaults.
	IStatusField (see page 17075)	This is class MidItems.IStatusField.
	IValidateField (see page 17076)	This is class MidItems.IValidateField.
	IValidateFields (see page 17077)	This is class MidItems.IValidateFields.
	IWebDisplay (see page 17079)	

	IWebDisplayReference (see page 17080)	
	IXMLDisplay (see page 17082)	This is class MidItems.IXMLDisplay.

Structs, Records, Enums

	Name	Description
	TCaptionPosition (see page 17090)	TCaptionPosition indicates the relative position of a label to the HTML control it labels.
	TFormMethod (see page 17091)	TFormMethod represents the method attribute on the <form> tag of an HTML form.
	TLayoutState (see page 17091)	This is record MidItems.TLayoutState.
	TTextAreaWrap (see page 17092)	TTextAreaWrap indicates how an HTML display control should treat items that are too wide for the physical width of the control.

Types

Name	Description
IMidasWebDataSet (see page 17093)	This is type MidItems.IMidasWebDataSet.
TWebButtonClass (see page 17093)	This is type MidItems.TWebButtonClass.
TWebDataDisplayClass (see page 17094)	This is type MidItems.TWebDataDisplayClass.
TWebDataInputClass (see page 17094)	This is type MidItems.TWebDataInputClass.
TWebStatusClass (see page 17094)	This is type MidItems.TWebStatusClass.

99.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TApplyUpdatesButton (see page 16480)	TApplyUpdatesButton generates an HTML control that sends an Update message to the XML broker on an InternetExpress application.
	TCaptionAttributes (see page 16489)	TCaptionAttributes customizes the appearance of the label that identifies a display control in a WebSnap or InternetExpress application.
	TCustomDataForm (see page 16493)	TCustomDataForm is the base class for components that generate an HTML form in an InternetExpress application.
	TCustomDataGrid (see page 16499)	TCustomDataGrid is the base class for Web items that generate an HTML table to display a dataset in an XML data packet.
	TCustomDataNavigator (see page 16511)	TCustomDataNavigator assembles the HTML generated by a set of TDataSetButton (see page 16576) and TXMLButton (see page 16989) descendants.
	TCustomFieldGroup (see page 16519)	TCustomFieldGroup is the base class for Web items that arrange information on an HTML form as a set of labeled fields.
	TCustomLayoutGroup (see page 16528)	TCustomLayoutGroup is the base class for Web items that arrange content produced by arbitrary Web items.
	TCustomQueryButtons (see page 16537)	TCustomQueryButtons assembles the HTML generated by a set of TQueryButton (see page 16718) descendants.
	TCustomQueryForm (see page 16545)	TCustomQueryForm is the base class for Web items that generate HTML forms for letting users enter application-defined values.
	TDataForm (see page 16554)	TDataForm generates an HTML form that displays database information and lets users apply updates.
	TDataGrid (see page 16560)	TDataGrid generates an HTML table to display a dataset in an XML data packet.
	TDataNavigator (see page 16569)	TDataNavigator assembles the HTML generated by a set of TDataSetButton (see page 16576) and TXMLButton (see page 16989) descendants.
	TDataSetButton (see page 16576)	TDataSetButton is the base class for Web items that create HTML buttons for navigating or editing data.
	TDeleteButton (see page 16586)	TDeleteButton generates an HTML control that deletes a record from an XML data packet.
	TFieldGroup (see page 16591)	TFieldGroup arranges database information on an HTML form as a set of labeled fields.
	TFieldRadioGroup (see page 16599)	TFieldRadioGroup creates an HTML radio group control to represent a field in an XML data packet.
	TFieldSelectOptions (see page 16612)	TFieldSelectOptions creates an HTML list control to represent a field in an XML data packet.
	TFieldStatus (see page 16624)	TFieldStatus generates an HTML control that indicates whether a record in an XML data packet has been modified by the end user.
	TFieldText (see page 16631)	TFieldText creates a single-line HTML input control to represent a field in an XML data packet.
	TFieldTextArea (see page 16640)	TFieldTextArea creates a multi-line HTML input control to represent a field in an XML data packet.
	TFirstButton (see page 16651)	TFirstButton generates an HTML control that moves to the first record in an XML data packet.
	TFormLayout (see page 16656)	TFormLayout lets child Web items adhere to layout rules imposed by their parent.
	TGridAttributes (see page 16664)	TGridAttributes describes the options the TDataGrid (see page 16560) component adds to an HTML <TABLE> tag.
	TGridRowAttributes (see page 16671)	TGridRowAttributes describes the options the TDataGrid (see page 16560) component adds to its HTML row tags (<TR>).
	TInsertButton (see page 16677)	TInsertButton generates an HTML control that inserts a new record in an XML data packet.
	TLastButton (see page 16682)	TLastButton generates an HTML control that moves to the last record in an XML data packet.

	TLayoutGroup (see page 16687)	TLayoutGroup generates the HTML to arrange content produced by other Web items.
	TNextButton (see page 16694)	TNextButton generates an HTML control that moves to the next record in an XML data packet.
	TNextPageButton (see page 16699)	TNextPageButton generates an HTML control that moves to the next set of records in an XML data packet.
	TPostButton (see page 16704)	TPostButton generates an HTML control that posts changes to an XML delta packet.
	TPriorButton (see page 16709)	TPriorButton generates an HTML control that moves to the previous record in an XML data packet.
	TPriorPageButton (see page 16713)	TPriorPageButton generates an HTML control that moves to the previous set of records in an XML data packet.
	TQueryButton (see page 16718)	TQueryButton is the base class for Web items that create HTML buttons that manage application-defined values.
	TQueryButtons (see page 16730)	TQueryButtons assembles the HTML generated by a set of TQueryButton (see page 16718) descendants.
	TQueryFieldGroup (see page 16736)	TQueryFieldGroup arranges information on an HTML form as a set of labeled fields.
	TQueryForm (see page 16745)	TQueryForm generates HTML forms for letting users enter application-defined values.
	TQueryRadioGroup (see page 16753)	TQueryRadioGroup creates an HTML radio group control to represent named parameter.
	TQuerySelectOptions (see page 16768)	TQuerySelectOptions creates an HTML list control to represent a named parameter.
	TQueryText (see page 16783)	TQueryText creates a single-line HTML input control to represent a named parameter.
	TQueryTextArea (see page 16795)	TQueryTextArea creates a multi-line HTML input control to represent named parameter.
	TResetQueryButton (see page 16809)	TResetQueryButton generates an HTML control that backs out changes to the values on a query form.
	TSelectOptionsColumn (see page 16814)	TSelectOptionsColumn creates an HTML list control to represent a field value in a data grid.
	TStatusColumn (see page 16826)	TStatusColumn generates an HTML control that indicates whether a record in a data grid has been modified by the end user.
	TSubmitQueryButton (see page 16833)	TSubmitQueryButton generates an HTML control that backs out changes to the values on a query form.
	TTextAreaColumn (see page 16838)	TTextAreaColumn creates a multi-line HTML input control to represent a field value in a data grid.
	TTextColumn (see page 16849)	TTextColumn creates a single-line HTML input control to represent a field value in a data grid.
	TUndoButton (see page 16858)	TUndoButton generates an HTML control that backs out changes to the current record.
	TWebButton (see page 16863)	TWebButton is the base class for Web items that create HTML button controls.
	TWebControlGroup (see page 16872)	TWebControlGroup is the base class for Web items that arrange HTML controls.
	TWebDataDisplay (see page 16884)	TWebDataDisplay is the base class for Web items that create HTML display controls.
	TWebDataInput (see page 16900)	TWebDataInput is the base class for Web items that create HTML display controls for entering information.
	TWebDisplay (see page 16911)	This is class MidItems.TWebDisplay.
	TWebDisplayReferenceGroup (see page 16915)	TWebDisplayReferenceGroup is the base class for Web items that arrange a set of controls that influence a control that displays data.
	TWebForm (see page 16922)	TWebForm is the base class for Web items that create HTML forms in InternetExpress applications.
	TWebListInput (see page 16935)	TWebListInput is the base class for Web items that create HTML display controls which present a fixed set of possible values.
	TWebRadioGroupInput (see page 16947)	TWebRadioGroupInput is the base class for Web items that create HTML radio group controls.
	TWebSelectOptionsInput (see page 16956)	TWebSelectOptionsInput is the base class for Web items that create HTML list controls.
	TWebStatus (see page 16963)	TWebStatus is the base class for Web items that indicate the status of records in an XML data packet.
	TWebTextAreaInput (see page 16971)	TWebTextAreaInput is the base class for Web items that create HTML multi-line text controls.

	TWebTextInput (see page 16980)	TWebTextInput is the base class for Web items that create HTML single-line text controls.
	TXMLButton (see page 16989)	TXMLButton is the base class for Web items that create HTML buttons that perform some action with the XML data packet.
	TXMLData (see page 16998)	TXMLData manages the relationship between an XML broker and a Web item that represents a dataset in that XML broker's data packet.
	TXMLDataParent (see page 17003)	This is class MidItems.TXMLDataParent.
	TXMLDataSet (see page 17006)	TXMLDataSet manages the relationship between a Web item and the XML dataset to which it is bound.
	TXMLDataSetParent (see page 17014)	This is class MidItems.TXMLDataSetParent.
	TXMLDisplayGroup (see page 17019)	TXMLDisplayGroup is the base class for Web items that arrange a set of special-purpose controls in an HTML form.
	TXMLDisplayParent (see page 17034)	This is class MidItems.TXMLDisplayParent.
	TXMLDisplayReferenceButton (see page 17037)	TXMLDisplayReferenceButton is the base class for Web items that generate HTML buttons for influencing controls that display data.
	TXMLDisplayReferenceGroup (see page 17046)	TXMLDisplayReferenceGroup is the base class for Web items that arrange a set of button controls that influence a control that displays data.

99.1.1 TApplyUpdatesButton Class

TApplyUpdatesButton generates an HTML control that sends an Update message to the XML broker on an InternetExpress application.

Class Hierarchy

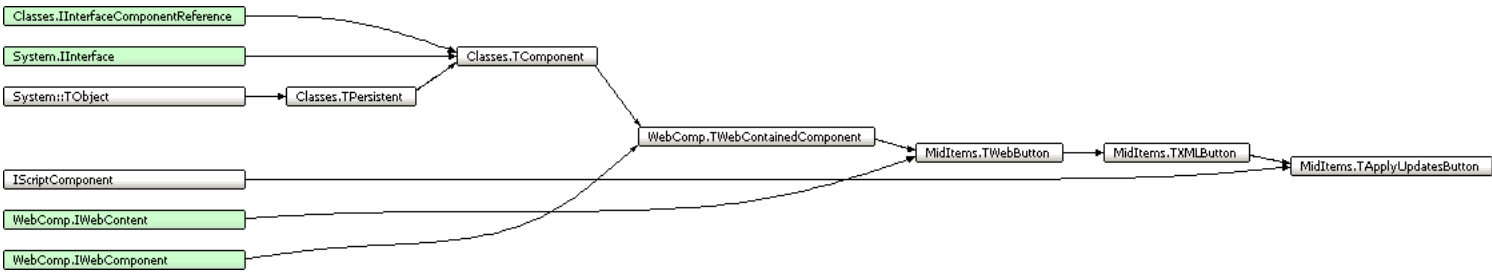

Pascal

```
TApplyUpdatesButton = class(TXMLButton, IScriptComponent);
```

C++

```
class TApplyUpdatesButton : public TXMLButton, public IScriptComponent;
```

File

MidItems

Description

Add TApplyUpdatesButton as the child of a TDataNavigator (see page 16569) component (or a layout group that is a child of a TDataNavigator (see page 16569) component) to enable the generated HTML form to submit changes made by end users. The resulting HTML button's action generates an HTTP message that forwards all changes to a specific XML data packet back to the XML broker on the InternetExpress application.

Use the properties of TApplyUpdatesButton to customize the appearance of the button and associate it with an XML broker. The associated XML broker adds the XML data packet to the HTML document and responds to update messages generated by the apply updates button on the HTML form.

See Also

TXMLBroker (see page 26049)

TDataNavigator (see page 16569)

TLayoutGroup (see page 16687)

Using the Web Page Editor

99.1.1.1 TApplyUpdatesButton Members

The following tables list the members exposed by TApplyUpdatesButton.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16485)	This is AddElements, a member of class TApplyUpdatesButton.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16868)	Returns the HTML that defines the button control.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetCaption (see page 16869)	Returns the value of the Caption (see page 16870) property.
V	GetCaption (see page 16994)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16485)	This is GetSubComponents, a member of class TApplyUpdatesButton.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXMLBroker (see page 16995)	This is GetXMLBroker, a member of class TXMLButton.
	GetXMLUseParent (see page 16995)	This is GetXMLUseParent, a member of class TXMLButton.
V	ImplAddElements (see page 16485)	This is ImplAddElements, a member of class TApplyUpdatesButton.
V	ImplContent (see page 16485)	Provides the underlying implementation for the Content method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.

V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16995)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	SetXMLBroker (see page 16996)	This is SetXMLBroker, a member of class TXMLButton.
	SetXMLUseParent (see page 16996)	This is SetXMLUseParent, a member of class TXMLButton.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultCaption (see page 16996)	This is DefaultCaption, a member of class TXMLButton.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.
R	XMLData (see page 16997)	This is XMLData, a member of class TXMLButton.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16484)	Creates and initializes an instance of TApplyUpdatesButton (see page 16480).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16994)	Frees an instance of TXMLButton (see page 16989).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Caption (see page 16486)	Specifies the caption that appears on the button.
	Custom (see page 16486)	Represents a tag option and its value in the generated INPUT tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16487)	Adds style attributes to the generated button control.
	StyleRule (see page 16487)	Applies a style in the page producer's style sheet to the button.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLBroker (see page 16488)	Associates the XML button with an XML broker.
	XMLUseParent (see page 16488)	Specifies whether the XML button is associated with its parent's XML broker.

99.1.1.2 TApplyUpdatesButton.Create Constructor

Creates and initializes an instance of TApplyUpdatesButton (see page 16480).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TApplyUpdatesButton(TComponent * AOwner);
```

Description

Call Create to instantiate TApplyUpdatesButton (see page 16480) at runtime. Buttons (see page 2289) added at design time using the Web Page editor are instantiated automatically.

99.1.1.3 TApplyUpdatesButton Methods

The methods of the TApplyUpdatesButton class are listed here.

Protected Methods

	Name	Description
	AddElements (see page 16485)	This is AddElements, a member of class TApplyUpdatesButton.
	GetSubComponents (see page 16485)	This is GetSubComponents, a member of class TApplyUpdatesButton.
V	ImplAddElements (see page 16485)	This is ImplAddElements, a member of class TApplyUpdatesButton.
V	ImplContent (see page 16485)	Provides the underlying implementation for the Content method.

99.1.1.3.1 TApplyUpdatesButton.AddElements Method

Pascal

```
procedure AddElements(AddIntf: IAddScriptElements);
```

C++

```
__fastcall AddElements(IAddScriptElements AddIntf);
```

Description

This is AddElements, a member of class TApplyUpdatesButton.

99.1.1.3.2 TApplyUpdatesButton.GetSubComponents Method

Pascal

```
function GetSubComponents: TObject;
```

C++

```
__fastcall TObject * GetSubComponents();
```

Description

This is GetSubComponents, a member of class TApplyUpdatesButton.

99.1.1.3.3 TApplyUpdatesButton.ImplAddElements Method

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); virtual;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

This is ImplAddElements, a member of class TApplyUpdatesButton.

99.1.1.3.4 TApplyUpdatesButton.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML that defines the button control. TWebButton introduces

ImplContent as an abstract or, in C++ terminology, pure virtual method. TWebButton descendants must override ImplContent to generate the definition of their particular button control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web button should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the button control.

See Also

Content (see page 16868)

ILayoutWebContent (see page 24774)

99.1.1.4 TApplyUpdatesButton Properties

The properties of the TApplyUpdatesButton class are listed here.

Published Properties

	Name	Description
	Caption (see page 16486)	Specifies the caption that appears on the button.
	Custom (see page 16486)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 16487)	Adds style attributes to the generated button control.
	StyleRule (see page 16487)	Applies a style in the page producer's style sheet to the button.
	XMLBroker (see page 16488)	Associates the XML button with an XML broker.
	XMLUseParent (see page 16488)	Specifies whether the XML button is associated with its parent's XML broker.

99.1.1.4.1 TApplyUpdatesButton.Caption Property

Specifies the caption that appears on the button.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the caption that appears on the generated HTML button. The value of Caption is used as the value of a VALUE parameter on the HTML INPUT tag generated to implement the button.

See Also

Custom (see page 16871)

GetCaption (see page 16869)

SetCaption (see page 16870)

99.1.1.4.2 TApplyUpdatesButton.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to represent one or more options that qualify the INPUT tag generated by the Web button. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

```
ALIGN=TOP TABINDEX=2
```

See Also

Caption (see page 16870)

Style (see page 16871)

99.1.1.4.3 TApplyUpdatesButton.Style Property

Adds style attributes to the generated button control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to add a STYLE option that qualifies the INPUT tag generated by the Web button. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated INPUT tag as the value of a STYLE option.

See Also

Custom (see page 16871)

StyleRule (see page 16872)

99.1.1.4.4 TApplyUpdatesButton.StyleRule Property

Applies a style in the page producer's style sheet to the button.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated button control. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the button's Style property.

See Also

Style (see page 16871)

Styles (see page 17127)

StylesFile (see page 17128)

Styles (see page 19830)

StylesFile (see page 19831)

99.1.1.4.5 TApplyUpdatesButton.XMLBroker Property

Associates the XML button with an XML broker.

Pascal

```
property XMLBroker: TXMLBroker;
```

C++

```
__property TXMLBroker XMLBroker;
```

Description

Set XMLBroker to specify the XML broker whose data packets the XML button manipulates.

When XML buttons are used by TDataNavigator, the data navigator already establishes a connection (indirectly) to an XML broker. If the XML button uses this same XML broker, you do not need to set the XMLBroker property. Instead, setting the XMLUseParent property to true (the default) automatically associates the XML button with the same broker as the one associated with the navigator.

See Also

XMLUseParent (see page 16997)

99.1.1.4.6 TApplyUpdatesButton.XMLUseParent Property

Specifies whether the XML button is associated with its parent's XML broker.

Pascal

```
property XMLUseParent: Boolean;
```

C++

```
__property Boolean XMLUseParent;
```

Description

Use XMLUseParent to indicate how the XML button determines the XML broker whose data packets it manipulates. When XMLUseParent is false, the XMLBroker property must be set to indicate the XML broker. When XMLUseParent is true, the XML broker associated with the XML button's parent determines the associated XML broker.

The XML button's parent is the Web item (typically TDataNavigator) that uses the XML button to assemble its own HTML content. The parent's associated broker may be defined through an indirect association. For example, the data navigator acts on another component, such as a field group, which has an associated broker.

If the parent does not have an associated XML broker, setting XMLUseParent to true sets the XMLBroker property to nil (Delphi) or NULL (C++).

See Also

XMLBroker (see page 16996)

99.1.2 TCaptionAttributes Class

TCaptionAttributes customizes the appearance of the label that identifies a display control in a WebSnap or InternetExpress application.

Class Hierarchy

Pascal

```
TCaptionAttributes = class(TPersistent);
```

C++

```
class TCaptionAttributes : public TPersistent;
```

File

MidItems

Description

TCaptionAttributes is the type of the CaptionAttributes property of Web items that generate display controls. It defines several properties that customize the appearance of the label generated by the Web item.

See Also

CaptionAttributes

99.1.2.1 TCaptionAttributes Members

The following tables list the members exposed by TCaptionAttributes.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 16490)	Creates an instance of TCaptionAttributes (see page 16489).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.

	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Parent (see page 16492)	Indicates the Web item that uses TCaptionAttributes (see page 16489) to customize the appearance of its generated caption.

Published Properties

	Name	Description
	Align (see page 16491)	Represents the Align option of an HTML tag.
	Custom (see page 16491)	Represents a tag option and its value.
	Style (see page 16492)	Adds style attributes to the generated caption.
	StyleRule (see page 16492)	Applies a style in the page producer's style sheet to the generated caption.
	VAlign (see page 16493)	Represents the VAlign option of an HTML tag.

99.1.2.2 TCaptionAttributes.Create Constructor

Creates an instance of TCaptionAttributes (see page 16489).

Pascal

```
constructor Create(AParent: TComponent);
```

C++

```
__fastcall TCaptionAttributes(TComponent * AParent);
```

Description

Do not create instances of TCaptionAttributes (see page 16489). This object is instantiated by the Web item that uses it to implement its CaptionAttributes property.

The AParent parameter specifies the Web item that creates a TCaptionAttributes (see page 16489) instance for its CaptionAttributes property. This becomes the value of the Parent (see page 16492) property.

See Also

CaptionAttributes (see page 16897)

Parent (see page 16492)

99.1.2.3 TCaptionAttributes Properties

The properties of the TCaptionAttributes class are listed here.

Public Properties

	Name	Description
R	Parent (see page 16492)	Indicates the Web item that uses TCaptionAttributes (see page 16489) to customize the appearance of its generated caption.

Published Properties

	Name	Description
	Align (see page 16491)	Represents the Align option of an HTML tag.
	Custom (see page 16491)	Represents a tag option and its value.
	Style (see page 16492)	Adds style attributes to the generated caption.
	StyleRule (see page 16492)	Applies a style in the page producer's style sheet to the generated caption.
	VAlign (see page 16493)	Represents the VAlign option of an HTML tag.

99.1.2.3.1 TCaptionAttributes.Align Property

Represents the Align option of an HTML tag.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Use Align to add an Align option to the tag that surrounds the caption. The Align option influences the horizontal alignment of the caption text.

The Web item that generates the caption inserts the specified Align option into the HTML tag when it generates its content.

See Also

VAlign (see page 16493)

99.1.2.3.2 TCaptionAttributes.Custom Property

Represents a tag option and its value.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to add one or more options to the tag that surrounds the caption. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces.

The Web item that generates the caption inserts the Custom string into the HTML tag when it generates its content.

See Also

Style (see page 16492)

StyleRule (see page 16492)

99.1.2.3.3 TCaptionAttributes.Parent Property

Indicates the Web item that uses TCaptionAttributes (see page 16489) to customize the appearance of its generated caption.

Pascal

```
property Parent: TComponent;
```

C++

```
__property TComponent * Parent;
```

Description

Use Parent to access the Web item that uses the TCaptionAttributes (see page 16489) object.

See Also

Create (see page 16490)

99.1.2.3.4 TCaptionAttributes.Style Property

Adds style attributes to the generated caption.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated caption. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the HTML tag that generates the caption as the value of a STYLE option.

See Also

Custom (see page 16491)

StyleRule (see page 16492)

99.1.2.3.5 TCaptionAttributes.StyleRule Property

Applies a style in the page producer's style sheet to the generated caption.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated caption. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

To apply a style definition that is not included in the style sheet, use the Style (see page 16492) property instead.

See Also

Style (see page 16492)

Styles (see page 17127)

Styles (see page 19830)

StylesFile (see page 17128)

StylesFile (see page 19831)

99.1.2.3.6 TCaptionAttributes.VAlign Property

Represents the VAlign option of an HTML tag.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

Use VAlign to add an VAlign option to the tag that surrounds the caption. The VAlign option influences the vertical alignment of the caption text.

The Web item that generates the caption inserts the specified VAlign option into the HTML tag when it generates its content.

See Also

Align (see page 16491)

99.1.3 TCustomDataForm Class

TCustomDataForm is the base class for components that generate an HTML form in an InternetExpress application.

Class Hierarchy

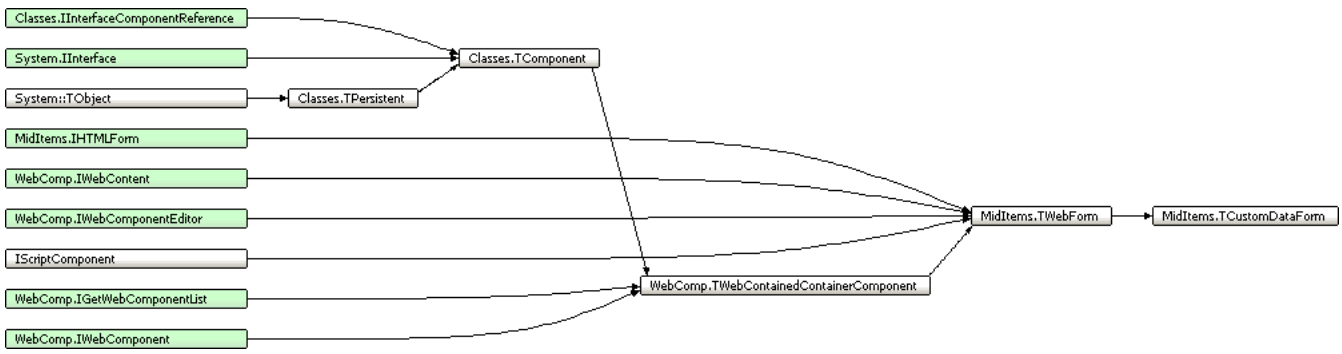

Pascal

```
TCustomDataForm = class(TWebForm);
```

C++

```
class TCustomDataForm : public TWebForm;
```

File

MidItems

Description

TCustomDataForm is the base class for components that generate HTML forms to display database information that users can view and edit. Such forms can be added to the items of an InternetExpress page producer (InetXPageProducer).

TCustomDataForm implements the ability to generate an HTML form that contains controls for displaying information from an XML data packet. Buttons (see page 2289) on the form manipulate that data, and submit updates back to the XML broker that supplies the XML data packet.

TCustomDataForm generates the HTML FORM tag and assembles the HTML generated by its Web items. It restricts the Web items that generate the contents of the form to components that generate data displays (TDataGrid (see page 16560) or TFieldGroup (see page 16591)), components that contain buttons which operate on those displays (TDataNavigator (see page 16569)), and components that manage the layout of HTML controls (TLayoutGroup (see page 16687)). At design time, you can add these Web items to the form using the Web page editor.

See Also

IWebContent (see page 24792)

TInetXPageProducer (see page 17145)

99.1.3.1 TCustomDataForm Members

The following tables list the members exposed by TCustomDataForm.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16927)	Used internally to generate any javascript declarations needed by the form.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16928)	Specifies whether a Web item of a specified type can be added to the WebComponents (see page 16934) property list.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16928)	Returns the HTML that defines the HTML form.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.

	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDefaultWebComponents (see page 16929)	Returns nil (Delphi) or NULL (C++).
	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetHTMLFormName (see page 16929)	Returns the name of the generated HTML form.
	GetHTMLFormTag (see page 16498)	This is GetHTMLFormTag, a member of class TCustomDataForm.
	GetHTMLFormVarName (see page 16929)	Returns the name of the javascript variable that refers to the generated form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetLayoutAttributes (see page 16930)	Returns the object that specifies how the generated form should look.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16930)	Returns the value of the WebComponents (see page 16934) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVisibleFields (see page 16930)	Returns the value of the VisibleFields (see page 16934) property.
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	ImplAddElements (see page 16931)	Provides the underlying implementation for the AddElements (see page 16927) method.
	ImplCanAddClass (see page 16498)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 16499)	Provides the underlying implementation for the Content method.
	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
	SetDefaultWebComponents (see page 16932)	Prevents an application from trying to assign a default set of Web items on the form.
	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	VisibleFields (see page 16934)	Lists the Web items whose generated content appears in the HTML form.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16926)	Creates and initializes an instance of TWebForm (see page 16922).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24742)	Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16926)	Frees an instance of TWebForm (see page 16922).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHTMLFormName (see page 17065)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 17065)	Returns the name of the javascript variable that refers to the generated form.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.

R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16933)	Represents a tag option and its value in the generated FORM tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLFormName (see page 17066)	Indicates the name of the generated HTML form.
R	HTMLFormVarName (see page 17066)	Indicates the name of the javascript variable that represents the HTML form.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16933)	Adds style attributes to the generated form.
	StyleRule (see page 16934)	Applies a style in the page producer's style sheet to the button.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebComponents (see page 16934)	Maintains a list of components that generate HTML content for the TWebForm (see page 16922) descendant.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.3.2 TCustomDataForm Methods

The methods of the TCustomDataForm class are listed here.

Protected Methods

	Name	Description
V	GetHTMLFormTag (see page 16498)	This is GetHTMLFormTag, a member of class TCustomDataForm.
V	ImplCanAddClass (see page 16498)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 16499)	Provides the underlying implementation for the Content method.

99.1.3.2.1 TCustomDataForm.GetHTMLFormTag Method

Pascal

```
function GetHTMLFormTag(Options: TWebContentOptions): string; virtual;
```

C++

```
virtual __fastcall AnsiString GetHTMLFormTag(TWebContentOptions Options);
```

Description

This is GetHTMLFormTag, a member of class TCustomDataForm.

99.1.3.2.2 TCustomDataForm.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebForm descendant can add a component of

type AClass to its WebComponents property list. AParent is the TWebForm instance to which the component will be added.

In TWebForm, ImplCanAddClass is abstract or, in C++ terminology, pure virtual. TWebForm descendants must override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML forms.

See Also

CanAddClass (see page 16928)

WebComponents (see page 16934)

99.1.3.2.3 TCustomDataForm.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML form. ImplContent is anabstract or, in C++ terminology, pure virtual method in TWebForm. TWebForm descendants override it to generate the HTML form.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebForm should format the HTML control it generates. Typically, this parameter is nil (Delphi) or NULL (C++) for forms.

ImplContent returns an HTML form definition.

See Also

Content (see page 16928)

AddElements (see page 16927)

99.1.4 TCustomDataGrid Class

TCustomDataGrid is the base class for Web items that generate an HTML table to display a dataset in an XML data packet.

Class Hierarchy

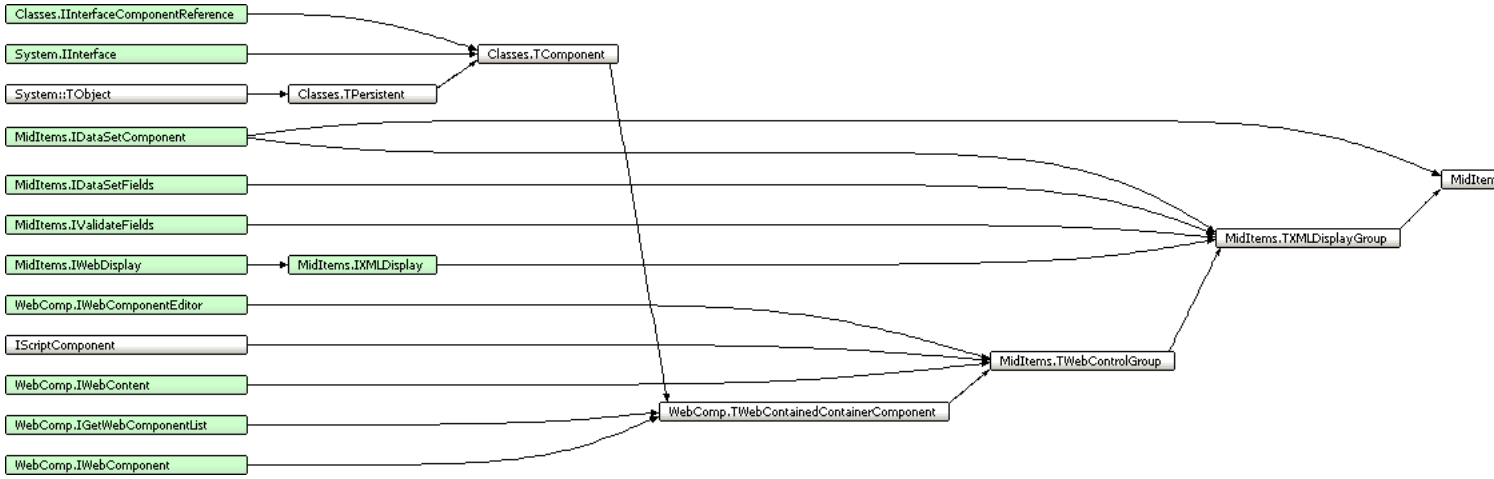

Pascal

```
TCustomDataGrid = class(TXMLDisplayGroup, IDataSetComponent);
```

C++

```
class TCustomDataGrid : public TXMLDisplayGroup, public IDataSetComponent;
```

File

MidItems

Description

Use TCustomDataGrid as a base class for Web items that generate an HTML table to display multiple records from a dataset as a set of rows and columns. Each TCustomDataGrid object maintains a list of Web items that generate the individual fields (columns) of the table. These Web items can be from the following classes: TTextColumn (see page 16849), TSelectOptionColumn, TTextAreaColumn (see page 16838), and TStatusColumn (see page 16826).

You can bind the data grid to an XML data packet or dataset field within an XML data packet. The column Web items can then use that binding to determine the source of the data they display.

TCustomDataGrid introduces a number of properties that affect the appearance of the generated table.

In order to let other Web items use the HTML table that the data grid generates, TCustomDataGrid inherits the IWebContent interface from TWebControlGroup (see page 16872).

Do not create instances of TCustomDataGrid. Instead, use a descendant of TCustomDataGrid such as TDataGrid (see page 16560).

See Also

TDataGrid (see page 16560)

IWebContent (see page 24792)

TSelectOptionsColumn (see page 16814)

TStatusColumn (see page 16826)

TTextAreaColumn (see page 16838)

TTextColumn (see page 16849)

99.1.4.1 TCustomDataGrid Members

The following tables list the members exposed by TCustomDataGrid.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
V	CreateDefaultFields (see page 17025)	This is CreateDefaultFields, a member of class TXMLDisplayGroup.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FindField (see page 17026)	This is FindField, a member of class TXMLDisplayGroup.
	FormatTable (see page 16506)	This is FormatTable, a member of class TCustomDataGrid.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 17026)	Returns the value of the DataSet property.
	GetDataSetField (see page 17026)	This is GetDataSetField, a member of class TXMLDisplayGroup.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetEnableValidateFields (see page 17026)	This is GetEnableValidateFields, a member of class TXMLDisplayGroup.
V	GetFieldsList (see page 17027)	This is GetFieldsList, a member of class TXMLDisplayGroup.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsMultipleRecordView (see page 17027)	This is GetIsMultipleRecordView, a member of class TXMLDisplayGroup.
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVisibleFields (see page 17027)	This is GetVisibleFields, a member of class TXMLDisplayGroup.
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	GetXMLBroker (see page 17027)	Returns the value of the XMLBroker property.
	GetXMLDataSet (see page 17028)	Returns the value of the XMLDataSet property.

	GetXMLDisplayName (see page 17028)	This is GetXMLDisplayName, a member of class TXMLDisplayGroup.
	GetXMLRowSetName (see page 17028)	This is GetXMLRowSetName, a member of class TXMLDisplayGroup.
	HasStatusField (see page 17028)	This is HasStatusField, a member of class TXMLDisplayGroup.
V	ImplAddElements (see page 16506)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 16507)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 16507)	Provides the underlying implementation for the Content method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
V	ImplGetSubComponents (see page 17029)	Provides the underlying implementation for the GetSubComponents method.
V	ImplIsMultipleRecordView (see page 16508)	This is ImplIsMultipleRecordView, a member of class TCustomDataGrid.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsFieldInUse (see page 17030)	This is IsFieldInUse, a member of class TXMLDisplayGroup.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 17030)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
	SetDataSetField (see page 17030)	This is SetDataSetField, a member of class TXMLDisplayGroup.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEnableValidateFields (see page 17031)	This is SetEnableValidateFields, a member of class TXMLDisplayGroup.
	SetHeadingAttributes (see page 16508)	This is SetHeadingAttributes, a member of class TCustomDataGrid.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetRowAttributes (see page 16508)	This is SetRowAttributes, a member of class TCustomDataGrid.
	SetTableAttributes (see page 16508)	This is SetTableAttributes, a member of class TCustomDataGrid.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	SetXMLBroker (see page 17031)	This is SetXMLBroker, a member of class TXMLDisplayGroup.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateDataSet (see page 17031)	This is ValidateDataSet, a member of class TXMLDisplayGroup.
	ValidateFields (see page 17031)	This is ValidateFields, a member of class TXMLDisplayGroup.

	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.
V	XMLDataChange (see page 17031)	This is XMLDataChange, a member of class TXMLDisplayGroup.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
	EnableValidateFields (see page 17032)	This is EnableValidateFields, a member of class TXMLDisplayGroup.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.
R	XMLDataSet (see page 17033)	Indicates the TXMLDataSet object that represents the dataset to which the Web item is bound.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16505)	Creates and initializes an instance of TCustomDataGrid (see page 16499).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 16506)	Frees an instance of TCustomDataGrid (see page 16499).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.

	GetDataSet (see page 17054)	Returns the value of the DataSet (see page 17055) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnableValidateFields (see page 17078)	This is GetEnableValidateFields, a member of class IValidateFields.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 17060)	This is GetFieldsList, a member of class IDataSetFields.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsMultipleRecordView (see page 17079)	This is GetIsMultipleRecordView, a member of class IWebDisplay.
	GetIsMultipleRecordView (see page 17082)	This is GetIsMultipleRecordView, a member of class IXMLDisplay.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	GetVisibleFields (see page 17060)	This is GetVisibleFields, a member of class IDataSetFields.
	GetXMLBroker (see page 17054)	Returns the value of the XMLBroker (see page 17056) property.
	GetXMLDataSet (see page 17055)	Returns the value of the XMLDataSet (see page 17056) property.
	GetXMLDisplayName (see page 17083)	This is GetXMLDisplayName, a member of class IXMLDisplay.
	GetXMLRowSetName (see page 17083)	This is GetXMLRowSetName, a member of class IXMLDisplay.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	HasStatusField (see page 17060)	This is HasStatusField, a member of class IDataSetFields.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsFieldInUse (see page 17060)	This is IsFieldInUse, a member of class IDataSetFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEnableValidateFields (see page 17078)	This is SetEnableValidateFields, a member of class IValidateFields.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateDataSet (see page 17060)	This is ValidateDataSet, a member of class IDataSetFields.
	ValidateFields (see page 17078)	This is ValidateFields, a member of class IValidateFields.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.

R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 17055)	Provides access to an interface for working with the associated dataset.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayRows (see page 16509)	Limits the number of rows in the HTML table.
	EnableValidateFields (see page 17078)	This is EnableValidateFields, a member of class IValidateFields.
	HeadingAttributes (see page 16509)	Determines the display characteristics of the table's header.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsMultipleRecordView (see page 17080)	
R	IsMultipleRecordView (see page 17083)	
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RowAttributes (see page 16510)	Determines the display characteristics of the table's body.
	TableAttributes (see page 16510)	Determines the basic display characteristics of the table.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleFields (see page 17061)	This is VisibleFields, a member of class IDataSetFields.
R	VisibleFields (see page 17032)	Indicates the controls displayed by the HTML that the XML display group generates.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.
R	XMLBroker (see page 17056)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
R	XMLDataSet (see page 17056)	Indicates the TXMLDataSet (see page 17006) object that represents the dataset to which the Web item is bound.
R	XMLDisplayName (see page 17084)	This is XMLDisplayName, a member of class IXMLDisplay.
R	XMLRowSetName (see page 17084)	This is XMLRowSetName, a member of class IXMLDisplay.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLBroker (see page 17033)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
	XMLDataSetField (see page 17033)	Binds the XML display group to a nested dataset from the XML data packet.

99.1.4.2 TCustomDataGrid.Create Constructor

Creates and initializes an instance of TCustomDataGrid (see page 16499).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomDataGrid(TComponent * AOwner);
```

Description

Call Create to instantiate TCustomDataGrid (see page 16499) at runtime. TCustomDataGrid (see page 16499) components added at design time using the Web Page editor are instantiated automatically.

99.1.4.3 TCustomDataGrid.Destroy Destructor

Frees an instance of TCustomDataGrid (see page 16499).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomDataGrid();
```

Description

Do not call Destroy directly. Instead, call Free, which checks whether the object reference is nil before calling Destroy.

See Also

Free (see page 22300)

99.1.4.4 TCustomDataGrid Methods

The methods of the TCustomDataGrid class are listed here.

Protected Methods

	Name	Description
	FormatTable (see page 16506)	This is FormatTable, a member of class TCustomDataGrid.
V	ImplAddElements (see page 16506)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 16507)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 16507)	Provides the underlying implementation for the Content method.
V	ImplIsMultipleRecordView (see page 16508)	This is ImplIsMultipleRecordView, a member of class TCustomDataGrid.
	SetHeadingAttributes (see page 16508)	This is SetHeadingAttributes, a member of class TCustomDataGrid.
	SetRowAttributes (see page 16508)	This is SetRowAttributes, a member of class TCustomDataGrid.
	SetTableAttributes (see page 16508)	This is SetTableAttributes, a member of class TCustomDataGrid.

99.1.4.4.1 TCustomDataGrid.FormatTable Method

Pascal

```
function FormatTable(Layout: TLayoutWebContent; Options: TWebContentOptions): string;
```

C++

```
__fastcall AnsiString FormatTable(TLayoutWebContent Layout, TWebContentOptions Options);
```

Description

This is FormatTable, a member of class TCustomDataGrid.

99.1.4.4.2 TCustomDataGrid.ImplAddElements Method

Provides the underlying implementation for the AddElements method.

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); override;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

The AddElements method calls ImplAddElements to add any script declarations by calling the AddIntf interface. These declarations allow the generated display control to call script methods in response to user events by ensuring that the required methods are declared.

As implemented in TWebControlGroup, ImplAddElements does nothing. TWebControlGroup descendants override ImplAddElements to add any required declarations.

See Also

AddElements (see page 16878)

99.1.4.4.3 TCustomDataGrid.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass (see page 16878)

WebFieldControls (see page 16884)

99.1.4.4.4 TCustomDataGrid.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup, ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content (see page 16879)

AddElements (see page 16878)

ILayoutWebContent (see page 24774)

99.1.4.4.5 TCustomDataGrid.ImplIsMultipleRecordView Method

Pascal

```
function ImplIsMultipleRecordView: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplIsMultipleRecordView();
```

Description

This is ImplIsMultipleRecordView, a member of class TCustomDataGrid.

99.1.4.4.6 TCustomDataGrid.SetHeadingAttributes Method

Pascal

```
procedure SetHeadingAttributes(const Value: TGridRowAttributes);
```

C++

```
__fastcall SetHeadingAttributes(const TGridRowAttributes Value);
```

Description

This is SetHeadingAttributes, a member of class TCustomDataGrid.

99.1.4.4.7 TCustomDataGrid.SetRowAttributes Method

Pascal

```
procedure SetRowAttributes(const Value: TGridRowAttributes);
```

C++

```
__fastcall SetRowAttributes(const TGridRowAttributes Value);
```

Description

This is SetRowAttributes, a member of class TCustomDataGrid.

99.1.4.4.8 TCustomDataGrid.SetTableAttributes Method

Pascal

```
procedure SetTableAttributes(const Value: TGridAttributes);
```

C++

```
__fastcall SetTableAttributes(const TGridAttributes Value);
```

Description

This is SetTableAttributes, a member of class TCustomDataGrid.

99.1.4.5 TCustomDataGrid Properties

The properties of the TCustomDataGrid class are listed here.

Public Properties

	Name	Description
	DisplayRows (see page 16509)	Limits the number of rows in the HTML table.
	HeadingAttributes (see page 16509)	Determines the display characteristics of the table's header.
	RowAttributes (see page 16510)	Determines the display characteristics of the table's body.
	TableAttributes (see page 16510)	Determines the basic display characteristics of the table.

99.1.4.5.1 TCustomDataGrid.DisplayRows Property

Limits the number of rows in the HTML table.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to specify the number of rows that appear in the generated HTML table.

Note that the number of rows in the HTML table does not limit the number of records that end users can view and edit. When DisplayRows is less than the number of records in the data packet, the control generated by TNextPageButton (see page 16699) or TPriorPageButton (see page 16713) lets end users scroll through the records in the data packet.

See Also

MaxRecords (see page 26072)

TNextPageButton (see page 16699)

TPriorPageButton (see page 16713)

99.1.4.5.2 TCustomDataGrid.HeadingAttributes Property

Determines the display characteristics of the table's header.

Pascal

```
property HeadingAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes HeadingAttributes;
```

Description

Use HeadingAttributes to specify any display attributes that apply only to the heading of the generated HTML table. HeadingAttributes determines the options that are added to the <TR> tag that defines the header of the grid. These include the color of the header, the alignment of the caption for each column, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the rows of the grid, use the TableAttributes (see page 16510) or RowAttributes (see page 16510) property instead.

See Also

TableAttributes (see page 16510)

RowAttributes (see page 16510)

99.1.4.5.3 TCustomDataGrid.RowAttributes Property

Determines the display characteristics of the table's body.

Pascal

```
property RowAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes RowAttributes;
```

Description

Use RowAttributes to specify any display attributes that apply only to the rows of the generated HTML table. RowAttributes determines the options that are added to the <TR> tag for each row in the body of the grid. These include the color of the rows, the alignment of data within each cell, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the header of the grid, use the TableAttributes (see page 16510) or HeadingAttributes (see page 16509) property instead.

See Also

TableAttributes (see page 16510)

HeadingAttributes (see page 16509)

99.1.4.5.4 TCustomDataGrid.TableAttributes Property

Determines the basic display characteristics of the table.

Pascal

```
property TableAttributes: TGridAttributes;
```

C++

```
__property TGridAttributes TableAttributes;
```

Description

Use TableAttributes to specify values for the most common options on the HTML <TABLE> tag. TableAttributes lets you specify the alignment of the table in its HTML document, its color and border size, the offset of data within each table cell, as well as any custom options or style rules.

Note: To influence the display attributes of only the header of the grid, or only the rows in its body, use the TableAttributes or HeadingAttributes (see page 16509) property instead.

See Also

RowAttributes (see page 16510)

DisplayRows (see page 16509)

HeadingAttributes (see page 16509)

99.1.5 TCustomDataNavigator Class

TCustomDataNavigator assembles the HTML generated by a set of TDataSetButton (see page 16576) and TXMLButton (see page 16989) descendants.

Class Hierarchy

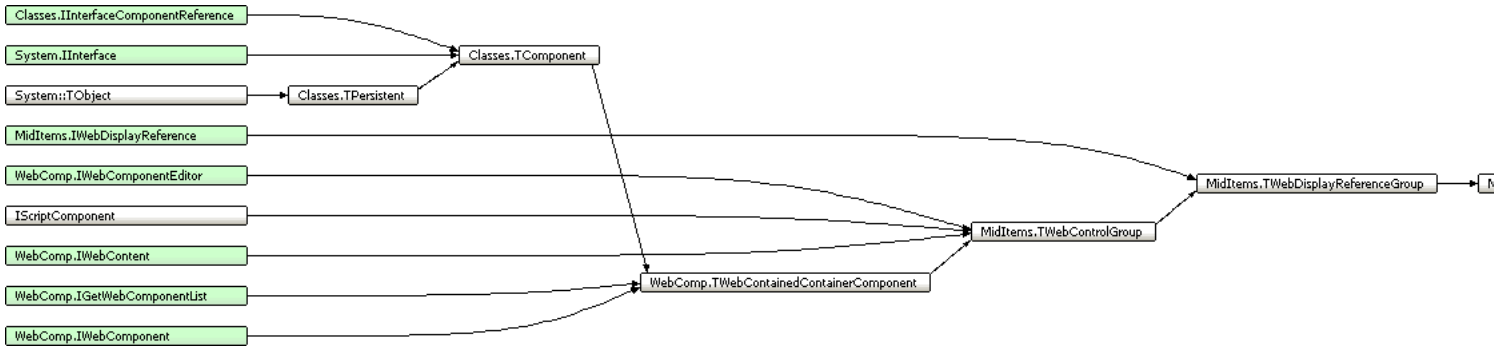

Pascal

```
TCustomDataNavigator = class(TXMLDisplayReferenceGroup);
```

C++

```
class TCustomDataNavigator : public TXMLDisplayReferenceGroup;
```

File

MidItems

Description

Use TCustomDataNavigator as a base class for Web items that generate the HTML to arrange a collection of button controls in a data form. Each TCustomDataNavigator object has an associated Web item that generates display controls in the form. The buttons arranged by the data navigator component manipulate the information these display controls show.

TCustomDataNavigator introduces properties that customize the appearance of the buttons it arranges.

Do not create instances of TCustomDataNavigator. TCustomDataNavigator is intended to act as a base class. Instead, instantiate a descendant class such as TDataNavigator (see page 16569).

See Also

TDataForm (see page 16554)

TFieldGroup (see page 16591)

TDataGrid (see page 16560)

IWebContent (see page 24792)

99.1.5.1 TCustomDataNavigator Members

The following tables list the members exposed by TCustomDataNavigator.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultButtons (see page 16516)	This is GetDefaultButtons, a member of class TCustomDataNavigator.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	GetWebDisplayComponent (see page 16920)	This is GetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
V	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
V	ImplAddElements (see page 17051)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 16516)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 16516)	Provides the underlying implementation for the Content method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetSubComponents (see page 16517)	Provides the underlying implementation for the GetSubComponents method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.

V	Notification (see page 16920)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
V	SetWebDisplayComponent (see page 16921)	This is SetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleButtons (see page 16517)	This is VisibleButtons, a member of class TCustomDataNavigator.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebDisplay (see page 16921)	This is WebDisplay, a member of class TWebDisplayReferenceGroup.
	WebDisplayComponent (see page 16921)	
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16877)	Creates and initializes an instance of TWebControlGroup (see page 16872).
V	Create (see page 16919)	Creates an instance of TWebDisplayReferenceGroup (see page 16915).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24742)	Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16877)	Frees an instance of TWebControlGroup (see page 16872).
V	Destroy (see page 16919)	Frees an instance of TWebDisplayReferenceGroup (see page 16915).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	GetWebDisplayComponent (see page 17081)	This is GetWebDisplayComponent, a member of class IWebDisplayReference.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16518)	Represents a tag option and its value in the generated TABLE tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16518)	Adds style attributes to the generated HTML table.
	StyleRule (see page 16519)	Applies a style in the page producer's style sheet to the group of buttons.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebDisplayComponent (see page 17081)	
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.
	XMLComponent (see page 17051)	Associates the group with a component that generates HTML display controls.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.5.2 TCustomDataNavigator Methods

The methods of the TCustomDataNavigator class are listed here.

Protected Methods

	Name	Description
	GetDefaultButtons (see page 16516)	This is GetDefaultButtons, a member of class TCustomDataNavigator.
	ImplCanAddClass (see page 16516)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 16516)	Provides the underlying implementation for the Content method.
	ImplGetSubComponents (see page 16517)	Provides the underlying implementation for the GetSubComponents method.
	VisibleButtons (see page 16517)	This is VisibleButtons, a member of class TCustomDataNavigator.

99.1.5.2.1 TCustomDataNavigator.GetDefaultButtons Method

Pascal

```
procedure GetDefaultButtons; virtual;
```

C++

```
virtual __fastcall GetDefaultButtons();
```

Description

This is GetDefaultButtons, a member of class TCustomDataNavigator.

99.1.5.2.2 TCustomDataNavigator.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass (see page 16878)

WebFieldControls (see page 16884)

99.1.5.2.3 TCustomDataNavigator.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup, ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content (see page 16879)

AddElements (see page 16878)

ILayoutWebContent (see page 24774)

99.1.5.2.4 TCustomDataNavigator.ImplGetSubComponents Method

Provides the underlying implementation for the GetSubComponents method.

Pascal

```
function ImplGetSubComponents: TObject; override;
```

C++

```
virtual __fastcall TObject * ImplGetSubComponents();
```

Description

The GetSubComponents method calls ImplGetSubComponents to retrieve the value of WebFieldControls as a TObject. TWebControlGroup descendants override this method if the list is not simply the value of WebFieldControls. For example, some TWebControlGroup descendants supply a default value (DefaultWebComponents) when the WebFieldControls property list is empty.

ImplGetSubComponents returns the TWebComponentList object that contains all subitems of the Web control group.

See Also

GetSubComponents (see page 16880)

DefaultWebComponents (see page 16883)

WebFieldControls (see page 16884)

99.1.5.2.5 TCustomDataNavigator.VisibleButtons Method

Pascal

```
function VisibleButtons: TWebComponentList;
```

C++

```
__fastcall TWebComponentList VisibleButtons();
```

Description

This is VisibleButtons, a member of class TCustomDataNavigator.

99.1.5.3 TCustomDataNavigator Properties

The properties of the TCustomDataNavigator class are listed here.

Public Properties

	Name	Description
	Custom (see page 16518)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 16518)	Adds style attributes to the generated HTML table.
	StyleRule (see page 16519)	Applies a style in the page producer's style sheet to the group of buttons.

99.1.5.3.1 TCustomDataNavigator.Custom Property

Represents a tag option and its value in the generated TABLE tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TCustomDataNavigator (see page 16511) generates an HTML table to lay out the buttons listed in the WebFieldControls (see page 16884) property. Use Custom to add options that qualify the TABLE tag. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add a border around the buttons and color the background blue, you can set Custom to

```
BORDER=2 BGCOLOR=BLUE
```

See Also

Style (see page 16518)

StyleRule (see page 16519)

WebFieldControls (see page 16884)

99.1.5.3.2 TCustomDataNavigator.Style Property

Adds style attributes to the generated HTML table.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

TCustomDataNavigator (see page 16511) generates an HTML table to lay out the buttons listed in the WebFieldControls (see page 16884) property. Use Style to add a STYLE option that qualifies the TABLE tag. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated TABLE tag as the value of a STYLE option.

See Also

Custom (see page 16518)

StyleRule (see page 16519)

WebFieldControls (see page 16884)

99.1.5.3.3 TCustomDataNavigator.StyleRule Property

Applies a style in the page producer's style sheet to the group of buttons.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

TCustomDataNavigator (see page 16511) generates an HTML table to lay out the buttons listed in the WebFieldControls (see page 16884) property. Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated table. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style (see page 16518) property.

See Also

Style (see page 16518)

Styles (see page 17127)

StylesFile (see page 17128)

WebFieldControls (see page 16884)

99.1.6 TCustomFieldGroup Class

TCustomFieldGroup is the base class for Web items that arrange information on an HTML form as a set of labeled fields.

Class Hierarchy

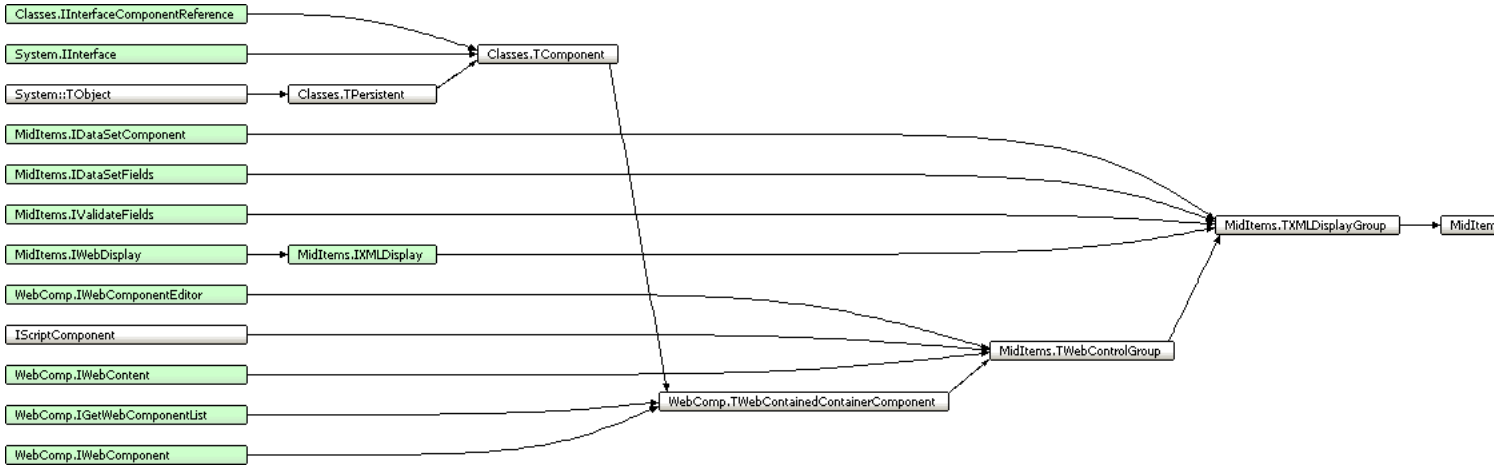

Pascal

```
TCustomFieldGroup = class(TXMLDisplayGroup);
```

C++

```
class TCustomFieldGroup : public TXMLDisplayGroup;
```

File

MidItems

Description

Use TCustomFieldGroup as a base class for Web items that display information in labeled fields on an HTML form. Each TCustomFieldGroup object maintains a list of Web items that generate the individual display fields. These Web items are restricted to classes that are appropriate for the specific TCustomFieldGroup descendant.

If the controls generated for TCustomFieldGroup by its Web items represent database information, you can bind the field group to an XML data packet or dataset field within an XML data packet. The Web items can use that binding to determine the source of the data they display.

TCustomFieldGroup introduces a number of properties that affect the appearance of the generated set of fields.

In order to let other Web items use the HTML that the TCustomFieldGroup object generates, TCustomFieldGroup inherits the IWebContent interface from TWebControlGroup (see page 16872).

Do not create instances of TCustomFieldGroup. Instead, use a descendant of TCustomFieldGroup such as TFieldGroup (see page 16591) or TQueryFieldGroup (see page 16736).

See Also

TFieldGroup (see page 16591)

IWebContent (see page 24792)

TQueryFieldGroup (see page 16736)

99.1.6.1 TCustomFieldGroup Members

The following tables list the members exposed by TCustomFieldGroup.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
V	CreateDefaultFields (see page 17025)	This is CreateDefaultFields, a member of class TXMLDisplayGroup.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FindField (see page 17026)	This is FindField, a member of class TXMLDisplayGroup.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 17026)	Returns the value of the DataSet property.
	GetDataSetField (see page 17026)	This is GetDataSetField, a member of class TXMLDisplayGroup.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetEnableValidateFields (see page 17026)	This is GetEnableValidateFields, a member of class TXMLDisplayGroup.
V	GetFieldsList (see page 17027)	This is GetFieldsList, a member of class TXMLDisplayGroup.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsMultipleRecordView (see page 17027)	This is GetIsMultipleRecordView, a member of class TXMLDisplayGroup.
V	GetLayoutAttributes (see page 16526)	Returns an object that specifies how the generated group of controls should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVisibleFields (see page 17027)	This is GetVisibleFields, a member of class TXMLDisplayGroup.
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	GetXMLBroker (see page 17027)	Returns the value of the XMLBroker property.
	GetXMLDataSet (see page 17028)	Returns the value of the XMLDataSet property.
	GetXMLDisplayName (see page 17028)	This is GetXMLDisplayName, a member of class TXMLDisplayGroup.
	GetXMLRowSetName (see page 17028)	This is GetXMLRowSetName, a member of class TXMLDisplayGroup.
	HasStatusField (see page 17028)	This is HasStatusField, a member of class TXMLDisplayGroup.
V	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
V	ImplAddElements (see page 17029)	Provides the underlying implementation for the AddElements method.

	ImplCanAddClass (see page 16881)	Provides the underlying implementation for the CanAddClass (see page 16878) method.
	ImplContent (see page 16526)	Provides the underlying implementation for the Content method.
	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	ImplGetSubComponents (see page 17029)	Provides the underlying implementation for the GetSubComponents method.
	ImplIsMultipleRecordView (see page 17029)	This is ImplIsMultipleRecordView, a member of class TXMLDisplayGroup.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsFieldInUse (see page 17030)	This is IsFieldInUse, a member of class TXMLDisplayGroup.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 17030)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
	SetDataSetField (see page 17030)	This is SetDataSetField, a member of class TXMLDisplayGroup.
	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEnableValidateFields (see page 17031)	This is SetEnableValidateFields, a member of class TXMLDisplayGroup.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	SetXMLBroker (see page 17031)	This is SetXMLBroker, a member of class TXMLDisplayGroup.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateDataSet (see page 17031)	This is ValidateDataSet, a member of class TXMLDisplayGroup.
	ValidateFields (see page 17031)	This is ValidateFields, a member of class TXMLDisplayGroup.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.
	XMLDataChange (see page 17031)	This is XMLDataChange, a member of class TXMLDisplayGroup.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
	EnableValidateFields (see page 17032)	This is EnableValidateFields, a member of class TXMLDisplayGroup.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.
R	XMLDataSet (see page 17033)	Indicates the TXMLDataSet object that represents the dataset to which the Web item is bound.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16525)	Creates and initializes an instance of TCustomFieldGroup (see page 16519).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 16525)	Frees an instance of TCustomFieldGroup (see page 16519).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDataSet (see page 17054)	Returns the value of the DataSet (see page 17055) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnableValidateFields (see page 17078)	This is GetEnableValidateFields, a member of class IValidateFields.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 17060)	This is GetFieldsList, a member of class IDataSetFields.

	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsMultipleRecordView (see page 17079)	This is GetIsMultipleRecordView, a member of class IWebDisplay.
	GetIsMultipleRecordView (see page 17082)	This is GetIsMultipleRecordView, a member of class IXMLDisplay.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	GetVisibleFields (see page 17060)	This is GetVisibleFields, a member of class IDataSetFields.
	GetXMLBroker (see page 17054)	Returns the value of the XMLBroker (see page 17056) property.
	GetXMLDataSet (see page 17055)	Returns the value of the XMLDataSet (see page 17056) property.
	GetXMLDisplayName (see page 17083)	This is GetXMLDisplayName, a member of class IXMLDisplay.
	GetXMLRowSetName (see page 17083)	This is GetXMLRowSetName, a member of class IXMLDisplay.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	HasStatusField (see page 17060)	This is HasStatusField, a member of class IDataSetFields.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsFieldInUse (see page 17060)	This is IsFieldInUse, a member of class IDataSetFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEnableValidateFields (see page 17078)	This is SetEnableValidateFields, a member of class IValidateFields.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateDataSet (see page 17060)	This is ValidateDataSet, a member of class IDataSetFields.
	ValidateFields (see page 17078)	This is ValidateFields, a member of class IValidateFields.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16527)	Represents a tag option and its value in the generated TABLE tag.
R	DataSet (see page 17055)	Provides access to an interface for working with the associated dataset.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EnableValidateFields (see page 17078)	This is EnableValidateFields, a member of class IValidateFields.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsMultipleRecordView (see page 17080)	
R	IsMultipleRecordView (see page 17083)	
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16527)	Adds style attributes to the generated set of fields.
	StyleRule (see page 16528)	Applies a style in the page producer's style sheet to the table.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleFields (see page 17061)	This is VisibleFields, a member of class IDataSetFields.
R	VisibleFields (see page 17032)	Indicates the controls displayed by the HTML that the XML display group generates.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.
R	XMLBroker (see page 17056)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
R	XMLDataSet (see page 17056)	Indicates the TXMLDataSet (see page 17006) object that represents the dataset to which the Web item is bound.
R	XMLDisplayName (see page 17084)	This is XMLDisplayName, a member of class IXMLDisplay.
R	XMLRowSetName (see page 17084)	This is XMLRowSetName, a member of class IXMLDisplay.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLBroker (see page 17033)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
	XMLDataSetField (see page 17033)	Binds the XML display group to a nested dataset from the XML data packet.

99.1.6.2 TCustomFieldGroup.Create Constructor

Creates and initializes an instance of TCustomFieldGroup (see page 16519).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomFieldGroup(TComponent * AOwner);
```

Description

Call Create to instantiate TCustomFieldGroup (see page 16519) at runtime. TCustomFieldGroup (see page 16519) components added at design time using the Web Page editor are instantiated automatically.

99.1.6.3 TCustomFieldGroup.Destroy Destructor

Frees an instance of TCustomFieldGroup (see page 16519).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomFieldGroup();
```

Description

Do not call Destroy directly. Instead, call Free, which checks whether the object reference is not nil before calling Destroy.

See Also

Free (see page 22300)

99.1.6.4 TCustomFieldGroup Methods

The methods of the TCustomFieldGroup class are listed here.

Protected Methods

	Name	Description
	GetLayoutAttributes (see page 16526)	Returns an object that specifies how the generated group of controls should look.
	ImplContent (see page 16526)	Provides the underlying implementation for the Content method.

99.1.6.4.1 TCustomFieldGroup.GetLayoutAttributes Method

Returns an object that specifies how the generated group of controls should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content method) to take into account layout properties of the child component.

As implemented by TWebControlGroup, GetLayoutAttributes returns nil (Delphi) or NULL (C++), because TWebControlGroup descendants not make use of layout attributes.

See Also

Content (see page 16879)

99.1.6.4.2 TCustomFieldGroup.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup,

ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content (see page 16879)

AddElements (see page 16878)

ILayoutWebContent (see page 24774)

99.1.6.5 TCustomFieldGroup Properties

The properties of the TCustomFieldGroup class are listed here.

Public Properties

	Name	Description
	Custom (see page 16527)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 16527)	Adds style attributes to the generated set of fields.
	StyleRule (see page 16528)	Applies a style in the page producer's style sheet to the table.

99.1.6.5.1 TCustomFieldGroup.Custom Property

Represents a tag option and its value in the generated TABLE tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TCustomFieldGroup (see page 16519) generates an HTML table to manage the layout of the HTML controls that its Web items generate. Use Custom to add options that qualify the TABLE tag. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to surround the set of fields with a border and give them a blue background, set Custom to

```
BORDER=2 BGCOLOR=BLUE
```

See Also

Style (see page 16527)

StyleRule (see page 16528)

99.1.6.5.2 TCustomFieldGroup.Style Property

Adds style attributes to the generated set of fields.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

TCustomFieldGroup (see page 16519) generates an HTML table to lay out the fields that its Web items generate. Use Style to add a STYLE option that qualifies the TABLE tag. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated TABLE tag as the value of a STYLE option.

See Also

Custom (see page 16527)

StyleRule (see page 16528)

99.1.6.5.3 TCustomFieldGroup.StyleRule Property

Applies a style in the page producer's style sheet to the table.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated set of fields. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style (see page 16527) property.

See Also

Style (see page 16527)

Styles (see page 17127)

StylesFile (see page 17128)

99.1.7 TCustomLayoutGroup Class

TCustomLayoutGroup is the base class for Web items that arrange content produced by arbitrary Web items.

Class Hierarchy

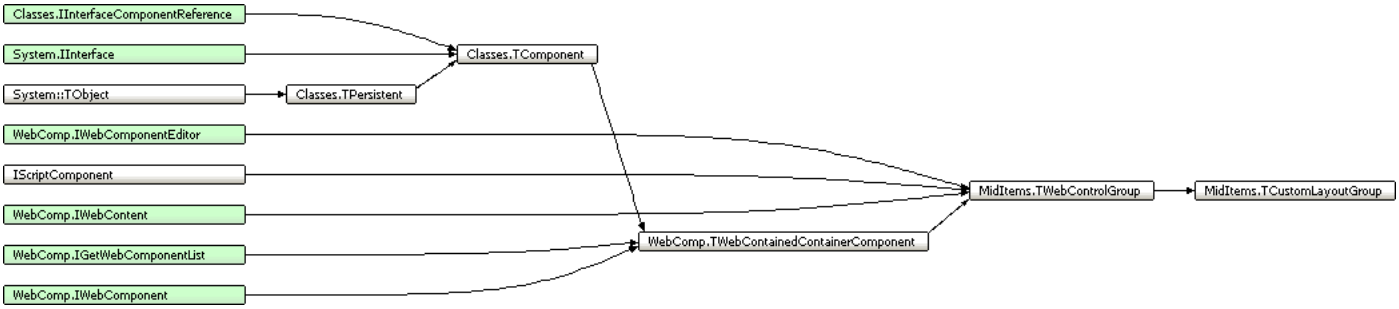

Pascal

```
TCustomLayoutGroup = class(TWebControlGroup);
```

C++

```
class TCustomLayoutGroup : public TWebControlGroup;
```

File

MidItems

Description

Use TCustomLayoutGroup as a base class for Web items that are used purely for layout. There are few restrictions on what Web items can use a layout group because it has no special content that it displays. It merely imposes a layout on content generated by other Web items.

TCustomLayoutGroup maintains a list of Web items that display information in the final HTML document, but imposes no restrictions of its own on the possible types of Web items. Instead, it restricts those Web items to the classes that its parent allows. In order to let the parent component use the HTML that TCustomLayoutGroup generates, TCustomLayoutGroup inherits the IWebContent interface from TWebControlGroup (see page 16872).

Do not create instances of TCustomLayoutGroup. Instead, use a descendant of TCustomLayoutGroup such as TLayoutGroup (see page 16687).

See Also

TLayoutGroup (see page 16687)

IWebContent (see page 24792)

99.1.7.1 TCustomLayoutGroup Members

The following tables list the members exposed by TCustomLayoutGroup.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.

	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetLayoutAttributes (see page 16534)	Returns an object that specifies how the generated group of controls should look.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
	ImplCanAddClass (see page 16534)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 16535)	Provides the underlying implementation for the Content method.
	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16533)	Creates and initializes an instance of TCustomLayoutGroup (see page 16528).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 16533)	Frees an instance of TCustomLayoutGroup (see page 16528).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16535)	Represents a tag option and its value in the generated TABLE tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayColumns (see page 16536)	Specifies the number of columns the layout group uses to lay out its child controls.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16536)	Adds style attributes to the generated HTML table.
	StyleRule (see page 16537)	Applies a style in the page producer's style sheet to the group of buttons.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.7.2 TCustomLayoutGroup.Create Constructor

Creates and initializes an instance of TCustomLayoutGroup (see page 16528).

Pascal

```
constructor Create(AComponent: TComponent); override;
```

C++

```
virtual __fastcall TCustomLayoutGroup(TComponent * AComponent);
```

Description

Call Create to instantiate TCustomLayoutGroup (see page 16528) at runtime. TCustomLayoutGroup (see page 16528) components added at design time using the Web Page editor are instantiated automatically.

99.1.7.3 TCustomLayoutGroup.Destroy Destructor

Frees an instance of TCustomLayoutGroup (see page 16528).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomLayoutGroup();
```

Description

Do not call Destroy directly. Instead, call Free, which checks whether the layout group is nil before calling Destroy.

See Also

Free (see page 22300)

99.1.7.4 TCustomLayoutGroup Methods

The methods of the TCustomLayoutGroup class are listed here.

Protected Methods

	Name	Description
	GetLayoutAttributes (see page 16534)	Returns an object that specifies how the generated group of controls should look.
	ImplCanAddClass (see page 16534)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 16535)	Provides the underlying implementation for the Content method.

99.1.7.4.1 TCustomLayoutGroup.GetLayoutAttributes Method

Returns an object that specifies how the generated group of controls should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content method) to take into account layout properties of the child component.

As implemented by TWebControlGroup, GetLayoutAttributes returns nil (Delphi) or NULL (C++), because TWebControlGroup descendants not make use of layout attributes.

See Also

Content (see page 16879)

99.1.7.4.2 TCustomLayoutGroup.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass (see page 16878)

WebFieldControls (see page 16884)

99.1.7.4.3 TCustomLayoutGroup.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup, ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content (see page 16879)

AddElements (see page 16878)

ILayoutWebContent (see page 24774)

99.1.7.5 TCustomLayoutGroup Properties

The properties of the TCustomLayoutGroup class are listed here.

Public Properties

	Name	Description
	Custom (see page 16535)	Represents a tag option and its value in the generated TABLE tag.
	DisplayColumns (see page 16536)	Specifies the number of columns the layout group uses to lay out its child controls.
	Style (see page 16536)	Adds style attributes to the generated HTML table.
	StyleRule (see page 16537)	Applies a style in the page producer's style sheet to the group of buttons.

99.1.7.5.1 TCustomLayoutGroup.Custom Property

Represents a tag option and its value in the generated TABLE tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TCustomLayoutGroup (see page 16528) generates an HTML table to lay out the controls generated by the Web items in its WebFieldControls (see page 16884) property. Use Custom to add options that qualify the TABLE tag. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add a border around the group of controls and color the background blue, you can set Custom to

```
BORDER=2 BGCOLOR=BLUE
```

See Also

Style (see page 16536)

StyleRule (see page 16537)

WebFieldControls (see page 16884)

DisplayColumns (see page 16536)

99.1.7.5.2 TCustomLayoutGroup.DisplayColumns Property

Specifies the number of columns the layout group uses to lay out its child controls.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

TCustomLayoutGroup (see page 16528) generates an HTML table to lay out the controls generated by the Web items in its WebFieldControls (see page 16884) property. It adds the HTML generated by each Web item listed in WebFieldControls (see page 16884) to a cell of the table, filling the cells out row by row until it displays all the Web items. DisplayColumns determines the number of cells in each row.

See Also

WebFieldControls (see page 16884)

99.1.7.5.3 TCustomLayoutGroup.Style Property

Adds style attributes to the generated HTML table.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

TCustomLayoutGroup (see page 16528) generates an HTML table to lay out the controls generated by the Web items in WebFieldControls (see page 16884). Use Style to add a STYLE option that qualifies the TABLE tag. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated TABLE tag as the value of a STYLE option.

See Also

Custom (see page 16535)

StyleRule (see page 16537)

WebFieldControls (see page 16884)

99.1.7.5.4 TCustomLayoutGroup.StyleRule Property

Applies a style in the page producer's style sheet to the group of buttons.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

TCustomLayoutGroup (see page 16528) generates an HTML table to lay out the controls generated by its Web items. Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated table. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style (see page 16536) property.

See Also

Style (see page 16536)

Styles (see page 19830)

StylesFile (see page 19831)

WebFieldControls (see page 16884)

99.1.8 TCustomQueryButtons Class

TCustomQueryButtons assembles the HTML generated by a set of TQueryButton (see page 16718) descendants.

Class Hierarchy

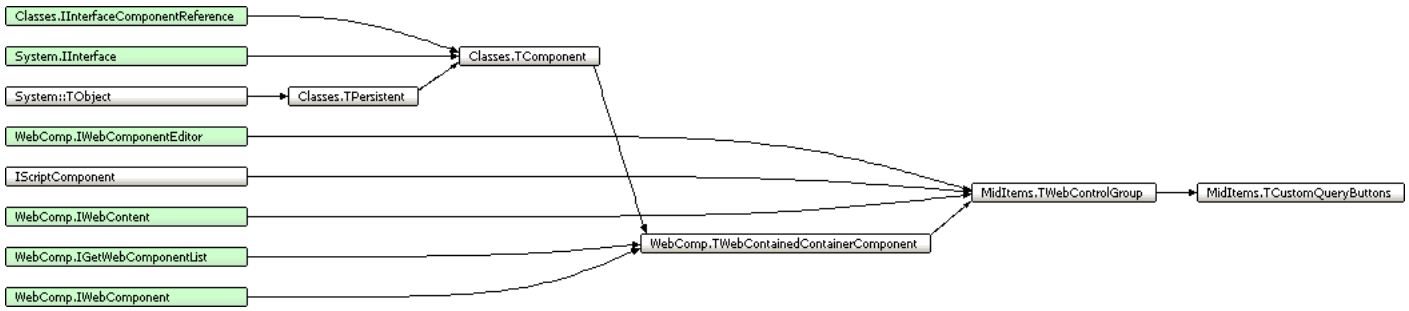

Pascal

```
TCustomQueryButtons = class(TWebControlGroup);
```

C++

```
class TCustomQueryButtons : public TWebControlGroup;
```

File

MidItems

Description

Use TCustomQueryButtons as a base class for Web items that generate the HTML to arrange a collection of button controls in a query form. The buttons arranged by the query buttons component manipulate information shown by display controls in the parent Query form.

TCustomQueryButtons introduces properties that customize the appearance of the buttons it arranges.

Do not create instances of TCustomQueryButtons. TCustomQueryButtons is intended to act as a base class. Instead, instantiate a descendant class such as TQueryButtons (see page 16730).

See Also

TQueryForm (see page 16745)

TQueryFieldGroup (see page 16736)

IWebContent (see page 24792)

99.1.8.1 TCustomQueryButtons Members

The following tables list the members exposed by TCustomQueryButtons.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultButtons (see page 16542)	This is GetDefaultButtons, a member of class TCustomQueryButtons.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.

	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	ImplAddElements (see page 16542)	Provides the underlying implementation for the AddElements method.
	ImplCanAddClass (see page 16543)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 16543)	Provides the underlying implementation for the Content method.
	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleButtons (see page 16543)	This is VisibleButtons, a member of class TCustomQueryButtons.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.

R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16877)	Creates and initializes an instance of TWebControlGroup (see page 16872).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24742)	Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16877)	Frees an instance of TWebControlGroup (see page 16872).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16544)	Represents a tag option and its value in the generated TABLE tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16544)	Adds style attributes to the generated HTML table.
	StyleRule (see page 16545)	Applies a style in the page producer's style sheet to the group of buttons.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.8.2 TCustomQueryButtons Methods

The methods of the TCustomQueryButtons class are listed here.

Protected Methods

	Name	Description
	GetDefaultButtons (see page 16542)	This is GetDefaultButtons, a member of class TCustomQueryButtons.
	ImplAddElements (see page 16542)	Provides the underlying implementation for the AddElements method.
	ImplCanAddClass (see page 16543)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 16543)	Provides the underlying implementation for the Content method.
	VisibleButtons (see page 16543)	This is VisibleButtons, a member of class TCustomQueryButtons.

99.1.8.2.1 TCustomQueryButtons.GetDefaultButtons Method

Pascal

```
procedure GetDefaultButtons; virtual;
```

C++

```
virtual __fastcall GetDefaultButtons();
```

Description

This is GetDefaultButtons, a member of class TCustomQueryButtons.

99.1.8.2.2 TCustomQueryButtons.ImplAddElements Method

Provides the underlying implementation for the AddElements method.

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); override;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

The AddElements method calls ImplAddElements to add any script declarations by calling the AddIntf interface. These declarations allow the generated display control to call script methods in response to user events by ensuring that the required methods are declared.

As implemented in TWebControlGroup, ImplAddElements does nothing. TWebControlGroup descendants override ImplAddElements to add any required declarations.

See Also

AddElements (see page 16878)

99.1.8.2.3 TCustomQueryButtons.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass (see page 16878)

WebFieldControls (see page 16884)

99.1.8.2.4 TCustomQueryButtons.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup, ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content (see page 16879)

AddElements (see page 16878)

ILayoutWebContent (see page 24774)

99.1.8.2.5 TCustomQueryButtons.VisibleButtons Method

Pascal

```
function VisibleButtons: TWebComponentList;
```

C++

```
__fastcall TWebComponentList VisibleButtons();
```

Description

This is VisibleButtons, a member of class TCustomQueryButtons.

99.1.8.3 TCustomQueryButtons Properties

The properties of the TCustomQueryButtons class are listed here.

Public Properties

	Name	Description
	Custom (see page 16544)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 16544)	Adds style attributes to the generated HTML table.
	StyleRule (see page 16545)	Applies a style in the page producer's style sheet to the group of buttons.

99.1.8.3.1 TCustomQueryButtons.Custom Property

Represents a tag option and its value in the generated TABLE tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TCustomQueryButtons (see page 16537) generates an HTML table to lay out the buttons listed in the WebFieldControls (see page 16884) property. Use Custom to add options that qualify the TABLE tag. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add a border around the query buttons and color the background blue, you can set Custom to

```
BORDER=2 BGCOLOR=BLUE
```

See Also

Style (see page 16544)

StyleRule (see page 16545)

WebFieldControls (see page 16884)

99.1.8.3.2 TCustomQueryButtons.Style Property

Adds style attributes to the generated HTML table.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

TCustomQueryButtons (see page 16537) generates an HTML table to lay out the buttons listed in the WebFieldControls (see page 16884) property. Use Style to add a STYLE option that qualifies the TABLE tag. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated TABLE tag as the value of a STYLE option.

See Also

Custom (see page 16544)

StyleRule (see page 16545)

WebFieldControls (see page 16884)

99.1.8.3.3 TCustomQueryButtons.StyleRule Property

Applies a style in the page producer's style sheet to the group of buttons.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

TCustomQueryButtons (see page 16537) generates an HTML table to lay out the buttons listed in the WebFieldControls (see page 16884) property. Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated table. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style (see page 16544) property.

See Also

Style (see page 16544)

Styles (see page 17127)

StylesFile (see page 17128)

WebFieldControls (see page 16884)

99.1.9 TCustomQueryForm Class

TCustomQueryForm is the base class for Web items that generate HTML forms for letting users enter application-defined values.

Class Hierarchy

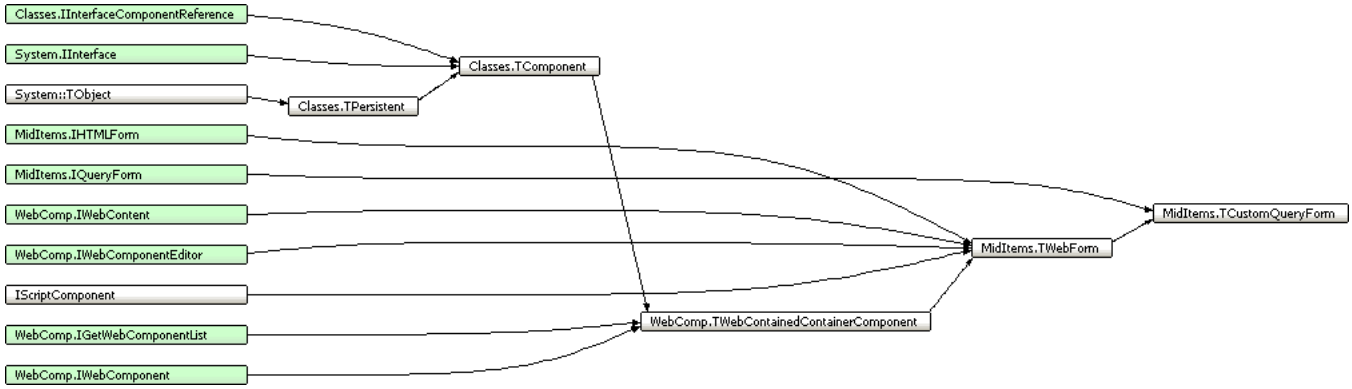

Pascal

```
TCustomQueryForm = class(TWebForm, IQueryForm);
```

C++

```
class TCustomQueryForm : public TWebForm, public IQueryForm;
```

File

MidItems

Description

Use TCustomQueryForm as a base class for components that generate an HTML form that displays application-defined values and submits them to an InternetExpress application. TCustomQueryForm introduces properties for specifying the method and target URL of messages submitted by the form.

When building an HTML form, TCustomQueryForm makes use of content generated by the components listed in its WebComponents (see page 16934) property. These components can include Web items that display application-defined values (TQueryFieldGroup (see page 16736)) or that generate buttons to operate on those values (TQueryButtons (see page 16730)). They can also include components that manage the layout of other HTML controls (TLayoutGroup (see page 16687)).

In order to let an InternetExpress page producer use the generated HTML query form, TCustomQueryForm inherits support for the IWebContent interface from TWebForm (see page 16922).

Do not create instances of TCustomQueryForm. Instead, create descendants of TCustomQueryForm, such as TQueryForm (see page 16745).

See Also

TQueryForm (see page 16745)

IWebContent (see page 24792)

TInetXPageProducer (see page 17145)

99.1.9.1 TCustomQueryForm Members

The following tables list the members exposed by TCustomQueryForm.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16927)	Used internally to generate any javascript declarations needed by the form.
	AssignStringsCallback (see page 16551)	This is AssignStringsCallback, a member of class TCustomQueryForm.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16928)	Specifies whether a Web item of a specified type can be added to the WebComponents (see page 16934) property list.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16928)	Returns the HTML that defines the HTML form.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultWebComponents (see page 16929)	Returns nil (Delphi) or NULL (C++).
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetHTMLFormName (see page 16929)	Returns the name of the generated HTML form.
V	GetHTMLFormTag (see page 16551)	This is GetHTMLFormTag, a member of class TCustomQueryForm.
	GetHTMLFormVarName (see page 16929)	Returns the name of the javascript variable that refers to the generated form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16930)	Returns the object that specifies how the generated form should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16930)	Returns the value of the WebComponents (see page 16934) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVisibleFields (see page 16930)	Returns the value of the VisibleFields (see page 16934) property.
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
V	ImplAddElements (see page 16552)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 16552)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 16552)	Provides the underlying implementation for the Content method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	MethodString (see page 16553)	This is MethodString, a member of class TCustomQueryForm.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.

V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16932)	Prevents an application from trying to assign a default set of Web items on the form.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	VisibleFields (see page 16934)	Lists the Web items whose generated content appears in the HTML form.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
	AssignStrings (see page 16551)	This is AssignStrings, a member of class TCustomQueryForm.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16550)	Creates and initializes an instance of TCustomQueryForm (see page 16545).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16926)	Frees an instance of TWebForm (see page 16922).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHTMLFormName (see page 17065)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 17065)	Returns the name of the javascript variable that refers to the generated form.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Action (see page 16553)	Specifies the path info portion of the URL to which the query submits request messages.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16933)	Represents a tag option and its value in the generated FORM tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLFormName (see page 17066)	Indicates the name of the generated HTML form.
R	HTMLFormVarName (see page 17066)	Indicates the name of the javascript variable that represents the HTML form.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
	Method (see page 16554)	This is Method, a member of class TCustomQueryForm.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16933)	Adds style attributes to the generated form.
	StyleRule (see page 16934)	Applies a style in the page producer's style sheet to the button.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebComponents (see page 16934)	Maintains a list of components that generate HTML content for the TWebForm (see page 16922) descendant.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.9.2 TCustomQueryForm.Create Constructor

Creates and initializes an instance of TCustomQueryForm (see page 16545).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomQueryForm(TComponent * AOwner);
```

Description

Call Create to instantiate a query form at runtime. Query forms added at design time using the Web Page editor are instantiated automatically.

99.1.9.3 TCustomQueryForm Methods

The methods of the TCustomQueryForm class are listed here.

Protected Methods

	Name	Description
	AssignStringsCallback (see page 16551)	This is AssignStringsCallback, a member of class TCustomQueryForm.
V	GetHTMLFormTag (see page 16551)	This is GetHTMLFormTag, a member of class TCustomQueryForm.
V	ImplAddElements (see page 16552)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 16552)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 16552)	Provides the underlying implementation for the Content method.
	MethodString (see page 16553)	This is MethodString, a member of class TCustomQueryForm.

Public Methods

	Name	Description
	AssignStrings (see page 16551)	This is AssignStrings, a member of class TCustomQueryForm.

99.1.9.3.1 TCustomQueryForm.AssignStrings Method

Pascal

```
procedure AssignStrings(Value: TStrings);
```

C++

```
__fastcall AssignStrings(TStrings Value);
```

Description

This is AssignStrings, a member of class TCustomQueryForm.

99.1.9.3.2 TCustomQueryForm.AssignStringsCallback Method

Pascal

```
procedure AssignStringsCallback(AComponent: TComponent);
```

C++

```
__fastcall AssignStringsCallback(TComponent * AComponent);
```

Description

This is AssignStringsCallback, a member of class TCustomQueryForm.

99.1.9.3.3 TCustomQueryForm.GetHTMLFormTag Method

Pascal

```
function GetHTMLFormTag(Options: TWebContentOptions): string; virtual;
```

C++

```
virtual __fastcall AnsiString GetHTMLFormTag(TWebContentOptions Options);
```

Description

This is GetHTMLFormTag, a member of class TCustomQueryForm.

99.1.9.3.4 TCustomQueryForm.ImplAddElements Method

Provides the underlying implementation for the AddElements method.

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); override;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

The AddElements method calls ImplAddElements to add any javascript declarations by calling the AddIntf interface. These declarations allow the generated display control to call javascript methods in response to user events by ensuring that the required methods are declared.

As implemented in TWebForm, ImplAddElements does nothing. TWebForm descendants can override ImplAddElements to add any required declarations.

See Also

AddElements (see page 16927)

99.1.9.3.5 TCustomQueryForm.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebForm descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebForm instance to which the component will be added.

In TWebForm, ImplCanAddClass is abstract or, in C++ terminology, pure virtual. TWebForm descendants must override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML forms.

See Also

CanAddClass (see page 16928)

WebComponents (see page 16934)

99.1.9.3.6 TCustomQueryForm.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML form. ImplContent is anabstract or, in C++ terminology, pure virtual method in TWebForm. TWebForm descendants override it to generate the HTML form.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebForm should format the HTML control it generates. Typically, this parameter is nil (Delphi) or NULL (C++) for forms.

ImplContent returns an HTML form definition.

See Also

Content (see page 16928)

AddElements (see page 16927)

99.1.9.3.7 TCustomQueryForm.MethodString Method

Pascal

```
function MethodString: string;
```

C++

```
__fastcall AnsiString MethodString();
```

Description

This is MethodString, a member of class TCustomQueryForm.

99.1.9.4 TCustomQueryForm Properties

The properties of the TCustomQueryForm class are listed here.

Public Properties

	Name	Description
	Action (see page 16553)	Specifies the path info portion of the URL to which the query submits request messages.
	Method (see page 16554)	This is Method, a member of class TCustomQueryForm.

99.1.9.4.1 TCustomQueryForm.Action Property

Specifies the path info portion of the URL to which the query submits request messages.

Pascal

```
property Action: string;
```

C++

```
__property AnsiString Action;
```

Description

Use Action to specify the target URL for messages that the query form submits. Action supplies the PathInfo property of Web request messages from the query form. This value is matched to the PathInfo property of a Web action item when determining what object responds to those messages.

See Also

TCustomQueryForm (see page 16545)

PathInfo (see page 15185)

PathInfo (see page 15138)

99.1.9.4.2 TCustomQueryForm.Method Property

Pascal

```
property Method: TFormMethod;
```

C++

```
__property TFormMethod Method;
```

Description

This is Method, a member of class TCustomQueryForm.

99.1.10 TDataForm Class

TDataForm generates an HTML form that displays database information and lets users apply updates.

Class Hierarchy

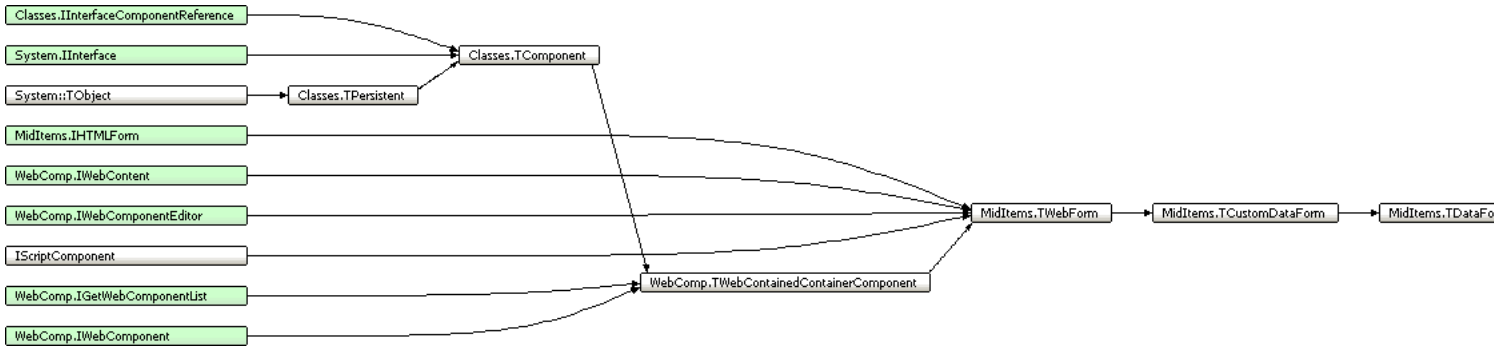

Pascal

```
TDataForm = class(TCustomDataForm);
```

C++

```
class TDataForm : public TCustomDataForm;
```

File

MidItems

Description

Add TDataForm to the Web items of an Internet Express page producer (TInetXPageProducer) to add an HTML form that displays database information that users can view and edit.

Data forms generate an HTML form that contains controls which display information from an XML data packet. Buttons (see page 2289) on the form manipulate that data, and submit updates back to the XML broker that supplies the XML data packet.

TDataForm generates the HTML FORM tag and assembles the HTML generated by its Web items. It restricts the Web items that

generate the contents of the form to components that generate data displays (TDataGrid (see page 16560) or TFieldGroup (see page 16591)), components that contain buttons which operate on those displays (TDataNavigator (see page 16569)), and components that manage the layout of HTML controls (TLayoutGroup (see page 16687)). At design time, you can add these Web items to the form using the Web page editor.

See Also

TDataGrid (see page 16560)

TBasePageItemsProducer (see page 18376)

TInetXPageProducer (see page 17145)

TFieldGroup (see page 16591)

TlayoutGroup

TDataNavigator (see page 16569)

TApplyUpdatesButton (see page 16480)

Using the Web Page Editor

99.1.10.1 TDataForm Members

The following tables list the members exposed by TDataForm.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16927)	Used internally to generate any javascript declarations needed by the form.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16928)	Specifies whether a Web item of a specified type can be added to the WebComponents (see page 16934) property list.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16928)	Returns the HTML that defines the HTML form.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultWebComponents (see page 16929)	Returns nil (Delphi) or NULL (C++).
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetHTMLFormName (see page 16929)	Returns the name of the generated HTML form.
V	GetHTMLFormTag (see page 16498)	This is GetHTMLFormTag, a member of class TCustomDataForm.
	GetHTMLFormVarName (see page 16929)	Returns the name of the javascript variable that refers to the generated form.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16930)	Returns the object that specifies how the generated form should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16930)	Returns the value of the WebComponents (see page 16934) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVisibleFields (see page 16930)	Returns the value of the VisibleFields (see page 16934) property.
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
V	ImplAddElements (see page 16931)	Provides the underlying implementation for the AddElements (see page 16927) method.
V	ImplCanAddClass (see page 16498)	Provides the underlying implementation for the CanAddClass method.
A	ImplCanAddClass (see page 16931)	Provides the underlying implementation for the CanAddClass (see page 16928) method.
V	ImplContent (see page 16499)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 16932)	Provides the underlying implementation for the Content (see page 16928) method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16932)	Prevents an application from trying to assign a default set of Web items on the form.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.

	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	VisibleFields (see page 16934)	Lists the Web items whose generated content appears in the HTML form.
	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 16926)	Creates and initializes an instance of TWebForm (see page 16922).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 24742)	Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 16926)	Frees an instance of TWebForm (see page 16922).
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.

	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHTMLFormName (see page 17065)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 17065)	Returns the name of the javascript variable that refers to the generated form.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLFormName (see page 17066)	Indicates the name of the generated HTML form.
R	HTMLFormVarName (see page 17066)	Indicates the name of the javascript variable that represents the HTML form.

	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebComponents (see page 16934)	Maintains a list of components that generate HTML content for the TWebForm (see page 16922) descendant.

Published Properties

	Name	Description
	Custom (see page 16559)	Represents a tag option and its value in the generated FORM tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16559)	Adds style attributes to the generated form.
	StyleRule (see page 16560)	Applies a style in the page producer's style sheet to the button.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.10.2 TDataForm Properties

The properties of the TDataForm class are listed here.

Published Properties

	Name	Description
	Custom (see page 16559)	Represents a tag option and its value in the generated FORM tag.
	Style (see page 16559)	Adds style attributes to the generated form.
	StyleRule (see page 16560)	Applies a style in the page producer's style sheet to the button.

99.1.10.2.1 TDataForm.Custom Property

Represents a tag option and its value in the generated FORM tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to represent one or more options that qualify the FORM tag generated by the TWebForm descendant. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to specify that the form includes files with the text it submits and indicate that the page returned after submitting the form goes into a separate window, use a Custom value such as the following:

```
ENCTYPE="multipart/form-data" TARGET="ResultFrame"
```

See Also

Style (see page 16933)

99.1.10.2.2 TDataForm.Style Property

Adds style attributes to the generated form.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to add a STYLE option that qualifies the FORM tag generated by the TWebForm descendant. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated FORM tag as the value of a STYLE option.

See Also

Custom (see page 16933)

StyleRule (see page 16934)

99.1.10.2.3 TDataForm.StyleRule Property

Applies a style in the page producer's style sheet to the button.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated form. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style property.

See Also

Style (see page 16933)

Styles (see page 17127)

StylesFile (see page 17128)

99.1.11 TDataGrid Class

TDataGrid generates an HTML table to display a dataset in an XML data packet.

Class Hierarchy

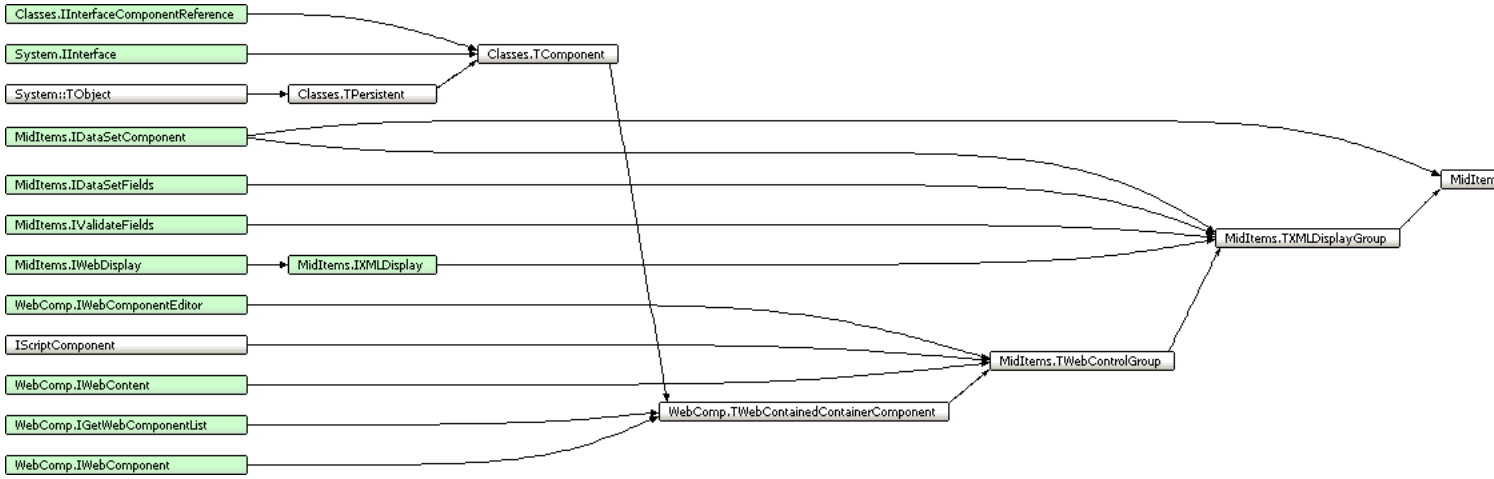

Pascal

```
TDataGrid = class(TCustomDataGrid);
```

C++

```
class TDataGrid : public TCustomDataGrid;
```

File

MidItems

Description

Add TDataGrid to the Web items of a TDataForm (see page 16554) component (or a layout group that is a child of a TDataForm (see page 16554) component.) TDataGrid generates an HTML table that displays multiple records from a dataset as a set of rows and columns.

Each TDataGrid object maintains a list of Web items that generate the individual fields (columns) of the table. These Web items can be from the following classes: TTextColumn (see page 16849), TSelectOptionColumn, TTextAreaColumn (see page 16838), and TStatusColumn (see page 16826). You can bind the data grid to an XML data packet or dataset field within an XML data packet. The column Web items can then use that binding to determine the source of the data they display.

You can also use the properties of TDataGrid to affect the appearance of the generated table.

See Also

TSelectOptionsColumn (see page 16814)

TStatusColumn (see page 16826)

TTextAreaColumn (see page 16838)

TTextColumn (see page 16849)

TDataForm (see page 16554)

TLayoutGroup (see page 16687)

TWebDataDisplay (see page 16884)

Using the Web Page Editor

99.1.11.1 TDataGrid Members

The following tables list the members exposed by TDataGrid.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
V	CreateDefaultFields (see page 17025)	This is CreateDefaultFields, a member of class TXMLDisplayGroup.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FindField (see page 17026)	This is FindField, a member of class TXMLDisplayGroup.
	FormatTable (see page 16506)	This is FormatTable, a member of class TCustomDataGrid.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 17026)	Returns the value of the DataSet property.
	GetDataSetField (see page 17026)	This is GetDataSetField, a member of class TXMLDisplayGroup.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetEnableValidateFields (see page 17026)	This is GetEnableValidateFields, a member of class TXMLDisplayGroup.
V	GetFieldsList (see page 17027)	This is GetFieldsList, a member of class TXMLDisplayGroup.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsMultipleRecordView (see page 17027)	This is GetIsMultipleRecordView, a member of class TXMLDisplayGroup.
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVisibleFields (see page 17027)	This is GetVisibleFields, a member of class TXMLDisplayGroup.
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	GetXMLBroker (see page 17027)	Returns the value of the XMLBroker property.
	GetXMLDataSet (see page 17028)	Returns the value of the XMLDataSet property.

	GetXMLDisplayName (see page 17028)	This is GetXMLDisplayName, a member of class TXMLDisplayGroup.
	GetXMLRowSetName (see page 17028)	This is GetXMLRowSetName, a member of class TXMLDisplayGroup.
	HasStatusField (see page 17028)	This is HasStatusField, a member of class TXMLDisplayGroup.
V	ImplAddElements (see page 16506)	Provides the underlying implementation for the AddElements method.
V	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
V	ImplAddElements (see page 17029)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 16507)	Provides the underlying implementation for the CanAddClass method.
V	ImplCanAddClass (see page 16881)	Provides the underlying implementation for the CanAddClass (see page 16878) method.
V	ImplContent (see page 16507)	Provides the underlying implementation for the Content method.
V	ImplContent (see page 16882)	Provides the underlying implementation for the Content (see page 16879) method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
V	ImplGetSubComponents (see page 17029)	Provides the underlying implementation for the GetSubComponents method.
V	ImplIsMultipleRecordView (see page 16508)	This is ImplIsMultipleRecordView, a member of class TCustomDataGrid.
V	ImplIsMultipleRecordView (see page 17029)	This is ImplIsMultipleRecordView, a member of class TXMLDisplayGroup.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsFieldInUse (see page 17030)	This is IsFieldInUse, a member of class TXMLDisplayGroup.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 17030)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
	SetDataSetField (see page 17030)	This is SetDataSetField, a member of class TXMLDisplayGroup.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEnableValidateFields (see page 17031)	This is SetEnableValidateFields, a member of class TXMLDisplayGroup.
	SetHeadingAttributes (see page 16508)	This is SetHeadingAttributes, a member of class TCustomDataGrid.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetRowAttributes (see page 16508)	This is SetRowAttributes, a member of class TCustomDataGrid.
	SetTableAttributes (see page 16508)	This is SetTableAttributes, a member of class TCustomDataGrid.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	SetXMLBroker (see page 17031)	This is SetXMLBroker, a member of class TXMLDisplayGroup.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateDataSet (see page 17031)	This is ValidateDataSet, a member of class TXMLDisplayGroup.
	ValidateFields (see page 17031)	This is ValidateFields, a member of class TXMLDisplayGroup.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.
	XMLDataChange (see page 17031)	This is XMLDataChange, a member of class TXMLDisplayGroup.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
	EnableValidateFields (see page 17032)	This is EnableValidateFields, a member of class TXMLDisplayGroup.
	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.
	XMLDataSet (see page 17033)	Indicates the TXMLDataSet object that represents the dataset to which the Web item is bound.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 16505)	Creates and initializes an instance of TCustomDataGrid (see page 16499).
	Create (see page 16877)	Creates and initializes an instance of TWebControlGroup (see page 16872).
	Create (see page 17024)	Creates and initializes an instance of TXMLDisplayGroup (see page 17019).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 24742)	Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 16506)	Frees an instance of TCustomDataGrid (see page 16499).
	Destroy (see page 16877)	Frees an instance of TWebControlGroup (see page 16872).

V	Destroy (see page 17024)	Frees an instance of TXMLDisplayGroup (see page 17019).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDataSet (see page 17054)	Returns the value of the DataSet (see page 17055) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnableValidateFields (see page 17078)	This is GetEnableValidateFields, a member of class IValidateFields.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 17060)	This is GetFieldsList, a member of class IDataSetFields.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsMultipleRecordView (see page 17079)	This is GetIsMultipleRecordView, a member of class IWebDisplay.
	GetIsMultipleRecordView (see page 17082)	This is GetIsMultipleRecordView, a member of class IXMLDisplay.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	GetVisibleFields (see page 17060)	This is GetVisibleFields, a member of class IDataSetFields.
	GetXMLBroker (see page 17054)	Returns the value of the XMLBroker (see page 17056) property.
	GetXMLDataSet (see page 17055)	Returns the value of the XMLDataSet (see page 17056) property.
	GetXMLDisplayName (see page 17083)	This is GetXMLDisplayName, a member of class IXMLDisplay.
	GetXMLRowSetName (see page 17083)	This is GetXMLRowSetName, a member of class IXMLDisplay.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	HasStatusField (see page 17060)	This is HasStatusField, a member of class IDataSetFields.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsFieldInUse (see page 17060)	This is IsFieldInUse, a member of class IDataSetFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEnableValidateFields (see page 17078)	This is SetEnableValidateFields, a member of class IValidateFields.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateDataSet (see page 17060)	This is ValidateDataSet, a member of class IDataSetFields.
	ValidateFields (see page 17078)	This is ValidateFields, a member of class IValidateFields.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 17055)	Provides access to an interface for working with the associated dataset.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EnableValidateFields (see page 17078)	This is EnableValidateFields, a member of class IValidateFields.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsMultipleRecordView (see page 17080)	
R	IsMultipleRecordView (see page 17083)	
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleFields (see page 17061)	This is VisibleFields, a member of class IDataSetFields.
R	VisibleFields (see page 17032)	Indicates the controls displayed by the HTML that the XML display group generates.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.
R	XMLBroker (see page 17056)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
R	XMLDataSet (see page 17056)	Indicates the TXMLDataSet (see page 17006) object that represents the dataset to which the Web item is bound.
R	XMLDisplayName (see page 17084)	This is XMLDisplayName, a member of class IXMLDisplay.
R	XMLRowSetName (see page 17084)	This is XMLRowSetName, a member of class IXMLDisplay.

Published Properties

	Name	Description
	DisplayRows (see page 16567)	Limits the number of rows in the HTML table.
	HeadingAttributes (see page 16567)	Determines the display characteristics of the table's header.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RowAttributes (see page 16568)	Determines the display characteristics of the table's body.
	TableAttributes (see page 16568)	Determines the basic display characteristics of the table.

	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLBroker (see page 17033)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
	XMLDataSetField (see page 17033)	Binds the XML display group to a nested dataset from the XML data packet.

99.1.11.2 TDataGrid Properties

The properties of the TDataGrid class are listed here.

Published Properties

	Name	Description
	DisplayRows (see page 16567)	Limits the number of rows in the HTML table.
	HeadingAttributes (see page 16567)	Determines the display characteristics of the table's header.
	RowAttributes (see page 16568)	Determines the display characteristics of the table's body.
	TableAttributes (see page 16568)	Determines the basic display characteristics of the table.

99.1.11.2.1 TDataGrid.DisplayRows Property

Limits the number of rows in the HTML table.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to specify the number of rows that appear in the generated HTML table.

Note that the number of rows in the HTML table does not limit the number of records that end users can view and edit. When DisplayRows is less than the number of records in the data packet, the control generated by TNextPageButton or TPriorPageButton lets end users scroll through the records in the data packet.

See Also

MaxRecords (see page 26072)

TNextPageButton (see page 16699)

TPriorPageButton (see page 16713)

99.1.11.2.2 TDataGrid.HeadingAttributes Property

Determines the display characteristics of the table's header.

Pascal

```
property HeadingAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes HeadingAttributes;
```

Description

Use HeadingAttributes to specify any display attributes that apply only to the heading of the generated HTML table. HeadingAttributes determines the options that are added to the <TR> tag that defines the header of the grid. These include the color of the header, the alignment of the caption for each column, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the rows of the grid, use the TableAttributes or RowAttributes property instead.

See Also

TableAttributes (see page 16510)

RowAttributes (see page 16510)

99.1.11.2.3 TDataGrid.RowAttributes Property

Determines the display characteristics of the table's body.

Pascal

```
property RowAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes RowAttributes;
```

Description

Use RowAttributes to specify any display attributes that apply only to the rows of the generated HTML table. RowAttributes determines the options that are added to the <TR> tag for each row in the body of the grid. These include the color of the rows, the alignment of data within each cell, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the header of the grid, use the TableAttributes or HeadingAttributes property instead.

See Also

TableAttributes (see page 16510)

HeadingAttributes (see page 16509)

99.1.11.2.4 TDataGrid.TableAttributes Property

Determines the basic display characteristics of the table.

Pascal

```
property TableAttributes: TGridAttributes;
```

C++

```
__property TGridAttributes TableAttributes;
```

Description

Use TableAttributes to specify values for the most common options on the HTML <TABLE> tag. TableAttributes lets you specify the alignment of the table in its HTML document, its color and border size, the offset of data within each table cell, as well as any custom options or style rules.

Note: To influence the display attributes of only the header of the grid, or only the rows in its body, use the TableAttributes or HeadingAttributes property instead.

See Also

RowAttributes (see page 16510)

DisplayRows (see page 16509)

HeadingAttributes (see page 16509)

99.1.12 TDataNavigator Class

TDataNavigator assembles the HTML generated by a set of TDataSetButton (see page 16576) and TXMLButton (see page 16989) descendants.

Class Hierarchy

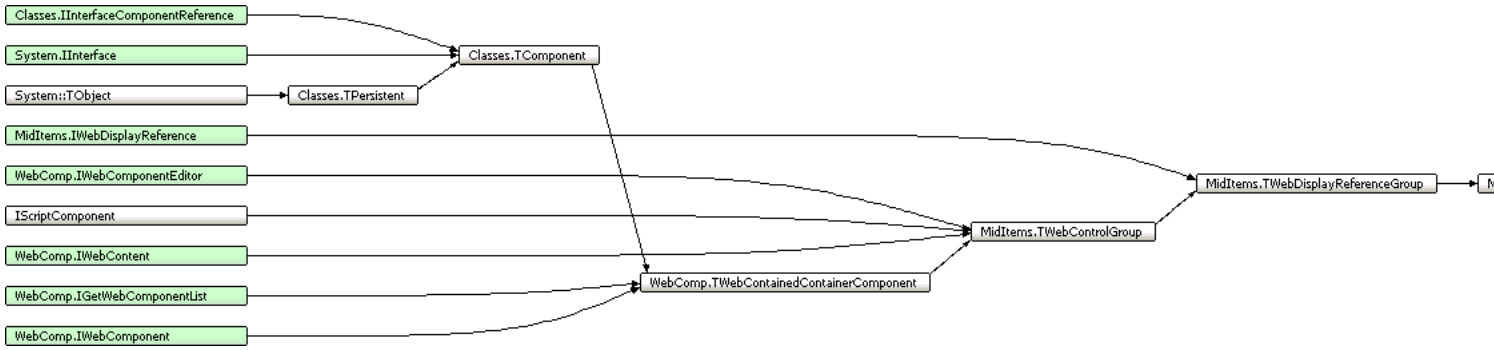

Pascal

```
TDataNavigator = class(TCustomDataNavigator);
```

C++

```
class TDataNavigator : public TCustomDataNavigator;
```

File

MidItems

Description

Add TDataNavigator as the child of a TDataForm (see page 16554) component (or a layout group that is a child of a TDataForm (see page 16554) component) to enable the generated HTML form to navigate the displayed data and apply updates that end users make to the data. TDataNavigator can be associated with the Web item that displays the data, such as TDataGrid (see page 16560) or TFieldGroup (see page 16591).

TDataNavigator generates HTML containing the controls generated by its children. These children must be descendants of TDataSetButton (see page 16576) or TXMLButton (see page 16989), such as TInsertButton (see page 16677) and TApplyUpdatesButton (see page 16480).

Use the properties of TDataNavigator to customize the appearance of the group of buttons.

See Also

TDataForm (see page 16554)

TFieldGroup (see page 16591)

TDataGrid (see page 16560)

Using the Web Page Editor

99.1.12.1 TDataNavigator Members

The following tables list the members exposed by TDataNavigator.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultButtons (see page 16516)	This is GetDefaultButtons, a member of class TCustomDataNavigator.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	GetWebDisplayComponent (see page 16920)	This is GetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
V	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
V	ImplAddElements (see page 17051)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 16516)	Provides the underlying implementation for the CanAddClass method.
V	ImplCanAddClass (see page 16881)	Provides the underlying implementation for the CanAddClass (see page 16878) method.
V	ImplContent (see page 16516)	Provides the underlying implementation for the Content method.
V	ImplContent (see page 16882)	Provides the underlying implementation for the Content (see page 16879) method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetSubComponents (see page 16517)	Provides the underlying implementation for the GetSubComponents method.
V	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16920)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
V	SetWebDisplayComponent (see page 16921)	This is SetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleButtons (see page 16517)	This is VisibleButtons, a member of class TCustomDataNavigator.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebDisplay (see page 16921)	This is WebDisplay, a member of class TWebDisplayReferenceGroup.
	WebDisplayComponent (see page 16921)	
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16877)	Creates and initializes an instance of TWebControlGroup (see page 16872).
V	Create (see page 16919)	Creates an instance of TWebDisplayReferenceGroup (see page 16915).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24742)	Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16877)	Frees an instance of TWebControlGroup (see page 16872).
V	Destroy (see page 16919)	Frees an instance of TWebDisplayReferenceGroup (see page 16915).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.

	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	GetWebDisplayComponent (see page 17081)	This is GetWebDisplayComponent, a member of class IWebDisplayReference.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebDisplayComponent (see page 17081)	
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Custom (see page 16574)	Represents a tag option and its value in the generated TABLE tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.

	Style (see page 16574)	Adds style attributes to the generated HTML table.
	StyleRule (see page 16575)	Applies a style in the page producer's style sheet to the group of buttons.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLComponent (see page 16575)	Associates the group with a component that generates HTML display controls.

99.1.12.2 TDataNavigator Properties

The properties of the TDataNavigator class are listed here.

Published Properties

	Name	Description
	Custom (see page 16574)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 16574)	Adds style attributes to the generated HTML table.
	StyleRule (see page 16575)	Applies a style in the page producer's style sheet to the group of buttons.
	XMLComponent (see page 16575)	Associates the group with a component that generates HTML display controls.

99.1.12.2.1 TDataNavigator.Custom Property

Represents a tag option and its value in the generated TABLE tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TCustomDataNavigator generates an HTML table to lay out the buttons listed in the WebFieldControls property. Use Custom to add options that qualify the TABLE tag. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add a border around the buttons and color the background blue, you can set Custom to

```
BORDER=2 BGCOLOR=BLUE
```

See Also

Style (see page 16518)

StyleRule (see page 16519)

WebFieldControls (see page 16884)

99.1.12.2.2 TDataNavigator.Style Property

Adds style attributes to the generated HTML table.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

TCustomDataNavigator generates an HTML table to lay out the buttons listed in the WebFieldControls property. Use Style to add

a STYLE option that qualifies the TABLE tag. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated TABLE tag as the value of a STYLE option.

See Also

Custom (see page 16518)

StyleRule (see page 16519)

WebFieldControls (see page 16884)

99.1.12.2.3 TDataNavigator.StyleRule Property

Applies a style in the page producer's style sheet to the group of buttons.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

TCustomDataNavigator generates an HTML table to lay out the buttons listed in the WebFieldControls property. Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated table. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style property.

See Also

Style (see page 16518)

Styles (see page 17127)

StylesFile (see page 17128)

WebFieldControls (see page 16884)

99.1.12.2.4 TDataNavigator.XMLComponent Property

Associates the group with a component that generates HTML display controls.

Pascal

```
property XMLComponent: TComponent;
```

C++

```
__property TComponent * XMLComponent;
```

Description

Use XMLComponent to associate the group of buttons with a Web item that generates the display controls those buttons influence. Typical values of XMLComponent are TFieldGroup or TDataGrid.

If all the buttons arranged by the TXMLDisplayReferenceGroup object operate on the same set of display controls, it is most efficient to specify the Web item generating these controls as the value of XMLComponent. Then, each button uses this value to determine its associated display controls when its XMLUseParent property is true. Any buttons that do not operate on the display controls generated by XMLComponent can set their own XMLComponent property to another Web item.

See Also

XMLComponent (see page 17045)

XMLUseParent (see page 17045)

TFieldGroup (see page 16591)

TDataGrid (see page 16560)

TQueryFieldGroup (see page 16736)

99.1.13 TDataSetButton Class

TDataSetButton is the base class for Web items that create HTML buttons for navigating or editing data.

Class Hierarchy

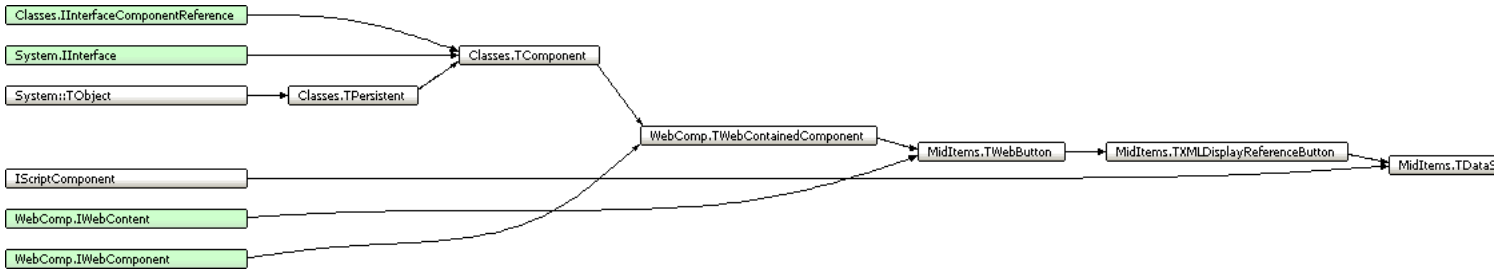

Pascal

```
TDataSetButton = class(TXMLDisplayReferenceButton, IScriptComponent);
```

C++

```
class TDataSetButton : public TXMLDisplayReferenceButton, public IScriptComponent;
```

File

MidItems

Description

Use TDataSetButton as a base class for components that generate the HTML statements defining a specialized button control for working with the data displayed by another Web item. TDataSetButton objects are used by TDataNavigator (see page 16569), which organizes the generated buttons.

TDataSetButton defines properties and methods for

Generating the HTML code to define the button control.

Associating the button with another Web item that generates HTML for displaying the data that the button manipulates.

Specifying tag attributes of the HTML button control, including those defined by a style sheet.

Fitting the component into a hierarchy of Web items, each of which generates part of a larger HTML document.

In order to allow other Web items to use the HTML generated by the button control, TDataSetButton inherits support for the IWebContent interface from TWebButton (see page 16863).

Do not create instances of TDataSetButton. TDataSetButton relies on anabstract or, in C++ terminology, pure virtual method to generate the HTML statements defining a button control. Instead, create descendants of TDataSetButton, such as TNextButton

(see page 16694), TPostButton (see page 16704), TInsertButton (see page 16677), and so on.

See Also

TFirstButton (see page 16651)

TNextButton (see page 16694)

TPriorButton (see page 16709)

TLastButton (see page 16682)

TNextPageButton (see page 16699)

TPriorPageButton (see page 16713)

TInsertButton (see page 16677)

TDeleteButton (see page 16586)

TUndoButton (see page 16858)

TPostButton (see page 16704)

IWebContent (see page 24792)

TDataNavigator (see page 16569)

99.1.13.1 TDataSetButton Members

The following tables list the members exposed by TDataSetButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 16581)	This is DefaultCaption, a member of class TDataSetButton.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	XMLMethodName (see page 16581)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16582)	This is AddElements, a member of class TDataSetButton.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16868)	Returns the HTML that defines the button control.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetCaption (see page 16582)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDisplayComponent (see page 17043)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 17043)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
V	GetMethodName (see page 16582)	This is GetMethodName, a member of class TDataSetButton.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16582)	This is GetSubComponents, a member of class TDataSetButton.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 17043)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	ImplAddElements (see page 16583)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 16583)	Provides the underlying implementation for the Content method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 17044)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayComponent (see page 17044)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 17044)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.

R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.
R	XMLDisplay (see page 17045)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16868)	Creates and initializes an instance of TWebButton (see page 16863).
V	Create (see page 17042)	Creates and initializes an instance of TXMLDisplayReferenceButton (see page 17037).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 17042)	Frees an instance of TXMLDisplayReferenceButton (see page 17037).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.

	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Caption (see page 16584)	Specifies the caption that appears on the button.
	Custom (see page 16584)	Represents a tag option and its value in the generated INPUT tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16584)	Adds style attributes to the generated button control.
	StyleRule (see page 16585)	Applies a style in the page producer's style sheet to the button.

	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLComponent (see page 16585)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 16586)	Specifies whether the button is associated with its parent's display object.

99.1.13.2 TDataSetButton Fields

The fields of the TDataSetButton class are listed here.

Protected Fields

	Name	Description
	DefaultCaption (see page 16581)	This is DefaultCaption, a member of class TDataSetButton.
	XMLMethodName (see page 16581)	This is XMLMethodName, a member of class TDataSetButton.

99.1.13.2.1 TDataSetButton.DefaultCaption Field

Pascal

```
DefaultCaption: string;
```

C++

```
AnsiString DefaultCaption;
```

Description

This is DefaultCaption, a member of class TDataSetButton.

99.1.13.2.2 TDataSetButton.XMLMethodName Field

Pascal

```
XMLMethodName: string;
```

C++

```
AnsiString XMLMethodName;
```

Description

This is XMLMethodName, a member of class TDataSetButton.

99.1.13.3 TDataSetButton Methods

The methods of the TDataSetButton class are listed here.

Protected Methods

	Name	Description
	AddElements (see page 16582)	This is AddElements, a member of class TDataSetButton.
V	GetCaption (see page 16582)	Returns the value of the Caption property.
V	GetMethodName (see page 16582)	This is GetMethodName, a member of class TDataSetButton.
	GetSubComponents (see page 16582)	This is GetSubComponents, a member of class TDataSetButton.
V	ImplAddElements (see page 16583)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 16583)	Provides the underlying implementation for the Content method.

99.1.13.3.1 TDataSetButton.AddElements Method

Pascal

```
procedure AddElements(AddIntf: IAddScriptElements);
```

C++

```
__fastcall AddElements(IAddScriptElements AddIntf);
```

Description

This is AddElements, a member of class TDataSetButton.

99.1.13.3.2 TDataSetButton.GetCaption Method

Returns the value of the Caption property.

Pascal

```
function GetCaption: string; override;
```

C++

```
virtual __fastcall AnsiString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption property.

See Also

Caption (see page 16870)

SetCaption (see page 16870)

99.1.13.3.3 TDataSetButton.GetMethodName Method

Pascal

```
function GetMethodName: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetMethodName();
```

Description

This is GetMethodName, a member of class TDataSetButton.

99.1.13.3.4 TDataSetButton.GetSubComponents Method

Pascal

```
function GetSubComponents: TObject;
```

C++

```
__fastcall TObject * GetSubComponents();
```

Description

This is GetSubComponents, a member of class TDataSetButton.

99.1.13.3.5 TDataSetButton.ImplAddElements Method

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); virtual;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

This is ImplAddElements, a member of class TDataSetButton.

99.1.13.3.6 TDataSetButton.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML that defines the button control. TWebButton introduces ImplContent as an abstract or, in C++ terminology, pure virtual method. TWebButton descendants must override ImplContent to generate the definition of their particular button control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web button should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the button control.

See Also

Content (see page 16868)

ILayoutWebContent (see page 24774)

99.1.13.4 TDataSetButton Properties

The properties of the TDataSetButton class are listed here.

Published Properties

	Name	Description
	Caption (see page 16584)	Specifies the caption that appears on the button.
	Custom (see page 16584)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 16584)	Adds style attributes to the generated button control.
	StyleRule (see page 16585)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 16585)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 16586)	Specifies whether the button is associated with its parent's display object.

99.1.13.4.1 TDataSetButton.Caption Property

Specifies the caption that appears on the button.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the caption that appears on the generated HTML button. The value of Caption is used as the value of a VALUE parameter on the HTML INPUT tag generated to implement the button.

See Also

Custom (see page 16871)

99.1.13.4.2 TDataSetButton.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to represent one or more options that qualify the INPUT tag generated by the Web button. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

```
ALIGN=TOP TABINDEX=2
```

See Also

Caption (see page 16870)

Style (see page 16871)

99.1.13.4.3 TDataSetButton.Style Property

Adds style attributes to the generated button control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to add a STYLE option that qualifies the INPUT tag generated by the Web button. The Style option consists of one or

more attribute descriptions such as

`font-family: arial; font-weight: bold; font-size: 10px`

These attributes are added to the generated INPUT tag as the value of a STYLE option.

See Also

Custom (see page 16871)

StyleRule (see page 16872)

99.1.13.4.4 TDataSetButton.StyleRule Property

Applies a style in the page producer's style sheet to the button.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated button control. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the button's Style property.

See Also

Style (see page 16871)

Styles (see page 17127)

StylesFile (see page 17128)

Styles (see page 19830)

StylesFile (see page 19831)

99.1.13.4.5 TDataSetButton.XMLComponent Property

Associates the button with the component that generates the display control it influences.

Pascal

```
property XMLComponent: TComponent;
```

C++

```
__property TComponent * XMLComponent;
```

Description

Set XMLComponent to specify the Web item that generates a display control that the button influences.

Note: Typically, descendants of TXMLDisplayReferenceButton are used by a descendant of TXMLDisplayReferenceGroup, which has its own XMLComponent property. If the button influences this same component, you need not set the XMLComponent property. Instead, setting the XMLUseParent property to true (the default) sets XMLComponent to the value of the parent's XMLComponent property.

See Also

XMLUseParent (see page 17045)

99.1.13.4.6 TDataSetButton.XMLUseParent Property

Specifies whether the button is associated with its parent's display object.

Pascal

```
property XMLUseParent: Boolean;
```

C++

```
__property Boolean XMLUseParent;
```

Description

Use XMLUseParent to indicate how the dataset button determines the component that it influences. When XMLUseParent is false, the XMLComponent property must be set to indicate the display component. When XMLUseParent is true, the button influences the component specified by its parent's XMLComponent property.

The button's parent is the Web item that uses the button to assemble its own HTML content (typically a TDataNavigator or TQueryButtons object). If the parent does not have an associated display component, setting XMLUseParent to true sets the XMLComponent property to nil (Delphi) or NULL (C++).

See Also

XMLComponent (see page 17045)

XMLComponent (see page 17051)

99.1.14 TDeleteButton Class

TDeleteButton generates an HTML control that deletes a record from an XML data packet.

Class Hierarchy

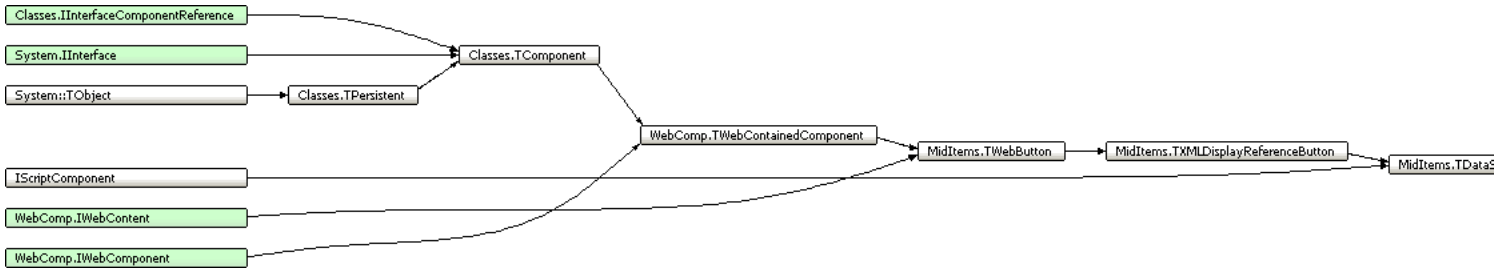

Pascal

```
TDeleteButton = class(TDataSetButton);
```

C++

```
class TDeleteButton : public TDataSetButton;
```

File

MidItems

Description

Add TDeleteButton as the child of a TDataNavigator (see page 16569) component (or a layout group that is a child of a TDataNavigator (see page 16569) component) to enable the generated HTML form to delete records. The resulting HTML button's action adds the record deletion to the emerging delta packet, which can then be submitted using an apply updates

button.

Use the properties of TDeleteButton to customize the appearance of the button and associate it with a Web item. The associated Web item displays the records that the delete button can delete.

See Also

TXMLBroker (see page 26049)

TDataNavigator (see page 16569)

TLayoutGroup (see page 16687)

TApplyUpdatesButton (see page 16480)

99.1.14.1 TDeleteButton Members

The following tables list the members exposed by TDeleteButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 16581)	This is DefaultCaption, a member of class TDataSetButton.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	XMLMethodName (see page 16581)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16582)	This is AddElements, a member of class TDataSetButton.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16868)	Returns the HTML that defines the button control.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetCaption (see page 16582)	Returns the value of the Caption property.
V	GetCaption (see page 16869)	Returns the value of the Caption (see page 16870) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDisplayComponent (see page 17043)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 17043)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
V	GetMethodName (see page 16582)	This is GetMethodName, a member of class TDataSetButton.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16582)	This is GetSubComponents, a member of class TDataSetButton.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 17043)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	ImplAddElements (see page 16583)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 16583)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 16869)	Provides the underlying implementation for the Content (see page 16868) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 17044)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayComponent (see page 17044)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 17044)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.
R	XMLDisplay (see page 17045)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.

	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16591)	Creates and initializes an instance of TDeleteButton (see page 16586).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 17042)	Frees an instance of TXMLDisplayReferenceButton (see page 17037).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 16870)	Specifies the caption that appears on the button.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16871)	Represents a tag option and its value in the generated INPUT tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16871)	Adds style attributes to the generated button control.
	StyleRule (see page 16872)	Applies a style in the page producer's style sheet to the button.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	XMLComponent (see page 17045)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 17045)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 16584)	Specifies the caption that appears on the button.
	Custom (see page 16584)	Represents a tag option and its value in the generated INPUT tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16584)	Adds style attributes to the generated button control.
	StyleRule (see page 16585)	Applies a style in the page producer's style sheet to the button.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLComponent (see page 16585)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 16586)	Specifies whether the button is associated with its parent's display object.

99.1.14.2 TDeleteButton.Create Constructor

Creates and initializes an instance of TDeleteButton (see page 16586).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDeleteButton(TComponent * AOwner);
```

Description

Call Create to instantiate TDeleteButton (see page 16586) at runtime. Buttons (see page 2289) added at design time using the Web Page editor are instantiated automatically.

99.1.15 TFieldGroup Class

TFieldGroup arranges database information on an HTML form as a set of labeled fields.

Class Hierarchy

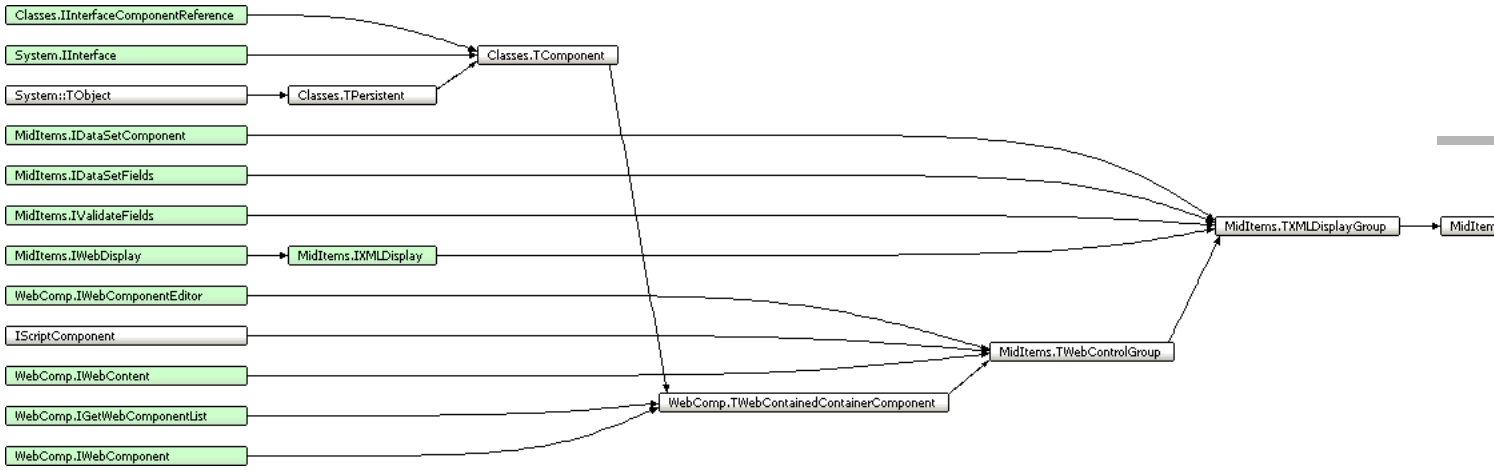

Pascal

```
TFieldGroup = class(TCustomFieldGroup);
```

C++

```
class TFieldGroup : public TCustomFieldGroup;
```

File

MidItems

Description

Add TFieldGroup to a data form component (TDataForm (see page 16554)) or a layout group (TLayoutGroup (see page 16687)) that is a child for a data form. TFieldGroup generates the HTML to display information from an XML data packet in labeled fields.

TFieldGroup maintains a list of Web items that generate the individual display fields. These Web items are restricted to the

following classes: TFieldText (see page 16631), TFieldTextArea (see page 16640), TFieldSelectOptions (see page 16612), TFieldRadioGroup (see page 16599), and TFieldStatus (see page 16624). You can bind the field group to an XML data packet or dataset field within an XML data packet. The Web items can then use that binding to determine the source of the data they display.

You can also use the properties of TFieldGroup to affect the overall appearance of the set of fields.

See Also

TDataForm (see page 16554)

TLayoutGroup (see page 16687)

TFieldRadioGroup (see page 16599)

TFieldStatus (see page 16624)

TFieldSelectOptions (see page 16612)

TFieldText (see page 16631)

TFieldTextArea (see page 16640)

Using the Web Page Editor

99.1.15.1 TFieldGroup Members

The following tables list the members exposed by TFieldGroup.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
V	CreateDefaultFields (see page 17025)	This is CreateDefaultFields, a member of class TXMLDisplayGroup.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FindField (see page 17026)	This is FindField, a member of class TXMLDisplayGroup.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.

	GetDataSet (see page 17026)	Returns the value of the DataSet property.
	GetDataSetField (see page 17026)	This is GetDataSetField, a member of class TXMLDisplayGroup.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetEnableValidateFields (see page 17026)	This is GetEnableValidateFields, a member of class TXMLDisplayGroup.
V	GetFieldsList (see page 17027)	This is GetFieldsList, a member of class TXMLDisplayGroup.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsMultipleRecordView (see page 17027)	This is GetIsMultipleRecordView, a member of class TXMLDisplayGroup.
V	GetLayoutAttributes (see page 16526)	Returns an object that specifies how the generated group of controls should look.
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVisibleFields (see page 17027)	This is GetVisibleFields, a member of class TXMLDisplayGroup.
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	GetXMLBroker (see page 17027)	Returns the value of the XMLBroker property.
	GetXMLDataSet (see page 17028)	Returns the value of the XMLDataSet property.
	GetXMLDisplayName (see page 17028)	This is GetXMLDisplayName, a member of class TXMLDisplayGroup.
	GetXMLRowSetName (see page 17028)	This is GetXMLRowSetName, a member of class TXMLDisplayGroup.
	HasStatusField (see page 17028)	This is HasStatusField, a member of class TXMLDisplayGroup.
V	ImplAddElements (see page 16597)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 16597)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 16526)	Provides the underlying implementation for the Content method.
V	ImplContent (see page 16882)	Provides the underlying implementation for the Content (see page 16879) method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
V	ImplGetSubComponents (see page 17029)	Provides the underlying implementation for the GetSubComponents method.
V	ImplIsMultipleRecordView (see page 17029)	This is ImplIsMultipleRecordView, a member of class TXMLDisplayGroup.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsFieldInUse (see page 17030)	This is IsFieldInUse, a member of class TXMLDisplayGroup.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 17030)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
	SetDataSetField (see page 17030)	This is SetDataSetField, a member of class TXMLDisplayGroup.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEnableValidateFields (see page 17031)	This is SetEnableValidateFields, a member of class TXMLDisplayGroup.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	SetXMLBroker (see page 17031)	This is SetXMLBroker, a member of class TXMLDisplayGroup.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateDataSet (see page 17031)	This is ValidateDataSet, a member of class TXMLDisplayGroup.
	ValidateFields (see page 17031)	This is ValidateFields, a member of class TXMLDisplayGroup.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.
V	XMLDataChange (see page 17031)	This is XMLDataChange, a member of class TXMLDisplayGroup.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
	EnableValidateFields (see page 17032)	This is EnableValidateFields, a member of class TXMLDisplayGroup.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.
R	XMLDataSet (see page 17033)	Indicates the TXMLDataSet object that represents the dataset to which the Web item is bound.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.

	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 16525)	Creates and initializes an instance of TCustomFieldGroup (see page 16519).
	Create (see page 16877)	Creates and initializes an instance of TWebControlGroup (see page 16872).
	Create (see page 17024)	Creates and initializes an instance of TXMLDisplayGroup (see page 17019).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 24742)	Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 16525)	Frees an instance of TCustomFieldGroup (see page 16519).
	Destroy (see page 16877)	Frees an instance of TWebControlGroup (see page 16872).
	Destroy (see page 17024)	Frees an instance of TXMLDisplayGroup (see page 17019).
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDataSet (see page 17054)	Returns the value of the DataSet (see page 17055) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnableValidateFields (see page 17078)	This is GetEnableValidateFields, a member of class IValidateFields.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 17060)	This is GetFieldsList, a member of class IDataSetFields.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsMultipleRecordView (see page 17079)	This is GetIsMultipleRecordView, a member of class IWebDisplay.
	GetIsMultipleRecordView (see page 17082)	This is GetIsMultipleRecordView, a member of class IXMLDisplay.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	GetVisibleFields (see page 17060)	This is GetVisibleFields, a member of class IDataSetFields.
	GetXMLBroker (see page 17054)	Returns the value of the XMLBroker (see page 17056) property.
	GetXMLDataSet (see page 17055)	Returns the value of the XMLDataSet (see page 17056) property.
	GetXMLDisplayName (see page 17083)	This is GetXMLDisplayName, a member of class IXMLDisplay.
	GetXMLRowSetName (see page 17083)	This is GetXMLRowSetName, a member of class IXMLDisplay.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.

	HasStatusField (see page 17060)	This is HasStatusField, a member of class IDataSetFields.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsFieldInUse (see page 17060)	This is IsFieldInUse, a member of class IDataSetFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEnableValidateFields (see page 17078)	This is SetEnableValidateFields, a member of class IValidateFields.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateDataSet (see page 17060)	This is ValidateDataSet, a member of class IDataSetFields.
	ValidateFields (see page 17078)	This is ValidateFields, a member of class IValidateFields.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 17055)	Provides access to an interface for working with the associated dataset.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EnableValidateFields (see page 17078)	This is EnableValidateFields, a member of class IValidateFields.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsMultipleRecordView (see page 17080)	
R	IsMultipleRecordView (see page 17083)	
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleFields (see page 17061)	This is VisibleFields, a member of class IDataSetFields.
R	VisibleFields (see page 17032)	Indicates the controls displayed by the HTML that the XML display group generates.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.
R	XMLBroker (see page 17056)	Indicates the XML broker that fetches the data packets to which the Web item is bound.

R	XMLDataSet (see page 17056)	Indicates the TXMLDataSet (see page 17006) object that represents the dataset to which the Web item is bound.
R	XMLDisplayName (see page 17084)	This is XMLDisplayName, a member of class IXMLDisplay.
R	XMLRowSetName (see page 17084)	This is XMLRowSetName, a member of class IXMLDisplay.

Published Properties

	Name	Description
	Custom (see page 16598)	Represents a tag option and its value in the generated TABLE tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16598)	Adds style attributes to the generated set of fields.
	StyleRule (see page 16599)	Applies a style in the page producer's style sheet to the table.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLBroker (see page 17033)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
	XMLDataSetField (see page 17033)	Binds the XML display group to a nested dataset from the XML data packet.

99.1.15.2 TFieldGroup Methods

The methods of the TFieldGroup class are listed here.

Protected Methods

	Name	Description
V	ImplAddElements (see page 16597)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 16597)	Provides the underlying implementation for the CanAddClass method.

99.1.15.2.1 TFieldGroup.ImplAddElements Method

Provides the underlying implementation for the AddElements method.

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); override;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

The AddElements method calls ImplAddElements to add any script declarations by calling the AddIntf interface. These declarations allow the generated display control to call script methods in response to user events by ensuring that the required methods are declared.

As implemented in TWebControlGroup, ImplAddElements does nothing. TWebControlGroup descendants override ImplAddElements to add any required declarations.

See Also

AddElements (see page 16878)

99.1.15.2.2 TFieldGroup.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass (see page 16878)

WebFieldControls (see page 16884)

99.1.15.3 TFieldGroup Properties

The properties of the TFieldGroup class are listed here.

Published Properties

	Name	Description
	Custom (see page 16598)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 16598)	Adds style attributes to the generated set of fields.
	StyleRule (see page 16599)	Applies a style in the page producer's style sheet to the table.

99.1.15.3.1 TFieldGroup.Custom Property

Represents a tag option and its value in the generated TABLE tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TCustomFieldGroup generates an HTML table to manage the layout of the HTML controls that its Web items generate. Use Custom to add options that qualify the TABLE tag. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to surround the set of fields with a border and give them a blue background, set Custom to

```
BORDER=2 BGCOLOR=BLUE
```

See Also

Style (see page 16527)

StyleRule (see page 16528)

99.1.15.3.2 TFieldGroup.Style Property

Adds style attributes to the generated set of fields.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

TCustomFieldGroup generates an HTML table to lay out the fields that its Web items generate. Use Style to add a STYLE option that qualifies the TABLE tag. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated TABLE tag as the value of a STYLE option.

See Also

Custom (see page 16527)

StyleRule (see page 16528)

99.1.15.3.3 TFieldGroup.StyleRule Property

Applies a style in the page producer's style sheet to the table.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated set of fields. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style property.

See Also

Style (see page 16527)

Styles (see page 17127)

StylesFile (see page 17128)

99.1.16 TFieldRadioGroup Class

TFieldRadioGroup creates an HTML radio group control to represent a field in an XML data packet.

Class Hierarchy

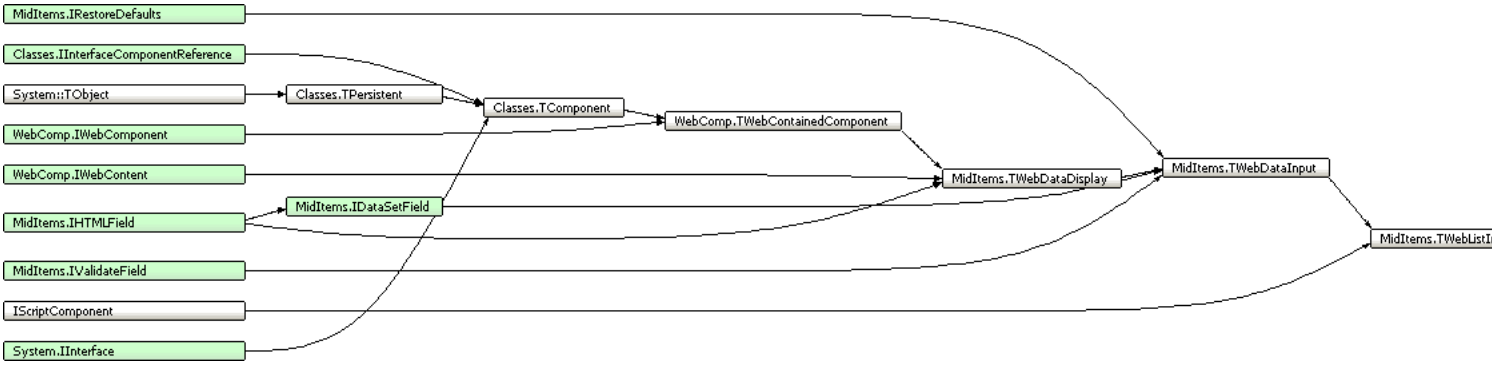

Pascal

```
TFieldRadioGroup = class(TWebRadioGroupInput);
```

C++

```
class TFieldRadioGroup : public TWebRadioGroupInput;
```

File

MidItems

Description

Add TFieldRadioGroup as the child of a TFieldGroup (see page 16591) component to add a radio group control that represents a field from the field group's associated dataset.

Use the properties and methods of TFieldRadioGroup to

Specify the field whose value the radio group displays.

Specify the possible values for the field from which the user can choose.

Specify a caption that describes the field.

Customize the appearance of the radio group.

Fit the radio group into the tab order of the generated HTML form.

See Also

TFieldGroup (see page 16591)

TXMLDisplayGroup (see page 17019)

TXMLDisplayGroup (see page 17019)

99.1.16.1 TFieldRadioGroup Members

The following tables list the members exposed by TFieldRadioGroup.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddAttributes (see page 16953)	This is AddAttributes, a member of class TWebRadioGroupInput.
	AddElements (see page 16941)	This is AddElements, a member of class TWebListInput.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 16942)	This is ControlContent, a member of class TWebListInput.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	EventContent (see page 16905)	This is EventContent, a member of class TWebDataInput.
V	EventContent (see page 16953)	This is EventContent, a member of class TWebRadioGroupInput.
V	FindAssociatedField (see page 16906)	This is FindAssociatedField, a member of class TWebDataInput.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
A	FormatInputs (see page 16942)	This is FormatInputs, a member of class TWebListInput.
V	FormatInputs (see page 16953)	This is FormatInputs, a member of class TWebRadioGroupInput.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 16906)	Returns the value of the Caption property.
V	GetCheckIndex (see page 16954)	This is GetCheckIndex, a member of class TWebRadioGroupInput.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetFieldName (see page 16906)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetParamName (see page 16906)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 16907)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetSubComponents (see page 16942)	This is GetSubComponents, a member of class TWebListInput.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplAddElements (see page 16942)	This is ImplAddElements, a member of class TWebListInput.
V	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
V	ImplGetFieldName (see page 16907)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.

V	ImplGetHTMLControlName (see page 16907)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 16907)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplGetRowSetFieldAttributes (see page 16954)	This is ImplGetRowSetFieldAttributes, a member of class TWebRadioGroupInput.
V	ImplRestoreDefaults (see page 16908)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16943)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 16908)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16908)	This is RestoreFieldDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16954)	This is RestoreFieldDefaults, a member of class TWebRadioGroupInput.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
V	SetCaption (see page 16908)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 16943)	This is SetDataSet, a member of class TWebListInput.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 16909)	This is SetFieldName, a member of class TWebDataInput.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItems (see page 16943)	This is SetItems, a member of class TWebListInput.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParamName (see page 16909)	This is SetParamName, a member of class TWebDataInput.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetValues (see page 16944)	This is SetValues, a member of class TWebListInput.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateField (see page 16909)	This is ValidateField, a member of class TWebDataInput.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.

	ParamName (see page 16910)	Specifies the name of the parameter whose value the generated control represents.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16889)	Creates and initializes an instance of TWebDataDisplay (see page 16884).
V	Create (see page 16940)	Creates and initializes an instance of TWebListInput (see page 16935).
V	Create (see page 16952)	Creates and initializes an instance of TWebRadioGroupInput (see page 16947).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
V	Destroy (see page 16941)	Frees an instance of TWebListInput (see page 16935).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 17057)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.

	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRowSetFieldAttributes (see page 17058)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 17058)	This is SetFieldName, a member of class IDataSetField.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateField (see page 17076)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.

R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Caption (see page 16606)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16606)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16606)	Specifies where the label appears relative to the generated display control.
	Custom (see page 16607)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 16607)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayColumns (see page 16608)	Specifies the number of radio buttons that appear in each row of the control.
	DisplayWidth (see page 16608)	Specifies the width of the radio group, in characters.
	FieldName (see page 16608)	Specifies the name of the field whose value the generated control represents.
	Items (see page 16609)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 16609)	Indicates which field in DataSet supplies the possible selections for the generated control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ReadOnly (see page 16610)	Indicates whether end users can use the generated control to set values.
	Style (see page 16610)	Adds style attributes to the generated display control.
	StyleRule (see page 16611)	This is StyleRule, a member of class TFieldRadioGroup.
	TabIndex (see page 16611)	Determines the position of the generated control in the HTML form's tab order.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Values (see page 16611)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 16612)	Indicates which field in DataSet supplies the possible values for the generated control.

99.1.16.2 TFieldRadioGroup Properties

The properties of the TFieldRadioGroup class are listed here.

Published Properties

	Name	Description
	Caption (see page 16606)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16606)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16606)	Specifies where the label appears relative to the generated display control.
	Custom (see page 16607)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 16607)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayColumns (see page 16608)	Specifies the number of radio buttons that appear in each row of the control.
	DisplayWidth (see page 16608)	Specifies the width of the radio group, in characters.
	FieldName (see page 16608)	Specifies the name of the field whose value the generated control represents.
	Items (see page 16609)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 16609)	Indicates which field in DataSet supplies the possible selections for the generated control.
	ReadOnly (see page 16610)	Indicates whether end users can use the generated control to set values.
	Style (see page 16610)	Adds style attributes to the generated display control.
	StyleRule (see page 16611)	This is StyleRule, a member of class TFieldRadioGroup.
	TabIndex (see page 16611)	Determines the position of the generated control in the HTML form's tab order.
	Values (see page 16611)	Lists the possible values that can be assigned to the control.

	ValuesField (see page 16612)	Indicates which field in DataSet supplies the possible values for the generated control.

99.1.16.2.1 TFieldRadioGroup.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 16897)

CaptionPosition (see page 16897)

99.1.16.2.2 TFieldRadioGroup.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 16897)

CaptionPosition (see page 16897)

HeadingAttributes (see page 16509)

99.1.16.2.3 TFieldRadioGroup.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 16897)

Caption (see page 16897)

99.1.16.2.4 TFieldRadioGroup.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 16899)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

99.1.16.2.5 TFieldRadioGroup.DataSet Property

Specifies a dataset that supplies the possible values listed in the generated control.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Use DataSet to specify a dataset from which the component can extract the selections that appear to the user and the underlying values they represent. The ItemsField property specifies the field in this dataset that lists the selections that appear in the generated control. The ValuesField property specifies the field in this dataset that lists the corresponding values that are assigned to the control.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

ItemsField (see page 16945)

ValuesField (see page 16946)

Items (see page 16945)

Values (see page 16946)

99.1.16.2.6 TFieldRadioGroup.DisplayColumns Property

Specifies the number of radio buttons that appear in each row of the control.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Use DisplayColumns to specify how many radio buttons can appear in a single line of the radio group. If DisplayColumns is less than the number of items, the radio group displays its buttons in multiple rows.

See Also

ItemsField (see page 16945)

Items (see page 16945)

DisplayWidth (see page 16955)

99.1.16.2.7 TFieldRadioGroup.DisplayWidth Property

Specifies the width of the radio group, in characters.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to limit the width of the generated control. TWebRadioGroup generates an INPUT tag of type RADIO to implement the radio group. DisplayWidth is added as a WIDTH option.

See Also

DisplayColumns (see page 16954)

Custom (see page 16898)

99.1.16.2.8 TFieldRadioGroup.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 17033)

XMLDataSetField (see page 17033)

ParamName (see page 16910)

Setting Web Item Properties

99.1.16.2.9 TFieldRadioGroup.Items Property

Lists the possible selections that appear in the generated control.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to supply a list of possible selections to the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the corresponding entry in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

ItemsField (see page 16945)

Values (see page 16946)

DataSet (see page 16944)

ValuesField (see page 16946)

99.1.16.2.10 TFieldRadioGroup.ItemsField Property

Indicates which field in DataSet supplies the possible selections for the generated control.

Pascal

```
property ItemsField: string;
```

C++

```
__property AnsiString ItemsField;
```

Description

Use ItemsField to specify which field in the dataset specified by the DataSet property contains selections that appear in the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is

assigned comes from the field specified by the ValuesField property.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 16945)

Values (see page 16946)

DataSet (see page 16944)

ValuesField (see page 16946)

99.1.16.2.11 TFieldRadioGroup.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the radio group. TWebRadioGroup generates an INPUT tag of type RADIO to implement the radio group. ReadOnly is added as a READONLY option.

See Also

DisplayWidth (see page 16955)

Custom (see page 16898)

99.1.16.2.12 TFieldRadioGroup.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 16898)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

Setting Web Item Properties

99.1.16.2.13 TFieldRadioGroup.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TFieldRadioGroup.

99.1.16.2.14 TFieldRadioGroup.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

99.1.16.2.15 TFieldRadioGroup.Values Property

Lists the possible values that can be assigned to the control.

Pascal

```
property Values: TStrings;
```

C++

```
__property TStrings Values;
```

Description

Use Values to supply a list of possible values for the generated control. When the end-user chooses one of the selections listed by the Items property, the control is assigned the corresponding value in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

Items (see page 16945)

ItemsField (see page 16945)

DataSet (see page 16944)

ValuesField (see page 16946)

99.1.16.2.16 TFieldRadioGroup.ValuesField Property

Indicates which field in DataSet supplies the possible values for the generated control.

Pascal

```
property ValuesField: string;
```

C++

```
__property AnsiString ValuesField;
```

Description

Use ValuesField to specify which field in the dataset specified by the DataSet property contains values that can be assigned to the generated control. When the end-user selects the corresponding value form the ItemsField field, the control gets the value in the ValuesField field.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 16945)

Values (see page 16946)

DataSet (see page 16944)

ItemsField (see page 16945)

99.1.17 TFieldSelectOptions Class

TFieldSelectOptions creates an HTML list control to represent a field in an XML data packet.

Class Hierarchy

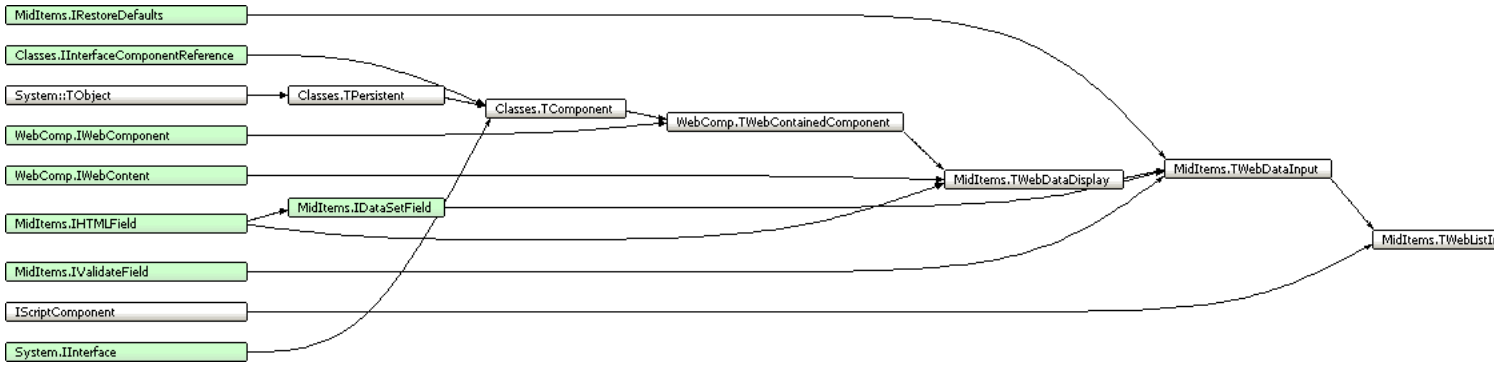

Pascal

```
TFieldSelectOptions = class(TWebSelectOptionsInput);
```

C++

```
class TFieldSelectOptions : public TWebSelectOptionsInput;
```

File

MidItems

Description

Add TFieldSelectOptions as the child of a TFieldGroup (see page 16591) component to add a list control that represents a field from the field group's associated dataset.

Use the properties and methods of TFieldSelectOptions to

Specify the field whose value the list box displays.

Specify the possible values for the field from which the user can choose.

Specify a caption that describes the field.

Customize the appearance of the list box.

Fit the list box into the tab order of the generated HTML form.

See Also

TFieldGroup (see page 16591)

TXMLDisplayGroup (see page 17019)

TXMLDisplayGroup (see page 17019)

99.1.17.1 TFieldSelectOptions Members

The following tables list the members exposed by TFieldSelectOptions.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddAttributes (see page 16962)	This is AddAttributes, a member of class TWebSelectOptionsInput.
	AddElements (see page 16941)	This is AddElements, a member of class TWebListInput.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 16942)	This is ControlContent, a member of class TWebListInput.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	EventContent (see page 16905)	This is EventContent, a member of class TWebDataInput.
V	EventContent (see page 16962)	This is EventContent, a member of class TWebSelectOptionsInput.
V	FindAssociatedField (see page 16906)	This is FindAssociatedField, a member of class TWebDataInput.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
A	FormatInputs (see page 16942)	This is FormatInputs, a member of class TWebListInput.
V	FormatInputs (see page 16962)	This is FormatInputs, a member of class TWebSelectOptionsInput.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 16906)	Returns the value of the Caption property.

	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetFieldName (see page 16906)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetParamName (see page 16906)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 16907)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
V	GetSelectIndex (see page 16962)	This is GetSelectIndex, a member of class TWebSelectOptionsInput.
	GetSubComponents (see page 16942)	This is GetSubComponents, a member of class TWebListInput.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplAddElements (see page 16942)	This is ImplAddElements, a member of class TWebListInput.
V	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
V	ImplGetFieldName (see page 16907)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 16907)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 16907)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplRestoreDefaults (see page 16908)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16943)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 16908)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16908)	This is RestoreFieldDefaults, a member of class TWebDataInput.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
V	SetCaption (see page 16908)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 16943)	This is SetDataSet, a member of class TWebListInput.

	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 16909)	This is SetFieldName, a member of class TWebDataInput.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItems (see page 16943)	This is SetItems, a member of class TWebListInput.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParamName (see page 16909)	This is SetParamName, a member of class TWebDataInput.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetValues (see page 16944)	This is SetValues, a member of class TWebListInput.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateField (see page 16909)	This is ValidateField, a member of class TWebDataInput.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 16910)	Specifies the name of the parameter whose value the generated control represents.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16889)	Creates and initializes an instance of TWebDataDisplay (see page 16884).
V	Create (see page 16940)	Creates and initializes an instance of TWebListInput (see page 16935).

	Create (see page 16961)	Creates and initializes an instance of TWebSelectOptionsInput (see page 16956).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
	Destroy (see page 16941)	Frees an instance of TWebListInput (see page 16935).
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 17057)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRowSetFieldAttributes (see page 17058)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 17058)	This is SetFieldName, a member of class IDataSetField.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateField (see page 17076)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Caption (see page 16618)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16618)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16619)	Specifies where the label appears relative to the generated display control.
	Custom (see page 16619)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 16620)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayRows (see page 16620)	Specifies the number of rows that appear in the generated list box.
	FieldName (see page 16621)	Specifies the name of the field whose value the generated control represents.
	Items (see page 16621)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 16621)	Indicates which field in DataSet supplies the possible selections for the generated control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16622)	Adds style attributes to the generated display control.
	StyleRule (see page 16622)	This is StyleRule, a member of class TFieldSelectOptions.
	TabIndex (see page 16623)	Determines the position of the generated control in the HTML form's tab order.
	Tag (see page 2808)	Stores an integer value as part of a component.

	Values (see page 16623)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 16623)	Indicates which field in DataSet supplies the possible values for the generated control.

99.1.17.2 TFieldSelectOptions Properties

The properties of the TFieldSelectOptions class are listed here.

Published Properties

	Name	Description
	Caption (see page 16618)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16618)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16619)	Specifies where the label appears relative to the generated display control.
	Custom (see page 16619)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 16620)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayRows (see page 16620)	Specifies the number of rows that appear in the generated list box.
	FieldName (see page 16621)	Specifies the name of the field whose value the generated control represents.
	Items (see page 16621)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 16621)	Indicates which field in DataSet supplies the possible selections for the generated control.
	Style (see page 16622)	Adds style attributes to the generated display control.
	StyleRule (see page 16622)	This is StyleRule, a member of class TFieldSelectOptions.
	TabIndex (see page 16623)	Determines the position of the generated control in the HTML form's tab order.
	Values (see page 16623)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 16623)	Indicates which field in DataSet supplies the possible values for the generated control.

99.1.17.2.1 TFieldSelectOptions.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 16897)

CaptionPosition (see page 16897)

99.1.17.2.2 TFieldSelectOptions.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 16897)

CaptionPosition (see page 16897)

HeadingAttributes (see page 16509)

99.1.17.2.3 TFieldSelectOptions.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 16897)

Caption (see page 16897)

99.1.17.2.4 TFieldSelectOptions.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 16899)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

99.1.17.2.5 TFieldSelectOptions.DataSet Property

Specifies a dataset that supplies the possible values listed in the generated control.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Use DataSet to specify a dataset from which the component can extract the selections that appear to the user and the underlying values they represent. The ItemsField property specifies the field in this dataset that lists the selections that appear in the generated control. The ValuesField property specifies the field in this dataset that lists the corresponding values that are assigned to the control.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

ItemsField (see page 16945)

ValuesField (see page 16946)

Items (see page 16945)

Values (see page 16946)

99.1.17.2.6 TFieldSelectOptions.DisplayRows Property

Specifies the number of rows that appear in the generated list box.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to specify how many rows appear in the generated list control. If DisplayRows is less than the number of items, the generated list displays only DisplayRows options at one time, but can scroll to display the hidden items. If DisplayRows is 1, the generated control becomes a drop-down list box.

See Also

ItemsField (see page 16945)

Items (see page 16945)

99.1.17.2.7 TFieldSelectOptions.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 17033)

XMLDataSetField (see page 17033)

ParamName (see page 16910)

Setting Web Item Properties

99.1.17.2.8 TFieldSelectOptions.Items Property

Lists the possible selections that appear in the generated control.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to supply a list of possible selections to the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the corresponding entry in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

ItemsField (see page 16945)

Values (see page 16946)

DataSet (see page 16944)

ValuesField (see page 16946)

99.1.17.2.9 TFieldSelectOptions.ItemsField Property

Indicates which field in DataSet supplies the possible selections for the generated control.

Pascal

```
property ItemsField: string;
```

C++

```
__property AnsiString ItemsField;
```

Description

Use ItemsField to specify which field in the dataset specified by the DataSet property contains selections that appear in the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the field specified by the ValuesField property.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 16945)

Values (see page 16946)

DataSet (see page 16944)

ValuesField (see page 16946)

99.1.17.2.10 TFieldSelectOptions.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 16898)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

Setting Web Item Properties

99.1.17.2.11 TFieldSelectOptions.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TFieldSelectOptions.

99.1.17.2.12 TFieldSelectOptions.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

99.1.17.2.13 TFieldSelectOptions.Values Property

Lists the possible values that can be assigned to the control.

Pascal

```
property Values: TStrings;
```

C++

```
__property TStrings Values;
```

Description

Use Values to supply a list of possible values for the generated control. When the end-user chooses one of the selections listed by the Items property, the control is assigned the corresponding value in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

Items (see page 16945)

ItemsField (see page 16945)

DataSet (see page 16944)

ValuesField (see page 16946)

99.1.17.2.14 TFieldSelectOptions.ValuesField Property

Indicates which field in DataSet supplies the possible values for the generated control.

Pascal

```
property ValuesField: string;
```

C++

```
__property AnsiString ValuesField;
```

Description

Use ValuesField to specify which field in the dataset specified by the DataSet property contains values that can be assigned to the generated control. When the end-user selects the corresponding value form the ItemsField field, the control gets the value in the ValuesField field.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 16945)

Values (see page 16946)

DataSet (see page 16944)

ItemsField (see page 16945)

99.1.18 TFieldStatus Class

TFieldStatus generates an HTML control that indicates whether a record in an XML data packet has been modified by the end user.

Class Hierarchy

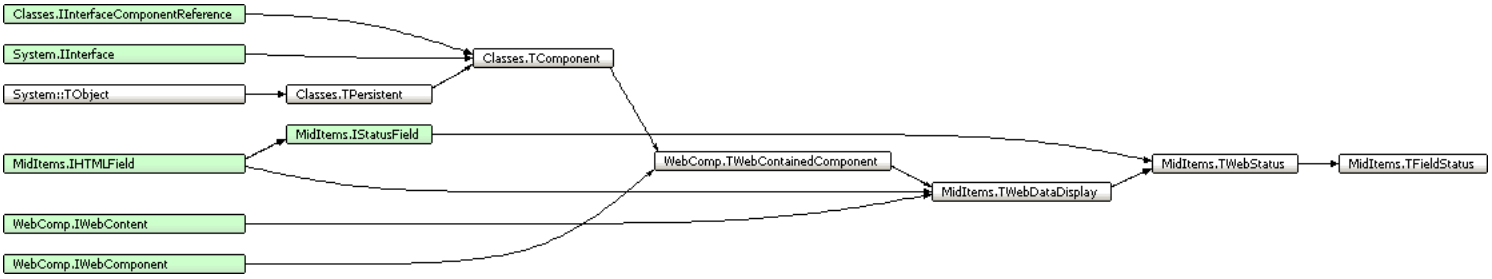

Pascal

```
TFieldStatus = class(TWebStatus);
```

C++

```
class TFieldStatus : public TWebStatus;
```

File

MidItems

Description

Add TFieldStatus as the child of a TFieldGroup (see page 16591) component to provide feedback about whether the current record has been modified. The generated control displays an M if the user has edited the record, an I if the user has inserted the record, and blank if the user has made no changes to the record.

Use the properties and methods of TFieldStatus to

Specify a caption that labels the status control.

Customize the appearance of the control.

Fit the control into the tab order of the generated HTML form.

See Also

TFieldGroup (see page 16591)

TXMLDisplayGroup (see page 17019)

TXMLDisplayGroup (see page 17019)

99.1.18.1 TFieldStatus Members

The following tables list the members exposed by TFieldStatus.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 16969)	This is ControlContent, a member of class TWebStatus.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 16969)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.

V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 16969)	Returns the name that other components can use to refer to the generated display control.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16889)	Creates and initializes an instance of TWebDataDisplay (see page 16884).
V	Create (see page 16968)	Creates and initializes an instance of TWebStatus (see page 16963).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	Identifier (see page 16969)	Returns the name under which record status information is stored in the XML data packet.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

V	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayWidth (see page 16971)	Specifies the width, in characters, of the generated HTML control.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	TabIndex (see page 16899)	Determines the position of the generated control in the HTML form's tab order.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Caption (see page 16629)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16629)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16630)	Specifies where the label appears relative to the generated display control.
	Custom (see page 16630)	Represents a tag option and its value in the tag for the display control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16630)	Adds style attributes to the generated display control.
	StyleRule (see page 16631)	This is StyleRule, a member of class TFieldStatus.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.18.2 TFieldStatus Properties

The properties of the TFieldStatus class are listed here.

Published Properties

	Name	Description
	Caption (see page 16629)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16629)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16630)	Specifies where the label appears relative to the generated display control.
	Custom (see page 16630)	Represents a tag option and its value in the tag for the display control.
	Style (see page 16630)	Adds style attributes to the generated display control.
	StyleRule (see page 16631)	This is StyleRule, a member of class TFieldStatus.

99.1.18.2.1 TFieldStatus.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 16897)

CaptionPosition (see page 16897)

99.1.18.2.2 TFieldStatus.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 16897)

CaptionPosition (see page 16897)

HeadingAttributes (see page 16509)

99.1.18.2.3 TFieldStatus.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 16897)

Caption (see page 16897)

99.1.18.2.4 TFieldStatus.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 16899)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

99.1.18.2.5 TFieldStatus.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 16898)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

Setting Web Item Properties

99.1.18.2.6 TFieldStatus.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TFieldStatus.

99.1.19 TFieldText Class

TFieldText creates a single-line HTML input control to represent a field in an XML data packet.

Class Hierarchy

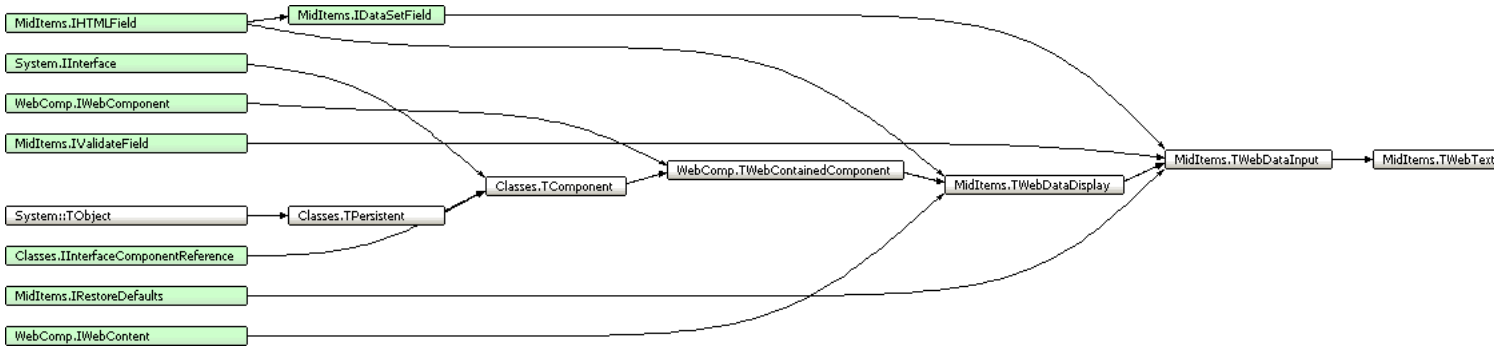

Pascal

```
TFieldText = class(TWebTextInput);
```

C++

```
class TFieldText : public TWebTextInput;
```

File

MidItems

Description

Add TFieldText as the child of a TFieldGroup (see page 16591) component to add a single-line edit control that represents a field from the field group's associated dataset.

Use the properties and methods of TFieldText to

Specify the field whose value the control displays.

Specify a caption that describes the field.

Customize the appearance of the edit control.

Fit the control into the tab order of the generated HTML form.

See Also

TFieldGroup (see page 16591)

TXMLDisplayGroup (see page 17019)

TXMLDisplayGroup (see page 17019)

99.1.19.1 TFieldText Members

The following tables list the members exposed by TFieldText.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddAttributes (see page 16986)	This is AddAttributes, a member of class TWebTextInput.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 16987)	This is ControlContent, a member of class TWebTextInput.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	EventContent (see page 16905)	This is EventContent, a member of class TWebDataInput.
V	FindAssociatedField (see page 16906)	This is FindAssociatedField, a member of class TWebDataInput.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 16906)	Returns the value of the Caption property.

	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetFieldName (see page 16906)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetParamName (see page 16906)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 16907)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
V	ImplGetFieldName (see page 16907)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 16907)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 16907)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplGetRowSetFieldAttributes (see page 16987)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextInput.
V	ImplRestoreDefaults (see page 16908)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 16908)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16908)	This is RestoreFieldDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16987)	This is RestoreFieldDefaults, a member of class TWebTextInput.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
V	SetCaption (see page 16908)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 16909)	This is SetFieldName, a member of class TWebDataInput.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParamName (see page 16909)	This is SetParamName, a member of class TWebDataInput.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateField (see page 16909)	This is ValidateField, a member of class TWebDataInput.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 16910)	Specifies the name of the parameter whose value the generated control represents.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16889)	Creates and initializes an instance of TWebDataDisplay (see page 16884).
V	Create (see page 16986)	Creates and initializes an instance of TWebTextInput (see page 16980).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.

	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 17057)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRowSetFieldAttributes (see page 17058)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.

V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 17058)	This is SetFieldName, a member of class IDataSetField.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateField (see page 17076)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Caption (see page 16637)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16637)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16637)	Specifies where the label appears relative to the generated display control.
	Custom (see page 16638)	Represents a tag option and its value in the tag for the display control.
	DisplayWidth (see page 16638)	Specifies the horizontal size of the generated control.
	FieldName (see page 16639)	Specifies the name of the field whose value the generated control represents.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ReadOnly (see page 16639)	Indicates whether end users can use the generated control to set values.
	Style (see page 16639)	Adds style attributes to the generated display control.
	StyleRule (see page 16640)	This is StyleRule, a member of class TFieldText.
	TabIndex (see page 16640)	Determines the position of the generated control in the HTML form's tab order.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.19.2 TFieldText Properties

The properties of the TFieldText class are listed here.

Published Properties

	Name	Description
	Caption (see page 16637)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16637)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16637)	Specifies where the label appears relative to the generated display control.

	Custom (see page 16638)	Represents a tag option and its value in the tag for the display control.
	DisplayWidth (see page 16638)	Specifies the horizontal size of the generated control.
	FieldName (see page 16639)	Specifies the name of the field whose value the generated control represents.
	ReadOnly (see page 16639)	Indicates whether end users can use the generated control to set values.
	Style (see page 16639)	Adds style attributes to the generated display control.
	StyleRule (see page 16640)	This is StyleRule, a member of class TFieldText.
	TabIndex (see page 16640)	Determines the position of the generated control in the HTML form's tab order.

99.1.19.2.1 TFieldText.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 16897)

CaptionPosition (see page 16897)

99.1.19.2.2 TFieldText.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 16897)

CaptionPosition (see page 16897)

HeadingAttributes (see page 16509)

99.1.19.2.3 TFieldText.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 16897)

Caption (see page 16897)

99.1.19.2.4 TFieldText.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 16899)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

99.1.19.2.5 TFieldText.DisplayWidth Property

Specifies the horizontal size of the generated control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to indicate the desired width of the generated HTML control. DisplayWidth specifies the number of characters that can be displayed in the control.

Note: DisplayWidth controls the size of the generated control, not the size of the text that users can enter. If the user enters a string longer than DisplayWidth characters, the text scrolls in the control.

99.1.19.2.6 TFieldText.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 17033)

XMLDataSetField (see page 17033)

ParamName (see page 16910)

Setting Web Item Properties

99.1.19.2.7 TFieldText.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the generated control.

99.1.19.2.8 TFieldText.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute

descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 16898)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

Setting Web Item Properties

99.1.19.2.9 TFieldText.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TFieldText.

99.1.19.2.10 TFieldText.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

99.1.20 TFieldTextArea Class

TFieldTextArea creates a multi-line HTML input control to represent a field in an XML data packet.

Class Hierarchy

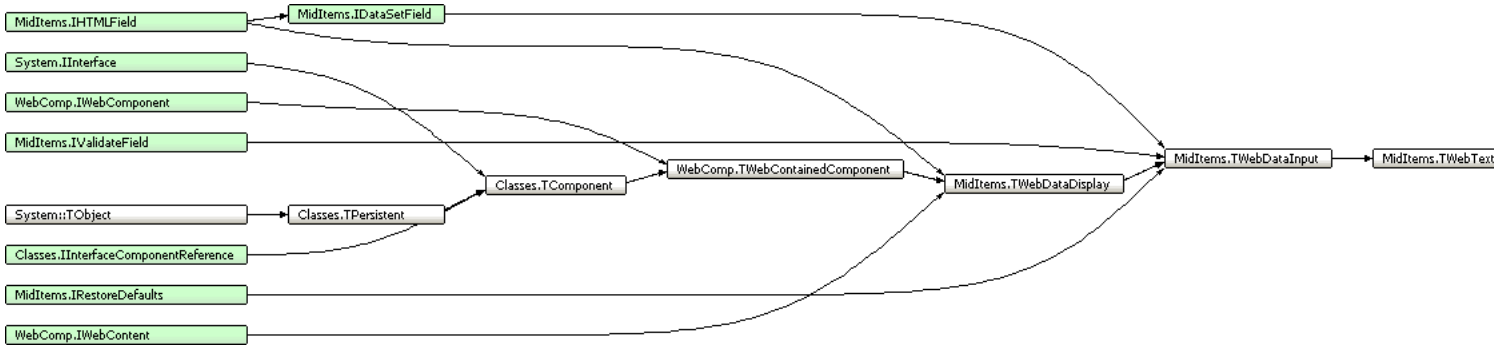

Pascal

```
TFieldTextArea = class(TWebTextAreaInput);
```

C++

```
class TFieldTextArea : public TWebTextAreaInput;
```

File

MidItems

Description

Add TFieldTextArea as the child of a TFieldGroup (see page 16591) component to add a multi-line edit control that represents a field from the field group's associated dataset.

Use the properties and methods of TFieldTextArea to

Specify the field whose value the control displays.

Specify a caption that describes the field.

Customize the appearance of the control.

Fit the control into the tab order of the generated HTML form.

See Also

IWebContent (see page 24792)

TDataGrid (see page 16560)

TFieldGroup (see page 16591)

TQueryFieldGroup (see page 16736)

TQueryTextArea (see page 16795)

TTextAreaColumn (see page 16838)

99.1.20.1 TFieldTextArea Members

The following tables list the members exposed by TFieldTextArea.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddAttributes (see page 16977)	This is AddAttributes, a member of class TWebTextAreaInput.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 16978)	This is ControlContent, a member of class TWebTextAreaInput.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	EventContent (see page 16905)	This is EventContent, a member of class TWebDataInput.
V	EventContent (see page 16978)	This is EventContent, a member of class TWebTextAreaInput.
V	FindAssociatedField (see page 16906)	This is FindAssociatedField, a member of class TWebDataInput.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 16906)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetFieldName (see page 16906)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetParamName (see page 16906)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 16907)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
V	ImplGetFieldName (see page 16907)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 16907)	Returns the name that other components can use to refer to the generated display control.

V	ImplGetRowSetFieldAttributes (see page 16907)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplGetRowSetFieldAttributes (see page 16978)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextAreaInput.
V	ImplRestoreDefaults (see page 16908)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 16908)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16908)	This is RestoreFieldDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16978)	This is RestoreFieldDefaults, a member of class TWebTextAreaInput.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
V	SetCaption (see page 16908)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 16909)	This is SetFieldName, a member of class TWebDataInput.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParamName (see page 16909)	This is SetParamName, a member of class TWebDataInput.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateField (see page 16909)	This is ValidateField, a member of class TWebDataInput.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 16910)	Specifies the name of the parameter whose value the generated control represents.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16889)	Creates and initializes an instance of TWebDataDisplay (see page 16884).
V	Create (see page 16977)	Creates and initializes an instance of TWebTextAreaInput (see page 16971).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 17057)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.

V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRowSetFieldAttributes (see page 17058)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 17058)	This is SetFieldName, a member of class IDataSetField.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateField (see page 17076)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Caption (see page 16646)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16647)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16647)	Specifies where the label appears relative to the generated display control.
	Custom (see page 16647)	Represents a tag option and its value in the tag for the display control.
	DisplayRows (see page 16648)	Specifies the height of the generated control.
	DisplayWidth (see page 16648)	Specifies the horizontal size of the generated control.
	FieldName (see page 16649)	Specifies the name of the field whose value the generated control represents.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ReadOnly (see page 16649)	Indicates whether end users can use the generated control to set values.
	Style (see page 16649)	Adds style attributes to the generated display control.
	StyleRule (see page 16650)	This is StyleRule, a member of class TFieldTextArea.
	TabIndex (see page 16650)	Determines the position of the generated control in the HTML form's tab order.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Wrap (see page 16650)	Specifies how the generated control treats lines that are longer than the display width.

99.1.20.2 TFieldTextArea Properties

The properties of the TFieldTextArea class are listed here.

Published Properties

	Name	Description
	Caption (see page 16646)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16647)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16647)	Specifies where the label appears relative to the generated display control.
	Custom (see page 16647)	Represents a tag option and its value in the tag for the display control.
	DisplayRows (see page 16648)	Specifies the height of the generated control.
	DisplayWidth (see page 16648)	Specifies the horizontal size of the generated control.
	FieldName (see page 16649)	Specifies the name of the field whose value the generated control represents.
	ReadOnly (see page 16649)	Indicates whether end users can use the generated control to set values.
	Style (see page 16649)	Adds style attributes to the generated display control.
	StyleRule (see page 16650)	This is StyleRule, a member of class TFieldTextArea.
	TabIndex (see page 16650)	Determines the position of the generated control in the HTML form's tab order.
	Wrap (see page 16650)	Specifies how the generated control treats lines that are longer than the display width.

99.1.20.2.1 TFieldTextArea.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 16897)

CaptionPosition (see page 16897)

99.1.20.2.2 TFieldTextArea.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 16897)

CaptionPosition (see page 16897)

HeadingAttributes (see page 16509)

99.1.20.2.3 TFieldTextArea.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 16897)

Caption (see page 16897)

99.1.20.2.4 TFieldTextArea.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 16899)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

99.1.20.2.5 TFieldTextArea.DisplayRows Property

Specifies the height of the generated control.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to indicate the desired height of the generated HTML control. DisplayRows specifies the number of lines of text that can be displayed in the control. This value is assigned to the ROWS option of the generated TEXTAREA tag.

See Also

DisplayWidth (see page 16979)

99.1.20.2.6 TFieldTextArea.DisplayWidth Property

Specifies the horizontal size of the generated control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to indicate the desired width of the generated HTML control. DisplayWidth specifies the number of characters that can be displayed in one line of the control. This value is assigned to the COLS option of the generated TEXTAREA tag.

See Also

DisplayRows (see page 16979)

99.1.20.2.7 TFieldTextArea.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 17033)

XMLDataSetField (see page 17033)

ParamName (see page 16910)

Setting Web Item Properties

99.1.20.2.8 TFieldTextArea.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the generated control. This value determines whether there is a READONLY option added to the generated TEXTAREA tag.

99.1.20.2.9 TFieldTextArea.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 16898)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

Setting Web Item Properties

99.1.20.2.10 TFieldTextArea.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TFieldTextArea.

99.1.20.2.11 TFieldTextArea.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

99.1.20.2.12 TFieldTextArea.Wrap Property

Specifies how the generated control treats lines that are longer than the display width.

Pascal

```
property Wrap: TTextAreaWrap;
```

C++

```
__property TTextAreaWrap Wrap;
```

Description

Use Wrap to specify how the generated control formats lines that are longer than the number of characters specified by DisplayWidth. TWebTextAreaInput uses Wrap to add a WRAP option to the generated TEXTAREA tag.

Wrap can have any of the following values:

Value	WRAP option value	Meaning
wrOff	OFF	The control does not wrap long lines. Only newline characters that are part of the text string cause the lines in the control to wrap.
wrPhysical	PHYSICAL	The control formats its text value by wrapping long lines so that no text is hidden. The control inserts carriage returns into the text string.
wrVirtual	VIRTUAL	The control formats its text value by wrapping long lines so that no text is hidden but does not change the actual value of the text string.

See Also

DisplayWidth (see page 16979)

99.1.21 TFirstButton Class

TFirstButton generates an HTML control that moves to the first record in an XML data packet.

Class Hierarchy

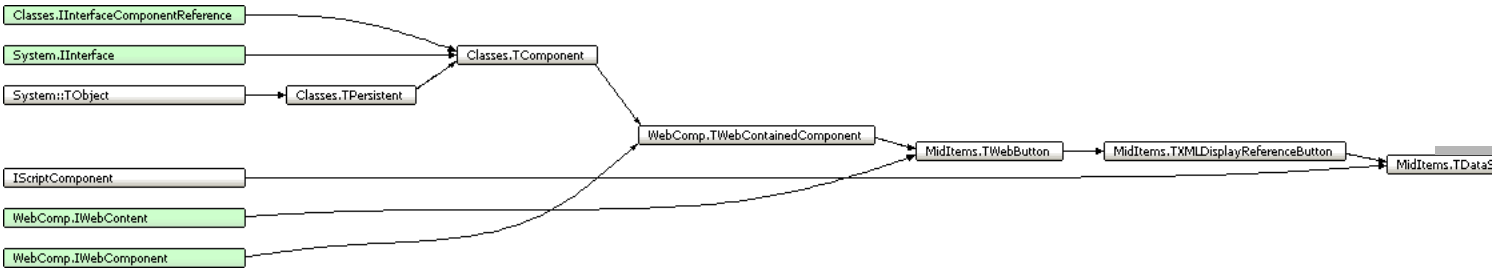

Pascal

```
TFirstButton = class(TDataSetButton);
```

C++

```
class TFirstButton : public TDataSetButton;
```

File

MidItems

Description

Add TFirstButton as the child of a TDataNavigator (see page 16569) component (or a layout group that is a child of a TDataNavigator (see page 16569) component) to enable the generated HTML form to move quickly to the first record in a dataset. The resulting HTML button's action changes the associated display control so that it shows the first record as current.

Use the properties of TFirstButton to customize the appearance of the button and associate it with a Web item. The associated Web item displays the records that the first button navigates.

See Also

TXMLBroker (see page 26049)

TDataNavigator (see page 16569)

TLayoutGroup (see page 16687)

99.1.21.1 TFirstButton Members

The following tables list the members exposed by TFirstButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 16581)	This is DefaultCaption, a member of class TDataSetButton.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	XMLMethodName (see page 16581)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16582)	This is AddElements, a member of class TDataSetButton.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16868)	Returns the HTML that defines the button control.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetCaption (see page 16582)	Returns the value of the Caption property.
V	GetCaption (see page 16869)	Returns the value of the Caption (see page 16870) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDisplayComponent (see page 17043)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 17043)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
V	GetMethodName (see page 16582)	This is GetMethodName, a member of class TDataSetButton.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16582)	This is GetSubComponents, a member of class TDataSetButton.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 17043)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	ImplAddElements (see page 16583)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 16583)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 16869)	Provides the underlying implementation for the Content (see page 16868) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 17044)	Forwards notification messages to all owned components.

	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayComponent (see page 17044)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 17044)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.
R	XMLDisplay (see page 17045)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16655)	Creates and initializes an instance of TFirstButton (see page 16651).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 17042)	Frees an instance of TXMLDisplayReferenceButton (see page 17037).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.

	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 16870)	Specifies the caption that appears on the button.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16871)	Represents a tag option and its value in the generated INPUT tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16871)	Adds style attributes to the generated button control.
	StyleRule (see page 16872)	Applies a style in the page producer's style sheet to the button.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	XMLComponent (see page 17045)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 17045)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 16584)	Specifies the caption that appears on the button.
	Custom (see page 16584)	Represents a tag option and its value in the generated INPUT tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16584)	Adds style attributes to the generated button control.
	StyleRule (see page 16585)	Applies a style in the page producer's style sheet to the button.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLComponent (see page 16585)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 16586)	Specifies whether the button is associated with its parent's display object.

99.1.21.2 TFirstButton.Create Constructor

Creates and initializes an instance of TFirstButton (see page 16651).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TFirstButton(TComponent * AOwner);
```

Description

Call Create to instantiate TFirstButton (see page 16651) at runtime. Buttons (see page 2289) added at design time using the Web Page editor are instantiated automatically.

99.1.22 TFormLayout Class

TFormLayout lets child Web items adhere to layout rules imposed by their parent.

Class Hierarchy

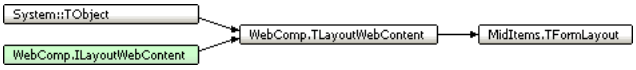

Pascal

```
TFormLayout = class(TLayoutWebContent);
```

C++

```
class TFormLayout : public TLayoutWebContent;
```

File

MidItems

Description

TFormLayout is the default implementation of the ILayoutWebContent interface for InternetExpress applications. The Web items that belong to an InetXPageProducer object pass a TFormLayout instance to their child Web items as the ParentLayout parameter of their Content method. This object provides helper functions that the child items use to fit their generated content into the parent object's HTML structure.

Use the TFormLayout properties to configure how the ILayoutWebContent methods convert HTML that child Web items generate into a final form.

See Also

TLayout

IWebContent (see page 24792)

ILayoutWebContent (see page 24774)

TInetXPageProducer (see page 17145)

99.1.22.1 TFormLayout Members

The following tables list the members exposed by TFormLayout.

Protected Methods

	Name	Description
	_AddRef (see page 24700)	Responds when the caller acquires a reference to this object's interface.
	_Release (see page 24701)	Responds when the caller releases a reference to this object's interface.
	EndButtons (see page 16659)	This is EndButtons, a member of class TFormLayout.
	EndFields (see page 16659)	This is EndFields, a member of class TFormLayout.
	EndRow (see page 16659)	This is EndRow, a member of class TFormLayout.
	EndTable (see page 16660)	This is EndTable, a member of class TFormLayout.
V	ImplLayoutButton (see page 16660)	Provides the underlying implementation for the LayoutButton method.
V	ImplLayoutField (see page 16660)	Provides the underlying implementation for the LayoutField method.
V	ImplLayoutLabelAndField (see page 16661)	Provides the underlying implementation for the LayoutLabelAndField method.
V	ImplLayoutTable (see page 16661)	Provides the underlying implementation for the LayoutTable method.

	LayoutButton (see page 24703)	Converts the HTML declaring a button into a form the associated Web item can use.
	LayoutField (see page 24704)	Converts the HTML declaring a field into a form the associated Web item can use.
	LayoutLabelAndField (see page 24704)	Converts the HTML declaring a labeled field into a form the associated Web item can use.
	LayoutTable (see page 24705)	This is LayoutTable, a member of class TLayoutWebContent.
	NextColumn (see page 16662)	This is NextColumn, a member of class TFormLayout.
V	QueryInterface (see page 24705)	Returns a reference to a specified interface if the object supports that interface.
	StartButtons (see page 16662)	This is StartButtons, a member of class TFormLayout.
	StartFields (see page 16662)	This is StartFields, a member of class TFormLayout.
	StartRow (see page 16662)	This is StartRow, a member of class TFormLayout.
	StartTable (see page 16663)	This is StartTable, a member of class TFormLayout.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 16658)	Creates a new TFormLayout (see page 16656) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndLayout (see page 16659)	Returns the string of HTML to insert after laying out all child items.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LayoutButton (see page 24775)	This is LayoutButton, a member of class ILayoutWebContent.
	LayoutField (see page 24775)	This is LayoutField, a member of class ILayoutWebContent.
	LayoutLabelAndField (see page 24775)	Converts the HTML declaring a labeled field into a form the Web item's parent can use.
	LayoutTable (see page 24776)	This is LayoutTable, a member of class ILayoutWebContent.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	BreakButtons (see page 16663)	Indicates whether the associated Web item uses a grid to lay out buttons that child Web items generate.

	ColumnCount (see page 16663)	Indicates the number of child elements appear in a single row when the associated Web item uses a grid to lay them out.
R	ParentLayout (see page 24706)	Provides access to the layout object of the Web item that is the parent of the Web item to which this object belongs.
	TableHeader (see page 16664)	Specifies the HTML to generate at the beginning of a table that lays out child elements.

99.1.22.2 TFormLayout.Create Constructor

Creates a new TFormLayout (see page 16656) object.

Pascal

```
constructor Create(AParentLayout: TLayout);
```

C++

```
__fastcall TFormLayout(TLayout AParentLayout);
```

Description

Call Create to instantiate a TFormLayout (see page 16656) object to use when fetching the content produced by child Web items.

AParentLayout is the layout object of the parent Web item (the grandparent of the child Web items that call this object's methods). AParentLayout becomes the value of the ParentLayout property.

See Also

ParentLayout (see page 24706)

99.1.22.3 TFormLayout Methods

The methods of the TFormLayout class are listed here.

Protected Methods

	Name	Description
	EndButtons (see page 16659)	This is EndButtons, a member of class TFormLayout.
	EndFields (see page 16659)	This is EndFields, a member of class TFormLayout.
	EndRow (see page 16659)	This is EndRow, a member of class TFormLayout.
	EndTable (see page 16660)	This is EndTable, a member of class TFormLayout.
V	ImplLayoutButton (see page 16660)	Provides the underlying implementation for the LayoutButton method.
V	ImplLayoutField (see page 16660)	Provides the underlying implementation for the LayoutField method.
V	ImplLayoutLabelAndField (see page 16661)	Provides the underlying implementation for the LayoutLabelAndField method.
V	ImplLayoutTable (see page 16661)	Provides the underlying implementation for the LayoutTable method.
	NextColumn (see page 16662)	This is NextColumn, a member of class TFormLayout.
	StartButtons (see page 16662)	This is StartButtons, a member of class TFormLayout.
	StartFields (see page 16662)	This is StartFields, a member of class TFormLayout.
	StartRow (see page 16662)	This is StartRow, a member of class TFormLayout.
	StartTable (see page 16663)	This is StartTable, a member of class TFormLayout.

Public Methods

	Name	Description
	EndLayout (see page 16659)	Returns the string of HTML to insert after laying out all child items.

99.1.22.3.1 TFormLayout.EndButtons Method

Pascal

```
function EndButtons: string;
```

C++

```
__fastcall AnsiString EndButtons();
```

Description

This is EndButtons, a member of class TFormLayout.

99.1.22.3.2 TFormLayout.EndFields Method

Pascal

```
function EndFields: string;
```

C++

```
__fastcall AnsiString EndFields();
```

Description

This is EndFields, a member of class TFormLayout.

99.1.22.3.3 TFormLayout.EndLayout Method

Returns the string of HTML to insert after laying out all child items.

Pascal

```
function EndLayout: string;
```

C++

```
__fastcall AnsiString EndLayout();
```

Description

EndLayout generates the tags that must be inserted after all the content that child Web items generate. TFormLayout (see page 16656) keeps track of the ILayoutWebContent method calls that are made so that it can know what tags need to be closed.

See Also

ILayoutWebContent (see page 24774)

99.1.22.3.4 TFormLayout.EndRow Method

Pascal

```
function EndRow: string;
```

C++

```
__fastcall AnsiString EndRow();
```

Description

This is EndRow, a member of class TFormLayout.

99.1.22.3.5 TFormLayout.EndTable Method

Pascal

```
function EndTable: string;
```

C++

```
__fastcall AnsiString EndTable();
```

Description

This is EndTable, a member of class TFormLayout.

99.1.22.3.6 TFormLayout.ImplLayoutButton Method

Provides the underlying implementation for the LayoutButton method.

Pascal

```
function ImplLayoutButton(const HTMLButton: string; Attributes: TLayoutAttributes): string;
override;
```

C++

```
virtual __fastcall AnsiString ImplLayoutButton(const AnsiString HTMLButton, TLayoutAttributes
Attributes);
```

Description

The LayoutButton method calls ImplLayoutButton to convert the HTML that declares a button (typically an <INPUT> tag) into the form that the associated Web item requires.

HTMLButton is the HTML that declares a button. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the button.

LayoutButton returns the HTML for the button in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutButton is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutButton method.

See Also

LayoutButton (see page 24703)

99.1.22.3.7 TFormLayout.ImplLayoutField Method

Provides the underlying implementation for the LayoutField method.

Pascal

```
function ImplLayoutField(const HTMLField: string; Attributes: TLayoutAttributes): string;
override;
```

C++

```
virtual __fastcall AnsiString ImplLayoutField(const AnsiString HTMLField, TLayoutAttributes
Attributes);
```

Description

The LayoutField method calls ImplLayoutField to convert the HTML that declares a field (an input control that represents a data field or a named value) into the form that the associated Web item requires.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field.

LayoutField returns the HTML for the field in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutField is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutField method.

See Also

LayoutField (see page 24704)

99.1.22.3.8 TFormLayout.ImplLayoutLabelAndField Method

Provides the underlying implementation for the LayoutLabelAndField method.

Pascal

```
function ImplLayoutLabelAndField(const HTMLLabel: string; const HTMLField: string; Attributes:
TLayoutAttributes): string; override;
```

C++

```
virtual __fastcall AnsiString ImplLayoutLabelAndField(const AnsiString HTMLLabel, const
AnsiString HTMLField, TLayoutAttributes Attributes);
```

Description

The LayoutLabelAndField method calls ImplLayoutLabelAndField to convert the HTML that declares a labeled field (an input control that represents a data field or a named value along with its label) into the form that the associated Web item requires.

HTMLLabel is the HTML that declares the label. This is produced by the child Web item.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field and its label.

LayoutLabelAndField returns the HTML for the field and label in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutLabelAndField is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutLabelAndField method.

See Also

LayoutLabelAndField (see page 24704)

99.1.22.3.9 TFormLayout.ImplLayoutTable Method

Provides the underlying implementation for the LayoutTable method.

Pascal

```
function ImplLayoutTable(const HTMLTable: string; Attributes: TLayoutAttributes): string;
override;
```

C++

```
virtual __fastcall AnsiString ImplLayoutTable(const AnsiString HTMLTable, TLayoutAttributes
Attributes);
```

Description

The LayoutTable method calls ImplLayoutTable to convert the HTML that declares a table (typically a structure used to layout the content produced by the child Web item's children) into the form that the associated Web item requires.

HTMLTable is the HTML that declares the table. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the table.

LayoutTable returns the HTML for the table in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutTable is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutTable method.

See Also

layouttable (see page 24705)

99.1.22.3.10 TFormLayout.NextColumn Method

Pascal

```
function NextColumn(SubItemCount: Integer): string;
```

C++

```
__fastcall AnsiString NextColumn(int SubItemCount);
```

Description

This is NextColumn, a member of class TFormLayout.

99.1.22.3.11 TFormLayout.StartButtons Method

Pascal

```
function StartButtons: string;
```

C++

```
__fastcall AnsiString StartButtons();
```

Description

This is StartButtons, a member of class TFormLayout.

99.1.22.3.12 TFormLayout.StartFields Method

Pascal

```
function StartFields(ColCount: Integer): string;
```

C++

```
__fastcall AnsiString StartFields(int ColCount);
```

Description

This is StartFields, a member of class TFormLayout.

99.1.22.3.13 TFormLayout.StartRow Method

Pascal

```
function StartRow: string;
```

C++

```
__fastcall AnsiString StartRow();
```

Description

This is StartRow, a member of class TFormLayout.

99.1.22.3.14 TFormLayout.StartTable Method

Pascal

```
function StartTable: string;
```

C++

```
__fastcall AnsiString StartTable();
```

Description

This is StartTable, a member of class TFormLayout.

99.1.22.4 TFormLayout Properties

The properties of the TFormLayout class are listed here.

Public Properties

	Name	Description
	BreakButtons (see page 16663)	Indicates whether the associated Web item uses a grid to lay out buttons that child Web items generate.
	ColumnCount (see page 16663)	Indicates the number of child elements appear in a single row when the associated Web item uses a grid to lay them out.
	TableHeader (see page 16664)	Specifies the HTML to generate at the beginning of a table that lays out child elements.

99.1.22.4.1 TFormLayout.BreakButtons Property

Indicates whether the associated Web item uses a grid to lay out buttons that child Web items generate.

Pascal

```
property BreakButtons: Boolean;
```

C++

```
__property Boolean BreakButtons;
```

Description

Set BreakButtons to true if the Web item that supplies this object to its child Web items uses an HTML table to position HTML buttons that those child Web items generate. Set BreakButtons to false if the Web item does not use an HTML table to position child buttons.

BreakButtons affects the whether the LayoutButton method on this object's ILayoutWebContent interface adds <TD> tags to the HTML that child Web items generate for HTML buttons.

See Also

layoutbutton (see page 24775)

99.1.22.4.2 TFormLayout.ColumnCount Property

Indicates the number of child elements appear in a single row when the associated Web item uses a grid to lay them out.

Pascal

```
property ColumnCount: Integer;
```

C++

```
__property int ColumnCount;
```

Description

Set ColumnCount to the number of child elements that appear in a single row when laying out child controls by rows.

ColumnCount affects where the LayoutField or LayoutButton method on this object's ILayoutWebContent interface adds <TR> and </TR> tags to the HTML that child Web items generate.

Note: ColumnCount only affects the LayoutButton method if the value of the BreakButtons (see page 16663) property is true.

See Also

layoutbutton (see page 24775)

layoutfield (see page 24775)

BreakButtons (see page 16663)

99.1.22.4.3 TFormLayout.TableHeader Property

Specifies the HTML to generate at the beginning of a table that lays out child elements.

Pascal

```
property TableHeader: string;
```

C++

```
__property AnsiString TableHeader;
```

Description

Set TableHeader to the HTML tag that appears at the beginning of the HTML table used to control the layout of child elements. TableHeader is a string that includes the <table> tag, along with any attributes (such as alignment) that affect the layout of child items.

See Also

layoutbutton (see page 24775)

layoutfield (see page 24775)

BreakButtons (see page 16663)

99.1.23 TGridAttributes Class

TGridAttributes describes the options the TDataGrid (see page 16560) component adds to an HTML <TABLE> tag.

Class Hierarchy

Pascal

```
TGridAttributes = class(TPersistent);
```

C++

```
class TGridAttributes : public TPersistent;
```

File

MidItems

Description

TGridAttributes implements the TableAttributes property of a TDataGrid (see page 16560) or TAdapterGrid component. It describes display attributes that apply to an HTML table generated by TDataGrid (see page 16560) or TAdapterGrid. The properties of TGridAttributes specify the options added to the generated HTML <TABLE> tag, as well as any styles that should be applied.

See Also

TCustomDataGrid (see page 16499)

TCustomAdapterGrid (see page 25286)

99.1.23.1 TGridAttributes Members

The following tables list the members exposed by TGridAttributes.

Protected Methods

	Name	Description
	AssignTo (see page 16667)	Copies the properties of an object to a destination object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 16666)	Creates an instance of TGridAttributes (see page 16664).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Parent (see page 16670)	Indicates the TDataGrid (see page 16560) object that uses this TGridAttributes (see page 16664) to describe its attributes.

Published Properties

	Name	Description
	Align (see page 16668)	Specifies the alignment of the table within the HTML document.
	BgColor (see page 16668)	Specifies the background color of the HTML table.
	Border (see page 16669)	Specifies the thickness of the border drawn around the HTML table.
	CellPadding (see page 16669)	This is CellPadding, a member of class TGridAttributes.
	CellSpacing (see page 16669)	Specifies the amount of space to leave between cells in the HTML table.
	Custom (see page 16670)	Represents a tag option and its value in an HTML tag.
	Style (see page 16670)	Adds style attributes to the generated table.
	StyleRule (see page 16671)	This is StyleRule, a member of class TGridAttributes.

99.1.23.2 TGridAttributes.Create Constructor

Creates an instance of TGridAttributes (see page 16664).

Pascal

```
constructor Create(AParent: TComponent);
```

C++

```
__fastcall TGridAttributes(TComponent * AParent);
```

Description

Most applications have no need to instantiate TGridAttributes (see page 16664). TGridAttributes (see page 16664) is instantiated by the TDataGrid (see page 16560) component that uses it.

The AParent parameter is the TDataGrid (see page 16560) or TAdapterGrid component that uses the TGridAttributes (see page 16664) instance. This parameter becomes the value of the Parent (see page 16670) property.

See Also

Parent (see page 16670)

TDataGrid (see page 16560)

TAdapterGrid (see page 25175)

99.1.23.3 TGridAttributes Methods

The methods of the TGridAttributes class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 16667)	Copies the properties of an object to a destination object.

99.1.23.3.1 TGridAttributes.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign (see page 2930)

EConvertError (see page 22724)

99.1.23.4 TGridAttributes Properties

The properties of the TGridAttributes class are listed here.

Public Properties

	Name	Description
	Parent (see page 16670)	Indicates the TDataGrid (see page 16560) object that uses this TGridAttributes (see page 16664) to describe its attributes.

Published Properties

	Name	Description
	Align (see page 16668)	Specifies the alignment of the table within the HTML document.
	BgColor (see page 16668)	Specifies the background color of the HTML table.
	Border (see page 16669)	Specifies the thickness of the border drawn around the HTML table.
	CellPadding (see page 16669)	This is CellPadding, a member of class TGridAttributes.

	CellSpacing (see page 16669)	Specifies the amount of space to leave between cells in the HTML table.
	Custom (see page 16670)	Represents a tag option and its value in an HTML tag.
	Style (see page 16670)	Adds style attributes to the generated table.
	StyleRule (see page 16671)	This is StyleRule, a member of class TGridAttributes.

99.1.23.4.1 TGridAttributes.Align Property

Specifies the alignment of the table within the HTML document.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Set Align to indicate the desired alignment of the table within the HTML document, or within the cell of a surrounding HTML table in a layout group. Align indicates the desired value of the Align option of the HTML table tag.

The default value of Align is haDefault, which means the web browser that reads the HTML document positions the HTML table where it wants.

See Also

cellpadding (see page 16669)

Border (see page 16669)

99.1.23.4.2 TGridAttributes.BgColor Property

Specifies the background color of the HTML table.

Pascal

```
property BgColor: THTMLBgColor;
```

C++

```
__property THTMLBgColor BgColor;
```

Description

Use BgColor to indicate the desired background color of the HTML table. The value of BgColor is the name of color. This can be one of the following strings:

Aqua	Black	Blue	Fuchsia
Gray	Green	Lime	Maroon
Navy	Olive	Purple	Red
Silver	Teal	White	Yellow

Note: The row or heading attributes can override this background color for a portion of the HTML table.

See Also

BgColor (see page 16675)

99.1.23.4.3 TGridAttributes.Border Property

Specifies the thickness of the border drawn around the HTML table.

Pascal

```
property Border: Integer;
```

C++

```
__property int Border;
```

Description

Set Border to specify whether a border should be drawn around the table, and if so, how thick it should be. The thickness specified by any particular value of border depends on the web browser that views the HTML table. For any browser, however, larger numbers indicate thicker borders. A value of -1 indicates that no border should be drawn around the table.

See Also

CellSpacing (see page 16669)

99.1.23.4.4 TGridAttributes.CellPadding Property

Pascal

```
property CellPadding: Integer;
```

C++

```
__property int CellPadding;
```

Description

This is CellPadding, a member of class TGridAttributes.

99.1.23.4.5 TGridAttributes.CellSpacing Property

Specifies the amount of space to leave between cells in the HTML table.

Pascal

```
property CellSpacing: Integer;
```

C++

```
__property int CellSpacing;
```

Description

Set CellSpacing to specify how many pixels of whitespace should appear between the cells in the table. A value of -1 indicates that the web browser should decide how to separate the cells of the HTML table.

Note: CellSpacing refers to the whitespace between cells in an HTML table, not the thickness of the lines drawn to separate rows and columns.

See Also

Border (see page 16669)

cellpadding (see page 16669)

99.1.23.4.6 TGridAttributes.Custom Property

Represents a tag option and its value in an HTML tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to represent one or more options to be added to the HTML <TABLE> tag. Each option and its value should have the form

`OPTIONNAME=Value`

The options specified by Custom should be separated by spaces. For example, to hide the side borders and add row dividers, you can set Custom to

`FRAME=HSIDES RULES=ROWS`

See Also

Style (see page 16670)

stylerule (see page 16671)

99.1.23.4.7 TGridAttributes.Parent Property

Indicates the TDataGrid (see page 16560) object that uses this TGridAttributes (see page 16664) to describe its attributes.

Pascal

```
property Parent: TComponent;
```

C++

```
__property TComponent * Parent;
```

Description

Read Parent to access the TDataGrid (see page 16560) that uses this TGridAttributes (see page 16664) object. The value of Parent is assigned by the constructor.

See Also

TDataGrid (see page 16560)

Create (see page 16666)

TAdapterGrid (see page 25175)

99.1.23.4.8 TGridAttributes.Style Property

Adds style attributes to the generated table.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the entire table. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the <TABLE> tag as the value of a STYLE option.

See Also

Custom (see page 16670)

stylerule (see page 16671)

99.1.23.4.9 TGridAttributes.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TGridAttributes.

99.1.24 TGridRowAttributes Class

TGridRowAttributes describes the options the TDataGrid (see page 16560) component adds to its HTML row tags (<TR>).

Class Hierarchy

Pascal

```
TGridRowAttributes = class(TPersistent);
```

C++

```
class TGridRowAttributes : public TPersistent;
```

File

MidItems

Description

TGridRowAttributes implements the RowAttributes and HeadingAttributes properties of a TDataGrid (see page 16560) or TAdapterGrid component. It describes display attributes that apply to the generated HTML table's rows. The properties of TGridRowAttributes specify the options added to the generated HTML <TR> tag, as well as any styles that should be applied to the header or body of the table.

See Also

TCustomDataGrid (see page 16499)

TCustomDataGrid (see page 16499)

TCustomAdapterGrid (see page 25286)

TCustomAdapterGrid (see page 25286)

99.1.24.1 TGridRowAttributes Members

The following tables list the members exposed by TGridRowAttributes.

Protected Methods

	Name	Description
	AssignTo (see page 16673)	Copies the properties of an object to a destination object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 16673)	Creates an instance of TGridRowAttributes (see page 16671).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Parent (see page 16676)	Indicates the TDataGrid (see page 16560) or TAdapterGrid object that uses this TGridRowAttributes (see page 16671) to describe its header or row attributes.

Published Properties

	Name	Description
	Align (see page 16674)	Specifies the alignment of the table within the HTML document.
	BgColor (see page 16675)	Specifies the background color of the rows or header of an HTML table.
	Custom (see page 16675)	Represents a tag option and its value in an HTML tag.
	Style (see page 16676)	Adds style attributes to the generated table body or header.
	StyleRule (see page 16676)	Applies a style in the page producer's style sheet to the generated table body or header.
	VAlign (see page 16677)	Specifies the vertical alignment within the cells of the HTML table body or header.

99.1.24.2 TGridRowAttributes.Create Constructor

Creates an instance of TGridRowAttributes (see page 16671).

Pascal

```
constructor Create(AParent: TComponent);
```

C++

```
__fastcall TGridRowAttributes(TComponent * AParent);
```

Description

Most applications have no need to instantiate TGridRowAttributes (see page 16671). TGridRowAttributes (see page 16671) is instantiated by the TDataGrid (see page 16560) or TAdapterGrid component that uses it.

The AParent parameter is the TDataGrid (see page 16560) or TAdapterGrid component that uses the TGridRowAttributes (see page 16671) instance. This parameter becomes the value of the Parent (see page 16676) property.

See Also

Parent (see page 16676)

TDataGrid (see page 16560)

TAdapterGrid (see page 25175)

99.1.24.3 TGridRowAttributes Methods

The methods of the TGridRowAttributes class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 16673)	Copies the properties of an object to a destination object.

99.1.24.3.1 TGridRowAttributes.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign (see page 2930)

EConvertError (see page 22724)

99.1.24.4 TGridRowAttributes Properties

The properties of the TGridRowAttributes class are listed here.

Public Properties

	Name	Description
R	Parent (see page 16676)	Indicates the TDataGrid (see page 16560) or TAdapterGrid object that uses this TGridRowAttributes (see page 16671) to describe its header or row attributes.

Published Properties

	Name	Description
	Align (see page 16674)	Specifies the alignment of the table within the HTML document.
	BgColor (see page 16675)	Specifies the background color of the rows or header of an HTML table.
	Custom (see page 16675)	Represents a tag option and its value in an HTML tag.
	Style (see page 16676)	Adds style attributes to the generated table body or header.
	StyleRule (see page 16676)	Applies a style in the page producer's style sheet to the generated table body or header.
	VAlign (see page 16677)	Specifies the vertical alignment within the cells of the HTML table body or header.

99.1.24.4.1 TGridRowAttributes.Align Property

Specifies the alignment of the table within the HTML document.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Set Align to indicate the desired alignment of the contents of each cell in the table row or header. Align indicates the desired value of the Align option of the <TR> tag.

The default value of Align is haDefault, which means the web browser that reads the HTML document positions the cell contents wherever it wants horizontally.

See Also

VAlign (see page 16677)

99.1.24.4.2 TGridRowAttributes.BgColor Property

Specifies the background color of the rows or header of an HTML table.

Pascal

```
property BgColor: THTMLBgColor;
```

C++

```
__property THTMLBgColor BgColor;
```

Description

Use BgColor to indicate the desired background color of the rows or heading of an HTML table. The value of BgColor is the name of color. This can be one of the following strings:

Aqua	Black	Blue	Fuchsia
Gray	Green	Lime	Maroon
Navy	Olive	Purple	Red
Silver	Teal	White	Yellow

Note: This property overrides the background color option of the HTML table itself.

See Also

BgColor (see page 16668)

99.1.24.4.3 TGridRowAttributes.Custom Property

Represents a tag option and its value in an HTML tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to represent one or more options to be added to the HTML <TR> tag. Each option and its value should have the form

```
OPTIONNAME=Value
```

To specify multiple options, separate the Name/value pairs with spaces.

See Also

Style (see page 16676)

StyleRule (see page 16676)

99.1.24.4.4 TGridRowAttributes.Parent Property

Indicates the TDataGrid (see page 16560) or TAdapterGrid object that uses this TGridRowAttributes (see page 16671) to describe its header or row attributes.

Pascal

```
property Parent: TComponent;
```

C++

```
__property TComponent * Parent;
```

Description

Read Parent to access the TDataGrid (see page 16560) or TAdapterGrid that uses this TGridRowAttributes (see page 16671) object. The value of Parent is assigned by the constructor.

See Also

TDataGrid (see page 16560)

TAdapterGrid (see page 25175)

Create (see page 16673)

99.1.24.4.5 TGridRowAttributes.Style Property

Adds style attributes to the generated table body or header.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the rows or header of the HTML table. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the <TR> tag as the value of a STYLE option.

See Also

Custom (see page 16675)

StyleRule (see page 16676)

99.1.24.4.6 TGridRowAttributes.StyleRule Property

Applies a style in the page producer's style sheet to the generated table body or header.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated table body or header. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

To apply a style definition that is not included in the style sheet, use the Style (see page 16676) property instead.

See Also

Style (see page 16676)

Styles (see page 17127)

StylesFile (see page 17128)

Styles (see page 19830)

StylesFile (see page 19831)

99.1.24.4.7 TGridRowAttributes.VAlign Property

Specifies the vertical alignment within the cells of the HTML table body or header.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

Set VAlign to indicate the desired vertical alignment of the contents of each cell. VAlign indicates the desired value of the VAlign option of the <TR> tag for the table rows or header.

The default value of VAlign is haVDefault, which means the web browser that reads the HTML document positions the contents of cells in the table element wherever it wishes vertically.

See Also

Align (see page 16674)

99.1.25 TInsertButton Class

TInsertButton generates an HTML control that inserts a new record in an XML data packet.

Class Hierarchy

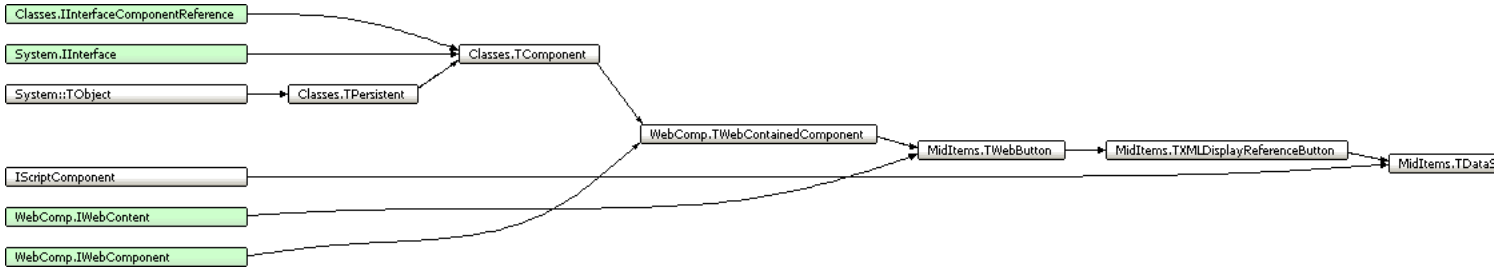

Pascal

```
TInsertButton = class(TDataSetButton);
```

C++

```
class TInsertButton : public TDataSetButton;
```

File

MidItems

Description

Add TInsertButton as the child of a TDataNavigator (see page 16569) component (or a layout group that is a child of a TDataNavigator (see page 16569) component) to enable the generated HTML form to add records to an XML data packet. The resulting HTML button's action adds a new blank record to the emerging XML delta packet and causes the associated display control to display that new record.

Use the properties of TInsertButton to customize the appearance of the button and associate it with a Web item. The associated Web item generates the HTML control that displays the newly inserted record.

See Also

TXMLBroker (see page 26049)

TDataNavigator (see page 16569)

TLayoutGroup (see page 16687)

TApplyUpdatesButton (see page 16480)

99.1.25.1 TInsertButton Members

The following tables list the members exposed by TInsertButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 16581)	This is DefaultCaption, a member of class TDataSetButton.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	XMLMethodName (see page 16581)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16582)	This is AddElements, a member of class TDataSetButton.

	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16868)	Returns the HTML that defines the button control.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetCaption (see page 16582)	Returns the value of the Caption property.
	GetCaption (see page 16869)	Returns the value of the Caption (see page 16870) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDisplayComponent (see page 17043)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 17043)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
	GetMethodName (see page 16582)	This is GetMethodName, a member of class TDataSetButton.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16582)	This is GetSubComponents, a member of class TDataSetButton.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 17043)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
	ImplAddElements (see page 16583)	This is ImplAddElements, a member of class TDataSetButton.
	ImplContent (see page 16583)	Provides the underlying implementation for the Content method.
	ImplContent (see page 16869)	Provides the underlying implementation for the Content (see page 16868) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 17044)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayComponent (see page 17044)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 17044)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.

	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.
	XMLDisplay (see page 17045)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
	Create (see page 16682)	Creates and initializes an instance of TInsertButton (see page 16677).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 17042)	Frees an instance of TXMLDisplayReferenceButton (see page 17037).
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 16870)	Specifies the caption that appears on the button.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.

	Custom (see page 16871)	Represents a tag option and its value in the generated INPUT tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16871)	Adds style attributes to the generated button control.
	StyleRule (see page 16872)	Applies a style in the page producer's style sheet to the button.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	XMLComponent (see page 17045)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 17045)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 16584)	Specifies the caption that appears on the button.
	Custom (see page 16584)	Represents a tag option and its value in the generated INPUT tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16584)	Adds style attributes to the generated button control.
	StyleRule (see page 16585)	Applies a style in the page producer's style sheet to the button.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLComponent (see page 16585)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 16586)	Specifies whether the button is associated with its parent's display object.

99.1.25.2 TInsertButton.Create Constructor

Creates and initializes an instance of TInsertButton (see page 16677).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TInsertButton(TComponent * AOwner);
```

Description

Call Create to instantiate TInsertButton (see page 16677) at runtime. Buttons (see page 2289) added at design time using the Web Page editor are instantiated automatically.

99.1.26 TLastButton Class

TLastButton generates an HTML control that moves to the last record in an XML data packet.

Class Hierarchy

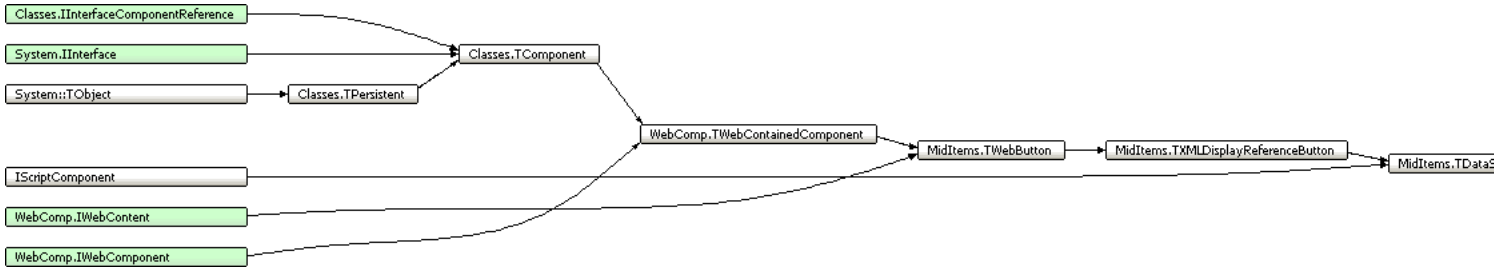

Pascal

```
TLastButton = class(TDataSetButton);
```

C++

```
class TLastButton : public TDataSetButton;
```

File

MidItems

Description

Add TLastButton as the child of a TDataNavigator (see page 16569) component (or a layout group that is a child of a TDataNavigator (see page 16569) component) to enable the generated HTML form to move quickly to the last record in a dataset. The resulting HTML button's action changes the associated display control so that it shows the last record as current.

Use the properties of TLastButton to customize the appearance of the button and associate it with a Web item. The associated Web item displays the records that the last button navigates.

See Also

TXMLBroker (see page 26049)

TDataNavigator (see page 16569)

TLayoutGroup (see page 16687)

99.1.26.1 TLastButton Members

The following tables list the members exposed by TLastButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 16581)	This is DefaultCaption, a member of class TDataSetButton.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	XMLMethodName (see page 16581)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16582)	This is AddElements, a member of class TDataSetButton.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.

	Content (see page 16868)	Returns the HTML that defines the button control.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetCaption (see page 16582)	Returns the value of the Caption property.
V	GetCaption (see page 16869)	Returns the value of the Caption (see page 16870) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDisplayComponent (see page 17043)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 17043)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
V	GetMethodName (see page 16582)	This is GetMethodName, a member of class TDataSetButton.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16582)	This is GetSubComponents, a member of class TDataSetButton.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 17043)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	ImplAddElements (see page 16583)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 16583)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 16869)	Provides the underlying implementation for the Content (see page 16868) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 17044)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayComponent (see page 17044)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 17044)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.
R	XMLDisplay (see page 17045)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16687)	Creates and initializes an instance of TLastButton (see page 16682).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 17042)	Frees an instance of TXMLDisplayReferenceButton (see page 17037).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 16870)	Specifies the caption that appears on the button.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16871)	Represents a tag option and its value in the generated INPUT tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.

	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16871)	Adds style attributes to the generated button control.
	StyleRule (see page 16872)	Applies a style in the page producer's style sheet to the button.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	XMLComponent (see page 17045)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 17045)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 16584)	Specifies the caption that appears on the button.
	Custom (see page 16584)	Represents a tag option and its value in the generated INPUT tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16584)	Adds style attributes to the generated button control.
	StyleRule (see page 16585)	Applies a style in the page producer's style sheet to the button.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLComponent (see page 16585)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 16586)	Specifies whether the button is associated with its parent's display object.

99.1.26.2 TLastButton.Create Constructor

Creates and initializes an instance of TLastButton (see page 16682).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TLastButton(TComponent * AOwner);
```

Description

Call Create to instantiate TLastButton (see page 16682) at runtime. Buttons (see page 2289) added at design time using the Web Page editor are instantiated automatically.

99.1.27 TLayoutGroup Class

TLayoutGroup generates the HTML to arrange content produced by other Web items.

Class Hierarchy

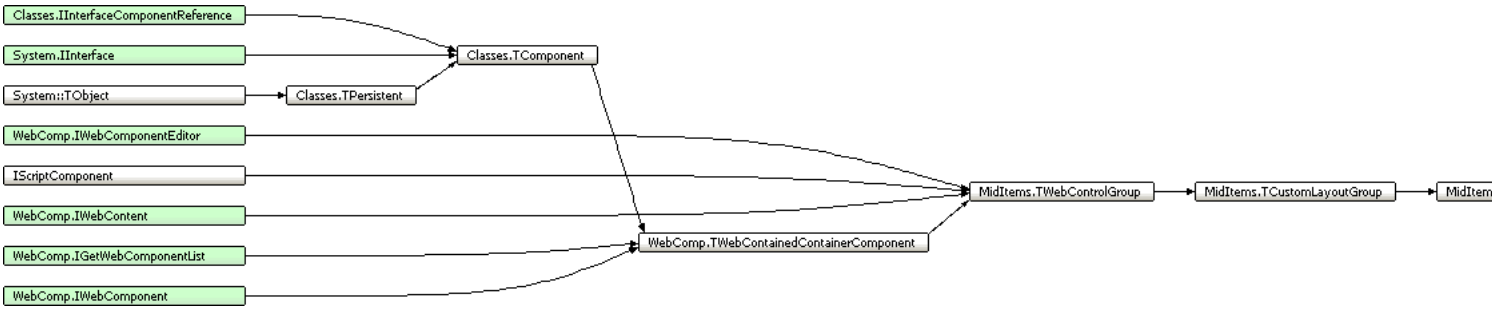

Pascal

```
TLayoutGroup = class(TCustomLayoutGroup);
```

C++

```
class TLayoutGroup : public TCustomLayoutGroup;
```

File

MidItems

Description

Use TLayoutGroup to adjust the layout of items in an HTML document. There are few restrictions on what Web items can use a layout group because it has no special content that it displays. It merely imposes a layout on content generated by other Web items.

TLayoutGroup maintains a list of Web items that display information in the final HTML document, but imposes no restrictions of its own on the possible types of Web items. Instead, it restricts those Web items to the classes that its parent allows. In order to let the parent component use the HTML that TLayoutGroup generates, TLayoutGroup inherits the IWebContent interface from TWebControlGroup (see page 16872).

See Also

IWebContent (see page 24792)

Using the Web Page Editor

99.1.27.1 TLayoutGroup Members

The following tables list the members exposed by TLayoutGroup.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.

	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetLayoutAttributes (see page 16534)	Returns an object that specifies how the generated group of controls should look.
	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
	ImplCanAddClass (see page 16534)	Provides the underlying implementation for the CanAddClass method.
	ImplCanAddClass (see page 16881)	Provides the underlying implementation for the CanAddClass (see page 16878) method.
	ImplContent (see page 16535)	Provides the underlying implementation for the Content method.
	ImplContent (see page 16882)	Provides the underlying implementation for the Content (see page 16879) method.
	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16533)	Creates and initializes an instance of TCustomLayoutGroup (see page 16528).
V	Create (see page 16877)	Creates and initializes an instance of TWebControlGroup (see page 16872).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24742)	Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.

	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 16533)	Frees an instance of TCustomLayoutGroup (see page 16528).
	Destroy (see page 16877)	Frees an instance of TWebControlGroup (see page 16872).
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.

	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Custom (see page 16692)	Represents a tag option and its value in the generated TABLE tag.
	DisplayColumns (see page 16693)	Specifies the number of columns the layout group uses to lay out its child controls.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16693)	Adds style attributes to the generated HTML table.
	StyleRule (see page 16694)	Applies a style in the page producer's style sheet to the group of buttons.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.27.2 TLayoutGroup Properties

The properties of the TLayoutGroup class are listed here.

Published Properties

	Name	Description
	Custom (see page 16692)	Represents a tag option and its value in the generated TABLE tag.
	DisplayColumns (see page 16693)	Specifies the number of columns the layout group uses to lay out its child controls.
	Style (see page 16693)	Adds style attributes to the generated HTML table.
	StyleRule (see page 16694)	Applies a style in the page producer's style sheet to the group of buttons.

99.1.27.2.1 TLayoutGroup.Custom Property

Represents a tag option and its value in the generated TABLE tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TCustomLayoutGroup generates an HTML table to lay out the controls generated by the Web items in its WebFieldControls property. Use Custom to add options that qualify the TABLE tag. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add a border around the group of controls and color the background blue, you can set Custom to

```
BORDER=2 BGCOLOR=BLUE
```

See Also

Style (see page 16536)

StyleRule (see page 16537)

WebFieldControls (see page 16884)

DisplayColumns (see page 16536)

99.1.27.2.2 TLayoutGroup.DisplayColumns Property

Specifies the number of columns the layout group uses to lay out its child controls.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

TCustomLayoutGroup generates an HTML table to lay out the controls generated by the Web items in its WebFieldControls property. It adds the HTML generated by each Web item listed in WebFieldControls to a cell of the table, filling the cells out row by row until it displays all the Web items. DisplayColumns determines the number of cells in each row.

See Also

WebFieldControls (see page 16884)

99.1.27.2.3 TLayoutGroup.Style Property

Adds style attributes to the generated HTML table.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

TCustomLayoutGroup generates an HTML table to lay out the controls generated by the Web items in WebFieldControls. Use Style to add a STYLE option that qualifies the TABLE tag. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated TABLE tag as the value of a STYLE option.

See Also

Custom (see page 16535)

StyleRule (see page 16537)

WebFieldControls (see page 16884)

99.1.27.2.4 TLayoutGroup.StyleRule Property

Applies a style in the page producer's style sheet to the group of buttons.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

TCustomLayoutGroup generates an HTML table to lay out the controls generated by its Web items. Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated table. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style property.

See Also

Style (see page 16536)

Styles (see page 19830)

StylesFile (see page 19831)

WebFieldControls (see page 16884)

99.1.28 TNextButton Class

TNextButton generates an HTML control that moves to the next record in an XML data packet.

Class Hierarchy

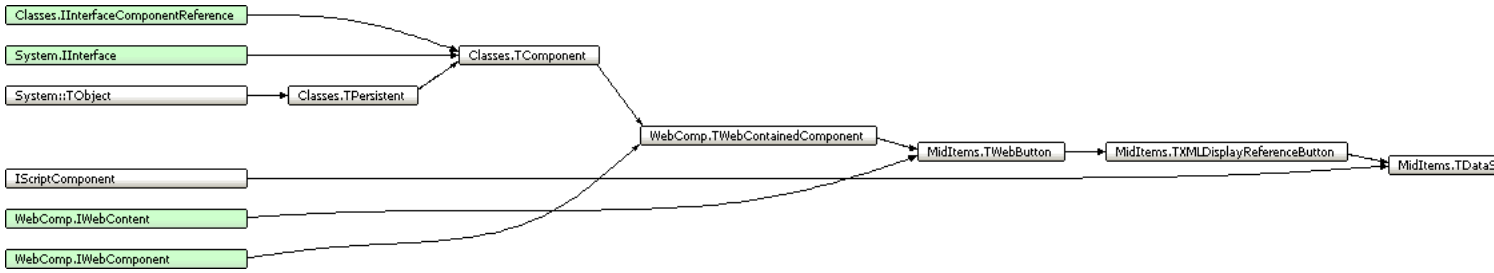

Pascal

```
TNextButton = class(TDataSetButton);
```

C++

```
class TNextButton : public TDataSetButton;
```

File

MidItems

Description

Add TNextButton as the child of a TDataNavigator (see page 16569) component (or a layout group that is a child of a TDataNavigator (see page 16569) component) to enable an associated control to move to the next record in a dataset. The resulting HTML button's action changes the current record displayed by the associated control.

Use the properties of TNextButton to customize the appearance of the button and associate it with a Web item. The associated Web item displays the records that the next button navigates.

See Also

TXMLBroker (see page 26049)

TDataNavigator (see page 16569)

TLayoutGroup (see page 16687)

99.1.28.1 TNextButton Members

The following tables list the members exposed by TNextButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 16581)	This is DefaultCaption, a member of class TDataSetButton.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	XMLMethodName (see page 16581)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16582)	This is AddElements, a member of class TDataSetButton.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16868)	Returns the HTML that defines the button control.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetCaption (see page 16582)	Returns the value of the Caption property.
V	GetCaption (see page 16869)	Returns the value of the Caption (see page 16870) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDisplayComponent (see page 17043)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 17043)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.

V	GetMethodName (see page 16582)	This is GetMethodName, a member of class TDataSetButton.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16582)	This is GetSubComponents, a member of class TDataSetButton.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 17043)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	ImplAddElements (see page 16583)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 16583)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 16869)	Provides the underlying implementation for the Content (see page 16868) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 17044)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayComponent (see page 17044)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 17044)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

R	XMLDisplay (see page 17045)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16699)	Creates and initializes an instance of TNextButton (see page 16694).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 17042)	Frees an instance of TXMLDisplayReferenceButton (see page 17037).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 16870)	Specifies the caption that appears on the button.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16871)	Represents a tag option and its value in the generated INPUT tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16871)	Adds style attributes to the generated button control.
	StyleRule (see page 16872)	Applies a style in the page producer's style sheet to the button.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	XMLComponent (see page 17045)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 17045)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 16584)	Specifies the caption that appears on the button.
	Custom (see page 16584)	Represents a tag option and its value in the generated INPUT tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16584)	Adds style attributes to the generated button control.
	StyleRule (see page 16585)	Applies a style in the page producer's style sheet to the button.

	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLComponent (see page 16585)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 16586)	Specifies whether the button is associated with its parent's display object.

99.1.28.2 TNextButton.Create Constructor

Creates and initializes an instance of TNextButton (see page 16694).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TNextButton(TComponent * AOwner);
```

Description

Call Create to instantiate TNextButton (see page 16694) at runtime. Buttons (see page 2289) added at design time using the Web Page editor are instantiated automatically.

99.1.29 TNextPageButton Class

TNextPageButton generates an HTML control that moves to the next set of records in an XML data packet.

Class Hierarchy

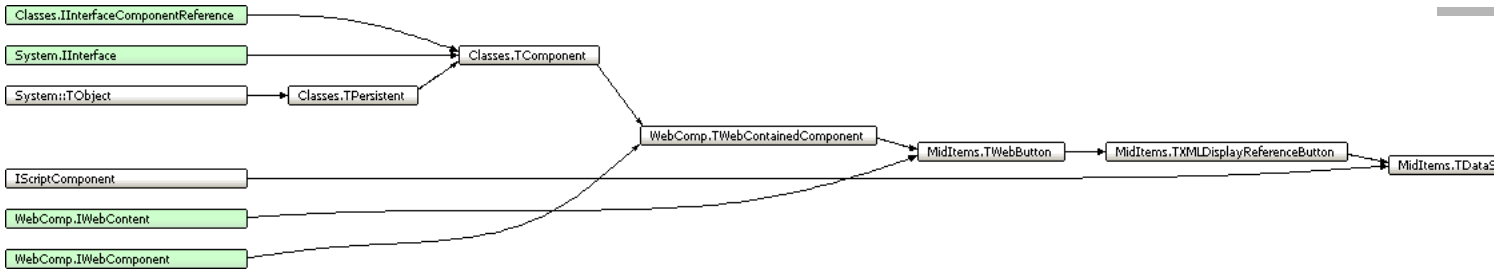

Pascal

```
TNextPageButton = class(TDataSetButton);
```

C++

```
class TNextPageButton : public TDataSetButton;
```

File

MidItems

Description

Add TNextPageButton as the child of a TDataNavigator (see page 16569) component (or a layout group that is a child of a TDataNavigator (see page 16569) component) to enable the generated HTML form to display the next group of records in a dataset. If the button is associated with a grid (TDataGrid (see page 16560)), the grid scrolls by the number of rows in the grid. If the button is associated with a collection of fields (TFieldGroup (see page 16591)), the current record changes to the next record in the data packet.

Use the properties of TNextPageButton to customize the appearance of the button and associate it with a display component.

See Also

TXMLBroker (see page 26049)

TDataNavigator (see page 16569)

TLayoutGroup (see page 16687)

TDataGrid (see page 16560)

TFieldGroup (see page 16591)

99.1.29.1 TNextPageButton Members

The following tables list the members exposed by TNextPageButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 16581)	This is DefaultCaption, a member of class TDataSetButton.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	XMLMethodName (see page 16581)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16582)	This is AddElements, a member of class TDataSetButton.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16868)	Returns the HTML that defines the button control.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetCaption (see page 16582)	Returns the value of the Caption property.
V	GetCaption (see page 16869)	Returns the value of the Caption (see page 16870) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDisplayComponent (see page 17043)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 17043)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
V	GetMethodName (see page 16582)	This is GetMethodName, a member of class TDataSetButton.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16582)	This is GetSubComponents, a member of class TDataSetButton.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 17043)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.

V	ImplAddElements (see page 16583)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 16583)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 16869)	Provides the underlying implementation for the Content (see page 16868) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 17044)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayComponent (see page 17044)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 17044)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.
R	XMLDisplay (see page 17045)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.

	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
	Create (see page 16703)	Creates and initializes an instance of TNextPageButton (see page 16699).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 17042)	Frees an instance of TXMLDisplayReferenceButton (see page 17037).
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 16870)	Specifies the caption that appears on the button.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16871)	Represents a tag option and its value in the generated INPUT tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16871)	Adds style attributes to the generated button control.
	StyleRule (see page 16872)	Applies a style in the page producer's style sheet to the button.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	XMLComponent (see page 17045)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 17045)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 16584)	Specifies the caption that appears on the button.
	Custom (see page 16584)	Represents a tag option and its value in the generated INPUT tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16584)	Adds style attributes to the generated button control.
	StyleRule (see page 16585)	Applies a style in the page producer's style sheet to the button.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLComponent (see page 16585)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 16586)	Specifies whether the button is associated with its parent's display object.

99.1.29.2 TNextPageButton.Create Constructor

Creates and initializes an instance of TNextPageButton (see page 16699).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TNextPageButton(TComponent * AOwner);
```

Description

Call Create to instantiate TNextPageButton (see page 16699) at runtime. Buttons (see page 2289) added at design time using the Web Page editor are instantiated automatically.

99.1.30 TPostButton Class

TPostButton generates an HTML control that posts changes to an XML delta packet.

Class Hierarchy

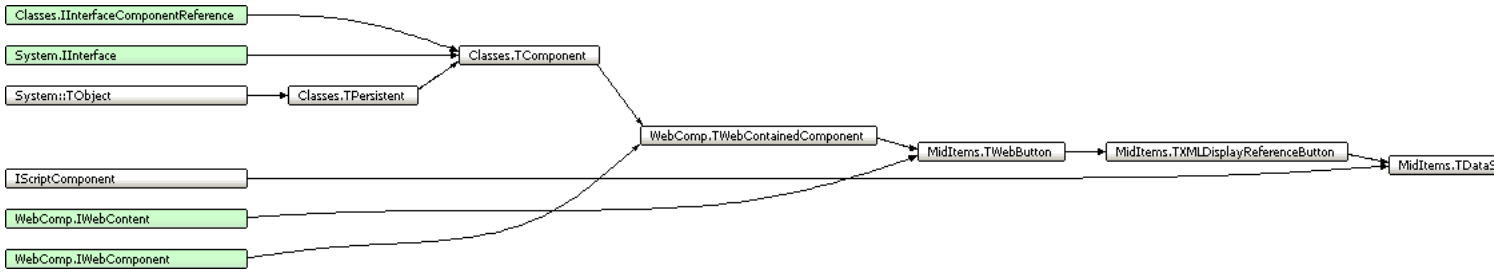

Pascal

```
TPostButton = class(TDataSetButton);
```

C++

```
class TPostButton : public TDataSetButton;
```

File

MidItems

Description

Add TPostButton as the child of a TDataNavigator (see page 16569) component (or a layout group that is a child of a TDataNavigator (see page 16569) component) to enable the generated HTML form to commit the edits on the current record of an associated display control. The resulting HTML button's action posts all edits to the current record to the emerging delta packet, which can then be submitted using an apply updates button.

Use the properties of TPostButton to customize the appearance of the button and associate it with a Web item that generates the display control.

See Also

TXMLBroker (see page 26049)

TDataNavigator (see page 16569)

TLayoutGroup (see page 16687)

TApplyUpdatesButton (see page 16480)

99.1.30.1 TPostButton Members

The following tables list the members exposed by TPostButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 16581)	This is DefaultCaption, a member of class TDataSetButton.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	XMLMethodName (see page 16581)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16582)	This is AddElements, a member of class TDataSetButton.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16868)	Returns the HTML that defines the button control.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetCaption (see page 16582)	Returns the value of the Caption property.
V	GetCaption (see page 16869)	Returns the value of the Caption (see page 16870) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDisplayComponent (see page 17043)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 17043)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
V	GetMethodName (see page 16582)	This is GetMethodName, a member of class TDataSetButton.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16582)	This is GetSubComponents, a member of class TDataSetButton.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 17043)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	ImplAddElements (see page 16583)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 16583)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 16869)	Provides the underlying implementation for the Content (see page 16868) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 17044)	Forwards notification messages to all owned components.

	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayComponent (see page 17044)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 17044)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.
R	XMLDisplay (see page 17045)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16708)	Creates and initializes an instance of TPostButton (see page 16704).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 17042)	Frees an instance of TXMLDisplayReferenceButton (see page 17037).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.

	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 16870)	Specifies the caption that appears on the button.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16871)	Represents a tag option and its value in the generated INPUT tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16871)	Adds style attributes to the generated button control.
	StyleRule (see page 16872)	Applies a style in the page producer's style sheet to the button.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	XMLComponent (see page 17045)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 17045)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 16584)	Specifies the caption that appears on the button.
	Custom (see page 16584)	Represents a tag option and its value in the generated INPUT tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16584)	Adds style attributes to the generated button control.
	StyleRule (see page 16585)	Applies a style in the page producer's style sheet to the button.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLComponent (see page 16585)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 16586)	Specifies whether the button is associated with its parent's display object.

99.1.30.2 TPostButton.Create Constructor

Creates and initializes an instance of TPostButton (see page 16704).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TPostButton(TComponent * AOwner);
```

Description

Call Create to instantiate TPostButton (see page 16704) at runtime. Buttons (see page 2289) added at design time using the Web Page editor are instantiated automatically.

99.1.31 TPriorButton Class

TPriorButton generates an HTML control that moves to the previous record in an XML data packet.

Class Hierarchy

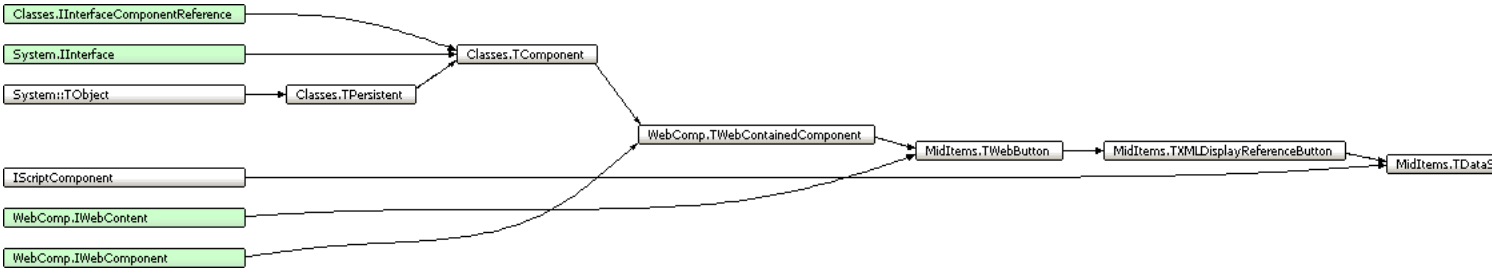

Pascal

```
TPriorButton = class(TDataSetButton);
```

C++

```
class TPriorButton : public TDataSetButton;
```

File

MidItems

Description

Add TPriorButton as the child of a TDataNavigator (see page 16569) component (or a layout group that is a child of a TDataNavigator (see page 16569) component) to enable an associated control to move to the previous record in a dataset. The resulting HTML button's action changes the current record displayed by the associated control.

Use the properties of TPriorButton to customize the appearance of the button and associate it with a Web item. The associated Web item displays the records that the prior button navigates.

See Also

TXMLBroker (see page 26049)

TDataNavigator (see page 16569)

TLayoutGroup (see page 16687)

99.1.31.1 TPriorButton Members

The following tables list the members exposed by TPriorButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 16581)	This is DefaultCaption, a member of class TDataSetButton.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	XMLMethodName (see page 16581)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16582)	This is AddElements, a member of class TDataSetButton.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16868)	Returns the HTML that defines the button control.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetCaption (see page 16582)	Returns the value of the Caption property.
V	GetCaption (see page 16869)	Returns the value of the Caption (see page 16870) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDisplayComponent (see page 17043)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 17043)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
V	GetMethodName (see page 16582)	This is GetMethodName, a member of class TDataSetButton.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16582)	This is GetSubComponents, a member of class TDataSetButton.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 17043)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	ImplAddElements (see page 16583)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 16583)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 16869)	Provides the underlying implementation for the Content (see page 16868) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 17044)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDisplayComponent (see page 17044)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 17044)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.
R	XMLDisplay (see page 17045)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16713)	Creates and initializes an instance of TPriorButton (see page 16709).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 17042)	Frees an instance of TXMLDisplayReferenceButton (see page 17037).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.

V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 16870)	Specifies the caption that appears on the button.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.

W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16871)	Represents a tag option and its value in the generated INPUT tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16871)	Adds style attributes to the generated button control.
	StyleRule (see page 16872)	Applies a style in the page producer's style sheet to the button.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	XMLComponent (see page 17045)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 17045)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 16584)	Specifies the caption that appears on the button.
	Custom (see page 16584)	Represents a tag option and its value in the generated INPUT tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16584)	Adds style attributes to the generated button control.
	StyleRule (see page 16585)	Applies a style in the page producer's style sheet to the button.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLComponent (see page 16585)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 16586)	Specifies whether the button is associated with its parent's display object.

99.1.31.2 TPriorButton.Create Constructor

Creates and initializes an instance of TPriorButton (see page 16709).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TPriorButton(TComponent * AOwner);
```

Description

Call Create to instantiate TPriorButton (see page 16709) at runtime. Buttons (see page 2289) added at design time using the Web Page editor are instantiated automatically.

99.1.32 TPriorPageButton Class

TPriorPageButton generates an HTML control that moves to the previous set of records in an XML data packet.

Class Hierarchy

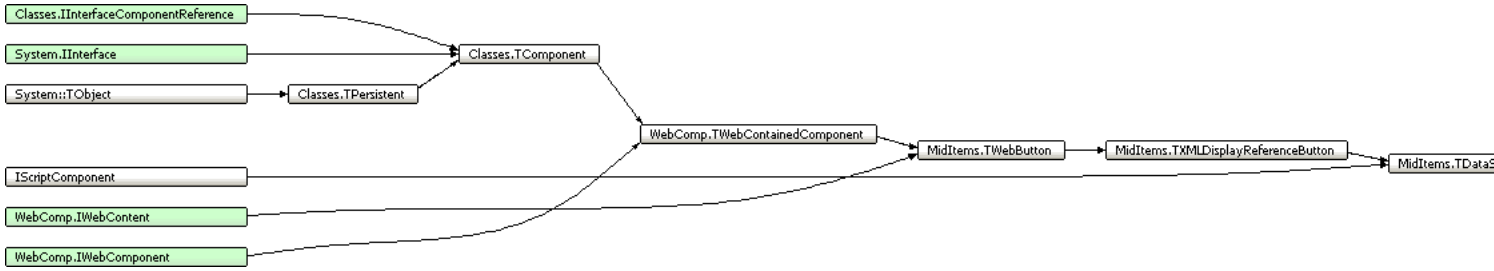

Pascal

```
TPriorPageButton = class(TDataSetButton);
```

C++

```
class TPriorPageButton : public TDataSetButton;
```

File

MidItems

Description

Add TPriorPageButton as the child of a TDataNavigator (see page 16569) component (or a layout group that is a child of a TDataNavigator (see page 16569) component) to enable the generated HTML form to display the previous group of records in a dataset. If the button is associated with a grid (TDataGrid (see page 16560)), the grid scrolls by the number of rows in the grid. If the button is associated with a collection of fields (TFieldGroup (see page 16591)), the current record changes to the previous record in the data packet.

Use the properties of TNextPageButton (see page 16699) to customize the appearance of the button and associate it with a display component.

See Also

TXMLBroker (see page 26049)

TDataNavigator (see page 16569)

TLayoutGroup (see page 16687)

TDataGrid (see page 16560)

TFieldGroup (see page 16591)

99.1.32.1 TPriorPageButton Members

The following tables list the members exposed by TPriorPageButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 16581)	This is DefaultCaption, a member of class TDataSetButton.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	XMLMethodName (see page 16581)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.

	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16582)	This is AddElements, a member of class TDataSetButton.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16868)	Returns the HTML that defines the button control.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetCaption (see page 16582)	Returns the value of the Caption property.
V	GetCaption (see page 16869)	Returns the value of the Caption (see page 16870) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDisplayComponent (see page 17043)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 17043)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
V	GetMethodName (see page 16582)	This is GetMethodName, a member of class TDataSetButton.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16582)	This is GetSubComponents, a member of class TDataSetButton.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 17043)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	ImplAddElements (see page 16583)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 16583)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 16869)	Provides the underlying implementation for the Content (see page 16868) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 17044)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayComponent (see page 17044)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 17044)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property

V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.
R	XMLDisplay (see page 17045)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16718)	Creates and initializes an instance of TPriorPageButton (see page 16713).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 17042)	Frees an instance of TXMLDisplayReferenceButton (see page 17037).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 16870)	Specifies the caption that appears on the button.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.

	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16871)	Represents a tag option and its value in the generated INPUT tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16871)	Adds style attributes to the generated button control.
	StyleRule (see page 16872)	Applies a style in the page producer's style sheet to the button.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	XMLComponent (see page 17045)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 17045)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 16584)	Specifies the caption that appears on the button.
	Custom (see page 16584)	Represents a tag option and its value in the generated INPUT tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16584)	Adds style attributes to the generated button control.
	StyleRule (see page 16585)	Applies a style in the page producer's style sheet to the button.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLComponent (see page 16585)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 16586)	Specifies whether the button is associated with its parent's display object.

99.1.32.2 TPriorPageButton.Create Constructor

Creates and initializes an instance of TPriorPageButton (see page 16713).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TPriorPageButton(TComponent * AOwner);
```

Description

Call Create to instantiate TPriorPageButton (see page 16713) at runtime. Buttons (see page 2289) added at design time using the Web Page editor are instantiated automatically.

99.1.33 TQueryButton Class

TQueryButton is the base class for Web items that create HTML buttons that manage application-defined values.

Class Hierarchy

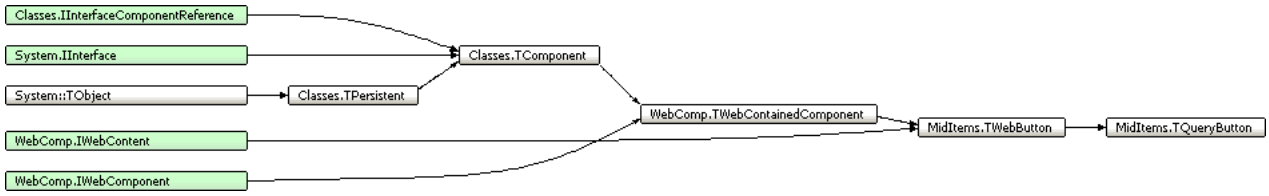

Pascal

```
TQueryButton = class(TWebButton);
```

C++

```
class TQueryButton : public TWebButton;
```

File

MidItems

Description

Use TQueryButton as a base class in an InternetExpress application for components that generate the HTML statements defining a specialized button control for managing the application-defined input controls. TQueryButton objects are used by TQueryForm (see page 16745), which organizes the generated buttons and the input controls they manipulate.

TQueryButton defines properties and methods for

Generating the HTML code to define the button control.

Specifying tag attributes of the HTML button control, including those defined by a style sheet.

Associating the button with a Query form whose controls it influences.

Fitting the component into a hierarchy of Web items, each of which generates part of a larger HTML document.

In order to allow other Web items to use the HTML generated by the button control, TQueryButton inherits support for the IWebContent interface from TWebButton (see page 16863).

Do not create instances of TQueryButton. TQueryButton relies on anabstract or, in C++ terminology, pure virtual method to generate the HTML statements defining a button control. Instead, create descendants of TQueryButton, such as TResetQueryButton (see page 16809) or TSubmitQueryButton (see page 16833).

See Also

TResetQueryButton (see page 16809)

TSubmitQueryButton (see page 16833)

IWebContent (see page 24792)

TQueryForm (see page 16745)

99.1.33.1 TQueryButton Members

The following tables list the members exposed by TQueryButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 16723)	This is DefaultCaption, a member of class TQueryButton.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	InputType (see page 16723)	This is InputType, a member of class TQueryButton.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.

	Content (see page 16868)	Returns the HTML that defines the button control.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16724)	This is EventContent, a member of class TQueryButton.
V	GetCaption (see page 16724)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetHTMLControlName (see page 16725)	This is GetHTMLControlName, a member of class TQueryButton.
	GetHTMLForm (see page 16725)	This is GetHTMLForm, a member of class TQueryButton.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetInputType (see page 16725)	This is GetInputType, a member of class TQueryButton.
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXMLComponent (see page 16725)	This is GetXMLComponent, a member of class TQueryButton.
V	ImplContent (see page 16725)	Provides the underlying implementation for the Content method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 16726)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	ParentXMLComponent (see page 16726)	This is ParentXMLComponent, a member of class TQueryButton.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	SetXMLComponent (see page 16727)	This is SetXMLComponent, a member of class TQueryButton.
	SetXMLUseParent (see page 16727)	This is SetXMLUseParent, a member of class TQueryButton.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16723)	Creates and initializes an instance of TQueryButton (see page 16718).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Caption (see page 16727)	Specifies the caption that appears on the button.
	Custom (see page 16728)	Represents a tag option and its value in the generated INPUT tag.

	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16728)	Adds style attributes to the generated button control.
	StyleRule (see page 16729)	Applies a style in the page producer's style sheet to the button.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLComponent (see page 16729)	Associates the button with the query form whose display controls it influences.
	XMLUseParent (see page 16729)	Specifies whether the button is associated with its parent's query form.

99.1.33.2 TQueryButton.Create Constructor

Creates and initializes an instance of TQueryButton (see page 16718).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TQueryButton(TComponent * AOwner);
```

Description

Call Create to instantiate a dataset button at runtime. Buttons (see page 2289) added at design time using the Web Page editor are instantiated automatically.

AOwner is the component that is responsible for freeing the query button. Typically, this is the Web module that contains the InterenetExpress page producer that uses the query button's generated HTML.

99.1.33.3 TQueryButton Fields

The fields of the TQueryButton class are listed here.

Protected Fields

	Name	Description
	DefaultCaption (see page 16723)	This is DefaultCaption, a member of class TQueryButton.
	InputType (see page 16723)	This is InputType, a member of class TQueryButton.

99.1.33.3.1 TQueryButton.DefaultCaption Field

Pascal

```
DefaultCaption: string;
```

C++

```
AnsiString DefaultCaption;
```

Description

This is DefaultCaption, a member of class TQueryButton.

99.1.33.3.2 TQueryButton.InputType Field

Pascal

```
InputType: string;
```

C++

```
AnsiString InputType;
```

Description

This is InputType, a member of class TQueryButton.

99.1.33.4 TQueryButton Methods

The methods of the TQueryButton class are listed here.

Protected Methods

	Name	Description
	EventContent (see page 16724)	This is EventContent, a member of class TQueryButton.
	GetCaption (see page 16724)	Returns the value of the Caption property.
	GetHTMLControlName (see page 16725)	This is GetHTMLControlName, a member of class TQueryButton.
	GetHTMLForm (see page 16725)	This is GetHTMLForm, a member of class TQueryButton.
	GetInputType (see page 16725)	This is GetInputType, a member of class TQueryButton.
	GetXMLComponent (see page 16725)	This is GetXMLComponent, a member of class TQueryButton.
	ImplContent (see page 16725)	Provides the underlying implementation for the Content method.
	Notification (see page 16726)	Forwards notification messages to all owned components.
	ParentXMLComponent (see page 16726)	This is ParentXMLComponent, a member of class TQueryButton.
	SetXMLComponent (see page 16727)	This is SetXMLComponent, a member of class TQueryButton.
	SetXMLUseParent (see page 16727)	This is SetXMLUseParent, a member of class TQueryButton.

99.1.33.4.1 TQueryButton.EventContent Method

Pascal

```
function EventContent(Options: TWebContentOptions): string; virtual;
```

C++

```
virtual __fastcall AnsiString EventContent(TWebContentOptions Options);
```

Description

This is EventContent, a member of class TQueryButton.

99.1.33.4.2 TQueryButton.GetCaption Method

Returns the value of the Caption property.

Pascal

```
function GetCaption: string; override;
```

C++

```
virtual __fastcall AnsiString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption property.

See Also

Caption (see page 16870)

SetCaption (see page 16870)

99.1.33.4.3 TQueryButton.GetHTMLControlName Method

Pascal

```
function GetHTMLControlName: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetHTMLControlName();
```

Description

This is GetHTMLControlName, a member of class TQueryButton.

99.1.33.4.4 TQueryButton.GetHTMLForm Method

Pascal

```
function GetHTMLForm: IHTMLForm;
```

C++

```
__fastcall IHTMLForm GetHTMLForm();
```

Description

This is GetHTMLForm, a member of class TQueryButton.

99.1.33.4.5 TQueryButton.GetInputType Method

Pascal

```
function GetInputType: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetInputType();
```

Description

This is GetInputType, a member of class TQueryButton.

99.1.33.4.6 TQueryButton.GetXMLComponent Method

Pascal

```
function GetXMLComponent: TComponent;
```

C++

```
__fastcall TComponent * GetXMLComponent();
```

Description

This is GetXMLComponent, a member of class TQueryButton.

99.1.33.4.7 TQueryButton.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML that defines the button control. TWebButton introduces ImplContent as an abstract or, in C++ terminology, pure virtual method. TWebButton descendants must override ImplContent to generate the definition of their particular button control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web button should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the button control.

See Also

Content (see page 16868)

ILayoutWebContent (see page 24774)

99.1.33.4.8 TQueryButton.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

99.1.33.4.9 TQueryButton.ParentXMLComponent Method

Pascal

```
function ParentXMLComponent: TComponent;
```

C++

```
__fastcall TComponent * ParentXMLComponent();
```

Description

This is ParentXMLComponent, a member of class TQueryButton.

99.1.33.4.10 TQueryButton.SetXMLComponent Method

Pascal

```
procedure SetXMLComponent(const Value: TComponent);
```

C++

```
__fastcall SetXMLComponent(const TComponent * Value);
```

Description

This is SetXMLComponent, a member of class TQueryButton.

99.1.33.4.11 TQueryButton.SetXMLUseParent Method

Pascal

```
procedure SetXMLUseParent(const Value: Boolean);
```

C++

```
__fastcall SetXMLUseParent(const Boolean Value);
```

Description

This is SetXMLUseParent, a member of class TQueryButton.

99.1.33.5 TQueryButton Properties

The properties of the TQueryButton class are listed here.

Published Properties

	Name	Description
	Caption (see page 16727)	Specifies the caption that appears on the button.
	Custom (see page 16728)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 16728)	Adds style attributes to the generated button control.
	StyleRule (see page 16729)	Applies a style in the page producer's style sheet to the button.
	XMLComponent (see page 16729)	Associates the button with the query form whose display controls it influences.
	XMLUseParent (see page 16729)	Specifies whether the button is associated with its parent's query form.

99.1.33.5.1 TQueryButton.Caption Property

Specifies the caption that appears on the button.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the caption that appears on the generated HTML button. The value of Caption is used as the value of a VALUE parameter on the HTML INPUT tag generated to implement the button.

See Also

Custom (see page 16871)

99.1.33.5.2 TQueryButton.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to represent one or more options that qualify the INPUT tag generated by the Web button. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

```
ALIGN=TOP TABINDEX=2
```

See Also

Caption (see page 16870)

Style (see page 16871)

99.1.33.5.3 TQueryButton.Style Property

Adds style attributes to the generated button control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to add a STYLE option that qualifies the INPUT tag generated by the Web button. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated INPUT tag as the value of a STYLE option.

See Also

Custom (see page 16871)

StyleRule (see page 16872)

99.1.33.5.4 TQueryButton.StyleRule Property

Applies a style in the page producer's style sheet to the button.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated button control. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the button's Style property.

See Also

Style (see page 16871)

Styles (see page 17127)

StylesFile (see page 17128)

Styles (see page 19830)

StylesFile (see page 19831)

99.1.33.5.5 TQueryButton.XMLComponent Property

Associates the button with the query form whose display controls it influences.

Pascal

```
property XMLComponent: TComponent;
```

C++

```
__property TComponent * XMLComponent;
```

Description

Set XMLComponent to specify the Web item that generates the form whose controls the query button influences. If this query form contains the TQueryButtons (see page 16730) object that formats the query button with its siblings, you need not set the XMLComponent property. Instead, setting the XMLUseParent (see page 16729) property to true (the default) sets XMLComponent to the Query form that contains the parent TQueryButtons (see page 16730) object.

See Also

XMLUseParent (see page 16729)

TQueryForm (see page 16745)

99.1.33.5.6 TQueryButton.XMLUseParent Property

Specifies whether the button is associated with its parent's query form.

Pascal

```
property XMLUseParent: Boolean;
```

C++

```
__property Boolean XMLUseParent;
```

Description

Use XMLUseParent to indicate how the query button determines the form whose controls it influences. When XMLUseParent is false, the XMLComponent (see page 16729) property must be set to indicate the query form. When XMLUseParent is true, the button influences controls on the query form that contains its parent TQueryButtons (see page 16730) component.

See Also

XMLComponent (see page 16729)

TQueryForm (see page 16745)

TQueryButtons (see page 16730)

99.1.34 TQueryButtons Class

TQueryButtons assembles the HTML generated by a set of TQueryButton (see page 16718) descendants.

Class Hierarchy

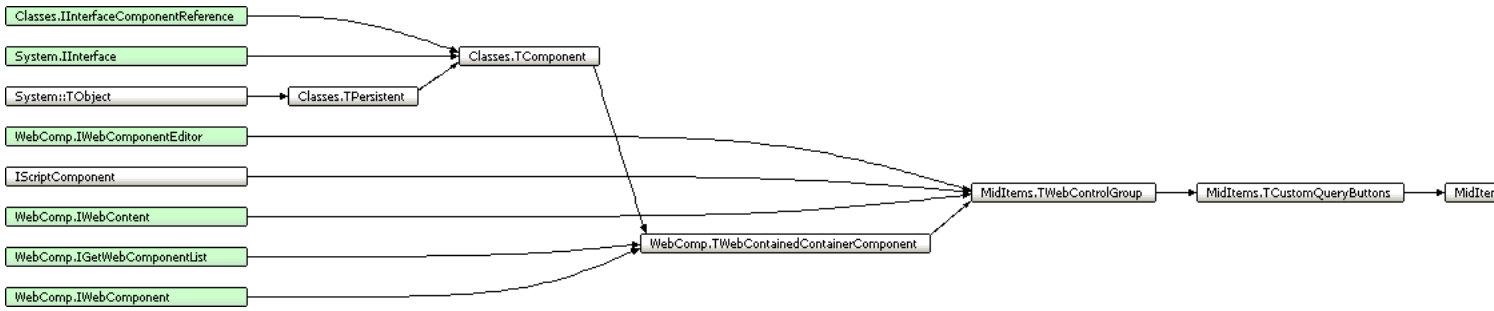

Pascal

```
TQueryButtons = class(TCustomQueryButtons);
```

C++

```
class TQueryButtons : public TCustomQueryButtons;
```

File

MidItems

Description

Add TQueryButtons as the child of a TQueryForm (see page 16745) component (or a layout group that is a child of a TQueryForm (see page 16745) component) to enable the generated HTML form to undo or submit changes made by end users. TQueryButtons generates HTML containing the controls generated by its children. These children must be descendants of TQueryButton (see page 16718), such as TSubmitQueryButton (see page 16833) and TResetQueryButton (see page 16809).

In order to let other Web items to use the HTML that defines the organized collection of controls, TQueryButtons inherits the IWebContent interface from TWebControlGroup (see page 16872).

Use the properties of TQueryButtons to customize the appearance of the group of buttons.

See Also

TQueryForm (see page 16745)

TResetQueryButton (see page 16809)

TSubmitQueryButton (see page 16833)

IWebContent (see page 24792)

TQueryFieldGroup (see page 16736)

Using the Web Page Editor

99.1.34.1 TQueryButtons Members

The following tables list the members exposed by TQueryButtons.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultButtons (see page 16542)	This is GetDefaultButtons, a member of class TCustomQueryButtons.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
V	ImplAddElements (see page 16542)	Provides the underlying implementation for the AddElements method.
V	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
V	ImplCanAddClass (see page 16543)	Provides the underlying implementation for the CanAddClass method.
V	ImplCanAddClass (see page 16881)	Provides the underlying implementation for the CanAddClass (see page 16878) method.
V	ImplContent (see page 16543)	Provides the underlying implementation for the Content method.
V	ImplContent (see page 16882)	Provides the underlying implementation for the Content (see page 16879) method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleButtons (see page 16543)	This is VisibleButtons, a member of class TCustomQueryButtons.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16877)	Creates and initializes an instance of TWebControlGroup (see page 16872).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24742)	Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16877)	Frees an instance of TWebControlGroup (see page 16872).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.

	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Custom (see page 16735)	Represents a tag option and its value in the generated TABLE tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16735)	Adds style attributes to the generated HTML table.
	StyleRule (see page 16736)	Applies a style in the page producer's style sheet to the group of buttons.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.34.2 TQueryButtons Properties

The properties of the TQueryButtons class are listed here.

Published Properties

	Name	Description
	Custom (see page 16735)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 16735)	Adds style attributes to the generated HTML table.
	StyleRule (see page 16736)	Applies a style in the page producer's style sheet to the group of buttons.

99.1.34.2.1 TQueryButtons.Custom Property

Represents a tag option and its value in the generated TABLE tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TCustomQueryButtons generates an HTML table to lay out the buttons listed in the WebFieldControls property. Use Custom to add options that qualify the TABLE tag. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add a border around the query buttons and color the background blue, you can set Custom to

```
BORDER=2 BGCOLOR=BLUE
```

See Also

Style (see page 16544)

StyleRule (see page 16545)

WebFieldControls (see page 16884)

99.1.34.2.2 TQueryButtons.Style Property

Adds style attributes to the generated HTML table.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

TCustomQueryButtons generates an HTML table to lay out the buttons listed in the WebFieldControls property. Use Style to add a STYLE option that qualifies the TABLE tag. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated TABLE tag as the value of a STYLE option.

See Also

Custom (see page 16544)

StyleRule (see page 16545)

WebFieldControls (see page 16884)

99.1.34.2.3 TQueryButtons.StyleRule Property

Applies a style in the page producer's style sheet to the group of buttons.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

TCustomQueryButtons generates an HTML table to lay out the buttons listed in the WebFieldControls property. Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated table. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style property.

See Also

Style (see page 16544)

Styles (see page 17127)

StylesFile (see page 17128)

WebFieldControls (see page 16884)

99.1.35 TQueryFieldGroup Class

TQueryFieldGroup arranges information on an HTML form as a set of labeled fields.

Class Hierarchy

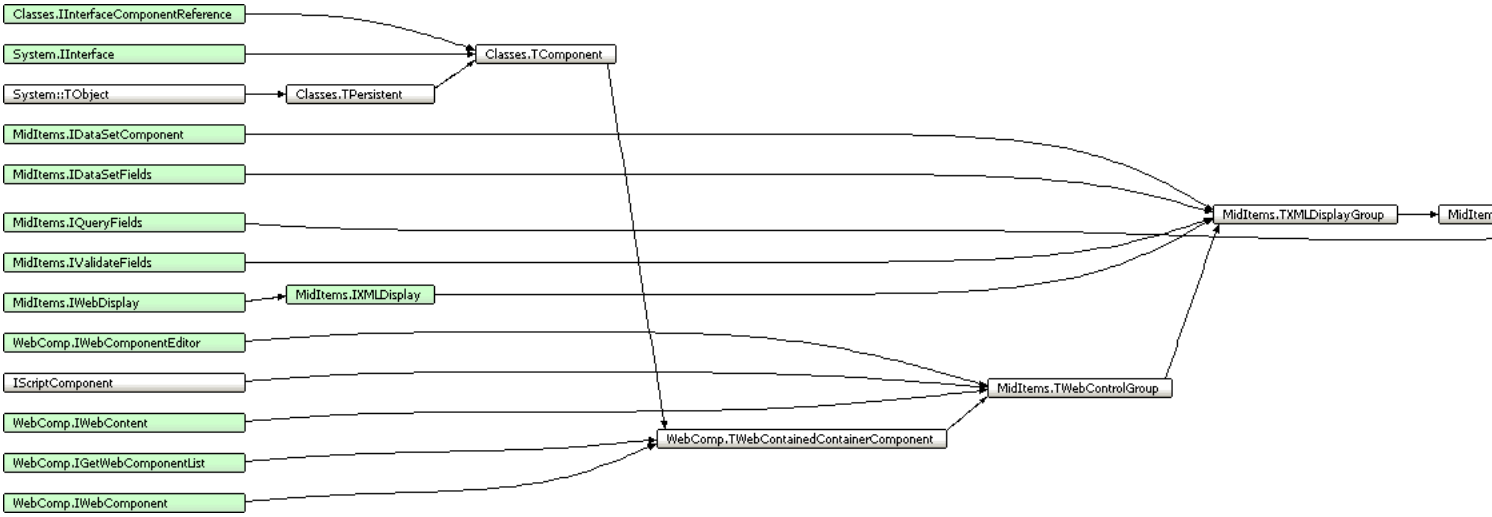

Pascal

```
TQueryFieldGroup = class(TCustomFieldGroup, IQueryFields);
```

C++

```
class TQueryFieldGroup : public TCustomFieldGroup, public IQueryFields;
```

File

MidItems

Description

Add TQueryFieldGroup to a query form component (TQueryForm (see page 16745)) or a layout group (TLayoutGroup (see page 16687)) that is a child for a query form. TQueryFieldGroup generates the HTML to display information in labeled fields. By default, TQueryFieldGroup displays information about the parameters for an XML broker, but it can be used to display any application-defined named values.

TQueryFieldGroup maintains a list of Web items that generate the individual display fields. These Web items are restricted to the following classes: TQueryText (see page 16783), TQueryTextArea (see page 16795), TQuerySelectOptions (see page 16768), and TQueryRadioGroup (see page 16753). If the query field group displays parameter values, you can bind the query field group to an XML data packet or dataset field within an XML data packet. The Web items can then use that binding to determine the source of the parameters they display.

You can also use the properties of TQueryFieldGroup to affect the overall appearance of the set of fields.

See Also

TQueryForm (see page 16745)

TLayoutGroup (see page 16687)

TQueryRadioGroup (see page 16753)

TQuerySelectOptions (see page 16768)

TQueryText (see page 16783)

TQueryTextArea (see page 16795)

TXMLBroker (see page 26049)

TWebDataDisplay (see page 16884)

Using the Web Page Editor

99.1.35.1 TQueryFieldGroup Members

The following tables list the members exposed by TQueryFieldGroup.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
V	CreateDefaultFields (see page 17025)	This is CreateDefaultFields, a member of class TXMLDisplayGroup.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FindField (see page 17026)	This is FindField, a member of class TXMLDisplayGroup.
	FindParam (see page 16743)	This is FindParam, a member of class TQueryFieldGroup.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 17026)	Returns the value of the DataSet property.
	GetDataSetField (see page 17026)	This is GetDataSetField, a member of class TXMLDisplayGroup.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetEnableValidateFields (see page 17026)	This is GetEnableValidateFields, a member of class TXMLDisplayGroup.
V	GetFieldsList (see page 17027)	This is GetFieldsList, a member of class TXMLDisplayGroup.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsMultipleRecordView (see page 17027)	This is GetIsMultipleRecordView, a member of class TXMLDisplayGroup.
V	GetLayoutAttributes (see page 16526)	Returns an object that specifies how the generated group of controls should look.
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetParamsList (see page 16743)	This is GetParamsList, a member of class TQueryFieldGroup.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVisibleFields (see page 17027)	This is GetVisibleFields, a member of class TXMLDisplayGroup.
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	GetXMLBroker (see page 17027)	Returns the value of the XMLBroker property.
	GetXMLDataSet (see page 17028)	Returns the value of the XMLDataSet property.
	GetXMLDisplayName (see page 17028)	This is GetXMLDisplayName, a member of class TXMLDisplayGroup.
	GetXMLRowSetName (see page 17028)	This is GetXMLRowSetName, a member of class TXMLDisplayGroup.
	HasStatusField (see page 17028)	This is HasStatusField, a member of class TXMLDisplayGroup.
V	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
V	ImplAddElements (see page 17029)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 16743)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 16526)	Provides the underlying implementation for the Content method.
V	ImplContent (see page 16882)	Provides the underlying implementation for the Content (see page 16879) method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
V	ImplGetSubComponents (see page 17029)	Provides the underlying implementation for the GetSubComponents method.
V	ImplIsMultipleRecordView (see page 17029)	This is ImplIsMultipleRecordView, a member of class TXMLDisplayGroup.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsFieldInUse (see page 17030)	This is IsFieldInUse, a member of class TXMLDisplayGroup.
	IsParamInUse (see page 16744)	This is IsParamInUse, a member of class TQueryFieldGroup.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 17030)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
	SetDataSetField (see page 17030)	This is SetDataSetField, a member of class TXMLDisplayGroup.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEnableValidateFields (see page 17031)	This is SetEnableValidateFields, a member of class TXMLDisplayGroup.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	SetXMLBroker (see page 17031)	This is SetXMLBroker, a member of class TXMLDisplayGroup.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateDataSet (see page 17031)	This is ValidateDataSet, a member of class TXMLDisplayGroup.
	ValidateFields (see page 17031)	This is ValidateFields, a member of class TXMLDisplayGroup.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.
V	XMLDataChange (see page 17031)	This is XMLDataChange, a member of class TXMLDisplayGroup.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
	EnableValidateFields (see page 17032)	This is EnableValidateFields, a member of class TXMLDisplayGroup.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.
R	XMLDataSet (see page 17033)	Indicates the TXMLDataSet object that represents the dataset to which the Web item is bound.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16525)	Creates and initializes an instance of TCustomFieldGroup (see page 16519).
V	Create (see page 16877)	Creates and initializes an instance of TWebControlGroup (see page 16872).
V	Create (see page 17024)	Creates and initializes an instance of TXMLDisplayGroup (see page 17019).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24742)	Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16525)	Frees an instance of TCustomFieldGroup (see page 16519).
V	Destroy (see page 16877)	Frees an instance of TWebControlGroup (see page 16872).

V	Destroy (see page 17024)	Frees an instance of TXMLDisplayGroup (see page 17019).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDataSet (see page 17054)	Returns the value of the DataSet (see page 17055) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnableValidateFields (see page 17078)	This is GetEnableValidateFields, a member of class IValidateFields.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 17060)	This is GetFieldsList, a member of class IDataSetFields.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsMultipleRecordView (see page 17079)	This is GetIsMultipleRecordView, a member of class IWebDisplay.
	GetIsMultipleRecordView (see page 17082)	This is GetIsMultipleRecordView, a member of class IXMLDisplay.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	GetVisibleFields (see page 17060)	This is GetVisibleFields, a member of class IDataSetFields.
	GetXMLBroker (see page 17054)	Returns the value of the XMLBroker (see page 17056) property.
	GetXMLDataSet (see page 17055)	Returns the value of the XMLDataSet (see page 17056) property.
	GetXMLDisplayName (see page 17083)	This is GetXMLDisplayName, a member of class IXMLDisplay.
	GetXMLRowSetName (see page 17083)	This is GetXMLRowSetName, a member of class IXMLDisplay.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	HasStatusField (see page 17060)	This is HasStatusField, a member of class IDataSetFields.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsFieldInUse (see page 17060)	This is IsFieldInUse, a member of class IDataSetFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEnableValidateFields (see page 17078)	This is SetEnableValidateFields, a member of class IValidateFields.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateDataSet (see page 17060)	This is ValidateDataSet, a member of class IDataSetFields.
	ValidateDataSet (see page 17074)	This is ValidateDataSet, a member of class IQueryFields.
	ValidateFields (see page 17078)	This is ValidateFields, a member of class IValidateFields.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 17055)	Provides access to an interface for working with the associated dataset.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EnableValidateFields (see page 17078)	This is EnableValidateFields, a member of class IValidateFields.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsMultipleRecordView (see page 17080)	
R	IsMultipleRecordView (see page 17083)	
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleFields (see page 17061)	This is VisibleFields, a member of class IDataSetFields.
R	VisibleFields (see page 17032)	Indicates the controls displayed by the HTML that the XML display group generates.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.
R	XMLBroker (see page 17056)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
R	XMLDataSet (see page 17056)	Indicates the TXMLDataSet (see page 17006) object that represents the dataset to which the Web item is bound.
R	XMLDisplayName (see page 17084)	This is XMLDisplayName, a member of class IXMLDisplay.
R	XMLRowSetName (see page 17084)	This is XMLRowSetName, a member of class IXMLDisplay.

Published Properties

	Name	Description
	Custom (see page 16744)	Represents a tag option and its value in the generated TABLE tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16745)	Adds style attributes to the generated set of fields.
	StyleRule (see page 16745)	Applies a style in the page producer's style sheet to the table.

	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLBroker (see page 17033)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
	XMLDataSetField (see page 17033)	Binds the XML display group to a nested dataset from the XML data packet.

99.1.35.2 TQueryFieldGroup Methods

The methods of the TQueryFieldGroup class are listed here.

Protected Methods

	Name	Description
	FindParam (see page 16743)	This is FindParam, a member of class TQueryFieldGroup.
	GetParamsList (see page 16743)	This is GetParamsList, a member of class TQueryFieldGroup.
V	ImplCanAddClass (see page 16743)	Provides the underlying implementation for the CanAddClass method.
	IsParamInUse (see page 16744)	This is IsParamInUse, a member of class TQueryFieldGroup.

99.1.35.2.1 TQueryFieldGroup.FindParam Method

Pascal

```
function FindParam(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindParam(const AnsiString AName);
```

Description

This is FindParam, a member of class TQueryFieldGroup.

99.1.35.2.2 TQueryFieldGroup.GetParamsList Method

Pascal

```
procedure GetParamsList(List: TStrings);
```

C++

```
__fastcall GetParamsList(TStrings List);
```

Description

This is GetParamsList, a member of class TQueryFieldGroup.

99.1.35.2.3 TQueryFieldGroup.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass (see page 16878)

WebFieldControls (see page 16884)

99.1.35.2.4 TQueryFieldGroup.IsParamInUse Method

Pascal

```
function IsParamInUse(AName: string): Boolean;
```

C++

```
__fastcall Boolean IsParamInUse(AnsiString AName);
```

Description

This is IsParamInUse, a member of class TQueryFieldGroup.

99.1.35.3 TQueryFieldGroup Properties

The properties of the TQueryFieldGroup class are listed here.

Published Properties

	Name	Description
	Custom (see page 16744)	Represents a tag option and its value in the generated TABLE tag.
	Style (see page 16745)	Adds style attributes to the generated set of fields.
	StyleRule (see page 16745)	Applies a style in the page producer's style sheet to the table.

99.1.35.3.1 TQueryFieldGroup.Custom Property

Represents a tag option and its value in the generated TABLE tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TCustomFieldGroup generates an HTML table to manage the layout of the HTML controls that its Web items generate. Use Custom to add options that qualify the TABLE tag. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to surround the set of fields with a border and give them a blue background, set Custom to

```
BORDER=2 BGCOLOR=BLUE
```

See Also

Style (see page 16527)

StyleRule (see page 16528)

99.1.35.3.2 TQueryFieldGroup.Style Property

Adds style attributes to the generated set of fields.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

TCustomFieldGroup generates an HTML table to lay out the fields that its Web items generate. Use Style to add a STYLE option that qualifies the TABLE tag. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated TABLE tag as the value of a STYLE option.

See Also

Custom (see page 16527)

StyleRule (see page 16528)

99.1.35.3.3 TQueryFieldGroup.StyleRule Property

Applies a style in the page producer's style sheet to the table.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated set of fields. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style property.

See Also

Style (see page 16527)

Styles (see page 17127)

StylesFile (see page 17128)

99.1.36 TQueryForm Class

TQueryForm generates HTML forms for letting users enter application-defined values.

Class Hierarchy

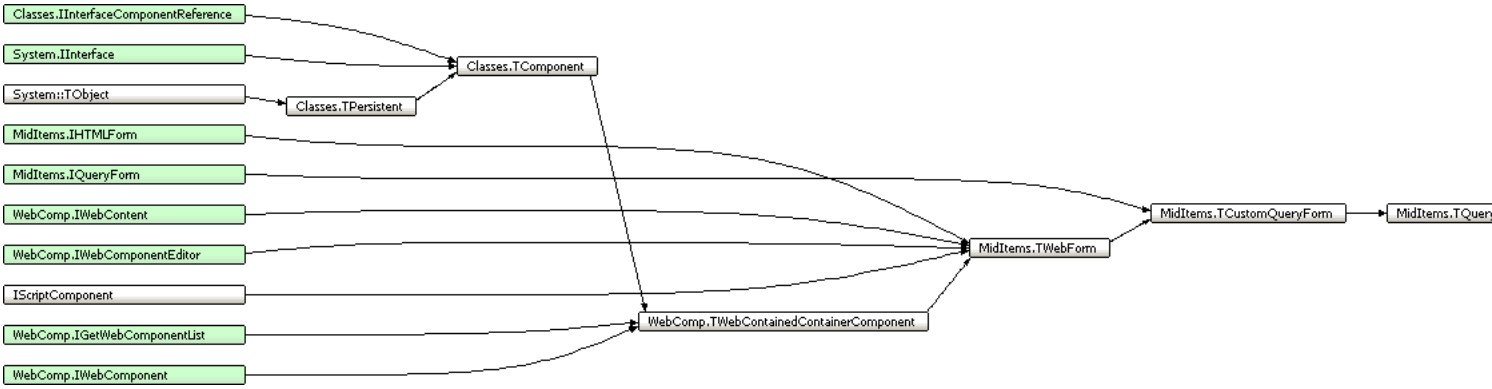

Pascal

```
TQueryForm = class(TCustomQueryForm);
```

C++

```
class TQueryForm : public TCustomQueryForm;
```

File

MidItems

Description

Add TQueryForm to the Web items of an InternetExpress page producer to add an HTML form that displays application-defined values that users can view and edit. Query forms submit these values to the InternetExpress application using a specified URL and method type.

TQueryForm generates the HTML FORM tag and assembles the HTML generated by its Web items. It restricts the Web items that generate the contents of the form to components that display application-defined values (TQueryFieldGroup (see page 16736)), components that contain buttons to operate on those displays (TQueryButtons (see page 16730)), and components that manage the layout of HTML controls (TLayoutGroup (see page 16687)). At design time, you can add these Web items to the form using the Web page editor.

See Also

TQueryButtons (see page 16730)

TBasePageItemsProducer (see page 18376)

TInetXPageProducer (see page 17145)

TQueryFieldGroup (see page 16736)

TLayoutGroup (see page 16687)

Using the Web Page Editor

99.1.36.1 TQueryForm Members

The following tables list the members exposed by TQueryForm.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16927)	Used internally to generate any javascript declarations needed by the form.
	AssignStringsCallback (see page 16551)	This is AssignStringsCallback, a member of class TCustomQueryForm.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16928)	Specifies whether a Web item of a specified type can be added to the WebComponents (see page 16934) property list.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16928)	Returns the HTML that defines the HTML form.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultWebComponents (see page 16929)	Returns nil (Delphi) or NULL (C++).
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetHTMLFormName (see page 16929)	Returns the name of the generated HTML form.
V	GetHTMLFormTag (see page 16551)	This is GetHTMLFormTag, a member of class TCustomQueryForm.
	GetHTMLFormVarName (see page 16929)	Returns the name of the javascript variable that refers to the generated form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16930)	Returns the object that specifies how the generated form should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16930)	Returns the value of the WebComponents (see page 16934) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVisibleFields (see page 16930)	Returns the value of the VisibleFields (see page 16934) property.
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
V	ImplAddElements (see page 16552)	Provides the underlying implementation for the AddElements method.
V	ImplAddElements (see page 16931)	Provides the underlying implementation for the AddElements (see page 16927) method.
V	ImplCanAddClass (see page 16552)	Provides the underlying implementation for the CanAddClass method.
A	ImplCanAddClass (see page 16931)	Provides the underlying implementation for the CanAddClass (see page 16928) method.
V	ImplContent (see page 16552)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 16932)	Provides the underlying implementation for the Content (see page 16928) method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.

V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	MethodString (see page 16553)	This is MethodString, a member of class TCustomQueryForm.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16932)	Prevents an application from trying to assign a default set of Web items on the form.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	VisibleFields (see page 16934)	Lists the Web items whose generated content appears in the HTML form.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
	AssignStrings (see page 16551)	This is AssignStrings, a member of class TCustomQueryForm.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16550)	Creates and initializes an instance of TCustomQueryForm (see page 16545).
V	Create (see page 16926)	Creates and initializes an instance of TWebForm (see page 16922).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24742)	Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16926)	Frees an instance of TWebForm (see page 16922).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHTMLFormName (see page 17065)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 17065)	Returns the name of the javascript variable that refers to the generated form.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLFormName (see page 17066)	Indicates the name of the generated HTML form.
R	HTMLFormVarName (see page 17066)	Indicates the name of the javascript variable that represents the HTML form.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebComponents (see page 16934)	Maintains a list of components that generate HTML content for the TWebForm (see page 16922) descendant.

Published Properties

	Name	Description
	Action (see page 16751)	Specifies the path info portion of the URL to which the query submits request messages.
	Custom (see page 16751)	Represents a tag option and its value in the generated FORM tag.
	Method (see page 16752)	This is Method, a member of class TQueryForm.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16752)	Adds style attributes to the generated form.
	StyleRule (see page 16752)	Applies a style in the page producer's style sheet to the button.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.36.2 TQueryForm Properties

The properties of the TQueryForm class are listed here.

Published Properties

	Name	Description
	Action (see page 16751)	Specifies the path info portion of the URL to which the query submits request messages.
	Custom (see page 16751)	Represents a tag option and its value in the generated FORM tag.
	Method (see page 16752)	This is Method, a member of class TQueryForm.
	Style (see page 16752)	Adds style attributes to the generated form.
	StyleRule (see page 16752)	Applies a style in the page producer's style sheet to the button.

99.1.36.2.1 TQueryForm.Action Property

Specifies the path info portion of the URL to which the query submits request messages.

Pascal

```
property Action: string;
```

C++

```
__property AnsiString Action;
```

Description

Use Action to specify the target URL for messages that the query form submits. Action supplies the PathInfo property of Web request messages from the query form. This value is matched to the PathInfo property of a Web action item when determining what object responds to those messages.

See Also

TCustomQueryForm (see page 16545)

PathInfo (see page 15185)

PathInfo (see page 15138)

99.1.36.2.2 TQueryForm.Custom Property

Represents a tag option and its value in the generated FORM tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to represent one or more options that qualify the FORM tag generated by the TWebForm descendant. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to specify that the form includes files with the text it submits and indicate that the page returned after submitting the form goes into a separate window, use a Custom value such as the following:

```
ENCTYPE="multipart/form-data" TARGET="ResultFrame"
```

See Also

Style (see page 16933)

99.1.36.2.3 TQueryForm.Method Property

Pascal

```
property Method: TFormMethod;
```

C++

```
__property TFormMethod Method;
```

Description

This is Method, a member of class TQueryForm.

99.1.36.2.4 TQueryForm.Style Property

Adds style attributes to the generated form.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to add a STYLE option that qualifies the FORM tag generated by the TWebForm descendant. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated FORM tag as the value of a STYLE option.

See Also

Custom (see page 16933)

StyleRule (see page 16934)

99.1.36.2.5 TQueryForm.StyleRule Property

Applies a style in the page producer's style sheet to the button.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated form. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style property.

See Also

Style (see page 16933)

Styles (see page 17127)

StylesFile (see page 17128)

99.1.37 TQueryRadioGroup Class

TQueryRadioGroup creates an HTML radio group control to represent named parameter.

Class Hierarchy

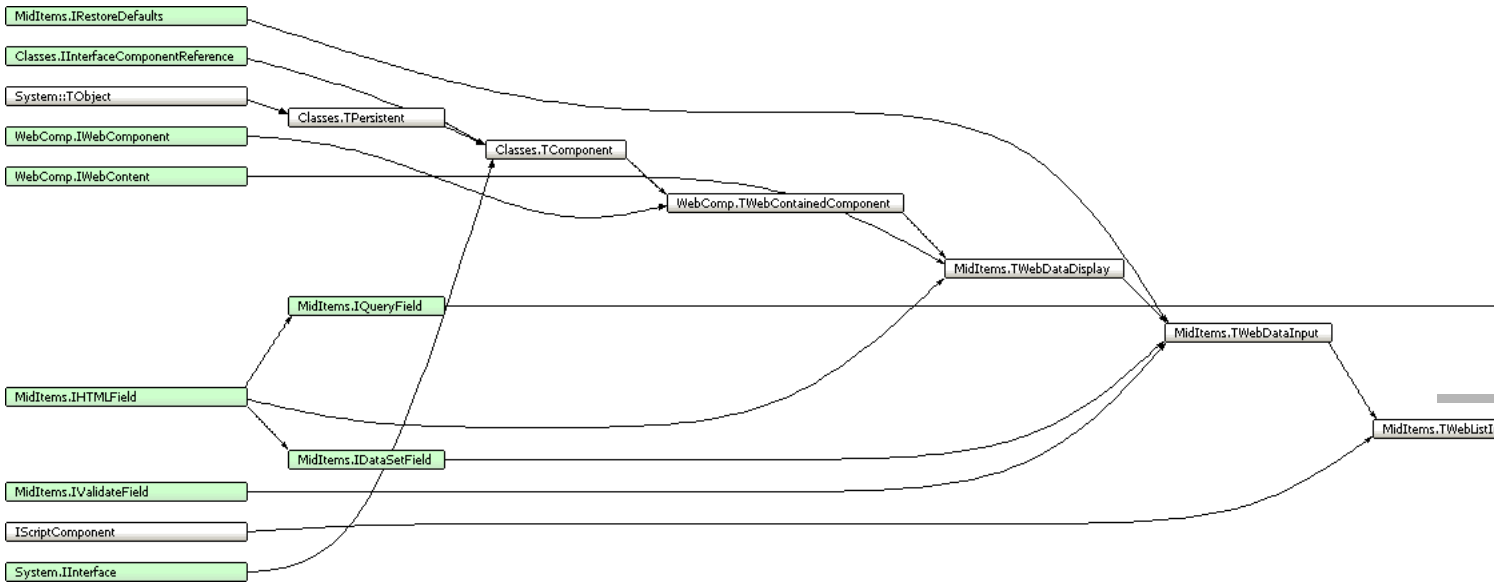

Pascal

```
TQueryRadioGroup = class(TWebRadioGroupInput, IQueryField);
```

C++

```
class TQueryRadioGroup : public TWebRadioGroupInput, public IQueryField;
```

File

MidItems

Description

Add TQueryRadioGroup as the child of a TQueryFieldGroup (see page 16736) component to add a radio group control that represents a parameter value. By default, the children of TQueryFieldGroup (see page 16736) represent the parameters of an XML broker. However, TQueryRadioGroup can represent any named value that the InternetExpress application uses.

Use the properties and methods of TQueryRadioGroup to

Specify the field or parameter whose value the radio group displays.

Specify the possible values from which the user can choose.

Initialize the value of the radio group.

Specify a caption that describes the value of the radio group.

Customize the appearance of the radio group.

Fit the radio group into the tab order of the generated HTML form.

See Also

TQueryFieldGroup (see page 16736)

TXMLDisplayGroup (see page 17019)

TXMLDisplayGroup (see page 17019)

TXMLBroker (see page 26049)

99.1.37.1 TQueryRadioGroup Members

The following tables list the members exposed by TQueryRadioGroup.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddAttributes (see page 16759)	This is AddAttributes, a member of class TQueryRadioGroup.
	AddElements (see page 16941)	This is AddElements, a member of class TWebListInput.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 16942)	This is ControlContent, a member of class TWebListInput.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	EventContent (see page 16905)	This is EventContent, a member of class TWebDataInput.
V	EventContent (see page 16953)	This is EventContent, a member of class TWebRadioGroupInput.
V	FindAssociatedField (see page 16906)	This is FindAssociatedField, a member of class TWebDataInput.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
A	FormatInputs (see page 16942)	This is FormatInputs, a member of class TWebListInput.
V	FormatInputs (see page 16953)	This is FormatInputs, a member of class TWebRadioGroupInput.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 16906)	Returns the value of the Caption property.
V	GetCheckIndex (see page 16759)	This is GetCheckIndex, a member of class TQueryRadioGroup.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetFieldName (see page 16906)	This is GetFieldName, a member of class TWebDataInput.

	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetParamName (see page 16906)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 16907)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetSubComponents (see page 16942)	This is GetSubComponents, a member of class TWebListInput.
	GetText (see page 16759)	This is GetText, a member of class TQueryRadioGroup.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplAddElements (see page 16942)	This is ImplAddElements, a member of class TWebListInput.
V	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
V	ImplGetFieldName (see page 16907)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 16907)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 16907)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplGetRowSetFieldAttributes (see page 16954)	This is ImplGetRowSetFieldAttributes, a member of class TWebRadioGroupInput.
V	ImplRestoreDefaults (see page 16908)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16943)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 16908)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16908)	This is RestoreFieldDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16954)	This is RestoreFieldDefaults, a member of class TWebRadioGroupInput.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
V	SetCaption (see page 16908)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 16943)	This is SetDataSet, a member of class TWebListInput.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 16909)	This is SetFieldName, a member of class TWebDataInput.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItems (see page 16943)	This is SetItems, a member of class TWebListInput.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParamName (see page 16909)	This is SetParamName, a member of class TWebDataInput.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetText (see page 16760)	This is SetText, a member of class TQueryRadioGroup.
	SetValues (see page 16944)	This is SetValues, a member of class TWebListInput.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateField (see page 16909)	This is ValidateField, a member of class TWebDataInput.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16889)	Creates and initializes an instance of TWebDataDisplay (see page 16884).
V	Create (see page 16940)	Creates and initializes an instance of TWebListInput (see page 16935).
V	Create (see page 16952)	Creates and initializes an instance of TWebRadioGroupInput (see page 16947).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).

	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
	Destroy (see page 16941)	Frees an instance of TWebListInput (see page 16935).
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 17057)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParamName (see page 17071)	This is GetParamName, a member of class IQueryField.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRowSetFieldAttributes (see page 17058)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsQueryField (see page 16760)	Indicates whether the component generates a control that can appear in a query form.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 17058)	This is SetFieldName, a member of class IDataSetField.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetParamName (see page 17071)	This is SetParamName, a member of class IQueryField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateField (see page 17076)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Caption (see page 16761)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16761)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16761)	Specifies where the label appears relative to the generated display control.
	Custom (see page 16762)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 16762)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayColumns (see page 16763)	Specifies the number of radio buttons that appear in each row of the control.
	DisplayWidth (see page 16763)	Specifies the width of the radio group, in characters.
	FieldName (see page 16764)	Specifies the name of the field whose value the generated control represents.
	Items (see page 16764)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 16765)	Indicates which field in DataSet supplies the possible selections for the generated control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParamName (see page 16765)	Specifies the name of the parameter whose value the generated control represents.
	ReadOnly (see page 16766)	Indicates whether end users can use the generated control to set values.
	Style (see page 16766)	Adds style attributes to the generated display control.
	StyleRule (see page 16766)	This is StyleRule, a member of class TQueryRadioGroup.

	TabIndex (see page 16767)	Determines the position of the generated control in the HTML form's tab order.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Text (see page 16767)	Specifies the initial value of the radio group.
	Values (see page 16767)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 16768)	Indicates which field in DataSet supplies the possible values for the generated control.

99.1.37.2 TQueryRadioGroup Methods

The methods of the TQueryRadioGroup class are listed here.

Protected Methods

	Name	Description
V	AddAttributes (see page 16759)	This is AddAttributes, a member of class TQueryRadioGroup.
V	GetCheckIndex (see page 16759)	This is GetCheckIndex, a member of class TQueryRadioGroup.
	GetText (see page 16759)	This is GetText, a member of class TQueryRadioGroup.
	SetText (see page 16760)	This is SetText, a member of class TQueryRadioGroup.

Public Methods

	Name	Description
V	IsQueryField (see page 16760)	Indicates whether the component generates a control that can appear in a query form.

99.1.37.2.1 TQueryRadioGroup.AddAttributes Method

Pascal

```
procedure AddAttributes(var Attrs: string); override;
```

C++

```
virtual __fastcall AddAttributes(AnsiString Attrs);
```

Description

This is AddAttributes, a member of class TQueryRadioGroup.

99.1.37.2.2 TQueryRadioGroup.GetCheckIndex Method

Pascal

```
function GetCheckIndex(ItemsStrings: TStrings; ValuesStrings: TStrings): Integer; override;
```

C++

```
virtual __fastcall int GetCheckIndex(TStrings ItemsStrings, TStrings ValuesStrings);
```

Description

This is GetCheckIndex, a member of class TQueryRadioGroup.

99.1.37.2.3 TQueryRadioGroup.GetText Method

Pascal

```
function GetText: string;
```

C++

```
__fastcall AnsiString GetText();
```

Description

This is GetText, a member of class TQueryRadioGroup.

99.1.37.2.4 TQueryRadioGroup.IsQueryField Method

Indicates whether the component generates a control that can appear in a query form.

Pascal

```
class function IsQueryField: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsQueryField();
```

Description

Call IsQueryField to determine whether a Web display component generates a control that represents a named value in a query form (TQueryForm (see page 16745)). IsQueryField returns true, because TQueryRadioGroup (see page 16753) is designed specifically for use by TQueryFieldGroup (see page 16736).

See Also

IsColumn (see page 16895)

TQueryFieldGroup (see page 16736)

TQueryForm (see page 16745)

99.1.37.2.5 TQueryRadioGroup.SetText Method

Pascal

```
procedure SetText(const Value: string);
```

C++

```
__fastcall SetText(const AnsiString Value);
```

Description

This is SetText, a member of class TQueryRadioGroup.

99.1.37.3 TQueryRadioGroup Properties

The properties of the TQueryRadioGroup class are listed here.

Published Properties

	Name	Description
	Caption (see page 16761)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16761)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16761)	Specifies where the label appears relative to the generated display control.
	Custom (see page 16762)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 16762)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayColumns (see page 16763)	Specifies the number of radio buttons that appear in each row of the control.
	DisplayWidth (see page 16763)	Specifies the width of the radio group, in characters.
	FieldName (see page 16764)	Specifies the name of the field whose value the generated control represents.
	Items (see page 16764)	Lists the possible selections that appear in the generated control.

	ItemsField (see page 16765)	Indicates which field in DataSet supplies the possible selections for the generated control.
	ParamName (see page 16765)	Specifies the name of the parameter whose value the generated control represents.
	ReadOnly (see page 16766)	Indicates whether end users can use the generated control to set values.
	Style (see page 16766)	Adds style attributes to the generated display control.
	StyleRule (see page 16766)	This is StyleRule, a member of class TQueryRadioGroup.
	TabIndex (see page 16767)	Determines the position of the generated control in the HTML form's tab order.
	Text (see page 16767)	Specifies the initial value of the radio group.
	Values (see page 16767)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 16768)	Indicates which field in DataSet supplies the possible values for the generated control.

99.1.37.3.1 TQueryRadioGroup.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 16897)

CaptionPosition (see page 16897)

99.1.37.3.2 TQueryRadioGroup.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 16897)

CaptionPosition (see page 16897)

HeadingAttributes (see page 16509)

99.1.37.3.3 TQueryRadioGroup.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 16897)

Caption (see page 16897)

99.1.37.3.4 TQueryRadioGroup.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 16899)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

99.1.37.3.5 TQueryRadioGroup.DataSet Property

Specifies a dataset that supplies the possible values listed in the generated control.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Use DataSet to specify a dataset from which the component can extract the selections that appear to the user and the underlying values they represent. The ItemsField property specifies the field in this dataset that lists the selections that appear in the generated control. The ValuesField property specifies the field in this dataset that lists the corresponding values that are assigned to the control.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

ItemsField (see page 16945)

ValuesField (see page 16946)

Items (see page 16945)

Values (see page 16946)

99.1.37.3.6 TQueryRadioGroup.DisplayColumns Property

Specifies the number of radio buttons that appear in each row of the control.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Use DisplayColumns to specify how many radio buttons can appear in a single line of the radio group. If DisplayColumns is less than the number of items, the radio group displays its buttons in multiple rows.

See Also

ItemsField (see page 16945)

Items (see page 16945)

DisplayWidth (see page 16955)

99.1.37.3.7 TQueryRadioGroup.DisplayWidth Property

Specifies the width of the radio group, in characters.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to limit the width of the generated control. TWebRadioGroup generates an INPUT tag of type RADIO to implement the radio group. DisplayWidth is added as a WIDTH option.

See Also

DisplayColumns (see page 16954)

Custom (see page 16898)

99.1.37.3.8 TQueryRadioGroup.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 17033)

XMLDataSetField (see page 17033)

ParamName (see page 16910)

Setting Web Item Properties

99.1.37.3.9 TQueryRadioGroup.Items Property

Lists the possible selections that appear in the generated control.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to supply a list of possible selections to the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the corresponding entry in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

ItemsField (see page 16945)

Values (see page 16946)

DataSet (see page 16944)

ValuesField (see page 16946)

99.1.37.3.10 TQueryRadioGroup.ItemsField Property

Indicates which field in DataSet supplies the possible selections for the generated control.

Pascal

```
property ItemsField: string;
```

C++

```
__property AnsiString ItemsField;
```

Description

Use ItemsField to specify which field in the dataset specified by the DataSet property contains selections that appear in the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the field specified by the ValuesField property.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 16945)

Values (see page 16946)

DataSet (see page 16944)

ValuesField (see page 16946)

99.1.37.3.11 TQueryRadioGroup.ParamName Property

Specifies the name of the parameter whose value the generated control represents.

Pascal

```
property ParamName: string;
```

C++

```
__property AnsiString ParamName;__property AnsiString ParamName;
```

Description

Use ParamName to bind the Web data input component to a parameter in the XML data packet or supply an application-specific name to the value for this control. ParamName is only used when the IsQueryField method returns true. By default, the controls in a query form represent parameter values. However, they can be used to represent any named value.

If ParamName identifies one of the parameters of an XML broker, the XML broker property of the parent Web item identifies the broker that uses the parameter.

See Also

FieldName (see page 16909)

IsQueryField (see page 16895)

Params (see page 26073)

xmlbroker (see page 17033)

Setting Web Item Properties

99.1.37.3.12 TQueryRadioGroup.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the radio group. TWebRadioGroup generates an INPUT tag of type RADIO to implement the radio group. ReadOnly is added as a READONLY option.

See Also

DisplayWidth (see page 16955)

Custom (see page 16898)

99.1.37.3.13 TQueryRadioGroup.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 16898)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

Setting Web Item Properties

99.1.37.3.14 TQueryRadioGroup.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TQueryRadioGroup.

99.1.37.3.15 TQueryRadioGroup.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

99.1.37.3.16 TQueryRadioGroup.Text Property

Specifies the initial value of the radio group.

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

Use Text to initialize the value that appears in the generated radio group. This should be one of the values listed in the Items (see page 16764) property or in the field referenced by the ItemsField (see page 16765) property.

See Also

Items (see page 16945)

ItemsField (see page 16945)

99.1.37.3.17 TQueryRadioGroup.Values Property

Lists the possible values that can be assigned to the control.

Pascal

```
property Values: TStrings;
```

C++

```
__property TStrings Values;
```

Description

Use Values to supply a list of possible values for the generated control. When the end-user chooses one of the selections listed by the Items property, the control is assigned the corresponding value in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

Items (see page 16945)

ItemsField (see page 16945)

DataSet (see page 16944)

ValuesField (see page 16946)

99.1.37.3.18 TQueryRadioGroup.ValuesField Property

Indicates which field in DataSet supplies the possible values for the generated control.

Pascal

```
property ValuesField: string;
```

C++

```
__property AnsiString ValuesField;
```

Description

Use ValuesField to specify which field in the dataset specified by the DataSet property contains values that can be assigned to the generated control. When the end-user selects the corresponding value form the ItemsField field, the control gets the value in the ValuesField field.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 16945)

Values (see page 16946)

DataSet (see page 16944)

ItemsField (see page 16945)

99.1.38 TQuerySelectOptions Class

TQuerySelectOptions creates an HTML list control to represent a named parameter.

Class Hierarchy

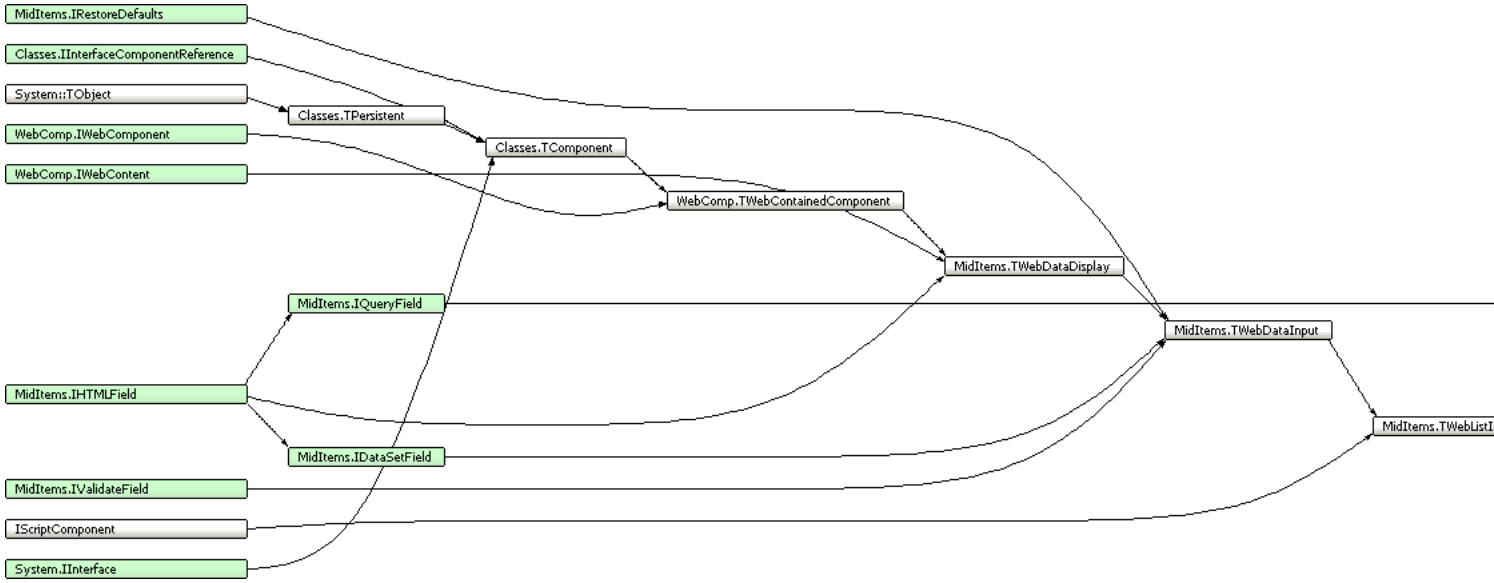

Pascal

```
TQuerySelectOptions = class(TWebSelectOptionsInput, IQueryField);
```

C++

```
class TQuerySelectOptions : public TWebSelectOptionsInput, public IQueryField;
```

File

MidItems

Description

Add TQuerySelectOptions as the child of a TQueryFieldGroup (see page 16736) component to add a list control that represents a parameter value. By default, the children of TQueryFieldGroup (see page 16736) represent the parameters of an XML broker. However, TQuerySelectOptions can represent any named value that the InternetExpress application uses.

Use the properties and methods of TQuerySelectOptions to

Specify the field or parameter whose value the list box displays.

Specify the possible values from which the user can choose.

Assign an initial value to the list box.

Specify a caption that describes the value of the list box.

Customize the appearance of the list box.

Fit the list box into the tab order of the generated HTML form.

See Also

TQueryFieldGroup (see page 16736)

TXMLDisplayGroup (see page 17019)

TXMLDisplayGroup (see page 17019)

TXMLBroker (see page 26049)

99.1.38.1 TQuerySelectOptions Members

The following tables list the members exposed by TQuerySelectOptions.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddAttributes (see page 16962)	This is AddAttributes, a member of class TWebSelectOptionsInput.
	AddElements (see page 16941)	This is AddElements, a member of class TWebListInput.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 16942)	This is ControlContent, a member of class TWebListInput.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	EventContent (see page 16905)	This is EventContent, a member of class TWebDataInput.
V	EventContent (see page 16962)	This is EventContent, a member of class TWebSelectOptionsInput.
V	FindAssociatedField (see page 16906)	This is FindAssociatedField, a member of class TWebDataInput.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
A	FormatInputs (see page 16942)	This is FormatInputs, a member of class TWebListInput.
V	FormatInputs (see page 16962)	This is FormatInputs, a member of class TWebSelectOptionsInput.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 16906)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetFieldName (see page 16906)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetParamName (see page 16906)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 16907)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
V	GetSelectIndex (see page 16775)	This is GetSelectIndex, a member of class TQuerySelectOptions.
	GetSubComponents (see page 16942)	This is GetSubComponents, a member of class TWebListInput.
	GetText (see page 16775)	This is GetText, a member of class TQuerySelectOptions.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplAddElements (see page 16942)	This is ImplAddElements, a member of class TWebListInput.
V	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
V	ImplGetFieldName (see page 16907)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 16907)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 16907)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplRestoreDefaults (see page 16908)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16943)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 16908)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16908)	This is RestoreFieldDefaults, a member of class TWebDataInput.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
V	SetCaption (see page 16908)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 16943)	This is SetDataSet, a member of class TWebListInput.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 16909)	This is SetFieldName, a member of class TWebDataInput.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItems (see page 16943)	This is SetItems, a member of class TWebListInput.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParamName (see page 16909)	This is SetParamName, a member of class TWebDataInput.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetText (see page 16776)	This is SetText, a member of class TQuerySelectOptions.
	SetValues (see page 16944)	This is SetValues, a member of class TWebListInput.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateField (see page 16909)	This is ValidateField, a member of class TWebDataInput.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16889)	Creates and initializes an instance of TWebDataDisplay (see page 16884).
V	Create (see page 16940)	Creates and initializes an instance of TWebListInput (see page 16935).
V	Create (see page 16961)	Creates and initializes an instance of TWebSelectOptionsInput (see page 16956).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
V	Destroy (see page 16941)	Frees an instance of TWebListInput (see page 16935).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 17057)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParamName (see page 17071)	This is GetParamName, a member of class IQueryField.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRowSetFieldAttributes (see page 17058)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsQueryField (see page 16775)	Indicates whether the component generates a control that can appear in a query form.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 17058)	This is SetFieldName, a member of class IDataSetField.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetParamName (see page 17071)	This is SetParamName, a member of class IQueryField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateField (see page 17076)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Caption (see page 16776)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16777)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16777)	Specifies where the label appears relative to the generated display control.
	Custom (see page 16777)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 16778)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayRows (see page 16778)	Specifies the number of rows that appear in the generated list box.
	FieldName (see page 16779)	Specifies the name of the field whose value the generated control represents.
	Items (see page 16779)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 16780)	Indicates which field in DataSet supplies the possible selections for the generated control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParamName (see page 16780)	Specifies the name of the parameter whose value the generated control represents.
	Style (see page 16781)	Adds style attributes to the generated display control.
	StyleRule (see page 16781)	This is StyleRule, a member of class TQuerySelectOptions.
	TabIndex (see page 16781)	Determines the position of the generated control in the HTML form's tab order.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Text (see page 16782)	Specifies the initial value selected in the list box.
	Values (see page 16782)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 16782)	Indicates which field in DataSet supplies the possible values for the generated control.

99.1.38.2 TQuerySelectOptions Methods

The methods of the TQuerySelectOptions class are listed here.

Protected Methods

	Name	Description
V	GetSelectIndex (see page 16775)	This is GetSelectIndex, a member of class TQuerySelectOptions.
	GetText (see page 16775)	This is GetText, a member of class TQuerySelectOptions.
	SetText (see page 16776)	This is SetText, a member of class TQuerySelectOptions.

Public Methods

	Name	Description
	IsQueryField (see page 16775)	Indicates whether the component generates a control that can appear in a query form.

99.1.38.2.1 TQuerySelectOptions.GetSelectIndex Method

Pascal

```
function GetSelectIndex(ItemsStrings: TStrings; ValuesStrings: TStrings): Integer; override;
```

C++

```
virtual __fastcall int GetSelectIndex(TStrings ItemsStrings, TStrings ValuesStrings);
```

Description

This is GetSelectIndex, a member of class TQuerySelectOptions.

99.1.38.2.2 TQuerySelectOptions.GetText Method

Pascal

```
function GetText: string;
```

C++

```
__fastcall AnsiString GetText();
```

Description

This is GetText, a member of class TQuerySelectOptions.

99.1.38.2.3 TQuerySelectOptions.IsQueryField Method

Indicates whether the component generates a control that can appear in a query form.

Pascal

```
class function IsQueryField: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsQueryField();
```

Description

Call IsQueryField to determine whether a Web display component generates a control that represents a named value in a query form (TQueryForm (see page 16745)). IsQueryField returns true, because TQuerySelectOptions (see page 16768) is designed specifically for use by TQueryFieldGroup (see page 16736).

See Also

IsColumn (see page 16895)

TQueryFieldGroup (see page 16736)

TQueryForm (see page 16745)

99.1.38.2.4 TQuerySelectOptions.SetText Method

Pascal

```
procedure SetText(const Value: string);
```

C++

```
__fastcall SetText(const AnsiString Value);
```

Description

This is SetText, a member of class TQuerySelectOptions.

99.1.38.3 TQuerySelectOptions Properties

The properties of the TQuerySelectOptions class are listed here.

Published Properties

	Name	Description
	Caption (see page 16776)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16777)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16777)	Specifies where the label appears relative to the generated display control.
	Custom (see page 16777)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 16778)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayRows (see page 16778)	Specifies the number of rows that appear in the generated list box.
	FieldName (see page 16779)	Specifies the name of the field whose value the generated control represents.
	Items (see page 16779)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 16780)	Indicates which field in DataSet supplies the possible selections for the generated control.
	ParamName (see page 16780)	Specifies the name of the parameter whose value the generated control represents.
	Style (see page 16781)	Adds style attributes to the generated display control.
	StyleRule (see page 16781)	This is StyleRule, a member of class TQuerySelectOptions.
	TabIndex (see page 16781)	Determines the position of the generated control in the HTML form's tab order.
	Text (see page 16782)	Specifies the initial value selected in the list box.
	Values (see page 16782)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 16782)	Indicates which field in DataSet supplies the possible values for the generated control.

99.1.38.3.1 TQuerySelectOptions.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 16897)

CaptionPosition (see page 16897)

99.1.38.3.2 TQuerySelectOptions.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 16897)

CaptionPosition (see page 16897)

HeadingAttributes (see page 16509)

99.1.38.3.3 TQuerySelectOptions.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 16897)

Caption (see page 16897)

99.1.38.3.4 TQuerySelectOptions.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to

add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 16899)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

99.1.38.3.5 TQuerySelectOptions.DataSet Property

Specifies a dataset that supplies the possible values listed in the generated control.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Use DataSet to specify a dataset from which the component can extract the selections that appear to the user and the underlying values they represent. The ItemsField property specifies the field in this dataset that lists the selections that appear in the generated control. The ValuesField property specifies the field in this dataset that lists the corresponding values that are assigned to the control.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

ItemsField (see page 16945)

ValuesField (see page 16946)

Items (see page 16945)

Values (see page 16946)

99.1.38.3.6 TQuerySelectOptions.DisplayRows Property

Specifies the number of rows that appear in the generated list box.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to specify how many rows appear in the generated list control. If DisplayRows is less than the number of

items, the generated list displays only DisplayRows options at one time, but can scroll to display the hidden items. If DisplayRows is 1, the generated control becomes a drop-down list box.

See Also

ItemsField (see page 16945)

Items (see page 16945)

99.1.38.3.7 TQuerySelectOptions.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 17033)

XMLDataSetField (see page 17033)

ParamName (see page 16910)

Setting Web Item Properties

99.1.38.3.8 TQuerySelectOptions.Items Property

Lists the possible selections that appear in the generated control.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to supply a list of possible selections to the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the corresponding entry in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

ItemsField (see page 16945)

Values (see page 16946)

DataSet (see page 16944)

ValuesField (see page 16946)

99.1.38.3.9 TQuerySelectOptions.ItemsField Property

Indicates which field in DataSet supplies the possible selections for the generated control.

Pascal

```
property ItemsField: string;
```

C++

```
__property AnsiString ItemsField;
```

Description

Use ItemsField to specify which field in the dataset specified by the DataSet property contains selections that appear in the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the field specified by the ValuesField property.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 16945)

Values (see page 16946)

DataSet (see page 16944)

ValuesField (see page 16946)

99.1.38.3.10 TQuerySelectOptions.ParamName Property

Specifies the name of the parameter whose value the generated control represents.

Pascal

```
property ParamName: string;
```

C++

```
__property AnsiString ParamName;__property AnsiString ParamName;
```

Description

Use ParamName to bind the Web data input component to a parameter in the XML data packet or supply an application-specific name to the value for this control. ParamName is only used when the IsQueryField method returns true. By default, the controls in a query form represent parameter values. However, they can be used to represent any named value.

If ParamName identifies one of the parameters of an XML broker, the XML broker property of the parent Web item identifies the broker that uses the parameter.

See Also

FieldName (see page 16909)

IsQueryField (see page 16895)

Params (see page 26073)

xmlbroker (see page 17033)

Setting Web Item Properties

99.1.38.3.11 TQuerySelectOptions.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 16898)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

Setting Web Item Properties

99.1.38.3.12 TQuerySelectOptions.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TQuerySelectOptions.

99.1.38.3.13 TQuerySelectOptions.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

99.1.38.3.14 TQuerySelectOptions.Text Property

Specifies the initial value selected in the list box.

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

Use Text to initialize the value of the generated list control. Text should specify one of the values listed in the Items (see page 16779) property or referenced by the ItemsField (see page 16780) property.

See Also

Items (see page 16945)

ItemsField (see page 16945)

99.1.38.3.15 TQuerySelectOptions.Values Property

Lists the possible values that can be assigned to the control.

Pascal

```
property Values: TStrings;
```

C++

```
__property TStrings Values;
```

Description

Use Values to supply a list of possible values for the generated control. When the end-user chooses one of the selections listed by the Items property, the control is assigned the corresponding value in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

Items (see page 16945)

ItemsField (see page 16945)

DataSet (see page 16944)

ValuesField (see page 16946)

99.1.38.3.16 TQuerySelectOptions.ValuesField Property

Indicates which field in DataSet supplies the possible values for the generated control.

Pascal

```
property ValuesField: string;
```

C++

```
__property AnsiString ValuesField;
```

Description

Use ValuesField to specify which field in the dataset specified by the DataSet property contains values that can be assigned to the generated control. When the end-user selects the corresponding value form the ItemsField field, the control gets the value in the ValuesField field.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 16945)

Values (see page 16946)

DataSet (see page 16944)

ItemsField (see page 16945)

99.1.39 TQueryText Class

TQueryText creates a single-line HTML input control to represent a named parameter.

Class Hierarchy

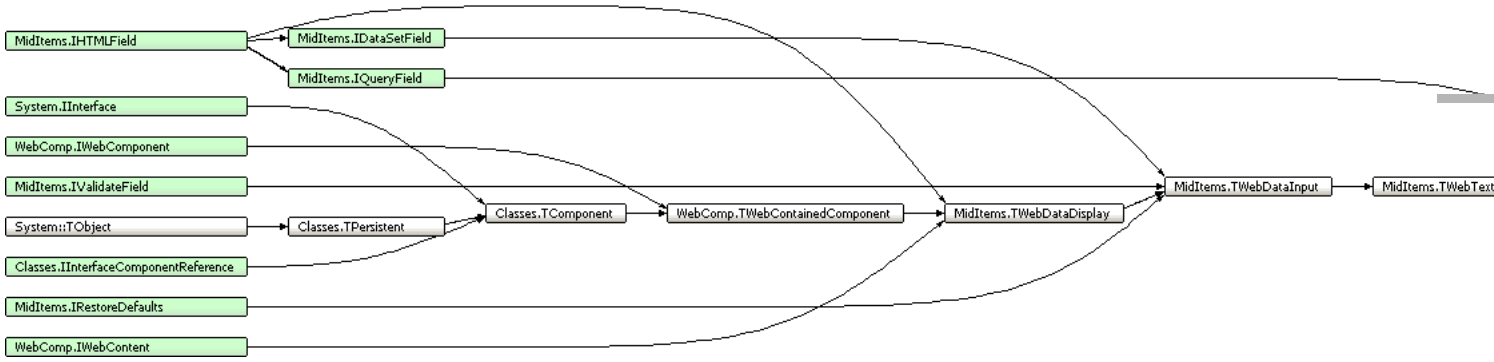

Pascal

```
TQueryText = class(TWebTextInput, IQueryField);
```

C++

```
class TQueryText : public TWebTextInput, public IQueryField;
```

File

MidItems

Description

Add TQueryText as the child of a TQueryFieldGroup (see page 16736) component to add an edit control that represents a parameter value. By default, the children of TQueryFieldGroup (see page 16736) represent the parameters of an XML broker. However, TQueryText can represent any named value that the InternetExpress application uses.

Use the properties and methods of TQueryText to

Specify the field or parameter whose value the control displays.

Assign an initial value to the control.

Specify a caption that describes the value of the control.

Customize the appearance of the input control.

Fit the control into the tab order of the generated HTML form.

See Also

TQueryFieldGroup (see page 16736)

TXMLDisplayGroup (see page 17019)

TXMLDisplayGroup (see page 17019)

TXMLBroker (see page 26049)

99.1.39.1 TQueryText Members

The following tables list the members exposed by TQueryText.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddAttributes (see page 16789)	This is AddAttributes, a member of class TQueryText.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 16987)	This is ControlContent, a member of class TWebTextInput.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	EventContent (see page 16905)	This is EventContent, a member of class TWebDataInput.
V	FindAssociatedField (see page 16906)	This is FindAssociatedField, a member of class TWebDataInput.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 16906)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetFieldName (see page 16906)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.

	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetParamName (see page 16906)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 16907)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetText (see page 16789)	This is GetText, a member of class TQueryText.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
	ImplGetFieldName (see page 16907)	This is ImplGetFieldName, a member of class TWebDataInput.
	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
	ImplGetHTMLControlName (see page 16907)	Returns the name that other components can use to refer to the generated display control.
	ImplGetRowSetFieldAttributes (see page 16907)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
	ImplGetRowSetFieldAttributes (see page 16987)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextInput.
	ImplRestoreDefaults (see page 16908)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 16908)	This is RestoreDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 16908)	This is RestoreFieldDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 16987)	This is RestoreFieldDefaults, a member of class TWebTextInput.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
	SetCaption (see page 16908)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 16909)	This is SetFieldName, a member of class TWebDataInput.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParamName (see page 16909)	This is SetParamName, a member of class TWebDataInput.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetText (see page 16790)	This is SetText, a member of class TQueryText.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateField (see page 16909)	This is ValidateField, a member of class TWebDataInput.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
	Create (see page 16788)	Creates and initializes an instance of TQueryText (see page 16783).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 17057)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParamName (see page 17071)	This is GetParamName, a member of class IQueryField.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRowSetFieldAttributes (see page 17058)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsQueryField (see page 16789)	Indicates whether the component generates a control that can appear in a query form.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 17058)	This is SetFieldName, a member of class IDataSetField.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetParamName (see page 17071)	This is SetParamName, a member of class IQueryField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateField (see page 17076)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Caption (see page 16790)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16791)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16791)	Specifies where the label appears relative to the generated display control.
	Custom (see page 16791)	Represents a tag option and its value in the tag for the display control.
	DisplayWidth (see page 16792)	Specifies the horizontal size of the generated control.
	FieldName (see page 16792)	Specifies the name of the field whose value the generated control represents.
	MaxWidth (see page 16793)	Limits the length of text users can type into the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParamName (see page 16793)	Specifies the name of the parameter whose value the generated control represents.
	ReadOnly (see page 16793)	Indicates whether end users can use the generated control to set values.
	Style (see page 16794)	Adds style attributes to the generated display control.
	StyleRule (see page 16794)	This is StyleRule, a member of class TQueryText.
	TabIndex (see page 16794)	Determines the position of the generated control in the HTML form's tab order.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Text (see page 16795)	Specifies the initial value of the generated control.

99.1.39.2 TQueryText.Create Constructor

Creates and initializes an instance of TQueryText (see page 16783).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TQueryText(TComponent * AOwner);
```

Description

Call Create to instantiate TQueryText (see page 16783) at runtime. TQueryText (see page 16783) components added at design time using the Web Page editor are instantiated automatically.

99.1.39.3 TQueryText Methods

The methods of the TQueryText class are listed here.

Protected Methods

	Name	Description
	AddAttributes (see page 16789)	This is AddAttributes, a member of class TQueryText.
	GetText (see page 16789)	This is GetText, a member of class TQueryText.
	SetText (see page 16790)	This is SetText, a member of class TQueryText.

Public Methods

	Name	Description
	IsQueryField (see page 16789)	Indicates whether the component generates a control that can appear in a query form.

99.1.39.3.1 TQueryText.AddAttributes Method

Pascal

```
procedure AddAttributes(var Attrs: string); override;
```

C++

```
virtual __fastcall AddAttributes(AnsiString Attrs);
```

Description

This is AddAttributes, a member of class TQueryText.

99.1.39.3.2 TQueryText.GetText Method

Pascal

```
function GetText: string;
```

C++

```
__fastcall AnsiString GetText();
```

Description

This is GetText, a member of class TQueryText.

99.1.39.3.3 TQueryText.IsQueryField Method

Indicates whether the component generates a control that can appear in a query form.

Pascal

```
class function IsQueryField: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsQueryField();
```

Description

Call IsQueryField to determine whether a Web display component generates a control that represents a named value in a query form (TQueryForm (see page 16745)). IsQueryField returns true, because TQueryText (see page 16783) is designed specifically for TQueryFieldGroup (see page 16736) to use.

See Also

IsColumn (see page 16895)

TQueryFieldGroup (see page 16736)

TQueryForm (see page 16745)

99.1.39.3.4 TQueryText.SetText Method

Pascal

```
procedure SetText(const Value: string);
```

C++

```
__fastcall SetText(const AnsiString Value);
```

Description

This is SetText, a member of class TQueryText.

99.1.39.4 TQueryText Properties

The properties of the TQueryText class are listed here.

Published Properties

	Name	Description
	Caption (see page 16790)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16791)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16791)	Specifies where the label appears relative to the generated display control.
	Custom (see page 16791)	Represents a tag option and its value in the tag for the display control.
	DisplayWidth (see page 16792)	Specifies the horizontal size of the generated control.
	FieldName (see page 16792)	Specifies the name of the field whose value the generated control represents.
	MaxWidth (see page 16793)	Limits the length of text users can type into the control.
	ParamName (see page 16793)	Specifies the name of the parameter whose value the generated control represents.
	ReadOnly (see page 16793)	Indicates whether end users can use the generated control to set values.
	Style (see page 16794)	Adds style attributes to the generated display control.
	StyleRule (see page 16794)	This is StyleRule, a member of class TQueryText.
	TabIndex (see page 16794)	Determines the position of the generated control in the HTML form's tab order.
	Text (see page 16795)	Specifies the initial value of the generated control.

99.1.39.4.1 TQueryText.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 16897)

CaptionPosition (see page 16897)

99.1.39.4.2 TQueryText.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 16897)

CaptionPosition (see page 16897)

HeadingAttributes (see page 16509)

99.1.39.4.3 TQueryText.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 16897)

Caption (see page 16897)

99.1.39.4.4 TQueryText.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 16899)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

99.1.39.4.5 TQueryText.DisplayWidth Property

Specifies the horizontal size of the generated control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to indicate the desired width of the generated HTML control. DisplayWidth specifies the number of characters that can be displayed in the control.

Note: DisplayWidth controls the size of the generated control, not the size of the text that users can enter. If the user enters a string longer than DisplayWidth characters, the text scrolls in the control.

99.1.39.4.6 TQueryText.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 17033)

XMLDataSetField (see page 17033)

ParamName (see page 16910)

Setting Web Item Properties

99.1.39.4.7 TQueryText.MaxWidth Property

Limits the length of text users can type into the control.

Pascal

```
property MaxWidth: Integer;
```

C++

```
__property int MaxWidth;
```

Description

Use MaxWidth to limit the length of strings (in characters) that users can type into the generated control.

Note: MaxWidth has no effect on the size of the generated control. To limit the size of the control, use the DisplayWidth (see page 16792) property.

See Also

DisplayWidth (see page 16988)

99.1.39.4.8 TQueryText.ParamName Property

Specifies the name of the parameter whose value the generated control represents.

Pascal

```
property ParamName: string;
```

C++

```
__property AnsiString ParamName;__property AnsiString ParamName;
```

Description

Use ParamName to bind the Web data input component to a parameter in the XML data packet or supply an application-specific name to the value for this control. ParamName is only used when the IsQueryField method returns true. By default, the controls in a query form represent parameter values. However, they can be used to represent any named value.

If ParamName identifies one of the parameters of an XML broker, the XML broker property of the parent Web item identifies the broker that uses the parameter.

See Also

FieldName (see page 16909)

IsQueryField (see page 16895)

Params (see page 26073)

xmlbroker (see page 17033)

Setting Web Item Properties

99.1.39.4.9 TQueryText.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the generated control.

99.1.39.4.10 TQueryText.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 16898)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

Setting Web Item Properties

99.1.39.4.11 TQueryText.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TQueryText.

99.1.39.4.12 TQueryText.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

99.1.39.4.13 TQueryText.Text Property

Specifies the initial value of the generated control.

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

Use Text to initialize the value that appears in the generated display control.

See Also

FieldName (see page 16909)

ParamName (see page 16910)

99.1.40 TQueryTextArea Class

TQueryTextArea creates a multi-line HTML input control to represent named parameter.

Class Hierarchy

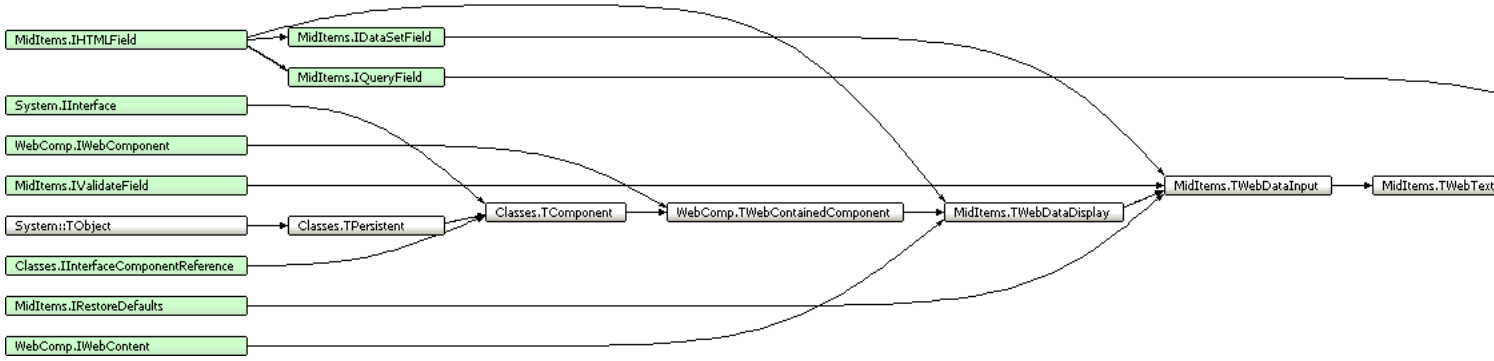

Pascal

```
TQueryTextArea = class(TWebTextAreaInput, IQueryField);
```

C++

```
class TQueryTextArea : public TWebTextAreaInput, public IQueryField;
```

File

MidItems

Description

Add TQueryTextArea as the child of a TQueryFieldGroup (see page 16736) component to add a multi-line edit control that

represents a parameter value. By default, the children of TQueryFieldGroup (see page 16736) represent the parameters of an XML broker. However, TQueryTextArea can represent any named value that the InternetExpress application uses.

Use the properties and methods of TQueryTextArea to

Specify the field or parameter whose value the control displays.

Assign an initial value to the control.

Specify a caption that describes the value of the control.

Customize the appearance of the control.

Fit the control into the tab order of the generated HTML form.

See Also

TQueryFieldGroup (see page 16736)

TXMLDisplayGroup (see page 17019)

TXMLDisplayGroup (see page 17019)

TXMLBroker (see page 26049)

99.1.40.1 TQueryTextArea Members

The following tables list the members exposed by TQueryTextArea.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddAttributes (see page 16801)	This is AddAttributes, a member of class TQueryTextArea.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
V	ControlContent (see page 16801)	This is ControlContent, a member of class TQueryTextArea.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	EventContent (see page 16905)	This is EventContent, a member of class TWebDataInput.
V	EventContent (see page 16978)	This is EventContent, a member of class TWebTextAreaInput.
V	FindAssociatedField (see page 16906)	This is FindAssociatedField, a member of class TWebDataInput.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 16906)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.

	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetFieldName (see page 16906)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetParamName (see page 16906)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 16907)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetText (see page 16802)	This is GetText, a member of class TQueryTextArea.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
	ImplGetFieldName (see page 16907)	This is ImplGetFieldName, a member of class TWebDataInput.
	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
	ImplGetHTMLControlName (see page 16907)	Returns the name that other components can use to refer to the generated display control.
	ImplGetRowSetFieldAttributes (see page 16907)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
	ImplGetRowSetFieldAttributes (see page 16978)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextAreaInput.
	ImplRestoreDefaults (see page 16908)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 16908)	This is RestoreDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 16908)	This is RestoreFieldDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 16978)	This is RestoreFieldDefaults, a member of class TWebTextAreaInput.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
	SetCaption (see page 16908)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 16909)	This is SetFieldName, a member of class TWebDataInput.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetLines (see page 16802)	This is SetLines, a member of class TQueryTextArea.

V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParamName (see page 16909)	This is SetParamName, a member of class TWebDataInput.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetText (see page 16803)	This is SetText, a member of class TQueryTextArea.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateField (see page 16909)	This is ValidateField, a member of class TWebDataInput.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16800)	Creates and initializes an instance of TQueryTextArea (see page 16795).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 16801)	Frees an instance of TQueryTextArea (see page 16795).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 17057)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParamName (see page 17071)	This is GetParamName, a member of class IQueryField.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRowSetFieldAttributes (see page 17058)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsQueryField (see page 16802)	Indicates whether the component generates a control that can appear in a query form.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 17058)	This is SetFieldName, a member of class IDataSetField.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetParamName (see page 17071)	This is SetParamName, a member of class IQueryField.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateField (see page 17076)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Text (see page 17072)	This is Text, a member of class IQueryField.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Caption (see page 16803)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16804)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16804)	Specifies where the label appears relative to the generated display control.
	Custom (see page 16804)	Represents a tag option and its value in the tag for the display control.
	DisplayRows (see page 16805)	Specifies the height of the generated control.
	DisplayWidth (see page 16805)	Specifies the horizontal size of the generated control.
	FieldName (see page 16805)	Specifies the name of the field whose value the generated control represents.
	Lines (see page 16806)	Specifies the initial value that appears in the multi-line edit control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParamName (see page 16806)	Specifies the name of the parameter whose value the generated control represents.
	ReadOnly (see page 16807)	Indicates whether end users can use the generated control to set values.
	Style (see page 16807)	Adds style attributes to the generated display control.
	StyleRule (see page 16808)	This is StyleRule, a member of class TQueryTextArea.
	TabIndex (see page 16808)	Determines the position of the generated control in the HTML form's tab order.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Wrap (see page 16808)	Specifies how the generated control treats lines that are longer than the display width.

99.1.40.2 TQueryTextArea.Create Constructor

Creates and initializes an instance of TQueryTextArea (see page 16795).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TQueryTextArea(TComponent * AOwner);
```

Description

Call Create to instantiate TQueryTextArea (see page 16795) at runtime. TQueryTextArea (see page 16795) components

added at design time using the Web Page editor are instantiated automatically.

99.1.40.3 TQueryTextArea.Destroy Destructor

Frees an instance of TQueryTextArea (see page 16795).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TQueryTextArea();
```

Description

Do not call Destroy directly in an application. Instead, use the Free method, which checks that the component reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

99.1.40.4 TQueryTextArea Methods

The methods of the TQueryTextArea class are listed here.

Protected Methods

	Name	Description
	AddAttributes (see page 16801)	This is AddAttributes, a member of class TQueryTextArea.
	ControlContent (see page 16801)	This is ControlContent, a member of class TQueryTextArea.
	GetText (see page 16802)	This is GetText, a member of class TQueryTextArea.
	SetLines (see page 16802)	This is SetLines, a member of class TQueryTextArea.
	SetText (see page 16803)	This is SetText, a member of class TQueryTextArea.

Public Methods

	Name	Description
	IsQueryField (see page 16802)	Indicates whether the component generates a control that can appear in a query form.

99.1.40.4.1 TQueryTextArea.AddAttributes Method

Pascal

```
procedure AddAttributes(var Attrs: string); override;
```

C++

```
virtual __fastcall AddAttributes(AnsiString Attrs);
```

Description

This is AddAttributes, a member of class TQueryTextArea.

99.1.40.4.2 TQueryTextArea.ControlContent Method

Pascal

```
function ControlContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall AnsiString ControlContent(TWebContentOptions Options);
```

Description

This is ControlContent, a member of class TQueryTextArea.

99.1.40.4.3 TQueryTextArea.GetText Method

Pascal

```
function GetText: string;
```

C++

```
__fastcall AnsiString GetText();
```

Description

This is GetText, a member of class TQueryTextArea.

99.1.40.4.4 TQueryTextArea.IsQueryField Method

Indicates whether the component generates a control that can appear in a query form.

Pascal

```
class function IsQueryField: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsQueryField();
```

Description

Call IsQueryField to determine whether a Web display component generates a control that represents a named value in a query form (TQueryForm (see page 16745)). IsQueryField returns true, because TQueryTextArea (see page 16795) is designed specifically for TQueryFieldGroup (see page 16736) to use.

See Also

IsColumn (see page 16895)

TQueryFieldGroup (see page 16736)

TQueryForm (see page 16745)

99.1.40.4.5 TQueryTextArea.SetLines Method

Pascal

```
procedure SetLines(const Value: TStrings);
```

C++

```
__fastcall SetLines(const TStrings Value);
```

Description

This is SetLines, a member of class TQueryTextArea.

99.1.40.4.6 TQueryTextArea.SetText Method

Pascal

```
procedure SetText(const Value: string);
```

C++

```
__fastcall SetText(const AnsiString Value);
```

Description

This is SetText, a member of class TQueryTextArea.

99.1.40.5 TQueryTextArea Properties

The properties of the TQueryTextArea class are listed here.

Published Properties

	Name	Description
	Caption (see page 16803)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16804)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16804)	Specifies where the label appears relative to the generated display control.
	Custom (see page 16804)	Represents a tag option and its value in the tag for the display control.
	DisplayRows (see page 16805)	Specifies the height of the generated control.
	DisplayWidth (see page 16805)	Specifies the horizontal size of the generated control.
	FieldName (see page 16805)	Specifies the name of the field whose value the generated control represents.
	Lines (see page 16806)	Specifies the initial value that appears in the multi-line edit control.
	ParamName (see page 16806)	Specifies the name of the parameter whose value the generated control represents.
	ReadOnly (see page 16807)	Indicates whether end users can use the generated control to set values.
	Style (see page 16807)	Adds style attributes to the generated display control.
	StyleRule (see page 16808)	This is StyleRule, a member of class TQueryTextArea.
	TabIndex (see page 16808)	Determines the position of the generated control in the HTML form's tab order.
	Wrap (see page 16808)	Specifies how the generated control treats lines that are longer than the display width.

99.1.40.5.1 TQueryTextArea.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 16897)

CaptionPosition (see page 16897)

99.1.40.5.2 TQueryTextArea.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 16897)

CaptionPosition (see page 16897)

HeadingAttributes (see page 16509)

99.1.40.5.3 TQueryTextArea.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 16897)

Caption (see page 16897)

99.1.40.5.4 TQueryTextArea.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 16899)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

99.1.40.5.5 TQueryTextArea.DisplayRows Property

Specifies the height of the generated control.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to indicate the desired height of the generated HTML control. DisplayRows specifies the number of lines of text that can be displayed in the control. This value is assigned to the ROWS option of the generated TEXTAREA tag.

See Also

DisplayWidth (see page 16979)

99.1.40.5.6 TQueryTextArea.DisplayWidth Property

Specifies the horizontal size of the generated control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to indicate the desired width of the generated HTML control. DisplayWidth specifies the number of characters that can be displayed in one line of the control. This value is assigned to the COLS option of the generated TEXTAREA tag.

See Also

DisplayRows (see page 16979)

99.1.40.5.7 TQueryTextArea.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 17033)

XMLDataSetField (see page 17033)

ParamName (see page 16910)

Setting Web Item Properties

99.1.40.5.8 TQueryTextArea.Lines Property

Specifies the initial value that appears in the multi-line edit control.

Pascal

```
property Lines: TStrings;
```

C++

```
__property TStrings Lines;
```

Description

Use Lines to initialize the value that appears in the generated display control.

Each string in the TStrings object designates a line in the text value. This may affect the formatting, depending on the value of Wrap (see page 16808).

See Also

FieldName (see page 16909)

ParamName (see page 16910)

Wrap (see page 16980)

99.1.40.5.9 TQueryTextArea.ParamName Property

Specifies the name of the parameter whose value the generated control represents.

Pascal

```
property ParamName: string;
```

C++

```
__property AnsiString ParamName;__property AnsiString ParamName;
```

Description

Use ParamName to bind the Web data input component to a parameter in the XML data packet or supply an application-specific name to the value for this control. ParamName is only used when the IsQueryField method returns true. By default, the controls in a query form represent parameter values. However, they can be used to represent any named value.

If ParamName identifies one of the parameters of an XML broker, the XML broker property of the parent Web item identifies the broker that uses the parameter.

See Also

FieldName (see page 16909)

IsQueryField (see page 16895)

Params (see page 26073)

xmlbroker (see page 17033)

Setting Web Item Properties

99.1.40.5.10 TQueryTextArea.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the generated control. This value determines whether there is a READONLY option added to the generated TEXTAREA tag.

99.1.40.5.11 TQueryTextArea.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 16898)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

Setting Web Item Properties

99.1.40.5.12 TQueryTextArea.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TQueryTextArea.

99.1.40.5.13 TQueryTextArea.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

99.1.40.5.14 TQueryTextArea.Wrap Property

Specifies how the generated control treats lines that are longer than the display width.

Pascal

```
property Wrap: TTextAreaWrap;
```

C++

```
__property TTextAreaWrap Wrap;
```

Description

Use Wrap to specify how the generated control formats lines that are longer than the number of characters specified by DisplayWidth. TWebTextAreaInput uses Wrap to add a WRAP option to the generated TEXTAREA tag.

Wrap can have any of the following values:

Value	WRAP option value	Meaning
wrOff	OFF	The control does not wrap long lines. Only newline characters that are part of the text string cause the lines in the control to wrap.
wrPhysical	PHYSICAL	The control formats its text value by wrapping long lines so that no text is hidden. The control inserts carriage returns into the text string.

wrVirtual	VIRTUAL	The control formats its text value by wrapping long lines so that no text is hidden but does not change the actual value of the text string.

See Also

DisplayWidth (see page 16979)

99.1.41 TResetQueryButton Class

TResetQueryButton generates an HTML control that backs out changes to the values on a query form.

Class Hierarchy

Pascal

```
TResetQueryButton = class(TQueryButton);
```

C++

```
class TResetQueryButton : public TQueryButton;
```

File

MidItems

Description

Add TResetQueryButton as the child of a TQueryButtons (see page 16730) component (or a layout group that is a child of a TQueryButtons (see page 16730) component) to enable the generated HTML form to undo edits made in the query form's display fields. The resulting HTML button's action backs out all edits to the values displayed on the query form.

Use the properties of TResetQueryButton to customize the appearance of the button.

See Also

TXMLBroker (see page 26049)

TQueryForm (see page 16745)

TQueryButtons (see page 16730)

TLayoutGroup (see page 16687)

99.1.41.1 TResetQueryButton Members

The following tables list the members exposed by TResetQueryButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 16723)	This is DefaultCaption, a member of class TQueryButton.

	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	InputType (see page 16723)	This is InputType, a member of class TQueryButton.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16868)	Returns the HTML that defines the button control.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16813)	This is EventContent, a member of class TResetQueryButton.
V	GetCaption (see page 16724)	Returns the value of the Caption property.
V	GetCaption (see page 16869)	Returns the value of the Caption (see page 16870) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetHTMLControlName (see page 16725)	This is GetHTMLControlName, a member of class TQueryButton.
	GetHTMLForm (see page 16725)	This is GetHTMLForm, a member of class TQueryButton.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetInputType (see page 16725)	This is GetInputType, a member of class TQueryButton.
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXMLComponent (see page 16725)	This is GetXMLComponent, a member of class TQueryButton.
V	ImplContent (see page 16725)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 16869)	Provides the underlying implementation for the Content (see page 16868) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16726)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	ParentXMLComponent (see page 16726)	This is ParentXMLComponent, a member of class TQueryButton.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	SetXMLComponent (see page 16727)	This is SetXMLComponent, a member of class TQueryButton.
	SetXMLUseParent (see page 16727)	This is SetXMLUseParent, a member of class TQueryButton.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16813)	Creates and initializes an instance of TResetQueryButton (see page 16809).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 16870)	Specifies the caption that appears on the button.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16871)	Represents a tag option and its value in the generated INPUT tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16871)	Adds style attributes to the generated button control.
	StyleRule (see page 16872)	Applies a style in the page producer's style sheet to the button.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Caption (see page 16727)	Specifies the caption that appears on the button.
	Custom (see page 16728)	Represents a tag option and its value in the generated INPUT tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16728)	Adds style attributes to the generated button control.
	StyleRule (see page 16729)	Applies a style in the page producer's style sheet to the button.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLComponent (see page 16729)	Associates the button with the query form whose display controls it influences.
	XMLUseParent (see page 16729)	Specifies whether the button is associated with its parent's query form.

99.1.41.2 TResetQueryButton.Create Constructor

Creates and initializes an instance of TResetQueryButton (see page 16809).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TResetQueryButton(TComponent * AOwner);
```

Description

Call Create to instantiate TResetQueryButton (see page 16809) at runtime. Buttons (see page 2289) added at design time using the Web Page editor are instantiated automatically.

99.1.41.3 TResetQueryButton Methods

The methods of the TResetQueryButton class are listed here.

Protected Methods

	Name	Description
V	EventContent (see page 16813)	This is EventContent, a member of class TResetQueryButton.

99.1.41.3.1 TResetQueryButton.EventContent Method

Pascal

```
function EventContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall AnsiString EventContent(TWebContentOptions Options);
```

Description

This is EventContent, a member of class TResetQueryButton.

99.1.42 TSelectOptionsColumn Class

TSelectOptionsColumn creates an HTML list control to represent a field value in a data grid.

Class Hierarchy

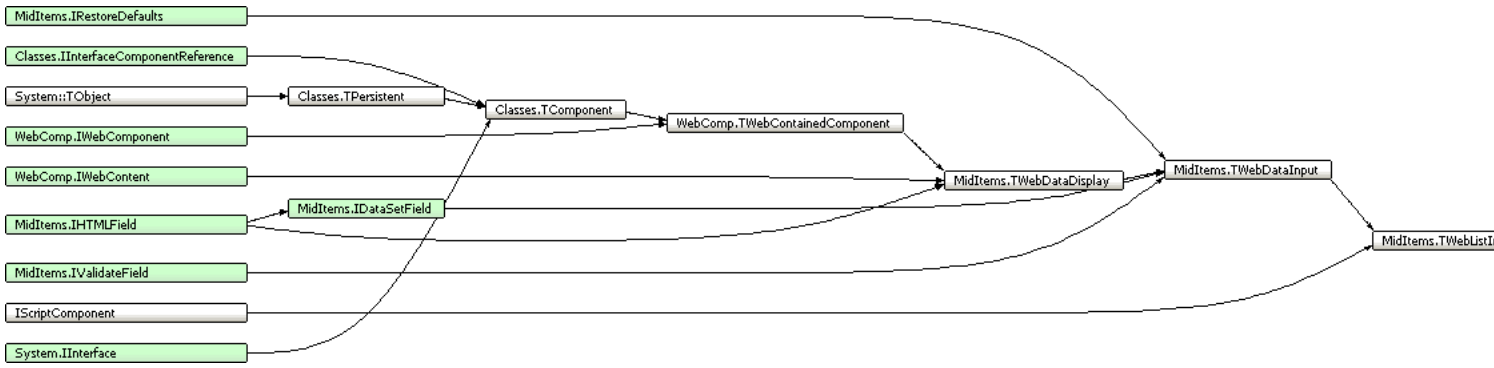

Pascal

```
TSelectOptionsColumn = class(TWebSelectOptionsInput);
```

C++

```
class TSelectOptionsColumn : public TWebSelectOptionsInput;
```

File

MidItems

Description

Add TSelectOptionsColumn as the child of a TDataGrid (see page 16560) component to add a list control as the display control for a column. TSelectOptionsColumn represents the value of a field in the TDataGrid (see page 16560)'s associated dataset.

Use the properties and methods of TSelectOptionsColumn to

Specify the field whose value the list box displays.

Specify the possible values for the field from which the user can choose.

Specify the caption that appears at the top of the column.

Customize the appearance of the list box.

Fit the list box into the tab order of the generated HTML form.

See Also

TDataGrid (see page 16560)

TXMLDisplayGroup (see page 17019)

TXMLDisplayGroup (see page 17019)

99.1.42.1 TSelectOptionsColumn Members

The following tables list the members exposed by TSelectOptionsColumn.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddAttributes (see page 16962)	This is AddAttributes, a member of class TWebSelectOptionsInput.
	AddElements (see page 16941)	This is AddElements, a member of class TWebListInput.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 16942)	This is ControlContent, a member of class TWebListInput.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	EventContent (see page 16905)	This is EventContent, a member of class TWebDataInput.
V	EventContent (see page 16962)	This is EventContent, a member of class TWebSelectOptionsInput.
V	FindAssociatedField (see page 16906)	This is FindAssociatedField, a member of class TWebDataInput.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
A	FormatInputs (see page 16942)	This is FormatInputs, a member of class TWebListInput.
V	FormatInputs (see page 16962)	This is FormatInputs, a member of class TWebSelectOptionsInput.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 16906)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetFieldName (see page 16906)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetParamName (see page 16906)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 16907)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
V	GetSelectIndex (see page 16962)	This is GetSelectIndex, a member of class TWebSelectOptionsInput.
	GetSubComponents (see page 16942)	This is GetSubComponents, a member of class TWebListInput.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplAddElements (see page 16942)	This is ImplAddElements, a member of class TWebListInput.
V	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
V	ImplGetFieldName (see page 16907)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 16907)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 16907)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplRestoreDefaults (see page 16908)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16943)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 16908)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16908)	This is RestoreFieldDefaults, a member of class TWebDataInput.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
V	SetCaption (see page 16908)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 16943)	This is SetDataSet, a member of class TWebListInput.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 16909)	This is SetFieldName, a member of class TWebDataInput.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItems (see page 16943)	This is SetItems, a member of class TWebListInput.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParamName (see page 16909)	This is SetParamName, a member of class TWebDataInput.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetValues (see page 16944)	This is SetValues, a member of class TWebListInput.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.

	ValidateField (see page 16909)	This is ValidateField, a member of class TWebDataInput.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	CaptionPosition (see page 16897)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 16910)	Specifies the name of the parameter whose value the generated control represents.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16889)	Creates and initializes an instance of TWebDataDisplay (see page 16884).
V	Create (see page 16940)	Creates and initializes an instance of TWebListInput (see page 16935).
V	Create (see page 16961)	Creates and initializes an instance of TWebSelectOptionsInput (see page 16956).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
V	Destroy (see page 16941)	Frees an instance of TWebListInput (see page 16935).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 17057)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRowSetFieldAttributes (see page 17058)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsColumn (see page 16819)	Indicates whether the component generates a control that can appear in the column of a data grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 17058)	This is SetFieldName, a member of class IDataSetField.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateField (see page 17076)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Caption (see page 16820)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16821)	Customizes the appearance of the generated control's label.
	Custom (see page 16821)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 16822)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayRows (see page 16822)	Specifies the number of rows that appear in the generated list box.
	FieldName (see page 16822)	Specifies the name of the field whose value the generated control represents.
	Items (see page 16823)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 16823)	Indicates which field in DataSet supplies the possible selections for the generated control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16824)	Adds style attributes to the generated display control.
	StyleRule (see page 16824)	This is StyleRule, a member of class TSelectOptionsColumn.
	TabIndex (see page 16825)	Determines the position of the generated control in the HTML form's tab order.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Values (see page 16825)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 16825)	Indicates which field in DataSet supplies the possible values for the generated control.

99.1.42.2 TSelectOptionsColumn Methods

The methods of the TSelectOptionsColumn class are listed here.

Public Methods

	Name	Description
V	IsColumn (see page 16819)	Indicates whether the component generates a control that can appear in the column of a data grid.

99.1.42.2.1 TSelectOptionsColumn.IsColumn Method

Indicates whether the component generates a control that can appear in the column of a data grid.

Pascal

```
class function IsColumn: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsColumn();
```

Description

Call IsColumn to determine whether a Web display component generates a control that represents a column in a data grid (TDataGrid (see page 16560)). IsColumn returns true, because TSelectOptionsColumn (see page 16814) is designed specifically for TDataGrid (see page 16560) to use.

See Also

IsColumn (see page 16895)

TDataGrid (see page 16560)

99.1.42.3 TSelectOptionsColumn Properties

The properties of the TSelectOptionsColumn class are listed here.

Published Properties

	Name	Description
	Caption (see page 16820)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16821)	Customizes the appearance of the generated control's label.
	Custom (see page 16821)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 16822)	Specifies a dataset that supplies the possible values listed in the generated control.
	DisplayRows (see page 16822)	Specifies the number of rows that appear in the generated list box.
	FieldName (see page 16822)	Specifies the name of the field whose value the generated control represents.
	Items (see page 16823)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 16823)	Indicates which field in DataSet supplies the possible selections for the generated control.
	Style (see page 16824)	Adds style attributes to the generated display control.
	StyleRule (see page 16824)	This is StyleRule, a member of class TSelectOptionsColumn.
	TabIndex (see page 16825)	Determines the position of the generated control in the HTML form's tab order.
	Values (see page 16825)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 16825)	Indicates which field in DataSet supplies the possible values for the generated control.

99.1.42.3.1 TSelectOptionsColumn.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 16897)

CaptionPosition (see page 16897)

99.1.42.3.2 TSelectOptionsColumn.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 16897)

CaptionPosition (see page 16897)

HeadingAttributes (see page 16509)

99.1.42.3.3 TSelectOptionsColumn.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 16899)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

99.1.42.3.4 TSelectOptionsColumn.DataSet Property

Specifies a dataset that supplies the possible values listed in the generated control.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Use DataSet to specify a dataset from which the component can extract the selections that appear to the user and the underlying values they represent. The ItemsField property specifies the field in this dataset that lists the selections that appear in the generated control. The ValuesField property specifies the field in this dataset that lists the corresponding values that are assigned to the control.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

ItemsField (see page 16945)

ValuesField (see page 16946)

Items (see page 16945)

Values (see page 16946)

99.1.42.3.5 TSelectOptionsColumn.DisplayRows Property

Specifies the number of rows that appear in the generated list box.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to specify how many rows appear in the generated list control. If DisplayRows is less than the number of items, the generated list displays only DisplayRows options at one time, but can scroll to display the hidden items. If DisplayRows is 1, the generated control becomes a drop-down list box.

See Also

ItemsField (see page 16945)

Items (see page 16945)

99.1.42.3.6 TSelectOptionsColumn.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 17033)

XMLDataSetField (see page 17033)

ParamName (see page 16910)

Setting Web Item Properties

99.1.42.3.7 TSelectOptionsColumn.Items Property

Lists the possible selections that appear in the generated control.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to supply a list of possible selections to the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the corresponding entry in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

ItemsField (see page 16945)

Values (see page 16946)

DataSet (see page 16944)

ValuesField (see page 16946)

99.1.42.3.8 TSelectOptionsColumn.ItemsField Property

Indicates which field in DataSet supplies the possible selections for the generated control.

Pascal

```
property ItemsField: string;
```

C++

```
__property AnsiString ItemsField;
```

Description

Use ItemsField to specify which field in the dataset specified by the DataSet property contains selections that appear in the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the field specified by the ValuesField property.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 16945)

Values (see page 16946)

DataSet (see page 16944)

ValuesField (see page 16946)

99.1.42.3.9 TSelectOptionsColumn.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 16898)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

Setting Web Item Properties

99.1.42.3.10 TSelectOptionsColumn.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TSelectOptionsColumn.

99.1.42.3.11 TSelectOptionsColumn.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

99.1.42.3.12 TSelectOptionsColumn.Values Property

Lists the possible values that can be assigned to the control.

Pascal

```
property Values: TStrings;
```

C++

```
__property TStrings Values;
```

Description

Use Values to supply a list of possible values for the generated control. When the end-user chooses one of the selections listed by the Items property, the control is assigned the corresponding value in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values properties, use the DataSet, ItemsField, and ValuesField properties.

See Also

Items (see page 16945)

ItemsField (see page 16945)

DataSet (see page 16944)

ValuesField (see page 16946)

99.1.42.3.13 TSelectOptionsColumn.ValuesField Property

Indicates which field in DataSet supplies the possible values for the generated control.

Pascal

```
property ValuesField: string;
```

C++

```
__property AnsiString ValuesField;
```

Description

Use ValuesField to specify which field in the dataset specified by the DataSet property contains values that can be assigned to the generated control. When the end-user selects the corresponding value form the ItemsField field, the control gets the value in the ValuesField field.

Note: If the list of possible values is not available in a dataset, use the Items and Values properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 16945)

Values (see page 16946)

DataSet (see page 16944)

ItemsField (see page 16945)

99.1.43 TStatusColumn Class

TStatusColumn generates an HTML control that indicates whether a record in a data grid has been modified by the end user.

Class Hierarchy

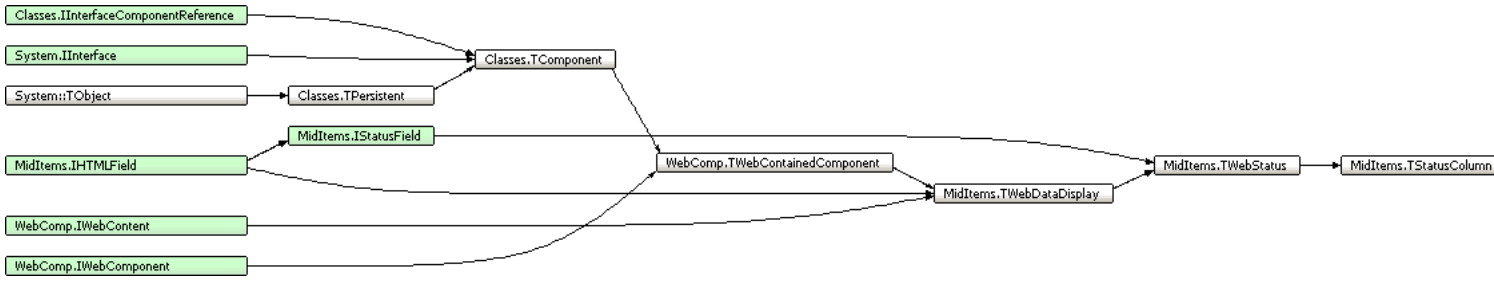

Pascal

```
TStatusColumn = class(TWebStatus);
```

C++

```
class TStatusColumn : public TWebStatus;
```

File

MidItems

Description

Add TStatusColumn as the child of a TDataGrid (see page 16560) component to provide feedback about whether records in the grid have been modified. The generated control displays an M if the user has edited a record, an I if the user has inserted the record, and blank if the user has made no changes to the record.

Use the properties and methods of TStatusColumn to

Specify the caption that appears at the top of the column.

Customize the appearance of the control.

Fit the control into the tab order of the generated HTML form.

See Also

TDataGrid (see page 16560)

TXMLDisplayGroup (see page 17019)

TXMLDisplayGroup (see page 17019)

99.1.43.1 TStatusColumn Members

The following tables list the members exposed by TStatusColumn.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 16969)	This is ControlContent, a member of class TWebStatus.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 16969)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 16969)	Returns the name that other components can use to refer to the generated display control.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.

V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	CaptionPosition (see page 16897)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16889)	Creates and initializes an instance of TWebDataDisplay (see page 16884).
V	Create (see page 16968)	Creates and initializes an instance of TWebStatus (see page 16963).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	Identifier (see page 16969)	Returns the name under which record status information is stored in the XML data packet.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsColumn (see page 16831)	Indicates whether the component generates a control that can appear in the column of a data grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	CaptionPosition (see page 16971)	Specifies where the label appears relative to the generated display control.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayWidth (see page 16971)	Specifies the width, in characters, of the generated HTML control.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	TabIndex (see page 16899)	Determines the position of the generated control in the HTML form's tab order.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Caption (see page 16831)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16832)	Customizes the appearance of the generated control's label.
	Custom (see page 16832)	Represents a tag option and its value in the tag for the display control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16832)	Adds style attributes to the generated display control.
	StyleRule (see page 16833)	This is StyleRule, a member of class TStatusColumn.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.43.2 TStatusColumn Methods

The methods of the TStatusColumn class are listed here.

Public Methods

	Name	Description
	IsColumn (see page 16831)	Indicates whether the component generates a control that can appear in the column of a data grid.

99.1.43.2.1 TStatusColumn.IsColumn Method

Indicates whether the component generates a control that can appear in the column of a data grid.

Pascal

```
class function IsColumn: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsColumn();
```

Description

Call IsColumn to determine whether a Web item generates a control that represents a column in a data grid (TDataGrid (see page 16560)). IsColumn returns true, because TStatusColumn (see page 16826) is designed specifically for TDataGrid (see page 16560) to use.

See Also

IsColumn (see page 16895)

TDataGrid (see page 16560)

99.1.43.3 TStatusColumn Properties

The properties of the TStatusColumn class are listed here.

Published Properties

	Name	Description
	Caption (see page 16831)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16832)	Customizes the appearance of the generated control's label.
	Custom (see page 16832)	Represents a tag option and its value in the tag for the display control.
	Style (see page 16832)	Adds style attributes to the generated display control.
	StyleRule (see page 16833)	This is StyleRule, a member of class TStatusColumn.

99.1.43.3.1 TStatusColumn.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 16897)

CaptionPosition (see page 16897)

99.1.43.3.2 TStatusColumn.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 16897)

CaptionPosition (see page 16897)

HeadingAttributes (see page 16509)

99.1.43.3.3 TStatusColumn.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 16899)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

99.1.43.3.4 TStatusColumn.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 16898)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

Setting Web Item Properties

99.1.43.3.5 TStatusColumn.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TStatusColumn.

99.1.44 TSubmitQueryButton Class

TSubmitQueryButton generates an HTML control that backs out changes to the values on a query form.

Class Hierarchy

Pascal

```
TSubmitQueryButton = class(TQueryButton);
```

C++

```
class TSubmitQueryButton : public TQueryButton;
```

File

MidItems

Description

Add TSubmitQueryButton as the child of a TQueryButtons (see page 16730) component (or a layout group that is a child of a TQueryButtons (see page 16730) component) to enable the generated HTML form to send the values entered in the query form back to the Web application. When the user presses the generated button, the values in the query form are sent to the Web application, using the path info specified by the query form's Action property.

Use the properties of TSubmitQueryButton to customize the appearance of the button.

See Also

TXMLBroker (see page 26049)

TQueryForm (see page 16745)

TQueryButtons (see page 16730)

TLayoutGroup (see page 16687)

TCustomQueryForm (see page 16545)

99.1.44.1 TSubmitQueryButton Members

The following tables list the members exposed by TSubmitQueryButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 16723)	This is DefaultCaption, a member of class TQueryButton.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	InputType (see page 16723)	This is InputType, a member of class TQueryButton.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16868)	Returns the HTML that defines the button control.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16838)	This is EventContent, a member of class TSubmitQueryButton.
V	GetCaption (see page 16724)	Returns the value of the Caption property.
V	GetCaption (see page 16869)	Returns the value of the Caption (see page 16870) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetHTMLControlName (see page 16725)	This is GetHTMLControlName, a member of class TQueryButton.
	GetHTMLForm (see page 16725)	This is GetHTMLForm, a member of class TQueryButton.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetInputType (see page 16725)	This is GetInputType, a member of class TQueryButton.
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXMLComponent (see page 16725)	This is GetXMLComponent, a member of class TQueryButton.
V	ImplContent (see page 16725)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 16869)	Provides the underlying implementation for the Content (see page 16868) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16726)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	ParentXMLComponent (see page 16726)	This is ParentXMLComponent, a member of class TQueryButton.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	SetXMLComponent (see page 16727)	This is SetXMLComponent, a member of class TQueryButton.
	SetXMLUseParent (see page 16727)	This is SetXMLUseParent, a member of class TQueryButton.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.

R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16838)	Creates and initializes an instance of TSubmitQueryButton (see page 16833).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 16870)	Specifies the caption that appears on the button.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16871)	Represents a tag option and its value in the generated INPUT tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16871)	Adds style attributes to the generated button control.
	StyleRule (see page 16872)	Applies a style in the page producer's style sheet to the button.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Caption (see page 16727)	Specifies the caption that appears on the button.
	Custom (see page 16728)	Represents a tag option and its value in the generated INPUT tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16728)	Adds style attributes to the generated button control.
	StyleRule (see page 16729)	Applies a style in the page producer's style sheet to the button.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLComponent (see page 16729)	Associates the button with the query form whose display controls it influences.
	XMLUseParent (see page 16729)	Specifies whether the button is associated with its parent's query form.

99.1.44.2 TSubmitQueryButton.Create Constructor

Creates and initializes an instance of TSubmitQueryButton (see page 16833).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSubmitQueryButton(TComponent * AOwner);
```

Description

Call Create to instantiate TSubmitQueryButton (see page 16833) at runtime. Buttons (see page 2289) added at design time using the Web Page editor are instantiated automatically.

99.1.44.3 TSubmitQueryButton Methods

The methods of the TSubmitQueryButton class are listed here.

Protected Methods

	Name	Description
	EventContent (see page 16838)	This is EventContent, a member of class TSubmitQueryButton.

99.1.44.3.1 TSubmitQueryButton.EventContent Method

Pascal

```
function EventContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall AnsiString EventContent(TWebContentOptions Options);
```

Description

This is EventContent, a member of class TSubmitQueryButton.

99.1.45 TTextAreaColumn Class

TTextAreaColumn creates a multi-line HTML input control to represent a field value in a data grid.

Class Hierarchy

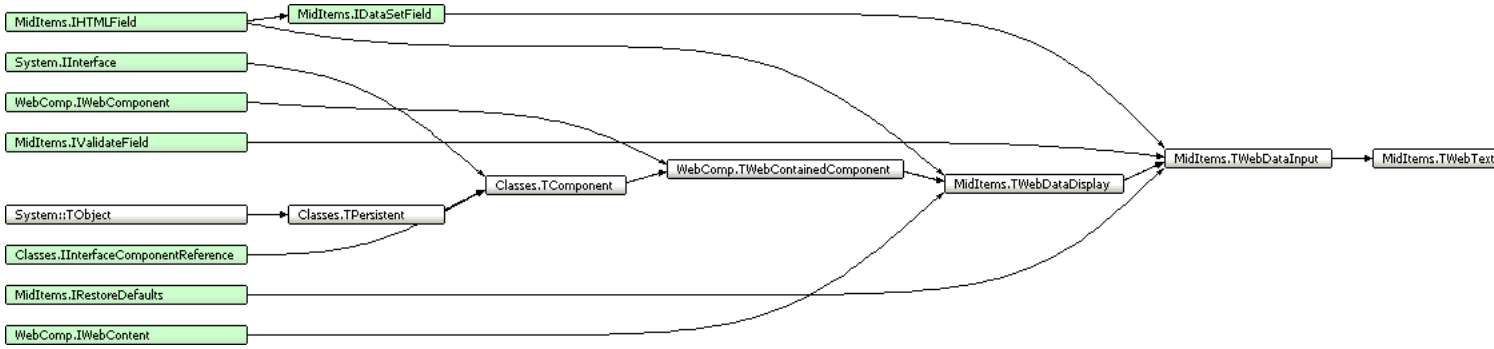

Pascal

```
TTextAreaColumn = class(TWebTextAreaInput);
```

C++

```
class TTextAreaColumn : public TWebTextAreaInput;
```

File

MidItems

Description

Add TTextAreaColumn as the child of a TDataGrid (see page 16560) component to add a multi-line edit control as the display control for a column. TTextAreaColumn represents the value of a field in the TDataGrid (see page 16560)'s associated dataset.

Use the properties and methods of TTextAreaColumn to

Specify the field whose value the control displays.

Specify the caption that appears at the top of the column.

Customize the appearance of the control.

Fit the control into the tab order of the generated HTML form.

See Also

IWebContent (see page 24792)

TDataGrid (see page 16560)

TFieldGroup (see page 16591)

TQueryFieldGroup (see page 16736)

TQueryTextArea (see page 16795)

99.1.45.1 TTextAreaColumn Members

The following tables list the members exposed by TTextAreaColumn.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddAttributes (see page 16977)	This is AddAttributes, a member of class TWebTextAreaInput.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 16978)	This is ControlContent, a member of class TWebTextAreaInput.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	EventContent (see page 16905)	This is EventContent, a member of class TWebDataInput.
V	EventContent (see page 16978)	This is EventContent, a member of class TWebTextAreaInput.
V	FindAssociatedField (see page 16906)	This is FindAssociatedField, a member of class TWebDataInput.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 16906)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetFieldName (see page 16906)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetParamName (see page 16906)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 16907)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
V	ImplGetFieldName (see page 16907)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 16907)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 16907)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplGetRowSetFieldAttributes (see page 16978)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextAreaInput.
V	ImplRestoreDefaults (see page 16908)	This is ImplRestoreDefaults, a member of class TWebDataInput.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 16908)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16908)	This is RestoreFieldDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16978)	This is RestoreFieldDefaults, a member of class TWebTextAreaInput.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
V	SetCaption (see page 16908)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 16909)	This is SetFieldName, a member of class TWebDataInput.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParamName (see page 16909)	This is SetParamName, a member of class TWebDataInput.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateField (see page 16909)	This is ValidateField, a member of class TWebDataInput.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	CaptionPosition (see page 16897)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 16910)	Specifies the name of the parameter whose value the generated control represents.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16889)	Creates and initializes an instance of TWebDataDisplay (see page 16884).
V	Create (see page 16977)	Creates and initializes an instance of TWebTextAreaInput (see page 16971).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 17057)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.

V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRowSetFieldAttributes (see page 17058)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsColumn (see page 16844)	Indicates whether the component generates a control that can appear in the column of a data grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 17058)	This is SetFieldName, a member of class IDataSetField.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateField (see page 17076)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Caption (see page 16845)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16845)	Customizes the appearance of the generated control's label.
	Custom (see page 16846)	Represents a tag option and its value in the tag for the display control.
	DisplayRows (see page 16846)	Specifies the height of the generated control.
	DisplayWidth (see page 16846)	Specifies the horizontal size of the generated control.
	FieldName (see page 16847)	Specifies the name of the field whose value the generated control represents.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ReadOnly (see page 16847)	Indicates whether end users can use the generated control to set values.
	Style (see page 16847)	Adds style attributes to the generated display control.
	StyleRule (see page 16848)	This is StyleRule, a member of class TTextAreaColumn.
	TabIndex (see page 16848)	Determines the position of the generated control in the HTML form's tab order.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Wrap (see page 16848)	Specifies how the generated control treats lines that are longer than the display width.

99.1.45.2 TTextAreaColumn Methods

The methods of the TTextAreaColumn class are listed here.

Public Methods

	Name	Description
	IsColumn (see page 16844)	Indicates whether the component generates a control that can appear in the column of a data grid.

99.1.45.2.1 TTextAreaColumn.IsColumn Method

Indicates whether the component generates a control that can appear in the column of a data grid.

Pascal

```
class function IsColumn: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsColumn();
```

Description

Call IsColumn to determine whether a Web display component generates a control that represents a column in a data grid (TDataGrid (see page 16560)). IsColumn returns true, because TTextAreaColumn (see page 16838) is designed specifically for TDataGrid (see page 16560) to use.

See Also

IsColumn (see page 16895)

TDataGrid (see page 16560)

99.1.45.3 TTextAreaColumn Properties

The properties of the TTextAreaColumn class are listed here.

Published Properties

	Name	Description
	Caption (see page 16845)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16845)	Customizes the appearance of the generated control's label.
	Custom (see page 16846)	Represents a tag option and its value in the tag for the display control.
	DisplayRows (see page 16846)	Specifies the height of the generated control.
	DisplayWidth (see page 16846)	Specifies the horizontal size of the generated control.
	FieldName (see page 16847)	Specifies the name of the field whose value the generated control represents.
	ReadOnly (see page 16847)	Indicates whether end users can use the generated control to set values.
	Style (see page 16847)	Adds style attributes to the generated display control.
	StyleRule (see page 16848)	This is StyleRule, a member of class TTextAreaColumn.
	TabIndex (see page 16848)	Determines the position of the generated control in the HTML form's tab order.
	Wrap (see page 16848)	Specifies how the generated control treats lines that are longer than the display width.

99.1.45.3.1 TTextAreaColumn.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 16897)

CaptionPosition (see page 16897)

99.1.45.3.2 TTextAreaColumn.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 16897)

CaptionPosition (see page 16897)

HeadingAttributes (see page 16509)

99.1.45.3.3 TTextAreaColumn.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 16899)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

99.1.45.3.4 TTextAreaColumn.DisplayRows Property

Specifies the height of the generated control.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to indicate the desired height of the generated HTML control. DisplayRows specifies the number of lines of text that can be displayed in the control. This value is assigned to the ROWS option of the generated TEXTAREA tag.

See Also

DisplayWidth (see page 16979)

99.1.45.3.5 TTextAreaColumn.DisplayWidth Property

Specifies the horizontal size of the generated control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to indicate the desired width of the generated HTML control. DisplayWidth specifies the number of characters that can be displayed in one line of the control. This value is assigned to the COLS option of the generated TEXTAREA tag.

See Also

DisplayRows (see page 16979)

99.1.45.3.6 TTextAreaColumn.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 17033)

XMLDataSetField (see page 17033)

ParamName (see page 16910)

Setting Web Item Properties

99.1.45.3.7 TTextAreaColumn.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the generated control. This value determines whether there is a READONLY option added to the generated TEXTAREA tag.

99.1.45.3.8 TTextAreaColumn.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 16898)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

Setting Web Item Properties

99.1.45.3.9 TTextAreaColumn.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TTextAreaColumn.

99.1.45.3.10 TTextAreaColumn.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

99.1.45.3.11 TTextAreaColumn.Wrap Property

Specifies how the generated control treats lines that are longer than the display width.

Pascal

```
property Wrap: TTextAreaWrap;
```

C++

```
__property TTextAreaWrap Wrap;
```

Description

Use Wrap to specify how the generated control formats lines that are longer than the number of characters specified by DisplayWidth. TWebTextAreaInput uses Wrap to add a WRAP option to the generated TEXTAREA tag.

Wrap can have any of the following values:

Value	WRAP option value	Meaning
wrOff	OFF	The control does not wrap long lines. Only newline characters that are part of the text string cause the lines in the control to wrap.
wrPhysical	PHYSICAL	The control formats its text value by wrapping long lines so that no text is hidden. The control inserts carriage returns into the text string.
wrVirtual	VIRTUAL	The control formats its text value by wrapping long lines so that no text is hidden but does not change the actual value of the text string.

See Also

DisplayWidth (see page 16979)

99.1.46 TTextColumn Class

TTextColumn creates a single-line HTML input control to represent a field value in a data grid.

Class Hierarchy

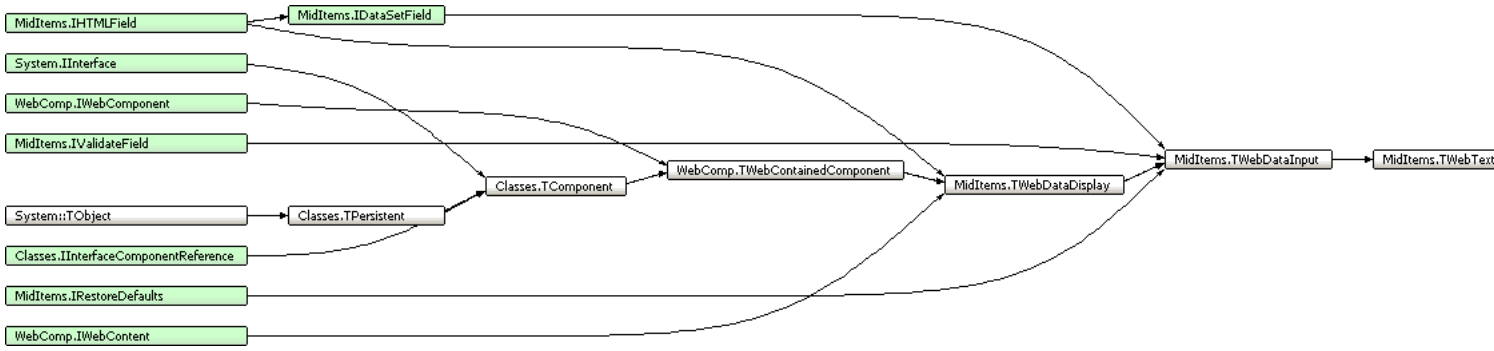

Pascal

```
TTextColumn = class(TWebTextInput);
```

C++

```
class TTextColumn : public TWebTextInput;
```

File

MidItems

Description

Add TTextColumn as the child of a TDataGrid (see page 16560) component to add an edit control as the display control for a

column. TTextColumn represents the value of a field in the TDataGrid (see page 16560)'s associated dataset.

Use the properties and methods of TTextColumn to

Specify the field whose value the control displays.

Specify the caption that appears at the top of the column.

Customize the appearance of the edit control.

Fit the control into the tab order of the generated HTML form.

See Also

TDataGrid (see page 16560)

TXMLDisplayGroup (see page 17019)

TXMLDisplayGroup (see page 17019)

99.1.46.1 TTextColumn Members

The following tables list the members exposed by TTextColumn.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddAttributes (see page 16986)	This is AddAttributes, a member of class TWebTextInput.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 16987)	This is ControlContent, a member of class TWebTextInput.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	EventContent (see page 16905)	This is EventContent, a member of class TWebDataInput.
V	FindAssociatedField (see page 16906)	This is FindAssociatedField, a member of class TWebDataInput.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 16906)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetFieldName (see page 16906)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.

	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetParamName (see page 16906)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 16907)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
V	ImplGetFieldName (see page 16907)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 16907)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 16907)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplGetRowSetFieldAttributes (see page 16987)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextInput.
V	ImplRestoreDefaults (see page 16908)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 16908)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16908)	This is RestoreFieldDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16987)	This is RestoreFieldDefaults, a member of class TWebTextInput.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
V	SetCaption (see page 16908)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 16909)	This is SetFieldName, a member of class TWebDataInput.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParamName (see page 16909)	This is SetParamName, a member of class TWebDataInput.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateField (see page 16909)	This is ValidateField, a member of class TWebDataInput.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	CaptionPosition (see page 16897)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 16910)	Specifies the name of the parameter whose value the generated control represents.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16889)	Creates and initializes an instance of TWebDataDisplay (see page 16884).
V	Create (see page 16986)	Creates and initializes an instance of TWebTextInput (see page 16980).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 17057)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRowSetFieldAttributes (see page 17058)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsColumn (see page 16854)	Indicates whether the component generates a control that can appear in the column of a data grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 17058)	This is SetFieldName, a member of class IDataSetField.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateField (see page 17076)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Caption (see page 16855)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16855)	Customizes the appearance of the generated control's label.
	Custom (see page 16856)	Represents a tag option and its value in the tag for the display control.
	DisplayWidth (see page 16856)	Specifies the horizontal size of the generated control.
	FieldName (see page 16857)	Specifies the name of the field whose value the generated control represents.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ReadOnly (see page 16857)	Indicates whether end users can use the generated control to set values.
	Style (see page 16857)	Adds style attributes to the generated display control.
	StyleRule (see page 16858)	This is StyleRule, a member of class TTextColumn.
	TabIndex (see page 16858)	Determines the position of the generated control in the HTML form's tab order.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.46.2 TTextColumn Methods

The methods of the TTextColumn class are listed here.

Public Methods

	Name	Description
V	IsColumn (see page 16854)	Indicates whether the component generates a control that can appear in the column of a data grid.

99.1.46.2.1 TTextColumn.IsColumn Method

Indicates whether the component generates a control that can appear in the column of a data grid.

Pascal

```
class function IsColumn: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsColumn();
```

Description

Call IsColumn to determine whether a Web display component generates a control that represents a column in a data grid (TDataGrid (see page 16560)). IsColumn returns true, because TTextColumn (see page 16849) is designed specifically for TDataGrid (see page 16560) to use.

See Also

IsColumn (see page 16895)

TDataGrid (see page 16560)

99.1.46.3 TTextColumn Properties

The properties of the TTextColumn class are listed here.

Published Properties

	Name	Description
	Caption (see page 16855)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16855)	Customizes the appearance of the generated control's label.
	Custom (see page 16856)	Represents a tag option and its value in the tag for the display control.
	DisplayWidth (see page 16856)	Specifies the horizontal size of the generated control.
	FieldName (see page 16857)	Specifies the name of the field whose value the generated control represents.
	ReadOnly (see page 16857)	Indicates whether end users can use the generated control to set values.
	Style (see page 16857)	Adds style attributes to the generated display control.
	StyleRule (see page 16858)	This is StyleRule, a member of class TTextColumn.
	TabIndex (see page 16858)	Determines the position of the generated control in the HTML form's tab order.

99.1.46.3.1 TTextColumn.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 16897)

CaptionPosition (see page 16897)

99.1.46.3.2 TTextColumn.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 16897)

CaptionPosition (see page 16897)

HeadingAttributes (see page 16509)

99.1.46.3.3 TTextColumn.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 16899)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

99.1.46.3.4 TTextColumn.DisplayWidth Property

Specifies the horizontal size of the generated control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to indicate the desired width of the generated HTML control. DisplayWidth specifies the number of characters that can be displayed in the control.

Note: DisplayWidth controls the size of the generated control, not the size of the text that users can enter. If the user enters a string longer than DisplayWidth characters, the text scrolls in the control.

99.1.46.3.5 TTextColumn.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 17033)

XMLDataSetField (see page 17033)

ParamName (see page 16910)

Setting Web Item Properties

99.1.46.3.6 TTextColumn.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the generated control.

99.1.46.3.7 TTextColumn.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 16898)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

Setting Web Item Properties

99.1.46.3.8 TTextColumn.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TTextColumn.

99.1.46.3.9 TTextColumn.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

99.1.47 TUndoButton Class

TUndoButton generates an HTML control that backs out changes to the current record.

Class Hierarchy

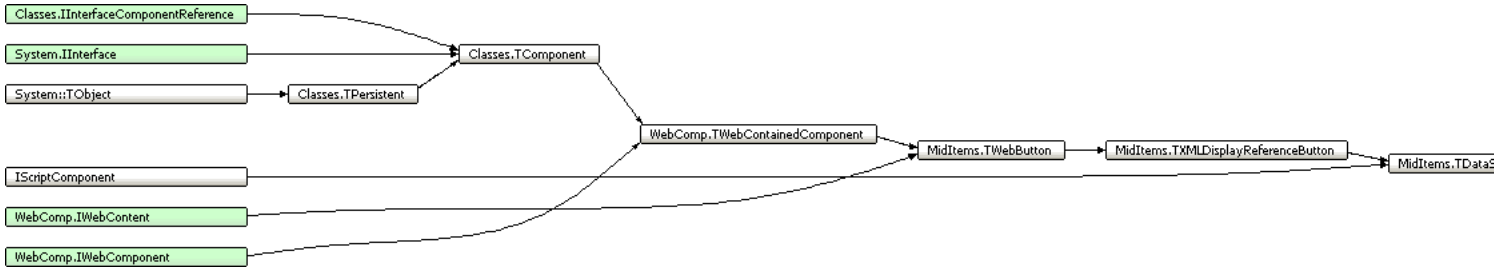

Pascal

```
TUndoButton = class(TDataSetButton);
```

C++

```
class TUndoButton : public TDataSetButton;
```

File

MidItems

Description

Add TUndoButton as the child of a TDataNavigator (see page 16569) component (or a layout group that is a child of a TDataNavigator (see page 16569) component) to enable the generated HTML form to undo edits made in an associated display control or controls. The resulting HTML button's action backs out all edits to the current record.

Use the properties of TUndoButton to customize the appearance of the button and associate it with a Web item that generates the display controls.

See Also

TXMLBroker (see page 26049)

TDataNavigator (see page 16569)

TLayoutGroup (see page 16687)

99.1.47.1 TUndoButton Members

The following tables list the members exposed by TUndoButton.

Protected Fields

	Name	Description
	DefaultCaption (see page 16581)	This is DefaultCaption, a member of class TDataSetButton.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	XMLMethodName (see page 16581)	This is XMLMethodName, a member of class TDataSetButton.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16582)	This is AddElements, a member of class TDataSetButton.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.

	Content (see page 16868)	Returns the HTML that defines the button control.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetCaption (see page 16582)	Returns the value of the Caption property.
V	GetCaption (see page 16869)	Returns the value of the Caption (see page 16870) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDisplayComponent (see page 17043)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 17043)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
V	GetMethodName (see page 16582)	This is GetMethodName, a member of class TDataSetButton.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16582)	This is GetSubComponents, a member of class TDataSetButton.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 17043)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	ImplAddElements (see page 16583)	This is ImplAddElements, a member of class TDataSetButton.
V	ImplContent (see page 16583)	Provides the underlying implementation for the Content method.
A	ImplContent (see page 16869)	Provides the underlying implementation for the Content (see page 16868) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 17044)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayComponent (see page 17044)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 17044)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.
R	XMLDisplay (see page 17045)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16863)	Creates and initializes an instance of TUndoButton (see page 16858).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 17042)	Frees an instance of TXMLDisplayReferenceButton (see page 17037).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 16870)	Specifies the caption that appears on the button.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16871)	Represents a tag option and its value in the generated INPUT tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.

	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16871)	Adds style attributes to the generated button control.
	StyleRule (see page 16872)	Applies a style in the page producer's style sheet to the button.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	XMLComponent (see page 17045)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 17045)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Caption (see page 16584)	Specifies the caption that appears on the button.
	Custom (see page 16584)	Represents a tag option and its value in the generated INPUT tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 16584)	Adds style attributes to the generated button control.
	StyleRule (see page 16585)	Applies a style in the page producer's style sheet to the button.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLComponent (see page 16585)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 16586)	Specifies whether the button is associated with its parent's display object.

99.1.47.2 TUndoButton.Create Constructor

Creates and initializes an instance of TUndoButton (see page 16858).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TUndoButton(TComponent * AOwner);
```

Description

Call Create to instantiate TUndoButton (see page 16858) at runtime. Buttons (see page 2289) added at design time using the Web Page editor are instantiated automatically.

99.1.48 TWebButton Class

TWebButton is the base class for Web items that create HTML button controls.

Class Hierarchy

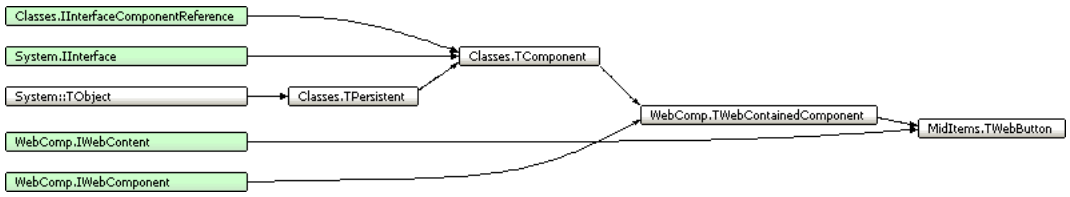

Pascal

```
TWebButton = class(TWebContainedComponent, IWebContent);
```

C++

```
class TWebButton : public TWebContainedComponent, public IWebContent;
```

File

MidItems

Description

Use TWebButton as a base class for components that generate the HTML statements defining a specialized button control in a WebSnap or InternetExpress application. TWebButton introduces the properties and methods for

Generating the HTML code to define the button control.

Defining a button caption.

Specifying tag attributes of the HTML button control, including those defined by a style sheet.

In order to allow other Web items to use the HTML generated by the button control, TWebButton implements the IWebContent interface.

Do not create instances of TWebButton. TWebButton relies on an abstract or, in C++ terminology, pure virtual method to generate the HTML statements defining a button control. Instead, create descendants of TWebButton that represent buttons with specific purposes.

See Also

TDeleteButton (see page 16586)

TFirstButton (see page 16651)

TInsertButton (see page 16677)

TLastButton (see page 16682)

TNextButton (see page 16694)

TNextPageButton (see page 16699)

TPostButton (see page 16704)

TPriorButton (see page 16709)

TPriorPageButton (see page 16713)

TUndoButton (see page 16858)

TResetQueryButton (see page 16809)

TSubmitQueryButton (see page 16833)

TApplyUpdatesButton (see page 16480)

IWebContent (see page 24792)

99.1.48.1 TWebButton Members

The following tables list the members exposed by TWebButton.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16868)	Returns the HTML that defines the button control.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetCaption (see page 16869)	Returns the value of the Caption (see page 16870) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
A	ImplContent (see page 16869)	Provides the underlying implementation for the Content (see page 16868) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 16868)	Creates and initializes an instance of TWebButton (see page 16863).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 16870)	Specifies the caption that appears on the button.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16871)	Represents a tag option and its value in the generated INPUT tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16871)	Adds style attributes to the generated button control.
	StyleRule (see page 16872)	Applies a style in the page producer's style sheet to the button.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.48.2 TWebButton.Create Constructor

Creates and initializes an instance of TWebButton (see page 16863).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebButton(TComponent * AOwner);
```

Description

Call Create to instantiate a Web button at runtime. Web buttons added at design time using the Web Page editor are instantiated automatically.

99.1.48.3 TWebButton Methods

The methods of the TWebButton class are listed here.

Protected Methods

	Name	Description
	Content (see page 16868)	Returns the HTML that defines the button control.
V	GetCaption (see page 16869)	Returns the value of the Caption (see page 16870) property.
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
A	ImplContent (see page 16869)	Provides the underlying implementation for the Content (see page 16868) method.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.

99.1.48.3.1 TWebButton.Content Method

Returns the HTML that defines the button control.

Pascal

```
function Content(Options: TWebContentOptions; Layout: TLayout): string;
```

C++

```
__fastcall AnsiString Content(TWebContentOptions Options, TLayout Layout);
```

Description

Content implements the IWebContent method of the same name. This method allows the Web button's parent to extract the HTML defining the button.

Content calls the virtual ImplContent (see page 16869) method to provide the content it returns. Each TWebButton (see page 16863) descendant overrides ImplContent (see page 16869) to generate the definition of its particular button control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web button should format the HTML control it generates. It implements the ILayoutWebContent interface.

Content returns a string of HTML defining the button control.

See Also

ImplContent (see page 16869)

IWebContent (see page 24792)

ILayoutWebContent (see page 24774)

99.1.48.3.2 TWebButton.GetCaption Method

Returns the value of the Caption (see page 16870) property.

Pascal

```
function GetCaption: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption (see page 16870) property.

See Also

Caption (see page 16870)

SetCaption (see page 16870)

99.1.48.3.3 TWebButton.GetLayoutAttributes Method

Returns the layout object that specifies how the generated control should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; virtual;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content (see page 16868) method) to take into account layout properties of the child component.

As implemented by TWebButton (see page 16863), GetLayoutAttributes returns nil (Delphi) or NULL (C++), because the layout of buttons does not make use of layout attributes.

See Also

Content (see page 16868)

99.1.48.3.4 TWebButton.ImplContent Method

Provides the underlying implementation for the Content (see page 16868) method.

Pascal

```
function ImplContent(Options: TWebContentOptions; Layout: TLayout): string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout Layout) = 0;
```

Description

The Content (see page 16868) method calls ImplContent to generate the HTML that defines the button control. TWebButton (see page 16863) introduces ImplContent as an abstract or, in C++ terminology, pure virtual method. TWebButton (see page 16863) descendants must override ImplContent to generate the definition of their particular button control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web button should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the button control.

See Also

Content (see page 16868)

ILayoutWebContent (see page 24774)

99.1.48.3.5 TWebButton.SetCaption Method

Sets the value of the Caption (see page 16870) property.

Pascal

```
procedure SetCaption(const AValue: string); virtual;
```

C++

```
virtual __fastcall SetCaption(const AnsiString AValue);
```

Description

SetCaption is the protected write implementation for the Caption (see page 16870) property.

See Also

Caption (see page 16870)

GetCaption (see page 16869)

99.1.48.4 TWebButton Properties

The properties of the TWebButton class are listed here.

Public Properties

	Name	Description
	Caption (see page 16870)	Specifies the caption that appears on the button.
	Custom (see page 16871)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 16871)	Adds style attributes to the generated button control.
	StyleRule (see page 16872)	Applies a style in the page producer's style sheet to the button.

99.1.48.4.1 TWebButton.Caption Property

Specifies the caption that appears on the button.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the caption that appears on the generated HTML button. The value of Caption is used as the value of a VALUE parameter on the HTML INPUT tag generated to implement the button.

See Also

Custom (see page 16871)

GetCaption (see page 16869)

SetCaption (see page 16870)

99.1.48.4.2 TWebButton.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to represent one or more options that qualify the INPUT tag generated by the Web button. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

```
ALIGN=TOP TABINDEX=2
```

See Also

Caption (see page 16870)

Style (see page 16871)

99.1.48.4.3 TWebButton.Style Property

Adds style attributes to the generated button control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to add a STYLE option that qualifies the INPUT tag generated by the Web button. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated INPUT tag as the value of a STYLE option.

See Also

Custom (see page 16871)

StyleRule (see page 16872)

99.1.48.4.4 TWebButton.StyleRule Property

Applies a style in the page producer's style sheet to the button.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated button control. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the button's Style (see page 16871) property.

See Also

Style (see page 16871)

Styles (see page 17127)

StylesFile (see page 17128)

Styles (see page 19830)

StylesFile (see page 19831)

99.1.49 TWebControlGroup Class

TWebControlGroup is the base class for Web items that arrange HTML controls.

Class Hierarchy

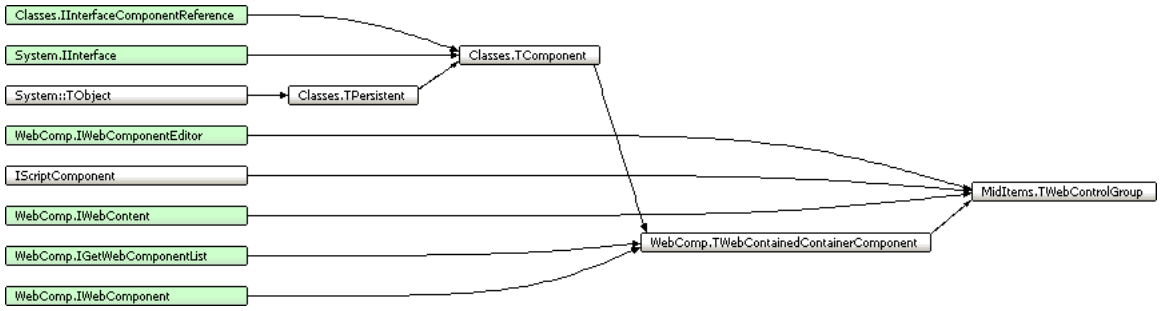

Pascal

```
TWebControlGroup = class(TWebContainedContainerComponent, IWebComponentEditor,
```

```
IScriptComponent, IWebContent);
```

C++

```
class TWebControlGroup : public TWebContainedContainerComponent, public IWebComponentEditor,
public IScriptComponent, public IWebContent;
```

File

MidItems

Description

Use TWebControlGroup as a base class for components that generate organized portions of the HTML document containing one or more related HTML controls. TWebControlGroup introduces the properties and methods for managing a collection of Web items that generate the HTML controls that make up the group.

In order to let other Web items to use the HTML that defines the organized collection of controls, TWebControlGroup implements the IWebContent interface.

Do not create instances of TWebControlGroup. TWebControlGroup manages the Web items that generate HTML defining controls, but does not combine the generated content into any content of its own. Instead, use a descendant of TWebControlGroup for a set of specialized controls. These descendants are listed in the See also list.

See Also

TDataNavigator (see page 16569)

TQueryButtons (see page 16730)

TDataGrid (see page 16560)

TFieldGroup (see page 16591)

TQueryFieldGroup (see page 16736)

IWebContent (see page 24792)

TAdapterFieldGroup (see page 25150)

TAdapterGrid (see page 25175)

TAdapterCommandColumn (see page 25052)

TAdapterForm (see page 25159)

99.1.49.1 TWebControlGroup Members

The following tables list the members exposed by TWebControlGroup.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.

	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
V	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
V	ImplCanAddClass (see page 16881)	Provides the underlying implementation for the CanAddClass (see page 16878) method.
V	ImplContent (see page 16882)	Provides the underlying implementation for the Content (see page 16879) method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.

V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 16877)	Creates and initializes an instance of TWebControlGroup (see page 16872).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 16877)	Frees an instance of TWebControlGroup (see page 16872).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.

	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.49.2 TWebControlGroup.Create Constructor

Creates and initializes an instance of TWebControlGroup (see page 16872).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebControlGroup(TComponent * AOwner);
```

Description

Call Create to instantiate TWebControlGroup (see page 16872) at runtime. TWebControlGroup (see page 16872) descendants added at design time are instantiated automatically.

99.1.49.3 TWebControlGroup.Destroy Destructor

Frees an instance of TWebControlGroup (see page 16872).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebControlGroup();
```

Description

Destroy removes the Web control group from its parent's list of components that generate HTML before freeing its own resources.

99.1.49.4 TWebControlGroup Methods

The methods of the TWebControlGroup class are listed here.

Protected Methods

	Name	Description
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.

	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
	ImplCanAddClass (see page 16881)	Provides the underlying implementation for the CanAddClass (see page 16878) method.
	ImplContent (see page 16882)	Provides the underlying implementation for the Content (see page 16879) method.
	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.

99.1.49.4.1 TWebControlGroup.AddElements Method

Used internally to generate any script declarations needed by the group of controls.

Pascal

```
procedure AddElements(AddIntf: IAddScriptElements);
```

C++

```
__fastcall AddElements(IAddScriptElements AddIntf);
```

Description

The page producer calls AddElements when it generates the block of declarations for the script language it uses. This method allows the generated group of controls to use classes and methods in the scripting language of the page producer.

AddElements calls on the virtual ImplAddElements (see page 16881) method, which uses the provided interface to return its declarations. Each TWebControlGroup (see page 16872) descendant overrides ImplAddElements (see page 16881) to add any declarations it needs.

See Also

ImplAddElements (see page 16881)

TAdapterPageProducer (see page 19802)

TInetXPageProducer (see page 17145)

99.1.49.4.2 TWebControlGroup.CanAddClass Method

Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.

Pascal

```
function CanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClass(TComponent * AParent, TClass AClass);
```

Description

The Web page editor calls CanAddClass to determine what Web items can be added to supply content for the TWebControlGroup (see page 16872) descendant. CanAddClass calls the virtual ImplCanAddClass (see page 16881) method to determine whether a class of Web item is valid.

AParent is the TWebControlGroup (see page 16872) descendant to whose WebFieldControls (see page 16884) property the Web item will be added.

AClass is the class of the proposed child Web item.

CanAddClass returns true if the TWebControlGroup (see page 16872) descendant can use content generated by a component of type AClass, false otherwise.

See Also

ImplCanAddClass (see page 16881)

WebFieldControls (see page 16884)

99.1.49.4.3 TWebControlGroup.ChangeFieldControls Method

Responds when the group of HTML controls changes.

Pascal

```
procedure ChangeFieldControls(Sender: TObject);
```

C++

```
__fastcall ChangeFieldControls(TObject * Sender);
```

Description

ChangeFieldControls is called automatically when the list of Web items that generate HTML controls in this group changes. (It is assigned as the OnChange event handler to the TWebComponentList object that implements the WebFieldControls (see page 16884) property.)

ChangeFieldControls sets the DefaultWebComponents (see page 16883) property to nil (Delphi) or NULL (C++) when an application explicitly assigns child items to the WebFieldControls (see page 16884) property.

See Also

OnChange (see page 24719)

WebFieldControls (see page 16884)

DefaultWebComponents (see page 16883)

DefaultWebComponents (see page 16883)

99.1.49.4.4 TWebControlGroup.Content Method

Returns the HTML that defines the group of HTML controls.

Pascal

```
function Content(Options: TWebContentOptions; ParentLayout: TLayout): string;
```

C++

```
__fastcall AnsiString Content(TWebContentOptions Options, TLayout ParentLayout);
```

Description

Content implements the IWebContent method of the same name. This method allows the TWebControlGroup (see page 16872) object's parent to extract the HTML that defines and lays out a set of controls.

Content calls the virtual ImplContent (see page 16882) method to provide the HTML it returns. Each TWebControlGroup (see page 16872) descendant overrides ImplContent (see page 16882) to generate its particular group of controls. The generated HTML can make use of any script declarations supplied by the AddElements (see page 16878) method.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the component should format the HTML it generates. It implements the ILayoutWebContent interface.

Content returns a string of HTML defining the group of controls.

See Also

ImplContent (see page 16882)

IWebContent (see page 24792)

AddElements (see page 16878)

ILayoutWebContent (see page 24774)

99.1.49.4.5 TWebControlGroup.GetDefaultWebComponents Method

Returns the value of the DefaultWebComponents (see page 16883) property.

Pascal

```
function GetDefaultWebComponents: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents returns the value of DefaultWebComponents (see page 16883). When no Web items have explicitly been added to the control group, this is a list of controls that appear by default. Once an application explicitly assigns controls to the WebFieldControls (see page 16884) property, GetDefaultWebComponents returns nil (Delphi) or NULL (C++).

See Also

WebFieldControls (see page 16884)

DefaultWebComponents (see page 16883)

SetDefaultWebComponents (see page 16883)

99.1.49.4.6 TWebControlGroup.GetLayoutAttributes Method

Returns an object that specifies how the generated group of controls should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; virtual;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content (see page 16879) method) to take into account layout properties of the child component.

As implemented by TWebControlGroup (see page 16872), GetLayoutAttributes returns nil (Delphi) or NULL (C++), because TWebControlGroup (see page 16872) descendants not make use of layout attributes.

See Also

Content (see page 16879)

99.1.49.4.7 TWebControlGroup.GetSubComponents Method

Returns the value of the WebFieldControls (see page 16884) property.

Pascal

```
function GetSubComponents: TObject;
```

C++

```
__fastcall TObject * GetSubComponents();
```

Description

GetSubComponents returns the TWebControlGroup (see page 16872)'s list of Web items that it uses to generate HTML content. This method is part of the mechanism that fits TWebControlGroup (see page 16872) into a hierarchy of Web items, each of which generates part of a larger HTML document.

See Also

WebFieldControls (see page 16884)

ImplGetSubComponents (see page 16882)

99.1.49.4.8 TWebControlGroup.ImplAddElements Method

Provides the underlying implementation for the AddElements (see page 16878) method.

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); virtual;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

The AddElements (see page 16878) method calls ImplAddElements to add any script declarations by calling the AddIntf interface. These declarations allow the generated display control to call script methods in response to user events by ensuring that the required methods are declared.

As implemented in TWebControlGroup (see page 16872), ImplAddElements does nothing. TWebControlGroup (see page 16872) descendants override ImplAddElements to add any required declarations.

See Also

AddElements (see page 16878)

99.1.49.4.9 TWebControlGroup.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass (see page 16878) method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass (see page 16878) method calls ImpleCanAddClass to determine whether the TWebControlGroup (see page 16872) descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup (see page 16872) instance to which the component will be added.

In TWebControlGroup (see page 16872), ImplCanAddClass always returns true. TWebControlGroup (see page 16872) descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass (see page 16878)

WebFieldControls (see page 16884)

99.1.49.4.10 TWebControlGroup.ImplContent Method

Provides the underlying implementation for the Content (see page 16879) method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content (see page 16879) method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup (see page 16872), ImplContent returns an empty string. TWebControlGroup (see page 16872) descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup (see page 16872) should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content (see page 16879)

AddElements (see page 16878)

ILayoutWebContent (see page 24774)

99.1.49.4.11 TWebControlGroup.ImplGetSubComponents Method

Provides the underlying implementation for the GetSubComponents (see page 16880) method.

Pascal

```
function ImplGetSubComponents: TObject; virtual;
```

C++

```
virtual __fastcall TObject * ImplGetSubComponents();
```

Description

The GetSubComponents (see page 16880) method calls ImplGetSubComponents to retrieve the value of WebFieldControls (see page 16884) as a TObject. TWebControlGroup (see page 16872) descendants override this method if the list is not simply the value of WebFieldControls (see page 16884). For example, some TWebControlGroup (see page 16872) descendants supply a default value (DefaultWebComponents (see page 16883)) when the WebFieldControls (see page 16884) property list is empty.

ImplGetSubComponents returns the TWebComponentList object that contains all subitems of the Web control group.

See Also

GetSubComponents (see page 16880)

DefaultWebComponents (see page 16883)

WebFieldControls (see page 16884)

99.1.49.4.12 TWebControlGroup.SetDefaultWebComponents Method

Sets the value of the DefaultWebComponents (see page 16883) property.

Pascal

```
procedure SetDefaultWebComponents(Value: TWebComponentList); override;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList Value);
```

Description

SetDefaultWebComponents assigns a list of Web items as the value of the DefaultWebComponents (see page 16883) property. This list specifies the controls that appear in the control group if the application does not explicitly assign any controls to the WebFieldControls (see page 16884) property.

See Also

DefaultWebComponents (see page 16883)

GetDefaultWebComponents (see page 16880)

WebFieldControls (see page 16884)

99.1.49.5 TWebControlGroup Properties

The properties of the TWebControlGroup class are listed here.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.

Public Properties

	Name	Description
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

99.1.49.5.1 TWebControlGroup.DefaultWebComponents Property

Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.

Pascal

```
property DefaultWebComponents: TWebComponentList;
```

C++

```
__property TWebComponentList DefaultWebComponents;
```

Description

Most TWebControlGroup (see page 16872) descendants have a default set of values for the WebFieldControls (see page 16884) property.

Note: For example, on Windows, TDataNavigator (see page 16569) has a default set of buttons and TFieldGroup (see page 16591) defaults to a control for each field in the associated data packet.

TWebControlGroup (see page 16872) descendants that supply default sets of Web items store these default sets as the value of the protected DefaultWebComponents property. Then, if the value of WebFieldControls (see page 16884) is not set, the TWebControlGroup (see page 16872) descendant uses DefaultWebComponents instead.

See Also

WebFieldControls (see page 16884)

99.1.49.5.2 TWebControlGroup.WebFieldControls Property

Lists the components that generate HTML controls for the Web control group.

Pascal

```
property WebFieldControls: TWebComponentList;
```

C++

```
__property TWebComponentList WebFieldControls;
```

Description

WebFieldControls lists the components that generate the related HTML controls that the Web control group assembles into a single unit. Each of these components supports the IWebContent interface, and some can contain their own list of Web items that help them generate their content.

The type of items that can be added to the WebFieldControls list are restricted by the CanAddClass (see page 16878) method. TWebControlGroup (see page 16872) does not restrict the items in the list at all, but descendant classes add restrictions to limit the list to related types of controls.

Note: For example, on Windows, TDataNavigator (see page 16569) limits the list to button generators that are descendants of TDataSetButton (see page 16576) or TXMLButton (see page 16989).

At design time, add components to the WebFieldControls property by bringing up the Web page editor for the containing page producer. Select the Web control group in the editor, and use the editor to add child controls.

See Also

TInetXPageProducer (see page 17145)

TAdapterPageProducer (see page 19802)

IWebContent (see page 24792)

CanAddClass (see page 16878)

99.1.50 TWebDataDisplay Class

TWebDataDisplay is the base class for Web items that create HTML display controls.

Class Hierarchy

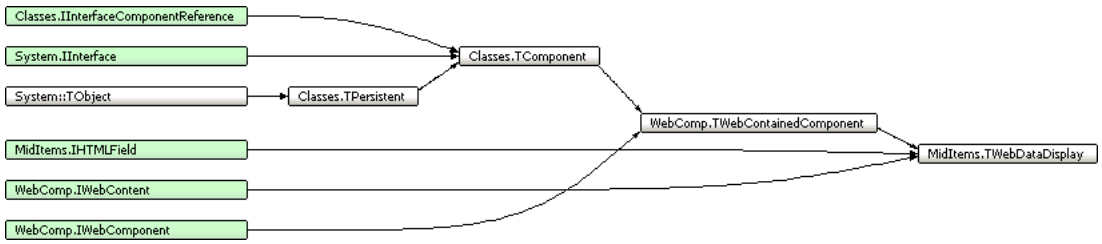

Pascal

```
TWebDataDisplay = class(TWebContainedComponent, IWebContent, IHTMLField);
```

C++

```
class TWebDataDisplay : public TWebContainedComponent, public IWebContent, public IHTMLField;
```

File

MidItems

Description

Use TWebDataDisplay as a base class for components that generate HTML controls which display information in a WebSnap or InternetExpress application. TWebDataDisplay descendants generate the individual controls that are organized by TAdapterGrid or TAdapterFieldGroup.

Note: On Windows, additional descendants, TDataGrid (see page 16560), TFieldGroup (see page 16591), or TQueryFieldGroup (see page 16736), can be used to generate the individual controls.

TWebDataDisplay introduces the properties and methods for

Generating a caption that describes the information displayed.

Customizing the appearance of the control that displays information.

Fitting the control into the tab order of the generated HTML form.

Fitting the component into a hierarchy of Web items that collectively generate the content of an HTML document.

In order to let other Web items to use the HTML control it generates, TWebDataDisplay implements the IWebContent interface.

Do not create instances of TWebDataDisplay. TWebDataDisplay relies on an abstract (pure virtual) method for generating the display control. Instead, use a descendant of TWebDataDisplay that generates an HTML control of the desired type.

See Also

IWebContent (see page 24792)

TAdapterGrid (see page 25175)

TAdapterFieldGroup (see page 25150)

TDataGrid (see page 16560)

TFieldGroup (see page 16591)

TQueryFieldGroup (see page 16736)

99.1.50.1 TWebDataDisplay Members

The following tables list the members exposed by TWebDataDisplay.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.

	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.

	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	CaptionPosition (see page 16897)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 16889)	Creates and initializes an instance of TWebDataDisplay (see page 16884).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 16897)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16897)	Customizes the appearance of the generated control's label.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.

	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16898)	Represents a tag option and its value in the tag for the display control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16899)	Adds style attributes to the generated display control.
	StyleRule (see page 16899)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 16899)	Determines the position of the generated control in the HTML form's tab order.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.50.2 TWebDataDisplay.Create Constructor

Creates and initializes an instance of TWebDataDisplay (see page 16884).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebDataDisplay(TComponent * AOwner);
```

Description

Call Create to instantiate TWebDataDisplay (see page 16884) at runtime. TWebDataDisplay (see page 16884) descendants added at design time are instantiated automatically.

99.1.50.3 TWebDataDisplay.Destroy Destructor

Frees an instance of TWebDataDisplay (see page 16884).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebDataDisplay();
```

Description

Do not call Destroy directly in an application. Instead, use the Free method, which checks that the component is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

99.1.50.4 TWebDataDisplay Methods

The methods of the TWebDataDisplay class are listed here.

Protected Methods

	Name	Description
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.

Public Methods

	Name	Description
V	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.

99.1.50.4.1 TWebDataDisplay.Content Method

Returns the HTML that defines the display control.

Pascal

```
function Content(Options: TWebContentOptions; ParentLayout: TLayout): string;
```

C++

```
__fastcall AnsiString Content(TWebContentOptions Options, TLayout ParentLayout);
```

Description

Content implements the IWebContent method of the same name. This method allows the TWebDataDisplay (see page 16884) component's parent to extract the HTML defining the display control.

Content calls the virtual ImplContent (see page 16894) method to provide the HTML content that it returns. Each TWebDataDisplay (see page 16884) descendant overrides ImplContent (see page 16894) to generate the definition of its particular control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the component should format the HTML control it generates. It supports the ILayoutWebContent interface.

Content returns a string of HTML defining the display control.

See Also

ImplContent (see page 16894)

IWebContent (see page 24792)

ILayoutWebContent (see page 24774)

99.1.50.4.2 TWebDataDisplay.ControlContent Method

Pascal

```
function ControlContent(Options: TWebContentOptions): string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString ControlContent(TWebContentOptions Options) = 0;
```

Description

This is ControlContent, a member of class TWebDataDisplay.

99.1.50.4.3 TWebDataDisplay.EventContent Method

Pascal

```
function EventContent(Options: TWebContentOptions): string; virtual;
```

C++

```
virtual __fastcall AnsiString EventContent(TWebContentOptions Options);
```

Description

This is EventContent, a member of class TWebDataDisplay.

99.1.50.4.4 TWebDataDisplay.FormatCaption Method

Pascal

```
function FormatCaption: string; virtual;
```

C++

```
virtual __fastcall AnsiString FormatCaption();
```

Description

This is FormatCaption, a member of class TWebDataDisplay.

99.1.50.4.5 TWebDataDisplay.GetCaption Method

Returns the value of the Caption (see page 16897) property.

Pascal

```
function GetCaption: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption (see page 16897) property.

See Also

Caption (see page 16897)

99.1.50.4.6 TWebDataDisplay.GetHTMLControlName Method

Returns the name that other components can use to refer to the generated display control.

Pascal

```
function GetHTMLControlName: string;
```

C++

```
__fastcall AnsiString GetHTMLControlName();
```

Description

GetHTMLControlName returns the name of the generated control. This value is assigned to a NAME option on the generated HTML tag.

GetHTMLControlName returns the value supplied by the ImplGetHTMLControlName (see page 16894) method. Unlike GetHTMLControlName, ImplGetHTMLControlName (see page 16894) is a virtual method that descendants can override to supply a name in class-specific ways.

See Also

ImplGetHTMLControlName (see page 16894)

99.1.50.4.7 TWebDataDisplay.GetHTMLForm Method

Used internally to retrieve information about the form in which the display control is defined.

Pascal

```
function GetHTMLForm: IHTMLForm;
```

C++

```
__fastcall IHTMLForm GetHTMLForm();
```

Description

TWebDataDisplay (see page 16884) and its descendants use GetHTMLForm to obtain information about the form in which the generated display control appears. This information is used to generate code for calling form methods in response to user events.

GetHTMLForm walks up the hierarchy of Web items (using the GetParentComponent method) until it finds a component that can supply an interface for the HTML form.

See Also

GetParentComponent

99.1.50.4.8 TWebDataDisplay.GetLayoutAttributes Method

Returns the value of the LayoutAttributes (see page 16898) property.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; virtual;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content (see page 16890) method) to take into account layout properties of the child component.

GetLayoutAttributes returns the value of the LayoutAttributes (see page 16898) property.

See Also

Content (see page 16890)

LayoutAttributes (see page 16898)

99.1.50.4.9 TWebDataDisplay.GetXmlDisplayName Method

Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.

Pascal

```
function GetXmlDisplayName: string;
```

C++

```
__fastcall AnsiString GetXmlDisplayName();
```

Description

GetXmlDisplayName is not available for cross-platform programming. It is introduced in TWebDataDisplay (see page 16884) for descendant classes that are only accessible on Windows.

On Windows, GetXmlDisplayName returns the name of the javascript object that groups the set of display controls which includes this TWebDataDisplay (see page 16884) descendant's generated control. The javascript object is generated by a TDataGrid (see page 16560) or TFieldGroup (see page 16591) component, and provides the display component's link to its XML data packet. Typically, the TDataGrid (see page 16560) or TFieldGroup (see page 16591) component is the parent of the TWebDataDisplay (see page 16884) component, but it may be higher in the parent hierarchy if the application uses layout groups.

See Also

TDataGrid (see page 16560)

TFieldGroup (see page 16591)

TQueryFieldGroup (see page 16736)

xmlbroker (see page 17033)

99.1.50.4.10 TWebDataDisplay.GetXmlRowSetName Method

Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.

Pascal

```
function GetXmlRowSetName: string;
```

C++

```
__fastcall AnsiString GetXmlRowSetName();
```

Description

GetXmlRowSetName is not available for cross-platform programming. It is introduced in TWebDataDisplay (see page 16884) for descendant classes that are only accessible on Windows.

On Windows, GetXmlRowSetName is used when generating HTML that accesses data from the XML data packet. It returns the name of the rowset that represents the data in the Web data display.

As implemented in TWebDataDisplay (see page 16884), GetXmlRowSetName walks up the parent hierarchy until it finds a Web item associated with a rowset, and then returns the name of that rowset.

See Also

GetXmlDisplayName (see page 16893)

99.1.50.4.11 TWebDataDisplay.ImplContent Method

Provides the underlying implementation for the Content (see page 16890) method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content (see page 16890) method calls ImplContent to generate the HTML that defines the display control and its label. ImplContent formats the caption and combines this HTML with the results of the ControlContent (see page 16891) method. ControlContent (see page 16891) is an abstract (pure virtual) method in TWebDataDisplay (see page 16884). TWebDataDisplay (see page 16884) descendants override it to generate the definition of their particular display control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebDataDisplay (see page 16884) should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the display control.

See Also

Content (see page 16890)

Caption (see page 16897)

ILayoutWebContent (see page 24774)

99.1.50.4.12 TWebDataDisplay.ImplGetHTMLControlName Method

Returns the name that other components can use to refer to the generated display control.

Pascal

```
function ImplGetHTMLControlName: string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplGetHTMLControlName();
```

Description

ImplGetHTMLControlName returns the name of the generated control. This value is assigned to a NAME option on the generated HTML tag.

ImplGetHTMLControlName returns the value of the Name property. TWebDataDisplay (see page 16884) descendants can override this method to return a more meaningful value such as the name of the field or parameter displayed in the control.

See Also

Name (see page 2805)

99.1.50.4.13 TWebDataDisplay.IsColumn Method

Indicates whether the component generates the controls that appear in the column of a grid.

Pascal

```
class function IsColumn: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsColumn();
```

Description

Call IsColumn to determine whether a TWebDataDisplay (see page 16884) descendant generates the controls that appear in a single column of an adapter grid (TAdapterGrid) or data grid (TDataGrid (see page 16560)). As implemented in TWebDataDisplay (see page 16884), IsColumn returns false. Descendant classes that generate controls for the columns of a grid override this method to return true.

See Also

IsQueryField (see page 16895)

TDataGrid (see page 16560)

TAdapterGrid (see page 25175)

99.1.50.4.14 TWebDataDisplay.IsQueryField Method

Indicates whether the component generates the controls that appear in a query form. Not used on Linux.

Pascal

```
class function IsQueryField: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsQueryField();
```

Description

IsQueryField is not available for cross-platform programming. It is introduced in TWebDataDisplay (see page 16884) for descendant classes that are only used on Windows.

On Windows, Call IsQueryField to determine whether a TWebDataDisplay (see page 16884) descendant generates the controls that represent a named value in a query form (TQueryForm (see page 16745)). As implemented in TWebDataDisplay (see page 16884), IsQueryField returns false. Descendant classes that generate controls for a TQueryFieldGroup (see page 16736) component override this method to return true.

See Also

IsColumn (see page 16895)

TQueryFieldGroup (see page 16736)

TQueryForm (see page 16745)

99.1.50.4.15 TWebDataDisplay.LabelContent Method

Pascal

```
function LabelContent: string; virtual;
```

C++

```
virtual __fastcall AnsiString LabelContent();
```

Description

This is LabelContent, a member of class TWebDataDisplay.

99.1.50.4.16 TWebDataDisplay.SetCaption Method

Sets the value of the Caption (see page 16897) property.

Pascal

```
procedure SetCaption(Value: string); virtual;
```

C++

```
virtual __fastcall SetCaption(AnsiString Value);
```

Description

SetCaption is the protected write implementation for the Caption (see page 16897) property.

See Also

Caption (see page 16897)

99.1.50.4.17 TWebDataDisplay.SetCaptionAttributes Method

Sets the value of the CaptionAttributes (see page 16897) property.

Pascal

```
procedure SetCaptionAttributes(const Value: TCaptionAttributes);
```

C++

```
__fastcall SetCaptionAttributes(const TCaptionAttributes Value);
```

Description

SetCaptionAttributes is the protected write implementation for the CaptionAttributes (see page 16897) property.

See Also

CaptionAttributes (see page 16897)

99.1.50.5 TWebDataDisplay Properties

The properties of the TWebDataDisplay class are listed here.

Protected Properties

	Name	Description
	CaptionPosition (see page 16897)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.

Public Properties

	Name	Description
	Caption (see page 16897)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16897)	Customizes the appearance of the generated control's label.

	Custom (see page 16898)	Represents a tag option and its value in the tag for the display control.
	Style (see page 16899)	Adds style attributes to the generated display control.
	StyleRule (see page 16899)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 16899)	Determines the position of the generated control in the HTML form's tab order.

99.1.50.5.1 TWebDataDisplay.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 16897)

CaptionPosition (see page 16897)

99.1.50.5.2 TWebDataDisplay.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption (see page 16897) property.

See Also

Caption (see page 16897)

CaptionPosition (see page 16897)

HeadingAttributes (see page 16509)

99.1.50.5.3 TWebDataDisplay.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 16897)

Caption (see page 16897)

99.1.50.5.4 TWebDataDisplay.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TWebDataDisplay (see page 16884) descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes (see page 16897) property instead.

See Also

Style (see page 16899)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

99.1.50.5.5 TWebDataDisplay.LayoutAttributes Property

Encodes attributes that describe how the generated control should look.

Pascal

```
property LayoutAttributes: TLayoutAttributes;
```

C++

```
__property TLayoutAttributes LayoutAttributes;
```

Description

LayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content (see page 16890) method) to take into account layout properties of the child component.

LayoutAttributes encodes the information in the CaptionPosition (see page 16897) and CaptionAttributes (see page 16897) properties. Some descendant classes add information from the Style (see page 16899) and StyleRule (see page 16899) properties as well.

See Also

GetLayoutAttributes (see page 16892)

CaptionPosition (see page 16897)

Content (see page 16890)

Style (see page 16899)

CaptionAttributes (see page 16897)

stylerule (see page 16899)

99.1.50.5.6 TWebDataDisplay.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes (see page 16897) property instead.

See Also

Custom (see page 16898)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

Setting Web Item Properties

99.1.50.5.7 TWebDataDisplay.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TWebDataDisplay.

99.1.50.5.8 TWebDataDisplay.TabIndex Property

Determines the position of the generated control in the HTML form's tab order.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

Use TabIndex to control the order that focus moves between controls when the end-user presses the Tab key. TabIndex specifies the position of the control in the tab order. If TabIndex is less than 0, the HTML tag's TABINDEX option is not set.

99.1.51 TWebDataInput Class

TWebDataInput is the base class for Web items that create HTML display controls for entering information.

Class Hierarchy

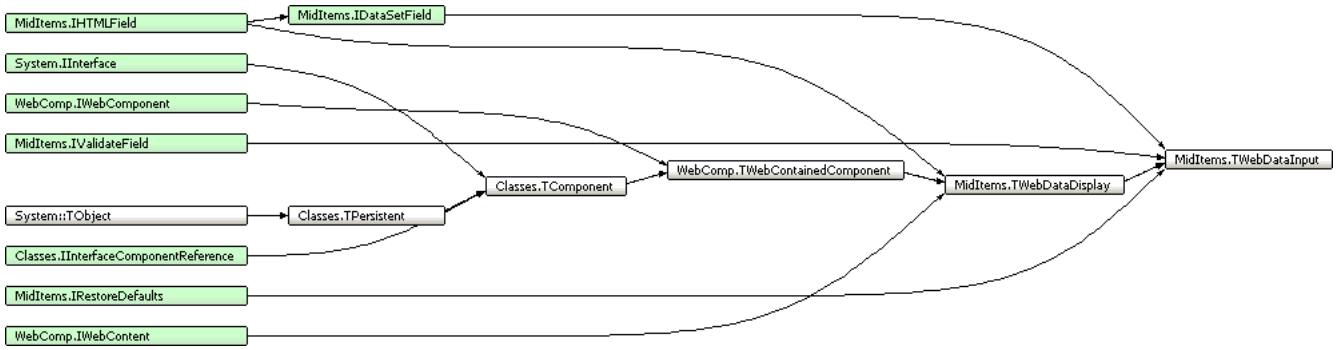

Pascal

```
TWebDataInput = class(TWebDataDisplay, IRestoreDefaults, IDataSetField, IValidateField);
```

C++

```
class TWebDataInput : public TWebDataDisplay, public IRestoreDefaults, public IDataSetField,
public IValidateField;
```

File

MidItems

Description

Use TWebDataInput as a base class for components that generate HTML controls which display information in an InternetExpress application and into which users can enter values. TWebDataInput descendants generate controls that are organized by TDataGrid (see page 16560), TFieldGroup (see page 16591), or TQueryFieldGroup (see page 16736).

TWebDataInput defines or inherits properties and methods for

Generating a caption that describes the information displayed.

Customizing the appearance of the control that displays information.

Specifying a field or parameter whose value the control displays.

Fitting the control into the tab order of the generated HTML form.

Fitting the component into a hierarchy of Web items that collectively generate the content of an HTML document.

In order to let other Web items to use the HTML control it generates, TWebDataInput inherits the IWebContent interface from TWebDataDisplay (see page 16884).

Do not create instances of TWebDataInput. TWebDataInput relies on an abstract (pure virtual) method for generating the display control. Instead, use a descendant of TWebDataInput that generates an HTML control of the desired type.

See Also

IWebContent (see page 24792)

TDataGrid (see page 16560)

TFieldGroup (see page 16591)

TQueryFieldGroup (see page 16736)

99.1.51.1 TWebDataInput Members

The following tables list the members exposed by TWebDataInput.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16905)	This is EventContent, a member of class TWebDataInput.
V	FindAssociatedField (see page 16906)	This is FindAssociatedField, a member of class TWebDataInput.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	GetCaption (see page 16906)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetFieldName (see page 16906)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetParamName (see page 16906)	This is GetParamName, a member of class TWebDataInput.

	GetRowSetFieldAttributes (see page 16907)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
V	ImplGetFieldName (see page 16907)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 16907)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 16907)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplRestoreDefaults (see page 16908)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 16908)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16908)	This is RestoreFieldDefaults, a member of class TWebDataInput.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16908)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 16909)	This is SetFieldName, a member of class TWebDataInput.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParamName (see page 16909)	This is SetParamName, a member of class TWebDataInput.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateField (see page 16909)	This is ValidateField, a member of class TWebDataInput.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	CaptionPosition (see page 16897)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 16910)	Specifies the name of the parameter whose value the generated control represents.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16889)	Creates and initializes an instance of TWebDataDisplay (see page 16884).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.

	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 16897)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16897)	Customizes the appearance of the generated control's label.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16898)	Represents a tag option and its value in the tag for the display control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.

	FieldName (see page 16909)	Specifies the name of the field whose value the generated control represents.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16899)	Adds style attributes to the generated display control.
	StyleRule (see page 16899)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 16899)	Determines the position of the generated control in the HTML form's tab order.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.51.2 TWebDataInput Methods

The methods of the TWebDataInput class are listed here.

Protected Methods

	Name	Description
V	EventContent (see page 16905)	This is EventContent, a member of class TWebDataInput.
V	FindAssociatedField (see page 16906)	This is FindAssociatedField, a member of class TWebDataInput.
V	GetCaption (see page 16906)	Returns the value of the Caption property.
	GetFieldName (see page 16906)	This is GetFieldName, a member of class TWebDataInput.
V	GetParamName (see page 16906)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 16907)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplGetFieldName (see page 16907)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 16907)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 16907)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
V	ImplRestoreDefaults (see page 16908)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	RestoreDefaults (see page 16908)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16908)	This is RestoreFieldDefaults, a member of class TWebDataInput.
V	SetCaption (see page 16908)	Sets the value of the Caption property.
	SetFieldName (see page 16909)	This is SetFieldName, a member of class TWebDataInput.
	SetParamName (see page 16909)	This is SetParamName, a member of class TWebDataInput.
	ValidateField (see page 16909)	This is ValidateField, a member of class TWebDataInput.

99.1.51.2.1 TWebDataInput.EventContent Method

Pascal

```
function EventContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall AnsiString EventContent(TWebContentOptions Options);
```

Description

This is EventContent, a member of class TWebDataInput.

99.1.51.2.2 TWebDataInput.FindAssociatedField Method

Pascal

```
function FindAssociatedField(DataSet: IInetXWebDataSet): TField; virtual;
```

C++

```
virtual __fastcall TField FindAssociatedField(IInetXWebDataSet DataSet);
```

Description

This is FindAssociatedField, a member of class TWebDataInput.

99.1.51.2.3 TWebDataInput.GetCaption Method

Returns the value of the Caption property.

Pascal

```
function GetCaption: string; override;
```

C++

```
virtual __fastcall AnsiString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption property.

See Also

Caption (see page 16897)

99.1.51.2.4 TWebDataInput.GetFieldName Method

Pascal

```
function GetFieldName: string;
```

C++

```
__fastcall AnsiString GetFieldName();
```

Description

This is GetFieldName, a member of class TWebDataInput.

99.1.51.2.5 TWebDataInput.GetParamName Method

Pascal

```
function GetParamName: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetParamName();
```

Description

This is GetParamName, a member of class TWebDataInput.

99.1.51.2.6 TWebDataInput.GetRowSetFieldAttributes Method

Pascal

```
function GetRowSetFieldAttributes(const FieldVarName: string): string;
```

C++

```
__fastcall AnsiString GetRowSetFieldAttributes(const AnsiString FieldVarName);
```

Description

This is GetRowSetFieldAttributes, a member of class TWebDataInput.

99.1.51.2.7 TWebDataInput.ImplGetFieldName Method

Pascal

```
function ImplGetFieldName: string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplGetFieldName();
```

Description

This is ImplGetFieldName, a member of class TWebDataInput.

99.1.51.2.8 TWebDataInput.ImplGetHTMLControlName Method

Returns the name that other components can use to refer to the generated display control.

Pascal

```
function ImplGetHTMLControlName: string; override;
```

C++

```
virtual __fastcall AnsiString ImplGetHTMLControlName();
```

Description

ImplGetHTMLControlName returns the name of the generated control. This value is assigned to a NAME option on the generated HTML tag.

ImplGetHTMLControlName returns the value of the Name property. TWebDataDisplay descendants can override this method to return a more meaningful value such as the name of the field or parameter displayed in the control.

See Also

Name (see page 2805)

99.1.51.2.9 TWebDataInput.ImplGetRowSetFieldAttributes Method

Pascal

```
function ImplGetRowSetFieldAttributes(const FieldVarName: string): string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplGetRowSetFieldAttributes(const AnsiString FieldVarName);
```

Description

This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.

99.1.51.2.10 TWebDataInput.ImplRestoreDefaults Method

Pascal

```
procedure ImplRestoreDefaults; virtual;
```

C++

```
virtual __fastcall ImplRestoreDefaults();
```

Description

This is ImplRestoreDefaults, a member of class TWebDataInput.

99.1.51.2.11 TWebDataInput.RestoreDefaults Method

Pascal

```
procedure RestoreDefaults;
```

C++

```
__fastcall RestoreDefaults();
```

Description

This is RestoreDefaults, a member of class TWebDataInput.

99.1.51.2.12 TWebDataInput.RestoreFieldDefaults Method

Pascal

```
procedure RestoreFieldDefaults(AField: TField); virtual;
```

C++

```
virtual __fastcall RestoreFieldDefaults(TField AField);
```

Description

This is RestoreFieldDefaults, a member of class TWebDataInput.

99.1.51.2.13 TWebDataInput.SetCaption Method

Sets the value of the Caption property.

Pascal

```
procedure SetCaption(Value: string); override;
```

C++

```
virtual __fastcall SetCaption(AnsiString Value);
```

Description

SetCaption is the protected write implementation for the Caption property.

See Also

Caption (see page 16897)

99.1.51.2.14 TWebDataInput.SetFieldName Method

Pascal

```
procedure SetFieldName(Value: string);
```

C++

```
__fastcall SetFieldName(AnsiString Value);
```

Description

This is SetFieldName, a member of class TWebDataInput.

99.1.51.2.15 TWebDataInput.SetParamName Method

Pascal

```
procedure SetParamName(Value: string);
```

C++

```
__fastcall SetParamName(AnsiString Value);
```

Description

This is SetParamName, a member of class TWebDataInput.

99.1.51.2.16 TWebDataInput.ValidateField Method

Pascal

```
function ValidateField(DataSet: IInetXWebDataSet; AddIntf: IAddScriptElements): Boolean;
```

C++

```
__fastcall Boolean ValidateField(IInetXWebDataSet DataSet, IAddScriptElements AddIntf);
```

Description

This is ValidateField, a member of class TWebDataInput.

99.1.51.3 TWebDataInput Properties

The properties of the TWebDataInput class are listed here.

Protected Properties

	Name	Description
	ParamName (see page 16910)	Specifies the name of the parameter whose value the generated control represents.

Public Properties

	Name	Description
	FieldName (see page 16909)	Specifies the name of the field whose value the generated control represents.

99.1.51.3.1 TWebDataInput.FieldName Property

Specifies the name of the field whose value the generated control represents.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;
```

Description

Use FieldName to bind the Web data input component to a field in the XML data packet. When the display control represents a field value, this is the field whose value is displayed and edited using the control. When the display control represents a parameter value, this is a field that is used to filter records that appear in data packets.

The dataset that contains the field is identified by the XMLBroker and XMLDataSetField properties of the Web item that acts as a parent to the Web data input component.

See Also

xmlbroker (see page 17033)

XMLDataSetField (see page 17033)

ParamName (see page 16910)

Setting Web Item Properties

99.1.51.3.2 TWebDataInput.ParamName Property

Specifies the name of the parameter whose value the generated control represents.

Pascal

```
property ParamName: string;
```

C++

```
__property AnsiString ParamName;
```

Description

Use ParamName to bind the Web data input component to a parameter in the XML data packet or supply an application-specific name to the value for this control. ParamName is only used when the IsQueryField (see page 16895) method returns true. By default, the controls in a query form represent parameter values. However, they can be used to represent any named value.

If ParamName identifies one of the parameters of an XML broker, the XML broker property of the parent Web item identifies the broker that uses the parameter.

See Also

FieldName (see page 16909)

IsQueryField (see page 16895)

Params (see page 26073)

xmlbroker (see page 17033)

Setting Web Item Properties

99.1.52 TWebDisplay Class

Class Hierarchy

System::TObject → MidItems.TWebDisplay

Pascal

```
TWebDisplay = class(TObject);
```

C++

```
class TWebDisplay : public TObject;
```

File

MidItems

Description

This is class MidItems.TWebDisplay.

99.1.52.1 TWebDisplay Members

The following tables list the members exposed by TWebDisplay.

Protected Methods

	Name	Description
	Changed (see page 16912)	This is Changed, a member of class TWebDisplay.
	ChangedDisplayComponent (see page 16913)	This is ChangedDisplayComponent, a member of class TWebDisplay.
	GetDisplayComponent (see page 16913)	This is GetDisplayComponent, a member of class TWebDisplay.
	SetDisplayComponent (see page 16913)	This is SetDisplayComponent, a member of class TWebDisplay.

Public Events

	Name	Description
	OnChange (see page 16914)	This is OnChange, a member of class TWebDisplay.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 16912)	This is Create, a member of class TWebDisplay.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Notification (see page 16913)	This is Notification, a member of class TWebDisplay.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	DisplayComponent (see page 16914)	This is DisplayComponent, a member of class TWebDisplay.
R	Parent (see page 16914)	This is Parent, a member of class TWebDisplay.

99.1.52.2 TWebDisplay.Create Constructor

Pascal

```
constructor Create(AParent: TComponent);
```

C++

```
__fastcall TWebDisplay(TComponent * AParent);
```

Description

This is Create, a member of class TWebDisplay.

99.1.52.3 TWebDisplay Methods

The methods of the TWebDisplay class are listed here.

Protected Methods

	Name	Description
	Changed (see page 16912)	This is Changed, a member of class TWebDisplay.
	ChangedDisplayComponent (see page 16913)	This is ChangedDisplayComponent, a member of class TWebDisplay.
	GetDisplayComponent (see page 16913)	This is GetDisplayComponent, a member of class TWebDisplay.
	SetDisplayComponent (see page 16913)	This is SetDisplayComponent, a member of class TWebDisplay.

Public Methods

	Name	Description
	Notification (see page 16913)	This is Notification, a member of class TWebDisplay.

99.1.52.3.1 TWebDisplay.Changed Method

Pascal

```
procedure Changed;
```

C++

```
__fastcall Changed();
```

Description

This is Changed, a member of class TWebDisplay.

99.1.52.3.2 TWebDisplay.ChangedDisplayComponent Method

Pascal

```
procedure ChangedDisplayComponent; virtual;
```

C++

```
virtual __fastcall ChangedDisplayComponent();
```

Description

This is ChangedDisplayComponent, a member of class TWebDisplay.

99.1.52.3.3 TWebDisplay.GetDisplayComponent Method

Pascal

```
function GetDisplayComponent: TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * GetDisplayComponent();
```

Description

This is GetDisplayComponent, a member of class TWebDisplay.

99.1.52.3.4 TWebDisplay.Notification Method

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); virtual;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

This is Notification, a member of class TWebDisplay.

99.1.52.3.5 TWebDisplay.SetDisplayComponent Method

Pascal

```
procedure SetDisplayComponent(const Value: TComponent);
```

C++

```
__fastcall SetDisplayComponent(const TComponent * Value);
```

Description

This is SetDisplayComponent, a member of class TWebDisplay.

99.1.52.4 TWebDisplay Properties

The properties of the TWebDisplay class are listed here.

Public Properties

	Name	Description
	DisplayComponent (see page 16914)	This is DisplayComponent, a member of class TWebDisplay.
R	Parent (see page 16914)	This is Parent, a member of class TWebDisplay.

99.1.52.4.1 TWebDisplay.DisplayComponent Property

Pascal

```
property DisplayComponent: TComponent;
```

C++

```
__property TComponent * DisplayComponent;
```

Description

This is DisplayComponent, a member of class TWebDisplay.

99.1.52.4.2 TWebDisplay.Parent Property

Pascal

```
property Parent: TComponent;
```

C++

```
__property TComponent * Parent;
```

Description

This is Parent, a member of class TWebDisplay.

99.1.52.5 TWebDisplay Events

The events of the TWebDisplay class are listed here.

Public Events

	Name	Description
	OnChange (see page 16914)	This is OnChange, a member of class TWebDisplay.

99.1.52.5.1 TWebDisplay.OnChange Event

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

This is OnChange, a member of class TWebDisplay.

99.1.53 TWebDisplayReferenceGroup Class

TWebDisplayReferenceGroup is the base class for Web items that arrange a set of controls that influence a control that displays data.

Class Hierarchy

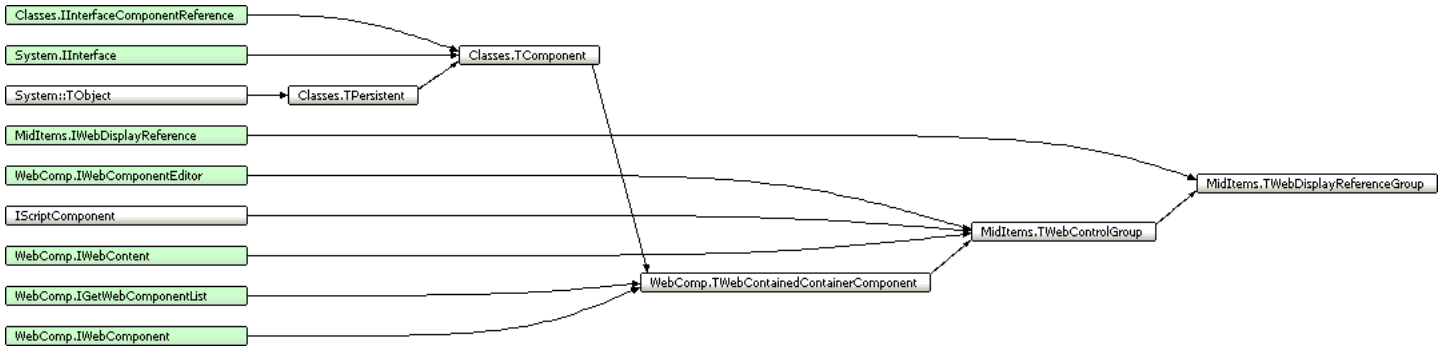

Pascal

```
TWebDisplayReferenceGroup = class(TWebControlGroup, IWebDisplayReference);
```

C++

```
class TWebDisplayReferenceGroup : public TWebControlGroup, public IWebDisplayReference;
```

File

MidItems

Description

Use TWebDisplayReferenceGroup as a base class for Web items that generate the HTML to arrange a collection of button controls. Each TWebDisplayReferenceGroup object has an associated Web item that generates controls that display data in the final HTML document. The controls arranged by TWebDisplayReferenceGroup manipulate the information the display controls show.

In order to let other Web items use the HTML that the TWebDisplayReferenceGroup object generates, TWebDisplayReferenceGroup inherits the IWebContent interface from TWebControlGroup (see page 16872).

Do not create instances of TWebDisplayReferenceGroup. TWebDisplayReferenceGroup manages the Web items that generate buttons and the association to a display generator, but does not produce any content. Instead, use a descendant of TWebDisplayReferenceGroup such as TDataNavigator (see page 16569) (in InternetExpress applications) or TAdapterCommandGroup (in WebSnap applications).

See Also

TDataNavigator (see page 16569)

IWebContent (see page 24792)

TAdapterCommandGroup (see page 25062)

99.1.53.1 TWebDisplayReferenceGroup Members

The following tables list the members exposed by TWebDisplayReferenceGroup.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	GetWebDisplayComponent (see page 16920)	This is GetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
V	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
V	ImplCanAddClass (see page 16881)	Provides the underlying implementation for the CanAddClass (see page 16878) method.
V	ImplContent (see page 16882)	Provides the underlying implementation for the Content (see page 16879) method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 16920)	Forwards notification messages to all owned components.

	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
V	SetWebDisplayComponent (see page 16921)	This is SetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebDisplay (see page 16921)	This is WebDisplay, a member of class TWebDisplayReferenceGroup.
	WebDisplayComponent (see page 16921)	
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16919)	Creates an instance of TWebDisplayReferenceGroup (see page 16915).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 16919)	Frees an instance of TWebDisplayReferenceGroup (see page 16915).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.53.2 TWebDisplayReferenceGroup.Create Constructor

Creates an instance of TWebDisplayReferenceGroup (see page 16915).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebDisplayReferenceGroup(TComponent * AOwner);
```

Description

Call TWebDisplayReferenceGroup (see page 16915) to instantiate TWebDisplayReferenceGroup (see page 16915) at runtime. TWebDisplayReferenceGroup (see page 16915) components added at design time are instantiated automatically.

99.1.53.3 TWebDisplayReferenceGroup.Destroy Destructor

Frees an instance of TWebDisplayReferenceGroup (see page 16915).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebDisplayReferenceGroup();
```

Description

Do not call Destroy directly. Instead, call Free, which checks whether the object is not nil before calling Destroy.

See Also

Free (see page 22300)

99.1.53.4 TWebDisplayReferenceGroup Methods

The methods of the TWebDisplayReferenceGroup class are listed here.

Protected Methods

	Name	Description
	GetWebDisplayComponent (see page 16920)	This is GetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
V	Notification (see page 16920)	Forwards notification messages to all owned components.
V	SetWebDisplayComponent (see page 16921)	This is SetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.

99.1.53.4.1 TWebDisplayReferenceGroup.GetWebDisplayComponent Method

Pascal

```
function GetWebDisplayComponent: TComponent;
```

C++

```
__fastcall TComponent * GetWebDisplayComponent();
```

Description

This is GetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.

99.1.53.4.2 TWebDisplayReferenceGroup.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

99.1.53.4.3 TWebDisplayReferenceGroup.SetWebDisplayComponent Method

Pascal

```
procedure SetWebDisplayComponent(const Value: TComponent); virtual;
```

C++

```
virtual __fastcall SetWebDisplayComponent(const TComponent * Value);
```

Description

This is SetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.

99.1.53.5 TWebDisplayReferenceGroup Properties

The properties of the TWebDisplayReferenceGroup class are listed here.

Protected Properties

	Name	Description
R	WebDisplay (see page 16921)	This is WebDisplay, a member of class TWebDisplayReferenceGroup.
	WebDisplayComponent (see page 16921)	

99.1.53.5.1 TWebDisplayReferenceGroup.WebDisplay Property

Pascal

```
property WebDisplay: TWebDisplay;
```

C++

```
__property TWebDisplay WebDisplay;
```

Description

This is WebDisplay, a member of class TWebDisplayReferenceGroup.

99.1.53.5.2 TWebDisplayReferenceGroup.WebDisplayComponent Property

Pascal

```
property WebDisplayComponent: TComponent;
```

C++

```
__property TComponent * WebDisplayComponent;
```

See Also

GetWebDisplayComponent (see page 17081)

99.1.54 TWebForm Class

TWebForm is the base class for Web items that create HTML forms in InternetExpress applications.

Class Hierarchy

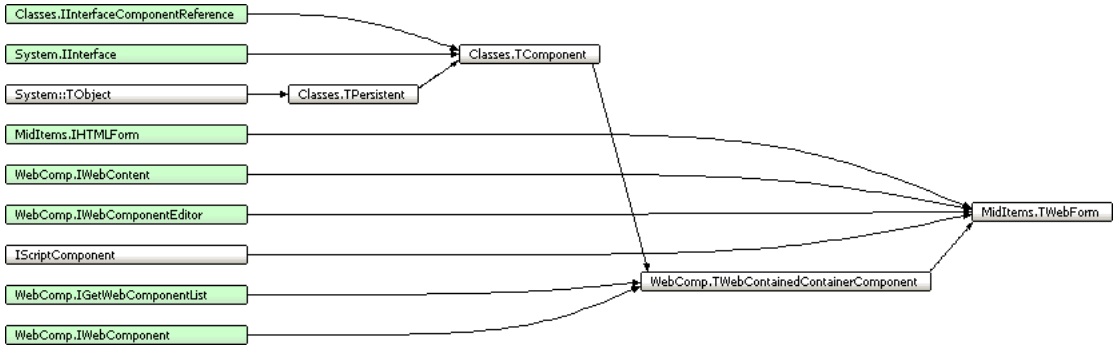

Pascal

```
TWebForm = class(TWebContainedContainerComponent, IWebContent, IWebComponentEditor,
IScriptComponent, IHTMLForm);
```

C++

```
class TWebForm : public TWebContainedContainerComponent, public IWebContent, public
IWebComponentEditor, public IScriptComponent, public IHTMLForm;
```

File

MidItems

Description

Use TWebForm as a base class for components that generate the HTML statements defining a form that is contained in an HTML document. The list of Web items that generate content for an InternetExpress page producer can only contain descendants of TWebForm.

In order to allow the InternetExpress page producer to use the HTML generated by the TWebForm descendant, TWebForm implements the IWebContent interface. In order to allow other Web items to generate javascript that refers to form elements, TWebForm implements the IHTMLForm (see page 17064) interface.

Do not create instances of TWebForm. TWebForm relies on an abstract or, in C++ terminology, pure virtual method to generate the HTML form. Instead, create descendants of TWebForm such a TDataForm (see page 16554) or TQueryForm (see page 16745).

See Also

TQueryForm (see page 16745)

TDataForm (see page 16554)

TBasePageItemsProducer (see page 18376)

TInetXPageProducer (see page 17145)

IWebContent (see page 24792)

IHTMLForm (see page 17064)

99.1.54.1 TWebForm Members

The following tables list the members exposed by TWebForm.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16927)	Used internally to generate any javascript declarations needed by the form.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16928)	Specifies whether a Web item of a specified type can be added to the WebComponents (see page 16934) property list.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16928)	Returns the HTML that defines the HTML form.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultWebComponents (see page 16929)	Returns nil (Delphi) or NULL (C++).
	GetHTMLFormName (see page 16929)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 16929)	Returns the name of the javascript variable that refers to the generated form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16930)	Returns the object that specifies how the generated form should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16930)	Returns the value of the WebComponents (see page 16934) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVisibleFields (see page 16930)	Returns the value of the VisibleFields (see page 16934) property.
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
V	ImplAddElements (see page 16931)	Provides the underlying implementation for the AddElements (see page 16927) method.
A	ImplCanAddClass (see page 16931)	Provides the underlying implementation for the CanAddClass (see page 16928) method.
A	ImplContent (see page 16932)	Provides the underlying implementation for the Content (see page 16928) method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.

V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16932)	Prevents an application from trying to assign a default set of Web items on the form.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	VisibleFields (see page 16934)	Lists the Web items whose generated content appears in the HTML form.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 16926)	Creates and initializes an instance of TWebForm (see page 16922).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 16926)	Frees an instance of TWebForm (see page 16922).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.

	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16933)	Represents a tag option and its value in the generated FORM tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLFormName (see page 17066)	Indicates the name of the generated HTML form.
R	HTMLFormVarName (see page 17066)	Indicates the name of the javascript variable that represents the HTML form.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16933)	Adds style attributes to the generated form.
	StyleRule (see page 16934)	Applies a style in the page producer's style sheet to the button.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebComponents (see page 16934)	Maintains a list of components that generate HTML content for the TWebForm (see page 16922) descendant.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.54.2 TWebForm.Create Constructor

Creates and initializes an instance of TWebForm (see page 16922).

Pascal

```
constructor Create(AComponent: TComponent); override;
```

C++

```
virtual __fastcall TWebForm(TComponent * AComponent);
```

Description

Call Create to instantiate a TWebForm (see page 16922) descendant at runtime. TWebForm (see page 16922) descendants added at design time using the Web Page editor are instantiated automatically.

99.1.54.3 TWebForm.Destroy Destructor

Frees an instance of TWebForm (see page 16922).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebForm();
```

Description

Destroy removes the form generator from its parent's list of components that generate HTML before freeing the TWebForm (see page 16922) instance.

99.1.54.4 TWebForm Methods

The methods of the TWebForm class are listed here.

Protected Methods

	Name	Description
	AddElements (see page 16927)	Used internally to generate any javascript declarations needed by the form.
	CanAddClass (see page 16928)	Specifies whether a Web item of a specified type can be added to the WebComponents (see page 16934) property list.
	Content (see page 16928)	Returns the HTML that defines the HTML form.
V	GetDefaultWebComponents (see page 16929)	Returns nil (Delphi) or NULL (C++).
	GetHTMLFormName (see page 16929)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 16929)	Returns the name of the javascript variable that refers to the generated form.
V	GetLayoutAttributes (see page 16930)	Returns the object that specifies how the generated form should look.
	GetSubComponents (see page 16930)	Returns the value of the WebComponents (see page 16934) property.
	GetVisibleFields (see page 16930)	Returns the value of the VisibleFields (see page 16934) property.
V	ImplAddElements (see page 16931)	Provides the underlying implementation for the AddElements (see page 16927) method.
A	ImplCanAddClass (see page 16931)	Provides the underlying implementation for the CanAddClass (see page 16928) method.
A	ImplContent (see page 16932)	Provides the underlying implementation for the Content (see page 16928) method.
V	SetDefaultWebComponents (see page 16932)	Prevents an application from trying to assign a default set of Web items on the form.

99.1.54.4.1 TWebForm.AddElements Method

Used internally to generate any javascript declarations needed by the form.

Pascal

```
procedure AddElements(AddIntf: IAddScriptElements);
```

C++

```
__fastcall AddElements(IAddScriptElements AddIntf);
```

Description

The InternetExpress page producer calls AddElements when it generates the block of javascript declarations in its HTML document. This method allows the generated form to use javascript classes and methods.

AddElements calls on the virtual ImplAddElements (see page 16931) method, which uses the provided interface to return javascript declarations. Each TWebForm (see page 16922) descendant overrides ImplAddElements (see page 16931) to add any declarations it needs.

See Also

ImplAddElements (see page 16931)

TInetXPageProducer (see page 17145)

99.1.54.4.2 TWebForm.CanAddClass Method

Specifies whether a Web item of a specified type can be added to the WebComponents (see page 16934) property list.

Pascal

```
function CanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClass(TComponent * AParent, TClass AClass);
```

Description

The Web page editor calls CanAddClass to determine what Web items can be added to supply content for the TWebForm (see page 16922) descendant. CanAddClass calls the virtual ImplCanAddClass (see page 16931) method to determine whether a class of Web item is valid.

AParent is the TWebForm (see page 16922) descendant to whose WebComponents (see page 16934) property the Web item will be added.

AClass is the class of the proposed child Web item.

CanAddClass returns true if the Web form can use content generated by a component of type AClass, false otherwise.

See Also

ImplCanAddClass (see page 16931)

WebComponents (see page 16934)

99.1.54.4.3 TWebForm.Content Method

Returns the HTML that defines the HTML form.

Pascal

```
function Content(Options: TWebContentOptions; Layout: TLayout): string;
```

C++

```
__fastcall AnsiString Content(TWebContentOptions Options, TLayout Layout);
```

Description

Content implements the IWebContent method of the same name. This method allows the page producer to extract the HTML defining the form.

Content calls the virtual ImplContent (see page 16932) method to provide the HTML form it returns. Each TWebForm (see page 16922) descendant overrides ImplContent (see page 16932) to generate its particular HTML form. The generated HTML can make use of any javascript declarations supplied by the AddElements (see page 16927) method.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the component should format the HTML form it generates. Typically, this parameter is nil (Delphi) or NULL (C++) for forms.

Content returns a string of HTML defining the form.

See Also

ImplContent (see page 16932)

IWebContent (see page 24792)

AddElements (see page 16927)

99.1.54.4.4 TWebForm.GetDefaultWebComponents Method

Returns nil (Delphi) or NULL (C++).

Pascal

```
function GetDefaultWebComponents: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents is implemented by all Web items that use sub Web items when generating HTML. (TWebForm (see page 16922) uses the subitems in the WebComponents (see page 16934) property). This method returns a list of Web items that are used by default when an application does not explicitly assign child Web items.

In TWebForm (see page 16922), GetDefaultWebComponents returns nil (Delphi) or NULL (C++) because Web forms do not any set of items that appear by default.

See Also

WebComponents (see page 16934)

GetDefaultWebComponents (see page 24744)

99.1.54.4.5 TWebForm.GetHTMLFormName Method

Returns the name of the generated HTML form.

Pascal

```
function GetHTMLFormName: string;
```

C++

```
__fastcall AnsiString GetHTMLFormName();
```

Description

TWebForm (see page 16922) uses GetHTMLFormName internally to specify the NAME option of the generated form and to declare a javascript variable that references the form.

GetHTMLFormName returns the value of the Name property.

See Also

GetHTMLFormVarName (see page 16929)

Name (see page 2805)

99.1.54.4.6 TWebForm.GetHTMLFormVarName Method

Returns the name of the javascript variable that refers to the generated form.

Pascal

```
function GetHTMLFormVarName: string;
```

C++

```
__fastcall AnsiString GetHTMLFormVarName();
```

Description

TWebForm (see page 16922) uses GetHTMLFormVarName internally to declare a javascript variable that references the form. GetHTMLFormVarName returns the value of the Name property.

See Also

GetHTMLFormName (see page 16929)

Name (see page 2805)

99.1.54.4.7 TWebForm.GetLayoutAttributes Method

Returns the object that specifies how the generated form should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; virtual;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content (see page 16928) method) to take into account layout properties of the child component.

As implemented by TWebForm (see page 16922), GetLayoutAttributes returns nil (Delphi) or NULL (C++), because the layout at the form level does not make use of layout attributes.

See Also

Content (see page 16928)

99.1.54.4.8 TWebForm.GetSubComponents Method

Returns the value of the WebComponents (see page 16934) property.

Pascal

```
function GetSubComponents: TObject;
```

C++

```
__fastcall TObject * GetSubComponents();
```

Description

GetSubComponents returns the TWebForm (see page 16922)'s list of Web items that it uses to generate HTML content. This method is part of the mechanism that fits TWebForm (see page 16922) into a hierarchy of Web items, each of which generates part of a larger HTML document.

See Also

WebComponents (see page 16934)

99.1.54.4.9 TWebForm.GetVisibleFields Method

Returns the value of the VisibleFields (see page 16934) property.

Pascal

```
function GetVisibleFields: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleFields();
```

Description

GetVisibleFields is the protected read implementation of the VisibleFields (see page 16934) property.

See Also

VisibleFields (see page 16934)

99.1.54.4.10 TWebForm.ImplAddElements Method

Provides the underlying implementation for the AddElements (see page 16927) method.

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); virtual;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

The AddElements (see page 16927) method calls ImplAddElements to add any javascript declarations by calling the AddIntf interface. These declarations allow the generated display control to call javascript methods in response to user events by ensuring that the required methods are declared.

As implemented in TWebForm (see page 16922), ImplAddElements does nothing. TWebForm (see page 16922) descendants can override ImplAddElements to add any required declarations.

See Also

AddElements (see page 16927)

99.1.54.4.11 TWebForm.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass (see page 16928) method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass) = 0;
```

Description

The CanAddClass (see page 16928) method calls ImpleCanAddClass to determine whether the TWebForm (see page 16922) descendant can add a component of type AClass to its WebComponents (see page 16934) property list. AParent is the TWebForm (see page 16922) instance to which the component will be added.

In TWebForm (see page 16922), ImplCanAddClass is abstract or, in C++ terminology, pure virtual. TWebForm (see page 16922) descendants must override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML forms.

See Also

CanAddClass (see page 16928)

WebComponents (see page 16934)

99.1.54.4.12 TWebForm.ImplContent Method

Provides the underlying implementation for the Content (see page 16928) method.

Pascal

```
function ImplContent(Options: TWebContentOptions; Layout: TLayout): string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout Layout) = 0;
```

Description

The Content (see page 16928) method calls ImplContent to generate the HTML form. ImplContent is anabstract or, in C++ terminology, pure virtual method in TWebForm (see page 16922). TWebForm (see page 16922) descendants override it to generate the HTML form.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebForm (see page 16922) should format the HTML control it generates. Typically, this parameter is nil (Delphi) or NULL (C++) for forms.

ImplContent returns an HTML form definition.

See Also

Content (see page 16928)

AddElements (see page 16927)

99.1.54.4.13 TWebForm.SetDefaultWebComponents Method

Prevents an application from trying to assign a default set of Web items on the form.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList); override;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

SetDefaultWebComponents is implemented by all Web items that use sub Web items when generating HTML. (TWebForm (see page 16922) uses the subitems in the WebComponents (see page 16934) property). This method assigns a list of Web items that are used by default when an application does not explicitly assign child Web items.

In TWebForm (see page 16922), SetDefaultWebComponents does nothing with the list supplied as the AList parameter. This is because web forms do not have default items.

See Also

WebComponents (see page 16934)

SetDefaultWebComponents (see page 24748)

99.1.54.5 TWebForm Properties

The properties of the TWebForm class are listed here.

Protected Properties

	Name	Description
R	VisibleFields (see page 16934)	Lists the Web items whose generated content appears in the HTML form.

Public Properties

	Name	Description
	Custom (see page 16933)	Represents a tag option and its value in the generated FORM tag.
	Style (see page 16933)	Adds style attributes to the generated form.
	StyleRule (see page 16934)	Applies a style in the page producer's style sheet to the button.
R	WebComponents (see page 16934)	Maintains a list of components that generate HTML content for the TWebForm (see page 16922) descendant.

99.1.54.5.1 TWebForm.Custom Property

Represents a tag option and its value in the generated FORM tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to represent one or more options that qualify the FORM tag generated by the TWebForm (see page 16922) descendant. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to specify that the form includes files with the text it submits and indicate that the page returned after submitting the form goes into a separate window, use a Custom value such as the following:

```
ENCTYPE="multipart/form-data" TARGET="ResultFrame"
```

See Also

Style (see page 16933)

99.1.54.5.2 TWebForm.Style Property

Adds style attributes to the generated form.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to add a STYLE option that qualifies the FORM tag generated by the TWebForm (see page 16922) descendant. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated FORM tag as the value of a STYLE option.

See Also

Custom (see page 16933)

StyleRule (see page 16934)

99.1.54.5.3 TWebForm.StyleRule Property

Applies a style in the page producer's style sheet to the button.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated form. The value of StyleRule specifies one of the user-defined style names defined in the InternetExpress page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the Style (see page 16933) property.

See Also

Style (see page 16933)

Styles (see page 17127)

StylesFile (see page 17128)

99.1.54.5.4 TWebForm.VisibleFields Property

Lists the Web items whose generated content appears in the HTML form.

Pascal

```
property VisibleFields: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleFields;
```

Description

VisibleFields is used internally when assembling the HTML form from the HTML generated by the form's Web items. For TWebForm (see page 16922), the value of VisibleFields is always the same as WebComponents (see page 16934).

Override GetVisibleFields (see page 16930) in a descendant class to provide a default set of that appear when the WebComponents (see page 16934) property is not set.

See Also

WebComponents (see page 16934)

99.1.54.5.5 TWebForm.WebComponents Property

Maintains a list of components that generate HTML content for the TWebForm (see page 16922) descendant.

Pascal

```
property WebComponents: TWebComponentList;
```

C++

```
__property TWebComponentList WebComponents;
```

Description

WebComponents lists the components that generate content for the Web form. Each of these components supports the IWebContent interface, and some can contain their own list of Web items that help them generate their content.

The type of items that can be added to the WebComponents list are restricted by the protected CanAddClass (see page 16928) method. Descendant classes override this method to indicate what types of items can be added to the list. For example, TQueryForm (see page 16745) limits the list to components that display or manipulate named values (TQueryButtons (see page 16730), TQueryFields) or layout groups that manage the layout of those items (TLayoutGroup (see page 16687)).

At design time, add components to the WebComponents property by bringing up the Web page editor for the containing TInetXPageProducer component. Select the TWebForm (see page 16922) descendant in the editor, and use the editor to add child controls.

See Also

IWebContent (see page 24792)

TInetXPageProducer (see page 17145)

CanAddClass (see page 16928)

99.1.55 TWebListInput Class

TWebListInput is the base class for Web items that create HTML display controls which present a fixed set of possible values.

Class Hierarchy

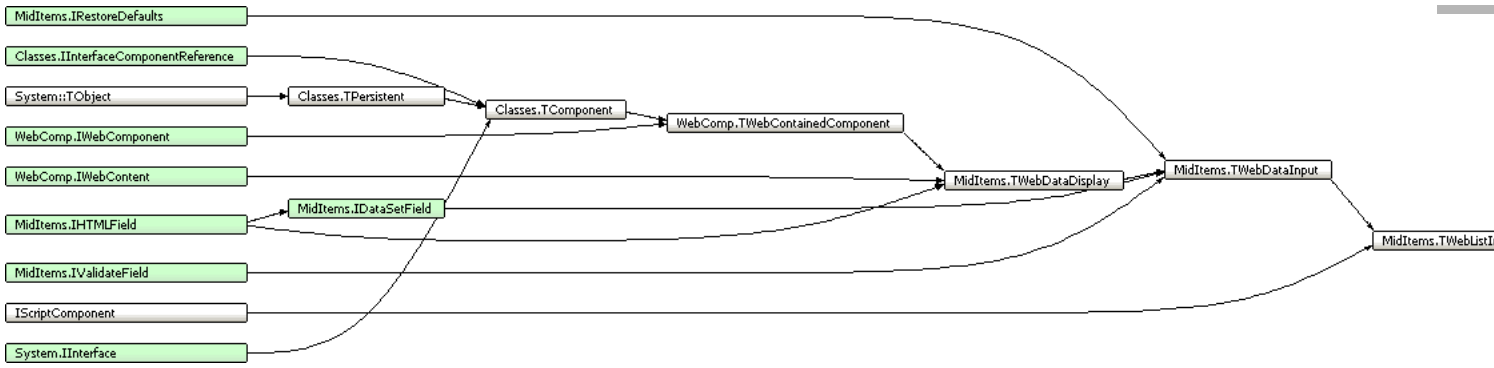

Pascal

```
TWebListInput = class(TWebDataInput, IScriptComponent);
```

C++

```
class TWebListInput : public TWebDataInput, public IScriptComponent;
```

File

MidItems

Description

Use TWebListInput as a base class for components that generate HTML controls for displaying a fixed set of possible values from which a user can choose. TWebListInput descendants generate controls that are organized by TDataGrid (see page 16560), TFieldGroup (see page 16591), or TQueryFieldGroup (see page 16736).

TWebListInput defines or inherits properties and methods for

Specifying a field or parameter whose value the control displays.

Specifying the possible values from which the user can choose.

Generating a caption that describes the information displayed.

Customizing the appearance of the control that displays information.

Fitting the control into the tab order of the generated HTML form.

Fitting the component into a hierarchy of Web items that collectively generate the content of an HTML document.

In order to let other Web items to use the HTML control it generates, TWebListInput inherits the IWebContent interface from TWebDataDisplay (see page 16884).

Do not create instances of TWebListInput. Instead, use a descendant of TWebListInput such as TFieldRadioGroup (see page 16599), TQueryRadioGroup (see page 16753), TFieldSelectOptions (see page 16612), TQuerySelectOptions (see page 16768), or TSelectOptionsColumn (see page 16814).

See Also

IWebContent (see page 24792)

TDataGrid (see page 16560)

TFieldGroup (see page 16591)

TQueryFieldGroup (see page 16736)

TFieldRadioGroup (see page 16599)

TQUeryRadioGroup

TFieldSelectOptions (see page 16612)

TQuerySelectOptions (see page 16768)

TSelectOptionsColumn (see page 16814)

99.1.55.1 TWebListInput Members

The following tables list the members exposed by TWebListInput.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16941)	This is AddElements, a member of class TWebListInput.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
V	ControlContent (see page 16942)	This is ControlContent, a member of class TWebListInput.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
	EventContent (see page 16905)	This is EventContent, a member of class TWebDataInput.
	FindAssociatedField (see page 16906)	This is FindAssociatedField, a member of class TWebDataInput.
	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
	FormatInputs (see page 16942)	This is FormatInputs, a member of class TWebListInput.
	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
	GetCaption (see page 16906)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetFieldName (see page 16906)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetParamName (see page 16906)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 16907)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetSubComponents (see page 16942)	This is GetSubComponents, a member of class TWebListInput.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplAddElements (see page 16942)	This is ImplAddElements, a member of class TWebListInput.
	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
	ImplGetFieldName (see page 16907)	This is ImplGetFieldName, a member of class TWebDataInput.
	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
	ImplGetHTMLControlName (see page 16907)	Returns the name that other components can use to refer to the generated display control.
	ImplGetRowSetFieldAttributes (see page 16907)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
	ImplRestoreDefaults (see page 16908)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 16943)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 16908)	This is RestoreDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 16908)	This is RestoreFieldDefaults, a member of class TWebDataInput.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.

V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
V	SetCaption (see page 16908)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 16943)	This is SetDataSet, a member of class TWebListInput.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 16909)	This is SetFieldName, a member of class TWebDataInput.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItems (see page 16943)	This is SetItems, a member of class TWebListInput.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParamName (see page 16909)	This is SetParamName, a member of class TWebDataInput.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetValues (see page 16944)	This is SetValues, a member of class TWebListInput.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateField (see page 16909)	This is ValidateField, a member of class TWebDataInput.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	CaptionPosition (see page 16897)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 16910)	Specifies the name of the parameter whose value the generated control represents.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16940)	Creates and initializes an instance of TWebListInput (see page 16935).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 16941)	Frees an instance of TWebListInput (see page 16935).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 17057)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRowSetFieldAttributes (see page 17058)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)

	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 17058)	This is SetFieldName, a member of class IDataSetField.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateField (see page 17076)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
	Caption (see page 16897)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16897)	Customizes the appearance of the generated control's label.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16898)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 16944)	Specifies a dataset that supplies the possible values listed in the generated control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	FieldName (see page 17058)	This is FieldName, a member of class IDataSetField.
	FieldName (see page 16909)	Specifies the name of the field whose value the generated control represents.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
	Items (see page 16945)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 16945)	Indicates which field in DataSet (see page 16944) supplies the possible selections for the generated control.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16899)	Adds style attributes to the generated display control.
	StyleRule (see page 16899)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 16899)	Determines the position of the generated control in the HTML form's tab order.
	Values (see page 16946)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 16946)	Indicates which field in DataSet (see page 16944) supplies the possible values for the generated control.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.55.2 TWebListInput.Create Constructor

Creates and initializes an instance of TWebListInput (see page 16935).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebListInput(TComponent * AOwner);
```

Description

Call Create to instantiate TWebListInput (see page 16935) at runtime. TWebListInput (see page 16935) descendants added at design time using the Web Page editor are instantiated automatically.

99.1.55.3 TWebListInput.Destroy Destructor

Frees an instance of TWebListInput (see page 16935).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebListInput();
```

Description

Do not call Destroy directly in an application. Instead, use the Free method, which checks that the component is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

99.1.55.4 TWebListInput Methods

The methods of the TWebListInput class are listed here.

Protected Methods

	Name	Description
	AddElements (see page 16941)	This is AddElements, a member of class TWebListInput.
V	ControlContent (see page 16942)	This is ControlContent, a member of class TWebListInput.
A	FormatInputs (see page 16942)	This is FormatInputs, a member of class TWebListInput.
	GetSubComponents (see page 16942)	This is GetSubComponents, a member of class TWebListInput.
V	ImplAddElements (see page 16942)	This is ImplAddElements, a member of class TWebListInput.
V	Notification (see page 16943)	Forwards notification messages to all owned components.
	SetDataSet (see page 16943)	This is SetDataSet, a member of class TWebListInput.
	SetItems (see page 16943)	This is SetItems, a member of class TWebListInput.
	SetValues (see page 16944)	This is SetValues, a member of class TWebListInput.

99.1.55.4.1 TWebListInput.AddElements Method

Pascal

```
procedure AddElements(AddIntf: IAddScriptElements);
```

C++

```
__fastcall AddElements(IAddScriptElements AddIntf);
```

Description

This is AddElements, a member of class TWebListInput.

99.1.55.4.2 TWebListInput.ControlContent Method

Pascal

```
function ControlContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall AnsiString ControlContent(TWebContentOptions Options);
```

Description

This is ControlContent, a member of class TWebListInput.

99.1.55.4.3 TWebListInput.FormatInputs Method

Pascal

```
function FormatInputs(ItemsStrings: TStrings; ValuesStrings: TStrings; Options:
TWebContentOptions): string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString FormatInputs(TStrings ItemsStrings, TStrings ValuesStrings,
TWebContentOptions Options) = 0;
```

Description

This is FormatInputs, a member of class TWebListInput.

99.1.55.4.4 TWebListInput.GetSubComponents Method

Pascal

```
function GetSubComponents: TObject;
```

C++

```
__fastcall TObject * GetSubComponents();
```

Description

This is GetSubComponents, a member of class TWebListInput.

99.1.55.4.5 TWebListInput.ImplAddElements Method

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); virtual;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

This is ImplAddElements, a member of class TWebListInput.

99.1.55.4.6 TWebListInput.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

99.1.55.4.7 TWebListInput.SetDataSet Method

Pascal

```
procedure SetDataSet(const Value: TDataSet);
```

C++

```
__fastcall SetDataSet(const TDataSet Value);
```

Description

This is SetDataSet, a member of class TWebListInput.

99.1.55.4.8 TWebListInput.SetItems Method

Pascal

```
procedure SetItems(const Value: TStrings);
```

C++

```
__fastcall SetItems(const TStrings Value);
```

Description

This is SetItems, a member of class TWebListInput.

99.1.55.4.9 TWebListInput.SetValues Method

Pascal

```
procedure SetValues(const Value: TStrings);
```

C++

```
__fastcall SetValues(const TStrings Value);
```

Description

This is SetValues, a member of class TWebListInput.

99.1.55.5 TWebListInput Properties

The properties of the TWebListInput class are listed here.

Public Properties

	Name	Description
	DataSet (see page 16944)	Specifies a dataset that supplies the possible values listed in the generated control.
	Items (see page 16945)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 16945)	Indicates which field in DataSet (see page 16944) supplies the possible selections for the generated control.
	Values (see page 16946)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 16946)	Indicates which field in DataSet (see page 16944) supplies the possible values for the generated control.

99.1.55.5.1 TWebListInput.DataSet Property

Specifies a dataset that supplies the possible values listed in the generated control.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Use DataSet to specify a dataset from which the component can extract the selections that appear to the user and the underlying values they represent. The ItemsField (see page 16945) property specifies the field in this dataset that lists the selections that appear in the generated control. The ValuesField (see page 16946) property specifies the field in this dataset that lists the corresponding values that are assigned to the control.

Note: If the list of possible values is not available in a dataset, use the Items (see page 16945) and Values (see page 16946) properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

ItemsField (see page 16945)

ValuesField (see page 16946)

Items (see page 16945)

Values (see page 16946)

99.1.55.5.2 TWebListInput.Items Property

Lists the possible selections that appear in the generated control.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to supply a list of possible selections to the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the corresponding entry in the Values (see page 16946) property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items and Values (see page 16946) properties, use the DataSet (see page 16944), ItemsField (see page 16945), and ValuesField (see page 16946) properties.

See Also

ItemsField (see page 16945)

Values (see page 16946)

DataSet (see page 16944)

ValuesField (see page 16946)

99.1.55.5.3 TWebListInput.ItemsField Property

Indicates which field in DataSet (see page 16944) supplies the possible selections for the generated control.

Pascal

```
property ItemsField: string;
```

C++

```
__property AnsiString ItemsField;
```

Description

Use ItemsField to specify which field in the dataset specified by the DataSet (see page 16944) property contains selections that appear in the generated control. The end-user can choose from these selections to assign a value to the generated control. The value that is assigned comes from the field specified by the ValuesField (see page 16946) property.

Note: If the list of possible values is not available in a dataset, use the Items (see page 16945) and Values (see page 16946) properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 16945)

Values (see page 16946)

DataSet (see page 16944)

ValuesField (see page 16946)

99.1.55.5.4 TWebListInput.Values Property

Lists the possible values that can be assigned to the control.

Pascal

```
property Values: TStrings;
```

C++

```
__property TStrings Values;
```

Description

Use Values to supply a list of possible values for the generated control. When the end-user chooses one of the selections listed by the Items (see page 16945) property, the control is assigned the corresponding value in the Values property.

Note: To use a dataset to supply items and values, rather than explicitly listing them in the Items (see page 16945) and Values properties, use the DataSet (see page 16944), ItemsField (see page 16945), and ValuesField (see page 16946) properties.

See Also

Items (see page 16945)

ItemsField (see page 16945)

DataSet (see page 16944)

ValuesField (see page 16946)

99.1.55.5.5 TWebListInput.ValuesField Property

Indicates which field in DataSet (see page 16944) supplies the possible values for the generated control.

Pascal

```
property ValuesField: string;
```

C++

```
__property AnsiString ValuesField;
```

Description

Use ValuesField to specify which field in the dataset specified by the DataSet (see page 16944) property contains values that can be assigned to the generated control. When the end-user selects the corresponding value form the ItemsField (see page 16945) field, the control gets the value in the ValuesField field.

Note: If the list of possible values is not available in a dataset, use the Items (see page 16945) and Values (see page 16946) properties instead to list the selections that appear to the user and the underlying values they represent.

See Also

Items (see page 16945)

Values (see page 16946)

DataSet (see page 16944)

ItemsField (see page 16945)

99.1.56 TWebRadioGroupInput Class

TWebRadioGroupInput is the base class for Web items that create HTML radio group controls.

Class Hierarchy

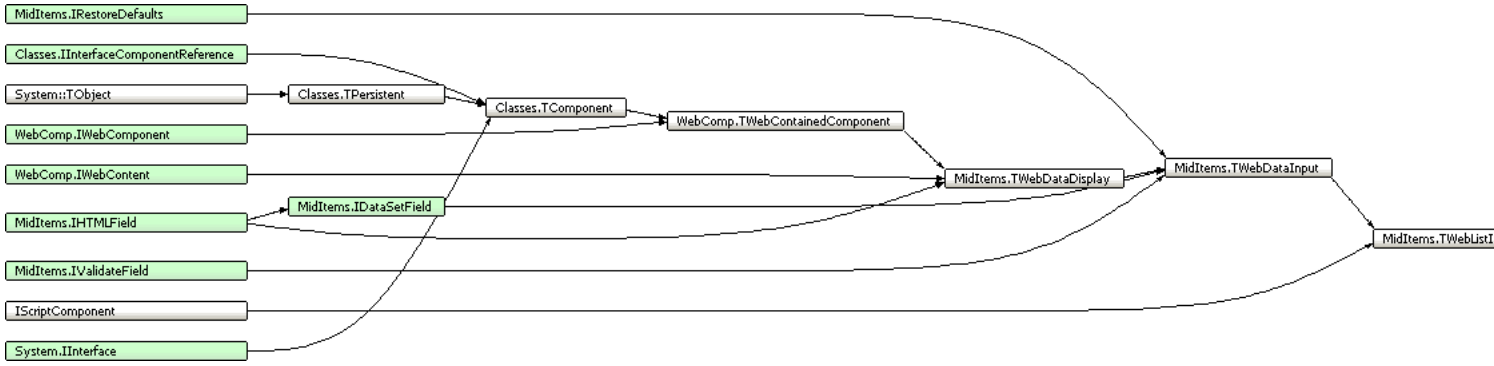

Pascal

```
TWebRadioGroupInput = class(TWebListInput);
```

C++

```
class TWebRadioGroupInput : public TWebListInput;
```

File

MidItems

Description

Use TWebRadioGroupInput as a base class for components that generate HTML radio groups that represent field or parameter values. TWebRadioGroupInput descendants generate controls that are organized by TFieldGroup (see page 16591) or TQueryFieldGroup (see page 16736).

TWebRadioGroupInput defines or inherits properties and methods for

Specifying a field or parameter whose value the radio group displays.

Specifying the possible values from which the user can choose.

Generating a caption that describes the information displayed.

Customizing the appearance of the radio group.

Fitting the radio group into the tab order of the generated HTML form.

Fitting the component into a hierarchy of Web items that collectively generate the content of an HTML document.

In order to let other Web items to use the HTML control it generates, TWebRadioGroupInput inherits the IWebContent interface from TWebDataDisplay (see page 16884).

Do not create instances of TWebRadioGroupInput. Instead, use a descendant of TWebRadioGroupInput such as TFieldRadioGroup (see page 16599) or TQueryRadioGroup (see page 16753).

See Also

IWebContent (see page 24792)

TFieldGroup (see page 16591)

TQueryFieldGroup (see page 16736)

TFieldRadioGroup (see page 16599)

TQUeryRadioGroup

99.1.56.1 TWebRadioGroupInput Members

The following tables list the members exposed by TWebRadioGroupInput.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddAttributes (see page 16953)	This is AddAttributes, a member of class TWebRadioGroupInput.
	AddElements (see page 16941)	This is AddElements, a member of class TWebListInput.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 16942)	This is ControlContent, a member of class TWebListInput.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16953)	This is EventContent, a member of class TWebRadioGroupInput.
V	FindAssociatedField (see page 16906)	This is FindAssociatedField, a member of class TWebDataInput.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	FormatInputs (see page 16953)	This is FormatInputs, a member of class TWebRadioGroupInput.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 16906)	Returns the value of the Caption property.
V	GetCheckIndex (see page 16954)	This is GetCheckIndex, a member of class TWebRadioGroupInput.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetFieldName (see page 16906)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetParamName (see page 16906)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 16907)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetSubComponents (see page 16942)	This is GetSubComponents, a member of class TWebListInput.

	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplAddElements (see page 16942)	This is ImplAddElements, a member of class TWebListInput.
V	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
V	ImplGetFieldName (see page 16907)	This is ImplGetFieldName, a member of class TWebDataInput.
V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 16907)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 16954)	This is ImplGetRowSetFieldAttributes, a member of class TWebRadioGroupInput.
V	ImplRestoreDefaults (see page 16908)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16943)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 16908)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16954)	This is RestoreFieldDefaults, a member of class TWebRadioGroupInput.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
V	SetCaption (see page 16908)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 16943)	This is SetDataSet, a member of class TWebListInput.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 16909)	This is SetFieldName, a member of class TWebDataInput.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItems (see page 16943)	This is SetItems, a member of class TWebListInput.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParamName (see page 16909)	This is SetParamName, a member of class TWebDataInput.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetValues (see page 16944)	This is SetValues, a member of class TWebListInput.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateField (see page 16909)	This is ValidateField, a member of class TWebDataInput.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	CaptionPosition (see page 16897)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 16910)	Specifies the name of the parameter whose value the generated control represents.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16952)	Creates and initializes an instance of TWebRadioGroupInput (see page 16947).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
V	Destroy (see page 16941)	Frees an instance of TWebListInput (see page 16935).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 17057)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRowSetFieldAttributes (see page 17058)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 17058)	This is SetFieldName, a member of class IDataSetField.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateField (see page 17076)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
	Caption (see page 16897)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16897)	Customizes the appearance of the generated control's label.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16898)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 16944)	Specifies a dataset that supplies the possible values listed in the generated control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayColumns (see page 16954)	Specifies the number of radio buttons that appear in each row of the control.
	DisplayWidth (see page 16955)	Specifies the width of the radio group, in characters.
	FieldName (see page 17058)	This is FieldName, a member of class IDataSetField.
	FieldName (see page 16909)	Specifies the name of the field whose value the generated control represents.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
	Items (see page 16945)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 16945)	Indicates which field in DataSet (see page 16944) supplies the possible selections for the generated control.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 16955)	Indicates whether end users can use the generated control to set values.
	Style (see page 16899)	Adds style attributes to the generated display control.
	StyleRule (see page 16899)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 16899)	Determines the position of the generated control in the HTML form's tab order.
	Values (see page 16946)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 16946)	Indicates which field in DataSet (see page 16944) supplies the possible values for the generated control.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.56.2 TWebRadioGroupInput.Create Constructor

Creates and initializes an instance of TWebRadioGroupInput (see page 16947).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebRadioGroupInput(TComponent * AOwner);
```

Description

Call Create to instantiate TWebRadioGroupInput (see page 16947) at runtime. TWebRadioGroupInput (see page 16947) descendants added at design time using the Web Page editor are instantiated automatically.

99.1.56.3 TWebRadioGroupInput Methods

The methods of the TWebRadioGroupInput class are listed here.

Protected Methods

	Name	Description
	AddAttributes (see page 16953)	This is AddAttributes, a member of class TWebRadioGroupInput.
	EventContent (see page 16953)	This is EventContent, a member of class TWebRadioGroupInput.
	FormatInputs (see page 16953)	This is FormatInputs, a member of class TWebRadioGroupInput.
	GetCheckIndex (see page 16954)	This is GetCheckIndex, a member of class TWebRadioGroupInput.
	ImplGetRowSetFieldAttributes (see page 16954)	This is ImplGetRowSetFieldAttributes, a member of class TWebRadioGroupInput.
	RestoreFieldDefaults (see page 16954)	This is RestoreFieldDefaults, a member of class TWebRadioGroupInput.

99.1.56.3.1 TWebRadioGroupInput.AddAttributes Method

Pascal

```
procedure AddAttributes(var Attrs: string); virtual;
```

C++

```
virtual __fastcall AddAttributes(AnsiString Attrs);
```

Description

This is AddAttributes, a member of class TWebRadioGroupInput.

99.1.56.3.2 TWebRadioGroupInput.EventContent Method

Pascal

```
function EventContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall AnsiString EventContent(TWebContentOptions Options);
```

Description

This is EventContent, a member of class TWebRadioGroupInput.

99.1.56.3.3 TWebRadioGroupInput.FormatInputs Method

Pascal

```
function FormatInputs(ItemsStrings: TStrings; ValuesStrings: TStrings; Options:
TWebContentOptions): string; override;
```

C++

```
virtual __fastcall AnsiString FormatInputs(TStrings ItemsStrings, TStrings ValuesStrings,
TWebContentOptions Options);
```

Description

This is FormatInputs, a member of class TWebRadioGroupInput.

99.1.56.3.4 TWebRadioGroupInput.GetCheckIndex Method

Pascal

```
function GetCheckIndex(ItemsStrings: TStrings; ValuesStrings: TStrings): Integer; virtual;
```

C++

```
virtual __fastcall int GetCheckIndex(TStrings ItemsStrings, TStrings ValuesStrings);
```

Description

This is GetCheckIndex, a member of class TWebRadioGroupInput.

99.1.56.3.5 TWebRadioGroupInput.ImplGetRowSetFieldAttributes Method

Pascal

```
function ImplGetRowSetFieldAttributes(const FieldVarName: string): string; override;
```

C++

```
virtual __fastcall AnsiString ImplGetRowSetFieldAttributes(const AnsiString FieldVarName);
```

Description

This is ImplGetRowSetFieldAttributes, a member of class TWebRadioGroupInput.

99.1.56.3.6 TWebRadioGroupInput.RestoreFieldDefaults Method

Pascal

```
procedure RestoreFieldDefaults(AField: TField); override;
```

C++

```
virtual __fastcall RestoreFieldDefaults(TField AField);
```

Description

This is RestoreFieldDefaults, a member of class TWebRadioGroupInput.

99.1.56.4 TWebRadioGroupInput Properties

The properties of the TWebRadioGroupInput class are listed here.

Public Properties

	Name	Description
	DisplayColumns (see page 16954)	Specifies the number of radio buttons that appear in each row of the control.
	DisplayWidth (see page 16955)	Specifies the width of the radio group, in characters.
	ReadOnly (see page 16955)	Indicates whether end users can use the generated control to set values.

99.1.56.4.1 TWebRadioGroupInput.DisplayColumns Property

Specifies the number of radio buttons that appear in each row of the control.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Use DisplayColumns to specify how many radio buttons can appear in a single line of the radio group. If DisplayColumns is less than the number of items, the radio group displays its buttons in multiple rows.

See Also

ItemsField (see page 16945)

Items (see page 16945)

DisplayWidth (see page 16955)

99.1.56.4.2 TWebRadioGroupInput.DisplayWidth Property

Specifies the width of the radio group, in characters.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to limit the width of the generated control. TWebRadioGroup generates an INPUT tag of type RADIO to implement the radio group. DisplayWidth is added as a WIDTH option.

See Also

DisplayColumns (see page 16954)

Custom (see page 16898)

99.1.56.4.3 TWebRadioGroupInput.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the radio group. TWebRadioGroup generates an INPUT tag of type RADIO to implement the radio group. ReadOnly is added as a READONLY option.

See Also

DisplayWidth (see page 16955)

Custom (see page 16898)

99.1.57 TWebSelectOptionsInput Class

TWebSelectOptionsInput is the base class for Web items that create HTML list controls.

Class Hierarchy

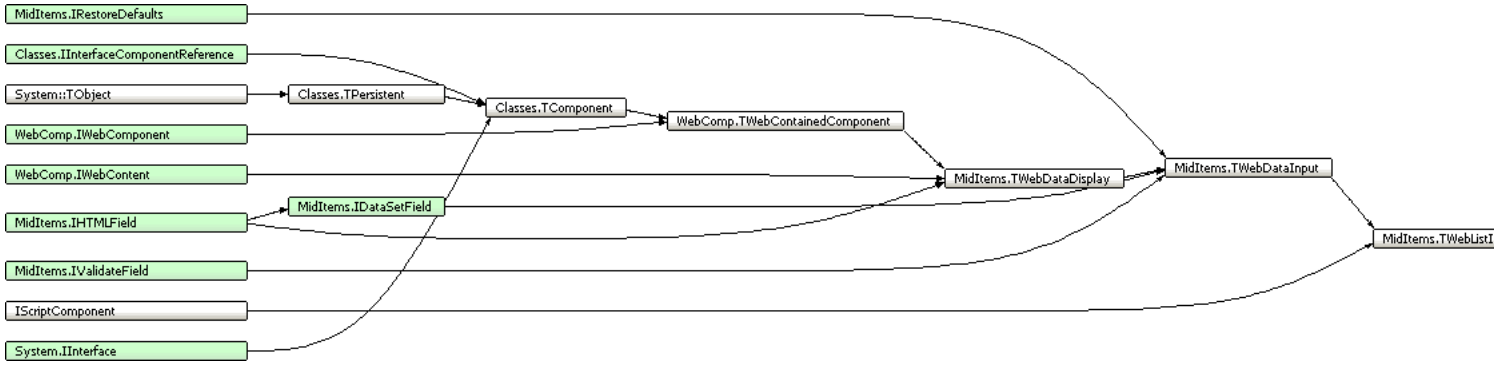

Pascal

```
TWebSelectOptionsInput = class(TWebListInput);
```

C++

```
class TWebSelectOptionsInput : public TWebListInput;
```

File

MidItems

Description

Use TWebSelectOptionsInput as a base class for components that generate HTML list controls that represent field or parameter values. TWebSelectOptionsInput descendants generate controls that are organized by TDataGrid (see page 16560), TFieldGroup (see page 16591), or TQueryFieldGroup (see page 16736).

TWebSelectOptionsInput defines or inherits properties and methods for

Specifying a field or parameter whose value the list box displays.

Specifying the possible values from which the user can choose.

Generating a caption that describes the information displayed.

Customizing the appearance of the list box.

Fitting the list box into the tab order of the generated HTML form.

Fitting the component into a hierarchy of Web items that collectively generate the content of an HTML document.

In order to let other Web items to use the HTML control it generates, TWebSelectOptionsInput inherits the IWebContent interface from TWebDataDisplay (see page 16884).

Do not create instances of TWebSelectOptionsInput. Instead, use a descendant of TWebSelectOptionsInput such as TFieldSelectOptions (see page 16612), TQuerySelectOptions (see page 16768), or TSelectOptionsColumn (see page 16814).

See Also

IWebContent (see page 24792)

TFieldGroup (see page 16591)

TQueryFieldGroup (see page 16736)

TFieldSelectOptions (see page 16612)

TQUerySelectOptions

TDataGrid (see page 16560)

TSelectOptionsColumn (see page 16814)

99.1.57.1 TWebSelectOptionsInput Members

The following tables list the members exposed by TWebSelectOptionsInput.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddAttributes (see page 16962)	This is AddAttributes, a member of class TWebSelectOptionsInput.
	AddElements (see page 16941)	This is AddElements, a member of class TWebListInput.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 16942)	This is ControlContent, a member of class TWebListInput.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16962)	This is EventContent, a member of class TWebSelectOptionsInput.
V	FindAssociatedField (see page 16906)	This is FindAssociatedField, a member of class TWebDataInput.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	FormatInputs (see page 16962)	This is FormatInputs, a member of class TWebSelectOptionsInput.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 16906)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetFieldName (see page 16906)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.

	GetParamName (see page 16906)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 16907)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetSelectIndex (see page 16962)	This is GetSelectIndex, a member of class TWebSelectOptionsInput.
	GetSubComponents (see page 16942)	This is GetSubComponents, a member of class TWebListInput.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplAddElements (see page 16942)	This is ImplAddElements, a member of class TWebListInput.
	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
	ImplGetFieldName (see page 16907)	This is ImplGetFieldName, a member of class TWebDataInput.
	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
	ImplGetHTMLControlName (see page 16907)	Returns the name that other components can use to refer to the generated display control.
	ImplGetRowSetFieldAttributes (see page 16907)	This is ImplGetRowSetFieldAttributes, a member of class TWebDataInput.
	ImplRestoreDefaults (see page 16908)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 16943)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 16908)	This is RestoreDefaults, a member of class TWebDataInput.
	RestoreFieldDefaults (see page 16908)	This is RestoreFieldDefaults, a member of class TWebDataInput.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
	SetCaption (see page 16908)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDataSet (see page 16943)	This is SetDataSet, a member of class TWebListInput.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 16909)	This is SetFieldName, a member of class TWebDataInput.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItems (see page 16943)	This is SetItems, a member of class TWebListInput.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParamName (see page 16909)	This is SetParamName, a member of class TWebDataInput.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetValues (see page 16944)	This is SetValues, a member of class TWebListInput.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateField (see page 16909)	This is ValidateField, a member of class TWebDataInput.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	CaptionPosition (see page 16897)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 16910)	Specifies the name of the parameter whose value the generated control represents.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16961)	Creates and initializes an instance of TWebSelectOptionsInput (see page 16956).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
V	Destroy (see page 16941)	Frees an instance of TWebListInput (see page 16935).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 17057)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRowSetFieldAttributes (see page 17058)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 17058)	This is SetFieldName, a member of class IDataSetField.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateField (see page 17076)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
	Caption (see page 16897)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16897)	Customizes the appearance of the generated control's label.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16898)	Represents a tag option and its value in the tag for the display control.
	DataSet (see page 16944)	Specifies a dataset that supplies the possible values listed in the generated control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayRows (see page 16963)	Specifies the number of rows that appear in the generated list box.
	FieldName (see page 17058)	This is FieldName, a member of class IDataSetField.
	FieldName (see page 16909)	Specifies the name of the field whose value the generated control represents.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
	Items (see page 16945)	Lists the possible selections that appear in the generated control.
	ItemsField (see page 16945)	Indicates which field in DataSet (see page 16944) supplies the possible selections for the generated control.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16899)	Adds style attributes to the generated display control.
	StyleRule (see page 16899)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 16899)	Determines the position of the generated control in the HTML form's tab order.
	Values (see page 16946)	Lists the possible values that can be assigned to the control.
	ValuesField (see page 16946)	Indicates which field in DataSet (see page 16944) supplies the possible values for the generated control.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.57.2 TWebSelectOptionsInput.Create Constructor

Creates and initializes an instance of TWebSelectOptionsInput (see page 16956).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebSelectOptionsInput(TComponent * AOwner);
```

Description

Call Create to instantiate TWebSelectOptionsInput (see page 16956) at runtime. TWebSelectOptionsInput (see page 16956) descendants added at design time using the Web Page editor are instantiated automatically.

99.1.57.3 TWebSelectOptionsInput Methods

The methods of the TWebSelectOptionsInput class are listed here.

Protected Methods

	Name	Description
	AddAttributes (see page 16962)	This is AddAttributes, a member of class TWebSelectOptionsInput.
	EventContent (see page 16962)	This is EventContent, a member of class TWebSelectOptionsInput.
	FormatInputs (see page 16962)	This is FormatInputs, a member of class TWebSelectOptionsInput.
	GetSelectIndex (see page 16962)	This is GetSelectIndex, a member of class TWebSelectOptionsInput.

99.1.57.3.1 TWebSelectOptionsInput.AddAttributes Method

Pascal

```
procedure AddAttributes(var Attrs: string); virtual;
```

C++

```
virtual __fastcall AddAttributes(AnsiString Attrs);
```

Description

This is AddAttributes, a member of class TWebSelectOptionsInput.

99.1.57.3.2 TWebSelectOptionsInput.EventContent Method

Pascal

```
function EventContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall AnsiString EventContent(TWebContentOptions Options);
```

Description

This is EventContent, a member of class TWebSelectOptionsInput.

99.1.57.3.3 TWebSelectOptionsInput.FormatInputs Method

Pascal

```
function FormatInputs(ItemsStrings: TStrings; ValuesStrings: TStrings; Options:
TWebContentOptions): string; override;
```

C++

```
virtual __fastcall AnsiString FormatInputs(TStrings ItemsStrings, TStrings ValuesStrings,
TWebContentOptions Options);
```

Description

This is FormatInputs, a member of class TWebSelectOptionsInput.

99.1.57.3.4 TWebSelectOptionsInput.GetSelectIndex Method

Pascal

```
function GetSelectIndex(ItemsStrings: TStrings; ValuesStrings: TStrings): Integer; virtual;
```

C++

```
virtual __fastcall int GetSelectIndex(TStrings ItemsStrings, TStrings ValuesStrings);
```

Description

This is GetSelectIndex, a member of class TWebSelectOptionsInput.

99.1.57.4 TWebSelectOptionsInput Properties

The properties of the TWebSelectOptionsInput class are listed here.

Public Properties

	Name	Description
	DisplayRows (see page 16963)	Specifies the number of rows that appear in the generated list box.

99.1.57.4.1 TWebSelectOptionsInput.DisplayRows Property

Specifies the number of rows that appear in the generated list box.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to specify how many rows appear in the generated list control. If DisplayRows is less than the number of items, the generated list displays only DisplayRows options at one time, but can scroll to display the hidden items. If DisplayRows is 1, the generated control becomes a drop-down list box.

See Also

ItemsField (see page 16945)

Items (see page 16945)

99.1.58 TWebStatus Class

TWebStatus is the base class for Web items that indicate the status of records in an XML data packet.

Class Hierarchy

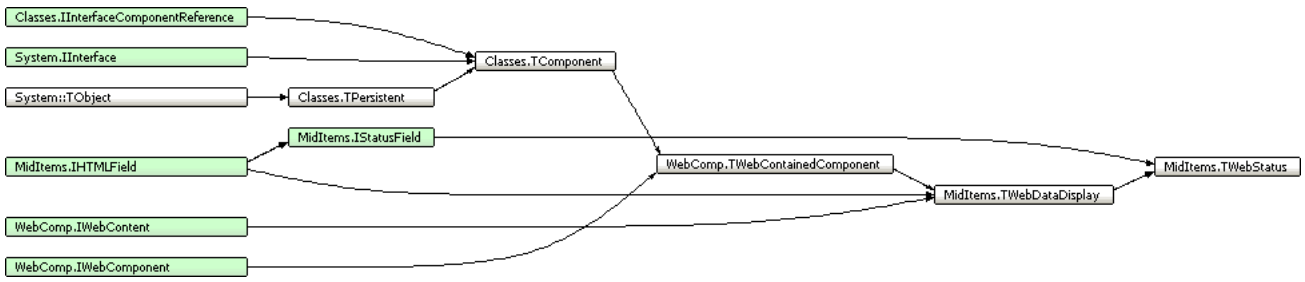

Pascal

```
TWebStatus = class(TWebDataDisplay, IStatusField);
```

C++

```
class TWebStatus : public TWebDataDisplay, public IStatusField;
```

File

MidItems

Description

Use TWebStatus as a base class for components that generate HTML controls which indicate whether records have been inserted or modified by the user. TWebStatus descendants generate controls that are added to TDataGrid (see page 16560) or TFieldGroup (see page 16591) to provide feedback about modifications to the underlying XML data packet.

TWebStatus defines or inherits properties and methods for

Generating a caption that describes the information displayed.

Customizing the appearance of the control.

Fitting the control into the tab order of the generated HTML form.

Fitting the component into a hierarchy of Web items that collectively generate the content of an HTML document.

In order to let other Web items to use the HTML control it generates, TWebStatus inherits the IWebContent interface from TWebDataDisplay (see page 16884).

Do not create instances of TWebStatus. Instead, use a descendant of TWebStatus such as TFieldStatus (see page 16624) or TStatusColumn (see page 16826).

See Also

IWebContent (see page 24792)

TDataGrid (see page 16560)

TFieldGroup (see page 16591)

TFieldStatus (see page 16624)

TStatusColumn (see page 16826)

99.1.58.1 TWebStatus Members

The following tables list the members exposed by TWebStatus.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.

	ControlContent (see page 16969)	This is ControlContent, a member of class TWebStatus.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
	GetCaption (see page 16969)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
	ImplGetHTMLControlName (see page 16969)	Returns the name that other components can use to refer to the generated display control.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.

	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16968)	Creates and initializes an instance of TWebStatus (see page 16963).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	Identifier (see page 16969)	Returns the name under which record status information is stored in the XML data packet.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 16970)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16970)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16971)	Specifies where the label appears relative to the generated display control.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16898)	Represents a tag option and its value in the tag for the display control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayWidth (see page 16971)	Specifies the width, in characters, of the generated HTML control.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16899)	Adds style attributes to the generated display control.
	StyleRule (see page 16899)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 16899)	Determines the position of the generated control in the HTML form's tab order.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.58.2 TWebStatus.Create Constructor

Creates and initializes an instance of TWebStatus (see page 16963).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebStatus(TComponent * AOwner);
```

Description

Call Create to instantiate TWebStatus (see page 16963) at runtime. TWebStatus (see page 16963) descendants added at design time using the Web Page editor are instantiated automatically.

99.1.58.3 TWebStatus Methods

The methods of the TWebStatus class are listed here.

Protected Methods

	Name	Description
V	ControlContent (see page 16969)	This is ControlContent, a member of class TWebStatus.
V	GetCaption (see page 16969)	Returns the value of the Caption property.
V	ImplGetHTMLControlName (see page 16969)	Returns the name that other components can use to refer to the generated display control.

Public Methods

	Name	Description
	Identifier (see page 16969)	Returns the name under which record status information is stored in the XML data packet.

99.1.58.3.1 TWebStatus.ControlContent Method

Pascal

```
function ControlContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall AnsiString ControlContent(TWebContentOptions Options);
```

Description

This is ControlContent, a member of class TWebStatus.

99.1.58.3.2 TWebStatus.GetCaption Method

Returns the value of the Caption property.

Pascal

```
function GetCaption: string; override;
```

C++

```
virtual __fastcall AnsiString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption property.

See Also

Caption (see page 16897)

99.1.58.3.3 TWebStatus.Identifier Method

Returns the name under which record status information is stored in the XML data packet.

Pascal

```
class function Identifier: string;
```

C++

```
__fastcall AnsiString Identifier();
```

Description

Call Identifier to determine the name of the pseudo-field in the XML data packet that stores the status information displayed by the generated control.

99.1.58.3.4 TWebStatus.ImplGetHTMLControlName Method

Returns the name that other components can use to refer to the generated display control.

Pascal

```
function ImplGetHTMLControlName: string; override;
```

C++

```
virtual __fastcall AnsiString ImplGetHTMLControlName();
```

Description

ImplGetHTMLControlName returns the name of the generated control. This value is assigned to a NAME option on the generated HTML tag.

ImplGetHTMLControlName returns the value of the Name property. TWebDataDisplay descendants can override this method to return a more meaningful value such as the name of the field or parameter displayed in the control.

See Also

Name (see page 2805)

99.1.58.4 TWebStatus Properties

The properties of the TWebStatus class are listed here.

Public Properties

	Name	Description
	Caption (see page 16970)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16970)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 16971)	Specifies where the label appears relative to the generated display control.
	DisplayWidth (see page 16971)	Specifies the width, in characters, of the generated HTML control.

99.1.58.4.1 TWebStatus.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 16897)

CaptionPosition (see page 16897)

99.1.58.4.2 TWebStatus.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 16897)

CaptionPosition (see page 16897)

HeadingAttributes (see page 16509)

99.1.58.4.3 TWebStatus.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 16897)

Caption (see page 16897)

99.1.58.4.4 TWebStatus.DisplayWidth Property

Specifies the width, in characters, of the generated HTML control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to control the width of the HTML control that displays record status information. By default, DisplayWidth is 1 because the TWebStatus (see page 16963) control values are one character (M for modified or I for inserted).

99.1.59 TWebTextAreaInput Class

TWebTextAreaInput is the base class for Web items that create HTML multi-line text controls.

Class Hierarchy

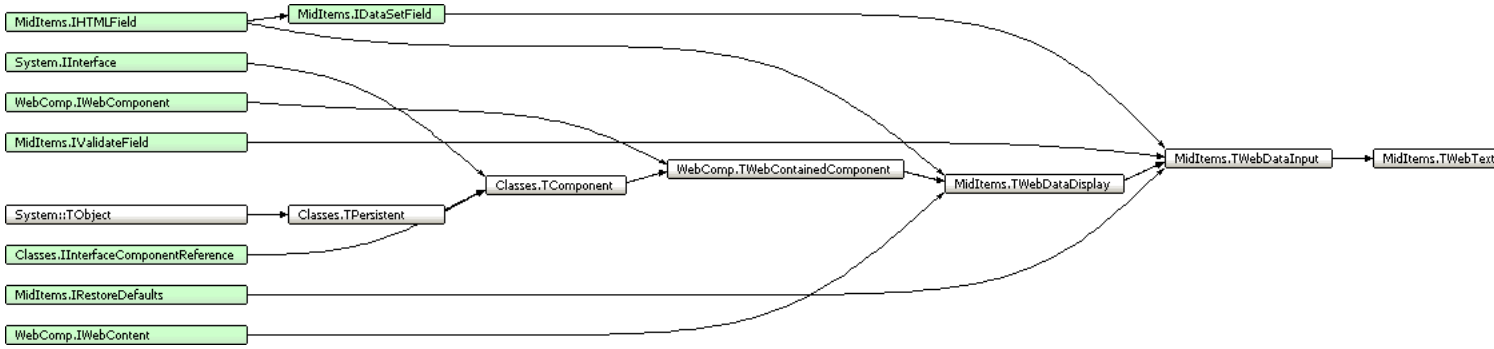

Pascal

```
TWebTextAreaInput = class(TWebDataInput);
```

C++

```
class TWebTextAreaInput : public TWebDataInput;
```

File

MidItems

Description

Use TWebTextAreaInput as a base class for components that generate HTML controls for displaying a field or parameter value in a multi-line text control. TWebTextAreaInput descendants generate controls that are organized by TDataGrid (see page 16560), TFieldGroup (see page 16591), or TQueryFieldGroup (see page 16736).

TWebTextAreaInput defines or inherits properties and methods for

Specifying a field or parameter whose value the control displays.

Generating a caption that describes the information displayed.

Customizing the appearance of the control that displays information.

Fitting the control into the tab order of the generated HTML form.

Fitting the component into a hierarchy of Web items that collectively generate the content of an HTML document.

In order to let other Web items to use the HTML control it generates, TWebTextAreaInput inherits the IWebContent interface from TWebDataDisplay (see page 16884).

Do not create instances of TWebTextAreaInput. Instead, use a descendant of TWebTextAreaInput such as TFieldTextArea (see page 16640), TQueryTextArea (see page 16795), or TTextAreaColumn (see page 16838).

See Also

IWebContent (see page 24792)

TDataGrid (see page 16560)

TFieldGroup (see page 16591)

TQueryFieldGroup (see page 16736)

TFieldTextArea (see page 16640)

TQueryTextArea (see page 16795)

TTextAreaColumn (see page 16838)

99.1.59.1 TWebTextAreaInput Members

The following tables list the members exposed by TWebTextAreaInput.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddAttributes (see page 16977)	This is AddAttributes, a member of class TWebTextAreaInput.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
V	ControlContent (see page 16978)	This is ControlContent, a member of class TWebTextAreaInput.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16978)	This is EventContent, a member of class TWebTextAreaInput.
V	FindAssociatedField (see page 16906)	This is FindAssociatedField, a member of class TWebDataInput.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 16906)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetFieldName (see page 16906)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetParamName (see page 16906)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 16907)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
V	ImplGetFieldName (see page 16907)	This is ImplGetFieldName, a member of class TWebDataInput.

V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 16907)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 16978)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextAreaInput.
V	ImplRestoreDefaults (see page 16908)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 16908)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16978)	This is RestoreFieldDefaults, a member of class TWebTextAreaInput.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
V	SetCaption (see page 16908)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 16909)	This is SetFieldName, a member of class TWebDataInput.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParamName (see page 16909)	This is SetParamName, a member of class TWebDataInput.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateField (see page 16909)	This is ValidateField, a member of class TWebDataInput.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	CaptionPosition (see page 16897)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 16910)	Specifies the name of the parameter whose value the generated control represents.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.

R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16977)	Creates and initializes an instance of TWebTextAreaInput (see page 16971).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 17057)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRowSetFieldAttributes (see page 17058)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.

V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 17058)	This is SetFieldName, a member of class IDataSetField.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateField (see page 17076)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
	Caption (see page 16897)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16897)	Customizes the appearance of the generated control's label.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16898)	Represents a tag option and its value in the tag for the display control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayRows (see page 16979)	Specifies the height of the generated control.
	DisplayWidth (see page 16979)	Specifies the horizontal size of the generated control.
	FieldName (see page 17058)	This is FieldName, a member of class IDataSetField.
	FieldName (see page 16909)	Specifies the name of the field whose value the generated control represents.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.

R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 16979)	Indicates whether end users can use the generated control to set values.
	Style (see page 16899)	Adds style attributes to the generated display control.
	StyleRule (see page 16899)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 16899)	Determines the position of the generated control in the HTML form's tab order.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Wrap (see page 16980)	Specifies how the generated control treats lines that are longer than the display width.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.59.2 TWebTextAreaInput.Create Constructor

Creates and initializes an instance of TWebTextAreaInput (see page 16971).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebTextAreaInput(TComponent * AOwner);
```

Description

Call Create to instantiate TWebTextAreaInput (see page 16971) at runtime. TWebTextAreaInput (see page 16971) descendants added at design time using the Web Page editor are instantiated automatically.

99.1.59.3 TWebTextAreaInput Methods

The methods of the TWebTextAreaInput class are listed here.

Protected Methods

	Name	Description
V	AddAttributes (see page 16977)	This is AddAttributes, a member of class TWebTextAreaInput.
V	ControlContent (see page 16978)	This is ControlContent, a member of class TWebTextAreaInput.
V	EventContent (see page 16978)	This is EventContent, a member of class TWebTextAreaInput.
V	ImplGetRowSetFieldAttributes (see page 16978)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextAreaInput.
V	RestoreFieldDefaults (see page 16978)	This is RestoreFieldDefaults, a member of class TWebTextAreaInput.

99.1.59.3.1 TWebTextAreaInput.AddAttributes Method

Pascal

```
procedure AddAttributes(var Attrs: string); virtual;
```

C++

```
virtual __fastcall AddAttributes(AnsiString Attrs);
```

Description

This is AddAttributes, a member of class TWebTextAreaInput.

99.1.59.3.2 TWebTextAreaInput.ControlContent Method

Pascal

```
function ControlContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall AnsiString ControlContent(TWebContentOptions Options);
```

Description

This is ControlContent, a member of class TWebTextAreaInput.

99.1.59.3.3 TWebTextAreaInput.EventContent Method

Pascal

```
function EventContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall AnsiString EventContent(TWebContentOptions Options);
```

Description

This is EventContent, a member of class TWebTextAreaInput.

99.1.59.3.4 TWebTextAreaInput.ImplGetRowSetFieldAttributes Method

Pascal

```
function ImplGetRowSetFieldAttributes(const FieldVarName: string): string; override;
```

C++

```
virtual __fastcall AnsiString ImplGetRowSetFieldAttributes(const AnsiString FieldVarName);
```

Description

This is ImplGetRowSetFieldAttributes, a member of class TWebTextAreaInput.

99.1.59.3.5 TWebTextAreaInput.RestoreFieldDefaults Method

Pascal

```
procedure RestoreFieldDefaults(AField: TField); override;
```

C++

```
virtual __fastcall RestoreFieldDefaults(TField AField);
```

Description

This is RestoreFieldDefaults, a member of class TWebTextAreaInput.

99.1.59.4 TWebTextAreaInput Properties

The properties of the TWebTextAreaInput class are listed here.

Public Properties

	Name	Description
	DisplayRows (see page 16979)	Specifies the height of the generated control.
	DisplayWidth (see page 16979)	Specifies the horizontal size of the generated control.
	ReadOnly (see page 16979)	Indicates whether end users can use the generated control to set values.
	Wrap (see page 16980)	Specifies how the generated control treats lines that are longer than the display width.

99.1.59.4.1 TWebTextAreaInput.DisplayRows Property

Specifies the height of the generated control.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayRows to indicate the desired height of the generated HTML control. DisplayRows specifies the number of lines of text that can be displayed in the control. This value is assigned to the ROWS option of the generated TEXTAREA tag.

See Also

DisplayWidth (see page 16979)

99.1.59.4.2 TWebTextAreaInput.DisplayWidth Property

Specifies the horizontal size of the generated control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to indicate the desired width of the generated HTML control. DisplayWidth specifies the number of characters that can be displayed in one line of the control. This value is assigned to the COLS option of the generated TEXTAREA tag.

See Also

DisplayRows (see page 16979)

99.1.59.4.3 TWebTextAreaInput.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the generated control. This value determines

whether there is a READONLY option added to the generated TEXTAREA tag.

99.1.59.4.4 TWebTextAreaInput.Wrap Property

Specifies how the generated control treats lines that are longer than the display width.

Pascal

```
property Wrap: TTextAreaWrap;
```

C++

```
__property TTextAreaWrap Wrap;
```

Description

Use Wrap to specify how the generated control formats lines that are longer than the number of characters specified by DisplayWidth (see page 16979). TWebTextAreaInput (see page 16971) uses Wrap to add a WRAP option to the generated TEXTAREA tag.

Wrap can have any of the following values:

Value	WRAP option value	Meaning
wrOff	OFF	The control does not wrap long lines. Only newline characters that are part of the text string cause the lines in the control to wrap.
wrPhysical	PHYSICAL	The control formats its text value by wrapping long lines so that no text is hidden. The control inserts carriage returns into the text string.
wrVirtual	VIRTUAL	The control formats its text value by wrapping long lines so that no text is hidden but does not change the actual value of the text string.

See Also

DisplayWidth (see page 16979)

99.1.60 TWebTextInput Class

TWebTextInput is the base class for Web items that create HTML single-line text controls.

Class Hierarchy

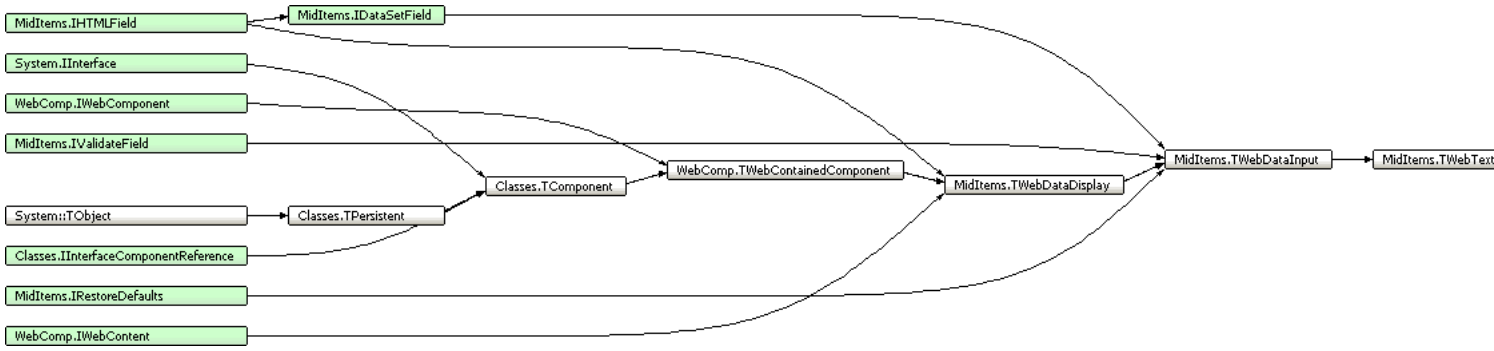

Pascal

```
TWebTextInput = class(TWebDataInput);
```

C++

```
class TWebTextInput : public TWebDataInput;
```

File

MidItems

Description

Use TWebTextInput as a base class for components that generate HTML controls for displaying a field or parameter value as a single-line text string. TWebTextInput descendants generate controls that are organized by TDataGrid (see page 16560), TFieldGroup (see page 16591), or TQueryFieldGroup (see page 16736).

TWebTextInput defines or inherits properties and methods for

Specifying a field or parameter whose value the control displays.

Generating a caption that describes the information displayed.

Customizing the appearance of the control that displays information.

Fitting the control into the tab order of the generated HTML form.

Fitting the component into a hierarchy of Web items that collectively generate the content of an HTML document.

In order to let other Web items to use the HTML control it generates, TWebTextInput inherits the IWebContent interface from TWebDataDisplay (see page 16884).

Do not create instances of TWebTextInput. Instead, use a descendant of TWebTextInput such as TFieldText (see page 16631), TQueryText (see page 16783), or TTextColumn (see page 16849).

See Also

IWebContent (see page 24792)

TDataGrid (see page 16560)

TFieldGroup (see page 16591)

TQueryFieldGroup (see page 16736)

TFieldText (see page 16631)

TQueryText (see page 16783)

TTextColumn (see page 16849)

99.1.60.1 TWebTextInput Members

The following tables list the members exposed by TWebTextInput.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddAttributes (see page 16986)	This is AddAttributes, a member of class TWebTextInput.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
V	ControlContent (see page 16987)	This is ControlContent, a member of class TWebTextInput.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	EventContent (see page 16905)	This is EventContent, a member of class TWebDataInput.
V	FindAssociatedField (see page 16906)	This is FindAssociatedField, a member of class TWebDataInput.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 16906)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetFieldName (see page 16906)	This is GetFieldName, a member of class TWebDataInput.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetParamName (see page 16906)	This is GetParamName, a member of class TWebDataInput.
	GetRowSetFieldAttributes (see page 16907)	This is GetRowSetFieldAttributes, a member of class TWebDataInput.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
V	ImplGetFieldName (see page 16907)	This is ImplGetFieldName, a member of class TWebDataInput.

V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetHTMLControlName (see page 16907)	Returns the name that other components can use to refer to the generated display control.
V	ImplGetRowSetFieldAttributes (see page 16987)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextInput.
V	ImplRestoreDefaults (see page 16908)	This is ImplRestoreDefaults, a member of class TWebDataInput.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 16908)	This is RestoreDefaults, a member of class TWebDataInput.
V	RestoreFieldDefaults (see page 16987)	This is RestoreFieldDefaults, a member of class TWebTextInput.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
V	SetCaption (see page 16908)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 16909)	This is SetFieldName, a member of class TWebDataInput.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParamName (see page 16909)	This is SetParamName, a member of class TWebDataInput.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateField (see page 16909)	This is ValidateField, a member of class TWebDataInput.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	CaptionPosition (see page 16897)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
	ParamName (see page 16910)	Specifies the name of the parameter whose value the generated control represents.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.

R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16986)	Creates and initializes an instance of TWebTextInput (see page 16980).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 17057)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRowSetFieldAttributes (see page 17058)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.

V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 17058)	This is SetFieldName, a member of class IDataSetField.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateField (see page 17076)	This is ValidateField, a member of class IValidateField.

Public Properties

	Name	Description
	Caption (see page 16897)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16897)	Customizes the appearance of the generated control's label.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16987)	Represents a tag option and its value in the tag for the display control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayWidth (see page 16988)	Specifies the horizontal size of the generated control.
	FieldName (see page 17058)	This is FieldName, a member of class IDataSetField.
	FieldName (see page 16909)	Specifies the name of the field whose value the generated control represents.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.

R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 16988)	Indicates whether end users can use the generated control to set values.
	Style (see page 16989)	Adds style attributes to the generated display control.
	StyleRule (see page 16989)	This is StyleRule, a member of class TWebTextInput.
	TabIndex (see page 16899)	Determines the position of the generated control in the HTML form's tab order.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.60.2 TWebTextInput.Create Constructor

Creates and initializes an instance of TWebTextInput (see page 16980).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebTextInput(TComponent * AOwner);
```

Description

Call Create to instantiate a TWebTextInput (see page 16980) descendant at runtime. TWebTextInput (see page 16980) descendants added at design time using the Web Page editor are instantiated automatically.

99.1.60.3 TWebTextInput Methods

The methods of the TWebTextInput class are listed here.

Protected Methods

	Name	Description
V	AddAttributes (see page 16986)	This is AddAttributes, a member of class TWebTextInput.
V	ControlContent (see page 16987)	This is ControlContent, a member of class TWebTextInput.
V	ImplGetRowSetFieldAttributes (see page 16987)	This is ImplGetRowSetFieldAttributes, a member of class TWebTextInput.
V	RestoreFieldDefaults (see page 16987)	This is RestoreFieldDefaults, a member of class TWebTextInput.

99.1.60.3.1 TWebTextInput.AddAttributes Method

Pascal

```
procedure AddAttributes(var Attrs: string); virtual;
```

C++

```
virtual __fastcall AddAttributes(AnsiString Attrs);
```

Description

This is AddAttributes, a member of class TWebTextInput.

99.1.60.3.2 TWebTextInput.ControlContent Method

Pascal

```
function ControlContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall AnsiString ControlContent(TWebContentOptions Options);
```

Description

This is ControlContent, a member of class TWebTextInput.

99.1.60.3.3 TWebTextInput.ImplGetRowSetFieldAttributes Method

Pascal

```
function ImplGetRowSetFieldAttributes(const FieldVarName: string): string; override;
```

C++

```
virtual __fastcall AnsiString ImplGetRowSetFieldAttributes(const AnsiString FieldVarName);
```

Description

This is ImplGetRowSetFieldAttributes, a member of class TWebTextInput.

99.1.60.3.4 TWebTextInput.RestoreFieldDefaults Method

Pascal

```
procedure RestoreFieldDefaults(AField: TField); override;
```

C++

```
virtual __fastcall RestoreFieldDefaults(TField AField);
```

Description

This is RestoreFieldDefaults, a member of class TWebTextInput.

99.1.60.4 TWebTextInput Properties

The properties of the TWebTextInput class are listed here.

Public Properties

	Name	Description
	Custom (see page 16987)	Represents a tag option and its value in the tag for the display control.
	DisplayWidth (see page 16988)	Specifies the horizontal size of the generated control.
	ReadOnly (see page 16988)	Indicates whether end users can use the generated control to set values.
	Style (see page 16989)	Adds style attributes to the generated display control.
	StyleRule (see page 16989)	This is StyleRule, a member of class TWebTextInput.

99.1.60.4.1 TWebTextInput.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 16899)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

99.1.60.4.2 TWebTextInput.DisplayWidth Property

Specifies the horizontal size of the generated control.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to indicate the desired width of the generated HTML control. DisplayWidth specifies the number of characters that can be displayed in the control.

Note: DisplayWidth controls the size of the generated control, not the size of the text that users can enter. If the user enters a string longer than DisplayWidth characters, the text scrolls in the control.

99.1.60.4.3 TWebTextInput.ReadOnly Property

Indicates whether end users can use the generated control to set values.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use ReadOnly to indicate whether users can change the value displayed by the generated control.

99.1.60.4.4 TWebTextInput.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 16898)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

Setting Web Item Properties

99.1.60.4.5 TWebTextInput.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TWebTextInput.

99.1.61 TXMLButton Class

TXMLButton is the base class for Web items that create HTML buttons that perform some action with the XML data packet.

Class Hierarchy

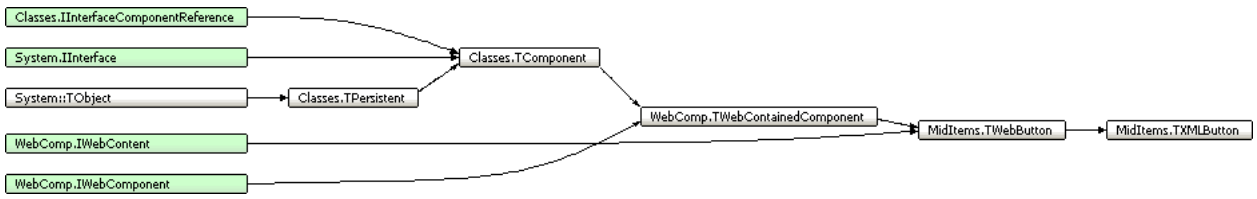

Pascal

```
TXMLButton = class(TWebButton);
```

C++

```
class TXMLButton : public TWebButton;
```

File

MidItems

Description

Use TXMLButton as a base class for components that generate the HTML statements defining a specialized button control for manipulating an XML data packet. TXMLButton defines properties and methods for

Generating the HTML code to define the button control.

Associating the button with an XML broker that fetches the XML data packet and responds to update messages.

Specifying tag attributes of the HTML button control, including those defined by a style sheet.

Fitting the component into a hierarchy of Web items, each of which generates part of a larger HTML document.

In order to allow other Web items to use the HTML generated by the button control, TXMLButton inherits support for the IWebContent interface from TWebButton (see page 16863).

Do not create instances of TXMLButton. TXMLButton relies on anabstract or, in C++ terminology, pure virtual method to generate the HTML statements defining a button control. Instead, create descendants of TXMLButton, such as TApplyUpdatesButton (see page 16480).

See Also

TApplyUpdatesButton (see page 16480)

IWebContent (see page 24792)

99.1.61.1 TXMLButton Members

The following tables list the members exposed by TXMLButton.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16868)	Returns the HTML that defines the button control.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetCaption (see page 16994)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.

	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXMLBroker (see page 16995)	This is GetXMLBroker, a member of class TXMLButton.
	GetXMLUseParent (see page 16995)	This is GetXMLUseParent, a member of class TXMLButton.
A	ImplContent (see page 16869)	Provides the underlying implementation for the Content (see page 16868) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 16995)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	SetXMLBroker (see page 16996)	This is SetXMLBroker, a member of class TXMLButton.
	SetXMLUseParent (see page 16996)	This is SetXMLUseParent, a member of class TXMLButton.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultCaption (see page 16996)	This is DefaultCaption, a member of class TXMLButton.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.

R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.
R	XMLData (see page 16997)	This is XMLData, a member of class TXMLButton.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 16993)	Creates and initializes an instance of TXMLButton (see page 16989).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 16994)	Frees an instance of TXMLButton (see page 16989).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 16870)	Specifies the caption that appears on the button.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16871)	Represents a tag option and its value in the generated INPUT tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16871)	Adds style attributes to the generated button control.
	StyleRule (see page 16872)	Applies a style in the page producer's style sheet to the button.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	XMLBroker (see page 16996)	Associates the XML button with an XML broker.
	XMLUseParent (see page 16997)	Specifies whether the XML button is associated with its parent's XML broker.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.61.2 TXMLButton.Create Constructor

Creates and initializes an instance of TXMLButton (see page 16989).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TXMLButton(TComponent * AOwner);
```

Description

Call Create to instantiate an XML button at runtime. Web buttons added at design time using the Web Page editor are instantiated automatically.

99.1.61.3 TXMLButton.Destroy Destructor

Frees an instance of TXMLButton (see page 16989).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLButton();
```

Description

Destroy removes the XML button from its parent's list of components that generate HTML before freeing the button instance.

99.1.61.4 TXMLButton Methods

The methods of the TXMLButton class are listed here.

Protected Methods

	Name	Description
	GetCaption (see page 16994)	Returns the value of the Caption property.
	GetXMLBroker (see page 16995)	This is GetXMLBroker, a member of class TXMLButton.
	GetXMLUseParent (see page 16995)	This is GetXMLUseParent, a member of class TXMLButton.
	Notification (see page 16995)	Forwards notification messages to all owned components.
	SetXMLBroker (see page 16996)	This is SetXMLBroker, a member of class TXMLButton.
	SetXMLUseParent (see page 16996)	This is SetXMLUseParent, a member of class TXMLButton.

99.1.61.4.1 TXMLButton.GetCaption Method

Returns the value of the Caption property.

Pascal

```
function GetCaption: string; override;
```

C++

```
virtual __fastcall AnsiString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption property.

See Also

Caption (see page 16870)

SetCaption (see page 16870)

99.1.61.4.2 TXMLButton.GetXMLBroker Method

Pascal

```
function GetXMLBroker: TXMLBroker;
```

C++

```
__fastcall TXMLBroker GetXMLBroker();
```

Description

This is GetXMLBroker, a member of class TXMLButton.

99.1.61.4.3 TXMLButton.GetXMLUseParent Method

Pascal

```
function GetXMLUseParent: Boolean;
```

C++

```
__fastcall Boolean GetXMLUseParent();
```

Description

This is GetXMLUseParent, a member of class TXMLButton.

99.1.61.4.4 TXMLButton.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

99.1.61.4.5 TXMLButton.SetXMLBroker Method

Pascal

```
procedure SetXMLBroker(const Value: TXMLBroker);
```

C++

```
__fastcall SetXMLBroker(const TXMLBroker Value);
```

Description

This is SetXMLBroker, a member of class TXMLButton.

99.1.61.4.6 TXMLButton.SetXMLUseParent Method

Pascal

```
procedure SetXMLUseParent(const Value: Boolean);
```

C++

```
__fastcall SetXMLUseParent(const Boolean Value);
```

Description

This is SetXMLUseParent, a member of class TXMLButton.

99.1.61.5 TXMLButton Properties

The properties of the TXMLButton class are listed here.

Protected Properties

	Name	Description
	DefaultCaption (see page 16996)	This is DefaultCaption, a member of class TXMLButton.
R	XMLData (see page 16997)	This is XMLData, a member of class TXMLButton.

Public Properties

	Name	Description
	XMLBroker (see page 16996)	Associates the XML button with an XML broker.
	XMLUseParent (see page 16997)	Specifies whether the XML button is associated with its parent's XML broker.

99.1.61.5.1 TXMLButton.DefaultCaption Property

Pascal

```
property DefaultCaption: string;
```

C++

```
__property AnsiString DefaultCaption;
```

Description

This is DefaultCaption, a member of class TXMLButton.

99.1.61.5.2 TXMLButton.XMLBroker Property

Associates the XML button with an XML broker.

Pascal

```
property XMLBroker: TXMLBroker;
```

C++

```
__property TXMLBroker XMLBroker;
```

Description

Set XMLBroker to specify the XML broker whose data packets the XML button manipulates.

When XML buttons are used by TDataNavigator (see page 16569), the data navigator already establishes a connection (indirectly) to an XML broker. If the XML button uses this same XML broker, you do not need to set the XMLBroker property. Instead, setting the XMLUseParent (see page 16997) property to true (the default) automatically associates the XML button with the same broker as the one associated with the navigator.

See Also

XMLUseParent (see page 16997)

99.1.61.5.3 TXMLButton.XMLData Property

Pascal

```
property XMLData: TXMLDataParent;
```

C++

```
__property TXMLDataParent XMLData;
```

Description

This is XMLData, a member of class TXMLButton.

99.1.61.5.4 TXMLButton.XMLUseParent Property

Specifies whether the XML button is associated with its parent's XML broker.

Pascal

```
property XMLUseParent: Boolean;
```

C++

```
__property Boolean XMLUseParent;
```

Description

Use XMLUseParent to indicate how the XML button determines the XML broker whose data packets it manipulates. When XMLUseParent is false, the XMLBroker (see page 16996) property must be set to indicate the XML broker. When XMLUseParent is true, the XML broker associated with the XML button's parent determines the associated XML broker.

The XML button's parent is the Web item (typically TDataNavigator (see page 16569)) that uses the XML button to assemble its own HTML content. The parent's associated broker may be defined through an indirect association. For example, the data navigator acts on another component, such as a field group, which has an associated broker.

If the parent does not have an associated XML broker, setting XMLUseParent to true sets the XMLBroker (see page 16996) property to nil (Delphi) or NULL (C++).

See Also

XMLBroker (see page 16996)

99.1.62 TXMLData Class

TXMLData manages the relationship between an XML broker and a Web item that represents a dataset in that XML broker's data packet.

Class Hierarchy

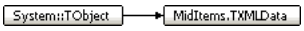

Pascal

```
TXMLData = class(TObject);
```

C++

```
class TXMLData : public TObject;
```

File

MidItems

Description

TXMLData represents the link to an XML broker for a Web item that is bound to a dataset in the XML broker's data packet. It is the base class for TXMLDataSet (see page 17006), which allows the Web item to bind to a dataset field nested in the XML broker's top-level dataset.

TXMLData does not support the ability to bind to a nested dataset field. Rather, it introduces the underlying support needed to keep track of the XML broker for components that supply this ability.

See Also

TXMLBroker (see page 26049)

IDataSetComponent (see page 17053)

TXMLDataSet (see page 17006)

99.1.62.1 TXMLData Members

The following tables list the members exposed by TXMLData.

Protected Methods

	Name	Description
	Changed (see page 17000)	This is Changed, a member of class TXMLData.
	ChangedXMLBroker (see page 17000)	This is ChangedXMLBroker, a member of class TXMLData.
	GetXMLBroker (see page 17000)	This is GetXMLBroker, a member of class TXMLData.
	SetXMLBroker (see page 17001)	This is SetXMLBroker, a member of class TXMLData.

Public Events

	Name	Description
	OnChange (see page 17002)	Occurs when the XMLBroker (see page 17002) property changes.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 16999)	Creates a new TXMLData (see page 16998) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Notification (see page 17001)	Responds to notifications when objects are created or destroyed.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Parent (see page 17001)	Indicates the Web item that uses this object to represent its link to an XML broker.
	XMLBroker (see page 17002)	Specifies the XML broker that generates the dataset to which the associated Web item is bound.

99.1.62.2 TXMLData.Create Constructor

Creates a new TXMLData (see page 16998) object.

Pascal

```
constructor Create(AParent: TComponent);
```

C++

```
__fastcall TXMLData(TComponent * AParent);
```

Description

Most applications do not need to call the TXMLData (see page 16998) constructor. The Web item that uses the TXMLData (see page 16998) object to represent its link to a dataset instantiates this object.

The AParent parameter specifies the Web item that uses the TXMLData (see page 16998) object to link to an XML broker. This becomes the value of the Parent (see page 17001) property.

See Also

Parent (see page 17001)

99.1.62.3 TXMLData Methods

The methods of the TXMLData class are listed here.

Protected Methods

	Name	Description
	Changed (see page 17000)	This is Changed, a member of class TXMLData.
V	ChangedXMLBroker (see page 17000)	This is ChangedXMLBroker, a member of class TXMLData.
V	GetXMLBroker (see page 17000)	This is GetXMLBroker, a member of class TXMLData.
V	SetXMLBroker (see page 17001)	This is SetXMLBroker, a member of class TXMLData.

Public Methods

	Name	Description
V	Notification (see page 17001)	Responds to notifications when objects are created or destroyed.

99.1.62.3.1 TXMLData.Changed Method

Pascal

```
procedure Changed;
```

C++

```
__fastcall Changed();
```

Description

This is Changed, a member of class TXMLData.

99.1.62.3.2 TXMLData.ChangedXMLBroker Method

Pascal

```
procedure ChangedXMLBroker; virtual;
```

C++

```
virtual __fastcall ChangedXMLBroker();
```

Description

This is ChangedXMLBroker, a member of class TXMLData.

99.1.62.3.3 TXMLData.GetXMLBroker Method

Pascal

```
function GetXMLBroker: TXMLBroker; virtual;
```

C++

```
virtual __fastcall TXMLBroker GetXMLBroker();
```

Description

This is GetXMLBroker, a member of class TXMLData.

99.1.62.3.4 TXMLData.Notification Method

Responds to notifications when objects are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); virtual;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method. It is called automatically when objects are created or destroyed. TXMLData (see page 16998) responds to notifications by setting the XMLBroker (see page 17002) property to nil (Delphi) or NULL (C++) if it is notified that this component is about to be destroyed.

See Also

XMLBroker (see page 17002)

99.1.62.3.5 TXMLData.SetXMLBroker Method

Pascal

```
procedure SetXMLBroker(const Value: TXMLBroker); virtual;
```

C++

```
virtual __fastcall SetXMLBroker(const TXMLBroker Value);
```

Description

This is SetXMLBroker, a member of class TXMLData.

99.1.62.4 TXMLData Properties

The properties of the TXMLData class are listed here.

Public Properties

	Name	Description
R	Parent (see page 17001)	Indicates the Web item that uses this object to represent its link to an XML broker.
	XMLBroker (see page 17002)	Specifies the XML broker that generates the dataset to which the associated Web item is bound.

99.1.62.4.1 TXMLData.Parent Property

Indicates the Web item that uses this object to represent its link to an XML broker.

Pascal

```
property Parent: TComponent;
```

C++

```
__property TComponent * Parent;
```

Description

Read Parent to determine the Web item that uses this object to bind to a dataset from an XML broker. The value of Parent is set by the constructor.

See Also

XMLBroker (see page 17002)

Create (see page 16999)

99.1.62.4.2 TXMLData.XMLBroker Property

Specifies the XML broker that generates the dataset to which the associated Web item is bound.

Pascal

```
property XMLBroker: TXMLBroker;
```

C++

```
__property TXMLBroker XMLBroker;
```

Description

The Web item specified by the Parent (see page 17001) property sets XMLBroker to bind to an XML broker.

When the value of XMLBroker changes, the TXMLData (see page 16998) object generates an OnChange (see page 17002) event.

See Also

Parent (see page 17001)

OnChange (see page 17002)

99.1.62.5 TXMLData Events

The events of the TXMLData class are listed here.

Public Events

	Name	Description
	OnChange (see page 17002)	Occurs when the XMLBroker (see page 17002) property changes.

99.1.62.5.1 TXMLData.OnChange Event

Occurs when the XMLBroker (see page 17002) property changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to respond immediately after the Web item specified by Parent (see page 17001) becomes bound to a different XML broker. This occurs when Parent (see page 17001) changes the XMLBroker (see page 17002) property explicitly in response to a change in its own properties, or when the XML broker object to which it is bound is destroyed.

See Also

XMLBroker (see page 17002)

Parent (see page 17001)

99.1.63 TXMLDataParent Class

Class Hierarchy

System::TObject → MidItems.TXMLData → MidItems.TXMLDataParent

Pascal

```
TXMLDataParent = class(TXMLData);
```

C++

```
class TXMLDataParent : public TXMLData;
```

File

MidItems

Description

This is class MidItems.TXMLDataParent.

99.1.63.1 TXMLDataParent Members

The following tables list the members exposed by TXMLDataParent.

Protected Methods

	Name	Description
	Changed (see page 17000)	This is Changed, a member of class TXMLData.
V	ChangedUseParent (see page 17004)	This is ChangedUseParent, a member of class TXMLDataParent.
V	ChangedXMLBroker (see page 17005)	This is ChangedXMLBroker, a member of class TXMLDataParent.
V	GetXMLBroker (see page 17005)	This is GetXMLBroker, a member of class TXMLDataParent.
	ParentXMLBroker (see page 17005)	This is ParentXMLBroker, a member of class TXMLDataParent.
	SetUseParent (see page 17005)	This is SetUseParent, a member of class TXMLDataParent.
V	SetXMLBroker (see page 17001)	This is SetXMLBroker, a member of class TXMLData.

Public Events

	Name	Description
	OnChange (see page 17002)	Occurs when the XMLBroker (see page 17002) property changes.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 16999)	Creates a new TXMLData (see page 16998) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Notification (see page 17001)	Responds to notifications when objects are created or destroyed.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Parent (see page 17001)	Indicates the Web item that uses this object to represent its link to an XML broker.
	XMLBroker (see page 17002)	Specifies the XML broker that generates the dataset to which the associated Web item is bound.

Published Properties

	Name	Description
	UseParent (see page 17006)	This is UseParent, a member of class TXMLDataParent.

99.1.63.2 TXMLDataParent Methods

The methods of the TXMLDataParent class are listed here.

Protected Methods

	Name	Description
V	ChangedUseParent (see page 17004)	This is ChangedUseParent, a member of class TXMLDataParent.
V	ChangedXMLBroker (see page 17005)	This is ChangedXMLBroker, a member of class TXMLDataParent.
V	GetXMLBroker (see page 17005)	This is GetXMLBroker, a member of class TXMLDataParent.
	ParentXMLBroker (see page 17005)	This is ParentXMLBroker, a member of class TXMLDataParent.
	SetUseParent (see page 17005)	This is SetUseParent, a member of class TXMLDataParent.

99.1.63.2.1 TXMLDataParent.ChangedUseParent Method

Pascal

```
procedure ChangedUseParent; virtual;
```

C++

```
virtual __fastcall ChangedUseParent();
```

Description

This is ChangedUseParent, a member of class TXMLDataParent.

99.1.63.2.2 TXMLDataParent.ChangedXMLBroker Method

Pascal

```
procedure ChangedXMLBroker; override;
```

C++

```
virtual __fastcall ChangedXMLBroker();
```

Description

This is ChangedXMLBroker, a member of class TXMLDataParent.

99.1.63.2.3 TXMLDataParent.GetXMLBroker Method

Pascal

```
function GetXMLBroker: TXMLBroker; override;
```

C++

```
virtual __fastcall TXMLBroker GetXMLBroker();
```

Description

This is GetXMLBroker, a member of class TXMLDataParent.

99.1.63.2.4 TXMLDataParent.ParentXMLBroker Method

Pascal

```
function ParentXMLBroker: TXMLBroker;
```

C++

```
__fastcall TXMLBroker ParentXMLBroker();
```

Description

This is ParentXMLBroker, a member of class TXMLDataParent.

99.1.63.2.5 TXMLDataParent.SetUseParent Method

Pascal

```
procedure SetUseParent(const Value: Boolean);
```

C++

```
__fastcall SetUseParent(const Boolean Value);
```

Description

This is SetUseParent, a member of class TXMLDataParent.

99.1.63.3 TXMLDataParent Properties

The properties of the TXMLDataParent class are listed here.

Published Properties

	Name	Description
	UseParent (see page 17006)	This is UseParent, a member of class TXMLDataParent.

99.1.63.3.1 TXMLDataParent.UseParent Property

Pascal

```
property UseParent: Boolean;
```

C++

```
__property Boolean UseParent;
```

Description

This is UseParent, a member of class TXMLDataParent.

99.1.64 TXMLDataSet Class

TXMLDataSet manages the relationship between a Web item and the XML dataset to which it is bound.

Class Hierarchy

Pascal

```
TXMLDataSet = class(TXMLData, INotifyConnectionChange);
```

C++

```
class TXMLDataSet : public TXMLData, public INotifyConnectionChange;
```

File

MidItems

Description

TXMLDataSet represents a Web item's link to an XML dataset. The Web item can be bound to an XML broker's data packet, or to a dataset field nested in an XML broker's data packet.

See Also

TXMLBroker (see page 26049)

IDataSetComponent (see page 17053)

TXMLDisplayGroup (see page 17019)

99.1.64.1 TXMLDataSet Members

The following tables list the members exposed by TXMLDataSet.

Protected Methods

	Name	Description
	_AddRef (see page 17009)	This is _AddRef, a member of class TXMLDataSet.

	_Release (see page 17009)	This is _Release, a member of class TXMLDataSet.
	Changed (see page 17000)	This is Changed, a member of class TXMLData.
V	ChangedDataSetField (see page 17009)	This is ChangedDataSetField, a member of class TXMLDataSet.
V	ChangedXMLBroker (see page 17010)	This is ChangedXMLBroker, a member of class TXMLDataSet.
	CompareXMLData (see page 17010)	This is CompareXMLData, a member of class TXMLDataSet.
	ConnectionChange (see page 17010)	This is ConnectionChange, a member of class TXMLDataSet.
	CreateDataSet (see page 17010)	This is CreateDataSet, a member of class TXMLDataSet.
	FindDataSet (see page 17010)	This is FindDataSet, a member of class TXMLDataSet.
	GetDataSet (see page 17011)	This is GetDataSet, a member of class TXMLDataSet.
V	GetDataSetField (see page 17011)	This is GetDataSetField, a member of class TXMLDataSet.
	GetDataSetPath (see page 17011)	This is GetDataSetPath, a member of class TXMLDataSet.
V	GetXMLBroker (see page 17000)	This is GetXMLBroker, a member of class TXMLData.
V	QueryInterface (see page 17011)	This is QueryInterface, a member of class TXMLDataSet.
	SetDataSetField (see page 17012)	This is SetDataSetField, a member of class TXMLDataSet.
V	SetXMLBroker (see page 17012)	This is SetXMLBroker, a member of class TXMLDataSet.
	WrapperDataChange (see page 17012)	This is WrapperDataChange, a member of class TXMLDataSet.

Protected Properties

	Name	Description
R	Parent (see page 17014)	Indicates the Web item that uses this object to represent its link to an XML broker.

Public Events

	Name	Description
	OnChange (see page 17002)	Occurs when the XMLBroker (see page 17002) property changes.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 17008)	Creates a new TXMLDataSet (see page 17006) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 17008)	TXMLDataSetFrees the memory associated with the TXMLDataSet (see page 17006) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Notification (see page 17001)	Responds to notifications when objects are created or destroyed.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	DataSet (see page 17012)	Provides access to an interface for working with the associated dataset.
	DataSetField (see page 17013)	Identifies a nested dataset in the associated XML broker's data packet.
R	DataSetPath (see page 17013)	Represents the parsed value of the DataSetField (see page 17013) property.
	XMLBroker (see page 17002)	Specifies the XML broker that generates the dataset to which the associated Web item is bound.

99.1.64.2 TXMLDataSet.Create Constructor

Creates a new TXMLDataSet (see page 17006) object.

Pascal

```
constructor Create(AParent: TComponent);
```

C++

```
__fastcall TXMLDataSet(TComponent * AParent);
```

Description

Most applications do not need to call the TXMLDataSet (see page 17006) constructor. The Web item that uses the TXMLDataSet (see page 17006) object to represent its link to a dataset instantiates this object.

The AParent parameter specifies the Web item that uses the TXMLDataSet (see page 17006) object to link to an XML dataset. This becomes the value of the Parent (see page 17014) property.

See Also

Parent (see page 17001)

99.1.64.3 TXMLDataSet.Destroy Destructor

TXMLDataSetFrees the memory associated with the TXMLDataSet (see page 17006) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLDataSet();
```

Description

Typically, applications do not free instances of TXMLDataSet (see page 17006). Instead, this object is freed by the Web item that uses it.

99.1.64.4 TXMLDataSet Methods

The methods of the TXMLDataSet class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 17009)	This is _AddRef, a member of class TXMLDataSet.
	_Release (see page 17009)	This is _Release, a member of class TXMLDataSet.
V	ChangedDataSetField (see page 17009)	This is ChangedDataSetField, a member of class TXMLDataSet.
V	ChangedXMLBroker (see page 17010)	This is ChangedXMLBroker, a member of class TXMLDataSet.
	CompareXMLData (see page 17010)	This is CompareXMLData, a member of class TXMLDataSet.
	ConnectionChange (see page 17010)	This is ConnectionChange, a member of class TXMLDataSet.
	CreateDataSet (see page 17010)	This is CreateDataSet, a member of class TXMLDataSet.
	FindDataSet (see page 17010)	This is FindDataSet, a member of class TXMLDataSet.
	GetDataSet (see page 17011)	This is GetDataSet, a member of class TXMLDataSet.
V	GetDataSetField (see page 17011)	This is GetDataSetField, a member of class TXMLDataSet.
	GetDataSetPath (see page 17011)	This is GetDataSetPath, a member of class TXMLDataSet.
V	QueryInterface (see page 17011)	This is QueryInterface, a member of class TXMLDataSet.
	SetDataSetField (see page 17012)	This is SetDataSetField, a member of class TXMLDataSet.
V	SetXMLBroker (see page 17012)	This is SetXMLBroker, a member of class TXMLDataSet.
	WrapperDataChange (see page 17012)	This is WrapperDataChange, a member of class TXMLDataSet.

99.1.64.4.1 TXMLDataSet._AddRef Method

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

This is _AddRef, a member of class TXMLDataSet.

99.1.64.4.2 TXMLDataSet._Release Method

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

This is _Release, a member of class TXMLDataSet.

99.1.64.4.3 TXMLDataSet.ChangedDataSetField Method

Pascal

```
procedure ChangedDataSetField; virtual;
```

C++

```
virtual __fastcall ChangedDataSetField();
```

Description

This is ChangedDataSetField, a member of class TXMLDataSet.

99.1.64.4.4 TXMLDataSet.ChangedXMLBroker Method

Pascal

```
procedure ChangedXMLBroker; override;
```

C++

```
virtual __fastcall ChangedXMLBroker();
```

Description

This is ChangedXMLBroker, a member of class TXMLDataSet.

99.1.64.4.5 TXMLDataSet.CompareXMLData Method

Pascal

```
function CompareXMLData(XMLDataSet: TXMLDataSet): Boolean;
```

C++

```
__fastcall Boolean CompareXMLData(TXMLDataSet XMLDataSet);
```

Description

This is CompareXMLData, a member of class TXMLDataSet.

99.1.64.4.6 TXMLDataSet.ConnectionChange Method

Pascal

```
procedure ConnectionChange(Sender: TComponent; Connecting: Boolean);
```

C++

```
__fastcall ConnectionChange(TComponent * Sender, Boolean Connecting);
```

Description

This is ConnectionChange, a member of class TXMLDataSet.

99.1.64.4.7 TXMLDataSet.CreateDataSet Method

Pascal

```
function CreateDataSet: IInetXWebDataSet;
```

C++

```
__fastcall IInetXWebDataSet CreateDataSet();
```

Description

This is CreateDataSet, a member of class TXMLDataSet.

99.1.64.4.8 TXMLDataSet.FindDataSet Method

Pascal

```
function FindDataSet: IInetXWebDataSet;
```

C++

```
__fastcall IInetXWebDataSet FindDataSet();
```

Description

This is FindDataSet, a member of class TXMLDataSet.

99.1.64.4.9 TXMLDataSet.GetDataSet Method

Pascal

```
function GetDataSet: IInetXWebDataSet;
```

C++

```
__fastcall IInetXWebDataSet GetDataSet();
```

Description

This is GetDataSet, a member of class TXMLDataSet.

99.1.64.4.10 TXMLDataSet.GetDataSetField Method

Pascal

```
function GetDataSetField: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetDataSetField();
```

Description

This is GetDataSetField, a member of class TXMLDataSet.

99.1.64.4.11 TXMLDataSet.GetDataSetPath Method

Pascal

```
function GetDataSetPath: TStrings;
```

C++

```
__fastcall TStrings GetDataSetPath();
```

Description

This is GetDataSetPath, a member of class TXMLDataSet.

99.1.64.4.12 TXMLDataSet.QueryInterface Method

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

This is QueryInterface, a member of class TXMLDataSet.

99.1.64.4.13 TXMLDataSet.SetDataSetField Method

Pascal

```
procedure SetDataSetField(const Value: string);
```

C++

```
__fastcall SetDataSetField(const AnsiString Value);
```

Description

This is SetDataSetField, a member of class TXMLDataSet.

99.1.64.4.14 TXMLDataSet.SetXMLBroker Method

Pascal

```
procedure SetXMLBroker(const Value: TXMLBroker); override;
```

C++

```
virtual __fastcall SetXMLBroker(const TXMLBroker Value);
```

Description

This is SetXMLBroker, a member of class TXMLDataSet.

99.1.64.4.15 TXMLDataSet.WrapperDataChange Method

Pascal

```
procedure WrapperDataChange(Sender: TObject);
```

C++

```
__fastcall WrapperDataChange(TObject * Sender);
```

Description

This is WrapperDataChange, a member of class TXMLDataSet.

99.1.64.5 TXMLDataSet Properties

The properties of the TXMLDataSet class are listed here.

Protected Properties

	Name	Description
R	Parent (see page 17014)	Indicates the Web item that uses this object to represent its link to an XML broker.

Public Properties

	Name	Description
R	DataSet (see page 17012)	Provides access to an interface for working with the associated dataset.
	DataSetField (see page 17013)	Identifies a nested dataset in the associated XML broker's data packet.
R	DataSetPath (see page 17013)	Represents the parsed value of the DataSetField (see page 17013) property.

99.1.64.5.1 TXMLDataSet.DataSet Property

Provides access to an interface for working with the associated dataset.

Pascal

```
property DataSet: IInetXWebDataSet;
```

C++

```
__property IInetXWebDataSet DataSet;
```

Description

Use DataSet to manipulate the associated dataset. DataSet is an interface that provides information about the available fields and parameters of the dataset and lets you respond when this information changes.

An advantage of using the DataSet property is that it handles the task of identifying the correct dataset within the associated XML broker's data packet. This lets you treat nested datasets in the same way that you interact with the XML broker's main dataset.

See Also

XMLBroker

DataSetField (see page 17013)

99.1.64.5.2 TXMLDataSet.DataSetField Property

Identifies a nested dataset in the associated XML broker's data packet.

Pascal

```
property DataSetField: string;
```

C++

```
__property AnsiString DataSetField;
```

Description

Use DataSetField, in conjunction with the XMLBroker (see page 17002) property, to identify the XML dataset that the TXMLDataSet (see page 17006) object represents. If the dataset is the dataset represented directly by the XML broker's data packet, DataSetField is an empty string and the XMLBroker (see page 17002) property fully describes the dataset. If the dataset is a nested detail set, DataSetField indicates what detail dataset to use from the data packet associated with XMLBroker (see page 17002).

Specify the name of the dataset field as the value of DataSetField. If the dataset is the detail set of a detail set, you can use periods to qualify dataset field names. Thus, for example, if XMLBroker (see page 17002) represents a customer dataset with a nested Orders dataset field that has its own nested LineItems field, you can specify the LineItems dataset by setting DataSetField to

```
Orders.LineItems
```

See Also

XMLBroker

DataSet (see page 17012)

DataSetPath (see page 17013)

99.1.64.5.3 TXMLDataSet.DataSetPath Property

Represents the parsed value of the DataSetField (see page 17013) property.

Pascal

```
property DataSetPath: TStrings;
```

C++

```
__property TStrings DataSetPath;
```

Description

DataSetPath is a parsed version of the DataSetField (see page 17013) property, where each dataset field is a separate string in the TStrings object. The order of the strings indicates the level of nesting, where the first string names a dataset field in the XML broker's data packet, the second string names a dataset field in that nested dataset, and so on.

See Also

XMLBroker

DataSetField (see page 17013)

99.1.64.5.4 TXMLDataSet.Parent Property

Indicates the Web item that uses this object to represent its link to an XML broker.

Pascal

```
property Parent: TComponent;
```

C++

```
__property TComponent * Parent;
```

Description

Read Parent to determine the Web item that uses this object to bind to a dataset from an XML broker. The value of Parent is set by the constructor.

See Also

XMLBroker (see page 17002)

Create (see page 16999)

99.1.65 TXMLDataSetParent Class

Class Hierarchy

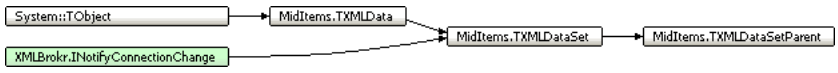

Pascal

```
TXMLDataSetParent = class(TXMLDataSet);
```

C++

```
class TXMLDataSetParent : public TXMLDataSet;
```

File

MidItems

Description

This is class MidItems.TXMLDataSetParent.

99.1.65.1 TXMLDataSetParent Members

The following tables list the members exposed by TXMLDataSetParent.

Protected Methods

	Name	Description
	_AddRef (see page 17009)	This is _AddRef, a member of class TXMLDataSet.
	_Release (see page 17009)	This is _Release, a member of class TXMLDataSet.
	Changed (see page 17000)	This is Changed, a member of class TXMLData.
V	ChangedDataSetField (see page 17017)	This is ChangedDataSetField, a member of class TXMLDataSetParent.
V	ChangedUseParent (see page 17017)	This is ChangedUseParent, a member of class TXMLDataSetParent.
V	ChangedXMLBroker (see page 17017)	This is ChangedXMLBroker, a member of class TXMLDataSetParent.
	CompareXMLData (see page 17010)	This is CompareXMLData, a member of class TXMLDataSet.
	ConnectionChange (see page 17010)	This is ConnectionChange, a member of class TXMLDataSet.
	CreateDataSet (see page 17010)	This is CreateDataSet, a member of class TXMLDataSet.
	FindDataSet (see page 17010)	This is FindDataSet, a member of class TXMLDataSet.
	GetDataSet (see page 17011)	This is GetDataSet, a member of class TXMLDataSet.
V	GetDataSetField (see page 17017)	This is GetDataSetField, a member of class TXMLDataSetParent.
	GetDataSetPath (see page 17011)	This is GetDataSetPath, a member of class TXMLDataSet.
V	GetXMLBroker (see page 17017)	This is GetXMLBroker, a member of class TXMLDataSetParent.
	ParentDataSetField (see page 17018)	This is ParentDataSetField, a member of class TXMLDataSetParent.
	ParentXMLBroker (see page 17018)	This is ParentXMLBroker, a member of class TXMLDataSetParent.
	ParentXMLDataSet (see page 17018)	This is ParentXMLDataSet, a member of class TXMLDataSetParent.
V	QueryInterface (see page 17011)	This is QueryInterface, a member of class TXMLDataSet.
	SetDataSetField (see page 17012)	This is SetDataSetField, a member of class TXMLDataSet.
	SetUseParent (see page 17018)	This is SetUseParent, a member of class TXMLDataSetParent.
V	SetXMLBroker (see page 17001)	This is SetXMLBroker, a member of class TXMLData.
V	SetXMLBroker (see page 17012)	This is SetXMLBroker, a member of class TXMLDataSet.
	WrapperDataChange (see page 17012)	This is WrapperDataChange, a member of class TXMLDataSet.

Protected Properties

	Name	Description
R	Parent (see page 17014)	Indicates the Web item that uses this object to represent its link to an XML broker.
	UseParent (see page 17019)	This is UseParent, a member of class TXMLDataSetParent.

Public Events

	Name	Description
	OnChange (see page 17002)	Occurs when the XMLBroker (see page 17002) property changes.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ConnectionChange (see page 26087)	This is ConnectionChange, a member of class INotifyConnectionChange.
	Create (see page 16999)	Creates a new TXMLData (see page 16998) object.
	Create (see page 17008)	Creates a new TXMLDataSet (see page 17006) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 17008)	TXMLDataSetFrees the memory associated with the TXMLDataSet (see page 17006) object.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Notification (see page 17001)	Responds to notifications when objects are created or destroyed.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	DataSet (see page 17012)	Provides access to an interface for working with the associated dataset.
	DataSetField (see page 17013)	Identifies a nested dataset in the associated XML broker's data packet.
R	DataSetPath (see page 17013)	Represents the parsed value of the DataSetField (see page 17013) property.
R	Parent (see page 17001)	Indicates the Web item that uses this object to represent its link to an XML broker.
	XMLBroker (see page 17002)	Specifies the XML broker that generates the dataset to which the associated Web item is bound.

99.1.65.2 TXMLDataSetParent Methods

The methods of the TXMLDataSetParent class are listed here.

Protected Methods

	Name	Description
V	ChangedDataSetField (see page 17017)	This is ChangedDataSetField, a member of class TXMLDataSetParent.
V	ChangedUseParent (see page 17017)	This is ChangedUseParent, a member of class TXMLDataSetParent.
V	ChangedXMLBroker (see page 17017)	This is ChangedXMLBroker, a member of class TXMLDataSetParent.
V	GetDataSetField (see page 17017)	This is GetDataSetField, a member of class TXMLDataSetParent.
V	GetXMLBroker (see page 17017)	This is GetXMLBroker, a member of class TXMLDataSetParent.
	ParentDataSetField (see page 17018)	This is ParentDataSetField, a member of class TXMLDataSetParent.
	ParentXMLBroker (see page 17018)	This is ParentXMLBroker, a member of class TXMLDataSetParent.
	ParentXMLDataSet (see page 17018)	This is ParentXMLDataSet, a member of class TXMLDataSetParent.
	SetUseParent (see page 17018)	This is SetUseParent, a member of class TXMLDataSetParent.

99.1.65.2.1 TXMLDataSetParent.ChangedDataSetField Method

Pascal

```
procedure ChangedDataSetField; override;
```

C++

```
virtual __fastcall ChangedDataSetField();
```

Description

This is ChangedDataSetField, a member of class TXMLDataSetParent.

99.1.65.2.2 TXMLDataSetParent.ChangedUseParent Method

Pascal

```
procedure ChangedUseParent; virtual;
```

C++

```
virtual __fastcall ChangedUseParent();
```

Description

This is ChangedUseParent, a member of class TXMLDataSetParent.

99.1.65.2.3 TXMLDataSetParent.ChangedXMLBroker Method

Pascal

```
procedure ChangedXMLBroker; override;
```

C++

```
virtual __fastcall ChangedXMLBroker();
```

Description

This is ChangedXMLBroker, a member of class TXMLDataSetParent.

99.1.65.2.4 TXMLDataSetParent.GetDataSetField Method

Pascal

```
function GetDataSetField: string; override;
```

C++

```
virtual __fastcall AnsiString GetDataSetField();
```

Description

This is GetDataSetField, a member of class TXMLDataSetParent.

99.1.65.2.5 TXMLDataSetParent.GetXMLBroker Method

Pascal

```
function GetXMLBroker: TXMLBroker; override;
```

C++

```
virtual __fastcall TXMLBroker GetXMLBroker();
```

Description

This is GetXMLBroker, a member of class TXMLDataSetParent.

99.1.65.2.6 TXMLDataSetParent.ParentDataSetField Method

Pascal

```
function ParentDataSetField: string;
```

C++

```
__fastcall AnsiString ParentDataSetField();
```

Description

This is ParentDataSetField, a member of class TXMLDataSetParent.

99.1.65.2.7 TXMLDataSetParent.ParentXMLBroker Method

Pascal

```
function ParentXMLBroker: TXMLBroker;
```

C++

```
__fastcall TXMLBroker ParentXMLBroker();
```

Description

This is ParentXMLBroker, a member of class TXMLDataSetParent.

99.1.65.2.8 TXMLDataSetParent.ParentXMLDataSet Method

Pascal

```
function ParentXMLDataSet: TXMLDataSet;
```

C++

```
__fastcall TXMLDataSet ParentXMLDataSet();
```

Description

This is ParentXMLDataSet, a member of class TXMLDataSetParent.

99.1.65.2.9 TXMLDataSetParent.SetUseParent Method

Pascal

```
procedure SetUseParent(const Value: Boolean);
```

C++

```
__fastcall SetUseParent(const Boolean Value);
```

Description

This is SetUseParent, a member of class TXMLDataSetParent.

99.1.65.3 TXMLDataSetParent Properties

The properties of the TXMLDataSetParent class are listed here.

Protected Properties

	Name	Description
	UseParent (see page 17019)	This is UseParent, a member of class TXMLDataSetParent.

99.1.65.3.1 TXMLDataSetParent.UseParent Property

Pascal

```
property UseParent: Boolean;
```

C++

```
__property Boolean UseParent;
```

Description

This is UseParent, a member of class TXMLDataSetParent.

99.1.66 TXMLDisplayGroup Class

TXMLDisplayGroup is the base class for Web items that arrange a set of special-purpose controls in an HTML form.

Class Hierarchy

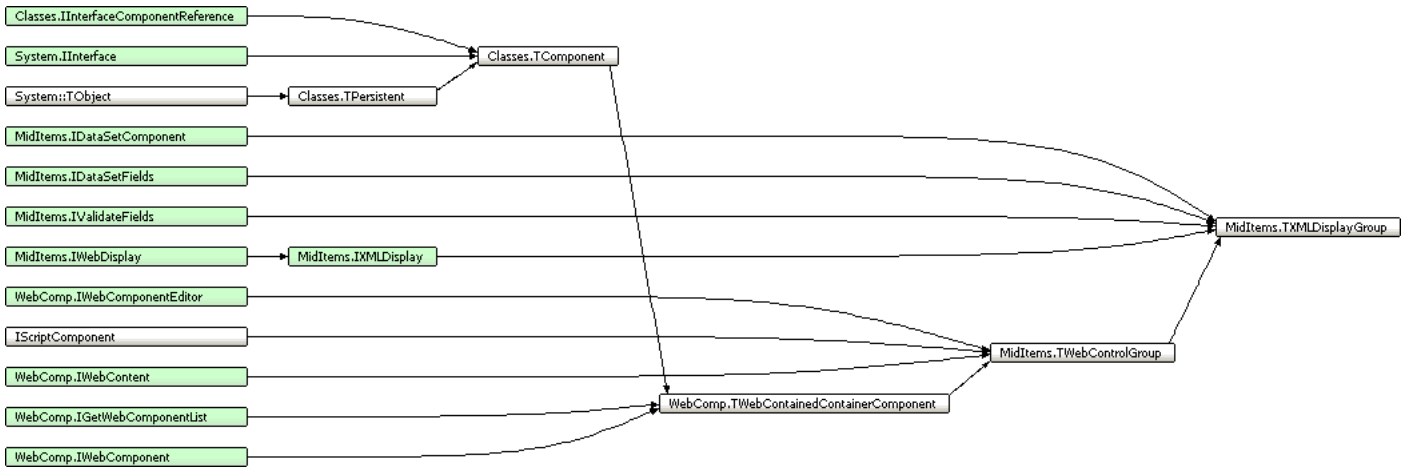

Pascal

```
TXMLDisplayGroup = class(TWebControlGroup, IDataSetComponent, IDataSetFields, IXMLDisplay,
IValidateFields);
```

C++

```
class TXMLDisplayGroup : public TWebControlGroup, public IDataSetComponent, public
IDataSetFields, public IXMLDisplay, public IValidateFields;
```

File

MidItems

Description

Use TXMLDisplayGroup as a base class for Web items that generate the HTML to arrange a collection of HTML controls. Each TXMLDisplayGroup object maintains a list of Web items that display information in the final HTML document. The XML display group arranges the controls that these items generate.

If the controls generated for TXMLDisplayGroup by its Web items represent database information, you can bind the XML display group to an XML data packet or dataset field within an XML data packet. The Web items can use that binding to determine the source of the data they display.

In order to let other Web items use the HTML that the TXMLDisplayGroup object generates, TXMLDisplayGroup inherits the IWebContent interface from TWebControlGroup (see page 16872).

Do not create instances of TXMLDisplayGroup. TXMLDisplayGroup manages the Web items that generate display controls, but does not produce any content. Instead, use a descendant of TXMLDisplayGroup that generates the HTML for its set of specialized controls such as TDataGrid (see page 16560), TFieldGroup (see page 16591) or TQueryFieldGroup (see page 16736).

See Also

TDataGrid (see page 16560)

TFieldGroup (see page 16591)

TQueryFieldGroup (see page 16736)

IWebContent (see page 24792)

99.1.66.1 TXMLDisplayGroup Members

The following tables list the members exposed by TXMLDisplayGroup.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
	CreateDefaultFields (see page 17025)	This is CreateDefaultFields, a member of class TXMLDisplayGroup.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FindField (see page 17026)	This is FindField, a member of class TXMLDisplayGroup.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDataSet (see page 17026)	Returns the value of the DataSet property.
	GetDataSetField (see page 17026)	This is GetDataSetField, a member of class TXMLDisplayGroup.

	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetEnableValidateFields (see page 17026)	This is GetEnableValidateFields, a member of class TXMLDisplayGroup.
	GetFieldsList (see page 17027)	This is GetFieldsList, a member of class TXMLDisplayGroup.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIsMultipleRecordView (see page 17027)	This is GetIsMultipleRecordView, a member of class TXMLDisplayGroup.
	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVisibleFields (see page 17027)	This is GetVisibleFields, a member of class TXMLDisplayGroup.
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	GetXMLBroker (see page 17027)	Returns the value of the XMLBroker property.
	GetXMLDataSet (see page 17028)	Returns the value of the XMLDataSet property.
	GetXMLDisplayName (see page 17028)	This is GetXMLDisplayName, a member of class TXMLDisplayGroup.
	GetXMLRowSetName (see page 17028)	This is GetXMLRowSetName, a member of class TXMLDisplayGroup.
	HasStatusField (see page 17028)	This is HasStatusField, a member of class TXMLDisplayGroup.
	ImplAddElements (see page 17029)	Provides the underlying implementation for the AddElements method.
	ImplCanAddClass (see page 16881)	Provides the underlying implementation for the CanAddClass (see page 16878) method.
	ImplContent (see page 16882)	Provides the underlying implementation for the Content (see page 16879) method.
	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	ImplGetSubComponents (see page 17029)	Provides the underlying implementation for the GetSubComponents method.
	ImplIsMultipleRecordView (see page 17029)	This is ImplIsMultipleRecordView, a member of class TXMLDisplayGroup.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsFieldInUse (see page 17030)	This is IsFieldInUse, a member of class TXMLDisplayGroup.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 17030)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
	SetDataSetField (see page 17030)	This is SetDataSetField, a member of class TXMLDisplayGroup.
	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEnableValidateFields (see page 17031)	This is SetEnableValidateFields, a member of class TXMLDisplayGroup.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.

V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	SetXMLBroker (see page 17031)	This is SetXMLBroker, a member of class TXMLDisplayGroup.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateDataSet (see page 17031)	This is ValidateDataSet, a member of class TXMLDisplayGroup.
	ValidateFields (see page 17031)	This is ValidateFields, a member of class TXMLDisplayGroup.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.
V	XMLDataChange (see page 17031)	This is XMLDataChange, a member of class TXMLDisplayGroup.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
	EnableValidateFields (see page 17032)	This is EnableValidateFields, a member of class TXMLDisplayGroup.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.
R	XMLDataSet (see page 17033)	Indicates the TXMLDataSet object that represents the dataset to which the Web item is bound.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 17024)	Creates and initializes an instance of TXMLDisplayGroup (see page 17019).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 17024)	Frees an instance of TXMLDisplayGroup (see page 17019).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
R	DataSet (see page 17055)	Provides access to an interface for working with the associated dataset.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsMultipleRecordView (see page 17080)	
R	IsMultipleRecordView (see page 17083)	
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleFields (see page 17032)	Indicates the controls displayed by the HTML that the XML display group generates.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.
R	XMLDisplayName (see page 17084)	This is XMLDisplayName, a member of class IXMLDisplay.
R	XMLRowSetName (see page 17084)	This is XMLRowSetName, a member of class IXMLDisplay.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XMLBroker (see page 17033)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
	XMLDataSetField (see page 17033)	Binds the XML display group to a nested dataset from the XML data packet.

99.1.66.2 TXMLDisplayGroup.Create Constructor

Creates and initializes an instance of TXMLDisplayGroup (see page 17019).

Pascal

```
constructor Create(AComponent: TComponent); override;
```

C++

```
virtual __fastcall TXMLDisplayGroup(TComponent * AComponent);
```

Description

Call Create to instantiate TXMLDisplayGroup (see page 17019) at runtime. TXMLDisplayGroup (see page 17019) components added at design time using the Web Page editor are instantiated automatically.

99.1.66.3 TXMLDisplayGroup.Destroy Destructor

Frees an instance of TXMLDisplayGroup (see page 17019).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLDisplayGroup();
```

Description

Do not call Destroy directly. Instead, call Free, which checks that the object is not nil before calling Destroy.

See Also

Free (see page 22300)

99.1.66.4 TXMLDisplayGroup Methods

The methods of the TXMLDisplayGroup class are listed here.

Protected Methods

	Name	Description
	CreateDefaultFields (see page 17025)	This is CreateDefaultFields, a member of class TXMLDisplayGroup.
	FindField (see page 17026)	This is FindField, a member of class TXMLDisplayGroup.
	GetDataSet (see page 17026)	Returns the value of the DataSet property.
	GetDataSetField (see page 17026)	This is GetDataSetField, a member of class TXMLDisplayGroup.
	GetEnableValidateFields (see page 17026)	This is GetEnableValidateFields, a member of class TXMLDisplayGroup.
	GetFieldsList (see page 17027)	This is GetFieldsList, a member of class TXMLDisplayGroup.
	GetIsMultipleRecordView (see page 17027)	This is GetIsMultipleRecordView, a member of class TXMLDisplayGroup.
	GetVisibleFields (see page 17027)	This is GetVisibleFields, a member of class TXMLDisplayGroup.
	GetXMLBroker (see page 17027)	Returns the value of the XMLBroker property.
	GetXMLDataSet (see page 17028)	Returns the value of the XMLDataSet property.
	GetXMLDisplayName (see page 17028)	This is GetXMLDisplayName, a member of class TXMLDisplayGroup.
	GetXMLRowSetName (see page 17028)	This is GetXMLRowSetName, a member of class TXMLDisplayGroup.
	HasStatusField (see page 17028)	This is HasStatusField, a member of class TXMLDisplayGroup.
	ImplAddElements (see page 17029)	Provides the underlying implementation for the AddElements method.
	ImplGetSubComponents (see page 17029)	Provides the underlying implementation for the GetSubComponents method.
	ImplIsMultipleRecordView (see page 17029)	This is ImplIsMultipleRecordView, a member of class TXMLDisplayGroup.
	IsFieldInUse (see page 17030)	This is IsFieldInUse, a member of class TXMLDisplayGroup.
	Notification (see page 17030)	Forwards notification messages to all owned components.
	SetDataSetField (see page 17030)	This is SetDataSetField, a member of class TXMLDisplayGroup.
	SetEnableValidateFields (see page 17031)	This is SetEnableValidateFields, a member of class TXMLDisplayGroup.
	SetXMLBroker (see page 17031)	This is SetXMLBroker, a member of class TXMLDisplayGroup.
	ValidateDataSet (see page 17031)	This is ValidateDataSet, a member of class TXMLDisplayGroup.
	ValidateFields (see page 17031)	This is ValidateFields, a member of class TXMLDisplayGroup.
	XMLDataChange (see page 17031)	This is XMLDataChange, a member of class TXMLDisplayGroup.

99.1.66.4.1 TXMLDisplayGroup.CreateDefaultFields Method

Pascal

```
procedure CreateDefaultFields; virtual;
```

C++

```
virtual __fastcall CreateDefaultFields();
```

Description

This is CreateDefaultFields, a member of class TXMLDisplayGroup.

99.1.66.4.2 TXMLDisplayGroup.FindField Method

Pascal

```
function FindField(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindField(const AnsiString AName);
```

Description

This is FindField, a member of class TXMLDisplayGroup.

99.1.66.4.3 TXMLDisplayGroup.GetDataSet Method

Returns the value of the DataSet property.

Pascal

```
function GetDataSet: IInetXWebDataSet;
```

C++

```
__fastcall IInetXWebDataSet GetDataSet();
```

Description

GetDataSet is the read access method for the DataSet property.

See Also

DataSet (see page 17055)

99.1.66.4.4 TXMLDisplayGroup.GetDataSetField Method

Pascal

```
function GetDataSetField: string;
```

C++

```
__fastcall AnsiString GetDataSetField();
```

Description

This is GetDataSetField, a member of class TXMLDisplayGroup.

99.1.66.4.5 TXMLDisplayGroup.GetEnableValidateFields Method

Pascal

```
function GetEnableValidateFields: Boolean;
```

C++

```
__fastcall Boolean GetEnableValidateFields();
```

Description

This is GetEnableValidateFields, a member of class TXMLDisplayGroup.

99.1.66.4.6 TXMLDisplayGroup.GetFieldsList Method

Pascal

```
procedure GetFieldsList(List: TStrings); virtual;
```

C++

```
virtual __fastcall GetFieldsList(TStrings List);
```

Description

This is GetFieldsList, a member of class TXMLDisplayGroup.

99.1.66.4.7 TXMLDisplayGroup.GetIsMultipleRecordView Method

Pascal

```
function GetIsMultipleRecordView: Boolean;
```

C++

```
__fastcall Boolean GetIsMultipleRecordView();
```

Description

This is GetIsMultipleRecordView, a member of class TXMLDisplayGroup.

99.1.66.4.8 TXMLDisplayGroup.GetVisibleFields Method

Pascal

```
function GetVisibleFields: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleFields();
```

Description

This is GetVisibleFields, a member of class TXMLDisplayGroup.

99.1.66.4.9 TXMLDisplayGroup.GetXMLBroker Method

Returns the value of the XMLBroker property.

Pascal

```
function GetXMLBroker: TXMLBroker;
```

C++

```
__fastcall TXMLBroker GetXMLBroker();
```

Description

GetXMLBroker is the read access method for the XMLBroker property.

See Also

XMLBroker (see page 17056)

99.1.66.4.10 TXMLDisplayGroup.GetXMLDataSet Method

Returns the value of the XMLDataSet property.

Pascal

```
function GetXMLDataSet: TXMLDataSet;
```

C++

```
__fastcall TXMLDataSet GetXMLDataSet();
```

Description

GetXMLDataSet is the read access method for the XMLDataSet property.

See Also

XMLDataSet (see page 17056)

99.1.66.4.11 TXMLDisplayGroup.GetXMLDisplayName Method

Pascal

```
function GetXMLDisplayName: string;
```

C++

```
__fastcall AnsiString GetXMLDisplayName();
```

Description

This is GetXMLDisplayName, a member of class TXMLDisplayGroup.

99.1.66.4.12 TXMLDisplayGroup.GetXMLRowSetName Method

Pascal

```
function GetXMLRowSetName: string;
```

C++

```
__fastcall AnsiString GetXMLRowSetName();
```

Description

This is GetXMLRowSetName, a member of class TXMLDisplayGroup.

99.1.66.4.13 TXMLDisplayGroup.HasStatusField Method

Pascal

```
function HasStatusField: Boolean;
```

C++

```
__fastcall Boolean HasStatusField();
```

Description

This is HasStatusField, a member of class TXMLDisplayGroup.

99.1.66.4.14 TXMLDisplayGroup.ImplAddElements Method

Provides the underlying implementation for the AddElements method.

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); override;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

The AddElements method calls ImplAddElements to add any script declarations by calling the AddIntf interface. These declarations allow the generated display control to call script methods in response to user events by ensuring that the required methods are declared.

As implemented in TWebControlGroup, ImplAddElements does nothing. TWebControlGroup descendants override ImplAddElements to add any required declarations.

See Also

AddElements (see page 16878)

99.1.66.4.15 TXMLDisplayGroup.ImplGetSubComponents Method

Provides the underlying implementation for the GetSubComponents method.

Pascal

```
function ImplGetSubComponents: TObject; override;
```

C++

```
virtual __fastcall TObject * ImplGetSubComponents();
```

Description

The GetSubComponents method calls ImplGetSubComponents to retrieve the value of WebFieldControls as a TObject. TWebControlGroup descendants override this method if the list is not simply the value of WebFieldControls. For example, some TWebControlGroup descendants supply a default value (DefaultWebComponents) when the WebFieldControls property list is empty.

ImplGetSubComponents returns the TWebComponentList object that contains all subitems of the Web control group.

See Also

GetSubComponents (see page 16880)

DefaultWebComponents (see page 16883)

WebFieldControls (see page 16884)

99.1.66.4.16 TXMLDisplayGroup.ImplIsMultipleRecordView Method

Pascal

```
function ImplIsMultipleRecordView: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplIsMultipleRecordView();
```

Description

This is ImplIsMultipleRecordView, a member of class TXMLDisplayGroup.

99.1.66.4.17 TXMLDisplayGroup.IsFieldInUse Method

Pascal

```
function IsFieldInUse(AName: string): Boolean;
```

C++

```
__fastcall Boolean IsFieldInUse(AnsiString AName);
```

Description

This is IsFieldInUse, a member of class TXMLDisplayGroup.

99.1.66.4.18 TXMLDisplayGroup.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

99.1.66.4.19 TXMLDisplayGroup.SetDataSetField Method

Pascal

```
procedure SetDataSetField(const Value: string);
```

C++

```
__fastcall SetDataSetField(const AnsiString Value);
```

Description

This is SetDataSetField, a member of class TXMLDisplayGroup.

99.1.66.4.20 TXMLDisplayGroup.SetEnableValidateFields Method

Pascal

```
procedure SetEnableValidateFields(Value: Boolean);
```

C++

```
__fastcall SetEnableValidateFields(Boolean Value);
```

Description

This is SetEnableValidateFields, a member of class TXMLDisplayGroup.

99.1.66.4.21 TXMLDisplayGroup.SetXMLBroker Method

Pascal

```
procedure SetXMLBroker(const Value: TXMLBroker);
```

C++

```
__fastcall SetXMLBroker(const TXMLBroker Value);
```

Description

This is SetXMLBroker, a member of class TXMLDisplayGroup.

99.1.66.4.22 TXMLDisplayGroup.ValidateDataSet Method

Pascal

```
procedure ValidateDataSet;
```

C++

```
__fastcall ValidateDataSet();
```

Description

This is ValidateDataSet, a member of class TXMLDisplayGroup.

99.1.66.4.23 TXMLDisplayGroup.ValidateFields Method

Pascal

```
function ValidateFields(AddIntf: IAddScriptElements): Boolean;
```

C++

```
__fastcall Boolean ValidateFields(IAddScriptElements AddIntf);
```

Description

This is ValidateFields, a member of class TXMLDisplayGroup.

99.1.66.4.24 TXMLDisplayGroup.XMLDataChange Method

Pascal

```
procedure XMLDataChange(Sender: TObject); virtual;
```

C++

```
virtual __fastcall XMLDataChange(TObject * Sender);
```

Description

This is XMLDataChange, a member of class TXMLDisplayGroup.

99.1.66.5 TXMLDisplayGroup Properties

The properties of the TXMLDisplayGroup class are listed here.

Protected Properties

	Name	Description
	EnableValidateFields (see page 17032)	This is EnableValidateFields, a member of class TXMLDisplayGroup.
R	XMLDataSet (see page 17033)	Indicates the TXMLDataSet object that represents the dataset to which the Web item is bound.

Public Properties

	Name	Description
R	VisibleFields (see page 17032)	Indicates the controls displayed by the HTML that the XML display group generates.

Published Properties

	Name	Description
	XMLBroker (see page 17033)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
	XMLDataSetField (see page 17033)	Binds the XML display group to a nested dataset from the XML data packet.

99.1.66.5.1 TXMLDisplayGroup.EnableValidateFields Property

Pascal

```
property EnableValidateFields: Boolean;
```

C++

```
__property Boolean EnableValidateFields;
```

Description

This is EnableValidateFields, a member of class TXMLDisplayGroup.

99.1.66.5.2 TXMLDisplayGroup.VisibleFields Property

Indicates the controls displayed by the HTML that the XML display group generates.

Pascal

```
property VisibleFields: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleFields;
```

Description

Use VisibleFields to access the components that generate HTML controls for the XML display group. If a set of Web items is explicitly assigned to the WebFieldControls (see page 16884) property, VisibleFields has the same value as WebFieldControls (see page 16884). Otherwise, VisibleFields represents a set of default Web items that generate display controls for all the fields or parameters of the associated dataset or dataset field.

When no Web items are assigned to WebFieldControls (see page 16884), the default Web items in the VisibleFields list are specific to the TXMLDisplayGroup (see page 17019) descendant.

See Also

WebFieldControls (see page 16884)

xmlbroker (see page 17033)

XMLDataSetField (see page 17033)

99.1.66.5.3 TXMLDisplayGroup.XMLBroker Property

Indicates the XML broker that fetches the data packets to which the Web item is bound.

Pascal

```
property XMLBroker: TXMLBroker;
```

C++

```
__property TXMLBroker XMLBroker;
```

Description

Use XMLBroker to access the TXMLBroker object that fetches XML data packets from the application server. The XML broker can be used to obtain information about the source of the dataset, the form that contains the XML data packet, the javascript variables that represent this data packet, and so on.

Note: The dataset represented by the Web item may not correspond directly to the XML broker's dataset. It may be a nested dataset contained by that dataset. The XMLDataSet property indicates whether the dataset is nested, and if so, describes the field that represents it.

See Also

XMLDataSet (see page 17056)

DataSet (see page 17055)

99.1.66.5.4 TXMLDisplayGroup.XMLDataSet Property

Indicates the TXMLDataSet object that represents the dataset to which the Web item is bound.

Pascal

```
property XMLDataSet: TXMLDataSet;
```

C++

```
__property TXMLDataSet XMLDataSet;
```

Description

XMLDataSet describes how to locate the appropriate dataset, given the XML broker that provides its XML data packet. This dataset may be the main dataset of the XML broker, or it may be a nested dataset that is represented by a dataset field within that dataset (or a dataset field nested in a dataset field, and so on).

See Also

DataSet (see page 17055)

XMLBroker (see page 17056)

99.1.66.5.5 TXMLDisplayGroup.XMLDataSetField Property

Binds the XML display group to a nested dataset from the XML data packet.

Pascal

```
property XMLDataSetField: string;
```

C++

```
__property AnsiString XMLDataSetField;
```

Description

Most XML display groups contain controls that represent database information. TDataGrid (see page 16560) and TFieldGroup (see page 16591) represent the fields from a dataset. By default, TQueryFieldGroup (see page 16736) represents the parameter values for a query or stored procedure or field values that filter a table. If the database information comes from the dataset represented by an XML broker's data packet, XMLDataSetField is an empty string and the XMLBroker (see page 17033) property fully describes the dataset. However, if the database information comes from a nested detail set, XMLDataSetField indicates what detail dataset to use from the data packet associated with XMLBroker (see page 17033).

Specify the name of the dataset field as the value of XMLDataSetField. If the XML display group is bound to the detail set of a detail set, you can use periods to qualify dataset field names. Thus, for example, if XMLBroker (see page 17033) represents a customer dataset with a nested Orders dataset field that has its own nested LineItems field, you can bind the XML display group to the LineItems dataset by setting XMLDataSetField to

```
Orders.LineItems
```

See Also

xmlbroker (see page 17033)

VisibleFields (see page 17032)

Setting Web Item Properties

99.1.67 TXMLDisplayParent Class

Class Hierarchy

Pascal

```
TXMLDisplayParent = class(TWebDisplay);
```

C++

```
class TXMLDisplayParent : public TWebDisplay;
```

File

MidItems

Description

This is class MidItems.TXMLDisplayParent.

99.1.67.1 TXMLDisplayParent Members

The following tables list the members exposed by TXMLDisplayParent.

Protected Methods

	Name	Description
	Changed (see page 16912)	This is Changed, a member of class TWebDisplay.
V	ChangedDisplayComponent (see page 17036)	This is ChangedDisplayComponent, a member of class TXMLDisplayParent.
V	ChangedUseParent (see page 17036)	This is ChangedUseParent, a member of class TXMLDisplayParent.
V	GetDisplayComponent (see page 17037)	This is GetDisplayComponent, a member of class TXMLDisplayParent.
	SetDisplayComponent (see page 16913)	This is SetDisplayComponent, a member of class TWebDisplay.
	SetUseParent (see page 17037)	This is SetUseParent, a member of class TXMLDisplayParent.

Public Events

	Name	Description
	OnChange (see page 16914)	This is OnChange, a member of class TWebDisplay.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 17036)	This is Create, a member of class TXMLDisplayParent.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Notification (see page 16913)	This is Notification, a member of class TWebDisplay.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	DisplayComponent (see page 16914)	This is DisplayComponent, a member of class TWebDisplay.
R	Parent (see page 16914)	This is Parent, a member of class TWebDisplay.

Published Properties

	Name	Description
	UseParent (see page 17037)	This is UseParent, a member of class TXMLDisplayParent.

99.1.67.2 TXMLDisplayParent.Create Constructor

Pascal

```
constructor Create(AParent: TComponent);
```

C++

```
__fastcall TXMLDisplayParent(TComponent * AParent);
```

Description

This is Create, a member of class TXMLDisplayParent.

99.1.67.3 TXMLDisplayParent Methods

The methods of the TXMLDisplayParent class are listed here.

Protected Methods

	Name	Description
	ChangedDisplayComponent (see page 17036)	This is ChangedDisplayComponent, a member of class TXMLDisplayParent.
	ChangedUseParent (see page 17036)	This is ChangedUseParent, a member of class TXMLDisplayParent.
	GetDisplayComponent (see page 17037)	This is GetDisplayComponent, a member of class TXMLDisplayParent.
	SetUseParent (see page 17037)	This is SetUseParent, a member of class TXMLDisplayParent.

99.1.67.3.1 TXMLDisplayParent.ChangedDisplayComponent Method

Pascal

```
procedure ChangedDisplayComponent; override;
```

C++

```
virtual __fastcall ChangedDisplayComponent();
```

Description

This is ChangedDisplayComponent, a member of class TXMLDisplayParent.

99.1.67.3.2 TXMLDisplayParent.ChangedUseParent Method

Pascal

```
procedure ChangedUseParent; virtual;
```

C++

```
virtual __fastcall ChangedUseParent();
```

Description

This is ChangedUseParent, a member of class TXMLDisplayParent.

99.1.67.3.3 TXMLDisplayParent.GetDisplayComponent Method

Pascal

```
function GetDisplayComponent: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetDisplayComponent();
```

Description

This is GetDisplayComponent, a member of class TXMLDisplayParent.

99.1.67.3.4 TXMLDisplayParent.SetUseParent Method

Pascal

```
procedure SetUseParent(const Value: Boolean);
```

C++

```
__fastcall SetUseParent(const Boolean Value);
```

Description

This is SetUseParent, a member of class TXMLDisplayParent.

99.1.67.4 TXMLDisplayParent Properties

The properties of the TXMLDisplayParent class are listed here.

Published Properties

	Name	Description
	UseParent (see page 17037)	This is UseParent, a member of class TXMLDisplayParent.

99.1.67.4.1 TXMLDisplayParent.UseParent Property

Pascal

```
property UseParent: Boolean;
```

C++

```
__property Boolean UseParent;
```

Description

This is UseParent, a member of class TXMLDisplayParent.

99.1.68 TXMLDisplayReferenceButton Class

TXMLDisplayReferenceButton is the base class for Web items that generate HTML buttons for influencing controls that display data.

Class Hierarchy

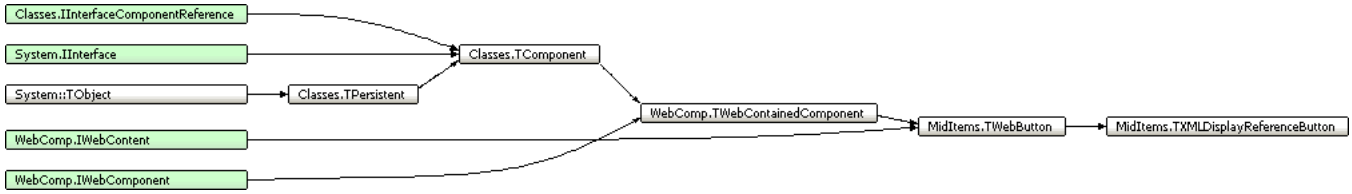

Pascal

```
TXMLDisplayReferenceButton = class(TWebButton);
```

C++

```
class TXMLDisplayReferenceButton : public TWebButton;
```

File

MidItems

Description

Use TXMLDisplayReferenceButton as a base class for components that generate the HTML statements defining a specialized button control for working with a data display control in the Web page produced by TInetXPageProducer.

TXMLDisplayReferenceButton defines properties and methods for

Generating the HTML code to define the button.

Associating the button with another Web item that generates a data display control.

Specifying tag attributes of the HTML button control, including those defined by a style sheet.

Fitting the component into a hierarchy of Web items, each of which generates part of a larger HTML document.

In order to allow other Web items to use the HTML generated by the button control, TXMLDisplayReferenceButton inherits support for the IWebContent interface from TWebButton (see page 16863).

Do not create instances of TXMLDisplayReferenceButton. TXMLDisplayReferenceButton relies on anabstract or, in C++ terminology, pure virtual method to generate the HTML statements defining a button control. Instead, create descendants of TXMLDisplayReferenceButton, such as TPostButton (see page 16704), TInsertButton (see page 16677), and so on.

See Also

TFirstButton (see page 16651)

TNextButton (see page 16694)

TPriorButton (see page 16709)

TLastButton (see page 16682)

TNextPageButton (see page 16699)

TPriorPageButton (see page 16713)

TInsertButton (see page 16677)

TDeleteButton (see page 16586)

TUndoButton (see page 16858)

TPostButton (see page 16704)

IWebContent (see page 24792)

TDataNavigator (see page 16569)

TResetQueryButton (see page 16809)

TSubmitQueryButton (see page 16833)

99.1.68.1 TXMLDisplayReferenceButton Members

The following tables list the members exposed by TXMLDisplayReferenceButton.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16868)	Returns the HTML that defines the button control.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetCaption (see page 16869)	Returns the value of the Caption (see page 16870) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDisplayComponent (see page 17043)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 17043)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXmlDisplayName (see page 17043)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
A	ImplContent (see page 16869)	Provides the underlying implementation for the Content (see page 16868) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 17044)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.

V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayComponent (see page 17044)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 17044)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.
R	XMLDisplay (see page 17045)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 17042)	Creates and initializes an instance of TXMLDisplayReferenceButton (see page 17037).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 17042)	Frees an instance of TXMLDisplayReferenceButton (see page 17037).

	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Caption (see page 16870)	Specifies the caption that appears on the button.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.

R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16871)	Represents a tag option and its value in the generated INPUT tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16871)	Adds style attributes to the generated button control.
	StyleRule (see page 16872)	Applies a style in the page producer's style sheet to the button.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	XMLComponent (see page 17045)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 17045)	Specifies whether the button is associated with its parent's display object.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.68.2 TXMLDisplayReferenceButton.Create Constructor

Creates and initializes an instance of TXMLDisplayReferenceButton (see page 17037).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TXMLDisplayReferenceButton(TComponent * AOwner);
```

Description

Call Create to instantiate a dataset button at runtime. Buttons (see page 2289) added at design time using the Web Page editor are instantiated automatically.

99.1.68.3 TXMLDisplayReferenceButton.Destroy Destructor

Frees an instance of TXMLDisplayReferenceButton (see page 17037).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLDisplayReferenceButton();
```

Description

Destroy removes the button from its parent's list of components that generate HTML.

99.1.68.4 TXMLDisplayReferenceButton Methods

The methods of the TXMLDisplayReferenceButton class are listed here.

Protected Methods

	Name	Description
	GetDisplayComponent (see page 17043)	This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	GetDisplayComponentParent (see page 17043)	This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.
	GetXmlDisplayName (see page 17043)	This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.
V	Notification (see page 17044)	Forwards notification messages to all owned components.
	SetDisplayComponent (see page 17044)	This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.
	SetDisplayComponentParent (see page 17044)	This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

99.1.68.4.1 TXMLDisplayReferenceButton.GetDisplayComponent Method

Pascal

```
function GetDisplayComponent: TComponent;
```

C++

```
__fastcall TComponent * GetDisplayComponent();
```

Description

This is GetDisplayComponent, a member of class TXMLDisplayReferenceButton.

99.1.68.4.2 TXMLDisplayReferenceButton.GetDisplayComponentParent Method

Pascal

```
function GetDisplayComponentParent: Boolean;
```

C++

```
__fastcall Boolean GetDisplayComponentParent();
```

Description

This is GetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

99.1.68.4.3 TXMLDisplayReferenceButton.GetXmlDisplayName Method

Pascal

```
function GetXmlDisplayName: string;
```

C++

```
__fastcall AnsiString GetXmlDisplayName();
```

Description

This is GetXmlDisplayName, a member of class TXMLDisplayReferenceButton.

99.1.68.4.4 TXMLDisplayReferenceButton.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

99.1.68.4.5 TXMLDisplayReferenceButton.SetDisplayComponent Method

Pascal

```
procedure SetDisplayComponent(const Value: TComponent);
```

C++

```
__fastcall SetDisplayComponent(const TComponent * Value);
```

Description

This is SetDisplayComponent, a member of class TXMLDisplayReferenceButton.

99.1.68.4.6 TXMLDisplayReferenceButton.SetDisplayComponentParent Method

Pascal

```
procedure SetDisplayComponentParent(const Value: Boolean);
```

C++

```
__fastcall SetDisplayComponentParent(const Boolean Value);
```

Description

This is SetDisplayComponentParent, a member of class TXMLDisplayReferenceButton.

99.1.68.5 TXMLDisplayReferenceButton Properties

The properties of the TXMLDisplayReferenceButton class are listed here.

Protected Properties

	Name	Description
	XMLDisplay (see page 17045)	This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

Public Properties

	Name	Description
	XMLComponent (see page 17045)	Associates the button with the component that generates the display control it influences.
	XMLUseParent (see page 17045)	Specifies whether the button is associated with its parent's display object.

99.1.68.5.1 TXMLDisplayReferenceButton.XMLComponent Property

Associates the button with the component that generates the display control it influences.

Pascal

```
property XMLComponent: TComponent;
```

C++

```
__property TComponent * XMLComponent;
```

Description

Set XMLComponent to specify the Web item that generates a display control that the button influences.

Note: Typically, descendants of TXMLDisplayReferenceButton (see page 17037) are used by a descendant of TXMLDisplayReferenceGroup (see page 17046), which has its own XMLComponent property. If the button influences this same component, you need not set the XMLComponent property. Instead, setting the XMLUseParent (see page 17045) property to true (the default) sets XMLComponent to the value of the parent's XMLComponent property.

See Also

XMLUseParent (see page 17045)

99.1.68.5.2 TXMLDisplayReferenceButton.XMLDisplay Property

Pascal

```
property XMLDisplay: TXMLDisplayParent;
```

C++

```
__property TXMLDisplayParent XMLDisplay;
```

Description

This is XMLDisplay, a member of class TXMLDisplayReferenceButton.

99.1.68.5.3 TXMLDisplayReferenceButton.XMLUseParent Property

Specifies whether the button is associated with its parent's display object.

Pascal

```
property XMLUseParent: Boolean;
```

C++

```
__property Boolean XMLUseParent;
```

Description

Use XMLUseParent to indicate how the dataset button determines the component that it influences. When XMLUseParent is false, the XMLComponent (see page 17045) property must be set to indicate the display component. When XMLUseParent is true, the button influences the component specified by its parent's XMLComponent (see page 17045) property.

The button's parent is the Web item that uses the button to assemble its own HTML content (typically a TDataNavigator (see page 16569) or TQueryButtons (see page 16730) object). If the parent does not have an associated display component, setting XMLUseParent to true sets the XMLComponent (see page 17045) property to nil (Delphi) or NULL (C++).

See Also

XMLComponent (see page 17045)

XMLComponent (see page 17051)

99.1.69 TXMLDisplayReferenceGroup Class

TXMLDisplayReferenceGroup is the base class for Web items that arrange a set of button controls that influence a control that displays data.

Class Hierarchy

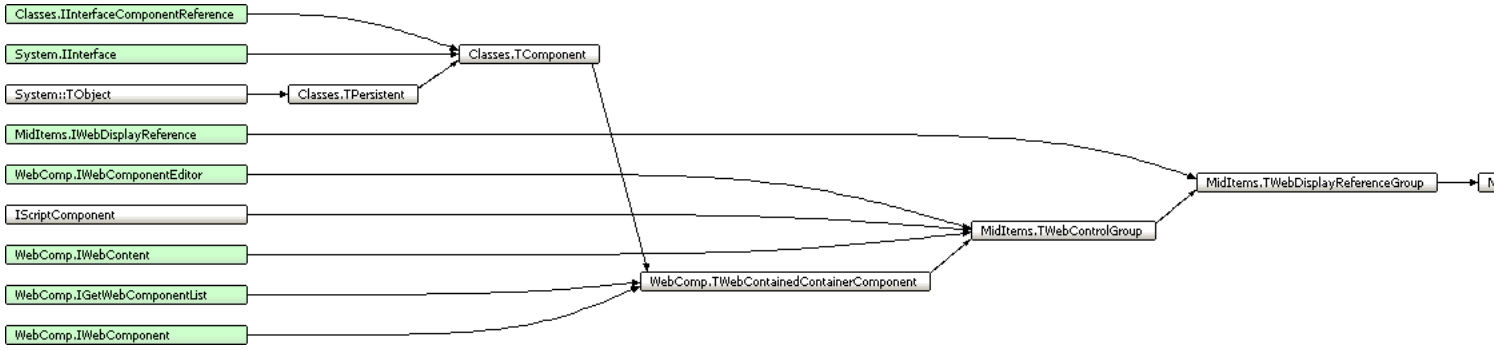

Pascal

```
TXMLDisplayReferenceGroup = class(TWebDisplayReferenceGroup);
```

C++

```
class TXMLDisplayReferenceGroup : public TWebDisplayReferenceGroup;
```

File

MidItems

Description

Use TXMLDisplayReferenceGroup as a base class for Web items that generate the HTML to arrange a collection of button controls. Each TXMLDisplayGroup (see page 17019) object has an associated Web item that generates controls that display data in the final HTML document. The buttons arranged by TXMLDisplayReferenceGroup manipulate the information the display controls show.

In order to let other Web items use the HTML that the TXMLDisplayReferenceGroup object generates,

TXMLDisplayReferenceGroup inherits the IWebContent interface from TWebControlGroup (see page 16872).

Do not create instances of TXMLDisplayReferenceGroup. TXMLDisplayReferenceGroup manages the Web items that generate buttons and the association to a display generator, but does not produce any content. Instead, use a descendant of TXMLDisplayReferenceGroup that generates the HTML for its set of specialized buttons such as TDataNavigator (see page 16569) or TQueryButtons (see page 16730).

See Also

TDataNavigator (see page 16569)

IWebContent (see page 24792)

99.1.69.1 TXMLDisplayReferenceGroup Members

The following tables list the members exposed by TXMLDisplayReferenceGroup.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.

	GetWebDisplayComponent (see page 16920)	This is GetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
V	ImplAddElements (see page 17051)	Provides the underlying implementation for the AddElements method.
V	ImplCanAddClass (see page 16881)	Provides the underlying implementation for the CanAddClass (see page 16878) method.
V	ImplContent (see page 16882)	Provides the underlying implementation for the Content (see page 16879) method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16920)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
V	SetWebDisplayComponent (see page 16921)	This is SetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.

R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebDisplay (see page 16921)	This is WebDisplay, a member of class TWebDisplayReferenceGroup.
	WebDisplayComponent (see page 16921)	
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16877)	Creates and initializes an instance of TWebControlGroup (see page 16872).
V	Create (see page 16919)	Creates an instance of TWebDisplayReferenceGroup (see page 16915).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24742)	Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16877)	Frees an instance of TWebControlGroup (see page 16872).
V	Destroy (see page 16919)	Frees an instance of TWebDisplayReferenceGroup (see page 16915).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.

	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	GetWebDisplayComponent (see page 17081)	This is GetWebDisplayComponent, a member of class IWebDisplayReference.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebDisplayComponent (see page 17081)	
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

	XMLComponent (see page 17051)	Associates the group with a component that generates HTML display controls.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

99.1.69.2 TXMLDisplayReferenceGroup Methods

The methods of the TXMLDisplayReferenceGroup class are listed here.

Protected Methods

	Name	Description
	ImplAddElements (see page 17051)	Provides the underlying implementation for the AddElements method.

99.1.69.2.1 TXMLDisplayReferenceGroup.ImplAddElements Method

Provides the underlying implementation for the AddElements method.

Pascal

```
procedure ImplAddElements(AddIntf: IAddScriptElements); override;
```

C++

```
virtual __fastcall ImplAddElements(IAddScriptElements AddIntf);
```

Description

The AddElements method calls ImplAddElements to add any script declarations by calling the AddIntf interface. These declarations allow the generated display control to call script methods in response to user events by ensuring that the required methods are declared.

As implemented in TWebControlGroup, ImplAddElements does nothing. TWebControlGroup descendants override ImplAddElements to add any required declarations.

See Also

AddElements (see page 16878)

99.1.69.3 TXMLDisplayReferenceGroup Properties

The properties of the TXMLDisplayReferenceGroup class are listed here.

Public Properties

	Name	Description
	XMLComponent (see page 17051)	Associates the group with a component that generates HTML display controls.

99.1.69.3.1 TXMLDisplayReferenceGroup.XMLComponent Property

Associates the group with a component that generates HTML display controls.

Pascal

```
property XMLComponent: TComponent;
```

C++

```
__property TComponent * XMLComponent;
```

Description

Use XMLComponent to associate the group of buttons with a Web item that generates the display controls those buttons influence. Typical values of XMLComponent are TFieldGroup (see page 16591) or TDataGrid (see page 16560).

If all the buttons arranged by the TXMLDisplayReferenceGroup (see page 17046) object operate on the same set of display controls, it is most efficient to specify the Web item generating these controls as the value of XMLComponent. Then, each button uses this value to determine its associated display controls when its XMLUseParent property is true. Any buttons that do not operate on the display controls generated by XMLComponent can set their own XMLComponent property to another Web item.

See Also

XMLComponent (see page 17045)

XMLUseParent (see page 17045)

TFieldGroup (see page 16591)

TDataGrid (see page 16560)

TQueryFieldGroup (see page 16736)

99.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IDataSetComponent (see page 17053)	IDataSetComponent defines the methods that bind a Web item to an XML broker.
	IDataSetField (see page 17057)	This is class MidItems.IDataSetField.
	IDataSetFields (see page 17059)	This is class MidItems.IDataSetFields.
	IFormatColumn (see page 17061)	This is class MidItems.IFormatColumn.
	IHTMLField (see page 17062)	IHTMLField is an interface that must be implemented by Web items that generate <INPUT> elements.
	IHTMLForm (see page 17064)	IHTMLForm is an interface that must be implemented by Web items that generate forms.
	IInetXWebDataSet (see page 17066)	IInetXWebDataSet lets a Web item interact with information about a dataset in an XML broker's data packet.
	IQueryField (see page 17070)	This is class MidItems.IQueryField.
	IQueryFields (see page 17073)	This is class MidItems.IQueryFields.
	IQueryForm (see page 17074)	This is class MidItems.IQueryForm.
	IRestoreDefaults (see page 17074)	This is class MidItems.IRestoreDefaults.
	IStatusField (see page 17075)	This is class MidItems.IStatusField.
	IValidateField (see page 17076)	This is class MidItems.IValidateField.
	IValidateFields (see page 17077)	This is class MidItems.IValidateFields.
	IWebDisplay (see page 17079)	
	IWebDisplayReference (see page 17080)	
	IXMLDisplay (see page 17082)	This is class MidItems.IXMLDisplay.

99.2.1 IDataSetComponent Interface

IDataSetComponent defines the methods that bind a Web item to an XML broker.

Class Hierarchy

MidItems.IDataSetComponent

Pascal

```
IDataSetComponent = interface;
```

C++

```
__interface IDataSetComponent;
```

File

MidItems

Description

The TXMLDisplayGroup (see page 17019) class implements IDataSetComponent to provide information about the dataset from which its child controls obtain their data. The child Web items use this information to determine the default data binding.

Most Web items that generate a group of data-aware controls inherit the support for IDataSetComponent from TXMLDisplayGroup (see page 17019).

See Also

TXMLDisplayGroup (see page 17019)

TXMLBroker (see page 26049)

99.2.1.1 IDataSetComponent Members

The following tables list the members exposed by IDataSetComponent.

Public Methods

	Name	Description
	GetDataSet (see page 17054)	Returns the value of the DataSet (see page 17055) property.
	GetXMLBroker (see page 17054)	Returns the value of the XMLBroker (see page 17056) property.
	GetXMLDataSet (see page 17055)	Returns the value of the XMLDataSet (see page 17056) property.

Public Properties

	Name	Description
R	DataSet (see page 17055)	Provides access to an interface for working with the associated dataset.
R	XMLBroker (see page 17056)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
R	XMLDataSet (see page 17056)	Indicates the TXMLDataSet (see page 17006) object that represents the dataset to which the Web item is bound.

99.2.1.2 IDataSetComponent Methods

The methods of the IDataSetComponent class are listed here.

Public Methods

	Name	Description
	GetDataSet (see page 17054)	Returns the value of the DataSet (see page 17055) property.
	GetXMLBroker (see page 17054)	Returns the value of the XMLBroker (see page 17056) property.
	GetXMLDataSet (see page 17055)	Returns the value of the XMLDataSet (see page 17056) property.

99.2.1.2.1 IDataSetComponent.GetDataSet Method

Returns the value of the DataSet (see page 17055) property.

Pascal

```
function GetDataSet: IInetXWebDataSet;
```

C++

```
__fastcall IInetXWebDataSet GetDataSet();
```

Description

GetDataSet is the read access method for the DataSet (see page 17055) property.

See Also

DataSet (see page 17055)

99.2.1.2.2 IDataSetComponent.GetXMLBroker Method

Returns the value of the XMLBroker (see page 17056) property.

Pascal

```
function GetXMLBroker: TXMLBroker;
```

C++

```
__fastcall TXMLBroker GetXMLBroker();
```

Description

GetXMLBroker is the read access method for the XMLBroker (see page 17056) property.

See Also

XMLBroker (see page 17056)

99.2.1.2.3 IDataSetComponent.GetXMLDataSet Method

Returns the value of the XMLDataSet (see page 17056) property.

Pascal

```
function GetXMLDataSet: TXMLDataSet;
```

C++

```
__fastcall TXMLDataSet GetXMLDataSet();
```

Description

GetXMLDataSet is the read access method for the XMLDataSet (see page 17056) property.

See Also

XMLDataSet (see page 17056)

99.2.1.3 IDataSetComponent Properties

The properties of the IDataSetComponent class are listed here.

Public Properties

	Name	Description
R	DataSet (see page 17055)	Provides access to an interface for working with the associated dataset.
R	XMLBroker (see page 17056)	Indicates the XML broker that fetches the data packets to which the Web item is bound.
R	XMLDataSet (see page 17056)	Indicates the TXMLDataSet (see page 17006) object that represents the dataset to which the Web item is bound.

99.2.1.3.1 IDataSetComponent.DataSet Property

Provides access to an interface for working with the associated dataset.

Pascal

```
property DataSet: IInetXWebDataSet;
```

C++

```
__property IInetXWebDataSet DataSet;
```

Description

The child Web items of the item that implements IDataSetComponent (see page 17053) use the DataSet property to obtain information about the dataset to which they are bound and to respond when this information changes. This information includes

details about the available fields and parameters in the dataset to which they can be bound.

See Also

XMLDataSet (see page 17056)

99.2.1.3.2 IDataSetComponent.XMLBroker Property

Indicates the XML broker that fetches the data packets to which the Web item is bound.

Pascal

```
property XMLBroker: TXMLBroker;
```

C++

```
__property TXMLBroker XMLBroker;
```

Description

Use XMLBroker to access the TXMLBroker object that fetches XML data packets from the application server. The XML broker can be used to obtain information about the source of the dataset, the form that contains the XML data packet, the javascript variables that represent this data packet, and so on.

Note: The dataset represented by the Web item may not correspond directly to the XML broker's dataset. It may be a nested dataset contained by that dataset. The XMLDataSet (see page 17056) property indicates whether the dataset is nested, and if so, describes the field that represents it.

See Also

XMLDataSet (see page 17056)

DataSet (see page 17055)

99.2.1.3.3 IDataSetComponent.XMLDataSet Property

Indicates the TXMLDataSet (see page 17006) object that represents the dataset to which the Web item is bound.

Pascal

```
property XMLDataSet: TXMLDataSet;
```

C++

```
__property TXMLDataSet XMLDataSet;
```

Description

XMLDataSet describes how to locate the appropriate dataset, given the XML broker that provides its XML data packet. This dataset may be the main dataset of the XML broker, or it may be a nested dataset that is represented by a dataset field within that dataset (or a dataset field nested in a dataset field, and so on).

See Also

DataSet (see page 17055)

XMLBroker (see page 17056)

99.2.2 IDataSetField Interface

Class Hierarchy

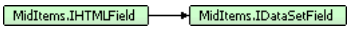

Pascal

```
IDataSetField = interface(IHTMLField);
```

C++

```
__interface IDataSetField : public IHTMLField;
```

File

MidItems

Description

This is class MidItems.IDataSetField.

99.2.2.1 IDataSetField Members

The following tables list the members exposed by IDataSetField.

Public Methods

	Name	Description
	GetFieldName (see page 17057)	This is GetFieldName, a member of class IDataSetField.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetRowSetFieldAttributes (see page 17058)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	SetFieldName (see page 17058)	This is SetFieldName, a member of class IDataSetField.

Public Properties

	Name	Description
	FieldName (see page 17058)	This is FieldName, a member of class IDataSetField.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.

99.2.2.2 IDataSetField Methods

The methods of the IDataSetField class are listed here.

Public Methods

	Name	Description
	GetFieldName (see page 17057)	This is GetFieldName, a member of class IDataSetField.
	GetRowSetFieldAttributes (see page 17058)	This is GetRowSetFieldAttributes, a member of class IDataSetField.
	SetFieldName (see page 17058)	This is SetFieldName, a member of class IDataSetField.

99.2.2.2.1 IDataSetField.GetFieldName Method

Pascal

```
function GetFieldName: string;
```

C++

```
__fastcall AnsiString GetFieldName();
```

Description

This is GetFieldName, a member of class IDataSetField.

99.2.2.2.2 IDataSetField.GetRowSetFieldAttributes Method

Pascal

```
function GetRowSetFieldAttributes(const FieldVarName: string): string;
```

C++

```
__fastcall AnsiString GetRowSetFieldAttributes(const AnsiString FieldVarName);
```

Description

This is GetRowSetFieldAttributes, a member of class IDataSetField.

99.2.2.2.3 IDataSetField.SetFieldName Method

Pascal

```
procedure SetFieldName(AFieldName: string);
```

C++

```
__fastcall SetFieldName(AnsiString AFieldName);
```

Description

This is SetFieldName, a member of class IDataSetField.

99.2.2.3 IDataSetField Properties

The properties of the IDataSetField class are listed here.

Public Properties

	Name	Description
	FieldName (see page 17058)	This is FieldName, a member of class IDataSetField.

99.2.2.3.1 IDataSetField.FieldName Property

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;
```

Description

This is FieldName, a member of class IDataSetField.

99.2.3 IDataSetFields Interface

Class Hierarchy

MidItems.IDataSetFields

Pascal

```
IDataSetFields = interface;
```

C++

```
__interface IDataSetFields;
```

File

MidItems

Description

This is class MidItems.IDataSetFields.

99.2.3.1 IDataSetFields Members

The following tables list the members exposed by IDataSetFields.

Public Methods

	Name	Description
	GetFieldsList (see page 17060)	This is GetFieldsList, a member of class IDataSetFields.
	GetVisibleFields (see page 17060)	This is GetVisibleFields, a member of class IDataSetFields.
	HasStatusField (see page 17060)	This is HasStatusField, a member of class IDataSetFields.
	IsFieldInUse (see page 17060)	This is IsFieldInUse, a member of class IDataSetFields.
	ValidateDataSet (see page 17060)	This is ValidateDataSet, a member of class IDataSetFields.

Public Properties

	Name	Description
R	VisibleFields (see page 17061)	This is VisibleFields, a member of class IDataSetFields.

99.2.3.2 IDataSetFields Methods

The methods of the IDataSetFields class are listed here.

Public Methods

	Name	Description
	GetFieldsList (see page 17060)	This is GetFieldsList, a member of class IDataSetFields.
	GetVisibleFields (see page 17060)	This is GetVisibleFields, a member of class IDataSetFields.
	HasStatusField (see page 17060)	This is HasStatusField, a member of class IDataSetFields.
	IsFieldInUse (see page 17060)	This is IsFieldInUse, a member of class IDataSetFields.
	ValidateDataSet (see page 17060)	This is ValidateDataSet, a member of class IDataSetFields.

99.2.3.2.1 IDataSetFields.GetFieldsList Method

Pascal

```
procedure GetFieldsList(AList: TStrings);
```

C++

```
__fastcall GetFieldsList(TStrings AList);
```

Description

This is GetFieldsList, a member of class IDataSetFields.

99.2.3.2.2 IDataSetFields.GetVisibleFields Method

Pascal

```
function GetVisibleFields: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleFields();
```

Description

This is GetVisibleFields, a member of class IDataSetFields.

99.2.3.2.3 IDataSetFields.HasStatusField Method

Pascal

```
function HasStatusField: Boolean;
```

C++

```
__fastcall Boolean HasStatusField();
```

Description

This is HasStatusField, a member of class IDataSetFields.

99.2.3.2.4 IDataSetFields.IsFieldInUse Method

Pascal

```
function IsFieldInUse(AName: string): Boolean;
```

C++

```
__fastcall Boolean IsFieldInUse(AnsiString AName);
```

Description

This is IsFieldInUse, a member of class IDataSetFields.

99.2.3.2.5 IDataSetFields.ValidateDataSet Method

Pascal

```
procedure ValidateDataSet;
```

C++

```
__fastcall ValidateDataSet();
```

Description

This is ValidateDataSet, a member of class IDataSetFields.

99.2.3.3 IDataSetFields Properties

The properties of the IDataSetFields class are listed here.

Public Properties

	Name	Description
R	VisibleFields (see page 17061)	This is VisibleFields, a member of class IDataSetFields.

99.2.3.3.1 IDataSetFields.VisibleFields Property

Pascal

```
property VisibleFields: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleFields;
```

Description

This is VisibleFields, a member of class IDataSetFields.

99.2.4 IFormatColumn Interface

Class Hierarchy

MidItems.IFormatColumn

Pascal

```
IFormatColumn = interface;
```

C++

```
__interface IFormatColumn;
```

File

MidItems

Description

This is class MidItems.IFormatColumn.

99.2.4.1 IFormatColumn Members

The following tables list the members exposed by IFormatColumn.

Public Methods

	Name	Description
	FormatColumnData (see page 17062)	This is FormatColumnData, a member of class IFormatColumn.
	FormatColumnHeading (see page 17062)	This is FormatColumnHeading, a member of class IFormatColumn.

99.2.4.2 IFormatColumn Methods

The methods of the IFormatColumn class are listed here.

Public Methods

	Name	Description
	FormatColumnData (see page 17062)	This is FormatColumnData, a member of class IFormatColumn.
	FormatColumnHeading (see page 17062)	This is FormatColumnHeading, a member of class IFormatColumn.

99.2.4.2.1 IFormatColumn.FormatColumnData Method

Pascal

```
function FormatColumnData(const Content: string; Options: TWebContentOptions): string;
```

C++

```
__fastcall AnsiString FormatColumnData(const AnsiString Content, TWebContentOptions Options);
```

Description

This is FormatColumnData, a member of class IFormatColumn.

99.2.4.2.2 IFormatColumn.FormatColumnHeading Method

Pascal

```
function FormatColumnHeading(Options: TWebContentOptions): string;
```

C++

```
__fastcall AnsiString FormatColumnHeading(TWebContentOptions Options);
```

Description

This is FormatColumnHeading, a member of class IFormatColumn.

99.2.5 IHTMLField Interface

IHTMLField is an interface that must be implemented by Web items that generate <INPUT> elements.

Class Hierarchy

MidItems.IHTMLField

Pascal

```
IHTMLField = interface;
```

C++

```
__interface IHTMLField;
```

File

MidItems

Description

The TWebDataDisplay (see page 16884) class implements IHTMLField to provide information about information about the

HTML control it represents. This interface is used when generating javascript that refers to the control.

Most Web items that generate HTML <INPUT> elements inherit the support for IHTMLField from TWebDataDisplay (see page 16884). However, when writing custom Web items that generate <INPUT> tags but that do not descend from TWebDataDisplay (see page 16884), component writers must ensure that the Web item supports the IHTMLField interface.

See Also

TWebDataDisplay (see page 16884)

99.2.5.1 IHTMLField Members

The following tables list the members exposed by IHTMLField.

Public Methods

	Name	Description
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.

Public Properties

	Name	Description
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.

99.2.5.2 IHTMLField Methods

The methods of the IHTMLField class are listed here.

Public Methods

	Name	Description
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.

99.2.5.2.1 IHTMLField.GetHTMLControlName Method

Returns the name of the generated HTML <INPUT> control.

Pascal

```
function GetHTMLControlName: string;
```

C++

```
__fastcall AnsiString GetHTMLControlName();
```

Description

GetHTMLFormName is the read access method for the HTMLControlName (see page 17064) property.

See Also

HTMLControlName (see page 17064)

99.2.5.3 IHTMLField Properties

The properties of the IHTMLField class are listed here.

Public Properties

	Name	Description
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.

99.2.5.3.1 IHTMLField.HTMLControlName Property

Indicates the name of the generated HTML control.

Pascal

```
property HTMLControlName: string;
```

C++

```
__property AnsiString HTMLControlName;
```

Description

HTMLControlName is the value of the NAME option on the generated <INPUT> tag. This value is used when creating javascript references to the control.

99.2.6 IHTMLForm Interface

IHTMLForm is an interface that must be implemented by Web items that generate forms.

Class Hierarchy

MidItems.IHTMLForm

Pascal

```
IHTMLForm = interface;
```

C++

```
__interface IHTMLForm;
```

File

MidItems

Description

Web items that produce forms (such as TQueryForm (see page 16745) or TDataForm (see page 16554)) must implement the IHTML form interface. This interface allows other Web items to access information about form elements when generating javascript declarations and method calls.

See Also

TQueryForm (see page 16745)

TDataForm (see page 16554)

99.2.6.1 IHTMLForm Members

The following tables list the members exposed by IHTMLForm.

Public Methods

	Name	Description
	GetHTMLFormName (see page 17065)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 17065)	Returns the name of the javascript variable that refers to the generated form.

Public Properties

	Name	Description
R	HTMLFormName (see page 17066)	Indicates the name of the generated HTML form.
R	HTMLFormVarName (see page 17066)	Indicates the name of the javascript variable that represents the HTML form.

99.2.6.2 IHTMLForm Methods

The methods of the IHTMLForm class are listed here.

Public Methods

	Name	Description
	GetHTMLFormName (see page 17065)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 17065)	Returns the name of the javascript variable that refers to the generated form.

99.2.6.2.1 IHTMLForm.GetHTMLFormName Method

Returns the name of the generated HTML form.

Pascal

```
function GetHTMLFormName: string;
```

C++

```
__fastcall AnsiString GetHTMLFormName();
```

Description

GetHTMLFormName is the read access method for the HTMLFormName (see page 17066) property.

See Also

HTMLFormName (see page 17066)

99.2.6.2.2 IHTMLForm.GetHTMLFormVarName Method

Returns the name of the javascript variable that refers to the generated form.

Pascal

```
function GetHTMLFormVarName: string;
```

C++

```
__fastcall AnsiString GetHTMLFormVarName();
```

Description

GetHTMLFormVarName is the read access method for the HTMLFormVarName (see page 17066) property.

See Also

HTMLFormVarName (see page 17066)

99.2.6.3 IHTMLForm Properties

The properties of the IHTMLForm class are listed here.

Public Properties

	Name	Description
R	HTMLFormName (see page 17066)	Indicates the name of the generated HTML form.
R	HTMLFormVarName (see page 17066)	Indicates the name of the javascript variable that represents the HTML form.

99.2.6.3.1 IHTMLForm.HTMLFormName Property

Indicates the name of the generated HTML form.

Pascal

```
property HTMLFormName: string;
```

C++

```
__property AnsiString HTMLFormName;
```

Description

HTMLFormName is the value of the NAME option on the generated form. Other Web items can use this property when they need to refer to the form name. For example, this property is used to declare the javascript variable that represents the form.

See Also

HTMLFormVarName (see page 17066)

99.2.6.3.2 IHTMLForm.HTMLFormVarName Property

Indicates the name of the javascript variable that represents the HTML form.

Pascal

```
property HTMLFormVarName: string;
```

C++

```
__property AnsiString HTMLFormVarName;
```

Description

HTMLFormVarName is the value of the javascript variable for the form. Other Web items can use this property when they need to refer to the form in generated javascript code.

See Also

HTMLFormName (see page 17066)

99.2.7 IInetXWebDataSet Interface

IInetXWebDataSet lets a Web item interact with information about a dataset in an XML broker's data packet.

Class Hierarchy

MidItems.IInetXWebDataSet

Pascal

```
IInetXWebDataSet = interface;
```

C++

```
__interface IInetXWebDataSet;
```

File

MidItems

Description

Web items use the IInetXWebDataSet interface (formerly known as IMidasWebDataSet (see page 17093)) to obtain information about a dataset supplied by an XML broker. By using this interface, the Web item does not need to worry about whether the dataset is the top-level dataset in the XML data packet or a nested dataset in that data packet.

IInetXWebDataSet defines methods for obtaining information about the fields and parameters of the associated dataset. It also defines methods for assigning an event handler that receives notifications when this information changes.

Note: VCL Method declarations that use IInetXWebDataSet use the _di_IInetXWebDataSet type instead. This type is a DelphiInterface wrapper around the IInetXWebDataSet interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IInetXWebDataSet >
_di_IInetXWebDataSet;
```

See Also

TXMLDataSet (see page 17006)

TXMLBroker (see page 26049)

99.2.7.1 IInetXWebDataSet Members

The following tables list the members exposed by IInetXWebDataSet.

Public Methods

	Name	Description
	AddOnDataChange (see page 17068)	Adds an event handler to the list of event handlers that are called when the fields or parameters of the dataset change.
	FetchParams (see page 17068)	Tells the XML broker that manages the associated dataset to fetch parameter values from the application server.
	FieldCount (see page 17068)	Returns the number of fields in the associated dataset.
	Fields (see page 17069)	Returns the set of fields in the associated dataset.
	ParamCount (see page 17069)	Returns the number of parameters for the associated dataset.
	Params (see page 17069)	Returns the set of parameters in the associated dataset.
	RemoveOnDataChange (see page 17070)	Removes an event handler from the list of event handlers that are called when the fields or parameters of the dataset change.

99.2.7.2 IInetXWebDataSet Methods

The methods of the IInetXWebDataSet class are listed here.

Public Methods

	Name	Description
	AddOnDataChange (see page 17068)	Adds an event handler to the list of event handlers that are called when the fields or parameters of the dataset change.

	FetchParams (see page 17068)	Tells the XML broker that manages the associated dataset to fetch parameter values from the application server.
	FieldCount (see page 17068)	Returns the number of fields in the associated dataset.
	Fields (see page 17069)	Returns the set of fields in the associated dataset.
	ParamCount (see page 17069)	Returns the number of parameters for the associated dataset.
	Params (see page 17069)	Returns the set of parameters in the associated dataset.
	RemoveOnDataChange (see page 17070)	Removes an event handler from the list of event handlers that are called when the fields or parameters of the dataset change.

99.2.7.2.1 IInetXWebDataSet.AddOnDataChange Method

Adds an event handler to the list of event handlers that are called when the fields or parameters of the dataset change.

Pascal

```
procedure AddOnDataChange(const Value: TNotifyEvent);
```

C++

```
__fastcall AddOnDataChange(const TNotifyEvent Value);
```

Description

Call AddOnDataChange to specify an event handler that is called when the fields or parameters of the dataset change. This allows Web items to update their data bindings to reflect changes in the associated data packet.

Value specifies the event handler that is added to the list.

See Also

RemoveOnDataChange (see page 17070)

99.2.7.2.2 IInetXWebDataSet.FetchParams Method

Tells the XML broker that manages the associated dataset to fetch parameter values from the application server.

Pascal

```
procedure FetchParams;
```

C++

```
__fastcall FetchParams();
```

Description

Call FetchParams to tell the XML broker to fetch parameter values from the application server. This ensures that the Params (see page 17069) method returns the current values of any parameters on the server.

See Also

FetchParams (see page 26058)

Params (see page 17069)

99.2.7.2.3 IInetXWebDataSet.FieldCount Method

Returns the number of fields in the associated dataset.

Pascal

```
function FieldCount: Integer;
```

C++

```
__fastcall int FieldCount();
```

Description

Call FieldCount to determine the number of fields returned by the Fields (see page 17069) method. This is the number of fields in the XML data packet belonging to the associated dataset.

See Also

Fields (see page 17069)

99.2.7.2.4 IInetXWebDataSet.Fields Method

Returns the set of fields in the associated dataset.

Pascal

```
function Fields: TFields;
```

C++

```
__fastcall TFields Fields();
```

Description

Call Fields to obtain access to the field objects (TField) that represent the fields in the XML data packet. Each field object can be used to read the value of the field, determine its type, and so on.

See Also

FieldCount (see page 17068)

99.2.7.2.5 IInetXWebDataSet.ParamCount Method

Returns the number of parameters for the associated dataset.

Pascal

```
function ParamCount: Integer;
```

C++

```
__fastcall int ParamCount();
```

Description

Call ParamCount to determine the number of parameters returned by the Params (see page 17069) method. This is the number of parameters in the XML data packet belonging to the associated dataset.

See Also

Params (see page 17069)

99.2.7.2.6 IInetXWebDataSet.Params Method

Returns the set of parameters in the associated dataset.

Pascal

```
function Params: TParams;
```

C++

```
__fastcall TParams Params();
```

Description

Call Params to obtain access to the parameter objects (TParam) that represent the parameters in the XML data packet. Each parameter object can be used to read the value of the parameter, determine its type, and so on.

See Also

ParamCount (see page 17069)

FetchParams (see page 17068)

99.2.7.2.7 IInetXWebDataSet.RemoveOnDataChange Method

Removes an event handler from the list of event handlers that are called when the fields or parameters of the dataset change.

Pascal

```
procedure RemoveOnDataChange(const Value: TNotifyEvent);
```

C++

```
__fastcall RemoveOnDataChange(const TNotifyEvent Value);
```

Description

Call RemoveOnDataChange to remove an event handler that was added using the AddOnDataChange (see page 17068) method.

Value specifies the event handler to remove from the list.

See Also

AddOnDataChange (see page 17068)

99.2.8 IQueryField Interface

Class Hierarchy

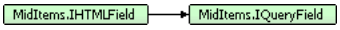

Pascal

```
IQueryField = interface(IHTMLField);
```

C++

```
__interface IQueryField : public IHTMLField;
```

File

MidItems

Description

This is class MidItems.IQueryField.

99.2.8.1 IQueryField Members

The following tables list the members exposed by IQueryField.

Public Methods

	Name	Description
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetParamName (see page 17071)	This is GetParamName, a member of class IQueryField.
	GetText (see page 17071)	This is GetText, a member of class IQueryField.
	SetParamName (see page 17071)	This is SetParamName, a member of class IQueryField.
	SetText (see page 17072)	This is SetText, a member of class IQueryField.

Public Properties

	Name	Description
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	ParamName (see page 17072)	This is ParamName, a member of class IQueryField.
	Text (see page 17072)	This is Text, a member of class IQueryField.

99.2.8.2 IQueryField Methods

The methods of the IQueryField class are listed here.

Public Methods

	Name	Description
	GetParamName (see page 17071)	This is GetParamName, a member of class IQueryField.
	GetText (see page 17071)	This is GetText, a member of class IQueryField.
	SetParamName (see page 17071)	This is SetParamName, a member of class IQueryField.
	SetText (see page 17072)	This is SetText, a member of class IQueryField.

99.2.8.2.1 IQueryField.GetParamName Method

Pascal

```
function GetParamName: string;
```

C++

```
__fastcall AnsiString GetParamName();
```

Description

This is GetParamName, a member of class IQueryField.

99.2.8.2.2 IQueryField.GetText Method

Pascal

```
function GetText: string;
```

C++

```
__fastcall AnsiString GetText();
```

Description

This is GetText, a member of class IQueryField.

99.2.8.2.3 IQueryField.SetParamName Method

Pascal

```
procedure SetParamName(AParamName: string);
```

C++

```
__fastcall SetParamName(AnsiString AParamName);
```

Description

This is SetParamName, a member of class IQueryField.

99.2.8.2.4 IQueryField.SetText Method

Pascal

```
procedure SetText(const Value: string);
```

C++

```
__fastcall SetText(const AnsiString Value);
```

Description

This is SetText, a member of class IQueryField.

99.2.8.3 IQueryField Properties

The properties of the IQueryField class are listed here.

Public Properties

	Name	Description
	ParamName (see page 17072)	This is ParamName, a member of class IQueryField.
	Text (see page 17072)	This is Text, a member of class IQueryField.

99.2.8.3.1 IQueryField.ParamName Property

Pascal

```
property ParamName: string;
```

C++

```
__property AnsiString ParamName;
```

Description

This is ParamName, a member of class IQueryField.

99.2.8.3.2 IQueryField.Text Property

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

This is Text, a member of class IQueryField.

99.2.9 IQueryFields Interface

Class Hierarchy

MidItems.IQueryFields

Pascal

```
IQueryFields = interface;
```

C++

```
__interface IQueryFields;
```

File

MidItems

Description

This is class MidItems.IQueryFields.

99.2.9.1 IQueryFields Members

The following tables list the members exposed by IQueryFields.

Public Methods

	Name	Description
	GetParamsList (see page 17073)	This is GetParamsList, a member of class IQueryFields.
	IsParamInUse (see page 17074)	This is IsParamInUse, a member of class IQueryFields.
	ValidateDataSet (see page 17074)	This is ValidateDataSet, a member of class IQueryFields.

99.2.9.2 IQueryFields Methods

The methods of the IQueryFields class are listed here.

Public Methods

	Name	Description
	GetParamsList (see page 17073)	This is GetParamsList, a member of class IQueryFields.
	IsParamInUse (see page 17074)	This is IsParamInUse, a member of class IQueryFields.
	ValidateDataSet (see page 17074)	This is ValidateDataSet, a member of class IQueryFields.

99.2.9.2.1 IQueryFields.GetParamsList Method

Pascal

```
procedure GetParamsList(AList: TStrings);
```

C++

```
__fastcall GetParamsList(TStrings AList);
```

Description

This is GetParamsList, a member of class IQueryFields.

99.2.9.2.2 IQueryFields.IsParamInUse Method

Pascal

```
function IsParamInUse(AName: string): Boolean;
```

C++

```
__fastcall Boolean IsParamInUse(AnsiString AName);
```

Description

This is IsParamInUse, a member of class IQueryFields.

99.2.9.2.3 IQueryFields.ValidateDataSet Method

Pascal

```
procedure ValidateDataSet;
```

C++

```
__fastcall ValidateDataSet();
```

Description

This is ValidateDataSet, a member of class IQueryFields.

99.2.10 IQueryForm Interface

Class Hierarchy

MidItems.IQueryForm

Pascal

```
IQueryForm = interface;
```

C++

```
__interface IQueryForm;
```

File

MidItems

Description

This is class MidItems.IQueryForm.

99.2.11 IRestoreDefaults Interface

Class Hierarchy

MidItems.IRestoreDefaults

Pascal

```
IRestoreDefaults = interface;
```

C++

```
__interface IRestoreDefaults;
```

File

MidItems

Description

This is class MidItems.IRestoreDefaults.

99.2.11.1 IRestoreDefaults Members

The following tables list the members exposed by IRestoreDefaults.

Public Methods

	Name	Description
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.

99.2.11.2 IRestoreDefaults Methods

The methods of the IRestoreDefaults class are listed here.

Public Methods

	Name	Description
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.

99.2.11.2.1 IRestoreDefaults.RestoreDefaults Method

Pascal

```
procedure RestoreDefaults;
```

C++

```
__fastcall RestoreDefaults();
```

Description

This is RestoreDefaults, a member of class IRestoreDefaults.

99.2.12 IStatusField Interface

Class Hierarchy

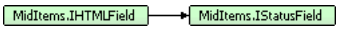

Pascal

```
IStatusField = interface(IHTMLField);
```

C++

```
__interface IStatusField : public IHTMLField;
```

File

MidItems

Description

This is class MidItems.IStatusField.

99.2.13 IValidateField Interface

Class Hierarchy

MidItems.IValidateField

Pascal

```
IValidateField = interface;
```

C++

```
__interface IValidateField;
```

File

MidItems

Description

This is class MidItems.IValidateField.

99.2.13.1 IValidateField Members

The following tables list the members exposed by IValidateField.

Public Methods

	Name	Description
	ValidateField (see page 17076)	This is ValidateField, a member of class IValidateField.

99.2.13.2 IValidateField Methods

The methods of the IValidateField class are listed here.

Public Methods

	Name	Description
	ValidateField (see page 17076)	This is ValidateField, a member of class IValidateField.

99.2.13.2.1 IValidateField.ValidateField Method

Pascal

```
function ValidateField(DataSet: IInetXWebDataSet; AddIntf: IAddScriptElements): Boolean;
```

C++

```
__fastcall Boolean ValidateField(IInetXWebDataSet DataSet, IAddScriptElements AddIntf);
```

Description

This is ValidateField, a member of class IValidateField.

99.2.14 IValidateFields Interface

Class Hierarchy

MidItems.IValidateFields

Pascal

```
IValidateFields = interface;
```

C++

```
__interface IValidateFields;
```

File

MidItems

Description

This is class MidItems.IValidateFields.

99.2.14.1 IValidateFields Members

The following tables list the members exposed by IValidateFields.

Public Methods

	Name	Description
	GetEnableValidateFields (see page 17078)	This is GetEnableValidateFields, a member of class IValidateFields.
	SetEnableValidateFields (see page 17078)	This is SetEnableValidateFields, a member of class IValidateFields.
	ValidateFields (see page 17078)	This is ValidateFields, a member of class IValidateFields.

Public Properties

	Name	Description
	EnableValidateFields (see page 17078)	This is EnableValidateFields, a member of class IValidateFields.

99.2.14.2 IValidateFields Methods

The methods of the IValidateFields class are listed here.

Public Methods

	Name	Description
	GetEnableValidateFields (see page 17078)	This is GetEnableValidateFields, a member of class IValidateFields.
	SetEnableValidateFields (see page 17078)	This is SetEnableValidateFields, a member of class IValidateFields.
	ValidateFields (see page 17078)	This is ValidateFields, a member of class IValidateFields.

99.2.14.2.1 IValidateFields.GetEnableValidateFields Method

Pascal

```
function GetEnableValidateFields: Boolean;
```

C++

```
__fastcall Boolean GetEnableValidateFields();
```

Description

This is GetEnableValidateFields, a member of class IValidateFields.

99.2.14.2.2 IValidateFields.SetEnableValidateFields Method

Pascal

```
procedure SetEnableValidateFields(Value: Boolean);
```

C++

```
__fastcall SetEnableValidateFields(Boolean Value);
```

Description

This is SetEnableValidateFields, a member of class IValidateFields.

99.2.14.2.3 IValidateFields.ValidateFields Method

Pascal

```
function ValidateFields(AddIntf: IAddScriptElements): Boolean;
```

C++

```
__fastcall Boolean ValidateFields(IAddScriptElements AddIntf);
```

Description

This is ValidateFields, a member of class IValidateFields.

99.2.14.3 IValidateFields Properties

The properties of the IValidateFields class are listed here.

Public Properties

	Name	Description
	EnableValidateFields (see page 17078)	This is EnableValidateFields, a member of class IValidateFields.

99.2.14.3.1 IValidateFields.EnableValidateFields Property

Pascal

```
property EnableValidateFields: Boolean;
```

C++

```
__property Boolean EnableValidateFields;
```

Description

This is EnableValidateFields, a member of class IValidateFields.

99.2.15 IWebDisplay Interface

Class Hierarchy

MidItems.IWebDisplay

Pascal

```
IWebDisplay = interface;
```

C++

```
__interface IWebDisplay;
```

File

MidItems

See Also

TWebDisplay (see page 16911)

99.2.15.1 IWebDisplay Members

The following tables list the members exposed by IWebDisplay.

Public Methods

	Name	Description
	GetIsMultipleRecordView (see page 17079)	This is GetIsMultipleRecordView, a member of class IWebDisplay.

Public Properties

	Name	Description
R	IsMultipleRecordView (see page 17080)	

99.2.15.2 IWebDisplay Methods

The methods of the IWebDisplay class are listed here.

Public Methods

	Name	Description
	GetIsMultipleRecordView (see page 17079)	This is GetIsMultipleRecordView, a member of class IWebDisplay.

99.2.15.2.1 IWebDisplay.GetIsMultipleRecordView Method

Pascal

```
function GetIsMultipleRecordView: Boolean;
```

C++

```
__fastcall Boolean GetIsMultipleRecordView();
```

Description

This is GetIsMultipleRecordView, a member of class IWebDisplay.

99.2.15.3 IWebDisplay Properties

The properties of the IWebDisplay class are listed here.

Public Properties

	Name	Description
R	IsMultipleRecordView (see page 17080)	

99.2.15.3.1 IWebDisplay.IsMultipleRecordView Property

Pascal

```
property IsMultipleRecordView: Boolean;
```

C++

```
__property Boolean IsMultipleRecordView;
```

Description

.

See Also

GetIsMultipleRecordView (see page 17079)

99.2.16 IWebDisplayReference Interface

Class Hierarchy

MidItems.IWebDisplayReference

Pascal

```
IWebDisplayReference = interface;
```

C++

```
__interface IWebDisplayReference;
```

File

MidItems

See Also

TWebDisplay (see page 16911)

99.2.16.1 IWebDisplayReference Members

The following tables list the members exposed by IWebDisplayReference.

Public Methods

	Name	Description
	GetWebDisplayComponent (see page 17081)	This is GetWebDisplayComponent, a member of class IWebDisplayReference.

Public Properties

	Name	Description
R	WebDisplayComponent (see page 17081)	

99.2.16.2 IWebDisplayReference Methods

The methods of the IWebDisplayReference class are listed here.

Public Methods

	Name	Description
	GetWebDisplayComponent (see page 17081)	This is GetWebDisplayComponent, a member of class IWebDisplayReference.

99.2.16.2.1 IWebDisplayReference.GetWebDisplayComponent Method

Pascal

```
function GetWebDisplayComponent: TComponent;
```

C++

```
__fastcall TComponent * GetWebDisplayComponent();
```

Description

This is GetWebDisplayComponent, a member of class IWebDisplayReference.

99.2.16.3 IWebDisplayReference Properties

The properties of the IWebDisplayReference class are listed here.

Public Properties

	Name	Description
R	WebDisplayComponent (see page 17081)	

99.2.16.3.1 IWebDisplayReference.WebDisplayComponent Property

Pascal

```
property WebDisplayComponent: TComponent;
```

C++

```
__property TComponent * WebDisplayComponent;
```

See Also

GetWebDisplayComponent (see page 17081)

99.2.17 IXMLDisplay Interface

Class Hierarchy

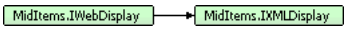

Pascal

```
IXMLDisplay = interface(IWebDisplay);
```

C++

```
__interface IXMLDisplay : public IWebDisplay;
```

File

MidItems

Description

This is class MidItems.IXMLDisplay.

99.2.17.1 IXMLDisplay Members

The following tables list the members exposed by IXMLDisplay.

Public Methods

	Name	Description
	GetIsMultipleRecordView (see page 17082)	This is GetIsMultipleRecordView, a member of class IXMLDisplay.
	GetXMLDisplayName (see page 17083)	This is GetXMLDisplayName, a member of class IXMLDisplay.
	GetXMLRowSetName (see page 17083)	This is GetXMLRowSetName, a member of class IXMLDisplay.

Public Properties

	Name	Description
R	IsMultipleRecordView (see page 17083)	
R	XMLDisplayName (see page 17084)	This is XMLDisplayName, a member of class IXMLDisplay.
R	XMLRowSetName (see page 17084)	This is XMLRowSetName, a member of class IXMLDisplay.

99.2.17.2 IXMLDisplay Methods

The methods of the IXMLDisplay class are listed here.

Public Methods

	Name	Description
	GetIsMultipleRecordView (see page 17082)	This is GetIsMultipleRecordView, a member of class IXMLDisplay.
	GetXMLDisplayName (see page 17083)	This is GetXMLDisplayName, a member of class IXMLDisplay.
	GetXMLRowSetName (see page 17083)	This is GetXMLRowSetName, a member of class IXMLDisplay.

99.2.17.2.1 IXMLDisplay.GetIsMultipleRecordView Method

Pascal

```
function GetIsMultipleRecordView: Boolean;
```

C++

```
__fastcall Boolean GetIsMultipleRecordView();
```

Description

This is GetIsMultipleRecordView, a member of class IXMLDisplay.

99.2.17.2.2 IXMLDisplay.GetXMLDisplayName Method

Pascal

```
function GetXMLDisplayName: string;
```

C++

```
__fastcall AnsiString GetXMLDisplayName();
```

Description

This is GetXMLDisplayName, a member of class IXMLDisplay.

99.2.17.2.3 IXMLDisplay.GetXMLRowSetName Method

Pascal

```
function GetXMLRowSetName: string;
```

C++

```
__fastcall AnsiString GetXMLRowSetName();
```

Description

This is GetXMLRowSetName, a member of class IXMLDisplay.

99.2.17.3 IXMLDisplay Properties

The properties of the IXMLDisplay class are listed here.

Public Properties

	Name	Description
R	IsMultipleRecordView (see page 17083)	
R	XMLDisplayName (see page 17084)	This is XMLDisplayName, a member of class IXMLDisplay.
R	XMLRowSetName (see page 17084)	This is XMLRowSetName, a member of class IXMLDisplay.

99.2.17.3.1 IXMLDisplay.IsMultipleRecordView Property

Pascal

```
property IsMultipleRecordView: Boolean;
```

C++

```
__property Boolean IsMultipleRecordView;
```

Description

.

See Also

GetIsMultipleRecordView (see page 17079)

99.2.17.3.2 IXMLDisplay.XMLDisplayName Property

Pascal

```
property XMLDisplayName: string;
```

C++

```
__property AnsiString XMLDisplayName;
```

Description

This is XMLDisplayName, a member of class IXMLDisplay.

99.2.17.3.3 IXMLDisplay.XMLRowSetName Property

Pascal

```
property XMLRowSetName: string;
```

C++

```
__property AnsiString XMLRowSetName;
```

Description

This is XMLRowSetName, a member of class IXMLDisplay.

99.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AddBoolAttrib (see page 17085)	This is function MidItems.AddBoolAttrib.
	AddCustomAttrib (see page 17085)	This is function MidItems.AddCustomAttrib.
	AddIntAttrib (see page 17086)	This is function MidItems.AddIntAttrib.
	AddQuotedAttrib (see page 17086)	This is function MidItems.AddQuotedAttrib.
	AddStringAttrib (see page 17086)	This is function MidItems.AddStringAttrib.
	CanAddClassHelper (see page 17087)	This is function MidItems.CanAddClassHelper.
	CreateDefaultButtonClasses (see page 17087)	This is function MidItems.CreateDefaultButtonClasses.
	DeclareSubmitForm (see page 17087)	This is function MidItems.DeclareSubmitForm.
	FindDispatcher (see page 17088)	This is function MidItems.FindDispatcher.
	FindProducer (see page 17088)	This is function MidItems.FindProducer.
	FormatColumnData (see page 17088)	This is function MidItems.FormatColumnData.
	FormatColumnHeading (see page 17089)	This is function MidItems.FormatColumnHeading.
	GetItemValuesFromDataSet (see page 17089)	This is function MidItems.GetItemValuesFromDataSet.

99.3.1 MidItems.AddBoolAttrib Function

Pascal

```
procedure AddBoolAttrib(var Attribs: string; const Attrib: string; Value: Boolean);
```

C++

```
AddBoolAttrib(AnsiString Attribs, const AnsiString Attrib, Boolean Value);
```

File

MidItems

Description

This is function MidItems.AddBoolAttrib.

99.3.2 MidItems.AddCustomAttrib Function

Pascal

```
procedure AddCustomAttrib(var Attribs: string; Value: string);
```

C++

```
AddCustomAttrib(AnsiString Attribs, AnsiString Value);
```

File

MidItems

Description

This is function MidItems.AddCustomAttrib.

99.3.3 MidItems.AddIntAttrib Function

Pascal

```
procedure AddIntAttrib(var Attribs: string; const Attrib: string; Value: Integer);
```

C++

```
AddIntAttrib(AnsiString Attribs, const AnsiString Attrib, int Value);
```

File

MidItems

Description

This is function MidItems.AddIntAttrib.

99.3.4 MidItems.AddQuotedAttrib Function

Pascal

```
procedure AddQuotedAttrib(var Attribs: string; const Attrib: string; const Value: string);
```

C++

```
AddQuotedAttrib(AnsiString Attribs, const AnsiString Attrib, const AnsiString Value);
```

File

MidItems

Description

This is function MidItems.AddQuotedAttrib.

99.3.5 MidItems.AddStringAttrib Function

Pascal

```
procedure AddStringAttrib(var Attribs: string; const Attrib: string; const Value: string);
```

C++

```
AddStringAttrib(AnsiString Attribs, const AnsiString Attrib, const AnsiString Value);
```

File

MidItems

Description

This is function MidItems.AddStringAttrib.

99.3.6 MidItems.CanAddClassHelper Function

Pascal

```
function CanAddClassHelper(AEditor: TComponent; AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
Boolean CanAddClassHelper(TComponent * AEditor, TComponent * AParent, TClass AClass);
```

File

MidItems

Description

This is function MidItems.CanAddClassHelper.

99.3.7 MidItems.CreateDefaultButtonClasses Function

Pascal

```
procedure CreateDefaultButtonClasses(const Classes: array of TWebButtonClass; Container:
TWebComponentList);
```

C++

```
CreateDefaultButtonClasses(const array of TWebButtonClass Classes, TWebComponentList
Container);
```

File

MidItems

Description

This is function MidItems.CreateDefaultButtonClasses.

99.3.8 MidItems.DeclareSubmitForm Function

Pascal

```
procedure DeclareSubmitForm(Component: TComponent; XMLBroker: TXMLBroker; AddIntf:
IAddScriptElements);
```

C++

```
DeclareSubmitForm(TComponent * Component, TXMLBroker XMLBroker, IAddScriptElements AddIntf);
```

File

MidItems

Description

This is function MidItems.DeclareSubmitForm.

99.3.9 MidItems.FindDispatcher Function

Pascal

```
function FindDispatcher(Component: TComponent): IWebDispatcherAccess;
```

C++

```
IWebDispatcherAccess FindDispatcher(TComponent * Component);
```

File

MidItems

Description

This is function MidItems.FindDispatcher.

99.3.10 MidItems.FindProducer Function

Pascal

```
function FindProducer(Component: TComponent): TCustomContentProducer;
```

C++

```
TCustomContentProducer FindProducer(TComponent * Component);
```

File

MidItems

Description

This is function MidItems.FindProducer.

99.3.11 MidItems.FormatColumnData Function

Pascal

```
function FormatColumnData(Field: TWebDataDisplay; Content: string): string;
```

C++

```
AnsiString FormatColumnData(TWebDataDisplay Field, AnsiString Content);
```

File

MidItems

Description

This is function MidItems.FormatColumnData.

99.3.12 MidItems.FormatColumnHeading Function

Pascal

```
function FormatColumnHeading(Field: TWebDataDisplay): string;
```

C++

```
AnsiString FormatColumnHeading(TWebDataDisplay Field);
```

File

MidItems

Description

This is function MidItems.FormatColumnHeading.

99.3.13 MidItems.GetItemValuesFromDataSet Function

Pascal

```
function GetItemValuesFromDataSet(DataSet: TDataSet; const ItemsField: string; const
ValuesField: string; var ItemsStrings: TStrings; var ValuesStrings: TStrings): Boolean;
```

C++

```
Boolean GetItemValuesFromDataSet(TDataSet DataSet, const AnsiString ItemsField, const
AnsiString ValuesField, TStrings ItemsStrings, TStrings ValuesStrings);
```

File

MidItems

Description

This is function MidItems.GetItemValuesFromDataSet.

99.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCaptionPosition (see page 17090)	TCaptionPosition indicates the relative position of a label to the HTML control it labels.
	TFormMethod (see page 17091)	TFormMethod represents the method attribute on the <form> tag of an HTML form.
	TLayoutState (see page 17091)	This is record MidItems.TLayoutState.
	TTextAreaWrap (see page 17092)	TTextAreaWrap indicates how an HTML display control should treat items that are too wide for the physical width of the control.

99.4.1 MidItems.TCaptionPosition Enumeration

TCaptionPosition indicates the relative position of a label to the HTML control it labels.

Pascal

```
TCaptionPosition = (
  capLeft,
  capRight,
  capAbove,
  capBelow
);
```

C++

```
enum TCaptionPosition {
  capLeft,
  capRight,
  capAbove,
  capBelow
};
```

File

MidItems

Description

TCaptionPosition can have one of the following values.

Value	Meaning
capLeft	The caption appears to the left of the HTML control.
capRight	The caption appears to the right of the HTML control.
capAbove	The caption appears above the HTML control.
capBelow	The caption appears below the HTML control.

99.4.2 MidItems.TFormMethod Enumeration

TFormMethod represents the method attribute on the <form> tag of an HTML form.

Pascal

```
TFormMethod = (
  fmPost,
  fmGet
);
```

C++

```
enum TFormMethod {
  fmPost,
  fmGet
};
```

File

MidItems

Description

HTML forms include a <form> tag that distinguishes them from other types of HTML pages. This<form> tag includes a method attribute, which determines the method header of the HTTP request message that the form sends to its server. The method type corresponds to the MethodType property of the HTTP request message.

Components that generate HTML forms use TFormMethod to specify the method attribute of the <form> tag they generate, and hence the method header of the messages that they send to the server.

The following table describes the possible values:

Value	Corresponding MethodType	Meaning
fmPost	mtPost	The value of the method attribute is 'post'. The form submits messages that post displayed values to the server application. Named value pairs are passed as content fields of the HTTP message.
fmGet	mtGet	The value of the method attribute is 'get'. The form submits messages that request values it can display. Named value pairs are supplied as query fields in the URL.

See Also

TWebRequest

99.4.3 MidItems.TLayoutState Enumeration

Pascal

```
TLayoutState = (
  lsUnknown,
  lsFields,
  lsButtons
);
```

C++

```
enum TLayoutState {
```

```
    lsUnknown,
    lsFields,
    lsButtons
  };
```

File

MidItems

Description

This is record MidItems.TLayoutState.

99.4.4 MidItems.TTextAreaWrap Enumeration

TTextAreaWrap indicates how an HTML display control should treat items that are too wide for the physical width of the control.

Pascal

```
  TTextAreaWrap = (
    wrOff,
    wrPhysical,
    wrVirtual
  );
```

C++

```
  enum TTextAreaWrap {
    wrOff,
    wrPhysical,
    wrVirtual
  };
```

File

MidItems

Description

TTextAreaWrap can have one of the following values.

Value	Meaning
wrOff	The control does not wrap long lines. Only newline characters that are part of the text string cause the lines in the control to wrap.
wrPhysical	The control formats its text value by wrapping long lines so that no text is hidden. The control inserts carriage returns into the text string.
wrVirtual	The control formats its text value by wrapping long lines so that no text is hidden but does not change the actual value of the text string.

99.5 Types

The following table lists types in this documentation.

Types

Name	Description
IMidasWebDataSet (see page 17093)	This is type MidItems.IMidasWebDataSet.
TWebButtonClass (see page 17093)	This is type MidItems.TWebButtonClass.
TWebDataDisplayClass (see page 17094)	This is type MidItems.TWebDataDisplayClass.
TWebDataInputClass (see page 17094)	This is type MidItems.TWebDataInputClass.
TWebStatusClass (see page 17094)	This is type MidItems.TWebStatusClass.

99.5.1 MidItems.IMidasWebDataSet Type

Pascal

```
IMidasWebDataSet = IInetXWebDataSet;
```

C++

```
IInetXWebDataSet IMidasWebDataSet;
```

File

MidItems

Description

This is type MidItems.IMidasWebDataSet.

99.5.2 MidItems.TWebButtonClass Type

Pascal

```
TWebButtonClass = class of TWebButton;
```

C++

```
TWebButton TWebButtonClass;
```

File

MidItems

Description

This is type MidItems.TWebButtonClass.

99.5.3 MidItems.TWebDataDisplayClass Type

Pascal

```
TWebDataDisplayClass = class of TWebDataDisplay;
```

C++

```
TWebDataDisplay TWebDataDisplayClass;
```

File

MidItems

Description

This is type MidItems.TWebDataDisplayClass.

99.5.4 MidItems.TWebDataInputClass Type

Pascal

```
TWebDataInputClass = class of TWebDataInput;
```

C++

```
TWebDataInput TWebDataInputClass;
```

File

MidItems

Description

This is type MidItems.TWebDataInputClass.

99.5.5 MidItems.TWebStatusClass Type

Pascal

```
TWebStatusClass = class of TWebStatus;
```

C++

```
TWebStatus TWebStatusClass;
```

File

MidItems

Description

This is type MidItems.TWebStatusClass.

99.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
FormMethodNames (see page 17095)	This is constant MidItems.FormMethodNames.
HTMLAlignValues (see page 17096)	This is constant MidItems.HTMLAlignValues.
HTMLVAlignValues (see page 17096)	This is constant MidItems.HTMLVAlignValues.
sActionAttr (see page 17096)	This is constant MidItems.sActionAttr.
sAlignAttr (see page 17097)	This is constant MidItems.sAlignAttr.
sBgColorAttr (see page 17097)	This is constant MidItems.sBgColorAttr.
sBorderAttr (see page 17097)	This is constant MidItems.sBorderAttr.
sCellPaddingAttr (see page 17098)	This is constant MidItems.sCellPaddingAttr.
sCellSpacingAttr (see page 17098)	This is constant MidItems.sCellSpacingAttr.
sCheckedAttr (see page 17098)	This is constant MidItems.sCheckedAttr.
sClassAttr (see page 17099)	This is constant MidItems.sClassAttr.
sColsAttr (see page 17099)	This is constant MidItems.sColsAttr.
sColSpanAttr (see page 17099)	This is constant MidItems.sColSpanAttr.
sEncTypeAttr (see page 17100)	This is constant MidItems.sEncTypeAttr.
sImageHeightAttr (see page 17100)	This is constant MidItems.sImageHeightAttr.
sImageWidthAttr (see page 17100)	This is constant MidItems.sImageWidthAttr.
sMaxLengthAttr (see page 17101)	This is constant MidItems.sMaxLengthAttr.
sMethodAttr (see page 17101)	This is constant MidItems.sMethodAttr.
sMultipleAttr (see page 17101)	This is constant MidItems.sMultipleAttr.
sNameAttr (see page 17102)	This is constant MidItems.sNameAttr.
sOnClickAttr (see page 17102)	This is constant MidItems.sOnClickAttr.
sReadOnlyAttr (see page 17102)	This is constant MidItems.sReadOnlyAttr.
sRowsAttr (see page 17103)	This is constant MidItems.sRowsAttr.
sSelectedAttr (see page 17103)	This is constant MidItems.sSelectedAttr.
sSizeAttr (see page 17103)	This is constant MidItems.sSizeAttr.
sStyleAttr (see page 17104)	This is constant MidItems.sStyleAttr.
sTabIndexAttr (see page 17104)	This is constant MidItems.sTabIndexAttr.
sTypeAttr (see page 17104)	This is constant MidItems.sTypeAttr.
sVAlignAttr (see page 17105)	This is constant MidItems.sVAlignAttr.
sValueAttr (see page 17105)	This is constant MidItems.sValueAttr.
sWidthAttr (see page 17105)	This is constant MidItems.sWidthAttr.
sWrapAttr (see page 17106)	This is constant MidItems.sWrapAttr.
TextAreaWrap (see page 17106)	This is constant MidItems.TextAreaWrap.

99.6.1 MidItems.FormMethodNames Constant

Pascal

```
FormMethodNames: array[TFormMethod] of string = ('POST', 'GET');
```

C++

```
array[TFormMethod] of string FormMethodNames = ('POST', 'GET');
```

File

MidItems

Description

This is constant MidItems.FormMethodNames.

99.6.2 MidItems.HTMLAlignValues Constant

Pascal

```
HTMLAlignValues: array[THTMLAlign] of string = ('', 'left', 'right', 'center');
```

C++

```
array[THTMLAlign] of string HTMLAlignValues = ('', 'left', 'right', 'center');
```

File

MidItems

Description

This is constant MidItems.HTMLAlignValues.

99.6.3 MidItems.HTMLVAlignValues Constant

Pascal

```
HTMLVAlignValues: array[THTMLVAlign] of string = ('', 'top', 'middle', 'bottom', 'baseline');
```

C++

```
array[THTMLVAlign] of string HTMLVAlignValues = ('', 'top', 'middle', 'bottom', 'baseline');
```

File

MidItems

Description

This is constant MidItems.HTMLVAlignValues.

99.6.4 MidItems.sActionAttr Constant

Pascal

```
sActionAttr = 'action';
```

C++

```
sActionAttr = 'action';
```

File

MidItems

Description

This is constant MidItems.sActionAttr.

99.6.5 MidItems.sAlignAttr Constant

Pascal

```
sAlignAttr = 'align';
```

C++

```
sAlignAttr = 'align';
```

File

MidItems

Description

This is constant MidItems.sAlignAttr.

99.6.6 MidItems.sBgColorAttr Constant

Pascal

```
sBgColorAttr = 'bgcolor';
```

C++

```
sBgColorAttr = 'bgcolor';
```

File

MidItems

Description

This is constant MidItems.sBgColorAttr.

99.6.7 MidItems.sBorderAttr Constant

Pascal

```
sBorderAttr = 'border';
```

C++

```
sBorderAttr = 'border';
```

File

MidItems

Description

This is constant MidItems.sBorderAttr.

99.6.8 MidItems.sCellPaddingAttr Constant

Pascal

```
sCellPaddingAttr = 'cellpadding';
```

C++

```
sCellPaddingAttr = 'cellpadding';
```

File

MidItems

Description

This is constant MidItems.sCellPaddingAttr.

99.6.9 MidItems.sCellSpacingAttr Constant

Pascal

```
sCellSpacingAttr = 'cellspacing';
```

C++

```
sCellSpacingAttr = 'cellspacing';
```

File

MidItems

Description

This is constant MidItems.sCellSpacingAttr.

99.6.10 MidItems.sCheckedAttr Constant

Pascal

```
sCheckedAttr = 'checked';
```

C++

```
sCheckedAttr = 'checked';
```

File

MidItems

Description

This is constant MidItems.sCheckedAttr.

99.6.11 MidItems.sClassAttr Constant

Pascal

```
sClassAttr = 'class';
```

C++

```
sClassAttr = 'class';
```

File

MidItems

Description

This is constant MidItems.sClassAttr.

99.6.12 MidItems.sColsAttr Constant

Pascal

```
sColsAttr = 'cols';
```

C++

```
sColsAttr = 'cols';
```

File

MidItems

Description

This is constant MidItems.sColsAttr.

99.6.13 MidItems.sColSpanAttr Constant

Pascal

```
sColSpanAttr = 'colspan';
```

C++

```
sColSpanAttr = 'colspan';
```

File

MidItems

Description

This is constant MidItems.sColSpanAttr.

99.6.14 MidItems.sEncTypeAttr Constant

Pascal

```
sEncTypeAttr = 'enctype';
```

C++

```
sEncTypeAttr = 'enctype';
```

File

MidItems

Description

This is constant MidItems.sEncTypeAttr.

99.6.15 MidItems.sImageHeightAttr Constant

Pascal

```
sImageHeightAttr = 'imageheight';
```

C++

```
sImageHeightAttr = 'imageheight';
```

File

MidItems

Description

This is constant MidItems.sImageHeightAttr.

99.6.16 MidItems.sImageWidthAttr Constant

Pascal

```
sImageWidthAttr = 'imagewidth';
```

C++

```
sImageWidthAttr = 'imagewidth';
```

File

MidItems

Description

This is constant MidItems.sImageWidthAttr.

99.6.17 MidItems.sMaxLengthAttr Constant

Pascal

```
sMaxLengthAttr = 'maxlength';
```

C++

```
sMaxLengthAttr = 'maxlength';
```

File

MidItems

Description

This is constant MidItems.sMaxLengthAttr.

99.6.18 MidItems.sMethodAttr Constant

Pascal

```
sMethodAttr = 'method';
```

C++

```
sMethodAttr = 'method';
```

File

MidItems

Description

This is constant MidItems.sMethodAttr.

99.6.19 MidItems.sMultipleAttr Constant

Pascal

```
sMultipleAttr = 'multiple';
```

C++

```
sMultipleAttr = 'multiple';
```

File

MidItems

Description

This is constant MidItems.sMultipleAttr.

99.6.20 MidItems.sNameAttr Constant

Pascal

```
sNameAttr = 'name';
```

C++

```
sNameAttr = 'name';
```

File

MidItems

Description

This is constant MidItems.sNameAttr.

99.6.21 MidItems.sOnClickAttr Constant

Pascal

```
sOnClickAttr = 'onclick';
```

C++

```
sOnClickAttr = 'onclick';
```

File

MidItems

Description

This is constant MidItems.sOnClickAttr.

99.6.22 MidItems.sReadOnlyAttr Constant

Pascal

```
sReadOnlyAttr = 'readonly';
```

C++

```
sReadOnlyAttr = 'readonly';
```

File

MidItems

Description

This is constant MidItems.sReadOnlyAttr.

99.6.23 MidItems.sRowsAttr Constant

Pascal

```
sRowsAttr = 'rows';
```

C++

```
sRowsAttr = 'rows';
```

File

MidItems

Description

This is constant MidItems.sRowsAttr.

99.6.24 MidItems.sSelectedAttr Constant

Pascal

```
sSelectedAttr = 'selected';
```

C++

```
sSelectedAttr = 'selected';
```

File

MidItems

Description

This is constant MidItems.sSelectedAttr.

99.6.25 MidItems.sSizeAttr Constant

Pascal

```
sSizeAttr = 'size';
```

C++

```
sSizeAttr = 'size';
```

File

MidItems

Description

This is constant MidItems.sSizeAttr.

99.6.26 MidItems.sStyleAttr Constant

Pascal

```
sStyleAttr = 'style';
```

C++

```
sStyleAttr = 'style';
```

File

MidItems

Description

This is constant MidItems.sStyleAttr.

99.6.27 MidItems.sTabIndexAttr Constant

Pascal

```
sTabIndexAttr = 'tabindex';
```

C++

```
sTabIndexAttr = 'tabindex';
```

File

MidItems

Description

This is constant MidItems.sTabIndexAttr.

99.6.28 MidItems.sTypeAttr Constant

Pascal

```
sTypeAttr = 'type';
```

C++

```
sTypeAttr = 'type';
```

File

MidItems

Description

This is constant MidItems.sTypeAttr.

99.6.29 MidItems.sVAlignAttr Constant

Pascal

```
sVAlignAttr = 'valign';
```

C++

```
sVAlignAttr = 'valign';
```

File

MidItems

Description

This is constant MidItems.sVAlignAttr.

99.6.30 MidItems.sValueAttr Constant

Pascal

```
sValueAttr = 'value';
```

C++

```
sValueAttr = 'value';
```

File

MidItems

Description

This is constant MidItems.sValueAttr.

99.6.31 MidItems.sWidthAttr Constant

Pascal

```
sWidthAttr = 'width';
```

C++

```
sWidthAttr = 'width';
```

File

MidItems

Description

This is constant MidItems.sWidthAttr.

99.6.32 MidItems.sWrapAttr Constant

Pascal

```
sWrapAttr = 'wrap';
```

C++

```
sWrapAttr = 'wrap';
```

File

MidItems

Description

This is constant MidItems.sWrapAttr.

99.6.33 MidItems.TextAreaWrap Constant

Pascal

```
TextAreaWrap: array[TTextAreaWrap] of string = ('OFF', 'PHYSICAL', 'VIRTUAL');
```

C++

```
array[TTextAreaWrap] of string TextAreaWrap = ('OFF', 'PHYSICAL', 'VIRTUAL');
```

File

MidItems

Description

This is constant MidItems.TextAreaWrap.

100 MidProd Namespace

This is namespace MidProd.

Topics

Name	Description
Classes (see page 17109)	The following table lists classes in this documentation.
Functions (see page 17160)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 17161)	The following table lists structs, records, enums in this documentation.
Types (see page 17162)	The following table lists types in this documentation.
Constants (see page 17164)	The following table lists constants in this documentation.

Classes

	Name	Description
	TBaseInetXPageProducer (see page 17109)	TBaseInetXPageProducer is an base class for InternetExpress page producer classes.
	TCustomInetXPageProducer (see page 17131)	TCustomInetXPageProducer is the base class for page producers that display live database information in an InternetExpress application.
	TInetXPageElements (see page 17142)	This is class MidProd.TInetXPageElements.
	TInetXPageProducer (see page 17145)	TInetXPageProducer assembles an HTML document that represents live database information from an application server.
	TMidasPageProducer (see page 17158)	TMidasPageProducer assembles an HTML document that represents live database information from an application server.

Constants

Name	Description
sXMLBroker (see page 17164)	This is constant MidProd.sXMLBroker.
sXMLReadyVar (see page 17164)	This is constant MidProd.sXMLReadyVar.

Functions

	Name	Description
	AddStdIncludes (see page 17160)	This is function MidProd.AddStdIncludes.
	PathInfoToRelativePath (see page 17160)	This is function MidProd.PathInfoToRelativePath.

Structs, Records, Enums

	Name	Description
	TStdInclude (see page 17161)	This is record MidProd.TStdInclude.

Types

Name	Description
TBaseMidasPageProducer (see page 17162)	This is type MidProd.TBaseMidasPageProducer.
TCustomMidasPageProducer (see page 17162)	This is type MidProd.TCustomMidasPageProducer.
TMidasPageElements (see page 17163)	This is type MidProd.TMidasPageElements.
TStdIncludes (see page 17163)	This is type MidProd.TStdIncludes.
TXMLDataEvent (see page 17163)	This is type MidProd.TXMLDataEvent.

100.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TBaseInetXPageProducer (see page 17109)	TBaseInetXPageProducer is an base class for InternetExpress page producer classes.
	TCustomInetXPageProducer (see page 17131)	TCustomInetXPageProducer is the base class for page producers that display live database information in an InternetExpress application.
	TInetXPageElements (see page 17142)	This is class MidProd.TInetXPageElements.
	TInetXPageProducer (see page 17145)	TInetXPageProducer assembles an HTML document that represents live database information from an application server.
	TMidasPageProducer (see page 17158)	TMidasPageProducer assembles an HTML document that represents live database information from an application server.

100.1.1 TBaseInetXPageProducer Class

TBaseInetXPageProducer is an base class for InternetExpress page producer classes.

Class Hierarchy

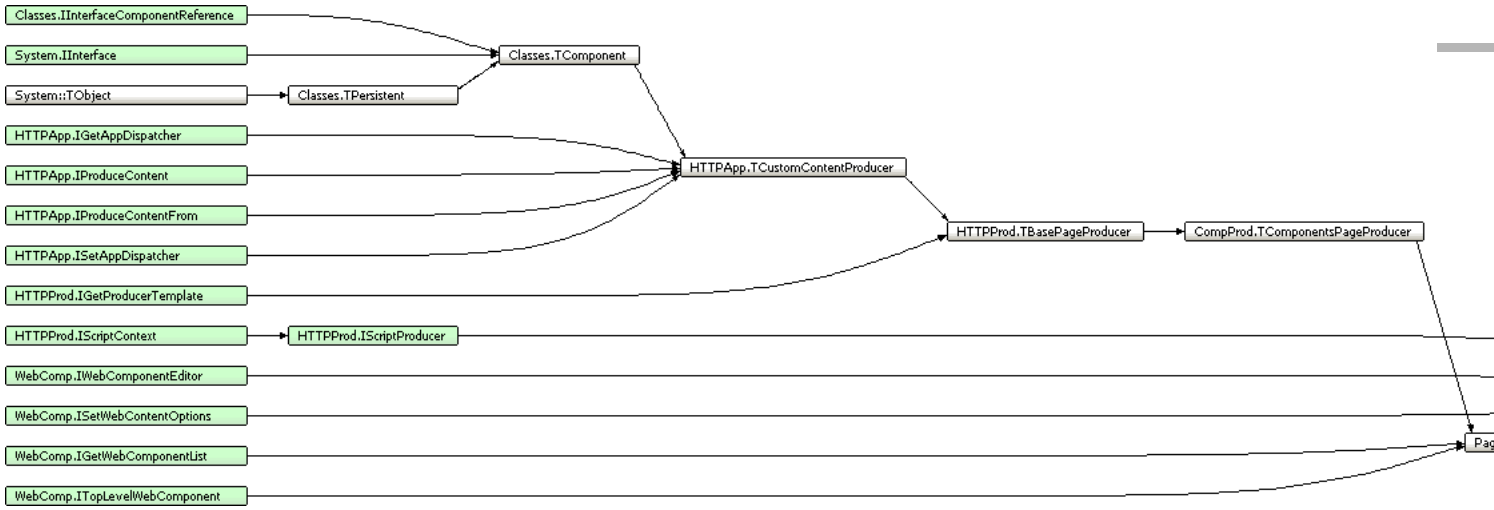

Pascal

```
TBaseInetXPageProducer = class(TBasePageItemsProducer, IWebComponentEditor, IScriptProducer,
ISetWebContentOptions);
```

C++

```
class TBaseInetXPageProducer : public TBasePageItemsProducer, public IWebComponentEditor,
public IScriptProducer, public ISetWebContentOptions;
```

File

MidProd

Description

Do not create instances of TBaseInetXPageProducer. Instead, use one of its descendant classes, such as TInetXPageProducer (see page 17145).

TBaseInetXPageProducer introduces the logic for working with an XML broker to obtain database information and for translating the special HTML-transparent tags that are used in InternetExpress applications. Unlike its immediate descendant (TCustomInetXPageProducer (see page 17131)) however, TBaseInetXPageProducer does not initialize the HTMLDoc property to the default template for an InternetExpress application, although it dynamically generates the InternetExpress-specific tags.

You can use TBaseInetXPageProducer as a base class when writing a page producer that does not use the default template, but which translates InternetExpress tags into HTML and javascript.

See Also

TInetXPageProducer (see page 17145)

TXMLBroker (see page 26049)

100.1.1.1 TBaseInetXPageProducer Members

The following tables list the members exposed by TBaseInetXPageProducer.

Protected Events

	Name	Description
	OnHTMLTag (see page 15280)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddIncludes (see page 17116)	This is AddIncludes, a member of class TBaseInetXPageProducer.
V	AddScriptComponents (see page 17117)	This is AddScriptComponents, a member of class TBaseInetXPageProducer.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 17117)	Indicates whether a specified class can be used as a child Web item.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CreatePageElements (see page 17118)	This is CreatePageElements, a member of class TBaseInetXPageProducer.
V	CreateScriptManager (see page 17118)	This is CreateScriptManager, a member of class TBaseInetXPageProducer.
V	DeclareFunctions (see page 17118)	This is DeclareFunctions, a member of class TBaseInetXPageProducer.
V	DeclareIncludes (see page 17118)	This is DeclareIncludes, a member of class TBaseInetXPageProducer.
V	DeclareRowSets (see page 17119)	This is DeclareRowSets, a member of class TBaseInetXPageProducer.
V	DeclareScript (see page 17119)	This is DeclareScript, a member of class TBaseInetXPageProducer.
V	DeclareStyles (see page 17119)	This is DeclareStyles, a member of class TBaseInetXPageProducer.
V	DeclareWarnings (see page 17119)	This is DeclareWarnings, a member of class TBaseInetXPageProducer.
V	DeclareXMLDocuments (see page 17120)	This is DeclareXMLDocuments, a member of class TBaseInetXPageProducer.
V	DeclareXMLIslands (see page 17120)	This is DeclareXMLIslands, a member of class TBaseInetXPageProducer.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoAfterGetContent (see page 17120)	This is DoAfterGetContent, a member of class TBaseInetXPageProducer.
	DoAfterGetXMLData (see page 17120)	This is DoAfterGetXMLData, a member of class TBaseInetXPageProducer.
	DoBeforeGetContent (see page 17120)	This is DoBeforeGetContent, a member of class TBaseInetXPageProducer.
	DoBeforeGetXMLData (see page 17121)	This is DoBeforeGetXMLData, a member of class TBaseInetXPageProducer.
	DoTagEvent (see page 17121)	This is DoTagEvent, a member of class TBaseInetXPageProducer.
	FindComponent (see page 5619)	This is FindComponent, a member of class TComponentsPageProducer.
	FindComponentName (see page 5619)	This is FindComponentName, a member of class TComponentsPageProducer.
	GeneratePageElements (see page 17122)	This is GeneratePageElements, a member of class TBaseInetXPageProducer.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 18381)	This is GetChildren, a member of class TBasePageItemsProducer.
	GetComponentContent (see page 5619)	This is GetComponentContent, a member of class TComponentsPageProducer.
	GetComponentList (see page 18382)	This is GetComponentList, a member of class TBasePageItemsProducer.
	GetContentOptions (see page 17122)	This is GetContentOptions, a member of class TBaseInetXPageProducer.
	GetDefaultComponentList (see page 18382)	This is GetDefaultComponentList, a member of class TBasePageItemsProducer.
	GetDefaultTemplate (see page 17122)	This is GetDefaultTemplate, a member of class TBaseInetXPageProducer.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetProducerTemplateFileName (see page 15276)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.
	GetProducerTemplateStream (see page 15276)	This is GetProducerTemplateStream, a member of class TBasePageProducer.
	GetScriptEngine (see page 15277)	This is GetScriptEngine, a member of class TBasePageProducer.
	GetScriptManager (see page 17123)	This is GetScriptManager, a member of class TBaseInetXPageProducer.
	GetTagID (see page 15277)	This is GetTagID, a member of class TBasePageProducer.
	GetTemplateFileName (see page 15277)	This is GetTemplateFileName, a member of class TBasePageProducer.
	GetTemplateStream (see page 15277)	This is GetTemplateStream, a member of class TBasePageProducer.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetUseXMLIslands (see page 17123)	This is GetUseXMLIslands, a member of class TBaseInetXPageProducer.
	GetXMLDelta (see page 17124)	This is GetXMLDelta, a member of class TBaseInetXPageProducer.
	GetXMLErrors (see page 17124)	This is GetXMLErrors, a member of class TBaseInetXPageProducer.
	HandleScriptTag (see page 15277)	This is HandleScriptTag, a member of class TBasePageProducer.
	HandleTag (see page 5620)	This is HandleTag, a member of class TComponentsPageProducer.
	HandleTag (see page 15278)	This is HandleTag, a member of class TBasePageProducer.
	ImplCanAddClass (see page 17124)	This is ImplCanAddClass, a member of class TBaseInetXPageProducer.
	ImplHandleTag (see page 15278)	This is ImplHandleTag, a member of class TBasePageProducer.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 15108)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.

	ServerScriptFromStream (see page 15278)	This is ServerScriptFromStream, a member of class TBasePageProducer.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 18382)	This is SetChildOrder, a member of class TBasePageItemsProducer.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetWebContentOptions (see page 17126)	This is SetWebContentOptions, a member of class TBaseInetXPageProducer.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
V	UseScriptEngine (see page 15278)	This is UseScriptEngine, a member of class TBasePageProducer.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	PageElements (see page 17127)	This is PageElements, a member of class TBaseInetXPageProducer.
R	ScriptManager (see page 17127)	This is ScriptManager, a member of class TBaseInetXPageProducer.

Public Events

	Name	Description
	OnAfterGetContent (see page 17129)	Occurs after the page producer generates its HTML document.
	OnAfterGetXMLData (see page 17129)	Occurs after the page producer translates fetches an XML data packet from an XML broker.
	OnBeforeGetContent (see page 17130)	Occurs when the Content method is called.
	OnBeforeGetXMLData (see page 17130)	Occurs before the page producer fetches an XML data packet from an XML broker.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Content (see page 15106)	Returns the content produced by the content producer.
V	Content (see page 15274)	Returns the content produced by the content producer.

	ContentFromStream (see page 17117)	Returns the content produced in response to input read from a stream.
	ContentFromString (see page 15107)	Returns the content produced in response to an input string.
	ContentFromString (see page 15275)	Returns the content produced in response to an input string.
	Create (see page 17115)	Creates an instance of a TBaseInetXPageProducer (see page 17109).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 17115)	Destroys the instance of the TBasePageItemsProducer.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Evaluate (see page 15324)	This is Evaluate, a member of class IScriptProducer.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindXMLBroker (see page 17121)	Returns the XML broker that supplies the data packet specified by a DATAPACKET tag.
	FindXMLBrokerName (see page 17122)	Returns the name of an XML broker given the parameters in a DATAPACKET tag..
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContent (see page 15324)	This is GetContent, a member of class IScriptProducer.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetErrors (see page 15324)	This is GetErrors, a member of class IScriptProducer.
	GetHTMLBlock (see page 15325)	This is GetHTMLBlock, a member of class IScriptProducer.
	GetHTMLBlockCount (see page 15325)	This is GetHTMLBlockCount, a member of class IScriptProducer.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProducerTemplateFileName (see page 15320)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.
	GetProducerTemplateStream (see page 15320)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.
	GetRequiredTags (see page 17123)	Returns the default set of HTML-transparent tags for an InternetExpress page producer.
	GetWebModuleContext (see page 15322)	This is GetWebModuleContext, a member of class IScriptContext.
	GetXMLData (see page 17124)	Obtains an XML data packet from a specified XML broker.
	HandleScriptError (see page 15325)	This is HandleScriptError, a member of class IScriptProducer.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParseStream (see page 15325)	This is ParseStream, a member of class IScriptProducer.
	ParseString (see page 15325)	This is ParseString, a member of class IScriptProducer.
	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	ReplaceTags (see page 15326)	This is ReplaceTags, a member of class IScriptProducer.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetContent (see page 15326)	This is SetContent, a member of class IScriptProducer.
	SetIncludePathURL (see page 17125)	Assigns a value to the IncludePathURL (see page 17127) property.
	SetStyles (see page 17125)	Assigns a value to the Styles (see page 17127) property.
	SetStylesFile (see page 17125)	Assigns a value to the StylesFile (see page 17128) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Write (see page 15326)	This is the overview for the Write method overload.
	WriteItem (see page 15327)	This is WriteItem, a member of class IScriptProducer.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Content (see page 15327)	This is Content, a member of class IScriptProducer.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.
	EnableXMLIslands (see page 17126)	Specifies whether XML data packets should be declared using XML islands if the requesting browser supports them.
R	Errors (see page 15327)	This is Errors, a member of class IScriptProducer.
R	HTMLBlockCount (see page 15328)	This is HTMLBlockCount, a member of class IScriptProducer.
R	HTMLBlocks (see page 15328)	This is HTMLBlocks, a member of class IScriptProducer.
	IncludePathURL (see page 17127)	Indicates the URL where the javascript libraries are installed.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ScriptEngine (see page 15279)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 15279)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.
	Styles (see page 17127)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 17128)	Specifies a file containing the individual styles for the page producer's style sheet.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

R	WebModuleContext (see page 15322)	This is WebModuleContext, a member of class IScriptContext.
R	WebModuleContext (see page 15279)	Provides access to the Web module context object.
R	WebPageItems (see page 18382)	Maintains a list of components that generate HTML content for the producer.

Published Events

	Name	Description
	OnHTMLTag (see page 5620)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

100.1.1.2 TBaseInetXPageProducer.Create Constructor

Creates an instance of a TBaseInetXPageProducer (see page 17109).

Pascal

```
constructor Create(AComponent: TComponent); override;
```

C++

```
virtual __fastcall TBaseInetXPageProducer(TComponent * AComponent);
```

Description

Call Create to instantiate a TBaseInetXPageProducer (see page 17109) component at runtime if it was not created at design time.

AOwner is the component, typically the Web module, that becomes the value of this page producer's Owner property. The Owner is responsible for freeing the TBaseInetXPageProducer (see page 17109) instance.

See Also

Owner (see page 2807)

100.1.1.3 TBaseInetXPageProducer.Destroy Destructor

Destroys the instance of the TBasePageItemsProducer.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBaseInetXPageProducer();
```

Description

Do not call Destroy directly. Instead call Free, which checks that the TBasePageItemsProducer reference is not nil before calling Destroy.

See Also

Create (see page 18380)

Free (see page 22300)

100.1.1.4 TBaseInetXPageProducer Methods

The methods of the TBaseInetXPageProducer class are listed here.

Protected Methods

	Name	Description
	AddIncludes (see page 17116)	This is AddIncludes, a member of class TBaseInetXPageProducer.
	AddScriptComponents (see page 17117)	This is AddScriptComponents, a member of class TBaseInetXPageProducer.
	CanAddClass (see page 17117)	Indicates whether a specified class can be used as a child Web item.
	CreatePageElements (see page 17118)	This is CreatePageElements, a member of class TBaseInetXPageProducer.
	CreateScriptManager (see page 17118)	This is CreateScriptManager, a member of class TBaseInetXPageProducer.
	DeclareFunctions (see page 17118)	This is DeclareFunctions, a member of class TBaseInetXPageProducer.
	DeclareIncludes (see page 17118)	This is DeclareIncludes, a member of class TBaseInetXPageProducer.
	DeclareRowSets (see page 17119)	This is DeclareRowSets, a member of class TBaseInetXPageProducer.
	DeclareScript (see page 17119)	This is DeclareScript, a member of class TBaseInetXPageProducer.
	DeclareStyles (see page 17119)	This is DeclareStyles, a member of class TBaseInetXPageProducer.
	DeclareWarnings (see page 17119)	This is DeclareWarnings, a member of class TBaseInetXPageProducer.
	DeclareXMLDocuments (see page 17120)	This is DeclareXMLDocuments, a member of class TBaseInetXPageProducer.
	DeclareXMLIslands (see page 17120)	This is DeclareXMLIslands, a member of class TBaseInetXPageProducer.
	DoAfterGetContent (see page 17120)	This is DoAfterGetContent, a member of class TBaseInetXPageProducer.
	DoAfterGetXMLData (see page 17120)	This is DoAfterGetXMLData, a member of class TBaseInetXPageProducer.
	DoBeforeGetContent (see page 17120)	This is DoBeforeGetContent, a member of class TBaseInetXPageProducer.
	DoBeforeGetXMLData (see page 17121)	This is DoBeforeGetXMLData, a member of class TBaseInetXPageProducer.
	DoTagEvent (see page 17121)	This is DoTagEvent, a member of class TBaseInetXPageProducer.
	GeneratePageElements (see page 17122)	This is GeneratePageElements, a member of class TBaseInetXPageProducer.
	GetContentOptions (see page 17122)	This is GetContentOptions, a member of class TBaseInetXPageProducer.
	GetDefaultTemplate (see page 17122)	This is GetDefaultTemplate, a member of class TBaseInetXPageProducer.
	GetScriptManager (see page 17123)	This is GetScriptManager, a member of class TBaseInetXPageProducer.
	GetUseXMLIslands (see page 17123)	This is GetUseXMLIslands, a member of class TBaseInetXPageProducer.
	GetXMLDelta (see page 17124)	This is GetXMLDelta, a member of class TBaseInetXPageProducer.
	GetXMLErrors (see page 17124)	This is GetXMLErrors, a member of class TBaseInetXPageProducer.
	ImplCanAddClass (see page 17124)	This is ImplCanAddClass, a member of class TBaseInetXPageProducer.
	SetWebContentOptions (see page 17126)	This is SetWebContentOptions, a member of class TBaseInetXPageProducer.

Public Methods

	Name	Description
	ContentFromStream (see page 17117)	Returns the content produced in response to input read from a stream.
	FindXMLBroker (see page 17121)	Returns the XML broker that supplies the data packet specified by a DATAPACKET tag.
	FindXMLBrokerName (see page 17122)	Returns the name of an XML broker given the parameters in a DATAPACKET tag..
	GetRequiredTags (see page 17123)	Returns the default set of HTML-transparent tags for an InternetExpress page producer.
	GetXMLData (see page 17124)	Obtains an XML data packet from a specified XML broker.
	SetIncludePathURL (see page 17125)	Assigns a value to the IncludePathURL (see page 17127) property.
	SetStyles (see page 17125)	Assigns a value to the Styles (see page 17127) property.
	SetStylesFile (see page 17125)	Assigns a value to the StylesFile (see page 17128) property.

100.1.1.4.1 TBaseInetXPageProducer.AddIncludes Method

Pascal

```
procedure AddIncludes; virtual;
```

C++

```
virtual __fastcall AddIncludes();
```

Description

This is AddIncludes, a member of class TBaseInetXPageProducer.

100.1.1.4.2 TBaseInetXPageProducer.AddScriptComponents Method

Pascal

```
procedure AddScriptComponents; virtual;
```

C++

```
virtual __fastcall AddScriptComponents();
```

Description

This is AddScriptComponents, a member of class TBaseInetXPageProducer.

100.1.1.4.3 TBaseInetXPageProducer.CanAddClass Method

Indicates whether a specified class can be used as a child Web item.

Pascal

```
function CanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClass(TComponent * AParent, TClass AClass);
```

Description

The Web page editor calls CanAddClass to determine what Web items can be added to supply content for the Web item that implements this interface.

AParent is the Web item that will act as a parent to the added Web item.

AClass is the class of the proposed child Web item.

CanAddClass returns true if the Web item can use content generated by a component of type AClass, false otherwise.

100.1.1.4.4 TBaseInetXPageProducer.ContentFromStream Method

Returns the content produced in response to input read from a stream.

Pascal

```
function ContentFromStream(InStream: TStream): string; override;
```

C++

```
virtual __fastcall AnsiString ContentFromStream(TStream InStream);
```

Description

Call ContentFromStream to retrieve the content produced by the content producer based on the information provided by the Stream parameter. ContentFromStream

1. Generates an OnBeforeGetContent (see page 17130) event.

2. Translates the HTML template that is the value of Stream, using information produced by the items listed in the WebPageItems property.
3. Generates an OnAfterGetContent (see page 17129) event.
4. Returns the HTML produced in step 2.

See Also

Content (see page 15274)

OnBeforeGetContent (see page 17130)

WebPageItems (see page 18382)

OnAfterGetContent (see page 17129)

100.1.1.4.5 TBaseInetXPageProducer.CreatePageElements Method

Pascal

```
function CreatePageElements: TInetxPageElements; virtual;
```

C++

```
virtual __fastcall TInetxPageElements CreatePageElements();
```

Description

This is CreatePageElements, a member of class TBaseInetXPageProducer.

100.1.1.4.6 TBaseInetXPageProducer.CreateScriptManager Method

Pascal

```
function CreateScriptManager(Options: TWebContentOptions): TObject; virtual;
```

C++

```
virtual __fastcall TObject * CreateScriptManager(TWebContentOptions Options);
```

Description

This is CreateScriptManager, a member of class TBaseInetXPageProducer.

100.1.1.4.7 TBaseInetXPageProducer.DeclareFunctions Method

Pascal

```
function DeclareFunctions: string; virtual;
```

C++

```
virtual __fastcall AnsiString DeclareFunctions();
```

Description

This is DeclareFunctions, a member of class TBaseInetXPageProducer.

100.1.1.4.8 TBaseInetXPageProducer.DeclareIncludes Method

Pascal

```
function DeclareIncludes: string; virtual;
```

C++

```
virtual __fastcall AnsiString DeclareIncludes();
```

Description

This is DeclareIncludes, a member of class TBaseInetXPageProducer.

100.1.1.4.9 TBaseInetXPageProducer.DeclareRowSets Method

Pascal

```
function DeclareRowSets: string; virtual;
```

C++

```
virtual __fastcall AnsiString DeclareRowSets();
```

Description

This is DeclareRowSets, a member of class TBaseInetXPageProducer.

100.1.1.4.10 TBaseInetXPageProducer.DeclareScript Method

Pascal

```
function DeclareScript(XMLIslands: Boolean): string; virtual;
```

C++

```
virtual __fastcall AnsiString DeclareScript(Boolean XMLIslands);
```

Description

This is DeclareScript, a member of class TBaseInetXPageProducer.

100.1.1.4.11 TBaseInetXPageProducer.DeclareStyles Method

Pascal

```
function DeclareStyles: string; virtual;
```

C++

```
virtual __fastcall AnsiString DeclareStyles();
```

Description

This is DeclareStyles, a member of class TBaseInetXPageProducer.

100.1.1.4.12 TBaseInetXPageProducer.DeclareWarnings Method

Pascal

```
function DeclareWarnings(Warnings: TStrings): string; virtual;
```

C++

```
virtual __fastcall AnsiString DeclareWarnings(TStrings Warnings);
```

Description

This is DeclareWarnings, a member of class TBaseInetXPageProducer.

100.1.1.4.13 TBaseInetXPageProducer.DeclareXMLDocuments Method

Pascal

```
function DeclareXMLDocuments: string; virtual;
```

C++

```
virtual __fastcall AnsiString DeclareXMLDocuments();
```

Description

This is DeclareXMLDocuments, a member of class TBaseInetXPageProducer.

100.1.1.4.14 TBaseInetXPageProducer.DeclareXMLIslands Method

Pascal

```
function DeclareXMLIslands: string; virtual;
```

C++

```
virtual __fastcall AnsiString DeclareXMLIslands();
```

Description

This is DeclareXMLIslands, a member of class TBaseInetXPageProducer.

100.1.1.4.15 TBaseInetXPageProducer.DoAfterGetContent Method

Pascal

```
procedure DoAfterGetContent;
```

C++

```
__fastcall DoAfterGetContent();
```

Description

This is DoAfterGetContent, a member of class TBaseInetXPageProducer.

100.1.1.4.16 TBaseInetXPageProducer.DoAfterGetXMLData Method

Pascal

```
procedure DoAfterGetXMLData(XMLBroker: TXMLBroker; var OwnerData: OleVariant);
```

C++

```
__fastcall DoAfterGetXMLData(TXMLBroker XMLBroker, OleVariant OwnerData);
```

Description

This is DoAfterGetXMLData, a member of class TBaseInetXPageProducer.

100.1.1.4.17 TBaseInetXPageProducer.DoBeforeGetContent Method

Pascal

```
procedure DoBeforeGetContent;
```

C++

```
__fastcall DoBeforeGetContent();
```

Description

This is DoBeforeGetContent, a member of class TBaseInetXPageProducer.

100.1.1.4.18 TBaseInetXPageProducer.DoBeforeGetXMLData Method

Pascal

```
procedure DoBeforeGetXMLData(XMLBroker: TXMLBroker; var OwnerData: OleVariant);
```

C++

```
__fastcall DoBeforeGetXMLData(TXMLBroker XMLBroker, OleVariant OwnerData);
```

Description

This is DoBeforeGetXMLData, a member of class TBaseInetXPageProducer.

100.1.1.4.19 TBaseInetXPageProducer.DoTagEvent Method

Pascal

```
procedure DoTagEvent(Tag: TTag; const TagString: string; TagParams: TStrings; var ReplaceText:
string); override;
```

C++

```
virtual __fastcall DoTagEvent(TTag Tag, const AnsiString TagString, TStrings TagParams,
AnsiString ReplaceText);
```

Description

This is DoTagEvent, a member of class TBaseInetXPageProducer.

100.1.1.4.20 TBaseInetXPageProducer.FindXMLBroker Method

Returns the XML broker that supplies the data packet specified by a DATAPACKET tag.

Pascal

```
function FindXMLBroker(TagParams: TStrings): TXMLBroker; virtual;
```

C++

```
virtual __fastcall TXMLBroker FindXMLBroker(TStrings TagParams);
```

Description

The page producer translates HTML-transparent tags of the form

```
<#DATAPACKET XMLBroker=BrokerName>
```

into the XML returned by the XML broker. To do so, it calls FindXMLBroker to locate the XML broker specified by the XMLBroker parameter.

TagParams is a set of Name=Value pairs. This comes from the parameter portion of the DATAPACKET tag.

FindXMLBroker returns the XML broker with the name specified by the XMLBroker parameter. If TagParams does not contain a string of the form XMLBroker=BrokerName, FindXMLBroker returns nil (Delphi) or NULL (C++).

See Also

FindXMLBrokerName (see page 17122)

GetXMLData (see page 17124)

100.1.1.4.21 TBaseInetXPageProducer.FindXMLBrokerName Method

Returns the name of an XML broker given the parameters in a DATAPACKET tag..

Pascal

```
function FindXMLBrokerName(TagParams: TStrings): string; virtual;
```

C++

```
virtual __fastcall AnsiString FindXMLBrokerName(TStrings TagParams);
```

Description

The page producer translates HTML-transparent tags of the form

```
<#DATAPACKET XMLBroker=BrokerName>
```

into the XML returned by the XML broker. To do so, it calls FindXMLBrokerName to parse the parameters for the value of the XMLBroker parameter.

TagParams is a set of Name=Value pairs. This comes from the parameter portion of the DATAPACKET tag.

FindXMLBrokerName returns the value string from the XMLBroker parameter. If TagParams does not contain a string of the form XMLBroker=BrokerName, FindXMLBrokerName returns an empty string.

See Also

FindXMLBroker (see page 17121)

GetXMLData (see page 17124)

100.1.1.4.22 TBaseInetXPageProducer.GeneratePageElements Method

Pascal

```
procedure GeneratePageElements; virtual;
```

C++

```
virtual __fastcall GeneratePageElements();
```

Description

This is GeneratePageElements, a member of class TBaseInetXPageProducer.

100.1.1.4.23 TBaseInetXPageProducer.GetContentOptions Method

Pascal

```
function GetContentOptions(var Owned: Boolean): TWebContentOptions; override;
```

C++

```
virtual __fastcall TWebContentOptions GetContentOptions(Boolean Owned);
```

Description

This is GetContentOptions, a member of class TBaseInetXPageProducer.

100.1.1.4.24 TBaseInetXPageProducer.GetDefaultTemplate Method

Pascal

```
function GetDefaultTemplate: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetDefaultTemplate();
```

Description

This is GetDefaultTemplate, a member of class TBaseInetXPageProducer.

100.1.1.4.25 TBaseInetXPageProducer.GetRequiredTags Method

Returns the default set of HTML-transparent tags for an InternetExpress page producer.

Pascal

```
class function GetRequiredTags: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetRequiredTags();
```

Description

GetRequiredTags dynamically generates the InternetExpress-specific tags that form the basis of an InternetExpress page producer's default template.

GetRequiredTags returns the string

```
<#INCLUDES> <#STYLES> <#WARNINGS> <#FORMS> <#SCRIPT>
```

See Also

HTMLDoc (see page 17140)

Content (see page 17138)

100.1.1.4.26 TBaseInetXPageProducer.GetScriptManager Method

Pascal

```
function GetScriptManager: IScriptManager;
```

C++

```
__fastcall IScriptManager GetScriptManager();
```

Description

This is GetScriptManager, a member of class TBaseInetXPageProducer.

100.1.1.4.27 TBaseInetXPageProducer.GetUseXMLIslands Method

Pascal

```
function GetUseXMLIslands: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetUseXMLIslands();
```

Description

This is GetUseXMLIslands, a member of class TBaseInetXPageProducer.

100.1.1.4.28 TBaseInetXPageProducer.GetXMLData Method

Obtains an XML data packet from a specified XML broker.

Pascal

```
function GetXMLData(XMLBroker: TXMLBroker; XMLOptions: TXMLOptions): string; virtual;
```

C++

```
virtual __fastcall AnsiString GetXMLData(TXMLBroker XMLBroker, TXMLOptions XMLOptions);
```

Description

GetXMLData first generates an OnBeforeGetXMLData (see page 17130) event. It then calls the RequestRecords method of the XML broker specified by XMLBroker, passing in the formatting options specified by XMLOptions. Finally, it generates an OnAfterGetXMLData (see page 17129) event.

GetXMLData returns the XML data packet that XMLBroker supplies.

See Also

OnAfterGetXMLData (see page 17129)

OnBeforeGetXMLData (see page 17130)

Content (see page 15274)

RequestRecords (see page 26064)

100.1.1.4.29 TBaseInetXPageProducer.GetXMLDelta Method

Pascal

```
function GetXMLDelta(XMLBroker: TXMLBroker): string;
```

C++

```
__fastcall AnsiString GetXMLDelta(TXMLBroker XMLBroker);
```

Description

This is GetXMLDelta, a member of class TBaseInetXPageProducer.

100.1.1.4.30 TBaseInetXPageProducer.GetXMLErrors Method

Pascal

```
function GetXMLErrors(XMLBroker: TXMLBroker): string;
```

C++

```
__fastcall AnsiString GetXMLErrors(TXMLBroker XMLBroker);
```

Description

This is GetXMLErrors, a member of class TBaseInetXPageProducer.

100.1.1.4.31 TBaseInetXPageProducer.ImplCanAddClass Method

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

This is ImplCanAddClass, a member of class TBaseInetXPageProducer.

100.1.1.4.32 TBaseInetXPageProducer.SetIncludePathURL Method

Assigns a value to the IncludePathURL (see page 17127) property.

Pascal

```
procedure SetIncludePathURL(const Value: string);
```

C++

```
__fastcall SetIncludePathURL(const AnsiString Value);
```

Description

SetIncludePathURL is the write implementation of the IncludePathURL (see page 17127) property.

See Also

IncludePathURL (see page 17127)

100.1.1.4.33 TBaseInetXPageProducer.SetStyles Method

Assigns a value to the Styles (see page 17127) property.

Pascal

```
procedure SetStyles(Value: TStrings);
```

C++

```
__fastcall SetStyles(TStrings Value);
```

Description

SetStyles is the write implementation of the Styles (see page 17127) property.

See Also

Styles (see page 17127)

100.1.1.4.34 TBaseInetXPageProducer.SetStylesFile Method

Assigns a value to the StylesFile (see page 17128) property.

Pascal

```
procedure SetStylesFile(const Value: TFileName);
```

C++

```
__fastcall SetStylesFile(const TFileName Value);
```

Description

SetStylesFile is the write implementation of the StylesFile (see page 17128) property.

See Also

StylesFile (see page 17128)

100.1.1.4.35 TBaseInetXPageProducer.SetWebContentOptions Method

Pascal

```
procedure SetWebContentOptions(AOptions: TWebContentOptions);
```

C++

```
__fastcall SetWebContentOptions(TWebContentOptions AOptions);
```

Description

This is SetWebContentOptions, a member of class TBaseInetXPageProducer.

100.1.1.5 TBaseInetXPageProducer Properties

The properties of the TBaseInetXPageProducer class are listed here.

Protected Properties

	Name	Description
	PageElements (see page 17127)	This is PageElements, a member of class TBaseInetXPageProducer.
	ScriptManager (see page 17127)	This is ScriptManager, a member of class TBaseInetXPageProducer.

Public Properties

	Name	Description
	EnableXMLIslands (see page 17126)	Specifies whether XML data packets should be declared using XML islands if the requesting browser supports them.
	IncludePathURL (see page 17127)	Indicates the URL where the javascript libraries are installed.
	Styles (see page 17127)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 17128)	Specifies a file containing the individual styles for the page producer's style sheet.

100.1.1.5.1 TBaseInetXPageProducer.EnableXMLIslands Property

Specifies whether XML data packets should be declared using XML islands if the requesting browser supports them.

Pascal

```
property EnableXMLIslands: Boolean;
```

C++

```
__property Boolean EnableXMLIslands;
```

Description

Set EnableXMLIslands to indicate whether the XML in the generated HTML document should be declared using XML islands. XML islands are only used if the document is created in response to a request message whose headers indicate that the browser supports XML islands.

Set EnableXMLIslands to false to prevent the page producer from declaring XML islands even if the requesting browser supports them.

See Also

GetXMLData (see page 17124)

Content (see page 15274)

100.1.1.5.2 TBaseInetXPageProducer.IncludePathURL Property

Indicates the URL where the javascript libraries are installed.

Pascal

```
property IncludePathURL: string;
```

C++

```
__property AnsiString IncludePathURL;
```

Description

Set IncludePathURL to indicate where the javascript libraries are installed. The page producer uses this value to create HTML statements that include the libraries in the generated HTML document. These statements have the form

```
<SCRIPT language=Javascript type="text/javascript" SRC="IncludePathURL/xmldb.js"> </SCRIPT>
```

The javascript libraries define classes and methods that allow the HTML document to interpret XML data packets if the Web browser is not XML-enabled, and that enable controls in the document to navigate the data, post changes, and apply updates.

See Also

SetIncludePathURL (see page 17125)

Content (see page 15274)

Creating Web Pages with an InternetExpress Page Producer

100.1.1.5.3 TBaseInetXPageProducer.PageElements Property

Pascal

```
property PageElements: TInetxPageElements;
```

C++

```
__property TInetxPageElements PageElements;
```

Description

This is PageElements, a member of class TBaseInetXPageProducer.

100.1.1.5.4 TBaseInetXPageProducer.ScriptManager Property

Pascal

```
property ScriptManager: IScriptManager;
```

C++

```
__property IScriptManager ScriptManager;
```

Description

This is ScriptManager, a member of class TBaseInetXPageProducer.

100.1.1.5.5 TBaseInetXPageProducer.Styles Property

Defines the styles that can be used by the page producer and its Web items.

Pascal

```
property Styles: TStrings;
```

C++

```
__property TStrings Styles;
```

Description

Use Styles to define a style sheet for the HTML document produced by TBaseInetXPageProducer (see page 17109). Each string in the TStrings object specifies a specific style by giving a style selector followed by a set of attributes in curly braces. These definitions can define styles for standard HTML elements such as

```
H2 B  {color: red}
```

or they can define styles that you name, such as

```
.MyStyle  {font-family: arial; font-weight: bold; font-size: 18px }
```

Note that user-defined style names must begin with a dot.

Styles for standard elements are applied automatically to HTML that uses the tag or tags that form the style selector. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector.

Note: If you are sharing a set of styles between multiple page producers or multiple applications, you can define the styles in a file. To define the styles in a file, use the StylesFile (see page 17128) property instead. If StylesFile (see page 17128) is defined, it takes precedence over the Styles property.

See Also

SetStyles (see page 17125)

StylesFile (see page 17128)

Content (see page 15274)

Customizing the InternetExpress Page Producer Template

Setting Web Item Properties

100.1.1.5.6 TBaseInetXPageProducer.StylesFile Property

Specifies a file containing the individual styles for the page producer's style sheet.

Pascal

```
property StylesFile: TFileName;
```

C++

```
__property TFileName StylesFile;
```

Description

Set StylesFile to the name of a file containing the style definitions used by the page producer. The page producer uses these style definitions to declare a style sheet in the HTML document it generates.

Styles (see page 17127) for standard elements are applied automatically to HTML in the document. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector of a style definition.

Note: If the StylesFile property is not set, you can specify the style definitions using the Styles (see page 17127) property instead.

See Also

SetStylesFile (see page 17125)

Styles (see page 17127)

Content (see page 15274)

Customizing the InternetExpress Page Producer Template

Setting Web Item Properties

100.1.1.6 TBaseInetXPageProducer Events

The events of the TBaseInetXPageProducer class are listed here.

Public Events

	Name	Description
	OnAfterGetContent (see page 17129)	Occurs after the page producer generates its HTML document.
	OnAfterGetXMLData (see page 17129)	Occurs after the page producer translates fetches an XML data packet from an XML broker.
	OnBeforeGetContent (see page 17130)	Occurs when the Content method is called.
	OnBeforeGetXMLData (see page 17130)	Occurs before the page producer fetches an XML data packet from an XML broker.

100.1.1.6.1 TBaseInetXPageProducer.OnAfterGetContent Event

Occurs after the page producer generates its HTML document.

Pascal

```
property OnAfterGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAfterGetContent;
```

Description

Write an OnAfterGetContent event handler to take specific action immediately after the page producer generates an HTML document. This event occurs immediately before the Content method returns the generated document. Typically it is used in applications where the page producer automatically responds to messages as the Provider (see page 18419) property of an action item or the Response producer property of an XML broker.

See Also

OnBeforeGetContent (see page 17130)

Content (see page 15274)

Producer (see page 15138)

100.1.1.6.2 TBaseInetXPageProducer.OnAfterGetXMLData Event

Occurs after the page producer translates fetches an XML data packet from an XML broker.

Pascal

```
property OnAfterGetXMLData: TXMLDataEvent;
```

C++

```
__property TXMLDataEvent OnAfterGetXMLData;
```

Description

Write an OnAfterGetXMLData event handler to take specific action immediately after the page producer obtains an XML data

packet from an XML broker. Typically, the OnAfterGetXMLData event handler responds to any values returned by the provider on the application server as the OwnerData parameter.

When the page producer generates its HTML document (that is, when the Content method executes), it adds code to define the XML data packets manipulated in the document. To fetch these XML data packets, the page producer calls an XML broker's RequestRecords method. OnAfterGetXMLData occurs after the page producer has returned an XML data packet but before it is assigned as part of the emerging content.

See Also

OnBeforeGetXMLData (see page 17130)

GetXMLData (see page 17124)

OnHTMLTag (see page 15280)

Content (see page 15274)

100.1.1.6.3 TBaseInetXPageProducer.OnBeforeGetContent Event

Occurs when the Content method is called.

Pascal

```
property OnBeforeGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnBeforeGetContent;
```

Description

Write an OnBeforeGetContent event handler to take specific action immediately before the page producer generates an HTML document. This event occurs when the Content method is called before the page producer starts assembling its HTML document. Typically it is used in applications where the page producer automatically responds to messages as the Provider (see page 18419) property of an action item or the Response producer property of an XML broker.

See Also

OnAfterGetContent (see page 17129)

Content (see page 15274)

Producer (see page 15138)

100.1.1.6.4 TBaseInetXPageProducer.OnBeforeGetXMLData Event

Occurs before the page producer fetches an XML data packet from an XML broker.

Pascal

```
property OnBeforeGetXMLData: TXMLDataEvent;
```

C++

```
__property TXMLDataEvent OnBeforeGetXMLData;
```

Description

Write an OnBeforeGetXMLData event handler to take specific action immediately before the page producer requests an XML data packet from an XML broker. Typically, the OnBeforeGetXMLData event handler encodes any values for the provider on the application server as the OwnerData parameter or initializes properties on the XML broker that affect the data packet.

When the page producer generates its HTML document (that is, when the Content method executes), it adds code to define the

XML data packets manipulated in the document. To fetch these XML data packets, the page producer calls an XML broker's RequestRecords method. OnBeforeGetXMLData occurs immediately before the page producer requests an XML data packet. Any value assigned to the OwnerData parameter is passed to the XML broker's OnRequestRecords event handler, and from there to the provider's OnBeforeGetRecords event handler.

See Also

OnAfterGetXMLData (see page 17129)

OnRequestRecords (see page 26080)

GetXMLData (see page 17124)

Content (see page 15274)

OnHTMLTag (see page 15280)

MaxRecords (see page 26072)

Params (see page 26073)

100.1.2 TCustomInetXPageProducer Class

TCustomInetXPageProducer is the base class for page producers that display live database information in an InternetExpress application.

Class Hierarchy

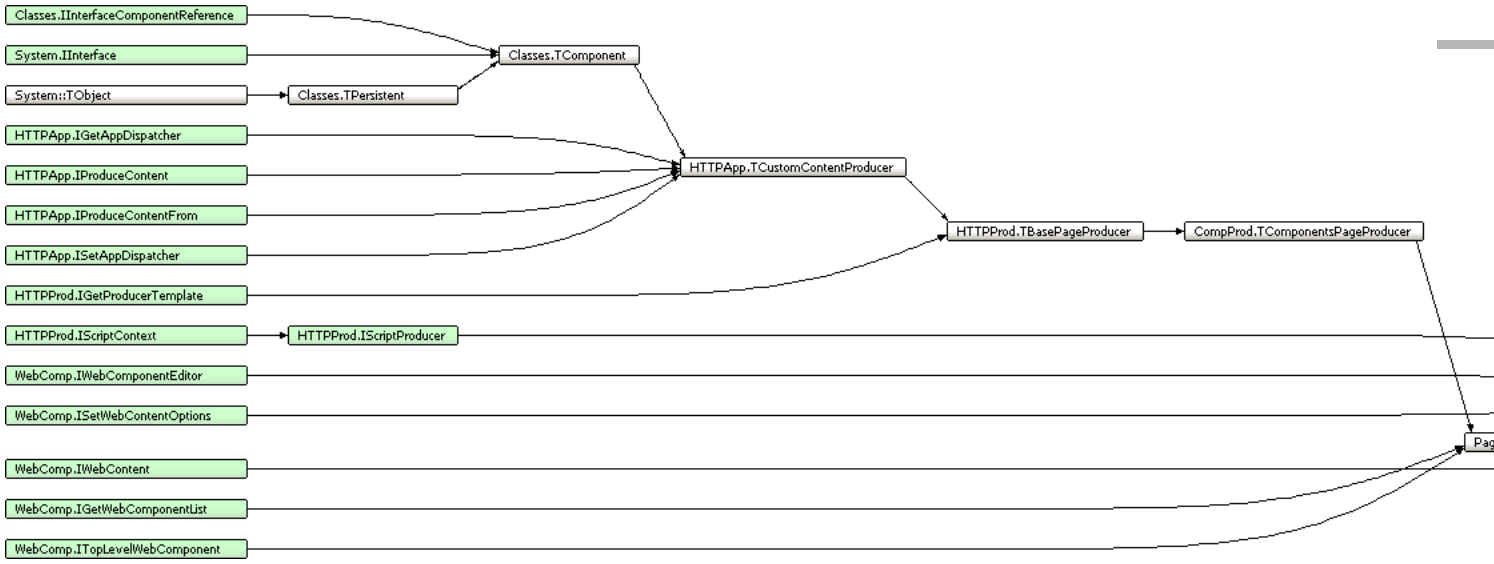

Pascal

```
TCustomInetXPageProducer = class(TBaseInetXPageProducer, IWebContent);
```

C++

```
class TCustomInetXPageProducer : public TBaseInetXPageProducer, public IWebContent;
```

File

MidProd

Description

Use TCustomInetXPageProducer as a base class for components that create Web pages in an InternetExpress Web application. These pages display database information from an application server and include data packets encoded in XML and javascript that lets Web browsers navigate that data, post changes, and apply updates back to the application server.

The InetX page producer creates its HTML document from a template. The InetX page producer has a default template that coordinates several top-level declarations for the HTML document with content produced by other components. The other components that generate HTML content for the InetX page producer can be added at design time using the Web Page editor.

Do not instantiate TCustomInetXPageProducer. To create a producer that generates HTML pages with javascript database functions and XML data packets, use a descendant class such as TInetXPageProducer (see page 17145).

See Also

TXMLBroker (see page 26049)

100.1.2.1 TCustomInetXPageProducer Members

The following tables list the members exposed by TCustomInetXPageProducer.

Protected Events

	Name	Description
	OnHTMLTag (see page 15280)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddIncludes (see page 17116)	This is AddIncludes, a member of class TBaseInetXPageProducer.
V	AddScriptComponents (see page 17117)	This is AddScriptComponents, a member of class TBaseInetXPageProducer.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 17117)	Indicates whether a specified class can be used as a child Web item.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CreatePageElements (see page 17118)	This is CreatePageElements, a member of class TBaseInetXPageProducer.
V	CreateScriptManager (see page 17118)	This is CreateScriptManager, a member of class TBaseInetXPageProducer.
V	DeclareFunctions (see page 17118)	This is DeclareFunctions, a member of class TBaseInetXPageProducer.
V	DeclareIncludes (see page 17118)	This is DeclareIncludes, a member of class TBaseInetXPageProducer.
V	DeclareRowSets (see page 17119)	This is DeclareRowSets, a member of class TBaseInetXPageProducer.
V	DeclareScript (see page 17119)	This is DeclareScript, a member of class TBaseInetXPageProducer.
V	DeclareStyles (see page 17119)	This is DeclareStyles, a member of class TBaseInetXPageProducer.
V	DeclareWarnings (see page 17119)	This is DeclareWarnings, a member of class TBaseInetXPageProducer.
V	DeclareXMLDocuments (see page 17120)	This is DeclareXMLDocuments, a member of class TBaseInetXPageProducer.
V	DeclareXMLIslands (see page 17120)	This is DeclareXMLIslands, a member of class TBaseInetXPageProducer.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoAfterGetContent (see page 17120)	This is DoAfterGetContent, a member of class TBaseInetXPageProducer.
	DoAfterGetXMLData (see page 17120)	This is DoAfterGetXMLData, a member of class TBaseInetXPageProducer.

	DoBeforeGetContent (see page 17120)	This is DoBeforeGetContent, a member of class TBaseInetXPageProducer.
	DoBeforeGetXMLData (see page 17121)	This is DoBeforeGetXMLData, a member of class TBaseInetXPageProducer.
V	DoTagEvent (see page 5618)	This is DoTagEvent, a member of class TComponentsPageProducer.
	DoTagEvent (see page 15276)	This is DoTagEvent, a member of class TBasePageProducer.
V	DoTagEvent (see page 17121)	This is DoTagEvent, a member of class TBaseInetXPageProducer.
V	FindComponent (see page 5619)	This is FindComponent, a member of class TComponentsPageProducer.
	FindComponentName (see page 5619)	This is FindComponentName, a member of class TComponentsPageProducer.
V	GeneratePageElements (see page 17122)	This is GeneratePageElements, a member of class TBaseInetXPageProducer.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 18381)	This is GetChildren, a member of class TBasePageItemsProducer.
V	GetComponentContent (see page 5619)	This is GetComponentContent, a member of class TComponentsPageProducer.
	GetComponentList (see page 18382)	This is GetComponentList, a member of class TBasePageItemsProducer.
V	GetContentOptions (see page 5619)	This is GetContentOptions, a member of class TComponentsPageProducer.
V	GetContentOptions (see page 17122)	This is GetContentOptions, a member of class TBaseInetXPageProducer.
	GetDefaultComponentList (see page 18382)	This is GetDefaultComponentList, a member of class TBasePageItemsProducer.
V	GetDefaultTemplate (see page 17122)	This is GetDefaultTemplate, a member of class TBaseInetXPageProducer.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetProducerTemplateFileName (see page 15276)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.
	GetProducerTemplateStream (see page 15276)	This is GetProducerTemplateStream, a member of class TBasePageProducer.
V	GetScriptEngine (see page 15277)	This is GetScriptEngine, a member of class TBasePageProducer.
	GetScriptManager (see page 17123)	This is GetScriptManager, a member of class TBaseInetXPageProducer.
	GetTagID (see page 15277)	This is GetTagID, a member of class TBasePageProducer.
V	GetTemplateFileName (see page 17139)	This is GetTemplateFileName, a member of class TCustomInetXPageProducer.
V	GetTemplateStream (see page 17139)	This is GetTemplateStream, a member of class TCustomInetXPageProducer.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	GetUseXMLIslands (see page 17123)	This is GetUseXMLIslands, a member of class TBaseInetXPageProducer.
	GetXMLDelta (see page 17124)	This is GetXMLDelta, a member of class TBaseInetXPageProducer.
	GetXMLErrors (see page 17124)	This is GetXMLErrors, a member of class TBaseInetXPageProducer.
V	HandleScriptTag (see page 15277)	This is HandleScriptTag, a member of class TBasePageProducer.
V	HandleTag (see page 5620)	This is HandleTag, a member of class TComponentsPageProducer.
V	HandleTag (see page 15278)	This is HandleTag, a member of class TBasePageProducer.
V	ImplCanAddClass (see page 17124)	This is ImplCanAddClass, a member of class TBaseInetXPageProducer.
	ImplHandleTag (see page 15278)	This is ImplHandleTag, a member of class TBasePageProducer.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 15108)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ServerScriptFromStream (see page 15278)	This is ServerScriptFromStream, a member of class TBasePageProducer.

	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 18382)	This is SetChildOrder, a member of class TBasePageItemsProducer.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetWebContentOptions (see page 17126)	This is SetWebContentOptions, a member of class TBaseInetXPageProducer.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
V	UseScriptEngine (see page 15278)	This is UseScriptEngine, a member of class TBasePageProducer.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebContent (see page 17140)	This is WebContent, a member of class TCustomInetXPageProducer.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	PageElements (see page 17127)	This is PageElements, a member of class TBaseInetXPageProducer.
R	ScriptManager (see page 17127)	This is ScriptManager, a member of class TBaseInetXPageProducer.

Public Events

	Name	Description
	OnAfterGetContent (see page 17129)	Occurs after the page producer generates its HTML document.
	OnAfterGetXMLData (see page 17129)	Occurs after the page producer translates fetches an XML data packet from an XML broker.
	OnBeforeGetContent (see page 17130)	Occurs when the Content method is called.
	OnBeforeGetXMLData (see page 17130)	Occurs before the page producer fetches an XML data packet from an XML broker.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Content (see page 17138)	Returns the HTML document assembled by the InternetExpress page producer.

	ContentFromStream (see page 15107)	Returns the content produced in response to input read from a stream.
	ContentFromStream (see page 15275)	Returns the content produced in response to input read from a stream.
	ContentFromStream (see page 17117)	Returns the content produced in response to input read from a stream.
	ContentFromString (see page 15107)	Returns the content produced in response to an input string.
	ContentFromString (see page 15275)	Returns the content produced in response to an input string.
	Create (see page 17137)	Creates and initializes an instance of TCustomInetXPageProducer (see page 17131).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 17137)	Frees an instance of TCustomInetXPageProducer (see page 17131).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Evaluate (see page 15324)	This is Evaluate, a member of class IScriptProducer.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindXMLBroker (see page 17121)	Returns the XML broker that supplies the data packet specified by a DATAPACKET tag.
	FindXMLBrokerName (see page 17122)	Returns the name of an XML broker given the parameters in a DATAPACKET tag..
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContent (see page 15324)	This is GetContent, a member of class IScriptProducer.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetErrors (see page 15324)	This is GetErrors, a member of class IScriptProducer.
	GetHTMLBlock (see page 15325)	This is GetHTMLBlock, a member of class IScriptProducer.
	GetHTMLBlockCount (see page 15325)	This is GetHTMLBlockCount, a member of class IScriptProducer.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProducerTemplateFileName (see page 15320)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.
	GetProducerTemplateStream (see page 15320)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.
	GetRequiredTags (see page 17123)	Returns the default set of HTML-transparent tags for an InternetExpress page producer.
	GetWebModuleContext (see page 15322)	This is GetWebModuleContext, a member of class IScriptContext.
	GetXMLData (see page 17124)	Obtains an XML data packet from a specified XML broker.
	HandleScriptError (see page 15325)	This is HandleScriptError, a member of class IScriptProducer.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParseStream (see page 15325)	This is ParseStream, a member of class IScriptProducer.
	ParseString (see page 15325)	This is ParseString, a member of class IScriptProducer.
	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	ReplaceTags (see page 15326)	This is ReplaceTags, a member of class IScriptProducer.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetContent (see page 15326)	This is SetContent, a member of class IScriptProducer.
	SetIncludePathURL (see page 17125)	Assigns a value to the IncludePathURL (see page 17127) property.
	SetStyles (see page 17125)	Assigns a value to the Styles (see page 17127) property.
	SetStylesFile (see page 17125)	Assigns a value to the StylesFile (see page 17128) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetWebContentOptions (see page 24782)	This is SetWebContentOptions, a member of class ISetWebContentOptions.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Write (see page 15326)	This is the overview for the Write method overload.
	WriteItem (see page 15327)	This is WriteItem, a member of class IScriptProducer.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.
	EnableXMLIslands (see page 17126)	Specifies whether XML data packets should be declared using XML islands if the requesting browser supports them.
R	Errors (see page 15327)	This is Errors, a member of class IScriptProducer.
R	HTMLBlockCount (see page 15328)	This is HTMLBlockCount, a member of class IScriptProducer.
R	HTMLBlocks (see page 15328)	This is HTMLBlocks, a member of class IScriptProducer.
	HTMLDoc (see page 17140)	Specifies the HTML template used to generate the value returned by the Content (see page 17138) method.
	HTMLFile (see page 17141)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content (see page 17138) method.
	IncludePathURL (see page 17127)	Indicates the URL where the javascript libraries are installed.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.

	ScriptEngine (see page 15279)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 15279)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.
	Styles (see page 17127)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 17128)	Specifies a file containing the individual styles for the page producer's style sheet.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebModuleContext (see page 15322)	This is WebModuleContext, a member of class IScriptContext.
R	WebModuleContext (see page 15279)	Provides access to the Web module context object.
R	WebPageItems (see page 18382)	Maintains a list of components that generate HTML content for the producer.

Published Events

	Name	Description
	OnHTMLTag (see page 5620)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

100.1.2.2 TCustomInetXPageProducer.Create Constructor

Creates and initializes an instance of TCustomInetXPageProducer (see page 17131).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomInetXPageProducer(TComponent * AOwner);
```

Description

Call Create to instantiate an InternetExpress page producer at runtime. InternetExpress page producers placed in a Web module at design time are instantiated automatically.

AOwner is a component, typically a Web module, that becomes the value of the Owner property. The Owner is responsible for freeing the TCustomInetXPageProducer (see page 17131) instance.

See Also

Owner (see page 2807)

100.1.2.3 TCustomInetXPageProducer.Destroy Destructor

Frees an instance of TCustomInetXPageProducer (see page 17131).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomInetXPageProducer();
```

Description

Do not call Destroy directly. Instead, call the Free method, which checks whether the reference to the producer is not nil before calling Destroy.

See Also

Free (see page 22300)

100.1.2.4 TCustomInetXPageProducer Methods

The methods of the TCustomInetXPageProducer class are listed here.

Protected Methods

	Name	Description
	GetTemplateFileName (see page 17139)	This is GetTemplateFileName, a member of class TCustomInetXPageProducer.
	GetTemplateStream (see page 17139)	This is GetTemplateStream, a member of class TCustomInetXPageProducer.
	WebContent (see page 17140)	This is WebContent, a member of class TCustomInetXPageProducer.

Public Methods

	Name	Description
	Content (see page 17138)	Returns the HTML document assembled by the InternetExpress page producer.

100.1.2.4.1 TCustomInetXPageProducer.Content Method

Returns the HTML document assembled by the InternetExpress page producer.

Pascal

```
function Content: string; override;
```

C++

```
virtual __fastcall AnsiString Content();
```

Description

Call Content to get the HTML document produced by TCustomInetXPageProducer (see page 17131). This document contains a mixture of HTML, XML, and javascript. It allows Web browsers to view and edit data from an application server.

The InternetExpress page producer creates its content by replacing HTML-transparent tags in the template specified by HTMLDoc (see page 17140) or HTMLFile (see page 17141).

By default, this template has the form

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#INCLUDES> <#STYLES> <#WARNINGS> <#FORMS> <#SCRIPT>
</BODY>
</HTML>
```

Although the template can be changed to include any combination of these tags with tags of the following forms

```
<#COMPONENT Name=WebItemName>
<#DATAPACKET XMLBroker=BrokerName>
```

which are also translated automatically. The template can replace the default HTML-transparent tags with the tag

```
<#BODYELEMENTS>
```

Finally, the template can include any tags translated by the OnHTMLTag event handler.

The Content method translates the tags listed above as follows:

<#INCLUDES> generates the statements that include the javascript libraries at the location specified by IncludePathURL (see

page 17127).

<#STYLES> generates the statements that defines a style sheet from the definitions listed in the Styles (see page 17127) or StylesFile (see page 17128) property.

<#WARNINGS> generates nothing at runtime. At design time, it adds warning messages for problems detected while generating the HTML document. You can see these messages in the Web page editor.

<#FORMS> generates the HTML produced by the components listed in the WebPageItems property. The HTML from each component is listed in the order it appears in WebPageItems.

<#SCRIPT> generates a block of javascript declarations that are used by the components listed in WebPageItems.

<#BODYELEMENTS> generates the same HTML as results from the 5 tags in the default template. It is useful when generating a template in an HTML editor when you want to use the default layout but add additional elements (such as style sheets) using the editor.

<#COMPONENT> is replaced with the content produced by the specified component. Typically, this is a component listed in the WebPageItems property, but it can be any component that supports the IWebContent interface and has the same Owner as the INETX page producer.

<#DATAPACKET> is replaced with the XML data packet obtained from the specified XML broker.

The string returned by Content can be used as the Content property of the response to an HTTP request message.

See Also

IWebContent (see page 24792)

IncludePathURL (see page 17127)

Styles (see page 17127)

StylesFile (see page 17128)

OnHTMLTag (see page 15280)

HTMLDoc (see page 17140)

HTMLFile (see page 17141)

WebPageItems (see page 18382)

Customizing the InternetExpress Page Producer Template

100.1.2.4.2 TCustomInetXPageProducer.GetTemplateFileName Method

Pascal

```
function GetTemplateFileName: string; override;
```

C++

```
virtual __fastcall AnsiString GetTemplateFileName();
```

Description

This is GetTemplateFileName, a member of class TCustomInetXPageProducer.

100.1.2.4.3 TCustomInetXPageProducer.GetTemplateStream Method

Pascal

```
function GetTemplateStream(out AOwned: Boolean): TStream; override;
```

C++

```
virtual __fastcall TStream GetTemplateStream(Boolean AOwned);
```

Description

This is GetTemplateStream, a member of class TCustomInetXPageProducer.

100.1.2.4.4 TCustomInetXPageProducer.WebContent Method

Pascal

```
function WebContent(Options: TWebContentOptions; Layout: TLayout): string;
```

C++

```
__fastcall AnsiString WebContent(TWebContentOptions Options, TLayout Layout);
```

Description

This is WebContent, a member of class TCustomInetXPageProducer.

100.1.2.5 TCustomInetXPageProducer Properties

The properties of the TCustomInetXPageProducer class are listed here.

Public Properties

	Name	Description
	HTMLDoc (see page 17140)	Specifies the HTML template used to generate the value returned by the Content (see page 17138) method.
	HTMLFile (see page 17141)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content (see page 17138) method.

100.1.2.5.1 TCustomInetXPageProducer.HTMLDoc Property

Specifies the HTML template used to generate the value returned by the Content (see page 17138) method.

Pascal

```
property HTMLDoc: TStrings;
```

C++

```
__property TStrings HTMLDoc;
```

Description

Set HTMLDoc to a set of strings that make up the HTML template. By default, HTMLDoc is initialized to the following template:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#INCLUDES> <#STYLES> <#WARNINGS> <#FORMS> <#SCRIPT>
</BODY>
</HTML>
```

Each string is a sequence of one or more HTML commands or HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. Following the tag name, the HTML-transparent tag can optionally include

parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. TCustomInetXPageProducer (see page 17131) automatically handles the conversion of the HTML-transparent tags in the default template, as well as the tags described in the help for the Content (see page 17138) method. If you set HTMLDoc to include other HTML-transparent tags, you must translate these in an OnHTMLTag event handler.

Note: Setting HTMLDoc clears the HTMLFile (see page 17141) property.

See Also

Content (see page 17138)

HTMLFile (see page 17141)

OnHTMLTag (see page 15280)

Creating Web Pages with an InternetExpress Page Producer

Customizing the InternetExpress Page Producer Template

100.1.2.5.2 TCustomInetXPageProducer.HTMLFile Property

Specifies the name of a file that contains the HTML template used to generate the value returned by the Content (see page 17138) method.

Pascal

```
property HTMLFile: TFileName;
```

C++

```
__property TFileName HTMLFile;
```

Description

Set HTMLFile to the name of the file that contains an HTML template. The HTML template is a sequence of HTML commands and HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLFile clears the HTMLDoc (see page 17140) property.

Note: When distributing applications that contain page producers which use the HTMLFile property, be sure to distribute the file along with the application. Also, be sure that the value of HTMLFile does not include a hard-coded path name that will not be available on the user's disk.

See Also

Content (see page 17138)

HTMLDoc (see page 17140)

Customizing the InternetExpress Page Producer Template

100.1.3 TInetXPageElements Class

Class Hierarchy

Pascal

```
TInetXPageElements = class(TObject);
```

C++

```
class TInetXPageElements : public TObject;
```

File

MidProd

Description

This is class MidProd.TInetXPageElements.

100.1.3.1 TInetXPageElements Members

The following tables list the members exposed by TInetXPageElements.

Public Fields

	Name	Description
	FormsContent (see page 17143)	This is FormsContent, a member of class TInetXPageElements.
	IncludesContent (see page 17143)	This is IncludesContent, a member of class TInetXPageElements.
	ScriptContent (see page 17144)	This is ScriptContent, a member of class TInetXPageElements.
	StylesContent (see page 17144)	This is StylesContent, a member of class TInetXPageElements.
	WarningsContent (see page 17144)	This is WarningsContent, a member of class TInetXPageElements.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BodyContent (see page 17144)	This is BodyContent, a member of class TInetXPageElements.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

100.1.3.2 TInetXPageElements Fields

The fields of the TInetXPageElements class are listed here.

Public Fields

	Name	Description
	FormsContent (see page 17143)	This is FormsContent, a member of class TInetXPageElements.
	IncludesContent (see page 17143)	This is IncludesContent, a member of class TInetXPageElements.
	ScriptContent (see page 17144)	This is ScriptContent, a member of class TInetXPageElements.
	StylesContent (see page 17144)	This is StylesContent, a member of class TInetXPageElements.
	WarningsContent (see page 17144)	This is WarningsContent, a member of class TInetXPageElements.

100.1.3.2.1 TInetXPageElements.FormsContent Field

Pascal

```
FormsContent: string;
```

C++

```
AnsiString FormsContent;
```

Description

This is FormsContent, a member of class TInetXPageElements.

100.1.3.2.2 TInetXPageElements.IncludesContent Field

Pascal

```
IncludesContent: string;
```

C++

```
AnsiString IncludesContent;
```

Description

This is IncludesContent, a member of class TInetXPageElements.

100.1.3.2.3 TInetXPageElements.ScriptContent Field

Pascal

```
ScriptContent: string;
```

C++

```
AnsiString ScriptContent;
```

Description

This is ScriptContent, a member of class TInetXPageElements.

100.1.3.2.4 TInetXPageElements.StylesContent Field

Pascal

```
StylesContent: string;
```

C++

```
AnsiString StylesContent;
```

Description

This is StylesContent, a member of class TInetXPageElements.

100.1.3.2.5 TInetXPageElements.WarningsContent Field

Pascal

```
WarningsContent: string;
```

C++

```
AnsiString WarningsContent;
```

Description

This is WarningsContent, a member of class TInetXPageElements.

100.1.3.3 TInetXPageElements Methods

The methods of the TInetXPageElements class are listed here.

Public Methods

	Name	Description
	BodyContent (see page 17144)	This is BodyContent, a member of class TInetXPageElements.

100.1.3.3.1 TInetXPageElements.BodyContent Method

Pascal

```
function BodyContent: string; virtual;
```

C++

```
virtual __fastcall AnsiString BodyContent();
```

Description

This is BodyContent, a member of class TInetXPageElements.

100.1.4 TInetXPageProducer Class

TInetXPageProducer assembles an HTML document that represents live database information from an application server.

Class Hierarchy

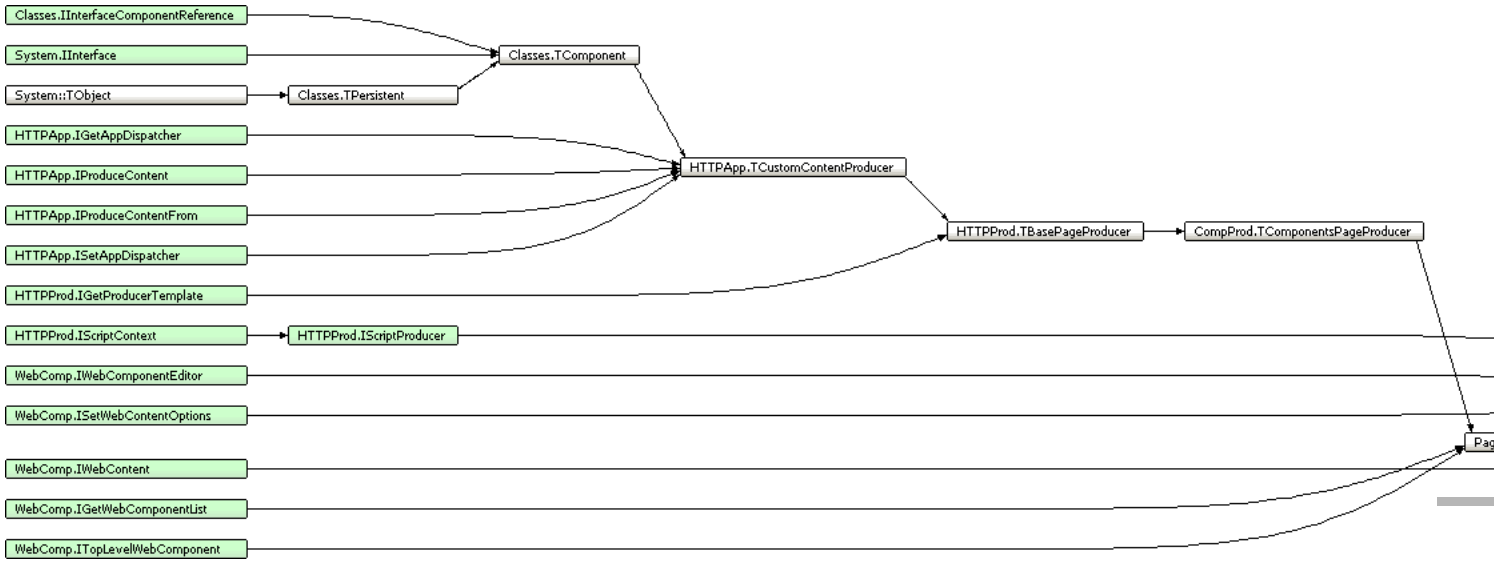

Pascal

```
TInetXPageProducer = class(TCustomInetXPageProducer);
```

C++

```
class TInetXPageProducer : public TCustomInetXPageProducer;
```

File

MidProd

Description

Use TInetXPageProducer to create Web pages for an InternetExpress application. These pages display database information from an application server and include data packets encoded in XML and javascript that lets Web browsers navigate that data, post changes, and apply updates back to the application server.

The InetX page producer is assigned as the Producer property of a Web action item or called from an OnAction event handler.

The InetX page producer creates its HTML document from a template. Unlike other page producers, the InetX page producer has a default template that coordinates several top-level declarations for the HTML document with content produced by other components. The other components that generate HTML content for the InetX page producer can be added at design time using the Web Page editor.

See Also

TWebActionItem (see page 15130)

TWebActionItem (see page 15130)

TXMLBroker (see page 26049)

The Web Module

100.1.4.1 TInetXPageProducer Members

The following tables list the members exposed by TInetXPageProducer.

Protected Events

	Name	Description
	OnHTMLTag (see page 15280)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddIncludes (see page 17116)	This is AddIncludes, a member of class TBaseInetXPageProducer.
	AddScriptComponents (see page 17117)	This is AddScriptComponents, a member of class TBaseInetXPageProducer.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 17117)	Indicates whether a specified class can be used as a child Web item.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CreatePageElements (see page 17118)	This is CreatePageElements, a member of class TBaseInetXPageProducer.
	CreateScriptManager (see page 17118)	This is CreateScriptManager, a member of class TBaseInetXPageProducer.
	DeclareFunctions (see page 17118)	This is DeclareFunctions, a member of class TBaseInetXPageProducer.
	DeclareIncludes (see page 17118)	This is DeclareIncludes, a member of class TBaseInetXPageProducer.
	DeclareRowSets (see page 17119)	This is DeclareRowSets, a member of class TBaseInetXPageProducer.
	DeclareScript (see page 17119)	This is DeclareScript, a member of class TBaseInetXPageProducer.
	DeclareStyles (see page 17119)	This is DeclareStyles, a member of class TBaseInetXPageProducer.
	DeclareWarnings (see page 17119)	This is DeclareWarnings, a member of class TBaseInetXPageProducer.
	DeclareXMLDocuments (see page 17120)	This is DeclareXMLDocuments, a member of class TBaseInetXPageProducer.
	DeclareXMLIslands (see page 17120)	This is DeclareXMLIslands, a member of class TBaseInetXPageProducer.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoAfterGetContent (see page 17120)	This is DoAfterGetContent, a member of class TBaseInetXPageProducer.
	DoAfterGetXMLData (see page 17120)	This is DoAfterGetXMLData, a member of class TBaseInetXPageProducer.
	DoBeforeGetContent (see page 17120)	This is DoBeforeGetContent, a member of class TBaseInetXPageProducer.
	DoBeforeGetXMLData (see page 17121)	This is DoBeforeGetXMLData, a member of class TBaseInetXPageProducer.
	DoTagEvent (see page 5618)	This is DoTagEvent, a member of class TComponentsPageProducer.
	DoTagEvent (see page 15276)	This is DoTagEvent, a member of class TBasePageProducer.
	DoTagEvent (see page 17121)	This is DoTagEvent, a member of class TBaseInetXPageProducer.
	FindComponent (see page 5619)	This is FindComponent, a member of class TComponentsPageProducer.
	FindComponentName (see page 5619)	This is FindComponentName, a member of class TComponentsPageProducer.
	GeneratePageElements (see page 17122)	This is GeneratePageElements, a member of class TBaseInetXPageProducer.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.

	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 18381)	This is GetChildren, a member of class TBasePageItemsProducer.
	GetComponentContent (see page 5619)	This is GetComponentContent, a member of class TComponentsPageProducer.
	GetComponentList (see page 18382)	This is GetComponentList, a member of class TBasePageItemsProducer.
	GetContentOptions (see page 5619)	This is GetContentOptions, a member of class TComponentsPageProducer.
	GetContentOptions (see page 17122)	This is GetContentOptions, a member of class TBaseInetXPageProducer.
	GetDefaultComponentList (see page 18382)	This is GetDefaultComponentList, a member of class TBasePageItemsProducer.
	GetDefaultTemplate (see page 17122)	This is GetDefaultTemplate, a member of class TBaseInetXPageProducer.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetProducerTemplateFileName (see page 15276)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.
	GetProducerTemplateStream (see page 15276)	This is GetProducerTemplateStream, a member of class TBasePageProducer.
	GetScriptEngine (see page 15277)	This is GetScriptEngine, a member of class TBasePageProducer.
	GetScriptManager (see page 17123)	This is GetScriptManager, a member of class TBaseInetXPageProducer.
	GetTagID (see page 15277)	This is GetTagID, a member of class TBasePageProducer.
	GetTemplateFileName (see page 15277)	This is GetTemplateFileName, a member of class TBasePageProducer.
	GetTemplateFileName (see page 17139)	This is GetTemplateFileName, a member of class TCustomInetXPageProducer.
	GetTemplateStream (see page 15277)	This is GetTemplateStream, a member of class TBasePageProducer.
	GetTemplateStream (see page 17139)	This is GetTemplateStream, a member of class TCustomInetXPageProducer.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetUseXMLIslands (see page 17123)	This is GetUseXMLIslands, a member of class TBaseInetXPageProducer.
	GetXMLDelta (see page 17124)	This is GetXMLDelta, a member of class TBaseInetXPageProducer.
	GetXMLErrors (see page 17124)	This is GetXMLErrors, a member of class TBaseInetXPageProducer.
	HandleScriptTag (see page 15277)	This is HandleScriptTag, a member of class TBasePageProducer.
	HandleTag (see page 5620)	This is HandleTag, a member of class TComponentsPageProducer.
	HandleTag (see page 15278)	This is HandleTag, a member of class TBasePageProducer.
	ImplCanAddClass (see page 17124)	This is ImplCanAddClass, a member of class TBaseInetXPageProducer.
	ImplHandleTag (see page 15278)	This is ImplHandleTag, a member of class TBasePageProducer.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 15108)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ServerScriptFromStream (see page 15278)	This is ServerScriptFromStream, a member of class TBasePageProducer.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 18382)	This is SetChildOrder, a member of class TBasePageItemsProducer.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.

	SetWebContentOptions (see page 17126)	This is SetWebContentOptions, a member of class TBaseInetXPageProducer.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
V	UseScriptEngine (see page 15278)	This is UseScriptEngine, a member of class TBasePageProducer.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebContent (see page 17140)	This is WebContent, a member of class TCustomInetXPageProducer.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	PageElements (see page 17127)	This is PageElements, a member of class TBaseInetXPageProducer.
R	ScriptManager (see page 17127)	This is ScriptManager, a member of class TBaseInetXPageProducer.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Content (see page 15106)	Returns the content produced by the content producer.
V	Content (see page 15274)	Returns the content produced by the content producer.
V	Content (see page 17138)	Returns the HTML document assembled by the InternetExpress page producer.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	ContentFromStream (see page 15107)	Returns the content produced in response to input read from a stream.
V	ContentFromStream (see page 15275)	Returns the content produced in response to input read from a stream.
V	ContentFromStream (see page 17117)	Returns the content produced in response to input read from a stream.
V	ContentFromString (see page 15107)	Returns the content produced in response to an input string.
V	ContentFromString (see page 15275)	Returns the content produced in response to an input string.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 15274)	Creates an instance of TBasePageProducer (see page 15269).
V	Create (see page 17115)	Creates an instance of a TBaseInetXPageProducer (see page 17109).
V	Create (see page 17137)	Creates and initializes an instance of TCustomInetXPageProducer (see page 17131).
V	Create (see page 18380)	Creates an instance of a TBasePageItemsProducer (see page 18376).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.

	Destroy (see page 17115)	Destroys the instance of the TBasePageItemsProducer.
	Destroy (see page 17137)	Frees an instance of TCustomInetXPageProducer (see page 17131).
	Destroy (see page 18381)	Destroys the instance of the TBasePageItemsProducer (see page 18376).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Evaluate (see page 15324)	This is Evaluate, a member of class IScriptProducer.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindXMLBroker (see page 17121)	Returns the XML broker that supplies the data packet specified by a DATAPACKET tag.
	FindXMLBrokerName (see page 17122)	Returns the name of an XML broker given the parameters in a DATAPACKET tag..
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContent (see page 15324)	This is GetContent, a member of class IScriptProducer.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetErrors (see page 15324)	This is GetErrors, a member of class IScriptProducer.
	GetHTMLBlock (see page 15325)	This is GetHTMLBlock, a member of class IScriptProducer.
	GetHTMLBlockCount (see page 15325)	This is GetHTMLBlockCount, a member of class IScriptProducer.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProducerTemplateFileName (see page 15320)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.
	GetProducerTemplateStream (see page 15320)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.
	GetRequiredTags (see page 17123)	Returns the default set of HTML-transparent tags for an InternetExpress page producer.
	GetWebModuleContext (see page 15322)	This is GetWebModuleContext, a member of class IScriptContext.
	GetXMLData (see page 17124)	Obtains an XML data packet from a specified XML broker.
	HandleScriptError (see page 15325)	This is HandleScriptError, a member of class IScriptProducer.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	ParseStream (see page 15325)	This is ParseStream, a member of class IScriptProducer.
	ParseString (see page 15325)	This is ParseString, a member of class IScriptProducer.
	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	ReplaceTags (see page 15326)	This is ReplaceTags, a member of class IScriptProducer.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetContent (see page 15326)	This is SetContent, a member of class IScriptProducer.
	SetIncludePathURL (see page 17125)	Assigns a value to the IncludePathURL (see page 17127) property.
	SetStyles (see page 17125)	Assigns a value to the Styles (see page 17127) property.
	SetStylesFile (see page 17125)	Assigns a value to the StylesFile (see page 17128) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetWebContentOptions (see page 24782)	This is SetWebContentOptions, a member of class ISetWebContentOptions.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	Write (see page 15326)	This is the overview for the Write method overload.
	WriteItem (see page 15327)	This is WriteItem, a member of class IScriptProducer.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Content (see page 15327)	This is Content, a member of class IScriptProducer.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.
R	Errors (see page 15327)	This is Errors, a member of class IScriptProducer.
R	HTMLBlockCount (see page 15328)	This is HTMLBlockCount, a member of class IScriptProducer.
R	HTMLBlocks (see page 15328)	This is HTMLBlocks, a member of class IScriptProducer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	StripParamQuotes (see page 15279)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebModuleContext (see page 15322)	This is WebModuleContext, a member of class IScriptContext.
R	WebModuleContext (see page 15279)	Provides access to the Web module context object.

Published Events

	Name	Description
	OnAfterGetContent (see page 17156)	Occurs after the page producer generates its HTML document.
	OnAfterGetXMLData (see page 17156)	Occurs after the page producer translates fetches an XML data packet from an XML broker.

	OnBeforeGetContent (see page 17157)	Occurs when the Content method is called.
	OnBeforeGetXMLData (see page 17157)	Occurs before the page producer fetches an XML data packet from an XML broker.
	OnHTMLTag (see page 5620)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Published Properties

	Name	Description
	EnableXMLIslands (see page 17151)	Specifies whether XML data packets should be declared using XML islands if the requesting browser supports them.
	HTMLDoc (see page 17152)	Specifies the HTML template used to generate the value returned by the Content method.
	HTMLFile (see page 17153)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	IncludePathURL (see page 17153)	Indicates the URL where the javascript libraries are installed.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ScriptEngine (see page 17154)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	Styles (see page 17154)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 17155)	Specifies a file containing the individual styles for the page producer's style sheet.
	Tag (see page 2808)	Stores an integer value as part of a component.
	WebPageItems (see page 17155)	Maintains a list of components that generate HTML content for the producer.

100.1.4.2 TInetXPageProducer Properties

The properties of the TInetXPageProducer class are listed here.

Published Properties

	Name	Description
	EnableXMLIslands (see page 17151)	Specifies whether XML data packets should be declared using XML islands if the requesting browser supports them.
	HTMLDoc (see page 17152)	Specifies the HTML template used to generate the value returned by the Content method.
	HTMLFile (see page 17153)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	IncludePathURL (see page 17153)	Indicates the URL where the javascript libraries are installed.
	ScriptEngine (see page 17154)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	Styles (see page 17154)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 17155)	Specifies a file containing the individual styles for the page producer's style sheet.
	WebPageItems (see page 17155)	Maintains a list of components that generate HTML content for the producer.

100.1.4.2.1 TInetXPageProducer.EnableXMLIslands Property

Specifies whether XML data packets should be declared using XML islands if the requesting browser supports them.

Pascal

```
property EnableXMLIslands: Boolean;
```

C++

```
__property Boolean EnableXMLIslands;
```

Description

Set EnableXMLIslands to indicate whether the XML in the generated HTML document should be declared using XML islands. XML islands are only used if the document is created in response to a request message whose headers indicate that the browser supports XML islands.

Set EnableXMLIslands to false to prevent the page producer from declaring XML islands even if the requesting browser supports them.

See Also

GetXMLData (see page 17124)

Content (see page 15274)

100.1.4.2.2 TInetXPageProducer.HTMLDoc Property

Specifies the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLDoc: TStrings;
```

C++

```
__property TStrings HTMLDoc;
```

Description

Set HTMLDoc to a set of strings that make up the HTML template. By default, HTMLDoc is initialized to the following template:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#INCLUDES> <#STYLES> <#WARNINGS> <#FORMS> <#SCRIPT>
</BODY>
</HTML>
```

Each string is a sequence of one or more HTML commands or HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. TCustomInetXPageProducer automatically handles the conversion of the HTML-transparent tags in the default template, as well as the tags described in the help for the Content method. If you set HTMLDoc to include other HTML-transparent tags, you must translate these in an OnHTMLTag event handler.

Note: Setting HTMLDoc clears the HTMLFile property.

See Also

Content (see page 17138)

HTMLFile (see page 17141)

OnHTMLTag (see page 15280)

Creating Web Pages with an InternetExpress Page Producer

Customizing the InternetExpress Page Producer Template

100.1.4.2.3 TInetXPageProducer.HTMLFile Property

Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLFile: TFileName;
```

C++

```
__property TFileName HTMLFile;
```

Description

Set HTMLFile to the name of the file that contains an HTML template. The HTML template is a sequence of HTML commands and HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLFile clears the HTMLDoc property.

Note: When distributing applications that contain page producers which use the HTMLFile property, be sure to distribute the file along with the application. Also, be sure that the value of HTMLFile does not include a hard-coded path name that will not be available on the user's disk.

See Also

Content (see page 17138)

HTMLDoc (see page 17140)

Customizing the InternetExpress Page Producer Template

100.1.4.2.4 TInetXPageProducer.IncludePathURL Property

Indicates the URL where the javascript libraries are installed.

Pascal

```
property IncludePathURL: string;
```

C++

```
__property AnsiString IncludePathURL;
```

Description

Set IncludePathURL to indicate where the javascript libraries are installed. The page producer uses this value to create HTML statements that include the libraries in the generated HTML document. These statements have the form

```
<SCRIPT language=Javascript type="text/javascript" SRC="IncludePathURL/xmldb.js"> </SCRIPT>
```

The javascript libraries define classes and methods that allow the HTML document to interpret XML data packets if the Web

browser is not XML-enabled, and that enable controls in the document to navigate the data, post changes, and apply updates.

See Also

SetIncludePathURL (see page 17125)

Content (see page 15274)

Creating Web Pages with an InternetExpress Page Producer

100.1.4.2.5 TInetXPageProducer.ScriptEngine Property

Names the script engine that generates or interprets script embedded in the documents that this producer creates.

Pascal

```
property ScriptEngine: string;
```

C++

```
__property AnsiString ScriptEngine;
```

Description

ScriptEngine indicates the scripting language that the producer uses in its generated documents. For example, if the producer generates code that includes jscript, the value of ScriptEngine is 'JSCRIPT'.

When ScriptEngine is defined, the page producer generates its content using a special script-producer component. When ScriptEngine is not defined, the page producer uses only the OnHTMLTag event, plus any other built-in translations provided by a descendant class.

See Also

OnHTMLTag (see page 15280)

100.1.4.2.6 TInetXPageProducer.Styles Property

Defines the styles that can be used by the page producer and its Web items.

Pascal

```
property Styles: TStrings;
```

C++

```
__property TStrings Styles;
```

Description

Use Styles to define a style sheet for the HTML document produced by TBaseInetXPageProducer. Each string in the TStrings object specifies a specific style by giving a style selector followed by a set of attributes in curly braces. These definitions can define styles for standard HTML elements such as

```
H2 B  {color: red}
```

or they can define styles that you name, such as

```
.MyStyle  {font-family: arial; font-weight: bold; font-size: 18px }
```

Note that user-defined style names must begin with a dot.

Styles for standard elements are applied automatically to HTML that uses the tag or tags that form the style selector. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector.

Note: If you are sharing a set of styles between multiple page producers or multiple applications, you can define the styles in a

file. To define the styles in a file, use the StylesFile property instead. If StylesFile is defined, it takes precedence over the Styles property.

See Also

SetStyles (see page 17125)

StylesFile (see page 17128)

Content (see page 15274)

Customizing the InternetExpress Page Producer Template

Setting Web Item Properties

100.1.4.2.7 TInetXPageProducer.StylesFile Property

Specifies a file containing the individual styles for the page producer's style sheet.

Pascal

```
property StylesFile: TFileName;
```

C++

```
__property TFileName StylesFile;
```

Description

Set StylesFile to the name of a file containing the style definitions used by the page producer. The page producer uses these style definitions to declare a style sheet in the HTML document it generates.

Styles for standard elements are applied automatically to HTML in the document. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector of a style definition.

Note: If the StylesFile property is not set, you can specify the style definitions using the Styles property instead.

See Also

SetStylesFile (see page 17125)

Styles (see page 17127)

Content (see page 15274)

Customizing the InternetExpress Page Producer Template

Setting Web Item Properties

100.1.4.2.8 TInetXPageProducer.WebPageItems Property

Maintains a list of components that generate HTML content for the producer.

Pascal

```
property WebPageItems: TWebComponentList;
```

C++

```
__property TWebComponentList WebPageItems;
```

Description

WebPageItems lists the components that generate content for the page items producer. Each of these components supports the IWebContent interface, and can contain its own list of Web items that help it generate its content.

The type of items that can be added to the WebPageItems list are restricted by a protected method that is abstract or, in C++ terminology, pure virtual in TBasePageItemsProducer. Descendant classes implement this protected method to indicate what types of items can be added to the list.

At design time, use the Web page editor to add components to the WebPageItems property.

Note: Typically, WebPageItems lists all the components that generate content for the page items producer, in the order that the generated content appears in the final HTML document. However, the page items producer can transform COMPONENT tags in its template using components that are not in this list, as long as they have the same owner as the page items producer..

See Also

IWebContent (see page 24792)

Creating Web Pages with an InternetExpress Page Producer

100.1.4.3 TInetXPageProducer Events

The events of the TInetXPageProducer class are listed here.

Published Events

	Name	Description
	OnAfterGetContent (see page 17156)	Occurs after the page producer generates its HTML document.
	OnAfterGetXMLData (see page 17156)	Occurs after the page producer translates fetches an XML data packet from an XML broker.
	OnBeforeGetContent (see page 17157)	Occurs when the Content method is called.
	OnBeforeGetXMLData (see page 17157)	Occurs before the page producer fetches an XML data packet from an XML broker.

100.1.4.3.1 TInetXPageProducer.OnAfterGetContent Event

Occurs after the page producer generates its HTML document.

Pascal

```
property OnAfterGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAfterGetContent;
```

Description

Write an OnAfterGetContent event handler to take specific action immediately after the page producer generates an HTML document. This event occurs immediately before the Content method returns the generated document. Typically it is used in applications where the page producer automatically responds to messages as the Provider property of an action item or the Response producer property of an XML broker.

See Also

OnBeforeGetContent (see page 17130)

Content (see page 15274)

Producer (see page 15138)

100.1.4.3.2 TInetXPageProducer.OnAfterGetXMLData Event

Occurs after the page producer translates fetches an XML data packet from an XML broker.

Pascal

```
property OnAfterGetXMLData: TXMLDataEvent;
```

C++

```
__property TXMLDataEvent OnAfterGetXMLData;
```

Description

Write an OnAfterGetXMLData event handler to take specific action immediately after the page producer obtains an XML data packet from an XML broker. Typically, the OnAfterGetXMLData event handler responds to any values returned by the provider on the application server as the OwnerData parameter.

When the page producer generates its HTML document (that is, when the Content method executes), it adds code to define the XML data packets manipulated in the document. To fetch these XML data packets, the page producer calls an XML broker's RequestRecords method. OnAfterGetXMLData occurs after the page producer has returned an XML data packet but before it is assigned as part of the emerging content.

See Also

OnBeforeGetXMLData (see page 17130)

GetXMLData (see page 17124)

OnHTMLTag (see page 15280)

Content (see page 15274)

100.1.4.3.3 TInetXPageProducer.OnBeforeGetContent Event

Occurs when the Content method is called.

Pascal

```
property OnBeforeGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnBeforeGetContent;
```

Description

Write an OnBeforeGetContent event handler to take specific action immediately before the page producer generates an HTML document. This event occurs when the Content method is called before the page producer starts assembling its HTML document. Typically it is used in applications where the page producer automatically responds to messages as the Provider property of an action item or the Response producer property of an XML broker.

See Also

OnAfterGetContent (see page 17129)

Content (see page 15274)

Producer (see page 15138)

100.1.4.3.4 TInetXPageProducer.OnBeforeGetXMLData Event

Occurs before the page producer fetches an XML data packet from an XML broker.

Pascal

```
property OnBeforeGetXMLData: TXMLDataEvent;
```

C++

```
__property TXMLDataEvent OnBeforeGetXMLData;
```

Description

Write an OnBeforeGetXMLData event handler to take specific action immediately before the page producer requests an XML data packet from an XML broker. Typically, the OnBeforeGetXMLData event handler encodes any values for the provider on the application server as the OwnerData parameter or initializes properties on the XML broker that affect the data packet.

When the page producer generates its HTML document (that is, when the Content method executes), it adds code to define the XML data packets manipulated in the document. To fetch these XML data packets, the page producer calls an XML broker's RequestRecords method. OnBeforeGetXMLData occurs immediately before the page producer requests an XML data packet. Any value assigned to the OwnerData parameter is passed to the XML broker's OnRequestRecords event handler, and from there to the provider's OnBeforeGetRecords event handler.

See Also

OnAfterGetXMLData (see page 17129)

OnRequestRecords (see page 26080)

GetXMLData (see page 17124)

Content (see page 15274)

OnHTMLTag (see page 15280)

MaxRecords (see page 26072)

Params (see page 26073)

100.1.5 TMidasPageProducer Class

TMidasPageProducer assembles an HTML document that represents live database information from an application server.

Class Hierarchy

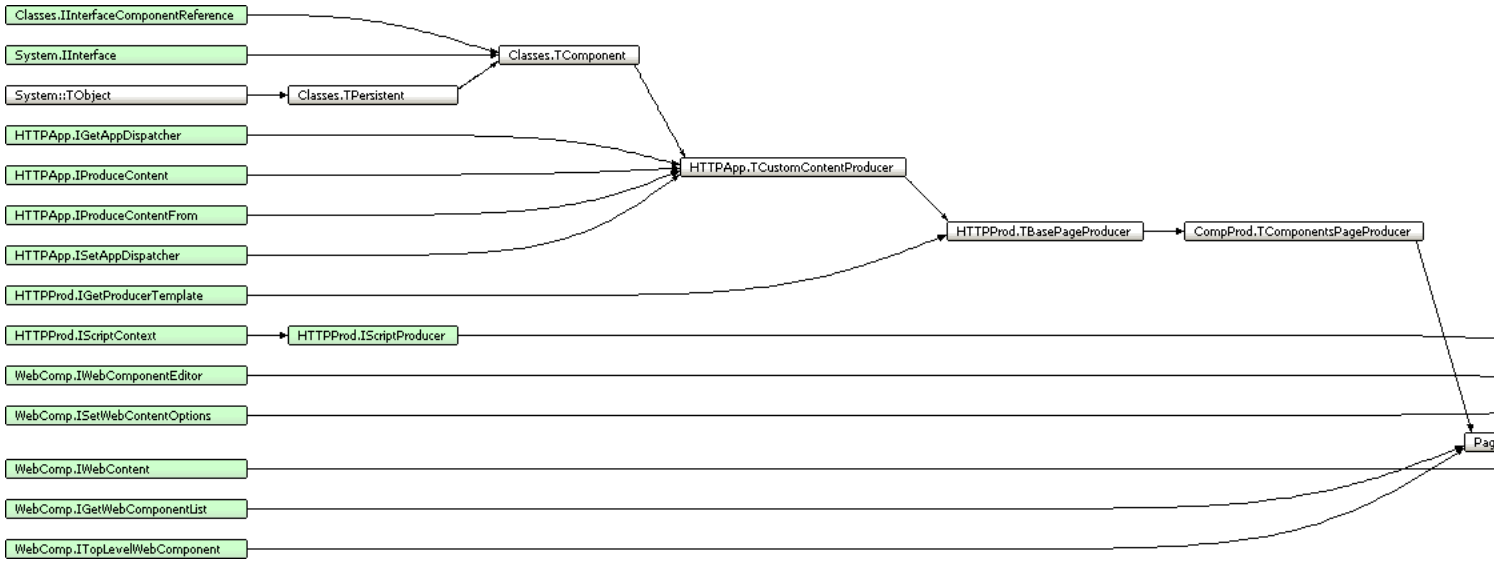

Pascal

```
TMidasPageProducer = class(TInetXPageProducer);
```

C++

```
class TMidasPageProducer : public TInetXPageProducer;
```

File

MidProd

Description

Do not use TMidasPageProducer. It is included only for backward compatibility. To add a page producer that assembles Web pages in an InternetExpress application, use TInetXPageProducer (see page 17145) instead.

See Also

TWebActionItem (see page 15130)

TWebActionItem (see page 15130)

TXMLBroker (see page 26049)

100.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AddStdIncludes (see page 17160)	This is function MidProd.AddStdIncludes.
	PathInfoToRelativePath (see page 17160)	This is function MidProd.PathInfoToRelativePath.

100.2.1 MidProd.AddStdIncludes Function

Pascal

```
procedure AddStdIncludes(StdIncludes: TStdIncludes; AddIntf: IAddScriptElements);
```

C++

```
AddStdIncludes(TStdIncludes StdIncludes, IAddScriptElements AddIntf);
```

File

MidProd

Description

This is function MidProd.AddStdIncludes.

100.2.2 MidProd.PathInfoToRelativePath Function

Pascal

```
function PathInfoToRelativePath(const PathInfo: string): string;
```

C++

```
AnsiString PathInfoToRelativePath(const AnsiString PathInfo);
```

File

MidProd

Description

This is function MidProd.PathInfoToRelativePath.

100.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TStdInclude (see page 17161)	This is record MidProd.TStdInclude.

100.3.1 MidProd.TStdInclude Enumeration

Pascal

```
TStdInclude = (
  jsDom,
  jsDB,
  jsDisp,
  jsBind,
  jsErrDisp,
  jsShow
);
```

C++

```
enum TStdInclude {
  jsDom,
  jsDB,
  jsDisp,
  jsBind,
  jsErrDisp,
  jsShow
};
```

File

MidProd

Description

This is record MidProd.TStdInclude.

100.4 Types

The following table lists types in this documentation.

Types

Name	Description
TBaseMidasPageProducer (see page 17162)	This is type MidProd.TBaseMidasPageProducer.
TCustomMidasPageProducer (see page 17162)	This is type MidProd.TCustomMidasPageProducer.
TMidasPageElements (see page 17163)	This is type MidProd.TMidasPageElements.
TStdIncludes (see page 17163)	This is type MidProd.TStdIncludes.
TXMLDataEvent (see page 17163)	This is type MidProd.TXMLDataEvent.

100.4.1 MidProd.TBaseMidasPageProducer Type

Pascal

```
TBaseMidasPageProducer = TBaseInetXPageProducer;
```

C++

```
TBaseInetXPageProducer TBaseMidasPageProducer;
```

File

MidProd

Description

This is type MidProd.TBaseMidasPageProducer.

100.4.2 MidProd.TCustomMidasPageProducer Type

Pascal

```
TCustomMidasPageProducer = TCustomInetXPageProducer;
```

C++

```
TCustomInetXPageProducer TCustomMidasPageProducer;
```

File

MidProd

Description

This is type MidProd.TCustomMidasPageProducer.

100.4.3 MidProd.TMidasPageElements Type

Pascal

```
TMidasPageElements = TInetXPageElements;
```

C++

```
TInetXPageElements TMidasPageElements;
```

File

MidProd

Description

This is type MidProd.TMidasPageElements.

100.4.4 MidProd.TStdIncludes Type

Pascal

```
TStdIncludes = set of TStdInclude;
```

C++

```
TStdInclude TStdIncludes;
```

File

MidProd

Description

This is type MidProd.TStdIncludes.

100.4.5 MidProd.TXMLDataEvent Type

Pascal

```
TXMLDataEvent = procedure (Sender: TObject; Request: TWebRequest; XMLBroker: TXMLBroker; var
OwnerData: OleVariant) of object;
```

C++

```
(Sender: TObject; Request: TWebRequest; XMLBroker: TXMLBroker; var OwnerData: OleVariant) (
TXMLDataEvent)();
```

File

MidProd

Description

This is type MidProd.TXMLDataEvent.

100.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
sXMLBroker (see page 17164)	This is constant MidProd.sXMLBroker.
sXMLReadyVar (see page 17164)	This is constant MidProd.sXMLReadyVar.

100.5.1 MidProd.sXMLBroker Constant

Pascal

```
sXMLBroker = 'XMLBROKER';
```

C++

```
sXMLBroker = 'XMLBROKER';
```

File

MidProd

Description

This is constant MidProd.sXMLBroker.

100.5.2 MidProd.sXMLReadyVar Constant

Pascal

```
sXMLReadyVar = 'xml_ready';
```

C++

```
sXMLReadyVar = 'xml_ready';
```

File

MidProd

Description

This is constant MidProd.sXMLReadyVar.

101 MPlayer Namespace

This is namespace MPlayer.

Topics

Name	Description
Classes (see page 17167)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 17229)	The following table lists structs, records, enums in this documentation.
Types (see page 17237)	The following table lists types in this documentation.

Classes

	Name	Description
	EMCIDeviceError (see page 17167)	EMCIDeviceError is the exception class for devices that provide a Media Control Interface (MCI) driver.
	TMediaPlayer (see page 17167)	TMediaPlayer controls devices that provide a Media Control Interface (MCI) driver.

Structs, Records, Enums

	Name	Description
	TMPBtnType (see page 17229)	TMPBtnType defines the possible buttons in a TMediaPlayer (see page 17167) component and TButtonSet (see page 17238) represents a set of those buttons.
	TMPButton (see page 17230)	This is record MPlayer.TMPButton.
	TMPDevCaps (see page 17231)	TMPDevCaps defines the capabilities of a TMediaPlayer (see page 17167) component and TMPDevCapsSet (see page 17238) represents a set of those capabilities.
	TMPDeviceTypes (see page 17231)	TMPDeviceTypes identifies a multimedia device.
	TMPGlyph (see page 17233)	This is record MPlayer.TMPGlyph.
	TMPModes (see page 17233)	TMPModes identifies the state of a TMediaPlayer (see page 17167) component.
	TMPNotifyValues (see page 17234)	TMPNotifyValues identifies the types of notifications a media device can give a TMediaPlayer (see page 17167) component.
	TMPTimeFormats (see page 17235)	TMPTimeFormats indicates the format a media player uses to indicate position.

Types

Name	Description
EMPNotify (see page 17237)	EMPNotify is the type of event handlers that respond when the user interacts with a TMediaPlayer (see page 17167) component.
EMPPostNotify (see page 17237)	EMPPostNotify is the type of a media player's OnPostClick event handler.
TButtonSet (see page 17238)	TButtonSet is a set of TMPBtnType (see page 17229) values.

TMPDevCapsSet (see page 17238)	This is type MPlayer.TMPDevCapsSet.

101.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EMCIDeviceError (see page 17167)	EMCIDeviceError is the exception class for devices that provide a Media Control Interface (MCI) driver.
	TMediaPlayer (see page 17167)	TMediaPlayer controls devices that provide a Media Control Interface (MCI) driver.

101.1.1 EMCIDeviceError Class

EMCIDeviceError is the exception class for devices that provide a Media Control Interface (MCI) driver.

Class Hierarchy

Pascal

```
EMCIDeviceError = class(Exception);
```

C++

```
class EMCIDeviceError : public Exception;
```

File

MPlayer

Description

EMCIDeviceError is raised if an error occurs when accessing a multimedia device. The most common cause for the exception is trying to access a multimedia device before it has been opened with the Open method of TMediaPlayer (see page 17167).

See Also

TMediaPlayer (see page 17167)

TMediaPlayer (see page 17167)

101.1.2 TMediaPlayer Class

TMediaPlayer controls devices that provide a Media Control Interface (MCI) driver.

Class Hierarchy

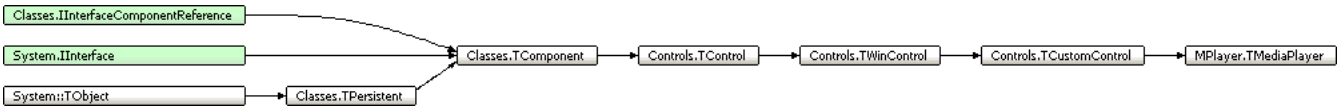

Pascal

```
TMediaPlayer = class(TCustomControl);
```

C++

```
class TMediaPlayer : public TCustomControl;
```

File

MPlayer

Description

The TMediaPlayer component includes a set of buttons (Play (see page 17190), Stop (see page 17195), Eject (see page 17184), and so on) that control a multimedia device such as a CD-ROM drive, MIDI sequencer, or VCR. A multimedia device may be hardware or software.

The media player component contains of multiple buttons. These buttons can be clicked with the mouse, but are not separate objects or button components.

ButtonValueAction

Button	Value	Action
Play (see page 17190)	btPlay	Plays the media player
Pause (see page 17189)	btPause	Pauses playing or recording. If already paused when clicked, resumes playing or recording.
Stop (see page 17195)	btStop	Stops playing or recording
Next (see page 17187)	btNext	Skips to the next track, or to the end if the medium doesn't use tracks
Prev	btPrev	Skips to the previous track, or to the beginning if the medium doesn't use tracks
Step (see page 17194)	btStep	Moves forward a number of frames
Back (see page 17182)	btBack	Moves backward a number of frames
Record	btRecord	Starts recording
Eject (see page 17184)	btEject	Ejects the medium

The multimedia device is played, paused, stopped, and so on when the user clicks the corresponding button on the TMediaPlayer component. The device can also be controlled by the control methods that correspond to the buttons (Play (see page 17190), Pause (see page 17189), Stop (see page 17195), Next (see page 17187), Previous (see page 17191), Step (see page 17194), Back (see page 17182), StartRecording (see page 17194), and Eject (see page 17184)).

The type of multimedia device (such as dtWaveAudio or dtVideodisc) is specified by the DeviceType (see page 17201) property. If the device stores its media in a file, the name of the media file is specified by the FileName (see page 17205) property. If DeviceType (see page 17201) is dtAutoSelect, the media player attempts to determine the type of device from the extension of the file specified by FileName (see page 17205).

To have the media player attempt to open the device specified by DeviceType (see page 17201) automatically when the media player component is created at runtime, set the AutoOpen (see page 17198) property to true.

See Also

Working with Multimedia

101.1.2.1 TMediaPlayer Members

The following tables list the members exposed by TMediaPlayer.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.

	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	AutoButtonSet (see page 17181)	Enables or Disables the media player's buttons.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 17182)	Generates an OnClick (see page 17222) event.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.

	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoNotify (see page 17184)	Generates an OnNotify (see page 17228) event.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.

	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 17186)	Provides responses when the user types arrow keys or the space bar.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 17186)	Initializes the media player after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MMNotify (see page 17187)	Responds to a notification that the MCI device has completed an operation.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 17188)	Responds when components are about to be inserted or removed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 17189)	Renders the image of the media player.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	PostClick (see page 17191)	Determines the action that takes place when an OnPostClick (see page 17228) event occurs.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.

	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
V	Updated (see page 17195)	Signals that the media player has finished updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.

	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.

	Back (see page 17182)	Steps backward a number of frames (determined by the value of the Frames (see page 17205) property) in the currently loaded medium.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Close (see page 17183)	Closes the open multimedia device.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 17180)	Creates a media player object.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 17180)	Destroys an instance of the media player.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	Eject (see page 17184)	Releases the loaded medium from the open multimedia device.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 17187)	Moves to the beginning of the next track of the currently loaded medium.
	Open (see page 17188)	Opens a multimedia device.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Pause (see page 17189)	Toggles the open multimedia device on or off.
	PauseOnly (see page 17190)	Pauses the open multimedia device.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	Play (see page 17190)	Plays the media loaded in the open multimedia device.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	Previous (see page 17191)	Sets the current position to the beginning of the previous track if the position was at the beginning of a track when Previous was called.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	Resume (see page 17192)	Resumes playing or recording the currently paused multimedia device.
	Rewind (see page 17193)	Sets the current position to the beginning of the the medium, which is stored in the Start (see page 17215) property.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Save (see page 17193)	Saves the currently loaded medium to the file specified in the FileName (see page 17205) property.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	StartRecording (see page 17194)	Begins recording from the current Position (see page 17213) or from the position specified in StartPos (see page 17216).
	Step (see page 17194)	Moves forward a number of frames (determined by the Frames (see page 17205) property) in the currently loaded medium.
	Stop (see page 17195)	Halts playing or recording.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Capabilities (see page 17199)	Determines the capabilities of the open multimedia device.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.

R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DeviceID (see page 17201)	Specifies the device ID for the currently open multimedia device.
	DisplayRect (see page 17202)	Specifies a rectangular area in the control specified by the Display (see page 17202) property that is used to display output from a multimedia device.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EndPos (see page 17204)	Specifies the position within the currently loaded medium at which to stop playing or recording.
R	Error (see page 17204)	Specifies the MCI error code returned by the media control method.
R	ErrorMessage (see page 17205)	Describes the error code stored in the Error (see page 17204) property.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Frames (see page 17205)	Specifies the number of frames moved forward or backward.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
R	Length (see page 17206)	Specifies the length of the medium in the open multimedia device.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	Mode (see page 17206)	Indicates the state of the currently open multimedia device.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	Notify (see page 17206)	Determines whether an OnNotify (see page 17228) event is generated.
R	NotifyValue (see page 17207)	Indicates the result of the last media control method that requested a notification.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	Position (see page 17213)	Specifies the current position within the currently loaded medium.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
R	Start (see page 17215)	Specifies the starting position within the currently loaded medium.
	StartPos (see page 17216)	Specifies the position within the currently loaded medium from which to begin playing or recording.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TimeFormat (see page 17217)	Determines the format used to specify position information.

R	TrackLength (see page 17218)	Reports the length of the track.
R	TrackPosition (see page 17218)	Reports the starting position of the track.
R	Tracks (see page 17218)	Specifies the number of playable tracks on the open multimedia device.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.
	Wait (see page 17221)	Determines whether a media control method returns control to the application only after it has been completed.

Published Events

	Name	Description
	OnClick (see page 17222)	Occurs when the user presses and releases the mouse button while the mouse pointer is over one of the control buttons, or when the user presses Spacebar while the media player control has focus.
	OnContextPopup (see page 17222)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnEnter (see page 17223)	Occurs when a control receives the input focus.
	OnExit (see page 17226)	Occurs when the input focus shifts away from one control to another.
	OnMouseEnter (see page 17227)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 17227)	Occurs when the user moves the mouse outside of a control.
	OnNotify (see page 17228)	Occurs upon the completion of a media control method.
	OnPostClick (see page 17228)	Occurs after the OnClick (see page 17222) event handler is called.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 17197)	Specifies how the control is anchored to its parent.
	AutoEnable (see page 17197)	Determines whether the media player automatically enables and disables individual buttons in the component.
	AutoOpen (see page 17198)	Determines if the media player is opened automatically when the application is run.
	AutoRewind (see page 17199)	Determines if the media player control rewinds before playing or recording.
	ColoredButtons (see page 17200)	Determines which buttons on the media player control have color.
	Constraints (see page 17200)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DeviceType (see page 17201)	Specifies a multimedia device type to open the media player.
	Display (see page 17202)	Specifies the display window for a multimedia device that uses a window for output.
	Enabled (see page 17203)	Controls whether the control responds to mouse, keyboard, and timer events.
	EnabledButtons (see page 17203)	Determines which buttons on the media player are usable.
	FileName (see page 17205)	Specifies the opened or saved media file.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentShowHint (see page 17208)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 17208)	Identifies the pop-up menu associated with the control.
	Shareable (see page 17213)	Determines whether more than one application can share a multimedia device.

	ShowHint (see page 17214)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 17216)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 17217)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 17219)	Determines whether the component appears onscreen.
	VisibleButtons (see page 17220)	Indicates which of the buttons on the media player are visible.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

101.1.2.2 TMediaPlayer.Create Constructor

Creates a media player object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TMediaPlayer(TComponent * AOwner);
```

Description

Call Create to instantiate a media player at runtime. For media players created at design time, Create is called automatically.

Create allocates memory for a media player and calls the inherited Create. Then it loads the bitmaps for the media player's buttons and initializes its properties, setting AutoEnable (see page 17197), AutoRewind (see page 17199), Colored, Enabled (see page 17203), and Visible (see page 17219) to true, AutoOpen (see page 17198) to false, and DeviceType (see page 17201) to dtAutoSelect.

See Also

Create (see page 5918)

ParentWindow (see page 6210)

101.1.2.3 TMediaPlayer.Destroy Destructor

Destroys an instance of the media player.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMediaPlayer();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free checks that the TMediaPlayer (see page 17167) reference is not nil before it calls Destroy.

Destroy first verifies that no device is open, then destroys the bitmaps used for the media player's buttons and calls the inherited Destroy.

See Also

Destroy (see page 5919)

Free (see page 22300)

101.1.2.4 TMediaPlayer Methods

The methods of the TMediaPlayer class are listed here.

Protected Methods

	Name	Description
	AutoButtonSet (see page 17181)	Enables or Disables the media player's buttons.
	Click (see page 17182)	Generates an OnClick (see page 17222) event.
	DoNotify (see page 17184)	Generates an OnNotify (see page 17228) event.
V	KeyDown (see page 17186)	Provides responses when the user types arrow keys or the space bar.
V	Loaded (see page 17186)	Initializes the media player after it is loaded from a stream.
	MMNotify (see page 17187)	Responds to a notification that the MCI device has completed an operation.
V	Notification (see page 17188)	Responds when components are about to be inserted or removed.
V	Paint (see page 17189)	Renders the image of the media player.
	PostClick (see page 17191)	Determines the action that takes place when an OnPostClick (see page 17228) event occurs.
V	Updated (see page 17195)	Signals that the media player has finished updating.

Public Methods

	Name	Description
	Back (see page 17182)	Steps backward a number of frames (determined by the value of the Frames (see page 17205) property) in the currently loaded medium.
	Close (see page 17183)	Closes the open multimedia device.
	Eject (see page 17184)	Releases the loaded medium from the open multimedia device.
	Next (see page 17187)	Moves to the beginning of the next track of the currently loaded medium.
	Open (see page 17188)	Opens a multimedia device.
	Pause (see page 17189)	Toggles the open multimedia device on or off.
	PauseOnly (see page 17190)	Pauses the open multimedia device.
	Play (see page 17190)	Plays the media loaded in the open multimedia device.
	Previous (see page 17191)	Sets the current position to the beginning of the previous track if the position was at the beginning of a track when Previous was called.
	Resume (see page 17192)	Resumes playing or recording the currently paused multimedia device.
	Rewind (see page 17193)	Sets the current position to the beginning of the the medium, which is stored in the Start (see page 17215) property.
	Save (see page 17193)	Saves the currently loaded medium to the file specified in the FileName (see page 17205) property.
	StartRecording (see page 17194)	Begins recording from the current Position (see page 17213) or from the position specified in StartPos (see page 17216).
	Step (see page 17194)	Moves forward a number of frames (determined by the Frames (see page 17205) property) in the currently loaded medium.
	Stop (see page 17195)	Halts playing or recording.

101.1.2.4.1 TMediaPlayer.AutoButtonSet Method

Enables or Disables the media player's buttons.

Pascal

```
procedure AutoButtonSet(Btn: TMPBtnType); dynamic;
```

C++

```
__fastcall AutoButtonSet(TMPBtnType Btn);
```

Description

AutoButtonSet implements the automatic enabling and disabling of the media player's buttons when the AutoEnable (see page 17197) property is true. The Btn parameter indicates which of the media player's buttons the user pressed.

See Also

AutoEnable (see page 17197)

EnabledButtons (see page 17203)

101.1.2.4.2 TMediaPlayer.Back Method

Steps backward a number of frames (determined by the value of the Frames (see page 17205) property) in the currently loaded medium.

Pascal

```
procedure Back;
```

C++

```
__fastcall Back();
```

Description

Back is called when the Back button on the media player control is clicked at runtime.

Upon completion, Back stores a numerical error code in the Error (see page 17204) property and the corresponding error message in the ErrorMessage (see page 17205) property.

The Wait (see page 17221) property determines whether control is returned to the application before the Back method has been completed. The Notify (see page 17206) property determines whether Back generates an OnNotify (see page 17228) event.

See Also

Error (see page 17204)

ErrorMessage (see page 17205)

Notify (see page 17206)

OnNotify (see page 17228)

Wait (see page 17221)

Adding Audio and/or Video Clips to an Application

101.1.2.4.3 TMediaPlayer.Click Method

Generates an OnClick (see page 17222) event.

Pascal

```
procedure Click(Button: TMPBtnType; var DoDefault: Boolean); dynamic; reintroduce;
```

C++

```
__fastcall Click(TMPBtnType Button, Boolean DoDefault);
```

Description

Click for media-player components is the implementation method for the OnClick (see page 17222) event. By default, Click does nothing other than call any event handler attached to the media player's OnClick (see page 17222) event. Click can be overridden in descendant classes to customize responses to clicks.

See Also

OnClick (see page 17222)

101.1.2.4.4 TMediaPlayer.Close Method

Closes the open multimedia device.

Pascal

```
procedure Close;
```

C++

```
__fastcall Close();
```

Description

Upon completion, Close stores a numerical error code in the Error (see page 17204) property, and the corresponding error message in the ErrorMessage (see page 17205) property.

The Wait (see page 17221) property determines whether control is returned to the application before the Close method is completed. The Notify (see page 17206) property determines whether Close generates an OnNotify (see page 17228) event.

Close is called automatically when the application is terminated.

C++ Examples:

```
/*
This code ejects the CD from the CD-ROM player and shuts
down the media player after 10 seconds. For the code to run
correctly, you must have your CD audio device installed
correctly, and the device must have software-ejecting
capabilities.
*/

Word TimeOver;

void __fastcall TForm1::FormClick(TObject *Sender)
{
  MediaPlayer1->DeviceType = dtCDAudio;
  MediaPlayer1->Open();
  MediaPlayer1->Play();
  Timer1->Enabled = true;
  TimeOver = 0;
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  if (TimeOver == 10)
  {
    MediaPlayer1->Eject();
    MediaPlayer1->Close();
    // disable the timer, as we are done
    Timer1->Enabled = false;
  }
  else
    TimeOver++;
}
```

Delphi Examples:

```
{
This code ejects the CD from the CD-ROM player and shuts
down the media player after 10 seconds. For the code to run
correctly, you must have your CD audio device installed
correctly, and the device must have software-ejecting
capabilities.
}
var
```

```
    TimerOver: Word;

  procedure TForm1.FormClick(Sender: TObject);
  begin
    MediaPlayer1.DeviceType := dtCDAudio;
    MediaPlayer1.Open;
    MediaPlayer1.Play;
  end;

  procedure TForm1.Timer1Timer(Sender: TObject);
  begin
    if TimeOver = 10 then
    begin
      MediaPlayer1.Eject;
      MediaPlayer1.Close;
      { disable the timer, as we are done }
      Timer1.Enabled := False;
    end
    else
      if (MediaPlayer1.Mode = mpOpen) then
       Inc(TimeOver);
  end;
```

See Also

Error (see page 17204)

ErrorMessage (see page 17205)

Notify (see page 17206)

OnNotify (see page 17228)

Wait (see page 17221)

101.1.2.4.5 TMediaPlayer.DoNotify Method

Generates an OnNotify (see page 17228) event.

Pascal

```
procedure DoNotify; dynamic;
```

C++

```
__fastcall DoNotify();
```

Description

DoNotify is the implementation method for a media-player component's OnNotify (see page 17228) event. The media player's MMNotify (see page 17187) method calls DoNotify after correcting the enabling and disabling of buttons and setting internal flags from the values passed in the notification message parameters.

By default, DoNotify does nothing except call any event handler attached to the media player's OnNotify (see page 17228) event. DoNotify can be overridden to provide other responses in addition to the inherited event-handler call.

See Also

MMNotify (see page 17187)

OnNotify (see page 17228)

101.1.2.4.6 TMediaPlayer.Eject Method

Releases the loaded medium from the open multimedia device.

Pascal

```
procedure Eject;
```

C++

```
__fastcall Eject();
```

Description

Eject is called when the Eject button on the media player control is clicked at runtime. It ejects the loaded medium from the open multimedia device.

Upon completion, Eject stores a numerical error code in the Error (see page 17204) property, and the corresponding error message in the ErrorMessage (see page 17205) property.

The Wait (see page 17221) property determines whether control is returned to the application before the Eject method has been completed. The Notify (see page 17206) property determines whether Eject generates an OnNotify (see page 17228) event.

C++ Examples:

```
/*
This code ejects the CD from the CD-ROM player and shuts
down the media player after 10 seconds. For the code to run
correctly, you must have your CD audio device installed
correctly, and the device must have software-ejecting
capabilities.
*/

Word TimeOver;

void __fastcall TForm1::FormClick(TObject *Sender)
{
  MediaPlayer1->DeviceType = dtCDAudio;
  MediaPlayer1->Open();
  MediaPlayer1->Play();
  Timer1->Enabled = true;
  TimeOver = 0;
}

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  if (TimeOver == 10)
  {
    MediaPlayer1->Eject();
    MediaPlayer1->Close();
    // disable the timer, as we are done
    Timer1->Enabled = false;
  }
  else
    TimeOver++;
}
```

Delphi Examples:

```
{
This code ejects the CD from the CD-ROM player and shuts
down the media player after 10 seconds. For the code to run
correctly, you must have your CD audio device installed
correctly, and the device must have software-ejecting
capabilities.
}
var
  TimerOver: Word;

procedure TForm1.FormClick(Sender: TObject);
```

```
begin
  MediaPlayer1.DeviceType := dtCDAudio;
  MediaPlayer1.Open;
  MediaPlayer1.Play;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  if TimeOver = 10 then
  begin
    MediaPlayer1.Eject;
    MediaPlayer1.Close;
    { disable the timer, as we are done }
    Timer1.Enabled := False;
  end
  else
    if (MediaPlayer1.Mode = mpOpen) then
     Inc(TimeOver);
end;
```

See Also

Error (see page 17204)

ErrorMessage (see page 17205)

Notify (see page 17206)

OnNotify (see page 17228)

Wait (see page 17221)

Adding Audio and/or Video Clips to an Application

101.1.2.4.7 TMediaPlayer.KeyDown Method

Provides responses when the user types arrow keys or the space bar.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

KeyDown overrides the inherited method to implement the media player's response to certain keystrokes. It changes the current button when the user types the left or right arrow keys. When the user presses the space bar, KeyDown simulates a click on the current button.

See Also

EnabledButtons (see page 17203)

KeyDown (see page 6163)

101.1.2.4.8 TMediaPlayer.Loaded Method

Initializes the media player after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Loaded provides an opportunity for a component to initialize itself after all its parts have loaded from a stream. TMediaPlayer (see page 17167) overrides the inherited method to implement the AutoOpen (see page 17198) property.

See Also

Create (see page 17180)

AutoOpen (see page 17198)

101.1.2.4.9 TMediaPlayer.MMNotify Method

Responds to a notification that the MCI device has completed an operation.

Pascal

```
procedure MMNotify(var Message: TMessage); message MM_MCINOTIFY;
```

C++

```
__fastcall MMNotify(TMessage Message);
```

Description

MMNotify responds to a notification message from Windows that a multimedia command completed by updating button states if the AutoEnable (see page 17197) property is true, setting internal flags to reflect the outcome of the command, then calling the DoNotify (see page 17184) method to trigger the OnNotify (see page 17228) event and any other special responses.

See Also

AutoEnable (see page 17197)

DoNotify (see page 17184)

OnNotify (see page 17228)

101.1.2.4.10 TMediaPlayer.Next Method

Moves to the beginning of the next track of the currently loaded medium.

Pascal

```
procedure Next;
```

C++

```
__fastcall Next();
```

Description

If the current position is at the last track when Next is called, Next makes the current position the beginning of the last track. If the multimedia device doesn't use tracks, Next goes to the end of the medium. Next is called when the Next button on the media player control is clicked at runtime.

Upon completion, Next stores a numerical error code in the Error (see page 17204) property, and the corresponding error message in the ErrorMessage (see page 17205) property.

The Wait (see page 17221) property determines whether control is returned to the application before the Next method has completed. The Notify (see page 17206) property determines whether Next generates an OnNotify (see page 17228) event.

See Also

Error (see page 17204)

ErrorMessage (see page 17205)

Notify (see page 17206)

OnNotify (see page 17228)

Wait (see page 17221)

Adding Audio and/or Video Clips to an Application

101.1.2.4.11 TMediaPlayer.Notification Method

Responds when components are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

By default, components pass along the notification to their owned components, if any. Notification calls the inherited Notification and then, if the Display (see page 17202) window is deleted at runtime, it sets the video display back to original window.

See Also

Notification (see page 5775)

Display (see page 17202)

101.1.2.4.12 TMediaPlayer.Open Method

Opens a multimedia device.

Pascal

```
procedure Open;
```

C++

```
__fastcall Open();
```

Description

Use Open to open a multimedia device. The multimedia device type must be specified in the DeviceType (see page 17201) property before a device can be opened.

Upon completion, Open stores a numerical error code in the Error (see page 17204) property, and the corresponding error message in the ErrorMessage (see page 17205) property.

The Wait (see page 17221) property determines whether control is returned to the application before the Open method is completed. The Notify (see page 17206) property determines whether Open generates an OnNotify (see page 17228) event.

See Also

Error (see page 17204)

ErrorMessage (see page 17205)

Notify (see page 17206)

OnNotify (see page 17228)

Wait (see page 17221)

Adding Audio and/or Video Clips to an Application

101.1.2.4.13 TMediaPlayer.Paint Method

Renders the image of the media player.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

When the media player component receives a WM_PAINT message, like other custom controls, it responds by initializing the control's canvas and calling Paint. Paint draws the frame and all visible buttons.

When creating a descendant media player class, Paint can be overridden to draw the image of the corresponding media player object.

See Also

VisibleButtons (see page 17220)

101.1.2.4.14 TMediaPlayer.Pause Method

Toggles the open multimedia device on or off.

Pascal

```
procedure Pause;
```

C++

```
__fastcall Pause();
```

Description

If the device is already paused when Pause is called, the device resumes playing or recording by calling the Resume (see page 17192) method. Pause is called when the Pause button on the media player control is clicked at runtime.

Upon completion, Pause stores a numerical error code in the Error (see page 17204) property and the corresponding error message in the ErrorMessage (see page 17205) property.

The Wait (see page 17221) property determines whether control is returned to the application before the Pause method has completed. The Notify (see page 17206) property determines whether Pause generates an OnNotify (see page 17228) event.

See Also

Error (see page 17204)

ErrorMessage (see page 17205)

Notify (see page 17206)

OnNotify (see page 17228)

Resume (see page 17192)

Wait (see page 17221)

Adding Audio and/or Video Clips to an Application

101.1.2.4.15 TMediaPlayer.PauseOnly Method

Pauses the open multimedia device.

Pascal

```
procedure PauseOnly;
```

C++

```
__fastcall PauseOnly();
```

Description

If the device is already paused when PauseOnly is called, the device will remain paused.

Upon completion, PauseOnly stores a numerical error code in the Error (see page 17204) property and the corresponding error message in the ErrorMessage (see page 17205) property.

The Wait (see page 17221) property determines whether control is returned to the application before the PauseOnly method has completed. The Notify (see page 17206) property determines whether PauseOnly generates an OnNotify (see page 17228) event.

See Also

Error (see page 17204)

ErrorMessage (see page 17205)

Notify (see page 17206)

OnNotify (see page 17228)

Wait (see page 17221)

101.1.2.4.16 TMediaPlayer.Play Method

Plays the media loaded in the open multimedia device.

Pascal

```
procedure Play;
```

C++

```
__fastcall Play();
```

Description

Play is called when the Play button on the media player control is clicked at runtime.

Upon completion, Play stores a numerical error code in the Error (see page 17204) property and the corresponding error message in the ErrorMessage (see page 17205) property.

The Wait (see page 17221) property determines whether control is returned to the application before the Play method has completed. The Notify (see page 17206) property determines whether Play generates an OnNotify (see page 17228) event.

If the StartPos (see page 17216) property is set, playing starts at the position specified in StartPos (see page 17216). Otherwise, playing starts at the current position, specified in the Position (see page 17213) property. Similarly, if the EndPos

(see page 17204) property is set, playing stops at the position specified in EndPos (see page 17204). Otherwise, playing stops at the end of the medium.

Whether the medium (specified in the Position (see page 17213) property) is rewound before playing starts depends on the AutoRewind (see page 17199) property.

See Also

AutoRewind (see page 17199)

EndPos (see page 17204)

Error (see page 17204)

ErrorMessage (see page 17205)

Notify (see page 17206)

OnNotify (see page 17228)

Position (see page 17213)

Resume (see page 17192)

StartPos (see page 17216)

Wait (see page 17221)

Adding Audio and/or Video Clips to an Application

101.1.2.4.17 TMediaPlayer.PostClick Method

Determines the action that takes place when an OnPostClick (see page 17228) event occurs.

Pascal

```
procedure PostClick(Button: TMPBtnType); dynamic;
```

C++

```
__fastcall PostClick(TMPBtnType Button);
```

Description

PostClick is the implementation method for the OnPostClick (see page 17228) event. By default, PostClick does nothing other than call any event handler attached to the media player's OnPostClick (see page 17228) event. PostClick can be overridden in descendant types to customize responses to clicks.

See Also

OnPostClick (see page 17228)

101.1.2.4.18 TMediaPlayer.Previous Method

Sets the current position to the beginning of the previous track if the position was at the beginning of a track when Previous was called.

Pascal

```
procedure Previous;
```

C++

```
__fastcall Previous();
```

Description

If the position is at the first track or somewhere other than the beginning of a track when Previous was called, Previous sets the current position to the beginning of the current track. If the device doesn't use tracks, Previous sets the current position to the beginning of the medium, which is specified in the Start (see page 17215) property. Previous is called when the Previous button on the media player control is clicked at runtime.

Upon completion, Previous stores a numerical error code in the Error (see page 17204) property and the corresponding error message in the ErrorMessage (see page 17205) property.

The Wait (see page 17221) property determines whether control is returned to the application before the Previous method has completed. The Notify (see page 17206) property determines whether Previous generates an OnNotify (see page 17228) event.

See Also

Error (see page 17204)

ErrorMessage (see page 17205)

Notify (see page 17206)

OnNotify (see page 17228)

Start (see page 17215)

Wait (see page 17221)

Adding Audio and/or Video Clips to an Application

101.1.2.4.19 TMediaPlayer.Resume Method

Resumes playing or recording the currently paused multimedia device.

Pascal

```
procedure Resume;
```

C++

```
__fastcall Resume();
```

Description

Resume is called when the Pause (see page 17189) button on the media player control is clicked at runtime, when the device is paused.

Upon completion, Resume stores a numerical error code in the Error (see page 17204) property, and the corresponding error message in the ErrorMessage (see page 17205) property.

The Wait (see page 17221) property determines whether control is returned to the application before the Resume method has completed. The Notify (see page 17206) property determines whether Resume generates an OnNotify (see page 17228) event.

See Also

Error (see page 17204)

ErrorMessage (see page 17205)

Notify (see page 17206)

OnNotify (see page 17228)

Wait (see page 17221)

101.1.2.4.20 TMediaPlayer.Rewind Method

Sets the current position to the beginning of the the medium, which is stored in the Start (see page 17215) property.

Pascal

```
procedure Rewind;
```

C++

```
__fastcall Rewind();
```

Description

Upon completion, Rewind stores a numerical error code in the Error (see page 17204) property, and the corresponding error message in the ErrorMessage (see page 17205) property.

The Wait (see page 17221) property determines whether control is returned to the application before the Rewind method has completed. The Notify (see page 17206) property determines whether Rewind generates an OnNotify (see page 17228) event.

See Also

Error (see page 17204)

ErrorMessage (see page 17205)

Notify (see page 17206)

OnNotify (see page 17228)

Start (see page 17215)

Wait (see page 17221)

101.1.2.4.21 TMediaPlayer.Save Method

Saves the currently loaded medium to the file specified in the FileName (see page 17205) property.

Pascal

```
procedure Save;
```

C++

```
__fastcall Save();
```

Description

Save is ignored for devices that don't use media stored in files (videodiscs, for example).

Upon completion, Save stores a numerical error code in the Error (see page 17204) property, and the corresponding error message in the ErrorMessage (see page 17205) property.

The Wait (see page 17221) property determines whether control is returned to the application before the Save method has completed. The Notify (see page 17206) property determines whether Save generates an OnNotify (see page 17228) event.

See Also

Error (see page 17204)

ErrorMessage (see page 17205)

FileName (see page 17205)

Notify (see page 17206)

OnNotify (see page 17228)

Wait (see page 17221)

101.1.2.4.22 TMediaPlayer.StartRecording Method

Begins recording from the current Position (see page 17213) or from the position specified in StartPos (see page 17216).

Pascal

```
procedure StartRecording;
```

C++

```
__fastcall StartRecording();
```

Description

StartRecording is called when the Record button on the media player control is clicked at runtime.

Upon completion, StartRecording stores a numerical error code in the Error (see page 17204) property and the corresponding error message in the ErrorMessage (see page 17205) property.

The Wait (see page 17221) property determines whether control is returned to the application before the StartRecording method has completed. The Notify (see page 17206) property determines whether StartRecording generates an OnNotify (see page 17228) event.

By default, the Notify (see page 17206) property becomes true, and the Wait (see page 17221) property becomes false upon completion of the StartRecording method. However, if these properties have been set to specific values prior to calling StartRecording, they remain unchanged.

See Also

Error (see page 17204)

ErrorMessage (see page 17205)

Notify (see page 17206)

OnNotify (see page 17228)

Position (see page 17213)

StartPos (see page 17216)

Wait (see page 17221)

101.1.2.4.23 TMediaPlayer.Step Method

Moves forward a number of frames (determined by the Frames (see page 17205) property) in the currently loaded medium.

Pascal

```
procedure Step;
```

C++

```
__fastcall Step();
```

Description

Step is called when the Step button on the media player control is clicked at runtime.

Upon completion, Step stores a numerical error code in the Error (see page 17204) property and the corresponding error message in the ErrorMessage (see page 17205) property.

The Wait (see page 17221) property determines whether control is returned to the application before the Step method has completed. The Notify (see page 17206) property determines whether Step generates an OnNotify (see page 17228) event.

See Also

Error (see page 17204)

ErrorMessage (see page 17205)

Frames (see page 17205)

Notify (see page 17206)

OnNotify (see page 17228)

Wait (see page 17221)

Adding Audio and/or Video Clips to an Application

101.1.2.4.24 TMediaPlayer.Stop Method

Halts playing or recording.

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

Stop is called when the Stop button on the media player control is clicked at runtime.

Upon completion, Stop stores a numerical error code in the Error (see page 17204) property, and the corresponding error message in the ErrorMessage (see page 17205) property.

The Wait (see page 17221) property determines whether control is returned to the application before the Stop method has completed. The Notify (see page 17206) property determines whether Stop generates an OnNotify (see page 17228) event.

See Also

Error (see page 17204)

ErrorMessage (see page 17205)

FileName (see page 17205)

Notify (see page 17206)

OnNotify (see page 17228)

Wait (see page 17221)

Adding Audio and/or Video Clips to an Application

101.1.2.4.25 TMediaPlayer.Updated Method

Signals that the media player has finished updating.

Pascal

```
procedure Updated; override;
```

C++

```
virtual __fastcall Updated();
```

Description

Updated calls the inherited Updated before it modifies the layout of the media player buttons and forces a repaint.

See Also

Updated (see page 2791)

101.1.2.5 TMediaPlayer Properties

The properties of the TMediaPlayer class are listed here.

Public Properties

	Name	Description
R	Capabilities (see page 17199)	Determines the capabilities of the open multimedia device.
R	DeviceID (see page 17201)	Specifies the device ID for the currently open multimedia device.
	DisplayRect (see page 17202)	Specifies a rectangular area in the control specified by the Display (see page 17202) property that is used to display output from a multimedia device.
	EndPos (see page 17204)	Specifies the position within the currently loaded medium at which to stop playing or recording.
R	Error (see page 17204)	Specifies the MCI error code returned by the media control method.
R	ErrorMessage (see page 17205)	Describes the error code stored in the Error (see page 17204) property.
	Frames (see page 17205)	Specifies the number of frames moved forward or backward.
R	Length (see page 17206)	Specifies the length of the medium in the open multimedia device.
R	Mode (see page 17206)	Indicates the state of the currently open multimedia device.
	Notify (see page 17206)	Determines whether an OnNotify (see page 17228) event is generated.
R	NotifyValue (see page 17207)	Indicates the result of the last media control method that requested a notification.
	Position (see page 17213)	Specifies the current position within the currently loaded medium.
R	Start (see page 17215)	Specifies the starting position within the currently loaded medium.
	StartPos (see page 17216)	Specifies the position within the currently loaded medium from which to begin playing or recording.
	TimeFormat (see page 17217)	Determines the format used to specify position information.
R	TrackLength (see page 17218)	Reports the length of the track.
R	TrackPosition (see page 17218)	Reports the starting position of the track.
R	Tracks (see page 17218)	Specifies the number of playable tracks on the open multimedia device.
	Wait (see page 17221)	Determines whether a media control method returns control to the application only after it has been completed.

Published Properties

	Name	Description
	Anchors (see page 17197)	Specifies how the control is anchored to its parent.
	AutoEnable (see page 17197)	Determines whether the media player automatically enables and disables individual buttons in the component.
	AutoOpen (see page 17198)	Determines if the media player is opened automatically when the application is run.
	AutoRewind (see page 17199)	Determines if the media player control rewinds before playing or recording.
	ColoredButtons (see page 17200)	Determines which buttons on the media player control have color.
	Constraints (see page 17200)	Specifies the size constraints for the control.
	DeviceType (see page 17201)	Specifies a multimedia device type to open the media player.
	Display (see page 17202)	Specifies the display window for a multimedia device that uses a window for output.
	Enabled (see page 17203)	Controls whether the control responds to mouse, keyboard, and timer events.
	EnabledButtons (see page 17203)	Determines which buttons on the media player are usable.
	FileName (see page 17205)	Specifies the opened or saved media file.

	ParentShowHint (see page 17208)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 17208)	Identifies the pop-up menu associated with the control.
	Shareable (see page 17213)	Determines whether more than one application can share a multimedia device.
	ShowHint (see page 17214)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 17216)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 17217)	Determines if the user can tab to a control.
	Visible (see page 17219)	Determines whether the component appears onscreen.
	VisibleButtons (see page 17220)	Indicates which of the buttons on the media player are visible.

101.1.2.5.1 TMediaPlayer.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

101.1.2.5.2 TMediaPlayer.AutoEnable Property

Determines whether the media player automatically enables and disables individual buttons in the component.

Pascal

```
property AutoEnable: Boolean;
```

C++

```
__property Boolean AutoEnable;
```

Description

Use AutoEnable to automatically enable or disable the control buttons on the media player. If AutoEnable is true, the media player automatically enables or disables its control buttons. The media player determines which buttons to enable or disable by the current mode specified in the Mode (see page 17206) property, and the current multimedia device type specified in the DeviceType (see page 17201) property.

AutoEnable overrides the EnabledButtons (see page 17203) property. The buttons enabled or disabled automatically by the media player supersede any buttons enabled or disabled with EnabledButtons (see page 17203).

If AutoEnable is false, the media player does not enable or disable buttons. The buttons must enabled or disabled with the EnabledButtons (see page 17203) property.

The following table shows whether buttons are automatically enabled or disabled for each device mode:

Button	Play (see page 17190)	Record	Pause (see page 17189)	Stop (see page 17195)	Not Open (see page 17188)
Back (see page 17182)	Enabled (see page 17203)	Enabled (see page 17203)	Enabled (see page 17203)	Enabled (see page 17203)	Disabled
Eject (see page 17184)	Enabled (see page 17203)	Enabled (see page 17203)	Enabled (see page 17203)	Enabled (see page 17203)	Disabled
Next (see page 17187)	Enabled (see page 17203)	Enabled (see page 17203)	Enabled (see page 17203)	Enabled (see page 17203)	Disabled
Pause (see page 17189)	Enabled (see page 17203)	Enabled (see page 17203)	Enabled (see page 17203)	Disabled	Disabled
Play (see page 17190)	Disabled	Disabled	Enabled (see page 17203)	Enabled (see page 17203)	Disabled
Prev	Enabled (see page 17203)	Enabled (see page 17203)	Enabled (see page 17203)	Enabled (see page 17203)	Disabled
Record	Disabled	Disabled	Enabled (see page 17203)	Enabled (see page 17203)	Disabled
Step (see page 17194)	Enabled (see page 17203)	Enabled (see page 17203)	Enabled (see page 17203)	Enabled (see page 17203)	Disabled
Stop (see page 17195)	Enabled (see page 17203)	Enabled (see page 17203)	Disabled	Disabled	Disabled

See Also

AutoButtonSet (see page 17181)

DeviceType (see page 17201)

EnabledButtons (see page 17203)

MMNotify (see page 17187)

Mode (see page 17206)

Adding Audio and/or Video Clips to an Application

101.1.2.5.3 TMediaPlayer.AutoOpen Property

Determines if the media player is opened automatically when the application is run.

Pascal

```
property AutoOpen: Boolean;
```

C++

```
__property Boolean AutoOpen;
```

Description

Use AutoOpen to have the media player automatically open at runtime.

If AutoOpen is true, the media player attempts to open the multimedia device specified by the DeviceType (see page 17201) property (or FileName (see page 17205) if DeviceType (see page 17201) is dtAutoSelect) when the form containing the media player component is created at runtime.

If AutoOpen is false, the device must be opened with a call to the Open (see page 17188) method.

If an error occurs when opening the device, an exception of type EMCIDeviceError (see page 17167) exception is raised that contains the error message. Upon completion, a numerical error code is stored in the Error (see page 17204) property, and the corresponding error message is stored in the ErrorMessage (see page 17205) property.

The Wait (see page 17221) property determines whether control is returned to the application before opening the multimedia device. The Notify (see page 17206) property determines whether opening the device generates an OnNotify (see page 17228) event.

See Also

DeviceType (see page 17201)

EMCIDeviceError (see page 17167)

Error (see page 17204)

ErrorMessage (see page 17205)

FileName (see page 17205)

Notify (see page 17206)

OnNotify (see page 17228)

Wait (see page 17221)

Adding Audio and/or Video Clips to an Application

101.1.2.5.4 TMediaPlayer.AutoRewind Property

Determines if the media player control rewinds before playing or recording.

Pascal

```
property AutoRewind: Boolean;
```

C++

```
__property Boolean AutoRewind;
```

Description

If AutoRewind is true and the current position is at the end of the medium, Play (see page 17190) or StartRecording (see page 17194) moves the current position to the beginning of the medium before playing or recording. If AutoRewind is false, the user must click the Prev button or write code to call Previous (see page 17191) to move to the beginning.

If values have been assigned to StartPos (see page 17216) or EndPos (see page 17204) or if the multimedia device uses tracks, AutoRewind has no effect on playing or recording. When calling Play (see page 17190) or StartRecording (see page 17194), the current position remains at the end of the medium.

See Also

Previous (see page 17191)

101.1.2.5.5 TMediaPlayer.Capabilities Property

Determines the capabilities of the open multimedia device.

Pascal

```
property Capabilities: TMPDevCapsSet;
```

C++

```
__property TMPDevCapsSet Capabilities;
```

Description

The various capabilities specified in Capabilities are determined when the device is opened with the Open (see page 17188) method.

Note: Currently, there is no way to check whether a device can step forward or backward. Capabilities includes mpCanStep only if the device type (specified in the DeviceType (see page 17201) property) is Animation, AVI Video, Digital Video, Overlay, or VCR.

See Also

DeviceType (see page 17201)

Open (see page 17188)

101.1.2.5.6 TMediaPlayer.ColoredButtons Property

Determines which buttons on the media player control have color.

Pascal

```
property ColoredButtons: TButtonSet;
```

C++

```
__property TButtonSet ColoredButtons;
```

Description

ColoredButtons controls which buttons on the media player are colored.

If a button is not colored with ColoredButtons, it appears in black-and-white when visible. All media player control buttons are colored by default.

See Also

EnabledButtons (see page 17203)

VisibleButtons (see page 17220)

101.1.2.5.7 TMediaPlayer.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the

title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

101.1.2.5.8 TMediaPlayer.DeviceID Property

Specifies the device ID for the currently open multimedia device.

Pascal

```
property DeviceID: Word;
```

C++

```
__property Word DeviceID;
```

Description

The value of DeviceID is determined when a device is opened with the Open (see page 17188) method. If no device is open, DeviceID is 0.

See Also

Open (see page 17188)

101.1.2.5.9 TMediaPlayer.DeviceType Property

Specifies a multimedia device type to open the media player.

Pascal

```
property DeviceType: TMPDeviceTypes;
```

C++

```
__property TMPDeviceTypes DeviceType;
```

Description

DeviceType specifies a multimedia device type to open with the Open (see page 17188) method. The default is dtAutoSelect.

If DeviceType is dtAutoSelect, the device type is determined by the file extension specified in the FileName (see page 17205) property. If no device type is associated with the extension, the correct device type must be explicitly specified by setting DeviceType to a value other than dtAutoSelect.

A multimedia device is typically associated with an appropriate file name extension when the device is installed. Associations are specified in the registry or SYSTEM.INI file. See the documentation for the specific device for instructions about how to associate file name extensions with the device.

See Also

FileName (see page 17205)

Open (see page 17188)

Adding Audio and/or Video Clips to an Application

101.1.2.5.10 TMediaPlayer.Display Property

Specifies the display window for a multimedia device that uses a window for output.

Pascal

```
property Display: TWinControl;
```

C++

```
__property TWinControl * Display;
```

Description

To use Display, assign the name of a windowed control such as a form or panel to Display to display output in that control.

The default value of Display is nil (Delphi) or NULL (C++), meaning that the device creates its own window to display output. Also, by freeing the control assigned to Display after the device has been opened, video output will be in its own default window.

Examples of multimedia devices that use a window to display output are Animation, AVI Video, Digital Video, Overlay, and VCR.

See Also

DisplayRect (see page 17202)

Adding Audio and/or Video Clips to an Application

101.1.2.5.11 TMediaPlayer.DisplayRect Property

Specifies a rectangular area in the control specified by the Display (see page 17202) property that is used to display output from a multimedia device.

Pascal

```
property DisplayRect: TRect;
```

C++

```
__property TRect DisplayRect;
```

Description

To display output in a specific rectangle area on a form assign a TRect record to DisplayRect. The Rect function can be used to create a TRect record.

Media that use a rectangle to display output usually perform best if the default DisplayRect size is used. To set DisplayRect to the default size, position the rectangle in the upper left corner and use 0, 0 for the lower right corner.

DisplayRect is ignored if Display (see page 17202) is nil (Delphi) or NULL (C++).

Examples of multimedia devices that use a window to display output are Animation, AVI Video, Digital Video, Overlay, and VCR.

Note: DisplayRect can be set only after the media device is opened.

See Also

Display (see page 17202)

Open (see page 17188)

101.1.2.5.12 TMediaPlayer.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

101.1.2.5.13 TMediaPlayer.EnabledButtons Property

Determines which buttons on the media player are usable.

Pascal

```
property EnabledButtons: TButtonSet;
```

C++

```
__property TButtonSet EnabledButtons;
```

Description

EnabledButtons controls which buttons on the media player are enabled and usable.

An enabled button is colored and usable. A disabled button is dimmed and not usable. If a button is not enabled with EnabledButtons, it is disabled. By default, all buttons are enabled.

If the AutoEnable (see page 17197) property is true, AutoEnable (see page 17197) supersedes EnabledButtons. The buttons automatically enabled or disabled by the media player override any buttons enabled or disabled with the EnabledButtons property.

See Also

ColoredButtons (see page 17200)

VisibleButtons (see page 17220)

Adding Audio and/or Video Clips to an Application

101.1.2.5.14 TMediaPlayer.EndPos Property

Specifies the position within the currently loaded medium at which to stop playing or recording.

Pascal

```
property EndPos: Longint;
```

C++

```
__property Longint EndPos;
```

Description

EndPos is specified using the current time format, which is specified in the TimeFormat (see page 17217) property.

The EndPos property affects only the next Play (see page 17190) or StartRecording (see page 17194) method called after setting EndPos. Reset EndPos to affect any subsequent calls to Play (see page 17190) or StartRecording (see page 17194).

See Also

Play (see page 17190)

StartRecording (see page 17194)

TimeFormat (see page 17217)

101.1.2.5.15 TMediaPlayer.Error Property

Specifies the MCI error code returned by the media control method.

Pascal

```
property Error: Longint;
```

C++

```
__property Longint Error;
```

Description

Error specifies the MCI error code returned by the most recent media control method (Back (see page 17182), Close (see page 17183), Eject (see page 17184), Next (see page 17187), Open (see page 17188), Pause (see page 17189), PauseOnly (see page 17190), Play (see page 17190), Previous (see page 17191), StartRecording (see page 17194), Resume (see page 17192), Rewind (see page 17193), Step (see page 17194), or Stop (see page 17195)). The ErrorMessage (see page 17205) property describes the Error.

The error code returned by media control methods are the same error code returned by the mciSendCommand function. The message describing the error code is stored in the ErrorMessage (see page 17205) property.

The value of Error is zero if the most recent media control method didn't cause an error. If a method results in an error, a value other than zero is stored in Error. If the error occurs during the opening of the device, an EMCIDeviceError (see page 17167) exception occurs.

See Also

EMCIDeviceError (see page 17167)

ErrorMessage (see page 17205)

Play (see page 17190)

StartRecording (see page 17194)

TimeFormat (see page 17217)

101.1.2.5.16 TMediaPlayer.ErrorMessage Property

Describes the error code stored in the Error (see page 17204) property.

Pascal

```
property ErrorMessage: string;
```

C++

```
__property AnsiString ErrorMessage;
```

Description

ErrorMessage specifies the error message that describes the error code returned from the most recent media control method (Back (see page 17182), Close (see page 17183), Eject (see page 17184), Next (see page 17187), Open (see page 17188), Pause (see page 17189), PauseOnly (see page 17190), Play (see page 17190), Previous (see page 17191), StartRecording (see page 17194), Resume (see page 17192), Rewind (see page 17193), Step (see page 17194), or Stop (see page 17195)).

The error code described by the message is stored in the Error (see page 17204) property.

See Also

Error (see page 17204)

101.1.2.5.17 TMediaPlayer.FileName Property

Specifies the opened or saved media file.

Pascal

```
property FileName: string;
```

C++

```
__property AnsiString FileName;
```

Description

FileName specifies the media file to be opened by the Open (see page 17188) method, or the file to save by the Save (see page 17193) method.

At design time, use a file open dialog box to specify the FileName property by clicking the ellipsis button (...) in the Object Inspector.

See Also

Open (see page 17188)

Save (see page 17193)

Adding Audio and/or Video Clips to an Application

101.1.2.5.18 TMediaPlayer.Frames Property

Specifies the number of frames moved forward or backward.

Pascal

```
property Frames: Longint;
```

C++

```
__property Longint Frames;
```

Description

Frames specifies the number of frames the Step (see page 17194) method steps forward or the Back (see page 17182) method steps backward. Frames defaults to ten percent of the length of the currently loaded medium, which is specified by the Length (see page 17206) property.

The definition of a frame varies by multimedia device. For display media, a frame is one still image.

See Also

Back (see page 17182)

Length (see page 17206)

Step (see page 17194)

101.1.2.5.19 TMediaPlayer.Length Property

Specifies the length of the medium in the open multimedia device.

Pascal

```
property Length: Longint;
```

C++

```
__property Longint Length;
```

Description

Length is specified using the current time format, which is specified by the TimeFormat (see page 17217) property.

See Also

TimeFormat (see page 17217)

101.1.2.5.20 TMediaPlayer.Mode Property

Indicates the state of the currently open multimedia device.

Pascal

```
property Mode: TMPModes;
```

C++

```
__property TMPModes Mode;
```

Description

Mode specifies the current state or mode of the currently open multimedia device.

The TMPModes (see page 17233) type defines the modes for a multimedia device used with a TMediaPlayer (see page 17167).

See Also

OnNotify (see page 17228)

101.1.2.5.21 TMediaPlayer.Notify Property

Determines whether an OnNotify (see page 17228) event is generated.

Pascal

```
property Notify: Boolean;
```

C++

```
__property Boolean Notify;
```

Description

Notify determines whether the next call to a media control method (Back (see page 17182), Close (see page 17183), Eject (see page 17184), Next (see page 17187), Open (see page 17188), Pause (see page 17189), PauseOnly (see page 17190), Play (see page 17190), Previous (see page 17191), StartRecording (see page 17194), Resume (see page 17192), Rewind (see page 17193), Step (see page 17194), or Stop (see page 17195)) generates an OnNotify (see page 17228) event when the method has completed.

If Notify is true, the next media control method generates OnNotify (see page 17228) event upon completion and stores the notification message in the NotifyValue (see page 17207) property. If Notify is false, the method does not generate an OnNotify (see page 17228) event and NotifyValue (see page 17207) remains unchanged.

Notify affects only the next call to a media control method. After an OnNotify (see page 17228) event, Notify must be reset to affect any subsequent media control methods.

By default, Play (see page 17190) and StartRecording (see page 17194) function as if Notify is true. Set Notify to false before calling Play (see page 17190) or StartRecording (see page 17194) to prevent an OnNotify (see page 17228) event from being generated when playing or recording has finished. By default, all other media control methods function as if Notify is false.

Tip: Set Notify to true if the next media control is expected to take a long time, so the application is notified when the media control method has completed. If Notify is set to true, it is recommended to set Wait (see page 17221) to false so that control returns to the application before the media control method is finished.

Note: When trying to resume a device that doesn't support Resume (see page 17192), the device is resumed as if the Play (see page 17190) method was called. If Notify was assigned true before calling Resume (see page 17192) (or any other media control method), Notify doesn't affect the call to Resume (see page 17192). Resume (see page 17192) does not generate an OnNotify (see page 17228) event upon completion, and NotifyValue (see page 17207) remains unchanged.

See Also

NotifyValue (see page 17207)

OnNotify (see page 17228)

Resume (see page 17192)

Wait (see page 17221)

101.1.2.5.22 TMediaPlayer.NotifyValue Property

Indicates the result of the last media control method that requested a notification.

Pascal

```
property NotifyValue: TMPNotifyValues;
```

C++

```
__property TMPNotifyValues NotifyValue;
```

Description

NotifyValue reports the result of the last media control method (Back (see page 17182), Close (see page 17183), Eject (

see page 17184), Next (see page 17187), Open (see page 17188), Pause (see page 17189), PauseOnly (see page 17190), Play (see page 17190), Previous (see page 17191), StartRecording (see page 17194), Resume (see page 17192), Rewind (see page 17193), Step (see page 17194), or Stop (see page 17195)) that requested a notification.

To request notification, set Notify (see page 17206) to true before calling a media control method.

See Also

OnNotify (see page 17228)

Notify (see page 17206)

101.1.2.5.23 TMediaPlayer.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

101.1.2.5.24 TMediaPlayer.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup

property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
```

```
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
```

```
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
```

```
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
```

```
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

101.1.2.5.25 TMediaPlayer.Position Property

Specifies the current position within the currently loaded medium.

Pascal

```
property Position: Longint;
```

C++

```
__property Longint Position;
```

Description

The value of Position is specified according to the current time format, which is specified in the TimeFormat (see page 17217) property.

Position defaults to the beginning of the medium. If the medium supports multiple tracks, Position defaults to the beginning of the first track.

See Also

TimeFormat (see page 17217)

101.1.2.5.26 TMediaPlayer.Shareable Property

Determines whether more than one application can share a multimedia device.

Pascal

```
property Shareable: Boolean;
```

C++

```
__property Boolean Shareable;
```

Description

If Shareable is false, no other components or applications can access the device. If Shareable is true, more than one component or application can access the device. Shareable defaults to false.

Be sure to set Shareable before opening a device.

Some devices are not shareable. If Shareable is set to true for a device that isn't shareable, the Open (see page 17188)

method fails and the error code is returned to the Error (see page 17204) property.

See Also

Error (see page 17204)

Open (see page 17188)

101.1.2.5.27 TMediaPlayer.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

101.1.2.5.28 TMediaPlayer.Start Property

Specifies the starting position within the currently loaded medium.

Pascal

```
property Start: Longint;
```

C++

```
__property Longint Start;
```

Description

Start is the beginning of the medium for devices that don't use tracks, or the beginning of the first track for devices that use tracks. Start is defined when a multimedia device is opened with the Open (see page 17188) method. Start is specified according to the current time format, which is stored in the TimeFormat (see page 17217) property. Start is read-only at runtime and is unavailable at design time.

See Also

Open (see page 17188)

TimeFormat (see page 17217)

101.1.2.5.29 TMediaPlayer.StartPos Property

Specifies the position within the currently loaded medium from which to begin playing or recording.

Pascal

```
property StartPos: Longint;
```

C++

```
__property Longint StartPos;
```

Description

StartPos is specified using the current time format, which is specified in the TimeFormat (see page 17217) property.

The StartPos property affects only the next Play (see page 17190) or StartRecording (see page 17194) method called after setting StartPos. Reset must be set to StartPos to affect any subsequent calls to Play (see page 17190) or StartRecording (see page 17194).

StartPos does not affect the current position of the medium (specified in the Position (see page 17213) property) until the next Play (see page 17190) or StartRecording (see page 17194) method is called.

See Also

Play (see page 17190)

StartRecording (see page 17194)

TimeFormat (see page 17217)

101.1.2.5.30 TMediaPlayer.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

101.1.2.5.31 TMediaPlayer.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

101.1.2.5.32 TMediaPlayer.TimeFormat Property

Determines the format used to specify position information.

Pascal

```
property TimeFormat: TMPTimeFormats;
```

C++

```
__property TMPTimeFormats TimeFormat;
```

Description

TimeFormat determines how the StartPos (see page 17216), Length (see page 17206), Position (see page 17213), Start (see page 17215), and EndPos (see page 17204) properties are interpreted. For example, if Position (see page 17213) is 180 and TimeFormat is tfMilliseconds, the current position is 180 milliseconds into the medium. If Position (see page 17213) is 180 and TimeFormat is tfMSF, the current position is 180 minutes into the medium.

Not all formats are supported by every device. When trying to set an unsupported format, the assignment is ignored.

The current timing information is always passed in a 4-byte integer. In some formats, the timing information returned is not really one integer, but single bytes of information packed in the long integer.

See Also

EndPos (see page 17204)

Length (see page 17206)

Position (see page 17213)

Start (see page 17215)

StartPos (see page 17216)

101.1.2.5.33 TMediaPlayer.TrackLength Property

Reports the length of the track.

Pascal

```
property TrackLength [TrackNum: Integer]: Longint;
```

C++

```
__property Longint TrackLength[int TrackNum];
```

Description

TrackLength reports the length of the track specified by the TrackNum index.

The value of TrackLength is specified according to the current time format, which is specified in the TimeFormat (see page 17217) property.

See Also

TimeFormat (see page 17217)

Tracks (see page 17218)

101.1.2.5.34 TMediaPlayer.TrackPosition Property

Reports the starting position of the track.

Pascal

```
property TrackPosition [TrackNum: Integer]: Longint;
```

C++

```
__property Longint TrackPosition[int TrackNum];
```

Description

TrackPosition reports the starting position of the track specified by the TrackNum index.

The value of TrackPosition is specified according to the current time format, which is specified in the TimeFormat (see page 17217) property.

See Also

TimeFormat (see page 17217)

Tracks (see page 17218)

101.1.2.5.35 TMediaPlayer.Tracks Property

Specifies the number of playable tracks on the open multimedia device.

Pascal

```
property Tracks: Longint;
```

C++

```
__property Longint Tracks;
```

Description

Tracks indicates how many of playable tracks are available on the open multimedia device.

Tracks is undefined for devices that don't use tracks.

See Also

TrackLength (see page 17218)

TrackPosition (see page 17218)

Adding Audio and/or Video Clips to an Application

101.1.2.5.36 TMediaPlayer.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
```

```
  edit control placed somewhere on the client area of the
  sheet.  Unchecking the checkbox will cause the client area
  (not the tab area) of the Tab Control to become invisible.
  }
  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    PageControl1.ActivePage.Visible := CheckBox1.Checked;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    CheckBox1.Checked := PageControl1.ActivePage.Visible;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

101.1.2.5.37 TMediaPlayer.VisibleButtons Property

Indicates which of the buttons on the media player are visible.

Pascal

```
property VisibleButtons: TButtonSet;
```

C++

```
__property TButtonSet VisibleButtons;
```

Description

VisibleButtons determines which buttons on the media player that are visible. If a button is not made visible with VisibleButtons, it does not appear on the media player control. By default, all buttons are visible when a media player component is added to a form.

See Also

ColoredButtons (see page 17200)

EnabledButtons (see page 17203)

101.1.2.5.38 TMediaPlayer.Wait Property

Determines whether a media control method returns control to the application only after it has been completed.

Pascal

```
property Wait: Boolean;
```

C++

```
__property Boolean Wait;
```

Description

Wait determines whether a media control method (Back (see page 17182), Close (see page 17183), Eject (see page 17184), Next (see page 17187), Open (see page 17188), Pause (see page 17189), PauseOnly (see page 17190), Play (see page 17190), Previous (see page 17191), StartRecording (see page 17194), Resume (see page 17192), Rewind (see page 17193), Step (see page 17194), or Stop (see page 17195)) returns control to the application only after it has been completed.

Wait is unavailable at design time.

If Wait is true, the media player component waits until the next media control method has completed before returning control to the application. If Wait is false, the application won't wait for the next media control method to finish before continuing.

Wait affects only the next media control method called after setting Wait. Wait must be reset to affect any subsequent call to a media control method.

By default, Play (see page 17190) and StartRecording (see page 17194) function as if Wait is false. Wait must be set to true before calling Play (see page 17190) or StartRecording (see page 17194) to prevent control from returning to the application before playing or recording has finished. By default, all other media control methods function as if Wait is true.

Note: Wait is usually set to false only if the next media control is expected to take a long time, so that the application can execute other code before the media control method has completed. If Wait is set to false, you should set Notify (see page 17206) to true so the application is notified when the media control method completes.

See Also

Notify (see page 17206)

101.1.2.6 TMediaPlayer Events

The events of the TMediaPlayer class are listed here.

Published Events

	Name	Description
	OnClick (see page 17222)	Occurs when the user presses and releases the mouse button while the mouse pointer is over one of the control buttons, or when the user presses Spacebar while the media player control has focus.
	OnContextPopup (see page 17222)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnEnter (see page 17223)	Occurs when a control receives the input focus.
	OnExit (see page 17226)	Occurs when the input focus shifts away from one control to another.
	OnMouseEnter (see page 17227)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 17227)	Occurs when the user moves the mouse outside of a control.
	OnNotify (see page 17228)	Occurs upon the completion of a media control method.
	OnPostClick (see page 17228)	Occurs after the OnClick (see page 17222) event handler is called.

101.1.2.6.1 TMediaPlayer.OnClick Event

Occurs when the user presses and releases the mouse button while the mouse pointer is over one of the control buttons, or when the user presses Spacebar while the media player control has focus.

Pascal

```
property OnClick: EMPNotify;
```

C++

```
__property EMPNotify OnClick;
```

Description

When the media player control has focus and the Spacebar is pressed, the user can use the Left Arrow or Right Arrow keys to select which control button to click.

The EMPNotify (see page 17237) type is a method pointer that is called when an OnClick event for a TMediaPlayer (see page 17167) components occurs. The Button argument can be one of the following values: btBack, btEject, btNext, btPause, btPlay, btPrev, btRecord, btStep, or btStop.

The default value of the event handler's DoDefault argument is true. If DoDefault is true, the media player control calls the method that corresponds to the clicked button. For example, if the user clicks the Play (see page 17190) button (btPlay), the Play (see page 17190) method is called.

If DoDefault is false, the user must write code that executes when a media player control button is clicked in the OnClick event handler. The following table lists the default methods corresponding to the media player control buttons:

Control button	Button value	Method called
Play (see page 17190)	btPlay	Play (see page 17190)
Record	btRecord	StartRecording (see page 17194)
Stop (see page 17195)	btStop	Stop (see page 17195)
Next (see page 17187)	btNext	Next (see page 17187)
Prev	btPrev	Previous (see page 17191)
Step (see page 17194)	btStep	Step (see page 17194)
Back (see page 17182)	btBack	Back (see page 17182)
Pause (see page 17189)	btPause	Pause (see page 17189)
Eject (see page 17184)	btEject	Eject (see page 17184)

See Also

Click (see page 17182)

101.1.2.6.2 TMediaPlayer.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

101.1.2.6.3 TMediaPlayer.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
```

```
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
```

```
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

101.1.2.6.4 TMediaPlayer.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

101.1.2.6.5 TMediaPlayer.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

101.1.2.6.6 TMediaPlayer.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse

outside of a control when the parent top-level form is not active.

101.1.2.6.7 TMediaPlayer.OnNotify Event

Occurs upon the completion of a media control method.

Pascal

```
property OnNotify: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnNotify;
```

Description

OnNotify occurs upon the completion of a media control method (Back (see page 17182), Close (see page 17183), Eject (see page 17184), Next (see page 17187), Open (see page 17188), Pause (see page 17189), PauseOnly (see page 17190), Play (see page 17190), Previous (see page 17191), Resume (see page 17192), Rewind (see page 17193), StartRecording (see page 17194), Step (see page 17194), or Stop (see page 17195)) when the Notify (see page 17206) property is set to true before the call to the media control method. After an OnNotify event, the Notify (see page 17206) property must be reset to true for the next OnNotify event to occur.

See Also

DoNotify (see page 17184)

MMNotify (see page 17187)

Notify (see page 17206)

101.1.2.6.8 TMediaPlayer.OnPostClick Event

Occurs after the OnClick (see page 17222) event handler is called.

Pascal

```
property OnPostClick: EMPPostNotify;
```

C++

```
__property EMPPostNotify OnPostClick;
```

Description

OnPostClick is generated after the code of the OnClick (see page 17222) event handler has been called. If Wait (see page 17221) is true when the media player was clicked, OnPostClick won't be called until the completion of the OnClick (see page 17222) code. If Wait (see page 17221) is false, control can return to the application before completion of the OnClick (see page 17222) code; therefore, the OnPostClick event may occur before the actions initiated by the OnClick (see page 17222) event have completed.

For example, if the user clicks the Play (see page 17190) button and the DoDefault parameter of the OnClick (see page 17222) event handler for the media player is true, the media is played. If the media is long enough, it will still be playing when the OnPostClick event is generated if Wait (see page 17221) is true. If Wait (see page 17221) is false, however, OnPostClick won't occur until the media has finished playing.

See Also

OnClick (see page 17222)

PostClick (see page 17191)

Wait (see page 17221)

101.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TMPBtnType (see page 17229)	TMPBtnType defines the possible buttons in a TMediaPlayer (see page 17167) component and TButtonSet (see page 17238) represents a set of those buttons.
	TMPDevCaps (see page 17231)	TMPDevCaps defines the capabilities of a TMediaPlayer (see page 17167) component and TMPDevCapsSet (see page 17238) represents a set of those capabilities.
	TMPDeviceTypes (see page 17231)	TMPDeviceTypes identifies a multimedia device.
	TMPGlyph (see page 17233)	This is record MPlayer.TMPGlyph.
	TMPModes (see page 17233)	TMPModes identifies the state of a TMediaPlayer (see page 17167) component.
	TMPNotifyValues (see page 17234)	TMPNotifyValues identifies the types of notifications a media device can give a TMediaPlayer (see page 17167) component.
	TMPTimeFormats (see page 17235)	TMPTimeFormats indicates the format a media player uses to indicate position.

Records

	Name	Description
	TMPButton (see page 17230)	This is record MPlayer.TMPButton.

101.2.1 MPlayer.TMPBtnType Enumeration

TMPBtnType defines the possible buttons in a TMediaPlayer (see page 17167) component and TButtonSet (see page 17238) represents a set of those buttons.

Pascal

```
TMPBtnType = (
  btPlay,
  btPause,
  btStop,
  btNext,
  btPrev,
  btStep,
  btBack,
  btRecord,
  btEject
);
```

C++

```
enum TMPBtnType {
  btPlay,
  btPause,
  btStop,
  btNext,
  btPrev,
  btStep,
  btBack,
  btRecord,
  btEject
```

```
};
```

File

MPlayer

Description

TMPBtnType represents the type of a media player button. TButtonSet (see page 17238) is a set of button types for the media player.

Each button type has an associated action on the media player. The following table lists the button types and their associated actions:

Value	Action
btPlay	Plays the media player.
btRecord	Starts recording.
btStop	Stops playing or recording.
btNext	Skips to the next track, or to the end if the medium doesn't use tracks.
btPrev	Skips to the previous track, or to the beginning if the medium doesn't use tracks.
btStep	Moves forward a number of frames.
btBack	Moves backward a number of frames.
btPause	Pauses playing or recording. If already paused when clicked, resumes playing or recording.
btEject	Ejects the medium.

See Also

TBitBtn

TSpeedButton

101.2.2 MPlayer.TMPButton Record

Pascal

```
TMPButton = record
  Visible: Boolean;
  Enabled: Boolean;
  Colored: Boolean;
  Auto: Boolean;
  Bitmaps: array[TMPGlyph] of TBitmap;
end;
```

C++

```
struct TMPButton {
  Boolean Visible;
  Boolean Enabled;
  Boolean Colored;
  Boolean Auto;
  array[TMPGlyph] of TBitmap Bitmaps;
};
```

File

MPlayer

Description

This is record MPlayer.TMPButton.

101.2.3 MPlayer.TMPDevCaps Enumeration

TMPDevCaps defines the capabilities of a TMediaPlayer (see page 17167) component and TMPDevCapsSet (see page 17238) represents a set of those capabilities.

Pascal

```
TMPDevCaps = (
  mpCanStep,
  mpCanEject,
  mpCanPlay,
  mpCanRecord,
  mpUsesWindow
);
```

C++

```
enum TMPDevCaps {
  mpCanStep,
  mpCanEject,
  mpCanPlay,
  mpCanRecord,
  mpUsesWindow
};
```

File

MPlayer

Description

TMPDevCaps represents the capabilities of a media player control. TMPDevCapsSet (see page 17238) is a set of TMPDevCaps values.

The following table lists the possible TMPDevCaps values:

Value	Capability
mpCanEject	Can eject media
mpCanPlay	Can play media
mpCanRecord	Can record media
mpCanStep	Can step forward or backward within media
mpUsesWindow	Uses a window for displaying output

See Also

TMediaPlayer

101.2.4 MPlayer.TMPDeviceTypes Enumeration

TMPDeviceTypes identifies a multimedia device.

Pascal

```
TMPDeviceTypes = (
  dtAutoSelect,
  dtAVIVideo,
  dtCDAudio,
  dtDAT,
  dtDigitalVideo,
  dtMMMovie,
  dtOther,
  dtOverlay,
  dtScanner,
  dtSequencer,
  dtVCR,
  dtVideodisc,
  dtWaveAudio
);
```

C++

```
enum TMPDeviceTypes {
  dtAutoSelect,
  dtAVIVideo,
  dtCDAudio,
  dtDAT,
  dtDigitalVideo,
  dtMMMovie,
  dtOther,
  dtOverlay,
  dtScanner,
  dtSequencer,
  dtVCR,
  dtVideodisc,
  dtWaveAudio
};
```

File

MPlayer

Description

TMPDeviceTypes type identifies the multimedia device types that can be opened by a TMediaPlayer (see page 17167) component.

The following table lists the possible values:

Value	Meaning
dtAutoSelect	the device type is determined by the file extension.
dtAVIVideo	AVI clip
dtCDAudio	Audio CD recording
dtDAT	DAT file
dtDigitalVideo	Digital video recording
dtMMMovie	MM recording
dtOther	media type not identified by other categories
dtOverlay	Overlay
dtScanner	scanner
dtSequencer	sequencer
dtVCR	VCR

dtVideodisc	Videodisc
dtWaveAudio	.wav file

See Also

TMediaPlayer

101.2.5 MPlayer.TMPGlyph Enumeration

Pascal

```
TMPGlyph = (
  mgEnabled,
  mgDisabled,
  mgColored
);
```

C++

```
enum TMPGlyph {
  mgEnabled,
  mgDisabled,
  mgColored
};
```

File

MPlayer

Description

This is record MPlayer.TMPGlyph.

101.2.6 MPlayer.TMPModes Enumeration

TMPModes identifies the state of a TMediaPlayer (see page 17167) component.

Pascal

```
TMPModes = (
  mpNotReady,
  mpStopped,
  mpPlaying,
  mpRecording,
  mpSeeking,
  mpPaused,
  mpOpen
);
```

C++

```
enum TMPModes {
  mpNotReady,
  mpStopped,
  mpPlaying,
  mpRecording,
  mpSeeking,
  mpPaused,
  mpOpen
```

```
};
```

File

MPlayer

Description

TMPModes type identifies the state of a media player. The following table lists the possible values:

Value	Mode
mpNotReady	Not ready
mpStopped	Stopped
mpPlaying	Playing
mpRecording	Recording
mpSeeking	Seeking
mpPaused	Paused
mpOpen	Open

See Also

TMediaPlayer

101.2.7 MPlayer.TMPNotifyValues Enumeration

TMPNotifyValues identifies the types of notifications a media device can give a TMediaPlayer (see page 17167) component.

Pascal

```
TMPNotifyValues = (
  nvSuccessful,
  nvSuperseded,
  nvAborted,
  nvFailure
);
```

C++

```
enum TMPNotifyValues {
  nvSuccessful,
  nvSuperseded,
  nvAborted,
  nvFailure
};
```

File

MPlayer

Description

The TMPNotifyValues type defines the notification values for a multimedia device used with a TMediaPlayer (see page 17167). The possible values for the TMPNotifyValues type are as follows.

Value	Meaning
nvSuccessful	Command completed successfully
nvSuperseded	Command was superseded by another command

nvAborted	Command was aborted by the user
nvFailure	Command failed

See Also

TMediaPlayer

101.2.8 MPlayer.TMPTimeFormats Enumeration

TMPTimeFormats indicates the format a media player uses to indicate position.

Pascal

```
TMPTimeFormats = (
  tfMilliseconds,
  tfHMS,
  tfMSF,
  tfFrames,
  tfSMPTE24,
  tfSMPTE25,
  tfSMPTE30,
  tfSMPTE30Drop,
  tfBytes,
  tfSamples,
  tfTMSF
);
```

C++

```
enum TMPTimeFormats {
  tfMilliseconds,
  tfHMS,
  tfMSF,
  tfFrames,
  tfSMPTE24,
  tfSMPTE25,
  tfSMPTE30,
  tfSMPTE30Drop,
  tfBytes,
  tfSamples,
  tfTMSF
};
```

File

MPlayer

Description

The TMPTimeFormats type defines the time formats for a multimedia device used with a TMediaPlayer (see page 17167). The following table lists the possible values for the TMPTimeFormats type:

Value	Time format
tfMilliseconds	Milliseconds are stored as a 4-byte integer variable.
tfHMS	Hours, minutes, and seconds packed into a 4-byte integer. From least significant to most significant byte, the data values are
Hours (least significant byte)	
Minutes	

Seconds	
Unused (most significant byte)	
tfMSF	Minutes, seconds, and frames packed into a 4-byte integer. From least significant to most significant byte, the data values are
Minutes (least significant byte)	
Seconds	
Frames	
Unused (most significant byte)	
tfFrames	Frames are stored as a 4-byte integer variable.
tfSMPTE24	24-frame SMPTE packs values in a 4-byte variable. From least significant to most significant byte, the data values are
Hours (least significant byte)	
Minutes	
Seconds	
Frames (most significant byte)	
SMPTE (Society of Motion Picture and Television Engineers) time is an absolute time format expressed in hours, minutes, seconds, and frames. The standard SMPTE division types are 24, 25, and 30 frames per second.	
tfSMPTE25	25-frame SMPTE packs data into a 4-byte variable in the same order as 24-frame SMPTE.
tfSMPTE30	30-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE.
tfSMPTE30Drop	30-drop-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE.
tfBytes	Bytes are stored as a 4-byte integer variable.
tfSamples	Samples are stored as a 4-byte integer variable.
tfTMSF	Tracks, minutes, seconds, and frames are packed in the 4-byte variable. From least significant to most significant byte, the data values are
Tracks (least significant byte)	
Minutes	
Seconds	
Frames (most significant byte)	

Note that MCI uses continuous track numbering.

Note: Functions provided with MCI to help decoding the 4-byte integer specified in a given time format are documented under MCI Macros for Encoding and Decoding Time Data in the MMSYSTEM.HLP Help file.

See Also

TMediaPlayer

101.3 Types

The following table lists types in this documentation.

Types

Name	Description
EMPNotify (see page 17237)	EMPNotify is the type of event handlers that respond when the user interacts with a TMediaPlayer (see page 17167) component.
EMPPostNotify (see page 17237)	EMPPostNotify is the type of a media player's OnPostClick event handler.
TButtonSet (see page 17238)	TButtonSet is a set of TMPBtnType (see page 17229) values.
TMPDevCapsSet (see page 17238)	This is type MPlayer.TMPDevCapsSet.

101.3.1 MPlayer.EMPNotify Type

EMPNotify is the type of event handlers that respond when the user interacts with a TMediaPlayer (see page 17167) component.

Pascal

```
EMPNotify = procedure (Sender: TObject; Button: TMPBtnType; var DoDefault: Boolean) of object;
```

C++

```
(Sender: TObject; Button: TMPBtnType; var DoDefault: Boolean) ( EMPNotify)();
```

File

MPlayer

Description

Sender is the TMediaPlayer (see page 17167) component whose button was clicked.

Button indicates the button that was clicked. It can be one of the following values: btBack, btEject, btNext, btPause, btPlay, btPrev, btRecord, btStep, or btStop.

DoDefault returns an indication of whether the media player should make its default response after the event handler exits.

See Also

TMediaPlayer

101.3.2 MPlayer.EMPPostNotify Type

EMPPostNotify is the type of a media player's OnPostClick event handler.

Pascal

```
EMPPostNotify = procedure (Sender: TObject; Button: TMPBtnType) of object;
```

C++

```
(Sender: TObject; Button: TMPBtnType) ( EMPPostNotify)();
```

File

MPlayer

Description

Sender is the TMediaPlayer (see page 17167) component whose button was clicked.

Button indicates the button that was clicked. It can be one of the following values: btBack, btEject, btNext, btPause, btPlay, btPrev, btRecord, btStep, or btStop.

See Also

TMediaPlayer (see page 17167)

101.3.3 MPlayer.TButtonSet Type

TButtonSet is a set of TMPBtnType (see page 17229) values.

Pascal

```
TButtonSet = set of TMPBtnType;
```

C++

```
TMPBtnType TButtonSet;
```

File

MPlayer

Description

TButtonSet represents a set of zero or more of the buttons that can appear on a TMPDeviceType control.

101.3.4 MPlayer.TMPDevCapsSet Type

Pascal

```
TMPDevCapsSet = set of TMPDevCaps;
```

C++

```
TMPDevCaps TMPDevCapsSet;
```

File

MPlayer

Description

This is type MPlayer.TMPDevCapsSet.

102 Mtsobj Namespace

This is namespace Mtsobj.

Topics

Name	Description
Classes (see page 17240)	The following table lists classes in this documentation.

Classes

	Name	Description
	TMtsAutoObject (see page 17240)	TMtsAutoObject encapsulates the objects and interfaces of the MTS application server.

102.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TMtsAutoObject (see page 17240)	TMtsAutoObject encapsulates the objects and interfaces of the MTS application server.

102.1.1 TMtsAutoObject Class

TMtsAutoObject encapsulates the objects and interfaces of the MTS application server.

Class Hierarchy

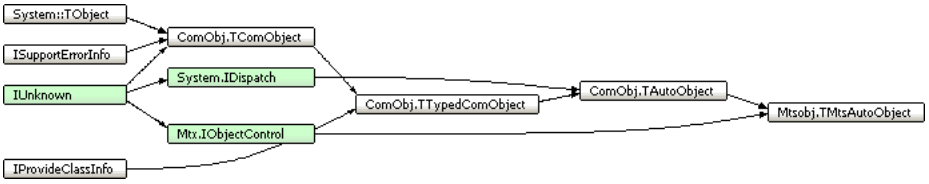

Pascal

```
TMtsAutoObject = class(TAutoObject, IObjectControl);
```

C++

```
class TMtsAutoObject : public TAutoObject, public IObjectControl;
```

File

Mtsobj

Description

Use a TMtsAutoObject object as the central repository for all objects in an MTS-enabled application server.

MTS Automation objects are dual-interface Automation servers that implement the IObjectControl interface. MTS Automation objects are like any Automation server with additional features that encapsulate MTS services such as transaction support and security.

See Also

TMTSDataModule

Writing Applications Using COM

102.1.1.1 TMtsAutoObject Members

The following tables list the members exposed by TMtsAutoObject.

Protected Methods

	Name	Description
	_AddRef (see page 5558)	This is _AddRef, a member of class TComObject.
	_Release (see page 5558)	This is _Release, a member of class TComObject.
	Activate (see page 17243)	This is Activate, a member of class TMtsAutoObject.
V	CanBePooled (see page 17243)	This is CanBePooled, a member of class TMtsAutoObject.
	Deactivate (see page 17243)	This is Deactivate, a member of class TMtsAutoObject.
	EventConnect (see page 5542)	This is EventConnect, a member of class TAutoObject.
V	EventSinkChanged (see page 5543)	This is EventSinkChanged, a member of class TAutoObject.
	GetClassInfo (see page 5583)	Returns a pointer to a TTypedComObject (see page 5581)'s type information.
V	GetIDsOfNames (see page 5543)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetTypeInfo (see page 5543)	Retrieves the type information for an object.
V	GetTypeInfoCount (see page 5544)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InterfaceSupportsErrorInfo (see page 5559)	This is InterfaceSupportsErrorInfo, a member of class TComObject.
V	Invoke (see page 5544)	Provides access to Automation properties and methods exposed by an object that supports the IDispatch interface.
V	OnActivate (see page 17246)	This is OnActivate, a member of class TMtsAutoObject.
V	OnDeactivate (see page 17246)	This is OnDeactivate, a member of class TMtsAutoObject.
	QueryInterface (see page 5560)	This is QueryInterface, a member of class TComObject.

Protected Properties

	Name	Description
R	AutoFactory (see page 5545)	The class factory that created this TAutoObject (see page 5540).
	EventSink (see page 5545)	This is EventSink, a member of class TAutoObject.
R	ObjectContext (see page 17247)	This is ObjectContext, a member of class TMtsAutoObject.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5555)	Instantiates a COM object that is not part of an aggregate.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateAggregated (see page 5556)	Instantiates a COM object that is part of an aggregate.
	CreateFromFactory (see page 5556)	Instantiates a COM object and initializes its values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5557)	Disposes of an instance of a COM object.
V	Destroy (see page 22287)	Disposes of an object instance.
	DisableCommit (see page 17243)	Prevents clients from completing transactions.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EnableCommit (see page 17244)	Allows clients to complete transactions without releasing state information.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 5544)	Provides an interface to a method that initializes the Automation object.
V	Initialize (see page 5558)	Provides an interface to a method that initializes the COM object.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	IsCallerInRole (see page 17245)	Indicates whether the current client matches a particular security role.
	IsInTransaction (see page 17245)	Indicates whether the MTS Automation object is running in a transaction.
	IsSecurityEnabled (see page 17246)	This is IsSecurityEnabled, a member of class TMtsAutoObject.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	ObjAddRef (see page 5559)	Implements the AddRef method of the IUnknown interface.
V	ObjQueryInterface (see page 5559)	Implements the QueryInterface (see page 5560) method of the IUnknown interface.
V	ObjRelease (see page 5559)	Implements the Release method of the IUnknown interface.
V	SafeCallException (see page 5560)	Handles OLE exceptions.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAbort (see page 17246)	Allows the MTS Automation object to be deactivated and aborts any current transactions.
	SetComplete (see page 17247)	Indicates that the MTS Automation object has successfully completed its current task.

Public Properties

	Name	Description
R	Controller (see page 5561)	Specifies the controlling IUnknown interface that controls interfaces within an aggregate object.
R	Factory (see page 5561)	Specifies the class factory object for the TComObject (see page 5553) class.
	Pooled (see page 17248)	Indicates whether the object can be pooled when installed under COM+.
R	RefCount (see page 5562)	Indicates the number of references to the COM object that are currently in use.
	ServerExceptionHandler (see page 5562)	Provides access to the SafeCall exception notification process.

102.1.1.2 TMtsAutoObject Methods

The methods of the TMtsAutoObject class are listed here.

Protected Methods

	Name	Description
	Activate (see page 17243)	This is Activate, a member of class TMtsAutoObject.
V	CanBePooled (see page 17243)	This is CanBePooled, a member of class TMtsAutoObject.
	Deactivate (see page 17243)	This is Deactivate, a member of class TMtsAutoObject.
V	OnActivate (see page 17246)	This is OnActivate, a member of class TMtsAutoObject.
V	OnDeactivate (see page 17246)	This is OnDeactivate, a member of class TMtsAutoObject.

Public Methods

	Name	Description
	DisableCommit (see page 17243)	Prevents clients from completing transactions.

	EnableCommit (see page 17244)	Allows clients to complete transactions without releasing state information.
	IsCallerInRole (see page 17245)	Indicates whether the current client matches a particular security role.
	IsInTransaction (see page 17245)	Indicates whether the MTS Automation object is running in a transaction.
	IsSecurityEnabled (see page 17246)	This is IsSecurityEnabled, a member of class TMtsAutoObject.
	SetAbort (see page 17246)	Allows the MTS Automation object to be deactivated and aborts any current transactions.
	SetComplete (see page 17247)	Indicates that the MTS Automation object has successfully completed its current task.

102.1.1.2.1 TMtsAutoObject.Activate Method

Pascal

```
procedure Activate; safecall;
```

C++

```
__fastcall Activate();
```

Description

This is Activate, a member of class TMtsAutoObject.

102.1.1.2.2 TMtsAutoObject.CanBePooled Method

Pascal

```
function CanBePooled: Bool; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall Bool CanBePooled();
```

Description

This is CanBePooled, a member of class TMtsAutoObject.

102.1.1.2.3 TMtsAutoObject.Deactivate Method

Pascal

```
procedure Deactivate; stdcall;
```

C++

```
__fastcall __stdcall Deactivate();
```

Description

This is Deactivate, a member of class TMtsAutoObject.

102.1.1.2.4 TMtsAutoObject.DisableCommit Method

Prevents clients from completing transactions.

Pascal

```
procedure DisableCommit;
```

C++

```
__fastcall DisableCommit();
```

Description

Call DisableCommit to indicate that the MTS Automation object cannot complete its current transaction or release state information until it receives further method invocations from the client. After calling DisableCommit, the current transaction (if any) cannot be completed until the MTS Automation object calls EnableCommit (see page 17244) or SetComplete (see page 17247).

With transactions that are started automatically when the MTS Automation object's interface is called, DisableCommit is not necessary. These transactions are not committed until the MTS Automation object calls SetComplete (see page 17247). However, when the MTS Automation object is enlisted in a transaction that was started previously, the client controls the transaction and can attempt to commit it before the MTS Automation object has completed its part. Calling DisableCommit prevents the client from prematurely committing a transaction. If a client tries to commit a transaction after the MTS Automation object has called DisableCommit, the transaction aborts.

Tip: Call DisableCommit at the beginning of a method to protect your MTS Automation object from committing prematurely in the event of an unexpected return. If the method call finishes normally, calling EnableCommit (see page 17244) or SetComplete (see page 17247) reverses this affect and allows the transaction to complete.

See Also

EnableCommit (see page 17244)

SetAbort (see page 17246)

SetComplete (see page 17247)

IsInTransaction (see page 17245)

102.1.1.2.5 TMtsAutoObject.EnableCommit Method

Allows clients to complete transactions without releasing state information.

Pascal

```
procedure EnableCommit;
```

C++

```
__fastcall EnableCommit();
```

Description

Call EnableCommit to indicate that the MTS Automation object can complete its current transaction without releasing state information. Unlike the SetComplete (see page 17247) method (which also signals that the current transaction can be committed), EnableCommit does not allow MTS to deactivate the MTS Automation object. This lets the MTS Automation object preserve state information over multiple interface calls, while still allowing clients that have enlisted the object in a transaction to commit.

By default, transactions can be completed when an interface call ends, unless the DisableCommit (see page 17243) method is called. There is no need to call EnableCommit unless the MTS Automation object has previously called DisableCommit (see page 17243).

Note: Although EnableCommit permits the current transaction to be completed, it does not automatically commit the transaction. The client that enlisted the MTS Automation object must still explicitly commit that transaction. If MTS started the transaction automatically (because the MTS Automation object requires a transaction for its interface calls), the transaction is not committed until the data module calls SetComplete (see page 17247).

See Also

DisableCommit (see page 17243)

SetComplete (see page 17247)

SetAbort (see page 17246)

IsInTransaction (see page 17245)

102.1.1.2.6 TMtsAutoObject.IsCallerInRole Method

Indicates whether the current client matches a particular security role.

Pascal

```
function IsCallerInRole(const Role: WideString): Bool;
```

C++

```
__fastcall Bool IsCallerInRole(const BSTR Role);
```

Description

MTS provides role-based security. Each level of access to your application server can be assigned a role. Using the MTS Explorer, administrators can prevent clients from using an interface based on their role. (This is called declarative security.) In addition, however, programs can provide more fine-grained security programmatically. To do this, use IsCallerInRole to provide access to blocks of code based on the caller's security role.

IsCallerInRole checks whether the current client is part of the role named by the Role parameter. It returns true if the client matches the named role, and False otherwise. Use IsCallerInRole to prevent clients from invoking certain blocks of code if they do not have the appropriate security role.

Note: MTS security is not available on servers running Windows95. If the MTS application is running on Windows95, IsCallerInRole always returns true . In addition, IsCallerInRole always returns True if the application is part of an MTS Library package (which runs in the client's process space).

102.1.1.2.7 TMtsAutoObject.IsInTransaction Method

Indicates whether the MTS Automation object is running in a transaction.

Pascal

```
function IsInTransaction: Bool;
```

C++

```
__fastcall Bool IsInTransaction();
```

Description

Call IsInTransaction to determine whether the MTS Automation object is currently running in a transaction. For example, if the MTS Automation object requires a transaction, use IsInTransaction to check that this requirement has not been turned off by an administrator using the MTS Explorer, and to raise an exception if it has.

Note: If IsInTransaction returns False, the EnableCommit (see page 17244) and DisableCommit (see page 17243) methods have no effect.

See Also

EnableCommit (see page 17244)

DisableCommit (see page 17243)

102.1.1.2.8 TMtsAutoObject.IsSecurityEnabled Method

Pascal

```
function IsSecurityEnabled: Bool;
```

C++

```
__fastcall Bool IsSecurityEnabled();
```

Description

This is IsSecurityEnabled, a member of class TMtsAutoObject.

102.1.1.2.9 TMtsAutoObject.OnActivate Method

Pascal

```
procedure OnActivate; virtual;
```

C++

```
virtual __fastcall OnActivate();
```

Description

This is OnActivate, a member of class TMtsAutoObject.

102.1.1.2.10 TMtsAutoObject.OnDeactivate Method

Pascal

```
procedure OnDeactivate; virtual;
```

C++

```
virtual __fastcall OnDeactivate();
```

Description

This is OnDeactivate, a member of class TMtsAutoObject.

102.1.1.2.11 TMtsAutoObject.SetAbort Method

Allows the MTS Automation object to be deactivated and aborts any current transactions.

Pascal

```
procedure SetAbort;
```

C++

```
__fastcall SetAbort();
```

Description

Call SetAbort to indicate that any current transaction that includes the MTS Automation object to abort. If MTS started a transaction automatically, the transaction aborts when the current method call exits. If the MTS Automation object was enlisted in a pre-existing transaction, the client transaction will eventually fail. Usually, SetAbort is called in the except block of a try...except statement.

In addition to aborting any current transaction, SetAbort indicates that the MTS Automation object can be deactivated when the current interface call exits. This means that state information is not preserved for the next time and the client calls the Automation object's interface. To prevent transactions from committing successfully without losing state information, use the

DisableCommit (see page 17243) method instead.

Tip: Call SetAbort at the beginning of a method to protect your MTS data module from committing prematurely in the event of an unexpected return. If the method call finishes normally, calling SetComplete (see page 17247) reverses this affect and allows the transaction to complete.

102.1.1.2.12 TMtsAutoObject.SetComplete Method

Indicates that the MTS Automation object has successfully completed its current task.

Pascal

```
procedure SetComplete;
```

C++

```
__fastcall SetComplete();
```

Description

Call SetComplete to indicate that the MTS Automation object no longer needs to maintain state information. After calling SetComplete, the MTS Automation object is deactivated when the current method call exits.

If the MTS Automation object is part of a transaction, SetComplete indicates that this part of the transaction can commit. If MTS started the transaction automatically when the client called the MTS Automation object's methods, MTS tries to commit the transaction when the current method exits.

Note: This transaction may still fail if the current transaction included another MTS object that called SetAbort (see page 17246) or DisableCommit (see page 17243).

See Also

EnableCommit (see page 17262)

102.1.1.3 TMtsAutoObject Properties

The properties of the TMtsAutoObject class are listed here.

Protected Properties

	Name	Description
	ObjectContext (see page 17247)	This is ObjectContext, a member of class TMtsAutoObject.

Public Properties

	Name	Description
	Pooled (see page 17248)	Indicates whether the object can be pooled when installed under COM+.

102.1.1.3.1 TMtsAutoObject.ObjectContext Property

Pascal

```
property ObjectContext: IObjectContext;
```

C++

```
__property IObjectContext ObjectContext;
```

Description

This is ObjectContext, a member of class TMtsAutoObject.

102.1.1.3.2 TMtsAutoObject.Pooled Property

Indicates whether the object can be pooled when installed under COM+.

Pascal

```
property Pooled: Boolean;
```

C++

```
__property Boolean Pooled;
```

Description

Set Pooled to indicate whether instances of the MTS Automation object can be pooled when installed under COM+. The MTS Automation object uses the Pooled property when implementing the IObjectControl method CanBePooled (see page 17243). This instructs COM+ that the object can be moved to an object pool when deactivated, so that the single instance is shared by all clients.

Warning: Do not set Pooled to True unless the MTS Automation object is stateless.

See Also

Object Pooling

103 MtsRdm Namespace

This is namespace MtsRdm.

Topics

Name	Description
Classes (see page 17250)	The following table lists classes in this documentation.

Classes

	Name	Description
	TMtsDataModule (see page 17250)	TMtsDataModule encapsulates the objects and interfaces of a transactional application server.

103.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TMtsDataModule (see page 17250)	TMtsDataModule encapsulates the objects and interfaces of a transactional application server.

103.1.1 TMtsDataModule Class

TMtsDataModule encapsulates the objects and interfaces of a transactional application server.

Class Hierarchy

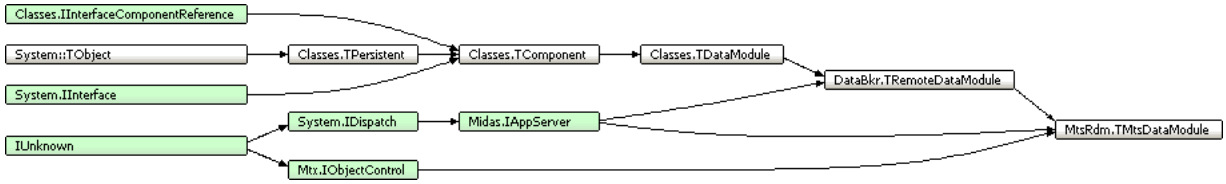

Pascal

```
TMtsDataModule = class(TRemoteDataModule, IAppServer, IObjectControl);
```

C++

```
class TMtsDataModule : public TRemoteDataModule, public IAppServer, public IObjectControl;
```

File

MtsRdm

Description

Use a TMtsDataModule object as the central repository for all objects in a transactional application server that is written in Delphi. MTS data modules can contain any nonvisual components. . Typically these are data access components, such as datasets and the provider components that handle communication between these datasets and client applications. MTS data modules can also contain other nonvisual components, such as TTimer, or TImageList.

MTS data modules are dual-interface Automation servers that implement the IAppServer interface. Connection components on client applications are designed to look for this interface. The methods in this interface, however, do not call the SetComplete (see page 17265) method, so the MTS data module is not transactional. You may want to expand the MTS data module's interface to provide alternate versions of these methods that use the MTS data module's transaction.

The default interaction between a client dataset and an MTS data module does not rely on state information, so it is safe to use the MTS data module with just-in-time activation. However, you may need to provide a mechanism for communicating persistent state information to take advantage of some features such as incremental fetches.

To create an MTS data module at design time, choose File|New|Other, and select Transactional Data Module from the multitier page of the new items dialog. If you are writing the application server in Delphi, the Transactional Data Module Wizard then creates a new descendant of TMtsDataModule that supports an interface that descends from IAppServer.

See Also

IAppServer (see page 16450)

Writing Applications Using COM

103.1.1.1 TMtsDataModule Members

The following tables list the members exposed by TMtsDataModule.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	Activate (see page 17255)	This is Activate, a member of class TMtsDataModule.
	AS_ApplyUpdates (see page 17255)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 17256)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 17257)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 17258)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 17258)	Returns a list of all the available providers on the remote data module.
	AS_GetRecords (see page 17259)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 17260)	Returns information from a specified record of the provider's dataset.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	CallComplete (see page 17261)	This is CallComplete, a member of class TMtsDataModule.
V	CanBePooled (see page 17261)	This is CanBePooled, a member of class TMtsDataModule.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Deactivate (see page 17261)	This is Deactivate, a member of class TMtsDataModule.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
V	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetProvider (see page 6595)	This is GetProvider, a member of class TRemoteDataModule.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.

V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
V	UpdateRegistry (see page 6597)	Registers the remote data module in the system registry as an automation server.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ObjectContext (see page 17266)	This is ObjectContext, a member of class TMtsDataModule.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 2829)	Generates an OnCreate (see page 2833) event.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 2830)	Generates an OnDestroy (see page 2834) event.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 17254)	Creates an MTS data module instance.
V	CreateNew (see page 2828)	Creates an instance of a data module and registers with the global screen object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2829)	Destroys the instance of the data module.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 6589)	Frees a remote data module instance.

	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableCommit (see page 17261)	Prevents clients from completing transactions.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EnableCommit (see page 17262)	Allows clients to complete transactions without releasing state information.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	IsCallerInRole (see page 17263)	Indicates whether the current client fits into a particular security role.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsInTransaction (see page 17263)	Indicates whether the MTS data module is running in a transaction.
	IsSecurityEnabled (see page 17264)	This is IsSecurityEnabled, a member of class TMtsDataModule.
	Lock (see page 6595)	Locks the remote data module so that it can't be locked by other threads.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterProvider (see page 6596)	Adds a provider to the list returned by AS_GetProviderNames (see page 6593).
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAbort (see page 17264)	Allows the MTS data module to be deactivated and aborts any current transactions.

	SetComplete (see page 17265)	Indicates that the MTS data module has successfully completed its current task.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Unlock (see page 6596)	Unlocks the remote data module after it was locked by a call to the Lock (see page 6595) method.
V	UnRegisterProvider (see page 6597)	Removes a provider from the list returned by AS_GetProviderNames (see page 6593).
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	ProviderCount (see page 6599)	Indicates the number of providers registered with the remote data module.
R	Providers (see page 6599)	This is Providers, a member of class TRemoteDataModule.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnActivate (see page 17267)	Occurs when an instance of TMtsDataModule (see page 17250) is activated.
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDeactivate (see page 17267)	Occurs when an instance of TMtsDataModule (see page 17250) is deactivated.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.

Published Properties

	Name	Description
	AutoComplete (see page 17265)	Specifies whether the IAppServer methods are transactional.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.
	Pooled (see page 17266)	Indicates whether the data module can be pooled when installed under COM+.
	Tag (see page 2808)	Stores an integer value as part of a component.

103.1.1.2 TMtsDataModule.Create Constructor

Creates an MTS data module instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TMtsDataModule(TComponent * AOwner);
```

Description

Most applications do not need to call the constructor for TMtsDataModule (see page 17250). Instead, add a remote data module to a server application using the transactional data module wizard on the multi-tier page of the new items dialog.

AOwner specifies the component that is responsible for freeing the remote data module (typically the application).

See Also

Create (see page 2827)

103.1.1.3 TMtsDataModule Methods

The methods of the TMtsDataModule class are listed here.

Protected Methods

	Name	Description
	Activate (see page 17255)	This is Activate, a member of class TMtsDataModule.
	AS_ApplyUpdates (see page 17255)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 17256)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 17257)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 17258)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 17258)	Returns a list of all the available providers on the remote data module.
	AS_GetRecords (see page 17259)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 17260)	Returns information from a specified record of the provider's dataset.
V	CallComplete (see page 17261)	This is CallComplete, a member of class TMtsDataModule.
V	CanBePooled (see page 17261)	This is CanBePooled, a member of class TMtsDataModule.
	Deactivate (see page 17261)	This is Deactivate, a member of class TMtsDataModule.

Public Methods

	Name	Description
	DisableCommit (see page 17261)	Prevents clients from completing transactions.
	EnableCommit (see page 17262)	Allows clients to complete transactions without releasing state information.
	IsCallerInRole (see page 17263)	Indicates whether the current client fits into a particular security role.
	IsInTransaction (see page 17263)	Indicates whether the MTS data module is running in a transaction.
	IsSecurityEnabled (see page 17264)	This is IsSecurityEnabled, a member of class TMtsDataModule.
	SetAbort (see page 17264)	Allows the MTS data module to be deactivated and aborts any current transactions.
	SetComplete (see page 17265)	Indicates that the MTS data module has successfully completed its current task.

103.1.1.3.1 TMtsDataModule.Activate Method

Pascal

```
procedure Activate; safecall;
```

C++

```
__fastcall Activate();
```

Description

This is Activate, a member of class TMtsDataModule.

103.1.1.3.2 TMtsDataModule.AS_ApplyUpdates Method

Applies updates received from a client dataset using a specified provider.

Pascal

```
function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_ApplyUpdates(const BSTR ProviderName, OleVariant Delta, int
MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

AS_AS_ApplyUpdates writes client updates to a dataset or database server. ProviderName specifies the name of the provider component that manages the update operation. Delta is a Variant containing the updated, inserted, and deleted records to write to the database. MaxErrors indicates the maximum number of errors to permit before rolling back the update operation; ErrorCount returns the actual number of errors encountered during the update operation. OwnerData is information supplied by the client application in a BeforeAS_ApplyUpdates event handler and returned to the client application in an AfterAS_ApplyUpdates event handler. It is passed to the provider's BeforeAS_ApplyUpdates event handler and returned from the provider's AfterAS_ApplyUpdates event handler.

AS_AS_ApplyUpdates The AS_ApplyUpdates_result parameter returns a Variant that is a data packet containing all records that could not be applied.

Note: Applications can only call the protected AS_AS_ApplyUpdates method using the IAppServer interface. To apply updates directly in code from the application server, use the provider component's AS_ApplyUpdates method.

See Also

ApplyUpdates (see page 8394)

applyupdates (see page 18459)

BeforeApplyUpdates (see page 18479)

AfterApplyUpdates (see page 18475)

BeforeApplyUpdates (see page 8511)

AfterApplyUpdates (see page 8497)

providers (see page 6599)

IAppServer (see page 16450)

103.1.1.3.3 TMtsDataModule.AS_DataRequest Method

Generates an OnAS_DataRequest event on the specified provider component.

Pascal

```
function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

Use AS_AS_DataRequest to invoke an OnAS_DataRequest event handler of the provider named by ProviderName. This method is provided to allow application developers to customize the communication between a client application and a provider component. There is no predefined meaning for the Data parameter or the return value AS_DataRequest_result parameter, which is returned to the caller.

Note: Applications can only call the protected AS_AS_DataRequest method using the IAppServer interface.

See Also

OnDataRequest (see page 18482)

providers (see page 6599)

DataRequest (see page 8404)

IAppServer (see page 16450)

103.1.1.3.4 TMtsDataModule.AS_Execute Method

AS_Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); safecall;
```

C++

```
__fastcall AS_Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant Params,
OleVariant OwnerData);
```

Description

Use AS_AS_Execute to pass an execute command to a remote provider, using a specified set of parameters. AS_AS_Execute calls the AS_Execute method of the specified provider after assigning any parameters. Output parameters that result from executing a query or stored procedure are returned in the Params parameter.

The provider's AS_Execute method generates a BeforeAS_Execute event, passes the command on to its associated dataset, and generates an AfterAS_Execute event. The dataset must represent a query or stored procedure that does not return a cursor.

The ProviderName parameter specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the query, or the name of a stored procedure that replaces the dataset's stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

The Params parameter encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams routine. To decode Params into a TParams object, use the Unpackparams routine.

The OwnerData parameter contains custom information that appears as an argument to the provider's BeforeAS_Execute and AfterAS_Execute event handlers. This information originates in a client dataset's BeforeAS_Execute event handler. The value returned in OwnerData is passed to the client dataset's AfterAS_Execute event handler.

See Also

DataSet (see page 18529)

beforeexecute (see page 18479)

AfterExecute (see page 18476)

Params (see page 8489)

Execute (see page 8413)

Execute (see page 18466)

BeforeExecute (see page 8514)

AfterExecute (see page 8504)

Options (see page 18440)

PackageParams (see page 8534)

UnpackParams (see page 8534)

Communicating with the Client Dataset

103.1.1.3.5 TMtsDataModule.AS_GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

Use AS_AS_GetParams to retrieve the current parameter values from the dataset associated with a provider.

The ProviderName parameter specifies the name of the provider component whose dataset has the parameters.

The OwnerData represents custom information that originates in a client dataset's BeforeAS_GetParams event handler and returns information that is passed to the client dataset's AfterAS_GetParams event handler.

AS_GetParams_result (C++) or AS_AS_GetParams (Delphi) returns the parameters, encoded as a Variant array, where each element is a 2-element Variant array containing a parameter name and a parameter value. These parameters can be converted to a TParams object using the UnpackParams procedure. If the dataset has no parameters, or if the provider does not support parameter fetching, AS_GetParams_result (C++) or AS_AS_GetParams (Delphi) returns a Null Variant.

Note: AS_AS_GetParams should not be used to retrieve output parameters from stateless application servers, because parameter values may be changed by other applications. When writing a stateless application server, retrieve parameter values using the AS_Execute method instead.

See Also

DataSet (see page 18529)

Params (see page 8489)

FetchParams (see page 8415)

GetParams (see page 18467)

BeforeGetParams (see page 18480)

aftergetparams (see page 18477)

AS_Execute (see page 16453)

UnpackParams (see page 8534)

Communicating with the Client Dataset

103.1.1.3.6 TMtsDataModule.AS_GetProviderNames Method

Returns a list of all the available providers on the remote data module.

Pascal

```
function AS_GetProviderNames: OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetProviderNames();
```

Description

Call AS_AS_GetProviderNames for a list of all registered providers available from a remote data module. These values indicate the possible values for the ProviderName property of a client dataset that connects to the remote data module. When calling any of the remote data modules through the IAppServer interface, pass one of these names to indicate the provider that is the target of the method call.

AS_AS_GetProviderNames returns the names of the providers in a Variant array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

Note: Applications can only call the protected AS_AS_GetProviderNames method using the IAppServer interface.

See Also

ProviderName (see page 8492)

providers (see page 6599)

IAppServer (see page 16450)

Exported (see page 18474)

103.1.1.3.7 TMtsDataModule.AS_GetRecords Method

Returns a data packet that contains the specified data.

Pascal

```
function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetRecords(const BSTR ProviderName, int Count, int RecsOut, int
Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

Call AS_AS_GetRecords to retrieve a specified number of records. AS_AS_GetRecords returns the requested records, starting with the current record of the provider's dataset. When working with a stateless remote data module, you may need to reposition the cursor or re-execute a query or stored procedure in the provider's BeforeAS_GetRecords event handler.

ProviderName gives the name of the provider component that provides the records.

Count indicates the number or type of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only Count records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants. For example, the following value requests metadata along with a data packet encoded in XML:

```
MetaDataOption + XMLOption
MetaDataOption || XMLOption
```

CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a table or stored procedure that replaces the provider's underlying database table or stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

Params is a Variant containing any parameters that should be passed to the provider's dataset before it executes to generate the

requested data. It returns any output parameters.

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_GetRecords event handler. This information is passed to the provider's BeforeAS_GetRecords event handler. OwnerData returns information supplied by the provider's AfterAS_GetRecords event handler.

Records are returned as a data packet in a Variant parameter, AS_GetRecords_result.

Note: Applications can only call the protected AS_AS_GetRecords method using the IAppServer interface. To fetch records directly in code from the application server, use the provider component's AS_GetRecords method.

See Also

Data (see page 8467)

Params (see page 8489)

GetNextPacket (see page 8423)

GetRecords (see page 18468)

BeforeGetRecords (see page 18480)

AfterGetRecords (see page 18477)

providers (see page 6599)

IAppServer (see page 16450)

103.1.1.3.8 TMtsDataModule.AS_RowRequest Method

Returns information from a specified record of the provider's dataset.

Pascal

```
function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_RowRequest(const BSTR ProviderName, OleVariant Row, int RequestType,
OleVariant OwnerData);
```

Description

Client datasets call AS_AS_RowRequest automatically to implement their FetchBlobs, FetchDetails , or RefreshRecord method.

The ProviderName parameter indicates the provider associated with the dataset from which information should be fetched.

The Row parameter is an OleVariant that describes the current record on the client dataset.

The RequestType parameter indicates the type of information required. It is an integer version of the TFetchOptions type. (An integer because the value may be marshaled to a remote application server). To create a value for RequestType, take the corresponding TFetchOptions value, cast it to a Byte, and cast the result to an Integer:

```
Integer(Byte([foBlobs, foDetails]));
(int) (Byte) (TFetchOptions() << foBlobs << foDetails);
```

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_RowRequest event handler. This information is passed to the provider's BeforeAS_RowRequest event handler. OwnerData returns custom information supplied by the provider's AfterAS_RowRequest event handler.

The requested data is returned as a delta packet.

Note: Applications can only call the protected AS_AS_RowRequest method using the IAppServer interface.

See Also

RowRequest (see page 18472)

BeforeRowRequest (see page 18482)

AfterRowRequest (see page 18478)

FetchBlobs (see page 8414)

FetchDetails (see page 8414)

RefreshRecord (see page 8442)

providers (see page 6599)

IAppServer (see page 16450)

103.1.1.3.9 TMtsDataModule.CallComplete Method

Pascal

```
procedure CallComplete(Complete: Boolean); virtual;
```

C++

```
virtual __fastcall CallComplete(Boolean Complete);
```

Description

This is CallComplete, a member of class TMtsDataModule.

103.1.1.3.10 TMtsDataModule.CanBePooled Method

Pascal

```
function CanBePooled: Bool; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall Bool CanBePooled();
```

Description

This is CanBePooled, a member of class TMtsDataModule.

103.1.1.3.11 TMtsDataModule.Deactivate Method

Pascal

```
procedure Deactivate; stdcall;
```

C++

```
__fastcall __stdcall Deactivate();
```

Description

This is Deactivate, a member of class TMtsDataModule.

103.1.1.3.12 TMtsDataModule.DisableCommit Method

Prevents clients from completing transactions.

Pascal

```
procedure DisableCommit;
```

C++

```
__fastcall DisableCommit();
```

Description

Call DisableCommit to indicate that the MTS data module can't complete its current transaction or release state information (properties) until it receives further method invocations from the client. After calling DisableCommit, the current transaction (if any) can't be completed until the MTS data module calls EnableCommit (see page 17262) or SetComplete (see page 17265).

With transactions that are started automatically when the MTS data module's interface is called, DisableCommit is not necessary. These automatically generated transactions are not committed until the MTS data module calls SetComplete (see page 17265). However, when the MTS data module is enlisted in a transaction that was started previously, the client controls the transaction and can attempt to commit it before the MTS data module has completed its part. Calling DisableCommit prevents the client from prematurely committing a transaction. If a client tries to commit a transaction after the MTS data module has called DisableCommit, the transaction aborts.

Tip: Call DisableCommit at the beginning of a method to protect your MTS data module from committing prematurely in the event of an unexpected return. If the method call finishes normally, calling EnableCommit (see page 17262) or SetComplete (see page 17265) reverses this affect and allows the transaction to complete.

See Also

EnableCommit (see page 17262)

SetComplete (see page 17265)

SetAbort (see page 17264)

IsInTransaction (see page 17263)

103.1.1.3.13 TMtsDataModule.EnableCommit Method

Allows clients to complete transactions without releasing state information.

Pascal

```
procedure EnableCommit;
```

C++

```
__fastcall EnableCommit();
```

Description

Call EnableCommit to indicate that the MTS data module can complete its current transaction without releasing state information (properties). Unlike the SetComplete (see page 17265) method (which also signals that the current transaction can be committed), EnableCommit does not allow the data module to be deactivated. This lets the MTS data module preserve state information over multiple interface calls, while still allowing clients that have enlisted the data module in a transaction to commit.

By default, transactions can be completed when interface calls finish, unless the DisableCommit (see page 17261) method is called. There is no need to call EnableCommit unless the MTS data module has previously called DisableCommit (see page 17261).

Note: Although EnableCommit permits the current transaction to be completed, it does not automatically commit the transaction. The client that enlisted the MTS data module in a transaction must still explicitly commit that transaction. If the transaction was started automatically (because the MTS data module requires a transaction for its interface calls), the transaction is not

committed until the data module calls SetComplete (see page 17265).

See Also

DisableCommit (see page 17261)

SetComplete (see page 17265)

SetAbort (see page 17264)

IsInTransaction (see page 17263)

103.1.1.3.14 TMtsDataModule.IsCallerInRole Method

Indicates whether the current client fits into a particular security role.

Pascal

```
function IsCallerInRole(const Role: WideString): Bool;
```

C++

```
__fastcall Bool IsCallerInRole(const BSTR Role);
```

Description

MTS provides role-based security. Each level of access to the application server can be assigned a name, called a role. Using the MTS Explorer or COM+ administrator, administrators can prevent clients from using an interface based on their role. (This is called declarative security.) In addition, however, programs can provide more fine-grained security programmatically. To do this, use IsCallerInRole to provide access to blocks of code based on the caller's security role.

IsCallerInRole checks whether the current client is part of the role named by the Role parameter. It returns true if the client fits the named role, and false otherwise. Use IsCallerInRole to prevent clients that do not have the appropriate security roles from invoking certain blocks of code.

Note: MTS security is not available on servers running Windows95. If the application server is running on Windows95, IsCallerInRole always returns true. In addition, IsCallerInRole always returns true if the application server is part of an MTS Library package (which runs in the client's process space).

See Also

issecurityenabled (see page 17264)

Using Transactional Data Modules

103.1.1.3.15 TMtsDataModule.IsInTransaction Method

Indicates whether the MTS data module is running in a transaction.

Pascal

```
function IsInTransaction: Bool;
```

C++

```
__fastcall Bool IsInTransaction();
```

Description

Call IsInTransaction to determine whether the MTS data module is currently running in a transaction. For example, if the MTS data module requires a transaction, use IsInTransaction to check that this requirement has not been turned off using the MTS Explorer or COM+ administrator, and to raise an exception if it has.

Note: If IsInTransaction returns false, the EnableCommit (see page 17262) and DisableCommit (see page 17261) methods

have no effect.

See Also

EnableCommit (see page 17262)

DisableCommit (see page 17261)

SetComplete (see page 17265)

SetAbort (see page 17264)

103.1.1.3.16 TMtsDataModule.IsSecurityEnabled Method

Pascal

```
function IsSecurityEnabled: Bool;
```

C++

```
__fastcall Bool IsSecurityEnabled();
```

Description

This is IsSecurityEnabled, a member of class TMtsDataModule.

103.1.1.3.17 TMtsDataModule.SetAbort Method

Allows the MTS data module to be deactivated and aborts any current transactions.

Pascal

```
procedure SetAbort;
```

C++

```
__fastcall SetAbort();
```

Description

Call SetAbort to indicate that any current transaction that includes the MTS data module should abort. If MTS or COM+ started a transaction automatically (because the data module requires one), that transaction aborts when the current method call exits. If the MTS data module was enlisted in a pre-existing transaction, the client transaction will eventually fail. Usually, SetAbort is called in the except block of a try..exceptstatement.

In addition to aborting any current transaction, SetAbort indicates that the MTS data module can be deactivated when the current interface call exits. This means that state information is not preserved for the next time the client calls the data module's interface. To prevent transactions from committing successfully without losing state information, use the DisableCommit (see page 17261) method instead.

Tip: Call SetAbort at the beginning of a method to protect your MTS data module from committing prematurely in the event of an unexpected return. If the method call finishes normally, calling SetComplete (see page 17265) reverses this affect and allows the transaction to complete.

See Also

SetComplete (see page 17265)

DisableCommit (see page 17261)

EnableCommit (see page 17262)

IsInTransaction (see page 17263)

103.1.1.3.18 TMtsDataModule.SetComplete Method

Indicates that the MTS data module has successfully completed its current task.

Pascal

```
procedure SetComplete;
```

C++

```
__fastcall SetComplete();
```

Description

Call SetComplete to indicate that the MTS data module no longer needs to maintain state information. After calling SetComplete, the MTS data module is deactivated when the current method call exits.

If the MTS data module is part of a transaction, SetComplete indicates that this part of the transaction can commit. If MTS or COM+ started the transaction automatically when the client called the MTS data module's method (because the MTS data module requires a transaction), it tries to commit the transaction when the current method exits.

Note: This transaction may still fail if the current transaction included another MTS or COM+ object that called SetAbort (see page 17264) or DisableCommit (see page 17261).

See Also

SetAbort (see page 17264)

DisableCommit (see page 17261)

EnableCommit (see page 17262)

IsInTransaction (see page 17263)

Extending the Interface of the Application Server

103.1.1.4 TMtsDataModule Properties

The properties of the TMtsDataModule class are listed here.

Protected Properties

	Name	Description
	ObjectContext (see page 17266)	This is ObjectContext, a member of class TMtsDataModule.

Published Properties

	Name	Description
	AutoComplete (see page 17265)	Specifies whether the IAppServer methods are transactional.
	Pooled (see page 17266)	Indicates whether the data module can be pooled when installed under COM+.

103.1.1.4.1 TMtsDataModule.AutoComplete Property

Specifies whether the IAppServer methods are transactional.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

AutoComplete controls whether calls made using the IAppServer interface end in a call to SetComplete (see page 17265) or SetAbort (see page 17264). By calling SetComplete (see page 17265) or SetAbort (see page 17264), the MTS data module indicates that no state information needs to be preserved when the call exits. This allows any MTS transactions to complete (or roll back) and allows COM+ or the MTS proxy to deactivate the MTS data module.

When AutoComplete is true, all IAppServer methods call SetComplete (see page 17265) when they are done with two exceptions:

AS_ApplyUpdates (see page 17255) calls SetAbort (see page 17264) if an exception is raised or if the error count exceeds the specified maximum.

AS_DataRequest (see page 17256) doesn't call either SetComplete (see page 17265) or SetAbort (see page 17264). You must add the appropriate call to the OnDataRequest event handler.

Set AutoComplete to false if the MTS data module needs to preserve state information (such as the current record in a dataset) after an IAppServer call exits. When AutoComplete is false, you must add your own interface methods that call SetComplete (see page 17265) or SetAbort (see page 17264) when the MTS data module can be safely deactivated.

See Also

IAppServer (see page 16450)

SetComplete (see page 17265)

SetAbort (see page 17264)

103.1.1.4.2 TMtsDataModule.ObjectContext Property

Pascal

```
property ObjectContext: IObjectContext;
```

C++

```
__property IObjectContext ObjectContext;
```

Description

This is ObjectContext, a member of class TMtsDataModule.

103.1.1.4.3 TMtsDataModule.Pooled Property

Indicates whether the data module can be pooled when installed under COM+.

Pascal

```
property Pooled: Boolean;
```

C++

```
__property Boolean Pooled;
```

Description

Set Pooled to indicate whether instances of the MTS data module can be pooled when installed under COM+. The MTS remote data module uses the Pooled property when implementing the IObjectControl method CanBePooled (see page 17261). This instructs COM+ that the object can be moved to an object pool when deactivated, so that the single instance is shared by all clients.

Warning: Do not set Pooled to True unless the MTS data module is stateless.

103.1.1.5 TMtsDataModule Events

The events of the TMtsDataModule class are listed here.

Published Events

	Name	Description
	OnActivate (see page 17267)	Occurs when an instance of TMtsDataModule (see page 17250) is activated.
	OnDeactivate (see page 17267)	Occurs when an instance of TMtsDataModule (see page 17250) is deactivated.

103.1.1.5.1 TMtsDataModule.OnActivate Event

Occurs when an instance of TMtsDataModule (see page 17250) is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to take specific action when COMN+ or the MTS proxy activates the MTS data module. This is the point when any database connections should be established. Database connections must not opened before the MTS data module is activated.

Note: In traditional MTS or COM+ programming, objects acquire their context object when they are activated. However, TMtsDataModule (see page 17250) handles this task automatically.

See Also

OnDeactivate (see page 17267)

103.1.1.5.2 TMtsDataModule.OnDeactivate Event

Occurs when an instance of TMtsDataModule (see page 17250) is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to take specific action when COM+ or the MTS proxy releases an instance of the MTS data module. Use this method to close any database connections that were opened when the MTS data module was activated. Database connections must not be left open after the MTS data module is deactivated.

Note: In traditional COM+ and MTS programming, objects release their context object when they are deactivated. However, TMtsDataModule (see page 17250) handles this task automatically.

See Also

OnActivate (see page 17267)

104 Mtx Namespace

This is namespace Mtx.

Topics

Name	Description
Interfaces (see page 17271)	The following table lists interfaces in this documentation.
Functions (see page 17292)	The following table lists functions in this documentation.
Constants (see page 17297)	The following table lists constants in this documentation.

Constants

Name	Description
CLASS_SharedPropertyGroupManager (see page 17297)	This is constant Mtx.CLASS_SharedPropertyGroupManager.
CLASS_TransactionContext (see page 17297)	This is constant Mtx.CLASS_TransactionContext.
CLASS_TransactionContextEx (see page 17298)	This is constant Mtx.CLASS_TransactionContextEx.
LockMethod (see page 17298)	This is constant Mtx.LockMethod.
LockSetGet (see page 17298)	This is constant Mtx.LockSetGet.
mtsErrCtxAborted (see page 17299)	This is constant Mtx.mtsErrCtxAborted.
mtsErrCtxAborting (see page 17299)	This is constant Mtx.mtsErrCtxAborting.
mtsErrCtxActivityTimeout (see page 17299)	This is constant Mtx.mtsErrCtxActivityTimeout.
mtsErrCtxNoContext (see page 17300)	This is constant Mtx.mtsErrCtxNoContext.
mtsErrCtxNoSecurity (see page 17300)	This is constant Mtx.mtsErrCtxNoSecurity.
mtsErrCtxNotRegistered (see page 17300)	This is constant Mtx.mtsErrCtxNotRegistered.
mtsErrCtxOldReference (see page 17301)	This is constant Mtx.mtsErrCtxOldReference.
mtsErrCtxRoleNotFound (see page 17301)	This is constant Mtx.mtsErrCtxRoleNotFound.
mtsErrCtxTMNotAvailable (see page 17301)	This is constant Mtx.mtsErrCtxTMNotAvailable.
mtsErrCtxWrongThread (see page 17302)	This is constant Mtx.mtsErrCtxWrongThread.
Process (see page 17302)	This is constant Mtx.Process.
Standard (see page 17302)	This is constant Mtx.Standard.

Functions

	Name	Description
	CreateSharedPropertyGroup (see page 17292)	Returns the interface for a SharedPropertyGroup that allows MTS objects within a server process to share state.
	CreateSharedPropertyGroupManager (see page 17293)	Returns an interface for a new shared property group manager that creates and coordinates shared properties of MTS objects.
	CreateTransactionContext (see page 17293)	Starts an MTS transaction.

	CreateTransactionContextEx (see page 17294)	Instantiates a TransactionContextEx object, which marks the beginning of a transaction.
	GetObjectContext (see page 17295)	Fetches the IObjectContext (see page 17274) interface for the current thread.
	SafeRef (see page 17295)	Returns a reference to a transactional object that is safe to pass outside its context.

Interfaces

	Name	Description
	IEnumNames (see page 17271)	This is class Mtx.IEnumNames.
	IGetContextProperties (see page 17273)	This is class Mtx.IGetContextProperties.
	IObjectContext (see page 17274)	This is class Mtx.IObjectContext.
	IObjectContextActivity (see page 17277)	This is class Mtx.IObjectContextActivity.
	IObjectControl (see page 17278)	This is class Mtx.IObjectControl.
	ISecurityProperty (see page 17279)	This is class Mtx.ISecurityProperty.
	ISharedProperty (see page 17281)	This is class Mtx.ISharedProperty.
	ISharedPropertyGroup (see page 17283)	This is class Mtx.ISharedPropertyGroup.
	ISharedPropertyGroupManager (see page 17285)	This is class Mtx.ISharedPropertyGroupManager.
	ITransactionContext (see page 17288)	This is class Mtx.ITransactionContext.
	ITransactionContextEx (see page 17289)	This is class Mtx.ITransactionContextEx.

104.1 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IEnumNames (see page 17271)	This is class Mtx.IEnumNames.
	IGetContextProperties (see page 17273)	This is class Mtx.IGetContextProperties.
	IObjectContext (see page 17274)	This is class Mtx.IObjectContext.
	IObjectContextActivity (see page 17277)	This is class Mtx.IObjectContextActivity.
	IObjectControl (see page 17278)	This is class Mtx.IObjectControl.
	ISecurityProperty (see page 17279)	This is class Mtx.ISecurityProperty.
	ISharedProperty (see page 17281)	This is class Mtx.ISharedProperty.
	ISharedPropertyGroup (see page 17283)	This is class Mtx.ISharedPropertyGroup.
	ISharedPropertyGroupManager (see page 17285)	This is class Mtx.ISharedPropertyGroupManager.
	ITransactionContext (see page 17288)	This is class Mtx.ITransactionContext.
	ITransactionContextEx (see page 17289)	This is class Mtx.ITransactionContextEx.

104.1.1 IEnumNames Interface

Class Hierarchy

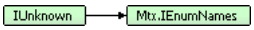

Pascal

```
IEnumNames = interface(IUnknown);
```

C++

```
__interface IEnumNames : public IUnknown;
```

File

Mtx

Description

This is class Mtx.IEnumNames.

104.1.1.1 IEnumNames Members

The following tables list the members exposed by IEnumNames.

Public Methods

	Name	Description
	Clone (see page 17272)	This is Clone, a member of class IEnumNames.
	Next (see page 17272)	This is Next, a member of class IEnumNames.
	Reset (see page 17272)	This is Reset, a member of class IEnumNames.
	Skip (see page 17272)	This is Skip, a member of class IEnumNames.

104.1.1.2 IEnumNames Methods

The methods of the IEnumNames class are listed here.

Public Methods

	Name	Description
	Clone (see page 17272)	This is Clone, a member of class IEnumNames.
	Next (see page 17272)	This is Next, a member of class IEnumNames.
	Reset (see page 17272)	This is Reset, a member of class IEnumNames.
	Skip (see page 17272)	This is Skip, a member of class IEnumNames.

104.1.1.2.1 IEnumNames.Clone Method

Pascal

```
function Clone(out enm: IEnumNames): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Clone(IEnumNames enm);
```

Description

This is Clone, a member of class IEnumNames.

104.1.1.2.2 IEnumNames.Next Method

Pascal

```
function Next(celt: UINT; out rgname: WideString; out pceltFetched: UINT): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Next(UINT celt, BSTR rgname, UINT pceltFetched);
```

Description

This is Next, a member of class IEnumNames.

104.1.1.2.3 IEnumNames.Reset Method

Pascal

```
function Reset: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Reset();
```

Description

This is Reset, a member of class IEnumNames.

104.1.1.2.4 IEnumNames.Skip Method

Pascal

```
function Skip(celt: UINT): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Skip(UINT celt);
```

Description

This is Skip, a member of class IEnumNames.

104.1.2 IGetContextProperties Interface

Class Hierarchy

Pascal

```
IGetContextProperties = interface(IUnknown);
```

C++

```
__interface IGetContextProperties : public IUnknown;
```

File

Mtx

Description

This is class Mtx.IGetContextProperties.

104.1.2.1 IGetContextProperties Members

The following tables list the members exposed by IGetContextProperties.

Public Methods

	Name	Description
	Count (see page 17273)	This is Count, a member of class IGetContextProperties.
	EnumNames (see page 17274)	This is EnumNames, a member of class IGetContextProperties.
	GetProperty (see page 17274)	This is GetProperty, a member of class IGetContextProperties.

104.1.2.2 IGetContextProperties Methods

The methods of the IGetContextProperties class are listed here.

Public Methods

	Name	Description
	Count (see page 17273)	This is Count, a member of class IGetContextProperties.
	EnumNames (see page 17274)	This is EnumNames, a member of class IGetContextProperties.
	GetProperty (see page 17274)	This is GetProperty, a member of class IGetContextProperties.

104.1.2.2.1 IGetContextProperties.Count Method

Pascal

```
function Count: Integer; safecall;
```

C++

```
__fastcall int Count();
```

Description

This is Count, a member of class IGetContextProperties.

104.1.2.2.2 IGetContextProperties.EnumNames Method

Pascal

```
function EnumNames: IEnumNames; safecall;
```

C++

```
__fastcall IEnumNames EnumNames();
```

Description

This is EnumNames, a member of class IGetContextProperties.

104.1.2.2.3 IGetContextProperties.GetProperty Method

Pascal

```
function GetProperty(const name: WideString): OleVariant; safecall;
```

C++

```
__fastcall OleVariant GetProperty(const BSTR name);
```

Description

This is GetProperty, a member of class IGetContextProperties.

104.1.3 IObjectContext Interface

Class Hierarchy

Pascal

```
IObjectContext = interface(IUnknown);
```

C++

```
__interface IObjectContext : public IUnknown;
```

File

Mtx

Description

This is class Mtx.IObjectContext.

104.1.3.1 IObjectContext Members

The following tables list the members exposed by IObjectContext.

Public Methods

	Name	Description
	CreateInstance (see page 17275)	This is CreateInstance, a member of class IObjectContext.
	DisableCommit (see page 17275)	This is DisableCommit, a member of class IObjectContext.
	EnableCommit (see page 17276)	This is EnableCommit, a member of class IObjectContext.
	IsCallerInRole (see page 17276)	This is IsCallerInRole, a member of class IObjectContext.
	IsInTransaction (see page 17276)	This is IsInTransaction, a member of class IObjectContext.
	IsSecurityEnabled (see page 17276)	This is IsSecurityEnabled, a member of class IObjectContext.
	SetAbort (see page 17276)	This is SetAbort, a member of class IObjectContext.
	SetComplete (see page 17277)	This is SetComplete, a member of class IObjectContext.

104.1.3.2 IObjectContext Methods

The methods of the IObjectContext class are listed here.

Public Methods

	Name	Description
	CreateInstance (see page 17275)	This is CreateInstance, a member of class IObjectContext.
	DisableCommit (see page 17275)	This is DisableCommit, a member of class IObjectContext.
	EnableCommit (see page 17276)	This is EnableCommit, a member of class IObjectContext.
	IsCallerInRole (see page 17276)	This is IsCallerInRole, a member of class IObjectContext.
	IsInTransaction (see page 17276)	This is IsInTransaction, a member of class IObjectContext.
	IsSecurityEnabled (see page 17276)	This is IsSecurityEnabled, a member of class IObjectContext.
	SetAbort (see page 17276)	This is SetAbort, a member of class IObjectContext.
	SetComplete (see page 17277)	This is SetComplete, a member of class IObjectContext.

104.1.3.2.1 IObjectContext.CreateInstance Method

Pascal

```
function CreateInstance(const cid: TGUID; const rid: TGUID; out pv): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT CreateInstance(const TGUID cid, const TGUID rid,  pv);
```

Description

This is CreateInstance, a member of class IObjectContext.

104.1.3.2.2 IObjectContext.DisableCommit Method

Pascal

```
procedure DisableCommit; safecall;
```

C++

```
__fastcall DisableCommit();
```

Description

This is DisableCommit, a member of class IObjectContext.

104.1.3.2.3 IObjectContext.EnableCommit Method

Pascal

```
procedure EnableCommit; safecall;
```

C++

```
__fastcall EnableCommit();
```

Description

This is EnableCommit, a member of class IObjectContext.

104.1.3.2.4 IObjectContext.IsCallerInRole Method

Pascal

```
function IsCallerInRole(const bstrRole: WideString): Bool; safecall;
```

C++

```
__fastcall Bool IsCallerInRole(const BSTR bstrRole);
```

Description

This is IsCallerInRole, a member of class IObjectContext.

104.1.3.2.5 IObjectContext.IsInTransaction Method

Pascal

```
function IsInTransaction: Bool; stdcall;
```

C++

```
__fastcall __stdcall Bool IsInTransaction();
```

Description

This is IsInTransaction, a member of class IObjectContext.

104.1.3.2.6 IObjectContext.IsSecurityEnabled Method

Pascal

```
function IsSecurityEnabled: Bool; stdcall;
```

C++

```
__fastcall __stdcall Bool IsSecurityEnabled();
```

Description

This is IsSecurityEnabled, a member of class IObjectContext.

104.1.3.2.7 IObjectContext.SetAbort Method

Pascal

```
procedure SetAbort; safecall;
```

C++

```
__fastcall SetAbort();
```

Description

This is SetAbort, a member of class IObjectContext.

104.1.3.2.8 IObjectContext.SetComplete Method

Pascal

```
procedure SetComplete; safecall;
```

C++

```
__fastcall SetComplete();
```

Description

This is SetComplete, a member of class IObjectContext.

104.1.4 IObjectContextActivity Interface

Class Hierarchy

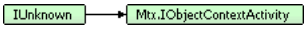

Pascal

```
IObjectContextActivity = interface(IUnknown);
```

C++

```
__interface IObjectContextActivity : public IUnknown;
```

File

Mtx

Description

This is class Mtx.IObjectContextActivity.

104.1.4.1 IObjectContextActivity Members

The following tables list the members exposed by IObjectContextActivity.

Public Methods

	Name	Description
	GetActivityId (see page 17278)	This is GetActivityId, a member of class IObjectContextActivity.

104.1.4.2 IObjectContextActivity Methods

The methods of the IObjectContextActivity class are listed here.

Public Methods

	Name	Description
	GetActivityId (see page 17278)	This is GetActivityId, a member of class IObjectContextActivity.

104.1.4.2.1 IObjectContextActivity.GetActivityId Method

Pascal

```
function GetActivityId: TGuid; safecall;
```

C++

```
__fastcall TGuid GetActivityId();
```

Description

This is GetActivityId, a member of class IObjectContextActivity.

104.1.5 IObjectControl Interface

Class Hierarchy

Pascal

```
IObjectControl = interface(IUnknown);
```

C++

```
__interface IObjectControl : public IUnknown;
```

File

Mtx

Description

This is class Mtx.IObjectControl.

104.1.5.1 IObjectControl Members

The following tables list the members exposed by IObjectControl.

Public Methods

	Name	Description
	Activate (see page 17279)	This is Activate, a member of class IObjectControl.
	CanBePooled (see page 17279)	This is CanBePooled, a member of class IObjectControl.
	Deactivate (see page 17279)	This is Deactivate, a member of class IObjectControl.

104.1.5.2 IObjectControl Methods

The methods of the IObjectControl class are listed here.

Public Methods

	Name	Description
	Activate (see page 17279)	This is Activate, a member of class IObjectControl.
	CanBePooled (see page 17279)	This is CanBePooled, a member of class IObjectControl.
	Deactivate (see page 17279)	This is Deactivate, a member of class IObjectControl.

104.1.5.2.1 IObjectControl.Activate Method

Pascal

```
procedure Activate; safecall;
```

C++

```
__fastcall Activate();
```

Description

This is Activate, a member of class IObjectControl.

104.1.5.2.2 IObjectControl.CanBePooled Method

Pascal

```
function CanBePooled: Bool; stdcall;
```

C++

```
__fastcall __stdcall Bool CanBePooled();
```

Description

This is CanBePooled, a member of class IObjectControl.

104.1.5.2.3 IObjectControl.Deactivate Method

Pascal

```
procedure Deactivate; stdcall;
```

C++

```
__fastcall __stdcall Deactivate();
```

Description

This is Deactivate, a member of class IObjectControl.

104.1.6 ISecurityProperty Interface

Class Hierarchy

Pascal

```
ISecurityProperty = interface(IUnknown);
```

C++

```
__interface ISecurityProperty : public IUnknown;
```

File

Mtx

Description

This is class Mtx.ISecurityProperty.

104.1.6.1 ISecurityProperty Members

The following tables list the members exposed by ISecurityProperty.

Public Methods

	Name	Description
	GetDirectCallerSID (see page 17280)	This is GetDirectCallerSID, a member of class ISecurityProperty.
	GetDirectCreatorSID (see page 17280)	This is GetDirectCreatorSID, a member of class ISecurityProperty.
	GetOriginalCallerSID (see page 17280)	This is GetOriginalCallerSID, a member of class ISecurityProperty.
	GetOriginalCreatorSID (see page 17281)	This is GetOriginalCreatorSID, a member of class ISecurityProperty.
	ReleaseSID (see page 17281)	This is ReleaseSID, a member of class ISecurityProperty.

104.1.6.2 ISecurityProperty Methods

The methods of the ISecurityProperty class are listed here.

Public Methods

	Name	Description
	GetDirectCallerSID (see page 17280)	This is GetDirectCallerSID, a member of class ISecurityProperty.
	GetDirectCreatorSID (see page 17280)	This is GetDirectCreatorSID, a member of class ISecurityProperty.
	GetOriginalCallerSID (see page 17280)	This is GetOriginalCallerSID, a member of class ISecurityProperty.
	GetOriginalCreatorSID (see page 17281)	This is GetOriginalCreatorSID, a member of class ISecurityProperty.
	ReleaseSID (see page 17281)	This is ReleaseSID, a member of class ISecurityProperty.

104.1.6.2.1 ISecurityProperty.GetDirectCallerSID Method

Pascal

```
function GetDirectCallerSID(out sid: PSID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetDirectCallerSID(PSID sid);
```

Description

This is GetDirectCallerSID, a member of class ISecurityProperty.

104.1.6.2.2 ISecurityProperty.GetDirectCreatorSID Method

Pascal

```
function GetDirectCreatorSID(out sid: PSID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetDirectCreatorSID(PSID sid);
```

Description

This is GetDirectCreatorSID, a member of class ISecurityProperty.

104.1.6.2.3 ISecurityProperty.GetOriginalCallerSID Method

Pascal

```
function GetOriginalCallerSID(out sid: PSID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetOriginalCallerSID(PSID sid);
```

Description

This is GetOriginalCallerSID, a member of class ISecurityProperty.

104.1.6.2.4 ISecurityProperty.GetOriginalCreatorSID Method

Pascal

```
function GetOriginalCreatorSID(out sid: PSID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetOriginalCreatorSID(PSID sid);
```

Description

This is GetOriginalCreatorSID, a member of class ISecurityProperty.

104.1.6.2.5 ISecurityProperty.ReleaseSID Method

Pascal

```
function ReleaseSID(sid: PSID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT ReleaseSID(PSID sid);
```

Description

This is ReleaseSID, a member of class ISecurityProperty.

104.1.7 ISharedProperty Interface

Class Hierarchy

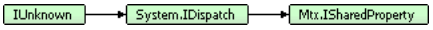

Pascal

```
ISharedProperty = interface(IDispatch);
```

C++

```
__interface ISharedProperty : public IDispatch;
```

File

Mtx

Description

This is class Mtx.ISharedProperty.

104.1.7.1 ISharedProperty Members

The following tables list the members exposed by ISharedProperty.

Public Methods

	Name	Description
	Get_Value (see page 17282)	This is Get_Value, a member of class ISharedProperty.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	Set_Value (see page 17282)	This is Set_Value, a member of class ISharedProperty.

Public Properties

	Name	Description
	Value (see page 17283)	This is Value, a member of class ISharedProperty.

104.1.7.2 ISharedProperty Methods

The methods of the ISharedProperty class are listed here.

Public Methods

	Name	Description
	Get_Value (see page 17282)	This is Get_Value, a member of class ISharedProperty.
	Set_Value (see page 17282)	This is Set_Value, a member of class ISharedProperty.

104.1.7.2.1 ISharedProperty.Get_Value Method

Pascal

```
function Get_Value: OleVariant; safecall;
```

C++

```
__fastcall OleVariant Get_Value();
```

Description

This is Get_Value, a member of class ISharedProperty.

104.1.7.2.2 ISharedProperty.Set_Value Method

Pascal

```
procedure Set_Value(pVal: OleVariant); safecall;
```

C++

```
__fastcall Set_Value(OleVariant pVal);
```

Description

This is Set_Value, a member of class ISharedProperty.

104.1.7.3 ISharedProperty Properties

The properties of the ISharedProperty class are listed here.

Public Properties

	Name	Description
	Value (see page 17283)	This is Value, a member of class ISharedProperty.

104.1.7.3.1 ISharedProperty.Value Property

Pascal

```
property Value: OleVariant;
```

C++

```
__property OleVariant Value;
```

Description

This is Value, a member of class ISharedProperty.

104.1.8 ISharedPropertyGroup Interface

Class Hierarchy

Pascal

```
ISharedPropertyGroup = interface(IDispatch);
```

C++

```
__interface ISharedPropertyGroup : public IDispatch;
```

File

Mtx

Description

This is class Mtx.ISharedPropertyGroup.

104.1.8.1 ISharedPropertyGroup Members

The following tables list the members exposed by ISharedPropertyGroup.

Public Methods

	Name	Description
	CreateProperty (see page 17284)	This is CreateProperty, a member of class ISharedPropertyGroup.
	CreatePropertyByPosition (see page 17284)	This is CreatePropertyByPosition, a member of class ISharedPropertyGroup.
	Get_PropertyByName (see page 17284)	This is Get_PropertyByName, a member of class ISharedPropertyGroup.
	Get_PropertyByPosition (see page 17285)	This is Get_PropertyByPosition, a member of class ISharedPropertyGroup.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.

Public Properties

	Name	Description
R	PropertyByName (see page 17285)	This is PropertyByName, a member of class ISharedPropertyGroup.
R	PropertyByPosition (see page 17285)	This is PropertyByPosition, a member of class ISharedPropertyGroup.

104.1.8.2 ISharedPropertyGroup Methods

The methods of the ISharedPropertyGroup class are listed here.

Public Methods

	Name	Description
	CreateProperty (see page 17284)	This is CreateProperty, a member of class ISharedPropertyGroup.
	CreatePropertyByPosition (see page 17284)	This is CreatePropertyByPosition, a member of class ISharedPropertyGroup.
	Get_PropertyByName (see page 17284)	This is Get_PropertyByName, a member of class ISharedPropertyGroup.
	Get_PropertyByPosition (see page 17285)	This is Get_PropertyByPosition, a member of class ISharedPropertyGroup.

104.1.8.2.1 ISharedPropertyGroup.CreateProperty Method

Pascal

```
function CreateProperty(const Name: WideString; out fExists: WordBool): ISharedProperty;
safecall;
```

C++

```
__fastcall ISharedProperty CreateProperty(const BSTR Name, WordBool fExists);
```

Description

This is CreateProperty, a member of class ISharedPropertyGroup.

104.1.8.2.2 ISharedPropertyGroup.CreatePropertyByPosition Method

Pascal

```
function CreatePropertyByPosition(Index: Integer; out fExists: WordBool): ISharedProperty;
safecall;
```

C++

```
__fastcall ISharedProperty CreatePropertyByPosition(int Index, WordBool fExists);
```

Description

This is CreatePropertyByPosition, a member of class ISharedPropertyGroup.

104.1.8.2.3 ISharedPropertyGroup.Get_PropertyByName Method

Pascal

```
function Get_PropertyByName(const Name: WideString): ISharedProperty; safecall;
```

C++

```
__fastcall ISharedProperty Get_PropertyByName(const BSTR Name);
```

Description

This is Get_PropertyByName, a member of class ISharedPropertyGroup.

104.1.8.2.4 ISharedPropertyGroup.Get_PropertyByPosition Method

Pascal

```
function Get_PropertyByPosition(Index: Integer): ISharedProperty; safecall;
```

C++

```
__fastcall ISharedProperty Get_PropertyByPosition(int Index);
```

Description

This is Get_PropertyByPosition, a member of class ISharedPropertyGroup.

104.1.8.3 ISharedPropertyGroup Properties

The properties of the ISharedPropertyGroup class are listed here.

Public Properties

	Name	Description
R	PropertyByName (see page 17285)	This is PropertyByName, a member of class ISharedPropertyGroup.
R	PropertyByPosition (see page 17285)	This is PropertyByPosition, a member of class ISharedPropertyGroup.

104.1.8.3.1 ISharedPropertyGroup.PropertyByName Property

Pascal

```
property PropertyByName [const Name: WideString]: ISharedProperty;
```

C++

```
__property ISharedProperty PropertyByName[BSTR const Name];
```

Description

This is PropertyByName, a member of class ISharedPropertyGroup.

104.1.8.3.2 ISharedPropertyGroup.PropertyByPosition Property

Pascal

```
property PropertyByPosition [Index: Integer]: ISharedProperty;
```

C++

```
__property ISharedProperty PropertyByPosition[int Index];
```

Description

This is PropertyByPosition, a member of class ISharedPropertyGroup.

104.1.9 ISharedPropertyGroupManager Interface

Class Hierarchy

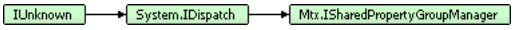

Pascal

```
ISharedPropertyGroupManager = interface(IDispatch);
```

C++

```
__interface ISharedPropertyGroupManager : public IDispatch;
```

File

Mtx

Description

This is class Mtx.ISharedPropertyGroupManager.

104.1.9.1 ISharedPropertyGroupManager Members

The following tables list the members exposed by ISharedPropertyGroupManager.

Public Methods

	Name	Description
	CreatePropertyGroup (see page 17286)	This is CreatePropertyGroup, a member of class ISharedPropertyGroupManager.
	Get__NewEnum (see page 17287)	This is Get__NewEnum, a member of class ISharedPropertyGroupManager.
	Get_Group (see page 17287)	This is Get_Group, a member of class ISharedPropertyGroupManager.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.

Public Properties

	Name	Description
R	_NewEnum (see page 17287)	This is _NewEnum, a member of class ISharedPropertyGroupManager.
R	Group (see page 17287)	This is Group, a member of class ISharedPropertyGroupManager.

104.1.9.2 ISharedPropertyGroupManager Methods

The methods of the ISharedPropertyGroupManager class are listed here.

Public Methods

	Name	Description
	CreatePropertyGroup (see page 17286)	This is CreatePropertyGroup, a member of class ISharedPropertyGroupManager.
	Get__NewEnum (see page 17287)	This is Get__NewEnum, a member of class ISharedPropertyGroupManager.
	Get_Group (see page 17287)	This is Get_Group, a member of class ISharedPropertyGroupManager.

104.1.9.2.1 ISharedPropertyGroupManager.CreatePropertyGroup Method

Pascal

```
function CreatePropertyGroup(const Name: WideString; var LockMode: Integer; var RelMode:
Integer; out fExists: WordBool): ISharedPropertyGroup; safecall;
```

C++

```
__fastcall ISharedPropertyGroup CreatePropertyGroup(const BSTR Name, int LockMode, int
RelMode, WordBool fExists);
```

Description

This is CreatePropertyGroup, a member of class ISharedPropertyGroupManager.

104.1.9.2.2 ISharedPropertyGroupManager.Get__NewEnum Method

Pascal

```
function Get__NewEnum: IUnknown; safecall;
```

C++

```
__fastcall IUnknown Get__NewEnum();
```

Description

This is Get__NewEnum, a member of class ISharedPropertyGroupManager.

104.1.9.2.3 ISharedPropertyGroupManager.Get_Group Method

Pascal

```
function Get_Group(const Name: WideString): ISharedPropertyGroup; safecall;
```

C++

```
__fastcall ISharedPropertyGroup Get_Group(const BSTR Name);
```

Description

This is Get_Group, a member of class ISharedPropertyGroupManager.

104.1.9.3 ISharedPropertyGroupManager Properties

The properties of the ISharedPropertyGroupManager class are listed here.

Public Properties

	Name	Description
R	_NewEnum (see page 17287)	This is _NewEnum, a member of class ISharedPropertyGroupManager.
R	Group (see page 17287)	This is Group, a member of class ISharedPropertyGroupManager.

104.1.9.3.1 ISharedPropertyGroupManager._NewEnum Property

Pascal

```
property _NewEnum: IUnknown;
```

C++

```
__property IUnknown _NewEnum;
```

Description

This is _NewEnum, a member of class ISharedPropertyGroupManager.

104.1.9.3.2 ISharedPropertyGroupManager.Group Property

Pascal

```
property Group [const Name: WideString]: ISharedPropertyGroup;
```

C++

```
__property ISharedPropertyGroup Group[BSTR const Name];
```

Description

This is Group, a member of class ISharedPropertyGroupManager.

104.1.10 ITransactionContext Interface

Class Hierarchy

Pascal

```
ITransactionContext = interface(IDispatch);
```

C++

```
__interface ITransactionContext : public IDispatch;
```

File

Mtx

Description

This is class Mtx.ITransactionContext.

104.1.10.1 ITransactionContext Members

The following tables list the members exposed by ITransactionContext.

Public Methods

	Name	Description
	Abort (see page 17289)	This is Abort, a member of class ITransactionContext.
	Commit (see page 17289)	This is Commit, a member of class ITransactionContext.
	CreateInstance (see page 17289)	This is CreateInstance, a member of class ITransactionContext.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.

104.1.10.2 ITransactionContext Methods

The methods of the ITransactionContext class are listed here.

Public Methods

	Name	Description
	Abort (see page 17289)	This is Abort, a member of class ITransactionContext.
	Commit (see page 17289)	This is Commit, a member of class ITransactionContext.
	CreateInstance (see page 17289)	This is CreateInstance, a member of class ITransactionContext.

104.1.10.2.1 ITransactionContext.Abort Method

Pascal

```
procedure Abort; safecall;
```

C++

```
__fastcall Abort();
```

Description

This is Abort, a member of class ITransactionContext.

104.1.10.2.2 ITransactionContext.Commit Method

Pascal

```
procedure Commit; safecall;
```

C++

```
__fastcall Commit();
```

Description

This is Commit, a member of class ITransactionContext.

104.1.10.2.3 ITransactionContext.CreateInstance Method

Pascal

```
function CreateInstance(const pszProgId: WideString): OleVariant; safecall;
```

C++

```
__fastcall OleVariant CreateInstance(const BSTR pszProgId);
```

Description

This is CreateInstance, a member of class ITransactionContext.

104.1.11 ITransactionContextEx Interface

Class Hierarchy

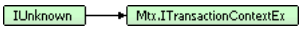

Pascal

```
ITransactionContextEx = interface(IUnknown);
```

C++

```
__interface ITransactionContextEx : public IUnknown;
```

File

Mtx

Description

This is class Mtx.ITransactionContextEx.

104.1.11.1 ITransactionContextEx Members

The following tables list the members exposed by ITransactionContextEx.

Public Methods

	Name	Description
	Abort (see page 17290)	This is Abort, a member of class ITransactionContextEx.
	Commit (see page 17290)	This is Commit, a member of class ITransactionContextEx.
	CreateInstance (see page 17290)	This is CreateInstance, a member of class ITransactionContextEx.

104.1.11.2 ITransactionContextEx Methods

The methods of the ITransactionContextEx class are listed here.

Public Methods

	Name	Description
	Abort (see page 17290)	This is Abort, a member of class ITransactionContextEx.
	Commit (see page 17290)	This is Commit, a member of class ITransactionContextEx.
	CreateInstance (see page 17290)	This is CreateInstance, a member of class ITransactionContextEx.

104.1.11.2.1 ITransactionContextEx.Abort Method

Pascal

```
function Abort: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Abort();
```

Description

This is Abort, a member of class ITransactionContextEx.

104.1.11.2.2 ITransactionContextEx.Commit Method

Pascal

```
function Commit: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Commit();
```

Description

This is Commit, a member of class ITransactionContextEx.

104.1.11.2.3 ITransactionContextEx.CreateInstance Method

Pascal

```
function CreateInstance(const cid: TGUID; const rid: TGUID; out pv): HRESULT; stdcall;
```

C++

```
__fastcall __stdcall HRESULT CreateInstance(const TGUID cid, const TGUID rid,  pv);
```

Description

This is CreateInstance, a member of class ITransactionContextEx.

104.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CreateSharedPropertyGroup (see page 17292)	Returns the interface for a SharedPropertyGroup that allows MTS objects within a server process to share state.
	CreateSharedPropertyGroupManager (see page 17293)	Returns an interface for a new shared property group manager that creates and coordinates shared properties of MTS objects.
	CreateTransactionContext (see page 17293)	Starts an MTS transaction.
	CreateTransactionContextEx (see page 17294)	Instantiates a TransactionContextEx object, which marks the beginning of a transaction.
	GetObjectContext (see page 17295)	Fetches the IObjectContext (see page 17274) interface for the current thread.
	SafeRef (see page 17295)	Returns a reference to a transactional object that is safe to pass outside its context.

104.2.1 Mtx.CreateSharedPropertyGroup Function

Returns the interface for a SharedPropertyGroup that allows MTS objects within a server process to share state.

Pascal

```
function CreateSharedPropertyGroup(const Name: WideString): ISharedPropertyGroup;
```

C++

```
ISharedPropertyGroup CreateSharedPropertyGroup(const BSTR Name);
```

File

Mtx

Description

CreateSharedPropertyGroup returns an interface to a new shared property group. If a property group of the specified name exists, it returns a reference to the existing shared property group. Use this function to create and access the shared properties of a shared property group.

CreateSharedPropertyGroup sets up a SharedPropertyGroup with the default values of LockSetGet (see page 17298) for locking (isolation) mode, and Process (see page 17302) for release mode. LockSetGet (see page 17298) locks a property during every get or set operation on a shared property, making each operation atomic. This ensures that two clients cannot read or write to the same property at the same time, but it does not prevent other clients from concurrently accessing other properties in the same group. The Process (see page 17302) release mode ensures that the property group is not destroyed until the process in which it was created has terminated. (Objects that hold references on a property group must still call Release on their references).

To override these defaults, call CreateSharedPropertyGroupManager (see page 17293) and use the resulting interface to create a property group.

ISharedPropertyGroup (see page 17283) has several methods that can be used to share state among multiple objects in a server process. See ISharedPropertyGroup (see page 17283) in the MTS documentation for a list of methods.

By using shared property groups, you eliminate name collisions because they establish unique name spaces for the shared

properties they contain. The Shared Property Manager also implements locks and semaphores to protect shared properties from simultaneous access, which could result in lost updates and could leave the properties in an unpredictable state.

Note: Only objects running in the same process can share properties. If you want instances of different components to share properties, you have to install the components in the same MTS package. Because there is a risk that administrators will move components from one package to another, it's safest to limit the use of a shared property group to instances of components that are defined in the same DLL.

See Also

CreateSharedPropertyGroupManager

104.2.2 Mtx.CreateSharedPropertyGroupManager Function

Returns an interface for a new shared property group manager that creates and coordinates shared properties of MTS objects.

Pascal

```
function CreateSharedPropertyGroupManager: ISharedPropertyGroupManager;
```

C++

```
ISharedPropertyGroupManager CreateSharedPropertyGroupManager();
```

File

Mtx

Description

Call CreateSharedPropertyGroupManager before creating shared property groups if you are not using the CreateSharedPropertyGroup (see page 17292) function. The ISharedPropertyGroupManager (see page 17285) interface lets you create and access shared properties with more control that is provided by CreateSharedPropertyGroup (see page 17292).

By using shared property groups, you eliminate name collisions because they establish unique name spaces for the shared properties they contain. The Shared Property Manager also implements locks and semaphores to protect shared properties from simultaneous access, which could result in lost updates and could leave the properties in an unpredictable state.

For more information about the ISharedPropertyGroupManager (see page 17285) interface, see the MTS documentation.

See Also

CreateSharedPropertyGroup

104.2.3 Mtx.CreateTransactionContext Function

Starts an MTS transaction.

Pascal

```
function CreateTransactionContext: ITransactionContext;
```

C++

```
ITransactionContext CreateTransactionContext();
```

File

Mtx

Description

CreateTransactionContext allows client applications to control MTS transactions. CreateTransactionContext returns an interface to the new transaction context. Use this interface to enlist MTS server objects in the transaction, and to commit or roll back the transaction when done.

See Also

CreateTransactionContextEx

TMtsAutoObject

TMtsAutoObject

TMtsDataModule

TMtsDataModule

104.2.4 Mtx.CreateTransactionContextEx Function

Instantiates a TransactionContextEx object, which marks the beginning of a transaction.

Pascal

```
function CreateTransactionContextEx: ITransactionContextEx;
```

C++

```
ITransactionContextEx CreateTransactionContextEx();
```

File

Mtx

Description

MTS base clients use CreateTransactionEx to combine the work of one or more MTS objects into an atomic transaction and to commit or abort the transaction. Clients initiate a transaction by instantiating a CreateTransactionEx object. Clients end a transaction by calling Complete or Abort on the object.

Method	Description
Abort	Aborts the work of all MTS objects participating in the current transaction. The transaction is completed on return from this method.
Commit	Attempts to commit the work of all MTS objects participating in the current transaction. If any of the MTS objects participating in the transaction call SetAbort or DisableCommit, or if a system error occurs, the transaction is aborted. Otherwise, the transaction is committed. In either case, the transaction is completed on return from this method.
CreateInstance	Instantiates another MTS object. If the component that provides the object is configured to support or require a transaction, then the new object runs under the transaction of the TransactionContextEx object.

See Also

CreateTransactionContext

TMtsAutoObject

TMtsAutoObject

TMtsDataModule

TMtsDataModule

104.2.5 Mtx.GetObjectContext Function

Fetches the IObjectContext (see page 17274) interface for the current thread.

Pascal

```
function GetObjectContext: IObjectContext;
```

C++

```
IObjectContext GetObjectContext();
```

File

Mtx

Description

When a COM object is created as part of the MTS environment or of a COM+ package (that is, a transactional object), a corresponding context object is also created. This context object exists for the entire lifetime of its transactional object, across one or more reactivation cycles. The object context keeps track of the object during deactivation. Call GetObjectContext to obtain an IObjectContext (see page 17274) interface pointer for the context object associated with the current thread.

Pass a pointer that receives the IObjectContext (see page 17274) interface pointer as ppInstanceContext.

GetObjectContext returns S_OK if it returns a pointer to the context object. Otherwise, it returns an error code.

Note: C++Builder automatically fetches the IObjectContext (see page 17274) interface when a transactional object is activated. The interface pointer is stored as the m_spObjectContext member of the transactional object's class.

Warning: When writing a transactional object that uses VCL classes, use the Get_ObjectContext method of TMtsDll instead. ComSvcs.h defines GetObjectContext using a macro so that it calls COM+ or MTS, depending on the system you use to build the transactional object. This macro interferes with the GetObjectContext method of some VCL classes, and so it is undefined when you include the VCL headers.

See Also

TMtsDll

104.2.6 Mtx.SafeRef Function

Returns a reference to a transactional object that is safe to pass outside its context.

Pascal

```
function SafeRef(const rid: TGUID; Unk: IUnknown): Pointer;
```

C++

```
void * SafeRef(const TGUID rid, IUnknown Unk);
```

File

Mtx

Description

SafeRef encapsulates the SafeRef MTS function that passes in

a reference to the interface ID of the interface (rid) that the current object wants to pass to another object or client

and

a reference to an interface on the current object (pUnk).

SafeRef returns a pointer to the interface specified in the rid parameter that is safe to pass outside the current object's context.

When using COM+, it is always safe to pass a self-reference to a client or other object. Thus, when you call SafeRef in an application that is running under a version of Windows that implemnts COM+, SafeRef simply returns the pointer passed as pUnk. Thus, it is not necessary to call SafeRef for COM+ applications. It will, however, do no harm in an application that is used under both MTS and COM+.

Under MTS, and should always call SafeRef before passing a self-reference to a client or other object and then pass the reference returned by this call. Under MTS, it is not safe for an object to pass a self pointer (Delphi) or this pointer (C++), or a self-reference obtained through an internal call to QueryInterface, to a client or to any other object. Once such a reference is passed outside the object's context, it is no longer a valid reference.

Calling SafeRef on a reference that is already safe returns the safe reference unchanged, except that the reference count on the interface is incremented.

When a client calls QueryInterface on a reference that is safe, MTS automatically ensures that the reference returned to the client is also a safe reference.

An object that obtains a safe reference must release the safe reference when it is finished with it.

Under MTS, safe references have different pointer values than their unsafe counterparts. For example, self or this and the safe version of self or this do not have the same value. It is important to be aware of this when testing whether two pointers refer to the same object. Calling QueryInterface for IID_IUnknown on each of the pointers and comparing the value of the returned pointers may result in the wrong conclusion. It is possible that both pointers refer to the same object, but that one is a safe reference and the other is not. If both references are safe references, they can be compared in the usual way. This is only a consideration for MTS objects, because clients should never have access to unsafe references.

Note: When writing a transactional object that may be installed under either MTS or COM+, use the SafeRef method of TMtsDll instead. TMtsDll automatically checks the current environment and returns the appropriate reference.

See Also

TMtsAutoObject

TMTSDll

104.3 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CLASS_SharedPropertyGroupManager (see page 17297)	This is constant Mtx.CLASS_SharedPropertyGroupManager.
CLASS_TransactionContext (see page 17297)	This is constant Mtx.CLASS_TransactionContext.
CLASS_TransactionContextEx (see page 17298)	This is constant Mtx.CLASS_TransactionContextEx.
LockMethod (see page 17298)	This is constant Mtx.LockMethod.
LockSetGet (see page 17298)	This is constant Mtx.LockSetGet.
mtsErrCtxAborted (see page 17299)	This is constant Mtx.mtsErrCtxAborted.
mtsErrCtxAborting (see page 17299)	This is constant Mtx.mtsErrCtxAborting.
mtsErrCtxActivityTimeout (see page 17299)	This is constant Mtx.mtsErrCtxActivityTimeout.
mtsErrCtxNoContext (see page 17300)	This is constant Mtx.mtsErrCtxNoContext.
mtsErrCtxNoSecurity (see page 17300)	This is constant Mtx.mtsErrCtxNoSecurity.
mtsErrCtxNotRegistered (see page 17300)	This is constant Mtx.mtsErrCtxNotRegistered.
mtsErrCtxOldReference (see page 17301)	This is constant Mtx.mtsErrCtxOldReference.
mtsErrCtxRoleNotFound (see page 17301)	This is constant Mtx.mtsErrCtxRoleNotFound.
mtsErrCtxTMNotAvailable (see page 17301)	This is constant Mtx.mtsErrCtxTMNotAvailable.
mtsErrCtxWrongThread (see page 17302)	This is constant Mtx.mtsErrCtxWrongThread.
Process (see page 17302)	This is constant Mtx.Process.
Standard (see page 17302)	This is constant Mtx.Standard.

104.3.1 Mtx.CLASS_SharedPropertyGroupManager Constant

Pascal

```
CLASS_SharedPropertyGroupManager: TGUID = '{2A005C11-A5DE-11CF-9E66-00AA00A3F464}';
```

C++

```
TGUID CLASS_SharedPropertyGroupManager = '{2A005C11-A5DE-11CF-9E66-00AA00A3F464}';
```

File

Mtx

Description

This is constant Mtx.CLASS_SharedPropertyGroupManager.

104.3.2 Mtx.CLASS_TransactionContext Constant

Pascal

```
CLASS_TransactionContext: TGUID = '{7999FC25-D3C6-11CF-ACAB-00A024A55AEF}';
```

C++

```
TGUID CLASS_TransactionContext = '{7999FC25-D3C6-11CF-ACAB-00A024A55AEF}';
```

File

Mtx

Description

This is constant Mtx.CLASS_TransactionContext.

104.3.3 Mtx.CLASS_TransactionContextEx Constant

Pascal

```
CLASS_TransactionContextEx: TGUID = '{5CB66670-D3D4-11CF-ACAB-00A024A55AEF}';
```

C++

```
TGUID CLASS_TransactionContextEx = '{5CB66670-D3D4-11CF-ACAB-00A024A55AEF}';
```

File

Mtx

Description

This is constant Mtx.CLASS_TransactionContextEx.

104.3.4 Mtx.LockMethod Constant

Pascal

```
LockMethod = $00000001;
```

C++

```
LockMethod = 0x00000001;
```

File

Mtx

Description

This is constant Mtx.LockMethod.

104.3.5 Mtx.LockSetGet Constant

Pascal

```
LockSetGet = $00000000;
```

C++

```
LockSetGet = 0x00000000;
```

File

Mtx

Description

This is constant Mtx.LockSetGet.

104.3.6 Mtx.mtsErrCtxAborted Constant

Pascal

```
mtsErrCtxAborted = $8004E002;
```

C++

```
mtsErrCtxAborted = 0x8004E002;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxAborted.

104.3.7 Mtx.mtsErrCtxAborting Constant

Pascal

```
mtsErrCtxAborting = $8004E003;
```

C++

```
mtsErrCtxAborting = 0x8004E003;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxAborting.

104.3.8 Mtx.mtsErrCtxActivityTimeout Constant

Pascal

```
mtsErrCtxActivityTimeout = $8004E006;
```

C++

```
mtsErrCtxActivityTimeout = 0x8004E006;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxActivityTimeout.

104.3.9 Mtx.mtsErrCtxNoContext Constant

Pascal

```
mtsErrCtxNoContext = $8004E004;
```

C++

```
mtsErrCtxNoContext = 0x8004E004;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxNoContext.

104.3.10 Mtx.mtsErrCtxNoSecurity Constant

Pascal

```
mtsErrCtxNoSecurity = $8004E00D;
```

C++

```
mtsErrCtxNoSecurity = 0x8004E00D;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxNoSecurity.

104.3.11 Mtx.mtsErrCtxNotRegistered Constant

Pascal

```
mtsErrCtxNotRegistered = $8004E005;
```

C++

```
mtsErrCtxNotRegistered = 0x8004E005;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxNotRegistered.

104.3.12 Mtx.mtsErrCtxOldReference Constant

Pascal

```
mtsErrCtxOldReference = $8004E007;
```

C++

```
mtsErrCtxOldReference = 0x8004E007;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxOldReference.

104.3.13 Mtx.mtsErrCtxRoleNotFound Constant

Pascal

```
mtsErrCtxRoleNotFound = $8004E00C;
```

C++

```
mtsErrCtxRoleNotFound = 0x8004E00C;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxRoleNotFound.

104.3.14 Mtx.mtsErrCtxTMNotAvailable Constant

Pascal

```
mtsErrCtxTMNotAvailable = $8004E00F;
```

C++

```
mtsErrCtxTMNotAvailable = 0x8004E00F;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxTMNotAvailable.

104.3.15 Mtx.mtsErrCtxWrongThread Constant

Pascal

```
mtsErrCtxWrongThread = $8004E00E;
```

C++

```
mtsErrCtxWrongThread = 0x8004E00E;
```

File

Mtx

Description

This is constant Mtx.mtsErrCtxWrongThread.

104.3.16 Mtx.Process Constant

Pascal

```
Process = $00000001;
```

C++

```
Process = 0x00000001;
```

File

Mtx

Description

This is constant Mtx.Process.

104.3.17 Mtx.Standard Constant

Pascal

```
Standard = $00000000;
```

C++

```
Standard = 0x00000000;
```

File

Mtx

Description

This is constant Mtx.Standard.

105 Mxarrays Namespace

This is namespace Mxarrays.

Topics

Name	Description
Classes (see page 17305)	The following table lists classes in this documentation.
Functions (see page 17386)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 17387)	The following table lists structs, records, enums in this documentation.
Types (see page 17390)	The following table lists types in this documentation.
Constants (see page 17394)	The following table lists constants in this documentation.

Classes

	Name	Description
	EArrayError (see page 17305)	EArrayError is the exception class for failures that occur when manipulating a descendant of TBaseArray (see page 17307).
	EDynArrayRangeError (see page 17306)	This is class Mxarrays.EDynArrayRangeError.
	ELowCapacityError (see page 17306)	ELowCapacityError is the exception class for when an attempt is made to allocate more memory than is available for the decision cube.
	EUnsupportedTypeError (see page 17307)	EUnsupportedTypeError is the exception class for failures that occur when adding dimensions to the cache of a decision cube.
	TBaseArray (see page 17307)	TBaseArray is the base class for all arrays used by the decision cube components.
	TCurrencyArray (see page 17322)	This is class Mxarrays.TCurrencyArray.
	TCustomArray (see page 17328)	This is class Mxarrays.TCustomArray.
	TDoubleArray (see page 17337)	This is class Mxarrays.TDoubleArray.
	TIndexArray (see page 17342)	This is class Mxarrays.TIndexArray.
	TIntArray (see page 17346)	This is class Mxarrays.TIntArray.
	TPointerArray (see page 17351)	This is class Mxarrays.TPointerArray.
	TSingleArray (see page 17356)	This is class Mxarrays.TSingleArray.
	TSmallIntArray (see page 17361)	TSmallIntArray is a dynamically resizable array of 16-bit integers.
	TStringArray (see page 17367)	This is class Mxarrays.TStringArray.
	TThreadCustomArray (see page 17374)	This is class Mxarrays.TThreadCustomArray.
	TTwoDimArray (see page 17377)	This is class Mxarrays.TTwoDimArray.
	TWordArray (see page 17380)	This is class Mxarrays.TWordArray.

Constants

Name	Description
vMaxCol (see page 17394)	This is constant Mxarrays.vMaxCol.
vMaxRow (see page 17394)	This is constant Mxarrays.vMaxRow.

Functions

	Name	Description
	GetMemoryCapacity (see page 17386)	This is function Mxarrays.GetMemoryCapacity.
	SetMemoryCapacity (see page 17386)	This is function Mxarrays.SetMemoryCapacity.

Structs, Records, Enums

	Name	Description
	TArrayFlags (see page 17387)	This is record Mxarrays.TArrayFlags.
	TDuplicates (see page 17387)	This is record Mxarrays.TDuplicates.
	TSortOrder (see page 17388)	This is record Mxarrays.TSortOrder.
	TStringItem (see page 17388)	This is record Mxarrays.TStringItem.

Types

Name	Description
PIndexElements (see page 17390)	This is type Mxarrays.PIndexElements.
PMatrixElements (see page 17390)	This is type Mxarrays.PMatrixElements.
TArrayFlagSet (see page 17391)	This is type Mxarrays.TArrayFlagSet.
TCompareProc (see page 17391)	This is type Mxarrays.TCompareProc.
TDynArrayNDX (see page 17391)	This is type Mxarrays.TDynArrayNDX.
TIndexElements (see page 17392)	This is type Mxarrays.TIndexElements.
TIndexNDX (see page 17392)	This is type Mxarrays.TIndexNDX.
TMatrixElements (see page 17392)	This is type Mxarrays.TMatrixElements.
TMatrixNDX (see page 17393)	This is type Mxarrays.TMatrixNDX.

105.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EArrayError (see page 17305)	EArrayError is the exception class for failures that occur when manipulating a descendant of TBaseArray (see page 17307).
	EDynArrayRangeError (see page 17306)	This is class Mxarrays.EDynArrayRangeError.
	ELowCapacityError (see page 17306)	ELowCapacityError is the exception class for when an attempt is made to allocate more memory than is available for the decision cube.
	EUnsupportedTypeError (see page 17307)	EUnsupportedTypeError is the exception class for failures that occur when adding dimensions to the cache of a decision cube.
	TBaseArray (see page 17307)	TBaseArray is the base class for all arrays used by the decision cube components.
	TCurrencyArray (see page 17322)	This is class Mxarrays.TCurrencyArray.
	TCustomArray (see page 17328)	This is class Mxarrays.TCustomArray.
	TDoubleArray (see page 17337)	This is class Mxarrays.TDoubleArray.
	TIndexArray (see page 17342)	This is class Mxarrays.TIndexArray.
	TIntArray (see page 17346)	This is class Mxarrays.TIntArray.
	TPointerArray (see page 17351)	This is class Mxarrays.TPointerArray.
	TSingleArray (see page 17356)	This is class Mxarrays.TSingleArray.
	TSmallIntArray (see page 17361)	TSmallIntArray is a dynamically resizable array of 16-bit integers.
	TStringArray (see page 17367)	This is class Mxarrays.TStringArray.
	TThreadCustomArray (see page 17374)	This is class Mxarrays.TThreadCustomArray.
	TTwoDimArray (see page 17377)	This is class Mxarrays.TTwoDimArray.
	TWordArray (see page 17380)	This is class Mxarrays.TWordArray.

105.1.1 EArrayError Class

EArrayError is the exception class for failures that occur when manipulating a descendant of TBaseArray (see page 17307).

Class Hierarchy

Pascal

```
EArrayError = class(Exception);
```

C++

```
class EArrayError : public Exception;
```

File

Mxarrays

Description

EArrayError is raised when an application tries to

Use an invalid index when accessing the elements of an array.

Add too many elements into a fixed-capacity array.

Insert an element into a sorted array.

See Also

TBaseArray (see page 17307)

105.1.2 EDynArrayRangeError Class

Class Hierarchy

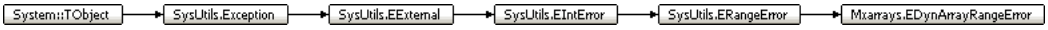

Pascal

```
EDynArrayRangeError = class(ERangeError);
```

C++

```
class EDynArrayRangeError : public ERangeError;
```

File

Mxarrays

Description

This is class Mxarrays.EDynArrayRangeError.

105.1.3 ELowCapacityError Class

ELowCapacityError is the exception class for when an attempt is made to allocate more memory than is available for the decision cube.

Class Hierarchy

Pascal

```
ELowCapacityError = class(Exception);
```

C++

```
class ELowCapacityError : public Exception;
```

File

Mxarrays

Description

ELowCapacityError is raised when the cache of a decision cube is too big. The memory available to the decision cube can be increased by changing the Capacity property of the decision cube or the memory demands of the decision cube can be reduced by deactivating some of the cube dimensions.

See Also

TCubeDim (see page 17754)

TDecisionCube (see page 17442)

105.1.4 EUnsupportedTypeError Class

EUnsupportedTypeError is the exception class for failures that occur when adding dimensions to the cache of a decision cube.

Class Hierarchy

Pascal

```
EUnsupportedTypeError = class(Exception);
```

C++

```
class EUnsupportedTypeError : public Exception;
```

File

Mxarrays

Description

EUnsupportedTypeError is raised when an invalid field type is chosen as an active dimension in a decision cube. For example, making a dimension that corresponds to a BLOB field active raises an EUnsupportedTypeError.

See Also

TBlobField (see page 6801)

TDecisionCube (see page 17442)

105.1.5 TBaseArray Class

TBaseArray is the base class for all arrays used by the decision cube components.

Class Hierarchy

Pascal

```
TBaseArray = class(TPersistent);
```

C++

```
class TBaseArray : public TPersistent;
```

File

Mxarrays

Description

Use TBaseArray as a base class when defining a specialized array for a specific data type. Like the TList or TStringList object, TBaseArray represents an ordered list of items that can grow dynamically. However, the elements of TBaseArray are arbitrary types whose size is specified in the constructor, while TStringList is limited to strings and TList is limited to pointers.

Descendants of TBaseArray represent arrays of specific data types such as short integers.

Note: TBaseArray is only available in the Enterprise edition.

See Also

TList (see page 2868)

TSmallIntArray (see page 17361)

TStringList (see page 3006)

105.1.5.1 TBaseArray Members

The following tables list the members exposed by TBaseArray.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BlockCopy (see page 17311)	This is BlockCopy, a member of class TBaseArray.
	ClearFlag (see page 17311)	This is ClearFlag, a member of class TBaseArray.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetAutoSize (see page 17313)	This is GetAutoSize, a member of class TBaseArray.
	GetOwner (see page 2932)	Returns the owner of an object.
	Grow (see page 17313)	This is Grow, a member of class TBaseArray.
	HasFlag (see page 17314)	This is HasFlag, a member of class TBaseArray.
	InternalHandleException (see page 17315)	This is InternalHandleException, a member of class TBaseArray.
	InvalidateItems (see page 17315)	This is InvalidateItems, a member of class TBaseArray.
	RemoveRange (see page 17316)	This is RemoveRange, a member of class TBaseArray.
	SetAutoSize (see page 17316)	This is SetAutoSize, a member of class TBaseArray.
	SetCapacity (see page 17317)	This is SetCapacity, a member of class TBaseArray.
	SetFlag (see page 17317)	This is SetFlag, a member of class TBaseArray.
	ValidateBounds (see page 17317)	This is ValidateBounds, a member of class TBaseArray.
	ValidIndex (see page 17318)	This is ValidIndex, a member of class TBaseArray.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 17311)	Clear removes all items from the array.
	Create (see page 17309)	Creates an instance of TBaseArray (see page 17307) for elements of a specific size.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 17312)	Removes an element from the array.
	Destroy (see page 17310)	Destroys an instance of TBaseArray (see page 17307).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Exchange (see page 17312)	Provides the interface for a method that swaps the values of two elements in the array.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindItem (see page 17312)	Locates a value in a sorted array and indicates whether an element with that value already exists.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItem (see page 17313)	Returns the element in the array at the indicated position.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	IndexOf (see page 17314)	Returns the position of the first element with a specified value.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Insert (see page 17314)	Inserts an element into the array and assigns it a value.
	InsertAt (see page 17315)	Adds an element to the array at the position specified by Index.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PutItem (see page 17315)	Replaces the value of an element in the array.
	RemoveItem (see page 17316)	Removes an element from the array.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Sort (see page 17317)	Provides the interface for a method that sorts the elements in the array.

Public Properties

	Name	Description
	AutoSize (see page 17318)	Indicates whether the array grows dynamically as items are added.
	Capacity (see page 17319)	Specifies the maximum number of items that can be added to the array.
	CompareProc (see page 17319)	Implements a sort order for the elements of the array.
	Count (see page 17320)	Specifies the number of items in the array.
	Duplicates (see page 17320)	Specifies whether duplicate items can be added to sorted arrays.
R	ItemSize (see page 17321)	Indicates the number of bytes used to store each item in the array.
	Limit (see page 17321)	Specifies the size of the array.
R	List (see page 17321)	Points to the array of values maintained by TBaseArray (see page 17307).
	SortOrder (see page 17322)	Determines whether the array is sorted.

105.1.5.2 TBaseArray.Create Constructor

Creates an instance of TBaseArray (see page 17307) for elements of a specific size.

Pascal

```
constructor Create(itemcount: Integer; iSize: Integer); virtual;
```

C++

```
virtual __fastcall TBaseArray(int itemcount, int iSize);
```

Description

Call Create to create an array containing elements with the size specified by the iSize parameter.Create sets the Capacity (see page 17319) property to Itemcount and the ItemSize (see page 17321) property to iSize. It also initializes the AutoSize (see page 17318) property to true and the Count (see page 17320) property to 0.

See Also

AutoSize (see page 17318)

Capacity (see page 17319)

Count (see page 17320)

ItemSize (see page 17321)

Create (see page 22287)

105.1.5.3 TBaseArray.Destroy Destructor

Destroys an instance of TBaseArray (see page 17307).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBaseArray();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the array reference is not nil before calling Destroy.

Destroy frees the memory allocated to hold the list of elements before calling the inherited destructor.

See Also

Free (see page 22300)

105.1.5.4 TBaseArray Methods

The methods of the TBaseArray class are listed here.

Protected Methods

	Name	Description
	BlockCopy (see page 17311)	This is BlockCopy, a member of class TBaseArray.
	ClearFlag (see page 17311)	This is ClearFlag, a member of class TBaseArray.
	GetAutoSize (see page 17313)	This is GetAutoSize, a member of class TBaseArray.
V	Grow (see page 17313)	This is Grow, a member of class TBaseArray.
	HasFlag (see page 17314)	This is HasFlag, a member of class TBaseArray.
	InternalHandleException (see page 17315)	This is InternalHandleException, a member of class TBaseArray.
V	InvalidateItems (see page 17315)	This is InvalidateItems, a member of class TBaseArray.
	RemoveRange (see page 17316)	This is RemoveRange, a member of class TBaseArray.
	SetAutoSize (see page 17316)	This is SetAutoSize, a member of class TBaseArray.
V	SetCapacity (see page 17317)	This is SetCapacity, a member of class TBaseArray.
	SetFlag (see page 17317)	This is SetFlag, a member of class TBaseArray.
	ValidateBounds (see page 17317)	This is ValidateBounds, a member of class TBaseArray.
	ValidIndex (see page 17318)	This is ValidIndex, a member of class TBaseArray.

Public Methods

	Name	Description
	Clear (see page 17311)	Clear removes all items from the array.

	Delete (see page 17312)	Removes an element from the array.
	Exchange (see page 17312)	Provides the interface for a method that swaps the values of two elements in the array.
	FindItem (see page 17312)	Locates a value in a sorted array and indicates whether an element with that value already exists.
	GetItem (see page 17313)	Returns the element in the array at the indicated position.
	IndexOf (see page 17314)	Returns the position of the first element with a specified value.
	Insert (see page 17314)	Inserts an element into the array and assigns it a value.
	InsertAt (see page 17315)	Adds an element to the array at the position specified by Index.
	PutItem (see page 17315)	Replaces the value of an element in the array.
	RemoveItem (see page 17316)	Removes an element from the array.
	Sort (see page 17317)	Provides the interface for a method that sorts the elements in the array.

105.1.5.4.1 TBaseArray.BlockCopy Method

Pascal

```
procedure BlockCopy(Source: TBaseArray; fromIndex: Integer; toIndex: Integer; numitems:
Integer);
```

C++

```
__fastcall BlockCopy(TBaseArray Source, int fromIndex, int toIndex, int numitems);
```

Description

This is BlockCopy, a member of class TBaseArray.

105.1.5.4.2 TBaseArray.Clear Method

Clear removes all items from the array.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the array and free all associated memory. Clear does the same thing as setting both the Count (see page 17320) and Capacity (see page 17319) properties to 0.

See Also

Capacity (see page 17319)

Count (see page 17320)

105.1.5.4.3 TBaseArray.ClearFlag Method

Pascal

```
procedure ClearFlag(aFlag: TArrayFlags);
```

C++

```
__fastcall ClearFlag(TArrayFlags aFlag);
```

Description

This is ClearFlag, a member of class TBaseArray.

105.1.5.4.4 TBaseArray.Delete Method

Removes an element from the array.

Pascal

```
procedure Delete(Index: Integer); virtual;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Call Delete to remove the element indicated by the Index parameter, where 0 is the first element, 1 is the second element, and so on. All subsequent elements in the array are moved up to fill the gap left when the element is removed.

See Also

Clear (see page 17311)

Insert (see page 17314)

105.1.5.4.5 TBaseArray.Exchange Method

Provides the interface for a method that swaps the values of two elements in the array.

Pascal

```
procedure Exchange(Index1: Integer; Index2: Integer); virtual;
```

C++

```
virtual __fastcall Exchange(int Index1, int Index2);
```

Description

The Exchange method for TBaseArray (see page 17307) does nothing other than provide an interface for a method that interchanges the values of two elements.

See Also

IndexOf (see page 17314)

GetItem (see page 17313)

105.1.5.4.6 TBaseArray.FindItem Method

Locates a value in a sorted array and indicates whether an element with that value already exists.

Pascal

```
function FindItem(var Index: Integer; var Value): Boolean;
```

C++

```
__fastcall Boolean FindItem(int Index,  Value);
```

Description

Use FindItem to obtain the index in a sorted array where an element with the value Value should be added. If an element with the value Value already exists in the array, FindItem returns true. If none of the elements in the array have the value Value, FindItem returns false.

The index where Value should appear is returned in the Index parameter. The value of Index is zero-based, where the first

element has the index 0, the second element has the index 1, and so on.

Note: FindItem assumes the SortOrder (see page 17322) property is TS_ASCENDING. For unsorted arrays, use the IndexOf (see page 17314) method instead. For descending arrays, invert the CompareProc (see page 17319) property before calling FindItem.

See Also

Duplicates (see page 17320)

IndexOf (see page 17314)

SortOrder (see page 17322)

105.1.5.4.7 TBaseArray.GetAutoSize Method

Pascal

```
function GetAutoSize: Boolean;
```

C++

```
__fastcall Boolean GetAutoSize();
```

Description

This is GetAutoSize, a member of class TBaseArray.

105.1.5.4.8 TBaseArray.GetItem Method

Returns the element in the array at the indicated position.

Pascal

```
procedure GetItem(index: Integer; var Value);
```

C++

```
__fastcall GetItem(int index,  Value);
```

Description

Call GetItem to obtain the value of an element in the array. The Index parameter indicates the element of the array, where the first element has the index 0, the second element has the index 1, and so on.

See Also

FindItem (see page 17312)

IndexOf (see page 17314)

105.1.5.4.9 TBaseArray.Grow Method

Pascal

```
procedure Grow; virtual;
```

C++

```
virtual __fastcall Grow();
```

Description

This is Grow, a member of class TBaseArray.

105.1.5.4.10 TBaseArray.HasFlag Method

Pascal

```
function HasFlag(aFlag: TArrayFlags): Boolean;
```

C++

```
__fastcall Boolean HasFlag(TArrayFlags aFlag);
```

Description

This is HasFlag, a member of class TBaseArray.

105.1.5.4.11 TBaseArray.IndexOf Method

Returns the position of the first element with a specified value.

Pascal

```
function IndexOf(var Item): Integer; virtual;
```

C++

```
virtual __fastcall int IndexOf( Item);
```

Description

Call IndexOf to obtain the position of the first occurrence of the value Value. IndexOf returns the 0-based index of the element. Thus, if Value matches the first element in the array, IndexOf returns 0, if Value matches the second element, IndexOf returns 1, and so on. If the value is not in the array, IndexOf returns -1.

Note: IndexOf does not work for arrays sorted in descending order.

See Also

FindItem (see page 17312)

GetItem (see page 17313)

105.1.5.4.12 TBaseArray.Insert Method

Inserts an element into the array and assigns it a value.

Pascal

```
procedure Insert(Index: Integer; var Value); virtual;
```

C++

```
virtual __fastcall Insert(int Index,  Value);
```

Description

Call InsertAt (see page 17315) to add an element to the array at the position specified by Index and assign it the value of Value. If Index is 0, the element is inserted at the beginning of the array. If Index is 1, the element is put in the second position of the array, and so on.

Note: Insert does not check the SortOrder (see page 17322) property. If the array is sorted, be sure to insert the element in the proper sort order position.

See Also

Delete (see page 17312)

PutItem (see page 17315)

SortOrder (see page 17322)

105.1.5.4.13 TBaseArray.InsertAt Method

Adds an element to the array at the position specified by Index.

Pascal

```
procedure InsertAt(Index: Integer; var Value);
```

C++

```
__fastcall InsertAt(int Index,  Value);
```

Description

InsertAt does the same thing as the Insert (see page 17314) method. For most purposes, use the Insert (see page 17314) method.

Descendants of TBaseArray (see page 17307) can override the Insert (see page 17314) method to raise an exception when an attempt is made to insert an element in a sorted array. InsertAt always succeeds as long as the value of Index is appropriate. InsertAt is intended for internal use when the value of Index is known to be appropriate.

See Also

Insert (see page 17314)

105.1.5.4.14 TBaseArray.InternalHandleException Method

Pascal

```
procedure InternalHandleException;
```

C++

```
__fastcall InternalHandleException();
```

Description

This is InternalHandleException, a member of class TBaseArray.

105.1.5.4.15 TBaseArray.InvalidateItems Method

Pascal

```
procedure InvalidateItems(atIndex: Integer; numItems: Integer); virtual;
```

C++

```
virtual __fastcall InvalidateItems(int atIndex, int numItems);
```

Description

This is InvalidateItems, a member of class TBaseArray.

105.1.5.4.16 TBaseArray.PutItem Method

Replaces the value of an element in the array.

Pascal

```
procedure PutItem(index: Integer; var Value);
```

C++

```
__fastcall PutItem(int index,  Value);
```

Description

Call PutItem to change the value of an element to the value Value. Unlike the Insert (see page 17314) method, PutItem does not add a new item to the array, but replaces the value of an existing element. The Index parameter indicates which element to change, where 0 specifies the first element, 1 specifies the second element, and so on.

Note: Calling PutItem with Index set to the value of the Count (see page 17320) property will add a new element to the end of the array. This is the only case where PutItem adds a new element instead of replacing an existing one.

See Also

Count (see page 17320)

GetItem (see page 17313)

Insert (see page 17314)

105.1.5.4.17 TBaseArray.RemoveItem Method

Removes an element from the array.

Pascal

```
procedure RemoveItem(Index: Integer);
```

C++

```
__fastcall RemoveItem(int Index);
```

Description

RemoveItem does the same thing as the Delete (see page 17312) method. Use the Delete (see page 17312) method instead.

See Also

Delete (see page 17312)

105.1.5.4.18 TBaseArray.RemoveRange Method

Pascal

```
procedure RemoveRange(atIndex: Integer; numItems: Integer);
```

C++

```
__fastcall RemoveRange(int atIndex, int numItems);
```

Description

This is RemoveRange, a member of class TBaseArray.

105.1.5.4.19 TBaseArray.SetAutoSize Method

Pascal

```
procedure SetAutoSize(aSize: Boolean);
```

C++

```
__fastcall SetAutoSize(Boolean aSize);
```

Description

This is SetAutoSize, a member of class TBaseArray.

105.1.5.4.20 TBaseArray.SetCapacity Method

Pascal

```
procedure SetCapacity(NewCapacity: Integer); virtual;
```

C++

```
virtual __fastcall SetCapacity(int NewCapacity);
```

Description

This is SetCapacity, a member of class TBaseArray.

105.1.5.4.21 TBaseArray.SetFlag Method

Pascal

```
procedure SetFlag(aFlag: TArrayFlags);
```

C++

```
__fastcall SetFlag(TArrayFlags aFlag);
```

Description

This is SetFlag, a member of class TBaseArray.

105.1.5.4.22 TBaseArray.Sort Method

Provides the interface for a method that sorts the elements in the array.

Pascal

```
procedure Sort(Compare: TCompareProc); virtual;
```

C++

```
virtual __fastcall Sort(TCompareProc Compare);
```

Description

The Sort method for TBaseArray (see page 17307) does nothing. Descendants of TBaseArray (see page 17307) override this method to sort the elements of the array using the ordering function provided in the Compare parameter.

See Also

CompareProc (see page 17319)

SortOrder (see page 17322)

105.1.5.4.23 TBaseArray.ValidateBounds Method

Pascal

```
function ValidateBounds(atIndex: Integer; var numItems: Integer): Boolean;
```

C++

```
__fastcall Boolean ValidateBounds(int atIndex, int numItems);
```

Description

This is ValidateBounds, a member of class TBaseArray.

105.1.5.4.24 TBaseArray.ValidIndex Method

Pascal

```
function ValidIndex(Index: Integer): Boolean;
```

C++

```
__fastcall Boolean ValidIndex(int Index);
```

Description

This is ValidIndex, a member of class TBaseArray.

105.1.5.5 TBaseArray Properties

The properties of the TBaseArray class are listed here.

Public Properties

	Name	Description
	AutoSize (see page 17318)	Indicates whether the array grows dynamically as items are added.
	Capacity (see page 17319)	Specifies the maximum number of items that can be added to the array.
	CompareProc (see page 17319)	Implements a sort order for the elements of the array.
	Count (see page 17320)	Specifies the number of items in the array.
	Duplicates (see page 17320)	Specifies whether duplicate items can be added to sorted arrays.
R	ItemSize (see page 17321)	Indicates the number of bytes used to store each item in the array.
	Limit (see page 17321)	Specifies the size of the array.
R	List (see page 17321)	Points to the array of values maintained by TBaseArray (see page 17307).
	SortOrder (see page 17322)	Determines whether the array is sorted.

105.1.5.5.1 TBaseArray.AutoSize Property

Indicates whether the array grows dynamically as items are added.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Set AutoSize to true to allow the array to hold an arbitrary number of values. When AutoSize is true, the Capacity (see page 17319) property increases automatically when new items are added beyond the current capacity of the array. When AutoSize is false, the Capacity (see page 17319) property must be explicitly set to the number of entries allowed in the array. If an attempt is made to add items beyond the number specified in the Capacity (see page 17319) property, an EArrayError (see page 17305) exception is raised if AutoSize is false.

See Also

Capacity (see page 17319)

Count (see page 17320)

EArrayError (see page 17305)

105.1.5.5.2 TBaseArray.Capacity Property

Specifies the maximum number of items that can be added to the array.

Pascal

```
property Capacity: Integer;
```

C++

```
__property int Capacity;
```

Description

Set Capacity to change the amount of memory allocated for the array. Trying to set Capacity to a value less than the Count (see page 17320) property will raise an EArrayError (see page 17305) exception.

If the AutoSize (see page 17318) property is true, the Capacity of the array increases automatically when necessary to accommodate new items as they are added.

Note: To determine the amount of memory used by the array, use Capacity * ItemSize (see page 17321).

See Also

AutoSize (see page 17318)

Count (see page 17320)

EArrayError (see page 17305)

ItemSize (see page 17321)

105.1.5.5.3 TBaseArray.CompareProc Property

Implements a sort order for the elements of the array.

Pascal

```
property CompareProc: TCompareProc;
```

C++

```
__property TCompareProc CompareProc;
```

Description

Set CompareProc to change the function that is used to sort the items of the array. If the SortOrder (see page 17322) property is not TS_NONE, CompareProc is used to locate the position in which to add new items to the array and to locate items using the FindItem (see page 17312) method.

CompareProc should compare the two items given as parameters and return a value less than 0 if item1 comes before item2, return a value greater than 0 if item1 comes after item2, and return 0 if item1 is the same as item2.

See Also

FindItem (see page 17312)

Sort (see page 17317)

SortOrder (see page 17322)

105.1.5.5.4 TBaseArray.Count Property

Specifies the number of items in the array.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read Count to determine the number of entries in the array. Set Count to increase or decrease the number of entries. Increasing Count always increases the Capacity (see page 17319) if necessary, even if the AutoSize (see page 17318) property is false. Decreasing the Count property truncates the array after Count items, but does not affect the Capacity (see page 17319) of the array.

When the value of Count is increased, new entries with the value of 0 are added to the end of the array. If the array is sorted, this will throw off the internal sort order, and cause the FindItem (see page 17312) method to fail. If the array is sorted, call the Sort (see page 17317) method to restore the sort order after filling the newly added members of the array.

See Also

AutoSize (see page 17318)

Capacity (see page 17319)

FindItem (see page 17312)

Sort (see page 17317)

SortOrder (see page 17322)

105.1.5.5.5 TBaseArray.Duplicates Property

Specifies whether duplicate items can be added to sorted arrays.

Pascal

```
property Duplicates: TDuplicates;
```

C++

```
__property TDuplicates Duplicates;
```

Description

Set Duplicates to specify what should happen when an attempt is made to add a duplicate item to a sorted array. The value of Duplicates should be one of the following.

Value	Meaning
dupIgnore	Ignore attempts to add duplicate items to the sorted array.
dupError	raise an EArrayError (see page 17305) exception when an attempt is made to add duplicate items to the sorted array.
dupAccept	Permit duplicate items in the sorted array.

Set Duplicates before adding any items to the list. Setting Duplicates to dupIgnore or dupError does nothing about duplicate strings that are already in the array.

Note: Duplicates does nothing if the array is not sorted.

See Also

SortOrder (see page 17322)

EArrayError (see page 17305)

105.1.5.5.6 TBaseArray.ItemSize Property

Indicates the number of bytes used to store each item in the array.

Pascal

```
property ItemSize: Integer;
```

C++

```
__property int ItemSize;
```

Description

Read ItemSize to determine the size of the elements of the array. The value of ItemSize is set by the constructor to the value of the iSize parameter. Descendants of TBaseArray (see page 17307) change this value to the size of the data type they represent.

See Also

Create (see page 17309)

105.1.5.5.7 TBaseArray.Limit Property

Specifies the size of the array.

Pascal

```
property Limit: Integer;
```

C++

```
__property int Limit;
```

Description

Use Limit to determine the size of the array. If the array is empty, Limit is the same as the Capacity (see page 17319) property. If the array is not empty, Limit is the same as the Count (see page 17320) property, even if the value of Capacity (see page 17319) is larger than Count (see page 17320). Setting Limit sets the Capacity (see page 17319) of the array.

See Also

Capacity (see page 17319)

Count (see page 17320)

105.1.5.5.8 TBaseArray.List Property

Points to the array of values maintained by TBaseArray (see page 17307).

Pascal

```
property List: Pointer;
```

C++

```
__property void * List;
```

Description

Use List to get direct access to the memory used to store the array items. Calculate the position of individual items in the list using the ItemSize (see page 17321) property.

To access specific items in the list, use the GetItem (see page 17313) method.

See Also

GetItem (see page 17313)

ItemSize (see page 17321)

105.1.5.5.9 TBaseArray.SortOrder Property

Determines whether the array is sorted.

Pascal

```
property SortOrder: TSortOrder;
```

C++

```
__property TSortOrder SortOrder;
```

Description

Set SortOrder to indicate whether the array should be sorted using the CompareProc (see page 17319) property. When an array is sorted, all new items are added in the correct sort order position so as to maintain the sort order. SortOrder can have one of the following values:

Value	Meaning
TS_NONE	The array is unsorted.
TS_ASCENDING	The elements of the array are sorted in ascending order.
TS_DESCENDING	The elements of the array are sorted in descending order.

Setting the SortOrder property does not sort the items already in the array. If the array is not empty when the SortOrder property is changed to TS_ASCENDING or TS_DESCENDING, be sure to explicitly call the Sort (see page 17317) method to sort the existing items.

Note: The Duplicates (see page 17320) property has no effect when SortOrder is TS_NONE.

See Also

CompareProc (see page 17319)

Duplicates (see page 17320)

Sort (see page 17317)

105.1.6 TCurrencyArray Class

Class Hierarchy

Pascal

```
TCurrencyArray = class(TBaseArray);
```

C++

```
class TCurrencyArray : public TBaseArray;
```

File

Mxarrays

Description

This is class Mxarrays.TCurrencyArray.

105.1.6.1 TCurrencyArray Members

The following tables list the members exposed by TCurrencyArray.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BlockCopy (see page 17311)	This is BlockCopy, a member of class TBaseArray.
	ClearFlag (see page 17311)	This is ClearFlag, a member of class TBaseArray.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetAutoSize (see page 17313)	This is GetAutoSize, a member of class TBaseArray.
	GetOwner (see page 2932)	Returns the owner of an object.
	Grow (see page 17313)	This is Grow, a member of class TBaseArray.
	HasFlag (see page 17314)	This is HasFlag, a member of class TBaseArray.
	InternalHandleException (see page 17315)	This is InternalHandleException, a member of class TBaseArray.
	InvalidateItems (see page 17315)	This is InvalidateItems, a member of class TBaseArray.
	RemoveRange (see page 17316)	This is RemoveRange, a member of class TBaseArray.
	SetAutoSize (see page 17316)	This is SetAutoSize, a member of class TBaseArray.
	SetCapacity (see page 17317)	This is SetCapacity, a member of class TBaseArray.
	SetFlag (see page 17317)	This is SetFlag, a member of class TBaseArray.
	ValidateBounds (see page 17317)	This is ValidateBounds, a member of class TBaseArray.
	ValidIndex (see page 17318)	This is ValidIndex, a member of class TBaseArray.

Public Methods

	Name	Description
	Add (see page 17325)	This is Add, a member of class TCurrencyArray.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 17325)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 17311)	Clear removes all items from the array.
	Create (see page 17324)	Creates an instance of TBaseArray for elements of a specific size.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 17312)	Removes an element from the array.

V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 17310)	Destroys an instance of TBaseArray (see page 17307).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Exchange (see page 17312)	Provides the interface for a method that swaps the values of two elements in the array.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Find (see page 17326)	This is Find, a member of class TCurrencyArray.
	FindItem (see page 17312)	Locates a value in a sorted array and indicates whether an element with that value already exists.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItem (see page 17326)	This is GetItem, a member of class TCurrencyArray.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	IndexOf (see page 17327)	Returns the position of the first element with a specified value.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Insert (see page 17314)	Inserts an element into the array and assigns it a value.
	InsertAt (see page 17315)	Adds an element to the array at the position specified by Index.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PutItem (see page 17327)	This is PutItem, a member of class TCurrencyArray.
	RemoveItem (see page 17316)	Removes an element from the array.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Sort (see page 17317)	Provides the interface for a method that sorts the elements in the array.

Public Properties

	Name	Description
	AutoSize (see page 17318)	Indicates whether the array grows dynamically as items are added.
	Capacity (see page 17319)	Specifies the maximum number of items that can be added to the array.
	CompareProc (see page 17319)	Implements a sort order for the elements of the array.
	Count (see page 17320)	Specifies the number of items in the array.
	Duplicates (see page 17320)	Specifies whether duplicate items can be added to sorted arrays.
	Items (see page 17327)	This is Items, a member of class TCurrencyArray.
R	ItemSize (see page 17321)	Indicates the number of bytes used to store each item in the array.
	Limit (see page 17321)	Specifies the size of the array.
R	List (see page 17321)	Points to the array of values maintained by TBaseArray (see page 17307).
	SortOrder (see page 17322)	Determines whether the array is sorted.

105.1.6.2 TCurrencyArray.Create Constructor

Creates an instance of TBaseArray for elements of a specific size.

Pascal

```
constructor Create(itemcount: Integer; dummy: Integer); override;
```

C++

```
virtual __fastcall TCurrencyArray(int itemcount, int dummy);
```

Description

Call Create to create an array containing elements with the size specified by the iSize parameter.Create sets the Capacity property to Itemcount and the ItemSize property to iSize. It also initializes the AutoSize property to true and the Count property to 0.

See Also

AutoSize (see page 17318)

Capacity (see page 17319)

Count (see page 17320)

ItemSize (see page 17321)

Create (see page 22287)

105.1.6.3 TCurrencyArray Methods

The methods of the TCurrencyArray class are listed here.

Public Methods

	Name	Description
	Add (see page 17325)	This is Add, a member of class TCurrencyArray.
V	Assign (see page 17325)	Copies the contents of another, similar object.
	Find (see page 17326)	This is Find, a member of class TCurrencyArray.
	GetItem (see page 17326)	This is GetItem, a member of class TCurrencyArray.
V	IndexOf (see page 17327)	Returns the position of the first element with a specified value.
	PutItem (see page 17327)	This is PutItem, a member of class TCurrencyArray.

105.1.6.3.1 TCurrencyArray.Add Method

Pascal

```
function Add(Value: Currency): Integer;
```

C++

```
__fastcall int Add(Currency Value);
```

Description

This is Add, a member of class TCurrencyArray.

105.1.6.3.2 TCurrencyArray.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo (see page 2931)

EConvertError (see page 22724)

105.1.6.3.3 TCurrencyArray.Find Method

Pascal

```
function Find(var Index: Integer; Value: Currency): Boolean;
```

C++

```
__fastcall Boolean Find(int Index, Currency Value);
```

Description

This is Find, a member of class TCurrencyArray.

105.1.6.3.4 TCurrencyArray.GetItem Method

Pascal

```
function GetItem(index: Integer): Currency;
```

C++

```
__fastcall Currency GetItem(int index);
```

Description

This is GetItem, a member of class TCurrencyArray.

105.1.6.3.5 TCurrencyArray.IndexOf Method

Returns the position of the first element with a specified value.

Pascal

```
function IndexOf(var Item): Integer; override;
```

C++

```
virtual __fastcall int IndexOf( Item);
```

Description

Call IndexOf to obtain the position of the first occurrence of the value Value. IndexOf returns the 0-based index of the element. Thus, if Value matches the first element in the array, IndexOf returns 0, if Value matches the second element, IndexOf returns 1, and so on. If the value is not in the array, IndexOf returns -1.

Note: IndexOf does not work for arrays sorted in descending order.

See Also

FindItem (see page 17312)

GetItem (see page 17313)

105.1.6.3.6 TCurrencyArray.PutItem Method

Pascal

```
procedure PutItem(index: Integer; value: Currency);
```

C++

```
__fastcall PutItem(int index, Currency value);
```

Description

This is PutItem, a member of class TCurrencyArray.

105.1.6.4 TCurrencyArray Properties

The properties of the TCurrencyArray class are listed here.

Public Properties

	Name	Description
	Items (see page 17327)	This is Items, a member of class TCurrencyArray.

105.1.6.4.1 TCurrencyArray.Items Property

Pascal

```
property Items [Index: Integer]: Currency;
```

C++

```
__property Currency Items[int Index];
```

Description

This is Items, a member of class TCurrencyArray.

105.1.7 TCustomArray Class

Class Hierarchy

Mxarrays.TCustomArray

Pascal

```
TCustomArray = class;
```

C++

```
class TCustomArray;
```

File

Mxarrays

Description

This is class Mxarrays.TCustomArray.

105.1.7.1 TCustomArray Members

The following tables list the members exposed by TCustomArray.

Protected Methods

	Name	Description
	GetCompProc (see page 17330)	This is GetCompProc, a member of class TCustomArray.
	GetDups (see page 17331)	This is GetDups, a member of class TCustomArray.
	GetItem (see page 17332)	This is GetItem, a member of class TCustomArray.
	GetMemberCount (see page 17332)	This is GetMemberCount, a member of class TCustomArray.
	GetSort (see page 17332)	This is GetSort, a member of class TCustomArray.
	SetCompProc (see page 17333)	This is SetCompProc, a member of class TCustomArray.
	SetDups (see page 17333)	This is SetDups, a member of class TCustomArray.
	SetItem (see page 17333)	This is SetItem, a member of class TCustomArray.
	SetSort (see page 17334)	This is SetSort, a member of class TCustomArray.

Public Methods

	Name	Description
	Add (see page 17330)	This is Add, a member of class TCustomArray.
	Assign (see page 17330)	This is Assign, a member of class TCustomArray.
	ConvertVar (see page 17330)	This is ConvertVar, a member of class TCustomArray.
	Create (see page 17329)	This is Create, a member of class TCustomArray.
V	Destroy (see page 17329)	This is Destroy, a member of class TCustomArray.
	GetCurrency (see page 17331)	This is GetCurrency, a member of class TCustomArray.
	GetDouble (see page 17331)	This is GetDouble, a member of class TCustomArray.
	GetInteger (see page 17331)	This is GetInteger, a member of class TCustomArray.
	IndexOf (see page 17332)	This is IndexOf, a member of class TCustomArray.
	IsBlank (see page 17332)	This is IsBlank, a member of class TCustomArray.
V	MemoryUsage (see page 17333)	This is MemoryUsage, a member of class TCustomArray.
	SetSize (see page 17334)	This is SetSize, a member of class TCustomArray.

Public Properties

	Name	Description
	BlankBoolVal (see page 17334)	This is BlankBoolVal, a member of class TCustomArray.
R	BlankCount (see page 17335)	This is BlankCount, a member of class TCustomArray.
	BlankDateVal (see page 17335)	This is BlankDateVal, a member of class TCustomArray.
	BlankStringVal (see page 17335)	This is BlankStringVal, a member of class TCustomArray.
	CompareProc (see page 17335)	This is CompareProc, a member of class TCustomArray.
R	DataType (see page 17335)	This is DataType, a member of class TCustomArray.
	Duplicates (see page 17336)	This is Duplicates, a member of class TCustomArray.
	Items (see page 17336)	This is Items, a member of class TCustomArray.
R	List (see page 17336)	This is List, a member of class TCustomArray.
R	MemberCount (see page 17336)	This is MemberCount, a member of class TCustomArray.
	Sorted (see page 17337)	This is Sorted, a member of class TCustomArray.

105.1.7.2 TCustomArray.Create Constructor

Pascal

```
constructor Create(Items: Integer; VarType: Integer);
```

C++

```
__fastcall TCustomArray(int Items, int VarType);
```

Description

This is Create, a member of class TCustomArray.

105.1.7.3 TCustomArray.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomArray();
```

Description

This is Destroy, a member of class TCustomArray.

105.1.7.4 TCustomArray Methods

The methods of the TCustomArray class are listed here.

Protected Methods

	Name	Description
	GetCompProc (see page 17330)	This is GetCompProc, a member of class TCustomArray.
	GetDups (see page 17331)	This is GetDups, a member of class TCustomArray.
	GetItem (see page 17332)	This is GetItem, a member of class TCustomArray.
	GetMemberCount (see page 17332)	This is GetMemberCount, a member of class TCustomArray.
	GetSort (see page 17332)	This is GetSort, a member of class TCustomArray.
	SetCompProc (see page 17333)	This is SetCompProc, a member of class TCustomArray.
	SetDups (see page 17333)	This is SetDups, a member of class TCustomArray.
	SetItem (see page 17333)	This is SetItem, a member of class TCustomArray.

	SetSort (see page 17334)	This is SetSort, a member of class TCustomArray.

Public Methods

	Name	Description
	Add (see page 17330)	This is Add, a member of class TCustomArray.
	Assign (see page 17330)	This is Assign, a member of class TCustomArray.
	ConvertVar (see page 17330)	This is ConvertVar, a member of class TCustomArray.
	GetCurrency (see page 17331)	This is GetCurrency, a member of class TCustomArray.
	GetDouble (see page 17331)	This is GetDouble, a member of class TCustomArray.
	GetInteger (see page 17331)	This is GetInteger, a member of class TCustomArray.
	IndexOf (see page 17332)	This is IndexOf, a member of class TCustomArray.
	IsBlank (see page 17332)	This is IsBlank, a member of class TCustomArray.
	MemoryUsage (see page 17333)	This is MemoryUsage, a member of class TCustomArray.
	SetSize (see page 17334)	This is SetSize, a member of class TCustomArray.

105.1.7.4.1 TCustomArray.Add Method

Pascal

```
function Add(Value: Variant): Integer;
```

C++

```
__fastcall int Add(Variant Value);
```

Description

This is Add, a member of class TCustomArray.

105.1.7.4.2 TCustomArray.Assign Method

Pascal

```
procedure Assign(Value: TCustomArray; bSorted: Boolean; bUnique: Boolean);
```

C++

```
__fastcall Assign(TCustomArray Value, Boolean bSorted, Boolean bUnique);
```

Description

This is Assign, a member of class TCustomArray.

105.1.7.4.3 TCustomArray.ConvertVar Method

Pascal

```
function ConvertVar(Value: Variant): Variant;
```

C++

```
__fastcall Variant ConvertVar(Variant Value);
```

Description

This is ConvertVar, a member of class TCustomArray.

105.1.7.4.4 TCustomArray.GetCompProc Method

Pascal

```
function GetCompProc: TCompareProc;
```

C++

```
__fastcall TCompareProc GetCompProc();
```

Description

This is GetCompProc, a member of class TCustomArray.

105.1.7.4.5 TCustomArray.GetCurrency Method

Pascal

```
function GetCurrency(Index: Integer): Currency;
```

C++

```
__fastcall Currency GetCurrency(int Index);
```

Description

This is GetCurrency, a member of class TCustomArray.

105.1.7.4.6 TCustomArray.GetDouble Method

Pascal

```
function GetDouble(Index: Integer): Double;
```

C++

```
__fastcall Double GetDouble(int Index);
```

Description

This is GetDouble, a member of class TCustomArray.

105.1.7.4.7 TCustomArray.GetDups Method

Pascal

```
function GetDups: TDuplicates;
```

C++

```
__fastcall TDuplicates GetDups();
```

Description

This is GetDups, a member of class TCustomArray.

105.1.7.4.8 TCustomArray.GetInteger Method

Pascal

```
function GetInteger(Index: Integer): Integer;
```

C++

```
__fastcall int GetInteger(int Index);
```

Description

This is GetInteger, a member of class TCustomArray.

105.1.7.4.9 TCustomArray.GetItem Method

Pascal

```
function GetItem(Index: Integer): Variant;
```

C++

```
__fastcall Variant GetItem(int Index);
```

Description

This is GetItem, a member of class TCustomArray.

105.1.7.4.10 TCustomArray.GetMemberCount Method

Pascal

```
function GetMemberCount: Integer;
```

C++

```
__fastcall int GetMemberCount();
```

Description

This is GetMemberCount, a member of class TCustomArray.

105.1.7.4.11 TCustomArray.GetSort Method

Pascal

```
function GetSort: Boolean;
```

C++

```
__fastcall Boolean GetSort();
```

Description

This is GetSort, a member of class TCustomArray.

105.1.7.4.12 TCustomArray.IndexOf Method

Pascal

```
function IndexOf(Value: Variant): Integer;
```

C++

```
__fastcall int IndexOf(Variant Value);
```

Description

This is IndexOf, a member of class TCustomArray.

105.1.7.4.13 TCustomArray.IsBlank Method

Pascal

```
function IsBlank(Index: Integer): Boolean;
```

C++

```
__fastcall Boolean IsBlank(int Index);
```

Description

This is IsBlank, a member of class TCustomArray.

105.1.7.4.14 TCustomArray.MemoryUsage Method

Pascal

```
function MemoryUsage: Integer; virtual;
```

C++

```
virtual __fastcall int MemoryUsage();
```

Description

This is MemoryUsage, a member of class TCustomArray.

105.1.7.4.15 TCustomArray.SetCompProc Method

Pascal

```
procedure SetCompProc(Proc: TCompareProc);
```

C++

```
__fastcall SetCompProc(TCompareProc Proc);
```

Description

This is SetCompProc, a member of class TCustomArray.

105.1.7.4.16 TCustomArray.SetDups Method

Pascal

```
procedure SetDups(Value: TDuplicates);
```

C++

```
__fastcall SetDups(TDuplicates Value);
```

Description

This is SetDups, a member of class TCustomArray.

105.1.7.4.17 TCustomArray.SetItem Method

Pascal

```
procedure SetItem(Index: Integer; Value: Variant);
```

C++

```
__fastcall SetItem(int Index, Variant Value);
```

Description

This is SetItem, a member of class TCustomArray.

105.1.7.4.18 TCustomArray.SetSize Method

Pascal

```
procedure SetSize(size: Integer);
```

C++

```
__fastcall SetSize(int size);
```

Description

This is SetSize, a member of class TCustomArray.

105.1.7.4.19 TCustomArray.SetSort Method

Pascal

```
procedure SetSort(Value: Boolean);
```

C++

```
__fastcall SetSort(Boolean Value);
```

Description

This is SetSort, a member of class TCustomArray.

105.1.7.5 TCustomArray Properties

The properties of the TCustomArray class are listed here.

Public Properties

	Name	Description
	BlankBoolVal (see page 17334)	This is BlankBoolVal, a member of class TCustomArray.
R	BlankCount (see page 17335)	This is BlankCount, a member of class TCustomArray.
	BlankDateVal (see page 17335)	This is BlankDateVal, a member of class TCustomArray.
	BlankStringVal (see page 17335)	This is BlankStringVal, a member of class TCustomArray.
	CompareProc (see page 17335)	This is CompareProc, a member of class TCustomArray.
R	DataType (see page 17335)	This is DataType, a member of class TCustomArray.
	Duplicates (see page 17336)	This is Duplicates, a member of class TCustomArray.
	Items (see page 17336)	This is Items, a member of class TCustomArray.
R	List (see page 17336)	This is List, a member of class TCustomArray.
R	MemberCount (see page 17336)	This is MemberCount, a member of class TCustomArray.
	Sorted (see page 17337)	This is Sorted, a member of class TCustomArray.

105.1.7.5.1 TCustomArray.BlankBoolVal Property

Pascal

```
property BlankBoolVal: Word;
```

C++

```
__property Word BlankBoolVal;
```

Description

This is BlankBoolVal, a member of class TCustomArray.

105.1.7.5.2 TCustomArray.BlankCount Property

Pascal

```
property BlankCount: Integer;
```

C++

```
__property int BlankCount;
```

Description

This is BlankCount, a member of class TCustomArray.

105.1.7.5.3 TCustomArray.BlankDateVal Property

Pascal

```
property BlankDateVal: Variant;
```

C++

```
__property Variant BlankDateVal;
```

Description

This is BlankDateVal, a member of class TCustomArray.

105.1.7.5.4 TCustomArray.BlankStringVal Property

Pascal

```
property BlankStringVal: string;
```

C++

```
__property AnsiString BlankStringVal;
```

Description

This is BlankStringVal, a member of class TCustomArray.

105.1.7.5.5 TCustomArray.CompareProc Property

Pascal

```
property CompareProc: TCompareProc;
```

C++

```
__property TCompareProc CompareProc;
```

Description

This is CompareProc, a member of class TCustomArray.

105.1.7.5.6 TCustomArray.DataType Property

Pascal

```
property DataType: Integer;
```

C++

```
__property int DataType;
```

Description

This is DataType, a member of class TCustomArray.

105.1.7.5.7 TCustomArray.Duplicates Property

Pascal

```
property Duplicates: TDuplicates;
```

C++

```
__property TDuplicates Duplicates;
```

Description

This is Duplicates, a member of class TCustomArray.

105.1.7.5.8 TCustomArray.Items Property

Pascal

```
property Items [I: Integer]: Variant;
```

C++

```
__property Variant Items[int I];
```

Description

This is Items, a member of class TCustomArray.

105.1.7.5.9 TCustomArray.List Property

Pascal

```
property List: Pointer;
```

C++

```
__property void * List;
```

Description

This is List, a member of class TCustomArray.

105.1.7.5.10 TCustomArray.MemberCount Property

Pascal

```
property MemberCount: Integer;
```

C++

```
__property int MemberCount;
```

Description

This is MemberCount, a member of class TCustomArray.

105.1.7.5.11 TCustomArray.Sorted Property

Pascal

```
property Sorted: Boolean;
```

C++

```
__property Boolean Sorted;
```

Description

This is Sorted, a member of class TCustomArray.

105.1.8 TDoubleArray Class

Class Hierarchy

Pascal

```
TDoubleArray = class(TBaseArray);
```

C++

```
class TDoubleArray : public TBaseArray;
```

File

Mxarrays

Description

This is class Mxarrays.TDoubleArray.

105.1.8.1 TDoubleArray Members

The following tables list the members exposed by TDoubleArray.

Protected Methods

	Name	Description
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BlockCopy (see page 17311)	This is BlockCopy, a member of class TBaseArray.
	ClearFlag (see page 17311)	This is ClearFlag, a member of class TBaseArray.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetAutoSize (see page 17313)	This is GetAutoSize, a member of class TBaseArray.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Grow (see page 17313)	This is Grow, a member of class TBaseArray.
	HasFlag (see page 17314)	This is HasFlag, a member of class TBaseArray.
	InternalHandleException (see page 17315)	This is InternalHandleException, a member of class TBaseArray.
V	InvalidateItems (see page 17315)	This is InvalidateItems, a member of class TBaseArray.
	RemoveRange (see page 17316)	This is RemoveRange, a member of class TBaseArray.
	SetAutoSize (see page 17316)	This is SetAutoSize, a member of class TBaseArray.
V	SetCapacity (see page 17317)	This is SetCapacity, a member of class TBaseArray.
	SetFlag (see page 17317)	This is SetFlag, a member of class TBaseArray.

	ValidateBounds (see page 17317)	This is ValidateBounds, a member of class TBaseArray.
	ValidIndex (see page 17318)	This is ValidIndex, a member of class TBaseArray.

Public Methods

	Name	Description
	Add (see page 17340)	This is Add, a member of class TDoubleArray.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 17340)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 17311)	Clear removes all items from the array.
V	Create (see page 17339)	Creates an instance of TBaseArray for elements of a specific size.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Delete (see page 17312)	Removes an element from the array.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 17310)	Destroys an instance of TBaseArray (see page 17307).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Exchange (see page 17312)	Provides the interface for a method that swaps the values of two elements in the array.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Find (see page 17341)	This is Find, a member of class TDoubleArray.
	FindItem (see page 17312)	Locates a value in a sorted array and indicates whether an element with that value already exists.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItem (see page 17341)	This is GetItem, a member of class TDoubleArray.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	IndexOf (see page 17341)	Returns the position of the first element with a specified value.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Insert (see page 17314)	Inserts an element into the array and assigns it a value.
	InsertAt (see page 17315)	Adds an element to the array at the position specified by Index.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PutItem (see page 17341)	This is PutItem, a member of class TDoubleArray.
	RemoveItem (see page 17316)	Removes an element from the array.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Sort (see page 17317)	Provides the interface for a method that sorts the elements in the array.

Public Properties

	Name	Description
	AutoSize (see page 17318)	Indicates whether the array grows dynamically as items are added.
	Capacity (see page 17319)	Specifies the maximum number of items that can be added to the array.
	CompareProc (see page 17319)	Implements a sort order for the elements of the array.
	Count (see page 17320)	Specifies the number of items in the array.
	Duplicates (see page 17320)	Specifies whether duplicate items can be added to sorted arrays.
	Items (see page 17342)	This is Items, a member of class TDoubleArray.
R	ItemSize (see page 17321)	Indicates the number of bytes used to store each item in the array.
	Limit (see page 17321)	Specifies the size of the array.
R	List (see page 17321)	Points to the array of values maintained by TBaseArray (see page 17307).
	SortOrder (see page 17322)	Determines whether the array is sorted.

105.1.8.2 TDoubleArray.Create Constructor

Creates an instance of TBaseArray for elements of a specific size.

Pascal

```
constructor Create(itemcount: Integer; dummy: Integer); override;
```

C++

```
virtual __fastcall TDoubleArray(int itemcount, int dummy);
```

Description

Call Create to create an array containing elements with the size specified by the iSize parameter.Create sets the Capacity property to Itemcount and the ItemSize property to iSize. It also initializes the AutoSize property to true and the Count property to 0.

See Also

AutoSize (see page 17318)

Capacity (see page 17319)

Count (see page 17320)

ItemSize (see page 17321)

Create (see page 22287)

105.1.8.3 TDoubleArray Methods

The methods of the TDoubleArray class are listed here.

Public Methods

	Name	Description
	Add (see page 17340)	This is Add, a member of class TDoubleArray.
V	Assign (see page 17340)	Copies the contents of another, similar object.
	Find (see page 17341)	This is Find, a member of class TDoubleArray.
	GetItem (see page 17341)	This is GetItem, a member of class TDoubleArray.
V	IndexOf (see page 17341)	Returns the position of the first element with a specified value.
	PutItem (see page 17341)	This is PutItem, a member of class TDoubleArray.

105.1.8.3.1 TDoubleArray.Add Method

Pascal

```
function Add(Value: Double): Integer;
```

C++

```
__fastcall int Add(Double Value);
```

Description

This is Add, a member of class TDoubleArray.

105.1.8.3.2 TDoubleArray.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo (see page 2931)

EConvertError (see page 22724)

105.1.8.3.3 TDoubleArray.Find Method

Pascal

```
function Find(var Index: Integer; Value: Double): Boolean;
```

C++

```
__fastcall Boolean Find(int Index, Double Value);
```

Description

This is Find, a member of class TDoubleArray.

105.1.8.3.4 TDoubleArray.GetItem Method

Pascal

```
function GetItem(index: Integer): Double;
```

C++

```
__fastcall Double GetItem(int index);
```

Description

This is GetItem, a member of class TDoubleArray.

105.1.8.3.5 TDoubleArray.IndexOf Method

Returns the position of the first element with a specified value.

Pascal

```
function IndexOf(var Item): Integer; override;
```

C++

```
virtual __fastcall int IndexOf( Item);
```

Description

Call IndexOf to obtain the position of the first occurrence of the value Value. IndexOf returns the 0-based index of the element. Thus, if Value matches the first element in the array, IndexOf returns 0, if Value matches the second element, IndexOf returns 1, and so on. If the value is not in the array, IndexOf returns -1.

Note: IndexOf does not work for arrays sorted in descending order.

See Also

FindItem (see page 17312)

GetItem (see page 17313)

105.1.8.3.6 TDoubleArray.PutItem Method

Pascal

```
procedure PutItem(index: Integer; value: Double);
```

C++

```
__fastcall PutItem(int index, Double value);
```

Description

This is PutItem, a member of class TDoubleArray.

105.1.8.4 TDoubleArray Properties

The properties of the TDoubleArray class are listed here.

Public Properties

	Name	Description
	Items (see page 17342)	This is Items, a member of class TDoubleArray.

105.1.8.4.1 TDoubleArray.Items Property

Pascal

```
property Items [Index: Integer]: Double;
```

C++

```
__property Double Items[int Index];
```

Description

This is Items, a member of class TDoubleArray.

105.1.9 TIndexArray Class

Class Hierarchy

Mxarrays.TIndexArray

Pascal

```
TIndexArray = class;
```

C++

```
class TIndexArray;
```

File

Mxarrays

Description

This is class Mxarrays.TIndexArray.

105.1.9.1 TIndexArray Members

The following tables list the members exposed by TIndexArray.

Protected Fields

	Name	Description
	idxElements (see page 17344)	This is idxElements, a member of class TIndexArray.

Public Methods

	Name	Description
	Add (see page 17344)	This is Add, a member of class TIndexArray.
	Create (see page 17343)	This is Create, a member of class TIndexArray.
V	Destroy (see page 17343)	This is Destroy, a member of class TIndexArray.
	Expand (see page 17344)	This is Expand, a member of class TIndexArray.
	SetSize (see page 17344)	This is SetSize, a member of class TIndexArray.

Public Properties

	Name	Description
	Autosize (see page 17345)	This is Autosize, a member of class TIndexArray.
	Capacity (see page 17345)	This is Capacity, a member of class TIndexArray.
R	Count (see page 17345)	This is Count, a member of class TIndexArray.
	Items (see page 17346)	This is Items, a member of class TIndexArray.
R	MemoryUsage (see page 17346)	This is MemoryUsage, a member of class TIndexArray.

105.1.9.2 TIndexArray.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TIndexArray();
```

Description

This is Create, a member of class TIndexArray.

105.1.9.3 TIndexArray.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIndexArray();
```

Description

This is Destroy, a member of class TIndexArray.

105.1.9.4 TIndexArray Fields

The fields of the TIndexArray class are listed here.

Protected Fields

	Name	Description
	idxElements (see page 17344)	This is idxElements, a member of class TIndexArray.

105.1.9.4.1 TIndexArray.idxElements Field

Pascal

```
idxElements: PIndexElements;
```

C++

```
PIndexElements idxElements;
```

Description

This is idxElements, a member of class TIndexArray.

105.1.9.5 TIndexArray Methods

The methods of the TIndexArray class are listed here.

Public Methods

	Name	Description
	Add (see page 17344)	This is Add, a member of class TIndexArray.
	Expand (see page 17344)	This is Expand, a member of class TIndexArray.
	SetSize (see page 17344)	This is SetSize, a member of class TIndexArray.

105.1.9.5.1 TIndexArray.Add Method

Pascal

```
function Add(const NewValue: TSmallIntArray): Integer;
```

C++

```
__fastcall int Add(const TSmallIntArray NewValue);
```

Description

This is Add, a member of class TIndexArray.

105.1.9.5.2 TIndexArray.Expand Method

Pascal

```
procedure Expand;
```

C++

```
__fastcall Expand();
```

Description

This is Expand, a member of class TIndexArray.

105.1.9.5.3 TIndexArray.SetSize Method

Pascal

```
procedure SetSize(Elements: TIndexNDX);
```

C++

```
__fastcall SetSize(TIndexNDX Elements);
```

Description

This is SetSize, a member of class TIndexArray.

105.1.9.6 TIndexArray Properties

The properties of the TIndexArray class are listed here.

Public Properties

	Name	Description
	Autosize (see page 17345)	This is Autosize, a member of class TIndexArray.
	Capacity (see page 17345)	This is Capacity, a member of class TIndexArray.
R	Count (see page 17345)	This is Count, a member of class TIndexArray.
	Items (see page 17346)	This is Items, a member of class TIndexArray.
R	MemoryUsage (see page 17346)	This is MemoryUsage, a member of class TIndexArray.

105.1.9.6.1 TIndexArray.Autosize Property

Pascal

```
property Autosize: Boolean;
```

C++

```
__property Boolean Autosize;
```

Description

This is Autosize, a member of class TIndexArray.

105.1.9.6.2 TIndexArray.Capacity Property

Pascal

```
property Capacity: TIndexNDX;
```

C++

```
__property TIndexNDX Capacity;
```

Description

This is Capacity, a member of class TIndexArray.

105.1.9.6.3 TIndexArray.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TIndexArray.

105.1.9.6.4 TIndexArray.Items Property

Pascal

```
property Items [Element: TIndexNDX]: TSmallIntArray;
```

C++

```
__property TSmallIntArray Items[TIndexNDX Element];
```

Description

This is Items, a member of class TIndexArray.

105.1.9.6.5 TIndexArray.MemoryUsage Property

Pascal

```
property MemoryUsage: Integer;
```

C++

```
__property int MemoryUsage;
```

Description

This is MemoryUsage, a member of class TIndexArray.

105.1.10 TIntArray Class

Class Hierarchy

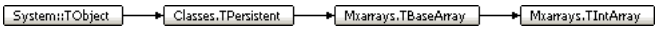

Pascal

```
TIntArray = class(TBaseArray);
```

C++

```
class TIntArray : public TBaseArray;
```

File

Mxarrays

Description

This is class Mxarrays.TIntArray.

105.1.10.1 TIntArray Members

The following tables list the members exposed by TIntArray.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BlockCopy (see page 17311)	This is BlockCopy, a member of class TBaseArray.
	ClearFlag (see page 17311)	This is ClearFlag, a member of class TBaseArray.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetAutoSize (see page 17313)	This is GetAutoSize, a member of class TBaseArray.
	GetOwner (see page 2932)	Returns the owner of an object.
	Grow (see page 17313)	This is Grow, a member of class TBaseArray.
	HasFlag (see page 17314)	This is HasFlag, a member of class TBaseArray.
	InternalHandleException (see page 17315)	This is InternalHandleException, a member of class TBaseArray.
	InvalidateItems (see page 17315)	This is InvalidateItems, a member of class TBaseArray.
	RemoveRange (see page 17316)	This is RemoveRange, a member of class TBaseArray.
	SetAutoSize (see page 17316)	This is SetAutoSize, a member of class TBaseArray.
	SetCapacity (see page 17317)	This is SetCapacity, a member of class TBaseArray.
	SetFlag (see page 17317)	This is SetFlag, a member of class TBaseArray.
	ValidateBounds (see page 17317)	This is ValidateBounds, a member of class TBaseArray.
	ValidIndex (see page 17318)	This is ValidIndex, a member of class TBaseArray.

Public Methods

	Name	Description
	Add (see page 17349)	This is Add, a member of class TIntArray.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 17349)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 17311)	Clear removes all items from the array.
	Create (see page 17348)	Creates an instance of TBaseArray for elements of a specific size.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 17312)	Removes an element from the array.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 17310)	Destroys an instance of TBaseArray (see page 17307).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Exchange (see page 17312)	Provides the interface for a method that swaps the values of two elements in the array.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Find (see page 17350)	This is Find, a member of class TIntArray.
	FindItem (see page 17312)	Locates a value in a sorted array and indicates whether an element with that value already exists.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItem (see page 17350)	This is GetItem, a member of class TIntArray.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	IndexOf (see page 17314)	Returns the position of the first element with a specified value.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

V	Insert (see page 17314)	Inserts an element into the array and assigns it a value.
	InsertAt (see page 17315)	Adds an element to the array at the position specified by Index.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PutItem (see page 17350)	This is PutItem, a member of class TIntArray.
	RemoveItem (see page 17316)	Removes an element from the array.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Sort (see page 17317)	Provides the interface for a method that sorts the elements in the array.

Public Properties

	Name	Description
	AutoSize (see page 17318)	Indicates whether the array grows dynamically as items are added.
	Capacity (see page 17319)	Specifies the maximum number of items that can be added to the array.
	CompareProc (see page 17319)	Implements a sort order for the elements of the array.
	Count (see page 17320)	Specifies the number of items in the array.
	Duplicates (see page 17320)	Specifies whether duplicate items can be added to sorted arrays.
	Items (see page 17351)	This is Items, a member of class TIntArray.
R	ItemSize (see page 17321)	Indicates the number of bytes used to store each item in the array.
	Limit (see page 17321)	Specifies the size of the array.
R	List (see page 17321)	Points to the array of values maintained by TBaseArray (see page 17307).
	SortOrder (see page 17322)	Determines whether the array is sorted.

105.1.10.2 TIntArray.Create Constructor

Creates an instance of TBaseArray for elements of a specific size.

Pascal

```
constructor Create(itemcount: Integer; dummy: Integer); override;
```

C++

```
virtual __fastcall TIntArray(int itemcount, int dummy);
```

Description

Call Create to create an array containing elements with the size specified by the iSize parameter.Create sets the Capacity property to Itemcount and the ItemSize property to iSize. It also initializes the AutoSize property to true and the Count property to 0.

See Also

AutoSize (see page 17318)

Capacity (see page 17319)

Count (see page 17320)

ItemSize (see page 17321)

Create (see page 22287)

105.1.10.3 TIntArray Methods

The methods of the TIntArray class are listed here.

Public Methods

	Name	Description
	Add (see page 17349)	This is Add, a member of class TIntArray.
V	Assign (see page 17349)	Copies the contents of another, similar object.
	Find (see page 17350)	This is Find, a member of class TIntArray.
	GetItem (see page 17350)	This is GetItem, a member of class TIntArray.
	PutItem (see page 17350)	This is PutItem, a member of class TIntArray.

105.1.10.3.1 TIntArray.Add Method

Pascal

```
function Add(Value: Integer): Integer;
```

C++

```
__fastcall int Add(int Value);
```

Description

This is Add, a member of class TIntArray.

105.1.10.3.2 TIntArray.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of

the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo (see page 2931)

EConvertError (see page 22724)

105.1.10.3.3 TIntArray.Find Method

Pascal

```
function Find(var Index: Integer; Value: Integer): Boolean;
```

C++

```
__fastcall Boolean Find(int Index, int Value);
```

Description

This is Find, a member of class TIntArray.

105.1.10.3.4 TIntArray.GetItem Method

Pascal

```
function GetItem(index: Integer): Integer;
```

C++

```
__fastcall int GetItem(int index);
```

Description

This is GetItem, a member of class TIntArray.

105.1.10.3.5 TIntArray.PutItem Method

Pascal

```
procedure PutItem(index: Integer; value: Integer);
```

C++

```
__fastcall PutItem(int index, int value);
```

Description

This is PutItem, a member of class TIntArray.

105.1.10.4 TIntArray Properties

The properties of the TIntArray class are listed here.

Public Properties

	Name	Description
	Items (see page 17351)	This is Items, a member of class TIntArray.

105.1.10.4.1 TIntArray.Items Property

Pascal

```
property Items [Index: Integer]: Integer;
```

C++

```
__property int Items[int Index];
```

Description

This is Items, a member of class TIntArray.

105.1.11 TPointerArray Class

Class Hierarchy

Pascal

```
TPointerArray = class(TBaseArray);
```

C++

```
class TPointerArray : public TBaseArray;
```

File

Mxarrays

Description

This is class Mxarrays.TPointerArray.

105.1.11.1 TPointerArray Members

The following tables list the members exposed by TPointerArray.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BlockCopy (see page 17311)	This is BlockCopy, a member of class TBaseArray.
	ClearFlag (see page 17311)	This is ClearFlag, a member of class TBaseArray.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetAutoSize (see page 17313)	This is GetAutoSize, a member of class TBaseArray.
	GetOwner (see page 2932)	Returns the owner of an object.
	Grow (see page 17313)	This is Grow, a member of class TBaseArray.
	HasFlag (see page 17314)	This is HasFlag, a member of class TBaseArray.
	InternalHandleException (see page 17315)	This is InternalHandleException, a member of class TBaseArray.
	RemoveRange (see page 17316)	This is RemoveRange, a member of class TBaseArray.
	SetAutoSize (see page 17316)	This is SetAutoSize, a member of class TBaseArray.
	SetCapacity (see page 17317)	This is SetCapacity, a member of class TBaseArray.
	SetFlag (see page 17317)	This is SetFlag, a member of class TBaseArray.
	ValidateBounds (see page 17317)	This is ValidateBounds, a member of class TBaseArray.

	ValidIndex (see page 17318)	This is ValidIndex, a member of class TBaseArray.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 17311)	Clear removes all items from the array.
	CloneItem (see page 17354)	This is CloneItem, a member of class TPointerArray.
	CopyFrom (see page 17354)	This is CopyFrom, a member of class TPointerArray.
	CopyTo (see page 17354)	This is CopyTo, a member of class TPointerArray.
	Create (see page 17353)	Creates an instance of TBaseArray for elements of a specific size.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 17312)	Removes an element from the array.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 17310)	Destroys an instance of TBaseArray (see page 17307).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Exchange (see page 17312)	Provides the interface for a method that swaps the values of two elements in the array.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItem (see page 17312)	Locates a value in a sorted array and indicates whether an element with that value already exists.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeItem (see page 17354)	This is FreeItem, a member of class TPointerArray.
	GetData (see page 17355)	This is GetData, a member of class TPointerArray.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItem (see page 17313)	Returns the element in the array at the indicated position.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	IndexOf (see page 17314)	Returns the position of the first element with a specified value.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 17314)	Inserts an element into the array and assigns it a value.
	InsertAt (see page 17315)	Adds an element to the array at the position specified by Index.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InvalidateItems (see page 17355)	This is InvalidateItems, a member of class TPointerArray.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PutData (see page 17355)	This is PutData, a member of class TPointerArray.
	PutItem (see page 17315)	Replaces the value of an element in the array.

	RemoveItem (see page 17316)	Removes an element from the array.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Sort (see page 17317)	Provides the interface for a method that sorts the elements in the array.

Public Properties

	Name	Description
	AsPtr (see page 17355)	This is AsPtr, a member of class TPointerArray.
	AutoSize (see page 17318)	Indicates whether the array grows dynamically as items are added.
	Capacity (see page 17319)	Specifies the maximum number of items that can be added to the array.
	CompareProc (see page 17319)	Implements a sort order for the elements of the array.
	Count (see page 17320)	Specifies the number of items in the array.
	Data (see page 17356)	This is Data, a member of class TPointerArray.
	Duplicates (see page 17320)	Specifies whether duplicate items can be added to sorted arrays.
R	ItemSize (see page 17321)	Indicates the number of bytes used to store each item in the array.
	Limit (see page 17321)	Specifies the size of the array.
R	List (see page 17321)	Points to the array of values maintained by TBaseArray (see page 17307).
	SortOrder (see page 17322)	Determines whether the array is sorted.

105.1.11.2 TPointerArray.Create Constructor

Creates an instance of TBaseArray for elements of a specific size.

Pascal

```
constructor Create(itemcount: Integer; dummy: Integer); override;
```

C++

```
virtual __fastcall TPointerArray(int itemcount, int dummy);
```

Description

Call Create to create an array containing elements with the size specified by the iSize parameter.Create sets the Capacity property to Itemcount and the ItemSize property to iSize. It also initializes the AutoSize property to true and the Count property to 0.

See Also

AutoSize (see page 17318)

Capacity (see page 17319)

Count (see page 17320)

ItemSize (see page 17321)

Create (see page 22287)

105.1.11.3 TPointerArray Methods

The methods of the TPointerArray class are listed here.

Public Methods

	Name	Description
V	CloneItem (see page 17354)	This is CloneItem, a member of class TPointerArray.
	CopyFrom (see page 17354)	This is CopyFrom, a member of class TPointerArray.
	CopyTo (see page 17354)	This is CopyTo, a member of class TPointerArray.
V	FreeItem (see page 17354)	This is FreeItem, a member of class TPointerArray.

	GetData (see page 17355)	This is GetData, a member of class TPointerArray.
	InvalidateItems (see page 17355)	This is InvalidateItems, a member of class TPointerArray.
	PutData (see page 17355)	This is PutData, a member of class TPointerArray.

105.1.11.3.1 TPointerArray.CloneItem Method

Pascal

```
function CloneItem(item: Pointer): Pointer; virtual;
```

C++

```
virtual __fastcall void * CloneItem(void * item);
```

Description

This is CloneItem, a member of class TPointerArray.

105.1.11.3.2 TPointerArray.CopyFrom Method

Pascal

```
procedure CopyFrom(var Source; toIndex: Integer; numItems: Integer);
```

C++

```
__fastcall CopyFrom( Source, int toIndex, int numItems);
```

Description

This is CopyFrom, a member of class TPointerArray.

105.1.11.3.3 TPointerArray.CopyTo Method

Pascal

```
procedure CopyTo(var Dest; fromIndex: Integer; numItems: Integer);
```

C++

```
__fastcall CopyTo( Dest, int fromIndex, int numItems);
```

Description

This is CopyTo, a member of class TPointerArray.

105.1.11.3.4 TPointerArray.FreeItem Method

Pascal

```
procedure FreeItem(item: Pointer); virtual;
```

C++

```
virtual __fastcall FreeItem(void * item);
```

Description

This is FreeItem, a member of class TPointerArray.

105.1.11.3.5 TPointerArray.GetData Method

Pascal

```
function GetData(index: Integer): Pointer;
```

C++

```
__fastcall void * GetData(int index);
```

Description

This is GetData, a member of class TPointerArray.

105.1.11.3.6 TPointerArray.InvalidateItems Method

Pascal

```
procedure InvalidateItems(atIndex: Integer; numItems: Integer); override;
```

C++

```
virtual __fastcall InvalidateItems(int atIndex, int numItems);
```

Description

This is InvalidateItems, a member of class TPointerArray.

105.1.11.3.7 TPointerArray.PutData Method

Pascal

```
procedure PutData(index: Integer; value: Pointer);
```

C++

```
__fastcall PutData(int index, void * value);
```

Description

This is PutData, a member of class TPointerArray.

105.1.11.4 TPointerArray Properties

The properties of the TPointerArray class are listed here.

Public Properties

	Name	Description
	AsPtr (see page 17355)	This is AsPtr, a member of class TPointerArray.
	Data (see page 17356)	This is Data, a member of class TPointerArray.

105.1.11.4.1 TPointerArray.AsPtr Property

Pascal

```
property AsPtr [Index: Integer]: Pointer;
```

C++

```
__property void * AsPtr[int Index];
```

Description

This is AsPtr, a member of class TPointerArray.

105.1.11.4.2 TPointerArray.Data Property

Pascal

```
property Data [Index: Integer]: Pointer;
```

C++

```
__property void * Data[int Index];
```

Description

This is Data, a member of class TPointerArray.

105.1.12 TSingleArray Class

Class Hierarchy

Pascal

```
TSingleArray = class(TBaseArray);
```

C++

```
class TSingleArray : public TBaseArray;
```

File

Mxarrays

Description

This is class Mxarrays.TSingleArray.

105.1.12.1 TSingleArray Members

The following tables list the members exposed by TSingleArray.

Protected Methods

	Name	Description
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BlockCopy (see page 17311)	This is BlockCopy, a member of class TBaseArray.
	ClearFlag (see page 17311)	This is ClearFlag, a member of class TBaseArray.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetAutoSize (see page 17313)	This is GetAutoSize, a member of class TBaseArray.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Grow (see page 17313)	This is Grow, a member of class TBaseArray.
	HasFlag (see page 17314)	This is HasFlag, a member of class TBaseArray.
	InternalHandleException (see page 17315)	This is InternalHandleException, a member of class TBaseArray.
V	InvalidateItems (see page 17315)	This is InvalidateItems, a member of class TBaseArray.
	RemoveRange (see page 17316)	This is RemoveRange, a member of class TBaseArray.

	SetAutoSize (see page 17316)	This is SetAutoSize, a member of class TBaseArray.
	SetCapacity (see page 17317)	This is SetCapacity, a member of class TBaseArray.
	SetFlag (see page 17317)	This is SetFlag, a member of class TBaseArray.
	ValidateBounds (see page 17317)	This is ValidateBounds, a member of class TBaseArray.
	ValidIndex (see page 17318)	This is ValidIndex, a member of class TBaseArray.

Public Methods

	Name	Description
	Add (see page 17359)	This is Add, a member of class TSingleArray.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 17359)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 17311)	Clear removes all items from the array.
	Create (see page 17358)	Creates an instance of TBaseArray for elements of a specific size.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 17312)	Removes an element from the array.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 17310)	Destroys an instance of TBaseArray (see page 17307).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Exchange (see page 17312)	Provides the interface for a method that swaps the values of two elements in the array.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Find (see page 17360)	This is Find, a member of class TSingleArray.
	FindItem (see page 17312)	Locates a value in a sorted array and indicates whether an element with that value already exists.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItem (see page 17360)	This is GetItem, a member of class TSingleArray.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	IndexOf (see page 17360)	Returns the position of the first element with a specified value.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 17314)	Inserts an element into the array and assigns it a value.
	InsertAt (see page 17315)	Adds an element to the array at the position specified by Index.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PutItem (see page 17360)	This is PutItem, a member of class TSingleArray.
	RemoveItem (see page 17316)	Removes an element from the array.

V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Sort (see page 17317)	Provides the interface for a method that sorts the elements in the array.

Public Properties

	Name	Description
	AutoSize (see page 17318)	Indicates whether the array grows dynamically as items are added.
	Capacity (see page 17319)	Specifies the maximum number of items that can be added to the array.
	CompareProc (see page 17319)	Implements a sort order for the elements of the array.
	Count (see page 17320)	Specifies the number of items in the array.
	Duplicates (see page 17320)	Specifies whether duplicate items can be added to sorted arrays.
	Items (see page 17361)	This is Items, a member of class TSingleArray.
R	ItemSize (see page 17321)	Indicates the number of bytes used to store each item in the array.
	Limit (see page 17321)	Specifies the size of the array.
R	List (see page 17321)	Points to the array of values maintained by TBaseArray (see page 17307).
	SortOrder (see page 17322)	Determines whether the array is sorted.

105.1.12.2 TSingleArray.Create Constructor

Creates an instance of TBaseArray for elements of a specific size.

Pascal

```
constructor Create(itemcount: Integer; dummy: Integer); override;
```

C++

```
virtual __fastcall TSingleArray(int itemcount, int dummy);
```

Description

Call Create to create an array containing elements with the size specified by the iSize parameter.Create sets the Capacity property to Itemcount and the ItemSize property to iSize. It also initializes the AutoSize property to true and the Count property to 0.

See Also

AutoSize (see page 17318)

Capacity (see page 17319)

Count (see page 17320)

ItemSize (see page 17321)

Create (see page 22287)

105.1.12.3 TSingleArray Methods

The methods of the TSingleArray class are listed here.

Public Methods

	Name	Description
	Add (see page 17359)	This is Add, a member of class TSingleArray.
V	Assign (see page 17359)	Copies the contents of another, similar object.
	Find (see page 17360)	This is Find, a member of class TSingleArray.
	GetItem (see page 17360)	This is GetItem, a member of class TSingleArray.
V	IndexOf (see page 17360)	Returns the position of the first element with a specified value.
	PutItem (see page 17360)	This is PutItem, a member of class TSingleArray.

105.1.12.3.1 TSingleArray.Add Method

Pascal

```
function Add(Value: Single): Integer;
```

C++

```
__fastcall int Add(Single Value);
```

Description

This is Add, a member of class TSingleArray.

105.1.12.3.2 TSingleArray.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo (see page 2931)

EConvertError (see page 22724)

105.1.12.3.3 TSingleArray.Find Method

Pascal

```
function Find(var Index: Integer; Value: Single): Boolean;
```

C++

```
__fastcall Boolean Find(int Index, Single Value);
```

Description

This is Find, a member of class TSingleArray.

105.1.12.3.4 TSingleArray.GetItem Method

Pascal

```
function GetItem(index: Integer): Single;
```

C++

```
__fastcall Single GetItem(int index);
```

Description

This is GetItem, a member of class TSingleArray.

105.1.12.3.5 TSingleArray.IndexOf Method

Returns the position of the first element with a specified value.

Pascal

```
function IndexOf(var Item): Integer; override;
```

C++

```
virtual __fastcall int IndexOf( Item);
```

Description

Call IndexOf to obtain the position of the first occurrence of the value Value. IndexOf returns the 0-based index of the element. Thus, if Value matches the first element in the array, IndexOf returns 0, if Value matches the second element, IndexOf returns 1, and so on. If the value is not in the array, IndexOf returns -1.

Note: IndexOf does not work for arrays sorted in descending order.

See Also

FindItem (see page 17312)

GetItem (see page 17313)

105.1.12.3.6 TSingleArray.PutItem Method

Pascal

```
procedure PutItem(index: Integer; value: Single);
```

C++

```
__fastcall PutItem(int index, Single value);
```

Description

This is PutItem, a member of class TSingleArray.

105.1.12.4 TSingleArray Properties

The properties of the TSingleArray class are listed here.

Public Properties

	Name	Description
	Items (see page 17361)	This is Items, a member of class TSingleArray.

105.1.12.4.1 TSingleArray.Items Property

Pascal

```
property Items [Index: Integer]: Single;
```

C++

```
__property Single Items[int Index];
```

Description

This is Items, a member of class TSingleArray.

105.1.13 TSmallIntArray Class

TSmallIntArray is a dynamically resizable array of 16-bit integers.

Class Hierarchy

System::TObject → Classes.TPersistent → Mxarrays.TBaseArray → Mxarrays.TSmallIntArray

Pascal

```
TSmallIntArray = class(TBaseArray);
```

C++

```
class TSmallIntArray : public TBaseArray;
```

File

Mxarrays

Description

Use TSmallIntArray to represent an array of short integers when the number of elements in the array is unknown. For example, decision cubes use TSmallIntArray objects to represent the coordinates of a cell in the multidimensional cache of summary values.

See Also

TDecisionCube (see page 17442)

105.1.13.1 TSmallIntArray Members

The following tables list the members exposed by TSmallIntArray.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BlockCopy (see page 17311)	This is BlockCopy, a member of class TBaseArray.
	ClearFlag (see page 17311)	This is ClearFlag, a member of class TBaseArray.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetAutoSize (see page 17313)	This is GetAutoSize, a member of class TBaseArray.
	GetOwner (see page 2932)	Returns the owner of an object.
	Grow (see page 17313)	This is Grow, a member of class TBaseArray.
	HasFlag (see page 17314)	This is HasFlag, a member of class TBaseArray.
	InternalHandleException (see page 17315)	This is InternalHandleException, a member of class TBaseArray.
	InvalidateItems (see page 17315)	This is InvalidateItems, a member of class TBaseArray.
	RemoveRange (see page 17316)	This is RemoveRange, a member of class TBaseArray.
	SetAutoSize (see page 17316)	This is SetAutoSize, a member of class TBaseArray.
	SetCapacity (see page 17317)	This is SetCapacity, a member of class TBaseArray.
	SetFlag (see page 17317)	This is SetFlag, a member of class TBaseArray.
	ValidateBounds (see page 17317)	This is ValidateBounds, a member of class TBaseArray.
	ValidIndex (see page 17318)	This is ValidIndex, a member of class TBaseArray.

Public Methods

	Name	Description
	Add (see page 17364)	Adds a new element to the array.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 17365)	Copies the Items (see page 17366) property of another TSmallIntArray (see page 17361) object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 17311)	Clear removes all items from the array.
	Create (see page 17363)	Creates an instance of TSmallIntArray (see page 17361).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 17312)	Removes an element from the array.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 17310)	Destroys an instance of TBaseArray (see page 17307).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Exchange (see page 17312)	Provides the interface for a method that swaps the values of two elements in the array.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItem (see page 17312)	Locates a value in a sorted array and indicates whether an element with that value already exists.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItem (see page 17365)	Returns the integer at the indicated position.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	IndexOf (see page 17314)	Returns the position of the first element with a specified value.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Insert (see page 17314)	Inserts an element into the array and assigns it a value.
	InsertAt (see page 17315)	Adds an element to the array at the position specified by Index.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PutItem (see page 17366)	Replaces the value of an element in the array.
	RemoveItem (see page 17316)	Removes an element from the array.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Sort (see page 17317)	Provides the interface for a method that sorts the elements in the array.

Public Properties

	Name	Description
	AutoSize (see page 17318)	Indicates whether the array grows dynamically as items are added.
	Capacity (see page 17319)	Specifies the maximum number of items that can be added to the array.
	CompareProc (see page 17319)	Implements a sort order for the elements of the array.
	Count (see page 17320)	Specifies the number of items in the array.
	Duplicates (see page 17320)	Specifies whether duplicate items can be added to sorted arrays.
	Items (see page 17366)	Lists the elements in the array.
R	ItemSize (see page 17321)	Indicates the number of bytes used to store each item in the array.
	Limit (see page 17321)	Specifies the size of the array.
R	List (see page 17321)	Points to the array of values maintained by TBaseArray (see page 17307).
	SortOrder (see page 17322)	Determines whether the array is sorted.

105.1.13.2 TSmallIntArray.Create Constructor

Creates an instance of TSmallIntArray (see page 17361).

Pascal

```
constructor Create(itemcount: Integer; dummy: Integer); override;
```

C++

```
virtual __fastcall TSmallIntArray(int itemcount, int dummy);
```

Description

Call Create to create and initialize a TSmallIntArray (see page 17361) object. Create sets the Capacity (see page 17319) property to Itemcount and ignores the iSize parameter.

Create also initializes the following properties:

ItemSize (see page 17321) is set to 2 (the size of a small integer).

CompareProc (see page 17319) is initialized to a function that returns the difference between two 16-bit integers.

AutoSize (see page 17318) is set to true.

Count (see page 17320) is set to 0.

See Also

AutoSize (see page 17318)

Capacity (see page 17319)

CompareProc (see page 17319)

Count (see page 17320)

ItemSize (see page 17321)

Create (see page 17309)

105.1.13.3 TSmallIntArray Methods

The methods of the TSmallIntArray class are listed here.

Public Methods

	Name	Description
	Add (see page 17364)	Adds a new element to the array.
	Assign (see page 17365)	Copies the Items (see page 17366) property of another TSmallIntArray (see page 17361) object.
	GetItem (see page 17365)	Returns the integer at the indicated position.
	PutItem (see page 17366)	Replaces the value of an element in the array.

105.1.13.3.1 TSmallIntArray.Add Method

Adds a new element to the array.

Pascal

```
function Add(Value: SmallInt): Integer;
```

C++

```
__fastcall int Add(SmallInt Value);
```

Description

Call Add to add an item to the end of an unsorted array, or to the appropriate position in a sorted array. To insert an element at a specified position within the array, use the Insert (see page 17314) method instead.

Note: Add assumes that sorted arrays are sorted in ascending order. To use Add on an array where SortOrder (see page 17322) is TS_DESCENDING, temporarily change the CompareProc (see page 17319) property to a function that multiplies the value returned by CompareProc (see page 17319) by -1.

See Also

CompareProc (see page 17319)

Insert (see page 17314)

SortOrder (see page 17322)

105.1.13.3.2 TSmallIntArray.Assign Method

Copies the Items (see page 17366) property of another TSmallIntArray (see page 17361) object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the array from another object. If the Source parameter is a TSmallIntArray (see page 17361) object, Assign replaces the Items (see page 17366) property with the array from the source array. Otherwise, Assign calls the inherited method which will assign values from any Source object that handles a TSmallIntArray (see page 17361) in its AssignTo method.

Note: Assign only changes the list of elements. It does not copy the AutoSize (see page 17318), Capacity (see page 17319), CompareProc (see page 17319), Duplicates (see page 17320), or SortOrder (see page 17322) properties. If the AutoSize (see page 17318) property is false, be sure that the value of Capacity (see page 17319) is at least as big as the Count (see page 17320) property of the source array. If the array is sorted, source items will appear in sorted order. If the Duplicates (see page 17320) property is not dupAccept, duplicate elements will not be copied from the source array.

See Also

AutoSize (see page 17318)

Capacity (see page 17319)

Count (see page 17320)

Items (see page 17366)

AssignTo (see page 2931)

105.1.13.3.3 TSmallIntArray.GetItem Method

Returns the integer at the indicated position.

Pascal

```
function GetItem(index: Integer): SmallInt;
```

C++

```
__fastcall SmallInt GetItem(int index);
```

Description

Call GetItem to obtain the value of an integer in the array. The Index parameter indicates the element of the array, where the first element has the index 0, the second element has the index 1, and so on.

See Also

FindItem (see page 17312)

IndexOf (see page 17314)

Items (see page 17366)

105.1.13.3.4 TSmallIntArray.PutItem Method

Replaces the value of an element in the array.

Pascal

```
procedure PutItem(index: Integer; value: SmallInt);
```

C++

```
__fastcall PutItem(int index, SmallInt value);
```

Description

Call PutItem to change the value of an integer to the value Value. Unlike the Insert (see page 17314) method, PutItem does not add a new item to the array, but replaces the value of an existing element. The Index parameter indicates which element to change, where 0 specifies the first element, 1 specifies the second element, and so on.

Note: Calling PutItem with Index set to the value of the Count (see page 17320) property will add a new element to the end of the array. This is the only case where PutItem adds a new element instead of replacing an existing one.

See Also

Count (see page 17320)

GetItem (see page 17365)

Insert (see page 17314)

Items (see page 17366)

105.1.13.4 TSmallIntArray Properties

The properties of the TSmallIntArray class are listed here.

Public Properties

	Name	Description
	Items (see page 17366)	Lists the elements in the array.

105.1.13.4.1 TSmallIntArray.Items Property

Lists the elements in the array.

Pascal

```
property Items [Index:Integer]: SmallInt;
```

C++

```
__property SmallInt Items[int Index];
```

Description

Use Items to read or set the value of the element indicated by the Index parameter, where 0 specifies the first element in the array, 1 specifies the second parameter, and so on.

Note: In Delphi code, Items is the default property for TSmallIntArray (see page 17361), which means the name of the Items property need not be included when using it. Thus, the line

```
FirstInt := SmallIntArray1.Items[0];
```

Note: can be written

```
FirstInt := SmallIntArray1[0].
```

See Also

GetItem (see page 17365)

List (see page 17321)

PutItem (see page 17366)

105.1.14 TStringArray Class

Class Hierarchy

Pascal

```
TStringArray = class(TBaseArray);
```

C++

```
class TStringArray : public TBaseArray;
```

File

Mxarrays

Description

This is class Mxarrays.TStringArray.

105.1.14.1 TStringArray Members

The following tables list the members exposed by TStringArray.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BlockCopy (see page 17311)	This is BlockCopy, a member of class TBaseArray.
	ClearFlag (see page 17311)	This is ClearFlag, a member of class TBaseArray.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetAutoSize (see page 17313)	This is GetAutoSize, a member of class TBaseArray.
	GetObject (see page 17371)	This is GetObject, a member of class TStringArray.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetString (see page 17372)	This is GetString, a member of class TStringArray.
	Grow (see page 17372)	This is Grow, a member of class TStringArray.
	HasFlag (see page 17314)	This is HasFlag, a member of class TBaseArray.
	InternalHandleException (see page 17315)	This is InternalHandleException, a member of class TBaseArray.
	InvalidateItems (see page 17373)	This is InvalidateItems, a member of class TStringArray.
	PutObject (see page 17373)	This is PutObject, a member of class TStringArray.
	PutString (see page 17373)	This is PutString, a member of class TStringArray.
	RemoveRange (see page 17316)	This is RemoveRange, a member of class TBaseArray.

	SetAutoSize (see page 17316)	This is SetAutoSize, a member of class TBaseArray.
V	SetCapacity (see page 17317)	This is SetCapacity, a member of class TBaseArray.
	SetFlag (see page 17317)	This is SetFlag, a member of class TBaseArray.
	ValidateBounds (see page 17317)	This is ValidateBounds, a member of class TBaseArray.
	ValidIndex (see page 17318)	This is ValidIndex, a member of class TBaseArray.

Public Methods

	Name	Description
	Add (see page 17370)	This is Add, a member of class TStringArray.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 17370)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 17311)	Clear removes all items from the array.
V	Create (see page 17369)	Creates an instance of TBaseArray for elements of a specific size.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Delete (see page 17312)	Removes an element from the array.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 17310)	Destroys an instance of TBaseArray (see page 17307).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Exchange (see page 17371)	Provides the interface for a method that swaps the values of two elements in the array.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Find (see page 17371)	This is Find, a member of class TStringArray.
	FindItem (see page 17312)	Locates a value in a sorted array and indicates whether an element with that value already exists.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItem (see page 17313)	Returns the element in the array at the indicated position.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	IndexOf (see page 17372)	Returns the position of the first element with a specified value.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Insert (see page 17372)	Inserts an element into the array and assigns it a value.
	InsertAt (see page 17315)	Adds an element to the array at the position specified by Index.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PutItem (see page 17315)	Replaces the value of an element in the array.
	RemoveItem (see page 17316)	Removes an element from the array.

V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Sort (see page 17373)	Provides the interface for a method that sorts the elements in the array.

Public Properties

	Name	Description
	AutoSize (see page 17318)	Indicates whether the array grows dynamically as items are added.
	Capacity (see page 17319)	Specifies the maximum number of items that can be added to the array.
	CompareProc (see page 17319)	Implements a sort order for the elements of the array.
	Count (see page 17320)	Specifies the number of items in the array.
	Duplicates (see page 17320)	Specifies whether duplicate items can be added to sorted arrays.
R	ItemSize (see page 17321)	Indicates the number of bytes used to store each item in the array.
	Limit (see page 17321)	Specifies the size of the array.
R	List (see page 17321)	Points to the array of values maintained by TBaseArray (see page 17307).
	SortOrder (see page 17322)	Determines whether the array is sorted.
	Strings (see page 17374)	This is Strings, a member of class TStringArray.

105.1.14.2 TStringArray.Create Constructor

Creates an instance of TBaseArray for elements of a specific size.

Pascal

```
constructor Create(itemcount: Integer; dummy: Integer); override;
```

C++

```
virtual __fastcall TStringArray(int itemcount, int dummy);
```

Description

Call Create to create an array containing elements with the size specified by the iSize parameter.Create sets the Capacity property to Itemcount and the ItemSize property to iSize. It also initializes the AutoSize property to true and the Count property to 0.

See Also

AutoSize (see page 17318)

Capacity (see page 17319)

Count (see page 17320)

ItemSize (see page 17321)

Create (see page 22287)

105.1.14.3 TStringArray Methods

The methods of the TStringArray class are listed here.

Protected Methods

	Name	Description
	GetObject (see page 17371)	This is GetObject, a member of class TStringArray.
	GetString (see page 17372)	This is GetString, a member of class TStringArray.
V	Grow (see page 17372)	This is Grow, a member of class TStringArray.
V	InvalidateItems (see page 17373)	This is InvalidateItems, a member of class TStringArray.
	PutObject (see page 17373)	This is PutObject, a member of class TStringArray.
	PutString (see page 17373)	This is PutString, a member of class TStringArray.

Public Methods

	Name	Description
	Add (see page 17370)	This is Add, a member of class TStringArray.
V	Assign (see page 17370)	Copies the contents of another, similar object.
V	Exchange (see page 17371)	Provides the interface for a method that swaps the values of two elements in the array.
	Find (see page 17371)	This is Find, a member of class TStringArray.
V	IndexOf (see page 17372)	Returns the position of the first element with a specified value.
V	Insert (see page 17372)	Inserts an element into the array and assigns it a value.
V	Sort (see page 17373)	Provides the interface for a method that sorts the elements in the array.

105.1.14.3.1 TStringArray.Add Method

Pascal

```
function Add(const S: String): Integer;
```

C++

```
__fastcall int Add(const AnsiString S);
```

Description

This is Add, a member of class TStringArray.

105.1.14.3.2 TStringArray.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the

contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo (see page 2931)

EConvertError (see page 22724)

105.1.14.3.3 TStringArray.Exchange Method

Provides the interface for a method that swaps the values of two elements in the array.

Pascal

```
procedure Exchange(Index1: Integer; Index2: Integer); override;
```

C++

```
virtual __fastcall Exchange(int Index1, int Index2);
```

Description

The Exchange method for TBaseArray does nothing other than provide an interface for a method that interchanges the values of two elements.

See Also

IndexOf (see page 17314)

GetItem (see page 17313)

105.1.14.3.4 TStringArray.Find Method

Pascal

```
function Find(S: string; var Index: Integer): Boolean;
```

C++

```
__fastcall Boolean Find(AnsiString S, int Index);
```

Description

This is Find, a member of class TStringArray.

105.1.14.3.5 TStringArray.GetObject Method

Pascal

```
function GetObject(Index: Integer): TObject;
```

C++

```
__fastcall TObject * GetObject(int Index);
```

Description

This is GetObject, a member of class TStringArray.

105.1.14.3.6 TStringArray.GetString Method

Pascal

```
function GetString(Index: Integer): string;
```

C++

```
__fastcall AnsiString GetString(int Index);
```

Description

This is GetString, a member of class TStringArray.

105.1.14.3.7 TStringArray.Grow Method

Pascal

```
procedure Grow; override;
```

C++

```
virtual __fastcall Grow();
```

Description

This is Grow, a member of class TStringArray.

105.1.14.3.8 TStringArray.IndexOf Method

Returns the position of the first element with a specified value.

Pascal

```
function IndexOf(var Item): Integer; override;
```

C++

```
virtual __fastcall int IndexOf( Item);
```

Description

Call IndexOf to obtain the position of the first occurrence of the value Value. IndexOf returns the 0-based index of the element. Thus, if Value matches the first element in the array, IndexOf returns 0, if Value matches the second element, IndexOf returns 1, and so on. If the value is not in the array, IndexOf returns -1.

Note: IndexOf does not work for arrays sorted in descending order.

See Also

FindItem (see page 17312)

GetItem (see page 17313)

105.1.14.3.9 TStringArray.Insert Method

Inserts an element into the array and assigns it a value.

Pascal

```
procedure Insert(Index: Integer; var Value); override;
```

C++

```
virtual __fastcall Insert(int Index,  Value);
```

Description

Call InsertAt to add an element to the array at the position specified by Index and assign it the value of Value. If Index is 0, the element is inserted at the beginning of the array. If Index is 1, the element is put in the second position of the array, and so on.

Note: Insert does not check the SortOrder property. If the array is sorted, be sure to insert the element in the proper sort order position.

See Also

Delete (see page 17312)

PutItem (see page 17315)

SortOrder (see page 17322)

105.1.14.3.10 TStringArray.InvalidateItems Method

Pascal

```
procedure InvalidateItems(atIndex: Integer; numItems: Integer); override;
```

C++

```
virtual __fastcall InvalidateItems(int atIndex, int numItems);
```

Description

This is InvalidateItems, a member of class TStringArray.

105.1.14.3.11 TStringArray.PutObject Method

Pascal

```
procedure PutObject(Index: Integer; AObject: TObject);
```

C++

```
__fastcall PutObject(int Index, TObject * AObject);
```

Description

This is PutObject, a member of class TStringArray.

105.1.14.3.12 TStringArray.PutString Method

Pascal

```
procedure PutString(Index: Integer; const S: string);
```

C++

```
__fastcall PutString(int Index, const AnsiString S);
```

Description

This is PutString, a member of class TStringArray.

105.1.14.3.13 TStringArray.Sort Method

Provides the interface for a method that sorts the elements in the array.

Pascal

```
procedure Sort(Compare: TCompareProc); override;
```

C++

```
virtual __fastcall Sort(TCompareProc Compare);
```

Description

The Sort method for TBaseArray does nothing. Descendants of TBaseArray override this method to sort the elements of the array using the ordering function provided in the Compare parameter.

See Also

CompareProc (see page 17319)

SortOrder (see page 17322)

105.1.14.4 TStringArray Properties

The properties of the TStringArray class are listed here.

Public Properties

	Name	Description
	Strings (see page 17374)	This is Strings, a member of class TStringArray.

105.1.14.4.1 TStringArray.Strings Property

Pascal

```
property Strings [Index: Integer]: String;
```

C++

```
__property AnsiString Strings[int Index];
```

Description

This is Strings, a member of class TStringArray.

105.1.15 TThreadCustomArray Class

Class Hierarchy

Pascal

```
TThreadCustomArray = class;
```

C++

```
class TThreadCustomArray;
```

File

Mxarrays

Description

This is class Mxarrays.TThreadCustomArray.

105.1.15.1 TThreadCustomArray Members

The following tables list the members exposed by TThreadCustomArray.

Public Methods

	Name	Description
	Add (see page 17376)	This is Add, a member of class TThreadCustomArray.
	Create (see page 17375)	This is Create, a member of class TThreadCustomArray.
V	Destroy (see page 17375)	This is Destroy, a member of class TThreadCustomArray.
	GetItem (see page 17376)	This is GetItem, a member of class TThreadCustomArray.
	LockArray (see page 17376)	This is LockArray, a member of class TThreadCustomArray.
	MemoryUsage (see page 17376)	This is MemoryUsage, a member of class TThreadCustomArray.
	UnlockArray (see page 17376)	This is UnlockArray, a member of class TThreadCustomArray.

105.1.15.2 TThreadCustomArray.Create Constructor

Pascal

```
constructor Create(Items: Integer; VarType: Integer);
```

C++

```
__fastcall TThreadCustomArray(int Items, int VarType);
```

Description

This is Create, a member of class TThreadCustomArray.

105.1.15.3 TThreadCustomArray.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TThreadCustomArray();
```

Description

This is Destroy, a member of class TThreadCustomArray.

105.1.15.4 TThreadCustomArray Methods

The methods of the TThreadCustomArray class are listed here.

Public Methods

	Name	Description
	Add (see page 17376)	This is Add, a member of class TThreadCustomArray.
	GetItem (see page 17376)	This is GetItem, a member of class TThreadCustomArray.
	LockArray (see page 17376)	This is LockArray, a member of class TThreadCustomArray.
	MemoryUsage (see page 17376)	This is MemoryUsage, a member of class TThreadCustomArray.
	UnlockArray (see page 17376)	This is UnlockArray, a member of class TThreadCustomArray.

105.1.15.4.1 TThreadCustomArray.Add Method

Pascal

```
function Add(Item: Variant): Integer;
```

C++

```
__fastcall int Add(Variant Item);
```

Description

This is Add, a member of class TThreadCustomArray.

105.1.15.4.2 TThreadCustomArray.GetItem Method

Pascal

```
function GetItem(Index: Integer): Variant;
```

C++

```
__fastcall Variant GetItem(int Index);
```

Description

This is GetItem, a member of class TThreadCustomArray.

105.1.15.4.3 TThreadCustomArray.LockArray Method

Pascal

```
function LockArray: TCustomArray;
```

C++

```
__fastcall TCustomArray LockArray();
```

Description

This is LockArray, a member of class TThreadCustomArray.

105.1.15.4.4 TThreadCustomArray.MemoryUsage Method

Pascal

```
function MemoryUsage: Integer;
```

C++

```
__fastcall int MemoryUsage();
```

Description

This is MemoryUsage, a member of class TThreadCustomArray.

105.1.15.4.5 TThreadCustomArray.UnlockArray Method

Pascal

```
procedure UnlockArray;
```

C++

```
__fastcall UnlockArray();
```

Description

This is UnlockArray, a member of class TThreadCustomArray.

105.1.16 TTwoDimArray Class

Class Hierarchy

Mxarrays.TTwoDimArray

Pascal

```
TTwoDimArray = class;
```

C++

```
class TTwoDimArray;
```

File

Mxarrays

Description

This is class Mxarrays.TTwoDimArray.

105.1.16.1 TTwoDimArray Members

The following tables list the members exposed by TTwoDimArray.

Protected Fields

	Name	Description
	mtxElements (see page 17378)	This is mtxElements, a member of class TTwoDimArray.

Public Methods

	Name	Description
	Create (see page 17377)	This is Create, a member of class TTwoDimArray.
V	Destroy (see page 17378)	This is Destroy, a member of class TTwoDimArray.
	SetSize (see page 17378)	This is SetSize, a member of class TTwoDimArray.

Public Properties

	Name	Description
R	columns (see page 17379)	This is columns, a member of class TTwoDimArray.
	Element (see page 17379)	This is Element, a member of class TTwoDimArray.
R	rows (see page 17379)	This is rows, a member of class TTwoDimArray.

105.1.16.2 TTwoDimArray.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TTwoDimArray();
```

Description

This is Create, a member of class TTwoDimArray.

105.1.16.3 TTwoDimArray.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTwoDimArray();
```

Description

This is Destroy, a member of class TTwoDimArray.

105.1.16.4 TTwoDimArray Fields

The fields of the TTwoDimArray class are listed here.

Protected Fields

	Name	Description
	mtxElements (see page 17378)	This is mtxElements, a member of class TTwoDimArray.

105.1.16.4.1 TTwoDimArray.mtxElements Field

Pascal

```
mtxElements: PMatrixElements;
```

C++

```
PMatrixElements mtxElements;
```

Description

This is mtxElements, a member of class TTwoDimArray.

105.1.16.5 TTwoDimArray Methods

The methods of the TTwoDimArray class are listed here.

Public Methods

	Name	Description
	SetSize (see page 17378)	This is SetSize, a member of class TTwoDimArray.

105.1.16.5.1 TTwoDimArray.SetSize Method

Pascal

```
procedure SetSize(NumRows: TDynArrayNDX; NumColumns: TMatrixNDX);
```

C++

```
__fastcall SetSize(TDynArrayNDX NumRows, TMatrixNDX NumColumns);
```

Description

This is SetSize, a member of class TTwoDimArray.

105.1.16.6 TTwoDimArray Properties

The properties of the TTwoDimArray class are listed here.

Public Properties

	Name	Description
R	columns (see page 17379)	This is columns, a member of class TTwoDimArray.
	Element (see page 17379)	This is Element, a member of class TTwoDimArray.
R	rows (see page 17379)	This is rows, a member of class TTwoDimArray.

105.1.16.6.1 TTwoDimArray.columns Property

Pascal

```
property columns: TMatrixNDX;
```

C++

```
__property TMatrixNDX columns;
```

Description

This is columns, a member of class TTwoDimArray.

105.1.16.6.2 TTwoDimArray.Element Property

Pascal

```
property Element [row: TDynArrayNDX; column: TMatrixNDX]: SmallInt;
```

C++

```
__property SmallInt Element[TDynArrayNDX; column: TMatrixNDX row];
```

Description

This is Element, a member of class TTwoDimArray.

105.1.16.6.3 TTwoDimArray.rows Property

Pascal

```
property rows: TDynArrayNDX;
```

C++

```
__property TDynArrayNDX rows;
```

Description

This is rows, a member of class TTwoDimArray.

105.1.17 TWordArray Class

Class Hierarchy

Pascal

```
TWordArray = class(TBaseArray);
```

C++

```
class TWordArray : public TBaseArray;
```

File

Mxarrays

Description

This is class Mxarrays.TWordArray.

105.1.17.1 TWordArray Members

The following tables list the members exposed by TWordArray.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BlockCopy (see page 17311)	This is BlockCopy, a member of class TBaseArray.
	ClearFlag (see page 17311)	This is ClearFlag, a member of class TBaseArray.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetAutoSize (see page 17313)	This is GetAutoSize, a member of class TBaseArray.
	GetOwner (see page 2932)	Returns the owner of an object.
	Grow (see page 17313)	This is Grow, a member of class TBaseArray.
	HasFlag (see page 17314)	This is HasFlag, a member of class TBaseArray.
	InternalHandleException (see page 17315)	This is InternalHandleException, a member of class TBaseArray.
	InvalidateItems (see page 17315)	This is InvalidateItems, a member of class TBaseArray.
	RemoveRange (see page 17316)	This is RemoveRange, a member of class TBaseArray.
	SetAutoSize (see page 17316)	This is SetAutoSize, a member of class TBaseArray.
	SetCapacity (see page 17317)	This is SetCapacity, a member of class TBaseArray.
	SetFlag (see page 17317)	This is SetFlag, a member of class TBaseArray.
	ValidateBounds (see page 17317)	This is ValidateBounds, a member of class TBaseArray.
	ValidIndex (see page 17318)	This is ValidIndex, a member of class TBaseArray.

Public Methods

	Name	Description
	Add (see page 17382)	This is Add, a member of class TWordArray.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 17383)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 17311)	Clear removes all items from the array.
V	Create (see page 17382)	Creates an instance of TBaseArray for elements of a specific size.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Delete (see page 17312)	Removes an element from the array.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 17310)	Destroys an instance of TBaseArray (see page 17307).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Exchange (see page 17312)	Provides the interface for a method that swaps the values of two elements in the array.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Find (see page 17383)	This is Find, a member of class TWordArray.
	FindItem (see page 17312)	Locates a value in a sorted array and indicates whether an element with that value already exists.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItem (see page 17384)	This is GetItem, a member of class TWordArray.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	IndexOf (see page 17384)	Returns the position of the first element with a specified value.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Insert (see page 17314)	Inserts an element into the array and assigns it a value.
	InsertAt (see page 17315)	Adds an element to the array at the position specified by Index.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PutItem (see page 17384)	This is PutItem, a member of class TWordArray.
	RemoveItem (see page 17316)	Removes an element from the array.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Sort (see page 17317)	Provides the interface for a method that sorts the elements in the array.

Public Properties

	Name	Description
	AutoSize (see page 17318)	Indicates whether the array grows dynamically as items are added.
	Capacity (see page 17319)	Specifies the maximum number of items that can be added to the array.
	CompareProc (see page 17319)	Implements a sort order for the elements of the array.
	Count (see page 17320)	Specifies the number of items in the array.
	Duplicates (see page 17320)	Specifies whether duplicate items can be added to sorted arrays.
	Items (see page 17385)	This is Items, a member of class TWordArray.
R	ItemSize (see page 17321)	Indicates the number of bytes used to store each item in the array.
	Limit (see page 17321)	Specifies the size of the array.
R	List (see page 17321)	Points to the array of values maintained by TBaseArray (see page 17307).
	SortOrder (see page 17322)	Determines whether the array is sorted.

105.1.17.2 TWordArray.Create Constructor

Creates an instance of TBaseArray for elements of a specific size.

Pascal

```
constructor Create(itemcount: Integer; dummy: Integer); override;
```

C++

```
virtual __fastcall TWordArray(int itemcount, int dummy);
```

Description

Call Create to create an array containing elements with the size specified by the iSize parameter.Create sets the Capacity property to Itemcount and the ItemSize property to iSize. It also initializes the AutoSize property to true and the Count property to 0.

See Also

AutoSize (see page 17318)

Capacity (see page 17319)

Count (see page 17320)

ItemSize (see page 17321)

Create (see page 22287)

105.1.17.3 TWordArray Methods

The methods of the TWordArray class are listed here.

Public Methods

	Name	Description
	Add (see page 17382)	This is Add, a member of class TWordArray.
V	Assign (see page 17383)	Copies the contents of another, similar object.
	Find (see page 17383)	This is Find, a member of class TWordArray.
	GetItem (see page 17384)	This is GetItem, a member of class TWordArray.
V	IndexOf (see page 17384)	Returns the position of the first element with a specified value.
	PutItem (see page 17384)	This is PutItem, a member of class TWordArray.

105.1.17.3.1 TWordArray.Add Method

Pascal

```
function Add(Value: Word): Integer;
```

C++

```
__fastcall int Add(Word Value);
```

Description

This is Add, a member of class TWordArray.

105.1.17.3.2 TWordArray.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo (see page 2931)

EConvertError (see page 22724)

105.1.17.3.3 TWordArray.Find Method

Pascal

```
function Find(var Index: Integer; Value: Word): Boolean;
```

C++

```
__fastcall Boolean Find(int Index, Word Value);
```

Description

This is Find, a member of class TWordArray.

105.1.17.3.4 TWordArray.GetItem Method

Pascal

```
function GetItem(index: Integer): Word;
```

C++

```
__fastcall Word GetItem(int index);
```

Description

This is GetItem, a member of class TWordArray.

105.1.17.3.5 TWordArray.IndexOf Method

Returns the position of the first element with a specified value.

Pascal

```
function IndexOf(var Item): Integer; override;
```

C++

```
virtual __fastcall int IndexOf( Item);
```

Description

Call IndexOf to obtain the position of the first occurrence of the value Value. IndexOf returns the 0-based index of the element. Thus, if Value matches the first element in the array, IndexOf returns 0, if Value matches the second element, IndexOf returns 1, and so on. If the value is not in the array, IndexOf returns -1.

Note: IndexOf does not work for arrays sorted in descending order.

See Also

FindItem (see page 17312)

GetItem (see page 17313)

105.1.17.3.6 TWordArray.PutItem Method

Pascal

```
procedure PutItem(index: Integer; value: Word);
```

C++

```
__fastcall PutItem(int index, Word value);
```

Description

This is PutItem, a member of class TWordArray.

105.1.17.4 TWordArray Properties

The properties of the TWordArray class are listed here.

Public Properties

	Name	Description
	Items (see page 17385)	This is Items, a member of class TWordArray.

105.1.17.4.1 TWordArray.Items Property

Pascal

```
property Items [Index: Integer]: Word;
```

C++

```
__property Word Items[int Index];
```

Description

This is Items, a member of class TWordArray.

105.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetMemoryCapacity (see page 17386)	This is function Mxarrays.GetMemoryCapacity.
	SetMemoryCapacity (see page 17386)	This is function Mxarrays.SetMemoryCapacity.

105.2.1 Mxarrays.GetMemoryCapacity Function

Pascal

```
function GetMemoryCapacity: Integer;
```

C++

```
int GetMemoryCapacity();
```

File

Mxarrays

Description

This is function Mxarrays.GetMemoryCapacity.

105.2.2 Mxarrays.SetMemoryCapacity Function

Pascal

```
procedure SetMemoryCapacity(Value: Integer);
```

C++

```
SetMemoryCapacity(int Value);
```

File

Mxarrays

Description

This is function Mxarrays.SetMemoryCapacity.

105.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TArrayFlags (see page 17387)	This is record Mxarrays.TArrayFlags.
	TDuplicates (see page 17387)	This is record Mxarrays.TDuplicates.
	TSortOrder (see page 17388)	This is record Mxarrays.TSortOrder.

Records

	Name	Description
	TStringItem (see page 17388)	This is record Mxarrays.TStringItem.

105.3.1 Mxarrays.TArrayFlags Enumeration

Pascal

```
TArrayFlags = (
  afOwnsData,
  afAutoSize,
  afCanCompare,
  afSortUnique
);
```

C++

```
enum TArrayFlags {
  afOwnsData,
  afAutoSize,
  afCanCompare,
  afSortUnique
};
```

File

Mxarrays

Description

This is record Mxarrays.TArrayFlags.

105.3.2 Mxarrays.TDuplicates Enumeration

Pascal

```
TDuplicates = (
  dupIgnore,
  dupAccept,
  dupError
);
```

C++

```
enum TDuplicates {
  dupIgnore,
  dupAccept,
  dupError
};
```

File

Mxarrays

Description

This is record Mxarrays.TDuplicates.

105.3.3 Mxarrays.TSortOrder Enumeration

Pascal

```
TSortOrder = (
  tsNone,
  tsAscending,
  tsDescending
);
```

C++

```
enum TSortOrder {
  tsNone,
  tsAscending,
  tsDescending
};
```

File

Mxarrays

Description

This is record Mxarrays.TSortOrder.

105.3.4 Mxarrays.TStringItem Record

Pascal

```
TStringItem = record
  FString: string;
  FObject: TObject;
end;
```

C++

```
struct TStringItem {
  AnsiString FString;
  TObject * FObject;
};
```

File

Mxarrays

Description

This is record Mxarrays.TStringItem.

105.4 Types

The following table lists types in this documentation.

Types

Name	Description
PIndexElements (see page 17390)	This is type Mxarrays.PIndexElements.
PMatrixElements (see page 17390)	This is type Mxarrays.PMatrixElements.
TArrayFlagSet (see page 17391)	This is type Mxarrays.TArrayFlagSet.
TCompareProc (see page 17391)	This is type Mxarrays.TCompareProc.
TDynArrayNDX (see page 17391)	This is type Mxarrays.TDynArrayNDX.
TIndexElements (see page 17392)	This is type Mxarrays.TIndexElements.
TIndexNDX (see page 17392)	This is type Mxarrays.TIndexNDX.
TMatrixElements (see page 17392)	This is type Mxarrays.TMatrixElements.
TMatrixNDX (see page 17393)	This is type Mxarrays.TMatrixNDX.

105.4.1 Mxarrays.PIndexElements Type

Pascal

```
PIndexElements = ^TIndexElements;
```

C++

```
^TIndexElements PIndexElements;
```

File

Mxarrays

Description

This is type Mxarrays.PIndexElements.

105.4.2 Mxarrays.PMatrixElements Type

Pascal

```
PMatrixElements = ^TMatrixElements;
```

C++

```
^TMatrixElements PMatrixElements;
```

File

Mxarrays

Description

This is type Mxarrays.PMatrixElements.

105.4.3 Mxarrays.TArrayFlagSet Type

Pascal

```
TArrayFlagSet = set of TArrayFlags;
```

C++

```
TArrayFlags TArrayFlagSet;
```

File

Mxarrays

Description

This is type Mxarrays.TArrayFlagSet.

105.4.4 Mxarrays.TCompareProc Type

Pascal

```
TCompareProc = function (var item1, item2): Integer;
```

C++

```
(var item1, item2): Integer ( TCompareProc)();
```

File

Mxarrays

Description

This is type Mxarrays.TCompareProc.

105.4.5 Mxarrays.TDynArrayNDX Type

Pascal

```
TDynArrayNDX = 0..vMaxRow;
```

C++

```
0..vMaxRow TDynArrayNDX;
```

File

Mxarrays

Description

This is type Mxarrays.TDynArrayNDX.

105.4.6 Mxarrays.TIndexElements Type

Pascal

```
TIndexElements = array[TIndexNDX] of TSmallIntArray;
```

C++

```
array[TIndexNDX] of TSmallIntArray TIndexElements;
```

File

Mxarrays

Description

This is type Mxarrays.TIndexElements.

105.4.7 Mxarrays.TIndexNDX Type

Pascal

```
TIndexNDX = 0..vMaxCol;
```

C++

```
0..vMaxCol TIndexNDX;
```

File

Mxarrays

Description

This is type Mxarrays.TIndexNDX.

105.4.8 Mxarrays.TMatrixElements Type

Pascal

```
TMatrixElements = array[TMatrixNDX] of TSmallIntArray;
```

C++

```
array[TMatrixNDX] of TSmallIntArray TMatrixElements;
```

File

Mxarrays

Description

This is type Mxarrays.TMatrixElements.

105.4.9 Mxarrays.TMatrixNDX Type

Pascal

```
TMatrixNDX = 0..vMaxCol;
```

C++

```
0..vMaxCol TMatrixNDX;
```

File

Mxarrays

Description

This is type Mxarrays.TMatrixNDX.

105.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
vMaxCol (see page 17394)	This is constant Mxarrays.vMaxCol.
vMaxRow (see page 17394)	This is constant Mxarrays.vMaxRow.

105.5.1 Mxarrays.vMaxCol Constant

Pascal

```
vMaxCol = High(Integer) div Sizeof(TSmallIntArray)-1;
```

C++

```
vMaxCol = High(Integer) div Sizeof(TSmallIntArray)-1;
```

File

Mxarrays

Description

This is constant Mxarrays.vMaxCol.

105.5.2 Mxarrays.vMaxRow Constant

Pascal

```
vMaxRow = (High(Integer)-$f) div Sizeof(SmallInt);
```

C++

```
vMaxRow = (High(Integer)-$f) div Sizeof(SmallInt);
```

File

Mxarrays

Description

This is constant Mxarrays.vMaxRow.

106 MXCOMMON Namespace

This is namespace MXCOMMON.

Topics

Name	Description
Classes (see page 17397)	The following table lists classes in this documentation.
Functions (see page 17427)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 17432)	The following table lists structs, records, enums in this documentation.
Types (see page 17436)	The following table lists types in this documentation.

Classes

	Name	Description
	EDimensionMapError (see page 17397)	This is class MXCOMMON.EDimensionMapError.
	TBinData (see page 17397)	This is class MXCOMMON.TBinData.
	TDimensionItem (see page 17405)	TDimensionItem is the base class for objects that represent dimensions and summaries in a crosstabulated data store.
	TDimensionItems (see page 17415)	TDimensionItems is the base class for objects that represent dimension maps.
	TTicks (see page 17424)	This is class MXCOMMON.TTicks.

Functions

	Name	Description
	AddToQuerySelect (see page 17427)	This is function MXCOMMON.AddToQuerySelect.
	BuildDataSetMap (see page 17427)	This is function MXCOMMON.BuildDataSetMap.
	CheckIfEmptyQuery (see page 17428)	This is function MXCOMMON.CheckIfEmptyQuery.
	FieldTypeToVarType (see page 17428)	This is function MXCOMMON.FieldTypeToVarType.
	FormatVariant (see page 17428)	This is function MXCOMMON.FormatVariant.
	GetAggName (see page 17429)	This is function MXCOMMON.GetAggName.
	HookToDataBase (see page 17429)	This is function MXCOMMON.HookToDataBase.
	IsAggValid (see page 17429)	This is function MXCOMMON.IsAggValid.
	IsBDEAvailable (see page 17430)	This is function MXCOMMON.IsBDEAvailable.
	NextArg (see page 17430)	This is function MXCOMMON.NextArg.
	UpdateDesigner (see page 17430)	This is function MXCOMMON.UpdateDesigner.
	VerifyRTQuery (see page 17431)	This is function MXCOMMON.VerifyRTQuery.

Structs, Records, Enums

	Name	Description
	TActiveFlags (see page 17432)	This is record MXCOMMON.TActiveFlags.
	TBinType (see page 17432)	This is record MXCOMMON.TBinType.
	TCDNotifyType (see page 17433)	This is record MXCOMMON.TCDNotifyType.
	TDimFlags (see page 17433)	This is record MXCOMMON.TDimFlags.
	TQueryError (see page 17434)	This is record MXCOMMON.TQueryError.
	TTickInfoRec (see page 17434)	This is record MXCOMMON.TTickInfoRec.

Types

Name	Description
PTickInfoRec (see page 17436)	This is type MXCOMMON.PTickInfoRec.
TBinTypes (see page 17436)	This is type MXCOMMON.TBinTypes.
TDimensionItemClass (see page 17436)	This is type MXCOMMON.TDimensionItemClass.

106.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EDimensionMapError (see page 17397)	This is class MXCOMMON.EDimensionMapError.
	TBinData (see page 17397)	This is class MXCOMMON.TBinData.
	TDimensionItem (see page 17405)	TDimensionItem is the base class for objects that represent dimensions and summaries in a crosstabulated data store.
	TDimensionItems (see page 17415)	TDimensionItems is the base class for objects that represent dimension maps.
	TTicks (see page 17424)	This is class MXCOMMON.TTicks.

106.1.1 EDimensionMapError Class

Class Hierarchy

Pascal

```
EDimensionMapError = class(Exception);
```

C++

```
class EDimensionMapError : public Exception;
```

File

MXCOMMON

Description

This is class MXCOMMON.EDimensionMapError.

106.1.2 TBinData Class

Class Hierarchy

Pascal

```
TBinData = class(TPersistent);
```

C++

```
class TBinData : public TPersistent;
```

File

MXCOMMON

Description

This is class MXCOMMON.TBinData.

106.1.2.1 TBinData Members

The following tables list the members exposed by TBinData.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Methods

	Name	Description
	AddBin (see page 17400)	This is AddBin, a member of class TBinData.
	AddBinValue (see page 17400)	This is AddBinValue, a member of class TBinData.
	AddBinValues (see page 17400)	This is AddBinValues, a member of class TBinData.
	AddIBinValue (see page 17401)	This is AddIBinValue, a member of class TBinData.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 17401)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BinValueCount (see page 17402)	This is BinValueCount, a member of class TBinData.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 17402)	This is Clear, a member of class TBinData.
	create (see page 17399)	This is create, a member of class TBinData.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	destroy (see page 17399)	Destroys the TPersistent instance and frees its memory.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAllBinValueCount (see page 17402)	This is GetAllBinValueCount, a member of class TBinData.
	GetBinCount (see page 17402)	This is GetBinCount, a member of class TBinData.
	GetBinName (see page 17402)	This is GetBinName, a member of class TBinData.
	GetBinNameDataType (see page 17403)	This is GetBinNameDataType, a member of class TBinData.
	GetBinValue (see page 17403)	This is GetBinValue, a member of class TBinData.
	GetBinValues (see page 17403)	This is GetBinValues, a member of class TBinData.
	GetIBinValue (see page 17403)	This is GetIBinValue, a member of class TBinData.
	GetIBinValueCount (see page 17404)	This is GetIBinValueCount, a member of class TBinData.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	GetMaxBinNameSize (see page 17404)	This is GetMaxBinNameSize, a member of class TBinData.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	BinCount (see page 17404)	This is BinCount, a member of class TBinData.
R	BinName (see page 17404)	This is BinName, a member of class TBinData.
R	NameList (see page 17405)	This is NameList, a member of class TBinData.
	OtherBinName (see page 17405)	This is OtherBinName, a member of class TBinData.
R	ValueList (see page 17405)	This is ValueList, a member of class TBinData.

106.1.2.2 TBinData.create Constructor

Pascal

```
constructor create;
```

C++

```
__fastcall TBinData();
```

Description

This is create, a member of class TBinData.

106.1.2.3 TBinData.destroy Destructor

Destroys the TPersistent instance and frees its memory.

Pascal

```
destructor destroy; override;
```

C++

```
virtual __fastcall ~TBinData();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

Free (see page 22300)

Destroy (see page 22287)

106.1.2.4 TBinData Methods

The methods of the TBinData class are listed here.

Public Methods

	Name	Description
	AddBin (see page 17400)	This is AddBin, a member of class TBinData.
	AddBinValue (see page 17400)	This is AddBinValue, a member of class TBinData.
	AddBinValues (see page 17400)	This is AddBinValues, a member of class TBinData.
	AddIBinValue (see page 17401)	This is AddIBinValue, a member of class TBinData.
V	Assign (see page 17401)	Copies the contents of another, similar object.
	BinValueCount (see page 17402)	This is BinValueCount, a member of class TBinData.
	Clear (see page 17402)	This is Clear, a member of class TBinData.
	GetAllBinValueCount (see page 17402)	This is GetAllBinValueCount, a member of class TBinData.
	GetBinCount (see page 17402)	This is GetBinCount, a member of class TBinData.
	GetBinName (see page 17402)	This is GetBinName, a member of class TBinData.
	GetBinNameDataType (see page 17403)	This is GetBinNameDataType, a member of class TBinData.
	GetBinValue (see page 17403)	This is GetBinValue, a member of class TBinData.
	GetBinValues (see page 17403)	This is GetBinValues, a member of class TBinData.
	GetIBinValue (see page 17403)	This is GetIBinValue, a member of class TBinData.
	GetIBinValueCount (see page 17404)	This is GetIBinValueCount, a member of class TBinData.
	GetMaxBinNameSize (see page 17404)	This is GetMaxBinNameSize, a member of class TBinData.

106.1.2.4.1 TBinData.AddBin Method

Pascal

```
function AddBin(BinName: string; vType: Integer): Integer;
```

C++

```
__fastcall int AddBin(AnsiString BinName, int vType);
```

Description

This is AddBin, a member of class TBinData.

106.1.2.4.2 TBinData.AddBinValue Method

Pascal

```
function AddBinValue(BinName: string; Value: Variant): Integer;
```

C++

```
__fastcall int AddBinValue(AnsiString BinName, Variant Value);
```

Description

This is AddBinValue, a member of class TBinData.

106.1.2.4.3 TBinData.AddBinValues Method

Pascal

```
procedure AddBinValues(BinName: string; const Values: array of const);
```

C++

```
__fastcall AddBinValues(AnsiString BinName, const array of const Values);
```

Description

This is AddBinValues, a member of class TBinData.

106.1.2.4.4 TBinData.AddIBinValue Method

Pascal

```
function AddIBinValue(iBin: Integer; value: variant): Integer;
```

C++

```
__fastcall int AddIBinValue(int iBin, variant value);
```

Description

This is AddIBinValue, a member of class TBinData.

106.1.2.4.5 TBinData.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Value: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Value);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo (see page 2931)

EConvertError (see page 22724)

106.1.2.4.6 TBinData.BinValueCount Method

Pascal

```
function BinValueCount(BinName: string): Integer;
```

C++

```
__fastcall int BinValueCount(AnsiString BinName);
```

Description

This is BinValueCount, a member of class TBinData.

106.1.2.4.7 TBinData.Clear Method

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

This is Clear, a member of class TBinData.

106.1.2.4.8 TBinData.GetAllBinValueCount Method

Pascal

```
function GetAllBinValueCount: Integer;
```

C++

```
__fastcall int GetAllBinValueCount();
```

Description

This is GetAllBinValueCount, a member of class TBinData.

106.1.2.4.9 TBinData.GetBinCount Method

Pascal

```
function GetBinCount: Integer;
```

C++

```
__fastcall int GetBinCount();
```

Description

This is GetBinCount, a member of class TBinData.

106.1.2.4.10 TBinData.GetBinName Method

Pascal

```
function GetBinName(Index: Integer): string;
```

C++

```
__fastcall AnsiString GetBinName(int Index);
```

Description

This is GetBinName, a member of class TBinData.

106.1.2.4.11 TBinData.GetBinNameDataType Method

Pascal

```
function GetBinNameDataType: TFieldType;
```

C++

```
__fastcall TFieldType GetBinNameDataType();
```

Description

This is GetBinNameDataType, a member of class TBinData.

106.1.2.4.12 TBinData.GetBinValue Method

Pascal

```
function GetBinValue(BinName: string; Index: Integer): Variant;
```

C++

```
__fastcall Variant GetBinValue(AnsiString BinName, int Index);
```

Description

This is GetBinValue, a member of class TBinData.

106.1.2.4.13 TBinData.GetBinValues Method

Pascal

```
function GetBinValues(BinName: string): Variant;
```

C++

```
__fastcall Variant GetBinValues(AnsiString BinName);
```

Description

This is GetBinValues, a member of class TBinData.

106.1.2.4.14 TBinData.GetIBinValue Method

Pascal

```
function GetIBinValue(iBin: Integer; ValueIndex: Integer): variant;
```

C++

```
__fastcall variant GetIBinValue(int iBin, int ValueIndex);
```

Description

This is GetIBinValue, a member of class TBinData.

106.1.2.4.15 TBinData.GetIBinValueCount Method

Pascal

```
function GetIBinValueCount(iBin: Integer): Integer;
```

C++

```
__fastcall int GetIBinValueCount(int iBin);
```

Description

This is GetIBinValueCount, a member of class TBinData.

106.1.2.4.16 TBinData.GetMaxBinNameSize Method

Pascal

```
function GetMaxBinNameSize: Integer;
```

C++

```
__fastcall int GetMaxBinNameSize();
```

Description

This is GetMaxBinNameSize, a member of class TBinData.

106.1.2.5 TBinData Properties

The properties of the TBinData class are listed here.

Public Properties

	Name	Description
R	BinCount (see page 17404)	This is BinCount, a member of class TBinData.
R	BinName (see page 17404)	This is BinName, a member of class TBinData.
R	NameList (see page 17405)	This is NameList, a member of class TBinData.
	OtherBinName (see page 17405)	This is OtherBinName, a member of class TBinData.
R	ValueList (see page 17405)	This is ValueList, a member of class TBinData.

106.1.2.5.1 TBinData.BinCount Property

Pascal

```
property BinCount: Integer;
```

C++

```
__property int BinCount;
```

Description

This is BinCount, a member of class TBinData.

106.1.2.5.2 TBinData.BinName Property

Pascal

```
property BinName [ Index:Integer ]: string;
```

C++

```
__property AnsiString BinName[int Index];
```

Description

This is BinName, a member of class TBinData.

106.1.2.5.3 TBinData.NameList Property

Pascal

```
property NameList: TStringList;
```

C++

```
__property TStringList NameList;
```

Description

This is NameList, a member of class TBinData.

106.1.2.5.4 TBinData.OtherBinName Property

Pascal

```
property OtherBinName: string;
```

C++

```
__property AnsiString OtherBinName;
```

Description

This is OtherBinName, a member of class TBinData.

106.1.2.5.5 TBinData.ValueList Property

Pascal

```
property ValueList: TList;
```

C++

```
__property TList ValueList;
```

Description

This is ValueList, a member of class TBinData.

106.1.3 TDimensionItem Class

TDimensionItem is the base class for objects that represent dimensions and summaries in a crosstabulated data store.

Class Hierarchy

Pascal

```
TDimensionItem = class(TCollectionItem);
```

C++

```
class TDimensionItem : public TCollectionItem;
```

File

MXCOMMON

Description

Use TDimensionItem as a base class when defining classes that represent the fields used by a crosstabulated data store. TDimensionItem describes a field from the data set used by a crosstabulated data store and the role that field plays in the data store. Crosstabulated data stores automatically create dimension items to represent the fields that make up their multidimensional arrays of summary values.

TDimensionItem has a single descendant, TCubeDim, that enhances the dimension item with the ability to represent date ranges.

See Also

TCubeDim (see page 17754)

TCustomDataStore (see page 17772)

TDimensionItems (see page 17415)

TDimensionItems (see page 17415)

106.1.3.1 TDimensionItem Members

The following tables list the members exposed by TDimensionItem.

Protected Methods

	Name	Description
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
V	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
V	InitializeRange (see page 17409)	This is InitializeRange, a member of class TDimensionItem.
V	NotifyCollection (see page 17410)	This is NotifyCollection, a member of class TDimensionItem.
V	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
V	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
V	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Protected Properties

	Name	Description
	Active (see page 17411)	This is Active, a member of class TDimensionItem.
R	Owner (see page 17414)	This is Owner, a member of class TDimensionItem.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 17408)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 17408)	Creates and initializes a TCollectionItem instance.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDimension (see page 17409)	Indicates whether the dimension item represents a dimension.
	IsSummary (see page 17410)	Indicates whether the dimension item represents a summary.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	ActiveFlag (see page 17411)	Indicates whether the dimension item is stored in the cache of a crosstabulated data store.
	BaseName (see page 17411)	Specifies the field name of the database field that is represented by or summarized in the dimension item.
	DerivedFrom (see page 17412)	Specifies the index of a dimension item from which a derived value is calculated.
	DimensionType (see page 17412)	Indicates the use of the dimension item.
	Fieldname (see page 17413)	This is Fieldname, a member of class TDimensionItem.
	FieldType (see page 17413)	Indicates the data type of the field that the dimension item represents.
	Format (see page 17414)	Indicates how field values are formatted on dimensions that represent numeric or temporal data.
	Name (see page 17414)	Denotes the name of a dimension or summary.

106.1.3.2 TDimensionItem.Create Constructor

Creates and initializes a TCollectionItem instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TDimensionItem(TCollection Collection);
```

Description

Do not call Create to instantiate a collection item. Instead, call the Add method of the collection to which the item should belong.

Collection identifies the TCollection instance to which the new item belongs.

See Also

Add (see page 2736)

Collection (see page 2761)

106.1.3.3 TDimensionItem Methods

The methods of the TDimensionItem class are listed here.

Protected Methods

	Name	Description
	InitializeRange (see page 17409)	This is InitializeRange, a member of class TDimensionItem.
	NotifyCollection (see page 17410)	This is NotifyCollection, a member of class TDimensionItem.

Public Methods

	Name	Description
	Assign (see page 17408)	Copies the contents of another, similar object.
	IsDimension (see page 17409)	Indicates whether the dimension item represents a dimension.
	IsSummary (see page 17410)	Indicates whether the dimension item represents a summary.

106.1.3.3.1 TDimensionItem.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Value: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Value);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the

inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo (see page 2931)

EConvertError (see page 22724)

106.1.3.3.2 TDimensionItem.InitializeRange Method

Pascal

```
procedure InitializeRange; virtual;
```

C++

```
virtual __fastcall InitializeRange();
```

Description

This is InitializeRange, a member of class TDimensionItem.

106.1.3.3.3 TDimensionItem.IsDimension Method

Indicates whether the dimension item represents a dimension.

Pascal

```
function IsDimension: Boolean;
```

C++

```
__fastcall Boolean IsDimension();
```

Description

Call IsDimension to determine the use of the dimension item. IsDimension returns true if the dimension item represents one of the dimensions of the multidimensional array maintained by a crosstabulated data store. IsDimension returns false if the dimension item represents a field that provides the summary values that fill the cells of the multidimensional array.

See Also

DimensionType (see page 17412)

IsSummary (see page 17410)

106.1.3.3.4 TDimensionItem.IsSummary Method

Indicates whether the dimension item represents a summary.

Pascal

```
function IsSummary: Boolean;
```

C++

```
__fastcall Boolean IsSummary();
```

Description

Call IsSummary to determine the use of the dimension item. IsSummary returns true if the dimension item represents a field that provides the summary values that fill the cells of the multidimensional array maintained by a crosstabulated data store. IsSummary returns false if the dimension item represents a one of the dimensions of the multidimensional array.

See Also

DimensionType (see page 17412)

IsDimension (see page 17409)

106.1.3.3.5 TDimensionItem.NotifyCollection Method

Pascal

```
procedure NotifyCollection(aType: TCDNotifyType); virtual;
```

C++

```
virtual __fastcall NotifyCollection(TCDNotifyType aType);
```

Description

This is NotifyCollection, a member of class TDimensionItem.

106.1.3.4 TDimensionItem Properties

The properties of the TDimensionItem class are listed here.

Protected Properties

	Name	Description
	Active (see page 17411)	This is Active, a member of class TDimensionItem.
R	Owner (see page 17414)	This is Owner, a member of class TDimensionItem.

Published Properties

	Name	Description
	ActiveFlag (see page 17411)	Indicates whether the dimension item is stored in the cache of a crosstabulated data store.
	BaseName (see page 17411)	Specifies the field name of the database field that is represented by or summarized in the dimension item.
	DerivedFrom (see page 17412)	Specifies the index of a dimension item from which a derived value is calculated.
	DimensionType (see page 17412)	Indicates the use of the dimension item.
	Fieldname (see page 17413)	This is Fieldname, a member of class TDimensionItem.
	FieldType (see page 17413)	Indicates the data type of the field that the dimension item represents.
	Format (see page 17414)	Indicates how field values are formatted on dimensions that represent numeric or temporal data.
	Name (see page 17414)	Denotes the name of a dimension or summary.

106.1.3.4.1 TDimensionItem.Active Property

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

This is Active, a member of class TDimensionItem.

106.1.3.4.2 TDimensionItem.ActiveFlag Property

Indicates whether the dimension item is stored in the cache of a crosstabulated data store.

Pascal

```
property ActiveFlag: TActiveFlags;
```

C++

```
__property TActiveFlags ActiveFlag;
```

Description

Set ActiveFlag to diActive when it is critical that the dimension item is available at all times. Set ActiveFlag to diAsNeeded instead if the dimension item is used less frequently. The decision cube swaps dimension items with Active (see page 17411) set to diAsNeeded in and out of the cache depending on when they are actually used, in order to conserve memory. Set ActiveFlag to diInactive to collapse the multidimensional array maintained by the data store so that it does not include the dimension item. When ActiveFlag is diInactive, the data store can not provide information about the dimension item to a decision source object.

The memory needs of a crosstabulated data store increase geometrically with each active dimension that is included. Memory needs increase linearly with each active summary that is included. Applications must find the best balance between the performance benefit of caching many dimensions and summaries so that they are immediately available to decision controls, and the need to keep the size of the cached multidimensional data array small enough so that it does not consume inordinate amounts of memory.

See Also

GetMemoryUsage (see page 17783)

MaxDimensions (see page 17793)

MaxSummaries (see page 17793)

106.1.3.4.3 TDimensionItem.BaseName Property

Specifies the field name of the database field that is represented by or summarized in the dimension item.

Pascal

```
property BaseName: string;
```

C++

```
__property AnsiString BaseName;
```

Description

Use BaseName to identify the field whose values underlie the dimension item. If the dimension item represents a dimension then

BaseName is the name of the field that corresponds to the dimension. Otherwise, BaseName is the field name of the field whose values are summarized by the dimension item.

Note: Basename is a fully qualified SQL selector. Table and field names must be correctly punctuated, including quotations.

See Also

DimensionType (see page 17412)

FieldName (see page 17413)

106.1.3.4.4 TDimensionItem.DerivedFrom Property

Specifies the index of a dimension item from which a derived value is calculated.

Pascal

```
property DerivedFrom: Integer;
```

C++

```
__property int DerivedFrom;
```

Description

Read DerivedFrom to determine whether the dimension item represents a field from the dataset of its decision cube. Dimension items that represent averages do not come directly from the dataset, but are derived from count and summary fields that appear in the dataset. When a derived average dimension item is created, DerivedFrom is set to the index of the count or summary dimension item. The corresponding summary or count dimension item is located by matching the BaseName (see page 17411) property when the derived values are calculated.

See Also

BaseName (see page 17411)

DataSet (see page 17789)

106.1.3.4.5 TDimensionItem.DimensionType Property

Indicates the use of the dimension item.

Pascal

```
property DimensionType: TDimFlags;
```

C++

```
__property TDimFlags DimensionType;
```

Description

Read DimensionType to determine whether the dimension item represents a dimension of the multidimensional array maintained by the crosstabulated data store or whether the dimension item represents a summary field that supplies the values to the cells of the multidimensional array. If the dimension item represents a summary, DimensionType also indicates the type of summary.

Do not set DimensionType if the dimension item represents a field from a query. The value of DimensionType is set automatically to reflect the semantics of the query if the crosstabulated data store that uses the dimension item is bound to a query. If the data store is bound to another type of data set, set DimensionType to a value that indicates the use of the dimension item in the multidimensional array.

DimensionType can have any of the following values:

Value	Meaning
dimDimension	The dimension item represents a dimension of the multidimensional array.
dimSum	The dimension item is a summary that represents the sum of a field value.
dimCount	The dimension item is a summary that represents the count of nonblank field values.
dimAverage	The dimension item is a summary that represents the average field value.
dimMin	The dimension item is a summary that represents the minimum field value. (not currently implemented)
dimMax	The dimension item is a summary that represents the maximum field value. (not currently implemented)
dimGenericAgg	The dimension item is some other type of summary.
dimUnknown	The use of the dimension item is not known. This occurs when the DataSet property of the data store is not based on a query. The value must be changed before the dimension item can be used.

See Also

IsDimension (see page 17409)

IsSummary (see page 17410)

106.1.3.4.6 TDimensionItem.Fieldname Property

Pascal

```
property Fieldname: string;
```

C++

```
__property AnsiString Fieldname;
```

Description

This is Fieldname, a member of class TDimensionItem.

106.1.3.4.7 TDimensionItem.FieldType Property

Indicates the data type of the field that the dimension item represents.

Pascal

```
property FieldType: TFieldType;
```

C++

```
__property TFieldType FieldType;
```

Description

Read FieldType to determine the type of data that makes up the values of the dimension item. FieldType is the same as the DataType property of the corresponding TField object from the dataset of the crosstabulated data store.

Do not set the FieldType property. The value of FieldType is set automatically to reflect the type of data provided by the dimension item.

See Also

DataSet (see page 17789)

DataType (see page 7159)

106.1.3.4.8 TDimensionItem.Format Property

Indicates how field values are formatted on dimensions that represent numeric or temporal data.

Pascal

```
property Format: String;
```

C++

```
__property AnsiString Format;
```

Description

Set Format on dimension items that represent numeric or temporal fields. The format specifiers that make up the Format string depend on the field type.

See the FormatFloat runtime library function for a description of the format specifiers for a numeric field. See the FormatDateTime runtime library function for a description of the format specifiers for a temporal field.

Decision controls that display value names read the value of Format, but can override it when formatting field values.

See Also

FormatDateTime (see page 22902)

FormatFloat (see page 22904)

106.1.3.4.9 TDimensionItem.Name Property

Denotes the name of a dimension or summary.

Pascal

```
property Name: string;
```

C++

```
__property AnsiString Name;
```

Description

Set Name to change the name provided by the GetDimensionName or GetSummaryName method of the crosstabulated data store. Decision controls that display dimension names read the value of Name, but can override it.

See Also

FieldName (see page 17413)

GetDimensionName (see page 17781)

GetSummaryName (see page 17784)

106.1.3.4.10 TDimensionItem.Owner Property

Pascal

```
property Owner: TCollection;
```

C++

```
__property TCollection Owner;
```

Description

This is Owner, a member of class TDimensionItem.

106.1.4 TDimensionItems Class

TDimensionItems is the base class for objects that represent dimension maps.

Class Hierarchy

Pascal

```
TDimensionItems = class(TCollection);
```

C++

```
class TDimensionItems : public TCollection;
```

File

MXCOMMON

Description

Use TDimensionItems as a base class when defining classes that represent the dimension map of a crosstabulated data store. TDimensionItems represents a collection of TDimensionItem (see page 17405) objects, where each TDimensionItem (see page 17405) corresponds to a field from the data set of the crosstabulated data store.

TDimensionItems has a single descendant, TCubeDims, that adds additional functionality to handle dimension maps that include date ranges.

See Also

TCubeDims (see page 17766)

TCustomDataStore (see page 17772)

TDimensionItem (see page 17405)

106.1.4.1 TDimensionItems Members

The following tables list the members exposed by TDimensionItems.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 17420)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	NotifyOwner (see page 17420)	This is NotifyOwner, a member of class TDimensionItems.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.

V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 17418)	Adds a dimension item to the end of the Items (see page 17422) array.
	AddDerivedField (see page 17418)	Adds a derived average summary to the end of the Items (see page 17422) array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AlreadyExists (see page 17419)	Indicates whether a dimension item with specified properties already exists in the Items (see page 17422) property array.
V	Assign (see page 17419)	Copies the list of dimension items from another TDimensionItems (see page 17415) object.
	AverageFieldCheck (see page 17419)	Determines whether the necessary summary dimension items over a base dimension exist so that a dimension item of type dimAverage can be computed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 17417)	Creates an instance of TDimensionItems (see page 17415).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ActiveDimensionCount (see page 17421)	Indicates the number of dimension items that represent active dimensions in the crosstabulated data store.
R	ActiveSummaryCount (see page 17421)	Indicates the number of dimension items that represent active summaries in the crosstabulated data store.
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	DimensionCount (see page 17422)	Indicates the number of dimension items in the Items (see page 17422) array with DimensionType set to dimDimension.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 17422)	Lists all dimension items maintained by the TDimensionItems (see page 17415) object.
R	SummaryCount (see page 17423)	Indicates the number of dimension items in the Items (see page 17422) array with DimensionType set to a value other than dimDimension.

106.1.4.2 TDimensionItems.Create Constructor

Creates an instance of TDimensionItems (see page 17415).

Pascal

```
constructor Create(Owner: TPersistent; ItemClass: TDimensionItemClass);
```

C++

```
__fastcall TDimensionItems(TPersistent * Owner, TDimensionItemClass ItemClass);
```

Description

Applications should not need to call the constructor of TDimensionItems (see page 17415). TCustomDataStore objects create TDimensionItems (see page 17415) objects to implement their DimensionMap property.

See Also

Create (see page 2734)

DimensionMap (see page 17790)

Destroy (see page 2735)

106.1.4.3 TDimensionItems Methods

The methods of the TDimensionItems class are listed here.

Protected Methods

	Name	Description
V	GetOwner (see page 17420)	Returns the owner of an object.
	NotifyOwner (see page 17420)	This is NotifyOwner, a member of class TDimensionItems.

Public Methods

	Name	Description
	Add (see page 17418)	Adds a dimension item to the end of the Items (see page 17422) array.
	AddDerivedField (see page 17418)	Adds a derived average summary to the end of the Items (see page 17422) array.
	AlreadyExists (see page 17419)	Indicates whether a dimension item with specified properties already exists in the Items (see page 17422) property array.
	Assign (see page 17419)	Copies the list of dimension items from another TDimensionItems (see page 17415) object.
	AverageFieldCheck (see page 17419)	Determines whether the necessary summary dimension items over a base dimension exist so that a dimension item of type dimAverage can be computed.

106.1.4.3.1 TDimensionItems.Add Method

Adds a dimension item to the end of the Items (see page 17422) array.

Pascal

```
function Add: TDimensionItem;
```

C++

```
__fastcall TDimensionItem Add();
```

Description

Do not call the Add method to add dimension items to a TDimensionItems (see page 17415) object. The dimension items are added automatically to reflect the fields in the data set of the associated crosstabulated data store.

Add adds a new dimension item to the collection and returns the new dimension item.

See Also

Items (see page 17422)

DataSet (see page 17789)

106.1.4.3.2 TDimensionItems.AddDerivedField Method

Adds a derived average summary to the end of the Items (see page 17422) array.

Pascal

```
function AddDerivedField(Index: Integer; dimType: tDimFlags): Integer;
```

C++

```
__fastcall int AddDerivedField(int Index, tDimFlags dimType);
```

Description

Summaries that represent average values are not taken directly from the source dataset of a decision cube, but are derived from count summaries and summation summaries. AddDerivedField creates a dimension item for an average summary. The Index parameter specifies the index of an existing dimension item whose BaseName property specifies the field over which averages should be computed. AddDerivedField checks for the summary dimension items needed to derive the average. If both an average and a summation are available for the base field, AddDerivedField creates the average summary dimension item. AddDerivedField returns true if the dimension item is successfully created, false otherwise.

Note: The DimType (Delphi) or dimType (C++) parameter is currently ignored. Descendants of AddDerivedField can override this method to derive other types of dimension items. In this case, DimType/dimType should specify the type of derived dimension item that should be created. When overriding AddDerivedField, call the inherited method if DimType/dimType is dimAverage.

See Also

AverageFieldCheck (see page 17419)

DimensionType (see page 17412)

106.1.4.3.3 TDimensionItems.AlreadyExists Method

Indicates whether a dimension item with specified properties already exists in the Items (see page 17422) property array.

Pascal

```
function AlreadyExists(BaseName: String; sType: TDimFlags): Boolean;
```

C++

```
__fastcall Boolean AlreadyExists(AnsiString BaseName, TDimFlags sType);
```

Description

Call AlreadyExists to determine whether there is a dimension item in the Items (see page 17422) property array with the BaseName property equal to the BaseName parameter and the DimensionType property equal to the sType parameter.

See Also

Items (see page 17422)

BaseName (see page 17411)

DimensionType (see page 17412)

106.1.4.3.4 TDimensionItems.Assign Method

Copies the list of dimension items from another TDimensionItems (see page 17415) object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Assign calls the inherited method to copy the properties of the Source parameter. The inherited method will copy properties from any object that implements an AssignTo method that can copy to a TDimensionItems (see page 17415) object.

See Also

AssignTo (see page 2931)

106.1.4.3.5 TDimensionItems.AverageFieldCheck Method

Determines whether the necessary summary dimension items over a base dimension exist so that a dimension item of type dimAverage can be computed.

Pascal

```
function AverageFieldCheck(Index: Integer; var SumIndex: Integer; var CountIndex: Integer):
Boolean;
```

C++

```
__fastcall Boolean AverageFieldCheck(int Index, int SumIndex, int CountIndex);
```

Description

Call AverageFieldCheck to ensure that the necessary summary dimension items exist for creating an average summary. The Index parameter specifies an existing dimension item whose BaseName property specifies the field over which the average should be computed. The SumIndex and CountIndex parameters return the summation and count dimension items that can be used to compute average values.

See Also

AddDerivedField (see page 17418)

BaseName (see page 17411)

DimensionType (see page 17412)

106.1.4.3.6 TDimensionItems.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

106.1.4.3.7 TDimensionItems.NotifyOwner Method

Pascal

```
procedure NotifyOwner(aType: TCDNotifyType);
```

C++

```
__fastcall NotifyOwner(TCDNotifyType aType);
```

Description

This is NotifyOwner, a member of class TDimensionItems.

106.1.4.4 TDimensionItems Properties

The properties of the TDimensionItems class are listed here.

Public Properties

	Name	Description
R	ActiveDimensionCount (see page 17421)	Indicates the number of dimension items that represent active dimensions in the crosstabulated data store.
R	ActiveSummaryCount (see page 17421)	Indicates the number of dimension items that represent active summaries in the crosstabulated data store.
R	DimensionCount (see page 17422)	Indicates the number of dimension items in the Items (see page 17422) array with DimensionType set to dimDimension.
	Items (see page 17422)	Lists all dimension items maintained by the TDimensionItems (see page 17415) object.
R	SummaryCount (see page 17423)	Indicates the number of dimension items in the Items (see page 17422) array with DimensionType set to a value other than dimDimension.

106.1.4.4.1 TDimensionItems.ActiveDimensionCount Property

Indicates the number of dimension items that represent active dimensions in the crosstabulated data store.

Pascal

```
property ActiveDimensionCount: Integer;
```

C++

```
__property int ActiveDimensionCount;
```

Description

Read ActiveDimensionCount to determine the number of dimensions in the multidimensional data array cached by the data store. The memory requirements of the cached array go up exponentially with each active dimension, so ActiveDimensionCount provides a rough sense of the size of the cached array.

ActiveDimensionCount is the number of dimension items in the Items (see page 17422) array that have their DimensionType property set to dimDimension and their Active property set to true. Change ActiveDimensionCount indirectly by changing the Active property on dimension items with DimensionType values of dimDimension.

The DimensionCount (see page 17422) property indicates the total number of dimension items with DimensionType set to dimDimension. It provides an upper limit to the ActiveDimensionCount property.

See Also

DimensionCount (see page 17422)

Items (see page 17422)

ActiveFlag (see page 17411)

DimensionType (see page 17412)

106.1.4.4.2 TDimensionItems.ActiveSummaryCount Property

Indicates the number of dimension items that represent active summaries in the crosstabulated data store.

Pascal

```
property ActiveSummaryCount: Integer;
```

C++

```
__property int ActiveSummaryCount;
```

Description

Read ActiveSummaryCount to determine the number of summaries in the cells of the multidimensional data array cached by the data store. The memory requirements of the cached array go up linearly with each active summary. ActiveSummaryCount is the number of different summaries that can be made available to decision controls.

ActiveSummaryCount is the number of dimension items in the Items (see page 17422) array that don't have their DimensionType property set to dimDimension and do have their Active property set to true. Change ActiveSummaryCount indirectly by changing the Active property on dimension items with DimensionType values other than dimDimension.

The SummaryCount (see page 17423) property indicates the total number of dimension items that do not have DimensionType set to dimDimension. It provides an upper limit to the ActiveSummaryCount property.

See Also

Items (see page 17422)

SummaryCount (see page 17423)

ActiveFlag (see page 17411)

DimensionType (see page 17412)

106.1.4.4.3 TDimensionItems.DimensionCount Property

Indicates the number of dimension items in the Items (see page 17422) array with DimensionType set to dimDimension.

Pascal

```
property DimensionCount: Integer;
```

C++

```
__property int DimensionCount;
```

Description

Read DimensionCount to determine the number of potential dimensions for the multidimensional data array cached by the data store. DimensionCount reflects the number of unsummarized fields in the data set of the crosstabulated data store. While the data store can make any of these dimensions available, usually only a subset are active at any time, in order to limit memory consumption. The ActiveDimensionCount (see page 17421) property gives the number of dimensions that are currently active.

See Also

ActiveDimensionCount (see page 17421)

DimensionType (see page 17412)

106.1.4.4.4 TDimensionItems.Items Property

Lists all dimension items maintained by the TDimensionItems (see page 17415) object.

Pascal

```
property Items [Index: Integer]: TDimensionItem;
```

C++

```
__property TDimensionItem Items[int Index];
```

Description

Use Items to access individual dimension items maintained by the TDimensionItems (see page 17415) object. The first dimension item has an Index of 0, the second dimension item has an Index of 1, and so on. For example, the following C++ code accesses an item to indicate that it can be swapped out of memory:

```
DecisionCube1->DimensionMap->Items[0]->ActiveFlag = diAsNeeded;
```

Each dimension item in the Items array represents a field from the data set of a crosstabulated data store. Each dimension item represents either a dimension (a field over which data is summarized), or a summary (a value summarized over the dimensions).

In Delphi Code, Items is the default property of TDimensionItems (see page 17415). This means that the name of the Items property can be eliminated from code that uses the Items property. Thus, the line

```
DecisionCube1.DimensionMap.Items[0].ActiveFlag := diAsNeeded;
```

could be written

```
DecisionCube1.DimensionMap[0].ActiveFlag := diAsNeeded;
```

Note: Do not change the Items property. The set of items in the Items array is set automatically to represent the fields from the data set. The properties of individual dimension items within the Items array may be changed to specify whether the dimension items should be cached by the data store, to change the name used when displaying the dimension items, and so on.

See Also

DataSet (see page 17789)

DimensionMap (see page 17790)

operator_[]

106.1.4.4.5 TDimensionItems.SummaryCount Property

Indicates the number of dimension items in the Items (see page 17422) array with DimensionType set to a value other than dimDimension.

Pascal

```
property SummaryCount: Integer;
```

C++

```
__property int SummaryCount;
```

Description

Read SummaryCount to determine the number of potential summaries that could be stored in the cells of the multidimensional data array cached by the data store. SummaryCount reflects the number of summarized fields in the data set of the crosstabulated data store. If memory is a serious concern, the data store can make any of these summaries inactive. The ActiveSummaryCount (see page 17421) property gives the number of summaries that are currently active.

See Also

ActiveSummaryCount (see page 17421)

DimensionType (see page 17412)

106.1.5 TTicks Class

Class Hierarchy

MXCOMMON.TTicks

Pascal

```
TTicks = class;
```

C++

```
class TTicks;
```

File

MXCOMMON

Description

This is class MXCOMMON.TTicks.

106.1.5.1 TTicks Members

The following tables list the members exposed by TTicks.

Public Methods

	Name	Description
	Clear (see page 17425)	This is Clear, a member of class TTicks.
	Create (see page 17424)	This is Create, a member of class TTicks.
	Destroy (see page 17425)	This is Destroy, a member of class TTicks.
	Dump (see page 17425)	This is Dump, a member of class TTicks.
	Ticks (see page 17425)	This is Ticks, a member of class TTicks.
	TicksIntArray (see page 17426)	This is TicksIntArray, a member of class TTicks.
	TicksSmallIntArray (see page 17426)	This is TicksSmallIntArray, a member of class TTicks.

Public Properties

	Name	Description
	NumberOfValues (see page 17426)	This is NumberOfValues, a member of class TTicks.

106.1.5.2 TTicks.Create Constructor

Pascal

```
constructor Create(FileName: string);
```

C++

```
__fastcall TTicks(AnsiString FileName);
```

Description

This is Create, a member of class TTicks.

106.1.5.3 TTicks.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTicks();
```

Description

This is Destroy, a member of class TTicks.

106.1.5.4 TTicks Methods

The methods of the TTicks class are listed here.

Public Methods

	Name	Description
	Clear (see page 17425)	This is Clear, a member of class TTicks.
	Dump (see page 17425)	This is Dump, a member of class TTicks.
	Ticks (see page 17425)	This is Ticks, a member of class TTicks.
	TicksIntArray (see page 17426)	This is TicksIntArray, a member of class TTicks.
	TicksSmallIntArray (see page 17426)	This is TicksSmallIntArray, a member of class TTicks.

106.1.5.4.1 TTicks.Clear Method

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

This is Clear, a member of class TTicks.

106.1.5.4.2 TTicks.Dump Method

Pascal

```
procedure Dump(bDetail: Boolean);
```

C++

```
__fastcall Dump(Boolean bDetail);
```

Description

This is Dump, a member of class TTicks.

106.1.5.4.3 TTicks.Ticks Method

Pascal

```
procedure Ticks(SectionName: string);
```

C++

```
__fastcall Ticks(AnsiString SectionName);
```

Description

This is Ticks, a member of class TTicks.

106.1.5.4.4 TTicks.TicksIntArray Method

Pascal

```
procedure TicksIntArray(SectionName: string; IntArray: TIntArray);
```

C++

```
__fastcall TicksIntArray(AnsiString SectionName, TIntArray IntArray);
```

Description

This is TicksIntArray, a member of class TTicks.

106.1.5.4.5 TTicks.TicksSmallIntArray Method

Pascal

```
procedure TicksSmallIntArray(SectionName: string; IntArray: TSmallIntArray);
```

C++

```
__fastcall TicksSmallIntArray(AnsiString SectionName, TSmallIntArray IntArray);
```

Description

This is TicksSmallIntArray, a member of class TTicks.

106.1.5.5 TTicks Properties

The properties of the TTicks class are listed here.

Public Properties

	Name	Description
	NumberOfValues (see page 17426)	This is NumberOfValues, a member of class TTicks.

106.1.5.5.1 TTicks.NumberOfValues Property

Pascal

```
property NumberOfValues: Integer;
```

C++

```
__property int NumberOfValues;
```

Description

This is NumberOfValues, a member of class TTicks.

106.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AddToQuerySelect (see page 17427)	This is function MXCOMMON.AddToQuerySelect.
	BuildDataSetMap (see page 17427)	This is function MXCOMMON.BuildDataSetMap.
	CheckIfEmptyQuery (see page 17428)	This is function MXCOMMON.CheckIfEmptyQuery.
	FieldTypeToVarType (see page 17428)	This is function MXCOMMON.FieldTypeToVarType.
	FormatVariant (see page 17428)	This is function MXCOMMON.FormatVariant.
	GetAggName (see page 17429)	This is function MXCOMMON.GetAggName.
	HookToDataBase (see page 17429)	This is function MXCOMMON.HookToDataBase.
	IsAggValid (see page 17429)	This is function MXCOMMON.IsAggValid.
	IsBDEAvailable (see page 17430)	This is function MXCOMMON.IsBDEAvailable.
	NextArg (see page 17430)	This is function MXCOMMON.NextArg.
	UpdateDesigner (see page 17430)	This is function MXCOMMON.UpdateDesigner.
	VerifyRTQuery (see page 17431)	This is function MXCOMMON.VerifyRTQuery.

106.2.1 MXCOMMON.AddToQuerySelect Function

Pascal

```
procedure AddToQuerySelect(var SQLString: string; Select: string);
```

C++

```
AddToQuerySelect(AnsiString SQLString, AnsiString Select);
```

File

MXCOMMON

Description

This is function MXCOMMON.AddToQuerySelect.

106.2.2 MXCOMMON.BuildDataSetMap Function

Pascal

```
function BuildDataSetMap(aDataSet: TDataSet; Map: TDimensionItems; var bParsed: Boolean; var
bDataSetMatch: Boolean): TQueryError;
```

C++

```
TQueryError BuildDataSetMap(TDataSet aDataSet, TDimensionItems Map, Boolean bParsed, Boolean
bDataSetMatch);
```

File

MXCOMMON

Description

This is function MXCOMMON.BuildDataSetMap.

106.2.3 MXCOMMON.CheckIfEmptyQuery Function

Pascal

```
function CheckIfEmptyQuery(var SQLString: string): Boolean;
```

C++

```
Boolean CheckIfEmptyQuery(AnsiString SQLString);
```

File

MXCOMMON

Description

This is function MXCOMMON.CheckIfEmptyQuery.

106.2.4 MXCOMMON.FieldTypeToVarType Function

Pascal

```
function FieldTypeToVarType(DataType: TFieldType): Integer;
```

C++

```
int FieldTypeToVarType(TFieldType DataType);
```

File

MXCOMMON

Description

This is function MXCOMMON.FieldTypeToVarType.

106.2.5 MXCOMMON.FormatVariant Function

Pascal

```
function FormatVariant(Value: Variant; FFormat: String): string;
```

C++

```
AnsiString FormatVariant(Variant Value, AnsiString FFormat);
```

File

MXCOMMON

Description

This is function MXCOMMON.FormatVariant.

106.2.6 MXCOMMON.GetAggName Function

Pascal

```
function GetAggName(aType: TDimFlags; FieldName: string): string;
```

C++

```
AnsiString GetAggName(TDimFlags aType, AnsiString FieldName);
```

File

MXCOMMON

Description

This is function MXCOMMON.GetAggName.

106.2.7 MXCOMMON.HookToDataBase Function

Pascal

```
function HookToDataBase(aDataSet: TDataSet): TDataBase;
```

C++

```
TDataBase HookToDataBase(TDataSet aDataSet);
```

File

MXCOMMON

Description

This is function MXCOMMON.HookToDataBase.

106.2.8 MXCOMMON.IsAggValid Function

Pascal

```
function IsAggValid(AggType: TDimFlags; FieldType: TFieldType): Boolean;
```

C++

```
Boolean IsAggValid(TDimFlags AggType, TFieldType FieldType);
```

File

MXCOMMON

Description

This is function MXCOMMON.IsAggValid.

106.2.9 MXCOMMON.IsBDEAvailable Function

Pascal

```
function IsBDEAvailable: Boolean;
```

C++

```
Boolean IsBDEAvailable();
```

File

MXCOMMON

Description

This is function MXCOMMON.IsBDEAvailable.

106.2.10 MXCOMMON.NextArg Function

Pascal

```
function NextArg(var aPos: Integer; Source: string): string;
```

C++

```
AnsiString NextArg(int aPos, AnsiString Source);
```

File

MXCOMMON

Description

This is function MXCOMMON.NextArg.

106.2.11 MXCOMMON.UpdateDesigner Function

Pascal

```
procedure UpdateDesigner(Sender: TComponent);
```

C++

```
UpdateDesigner(TComponent * Sender);
```

File

MXCOMMON

Description

This is function MXCOMMON.UpdateDesigner.

106.2.12 MXCOMMON.VerifyRTQuery Function

Pascal

```
function VerifyRTQuery(aDataSet: TDataSet; Map: TDimensionItems; var bDataSetMatch: Boolean):
TQueryError;
```

C++

```
TQueryError VerifyRTQuery(TDataSet aDataSet, TDimensionItems Map, Boolean bDataSetMatch);
```

File

MXCOMMON

Description

This is function MXCOMMON.VerifyRTQuery.

106.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TActiveFlags (see page 17432)	This is record MXCOMMON.TActiveFlags.
	TBinType (see page 17432)	This is record MXCOMMON.TBinType.
	TCDNotifyType (see page 17433)	This is record MXCOMMON.TCDNotifyType.
	TDimFlags (see page 17433)	This is record MXCOMMON.TDimFlags.
	TQueryError (see page 17434)	This is record MXCOMMON.TQueryError.

Records

	Name	Description
	TTickInfoRec (see page 17434)	This is record MXCOMMON.TTickInfoRec.

106.3.1 MXCOMMON.TActiveFlags Enumeration

Pascal

```
TActiveFlags = (
  diActive,
  diAsNeeded,
  diInactive
);
```

C++

```
enum TActiveFlags {
  diActive,
  diAsNeeded,
  diInactive
};
```

File

MXCOMMON

Description

This is record MXCOMMON.TActiveFlags.

106.3.2 MXCOMMON.TBinType Enumeration

Pascal

```
TBinType = (
  binNone,
  binYear,
  binQuarter,
  binMonth,
  binSet,
```

```
    binCustom
  );
```

C++

```
  enum TBinType {
    binNone,
    binYear,
    binQuarter,
    binMonth,
    binSet,
    binCustom
  };
```

File

MXCOMMON

Description

This is record MXCOMMON.TBinType.

106.3.3 MXCOMMON.TCDNotifyType Enumeration

Pascal

```
  TCDNotifyType = (
    tdChanged
  );
```

C++

```
  enum TCDNotifyType {
    tdChanged
  };
```

File

MXCOMMON

Description

This is record MXCOMMON.TCDNotifyType.

106.3.4 MXCOMMON.TDimFlags Enumeration

Pascal

```
  TDimFlags = (
    dimDimension,
    dimSum,
    dimCount,
    dimAverage,
    dimMin,
    dimMax,
    dimGenericAgg,
    dimUnknown
  );
```

C++

```
  enum TDimFlags {
```

```
    dimDimension,
    dimSum,
    dimCount,
    dimAverage,
    dimMin,
    dimMax,
    dimGenericAgg,
    dimUnknown
  };
```

File

MXCOMMON

Description

This is record MXCOMMON.TDimFlags.

106.3.5 MXCOMMON.TQueryError Enumeration

Pascal

```
  TQueryError = (
    tqeOK,
    tqeNoDimensions,
    tqeNoAggs,
    tqeNotGrouped,
    tqeNotInitialized,
    tqeMapCorrupt,
    tqeUnKnownDims
  );
```

C++

```
  enum TQueryError {
    tqeOK,
    tqeNoDimensions,
    tqeNoAggs,
    tqeNotGrouped,
    tqeNotInitialized,
    tqeMapCorrupt,
    tqeUnKnownDims
  };
```

File

MXCOMMON

Description

This is record MXCOMMON.TQueryError.

106.3.6 MXCOMMON.TTickInfoRec Record

Pascal

```
  TTickInfoRec = record
    EndTicks: Integer;
    StartTicks: Integer;
    TotalTicks: Integer;
    SName: string;
```

```
end;
```

C++

```
struct TTickInfoRec {
  int EndTicks;
  int StartTicks;
  int TotalTicks;
  AnsiString SName;
};
```

File

MXCOMMON

Description

This is record MXCOMMON.TTickInfoRec.

106.4 Types

The following table lists types in this documentation.

Types

Name	Description
PTickInfoRec (see page 17436)	This is type MXCOMMON.PTickInfoRec.
TBinTypes (see page 17436)	This is type MXCOMMON.TBinTypes.
TDimensionItemClass (see page 17436)	This is type MXCOMMON.TDimensionItemClass.

106.4.1 MXCOMMON.PTickInfoRec Type

Pascal

```
PTickInfoRec = ^TTickInfoRec;
```

C++

```
^TTickInfoRec PTickInfoRec;
```

File

MXCOMMON

Description

This is type MXCOMMON.PTickInfoRec.

106.4.2 MXCOMMON.TBinTypes Type

Pascal

```
TBinTypes = set of TBinType;
```

C++

```
TBinType TBinTypes;
```

File

MXCOMMON

Description

This is type MXCOMMON.TBinTypes.

106.4.3 MXCOMMON.TDimensionItemClass Type

Pascal

```
TDimensionItemClass = class of TDimensionItem;
```

C++

```
TDimensionItem TDimensionItemClass;
```

File

MXCOMMON

Description

This is type MXCOMMON.TDimensionItemClass.

107 MXDB Namespace

This is namespace MXDB.

Topics

Name	Description
Classes (see page 17441)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 17510)	The following table lists structs, records, enums in this documentation.
Types (see page 17514)	The following table lists types in this documentation.

Classes

	Name	Description
	EArrayError (see page 17441)	This is class MXDB.EArrayError.
	EDimIndexError (see page 17441)	EDimIndexError is the exception class for when an invalid index is used to specify a dimension in a decision source.
	TDecisionCube (see page 17442)	TDecisionCube maintains crosstabulated data that it provides to a TDecisionSource (see page 17462) object.
	TDecisionDataLink (see page 17458)	This is class MXDB.TDecisionDataLink.
	TDecisionSource (see page 17462)	TDecisionSource links a decision control to a configuration (pivot state) of the data from a decision cube.
	TDimInfoArray (see page 17497)	This is class MXDB.TDimInfoArray.
	TIArray (see page 17501)	This is class MXDB.TIArray.
	TPivotState (see page 17504)	This is class MXDB.TPivotState.
	TQADecisionSource (see page 17508)	This is class MXDB.TQADecisionSource.

Structs, Records, Enums

	Name	Description
	TDecisionControlType (see page 17510)	This is record MXDB.TDecisionControlType.
	TDecisionDataEvent (see page 17510)	This is record MXDB.TDecisionDataEvent.
	TDimGroup (see page 17511)	This is record MXDB.TDimGroup.
	TDimInfo (see page 17511)	This is record MXDB.TDimInfo.
	TDimRange (see page 17512)	This is record MXDB.TDimRange.
	TDimState (see page 17512)	This is record MXDB.TDimState.
	TRowStates (see page 17513)	This is record MXDB.TRowStates.

Types

Name	Description
PArrayDimInfo (see page 17514)	This is type MXDB.PArrayDimInfo.

PArrayInt (see page 17514)	This is type MXDB.PArrayInt.
PDimInfo (see page 17515)	This is type MXDB.PDimInfo.
TArrayDimInfo (see page 17515)	This is type MXDB.TArrayDimInfo.
TArrayInt (see page 17515)	This is type MXDB.TArrayInt.
TBigStr (see page 17516)	This is type MXDB.TBigStr.
TRowState (see page 17516)	This is type MXDB.TRowState.

107.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EArrayError (see page 17441)	This is class MXDB.EArrayError.
	EDimIndexError (see page 17441)	EDimIndexError is the exception class for when an invalid index is used to specify a dimension in a decision source.
	TDecisionCube (see page 17442)	TDecisionCube maintains crosstabulated data that it provides to a TDecisionSource (see page 17462) object.
	TDecisionDataLink (see page 17458)	This is class MXDB.TDecisionDataLink.
	TDecisionSource (see page 17462)	TDecisionSource links a decision control to a configuration (pivot state) of the data from a decision cube.
	TDimInfoArray (see page 17497)	This is class MXDB.TDimInfoArray.
	TIArray (see page 17501)	This is class MXDB.TIArray.
	TPivotState (see page 17504)	This is class MXDB.TPivotState.
	TQADecisionSource (see page 17508)	This is class MXDB.TQADecisionSource.

107.1.1 EArrayError Class

Class Hierarchy

Pascal

```
EArrayError = class(Exception);
```

C++

```
class EArrayError : public Exception;
```

File

MXDB

Description

This is class MXDB.EArrayError.

107.1.2 EDimIndexError Class

EDimIndexError is the exception class for when an invalid index is used to specify a dimension in a decision source.

Class Hierarchy

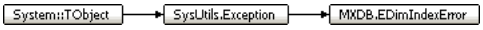

Pascal

```
EDimIndexError = class(Exception);
```

C++

```
class EDimIndexError : public Exception;
```

File

MXDB

Description

EDimIndexError is raised when any of the methods of a decision source are called with a dimension index that does not correspond to a dimension.

See Also

TDecisionSource (see page 17462)

107.1.3 TDecisionCube Class

TDecisionCube maintains crosstabulated data that it provides to a TDecisionSource (see page 17462) object.

Class Hierarchy

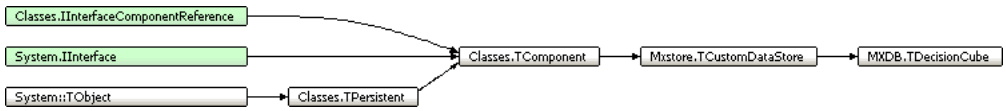

Pascal

```
TDecisionCube = class(TCustomDataStore);
```

C++

```
class TDecisionCube : public TCustomDataStore;
```

File

MXDB

Description

Add a TDecisionCube object to a form or data module to allow an application to represent crosstabulated data. TDecisionCube analyzes the data from a source dataset into a multidimensional structure where each dimension corresponds to a field in the dataset over which information is summarized. Any dataset can be used as the source of a TDecisionCube object, but it is easiest to work with TDecisionQuery.

Usually, the summarized values are calculated by the dataset that is the source of the decision cubes. However, average values can be calculated internally by the Decision Cube instead. Internally calculated averages have the advantage that they can be pivoted, subtotaled, and drilled-in correctly by Decision grids that display the data from the Decision Cube.

TDecisionCube introduces properties and methods that allow a crosstabulated data store to provide its data to a TDecisionSource (see page 17462) object. Decision grids and Decision graphs do not retrieve data directly from the decision cube. They use a TDecisionSource (see page 17462) object, which represents a configuration of the dimensions from the TDecisionCube object.

See Also

TDecisionGraph (see page 17521)

TDecisionGrid (see page 17586)

TDecisionPivot (see page 17680)

TDecisionQuery (see page 17854)

TDecisionSource (see page 17462)

Overview of Decision Support Components

Using Decision Cubes

Using Datasets with Decision Support Components

Decision Support Components and Memory Control

Using Decision Grids

107.1.3.1 TDecisionCube Members

The following tables list the members exposed by TDecisionCube.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActiveChanged (see page 17778)	This is ActiveChanged, a member of class TCustomDataStore.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BinMapHasBinData (see page 17778)	This is BinMapHasBinData, a member of class TCustomDataStore.
V	CanDimBeClosed (see page 17448)	This is CanDimBeClosed, a member of class TDecisionCube.
V	CanSumBeClosed (see page 17448)	This is CanSumBeClosed, a member of class TDecisionCube.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterClose (see page 17779)	This is DoAfterClose, a member of class TCustomDataStore.
V	DoAfterOpen (see page 17779)	This is DoAfterOpen, a member of class TCustomDataStore.
V	DoBeforeClose (see page 17779)	This is DoBeforeClose, a member of class TCustomDataStore.
V	DoBeforeOpen (see page 17780)	This is DoBeforeOpen, a member of class TCustomDataStore.
V	DoOnRefresh (see page 17780)	This is DoOnRefresh, a member of class TCustomDataStore.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetCurrentSummary (see page 17780)	This is GetCurrentSummary, a member of class TCustomDataStore.
	GetDataSet (see page 17780)	This is GetDataSet, a member of class TCustomDataStore.
	GetDomain (see page 17781)	This is GetDomain, a member of class TCustomDataStore.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.

V	LayoutChanged (see page 17785)	This is LayoutChanged, a member of class TCustomDataStore.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 17785)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetCurrentSummary (see page 17786)	This is SetCurrentSummary, a member of class TCustomDataStore.
	SetDataSet (see page 17786)	This is SetDataSet, a member of class TCustomDataStore.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	DataCache (see page 17789)	This is DataCache, a member of class TCustomDataStore.
R	DataLink (see page 17789)	This is DataLink, a member of class TCustomDataStore.
	DataSource (see page 17790)	This is DataSource, a member of class TCustomDataStore.
R	InternalDataSource (see page 17792)	This is InternalDataSource, a member of class TCustomDataStore.
R	State (see page 17794)	This is State, a member of class TCustomDataStore.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CalcSubTotals (see page 17778)	Calculates dimension subtotals.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 17447)	Creates an instance of TDecisionCube (see page 17442).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

	Destroy (see page 17447)	Destroys an instance of TDecisionCube (see page 17442).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetDetailSQL (see page 17448)	Returns an SQL statement that can be used to generate the described subset of the data in the decision cube.
	GetDimensionMemberCount (see page 17780)	Returns the number of distinct values for a dimension of the data store.
	GetDimensionName (see page 17781)	Returns the name of an active dimension.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMemberAsString (see page 17782)	Returns the specified value from a dimension of the data store, formatted as a string.
	GetMemberAsVariant (see page 17782)	Returns the specified value from a dimension of the crosstabulated data store.
	GetMemoryUsage (see page 17783)	Returns the number of bytes of memory used to cache the multidimensional array of summaries along all active dimensions.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetSQL (see page 17449)	Returns an SQL statement that can be used to generate the described subset of the data in the decision cube
	GetSummaryAsString (see page 17783)	Returns the value of the current summary for a set of values along all the active dimensions.
	GetSummaryAsVariant (see page 17784)	Returns the value of the current summary for a set of values along all the active dimensions.
	GetSummaryName (see page 17784)	Returns the name of an active summary.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 17785)	Updates the dimension map of the crosstabulated data store.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)

V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	ShowCubeDialog (see page 17450)	Displays the Decision Cube editor.
V	StateChanged (see page 17450)	Updates the dimension map when the pivot state changes.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Active (see page 17451)	Specifies whether the decision cube can provide data.
	BinData (see page 17787)	Indicates whether all the dimensions in the multidimensional array include a cell for every value along the dimension.
	Capacity (see page 17788)	Limits the memory that may be used by the cached array of summary values.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	CurrentSummary (see page 17788)	Specifies the index of the current summary from the set of active summaries maintained by the crosstabulated data store.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignState (see page 17452)	Specifies how much data is displayed by the decision cube at design time.
R	DimensionCount (see page 17790)	Specifies the number of dimensions cached by the crosstabulated data store.
R	DimensionMapCount (see page 17791)	Specifies the total number of fields from the dataset that make up the dimensions and summaries of the dimension map.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	SummaryCount (see page 17794)	Specifies the number of summary fields cached by the crosstabulated data store.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterClose (see page 17456)	Occurs immediately after the crosstabulated data store has been deactivated.
	AfterOpen (see page 17456)	Occurs immediately after the crosstabulated data store has been activated.
	BeforeClose (see page 17456)	Occurs immediately before the crosstabulated data store is deactivated.
	BeforeOpen (see page 17457)	Occurs immediately before the crosstabulated data store is activated.
	OnLowCapacity (see page 17457)	Occurs when the multidimensional cache requires more memory than is allowed by the Capacity property.
	OnRefresh (see page 17458)	Occurs immediately before the dimension map is changed.

Published Properties

	Name	Description
	DataSet (see page 17451)	Specifies the dataset that provides the fields which make up the dimensions and summaries maintained by the crosstabulated data store.
	DimensionMap (see page 17453)	Describes the dimensions and summaries maintained by the crosstabulated data store.
	MaxCells (see page 17453)	Limits the number of cells that can be included in the cache at any one time.
	MaxDimensions (see page 17454)	Limits the number of dimensions that can be cached by the crosstabulated data store at any one time.
	MaxSummaries (see page 17454)	Limits the number of summaries that can be cached by the crosstabulated data store at one time.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ShowProgressDialog (see page 17455)	Determines whether a progress bar appears when the cache of crosstabulated values is built.
	Tag (see page 2808)	Stores an integer value as part of a component.

107.1.3.2 TDecisionCube.Create Constructor

Creates an instance of TDecisionCube (see page 17442).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDecisionCube(TComponent * AOwner);
```

Description

Call Create to instantiate a decision cube at runtime. Decision cubes placed on a form or data module at design time are created automatically.

Create allocates the memory for the decision cube and its helper objects and initializes the Active (see page 17451) property to false and the DesignState (see page 17452) property to dsAllData.

See Also

DesignState (see page 17452)

Create (see page 17776)

Active (see page 17451)

107.1.3.3 TDecisionCube.Destroy Destructor

Destroys an instance of TDecisionCube (see page 17442).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDecisionCube();
```

Description

Do not call the Destroy method directly. Instead call Free. Free checks that the decision cube is not nil, and only then calls Destroy.

See Also

Create (see page 17447)

Free (see page 22300)

107.1.3.4 TDecisionCube Methods

The methods of the TDecisionCube class are listed here.

Protected Methods

	Name	Description
	CanDimBeClosed (see page 17448)	This is CanDimBeClosed, a member of class TDecisionCube.
	CanSumBeClosed (see page 17448)	This is CanSumBeClosed, a member of class TDecisionCube.

Public Methods

	Name	Description
	GetDetailSQL (see page 17448)	Returns an SQL statement that can be used to generate the described subset of the data in the decision cube.
	GetSQL (see page 17449)	Returns an SQL statement that can be used to generate the described subset of the data in the decision cube
	ShowCubeDialog (see page 17450)	Displays the Decision Cube editor.
V	StateChanged (see page 17450)	Updates the dimension map when the pivot state changes.

107.1.3.4.1 TDecisionCube.CanDimBeClosed Method

Pascal

```
function CanDimBeClosed(iMapIndex: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanDimBeClosed(int iMapIndex);
```

Description

This is CanDimBeClosed, a member of class TDecisionCube.

107.1.3.4.2 TDecisionCube.CanSumBeClosed Method

Pascal

```
function CanSumBeClosed(iMapIndex: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanSumBeClosed(int iMapIndex);
```

Description

This is CanSumBeClosed, a member of class TDecisionCube.

107.1.3.4.3 TDecisionCube.GetDetailSQL Method

Returns an SQL statement that can be used to generate the described subset of the data in the decision cube.

Pascal

```
function GetDetailSQL(ValueArray: TSmallIntArray; SelectList: string; bActive: Boolean):
string;
```

C++

```
__fastcall AnsiString GetDetailSQL(TSmallIntArray ValueArray, AnsiString SelectList, Boolean
bActive);
```

Description

Use the SQL statement returned by GetDetailSQL to generate a dataset that represents the records that gave rise to a subset of the values in the decision cube. The resulting query produces the records represented in the projection described by the ValueArray parameter. The dataset will include all fields that make up the dimensions of the decision cube, plus any fields listed in the SelectList parameter. The query does not include any summary values or grouping, such as must be imposed by the dataset of the decision cube.

Each entry in the ValueArray parameter corresponds to a dimension of the decision cube, where the first entry corresponds to the first dimension, the second entry corresponds to the second dimension, and so on. If the value of an entry in the value array is less than 0, all field values over that dimension are included in the new dataset. If the value of an entry in the value array is

equal to or greater than 0, it is interpreted as the index of a value over the dimension field. When the ValueArray specifies a field value, the SQL statement restricts the dataset to records that match that value on the corresponding field of the original database table. Thus, ValueArray represents the dimension array with a set of drilled dimensions.

bActive specifies whether the ValueArray parameter includes inactive dimensions. Only active dimensions are included when bActive is true. Also when bActive is true, only the fields that correspond to active dimensions are included automatically in the SQL query.

The SelectList parameter is a list of comma delimited field names for the fields that are included in the result set of the SQL query. The query always includes the fields that correspond to the dimensions represented by the ValueArray parameter. If SelectList consists of a single asterisk (*), all the fields in the original dataset are included.

See Also

DataSet (see page 17789)

GetSQL (see page 17449)

SQL (see page 10089)

107.1.3.4.4 TDecisionCube.GetSQL Method

Returns an SQL statement that can be used to generate the described subset of the data in the decision cube

Pascal

```
function GetSQL(ValueArray: TSmallIntArray; bActive: Boolean): string;
```

C++

```
__fastcall AnsiString GetSQL(TSmallIntArray ValueArray, Boolean bActive);
```

Description

Use the SQL statement returned by GetSQL to view the data that gave rise to a subset of the information in the decision cube. The resulting query produces a dataset containing the projection described by the ValueArray parameter. All the summaries of the current dataset are included in the resulting query.

Each entry in the ValueArray parameter corresponds to a dimension of the decision cube, where the first entry corresponds to the first dimension, the second entry corresponds to the second dimension, and so on. If the value of an entry in the value array is less than 0, all field values over that dimension are included in the new dataset. If the value of an entry in the value array is equal to or greater than 0, it is interpreted as the index of a value over the dimension field. When the ValueArray specifies a field value, the SQL statement restricts the dataset to records that match that value on the corresponding field of the original database table. Thus, ValueArray represents the dimension array with a set of drilled dimensions.

bActive specifies whether the ValueArray parameter includes inactive dimensions. Only active dimensions are included when bActive is true.

To further restrict the dataset so that it includes only a subset of the dimensions in the decision cube, use the GetDetailSQL (see page 17448) method.

See Also

DataSet (see page 17789)

GetDetailSQL (see page 17448)

SQL (see page 10089)

107.1.3.4.5 TDecisionCube.ShowCubeDialog Method

Displays the Decision Cube editor.

Pascal

```
procedure ShowCubeDialog;
```

C++

```
__fastcall ShowCubeDialog();
```

Description

Call ShowCubeDialog to allow the user to change the active dimensions of the data cube using the Decision Cube editor. ShowCubeDialog brings up the same dialog that appears when Decision Cube editor is selected from the pop-up menu of the decision cube component at design time.

The Decision Cube editor allows the user to make dimensions active or inactive and to change the names of dimensions. It also controls how much information is shown in a decision control at design time.

See Also

DimensionCount (see page 17790)

DimensionMap (see page 17790)

SummaryCount (see page 17794)

GetDimensionName (see page 17781)

107.1.3.4.6 TDecisionCube.StateChanged Method

Updates the dimension map when the pivot state changes.

Pascal

```
procedure StateChanged; override;
```

C++

```
virtual __fastcall StateChanged();
```

Description

StateChanged is called automatically when dimensions are opened or closed, the current summary changes, or dimensions are pivoted. In response, the decision cube updates the dimension map to reflect the current state of its dimensions and summaries.

See Also

DimensionMap (see page 17790)

107.1.3.5 TDecisionCube Properties

The properties of the TDecisionCube class are listed here.

Public Properties

	Name	Description
	Active (see page 17451)	Specifies whether the decision cube can provide data.
	DesignState (see page 17452)	Specifies how much data is displayed by the decision cube at design time.

Published Properties

	Name	Description
	DataSet (see page 17451)	Specifies the dataset that provides the fields which make up the dimensions and summaries maintained by the crosstabulated data store.
	DimensionMap (see page 17453)	Describes the dimensions and summaries maintained by the crosstabulated data store.
	MaxCells (see page 17453)	Limits the number of cells that can be included in the cache at any one time.
	MaxDimensions (see page 17454)	Limits the number of dimensions that can be cached by the crosstabulated data store at any one time.
	MaxSummaries (see page 17454)	Limits the number of summaries that can be cached by the crosstabulated data store at one time.
	ShowProgressDialog (see page 17455)	Determines whether a progress bar appears when the cache of crosstabulated values is built.

107.1.3.5.1 TDecisionCube.Active Property

Specifies whether the decision cube can provide data.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Read Active to determine whether the decision cube is ready to provide data to a decision control. Set Active to true to cause the decision cube to fetch and analyze data from the dataset specified in the DataSet (see page 17451) property.

When Active is false, the decision cube is closed and the decision source cannot read data from the decision cube. When Active is true, data can be read from the decision cube.

Note: The Active property cannot be set to true unless the dataset specified by the DataSet (see page 17451) property is already active.

See Also

DataSet (see page 17789)

AfterClose (see page 17795)

AfterOpen (see page 17795)

BeforeClose (see page 17796)

BeforeOpen (see page 17796)

107.1.3.5.2 TDecisionCube.DataSet Property

Specifies the dataset that provides the fields which make up the dimensions and summaries maintained by the crosstabulated data store.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Set DataSet to the dataset that contains the crosstabulated data for the data store. Usually, DataSet is a TDecisionQuery object. TDecisionQuery objects are SQL queries that are grouped appropriately to calculate summaries for a crosstabulated data store.

The DataSet property can also be set to a TQuery object where the SQL has the same form as that of a TDecisionQuery or to any other dataset that contains summarized values in the same structure as the result of a TDecisionQuery query.

If the DataSet property is a TQuery or TDecisionQuery object, the crosstabulated data store automatically determines which fields are dimensions and which are summaries. If the DataSet property is not a query, the fields in the dimension map must be explicitly identified as dimensions or summaries, and, if they represent summaries, the type of summary must be indicated.

Note: The summary values for the crosstabulated data store must be calculated by the TDataSet object. The data store does not summarize raw data, but analyzes summarized data into dimensions.

See Also

DimensionMap (see page 17790)

TDecisionQuery (see page 17854)

TQuery (see page 10059)

107.1.3.5.3 TDecisionCube.DesignState Property

Specifies how much data is displayed by the decision cube at design time.

Pascal

```
property DesignState: TCubeDataState;
```

C++

```
__property TCubeDataState DesignState;
```

Description

Set DesignState to limit the amount of information that is provided when the decision cube is active at design time. Building the cache of the decision cube can be time and memory intensive. Limiting the information the decision cube can supply at design time can improve design time performance when the decision cube has many dimensions or when the dimensions have many values.

DesignState has one of the following values:

Value	Meaning
dsNoData	The decision cube can't be activated at design time.
dsMetaData	The decision cube provides the dimension captions, but no listing of values along the dimensions, and no summarized data.
dsDimensionData	The decision cube provides dimension captions, and the values of the dimension fields, but no summarized data.
dsAllData	The decision cube provides all possible data: dimension captions, the values of dimension fields, and summarized data.

Note: The Value of DesignState can be changed at design time using the Decision Cube editor.

See Also

ShowCubeDialog (see page 17450)

Active (see page 17451)

107.1.3.5.4 TDecisionCube.DimensionMap Property

Describes the dimensions and summaries maintained by the crosstabulated data store.

Pascal

```
property DimensionMap: TCubeDims;
```

C++

```
__property TCubeDims DimensionMap;
```

Description

Read DimensionMap to access the dimensions and summaries of the data store. The individual entries of the dimension map represent the fields of the dataset specified by the DataSet property or calculated fields derived from those fields, such as averages. DimensionMap provides information about these fields from the dataset that make up the crosstabulated data store, including the field names and the use of the fields within the multidimensional array.

When the data store is not active, use the properties of the TCubeDim objects to change the information cached by the data store. Use the ActiveFlag property of individual TCubeDim objects to specify whether those dimensions or summaries from the dataset are cached by the crosstabulated data store. Use the BinType property of the TCubeDim objects to allow cells along temporal dimensions to represent date ranges or to put the dimensions in a paged (permanently drilled) state.

If the dataset is not a query, use the DimensionType property of the TCubeDim objects to identify which fields are dimensions and which are summaries, and to identify the type of summary represented by the summary fields.

To change the properties of the entries in the dimension map when the data store is active, use the Refresh method.

Note: At design time, the DimensionType, Name, and ActiveFlag properties of the TCubeDim objects that make up the dimension map can be set using the Decision Cube Editor.

See Also

DataSet (see page 17789)

Refresh (see page 17785)

BinType (see page 17763)

ActiveFlag (see page 17411)

DimensionType (see page 17412)

Name (see page 17414)

Decision Cube Properties and Events

Using the Decision Cube Editor

107.1.3.5.5 TDecisionCube.MaxCells Property

Limits the number of cells that can be included in the cache at any one time.

Pascal

```
property MaxCells: Integer;
```

C++

```
__property int MaxCells;
```

Description

Use MaxCells to limit the memory usage of the crosstabulated data store. Memory consumption increases geometrically with

each dimension that is cached in the multidimensional array and linearly with each summary. MaxCells prevents too many dimensions or summaries from being cached at once.

When the ActiveFlag property of a dimension or summary is diActive, it is always added to the cache. When the ActiveFlag property of a dimension or summary is diAsNeeded, it can be added only if it does not cause the number of cells to exceed MaxCells. The data store swaps in as-needed dimensions and summaries when they are used by the decision source, making room, if necessary, by swapping out closed dimensions or summaries that are not current.

Note: Be sure to set MaxCells to a value big enough to accommodate all active dimensions and summaries plus the number of as-needed dimensions and summaries that can be open at one time.

See Also

MaxDimensions (see page 17793)

MaxSummaries (see page 17793)

ActiveFlag (see page 17411)

107.1.3.5.6 TDecisionCube.MaxDimensions Property

Limits the number of dimensions that can be cached by the crosstabulated data store at any one time.

Pascal

```
property MaxDimensions: Integer;
```

C++

```
__property int MaxDimensions;
```

Description

Use MaxDimensions to limit the memory usage of the data store. Memory consumption increases geometrically with each dimension that is cached in the multidimensional array. MaxDimensions prevents too many dimensions to be cached at once.

When the ActiveFlag property of a dimension is diActive, the dimension is always added to the cache. When the ActiveFlag property of a dimension is diAsNeeded, it can be added only if it does not cause the number of active dimensions to exceed MaxDimensions. The data store swaps in as-needed dimensions when they are open in the decision source, making room, if necessary, by swapping out closed dimensions.

Note: Be sure to set MaxDimensions to a value big enough to accommodate all active dimensions plus the number of as-needed dimensions that can be open at one time.

See Also

DimensionCount (see page 17790)

MaxCells (see page 17792)

MaxSummaries (see page 17793)

ActiveFlag (see page 17411)

107.1.3.5.7 TDecisionCube.MaxSummaries Property

Limits the number of summaries that can be cached by the crosstabulated data store at one time.

Pascal

```
property MaxSummaries: Integer;
```

C++

```
__property int MaxSummaries;
```

Description

Use MaxSummaries to limit the memory usage of the data store. Memory consumption increases linearly with each summary that is cached in the multidimensional array. MaxSummaries prevents too many summaries to be cached at once.

When the ActiveFlag property of a summary is diActive, the summary is always added to the cache. When the ActiveFlag property is diAsNeeded, it can be added only if it does not cause the number of active summaries to exceed MaxSummaries. The data store swaps in as-needed summaries when they are chosen as the current summary, making room, if necessary, by swapping out other as-needed summaries.

See Also

MaxCells (see page 17792)

MaxDimensions (see page 17793)

SummaryCount (see page 17794)

ActiveFlag (see page 17411)

107.1.3.5.8 TDecisionCube.ShowProgressDialog Property

Determines whether a progress bar appears when the cache of crosstabulated values is built.

Pascal

```
property ShowProgressDialog: Boolean;
```

C++

```
__property Boolean ShowProgressDialog;
```

Description

Set ShowProgressDialog to specify whether a progress bar should appear to provide users with a visual indication when the cache is being built. If the crosstabulated data store includes may dimensions, rebuilding the cache can take a noticeable amount of time. The progress bar provides users with feedback about how long this process will take.

See Also

DimensionCount (see page 17790)

TProgressBar (see page 4615)

107.1.3.6 TDecisionCube Events

The events of the TDecisionCube class are listed here.

Published Events

	Name	Description
	AfterClose (see page 17456)	Occurs immediately after the crosstabulated data store has been deactivated.
	AfterOpen (see page 17456)	Occurs immediately after the crosstabulated data store has been activated.
	BeforeClose (see page 17456)	Occurs immediately before the crosstabulated data store is deactivated.
	BeforeOpen (see page 17457)	Occurs immediately before the crosstabulated data store is activated.
	OnLowCapacity (see page 17457)	Occurs when the multidimensional cache requires more memory than is allowed by the Capacity property.
	OnRefresh (see page 17458)	Occurs immediately before the dimension map is changed.

107.1.3.6.1 TDecisionCube.AfterClose Event

Occurs immediately after the crosstabulated data store has been deactivated.

Pascal

```
property AfterClose: TCubeNotifyEvent;
```

C++

```
__property TCubeNotifyEvent AfterClose;
```

Description

Write an AfterClose event handler to take specific action when the crosstabulated data store has been shut down. For example, an application could provide feedback about the state of the data store, or update the dimension map for the next time the data store is activated.

See Also

BeforeClose (see page 17796)

DimensionMap (see page 17790)

107.1.3.6.2 TDecisionCube.AfterOpen Event

Occurs immediately after the crosstabulated data store has been activated.

Pascal

```
property AfterOpen: TCubeNotifyEvent;
```

C++

```
__property TCubeNotifyEvent AfterOpen;
```

Description

Write an AfterOpen event handler to take specific action when the crosstabulated data store is activated. This is the first opportunity to determine how much memory is consumed by the multidimensional cache maintained by the data store.

See Also

BeforeOpen (see page 17796)

GetMemoryUsage (see page 17783)

107.1.3.6.3 TDecisionCube.BeforeClose Event

Occurs immediately before the crosstabulated data store is deactivated.

Pascal

```
property BeforeClose: TCubeNotifyEvent;
```

C++

```
__property TCubeNotifyEvent BeforeClose;
```

Description

Write a BeforeClose event handler to take specific action before the data store shuts down. For example, an application can store information about the current state of the cache before it is cleared and all associated memory freed.

See Also

AfterClose (see page 17795)

107.1.3.6.4 TDecisionCube.BeforeOpen Event

Occurs immediately before the crosstabulated data store is activated.

Pascal

```
property BeforeOpen: TCubeNotifyEvent;
```

C++

```
__property TCubeNotifyEvent BeforeOpen;
```

Description

Write a BeforeOpen event handler to take specific action before the crosstabulated data store is activated. This is an appropriate time to adjust the value of Capacity, or set the ShowProgressDialog based on the number of active dimensions in the DimensionMap.

This is the last opportunity to adjust the entries in the DimensionMap before the cache is built. If the data store has never been active before this, the DimensionMap property will be nil (Delphi) or NULL (C++) at this point. A BeforeOpen event handler can build a dimension map programmatically if it is nil (Delphi) or NULL (C++). If the dimension map is not nil (Delphi) or NULL (C++), the properties of the dimension items can be changed, but the order and number of the dimension items themselves should not be changed.

See Also

AfterOpen (see page 17795)

Capacity (see page 17788)

DimensionMap (see page 17790)

ShowProgressDialog (see page 17794)

107.1.3.6.5 TDecisionCube.OnLowCapacity Event

Occurs when the multidimensional cache requires more memory than is allowed by the Capacity property.

Pascal

```
property OnLowCapacity: TCapacityErrorEvent;
```

C++

```
__property TCapacityErrorEvent OnLowCapacity;
```

Description

Write an OnLowCapacity event handler to respond to ELowCapacityError exceptions. An OnLowCapacity event handler can change the value of Capacity or deactivate some of the dimensions in the dimension map in order to allow the cache to fit in the available memory. If the memory problem is handled by the OnLowCapacity event handler, change the Action parameter to eaContinue to allow the data store to continue building the cache. Otherwise, an ELowCapacityError exception will be raised.

See Also

Capacity (see page 17788)

DimensionMap (see page 17790)

ELowCapacityError (see page 17306)

107.1.3.6.6 TDecisionCube.OnRefresh Event

Occurs immediately before the dimension map is changed.

Pascal

```
property OnRefresh: TCubeRefreshEvent;
```

C++

```
__property TCubeRefreshEvent OnRefresh;
```

Description

Write an OnRefresh event handler to take specific action when the dimension map changes. The DimMap parameter specifies the new value that is about to be set for the DimensionMap property. Use an OnRefresh event handler to change the properties of these dimension items before the DimensionMap property is set and the cache is rebuilt. This is also an opportunity to reset the MaxDimensions or MaxSummaries property to accommodate the dimension map that is about to be set.

See Also

DimensionMap (see page 17790)

MaxDimensions (see page 17793)

MaxSummaries (see page 17793)

Refresh (see page 17785)

Decision Cube Properties and Events

107.1.4 TDecisionDataLink Class

Class Hierarchy

Pascal

```
TDecisionDataLink = class(TPersistent);
```

C++

```
class TDecisionDataLink : public TPersistent;
```

File

MXDB

Description

This is class MXDB.TDecisionDataLink.

107.1.4.1 TDecisionDataLink Members

The following tables list the members exposed by TDecisionDataLink.

Protected Fields

	Name	Description
	FBlocked (see page 17460)	This is FBlocked, a member of class TDecisionDataLink.

	FDecisionSource (see page 17461)	This is FDecisionSource, a member of class TDecisionDataLink.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	DecisionDataEvent (see page 17461)	This is DecisionDataEvent, a member of class TDecisionDataLink.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetDecisionSource (see page 17461)	This is SetDecisionSource, a member of class TDecisionDataLink.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 17459)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 17460)	Destroys the TPersistent instance and frees its memory.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	DecisionSource (see page 17462)	This is DecisionSource, a member of class TDecisionDataLink.

107.1.4.2 TDecisionDataLink.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TDecisionDataLink();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

107.1.4.3 TDecisionDataLink.Destroy Destructor

Destroys the TPersistent instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDecisionDataLink();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

Free (see page 22300)

Destroy (see page 22287)

107.1.4.4 TDecisionDataLink Fields

The fields of the TDecisionDataLink class are listed here.

Protected Fields

	Name	Description
	FBlocked (see page 17460)	This is FBlocked, a member of class TDecisionDataLink.
	FDecisionSource (see page 17461)	This is FDecisionSource, a member of class TDecisionDataLink.

107.1.4.4.1 TDecisionDataLink.FBlocked Field

Pascal

```
FBlocked: Boolean;
```

C++

```
Boolean FBlocked;
```

Description

This is FBlocked, a member of class TDecisionDataLink.

107.1.4.4.2 TDecisionDataLink.FDecisionSource Field

Pascal

```
FDecisionSource: TDecisionSource;
```

C++

```
TDecisionSource FDecisionSource;
```

Description

This is FDecisionSource, a member of class TDecisionDataLink.

107.1.4.5 TDecisionDataLink Methods

The methods of the TDecisionDataLink class are listed here.

Protected Methods

	Name	Description
	DecisionDataEvent (see page 17461)	This is DecisionDataEvent, a member of class TDecisionDataLink.
	SetDecisionSource (see page 17461)	This is SetDecisionSource, a member of class TDecisionDataLink.

107.1.4.5.1 TDecisionDataLink.DecisionDataEvent Method

Pascal

```
procedure DecisionDataEvent(Event: TDecisionDataEvent); virtual;
```

C++

```
virtual __fastcall DecisionDataEvent(TDecisionDataEvent Event);
```

Description

This is DecisionDataEvent, a member of class TDecisionDataLink.

107.1.4.5.2 TDecisionDataLink.SetDecisionSource Method

Pascal

```
procedure SetDecisionSource(source: TDecisionSource);
```

C++

```
__fastcall SetDecisionSource(TDecisionSource source);
```

Description

This is SetDecisionSource, a member of class TDecisionDataLink.

107.1.4.6 TDecisionDataLink Properties

The properties of the TDecisionDataLink class are listed here.

Public Properties

	Name	Description
	DecisionSource (see page 17462)	This is DecisionSource, a member of class TDecisionDataLink.

107.1.4.6.1 TDecisionDataLink.DecisionSource Property

Pascal

```
property DecisionSource: TDecisionSource;
```

C++

```
__property TDecisionSource DecisionSource;
```

Description

This is DecisionSource, a member of class TDecisionDataLink.

107.1.5 TDecisionSource Class

TDecisionSource links a decision control to a configuration (pivot state) of the data from a decision cube.

Class Hierarchy

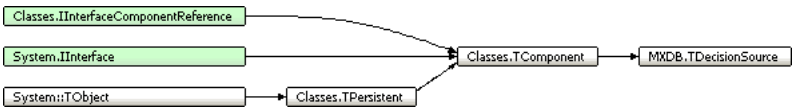

Pascal

```
TDecisionSource = class(TComponent);
```

C++

```
class TDecisionSource : public TComponent;
```

File

MXDB

Description

Use TDecisionSource to provide a decision grid or decision graph with cross-tabulated data from a decision cube. TDecisionSource represents a particular pivot state of dimensions from the decision cube into row and column dimensions. TDecisionSource also provides methods to drill down the display of a dimension to focus on summaries over a single value, or hide or show dimensions in the pivot state.

All decision controls that share a decision source display the same pivot state. For example, a form could contain a decision grid and a decision graph linked to the same decision source. Pivoting the dimensions of the decision grid would automatically update the image of the graph accordingly. To display two separate projections of the information from a decision cube in a single form, use two separate decision sources.

The pivot state of a decision source can be changed by manipulating a decision grid or by using a decision pivot.

See Also

TDecisionCube (see page 17442)

TDecisionGraph (see page 17521)

TDecisionGrid (see page 17586)

TDecisionPivot (see page 17680)

Creating Decision Grids

Overview of Decision Support Components

Using Decision Graphs

Decision Grid Properties

Using Decision Grids

Customizing Decision Graphs

Creating Decision Graphs

Using Decision Sources

107.1.5.1 TDecisionSource Members

The following tables list the members exposed by TDecisionSource.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDimGroup (see page 17483)	This is SetDimGroup, a member of class TDecisionSource.
	SetDimState (see page 17484)	This is SetDimState, a member of class TDecisionSource.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Fields

	Name	Description
	FOnAfterPivot (see page 17468)	This is FOnAfterPivot, a member of class TDecisionSource.
	FOnBeforePivot (see page 17468)	This is FOnBeforePivot, a member of class TDecisionSource.
	FOnLayoutChange (see page 17468)	This is FOnLayoutChange, a member of class TDecisionSource.
	FOnNewDimensions (see page 17468)	This is FOnNewDimensions, a member of class TDecisionSource.
	FOnStateChange (see page 17468)	This is FOnStateChange, a member of class TDecisionSource.
	FonSummaryChange (see page 17469)	This is FonSummaryChange, a member of class TDecisionSource.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CloseDimIndexRight (see page 17470)	Closes a dimension.
V	Create (see page 17467)	Creates and initializes an instance of TDecisionSource (see page 17462).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	DefineProperties (see page 17470)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	Destroy (see page 17467)	Destroys an instance of TDecisionSource (see page 17462).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DrillDimIndex (see page 17471)	Collapses a dimension to focus on a single value.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	Get2DDataAsVariant (see page 17472)	Returns the summary for a two dimensional projection of the current pivot state.
	GetActiveDim (see page 17472)	Converts a row-specific or column-specific dimension index into an absolute dimension index.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetDataAsString (see page 17473)	Returns the data value that should appear in the cell of a decision grid, formatted as a string.
	GetDataAsVariant (see page 17473)	Returns the data value that should appear in the cell of a decision grid.
	GetDimensionMemberCount (see page 17474)	Returns the number of distinct values to be displayed for a given dimension.
	GetDimensionName (see page 17474)	Returns the display name for a dimension.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetGroup (see page 17475)	Indicates where a dimension appears in the current pivot state.
	GetGroupCount (see page 17475)	Returns the number of dimensions there are of a specified type.
	GetGroupExtent (see page 17476)	Returns the range of cell indexes for a single value of a dimension.
	GetIndex (see page 17477)	Returns the row-specific or column-specific index of a dimension.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMemberAsString (see page 17477)	Returns the label string for a dimension value.
	GetMemberAsVariant (see page 17478)	Returns a value along a dimension.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetRowState (see page 17478)	Indicates whether the adjacent dimensions are open or closed.
	GetState (see page 17479)	Returns the state of a dimension.
	GetSummaryName (see page 17479)	Returns the name of an active summary in the decision cube.
	GetValue (see page 17480)	Returns the drill value of a drilled or paged dimension.
	GetValueArray (see page 17480)	Returns the values of all dimension fields over which a data cell is summarized.
	GetValueIndex (see page 17481)	Returns the index of a single dimension value that corresponds to a cell in a decision grid.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveDimIndexes (see page 17481)	Changes the position of a dimension.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	OpenDimIndexLeft (see page 17482)	Opens the first closed dimension that appears before a specified dimension.
	OpenDimIndexRight (see page 17482)	Opens the first closed dimension that appears after a specified dimension.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

	SetCurrentSummary (see page 17483)	Specifies which summary is provided by the decision source.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SwapDimIndexes (see page 17484)	Interchanges the positions of two dimensions.
	ToggleDimIndex (see page 17484)	Changes whether a dimension is open, closed, or drilled.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CurrentSum (see page 17486)	Indicates the index of the current summary from the set of active summaries maintained by the decision cube.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	nColDims (see page 17489)	Indicates the number of column dimensions in the current pivot state of the decision source.
R	nDataCols (see page 17489)	Indicates the number of columns provided to the data region of a decision grid.
R	nDataRows (see page 17490)	Indicates the number of rows provided to the data region of a decision grid.
R	nDims (see page 17490)	Indicates the number of active dimensions in the decision cube.
R	nOpenColDims (see page 17491)	Indicates the number of column dimensions provided by the decision source.
R	nOpenRowDims (see page 17491)	Indicates the number of row dimensions provided by the decision source.
R	nRowDims (see page 17491)	Indicates the number of row dimensions in the current pivot state of the decision source.
R	nSums (see page 17492)	Indicates the number of active summaries in the decision cube.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Ready (see page 17492)	Indicates whether the decision source is linked to an active decision cube.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnAfterPivot (see page 17494)	Occurs immediately after the current pivot state changes.
	OnBeforePivot (see page 17494)	Occurs just before the current pivot state changes.
	OnLayoutChange (see page 17495)	Occurs immediately after decision controls are updated to reflect a change to the pivot state of the decision source.
	OnNewDimensions (see page 17495)	Occurs immediately after the data provided by the decision cube changes.
	OnStateChange (see page 17496)	Occurs when the DecisionCube (see page 17487) property changes.
	OnSummaryChange (see page 17496)	Occurs immediately after the CurrentSum (see page 17486) property changes.

Published Properties

	Name	Description
	ControlType (see page 17485)	Specifies how the decision source responds to messages from the decision pivot.
	DecisionCube (see page 17487)	Specifies the decision cube that provides the crosstabulated data for the decision source.
	Name (see page 17487)	Specifies the name of the component as referenced in code.
	SparseCols (see page 17493)	Indicates whether empty columns are removed from the current projection.
	SparseRows (see page 17493)	Indicates whether empty rows are removed from the current projection.
	Tag (see page 2808)	Stores an integer value as part of a component.

107.1.5.2 TDecisionSource.Create Constructor

Creates and initializes an instance of TDecisionSource (see page 17462).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDecisionSource(TComponent * AOwner);
```

Description

Call Create to instantiate a decision source at runtime. Decision source objects placed in forms or data modules at design time are created automatically.

See Also

Create (see page 2766)

107.1.5.3 TDecisionSource.Destroy Destructor

Destroys an instance of TDecisionSource (see page 17462).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDecisionSource();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the decision source reference is not nil and only then calls Destroy.

For all decision controls linked to the decision source, Destroy changes the DecisionSource property to nil (Delphi) or NULL (C++).

See Also

Free (see page 22300)

DecisionSource (see page 17521)

DecisionSource (see page 17578)

107.1.5.4 TDecisionSource Fields

The fields of the TDecisionSource class are listed here.

Public Fields

	Name	Description
	FOnAfterPivot (see page 17468)	This is FOnAfterPivot, a member of class TDecisionSource.
	FOnBeforePivot (see page 17468)	This is FOnBeforePivot, a member of class TDecisionSource.
	FOnLayoutChange (see page 17468)	This is FOnLayoutChange, a member of class TDecisionSource.
	FOnNewDimensions (see page 17468)	This is FOnNewDimensions, a member of class TDecisionSource.
	FOnStateChange (see page 17468)	This is FOnStateChange, a member of class TDecisionSource.

	FonSummaryChange (see page 17469)	This is FonSummaryChange, a member of class TDecisionSource.

107.1.5.4.1 TDecisionSource.FOnAfterPivot Field

Pascal

```
FOnAfterPivot: TNotifyEvent;
```

C++

```
TNotifyEvent FOnAfterPivot;
```

Description

This is FOnAfterPivot, a member of class TDecisionSource.

107.1.5.4.2 TDecisionSource.FOnBeforePivot Field

Pascal

```
FOnBeforePivot: TNotifyEvent;
```

C++

```
TNotifyEvent FOnBeforePivot;
```

Description

This is FOnBeforePivot, a member of class TDecisionSource.

107.1.5.4.3 TDecisionSource.FOnLayoutChange Field

Pascal

```
FOnLayoutChange: TNotifyEvent;
```

C++

```
TNotifyEvent FOnLayoutChange;
```

Description

This is FOnLayoutChange, a member of class TDecisionSource.

107.1.5.4.4 TDecisionSource.FOnNewDimensions Field

Pascal

```
FOnNewDimensions: TNotifyEvent;
```

C++

```
TNotifyEvent FOnNewDimensions;
```

Description

This is FOnNewDimensions, a member of class TDecisionSource.

107.1.5.4.5 TDecisionSource.FOnStateChange Field

Pascal

```
FOnStateChange: TNotifyEvent;
```

C++

```
TNotifyEvent FOnStateChange;
```

Description

This is FOnStateChange, a member of class TDecisionSource.

107.1.5.4.6 TDecisionSource.FonSummaryChange Field

Pascal

```
FonSummaryChange: TNotifyEvent;
```

C++

```
TNotifyEvent FonSummaryChange;
```

Description

This is FonSummaryChange, a member of class TDecisionSource.

107.1.5.5 TDecisionSource Methods

The methods of the TDecisionSource class are listed here.

Protected Methods

	Name	Description
	SetDimGroup (see page 17483)	This is SetDimGroup, a member of class TDecisionSource.
	SetDimState (see page 17484)	This is SetDimState, a member of class TDecisionSource.

Public Methods

	Name	Description
	CloseDimIndexRight (see page 17470)	Closes a dimension.
V	DefineProperties (see page 17470)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DrillDimIndex (see page 17471)	Collapses a dimension to focus on a single value.
	Get2DDataAsVariant (see page 17472)	Returns the summary for a two dimensional projection of the current pivot state.
	GetActiveDim (see page 17472)	Converts a row-specific or column-specific dimension index into an absolute dimension index.
	GetDataAsString (see page 17473)	Returns the data value that should appear in the cell of a decision grid, formatted as a string.
	GetDataAsVariant (see page 17473)	Returns the data value that should appear in the cell of a decision grid.
	GetDimensionMemberCount (see page 17474)	Returns the number of distinct values to be displayed for a given dimension.
	GetDimensionName (see page 17474)	Returns the display name for a dimension.
	GetGroup (see page 17475)	Indicates where a dimension appears in the current pivot state.
	GetGroupCount (see page 17475)	Returns the number of dimensions there are of a specified type.
	GetGroupExtent (see page 17476)	Returns the range of cell indexes for a single value of a dimension.
	GetIndex (see page 17477)	Returns the row-specific or column-specific index of a dimension.
	GetMemberAsString (see page 17477)	Returns the label string for a dimension value.
	GetMemberAsVariant (see page 17478)	Returns a value along a dimension.
	GetRowState (see page 17478)	Indicates whether the adjacent dimensions are open or closed.
	GetState (see page 17479)	Returns the state of a dimension.
	GetSummaryName (see page 17479)	Returns the name of an active summary in the decision cube.
	GetValue (see page 17480)	Returns the drill value of a drilled or paged dimension.
	GetValueArray (see page 17480)	Returns the values of all dimension fields over which a data cell is summarized.
	GetValueIndex (see page 17481)	Returns the index of a single dimension value that corresponds to a cell in a decision grid.

	MoveDimIndexes (see page 17481)	Changes the position of a dimension.
	OpenDimIndexLeft (see page 17482)	Opens the first closed dimension that appears before a specified dimension.
	OpenDimIndexRight (see page 17482)	Opens the first closed dimension that appears after a specified dimension.
	SetCurrentSummary (see page 17483)	Specifies which summary is provided by the decision source.
	SwapDimIndexes (see page 17484)	Interchanges the positions of two dimensions.
	ToggleDimIndex (see page 17484)	Changes whether a dimension is open, closed, or drilled.

107.1.5.5.1 TDecisionSource.CloseDimIndexRight Method

Closes a dimension.

Pascal

```
procedure CloseDimIndexRight(dimGroup: TDimGroup; Index: Integer; bOpen: Boolean);
```

C++

```
__fastcall CloseDimIndexRight(TDimGroup dimGroup, int Index, Boolean bOpen);
```

Description

Call CloseDimIndexRight to close a dimension so that it does not appear in the projection displayed by decision controls that use the decision source.

CloseDimIndexRight closes the first open dimension after the dimension identified by the Index parameter.

The dimGroup parameter indicates whether the dimension to be closed is a row dimension (dgRow) or a column dimension (dgCol). No other values for dimGroup should be used with the CloseDimIndexRight method.

If the dimGroup parameter is dgRow, Index identifies a row dimension, where 0 specifies the first row dimension, 1 specifies the second row dimension, and so on. If the dimGroup parameter is dgCol, Index identifies a column dimension, starting with an index of 0 for the first column dimension. No other values for dimGroup should be used with the CloseDimIndexRight method. To close the first open row or column dimension, set Index to -1.

The bOpen parameter specifies whether closed dimensions are counted when interpreting Index. Only open dimensions are counted when bOpen is true.

See Also

GetActiveDim (see page 17472)

GetRowState (see page 17478)

OpenDimIndexLeft (see page 17482)

OpenDimIndexRight (see page 17482)

ToggleDimIndex (see page 17484)

107.1.5.5.2 TDecisionSource.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are

defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor (see page 2839)

DefineBinaryProperty (see page 2837)

DefineProperties (see page 2931)

DefineProperty (see page 2838)

107.1.5.5.3 TDecisionSource.DrillDimIndex Method

Collapses a dimension to focus on a single value.

Pascal

```
procedure DrillDimIndex(dimGroup: TDimGroup; Index: Integer; ValueIndex: Integer; bOpen:
Boolean);
```

C++

```
__fastcall DrillDimIndex(TDimGroup dimGroup, int Index, int ValueIndex, Boolean bOpen);
```

Description

Call DrillDimIndex to "drill down" on a dimension, so that only the summaries for a single value are displayed.

The dimGroup parameter specifies the type of dimension to be drilled. If dimGroup is dgRow, the Index parameter identifies a row dimension to be drilled. If dimGroup is dgCol, Index identifies a column dimension to be drilled. No other values for dimGroup should be used with the DrillDimIndex method. The first row or column dimension has an Index of 0, the second dimension has an Index of 1, and so on.

The ValueIndex parameter is the index of single value from all values over the dimension that should be displayed. To display only the summaries over the entire dimension, rather than for a single value along the dimension, set ValueIndex to -1.

The bOpen parameter specifies whether closed dimensions are counted when interpreting Index. Only open dimensions are counted when bOpen is true.

To restore the dimension from its drilled state so that it displays all values again, use the ToggleDimIndex (see page 17484) method.

Note: Do not drill a dimension of the Decision Source unless the CurrentSum (see page 17486) property specifies an additive summary. Additive Summaries are SUM and COUNT. AVERAGE (unless calculated internally by the Decision Cube), MAX, and MIN are not additive.

See Also

GetActiveDim (see page 17472)

GetMemberAsString (see page 17477)

GetValueIndex (see page 17481)

ToggleDimIndex (see page 17484)

CurrentSum (see page 17486)

107.1.5.5.4 TDecisionSource.Get2DDataAsVariant Method

Returns the summary for a two dimensional projection of the current pivot state.

Pascal

```
function Get2DDataAsVariant(iDimA: Integer; iDimB: Integer; aValueIndex: Integer; bValueIndex:
Integer): Variant;
```

C++

```
__fastcall Variant Get2DDataAsVariant(int iDimA, int iDimB, int aValueIndex, int bValueIndex);
```

Description

Call Get2DDataAsVariant to obtain the summary value from the decision cube that corresponds to the coordinates of aValueIndex on the iDimA dimension and bValueIndex on the iDimB dimension. The returned value is a variant that summarizes all values over the other active dimensions, unless they are drilled. For drilled dimensions, only the drilled value is included in the summary.

The iDimA and iDimB parameters specify the dimensions for the two dimensional projection, where 0 indicates the first active dimension, 1 indicates the second active dimension, and so on. The dimension indexes are expressed as absolute indexes. To obtain an absolute index from a row or column index, use the GetActiveDim (see page 17472) method. The aValueIndex and bValueIndex parameters are the indexes of the values on the two dimensions, where 0 indicates the first value, 1 indicates the second value, and so on. Use the GetMemberAsString (see page 17477) or GetMemberAsVariant (see page 17478) method to obtain the field values that correspond to the aValueIndex and bValueIndex parameters.

See Also

DrillDimIndex (see page 17471)

GetActiveDim (see page 17472)

GetDataAsVariant (see page 17473)

GetMemberAsString (see page 17477)

GetMemberAsVariant (see page 17478)

107.1.5.5.5 TDecisionSource.GetActiveDim Method

Converts a row-specific or column-specific dimension index into an absolute dimension index.

Pascal

```
function GetActiveDim(dimGroup: TDimGroup; index: Integer; bOpen: Boolean): Integer;
```

C++

```
__fastcall int GetActiveDim(TDimGroup dimGroup, int index, Boolean bOpen);
```

Description

Call GetActiveDim to obtain the index of a dimension in the decision cube from its position in a decision grid. The names and values of row dimensions appear in the label cells on the right of the decision grid. The first row dimension appears in the first column, the second row dimension appears in the second column, and so on. Similarly, column dimensions appear in the label cells on the top of the decision grid, with the first column dimension appearing in the first row of the grid.

The dimGroup parameter indicates whether the dimension of interest is a row dimension or column dimension. If dimGroup is dgRow, the dimension is a row dimension. If dimGroup is dgCol, the dimension is a column dimension. GetActiveDim is only implemented for dimGroup values of dgRow and dgCol.

The Index parameter indicates the position of the dimension in the set of row or column dimensions, where 0 specifies the first

row or column dimension, 1 specifies the second dimension, and so on.

The bOpen parameter specifies whether closed dimensions are counted when interpreting Index. Only open dimensions are counted when bOpen is true.

GetActiveDim returns the absolute index of the dimension. Absolute indexes are used by many methods of the decision source, such as GetDimensionName (see page 17474) or GetMemberAsString (see page 17477).

Note: To obtain a row-specific index or column-specific index from an absolute dimension index, use the GetIndex (see page 17477) method.

See Also

GetDimensionName (see page 17474)

GetGroup (see page 17475)

GetIndex (see page 17477)

GetMemberAsString (see page 17477)

107.1.5.5.6 TDecisionSource.GetDataAsString Method

Returns the data value that should appear in the cell of a decision grid, formatted as a string.

Pascal

```
function GetDataAsString(ARow: Integer; ACol: Integer; var SubLevel: Integer): String;
```

C++

```
__fastcall AnsiString GetDataAsString(int ARow, int ACol, int SubLevel);
```

Description

Call GetDataAsString to obtain the display string for a data cell in a decision grid. The ARow and ACol parameters indicate the row and column numbers of the cell in the decision grid, where (0,0) specifies the top left cell in the data region.

GetDataAsString computes the dimension coordinates in the decision cube that correspond to the row and column indexes, making note of all dimensions over which subtotals must be taken. It then obtains the summary value from the decision cube, combining values for each summarized dimension. GetDataAsString returns the final summary value, formatted as a string. It also returns the number of summarized dimensions in the SubLevel parameter.

See Also

GetDataAsVariant (see page 17473)

107.1.5.5.7 TDecisionSource.GetDataAsVariant Method

Returns the data value that should appear in the cell of a decision grid.

Pascal

```
function GetDataAsVariant(Arow: Integer; ACol: Integer; var SubLevel: Integer): Variant;
```

C++

```
__fastcall Variant GetDataAsVariant(int Arow, int ACol, int SubLevel);
```

Description

Call GetDataAsVariant to obtain the value of a data cell in a decision grid. The ARow and ACol parameters indicate the row and column numbers of the cell in the decision grid, where (0,0) specifies the top left cell in the data region.

GetDataAsVariant computes the dimension coordinates in the decision cube that correspond to the row and column indexes,

making note of all dimensions over which subtotals must be taken. It then obtains the summary value from the decision cube, combining values for each summarized dimension. GetDataAsVariant returns the final summary value. It also returns the number of summarized dimensions in the SubLevel parameter.

See Also

Get2DDataAsVariant (see page 17472)

GetDataAsString (see page 17473)

107.1.5.5.8 TDecisionSource.GetDimensionMemberCount Method

Returns the number of distinct values to be displayed for a given dimension.

Pascal

```
function GetDimensionMemberCount(iDim: Integer): Integer;
```

C++

```
__fastcall int GetDimensionMemberCount(int iDim);
```

Description

Call GetDimensionMemberCount to determine the number of values in the dimension identified by the iDim parameter. The iDim parameter is the absolute index of the dimension, where 0 indicates the first active dimension in the decision cube, 1 indicates the second active dimension, and so on. To convert from a row-specific or column-specific dimension index into an absolute dimension index, use the GetActiveDim (see page 17472) method.

GetDimensionMemberCount provides an upper limit to the ValueIndex parameter of the GetMemberAsString (see page 17477) or GetMemberAsVariant (see page 17478) method.

Note: The value returned by GetDimensionMemberCount may change depending on the pivot state.

See Also

GetActiveDim (see page 17472)

GetMemberAsString (see page 17477)

GetMemberAsVariant (see page 17478)

SparseCols (see page 17493)

SparseRows (see page 17493)

107.1.5.5.9 TDecisionSource.GetDimensionName Method

Returns the display name for a dimension.

Pascal

```
function GetDimensionName(iDim: Integer): String;
```

C++

```
__fastcall AnsiString GetDimensionName(int iDim);
```

Description

Call GetDimensionName to obtain a name to display as a label for the dimension. The iDim parameter is the absolute index of the dimension, where 0 indicates the first active dimension in the decision cube, 1 indicates the second active dimension, and so on. To convert from a row-specific or column-specific dimension index into an absolute dimension index, use the GetActiveDim (see page 17472) method.

The name of a dimension is often the name of the field the dimension represents. The dimension name can be changed in the Decision Cube Editor or by setting the Name (see page 17487) property of the dimension in the dimension map of the decision cube.

See Also

GetActiveDim (see page 17472)

DimensionMap (see page 17790)

Name (see page 17414)

107.1.5.5.10 TDecisionSource.GetGroup Method

Indicates where a dimension appears in the current pivot state.

Pascal

```
function GetGroup(iDim: Integer): TDimGroup;
```

C++

```
__fastcall TDimGroup GetGroup(int iDim);
```

Description

Call GetGroup to determine how the dimension specified by the iDim parameter is used in the current pivot state. The iDim parameter is the absolute index of the dimension, where 0 is the first active dimension, 1 is the second active dimension, and so on.

If the dimension is a row dimension (displayed to the left of a decision grid or on the y-axis of a decision graph), GetGroup returns dgRow. If it is a column dimension (displayed at the top of a decision grid or on the x-axis of a decision graph), GetGroup returns dgCol. If the dimension is a paged dimension (not appearing in the decision control but limiting the data to a single value along the dimension), GetGroup returns dgPage. GetGroup never returns dgSum or dgNone.

Note: To determine the position of the dimension within the row dimensions or column dimensions, use the GetIndex (see page 17477) method.

See Also

GetActiveDim (see page 17472)

GetIndex (see page 17477)

GetGroupCount (see page 17475)

107.1.5.5.11 TDecisionSource.GetGroupCount Method

Returns the number of dimensions there are of a specified type.

Pascal

```
function GetGroupCount(dimGroup: TDimGroup; bOpen: Boolean): Integer;
```

C++

```
__fastcall int GetGroupCount(TDimGroup dimGroup, Boolean bOpen);
```

Description

Call GetGroupCount to determine how many row, column, or paged dimensions exist in the current pivot state.

Specify the type of dimension to count as the value of dimGroup. This value can be dgRow, dgCol, or dgPage. Do not use other TDimGroup (see page 17511) values with GetGroupCount.

bOpen indicates whether to count all dimensions (including those currently closed), or only the open dimensions.

Note: You can get the same information about row and column dimensions using the nRowDims (see page 17491), nOpenRowDims (see page 17491), nColDims (see page 17489), or nOpenColDims (see page 17491) property.

See Also

GetGroup (see page 17475)

GetGroupExtent (see page 17476)

nRowDims (see page 17491)

nOpenRowDims (see page 17491)

nColDims (see page 17489)

nOpenColDims (see page 17491)

107.1.5.5.12 TDecisionSource.GetGroupExtent Method

Returns the range of cell indexes for a single value of a dimension.

Pascal

```
function GetGroupExtent(dimGroup: TDimGroup; Index: Integer; Cell: Integer): TDimRange;
```

C++

```
__fastcall TDimRange GetGroupExtent(TDimGroup dimGroup, int Index, int Cell);
```

Description

Call GetGroupExtent to find the range of cells in a decision grid that has the same value on a given dimension as the cell specified by the Cell parameter.

Cell is the index of the row or column containing the cell, where 0 specifies the first row or column in the data area of a decision grid.

If dimGroup is dgRow, Cell identifies a row of the grid. If dimGroup is dgCol, Cell identifies a column of the grid. Do not use other values for dimGroup.

Index specifies the row-specific or column-specific index of the dimension.

For example, if a decision grid has two row dimensions listed in the first two columns of the decision grid, the line

```
MyRange := GetGroupExtent(dgRow, 0, 3);
MyRange = DecisionSource1->GetGroupExtent(dgRow, 0, 3);
```

sets MyRange to the range of row indexes for cells that have the same value on the first row dimension as the third cell. The number of cells in this range is the same as the value returned by GetDimensionMemberCount (see page 17474) for the second open row dimension. The line

```
MyRange := GetGroupExtent(dgRow, 1, 3);
MyRange = DecisionSource1->GetGroupExtent(dgRow, 1, 3);
```

sets MyRange to the range from 3 to 3, because the second row dimension is the innermost row dimension, so each value is represented by a single row.

Note: If the dimension includes subtotals, the subtotal cell is included in the range returned by GetGroupExtent.

See Also

GetDimensionMemberCount (see page 17474)

107.1.5.5.13 TDecisionSource.GetIndex Method

Returns the row-specific or column-specific index of a dimension.

Pascal

```
function GetIndex(iDim: Integer; bOpen: Boolean): Integer;
```

C++

```
__fastcall int GetIndex(int iDim, Boolean bOpen);
```

Description

Call GetIndex to obtain the row-specific index or column-specific index of a dimension from its index in the decision cube. GetIndex returns 0 for the first row dimension or the first column dimension. GetIndex returns 1 for the second row dimension or the second column dimension, and so on. The iDim parameter is the absolute index of the dimension. If the bOpen parameter is true, GetIndex is the position of the dimension when only open dimensions are counted. If bOpen is false, all dimensions, including closed dimensions are counted.

Use the GetGroup (see page 17475) method to determine whether the index returned by GetIndex is a row-specific index (GetGroup (see page 17475) returns dgRow) or a column-specific index (GetGroup (see page 17475) returns dgCol).

To obtain the index of a dimension in the decision cube from its row-specific or column-specific index, use the GetActiveDim (see page 17472) method.

See Also

GetActiveDim (see page 17472)

GetGroup (see page 17475)

107.1.5.5.14 TDecisionSource.GetMemberAsString Method

Returns the label string for a dimension value.

Pascal

```
function GetMemberAsString(iDim: Integer; ValueIndex: Integer): String;
```

C++

```
__fastcall AnsiString GetMemberAsString(int iDim, int ValueIndex);
```

Description

Call GetMemberAsString to obtain the value indicated by the ValueIndex parameter along the dimension specified by the iDim parameter, formatted as a string.

The iDim parameter is the absolute index of the dimension, where 0 indicates the first active dimension in the decision cube, 1 indicates the second active dimension, and so on. To convert from a row-specific or column-specific dimension index into an absolute dimension index, use the GetActiveDim (see page 17472) method.

The ValueIndex parameter is the index of a value along the dimension, where 0 specifies the first value of the dimension and GetDimensionMemberCount (see page 17474) returns the number of values along the dimension.

See Also

GetActiveDim (see page 17472)

GetDimensionMemberCount (see page 17474)

GetDimensionName (see page 17474)

GetMemberAsVariant (see page 17478)

107.1.5.5.15 TDecisionSource.GetMemberAsVariant Method

Returns a value along a dimension.

Pascal

```
function GetMemberAsVariant(iDim: Integer; ValueIndex: Integer): Variant;
```

C++

```
__fastcall Variant GetMemberAsVariant(int iDim, int ValueIndex);
```

Description

Call GetMemberAsVariant to obtain the value indicated by the ValueIndex parameter along the dimension specified by the iDim parameter.

The iDim parameter is the absolute index of the dimension, where 0 indicates the first active dimension in the decision cube, 1 indicates the second active dimension, and so on. To convert from a row-specific or column-specific dimension index into an absolute dimension index, use the GetActiveDim (see page 17472) method.

The ValueIndex parameter is the index of a value along the dimension, where 0 specifies the first value of the dimension and GetDimensionMemberCount (see page 17474) returns the number of values along the dimension.

See Also

GetActiveDim (see page 17472)

GetDimensionMemberCount (see page 17474)

GetDimensionName (see page 17474)

GetMemberAsString (see page 17477)

107.1.5.5.16 TDecisionSource.GetRowState Method

Indicates whether the adjacent dimensions are open or closed.

Pascal

```
function GetRowState(iDim: Integer): TRowState;
```

C++

```
__fastcall TRowState GetRowState(int iDim);
```

Description

Call GetRowState to determine the state of the dimensions that are adjacent to a dimension in the current pivot state. The iDim parameter specifies the absolute index of the dimension whose neighbors are to be checked. When iDim is 0, the dimension is the first active dimension in the decision cube. When iDim is 1, it is the second active dimension, and so on.

The return value includes rcPrevOpen if the previous dimension is open or rcPrevClosed if the previous dimension is closed. The return value includes rcNextOpen if the following dimension is open or rcNextClosed if it is closed. The first row or column dimension does not include either rcPrevOpen or rcPrevClosed. The last row or column does not include either rcNextOpen or rcNextClosed.

Note: Do not be confused by the name of the GetRowState method. It provides information about the adjacent column dimensions if iDim specifies a column dimension, or information about the adjacent row dimensions if iDim specifies a row dimension.

See Also

GetState (see page 17479)

107.1.5.5.17 TDecisionSource.GetState Method

Returns the state of a dimension.

Pascal

```
function GetState(iDim: Integer): TDimState;
```

C++

```
__fastcall TDimState GetState(int iDim);
```

Description

Call GetState to determine whether a specified dimension is open, closed, or drilled. The iDim parameter is the absolute index of the dimension, where 0 specifies the first active dimension in the decision cube, 1 specifies the second active dimension, and so on. GetState returns one of the following values:

Value	Meaning
dmClosed	The dimension is closed. None of its values appear in the decision control.
dmOpen	The dimension is open. All of its values appear in the decision control.
dmDrilled	The dimension is drilled. Only a single value (or a summary over all values) along the dimension appears in the decision control.
dmPaged	The dimension can only display a single value or summarize all values together.
dmNone	The dimension is not active.

Note: The state dmDrilled appears the same as dmPaged in a decision control. However, drilled dimensions can be changed to open or closed, while paged dimensions cannot change their state. Paged dimensions use considerably less memory than other types.

See Also

GetRowState (see page 17478)

GetValue (see page 17480)

ToggleDimIndex (see page 17484)

107.1.5.5.18 TDecisionSource.GetSummaryName Method

Returns the name of an active summary in the decision cube.

Pascal

```
function GetSummaryName(iSum: Integer): String;
```

C++

```
__fastcall AnsiString GetSummaryName(int iSum);
```

Description

Call GetSummaryName to determine the name of a summary. The iSum parameter is the index of the summary from the list of active summaries available from the decision cube. Use GetSummaryName to determine the name of the possible summaries that can be set using the SetCurrentSummary (see page 17483) method.

See Also

SetCurrentSummary (see page 17483)

SummaryCount (see page 17794)

107.1.5.5.19 TDecisionSource.GetValue Method

Returns the drill value of a drilled or paged dimension.

Pascal

```
function GetValue(iDim: Integer): Integer;
```

C++

```
__fastcall int GetValue(int iDim);
```

Description

Call GetValue to determine the value focused on by a drilled or paged dimension. Drilled dimensions and paged dimensions display only the data for a single field value along the dimension. GetValue returns the index of that single value. The value itself can be obtained from this index by calling the GetMemberAsString (see page 17477) or GetMemberAsVariant (see page 17478) method.

The iDim parameter is the absolute index of the drilled or paged dimension, where 0 specifies the first active dimension in the decision cube, 1 specifies the second active dimension, and so on.

Note: If the dimension is drilled so that it displays only summaries over all the values in the dimension, GetValue returns -1. If the dimension specified by the iDim parameter is not drilled or paged, GetValue returns -1.

See Also

DrillDimIndex (see page 17471)

GetMemberAsString (see page 17477)

GetMemberAsVariant (see page 17478)

GetState (see page 17479)

107.1.5.5.20 TDecisionSource.GetValueArray Method

Returns the values of all dimension fields over which a data cell is summarized.

Pascal

```
function GetValueArray(ACol: Integer; ARow: Integer; var ValueArray: TSmallIntArray): Boolean;
```

C++

```
__fastcall Boolean GetValueArray(int ACol, int ARow, TSmallIntArray ValueArray);
```

Description

Call GetValueArray to obtain the coordinates of a data cell from a decision grid in the multidimensional array maintained by the decision cube. The ACol and ARow parameters represent the indexes of the column and row containing the cell, where (0,0) specifies the upper left data cell, not the top left cell in the grid. The ValueArray parameter returns the indexes of the field values along each active dimension. An index of -1 indicates that the cell represents a subtotal across all values in the corresponding dimension. Use the GetMemberAsString (see page 17477) or GetMemberAsVariant (see page 17478) method to convert the indexes in the ValueArray parameter into field values. GetValueArray returns a boolean indicating whether the function was successful.

See Also

GetMemberAsString (see page 17477)

GetMemberAsVariant (see page 17478)

GetValueIndex (see page 17481)

107.1.5.5.21 TDecisionSource.GetValueIndex Method

Returns the index of a single dimension value that corresponds to a cell in a decision grid.

Pascal

```
function GetValueIndex(dimGroup: TDimGroup; Index: Integer; Cell: Integer; var isBreak:
Boolean; var isSum: Boolean): Integer;
```

C++

```
__fastcall int GetValueIndex(TDimGroup dimGroup, int Index, int Cell, Boolean isBreak, Boolean
isSum);
```

Description

Call GetValueIndex to obtain the coordinate of a cell in a decision grid along the dimension specified by the Index parameter. The dimGroup parameter indicates whether the dimension is a row dimension (dgRow) or a column dimension (dgCol). Do not call GetValueIndex with dimGroup set to any other TDimGroup (see page 17511) value. The Index parameter is the row- or column-specific index of the dimension. The first row or column dimension has an Index of 0, the second dimension has an Index of 1, and so on. The Cell parameter is the index of the cell row or column, where 0 specifies the first row or column in the data region of the grid.

In addition to returning the index of the dimension value, GetValueIndex provides additional information about the position of Cell within the cells for the dimension. GetValueIndex sets isBreak to truewhen Cell is the first cell in the dimension group with the returned value on the specified dimension. GetValueIndex sets iSum to truewhen Cell represents a subtotal over the dimension.

See Also

GetActiveDim (see page 17472)

GetGroupExtent (see page 17476)

GetValueArray (see page 17480)

107.1.5.5.22 TDecisionSource.MoveDimIndexes Method

Changes the position of a dimension.

Pascal

```
procedure MoveDimIndexes(SdimGroup: TDimGroup; DdimGroup: TDimGroup; SIndex: Integer; DIndex:
Integer; bOpen: Boolean);
```

C++

```
__fastcall MoveDimIndexes(TDimGroup SdimGroup, TDimGroup DdimGroup, int SIndex, int DIndex,
Boolean bOpen);
```

Description

Call MoveDimIndexes to change the pivot state of the decision grid. MoveDimIndexes moves the dimension indicated by the SIndex and SdimGroup parameters to the position indicated by the DIndex and DdimGroup parameters.

The SdimGroup and DdimGroup parameters specify whether the dimension position is along the rows of a decision grid (SdimGroup or DdimGroup is dgRow), or across the columns of the decision grid (SdimGroup or DdimGroup is dgCol). The

SIndex and DIndex parameters indicate the index of the dimension position among the row dimensions or column dimensions, where 0 specifies the first position, 1 specifies the second position, and so on. The bOpen parameter specifies whether closed dimensions are counted when interpreting the SIndex and DIndex parameters. When bOpen is true, only open dimensions are counted.

Note: Do not move a dimension of the Decision Source unless the CurrentSum (see page 17486) property specifies an additive summary. Additive Summaries are SUM and COUNT. AVERAGE (unless calculated internally by the Decision Cube), MAX, and MIN are not additive.

See Also

GetActiveDim (see page 17472)

SwapDimIndexes (see page 17484)

CurrentSum (see page 17486)

107.1.5.5.23 TDecisionSource.OpenDimIndexLeft Method

Opens the first closed dimension that appears before a specified dimension.

Pascal

```
procedure OpenDimIndexLeft(dimGroup: TDimGroup; Index: Integer; bOpen: Boolean);
```

C++

```
__fastcall OpenDimIndexLeft(TDimGroup dimGroup, int Index, Boolean bOpen);
```

Description

Call OpenDimIndexLeft to reopen a closed dimension so that its values appear in the decision control.

OpenDimIndexLeft opens the first closed dimension before the dimension identified by the Index parameter. If the dimGroup parameter is dgRow, Index identifies a row dimension, where 0 specifies the first row dimension, 1 specifies the second row dimension, and so on. If the dimGroup parameter is dgCol, Index identifies a column dimension, starting with an index of 0 for the first column dimension. No other values for dimGroup should be used with the OpenDimIndexLeft method. To open the last closed row or column dimension, set Index to -1. The bOpen parameter specifies whether closed dimensions are counted when interpreting Index. Only open dimensions are counted when bOpen is true.

See Also

CloseDimIndexRight (see page 17470)

GetActiveDim (see page 17472)

GetRowState (see page 17478)

OpenDimIndexRight (see page 17482)

ToggleDimIndex (see page 17484)

107.1.5.5.24 TDecisionSource.OpenDimIndexRight Method

Opens the first closed dimension that appears after a specified dimension.

Pascal

```
procedure OpenDimIndexRight(dimGroup: TDimGroup; Index: Integer; bOpen: Boolean);
```

C++

```
__fastcall OpenDimIndexRight(TDimGroup dimGroup, int Index, Boolean bOpen);
```

Description

Call OpenDimIndexRight to reopen a closed dimension so that its values appear in the decision control.

OpenDimIndexRight opens the first closed dimension after the dimension identified by the Index parameter. If the dimGroup parameter is dgRow, Index identifies a row dimension, where 0 specifies the first row dimension, 1 specifies the second row dimension, and so on. If the dimGroup parameter is dgCol, Index identifies a column dimension, starting with an index of 0 for the first column dimension. No other values for dimGroup should be used with the OpenDimIndexRight method. To open the first closed row or column dimension, set Index to -1. The bOpen parameter specifies whether closed dimensions are counted when interpreting Index. Only open dimensions are counted when bOpen is true.

See Also

CloseDimIndexRight (see page 17470)

GetActiveDim (see page 17472)

GetRowState (see page 17478)

OpenDimIndexLeft (see page 17482)

ToggleDimIndex (see page 17484)

107.1.5.5.25 TDecisionSource.SetCurrentSummary Method

Specifies which summary is provided by the decision source.

Pascal

```
procedure SetCurrentSummary(Value: Integer);
```

C++

```
__fastcall SetCurrentSummary(int Value);
```

Description

The decision cube maintains a set of active summaries in its cache. The decision source can only present one of those summaries to decision controls. Call SetCurrentSummary to specify which of the active summaries is surfaced by the decision source. The Value parameter specifies the index of the summary in the list of active summaries maintained by the decision cube.

To obtain the names of the active summaries from which the current summary is chosen, use the GetSummaryName (see page 17479) method.

See Also

GetSummaryName (see page 17479)

CurrentSummary (see page 17788)

107.1.5.5.26 TDecisionSource.SetDimGroup Method

Pascal

```
procedure SetDimGroup(iDim: Integer; Group: TDimGroup; Index: Integer; bOpen: Boolean);
```

C++

```
__fastcall SetDimGroup(int iDim, TDimGroup Group, int Index, Boolean bOpen);
```

Description

This is SetDimGroup, a member of class TDecisionSource.

107.1.5.5.27 TDecisionSource.SetDimState Method

Pascal

```
procedure SetDimState(iDim: Integer; State: TDimState; ValueIndex: Integer);
```

C++

```
__fastcall SetDimState(int iDim, TDimState State, int ValueIndex);
```

Description

This is SetDimState, a member of class TDecisionSource.

107.1.5.5.28 TDecisionSource.SwapDimIndexes Method

Interchanges the positions of two dimensions.

Pascal

```
procedure SwapDimIndexes(SdimGroup: TDimGroup; DdimGroup: TDimGroup; SIndex: Integer; DIndex:
Integer; bOpen: Boolean);
```

C++

```
__fastcall SwapDimIndexes(TDimGroup SdimGroup, TDimGroup DdimGroup, int SIndex, int DIndex,
Boolean bOpen);
```

Description

Call SwapDimIndexes to change the pivot state of the decision grid. SwapDimIndexes interchanges the positions of the dimension indicated by the SIndex and SdimGroup parameters and the dimension indicated by the DIndex and DdimGroup parameters.

The SdimGroup and DdimGroup parameters specify whether the dimension position is along the rows of a decision grid (SdimGroup or DdimGroup is dgRow), or across the columns of the decision grid (SdimGroup or DdimGroup is false). The SIndex and DIndex parameters indicate the index of the dimension position among the row dimensions or column dimensions, where 0 specifies the first position, 1 specifies the second position, and so on. The bOpen parameter specifies whether closed dimensions are considered when interpreting the SIndex and DIndex parameters. When bOpen is true, only open dimensions are counted.

Note: Do not move a dimension of the Decision Source unless the CurrentSum (see page 17486) property specifies an additive summary. Additive Summaries are SUM and COUNT. AVERAGE (unless calculated internally by the Decision Cube), MAX, and MIN are not additive.

See Also

GetActiveDim (see page 17472)

MoveDimIndexes (see page 17481)

107.1.5.5.29 TDecisionSource.ToggleDimIndex Method

Changes whether a dimension is open, closed, or drilled.

Pascal

```
procedure ToggleDimIndex(dimGroup: TDimGroup; Index: Integer; bOpen: Boolean);
```

C++

```
__fastcall ToggleDimIndex(TDimGroup dimGroup, int Index, Boolean bOpen);
```

Description

Call ToggleDimIndex to open a closed dimension, close an open dimension, or expand a drilled dimension.

When dimGroup is dgRow, the Index parameter specifies the row-specific index of the dimension. When dimGroup is dgCol, Index specifies the column-specific index of the dimension. The first row or column dimension has an Index of 0, the second dimension has an Index of 1, and so on. The bOpen parameter specifies whether closed dimensions should be counted when interpreting the Index parameter. When bOpen is true, only open dimensions are counted.

See Also

CloseDimIndexRight (see page 17470)

DrillDimIndex (see page 17471)

GetActiveDim (see page 17472)

GetState (see page 17479)

OpenDimIndexLeft (see page 17482)

OpenDimIndexRight (see page 17482)

107.1.5.6 TDecisionSource Properties

The properties of the TDecisionSource class are listed here.

Public Properties

	Name	Description
R	CurrentSum (see page 17486)	Indicates the index of the current summary from the set of active summaries maintained by the decision cube.
R	nColDims (see page 17489)	Indicates the number of column dimensions in the current pivot state of the decision source.
R	nDataCols (see page 17489)	Indicates the number of columns provided to the data region of a decision grid.
R	nDataRows (see page 17490)	Indicates the number of rows provided to the data region of a decision grid.
R	nDims (see page 17490)	Indicates the number of active dimensions in the decision cube.
R	nOpenColDims (see page 17491)	Indicates the number of column dimensions provided by the decision source.
R	nOpenRowDims (see page 17491)	Indicates the number of row dimensions provided by the decision source.
R	nRowDims (see page 17491)	Indicates the number of row dimensions in the current pivot state of the decision source.
R	nSums (see page 17492)	Indicates the number of active summaries in the decision cube.
R	Ready (see page 17492)	Indicates whether the decision source is linked to an active decision cube.

Published Properties

	Name	Description
	ControlType (see page 17485)	Specifies how the decision source responds to messages from the decision pivot.
	DecisionCube (see page 17487)	Specifies the decision cube that provides the crosstabulated data for the decision source.
	Name (see page 17487)	Specifies the name of the component as referenced in code.
	SparseCols (see page 17493)	Indicates whether empty columns are removed from the current projection.
	SparseRows (see page 17493)	Indicates whether empty rows are removed from the current projection.

107.1.5.6.1 TDecisionSource.ControlType Property

Specifies how the decision source responds to messages from the decision pivot.

Pascal

```
property ControlType: TDecisionControlType;
```

C++

```
__property TDecisionControlType ControlType;
```

Description

Use ControlType to specify what combination of row or column dimensions can be open at the same time. ControlType has one of the following values:

Value	Meaning
xtCheck	Pressing dimension buttons opens or closes an individual dimension, changing the number of open dimensions.
xtRadio	Pressing dimension buttons opens or closes the selected dimension and closes all other dimensions in that direction. When ControlType is xtRadio, at most one row dimension and one column dimension may be open at a time.
xtRadionEx	Pressing dimension buttons opens the selected dimension and closes all other dimensions in that direction or closes the selected dimension and opens another dimension in the same direction. When ControlType is xtRadioEx, exactly one row dimension and one column dimension must be open at any time.

See Also

TDecisionPivot (see page 17680)

Using Decision Sources

107.1.5.6.2 TDecisionSource.CurrentSum Property

Indicates the index of the current summary from the set of active summaries maintained by the decision cube.

Pascal

```
property CurrentSum: Integer;
```

C++

```
__property int CurrentSum;
```

Description

Read CurrentSum to determine which of the active summaries is used to provide values to clients of the decision source. CurrentSum returns 0 if the first active summary is the current summary. CurrentSum returns nSums (see page 17492)-1 if the last active summary is the current summary.

To change the current summary, use the SetCurrentSummary (see page 17483) method. Use the GetSummaryName (see page 17479) method to obtain the name of the current summary.

Note: Do not pivot, subtotal, or drill the information provided by the Decision source unless the CurrentSum property refers to an additive summary. Additive Summaries are SUM and COUNT. AVERAGE (unless calculated internally by the Decision Cube), MAX, and MIN are not additive.

See Also

GetSummaryName (see page 17479)

nSums (see page 17492)

OnSummaryChange (see page 17496)

SetCurrentSummary (see page 17483)

107.1.5.6.3 TDecisionSource.DecisionCube Property

Specifies the decision cube that provides the crosstabulated data for the decision source.

Pascal

```
property DecisionCube: TDecisionCube;
```

C++

```
__property TDecisionCube DecisionCube;
```

Description

Set DecisionCube to specify where the decision source should fetch its data. The decision cube maintains a multidimensional array of summaries in a cache. The decision source can provide decision controls with any projection of the data in that cache.

See Also

TDecisionCube (see page 17442)

107.1.5.6.4 TDecisionSource.Name Property

Specifies the name of the component as referenced in code.

Pascal

```
property Name: TComponentName;
```

C++

```
__property TComponentName Name;
```

Description

Use Name to change the name of a component to reflect its purpose in the current application. By default, the IDE assigns sequential names based on the type of the component, such as 'Button1', 'Button2', and so on.

Use Name to refer to the component in code.

Warning: Changing Name at runtime causes any references to the old name to become undefined. Any subsequent code that uses the old name will cause an exception.

C++ Examples:

```
/*
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < ComponentCount; i++)
    ListBox1->Items->InsertObject(0,
```

```
                                    Components[i]->Name,
                                    (TObject *)Components[i]);
}

void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  if (Button == mbRight)
  {
    TClass ClassRef;
    int Index = ListBox1->ItemAtPos(Point(X,Y), true);
    // only components that are controls have a position
    // make sure the component is a control
    for (ClassRef = ListBox1->Items->Objects[Index]->ClassType();
         ClassRef != NULL;
         ClassRef = ClassRef->ClassParent())
      if (String(ClassRef->ClassName()) == "TControl")
      {
        TControl *TheObject = (TControl *)ListBox1->Items->Objects[Index];
        StatusBar1->SimpleText =
          TheObject->Name + " is at (" +
          IntToStr(TheObject->Left) + ", " +
          IntToStr(TheObject->Top) + ")";
         break;
      }
    if (ClassRef == NULL) // if it wasn't a control
      MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component’s coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to Form1.ComponentCount-1 do
    ListBox1.Items.InsertObject(
      0, Form1.Components[I].Name,
      Form1.Components[I] as TObject);
end;

procedure TForm1.ListBox1MouseUp(
  Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X,Y: integer);
var
  APoint: TPoint;
  Index: integer;
  TheObject: TControl;
begin
  if Button = mbRight then
  begin
```

```
    APoint.X := X;
    APoint.Y := Y;
    Index := ListBox1.ItemAtPos(APoint, True);
    if (ListBox1.Items.Objects[Index] is TControl) then
    begin
      TheObject :=
        (ListBox1.Items.Objects[Index] as TControl);
      StatusBar1.SimpleText := TheObject.Name + ' is at (' +
                               IntToStr(TheObject.Left) + ', ' +
                               IntToStr(TheObject.Top) + ') ';
    end
    else
      Beep;
    end;
  end;
```

See Also

ChangeName (see page 2770)

SetName (see page 2789)

ValidateRename (see page 2792)

Defining New Persistent Fields

Using TSQLQuery

Using TSQLStoredProc

Using TSQLTable

107.1.5.6.5 TDecisionSource.nColDims Property

Indicates the number of column dimensions in the current pivot state of the decision source.

Pascal

```
property nColDims: Integer;
```

C++

```
__property int nColDims;
```

Description

Read nColDims to determine the number of column dimensions the decision source can provide to a decision control. Column dimensions appear on the y axis of a decision graph or listed along the top of a decision grid. nColDims counts both the open and closed column dimensions. To determine the number of column dimensions actually provided to decision controls, use the nOpenColDims (see page 17491) property.

See Also

GetActiveDim (see page 17472)

nDims (see page 17490)

nOpenColDims (see page 17491)

nRowDims (see page 17491)

107.1.5.6.6 TDecisionSource.nDataCols Property

Indicates the number of columns provided to the data region of a decision grid.

Pascal

```
property nDataCols: Integer;
```

C++

```
__property int nDataCols;
```

Description

Read nDataCols to determine the number of summarized values the decision source provides for a row in a decision grid. Each of the columns counted by nDataCols represents a combination of values or dimension subtotals for each of the open column dimensions.

107.1.5.6.7 TDecisionSource.nDataRows Property

Indicates the number of rows provided to the data region of a decision grid.

Pascal

```
property nDataRows: Integer;
```

C++

```
__property int nDataRows;
```

Description

Read nDataRows to determine the number of summarized values the decision source provides for a column in a decision grid. Each of the rows counted by nDataRows represents a combination of values or dimension subtotals for each of the open row dimensions.

See Also

nOpenRowDims (see page 17491)

107.1.5.6.8 TDecisionSource.nDims Property

Indicates the number of active dimensions in the decision cube.

Pascal

```
property nDims: Integer;
```

C++

```
__property int nDims;
```

Description

Read nDims to determine the number of dimensions the decision source can provide to a decision control. The decision source does not necessarily provide decision controls with all the dimensions counted by the nDims property. Some of the dimensions may be closed.

The dimensions available through the decision source are divided into column dimensions and row dimensions. These are counted by the nColDims (see page 17489) and nRowDims (see page 17491) properties, respectively. nDims is equal to the sum of nColDims (see page 17489) and nRowDims (see page 17491).

See Also

nColDims (see page 17489)

nRowDims (see page 17491)

107.1.5.6.9 TDecisionSource.nOpenColDims Property

Indicates the number of column dimensions provided by the decision source.

Pascal

```
property nOpenColDims: Integer;
```

C++

```
__property int nOpenColDims;
```

Description

Read nOpenColDims to determine the number of open column dimensions the decision source provides to a decision control. Column dimensions appear on the y axis of a decision graph or listed along the top of a decision grid. nOpenColDims counts only the open column dimensions that appear in the decision control. To determine the number of column dimensions potentially available in the current pivot state, use the nColDims (see page 17489) property.

See Also

GetActiveDim (see page 17472)

nColDims (see page 17489)

nOpenRowDims (see page 17491)

107.1.5.6.10 TDecisionSource.nOpenRowDims Property

Indicates the number of row dimensions provided by the decision source.

Pascal

```
property nOpenRowDims: Integer;
```

C++

```
__property int nOpenRowDims;
```

Description

Read nOpenRowDims to determine the number of open row dimensions the decision source provides to a decision control. Row dimensions appear on the x axis of a decision graph or listed along the left side of a decision grid. nOpenRowDims counts only the open row dimensions that appear in the decision control. To determine the number of row dimensions potentially available in the current pivot state, use the nRowDims (see page 17491) property.

See Also

GetActiveDim (see page 17472)

nOpenColDims (see page 17491)

nRowDims (see page 17491)

107.1.5.6.11 TDecisionSource.nRowDims Property

Indicates the number of row dimensions in the current pivot state of the decision source.

Pascal

```
property nRowDims: Integer;
```

C++

```
__property int nRowDims;
```

Description

Read nRowDims to determine the number of row dimensions the decision source can provide to a decision control. Row dimensions appear on the x axis of a decision graph or listed along the left side of a decision grid. nRowDims counts both the open and closed row dimensions. To determine the number of row dimensions actually provided to decision controls, use the nOpenRowDims (see page 17491) property.

See Also

GetActiveDim (see page 17472)

nColDims (see page 17489)

nDims (see page 17490)

nOpenRowDims (see page 17491)

107.1.5.6.12 TDecisionSource.nSums Property

Indicates the number of active summaries in the decision cube.

Pascal

```
property nSums: Integer;
```

C++

```
__property int nSums;
```

Description

Read nSums to determine the number of summary types the decision source can provide to a decision control. The decision source can only provide decision controls with a single summary at a time. The CurrentSum (see page 17486) property indicates which of the nSums possibilities is currently provided by the decision source.

See Also

CurrentSum (see page 17486)

SetCurrentSummary (see page 17483)

107.1.5.6.13 TDecisionSource.Ready Property

Indicates whether the decision source is linked to an active decision cube.

Pascal

```
property Ready: Boolean;
```

C++

```
__property Boolean Ready;
```

Description

Read Ready to determine whether the decision source can provide data to a decision control.

See Also

DecisionCube (see page 17487)

Active (see page 17451)

107.1.5.6.14 TDecisionSource.SparseCols Property

Indicates whether empty columns are removed from the current projection.

Pascal

```
property SparseCols: Boolean;
```

C++

```
__property Boolean SparseCols;
```

Description

Set SparseCols to true to display columns that are empty in the current pivot state. Set SparseCols to false to show only those dimension values that have summarized data over one or more of the row dimensions.

Note: SparseCols does not change the value returned by GetDimensionMemberCount (see page 17474).

See Also

GetDimensionMemberCount (see page 17474)

SparseRows (see page 17493)

Using Decision Sources

107.1.5.6.15 TDecisionSource.SparseRows Property

Indicates whether empty rows are removed from the current projection.

Pascal

```
property SparseRows: Boolean;
```

C++

```
__property Boolean SparseRows;
```

Description

Set SparseRows to true to display rows that are empty in the current pivot state. Set SparseRows to false to show only those dimension values that have summarized data over one or more of the column dimensions.

Note: SparseRows doesn't change the value returned by GetDimensionMemberCount (see page 17474).

See Also

GetDimensionMemberCount (see page 17474)

SparseCols (see page 17493)

Using Decision Sources

107.1.5.7 TDecisionSource Events

The events of the TDecisionSource class are listed here.

Published Events

	Name	Description
	OnAfterPivot (see page 17494)	Occurs immediately after the current pivot state changes.
	OnBeforePivot (see page 17494)	Occurs just before the current pivot state changes.

	OnLayoutChange (see page 17495)	Occurs immediately after decision controls are updated to reflect a change to the pivot state of the decision source.
	OnNewDimensions (see page 17495)	Occurs immediately after the data provided by the decision cube changes.
	OnStateChange (see page 17496)	Occurs when the DecisionCube (see page 17487) property changes.
	OnSummaryChange (see page 17496)	Occurs immediately after the CurrentSum (see page 17486) property changes.

107.1.5.7.1 TDecisionSource.OnAfterPivot Event

Occurs immediately after the current pivot state changes.

Pascal

```
property OnAfterPivot: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAfterPivot;
```

Description

Write an OnAfterPivot event handler to take specific action after the information provided by the decision source changes but before any decision controls are updated to reflect the change. This is a good opportunity to save information about the pivot state.

When the pivot state of the decision source changes, an OnBeforePivot (see page 17494) occurs when the results of the change have been calculated, and OnAfterPivot occurs when the decision source has been updated to reflect the change. Once the decision controls that are linked to the decision pivot are updated to reflect the change, an OnLayoutChange (see page 17495) event occurs.

See Also

OnBeforePivot (see page 17494)

OnLayoutChange (see page 17495)

OnNewDimensions (see page 17495)

OnStateChange (see page 17496)

TDecisionPivot (see page 17680)

Using Decision Sources

107.1.5.7.2 TDecisionSource.OnBeforePivot Event

Occurs just before the current pivot state changes.

Pascal

```
property OnBeforePivot: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnBeforePivot;
```

Description

Write an OnBeforePivot event handler to take specific action just before the information provided by the decision source changes. When OnBeforePivot occurs, the results of the pivot have been computed but not implemented. Use the GetGroup (see page 17475), GetIndex (see page 17477), GetState (see page 17479), GetValue (see page 17480), and GetRowState (see page 17478) methods to get information about the pivot state that is about to be implemented. To get information about the pivot state that is ending, save the dimension information in an OnAfterPivot (see page 17494) event handler. The information about the new state can then be compared to information captured the last time the pivot state was

changed. The OnBeforePivot event handler can make additional alterations to the pivot state using the methods of the decision source.

See Also

OnAfterPivot (see page 17494)

GetGroup (see page 17475)

GetIndex (see page 17477)

GetRowState (see page 17478)

GetState (see page 17479)

GetValue (see page 17480)

TDecisionPivot (see page 17680)

Using Decision Sources

107.1.5.7.3 TDecisionSource.OnLayoutChange Event

Occurs immediately after decision controls are updated to reflect a change to the pivot state of the decision source.

Pascal

```
property OnLayoutChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnLayoutChange;
```

Description

Write an OnLayoutChange event handler to respond to changes in the configuration of the active dimensions. OnLayoutChange occurs when the pivot or drill state changes, but the information cached by the decision cube remains the same. When OnLayoutChange occurs, the information the decision source can potentially provide does not change, only the current configuration of that information. To respond to changes in the data cached by the decision cube, use the OnNewDimensions (see page 17495) event.

See Also

OnAfterPivot (see page 17494)

OnNewDimensions (see page 17495)

TDecisionPivot (see page 17680)

Using Decision Sources

107.1.5.7.4 TDecisionSource.OnNewDimensions Event

Occurs immediately after the data provided by the decision cube changes.

Pascal

```
property OnNewDimensions: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnNewDimensions;
```

Description

Write an OnNewDimensions event handler to respond to changes that alter the data array, such as when the active property of a

dimension in the data cube changes. When OnNewDimensions occurs, the information the decision source can provide changes. To respond to changes that reorganize the data but do not change the data cached by the data cube, use the OnEndPivot or OnLayoutChange (see page 17495) event. To respond to changes in the data cube itself, use the OnStateChange (see page 17496) event.

See Also

OnAfterPivot (see page 17494)

OnLayoutChange (see page 17495)

OnStateChange (see page 17496)

OnSummaryChange (see page 17496)

Using Decision Sources

107.1.5.7.5 TDecisionSource.OnStateChange Event

Occurs when the DecisionCube (see page 17487) property changes.

Pascal

```
property OnStateChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnStateChange;
```

Description

Write an OnStateChange event handler to respond when the DecisionCube (see page 17487) property changes or when the decision cube's Active property changes. To respond to less drastic changes, use the OnNewDimensions (see page 17495) event, the OnLayoutChange (see page 17495) event, or the OnEndPivot event.

See Also

DecisionCube (see page 17487)

OnAfterPivot (see page 17494)

OnLayoutChange (see page 17495)

OnNewDimensions (see page 17495)

Active (see page 17451)

Using Decision Sources

107.1.5.7.6 TDecisionSource.OnSummaryChange Event

Occurs immediately after the CurrentSum (see page 17486) property changes.

Pascal

```
property OnSummaryChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSummaryChange;
```

Description

Write an OnSummaryChange event handler to take specific action when the summary provided by the decision source changes. The value of the CurrentSum (see page 17486) property reflects the new current summary.

See Also

CurrentSum (see page 17486)

OnNewDimensions (see page 17495)

Using Decision Sources

107.1.6 TDimInfoArray Class

Class Hierarchy

MXDB.TDimInfoArray

Pascal

```
TDimInfoArray = class;
```

C++

```
class TDimInfoArray;
```

File

MXDB

Description

This is class MXDB.TDimInfoArray.

107.1.6.1 TDimInfoArray Members

The following tables list the members exposed by TDimInfoArray.

Protected Methods

	Name	Description
	Find (see page 17499)	This is Find, a member of class TDimInfoArray.
	GetItem (see page 17500)	This is GetItem, a member of class TDimInfoArray.

Public Methods

	Name	Description
	Assign (see page 17498)	This is Assign, a member of class TDimInfoArray.
	Create (see page 17498)	This is Create, a member of class TDimInfoArray.
V	Destroy (see page 17498)	This is Destroy, a member of class TDimInfoArray.
	GetGroupArray (see page 17499)	This is GetGroupArray, a member of class TDimInfoArray.
	GetGroupIndex (see page 17499)	This is GetGroupIndex, a member of class TDimInfoArray.
	GetGroupItem (see page 17499)	This is GetGroupItem, a member of class TDimInfoArray.
	GetGroupSize (see page 17499)	This is GetGroupSize, a member of class TDimInfoArray.
	IsEqual (see page 17500)	This is IsEqual, a member of class TDimInfoArray.

Public Properties

	Name	Description
R	Items (see page 17500)	This is Items, a member of class TDimInfoArray.
R	Limit (see page 17500)	This is Limit, a member of class TDimInfoArray.

107.1.6.2 TDimInfoArray.Create Constructor

Pascal

```
constructor Create(ALimit: Integer);
```

C++

```
__fastcall TDimInfoArray(int ALimit);
```

Description

This is Create, a member of class TDimInfoArray.

107.1.6.3 TDimInfoArray.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDimInfoArray();
```

Description

This is Destroy, a member of class TDimInfoArray.

107.1.6.4 TDimInfoArray Methods

The methods of the TDimInfoArray class are listed here.

Protected Methods

	Name	Description
	Find (see page 17499)	This is Find, a member of class TDimInfoArray.
	GetItem (see page 17500)	This is GetItem, a member of class TDimInfoArray.

Public Methods

	Name	Description
	Assign (see page 17498)	This is Assign, a member of class TDimInfoArray.
	GetGroupArray (see page 17499)	This is GetGroupArray, a member of class TDimInfoArray.
	GetGroupIndex (see page 17499)	This is GetGroupIndex, a member of class TDimInfoArray.
	GetGroupItem (see page 17499)	This is GetGroupItem, a member of class TDimInfoArray.
	GetGroupSize (see page 17499)	This is GetGroupSize, a member of class TDimInfoArray.
	IsEqual (see page 17500)	This is IsEqual, a member of class TDimInfoArray.

107.1.6.4.1 TDimInfoArray.Assign Method

Pascal

```
procedure Assign(Value: TDimInfoArray);
```

C++

```
__fastcall Assign(TDimInfoArray Value);
```

Description

This is Assign, a member of class TDimInfoArray.

107.1.6.4.2 TDimInfoArray.Find Method

Pascal

```
function Find(Name: string; var pos: Integer): Boolean;
```

C++

```
__fastcall Boolean Find(AnsiString Name, int pos);
```

Description

This is Find, a member of class TDimInfoArray.

107.1.6.4.3 TDimInfoArray.GetGroupArray Method

Pascal

```
function GetGroupArray(Group: TDimGroup; bOpen: Boolean): TIArray;
```

C++

```
__fastcall TIArray GetGroupArray(TDimGroup Group, Boolean bOpen);
```

Description

This is GetGroupArray, a member of class TDimInfoArray.

107.1.6.4.4 TDimInfoArray.GetGroupIndex Method

Pascal

```
function GetGroupIndex(Group: TDimGroup; Index: Integer; bOpen: Boolean): Integer;
```

C++

```
__fastcall int GetGroupIndex(TDimGroup Group, int Index, Boolean bOpen);
```

Description

This is GetGroupIndex, a member of class TDimInfoArray.

107.1.6.4.5 TDimInfoArray.GetGroupItem Method

Pascal

```
function GetGroupItem(Group: TDimGroup; Index: Integer; bOpen: Boolean): PDimInfo;
```

C++

```
__fastcall PDimInfo GetGroupItem(TDimGroup Group, int Index, Boolean bOpen);
```

Description

This is GetGroupItem, a member of class TDimInfoArray.

107.1.6.4.6 TDimInfoArray.GetGroupSize Method

Pascal

```
function GetGroupSize(Group: TDimGroup; bOpen: Boolean): Integer;
```

C++

```
__fastcall int GetGroupSize(TDimGroup Group, Boolean bOpen);
```

Description

This is GetGroupSize, a member of class TDimInfoArray.

107.1.6.4.7 TDimInfoArray.GetItem Method

Pascal

```
function GetItem(Index: Integer): PDimInfo;
```

C++

```
__fastcall PDimInfo GetItem(int Index);
```

Description

This is GetItem, a member of class TDimInfoArray.

107.1.6.4.8 TDimInfoArray.IsEqual Method

Pascal

```
function IsEqual(Value: TDimInfoArray): Boolean;
```

C++

```
__fastcall Boolean IsEqual(TDimInfoArray Value);
```

Description

This is IsEqual, a member of class TDimInfoArray.

107.1.6.5 TDimInfoArray Properties

The properties of the TDimInfoArray class are listed here.

Public Properties

	Name	Description
R	Items (see page 17500)	This is Items, a member of class TDimInfoArray.
R	Limit (see page 17500)	This is Limit, a member of class TDimInfoArray.

107.1.6.5.1 TDimInfoArray.Items Property

Pascal

```
property Items [Index: Integer]: PDimInfo;
```

C++

```
__property PDimInfo Items[int Index];
```

Description

This is Items, a member of class TDimInfoArray.

107.1.6.5.2 TDimInfoArray.Limit Property

Pascal

```
property Limit: Integer;
```

C++

```
__property int Limit;
```

Description

This is Limit, a member of class TDimInfoArray.

107.1.7 TIArray Class

Class Hierarchy

MXDB.TIArray

Pascal

```
TIArray = class;
```

C++

```
class TIArray;
```

File

MXDB

Description

This is class MXDB.TIArray.

107.1.7.1 TIArray Members

The following tables list the members exposed by TIArray.

Protected Methods

	Name	Description
	GetItem (see page 17503)	This is GetItem, a member of class TIArray.
	SetItem (see page 17503)	This is SetItem, a member of class TIArray.

Public Methods

	Name	Description
	Assign (see page 17502)	This is Assign, a member of class TIArray.
	Create (see page 17502)	This is Create, a member of class TIArray.
V	Destroy (see page 17502)	This is Destroy, a member of class TIArray.
	InsertAt (see page 17503)	This is InsertAt, a member of class TIArray.
	RemoveItem (see page 17503)	This is RemoveItem, a member of class TIArray.

Public Properties

	Name	Description
	AutoSize (see page 17504)	This is AutoSize, a member of class TIArray.
	Items (see page 17504)	This is Items, a member of class TIArray.
R	Limit (see page 17504)	This is Limit, a member of class TIArray.

107.1.7.2 TIArray.Create Constructor

Pascal

```
constructor Create(ALimit: Integer; ABlockSize: Integer);
```

C++

```
__fastcall TIArray(int ALimit, int ABlockSize);
```

Description

This is Create, a member of class TIArray.

107.1.7.3 TIArray.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIArray();
```

Description

This is Destroy, a member of class TIArray.

107.1.7.4 TIArray Methods

The methods of the TIArray class are listed here.

Protected Methods

	Name	Description
	GetItem (see page 17503)	This is GetItem, a member of class TIArray.
	SetItem (see page 17503)	This is SetItem, a member of class TIArray.

Public Methods

	Name	Description
	Assign (see page 17502)	This is Assign, a member of class TIArray.
	InsertAt (see page 17503)	This is InsertAt, a member of class TIArray.
	RemoveItem (see page 17503)	This is RemoveItem, a member of class TIArray.

107.1.7.4.1 TIArray.Assign Method

Pascal

```
procedure Assign(Value: TIArray);
```

C++

```
__fastcall Assign(TIArray Value);
```

Description

This is Assign, a member of class TIArray.

107.1.7.4.2 TIArray.GetItem Method

Pascal

```
function GetItem(Index: Integer): Integer;
```

C++

```
__fastcall int GetItem(int Index);
```

Description

This is GetItem, a member of class TIArray.

107.1.7.4.3 TIArray.InsertAt Method

Pascal

```
procedure InsertAt(Index: Integer; Value: Integer);
```

C++

```
__fastcall InsertAt(int Index, int Value);
```

Description

This is InsertAt, a member of class TIArray.

107.1.7.4.4 TIArray.RemoveItem Method

Pascal

```
function RemoveItem(Index: Integer): Integer;
```

C++

```
__fastcall int RemoveItem(int Index);
```

Description

This is RemoveItem, a member of class TIArray.

107.1.7.4.5 TIArray.SetItem Method

Pascal

```
procedure SetItem(Index: Integer; Value: Integer);
```

C++

```
__fastcall SetItem(int Index, int Value);
```

Description

This is SetItem, a member of class TIArray.

107.1.7.5 TIArray Properties

The properties of the TIArray class are listed here.

Public Properties

	Name	Description
	AutoSize (see page 17504)	This is AutoSize, a member of class TIArray.
	Items (see page 17504)	This is Items, a member of class TIArray.
R	Limit (see page 17504)	This is Limit, a member of class TIArray.

107.1.7.5.1 TIArray.AutoSize Property

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

This is AutoSize, a member of class TIArray.

107.1.7.5.2 TIArray.Items Property

Pascal

```
property Items [Index: Integer]: Integer;
```

C++

```
__property int Items[int Index];
```

Description

This is Items, a member of class TIArray.

107.1.7.5.3 TIArray.Limit Property

Pascal

```
property Limit: Integer;
```

C++

```
__property int Limit;
```

Description

This is Limit, a member of class TIArray.

107.1.8 TPivotState Class

Class Hierarchy

MXDB.TPivotState

Pascal

```
TPivotState = class;
```

C++

```
class TPivotState;
```

File

MXDB

Description

This is class MXDB.TPivotState.

107.1.8.1 TPivotState Members

The following tables list the members exposed by TPivotState.

Protected Methods

	Name	Description
	Assign (see page 17508)	This is Assign, a member of class TPivotState.
	IsEqual (see page 17508)	This is IsEqual, a member of class TPivotState.

Public Fields

	Name	Description
	DimInfo (see page 17506)	This is DimInfo, a member of class TPivotState.
	FColSparse (see page 17506)	This is FColSparse, a member of class TPivotState.
	FColSubs (see page 17506)	This is FColSubs, a member of class TPivotState.
	FCurrentSum (see page 17507)	This is FCurrentSum, a member of class TPivotState.
	FDims (see page 17507)	This is FDims, a member of class TPivotState.
	FRowSparse (see page 17507)	This is FRowSparse, a member of class TPivotState.
	FRowSubs (see page 17507)	This is FRowSubs, a member of class TPivotState.
	FSums (see page 17507)	This is FSums, a member of class TPivotState.

Public Methods

	Name	Description
	Create (see page 17505)	This is Create, a member of class TPivotState.
V	Destroy (see page 17505)	This is Destroy, a member of class TPivotState.

107.1.8.2 TPivotState.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TPivotState();
```

Description

This is Create, a member of class TPivotState.

107.1.8.3 TPivotState.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPivotState();
```

Description

This is Destroy, a member of class TPivotState.

107.1.8.4 TPivotState Fields

The fields of the TPivotState class are listed here.

Public Fields

	Name	Description
	DimInfo (see page 17506)	This is DimInfo, a member of class TPivotState.
	FColSparse (see page 17506)	This is FColSparse, a member of class TPivotState.
	FColSubs (see page 17506)	This is FColSubs, a member of class TPivotState.
	FCurrentSum (see page 17507)	This is FCurrentSum, a member of class TPivotState.
	FDims (see page 17507)	This is FDims, a member of class TPivotState.
	FRowSparse (see page 17507)	This is FRowSparse, a member of class TPivotState.
	FRowSubs (see page 17507)	This is FRowSubs, a member of class TPivotState.
	FSums (see page 17507)	This is FSums, a member of class TPivotState.

107.1.8.4.1 TPivotState.DimInfo Field

Pascal

```
DimInfo: TDimInfoArray;
```

C++

```
TDimInfoArray DimInfo;
```

Description

This is DimInfo, a member of class TPivotState.

107.1.8.4.2 TPivotState.FColSparse Field

Pascal

```
FColSparse: Boolean;
```

C++

```
Boolean FColSparse;
```

Description

This is FColSparse, a member of class TPivotState.

107.1.8.4.3 TPivotState.FColSubs Field

Pascal

```
FColSubs: Boolean;
```

C++

```
Boolean FColSubs;
```

Description

This is FColSubs, a member of class TPivotState.

107.1.8.4.4 TPivotState.FCurrentSum Field

Pascal

```
FCurrentSum: Integer;
```

C++

```
int FCurrentSum;
```

Description

This is FCurrentSum, a member of class TPivotState.

107.1.8.4.5 TPivotState.FDims Field

Pascal

```
FDims: Integer;
```

C++

```
int FDims;
```

Description

This is FDims, a member of class TPivotState.

107.1.8.4.6 TPivotState.FRowSparse Field

Pascal

```
FRowSparse: Boolean;
```

C++

```
Boolean FRowSparse;
```

Description

This is FRowSparse, a member of class TPivotState.

107.1.8.4.7 TPivotState.FRowSubs Field

Pascal

```
FRowSubs: Boolean;
```

C++

```
Boolean FRowSubs;
```

Description

This is FRowSubs, a member of class TPivotState.

107.1.8.4.8 TPivotState.FSums Field

Pascal

```
FSums: Integer;
```

C++

```
int FSums;
```

Description

This is FSums, a member of class TPivotState.

107.1.8.5 TPivotState Methods

The methods of the TPivotState class are listed here.

Protected Methods

	Name	Description
	Assign (see page 17508)	This is Assign, a member of class TPivotState.
	IsEqual (see page 17508)	This is IsEqual, a member of class TPivotState.

107.1.8.5.1 TPivotState.Assign Method

Pascal

```
procedure Assign(Value: TPivotState);
```

C++

```
__fastcall Assign(TPivotState Value);
```

Description

This is Assign, a member of class TPivotState.

107.1.8.5.2 TPivotState.IsEqual Method

Pascal

```
function IsEqual(Value: TPivotState): Boolean;
```

C++

```
__fastcall Boolean IsEqual(TPivotState Value);
```

Description

This is IsEqual, a member of class TPivotState.

107.1.9 TQADecisionSource Class

Class Hierarchy

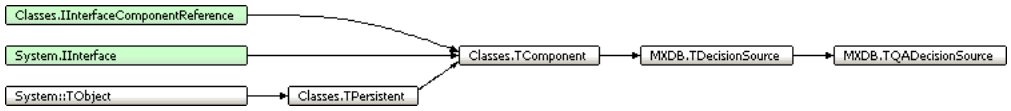

Pascal

```
TQADecisionSource = class(TDecisionSource);
```

C++

```
class TQADecisionSource : public TDecisionSource;
```

File

MXDB

Description

This is class MXDB.TQADecisionSource.

107.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TDecisionControlType (see page 17510)	This is record MXDB.TDecisionControlType.
	TDecisionDataEvent (see page 17510)	This is record MXDB.TDecisionDataEvent.
	TDimGroup (see page 17511)	This is record MXDB.TDimGroup.
	TDimState (see page 17512)	This is record MXDB.TDimState.
	TRowStates (see page 17513)	This is record MXDB.TRowStates.

Records

	Name	Description
	TDimInfo (see page 17511)	This is record MXDB.TDimInfo.
	TDimRange (see page 17512)	This is record MXDB.TDimRange.

107.2.1 MXDB.TDecisionControlType Enumeration

Pascal

```
TDecisionControlType = (
  xtCheck,
  xtRadio,
  xtRadioEx
);
```

C++

```
enum TDecisionControlType {
  xtCheck,
  xtRadio,
  xtRadioEx
};
```

File

MXDB

Description

This is record MXDB.TDecisionControlType.

107.2.2 MXDB.TDecisionDataEvent Enumeration

Pascal

```
TDecisionDataEvent = (
  xeStateChanged,
  xeSummaryChanged,
  xePivot,
  xeNewMetaData,
```

```
    xeSourceChange
  );
```

C++

```
  enum TDecisionDataEvent {
    xeStateChanged,
    xeSummaryChanged,
    xePivot,
    xeNewMetaData,
    xeSourceChange
  };
```

File

MXDB

Description

This is record MXDB.TDecisionDataEvent.

107.2.3 MXDB.TDimGroup Enumeration

Pascal

```
  TDimGroup = (
    dgRow,
    dgCol,
    dgSum,
    dgPage,
    dgNone
  );
```

C++

```
  enum TDimGroup {
    dgRow,
    dgCol,
    dgSum,
    dgPage,
    dgNone
  };
```

File

MXDB

Description

This is record MXDB.TDimGroup.

107.2.4 MXDB.TDimInfo Record

Pascal

```
  TDimInfo = record
    iState: TDimState;
    iValue: Integer;
    iIndex: Integer;
    iActiveIndex: Integer;
    iRowState: TRowState;
    iGroup: TDimGroup;
```

```
end;
```

C++

```
struct TDimInfo {
  TDimState iState;
  int iValue;
  int iIndex;
  int iActiveIndex;
  TRowState iRowState;
  TDimGroup iGroup;
};
```

File

MXDB

Description

This is record MXDB.TDimInfo.

107.2.5 MXDB.TDimRange Record

Pascal

```
TDimRange = record
  First: Integer;
  Last: Integer;
end;
```

C++

```
struct TDimRange {
  int First;
  int Last;
};
```

File

MXDB

Description

This is record MXDB.TDimRange.

107.2.6 MXDB.TDimState Enumeration

Pascal

```
TDimState = (
  dmClosed,
  dmOpen,
  dmDrilled,
  dmPaged,
  dmNone
);
```

C++

```
enum TDimState {
  dmClosed,
  dmOpen,
```

```
    dmDrilled,
    dmPaged,
    dmNone
  };
```

File

MXDB

Description

This is record MXDB.TDimState.

107.2.7 MXDB.TRowStates Enumeration

Pascal

```
  TRowStates = (
    rcNextOpen,
    rcPrevOpen,
    rcNextClosed,
    rcPrevClosed
  );
```

C++

```
  enum TRowStates {
    rcNextOpen,
    rcPrevOpen,
    rcNextClosed,
    rcPrevClosed
  };
```

File

MXDB

Description

This is record MXDB.TRowStates.

107.3 Types

The following table lists types in this documentation.

Types

Name	Description
PArrayDimInfo (see page 17514)	This is type MXDB.PArrayDimInfo.
PArrayInt (see page 17514)	This is type MXDB.PArrayInt.
PDimInfo (see page 17515)	This is type MXDB.PDimInfo.
TArrayDimInfo (see page 17515)	This is type MXDB.TArrayDimInfo.
TArrayInt (see page 17515)	This is type MXDB.TArrayInt.
TBigStr (see page 17516)	This is type MXDB.TBigStr.
TRowState (see page 17516)	This is type MXDB.TRowState.

107.3.1 MXDB.PArrayDimInfo Type

Pascal

```
PArrayDimInfo = ^TArrayDimInfo;
```

C++

```
^TArrayDimInfo PArrayDimInfo;
```

File

MXDB

Description

This is type MXDB.PArrayDimInfo.

107.3.2 MXDB.PArrayInt Type

Pascal

```
PArrayInt = ^TArrayInt;
```

C++

```
^TArrayInt PArrayInt;
```

File

MXDB

Description

This is type MXDB.PArrayInt.

107.3.3 MXDB.PDimInfo Type

Pascal

```
PDimInfo = ^TDimInfo;
```

C++

```
^TDimInfo PDimInfo;
```

File

MXDB

Description

This is type MXDB.PDimInfo.

107.3.4 MXDB.TArrayDimInfo Type

Pascal

```
TArrayDimInfo = array[0..999] of TDimInfo;
```

C++

```
array[0..999] of TDimInfo TArrayDimInfo;
```

File

MXDB

Description

This is type MXDB.TArrayDimInfo.

107.3.5 MXDB.TArrayInt Type

Pascal

```
TArrayInt = array[0..MaxInt div 8] of Integer;
```

C++

```
array[0..MaxInt div 8] of Integer TArrayInt;
```

File

MXDB

Description

This is type MXDB.TArrayInt.

107.3.6 MXDB.TBigStr Type

Pascal

```
TBigStr = array[0..100000] of Char;
```

C++

```
array[0..100000] of Char TBigStr;
```

File

MXDB

Description

This is type MXDB.TBigStr.

107.3.7 MXDB.TRowState Type

Pascal

```
TRowState = set of TRowStates;
```

C++

```
TRowStates TRowState;
```

File

MXDB

Description

This is type MXDB.TRowState.

108 MXGRAPH Namespace

This is namespace MXGRAPH.

Topics

Name	Description
Classes (see page 17518)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 17549)	The following table lists structs, records, enums in this documentation.
Types (see page 17550)	The following table lists types in this documentation.

Classes

	Name	Description
	TCustomDecisionGraph (see page 17518)	TCustomDecisionGraph is the base class for controls that graph crosstabulated data.
	TDecisionGraph (see page 17521)	TDecisionGraph displays a graph of crosstabulated data.
	TDecisionGraphDataLink (see page 17546)	This is class MXGRAPH.TDecisionGraphDataLink.

Structs, Records, Enums

	Name	Description
	TSeriesType (see page 17549)	This is record MXGRAPH.TSeriesType.

Types

Name	Description
TDimEvent (see page 17550)	This is type MXGRAPH.TDimEvent.

108.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCustomDecisionGraph (see page 17518)	TCustomDecisionGraph is the base class for controls that graph crosstabulated data.
	TDecisionGraph (see page 17521)	TDecisionGraph displays a graph of crosstabulated data.
	TDecisionGraphDataLink (see page 17546)	This is class MXGRAPH.TDecisionGraphDataLink.

108.1.1 TCustomDecisionGraph Class

TCustomDecisionGraph is the base class for controls that graph crosstabulated data.

Class Hierarchy

Pascal

```
TCustomDecisionGraph = class(TCustomChart);
```

C++

```
class TCustomDecisionGraph : public TCustomChart;
```

File

MXGRAPH

Description

Use TCustomDecisionGraph as a base class when defining objects that graph the data from a decision source. Decision graphs chart a single summary that is calculated over a number of dimensions. The dimensions appear as axes of the graph.

The field that is summarized and the fields (dimensions) over which it is summarized can be changed dynamically at runtime to view any configuration of summary and dimensions chosen from the dimension map of a decision cube. Changes to the dimensions and summary can be initiated using a TDecisionPivot object.

Do not create instances of TCustomDecisionGraph. To put a decision graph on a form, use TDecisionGraph (see page 17521).

See Also

TDecisionCube (see page 17442)

TDecisionGraph (see page 17521)

TDecisionPivot (see page 17680)

TDecisionSource (see page 17462)

108.1.1.1 TCustomDecisionGraph Members

The following tables list the members exposed by TCustomDecisionGraph.

Protected Methods

	Name	Description
	Notification (see page 17520)	This is Notification, a member of class TCustomDecisionGraph.

Public Methods

	Name	Description
	Create (see page 17519)	Creates and initializes an instance of TCustomDecisionGraph (see page 17518).
	Destroy (see page 17519)	Destroys an instance of TCustomDecisionGraph (see page 17518).
	Updated (see page 17520)	Responds when the properties of the decision graph change.

Public Properties

	Name	Description
	DecisionSource (see page 17521)	Specifies the decision source object that supplies data values for the decision graph.

108.1.1.2 TCustomDecisionGraph.Create Constructor

Creates and initializes an instance of TCustomDecisionGraph (see page 17518).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomDecisionGraph(TComponent * AOwner);
```

Description

Call Create to create a decision graph at runtime. Do not create instances of TCustomDecisionGraph (see page 17518). Instead, create a descendant of TCustomDecisionGraph (see page 17518) such as TDecisionGraph (see page 17521). Decision graphs placed in forms at design time are created automatically.

Pass a single Component as a parameter to provide the decision graph with an Owner that is responsible for freeing it.

See Also

Destroy (see page 17519)

TDecisionGraph (see page 17521)

108.1.1.3 TCustomDecisionGraph.Destroy Destructor

Destroys an instance of TCustomDecisionGraph (see page 17518).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomDecisionGraph();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the decision grid is not nil and only then calls Destroy.

See Also

Free (see page 22300)

108.1.1.4 TCustomDecisionGraph Methods

The methods of the TCustomDecisionGraph class are listed here.

Protected Methods

	Name	Description
	Notification (see page 17520)	This is Notification, a member of class TCustomDecisionGraph.

Public Methods

	Name	Description
	Updated (see page 17520)	Responds when the properties of the decision graph change.

108.1.1.4.1 TCustomDecisionGraph.Notification Method

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

This is Notification, a member of class TCustomDecisionGraph.

108.1.1.4.2 TCustomDecisionGraph.Updated Method

Responds when the properties of the decision graph change.

Pascal

```
procedure Updated; override;
```

C++

```
virtual __fastcall Updated();
```

Description

Do not call the Updated method. It is called automatically when the properties of the decision graph change. TCustomDecisionGraph (see page 17518) uses this method to refresh the layout of the graph to reflect any changes that may have been made to the series.

See Also

Series

108.1.1.5 TCustomDecisionGraph Properties

The properties of the TCustomDecisionGraph class are listed here.

Public Properties

	Name	Description
	DecisionSource (see page 17521)	Specifies the decision source object that supplies data values for the decision graph.

108.1.1.5.1 TCustomDecisionGraph.DecisionSource Property

Specifies the decision source object that supplies data values for the decision graph.

Pascal

```
property DecisionSource: TDecisionSource;
```

C++

```
__property TDecisionSource DecisionSource;
```

Description

Set DecisionSource to link the decision graph to the crosstabulated data from a decision cube. The DecisionSource object maintains information about the current pivot state of the data. Thus, any two decision controls that share the same decision source will display the data from the decision cube using the same configuration of dimensions, including any closed dimensions or drilled down values.

See Also

TDecisionSource (see page 17462)

TDecisionGraph (see page 17521)

108.1.2 TDecisionGraph Class

TDecisionGraph displays a graph of crosstabulated data.

Class Hierarchy

Pascal

```
TDecisionGraph = class(TCustomDecisionGraph);
```

C++

```
class TDecisionGraph : public TCustomDecisionGraph;
```

File

MXGRAPH

Description

Use TDecisionGraph to put a decision graph in a form. Decision graphs chart a single summary that is calculated over a number of dimensions. The axes of the graph correspond to these dimensions, which are provided by a decision source object. By default, the first row dimension appears as the x axis and the first column dimension appears as the y axis.

The field that is summarized and the fields (dimensions) over which it is summarized can be changed dynamically at runtime to view any configuration of summary and dimensions chosen from the dimension map of a decision cube. Changes to the dimensions and summary can be initiated using a TDecisionPivot object.

TDecisionGraph publishes many of the properties, events, and methods inherited from TCustomDecisionGraph (see page 17518), but introduces no new behavior.

To present the crosstabulated data in a grid, use TDecisionGrid instead.

See Also

TDecisionCube (see page 17442)

TDecisionGrid (see page 17586)

TDecisionPivot (see page 17680)

TDecisionSource (see page 17462)

Overview of Decision Support Components

Using Decision Graphs

Creating and Using Decision Graphs

Using Decision Pivots

Customizing Decision Graphs

Creating Decision Graphs

About Crosstabs

108.1.2.1 TDecisionGraph Members

The following tables list the members exposed by TDecisionGraph.

Protected Methods

	Name	Description
	Notification (see page 17520)	This is Notification, a member of class TCustomDecisionGraph.

Public Methods

	Name	Description
	Create (see page 17519)	Creates and initializes an instance of TCustomDecisionGraph (see page 17518).
	Destroy (see page 17519)	Destroys an instance of TCustomDecisionGraph (see page 17518).
	Updated (see page 17520)	Responds when the properties of the decision graph change.

Published Properties

	Name	Description
	Align (see page 17526)	This is Align, a member of class TDecisionGraph.
	AllowPanning (see page 17526)	This is AllowPanning, a member of class TDecisionGraph.
	AllowZoom (see page 17526)	This is AllowZoom, a member of class TDecisionGraph.
	Anchors (see page 17526)	This is Anchors, a member of class TDecisionGraph.
	AnimatedZoom (see page 17527)	This is AnimatedZoom, a member of class TDecisionGraph.
	AnimatedZoomSteps (see page 17527)	This is AnimatedZoomSteps, a member of class TDecisionGraph.
	AutoSize (see page 17527)	This is AutoSize, a member of class TDecisionGraph.
	AxisVisible (see page 17527)	This is AxisVisible, a member of class TDecisionGraph.
	BackColor (see page 17528)	This is BackColor, a member of class TDecisionGraph.
	BackImage (see page 17528)	This is BackImage, a member of class TDecisionGraph.
	BackImageInside (see page 17528)	This is BackImageInside, a member of class TDecisionGraph.
	BackImageMode (see page 17528)	This is BackImageMode, a member of class TDecisionGraph.
	BevelInner (see page 17528)	This is BevelInner, a member of class TDecisionGraph.
	BevelOuter (see page 17529)	This is BevelOuter, a member of class TDecisionGraph.
	BevelWidth (see page 17529)	This is BevelWidth, a member of class TDecisionGraph.
	BorderStyle (see page 17529)	This is BorderStyle, a member of class TDecisionGraph.
	BorderWidth (see page 17529)	This is BorderWidth, a member of class TDecisionGraph.

	BottomAxis (see page 17530)	This is BottomAxis, a member of class TDecisionGraph.
	BottomWall (see page 17530)	This is BottomWall, a member of class TDecisionGraph.
	Chart3DPercent (see page 17530)	This is Chart3DPercent, a member of class TDecisionGraph.
	ClipPoints (see page 17530)	This is ClipPoints, a member of class TDecisionGraph.
	Color (see page 17530)	This is Color, a member of class TDecisionGraph.
	Constraints (see page 17531)	This is Constraints, a member of class TDecisionGraph.
	DecisionSource (see page 17531)	Specifies the decision source object that supplies data values for the decision graph.
	DragCursor (see page 17531)	This is DragCursor, a member of class TDecisionGraph.
	DragKind (see page 17531)	This is DragKind, a member of class TDecisionGraph.
	DragMode (see page 17532)	This is DragMode, a member of class TDecisionGraph.
	Enabled (see page 17532)	This is Enabled, a member of class TDecisionGraph.
	Foot (see page 17532)	This is Foot, a member of class TDecisionGraph.
	Frame (see page 17532)	This is Frame, a member of class TDecisionGraph.
	Gradient (see page 17533)	This is Gradient, a member of class TDecisionGraph.
	LeftAxis (see page 17533)	This is LeftAxis, a member of class TDecisionGraph.
	LeftWall (see page 17533)	This is LeftWall, a member of class TDecisionGraph.
	Legend (see page 17533)	This is Legend, a member of class TDecisionGraph.
	MarginBottom (see page 17533)	This is MarginBottom, a member of class TDecisionGraph.
	MarginLeft (see page 17534)	This is MarginLeft, a member of class TDecisionGraph.
	MarginRight (see page 17534)	This is MarginRight, a member of class TDecisionGraph.
	MarginTop (see page 17534)	This is MarginTop, a member of class TDecisionGraph.
	MaxPointsPerPage (see page 17534)	This is MaxPointsPerPage, a member of class TDecisionGraph.
	Monochrome (see page 17535)	This is Monochrome, a member of class TDecisionGraph.
	OnAfterDraw (see page 17535)	This is OnAfterDraw, a member of class TDecisionGraph.
	OnAllowScroll (see page 17535)	This is OnAllowScroll, a member of class TDecisionGraph.
	OnCanResize (see page 17535)	This is OnCanResize, a member of class TDecisionGraph.
	OnClick (see page 17535)	This is OnClick, a member of class TDecisionGraph.
	OnClickAxis (see page 17536)	This is OnClickAxis, a member of class TDecisionGraph.
	OnClickBackground (see page 17536)	This is OnClickBackground, a member of class TDecisionGraph.
	OnClickLegend (see page 17536)	This is OnClickLegend, a member of class TDecisionGraph.
	OnClickSeries (see page 17536)	This is OnClickSeries, a member of class TDecisionGraph.
	OnConstrainedResize (see page 17537)	This is OnConstrainedResize, a member of class TDecisionGraph.
	OnDblClick (see page 17537)	This is OnDblClick, a member of class TDecisionGraph.
	OnDockDrop (see page 17537)	This is OnDockDrop, a member of class TDecisionGraph.
	OnDockOver (see page 17537)	This is OnDockOver, a member of class TDecisionGraph.
	OnDragDrop (see page 17537)	This is OnDragDrop, a member of class TDecisionGraph.
	OnDragOver (see page 17538)	This is OnDragOver, a member of class TDecisionGraph.
	OnEndDock (see page 17538)	This is OnEndDock, a member of class TDecisionGraph.
	OnEndDrag (see page 17538)	This is OnEndDrag, a member of class TDecisionGraph.
	OnEnter (see page 17538)	This is OnEnter, a member of class TDecisionGraph.
	OnExit (see page 17539)	This is OnExit, a member of class TDecisionGraph.
	OnGetAxisLabel (see page 17539)	This is OnGetAxisLabel, a member of class TDecisionGraph.
	OnGetLegendPos (see page 17539)	This is OnGetLegendPos, a member of class TDecisionGraph.
	OnGetLegendRect (see page 17539)	This is OnGetLegendRect, a member of class TDecisionGraph.
	OnGetLegendText (see page 17539)	This is OnGetLegendText, a member of class TDecisionGraph.
	OnGetNextAxisLabel (see page 17540)	This is OnGetNextAxisLabel, a member of class TDecisionGraph.
	OnGetSiteInfo (see page 17540)	This is OnGetSiteInfo, a member of class TDecisionGraph.
	OnMouseDown (see page 17540)	This is OnMouseDown, a member of class TDecisionGraph.
	OnMouseMove (see page 17540)	This is OnMouseMove, a member of class TDecisionGraph.
	OnMouseUp (see page 17541)	This is OnMouseUp, a member of class TDecisionGraph.
	OnPageChange (see page 17541)	This is OnPageChange, a member of class TDecisionGraph.
	OnResize (see page 17541)	This is OnResize, a member of class TDecisionGraph.
	OnScroll (see page 17541)	This is OnScroll, a member of class TDecisionGraph.
	OnStartDock (see page 17541)	This is OnStartDock, a member of class TDecisionGraph.

	OnStartDrag (see page 17542)	This is OnStartDrag, a member of class TDecisionGraph.
	OnUnDock (see page 17542)	This is OnUnDock, a member of class TDecisionGraph.
	OnUndoZoom (see page 17542)	This is OnUndoZoom, a member of class TDecisionGraph.
	OnZoom (see page 17542)	This is OnZoom, a member of class TDecisionGraph.
	Page (see page 17543)	This is Page, a member of class TDecisionGraph.
	ParentColor (see page 17543)	This is ParentColor, a member of class TDecisionGraph.
	ParentShowHint (see page 17543)	This is ParentShowHint, a member of class TDecisionGraph.
	PopupMenu (see page 17543)	This is PopupMenu, a member of class TDecisionGraph.
	RightAxis (see page 17543)	This is RightAxis, a member of class TDecisionGraph.
	ScaleLastPage (see page 17544)	This is ScaleLastPage, a member of class TDecisionGraph.
	SeriesList (see page 17544)	This is SeriesList, a member of class TDecisionGraph.
	ShowHint (see page 17544)	This is ShowHint, a member of class TDecisionGraph.
	TabOrder (see page 17544)	This is TabOrder, a member of class TDecisionGraph.
	TabStop (see page 17545)	This is TabStop, a member of class TDecisionGraph.
	Title (see page 17545)	This is Title, a member of class TDecisionGraph.
	TopAxis (see page 17545)	This is TopAxis, a member of class TDecisionGraph.
	View3D (see page 17545)	This is View3D, a member of class TDecisionGraph.
	View3DWalls (see page 17545)	This is View3DWalls, a member of class TDecisionGraph.
	Visible (see page 17546)	This is Visible, a member of class TDecisionGraph.

108.1.2.2 TDecisionGraph Properties

The properties of the TDecisionGraph class are listed here.

Published Properties

	Name	Description
	Align (see page 17526)	This is Align, a member of class TDecisionGraph.
	AllowPanning (see page 17526)	This is AllowPanning, a member of class TDecisionGraph.
	AllowZoom (see page 17526)	This is AllowZoom, a member of class TDecisionGraph.
	Anchors (see page 17526)	This is Anchors, a member of class TDecisionGraph.
	AnimatedZoom (see page 17527)	This is AnimatedZoom, a member of class TDecisionGraph.
	AnimatedZoomSteps (see page 17527)	This is AnimatedZoomSteps, a member of class TDecisionGraph.
	AutoSize (see page 17527)	This is AutoSize, a member of class TDecisionGraph.
	AxisVisible (see page 17527)	This is AxisVisible, a member of class TDecisionGraph.
	BackColor (see page 17528)	This is BackColor, a member of class TDecisionGraph.
	BackImage (see page 17528)	This is BackImage, a member of class TDecisionGraph.
	BackImageInside (see page 17528)	This is BackImageInside, a member of class TDecisionGraph.
	BackImageMode (see page 17528)	This is BackImageMode, a member of class TDecisionGraph.
	BevelInner (see page 17528)	This is BevelInner, a member of class TDecisionGraph.
	BevelOuter (see page 17529)	This is BevelOuter, a member of class TDecisionGraph.
	BevelWidth (see page 17529)	This is BevelWidth, a member of class TDecisionGraph.
	BorderStyle (see page 17529)	This is BorderStyle, a member of class TDecisionGraph.
	BorderWidth (see page 17529)	This is BorderWidth, a member of class TDecisionGraph.
	BottomAxis (see page 17530)	This is BottomAxis, a member of class TDecisionGraph.
	BottomWall (see page 17530)	This is BottomWall, a member of class TDecisionGraph.
	Chart3DPercent (see page 17530)	This is Chart3DPercent, a member of class TDecisionGraph.
	ClipPoints (see page 17530)	This is ClipPoints, a member of class TDecisionGraph.
	Color (see page 17530)	This is Color, a member of class TDecisionGraph.
	Constraints (see page 17531)	This is Constraints, a member of class TDecisionGraph.
	DecisionSource (see page 17531)	Specifies the decision source object that supplies data values for the decision graph.
	DragCursor (see page 17531)	This is DragCursor, a member of class TDecisionGraph.
	DragKind (see page 17531)	This is DragKind, a member of class TDecisionGraph.
	DragMode (see page 17532)	This is DragMode, a member of class TDecisionGraph.

	Enabled (see page 17532)	This is Enabled, a member of class TDecisionGraph.
	Foot (see page 17532)	This is Foot, a member of class TDecisionGraph.
	Frame (see page 17532)	This is Frame, a member of class TDecisionGraph.
	Gradient (see page 17533)	This is Gradient, a member of class TDecisionGraph.
	LeftAxis (see page 17533)	This is LeftAxis, a member of class TDecisionGraph.
	LeftWall (see page 17533)	This is LeftWall, a member of class TDecisionGraph.
	Legend (see page 17533)	This is Legend, a member of class TDecisionGraph.
	MarginBottom (see page 17533)	This is MarginBottom, a member of class TDecisionGraph.
	MarginLeft (see page 17534)	This is MarginLeft, a member of class TDecisionGraph.
	MarginRight (see page 17534)	This is MarginRight, a member of class TDecisionGraph.
	MarginTop (see page 17534)	This is MarginTop, a member of class TDecisionGraph.
	MaxPointsPerPage (see page 17534)	This is MaxPointsPerPage, a member of class TDecisionGraph.
	Monochrome (see page 17535)	This is Monochrome, a member of class TDecisionGraph.
	OnAfterDraw (see page 17535)	This is OnAfterDraw, a member of class TDecisionGraph.
	OnAllowScroll (see page 17535)	This is OnAllowScroll, a member of class TDecisionGraph.
	OnCanResize (see page 17535)	This is OnCanResize, a member of class TDecisionGraph.
	OnClick (see page 17535)	This is OnClick, a member of class TDecisionGraph.
	OnClickAxis (see page 17536)	This is OnClickAxis, a member of class TDecisionGraph.
	OnClickBackground (see page 17536)	This is OnClickBackground, a member of class TDecisionGraph.
	OnClickLegend (see page 17536)	This is OnClickLegend, a member of class TDecisionGraph.
	OnClickSeries (see page 17536)	This is OnClickSeries, a member of class TDecisionGraph.
	OnConstrainedResize (see page 17537)	This is OnConstrainedResize, a member of class TDecisionGraph.
	OnDblClick (see page 17537)	This is OnDblClick, a member of class TDecisionGraph.
	OnDockDrop (see page 17537)	This is OnDockDrop, a member of class TDecisionGraph.
	OnDockOver (see page 17537)	This is OnDockOver, a member of class TDecisionGraph.
	OnDragDrop (see page 17537)	This is OnDragDrop, a member of class TDecisionGraph.
	OnDragOver (see page 17538)	This is OnDragOver, a member of class TDecisionGraph.
	OnEndDock (see page 17538)	This is OnEndDock, a member of class TDecisionGraph.
	OnEndDrag (see page 17538)	This is OnEndDrag, a member of class TDecisionGraph.
	OnEnter (see page 17538)	This is OnEnter, a member of class TDecisionGraph.
	OnExit (see page 17539)	This is OnExit, a member of class TDecisionGraph.
	OnGetAxisLabel (see page 17539)	This is OnGetAxisLabel, a member of class TDecisionGraph.
	OnGetLegendPos (see page 17539)	This is OnGetLegendPos, a member of class TDecisionGraph.
	OnGetLegendRect (see page 17539)	This is OnGetLegendRect, a member of class TDecisionGraph.
	OnGetLegendText (see page 17539)	This is OnGetLegendText, a member of class TDecisionGraph.
	OnGetNextAxisLabel (see page 17540)	This is OnGetNextAxisLabel, a member of class TDecisionGraph.
	OnGetSiteInfo (see page 17540)	This is OnGetSiteInfo, a member of class TDecisionGraph.
	OnMouseDown (see page 17540)	This is OnMouseDown, a member of class TDecisionGraph.
	OnMouseMove (see page 17540)	This is OnMouseMove, a member of class TDecisionGraph.
	OnMouseUp (see page 17541)	This is OnMouseUp, a member of class TDecisionGraph.
	OnPageChange (see page 17541)	This is OnPageChange, a member of class TDecisionGraph.
	OnResize (see page 17541)	This is OnResize, a member of class TDecisionGraph.
	OnScroll (see page 17541)	This is OnScroll, a member of class TDecisionGraph.
	OnStartDock (see page 17541)	This is OnStartDock, a member of class TDecisionGraph.
	OnStartDrag (see page 17542)	This is OnStartDrag, a member of class TDecisionGraph.
	OnUnDock (see page 17542)	This is OnUnDock, a member of class TDecisionGraph.
	OnUndoZoom (see page 17542)	This is OnUndoZoom, a member of class TDecisionGraph.
	OnZoom (see page 17542)	This is OnZoom, a member of class TDecisionGraph.
	Page (see page 17543)	This is Page, a member of class TDecisionGraph.
	ParentColor (see page 17543)	This is ParentColor, a member of class TDecisionGraph.
	ParentShowHint (see page 17543)	This is ParentShowHint, a member of class TDecisionGraph.
	PopupMenu (see page 17543)	This is PopupMenu, a member of class TDecisionGraph.
	RightAxis (see page 17543)	This is RightAxis, a member of class TDecisionGraph.
	ScaleLastPage (see page 17544)	This is ScaleLastPage, a member of class TDecisionGraph.

	SeriesList (see page 17544)	This is SeriesList, a member of class TDecisionGraph.
	ShowHint (see page 17544)	This is ShowHint, a member of class TDecisionGraph.
	TabOrder (see page 17544)	This is TabOrder, a member of class TDecisionGraph.
	TabStop (see page 17545)	This is TabStop, a member of class TDecisionGraph.
	Title (see page 17545)	This is Title, a member of class TDecisionGraph.
	TopAxis (see page 17545)	This is TopAxis, a member of class TDecisionGraph.
	View3D (see page 17545)	This is View3D, a member of class TDecisionGraph.
	View3DWalls (see page 17545)	This is View3DWalls, a member of class TDecisionGraph.
	Visible (see page 17546)	This is Visible, a member of class TDecisionGraph.

108.1.2.2.1 TDecisionGraph.Align Property

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

This is Align, a member of class TDecisionGraph.

108.1.2.2.2 TDecisionGraph.AllowPanning Property

Pascal

```
property AllowPanning;
```

C++

```
__property AllowPanning;
```

Description

This is AllowPanning, a member of class TDecisionGraph.

108.1.2.2.3 TDecisionGraph.AllowZoom Property

Pascal

```
property AllowZoom;
```

C++

```
__property AllowZoom;
```

Description

This is AllowZoom, a member of class TDecisionGraph.

108.1.2.2.4 TDecisionGraph.Anchors Property

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

This is Anchors, a member of class TDecisionGraph.

108.1.2.2.5 TDecisionGraph.AnimatedZoom Property

Pascal

```
property AnimatedZoom;
```

C++

```
__property AnimatedZoom;
```

Description

This is AnimatedZoom, a member of class TDecisionGraph.

108.1.2.2.6 TDecisionGraph.AnimatedZoomSteps Property

Pascal

```
property AnimatedZoomSteps;
```

C++

```
__property AnimatedZoomSteps;
```

Description

This is AnimatedZoomSteps, a member of class TDecisionGraph.

108.1.2.2.7 TDecisionGraph.AutoSize Property

Pascal

```
property AutoSize;
```

C++

```
__property AutoSize;
```

Description

This is AutoSize, a member of class TDecisionGraph.

108.1.2.2.8 TDecisionGraph.AxisVisible Property

Pascal

```
property AxisVisible;
```

C++

```
__property AxisVisible;
```

Description

This is AxisVisible, a member of class TDecisionGraph.

108.1.2.2.9 TDecisionGraph.BackColor Property

Pascal

```
property BackColor;
```

C++

```
__property BackColor;
```

Description

This is BackColor, a member of class TDecisionGraph.

108.1.2.2.10 TDecisionGraph.BackImage Property

Pascal

```
property BackImage;
```

C++

```
__property BackImage;
```

Description

This is BackImage, a member of class TDecisionGraph.

108.1.2.2.11 TDecisionGraph.BackImageInside Property

Pascal

```
property BackImageInside;
```

C++

```
__property BackImageInside;
```

Description

This is BackImageInside, a member of class TDecisionGraph.

108.1.2.2.12 TDecisionGraph.BackImageMode Property

Pascal

```
property BackImageMode;
```

C++

```
__property BackImageMode;
```

Description

This is BackImageMode, a member of class TDecisionGraph.

108.1.2.2.13 TDecisionGraph.BevelInner Property

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

This is BevelInner, a member of class TDecisionGraph.

108.1.2.2.14 TDecisionGraph.BevelOuter Property

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

This is BevelOuter, a member of class TDecisionGraph.

108.1.2.2.15 TDecisionGraph.BevelWidth Property

Pascal

```
property BevelWidth;
```

C++

```
__property BevelWidth;
```

Description

This is BevelWidth, a member of class TDecisionGraph.

108.1.2.2.16 TDecisionGraph.BorderStyle Property

Pascal

```
property BorderStyle;
```

C++

```
__property BorderStyle;
```

Description

This is BorderStyle, a member of class TDecisionGraph.

108.1.2.2.17 TDecisionGraph.BorderWidth Property

Pascal

```
property BorderWidth;
```

C++

```
__property BorderWidth;
```

Description

This is BorderWidth, a member of class TDecisionGraph.

108.1.2.2.18 TDecisionGraph.BottomAxis Property

Pascal

```
property BottomAxis;
```

C++

```
__property BottomAxis;
```

Description

This is BottomAxis, a member of class TDecisionGraph.

108.1.2.2.19 TDecisionGraph.BottomWall Property

Pascal

```
property BottomWall;
```

C++

```
__property BottomWall;
```

Description

This is BottomWall, a member of class TDecisionGraph.

108.1.2.2.20 TDecisionGraph.Chart3DPercent Property

Pascal

```
property Chart3DPercent;
```

C++

```
__property Chart3DPercent;
```

Description

This is Chart3DPercent, a member of class TDecisionGraph.

108.1.2.2.21 TDecisionGraph.ClipPoints Property

Pascal

```
property ClipPoints;
```

C++

```
__property ClipPoints;
```

Description

This is ClipPoints, a member of class TDecisionGraph.

108.1.2.2.22 TDecisionGraph.Color Property

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

This is Color, a member of class TDecisionGraph.

108.1.2.2.23 TDecisionGraph.Constraints Property

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

This is Constraints, a member of class TDecisionGraph.

108.1.2.2.24 TDecisionGraph.DecisionSource Property

Specifies the decision source object that supplies data values for the decision graph.

Pascal

```
property DecisionSource: TDecisionSource;
```

C++

```
__property TDecisionSource DecisionSource;
```

Description

Set DecisionSource to link the decision graph to the crosstabulated data from a decision cube. The DecisionSource object maintains information about the current pivot state of the data. Thus, any two decision controls that share the same decision source will display the data from the decision cube using the same configuration of dimensions, including any closed dimensions or drilled down values.

See Also

TDecisionSource (see page 17462)

TDecisionGraph (see page 17521)

108.1.2.2.25 TDecisionGraph.DragCursor Property

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

This is DragCursor, a member of class TDecisionGraph.

108.1.2.2.26 TDecisionGraph.DragKind Property

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

This is DragKind, a member of class TDecisionGraph.

108.1.2.2.27 TDecisionGraph.DragMode Property

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

This is DragMode, a member of class TDecisionGraph.

108.1.2.2.28 TDecisionGraph.Enabled Property

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

This is Enabled, a member of class TDecisionGraph.

108.1.2.2.29 TDecisionGraph.Foot Property

Pascal

```
property Foot;
```

C++

```
__property Foot;
```

Description

This is Foot, a member of class TDecisionGraph.

108.1.2.2.30 TDecisionGraph.Frame Property

Pascal

```
property Frame;
```

C++

```
__property Frame;
```

Description

This is Frame, a member of class TDecisionGraph.

108.1.2.2.31 TDecisionGraph.Gradient Property

Pascal

```
property Gradient;
```

C++

```
__property Gradient;
```

Description

This is Gradient, a member of class TDecisionGraph.

108.1.2.2.32 TDecisionGraph.LeftAxis Property

Pascal

```
property LeftAxis;
```

C++

```
__property LeftAxis;
```

Description

This is LeftAxis, a member of class TDecisionGraph.

108.1.2.2.33 TDecisionGraph.LeftWall Property

Pascal

```
property LeftWall;
```

C++

```
__property LeftWall;
```

Description

This is LeftWall, a member of class TDecisionGraph.

108.1.2.2.34 TDecisionGraph.Legend Property

Pascal

```
property Legend;
```

C++

```
__property Legend;
```

Description

This is Legend, a member of class TDecisionGraph.

108.1.2.2.35 TDecisionGraph.MarginBottom Property

Pascal

```
property MarginBottom;
```

C++

```
__property MarginBottom;
```

Description

This is MarginBottom, a member of class TDecisionGraph.

108.1.2.2.36 TDecisionGraph.MarginLeft Property

Pascal

```
property MarginLeft;
```

C++

```
__property MarginLeft;
```

Description

This is MarginLeft, a member of class TDecisionGraph.

108.1.2.2.37 TDecisionGraph.MarginRight Property

Pascal

```
property MarginRight;
```

C++

```
__property MarginRight;
```

Description

This is MarginRight, a member of class TDecisionGraph.

108.1.2.2.38 TDecisionGraph.MarginTop Property

Pascal

```
property MarginTop;
```

C++

```
__property MarginTop;
```

Description

This is MarginTop, a member of class TDecisionGraph.

108.1.2.2.39 TDecisionGraph.MaxPointsPerPage Property

Pascal

```
property MaxPointsPerPage;
```

C++

```
__property MaxPointsPerPage;
```

Description

This is MaxPointsPerPage, a member of class TDecisionGraph.

108.1.2.2.40 TDecisionGraph.Monochrome Property

Pascal

```
property Monochrome;
```

C++

```
__property Monochrome;
```

Description

This is Monochrome, a member of class TDecisionGraph.

108.1.2.2.41 TDecisionGraph.OnAfterDraw Property

Pascal

```
property OnAfterDraw;
```

C++

```
__property OnAfterDraw;
```

Description

This is OnAfterDraw, a member of class TDecisionGraph.

108.1.2.2.42 TDecisionGraph.OnAllowScroll Property

Pascal

```
property OnAllowScroll;
```

C++

```
__property OnAllowScroll;
```

Description

This is OnAllowScroll, a member of class TDecisionGraph.

108.1.2.2.43 TDecisionGraph.OnCanResize Property

Pascal

```
property OnCanResize;
```

C++

```
__property OnCanResize;
```

Description

This is OnCanResize, a member of class TDecisionGraph.

108.1.2.2.44 TDecisionGraph.OnClick Property

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

This is OnClick, a member of class TDecisionGraph.

108.1.2.2.45 TDecisionGraph.OnClickAxis Property

Pascal

```
property OnClickAxis;
```

C++

```
__property OnClickAxis;
```

Description

This is OnClickAxis, a member of class TDecisionGraph.

108.1.2.2.46 TDecisionGraph.OnClickBackground Property

Pascal

```
property OnClickBackground;
```

C++

```
__property OnClickBackground;
```

Description

This is OnClickBackground, a member of class TDecisionGraph.

108.1.2.2.47 TDecisionGraph.OnClickLegend Property

Pascal

```
property OnClickLegend;
```

C++

```
__property OnClickLegend;
```

Description

This is OnClickLegend, a member of class TDecisionGraph.

108.1.2.2.48 TDecisionGraph.OnClickSeries Property

Pascal

```
property OnClickSeries;
```

C++

```
__property OnClickSeries;
```

Description

This is OnClickSeries, a member of class TDecisionGraph.

108.1.2.2.49 TDecisionGraph.OnConstrainedResize Property

Pascal

```
property OnConstrainedResize;
```

C++

```
__property OnConstrainedResize;
```

Description

This is OnConstrainedResize, a member of class TDecisionGraph.

108.1.2.2.50 TDecisionGraph.OnDblClick Property

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

This is OnDblClick, a member of class TDecisionGraph.

108.1.2.2.51 TDecisionGraph.OnDockDrop Property

Pascal

```
property OnDockDrop;
```

C++

```
__property OnDockDrop;
```

Description

This is OnDockDrop, a member of class TDecisionGraph.

108.1.2.2.52 TDecisionGraph.OnDockOver Property

Pascal

```
property OnDockOver;
```

C++

```
__property OnDockOver;
```

Description

This is OnDockOver, a member of class TDecisionGraph.

108.1.2.2.53 TDecisionGraph.OnDragDrop Property

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

This is OnDragDrop, a member of class TDecisionGraph.

108.1.2.2.54 TDecisionGraph.OnDragOver Property

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

This is OnDragOver, a member of class TDecisionGraph.

108.1.2.2.55 TDecisionGraph.OnEndDock Property

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

This is OnEndDock, a member of class TDecisionGraph.

108.1.2.2.56 TDecisionGraph.OnEndDrag Property

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

This is OnEndDrag, a member of class TDecisionGraph.

108.1.2.2.57 TDecisionGraph.OnEnter Property

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

This is OnEnter, a member of class TDecisionGraph.

108.1.2.2.58 TDecisionGraph.OnExit Property

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

This is OnExit, a member of class TDecisionGraph.

108.1.2.2.59 TDecisionGraph.OnGetAxisLabel Property

Pascal

```
property OnGetAxisLabel;
```

C++

```
__property OnGetAxisLabel;
```

Description

This is OnGetAxisLabel, a member of class TDecisionGraph.

108.1.2.2.60 TDecisionGraph.OnGetLegendPos Property

Pascal

```
property OnGetLegendPos;
```

C++

```
__property OnGetLegendPos;
```

Description

This is OnGetLegendPos, a member of class TDecisionGraph.

108.1.2.2.61 TDecisionGraph.OnGetLegendRect Property

Pascal

```
property OnGetLegendRect;
```

C++

```
__property OnGetLegendRect;
```

Description

This is OnGetLegendRect, a member of class TDecisionGraph.

108.1.2.2.62 TDecisionGraph.OnGetLegendText Property

Pascal

```
property OnGetLegendText;
```

C++

```
__property OnGetLegendText;
```

Description

This is OnGetLegendText, a member of class TDecisionGraph.

108.1.2.2.63 TDecisionGraph.OnGetNextAxisLabel Property

Pascal

```
property OnGetNextAxisLabel;
```

C++

```
__property OnGetNextAxisLabel;
```

Description

This is OnGetNextAxisLabel, a member of class TDecisionGraph.

108.1.2.2.64 TDecisionGraph.OnGetSiteInfo Property

Pascal

```
property OnGetSiteInfo;
```

C++

```
__property OnGetSiteInfo;
```

Description

This is OnGetSiteInfo, a member of class TDecisionGraph.

108.1.2.2.65 TDecisionGraph.OnMouseDown Property

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

This is OnMouseDown, a member of class TDecisionGraph.

108.1.2.2.66 TDecisionGraph.OnMouseMove Property

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

This is OnMouseMove, a member of class TDecisionGraph.

108.1.2.2.67 TDecisionGraph.OnMouseUp Property

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

This is OnMouseUp, a member of class TDecisionGraph.

108.1.2.2.68 TDecisionGraph.OnPageChange Property

Pascal

```
property OnPageChange;
```

C++

```
__property OnPageChange;
```

Description

This is OnPageChange, a member of class TDecisionGraph.

108.1.2.2.69 TDecisionGraph.OnResize Property

Pascal

```
property OnResize;
```

C++

```
__property OnResize;
```

Description

This is OnResize, a member of class TDecisionGraph.

108.1.2.2.70 TDecisionGraph.OnScroll Property

Pascal

```
property OnScroll;
```

C++

```
__property OnScroll;
```

Description

This is OnScroll, a member of class TDecisionGraph.

108.1.2.2.71 TDecisionGraph.OnStartDock Property

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

This is OnStartDock, a member of class TDecisionGraph.

108.1.2.2.72 TDecisionGraph.OnStartDrag Property

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

This is OnStartDrag, a member of class TDecisionGraph.

108.1.2.2.73 TDecisionGraph.OnUnDock Property

Pascal

```
property OnUnDock;
```

C++

```
__property OnUnDock;
```

Description

This is OnUnDock, a member of class TDecisionGraph.

108.1.2.2.74 TDecisionGraph.OnUndoZoom Property

Pascal

```
property OnUndoZoom;
```

C++

```
__property OnUndoZoom;
```

Description

This is OnUndoZoom, a member of class TDecisionGraph.

108.1.2.2.75 TDecisionGraph.OnZoom Property

Pascal

```
property OnZoom;
```

C++

```
__property OnZoom;
```

Description

This is OnZoom, a member of class TDecisionGraph.

108.1.2.2.76 TDecisionGraph.Page Property

Pascal

```
property Page;
```

C++

```
__property Page;
```

Description

This is Page, a member of class TDecisionGraph.

108.1.2.2.77 TDecisionGraph.ParentColor Property

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

This is ParentColor, a member of class TDecisionGraph.

108.1.2.2.78 TDecisionGraph.ParentShowHint Property

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

This is ParentShowHint, a member of class TDecisionGraph.

108.1.2.2.79 TDecisionGraph.PopupMenu Property

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

This is PopupMenu, a member of class TDecisionGraph.

108.1.2.2.80 TDecisionGraph.RightAxis Property

Pascal

```
property RightAxis;
```

C++

```
__property RightAxis;
```

Description

This is RightAxis, a member of class TDecisionGraph.

108.1.2.2.81 TDecisionGraph.ScaleLastPage Property

Pascal

```
property ScaleLastPage;
```

C++

```
__property ScaleLastPage;
```

Description

This is ScaleLastPage, a member of class TDecisionGraph.

108.1.2.2.82 TDecisionGraph.SeriesList Property

Pascal

```
property SeriesList;
```

C++

```
__property SeriesList;
```

Description

This is SeriesList, a member of class TDecisionGraph.

108.1.2.2.83 TDecisionGraph.ShowHint Property

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

This is ShowHint, a member of class TDecisionGraph.

108.1.2.2.84 TDecisionGraph.TabOrder Property

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

This is TabOrder, a member of class TDecisionGraph.

108.1.2.2.85 TDecisionGraph.TabStop Property

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

This is TabStop, a member of class TDecisionGraph.

108.1.2.2.86 TDecisionGraph.Title Property

Pascal

```
property Title;
```

C++

```
__property Title;
```

Description

This is Title, a member of class TDecisionGraph.

108.1.2.2.87 TDecisionGraph.TopAxis Property

Pascal

```
property TopAxis;
```

C++

```
__property TopAxis;
```

Description

This is TopAxis, a member of class TDecisionGraph.

108.1.2.2.88 TDecisionGraph.View3D Property

Pascal

```
property View3D;
```

C++

```
__property View3D;
```

Description

This is View3D, a member of class TDecisionGraph.

108.1.2.2.89 TDecisionGraph.View3DWalls Property

Pascal

```
property View3DWalls;
```

C++

```
__property View3DWalls;
```

Description

This is View3DWalls, a member of class TDecisionGraph.

108.1.2.2.90 TDecisionGraph.Visible Property

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

This is Visible, a member of class TDecisionGraph.

108.1.3 TDecisionGraphDataLink Class

Class Hierarchy

System::TObject → Classes.TPersistent → MXDB.TDecisionDataLink → MXGRAPH.TDecisionGraphDataLink

Pascal

```
TDecisionGraphDataLink = class(TDecisionDataLink);
```

C++

```
class TDecisionGraphDataLink : public TDecisionDataLink;
```

File

MXGRAPH

Description

This is class MXGRAPH.TDecisionGraphDataLink.

108.1.3.1 TDecisionGraphDataLink Members

The following tables list the members exposed by TDecisionGraphDataLink.

Protected Fields

	Name	Description
	FBlocked (see page 17460)	This is FBlocked, a member of class TDecisionDataLink.
	FDecisionSource (see page 17461)	This is FDecisionSource, a member of class TDecisionDataLink.

Protected Methods

	Name	Description
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	DecisionDataEvent (see page 17548)	This is DecisionDataEvent, a member of class TDecisionGraphDataLink.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetDecisionSource (see page 17461)	This is SetDecisionSource, a member of class TDecisionDataLink.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 17547)	This is Create, a member of class TDecisionGraphDataLink.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 17548)	Destroys the TPersistent instance and frees its memory.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	DecisionSource (see page 17462)	This is DecisionSource, a member of class TDecisionDataLink.

108.1.3.2 TDecisionGraphDataLink.Create Constructor

Pascal

```
constructor Create(AGraph: TCustomDecisionGraph);
```

C++

```
__fastcall TDecisionGraphDataLink(TCustomDecisionGraph AGraph);
```

Description

This is Create, a member of class TDecisionGraphDataLink.

108.1.3.3 TDecisionGraphDataLink.Destroy Destructor

Destroys the TPersistent instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDecisionGraphDataLink();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

Free (see page 22300)

Destroy (see page 22287)

108.1.3.4 TDecisionGraphDataLink Methods

The methods of the TDecisionGraphDataLink class are listed here.

Protected Methods

	Name	Description
	DecisionDataEvent (see page 17548)	This is DecisionDataEvent, a member of class TDecisionGraphDataLink.

108.1.3.4.1 TDecisionGraphDataLink.DecisionDataEvent Method

Pascal

```
procedure DecisionDataEvent(Event: TDecisionDataEvent); override;
```

C++

```
virtual __fastcall DecisionDataEvent(TDecisionDataEvent Event);
```

Description

This is DecisionDataEvent, a member of class TDecisionGraphDataLink.

108.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TSeriesType (see page 17549)	This is record MXGRAPH.TSeriesType.

108.2.1 MXGRAPH.TSeriesType Enumeration

Pascal

```
TSeriesType = (
  stNormal,
  stTemplate,
  st1D
);
```

C++

```
enum TSeriesType {
  stNormal,
  stTemplate,
  st1D
};
```

File

MXGRAPH

Description

This is record MXGRAPH.TSeriesType.

108.3 Types

The following table lists types in this documentation.

Types

Name	Description
TDimEvent (see page 17550)	This is type MXGRAPH.TDimEvent.

108.3.1 MXGRAPH.TDimEvent Type

Pascal

```
TDimEvent = procedure (Sender: TObject; iDim: Integer) of object;
```

C++

```
(Sender: TObject; iDim: Integer) ( TDimEvent)();
```

File

MXGRAPH

Description

This is type MXGRAPH.TDimEvent.

109 MXGRID Namespace

This is namespace MXGRID.

Topics

Name	Description
Classes (see page 17553)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 17672)	The following table lists structs, records, enums in this documentation.
Types (see page 17676)	The following table lists types in this documentation.

Classes

	Name	Description
	TCustomDecisionGrid (see page 17553)	TCustomDecisonGrid is the base class for controls that present cross-tabulated data in a two-dimensional grid.
	TDecisionGrid (see page 17586)	TDecisionGrid presents crosstabulated data from a decision source in a grid format.
	TDecisionGridDataLink (see page 17658)	This is class MXGRID.TDecisionGridDataLink.
	TDisplayDim (see page 17661)	TDisplayDim represents a dimension in a decision grid.
	TDisplayDims (see page 17666)	TDisplayDims represents a collection of TDisplayDim (see page 17661) objects.
	TValueArray (see page 17671)	This is class MXGRID.TValueArray.

Structs, Records, Enums

	Name	Description
	TButtonPlace (see page 17672)	This is record MXGRID.TButtonPlace.
	TDDNotifyType (see page 17672)	This is record MXGRID.TDDNotifyType.
	TDecisionCellType (see page 17673)	This is record MXGRID.TDecisionCellType.
	TDecisionDrawStates (see page 17673)	This is record MXGRID.TDecisionDrawStates.
	TDecisionGridOption (see page 17674)	This is record MXGRID.TDecisionGridOption.
	TDecisionGridState (see page 17675)	This is record MXGRID.TDecisionGridState.
	TDecisionPivotState (see page 17675)	This is record MXGRID.TDecisionPivotState.

Types

Name	Description
TDecisionDrawCellEvent (see page 17676)	This is type MXGRID.TDecisionDrawCellEvent.
TDecisionDrawState (see page 17676)	This is type MXGRID.TDecisionDrawState.
TDecisionExamineCellEvent (see page 17677)	This is type MXGRID.TDecisionExamineCellEvent.
TDecisionGridOptions (see page 17677)	This is type MXGRID.TDecisionGridOptions.

TDisplayDimClass (see page 17677)	This is type MXGRID.TDisplayDimClass.

109.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCustomDecisionGrid (see page 17553)	TCustomDecisonGrid is the base class for controls that present cross-tabulated data in a two-dimensional grid.
	TDecisionGrid (see page 17586)	TDecisionGrid presents crosstabulated data from a decision source in a grid format.
	TDecisionGridDataLink (see page 17658)	This is class MXGRID.TDecisionGridDataLink.
	TDisplayDim (see page 17661)	TDisplayDim represents a dimension in a decision grid.
	TDisplayDims (see page 17666)	TDisplayDims represents a collection of TDisplayDim (see page 17661) objects.
	TValueArray (see page 17671)	This is class MXGRID.TValueArray.

109.1.1 TCustomDecisionGrid Class

TCustomDecisonGrid is the base class for controls that present cross-tabulated data in a two-dimensional grid.

Class Hierarchy

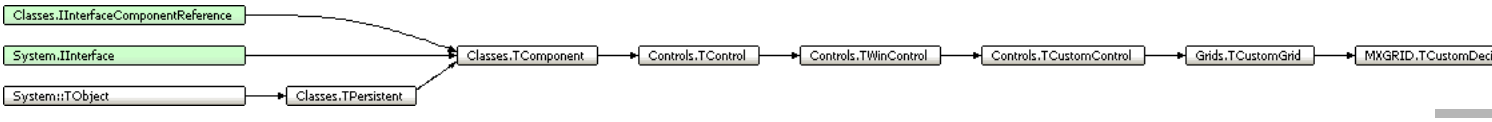

Pascal

```
TCustomDecisionGrid = class(TCustomGrid);
```

C++

```
class TCustomDecisionGrid : public TCustomGrid;
```

File

MXGRID

Description

Use TCustomDecisionGrid as a base class when defining a decision grid object. Decision grids can present a single summary that is calculated over an arbitrary number of dimensions.

The field that is summarized and the fields (dimensions) over which it is summarized can be changed dynamically at runtime to view any configuration of summary and dimensions chosen from the dimension map of a decision cube. Changes to the dimensions and summary can be initiated by direct manipulation of the decision grid, or by using a TDecisionPivot object.

Do not create instances of TCustomDecisionGrid. To add a decision grid to a form, use the TDecisionGrid (see page 17586) object.

See Also

TDecisionCube (see page 17442)

TDecisionGrid (see page 17586)

TDecisionPivot (see page 17680)

TDecisionSource (see page 17462)

109.1.1.1 TCustomDecisionGrid Members

The following tables list the members exposed by TCustomDecisionGrid.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 14814)	This is DesignOptionsBoost, a member of class TCustomGrid.

	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FGridState (see page 14814)	This is FGridState, a member of class TCustomGrid.
	FGridStateX (see page 17567)	This is FGridStateX, a member of class TCustomDecisionGrid.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveCellExtents (see page 14814)	This is FSaveCellExtents, a member of class TCustomGrid.
	VirtualView (see page 14814)	This is VirtualView, a member of class TCustomGrid.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AdjustSize (see page 14816)	Adds or deletes rows or columns from the grid.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	BeginColumnDrag (see page 14817)	Starts the dragging of a column in the grid.
	BeginRowDrag (see page 14817)	Starts the dragging of a row in the grid.
	BoxRect (see page 14818)	Returns the screen coordinates of a specified set of cells.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcDrawInfo (see page 14818)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 14818)	Calculates the pixel locations of the fixed cells.
V	CalcSizingState (see page 14819)	Determines the appearance of the mouse cursor during resizing.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanEditAcceptKey (see page 14819)	Indicates whether the Key parameter can be entered into the current cell.
	CanEditModify (see page 14819)	Indicates whether the current cell in the grid can be edited.
V	CanEditShow (see page 14820)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 14820)	Indicates whether the grid should respond to a given key combination.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.

	ChangeGridOrientation (see page 14821)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CheckColumnDrag (see page 14821)	CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 14822)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
	Click (see page 5743)	Respond to user click.
	ColumnMoved (see page 14822)	Provides the interface for a method that responds when the position of a column changes.
V	ColWidthsChanged (see page 17569)	Responds when the column widths change.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateEditor (see page 14823)	Creates the inplace editor control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 14823)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
V	DefineProperties (see page 14824)	Adds the ColWidths (see page 14845) and RowHeights (see page 14855) properties to the values saved with the grid.
V	DeleteColumn (see page 14824)	Deletes a specified column from the grid.
V	DeleteRow (see page 14825)	Deletes a specified from the grid.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
V	DoExit (see page 14825)	Hides the inplace editor when selection moves to another control in a form.

	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelDown (see page 14825)	Performs tasks when the mouse wheel is rotated downward.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoMouseWheelUp (see page 14826)	Performs tasks when the mouse wheel is rotated upward.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawCell (see page 17570)	Draws a cell in the grid.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EndColumnDrag (see page 14827)	Ends the dragging of a column in the grid.
	EndRowDrag (see page 14827)	Ends the dragging of a row in the grid.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	FocusCell (see page 14827)	Changes the current cell to a specified cell.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetCells (see page 17570)	Returns the value of a cell in the grid, formatted as a string.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEditLimit (see page 14828)	Provides the interface for a method that limits the number of characters the user can type into cells.
	GetEditMask (see page 14828)	Provides the interface for a method that returns an input mask for the specified cell.
	GetEditStyle (see page 14828)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
	GetEditText (see page 14829)	Returns the value of the indicated cell formatted for editing.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetGridHeight (see page 14829)	Returns the value of the GridHeight (see page 14850) property.
	GetGridWidth (see page 14830)	Returns the value of the GridWidth (see page 14851) property.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HideEdit (see page 14830)	Hides the inplace editor.
	HideEditor (see page 14830)	Hides the inplace editor when it is not in use.
	InvalidateCell (see page 14831)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCol (see page 14831)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	InvalidateEditor (see page 14832)	Causes the inplace editor to repaint itself, moving to a new location if necessary.
	InvalidateGrid (see page 14832)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 14832)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
V	KeyDown (see page 14833)	Provides special processing when the user presses a key.
	KeyPress (see page 6163)	Respond to keyboard input.
V	KeyPress (see page 14833)	Provides special processing when the user presses a key.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 17570)	Starts the drag of a dimension to a new location.
V	MouseMove (see page 17571)	Computes the drop site of a pivoting dimension.
V	MouseUp (see page 17571)	Rearranges the dimensions after a pivot and updates the decision source.
	MoveColRow (see page 14835)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 14836)	Moves a column to a new position.
	MoveRow (see page 14836)	Moves a row to a new position.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 17572)	Renders the image of the decision grid.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.

	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	RowHeightsChanged (see page 17573)	Responds when the row heights change.
	RowMoved (see page 14837)	Provides the interface for a method that responds when the position of a row changes.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	ScrollData (see page 14838)	Scrolls the visual image of the cells in the grid.
	SelectCell (see page 17573)	Indicates whether the user can navigate to a specified cell.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEditText (see page 14839)	Provides the interface for a method that updates the text associated with a cell.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	ShowEditor (see page 14839)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 14839)	Displays the inplace editor, with its text set to the specified character.

	SizeChanged (see page 14840)	Responds to changes in the size of the grid.
	Sizing (see page 14840)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	TimedScroll (see page 14840)	Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.
V	TopLeftChanged (see page 17573)	Generates an OnTopLeftChanged (see page 17586) event.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateDesigner (see page 14841)	Notifies the designer when rows and columns of the grid are moved or resized.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 14843)	Determines whether a single line border is drawn around the grid.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	CaptionColor (see page 17574)	Specifies the background color of cells containing the names of the row and column dimensions.
	CaptionFont (see page 17575)	Specifies the font used to write the names of the row and column dimensions.
	Col (see page 14843)	Specifies the index of the column that contains the selected cell.
R	ColCount (see page 17576)	Indicates the total number of columns in the decision grid.
	Color (see page 5812)	Specifies the background color of the control.
	Color (see page 14844)	Specifies the background color of the control.
	ColWidths (see page 14845)	Indicates the width (in pixels) of all the columns in the grid.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DataColor (see page 17576)	Specifies the background color of the cells containing summarized values.
	DataFont (see page 17577)	Specifies the font used to write the summarized values in the scrollable region of the decision grid.

	DataSumColor (see page 17577)	Specifies the background color of cells that represent subtotals over entire dimensions.
	DefaultColWidth (see page 17578)	Specifies the width of all columns that are not explicitly resized.
	DefaultDrawing (see page 14847)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 17578)	Specifies the height of all rows that are not explicitly resized.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	Dimensions (see page 17579)	Provides access to the collection of TDisplayDim (see page 17661) objects that control the appearance of each dimension.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EditorMode (see page 14848)	Determines whether the current cell can be edited.
	FixedColor (see page 14848)	Specifies the background color of the fixed rows and columns in the grid.
R	FixedCols (see page 17579)	Indicates the number of columns used for dimension captions and dimension value labels.
R	FixedRows (see page 17580)	Indicates the number of rows used for dimension captions and dimension value labels.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
R	GridHeight (see page 14850)	Specifies the height of the grid in pixels.
	GridLineColor (see page 17580)	Specifies the color of the lines that separate the cells in the decision grid.
	GridLineWidth (see page 17580)	Specifies the width, in pixels, of the lines that separate the cells of the grid.
R	GridWidth (see page 14851)	Specifies the width of the grid in pixels.
R	HitTest (see page 14851)	Indicates the current position of the mouse cursor.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
R	InplaceEditor (see page 14852)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	LabelColor (see page 17581)	Specifies the color of the individual row and column labels.
	LabelFont (see page 17581)	Specifies the font used to write the individual row and column labels.
	LabelSumColor (see page 17581)	Specifies the background color for the subtotal labels.
	LeftCol (see page 14852)	Specifies the index of the first visible scrollable column in the grid.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	Options (see page 17582)	Specifies settings that affect the entire grid display.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 14853)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	Row (see page 14854)	Specifies the index of the row that contains the selected cell.
R	RowCount (see page 17582)	Indicates the total number of rows in the decision grid.
	RowHeights (see page 14855)	Gives the height (in pixels) of all rows in the grid.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ScrollBars (see page 14856)	Specifies whether the grid includes horizontal and vertical scroll bars.
	Selection (see page 14856)	Indicates the boundaries of the current selection.
	TabStops (see page 14858)	Indicates whether the user can tab to specified columns in the grid.
	Text (see page 5842)	Contains a text string associated with the control.
	TopRow (see page 14858)	Specifies the index of the first visible scrollable row in the grid.
R	VisibleColCount (see page 14859)	Indicates the number of scrollable columns visible in the grid.

R	VisibleRowCount (see page 14859)	Indicates the number of scrollable rows visible in the grid.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnDecisionDrawCell (see page 17584)	Occurs just before a cell in the grid is drawn.
	OnDecisionExamineCell (see page 17585)	Occurs when the user right-clicks on a cell in the data region of the grid.
	OnTopLeftChanged (see page 17586)	Occurs when the data region of the grid scrolls.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	CellDrawState (see page 17568)	Retrieves information about a data cell in the grid.
	CellRect (see page 17569)	Returns the screen coordinates of a cell in the grid.
	CellValueArray (see page 17569)	Returns the values of all dimension fields over which a data cell is summarized.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 17566)	Creates and initializes an instance of TCustomDecisionGrid (see page 17553).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 17566)	Destroys an instance of TCustomDecisionGrid (see page 17553).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.

	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	MouseCoord (see page 14834)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Notification (see page 17572)	Links the buttons in TDecisionPivot to the decision source.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.

R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Cells (see page 17575)	Provides indexed access to the values of all cells in the decision grid.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DecisionSource (see page 17578)	Specifies the decision source that supplies crosstabulated data to the decision grid.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowCubeEditor (see page 17583)	Determines whether the Cube Editor can be displayed.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.

	Totals (see page 17583)	Indicates whether the grid includes subtotals.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	TabStop (see page 14857)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

109.1.1.2 TCustomDecisionGrid.Create Constructor

Creates and initializes an instance of TCustomDecisionGrid (see page 17553).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomDecisionGrid(TComponent * AOwner);
```

Description

Call Create to instantiate a decision grid at runtime. Decision grids added to forms at design time are created automatically.

AOwner is the form that contains the decision grid. It becomes the value of the Owner property.

See Also

Owner (see page 2807)

109.1.1.3 TCustomDecisionGrid.Destroy Destructor

Destroys an instance of TCustomDecisionGrid (see page 17553).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomDecisionGrid();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the decision grid is not nil, and only then calls Destroy. Destroy frees all helper objects used by TCustomDecisionGrid (see page 17553).

See Also

Free (see page 22300)

109.1.1.4 TCustomDecisionGrid Fields

The fields of the TCustomDecisionGrid class are listed here.

Protected Fields

	Name	Description
	FGridStateX (see page 17567)	This is FGridStateX, a member of class TCustomDecisionGrid.

109.1.1.4.1 TCustomDecisionGrid.FGridStateX Field

Pascal

```
FGridStateX: TDecisionGridState;
```

C++

```
TDecisionGridState FGridStateX;
```

Description

This is FGridStateX, a member of class TCustomDecisionGrid.

109.1.1.5 TCustomDecisionGrid Methods

The methods of the TCustomDecisionGrid class are listed here.

Protected Methods

	Name	Description
	ColWidthsChanged (see page 17569)	Responds when the column widths change.
	DrawCell (see page 17570)	Draws a cell in the grid.
	GetCells (see page 17570)	Returns the value of a cell in the grid, formatted as a string.
	MouseDown (see page 17570)	Starts the drag of a dimension to a new location.
	MouseMove (see page 17571)	Computes the drop site of a pivoting dimension.
	MouseUp (see page 17571)	Rearranges the dimensions after a pivot and updates the decision source.
	Paint (see page 17572)	Renders the image of the decision grid.
	RowHeightsChanged (see page 17573)	Responds when the row heights change.
	SelectCell (see page 17573)	Indicates whether the user can navigate to a specified cell.
	TopLeftChanged (see page 17573)	Generates an OnTopLeftChanged (see page 17586) event.

Public Methods

	Name	Description
	CellDrawState (see page 17568)	Retrieves information about a data cell in the grid.
	CellRect (see page 17569)	Returns the screen coordinates of a cell in the grid.
	CellValueArray (see page 17569)	Returns the values of all dimension fields over which a data cell is summarized.

	Notification (see page 17572)	Links the buttons in TDecisionPivot to the decision source.

109.1.1.5.1 TCustomDecisionGrid.CellDrawState Method

Retrieves information about a data cell in the grid.

Pascal

```
function CellDrawState(ACol: Integer; ARow: Integer; var Value: string; var DrawState:
TDecisionDrawState): boolean;
```

C++

```
__fastcall boolean CellDrawState(int ACol, int ARow, AnsiString Value, TDecisionDrawState
DrawState);
```

Description

Call CellDrawState to obtain information about a cell in the scrollable data region of the grid. The ACol and ARow parameters specify the index of the cell, where (0,0) is the upper left cell in the data region, not the upper left cell in the grid. The Value parameter returns the value of the summary in the cell, formatted for display. The DrawState parameter returns information about the type of information contained in the cell. For more information about the possible values of DrawState, see the entry for the OnDecisionDrawCell (see page 17584) event.

CellDrawState returns true if the Value and DrawState parameters are correctly filled in. CellDrawState returns false if the cell specified by ACol and ARow is not in the data region of the grid.

The ADrawState parameter indicates the type of cell that is about to be drawn. ADrawState includes one or more of the following values:

Value	Meaning
dsGroupStart	The cell is the first row or column label for a dimension. Always appears with dsRowValue or dsColValue.
dsRowCaption	The cell displays a row caption (the name of a dimension whose values are listed down a column on the left of the grid).
dsColCaption	The cell displays a column caption (the name of a dimension whose values are listed along the top of the grid).
dsSum	The cell displays a subtotal label or value. Always appears with dsRowValue, dsColValue, or dsData.
dsRowValue	The cell is a row label. It represents a value from a dimension listed down a column on the left of the grid or a summary label that appears at the end of each group of dimension values.
dsColValue	The cell is a column label. It represents a value from a dimension listed across the top of the grid or a summary label that appears at the end of the dimension.
dsData	The cell is in the scrollable data region of the grid.
dsOpenAfter	The cell has a plus sign (+) on the right that will open following dimension when clicked. Always appears with dsRowCaption, dsColCaption, dsRowPlus, or dsColPlus.
dsCloseAfter	The cell has a minus sign (-) on the right that will close the following dimension when clicked. Always appears with dsRowCaption, dsColCaption, dsRowPlus, or dsColPlus.
dsRowIndicator	The cell appears on the left grid margin that is used to display the open and close indicators (+ and -).
dsColIndicator	The cell appears on the upper grid margin that is used to display the open and close indicators (+ and -).
dsRowPlus	The cell appears in the left margin and contains a plus sign (+) indicating that the adjacent dimension is closed. Always appears with dsRowIndicator.
dsColPlus	The cell appears in the top margin and contains a plus sign (+) indicating that the adjacent dimension is closed. Always appears with dsColIndicator.
dsNone	The cell has no function.

See Also

Cells (see page 17575)

OnDecisionDrawCell (see page 17584)

109.1.1.5.2 TCustomDecisionGrid.CellRect Method

Returns the screen coordinates of a cell in the grid.

Pascal

```
function CellRect(ACol: Longint; ARow: Longint): TRect;
```

C++

```
__fastcall TRect CellRect(Longint ACol, Longint ARow);
```

Description

Call CellRect to obtain the boundaries, in pixels, of the cell in the column and row indicated by the ACol and ARow parameters. The top left cell that contains data is specified when ACol and ARow are set to zero.

If the indicated cell is not visible, CellRect returns an empty rectangle.

See Also

Cells (see page 17575)

109.1.1.5.3 TCustomDecisionGrid.CellValueArray Method

Returns the values of all dimension fields over which a data cell is summarized.

Pascal

```
function CellValueArray(ACol: Integer; ARow: Integer; var ValueArray: TValueArray): boolean;
```

C++

```
__fastcall boolean CellValueArray(int ACol, int ARow, TValueArray ValueArray);
```

Description

Call CellValueArray to obtain the coordinates of a data cell in the multidimensional array maintained by the decision cube. The ACol and ARow parameters represent the indexes of the column and row containing the cell, where (0,0) specifies the upper left data cell, not the top left cell in the grid. The ValueArray parameter returns the indexes of the field values for each dimension over which the cell summarizes data. Use the GetMemberAsString or GetMemberAsVariant method of the decision source to convert the indexes in the ValueArray parameter into field values. CellValueArray returns a boolean indicating whether the function was successful.

See Also

GetMemberAsString (see page 17477)

GetMemberAsVariant (see page 17478)

TValueArray

TSmallIntArray (see page 17361)

109.1.1.5.4 TCustomDecisionGrid.ColWidthsChanged Method

Responds when the column widths change.

Pascal

```
procedure ColWidthsChanged; override;
```

C++

```
virtual __fastcall ColWidthsChanged();
```

Description

ColWidthsChanged is called immediately after the column widths change.

ColWidthsChanged adjusts the scroll bar to accommodate the change. At design time, ColWidthsChanged forces all columns to have the same size when the columns are resized using the mouse.

See Also

ScrollBars (see page 14856)

109.1.1.5.5 TCustomDecisionGrid.DrawCell Method

Draws a cell in the grid.

Pascal

```
procedure DrawCell(ACol: Longint; ARow: Longint; ARect: TRect; AState: TGridDrawState);
override;
```

C++

```
virtual __fastcall DrawCell(Longint ACol, Longint ARow, TRect ARect, TGridDrawState AState);
```

Description

DrawCell is called automatically when the cell specified by the ACol and ARow parameters needs to be painted. DrawCell generates an OnDecisionDrawCell (see page 17584) event, and then draws the cell using the color, font, and value for the cell.

See Also

OnDecisionDrawCell (see page 17584)

109.1.1.5.6 TCustomDecisionGrid.GetCells Method

Returns the value of a cell in the grid, formatted as a string.

Pascal

```
function GetCells(ACol: Integer; ARow: Integer): String;
```

C++

```
__fastcall AnsiString GetCells(int ACol, int ARow);
```

Description

GetCells is the protected property access method for the Cells (see page 17575) property. Override GetCells to change the implementation of the Cells (see page 17575) property.

See Also

Cells (see page 17575)

109.1.1.5.7 TCustomDecisionGrid.MouseDown Method

Starts the drag of a dimension to a new location.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

MouseDown is called automatically when the user clicks the mouse in a cell of the grid. After calling the inherited method, MouseDown starts a pivot operation if the cell is a dimension label that can be dragged.

When overriding the MouseDown method, be sure to call the inherited method.

See Also

MouseMove (see page 17571)

MouseUp (see page 17571)

MouseDown (see page 14834)

109.1.1.5.8 TCustomDecisionGrid.MouseMove Method

Computes the drop site of a pivoting dimension.

Pascal

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseMove(TShiftState Shift, int X, int Y);
```

Description

MouseMove is called automatically when the user moves the mouse. After calling the inherited method, MouseMove calculates the new pivot state if a dimension is being dragged to a new location.

When overriding the MouseMove method, be sure to call the inherited method.

See Also

MouseDown (see page 17570)

MouseUp (see page 17571)

MouseMove (see page 14834)

109.1.1.5.9 TCustomDecisionGrid.MouseUp Method

Rearranges the dimensions after a pivot and updates the decision source.

Pascal

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

C++

```
virtual __fastcall MouseUp(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

MouseUp is called automatically when the user releases the mouse. After calling the inherited method, MouseUp checks whether a dimension was just dragged to a new location. If a dimension was just moved, MouseUp changes the Dimensions (

see page 17579) property appropriately and updates the decision source to reflect the new configuration.

When overriding the MouseUp method, be sure to call the inherited method.

See Also

Dimensions (see page 17579)

MouseDown (see page 17570)

MouseMove (see page 17571)

MouseUp (see page 14835)

109.1.1.5.10 TCustomDecisionGrid.Notification Method

Links the buttons in TDecisionPivot to the decision source.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

After calling the inherited method, Notification performs the necessary internal adjustments when new elements are added to the decision grid. When overriding the Notification method, be sure to call the inherited method.

See Also

DecisionSource (see page 17578)

TDecisionPivot (see page 17680)

109.1.1.5.11 TCustomDecisionGrid.Paint Method

Renders the image of the decision grid.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Paint is called when the grid receives a Windows paint message. Paint draws the grid, calling the DrawCell (see page 17570) method when it is time to paint the contents of each cell. After all the cells are drawn, Paint draws the grid lines.

If DefaultDrawing is false, Paint does not draw the background of the cells or draw the focus rectangle around the selected cell when it has focus.

See Also

DrawCell (see page 17570)

DefaultDrawing (see page 14847)

109.1.1.5.12 TCustomDecisionGrid.RowHeightsChanged Method

Responds when the row heights change.

Pascal

```
procedure RowHeightsChanged; override;
```

C++

```
virtual __fastcall RowHeightsChanged();
```

Description

RowHeightsChanged is called immediately after the row heights change.

RowHeightsChanged adjusts the scroll bar to accommodate the change. At design time, RowHeightsChanged forces all rows to have the same size when the rows are resized using the mouse.

See Also

ScrollBars (see page 14856)

109.1.1.5.13 TCustomDecisionGrid.SelectCell Method

Indicates whether the user can navigate to a specified cell.

Pascal

```
function SelectCell(Arow: LongInt; AColumn: LongInt): Boolean; override;
```

C++

```
virtual __fastcall Boolean SelectCell(LongInt Arow, LongInt AColumn);
```

Description

Decision grids call SelectCell internally before attempting to move selection to the cell identified by the ACol and ARow parameters. ACol and ARow are the column and row indexes for the cell, where the first column has index 0, and the top row has index 0.

As implemented in TCustomDecisionGrid (see page 17553), SelectCell always returns True, indicating that focus can move to the indicated cell.

See Also

Row (see page 14854)

Col (see page 14843)

OnDecisionExamineCell (see page 17585)

109.1.1.5.14 TCustomDecisionGrid.TopLeftChanged Method

Generates an OnTopLeftChanged (see page 17586) event.

Pascal

```
procedure TopLeftChanged; override;
```

C++

```
virtual __fastcall TopLeftChanged();
```

Description

Decision grids call TopLeftChanged internally, whenever the data region of the grid scrolls.

See Also

OnTopLeftChanged (see page 17586)

109.1.1.6 TCustomDecisionGrid Properties

The properties of the TCustomDecisionGrid class are listed here.

Protected Properties

	Name	Description
	CaptionColor (see page 17574)	Specifies the background color of cells containing the names of the row and column dimensions.
	CaptionFont (see page 17575)	Specifies the font used to write the names of the row and column dimensions.
R	ColCount (see page 17576)	Indicates the total number of columns in the decision grid.
	DataColor (see page 17576)	Specifies the background color of the cells containing summarized values.
	DataFont (see page 17577)	Specifies the font used to write the summarized values in the scrollable region of the decision grid.
	DataSumColor (see page 17577)	Specifies the background color of cells that represent subtotals over entire dimensions.
	DefaultColWidth (see page 17578)	Specifies the width of all columns that are not explicitly resized.
	DefaultRowHeight (see page 17578)	Specifies the height of all rows that are not explicitly resized.
	Dimensions (see page 17579)	Provides access to the collection of TDisplayDim (see page 17661) objects that control the appearance of each dimension.
R	FixedCols (see page 17579)	Indicates the number of columns used for dimension captions and dimension value labels.
R	FixedRows (see page 17580)	Indicates the number of rows used for dimension captions and dimension value labels.
	GridLineColor (see page 17580)	Specifies the color of the lines that separate the cells in the decision grid.
	GridLineWidth (see page 17580)	Specifies the width, in pixels, of the lines that separate the cells of the grid.
	LabelColor (see page 17581)	Specifies the color of the individual row and column labels.
	LabelFont (see page 17581)	Specifies the font used to write the individual row and column labels.
	LabelSumColor (see page 17581)	Specifies the background color for the subtotal labels.
	Options (see page 17582)	Specifies settings that affect the entire grid display.
R	RowCount (see page 17582)	Indicates the total number of rows in the decision grid.

Public Properties

	Name	Description
R	Cells (see page 17575)	Provides indexed access to the values of all cells in the decision grid.
	DecisionSource (see page 17578)	Specifies the decision source that supplies crosstabulated data to the decision grid.
	ShowCubeEditor (see page 17583)	Determines whether the Cube Editor can be displayed.
	Totals (see page 17583)	Indicates whether the grid includes subtotals.

109.1.1.6.1 TCustomDecisionGrid.CaptionColor Property

Specifies the background color of cells containing the names of the row and column dimensions.

Pascal

```
property CaptionColor: TColor;
```

C++

```
__property TColor CaptionColor;
```

Description

Set CaptionColor to the color that should be used to make the names of the row and column dimensions stand out. These are the names for the dimensions that are set in the Decision Cube Editor.

The actual values of the fields that make up the row and column dimensions are drawn using the LabelColor (see page 17581) property.

See Also

CaptionFont (see page 17575)

DataColor (see page 17576)

DataSumColor (see page 17577)

Dimensions (see page 17579)

LabelColor (see page 17581)

LabelSumColor (see page 17581)

109.1.1.6.2 TCustomDecisionGrid.CaptionFont Property

Specifies the font used to write the names of the row and column dimensions.

Pascal

```
property CaptionFont: TFont;
```

C++

```
__property TFont CaptionFont;
```

Description

Set CaptionFont to the font that should be used to make the names of the row and column dimensions stand out. These are the names for the dimensions that are set in the Decision Cube Editor.

The actual values of the fields that make up the row and column dimensions are written using the LabelFont (see page 17581) property.

See Also

CaptionColor (see page 17574)

DataFont (see page 17577)

Dimensions (see page 17579)

LabelFont (see page 17581)

TFont (see page 14497)

109.1.1.6.3 TCustomDecisionGrid.Cells Property

Provides indexed access to the values of all cells in the decision grid.

Pascal

```
property Cells [ACol, ARow: Integer]: String;
```

C++

```
__property AnsiString Cells[int ACol, ARow];
```

Description

Use Cells to obtain the value of a cell in the grid, formatted as a string. The ACol and ARow parameters indicate the column and row indexes of the cell, where (0,0) indicates the top left cell of the data region (not the top left cell of the grid). Cells that have been closed up using the Display Data or Display SubTotals menu items are still counted even though they are not visible.

If the cell is a row or column caption (negative value for ACol or ARow), Cells returns the name of the dimension. If the cell is a row or column label, Cells returns the field value of the dimension field. If the cell is a subtotal label, Cells returns the string Sum. If the cell is in the data region of the grid, Cells returns the summarized value.

See Also

CellDrawState (see page 17568)

CellValueArray (see page 17569)

109.1.1.6.4 TCustomDecisionGrid.ColCount Property

Indicates the total number of columns in the decision grid.

Pascal

```
property ColCount: integer;
```

C++

```
__property int ColCount;
```

Description

Read ColCount to determine the number of columns in the decision grid, including the columns containing dimension captions and label cells. Compute the number of columns in the data region of the decision grid by subtracting the value of FixedCols (see page 17579) from ColCount.

See Also

FixedCols (see page 17579)

109.1.1.6.5 TCustomDecisionGrid.DataColor Property

Specifies the background color of the cells containing summarized values.

Pascal

```
property DataColor: TColor;
```

C++

```
__property TColor DataColor;
```

Description

Set DataColor to the color that should be used for the scrollable data region of the decision grid. The data region contains the values of the current summary of the decision cube, where each cell represents the value of a field summarized over the values indicated by the row and column dimensions.

The subtotals that represent values summarized over all values within a given dimension (as well as the single value in other dimensions) are drawn using the DataSumColor (see page 17577) property.

See Also

CaptionColor (see page 17574)

DataFont (see page 17577)

DataSumColor (see page 17577)

LabelColor (see page 17581)

LabelSumColor (see page 17581)

109.1.1.6.6 TCustomDecisionGrid.DataFont Property

Specifies the font used to write the summarized values in the scrollable region of the decision grid.

Pascal

```
property DataFont: TFont;
```

C++

```
__property TFont DataFont;
```

Description

Set DataFont to the font that should be used for the data region of the decision grid. The data region contains the values of the current summary of the decision cube, where each cell represents the value of a field summarized over the values indicated by the row and column dimensions. It also includes the dimension subtotals over the summary field.

See Also

CaptionFont (see page 17575)

DataColor (see page 17576)

LabelFont (see page 17581)

109.1.1.6.7 TCustomDecisionGrid.DataSumColor Property

Specifies the background color of cells that represent subtotals over entire dimensions.

Pascal

```
property DataSumColor: TColor;
```

C++

```
__property TColor DataSumColor;
```

Description

Set DataSumColor to make the subtotals that represent an entire dimension stand out from the summary values that summarize information over a single value in each dimension. The subtotal cells are placed in the scrollable data region at the ends of the set of rows or columns that correspond to a given dimension.

See Also

CaptionColor (see page 17574)

DataColor (see page 17576)

DataFont (see page 17577)

LabelColor (see page 17581)

LabelSumColor (see page 17581)

Totals (see page 17583)

109.1.1.6.8 TCustomDecisionGrid.DecisionSource Property

Specifies the decision source that supplies crosstabulated data to the decision grid.

Pascal

```
property DecisionSource: TDecisionSource;
```

C++

```
__property TDecisionSource DecisionSource;
```

Description

Set DecisionSource to link the decision grid to the crosstabulated data from a decision cube. The DecisionSource object maintains information about the current pivot state of the data. Thus, any two decision controls that share the same decision source will display the data from the decision cube using the same configuration of row and column dimensions, including any closed dimensions or drilled down values.

See Also

TDecisionCube (see page 17442)

TDecisionSource (see page 17462)

109.1.1.6.9 TCustomDecisionGrid.DefaultColWidth Property

Specifies the width of all columns that are not explicitly resized.

Pascal

```
property DefaultColWidth: Integer;
```

C++

```
__property int DefaultColWidth;
```

Description

Set DefaultColWidth to the desired width of the columns in the decision grid that are not resized using the mouse. Setting DefaultColWidth changes the size of all columns in the grid, including columns that have been resized.

See Also

DefaultRowHeight (see page 17578)

109.1.1.6.10 TCustomDecisionGrid.DefaultRowHeight Property

Specifies the height of all rows that are not explicitly resized.

Pascal

```
property DefaultRowHeight: Integer;
```

C++

```
__property int DefaultRowHeight;
```

Description

Set DefaultRowHeight to the desired height of the rows in the decision grid that are not resized using the mouse. Setting DefaultRowHeight changes the size of all rows in the grid, including rows that have been resized.

See Also

DefaultColWidth (see page 17578)

109.1.1.6.11 TCustomDecisionGrid.Dimensions Property

Provides access to the collection of TDisplayDim (see page 17661) objects that control the appearance of each dimension.

Pascal

```
property Dimensions: TDisplayDims;
```

C++

```
__property TDisplayDims Dimensions;
```

Description

Set the properties of the individual TDisplayDim (see page 17661) objects to control the appearance of individual dimensions within the grid. The TDisplayDim (see page 17661) objects have properties that can override the various color and font properties of the grid to make individual dimensions stand out. They also allow the dimension names, alignments, and formats to be customized, and permit the dimension subtotals for a dimension to be turned on or off.

Note: Do not change the Dimensions property, only the properties of individual TDisplayDim (see page 17661) objects accessed through the Dimensions property. To programmatically change the configuration of dimensions that appears in the grid, use the decision source object.

See Also

CaptionColor (see page 17574)

CaptionFont (see page 17575)

DataColor (see page 17576)

DataFont (see page 17577)

DataSumColor (see page 17577)

LabelColor (see page 17581)

LabelFont (see page 17581)

LabelSumColor (see page 17581)

TDecisionSource (see page 17462)

TDisplayDim (see page 17661)

109.1.1.6.12 TCustomDecisionGrid.FixedCols Property

Indicates the number of columns used for dimension captions and dimension value labels.

Pascal

```
property FixedCols: integer;
```

C++

```
__property int FixedCols;
```

Description

Read FixedCols to determine the number of columns in the decision grid that are not part of the data region. Subtract FixedCols from the value of the ColCount (see page 17576) property to obtain the number of columns in the data region.

See Also

ColCount (see page 17576)

109.1.1.6.13 TCustomDecisionGrid.FixedRows Property

Indicates the number of rows used for dimension captions and dimension value labels.

Pascal

```
property FixedRows: integer;
```

C++

```
__property int FixedRows;
```

Description

Read FixedRows to determine the number of rows in the decision grid that are not part of the data region. Subtract FixedRows from the value of the RowCount (see page 17582) property to obtain the number of rows in the data region.

See Also

RowCount (see page 17582)

109.1.1.6.14 TCustomDecisionGrid.GridLineColor Property

Specifies the color of the lines that separate the cells in the decision grid.

Pascal

```
property GridLineColor: TColor;
```

C++

```
__property TColor GridLineColor;
```

Description

Set GridLineColor to specify the color of the lines drawn between the cells of the grid. The grid lines only appear when the Options (see page 17582) property includes cgGridLines.

See Also

GridLineWidth (see page 17580)

Options (see page 17582)

109.1.1.6.15 TCustomDecisionGrid.GridLineWidth Property

Specifies the width, in pixels, of the lines that separate the cells of the grid.

Pascal

```
property GridLineWidth: Integer;
```

C++

```
__property int GridLineWidth;
```

Description

Set GridLineWidth to make the lines that separate the cells in the decision grid heavier or lighter. The grid lines only appear when the Options (see page 17582) property includes cgGridLines.

See Also

GridLineColor (see page 17580)

Options (see page 17582)

109.1.1.6.16 TCustomDecisionGrid.LabelColor Property

Specifies the color of the individual row and column labels.

Pascal

```
property LabelColor: TColor;
```

C++

```
__property TColor LabelColor;
```

Description

Set LabelColor to the color that should be used to make the row and column labels stand out. The row and column labels of the decision grid contain the values of the fields that correspond to the dimensions of the decision cube. These are the values over which the data is summarized.

See Also

CaptionColor (see page 17574)

DataColor (see page 17576)

DataSumColor (see page 17577)

LabelFont (see page 17581)

LabelSumColor (see page 17581)

109.1.1.6.17 TCustomDecisionGrid.LabelFont Property

Specifies the font used to write the individual row and column labels.

Pascal

```
property LabelFont: TFont;
```

C++

```
__property TFont LabelFont;
```

Description

Set LabelFont to the font that should be used to write the labels that display the values of the dimensions over which data is summarized. The row and column labels of the decision grid contain the values of the fields that correspond to the dimensions of the decision cube. LabelFont is also used when writing the labels for rows or columns that represent dimension subtotals.

See Also

CaptionFont (see page 17575)

DataFont (see page 17577)

LabelColor (see page 17581)

LabelSumColor (see page 17581)

109.1.1.6.18 TCustomDecisionGrid.LabelSumColor Property

Specifies the background color for the subtotal labels.

Pascal

```
property LabelSumColor: TColor;
```

C++

```
__property TColor LabelSumColor;
```

Description

Set LabelSumColor make the subtotal labels stand out from the labels that represent individual field values along a dimension. The subtotal labels appear at the end of the set of field value labels for an individual dimension.

See Also

CaptionColor (see page 17574)

DataColor (see page 17576)

DataSumColor (see page 17577)

LabelColor (see page 17581)

LabelFont (see page 17581)

Totals (see page 17583)

109.1.1.6.19 TCustomDecisionGrid.Options Property

Specifies settings that affect the entire grid display.

Pascal

```
property Options: TDecisionGridOptions;
```

C++

```
__property TDecisionGridOptions Options;
```

Description

Set Options to include the desired properties for the grid. Options is a set drawn from the following values:

Value	Meaning
cgGridLines	Vertical and horizontal lines are drawn to separate the cells in the scrollable data region.
cgOutliner	The dimension caption cells contain plus (+) and minus (-) signs that open or close dimensions in the grid when clicked. The minus sign closes (hides) an adjacent dimension. The plus sign opens (shows) an adjacent dimension.
cgPivotable	Dimensions (see page 17579) can be rearranged by clicking on the dimension caption cells and dragging them to a new position.

Note: If Options does not include cgOutliner, individual dimensions can still be opened and closed using the decision pivot. Similarly, if Options does not include cgPivotable, individual dimensions can still be opened and closed with the decision pivot.

See Also

GridLineColor (see page 17580)

GridLineWidth (see page 17580)

TDecisionPivot (see page 17680)

109.1.1.6.20 TCustomDecisionGrid.RowCount Property

Indicates the total number of rows in the decision grid.

Pascal

```
property RowCount: integer;
```

C++

```
__property int RowCount;
```

Description

Read RowCount to determine the number of rows in the decision grid, including the rows containing dimension captions and label cells. Compute the number of rows in the data region of the decision grid by subtracting the value of FixedRows (see page 17580) from RowCount.

See Also

FixedRows (see page 17580)

109.1.1.6.21 TCustomDecisionGrid.ShowCubeEditor Property

Determines whether the Cube Editor can be displayed.

Pascal

```
property ShowCubeEditor: boolean;
```

C++

```
__property boolean ShowCubeEditor;
```

Description

ShowCubeEditor determines whether the Cube Editor for the associated TDecisionCube can be invoked. If set to true, a menu item to invoke the Cube Editor is added to the speed menu displayed on a right-click on the decision grid at runtime.

See Also

Dimensions (see page 17579)

ShowCubeDialog (see page 17450)

TDecisionCube (see page 17442)

109.1.1.6.22 TCustomDecisionGrid.Totals Property

Indicates whether the grid includes subtotals.

Pascal

```
property Totals: boolean;
```

C++

```
__property boolean Totals;
```

Description

Set Totals to turn row and column subtotals on or off for the whole grid. To turn subtotals on or off for individual dimensions, use the Dimensions (see page 17579) property.

See Also

Dimensions (see page 17579)

109.1.1.7 TCustomDecisionGrid Events

The events of the TCustomDecisionGrid class are listed here.

Public Events

	Name	Description
	OnDecisionDrawCell (see page 17584)	Occurs just before a cell in the grid is drawn.
	OnDecisionExamineCell (see page 17585)	Occurs when the user right-clicks on a cell in the data region of the grid.
	OnTopLeftChanged (see page 17586)	Occurs when the data region of the grid scrolls.

109.1.1.7.1 TCustomDecisionGrid.OnDecisionDrawCell Event

Occurs just before a cell in the grid is drawn.

Pascal

```
property OnDecisionDrawCell: TDecisionDrawCellEvent;
```

C++

```
__property TDecisionDrawCellEvent OnDecisionDrawCell;
```

Description

Write an OnDecisionDrawCell event handler to change the color, font, or text used to display an individual cell in the grid. Change the AColor, AFont, or Value parameter to change the color, font, or text respectively.

For example, to change the row and column labels from the values of the fields in the dataset to customized labels, change the Value parameter of the event handler when the Row and Column parameters represent a label cell.

The Row and Column parameters indicate the row and column of a cell within the grid, where (0,0) specifies the upper left summary value, not the upper left corner of the grid. Thus, the label cells contain negative row or column indexes. In addition, the AState parameter includes gdFixed when the cell is a label cell. AState also indicates whether the cell being drawn is selected.

The ADrawState parameter indicates the type of cell that is about to be drawn. ADrawState includes one or more of the following values:

Value	Meaning
dsGroupStart	The cell is the first row or column label for a dimension. Always appears with dsRowValue or dsColValue.
dsRowCaption	The cell displays a row caption (the name of a dimension whose values are listed down a column on the left of the grid).
dsColCaption	The cell displays a column caption (the name of a dimension whose values are listed along the top of the grid).
dsSum	The cell displays a subtotal label or value. Always appears with dsRowValue, dsColValue, or dsData.
dsRowValue	The cell is a row label. It represents a value from a dimension listed down a column on the left of the grid or a summary label that appears at the end of each group of dimension values.
dsColValue	The cell is a column label. It represents a value from a dimension listed across the top of the grid or a summary label that appears at the end of the dimension.
dsData	The cell is in the scrollable data region of the grid.
dsOpenAfter	The cell has a plus sign (+) on the right that will open following dimension when clicked. Always appears with dsRowCaption, dsColCaption, dsRowPlus, or dsColPlus.
dsCloseAfter	The cell has a minus sign (-) on the right that will close the following dimension when clicked. Always appears with dsRowCaption, dsColCaption, dsRowPlus, or dsColPlus.

dsRowIndicator	The cell appears on the left grid margin that is used to display the open and close indicators (+ and -).
dsColIndicator	The cell appears on the upper grid margin that is used to display the open and close indicators (+ and -).
dsRowPlus	The cell appears in the left margin and contains a plus sign (+) indicating that the adjacent dimension is closed. Always appears with dsRowIndicator.
dsColPlus	The cell appears in the top margin and contains a plus sign (+) indicating that the adjacent dimension is closed. Always appears with dsColIndicator.
dsNone	The cell has no function.

See Also

CaptionColor (see page 17574)

CaptionFont (see page 17575)

DataColor (see page 17576)

DataFont (see page 17577)

DataSumColor (see page 17577)

LabelColor (see page 17581)

LabelFont (see page 17581)

LabelSumColor (see page 17581)

109.1.1.7.2 TCustomDecisionGrid.OnDecisionExamineCell Event

Occurs when the user right-clicks on a cell in the data region of the grid.

Pascal

```
property OnDecisionExamineCell: TDecisionExamineCellEvent;
```

C++

```
__property TDecisionExamineCellEvent OnDecisionExamineCell;
```

Description

Write an OnDecisionExamineCell event handler to take specific action when the user right-clicks on one of the cells that contains summarized data. The ICol and IRow parameters indicate the column and row indexes of the cell. The column and row indexes specify the value 0 for the top or leftmost cell in the scrollable data region, not the top left cell of the decision grid. The ISum parameter is the index of the current summary in the decision cube, where the first active summary has an index of 0. The ValueArray parameter gives the coordinates of the cell in terms of the dimensions of the decision source. If the cell represents a subtotal value, the entry in the ValueArray for the dimension over which the subtotal is calculated is a negative number.

Use the Cells (see page 17575) property to obtain the value of the summary in the cell. Use the GetDimensionName method of the decision source to get the name of each dimension represented by an entry in the ValueArray parameter. Use the GetValueAsString or GetValueAsVariant method of the decision source to obtain the label values for each dimension represented in the ValueArray parameter.

See Also

Cells (see page 17575)

DecisionSource (see page 17578)

GetValueArray (see page 17480)

GetDimensionName (see page 17474)

GetDataAsString (see page 17473)

GetDataAsVariant (see page 17473)

TSmallIntArray (see page 17361)

109.1.1.7.3 TCustomDecisionGrid.OnTopLeftChanged Event

Occurs when the data region of the grid scrolls.

Pascal

```
property OnTopLeftChanged: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnTopLeftChanged;
```

Description

Write an OnTopLeftChanged event handler to take specific action when the summarized data in the grid scrolls. Use the Cells (see page 17575) property or the CellDrawState (see page 17568) method to obtain the new values of the data cells.

See Also

Cells (see page 17575)

CellDrawState (see page 17568)

CellValueArray (see page 17569)

109.1.2 TDecisionGrid Class

TDecisionGrid presents crosstabulated data from a decision source in a grid format.

Class Hierarchy

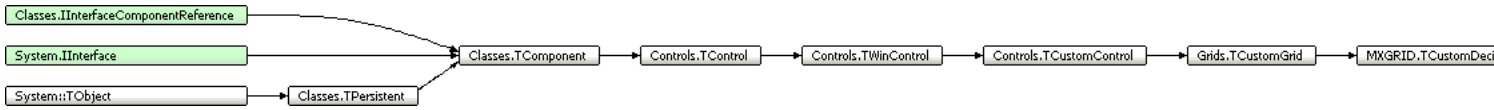

Pascal

```
TDecisionGrid = class(TCustomDecisionGrid);
```

C++

```
class TDecisionGrid : public TCustomDecisionGrid;
```

File

MXGRID

Description

Use TDecisionGrid to put a decision grid in a form. Decision grids present a single summary that is calculated over an arbitrary number of dimensions. The dimensions can appear as rows or columns in the grid, with the field values of the dimensions listed as labels along the top and left edges of the grid.

The field that is summarized and the fields (dimensions) over which it is summarized can be changed dynamically at runtime to view any configuration of summary and dimensions chosen from the dimension map of a decision cube. Changes to the dimensions and summary can be initiated by direct manipulation of the decision grid, or by using a TDecisionPivot object.

TDecisionGrid publishes many of the properties, events, and methods inherited from TCustomDecisionGrid (see page 17553), but introduces no new behavior.

To present the crosstabulated data in a chart, use TDecisionGraph instead.

See Also

TDecisionCube (see page 17442)

TDecisionGraph (see page 17521)

TDecisionPivot (see page 17680)

TDecisionSource (see page 17462)

Creating Decision Grids

Overview of Decision Support Components

Decision Grid Properties

Using Decision Pivots

Using Decision Grids

Creating and Using Decision Grids

About Crosstabs

109.1.2.1 TDecisionGrid Members

The following tables list the members exposed by TDecisionGrid.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 14814)	This is DesignOptionsBoost, a member of class TCustomGrid.
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FGridState (see page 14814)	This is FGridState, a member of class TCustomGrid.
	FGridStateX (see page 17567)	This is FGridStateX, a member of class TCustomDecisionGrid.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveCellExtents (see page 14814)	This is FSaveCellExtents, a member of class TCustomGrid.
	VirtualView (see page 14814)	This is VirtualView, a member of class TCustomGrid.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AdjustSize (see page 14816)	Adds or deletes rows or columns from the grid.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	BeginColumnDrag (see page 14817)	Starts the dragging of a column in the grid.
	BeginRowDrag (see page 14817)	Starts the dragging of a row in the grid.
	BoxRect (see page 14818)	Returns the screen coordinates of a specified set of cells.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcDrawInfo (see page 14818)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 14818)	Calculates the pixel locations of the fixed cells.
V	CalcSizingState (see page 14819)	Determines the appearance of the mouse cursor during resizing.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanEditAcceptKey (see page 14819)	Indicates whether the Key parameter can be entered into the current cell.
	CanEditModify (see page 14819)	Indicates whether the current cell in the grid can be edited.
V	CanEditShow (see page 14820)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 14820)	Indicates whether the grid should respond to a given key combination.

	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	CellRect (see page 14821)	Returns the screen coordinates of a cell in the grid.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeGridOrientation (see page 14821)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CheckColumnDrag (see page 14821)	CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 14822)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
	Click (see page 5743)	Respond to user click.
	ColumnMoved (see page 14822)	Provides the interface for a method that responds when the position of a column changes.
	ColWidthsChanged (see page 14822)	Responds when the column widths change.
	ColWidthsChanged (see page 17569)	Responds when the column widths change.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateEditor (see page 14823)	Creates the inplace editor control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 14823)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DefineProperties (see page 14824)	Adds the ColWidths (see page 14845) and RowHeights (see page 14855) properties to the values saved with the grid.
	DeleteColumn (see page 14824)	Deletes a specified column from the grid.
	DeleteRow (see page 14825)	Deletes a specified from the grid.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.

	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
V	DoExit (see page 14825)	Hides the inplace editor when selection moves to another control in a form.
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
V	DoMouseWheelDown (see page 14825)	Performs tasks when the mouse wheel is rotated downward.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
V	DoMouseWheelUp (see page 14826)	Performs tasks when the mouse wheel is rotated upward.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
A	DrawCell (see page 14826)	Provides the prototype for a method that draws the contents of a cell in the grid.
V	DrawCell (see page 17570)	Draws a cell in the grid.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EndColumnDrag (see page 14827)	Ends the dragging of a column in the grid.
	EndRowDrag (see page 14827)	Ends the dragging of a row in the grid.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	FocusCell (see page 14827)	Changes the current cell to a specified cell.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetCells (see page 17570)	Returns the value of a cell in the grid, formatted as a string.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEditLimit (see page 14828)	Provides the interface for a method that limits the number of characters the user can type into cells.
	GetEditMask (see page 14828)	Provides the interface for a method that returns an input mask for the specified cell.
	GetEditStyle (see page 14828)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.

	GetEditText (see page 14829)	Returns the value of the indicated cell formatted for editing.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetGridHeight (see page 14829)	Returns the value of the GridHeight (see page 14850) property.
	GetGridWidth (see page 14830)	Returns the value of the GridWidth (see page 14851) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HideEdit (see page 14830)	Hides the inplace editor.
	HideEditor (see page 14830)	Hides the inplace editor when it is not in use.
	InvalidateCell (see page 14831)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCol (see page 14831)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	InvalidateEditor (see page 14832)	Causes the inplace editor to repaint itself, moving to a new location if necessary.
	InvalidateGrid (see page 14832)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 14832)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyDown (see page 14833)	Provides special processing when the user presses a key.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyPress (see page 14833)	Provides special processing when the user presses a key.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseDown (see page 14834)	Provides special processing when the user presses a mouse button.
	MouseDown (see page 17570)	Starts the drag of a dimension to a new location.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseMove (see page 14834)	Provides special processing when the user moves the mouse.
	MouseMove (see page 17571)	Computes the drop site of a pivoting dimension.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	MouseUp (see page 14835)	Provides special processing when the user releases the mouse button.
	MouseUp (see page 17571)	Rearranges the dimensions after a pivot and updates the decision source.
	MoveColRow (see page 14835)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 14836)	Moves a column to a new position.

	MoveRow (see page 14836)	Moves a row to a new position.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 5920)	Renders the image of a custom control.
V	Paint (see page 14836)	Draws the image of the grid control on the screen.
V	Paint (see page 17572)	Renders the image of the decision grid.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	RowHeightsChanged (see page 14837)	Responds when the row heights change.
V	RowHeightsChanged (see page 17573)	Responds when the row heights change.
	RowMoved (see page 14837)	Provides the interface for a method that responds when the position of a row changes.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	ScrollData (see page 14838)	Scrolls the visual image of the cells in the grid.
V	SelectCell (see page 14838)	Determines whether a particular cell in the grid can be selected.
V	SelectCell (see page 17573)	Indicates whether the user can navigate to a specified cell.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEditText (see page 14839)	Provides the interface for a method that updates the text associated with a cell.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.

	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	ShowEditor (see page 14839)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 14839)	Displays the inplace editor, with its text set to the specified character.
	SizeChanged (see page 14840)	Responds to changes in the size of the grid.
	Sizing (see page 14840)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	TimedScroll (see page 14840)	Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.
	TopLeftChanged (see page 14841)	Updates the position of the inplace editor when the grid scrolls.
V	TopLeftChanged (see page 17573)	Generates an OnTopLeftChanged (see page 17586) event.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateDesigner (see page 14841)	Notifies the designer when rows and columns of the grid are moved or resized.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.

	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Col (see page 14843)	Specifies the index of the column that contains the selected cell.
	ColWidths (see page 14845)	Indicates the width (in pixels) of all the columns in the grid.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	EditorMode (see page 14848)	Determines whether the current cell can be edited.
	FixedColor (see page 14848)	Specifies the background color of the fixed rows and columns in the grid.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
R	GridHeight (see page 14850)	Specifies the height of the grid in pixels.
R	GridWidth (see page 14851)	Specifies the width of the grid in pixels.
R	HitTest (see page 14851)	Indicates the current position of the mouse cursor.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
R	InplaceEditor (see page 14852)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	LeftCol (see page 14852)	Specifies the index of the first visible scrollable column in the grid.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	Row (see page 14854)	Specifies the index of the row that contains the selected cell.
	RowHeights (see page 14855)	Gives the height (in pixels) of all rows in the grid.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Selection (see page 14856)	Indicates the boundaries of the current selection.
	TabStops (see page 14858)	Indicates whether the user can tab to specified columns in the grid.
	Text (see page 5842)	Contains a text string associated with the control.
	TopRow (see page 14858)	Specifies the index of the first visible scrollable row in the grid.
R	VisibleColCount (see page 14859)	Indicates the number of scrollable columns visible in the grid.
R	VisibleRowCount (see page 14859)	Indicates the number of scrollable rows visible in the grid.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.

	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	CellDrawState (see page 17568)	Retrieves information about a data cell in the grid.
	CellRect (see page 17569)	Returns the screen coordinates of a cell in the grid.
	CellValueArray (see page 17569)	Returns the values of all dimension fields over which a data cell is summarized.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 5918)	Creates an instance of TCustomControl (see page 5908).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 14813)	Creates an instance of TCustomGrid (see page 14800).
V	Create (see page 17566)	Creates and initializes an instance of TCustomDecisionGrid (see page 17553).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 14813)	Destroys an instance of TCustomGrid (see page 14800).
V	Destroy (see page 17566)	Destroys an instance of TCustomDecisionGrid (see page 17553).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseCoord (see page 14834)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Notification (see page 17572)	Links the buttons in TDecisionPivot to the decision source.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Cells (see page 17575)	Provides indexed access to the values of all cells in the decision grid.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	ColCount (see page 17606)	Indicates the total number of columns in the decision grid.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
	FixedCols (see page 17615)	Indicates the number of columns used for dimension captions and dimension value labels.
	FixedRows (see page 17615)	Indicates the number of rows used for dimension captions and dimension value labels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	RowCount (see page 17624)	Indicates the total number of rows in the decision grid.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 17630)	Occurs when the user clicks the control.
	OnDblClick (see page 17632)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDecisionDrawCell (see page 17632)	Occurs just before a cell in the grid is drawn.
	OnDecisionExamineCell (see page 17634)	Occurs when the user right-clicks on a cell in the data region of the grid.
	OnDragDrop (see page 17634)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 17637)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 17639)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

	OnEnter (see page 17640)	Occurs when a control receives the input focus.
	OnExit (see page 17642)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 17644)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 17647)	Occurs when a key is pressed.
	OnKeyUp (see page 17648)	Occurs when the user releases a key that has been pressed.
	OnMouseDown (see page 17650)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 17652)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 17654)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 17656)	Occurs immediately after the control is resized.
	OnStartDrag (see page 17657)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnTopLeftChanged (see page 17658)	Occurs when the data region of the grid scrolls.

Published Properties

	Name	Description
	Align (see page 17601)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 17605)	Specifies how the control is anchored to its parent.
	BorderStyle (see page 17605)	Determines whether a single line border is drawn around the grid.
	CaptionColor (see page 17605)	Specifies the background color of cells containing the names of the row and column dimensions.
	CaptionFont (see page 17606)	Specifies the font used to write the names of the row and column dimensions.
	Color (see page 17607)	Specifies the background color of the control.
	Constraints (see page 17608)	Specifies the size constraints for the control.
	Ctl3D (see page 17608)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DataColor (see page 17609)	Specifies the background color of the cells containing summarized values.
	DataFont (see page 17609)	Specifies the font used to write the summarized values in the scrollable region of the decision grid.
	DataSumColor (see page 17610)	Specifies the background color of cells that represent subtotals over entire dimensions.
	DecisionSource (see page 17610)	Specifies the decision source that supplies crosstabulated data to the decision grid.
	DefaultColWidth (see page 17610)	Specifies the width of all columns that are not explicitly resized.
	DefaultDrawing (see page 17611)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 17611)	Specifies the height of all rows that are not explicitly resized.
	Dimensions (see page 17612)	Provides access to the collection of TDisplayDim objects that control the appearance of each dimension.
	DragCursor (see page 17612)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 17613)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 17614)	Controls whether the control responds to mouse, keyboard, and timer events.
	GridLineColor (see page 17615)	Specifies the color of the lines that separate the cells in the decision grid.
	GridLineWidth (see page 17616)	Specifies the width, in pixels, of the lines that separate the cells of the grid.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	LabelColor (see page 17616)	Specifies the color of the individual row and column labels.
	LabelFont (see page 17616)	Specifies the font used to write the individual row and column labels.
	LabelSumColor (see page 17617)	Specifies the background color for the subtotal labels.

	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 17617)	Specifies settings that affect the entire grid display.
	ParentColor (see page 17618)	Determines where a control looks for its color information.
	ParentCtl3D (see page 17618)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 17619)	Determines where a control looks for its font information.
	ParentShowHint (see page 17619)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 17620)	Identifies the pop-up menu associated with the control.
	ScrollBars (see page 17625)	Specifies whether the grid includes horizontal and vertical scroll bars.
	ShowCubeEditor (see page 17625)	Determines whether the Cube Editor can be displayed.
	ShowHint (see page 17625)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 17627)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 17627)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Totals (see page 17628)	Indicates whether the grid includes subtotals.
	Visible (see page 17628)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

109.1.2.2 TDecisionGrid Properties

The properties of the TDecisionGrid class are listed here.

Public Properties

	Name	Description
	ColCount (see page 17606)	Indicates the total number of columns in the decision grid.
	FixedCols (see page 17615)	Indicates the number of columns used for dimension captions and dimension value labels.
	FixedRows (see page 17615)	Indicates the number of rows used for dimension captions and dimension value labels.
	RowCount (see page 17624)	Indicates the total number of rows in the decision grid.

Published Properties

	Name	Description
	Align (see page 17601)	Determines how the control aligns within its container (parent control).
	Anchors (see page 17605)	Specifies how the control is anchored to its parent.
	BorderStyle (see page 17605)	Determines whether a single line border is drawn around the grid.
	CaptionColor (see page 17605)	Specifies the background color of cells containing the names of the row and column dimensions.
	CaptionFont (see page 17606)	Specifies the font used to write the names of the row and column dimensions.
	Color (see page 17607)	Specifies the background color of the control.
	Constraints (see page 17608)	Specifies the size constraints for the control.
	Ctl3D (see page 17608)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DataColor (see page 17609)	Specifies the background color of the cells containing summarized values.
	DataFont (see page 17609)	Specifies the font used to write the summarized values in the scrollable region of the decision grid.
	DataSumColor (see page 17610)	Specifies the background color of cells that represent subtotals over entire dimensions.
	DecisionSource (see page 17610)	Specifies the decision source that supplies crosstabulated data to the decision grid.
	DefaultColWidth (see page 17610)	Specifies the width of all columns that are not explicitly resized.

	DefaultDrawing (see page 17611)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 17611)	Specifies the height of all rows that are not explicitly resized.
	Dimensions (see page 17612)	Provides access to the collection of TDisplayDim objects that control the appearance of each dimension.
	DragCursor (see page 17612)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 17613)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 17614)	Controls whether the control responds to mouse, keyboard, and timer events.
	GridLineColor (see page 17615)	Specifies the color of the lines that separate the cells in the decision grid.
	GridLineWidth (see page 17616)	Specifies the width, in pixels, of the lines that separate the cells of the grid.
	LabelColor (see page 17616)	Specifies the color of the individual row and column labels.
	LabelFont (see page 17616)	Specifies the font used to write the individual row and column labels.
	LabelSumColor (see page 17617)	Specifies the background color for the subtotal labels.
	Options (see page 17617)	Specifies settings that affect the entire grid display.
	ParentColor (see page 17618)	Determines where a control looks for its color information.
	ParentCtl3D (see page 17618)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 17619)	Determines where a control looks for its font information.
	ParentShowHint (see page 17619)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 17620)	Identifies the pop-up menu associated with the control.
	ScrollBars (see page 17625)	Specifies whether the grid includes horizontal and vertical scroll bars.
	ShowCubeEditor (see page 17625)	Determines whether the Cube Editor can be displayed.
	ShowHint (see page 17625)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 17627)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 17627)	Determines if the user can tab to a control.
	Totals (see page 17628)	Indicates whether the grid includes subtotals.
	Visible (see page 17628)	Determines whether the component appears onscreen.

109.1.2.2.1 TDecisionGrid.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
```

```
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

```
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

109.1.2.2.2 TDecisionGrid.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

109.1.2.2.3 TDecisionGrid.BorderStyle Property

Determines whether a single line border is drawn around the grid.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Set BorderStyle to bsSingle to add a single line border around the grid's image. Set BorderStyle to bsNone to omit the border.

109.1.2.2.4 TDecisionGrid.CaptionColor Property

Specifies the background color of cells containing the names of the row and column dimensions.

Pascal

```
property CaptionColor: TColor;
```

C++

```
__property TColor CaptionColor;
```

Description

Set CaptionColor to the color that should be used to make the names of the row and column dimensions stand out. These are the names for the dimensions that are set in the Decision Cube Editor.

The actual values of the fields that make up the row and column dimensions are drawn using the LabelColor property.

See Also

CaptionFont (see page 17575)

DataColor (see page 17576)

DataSumColor (see page 17577)

Dimensions (see page 17579)

LabelColor (see page 17581)

LabelSumColor (see page 17581)

109.1.2.2.5 TDecisionGrid.CaptionFont Property

Specifies the font used to write the names of the row and column dimensions.

Pascal

```
property CaptionFont: TFont;
```

C++

```
__property TFont CaptionFont;
```

Description

Set CaptionFont to the font that should be used to make the names of the row and column dimensions stand out. These are the names for the dimensions that are set in the Decision Cube Editor.

The actual values of the fields that make up the row and column dimensions are written using the LabelFont property.

See Also

CaptionColor (see page 17574)

DataFont (see page 17577)

Dimensions (see page 17579)

LabelFont (see page 17581)

TFont (see page 14497)

109.1.2.2.6 TDecisionGrid.ColCount Property

Indicates the total number of columns in the decision grid.

Pascal

```
property ColCount: Longint;
```

C++

```
__property int ColCount;__property Longint ColCount;
```

Description

Read ColCount to determine the number of columns in the decision grid, including the columns containing dimension captions

and label cells. Compute the number of columns in the data region of the decision grid by subtracting the value of FixedCols from ColCount.

See Also

FixedCols (see page 17579)

109.1.2.2.7 TDecisionGrid.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

109.1.2.2.8 TDecisionGrid.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

109.1.2.2.9 TDecisionGrid.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

109.1.2.2.10 TDecisionGrid.DataColor Property

Specifies the background color of the cells containing summarized values.

Pascal

```
property DataColor: TColor;
```

C++

```
__property TColor DataColor;
```

Description

Set DataColor to the color that should be used for the scrollable data region of the decision grid. The data region contains the values of the current summary of the decision cube, where each cell represents the value of a field summarized over the values indicated by the row and column dimensions.

The subtotals that represent values summarized over all values within a given dimension (as well as the single value in other dimensions) are drawn using the DataSumColor property.

See Also

CaptionColor (see page 17574)

DataFont (see page 17577)

DataSumColor (see page 17577)

LabelColor (see page 17581)

LabelSumColor (see page 17581)

109.1.2.2.11 TDecisionGrid.DataFont Property

Specifies the font used to write the summarized values in the scrollable region of the decision grid.

Pascal

```
property DataFont: TFont;
```

C++

```
__property TFont DataFont;
```

Description

Set DataFont to the font that should be used for the data region of the decision grid. The data region contains the values of the current summary of the decision cube, where each cell represents the value of a field summarized over the values indicated by the row and column dimensions. It also includes the dimension subtotals over the summary field.

See Also

CaptionFont (see page 17575)

DataColor (see page 17576)

LabelFont (see page 17581)

109.1.2.2.12 TDecisionGrid.DataSumColor Property

Specifies the background color of cells that represent subtotals over entire dimensions.

Pascal

```
property DataSumColor: TColor;
```

C++

```
__property TColor DataSumColor;
```

Description

Set DataSumColor to make the subtotals that represent an entire dimension stand out from the summary values that summarize information over a single value in each dimension. The subtotal cells are placed in the scrollable data region at the ends of the set of rows or columns that correspond to a given dimension.

See Also

CaptionColor (see page 17574)

DataColor (see page 17576)

DataFont (see page 17577)

LabelColor (see page 17581)

LabelSumColor (see page 17581)

Totals (see page 17583)

109.1.2.2.13 TDecisionGrid.DecisionSource Property

Specifies the decision source that supplies crosstabulated data to the decision grid.

Pascal

```
property DecisionSource: TDecisionSource;
```

C++

```
__property TDecisionSource DecisionSource;
```

Description

Set DecisionSource to link the decision grid to the crosstabulated data from a decision cube. The DecisionSource object maintains information about the current pivot state of the data. Thus, any two decision controls that share the same decision source will display the data from the decision cube using the same configuration of row and column dimensions, including any closed dimensions or drilled down values.

See Also

TDecisionCube (see page 17442)

TDecisionSource (see page 17462)

109.1.2.2.14 TDecisionGrid.DefaultColWidth Property

Specifies the width of all columns that are not explicitly resized.

Pascal

```
property DefaultColWidth: Integer;
```

C++

```
__property int DefaultColWidth;__property int DefaultColWidth;
```

Description

Set DefaultColWidth to the desired width of the columns in the decision grid that are not resized using the mouse. Setting DefaultColWidth changes the size of all columns in the grid, including columns that have been resized.

See Also

DefaultRowHeight (see page 17578)

109.1.2.2.15 TDecisionGrid.DefaultDrawing Property

Determines whether the Cells are painted when the grid is drawn.

Pascal

```
property DefaultDrawing: Boolean;
```

C++

```
__property Boolean DefaultDrawing;
```

Description

Set DefaultDrawing to true to have the grid automatically paint the cells in the grid, including adding the background color, the 3-D effects on the fixed cells, and the focus rectangle around the selected cell. Set DefaultDrawing to false to skip the painting of the cells when the grid is drawn.

When DefaultDrawing is false, the cells should be painted using the DrawCell method. For TCustomGrid, DrawCell is an abstract method. Descendants of TCustomGrid use DrawCell to provide their own customized drawing code, or to generate an OnDrawCell event.

See Also

DrawCell (see page 14826)

OnDrawCell (see page 14914)

109.1.2.2.16 TDecisionGrid.DefaultRowHeight Property

Specifies the height of all rows that are not explicitly resized.

Pascal

```
property DefaultRowHeight: Integer;
```

C++

```
__property int DefaultRowHeight;__property int DefaultRowHeight;
```

Description

Set DefaultRowHeight to the desired height of the rows in the decision grid that are not resized using the mouse. Setting DefaultRowHeight changes the size of all rows in the grid, including rows that have been resized.

See Also

DefaultColWidth (see page 17578)

109.1.2.2.17 TDecisionGrid.Dimensions Property

Provides access to the collection of TDisplayDim objects that control the appearance of each dimension.

Pascal

```
property Dimensions: TDisplayDims;
```

C++

```
__property TDisplayDims Dimensions;
```

Description

Set the properties of the individual TDisplayDim objects to control the appearance of individual dimensions within the grid. The TDisplayDim objects have properties that can override the various color and font properties of the grid to make individual dimensions stand out. They also allow the dimension names, alignments, and formats to be customized, and permit the dimension subtotals for a dimension to be turned on or off.

Note: Do not change the Dimensions property, only the properties of individual TDisplayDim objects accessed through the Dimensions property. To programmatically change the configuration of dimensions that appears in the grid, use the decision source object.

See Also

CaptionColor (see page 17574)

CaptionFont (see page 17575)

DataColor (see page 17576)

DataFont (see page 17577)

DataSumColor (see page 17577)

LabelColor (see page 17581)

LabelFont (see page 17581)

LabelSumColor (see page 17581)

TDecisionSource (see page 17462)

TDisplayDim (see page 17661)

109.1.2.2.18 TDecisionGrid.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

109.1.2.2.19 TDecisionGrid.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

109.1.2.2.20 TDecisionGrid.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

109.1.2.2.21 TDecisionGrid.FixedCols Property

Indicates the number of columns used for dimension captions and dimension value labels.

Pascal

```
property FixedCols: Integer;
```

C++

```
__property int FixedCols;__property int FixedCols;
```

Description

Read FixedCols to determine the number of columns in the decision grid that are not part of the data region. Subtract FixedCols from the value of the ColCount property to obtain the number of columns in the data region.

See Also

ColCount (see page 17576)

109.1.2.2.22 TDecisionGrid.FixedRows Property

Indicates the number of rows used for dimension captions and dimension value labels.

Pascal

```
property FixedRows: Integer;
```

C++

```
__property int FixedRows;__property int FixedRows;
```

Description

Read FixedRows to determine the number of rows in the decision grid that are not part of the data region. Subtract FixedRows from the value of the RowCount property to obtain the number of rows in the data region.

See Also

RowCount (see page 17582)

109.1.2.2.23 TDecisionGrid.GridLineColor Property

Specifies the color of the lines that separate the cells in the decision grid.

Pascal

```
property GridLineColor: TColor;
```

C++

```
__property TColor GridLineColor;
```

Description

Set GridLineColor to specify the color of the lines drawn between the cells of the grid. The grid lines only appear when the Options property includes cgGridLines.

See Also

GridLineWidth (see page 17580)

Options (see page 17582)

109.1.2.2.24 TDecisionGrid.GridLineWidth Property

Specifies the width, in pixels, of the lines that separate the cells of the grid.

Pascal

```
property GridLineWidth: Integer;
```

C++

```
__property int GridLineWidth;__property int GridLineWidth;
```

Description

Set GridLineWidth to make the lines that separate the cells in the decision grid heavier or lighter. The grid lines only appear when the Options property includes cgGridLines.

See Also

GridLineColor (see page 17580)

Options (see page 17582)

109.1.2.2.25 TDecisionGrid.LabelColor Property

Specifies the color of the individual row and column labels.

Pascal

```
property LabelColor: TColor;
```

C++

```
__property TColor LabelColor;
```

Description

Set LabelColor to the color that should be used to make the row and column labels stand out. The row and column labels of the decision grid contain the values of the fields that correspond to the dimensions of the decision cube. These are the values over which the data is summarized.

See Also

CaptionColor (see page 17574)

DataColor (see page 17576)

DataSumColor (see page 17577)

LabelFont (see page 17581)

LabelSumColor (see page 17581)

109.1.2.2.26 TDecisionGrid.LabelFont Property

Specifies the font used to write the individual row and column labels.

Pascal

```
property LabelFont: TFont;
```

C++

```
__property TFont LabelFont;
```

Description

Set LabelFont to the font that should be used to write the labels that display the values of the dimensions over which data is summarized. The row and column labels of the decision grid contain the values of the fields that correspond to the dimensions of the decision cube. LabelFont is also used when writing the labels for rows or columns that represent dimension subtotals.

See Also

CaptionFont (see page 17575)

DataFont (see page 17577)

LabelColor (see page 17581)

LabelSumColor (see page 17581)

109.1.2.2.27 TDecisionGrid.LabelSumColor Property

Specifies the background color for the subtotal labels.

Pascal

```
property LabelSumColor: TColor;
```

C++

```
__property TColor LabelSumColor;
```

Description

Set LabelSumColor make the subtotal labels stand out from the labels that represent individual field values along a dimension. The subtotal labels appear at the end of the set of field value labels for an individual dimension.

See Also

CaptionColor (see page 17574)

DataColor (see page 17576)

DataSumColor (see page 17577)

LabelColor (see page 17581)

LabelFont (see page 17581)

Totals (see page 17583)

109.1.2.2.28 TDecisionGrid.Options Property

Specifies settings that affect the entire grid display.

Pascal

```
property Options: TGridOptions;
```

C++

```
__property TDecisionGridOptions Options;__property TGridOptions Options;
```

Description

Set Options to include the desired properties for the grid. Options is a set drawn from the following values:

Value	Meaning
cgGridLines	Vertical and horizontal lines are drawn to separate the cells in the scrollable data region.

cgOutliner	The dimension caption cells contain plus (+) and minus (-) signs that open or close dimensions in the grid when clicked. The minus sign closes (hides) an adjacent dimension. The plus sign opens (shows) an adjacent dimension.
cgPivotable	Dimensions can be rearranged by clicking on the dimension caption cells and dragging them to a new position.

Note: If Options does not include cgOutliner, individual dimensions can still be opened and closed using the decision pivot. Similarly, if Options does not include cgPivotable, individual dimensions can still be opened and closed with the decision pivot.

See Also

GridLineColor (see page 17580)

GridLineWidth (see page 17580)

TDecisionPivot (see page 17680)

109.1.2.2.29 TDecisionGrid.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

109.1.2.2.30 TDecisionGrid.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

109.1.2.2.31 TDecisionGrid.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

109.1.2.2.32 TDecisionGrid.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

109.1.2.2.33 TDecisionGrid.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
```

```
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
```

```
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
```

```
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
```

```
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

109.1.2.2.34 TDecisionGrid.RowCount Property

Indicates the total number of rows in the decision grid.

Pascal

```
property RowCount: Longint;
```

C++

```
__property int RowCount;__property Longint RowCount;
```

Description

Read RowCount to determine the number of rows in the decision grid, including the rows containing dimension captions and label cells. Compute the number of rows in the data region of the decision grid by subtracting the value of FixedRows from RowCount.

See Also

FixedRows (see page 17580)

109.1.2.2.35 TDecisionGrid.ScrollBars Property

Specifies whether the grid includes horizontal and vertical scroll bars.

Pascal

```
property ScrollBars: TScrollStyle;
```

C++

```
__property TScrollStyle ScrollBars;
```

Description

Use ScrollBars to give the grid horizontal or vertical scroll bars.

Note: If all the cells in the grid fit in the ClientWidth, no horizontal scroll bar appears, even if ScrollBars is ssHorizontal or ssBoth. If all the cells fit in the ClientHeight, no vertical scroll bar appears, even if ScrollBars is ssVertical or ssBoth.

See Also

ClientHeight (see page 5808)

ClientWidth (see page 5811)

109.1.2.2.36 TDecisionGrid.ShowCubeEditor Property

Determines whether the Cube Editor can be displayed.

Pascal

```
property ShowCubeEditor: boolean;
```

C++

```
__property boolean ShowCubeEditor;
```

Description

ShowCubeEditor determines whether the Cube Editor for the associated TDecisionCube can be invoked. If set to true, a menu item to invoke the Cube Editor is added to the speed menu displayed on a right-click on the decision grid at runtime.

See Also

Dimensions (see page 17579)

ShowCubeDialog (see page 17450)

TDecisionCube (see page 17442)

109.1.2.2.37 TDecisionGrid.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
```

```
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

109.1.2.2.38 TDecisionGrid.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

109.1.2.2.39 TDecisionGrid.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

109.1.2.2.40 TDecisionGrid.Totals Property

Indicates whether the grid includes subtotals.

Pascal

```
property Totals: boolean;
```

C++

```
__property boolean Totals;
```

Description

Set Totals to turn row and column subtotals on or off for the whole grid. To turn subtotals on or off for individual dimensions, use the Dimensions property.

See Also

Dimensions (see page 17579)

109.1.2.2.41 TDecisionGrid.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
```

```
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

109.1.2.3 TDecisionGrid Events

The events of the TDecisionGrid class are listed here.

Published Events

	Name	Description
	OnClick (see page 17630)	Occurs when the user clicks the control.
	OnDblClick (see page 17632)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDecisionDrawCell (see page 17632)	Occurs just before a cell in the grid is drawn.
	OnDecisionExamineCell (see page 17634)	Occurs when the user right-clicks on a cell in the data region of the grid.
	OnDragDrop (see page 17634)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 17637)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 17639)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 17640)	Occurs when a control receives the input focus.
	OnExit (see page 17642)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 17644)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 17647)	Occurs when a key is pressed.
	OnKeyUp (see page 17648)	Occurs when the user releases a key that has been pressed.
	OnMouseDown (see page 17650)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 17652)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 17654)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnResize (see page 17656)	Occurs immediately after the control is resized.
	OnStartDrag (see page 17657)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnTopLeftChanged (see page 17658)	Occurs when the data region of the grid scrolls.

109.1.2.3.1 TDecisionGrid.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.

- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

109.1.2.3.2 TDecisionGrid.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

109.1.2.3.3 TDecisionGrid.OnDecisionDrawCell Event

Occurs just before a cell in the grid is drawn.

Pascal

```
property OnDecisionDrawCell: TDecisionDrawCellEvent;
```

C++

```
__property TDecisionDrawCellEvent OnDecisionDrawCell;
```

Description

Write an OnDecisionDrawCell event handler to change the color, font, or text used to display an individual cell in the grid. Change the AColor, AFont, or Value parameter to change the color, font, or text respectively.

For example, to change the row and column labels from the values of the fields in the dataset to customized labels, change the Value parameter of the event handler when the Row and Column parameters represent a label cell.

The Row and Column parameters indicate the row and column of a cell within the grid, where (0,0) specifies the upper left summary value, not the upper left corner of the grid. Thus, the label cells contain negative row or column indexes. In addition, the AState parameter includes gdFixed when the cell is a label cell. AState also indicates whether the cell being drawn is selected.

The ADrawState parameter indicates the type of cell that is about to be drawn. ADrawState includes one or more of the following values:

Value	Meaning
dsGroupStart	The cell is the first row or column label for a dimension. Always appears with dsRowValue or dsColValue.
dsRowCaption	The cell displays a row caption (the name of a dimension whose values are listed down a column on the left of the grid).
dsColCaption	The cell displays a column caption (the name of a dimension whose values are listed along the top of the grid).
dsSum	The cell displays a subtotal label or value. Always appears with dsRowValue, dsColValue, or dsData.
dsRowValue	The cell is a row label. It represents a value from a dimension listed down a column on the left of the grid or a summary label that appears at the end of each group of dimension values.
dsColValue	The cell is a column label. It represents a value from a dimension listed across the top of the grid or a summary label that appears at the end of the dimension.
dsData	The cell is in the scrollable data region of the grid.
dsOpenAfter	The cell has a plus sign (+) on the right that will open following dimension when clicked. Always appears with dsRowCaption, dsColCaption, dsRowPlus, or dsColPlus.
dsCloseAfter	The cell has a minus sign (-) on the right that will close the following dimension when clicked. Always appears with dsRowCaption, dsColCaption, dsRowPlus, or dsColPlus.
dsRowIndicator	The cell appears on the left grid margin that is used to display the open and close indicators (+ and -).
dsColIndicator	The cell appears on the upper grid margin that is used to display the open and close indicators (+ and -).
dsRowPlus	The cell appears in the left margin and contains a plus sign (+) indicating that the adjacent dimension is closed. Always appears with dsRowIndicator.
dsColPlus	The cell appears in the top margin and contains a plus sign (+) indicating that the adjacent dimension is closed. Always appears with dsColIndicator.
dsNone	The cell has no function.

See Also

CaptionColor (see page 17574)

CaptionFont (see page 17575)

DataColor (see page 17576)

DataFont (see page 17577)

DataSumColor (see page 17577)

LabelColor (see page 17581)

LabelFont (see page 17581)

LabelSumColor (see page 17581)

109.1.2.3.4 TDecisionGrid.OnDecisionExamineCell Event

Occurs when the user right-clicks on a cell in the data region of the grid.

Pascal

```
property OnDecisionExamineCell: TDecisionExamineCellEvent;
```

C++

```
__property TDecisionExamineCellEvent OnDecisionExamineCell;
```

Description

Write an OnDecisionExamineCell event handler to take specific action when the user right-clicks on one of the cells that contains summarized data. The ICol and IRow parameters indicate the column and row indexes of the cell. The column and row indexes specify the value 0 for the top or leftmost cell in the scrollable data region, not the top left cell of the decision grid. The ISum parameter is the index of the current summary in the decision cube, where the first active summary has an index of 0. The ValueArray parameter gives the coordinates of the cell in terms of the dimensions of the decision source. If the cell represents a subtotal value, the entry in the ValueArray for the dimension over which the subtotal is calculated is a negative number.

Use the Cells property to obtain the value of the summary in the cell. Use the GetDimensionName method of the decision source to get the name of each dimension represented by an entry in the ValueArray parameter. Use the GetValueAsString or GetValueAsVariant method of the decision source to obtain the label values for each dimension represented in the ValueArray parameter.

See Also

Cells (see page 17575)

DecisionSource (see page 17578)

GetValueArray (see page 17480)

GetDimensionName (see page 17474)

GetDataAsString (see page 17473)

GetDataAsVariant (see page 17473)

TSmallIntArray (see page 17361)

109.1.2.3.5 TDecisionGrid.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the

OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;
```

```
procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

109.1.2.3.6 TDecisionGrid.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
```

```
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

109.1.2.3.7 TDecisionGrid.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

109.1.2.3.8 TDecisionGrid.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
```

```
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
```

```
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

109.1.2.3.9 TDecisionGrid.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

109.1.2.3.10 TDecisionGrid.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not

generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
```

```
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;
```

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

109.1.2.3.11 TDecisionGrid.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

109.1.2.3.12 TDecisionGrid.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

```
void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

109.1.2.3.13 TDecisionGrid.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
```

```
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
```

```
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

109.1.2.3.14 TDecisionGrid.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
```

```
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
```

```
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

109.1.2.3.15 TDecisionGrid.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
```

```
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
```

```
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

109.1.2.3.16 TDecisionGrid.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

109.1.2.3.17 TDecisionGrid.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

109.1.2.3.18 TDecisionGrid.OnTopLeftChanged Event

Occurs when the data region of the grid scrolls.

Pascal

```
property OnTopLeftChanged: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnTopLeftChanged;
```

Description

Write an OnTopLeftChanged event handler to take specific action when the summarized data in the grid scrolls. Use the Cells property or the CellDrawState method to obtain the new values of the data cells.

See Also

Cells (see page 17575)

CellDrawState (see page 17568)

CellValueArray (see page 17569)

109.1.3 TDecisionGridDataLink Class

Class Hierarchy

Pascal

```
TDecisionGridDataLink = class(TDecisionDataLink);
```

C++

```
class TDecisionGridDataLink : public TDecisionDataLink;
```

File

MXGRID

Description

This is class MXGRID.TDecisionGridDataLink.

109.1.3.1 TDecisionGridDataLink Members

The following tables list the members exposed by TDecisionGridDataLink.

Protected Fields

	Name	Description
	FBlocked (see page 17460)	This is FBlocked, a member of class TDecisionDataLink.
	FDecisionSource (see page 17461)	This is FDecisionSource, a member of class TDecisionDataLink.

Protected Methods

	Name	Description
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	DecisionDataEvent (see page 17660)	This is DecisionDataEvent, a member of class TDecisionGridDataLink.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetDecisionSource (see page 17461)	This is SetDecisionSource, a member of class TDecisionDataLink.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 17660)	This is Create, a member of class TDecisionGridDataLink.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 17660)	Destroys the TPersistent instance and frees its memory.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	DecisionSource (see page 17462)	This is DecisionSource, a member of class TDecisionDataLink.

109.1.3.2 TDecisionGridDataLink.Create Constructor

Pascal

```
constructor Create(AGrid: TCustomDecisionGrid);
```

C++

```
__fastcall TDecisionGridDataLink(TCustomDecisionGrid AGrid);
```

Description

This is Create, a member of class TDecisionGridDataLink.

109.1.3.3 TDecisionGridDataLink.Destroy Destructor

Destroys the TPersistent instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDecisionGridDataLink();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

Free (see page 22300)

Destroy (see page 22287)

109.1.3.4 TDecisionGridDataLink Methods

The methods of the TDecisionGridDataLink class are listed here.

Protected Methods

	Name	Description
	DecisionDataEvent (see page 17660)	This is DecisionDataEvent, a member of class TDecisionGridDataLink.

109.1.3.4.1 TDecisionGridDataLink.DecisionDataEvent Method

Pascal

```
procedure DecisionDataEvent(Event: TDecisionDataEvent); override;
```

C++

```
virtual __fastcall DecisionDataEvent(TDecisionDataEvent Event);
```

Description

This is DecisionDataEvent, a member of class TDecisionGridDataLink.

109.1.4 TDisplayDim Class

TDisplayDim represents a dimension in a decision grid.

Class Hierarchy

Pascal

```
TDisplayDim = class(TCollectionItem);
```

C++

```
class TDisplayDim : public TCollectionItem;
```

File

MXGRID

Description

Use TDisplayDim to set the properties of a dimension in a decision grid. Decision grids maintain a collection of TDisplayDim objects that describe the dimension labels that run along their left and top edges. The properties of TDisplayDim can be used to make the labels for a single dimension stand out, or to turn on or off subtotals over the dimension.

See Also

TDecisionGrid (see page 17586)

TDisplayDims (see page 17666)

TDisplayDims (see page 17666)

109.1.4.1 TDisplayDim Members

The following tables list the members exposed by TDisplayDim.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	assign (see page 17663)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 17662)	Creates and initializes a TCollectionItem instance.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	Alignment (see page 17664)	Specifies the placement of values within the label cells.
	Color (see page 17664)	Specifies the background color of the label cells for the dimension.
	DisplayName (see page 17665)	Describes the dimension in the caption cell.
	FieldName (see page 17665)	Names the field in the underlying data set that the dimension represents.
	Format (see page 17665)	Specifies the format of the value labels along the dimension.
	Subtotals (see page 17666)	Indicates whether the grid should display subtotals over the dimension.

109.1.4.2 TDisplayDim.Create Constructor

Creates and initializes a TCollectionItem instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TDisplayDim(TCollection Collection);
```

Description

Do not call Create to instantiate a collection item. Instead, call the Add method of the collection to which the item should belong.

Collection identifies the TCollection instance to which the new item belongs.

See Also

Add (see page 2736)

Collection (see page 2761)

109.1.4.3 TDisplayDim Methods

The methods of the TDisplayDim class are listed here.

Public Methods

	Name	Description
	assign (see page 17663)	Copies the contents of another, similar object.

109.1.4.3.1 TDisplayDim.assign Method

Copies the contents of another, similar object.

Pascal

```
procedure assign(Value: TPersistent); override;
```

C++

```
virtual __fastcall assign(TPersistent * Value);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo (see page 2931)

EConvertError (see page 22724)

109.1.4.4 TDisplayDim Properties

The properties of the TDisplayDim class are listed here.

Published Properties

	Name	Description
	Alignment (see page 17664)	Specifies the placement of values within the label cells.
	Color (see page 17664)	Specifies the background color of the label cells for the dimension.
	DisplayName (see page 17665)	Describes the dimension in the caption cell.
	FieldName (see page 17665)	Names the field in the underlying data set that the dimension represents.
	Format (see page 17665)	Specifies the format of the value labels along the dimension.
	Subtotals (see page 17666)	Indicates whether the grid should display subtotals over the dimension.

109.1.4.4.1 TDisplayDim.Alignment Property

Specifies the placement of values within the label cells.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Set Alignment to determine how the values of the dimension field are aligned in the label cells. Alignment can be one of the following values:

Value	Meaning
taLeftJustify	Values appear lined up along the left edge of the label cells.
taRightJustify	Values appear lined up along the right edge of the label cells.

See Also

GetMemberAsString (see page 17477)

109.1.4.4.2 TDisplayDim.Color Property

Specifies the background color of the label cells for the dimension.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Set Color to override the LabelColor property of the decision grid for the labels along a single dimension. Cells that display the field values along this dimension use Color for the background color.

See Also

LabelColor (see page 17581)

109.1.4.4.3 TDisplayDim.DisplayName Property

Describes the dimension in the caption cell.

Pascal

```
property DisplayName: string;
```

C++

```
__property AnsiString DisplayName;
```

Description

Set DisplayName to the name that appears in the caption cell that displays the name of this dimension. DisplayName overrides the name of the dimension returned by the decision source in its GetDimensionName method.

See Also

FieldName (see page 17665)

GetDimensionName (see page 17474)

109.1.4.4.4 TDisplayDim.FieldName Property

Names the field in the underlying data set that the dimension represents.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;
```

Description

Read FieldName to determine the name of the field on which the dimension item is based. FieldName is the same as the FieldName property of the TField object for the field in the dataset.

Note: Do not change the FieldName property. Changing the FieldName property loses the link to the underlying field in the data set.

See Also

DisplayName (see page 17665)

FieldName (see page 7164)

109.1.4.4.5 TDisplayDim.Format Property

Specifies the format of the value labels along the dimension.

Pascal

```
property Format: String;
```

C++

```
__property AnsiString Format;
```

Description

Set Format to specify how the values of the dimension field should be formatted into display labels. The format specifiers that make up the Format string depend on the field type.

See the FormatFloat runtime library function for a description of the format specifiers for a numeric field. See the FormatDateTime runtime library function for a description of the format specifiers for a temporal field. Format has no effect on other field types.

See Also

FormatDateTime (see page 22902)

FormatFloat (see page 22904)

109.1.4.4.6 TDisplayDim.Subtotals Property

Indicates whether the grid should display subtotals over the dimension.

Pascal

```
property Subtotals: Boolean;
```

C++

```
__property Boolean Subtotals;
```

Description

Set Subtotals to true to add a row or column of subtotals following the last value of the dimension. The subtotal row or column contains values of the summary field summarized over all values in the dimension. Set Subtotals to false to omit the subtotal row or column at the end of the dimension.

To turn subtotals on or off for all dimensions in the grid at once, use the Totals property of the decision grid.

See Also

Totals (see page 17583)

109.1.5 TDisplayDims Class

TDisplayDims represents a collection of TDisplayDim (see page 17661) objects.

Class Hierarchy

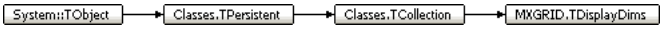

Pascal

```
TDisplayDims = class(TCollection);
```

C++

```
class TDisplayDims : public TCollection;
```

File

MXGRID

Description

Use TDisplayDims to describe the dimensions of a decision grid. Each TDisplayDim (see page 17661) object maintained by TDisplayDims corresponds to a field from the decision source. Decision grids use TDisplayDims to specify the appearance of the dimension labels that appear along the left and top edges of the grid.

See Also

TDecisionGraph (see page 17521)

TDecisionGrid (see page 17586)

TDecisionSource (see page 17462)

TDisplayDim (see page 17661)

109.1.5.1 TDisplayDims Members

The following tables list the members exposed by TDisplayDims.

Protected Fields

	Name	Description
	FGrid (see page 17669)	This is FGrid, a member of class TDisplayDims.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 17669)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	NotifyOwner (see page 17670)	This is NotifyOwner, a member of class TDisplayDims.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 2736)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 2734)	Creates and initializes a collection.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
	Destroy (see page 2735)	Destroys the collection and each item in it.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 17670)	Lists the TDisplayDim (see page 17661) objects in the collection.

109.1.5.2 TDisplayDims Fields

The fields of the TDisplayDims class are listed here.

Protected Fields

	Name	Description
	FGrid (see page 17669)	This is FGrid, a member of class TDisplayDims.

109.1.5.2.1 TDisplayDims.FGrid Field

Pascal

```
FGrid: TCustomDecisionGrid;
```

C++

```
TCustomDecisionGrid FGrid;
```

Description

This is FGrid, a member of class TDisplayDims.

109.1.5.3 TDisplayDims Methods

The methods of the TDisplayDims class are listed here.

Protected Methods

	Name	Description
V	GetOwner (see page 17669)	Returns the owner of an object.
	NotifyOwner (see page 17670)	This is NotifyOwner, a member of class TDisplayDims.

109.1.5.3.1 TDisplayDims.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

109.1.5.3.2 TDisplayDims.NotifyOwner Method

Pascal

```
procedure NotifyOwner(aType: TDDNotifyType);
```

C++

```
__fastcall NotifyOwner(TDDNotifyType aType);
```

Description

This is NotifyOwner, a member of class TDisplayDims.

109.1.5.4 TDisplayDims Properties

The properties of the TDisplayDims class are listed here.

Public Properties

	Name	Description
	Items (see page 17670)	Lists the TDisplayDim (see page 17661) objects in the collection.

109.1.5.4.1 TDisplayDims.Items Property

Lists the TDisplayDim (see page 17661) objects in the collection.

Pascal

```
property Items [Index: Integer]: TDisplayDim;
```

C++

```
__property TDisplayDim Items[int Index];
```

Description

Use Items to access the properties of the TDisplayDim (see page 17661) objects maintained by TDisplayDims (see page 17666).

Note: In Delphi code Items is the default property of TDisplayDims (see page 17666), so the name of the Items property can be left off when using this property. Thus the line

```
MyDim := DecisionGrid1.Dimensions.Items[0];
```

Note: can be written

```
MyDim := DecisionGrid1.Dimensions[0];
```

The TDisplayDim (see page 17661) objects in the Items array should correspond exactly to the dimensions of the decision source used by the decision control associated with the TDisplayDims (see page 17666) object. Do not directly change the elements of the Items array. To rearrange the display dimensions, use the SwapDimIndexes method of the decision source. To hide display dimensions, use the CloseDimIndexRight method of the decision source. To show hidden dimensions, use the OpenDimIndexLeft or OpenDimIndexRight method of the decision source.

See Also

TDecisionSource (see page 17462)

CloseDimIndexRight (see page 17470)

OpenDimIndexLeft (see page 17482)

OpenDimIndexRight (see page 17482)

SwapDimIndexes (see page 17484)

operator_[]

109.1.6 TValueArray Class

Class Hierarchy

Pascal

```
TValueArray = class(TSmallIntArray);
```

C++

```
class TValueArray : public TSmallIntArray;
```

File

MXGRID

Description

This is class MXGRID.TValueArray.

109.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TButtonPlace (see page 17672)	This is record MXGRID.TButtonPlace.
	TDDNotifyType (see page 17672)	This is record MXGRID.TDDNotifyType.
	TDecisionCellType (see page 17673)	This is record MXGRID.TDecisionCellType.
	TDecisionDrawStates (see page 17673)	This is record MXGRID.TDecisionDrawStates.
	TDecisionGridOption (see page 17674)	This is record MXGRID.TDecisionGridOption.
	TDecisionGridState (see page 17675)	This is record MXGRID.TDecisionGridState.
	TDecisionPivotState (see page 17675)	This is record MXGRID.TDecisionPivotState.

109.2.1 MXGRID.TButtonPlace Enumeration

Pascal

```
TButtonPlace = (
  bpLeft,
  bpMiddle,
  bpRight
);
```

C++

```
enum TButtonPlace {
  bpLeft,
  bpMiddle,
  bpRight
};
```

File

MXGRID

Description

This is record MXGRID.TButtonPlace.

109.2.2 MXGRID.TDDNotifyType Enumeration

Pascal

```
TDDNotifyType = (
  tdDisplay,
  tdSubTotals,
  tdMetaData
);
```

C++

```
enum TDDNotifyType {
```

```
    tdDisplay,
    tdSubTotals,
    tdMetaData
  };
```

File

MXGRID

Description

This is record MXGRID.TDDNotifyType.

109.2.3 MXGRID.TDecisionCellType Enumeration

Pascal

```
  TDecisionCellType = (
    ctNone,
    ctCaptionRow,
    ctCaptionCol,
    ctNewCaptionCol,
    ctRowLabel,
    ctColLabel,
    ctData,
    ctRowPlus,
    ctColPlus,
    ctRowIndicator,
    ctColIndicator
  );
```

C++

```
  enum TDecisionCellType {
    ctNone,
    ctCaptionRow,
    ctCaptionCol,
    ctNewCaptionCol,
    ctRowLabel,
    ctColLabel,
    ctData,
    ctRowPlus,
    ctColPlus,
    ctRowIndicator,
    ctColIndicator
  };
```

File

MXGRID

Description

This is record MXGRID.TDecisionCellType.

109.2.4 MXGRID.TDecisionDrawStates Enumeration

Pascal

```
  TDecisionDrawStates = (
    dsGroupStart,
```

```
    dsRowCaption,
    dsColCaption,
    dsSum,
    dsRowValue,
    dsColValue,
    dsData,
    dsOpenAfter,
    dsCloseAfter,
    dsCloseBefore,
    dsOpenBefore,
    dsRowIndicator,
    dsColIndicator,
    dsRowPlus,
    dsColPlus,
    dsNone
  );
```

C++

```
  enum TDecisionDrawStates {
    dsGroupStart,
    dsRowCaption,
    dsColCaption,
    dsSum,
    dsRowValue,
    dsColValue,
    dsData,
    dsOpenAfter,
    dsCloseAfter,
    dsCloseBefore,
    dsOpenBefore,
    dsRowIndicator,
    dsColIndicator,
    dsRowPlus,
    dsColPlus,
    dsNone
  };
```

File

MXGRID

Description

This is record MXGRID.TDecisionDrawStates.

109.2.5 MXGRID.TDecisionGridOption Enumeration

Pascal

```
  TDecisionGridOption = (
    cgGridLines,
    cgOutliner,
    cgPivotable
  );
```

C++

```
  enum TDecisionGridOption {
    cgGridLines,
    cgOutliner,
    cgPivotable
  };
```

File

MXGRID

Description

This is record MXGRID.TDecisionGridOption.

109.2.6 MXGRID.TDecisionGridState Enumeration

Pascal

```
TDecisionGridState = (
  csNormal,
  csPivoting
);
```

C++

```
enum TDecisionGridState {
  csNormal,
  csPivoting
};
```

File

MXGRID

Description

This is record MXGRID.TDecisionGridState.

109.2.7 MXGRID.TDecisionPivotState Enumeration

Pascal

```
TDecisionPivotState = (
  psNone,
  psLeftPivot,
  psMiddlePivot,
  psRightPivot,
  psTopPivot
);
```

C++

```
enum TDecisionPivotState {
  psNone,
  psLeftPivot,
  psMiddlePivot,
  psRightPivot,
  psTopPivot
};
```

File

MXGRID

Description

This is record MXGRID.TDecisionPivotState.

109.3 Types

The following table lists types in this documentation.

Types

Name	Description
TDecisionDrawCellEvent (see page 17676)	This is type MXGRID.TDecisionDrawCellEvent.
TDecisionDrawState (see page 17676)	This is type MXGRID.TDecisionDrawState.
TDecisionExamineCellEvent (see page 17677)	This is type MXGRID.TDecisionExamineCellEvent.
TDecisionGridOptions (see page 17677)	This is type MXGRID.TDecisionGridOptions.
TDisplayDimClass (see page 17677)	This is type MXGRID.TDisplayDimClass.

109.3.1 MXGRID.TDecisionDrawCellEvent Type

Pascal

```
TDecisionDrawCellEvent = procedure (Sender: TObject; Col, Row: Longint; var Value: string; var
aFont: TFont; var aColor: TColor; AState: TGridDrawState; aDrawState: TDecisionDrawState) of
object;
```

C++

```
(Sender: TObject; Col, Row: Longint; var Value: string; var aFont: TFont; var aColor: TColor;
AState: TGridDrawState; aDrawState: TDecisionDrawState) ( TDecisionDrawCellEvent)();
```

File

MXGRID

Description

This is type MXGRID.TDecisionDrawCellEvent.

109.3.2 MXGRID.TDecisionDrawState Type

Pascal

```
TDecisionDrawState = set of TDecisionDrawStates;
```

C++

```
TDecisionDrawStates TDecisionDrawState;
```

File

MXGRID

Description

This is type MXGRID.TDecisionDrawState.

109.3.3 MXGRID.TDecisionExamineCellEvent Type

Pascal

```
TDecisionExamineCellEvent = procedure (Sender: TObject; iCol, iRow: Longint; iSum: Integer;
const ValueArray: TValueArray) of object;
```

C++

```
(Sender: TObject; iCol, iRow: Longint; iSum: Integer; const ValueArray: TValueArray) (
TDecisionExamineCellEvent)();
```

File

MXGRID

Description

This is type MXGRID.TDecisionExamineCellEvent.

109.3.4 MXGRID.TDecisionGridOptions Type

Pascal

```
TDecisionGridOptions = set of TDecisionGridOption;
```

C++

```
TDecisionGridOption TDecisionGridOptions;
```

File

MXGRID

Description

This is type MXGRID.TDecisionGridOptions.

109.3.5 MXGRID.TDisplayDimClass Type

Pascal

```
TDisplayDimClass = class of TDisplayDim;
```

C++

```
TDisplayDim TDisplayDimClass;
```

File

MXGRID

Description

This is type MXGRID.TDisplayDimClass.

110 MXPIVSRC Namespace

This is namespace MXPIVSRC.

Topics

Name	Description
Classes (see page 17680)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 17741)	The following table lists structs, records, enums in this documentation.
Types (see page 17743)	The following table lists types in this documentation.

Classes

	Name	Description
	TDecisionPivot (see page 17680)	TDecisionPivot allows the user to control the pivot state of a decision source.
	TPivotDataLink (see page 17737)	This is class MXPIVSRC.TPivotDataLink.

Structs, Records, Enums

	Name	Description
	TDecisionButtonPosition (see page 17741)	This is record MXPIVSRC.TDecisionButtonPosition.
	TDecisionPivotOption (see page 17741)	This is record MXPIVSRC.TDecisionPivotOption.

Types

Name	Description
TDecisionPivotOptions (see page 17743)	This is type MXPIVSRC.TDecisionPivotOptions.

110.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TDecisionPivot (see page 17680)	TDecisionPivot allows the user to control the pivot state of a decision source.
	TPivotDataLink (see page 17737)	This is class MXPIVSRC.TPivotDataLink.

110.1.1 TDecisionPivot Class

TDecisionPivot allows the user to control the pivot state of a decision source.

Class Hierarchy

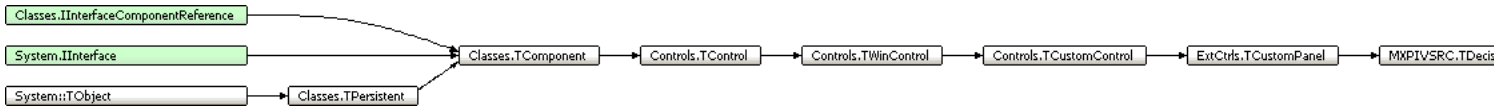

Pascal

```
TDecisionPivot = class(TCustomPanel);
```

C++

```
class TDecisionPivot : public TCustomPanel;
```

File

MXPIVSRC

Description

Add a decision pivot to a form to allow users to control the pivot state of decision graphs and decision grids. Decision pivots contain buttons that correspond to the active dimensions of a decision cube and a drop-down list box for choosing the current summary.

Pressing the dimension buttons opens or closes the dimensions, controlling whether the decision source provides detailed data values for those dimensions. Right-clicking the buttons of a decision pivot brings up a menu that allows users to control the drill state of a dimension. Drilled dimensions focus on a single value within the dimension. Dragging the buttons on a decision pivot rearranges the dimensions of the decision source into a new pivot configuration.

Note: Do not use the Decision pivot to pivot, subtotal, or drill in on summaries that are not additive. SUM and COUNT summaries are additive, AVERAGE (unless calculated by the Decision cube), MAX, and MIN are not.

See Also

TDecisionCube (see page 17442)

TDecisionGraph (see page 17521)

TDecisionGrid (see page 17586)

TDecisionSource (see page 17462)

Overview of Decision Support Components

Using Decision Graphs

Using Decision Pivots

Customizing Decision Graphs

About Crosstabs

110.1.1.1 TDecisionPivot Members

The following tables list the members exposed by TDecisionPivot.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.

	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 17692)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanAutoSize (see page 12537)	Adjusts the way the panel responds to resize attempts.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 12537)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DimStateChange (see page 17693)	This is DimStateChange, a member of class TDecisionPivot.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorderStyle (see page 17693)	This is GetBorderStyle, a member of class TDecisionPivot.
	GetBorderWidth (see page 17693)	This is GetBorderWidth, a member of class TDecisionPivot.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.

	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	NewDimLayout (see page 17694)	This is NewDimLayout, a member of class TDecisionPivot.
	NewPanelSetup (see page 17694)	This is NewPanelSetup, a member of class TDecisionPivot.
	Notification (see page 17694)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 5920)	Renders the image of a custom control.
	Paint (see page 12538)	Renders the image of the panel.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.

	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetBorderStyle (see page 17695)	This is SetBorderStyle, a member of class TDecisionPivot.
	SetBorderWidth (see page 17695)	This is SetBorderWidth, a member of class TDecisionPivot.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBackground (see page 12539)	This is SetParentBackground, a member of class TCustomPanel.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.

	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelKind (see page 6193)	Specifies the control's bevel style.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
R	Cols (see page 17707)	This is Cols, a member of class TDecisionPivot.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
R	Extras (see page 17710)	This is Extras, a member of class TDecisionPivot.
	FullRepaint (see page 12545)	Determines how the panel repaints itself when it is resized.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	Locked (see page 12545)	Determines whether a panel that is used as a toolbar is replaced by a toolbar supplied by an OLE server.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	Pages (see page 17713)	This is Pages, a member of class TDecisionPivot.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
R	Rows (see page 17719)	This is Rows, a member of class TDecisionPivot.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	VerticalAlignment (see page 12547)	Sets the vertical position of the caption.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.

	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 17691)	Creates and initializes an instance of TDecisionPivot (see page 17680).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 17692)	Destroys an instance of TDecisionPivot (see page 17680).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
V	GetControlsAlignment (see page 12538)	Returns the alignment of text in the panel.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.

V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 17695)	Resizes the decision pivot.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.

	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentBackground (see page 12546)	Determines whether control uses parent's theme background.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 17724)	Occurs when the user clicks the control.
	OnDblClick (see page 17726)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 17726)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 17728)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 17731)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 17732)	Occurs when a control receives the input focus.
	OnExit (see page 17734)	Occurs when the input focus shifts away from one control to another.
	OnResize (see page 17736)	Occurs immediately after the control is resized.
	OnStartDrag (see page 17737)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 17696)	Determines how the control aligns within its container (parent control).
	Alignment (see page 17700)	Determines how the caption is aligned within the panel.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	BevelInner (see page 17700)	Determines the style of the inner bevel of a panel.
	BevelOuter (see page 17701)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 17702)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BorderStyle (see page 17703)	Determines whether a line is drawn around the perimeter of the decision pivot.
	BorderWidth (see page 17704)	Specifies the distance, in pixels, between the outer and inner bevels.

	ButtonAutoSize (see page 17704)	Specifies whether the dimension buttons resize automatically when the decision pivot resizes.
	ButtonHeight (see page 17705)	Specifies the height, in pixels, of the dimension buttons.
	ButtonSpacing (see page 17705)	Specifies the number of pixels that appear between dimension buttons.
	ButtonWidth (see page 17705)	Specifies the width, in pixels, of the dimension buttons.
	Color (see page 17706)	Specifies the background color of the control.
	Ctl3D (see page 17707)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DecisionSource (see page 17707)	Specifies the decision source object that the decision pivot controls.
	DragCursor (see page 17708)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 17708)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 17710)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 17710)	Controls the attributes of text written on or in the control.
	GroupLayout (see page 17711)	Specifies how the dimension buttons and summary control are arranged on the decision pivot.
	Groups (see page 17712)	Specifies which controls appear in a decision pivot.
	GroupSpacing (see page 17712)	Determines the number of pixels that appear between control groups.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentColor (see page 17713)	Determines where a control looks for its color information.
	ParentCtl3D (see page 17713)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 17714)	Determines where a control looks for its font information.
	ParentShowHint (see page 17714)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 17715)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 17720)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 17721)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 17722)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 17722)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

110.1.1.2 TDecisionPivot.Create Constructor

Creates and initializes an instance of TDecisionPivot (see page 17680).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDecisionPivot(TComponent * AOwner);
```

Description

Call Create to instantiate a decision pivot at runtime. Decision pivots placed on forms at design time are created Pass a single Component as a parameter to provide the decision pivot with an Owner that is responsible for freeing it.

See Also

Owner (see page 2807)

Destroy (see page 17692)

110.1.1.3 TDecisionPivot.Destroy Destructor

Destroys an instance of TDecisionPivot (see page 17680).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDecisionPivot();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the decision pivot reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

110.1.1.4 TDecisionPivot Methods

The methods of the TDecisionPivot class are listed here.

Protected Methods

	Name	Description
V	AdjustClientRect (see page 17692)	Adjusts the ClientRect property for the idiosyncrasies of the window.
	DimStateChange (see page 17693)	This is DimStateChange, a member of class TDecisionPivot.
	GetBorderStyle (see page 17693)	This is GetBorderStyle, a member of class TDecisionPivot.
	GetBorderWidth (see page 17693)	This is GetBorderWidth, a member of class TDecisionPivot.
	NewDimLayout (see page 17694)	This is NewDimLayout, a member of class TDecisionPivot.
	NewPanelSetup (see page 17694)	This is NewPanelSetup, a member of class TDecisionPivot.
V	Notification (see page 17694)	Responds to notifications that components are being created or destroyed.
	SetBorderStyle (see page 17695)	This is SetBorderStyle, a member of class TDecisionPivot.
	SetBorderWidth (see page 17695)	This is SetBorderWidth, a member of class TDecisionPivot.

Public Methods

	Name	Description
V	SetBounds (see page 17695)	Resizes the decision pivot.

110.1.1.4.1 TDecisionPivot.AdjustClientRect Method

Adjusts the ClientRect property for the idiosyncrasies of the window.

Pascal

```
procedure AdjustClientRect(var Rect: TRect); override;
```

C++

```
virtual __fastcall AdjustClientRect(TRect Rect);
```

Description

AdjustClientRect is called internally when the panel needs accurate information on where to place child controls within the client area. AdjustClientRect reduces ClientRect by the width of the inner and outer bevels so that child objects are not placed on the beveled region.

See Also

ClientRect

Controls (see page 6197)

110.1.1.4.2 TDecisionPivot.DimStateChange Method

Pascal

```
procedure DimStateChange(iDim: Integer);
```

C++

```
__fastcall DimStateChange(int iDim);
```

Description

This is DimStateChange, a member of class TDecisionPivot.

110.1.1.4.3 TDecisionPivot.GetBorderStyle Method

Pascal

```
function GetBorderStyle: TBorderStyle;
```

C++

```
__fastcall TBorderStyle GetBorderStyle();
```

Description

This is GetBorderStyle, a member of class TDecisionPivot.

110.1.1.4.4 TDecisionPivot.GetBorderWidth Method

Pascal

```
function GetBorderWidth: TBorderWidth;
```

C++

```
__fastcall TBorderWidth GetBorderWidth();
```

Description

This is GetBorderWidth, a member of class TDecisionPivot.

110.1.1.4.5 TDecisionPivot.NewDimLayout Method

Pascal

```
procedure NewDimLayout;
```

C++

```
__fastcall NewDimLayout();
```

Description

This is NewDimLayout, a member of class TDecisionPivot.

110.1.1.4.6 TDecisionPivot.NewPanelSetup Method

Pascal

```
procedure NewPanelSetup;
```

C++

```
__fastcall NewPanelSetup();
```

Description

This is NewPanelSetup, a member of class TDecisionPivot.

110.1.1.4.7 TDecisionPivot.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

110.1.1.4.8 TDecisionPivot.SetBorderStyle Method

Pascal

```
procedure SetBorderStyle(Value: TBorderStyle);
```

C++

```
__fastcall SetBorderStyle(TBorderStyle Value);
```

Description

This is SetBorderStyle, a member of class TDecisionPivot.

110.1.1.4.9 TDecisionPivot.SetBorderWidth Method

Pascal

```
procedure SetBorderWidth(Value: TBorderWidth);
```

C++

```
__fastcall SetBorderWidth(TBorderWidth Value);
```

Description

This is SetBorderWidth, a member of class TDecisionPivot.

110.1.1.4.10 TDecisionPivot.SetBounds Method

Resizes the decision pivot.

Pascal

```
procedure SetBounds(Left: Integer; Top: Integer; Height: Integer; Width: Integer); override;
```

C++

```
virtual __fastcall SetBounds(int Left, int Top, int Height, int Width);
```

Description

Use SetBounds to change all of the decision pivot's boundary properties at once. The same effect can be achieved by setting the Left, Top, Width, and Height properties separately. By setting all four properties at once, SetBounds ensures that the control will not repaint between changes. Specify the values for the Left, Top, Width, and Height properties as the value of the Left, Top, Width, and Height parameters, respectively.

SetBounds overrides the inherited method to adjust the layout of the buttons in the decision pivot when the size changes.

See Also

ButtonAutoSize (see page 17704)

Height (see page 5825)

Left (see page 5830)

Top (see page 5844)

Width (see page 5849)

110.1.1.5 TDecisionPivot Properties

The properties of the TDecisionPivot class are listed here.

Protected Properties

	Name	Description
R	Cols (see page 17707)	This is Cols, a member of class TDecisionPivot.
R	Extras (see page 17710)	This is Extras, a member of class TDecisionPivot.
R	Pages (see page 17713)	This is Pages, a member of class TDecisionPivot.
R	Rows (see page 17719)	This is Rows, a member of class TDecisionPivot.

Published Properties

	Name	Description
	Align (see page 17696)	Determines how the control aligns within its container (parent control).
	Alignment (see page 17700)	Determines how the caption is aligned within the panel.
	BevelInner (see page 17700)	Determines the style of the inner bevel of a panel.
	BevelOuter (see page 17701)	Determines the style of the outer bevel of a panel.
	BevelWidth (see page 17702)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
	BorderStyle (see page 17703)	Determines whether a line is drawn around the perimeter of the decision pivot.
	BorderWidth (see page 17704)	Specifies the distance, in pixels, between the outer and inner bevels.
	ButtonAutoSize (see page 17704)	Specifies whether the dimension buttons resize automatically when the decision pivot resizes.
	ButtonHeight (see page 17705)	Specifies the height, in pixels, of the dimension buttons.
	ButtonSpacing (see page 17705)	Specifies the number of pixels that appear between dimension buttons.
	ButtonWidth (see page 17705)	Specifies the width, in pixels, of the dimension buttons.
	Color (see page 17706)	Specifies the background color of the control.
	Ctl3D (see page 17707)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DecisionSource (see page 17707)	Specifies the decision source object that the decision pivot controls.
	DragCursor (see page 17708)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 17708)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 17710)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 17710)	Controls the attributes of text written on or in the control.
	GroupLayout (see page 17711)	Specifies how the dimension buttons and summary control are arranged on the decision pivot.
	Groups (see page 17712)	Specifies which controls appear in a decision pivot.
	GroupSpacing (see page 17712)	Determines the number of pixels that appear between control groups.
	ParentColor (see page 17713)	Determines where a control looks for its color information.
	ParentCtl3D (see page 17713)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 17714)	Determines where a control looks for its font information.
	ParentShowHint (see page 17714)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 17715)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 17720)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 17721)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 17722)	Determines if the user can tab to a control.
	Visible (see page 17722)	Determines whether the component appears onscreen.

110.1.1.5.1 TDecisionPivot.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;
```

```
  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
```

```
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
```

```
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

110.1.1.5.2 TDecisionPivot.Alignment Property

Determines how the caption is aligned within the panel.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use the Alignment property to specify if the caption of a panel appears on the left, right, or center of the panel. For example, to use a panel as a status bar that displays the TApplication. ::Hint property as the panel's Caption property, set Alignment to taLeftJustify to place the hint text on the left side of the panel.

Note: Use the VerticalAlignment property to control the vertical position of the caption.

See Also

Hint (see page 13712)

Caption (see page 5806)

110.1.1.5.3 TDecisionPivot.BevelInner Property

Determines the style of the inner bevel of a panel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TPanelBevel BevelInner;__property TBevelCut BevelInner;
```

Description

Use BevelInner to provide the panel with a three-dimensional beveled look. A panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel, separated by the number of pixels specified in the BorderWidth property. The width of the bevels, if they are drawn, is the BevelWidth property.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelOuter (see page 12541)

BevelWidth (see page 12542)

BorderWidth (see page 12543)

110.1.1.5.4 TDecisionPivot.BevelOuter Property

Determines the style of the outer bevel of a panel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TPanelBevel BevelOuter;__property TBevelCut BevelOuter;
```

Description

Use BevelOuter to provide the panel with a three-dimensional beveled look. A panel component has two bevels, an outer bevel drawn next to the border of the control, and an inner bevel drawn inside the outer bevel, separated by the number of pixels specified in the BorderWidth property. The width of the outer bevel is specified in the BevelWidth property in pixels.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelInner (see page 12540)

BevelWidth (see page 12542)

BorderWidth (see page 12543)

110.1.1.5.5 TDecisionPivot.BevelWidth Property

Determines the width, in pixels, of both the inner and outer bevels of a panel.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify how wide the inner or outer bevel should be. Do not confuse BevelWidth, which is the width of the bevels, with BorderWidth, which is the space between the bevels.

If both the BevelInner and BevelOuter properties are bvNone, BevelWidth has no effect. To remove both bevels, set the BevelInner and BevelOuter properties to bvNone, rather than setting the BevelWidth to 0, as this involves less overhead when painting.

C++ Examples:

```
/*
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
*/
void __fastcall TForm1::CreateStatusLineClick(TObject *Sender)
{
  Panel1->Align = alBottom;
  Panel1->BevelInner = bvLowered;
  Panel1->BevelOuter = bvRaised;
  Panel1->BorderWidth = 1;
  Panel1->BevelWidth = 1;
}
```

Delphi Examples:

```
{
This example uses a panel component and a button named
CreateStatusLine on a form. The code moves the panel to the
bottom of the form when the user clicks the button and gives
the panel the appearance of a status line by changing the
value of the BevelInner, BevelOuter, BevelWidth, and
BorderWidth properties.
}
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

See Also

BevelInner (see page 12540)

BevelOuter (see page 12541)

BorderWidth (see page 12543)

110.1.1.5.6 TDecisionPivot.BorderStyle Property

Determines whether a line is drawn around the perimeter of the decision pivot.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to specify whether the decision pivot has a single line drawn around it. When BorderStyle is bsSingle, a single line is drawn around the panel outside of the outer bevel.

Note: Do not confuse the line drawn around the decision pivot with the BorderWidth (see page 17704) property. The BorderWidth (see page 17704) is the distance between the outer and inner bevels.

See Also

BorderWidth (see page 17704)

110.1.1.5.7 TDecisionPivot.BorderWidth Property

Specifies the distance, in pixels, between the outer and inner bevels.

Pascal

```
property BorderWidth: TBorderWidth;
```

C++

```
__property TBorderWidth BorderWidth;
```

Description

Use BorderWidth to specify how wide the border around the decision pivot should be. A value of 0 means no border should appear.

The border of the decision pivot is the area between the outer and inner bevels. It is visible only if the inner bevel is raised or lowered, but affects the inset of the controls in the pivot even if BevelInner (see page 17700) is bvNone.

Do not confuse the border of the pivot with line drawn around it. The line around the pivot is specified by the BorderStyle (see page 17703) property.

See Also

BorderStyle (see page 17703)

BevelInner (see page 12540)

BevelOuter (see page 12541)

110.1.1.5.8 TDecisionPivot.ButtonAutoSize Property

Specifies whether the dimension buttons resize automatically when the decision pivot resizes.

Pascal

```
property ButtonAutoSize: Boolean;
```

C++

```
__property Boolean ButtonAutoSize;
```

Description

Set ButtonAutoSize to true to cause the dimension buttons to expand and contract automatically when the size of the decision

pivot changes. Set ButtonAutoSize to false to specify fixed size buttons.

Note: The value of ButtonAutoSize is ignored when GroupLayout (see page 17711) is xtTopLeft.

See Also

ButtonHeight (see page 17705)

ButtonWidth (see page 17705)

GroupLayout (see page 17711)

110.1.1.5.9 TDecisionPivot.ButtonHeight Property

Specifies the height, in pixels, of the dimension buttons.

Pascal

```
property ButtonHeight: Integer;
```

C++

```
__property int ButtonHeight;
```

Description

Use ButtonHeight to determine the Height of the dimension buttons. Do not set ButtonHeight when the ButtonAutoSize (see page 17704) property is true. When ButtonAutoSize (see page 17704) is true, the height of the buttons is reset automatically whenever the layout of the decision pivot changes.

See Also

ButtonAutoSize (see page 17704)

ButtonWidth (see page 17705)

110.1.1.5.10 TDecisionPivot.ButtonSpacing Property

Specifies the number of pixels that appear between dimension buttons.

Pascal

```
property ButtonSpacing: Integer;
```

C++

```
__property int ButtonSpacing;
```

Description

Set ButtonSpacing to control the spacing between the buttons in a single group. Increase ButtonSpacing to make a decision pivot look less crowded. Decrease ButtonSpacing to make the row and column groups look more compact.

See Also

GroupSpacing (see page 17712)

110.1.1.5.11 TDecisionPivot.ButtonWidth Property

Specifies the width, in pixels, of the dimension buttons.

Pascal

```
property ButtonWidth: Integer;
```

C++

```
__property int ButtonWidth;
```

Description

Use ButtonWidth to determine the Width of the dimension buttons. Do not set ButtonWidth when the ButtonAutoSize (see page 17704) property is true. When ButtonAutoSize (see page 17704) is true, the width of the buttons is reset automatically whenever the layout of the decision pivot changes.

See Also

ButtonAutoSize (see page 17704)

ButtonHeight (see page 17705)

110.1.1.5.12 TDecisionPivot.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
```

```
end;
```

See Also

ParentColor (see page 5834)

110.1.1.5.13 TDecisionPivot.Cols Property

Pascal

```
property Cols: Integer;
```

C++

```
__property int Cols;
```

Description

This is Cols, a member of class TDecisionPivot.

110.1.1.5.14 TDecisionPivot.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

110.1.1.5.15 TDecisionPivot.DecisionSource Property

Specifies the decision source object that the decision pivot controls.

Pascal

```
property DecisionSource: TDecisionSource;
```

C++

```
__property TDecisionSource DecisionSource;
```

Description

Set DecisionSource to link the decision pivot to a decision source component. The controls in the decision pivot change the pivot state of the decision source.

The pivot state of the decision source can also change in response to another control, such as a decision grid. When the pivot state of the decision source changes, the decision pivot updates the layout of the dimension buttons and the selection in the summary combo box.

See Also

TDecisionGrid (see page 17586)

110.1.1.5.16 TDecisionPivot.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

110.1.1.5.17 TDecisionPivot.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
```

```
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

110.1.1.5.18 TDecisionPivot.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

110.1.1.5.19 TDecisionPivot.Extras Property

Pascal

```
property Extras: Integer;
```

C++

```
__property int Extras;
```

Description

This is Extras, a member of class TDecisionPivot.

110.1.1.5.20 TDecisionPivot.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

110.1.1.5.21 TDecisionPivot.GroupLayout Property

Specifies how the dimension buttons and summary control are arranged on the decision pivot.

Pascal

```
property GroupLayout: TDecisionButtonPosition;
```

C++

```
__property TDecisionButtonPosition GroupLayout;
```

Description

Set Layout to specify the organization of a decision pivot. Layout can have one of the following values:

Value	Meaning
xtHorizontal	The controls in the pivot are lined up side by side.
xtVertical	The controls in the pivot are lined up top to bottom.
xtLeftTop	The controls in the pivot are lined up along the edges of the pivot, with the row dimensions down the left side, the column dimensions along the top, and the summary control in the upper left corner.

When Layout is xtLeftTop, the center of the decision pivot is empty. A decision graph or decision grid can be placed in this empty spot so that the buttons line up with the dimensions they control.

When Layout is xtHorizontal or xtVertical, the summary control appears first, followed by a the row dimensions and then the column dimensions. The Groups (see page 17712) property can omit any of these from the layout.

See Also

Groups (see page 17712)

110.1.1.5.22 TDecisionPivot.Groups Property

Specifies which controls appear in a decision pivot.

Pascal

```
property Groups: TDecisionPivotOptions;
```

C++

```
__property TDecisionPivotOptions Groups;
```

Description

Set Groups to limit the types of controls that appear in the decision pivot. This allows greater flexibility over where the controls appear in a form. For example, a form can contain three decision pivots: one containing only row dimension buttons, one containing only column dimension buttons, and one containing only the summary selection combo box. These three pivots can then be arranged anywhere on the form.

Groups is a set containing one or more of the following values:

Value	Meaning
xtRows	The pivot contains row dimension buttons.
xtColumns	The pivot contains column dimension buttons.
xtSummaries	The pivot contains a summary selection combo box.

See Also

GroupLayout (see page 17711)

110.1.1.5.23 TDecisionPivot.GroupSpacing Property

Determines the number of pixels that appear between control groups.

Pascal

```
property GroupSpacing: Integer;
```

C++

```
__property int GroupSpacing;
```

Description

Set GroupSpacing to control the spaces that appear between the summary combo box, the group of row dimension buttons, and the group of column dimension buttons. Increase GroupSpacing to make the groups of controls appear more distinct. Decrease GroupSpacing to make the decision pivot look more compact.

See Also

ButtonSpacing (see page 17705)

Groups (see page 17712)

110.1.1.5.24 TDecisionPivot.Pages Property

Pascal

```
property Pages: Integer;
```

C++

```
__property int Pages;
```

Description

This is Pages, a member of class TDecisionPivot.

110.1.1.5.25 TDecisionPivot.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

110.1.1.5.26 TDecisionPivot.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

110.1.1.5.27 TDecisionPivot.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

110.1.1.5.28 TDecisionPivot.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

110.1.1.5.29 TDecisionPivot.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
```

```
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Cut1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      const int num_items = 4;
      for (int index = 0; index < num_items; ++index)
      {
          TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
          PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
          NewItem->Caption = "Menu Item " + IntToStr(index);
          NewItem->Tag = index;
          NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
          TNotifyEvent();
      }
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
          {
```

```
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
```

```
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
```

```
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

110.1.1.5.30 TDecisionPivot.Rows Property

Pascal

```
property Rows: Integer;
```

C++

```
__property int Rows;
```

Description

This is Rows, a member of class TDecisionPivot.

110.1.1.5.31 TDecisionPivot.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

110.1.1.5.32 TDecisionPivot.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

110.1.1.5.33 TDecisionPivot.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

110.1.1.5.34 TDecisionPivot.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
```

```
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

110.1.1.6 TDecisionPivot Events

The events of the TDecisionPivot class are listed here.

Published Events

	Name	Description
	OnClick (see page 17724)	Occurs when the user clicks the control.
	OnDblClick (see page 17726)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 17726)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 17728)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 17731)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 17732)	Occurs when a control receives the input focus.
	OnExit (see page 17734)	Occurs when the input focus shifts away from one control to another.
	OnResize (see page 17736)	Occurs immediately after the control is resized.
	OnStartDrag (see page 17737)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

110.1.1.6.1 TDecisionPivot.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
```

```
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

110.1.1.6.2 TDecisionPivot.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

110.1.1.6.3 TDecisionPivot.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
```

```
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

110.1.1.6.4 TDecisionPivot.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

110.1.1.6.5 TDecisionPivot.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
```

```
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

110.1.1.6.6 TDecisionPivot.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;
```

```
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

110.1.1.6.7 TDecisionPivot.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
```

```
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

110.1.1.6.8 TDecisionPivot.OnResize Event

Occurs immediately after the control is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

Note: Some controls call this event handler when the control is first created.

Delphi Examples:

```
{
The following code keeps the right edge on Button1 against
the right edge of Form1 when Form1 is resized.
}
procedure TForm1.FormResize(Sender: TObject);
begin
  Button1.Left := Form1.ClientWidth-Button1.Width;
end;
```

See Also

OnCanResize (see page 5851)

SetBounds (see page 5783)

AutoSize (see page 5803)

OnConstrainedResize (see page 5853)

110.1.1.6.9 TDecisionPivot.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

110.1.2 TPivotDataLink Class

Class Hierarchy

Pascal

```
TPivotDataLink = class(TDecisionDataLink);
```

C++

```
class TPivotDataLink : public TDecisionDataLink;
```

File

MXPIVSRC

Description

This is class MXPIVSRC.TPivotDataLink.

110.1.2.1 TPivotDataLink Members

The following tables list the members exposed by TPivotDataLink.

Protected Fields

	Name	Description
	FBlocked (see page 17460)	This is FBlocked, a member of class TDecisionDataLink.
	FDecisionSource (see page 17461)	This is FDecisionSource, a member of class TDecisionDataLink.

Protected Methods

	Name	Description
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	DecisionDataEvent (see page 17739)	This is DecisionDataEvent, a member of class TPivotDataLink.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetDecisionSource (see page 17461)	This is SetDecisionSource, a member of class TDecisionDataLink.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 17739)	This is Create, a member of class TPivotDataLink.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 17739)	Destroys the TPersistent instance and frees its memory.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	DecisionSource (see page 17462)	This is DecisionSource, a member of class TDecisionDataLink.

110.1.2.2 TPivotDataLink.Create Constructor

Pascal

```
constructor Create(aPivot: TDecisionPivot);
```

C++

```
__fastcall TPivotDataLink(TDecisionPivot aPivot);
```

Description

This is Create, a member of class TPivotDataLink.

110.1.2.3 TPivotDataLink.Destroy Destructor

Destroys the TPersistent instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPivotDataLink();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

Free (see page 22300)

Destroy (see page 22287)

110.1.2.4 TPivotDataLink Methods

The methods of the TPivotDataLink class are listed here.

Protected Methods

	Name	Description
	DecisionDataEvent (see page 17739)	This is DecisionDataEvent, a member of class TPivotDataLink.

110.1.2.4.1 TPivotDataLink.DecisionDataEvent Method

Pascal

```
procedure DecisionDataEvent(Event: TDecisionDataEvent); override;
```

C++

```
virtual __fastcall DecisionDataEvent(TDecisionDataEvent Event);
```

Description

This is DecisionDataEvent, a member of class TPivotDataLink.

110.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TDecisionButtonPosition (see page 17741)	This is record MXPIVSRC.TDecisionButtonPosition.
	TDecisionPivotOption (see page 17741)	This is record MXPIVSRC.TDecisionPivotOption.

110.2.1 MXPIVSRC.TDecisionButtonPosition Enumeration

Pascal

```
TDecisionButtonPosition = (
  xtHorizontal,
  xtVertical,
  xtLeftTop
);
```

C++

```
enum TDecisionButtonPosition {
  xtHorizontal,
  xtVertical,
  xtLeftTop
};
```

File

MXPIVSRC

Description

This is record MXPIVSRC.TDecisionButtonPosition.

110.2.2 MXPIVSRC.TDecisionPivotOption Enumeration

Pascal

```
TDecisionPivotOption = (
  xtRows,
  xtColumns,
  xtSummaries
);
```

C++

```
enum TDecisionPivotOption {
  xtRows,
  xtColumns,
  xtSummaries
};
```

File

MXPIVSRC

Description

This is record MXPIVSRC.TDecisionPivotOption.

110.3 Types

The following table lists types in this documentation.

Types

Name	Description
TDecisionPivotOptions (see page 17743)	This is type MXPIVSRC.TDecisionPivotOptions.

110.3.1 MXPIVSRC.TDecisionPivotOptions Type

Pascal

```
TDecisionPivotOptions = set of TDecisionPivotOption;
```

C++

```
TDecisionPivotOption TDecisionPivotOptions;
```

File

MXPIVSRC

Description

This is type MXPIVSRC.TDecisionPivotOptions.

111 Mxstore Namespace

This is namespace Mxstore.

Topics

Name	Description
Classes (see page 17747)	The following table lists classes in this documentation.
Functions (see page 17836)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 17837)	The following table lists structs, records, enums in this documentation.
Types (see page 17843)	The following table lists types in this documentation.
Constants (see page 17849)	The following table lists constants in this documentation.

Classes

	Name	Description
	ECacheError (see page 17747)	ECacheError is the exception class for failures that occur when building the cache of a decision cube.
	TAggDefinition (see page 17748)	This is class Mxstore.TAggDefinition.
	TBuilderDim (see page 17749)	This is class Mxstore.TBuilderDim.
	TCubeDim (see page 17754)	TCubeDim represents an entry in the dimension map of a decision cube.
	TCubeDims (see page 17766)	TCubeDims is a collection of TCubeDim (see page 17754) objects.
	TCustomDataStore (see page 17772)	TCustomDataStore is the base class for components that analyze and maintain crosstabulated data.
	TDataCache (see page 17797)	This is class Mxstore.TDataCache.
	TDimension (see page 17809)	This is class Mxstore.TDimension.
	TFieldDefinition (see page 17814)	This is class Mxstore.TFieldDefinition.
	TIndexInfo (see page 17818)	This is class Mxstore.TIndexInfo.
	TMultiDimDataLink (see page 17822)	This is class Mxstore.TMultiDimDataLink.
	TSummary (see page 17828)	This is class Mxstore.TSummary.

Constants

Name	Description
LargeValueCount (see page 17849)	This is constant Mxstore.LargeValueCount.
MaxBinDimensions (see page 17849)	This is constant Mxstore.MaxBinDimensions.
NonSparseAgg (see page 17850)	This is constant Mxstore.NonSparseAgg.
SparseAgg (see page 17850)	This is constant Mxstore.SparseAgg.
SparseSum (see page 17850)	This is constant Mxstore.SparseSum.
SparseUnknown (see page 17851)	This is constant Mxstore.SparseUnknown.

SubTotal (see page 17851)	This is constant Mxstore.SubTotal.

Functions

	Name	Description
	TestMatch (see page 17836)	This is function Mxstore.TestMatch.

Structs, Records, Enums

	Name	Description
	TBuildType (see page 17837)	This is record Mxstore.TBuildType.
	TCacheStateFlags (see page 17837)	This is record Mxstore.TCacheStateFlags.
	TCalcTotalsFlags (see page 17838)	This is record Mxstore.TCalcTotalsFlags.
	TCubeDataState (see page 17838)	This is record Mxstore.TCubeDataState.
	TCubeState (see page 17839)	This is record Mxstore.TCubeState.
	TErrorAction (see page 17839)	This is record Mxstore.TErrorAction.
	TFormatType (see page 17840)	This is record Mxstore.TFormatType.
	TIndexFlag (see page 17840)	This is record Mxstore.TIndexFlag.
	TIndexInfoRec (see page 17841)	This is record Mxstore.TIndexInfoRec.
	TLookupStateFlags (see page 17841)	This is record Mxstore.TLookupStateFlags.

Types

Name	Description
PIndexInfoRec (see page 17843)	This is type Mxstore.PIndexInfoRec.
TAggProc (see page 17843)	This is type Mxstore.TAggProc.
TCacheState (see page 17844)	This is type Mxstore.TCacheState.
TCalcTotals (see page 17844)	This is type Mxstore.TCalcTotals.
TCapacityErrorEvent (see page 17844)	This is type Mxstore.TCapacityErrorEvent.
TCubeDimClass (see page 17845)	This is type Mxstore.TCubeDimClass.
TCubeDimTransformEvent (see page 17845)	This is type Mxstore.TCubeDimTransformEvent.
TCubeNotifyEvent (see page 17845)	This is type Mxstore.TCubeNotifyEvent.
TCubeRefreshEvent (see page 17846)	This is type Mxstore.TCubeRefreshEvent.
TDerivedAggProc (see page 17846)	This is type Mxstore.TDerivedAggProc.
TDimFlagSet (see page 17846)	This is type Mxstore.TDimFlagSet.
TFieldArgs (see page 17847)	This is type Mxstore.TFieldArgs.
TIndexFlags (see page 17847)	This is type Mxstore.TIndexFlags.
TLookupState (see page 17847)	This is type Mxstore.TLookupState.
TSumMethod (see page 17848)	This is type Mxstore.TSumMethod.

111.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ECacheError (see page 17747)	ECacheError is the exception class for failures that occur when building the cache of a decision cube.
	TAggDefinition (see page 17748)	This is class Mxstore.TAggDefinition.
	TBuilderDim (see page 17749)	This is class Mxstore.TBuilderDim.
	TCubeDim (see page 17754)	TCubeDim represents an entry in the dimension map of a decision cube.
	TCubeDims (see page 17766)	TCubeDims is a collection of TCubeDim (see page 17754) objects.
	TCustomDataStore (see page 17772)	TCustomDataStore is the base class for components that analyze and maintain crosstabulated data.
	TDataCache (see page 17797)	This is class Mxstore.TDataCache.
	TDimension (see page 17809)	This is class Mxstore.TDimension.
	TFieldDefinition (see page 17814)	This is class Mxstore.TFieldDefinition.
	TIndexInfo (see page 17818)	This is class Mxstore.TIndexInfo.
	TMultiDimDataLink (see page 17822)	This is class Mxstore.TMultiDimDataLink.
	TSummary (see page 17828)	This is class Mxstore.TSummary.

111.1.1 ECacheError Class

ECacheError is the exception class for failures that occur when building the cache of a decision cube.

Class Hierarchy

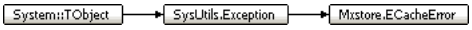

Pascal

```
ECacheError = class(Exception);
```

C++

```
class ECacheError : public Exception;
```

File

Mxstore

Description

ECacheError is raised when

The dataset for a decision cube includes too many fields.

The dataset for a decision cube does not include the necessary summaries for computing a derived value such as an average.

A dimension item in the dimension map specifies a date range for dimension that does not represent a date or date-time field.

See Also

TDecisionCube (see page 17442)

111.1.2 TAggDefinition Class

Class Hierarchy

Mxstore.TAggDefinition

Pascal

```
TAggDefinition = class;
```

C++

```
class TAggDefinition;
```

File

Mxstore

Description

This is class Mxstore.TAggDefinition.

111.1.2.1 TAggDefinition Members

The following tables list the members exposed by TAggDefinition.

Public Properties

	Name	Description
	AggProc (see page 17748)	This is AggProc, a member of class TAggDefinition.

111.1.2.2 TAggDefinition Properties

The properties of the TAggDefinition class are listed here.

Public Properties

	Name	Description
	AggProc (see page 17748)	This is AggProc, a member of class TAggDefinition.

111.1.2.2.1 TAggDefinition.AggProc Property

Pascal

```
property AggProc: TDerivedAggProc;
```

C++

```
__property TDerivedAggProc AggProc;
```

Description

This is AggProc, a member of class TAggDefinition.

111.1.3 TBuilderDim Class

Class Hierarchy

Pascal

```
TBuilderDim = class(TDimension);
```

C++

```
class TBuilderDim : public TDimension;
```

File

Mxstore

Description

This is class Mxstore.TBuilderDim.

111.1.3.1 TBuilderDim Members

The following tables list the members exposed by TBuilderDim.

Protected Methods

	Name	Description
	GetCompProc (see page 17330)	This is GetCompProc, a member of class TCustomArray.
	GetDups (see page 17331)	This is GetDups, a member of class TCustomArray.
	GetItem (see page 17332)	This is GetItem, a member of class TCustomArray.
	GetLastVal (see page 17751)	This is GetLastVal, a member of class TBuilderDim.
	GetMemberCount (see page 17332)	This is GetMemberCount, a member of class TCustomArray.
	GetSort (see page 17332)	This is GetSort, a member of class TCustomArray.
	SetCompProc (see page 17333)	This is SetCompProc, a member of class TCustomArray.
	SetDups (see page 17333)	This is SetDups, a member of class TCustomArray.
	SetItem (see page 17333)	This is SetItem, a member of class TCustomArray.
	SetLastVal (see page 17753)	This is SetLastVal, a member of class TBuilderDim.
	SetSort (see page 17334)	This is SetSort, a member of class TCustomArray.

Public Methods

	Name	Description
	Add (see page 17751)	This is Add, a member of class TBuilderDim.
	AddSummary (see page 17751)	This is AddSummary, a member of class TBuilderDim.
	Assign (see page 17330)	This is Assign, a member of class TCustomArray.
	AssignSorted (see page 17811)	This is AssignSorted, a member of class TDimension.
	ClearFlag (see page 17811)	This is ClearFlag, a member of class TDimension.
	ConvertVar (see page 17330)	This is ConvertVar, a member of class TCustomArray.
	Create (see page 17750)	This is Create, a member of class TBuilderDim.
V	Destroy (see page 17750)	This is Destroy, a member of class TBuilderDim.
	GetCurrency (see page 17331)	This is GetCurrency, a member of class TCustomArray.
	GetDouble (see page 17331)	This is GetDouble, a member of class TCustomArray.
	GetInteger (see page 17331)	This is GetInteger, a member of class TCustomArray.
	GetSumCount (see page 17752)	This is GetSumCount, a member of class TBuilderDim.
	GetSummary (see page 17752)	This is GetSummary, a member of class TBuilderDim.

	HasFlag (see page 17812)	This is HasFlag, a member of class TDimension.
	IndexOf (see page 17332)	This is IndexOf, a member of class TCustomArray.
	InitSummary (see page 17752)	This is InitSummary, a member of class TBuilderDim.
	IsBlank (see page 17332)	This is IsBlank, a member of class TCustomArray.
	IsString (see page 17812)	This is IsString, a member of class TDimension.
	MatchLastVal (see page 17752)	This is MatchLastVal, a member of class TBuilderDim.
V	MemoryUsage (see page 17333)	This is MemoryUsage, a member of class TCustomArray.
	SetFlag (see page 17812)	This is SetFlag, a member of class TDimension.
	SetRangeCounting (see page 17812)	This is SetRangeCounting, a member of class TDimension.
	SetSize (see page 17334)	This is SetSize, a member of class TCustomArray.

Public Properties

	Name	Description
W	Attributes (see page 17813)	This is Attributes, a member of class TDimension.
	BlankBoolVal (see page 17334)	This is BlankBoolVal, a member of class TCustomArray.
R	BlankCount (see page 17335)	This is BlankCount, a member of class TCustomArray.
	BlankDateVal (see page 17335)	This is BlankDateVal, a member of class TCustomArray.
	BlankStringVal (see page 17335)	This is BlankStringVal, a member of class TCustomArray.
	CompareProc (see page 17335)	This is CompareProc, a member of class TCustomArray.
R	DataType (see page 17335)	This is DataType, a member of class TCustomArray.
	DimensionName (see page 17813)	This is DimensionName, a member of class TDimension.
	Duplicates (see page 17336)	This is Duplicates, a member of class TCustomArray.
R	FieldDefinition (see page 17813)	This is FieldDefinition, a member of class TDimension.
	FieldName (see page 17813)	This is FieldName, a member of class TDimension.
	FieldType (see page 17814)	This is FieldType, a member of class TDimension.
	GroupBreak (see page 17753)	This is GroupBreak, a member of class TBuilderDim.
	Items (see page 17336)	This is Items, a member of class TCustomArray.
	LastVal (see page 17753)	This is LastVal, a member of class TBuilderDim.
R	List (see page 17336)	This is List, a member of class TCustomArray.
R	MemberCount (see page 17336)	This is MemberCount, a member of class TCustomArray.
	Position (see page 17814)	This is Position, a member of class TDimension.
	Range (see page 17814)	This is Range, a member of class TDimension.
	Sorted (see page 17337)	This is Sorted, a member of class TCustomArray.
R	SumCount (see page 17753)	This is SumCount, a member of class TBuilderDim.

111.1.3.2 TBuilderDim.Create Constructor

Pascal

```
constructor Create(Items: Cardinal; DataType: TFieldType);
```

C++

```
__fastcall TBuilderDim(unsigned Items, TFieldType DataType);
```

Description

This is Create, a member of class TBuilderDim.

111.1.3.3 TBuilderDim.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBuilderDim();
```

Description

This is Destroy, a member of class TBuilderDim.

111.1.3.4 TBuilderDim Methods

The methods of the TBuilderDim class are listed here.

Protected Methods

	Name	Description
	GetLastVal (see page 17751)	This is GetLastVal, a member of class TBuilderDim.
	SetLastVal (see page 17753)	This is SetLastVal, a member of class TBuilderDim.

Public Methods

	Name	Description
	Add (see page 17751)	This is Add, a member of class TBuilderDim.
	AddSummary (see page 17751)	This is AddSummary, a member of class TBuilderDim.
	GetSumCount (see page 17752)	This is GetSumCount, a member of class TBuilderDim.
	GetSummary (see page 17752)	This is GetSummary, a member of class TBuilderDim.
	InitSummary (see page 17752)	This is InitSummary, a member of class TBuilderDim.
	MatchLastVal (see page 17752)	This is MatchLastVal, a member of class TBuilderDim.

111.1.3.4.1 TBuilderDim.Add Method

Pascal

```
procedure Add(Value: Variant);
```

C++

```
__fastcall Add(Variant Value);
```

Description

This is Add, a member of class TBuilderDim.

111.1.3.4.2 TBuilderDim.AddSummary Method

Pascal

```
procedure AddSummary(Value: Variant);
```

C++

```
__fastcall AddSummary(Variant Value);
```

Description

This is AddSummary, a member of class TBuilderDim.

111.1.3.4.3 TBuilderDim.GetLastVal Method

Pascal

```
function GetLastVal: Variant;
```

C++

```
__fastcall Variant GetLastVal();
```

Description

This is GetLastVal, a member of class TBuilderDim.

111.1.3.4.4 TBuilderDim.GetSumCount Method

Pascal

```
function GetSumCount: Integer;
```

C++

```
__fastcall int GetSumCount();
```

Description

This is GetSumCount, a member of class TBuilderDim.

111.1.3.4.5 TBuilderDim.GetSummary Method

Pascal

```
function GetSummary(Value: Variant): Variant;
```

C++

```
__fastcall Variant GetSummary(Variant Value);
```

Description

This is GetSummary, a member of class TBuilderDim.

111.1.3.4.6 TBuilderDim.InitSummary Method

Pascal

```
procedure InitSummary(DataType: Integer);
```

C++

```
__fastcall InitSummary(int DataType);
```

Description

This is InitSummary, a member of class TBuilderDim.

111.1.3.4.7 TBuilderDim.MatchLastVal Method

Pascal

```
function MatchLastVal(Value: Variant): Boolean;
```

C++

```
__fastcall Boolean MatchLastVal(Variant Value);
```

Description

This is MatchLastVal, a member of class TBuilderDim.

111.1.3.4.8 TBuilderDim.SetLastVal Method

Pascal

```
procedure SetLastVal(Value: Variant);
```

C++

```
__fastcall SetLastVal(Variant Value);
```

Description

This is SetLastVal, a member of class TBuilderDim.

111.1.3.5 TBuilderDim Properties

The properties of the TBuilderDim class are listed here.

Public Properties

	Name	Description
	GroupBreak (see page 17753)	This is GroupBreak, a member of class TBuilderDim.
	LastVal (see page 17753)	This is LastVal, a member of class TBuilderDim.
R	SumCount (see page 17753)	This is SumCount, a member of class TBuilderDim.

111.1.3.5.1 TBuilderDim.GroupBreak Property

Pascal

```
property GroupBreak: Boolean;
```

C++

```
__property Boolean GroupBreak;
```

Description

This is GroupBreak, a member of class TBuilderDim.

111.1.3.5.2 TBuilderDim.LastVal Property

Pascal

```
property LastVal: Variant;
```

C++

```
__property Variant LastVal;
```

Description

This is LastVal, a member of class TBuilderDim.

111.1.3.5.3 TBuilderDim.SumCount Property

Pascal

```
property SumCount: Integer;
```

C++

```
__property int SumCount;
```

Description

This is SumCount, a member of class TBuilderDim.

111.1.4 TCubeDim Class

TCubeDim represents an entry in the dimension map of a decision cube.

Class Hierarchy

System::TObject → Classes.TPersistent → Classes.TCollectionItem → MXCOMMON.TDimensionItem → Mxstore.TCubeDim

Pascal

```
TCubeDim = class(TDimensionItem);
```

C++

```
class TCubeDim : public TDimensionItem;
```

File

Mxstore

Description

Use TCubeDim to obtain information about the dimensions or summaries maintained by a decision cube. The decision cube maintains a collection of TCubeDim objects that describe the fields over which the information in the decision cube is summarized and the fields that supply those summarized values. The properties of TCubeDim can be used to change the display names of the dimensions of the decision cube, or to expand or collapse individual dimensions of the multidimensional array of values cached by the decision cube.

TCubeDim introduces properties and methods to allow a decision cube to include date ranges as the values along one of its dimensions.

See Also

TCubeDims (see page 17766)

TDecisionCube (see page 17442)

111.1.4.1 TCubeDim Members

The following tables list the members exposed by TCubeDim.

Protected Methods

	Name	Description
	AssignBinTypeFormat (see page 17758)	This is AssignBinTypeFormat, a member of class TCubeDim.
	AssignBinTypeTransform (see page 17758)	This is AssignBinTypeTransform, a member of class TCubeDim.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DataSetTransform (see page 17759)	This is DataSetTransform, a member of class TCubeDim.
V	DoTransform (see page 17759)	This is DoTransform, a member of class TCubeDim.
V	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
	GetLoaded (see page 17760)	This is GetLoaded, a member of class TCubeDim.
V	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.

	InitializeRange (see page 17760)	This is InitializeRange, a member of class TCubeDim.
	MonthTransform (see page 17761)	This is MonthTransform, a member of class TCubeDim.
	NotifyCollection (see page 17761)	This is NotifyCollection, a member of class TCubeDim.
	QuarterTransform (see page 17761)	This is QuarterTransform, a member of class TCubeDim.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.
	SetLoaded (see page 17762)	This is SetLoaded, a member of class TCubeDim.
	YearTransform (see page 17762)	This is YearTransform, a member of class TCubeDim.

Protected Properties

	Name	Description
	Active (see page 17411)	This is Active, a member of class TDimensionItem.
	Dirty (see page 17764)	This is Dirty, a member of class TCubeDim.
R	Owner (see page 17414)	This is Owner, a member of class TDimensionItem.
	wasActive (see page 17766)	This is wasActive, a member of class TCubeDim.

Public Events

	Name	Description
	OnTransform (see page 17766)	This is OnTransform, a member of class TCubeDim.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 17757)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 17756)	Creates and initializes a TCollectionItem instance.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DefineProperties (see page 17759)	Designates methods for storing the dimension's unpublished data on a stream such as a form file.
	Destroy (see page 17757)	Destroys the TCollectionItem instance and frees its memory.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetBinValues (see page 17760)	Returns the set of values that are summarized by a single value along a dimension.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	IsBinData (see page 17761)	Indicates whether the dimension includes a member for every field value.
	IsDimension (see page 17409)	Indicates whether the dimension item represents a dimension.
	IsSummary (see page 17410)	Indicates whether the dimension item represents a summary.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	BinData (see page 17762)	Represents information about the implementation of paged dimensions.
	BinFormat (see page 17763)	Indicates how range values are formatted.
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
	Loaded (see page 17764)	Indicates whether the dimension item is currently cached by the decision cube.
	StartDate (see page 17765)	Determines how values are grouped along a dimension.
	StartValue (see page 17765)	Determines how values are grouped along a dimension.

Published Properties

	Name	Description
	ActiveFlag (see page 17411)	Indicates whether the dimension item is stored in the cache of a crosstabulated data store.
	BaseName (see page 17411)	Specifies the field name of the database field that is represented by or summarized in the dimension item.
	BinType (see page 17763)	Indicates how the values of the field represented by the dimension are grouped.
	DerivedFrom (see page 17412)	Specifies the index of a dimension item from which a derived value is calculated.
	DimensionType (see page 17412)	Indicates the use of the dimension item.
	Fieldname (see page 17413)	This is Fieldname, a member of class TDimensionItem.
	FieldType (see page 17413)	Indicates the data type of the field that the dimension item represents.
	Format (see page 17414)	Indicates how field values are formatted on dimensions that represent numeric or temporal data.
	Name (see page 17414)	Denotes the name of a dimension or summary.
	ValueCount (see page 17765)	Indicates the number of distinct values exist for the dimension.

111.1.4.2 TCubeDim.Create Constructor

Creates and initializes a TCollectionItem instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TCubeDim(TCollection Collection);
```

Description

Do not call Create to instantiate a collection item. Instead, call the Add method of the collection to which the item should belong.

Collection identifies the TCollection instance to which the new item belongs.

See Also

Add (see page 2736)

Collection (see page 2761)

111.1.4.3 TCubeDim.Destroy Destructor

Destroys the TCollectionItem instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCubeDim();
```

Description

Destroy is called indirectly by TCollection's Clear or Delete method.

See Also

Clear (see page 2741)

Delete (see page 2741)

Free (see page 22300)

111.1.4.4 TCubeDim Methods

The methods of the TCubeDim class are listed here.

Protected Methods

	Name	Description
	AssignBinTypeFormat (see page 17758)	This is AssignBinTypeFormat, a member of class TCubeDim.
	AssignBinTypeTransform (see page 17758)	This is AssignBinTypeTransform, a member of class TCubeDim.
	DataSetTransform (see page 17759)	This is DataSetTransform, a member of class TCubeDim.
V	DoTransform (see page 17759)	This is DoTransform, a member of class TCubeDim.
	GetLoaded (see page 17760)	This is GetLoaded, a member of class TCubeDim.
V	InitializeRange (see page 17760)	This is InitializeRange, a member of class TCubeDim.
	MonthTransform (see page 17761)	This is MonthTransform, a member of class TCubeDim.
V	NotifyCollection (see page 17761)	This is NotifyCollection, a member of class TCubeDim.
	QuarterTransform (see page 17761)	This is QuarterTransform, a member of class TCubeDim.
	SetLoaded (see page 17762)	This is SetLoaded, a member of class TCubeDim.
	YearTransform (see page 17762)	This is YearTransform, a member of class TCubeDim.

Public Methods

	Name	Description
V	Assign (see page 17757)	Copies the contents of another, similar object.
V	DefineProperties (see page 17759)	Designates methods for storing the dimension's unpublished data on a stream such as a form file.
	GetBinValues (see page 17760)	Returns the set of values that are summarized by a single value along a dimension.
	IsBinData (see page 17761)	Indicates whether the dimension includes a member for every field value.

111.1.4.4.1 TCubeDim.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Value: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Value);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo (see page 2931)

EConvertError (see page 22724)

111.1.4.4.2 TCubeDim.AssignBinTypeFormat Method

Pascal

```
function AssignBinTypeFormat(Bins: TBinType): string;
```

C++

```
__fastcall AnsiString AssignBinTypeFormat(TBinType Bins);
```

Description

This is AssignBinTypeFormat, a member of class TCubeDim.

111.1.4.4.3 TCubeDim.AssignBinTypeTransform Method

Pascal

```
function AssignBinTypeTransform(Bins: TBinType): TCubeDimTransformEvent;
```

C++

```
__fastcall TCubeDimTransformEvent AssignBinTypeTransform(TBinType Bins);
```

Description

This is AssignBinTypeTransform, a member of class TCubeDim.

111.1.4.4.4 TCubeDim.DataSetTransform Method

Pascal

```
procedure DataSetTransform(var Value: Variant; CubeDim: TCubeDim);
```

C++

```
__fastcall DataSetTransform(Variant Value, TCubeDim CubeDim);
```

Description

This is DataSetTransform, a member of class TCubeDim.

111.1.4.4.5 TCubeDim.DefineProperties Method

Designates methods for storing the dimension's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

Do not call DefineProperties. It is called automatically when forms or data modules containing the dimension's decision cube component are loaded or saved.

DefineProperties allows the dimension to load and save information as if it were a set of published property values. TCubeDim (see page 17754) uses this method to save some of its unpublished properties, including

an indication of whether the dimension is open or closed.

the public StartValue (see page 17765) property.

DefineProperties also allows the dimension to read information that was previously saved in other formats so as to preserve backward compatibility.

See Also

DefineProperties (see page 2931)

StartValue (see page 17765)

111.1.4.4.6 TCubeDim.DoTransform Method

Pascal

```
procedure DoTransform(var Value: Variant); virtual;
```

C++

```
virtual __fastcall DoTransform(Variant Value);
```

Description

This is DoTransform, a member of class TCubeDim.

111.1.4.4.7 TCubeDim.GetBinValues Method

Returns the set of values that are summarized by a single value along a dimension.

Pascal

```
function GetBinValues(Value: Variant): Variant;
```

C++

```
__fastcall Variant GetBinValues(Variant Value);
```

Description

Call GetBinValues to determine the actual values that underlie a single dimension member when the BinType (see page 17763) property is not binNone. The Value parameter is the single dimension member that represents a date range or the single value of a paged dimension. The value of Value can be obtained by calling the GetMemberAsVariant method of the decision cube or decision source.

If the BinType (see page 17763) property of the dimension is binYear, binQuarter, or binMonth, GetBinValues returns the starting and ending values of the range that includes the Value parameter. If the BinType (see page 17763) property is binSet, GetBinValues returns the single drilled value of the dimension.

See Also

BinType (see page 17763)

GetMemberAsVariant (see page 17782)

GetMemberAsVariant (see page 17478)

111.1.4.4.8 TCubeDim.GetLoaded Method

Pascal

```
function GetLoaded: Boolean;
```

C++

```
__fastcall Boolean GetLoaded();
```

Description

This is GetLoaded, a member of class TCubeDim.

111.1.4.4.9 TCubeDim.InitializeRange Method

Pascal

```
procedure InitializeRange; override;
```

C++

```
virtual __fastcall InitializeRange();
```

Description

This is InitializeRange, a member of class TCubeDim.

111.1.4.4.10 TCubeDim.IsBinData Method

Indicates whether the dimension includes a member for every field value.

Pascal

```
function IsBinData: Boolean;
```

C++

```
__fastcall Boolean IsBinData();
```

Description

Call IsBinData to determine if the dimension item represents a dimension of the decision cube that groups values into date ranges or represents a paged dimension. IsBinData returns true if the value of the BinType (see page 17763) property is not binNone.

See Also

BinType (see page 17763)

111.1.4.4.11 TCubeDim.MonthTransform Method

Pascal

```
procedure MonthTransform(var Value: Variant; CubeDim: TCubeDim);
```

C++

```
__fastcall MonthTransform(Variant Value, TCubeDim CubeDim);
```

Description

This is MonthTransform, a member of class TCubeDim.

111.1.4.4.12 TCubeDim.NotifyCollection Method

Pascal

```
procedure NotifyCollection(aType: TCDNotifyType); override;
```

C++

```
virtual __fastcall NotifyCollection(TCDNotifyType aType);
```

Description

This is NotifyCollection, a member of class TCubeDim.

111.1.4.4.13 TCubeDim.QuarterTransform Method

Pascal

```
procedure QuarterTransform(var Value: Variant; CubeDim: TCubeDim);
```

C++

```
__fastcall QuarterTransform(Variant Value, TCubeDim CubeDim);
```

Description

This is QuarterTransform, a member of class TCubeDim.

111.1.4.4.14 TCubeDim.SetLoaded Method

Pascal

```
procedure SetLoaded(Value: Boolean);
```

C++

```
__fastcall SetLoaded(Boolean Value);
```

Description

This is SetLoaded, a member of class TCubeDim.

111.1.4.4.15 TCubeDim.YearTransform Method

Pascal

```
procedure YearTransform(var Value: Variant; CubeDim: TCubeDim);
```

C++

```
__fastcall YearTransform(Variant Value, TCubeDim CubeDim);
```

Description

This is YearTransform, a member of class TCubeDim.

111.1.4.5 TCubeDim Properties

The properties of the TCubeDim class are listed here.

Protected Properties

	Name	Description
	Dirty (see page 17764)	This is Dirty, a member of class TCubeDim.
	wasActive (see page 17766)	This is wasActive, a member of class TCubeDim.

Public Properties

	Name	Description
R	BinData (see page 17762)	Represents information about the implementation of paged dimensions.
	BinFormat (see page 17763)	Indicates how range values are formatted.
	Loaded (see page 17764)	Indicates whether the dimension item is currently cached by the decision cube.
	StartDate (see page 17765)	Determines how values are grouped along a dimension.
	StartValue (see page 17765)	Determines how values are grouped along a dimension.

Published Properties

	Name	Description
	BinType (see page 17763)	Indicates how the values of the field represented by the dimension are grouped.
	ValueCount (see page 17765)	Indicates the number of distinct values exist for the dimension.

111.1.4.5.1 TCubeDim.BinData Property

Represents information about the implementation of paged dimensions.

Pascal

```
property BinData: TBinData;
```

C++

```
__property TBinData BinData;
```

Description

Do not use the BinData property. It is used internally to manage paged (permanently drilled) dimensions.

See Also

BinType (see page 17763)

111.1.4.5.2 TCubeDim.BinFormat Property

Indicates how range values are formatted.

Pascal

```
property BinFormat: string;
```

C++

```
__property AnsiString BinFormat;
```

Description

Set BinFormat to indicate how the labels for date ranges should appear. Use date and time formatting strings, such as those used with the FormatDateTime function.

See Also

BinType (see page 17763)

Format (see page 17414)

FormatDateTime (see page 22902)

111.1.4.5.3 TCubeDim.BinType Property

Indicates how the values of the field represented by the dimension are grouped.

Pascal

```
property BinType: TBinType;
```

C++

```
__property TBinType BinType;
```

Description

Set BinType to restrict the values that appear along the dimension. BinType can have one of the following values:

Value	Meaning
binNone	All values along the dimension appear.
binYear	Each cell along the dimension represents all field values that represent dates from the same year.
binQuarter	Each cell along the dimension represents all field values that represent dates from the same quarter.
binMonth	Each cell along the dimension represents all field values that represent dates from the same month.
binSet	The dimension is in a paged (permanently drilled) state. The drill-down value of the dimension can be changed, but the dimension can not be opened or closed.
binCustom	binCustom is not currently used but is provided to allow component writers to support other types of ranges.

If BinType is binYear, binQuarter, or binMonth, use the BinFormat (see page 17763) property to indicate how the values of the ranges along the dimension should be formatted.

If BinType is binSet, use the StartValue (see page 17765) property to specify which value along the dimension is used as the drill-down value. Unlike the drill-down value of a normal dimension, which is part of the pivot state stored by the decision source, the drill-down value of a paged dimension is the only value cached by the decision cube. Changing the drill-down value of paged dimensions causes the decision cube to rebuild its cache, which can be very time-consuming.

Note: When BinType is binSet, the dimension item consumes considerably less memory in the decision cube.

Note: Do not set BinType to binYear, binQuarter, or binMonth if the dimension does not represent a date or date-time field.

See Also

BinFormat (see page 17763)

FieldType (see page 17413)

StartValue (see page 17765)

111.1.4.5.4 TCubeDim.Dirty Property

Pascal

```
property Dirty: Boolean;
```

C++

```
__property Boolean Dirty;
```

Description

This is Dirty, a member of class TCubeDim.

111.1.4.5.5 TCubeDim.Loaded Property

Indicates whether the dimension item is currently cached by the decision cube.

Pascal

```
property Loaded: Boolean;
```

C++

```
__property Boolean Loaded;
```

Description

Read Loaded to determine whether the dimension item is currently available through the decision cube. When the ActiveFlag property is diActive, Loaded is always true. When the value of ActiveFlag is diInactive, Loaded is always false. When the value of ActiveFlag is diAsNeeded, use Loaded to determine whether the decision cube has cached the dimension item so that it is available to a decision source.

When Loaded is true, the dimension item consumes memory. When Loaded is false, the dimension item is unavailable.

See Also

ActiveFlag (see page 17411)

111.1.4.5.6 TCubeDim.StartDate Property

Determines how values are grouped along a dimension.

Pascal

```
property StartDate: TDate;
```

C++

```
__property TDate StartDate;
```

Description

When the BinType (see page 17763) property is binYear, binQuarter, or binMonth, set StartValue (see page 17765) to the first date in the first range of the dimension. If the BinType (see page 17763) property is binSet, set StartValue (see page 17765) to the drill-down value of the dimension.

See Also

BinType (see page 17763)

StartValue (see page 17765)

111.1.4.5.7 TCubeDim.StartValue Property

Determines how values are grouped along a dimension.

Pascal

```
property StartValue: String;
```

C++

```
__property AnsiString StartValue;
```

Description

When the BinType (see page 17763) property is binYear, binQuarter, or binMonth, set StartValue to the first date (expressed as a string) in the first range of the dimension. If the BinType (see page 17763) property is binSet, set StartValue to the drill-down value of the dimension.

See Also

BinType (see page 17763)

StartDate (see page 17765)

111.1.4.5.8 TCubeDim.ValueCount Property

Indicates the number of distinct values exist for the dimension.

Pascal

```
property ValueCount: Integer;
```

C++

```
__property int ValueCount;
```

Description

Read ValueCount to determine the number of distinct values for this dimension. The decision cube sets this value using its GetDimensionMemberCount method.

See Also

BinType (see page 17763)

StartValue (see page 17765)

GetDimensionMemberCount (see page 17780)

111.1.4.5.9 TCubeDim.wasActive Property

Pascal

```
property wasActive: Boolean;
```

C++

```
__property Boolean wasActive;
```

Description

This is wasActive, a member of class TCubeDim.

111.1.4.6 TCubeDim Events

The events of the TCubeDim class are listed here.

Public Events

	Name	Description
	OnTransform (see page 17766)	This is OnTransform, a member of class TCubeDim.

111.1.4.6.1 TCubeDim.OnTransform Event

Pascal

```
property OnTransform: TCubeDimTransformEvent;
```

C++

```
__property TCubeDimTransformEvent OnTransform;
```

Description

This is OnTransform, a member of class TCubeDim.

111.1.5 TCubeDims Class

TCubeDims is a collection of TCubeDim (see page 17754) objects.

Class Hierarchy

Pascal

```
TCubeDims = class(TDimensionItems);
```

C++

```
class TCubeDims : public TDimensionItems;
```

File

Mxstore

Description

Use TCubeDims as the dimension map of a crosstabulated data store. Each TCubeDim (see page 17754) object maintained by TCubeDims corresponds to a field from the data set of the crosstabulated data store. Each TCubeDim (see page 17754) object can be used as either a dimension (a field over which data is summarized), or a summary (a value summarized over the dimensions).

TCubeDims introduces new properties to handle dimension maps that include date ranges.

See Also

TCubeDim (see page 17754)

TDecisionCube (see page 17442)

111.1.5.1 TCubeDims Members

The following tables list the members exposed by TCubeDims.

Protected Methods

	Name	Description
	Add (see page 17769)	Adds a dimension item to the end of the Items array.
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetDirtyFlag (see page 17770)	This is GetDirtyFlag, a member of class TCubeDims.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 17770)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	NotifyOwner (see page 17420)	This is NotifyOwner, a member of class TDimensionItems.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	AddDerivedField (see page 17418)	Adds a derived average summary to the end of the Items (see page 17422) array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	AlreadyExists (see page 17419)	Indicates whether a dimension item with specified properties already exists in the Items (see page 17422) property array.
V	Assign (see page 17770)	Copies the list of dimension items from another TDimensionItems object.
	AverageFieldCheck (see page 17419)	Determines whether the necessary summary dimension items over a base dimension exist so that a dimension item of type dimAverage can be computed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 17769)	This is Create, a member of class TCubeDims.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ActiveDimensionCount (see page 17421)	Indicates the number of dimension items that represent active dimensions in the crosstabulated data store.
R	ActiveSummaryCount (see page 17421)	Indicates the number of dimension items that represent active summaries in the crosstabulated data store.
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.

R	DimensionCount (see page 17422)	Indicates the number of dimension items in the Items (see page 17422) array with DimensionType set to dimDimension.
R	IsDirty (see page 17771)	Indicates that the fields that correspond to the TCubeDim (see page 17754) objects in the Items (see page 17771) array have changed.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 17771)	Lists all dimension items maintained by the TCubeDims (see page 17766) object.
R	SummaryCount (see page 17423)	Indicates the number of dimension items in the Items (see page 17422) array with DimensionType set to a value other than dimDimension.

111.1.5.2 TCubeDims.Create Constructor

Pascal

```
constructor Create(FOwner: TPersistent; ItemClass: TCubeDimClass);
```

C++

```
__fastcall TCubeDims(TPersistent * FOwner, TCubeDimClass ItemClass);
```

Description

This is Create, a member of class TCubeDims.

111.1.5.3 TCubeDims Methods

The methods of the TCubeDims class are listed here.

Protected Methods

	Name	Description
	Add (see page 17769)	Adds a dimension item to the end of the Items array.
	GetDirtyFlag (see page 17770)	This is GetDirtyFlag, a member of class TCubeDims.
V	GetOwner (see page 17770)	Returns the owner of an object.

Public Methods

	Name	Description
V	Assign (see page 17770)	Copies the list of dimension items from another TDimensionItems object.

111.1.5.3.1 TCubeDims.Add Method

Adds a dimension item to the end of the Items array.

Pascal

```
function Add: TCubeDim;
```

C++

```
__fastcall TCubeDim Add();
```

Description

Do not call the Add method to add dimension items to a TDimensionItems object. The dimension items are added automatically to reflect the fields in the data set of the associated crosstabulated data store.

Add adds a new dimension item to the collection and returns the new dimension item.

See Also

Items (see page 17422)

DataSet (see page 17789)

111.1.5.3.2 TCubeDims.Assign Method

Copies the list of dimension items from another TDimensionItems object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Assign calls the inherited method to copy the properties of the Source parameter. The inherited method will copy properties from any object that implements an AssignTo method that can copy to a TDimensionItems object.

See Also

AssignTo (see page 2931)

111.1.5.3.3 TCubeDims.GetDirtyFlag Method

Pascal

```
function GetDirtyFlag: Boolean;
```

C++

```
__fastcall Boolean GetDirtyFlag();
```

Description

This is GetDirtyFlag, a member of class TCubeDims.

111.1.5.3.4 TCubeDims.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

111.1.5.4 TCubeDims Properties

The properties of the TCubeDims class are listed here.

Public Properties

	Name	Description
	IsDirty (see page 17771)	Indicates that the fields that correspond to the TCubeDim (see page 17754) objects in the Items (see page 17771) array have changed.
	Items (see page 17771)	Lists all dimension items maintained by the TCubeDims (see page 17766) object.

111.1.5.4.1 TCubeDims.IsDirty Property

Indicates that the fields that correspond to the TCubeDim (see page 17754) objects in the Items (see page 17771) array have changed.

Pascal

```
property IsDirty: Boolean;
```

C++

```
__property Boolean IsDirty;
```

Description

Read IsDirty to determine whether the information cached by the decision cube is valid. When any of the properties of the TCubeDim (see page 17754) objects in the Items (see page 17771) array change, IsDirty becomes true. When the value of IsDirty is read, IsDirty is reset to false.

See Also

Items (see page 17771)

TCubeDim (see page 17754)

111.1.5.4.2 TCubeDims.Items Property

Lists all dimension items maintained by the TCubeDims (see page 17766) object.

Pascal

```
property Items [Index: Integer]: TCubeDim;
```

C++

```
__property TCubeDim Items[int Index];
```

Description

Use Items to access individual dimensions or summaries maintained by the TCubeDims (see page 17766) object. The first TCubeDim (see page 17754) object has an Index of 0, the second has an Index of 1, and so on.

Each TCubeDim (see page 17754) object in the Items array represents a field from the data set of a decision cube. Each item represents either a dimension (a field over which data is summarized), or a summary (a value summarized over the dimensions).

In Delphi code, Items is the default property of TCubeDims (see page 17766). This means that the name of the Items property can be eliminated from code that uses the Items property. Thus, the line

```
DecisionCube1.DimensionMap.Items[0].ActiveFlag := diAsNeeded;
```

could be written

```
DecisionCube1.DimensionMap[0].ActiveFlag := diAsNeeded;
```

Note: Do not change the Items property. The set of items in the Items array is set automatically to represent the fields from the data set. The properties of individual TCubeDim (see page 17754) objects within the Items array may be changed to specify whether they should be cached by the data store, the name to use when displaying them, and so on.

See Also

DataSet (see page 17789)

DimensionMap (see page 17790)

operator_[]

111.1.6 TCustomDataStore Class

TCustomDataStore is the base class for components that analyze and maintain crosstabulated data.

Class Hierarchy

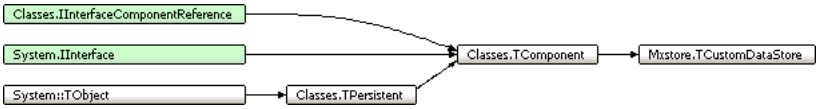

Pascal

```
TCustomDataStore = class(TComponent);
```

C++

```
class TCustomDataStore : public TComponent;
```

File

Mxstore

Description

Use TCustomDataStore as a base class when defining classes that fetch data from appropriately structured datasets and analyze it into a multidimensional array of summary values. Each dimension corresponds to a field in the dataset over which information is summarized. TCustomDataStore maintains a cached projection of this array over a set of active dimensions. The cache allows TCustomDataStore to quickly provide a projection of any subset of the active dimensions.

TCustomDataStore introduces properties and methods that describe the dimensions of the crosstabulated data array and the summary values that are computed over those dimensions.

Do not create instances of TCustomDataStore. To make the crosstabulated data from a TCustomDataStore object available to the objects in a form, use a TDecisionCube object.

See Also

TDecisionCube (see page 17442)

111.1.6.1 TCustomDataStore Members

The following tables list the members exposed by TCustomDataStore.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActiveChanged (see page 17778)	This is ActiveChanged, a member of class TCustomDataStore.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BinMapHasBinData (see page 17778)	This is BinMapHasBinData, a member of class TCustomDataStore.
V	CanDimBeClosed (see page 17778)	This is CanDimBeClosed, a member of class TCustomDataStore.
V	CanSumBeClosed (see page 17779)	This is CanSumBeClosed, a member of class TCustomDataStore.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterClose (see page 17779)	This is DoAfterClose, a member of class TCustomDataStore.
V	DoAfterOpen (see page 17779)	This is DoAfterOpen, a member of class TCustomDataStore.
V	DoBeforeClose (see page 17779)	This is DoBeforeClose, a member of class TCustomDataStore.
V	DoBeforeOpen (see page 17780)	This is DoBeforeOpen, a member of class TCustomDataStore.
V	DoOnRefresh (see page 17780)	This is DoOnRefresh, a member of class TCustomDataStore.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetCurrentSummary (see page 17780)	This is GetCurrentSummary, a member of class TCustomDataStore.
	GetDataSet (see page 17780)	This is GetDataSet, a member of class TCustomDataStore.
	GetDomain (see page 17781)	This is GetDomain, a member of class TCustomDataStore.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	LayoutChanged (see page 17785)	This is LayoutChanged, a member of class TCustomDataStore.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 17785)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetCurrentSummary (see page 17786)	This is SetCurrentSummary, a member of class TCustomDataStore.
	SetDataSet (see page 17786)	This is SetDataSet, a member of class TCustomDataStore.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.

V	StateChanged (see page 17786)	This is StateChanged, a member of class TCustomDataStore.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Active (see page 17787)	This is Active, a member of class TCustomDataStore.
R	DataCache (see page 17789)	This is DataCache, a member of class TCustomDataStore.
R	DataLink (see page 17789)	This is DataLink, a member of class TCustomDataStore.
	DataSource (see page 17790)	This is DataSource, a member of class TCustomDataStore.
	DesignState (see page 17790)	This is DesignState, a member of class TCustomDataStore.
R	InternalDataSource (see page 17792)	This is InternalDataSource, a member of class TCustomDataStore.
R	State (see page 17794)	This is State, a member of class TCustomDataStore.

Public Events

	Name	Description
	AfterClose (see page 17795)	Occurs immediately after the crosstabulated data store has been deactivated.
	AfterOpen (see page 17795)	Occurs immediately after the crosstabulated data store has been activated.
	BeforeClose (see page 17796)	Occurs immediately before the crosstabulated data store is deactivated.
	BeforeOpen (see page 17796)	Occurs immediately before the crosstabulated data store is activated.
	OnLowCapacity (see page 17796)	Occurs when the multidimensional cache requires more memory than is allowed by the Capacity (see page 17788) property.
	OnRefresh (see page 17797)	Occurs immediately before the dimension map is changed.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CalcSubTotals (see page 17778)	Calculates dimension subtotals.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 17776)	Creates an instance of TCustomDataStore (see page 17772).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 17777)	Destroys an instance of TCustomDataStore (see page 17772).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetDimensionMemberCount (see page 17780)	Returns the number of distinct values for a dimension of the data store.
	GetDimensionName (see page 17781)	Returns the name of an active dimension.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMemberAsString (see page 17782)	Returns the specified value from a dimension of the data store, formatted as a string.
	GetMemberAsVariant (see page 17782)	Returns the specified value from a dimension of the crosstabulated data store.
	GetMemoryUsage (see page 17783)	Returns the number of bytes of memory used to cache the multidimensional array of summaries along all active dimensions.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetSummaryAsString (see page 17783)	Returns the value of the current summary for a set of values along all the active dimensions.
	GetSummaryAsVariant (see page 17784)	Returns the value of the current summary for a set of values along all the active dimensions.
	GetSummaryName (see page 17784)	Returns the name of an active summary.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 17785)	Updates the dimension map of the crosstabulated data store.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	BinData (see page 17787)	Indicates whether all the dimensions in the multidimensional array include a cell for every value along the dimension.
	Capacity (see page 17788)	Limits the memory that may be used by the cached array of summary values.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	CurrentSummary (see page 17788)	Specifies the index of the current summary from the set of active summaries maintained by the crosstabulated data store.
	DataSet (see page 17789)	Specifies the dataset that provides the fields which make up the dimensions and summaries maintained by the crosstabulated data store.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DimensionCount (see page 17790)	Specifies the number of dimensions cached by the crosstabulated data store.
	DimensionMap (see page 17790)	Describes the dimensions and summaries maintained by the crosstabulated data store.
R	DimensionMapCount (see page 17791)	Specifies the total number of fields from the dataset that make up the dimensions and summaries of the dimension map.
	MaxCells (see page 17792)	Limits the number of cells that can be included in the cache at any one time.
	MaxDimensions (see page 17793)	Limits the number of dimensions that can be cached by the crosstabulated data store at any one time.
	MaxSummaries (see page 17793)	Limits the number of summaries that can be cached by the crosstabulated data store at one time.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ShowProgressDialog (see page 17794)	Determines whether a progress bar appears when the cache of crosstabulated values is built.
R	SummaryCount (see page 17794)	Specifies the number of summary fields cached by the crosstabulated data store.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

111.1.6.2 TCustomDataStore.Create Constructor

Creates an instance of TCustomDataStore (see page 17772).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomDataStore(TComponent * AOwner);
```

Description

Create allocates the helper objects used by the crosstabulated data store to manage the dimension map and the crosstabulated data.

See Also

Create (see page 2766)

111.1.6.3 TCustomDataStore.Destroy Destructor

Destroys an instance of TCustomDataStore (see page 17772).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomDataStore();
```

Description

Do not call the Destroy method directly. Instead call Free. Free verifies that the decision cube is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

111.1.6.4 TCustomDataStore Methods

The methods of the TCustomDataStore class are listed here.

Protected Methods

	Name	Description
	ActiveChanged (see page 17778)	This is ActiveChanged, a member of class TCustomDataStore.
	BinMapHasBinData (see page 17778)	This is BinMapHasBinData, a member of class TCustomDataStore.
	CanDimBeClosed (see page 17778)	This is CanDimBeClosed, a member of class TCustomDataStore.
	CanSumBeClosed (see page 17779)	This is CanSumBeClosed, a member of class TCustomDataStore.
	DoAfterClose (see page 17779)	This is DoAfterClose, a member of class TCustomDataStore.
	DoAfterOpen (see page 17779)	This is DoAfterOpen, a member of class TCustomDataStore.
	DoBeforeClose (see page 17779)	This is DoBeforeClose, a member of class TCustomDataStore.
	DoBeforeOpen (see page 17780)	This is DoBeforeOpen, a member of class TCustomDataStore.
	DoOnRefresh (see page 17780)	This is DoOnRefresh, a member of class TCustomDataStore.
	GetCurrentSummary (see page 17780)	This is GetCurrentSummary, a member of class TCustomDataStore.
	GetDataSet (see page 17780)	This is GetDataSet, a member of class TCustomDataStore.
	GetDomain (see page 17781)	This is GetDomain, a member of class TCustomDataStore.
	LayoutChanged (see page 17785)	This is LayoutChanged, a member of class TCustomDataStore.
	Notification (see page 17785)	Forwards notification messages to all owned components.
	SetCurrentSummary (see page 17786)	This is SetCurrentSummary, a member of class TCustomDataStore.
	SetDataSet (see page 17786)	This is SetDataSet, a member of class TCustomDataStore.
	StateChanged (see page 17786)	This is StateChanged, a member of class TCustomDataStore.

Public Methods

	Name	Description
	CalcSubTotals (see page 17778)	Calculates dimension subtotals.
	GetDimensionMemberCount (see page 17780)	Returns the number of distinct values for a dimension of the data store.
	GetDimensionName (see page 17781)	Returns the name of an active dimension.
	GetMemberAsString (see page 17782)	Returns the specified value from a dimension of the data store, formatted as a string.
	GetMemberAsVariant (see page 17782)	Returns the specified value from a dimension of the crosstabulated data store.
	GetMemoryUsage (see page 17783)	Returns the number of bytes of memory used to cache the multidimensional array of summaries along all active dimensions.
	GetSummaryAsString (see page 17783)	Returns the value of the current summary for a set of values along all the active dimensions.

	GetSummaryAsVariant (see page 17784)	Returns the value of the current summary for a set of values along all the active dimensions.
	GetSummaryName (see page 17784)	Returns the name of an active summary.
	Refresh (see page 17785)	Updates the dimension map of the crosstabulated data store.

111.1.6.4.1 TCustomDataStore.ActiveChanged Method

Pascal

```
procedure ActiveChanged; virtual;
```

C++

```
virtual __fastcall ActiveChanged();
```

Description

This is ActiveChanged, a member of class TCustomDataStore.

111.1.6.4.2 TCustomDataStore.BinMapHasBinData Method

Pascal

```
function BinMapHasBinData: Boolean;
```

C++

```
__fastcall Boolean BinMapHasBinData();
```

Description

This is BinMapHasBinData, a member of class TCustomDataStore.

111.1.6.4.3 TCustomDataStore.CalcSubTotals Method

Calculates dimension subtotals.

Pascal

```
procedure CalcSubTotals;
```

C++

```
__fastcall CalcSubTotals();
```

Description

CalcSubTotals calculates dimension subtotals in cached data. CalcSubtotal is used internally by TCustomDataStore (see page 17772); an application should not call CalcSubTotals directly.

See Also

Create (see page 2766)

111.1.6.4.4 TCustomDataStore.CanDimBeClosed Method

Pascal

```
function CanDimBeClosed(iMapIndex: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanDimBeClosed(int iMapIndex);
```

Description

This is CanDimBeClosed, a member of class TCustomDataStore.

111.1.6.4.5 TCustomDataStore.CanSumBeClosed Method

Pascal

```
function CanSumBeClosed(iMapIndex: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanSumBeClosed(int iMapIndex);
```

Description

This is CanSumBeClosed, a member of class TCustomDataStore.

111.1.6.4.6 TCustomDataStore.DoAfterClose Method

Pascal

```
procedure DoAfterClose; virtual;
```

C++

```
virtual __fastcall DoAfterClose();
```

Description

This is DoAfterClose, a member of class TCustomDataStore.

111.1.6.4.7 TCustomDataStore.DoAfterOpen Method

Pascal

```
procedure DoAfterOpen; virtual;
```

C++

```
virtual __fastcall DoAfterOpen();
```

Description

This is DoAfterOpen, a member of class TCustomDataStore.

111.1.6.4.8 TCustomDataStore.DoBeforeClose Method

Pascal

```
procedure DoBeforeClose; virtual;
```

C++

```
virtual __fastcall DoBeforeClose();
```

Description

This is DoBeforeClose, a member of class TCustomDataStore.

111.1.6.4.9 TCustomDataStore.DoBeforeOpen Method

Pascal

```
procedure DoBeforeOpen; virtual;
```

C++

```
virtual __fastcall DoBeforeOpen();
```

Description

This is DoBeforeOpen, a member of class TCustomDataStore.

111.1.6.4.10 TCustomDataStore.DoOnRefresh Method

Pascal

```
procedure DoOnRefresh(DimMap: TCubeDims); virtual;
```

C++

```
virtual __fastcall DoOnRefresh(TCubeDims DimMap);
```

Description

This is DoOnRefresh, a member of class TCustomDataStore.

111.1.6.4.11 TCustomDataStore.GetCurrentSummary Method

Pascal

```
function GetCurrentSummary: Integer;
```

C++

```
__fastcall int GetCurrentSummary();
```

Description

This is GetCurrentSummary, a member of class TCustomDataStore.

111.1.6.4.12 TCustomDataStore.GetDataSet Method

Pascal

```
function GetDataSet: TDataSet;
```

C++

```
__fastcall TDataSet GetDataSet();
```

Description

This is GetDataSet, a member of class TCustomDataStore.

111.1.6.4.13 TCustomDataStore.GetDimensionMemberCount Method

Returns the number of distinct values for a dimension of the data store.

Pascal

```
function GetDimensionMemberCount(Dimension: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int GetDimensionMemberCount(int Dimension);
```

Description

Call GetDimensionMemberCount to determine the number of distinct field values in the dimension specified by the Dimension parameter. Zero specifies the first active dimension, and DimensionCount (see page 17790)-1 specifies the last active dimension.

To obtain the individual values of the dimension, use the GetMemberAsString (see page 17782) or GetMemberAsVariant (see page 17782) method. GetDimensionMemberCount returns a value that provides an upper limit to the Index parameter of the GetMemberAsString (see page 17782) or GetMemberAsVariant (see page 17782) method.

To obtain the name of the dimension referred to by the Dimension parameter, use the GetDimensionName (see page 17781) method.

See Also

DimensionCount (see page 17790)

GetDimensionName (see page 17781)

GetMemberAsString (see page 17782)

GetMemberAsVariant (see page 17782)

111.1.6.4.14 TCustomDataStore.GetDimensionName Method

Returns the name of an active dimension.

Pascal

```
function GetDimensionName(Dimension: Integer): String; virtual;
```

C++

```
virtual __fastcall AnsiString GetDimensionName(int Dimension);
```

Description

Call GetDimensionName to obtain the name of the dimension indexed by the Dimension parameter. Zero specifies the first active dimension and DimensionCount (see page 17790)-1 specifies the last active dimension. The name of a dimension is the name of the database field the dimension represents, unless the Name property of the corresponding TDimensionItem object is changed.

See Also

DimensionCount (see page 17790)

Name (see page 17414)

111.1.6.4.15 TCustomDataStore.GetDomain Method

Pascal

```
function GetDomain(DimensionIDs: TIntArray; ATotals: Boolean; Domain: TTwoDimArray): Integer;
```

C++

```
__fastcall int GetDomain(TIntArray DimensionIDs, Boolean ATotals, TTwoDimArray Domain);
```

Description

This is GetDomain, a member of class TCustomDataStore.

111.1.6.4.16 TCustomDataStore.GetMemberAsString Method

Returns the specified value from a dimension of the data store, formatted as a string.

Pascal

```
function GetMemberAsString(Dimension: Integer; Index: Integer): String; virtual;
```

C++

```
virtual __fastcall AnsiString GetMemberAsString(int Dimension, int Index);
```

Description

Call GetMemberAsString to retrieve one of the values from the dimension specified by the Dimension parameter. Zero specifies the first dimension, and DimensionCount (see page 17790)-1 specifies the last dimension. The Index parameter indicates which value along the dimension is returned, where 0 is the first value. GetDimensionMemberCount (see page 17780) returns the number of values included in the dimension.

GetMemberAsString returns a field value formatted as a string. To obtain the field value as an unformatted variant, use the GetMemberAsVariant (see page 17782) method.

See Also

DimensionCount (see page 17790)

GetDimensionMemberCount (see page 17780)

GetDimensionName (see page 17781)

GetMemberAsVariant (see page 17782)

111.1.6.4.17 TCustomDataStore.GetMemberAsVariant Method

Returns the specified value from a dimension of the crosstabulated data store.

Pascal

```
function GetMemberAsVariant(Dimension: Integer; Index: Integer): Variant;
```

C++

```
__fastcall Variant GetMemberAsVariant(int Dimension, int Index);
```

Description

Call GetMemberAsVariant to retrieve one of the values from the dimension specified by the Dimension parameter. Zero specifies the first dimension, and DimensionCount (see page 17790)-1 specifies the last dimension. The Index parameter indicates which value along the dimension is returned, where 0 is the first value. GetDimensionMemberCount (see page 17780) returns the number of values included in the dimension.

See Also

DimensionCount (see page 17790)

GetDimensionMemberCount (see page 17780)

GetDimensionName (see page 17781)

GetMemberAsString (see page 17782)

111.1.6.4.18 TCustomDataStore.GetMemoryUsage Method

Returns the number of bytes of memory used to cache the multidimensional array of summaries along all active dimensions.

Pascal

```
function GetMemoryUsage: Integer;
```

C++

```
__fastcall int GetMemoryUsage();
```

Description

Call GetMemoryUsage to determine the size of the cache that stores the crosstabulated information for the data store. If the returned value is too large, consider making some of the active dimensions paged by setting their BinType property to binSet or allowing the data store to swap some dimensions out by setting their ActiveFlag property to diAsNeeded. The cube dimension can be accessed using the DimensionMap (see page 17790) property.

The storage needs of the crosstabulated data store increase geometrically with each active dimension that is added. Applications must find the best balance between the performance benefit of caching many dimensions so that they are immediately available to decision controls, and the need to keep the number of active dimensions small enough so that the cache does not consume inordinate amounts of memory.

See Also

DimensionMap (see page 17790)

BinType (see page 17763)

ActiveFlag (see page 17411)

111.1.6.4.19 TCustomDataStore.GetSummaryAsString Method

Returns the value of the current summary for a set of values along all the active dimensions.

Pascal

```
function GetSummaryAsString(Coord: TSmallIntArray): String; virtual;
```

C++

```
virtual __fastcall AnsiString GetSummaryAsString(TSmallIntArray Coord);
```

Description

Call GetSummaryAsString to obtain the value of a single cell from the multidimensional array of summary values. Set the Coord parameter to an array of indexes, where each index identifies a value along one of the dimensions of the array. The first value of Coord is the index of the desired value along the first active dimension, the second value of Coord is the index of the desired value along the second active dimension, and so on. Coord should contain DimensionCount (see page 17790) values.

To obtain the names of the active dimensions that correspond to the entries in the Coord parameter, use the GetDimensionName (see page 17781) method. To obtain the name of the specified value along any of the dimensions, use the GetMemberAsString (see page 17782) method.

GetSummaryAsString formats the summarized value as a string. To obtain the summary as a variant, use the GetSummaryAsVariant (see page 17784) method.

See Also

CurrentSummary (see page 17788)

DimensionCount (see page 17790)

GetDimensionName (see page 17781)

GetMemberAsString (see page 17782)

GetSummaryAsVariant (see page 17784)

111.1.6.4.20 TCustomDataStore.GetSummaryAsVariant Method

Returns the value of the current summary for a set of values along all the active dimensions.

Pascal

```
function GetSummaryAsVariant(Coord: TSmallIntArray): Variant; virtual;
```

C++

```
virtual __fastcall Variant GetSummaryAsVariant(TSmallIntArray Coord);
```

Description

Call GetSummaryAsVariant to obtain the value of a single cell from the multidimensional array of summary values. Set the Coord parameter to an array of indexes, where each index identifies a value along one of the dimensions of the array. The first value of Coord is the index of the desired value along the first active dimension, the second value of Coord is the index of the desired value along the second active dimension, and so on. Coord should contain DimensionCount (see page 17790) values.

To obtain the names of the active dimensions that correspond to the entries in the Coord parameter, use the GetDimensionName (see page 17781) method. To obtain the name of the specified value along any of the dimensions, use the GetMemberAsString (see page 17782) method.

To obtain the summary formatted as a string, use the GetSummaryAsString (see page 17783) method.

See Also

CurrentSummary (see page 17788)

DimensionCount (see page 17790)

GetDimensionName (see page 17781)

GetMemberAsString (see page 17782)

GetSummaryAsString (see page 17783)

111.1.6.4.21 TCustomDataStore.GetSummaryName Method

Returns the name of an active summary.

Pascal

```
function GetSummaryName(ISum: Integer): String; virtual;
```

C++

```
virtual __fastcall AnsiString GetSummaryName(int ISum);
```

Description

Call GetSummaryName to obtain the name of the summary specified by the ISum parameter. Zero specifies the first active summary and SummaryCount (see page 17794)-1 specifies the last active summary.

See Also

SummaryCount (see page 17794)

111.1.6.4.22 TCustomDataStore.LayoutChanged Method

Pascal

```
procedure LayoutChanged; virtual;
```

C++

```
virtual __fastcall LayoutChanged();
```

Description

This is LayoutChanged, a member of class TCustomDataStore.

111.1.6.4.23 TCustomDataStore.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

111.1.6.4.24 TCustomDataStore.Refresh Method

Updates the dimension map of the crosstabulated data store.

Pascal

```
procedure Refresh(DimMap: TCubeDims; bForce: Boolean);
```

C++

```
__fastcall Refresh(TCubeDims DimMap, Boolean bForce);
```

Description

To change the properties of the dimensions in the dimension map, make a copy of the DimensionMap (see page 17790) property, make any desired changes to the properties of the TCubeDim (see page 17754) objects, and call Refresh, passing in the altered dimension map. The bForce parameter specifies whether the cache should be rebuilt unconditionally. When bForce is

false, the cache will not be rebuilt unless the changes require it.

When the data store is active, call Refresh rather than directly altering the dimension map. Do not call Refresh while the data store is building the cache. Use the BeforeOpen (see page 17796) and AfterOpen (see page 17795) events to determine when the cache is being rebuilt.

See Also

AfterOpen (see page 17795)

BeforeOpen (see page 17796)

DimensionMap (see page 17790)

TCubeDim (see page 17754)

111.1.6.4.25 TCustomDataStore.SetCurrentSummary Method

Pascal

```
procedure SetCurrentSummary(Value: Integer);
```

C++

```
__fastcall SetCurrentSummary(int Value);
```

Description

This is SetCurrentSummary, a member of class TCustomDataStore.

111.1.6.4.26 TCustomDataStore.SetDataSet Method

Pascal

```
procedure SetDataSet(ADataSet: TDataSet);
```

C++

```
__fastcall SetDataSet(TDataSet ADataSet);
```

Description

This is SetDataSet, a member of class TCustomDataStore.

111.1.6.4.27 TCustomDataStore.StateChanged Method

Pascal

```
procedure StateChanged; virtual;
```

C++

```
virtual __fastcall StateChanged();
```

Description

This is StateChanged, a member of class TCustomDataStore.

111.1.6.5 TCustomDataStore Properties

The properties of the TCustomDataStore class are listed here.

Protected Properties

	Name	Description
	Active (see page 17787)	This is Active, a member of class TCustomDataStore.
R	DataCache (see page 17789)	This is DataCache, a member of class TCustomDataStore.
R	DataLink (see page 17789)	This is DataLink, a member of class TCustomDataStore.
	DataSource (see page 17790)	This is DataSource, a member of class TCustomDataStore.
	DesignState (see page 17790)	This is DesignState, a member of class TCustomDataStore.
R	InternalDataSource (see page 17792)	This is InternalDataSource, a member of class TCustomDataStore.
R	State (see page 17794)	This is State, a member of class TCustomDataStore.

Public Properties

	Name	Description
	BinData (see page 17787)	Indicates whether all the dimensions in the multidimensional array include a cell for every value along the dimension.
	Capacity (see page 17788)	Limits the memory that may be used by the cached array of summary values.
	CurrentSummary (see page 17788)	Specifies the index of the current summary from the set of active summaries maintained by the crosstabulated data store.
	DataSet (see page 17789)	Specifies the dataset that provides the fields which make up the dimensions and summaries maintained by the crosstabulated data store.
R	DimensionCount (see page 17790)	Specifies the number of dimensions cached by the crosstabulated data store.
	DimensionMap (see page 17790)	Describes the dimensions and summaries maintained by the crosstabulated data store.
R	DimensionMapCount (see page 17791)	Specifies the total number of fields from the dataset that make up the dimensions and summaries of the dimension map.
	MaxCells (see page 17792)	Limits the number of cells that can be included in the cache at any one time.
	MaxDimensions (see page 17793)	Limits the number of dimensions that can be cached by the crosstabulated data store at any one time.
	MaxSummaries (see page 17793)	Limits the number of summaries that can be cached by the crosstabulated data store at one time.
	ShowProgressDialog (see page 17794)	Determines whether a progress bar appears when the cache of crosstabulated values is built.
R	SummaryCount (see page 17794)	Specifies the number of summary fields cached by the crosstabulated data store.

111.1.6.5.1 TCustomDataStore.Active Property

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

This is Active, a member of class TCustomDataStore.

111.1.6.5.2 TCustomDataStore.BinData Property

Indicates whether all the dimensions in the multidimensional array include a cell for every value along the dimension.

Pascal

```
property BinData: Boolean;
```

C++

```
__property Boolean BinData;
```

Description

Read BinData to determine whether any of the active dimensions have been grouped into ranges or been put into a paged

(permanently drilled) state. When BinData is true, at least one dimension has been paged or has been collapsed into ranges of values. When BinData is false, all dimensions include entries for every value of the dimension field that appears in the dataset.

See Also

IsBinData (see page 17761)

111.1.6.5.3 TCustomDataStore.Capacity Property

Limits the memory that may be used by the cached array of summary values.

Pascal

```
property Capacity: Integer;
```

C++

```
__property int Capacity;
```

Description

Set Capacity to limit the number of bytes that may be used by the crosstabulated data store for storing the multidimensional array of summary values. If the number of active dimensions and summaries causes the cache to exceed this value, an ELowCapacityError is raised.

See Also

ELowCapacityError (see page 17306)

OnLowCapacity (see page 17796)

ActiveFlag (see page 17411)

111.1.6.5.4 TCustomDataStore.CurrentSummary Property

Specifies the index of the current summary from the set of active summaries maintained by the crosstabulated data store.

Pascal

```
property CurrentSummary: Integer;
```

C++

```
__property int CurrentSummary;
```

Description

Use CurrentSummary to determine which of the active summaries is used to provide values to clients of the data store. CurrentSummary returns 0 if the first active summary is the current summary. CurrentSummary returns SummaryCount (see page 17794)-1 if the last active summary is the current summary.

The dataset specified by the DataSet (see page 17789) property provides all the fields that make up the dimensions and summaries of the dimension map. At any time, a subset of all the dimensions and summaries provided by the dataset is cached by the crosstabulated data store and available to decision source objects. Of all the active (cached) summaries, only one can be provided to decision source objects at a time. That one summary is the current summary.

See Also

DataSet (see page 17789)

DimensionMap (see page 17790)

GetSummaryName (see page 17784)

SummaryCount (see page 17794)

111.1.6.5.5 TCustomDataStore.DataCache Property

Pascal

```
property DataCache: TDataCache;
```

C++

```
__property TDataCache DataCache;
```

Description

This is DataCache, a member of class TCustomDataStore.

111.1.6.5.6 TCustomDataStore.DataLink Property

Pascal

```
property DataLink: TMultiDimDataLink;
```

C++

```
__property TMultiDimDataLink DataLink;
```

Description

This is DataLink, a member of class TCustomDataStore.

111.1.6.5.7 TCustomDataStore.DataSet Property

Specifies the dataset that provides the fields which make up the dimensions and summaries maintained by the crosstabulated data store.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Set DataSet to the dataset that contains the crosstabulated data for the data store. Usually, DataSet is a TDecisionQuery object. TDecisionQuery objects are SQL queries that are grouped appropriately to calculate summaries for a crosstabulated data store.

The DataSet property can also be set to a TQuery object where the SQL has the same form as that of a TDecisionQuery or to any other dataset that contains summarized values in the same structure as the result of a TDecisionQuery query.

If the DataSet property is a TQuery or TDecisionQuery object, the crosstabulated data store automatically determines which fields are dimensions and which are summaries. If the DataSet property is not a query, the fields in the dimension map must be explicitly identified as dimensions or summaries, and, if they represent summaries, the type of summary must be indicated.

Note: The summary values for the crosstabulated data store must be calculated by the TDataSet object. The data store does not summarize raw data, but analyzes summarized data into dimensions.

See Also

DimensionMap (see page 17790)

TDecisionQuery (see page 17854)

TQuery (see page 10059)

111.1.6.5.8 TCustomDataStore.DataSource Property

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

This is DataSource, a member of class TCustomDataStore.

111.1.6.5.9 TCustomDataStore.DesignState Property

Pascal

```
property DesignState: TCubeDataState;
```

C++

```
__property TCubeDataState DesignState;
```

Description

This is DesignState, a member of class TCustomDataStore.

111.1.6.5.10 TCustomDataStore.DimensionCount Property

Specifies the number of dimensions cached by the crosstabulated data store.

Pascal

```
property DimensionCount: Integer;
```

C++

```
__property int DimensionCount;
```

Description

Read DimensionCount to determine the number of active dimensions. The dataset specified by the DataSet (see page 17789) property provides all the fields that make up the dimensions and summaries of the dimension map. At any time, a subset of all the dimensions and summaries provided by the dataset is cached by the crosstabulated data store and available to decision source objects. DimensionCount is the number of active (cached) dimensions that can be provided to decision source objects.

See Also

DataSet (see page 17789)

DimensionMap (see page 17790)

TDecisionSource (see page 17462)

111.1.6.5.11 TCustomDataStore.DimensionMap Property

Describes the dimensions and summaries maintained by the crosstabulated data store.

Pascal

```
property DimensionMap: TCubeDims;
```

C++

```
__property TCubeDims DimensionMap;
```

Description

Read DimensionMap to access the dimensions and summaries of the data store. The individual entries of the dimension map represent the fields of the dataset specified by the DataSet (see page 17789) property or calculated fields derived from those fields, such as averages. DimensionMap provides information about these fields from the dataset that make up the crosstabulated data store, including the field names and the use of the fields within the multidimensional array.

When the data store is not active, use the properties of the TCubeDim (see page 17754) objects to change the information cached by the data store. Use the ActiveFlag property of individual TCubeDim (see page 17754) objects to specify whether those dimensions or summaries from the dataset are cached by the crosstabulated data store. Use the BinType property of the TCubeDim (see page 17754) objects to allow cells along temporal dimensions to represent date ranges or to put the dimensions in a paged (permanently drilled) state.

If the dataset is not a query, use the DimensionType property of the TCubeDim (see page 17754) objects to identify which fields are dimensions and which are summaries, and to identify the type of summary represented by the summary fields.

To change the properties of the entries in the dimension map when the data store is active, use the Refresh (see page 17785) method.

Note: At design time, the DimensionType, Name, and ActiveFlag properties of the TCubeDim (see page 17754) objects that make up the dimension map can be set using the Decision Cube Editor.

See Also

DataSet (see page 17789)

Refresh (see page 17785)

BinType (see page 17763)

ActiveFlag (see page 17411)

DimensionType (see page 17412)

Name (see page 17414)

Decision Cube Properties and Events

Using the Decision Cube Editor

111.1.6.5.12 TCustomDataStore.DimensionMapCount Property

Specifies the total number of fields from the dataset that make up the dimensions and summaries of the dimension map.

Pascal

```
property DimensionMapCount: Integer;
```

C++

```
__property int DimensionMapCount;
```

Description

Read DimensionMapCount to determine the number of TCubeDim (see page 17754) objects that can be accessed through the DimensionMap (see page 17790) property. The value of DimensionMapCount is the number of fields in the dataset specified by the DataSet (see page 17789) property plus the number of derived fields based on those fields, such as fields for average values that are derived from count and summary fields.

The value of DimensionMapCount is not the same as DimensionCount (see page 17790) + SummaryCount (see page

17794). DimensionCount (see page 17790) and SummaryCount (see page 17794) count the number of dimensions and summaries that are cached by the crosstabulated data cube and available to the decision source. DimensionMapCount also counts any inactive dimensions or summaries that are not included in the cache.

See Also

DataSet (see page 17789)

DimensionCount (see page 17790)

DimensionMap (see page 17790)

SummaryCount (see page 17794)

TCubeDim (see page 17754)

111.1.6.5.13 TCustomDataStore.InternalDataSource Property

Pascal

```
property InternalDataSource: TDataSource;
```

C++

```
__property TDataSource InternalDataSource;
```

Description

This is InternalDataSource, a member of class TCustomDataStore.

111.1.6.5.14 TCustomDataStore.MaxCells Property

Limits the number of cells that can be included in the cache at any one time.

Pascal

```
property MaxCells: Integer;
```

C++

```
__property int MaxCells;
```

Description

Use MaxCells to limit the memory usage of the crosstabulated data store. Memory consumption increases geometrically with each dimension that is cached in the multidimensional array and linearly with each summary. MaxCells prevents too many dimensions or summaries from being cached at once.

When the ActiveFlag property of a dimension or summary is diActive, it is always added to the cache. When the ActiveFlag property of a dimension or summary is diAsNeeded, it can be added only if it does not cause the number of cells to exceed MaxCells. The data store swaps in as-needed dimensions and summaries when they are used by the decision source, making room, if necessary, by swapping out closed dimensions or summaries that are not current.

Note: Be sure to set MaxCells to a value big enough to accommodate all active dimensions and summaries plus the number of as-needed dimensions and summaries that can be open at one time.

See Also

MaxDimensions (see page 17793)

MaxSummaries (see page 17793)

ActiveFlag (see page 17411)

111.1.6.5.15 TCustomDataStore.MaxDimensions Property

Limits the number of dimensions that can be cached by the crosstabulated data store at any one time.

Pascal

```
property MaxDimensions: Integer;
```

C++

```
__property int MaxDimensions;
```

Description

Use MaxDimensions to limit the memory usage of the data store. Memory consumption increases geometrically with each dimension that is cached in the multidimensional array. MaxDimensions prevents too many dimensions to be cached at once.

When the ActiveFlag property of a dimension is diActive, the dimension is always added to the cache. When the ActiveFlag property of a dimension is diAsNeeded, it can be added only if it does not cause the number of active dimensions to exceed MaxDimensions. The data store swaps in as-needed dimensions when they are open in the decision source, making room, if necessary, by swapping out closed dimensions.

Note: Be sure to set MaxDimensions to a value big enough to accommodate all active dimensions plus the number of as-needed dimensions that can be open at one time.

See Also

DimensionCount (see page 17790)

MaxCells (see page 17792)

MaxSummaries (see page 17793)

ActiveFlag (see page 17411)

111.1.6.5.16 TCustomDataStore.MaxSummaries Property

Limits the number of summaries that can be cached by the crosstabulated data store at one time.

Pascal

```
property MaxSummaries: Integer;
```

C++

```
__property int MaxSummaries;
```

Description

Use MaxSummaries to limit the memory usage of the data store. Memory consumption increases linearly with each summary that is cached in the multidimensional array. MaxSummaries prevents too many summaries to be cached at once.

When the ActiveFlag property of a summary is diActive, the summary is always added to the cache. When the ActiveFlag property is diAsNeeded, it can be added only if it does not cause the number of active summaries to exceed MaxSummaries. The data store swaps in as-needed summaries when they are chosen as the current summary, making room, if necessary, by swapping out other as-needed summaries.

See Also

MaxCells (see page 17792)

MaxDimensions (see page 17793)

SummaryCount (see page 17794)

ActiveFlag (see page 17411)

111.1.6.5.17 TCustomDataStore.ShowProgressDialog Property

Determines whether a progress bar appears when the cache of crosstabulated values is built.

Pascal

```
property ShowProgressDialog: Boolean;
```

C++

```
__property Boolean ShowProgressDialog;
```

Description

Set ShowProgressDialog to specify whether a progress bar should appear to provide users with a visual indication when the cache is being built. If the crosstabulated data store includes may dimensions, rebuilding the cache can take a noticeable amount of time. The progress bar provides users with feedback about how long this process will take.

See Also

DimensionCount (see page 17790)

TProgressBar (see page 4615)

111.1.6.5.18 TCustomDataStore.State Property

Pascal

```
property State: TCubeState;
```

C++

```
__property TCubeState State;
```

Description

This is State, a member of class TCustomDataStore.

111.1.6.5.19 TCustomDataStore.SummaryCount Property

Specifies the number of summary fields cached by the crosstabulated data store.

Pascal

```
property SummaryCount: Integer;
```

C++

```
__property int SummaryCount;
```

Description

Read SummaryCount to determine the number of active summaries in the dimension map. The dataset specified by the DataSet (see page 17789) property provides all the fields that make up the dimensions and summaries of the dimension map. At any time, a subset of all the dimensions and summaries provided by the dataset is cached by the crosstabulated data store and available to decision source objects. SummaryCount is the number of active (cached) summaries that can provide data to decision source objects.

See Also

DataSet (see page 17789)

DimensionMap (see page 17790)

TDecisionSource (see page 17462)

111.1.6.6 TCustomDataStore Events

The events of the TCustomDataStore class are listed here.

Public Events

	Name	Description
	AfterClose (see page 17795)	Occurs immediately after the crosstabulated data store has been deactivated.
	AfterOpen (see page 17795)	Occurs immediately after the crosstabulated data store has been activated.
	BeforeClose (see page 17796)	Occurs immediately before the crosstabulated data store is deactivated.
	BeforeOpen (see page 17796)	Occurs immediately before the crosstabulated data store is activated.
	OnLowCapacity (see page 17796)	Occurs when the multidimensional cache requires more memory than is allowed by the Capacity (see page 17788) property.
	OnRefresh (see page 17797)	Occurs immediately before the dimension map is changed.

111.1.6.6.1 TCustomDataStore.AfterClose Event

Occurs immediately after the crosstabulated data store has been deactivated.

Pascal

```
property AfterClose: TCubeNotifyEvent;
```

C++

```
__property TCubeNotifyEvent AfterClose;
```

Description

Write an AfterClose event handler to take specific action when the crosstabulated data store has been shut down. For example, an application could provide feedback about the state of the data store, or update the dimension map for the next time the data store is activated.

See Also

BeforeClose (see page 17796)

DimensionMap (see page 17790)

111.1.6.6.2 TCustomDataStore.AfterOpen Event

Occurs immediately after the crosstabulated data store has been activated.

Pascal

```
property AfterOpen: TCubeNotifyEvent;
```

C++

```
__property TCubeNotifyEvent AfterOpen;
```

Description

Write an AfterOpen event handler to take specific action when the crosstabulated data store is activated. This is the first opportunity to determine how much memory is consumed by the multidimensional cache maintained by the data store.

See Also

BeforeOpen (see page 17796)

GetMemoryUsage (see page 17783)

111.1.6.6.3 TCustomDataStore.BeforeClose Event

Occurs immediately before the crosstabulated data store is deactivated.

Pascal

```
property BeforeClose: TCubeNotifyEvent;
```

C++

```
__property TCubeNotifyEvent BeforeClose;
```

Description

Write a BeforeClose event handler to take specific action before the data store shuts down. For example, an application can store information about the current state of the cache before it is cleared and all associated memory freed.

See Also

AfterClose (see page 17795)

111.1.6.6.4 TCustomDataStore.BeforeOpen Event

Occurs immediately before the crosstabulated data store is activated.

Pascal

```
property BeforeOpen: TCubeNotifyEvent;
```

C++

```
__property TCubeNotifyEvent BeforeOpen;
```

Description

Write a BeforeOpen event handler to take specific action before the crosstabulated data store is activated. This is an appropriate time to adjust the value of Capacity (see page 17788), or set the ShowProgressDialog (see page 17794) based on the number of active dimensions in the DimensionMap (see page 17790).

This is the last opportunity to adjust the entries in the DimensionMap (see page 17790) before the cache is built. If the data store has never been active before this, the DimensionMap (see page 17790) property will be nil (Delphi) or NULL (C++) at this point. A BeforeOpen event handler can build a dimension map programmatically if it is nil (Delphi) or NULL (C++). If the dimension map is not nil (Delphi) or NULL (C++), the properties of the dimension items can be changed, but the order and number of the dimension items themselves should not be changed.

See Also

AfterOpen (see page 17795)

Capacity (see page 17788)

DimensionMap (see page 17790)

ShowProgressDialog (see page 17794)

111.1.6.6.5 TCustomDataStore.OnLowCapacity Event

Occurs when the multidimensional cache requires more memory than is allowed by the Capacity (see page 17788) property.

Pascal

```
property OnLowCapacity: TCapacityErrorEvent;
```

C++

```
__property TCapacityErrorEvent OnLowCapacity;
```

Description

Write an OnLowCapacity event handler to respond to ELowCapacityError exceptions. An OnLowCapacity event handler can change the value of Capacity (see page 17788) or deactivate some of the dimensions in the dimension map in order to allow the cache to fit in the available memory. If the memory problem is handled by the OnLowCapacity event handler, change the Action parameter to eaContinue to allow the data store to continue building the cache. Otherwise, an ELowCapacityError exception will be raised.

See Also

Capacity (see page 17788)

DimensionMap (see page 17790)

ELowCapacityError (see page 17306)

111.1.6.6.6 TCustomDataStore.OnRefresh Event

Occurs immediately before the dimension map is changed.

Pascal

```
property OnRefresh: TCubeRefreshEvent;
```

C++

```
__property TCubeRefreshEvent OnRefresh;
```

Description

Write an OnRefresh event handler to take specific action when the dimension map changes. The DimMap parameter specifies the new value that is about to be set for the DimensionMap (see page 17790) property. Use an OnRefresh event handler to change the properties of these dimension items before the DimensionMap (see page 17790) property is set and the cache is rebuilt. This is also an opportunity to reset the MaxDimensions (see page 17793) or MaxSummaries (see page 17793) property to accommodate the dimension map that is about to be set.

See Also

DimensionMap (see page 17790)

MaxDimensions (see page 17793)

MaxSummaries (see page 17793)

Refresh (see page 17785)

Decision Cube Properties and Events

111.1.7 TDataCache Class

Class Hierarchy

Pascal

```
TDataCache = class;
```

C++

```
class TDataCache;
```

File

Mxstore

Description

This is class Mxstore.TDataCache.

111.1.7.1 TDataCache Members

The following tables list the members exposed by TDataCache.

Protected Methods

	Name	Description
	FreeCache (see page 17802)	This is FreeCache, a member of class TDataCache.

Public Methods

	Name	Description
	AddAggIndex (see page 17800)	This is AddAggIndex, a member of class TDataCache.
	AddIndex (see page 17800)	This is AddIndex, a member of class TDataCache.
	AppendDimension (see page 17800)	This is AppendDimension, a member of class TDataCache.
	AppendSummary (see page 17801)	This is AppendSummary, a member of class TDataCache.
	CalcSubTotals (see page 17801)	This is CalcSubTotals, a member of class TDataCache.
	ClearIndexInfo (see page 17801)	This is ClearIndexInfo, a member of class TDataCache.
	Create (see page 17799)	This is Create, a member of class TDataCache.
	CreateTable (see page 17801)	This is CreateTable, a member of class TDataCache.
V	Destroy (see page 17799)	This is Destroy, a member of class TDataCache.
	DimensionFromFieldName (see page 17801)	This is DimensionFromFieldName, a member of class TDataCache.
	GetDimensionMember (see page 17802)	This is GetDimensionMember, a member of class TDataCache.
	GetDimensionMemberAsVariant (see page 17802)	This is GetDimensionMemberAsVariant, a member of class TDataCache.
	GetDimensionMemberCount (see page 17802)	This is GetDimensionMemberCount, a member of class TDataCache.
	GetDimensionName (see page 17803)	This is GetDimensionName, a member of class TDataCache.
	GetDomain (see page 17803)	This is GetDomain, a member of class TDataCache.
	GetIndexCount (see page 17803)	This is GetIndexCount, a member of class TDataCache.
	GetMemoryUsage (see page 17803)	This is GetMemoryUsage, a member of class TDataCache.
	GetSummaryAsString (see page 17803)	This is GetSummaryAsString, a member of class TDataCache.
	GetSummaryAsVariant (see page 17804)	This is GetSummaryAsVariant, a member of class TDataCache.
	GetSummaryName (see page 17804)	This is GetSummaryName, a member of class TDataCache.
	HasSubTotals (see page 17804)	This is HasSubTotals, a member of class TDataCache.
	HasValidSubTotals (see page 17804)	This is HasValidSubTotals, a member of class TDataCache.
	IncSummaryIndex (see page 17805)	This is IncSummaryIndex, a member of class TDataCache.
	IsDimension (see page 17805)	This is IsDimension, a member of class TDataCache.
	IsSummary (see page 17805)	This is IsSummary, a member of class TDataCache.
	SummaryFromCubeDimIndex (see page 17805)	This is SummaryFromCubeDimIndex, a member of class TDataCache.
	SummaryFromFieldName (see page 17805)	This is SummaryFromFieldName, a member of class TDataCache.
	SummaryFromPosition (see page 17806)	This is SummaryFromPosition, a member of class TDataCache.

Public Properties

	Name	Description
	Active (see page 17806)	This is Active, a member of class TDataCache.
	CurrentSummary (see page 17807)	This is CurrentSummary, a member of class TDataCache.
R	DimensionCount (see page 17807)	This is DimensionCount, a member of class TDataCache.

	Dimensions (see page 17807)	This is Dimensions, a member of class TDataCache.
	ErrorCode (see page 17807)	This is ErrorCode, a member of class TDataCache.
	HasIndex (see page 17807)	This is HasIndex, a member of class TDataCache.
R	IndexCount (see page 17808)	This is IndexCount, a member of class TDataCache.
	PreCalculateTotals (see page 17808)	This is PreCalculateTotals, a member of class TDataCache.
	Sparsing (see page 17808)	This is Sparsing, a member of class TDataCache.
	Success (see page 17808)	This is Success, a member of class TDataCache.
R	Summaries (see page 17809)	This is Summaries, a member of class TDataCache.
R	SummaryCount (see page 17809)	This is SummaryCount, a member of class TDataCache.

111.1.7.2 TDataCache.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TDataCache();
```

Description

This is Create, a member of class TDataCache.

111.1.7.3 TDataCache.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataCache();
```

Description

This is Destroy, a member of class TDataCache.

111.1.7.4 TDataCache Methods

The methods of the TDataCache class are listed here.

Protected Methods

	Name	Description
	FreeCache (see page 17802)	This is FreeCache, a member of class TDataCache.

Public Methods

	Name	Description
	AddAggIndex (see page 17800)	This is AddAggIndex, a member of class TDataCache.
	AddIndex (see page 17800)	This is AddIndex, a member of class TDataCache.
	AppendDimension (see page 17800)	This is AppendDimension, a member of class TDataCache.
	AppendSummary (see page 17801)	This is AppendSummary, a member of class TDataCache.
	CalcSubTotals (see page 17801)	This is CalcSubTotals, a member of class TDataCache.
	ClearIndexInfo (see page 17801)	This is ClearIndexInfo, a member of class TDataCache.
	CreateTable (see page 17801)	This is CreateTable, a member of class TDataCache.
	DimensionFromFieldName (see page 17801)	This is DimensionFromFieldName, a member of class TDataCache.
	GetDimensionMember (see page 17802)	This is GetDimensionMember, a member of class TDataCache.

	GetDimensionMemberAsVariant (see page 17802)	This is GetDimensionMemberAsVariant, a member of class TDataCache.
	GetDimensionMemberCount (see page 17802)	This is GetDimensionMemberCount, a member of class TDataCache.
	GetDimensionName (see page 17803)	This is GetDimensionName, a member of class TDataCache.
	GetDomain (see page 17803)	This is GetDomain, a member of class TDataCache.
	GetIndexCount (see page 17803)	This is GetIndexCount, a member of class TDataCache.
	GetMemoryUsage (see page 17803)	This is GetMemoryUsage, a member of class TDataCache.
	GetSummaryAsString (see page 17803)	This is GetSummaryAsString, a member of class TDataCache.
	GetSummaryAsVariant (see page 17804)	This is GetSummaryAsVariant, a member of class TDataCache.
	GetSummaryName (see page 17804)	This is GetSummaryName, a member of class TDataCache.
	HasSubTotals (see page 17804)	This is HasSubTotals, a member of class TDataCache.
	HasValidSubTotals (see page 17804)	This is HasValidSubTotals, a member of class TDataCache.
	IncSummaryIndex (see page 17805)	This is IncSummaryIndex, a member of class TDataCache.
	IsDimension (see page 17805)	This is IsDimension, a member of class TDataCache.
	IsSummary (see page 17805)	This is IsSummary, a member of class TDataCache.
	SummaryFromCubeDimIndex (see page 17805)	This is SummaryFromCubeDimIndex, a member of class TDataCache.
	SummaryFromFieldName (see page 17805)	This is SummaryFromFieldName, a member of class TDataCache.
	SummaryFromPosition (see page 17806)	This is SummaryFromPosition, a member of class TDataCache.

111.1.7.4.1 TDataCache.AddAggIndex Method

Pascal

```
function AddAggIndex(SumIndex: TSmallIntArray; BuilderDims: TList): Integer;
```

C++

```
__fastcall int AddAggIndex(TSmallIntArray SumIndex, TList BuilderDims);
```

Description

This is AddAggIndex, a member of class TDataCache.

111.1.7.4.2 TDataCache.AddIndex Method

Pascal

```
function AddIndex(SumIndex: TSmallIntArray; bSparse: Boolean): Integer;
```

C++

```
__fastcall int AddIndex(TSmallIntArray SumIndex, Boolean bSparse);
```

Description

This is AddIndex, a member of class TDataCache.

111.1.7.4.3 TDataCache.AppendDimension Method

Pascal

```
function AppendDimension(Value: TDimension): Integer;
```

C++

```
__fastcall int AppendDimension(TDimension Value);
```

Description

This is AppendDimension, a member of class TDataCache.

111.1.7.4.4 TDataCache.AppendSummary Method

Pascal

```
function AppendSummary(Value: TSummary): Integer;
```

C++

```
__fastcall int AppendSummary(TSummary Value);
```

Description

This is AppendSummary, a member of class TDataCache.

111.1.7.4.5 TDataCache.CalcSubTotals Method

Pascal

```
procedure CalcSubTotals;
```

C++

```
__fastcall CalcSubTotals();
```

Description

This is CalcSubTotals, a member of class TDataCache.

111.1.7.4.6 TDataCache.ClearIndexInfo Method

Pascal

```
procedure ClearIndexInfo;
```

C++

```
__fastcall ClearIndexInfo();
```

Description

This is ClearIndexInfo, a member of class TDataCache.

111.1.7.4.7 TDataCache.CreateTable Method

Pascal

```
procedure CreateTable(Const Filename: String);
```

C++

```
__fastcall CreateTable(const AnsiString Filename);
```

Description

This is CreateTable, a member of class TDataCache.

111.1.7.4.8 TDataCache.DimensionFromFieldName Method

Pascal

```
function DimensionFromFieldName(FldName: string): TDimension;
```

C++

```
__fastcall TDimension DimensionFromFieldName(AnsiString FldName);
```

Description

This is DimensionFromFieldName, a member of class TDataCache.

111.1.7.4.9 TDataCache.FreeCache Method

Pascal

```
procedure FreeCache;
```

C++

```
__fastcall FreeCache();
```

Description

This is FreeCache, a member of class TDataCache.

111.1.7.4.10 TDataCache.GetDimensionMember Method

Pascal

```
function GetDimensionMember(DimIndex: Integer; MemberIndex: Integer): String;
```

C++

```
__fastcall AnsiString GetDimensionMember(int DimIndex, int MemberIndex);
```

Description

This is GetDimensionMember, a member of class TDataCache.

111.1.7.4.11 TDataCache.GetDimensionMemberAsVariant Method

Pascal

```
function GetDimensionMemberAsVariant(DimIndex: Integer; MemberIndex: Integer): Variant;
```

C++

```
__fastcall Variant GetDimensionMemberAsVariant(int DimIndex, int MemberIndex);
```

Description

This is GetDimensionMemberAsVariant, a member of class TDataCache.

111.1.7.4.12 TDataCache.GetDimensionMemberCount Method

Pascal

```
function GetDimensionMemberCount(DimIndex: Integer): Integer;
```

C++

```
__fastcall int GetDimensionMemberCount(int DimIndex);
```

Description

This is GetDimensionMemberCount, a member of class TDataCache.

111.1.7.4.13 TDataCache.GetDimensionName Method

Pascal

```
function GetDimensionName(DimIndex: Integer): String;
```

C++

```
__fastcall AnsiString GetDimensionName(int DimIndex);
```

Description

This is GetDimensionName, a member of class TDataCache.

111.1.7.4.14 TDataCache.GetDomain Method

Pascal

```
function GetDomain(DimensionIDs: TIntArray; nDims: Integer; ATotals: Boolean; Domain:
TTwoDimArray): Integer;
```

C++

```
__fastcall int GetDomain(TIntArray DimensionIDs, int nDims, Boolean ATotals, TTwoDimArray
Domain);
```

Description

This is GetDomain, a member of class TDataCache.

111.1.7.4.15 TDataCache.GetIndexCount Method

Pascal

```
function GetIndexCount: Integer;
```

C++

```
__fastcall int GetIndexCount();
```

Description

This is GetIndexCount, a member of class TDataCache.

111.1.7.4.16 TDataCache.GetMemoryUsage Method

Pascal

```
function GetMemoryUsage: Integer;
```

C++

```
__fastcall int GetMemoryUsage();
```

Description

This is GetMemoryUsage, a member of class TDataCache.

111.1.7.4.17 TDataCache.GetSummaryAsString Method

Pascal

```
function GetSummaryAsString(SumIndex: TSmallIntArray): String;
```

C++

```
__fastcall AnsiString GetSummaryAsString(TSmallIntArray SumIndex);
```

Description

This is GetSummaryAsString, a member of class TDataCache.

111.1.7.4.18 TDataCache.GetSummaryAsVariant Method

Pascal

```
function GetSummaryAsVariant(SumIndex: TSmallIntArray): Variant;
```

C++

```
__fastcall Variant GetSummaryAsVariant(TSmallIntArray SumIndex);
```

Description

This is GetSummaryAsVariant, a member of class TDataCache.

111.1.7.4.19 TDataCache.GetSummaryName Method

Pascal

```
function GetSummaryName(ISum: Integer): String;
```

C++

```
__fastcall AnsiString GetSummaryName(int ISum);
```

Description

This is GetSummaryName, a member of class TDataCache.

111.1.7.4.20 TDataCache.HasSubTotals Method

Pascal

```
function HasSubTotals(SumIndex: TSmallIntArray): Boolean;
```

C++

```
__fastcall Boolean HasSubTotals(TSmallIntArray SumIndex);
```

Description

This is HasSubTotals, a member of class TDataCache.

111.1.7.4.21 TDataCache.HasValidSubTotals Method

Pascal

```
function HasValidSubTotals(Summary: TSummary; SumIndex: TSmallIntArray): Boolean;
```

C++

```
__fastcall Boolean HasValidSubTotals(TSummary Summary, TSmallIntArray SumIndex);
```

Description

This is HasValidSubTotals, a member of class TDataCache.

111.1.7.4.22 TDataCache.IncSummaryIndex Method

Pascal

```
function IncSummaryIndex(Summary: TSummary; SumIndex: TSmallIntArray; rangeCount:
TSmallIntArray; var bGroupBreak: Boolean): Boolean;
```

C++

```
__fastcall Boolean IncSummaryIndex(TSummary Summary, TSmallIntArray SumIndex, TSmallIntArray
rangeCount, Boolean bGroupBreak);
```

Description

This is IncSummaryIndex, a member of class TDataCache.

111.1.7.4.23 TDataCache.IsDimension Method

Pascal

```
function IsDimension(Position: Integer): Boolean;
```

C++

```
__fastcall Boolean IsDimension(int Position);
```

Description

This is IsDimension, a member of class TDataCache.

111.1.7.4.24 TDataCache.IsSummary Method

Pascal

```
function IsSummary(Position: Integer): Boolean;
```

C++

```
__fastcall Boolean IsSummary(int Position);
```

Description

This is IsSummary, a member of class TDataCache.

111.1.7.4.25 TDataCache.SummaryFromCubeDimIndex Method

Pascal

```
function SummaryFromCubeDimIndex(Index: Integer): TSummary;
```

C++

```
__fastcall TSummary SummaryFromCubeDimIndex(int Index);
```

Description

This is SummaryFromCubeDimIndex, a member of class TDataCache.

111.1.7.4.26 TDataCache.SummaryFromFieldName Method

Pascal

```
function SummaryFromFieldName(FldName: string): TSummary;
```

C++

```
__fastcall TSummary SummaryFromFieldName(AnsiString FldName);
```

Description

This is SummaryFromFieldName, a member of class TDataCache.

111.1.7.4.27 TDataCache.SummaryFromPosition Method

Pascal

```
function SummaryFromPosition(Position: Integer): TSummary;
```

C++

```
__fastcall TSummary SummaryFromPosition(int Position);
```

Description

This is SummaryFromPosition, a member of class TDataCache.

111.1.7.5 TDataCache Properties

The properties of the TDataCache class are listed here.

Public Properties

	Name	Description
	Active (see page 17806)	This is Active, a member of class TDataCache.
	CurrentSummary (see page 17807)	This is CurrentSummary, a member of class TDataCache.
R	DimensionCount (see page 17807)	This is DimensionCount, a member of class TDataCache.
	Dimensions (see page 17807)	This is Dimensions, a member of class TDataCache.
	ErrorCode (see page 17807)	This is ErrorCode, a member of class TDataCache.
	HasIndex (see page 17807)	This is HasIndex, a member of class TDataCache.
R	IndexCount (see page 17808)	This is IndexCount, a member of class TDataCache.
	PreCalculateTotals (see page 17808)	This is PreCalculateTotals, a member of class TDataCache.
	Sparsing (see page 17808)	This is Sparsing, a member of class TDataCache.
	Success (see page 17808)	This is Success, a member of class TDataCache.
R	Summaries (see page 17809)	This is Summaries, a member of class TDataCache.
R	SummaryCount (see page 17809)	This is SummaryCount, a member of class TDataCache.

111.1.7.5.1 TDataCache.Active Property

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

This is Active, a member of class TDataCache.

111.1.7.5.2 TDataCache.CurrentSummary Property

Pascal

```
property CurrentSummary: Integer;
```

C++

```
__property int CurrentSummary;
```

Description

This is CurrentSummary, a member of class TDataCache.

111.1.7.5.3 TDataCache.DimensionCount Property

Pascal

```
property DimensionCount: Integer;
```

C++

```
__property int DimensionCount;
```

Description

This is DimensionCount, a member of class TDataCache.

111.1.7.5.4 TDataCache.Dimensions Property

Pascal

```
property Dimensions [Index: Integer]: TDimension;
```

C++

```
__property TDimension Dimensions[int Index];
```

Description

This is Dimensions, a member of class TDataCache.

111.1.7.5.5 TDataCache.ErrorCode Property

Pascal

```
property ErrorCode: Integer;
```

C++

```
__property int ErrorCode;
```

Description

This is ErrorCode, a member of class TDataCache.

111.1.7.5.6 TDataCache.HasIndex Property

Pascal

```
property HasIndex: Boolean;
```

C++

```
__property Boolean HasIndex;
```

Description

This is HasIndex, a member of class TDataCache.

111.1.7.5.7 TDataCache.IndexCount Property

Pascal

```
property IndexCount: Integer;
```

C++

```
__property int IndexCount;
```

Description

This is IndexCount, a member of class TDataCache.

111.1.7.5.8 TDataCache.PreCalculateTotals Property

Pascal

```
property PreCalculateTotals: Boolean;
```

C++

```
__property Boolean PreCalculateTotals;
```

Description

This is PreCalculateTotals, a member of class TDataCache.

111.1.7.5.9 TDataCache.Sparsing Property

Pascal

```
property Sparsing: Boolean;
```

C++

```
__property Boolean Sparsing;
```

Description

This is Sparsing, a member of class TDataCache.

111.1.7.5.10 TDataCache.Success Property

Pascal

```
property Success: Boolean;
```

C++

```
__property Boolean Success;
```

Description

This is Success, a member of class TDataCache.

111.1.7.5.11 TDataCache.Summaries Property

Pascal

```
property Summaries [Index: Integer]: TSummary;
```

C++

```
__property TSummary Summaries[int Index];
```

Description

This is Summaries, a member of class TDataCache.

111.1.7.5.12 TDataCache.SummaryCount Property

Pascal

```
property SummaryCount: Integer;
```

C++

```
__property int SummaryCount;
```

Description

This is SummaryCount, a member of class TDataCache.

111.1.8 TDimension Class

Class Hierarchy

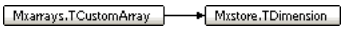

Pascal

```
TDimension = class(TCustomArray);
```

C++

```
class TDimension : public TCustomArray;
```

File

Mxstore

Description

This is class Mxstore.TDimension.

111.1.8.1 TDimension Members

The following tables list the members exposed by TDimension.

Protected Methods

	Name	Description
	GetCompProc (see page 17330)	This is GetCompProc, a member of class TCustomArray.
	GetDups (see page 17331)	This is GetDups, a member of class TCustomArray.
	GetItem (see page 17332)	This is GetItem, a member of class TCustomArray.

	GetMemberCount (see page 17332)	This is GetMemberCount, a member of class TCustomArray.
	GetSort (see page 17332)	This is GetSort, a member of class TCustomArray.
	SetCompProc (see page 17333)	This is SetCompProc, a member of class TCustomArray.
	SetDups (see page 17333)	This is SetDups, a member of class TCustomArray.
	SetItem (see page 17333)	This is SetItem, a member of class TCustomArray.
	SetSort (see page 17334)	This is SetSort, a member of class TCustomArray.

Public Methods

	Name	Description
	Add (see page 17330)	This is Add, a member of class TCustomArray.
	Assign (see page 17330)	This is Assign, a member of class TCustomArray.
	AssignSorted (see page 17811)	This is AssignSorted, a member of class TDimension.
	ClearFlag (see page 17811)	This is ClearFlag, a member of class TDimension.
	ConvertVar (see page 17330)	This is ConvertVar, a member of class TCustomArray.
	Create (see page 17810)	This is Create, a member of class TDimension.
V	Destroy (see page 17811)	This is Destroy, a member of class TDimension.
	GetCurrency (see page 17331)	This is GetCurrency, a member of class TCustomArray.
	GetDouble (see page 17331)	This is GetDouble, a member of class TCustomArray.
	GetInteger (see page 17331)	This is GetInteger, a member of class TCustomArray.
	HasFlag (see page 17812)	This is HasFlag, a member of class TDimension.
	IndexOf (see page 17332)	This is IndexOf, a member of class TCustomArray.
	IsBlank (see page 17332)	This is IsBlank, a member of class TCustomArray.
	IsString (see page 17812)	This is IsString, a member of class TDimension.
V	MemoryUsage (see page 17333)	This is MemoryUsage, a member of class TCustomArray.
	SetFlag (see page 17812)	This is SetFlag, a member of class TDimension.
	SetRangeCounting (see page 17812)	This is SetRangeCounting, a member of class TDimension.
	SetSize (see page 17334)	This is SetSize, a member of class TCustomArray.

Public Properties

	Name	Description
W	Attributes (see page 17813)	This is Attributes, a member of class TDimension.
	BlankBoolVal (see page 17334)	This is BlankBoolVal, a member of class TCustomArray.
R	BlankCount (see page 17335)	This is BlankCount, a member of class TCustomArray.
	BlankDateVal (see page 17335)	This is BlankDateVal, a member of class TCustomArray.
	BlankStringVal (see page 17335)	This is BlankStringVal, a member of class TCustomArray.
	CompareProc (see page 17335)	This is CompareProc, a member of class TCustomArray.
R	DataType (see page 17335)	This is DataType, a member of class TCustomArray.
	DimensionName (see page 17813)	This is DimensionName, a member of class TDimension.
	Duplicates (see page 17336)	This is Duplicates, a member of class TCustomArray.
R	FieldDefinition (see page 17813)	This is FieldDefinition, a member of class TDimension.
	FieldName (see page 17813)	This is FieldName, a member of class TDimension.
	FieldType (see page 17814)	This is FieldType, a member of class TDimension.
	Items (see page 17336)	This is Items, a member of class TCustomArray.
R	List (see page 17336)	This is List, a member of class TCustomArray.
R	MemberCount (see page 17336)	This is MemberCount, a member of class TCustomArray.
	Position (see page 17814)	This is Position, a member of class TDimension.
	Range (see page 17814)	This is Range, a member of class TDimension.
	Sorted (see page 17337)	This is Sorted, a member of class TCustomArray.

111.1.8.2 TDimension.Create Constructor

Pascal

```
constructor Create(Items: Cardinal; DataType: TFieldType);
```

C++

```
__fastcall TDimension(unsigned Items, TFieldType DataType);
```

Description

This is Create, a member of class TDimension.

111.1.8.3 TDimension.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDimension();
```

Description

This is Destroy, a member of class TDimension.

111.1.8.4 TDimension Methods

The methods of the TDimension class are listed here.

Public Methods

	Name	Description
	AssignSorted (see page 17811)	This is AssignSorted, a member of class TDimension.
	ClearFlag (see page 17811)	This is ClearFlag, a member of class TDimension.
	HasFlag (see page 17812)	This is HasFlag, a member of class TDimension.
	IsString (see page 17812)	This is IsString, a member of class TDimension.
	SetFlag (see page 17812)	This is SetFlag, a member of class TDimension.
	SetRangeCounting (see page 17812)	This is SetRangeCounting, a member of class TDimension.

111.1.8.4.1 TDimension.AssignSorted Method

Pascal

```
procedure AssignSorted(Dim: TDimension; bUnique: Boolean);
```

C++

```
__fastcall AssignSorted(TDimension Dim, Boolean bUnique);
```

Description

This is AssignSorted, a member of class TDimension.

111.1.8.4.2 TDimension.ClearFlag Method

Pascal

```
procedure ClearFlag(aFlag: TDimFlags);
```

C++

```
__fastcall ClearFlag(TDimFlags aFlag);
```

Description

This is ClearFlag, a member of class TDimension.

111.1.8.4.3 TDimension.HasFlag Method

Pascal

```
function HasFlag(aFlag: TDimFlags): Boolean;
```

C++

```
__fastcall Boolean HasFlag(TDimFlags aFlag);
```

Description

This is HasFlag, a member of class TDimension.

111.1.8.4.4 TDimension.IsString Method

Pascal

```
function IsString: Boolean;
```

C++

```
__fastcall Boolean IsString();
```

Description

This is IsString, a member of class TDimension.

111.1.8.4.5 TDimension.SetFlag Method

Pascal

```
procedure SetFlag(aFlag: TDimFlags);
```

C++

```
__fastcall SetFlag(TDimFlags aFlag);
```

Description

This is SetFlag, a member of class TDimension.

111.1.8.4.6 TDimension.SetRangeCounting Method

Pascal

```
procedure SetRangeCounting(bRange: Boolean);
```

C++

```
__fastcall SetRangeCounting(Boolean bRange);
```

Description

This is SetRangeCounting, a member of class TDimension.

111.1.8.5 TDimension Properties

The properties of the TDimension class are listed here.

Public Properties

	Name	Description
W	Attributes (see page 17813)	This is Attributes, a member of class TDimension.
	DimensionName (see page 17813)	This is DimensionName, a member of class TDimension.
R	FieldDefinition (see page 17813)	This is FieldDefinition, a member of class TDimension.
	FieldName (see page 17813)	This is FieldName, a member of class TDimension.
	FieldType (see page 17814)	This is FieldType, a member of class TDimension.
	Position (see page 17814)	This is Position, a member of class TDimension.
	Range (see page 17814)	This is Range, a member of class TDimension.

111.1.8.5.1 TDimension.Attributes Property

Pascal

```
property Attributes: TDimFlags;
```

C++

```
__property TDimFlags Attributes;
```

Description

This is Attributes, a member of class TDimension.

111.1.8.5.2 TDimension.DimensionName Property

Pascal

```
property DimensionName: String;
```

C++

```
__property AnsiString DimensionName;
```

Description

This is DimensionName, a member of class TDimension.

111.1.8.5.3 TDimension.FieldDefinition Property

Pascal

```
property FieldDefinition: TFieldDefinition;
```

C++

```
__property TFieldDefinition FieldDefinition;
```

Description

This is FieldDefinition, a member of class TDimension.

111.1.8.5.4 TDimension.FieldName Property

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;
```

Description

This is FieldName, a member of class TDimension.

111.1.8.5.5 TDimension.FieldType Property

Pascal

```
property FieldType: TFieldType;
```

C++

```
__property TFieldType FieldType;
```

Description

This is FieldType, a member of class TDimension.

111.1.8.5.6 TDimension.Position Property

Pascal

```
property Position: Integer;
```

C++

```
__property int Position;
```

Description

This is Position, a member of class TDimension.

111.1.8.5.7 TDimension.Range Property

Pascal

```
property Range: Integer;
```

C++

```
__property int Range;
```

Description

This is Range, a member of class TDimension.

111.1.9 TFieldDefinition Class

Class Hierarchy

Mxstore.TFieldDefinition

Pascal

```
TFieldDefinition = class;
```

C++

```
class TFieldDefinition;
```

File

Mxstore

Description

This is class Mxstore.TFieldDefinition.

111.1.9.1 TFieldDefinition Members

The following tables list the members exposed by TFieldDefinition.

Public Methods

	Name	Description
	Create (see page 17815)	This is Create, a member of class TFieldDefinition.
	FormatVariantToStr (see page 17815)	This is FormatVariantToStr, a member of class TFieldDefinition.
	SetFieldType (see page 17816)	This is SetFieldType, a member of class TFieldDefinition.
	SetName (see page 17816)	This is SetName, a member of class TFieldDefinition.

Public Properties

	Name	Description
	DisplayName (see page 17816)	This is DisplayName, a member of class TFieldDefinition.
	FieldNo (see page 17817)	This is FieldNo, a member of class TFieldDefinition.
	FieldType (see page 17817)	This is FieldType, a member of class TFieldDefinition.
	FormatString (see page 17817)	This is FormatString, a member of class TFieldDefinition.
	Precision (see page 17817)	This is Precision, a member of class TFieldDefinition.
	Width (see page 17817)	This is Width, a member of class TFieldDefinition.

111.1.9.2 TFieldDefinition.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TFieldDefinition();
```

Description

This is Create, a member of class TFieldDefinition.

111.1.9.3 TFieldDefinition Methods

The methods of the TFieldDefinition class are listed here.

Public Methods

	Name	Description
	FormatVariantToStr (see page 17815)	This is FormatVariantToStr, a member of class TFieldDefinition.
	SetFieldType (see page 17816)	This is SetFieldType, a member of class TFieldDefinition.
	SetName (see page 17816)	This is SetName, a member of class TFieldDefinition.

111.1.9.3.1 TFieldDefinition.FormatVariantToStr Method

Pascal

```
function FormatVariantToStr(Value: Variant): string;
```

C++

```
__fastcall AnsiString FormatVariantToStr(Variant Value);
```

Description

This is FormatVariantToStr, a member of class TFieldDefinition.

111.1.9.3.2 TFieldDefinition.SetFieldType Method

Pascal

```
procedure SetFieldType(FType: TFieldType);
```

C++

```
__fastcall SetFieldType(TFieldType FType);
```

Description

This is SetFieldType, a member of class TFieldDefinition.

111.1.9.3.3 TFieldDefinition.SetName Method

Pascal

```
procedure SetName(Value: String);
```

C++

```
__fastcall SetName(AnsiString Value);
```

Description

This is SetName, a member of class TFieldDefinition.

111.1.9.4 TFieldDefinition Properties

The properties of the TFieldDefinition class are listed here.

Public Properties

	Name	Description
	DisplayName (see page 17816)	This is DisplayName, a member of class TFieldDefinition.
	FieldNo (see page 17817)	This is FieldNo, a member of class TFieldDefinition.
	FieldType (see page 17817)	This is FieldType, a member of class TFieldDefinition.
	FormatString (see page 17817)	This is FormatString, a member of class TFieldDefinition.
	Precision (see page 17817)	This is Precision, a member of class TFieldDefinition.
	Width (see page 17817)	This is Width, a member of class TFieldDefinition.

111.1.9.4.1 TFieldDefinition.DisplayName Property

Pascal

```
property DisplayName: String;
```

C++

```
__property AnsiString DisplayName;
```

Description

This is DisplayName, a member of class TFieldDefinition.

111.1.9.4.2 TFieldDefinition.FieldNo Property

Pascal

```
property FieldNo: Integer;
```

C++

```
__property int FieldNo;
```

Description

This is FieldNo, a member of class TFieldDefinition.

111.1.9.4.3 TFieldDefinition.FieldType Property

Pascal

```
property FieldType: TFieldType;
```

C++

```
__property TFieldType FieldType;
```

Description

This is FieldType, a member of class TFieldDefinition.

111.1.9.4.4 TFieldDefinition.FormatString Property

Pascal

```
property FormatString: string;
```

C++

```
__property AnsiString FormatString;
```

Description

This is FormatString, a member of class TFieldDefinition.

111.1.9.4.5 TFieldDefinition.Precision Property

Pascal

```
property Precision: Integer;
```

C++

```
__property int Precision;
```

Description

This is Precision, a member of class TFieldDefinition.

111.1.9.4.6 TFieldDefinition.Width Property

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

This is Width, a member of class TFieldDefinition.

111.1.10 TIndexInfo Class

Class Hierarchy

Mxstore.TIndexInfo

Pascal

```
TIndexInfo = class;
```

C++

```
class TIndexInfo;
```

File

Mxstore

Description

This is class Mxstore.TIndexInfo.

111.1.10.1 TIndexInfo Members

The following tables list the members exposed by TIndexInfo.

Public Methods

	Name	Description
	Add (see page 17819)	This is Add, a member of class TIndexInfo.
	AddOffset (see page 17819)	This is AddOffset, a member of class TIndexInfo.
	Create (see page 17818)	This is Create, a member of class TIndexInfo.
V	Destroy (see page 17819)	This is Destroy, a member of class TIndexInfo.
	GetCapacity (see page 17820)	This is GetCapacity, a member of class TIndexInfo.
	IsSparse (see page 17820)	This is IsSparse, a member of class TIndexInfo.
	IsSparseAgg (see page 17820)	This is IsSparseAgg, a member of class TIndexInfo.
	LockIndex (see page 17820)	This is LockIndex, a member of class TIndexInfo.
	SetCapacity (see page 17821)	This is SetCapacity, a member of class TIndexInfo.
	UnlockIndex (see page 17821)	This is UnlockIndex, a member of class TIndexInfo.

Public Properties

	Name	Description
	AddAggs (see page 17821)	This is AddAggs, a member of class TIndexInfo.
	Capacity (see page 17821)	This is Capacity, a member of class TIndexInfo.
R	Count (see page 17822)	This is Count, a member of class TIndexInfo.

111.1.10.2 TIndexInfo.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TIndexInfo();
```

Description

This is Create, a member of class TIndexInfo.

111.1.10.3 TIndexInfo.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TIndexInfo();
```

Description

This is Destroy, a member of class TIndexInfo.

111.1.10.4 TIndexInfo Methods

The methods of the TIndexInfo class are listed here.

Public Methods

	Name	Description
	Add (see page 17819)	This is Add, a member of class TIndexInfo.
	AddOffset (see page 17819)	This is AddOffset, a member of class TIndexInfo.
	GetCapacity (see page 17820)	This is GetCapacity, a member of class TIndexInfo.
	IsSparse (see page 17820)	This is IsSparse, a member of class TIndexInfo.
	IsSparseAgg (see page 17820)	This is IsSparseAgg, a member of class TIndexInfo.
	LockIndex (see page 17820)	This is LockIndex, a member of class TIndexInfo.
	SetCapacity (see page 17821)	This is SetCapacity, a member of class TIndexInfo.
	UnlockIndex (see page 17821)	This is UnlockIndex, a member of class TIndexInfo.

111.1.10.4.1 TIndexInfo.Add Method

Pascal

```
function Add(pIdxRec: PIndexInfoRec): Integer;
```

C++

```
__fastcall int Add(PIndexInfoRec pIdxRec);
```

Description

This is Add, a member of class TIndexInfo.

111.1.10.4.2 TIndexInfo.AddOffset Method

Pascal

```
procedure AddOffset(Index: Integer; IdxType: Integer);
```

C++

```
__fastcall AddOffset(int Index, int IdxType);
```

Description

This is AddOffset, a member of class TIndexInfo.

111.1.10.4.3 TIndexInfo.GetCapacity Method

Pascal

```
function GetCapacity: Integer;
```

C++

```
__fastcall int GetCapacity();
```

Description

This is GetCapacity, a member of class TIndexInfo.

111.1.10.4.4 TIndexInfo.IsSparse Method

Pascal

```
function IsSparse(Index: Integer): Boolean;
```

C++

```
__fastcall Boolean IsSparse(int Index);
```

Description

This is IsSparse, a member of class TIndexInfo.

111.1.10.4.5 TIndexInfo.IsSparseAgg Method

Pascal

```
function IsSparseAgg(Index: Integer): Boolean;
```

C++

```
__fastcall Boolean IsSparseAgg(int Index);
```

Description

This is IsSparseAgg, a member of class TIndexInfo.

111.1.10.4.6 TIndexInfo.LockIndex Method

Pascal

```
function LockIndex: TIntArray;
```

C++

```
__fastcall TIntArray LockIndex();
```

Description

This is LockIndex, a member of class TIndexInfo.

111.1.10.4.7 TIndexInfo.SetCapacity Method

Pascal

```
procedure SetCapacity(Value: Integer);
```

C++

```
__fastcall SetCapacity(int Value);
```

Description

This is SetCapacity, a member of class TIndexInfo.

111.1.10.4.8 TIndexInfo.UnlockIndex Method

Pascal

```
procedure UnlockIndex;
```

C++

```
__fastcall UnlockIndex();
```

Description

This is UnlockIndex, a member of class TIndexInfo.

111.1.10.5 TIndexInfo Properties

The properties of the TIndexInfo class are listed here.

Public Properties

	Name	Description
	AddAggs (see page 17821)	This is AddAggs, a member of class TIndexInfo.
	Capacity (see page 17821)	This is Capacity, a member of class TIndexInfo.
R	Count (see page 17822)	This is Count, a member of class TIndexInfo.

111.1.10.5.1 TIndexInfo.AddAggs Property

Pascal

```
property AddAggs: Boolean;
```

C++

```
__property Boolean AddAggs;
```

Description

This is AddAggs, a member of class TIndexInfo.

111.1.10.5.2 TIndexInfo.Capacity Property

Pascal

```
property Capacity: Integer;
```

C++

```
__property int Capacity;
```

Description

This is Capacity, a member of class TIndexInfo.

111.1.10.5.3 TIndexInfo.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TIndexInfo.

111.1.11 TMultiDimDataLink Class

Class Hierarchy

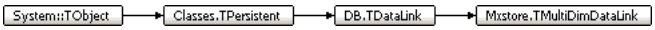

Pascal

```
TMultiDimDataLink = class(TDataLink);
```

C++

```
class TMultiDimDataLink : public TDataLink;
```

File

Mxstore

Description

This is class Mxstore.TMultiDimDataLink.

111.1.11.1 TMultiDimDataLink Members

The following tables list the members exposed by TMultiDimDataLink.

Protected Methods

	Name	Description
	ActiveChanged (see page 17825)	Responds to changes in the Active property.
	AddDimension (see page 17825)	This is AddDimension, a member of class TMultiDimDataLink.
	AddSummary (see page 17826)	This is AddSummary, a member of class TMultiDimDataLink.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CheckBrowseMode (see page 6880)	Allows the data-aware object to respond just before an event that changes the dataset state.
	CreateSummaryIndex (see page 17826)	This is CreateSummaryIndex, a member of class TMultiDimDataLink.
	DataEvent (see page 6881)	Responds to various events that occur while working with data.
	DataSetChanged (see page 6881)	Responds to changes in the dataset.
	DataSetScrolled (see page 6882)	Allows a response to scrolling the representation of the dataset within the data-aware object.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EditingChanged (see page 6883)	Responds to changes in the editing state of the data source.
	FetchAndBinValues (see page 17826)	This is FetchAndBinValues, a member of class TMultiDimDataLink.
	FetchValues (see page 17826)	This is FetchValues, a member of class TMultiDimDataLink.
	FocusControl (see page 6884)	Provides the interface for a method to force focus to the data-aware component associated with the TDataLink (see page 6876).
	GetActiveRecord (see page 6884)	This is GetActiveRecord, a member of class TDataLink.
	GetBOF (see page 6884)	This is GetBOF, a member of class TDataLink.
	GetBufferCount (see page 6884)	This is GetBufferCount, a member of class TDataLink.
	GetEOF (see page 6885)	This is GetEOF, a member of class TDataLink.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRecordCount (see page 6885)	This is GetRecordCount, a member of class TDataLink.
	LayoutChanged (see page 17827)	Responds to changes in the representation of the data by the data-aware object.
	MoveBy (see page 6885)	Repositions the dataset to another record relative to its active record.
	RecordChanged (see page 6886)	Responds to changes in the contents of the current record or field of the dataset.
	SetActiveRecord (see page 6886)	This is SetActiveRecord, a member of class TDataLink.
	SetBufferCount (see page 6887)	This is SetBufferCount, a member of class TDataLink.
	UpdateCache (see page 17827)	This is UpdateCache, a member of class TMultiDimDataLink.
	UpdateData (see page 6887)	Provides an interface for writing edits to a record in the dataset.
	UpdateDimensions (see page 17827)	This is UpdateDimensions, a member of class TMultiDimDataLink.
	UpdateFormatStrings (see page 17827)	This is UpdateFormatStrings, a member of class TMultiDimDataLink.

Protected Properties

	Name	Description
	VisualControl (see page 6894)	Indicates whether a visual control uses the datalink.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 17824)	This is Create, a member of class TMultiDimDataLink.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 17824)	Destroys an instance of TDataLink.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6882)	Attempts to drive the data source into an editing state if it is not already in one.
	ExecuteAction (see page 6883)	Invokes an action with the datasource as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	UpdateAction (see page 6887)	Updates an action to reflect the current state of the TDataLink (see page 6876).
	UpdateRecord (see page 6888)	Responds to notifications that edits should be posted to the dataset.

Public Properties

	Name	Description
R	Active (see page 6888)	Specifies whether the dataset that TDataLink (see page 6876) manages is active.
	ActiveRecord (see page 6889)	Specifies the index of the current record within the internal record buffer maintained by the dataset for the Owner of the TDataLink (see page 6876) object.
R	BOF (see page 6889)	Indicates whether the first record in a dataset is active.
	BufferCount (see page 6890)	Specifies the number of records cached by the dataset for the Owner of the TDataLink (see page 6876) object.
R	DataSet (see page 6890)	Specifies the TDataSet (see page 6894) object that represents the data that this TDataLink (see page 6876) is tracking.
	DataSource (see page 6891)	Specifies the TDataSource (see page 7070) object that the Owner of the TDataLink (see page 6876) uses to connect to the dataset.
	DataSourceFixed (see page 6891)	Indicates whether the DataSource (see page 6891) property can be set.
R	Editing (see page 6892)	Indicates whether the DataSource (see page 6891) is in edit mode.
R	Eof (see page 6892)	Indicates whether the last record in a dataset is active.
	ReadOnly (see page 6893)	Indicates whether the dataset for this TDataLink (see page 6876) can be modified.
R	RecordCount (see page 6893)	Indicates the number of records in the internal record buffer maintained by the dataset.

111.1.11.2 TMultiDimDataLink.Create Constructor

Pascal

```
constructor Create(AStore: TCustomDataStore);
```

C++

```
__fastcall TMultiDimDataLink(TCustomDataStore AStore);
```

Description

This is Create, a member of class TMultiDimDataLink.

111.1.11.3 TMultiDimDataLink.Destroy Destructor

Destroys an instance of TDataLink.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMultiDimDataLink();
```

Description

The TDataLink object should be destroyed in the destructor of its Owner, where that Owner calls Create from its constructor.

Do not call Destroy directly in an application. Instead, call the Free method. Free checks that the TDataLink reference is not nil and only then calls Destroy.

Before calling the inherited destructor, Destroy removes any reference to the TDataLink from the data source object.

See Also

DataSource (see page 6891)

Free (see page 22300)

111.1.11.4 TMultiDimDataLink Methods

The methods of the TMultiDimDataLink class are listed here.

Protected Methods

	Name	Description
V	ActivateChanged (see page 17825)	Responds to changes in the Active property.
	AddDimension (see page 17825)	This is AddDimension, a member of class TMultiDimDataLink.
	AddSummary (see page 17826)	This is AddSummary, a member of class TMultiDimDataLink.
	CreateSummaryIndex (see page 17826)	This is CreateSummaryIndex, a member of class TMultiDimDataLink.
	FetchAndBinValues (see page 17826)	This is FetchAndBinValues, a member of class TMultiDimDataLink.
	FetchValues (see page 17826)	This is FetchValues, a member of class TMultiDimDataLink.
V	LayoutChanged (see page 17827)	Responds to changes in the representation of the data by the data-aware object.
	UpdateCache (see page 17827)	This is UpdateCache, a member of class TMultiDimDataLink.
	UpdateDimensions (see page 17827)	This is UpdateDimensions, a member of class TMultiDimDataLink.
	UpdateFormatStrings (see page 17827)	This is UpdateFormatStrings, a member of class TMultiDimDataLink.

111.1.11.4.1 TMultiDimDataLink.ActiveChanged Method

Responds to changes in the Active property.

Pascal

```
procedure ActiveChanged; override;
```

C++

```
virtual __fastcall ActiveChanged();
```

Description

The ActiveChanged method defined by TDataLink merely provides an interface for a method that can respond to changes in the Active property. Derived objects that do not need to respond to such changes can allow the inherited method to ignore them.

See Also

Active (see page 6888)

111.1.11.4.2 TMultiDimDataLink.AddDimension Method

Pascal

```
function AddDimension(DimMap: TCubeDim; Fld: TField): Integer;
```

C++

```
__fastcall int AddDimension(TCubeDim DimMap, TField Fld);
```

Description

This is AddDimension, a member of class TMultiDimDataLink.

111.1.11.4.3 TMultiDimDataLink.AddSummary Method

Pascal

```
procedure AddSummary(DimMap: TCubeDim; Fld: TField);
```

C++

```
__fastcall AddSummary(TCubeDim DimMap, TField Fld);
```

Description

This is AddSummary, a member of class TMultiDimDataLink.

111.1.11.4.4 TMultiDimDataLink.CreateSummaryIndex Method

Pascal

```
procedure CreateSummaryIndex(DimAllList: TList);
```

C++

```
__fastcall CreateSummaryIndex(TList DimAllList);
```

Description

This is CreateSummaryIndex, a member of class TMultiDimDataLink.

111.1.11.4.5 TMultiDimDataLink.FetchAndBinValues Method

Pascal

```
procedure FetchAndBinValues(DimAllList: TList);
```

C++

```
__fastcall FetchAndBinValues(TList DimAllList);
```

Description

This is FetchAndBinValues, a member of class TMultiDimDataLink.

111.1.11.4.6 TMultiDimDataLink.FetchValues Method

Pascal

```
procedure FetchValues(DimAllList: TList);
```

C++

```
__fastcall FetchValues(TList DimAllList);
```

Description

This is FetchValues, a member of class TMultiDimDataLink.

111.1.11.4.7 TMultiDimDataLink.LayoutChanged Method

Responds to changes in the representation of the data by the data-aware object.

Pascal

```
procedure LayoutChanged; override;
```

C++

```
virtual __fastcall LayoutChanged();
```

Description

The LayoutChanged method simply calls the DataSetChanged method. LayoutChanged is intended to provide an interface for a method that can respond after changes in the representation of the data by the data-aware object. An example of such a change would be changing the number or order of the columns in a TCustomDBGrid.

See Also

DataSetChanged (see page 6881)

111.1.11.4.8 TMultiDimDataLink.UpdateCache Method

Pascal

```
procedure UpdateCache(Sender: TObject);
```

C++

```
__fastcall UpdateCache(TObject * Sender);
```

Description

This is UpdateCache, a member of class TMultiDimDataLink.

111.1.11.4.9 TMultiDimDataLink.UpdateDimensions Method

Pascal

```
procedure UpdateDimensions(DimAllList: TList);
```

C++

```
__fastcall UpdateDimensions(TList DimAllList);
```

Description

This is UpdateDimensions, a member of class TMultiDimDataLink.

111.1.11.4.10 TMultiDimDataLink.UpdateFormatStrings Method

Pascal

```
procedure UpdateFormatStrings;
```

C++

```
__fastcall UpdateFormatStrings();
```

Description

This is UpdateFormatStrings, a member of class TMultiDimDataLink.

111.1.12 TSummary Class

Class Hierarchy

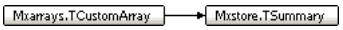

Pascal

```
TSummary = class(TCustomArray);
```

C++

```
class TSummary : public TCustomArray;
```

File

Mxstore

Description

This is class Mxstore.TSummary.

111.1.12.1 TSummary Members

The following tables list the members exposed by TSummary.

Protected Methods

	Name	Description
	GetCompProc (see page 17330)	This is GetCompProc, a member of class TCustomArray.
	GetDups (see page 17331)	This is GetDups, a member of class TCustomArray.
	GetItem (see page 17332)	This is GetItem, a member of class TCustomArray.
	GetMemberCount (see page 17332)	This is GetMemberCount, a member of class TCustomArray.
	GetSort (see page 17332)	This is GetSort, a member of class TCustomArray.
	HasFlag (see page 17831)	This is HasFlag, a member of class TSummary.
	SetAggregator (see page 17832)	This is SetAggregator, a member of class TSummary.
	SetCompProc (see page 17333)	This is SetCompProc, a member of class TCustomArray.
	SetDups (see page 17333)	This is SetDups, a member of class TCustomArray.
	SetItem (see page 17333)	This is SetItem, a member of class TCustomArray.
	SetSort (see page 17334)	This is SetSort, a member of class TCustomArray.

Public Events

	Name	Description
	SumMethod (see page 17834)	This is SumMethod, a member of class TSummary.

Public Methods

	Name	Description
	Add (see page 17330)	This is Add, a member of class TCustomArray.
	AddIndexInfo (see page 17830)	This is AddIndexInfo, a member of class TSummary.
	AddSubTotal (see page 17830)	This is AddSubTotal, a member of class TSummary.
	AddSum (see page 17830)	This is AddSum, a member of class TSummary.
	Assign (see page 17330)	This is Assign, a member of class TCustomArray.
	ClearTotals (see page 17831)	This is ClearTotals, a member of class TSummary.
	ConvertVar (see page 17330)	This is ConvertVar, a member of class TCustomArray.
	Create (see page 17829)	This is Create, a member of class TSummary.
	Destroy (see page 17829)	This is Destroy, a member of class TSummary.

	GetCurrency (see page 17331)	This is GetCurrency, a member of class TCustomArray.
	GetDouble (see page 17331)	This is GetDouble, a member of class TCustomArray.
	GetInteger (see page 17331)	This is GetInteger, a member of class TCustomArray.
	IndexOf (see page 17332)	This is IndexOf, a member of class TCustomArray.
	IsBlank (see page 17332)	This is IsBlank, a member of class TCustomArray.
	IsSparse (see page 17831)	This is IsSparse, a member of class TSummary.
V	MemoryUsage (see page 17831)	This is MemoryUsage, a member of class TSummary.
	SetSize (see page 17334)	This is SetSize, a member of class TCustomArray.
	UpdateIndexInfo (see page 17832)	This is UpdateIndexInfo, a member of class TSummary.

Public Properties

	Name	Description
	AggDefinition (see page 17832)	This is AggDefinition, a member of class TSummary.
W	Attributes (see page 17833)	This is Attributes, a member of class TSummary.
	BlankBoolVal (see page 17334)	This is BlankBoolVal, a member of class TCustomArray.
R	BlankCount (see page 17335)	This is BlankCount, a member of class TCustomArray.
	BlankDateVal (see page 17335)	This is BlankDateVal, a member of class TCustomArray.
	BlankStringVal (see page 17335)	This is BlankStringVal, a member of class TCustomArray.
	CompareProc (see page 17335)	This is CompareProc, a member of class TCustomArray.
	CubeDimIndex (see page 17833)	This is CubeDimIndex, a member of class TSummary.
R	DataType (see page 17335)	This is DataType, a member of class TCustomArray.
	Duplicates (see page 17336)	This is Duplicates, a member of class TCustomArray.
R	FieldDefinition (see page 17833)	This is FieldDefinition, a member of class TSummary.
	FieldName (see page 17833)	This is FieldName, a member of class TSummary.
	FieldType (see page 17833)	This is FieldType, a member of class TSummary.
R	IsDerived (see page 17834)	This is IsDerived, a member of class TSummary.
	Items (see page 17336)	This is Items, a member of class TCustomArray.
R	List (see page 17336)	This is List, a member of class TCustomArray.
R	MemberCount (see page 17336)	This is MemberCount, a member of class TCustomArray.
	Name (see page 17834)	This is Name, a member of class TSummary.
	Position (see page 17834)	This is Position, a member of class TSummary.
	Sorted (see page 17337)	This is Sorted, a member of class TCustomArray.

111.1.12.2 TSummary.Create Constructor

Pascal

```
constructor Create(Items: Cardinal; DataType: TFieldType);
```

C++

```
__fastcall TSummary(unsigned Items, TFieldType DataType);
```

Description

This is Create, a member of class TSummary.

111.1.12.3 TSummary.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSummary();
```

Description

This is Destroy, a member of class TSummary.

111.1.12.4 TSummary Methods

The methods of the TSummary class are listed here.

Protected Methods

	Name	Description
	HasFlag (see page 17831)	This is HasFlag, a member of class TSummary.
	SetAggregator (see page 17832)	This is SetAggregator, a member of class TSummary.

Public Methods

	Name	Description
	AddIndexInfo (see page 17830)	This is AddIndexInfo, a member of class TSummary.
	AddSubTotal (see page 17830)	This is AddSubTotal, a member of class TSummary.
	AddSum (see page 17830)	This is AddSum, a member of class TSummary.
	ClearTotals (see page 17831)	This is ClearTotals, a member of class TSummary.
	IsSparse (see page 17831)	This is IsSparse, a member of class TSummary.
	MemoryUsage (see page 17831)	This is MemoryUsage, a member of class TSummary.
	UpdateIndexInfo (see page 17832)	This is UpdateIndexInfo, a member of class TSummary.

111.1.12.4.1 TSummary.AddIndexInfo Method

Pascal

```
function AddIndexInfo(BTotal: Boolean; bSparse: Boolean; iAggOffset: Integer): Integer;
```

C++

```
__fastcall int AddIndexInfo(Boolean BTotal, Boolean bSparse, int iAggOffset);
```

Description

This is AddIndexInfo, a member of class TSummary.

111.1.12.4.2 TSummary.AddSubTotal Method

Pascal

```
function AddSubTotal(Value: Variant): Integer;
```

C++

```
__fastcall int AddSubTotal(Variant Value);
```

Description

This is AddSubTotal, a member of class TSummary.

111.1.12.4.3 TSummary.AddSum Method

Pascal

```
procedure AddSum(var SumIndex: TSmallIntArray; vNew: Variant);
```

C++

```
__fastcall AddSum(TSmallIntArray SumIndex, Variant vNew);
```

Description

This is AddSum, a member of class TSummary.

111.1.12.4.4 TSummary.ClearTotals Method

Pascal

```
procedure ClearTotals;
```

C++

```
__fastcall ClearTotals();
```

Description

This is ClearTotals, a member of class TSummary.

111.1.12.4.5 TSummary.HasFlag Method

Pascal

```
function HasFlag(aFlag: TDimFlags): Boolean;
```

C++

```
__fastcall Boolean HasFlag(TDimFlags aFlag);
```

Description

This is HasFlag, a member of class TSummary.

111.1.12.4.6 TSummary.IsSparse Method

Pascal

```
function IsSparse(Index: Integer): Boolean;
```

C++

```
__fastcall Boolean IsSparse(int Index);
```

Description

This is IsSparse, a member of class TSummary.

111.1.12.4.7 TSummary.MemoryUsage Method

Pascal

```
function MemoryUsage: Integer; override;
```

C++

```
virtual __fastcall int MemoryUsage();
```

Description

This is MemoryUsage, a member of class TSummary.

111.1.12.4.8 TSummary.SetAggregator Method

Pascal

```
function SetAggregator(aName: string; DimMap: TCubeDims; dimType: TDimFlags; var dIdx:
Integer): Boolean;
```

C++

```
__fastcall Boolean SetAggregator(AnsiString aName, TCubeDims DimMap, TDimFlags dimType, int
dIdx);
```

Description

This is SetAggregator, a member of class TSummary.

111.1.12.4.9 TSummary.UpdateIndexInfo Method

Pascal

```
procedure UpdateIndexInfo(Index: Integer; Value: Variant);
```

C++

```
__fastcall UpdateIndexInfo(int Index, Variant Value);
```

Description

This is UpdateIndexInfo, a member of class TSummary.

111.1.12.5 TSummary Properties

The properties of the TSummary class are listed here.

Public Properties

	Name	Description
	AggDefinition (see page 17832)	This is AggDefinition, a member of class TSummary.
W	Attributes (see page 17833)	This is Attributes, a member of class TSummary.
	CubeDimIndex (see page 17833)	This is CubeDimIndex, a member of class TSummary.
R	FieldDefinition (see page 17833)	This is FieldDefinition, a member of class TSummary.
	FieldName (see page 17833)	This is FieldName, a member of class TSummary.
	FieldType (see page 17833)	This is FieldType, a member of class TSummary.
R	IsDerived (see page 17834)	This is IsDerived, a member of class TSummary.
	Name (see page 17834)	This is Name, a member of class TSummary.
	Position (see page 17834)	This is Position, a member of class TSummary.

111.1.12.5.1 TSummary.AggDefinition Property

Pascal

```
property AggDefinition: TAggDefinition;
```

C++

```
__property TAggDefinition AggDefinition;
```

Description

This is AggDefinition, a member of class TSummary.

111.1.12.5.2 TSummary.Attributes Property

Pascal

```
property Attributes: TDimFlags;
```

C++

```
__property TDimFlags Attributes;
```

Description

This is Attributes, a member of class TSummary.

111.1.12.5.3 TSummary.CubeDimIndex Property

Pascal

```
property CubeDimIndex: Integer;
```

C++

```
__property int CubeDimIndex;
```

Description

This is CubeDimIndex, a member of class TSummary.

111.1.12.5.4 TSummary.FieldDefinition Property

Pascal

```
property FieldDefinition: TFieldDefinition;
```

C++

```
__property TFieldDefinition FieldDefinition;
```

Description

This is FieldDefinition, a member of class TSummary.

111.1.12.5.5 TSummary.FieldName Property

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;
```

Description

This is FieldName, a member of class TSummary.

111.1.12.5.6 TSummary.FieldType Property

Pascal

```
property FieldType: TFieldType;
```

C++

```
__property TFieldType FieldType;
```

Description

This is FieldType, a member of class TSummary.

111.1.12.5.7 TSummary.IsDerived Property

Pascal

```
property IsDerived: Boolean;
```

C++

```
__property Boolean IsDerived;
```

Description

This is IsDerived, a member of class TSummary.

111.1.12.5.8 TSummary.Name Property

Pascal

```
property Name: String;
```

C++

```
__property AnsiString Name;
```

Description

This is Name, a member of class TSummary.

111.1.12.5.9 TSummary.Position Property

Pascal

```
property Position: Integer;
```

C++

```
__property int Position;
```

Description

This is Position, a member of class TSummary.

111.1.12.6 TSummary Events

The events of the TSummary class are listed here.

Public Events

	Name	Description
	SumMethod (see page 17834)	This is SumMethod, a member of class TSummary.

111.1.12.6.1 TSummary.SumMethod Event

Pascal

```
property SumMethod: TSumMethod;
```

C++

```
__property TSumMethod SumMethod;
```

Description

This is SumMethod, a member of class TSummary.

111.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	TestMatch (see page 17836)	This is function Mxstore.TestMatch.

111.2.1 Mxstore.TestMatch Function

Pascal

```
function TestMatch(SumIdx: TSmallIntArray): Boolean;
```

C++

```
Boolean TestMatch(TSmallIntArray SumIdx);
```

File

Mxstore

Description

This is function Mxstore.TestMatch.

111.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TBuildType (see page 17837)	This is record Mxstore.TBuildType.
	TCacheStateFlags (see page 17837)	This is record Mxstore.TCacheStateFlags.
	TCalcTotalsFlags (see page 17838)	This is record Mxstore.TCalcTotalsFlags.
	TCubeDataState (see page 17838)	This is record Mxstore.TCubeDataState.
	TCubeState (see page 17839)	This is record Mxstore.TCubeState.
	TErrorAction (see page 17839)	This is record Mxstore.TErrorAction.
	TFormatType (see page 17840)	This is record Mxstore.TFormatType.
	TIndexFlag (see page 17840)	This is record Mxstore.TIndexFlag.
	TLookupStateFlags (see page 17841)	This is record Mxstore.TLookupStateFlags.

Records

	Name	Description
	TIndexInfoRec (see page 17841)	This is record Mxstore.TIndexInfoRec.

111.3.1 Mxstore.TBuildType Enumeration

Pascal

```
TBuildType = (
  btHardRebuild,
  btSoftRebuild,
  btNoRebuild
);
```

C++

```
enum TBuildType {
  btHardRebuild,
  btSoftRebuild,
  btNoRebuild
};
```

File

Mxstore

Description

This is record Mxstore.TBuildType.

111.3.2 Mxstore.TCacheStateFlags Enumeration

Pascal

```
TCacheStateFlags = (
```

```
    csHasIndex,
    csDirty,
    csSuccess,
    csShowProgress,
    csRefreshing
  );
```

C++

```
  enum TCacheStateFlags {
    csHasIndex,
    csDirty,
    csSuccess,
    csShowProgress,
    csRefreshing
  };
```

File

Mxstore

Description

This is record Mxstore.TCacheStateFlags.

111.3.3 Mxstore.TCalcTotalsFlags Enumeration

Pascal

```
  TCalcTotalsFlags = (
    ctPreCalc,
    ctRunning,
    ctNone
  );
```

C++

```
  enum TCalcTotalsFlags {
    ctPreCalc,
    ctRunning,
    ctNone
  };
```

File

Mxstore

Description

This is record Mxstore.TCalcTotalsFlags.

111.3.4 Mxstore.TCubeDataState Enumeration

Pascal

```
  TCubeDataState = (
    dsNoData,
    dsMetaData,
    dsDimensionData,
    dsAllData
  );
```

C++

```
enum TCubeDataState {
  dsNoData,
  dsMetaData,
  dsDimensionData,
  dsAllData
};
```

File

Mxstore

Description

This is record Mxstore.TCubeDataState.

111.3.5 Mxstore.TCubeState Enumeration

Pascal

```
TCubeState = (
  dcInactive,
  dcBrowseMetaData,
  dcBrowseMemberData,
  dcBrowseAllData
);
```

C++

```
enum TCubeState {
  dcInactive,
  dcBrowseMetaData,
  dcBrowseMemberData,
  dcBrowseAllData
};
```

File

Mxstore

Description

This is record Mxstore.TCubeState.

111.3.6 Mxstore.TErrorAction Enumeration

Pascal

```
TErrorAction = (
  eaFail,
  eaContinue
);
```

C++

```
enum TErrorAction {
  eaFail,
  eaContinue
};
```

File

Mxstore

Description

This is record Mxstore.TErrorAction.

111.3.7 Mxstore.TFormatType Enumeration

Pascal

```
TFormatType = (
  fxNone,
  fxFloat,
  fxCurrency,
  fxDateTime,
  fxTime,
  fxDate,
  fxString,
  fxInteger,
  fxBoolean
);
```

C++

```
enum TFormatType {
  fxNone,
  fxFloat,
  fxCurrency,
  fxDateTime,
  fxTime,
  fxDate,
  fxString,
  fxInteger,
  fxBoolean
};
```

File

Mxstore

Description

This is record Mxstore.TFormatType.

111.3.8 Mxstore.TIndexFlag Enumeration

Pascal

```
TIndexFlag = (
  idxNormal,
  idxSparsed,
  idxFiltered,
  idxSubTotals,
  idxDeleted
);
```

C++

```
enum TIndexFlag {
```

```
    idxNormal,
    idxSparsed,
    idxFiltered,
    idxSubTotals,
    idxDeleted
  };
```

File

Mxstore

Description

This is record Mxstore.TIndexFlag.

111.3.9 Mxstore.TIndexInfoRec Record

Pascal

```
  TIndexInfoRec = record
    AggOffset: Integer;
    SparseCnt: Integer;
    SubTotalCnt: Integer;
    Flags: TIndexFlags;
  end;
```

C++

```
  struct TIndexInfoRec {
    int AggOffset;
    int SparseCnt;
    int SubTotalCnt;
    TIndexFlags Flags;
  };
```

File

Mxstore

Description

This is record Mxstore.TIndexInfoRec.

111.3.10 Mxstore.TLookupStateFlags Enumeration

Pascal

```
  TLookupStateFlags = (
    lsSparsing,
    lsCursor,
    lsShowProgress
  );
```

C++

```
  enum TLookupStateFlags {
    lsSparsing,
    lsCursor,
    lsShowProgress
  };
```

File

Mxstore

Description

This is record Mxstore.TLookupStateFlags.

111.4 Types

The following table lists types in this documentation.

Types

Name	Description
PIndexInfoRec (see page 17843)	This is type Mxstore.PIndexInfoRec.
TAggProc (see page 17843)	This is type Mxstore.TAggProc.
TCacheState (see page 17844)	This is type Mxstore.TCacheState.
TCalcTotals (see page 17844)	This is type Mxstore.TCalcTotals.
TCapacityErrorEvent (see page 17844)	This is type Mxstore.TCapacityErrorEvent.
TCubeDimClass (see page 17845)	This is type Mxstore.TCubeDimClass.
TCubeDimTransformEvent (see page 17845)	This is type Mxstore.TCubeDimTransformEvent.
TCubeNotifyEvent (see page 17845)	This is type Mxstore.TCubeNotifyEvent.
TCubeRefreshEvent (see page 17846)	This is type Mxstore.TCubeRefreshEvent.
TDerivedAggProc (see page 17846)	This is type Mxstore.TDerivedAggProc.
TDimFlagSet (see page 17846)	This is type Mxstore.TDimFlagSet.
TFieldArgs (see page 17847)	This is type Mxstore.TFieldArgs.
TIndexFlags (see page 17847)	This is type Mxstore.TIndexFlags.
TLookupState (see page 17847)	This is type Mxstore.TLookupState.
TSumMethod (see page 17848)	This is type Mxstore.TSumMethod.

111.4.1 Mxstore.PIndexInfoRec Type

Pascal

```
PIndexInfoRec = ^TIndexInfoRec;
```

C++

```
^TIndexInfoRec PIndexInfoRec;
```

File

Mxstore

Description

This is type Mxstore.PIndexInfoRec.

111.4.2 Mxstore.TAggProc Type

Pascal

```
TAggProc = function (eCnt, Range: Integer; Summary: TSummary; SumIndex: TSmallIntArray; var
vNew: Variant): Boolean;
```

C++

```
(eCnt, Range: Integer; Summary: TSummary; SumIndex: TSmallIntArray; var vNew: Variant):
Boolean ( TAggProc)();
```

File

Mxstore

Description

This is type Mxstore.TAggProc.

111.4.3 Mxstore.TCacheState Type

Pascal

```
TCacheState = set of TCacheStateFlags;
```

C++

```
TCacheStateFlags TCacheState;
```

File

Mxstore

Description

This is type Mxstore.TCacheState.

111.4.4 Mxstore.TCalcTotals Type

Pascal

```
TCalcTotals = set of TCalcTotalsFlags;
```

C++

```
TCalcTotalsFlags TCalcTotals;
```

File

Mxstore

Description

This is type Mxstore.TCalcTotals.

111.4.5 Mxstore.TCapacityErrorEvent Type

Pascal

```
TCapacityErrorEvent = procedure (var EAction: TErrorAction) of object;
```

C++

```
(var EAction: TErrorAction) ( TCapacityErrorEvent)();
```

File

Mxstore

Description

This is type Mxstore.TCapacityErrorEvent.

111.4.6 Mxstore.TCubeDimClass Type

Pascal

```
TCubeDimClass = class of TCubeDim;
```

C++

```
TCubeDim TCubeDimClass;
```

File

Mxstore

Description

This is type Mxstore.TCubeDimClass.

111.4.7 Mxstore.TCubeDimTransformEvent Type

Pascal

```
TCubeDimTransformEvent = procedure (var Value: Variant; Data: TCubeDim) of object;
```

C++

```
(var Value: Variant; Data: TCubeDim) ( TCubeDimTransformEvent)();
```

File

Mxstore

Description

This is type Mxstore.TCubeDimTransformEvent.

111.4.8 Mxstore.TCubeNotifyEvent Type

Pascal

```
TCubeNotifyEvent = procedure (DataCube: TCustomDataStore) of object;
```

C++

```
(DataCube: TCustomDataStore) ( TCubeNotifyEvent)();
```

File

Mxstore

Description

This is type Mxstore.TCubeNotifyEvent.

111.4.9 Mxstore.TCubeRefreshEvent Type

Pascal

```
TCubeRefreshEvent = procedure (DataCube: TCustomDataStore; DimMap: TCubeDims) of object;
```

C++

```
(DataCube: TCustomDataStore; DimMap: TCubeDims) ( TCubeRefreshEvent)();
```

File

Mxstore

Description

This is type Mxstore.TCubeRefreshEvent.

111.4.10 Mxstore.TDerivedAggProc Type

Pascal

```
TDerivedAggProc = function (Val1, Val2: Variant): Variant;
```

C++

```
(Val1, Val2: Variant): Variant ( TDerivedAggProc)();
```

File

Mxstore

Description

This is type Mxstore.TDerivedAggProc.

111.4.11 Mxstore.TDimFlagSet Type

Pascal

```
TDimFlagSet = set of TDimFlags;
```

C++

```
TDimFlags TDimFlagSet;
```

File

Mxstore

Description

This is type Mxstore.TDimFlagSet.

111.4.12 Mxstore.TFieldArgs Type

Pascal

```
TFieldArgs = array[0..255] of Byte;
```

C++

```
array[0..255] of Byte TFieldArgs;
```

File

Mxstore

Description

This is type Mxstore.TFieldArgs.

111.4.13 Mxstore.TIndexFlags Type

Pascal

```
TIndexFlags = set of TIndexFlag;
```

C++

```
TIndexFlag TIndexFlags;
```

File

Mxstore

Description

This is type Mxstore.TIndexFlags.

111.4.14 Mxstore.TLookupState Type

Pascal

```
TLookupState = set of TLookupStateFlags;
```

C++

```
TLookupStateFlags TLookupState;
```

File

Mxstore

Description

This is type Mxstore.TLookupState.

111.4.15 Mxstore.TSumMethod Type

Pascal

```
TSumMethod = function (SumIndex: TSmallIntArray; Summary: TSummary; var Value: Variant):
Boolean of object;
```

C++

```
(SumIndex: TSmallIntArray; Summary: TSummary; var Value: Variant): Boolean ( TSumMethod)();
```

File

Mxstore

Description

This is type Mxstore.TSumMethod.

111.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
LargeValueCount (see page 17849)	This is constant Mxstore.LargeValueCount.
MaxBinDimensions (see page 17849)	This is constant Mxstore.MaxBinDimensions.
NonSparseAgg (see page 17850)	This is constant Mxstore.NonSparseAgg.
SparseAgg (see page 17850)	This is constant Mxstore.SparseAgg.
SparseSum (see page 17850)	This is constant Mxstore.SparseSum.
SparseUnknown (see page 17851)	This is constant Mxstore.SparseUnknown.
SubTotal (see page 17851)	This is constant Mxstore.SubTotal.

111.5.1 Mxstore.LargeValueCount Constant

Pascal

```
LargeValueCount = 50;
```

C++

```
LargeValueCount = 50;
```

File

Mxstore

Description

This is constant Mxstore.LargeValueCount.

111.5.2 Mxstore.MaxBinDimensions Constant

Pascal

```
MaxBinDimensions = 16;
```

C++

```
MaxBinDimensions = 16;
```

File

Mxstore

Description

This is constant Mxstore.MaxBinDimensions.

111.5.3 Mxstore.NonSparseAgg Constant

Pascal

```
NonSparseAgg = -4;
```

C++

```
NonSparseAgg = -4;
```

File

Mxstore

Description

This is constant Mxstore.NonSparseAgg.

111.5.4 Mxstore.SparseAgg Constant

Pascal

```
SparseAgg = -2;
```

C++

```
SparseAgg = -2;
```

File

Mxstore

Description

This is constant Mxstore.SparseAgg.

111.5.5 Mxstore.SparseSum Constant

Pascal

```
SparseSum = -1;
```

C++

```
SparseSum = -1;
```

File

Mxstore

Description

This is constant Mxstore.SparseSum.

111.5.6 Mxstore.SparseUnknown Constant

Pascal

```
SparseUnknown = -3;
```

C++

```
SparseUnknown = -3;
```

File

Mxstore

Description

This is constant Mxstore.SparseUnknown.

111.5.7 Mxstore.SubTotal Constant

Pascal

```
SubTotal = -1;
```

C++

```
SubTotal = -1;
```

File

Mxstore

Description

This is constant Mxstore.SubTotal.

112 MXTABLES Namespace

This is namespace MXTABLES.

Topics

Name	Description
Classes (see page 17854)	The following table lists classes in this documentation.
Functions (see page 17862)	The following table lists functions in this documentation.
Types (see page 17864)	The following table lists types in this documentation.

Classes

	Name	Description
	TDecisionQuery (see page 17854)	TDecisionQuery is a specialized query that prepares crosstabulated data for a decision cube.
	TQueryDim (see page 17855)	This is class MXTABLES.TQueryDim.
	TQueryDims (see page 17857)	This is class MXTABLES.TQueryDims.

Functions

	Name	Description
	AddDimensionItem (see page 17862)	This is function MXTABLES.AddDimensionItem.
	BuildDimensionMap (see page 17862)	This is function MXTABLES.BuildDimensionMap.
	BuildMasterDimensionMap (see page 17863)	This is function MXTABLES.BuildMasterDimensionMap.
	RemoveDimensionItem (see page 17863)	This is function MXTABLES.RemoveDimensionItem.

Types

Name	Description
TQueryDimClass (see page 17864)	This is type MXTABLES.TQueryDimClass.

112.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TDecisionQuery (see page 17854)	TDecisionQuery is a specialized query that prepares crosstabulated data for a decision cube.
	TQueryDim (see page 17855)	This is class MXTABLES.TQueryDim.
	TQueryDims (see page 17857)	This is class MXTABLES.TQueryDims.

112.1.1 TDecisionQuery Class

TDecisionQuery is a specialized query that prepares crosstabulated data for a decision cube.

Class Hierarchy

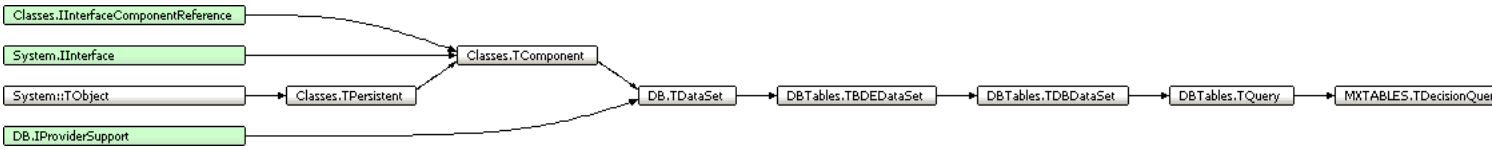

Pascal

```
TDecisionQuery = class(TQuery);
```

C++

```
class TDecisionQuery : public TQuery;
```

File

MXTABLES

Description

Use a TDecisionQuery as the dataset of a decision cube to ensure that the data is properly prepared. TDecisionQuery allows the dimensions and summaries to be defined directly on the dataset analyzed in a decision control. It then prepares an SQL query to provide the information from that dataset in a format that can be used by a decision cube.

Decision cubes can use other types of datasets, but the information must be structured in the same way as the output of a TDecisionQuery object.

Note: In the Decision Query editor that is displayed from the Decision query component, the query is initially handled in ANSI-92 SQL syntax, then translated (if necessary) into the dialect used by the data source of the query. Use only ANSI standard SQL in the Decision Query editor. Use the dialect of the data source in the SQL property of the Decision Query.

See Also

TDecisionCube (see page 17442)

Overview of Decision Support Components

Using Decision Graphs

Using Datasets with Decision Support Components

Creating Decision Datasets with the Decision Query Editor

112.1.2 TQueryDim Class

Class Hierarchy

System::TObject → Classes.TPersistent → Classes.TCollectionItem → MXCOMMON.TDimensionItem → MXTABLES.TQueryDim

Pascal

```
TQueryDim = class(TDimensionItem);
```

C++

```
class TQueryDim : public TDimensionItem;
```

File

MXTABLES

Description

This is class MXTABLES.TQueryDim.

112.1.2.1 TQueryDim Members

The following tables list the members exposed by TQueryDim.

Protected Methods

	Name	Description
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
V	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
V	InitializeRange (see page 17409)	This is InitializeRange, a member of class TDimensionItem.
V	NotifyCollection (see page 17410)	This is NotifyCollection, a member of class TDimensionItem.
V	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
V	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
V	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Protected Properties

	Name	Description
R	Owner (see page 17414)	This is Owner, a member of class TDimensionItem.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	Assign (see page 17408)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2757)	Creates and initializes a TCollectionItem (see page 2755) instance.
V	Create (see page 17408)	Creates and initializes a TCollectionItem instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDimension (see page 17409)	Indicates whether the dimension item represents a dimension.
	IsSummary (see page 17410)	Indicates whether the dimension item represents a summary.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Active (see page 17857)	This is Active, a member of class TQueryDim.
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	ActiveFlag (see page 17411)	Indicates whether the dimension item is stored in the cache of a crosstabulated data store.
	BaseName (see page 17411)	Specifies the field name of the database field that is represented by or summarized in the dimension item.
	DerivedFrom (see page 17412)	Specifies the index of a dimension item from which a derived value is calculated.
	DimensionType (see page 17412)	Indicates the use of the dimension item.
	Fieldname (see page 17413)	This is Fieldname, a member of class TDimensionItem.
	FieldType (see page 17413)	Indicates the data type of the field that the dimension item represents.

	Format (see page 17414)	Indicates how field values are formatted on dimensions that represent numeric or temporal data.
	Name (see page 17414)	Denotes the name of a dimension or summary.

112.1.2.2 TQueryDim Properties

The properties of the TQueryDim class are listed here.

Public Properties

	Name	Description
	Active (see page 17857)	This is Active, a member of class TQueryDim.

112.1.2.2.1 TQueryDim.Active Property

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

This is Active, a member of class TQueryDim.

112.1.3 TQueryDims Class

Class Hierarchy

System::TObject → Classes.TPersistent → Classes.TCollection → MXCOMMON.TDimensionItems → MXTABLES.TQueryDims

Pascal

```
TQueryDims = class(TDimensionItems);
```

C++

```
class TQueryDims : public TDimensionItems;
```

File

MXTABLES

Description

This is class MXTABLES.TQueryDims.

112.1.3.1 TQueryDims Members

The following tables list the members exposed by TQueryDims.

Protected Methods

	Name	Description
	Added (see page 2736)	Responds when items are added to the collection.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.

	Create (see page 17859)	This is Create, a member of class TQueryDims.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetOwner (see page 17420)	Returns the owner of an object.
	GetQueryDim (see page 17860)	This is GetQueryDim, a member of class TQueryDims.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	NotifyOwner (see page 17420)	This is NotifyOwner, a member of class TDimensionItems.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
	SetQueryDim (see page 17860)	This is SetQueryDim, a member of class TQueryDims.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 17860)	Adds a dimension item to the end of the Items array.
	AddDerivedField (see page 17418)	Adds a derived average summary to the end of the Items (see page 17422) array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AlreadyExists (see page 17419)	Indicates whether a dimension item with specified properties already exists in the Items (see page 17422) property array.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	Assign (see page 17419)	Copies the list of dimension items from another TDimensionItems (see page 17415) object.
	AverageFieldCheck (see page 17419)	Determines whether the necessary summary dimension items over a base dimension exist so that a dimension item of type dimAverage can be computed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.

V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ActiveDimensionCount (see page 17421)	Indicates the number of dimension items that represent active dimensions in the crosstabulated data store.
R	ActiveSummaryCount (see page 17421)	Indicates the number of dimension items that represent active summaries in the crosstabulated data store.
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	DimensionCount (see page 17422)	Indicates the number of dimension items in the Items (see page 17422) array with DimensionType set to dimDimension.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 17861)	Lists all dimension items maintained by the TDimensionItems object.
R	SummaryCount (see page 17423)	Indicates the number of dimension items in the Items (see page 17422) array with DimensionType set to a value other than dimDimension.

112.1.3.2 TQueryDims.Create Constructor

Pascal

```
constructor Create(Owner: TPersistent; ItemClass: TQueryDimClass);
```

C++

```
__fastcall TQueryDims(TPersistent * Owner, TQueryDimClass ItemClass);
```

Description

This is Create, a member of class TQueryDims.

112.1.3.3 TQueryDims Methods

The methods of the TQueryDims class are listed here.

Protected Methods

	Name	Description
	GetQueryDim (see page 17860)	This is GetQueryDim, a member of class TQueryDims.
	SetQueryDim (see page 17860)	This is SetQueryDim, a member of class TQueryDims.

Public Methods

	Name	Description
	Add (see page 17860)	Adds a dimension item to the end of the Items array.

112.1.3.3.1 TQueryDims.Add Method

Adds a dimension item to the end of the Items array.

Pascal

```
function Add: TQueryDim;
```

C++

```
__fastcall TQueryDim Add();
```

Description

Do not call the Add method to add dimension items to a TDimensionItems object. The dimension items are added automatically to reflect the fields in the data set of the associated crosstabulated data store.

Add adds a new dimension item to the collection and returns the new dimension item.

See Also

Items (see page 17422)

DataSet (see page 17789)

112.1.3.3.2 TQueryDims.GetQueryDim Method

Pascal

```
function GetQueryDim(Index: Integer): TQueryDim;
```

C++

```
__fastcall TQueryDim GetQueryDim(int Index);
```

Description

This is GetQueryDim, a member of class TQueryDims.

112.1.3.3.3 TQueryDims.SetQueryDim Method

Pascal

```
procedure SetQueryDim(Index: Integer; Value: TQueryDim);
```

C++

```
__fastcall SetQueryDim(int Index, TQueryDim Value);
```

Description

This is SetQueryDim, a member of class TQueryDims.

112.1.3.4 TQueryDims Properties

The properties of the TQueryDims class are listed here.

Public Properties

	Name	Description
	Items (see page 17861)	Lists all dimension items maintained by the TDimensionItems object.

112.1.3.4.1 TQueryDims.Items Property

Lists all dimension items maintained by the TDimensionItems object.

Pascal

```
property Items [Index: Integer]: TQueryDim;
```

C++

```
__property TQueryDim Items[int Index];
```

Description

Use Items to access individual dimension items maintained by the TDimensionItems object. The first dimension item has an Index of 0, the second dimension item has an Index of 1, and so on. For example, the following C++ code accesses an item to indicate that it can be swapped out of memory:

```
DecisionCube1->DimensionMap->Items[0]->ActiveFlag = diAsNeeded;
```

Each dimension item in the Items array represents a field from the data set of a crosstabulated data store. Each dimension item represents either a dimension (a field over which data is summarized), or a summary (a value summarized over the dimensions).

In Delphi Code, Items is the default property of TDimensionItems. This means that the name of the Items property can be eliminated from code that uses the Items property. Thus, the line

```
DecisionCube1.DimensionMap.Items[0].ActiveFlag := diAsNeeded;
```

could be written

```
DecisionCube1.DimensionMap[0].ActiveFlag := diAsNeeded;
```

Note: Do not change the Items property. The set of items in the Items array is set automatically to represent the fields from the data set. The properties of individual dimension items within the Items array may be changed to specify whether the dimension items should be cached by the data store, to change the name used when displaying the dimension items, and so on.

See Also

DataSet (see page 17789)

DimensionMap (see page 17790)

operator_[]

112.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AddDimensionItem (see page 17862)	This is function MXTABLES.AddDimensionItem.
	BuildDimensionMap (see page 17862)	This is function MXTABLES.BuildDimensionMap.
	BuildMasterDimensionMap (see page 17863)	This is function MXTABLES.BuildMasterDimensionMap.
	RemoveDimensionItem (see page 17863)	This is function MXTABLES.RemoveDimensionItem.

112.2.1 MXTABLES.AddDimensionItem Function

Pascal

```
function AddDimensionItem(myQuery: TXtabQuery; Map: TQueryDims; index: integer; Name: string):
TQueryError;
```

C++

```
TQueryError AddDimensionItem(TXtabQuery myQuery, TQueryDims Map, int index, AnsiString Name);
```

File

MXTABLES

Description

This is function MXTABLES.AddDimensionItem.

112.2.2 MXTABLES.BuildDimensionMap Function

Pascal

```
function BuildDimensionMap(myQuery: TXtabQuery; Map: TQueryDims; SQLString: string):
TQueryError;
```

C++

```
TQueryError BuildDimensionMap(TXtabQuery myQuery, TQueryDims Map, AnsiString SQLString);
```

File

MXTABLES

Description

This is function MXTABLES.BuildDimensionMap.

112.2.3 MXTABLES.BuildMasterDimensionMap Function

Pascal

```
function BuildMasterDimensionMap(myQuery: TXtabQuery; Map: TQueryDims): TQueryError;
```

C++

```
TQueryError BuildMasterDimensionMap(TXtabQuery myQuery, TQueryDims Map);
```

File

MXTABLES

Description

This is function MXTABLES.BuildMasterDimensionMap.

112.2.4 MXTABLES.RemoveDimensionItem Function

Pascal

```
function RemoveDimensionItem(myQuery: TXtabQuery; Map: TQueryDims; index: integer):
TQueryError;
```

C++

```
TQueryError RemoveDimensionItem(TXtabQuery myQuery, TQueryDims Map, int index);
```

File

MXTABLES

Description

This is function MXTABLES.RemoveDimensionItem.

112.3 Types

The following table lists types in this documentation.

Types

Name	Description
TQueryDimClass (see page 17864)	This is type MXTABLES.TQueryDimClass.

112.3.1 MXTABLES.TQueryDimClass Type

Pascal

```
TQueryDimClass = class of TQueryDim;
```

C++

```
TQueryDim TQueryDimClass;
```

File

MXTABLES

Description

This is type MXTABLES.TQueryDimClass.

113 ObjBrkr Namespace

This is namespace ObjBrkr.

Topics

Name	Description
Classes (see page 17866)	The following table lists classes in this documentation.

Classes

	Name	Description
	EBrokerException (see page 17866)	EBrokerException is the exception raised by simple object brokers when they can't locate an available server.
	TServerCollection (see page 17866)	TServerCollection is a collection of TServerItem (see page 17871) objects.
	TServerItem (see page 17871)	TServerItem describes an application server for a simple object broker.
	TSimpleObjectBroker (see page 17877)	TSimpleObjectBroker maintains a list of available application servers for a connection component.

113.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EBrokerException (see page 17866)	EBrokerException is the exception raised by simple object brokers when they can't locate an available server.
	TServerCollection (see page 17866)	TServerCollection is a collection of TServerItem (see page 17871) objects.
	TServerItem (see page 17871)	TServerItem describes an application server for a simple object broker.
	TSimpleObjectBroker (see page 17877)	TSimpleObjectBroker maintains a list of available application servers for a connection component.

113.1.1 EBrokerException Class

EBrokerException is the exception raised by simple object brokers when they can't locate an available server.

Class Hierarchy

System::TObject → SysUtils.Exception → ObjBrkr.EBrokerException

Pascal

```
EBrokerException = class(Exception);
```

C++

```
class EBrokerException : public Exception;
```

File

ObjBrkr

Description

EBrokerException is raised when a simple object broker can't locate an available server.

See Also

TSimpleObjectBroker (see page 17877)

TServerCollection (see page 17866)

113.1.2 TServerCollection Class

TServerCollection is a collection of TServerItem (see page 17871) objects.

Class Hierarchy

System::TObject → Classes.TPersistent → Classes.TCollection → Classes.TOwnedCollection → ObjBrkr.TServerCollection

Pascal

```
TServerCollection = class(TOwnedCollection);
```

C++

```
class TServerCollection : public TOwnedCollection;
```

File

ObjBrkr

Description

TSimpleObjectBroker (see page 17877) uses TServerCollection to represent the set of available application servers to which a client application can connect. Use the properties and methods of TServerCollection to manage the set of servers. For example, you can add items as new server systems come on line, disable server items when the corresponding system is temporarily out of commission, and so on.

See Also

TServerItem (see page 17871)

TSimpleObjectBroker (see page 17877)

113.1.2.1 TServerCollection Members

The following tables list the members exposed by TServerCollection.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 2915)	Returns the owner of an object.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 2736)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 17869)	Creates and initializes a TServerCollection (see page 17866) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
	Destroy (see page 2735)	Destroys the collection and each item in it.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	FindServer (see page 17869)	Returns the server from the Items (see page 17871) property array that has a specified computer name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetBalancedName (see page 17870)	Returns the ComputerName property of a randomly chosen server from the Items (see page 17871) property array.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNextName (see page 17870)	This is GetNextName, a member of class TServerCollection.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.

	Items (see page 17871)	Lists the TServerItem (see page 17871) objects in the collection.

113.1.2.2 TServerCollection.Create Constructor

Creates and initializes a TServerCollection (see page 17866) object.

Pascal

```
constructor Create(AOwner: TComponent);
```

C++

```
__fastcall TServerCollection(TComponent * AOwner);
```

Description

Applications should not need to instantiate TServerCollection (see page 17866). The simple object broker that uses TServerCollection (see page 17866) to manage its list of available servers creates this object from its own constructor.

See Also

Servers (see page 17885)

113.1.2.3 TServerCollection Methods

The methods of the TServerCollection class are listed here.

Public Methods

	Name	Description
	FindServer (see page 17869)	Returns the server from the Items (see page 17871) property array that has a specified computer name.
	GetBalancedName (see page 17870)	Returns the ComputerName property of a randomly chosen server from the Items (see page 17871) property array.
	GetNextName (see page 17870)	This is GetNextName, a member of class TServerCollection.

113.1.2.3.1 TServerCollection.FindServer Method

Returns the server from the Items (see page 17871) property array that has a specified computer name.

Pascal

```
function FindServer(const ComputerName: string): TServerItem;
```

C++

```
__fastcall TServerItem FindServer(const AnsiString ComputerName);
```

Description

Call FindServer to locate a server in the items property array given its ComputerName property. The ComputerName parameter specifies this computer name. Usually, this value is obtained by a previous call to GetBalancedName (see page 17870) or GetNextName (see page 17870).

Note: FindServer returns the first item it finds with the given ComputerName property. Unlike the GetBalancedName (see page 17870) and GetNextName (see page 17870) methods, it does not ignore servers that have failed or been disabled (HasFailed is true, or Enabled is false).

See Also

GetBalancedName (see page 17870)

getnextname (see page 17870)

ComputerName (see page 17874)

Items (see page 17871)

Enabled (see page 17875)

HasFailed (see page 17876)

113.1.2.3.2 TServerCollection.GetBalancedName Method

Returns the ComputerName property of a randomly chosen server from the Items (see page 17871) property array.

Pascal

```
function GetBalancedName: string;
```

C++

```
__fastcall AnsiString GetBalancedName();
```

Description

Call GetBalancedName to get the name of an available server. Servers in the Items (see page 17871) property array are considered available when no connection attempts have failed (HasFailed is false) and their Enabled property has not been set to false. GetBalancedName selects an available TServerItem (see page 17871) at random from the Items (see page 17871) property array and returns its ComputerName property.

See Also

Items (see page 17871)

LoadBalanced (see page 17884)

ComputerName (see page 17874)

HasFailed (see page 17876)

Enabled (see page 17875)

getnextname (see page 17870)

113.1.2.3.3 TServerCollection.GetNextName Method

Pascal

```
function GetNextName: string;
```

C++

```
__fastcall AnsiString GetNextName();
```

Description

This is GetNextName, a member of class TServerCollection.

113.1.2.4 TServerCollection Properties

The properties of the TServerCollection class are listed here.

Public Properties

	Name	Description
	Items (see page 17871)	Lists the TServerItem (see page 17871) objects in the collection.

113.1.2.4.1 TServerCollection.Items Property

Lists the TServerItem (see page 17871) objects in the collection.

Pascal

```
property Items [Index: Integer]: TServerItem;
```

C++

```
__property TServerItem Items[int Index];
```

Description

Use Items to access individual TServerItem (see page 17871) objects maintained by TServerCollection (see page 17866).

Note: In Delphi, Items is the default property of TServerCollection (see page 17866), so the name of the Items property can be left off when using this property. Thus the line

```
TSimpleObjectBroker1.Servers.Items[I].Enabled := False;
```

can be written

```
TSimpleObjectBroker1.Servers[I].Enabled := False;
```

Note: In C++, Items can be accessed using the [] operator, to achieve an affect similar to the default property in Delphi.

See Also

Enabled (see page 17875)

GetBalancedName (see page 17870)

getnextname (see page 17870)

FindServer (see page 17869)

operator_[]

113.1.3 TServerItem Class

TServerItem describes an application server for a simple object broker.

Class Hierarchy

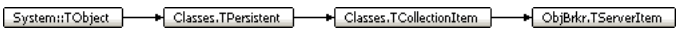

Pascal

```
TServerItem = class(TCollectionItem);
```

C++

```
class TServerItem : public TCollectionItem;
```

File

ObjBrkr

Description

Use TServerItem to describe an application server to which client applications can connect. TSimpleObjectBroker (see page 17877) maintains a list of these descriptions. Then, when a connection component needs to open a connection, it can use the information stored in a server item to identify an available application server.

See Also

TSimpleObjectBroker (see page 17877)

TRemoteDataModule (see page 6585)

TDispatchConnection (see page 16017)

113.1.3.1 TServerItem Members

The following tables list the members exposed by TServerItem.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 17874)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 17873)	Creates and initializes a TServerItem (see page 17871) instance.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
	HasFailed (see page 17876)	Indicates whether a connection attempt to the associated server machine has failed.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	ComputerName (see page 17874)	Identifies the machine where the application server resides.
	Enabled (see page 17875)	Enables or disables the server item.
	Port (see page 17876)	Specifies the port number for the associated server application.

113.1.3.2 TServerItem.Create Constructor

Creates and initializes a TServerItem (see page 17871) instance.

Pascal

```
constructor Create(AOwner: TCollection); override;
```

C++

```
virtual __fastcall TServerItem(TCollection AOwner);
```

Description

Do not call Create directly to add a new server item to a server collection. Instead, use the Add or Insert method of the TServerCollection (see page 17866) object.

The AOwner parameter specifies the server collection that manages the newly created server item.

After calling the inherited constructor, Create initializes the Enabled (see page 17875) property to true, the HasFailed (see page 17876) property to false, and the Port (see page 17876) property to 211.

See Also

Add (see page 2736)

Insert (see page 2746)

Port (see page 17876)

HasFailed (see page 17876)

Enabled (see page 17875)

113.1.3.3 TServerItem Methods

The methods of the TServerItem class are listed here.

Protected Methods

	Name	Description
	GetDisplayName (see page 17874)	Returns the name of the collection item as it appears in the collection editor.

113.1.3.3.1 TServerItem.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName (see page 2761)

GetNamePath (see page 2759)

113.1.3.4 TServerItem Properties

The properties of the TServerItem class are listed here.

Public Properties

	Name	Description
	HasFailed (see page 17876)	Indicates whether a connection attempt to the associated server machine has failed.

Published Properties

	Name	Description
	ComputerName (see page 17874)	Identifies the machine where the application server resides.
	Enabled (see page 17875)	Enables or disables the server item.
	Port (see page 17876)	Specifies the port number for the associated server application.

113.1.3.4.1 TServerItem.ComputerName Property

Identifies the machine where the application server resides.

Pascal

```
property ComputerName: string;
```

C++

```
__property AnsiString ComputerName;
```

Description

Set ComputerName to a string that identifies the machine where the application server resides. The value of this string depends on the type of connection component that requests this value from its object broker.

If the connection component is a TDCOMConnection object, ComputerName is the Windows computer name. (This value can be found in the Control panel under network settings.)

If the connection component is a TSocketConnection object, ComputerName is either the host name or IP address of the system running the application server. Socket connection components parse this string to determine which it is.

If the connection component is a TWebConnection or TSoapConnection object, ComputerName is the URL of the dispatcher (httpsrvr.dll or soapsrvr.dll (running on the system with the application server.

See Also

GetBalancedName (see page 17870)

getnextname (see page 17870)

Port (see page 17876)

computername (see page 16014)

Address (see page 18935)

Host (see page 18936)

ObjectBroker (see page 16028)

113.1.3.4.2 TServerItem.Enabled Property

Enables or disables the server item.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Set Enabled to indicate whether the server should made available in the simple object broker. When Enabled is true, the ComputerName (see page 17874) property can be supplied to connection components by the object broker's GetComputerForGUID or GetComputerForProgID methods. When Enabled is false, the server item is not returned.

Note: Even when the server item is disabled, it is still returned by TServerCollection (see page 17866)'s FindServer method.

See Also

GetComputerForGUID (see page 17881)

GetComputerForProgID (see page 17882)

HasFailed (see page 17876)

FindServer (see page 17869)

113.1.3.4.3 TServerItem.HasFailed Property

Indicates whether a connection attempt to the associated server machine has failed.

Pascal

```
property HasFailed: Boolean;
```

C++

```
__property Boolean HasFailed;
```

Description

When connection components can't successfully open a connection to an application server, they inform the object broker by calling SetConnectStatus. The simple object broker, in turn, records this fact by setting the HasFailed property to true.

When HasFailed is false, the ComputerName (see page 17874) property can be supplied to connection components by the object broker's GetComputerForGUID or GetComputerForProgID methods. When HasFailed is true, the server item is not returned.

Once a server has failed it is presumed to be down until the application explicitly resets the HasFailed property to false.

Do not use HasFailed to disable server items. Instead, use the Enabled (see page 17875) property to disable servers for any reason other than a failed connection.

Note: Even when HasFailed is true, the server item is still returned by TServerCollection (see page 17866)'s FindServer method.

See Also

SetConnectStatus (see page 17884)

Enabled (see page 17875)

GetComputerForGUID (see page 17881)

GetComputerForProgID (see page 17882)

FindServer (see page 17869)

113.1.3.4.4 TServerItem.Port Property

Specifies the port number for the associated server application.

Pascal

```
property Port: Integer;
```

C++

```
__property int Port;
```

Description

Read Port to obtain a port number when using the server item to supply information for connecting using TCP/IP. This value is returned by the GetPortForComputer method of the simple object broker.

By default, the value of Port is 211, which is the default port number for the socket dispatcher application ScktSrvr.exe. If the server listens on a different port, change this property to indicate the listening port.

Note: Server items that supply a Port number should specify the server system's IP address or host name as the value of ComputerName (see page 17874).

See Also

ComputerName (see page 17874)

113.1.4 TSimpleObjectBroker Class

TSimpleObjectBroker maintains a list of available application servers for a connection component.

Class Hierarchy

Pascal

```
TSimpleObjectBroker = class(TCustomObjectBroker);
```

C++

```
class TSimpleObjectBroker : public TCustomObjectBroker;
```

File

ObjBrkr

Description

TSimpleObjectBroker maintains a list of servers, all of which run the application server for a multi-tiered application. When a connection component requests a server, the simple object broker passes the name of one of the available servers in the list. This process allows client applications to locate server systems dynamically at runtime. If there are redundant server systems, this allows client applications to continue functioning when a server goes down.

TSimpleObjectBroker can randomize the selection of servers from among the currently available servers in its list to provide a simple system of load balancing.

See Also

TDispatchConnection (see page 16017)

TServerCollection (see page 17866)

113.1.4.1 TSimpleObjectBroker Members

The following tables list the members exposed by TSimpleObjectBroker.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.

	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

	Create (see page 17880)	Creates and initializes a simple object broker.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 17880)	Destroys an instance of TSimpleObjectBroker (see page 17877).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComputerForGUID (see page 17881)	Returns a string that identifies an available server machine.
	GetComputerForProgID (see page 17882)	Returns a string that identifies an available server machine.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetPortForComputer (see page 17882)	Returns a port number for connecting to a specified remote server.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LoadFromStream (see page 17883)	Sets the value of the Servers (see page 17885) property from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToStream (see page 17883)	Saves the value of the Servers (see page 17885) property to a stream.
	SetConnectStatus (see page 17884)	Records whether a connection was established to a specified server.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ServerData (see page 17885)	Encapsulates the list of servers as an OleVariant.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	LoadBalanced (see page 17884)	Specifies how the simple object broker picks a server from its list.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Servers (see page 17885)	Maintains the list of servers that can be provided to connection components.
	Tag (see page 2808)	Stores an integer value as part of a component.

113.1.4.2 TSimpleObjectBroker.Create Constructor

Creates and initializes a simple object broker.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSimpleObjectBroker(TComponent * AOwner);
```

Description

Call Create to instantiate a simple object broker at runtime.

After calling the inherited constructor, Create initializes the LoadBalanced (see page 17884) property to false and creates the TServerCollection (see page 17866) that implements the Servers (see page 17885) property.

See Also

Create (see page 2766)

Servers (see page 17885)

LoadBalanced (see page 17884)

113.1.4.3 TSimpleObjectBroker.Destroy Destructor

Destroys an instance of TSimpleObjectBroker (see page 17877).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSimpleObjectBroker();
```

Description

Do not call Destroy directly. Instead, call Free, which verifies that the simple object broker is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

113.1.4.4 TSimpleObjectBroker Methods

The methods of the TSimpleObjectBroker class are listed here.

Public Methods

	Name	Description
	GetComputerForGUID (see page 17881)	Returns a string that identifies an available server machine.
	GetComputerForProgID (see page 17882)	Returns a string that identifies an available server machine.
	GetPortForComputer (see page 17882)	Returns a port number for connecting to a specified remote server.
	LoadFromStream (see page 17883)	Sets the value of the Servers (see page 17885) property from a stream.
	SaveToStream (see page 17883)	Saves the value of the Servers (see page 17885) property to a stream.
	SetConnectStatus (see page 17884)	Records whether a connection was established to a specified server.

113.1.4.4.1 TSimpleObjectBroker.GetComputerForGUID Method

Returns a string that identifies an available server machine.

Pascal

```
function GetComputerForGUID(GUID: TGUID): string; override;
```

C++

```
virtual __fastcall AnsiString GetComputerForGUID(TGUID GUID);
```

Description

Call GetComputerForGUID to obtain the name of an available server machine. The GUID parameter (which specifies the global unique identifier of the desired server interface) is ignored.

GetComputerForGUID returns the ComputerName property of an available server from the Servers (see page 17885) property. If the LoadBalanced (see page 17884) property is true, this server is chosen at random from available servers. If the LoadBalanced (see page 17884) property is false, this server is the first available server listed in the Servers (see page 17885) property. If no servers are available, GetComputerForGUID raises an EBrokerException (see page 17866) exception.

A server is considered available if its Enabled property is true and its HasFailed property is false. The broker automatically sets HasFailed to false when it is notified of a connection failure for that server.

See Also

GetComputerForProgID (see page 17882)

ComputerName (see page 17874)

LoadBalanced (see page 17884)

EBrokerException (see page 17866)

Enabled (see page 17875)

HasFailed (see page 17876)

SetConnectStatus (see page 17884)

113.1.4.4.2 TSimpleObjectBroker.GetComputerForProgID Method

Returns a string that identifies an available server machine.

Pascal

```
function GetComputerForProgID(const ProgID): string; override;
```

C++

```
virtual __fastcall AnsiString GetComputerForProgID(const  ProgID);
```

Description

Call GetComputerForProgID to obtain the name of an available server machine. The ProgID parameter (which specifies the desired server interface) is ignored.

TSimpleObjectBroker (see page 17877) returns the ComputerName property of an available server from the Servers (see page 17885) property. If the LoadBalanced (see page 17884) property is true, this server is chosen at random from available servers. If the LoadBalanced (see page 17884) property is false, this server is the first available server listed in the Servers (see page 17885) property. If no servers are available, GetComputerForGUID (see page 17881) raises an EBrokerException (see page 17866) exception.

A server is considered available if its Enabled property is true and its HasFailed property is false. The broker automatically sets HasFailed to false when it is notified of a connection failure for that server.

See Also

GetComputerForGUID (see page 17881)

ComputerName (see page 17874)

LoadBalanced (see page 17884)

EBrokerException (see page 17866)

Enabled (see page 17875)

HasFailed (see page 17876)

SetConnectStatus (see page 17884)

113.1.4.4.3 TSimpleObjectBroker.GetPortForComputer Method

Returns a port number for connecting to a specified remote server.

Pascal

```
function GetPortForComputer(const ComputerName: string): Integer; override;
```

C++

```
virtual __fastcall int GetPortForComputer(const AnsiString ComputerName);
```

Description

Socket connection components call GetPortForComputer to get a value for their Port property. The ComputerName parameter specifies a Host name or IP address that was returned by GetComputerForGUID (see page 17881) or GetComputerForProgID (see page 17882). This value is used to identify a server in the list maintained by the Servers (see page 17885) property.

There are no connection components that explicitly use a Port number except on the Windows product. GetPortForComputer is

included in TCustomObjectBroker to allow for the possibility of connection components that use an explicit Port number. The ComputerName parameter is a Host name or IP address that is returned by the GetComputerForGUID (see page 17881) or GetComputerForProgID (see page 17882) method. This value is used to identify a server in the list maintained by the Servers (see page 17885) property.

See Also

GetComputerForProgID (see page 17882)

GetComputerForGUID (see page 17881)

Port (see page 18938)

ComputerName (see page 17874)

Port (see page 17876)

113.1.4.4.4 TSimpleObjectBroker.LoadFromStream Method

Sets the value of the Servers (see page 17885) property from a stream.

Pascal

```
procedure LoadFromStream(Stream: TStream);
```

C++

```
__fastcall LoadFromStream(TStream Stream);
```

Description

Call LoadFromStream to load a list of available servers that has been previously saved using the SaveToStream (see page 17883) method.

See Also

SaveToStream (see page 17883)

ServerData (see page 17885)

113.1.4.4.5 TSimpleObjectBroker.SaveToStream Method

Saves the value of the Servers (see page 17885) property to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream);
```

C++

```
__fastcall SaveToStream(TStream Stream);
```

Description

Call SaveToStream to save the list of available servers. The servers can be restored later by calling the LoadFromStream (see page 17883) method.

See Also

LoadFromStream (see page 17883)

ServerData (see page 17885)

113.1.4.4.6 TSimpleObjectBroker.SetConnectStatus Method

Records whether a connection was established to a specified server.

Pascal

```
procedure SetConnectStatus(ComputerName: string; Success: Boolean); override;
```

C++

```
virtual __fastcall SetConnectStatus(AnsiString ComputerName, Boolean Success);
```

Description

After calling GetComputerForGUID (see page 17881) or GetComputerForProgID (see page 17882), connection components use the returned value to open a connection to the identified server. They then call SetConnectStatus to inform the broker of whether the connection was successfully opened.

SetConnectStatus sets the HasFailed property of the corresponding server to true if the connection was unsuccessful. It sets the HasFailed property to false if the connection was successful.

When the HasFailed property of a server is true, that server is ignored when the GetComputerForGUID (see page 17881) or GetComputerForProgID (see page 17882) method looks for available connections.

See Also

GetComputerForGUID (see page 17881)

GetComputerForProgID (see page 17882)

113.1.4.5 TSimpleObjectBroker Properties

The properties of the TSimpleObjectBroker class are listed here.

Public Properties

	Name	Description
	ServerData (see page 17885)	Encapsulates the list of servers as an OleVariant.

Published Properties

	Name	Description
	LoadBalanced (see page 17884)	Specifies how the simple object broker picks a server from its list.
	Servers (see page 17885)	Maintains the list of servers that can be provided to connection components.

113.1.4.5.1 TSimpleObjectBroker.LoadBalanced Property

Specifies how the simple object broker picks a server from its list.

Pascal

```
property LoadBalanced: Boolean;
```

C++

```
__property Boolean LoadBalanced;
```

Description

Set LoadBalanced to true to indicate that the simple object broker should supply computer names randomly from the available servers listed in the Servers (see page 17885) property. When LoadBalanced is false, GetComputerForGUID (see page 17881) and GetComputerForProgID (see page 17882) always return the first available server.

If the simple object broker is used by multiple connection components in the client application, it is preferable to set LoadBalanced to true in order to distribute the work among available servers.

When there is only a single connection component, there is no need to randomize the selection of servers, because only one connection will be used at a time. However, when deploying client applications, you may want to reorder the Servers (see page 17885) list in the simple object broker for each client, so that different clients give preference to different servers.

See Also

Servers (see page 17885)

GetComputerForGUID (see page 17881)

GetComputerForProgID (see page 17882)

113.1.4.5.2 TSimpleObjectBroker.ServerData Property

Encapsulates the list of servers as an OleVariant.

Pascal

```
property ServerData: OleVariant;
```

C++

```
__property OleVariant ServerData;
```

Description

Reading ServerData returns the value of the Servers (see page 17885) property stored in the internal component format. Writing ServerData sets the value of the Servers (see page 17885) property from an OleVariant that was saved previously.

Use the ServerData property to provide a mechanism for loading and saving the list of servers represented by the Servers (see page 17885) property. For example, An application can write the value of ServerData to an .ini file or the system registry. Later, it can read the value it saved as an OleVariant, and set ServerData to restore the Servers (see page 17885) property to the value that was saved.

Note: The implementation of ServerData uses the SaveToStream (see page 17883) method to encode the Servers (see page 17885) property as an OleVariant and the LoadFromStream (see page 17883) method to decode that OleVariant and restore the Servers (see page 17885) property.

See Also

Servers (see page 17885)

LoadFromStream (see page 17883)

SaveToStream (see page 17883)

113.1.4.5.3 TSimpleObjectBroker.Servers Property

Maintains the list of servers that can be provided to connection components.

Pascal

```
property Servers: TServerCollection;
```

C++

```
__property TServerCollection Servers;
```

Description

Use Servers to specify the list of servers that the simple object broker can return from the GetComputerForProgID (see page

17882) and GetComputerForGUID (see page 17881) methods.

Each item in the collection keeps track of whether the last connection attempt to that server was successful, and if not, the simple object broker ignores that server. In addition, individual servers in the collection can be disabled so that the simple object broker ignores them until they are re-enabled.

See Also

ServerData (see page 17885)

GetComputerForProgID (see page 17882)

GetComputerForGUID (see page 17881)

TServerItem (see page 17871)

114 OleAuto Namespace

This is namespace OleAuto.

Topics

Name	Description
Classes (see page 17889)	The following table lists classes in this documentation.
Functions (see page 17907)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 17913)	The following table lists structs, records, enums in this documentation.
Types (see page 17915)	The following table lists types in this documentation.
Variables (see page 17916)	The following table lists variables in this documentation.
Constants (see page 17917)	The following table lists constants in this documentation.

Classes

	Name	Description
	EOleError (see page 17889)	This is class OleAuto.EOleError.
	EOleException (see page 17889)	This is class OleAuto.EOleException.
	EOleSysError (see page 17892)	This is class OleAuto.EOleSysError.
	TAutoDispatch (see page 17894)	This is class OleAuto.TAutoDispatch.
	TAutomation (see page 17897)	This is class OleAuto.TAutomation.
	TAutoObject (see page 17901)	This is class OleAuto.TAutoObject.
	TRegistryClass (see page 17905)	This is class OleAuto.TRegistryClass.

Constants

Name	Description
MaxDispArgs (see page 17917)	This is constant OleAuto.MaxDispArgs.

Functions

	Name	Description
	ClassIDToProgID (see page 17907)	This is function OleAuto.ClassIDToProgID.
	ClassIDToString (see page 17907)	This is function OleAuto.ClassIDToString.
	CreateOleObject (see page 17908)	This is function OleAuto.CreateOleObject.
	DispInvoke (see page 17908)	This is function OleAuto.DispInvoke.
	DispInvokeError (see page 17908)	This is function OleAuto.DispInvokeError.
	DllCanUnloadNow (see page 17909)	This is function OleAuto.DllCanUnloadNow.
	DllGetClassObject (see page 17909)	This is function OleAuto.DllGetClassObject.
	DllRegisterServer (see page 17909)	This is function OleAuto.DllRegisterServer.

	DllUnregisterServer (see page 17910)	This is function OleAuto.DllUnregisterServer.
	GetActiveOleObject (see page 17910)	This is function OleAuto.GetActiveOleObject.
	OleCheck (see page 17910)	This is function OleAuto.OleCheck.
	OleError (see page 17911)	This is function OleAuto.OleError.
	ProgIDToClassID (see page 17911)	This is function OleAuto.ProgIDToClassID.
	StringToClassID (see page 17911)	This is function OleAuto.StringToClassID.
	VarFromInterface (see page 17912)	This is function OleAuto.VarFromInterface.
	VarToAutoObject (see page 17912)	This is function OleAuto.VarToAutoObject.
	VarToInterface (see page 17912)	This is function OleAuto.VarToInterface.

Structs, Records, Enums

	Name	Description
	TAutoClassInfo (see page 17913)	This is record OleAuto.TAutoClassInfo.
	TAutoClassInstancing (see page 17913)	This is record OleAuto.TAutoClassInstancing.
	TStartMode (see page 17914)	This is record OleAuto.TStartMode.

Types

Name	Description
TAutoClass (see page 17915)	This is type OleAuto.TAutoClass.
TLastReleaseEvent (see page 17915)	This is type OleAuto.TLastReleaseEvent.

Variables

Name	Description
Automation (see page 17916)	This is variable OleAuto.Automation.

114.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EOleError (see page 17889)	This is class OleAuto.EOleError.
	EOleException (see page 17889)	This is class OleAuto.EOleException.
	EOleSysError (see page 17892)	This is class OleAuto.EOleSysError.
	TAutoDispatch (see page 17894)	This is class OleAuto.TAutoDispatch.
	TAutomation (see page 17897)	This is class OleAuto.TAutomation.
	TAutoObject (see page 17901)	This is class OleAuto.TAutoObject.
	TRegistryClass (see page 17905)	This is class OleAuto.TRegistryClass.

114.1.1 EOleError Class

Class Hierarchy

Pascal

```
EOleError = class(Exception);
```

C++

```
class EOleError : public Exception;
```

File

OleAuto

Description

This is class OleAuto.EOleError.

114.1.2 EOleException Class

Class Hierarchy

Pascal

```
EOleException = class(EOleError);
```

C++

```
class EOleException : public EOleError;
```

File

OleAuto

Description

This is class OleAuto.EOleException.

114.1.2.1 EOleException Members

The following tables list the members exposed by EOleException.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 17891)	This is Create, a member of class EOleException.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ErrorCode (see page 17891)	This is ErrorCode, a member of class EOleException.

	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
R	HelpFile (see page 17891)	This is HelpFile, a member of class EOleException.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.
R	Source (see page 17892)	This is Source, a member of class EOleException.

114.1.2.2 EOleException.Create Constructor

Pascal

```
constructor Create(const ExcepInfo: TExcepInfo);
```

C++

```
__fastcall EOleException(const TExcepInfo ExcepInfo);
```

Description

This is Create, a member of class EOleException.

114.1.2.3 EOleException Properties

The properties of the EOleException class are listed here.

Public Properties

	Name	Description
R	ErrorCode (see page 17891)	This is ErrorCode, a member of class EOleException.
R	HelpFile (see page 17891)	This is HelpFile, a member of class EOleException.
R	Source (see page 17892)	This is Source, a member of class EOleException.

114.1.2.3.1 EOleException.ErrorCode Property

Pascal

```
property ErrorCode: Integer;
```

C++

```
__property int ErrorCode;
```

Description

This is ErrorCode, a member of class EOleException.

114.1.2.3.2 EOleException.HelpFile Property

Pascal

```
property HelpFile: string;
```

C++

```
__property AnsiString HelpFile;
```

Description

This is HelpFile, a member of class EOleException.

114.1.2.3.3 EOleException.Source Property

Pascal

```
property Source: string;
```

C++

```
__property AnsiString Source;
```

Description

This is Source, a member of class EOleException.

114.1.3 EOleSysError Class

Class Hierarchy

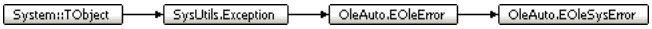

Pascal

```
EOleSysError = class(EOleError);
```

C++

```
class EOleSysError : public EOleError;
```

File

OleAuto

Description

This is class OleAuto.EOleSysError.

114.1.3.1 EOleSysError Members

The following tables list the members exposed by EOleSysError.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 17893)	This is Create, a member of class EOleSysError.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.

	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ErrorCode (see page 17894)	This is ErrorCode, a member of class EOleSysError.
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.

114.1.3.2 EOleSysError.Create Constructor

Pascal

```
constructor Create(ErrorCode: Integer);
```

C++

```
__fastcall EOleSysError(int ErrorCode);
```

Description

This is Create, a member of class EOleSysError.

114.1.3.3 EOleSysError Properties

The properties of the EOleSysError class are listed here.

Public Properties

	Name	Description
R	ErrorCode (see page 17894)	This is ErrorCode, a member of class EOleSysError.

114.1.3.3.1 EOleSysError.ErrorCode Property

Pascal

```
property ErrorCode: Integer;
```

C++

```
__property int ErrorCode;
```

Description

This is ErrorCode, a member of class EOleSysError.

114.1.4 TAutoDispatch Class

Class Hierarchy

Pascal

```
TAutoDispatch = class(IDispatch);
```

C++

```
class TAutoDispatch : public IDispatch;
```

File

OleAuto

Description

This is class OleAuto.TAutoDispatch.

114.1.4.1 TAutoDispatch Members

The following tables list the members exposed by TAutoDispatch.

Public Methods

	Name	Description
V	AddRef (see page 17895)	This is AddRef, a member of class TAutoDispatch.
	Create (see page 17895)	This is Create, a member of class TAutoDispatch.
V	GetAutoObject (see page 17895)	This is GetAutoObject, a member of class TAutoDispatch.
V	GetIDsOfNames (see page 17896)	This is GetIDsOfNames, a member of class TAutoDispatch.
V	GetTypeInfo (see page 17896)	This is GetTypeInfo, a member of class TAutoDispatch.
V	GetTypeInfoCount (see page 17896)	This is GetTypeInfoCount, a member of class TAutoDispatch.
V	Invoke (see page 17896)	This is Invoke, a member of class TAutoDispatch.
V	QueryInterface (see page 17897)	This is QueryInterface, a member of class TAutoDispatch.
V	Release (see page 17897)	This is Release, a member of class TAutoDispatch.

Public Properties

	Name	Description
R	AutoObject (see page 17897)	This is AutoObject, a member of class TAutoDispatch.

114.1.4.2 TAutoDispatch.Create Constructor

Pascal

```
constructor Create(AutoObject: TAutoObject);
```

C++

```
__fastcall TAutoDispatch(TAutoObject AutoObject);
```

Description

This is Create, a member of class TAutoDispatch.

114.1.4.3 TAutoDispatch Methods

The methods of the TAutoDispatch class are listed here.

Public Methods

	Name	Description
	AddRef (see page 17895)	This is AddRef, a member of class TAutoDispatch.
	GetAutoObject (see page 17895)	This is GetAutoObject, a member of class TAutoDispatch.
	GetIDsOfNames (see page 17896)	This is GetIDsOfNames, a member of class TAutoDispatch.
	GetTypeInfo (see page 17896)	This is GetTypeInfo, a member of class TAutoDispatch.
	GetTypeInfoCount (see page 17896)	This is GetTypeInfoCount, a member of class TAutoDispatch.
	Invoke (see page 17896)	This is Invoke, a member of class TAutoDispatch.
	QueryInterface (see page 17897)	This is QueryInterface, a member of class TAutoDispatch.
	Release (see page 17897)	This is Release, a member of class TAutoDispatch.

114.1.4.3.1 TAutoDispatch.AddRef Method

Pascal

```
function AddRef: Longint; override;
```

C++

```
virtual __fastcall Longint AddRef();
```

Description

This is AddRef, a member of class TAutoDispatch.

114.1.4.3.2 TAutoDispatch.GetAutoObject Method

Pascal

```
function GetAutoObject: TAutoObject; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall TAutoObject GetAutoObject();
```

Description

This is GetAutoObject, a member of class TAutoDispatch.

114.1.4.3.3 TAutoDispatch.GetIDsOfNames Method

Pascal

```
function GetIDsOfNames(const iid: TIID; rgszNames: POleStrList; cNames: Integer; lcid: TLCID;
rgdispid: PDispIDList): HResult; override;
```

C++

```
virtual __fastcall HRESULT GetIDsOfNames(const TIID iid, POleStrList rgszNames, int cNames,
TLCID lcid, PDispIDList rgdispid);
```

Description

This is GetIDsOfNames, a member of class TAutoDispatch.

114.1.4.3.4 TAutoDispatch.GetTypeInfo Method

Pascal

```
function GetTypeInfo(itinfo: Integer; lcid: TLCID; var tinfo: ITypeInfo): HResult; override;
```

C++

```
virtual __fastcall HRESULT GetTypeInfo(int itinfo, TLCID lcid, ITypeInfo tinfo);
```

Description

This is GetTypeInfo, a member of class TAutoDispatch.

114.1.4.3.5 TAutoDispatch.GetTypeInfoCount Method

Pascal

```
function GetTypeInfoCount(var ctinfo: Integer): HResult; override;
```

C++

```
virtual __fastcall HRESULT GetTypeInfoCount(int ctinfo);
```

Description

This is GetTypeInfoCount, a member of class TAutoDispatch.

114.1.4.3.6 TAutoDispatch.Invoke Method

Pascal

```
function Invoke(dispIDMember: TDispID; const iid: TIID; lcid: TLCID; flags: Word; var
dispParams: TDispParams; varResult: PVariant; excepInfo: PExcepInfo; argErr: PInteger):
HResult; override;
```

C++

```
virtual __fastcall HRESULT Invoke(TDispID dispIDMember, const TIID iid, TLCID lcid, Word
flags, TDispParams dispParams, PVariant varResult, PExcepInfo excepInfo, PInteger argErr);
```

Description

This is Invoke, a member of class TAutoDispatch.

114.1.4.3.7 TAutoDispatch.QueryInterface Method

Pascal

```
function QueryInterface(const iid: TIID; var obj): HResult; override;
```

C++

```
virtual __fastcall HRESULT QueryInterface(const TIID iid,  obj);
```

Description

This is QueryInterface, a member of class TAutoDispatch.

114.1.4.3.8 TAutoDispatch.Release Method

Pascal

```
function Release: Longint; override;
```

C++

```
virtual __fastcall Longint Release();
```

Description

This is Release, a member of class TAutoDispatch.

114.1.4.4 TAutoDispatch Properties

The properties of the TAutoDispatch class are listed here.

Public Properties

	Name	Description
R	AutoObject (see page 17897)	This is AutoObject, a member of class TAutoDispatch.

114.1.4.4.1 TAutoDispatch.AutoObject Property

Pascal

```
property AutoObject: TAutoObject;
```

C++

```
__property TAutoObject AutoObject;
```

Description

This is AutoObject, a member of class TAutoDispatch.

114.1.5 TAutomation Class

Class Hierarchy

OleAuto.TAutomation

Pascal

```
TAutomation = class;
```

C++

```
class TAutomation;
```

File

OleAuto

Description

This is class OleAuto.TAutomation.

114.1.5.1 TAutomation Members

The following tables list the members exposed by TAutomation.

Public Events

	Name	Description
	OnLastRelease (see page 17900)	This is OnLastRelease, a member of class TAutomation.

Public Methods

	Name	Description
	Create (see page 17898)	This is Create, a member of class TAutomation.
V	Destroy (see page 17898)	This is Destroy, a member of class TAutomation.
	RegisterClass (see page 17899)	This is RegisterClass, a member of class TAutomation.
	UpdateRegistry (see page 17899)	This is UpdateRegistry, a member of class TAutomation.

Public Properties

	Name	Description
R	AutoObjectCount (see page 17899)	This is AutoObjectCount, a member of class TAutomation.
	IsInprocServer (see page 17900)	This is IsInprocServer, a member of class TAutomation.
R	StartMode (see page 17900)	This is StartMode, a member of class TAutomation.

114.1.5.2 TAutomation.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TAutomation();
```

Description

This is Create, a member of class TAutomation.

114.1.5.3 TAutomation.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAutomation();
```

Description

This is Destroy, a member of class TAutomation.

114.1.5.4 TAutomation Methods

The methods of the TAutomation class are listed here.

Public Methods

	Name	Description
	RegisterClass (see page 17899)	This is RegisterClass, a member of class TAutomation.
	UpdateRegistry (see page 17899)	This is UpdateRegistry, a member of class TAutomation.

114.1.5.4.1 TAutomation.RegisterClass Method

Pascal

```
procedure RegisterClass(const AutoClassInfo: TAutoClassInfo);
```

C++

```
__fastcall RegisterClass(const TAutoClassInfo AutoClassInfo);
```

Description

This is RegisterClass, a member of class TAutomation.

114.1.5.4.2 TAutomation.UpdateRegistry Method

Pascal

```
procedure UpdateRegistry(Register: Boolean);
```

C++

```
__fastcall UpdateRegistry(Boolean Register);
```

Description

This is UpdateRegistry, a member of class TAutomation.

114.1.5.5 TAutomation Properties

The properties of the TAutomation class are listed here.

Public Properties

	Name	Description
R	AutoObjectCount (see page 17899)	This is AutoObjectCount, a member of class TAutomation.
	IsInprocServer (see page 17900)	This is IsInprocServer, a member of class TAutomation.
R	StartMode (see page 17900)	This is StartMode, a member of class TAutomation.

114.1.5.5.1 TAutomation.AutoObjectCount Property

Pascal

```
property AutoObjectCount: Integer;
```

C++

```
__property int AutoObjectCount;
```

Description

This is AutoObjectCount, a member of class TAutomation.

114.1.5.5.2 TAutomation.IsInprocServer Property

Pascal

```
property IsInprocServer: Boolean;
```

C++

```
__property Boolean IsInprocServer;
```

Description

This is IsInprocServer, a member of class TAutomation.

114.1.5.5.3 TAutomation.StartMode Property

Pascal

```
property StartMode: TStartMode;
```

C++

```
__property TStartMode StartMode;
```

Description

This is StartMode, a member of class TAutomation.

114.1.5.6 TAutomation Events

The events of the TAutomation class are listed here.

Public Events

	Name	Description
	OnLastRelease (see page 17900)	This is OnLastRelease, a member of class TAutomation.

114.1.5.6.1 TAutomation.OnLastRelease Event

Pascal

```
property OnLastRelease: TLastReleaseEvent;
```

C++

```
__property TLastReleaseEvent OnLastRelease;
```

Description

This is OnLastRelease, a member of class TAutomation.

114.1.6 TAutoObject Class

Class Hierarchy

System::TObject → OleAuto.TAutoObject

Pascal

```
TAutoObject = class(TObject);
```

C++

```
class TAutoObject : public TObject;
```

File

OleAuto

Description

This is class OleAuto.TAutoObject.

114.1.6.1 TAutoObject Members

The following tables list the members exposed by TAutoObject.

Protected Methods

	Name	Description
V	CreateAutoDispatch (see page 17903)	This is CreateAutoDispatch, a member of class TAutoObject.
V	GetExceptionInfo (see page 17904)	This is GetExceptionInfo, a member of class TAutoObject.

Public Methods

	Name	Description
	AddRef (see page 17903)	This is AddRef, a member of class TAutoObject.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 17902)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 17902)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Release (see page 17904)	This is Release, a member of class TAutoObject.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	AutoDispatch (see page 17904)	This is AutoDispatch, a member of class TAutoObject.
R	OleObject (see page 17904)	This is OleObject, a member of class TAutoObject.
R	RefCount (see page 17905)	This is RefCount, a member of class TAutoObject.

114.1.6.2 TAutoObject.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TAutoObject();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

114.1.6.3 TAutoObject.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAutoObject();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

114.1.6.4 TAutoObject Methods

The methods of the TAutoObject class are listed here.

Protected Methods

	Name	Description
	CreateAutoDispatch (see page 17903)	This is CreateAutoDispatch, a member of class TAutoObject.
	GetExceptionInfo (see page 17904)	This is GetExceptionInfo, a member of class TAutoObject.

Public Methods

	Name	Description
	AddRef (see page 17903)	This is AddRef, a member of class TAutoObject.
	Release (see page 17904)	This is Release, a member of class TAutoObject.

114.1.6.4.1 TAutoObject.AddRef Method

Pascal

```
function AddRef: Integer;
```

C++

```
__fastcall int AddRef();
```

Description

This is AddRef, a member of class TAutoObject.

114.1.6.4.2 TAutoObject.CreateAutoDispatch Method

Pascal

```
function CreateAutoDispatch: TAutoDispatch; virtual;
```

C++

```
virtual __fastcall TAutoDispatch CreateAutoDispatch();
```

Description

This is CreateAutoDispatch, a member of class TAutoObject.

114.1.6.4.3 TAutoObject.GetExceptionInfo Method

Pascal

```
procedure GetExceptionInfo(ExceptObject: TObject; var ExcepInfo: TExcepInfo); virtual;
```

C++

```
virtual __fastcall GetExceptionInfo(TObject * ExceptObject, TExcepInfo ExcepInfo);
```

Description

This is GetExceptionInfo, a member of class TAutoObject.

114.1.6.4.4 TAutoObject.Release Method

Pascal

```
function Release: Integer;
```

C++

```
__fastcall int Release();
```

Description

This is Release, a member of class TAutoObject.

114.1.6.5 TAutoObject Properties

The properties of the TAutoObject class are listed here.

Public Properties

	Name	Description
R	AutoDispatch (see page 17904)	This is AutoDispatch, a member of class TAutoObject.
R	OleObject (see page 17904)	This is OleObject, a member of class TAutoObject.
R	RefCount (see page 17905)	This is RefCount, a member of class TAutoObject.

114.1.6.5.1 TAutoObject.AutoDispatch Property

Pascal

```
property AutoDispatch: TAutoDispatch;
```

C++

```
__property TAutoDispatch AutoDispatch;
```

Description

This is AutoDispatch, a member of class TAutoObject.

114.1.6.5.2 TAutoObject.OleObject Property

Pascal

```
property OleObject: Variant;
```

C++

```
__property Variant OleObject;
```

Description

This is OleObject, a member of class TAutoObject.

114.1.6.5.3 TAutoObject.RefCount Property

Pascal

```
property RefCount: Integer;
```

C++

```
__property int RefCount;
```

Description

This is RefCount, a member of class TAutoObject.

114.1.7 TRegistryClass Class

Class Hierarchy

OleAuto.TRegistryClass

Pascal

```
TRegistryClass = class;
```

C++

```
class TRegistryClass;
```

File

OleAuto

Description

This is class OleAuto.TRegistryClass.

114.1.7.1 TRegistryClass Members

The following tables list the members exposed by TRegistryClass.

Public Methods

	Name	Description
	Create (see page 17905)	This is Create, a member of class TRegistryClass.
V	Destroy (see page 17906)	This is Destroy, a member of class TRegistryClass.
	UpdateRegistry (see page 17906)	This is UpdateRegistry, a member of class TRegistryClass.

114.1.7.2 TRegistryClass.Create Constructor

Pascal

```
constructor Create(const AutoClassInfo: TAutoClassInfo);
```

C++

```
__fastcall TRegistryClass(const TAutoClassInfo AutoClassInfo);
```

Description

This is Create, a member of class TRegistryClass.

114.1.7.3 TRegistryClass.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRegistryClass();
```

Description

This is Destroy, a member of class TRegistryClass.

114.1.7.4 TRegistryClass Methods

The methods of the TRegistryClass class are listed here.

Public Methods

	Name	Description
	UpdateRegistry (see page 17906)	This is UpdateRegistry, a member of class TRegistryClass.

114.1.7.4.1 TRegistryClass.UpdateRegistry Method

Pascal

```
procedure UpdateRegistry(Register: Boolean);
```

C++

```
__fastcall UpdateRegistry(Boolean Register);
```

Description

This is UpdateRegistry, a member of class TRegistryClass.

114.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	ClassIDToProgID (see page 17907)	This is function OleAuto.ClassIDToProgID.
	ClassIDToString (see page 17907)	This is function OleAuto.ClassIDToString.
	CreateOleObject (see page 17908)	This is function OleAuto.CreateOleObject.
	DispInvoke (see page 17908)	This is function OleAuto.DispInvoke.
	DispInvokeError (see page 17908)	This is function OleAuto.DispInvokeError.
	DllCanUnloadNow (see page 17909)	This is function OleAuto.DllCanUnloadNow.
	DllGetClassObject (see page 17909)	This is function OleAuto.DllGetClassObject.
	DllRegisterServer (see page 17909)	This is function OleAuto.DllRegisterServer.
	DllUnregisterServer (see page 17910)	This is function OleAuto.DllUnregisterServer.
	GetActiveOleObject (see page 17910)	This is function OleAuto.GetActiveOleObject.
	OleCheck (see page 17910)	This is function OleAuto.OleCheck.
	OleError (see page 17911)	This is function OleAuto.OleError.
	ProgIDToClassID (see page 17911)	This is function OleAuto.ProgIDToClassID.
	StringToClassID (see page 17911)	This is function OleAuto.StringToClassID.
	VarFromInterface (see page 17912)	This is function OleAuto.VarFromInterface.
	VarToAutoObject (see page 17912)	This is function OleAuto.VarToAutoObject.
	VarToInterface (see page 17912)	This is function OleAuto.VarToInterface.

114.2.1 OleAuto.ClassIDToProgID Function

Pascal

```
function ClassIDToProgID(const ClassID: TCLSID): string;
```

C++

```
AnsiString ClassIDToProgID(const TCLSID ClassID);
```

File

OleAuto

Description

This is function OleAuto.ClassIDToProgID.

114.2.2 OleAuto.ClassIDToString Function

Pascal

```
function ClassIDToString(const ClassID: TCLSID): string;
```

C++

```
AnsiString ClassIDToString(const TCLSID ClassID);
```

File

OleAuto

Description

This is function OleAuto.ClassIDToString.

114.2.3 OleAuto.CreateOleObject Function

Pascal

```
function CreateOleObject(const ClassName: string): Variant;
```

C++

```
Variant CreateOleObject(const AnsiString ClassName);
```

File

OleAuto

Description

This is function OleAuto.CreateOleObject.

114.2.4 OleAuto.DispInvoke Function

Pascal

```
procedure DispInvoke(Dispatch: IDispatch; CallDesc: PCallDesc; DispIDs: PDispIDList; Params:
Pointer; Result: PVariant);
```

C++

```
DispInvoke(IDispatch Dispatch, PCallDesc CallDesc, PDispIDList DispIDs, void * Params,
PVariant Result);
```

File

OleAuto

Description

This is function OleAuto.DispInvoke.

114.2.5 OleAuto.DispInvokeError Function

Pascal

```
procedure DispInvokeError(Status: HResult; const ExcepInfo: TExcepInfo);
```

C++

```
DispInvokeError(HRESULT Status, const TExcepInfo ExcepInfo);
```

File

OleAuto

Description

This is function OleAuto.DispInvokeError.

114.2.6 OleAuto.DllCanUnloadNow Function

Pascal

```
function DllCanUnloadNow: HResult; stdcall;
```

C++

```
__stdcall HRESULT DllCanUnloadNow();
```

File

OleAuto

Description

This is function OleAuto.DllCanUnloadNow.

114.2.7 OleAuto.DllGetClassObject Function

Pascal

```
function DllGetClassObject(const CLSID: TCLSID; const IID: TIID; var Obj): HResult; stdcall;
```

C++

```
__stdcall HRESULT DllGetClassObject(const TCLSID CLSID, const TIID IID,  Obj);
```

File

OleAuto

Description

This is function OleAuto.DllGetClassObject.

114.2.8 OleAuto.DllRegisterServer Function

Pascal

```
function DllRegisterServer: HResult; stdcall;
```

C++

```
__stdcall HRESULT DllRegisterServer();
```

File

OleAuto

Description

This is function OleAuto.DllRegisterServer.

114.2.9 OleAuto.DllUnregisterServer Function

Pascal

```
function DllUnregisterServer: HResult; stdcall;
```

C++

```
__stdcall HRESULT DllUnregisterServer();
```

File

OleAuto

Description

This is function OleAuto.DllUnregisterServer.

114.2.10 OleAuto.GetActiveOleObject Function

Pascal

```
function GetActiveOleObject(const ClassName: string): Variant;
```

C++

```
Variant GetActiveOleObject(const AnsiString ClassName);
```

File

OleAuto

Description

This is function OleAuto.GetActiveOleObject.

114.2.11 OleAuto.OleCheck Function

Pascal

```
procedure OleCheck(Result: HResult);
```

C++

```
OleCheck(HRESULT Result);
```

File

OleAuto

Description

This is function OleAuto.OleCheck.

114.2.12 OleAuto.OleError Function

Pascal

```
procedure OleError(ErrorCode: HResult);
```

C++

```
OleError(HRESULT ErrorCode);
```

File

OleAuto

Description

This is function OleAuto.OleError.

114.2.13 OleAuto.ProgIDToClassID Function

Pascal

```
function ProgIDToClassID(const ProgID: string): TCLSID;
```

C++

```
TCLSID ProgIDToClassID(const AnsiString ProgID);
```

File

OleAuto

Description

This is function OleAuto.ProgIDToClassID.

114.2.14 OleAuto.StringToClassID Function

Pascal

```
function StringToClassID(const S: string): TCLSID;
```

C++

```
TCLSID StringToClassID(const AnsiString S);
```

File

OleAuto

Description

This is function OleAuto.StringToClassID.

114.2.15 OleAuto.VarFromInterface Function

Pascal

```
function VarFromInterface(Unknown: IUnknown): Variant;
```

C++

```
Variant VarFromInterface(IUnknown Unknown);
```

File

OleAuto

Description

This is function OleAuto.VarFromInterface.

114.2.16 OleAuto.VarToAutoObject Function

Pascal

```
function VarToAutoObject(const V: Variant): TAutoObject;
```

C++

```
TAutoObject VarToAutoObject(const Variant V);
```

File

OleAuto

Description

This is function OleAuto.VarToAutoObject.

114.2.17 OleAuto.VarToInterface Function

Pascal

```
function VarToInterface(const V: Variant): IDispatch;
```

C++

```
IDispatch VarToInterface(const Variant V);
```

File

OleAuto

Description

This is function OleAuto.VarToInterface.

114.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAutoClassInstancing (see page 17913)	This is record OleAuto.TAutoClassInstancing.
	TStartMode (see page 17914)	This is record OleAuto.TStartMode.

Records

	Name	Description
	TAutoClassInfo (see page 17913)	This is record OleAuto.TAutoClassInfo.

114.3.1 OleAuto.TAutoClassInfo Record

Pascal

```
TAutoClassInfo = record
  AutoClass: TAutoClass;
  ProgID: string;
  ClassID: string;
  Description: string;
  Instancing: TAutoClassInstancing;
end;
```

C++

```
struct TAutoClassInfo {
  TAutoClass AutoClass;
  AnsiString ProgID;
  AnsiString ClassID;
  AnsiString Description;
  TAutoClassInstancing Instancing;
};
```

File

OleAuto

Description

This is record OleAuto.TAutoClassInfo.

114.3.2 OleAuto.TAutoClassInstancing Enumeration

Pascal

```
TAutoClassInstancing = (
  acInternal,
  acSingleInstance,
  acMultiInstance
);
```

C++

```
enum TAutoClassInstancing {
  acInternal,
  acSingleInstance,
  acMultiInstance
};
```

File

OleAuto

Description

This is record OleAuto.TAutoClassInstancing.

114.3.3 OleAuto.TStartMode Enumeration

Pascal

```
TStartMode = (
  smStandalone,
  smAutomation,
  smRegServer,
  smUnregServer
);
```

C++

```
enum TStartMode {
  smStandalone,
  smAutomation,
  smRegServer,
  smUnregServer
};
```

File

OleAuto

Description

This is record OleAuto.TStartMode.

114.4 Types

The following table lists types in this documentation.

Types

Name	Description
TAutoClass (see page 17915)	This is type OleAuto.TAutoClass.
TLastReleaseEvent (see page 17915)	This is type OleAuto.TLastReleaseEvent.

114.4.1 OleAuto.TAutoClass Type

Pascal

```
TAutoClass = class of TAutoObject;
```

C++

```
TAutoObject TAutoClass;
```

File

OleAuto

Description

This is type OleAuto.TAutoClass.

114.4.2 OleAuto.TLastReleaseEvent Type

Pascal

```
TLastReleaseEvent = procedure (var Shutdown: Boolean) of object;
```

C++

```
(var Shutdown: Boolean) ( TLastReleaseEvent)();
```

File

OleAuto

Description

This is type OleAuto.TLastReleaseEvent.

114.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
Automation (see page 17916)	This is variable OleAuto.Automation.

114.5.1 OleAuto.Automation Variable

Pascal

```
Automation: TAutomation;
```

C++

```
TAutomation Automation;
```

File

OleAuto

Description

This is variable OleAuto.Automation.

114.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
MaxDispArgs (see page 17917)	This is constant OleAuto.MaxDispArgs.

114.6.1 OleAuto.MaxDispArgs Constant

Pascal

```
MaxDispArgs = 32;
```

C++

```
MaxDispArgs = 32;
```

File

OleAuto

Description

This is constant OleAuto.MaxDispArgs.

115 OleCtnrs Namespace

This is namespace OleCtnrs.

Topics

Name	Description
Classes (see page 17921)	The following table lists classes in this documentation.
Interfaces (see page 18009)	The following table lists interfaces in this documentation.
Functions (see page 18011)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 18012)	The following table lists structs, records, enums in this documentation.
Types (see page 18016)	The following table lists types in this documentation.
Constants (see page 18017)	The following table lists constants in this documentation.

Classes

	Name	Description
	TOleContainer (see page 17921)	TOleContainer embeds or links OLE objects.
	TOleForm (see page 18006)	TOleForm is an object that provides OLE support for a TForm object.

Constants

Name	Description
ovDiscardUndoState (see page 18017)	This is constant OleCtnrs.ovDiscardUndoState.
ovHide (see page 18017)	This is constant OleCtnrs.ovHide.
ovInPlaceActivate (see page 18018)	This is constant OleCtnrs.ovInPlaceActivate.
ovOpen (see page 18018)	This is constant OleCtnrs.ovOpen.
ovPrimary (see page 18018)	This is constant OleCtnrs.ovPrimary.
ovShow (see page 18019)	This is constant OleCtnrs.ovShow.
ovUIActivate (see page 18019)	This is constant OleCtnrs.ovUIActivate.

Functions

	Name	Description
	DestroyMetaPict (see page 18011)	Frees a metafile resource given its handle.

Interfaces

	Name	Description
	IVCLFrameForm (see page 18009)	This is class OleCtnrs.IVCLFrameForm.

Structs, Records, Enums

	Name	Description
	TAutoActivate (see page 18012)	This is record OleCtnrs.TAutoActivate.
	TCreateInfo (see page 18012)	This is record OleCtnrs.TCreateInfo.
	TCreateType (see page 18013)	This is record OleCtnrs.TCreateType.
	TObjectState (see page 18013)	This is record OleCtnrs.TObjectState.
	TSizeMode (see page 18014)	This is record OleCtnrs.TSizeMode.
	TVerbInfo (see page 18014)	This is record OleCtnrs.TVerbInfo.

Types

Name	Description
TObjectMoveEvent (see page 18016)	This is type OleCtnrs.TObjectMoveEvent.

115.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TOleContainer (see page 17921)	TOleContainer embeds or links OLE objects.
	TOleForm (see page 18006)	TOleForm is an object that provides OLE support for a TForm object.

115.1.1 TOleContainer Class

TOleContainer embeds or links OLE objects.

Class Hierarchy

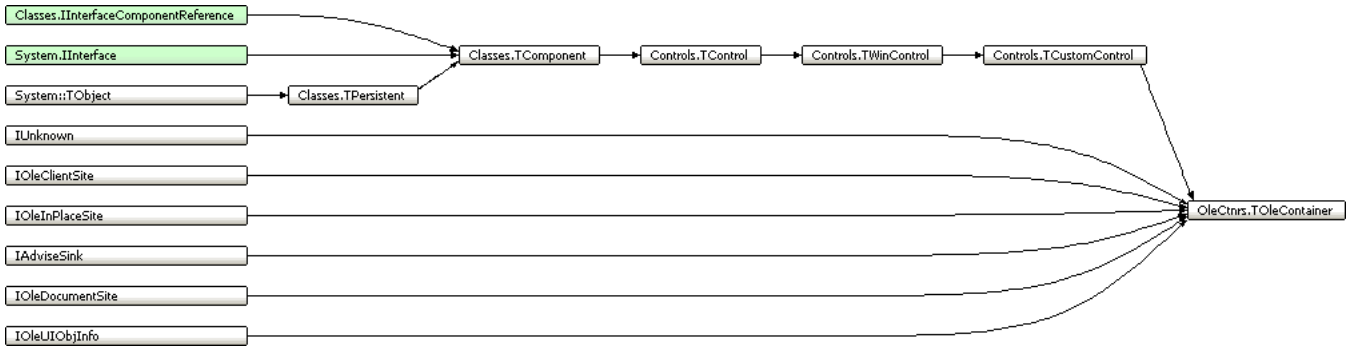

Pascal

```
TOleContainer = class(TCustomControl, IUnknown, IOleClientSite, IOleInPlaceSite, IAdviseSink,
IOleDocumentSite, IOleUIObjInfo);
```

C++

```
class TOleContainer : public TCustomControl, public IUnknown, public IOleClientSite, public
IOleInPlaceSite, public IAdviseSink, public IOleDocumentSite, public IOleUIObjInfo;
```

File

OleCtnrs

Description

Use TOleContainer to handle many of the complexities of OLE 2.0. TOleContainer lets the user choose an OLE object to insert by simply calling the InsertObjectDialog (see page 17943) method. TOleContainer can create either an embedded OLE object or a linked OLE object.

TOleContainer automatically handles menu merging -- the process of combining the container form's menu with that of an in-place activated OLE object's server application. The menu items' GroupIndex property settings control how menus are merged. Those main menu items with GroupIndex values of 0, 2, and 4 remain; TOleContainer merges the server application's menus and replaces the main menu items with GroupIndex values of 1, 3, and 5 (if they exist).

OLE objects that are activated in-place add their server's toolbars directly into the container application's window. Normally, any

panels used for toolbars are replaced by the OLE object's server's toolbars. Prevent this by setting a panel's Locked property to true.

When using TOleContainer in an SDI application (the main form's FormStyle property is fsNormal rather than fsMDIForm), place the TOleContainer component inside a panel whose Align (see page 17951) property is set to alClient. TOleContainer replaces an SDI form's toolbars as described above, and using a panel automatically adjusts the amount of space available to the OLE container.

See Also

TForm (see page 13992)

TMainMenu (see page 16056)

TMenuItem (see page 16102)

TOleControl (see page 18030)

TPanel (see page 13032)

Creating Clients for Servers That Do Not Have a Type Library

Active Documents

Building COM Applications

115.1.1.1 TOleContainer Members

The following tables list the members exposed by TOleContainer.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.

	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 17935)	Called when an application uses a component interface.
	_Release (see page 17935)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 17936)	Sends a CM_CHANGED message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 17939)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
V	DblClick (see page 17940)	OnDblClick event dispatcher.

	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 17940)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 17941)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.

	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 17942)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 17943)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 17945)	OnMouseDown event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 17945)	Renders the image of a custom control.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 17947)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.

	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.

	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ChangeIconDialog (see page 17936)	Displays the Change Icon OLE dialog box.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Close (see page 17937)	Deactivates the OLE object.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Copy (see page 17937)	Copies the OLE object to the Windows clipboard.
V	Create (see page 17933)	Creates and initializes an instance of TOleContainer (see page 17921).
	CreateLinkToFile (see page 17937)	Creates a linked OLE object from the contents of a file.
	CreateObject (see page 17938)	Creates an embedded OLE object given its class name (also known as programmatic identifier).
	CreateObjectFromFile (see page 17938)	Creates an embedded OLE object from the contents of a file.
	CreateObjectFromInfo (see page 17939)	Creates an OLE object matching the specified TCreateInfo (see page 18012) record.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 17934)	Destroys an instance of TOleContainer (see page 17921).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DestroyObject (see page 17941)	Destroys the container's OLE object.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DoVerb (see page 17941)	Requests that the OLE object perform some action.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIconMetaPict (see page 17942)	Returns the global memory handle of a metafile containing the OLE object's icon.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InsertObjectDialog (see page 17943)	Displays the Insert Object OLE dialog box.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	LoadFromFile (see page 17944)	Loads an OLE object from the specified file.
	LoadFromStream (see page 17944)	Loads an OLE object from the specified stream.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ObjectPropertiesDialog (see page 17945)	Displays the OLE Object Properties property sheet.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Paste (see page 17946)	Pastes the contents of the Windows clipboard as an embedded object.
	PasteSpecialDialog (see page 17946)	Displays the Paste (see page 17946) Special OLE dialog box.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	Run (see page 17947)	Runs the OLE object's server application, but doesn't open or activate the OLE object itself.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveAsDocument (see page 17948)	Saves the OLE object to the specified file in native OLE Document format.
	SaveToFile (see page 17948)	Saves the OLE object to the specified file.
	SaveToStream (see page 17949)	Saves the OLE object to the specified stream.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UpdateObject (see page 17949)	Refreshes the OLE object.
	UpdateVerbs (see page 17949)	Refreshes the list of verbs the OLE object can respond to.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	CanPaste (see page 17957)	Indicates whether the Windows clipboard contains an OLE object that can be pasted into the OLE container.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.

R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
R	Linked (see page 17965)	Indicates whether the OLE object is linked or embedded.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Modified (see page 17965)	Indicates whether the OLE object has been modified.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	NewInserted (see page 17966)	Indicates whether the OLE object was created rather than pasted in or created as a link to a file.
R	ObjectVerbs (see page 17966)	Lists the verbs that the OLE object supports.
R	OleClassName (see page 17967)	Specifies the class name of the OLE object.
R	OleObject (see page 17967)	Represents the OLE object hosted by the OLE container.
R	OleObjectInterface (see page 17968)	Provides access to the IOleObject interface for the OLE object.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	PrimaryVerb (see page 17974)	Specifies the index of the primary (default) verb for the OLE object.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
R	SourceDoc (see page 17977)	Specifies the name of the source document for a linked OLE object.
R	State (see page 17977)	Describes the state of the OLE object.
R	StorageInterface (see page 17977)	Provides access to the IStorage interface for the OLE object.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnActivate (see page 17981)	Occurs immediately after the OLE object is activated.
	OnDeactivate (see page 17981)	Occurs immediately after the OLE object is deactivated.
	OnDragDrop (see page 17982)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 17984)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 17986)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 17987)	Occurs when a control receives the input focus.
	OnExit (see page 17990)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 17991)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 17994)	Occurs when a key is pressed.
	OnKeyUp (see page 17995)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 17997)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 17998)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 18000)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 18002)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnObjectMove (see page 18005)	Occurs when the OLE container is notified that the OLE object has moved or resized.
	OnResize (see page 18005)	Occurs immediately after the OLE container window is resized.
	OnStartDrag (see page 18005)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 17951)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AllowActiveDoc (see page 17954)	Specifies whether the OLE container supports the IOleDocumentSite interface.
	AllowInPlace (see page 17955)	Determines whether the OLE container allows in-place activation.
	Anchors (see page 17955)	Specifies how the control is anchored to its parent.
	AutoActivate (see page 17956)	Determines how the object in an OLE container is activated.
	AutoVerbMenu (see page 17956)	Determines whether the OLE container automatically creates a pop-up menu containing the OLE object's verbs.
	BorderStyle (see page 17957)	Determines whether a single line is drawn around the OLE container.
	Caption (see page 17957)	Specifies a text string that identifies the control to the user.
	Color (see page 17960)	Specifies the background color of the control.
	Constraints (see page 17961)	Specifies the size constraints for the control.
	CopyOnSave (see page 17961)	Determines whether the SaveToFile (see page 17948) and SaveToStream (see page 17949) methods first create a copy of the OLE object.
	Ctl3D (see page 17962)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 17962)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 17963)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 17964)	Controls whether the control responds to mouse, keyboard, and timer events.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.

	Iconic (see page 17965)	Determines whether the OLE object is displayed as an icon.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldStreamFormat (see page 17966)	Indicates whether the OLE object is loaded and saved using the VCL format.
	ParentColor (see page 17968)	Determines where a control looks for its color information.
	ParentCtl3D (see page 17969)	Determines where a component looks to determine if it should appear three dimensional.
	ParentShowHint (see page 17969)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 17969)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 17974)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SizeMode (see page 17976)	Determines how the OLE object is sized in the OLE container.
	TabOrder (see page 17978)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 17978)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 17979)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

115.1.1.2 TOleContainer.Create Constructor

Creates and initializes an instance of TOleContainer (see page 17921).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TOleContainer(TComponent * AOwner);
```

Description

Call Create to instantiate a TOleContainer (see page 17921) object at runtime. OLE containers placed on forms at design time are created automatically.

Pass a component to the constructor to supply the OLE container with an Owner that is responsible for freeing it.

After calling the inherited constructor, Create

Sets ControlStyle to [csClickEvents, csSetCaption, csOpaque, csDoubleClicks].

Sets TabStop (see page 17978) to true.

Sets ParentColor (see page 17968) to false and BorderStyle (see page 17957) to bsSingle.

Sets AllowActiveDoc (see page 17954) and AllowInPlace (see page 17955) to true.

Sets AutoActivate (see page 17956) to aaDoubleClick and AutoVerbMenu (see page 17956) to true.

Sets CopyOnSave (see page 17961) to true.

See Also

AllowActiveDoc (see page 17954)

AllowInPlace (see page 17955)

AutoActivate (see page 17956)

AutoVerbMenu (see page 17956)

BorderStyle (see page 17957)

ControlStyle (see page 5814)

CopyOnSave (see page 17961)

ParentColor (see page 5834)

TabStop (see page 6211)

115.1.1.3 TOleContainer.Destroy Destructor

Destroys an instance of TOleContainer (see page 17921).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TOleContainer();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TOleContainer (see page 17921) reference is not nil, and only then calls Destroy.Destroy frees the OLE object before calling the inherited destructor.

See Also

DestroyObject (see page 17941)

Free (see page 22300)

115.1.1.4 TOleContainer Methods

The methods of the TOleContainer class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 17935)	Called when an application uses a component interface.
	_Release (see page 17935)	Called when an application releases a component interface.
	Changed (see page 17936)	Sends a CM_CHANGED message to the control.
V	CreateParams (see page 17939)	Initializes a window-creation parameter data structure.
V	DblClick (see page 17940)	OnDblClick event dispatcher.
V	DefineProperties (see page 17940)	Provides methods to read and write the IsControl property to a stream such as a form file.
V	DoEnter (see page 17941)	Respond to receiving input focus.
V	GetPopupMenu (see page 17942)	Returns the pop-up menu associated with the control.
V	KeyDown (see page 17943)	Respond to key press events.
V	MouseDown (see page 17945)	OnMouseDown event dispatcher.
V	Paint (see page 17945)	Renders the image of a custom control.
V	QueryInterface (see page 17947)	Returns a reference to a specified interface if the object supports that interface.

Public Methods

	Name	Description
	ChangeIconDialog (see page 17936)	Displays the Change Icon OLE dialog box.
	Close (see page 17937)	Deactivates the OLE object.
	Copy (see page 17937)	Copies the OLE object to the Windows clipboard.
	CreateLinkToFile (see page 17937)	Creates a linked OLE object from the contents of a file.

	CreateObject (see page 17938)	Creates an embedded OLE object given its class name (also known as programmatic identifier).
	CreateObjectFromFile (see page 17938)	Creates an embedded OLE object from the contents of a file.
	CreateObjectFromInfo (see page 17939)	Creates an OLE object matching the specified TCreateInfo (see page 18012) record.
	DestroyObject (see page 17941)	Destroys the container's OLE object.
	DoVerb (see page 17941)	Requests that the OLE object perform some action.
	GetIconMetaPict (see page 17942)	Returns the global memory handle of a metafile containing the OLE object's icon.
	InsertObjectDialog (see page 17943)	Displays the Insert Object OLE dialog box.
	LoadFromFile (see page 17944)	Loads an OLE object from the specified file.
	LoadFromStream (see page 17944)	Loads an OLE object from the specified stream.
	ObjectPropertiesDialog (see page 17945)	Displays the OLE Object Properties property sheet.
	Paste (see page 17946)	Pastes the contents of the Windows clipboard as an embedded object.
	PasteSpecialDialog (see page 17946)	Displays the Paste (see page 17946) Special OLE dialog box.
	Run (see page 17947)	Runs the OLE object's server application, but doesn't open or activate the OLE object itself.
	SaveAsDocument (see page 17948)	Saves the OLE object to the specified file in native OLE Document format.
	SaveToFile (see page 17948)	Saves the OLE object to the specified file.
	SaveToStream (see page 17949)	Saves the OLE object to the specified stream.
	UpdateObject (see page 17949)	Refreshes the OLE object.
	UpdateVerbs (see page 17949)	Refreshes the list of verbs the OLE object can respond to.

115.1.1.4.1 TOleContainer._AddRef Method

Called when an application uses a component interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef is a basic implementation of the IInterface method, _AddRef.

Note: If the component is a wrapper for a COM object, __AddRef calls the _AddRef method of that COM object, and returns the resulting reference count.

In all other cases, __AddRef simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __AddRef to implement reference counting.

See Also

Invoke (see page 2781)

QueryInterface (see page 2784)

_Release (see page 2769)

ComObject (see page 2793)

_AddRef (see page 22397)

115.1.1.4.2 TOleContainer._Release Method

Called when an application releases a component interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release is a basic implementation of the IInterface method, _Release.

__Release returns the resulting value of the reference count for the component's interface.

Note:

In all other cases, __Release simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __Release to implement reference counting.

See Also

Invoke (see page 2781)

QueryInterface (see page 2784)

_AddRef (see page 2769)

ComObject (see page 2793)

115.1.1.4.3 TOleContainer.Changed Method

Sends a CM_CHANGED message to the control.

Pascal

```
procedure Changed; dynamic;
```

C++

```
__fastcall Changed();
```

Description

Call Changed after changing the properties of the control in a way that affects the properties of a parent object. Changed sends a CM_CHANGED message back to the control. If the control is a windowed control, this message is bubbled up to the parent so that it can make any necessary adjustments.

See Also

Parent (see page 5832)

115.1.1.4.4 TOleContainer.ChangeIconDialog Method

Displays the Change Icon OLE dialog box.

Pascal

```
function ChangeIconDialog: Boolean;
```

C++

```
__fastcall Boolean ChangeIconDialog();
```

Description

Call ChangeIconDialog to let the user choose a different icon (from .ICO, .DLL, and .EXE files) to use when displaying the OLE object as an icon. An OLE object must already be loaded before calling ChangeIconDialog. ChangeIconDialog returns true if the

dialog box was successfully displayed and the user chose the OK button. ChangeIconDialog returns false otherwise.

See Also

Iconic (see page 17965)

ObjectPropertiesDialog (see page 17945)

115.1.1.4.5 TOleContainer.Close Method

Deactivates the OLE object.

Pascal

```
procedure Close;
```

C++

```
__fastcall Close();
```

Description

Call Close to deactivate the OLE object and terminate its server application, without removing it from the container. Any changes the user made to the OLE object are automatically saved. After calling Close, the container's State (see page 17977) property is osLoaded.

Note: If Close is called when no OLE object is loaded, TOleContainer (see page 17921) raises an exception.

See Also

OnDeactivate (see page 17981)

State (see page 17977)

115.1.1.4.6 TOleContainer.Copy Method

Copies the OLE object to the Windows clipboard.

Pascal

```
procedure Copy;
```

C++

```
__fastcall Copy();
```

Description

Call Copy to write the OLE object to the clipboard. If there is no OLE object in the container, Copy raises an exception.

See Also

Paste (see page 17946)

PasteSpecialDialog (see page 17946)

SaveToFile (see page 17948)

SaveToStream (see page 17949)

115.1.1.4.7 TOleContainer.CreateLinkToFile Method

Creates a linked OLE object from the contents of a file.

Pascal

```
procedure CreateLinkToFile(const FileName: string; Iconic: Boolean);
```

C++

```
__fastcall CreateLinkToFile(const AnsiString FileName, Boolean Iconic);
```

Description

Call CreateLinkToFile to create a linked OLE object from the file specified by the FileName parameter. Iconic (see page 17965) specifies whether the object is shown as an icon (true) or displayed as it would be in the server application (false). If there is already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See Also

CreateObject (see page 17938)

CreateObjectFromFile (see page 17938)

CreateObjectFromInfo (see page 17939)

Creating Clients for Servers That Do Not Have a Type Library

115.1.1.4.8 TOleContainer.CreateObject Method

Creates an embedded OLE object given its class name (also known as programmatic identifier).

Pascal

```
procedure CreateObject(const OleClassName: string; Iconic: Boolean);
```

C++

```
__fastcall CreateObject(const AnsiString OleClassName, Boolean Iconic);
```

Description

Call CreateObject to create an OLE object using the class name (also known as programmatic identifier) specified by the OleClassName (see page 17967) parameter. The Iconic (see page 17965) parameter indicates whether the object is shown as an icon (true) or displayed as it would be in the server application (false). If there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See Also

CreateLinkToFile (see page 17937)

CreateObjectFromFile (see page 17938)

CreateObjectFromInfo (see page 17939)

OleClassName (see page 17967)

Creating Clients for Servers That Do Not Have a Type Library

115.1.1.4.9 TOleContainer.CreateObjectFromFile Method

Creates an embedded OLE object from the contents of a file.

Pascal

```
procedure CreateObjectFromFile(const FileName: string; Iconic: Boolean);
```

C++

```
__fastcall CreateObjectFromFile(const AnsiString FileName, Boolean Iconic);
```

Description

Call CreateObjectFromFile to create an OLE object from the file specified by the FileName parameter. The Iconic (see page 17965) parameter indicates whether the object is shown as an icon (true) or displayed as it would be in the server application (false). If there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See Also

CreateLinkToFile (see page 17937)

CreateObject (see page 17938)

CreateObjectFromInfo (see page 17939)

Creating Clients for Servers That Do Not Have a Type Library

115.1.1.4.10 TOleContainer.CreateObjectFromInfo Method

Creates an OLE object matching the specified TCreateInfo (see page 18012) record.

Pascal

```
procedure CreateObjectFromInfo(const CreateInfo: TCreateInfo);
```

C++

```
__fastcall CreateObjectFromInfo(const TCreateInfo CreateInfo);
```

Description

Call CreateObjectFromInfo to create a linked or embedded OLE object based on the specifications in the CreateInfo parameter. CreateInfo specifies whether the object should be created from a file, from a data transfer object, or from a class identifier, and provides the necessary supporting information. CreateInfo also indicates whether the OLE object should be displayed as an Icon, and allows the icon to be specified. If there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

Note: The CreateLinkToFile (see page 17937), CreateObject (see page 17938), and CreateObjectFromFile (see page 17938) methods are one-line calls for the most common types of OLE objects. They can be used instead of filling out the TCreateInfo (see page 18012) record.

See Also

CreateLinkToFile (see page 17937)

CreateObject (see page 17938)

CreateObjectFromFile (see page 17938)

Creating Clients for Servers That Do Not Have a Type Library

115.1.1.4.11 TOleContainer.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override

CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

115.1.1.4.12 TOleContainer.DblClick Method

OnDblClick event dispatcher.

Pascal

```
procedure DblClick; override;
```

C++

```
virtual __fastcall DblClick();
```

Description

DblClick calls any event handler attached to the OnDblClick event. Override DblClick to provide other responses in addition to the inherited event-handler call when the user double-clicks the control.

The DblClick method is triggered by left mouse-button double-click message (WM_LBUTTONDBLCLK) from Windows.

See Also

Click (see page 5743)

OnDblClick (see page 5854)

115.1.1.4.13 TOleContainer.DefineProperties Method

Provides methods to read and write the IsControl property to a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

DefineProperties is called automatically by the streaming system that loads and saves VCL components. It allows persistent objects to read and write properties that are not published. Published properties are loaded and stored automatically.

DefineProperties uses the methods of the filer passed as the Filer parameter to assign methods that can load and save the IsControl property.

Override DefineProperties to persist unpublished properties. Call the DefineBinaryProperty or DefineProperty method of the Filer parameter to assign methods that load or save property values.

Note: When overriding this method, call the parent class's DefineProperties method first.

See Also

DefineBinaryProperty (see page 2837)

DefineProperty (see page 2838)

DefineProperties (see page 2771)

IsControl (see page 5829)

115.1.1.4.14 TOleContainer.DestroyObject Method

Destroys the container's OLE object.

Pascal

```
procedure DestroyObject;
```

C++

```
__fastcall DestroyObject();
```

Description

Call DestroyObject to destroy the OLE object, discarding any unsaved changes the user made to it.

See Also

Close (see page 17937)

OleObject (see page 17967)

Creating Clients for Servers That Do Not Have a Type Library

115.1.1.4.15 TOleContainer.DoEnter Method

Respond to receiving input focus.

Pascal

```
procedure DoEnter; override;
```

C++

```
virtual __fastcall DoEnter();
```

Description

DoEnter is called automatically when the control receives the input focus. As implemented in TWinControl, DoEnter calls the OnEnter event handler, if defined.

Descendant classes that override DoEnter should always call the inherited method.

See Also

DoExit (see page 6144)

OnEnter (see page 6219)

115.1.1.4.16 TOleContainer.DoVerb Method

Requests that the OLE object perform some action.

Pascal

```
procedure DoVerb(Verb: Integer);
```

C++

```
__fastcall DoVerb(int Verb);
```

Description

Call DoVerb to request that the OLE object perform one of the actions listed by the ObjectVerbs (see page 17966) property. OLE defines several verbs, such as ovShow (see page 18019) (to display the OLE object) and ovPrimary (see page 18018) (the default action, usually to activate the OLE object). OLE objects can also define their own custom verbs. Use the ObjectVerbs (see page 17966) property to get a list of those custom verbs.

See Also

ObjectVerbs (see page 17966)

Creating Clients for Servers That Do Not Have a Type Library

115.1.1.4.17 TOleContainer.GetIconMetaPict Method

Returns the global memory handle of a metafile containing the OLE object's icon.

Pascal

```
function GetIconMetaPict: HGlobal;
```

C++

```
__fastcall HGlobal GetIconMetaPict();
```

Description

Call GetIconMetaPict to obtain a metafile handle for drawing the OLE object's icon on another canvas. Use the global DestroyMetaPict (see page 18011) procedure to destroy the handle. If the OLE container is empty, GetIconMetaPict raises an exception.

See Also

DestroyMetaPict (see page 18011)

TMetafile (see page 14549)

TMetafileCanvas (see page 14564)

115.1.1.4.18 TOleContainer.GetPopupMenu Method

Returns the pop-up menu associated with the control.

Pascal

```
function GetPopupMenu: TPopupMenu; override;
```

C++

```
virtual __fastcall TPopupMenu GetPopupMenu();
```

Description

Call GetPopupMenu to find the popup menu associated with the control. Override GetPopupMenu to change which popup menu is associated with the control or to specify different popup menus depending on the state of the control.

GetPopupMenu returns the value of the Popup property.

See Also

PopupMenu (see page 5835)

115.1.1.4.19 TOleContainer.InsertObjectDialog Method

Displays the Insert Object OLE dialog box.

Pascal

```
function InsertObjectDialog: Boolean;
```

C++

```
__fastcall Boolean InsertObjectDialog();
```

Description

Call InsertObjectDialog to let the user create an OLE object. The OLE object can be embedded or linked. InsertObjectDialog returns true if the dialog box was successfully displayed and the user chose the OK button. InsertObjectDialog returns false otherwise. If the user chose OK and there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See Also

CreateObjectFromInfo (see page 17939)

ObjectPropertiesDialog (see page 17945)

PasteSpecialDialog (see page 17946)

115.1.1.4.20 TOleContainer.KeyDown Method

Respond to key press events.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-down message (WM_KEYDOWN) from Windows, its message handler calls the DoKeyDown method. If DoKeyDown determines that the control should, in fact, process the character, it calls KeyDown, passing the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

See Also

KeyUp (see page 6164)

OnKeyDown (see page 6225)

DoKeyDown (see page 6145)

DoKeyPress (see page 6146)

DoKeyUp (see page 6146)

115.1.1.4.21 TOleContainer.LoadFromFile Method

Loads an OLE object from the specified file.

Pascal

```
procedure LoadFromFile(const FileName: string);
```

C++

```
__fastcall LoadFromFile(const AnsiString FileName);
```

Description

Call LoadFromFile to load an OLE object that was saved using the SaveToFile (see page 17948) method. If OldStreamFormat (see page 17966) is true, LoadFromFile loads files saved by an TOleContainer (see page 17921) object as well as files saved using the current format; if OldStreamFormat (see page 17966) is false, LoadFromFile will not load files saved by the library. If there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See Also

LoadFromStream (see page 17944)

OldStreamFormat (see page 17966)

SaveToFile (see page 17948)

115.1.1.4.22 TOleContainer.LoadFromStream Method

Loads an OLE object from the specified stream.

Pascal

```
procedure LoadFromStream(Stream: TStream);
```

C++

```
__fastcall LoadFromStream(TStream Stream);
```

Description

Call LoadFromStream to load an OLE object from a stream. If OldStreamFormat (see page 17966) is true, LoadFromStream loads OLE objects saved by a TOleContainer (see page 17921) object as well as OLE objects saved using the current format; if OldStreamFormat (see page 17966) is false, LoadFromStream will not load OLE objects saved by the library. If there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See Also

LoadFromFile (see page 17944)

OldStreamFormat (see page 17966)

SaveToStream (see page 17949)

TStream (see page 2973)

115.1.1.4.23 TOleContainer.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove (see page 5773)

MouseUp (see page 5774)

OnMouseDown (see page 5862)

115.1.1.4.24 TOleContainer.ObjectPropertiesDialog Method

Displays the OLE Object Properties property sheet.

Pascal

```
function ObjectPropertiesDialog: Boolean;
```

C++

```
__fastcall Boolean ObjectPropertiesDialog();
```

Description

Call ObjectPropertiesDialog to let the user see and edit the properties of the OLE object. Any changes the user makes in the property sheet are automatically made to the OLE object. ObjectPropertiesDialog returns true if the property sheet was successfully displayed and the user chose the OK button. It returns false otherwise.

If the OLE container is empty, ObjectPropertiesDialog raises an exception.

See Also

DoVerb (see page 17941)

115.1.1.4.25 TOleContainer.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas (see page 5921)

PaintWindow (see page 5920)

GetDeviceContext (see page 6155)

ClipRect (see page 14486)

115.1.1.4.26 TOleContainer.Paste Method

Pastes the contents of the Windows clipboard as an embedded object.

Pascal

```
procedure Paste;
```

C++

```
__fastcall Paste();
```

Description

Call Paste to embed the object on the clipboard in the OLE container. If there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See Also

CanPaste (see page 17957)

CreateObjectFromInfo (see page 17939)

PasteSpecialDialog (see page 17946)

115.1.1.4.27 TOleContainer.PasteSpecialDialog Method

Displays the Paste (see page 17946) Special OLE dialog box.

Pascal

```
function PasteSpecialDialog: Boolean;
```

C++

```
__fastcall Boolean PasteSpecialDialog();
```

Description

Call PasteSpecialDialog to let the user control how the contents of the Windows clipboard are pasted into the container. The

Paste (see page 17946) Special dialog box lets the user select the format of the data, whether it should be embedded or linked, and whether the OLE object should be displayed as an icon (and if so, to choose a different icon). PasteSpecialDialog returns true if the dialog box was successfully displayed and the user chose the OK button. It returns false otherwise. If the user chose the OK button when there's already an OLE object in the container, it is destroyed and any changes the user made to it are discarded.

See Also

CanPaste (see page 17957)

InsertObjectDialog (see page 17943)

Paste (see page 17946)

115.1.1.4.28 TOleContainer.QueryInterface Method

Returns a reference to a specified interface if the object supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the component supports the interface specified by IID, and if so, returns a reference to that interface as the Obj parameter. If the component does not support the interface, the Obj parameter returns nil (Delphi) or NULL (C++).

Note: For components that act as COM object wrapper, QueryInterface calls the QueryInterface method of the internal COM object.

See Also

_AddRef (see page 2769)

_Release (see page 2769)

QueryInterface

QueryInterface (see page 22398)

Invoke (see page 2781)

ComObject (see page 2793)

115.1.1.4.29 TOleContainer.Run Method

Runs the OLE object's server application, but doesn't open or activate the OLE object itself.

Pascal

```
procedure Run;
```

C++

```
__fastcall Run();
```

Description

Call Run to ensure that the server application is running. Once the server application is running, opening and activating the OLE object happens much more quickly. An OLE object must already be loaded in the container before calling Run, Otherwise, Run

raises an exception.

See Also

State (see page 17977)

115.1.1.4.30 TOleContainer.SaveAsDocument Method

Saves the OLE object to the specified file in native OLE Document format.

Pascal

```
procedure SaveAsDocument(const FileName: string);
```

C++

```
__fastcall SaveAsDocument(const AnsiString FileName);
```

Description

Call SaveAsDocument to save the OLE object in its native OLE Document format. An OLE object must already be loaded in the container before calling SaveAsDocument. Otherwise, SaveAsDocument raises an exception.

Note: Saving an OLE object to a file or a stream limits the ability of another application, such as Word, to open the file, even if it is saved with the correct extension. Saving to a file or a stream loses information necessary to open such a file. Use SaveAsDocument when needing to save files in a native OLE Document format.

See Also

SaveToFile (see page 17948)

SaveToStream (see page 17949)

115.1.1.4.31 TOleContainer.SaveToFile Method

Saves the OLE object to the specified file.

Pascal

```
procedure SaveToFile(const FileName: string);
```

C++

```
__fastcall SaveToFile(const AnsiString FileName);
```

Description

Call SaveToFile to save the OLE object. If CopyOnSave (see page 17961) is true, SaveToFile first creates a temporary OLE object that compresses redundant data, saving space in the file that is saved. If OldStreamFormat (see page 17966) is true, SaveToFile writes a file that can be read by a TOleContainer (see page 17921) component.

An OLE object must already be loaded in the container before calling SaveToFile. Otherwise, SaveToFile raises an exception.

See Also

Copy (see page 17937)

CopyOnSave (see page 17961)

LoadFromFile (see page 17944)

OldStreamFormat (see page 17966)

SaveToStream (see page 17949)

115.1.1.4.32 TOleContainer.SaveToStream Method

Saves the OLE object to the specified stream.

Pascal

```
procedure SaveToStream(Stream: TStream);
```

C++

```
__fastcall SaveToStream(TStream Stream);
```

Description

Call SaveToStream to save the OLE object. If CopyOnSave (see page 17961) is true, SaveToStream first creates a temporary OLE object that compresses redundant data, reducing the information that must be saved. If OldStreamFormat (see page 17966) is true, SaveToStream writes the object in a format that can be read by a TOleContainer (see page 17921) component.

An OLE object must already be loaded in the container before calling SaveToStream. Otherwise, SaveToStream raises an exception.

See Also

Copy (see page 17937)

CopyOnSave (see page 17961)

LoadFromStream (see page 17944)

OldStreamFormat (see page 17966)

SaveToFile (see page 17948)

115.1.1.4.33 TOleContainer.UpdateObject Method

Refreshes the OLE object.

Pascal

```
procedure UpdateObject;
```

C++

```
__fastcall UpdateObject();
```

Description

Linked (see page 17965) OLE objects and embedded objects that contain linked OLE objects get out of date when the source of the link is updated. UpdateObject rereads the source to ensure that the OLE object has current data. If there is no OLE object loaded in the container, UpdateObject has no effect.

See Also

Linked (see page 17965)

UpdateVerbs (see page 17949)

115.1.1.4.34 TOleContainer.UpdateVerbs Method

Refreshes the list of verbs the OLE object can respond to.

Pascal

```
procedure UpdateVerbs;
```

C++

```
__fastcall UpdateVerbs();
```

Description

TOleContainer (see page 17921) automatically calls UpdateVerbs when the ObjectVerbs (see page 17966) property is first read, but some OLE objects change their verbs when they perform some actions. The Media Clip OLE object, for example, changes its Play verb to Stop while running. Call UpdateVerbs to ensure that the list of verbs is appropriate to the state of the OLE object.

An OLE object must already be loaded in the container before calling UpdateVerbs. Otherwise, UpdateVerbs raises an exception.

See Also

AutoVerbMenu (see page 17956)

ObjectVerbs (see page 17966)

115.1.1.5 TOleContainer Properties

The properties of the TOleContainer class are listed here.

Public Properties

	Name	Description
R	CanPaste (see page 17957)	Indicates whether the Windows clipboard contains an OLE object that can be pasted into the OLE container.
R	Linked (see page 17965)	Indicates whether the OLE object is linked or embedded.
	Modified (see page 17965)	Indicates whether the OLE object has been modified.
R	NewInserted (see page 17966)	Indicates whether the OLE object was created rather than pasted in or created as a link to a file.
R	ObjectVerbs (see page 17966)	Lists the verbs that the OLE object supports.
R	OleClassName (see page 17967)	Specifies the class name of the OLE object.
R	OleObject (see page 17967)	Represents the OLE object hosted by the OLE container.
R	OleObjectInterface (see page 17968)	Provides access to the IOleObject interface for the OLE object.
R	PrimaryVerb (see page 17974)	Specifies the index of the primary (default) verb for the OLE object.
R	SourceDoc (see page 17977)	Specifies the name of the source document for a linked OLE object.
R	State (see page 17977)	Describes the state of the OLE object.
R	StorageInterface (see page 17977)	Provides access to the IStorage interface for the OLE object.

Published Properties

	Name	Description
	Align (see page 17951)	Determines how the control aligns within its container (parent control).
	AllowActiveDoc (see page 17954)	Specifies whether the OLE container supports the IOleDocumentSite interface.
	AllowInPlace (see page 17955)	Determines whether the OLE container allows in-place activation.
	Anchors (see page 17955)	Specifies how the control is anchored to its parent.
	AutoActivate (see page 17956)	Determines how the object in an OLE container is activated.
	AutoVerbMenu (see page 17956)	Determines whether the OLE container automatically creates a pop-up menu containing the OLE object's verbs.
	BorderStyle (see page 17957)	Determines whether a single line is drawn around the OLE container.
	Caption (see page 17957)	Specifies a text string that identifies the control to the user.
	Color (see page 17960)	Specifies the background color of the control.
	Constraints (see page 17961)	Specifies the size constraints for the control.
	CopyOnSave (see page 17961)	Determines whether the SaveToFile (see page 17948) and SaveToStream (see page 17949) methods first create a copy of the OLE object.
	Ctl3D (see page 17962)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

	DragCursor (see page 17962)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 17963)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 17964)	Controls whether the control responds to mouse, keyboard, and timer events.
	Iconic (see page 17965)	Determines whether the OLE object is displayed as an icon.
	OldStreamFormat (see page 17966)	Indicates whether the OLE object is loaded and saved using the VCL format.
	ParentColor (see page 17968)	Determines where a control looks for its color information.
	ParentCtl3D (see page 17969)	Determines where a component looks to determine if it should appear three dimensional.
	ParentShowHint (see page 17969)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 17969)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 17974)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SizeMode (see page 17976)	Determines how the OLE object is sized in the OLE container.
	TabOrder (see page 17978)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 17978)	Determines if the user can tab to a control.
	Visible (see page 17979)	Determines whether the component appears onscreen.

115.1.1.5.1 TOleContainer.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
```

```
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
```

```
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
```

```
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

115.1.1.5.2 TOleContainer.AllowActiveDoc Property

Specifies whether the OLE container supports the IOleDocumentSite interface.

Pascal

```
property AllowActiveDoc: Boolean;
```

C++

```
__property Boolean AllowActiveDoc;
```

Description

Set AllowActiveDoc to true to specify that the OLE container can house an ActiveDoc (DocObject). Set AllowActiveDoc to false to limit the OLE container to hosting OLE controls that use the more usual interfaces such as IOleClientSite. ActiveDoc-enabled controls have a slightly different border. Set AllowActiveDoc to false to maintain a consistent appearance in forms that do not need to support the IOleDocumentSite interface.

See Also

AllowInPlace (see page 17955)

115.1.1.5.3 TOleContainer.AllowInPlace Property

Determines whether the OLE container allows in-place activation.

Pascal

```
property AllowInPlace: Boolean;
```

C++

```
__property Boolean AllowInPlace;
```

Description

Set AllowInPlace to specify what happens when the user activates the OLE object. If AllowInPlace is true and Iconic (see page 17965) is false, the OLE object is activated in place. If AllowInPlace is false, the OLE object is activated in a separate window. The default value is true.

See Also

AllowActiveDoc (see page 17954)

AutoActivate (see page 17956)

Iconic (see page 17965)

115.1.1.5.4 TOleContainer.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

115.1.1.5.5 TOleContainer.AutoActivate Property

Determines how the object in an OLE container is activated.

Pascal

```
property AutoActivate: TAutoActivate;
```

C++

```
__property TAutoActivate AutoActivate;
```

Description

Set AutoActivate to specify what actions activate the OLE object. AutoActivate takes one of the values in the following table:

Values	Meaning
aaManual	The OLE object must be activated programmatically by calling DoVerb (see page 17941) (ovShow (see page 18019)).
aaGetFocus	The OLE object is activated when the OLE container gets focus (by clicking on it with the mouse or pressing Tab to move the focus to it).
aaDoubleClick	(Default) The OLE object is activated by double-clicking it or pressing Enter while the container has the focus.

See Also

AllowActiveDoc (see page 17954)

DoVerb (see page 17941)

115.1.1.5.6 TOleContainer.AutoVerbMenu Property

Determines whether the OLE container automatically creates a pop-up menu containing the OLE object's verbs.

Pascal

```
property AutoVerbMenu: Boolean;
```

C++

```
__property Boolean AutoVerbMenu;
```

Description

Use AutoVerbMenu to specify whether the OLE object can replace the OLE container's popup menu. If AutoVerbMenu is true (the default), the OLE object replaces any existing PopupMenu (see page 17969) property. If AutoVerbMenu is false, no popup menu is automatically created.

See Also

DoVerb (see page 17941)

ObjectVerbs (see page 17966)

PopupMenu (see page 5835)

115.1.1.5.7 TOleContainer.BorderStyle Property

Determines whether a single line is drawn around the OLE container.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Set BorderStyle to specify whether the OLE container should be framed by a single line. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

See Also

Ctl3D (see page 6198)

115.1.1.5.8 TOleContainer.CanPaste Property

Indicates whether the Windows clipboard contains an OLE object that can be pasted into the OLE container.

Pascal

```
property CanPaste: Boolean;
```

C++

```
__property Boolean CanPaste;
```

Description

Read CanPaste before trying to paste or paste link an OLE object from the clipboard.

See Also

Paste (see page 17946)

PasteSpecialDialog (see page 17946)

115.1.1.5.9 TOleContainer.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of

character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
```

```
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
```

```
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

115.1.1.5.10 TOleContainer.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

115.1.1.5.11 TOleContainer.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

115.1.1.5.12 TOleContainer.CopyOnSave Property

Determines whether the SaveToFile (see page 17948) and SaveToStream (see page 17949) methods first create a copy of the OLE object.

Pascal

```
property CopyOnSave: Boolean;
```

C++

```
__property Boolean CopyOnSave;
```

Description

When CopyOnSave is true, the SaveToFile (see page 17948) and SaveToStream (see page 17949) methods create a

temporary OLE object, compressing redundant data to save space. This compressed copy is then saved to the file or stream. When CopyOnSave is false, the OLE object is saved directly to the file or stream without using a temporary copy to compress redundant information.

Set CopyOnSave to false when the container holds very large embedded objects and there might not be enough memory to make the temporary copy. The default value is true.

See Also

OleObject (see page 17967)

SaveToFile (see page 17948)

SaveToStream (see page 17949)

115.1.1.5.13 TOleContainer.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

115.1.1.5.14 TOleContainer.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

115.1.1.5.15 TOleContainer.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

115.1.1.5.16 TOleContainer.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

115.1.1.5.17 TOleContainer.Iconic Property

Determines whether the OLE object is displayed as an icon.

Pascal

```
property Iconic: Boolean;
```

C++

```
__property Boolean Iconic;
```

Description

Set Iconic to specify whether the OLE container displays the icon for the server application instead of the image of the object. If Iconic is true, the OLE object is displayed as an icon; if false, it's displayed as it would be in the server application.

See Also

GetIconMetaPict (see page 17942)

115.1.1.5.18 TOleContainer.Linked Property

Indicates whether the OLE object is linked or embedded.

Pascal

```
property Linked: Boolean;
```

C++

```
__property Boolean Linked;
```

Description

Read Linked to determine whether the OLE object is linked to the OLE server. If Linked is true, the OLE object is linked; if false, it's embedded. An OLE object must already be loaded in the container before accessing the Linked property.

See Also

UpdateObject (see page 17949)

115.1.1.5.19 TOleContainer.Modified Property

Indicates whether the OLE object has been modified.

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

Use Modified to determine when the OLE object has been modified (including being deleted or replaced by another OLE object). To keep track of incremental changes, set Modified to false and later check it to see if the OLE object has been modified after since Modified was set to false. If there is no OLE object loaded into the container, Modified returns false.

See Also

OleObject (see page 17967)

115.1.1.5.20 TOleContainer.NewInserted Property

Indicates whether the OLE object was created rather than pasted in or created as a link to a file.

Pascal

```
property NewInserted: Boolean;
```

C++

```
__property Boolean NewInserted;
```

Description

Read NewInserted to determine whether the OLE object was newly created via a call to InsertObjectDialog (see page 17943). If there is no OLE object loaded into the container, NewInserted returns false. If NewInserted is true after a call to InsertObjectDialog (see page 17943), call DoVerb (see page 17941)(ovShow (see page 18019)) to let the user edit the object.

See Also

DoVerb (see page 17941)

InsertObjectDialog (see page 17943)

OleObject (see page 17967)

115.1.1.5.21 TOleContainer.ObjectVerbs Property

Lists the verbs that the OLE object supports.

Pascal

```
property ObjectVerbs: TStrings;
```

C++

```
__property TStrings ObjectVerbs;
```

Description

Read ObjectVerbs to determine the verbs that can be used with the OLE object. The verb names can have embedded ampersand (&) characters to indicate shortcut keys. An OLE object must already be loaded in the container before accessing the ObjectVerbs property.

See Also

AutoVerbMenu (see page 17956)

DoVerb (see page 17941)

PrimaryVerb (see page 17974)

TStrings (see page 3029)

115.1.1.5.22 TOleContainer.OldStreamFormat Property

Indicates whether the OLE object is loaded and saved using the VCL format.

Pascal

```
property OldStreamFormat: Boolean;
```

C++

```
__property Boolean OldStreamFormat;
```

Description

Set OldStreamFormat to true to read OLE objects that were saved to file using VCL or to save OLE objects that will be used by VCL. If OldStreamFormat is false (the default), the LoadFromFile (see page 17944), LoadFromStream (see page 17944), SaveToFile (see page 17948), and SaveToStream (see page 17949) methods use the newer format that contains extra header information. The extra header information is used to validate a file or stream. This property is provided for backward compatibility only.

See Also

LoadFromFile (see page 17944)

LoadFromStream (see page 17944)

SaveToFile (see page 17948)

SaveToStream (see page 17949)

115.1.1.5.23 TOleContainer.OleClassName Property

Specifies the class name of the OLE object.

Pascal

```
property OleClassName: string;
```

C++

```
__property AnsiString OleClassName;
```

Description

Read OleClassName to obtain the class name (programmatic identifier) for the OLE object. The class name is a string describing the class of OLE object that is registered in the Windows registry. An OLE object must already be loaded in the container before accessing the OleClassName property.

See Also

OleObject (see page 17967)

115.1.1.5.24 TOleContainer.OleObject Property

Represents the OLE object hosted by the OLE container.

Pascal

```
property OleObject: Variant;
```

C++

```
__property Variant OleObject;
```

Description

Read OleObject to access the OLE object hosted by the OLE container as a variant. To interact with the OLE object using the IOleObject interface, use the OleObjectInterface (see page 17968) property instead. If the OLE object is not loaded in the container before reading OleObject, an EOleError exception is raised.

See Also

EOleError (see page 5528)

OleObjectInterface (see page 17968)

115.1.1.5.25 TOleContainer.OleObjectInterface Property

Provides access to the IOleObject interface for the OLE object.

Pascal

```
property OleObjectInterface: IOleObject;
```

C++

```
__property IOleObject OleObjectInterface;
```

Description

Use OleObjectInterface for low-level access to the OLE API. If there is no OLE object loaded into the container, OleObjectInterface returns nil (Delphi) or NULL (C++).

See Also

OleObject (see page 17967)

Creating Clients for Servers That Do Not Have a Type Library

115.1.1.5.26 TOleContainer.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

115.1.1.5.27 TOleContainer.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

115.1.1.5.28 TOleContainer.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

115.1.1.5.29 TOleContainer.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
```

```
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
```

```
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
```

```
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
```

```
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

115.1.1.5.30 TOleContainer.PrimaryVerb Property

Specifies the index of the primary (default) verb for the OLE object.

Pascal

```
property PrimaryVerb: Integer;
```

C++

```
__property int PrimaryVerb;
```

Description

Use PrimaryVerb to determine which verb in the ObjectVerbs (see page 17966) list specifies the default action for the OLE object. In a call to the DoVerb (see page 17941) method, DoVerb (see page 17941)(ovPrimary (see page 18018)) is the same as DoVerb (see page 17941)(PrimaryVerb). An OLE object must already be loaded in the container before accessing the PrimaryVerb property.

See Also

DoVerb (see page 17941)

ObjectVerbs (see page 17966)

115.1.1.5.31 TOleContainer.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
```

```
    Edit1.Hint := 'Enter your name';
    Edit1.ShowHint := True;
    with ListBox1 do
    begin
      for I := 1 to 10 do
        Items.Add('Item ' + IntToStr(I));
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

115.1.1.5.32 TOleContainer.SizeMode Property

Determines how the OLE object is sized in the OLE container.

Pascal

```
property SizeMode: TSizeMode;
```

C++

```
__property TSizeMode SizeMode;
```

Description

Set SizeMode to specify how the OLE object is fit into the client area of the OLE container. The following table gives the possible values :

Value	Meaning
smClip	(Default)- Displays the OLE object at its normal size, clipping any parts that don't fit within the container.
smCenter	Displays the OLE object at its normal size, centering it within the container.
smScale	Scales or shrinks the view of the OLE object to fit within the container, by scaling width and height proportionally.
smStretch	Scales or shrinks the view of the OLE object to fill the OLE container, without regard to preserving the proportions of the OLE object.
smAutoSize	Displays the OLE object at its normal size and automatically resizes the container to fit the size of the OLE object.

See Also

AllowInPlace (see page 17955)

ClientRect

115.1.1.5.33 TOleContainer.SourceDoc Property

Specifies the name of the source document for a linked OLE object.

Pascal

```
property SourceDoc: string;
```

C++

```
__property AnsiString SourceDoc;
```

Description

Read SourceDoc to obtain the name of the OLE object when it is linked. An OLE object must already be loaded in the container before accessing the SourceDoc property. If the OLE object isn't linked, SourceDoc returns an empty string.

See Also

Linked (see page 17965)

OleClassName (see page 17967)

115.1.1.5.34 TOleContainer.State Property

Describes the state of the OLE object.

Pascal

```
property State: TObjectState;
```

C++

```
__property TObjectState State;
```

Description

Read State to determine the state of the OLE object and its server application. State has one of the following values:

Values	Meaning
osEmpty	There is no OLE object in the container.
osLoaded	There is an OLE object in the container, but its server application isn't currently running.
osRunning	The OLE object's server is running.
osOpen	The OLE object is open in a separate window.
osInPlaceActive	The OLE object is activated in place, but hasn't yet merged its menus or toolbars. This is an intermediate state; the State property will shift to osUIActive as soon as the menus and toolbars have been merged.
osUIActive	The OLE object is activated in place and menus and toolbars have been merged

See Also

AllowInPlace (see page 17955)

Iconic (see page 17965)

115.1.1.5.35 TOleContainer.StorageInterface Property

Provides access to the IStorage interface for the OLE object.

Pascal

```
property StorageInterface: IStorage;
```

C++

```
__property IStorage StorageInterface;
```

Description

Use StorageInterface for low-level access to the OLE API. If there is no OLE object loaded into the container, StorageInterface returns nil (Delphi) or NULL (C++).

See Also

OleObjectInterface (see page 17968)

115.1.1.5.36 TOleContainer.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

115.1.1.5.37 TOleContainer.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

115.1.1.5.38 TOleContainer.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
```

```
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

115.1.1.6 TOleContainer Events

The events of the TOleContainer class are listed here.

Published Events

	Name	Description
	OnActivate (see page 17981)	Occurs immediately after the OLE object is activated.
	OnDeactivate (see page 17981)	Occurs immediately after the OLE object is deactivated.
	OnDragDrop (see page 17982)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 17984)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 17986)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

	OnEnter (see page 17987)	Occurs when a control receives the input focus.
	OnExit (see page 17990)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 17991)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 17994)	Occurs when a key is pressed.
	OnKeyUp (see page 17995)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 17997)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 17998)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 18000)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 18002)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnObjectMove (see page 18005)	Occurs when the OLE container is notified that the OLE object has moved or resized.
	OnResize (see page 18005)	Occurs immediately after the OLE container window is resized.
	OnStartDrag (see page 18005)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

115.1.1.6.1 TOleContainer.OnActivate Event

Occurs immediately after the OLE object is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to take specific action when the OLE object becomes active.

See Also

AutoActivate (see page 17956)

DoVerb (see page 17941)

OnDeactivate (see page 17981)

State (see page 17977)

115.1.1.6.2 TOleContainer.OnDeactivate Event

Occurs immediately after the OLE object is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to respond when the OLE object is deactivated.

See Also

Close (see page 17937)

OnActivate (see page 17981)

State (see page 17977)

115.1.1.6.3 TOleContainer.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
```

```
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

115.1.1.6.4 TOleContainer.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}
```

```
/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
```

```
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

115.1.1.6.5 TOleContainer.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
```

```
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

115.1.1.6.6 TOleContainer.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
```

```
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
```

```
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

115.1.1.6.7 TOleContainer.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
```

```
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

115.1.1.6.8 TOleContainer.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
```

```
  event to FormKeyUp.  Set the KeyPreview on the form to True.
  }
  var
    FormColor: TColor;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    KeyPreview := True;
    FormColor := Form1.Color;
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
  {
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
```

```
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

115.1.1.6.9 TOleContainer.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not

generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

115.1.1.6.10 TOleContainer.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
```

```
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
```

```
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

115.1.1.6.11 TOleContainer.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate

event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

115.1.1.6.12 TOleContainer.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
```

```
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
```

```
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

115.1.1.6.13 TOleContainer.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
```

```
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
```

```
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

115.1.1.6.14 TOleContainer.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area

of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

115.1.1.6.15 TOleContainer.OnObjectMove Event

Occurs when the OLE container is notified that the OLE object has moved or resized.

Pascal

```
property OnObjectMove: TObjectMoveEvent;
```

C++

```
__property TObjectMoveEvent OnObjectMove;
```

Description

Write an OnObjectMove event handler to respond when the OLE object changes size or position. The Bounds parameter specifies the boundaries of the in-place object in the client coordinates of OLE container. OnObjectMove occurs before the OLE object's image is updated to reflect the change.

See Also

OnResize (see page 18005)

115.1.1.6.16 TOleContainer.OnResize Event

Occurs immediately after the OLE container window is resized.

Pascal

```
property OnResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnResize;
```

Description

Write an OnResize event handler to respond to changes in the size of the OLE container window. If SizeMode (see page 17976) is smAutoSize, the container window changes size whenever the OLE object resizes.

See Also

OnObjectMove (see page 18005)

SizeMode (see page 17976)

115.1.1.6.17 TOleContainer.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

115.1.2 TOleForm Class

TOleForm is an object that provides OLE support for a TForm object.

Class Hierarchy

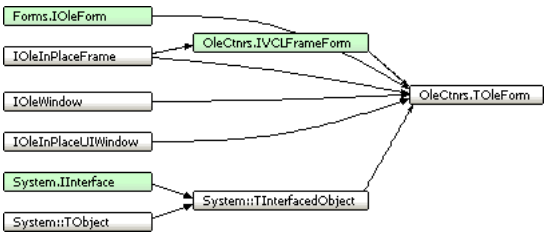

Pascal

```
TOleForm = class(TInterfacedObject, IOleForm, IOleWindow, IOleInPlaceUIWindow,
IOleInPlaceFrame, IVCLFrameForm);
```

C++

```
class TOleForm : public TInterfacedObject, public IOleForm, public IOleWindow, public
IOleInPlaceUIWindow, public IOleInPlaceFrame, public IVCLFrameForm;
```

File

OleCtnrs

Description

TOleForm implements the IOleForm, IOleWindow, IOleInPlaceUIWindow, and IOleInPlaceFrame interfaces, as well as an interface for internal use. The implementation of the TOleForm object is used internally by the Visual Component Library.

115.1.2.1 TOleForm Members

The following tables list the members exposed by TOleForm.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddContainer (see page 18010)	This is AddContainer, a member of class IVCLFrameForm.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearBorderSpace (see page 18010)	This is ClearBorderSpace, a member of class IVCLFrameForm.
	Create (see page 18008)	Create creates a new instance of the TOleForm (see page 18006) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 18008)	Destroy frees the memory associated with this instance of the TOleForm (see page 18006) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Form (see page 18010)	This is Form, a member of class IVCLFrameForm.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	OnDestroy (see page 14340)	This is OnDestroy, a member of class IOleForm.
	OnResize (see page 14341)	This is OnResize, a member of class IOleForm.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RemoveContainer (see page 18010)	This is RemoveContainer, a member of class IVCLFrameForm.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

115.1.2.2 TOleForm.Create Constructor

Create creates a new instance of the TOleForm (see page 18006) object.

Pascal

```
constructor Create(Form: TCustomForm);
```

C++

```
__fastcall TOleForm(TCustomForm Form);
```

Description

Create is used to create a new instance of the TOleForm (see page 18006) object. Form (see page 18010) is the form upon which to base the new TOleForm (see page 18006) object.

See Also

Destroy (see page 18008)

115.1.2.3 TOleForm.Destroy Destructor

Destroy frees the memory associated with this instance of the TOleForm (see page 18006) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TOleForm();
```

Description

Do not call Destroy directly. Instead, call Free, which calls Destroy if this instance of the TOleForm (see page 18006) object is not nil.

Destroy frees private objects allocated and used by this instance of the TOleForm (see page 18006) object.

See Also

Create (see page 18008)

115.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IVCLFrameForm (see page 18009)	This is class OleCtnrs.IVCLFrameForm.

115.2.1 IVCLFrameForm Interface

Class Hierarchy

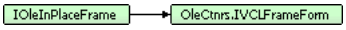

Pascal

```
IVCLFrameForm = interface(IOleInPlaceFrame);
```

C++

```
__interface IVCLFrameForm : public IOleInPlaceFrame;
```

File

OleCtnrs

Description

This is class OleCtnrs.IVCLFrameForm.

115.2.1.1 IVCLFrameForm Members

The following tables list the members exposed by IVCLFrameForm.

Public Methods

	Name	Description
	AddContainer (see page 18010)	This is AddContainer, a member of class IVCLFrameForm.
	ClearBorderSpace (see page 18010)	This is ClearBorderSpace, a member of class IVCLFrameForm.
	Form (see page 18010)	This is Form, a member of class IVCLFrameForm.
	RemoveContainer (see page 18010)	This is RemoveContainer, a member of class IVCLFrameForm.

115.2.1.2 IVCLFrameForm Methods

The methods of the IVCLFrameForm class are listed here.

Public Methods

	Name	Description
	AddContainer (see page 18010)	This is AddContainer, a member of class IVCLFrameForm.
	ClearBorderSpace (see page 18010)	This is ClearBorderSpace, a member of class IVCLFrameForm.
	Form (see page 18010)	This is Form, a member of class IVCLFrameForm.

	RemoveContainer (see page 18010)	This is RemoveContainer, a member of class IVCLFrameForm.

115.2.1.2.1 IVCLFrameForm.AddContainer Method

Pascal

```
procedure AddContainer(Instance: TOleContainer);
```

C++

```
__fastcall AddContainer(TOleContainer Instance);
```

Description

This is AddContainer, a member of class IVCLFrameForm.

115.2.1.2.2 IVCLFrameForm.ClearBorderSpace Method

Pascal

```
procedure ClearBorderSpace;
```

C++

```
__fastcall ClearBorderSpace();
```

Description

This is ClearBorderSpace, a member of class IVCLFrameForm.

115.2.1.2.3 IVCLFrameForm.Form Method

Pascal

```
function Form: TCustomForm;
```

C++

```
__fastcall TCustomForm Form();
```

Description

This is Form, a member of class IVCLFrameForm.

115.2.1.2.4 IVCLFrameForm.RemoveContainer Method

Pascal

```
procedure RemoveContainer(Instance: TOleContainer);
```

C++

```
__fastcall RemoveContainer(TOleContainer Instance);
```

Description

This is RemoveContainer, a member of class IVCLFrameForm.

115.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DestroyMetaPict (see page 18011)	Frees a metafile resource given its handle.

115.3.1 OleCtnrs.DestroyMetaPict Function

Frees a metafile resource given its handle.

Pascal

```
procedure DestroyMetaPict(MetaPict: HGlobal);
```

C++

```
DestroyMetaPict(HGlobal MetaPict);
```

File

OleCtnrs

Description

Call DestroyMetaPict to free a metafile obtained using the GetIconMetaPict method of TOleContainer (see page 17921).

You do not need to use DestroyMetaPict to free metafiles represented using TMetafile. These are freed automatically when the TMetafile instance is destroyed. However, if you assign a new value to the Handle property of TMetafile, thereby replacing its old metafile, you must free the old metafile by calling DestroyMetaPict.

See Also

TOleContainer (see page 17921)

TMetafile

TMetafile

115.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAutoActivate (see page 18012)	This is record OleCtnrs.TAutoActivate.
	TCreateType (see page 18013)	This is record OleCtnrs.TCreateType.
	TObjectState (see page 18013)	This is record OleCtnrs.TObjectState.
	TSizeMode (see page 18014)	This is record OleCtnrs.TSizeMode.

Records

	Name	Description
	TCreateInfo (see page 18012)	This is record OleCtnrs.TCreateInfo.
	TVerbInfo (see page 18014)	This is record OleCtnrs.TVerbInfo.

115.4.1 OleCtnrs.TAutoActivate Enumeration

Pascal

```
TAutoActivate = (
  aaManual,
  aaGetFocus,
  aaDoubleClick
);
```

C++

```
enum TAutoActivate {
  aaManual,
  aaGetFocus,
  aaDoubleClick
};
```

File

OleCtnrs

Description

This is record OleCtnrs.TAutoActivate.

115.4.2 OleCtnrs.TCreateInfo Record

Pascal

```
TCreateInfo = record
  CreateType: TCreateType;
  ShowAsIcon: Boolean;
  IconMetaPict: HGlobal;
  ClassID: TCLSID;
  FileName: WideString;
```

```
    DataObject: IDataObject;
  end;
```

C++

```
  struct TCreateInfo {
    TCreateType CreateType;
    Boolean ShowAsIcon;
    HGlobal IconMetaPict;
    TCLSID ClassID;
    BSTR FileName;
    IDataObject DataObject;
  };
```

File

OleCtnrs

Description

This is record OleCtnrs.TCreateInfo.

115.4.3 OleCtnrs.TCreateType Enumeration

Pascal

```
  TCreateType = (
    ctNewObject,
    ctFromFile,
    ctLinkToFile,
    ctFromData,
    ctLinkFromData
  );
```

C++

```
  enum TCreateType {
    ctNewObject,
    ctFromFile,
    ctLinkToFile,
    ctFromData,
    ctLinkFromData
  };
```

File

OleCtnrs

Description

This is record OleCtnrs.TCreateType.

115.4.4 OleCtnrs.TObjectState Enumeration

Pascal

```
  TObjectState = (
    osEmpty,
    osLoaded,
    osRunning,
    osOpen,
    osInPlaceActive,
```

```
    osUIActive
  );
```

C++

```
  enum TObjectState {
    osEmpty,
    osLoaded,
    osRunning,
    osOpen,
    osInPlaceActive,
    osUIActive
  };
```

File

OleCtnrs

Description

This is record OleCtnrs.TObjectState.

115.4.5 OleCtnrs.TSizeMode Enumeration

Pascal

```
  TSizeMode = (
    smClip,
    smCenter,
    smScale,
    smStretch,
    smAutoSize
  );
```

C++

```
  enum TSizeMode {
    smClip,
    smCenter,
    smScale,
    smStretch,
    smAutoSize
  };
```

File

OleCtnrs

Description

This is record OleCtnrs.TSizeMode.

115.4.6 OleCtnrs.TVerbInfo Record

Pascal

```
  TVerbInfo = packed record
    Verb: Smallint;
    Flags: Word;
  end;
```

C++

```
struct TVerbInfo {
  Smallint Verb;
  Word Flags;
};
```

File

OleCtnrs

Description

This is record OleCtnrs.TVerbInfo.

115.5 Types

The following table lists types in this documentation.

Types

Name	Description
TObjectMoveEvent (see page 18016)	This is type OleCtnrs.TObjectMoveEvent.

115.5.1 OleCtnrs.TObjectMoveEvent Type

Pascal

```
TObjectMoveEvent = procedure (OleContainer: TOleContainer; const Bounds: TRect) of object;
```

C++

```
(OleContainer: TOleContainer; const Bounds: TRect) ( TObjectMoveEvent)();
```

File

OleCtnrs

Description

This is type OleCtnrs.TObjectMoveEvent.

115.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
ovDiscardUndoState (see page 18017)	This is constant OleCtnrs.ovDiscardUndoState.
ovHide (see page 18017)	This is constant OleCtnrs.ovHide.
ovInPlaceActivate (see page 18018)	This is constant OleCtnrs.ovInPlaceActivate.
ovOpen (see page 18018)	This is constant OleCtnrs.ovOpen.
ovPrimary (see page 18018)	This is constant OleCtnrs.ovPrimary.
ovShow (see page 18019)	This is constant OleCtnrs.ovShow.
ovUIActivate (see page 18019)	This is constant OleCtnrs.ovUIActivate.

115.6.1 OleCtnrs.ovDiscardUndoState Constant

Pascal

```
ovDiscardUndoState = -6;
```

C++

```
ovDiscardUndoState = -6;
```

File

OleCtnrs

Description

This is constant OleCtnrs.ovDiscardUndoState.

115.6.2 OleCtnrs.ovHide Constant

Pascal

```
ovHide = -3;
```

C++

```
ovHide = -3;
```

File

OleCtnrs

Description

This is constant OleCtnrs.ovHide.

115.6.3 OleCtnrs.ovInPlaceActivate Constant

Pascal

```
ovInPlaceActivate = -5;
```

C++

```
ovInPlaceActivate = -5;
```

File

OleCtnrs

Description

This is constant OleCtnrs.ovInPlaceActivate.

115.6.4 OleCtnrs.ovOpen Constant

Pascal

```
ovOpen = -2;
```

C++

```
ovOpen = -2;
```

File

OleCtnrs

Description

This is constant OleCtnrs.ovOpen.

115.6.5 OleCtnrs.ovPrimary Constant

Pascal

```
ovPrimary = -65536;
```

C++

```
ovPrimary = -65536;
```

File

OleCtnrs

Description

This is constant OleCtnrs.ovPrimary.

115.6.6 OleCtnrs.ovShow Constant

Pascal

```
ovShow = -1;
```

C++

```
ovShow = -1;
```

File

OleCtnrs

Description

This is constant OleCtnrs.ovShow.

115.6.7 OleCtnrs.ovUIActivate Constant

Pascal

```
ovUIActivate = -4;
```

C++

```
ovUIActivate = -4;
```

File

OleCtnrs

Description

This is constant OleCtnrs.ovUIActivate.

116 OleCtrls Namespace

This is namespace OleCtrls.

Topics

Name	Description
Classes (see page 18023)	The following table lists classes in this documentation.
Functions (see page 18088)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 18090)	The following table lists structs, records, enums in this documentation.
Types (see page 18093)	The following table lists types in this documentation.
Variables (see page 18096)	The following table lists variables in this documentation.

Classes

	Name	Description
	EOleCtrlError (see page 18023)	EOleCtrlError is the exception class for ActiveX controls.
	TEnumPropDesc (see page 18023)	TEnumPropDesc provides the mapping between strings used to name the values of a property in an OLE control object and the values themselves.
	TEventDispatch (see page 18026)	This is class OleCtrls.TEventDispatch.
	TOleControl (see page 18030)	TOleControl is the base class from which ActiveX controls are derived.

Functions

	Name	Description
	FontToOleFont (see page 18088)	Returns a Variant containing an IFontDispatch interface that represents a TFont object.
	OleFontToFont (see page 18088)	Fills in a TFont structure to represent an IFontDispatch.

Structs, Records, Enums

	Name	Description
	TControlData (see page 18090)	This is record OleCtrls.TControlData.
	TControlData2 (see page 18091)	This is record OleCtrls.TControlData2.
	TEnumValue (see page 18091)	This is record OleCtrls.TEnumValue.

Types

Name	Description
PControlData (see page 18093)	This is type OleCtrls.PControlData.
PControlData2 (see page 18093)	This is type OleCtrls.PControlData2.
PEnumValueList (see page 18094)	This is type OleCtrls.PEnumValueList.
TEnumValueList (see page 18094)	This is type OleCtrls.TEnumValueList.

TOleEnum (see page 18094)	This is type OleCtrls.TOleEnum.
TServiceQuery (see page 18095)	This is type OleCtrls.TServiceQuery.

Variables

Name	Description
EmptyParam (see page 18096)	This is variable OleCtrls.EmptyParam.

116.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EOleCtrlError (see page 18023)	EOleCtrlError is the exception class for ActiveX controls.
	TEnumPropDesc (see page 18023)	TEnumPropDesc provides the mapping between strings used to name the values of a property in an OLE control object and the values themselves.
	TEventDispatch (see page 18026)	This is class OleCtrls.TEventDispatch.
	TOleControl (see page 18030)	TOleControl is the base class from which ActiveX controls are derived.

116.1.1 EOleCtrlError Class

EOleCtrlError is the exception class for ActiveX controls.

Class Hierarchy

System::TObject → SysUtils.Exception → OleCtrls.EOleCtrlError

Pascal

```
EOleCtrlError = class(Exception);
```

C++

```
class EOleCtrlError : public Exception;
```

File

OleCtrls

Description

EOleCtrlError is raised when an application cannot link to an ActiveX control.

See Also

EOleError (see page 5528)

116.1.2 TEnumPropDesc Class

TEnumPropDesc provides the mapping between strings used to name the values of a property in an OLE control object and the values themselves.

Class Hierarchy

OleCtrls.TEnumPropDesc

Pascal

```
TEnumPropDesc = class;
```

C++

```
class TEnumPropDesc;
```

File

OleCtrls

Description

Use TEnumPropDesc to interpret the strings used by an OLE control object to represent the values of one of its properties. TEnumPropDesc can provide a list of possible value strings to a callback function, or map between the string names of property values and the values themselves. TEnumPropDesc objects are made available through the GetEnumPropDesc method of an OLE control object.

See Also

TOleControl (see page 18030)

116.1.2.1 TEnumPropDesc Members

The following tables list the members exposed by TEnumPropDesc.

Public Methods

	Name	Description
	Create (see page 18024)	Creates an instance of TEnumPropDesc (see page 18023).
	Destroy (see page 18025)	Destroys an instance of TEnumPropDesc (see page 18023).
	GetStrings (see page 18025)	Calls the callback Proc for each property value string.
	StringToValue (see page 18026)	Maps a string representation of the property value to the integer value.
	ValueToString (see page 18026)	Maps an integer that indicates the value of the property to the string representation of that value used by the OLE control object.

116.1.2.2 TEnumPropDesc.Create Constructor

Creates an instance of TEnumPropDesc (see page 18023).

Pascal

```
constructor Create(DispID: Integer; ValueCount: Integer; const TypeInfo: ITypeInfo);
```

C++

```
__fastcall TEnumPropDesc(int DispID, int ValueCount, const ITypeInfo TypeInfo);
```

Description

Create is called by an OLE control object to create an instance of TEnumPropDesc (see page 18023) to represent the values of one of its properties. Applications should not need to create instances of TEnumPropDesc (see page 18023). When working with OLE properties, get the TEnumPropDesc (see page 18023) objects from the OLE control object.

DispID identifies the property of the OLE object, ValueCount is the number of distinct values the property can take TypeInfo is an ITypeInfo object that the constructor can use to capture the values and their strings.

See Also

GetEnumPropDesc (see page 18054)

116.1.2.3 TEnumPropDesc.Destroy Destructor

Destroys an instance of TEnumPropDesc (see page 18023).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TEnumPropDesc();
```

Description

Do not call Destroy directly in an application. The TOleControl (see page 18030) object that owns the TEnumPropDesc (see page 18023) object will destroy it automatically.

Destroy frees the internal list used to map value strings to property values.

See Also

TOleControl (see page 18030)

Free (see page 22300)

116.1.2.4 TEnumPropDesc Methods

The methods of the TEnumPropDesc class are listed here.

Public Methods

	Name	Description
	GetStrings (see page 18025)	Calls the callback Proc for each property value string.
	StringToValue (see page 18026)	Maps a string representation of the property value to the integer value.
	ValueToString (see page 18026)	Maps an integer that indicates the value of the property to the string representation of that value used by the OLE control object.

116.1.2.4.1 TEnumPropDesc.GetStrings Method

Calls the callback Proc for each property value string.

Pascal

```
procedure GetStrings(Proc: TGetStrProc);
```

C++

```
__fastcall GetStrings(TGetStrProc Proc);
```

Description

Use GetStrings to execute code for every property value string represented in the TEnumPropDesc (see page 18023) object. The callback Proc is called for every value, with the S parameter set to a string that gives the integer value followed by the string representation of that value.

For example, consider a TEnumPropDesc (see page 18023) object that represents a color property, with values equal to TColor values. The callback Proc would be called for every built-in color name with strings of the form '255 - clRed'.

See Also

StringToValue (see page 18026)

ValueToString (see page 18026)

116.1.2.4.2 TEnumPropDesc.StringToValue Method

Maps a string representation of the property value to the integer value.

Pascal

```
function StringToValue(const S: string): Integer;
```

C++

```
__fastcall int StringToValue(const AnsiString S);
```

Description

Use StringToValue to map the string representation of a property value used by an OLE control object to an integer value that is a more accurate representation of the property value. If the property value is not an integer, the integer associated with the string can be a pointer to the value.

See Also

ValueToString (see page 18026)

116.1.2.4.3 TEnumPropDesc.ValueToString Method

Maps an integer that indicates the value of the property to the string representation of that value used by the OLE control object.

Pascal

```
function ValueToString(V: Integer): string;
```

C++

```
__fastcall AnsiString ValueToString(int V);
```

Description

Use ValueToString to map the integer value of a property value to the string representation used by an OLE control object. If the property value is not an integer, the integer associated with the string can be a pointer to the value.

See Also

StringToValue (see page 18026)

116.1.3 TEventDispatch Class

Class Hierarchy

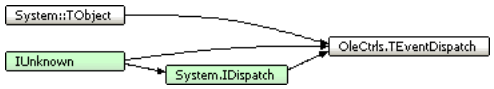

Pascal

```
TEventDispatch = class(TObject, IUnknown, IDispatch);
```

C++

```
class TEventDispatch : public TObject, public IUnknown, public IDispatch;
```

File

OleCtrls

Description

This is class OleCtrls.TEventDispatch.

116.1.3.1 TEventDispatch Members

The following tables list the members exposed by TEventDispatch.

Protected Methods

	Name	Description
	_AddRef (see page 18028)	This is _AddRef, a member of class TEventDispatch.
	_Release (see page 18028)	This is _Release, a member of class TEventDispatch.
	GetIDsOfNames (see page 18029)	This is GetIDsOfNames, a member of class TEventDispatch.
	GetTypeInfo (see page 18029)	This is GetTypeInfo, a member of class TEventDispatch.
	GetTypeInfoCount (see page 18029)	This is GetTypeInfoCount, a member of class TEventDispatch.
V	Invoke (see page 18029)	This is Invoke, a member of class TEventDispatch.
	QueryInterface (see page 18030)	This is QueryInterface, a member of class TEventDispatch.

Protected Properties

	Name	Description
R	Control (see page 18030)	This is Control, a member of class TEventDispatch.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 18028)	This is Create, a member of class TEventDispatch.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

116.1.3.2 TEventDispatch.Create Constructor

Pascal

```
constructor Create(Control: TOleControl);
```

C++

```
__fastcall TEventDispatch(TOleControl Control);
```

Description

This is Create, a member of class TEventDispatch.

116.1.3.3 TEventDispatch Methods

The methods of the TEventDispatch class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 18028)	This is _AddRef, a member of class TEventDispatch.
	_Release (see page 18028)	This is _Release, a member of class TEventDispatch.
	GetIDsOfNames (see page 18029)	This is GetIDsOfNames, a member of class TEventDispatch.
	GetTypeInfo (see page 18029)	This is GetTypeInfo, a member of class TEventDispatch.
	GetTypeInfoCount (see page 18029)	This is GetTypeInfoCount, a member of class TEventDispatch.
	Invoke (see page 18029)	This is Invoke, a member of class TEventDispatch.
	QueryInterface (see page 18030)	This is QueryInterface, a member of class TEventDispatch.

116.1.3.3.1 TEventDispatch._AddRef Method

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

This is _AddRef, a member of class TEventDispatch.

116.1.3.3.2 TEventDispatch._Release Method

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

This is _Release, a member of class TEventDispatch.

116.1.3.3.3 TEventDispatch.GetIDsOfNames Method

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int NameCount, int
LocaleID, void * DispIDs);
```

Description

This is GetIDsOfNames, a member of class TEventDispatch.

116.1.3.3.4 TEventDispatch.GetTypeInfo Method

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

This is GetTypeInfo, a member of class TEventDispatch.

116.1.3.3.5 TEventDispatch.GetTypeInfoCount Method

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

This is GetTypeInfoCount, a member of class TEventDispatch.

116.1.3.3.6 TEventDispatch.Invoke Method

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word
Flags,  Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is Invoke, a member of class TEventDispatch.

116.1.3.3.7 TEventDispatch.QueryInterface Method

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

This is QueryInterface, a member of class TEventDispatch.

116.1.3.4 TEventDispatch Properties

The properties of the TEventDispatch class are listed here.

Protected Properties

	Name	Description
	Control (see page 18030)	This is Control, a member of class TEventDispatch.

116.1.3.4.1 TEventDispatch.Control Property

Pascal

```
property Control: TOleControl;
```

C++

```
__property TOleControl Control;
```

Description

This is Control, a member of class TEventDispatch.

116.1.4 TOleControl Class

TOleControl is the base class from which ActiveX controls are derived.

Class Hierarchy

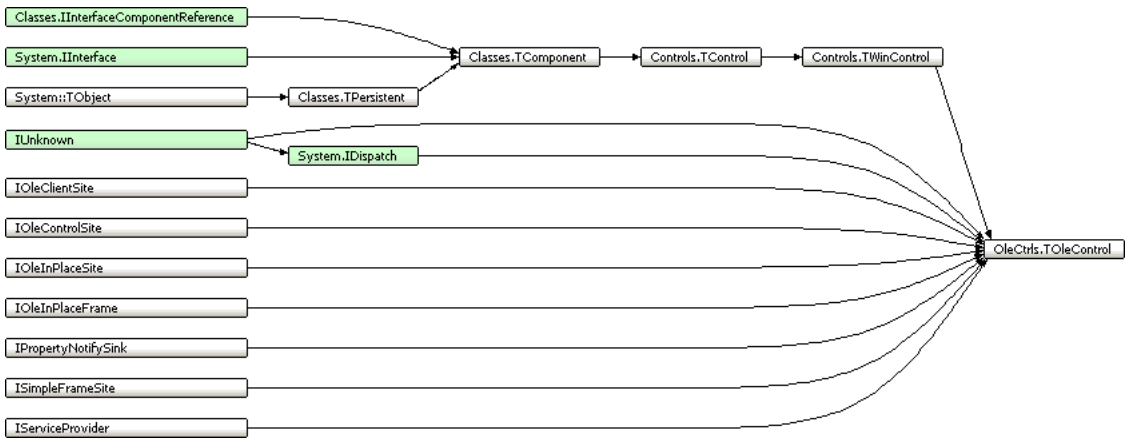

Pascal

```
TOleControl = class(TWinControl, IUnknown, IOleClientSite, IOleControlSite, IOleInPlaceSite,
IOleInPlaceFrame, IDispatch, IPropertyNotifySink, ISimpleFrameSite, IServiceProvider);
```

C++

```
class TOleControl : public TWinControl, public IUnknown, public IOleClientSite, public
IOleControlSite, public IOleInPlaceSite, public IOleInPlaceFrame, public IDispatch, public
IPropertyNotifySink, public ISimpleFrameSite, public IServiceProvider;
```

File

OleCtrls

Description

TOleControl is derived from TWinControl and handles the interactions with COM necessary for using ActiveX controls. Most application do not need to explicitly derive a component from this class.

See Also

TOleContainer (see page 17921)

Code Generated When You Import Type Library Information

Using Component Wrappers

116.1.4.1 TOleControl Members

The following tables list the members exposed by TOleControl.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.

	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 18046)	Called when an application uses a component interface.
	_Release (see page 18047)	Called when an application releases a component interface.
	_SetColorProp (see page 18047)	This is _SetColorProp, a member of class TOleControl.
	_SetTColorProp (see page 18048)	This is _SetTColorProp, a member of class TOleControl.
	_SetTFontProp (see page 18048)	This is _SetTFontProp, a member of class TOleControl.
	_SetTOleEnumProp (see page 18048)	This is _SetTOleEnumProp, a member of class TOleControl.
	_SetTPictureProp (see page 18048)	This is _SetTPictureProp, a member of class TOleControl.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.

	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanInPlaceActivate (see page 18049)	This is CanInPlaceActivate, a member of class TOleControl.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ContextSensitiveHelp (see page 18049)	This is ContextSensitiveHelp, a member of class TOleControl.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 18049)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DeactivateAndUndo (see page 18050)	This is DeactivateAndUndo, a member of class TOleControl.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 18050)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 18051)	Destroys the window created in the CreateWindowHandle method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DiscardUndoState (see page 18051)	This is DiscardUndoState, a member of class TOleControl.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.

	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EnableModeless (see page 18052)	This is EnableModeless, a member of class TOleControl.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorder (see page 18052)	This is GetBorder, a member of class TOleControl.
	GetByteProp (see page 18052)	This is GetByteProp, a member of class TOleControl.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetColorProp (see page 18053)	This is GetColorProp, a member of class TOleControl.
	GetCompProp (see page 18053)	This is GetCompProp, a member of class TOleControl.
	GetContainer (see page 18053)	This is GetContainer, a member of class TOleControl.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetCurrencyProp (see page 18053)	This is GetCurrencyProp, a member of class TOleControl.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDoubleProp (see page 18053)	This is GetDoubleProp, a member of class TOleControl.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetExtendedControl (see page 18054)	This is GetExtendedControl, a member of class TOleControl.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDispatchProp (see page 18055)	This is GetIDispatchProp, a member of class TOleControl.
	GetIDsOfNames (see page 18055)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIntegerProp (see page 18055)	This is GetIntegerProp, a member of class TOleControl.
	GetIUnknownProp (see page 18055)	This is GetIUnknownProp, a member of class TOleControl.
	GetMoniker (see page 18056)	This is GetMoniker, a member of class TOleControl.
	GetOleBoolProp (see page 18056)	This is GetOleBoolProp, a member of class TOleControl.
	GetOleDateProp (see page 18056)	This is GetOleDateProp, a member of class TOleControl.
	GetOleEnumProp (see page 18057)	This is GetOleEnumProp, a member of class TOleControl.
	GetOleVariantProp (see page 18057)	This is GetOleVariantProp, a member of class TOleControl.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.

	GetProperty (see page 18058)	This is GetProperty, a member of class TOleControl.
	GetShortIntProp (see page 18058)	This is GetShortIntProp, a member of class TOleControl.
	GetSingleProp (see page 18058)	This is GetSingleProp, a member of class TOleControl.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetSmallintProp (see page 18058)	This is GetSmallintProp, a member of class TOleControl.
	GetStringProp (see page 18059)	This is GetStringProp, a member of class TOleControl.
	GetTColorProp (see page 18059)	This is GetTColorProp, a member of class TOleControl.
	GetTDateTimeProp (see page 18059)	This is GetTDateTimeProp, a member of class TOleControl.
	GetTFontProp (see page 18059)	This is GetTFontProp, a member of class TOleControl.
	GetTOleEnumProp (see page 18059)	This is GetTOleEnumProp, a member of class TOleControl.
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTPictureProp (see page 18060)	This is GetTPictureProp, a member of class TOleControl.
	GetTypeInfo (see page 18060)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 18060)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariantProp (see page 18061)	This is GetVariantProp, a member of class TOleControl.
	GetWideStringProp (see page 18061)	This is GetWideStringProp, a member of class TOleControl.
	GetWindowContext (see page 18061)	This is GetWindowContext, a member of class TOleControl.
	GetWordBoolProp (see page 18061)	This is GetWordBoolProp, a member of class TOleControl.
	GetWordProp (see page 18062)	This is GetWordProp, a member of class TOleControl.
A	InitControlData (see page 18062)	This is InitControlData, a member of class TOleControl.
V	InitControlInterface (see page 18062)	This is InitControlInterface, a member of class TOleControl.
	InsertMenus (see page 18062)	This is InsertMenus, a member of class TOleControl.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 18062)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	InvokeEvent (see page 18063)	This is InvokeEvent, a member of class TOleControl.
	InvokeMethod (see page 18063)	This is InvokeMethod, a member of class TOleControl.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	LockInPlaceActive (see page 18064)	This is LockInPlaceActive, a member of class TOleControl.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	OleControlSite_TranslateAccelerator (see page 18064)	This is OleControlSite_TranslateAccelerator, a member of class TOleControl.
	OleInPlaceFrame_GetWindow (see page 18064)	This is OleInPlaceFrame_GetWindow, a member of class TOleControl.
	OleInPlaceFrame_TranslateAccelerator (see page 18065)	This is OleInPlaceFrame_TranslateAccelerator, a member of class TOleControl.
	OleInPlaceSite_GetWindow (see page 18065)	This is OleInPlaceSite_GetWindow, a member of class TOleControl.
V	OnChanged (see page 18065)	This is OnChanged, a member of class TOleControl.
	OnControlInfoChanged (see page 18065)	This is OnControlInfoChanged, a member of class TOleControl.
	OnFocus (see page 18066)	This is OnFocus, a member of class TOleControl.
	OnInPlaceActivate (see page 18066)	This is OnInPlaceActivate, a member of class TOleControl.

	OnInPlaceDeactivate (see page 18066)	This is OnInPlaceDeactivate, a member of class TOleControl.
	OnPosRectChange (see page 18066)	This is OnPosRectChange, a member of class TOleControl.
V	OnRequestEdit (see page 18066)	This is OnRequestEdit, a member of class TOleControl.
	OnShowWindow (see page 18067)	This is OnShowWindow, a member of class TOleControl.
	OnUIActivate (see page 18067)	This is OnUIActivate, a member of class TOleControl.
	OnUIDeactivate (see page 18067)	This is OnUIDeactivate, a member of class TOleControl.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
V	PaletteChanged (see page 18067)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PictureChanged (see page 18068)	This is PictureChanged, a member of class TOleControl.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	PostMessageFilter (see page 18068)	This is PostMessageFilter, a member of class TOleControl.
	PreMessageFilter (see page 18069)	This is PreMessageFilter, a member of class TOleControl.
V	QueryInterface (see page 18069)	Returns a reference to a specified interface if the object supports that interface.
	QueryService (see page 18069)	This is QueryService, a member of class TOleControl.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveMenus (see page 18070)	This is RemoveMenus, a member of class TOleControl.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	RequestBorderSpace (see page 18070)	This is RequestBorderSpace, a member of class TOleControl.
	RequestNewObjectLayout (see page 18070)	This is RequestNewObjectLayout, a member of class TOleControl.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	SaveObject (see page 18070)	This is SaveObject, a member of class TOleControl.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Scroll (see page 18070)	This is Scroll, a member of class TOleControl.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActiveObject (see page 18071)	This is SetActiveObject, a member of class TOleControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetBorderSpace (see page 18071)	This is SetBorderSpace, a member of class TOleControl.
	SetByteProp (see page 18071)	This is SetByteProp, a member of class TOleControl.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetColorProp (see page 18072)	This is SetColorProp, a member of class TOleControl.
	SetCompProp (see page 18072)	This is SetCompProp, a member of class TOleControl.
	SetCurrencyProp (see page 18072)	This is SetCurrencyProp, a member of class TOleControl.

	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDoubleProp (see page 18072)	This is SetDoubleProp, a member of class TOleControl.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIDispatchProp (see page 18073)	This is SetIDispatchProp, a member of class TOleControl.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetIntegerProp (see page 18073)	This is SetIntegerProp, a member of class TOleControl.
	SetIUnknownProp (see page 18073)	This is SetIUnknownProp, a member of class TOleControl.
	SetMenu (see page 18073)	This is SetMenu, a member of class TOleControl.
V	SetName (see page 18073)	Sets the name of the control.
	SetOleBoolProp (see page 18074)	This is SetOleBoolProp, a member of class TOleControl.
	SetOleDateProp (see page 18074)	This is SetOleDateProp, a member of class TOleControl.
	SetOleEnumProp (see page 18074)	This is SetOleEnumProp, a member of class TOleControl.
	SetOleVariantProp (see page 18075)	This is SetOleVariantProp, a member of class TOleControl.
V	SetParent (see page 18075)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetProperty (see page 18076)	This is SetProperty, a member of class TOleControl.
	SetShortIntProp (see page 18076)	This is SetShortIntProp, a member of class TOleControl.
	SetSingleProp (see page 18076)	This is SetSingleProp, a member of class TOleControl.
	SetSmallintProp (see page 18076)	This is SetSmallintProp, a member of class TOleControl.
	SetStatusText (see page 18077)	This is SetStatusText, a member of class TOleControl.
	SetStringProp (see page 18077)	This is SetStringProp, a member of class TOleControl.
	SetTColorProp (see page 18077)	This is SetTColorProp, a member of class TOleControl.
	SetTDateTimeProp (see page 18077)	This is SetTDateTimeProp, a member of class TOleControl.
	SetTFontProp (see page 18077)	This is SetTFontProp, a member of class TOleControl.
	SetTOleEnumProp (see page 18078)	This is SetTOleEnumProp, a member of class TOleControl.
	SetTPictureProp (see page 18078)	This is SetTPictureProp, a member of class TOleControl.
	SetVariantProp (see page 18078)	This is SetVariantProp, a member of class TOleControl.
	SetWideStringProp (see page 18078)	This is SetWideStringProp, a member of class TOleControl.
	SetWordBoolProp (see page 18079)	This is SetWordBoolProp, a member of class TOleControl.
	SetWordProp (see page 18079)	This is SetWordProp, a member of class TOleControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	ShowObject (see page 18079)	This is ShowObject, a member of class TOleControl.
	ShowPropertyFrame (see page 18079)	This is ShowPropertyFrame, a member of class TOleControl.
V	StandardEvent (see page 18080)	This is StandardEvent, a member of class TOleControl.
V	SuppressException (see page 18080)	This is SuppressException, a member of class TOleControl.
	TransformCoords (see page 18080)	This is TransformCoords, a member of class TOleControl.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 18080)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	ControlData (see page 18086)	This is ControlData, a member of class TOleControl.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.

	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	ServiceQuery (see page 18087)	This is ServiceQuery, a member of class TOleControl.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	BrowseProperties (see page 18048)	Notifies the ActiveX control to display its Properties dialog box.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 18043)	Creates a new instance of TOleControl (see page 18030).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 18050)	Handles all otherwise-unhandled messages.
	Destroy (see page 18043)	Frees the memory associated with this instance of TOleControl (see page 18030).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DoObjectVerb (see page 18051)	Notifies the ActiveX control to perform one of its actions.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetEnumPropDesc (see page 18054)	Returns descriptions of the ActiveX control's properties.
	GetHelpContext (see page 18054)	Retrieves the Help context of a member of the ActiveX control.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetObjectVerbs (see page 18056)	Retrieves the names of the verbs implemented by the ActiveX control.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetPropDisplayString (see page 18057)	Returns the display string for an interface for the ActiveX control.
	GetPropDisplayStrings (see page 18057)	Returns the display strings for an interface for the ActiveX control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsCustomProperty (see page 18063)	Indicates whether an interface for the ActiveX control is a custom property.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsPropPageProperty (see page 18064)	Indicates whether an interface for the OCX control is a property page property.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.

	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 18071)	Sets the boundaries of the ActiveX control.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetPropDisplayString (see page 18075)	Sets the display string for an interface for the ActiveX control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	ShowAboutBox (see page 18079)	Notifies the ActiveX control to display its About box.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 18085)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
R	DefaultDispatch (see page 18086)	The default dispatch interface.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	OleObject (see page 18086)	The ActiveX control associated with the TOleControl (see page 18030) object.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	PerPropBrowsing (see page 18086)	Provides access to information about the property page properties of the OleControl.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 18087)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

116.1.4.2 TOleControl.Create Constructor

Creates a new instance of TOleControl (see page 18030).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TOleControl(TComponent * AOwner);
```

Description

Use Create to create a new instance of the TOleControl (see page 18030) object. Create calls the inherited Create method and creates and initializes internal objects.

See Also

Destroy (see page 18043)

Create (see page 6123)

116.1.4.3 TOleControl.Destroy Destructor

Frees the memory associated with this instance of TOleControl (see page 18030).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TOleControl();
```

Description

Do not call Destroy. Instead, call Free, which calls Destroy if this instance of TOleControl (see page 18030) is not nil.

See Also

Create (see page 18043)

Free (see page 22300)

116.1.4.4 TOleControl Methods

The methods of the TOleControl class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 18046)	Called when an application uses a component interface.
	_Release (see page 18047)	Called when an application releases a component interface.
	_SetColorProp (see page 18047)	This is _SetColorProp, a member of class TOleControl.
	_SetTColorProp (see page 18048)	This is _SetTColorProp, a member of class TOleControl.
	_SetTFontProp (see page 18048)	This is _SetTFontProp, a member of class TOleControl.
	_SetTOleEnumProp (see page 18048)	This is _SetTOleEnumProp, a member of class TOleControl.
	_SetTPictureProp (see page 18048)	This is _SetTPictureProp, a member of class TOleControl.
	CanInPlaceActivate (see page 18049)	This is CanInPlaceActivate, a member of class TOleControl.
	ContextSensitiveHelp (see page 18049)	This is ContextSensitiveHelp, a member of class TOleControl.
V	CreateWnd (see page 18049)	Creates the underlying window.
	DeactivateAndUndo (see page 18050)	This is DeactivateAndUndo, a member of class TOleControl.
V	DefineProperties (see page 18050)	Provides methods to read and write the IsControl property to a stream such as a form file.
V	DestroyWindowHandle (see page 18051)	Destroys the window created in the CreateWindowHandle method.
	DiscardUndoState (see page 18051)	This is DiscardUndoState, a member of class TOleControl.
	EnableModeless (see page 18052)	This is EnableModeless, a member of class TOleControl.
	GetBorder (see page 18052)	This is GetBorder, a member of class TOleControl.
	GetByteProp (see page 18052)	This is GetByteProp, a member of class TOleControl.
	GetColorProp (see page 18053)	This is GetColorProp, a member of class TOleControl.
	GetCompProp (see page 18053)	This is GetCompProp, a member of class TOleControl.
	GetContainer (see page 18053)	This is GetContainer, a member of class TOleControl.
	GetCurrencyProp (see page 18053)	This is GetCurrencyProp, a member of class TOleControl.
	GetDoubleProp (see page 18053)	This is GetDoubleProp, a member of class TOleControl.
	GetExtendedControl (see page 18054)	This is GetExtendedControl, a member of class TOleControl.
	GetIDispatchProp (see page 18055)	This is GetIDispatchProp, a member of class TOleControl.
	GetIDsOfNames (see page 18055)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIntegerProp (see page 18055)	This is GetIntegerProp, a member of class TOleControl.
	GetIUnknownProp (see page 18055)	This is GetIUnknownProp, a member of class TOleControl.
	GetMoniker (see page 18056)	This is GetMoniker, a member of class TOleControl.
	GetOleBoolProp (see page 18056)	This is GetOleBoolProp, a member of class TOleControl.
	GetOleDateProp (see page 18056)	This is GetOleDateProp, a member of class TOleControl.
	GetOleEnumProp (see page 18057)	This is GetOleEnumProp, a member of class TOleControl.
	GetOleVariantProp (see page 18057)	This is GetOleVariantProp, a member of class TOleControl.
	GetProperty (see page 18058)	This is GetProperty, a member of class TOleControl.
	GetShortIntProp (see page 18058)	This is GetShortIntProp, a member of class TOleControl.
	GetSingleProp (see page 18058)	This is GetSingleProp, a member of class TOleControl.
	GetSmallintProp (see page 18058)	This is GetSmallintProp, a member of class TOleControl.
	GetStringProp (see page 18059)	This is GetStringProp, a member of class TOleControl.
	GetTColorProp (see page 18059)	This is GetTColorProp, a member of class TOleControl.
	GetTDateTimeProp (see page 18059)	This is GetTDateTimeProp, a member of class TOleControl.
	GetTFontProp (see page 18059)	This is GetTFontProp, a member of class TOleControl.

	GetTOleEnumProp (see page 18059)	This is GetTOleEnumProp, a member of class TOleControl.
	GetTPictureProp (see page 18060)	This is GetTPictureProp, a member of class TOleControl.
	GetTypeInfo (see page 18060)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 18060)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariantProp (see page 18061)	This is GetVariantProp, a member of class TOleControl.
	GetWideStringProp (see page 18061)	This is GetWideStringProp, a member of class TOleControl.
	GetWindowContext (see page 18061)	This is GetWindowContext, a member of class TOleControl.
	GetWordBoolProp (see page 18061)	This is GetWordBoolProp, a member of class TOleControl.
	GetWordProp (see page 18062)	This is GetWordProp, a member of class TOleControl.
A	InitControlData (see page 18062)	This is InitControlData, a member of class TOleControl.
V	InitControlInterface (see page 18062)	This is InitControlInterface, a member of class TOleControl.
	InsertMenus (see page 18062)	This is InsertMenus, a member of class TOleControl.
	Invoke (see page 18062)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	InvokeEvent (see page 18063)	This is InvokeEvent, a member of class TOleControl.
	InvokeMethod (see page 18063)	This is InvokeMethod, a member of class TOleControl.
	LockInPlaceActive (see page 18064)	This is LockInPlaceActive, a member of class TOleControl.
	OleControlSite_TranslateAccelerator (see page 18064)	This is OleControlSite_TranslateAccelerator, a member of class TOleControl.
	OleInPlaceFrame_GetWindow (see page 18064)	This is OleInPlaceFrame_GetWindow, a member of class TOleControl.
	OleInPlaceFrame_TranslateAccelerator (see page 18065)	This is OleInPlaceFrame_TranslateAccelerator, a member of class TOleControl.
	OleInPlaceSite_GetWindow (see page 18065)	This is OleInPlaceSite_GetWindow, a member of class TOleControl.
V	OnChanged (see page 18065)	This is OnChanged, a member of class TOleControl.
	OnControlInfoChanged (see page 18065)	This is OnControlInfoChanged, a member of class TOleControl.
	OnFocus (see page 18066)	This is OnFocus, a member of class TOleControl.
	OnInPlaceActivate (see page 18066)	This is OnInPlaceActivate, a member of class TOleControl.
	OnInPlaceDeactivate (see page 18066)	This is OnInPlaceDeactivate, a member of class TOleControl.
	OnPosRectChange (see page 18066)	This is OnPosRectChange, a member of class TOleControl.
V	OnRequestEdit (see page 18066)	This is OnRequestEdit, a member of class TOleControl.
	OnShowWindow (see page 18067)	This is OnShowWindow, a member of class TOleControl.
	OnUIActivate (see page 18067)	This is OnUIActivate, a member of class TOleControl.
	OnUIDeactivate (see page 18067)	This is OnUIDeactivate, a member of class TOleControl.
V	PaletteChanged (see page 18067)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PictureChanged (see page 18068)	This is PictureChanged, a member of class TOleControl.
	PostMessageFilter (see page 18068)	This is PostMessageFilter, a member of class TOleControl.
	PreMessageFilter (see page 18069)	This is PreMessageFilter, a member of class TOleControl.
V	QueryInterface (see page 18069)	Returns a reference to a specified interface if the object supports that interface.
	QueryService (see page 18069)	This is QueryService, a member of class TOleControl.
	RemoveMenus (see page 18070)	This is RemoveMenus, a member of class TOleControl.
	RequestBorderSpace (see page 18070)	This is RequestBorderSpace, a member of class TOleControl.
	RequestNewObjectLayout (see page 18070)	This is RequestNewObjectLayout, a member of class TOleControl.
	SaveObject (see page 18070)	This is SaveObject, a member of class TOleControl.
	Scroll (see page 18070)	This is Scroll, a member of class TOleControl.
	SetActiveObject (see page 18071)	This is SetActiveObject, a member of class TOleControl.
	SetBorderSpace (see page 18071)	This is SetBorderSpace, a member of class TOleControl.
	SetByteProp (see page 18071)	This is SetByteProp, a member of class TOleControl.
	SetColorProp (see page 18072)	This is SetColorProp, a member of class TOleControl.
	SetCompProp (see page 18072)	This is SetCompProp, a member of class TOleControl.
	SetCurrencyProp (see page 18072)	This is SetCurrencyProp, a member of class TOleControl.
	SetDoubleProp (see page 18072)	This is SetDoubleProp, a member of class TOleControl.
	SetIDispatchProp (see page 18073)	This is SetIDispatchProp, a member of class TOleControl.
	SetIntegerProp (see page 18073)	This is SetIntegerProp, a member of class TOleControl.
	SetIUnknownProp (see page 18073)	This is SetIUnknownProp, a member of class TOleControl.
	SetMenu (see page 18073)	This is SetMenu, a member of class TOleControl.

	SetName (see page 18073)	Sets the name of the control.
	SetOleBoolProp (see page 18074)	This is SetOleBoolProp, a member of class TOleControl.
	SetOleDateProp (see page 18074)	This is SetOleDateProp, a member of class TOleControl.
	SetOleEnumProp (see page 18074)	This is SetOleEnumProp, a member of class TOleControl.
	SetOleVariantProp (see page 18075)	This is SetOleVariantProp, a member of class TOleControl.
	SetParent (see page 18075)	Sets the parent of the control.
	SetProperty (see page 18076)	This is SetProperty, a member of class TOleControl.
	SetShortIntProp (see page 18076)	This is SetShortIntProp, a member of class TOleControl.
	SetSingleProp (see page 18076)	This is SetSingleProp, a member of class TOleControl.
	SetSmallintProp (see page 18076)	This is SetSmallintProp, a member of class TOleControl.
	SetStatusText (see page 18077)	This is SetStatusText, a member of class TOleControl.
	SetStringProp (see page 18077)	This is SetStringProp, a member of class TOleControl.
	SetTColorProp (see page 18077)	This is SetTColorProp, a member of class TOleControl.
	SetTDateTimeProp (see page 18077)	This is SetTDateTimeProp, a member of class TOleControl.
	SetTFontProp (see page 18077)	This is SetTFontProp, a member of class TOleControl.
	SetTOleEnumProp (see page 18078)	This is SetTOleEnumProp, a member of class TOleControl.
	SetTPictureProp (see page 18078)	This is SetTPictureProp, a member of class TOleControl.
	SetVariantProp (see page 18078)	This is SetVariantProp, a member of class TOleControl.
	SetWideStringProp (see page 18078)	This is SetWideStringProp, a member of class TOleControl.
	SetWordBoolProp (see page 18079)	This is SetWordBoolProp, a member of class TOleControl.
	SetWordProp (see page 18079)	This is SetWordProp, a member of class TOleControl.
	ShowObject (see page 18079)	This is ShowObject, a member of class TOleControl.
	ShowPropertyFrame (see page 18079)	This is ShowPropertyFrame, a member of class TOleControl.
	StandardEvent (see page 18080)	This is StandardEvent, a member of class TOleControl.
	SuppressException (see page 18080)	This is SuppressException, a member of class TOleControl.
	TransformCoords (see page 18080)	This is TransformCoords, a member of class TOleControl.
	WndProc (see page 18080)	Provides specific message responses for the control.

Public Methods

	Name	Description
	BrowseProperties (see page 18048)	Notifies the ActiveX control to display its Properties dialog box.
	DefaultHandler (see page 18050)	Handles all otherwise-unhandled messages.
	DoObjectVerb (see page 18051)	Notifies the ActiveX control to perform one of its actions.
	GetEnumPropDesc (see page 18054)	Returns descriptions of the ActiveX control's properties.
	GetHelpContext (see page 18054)	Retrieves the Help context of a member of the ActiveX control.
	GetObjectVerbs (see page 18056)	Retrieves the names of the verbs implemented by the ActiveX control.
	GetPropDisplayString (see page 18057)	Returns the display string for an interface for the ActiveX control.
	GetPropDisplayStrings (see page 18057)	Returns the display strings for an interface for the ActiveX control.
	IsCustomProperty (see page 18063)	Indicates whether an interface for the ActiveX control is a custom property.
	IsPropPageProperty (see page 18064)	Indicates whether an interface for the OCX control is a property page property.
	SetBounds (see page 18071)	Sets the boundaries of the ActiveX control.
	SetPropDisplayString (see page 18075)	Sets the display string for an interface for the ActiveX control.
	ShowAboutBox (see page 18079)	Notifies the ActiveX control to display its About box.

116.1.4.4.1 TOleControl._AddRef Method

Called when an application uses a component interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef is a basic implementation of the IInterface method, _AddRef.

Note: If the component is a wrapper for a COM object, __AddRef calls the _AddRef method of that COM object, and returns the resulting reference count.

In all other cases, __AddRef simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __AddRef to implement reference counting.

See Also

Invoke (see page 2781)

QueryInterface (see page 2784)

_Release (see page 2769)

ComObject (see page 2793)

_AddRef (see page 22397)

116.1.4.4.2 TOleControl._Release Method

Called when an application releases a component interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release is a basic implementation of the IInterface method, _Release.

__Release returns the resulting value of the reference count for the component's interface.

Note:

In all other cases, __Release simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __Release to implement reference counting.

See Also

Invoke (see page 2781)

QueryInterface (see page 2784)

_AddRef (see page 2769)

ComObject (see page 2793)

116.1.4.4.3 TOleControl._SetColorProp Method

Pascal

```
procedure _SetColorProp(Index: Integer; Value: TColor);
```

C++

```
__fastcall _SetColorProp(int Index, TColor Value);
```

Description

This is _SetColorProp, a member of class TOleControl.

116.1.4.4.4 TOleControl._SetTColorProp Method

Pascal

```
procedure _SetTColorProp(Index: Integer; Value: TColor);
```

C++

```
__fastcall _SetTColorProp(int Index, TColor Value);
```

Description

This is _SetTColorProp, a member of class TOleControl.

116.1.4.4.5 TOleControl._SetTFontProp Method

Pascal

```
procedure _SetTFontProp(Index: Integer; Value: TFont);
```

C++

```
__fastcall _SetTFontProp(int Index, TFont Value);
```

Description

This is _SetTFontProp, a member of class TOleControl.

116.1.4.4.6 TOleControl._SetTOleEnumProp Method

Pascal

```
procedure _SetTOleEnumProp(Index: Integer; Value: TOleEnum);
```

C++

```
__fastcall _SetTOleEnumProp(int Index, TOleEnum Value);
```

Description

This is _SetTOleEnumProp, a member of class TOleControl.

116.1.4.4.7 TOleControl._SetTPictureProp Method

Pascal

```
procedure _SetTPictureProp(Index: Integer; Value: TPicture);
```

C++

```
__fastcall _SetTPictureProp(int Index, TPicture Value);
```

Description

This is _SetTPictureProp, a member of class TOleControl.

116.1.4.4.8 TOleControl.BrowseProperties Method

Notifies the ActiveX control to display its Properties dialog box.

Pascal

```
procedure BrowseProperties;
```

C++

```
__fastcall BrowseProperties();
```

Description

Use BrowseProperties to notify the ActiveX control to display its Properties dialog box. The Properties dialog box contains property pages for the ActiveX control.

116.1.4.4.9 TOleControl.CanInPlaceActivate Method

Pascal

```
function CanInPlaceActivate: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT CanInPlaceActivate();
```

Description

This is CanInPlaceActivate, a member of class TOleControl.

116.1.4.4.10 TOleControl.ContextSensitiveHelp Method

Pascal

```
function ContextSensitiveHelp(fEnterMode: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT ContextSensitiveHelp(BOOL fEnterMode);
```

Description

This is ContextSensitiveHelp, a member of class TOleControl.

116.1.4.4.11 TOleControl.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams (see page 6135)

CreateWindowHandle (see page 6137)

Perform (see page 5776)

DestroyWnd (see page 6141)

116.1.4.4.12 TOleControl.DeactivateAndUndo Method

Pascal

```
function DeactivateAndUndo: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT DeactivateAndUndo();
```

Description

This is DeactivateAndUndo, a member of class TOleControl.

116.1.4.4.13 TOleControl.DefaultHandler Method

Handles all otherwise-unhandled messages.

Pascal

```
procedure DefaultHandler(var Message); override;
```

C++

```
virtual __fastcall DefaultHandler( Message);
```

Description

DefaultHandler handles all messages not already handled in a control's WndProc (see page 18080) or message handling methods. The inherited method should always be called for unhandled messages.

See Also

DefaultHandler (see page 5748)

116.1.4.4.14 TOleControl.DefineProperties Method

Provides methods to read and write the IsControl property to a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

DefineProperties is called automatically by the streaming system that loads and saves VCL components. It allows persistent objects to read and write properties that are not published. Published properties are loaded and stored automatically.

DefineProperties uses the methods of the filer passed as the Filer parameter to assign methods that can load and save the IsControl property.

Override DefineProperties to persist unpublished properties. Call the DefineBinaryProperty or DefineProperty method of the Filer parameter to assign methods that load or save property values.

Note: When overriding this method, call the parent class's DefineProperties method first.

See Also

DefineBinaryProperty (see page 2837)

DefineProperty (see page 2838)

DefineProperties (see page 2771)

IsControl (see page 5829)

116.1.4.4.15 TOleControl.DestroyWindowHandle Method

Destroys the window created in the CreateWindowHandle method.

Pascal

```
procedure DestroyWindowHandle; override;
```

C++

```
virtual __fastcall DestroyWindowHandle();
```

Description

Call DestroyWindowHandle to dispose of the window for the control.

TWinControl's Destroy method calls DestroyWindowHandle to destroy any window associated with a windowed control before destroying the object. The DestroyWnd method also calls DestroyWindowHandle.

See Also

CreateWindowHandle (see page 6137)

Destroy (see page 6124)

DestroyWnd (see page 6141)

116.1.4.4.16 TOleControl.DiscardUndoState Method

Pascal

```
function DiscardUndoState: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT DiscardUndoState();
```

Description

This is DiscardUndoState, a member of class TOleControl.

116.1.4.4.17 TOleControl.DoObjectVerb Method

Notifies the ActiveX control to perform one of its actions.

Pascal

```
procedure DoObjectVerb(Verb: Integer);
```

C++

```
__fastcall DoObjectVerb(int Verb);
```

Description

Use DoObjectVerb to notify the ActiveX control to perform one of its actions. Verb is an integer that corresponds to the action. Most application-defined verbs are small positive integers.

Verb Name	Verb
OLEIVERB_PRIMARY	0
OLEIVERB_SHOW	–1
OLEIVERB_OPEN	–2
OLEIVERB_HIDE	–3
OLEIVERB_UIACTIVATE	–4
OLEIVERB_INPLACEACTIVATE	–5
OLEIVERB_DISCARDUNDOSTATE	–6
OLEIVERB_PROPERTIES	–7

For more information about verbs, search the OLE.HLP file or the latest Microsoft Developer Network CD.

116.1.4.4.18 TOleControl.EnableModeless Method

Pascal

```
function EnableModeless(fEnable: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT EnableModeless(BOOL fEnable);
```

Description

This is EnableModeless, a member of class TOleControl.

116.1.4.4.19 TOleControl.GetBorder Method

Pascal

```
function GetBorder(out rectBorder: TRect): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetBorder(TRect rectBorder);
```

Description

This is GetBorder, a member of class TOleControl.

116.1.4.4.20 TOleControl.GetByteProp Method

Pascal

```
function GetByteProp(Index: Integer): Byte;
```

C++

```
__fastcall Byte GetByteProp(int Index);
```

Description

This is GetByteProp, a member of class TOleControl.

116.1.4.4.21 TOleControl.GetColorProp Method

Pascal

```
function GetColorProp(Index: Integer): TColor;
```

C++

```
__fastcall TColor GetColorProp(int Index);
```

Description

This is GetColorProp, a member of class TOleControl.

116.1.4.4.22 TOleControl.GetCompProp Method

Pascal

```
function GetCompProp(Index: Integer): Comp;
```

C++

```
__fastcall Comp GetCompProp(int Index);
```

Description

This is GetCompProp, a member of class TOleControl.

116.1.4.4.23 TOleControl.GetContainer Method

Pascal

```
function GetContainer(out container: IOleContainer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetContainer(IOleContainer container);
```

Description

This is GetContainer, a member of class TOleControl.

116.1.4.4.24 TOleControl.GetCurrencyProp Method

Pascal

```
function GetCurrencyProp(Index: Integer): Currency;
```

C++

```
__fastcall Currency GetCurrencyProp(int Index);
```

Description

This is GetCurrencyProp, a member of class TOleControl.

116.1.4.4.25 TOleControl.GetDoubleProp Method

Pascal

```
function GetDoubleProp(Index: Integer): Double;
```

C++

```
__fastcall Double GetDoubleProp(int Index);
```

Description

This is GetDoubleProp, a member of class TOleControl.

116.1.4.4.26 TOleControl.GetEnumPropDesc Method

Returns descriptions of the ActiveX control's properties.

Pascal

```
function GetEnumPropDesc(DispID: Integer): TEnumPropDesc;
```

C++

```
__fastcall TEnumPropDesc GetEnumPropDesc(int DispID);
```

Description

Use GetEnumPropDesc to return descriptions of the ActiveX control's properties. GetEnumPropDesc returns the descriptions in a TEnumPropDesc (see page 18023) object.

116.1.4.4.27 TOleControl.GetExtendedControl Method

Pascal

```
function GetExtendedControl(out disp: IDispatch): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetExtendedControl(IDispatch disp);
```

Description

This is GetExtendedControl, a member of class TOleControl.

116.1.4.4.28 TOleControl.GetHelpContext Method

Retrieves the Help context of a member of the ActiveX control.

Pascal

```
function GetHelpContext(Member: string; var HelpCtx: Integer; var HelpFile: string): Boolean;
```

C++

```
__fastcall Boolean GetHelpContext(AnsiString Member, int HelpCtx, AnsiString HelpFile);
```

Description

Use GetHelpContext to retrieve the Help context of a member of the ActiveX control.

Member is the name of a property, method, or event of the ActiveX control. To retrieve the help context of the ActiveX control itself, set Member to nil (Delphi) or NULL (C++).

HelpCtx is a variable in which the Help context identifier of Member is to be stored.

HelpFile is a variable in which the name of the Help file containing HelpCtx is to be stored.

The information about the ActiveX control is retrieved from its ITypeInfo interfaces. For more information, search the OLE.HLP file for ITypeInfo.

GetHelpContext returns false if Member is not found; otherwise, GetHelpContext returns true.

116.1.4.4.29 TOleControl.GetIDispatchProp Method

Pascal

```
function GetIDispatchProp(Index: Integer): IDispatch;
```

C++

```
__fastcall IDispatch GetIDispatchProp(int Index);
```

Description

This is GetIDispatchProp, a member of class TOleControl.

116.1.4.4.30 TOleControl.GetIDsOfNames Method

Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int NameCount, int
LocaleID, void * DispIDs);
```

Description

GetIDsOfNames implements the IDispatch interface GetIDsOfNames method. For components that support interfaces, GetIDsOfNames calls this method for the interface supported by the component, passing the specified parameters. The returned value can be used on subsequent calls to the Invoke method.

See Also

GetTypeInfo (see page 2779)

GetTypeInfoCount (see page 2779)

Invoke (see page 2781)

116.1.4.4.31 TOleControl.GetIntegerProp Method

Pascal

```
function GetIntegerProp(Index: Integer): Integer;
```

C++

```
__fastcall int GetIntegerProp(int Index);
```

Description

This is GetIntegerProp, a member of class TOleControl.

116.1.4.4.32 TOleControl.GetIUnknownProp Method

Pascal

```
function GetIUnknownProp(Index: Integer): IUnknown;
```

C++

```
__fastcall IUnknown GetIUnknownProp(int Index);
```

Description

This is GetIUnknownProp, a member of class TOleControl.

116.1.4.4.33 TOleControl.GetMoniker Method

Pascal

```
function GetMoniker(dwAssign: Longint; dwWhichMoniker: Longint; out mk: IMoniker): HResult;
stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetMoniker(Longint dwAssign, Longint dwWhichMoniker, IMoniker mk);
```

Description

This is GetMoniker, a member of class TOleControl.

116.1.4.4.34 TOleControl.GetObjectVerbs Method

Retrieves the names of the verbs implemented by the ActiveX control.

Pascal

```
procedure GetObjectVerbs(List: TStrings);
```

C++

```
__fastcall GetObjectVerbs(TStrings List);
```

Description

Use GetObjectVerbs to retrieve the names of the verbs implemented by the ActiveX control.

List is where the names of the verbs are to be stored.

116.1.4.4.35 TOleControl.GetOleBoolProp Method

Pascal

```
function GetOleBoolProp(Index: Integer): TOleBool;
```

C++

```
__fastcall TOleBool GetOleBoolProp(int Index);
```

Description

This is GetOleBoolProp, a member of class TOleControl.

116.1.4.4.36 TOleControl.GetOleDateProp Method

Pascal

```
function GetOleDateProp(Index: Integer): TOleDate;
```

C++

```
__fastcall TOleDate GetOleDateProp(int Index);
```

Description

This is GetOleDateProp, a member of class TOleControl.

116.1.4.4.37 TOleControl.GetOleEnumProp Method

Pascal

```
function GetOleEnumProp(Index: Integer): TOleEnum;
```

C++

```
__fastcall TOleEnum GetOleEnumProp(int Index);
```

Description

This is GetOleEnumProp, a member of class TOleControl.

116.1.4.4.38 TOleControl.GetOleVariantProp Method

Pascal

```
function GetOleVariantProp(Index: Integer): OleVariant;
```

C++

```
__fastcall OleVariant GetOleVariantProp(int Index);
```

Description

This is GetOleVariantProp, a member of class TOleControl.

116.1.4.4.39 TOleControl.GetPropDisplayString Method

Returns the display string for an interface for the ActiveX control.

Pascal

```
function GetPropDisplayString(DispID: Integer): string;
```

C++

```
__fastcall AnsiString GetPropDisplayString(int DispID);
```

Description

GetPropDisplayString returns the display string for an interface for the ActiveX control.

DispID is the identifier for the interface. For more information, search the OLE.HLP file for DISPID.

For more information about GetPropDisplayString, search the latest Microsoft Developer Network CD for GetDisplayStrings.

See Also

GetPropDisplayStrings (see page 18057)

SetPropDisplayString (see page 18075)

116.1.4.4.40 TOleControl.GetPropDisplayStrings Method

Returns the display strings for an interface for the ActiveX control.

Pascal

```
procedure GetPropDisplayStrings(DispID: Integer; List: TStrings);
```

C++

```
__fastcall GetPropDisplayStrings(int DispID, TStrings List);
```

Description

List is where the strings are to be stored.

For more information about GetPropDisplayStrings, search the latest Microsoft Developer Network CD for GetDisplayStrings.

See Also

GetPropDisplayString (see page 18057)

116.1.4.4.41 TOleControl.GetProperty Method

Pascal

```
procedure GetProperty(Index: Integer; var Value: TVarData);
```

C++

```
__fastcall GetProperty(int Index, TVarData Value);
```

Description

This is GetProperty, a member of class TOleControl.

116.1.4.4.42 TOleControl.GetShortIntProp Method

Pascal

```
function GetShortIntProp(Index: Integer): ShortInt;
```

C++

```
__fastcall ShortInt GetShortIntProp(int Index);
```

Description

This is GetShortIntProp, a member of class TOleControl.

116.1.4.4.43 TOleControl.GetSingleProp Method

Pascal

```
function GetSingleProp(Index: Integer): Single;
```

C++

```
__fastcall Single GetSingleProp(int Index);
```

Description

This is GetSingleProp, a member of class TOleControl.

116.1.4.4.44 TOleControl.GetSmallintProp Method

Pascal

```
function GetSmallintProp(Index: Integer): Smallint;
```

C++

```
__fastcall Smallint GetSmallintProp(int Index);
```

Description

This is GetSmallintProp, a member of class TOleControl.

116.1.4.4.45 TOleControl.GetStringProp Method

Pascal

```
function GetStringProp(Index: Integer): string;
```

C++

```
__fastcall AnsiString GetStringProp(int Index);
```

Description

This is GetStringProp, a member of class TOleControl.

116.1.4.4.46 TOleControl.GetTColorProp Method

Pascal

```
function GetTColorProp(Index: Integer): TColor;
```

C++

```
__fastcall TColor GetTColorProp(int Index);
```

Description

This is GetTColorProp, a member of class TOleControl.

116.1.4.4.47 TOleControl.GetTDateTimeProp Method

Pascal

```
function GetTDateTimeProp(Index: Integer): TDateTime;
```

C++

```
__fastcall TDateTime GetTDateTimeProp(int Index);
```

Description

This is GetTDateTimeProp, a member of class TOleControl.

116.1.4.4.48 TOleControl.GetTFontProp Method

Pascal

```
function GetTFontProp(Index: Integer): TFont;
```

C++

```
__fastcall TFont GetTFontProp(int Index);
```

Description

This is GetTFontProp, a member of class TOleControl.

116.1.4.4.49 TOleControl.GetTOleEnumProp Method

Pascal

```
function GetTOleEnumProp(Index: Integer): TOleEnum;
```

C++

```
__fastcall TOleEnum GetTOleEnumProp(int Index);
```

Description

This is GetTOleEnumProp, a member of class TOleControl.

116.1.4.4.50 TOleControl.GetTPictureProp Method

Pascal

```
function GetTPictureProp(Index: Integer): TPicture;
```

C++

```
__fastcall TPicture GetTPictureProp(int Index);
```

Description

This is GetTPictureProp, a member of class TOleControl.

116.1.4.4.51 TOleControl.GetTypeInfo Method

Retrieves the type information for an object.

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

GetTypeInfo implements the IDispatch interface GetTypeInfo method. For components that support interfaces, GetTypeInfo calls the GetTypeInfo method for the interface supported by the component, passing it the specified parameters. Use the returned value to get the type information for an interface implemented by the component.

See Also

GetIDsOfNames (see page 2777)

GetTypeInfoCount (see page 2779)

Invoke (see page 2781)

116.1.4.4.52 TOleControl.GetTypeInfoCount Method

Returns the number of type information interfaces that an object provides (either 0 or 1).

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

GetTypeInfoCount implements the IDispatch interface GetTypeInfoCount method. For components that support interfaces, GetTypeInfoCount calls this method for the interface supported by the component. The Count parameter points to a location that receives the number of type information interfaces provided by the object. If the object provides type information, this number is 1; otherwise the number is 0.

See Also

GetIDsOfNames (see page 2777)

GetTypeInfo (see page 2779)

Invoke (see page 2781)

116.1.4.4.53 TOleControl.GetVariantProp Method

Pascal

```
function GetVariantProp(Index: Integer): Variant;
```

C++

```
__fastcall Variant GetVariantProp(int Index);
```

Description

This is GetVariantProp, a member of class TOleControl.

116.1.4.4.54 TOleControl.GetWideStringProp Method

Pascal

```
function GetWideStringProp(Index: Integer): WideString;
```

C++

```
__fastcall BSTR GetWideStringProp(int Index);
```

Description

This is GetWideStringProp, a member of class TOleControl.

116.1.4.4.55 TOleControl.GetWindowContext Method

Pascal

```
function GetWindowContext(out frame: IOleInPlaceFrame; out doc: IOleInPlaceUIWindow; out
rcPosRect: TRect; out rcClipRect: TRect; out frameInfo: TOleInPlaceFrameInfo): HResult;
stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetWindowContext(IOleInPlaceFrame frame, IOleInPlaceUIWindow doc,
TRect rcPosRect, TRect rcClipRect, TOleInPlaceFrameInfo frameInfo);
```

Description

This is GetWindowContext, a member of class TOleControl.

116.1.4.4.56 TOleControl.GetWordBoolProp Method

Pascal

```
function GetWordBoolProp(Index: Integer): WordBool;
```

C++

```
__fastcall WordBool GetWordBoolProp(int Index);
```

Description

This is GetWordBoolProp, a member of class TOleControl.

116.1.4.4.57 TOleControl.GetWordProp Method

Pascal

```
function GetWordProp(Index: Integer): Word;
```

C++

```
__fastcall Word GetWordProp(int Index);
```

Description

This is GetWordProp, a member of class TOleControl.

116.1.4.4.58 TOleControl.InitControlData Method

Pascal

```
procedure InitControlData; virtual; abstract;
```

C++

```
virtual __fastcall InitControlData() = 0;
```

Description

This is InitControlData, a member of class TOleControl.

116.1.4.4.59 TOleControl.InitControlInterface Method

Pascal

```
procedure InitControlInterface(const Obj: IUnknown); virtual;
```

C++

```
virtual __fastcall InitControlInterface(const IUnknown Obj);
```

Description

This is InitControlInterface, a member of class TOleControl.

116.1.4.4.60 TOleControl.InsertMenus Method

Pascal

```
function InsertMenus(hmenuShared: HMenu; var menuWidths: TOleMenuGroupWidths): HResult;
stdcall;
```

C++

```
__fastcall __stdcall HRESULT InsertMenus(HMenu hmenuShared, TOleMenuGroupWidths menuWidths);
```

Description

This is InsertMenus, a member of class TOleControl.

116.1.4.4.61 TOleControl.Invoke Method

Provides access to Automation properties and methods when the component wraps an Automation object.

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
```

```
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word Flags,
Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

Invoke is the standard mechanism for accessing the exposed properties and methods of an Automation object. For components that wrap the IDispatch interface of an Automation object, Invoke calls the Invoke method for the interface supported by the component, passing it the parameters specified by the function.

See Also

GetIDsOfNames (see page 2777)

GetTypeInfo (see page 2779)

GetTypeInfoCount (see page 2779)

ComObject (see page 2793)

116.1.4.4.62 TOleControl.InvokeEvent Method

Pascal

```
procedure InvokeEvent(DispID: TDispID; var Params: TDispParams); virtual;
```

C++

```
virtual __fastcall InvokeEvent(TDispID DispID, TDispParams Params);
```

Description

This is InvokeEvent, a member of class TOleControl.

116.1.4.4.63 TOleControl.InvokeMethod Method

Pascal

```
procedure InvokeMethod(const DispInfo; Result: Pointer);
```

C++

```
__fastcall InvokeMethod(const  DispInfo, void * Result);
```

Description

This is InvokeMethod, a member of class TOleControl.

116.1.4.4.64 TOleControl.IsCustomProperty Method

Indicates whether an interface for the ActiveX control is a custom property.

Pascal

```
function IsCustomProperty(DispID: Integer): Boolean;
```

C++

```
__fastcall Boolean IsCustomProperty(int DispID);
```

Description

Use IsCustomProperty to determine whether an interface for the ActiveX control is a custom property.

DispID is the identifier for the interface. For more information, search the OLE.HLP file for DISPID.

IsCustomProperty returns true if an interface is a custom property; otherwise, IsCustomProperty returns false.

116.1.4.4.65 TOleControl.IsPropPageProperty Method

Indicates whether an interface for the OCX control is a property page property.

Pascal

```
function IsPropPageProperty(DispID: Integer): Boolean;
```

C++

```
__fastcall Boolean IsPropPageProperty(int DispID);
```

Description

Use IsPropPageProperty to determine whether an interface for the OCX control is a property page property.

DispID is the identifier for the interface. For more information, search the OLE.HLP file for DISPID.

IsPropPageProperty returns true if an interface is a property page property; otherwise, IsPropPageProperty returns false.

116.1.4.4.66 TOleControl.LockInPlaceActive Method

Pascal

```
function LockInPlaceActive(fLock: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT LockInPlaceActive(BOOL fLock);
```

Description

This is LockInPlaceActive, a member of class TOleControl.

116.1.4.4.67 TOleControl.OleControlSite_TranslateAccelerator Method

Pascal

```
function OleControlSite_TranslateAccelerator(msg: PMsg; grfModifiers: Longint): HResult;
stdcall;
```

C++

```
__fastcall __stdcall HRESULT OleControlSite_TranslateAccelerator(PMsg msg, Longint
grfModifiers);
```

Description

This is OleControlSite_TranslateAccelerator, a member of class TOleControl.

116.1.4.4.68 TOleControl.OleInPlaceFrame_GetWindow Method

Pascal

```
function OleInPlaceFrame_GetWindow(out wnd: HWnd): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OleInPlaceFrame_GetWindow(HWnd wnd);
```

Description

This is OleInPlaceFrame_GetWindow, a member of class TOleControl.

116.1.4.4.69 TOleControl.OleInPlaceFrame_TranslateAccelerator Method

Pascal

```
function OleInPlaceFrame_TranslateAccelerator(var msg: TMsg; wID: Word): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OleInPlaceFrame_TranslateAccelerator(TMsg msg, Word wID);
```

Description

This is OleInPlaceFrame_TranslateAccelerator, a member of class TOleControl.

116.1.4.4.70 TOleControl.OleInPlaceSite_GetWindow Method

Pascal

```
function OleInPlaceSite_GetWindow(out wnd: HWnd): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OleInPlaceSite_GetWindow(HWnd wnd);
```

Description

This is OleInPlaceSite_GetWindow, a member of class TOleControl.

116.1.4.4.71 TOleControl.OnChanged Method

Pascal

```
function OnChanged(dispid: TDispID): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT OnChanged(TDispID dispid);
```

Description

This is OnChanged, a member of class TOleControl.

116.1.4.4.72 TOleControl.OnControlInfoChanged Method

Pascal

```
function OnControlInfoChanged: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnControlInfoChanged();
```

Description

This is OnControlInfoChanged, a member of class TOleControl.

116.1.4.4.73 TOleControl.OnFocus Method

Pascal

```
function OnFocus(fGotFocus: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnFocus(BOOL fGotFocus);
```

Description

This is OnFocus, a member of class TOleControl.

116.1.4.4.74 TOleControl.OnInPlaceActivate Method

Pascal

```
function OnInPlaceActivate: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnInPlaceActivate();
```

Description

This is OnInPlaceActivate, a member of class TOleControl.

116.1.4.4.75 TOleControl.OnInPlaceDeactivate Method

Pascal

```
function OnInPlaceDeactivate: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnInPlaceDeactivate();
```

Description

This is OnInPlaceDeactivate, a member of class TOleControl.

116.1.4.4.76 TOleControl.OnPosRectChange Method

Pascal

```
function OnPosRectChange(const rcPosRect: TRect): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnPosRectChange(const TRect rcPosRect);
```

Description

This is OnPosRectChange, a member of class TOleControl.

116.1.4.4.77 TOleControl.OnRequestEdit Method

Pascal

```
function OnRequestEdit(dispid: TDispID): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT OnRequestEdit(TDispID dispid);
```

Description

This is OnRequestEdit, a member of class TOleControl.

116.1.4.4.78 TOleControl.OnShowWindow Method

Pascal

```
function OnShowWindow(fShow: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnShowWindow(BOOL fShow);
```

Description

This is OnShowWindow, a member of class TOleControl.

116.1.4.4.79 TOleControl.OnUIActivate Method

Pascal

```
function OnUIActivate: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnUIActivate();
```

Description

This is OnUIActivate, a member of class TOleControl.

116.1.4.4.80 TOleControl.OnUIDeactivate Method

Pascal

```
function OnUIDeactivate(fUndoable: BOOL): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT OnUIDeactivate(BOOL fUndoable);
```

Description

This is OnUIDeactivate, a member of class TOleControl.

116.1.4.4.81 TOleControl.PaletteChanged Method

Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.

Pascal

```
function PaletteChanged(Foreground: Boolean): Boolean; override;
```

C++

```
virtual __fastcall Boolean PaletteChanged(Boolean Foreground);
```

Description

PaletteChanged is called automatically when the Windows system palette changes. TWinControl responds to these notifications by trying to realize the windowed control's palette, if any, into the current device context, and then passing on the PaletteChanged notification to each of the child controls in turn. If the change in the system palette causes the actual value of the control's palette to change, PaletteChanged will invalidate the control so that it can repaint with the new palette.

Windows paints the active window with a foreground palette, while other windows are forced to use background palettes. Background palettes are approximate matches to the colors specified by the control's logical palette, given the limitations imposed by implementing the foreground palette. Windows only allows a single foreground palette.

PaletteChanged allows the control to obtain a new realization of its palette when the system palette changes. When Foreground is true, the form has been activated and the control is specifying the new foreground palette. When Foreground is false, another application or form has changed the foreground palette, and controls that are sensitive to the available palette should realize new background palettes to best match their logical palettes.

If the control does not have a logical palette to be realized into the current device context, GetPalette will return a handle of 0, and PaletteChanged will not try to realize a palette for the windowed control. Instead, it simply passes the notification on to the child controls. If the windowed control or any of its child controls realize a palette into the current device context in response to PaletteChanged, this method returns true.

Override PaletteChanged to change the way in which controls are given an opportunity to realize a new palette when the Windows system palette changes. For example, override PaletteChanged to change the order in which child controls realize their palettes, giving a particular control the foreground palette when the form is activated, or to pass palette change notification to custom controls implemented outside of the VCL, such as in DLLs.

Override GetPalette instead to affect whether the windowed control actually has a palette to realize, or to change the value of the control's logical palette.

Note: PaletteChanged is called only when the run-time video mode requires palette support, such as for 256 color mode, but not 16 million color mode.

See Also

GetDeviceContext (see page 6155)

GetPalette (see page 5765)

116.1.4.4.82 TOleControl.PictureChanged Method

Pascal

```
procedure PictureChanged(Sender: TObject);
```

C++

```
__fastcall PictureChanged(TObject * Sender);
```

Description

This is PictureChanged, a member of class TOleControl.

116.1.4.4.83 TOleControl.PostMessageFilter Method

Pascal

```
function PostMessageFilter(wnd: HWnd; msg: Integer; wp: Integer; lp: Integer; out res:
Integer; Cookie: Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT PostMessageFilter(HWnd wnd, int msg, int wp, int lp, int res,
Longint Cookie);
```

Description

This is PostMessageFilter, a member of class TOleControl.

116.1.4.4.84 TOleControl.PreMessageFilter Method

Pascal

```
function PreMessageFilter(wnd: HWnd; msg: Integer; wp: Integer; lp: Integer; out res: Integer;
out Cookie: Longint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT PreMessageFilter(HWnd wnd, int msg, int wp, int lp, int res,
Longint Cookie);
```

Description

This is PreMessageFilter, a member of class TOleControl.

116.1.4.4.85 TOleControl.QueryInterface Method

Returns a reference to a specified interface if the object supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override;
```

C++

```
virtual __fastcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the component supports the interface specified by IID, and if so, returns a reference to that interface as the Obj parameter. If the component does not support the interface, the Obj parameter returns nil (Delphi) or NULL (C++).

Note: For components that act as COM object wrapper, QueryInterface calls the QueryInterface method of the internal COM object.

See Also

_AddRef (see page 2769)

_Release (see page 2769)

QueryInterface

QueryInterface (see page 22398)

Invoke (see page 2781)

ComObject (see page 2793)

116.1.4.4.86 TOleControl.QueryService Method

Pascal

```
function QueryService(const rsid: TGuid; const iid: TGuid; out obj): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT QueryService(const TGuid rsid, const TGuid iid,  obj);
```

Description

This is QueryService, a member of class TOleControl.

116.1.4.4.87 TOleControl.RemoveMenus Method

Pascal

```
function RemoveMenus(hmenuShared: HMenu): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT RemoveMenus(HMenu hmenuShared);
```

Description

This is RemoveMenus, a member of class TOleControl.

116.1.4.4.88 TOleControl.RequestBorderSpace Method

Pascal

```
function RequestBorderSpace(const borderwidths: TRect): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT RequestBorderSpace(const TRect borderwidths);
```

Description

This is RequestBorderSpace, a member of class TOleControl.

116.1.4.4.89 TOleControl.RequestNewObjectLayout Method

Pascal

```
function RequestNewObjectLayout: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT RequestNewObjectLayout();
```

Description

This is RequestNewObjectLayout, a member of class TOleControl.

116.1.4.4.90 TOleControl.SaveObject Method

Pascal

```
function SaveObject: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SaveObject();
```

Description

This is SaveObject, a member of class TOleControl.

116.1.4.4.91 TOleControl.Scroll Method

Pascal

```
function Scroll(scrollExtent: TPoint): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Scroll(TPoint scrollExtent);
```

Description

This is Scroll, a member of class TOleControl.

116.1.4.4.92 TOleControl.SetActiveObject Method

Pascal

```
function SetActiveObject(const activeObject: IOleInPlaceActiveObject; pszObjName: POleStr):
HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetActiveObject(const IOleInPlaceActiveObject activeObject,
POleStr pszObjName);
```

Description

This is SetActiveObject, a member of class TOleControl.

116.1.4.4.93 TOleControl.SetBorderSpace Method

Pascal

```
function SetBorderSpace(pborderwidths: PRect): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetBorderSpace(PRect pborderwidths);
```

Description

This is SetBorderSpace, a member of class TOleControl.

116.1.4.4.94 TOleControl.SetBounds Method

Sets the boundaries of the ActiveX control.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to set the boundaries of the ActiveX control.

ALeft, ATop, AWidth, and AHeight are the x-coordinate of the upper-left corner, the y-coordinate of the upper-left corner, the width and the height of the ActiveX control.

116.1.4.4.95 TOleControl.SetByteProp Method

Pascal

```
procedure SetByteProp(Index: Integer; Value: Byte);
```

C++

```
__fastcall SetByteProp(int Index, Byte Value);
```

Description

This is SetByteProp, a member of class TOleControl.

116.1.4.4.96 TOleControl.SetColorProp Method

Pascal

```
procedure SetColorProp(Index: Integer; Value: TColor);
```

C++

```
__fastcall SetColorProp(int Index, TColor Value);
```

Description

This is SetColorProp, a member of class TOleControl.

116.1.4.4.97 TOleControl.SetCompProp Method

Pascal

```
procedure SetCompProp(Index: Integer; const Value: Comp);
```

C++

```
__fastcall SetCompProp(int Index, const Comp Value);
```

Description

This is SetCompProp, a member of class TOleControl.

116.1.4.4.98 TOleControl.SetCurrencyProp Method

Pascal

```
procedure SetCurrencyProp(Index: Integer; const Value: Currency);
```

C++

```
__fastcall SetCurrencyProp(int Index, const Currency Value);
```

Description

This is SetCurrencyProp, a member of class TOleControl.

116.1.4.4.99 TOleControl.SetDoubleProp Method

Pascal

```
procedure SetDoubleProp(Index: Integer; const Value: Double);
```

C++

```
__fastcall SetDoubleProp(int Index, const Double Value);
```

Description

This is SetDoubleProp, a member of class TOleControl.

116.1.4.4.100 TOleControl.SetIDispatchProp Method

Pascal

```
procedure SetIDispatchProp(Index: Integer; const Value: IDispatch);
```

C++

```
__fastcall SetIDispatchProp(int Index, const IDispatch Value);
```

Description

This is SetIDispatchProp, a member of class TOleControl.

116.1.4.4.101 TOleControl.SetIntegerProp Method

Pascal

```
procedure SetIntegerProp(Index: Integer; Value: Integer);
```

C++

```
__fastcall SetIntegerProp(int Index, int Value);
```

Description

This is SetIntegerProp, a member of class TOleControl.

116.1.4.4.102 TOleControl.SetIUnknownProp Method

Pascal

```
procedure SetIUnknownProp(Index: Integer; const Value: IUnknown);
```

C++

```
__fastcall SetIUnknownProp(int Index, const IUnknown Value);
```

Description

This is SetIUnknownProp, a member of class TOleControl.

116.1.4.4.103 TOleControl.SetMenu Method

Pascal

```
function SetMenu(hmenuShared: HMenu; holemenu: HMenu; hwndActiveObject: HWnd): HResult;
stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetMenu(HMenu hmenuShared, HMenu holemenu, HWnd hwndActiveObject);
```

Description

This is SetMenu, a member of class TOleControl.

116.1.4.4.104 TOleControl.SetName Method

Sets the name of the control.

Pascal

```
procedure SetName(const Value: TComponentName); override;
```

C++

```
virtual __fastcall SetName(const TComponentName Value);
```

Description

SetName is the protected write implementation of the Name property. TControl updates the Text or Caption property, if appropriate, after changing the value of Name.

Override SetName to execute additional code when the value of the Name property changes.

See Also

Name

Caption (see page 5806)

Text (see page 5842)

116.1.4.4.105 TOleControl.SetOleBoolProp Method

Pascal

```
procedure SetOleBoolProp(Index: Integer; Value: TOleBool);
```

C++

```
__fastcall SetOleBoolProp(int Index, TOleBool Value);
```

Description

This is SetOleBoolProp, a member of class TOleControl.

116.1.4.4.106 TOleControl.SetOleDateProp Method

Pascal

```
procedure SetOleDateProp(Index: Integer; const Value: TOleDate);
```

C++

```
__fastcall SetOleDateProp(int Index, const TOleDate Value);
```

Description

This is SetOleDateProp, a member of class TOleControl.

116.1.4.4.107 TOleControl.SetOleEnumProp Method

Pascal

```
procedure SetOleEnumProp(Index: Integer; Value: TOleEnum);
```

C++

```
__fastcall SetOleEnumProp(int Index, TOleEnum Value);
```

Description

This is SetOleEnumProp, a member of class TOleControl.

116.1.4.4.108 TOleControl.SetOleVariantProp Method

Pascal

```
procedure SetOleVariantProp(Index: Integer; const Value: OleVariant);
```

C++

```
__fastcall SetOleVariantProp(int Index, const OleVariant Value);
```

Description

This is SetOleVariantProp, a member of class TOleControl.

116.1.4.4.109 TOleControl.SetParent Method

Sets the parent of the control.

Pascal

```
procedure SetParent(AParent: TWinControl); override;
```

C++

```
virtual __fastcall SetParent(TWinControl * AParent);
```

Description

SetParent is the protected implementation of the Parent property. Override SetParent to execute additional code when the value of the Parent property changes.

The AParent parameter specifies the new parent of the control.

If the control already has a parent, SetParent removes the control from that parent's list of controls by calling the parent's RemoveControl method. If AParent is not nil (Delphi) or NULL (C++), SetParent calls its InsertControl method to add the control to its list of controls.

The TControl destructor calls SetParent(nil (Delphi) or NULL (C++)) to remove the control from its parent's control list before destroying the component.

See Also

Destroy (see page 5732)

InsertControl (see page 6160)

Parent (see page 5832)

RemoveControl (see page 6172)

116.1.4.4.110 TOleControl.SetPropDisplayString Method

Sets the display string for an interface for the ActiveX control.

Pascal

```
procedure SetPropDisplayString(DispID: Integer; const Value: string);
```

C++

```
__fastcall SetPropDisplayString(int DispID, const AnsiString Value);
```

Description

Use SetPropDisplayString to set the display string for an interface for the ActiveX control.

DispID is the identifier for the interface. For more information, search the OLE.HLP file for DISPID.

Value is the new display string for the interface.

See Also

GetPropDisplayStrings (see page 18057)

SetPropDisplayString

116.1.4.4.111 TOleControl.SetProperty Method

Pascal

```
procedure SetProperty(Index: Integer; const Value: TVarData);
```

C++

```
__fastcall SetProperty(int Index, const TVarData Value);
```

Description

This is SetProperty, a member of class TOleControl.

116.1.4.4.112 TOleControl.SetShortIntProp Method

Pascal

```
procedure SetShortIntProp(Index: Integer; Value: Shortint);
```

C++

```
__fastcall SetShortIntProp(int Index, Shortint Value);
```

Description

This is SetShortIntProp, a member of class TOleControl.

116.1.4.4.113 TOleControl.SetSingleProp Method

Pascal

```
procedure SetSingleProp(Index: Integer; const Value: Single);
```

C++

```
__fastcall SetSingleProp(int Index, const Single Value);
```

Description

This is SetSingleProp, a member of class TOleControl.

116.1.4.4.114 TOleControl.SetSmallintProp Method

Pascal

```
procedure SetSmallintProp(Index: Integer; Value: Smallint);
```

C++

```
__fastcall SetSmallintProp(int Index, Smallint Value);
```

Description

This is SetSmallintProp, a member of class TOleControl.

116.1.4.4.115 TOleControl.SetStatusText Method

Pascal

```
function SetStatusText(pszStatusText: POleStr): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT SetStatusText(POleStr pszStatusText);
```

Description

This is SetStatusText, a member of class TOleControl.

116.1.4.4.116 TOleControl.SetStringProp Method

Pascal

```
procedure SetStringProp(Index: Integer; const Value: string);
```

C++

```
__fastcall SetStringProp(int Index, const AnsiString Value);
```

Description

This is SetStringProp, a member of class TOleControl.

116.1.4.4.117 TOleControl.SetTColorProp Method

Pascal

```
procedure SetTColorProp(Index: Integer; Value: TColor);
```

C++

```
__fastcall SetTColorProp(int Index, TColor Value);
```

Description

This is SetTColorProp, a member of class TOleControl.

116.1.4.4.118 TOleControl.SetTDateTimeProp Method

Pascal

```
procedure SetTDateTimeProp(Index: Integer; const Value: TDateTime);
```

C++

```
__fastcall SetTDateTimeProp(int Index, const TDateTime Value);
```

Description

This is SetTDateTimeProp, a member of class TOleControl.

116.1.4.4.119 TOleControl.SetTFontProp Method

Pascal

```
procedure SetTFontProp(Index: Integer; Value: TFont);
```

C++

```
__fastcall SetTFontProp(int Index, TFont Value);
```

Description

This is SetTFontProp, a member of class TOleControl.

116.1.4.4.120 TOleControl.SetTOleEnumProp Method

Pascal

```
procedure SetTOleEnumProp(Index: Integer; Value: TOleEnum);
```

C++

```
__fastcall SetTOleEnumProp(int Index, TOleEnum Value);
```

Description

This is SetTOleEnumProp, a member of class TOleControl.

116.1.4.4.121 TOleControl.SetTPictureProp Method

Pascal

```
procedure SetTPictureProp(Index: Integer; Value: TPicture);
```

C++

```
__fastcall SetTPictureProp(int Index, TPicture Value);
```

Description

This is SetTPictureProp, a member of class TOleControl.

116.1.4.4.122 TOleControl.SetVariantProp Method

Pascal

```
procedure SetVariantProp(Index: Integer; const Value: Variant);
```

C++

```
__fastcall SetVariantProp(int Index, const Variant Value);
```

Description

This is SetVariantProp, a member of class TOleControl.

116.1.4.4.123 TOleControl.SetWideStringProp Method

Pascal

```
procedure SetWideStringProp(Index: Integer; const Value: WideString);
```

C++

```
__fastcall SetWideStringProp(int Index, const BSTR Value);
```

Description

This is SetWideStringProp, a member of class TOleControl.

116.1.4.4.124 TOleControl.SetWordBoolProp Method

Pascal

```
procedure SetWordBoolProp(Index: Integer; Value: WordBool);
```

C++

```
__fastcall SetWordBoolProp(int Index, WordBool Value);
```

Description

This is SetWordBoolProp, a member of class TOleControl.

116.1.4.4.125 TOleControl.SetWordProp Method

Pascal

```
procedure SetWordProp(Index: Integer; Value: Word);
```

C++

```
__fastcall SetWordProp(int Index, Word Value);
```

Description

This is SetWordProp, a member of class TOleControl.

116.1.4.4.126 TOleControl.ShowAboutBox Method

Notifies the ActiveX control to display its About box.

Pascal

```
procedure ShowAboutBox;
```

C++

```
__fastcall ShowAboutBox();
```

Description

Use ShowAboutBox to notify the ActiveX control to display its About box.

116.1.4.4.127 TOleControl.ShowObject Method

Pascal

```
function ShowObject: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT ShowObject();
```

Description

This is ShowObject, a member of class TOleControl.

116.1.4.4.128 TOleControl.ShowPropertyFrame Method

Pascal

```
function ShowPropertyFrame: HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT ShowPropertyFrame();
```

Description

This is ShowPropertyFrame, a member of class TOleControl.

116.1.4.4.129 TOleControl.StandardEvent Method

Pascal

```
procedure StandardEvent(DispID: TDispID; var Params: TDispParams); virtual;
```

C++

```
virtual __fastcall StandardEvent(TDispID DispID, TDispParams Params);
```

Description

This is StandardEvent, a member of class TOleControl.

116.1.4.4.130 TOleControl.SuppressException Method

Pascal

```
function SuppressException(E: Exception): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean SuppressException(Exception E);
```

Description

This is SuppressException, a member of class TOleControl.

116.1.4.4.131 TOleControl.TransformCoords Method

Pascal

```
function TransformCoords(var ptlHimetric: TPoint; var ptfContainer: TPointF; flags: Longint):
HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT TransformCoords(TPoint ptlHimetric, TPointF ptfContainer, Longint
flags);
```

Description

This is TransformCoords, a member of class TOleControl.

116.1.4.4.132 TOleControl.WndProc Method

Provides specific message responses for the control.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

Override WndProc to change the initial Windows message handler for the control. The WindowProc property is initialized to point to the WndProc method.

WndProc for TWinControl overrides the inherited method to define responses for focus, mouse, and keyboard messages. It sends all others to its inherited WndProc.

When overriding WndProc to provide specialized responses to messages, call the inherited WndProc at the end to dispatch any unhandled messages.

C++ Examples:

```
/*
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
*/
class TMyListBoxDescendant : public TListBox
{
__published:    // IDE-managed Components
    void __fastcall SubClassWndProc(Messages::TMessage &Message);
    void __fastcall ToggleSubClass(bool On);
    void __fastcall OnDrawItemProc(
      TWinControl *Control, int Index, const TRect &Rect,
      TOwnerDrawState State);
private:    // User declarations
public:     // User declarations
    __fastcall TMyListBoxDescendant(TComponent* Owner);
};

TForm1 *Form1;
TMyListBoxDescendant *MyListBoxDescendant1;
Graphics::TBitmap *bitmap0;

const WM_STYLEMESSAGE = WM_USER + 2000;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

__fastcall TMyListBoxDescendant::TMyListBoxDescendant(TComponent* Owner)
    : TListBox(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
```

```
    0);
}

void __fastcall TForm1::SubClassRadioGroup1Click(TObject *Sender)
{
  MyListBoxDescendant1->ToggleSubClass(SubClassRadioGroup1->ItemIndex == 0);
}

void __fastcall TMyListBoxDescendant::SubClassWndProc(Messages::TMessage &Message)
{
  if (Message.Msg == WM_STYLEMESSAGE)
    Style = (TListBoxStyle)Message.WParam;
  else
    WndProc(Message);
}

void __fastcall TMyListBoxDescendant::ToggleSubClass(bool On)
{
  if (On)
    WindowProc = SubClassWndProc;
  else
    WindowProc = WndProc;
}

void __fastcall TMyListBoxDescendant::OnDrawItemProc(TWinControl *Control, int Index,
      const TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *bitmap; // temporary variable for item's bitmap
  int Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *canvas = ((TListBox *)Control)->Canvas;
  canvas->FillRect(Rect); // clear the rectangle
  bitmap = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
  if (bitmap)
  {
    canvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, bitmap->Width, bitmap->Height),
      bitmap, Bounds(0, 0, bitmap->Width, bitmap->Height), clRed); // render bitmap
    Offset += bitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  canvas->TextOut(Rect.Left + Offset, Rect.Top, ((TListBox *)Control)->Items->Strings[Index]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyListBoxDescendant1 = new TMyListBoxDescendant(Form1);
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Parent = Form1;
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Left =
    SubClassRadioGroup1->Left + SubClassRadioGroup1->Width + 30;;
  MyListBoxDescendant1->Top = SubClassRadioGroup1->Top;
  MyListBoxDescendant1->Height = SubClassRadioGroup1->Height;
  MyListBoxDescendant1->OnDrawItem =
    MyListBoxDescendant1->OnDrawItemProc;

  bitmap0 = new Graphics::TBitmap;
  ImageList1->GetBitmap(0, bitmap0);
  MyListBoxDescendant1->Items->AddObject("Butterfly", bitmap0);

  SubClassRadioGroup1->Items->Add("SubClassWndProc");
  SubClassRadioGroup1->Items->Add("WndProc");
  SubClassRadioGroup1->ItemIndex = 2;
}
```

Delphi Examples:

```
{
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
}

type
  TMyListBoxDescendant = class(TlistBox)
    procedure SubClassWndProc(var Message: TMessage);
    procedure ToggleSubClass(On: Boolean);
    procedure OnDrawItemProc(
      Control: TWinControl;
      Index: Integer;
      Rect:TRect;
      State: TOwnerDrawState);
  end;
  TForm1 = class(TForm)
    SubClassRadioGroup1: TRadioGroup;
    Button1: TButton;
    ImageList1: TImageList;
    Button2: TButton;
    procedure SubClassRadioGroup1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  MyListBoxDescendant1: TMyListBoxDescendant;
  bitmap0: TBitmap;

implementation

{$R *.dfm}

const WM_STYLEMESSAGE = WM_USER + 2000;

procedure TForm1.Button1Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
end;

procedure TMyListBoxDescendant.SubClassWndProc(var Message: TMessage);
begin
```

```
    if (Message.Msg = WM_STYLEMESSAGE) then
      Style:= TListBoxStyle(Message.WParam)
    else
      WndProc(Message);
  end;

  procedure TMyListBoxDescendant.ToggleSubClass(On: Boolean);
  begin
    if On then
      WindowProc := SubClassWndProc
    else
      WindowProc := WndProc;
  end;

  procedure TForm1.SubClassRadioGroup1Click(Sender: TObject);
  begin
    MyListBoxDescendant1.ToggleSubClass(
      SubClassRadioGroup1.ItemIndex = 0);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    MyListBoxDescendant1:= TMyListBoxDescendant.Create(self);
    MyListBoxDescendant1.Visible:= True;
    MyListBoxDescendant1.Parent:= Form1;
    MyListBoxDescendant1.Visible:= True;
    MyListBoxDescendant1.Left:=
      SubClassRadioGroup1.Left + SubClassRadioGroup1.Width + 30;;
    MyListBoxDescendant1.Top:= SubClassRadioGroup1.Top;
    MyListBoxDescendant1.Height:= SubClassRadioGroup1.Height;
    MyListBoxDescendant1.OnDrawItem:=
      MyListBoxDescendant1.OnDrawItemProc;

    bitmap0 := TBitmap.Create;
    ImageList1.GetBitmap(0, bitmap0);
    MyListBoxDescendant1.Items.AddObject('Butterfly', bitmap0);

    SubClassRadioGroup1.Items.Add('SubClassWndProc');
    SubClassRadioGroup1.Items.Add('WndProc');
    SubClassRadioGroup1.ItemIndex := 2;
  end;

  procedure TMyListBoxDescendant.OnDrawItemProc(
    Control: TWinControl;
    Index: Integer;
    Rect:TRect;
    State: TOwnerDrawState);
  var
    Bitmap: TBitmap;      { temporary variable for the item's bitmap }
    Offset: Integer;      { text offset width }
  begin
    { draw on control canvas, not on the form }
    with (Control as TListBox).Canvas do
    begin
      FillRect(Rect);       { clear the rectangle }
      Offset := 2;          { provide default offset }
      Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
      if Bitmap <> nil then
      begin
        BrushCopy(
          Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
          Bitmap,
          Bounds(0, 0, Bitmap.Width, Bitmap.Height),
          clRed);  {render bitmap}
        Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
      end;
```

```
    TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
text }
  end;
end;
```

See Also

DefWndProc (see page 6199)

MainWndProc (see page 6165)

WndProc (see page 5791)

WindowProc (see page 5872)

116.1.4.5 TOleControl Properties

The properties of the TOleControl class are listed here.

Protected Properties

	Name	Description
	ControlData (see page 18086)	This is ControlData, a member of class TOleControl.

Public Properties

	Name	Description
	Anchors (see page 18085)	Specifies how the control is anchored to its parent.
R	DefaultDispatch (see page 18086)	The default dispatch interface.
R	OleObject (see page 18086)	The ActiveX control associated with the TOleControl (see page 18030) object.
R	PerPropBrowsing (see page 18086)	Provides access to information about the property page properties of the OleControl.
	TabStop (see page 18087)	Determines if the user can tab to a control.

116.1.4.5.1 TOleControl.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

116.1.4.5.2 TOleControl.ControlData Property

Pascal

```
property ControlData: PControlData;
```

C++

```
__property PControlData ControlData;
```

Description

This is ControlData, a member of class TOleControl.

116.1.4.5.3 TOleControl.DefaultDispatch Property

The default dispatch interface.

Pascal

```
property DefaultDispatch: IDispatch;
```

C++

```
__property IDispatch DefaultDispatch;
```

Description

DefaultDispatch default dispatch interface that the TOleControl (see page 18030) object encapsulates. For those ActiveX controls that have multiple dispatch interfaces, one is typically identified as the default.

116.1.4.5.4 TOleControl.OleObject Property

The ActiveX control associated with the TOleControl (see page 18030) object.

Pascal

```
property OleObject: Variant;
```

C++

```
__property Variant OleObject;
```

Description

OleObject is used to get the ActiveX control that is associated with the TOleControl (see page 18030)

116.1.4.5.5 TOleControl.PerPropBrowsing Property

Provides access to information about the property page properties of the OleControl.

Pascal

```
property PerPropBrowsing: IPerPropertyBrowsing;
```

C++

```
__property IPerPropertyBrowsing PerPropBrowsing;
```

Description

PerPropBrowsing is an interface to information about the properties of the OleControl as they appear on the associated property page.

See Also

BrowseProperties (see page 18048)

116.1.4.5.6 TOleControl.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

116.1.4.6 TOleControl Events

The events of the TOleControl class are listed here.

Public Events

	Name	Description
	ServiceQuery (see page 18087)	This is ServiceQuery, a member of class TOleControl.

116.1.4.6.1 TOleControl.ServiceQuery Event

Pascal

```
property ServiceQuery: TServiceQuery;
```

C++

```
__property TServiceQuery ServiceQuery;
```

Description

This is ServiceQuery, a member of class TOleControl.

116.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FontToOleFont (see page 18088)	Returns a Variant containing an IFontDispatch interface that represents a TFont object.
	OleFontToFont (see page 18088)	Fills in a TFont structure to represent an IFontDispatch.

116.2.1 OleCtrls.FontToOleFont Function

Returns a Variant containing an IFontDispatch interface that represents a TFont object.

Pascal

```
function FontToOleFont(Font: TFont): Variant;
```

C++

```
Variant FontToOleFont(TFont Font);
```

File

OleCtrls

Description

Use FontToOleFont to convert a font from TFont to IFontDispatch format. The interface uses the TFont object to implement the methods of the interface.

See Also

OleFontToFont (see page 18088)

OleStrToStrVar

StringToOleStr

OleStrToString

116.2.2 OleCtrls.OleFontToFont Function

Fills in a TFont structure to represent an IFontDispatch.

Pascal

```
procedure OleFontToFont(const OleFont: Variant; Font: TFont);
```

C++

```
OleFontToFont(const Variant OleFont, TFont Font);
```

File

OleCtrls

Description

Use OleFontToFont to convert a font from COM (IFontDispatch) to TFont format.

See Also

FontToOleFont (see page 18088)

OleStrToStrVar

StringToOleStr

OleStrToString

116.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TControlData (see page 18090)	This is record OleCtrls.TControlData.
	TControlData2 (see page 18091)	This is record OleCtrls.TControlData2.
	TEnumValue (see page 18091)	This is record OleCtrls.TEnumValue.

116.3.1 OleCtrls.TControlData Record

Pascal

```
TControlData = record
  ClassID: TGUID;
  EventIID: TGUID;
  EventCount: Longint;
  EventDispIDs: Pointer;
  LicenseKey: Pointer;
  Flags: DWORD;
  Version: Integer;
  FontCount: Integer;
  FontIDs: PDispIDList;
  PictureCount: Integer;
  PictureIDs: PDispIDList;
  Reserved: Integer;
  InstanceCount: Integer;
  EnumPropDescs: TList;
end;
```

C++

```
struct TControlData {
  TGUID ClassID;
  TGUID EventIID;
  Longint EventCount;
  void * EventDispIDs;
  void * LicenseKey;
  DWORD Flags;
  int Version;
  int FontCount;
  PDispIDList FontIDs;
  int PictureCount;
  PDispIDList PictureIDs;
  int Reserved;
  int InstanceCount;
  TList EnumPropDescs;
};
```

File

OleCtrls

Description

This is record OleCtrls.TControlData.

116.3.2 OleCtrls.TControlData2 Record

Pascal

```
TControlData2 = record
  ClassID: TGUID;
  EventIID: TGUID;
  EventCount: Longint;
  EventDispIDs: Pointer;
  LicenseKey: Pointer;
  Flags: DWORD;
  Version: Integer;
  FontCount: Integer;
  FontIDs: PDispIDList;
  PictureCount: Integer;
  PictureIDs: PDispIDList;
  Reserved: Integer;
  InstanceCount: Integer;
  EnumPropDescs: TList;
  FirstEventOfs: Cardinal;
end;
```

C++

```
struct TControlData2 {
  TGUID ClassID;
  TGUID EventIID;
  Longint EventCount;
  void * EventDispIDs;
  void * LicenseKey;
  DWORD Flags;
  int Version;
  int FontCount;
  PDispIDList FontIDs;
  int PictureCount;
  PDispIDList PictureIDs;
  int Reserved;
  int InstanceCount;
  TList EnumPropDescs;
  unsigned FirstEventOfs;
};
```

File

OleCtrls

Description

This is record OleCtrls.TControlData2.

116.3.3 OleCtrls.TEnumValue Record

Pascal

```
TEnumValue = record
  Value: Longint;
  Ident: string;
end;
```

C++

```
struct TEnumValue {
  Longint Value;
  AnsiString Ident;
};
```

File

OleCtrls

Description

This is record OleCtrls.TEnumValue.

116.4 Types

The following table lists types in this documentation.

Types

Name	Description
PControlData (see page 18093)	This is type OleCtrls.PControlData.
PControlData2 (see page 18093)	This is type OleCtrls.PControlData2.
PEnumValueList (see page 18094)	This is type OleCtrls.PEnumValueList.
TEnumValueList (see page 18094)	This is type OleCtrls.TEnumValueList.
TOleEnum (see page 18094)	This is type OleCtrls.TOleEnum.
TServiceQuery (see page 18095)	This is type OleCtrls.TServiceQuery.

116.4.1 OleCtrls.PControlData Type

Pascal

```
PControlData = ^TControlData;
```

C++

```
^TControlData PControlData;
```

File

OleCtrls

Description

This is type OleCtrls.PControlData.

116.4.2 OleCtrls.PControlData2 Type

Pascal

```
PControlData2 = ^TControlData2;
```

C++

```
^TControlData2 PControlData2;
```

File

OleCtrls

Description

This is type OleCtrls.PControlData2.

116.4.3 OleCtrls.PEnumValueList Type

Pascal

```
PEnumValueList = ^TEnumValueList;
```

C++

```
^TEnumValueList PEnumValueList;
```

File

OleCtrls

Description

This is type OleCtrls.PEnumValueList.

116.4.4 OleCtrls.TEnumValueList Type

Pascal

```
TEnumValueList = array[0..32767] of TEnumValue;
```

C++

```
array[0..32767] of TEnumValue TEnumValueList;
```

File

OleCtrls

Description

This is type OleCtrls.TEnumValueList.

116.4.5 OleCtrls.TOleEnum Type

Pascal

```
TOleEnum = ActiveX.TOleEnum;
```

C++

```
ActiveX.TOleEnum TOleEnum;
```

File

OleCtrls

Description

This is type OleCtrls.TOleEnum.

116.4.6 OleCtrls.TServiceQuery Type

Pascal

```
TServiceQuery = function (const rsid, iid: TGuid; out obj): HResult of object;
```

C++

```
(const rsid, iid: TGuid; out obj): HResult ( TServiceQuery)();
```

File

OleCtrls

Description

This is type OleCtrls.TServiceQuery.

116.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
EmptyParam (see page 18096)	This is variable OleCtrls.EmptyParam.

116.5.1 OleCtrls.EmptyParam Variable

Pascal

```
EmptyParam: OleVariant;
```

C++

```
OleVariant EmptyParam;
```

File

OleCtrls

Description

This is variable OleCtrls.EmptyParam.

117 OleServer Namespace

This is namespace OleServer.

Topics

Name	Description
Classes (see page 18098)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 18117)	The following table lists structs, records, enums in this documentation.
Types (see page 18119)	The following table lists types in this documentation.

Classes

	Name	Description
	TOleServer (see page 18098)	TOleServer is the base class for components that represent an imported COM server.
	TServerEventDispatch (see page 18112)	This is class OleServer.TServerEventDispatch.

Structs, Records, Enums

	Name	Description
	TConnectKind (see page 18117)	This is record OleServer.TConnectKind.
	TServerData (see page 18117)	This is record OleServer.TServerData.

Types

Name	Description
PServerData (see page 18119)	This is type OleServer.PServerData.
TVariantArray (see page 18119)	This is type OleServer.TVariantArray.

117.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TOleServer (see page 18098)	TOleServer is the base class for components that represent an imported COM server.
	TServerEventDispatch (see page 18112)	This is class OleServer.TServerEventDispatch.

117.1.1 TOleServer Class

TOleServer is the base class for components that represent an imported COM server.

Class Hierarchy

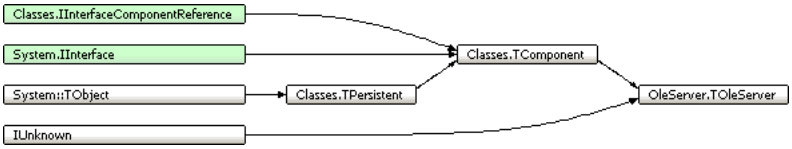

Pascal

```
TOleServer = class(TComponent, IUnknown);
```

C++

```
class TOleServer : public TComponent, public IUnknown;
```

File

OleServer

Description

TOleServer is the base class for COM servers that are imported by choosing Project|Import Type Library from the IDE. TOleServer introduces several properties and methods that manage the connection to a COM server and for invoking its default events.

Descendants of TOleServer expose the properties, methods, and events of the server CoClass. When an application calls one of the TOleServer descendant's methods or sets one of its properties, the component automatically establishes a connection to the COM server. You can also connect explicitly by calling the Connect (see page 18104) method.

Do not create instances of TOleServer. TOleServer relies on abstract pure virtual methods to establish and break a connection to the COM server. Instead, use a descendant of TOleServer that wraps the desired server. Numerous TOleServer descendants are provided on the Servers tab of the Component Palette. In addition, you can create your own descendants by importing a server's type library.

See Also

Code Generated When You Import Type Library Information

117.1.1.1 TOleServer Members

The following tables list the members exposed by TOleServer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 18104)	Increments the reference count for the number of interface references dependent on the OLE server.
	_Release (see page 18104)	Decrements the reference count for the calling interface on a object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ConnectEvents (see page 18105)	Establishes a connection using the EventDispatch (see page 18111) property as an event sink.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DisconnectEvents (see page 18106)	Terminates a connection that was established using the ConnectEvents (see page 18105) method.
	GetAutoConnect (see page 18106)	Returns the value of the AutoConnect (see page 18110) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetConnectKind (see page 18106)	Returns the value of the ConnectKind (see page 18110) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetServer (see page 18107)	Returns an interface for the COM server.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
A	InitServerData (see page 18107)	Initializes the ServerData (see page 18111) property to reflect the type library information of the COM server.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	InvokeEvent (see page 18107)	Dispatches an event from the COM server to the appropriate event handler.
V	Loaded (see page 18108)	Connects to the COM server when the Loaded object is first loaded into memory if AutoConnect (see page 18110) is True true.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 18108)	Returns a reference to a specified interface if the OLE server supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoConnect (see page 18109)	Sets the value of the AutoConnect (see page 18110) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnectKind (see page 18109)	Sets the value of the ConnectKind (see page 18110) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	EventDispatch (see page 18111)	This is EventDispatch, a member of class TOleServer.
	ServerData (see page 18111)	Encodes information about the COM server to which ServerData connects.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Connect (see page 18104)	Establishes a connection to the COM server.
V	Create (see page 18101)	Allocates memory and constructs a safely initialized instance of a component.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 18102)	Disposes of the component and its owned components.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
A	Disconnect (see page 18105)	Terminates the connection to the COM server.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	AutoConnect (see page 18110)	Specifies whether the component automatically connects to the server.
	ConnectKind (see page 18110)	Indicates the type of process to which the ConnectKind component establishes a connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RemoteMachineName (see page 18111)	Specifies the machine running the COM server when ConnectKind (see page 18110) is ckRemote.
	Tag (see page 2808)	Stores an integer value as part of a component.

117.1.1.2 TOleServer.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TOleServer(TComponent * AOwner);
```

Description

All objects have a Create method that constructs the object. TComponent redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner, as indicated by the AOwner parameter

sets the ComponentStyle property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components (see page 2799)

ComponentStyle (see page 2805)

Create (see page 22287)

Destroy (see page 2766)

Free (see page 22300)

Owner (see page 2807)

117.1.1.3 TOleServer.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TOleServer();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

Destroy (see page 22287)

DestroyComponents (see page 2771)

Destroying (see page 2772)

Free (see page 22300)

Notification (see page 2783)

Owner (see page 2807)

Release (see page 13885)

Components (see page 2799)

117.1.1.4 TOleServer Methods

The methods of the TOleServer class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 18104)	Increments the reference count for the number of interface references dependent on the OLE server.
	_Release (see page 18104)	Decrements the reference count for the calling interface on a object.
	ConnectEvents (see page 18105)	Establishes a connection using the EventDispatch (see page 18111) property as an event sink.
	DisconnectEvents (see page 18106)	Terminates a connection that was established using the ConnectEvents (see page 18105) method.
	GetAutoConnect (see page 18106)	Returns the value of the AutoConnect (see page 18110) property.
	GetConnectKind (see page 18106)	Returns the value of the ConnectKind (see page 18110) property.
V	GetServer (see page 18107)	Returns an interface for the COM server.
A	InitServerData (see page 18107)	Initializes the ServerData (see page 18111) property to reflect the type library information of the COM server.
V	InvokeEvent (see page 18107)	Dispatches an event from the COM server to the appropriate event handler.
V	Loaded (see page 18108)	Connects to the COM server when the Loaded object is first loaded into memory if AutoConnect (see page 18110) is True true.
V	QueryInterface (see page 18108)	Returns a reference to a specified interface if the OLE server supports that interface.
	SetAutoConnect (see page 18109)	Sets the value of the AutoConnect (see page 18110) property.
	SetConnectKind (see page 18109)	Sets the value of the ConnectKind (see page 18110) property.

Public Methods

	Name	Description
A	Connect (see page 18104)	Establishes a connection to the COM server.
A	Disconnect (see page 18105)	Terminates the connection to the COM server.

117.1.1.4.1 TOleServer._AddRef Method

Increments the reference count for the number of interface references dependent on the OLE server.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

_AddRef implements the IUnknown interface method, AddRef, which increments the reference count of objects using the OLE server's interface. _AddRef returns the resulting value of the reference count.

See Also

Invoke (see page 2781)

QueryInterface (see page 18108)

_Release (see page 18104)

117.1.1.4.2 TOleServer._Release Method

Decrements the reference count for the calling interface on a object.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

_Release implements the IUnknown interface method, Release, which decrements the reference count of objects using the OLE server's interface. If the reference count on the interface falls to 0, the server object is freed from memory.

_Release returns the resulting value of the reference count.

See Also

Invoke (see page 2781)

QueryInterface (see page 18108)

_AddRef (see page 18104)

117.1.1.4.3 TOleServer.Connect Method

Establishes a connection to the COM server.

Pascal

```
procedure Connect; virtual; abstract;
```

C++

```
virtual __fastcall Connect() = 0;
```

Description

Call Connect to connect the Connect component to the COM server it represents. If the AutoConnect (see page 18110)

property is True true, the Connect component automatically calls Connect when the application is first run.

Do not use Connect if the value of ConnectKind (see page 18110) is ckAttachToInterface. When attaching to an interface, use the ConnectTo method, which is introduced by descendant classes, instead.

If the value of ConnectKind (see page 18110) is ckRemote, the RemoteMachineName (see page 18111) property must be set before calling Connect.

Connect introduces Connect as an abstract a pure virtual method. Descendant classes must override this method to let the Connect component connect to a specific type of COM server.

See Also

AutoConnect (see page 18110)

ConnectKind (see page 18110)

RemoteMachineName (see page 18111)

Disconnect (see page 18105)

Using Component Wrappers

117.1.1.4.4 TOleServer.ConnectEvents Method

Establishes a connection using the EventDispatch (see page 18111) property as an event sink.

Pascal

```
procedure ConnectEvents(const Obj: IUnknown);
```

C++

```
__fastcall ConnectEvents(const IUnknown Obj);
```

Description

ConnectEvents descendants use ConnectEvents internally to implement their event handlers for events on the COM server. If the ConnectEvents descendant does not expose any server events, ConnectEvents is not used.

The Obj parameter is an interface for the COM server that generates events.

See Also

EventDispatch (see page 18111)

DisconnectEvents (see page 18106)

117.1.1.4.5 TOleServer.Disconnect Method

Terminates the connection to the COM server.

Pascal

```
procedure Disconnect; virtual; abstract;
```

C++

```
virtual __fastcall Disconnect() = 0;
```

Description

Call Disconnect to terminate a connection formed using the Connect (see page 18104) method.

Disconnect introduces Disconnect as an abstract a pure virtual method. Descendant classes must override this method to let the Disconnect component disconnect from a specific type of COM server.

See Also

Connect (see page 18104)

Using Component Wrappers

117.1.1.4.6 TOleServer.DisconnectEvents Method

Terminates a connection that was established using the ConnectEvents (see page 18105) method.

Pascal

```
procedure DisconnectEvents(const Obj: Iunknown);
```

C++

```
__fastcall DisconnectEvents(const Iunknown Obj);
```

Description

DisconnectEvents descendants use DisconnectEvents internally to terminate the connection established by ConnectEvents (see page 18105). If the DisconnectEvents descendant does not expose any server events, DisconnectEvents is not used.

The Obj parameter is an interface for the COM server that generates events.

See Also

ConnectEvents (see page 18105)

117.1.1.4.7 TOleServer.GetAutoConnect Method

Returns the value of the AutoConnect (see page 18110) property.

Pascal

```
function GetAutoConnect: Boolean;
```

C++

```
__fastcall Boolean GetAutoConnect();
```

Description

GetAutoConnect is the protected read implementation for the AutoConnect (see page 18110) property.

See Also

AutoConnect (see page 18110)

117.1.1.4.8 TOleServer.GetConnectKind Method

Returns the value of the ConnectKind (see page 18110) property.

Pascal

```
function GetConnectKind: TConnectKind;
```

C++

```
__fastcall TConnectKind GetConnectKind();
```

Description

GetConnectKind is the protected read implementation for the ConnectKind (see page 18110) property.

See Also

ConnectKind (see page 18110)

117.1.1.4.9 TOleServer.GetServer Method

Returns an interface for the COM server.

Pascal

```
function GetServer: IUnknown; virtual;
```

C++

```
virtual __fastcall IUnknown GetServer();
```

Description

GetServer descendants call GetServer from the implementation of their Connect (see page 18104) methods. GetServer obtains an interface for the COM server described by ServerData (see page 18111). The ConnectKind (see page 18110) property indicates how GetServer can obtain the interface.

See Also

Connect (see page 18104)

ConnectKind (see page 18110)

ServerData (see page 18111)

117.1.1.4.10 TOleServer.InitServerData Method

Initializes the ServerData (see page 18111) property to reflect the type library information of the COM server.

Pascal

```
procedure InitServerData; virtual; abstract;
```

C++

```
virtual __fastcall InitServerData() = 0;
```

Description

The InitServerData constructor calls InitServerData to initialize ServerData (see page 18111) with information from the COM server's type library.

InitServerData is an abstract a pure virtual method in InitServerData. Each InitServerData descendant implements this method so that the ServerData (see page 18111) property reflects its particular COM server.

See Also

ServerData (see page 18111)

Create (see page 18101)

117.1.1.4.11 TOleServer.InvokeEvent Method

Dispatches an event from the COM server to the appropriate event handler.

Pascal

```
procedure InvokeEvent(DispID: TDispID; var Params: TVariantArray); virtual;
```

C++

```
virtual __fastcall InvokeEvent(TDispID DispID, TVariantArray Params);
```

Description

InvokeEvent descendants use InvokeEvent internally to link their event handlers to events on the COM server.

DispID is the dispatch ID for the appropriate event handler.

Params passes the parameters of the event handler as an array of OleVariants.

As implemented in InvokeEvent, InvokeEvent does nothing, because InvokeEvent has no events. Descendants that surface events from a COM server override this method so that the event sink specified by EventDispatch (see page 18111) can pass events on to the appropriate event handlers.

See Also

EventDispatch (see page 18111)

117.1.1.4.12 TOleServer.Loaded Method

Connects to the COM server when the Loaded object is first loaded into memory if AutoConnect (see page 18110) is True true.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

The VCL streaming system automatically calls Loaded after the Loaded component is loaded into memory.

Loaded checks the AutoConnect (see page 18110) property, and if it is True true, calls the Connect (see page 18104) method to establish a connection.

See Also

AutoConnect (see page 18110)

Loaded (see page 2782)

117.1.1.4.13 TOleServer.QueryInterface Method

Returns a reference to a specified interface if the OLE server supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override;
```

C++

```
virtual __fastcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface implements the IUnknown interface method QueryInterface. QueryInterface returns a reference to the interface specified by the IID parameter as the Obj parameter. If the OLE server does not support the interface, the Obj parameter returns nil NULL and QueryInterface returns a nonzero error code.

See Also

_AddRef (see page 18104)

_Release (see page 18104)

Invoke (see page 2781)

117.1.1.4.14 TOleServer.SetAutoConnect Method

Sets the value of the AutoConnect (see page 18110) property.

Pascal

```
procedure SetAutoConnect(flag: Boolean);
```

C++

```
__fastcall SetAutoConnect(Boolean flag);
```

Description

SetAutoConnect is the protected write implementation for the AutoConnect (see page 18110) property.

See Also

AutoConnect (see page 18110)

117.1.1.4.15 TOleServer.SetConnectKind Method

Sets the value of the ConnectKind (see page 18110) property.

Pascal

```
procedure SetConnectKind(ck: TConnectKind);
```

C++

```
__fastcall SetConnectKind(TConnectKind ck);
```

Description

SetConnectKind is the protected write implementation for the ConnectKind (see page 18110) property.

See Also

ConnectKind (see page 18110)

117.1.1.5 TOleServer Properties

The properties of the TOleServer class are listed here.

Protected Properties

	Name	Description
	EventDispatch (see page 18111)	This is EventDispatch, a member of class TOleServer.
	ServerData (see page 18111)	Encodes information about the COM server to which ServerData connects.

Published Properties

	Name	Description
	AutoConnect (see page 18110)	Specifies whether the component automatically connects to the server.
	ConnectKind (see page 18110)	Indicates the type of process to which the ConnectKind component establishes a connection.
	RemoteMachineName (see page 18111)	Specifies the machine running the COM server when ConnectKind (see page 18110) is ckRemote.

117.1.1.5.1 TOleServer.AutoConnect Property

Specifies whether the component automatically connects to the server.

Pascal

```
property AutoConnect: Boolean;
```

C++

```
__property Boolean AutoConnect;
```

Description

Set AutoConnect to True true to specify whether the server should be loaded automatically when the application first starts up. If AutoConnect is False false, the application establishes a connection when it calls one of the AutoConnect descendant's methods, sets a property, or calls the Connect (see page 18104) method.

Note: AutoConnect only establishes a connection when the application is first run. Changing the AutoConnect property at runtime has no effect.

See Also

Connect (see page 18104)

Using Component Wrappers

117.1.1.5.2 TOleServer.ConnectKind Property

Indicates the type of process to which the ConnectKind component establishes a connection.

Pascal

```
property ConnectKind: TConnectKind;
```

C++

```
__property TConnectKind ConnectKind;
```

Description

Set ConnectKind to indicate how the ConnectKind component establishes a connection. ConnectKind establishes the connection when the application is run (if AutoConnect (see page 18110) is True true) or when the application calls the Connect (see page 18104) (or ConnectTo) method.

The following table lists the possible values:

Value	Meaning
ckRunningOrNew	Attach to a running server or create a new instance of the server.
ckNewInstance	Always create a new instance of the server.
ckRunningInstance	Only attach to a running instance of the server.
ckRemote	Bind to a remote instance of the server. When using this option, you must supply a value for RemoteMachineName (see page 18111).
ckAttachToInterface	Don't bind to the server. Instead, the application supplies an interface using the ConnectTo method, which is introduced in descendant classes. This option can't be used with the AutoConnect (see page 18110) property.

See Also

Connect (see page 18104)

RemoteMachineName (see page 18111)

AutoConnect (see page 18110)

Using Component Wrappers

117.1.1.5.3 TOleServer.EventDispatch Property

Pascal

```
property EventDispatch: TServerEventDispatch;
```

C++

```
__property TServerEventDispatch EventDispatch;
```

Description

This is EventDispatch, a member of class TOleServer.

117.1.1.5.4 TOleServer.RemoteMachineName Property

Specifies the machine running the COM server when ConnectKind (see page 18110) is ckRemote.

Pascal

```
property RemoteMachineName: string;
```

C++

```
__property AnsiString RemoteMachineName;
```

Description

Use RemoteMachineName when the COM server instance is running remotely on another machine. Specify the name of the machine on which the COM server is running as the value of RemoteMachineName.

Note: RemoteMachineName is only used if ConnectKind (see page 18110) is ckRemote. Setting RemoteMachineName to a non-empty string sets ConnectKind (see page 18110) to ckRemote.

See Also

ConnectKind (see page 18110)

Connect (see page 18104)

Using Component Wrappers

117.1.1.5.5 TOleServer.ServerData Property

Encodes information about the COM server to which ServerData connects.

Pascal

```
property ServerData: PServerData;
```

C++

```
__property PServerData ServerData;
```

Description

ServerData uses ServerData internally to store type library information describing its COM server. Each ServerData descendant initializes this property in the constructor, specifying the information it needs to establish a connection.

See Also

InitServerData (see page 18107)

117.1.2 TServerEventDispatch Class

Class Hierarchy

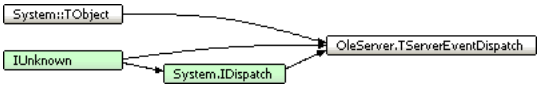

Pascal

```
TServerEventDispatch = class(TObject, IUnknown, IDispatch);
```

C++

```
class TServerEventDispatch : public TObject, public IUnknown, public IDispatch;
```

File

OleServer

Description

This is class OleServer.TServerEventDispatch.

117.1.2.1 TServerEventDispatch Members

The following tables list the members exposed by TServerEventDispatch.

Protected Methods

	Name	Description
	_AddRef (see page 18114)	This is _AddRef, a member of class TServerEventDispatch.
	_Release (see page 18114)	This is _Release, a member of class TServerEventDispatch.
	GetIDsOfNames (see page 18114)	This is GetIDsOfNames, a member of class TServerEventDispatch.
	GetTypeInfo (see page 18114)	This is GetTypeInfo, a member of class TServerEventDispatch.
	GetTypeInfoCount (see page 18114)	This is GetTypeInfoCount, a member of class TServerEventDispatch.
	Invoke (see page 18115)	This is Invoke, a member of class TServerEventDispatch.
	QueryInterface (see page 18115)	This is QueryInterface, a member of class TServerEventDispatch.
	ServerDisconnect (see page 18115)	This is ServerDisconnect, a member of class TServerEventDispatch.

Protected Properties

	Name	Description
R	Server (see page 18116)	This is Server, a member of class TServerEventDispatch.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 18113)	This is Create, a member of class TServerEventDispatch.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

117.1.2.2 TServerEventDispatch.Create Constructor

Pascal

```
constructor Create(Server: TOleServer);
```

C++

```
__fastcall TServerEventDispatch(TOleServer Server);
```

Description

This is Create, a member of class TServerEventDispatch.

117.1.2.3 TServerEventDispatch Methods

The methods of the TServerEventDispatch class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 18114)	This is _AddRef, a member of class TServerEventDispatch.
	_Release (see page 18114)	This is _Release, a member of class TServerEventDispatch.
	GetIDsOfNames (see page 18114)	This is GetIDsOfNames, a member of class TServerEventDispatch.
	GetTypeInfo (see page 18114)	This is GetTypeInfo, a member of class TServerEventDispatch.
	GetTypeInfoCount (see page 18114)	This is GetTypeInfoCount, a member of class TServerEventDispatch.
	Invoke (see page 18115)	This is Invoke, a member of class TServerEventDispatch.
	QueryInterface (see page 18115)	This is QueryInterface, a member of class TServerEventDispatch.
	ServerDisconnect (see page 18115)	This is ServerDisconnect, a member of class TServerEventDispatch.

117.1.2.3.1 TServerEventDispatch._AddRef Method

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

This is _AddRef, a member of class TServerEventDispatch.

117.1.2.3.2 TServerEventDispatch._Release Method

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

This is _Release, a member of class TServerEventDispatch.

117.1.2.3.3 TServerEventDispatch.GetIDsOfNames Method

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int NameCount, int
LocaleID, void * DispIDs);
```

Description

This is GetIDsOfNames, a member of class TServerEventDispatch.

117.1.2.3.4 TServerEventDispatch.GetTypeInfo Method

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

This is GetTypeInfo, a member of class TServerEventDispatch.

117.1.2.3.5 TServerEventDispatch.GetTypeInfoCount Method

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

This is GetTypeInfoCount, a member of class TServerEventDispatch.

117.1.2.3.6 TServerEventDispatch.Invoke Method

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word Flags,
Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is Invoke, a member of class TServerEventDispatch.

117.1.2.3.7 TServerEventDispatch.QueryInterface Method

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

This is QueryInterface, a member of class TServerEventDispatch.

117.1.2.3.8 TServerEventDispatch.ServerDisconnect Method

Pascal

```
function ServerDisconnect: Boolean;
```

C++

```
__fastcall Boolean ServerDisconnect();
```

Description

This is ServerDisconnect, a member of class TServerEventDispatch.

117.1.2.4 TServerEventDispatch Properties

The properties of the TServerEventDispatch class are listed here.

Protected Properties

	Name	Description
R	Server (see page 18116)	This is Server, a member of class TServerEventDispatch.

117.1.2.4.1 TServerEventDispatch.Server Property

Pascal

```
property Server: TOleServer;
```

C++

```
__property TOleServer Server;
```

Description

This is Server, a member of class TServerEventDispatch.

117.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TConnectKind (see page 18117)	This is record OleServer.TConnectKind.

Records

	Name	Description
	TServerData (see page 18117)	This is record OleServer.TServerData.

117.2.1 OleServer.TConnectKind Enumeration

Pascal

```
TConnectKind = (
  ckRunningOrNew,
  ckNewInstance,
  ckRunningInstance,
  ckRemote,
  ckAttachToInterface
);
```

C++

```
enum TConnectKind {
  ckRunningOrNew,
  ckNewInstance,
  ckRunningInstance,
  ckRemote,
  ckAttachToInterface
};
```

File

OleServer

Description

This is record OleServer.TConnectKind.

117.2.2 OleServer.TServerData Record

Pascal

```
TServerData = record
  ClassID: TGUID;
  IntfIID: TGUID;
  EventIID: TGUID;
  LicenseKey: Pointer;
  Version: Integer;
  InstanceCount: Integer;
```

```
end;
```

C++

```
struct TServerData {
  TGUID ClassID;
  TGUID IntfIID;
  TGUID EventIID;
  void * LicenseKey;
  int Version;
  int InstanceCount;
};
```

File

OleServer

Description

This is record OleServer.TServerData.

117.3 Types

The following table lists types in this documentation.

Types

Name	Description
PServerData (see page 18119)	This is type OleServer.PServerData.
TVariantArray (see page 18119)	This is type OleServer.TVariantArray.

117.3.1 OleServer.PServerData Type

Pascal

```
PServerData = ^TServerData;
```

C++

```
^TServerData PServerData;
```

File

OleServer

Description

This is type OleServer.PServerData.

117.3.2 OleServer.TVariantArray Type

Pascal

```
TVariantArray = Array of OleVariant;
```

C++

```
Array of OleVariant TVariantArray;
```

File

OleServer

Description

This is type OleServer.TVariantArray.

118 OPConvert Namespace

This is namespace OPConvert.

Topics

Name	Description
Interfaces (see page 18122)	The following table lists interfaces in this documentation.
Structs, Records, Enums (see page 18132)	The following table lists structs, records, enums in this documentation.
Types (see page 18133)	The following table lists types in this documentation.

Interfaces

	Name	Description
	IOPConvert (see page 18122)	IOPConvert is the interface for marshaling and unmarshaling method calls on an invokable interface.

Structs, Records, Enums

	Name	Description
	TSOAPConvertOption (see page 18132)	This is record OPConvert.TSOAPConvertOption.

Types

Name	Description
TSOAPConvertOptions (see page 18133)	This is type OPConvert.TSOAPConvertOptions.

118.1 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IOPConvert (see page 18122)	IOPConvert is the interface for marshaling and unmarshaling method calls on an invokable interface.

118.1.1 IOPConvert Interface

IOPConvert is the interface for marshaling and unmarshaling method calls on an invokable interface.

Class Hierarchy

OPConvert.IOPConvert

Pascal

```
IOPConvert = interface;
```

C++

```
__interface IOPConvert;
```

File

OPConvert

Description

IOPConvert is implemented by classes that marshal and unmarshal method calls in Web Service applications. It defines methods for marshaling and unmarshaling method calls, as well as any exceptions that occur when attempting to execute those method calls.

The IOPConvert interface is used by remote interfaced objects on the client (TRIO descendants) and invoker components on the server (TSoapPascalInvoker descendants). Both the remote interfaced object and the invoker classes use the same IOPConvert implementation class, which represents a single encoding system (such as SOAP).

Classes (see page 2685) that implement IOPConvert use the runtime type information (RTTI) on an invokable interface to interpret or encode method calls. They do not execute these calls, but rather, convert between transportable strings that are the encoding of interface method calls and an invocation context (TInvContext object).

Note: C++ method declarations that use IOPConvert use the _di_IOPConvert type instead. This type is a DelphiInterface wrapper around the IOPConvert interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IOPConvert >  _di_IOPConvert;
```

See Also

TOPToSoapDomConvert (see page 18137)

TRio

TSoapPascalInvoker (see page 20020)

TInvContext (see page 15516)

118.1.1.1 IOPConvert Members

The following tables list the members exposed by IOPConvert.

Public Methods

	Name	Description
	GetAttachments (see page 18124)	Returns the value of the Attachments (see page 18130) property.
	GetEncoding (see page 18124)	Returns the value of the Encoding (see page 18130) property.
	GetOptions (see page 18124)	Returns the value of the Options (see page 18131) property.
	GetTempDir (see page 18124)	Returns the value of the TempDir (see page 18131) property.
	InvContextToMsg (see page 18125)	Marshals a method call from an invocation context to a transportable string.
	MakeFault (see page 18125)	Encodes an exception that was raised when trying to execute a method on an invokable interface.
	MakeResponse (see page 18126)	Encodes the results of successfully executing a method on an invokable interface.
	MsgToInvContext (see page 18126)	This is the overview for the MsgToInvContext method overload.
	ProcessResponse (see page 18127)	This is the overview for the ProcessResponse method overload.
	SetAttachments (see page 18128)	Sets the value of the Attachments (see page 18130) property.
	SetEncoding (see page 18129)	Sets the value of the Encoding (see page 18130) property.
	SetOptions (see page 18129)	Sets the value of the Options (see page 18131) property.
	SetTempDir (see page 18129)	Sets the value of the TempDir (see page 18131) property.

Public Properties

	Name	Description
	Attachments (see page 18130)	Lists the objects that encapsulate the attachments included with the SOAP request or its response.
	Encoding (see page 18130)	Indicates the character set to use for encoded method calls.
	Options (see page 18131)	Controls (see page 5711) how certain types of conversions are performed.
	TempDir (see page 18131)	Specifies the directory where temporary files for attachments are saved.

118.1.1.2 IOPConvert Methods

The methods of the IOPConvert class are listed here.

Public Methods

	Name	Description
	GetAttachments (see page 18124)	Returns the value of the Attachments (see page 18130) property.
	GetEncoding (see page 18124)	Returns the value of the Encoding (see page 18130) property.
	GetOptions (see page 18124)	Returns the value of the Options (see page 18131) property.
	GetTempDir (see page 18124)	Returns the value of the TempDir (see page 18131) property.
	InvContextToMsg (see page 18125)	Marshals a method call from an invocation context to a transportable string.
	MakeFault (see page 18125)	Encodes an exception that was raised when trying to execute a method on an invokable interface.
	MakeResponse (see page 18126)	Encodes the results of successfully executing a method on an invokable interface.
	MsgToInvContext (see page 18126)	This is the overview for the MsgToInvContext method overload.
	ProcessResponse (see page 18127)	This is the overview for the ProcessResponse method overload.
	SetAttachments (see page 18128)	Sets the value of the Attachments (see page 18130) property.
	SetEncoding (see page 18129)	Sets the value of the Encoding (see page 18130) property.
	SetOptions (see page 18129)	Sets the value of the Options (see page 18131) property.
	SetTempDir (see page 18129)	Sets the value of the TempDir (see page 18131) property.

118.1.1.2.1 IOPConvert.GetAttachments Method

Returns the value of the Attachments (see page 18130) property.

Pascal

```
function GetAttachments: TSoapDataList;
```

C++

```
__fastcall TSoapDataList GetAttachments();
```

Description

GetAttachments is the read implementation for the Attachments (see page 18130) property.

See Also

Attachments (see page 18130)

118.1.1.2.2 IOPConvert.GetEncoding Method

Returns the value of the Encoding (see page 18130) property.

Pascal

```
function GetEncoding: WideString;
```

C++

```
__fastcall BSTR GetEncoding();
```

Description

GetEncoding is the read implementation for the Encoding (see page 18130) property.

See Also

Encoding (see page 18130)

118.1.1.2.3 IOPConvert.GetOptions Method

Returns the value of the Options (see page 18131) property.

Pascal

```
function GetOptions: TSOAPConvertOptions;
```

C++

```
__fastcall TSOAPConvertOptions GetOptions();
```

Description

GetOptions is the read implementation for the Options (see page 18131) property.

See Also

Options (see page 18131)

118.1.1.2.4 IOPConvert.GetTempDir Method

Returns the value of the TempDir (see page 18131) property.

Pascal

```
function GetTempDir: string;
```

C++

```
__fastcall AnsiString GetTempDir();
```

Description

GetTempDir is the read implementation for the TempDir (see page 18131) property.

See Also

TempDir (see page 18131)

118.1.1.2.5 IOPConvert.InvContextToMsg Method

Marshals a method call from an invocation context to a transportable string.

Pascal

```
function InvContextToMsg(const IntfMD: TIntfMetaData; MethNum: Integer; Con: TInvContext;
Headers: THeaderList): TStream;
```

C++

```
__fastcall TStream InvContextToMsg(const TIntfMetaData IntfMD, int MethNum, TInvContext Con,
THeaderList Headers);
```

Description

InvContextToMsg looks up a specified method call, including the types and values of any arguments, from an invocation context, and returns an encoded version of that method call that can be transported to the application that executes it.

IntfMD represents the runtime type information (RTTI) from the interface whose method is called.

MethNum identifies the called method by its position in the interface's method table.

Con is the invocation context that contains details of the method call, including the parameter values passed in, the calling conventions, and any other temporary data used when making the call.

Headers is a list of TSOAPHeader descendants that represent the headers of the SOAP envelope that contained the method call.

InvContextToMsg returns a stream from which an application can read the encoded version of the method call. This encoded method call can then be passed to an IWebNode interface for execution.

See Also

MsgToInvContext (see page 18127)

Execute (see page 25432)

118.1.1.2.6 IOPConvert.MakeFault Method

Encodes an exception that was raised when trying to execute a method on an invokable interface.

Pascal

```
procedure MakeFault(const Ex: Exception; EStream: TStream);
```

C++

```
__fastcall MakeFault(const Exception Ex, TStream EStream);
```

Description

MakeFault encodes an exception that a Web Service application catches when trying to execute a method on an invokable interface. It generates an encoded exception packet, which the client can decode and use to raise the exception on the client application.

Ex is the exception object that represents the exception that occurred.

EStream is a stream to which MakeFault writes the encoded exception packet.

See Also

MakeResponse (see page 18126)

118.1.1.2.7 IOPConvert.MakeResponse Method

Encodes the results of successfully executing a method on an invokable interface.

Pascal

```
procedure MakeResponse(const IntfMD: TIntfMetaData; const MethNum: Integer; Context:
TInvContext; Response: TStream; Headers: THeaderList);
```

C++

```
__fastcall MakeResponse(const TIntfMetaData IntfMD, const int MethNum, TInvContext Context,
TStream Response, THeaderList Headers);
```

Description

MakeResponse retrieves the results of executing a method on an invokable interface from an invocation context and packages them into a transportable string, which it writes to a stream. This string is passed back to the client, which can unpack it using the ProcessResponse (see page 18128) method.

IntfMD represents the runtime type information (RTTI) from the interface whose method was executed.

MethNum identifies the called method by its position in the interface's method table.

Context is the invocation context that contains details of the method call.

Response is a stream to which MakeResponse writes the encoded version of the method result and any output parameters.

Headers is a list of TSOAPHeader descendants that represent the headers that should be added to the response message.

See Also

MakeFault (see page 18125)

ProcessResponse (see page 18128)

118.1.1.2.8 MsgToInvContext Method

This is the overview for the MsgToInvContext method overload.

Overload List

	Name	Description
	IOPConvert.MsgToInvContext (InvString, TIntfMetaData, Integer, TInvContext) (see page 18127)	Unpacks an encoded method call into an invocation context so that it can be executed.

	IOPConvert.MsgToInvContext (TStream, TIntfMetaData, Integer, TInvContext, THeaderList) (see page 18127)	Unpacks an encoded method call into an invocation context so that it can be executed.

118.1.1.2.8.1 IOPConvert.MsgToInvContext Method (InvString, TIntfMetaData, Integer, TInvContext)

Unpacks an encoded method call into an invocation context so that it can be executed.

Pascal

```
procedure MsgToInvContext(const Request: InvString; const IntfMD: TIntfMetaData; var MethNum:
Integer; Context: TInvContext); overload;
procedure MsgToInvContext(const Request: TStream; const IntfMD: TIntfMetaData; var MethNum:
Integer; Context: TInvContext; Headers: THeaderList); overload;
```

C++

```
__fastcall MsgToInvContext(const InvString Request, const TIntfMetaData IntfMD, int MethNum,
TInvContext Context);
__fastcall MsgToInvContext(const TStream Request, const TIntfMetaData IntfMD, int MethNum,
TInvContext Context, THeaderList Headers);
```

Description

MsgToInvContext takes an encoded message that a server application receives and uses it to fill out an invocation context, which can then be used to execute the method call.

Request is the encoded method call, or a stream from which to read that method call. It is generated on the client by a call to InvContextToMsg (see page 18125).

IntfMD represents the runtime type information (RTTI) from the interface whose method is called.

MethNum returns the position of the called method in the interface's method table.

Context is the invocation context to which this method adds the information encoded in Request.

Headers is an object list to which MsgToInvContext adds TSOAPHeader descendants to represent every header in the encoded message.

See Also

InvContextToMsg (see page 18125)

118.1.1.2.9 ProcessResponse Method

This is the overview for the ProcessResponse method overload.

Overload List

	Name	Description
	IOPConvert.ProcessResponse (InvString, TIntfMetaData, TIntfMethEntry, TInvContext) (see page 18128)	Unpacks the response packet from a method call into an invocation context.
	IOPConvert.ProcessResponse (TStream, TIntfMetaData, TIntfMethEntry, TInvContext, THeaderList) (see page 18128)	Unpacks the response packet from a method call into an invocation context.

118.1.1.2.9.1 IOPConvert.ProcessResponse Method (InvString, TIntfMetaData, TIntfMethEntry, TInvContext)

Unpacks the response packet from a method call into an invocation context.

Pascal

```
procedure ProcessResponse(const Resp: TStream; const IntfMD: TIntfMetaData; const MD:
TIntfMethEntry; Context: TInvContext; Headers: THeaderList); overload;
procedure ProcessResponse(const Resp: InvString; const IntfMD: TIntfMetaData; const MD:
TIntfMethEntry; Context: TInvContext); overload; deprecated;
```

C++

```
__fastcall ProcessResponse(const TStream Resp, const TIntfMetaData IntfMD, const
TIntfMethEntry MD, TInvContext Context, THeaderList Headers);
__fastcall ProcessResponse(const InvString Resp, const TIntfMetaData IntfMD, const
TIntfMethEntry MD, TInvContext Context);
```

Description

ProcessResponse takes the encoded results of executing a method on an invokable interface from an invocation context and unpacks them into an invocation context.

Response is the encoded results of executing a method on the invokable interface, or a stream from which those results can be read. It is generated on the server by the MakeResponse (see page 18126) method if the call was successful, or the MakeFault (see page 18125) method if the call was unsuccessful.

XMLDoc (see page 26095) is the interface to an XML document that represents the results of executing a method on the invokable interface. The values of its nodes come from the MakeResponse (see page 18126) method if the call was successful, or the MakeFault (see page 18125) method if the call was unsuccessful.

IntfMD describes the runtime type information (RTTI) of the interface whose method was called.

MD describes the runtime type information (RTTI) of the method that was called.

Context is the invocation context that ProcessResponse fills out based on the contents of Response.

Headers is a list that ProcessResponse fills with TSOAPHeader descendants that represent the headers of the encoded result message.

See Also

MakeFault (see page 18125)

MakeResponse (see page 18126)

118.1.1.2.10 IOPConvert.SetAttachments Method

Sets the value of the Attachments (see page 18130) property.

Pascal

```
procedure SetAttachments(Value: TSoapDataList);
```

C++

```
__fastcall SetAttachments(TSoapDataList Value);
```

Description

SetAttachments is the write implementation for the Attachments (see page 18130) property.

Value is a list of TSoapAttachment objects that encapsulate the current set of attachments.

See Also

Attachments (see page 18130)

118.1.1.2.11 IOPConvert.SetEncoding Method

Sets the value of the Encoding (see page 18130) property.

Pascal

```
procedure SetEncoding(const Encoding: WideString);
```

C++

```
__fastcall SetEncoding(const BSTR Encoding);
```

Description

SetEncoding is the write implementation for the Encoding (see page 18130) property.

Encoding (see page 18130) is the name of the new encoding to use for method calls.

See Also

Encoding (see page 18130)

118.1.1.2.12 IOPConvert.SetOptions Method

Sets the value of the Options (see page 18131) property.

Pascal

```
procedure SetOptions(const Value: TSOAPConvertOptions);
```

C++

```
__fastcall SetOptions(const TSOAPConvertOptions Value);
```

Description

SetOptions is the write implementation for the Options (see page 18131) property.

Value is the new set of options to use when marshaling and unmarshaling method calls.

See Also

Options (see page 18131)

118.1.1.2.13 IOPConvert.SetTempDir Method

Sets the value of the TempDir (see page 18131) property.

Pascal

```
procedure SetTempDir(const Value: string);
```

C++

```
__fastcall SetTempDir(const AnsiString Value);
```

Description

SetTempDir is the write implementation for the TempDir (see page 18131) property.

Value is the new temporary directory to use.

See Also

TempDir (see page 18131)

118.1.1.3 IOPConvert Properties

The properties of the IOPConvert class are listed here.

Public Properties

	Name	Description
	Attachments (see page 18130)	Lists the objects that encapsulate the attachments included with the SOAP request or its response.
	Encoding (see page 18130)	Indicates the character set to use for encoded method calls.
	Options (see page 18131)	Controls (see page 5711) how certain types of conversions are performed.
	TempDir (see page 18131)	Specifies the directory where temporary files for attachments are saved.

118.1.1.3.1 IOPConvert.Attachments Property

Lists the objects that encapsulate the attachments included with the SOAP request or its response.

Pascal

```
property Attachments: TSoapDataList;
```

C++

```
__property TSoapDataList Attachments;
```

Description

When a method on an invokable interface requires the user to send or receive attachments, these attachments are each represented by a TSoapAttachment object. Attachments lists the TSoapAttachment objects for each attachment of the current method.

Note: When the application that contains IOPConvert (see page 18122) receives attachments, it writes them to a temporary file rather than storing them in memory. The location for these temporary files is specified by the TempDir (see page 18131) property.

See Also

TempDir (see page 18131)

TSOAPAttachment (see page 15583)

118.1.1.3.2 IOPConvert.Encoding Property

Indicates the character set to use for encoded method calls.

Pascal

```
property Encoding: WideString;
```

C++

```
__property BSTR Encoding;
```

Description

Use Encoding to indicate the character set that is used for encoded method calls. If you do not specify an encoding (that is, if Encoding is an empty string), UTF-8 is assumed.

See Also

DefaultDOMVendor

Encoding (see page 26406)

118.1.1.3.3 IOPConvert.Options Property

Controls (see page 5711) how certain types of conversions are performed.

Pascal

```
property Options: TSOAPConvertOptions;
```

C++

```
__property TSOAPConvertOptions Options;
```

Description

Set Options to indicate how you want to handle various types of conversions. This set can include zero or more of the TSOAPConvertOption (see page 18132) flags.

See Also

MsgToInvContext (see page 18127)

ClassOptions (see page 15565)

InvContextToMsg (see page 18125)

118.1.1.3.4 IOPConvert.TempDir Property

Specifies the directory where temporary files for attachments are saved.

Pascal

```
property TempDir: string;
```

C++

```
__property AnsiString TempDir;
```

Description

When the application receives an attachment as the value of a parameter (or the return value) of the current method on the invokable interface, it writes the attachment to a temporary file. This temporary file becomes the value of the CacheFile property on the TSoapAttachment object that represents the attachment.

Use TempDir to specify the directory where temporary files for attachments are written.

Note: If TempDir is not set, the application uses the default temporary directory. On Windows, this is the directory that the Windows API GetTempPath returns. On Linux, this is obtained from the TMPDIR environment setting (or, if TMPDIR is not set, /tmp/)

See Also

Attachments (see page 18130)

CacheFile (see page 15592)

118.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TSOAPConvertOption (see page 18132)	This is record OPConvert.TSOAPConvertOption.

118.2.1 OPConvert.TSOAPConvertOption Enumeration

Pascal

```
TSOAPConvertOption = (
  soSendUntyped,
  soSendMultiRefObj,
  soSendMultiRefArray,
  soTryAllSchema,
  soRootRefNodesToBody,
  soDocument,
  soReturnSuccessForFault,
  soUTF8InHeader,
  soDontSendEmptyNodes,
  soCacheMimeResponse,
  soDontClearOutBoundHeaders,
  soCustomFaultAtDetailsNode,
  soLiteralParams,
  soUTF8EncodeXML,
  soXXXXHdr
);
```

C++

```
enum TSOAPConvertOption {
  soSendUntyped,
  soSendMultiRefObj,
  soSendMultiRefArray,
  soTryAllSchema,
  soRootRefNodesToBody,
  soDocument,
  soReturnSuccessForFault,
  soUTF8InHeader,
  soDontSendEmptyNodes,
  soCacheMimeResponse,
  soDontClearOutBoundHeaders,
  soCustomFaultAtDetailsNode,
  soLiteralParams,
  soUTF8EncodeXML,
  soXXXXHdr
};
```

File

OPConvert

Description

This is record OPConvert.TSOAPConvertOption.

118.3 Types

The following table lists types in this documentation.

Types

Name	Description
TSOAPConvertOptions (see page 18133)	This is type OPConvert.TSOAPConvertOptions.

118.3.1 OPConvert.TSOAPConvertOptions Type

Pascal

```
TSOAPConvertOptions = set of TSOAPConvertOption;
```

C++

```
TSOAPConvertOption TSOAPConvertOptions;
```

File

OPConvert

Description

This is type OPConvert.TSOAPConvertOptions.

119 OPToSOAPDomConv Namespace

This is namespace OPToSOAPDomConv.

Topics

Name	Description
Classes (see page 18137)	The following table lists classes in this documentation.
Functions (see page 18178)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 18179)	The following table lists structs, records, enums in this documentation.
Types (see page 18182)	The following table lists types in this documentation.
Variables (see page 18185)	The following table lists variables in this documentation.
Constants (see page 18186)	The following table lists constants in this documentation.

Classes

	Name	Description
	ESOAPDomConvertError (see page 18137)	ESOAPDomConvertError is the class for exceptions that occur when marshaling or unmarshaling a SOAP request.
	TOPToSoapDomConvert (see page 18137)	TOPToSoapDomConvert handles the marshaling and unmarshaling of SOAP method calls.
	TSOAPDomConv (see page 18150)	TSOAPDomConv is the base class for components that use a DOM to interpret and create SOAP method calls.

Constants

Name	Description
SVarArrayType (see page 18186)	This is constant OPToSOAPDomConv.SVarArrayType.

Functions

	Name	Description
	GetOrdPropEx (see page 18178)	This is function OPToSOAPDomConv.GetOrdPropEx.

Structs, Records, Enums

	Name	Description
	ConvNodeState (see page 18179)	This is record OPToSOAPDomConv.ConvNodeState.
	TMultiRefNodeElem (see page 18179)	This is record OPToSOAPDomConv.TMultiRefNodeElem.
	TMultiRefNodeMapElem (see page 18180)	This is record OPToSOAPDomConv.TMultiRefNodeMapElem.
	TSOAPArrayElemDesc (see page 18180)	This is record OPToSOAPDomConv.TSOAPArrayElemDesc.

Types

Name	Description
TMemberDataNotReceivedEvent (see page 18182)	This is type OPToSOAPDomConv.TMemberDataNotReceivedEvent.
TMultiRefNodeMap (see page 18182)	This is type OPToSOAPDomConv.TMultiRefNodeMap.
TMultiRefNodes (see page 18183)	This is type OPToSOAPDomConv.TMultiRefNodes.
TSOAPArrayDesc (see page 18183)	This is type OPToSOAPDomConv.TSOAPArrayDesc.
TUnhandledNodeEvent (see page 18183)	This is type OPToSOAPDomConv.TUnhandledNodeEvent.
TXMLNodeArray (see page 18184)	This is type OPToSOAPDomConv.TXMLNodeArray.

Variables

Name	Description
DefArrayElemName (see page 18185)	This is variable OPToSOAPDomConv.DefArrayElemName.

119.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ESOAPDomConvertError (see page 18137)	ESOAPDomConvertError is the class for exceptions that occur when marshaling or unmarshaling a SOAP request.
	TOPToSoapDomConvert (see page 18137)	TOPToSoapDomConvert handles the marshaling and unmarshaling of SOAP method calls.
	TSOAPDomConv (see page 18150)	TSOAPDomConv is the base class for components that use a DOM to interpret and create SOAP method calls.

119.1.1 ESOAPDomConvertError Class

ESOAPDomConvertError is the class for exceptions that occur when marshaling or unmarshaling a SOAP request.

Class Hierarchy

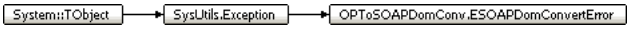

Pascal

```
ESOAPDomConvertError = class(Exception);
```

C++

```
class ESOAPDomConvertError : public Exception;
```

File

OPToSOAPDomConv

Description

TOPToSoapDomConvert (see page 18137) raises ESOAPDomConvertError exceptions when it can't marshal or unmarshal a SOAP packet that encodes a method call or its results.

See Also

TOPToSoapDOMConvert

119.1.2 TOPToSoapDomConvert Class

TOPToSoapDomConvert handles the marshaling and unmarshaling of SOAP method calls.

Class Hierarchy

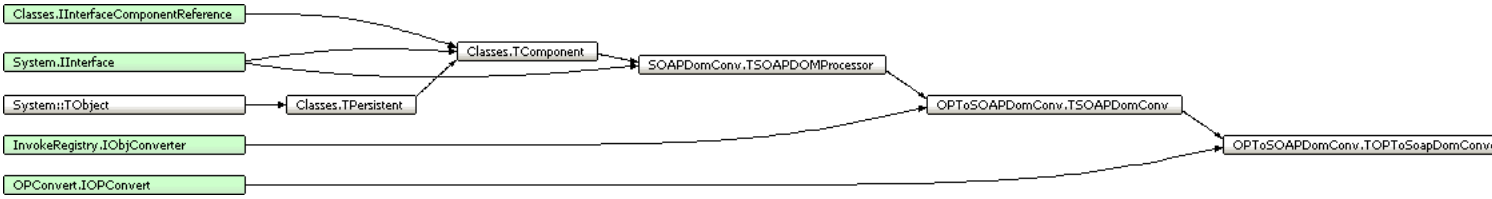

Pascal

```
TOPToSoapDomConvert = class(TSOAPDomConv, IOPConvert);
```

C++

```
class TOPToSoapDomConvert : public TSOAPDomConv, public IOPConvert;
```

File

OPToSOAPDomConv

Description

TOPToSoapDomConvert is an implementation of the IOPConvert interface that handles the marshaling and unmarshaling of SOAP method calls. It uses a DOM parser to parse and edit the SOAP encoding of a method call and its results.

Typically, applications do not use TOPToSoapDomConvert directly. Rather, It is created and used by THTTPRio on client applications and THTTPSoapPascalInvoker or THTTPSoapCppInvoker on server applications.

TOPToSoapDomConvert converts between the SOAP encoding of a method call or its results and an invocation context. If the method call results in an exception, TOPToSoapDomConvert generates a SOAP error packet for that exception so that the client can raise the exception remotely.

See Also

THTTPRio

THTTPSoapPascalInvoker (see page 19951)

THTTPSoapCppInvoker (see page 19950)

IOPConvert (see page 18122)

TInvContext (see page 15516)

ESOAPDomConvertError (see page 18137)

119.1.2.1 TOPToSoapDomConvert Members

The following tables list the members exposed by TOPToSoapDomConvert.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FHeaderProcessors (see page 19915)	This is FHeaderProcessors, a member of class TSOAPDOMProcessor.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_AddRef (see page 19916)	Called when an application uses a component interface.

	_Release (see page 2769)	Called when an application releases a component interface.
	_Release (see page 19917)	Called when an application releases a component interface.
	AddAttachment (see page 18157)	Stores information about a SOAP attachment so that it is included in the final message that is created.
	AddMultiRefNode (see page 18158)	This is AddMultiRefNode, a member of class TSOAPDomConv.
	AddNamespaceDecl (see page 18158)	This is AddNamespaceDecl, a member of class TSOAPDomConv.
	AddObjectAsWriting (see page 18158)	This is AddObjectAsWriting, a member of class TSOAPDomConv.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckEncodingStyle (see page 18158)	This is CheckEncodingStyle, a member of class TSOAPDomConv.
	ChildNodesAreNull (see page 18159)	This is ChildNodesAreNull, a member of class TSOAPDomConv.
	ConvertByteArrayToSoap (see page 18159)	This is ConvertByteArrayToSoap, a member of class TSOAPDomConv.
	ConvertEnumToSoap (see page 18159)	This is ConvertEnumToSoap, a member of class TSOAPDomConv.
	ConvertNativeArrayToSoap (see page 18159)	This is ConvertNativeArrayToSoap, a member of class TSOAPDomConv.
	ConvertObjectToSOAP (see page 18160)	This is ConvertObjectToSOAP, a member of class TSOAPDomConv.
	ConvertSoapToEnum (see page 18161)	This is ConvertSoapToEnum, a member of class TSOAPDomConv.
	ConvertSoapToNativeArray (see page 18161)	This is ConvertSoapToNativeArray, a member of class TSOAPDomConv.
	ConvertSoapToNativeArrayElem (see page 18161)	This is ConvertSoapToNativeArrayElem, a member of class TSOAPDomConv.
	ConvertSOAPToObject (see page 18162)	This is ConvertSOAPToObject, a member of class TSOAPDomConv.
	ConvertSoapToVariant (see page 18162)	This is ConvertSoapToVariant, a member of class TSOAPDomConv.
	ConvertVariantToSoap (see page 18162)	This is ConvertVariantToSoap, a member of class TSOAPDomConv.
	CreateMultiRefNode (see page 18163)	This is CreateMultiRefNode, a member of class TSOAPDomConv.
	CreateNULLNode (see page 18163)	This is CreateNULLNode, a member of class TSOAPDomConv.
	CreateObjectNode (see page 18163)	This is CreateObjectNode, a member of class TSOAPDomConv.
	CreateScalarNodeXS (see page 18163)	This is CreateScalarNodeXS, a member of class TSOAPDomConv.
	CreateTypedNode (see page 18164)	This is CreateTypedNode, a member of class TSOAPDomConv.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FinalizeMultiRefNodes (see page 18164)	This is FinalizeMultiRefNodes, a member of class TSOAPDomConv.
	FindAttachment (see page 18164)	Returns a TSOAPAttachment instance that represent the attachment with a specified ID.
V	FindHeaderProcessor (see page 19919)	Locates a processor to interpret or generate a specified SOAP header.
	FindMultiRefNodeByID (see page 18164)	This is FindMultiRefNodeByID, a member of class TSOAPDomConv.
	FindMultiRefNodeByInstance (see page 18165)	This is FindMultiRefNodeByInstance, a member of class TSOAPDomConv.
	FindNodeByHREF (see page 18165)	This is FindNodeByHREF, a member of class TSOAPDomConv.
	FindPrefixForURI (see page 18165)	This is FindPrefixForURI, a member of class TSOAPDomConv.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDataNode (see page 18165)	This is GetDataNode, a member of class TSOAPDomConv.
	GetElementType (see page 18166)	This is GetElementType, a member of class TSOAPDomConv.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetNewID (see page 18166)	This is GetNewID, a member of class TSOAPDomConv.
	GetNodeAsText (see page 18166)	This is GetNodeAsText, a member of class TSOAPDomConv.
	GetObjectPropAsText (see page 18166)	This is GetObjectPropAsText, a member of class TSOAPDomConv.
	GetOptions (see page 18166)	This is GetOptions, a member of class TSOAPDomConv.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeBySchemaNS (see page 18167)	This is GetTypeBySchemaNS, a member of class TSOAPDomConv.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).

	InitObjectFromSOAP (see page 18167)	Initializes an instance of a remotable object to reflect the values in its SOAP representation.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsNodeAVarArray (see page 18167)	This is IsNodeAVarArray, a member of class TSOAPDomConv.
	IsObjectWriting (see page 18168)	This is IsObjectWriting, a member of class TSOAPDomConv.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	LoadObject (see page 18168)	This is LoadObject, a member of class TSOAPDomConv.
	MakeArrayNode (see page 18168)	This is the overview for the MakeArrayNode method overload.
	MakeHeaderNodes (see page 19919)	Generates all the SOAP headers that this object knows how to create.
	MultiRefObject (see page 18169)	This is MultiRefObject, a member of class TSOAPDomConv.
	NewXMLDocument (see page 18147)	Creates a new empty XML document to hold the SOAP representation and returns its interface.
	NodeIsNULL (see page 18169)	This is NodeIsNULL, a member of class TSOAPDomConv.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	ObjectMemberNoShow (see page 18169)	This is ObjectMemberNoShow, a member of class TSOAPDomConv.
	ObjInstanceToSOAP (see page 18169)	Returns the SOAP representation of a remotable object.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadHeader (see page 18170)	This is ReadHeader, a member of class TSOAPDomConv.
	ReadRectDynArray (see page 18170)	This is ReadRectDynArray, a member of class TSOAPDomConv.
	ReadRectDynArrayElem (see page 18171)	This is ReadRectDynArrayElem, a member of class TSOAPDomConv.
	ReadRow (see page 18171)	This is ReadRow, a member of class TSOAPDomConv.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadVarArrayDim (see page 18171)	This is ReadVarArrayDim, a member of class TSOAPDomConv.
	ReadVariant (see page 18171)	This is ReadVariant, a member of class TSOAPDomConv.
	RemoveObjectAsWriting (see page 18172)	This is RemoveObjectAsWriting, a member of class TSOAPDomConv.
	ResetMultiRef (see page 18172)	This is ResetMultiRef, a member of class TSOAPDomConv.
	SerializationOptions (see page 18172)	This is the overview for the SerializationOptions method overload.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetNodeType (see page 18173)	This is SetNodeType, a member of class TSOAPDomConv.
	SetObjectPropFromText (see page 18173)	This is SetObjectPropFromText, a member of class TSOAPDomConv.
	SetOptions (see page 18173)	This is SetOptions, a member of class TSOAPDomConv.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	UnhandledNode (see page 18174)	This is UnhandledNode, a member of class TSOAPDomConv.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteHeader (see page 18174)	This is WriteHeader, a member of class TSOAPDomConv.
	WriteNonRectDynArray (see page 18174)	This is WriteNonRectDynArray, a member of class TSOAPDomConv.
	WriteNonRectDynArrayElem (see page 18174)	This is WriteNonRectDynArrayElem, a member of class TSOAPDomConv.
	WriteRectDynArray (see page 18175)	This is WriteRectDynArray, a member of class TSOAPDomConv.
	WriteRectDynArrayElem (see page 18175)	This is WriteRectDynArrayElem, a member of class TSOAPDomConv.

	WriteState (see page 2793)	This is WriteState, a member of class TComponent.
	WriteVarArray (see page 18175)	This is WriteVarArray, a member of class TSOAPDomConv.
	WriteVarArrayAsB64 (see page 18175)	This is WriteVarArrayAsB64, a member of class TSOAPDomConv.
	WriteVariant (see page 18176)	This is WriteVariant, a member of class TSOAPDomConv.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddAttachment (see page 15611)	Stores information about a SOAP attachment so that it is included in the final message that is created.
	AddHeaderProcessor (see page 19917)	Adds a processor to the internal list of interfaces to use for interpreting and creating SOAP headers.
	AfterConstruction (see page 19918)	Responds after the last constructor executes.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ConvertNativeDataToSoap (see page 18160)	Converts a native type to its SOAP representation.
	ConvertSoapToNativeData (see page 18161)	Converts the SOAP representation of a value to a pointer to the corresponding native representation.
	Create (see page 18143)	Creates a new TOPToSoapDomConvert (see page 18137) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DefaultProcessHeader (see page 19918)	Handles the interpretation of a SOAP header when no processor can be found.
	Destroy (see page 18143)	Frees the memory associated with the TOPToSoapDomConvert (see page 18137) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAttachment (see page 15612)	Returns a TSOAPAttachment (see page 15583) instance that represent the attachment with a specified ID.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAttachments (see page 18124)	Returns the value of the Attachments (see page 18130) property.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEncoding (see page 18124)	Returns the value of the Encoding (see page 18130) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetOptions (see page 18124)	Returns the value of the Options (see page 18131) property.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetTempDir (see page 18124)	Returns the value of the TempDir (see page 18131) property.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InitObjectFromSOAP (see page 15612)	Initializes an instance of a remotable object to reflect the values in its SOAP representation.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InvContextToMsg (see page 18144)	Marshals a method call from an invocation context to a transportable string.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	MakeFault (see page 18145)	Encodes an exception that was raised when trying to execute a method on an invokable interface.
V	MakeResponse (see page 18145)	Encodes the results of successfully executing a method on an invokable interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MsgToInvContext (see page 18126)	This is the overview for the MsgToInvContext method overload.
V	MsgToInvContext (see page 18146)	This is the overview for the MsgToInvContext method overload.
V	NewInstance (see page 19919)	Allocates memory for each instance of TSOAPDOMProcessor (see page 19912).
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ObjInstanceToSOAP (see page 15612)	Returns the SOAP representation of a remotable object.
	ProcessResponse (see page 18127)	This is the overview for the ProcessResponse method overload.
V	ProcessResponse (see page 18147)	This is the overview for the ProcessResponse method overload.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttachments (see page 18128)	Sets the value of the Attachments (see page 18130) property.
	SetEncoding (see page 18129)	Sets the value of the Encoding (see page 18130) property.
	SetOptions (see page 18129)	Sets the value of the Options (see page 18131) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTempDir (see page 18129)	Sets the value of the TempDir (see page 18131) property.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Attachments (see page 18148)	Lists the objects that encapsulate the attachments included with the SOAP request or its response.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.

	Options (see page 18131)	Controls (see page 5711) how certain types of conversions are performed.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnMemberDataNotReceived (see page 18177)	Occurs when converting a SOAP representation to a remotable object and an expected class member was not found.
	OnUnhandledNode (see page 18177)	Occurs when a SOAP representation includes an XML node that the converter does not know how to handle.

Published Properties

	Name	Description
	Encoding (see page 18149)	Indicates the character set to use for encoded method calls.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 18176)	Controls (see page 5711) how the TSOAPDomConv (see page 18150) object performs certain types of conversions.
	Tag (see page 2808)	Stores an integer value as part of a component.
	TempDir (see page 18149)	Specifies the directory where temporary files for attachments are saved.
	WSDLView (see page 18150)	Provides access to information in a WSDL document that represents the SOAP request.

119.1.2.2 TOPToSoapDomConvert.Create Constructor

Creates a new TOPToSoapDomConvert (see page 18137) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TOPToSoapDomConvert(TComponent * AOwner);
```

Description

Most applications do not create instances of TOPToSoapDomConvert (see page 18137). It is created by the THTTPRio and TSoapPascalInvoker objects to handle the marshaling and unmarshaling of SOAP method calls.

The AOwner parameter specifies the component that is responsible for freeing the TOPToSoapDomConvert (see page 18137) instance.

See Also

Owner (see page 2807)

THTTPRio

TSoapPascalInvoker (see page 20020)

119.1.2.3 TOPToSoapDomConvert.Destroy Destructor

Frees the memory associated with the TOPToSoapDomConvert (see page 18137) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TOPToSoapDomConvert();
```

Description

Typically, TOPToSoapDomConvert (see page 18137) objects are freed by the objects that create them to handle the marshaling and unmarshaling of SOAP method calls.

Do not call Destroy directly in an application. Instead, use the Free method, that checks whether the reference is nil before calling Destroy.

See Also

Create (see page 18143)

Free (see page 22300)

119.1.2.4 TOPToSoapDomConvert Methods

The methods of the TOPToSoapDomConvert class are listed here.

Protected Methods

	Name	Description
	NewXMLDocument (see page 18147)	Creates a new empty XML document to hold the SOAP representation and returns its interface.

Public Methods

	Name	Description
	InvContextToMsg (see page 18144)	Marshals a method call from an invocation context to a transportable string.
V	MakeFault (see page 18145)	Encodes an exception that was raised when trying to execute a method on an invokable interface.
V	MakeResponse (see page 18145)	Encodes the results of successfully executing a method on an invokable interface.
V	MsgToInvContext (see page 18146)	This is the overview for the MsgToInvContext method overload.
V	ProcessResponse (see page 18147)	This is the overview for the ProcessResponse method overload.

119.1.2.4.1 TOPToSoapDomConvert.InvContextToMsg Method

Marshals a method call from an invocation context to a transportable string.

Pascal

```
function InvContextToMsg(const IntfMD: TIntfMetaData; MethNum: Integer; Con: TInvContext;
Headers: THeaderList): TStream;
```

C++

```
__fastcall TStream InvContextToMsg(const TIntfMetaData IntfMD, int MethNum, TInvContext Con,
THeaderList Headers);
```

Description

InvContextToMsg looks up a specified method call, including the types and values of any arguments, from an invocation context, and returns an encoded version of that method call that can be transported to the application that executes it.

IntfMD represents the runtime type information (RTTI) from the interface whose method is called.

MethNum identifies the called method by its position in the interface's method table.

Con is the invocation context that contains details of the method call, including the parameter values passed in, the calling conventions, and any other temporary data used when making the call.

Headers is a list of TSOAPHeader descendants that represent the headers of the SOAP envelope that contained the method call.

InvContextToMsg returns a stream from which an application can read the encoded version of the method call.

See Also

MsgToInvContext (see page 18146)

119.1.2.4.2 TOPToSoapDomConvert.MakeFault Method

Encodes an exception that was raised when trying to execute a method on an invokable interface.

Pascal

```
procedure MakeFault(const Ex: Exception; EStream: TStream); virtual;
```

C++

```
virtual __fastcall MakeFault(const Exception Ex, TStream EStream);
```

Description

MakeFault encodes an exception that a Web Service application catches when trying to execute a method on an invokable interface. It generates an encoded exception packet, which the client can decode and use to raise the exception on the client application.

Ex is the exception object that represents the exception that occurred.

EStream is a stream to which MakeFault writes the encoded exception packet.

See Also

MakeResponse (see page 18145)

119.1.2.4.3 TOPToSoapDomConvert.MakeResponse Method

Encodes the results of successfully executing a method on an invokable interface.

Pascal

```
procedure MakeResponse(const IntfMD: TIntfMetaData; const MethNum: Integer; Context:
TInvContext; Response: TStream; Headers: THeaderLIst); virtual;
```

C++

```
virtual __fastcall MakeResponse(const TIntfMetaData IntfMD, const int MethNum, TInvContext
Context, TStream Response, THeaderLIst Headers);
```

Description

MakeResponse retrieves the results of executing a method on an invokable interface from an invocation context and packages them into a transportable string, which it writes to a stream. This string is passed back to the client, which can unpack it using the ProcessResponse (see page 18147) method.

IntfMD represents the runtime type information (RTTI) from the interface whose method was executed.

MethNum identifies the called method by its position in the interface's method table.

Context is the invocation context that contains details of the method call.

Headers is a list of TSOAPHeader descendants that should be converted to headers on the encoded response.

Response is a stream to which MakeResponse writes the encoded version of the method result and any output parameters.

Headers is a list of TSOAPHeader descendants that represent the headers that should be added to the response message.

See Also

MakeFault (see page 18145)

ProcessResponse (see page 18147)

119.1.2.4.4 MsgToInvContext Method

This is the overview for the MsgToInvContext method overload.

Overload List

	Name	Description
	TOPToSoapDomConvert.MsgToInvContext (InvString, TIntfMetaData, Integer, TInvContext) (see page 18146)	Unpacks an encoded method call into an invocation context so that it can be executed.
	TOPToSoapDomConvert.MsgToInvContext (TStream, TIntfMetaData, Integer, TInvContext, THeaderList) (see page 18146)	Unpacks an encoded method call into an invocation context so that it can be executed.

119.1.2.4.4.1 TOPToSoapDomConvert.MsgToInvContext Method (InvString, TIntfMetaData, Integer, TInvContext)

Unpacks an encoded method call into an invocation context so that it can be executed.

Pascal

```
procedure MsgToInvContext(const Request: InvString; const IntfMD: TIntfMetaData; var MethNum:
Integer; Context: TInvContext); virtual; overload;
procedure MsgToInvContext(const Request: TStream; const IntfMD: TIntfMetaData; var MethNum:
Integer; Context: TInvContext; Headers: THeaderList); virtual; overload;
```

C++

```
virtual __fastcall MsgToInvContext(const InvString Request, const TIntfMetaData IntfMD, int
MethNum, TInvContext Context);
virtual __fastcall MsgToInvContext(const TStream Request, const TIntfMetaData IntfMD, int
MethNum, TInvContext Context, THeaderList Headers);
```

Description

MsgToInvContext takes an encoded message that a server application receives and uses it to fill out an invocation context, which can then be used to execute the method call.

Request is the encoded method call, or a stream from which to read that method call. It is generated on the client by a call to InvContextToMsg (see page 18144).

IntfMD represents the runtime type information (RTTI) from the interface whose method is called.

MethNum returns the position of the called method in the interface's method table.

Context is the invocation context to which this method adds the information encoded in Request.

Headers is an object list to which MsgToInvContext adds TSOAPHeader descendants to represent every header in the encoded message.

See Also

InvContextToMsg (see page 18144)

119.1.2.4.5 TOPToSoapDomConvert.NewXMLDocument Method

Creates a new empty XML document to hold the SOAP representation and returns its interface.

Pascal

```
function NewXMLDocument: IXMLDocument;
```

C++

```
__fastcall IXMLDocument NewXMLDocument();
```

Description

TOPToSoapDomConvert (see page 18137) uses NewXMLDocument internally to create the object that holds the SOAP representation of a message. It returns an interface for working with this representation. Note that the XML document that NewXMLDocument returns is empty. TOPToSoapDomConvert (see page 18137) subsequently uses the methods of the interface to fill out the body of this document.

119.1.2.4.6 ProcessResponse Method

This is the overview for the ProcessResponse method overload.

Overload List

	Name	Description
	TOPToSoapDomConvert.ProcessResponse (InvString, TIntfMetaData, TIntfMethEntry, TInvContext) (see page 18147)	Unpacks the response packet from a method call into an invocation context.
	TOPToSoapDomConvert.ProcessResponse (IXMLDocument, TIntfMetaData, TIntfMethEntry, TInvContext, THeaderList) (see page 18147)	Unpacks the response packet from a method call into an invocation context.
	TOPToSoapDomConvert.ProcessResponse (TStream, TIntfMetaData, TIntfMethEntry, TInvContext, THeaderList) (see page 18147)	Unpacks the response packet from a method call into an invocation context.

119.1.2.4.6.1 TOPToSoapDomConvert.ProcessResponse Method (IXMLDocument, TIntfMetaData, TIntfMethEntry, TInvContext, THeaderList)

Unpacks the response packet from a method call into an invocation context.

Pascal

```
procedure ProcessResponse(const Resp: InvString; const IntfMD: TIntfMetaData; const MD:
TIntfMethEntry; Context: TInvContext); virtual; overload;
procedure ProcessResponse(const Resp: TStream; const IntfMD: TIntfMetaData; const MD:
TIntfMethEntry; Context: TInvContext; Headers: THeaderList); virtual; overload;
procedure ProcessResponse(const XMLDoc: IXMLDocument; const IntfMD: TIntfMetaData; const MD:
TIntfMethEntry; Context: TInvContext; Headers: THeaderList); virtual; overload;
```

C++

```
virtual __fastcall ProcessResponse(const InvString Resp, const TIntfMetaData IntfMD, const
TIntfMethEntry MD, TInvContext Context);
virtual __fastcall ProcessResponse(const TStream Resp, const TIntfMetaData IntfMD, const
TIntfMethEntry MD, TInvContext Context, THeaderList Headers);
virtual __fastcall ProcessResponse(const IXMLDocument XMLDoc, const TIntfMetaData IntfMD,
const TIntfMethEntry MD, TInvContext Context, THeaderList Headers);
```

Description

ProcessResponse takes the encoded results of executing a method on an invokable interface from an invocation context and unpacks them into an invocation context.

Response is the encoded results of executing a method on the invokable interface, or a stream from which those results can be read. It is generated on the server by the MakeResponse (see page 18145) method if the call was successful, or the MakeFault (see page 18145) method if the call was unsuccessful.

XMLDoc (see page 26095) is the interface to an XML document that represents the results of executing a method on the invokable interface. The values of its nodes come from the MakeResponse (see page 18145) method if the call was successful, or the MakeFault (see page 18145) method if the call was unsuccessful.

IntfMD describes the runtime type information (RTTI) of the interface whose method was called.

MD describes the runtime type information (RTTI) of the method that was called.

Context is the invocation context that ProcessResponse fills out based on the contents of Response.

Headers is a list that ProcessResponse fills with TSOAPHeader descendants that represent the headers of the encoded result message.

See Also

MakeFault (see page 18145)

MakeResponse (see page 18145)

119.1.2.5 TOPToSoapDomConvert Properties

The properties of the TOPToSoapDomConvert class are listed here.

Public Properties

	Name	Description
	Attachments (see page 18148)	Lists the objects that encapsulate the attachments included with the SOAP request or its response.

Published Properties

	Name	Description
	Encoding (see page 18149)	Indicates the character set to use for encoded method calls.
	TempDir (see page 18149)	Specifies the directory where temporary files for attachments are saved.
	WSDLView (see page 18150)	Provides access to information in a WSDL document that represents the SOAP request.

119.1.2.5.1 TOPToSoapDomConvert.Attachments Property

Lists the objects that encapsulate the attachments included with the SOAP request or its response.

Pascal

```
property Attachments: TSOAPDataList;
```

C++

```
__property TSOAPDataList Attachments;
```

Description

When a method on an invokable interface requires the user to send or receive attachments, these attachments are each represented by a TSoapAttachment object. Attachments lists the TSoapAttachment objects for each attachment of the current

method.

Note: When the application that contains TOPToSoapDomConvert (see page 18137) receives attachments, it writes them to a temporary file rather than storing them in memory. The location for these temporary files is specified by the TempDir (see page 18149) property.

See Also

TempDir (see page 18149)

TSOAPAttachment (see page 15583)

119.1.2.5.2 TOPToSoapDomConvert.Encoding Property

Indicates the character set to use for encoded method calls.

Pascal

```
property Encoding: WideString;
```

C++

```
__property BSTR Encoding;
```

Description

Use Encoding to indicate the character set that is used for encoded method calls. Encoding causes the converter to make character set conversions using the DOM, rather than at the transport level. This approach is more efficient than transport-level encoding. If you set Encoding, the Options (see page 18176) property should not include soUTF8EncodeXML and soUTF8InHeader.

If you do not specify an encoding (that is, if Encoding is an empty string), TOPToSoapDomConvert (see page 18137) converts characters at the transport level assuming UTF-8.

TOPToSoapDomConvert (see page 18137) uses the DOM parser identified by the global DefaultDOMVendor variable to perform its conversions. Encoding must indicate an encoding that this default DOM parser supports.

Note: Some Web Services (such as Apache or .NET) require transport level encoding rather than DOM level encoding. In such cases, it is possible that setting Encoding can cause problems because the Web Service converts characters at the transport level and then converts a second time because of the encoding attribute that Encoding adds to the XML representation. If a Web Service requires transport-level encoding and assumes UTF8, Encoding should be an empty string and the Options (see page 18176) property should include soUTF8EncodeXML and soUTF8InHeader.

See Also

DefaultDOMVendor

Encoding (see page 26406)

Options (see page 18176)

119.1.2.5.3 TOPToSoapDomConvert.TempDir Property

Specifies the directory where temporary files for attachments are saved.

Pascal

```
property TempDir: string;
```

C++

```
__property AnsiString TempDir;
```

Description

When the application that includes this TOPToSoapDomConvert (see page 18137) instance receives an attachment as the value of a parameter (or the return value) of the current method, TOPToSoapDomConvert (see page 18137) writes the attachment to a temporary file. This temporary file becomes the value of the CacheFile property on the TSoapAttachment object that represents the attachment.

Use TempDir to specify the directory where TOPToSoapDomConvert (see page 18137) writes the temporary files for attachments.

Note: If TempDir is not set, the application uses the default temporary directory. On Windows, this is the directory that the Windows API GetTempPath returns. On Linux, this is obtained from the TMPDIR environment setting (or, if TMPDIR is not set, /tmp/)

See Also

Attachments (see page 18148)

CacheFile (see page 15592)

119.1.2.5.4 TOPToSoapDomConvert.WSDLView Property

Provides access to information in a WSDL document that represents the SOAP request.

Pascal

```
property WSDLView: TWSDLView;
```

C++

```
__property TWSDLView WSDLView;
```

Description

WSDLView represents information about the SOAP request when the application uses a WSDL document to define information about the interface. If the application does not use a WSDL document, WSDLView is nil (Delphi) or NULL (C++).

WSDL includes the parsed WSDL document, as well as properties to identify the target method call.

119.1.3 TSOAPDomConv Class

TSOAPDomConv is the base class for components that use a DOM to interpret and create SOAP method calls.

Class Hierarchy

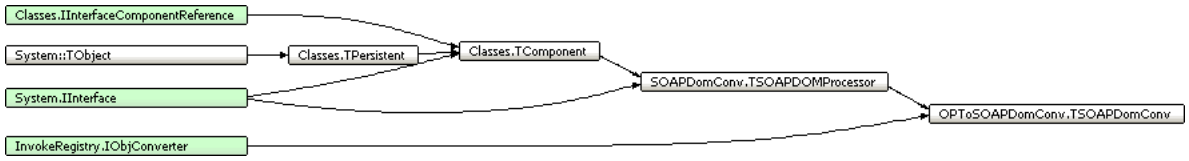

Pascal

```
TSOAPDomConv = class(TSOAPDOMProcessor, IObjConverter);
```

C++

```
class TSOAPDomConv : public TSOAPDOMProcessor, public IObjConverter;
```

File

OPToSOAPDomConv

Description

TSOAPDomConv is the base class for DOM-based components that interpret and create SOAP method calls. It handles the logic of converting between the SOAP representation of a type and its native representation. TSOAPDomConv uses the remotable type registry (RemTypeRegistry) to look up the native type or class that corresponds to a type from a SOAP method call and to generate an instance of that type to represent an incoming value. It uses runtime type information (RTTI) to convert a native type or remotable class to its SOAP representation.

TSOAPDomConv implements the IObjConverter interface.

Do not create instances of TSOAPDomConv. It is intended only as a base class for components that interpret or create SOAP method calls.

See Also

TOPToSoapDomConvert (see page 18137)

RemTypeRegistry

IObjConverter (see page 15610)

119.1.3.1 TSOAPDomConv Members

The following tables list the members exposed by TSOAPDomConv.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FHeaderProcessors (see page 19915)	This is FHeaderProcessors, a member of class TSOAPDOMProcessor.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_AddRef (see page 19916)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	_Release (see page 19917)	Called when an application releases a component interface.
	AddAttachment (see page 18157)	Stores information about a SOAP attachment so that it is included in the final message that is created.
	AddMultiRefNode (see page 18158)	This is AddMultiRefNode, a member of class TSOAPDomConv.
	AddNamespaceDecl (see page 18158)	This is AddNamespaceDecl, a member of class TSOAPDomConv.
	AddObjectAsWriting (see page 18158)	This is AddObjectAsWriting, a member of class TSOAPDomConv.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckEncodingStyle (see page 18158)	This is CheckEncodingStyle, a member of class TSOAPDomConv.
	ChildNodesAreNull (see page 18159)	This is ChildNodesAreNull, a member of class TSOAPDomConv.
	ConvertByteArrayToSoap (see page 18159)	This is ConvertByteArrayToSoap, a member of class TSOAPDomConv.
	ConvertEnumToSoap (see page 18159)	This is ConvertEnumToSoap, a member of class TSOAPDomConv.
	ConvertNativeArrayToSoap (see page 18159)	This is ConvertNativeArrayToSoap, a member of class TSOAPDomConv.
	ConvertObjectToSOAP (see page 18160)	This is ConvertObjectToSOAP, a member of class TSOAPDomConv.

	ConvertSoapToEnum (see page 18161)	This is ConvertSoapToEnum, a member of class TSOAPDomConv.
	ConvertSoapToNativeArray (see page 18161)	This is ConvertSoapToNativeArray, a member of class TSOAPDomConv.
	ConvertSoapToNativeArrayElem (see page 18161)	This is ConvertSoapToNativeArrayElem, a member of class TSOAPDomConv.
	ConvertSOAPToObject (see page 18162)	This is ConvertSOAPToObject, a member of class TSOAPDomConv.
	ConvertSoapToVariant (see page 18162)	This is ConvertSoapToVariant, a member of class TSOAPDomConv.
	ConvertVariantToSoap (see page 18162)	This is ConvertVariantToSoap, a member of class TSOAPDomConv.
	CreateMultiRefNode (see page 18163)	This is CreateMultiRefNode, a member of class TSOAPDomConv.
	CreateNULLNode (see page 18163)	This is CreateNULLNode, a member of class TSOAPDomConv.
	CreateObjectNode (see page 18163)	This is CreateObjectNode, a member of class TSOAPDomConv.
	CreateScalarNodeXS (see page 18163)	This is CreateScalarNodeXS, a member of class TSOAPDomConv.
	CreateTypedNode (see page 18164)	This is CreateTypedNode, a member of class TSOAPDomConv.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FinalizeMultiRefNodes (see page 18164)	This is FinalizeMultiRefNodes, a member of class TSOAPDomConv.
	FindAttachment (see page 18164)	Returns a TSOAPAttachment instance that represent the attachment with a specified ID.
	FindHeaderProcessor (see page 19919)	Locates a processor to interpret or generate a specified SOAP header.
	FindMultiRefNodeByID (see page 18164)	This is FindMultiRefNodeByID, a member of class TSOAPDomConv.
	FindMultiRefNodeByInstance (see page 18165)	This is FindMultiRefNodeByInstance, a member of class TSOAPDomConv.
	FindNodeByHREF (see page 18165)	This is FindNodeByHREF, a member of class TSOAPDomConv.
	FindPrefixForURI (see page 18165)	This is FindPrefixForURI, a member of class TSOAPDomConv.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDataNode (see page 18165)	This is GetDataNode, a member of class TSOAPDomConv.
	GetElementType (see page 18166)	This is GetElementType, a member of class TSOAPDomConv.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetNewID (see page 18166)	This is GetNewID, a member of class TSOAPDomConv.
	GetNodeAsText (see page 18166)	This is GetNodeAsText, a member of class TSOAPDomConv.
	GetObjectPropAsText (see page 18166)	This is GetObjectPropAsText, a member of class TSOAPDomConv.
	GetOptions (see page 18166)	This is GetOptions, a member of class TSOAPDomConv.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeBySchemaNS (see page 18167)	This is GetTypeBySchemaNS, a member of class TSOAPDomConv.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InitObjectFromSOAP (see page 18167)	Initializes an instance of a remotable object to reflect the values in its SOAP representation.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsNodeAVarArray (see page 18167)	This is IsNodeAVarArray, a member of class TSOAPDomConv.
	IsObjectWriting (see page 18168)	This is IsObjectWriting, a member of class TSOAPDomConv.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	LoadObject (see page 18168)	This is LoadObject, a member of class TSOAPDomConv.
	MakeArrayNode (see page 18168)	This is the overview for the MakeArrayNode method overload.
	MakeHeaderNodes (see page 19919)	Generates all the SOAP headers that this object knows how to create.
	MultiRefObject (see page 18169)	This is MultiRefObject, a member of class TSOAPDomConv.
	NodeIsNULL (see page 18169)	This is NodeIsNULL, a member of class TSOAPDomConv.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	ObjectMemberNoShow (see page 18169)	This is ObjectMemberNoShow, a member of class TSOAPDomConv.
	ObjInstanceToSOAP (see page 18169)	Returns the SOAP representation of a remotable object.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.

	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadHeader (see page 18170)	This is ReadHeader, a member of class TSOAPDomConv.
	ReadRectDynArray (see page 18170)	This is ReadRectDynArray, a member of class TSOAPDomConv.
	ReadRectDynArrayElem (see page 18171)	This is ReadRectDynArrayElem, a member of class TSOAPDomConv.
	ReadRow (see page 18171)	This is ReadRow, a member of class TSOAPDomConv.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadVarArrayDim (see page 18171)	This is ReadVarArrayDim, a member of class TSOAPDomConv.
	ReadVariant (see page 18171)	This is ReadVariant, a member of class TSOAPDomConv.
	RemoveObjectAsWriting (see page 18172)	This is RemoveObjectAsWriting, a member of class TSOAPDomConv.
	ResetMultiRef (see page 18172)	This is ResetMultiRef, a member of class TSOAPDomConv.
	SerializationOptions (see page 18172)	This is the overview for the SerializationOptions method overload.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetNodeType (see page 18173)	This is SetNodeType, a member of class TSOAPDomConv.
	SetObjectPropFromText (see page 18173)	This is SetObjectPropFromText, a member of class TSOAPDomConv.
	SetOptions (see page 18173)	This is SetOptions, a member of class TSOAPDomConv.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	UnhandledNode (see page 18174)	This is UnhandledNode, a member of class TSOAPDomConv.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteHeader (see page 18174)	This is WriteHeader, a member of class TSOAPDomConv.
	WriteNonRectDynArray (see page 18174)	This is WriteNonRectDynArray, a member of class TSOAPDomConv.
	WriteNonRectDynArrayElem (see page 18174)	This is WriteNonRectDynArrayElem, a member of class TSOAPDomConv.
	WriteRectDynArray (see page 18175)	This is WriteRectDynArray, a member of class TSOAPDomConv.
	WriteRectDynArrayElem (see page 18175)	This is WriteRectDynArrayElem, a member of class TSOAPDomConv.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.
	WriteVarArray (see page 18175)	This is WriteVarArray, a member of class TSOAPDomConv.
	WriteVarArrayAsB64 (see page 18175)	This is WriteVarArrayAsB64, a member of class TSOAPDomConv.
	WriteVariant (see page 18176)	This is WriteVariant, a member of class TSOAPDomConv.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddHeaderProcessor (see page 19917)	Adds a processor to the internal list of interfaces to use for interpreting and creating SOAP headers.
	AfterConstruction (see page 19918)	Responds after the last constructor executes.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ConvertNativeDataToSoap (see page 18160)	Converts a native type to its SOAP representation.
	ConvertSoapToNativeData (see page 18161)	Converts the SOAP representation of a value to a pointer to the corresponding native representation.
V	Create (see page 18155)	Creates a new TSOAPDomConv (see page 18150) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	DefaultProcessHeader (see page 19918)	Handles the interpretation of a SOAP header when no processor can be found.
V	Destroy (see page 18156)	Frees the memory associated with the TSOAPDomConv (see page 18150) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 19919)	Allocates memory for each instance of TSOAPDOMProcessor (see page 19912).
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.

V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnMemberDataNotReceived (see page 18177)	Occurs when converting a SOAP representation to a remotable object and an expected class member was not found.
	OnUnhandledNode (see page 18177)	Occurs when a SOAP representation includes an XML node that the converter does not know how to handle.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 18176)	Controls (see page 5711) how the TSOAPDomConv (see page 18150) object performs certain types of conversions.
	Tag (see page 2808)	Stores an integer value as part of a component.

119.1.3.2 TSOAPDomConv.Create Constructor

Creates a new TSOAPDomConv (see page 18150) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSOAPDomConv(TComponent * AOwner);
```

Description

Typically, applications do not need to create instances of TSOAPDomConv (see page 18150). Instead, it is created and used by THTTPRio on client applications and THTTPSoapPascalInvoker or THTTPSoapCppInvoker on server applications.

The AOwner parameter specifies the component that is responsible for freeing the TSOAPDomConv (see page 18150) instance.

See Also

Owner (see page 2807)

THTTPRio

THTTPSoapPascalInvoker (see page 19951)

THTTPSoapCppInvoker (see page 19950)

119.1.3.3 TSOAPDomConv.Destroy Destructor

Frees the memory associated with the TSOAPDomConv (see page 18150) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSOAPDomConv();
```

Description

Do not call Destroy to free an instance of TSOAPDomConv (see page 18150). Instead, use the Free method, which first checks for a nil reference before calling Destroy.

See Also

Free (see page 22300)

119.1.3.4 TSOAPDomConv Methods

The methods of the TSOAPDomConv class are listed here.

Protected Methods

	Name	Description
	AddAttachment (see page 18157)	Stores information about a SOAP attachment so that it is included in the final message that is created.
	AddMultiRefNode (see page 18158)	This is AddMultiRefNode, a member of class TSOAPDomConv.
	AddNamespaceDecl (see page 18158)	This is AddNamespaceDecl, a member of class TSOAPDomConv.
	AddObjectAsWriting (see page 18158)	This is AddObjectAsWriting, a member of class TSOAPDomConv.
	CheckEncodingStyle (see page 18158)	This is CheckEncodingStyle, a member of class TSOAPDomConv.
	ChildNodesAreNull (see page 18159)	This is ChildNodesAreNull, a member of class TSOAPDomConv.
	ConvertByteArrayToSoap (see page 18159)	This is ConvertByteArrayToSoap, a member of class TSOAPDomConv.
	ConvertEnumToSoap (see page 18159)	This is ConvertEnumToSoap, a member of class TSOAPDomConv.
	ConvertNativeArrayToSoap (see page 18159)	This is ConvertNativeArrayToSoap, a member of class TSOAPDomConv.
	ConvertObjectToSOAP (see page 18160)	This is ConvertObjectToSOAP, a member of class TSOAPDomConv.
	ConvertSoapToEnum (see page 18161)	This is ConvertSoapToEnum, a member of class TSOAPDomConv.
	ConvertSoapToNativeArray (see page 18161)	This is ConvertSoapToNativeArray, a member of class TSOAPDomConv.
	ConvertSoapToNativeArrayElem (see page 18161)	This is ConvertSoapToNativeArrayElem, a member of class TSOAPDomConv.
	ConvertSOAPToObject (see page 18162)	This is ConvertSOAPToObject, a member of class TSOAPDomConv.
	ConvertSoapToVariant (see page 18162)	This is ConvertSoapToVariant, a member of class TSOAPDomConv.
	ConvertVariantToSoap (see page 18162)	This is ConvertVariantToSoap, a member of class TSOAPDomConv.
	CreateMultiRefNode (see page 18163)	This is CreateMultiRefNode, a member of class TSOAPDomConv.
	CreateNULLNode (see page 18163)	This is CreateNULLNode, a member of class TSOAPDomConv.
	CreateObjectNode (see page 18163)	This is CreateObjectNode, a member of class TSOAPDomConv.
	CreateScalarNodeXS (see page 18163)	This is CreateScalarNodeXS, a member of class TSOAPDomConv.
	CreateTypedNode (see page 18164)	This is CreateTypedNode, a member of class TSOAPDomConv.
	FinalizeMultiRefNodes (see page 18164)	This is FinalizeMultiRefNodes, a member of class TSOAPDomConv.
	FindAttachment (see page 18164)	Returns a TSOAPAttachment instance that represent the attachment with a specified ID.
	FindMultiRefNodeByID (see page 18164)	This is FindMultiRefNodeByID, a member of class TSOAPDomConv.
	FindMultiRefNodeByInstance (see page 18165)	This is FindMultiRefNodeByInstance, a member of class TSOAPDomConv.
	FindNodeByHREF (see page 18165)	This is FindNodeByHREF, a member of class TSOAPDomConv.

	FindPrefixForURI (see page 18165)	This is FindPrefixForURI, a member of class TSOAPDomConv.
	GetDataNode (see page 18165)	This is GetDataNode, a member of class TSOAPDomConv.
	GetElementType (see page 18166)	This is GetElementType, a member of class TSOAPDomConv.
	GetNewID (see page 18166)	This is GetNewID, a member of class TSOAPDomConv.
	GetNodeAsText (see page 18166)	This is GetNodeAsText, a member of class TSOAPDomConv.
	GetObjectPropAsText (see page 18166)	This is GetObjectPropAsText, a member of class TSOAPDomConv.
	GetOptions (see page 18166)	This is GetOptions, a member of class TSOAPDomConv.
	GetTypeBySchemaNS (see page 18167)	This is GetTypeBySchemaNS, a member of class TSOAPDomConv.
	InitObjectFromSOAP (see page 18167)	Initializes an instance of a remotable object to reflect the values in its SOAP representation.
	IsNodeAVarArray (see page 18167)	This is IsNodeAVarArray, a member of class TSOAPDomConv.
	IsObjectWriting (see page 18168)	This is IsObjectWriting, a member of class TSOAPDomConv.
	LoadObject (see page 18168)	This is LoadObject, a member of class TSOAPDomConv.
	MakeArrayNode (see page 18168)	This is the overview for the MakeArrayNode method overload.
	MultiRefObject (see page 18169)	This is MultiRefObject, a member of class TSOAPDomConv.
	NodeIsNULL (see page 18169)	This is NodeIsNULL, a member of class TSOAPDomConv.
	ObjectMemberNoShow (see page 18169)	This is ObjectMemberNoShow, a member of class TSOAPDomConv.
	ObjInstanceToSOAP (see page 18169)	Returns the SOAP representation of a remotable object.
	ReadHeader (see page 18170)	This is ReadHeader, a member of class TSOAPDomConv.
	ReadRectDynArray (see page 18170)	This is ReadRectDynArray, a member of class TSOAPDomConv.
	ReadRectDynArrayElem (see page 18171)	This is ReadRectDynArrayElem, a member of class TSOAPDomConv.
	ReadRow (see page 18171)	This is ReadRow, a member of class TSOAPDomConv.
	ReadVarArrayDim (see page 18171)	This is ReadVarArrayDim, a member of class TSOAPDomConv.
	ReadVariant (see page 18171)	This is ReadVariant, a member of class TSOAPDomConv.
	RemoveObjectAsWriting (see page 18172)	This is RemoveObjectAsWriting, a member of class TSOAPDomConv.
	ResetMultiRef (see page 18172)	This is ResetMultiRef, a member of class TSOAPDomConv.
	SerializationOptions (see page 18172)	This is the overview for the SerializationOptions method overload.
	SetNodeType (see page 18173)	This is SetNodeType, a member of class TSOAPDomConv.
	SetObjectPropFromText (see page 18173)	This is SetObjectPropFromText, a member of class TSOAPDomConv.
	SetOptions (see page 18173)	This is SetOptions, a member of class TSOAPDomConv.
	UnhandledNode (see page 18174)	This is UnhandledNode, a member of class TSOAPDomConv.
	WriteHeader (see page 18174)	This is WriteHeader, a member of class TSOAPDomConv.
	WriteNonRectDynArray (see page 18174)	This is WriteNonRectDynArray, a member of class TSOAPDomConv.
	WriteNonRectDynArrayElem (see page 18174)	This is WriteNonRectDynArrayElem, a member of class TSOAPDomConv.
	WriteRectDynArray (see page 18175)	This is WriteRectDynArray, a member of class TSOAPDomConv.
	WriteRectDynArrayElem (see page 18175)	This is WriteRectDynArrayElem, a member of class TSOAPDomConv.
	WriteVarArray (see page 18175)	This is WriteVarArray, a member of class TSOAPDomConv.
	WriteVarArrayAsB64 (see page 18175)	This is WriteVarArrayAsB64, a member of class TSOAPDomConv.
	WriteVariant (see page 18176)	This is WriteVariant, a member of class TSOAPDomConv.

Public Methods

	Name	Description
	ConvertNativeDataToSoap (see page 18160)	Converts a native type to its SOAP representation.
	ConvertSoapToNativeData (see page 18161)	Converts the SOAP representation of a value to a pointer to the corresponding native representation.

119.1.3.4.1 TSOAPDomConv.AddAttachment Method

Stores information about a SOAP attachment so that it is included in the final message that is created.

Pascal

```
procedure AddAttachment(Attachment: TSOAPAttachment; const AContentId: string);
```

C++

```
__fastcall AddAttachment(TSOAPAttachment Attachment, const AnsiString AContentId);
```

Description

AddAttachment stores a SOAP attachment so that it can be used to generate part of the multipart form that includes the SOAP envelope as another of its parts.

Attachment is the attachment that should be added to the multipart form.

AContentId is a string that uniquely identifies the attachment. It is added as a header to the part of the form that contains the attachment so that XML in the SOAP envelope can refer to the attachment. When interpreting the encoded SOAP envelope, this value is passed to the FindAttachment method so that the information in the attachment part can be used to recreate the TSOAPAttachment instance.

See Also

FindAttachment (see page 15612)

119.1.3.4.2 TSOAPDomConv.AddMultiRefNode Method

Pascal

```
procedure AddMultiRefNode(const ID: string; Instance: Pointer);
```

C++

```
__fastcall AddMultiRefNode(const AnsiString ID, void * Instance);
```

Description

This is AddMultiRefNode, a member of class TSOAPDomConv.

119.1.3.4.3 TSOAPDomConv.AddNamespaceDecl Method

Pascal

```
function AddNamespaceDecl(Node: IXMLNode; const URI: InvString): InvString;
```

C++

```
__fastcall InvString AddNamespaceDecl(IXMLNode Node, const InvString URI);
```

Description

This is AddNamespaceDecl, a member of class TSOAPDomConv.

119.1.3.4.4 TSOAPDomConv.AddObjectAsWriting Method

Pascal

```
procedure AddObjectAsWriting(Instance: TObject);
```

C++

```
__fastcall AddObjectAsWriting(TObject * Instance);
```

Description

This is AddObjectAsWriting, a member of class TSOAPDomConv.

119.1.3.4.5 TSOAPDomConv.CheckEncodingStyle Method

Pascal

```
procedure CheckEncodingStyle(Node: IXMLNode);
```

C++

```
__fastcall CheckEncodingStyle(IXMLNode Node);
```

Description

This is CheckEncodingStyle, a member of class TSOAPDomConv.

119.1.3.4.6 TSOAPDomConv.ChildNodesAreNull Method

Pascal

```
function ChildNodesAreNull(Node: IXMLNode): Boolean;
```

C++

```
__fastcall Boolean ChildNodesAreNull(IXMLNode Node);
```

Description

This is ChildNodesAreNull, a member of class TSOAPDomConv.

119.1.3.4.7 TSOAPDomConv.ConvertByteArrayToSoap Method

Pascal

```
procedure ConvertByteArrayToSoap(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString;
Info: PTypeInfo; P: Pointer);
```

C++

```
__fastcall ConvertByteArrayToSoap(IXMLNode RootNode, IXMLNode Node, const InvString Name,
PTypeInfo Info, void * P);
```

Description

This is ConvertByteArrayToSoap, a member of class TSOAPDomConv.

119.1.3.4.8 TSOAPDomConv.ConvertEnumToSoap Method

Pascal

```
function ConvertEnumToSoap(Info: PTypeInfo; P: Pointer; NumIndirect: Integer): InvString;
```

C++

```
__fastcall InvString ConvertEnumToSoap(PTypeInfo Info, void * P, int NumIndirect);
```

Description

This is ConvertEnumToSoap, a member of class TSOAPDomConv.

119.1.3.4.9 TSOAPDomConv.ConvertNativeArrayToSoap Method

Pascal

```
procedure ConvertNativeArrayToSoap(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString;
Info: PTypeInfo; P: Pointer; NumIndirect: Integer; InlineElements: Boolean = False);
```

C++

```
__fastcall ConvertNativeArrayToSoap(IXMLNode RootNode, IXMLNode Node, const InvString Name,
PTypeInfo Info, void * P, int NumIndirect, Boolean InlineElements = False);
```

Description

This is ConvertNativeArrayToSoap, a member of class TSOAPDomConv.

119.1.3.4.10 TSOAPDomConv.ConvertNativeDataToSoap Method

Converts a native type to its SOAP representation.

Pascal

```
procedure ConvertNativeDataToSoap(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString;
Info: PTypeInfo; P: Pointer; NumIndirect: Integer); dynamic;
```

C++

```
__fastcall ConvertNativeDataToSoap(IXMLNode RootNode, IXMLNode Node, const InvString Name,
PTypeInfo Info, void * P, int NumIndirect);
```

Description

ConvertNativeDataToSoap performs the conversion from a native type to its SOAP representation and adds that to the emerging DOM representation of a SOAP packet. It uses the options specified by the Options (see page 18176) property to determine whether to include type data and how to handle multiple references to the same remotable object.

RootNode is the node in the emerging SOAP representation that acts as a parent to any elements that represent an object that has multiple references.

Node is the parent node to which ConvertNativeDataToSoap adds a new child node that represents the native value.

Name is the type name of the SOAP encoding of the native type.

Info is a TypeInfo pointer that indicates the type in which the native data appears.

P is a pointer to the value to convert. It points (possibly with additional levels of indirection) to a value of the type indicated by Info.

NumIndirect indicates the number of levels of indirection by which P points to a value of the type indicated by Info.

See Also

ConvertSoapToNativeData (see page 18161)

Options (see page 18176)

119.1.3.4.11 TSOAPDomConv.ConvertObjectToSOAP Method

Pascal

```
procedure ConvertObjectToSOAP(const Name: InvString; ObjP: Pointer; RootNode: IXMLNOde; Node:
IXMLNOde; NumIndirect: Integer);
```

C++

```
__fastcall ConvertObjectToSOAP(const InvString Name, void * ObjP, IXMLNOde RootNode, IXMLNOde
Node, int NumIndirect);
```

Description

This is ConvertObjectToSOAP, a member of class TSOAPDomConv.

119.1.3.4.12 TSOAPDomConv.ConvertSoapToEnum Method

Pascal

```
function ConvertSoapToEnum(Info: PTypeInfo; S: InvString; IsNull: Boolean): Integer;
```

C++

```
__fastcall int ConvertSoapToEnum(PTypeInfo Info, InvString S, Boolean IsNull);
```

Description

This is ConvertSoapToEnum, a member of class TSOAPDomConv.

119.1.3.4.13 TSOAPDomConv.ConvertSoapToNativeArray Method

Pascal

```
function ConvertSoapToNativeArray(DataP: Pointer; TypeInfo: PTypeInfo; RootNode: IXMLNode;
Node: IXMLNode): Pointer;
```

C++

```
__fastcall void * ConvertSoapToNativeArray(void * DataP, PTypeInfo TypeInfo, IXMLNode
RootNode, IXMLNode Node);
```

Description

This is ConvertSoapToNativeArray, a member of class TSOAPDomConv.

119.1.3.4.14 TSOAPDomConv.ConvertSoapToNativeArrayElem Method

Pascal

```
function ConvertSoapToNativeArrayElem(ArrayInfo: PTypeInfo; ElemInfo: PTypeInfo; RootNode:
IXMLNode; Node: IXMLNode; ArrayDesc: TSOAPArrayDesc; Dims: Integer; CurDim: Integer; DataP:
Pointer): Pointer;
```

C++

```
__fastcall void * ConvertSoapToNativeArrayElem(PTypeInfo ArrayInfo, PTypeInfo ElemInfo,
IXMLNode RootNode, IXMLNode Node, TSOAPArrayDesc ArrayDesc, int Dims, int CurDim, void *
DataP);
```

Description

This is ConvertSoapToNativeArrayElem, a member of class TSOAPDomConv.

119.1.3.4.15 TSOAPDomConv.ConvertSoapToNativeData Method

Converts the SOAP representation of a value to a pointer to the corresponding native representation.

Pascal

```
procedure ConvertSoapToNativeData(DataP: Pointer; TypeInfo: PTypeInfo; Context: TDataContext;
RootNode: IXMLNode; Node: IXMLNode; Translate: Boolean; ByRef: Boolean; NumIndirect: Integer);
dynamic;
```

C++

```
__fastcall ConvertSoapToNativeData(void * DataP, PTypeInfo TypeInfo, TDataContext Context,
IXMLNode RootNode, IXMLNode Node, Boolean Translate, Boolean ByRef, int NumIndirect);
```

Description

ConvertSoapToNativeData performs the conversion from a DOM node that represents a SOAP encoding of a value to the

corresponding native representation.

DataP is a pointer to the memory allocated to hold the native representation of the value, and points to the results of the conversion.

TypeInfo is the typeinfo pointer that describes the native type that will represent the value.

Context is the TDataContext object that stores the value.

RootNode is the node in the SOAP packet that acts as a parent to any elements to which there are multiple references.

Node is the node in the SOAP packet whose value is to be translated.

Translate indicates whether Node represents a value to be translated, as opposed to a reference to value that is found elsewhere in the SOAP packet.

ByRef indicates whether the node corresponds to a parameter that is passed by reference.

NumIndirect indicates the number of levels of indirection by which DataP points to its value.

See Also

ConvertNativeDataToSoap (see page 18160)

Options (see page 18176)

119.1.3.4.16 TSOAPDomConv.ConvertSOAPToObject Method

Pascal

```
function ConvertSOAPToObject(RootNode: IXMLNode; Node: IXMLNode; AClass: TClass; const URI:
WideString; const TypeName: WideString; ObjP: Pointer; NumIndirect: Integer): TObject;
```

C++

```
__fastcall TObject * ConvertSOAPToObject(IXMLNode RootNode, IXMLNode Node, TClass AClass,
const BSTR URI, const BSTR TypeName, void * ObjP, int NumIndirect);
```

Description

This is ConvertSOAPToObject, a member of class TSOAPDomConv.

119.1.3.4.17 TSOAPDomConv.ConvertSoapToVariant Method

Pascal

```
procedure ConvertSoapToVariant(Node: IXMLNode; InvData: Pointer);
```

C++

```
__fastcall ConvertSoapToVariant(IXMLNode Node, void * InvData);
```

Description

This is ConvertSoapToVariant, a member of class TSOAPDomConv.

119.1.3.4.18 TSOAPDomConv.ConvertVariantToSoap Method

Pascal

```
procedure ConvertVariantToSoap(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString;
Info: PTypeInfo; P: PVarData; NumIndirect: Integer; V: Variant; UseVariant: Boolean);
```

C++

```
__fastcall ConvertVariantToSoap(IXMLNode RootNode, IXMLNode Node, const InvString Name,
```

```
PTypeInfo Info, PVarData P, int NumIndirect, Variant V, Boolean UseVariant);
```

Description

This is ConvertVariantToSoap, a member of class TSOAPDomConv.

119.1.3.4.19 TSOAPDomConv.CreateMultiRefNode Method

Pascal

```
function CreateMultiRefNode(RootNode: IXMLNode; const Name: InvString; const ID: InvString):
IXMLNode;
```

C++

```
__fastcall IXMLNode CreateMultiRefNode(IXMLNode RootNode, const InvString Name, const
InvString ID);
```

Description

This is CreateMultiRefNode, a member of class TSOAPDomConv.

119.1.3.4.20 TSOAPDomConv.CreateNULLNode Method

Pascal

```
function CreateNULLNode(RootNode: IXMLNode; ParentNode: IXMLNode; const Name: InvString;
UseParentNode: Boolean = False): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateNULLNode(IXMLNode RootNode, IXMLNode ParentNode, const InvString
Name, Boolean UseParentNode = False);
```

Description

This is CreateNULLNode, a member of class TSOAPDomConv.

119.1.3.4.21 TSOAPDomConv.CreateObjectNode Method

Pascal

```
function CreateObjectNode(Instance: TObject; RootNode: IXMLNode; ParentNode: IXMLNode; const
Name: InvString; const URI: InvString; ObjConvOpts: TObjectConvertOptions): InvString;
```

C++

```
__fastcall InvString CreateObjectNode(TObject * Instance, IXMLNode RootNode, IXMLNode
ParentNode, const InvString Name, const InvString URI, TObjectConvertOptions ObjConvOpts);
```

Description

This is CreateObjectNode, a member of class TSOAPDomConv.

119.1.3.4.22 TSOAPDomConv.CreateScalarNodeXS Method

Pascal

```
function CreateScalarNodeXS(RootNode: IXMLNode; ParentNode: IXMLNode; const NodeName:
WideString; const URI: WideString; const TypeName: WideString; const Value: WideString;
GenPre: Boolean = False): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateScalarNodeXS(IXMLNode RootNode, IXMLNode ParentNode, const BSTR
NodeName, const BSTR URI, const BSTR TypeName, const BSTR Value, Boolean GenPre = False);
```

Description

This is CreateScalarNodeXS, a member of class TSOAPDomConv.

119.1.3.4.23 TSOAPDomConv.CreateTypedNode Method

Pascal

```
function CreateTypedNode(RootNode: IXMLNode; ParentNode: IXMLNode; const NodeName: WideString;
const URI: WideString; TypeName: WideString; GenPre: Boolean = False): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateTypedNode(IXMLNode RootNode, IXMLNode ParentNode, const BSTR
NodeName, const BSTR URI, BSTR TypeName, Boolean GenPre = False);
```

Description

This is CreateTypedNode, a member of class TSOAPDomConv.

119.1.3.4.24 TSOAPDomConv.FinalizeMultiRefNodes Method

Pascal

```
procedure FinalizeMultiRefNodes;
```

C++

```
__fastcall FinalizeMultiRefNodes();
```

Description

This is FinalizeMultiRefNodes, a member of class TSOAPDomConv.

119.1.3.4.25 TSOAPDomConv.FindAttachment Method

Returns a TSOAPAttachment instance that represent the attachment with a specified ID.

Pascal

```
function FindAttachment(const AContentId: string): TSOAPAttachment;
```

C++

```
__fastcall TSOAPAttachment FindAttachment(const AnsiString AContentId);
```

Description

FindAttachment locates the attachment that is identified by a specified content ID (HREF).

AContentId is the unique identifier for the attachment that was supplied to the AddAttachment method when the attachment was first added to the multipart form.

FindAttachment returns a TSOAPAttachment instance that represents the attachment.

See Also

AddAttachment (see page 15611)

119.1.3.4.26 TSOAPDomConv.FindMultiRefNodeByID Method

Pascal

```
function FindMultiRefNodeByID(const ID: string): Pointer;
```

C++

```
__fastcall void * FindMultiRefNodeByID(const AnsiString ID);
```

Description

This is FindMultiRefNodeByID, a member of class TSOAPDomConv.

119.1.3.4.27 TSOAPDomConv.FindMultiRefNodeByInstance Method

Pascal

```
function FindMultiRefNodeByInstance(Instance: Pointer): string;
```

C++

```
__fastcall AnsiString FindMultiRefNodeByInstance(void * Instance);
```

Description

This is FindMultiRefNodeByInstance, a member of class TSOAPDomConv.

119.1.3.4.28 TSOAPDomConv.FindNodeByHREF Method

Pascal

```
function FindNodeByHREF(RootNode: IXMLNode; const HREF: InvString): IXMLNode;
```

C++

```
__fastcall IXMLNode FindNodeByHREF(IXMLNode RootNode, const InvString HREF);
```

Description

This is FindNodeByHREF, a member of class TSOAPDomConv.

119.1.3.4.29 TSOAPDomConv.FindPrefixForURI Method

Pascal

```
function FindPrefixForURI(RootNode: IXMLNode; Node: IXMLNode; const URI: InvString;
DeclIfNone: Boolean = False): InvString;
```

C++

```
__fastcall InvString FindPrefixForURI(IXMLNode RootNode, IXMLNode Node, const InvString URI,
Boolean DeclIfNone = False);
```

Description

This is FindPrefixForURI, a member of class TSOAPDomConv.

119.1.3.4.30 TSOAPDomConv.GetDataNode Method

Pascal

```
function GetDataNode(RootNode: IXMLNode; Node: IXMLNode; var ID: InvString): IXMLNode;
```

C++

```
__fastcall IXMLNode GetDataNode(IXMLNode RootNode, IXMLNode Node, InvString ID);
```

Description

This is GetDataNode, a member of class TSOAPDomConv.

119.1.3.4.31 TSOAPDomConv.GetElementType Method

Pascal

```
function GetElementType(Node: IXMLNode; var TypeURI: InvString; var TypeName: InvString):
Boolean;
```

C++

```
__fastcall Boolean GetElementType(IXMLNode Node, InvString TypeURI, InvString TypeName);
```

Description

This is GetElementType, a member of class TSOAPDomConv.

119.1.3.4.32 TSOAPDomConv.GetNewID Method

Pascal

```
function GetNewID: string;
```

C++

```
__fastcall AnsiString GetNewID();
```

Description

This is GetNewID, a member of class TSOAPDomConv.

119.1.3.4.33 TSOAPDomConv.GetNodeAsText Method

Pascal

```
function GetNodeAsText(Node: IXMLNode): InvString;
```

C++

```
__fastcall InvString GetNodeAsText(IXMLNode Node);
```

Description

This is GetNodeAsText, a member of class TSOAPDomConv.

119.1.3.4.34 TSOAPDomConv.GetObjectPropAsText Method

Pascal

```
function GetObjectPropAsText(Instance: TObject; PropInfo: PPropInfo): WideString;
```

C++

```
__fastcall BSTR GetObjectPropAsText(TObject * Instance, PPropInfo PropInfo);
```

Description

This is GetObjectPropAsText, a member of class TSOAPDomConv.

119.1.3.4.35 TSOAPDomConv.GetOptions Method

Pascal

```
function GetOptions: TSOAPConvertOptions;
```

C++

```
__fastcall TSOAPConvertOptions GetOptions();
```

Description

This is GetOptions, a member of class TSOAPDomConv.

119.1.3.4.36 TSOAPDomConv.GetTypeBySchemaNS Method

Pascal

```
function GetTypeBySchemaNS(Node: IXMLNode; const URI: InvString): Variant;
```

C++

```
__fastcall Variant GetTypeBySchemaNS(IXMLNode Node, const InvString URI);
```

Description

This is GetTypeBySchemaNS, a member of class TSOAPDomConv.

119.1.3.4.37 TSOAPDomConv.InitObjectFromSOAP Method

Initializes an instance of a remotable object to reflect the values in its SOAP representation.

Pascal

```
procedure InitObjectFromSOAP(Instance: TObject; RootNode: IXMLNode; Node: IXMLNode);
```

C++

```
__fastcall InitObjectFromSOAP(TObject * Instance, IXMLNode RootNode, IXMLNode Node);
```

Description

InitObjectFromSOAP assigns values to the published properties of a remotable object that represents a node from the SOAP envelope.

Instance is the remotable object to initialize from its SOAP representation.

RootNode is an ancestor node of the node for the remotable object that can contain information needed for initializing the remotable object.

Node is the node in the SOAP envelope that represents the remotable object.

See Also

ObjInstanceToSOAP (see page 15612)

119.1.3.4.38 TSOAPDomConv.IsNodeAVarArray Method

Pascal

```
function IsNodeAVarArray(const Node: IXMLNode; var VT: TVarType): Boolean;
```

C++

```
__fastcall Boolean IsNodeAVarArray(const IXMLNode Node, TVarType VT);
```

Description

This is IsNodeAVarArray, a member of class TSOAPDomConv.

119.1.3.4.39 TSOAPDomConv.IsObjectWriting Method

Pascal

```
function IsObjectWriting(Instance: TObject): Boolean;
```

C++

```
__fastcall Boolean IsObjectWriting(TObject * Instance);
```

Description

This is IsObjectWriting, a member of class TSOAPDomConv.

119.1.3.4.40 TSOAPDomConv.LoadObject Method

Pascal

```
procedure LoadObject(Instance: TObject; RootNode: IXMLNode; Node: IXMLNode);
```

C++

```
__fastcall LoadObject(TObject * Instance, IXMLNode RootNode, IXMLNode Node);
```

Description

This is LoadObject, a member of class TSOAPDomConv.

119.1.3.4.41 MakeArrayNode Method

This is the overview for the MakeArrayNode method overload.

Overload List

	Name	Description
	TSOAPDomConv.MakeArrayNode (IXMLNode, IXMLNode, InvString, InvString, InvString, array of Integer) (see page 18169)	This is MakeArrayNode, a member of class TSOAPDomConv.
	TSOAPDomConv.MakeArrayNode (IXMLNode, IXMLNode, InvString, InvString, InvString, Integer, Integer) (see page 18168)	This is MakeArrayNode, a member of class TSOAPDomConv.

119.1.3.4.41.1 TSOAPDomConv.MakeArrayNode Method (IXMLNode, IXMLNode, InvString, InvString, InvString, Integer, Integer)

Pascal

```
function MakeArrayNode(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString; const URI:
InvString; const TypeName: InvString; Dim: Integer; Len: Integer): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode MakeArrayNode(IXMLNode RootNode, IXMLNode Node, const InvString Name,
const InvString URI, const InvString TypeName, int Dim, int Len);
```

Description

This is MakeArrayNode, a member of class TSOAPDomConv.

119.1.3.4.41.2 TSOAPDomConv.MakeArrayNode Method (IXMLNode, IXMLNode, InvString, InvString, InvString, array of Integer)

Pascal

```
function MakeArrayNode(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString; const URI:
InvString; const TypeName: InvString; Indices: array of Integer): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode MakeArrayNode(IXMLNode RootNode, IXMLNode Node, const InvString Name,
const InvString URI, const InvString TypeName, array of Integer Indices);
```

Description

This is MakeArrayNode, a member of class TSOAPDomConv.

119.1.3.4.42 TSOAPDomConv.MultiRefObject Method

Pascal

```
function MultiRefObject(Cls: TClass): Boolean;
```

C++

```
__fastcall Boolean MultiRefObject(TClass Cls);
```

Description

This is MultiRefObject, a member of class TSOAPDomConv.

119.1.3.4.43 TSOAPDomConv.NodeIsNULL Method

Pascal

```
function NodeIsNULL(Node: IXMLNode): Boolean;
```

C++

```
__fastcall Boolean NodeIsNULL(IXMLNode Node);
```

Description

This is NodeIsNULL, a member of class TSOAPDomConv.

119.1.3.4.44 TSOAPDomConv.ObjectMemberNoShow Method

Pascal

```
procedure ObjectMemberNoShow(const ClassName: string; const MemberName: string);
```

C++

```
__fastcall ObjectMemberNoShow(const AnsiString ClassName, const AnsiString MemberName);
```

Description

This is ObjectMemberNoShow, a member of class TSOAPDomConv.

119.1.3.4.45 TSOAPDomConv.ObjInstanceToSOAP Method

Returns the SOAP representation of a remotable object.

Pascal

```
function ObjInstanceToSOAP(Instance: TObject; RootNode: IXMLNode; ParentNode: IXMLNode; const
NodeName: InvString; const NodeNamespace: InvString; ObjConvOpts: TObjectConvertOptions; out
RefID: InvString): IXMLNode;
```

C++

```
__fastcall IXMLNode ObjInstanceToSOAP(TObject * Instance, IXMLNode RootNode, IXMLNode
ParentNode, const InvString NodeName, const InvString NodeNamespace, TObjectConvertOptions
ObjConvOpts, InvString RefID);
```

Description

ObjInstanceToSOAP adds a node to represent the value of a remotable object.

Instance is the remotable object to encode as its SOAP representation.

RootNode is an ancestor of the node to which information about the node can be added as attributes.

ParentNode is the node in the SOAP representation that should act as a parent to the node that ObjInstanceToSOAP generates.

NodeName is the name of the node that is generated.

NodeNamespace is the URI for the namespace in which NodeName is defined.

ObjConvOpts are flags that customize the way the node is serialized.

RefID returns a reference ID for the node. This is used if the node is multi-referenced.

ObjInstanceToSOAP returns the XML for the newly generated node.

See Also

InitObjectFromSOAP (see page 15612)

119.1.3.4.46 TSOAPDomConv.ReadHeader Method

Pascal

```
procedure ReadHeader(const EnvNode: IXMLNode; const HdrNode: IXMLNode; Headers: THeaderList);
```

C++

```
__fastcall ReadHeader(const IXMLNode EnvNode, const IXMLNode HdrNode, THeaderList Headers);
```

Description

This is ReadHeader, a member of class TSOAPDomConv.

119.1.3.4.47 TSOAPDomConv.ReadRectDynArray Method

Pascal

```
procedure ReadRectDynArray(RootNode: IXMLNode; Node: IXMLNode; Info: PTypeInfo; Dims: Integer;
P: Pointer; CurElem: Integer);
```

C++

```
__fastcall ReadRectDynArray(IXMLNode RootNode, IXMLNode Node, PTypeInfo Info, int Dims, void *
P, int CurElem);
```

Description

This is ReadRectDynArray, a member of class TSOAPDomConv.

119.1.3.4.48 TSOAPDomConv.ReadRectDynArrayElem Method

Pascal

```
procedure ReadRectDynArrayElem(RootNode: IXMLNode; Node: IXMLNode; Info: PTypeInfo; Size:
Integer; Dim: Integer; P: Pointer; var CurElem: Integer);
```

C++

```
__fastcall ReadRectDynArrayElem(IXMLNode RootNode, IXMLNode Node, PTypeInfo Info, int Size,
int Dim, void * P, int CurElem);
```

Description

This is ReadRectDynArrayElem, a member of class TSOAPDomConv.

119.1.3.4.49 TSOAPDomConv.ReadRow Method

Pascal

```
procedure ReadRow(RootNode: IXMLNode; Node: IXMLNode; var CurElem: Integer; Size: Integer; P:
Pointer; Info: PTypeInfo);
```

C++

```
__fastcall ReadRow(IXMLNode RootNode, IXMLNode Node, int CurElem, int Size, void * P,
PTypeInfo Info);
```

Description

This is ReadRow, a member of class TSOAPDomConv.

119.1.3.4.50 TSOAPDomConv.ReadVarArrayDim Method

Pascal

```
function ReadVarArrayDim(Node: IXMLNode; IsVarVArray: Boolean = False; VT: TVarType = 0):
Variant;
```

C++

```
__fastcall Variant ReadVarArrayDim(IXMLNode Node, Boolean IsVarVArray = False, TVarType VT =
0);
```

Description

This is ReadVarArrayDim, a member of class TSOAPDomConv.

119.1.3.4.51 TSOAPDomConv.ReadVariant Method

Pascal

```
procedure ReadVariant(Node: IXMLNode; P: Pointer);
```

C++

```
__fastcall ReadVariant(IXMLNode Node, void * P);
```

Description

This is ReadVariant, a member of class TSOAPDomConv.

119.1.3.4.52 TSOAPDomConv.RemoveObjectAsWriting Method

Pascal

```
procedure RemoveObjectAsWriting(Instance: TObject);
```

C++

```
__fastcall RemoveObjectAsWriting(TObject * Instance);
```

Description

This is RemoveObjectAsWriting, a member of class TSOAPDomConv.

119.1.3.4.53 TSOAPDomConv.ResetMultiRef Method

Pascal

```
procedure ResetMultiRef;
```

C++

```
__fastcall ResetMultiRef();
```

Description

This is ResetMultiRef, a member of class TSOAPDomConv.

119.1.3.4.54 SerializationOptions Method

This is the overview for the SerializationOptions method overload.

Overload List

	Name	Description
	TSOAPDomConv.SerializationOptions (PTypeInfo) (see page 18172)	This is SerializationOptions, a member of class TSOAPDomConv.
	TSOAPDomConv.SerializationOptions (TClass) (see page 18172)	This is SerializationOptions, a member of class TSOAPDomConv.
	TSOAPDomConv.SerializationOptions (TObject) (see page 18173)	This is SerializationOptions, a member of class TSOAPDomConv.

119.1.3.4.54.1 TSOAPDomConv.SerializationOptions Method (PTypeInfo)

Pascal

```
function SerializationOptions(ATypeInfo: PTypeInfo): TSerializationOptions; overload;
```

C++

```
__fastcall TSerializationOptions SerializationOptions(PTypeInfo ATypeInfo);
```

Description

This is SerializationOptions, a member of class TSOAPDomConv.

119.1.3.4.54.2 TSOAPDomConv.SerializationOptions Method (TClass)

Pascal

```
function SerializationOptions(Cls: TClass): TSerializationOptions; overload;
```

C++

```
__fastcall TSerializationOptions SerializationOptions(TClass Cls);
```

Description

This is SerializationOptions, a member of class TSOAPDomConv.

119.1.3.4.54.3 TSOAPDomConv.SerializationOptions Method (TObject)

Pascal

```
function SerializationOptions(Obj: TObject): TSerializationOptions; overload;
```

C++

```
__fastcall TSerializationOptions SerializationOptions(TObject * Obj);
```

Description

This is SerializationOptions, a member of class TSOAPDomConv.

119.1.3.4.55 TSOAPDomConv.SetNodeType Method

Pascal

```
procedure SetNodeType(RootNode: IXMLNode; InstNode: IXMLNode; const ElemURI: InvString; const
TypeName: InvString);
```

C++

```
__fastcall SetNodeType(IXMLNode RootNode, IXMLNode InstNode, const InvString ElemURI, const
InvString TypeName);
```

Description

This is SetNodeType, a member of class TSOAPDomConv.

119.1.3.4.56 TSOAPDomConv.SetObjectPropFromText Method

Pascal

```
procedure SetObjectPropFromText(Instance: TObject; PropInfo: PPropInfo; const SoapData:
WideString);
```

C++

```
__fastcall SetObjectPropFromText(TObject * Instance, PPropInfo PropInfo, const BSTR SoapData);
```

Description

This is SetObjectPropFromText, a member of class TSOAPDomConv.

119.1.3.4.57 TSOAPDomConv.SetOptions Method

Pascal

```
procedure SetOptions(const Value: TSOAPConvertOptions);
```

C++

```
__fastcall SetOptions(const TSOAPConvertOptions Value);
```

Description

This is SetOptions, a member of class TSOAPDomConv.

119.1.3.4.58 TSOAPDomConv.UnhandledNode Method

Pascal

```
procedure UnhandledNode(const Name: string; NodeXML: WideString);
```

C++

```
__fastcall UnhandledNode(const AnsiString Name, BSTR NodeXML);
```

Description

This is UnhandledNode, a member of class TSOAPDomConv.

119.1.3.4.59 TSOAPDomConv.WriteHeader Method

Pascal

```
procedure WriteHeader(const Header: TObject; RootNode: IXMLNode; ParentNode: IXMLNode);
```

C++

```
__fastcall WriteHeader(const TObject * Header, IXMLNode RootNode, IXMLNode ParentNode);
```

Description

This is WriteHeader, a member of class TSOAPDomConv.

119.1.3.4.60 TSOAPDomConv.WriteNonRectDynArray Method

Pascal

```
procedure WriteNonRectDynArray(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString;
Info: PTypeInfo; const URI: InvString; const TypeName: InvString; P: Pointer; Dim: Integer);
```

C++

```
__fastcall WriteNonRectDynArray(IXMLNode RootNode, IXMLNode Node, const InvString Name,
PTypeInfo Info, const InvString URI, const InvString TypeName, void * P, int Dim);
```

Description

This is WriteNonRectDynArray, a member of class TSOAPDomConv.

119.1.3.4.61 TSOAPDomConv.WriteNonRectDynArrayElem Method

Pascal

```
function WriteNonRectDynArrayElem(RootNode: IXMLNode; Node: IXMLNode; Info: PTypeInfo; const
URI: InvString; const TypeName: InvString; P: Pointer; Dim: Integer): Integer;
```

C++

```
__fastcall int WriteNonRectDynArrayElem(IXMLNode RootNode, IXMLNode Node, PTypeInfo Info,
const InvString URI, const InvString TypeName, void * P, int Dim);
```

Description

This is WriteNonRectDynArrayElem, a member of class TSOAPDomConv.

119.1.3.4.62 TSOAPDomConv.WriteRectDynArray Method

Pascal

```
procedure WriteRectDynArray(RootNode: IXMLNode; Node: IXMLNode; Info: PTypeInfo; Dims:
Integer; P: Pointer; const TypeName: InvString);
```

C++

```
__fastcall WriteRectDynArray(IXMLNode RootNode, IXMLNode Node, PTypeInfo Info, int Dims, void
* P, const InvString TypeName);
```

Description

This is WriteRectDynArray, a member of class TSOAPDomConv.

119.1.3.4.63 TSOAPDomConv.WriteRectDynArrayElem Method

Pascal

```
procedure WriteRectDynArrayElem(RootNode: IXMLNode; Node: IXMLNode; Info: PTypeInfo; Size:
Integer; Dim: Integer; P: Pointer; const TypeName: InvString);
```

C++

```
__fastcall WriteRectDynArrayElem(IXMLNode RootNode, IXMLNode Node, PTypeInfo Info, int Size,
int Dim, void * P, const InvString TypeName);
```

Description

This is WriteRectDynArrayElem, a member of class TSOAPDomConv.

119.1.3.4.64 TSOAPDomConv.WriteVarArray Method

Pascal

```
procedure WriteVarArray(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString; V: Variant);
```

C++

```
__fastcall WriteVarArray(IXMLNode RootNode, IXMLNode Node, const InvString Name, Variant V);
```

Description

This is WriteVarArray, a member of class TSOAPDomConv.

119.1.3.4.65 TSOAPDomConv.WriteVarArrayAsB64 Method

Pascal

```
procedure WriteVarArrayAsB64(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString; V:
Variant);
```

C++

```
__fastcall WriteVarArrayAsB64(IXMLNode RootNode, IXMLNode Node, const InvString Name, Variant
V);
```

Description

This is WriteVarArrayAsB64, a member of class TSOAPDomConv.

119.1.3.4.66 TSOAPDomConv.WriteVariant Method

Pascal

```
procedure WriteVariant(RootNode: IXMLNode; Node: IXMLNode; const Name: InvString; V: Variant);
```

C++

```
__fastcall WriteVariant(IXMLNode RootNode, IXMLNode Node, const InvString Name, Variant V);
```

Description

This is WriteVariant, a member of class TSOAPDomConv.

119.1.3.5 TSOAPDomConv Properties

The properties of the TSOAPDomConv class are listed here.

Published Properties

	Name	Description
	Options (see page 18176)	Controls (see page 5711) how the TSOAPDomConv (see page 18150) object performs certain types of conversions.

119.1.3.5.1 TSOAPDomConv.Options Property

Controls (see page 5711) how the TSOAPDomConv (see page 18150) object performs certain types of conversions.

Pascal

```
property Options: TSOAPConvertOptions;
```

C++

```
__property TSOAPConvertOptions Options;
```

Description

Set Options to indicate how you want the TSOAPDomConv (see page 18150) object to handle various types of conversions. This set can include zero or more of the TSOAPConvertOption flags.

See Also

ConvertNativeDataToSoap (see page 18160)

ClassOptions (see page 15565)

ConvertSoapToNativeData (see page 18161)

119.1.3.6 TSOAPDomConv Events

The events of the TSOAPDomConv class are listed here.

Published Events

	Name	Description
	OnMemberDataNotReceived (see page 18177)	Occurs when converting a SOAP representation to a remotable object and an expected class member was not found.
	OnUnhandledNode (see page 18177)	Occurs when a SOAP representation includes an XML node that the converter does not know how to handle.

119.1.3.6.1 TSOAPDomConv.OnMemberDataNotReceived Event

Occurs when converting a SOAP representation to a remotable object and an expected class member was not found.

Pascal

```
property OnMemberDataNotReceived: TMemberDataNotReceivedEvent;
```

C++

```
__property TMemberDataNotReceivedEvent OnMemberDataNotReceived;
```

Description

Write an OnMemberDataNotReceived to detect cases where the converter does not receive the information it excepts When a method on an invokable interface requires the user to send or receive OnMemberDataNotReceived, these for deserializing a remotable type (a TRemotable descendant). This event is especially useful during debugging. Typically, after detecting a problem using OnMemberDataNotReceived, the problem is then handled by changing the code that declares or registers the remotable type.

See Also

TRemotable (see page 15556)

119.1.3.6.2 TSOAPDomConv.OnUnhandledNode Event

Occurs when a SOAP representation includes an XML node that the converter does not know how to handle.

Pascal

```
property OnUnhandledNode: TUnhandledNodeEvent;
```

C++

```
__property TUnhandledNodeEvent OnUnhandledNode;
```

Description

Write an OnUnhandledNode to detect and, possibly, fix cases where the converter can't correctly generate the native code corresponding to a node in a SOAP packet.

119.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetOrdPropEx (see page 18178)	This is function OPToSOAPDomConv.GetOrdPropEx.

119.2.1 OPToSOAPDomConv.GetOrdPropEx Function

Pascal

```
function GetOrdPropEx(Instance: TObject; PropInfo: PPropInfo): Longint;
```

C++

```
Longint GetOrdPropEx(TObject * Instance, PPropInfo PropInfo);
```

File

OPToSOAPDomConv

Description

This is function OPToSOAPDomConv.GetOrdPropEx.

119.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	ConvNodeState (see page 18179)	This is record OPToSOAPDomConv.ConvNodeState.

Records

	Name	Description
	TMultiRefNodeElem (see page 18179)	This is record OPToSOAPDomConv.TMultiRefNodeElem.
	TMultiRefNodeMapElem (see page 18180)	This is record OPToSOAPDomConv.TMultiRefNodeMapElem.
	TSOAPArrayElemDesc (see page 18180)	This is record OPToSOAPDomConv.TSOAPArrayElemDesc.

119.3.1 OPToSOAPDomConv.ConvNodeState Enumeration

Pascal

```
ConvNodeState = (
  nsClientSend,
  nsServerReceive,
  nsServerSend,
  nsClientReceive
);
```

C++

```
enum ConvNodeState {
  nsClientSend,
  nsServerReceive,
  nsServerSend,
  nsClientReceive
};
```

File

OPToSOAPDomConv

Description

This is record OPToSOAPDomConv.ConvNodeState.

119.3.2 OPToSOAPDomConv.TMultiRefNodeElem Record

Pascal

```
TMultiRefNodeElem = record
  Node: IXMLNode;
  MultiRefChildren: TXMLNodeArray;
end;
```

C++

```
struct TMultiRefNodeElem {
  IXMLNode Node;
  TXMLNodeArray MultiRefChildren;
};
```

File

OPToSOAPDomConv

Description

This is record OPToSOAPDomConv.TMultiRefNodeElem.

119.3.3 OPToSOAPDomConv.TMultiRefNodeMapElem Record

Pascal

```
TMultiRefNodeMapElem = record
  Instance: Pointer;
  ID: string;
end;
```

C++

```
struct TMultiRefNodeMapElem {
  void * Instance;
  AnsiString ID;
};
```

File

OPToSOAPDomConv

Description

This is record OPToSOAPDomConv.TMultiRefNodeMapElem.

119.3.4 OPToSOAPDomConv.TSOAPArrayElemDesc Record

Pascal

```
TSOAPArrayElemDesc = record
  MultiDim: Boolean;
  Dims: TIntegerDynArray;
end;
```

C++

```
struct TSOAPArrayElemDesc {
  Boolean MultiDim;
  TIntegerDynArray Dims;
};
```

File

OPToSOAPDomConv

Description

This is record OPToSOAPDomConv.TSOAPArrayElemDesc.

119.4 Types

The following table lists types in this documentation.

Types

Name	Description
TMemberDataNotReceivedEvent (see page 18182)	This is type OPToSOAPDomConv.TMemberDataNotReceivedEvent.
TMultiRefNodeMap (see page 18182)	This is type OPToSOAPDomConv.TMultiRefNodeMap.
TMultiRefNodes (see page 18183)	This is type OPToSOAPDomConv.TMultiRefNodes.
TSOAPArrayDesc (see page 18183)	This is type OPToSOAPDomConv.TSOAPArrayDesc.
TUnhandledNodeEvent (see page 18183)	This is type OPToSOAPDomConv.TUnhandledNodeEvent.
TXMLNodeArray (see page 18184)	This is type OPToSOAPDomConv.TXMLNodeArray.

119.4.1 OPToSOAPDomConv.TMemberDataNotReceivedEvent Type

Pascal

```
TMemberDataNotReceivedEvent = procedure (const ClassName: string; const Member: string) of
object;
```

C++

```
(const ClassName: string; const Member: string) ( TMemberDataNotReceivedEvent)();
```

File

OPToSOAPDomConv

Description

This is type OPToSOAPDomConv.TMemberDataNotReceivedEvent.

119.4.2 OPToSOAPDomConv.TMultiRefNodeMap Type

Pascal

```
TMultiRefNodeMap = array of TMultiRefNodeMapElem;
```

C++

```
array of TMultiRefNodeMapElem TMultiRefNodeMap;
```

File

OPToSOAPDomConv

Description

This is type OPToSOAPDomConv.TMultiRefNodeMap.

119.4.3 OPToSOAPDomConv.TMultiRefNodes Type

Pascal

```
TMultiRefNodes = array of TMultiRefNodeElem;
```

C++

```
array of TMultiRefNodeElem TMultiRefNodes;
```

File

OPToSOAPDomConv

Description

This is type OPToSOAPDomConv.TMultiRefNodes.

119.4.4 OPToSOAPDomConv.TSOAPArrayDesc Type

Pascal

```
TSOAPArrayDesc = array of TSOAPArrayElemDesc;
```

C++

```
array of TSOAPArrayElemDesc TSOAPArrayDesc;
```

File

OPToSOAPDomConv

Description

This is type OPToSOAPDomConv.TSOAPArrayDesc.

119.4.5 OPToSOAPDomConv.TUnhandledNodeEvent Type

Pascal

```
TUnhandledNodeEvent = procedure (const Name: string; NodeXML: WideString) of object;
```

C++

```
(const Name: string; NodeXML: WideString) ( TUnhandledNodeEvent)();
```

File

OPToSOAPDomConv

Description

This is type OPToSOAPDomConv.TUnhandledNodeEvent.

119.4.6 OPToSOAPDomConv.TXMLNodeArray Type

Pascal

```
TXMLNodeArray = array of IXMLNode;
```

C++

```
array of IXMLNode TXMLNodeArray;
```

File

OPToSOAPDomConv

Description

This is type OPToSOAPDomConv.TXMLNodeArray.

119.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
DefArrayElemName (see page 18185)	This is variable OPToSOAPDomConv.DefArrayElemName.

119.5.1 OPToSOAPDomConv.DefArrayElemName Variable

Pascal

```
DefArrayElemName: string = 'item';
```

C++

```
AnsiString DefArrayElemName = 'item';
```

File

OPToSOAPDomConv

Description

This is variable OPToSOAPDomConv.DefArrayElemName.

119.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
SVarArrayType (see page 18186)	This is constant OPToSOAPDomConv.SVarArrayType.

119.6.1 OPToSOAPDomConv.SVarArrayType Constant

Pascal

```
SVarArrayType = 'VarArrayType';
```

C++

```
SVarArrayType = 'VarArrayType';
```

File

OPToSOAPDomConv

Description

This is constant OPToSOAPDomConv.SVarArrayType.

120 OrbPas Namespace

This is namespace OrbPas.

Topics

Name	Description
Interfaces (see page 18189)	The following table lists interfaces in this documentation.
Functions (see page 18217)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 18222)	The following table lists structs, records, enums in this documentation.
Types (see page 18224)	The following table lists types in this documentation.

Functions

	Name	Description
	AnyToVariant (see page 18217)	This is function OrbPas.AnyToVariant.
	BindStub (see page 18217)	This is function OrbPas.BindStub.
	CorbaAnyType (see page 18218)	This is function OrbPas.CorbaAnyType.
	CorbaDuplicateAny (see page 18218)	This is function OrbPas.CorbaDuplicateAny.
	CorbaReleaseAny (see page 18218)	This is function OrbPas.CorbaReleaseAny.
	CorbaStringFree (see page 18219)	This is function OrbPas.CorbaStringFree.
	CorbaWStringFree (see page 18219)	This is function OrbPas.CorbaWStringFree.
	CreateSkeleton (see page 18219)	This is function OrbPas.CreateSkeleton.
	CreateUserException (see page 18220)	This is function OrbPas.CreateUserException.
	InitORB (see page 18220)	This is function OrbPas.InitORB.
	RegisterUserException (see page 18220)	This is function OrbPas.RegisterUserException.
	SequenceToVariant (see page 18221)	This is function OrbPas.SequenceToVariant.
	UnRegisterUserException (see page 18221)	This is function OrbPas.UnRegisterUserException.
	VariantToAny (see page 18221)	This is function OrbPas.VariantToAny.

Interfaces

	Name	Description
	IBOA (see page 18189)	This is class OrbPas.IBOA.
	ICorbaObj (see page 18191)	This is class OrbPas.ICorbaObj.
	IMarshalInBuffer (see page 18192)	This is class OrbPas.IMarshalInBuffer.
	IMarshalOutBuffer (see page 18196)	This is class OrbPas.IMarshalOutBuffer.
	IORB (see page 18201)	This is class OrbPas.IORB.
	ISkeleton (see page 18205)	This is class OrbPas.ISkeleton.
	ISkeletonObject (see page 18206)	This is class OrbPas.ISkeletonObject.

	IStub (see page 18208)	This is class OrbPas.IStub.
	IStubObject (see page 18211)	This is class OrbPas.IStubObject.
	ITypeCode (see page 18212)	This is class OrbPas.ITypeCode.

Structs, Records, Enums

	Name	Description
	TCKind (see page 18222)	This is record OrbPas.TCKind.
	TStructMember (see page 18223)	This is record OrbPas.TStructMember.

Types

Name	Description
CorbaBoolean (see page 18224)	This is type OrbPas.CorbaBoolean.
CorbaULong (see page 18224)	This is type OrbPas.CorbaULong.
PCorbaAny (see page 18225)	This is type OrbPas.PCorbaAny.
PCorbaInterfaceDef (see page 18225)	This is type OrbPas.PCorbaInterfaceDef.
PCorbaObject (see page 18225)	This is type OrbPas.PCorbaObject.
PCorbaOrb (see page 18226)	This is type OrbPas.PCorbaOrb.
PExceptionDescription (see page 18226)	This is type OrbPas.PExceptionDescription.
PMarshalInbuffer (see page 18226)	This is type OrbPas.PMarshalInbuffer.
PMarshalOutbuffer (see page 18227)	This is type OrbPas.PMarshalOutbuffer.
PUserExceptionProxy (see page 18227)	This is type OrbPas.PUserExceptionProxy.
TArgv (see page 18227)	This is type OrbPas.TArgv.
TCopyUserExceptionProc (see page 18228)	This is type OrbPas.TCopyUserExceptionProc.
TStructMembers (see page 18228)	This is type OrbPas.TStructMembers.
TThrowUserExceptionProc (see page 18228)	This is type OrbPas.TThrowUserExceptionProc.
TUserExceptionFactoryProc (see page 18229)	This is type OrbPas.TUserExceptionFactoryProc.

120.1 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IBOA (see page 18189)	This is class OrbPas.IBOA.
	ICorbaObj (see page 18191)	This is class OrbPas.ICorbaObj.
	IMarshalInBuffer (see page 18192)	This is class OrbPas.IMarshalInBuffer.
	IMarshalOutBuffer (see page 18196)	This is class OrbPas.IMarshalOutBuffer.
	IORB (see page 18201)	This is class OrbPas.IORB.
	ISkeleton (see page 18205)	This is class OrbPas.ISkeleton.
	ISkeletonObject (see page 18206)	This is class OrbPas.ISkeletonObject.
	IStub (see page 18208)	This is class OrbPas.IStub.
	IStubObject (see page 18211)	This is class OrbPas.IStubObject.
	ITypeCode (see page 18212)	This is class OrbPas.ITypeCode.

120.1.1 IBOA Interface

Class Hierarchy

OrbPas.IBOA

Pascal

```
IBOA = interface;
```

C++

```
__interface IBOA;
```

File

OrbPas

Description

This is class OrbPas.IBOA.

120.1.1.1 IBOA Members

The following tables list the members exposed by IBOA.

Public Methods

	Name	Description
	Deactivate (see page 18190)	This is Deactivate, a member of class IBOA.
	GetPrincipal (see page 18190)	This is GetPrincipal, a member of class IBOA.
	GetPrincipalLength (see page 18190)	This is GetPrincipalLength, a member of class IBOA.
	ImplIsReady (see page 18190)	This is ImplIsReady, a member of class IBOA.
	ObjIsReady (see page 18191)	This is ObjIsReady, a member of class IBOA.

120.1.1.2 IBOA Methods

The methods of the IBOA class are listed here.

Public Methods

	Name	Description
	Deactivate (see page 18190)	This is Deactivate, a member of class IBOA.
	GetPrincipal (see page 18190)	This is GetPrincipal, a member of class IBOA.
	GetPrincipalLength (see page 18190)	This is GetPrincipalLength, a member of class IBOA.
	ImplIsReady (see page 18190)	This is ImplIsReady, a member of class IBOA.
	ObjIsReady (see page 18191)	This is ObjIsReady, a member of class IBOA.

120.1.1.2.1 IBOA.Deactivate Method

Pascal

```
procedure Deactivate(const Skeleton: ISkeleton); stdcall;
```

C++

```
__fastcall __stdcall Deactivate(const ISkeleton Skeleton);
```

Description

This is Deactivate, a member of class IBOA.

120.1.1.2.2 IBOA.GetPrincipal Method

Pascal

```
function GetPrincipal(Skeleton: ISkeleton; Bytes: Pointer): Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetPrincipal(ISkeleton Skeleton, void * Bytes);
```

Description

This is GetPrincipal, a member of class IBOA.

120.1.1.2.3 IBOA.GetPrincipalLength Method

Pascal

```
function GetPrincipalLength(Skeleton: ISkeleton): Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetPrincipalLength(ISkeleton Skeleton);
```

Description

This is GetPrincipalLength, a member of class IBOA.

120.1.1.2.4 IBOA.ImplIsReady Method

Pascal

```
procedure ImplIsReady; stdcall;
```

C++

```
__fastcall __stdcall ImplIsReady();
```

Description

This is ImplIsReady, a member of class IBOA.

120.1.1.2.5 IBOA.ObjIsReady Method

Pascal

```
procedure ObjIsReady(const Skeleton: ISkeleton); stdcall;
```

C++

```
__fastcall __stdcall ObjIsReady(const ISkeleton Skeleton);
```

Description

This is ObjIsReady, a member of class IBOA.

120.1.2 ICorbaObj Interface

Class Hierarchy

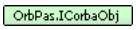

Pascal

```
ICorbaObj = interface;
```

C++

```
__interface ICorbaObj;
```

File

OrbPas

Description

This is class OrbPas.ICorbaObj.

120.1.2.1 ICorbaObj Members

The following tables list the members exposed by ICorbaObj.

Public Methods

	Name	Description
	CorbaObject (see page 18192)	This is CorbaObject, a member of class ICorbaObj.
	IsLocal (see page 18192)	This is IsLocal, a member of class ICorbaObj.

120.1.2.2 ICorbaObj Methods

The methods of the ICorbaObj class are listed here.

Public Methods

	Name	Description
	CorbaObject (see page 18192)	This is CorbaObject, a member of class ICorbaObj.
	IsLocal (see page 18192)	This is IsLocal, a member of class ICorbaObj.

120.1.2.2.1 ICorbaObj.CorbaObject Method

Pascal

```
function CorbaObject: PCorbaObject; stdcall;
```

C++

```
__fastcall __stdcall PCorbaObject CorbaObject();
```

Description

This is CorbaObject, a member of class ICorbaObj.

120.1.2.2.2 ICorbaObj.IsLocal Method

Pascal

```
function IsLocal: CorbaBoolean; stdcall;
```

C++

```
__fastcall __stdcall CorbaBoolean IsLocal();
```

Description

This is IsLocal, a member of class ICorbaObj.

120.1.3 IMarshalInBuffer Interface

Class Hierarchy

OrbPas.IMarshalInBuffer

Pascal

```
IMarshalInBuffer = interface;
```

C++

```
__interface IMarshalInBuffer;
```

File

OrbPas

Description

This is class OrbPas.IMarshalInBuffer.

120.1.3.1 IMarshalInBuffer Members

The following tables list the members exposed by IMarshalInBuffer.

Public Methods

	Name	Description
	Buffer (see page 18193)	This is Buffer, a member of class IMarshalInBuffer.
	GetAny (see page 18194)	This is GetAny, a member of class IMarshalInBuffer.
	GetChar (see page 18194)	This is GetChar, a member of class IMarshalInBuffer.
	GetDouble (see page 18194)	This is GetDouble, a member of class IMarshalInBuffer.
	GetFloat (see page 18194)	This is GetFloat, a member of class IMarshalInBuffer.
	GetLong (see page 18194)	This is GetLong, a member of class IMarshalInBuffer.
	GetObject (see page 18195)	This is GetObject, a member of class IMarshalInBuffer.
	GetShort (see page 18195)	This is GetShort, a member of class IMarshalInBuffer.
	GetText (see page 18195)	This is GetText, a member of class IMarshalInBuffer.
	GetUnsignedChar (see page 18195)	This is GetUnsignedChar, a member of class IMarshalInBuffer.
	GetUnsignedLong (see page 18196)	This is GetUnsignedLong, a member of class IMarshalInBuffer.
	GetUnsignedShort (see page 18196)	This is GetUnsignedShort, a member of class IMarshalInBuffer.
	GetWidechar (see page 18196)	This is GetWidechar, a member of class IMarshalInBuffer.
	GetWideText (see page 18196)	This is GetWideText, a member of class IMarshalInBuffer.

120.1.3.2 IMarshalInBuffer Methods

The methods of the IMarshalInBuffer class are listed here.

Public Methods

	Name	Description
	Buffer (see page 18193)	This is Buffer, a member of class IMarshalInBuffer.
	GetAny (see page 18194)	This is GetAny, a member of class IMarshalInBuffer.
	GetChar (see page 18194)	This is GetChar, a member of class IMarshalInBuffer.
	GetDouble (see page 18194)	This is GetDouble, a member of class IMarshalInBuffer.
	GetFloat (see page 18194)	This is GetFloat, a member of class IMarshalInBuffer.
	GetLong (see page 18194)	This is GetLong, a member of class IMarshalInBuffer.
	GetObject (see page 18195)	This is GetObject, a member of class IMarshalInBuffer.
	GetShort (see page 18195)	This is GetShort, a member of class IMarshalInBuffer.
	GetText (see page 18195)	This is GetText, a member of class IMarshalInBuffer.
	GetUnsignedChar (see page 18195)	This is GetUnsignedChar, a member of class IMarshalInBuffer.
	GetUnsignedLong (see page 18196)	This is GetUnsignedLong, a member of class IMarshalInBuffer.
	GetUnsignedShort (see page 18196)	This is GetUnsignedShort, a member of class IMarshalInBuffer.
	GetWidechar (see page 18196)	This is GetWidechar, a member of class IMarshalInBuffer.
	GetWideText (see page 18196)	This is GetWideText, a member of class IMarshalInBuffer.

120.1.3.2.1 IMarshalInBuffer.Buffer Method

Pascal

```
function Buffer: PMarshalInbuffer; stdcall;
```

C++

```
__fastcall __stdcall PMarshalInbuffer Buffer();
```

Description

This is Buffer, a member of class IMarshalInBuffer.

120.1.3.2.2 IMarshalInBuffer.GetAny Method

Pascal

```
function GetAny: PCorbaAny; stdcall;
```

C++

```
__fastcall __stdcall PCorbaAny GetAny();
```

Description

This is GetAny, a member of class IMarshalInBuffer.

120.1.3.2.3 IMarshalInBuffer.GetChar Method

Pascal

```
function GetChar: Shortint; stdcall;
```

C++

```
__fastcall __stdcall Shortint GetChar();
```

Description

This is GetChar, a member of class IMarshalInBuffer.

120.1.3.2.4 IMarshalInBuffer.GetDouble Method

Pascal

```
function GetDouble: Double; stdcall;
```

C++

```
__fastcall __stdcall Double GetDouble();
```

Description

This is GetDouble, a member of class IMarshalInBuffer.

120.1.3.2.5 IMarshalInBuffer.GetFloat Method

Pascal

```
function GetFloat: Single; stdcall;
```

C++

```
__fastcall __stdcall Single GetFloat();
```

Description

This is GetFloat, a member of class IMarshalInBuffer.

120.1.3.2.6 IMarshalInBuffer.GetLong Method

Pascal

```
function GetLong: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetLong();
```

Description

This is GetLong, a member of class IMarshalInBuffer.

120.1.3.2.7 IMarshalInBuffer.GetObject Method

Pascal

```
procedure GetObject(out Obj: ICorbaObj; ref_cnt: CorbaBoolean = True); stdcall;
```

C++

```
__fastcall __stdcall GetObject(ICorbaObj Obj, CorbaBoolean ref_cnt = True);
```

Description

This is GetObject, a member of class IMarshalInBuffer.

120.1.3.2.8 IMarshalInBuffer.GetShort Method

Pascal

```
function GetShort: Smallint; stdcall;
```

C++

```
__fastcall __stdcall Smallint GetShort();
```

Description

This is GetShort, a member of class IMarshalInBuffer.

120.1.3.2.9 IMarshalInBuffer.GetText Method

Pascal

```
function GetText: PChar; stdcall;
```

C++

```
__fastcall __stdcall const char * GetText();
```

Description

This is GetText, a member of class IMarshalInBuffer.

120.1.3.2.10 IMarshalInBuffer.GetUnsignedChar Method

Pascal

```
function GetUnsignedChar: Byte; stdcall;
```

C++

```
__fastcall __stdcall Byte GetUnsignedChar();
```

Description

This is GetUnsignedChar, a member of class IMarshalInBuffer.

120.1.3.2.11 IMarshalInBuffer.GetUnsignedLong Method

Pascal

```
function GetUnsignedLong: UINT; stdcall;
```

C++

```
__fastcall __stdcall UINT GetUnsignedLong();
```

Description

This is GetUnsignedLong, a member of class IMarshalInBuffer.

120.1.3.2.12 IMarshalInBuffer.GetUnsignedShort Method

Pascal

```
function GetUnsignedShort: Word; stdcall;
```

C++

```
__fastcall __stdcall Word GetUnsignedShort();
```

Description

This is GetUnsignedShort, a member of class IMarshalInBuffer.

120.1.3.2.13 IMarshalInBuffer.GetWidechar Method

Pascal

```
function GetWidechar: Word; stdcall;
```

C++

```
__fastcall __stdcall Word GetWidechar();
```

Description

This is GetWidechar, a member of class IMarshalInBuffer.

120.1.3.2.14 IMarshalInBuffer.GetWideText Method

Pascal

```
function GetWideText: PWideChar; stdcall;
```

C++

```
__fastcall __stdcall PWideChar GetWideText();
```

Description

This is GetWideText, a member of class IMarshalInBuffer.

120.1.4 IMarshalOutBuffer Interface

Class Hierarchy

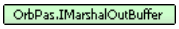

Pascal

```
IMarshalOutBuffer = interface;
```

C++

```
__interface IMarshalOutBuffer;
```

File

OrbPas

Description

This is class OrbPas.IMarshalOutBuffer.

120.1.4.1 IMarshalOutBuffer Members

The following tables list the members exposed by IMarshalOutBuffer.

Public Methods

	Name	Description
	Buffer (see page 18198)	This is Buffer, a member of class IMarshalOutBuffer.
	PutAny (see page 18198)	This is PutAny, a member of class IMarshalOutBuffer.
	PutChar (see page 18198)	This is PutChar, a member of class IMarshalOutBuffer.
	PutDouble (see page 18198)	This is PutDouble, a member of class IMarshalOutBuffer.
	PutFloat (see page 18198)	This is PutFloat, a member of class IMarshalOutBuffer.
	PutLong (see page 18199)	This is PutLong, a member of class IMarshalOutBuffer.
	PutObject (see page 18199)	This is PutObject, a member of class IMarshalOutBuffer.
	PutShort (see page 18199)	This is PutShort, a member of class IMarshalOutBuffer.
	PutText (see page 18199)	This is PutText, a member of class IMarshalOutBuffer.
	PutUnsignedChar (see page 18200)	This is PutUnsignedChar, a member of class IMarshalOutBuffer.
	PutUnsignedLong (see page 18200)	This is PutUnsignedLong, a member of class IMarshalOutBuffer.
	PutUnsignedShort (see page 18200)	This is PutUnsignedShort, a member of class IMarshalOutBuffer.
	PutWidechar (see page 18200)	This is PutWidechar, a member of class IMarshalOutBuffer.
	PutWideText (see page 18200)	This is PutWideText, a member of class IMarshalOutBuffer.

120.1.4.2 IMarshalOutBuffer Methods

The methods of the IMarshalOutBuffer class are listed here.

Public Methods

	Name	Description
	Buffer (see page 18198)	This is Buffer, a member of class IMarshalOutBuffer.
	PutAny (see page 18198)	This is PutAny, a member of class IMarshalOutBuffer.
	PutChar (see page 18198)	This is PutChar, a member of class IMarshalOutBuffer.
	PutDouble (see page 18198)	This is PutDouble, a member of class IMarshalOutBuffer.
	PutFloat (see page 18198)	This is PutFloat, a member of class IMarshalOutBuffer.
	PutLong (see page 18199)	This is PutLong, a member of class IMarshalOutBuffer.
	PutObject (see page 18199)	This is PutObject, a member of class IMarshalOutBuffer.
	PutShort (see page 18199)	This is PutShort, a member of class IMarshalOutBuffer.
	PutText (see page 18199)	This is PutText, a member of class IMarshalOutBuffer.
	PutUnsignedChar (see page 18200)	This is PutUnsignedChar, a member of class IMarshalOutBuffer.
	PutUnsignedLong (see page 18200)	This is PutUnsignedLong, a member of class IMarshalOutBuffer.
	PutUnsignedShort (see page 18200)	This is PutUnsignedShort, a member of class IMarshalOutBuffer.
	PutWidechar (see page 18200)	This is PutWidechar, a member of class IMarshalOutBuffer.

	PutWideText (see page 18200)	This is PutWideText, a member of class IMarshalOutBuffer.

120.1.4.2.1 IMarshalOutBuffer.Buffer Method

Pascal

```
function Buffer: PMarshalOutbuffer; stdcall;
```

C++

```
__fastcall __stdcall PMarshalOutbuffer Buffer();
```

Description

This is Buffer, a member of class IMarshalOutBuffer.

120.1.4.2.2 IMarshalOutBuffer.PutAny Method

Pascal

```
procedure PutAny(Value: PCorbaAny); stdcall;
```

C++

```
__fastcall __stdcall PutAny(PCorbaAny Value);
```

Description

This is PutAny, a member of class IMarshalOutBuffer.

120.1.4.2.3 IMarshalOutBuffer.PutChar Method

Pascal

```
procedure PutChar(Value: Shortint); stdcall;
```

C++

```
__fastcall __stdcall PutChar(Shortint Value);
```

Description

This is PutChar, a member of class IMarshalOutBuffer.

120.1.4.2.4 IMarshalOutBuffer.PutDouble Method

Pascal

```
procedure PutDouble(Value: Double); stdcall;
```

C++

```
__fastcall __stdcall PutDouble(Double Value);
```

Description

This is PutDouble, a member of class IMarshalOutBuffer.

120.1.4.2.5 IMarshalOutBuffer.PutFloat Method

Pascal

```
procedure PutFloat(Value: Single); stdcall;
```

C++

```
__fastcall __stdcall PutFloat(Single Value);
```

Description

This is PutFloat, a member of class IMarshalOutBuffer.

120.1.4.2.6 IMarshalOutBuffer.PutLong Method

Pascal

```
procedure PutLong(Value: Integer); stdcall;
```

C++

```
__fastcall __stdcall PutLong(int Value);
```

Description

This is PutLong, a member of class IMarshalOutBuffer.

120.1.4.2.7 IMarshalOutBuffer.PutObject Method

Pascal

```
procedure PutObject(const Value: ICorbaObj); stdcall;
```

C++

```
__fastcall __stdcall PutObject(const ICorbaObj Value);
```

Description

This is PutObject, a member of class IMarshalOutBuffer.

120.1.4.2.8 IMarshalOutBuffer.PutShort Method

Pascal

```
procedure PutShort(Value: Smallint); stdcall;
```

C++

```
__fastcall __stdcall PutShort(Smallint Value);
```

Description

This is PutShort, a member of class IMarshalOutBuffer.

120.1.4.2.9 IMarshalOutBuffer.PutText Method

Pascal

```
procedure PutText(Value: PChar); stdcall;
```

C++

```
__fastcall __stdcall PutText(const char * Value);
```

Description

This is PutText, a member of class IMarshalOutBuffer.

120.1.4.2.10 IMarshalOutBuffer.PutUnsignedChar Method

Pascal

```
procedure PutUnsignedChar(Value: Byte); stdcall;
```

C++

```
__fastcall __stdcall PutUnsignedChar(Byte Value);
```

Description

This is PutUnsignedChar, a member of class IMarshalOutBuffer.

120.1.4.2.11 IMarshalOutBuffer.PutUnsignedLong Method

Pascal

```
procedure PutUnsignedLong(Value: UINT); stdcall;
```

C++

```
__fastcall __stdcall PutUnsignedLong(UINT Value);
```

Description

This is PutUnsignedLong, a member of class IMarshalOutBuffer.

120.1.4.2.12 IMarshalOutBuffer.PutUnsignedShort Method

Pascal

```
procedure PutUnsignedShort(Value: Word); stdcall;
```

C++

```
__fastcall __stdcall PutUnsignedShort(Word Value);
```

Description

This is PutUnsignedShort, a member of class IMarshalOutBuffer.

120.1.4.2.13 IMarshalOutBuffer.PutWidechar Method

Pascal

```
procedure PutWidechar(Value: Word); stdcall;
```

C++

```
__fastcall __stdcall PutWidechar(Word Value);
```

Description

This is PutWidechar, a member of class IMarshalOutBuffer.

120.1.4.2.14 IMarshalOutBuffer.PutWideText Method

Pascal

```
procedure PutWideText(Value: PWideChar); stdcall;
```

C++

```
__fastcall __stdcall PutWideText(PWideChar Value);
```

Description

This is PutWideText, a member of class IMarshalOutBuffer.

120.1.5 IORB Interface

Class Hierarchy

OrbPas.IORB

Pascal

```
IORB = interface;
```

C++

```
__interface IORB;
```

File

OrbPas

Description

This is class OrbPas.IORB.

120.1.5.1 IORB Members

The following tables list the members exposed by IORB.

Public Methods

	Name	Description
	BOAInit (see page 18202)	This is BOAInit, a member of class IORB.
	CreateAliasTC (see page 18202)	This is CreateAliasTC, a member of class IORB.
	CreateArrayTC (see page 18202)	This is CreateArrayTC, a member of class IORB.
	CreateObjRefTC (see page 18203)	This is CreateObjRefTC, a member of class IORB.
	CreateSequenceTC (see page 18203)	This is CreateSequenceTC, a member of class IORB.
	CreateStructTC (see page 18203)	This is CreateStructTC, a member of class IORB.
	CreateTC (see page 18203)	This is CreateTC, a member of class IORB.
	DispatchStruct (see page 18203)	This is DispatchStruct, a member of class IORB.
	FindRepositoryTC (see page 18204)	This is FindRepositoryTC, a member of class IORB.
	MakeAny (see page 18204)	This is MakeAny, a member of class IORB.
	ObjectToString (see page 18204)	This is ObjectToString, a member of class IORB.
	Orb (see page 18204)	This is Orb, a member of class IORB.
	Shutdown (see page 18205)	This is Shutdown, a member of class IORB.
	StringToObject (see page 18205)	This is StringToObject, a member of class IORB.

120.1.5.2 IORB Methods

The methods of the IORB class are listed here.

Public Methods

	Name	Description
	BOAInit (see page 18202)	This is BOAInit, a member of class IORB.

	CreateAliasTC (see page 18202)	This is CreateAliasTC, a member of class IORB.
	CreateArrayTC (see page 18202)	This is CreateArrayTC, a member of class IORB.
	CreateObjRefTC (see page 18203)	This is CreateObjRefTC, a member of class IORB.
	CreateSequenceTC (see page 18203)	This is CreateSequenceTC, a member of class IORB.
	CreateStructTC (see page 18203)	This is CreateStructTC, a member of class IORB.
	CreateTC (see page 18203)	This is CreateTC, a member of class IORB.
	DispatchStruct (see page 18203)	This is DispatchStruct, a member of class IORB.
	FindRepositoryTC (see page 18204)	This is FindRepositoryTC, a member of class IORB.
	MakeAny (see page 18204)	This is MakeAny, a member of class IORB.
	ObjectToString (see page 18204)	This is ObjectToString, a member of class IORB.
	Orb (see page 18204)	This is Orb, a member of class IORB.
	Shutdown (see page 18205)	This is Shutdown, a member of class IORB.
	StringToObject (see page 18205)	This is StringToObject, a member of class IORB.

120.1.5.2.1 IORB.BOAInit Method

Pascal

```
procedure BOAInit(Argc: Integer; const Argv: TArgv; out BOA: IBOA); stdcall;
```

C++

```
__fastcall __stdcall BOAInit(int Argc, const TArgv Argv, IBOA BOA);
```

Description

This is BOAInit, a member of class IORB.

120.1.5.2.2 IORB.CreateAliasTC Method

Pascal

```
procedure CreateAliasTC(RepID: PChar; TypeName: PChar; const TC: ITypeCode; out Result:
ITypeCode); stdcall;
```

C++

```
__fastcall __stdcall CreateAliasTC(const char * RepID, const char * TypeName, const ITypeCode
TC, ITypeCode Result);
```

Description

This is CreateAliasTC, a member of class IORB.

120.1.5.2.3 IORB.CreateArrayTC Method

Pascal

```
procedure CreateArrayTC(Bound: CorbaULong; const TC: ITypeCode; out Result: ITypeCode);
stdcall;
```

C++

```
__fastcall __stdcall CreateArrayTC(CorbaULong Bound, const ITypeCode TC, ITypeCode Result);
```

Description

This is CreateArrayTC, a member of class IORB.

120.1.5.2.4 IORB.CreateObjRefTC Method

Pascal

```
procedure CreateObjRefTC(RepId: PChar; Name: PChar; out Result: ITypeCode); stdcall;
```

C++

```
__fastcall __stdcall CreateObjRefTC(const char * RepId, const char * Name, ITypeCode Result);
```

Description

This is CreateObjRefTC, a member of class IORB.

120.1.5.2.5 IORB.CreateSequenceTC Method

Pascal

```
procedure CreateSequenceTC(Bound: CorbaULong; const TC: ITypeCode; out Result: ITypeCode);
stdcall;
```

C++

```
__fastcall __stdcall CreateSequenceTC(CorbaULong Bound, const ITypeCode TC, ITypeCode Result);
```

Description

This is CreateSequenceTC, a member of class IORB.

120.1.5.2.6 IORB.CreateStructTC Method

Pascal

```
procedure CreateStructTC(Kind: TCKind; RepId: PChar; Name: PChar; Members: TStructMembers;
nMember: Integer; out Result: ITypeCode); stdcall;
```

C++

```
__fastcall __stdcall CreateStructTC(TCKind Kind, const char * RepId, const char * Name,
TStructMembers Members, int nMember, ITypeCode Result);
```

Description

This is CreateStructTC, a member of class IORB.

120.1.5.2.7 IORB.CreateTC Method

Pascal

```
procedure CreateTC(Kind: TCKind; out TC: ITypeCode); stdcall;
```

C++

```
__fastcall __stdcall CreateTC(TCKind Kind, ITypeCode TC);
```

Description

This is CreateTC, a member of class IORB.

120.1.5.2.8 IORB.DispatchStruct Method

Pascal

```
function DispatchStruct(Any: PCorbaAny; CallDesc: PCallDesc; Params: Pointer; out Result:
```

```
Variant): Integer; stdcall;
```

C++

```
__fastcall __stdcall int DispatchStruct(PCorbaAny Any, PCallDesc CallDesc, void * Params,
Variant Result);
```

Description

This is DispatchStruct, a member of class IORB.

120.1.5.2.9 IORB.FindRepositoryTC Method

Pascal

```
procedure FindRepositoryTC(RepID: PChar; out TC: ITypeCode); stdcall;
```

C++

```
__fastcall __stdcall FindRepositoryTC(const char * RepID, ITypeCode TC);
```

Description

This is FindRepositoryTC, a member of class IORB.

120.1.5.2.10 IORB.MakeAny Method

Pascal

```
function MakeAny(const TC: ITypeCode; VS: array of Variant): PCorbaAny; stdcall;
```

C++

```
__fastcall __stdcall PCorbaAny MakeAny(const ITypeCode TC, array of Variant VS);
```

Description

This is MakeAny, a member of class IORB.

120.1.5.2.11 IORB.ObjectToString Method

Pascal

```
function ObjectToString(const Obj: IStub): PChar; stdcall;
```

C++

```
__fastcall __stdcall const char * ObjectToString(const IStub Obj);
```

Description

This is ObjectToString, a member of class IORB.

120.1.5.2.12 IORB.Orb Method

Pascal

```
function Orb: PCorbaOrb;
```

C++

```
__fastcall PCorbaOrb Orb();
```

Description

This is Orb, a member of class IORB.

120.1.5.2.13 IORB.Shutdown Method

Pascal

```
procedure Shutdown; stdcall;
```

C++

```
__fastcall __stdcall Shutdown();
```

Description

This is Shutdown, a member of class IORB.

120.1.5.2.14 IORB.StringToObject Method

Pascal

```
procedure StringToObject(Str: PChar; out Obj: IStub); stdcall;
```

C++

```
__fastcall __stdcall StringToObject(const char * Str, IStub Obj);
```

Description

This is StringToObject, a member of class IORB.

120.1.6 ISkeleton Interface

Class Hierarchy

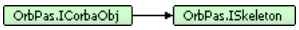

Pascal

```
ISkeleton = interface(ICorbaObj);
```

C++

```
__interface ISkeleton : public ICorbaObj;
```

File

OrbPas

Description

This is class OrbPas.ISkeleton.

120.1.6.1 ISkeleton Members

The following tables list the members exposed by ISkeleton.

Public Methods

	Name	Description
	CorbaObject (see page 18192)	This is CorbaObject, a member of class ICorbaObj.
	GetImplementation (see page 18206)	This is GetImplementation, a member of class ISkeleton.
	GetReplyBuffer (see page 18206)	This is GetReplyBuffer, a member of class ISkeleton.

	IsLocal (see page 18192)	This is IsLocal, a member of class ICorbaObj.

120.1.6.2 ISkeleton Methods

The methods of the ISkeleton class are listed here.

Public Methods

	Name	Description
	GetImplementation (see page 18206)	This is GetImplementation, a member of class ISkeleton.
	GetReplyBuffer (see page 18206)	This is GetReplyBuffer, a member of class ISkeleton.

120.1.6.2.1 ISkeleton.GetImplementation Method

Pascal

```
procedure GetImplementation(out Impl: IUnknown); stdcall;
```

C++

```
__fastcall __stdcall GetImplementation(IUnknown Impl);
```

Description

This is GetImplementation, a member of class ISkeleton.

120.1.6.2.2 ISkeleton.GetReplyBuffer Method

Pascal

```
procedure GetReplyBuffer(cookie: Pointer; out Outbuf: IMarshalOutBuffer); stdcall;
```

C++

```
__fastcall __stdcall GetReplyBuffer(void * cookie, IMarshalOutBuffer Outbuf);
```

Description

This is GetReplyBuffer, a member of class ISkeleton.

120.1.7 ISkeletonObject Interface

Class Hierarchy

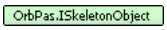

Pascal

```
ISkeletonObject = interface;
```

C++

```
__interface ISkeletonObject;
```

File

OrbPas

Description

This is class OrbPas.ISkeletonObject.

120.1.7.1 ISkeletonObject Members

The following tables list the members exposed by ISkeletonObject.

Public Methods

	Name	Description
	Execute (see page 18207)	This is Execute, a member of class ISkeletonObject.
	GetImplementation (see page 18207)	This is GetImplementation, a member of class ISkeletonObject.
	GetSkeleton (see page 18207)	This is GetSkeleton, a member of class ISkeletonObject.

120.1.7.2 ISkeletonObject Methods

The methods of the ISkeletonObject class are listed here.

Public Methods

	Name	Description
	Execute (see page 18207)	This is Execute, a member of class ISkeletonObject.
	GetImplementation (see page 18207)	This is GetImplementation, a member of class ISkeletonObject.
	GetSkeleton (see page 18207)	This is GetSkeleton, a member of class ISkeletonObject.

120.1.7.2.1 ISkeletonObject.Execute Method

Pascal

```
function Execute(Operation: PChar; const Strm: IMarshalInBuffer; Cookie: Pointer):
CorbaBoolean; stdcall;
```

C++

```
__fastcall __stdcall CorbaBoolean Execute(const char * Operation, const IMarshalInBuffer Strm,
void * Cookie);
```

Description

This is Execute, a member of class ISkeletonObject.

120.1.7.2.2 ISkeletonObject.GetImplementation Method

Pascal

```
procedure GetImplementation(out Impl: IUnknown); stdcall;
```

C++

```
__fastcall __stdcall GetImplementation(IUnknown Impl);
```

Description

This is GetImplementation, a member of class ISkeletonObject.

120.1.7.2.3 ISkeletonObject.GetSkeleton Method

Pascal

```
procedure GetSkeleton(out Skeleton: ISkeleton); stdcall;
```

C++

```
__fastcall __stdcall GetSkeleton(ISkeleton Skeleton);
```

Description

This is GetSkeleton, a member of class ISkeletonObject.

120.1.8 IStub Interface

Class Hierarchy

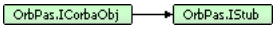

Pascal

```
IStub = interface(ICorbaObj);
```

C++

```
__interface IStub : public ICorbaObj;
```

File

OrbPas

Description

This is class OrbPas.IStub.

120.1.8.1 IStub Members

The following tables list the members exposed by IStub.

Public Methods

	Name	Description
	CorbaObject (see page 18192)	This is CorbaObject, a member of class ICorbaObj.
	CreateRequest (see page 18209)	This is CreateRequest, a member of class IStub.
	Dispatch (see page 18209)	This is Dispatch, a member of class IStub.
	GetInterface (see page 18209)	This is GetInterface, a member of class IStub.
	Hash (see page 18209)	This is Hash, a member of class IStub.
	Invoke (see page 18210)	This is Invoke, a member of class IStub.
	IsA (see page 18210)	This is IsA, a member of class IStub.
	IsLocal (see page 18192)	This is IsLocal, a member of class ICorbaObj.
	NonExistent (see page 18210)	This is NonExistent, a member of class IStub.
	RepositoryID (see page 18210)	This is RepositoryID, a member of class IStub.
	SetPrincipal (see page 18211)	This is SetPrincipal, a member of class IStub.

120.1.8.2 IStub Methods

The methods of the IStub class are listed here.

Public Methods

	Name	Description
	CreateRequest (see page 18209)	This is CreateRequest, a member of class IStub.

	Dispatch (see page 18209)	This is Dispatch, a member of class IStub.
	GetInterface (see page 18209)	This is GetInterface, a member of class IStub.
	Hash (see page 18209)	This is Hash, a member of class IStub.
	Invoke (see page 18210)	This is Invoke, a member of class IStub.
	IsA (see page 18210)	This is IsA, a member of class IStub.
	NonExistent (see page 18210)	This is NonExistent, a member of class IStub.
	RepositoryID (see page 18210)	This is RepositoryID, a member of class IStub.
	SetPrincipal (see page 18211)	This is SetPrincipal, a member of class IStub.

120.1.8.2.1 IStub.CreateRequest Method

Pascal

```
procedure CreateRequest(Operation: PChar; ResponseExpected: CorbaBoolean; out Outbuf:
IMarshalOutBuffer); stdcall;
```

C++

```
__fastcall __stdcall CreateRequest(const char * Operation, CorbaBoolean ResponseExpected,
IMarshalOutBuffer Outbuf);
```

Description

This is CreateRequest, a member of class IStub.

120.1.8.2.2 IStub.Dispatch Method

Pascal

```
function Dispatch(CallDesc: PCallDesc; Params: Pointer; out Result: Variant): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Dispatch(PCallDesc CallDesc, void * Params, Variant Result);
```

Description

This is Dispatch, a member of class IStub.

120.1.8.2.3 IStub.GetInterface Method

Pascal

```
function GetInterface: PCorbaInterfaceDef; stdcall;
```

C++

```
__fastcall __stdcall PCorbaInterfaceDef GetInterface();
```

Description

This is GetInterface, a member of class IStub.

120.1.8.2.4 IStub.Hash Method

Pascal

```
function Hash(Maximum: CorbaULong): CorbaULong; stdcall;
```

C++

```
__fastcall __stdcall CorbaULong Hash(CorbaULong Maximum);
```

Description

This is Hash, a member of class IStub.

120.1.8.2.5 IStub.Invoke Method

Pascal

```
procedure Invoke(const Outbuf: IMarshalOutBuffer; out Inbuf: IMarshalInBuffer); stdcall;
```

C++

```
__fastcall __stdcall Invoke(const IMarshalOutBuffer Outbuf, IMarshalInBuffer Inbuf);
```

Description

This is Invoke, a member of class IStub.

120.1.8.2.6 IStub.IsA Method

Pascal

```
function IsA(LogicalTypeId: PChar): CorbaBoolean; stdcall;
```

C++

```
__fastcall __stdcall CorbaBoolean IsA(const char * LogicalTypeId);
```

Description

This is IsA, a member of class IStub.

120.1.8.2.7 IStub.NonExistent Method

Pascal

```
function NonExistent: CorbaBoolean; stdcall;
```

C++

```
__fastcall __stdcall CorbaBoolean NonExistent();
```

Description

This is NonExistent, a member of class IStub.

120.1.8.2.8 IStub.RepositoryID Method

Pascal

```
function RepositoryID: PChar; stdcall;
```

C++

```
__fastcall __stdcall const char * RepositoryID();
```

Description

This is RepositoryID, a member of class IStub.

120.1.8.2.9 IStub.SetPrincipal Method

Pascal

```
procedure SetPrincipal(Bytes: Pointer; Length: CorbaULong); stdcall;
```

C++

```
__fastcall __stdcall SetPrincipal(void * Bytes, CorbaULong Length);
```

Description

This is SetPrincipal, a member of class IStub.

120.1.9 IStubObject Interface

Class Hierarchy

OrbPas.IStubObject

Pascal

```
IStubObject = interface;
```

C++

```
__interface IStubObject;
```

File

OrbPas

Description

This is class OrbPas.IStubObject.

120.1.9.1 IStubObject Members

The following tables list the members exposed by IStubObject.

Public Methods

	Name	Description
	GetStub (see page 18212)	This is GetStub, a member of class IStubObject.

120.1.9.2 IStubObject Methods

The methods of the IStubObject class are listed here.

Public Methods

	Name	Description
	GetStub (see page 18212)	This is GetStub, a member of class IStubObject.

120.1.9.2.1 IStubObject.GetStub Method

Pascal

```
procedure GetStub(out stub: IStub); stdcall;
```

C++

```
__fastcall __stdcall GetStub(IStub stub);
```

Description

This is GetStub, a member of class IStubObject.

120.1.10 ITypeCode Interface

Class Hierarchy

OrbPas.ITypeCode

Pascal

```
ITypeCode = interface;
```

C++

```
__interface ITypeCode;
```

File

OrbPas

Description

This is class OrbPas.ITypeCode.

120.1.10.1 ITypeCode Members

The following tables list the members exposed by ITypeCode.

Public Methods

	Name	Description
	ContentType (see page 18213)	This is ContentType, a member of class ITypeCode.
	DefaultIndex (see page 18213)	This is DefaultIndex, a member of class ITypeCode.
	DiscriminatorType (see page 18213)	This is DiscriminatorType, a member of class ITypeCode.
	Equal (see page 18214)	This is Equal, a member of class ITypeCode.
	Id (see page 18214)	This is Id, a member of class ITypeCode.
	Kind (see page 18214)	This is Kind, a member of class ITypeCode.
	Length (see page 18214)	This is Length, a member of class ITypeCode.
	MemberCount (see page 18215)	This is MemberCount, a member of class ITypeCode.
	MemberLabel (see page 18215)	This is MemberLabel, a member of class ITypeCode.
	MemberName (see page 18215)	This is MemberName, a member of class ITypeCode.
	MemberType (see page 18215)	This is MemberType, a member of class ITypeCode.
	Name (see page 18215)	This is Name, a member of class ITypeCode.
	Parameter (see page 18216)	This is Parameter, a member of class ITypeCode.
	ParameterCount (see page 18216)	This is ParameterCount, a member of class ITypeCode.

120.1.10.2 ITypeCode Methods

The methods of the ITypeCode class are listed here.

Public Methods

	Name	Description
	ContentType (see page 18213)	This is ContentType, a member of class ITypeCode.
	DefaultIndex (see page 18213)	This is DefaultIndex, a member of class ITypeCode.
	DiscriminatorType (see page 18213)	This is DiscriminatorType, a member of class ITypeCode.
	Equal (see page 18214)	This is Equal, a member of class ITypeCode.
	Id (see page 18214)	This is Id, a member of class ITypeCode.
	Kind (see page 18214)	This is Kind, a member of class ITypeCode.
	Length (see page 18214)	This is Length, a member of class ITypeCode.
	MemberCount (see page 18215)	This is MemberCount, a member of class ITypeCode.
	MemberLabel (see page 18215)	This is MemberLabel, a member of class ITypeCode.
	MemberName (see page 18215)	This is MemberName, a member of class ITypeCode.
	MemberType (see page 18215)	This is MemberType, a member of class ITypeCode.
	Name (see page 18215)	This is Name, a member of class ITypeCode.
	Parameter (see page 18216)	This is Parameter, a member of class ITypeCode.
	ParameterCount (see page 18216)	This is ParameterCount, a member of class ITypeCode.

120.1.10.2.1 ITypeCode.ContentType Method

Pascal

```
procedure ContentType(out tc: ITypeCode); stdcall;
```

C++

```
__fastcall __stdcall ContentType(ITypeCode tc);
```

Description

This is ContentType, a member of class ITypeCode.

120.1.10.2.2 ITypeCode.DefaultIndex Method

Pascal

```
function DefaultIndex: Integer; stdcall;
```

C++

```
__fastcall __stdcall int DefaultIndex();
```

Description

This is DefaultIndex, a member of class ITypeCode.

120.1.10.2.3 ITypeCode.DiscriminatorType Method

Pascal

```
procedure DiscriminatorType(out tc: ITypeCode); stdcall;
```

C++

```
__fastcall __stdcall DiscriminatorType(ITypeCode tc);
```

Description

This is DiscriminatorType, a member of class ITypeCode.

120.1.10.2.4 ITypeCode.Equal Method

Pascal

```
function Equal(const tc: ITypeCode): CorbaBoolean; stdcall;
```

C++

```
__fastcall __stdcall CorbaBoolean Equal(const ITypeCode tc);
```

Description

This is Equal, a member of class ITypeCode.

120.1.10.2.5 ITypeCode.Id Method

Pascal

```
function Id: PChar; stdcall;
```

C++

```
__fastcall __stdcall const char * Id();
```

Description

This is Id, a member of class ITypeCode.

120.1.10.2.6 ITypeCode.Kind Method

Pascal

```
function Kind: TCKind; stdcall;
```

C++

```
__fastcall __stdcall TCKind Kind();
```

Description

This is Kind, a member of class ITypeCode.

120.1.10.2.7 ITypeCode.Length Method

Pascal

```
function Length: CorbaULong; stdcall;
```

C++

```
__fastcall __stdcall CorbaULong Length();
```

Description

This is Length, a member of class ITypeCode.

120.1.10.2.8 ITypeCode.MemberCount Method

Pascal

```
function MemberCount: CorbaULong; stdcall;
```

C++

```
__fastcall __stdcall CorbaULong MemberCount();
```

Description

This is MemberCount, a member of class ITypeCode.

120.1.10.2.9 ITypeCode.MemberLabel Method

Pascal

```
function MemberLabel(index: CorbaULong): PCorbaAny; stdcall;
```

C++

```
__fastcall __stdcall PCorbaAny MemberLabel(CorbaULong index);
```

Description

This is MemberLabel, a member of class ITypeCode.

120.1.10.2.10 ITypeCode.MemberName Method

Pascal

```
function MemberName(index: CorbaULong): PChar; stdcall;
```

C++

```
__fastcall __stdcall const char * MemberName(CorbaULong index);
```

Description

This is MemberName, a member of class ITypeCode.

120.1.10.2.11 ITypeCode.MemberType Method

Pascal

```
procedure MemberType(index: CorbaULong; out tc: ITypeCode); stdcall;
```

C++

```
__fastcall __stdcall MemberType(CorbaULong index, ITypeCode tc);
```

Description

This is MemberType, a member of class ITypeCode.

120.1.10.2.12 ITypeCode.Name Method

Pascal

```
function Name: PChar; stdcall;
```

C++

```
__fastcall __stdcall const char * Name();
```

Description

This is Name, a member of class ITypeCode.

120.1.10.2.13 ITypeCode.Parameter Method

Pascal

```
function Parameter(index: Integer): PCorbaAny; stdcall;
```

C++

```
__fastcall __stdcall PCorbaAny Parameter(int index);
```

Description

This is Parameter, a member of class ITypeCode.

120.1.10.2.14 ITypeCode.ParameterCount Method

Pascal

```
function ParameterCount: Integer; stdcall;
```

C++

```
__fastcall __stdcall int ParameterCount();
```

Description

This is ParameterCount, a member of class ITypeCode.

120.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AnyToVariant (see page 18217)	This is function OrbPas.AnyToVariant.
	BindStub (see page 18217)	This is function OrbPas.BindStub.
	CorbaAnyType (see page 18218)	This is function OrbPas.CorbaAnyType.
	CorbaDuplicateAny (see page 18218)	This is function OrbPas.CorbaDuplicateAny.
	CorbaReleaseAny (see page 18218)	This is function OrbPas.CorbaReleaseAny.
	CorbaStringFree (see page 18219)	This is function OrbPas.CorbaStringFree.
	CorbaWStringFree (see page 18219)	This is function OrbPas.CorbaWStringFree.
	CreateSkeleton (see page 18219)	This is function OrbPas.CreateSkeleton.
	CreateUserException (see page 18220)	This is function OrbPas.CreateUserException.
	InitORB (see page 18220)	This is function OrbPas.InitORB.
	RegisterUserException (see page 18220)	This is function OrbPas.RegisterUserException.
	SequenceToVariant (see page 18221)	This is function OrbPas.SequenceToVariant.
	UnRegisterUserException (see page 18221)	This is function OrbPas.UnRegisterUserException.
	VariantToAny (see page 18221)	This is function OrbPas.VariantToAny.

120.2.1 OrbPas.AnyToVariant Function

Pascal

```
function AnyToVariant(Value: PCorbaAny; OV: PVariant): Boolean;
```

C++

```
Boolean AnyToVariant(PCorbaAny Value, PVariant OV);
```

File

OrbPas

Description

This is function OrbPas.AnyToVariant.

120.2.2 OrbPas.BindStub Function

Pascal

```
procedure BindStub(RepositoryID: PChar; InstanceName: PChar; HostName: PChar; const Orb: IORB;
RefCountServer: CorbaBoolean; out Stub: IStub);
```

C++

```
BindStub(const char * RepositoryID, const char * InstanceName, const char * HostName, const
IORB Orb, CorbaBoolean RefCountServer, IStub Stub);
```

File

OrbPas

Description

This is function OrbPas.BindStub.

120.2.3 OrbPas.CorbaAnyType Function

Pascal

```
procedure CorbaAnyType(Any: PCorbaAny; out TypeCode: ITypeCode);
```

C++

```
CorbaAnyType(PCorbaAny Any, ITypeCode TypeCode);
```

File

OrbPas

Description

This is function OrbPas.CorbaAnyType.

120.2.4 OrbPas.CorbaDuplicateAny Function

Pascal

```
function CorbaDuplicateAny(Any: PCorbaAny): PCorbaAny;
```

C++

```
PCorbaAny CorbaDuplicateAny(PCorbaAny Any);
```

File

OrbPas

Description

This is function OrbPas.CorbaDuplicateAny.

120.2.5 OrbPas.CorbaReleaseAny Function

Pascal

```
procedure CorbaReleaseAny(Any: PCorbaAny);
```

C++

```
CorbaReleaseAny(PCorbaAny Any);
```

File

OrbPas

Description

This is function OrbPas.CorbaReleaseAny.

120.2.6 OrbPas.CorbaStringFree Function

Pascal

```
procedure CorbaStringFree(Str: PChar);
```

C++

```
CorbaStringFree(const char * Str);
```

File

OrbPas

Description

This is function OrbPas.CorbaStringFree.

120.2.7 OrbPas.CorbaWStringFree Function

Pascal

```
procedure CorbaWStringFree(Str: PWideChar);
```

C++

```
CorbaWStringFree(PWideChar Str);
```

File

OrbPas

Description

This is function OrbPas.CorbaWStringFree.

120.2.8 OrbPas.CreateSkeleton Function

Pascal

```
procedure CreateSkeleton(InterfaceName: PChar; const DelphiObject: ISkeletonObject; Serialize:
CorbaBoolean; InstanceName: PChar; RepositoryID: PChar; ClientRefCount: CorbaBoolean; out
Skeleton: ISkeleton);
```

C++

```
CreateSkeleton(const char * InterfaceName, const ISkeletonObject DelphiObject, CorbaBoolean
Serialize, const char * InstanceName, const char * RepositoryID, CorbaBoolean ClientRefCount,
ISkeleton Skeleton);
```

File

OrbPas

Description

This is function OrbPas.CreateSkeleton.

120.2.9 OrbPas.CreateUserException Function

Pascal

```
function CreateUserException(copy: TCopyUserExceptionProc; throwobj: TThrowUserExceptionProc):
PUserExceptionProxy;
```

C++

```
PUserExceptionProxy CreateUserException(TCopyUserExceptionProc copy, TThrowUserExceptionProc
throwobj);
```

File

OrbPas

Description

This is function OrbPas.CreateUserException.

120.2.10 OrbPas.InitORB Function

Pascal

```
procedure InitORB(const Argv: TArgv; out Orb: IORB);
```

C++

```
InitORB(const TArgv Argv, IORB Orb);
```

File

OrbPas

Description

This is function OrbPas.InitORB.

120.2.11 OrbPas.RegisterUserException Function

Pascal

```
function RegisterUserException(Name: PChar; RepositoryID: PChar; Factory:
TUserExceptionFactoryProc): PExceptionDescription;
```

C++

```
PExceptionDescription RegisterUserException(const char * Name, const char * RepositoryID,
TUserExceptionFactoryProc Factory);
```

File

OrbPas

Description

This is function OrbPas.RegisterUserException.

120.2.12 OrbPas.SequenceToVariant Function

Pascal

```
function SequenceToVariant(Value: PCorbaAny; OV: PVariant): Boolean;
```

C++

```
Boolean SequenceToVariant(PCorbaAny Value, PVariant OV);
```

File

OrbPas

Description

This is function OrbPas.SequenceToVariant.

120.2.13 OrbPas.UnRegisterUserException Function

Pascal

```
procedure UnRegisterUserException(Description: PExceptionDescription);
```

C++

```
UnRegisterUserException(PExceptionDescription Description);
```

File

OrbPas

Description

This is function OrbPas.UnRegisterUserException.

120.2.14 OrbPas.VariantToAny Function

Pascal

```
function VariantToAny(Value: PVariant): PCorbaAny;
```

C++

```
PCorbaAny VariantToAny(PVariant Value);
```

File

OrbPas

Description

This is function OrbPas.VariantToAny.

120.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCKind (see page 18222)	This is record OrbPas.TCKind.

Records

	Name	Description
	TStructMember (see page 18223)	This is record OrbPas.TStructMember.

120.3.1 OrbPas.TCKind Enumeration

Pascal

```
TCKind = (
  tk_null,
  tk_void,
  tk_short,
  tk_long,
  tk_ushort,
  tk_ulong,
  tk_float,
  tk_double,
  tk_boolean,
  tk_char,
  tk_octet,
  tk_any,
  tk_TypeCode,
  tk_Principal,
  tk_objref,
  tk_struct,
  tk_union,
  tk_enum,
  tk_string,
  tk_sequence,
  tk_array,
  tk_alias,
  tk_except,
  tk_longlong,
  tk_ulonglong,
  tk_longdouble,
  tk_wchar,
  tk_wstring,
  tk_fixed
);
```

C++

```
enum TCKind {
  tk_null,
  tk_void,
  tk_short,
  tk_long,
  tk_ushort,
```

```
    tk_ulong,
    tk_float,
    tk_double,
    tk_boolean,
    tk_char,
    tk_octet,
    tk_any,
    tk_TypeCode,
    tk_Principal,
    tk_objref,
    tk_struct,
    tk_union,
    tk_enum,
    tk_string,
    tk_sequence,
    tk_array,
    tk_alias,
    tk_except,
    tk_longlong,
    tk_ulonglong,
    tk_longdouble,
    tk_wchar,
    tk_wstring,
    tk_fixed
  };
```

File

OrbPas

Description

This is record OrbPas.TCKind.

120.3.2 OrbPas.TStructMember Record

Pascal

```
  TStructMember = record
    Name: string;
    TC: ITypeCode;
  end;
```

C++

```
  struct TStructMember {
    AnsiString Name;
    ITypeCode TC;
  };
```

File

OrbPas

Description

This is record OrbPas.TStructMember.

120.4 Types

The following table lists types in this documentation.

Types

Name	Description
CorbaBoolean (see page 18224)	This is type OrbPas.CorbaBoolean.
CorbaULong (see page 18224)	This is type OrbPas.CorbaULong.
PCorbaAny (see page 18225)	This is type OrbPas.PCorbaAny.
PCorbaInterfaceDef (see page 18225)	This is type OrbPas.PCorbaInterfaceDef.
PCorbaObject (see page 18225)	This is type OrbPas.PCorbaObject.
PCorbaOrb (see page 18226)	This is type OrbPas.PCorbaOrb.
PExceptionDescription (see page 18226)	This is type OrbPas.PExceptionDescription.
PMarshalInbuffer (see page 18226)	This is type OrbPas.PMarshalInbuffer.
PMarshalOutbuffer (see page 18227)	This is type OrbPas.PMarshalOutbuffer.
PUserExceptionProxy (see page 18227)	This is type OrbPas.PUserExceptionProxy.
TArgv (see page 18227)	This is type OrbPas.TArgv.
TCopyUserExceptionProc (see page 18228)	This is type OrbPas.TCopyUserExceptionProc.
TStructMembers (see page 18228)	This is type OrbPas.TStructMembers.
TThrowUserExceptionProc (see page 18228)	This is type OrbPas.TThrowUserExceptionProc.
TUserExceptionFactoryProc (see page 18229)	This is type OrbPas.TUserExceptionFactoryProc.

120.4.1 OrbPas.CorbaBoolean Type

Pascal

```
CorbaBoolean = ByteBool;
```

C++

```
ByteBool CorbaBoolean;
```

File

OrbPas

Description

This is type OrbPas.CorbaBoolean.

120.4.2 OrbPas.CorbaULong Type

Pascal

```
CorbaULong = Integer;
```

C++

```
int CorbaULong;
```

File

OrbPas

Description

This is type OrbPas.CorbaULong.

120.4.3 OrbPas.PCorbaAny Type

Pascal

```
PCorbaAny = type Pointer;
```

C++

```
type Pointer PCorbaAny;
```

File

OrbPas

Description

This is type OrbPas.PCorbaAny.

120.4.4 OrbPas.PCorbaInterfaceDef Type

Pascal

```
PCorbaInterfaceDef = type Pointer;
```

C++

```
type Pointer PCorbaInterfaceDef;
```

File

OrbPas

Description

This is type OrbPas.PCorbaInterfaceDef.

120.4.5 OrbPas.PCorbaObject Type

Pascal

```
PCorbaObject = type Pointer;
```

C++

```
type Pointer PCorbaObject;
```

File

OrbPas

Description

This is type OrbPas.PCorbaObject.

120.4.6 OrbPas.PCorbaOrb Type

Pascal

```
PCorbaOrb = type Pointer;
```

C++

```
type Pointer PCorbaOrb;
```

File

OrbPas

Description

This is type OrbPas.PCorbaOrb.

120.4.7 OrbPas.PExceptionDescription Type

Pascal

```
PExceptionDescription = type Pointer;
```

C++

```
type Pointer PExceptionDescription;
```

File

OrbPas

Description

This is type OrbPas.PExceptionDescription.

120.4.8 OrbPas.PMarshalInbuffer Type

Pascal

```
PMarshalInbuffer = type Pointer;
```

C++

```
type Pointer PMarshalInbuffer;
```

File

OrbPas

Description

This is type OrbPas.PMarshalInbuffer.

120.4.9 OrbPas.PMarshalOutbuffer Type

Pascal

```
PMarshalOutbuffer = type Pointer;
```

C++

```
type Pointer PMarshalOutbuffer;
```

File

OrbPas

Description

This is type OrbPas.PMarshalOutbuffer.

120.4.10 OrbPas.PUserExceptionProxy Type

Pascal

```
PUserExceptionProxy = type Pointer;
```

C++

```
type Pointer PUserExceptionProxy;
```

File

OrbPas

Description

This is type OrbPas.PUserExceptionProxy.

120.4.11 OrbPas.TArgv Type

Pascal

```
TArgv = array of string;
```

C++

```
array of string TArgv;
```

File

OrbPas

Description

This is type OrbPas.TArgv.

120.4.12 OrbPas.TCopyUserExceptionProc Type

Pascal

```
TCopyUserExceptionProc = procedure (const InBuf: IMarshalInBuffer) of object;
```

C++

```
(const InBuf: IMarshalInBuffer) ( TCopyUserExceptionProc)();
```

File

OrbPas

Description

This is type OrbPas.TCopyUserExceptionProc.

120.4.13 OrbPas.TStructMembers Type

Pascal

```
TStructMembers = array of TStructMember;
```

C++

```
array of TStructMember TStructMembers;
```

File

OrbPas

Description

This is type OrbPas.TStructMembers.

120.4.14 OrbPas.TThrowUserExceptionProc Type

Pascal

```
TThrowUserExceptionProc = procedure of object;
```

C++

```
( TThrowUserExceptionProc)();
```

File

OrbPas

Description

This is type OrbPas.TThrowUserExceptionProc.

120.4.15 OrbPas.TUserExceptionFactoryProc Type

Pascal

```
TUserExceptionFactoryProc = function : PUserExceptionProxy;
```

C++

```
: PUserExceptionProxy ( TUserExceptionFactoryProc)();
```

File

OrbPas

Description

This is type OrbPas.TUserExceptionFactoryProc.

121 Outline Namespace

This is namespace Outline.

Topics

Name	Description
Classes (see page 18233)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 18366)	The following table lists structs, records, enums in this documentation.
Types (see page 18371)	The following table lists types in this documentation.

Classes

	Name	Description
	EOutlineError (see page 18233)	EOutlineError is the exception class for outline components.
	OutlineError (see page 18234)	OutlineError is a special exception class for invalid outline-node indexes.
	TCustomOutline (see page 18234)	TCustomOutline is the base class for controls that display outline nodes (TOutlineNode (see page 18349) objects).
	TOutline (see page 18279)	TOutline is a multilevel outline that displays the values of TOutlineNode (see page 18349) objects.
	TOutlineNode (see page 18349)	TOutlineNode represents a node in a TCustomOutline (see page 18234) object.

Structs, Records, Enums

	Name	Description
	TAttachMode (see page 18366)	TAttachMode indicates how an outline node should be inserted into an outline.
	TOutlineBitmap (see page 18367)	This is record Outline.TOutlineBitmap.
	TOutlineNodeCompare (see page 18367)	This is record Outline.TOutlineNodeCompare.
	TOutlineOption (see page 18368)	TOutlineOptions (see page 18373) represents the style in which an outline is drawn.
	TOutlineStyle (see page 18368)	TOutlineStyle determines how the nodes of an outline are drawn.
	TOutlineType (see page 18369)	TOutlineType indicates whether an outline is owner-drawn.

Types

Name	Description
EOutlineChange (see page 18371)	EOutlineChange is the type of event handlers that respond when the nodes in an outline control change.
TBitmapArray (see page 18371)	This is type Outline.TBitmapArray.
TBitmapArrayRange (see page 18372)	This is type Outline.TBitmapArrayRange.
TChangeRange (see page 18372)	TChangeRange indicates the direction in which a node is moved.

TOutlineBitmaps (see page 18372)	This is type Outline.TOutlineBitmaps.
TOutlineOptions (see page 18373)	TOutlineOptions represents the style in which an outline is drawn.

121.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EOutlineError (see page 18233)	EOutlineError is the exception class for outline components.
	OutlineError (see page 18234)	OutlineError is a special exception class for invalid outline-node indexes.
	TCustomOutline (see page 18234)	TCustomOutline is the base class for controls that display outline nodes (TOutlineNode (see page 18349) objects).
	TOutline (see page 18279)	TOutline is a multilevel outline that displays the values of TOutlineNode (see page 18349) objects.
	TOutlineNode (see page 18349)	TOutlineNode represents a node in a TCustomOutline (see page 18234) object.

121.1.1 EOutlineError Class

EOutlineError is the exception class for outline components.

Class Hierarchy

Pascal

```
EOutlineError = class(Exception);
```

C++

```
class EOutlineError : public Exception;
```

File

Outline

Description

EOutlineError is raised when an error occurs as an application works with an outline component. For example, an EOutlineError exception occurs when an application

Attempts to access an outline using an invalid node index.

Attempts to expand a node whose parent isn't expanded.

Attempts to load items from a file using an invalid file extension.

Note: EOutlineError is raised by the TOutline (see page 18279) component, which is maintained for backward compatibility.

See Also

OutlineError (see page 18234)

121.1.2 OutlineError Class

OutlineError is a special exception class for invalid outline-node indexes.

Class Hierarchy

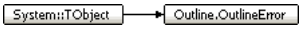

Pascal

```
OutlineError = class(TObject);
```

C++

```
class OutlineError : public TObject;
```

File

Outline

Description

OutlineError is raised when an application cannot access a specified outline node. In most cases, this occurs because the index used is invalid.

OutlineError is raised internally to abort look-ups of outline components. Because it does not descend from Exception, it never appears at runtime—even if the debugger is set to break on exceptions. Thus it is a useful tool for aborting deep recursions.

Note: OutlineError is raised by the TOutline (see page 18279) component, which is maintained for backward compatibility.

See Also

EOutlineError (see page 18233)

Exception (see page 22750)

121.1.3 TCustomOutline Class

TCustomOutline is the base class for controls that display outline nodes (TOutlineNode (see page 18349) objects).

Class Hierarchy

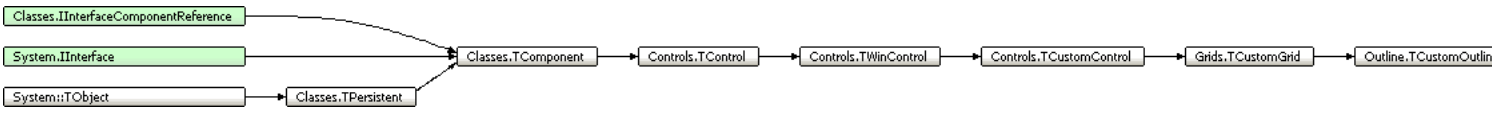

Pascal

```
TCustomOutline = class(TCustomGrid);
```

C++

```
class TCustomOutline : public TCustomGrid;
```

File

Outline

Description

Use TCustomOutline as a base class when defining a control that displays outline nodes in a two-dimensional layout. TCustomOutline introduces the ability to manage a hierarchy of outline nodes, including the ability to control the appearance and layout of those nodes and to expand or collapse branches of the outline tree.

Do not create instances of TCustomOutline. To add an outline control to a form, use the TOutline (see page 18279) object, which publishes many of the properties, events, and methods of TCustomOutline, or TTreeView, which wraps the Windows control that performs the same function.

See Also

TOutline (see page 18279)

TOutlineNode (see page 18349)

TTreeView (see page 5293)

121.1.3.1 TCustomOutline Members

The following tables list the members exposed by TCustomOutline.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnCollapse (see page 18278)	Occurs immediately after a node in the outline is collapsed.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 18278)	Occurs when a node must be drawn in an owner-draw outline.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnExpand (see page 18279)	Occurs immediately after a node in the outline is expanded.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.

	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 14814)	This is DesignOptionsBoost, a member of class TCustomGrid.
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FGridState (see page 14814)	This is FGridState, a member of class TCustomGrid.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveCellExtents (see page 14814)	This is FSaveCellExtents, a member of class TCustomGrid.
	VirtualView (see page 14814)	This is VirtualView, a member of class TCustomGrid.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AdjustSize (see page 14816)	Adds or deletes rows or columns from the grid.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BadIndex (see page 18253)	Indicates whether a node's index is up to date.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	BeginColumnDrag (see page 14817)	Starts the dragging of a column in the grid.

	BeginRowDrag (see page 14817)	Starts the dragging of a row in the grid.
	BoxRect (see page 14818)	Returns the screen coordinates of a specified set of cells.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcDrawInfo (see page 14818)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 14818)	Calculates the pixel locations of the fixed cells.
	CalcSizingState (see page 14819)	Determines the appearance of the mouse cursor during resizing.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanEditAcceptKey (see page 14819)	Indicates whether the Key parameter can be entered into the current cell.
	CanEditModify (see page 14819)	Indicates whether the current cell in the grid can be edited.
	CanEditShow (see page 14820)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 14820)	Indicates whether the grid should respond to a given key combination.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	CellRect (see page 14821)	Returns the screen coordinates of a cell in the grid.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeGridOrientation (see page 14821)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CheckColumnDrag (see page 14821)	CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 14822)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
	Click (see page 18254)	Updates the SelectedItem (see page 18277) property.
	Collapse (see page 18254)	Generates an OnCollapse (see page 18278) event.
	ColumnMoved (see page 14822)	Provides the interface for a method that responds when the position of a column changes.
	ColWidthsChanged (see page 14822)	Responds when the column widths change.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateEditor (see page 14823)	Creates the inplace editor control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 14823)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 18254)	Expands or collapses the selected node.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 18255)	Creates a binary property for customized bitmap values.
	DeleteColumn (see page 14824)	Deletes a specified column from the grid.
	DeleteNode (see page 18256)	Fixes up the outline after a node has been deleted.
	DeleteRow (see page 14825)	Deletes a specified from the grid.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.

	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoExit (see page 14825)	Hides the inplace editor when selection moves to another control in a form.
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelDown (see page 14825)	Performs tasks when the mouse wheel is rotated downward.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoMouseWheelUp (see page 14826)	Performs tasks when the mouse wheel is rotated upward.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawCell (see page 18256)	Draws a node in the outline.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EndColumnDrag (see page 14827)	Ends the dragging of a column in the grid.
	EndRowDrag (see page 14827)	Ends the dragging of a row in the grid.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	Expand (see page 18257)	Generates an OnExpand (see page 18279) event.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	FocusCell (see page 14827)	Changes the current cell to a specified cell.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.

	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEditLimit (see page 14828)	Provides the interface for a method that limits the number of characters the user can type into cells.
	GetEditMask (see page 14828)	Provides the interface for a method that returns an input mask for the specified cell.
	GetEditStyle (see page 14828)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
	GetEditText (see page 14829)	Returns the value of the indicated cell formatted for editing.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetGridHeight (see page 14829)	Returns the value of the GridHeight (see page 14850) property.
	GetGridWidth (see page 14830)	Returns the value of the GridWidth (see page 14851) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HideEdit (see page 14830)	Hides the inplace editor.
	HideEditor (see page 14830)	Hides the inplace editor when it is not in use.
	InvalidateCell (see page 14831)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCol (see page 14831)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	InvalidateEditor (see page 14832)	Causes the inplace editor to repaint itself, moving to a new location if necessary.
	InvalidateGrid (see page 14832)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 14832)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 18262)	Provides special processing when the user presses a key.
	KeyPress (see page 18262)	Provides special processing when a key is pressed.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 18263)	Sets up the outline after it has been streamed in.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 18264)	Updates the SelectedItem (see page 18277) property.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseMove (see page 14834)	Provides special processing when the user moves the mouse.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	MouseUp (see page 14835)	Provides special processing when the user releases the mouse button.
	Move (see page 18265)	Repositions a node in the outline.
	MoveColRow (see page 14835)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 14836)	Moves a column to a new position.

	MoveRow (see page 14836)	Moves a row to a new position.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 5920)	Renders the image of a custom control.
V	Paint (see page 14836)	Draws the image of the grid control on the screen.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	RowHeightsChanged (see page 14837)	Responds when the row heights change.
	RowMoved (see page 14837)	Provides the interface for a method that responds when the position of a row changes.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	ScrollData (see page 14838)	Scrolls the visual image of the cells in the grid.
V	SelectCell (see page 14838)	Determines whether a particular cell in the grid can be selected.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayWidth (see page 18266)	Sets the display width of the outline.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEditText (see page 14839)	Provides the interface for a method that updates the text associated with a cell.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetGoodIndex (see page 18266)	Returns the first node in the Items (see page 18271) array that must be reindexed to make the index of the Value parameter correct.

	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetLevel (see page 18267)	Sets the level of an outline node.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	ShowEditor (see page 14839)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 14839)	Displays the inplace editor, with its text set to the specified character.
	SizeChanged (see page 14840)	Responds to changes in the size of the grid.
	Sizing (see page 14840)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	TimedScroll (see page 14840)	Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.
	TopLeftChanged (see page 14841)	Updates the position of the inplace editor when the grid scrolls.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateDesigner (see page 14841)	Notifies the designer when rows and columns of the grid are moved or resized.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.

	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 14843)	Determines whether a single line border is drawn around the grid.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Col (see page 14843)	Specifies the index of the column that contains the selected cell.
	ColCount (see page 14844)	Specifies the number of columns in the grid.
	Color (see page 5812)	Specifies the background color of the control.
	Color (see page 14844)	Specifies the background color of the control.
	ColWidths (see page 14845)	Indicates the width (in pixels) of all the columns in the grid.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColWidth (see page 14846)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 14847)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 14847)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EditorMode (see page 14848)	Determines whether the current cell can be edited.
	FixedColor (see page 14848)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 14848)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 14849)	Specifies the number of rows on the top of the grid that cannot be scrolled.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
R	GridHeight (see page 14850)	Specifies the height of the grid in pixels.
	GridLineWidth (see page 14850)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
R	GridWidth (see page 14851)	Specifies the width of the grid in pixels.
R	HitTest (see page 14851)	Indicates the current position of the mouse cursor.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
R	InplaceEditor (see page 14852)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	ItemHeight (see page 18270)	Specifies the height of each node in the outline.
	ItemSeparator (see page 18271)	Specifies a delimiting string for outline nodes.
	LeftCol (see page 14852)	Specifies the index of the first visible scrollable column in the grid.
	Lines (see page 18272)	Lists the Text property values of all nodes in the outline.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	Options (see page 18272)	Determines how the nodes in the outline are drawn.
	OutlineStyle (see page 18273)	Specifies how the outline tree is displayed.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 14853)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PictureClosed (see page 18274)	Specifies the bitmap used to represent a collapsed node that has subitems.

	PictureLeaf (see page 18274)	Specifies the bitmap used to represent an item that does not have subitems.
	PictureMinus (see page 18275)	Specifies the bitmap used to represent an expanded item that has subitems.
	PictureOpen (see page 18275)	Specifies the bitmap used to represent an expanded item that has subitems.
	PicturePlus (see page 18276)	Specifies the bitmap used to represent a collapsed item that has subitems.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	RowCount (see page 14854)	Specifies the number of rows in the grid.
	RowHeights (see page 14855)	Gives the height (in pixels) of all rows in the grid.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ScrollBars (see page 14856)	Specifies whether the grid includes horizontal and vertical scroll bars.
	Selection (see page 14856)	Indicates the boundaries of the current selection.
	Style (see page 18277)	Determines whether the nodes and tree are drawn automatically.
	TabStops (see page 14858)	Indicates whether the user can tab to specified columns in the grid.
	Text (see page 5842)	Contains a text string associated with the control.
	TopRow (see page 14858)	Specifies the index of the first visible scrollable row in the grid.
R	VisibleColCount (see page 14859)	Indicates the number of scrollable columns visible in the grid.
R	VisibleRowCount (see page 14859)	Indicates the number of scrollable rows visible in the grid.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Add (see page 18250)	Adds a node to the outline.
	AddChild (see page 18250)	Adds a child node to a node in the outline.
	AddChildObject (see page 18251)	Adds a child node containing data to a node in the outline.
	AddObject (see page 18252)	Adds a node containing data to the outline.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BeginUpdate (see page 18253)	Prevents the updating of the outline until the EndUpdate (see page 18257) method is called.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 18253)	Removes all items from the outline.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.

	Create (see page 18248)	Creates an instance of TCustomOutline (see page 18234).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 18255)	Deletes a node from the outline.
	Destroy (see page 18248)	Destroys an instance of TCustomOutline (see page 18234).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	EndUpdate (see page 18257)	Performs the reindexing and repaints deferred by a call to BeginUpdate (see page 18253).
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	FullCollapse (see page 18257)	Causes all top-level nodes in the outline to collapse.
	FullExpand (see page 18258)	Expands every node in the outline.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetDataItem (see page 18258)	Returns the index of the first node in the outline that has the Value pointer as its Data property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItem (see page 18259)	Returns the index of the node at the position indicated by the X and Y parameters.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNodeDisplayWidth (see page 18259)	Returns the length of the row containing a specified node.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.

	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextItem (see page 18259)	Returns the index of the first node in the outline with the Value parameter as its Text property.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	GetVisibleNode (see page 18260)	Returns the TOutlineNode (see page 18349) object for the node indicated by a position in the image of the outline.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 18260)	Inserts a node at a specified position.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InsertObject (see page 18261)	Inserts a node containing data at a specified position.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	LoadFromFile (see page 18263)	Loads the outline from a text image stored in a file.
	LoadFromStream (see page 18264)	Loads the outline from a text image read from a stream.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseCoord (see page 14834)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

	SaveToFile (see page 18265)	Saves the Lines (see page 18272) property to a file.
	SaveToStream (see page 18265)	Writes the Lines (see page 18272) property to a stream.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	SetUpdateState (see page 18267)	Enables or disables the maintenance of the node indexes.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	Canvas (see page 18268)	Specifies the TCanvas object that presents a drawing surface for the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.

R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
R	ItemCount (see page 18270)	Indicates the number of nodes in the outline.
R	Items (see page 18271)	Lists the nodes in the outline.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	Row (see page 18276)	Specifies the index of the row that contains the selected cell.
	SelectedItem (see page 18277)	Specifies the index of the selected node.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.

	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	TabStop (see page 14857)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

121.1.3.2 TCustomOutline.Create Constructor

Creates an instance of TCustomOutline (see page 18234).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomOutline(TComponent * AOwner);
```

Description

Call Create to instantiate a TCustomOutline (see page 18234) object at runtime. Outlines placed on forms at design time are created automatically.

Create initializes the outline object, setting the values of some internal properties, as well as setting

Color to clWindow and ParentColor to false.

Height to 97 and Width to 121.

ItemSeparator (see page 18271) to '\'.

Options (see page 18272) to [ooDrawTreeRoot, ooDrawFocusRect].

OutlineStyle (see page 18273) to osTreePictureText.

See Also

Height (see page 5825)

ItemSeparator (see page 18271)

Options (see page 18272)

OutlineStyle (see page 18273)

Color (see page 5812)

ParentColor (see page 5834)

Create (see page 5918)

Width (see page 5849)

121.1.3.3 TCustomOutline.Destroy Destructor

Destroys an instance of TCustomOutline (see page 18234).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomOutline();
```

Description

Destroy frees all memory associated with the outline. While the destructor frees the TOutlineNode (see page 18349) objects that comprise the outline, any objects associated with those nodes are not freed.

121.1.3.4 TCustomOutline Methods

The methods of the TCustomOutline class are listed here.

Protected Methods

	Name	Description
	BadIndex (see page 18253)	Indicates whether a node's index is up to date.
V	Click (see page 18254)	Updates the SelectedItem (see page 18277) property.
	Collapse (see page 18254)	Generates an OnCollapse (see page 18278) event.
V	DblClick (see page 18254)	Expands or collapses the selected node.
V	DefineProperties (see page 18255)	Creates a binary property for customized bitmap values.
	DeleteNode (see page 18256)	Fixes up the outline after a node has been deleted.
V	DrawCell (see page 18256)	Draws a node in the outline.
	Expand (see page 18257)	Generates an OnExpand (see page 18279) event.
V	KeyDown (see page 18262)	Provides special processing when the user presses a key.
V	KeyPress (see page 18262)	Provides special processing when a key is pressed.
V	Loaded (see page 18263)	Sets up the outline after it has been streamed in.
V	MouseDown (see page 18264)	Updates the SelectedItem (see page 18277) property.
	Move (see page 18265)	Repositions a node in the outline.
	SetDisplayWidth (see page 18266)	Sets the display width of the outline.
	SetGoodIndex (see page 18266)	Returns the first node in the Items (see page 18271) array that must be reindexed to make the index of the Value parameter correct.
	SetLevel (see page 18267)	Sets the level of an outline node.

Public Methods

	Name	Description
	Add (see page 18250)	Adds a node to the outline.
	AddChild (see page 18250)	Adds a child node to a node in the outline.
	AddChildObject (see page 18251)	Adds a child node containing data to a node in the outline.
	AddObject (see page 18252)	Adds a node containing data to the outline.
	BeginUpdate (see page 18253)	Prevents the updating of the outline until the EndUpdate (see page 18257) method is called.
	Clear (see page 18253)	Removes all items from the outline.
	Delete (see page 18255)	Deletes a node from the outline.
	EndUpdate (see page 18257)	Performs the reindexing and repaints deferred by a call to BeginUpdate (see page 18253).
	FullCollapse (see page 18257)	Causes all top-level nodes in the outline to collapse.
	FullExpand (see page 18258)	Expands every node in the outline.
	GetDataItem (see page 18258)	Returns the index of the first node in the outline that has the Value pointer as its Data property.
	GetItem (see page 18259)	Returns the index of the node at the position indicated by the X and Y parameters.
	GetNodeDisplayWidth (see page 18259)	Returns the length of the row containing a specified node.
	GetTextItem (see page 18259)	Returns the index of the first node in the outline with the Value parameter as its Text property.
	GetVisibleNode (see page 18260)	Returns the TOutlineNode (see page 18349) object for the node indicated by a position in the image of the outline.
	Insert (see page 18260)	Inserts a node at a specified position.

	InsertObject (see page 18261)	Inserts a node containing data at a specified position.
	LoadFromFile (see page 18263)	Loads the outline from a text image stored in a file.
	LoadFromStream (see page 18264)	Loads the outline from a text image read from a stream.
	SaveToFile (see page 18265)	Saves the Lines (see page 18272) property to a file.
	SaveToStream (see page 18265)	Writes the Lines (see page 18272) property to a stream.
	SetUpdateState (see page 18267)	Enables or disables the maintenance of the node indexes.

121.1.3.4.1 TCustomOutline.Add Method

Adds a node to the outline.

Pascal

```
function Add(Index: LongInt; const Text: string): LongInt;
```

C++

```
__fastcall LongInt Add(LongInt Index, const AnsiString Text);
```

Description

Use Add to add a TOutlineNode (see page 18349) object to the outline. Text specifies the Text property value of the new node.

Index specifies where to add the new node. The new node is positioned in the outline as the last sibling of the node identified by Index. To add nodes to the beginning of the outline, specify zero (0) as the Index parameter.

Nodes that appear after the new node are moved down one row and reindexed in the Items (see page 18271) property array with valid Index values. If BeginUpdate (see page 18253) has been called, the Items (see page 18271) array is not reindexed until the EndUpdate (see page 18257) method is called.

Add returns the Index property of the new node.

See Also

AddChild (see page 18250)

AddObject (see page 18252)

BeginUpdate (see page 18253)

Delete (see page 18255)

EndUpdate (see page 18257)

Insert (see page 18260)

Items (see page 18271)

Index (see page 18362)

Text (see page 18364)

121.1.3.4.2 TCustomOutline.AddChild Method

Adds a child node to a node in the outline.

Pascal

```
function AddChild(Index: LongInt; const Text: string): LongInt;
```

C++

```
__fastcall LongInt AddChild(LongInt Index, const AnsiString Text);
```

Description

Use AddChild to add a TOutlineNode (see page 18349) object as the child of a node in the outline. Text specifies the Text property value of the new node.

Index specifies where to add the new node. The new node is positioned in the outline as the last child of the node identified by Index. To add nodes to the top level of the outline (off the root), specify zero (0) as the Index parameter.

Nodes that appear after the new node are moved down one row and reindexed in the Items (see page 18271) property array with valid Index values. If BeginUpdate (see page 18253) has been called, the Items (see page 18271) array is not reindexed until the EndUpdate (see page 18257) method is called.

AddChild returns the Index property of the new node.

See Also

Add (see page 18250)

AddChildObject (see page 18251)

BeginUpdate (see page 18253)

Delete (see page 18255)

EndUpdate (see page 18257)

Insert (see page 18260)

Items (see page 18271)

Index (see page 18362)

Text (see page 18364)

121.1.3.4.3 TCustomOutline.AddChildObject Method

Adds a child node containing data to a node in the outline.

Pascal

```
function AddChildObject(Index: LongInt; const Text: string; const Data: Pointer): LongInt;
```

C++

```
__fastcall LongInt AddChildObject(LongInt Index, const AnsiString Text, const void * Data);
```

Description

Use AddChildObject to add a TOutlineNode (see page 18349) object as the child of a node in the outline and assign it a data pointer. Text specifies the value of the Text property for the new node. Data specifies the value of the Data property for the new node.

Index specifies where to add the new node. The new node is positioned in the outline as the last child of the node identified by Index. To add nodes to the top level of the outline (off the root), specify zero (0) as the Index parameter.

Nodes that appear after the new node are moved down one row and reindexed in the Items (see page 18271) property array with valid Index values. If BeginUpdate (see page 18253) has been called, the Items (see page 18271) array is not reindexed until the EndUpdate (see page 18257) method is called.

AddChildObject returns the Index property of the new node.

See Also

AddChild (see page 18250)

AddObject (see page 18252)

BeginUpdate (see page 18253)

Delete (see page 18255)

EndUpdate (see page 18257)

InsertObject (see page 18261)

Items (see page 18271)

Data (see page 18360)

Index (see page 18362)

Text (see page 18364)

121.1.3.4.4 TCustomOutline.AddObject Method

Adds a node containing data to the outline.

Pascal

```
function AddObject(Index: LongInt; const Text: string; const Data: Pointer): LongInt;
```

C++

```
__fastcall LongInt AddObject(LongInt Index, const AnsiString Text, const void * Data);
```

Description

Use AddObject to add a TOutlineNode (see page 18349) object to the outline and assign it a data pointer. Text specifies the value of the Text property for the new node. Data specifies the value of the Data property for the new node.

Index specifies where to add the new node. The new node is positioned in the outline as the last sibling of the node identified by Index. To add nodes to the beginning of the outline, specify zero (0) as the Index parameter.

Nodes that appear after the new node are moved down one row and reindexed in the Items (see page 18271) property array with valid Index values. If BeginUpdate (see page 18253) has been called, the Items (see page 18271) array is not reindexed until the EndUpdate (see page 18257) method is called.

AddChild (see page 18250) returns the Index property of the new node.

See Also

Add (see page 18250)

AddChildObject (see page 18251)

BeginUpdate (see page 18253)

Delete (see page 18255)

EndUpdate (see page 18257)

InsertObject (see page 18261)

Items (see page 18271)

Data (see page 18360)

Index (see page 18362)

Text (see page 18364)

121.1.3.4.5 TCustomOutline.BadIndex Method

Indicates whether a node's index is up to date.

Pascal

```
function BadIndex(Value: TOutlineNode): Boolean;
```

C++

```
__fastcall Boolean BadIndex(TOutlineNode Value);
```

Description

Use BadIndex to determine whether the Index property of the TOutlineNode (see page 18349) object specified by the Value parameter accurately reflects the position of the node in the outline.

If nodes are added or removed after a call to BeginUpdate (see page 18253), the indexes may be out of date until the EndUpdate (see page 18257) method is called.

See Also

BeginUpdate (see page 18253)

EndUpdate (see page 18257)

Index (see page 18362)

121.1.3.4.6 TCustomOutline.BeginUpdate Method

Prevents the updating of the outline until the EndUpdate (see page 18257) method is called.

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

Call BeginUpdate before adding, removing, or moving multiple outline items. BeginUpdate speeds processing time by deferring repaints and the adjustment of index values until EndUpdate (see page 18257) is called.

Note: Make sure that EndUpdate (see page 18257) is called after all changes have been made to the nodes in the Items (see page 18271) array.

See Also

BadIndex (see page 18253)

EndUpdate (see page 18257)

Items (see page 18271)

121.1.3.4.7 TCustomOutline.Clear Method

Removes all items from the outline.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the outline of all its nodes. Clear frees the memory for the removed items but not any memory for associated data.

121.1.3.4.8 TCustomOutline.Click Method

Updates the SelectedItem (see page 18277) property.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Click is called automatically when the user clicks the mouse on a node in the outline. Click updates the SelectedItem (see page 18277) property before the OnClick event handler is called by the inherited method.

See Also

DblClick (see page 18254)

SelectedItem (see page 18277)

121.1.3.4.9 TCustomOutline.Collapse Method

Generates an OnCollapse (see page 18278) event.

Pascal

```
procedure Collapse(Index: LongInt); dynamic;
```

C++

```
__fastcall Collapse(LongInt Index);
```

Description

Collapse is called automatically after a node in the outline is collapsed. It calls the OnCollapse (see page 18278) event handler if it is assigned.

Override the Collapse method to provide additional processing after nodes in the outline have been collapsed, or to block the OnCollapse (see page 18278) event.

See Also

Expand (see page 18257)

OnCollapse (see page 18278)

121.1.3.4.10 TCustomOutline.DblClick Method

Expands or collapses the selected node.

Pascal

```
procedure DblClick; override;
```

C++

```
virtual __fastcall DblClick();
```

Description

DblClick is called automatically when the user double clicks the mouse on a node in the outline. After calling the OnDblClick event handler, DblClick updates the SelectedItem (see page 18277) property and expands the selected node if it is collapsed or collapses the selected node if it is expanded.

See Also

Click (see page 18254)

SelectedItem (see page 18277)

Expanded (see page 18361)

121.1.3.4.11 TCustomOutline.DefineProperties Method

Creates a binary property for customized bitmap values.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

The streaming system calls DefineProperties to allow the outline to define new properties that should be saved with the outline object. DefineProperties checks whether the outline uses any bitmaps other than the default values of PictureClosed (see page 18274), PictureLeaf (see page 18274), PictureMinus (see page 18275), PictureOpen (see page 18275), and PicturePlus (see page 18276). If the outline has customized any of these bitmaps, DefineProperties uses the DefineBinaryProperty method of the Filer parameter to create a binary property for the customized values.

See Also

PictureClosed (see page 18274)

PictureLeaf (see page 18274)

PictureMinus (see page 18275)

PictureOpen (see page 18275)

PicturePlus (see page 18276)

TWriter (see page 3175)

DefineBinaryProperty (see page 2837)

121.1.3.4.12 TCustomOutline.Delete Method

Deletes a node from the outline.

Pascal

```
procedure Delete(Index: LongInt);
```

C++

```
__fastcall Delete(LongInt Index);
```

Description

Call Delete to remove a node and all its descendants from the outline. Index is the index of the node to delete.

Delete removes the node and its children, frees all associated memory, and then calls DeleteNode (see page 18256) to update the indexes and make sure that SelectedItem (see page 18277) is valid.

See Also

Add (see page 18250)

DeleteNode (see page 18256)

SelectedItem (see page 18277)

121.1.3.4.13 TCustomOutline.DeleteNode Method

Fixes up the outline after a node has been deleted.

Pascal

```
procedure DeleteNode(Node: TOutlineNode; CurIndex: LongInt);
```

C++

```
__fastcall DeleteNode(TOutlineNode Node, LongInt CurIndex);
```

Description

DeleteNode is called automatically whenever a node in the outline is destroyed. It fixes up the index and makes sure that the SelectedItem (see page 18277) property is valid.

To Delete (see page 18255) a node from the outline, call the Delete (see page 18255) method of the outline or free the TOulineNode object.

See Also

Delete (see page 18255)

SelectedItem (see page 18277)

121.1.3.4.14 TCustomOutline.DrawCell Method

Draws a node in the outline.

Pascal

```
procedure DrawCell(ACol: Longint; ARow: Longint; ARect: TRect; AState: TGridDrawState);
override;
```

C++

```
virtual __fastcall DrawCell(Longint ACol, Longint ARow, TRect ARect, TGridDrawState AState);
```

Description

DrawCell is called automatically whenever the outline needs to draw a node. If the Style (see page 18277) property is otOwnerDraw and an OnDrawItem (see page 18278) event handler has been supplied, DrawCell generates an OnDrawItem (see page 18278) event. Otherwise, DrawCell draws the node in the manner indicated by the OutlineStyle (see page 18273) property.

See Also

OnDrawItem (see page 18278)

OutlineStyle (see page 18273)

121.1.3.4.15 TCustomOutline.EndUpdate Method

Performs the reindexing and repaints deferred by a call to BeginUpdate (see page 18253).

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

Call EndUpdate at the end of a segment of code that began with a call to BeginUpdate (see page 18253). Every time BeginUpdate (see page 18253) is called, an internal counter is incremented. When EndUpdate is called, the counter is decremented. When every call to BeginUpdate (see page 18253) has been matched by a call to EndUpdate, EndUpdate fixes up the indexes and repaints the outline.

See Also

BeginUpdate (see page 18253)

Items (see page 18271)

121.1.3.4.16 TCustomOutline.Expand Method

Generates an OnExpand (see page 18279) event.

Pascal

```
procedure Expand(Index: LongInt); dynamic;
```

C++

```
__fastcall Expand(LongInt Index);
```

Description

Expand is called automatically after a node in the outline is expanded. It calls the OnExpand (see page 18279) event handler if it is assigned.

Override the Expand method to provide additional processing after nodes in the outline have been expanded, or to block the OnExpand (see page 18279) event.

See Also

Collapse (see page 18254)

OnExpand (see page 18279)

121.1.3.4.17 TCustomOutline.FullCollapse Method

Causes all top-level nodes in the outline to collapse.

Pascal

```
procedure FullCollapse;
```

C++

```
__fastcall FullCollapse();
```

Description

Call FullCollapse to shrink the outline as much as possible. FullCollapse sets the Expanded property of each TOutlineNode (see page 18349) that is a direct descendant of the root of the outline to false. This causes all the descendants of these top-level nodes to be hidden. Depending on the value of OutlineStyle (see page 18273), each collapsed top-level node is then represented with the PictureClosed (see page 18274) bitmap and/or the PicturePlus (see page 18276) bitmap.

See Also

OutlineStyle (see page 18273)

PictureClosed (see page 18274)

PicturePlus (see page 18276)

Expanded (see page 18361)

121.1.3.4.18 TCustomOutline.FullExpand Method

Expands every node in the outline.

Pascal

```
procedure FullExpand;
```

C++

```
__fastcall FullExpand();
```

Description

Call FullExpand to show all nodes in the outline. FullExpand calls the FullExpand method of the root node. This causes the Expanded property of all nodes in the outline to be set to true. Every node in the Items (see page 18271) array appears fully expanded in the outline. Depending on the value of OutlineStyle (see page 18273), each node that has descendants is then represented with the PictureOpen (see page 18275) bitmap and/or the PictureMinus (see page 18275) bitmap. The appearance of leaf nodes is unaffected.

See Also

OutlineStyle (see page 18273)

PictureMinus (see page 18275)

PictureOpen (see page 18275)

FullExpand (see page 18355)

121.1.3.4.19 TCustomOutline.GetDataItem Method

Returns the index of the first node in the outline that has the Value pointer as its Data property.

Pascal

```
function GetDataItem(Value: Pointer): Longint;
```

C++

```
__fastcall Longint GetDataItem(void * Value);
```

Description

Call GetDataItem to locate the node that corresponds to the Value parameter. GetDataItem returns an index into the Items (see page 18271) property array.

See Also

Items (see page 18271)

Data (see page 18360)

121.1.3.4.20 TCustomOutline.GetItem Method

Returns the index of the node at the position indicated by the X and Y parameters.

Pascal

```
function GetItem(X: Integer; Y: Integer): LongInt;
```

C++

```
__fastcall LongInt GetItem(int X, int Y);
```

Description

Call GetItem to locate the node at the specified screen pixel coordinates. GetItem returns an index into the Items (see page 18271) property array.

See Also

Items (see page 18271)

121.1.3.4.21 TCustomOutline.GetNodeDisplayWidth Method

Returns the length of the row containing a specified node.

Pascal

```
function GetNodeDisplayWidth(Node: TOutlineNode): Integer;
```

C++

```
__fastcall int GetNodeDisplayWidth(TOutlineNode Node);
```

Description

Call GetNodeDisplayWidth to determine the length, in pixels, of the line in the outline that contains the node specified by the Node parameter. GetNodeDisplayWidth adjusts for the lines and bitmaps specified by the OutlineStyle (see page 18273) property.

See Also

ItemHeight (see page 18270)

OutlineStyle (see page 18273)

121.1.3.4.22 TCustomOutline.GetTextItem Method

Returns the index of the first node in the outline with the Value parameter as its Text property.

Pascal

```
function GetTextItem(const Value: string): Longint;
```

C++

```
__fastcall Longint GetTextItem(const AnsiString Value);
```

Description

Call GetTextItem to locate the node that corresponds to the Value parameter. GetTextItem returns an index into the Items (see

page 18271) property array.

See Also

Items (see page 18271)

Text (see page 18364)

121.1.3.4.23 TCustomOutline.GetVisibleNode Method

Returns the TOutlineNode (see page 18349) object for the node indicated by a position in the image of the outline.

Pascal

```
function GetVisibleNode(Index: LongInt): TOutlineNode;
```

C++

```
__fastcall TOutlineNode GetVisibleNode(LongInt Index);
```

Description

Call GetVisibleNode to locate a node with the position given by the Index parameter, where all nodes that are hidden because their parent node is collapsed are ignored. An Index of 0 indicates the first visible node, an Index of 1 indicates the second visible node, and so on. If the outline is fully expanded, GetVisibleNode returns the value of Items (see page 18271)[Index+1].

See Also

Items (see page 18271)

121.1.3.4.24 TCustomOutline.Insert Method

Inserts a node at a specified position.

Pascal

```
function Insert(Index: LongInt; const Text: string): LongInt;
```

C++

```
__fastcall LongInt Insert(LongInt Index, const AnsiString Text);
```

Description

Use Insert to add a TOutlineNode (see page 18349) object to the outline. Text specifies the Text property value of the new node.

Index specifies where to add the new node. The new node is positioned in the outline in the position currently occupied by the node identified by Index. The new node is inserted at the same level as the node currently identified by Index, so that the original node and the new node are siblings and share the same parent. To add nodes to the beginning of the outline, specify zero (0) as the Index parameter.

Nodes that appear after the new node, including the one specified by Index before Insert is called, are moved down one row and reindexed in the Items (see page 18271) property array with valid Index values. If BeginUpdate (see page 18253) has been called, the Items (see page 18271) array is not reindexed until the EndUpdate (see page 18257) method is called.

Insert returns the Index property of the new node.

See Also

Add (see page 18250)

AddChild (see page 18250)

BeginUpdate (see page 18253)

Delete (see page 18255)

EndUpdate (see page 18257)

InsertObject (see page 18261)

Items (see page 18271)

Index (see page 18362)

Text (see page 18364)

121.1.3.4.25 TCustomOutline.InsertObject Method

Inserts a node containing data at a specified position.

Pascal

```
function InsertObject(Index: LongInt; const Text: string; const Data: Pointer): LongInt;
```

C++

```
__fastcall LongInt InsertObject(LongInt Index, const AnsiString Text, const void * Data);
```

Description

Use InsertObject to add a TOutlineNode (see page 18349) object to the outline and assign it a data pointer. Text specifies the value of the Text property for the new node. Data specifies the value of the Data property for the new node.

Index specifies where to add the new node. The new node is positioned in the outline in the position currently occupied by the node identified by Index. The new node is inserted at the same level as the node currently identified by Index, so that the original node and the new node are siblings and share the same parent. To add nodes to the beginning of the outline, specify zero (0) as the Index parameter.

Nodes that appear after the new node, including the one specified by Index before Insert (see page 18260) is called, are moved down one row and reindexed in the Items (see page 18271) property array with valid Index values. If BeginUpdate (see page 18253) has been called, the Items (see page 18271) array is not reindexed until the EndUpdate (see page 18257) method is called.

Insert (see page 18260) returns the Index property of the new node.

See Also

AddObject (see page 18252)

AddChildObject (see page 18251)

BeginUpdate (see page 18253)

Delete (see page 18255)

EndUpdate (see page 18257)

Insert (see page 18260)

Items (see page 18271)

Data (see page 18360)

Index (see page 18362)

Text (see page 18364)

121.1.3.4.26 TCustomOutline.KeyDown Method

Provides special processing when the user presses a key.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

KeyDown is called automatically when the user presses a key. After calling the inherited method, which generates an OnKeyDown event, KeyDown processes the following keystrokes:

Key	Behavior
Home	Selects the first node in the outline.
End	Selects the last node in the outline.
Return	Toggles the selected node between its collapsed and expanded states.
Ctrl+*	Fully expands the outline.
Right arrow	Selects the first child of the selected node.
Left arrow	Selects the parent of the selected node.
Ctrl+Up arrow	Selects the node following the selected node.
Ctrl+Down arrow	Selects the node preceding the selected node.

See Also

KeyPress (see page 18262)

SelectedItem (see page 18277)

KeyDown (see page 14833)

121.1.3.4.27 TCustomOutline.KeyPress Method

Provides special processing when a key is pressed.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress is called automatically after the user presses a key but before it is released. After calling the inherited method, which generates an OnKeyPress event, KeyPress processes the following characters :

Key	Behavior
+	Expands the selected node.
-	Collapses the selected node.
*	Fully expands the outline.

See Also

KeyDown (see page 18262)

KeyPress (see page 14833)

121.1.3.4.28 TCustomOutline.Loaded Method

Sets up the outline after it has been streamed in.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

The streaming system calls Loaded after the outline has been fully loaded into memory. Loaded makes sure that all indexes are correct, loads any custom bitmaps that replaced the default values, and selects the first node in the outline.

See Also

DefineProperties (see page 18255)

SelectedItem (see page 18277)

SetGoodIndex (see page 18266)

121.1.3.4.29 TCustomOutline.LoadFromFile Method

Loads the outline from a text image stored in a file.

Pascal

```
procedure LoadFromFile(const FileName: string);
```

C++

```
__fastcall LoadFromFile(const AnsiString FileName);
```

Description

Call LoadFromFile to fill an outline from the file specified by the FileName parameter. Each line in the file represents a node in the outline. The file does not include the root node.

Leading tabs and spaces are converted into levels of the outline. Text without any leading tabs or spaces becomes level 1 items. For example, the following set of strings represents an outline with three first level children descended from the root, and an additional, second level child on the first and third nodes:

```
ChildA
 ChildA1
ChildB
ChildC
 ChildC1
```

To fill the outline from a text image read from a stream, use the LoadFromStream (see page 18264) method. To fill the outline from a text image represented as a TStrings object, use the Lines (see page 18272) property.

See Also

Lines (see page 18272)

LoadFromStream (see page 18264)

SaveToFile (see page 18265)

121.1.3.4.30 TCustomOutline.LoadFromStream Method

Loads the outline from a text image read from a stream.

Pascal

```
procedure LoadFromStream(Stream: TStream);
```

C++

```
__fastcall LoadFromStream(TStream Stream);
```

Description

Call LoadFromStream to fill an outline from the stream specified by the Stream parameter. For example, use LoadFromStream to load an outline from an image stored in a database table, using a TBlobStream object. Each node in the outline is separated by a newline character. The stream does not provide the root node.

Leading tabs and spaces are converted into levels of the outline. Text without any leading tabs or spaces becomes level 1 items. For example, the following set of strings represents an outline with three first level children descended from the root, and an additional, second level child on the first and third nodes:

```
ChildA
 ChildA1
ChildB
ChildC
 ChildC1
```

To fill the outline from a text image read from a file, use the LoadFromFile (see page 18263) method. To fill the outline from a text image represented as a TStrings object, use the Lines (see page 18272) property.

See Also

Lines (see page 18272)

LoadFromFile (see page 18263)

SaveToStream (see page 18265)

121.1.3.4.31 TCustomOutline.MouseDown Method

Updates the SelectedItem (see page 18277) property.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

After calling the inherited method to generate an OnMouseDown event and locate the line under the mouse, MouseDown sets the SelectedItem (see page 18277) property to the node in the line under the mouse.

See Also

SelectedItem (see page 18277)

121.1.3.4.32 TCustomOutline.Move Method

Repositions a node in the outline.

Pascal

```
procedure Move(Destination: LongInt; Source: LongInt; AttachMode: TAttachMode);
```

C++

```
__fastcall Move(LongInt Destination, LongInt Source, TAttachMode AttachMode);
```

Description

Call Move to reposition the node with the index specified by the Source parameter to the position indicated by the Destination parameter. The meaning of the Destination parameter depends on the value of the AttachMode parameter.

See Also

Add (see page 18250)

AddChild (see page 18250)

Insert (see page 18260)

Items (see page 18271)

121.1.3.4.33 TCustomOutline.SaveToFile Method

Saves the Lines (see page 18272) property to a file.

Pascal

```
procedure SaveToFile(const FileName: string);
```

C++

```
__fastcall SaveToFile(const AnsiString FileName);
```

Description

Call SaveToFile to save a text image of the outline to the file specified by the FileName parameter. The image saved is the value of the Lines (see page 18272) property, where each line of the file represents a node, and each level of the outline is represented by a tab character.

To write the outline to an arbitrary stream, use the SaveToStream (see page 18265) method.

See Also

Lines (see page 18272)

LoadFromFile (see page 18263)

SaveToStream (see page 18265)

121.1.3.4.34 TCustomOutline.SaveToStream Method

Writes the Lines (see page 18272) property to a stream.

Pascal

```
procedure SaveToStream(Stream: TStream);
```

C++

```
__fastcall SaveToStream(TStream Stream);
```

Description

Call SaveToStream to write a text image of the outline to the stream specified by the Stream parameter. The image saved is the value of the Lines (see page 18272) property, where each line of text represents a node, and each level of the outline is represented by a tab character.

To write the outline to a file, use the SaveToFile (see page 18265) method.

See Also

Lines (see page 18272)

LoadFromStream (see page 18264)

SaveToFile (see page 18265)

121.1.3.4.35 TCustomOutline.SetDisplayWidth Method

Sets the display width of the outline.

Pascal

```
procedure SetDisplayWidth(Value: Integer);
```

C++

```
__fastcall SetDisplayWidth(int Value);
```

Description

Call SetDisplayWidth to set the width of the image of the outline to the number of pixels specified by the Value parameter. Value should large enough to display the most deeply nested node. Use the GetNodeDisplayWidth (see page 18259) method to determine the size needed to display the visible nodes.

If the display width is greater than the client width of the control, users can scroll the image to view the entire width.

See Also

ClientWidth (see page 5811)

GetNodeDisplayWidth (see page 18259)

GetVisibleNode (see page 18260)

121.1.3.4.36 TCustomOutline.SetGoodIndex Method

Returns the first node in the Items (see page 18271) array that must be reindexed to make the index of the Value parameter correct.

Pascal

```
function SetGoodIndex(Value: TOutlineNode): TOutlineNode;
```

C++

```
__fastcall TOutlineNode SetGoodIndex(TOutlineNode Value);
```

Description

Call SetGoodIndex to determine how much of the Items (see page 18271) array must be reindexed after changing the nodes in the outline. All nodes from the root to the value returned by SetGoodIndex do not need to be reindexed.

See Also

BadIndex (see page 18253)

121.1.3.4.37 TCustomOutline.SetLevel Method

Sets the level of an outline node.

Pascal

```
procedure SetLevel(Node: TOutlineNode; CurLevel: Cardinal; NewLevel: Cardinal);
```

C++

```
__fastcall SetLevel(TOutlineNode Node, unsigned CurLevel, unsigned NewLevel);
```

Description

Call SetLevel to change the nesting depth of the node specified by the Node parameter. CurLevel is the node's current level and NewLevel is the node's new level.

For example, consider the outline specified by the following value of Lines (see page 18272):

```
NodeA
 NodeB
  NodeC
 NodeD
```

Calling SetLevel for NodeD with CurLevel set to 2 and NewLevel set to 4 results in the outline with the following value of Lines (see page 18272):

```
NodeA
 NodeB
  NodeC
   NodeD
```

That is, the parent of NodeD was changed from NodeA to NodeC.

See Also

Lines (see page 18272)

121.1.3.4.38 TCustomOutline.SetUpdateState Method

Enables or disables the maintenance of the node indexes.

Pascal

```
procedure SetUpdateState(Value: Boolean);
```

C++

```
__fastcall SetUpdateState(Boolean Value);
```

Description

Call SetUpdateState with Value set to true to block the recomputation of indexes. Call SetUpdateState with Value set to false to recompute all indexes in the outline and enable the maintenance of indexes with every change to the nodes in the outline.

Unlike calls to BeginUpdate (see page 18253) and EndUpdate (see page 18257), calls to SetUpdateState do not nest: Calling SetUpdateState with Value set to false always recomputes the indexes regardless of the number of previous calls to SetUpdateState with Value set to true.

Note: Calling SetUpdateState takes precedence over any enabling or disabling of index maintenance provided by BeginUpdate (see page 18253) or EndUpdate (see page 18257).

See Also

BeginUpdate (see page 18253)

EndUpdate (see page 18257)

121.1.3.5 TCustomOutline Properties

The properties of the TCustomOutline class are listed here.

Protected Properties

	Name	Description
	ItemHeight (see page 18270)	Specifies the height of each node in the outline.
	ItemSeparator (see page 18271)	Specifies a delimiting string for outline nodes.
	Lines (see page 18272)	Lists the Text property values of all nodes in the outline.
	Options (see page 18272)	Determines how the nodes in the outline are drawn.
	OutlineStyle (see page 18273)	Specifies how the outline tree is displayed.
	PictureClosed (see page 18274)	Specifies the bitmap used to represent a collapsed node that has subitems.
	PictureLeaf (see page 18274)	Specifies the bitmap used to represent an item that does not have subitems.
	PictureMinus (see page 18275)	Specifies the bitmap used to represent an expanded item that has subitems.
	PictureOpen (see page 18275)	Specifies the bitmap used to represent an expanded item that has subitems.
	PicturePlus (see page 18276)	Specifies the bitmap used to represent a collapsed item that has subitems.
	Style (see page 18277)	Determines whether the nodes and tree are drawn automatically.

Public Properties

	Name	Description
	Canvas (see page 18268)	Specifies the TCanvas object that presents a drawing surface for the control.
R	ItemCount (see page 18270)	Indicates the number of nodes in the outline.
R	Items (see page 18271)	Lists the nodes in the outline.
	Row (see page 18276)	Specifies the index of the row that contains the selected cell.
	SelectedItem (see page 18277)	Specifies the index of the selected node.

121.1.3.5.1 TCustomOutline.Canvas Property

Specifies the TCanvas object that presents a drawing surface for the control.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control. Canvas encapsulates a Windows device context, providing all the tools and methods needed for drawing and painting.

Canvas is a protected property that is usually redeclared as public in descendants of TCustomControl.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
```

```
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas,Rect.Left,Rect.Top,index);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(index) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TStringGrid *myStringGrid;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TStringGrid")
  {
    myStringGrid = (TStringGrid *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myStringGrid->Canvas->MoveTo(myStringGrid->ClientRect.Left, myStringGrid->ClientRect.Top);
    myStringGrid->Canvas->LineTo(myStringGrid->ClientRect.Right,
myStringGrid->ClientRect.Bottom);
    myStringGrid = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
```

```
    ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  end;

  procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  begin
    CanSelect:= True;
  end;
  {
  The following code uses ClientRect to find and draw a line
  from the top left to the bottom right of the current
  control.  Select a TControl in the ListBox list.  This
  example assumes that the current object is a descendent of
  TControl and has a public Canvas property.
  }
    if (ListBox1.Items.Objects[Index] is TStringGrid) then
    begin
      myStringGrid:= (ListBox1.Items.Objects[Index] as TStringGrid);
      with myStringGrid.ClientRect do
      begin // make sure we are using the TControl's canvas and not the form's!
        myStringGrid.Canvas.MoveTo(Left, Top);
        myStringGrid.Canvas.LineTo(Right, Bottom);
      end;
    myStringGrid:= nil;
    end;
```

See Also

Paint (see page 5920)

121.1.3.5.2 TCustomOutline.ItemCount Property

Indicates the number of nodes in the outline.

Pascal

```
property ItemCount: LongInt;
```

C++

```
__property LongInt ItemCount;
```

Description

Read ItemCount to determine the number of nodes in the outline. Use ItemCount with the Items (see page 18271) property array to iterate over all nodes in the outline.

Note: Reading ItemCount can be computationally expensive, as it causes all the indexes used by the outline to be updated.

See Also

Items (see page 18271)

121.1.3.5.3 TCustomOutline.ItemHeight Property

Specifies the height of each node in the outline.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Set ItemHeight to the height, in pixels, of each node in the outline. Changing the ItemHeight property also changes the font used to display the nodes.

See Also

ItemCount (see page 18270)

Items (see page 18271)

121.1.3.5.4 TCustomOutline.Items Property

Lists the nodes in the outline.

Pascal

```
property Items [Index: LongInt]: TOutlineNode;
```

C++

```
__property TOutlineNode Items[LongInt Index];
```

Description

Use Items to access a node in the outline. Each item is a TOutlineNode (see page 18349) object.

Index is a number specifying the node to return, where Items[0] refers to the root, Items[1] is the first (topmost) node, and so on.

Note: Items is the default property of TCustomOutline (see page 18234). This means that the name of the property may be omitted in code. Thus the line

Note: Outline1.Items[1].Text := Edit1.Text;

Note: can also be written

Note: Outline1[1].Text := Edit1.Text;

Note: In C++, Items can be accessed using the [] operator, to achieve an affect similar to the default property in Delphi.

See Also

ItemCount (see page 18270)

ItemHeight (see page 18270)

ItemSeparator (see page 18271)

Lines (see page 18272)

operator_[]

121.1.3.5.5 TCustomOutline.ItemSeparator Property

Specifies a delimiting string for outline nodes.

Pascal

```
property ItemSeparator: string;
```

C++

```
__property AnsiString ItemSeparator;
```

Description

Set ItemSeparator to the string that should appear between node names in the path description of any individual node in the outline. ItemSeparator is used to implement the FullPath read method of TOutlineNode (see page 18349).

ItemSeparator is initialized to the backslash (\) character when the TCustomOutline (see page 18234) object is created.

See Also

Items (see page 18271)

FullPath (see page 18361)

121.1.3.5.6 TCustomOutline.Lines Property

Lists the Text property values of all nodes in the outline.

Pascal

```
property Lines: TStrings;
```

C++

```
__property TStrings Lines;
```

Description

Read Lines to obtain a text representation of the outline. Set Lines to fill the outline from a set of text strings. Each string in the Lines property represents a node in the outline. Lines does not include the root node.

Leading tabs and spaces are converted into levels of the outline. Text without any leading tabs or spaces becomes level 1 items. For example, the following set of strings represents an outline with three first level children descended from the root, and an additional, second level child on the first and third nodes:

```
ChildA
 ChildA1
ChildB
ChildC
 ChildC1
```

The Lines property is primarily useful for filling an outline with items at design time. While Lines can be used to access the individual nodes at run time, it is much quicker to access an item with the Items (see page 18271) property.

See Also

Items (see page 18271)

Text (see page 18364)

121.1.3.5.7 TCustomOutline.Options Property

Determines how the nodes in the outline are drawn.

Pascal

```
property Options: TOutlineOptions;
```

C++

```
__property TOutlineOptions Options;
```

Description

Set Options to any combination of the following values:

Value	Meaning
ooDrawTreeRoot	The first item (Index value of 1) is connected to the root item by the outline tree. This means that the tree will extend from the top of the outline to all the first level items. Without ooDrawTreeRoot, all first level items appear leftmost in the outline, not connected by the tree.
ooDrawFocusRect	The outline draws a focus rectangle around the selected item.
ooStretchBitmaps	The outline stretches the standard bitmaps (PictureLeaf (see page 18274), PictureOpen (see page 18275), PictureClosed (see page 18274), PicturePlus (see page 18276), PictureMinus (see page 18275)) to fit in the size of the item, determined by the size of the Font of the Text. Without ooStretchBitmaps, the bitmaps will be cropped if larger than the height of the item text, or won't fill up the entire item space if smaller than the text.

See Also

Items (see page 18271)

OutlineStyle (see page 18273)

PictureClosed (see page 18274)

PictureLeaf (see page 18274)

PictureMinus (see page 18275)

PictureOpen (see page 18275)

PicturePlus (see page 18276)

SelectedItem (see page 18277)

121.1.3.5.8 TCustomOutline.OutlineStyle Property

Specifies how the outline tree is displayed.

Pascal

```
property OutlineStyle: TOutlineStyle;
```

C++

```
__property TOutlineStyle OutlineStyle;
```

Description

Set OutlineStyle to one of the following values:

Value	Meaning
osText	Displays node text (specified in Text) only.
osPlusMinusText	Displays plus picture (specified in PicturePlus (see page 18276)), minus picture (specified in PictureMinus (see page 18275)), and node text (specified in Text).
osPictureText	Displays open picture (specified in PictureOpen (see page 18275)), closed picture (specified in PictureClosed (see page 18274)), leaf picture (specified in PictureLeaf (see page 18274)) and node text (specified in Text).

osPlusMinusPictureText	Displays plus picture (specified in PicturePlus (see page 18276)), minus picture (specified in PictureMinus (see page 18275)), open picture (specified in PictureOpen (see page 18275)), closed picture (specified in PictureClosed (see page 18274)), leaf picture (specified in PictureLeaf (see page 18274)), and node text (specified in Text).
osTreeText	Displays lines connecting outline nodes and node text (specified in Text).
osTreePictureText	Displays lines connecting outline nodes, open picture (specified in PictureOpen (see page 18275)), closed picture (specified in PictureClosed (see page 18274)), leaf picture (specified in PictureLeaf (see page 18274)), and node text (specified in Text).

Note: If the Style (see page 18277) property is otOwnerDraw, the value of OutlineStyle is ignored. The connecting lines, node bitmaps, and node text must all be drawn in an OnDrawItem (see page 18278) event handler.

See Also

Options (see page 18272)

PictureClosed (see page 18274)

PictureLeaf (see page 18274)

PictureMinus (see page 18275)

PictureOpen (see page 18275)

PicturePlus (see page 18276)

Style (see page 18277)

Text (see page 18364)

121.1.3.5.9 TCustomOutline.PictureClosed Property

Specifies the bitmap used to represent a collapsed node that has subitems.

Pascal

```
property PictureClosed: TBitmap;
```

C++

```
__property TBitmap PictureClosed;
```

Description

By default, PictureClosed is a closed file folder bitmap. Set PictureClosed to change the bitmap used for collapsed nodes.

Note: The bitmap is not displayed in the outline unless the OutlineStyle (see page 18273) property is set to osPictureText, osPlusMinusPictureText, or osTreePictureText.

See Also

OutlineStyle (see page 18273)

PictureLeaf (see page 18274)

PictureMinus (see page 18275)

PictureOpen (see page 18275)

121.1.3.5.10 TCustomOutline.PictureLeaf Property

Specifies the bitmap used to represent an item that does not have subitems.

Pascal

```
property PictureLeaf: TBitmap;
```

C++

```
__property TBitmap PictureLeaf;
```

Description

By default, PictureLeaf is a document bitmap. Set PictureLeaf to change the bitmap used for leaf nodes.

Note: The bitmap is not displayed in the outline unless the OutlineStyle (see page 18273) property is set to osPictureText, osPlusMinusPictureText, or osTreePictureText.

See Also

OutlineStyle (see page 18273)

PictureClosed (see page 18274)

PictureOpen (see page 18275)

121.1.3.5.11 TCustomOutline.PictureMinus Property

Specifies the bitmap used to represent an expanded item that has subitems.

Pascal

```
property PictureMinus: TBitmap;
```

C++

```
__property TBitmap PictureMinus;
```

Description

By default, PictureMinus is a minus sign (-). Set PictureMinus to change the bitmap used for collapsible nodes.

Note: The bitmap is not displayed in the outline unless the OutlineStyle (see page 18273) property is set to osPlusMinusPictureText or osPlusMinusText.

See Also

OutlineStyle (see page 18273)

PictureOpen (see page 18275)

PicturePlus (see page 18276)

121.1.3.5.12 TCustomOutline.PictureOpen Property

Specifies the bitmap used to represent an expanded item that has subitems.

Pascal

```
property PictureOpen: TBitmap;
```

C++

```
__property TBitmap PictureOpen;
```

Description

By default, the PictureOpen property is an open file folder. Set PictureOpen to change the bitmap used for expanded nodes.

Note: The bitmap is not displayed in the outline unless the OutlineStyle (see page 18273) property is set to osPictureText,

osPlusMinusPictureText, or osTreePictureText.

See Also

OutlineStyle (see page 18273)

PictureClosed (see page 18274)

PictureLeaf (see page 18274)

PicturePlus (see page 18276)

121.1.3.5.13 TCustomOutline.PicturePlus Property

Specifies the bitmap used to represent a collapsed item that has subitems.

Pascal

```
property PicturePlus: TBitmap;
```

C++

```
__property TBitmap PicturePlus;
```

Description

By default, the PicturePlus property is a plus sign (+). Set PicturePlus to change the bitmap used for collapsed nodes.

Note: The bitmap will not be displayed in the outline unless the OutlineStyle (see page 18273) property is set to osPlusMinusPictureText or osPlusMinusText.

See Also

OutlineStyle (see page 18273)

PictureClosed (see page 18274)

PictureMinus (see page 18275)

121.1.3.5.14 TCustomOutline.Row Property

Specifies the index of the row that contains the selected cell.

Pascal

```
property Row: Longint;
```

C++

```
__property Longint Row;
```

Description

Use Row at runtime to determine the current row in the grid. Setting Row moves focus to the cell in the current column that is in the new row. The first row has an index of 0, the second row an index of 1, and so on.

The selected cell in the grid can be located by reading the Row property and the Col property to obtain the indexes of its row and column. When focus moves to the grid, the selected cell gets input focus.

Delphi Examples:

```
{
This example displays a string grid, a list box, and a
button on a form. When the application runs, strings are put
in the cells of row 1 of the string grid. When the user
clicks the button, The strings in the current row of the
grid are displayed in the list box.
```

```
  Note that when the grid contains an empty cell, the list box
  has a corresponding empty string.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    ListBox1.Items := StringGrid1.Rows[StringGrid1.Row];
  end;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with StringGrid1 do
    begin
      Cells[1,0] := 'Column 1';
      Cells[2,0] := 'Column 2';
      Cells[3,0] := 'Column 3';
      Cells[4,0] := 'Column 4';
      Cells[0,1] := 'Row 1';
      Cells[1,1] := 'Object';
      Cells[2,1] := 'Pascal';
      Cells[3,1] := 'is';
      Cells[4,1] := 'excellent';
      Cells[0,2] := 'Row 2';
      Cells[1,2] := 'Delphi';
      Cells[2,2] := 'is';
      Cells[4,2] := 'RAD';
    end;
  end;
```

See Also

Col (see page 14843)

121.1.3.5.15 TCustomOutline.SelectedItem Property

Specifies the index of the selected node.

Pascal

```
property SelectedItem: Longint;
```

C++

```
__property Longint SelectedItem;
```

Description

Read SelectedItem to determine the Index of the selected node. If no node is selected, SelectedItem is 0. Set SelectedItem to set focus to a particular node in the outline.

See Also

Items (see page 18271)

121.1.3.5.16 TCustomOutline.Style Property

Determines whether the nodes and tree are drawn automatically.

Pascal

```
property Style: TOutlineType;
```

C++

```
__property TOutlineType Style;
```

Description

By default, Style is osStandard, meaning that the outline displays items in the style determined by the OutlineStyle (see page

18273) property. Set Style to otOwnerDraw to create owner-draw outlines, meaning that items must be drawn on the canvas of the outline by code in an OnDrawItem (see page 18278) event handler.

Owner-draw outlines can display items other than the Text of a node and the standard bitmaps specified in the PictureClosed (see page 18274), PictureOpen (see page 18275), PictureMinus (see page 18275), PicturePlus (see page 18276), and PictureLeaf (see page 18274) properties. Owner-draw outlines require more programming, however, as the application must specify how to render the image for each node in an OnDrawItem (see page 18278) event handler.

See Also

Canvas (see page 5921)

OnDrawItem (see page 18278)

OutlineStyle (see page 18273)

121.1.3.6 TCustomOutline Events

The events of the TCustomOutline class are listed here.

Protected Events

	Name	Description
	OnCollapse (see page 18278)	Occurs immediately after a node in the outline is collapsed.
	OnDrawItem (see page 18278)	Occurs when a node must be drawn in an owner-draw outline.
	OnExpand (see page 18279)	Occurs immediately after a node in the outline is expanded.

121.1.3.6.1 TCustomOutline.OnCollapse Event

Occurs immediately after a node in the outline is collapsed.

Pascal

```
property OnCollapse: EOutlineChange;
```

C++

```
__property EOutlineChange OnCollapse;
```

Description

Write an OnCollapse event handler to perform special processing when a node in the outline is collapsed. The event handler supplies the index of the node that collapsed.

See Also

OnExpand (see page 18279)

121.1.3.6.2 TCustomOutline.OnDrawItem Event

Occurs when a node must be drawn in an owner-draw outline.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

When the Style (see page 18277) property is set to otOwnerDraw, specify an OnDrawItem event handler to render the image

of each node in the outline. An OnDrawItem event handler can display items other than the Text of a node and the standard bitmaps specified in the PictureClosed (see page 18274), PictureOpen (see page 18275), PictureMinus (see page 18275), PicturePlus (see page 18276), and PictureLeaf (see page 18274) properties.

Draw the image of the node specified by the Index parameter onto the canvas of the outline.

See Also

Canvas (see page 5921)

PictureClosed (see page 18274)

PictureLeaf (see page 18274)

PictureMinus (see page 18275)

PictureOpen (see page 18275)

PicturePlus (see page 18276)

Style (see page 18277)

Text (see page 18364)

121.1.3.6.3 TCustomOutline.OnExpand Event

Occurs immediately after a node in the outline is expanded.

Pascal

```
property OnExpand: EOutlineChange;
```

C++

```
__property EOutlineChange OnExpand;
```

Description

Write an OnExpand event handler to perform special processing when a node in the outline is expanded. The event handler identifies the node that was just expanded.

See Also

OnCollapse (see page 18278)

121.1.4 TOutline Class

TOutline is a multilevel outline that displays the values of TOutlineNode (see page 18349) objects.

Class Hierarchy

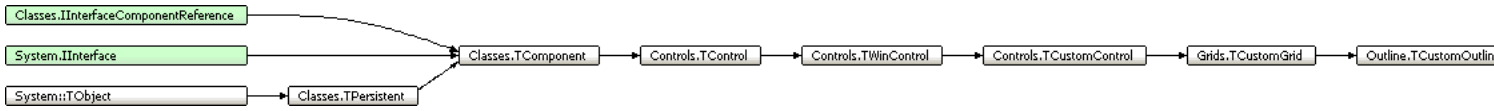

Pascal

```
TOutline = class(TCustomOutline);
```

C++

```
class TOutline : public TCustomOutline;
```

File

Outline

Description

Use an outline to visually organize information in a hierarchical tree. Each item in an outline is contained in a TOutlineNode (see page 18349) object. Outline (see page 18231) controls allow an application to manage a hierarchy of outline nodes, including the ability to control the appearance and layout of those nodes and the ability to expand or collapse branches of the outline tree.

TOutline is provided primarily for backward compatibility. New applications should use TTreeView instead.

See Also

TOutlineNode (see page 18349)

TTreeView (see page 5293)

121.1.4.1 TOutline Members

The following tables list the members exposed by TOutline.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 14814)	This is DesignOptionsBoost, a member of class TCustomGrid.
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FGridState (see page 14814)	This is FGridState, a member of class TCustomGrid.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveCellExtents (see page 14814)	This is FSaveCellExtents, a member of class TCustomGrid.

	VirtualView (see page 14814)	This is VirtualView, a member of class TCustomGrid.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AdjustSize (see page 14816)	Adds or deletes rows or columns from the grid.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BadIndex (see page 18253)	Indicates whether a node's index is up to date.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	BeginColumnDrag (see page 14817)	Starts the dragging of a column in the grid.
	BeginRowDrag (see page 14817)	Starts the dragging of a row in the grid.
	BoxRect (see page 14818)	Returns the screen coordinates of a specified set of cells.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcDrawInfo (see page 14818)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 14818)	Calculates the pixel locations of the fixed cells.
V	CalcSizingState (see page 14819)	Determines the appearance of the mouse cursor during resizing.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanEditAcceptKey (see page 14819)	Indicates whether the Key parameter can be entered into the current cell.
	CanEditModify (see page 14819)	Indicates whether the current cell in the grid can be edited.
V	CanEditShow (see page 14820)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 14820)	Indicates whether the grid should respond to a given key combination.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	CellRect (see page 14821)	Returns the screen coordinates of a cell in the grid.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeGridOrientation (see page 14821)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CheckColumnDrag (see page 14821)	CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 14822)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
	Click (see page 5743)	Respond to user click.
V	Click (see page 18254)	Updates the SelectedItem (see page 18277) property.
	Collapse (see page 18254)	Generates an OnCollapse (see page 18278) event.
	ColumnMoved (see page 14822)	Provides the interface for a method that responds when the position of a column changes.

	ColWidthsChanged (see page 14822)	Responds when the column widths change.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateEditor (see page 14823)	Creates the inplace editor control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 14823)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
V	DblClick (see page 18254)	Expands or collapses the selected node.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
V	DefineProperties (see page 14824)	Adds the ColWidths (see page 14845) and RowHeights (see page 14855) properties to the values saved with the grid.
V	DefineProperties (see page 18255)	Creates a binary property for customized bitmap values.
V	DeleteColumn (see page 14824)	Deletes a specified column from the grid.
	DeleteNode (see page 18256)	Fixes up the outline after a node has been deleted.
V	DeleteRow (see page 14825)	Deletes a specified from the grid.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
V	DoExit (see page 14825)	Hides the inplace editor when selection moves to another control in a form.
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.

	DoMouseWheelDown (see page 14825)	Performs tasks when the mouse wheel is rotated downward.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoMouseWheelUp (see page 14826)	Performs tasks when the mouse wheel is rotated upward.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawCell (see page 14826)	Provides the prototype for a method that draws the contents of a cell in the grid.
	DrawCell (see page 18256)	Draws a node in the outline.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EndColumnDrag (see page 14827)	Ends the dragging of a column in the grid.
	EndRowDrag (see page 14827)	Ends the dragging of a row in the grid.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	Expand (see page 18257)	Generates an OnExpand (see page 18279) event.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	FocusCell (see page 14827)	Changes the current cell to a specified cell.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEditLimit (see page 14828)	Provides the interface for a method that limits the number of characters the user can type into cells.
	GetEditMask (see page 14828)	Provides the interface for a method that returns an input mask for the specified cell.
	GetEditStyle (see page 14828)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
	GetEditText (see page 14829)	Returns the value of the indicated cell formatted for editing.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetGridHeight (see page 14829)	Returns the value of the GridHeight (see page 14850) property.
	GetGridWidth (see page 14830)	Returns the value of the GridWidth (see page 14851) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.

	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HideEdit (see page 14830)	Hides the inplace editor.
	HideEditor (see page 14830)	Hides the inplace editor when it is not in use.
	InvalidateCell (see page 14831)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCol (see page 14831)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	InvalidateEditor (see page 14832)	Causes the inplace editor to repaint itself, moving to a new location if necessary.
	InvalidateGrid (see page 14832)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 14832)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
V	KeyDown (see page 14833)	Provides special processing when the user presses a key.
V	KeyDown (see page 18262)	Provides special processing when the user presses a key.
	KeyPress (see page 6163)	Respond to keyboard input.
V	KeyPress (see page 14833)	Provides special processing when the user presses a key.
V	KeyPress (see page 18262)	Provides special processing when a key is pressed.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 18263)	Sets up the outline after it has been streamed in.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
V	MouseDown (see page 14834)	Provides special processing when the user presses a mouse button.
V	MouseDown (see page 18264)	Updates the SelectedItem (see page 18277) property.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
V	MouseMove (see page 14834)	Provides special processing when the user moves the mouse.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	MouseUp (see page 14835)	Provides special processing when the user releases the mouse button.
	Move (see page 18265)	Repositions a node in the outline.
	MoveColRow (see page 14835)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 14836)	Moves a column to a new position.
	MoveRow (see page 14836)	Moves a row to a new position.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 5920)	Renders the image of a custom control.
V	Paint (see page 14836)	Draws the image of the grid control on the screen.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.

	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	RowHeightsChanged (see page 14837)	Responds when the row heights change.
	RowMoved (see page 14837)	Provides the interface for a method that responds when the position of a row changes.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	ScrollData (see page 14838)	Scrolls the visual image of the cells in the grid.
	SelectCell (see page 14838)	Determines whether a particular cell in the grid can be selected.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayWidth (see page 18266)	Sets the display width of the outline.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEditText (see page 14839)	Provides the interface for a method that updates the text associated with a cell.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetGoodIndex (see page 18266)	Returns the first node in the Items (see page 18271) array that must be reindexed to make the index of the Value parameter correct.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetLevel (see page 18267)	Sets the level of an outline node.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.

V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	ShowEditor (see page 14839)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 14839)	Displays the inplace editor, with its text set to the specified character.
	SizeChanged (see page 14840)	Responds to changes in the size of the grid.
	Sizing (see page 14840)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	TimedScroll (see page 14840)	Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.
	TopLeftChanged (see page 14841)	Updates the position of the inplace editor when the grid scrolls.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateDesigner (see page 14841)	Notifies the designer when rows and columns of the grid are moved or resized.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Col (see page 14843)	Specifies the index of the column that contains the selected cell.
	ColCount (see page 14844)	Specifies the number of columns in the grid.
	ColWidths (see page 14845)	Indicates the width (in pixels) of all the columns in the grid.

	DefaultColWidth (see page 14846)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 14847)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 14847)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	EditorMode (see page 14848)	Determines whether the current cell can be edited.
	FixedColor (see page 14848)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 14848)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	FixedRows (see page 14849)	Specifies the number of rows on the top of the grid that cannot be scrolled.
R	GridHeight (see page 14850)	Specifies the height of the grid in pixels.
	GridLineWidth (see page 14850)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
R	GridWidth (see page 14851)	Specifies the width of the grid in pixels.
R	HitTest (see page 14851)	Indicates the current position of the mouse cursor.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
R	InplaceEditor (see page 14852)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	LeftCol (see page 14852)	Specifies the index of the first visible scrollable column in the grid.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	Row (see page 14854)	Specifies the index of the row that contains the selected cell.
	RowCount (see page 14854)	Specifies the number of rows in the grid.
	RowHeights (see page 14855)	Gives the height (in pixels) of all rows in the grid.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Selection (see page 14856)	Indicates the boundaries of the current selection.
	TabStops (see page 14858)	Indicates whether the user can tab to specified columns in the grid.
	Text (see page 5842)	Contains a text string associated with the control.
	TopRow (see page 14858)	Specifies the index of the first visible scrollable row in the grid.
R	VisibleColCount (see page 14859)	Indicates the number of scrollable columns visible in the grid.
R	VisibleRowCount (see page 14859)	Indicates the number of scrollable rows visible in the grid.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Add (see page 18250)	Adds a node to the outline.
	AddChild (see page 18250)	Adds a child node to a node in the outline.
	AddChildObject (see page 18251)	Adds a child node containing data to a node in the outline.
	AddObject (see page 18252)	Adds a node containing data to the outline.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.

	BeginUpdate (see page 18253)	Prevents the updating of the outline until the EndUpdate (see page 18257) method is called.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 18253)	Removes all items from the outline.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 5918)	Creates an instance of TCustomControl (see page 5908).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 14813)	Creates an instance of TCustomGrid (see page 14800).
V	Create (see page 18248)	Creates an instance of TCustomOutline (see page 18234).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 18255)	Deletes a node from the outline.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 14813)	Destroys an instance of TCustomGrid (see page 14800).
V	Destroy (see page 18248)	Destroys an instance of TCustomOutline (see page 18234).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.

	EndUpdate (see page 18257)	Performs the reindexing and repaints deferred by a call to BeginUpdate (see page 18253).
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	FullCollapse (see page 18257)	Causes all top-level nodes in the outline to collapse.
	FullExpand (see page 18258)	Expands every node in the outline.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetDataItem (see page 18258)	Returns the index of the first node in the outline that has the Value pointer as its Data property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItem (see page 18259)	Returns the index of the node at the position indicated by the X and Y parameters.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNodeDisplayWidth (see page 18259)	Returns the length of the row containing a specified node.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextItem (see page 18259)	Returns the index of the first node in the outline with the Value parameter as its Text property.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	GetVisibleNode (see page 18260)	Returns the TOutlineNode (see page 18349) object for the node indicated by a position in the image of the outline.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 18260)	Inserts a node at a specified position.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InsertObject (see page 18261)	Inserts a node containing data at a specified position.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.

	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	LoadFromFile (see page 18263)	Loads the outline from a text image stored in a file.
	LoadFromStream (see page 18264)	Loads the outline from a text image read from a stream.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseCoord (see page 14834)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 18265)	Saves the Lines (see page 18272) property to a file.
	SaveToStream (see page 18265)	Writes the Lines (see page 18272) property to a stream.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	SetUpdateState (see page 18267)	Enables or disables the maintenance of the node indexes.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	Canvas (see page 18268)	Specifies the TCanvas object that presents a drawing surface for the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
R	ItemCount (see page 18270)	Indicates the number of nodes in the outline.
R	Items (see page 18271)	Lists the nodes in the outline.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.

	Row (see page 18276)	Specifies the index of the row that contains the selected cell.
	SelectedItem (see page 18277)	Specifies the index of the selected node.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 18320)	Occurs when the user clicks the control.
	OnCollapse (see page 18322)	Occurs immediately after a node in the outline is collapsed.
	OnContextPopup (see page 18322)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 18323)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 18323)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 18325)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 18328)	Occurs when a node must be drawn in an owner-draw outline.
	OnEndDock (see page 18328)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 18329)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 18330)	Occurs when a control receives the input focus.
	OnExit (see page 18332)	Occurs when the input focus shifts away from one control to another.
	OnExpand (see page 18334)	Occurs immediately after a node in the outline is expanded.
	OnKeyDown (see page 18334)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 18337)	Occurs when a key is pressed.
	OnKeyUp (see page 18338)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 18339)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 18340)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 18343)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 18343)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 18343)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 18346)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 18348)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 18348)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 18294)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	BorderStyle (see page 18298)	Determines whether a single line border is drawn around the grid.
	Color (see page 18298)	Specifies the background color of the control.
	Ctl3D (see page 18299)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 18299)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 18299)	Specifies whether the control is being dragged normally or for docking.

	DragMode (see page 18300)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 18301)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 18302)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ItemHeight (see page 18303)	Specifies the height of each node in the outline.
	ItemSeparator (see page 18303)	Specifies a delimiting string for outline nodes.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Lines (see page 18304)	Lists the Text property values of all nodes in the outline.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 18304)	Determines how the nodes in the outline are drawn.
	OutlineStyle (see page 18305)	Specifies how the outline tree is displayed.
	ParentColor (see page 18306)	Determines where a control looks for its color information.
	ParentCtl3D (see page 18306)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 18307)	Determines where a control looks for its font information.
	ParentShowHint (see page 18307)	Determines where a control looks to find out if its Help Hint should be shown.
	PictureClosed (see page 18308)	Specifies the bitmap used to represent a collapsed node that has subitems.
	PictureLeaf (see page 18308)	Specifies the bitmap used to represent an item that does not have subitems.
	PictureMinus (see page 18308)	Specifies the bitmap used to represent an expanded item that has subitems.
	PictureOpen (see page 18309)	Specifies the bitmap used to represent an expanded item that has subitems.
	PicturePlus (see page 18309)	Specifies the bitmap used to represent a collapsed item that has subitems.
	PopupMenu (see page 18310)	Identifies the pop-up menu associated with the control.
	ScrollBars (see page 18314)	Specifies whether the grid includes horizontal and vertical scroll bars.
	ShowHint (see page 18315)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 18316)	Determines whether the nodes and tree are drawn automatically.
	TabOrder (see page 18317)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 18317)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 18318)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

121.1.4.2 TOutline Properties

The properties of the TOutline class are listed here.

Published Properties

	Name	Description
	Align (see page 18294)	Determines how the control aligns within its container (parent control).
	BorderStyle (see page 18298)	Determines whether a single line border is drawn around the grid.
	Color (see page 18298)	Specifies the background color of the control.
	Ctl3D (see page 18299)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 18299)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 18299)	Specifies whether the control is being dragged normally or for docking.

	DragMode (see page 18300)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 18301)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 18302)	Controls the attributes of text written on or in the control.
	ItemHeight (see page 18303)	Specifies the height of each node in the outline.
	ItemSeparator (see page 18303)	Specifies a delimiting string for outline nodes.
	Lines (see page 18304)	Lists the Text property values of all nodes in the outline.
	Options (see page 18304)	Determines how the nodes in the outline are drawn.
	OutlineStyle (see page 18305)	Specifies how the outline tree is displayed.
	ParentColor (see page 18306)	Determines where a control looks for its color information.
	ParentCtl3D (see page 18306)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 18307)	Determines where a control looks for its font information.
	ParentShowHint (see page 18307)	Determines where a control looks to find out if its Help Hint should be shown.
	PictureClosed (see page 18308)	Specifies the bitmap used to represent a collapsed node that has subitems.
	PictureLeaf (see page 18308)	Specifies the bitmap used to represent an item that does not have subitems.
	PictureMinus (see page 18308)	Specifies the bitmap used to represent an expanded item that has subitems.
	PictureOpen (see page 18309)	Specifies the bitmap used to represent an expanded item that has subitems.
	PicturePlus (see page 18309)	Specifies the bitmap used to represent a collapsed item that has subitems.
	PopupMenu (see page 18310)	Identifies the pop-up menu associated with the control.
	ScrollBars (see page 18314)	Specifies whether the grid includes horizontal and vertical scroll bars.
	ShowHint (see page 18315)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 18316)	Determines whether the nodes and tree are drawn automatically.
	TabOrder (see page 18317)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 18317)	Determines if the user can tab to a control.
	Visible (see page 18318)	Determines whether the component appears onscreen.

121.1.4.2.1 TOutline.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
```

```
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
```

```
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

121.1.4.2.2 TOutline.BorderStyle Property

Determines whether a single line border is drawn around the grid.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Set BorderStyle to bsSingle to add a single line border around the grid's image. Set BorderStyle to bsNone to omit the border.

121.1.4.2.3 TOutline.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
```

```
end;
```

See Also

ParentColor (see page 5834)

121.1.4.2.4 TOutline.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

121.1.4.2.5 TOutline.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

121.1.4.2.6 TOutline.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

121.1.4.2.7 TOutline.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
```

```
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

121.1.4.2.8 TOutline.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

121.1.4.2.9 TOutline.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

121.1.4.2.10 TOutline.ItemHeight Property

Specifies the height of each node in the outline.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Set ItemHeight to the height, in pixels, of each node in the outline. Changing the ItemHeight property also changes the font used to display the nodes.

See Also

ItemCount (see page 18270)

Items (see page 18271)

121.1.4.2.11 TOutline.ItemSeparator Property

Specifies a delimiting string for outline nodes.

Pascal

```
property ItemSeparator: string;
```

C++

```
__property AnsiString ItemSeparator;
```

Description

Set ItemSeparator to the string that should appear between node names in the path description of any individual node in the outline. ItemSeparator is used to implement the FullPath read method of TOutlineNode.

ItemSeparator is initialized to the backslash (\) character when the TCustomOutline object is created.

See Also

Items (see page 18271)

FullPath (see page 18361)

121.1.4.2.12 TOutline.Lines Property

Lists the Text property values of all nodes in the outline.

Pascal

```
property Lines: TStrings;
```

C++

```
__property TStrings Lines;
```

Description

Read Lines to obtain a text representation of the outline. Set Lines to fill the outline from a set of text strings. Each string in the Lines property represents a node in the outline. Lines does not include the root node.

Leading tabs and spaces are converted into levels of the outline. Text without any leading tabs or spaces becomes level 1 items. For example, the following set of strings represents an outline with three first level children descended from the root, and an additional, second level child on the first and third nodes:

```
ChildA
 ChildA1
ChildB
ChildC
 ChildC1
```

The Lines property is primarily useful for filling an outline with items at design time. While Lines can be used to access the individual nodes at run time, it is much quicker to access an item with the Items property.

See Also

Items (see page 18271)

Text (see page 18364)

121.1.4.2.13 TOutline.Options Property

Determines how the nodes in the outline are drawn.

Pascal

```
property Options: TGridOptions;
```

C++

```
__property TOutlineOptions Options;__property TGridOptions Options;
```

Description

Set Options to any combination of the following values:

Value	Meaning
ooDrawTreeRoot	The first item (Index value of 1) is connected to the root item by the outline tree. This means that the tree will extend from the top of the outline to all the first level items. Without ooDrawTreeRoot, all first level items appear leftmost in the outline, not connected by the tree.
ooDrawFocusRect	The outline draws a focus rectangle around the selected item.

ooStretchBitmaps	The outline stretches the standard bitmaps (PictureLeaf, PictureOpen, PictureClosed, PicturePlus, PictureMinus) to fit in the size of the item, determined by the size of the Font of the Text. Without ooStretchBitmaps, the bitmaps will be cropped if larger than the height of the item text, or won't fill up the entire item space if smaller than the text.

See Also

Items (see page 18271)

OutlineStyle (see page 18273)

PictureClosed (see page 18274)

PictureLeaf (see page 18274)

PictureMinus (see page 18275)

PictureOpen (see page 18275)

PicturePlus (see page 18276)

SelectedItem (see page 18277)

121.1.4.2.14 TOutline.OutlineStyle Property

Specifies how the outline tree is displayed.

Pascal

```
property OutlineStyle: TOutlineStyle;
```

C++

```
__property TOutlineStyle OutlineStyle;
```

Description

Set OutlineStyle to one of the following values:

Value	Meaning
osText	Displays node text (specified in Text) only.
osPlusMinusText	Displays plus picture (specified in PicturePlus), minus picture (specified in PictureMinus), and node text (specified in Text).
osPictureText	Displays open picture (specified in PictureOpen), closed picture (specified in PictureClosed), leaf picture (specified in PictureLeaf) and node text (specified in Text).
osPlusMinusPictureText	Displays plus picture (specified in PicturePlus), minus picture (specified in PictureMinus), open picture (specified in PictureOpen), closed picture (specified in PictureClosed), leaf picture (specified in PictureLeaf), and node text (specified in Text).
osTreeText	Displays lines connecting outline nodes and node text (specified in Text).
osTreePictureText	Displays lines connecting outline nodes, open picture (specified in PictureOpen), closed picture (specified in PictureClosed), leaf picture (specified in PictureLeaf), and node text (specified in Text).

Note: If the Style property is otOwnerDraw, the value of OutlineStyle is ignored. The connecting lines, node bitmaps, and node text must all be drawn in an OnDrawItem event handler.

See Also

Options (see page 18272)

PictureClosed (see page 18274)

PictureLeaf (see page 18274)

PictureMinus (see page 18275)

PictureOpen (see page 18275)

PicturePlus (see page 18276)

Style (see page 18277)

Text (see page 18364)

121.1.4.2.15 TOutline.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

121.1.4.2.16 TOutline.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

121.1.4.2.17 TOutline.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

121.1.4.2.18 TOutline.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

121.1.4.2.19 TOutline.PictureClosed Property

Specifies the bitmap used to represent a collapsed node that has subitems.

Pascal

```
property PictureClosed: TBitmap;
```

C++

```
__property TBitmap PictureClosed;
```

Description

By default, PictureClosed is a closed file folder bitmap. Set PictureClosed to change the bitmap used for collapsed nodes.

Note: The bitmap is not displayed in the outline unless the OutlineStyle property is set to osPictureText, osPlusMinusPictureText, or osTreePictureText.

See Also

OutlineStyle (see page 18273)

PictureLeaf (see page 18274)

PictureMinus (see page 18275)

PictureOpen (see page 18275)

121.1.4.2.20 TOutline.PictureLeaf Property

Specifies the bitmap used to represent an item that does not have subitems.

Pascal

```
property PictureLeaf: TBitmap;
```

C++

```
__property TBitmap PictureLeaf;
```

Description

By default, PictureLeaf is a document bitmap. Set PictureLeaf to change the bitmap used for leaf nodes.

Note: The bitmap is not displayed in the outline unless the OutlineStyle property is set to osPictureText, osPlusMinusPictureText, or osTreePictureText.

See Also

OutlineStyle (see page 18273)

PictureClosed (see page 18274)

PictureOpen (see page 18275)

121.1.4.2.21 TOutline.PictureMinus Property

Specifies the bitmap used to represent an expanded item that has subitems.

Pascal

```
property PictureMinus: TBitmap;
```

C++

```
__property TBitmap PictureMinus;
```

Description

By default, PictureMinus is a minus sign (-). Set PictureMinus to change the bitmap used for collapsible nodes.

Note: The bitmap is not displayed in the outline unless the OutlineStyle property is set to osPlusMinusPictureText or osPlusMinusText.

See Also

OutlineStyle (see page 18273)

PictureOpen (see page 18275)

PicturePlus (see page 18276)

121.1.4.2.22 TOutline.PictureOpen Property

Specifies the bitmap used to represent an expanded item that has subitems.

Pascal

```
property PictureOpen: TBitmap;
```

C++

```
__property TBitmap PictureOpen;
```

Description

By default, the PictureOpen property is an open file folder. Set PictureOpen to change the bitmap used for expanded nodes.

Note: The bitmap is not displayed in the outline unless the OutlineStyle property is set to osPictureText, osPlusMinusPictureText, or osTreePictureText.

See Also

OutlineStyle (see page 18273)

PictureClosed (see page 18274)

PictureLeaf (see page 18274)

PicturePlus (see page 18276)

121.1.4.2.23 TOutline.PicturePlus Property

Specifies the bitmap used to represent a collapsed item that has subitems.

Pascal

```
property PicturePlus: TBitmap;
```

C++

```
__property TBitmap PicturePlus;
```

Description

By default, the PicturePlus property is a plus sign (+). Set PicturePlus to change the bitmap used for collapsed nodes.

Note: The bitmap will not be displayed in the outline unless the OutlineStyle property is set to osPlusMinusPictureText or osPlusMinusText.

See Also

OutlineStyle (see page 18273)

PictureClosed (see page 18274)

PictureMinus (see page 18275)

121.1.4.2.24 TOutline.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
```

```
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
```

```
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;
```

```
procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
```

```
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

121.1.4.2.25 TOutline.ScrollBars Property

Specifies whether the grid includes horizontal and vertical scroll bars.

Pascal

```
property ScrollBars: TScrollStyle;
```

C++

```
__property TScrollStyle ScrollBars;
```

Description

Use ScrollBars to give the grid horizontal or vertical scroll bars.

Note: If all the cells in the grid fit in the ClientWidth, no horizontal scroll bar appears, even if ScrollBars is ssHorizontal or ssBoth. If all the cells fit in the ClientHeight, no vertical scroll bar appears, even if ScrollBars is ssVertical or ssBoth.

See Also

ClientHeight (see page 5808)

ClientWidth (see page 5811)

121.1.4.2.26 TOutline.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

121.1.4.2.27 TOutline.Style Property

Determines whether the nodes and tree are drawn automatically.

Pascal

```
property Style: TOutlineType;
```

C++

```
__property TOutlineType Style;
```

Description

By default, Style is osStandard, meaning that the outline displays items in the style determined by the OutlineStyle property. Set Style to otOwnerDraw to create owner-draw outlines, meaning that items must be drawn on the canvas of the outline by code in an OnDrawItem event handler.

Owner-draw outlines can display items other than the Text of a node and the standard bitmaps specified in the PictureClosed, PictureOpen, PictureMinus, PicturePlus, and PictureLeaf properties. Owner-draw outlines require more programming, however, as the application must specify how to render the image for each node in an OnDrawItem event handler.

See Also

Canvas (see page 5921)

OnDrawItem (see page 18278)

OutlineStyle (see page 18273)

121.1.4.2.28 TOutline.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

121.1.4.2.29 TOutline.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

121.1.4.2.30 TOutline.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
```

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

121.1.4.3 TOutline Events

The events of the TOutline class are listed here.

Published Events

	Name	Description
	OnClick (see page 18320)	Occurs when the user clicks the control.
	OnCollapse (see page 18322)	Occurs immediately after a node in the outline is collapsed.
	OnContextPopup (see page 18322)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 18323)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 18323)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 18325)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 18328)	Occurs when a node must be drawn in an owner-draw outline.
	OnEndDock (see page 18328)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 18329)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 18330)	Occurs when a control receives the input focus.
	OnExit (see page 18332)	Occurs when the input focus shifts away from one control to another.
	OnExpand (see page 18334)	Occurs immediately after a node in the outline is expanded.
	OnKeyDown (see page 18334)	Occurs when a user presses any key while the control has focus.

	OnKeyPress (see page 18337)	Occurs when a key is pressed.
	OnKeyUp (see page 18338)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 18339)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 18340)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 18343)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 18343)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 18343)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 18346)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 18348)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 18348)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

121.1.4.3.1 TOutline.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
```

```
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

121.1.4.3.2 TOutline.OnCollapse Event

Occurs immediately after a node in the outline is collapsed.

Pascal

```
property OnCollapse: EOutlineChange;
```

C++

```
__property EOutlineChange OnCollapse;
```

Description

Write an OnCollapse event handler to perform special processing when a node in the outline is collapsed. The event handler supplies the index of the node that collapsed.

See Also

OnExpand (see page 18279)

121.1.4.3.3 TOutline.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

121.1.4.3.4 TOutline.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

121.1.4.3.5 TOutline.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}
```

```
/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
```

```
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

121.1.4.3.6 TOutline.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;
```

```
procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

121.1.4.3.7 TOutline.OnDrawItem Event

Occurs when a node must be drawn in an owner-draw outline.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

When the Style property is set to otOwnerDraw, specify an OnDrawItem event handler to render the image of each node in the outline. An OnDrawItem event handler can display items other than the Text of a node and the standard bitmaps specified in the PictureClosed, PictureOpen, PictureMinus, PicturePlus, and PictureLeaf properties.

Draw the image of the node specified by the Index parameter onto the canvas of the outline.

See Also

Canvas (see page 5921)

PictureClosed (see page 18274)

PictureLeaf (see page 18274)

PictureMinus (see page 18275)

PictureOpen (see page 18275)

PicturePlus (see page 18276)

Style (see page 18277)

Text (see page 18364)

121.1.4.3.8 TOutline.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

121.1.4.3.9 TOutline.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

121.1.4.3.10 TOutline.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}
```

```
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
```

```
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

121.1.4.3.11 TOutline.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

121.1.4.3.12 TOutline.OnExpand Event

Occurs immediately after a node in the outline is expanded.

Pascal

```
property OnExpand: EOutlineChange;
```

C++

```
__property EOutlineChange OnExpand;
```

Description

Write an OnExpand event handler to perform special processing when a node in the outline is expanded. The event handler identifies the node that was just expanded.

See Also

OnCollapse (see page 18278)

121.1.4.3.13 TOutline.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
```

```
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;
```

```
procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
```

```
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

121.1.4.3.14 TOutline.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
```

```
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

121.1.4.3.15 TOutline.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
```

```
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

121.1.4.3.16 TOutline.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

121.1.4.3.17 TOutline.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
```

```
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
```

```
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

121.1.4.3.18 TOutline.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

121.1.4.3.19 TOutline.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

121.1.4.3.20 TOutline.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
```

```
  status bar.  (Set the Width of the status panels to 150
  before running this example).  When the user presses a mouse
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

121.1.4.3.21 TOutline.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

121.1.4.3.22 TOutline.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

121.1.4.3.23 TOutline.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

121.1.5 TOutlineNode Class

TOutlineNode represents a node in a TCustomOutline (see page 18234) object.

Class Hierarchy

Pascal

```
TOutlineNode = class(TPersistent);
```

C++

```
class TOutlineNode : public TPersistent;
```

File

Outline

Description

Use TOutlineNode to describe a node in a Win 3.1 outline. Each outline node appears as a line, or row, of the outline. An outline node can have from 0 to 16368 subnodes, which are subordinate to the parent node in the outline structure.

See Also

TOutline (see page 18279)

121.1.5.1 TOutlineNode Members

The following tables list the members exposed by TOutlineNode.

Protected Methods

	Name	Description
	AddNode (see page 18353)	This is AddNode, a member of class TOutlineNode.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDataItem (see page 18355)	This is GetDataItem, a member of class TOutlineNode.
	GetNodeAtIndex (see page 18357)	This is GetNodeAtIndex, a member of class TOutlineNode.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRowOfNode (see page 18357)	This is GetRowOfNode, a member of class TOutlineNode.
	GetTextItem (see page 18358)	This is GetTextItem, a member of class TOutlineNode.
	GetVisibleNode (see page 18358)	This is GetVisibleNode, a member of class TOutlineNode.
	HasAsParent (see page 18358)	This is HasAsParent, a member of class TOutlineNode.
	InsertNode (see page 18358)	This is InsertNode, a member of class TOutlineNode.
	InternalRemove (see page 18359)	This is InternalRemove, a member of class TOutlineNode.
	Remove (see page 18359)	This is Remove, a member of class TOutlineNode.
	WriteNode (see page 18359)	This is WriteNode, a member of class TOutlineNode.

Protected Properties

	Name	Description
R	ExpandCount (see page 18361)	This is ExpandCount, a member of class TOutlineNode.
R	Items (see page 18363)	This is Items, a member of class TOutlineNode.
R	List (see page 18364)	This is List, a member of class TOutlineNode.
R	Outline (see page 18364)	This is Outline, a member of class TOutlineNode.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ChangeLevelBy (see page 18353)	Changes the level of an outline node.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Collapse (see page 18354)	Collapses an outline item.
	Create (see page 18351)	Creates a new TOutlineNode (see page 18349) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 18352)	Frees the memory associated with the TOutlineNode (see page 18349) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Expand (see page 18354)	Expands an outline item.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	FullExpand (see page 18355)	Expands an outline node and all its descendants.
	GetDisplayWidth (see page 18355)	Returns the distance, in pixels, from the left edge of the outline to the right edge of the outline node.
	getFirstChild (see page 18356)	This is getFirstChild, a member of class TOutlineNode.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetLastChild (see page 18356)	Returns the Index (see page 18362) value of the last subnode.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNextChild (see page 18356)	Returns the Index (see page 18362) value of the next child of an outline node.
	GetPrevChild (see page 18357)	Returns the Index (see page 18362) value of the previous child of an outline node.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveTo (see page 18359)	Moves an outline node from one location within the outline to another.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Data (see page 18360)	Points to application-specific data associated with the node.
	Expanded (see page 18361)	Determines whether descendants of the outline node are displayed.
R	FullPath (see page 18361)	Describes an outline node, using its text and the text of all ancestor nodes in the outline.
R	HasItems (see page 18362)	Indicates whether the outline node has any subnodes.
R	Index (see page 18362)	Identifies the node in the Items (see page 18363) array maintained by the outline.
R	IsVisible (see page 18363)	Indicates whether the outline node is visible in the outline.
	Level (see page 18363)	Indicates the level of indentation of a node within the outline.
R	Parent (see page 18364)	Identifies the immediate ancestor of the node.
	Text (see page 18364)	Specifies the string that identifies the outline node.
R	TopItem (see page 18365)	Indicates which outline node is the topmost ancestor of a node.

121.1.5.2 TOutlineNode.Create Constructor

Creates a new TOutlineNode (see page 18349) object.

Pascal

```
constructor Create(AOwner: TCustomOutline);
```

C++

```
__fastcall TOutlineNode(TCustomOutline AOwner);
```

Description

Do not call the constructor for TOutlineNode (see page 18349) directly. Instead, use one of the methods of the outline component in which the node should appear. These methods are as follows:

Method	Action
Add	Creates the node as the last sibling of a specified node.

AddObject	Same as Add, but also initializing the node's Data (see page 18360) property.
AddChild	Creates the node as the last child of a specified node.
AddChildObject	Same as AddChild, but also initializing the node's Data (see page 18360) property.
Insert	Creates the node in the current position of an existing node.
InsertObject	Same as Insert, but also initializing the node's Data (see page 18360) property.

See Also

Data (see page 18360)

Add (see page 18250)

AddObject (see page 18252)

AddChild (see page 18250)

Insert (see page 18260)

AddChildObject (see page 18251)

InsertObject (see page 18261)

121.1.5.3 TOutlineNode.Destroy Destructor

Frees the memory associated with the TOutlineNode (see page 18349) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TOutlineNode();
```

Description

Do not call Destroy directly. The preferred method of deleting nodes is to use the Delete method of the containing outline component.

See Also

Free (see page 22300)

Delete (see page 18255)

121.1.5.4 TOutlineNode Methods

The methods of the TOutlineNode class are listed here.

Protected Methods

	Name	Description
	AddNode (see page 18353)	This is AddNode, a member of class TOutlineNode.
	GetDataItem (see page 18355)	This is GetDataItem, a member of class TOutlineNode.
	GetNodeAtIndex (see page 18357)	This is GetNodeAtIndex, a member of class TOutlineNode.
	GetRowOfNode (see page 18357)	This is GetRowOfNode, a member of class TOutlineNode.
	GetTextItem (see page 18358)	This is GetTextItem, a member of class TOutlineNode.
	GetVisibleNode (see page 18358)	This is GetVisibleNode, a member of class TOutlineNode.

	HasAsParent (see page 18358)	This is HasAsParent, a member of class TOutlineNode.
	InsertNode (see page 18358)	This is InsertNode, a member of class TOutlineNode.
	InternalRemove (see page 18359)	This is InternalRemove, a member of class TOutlineNode.
	Remove (see page 18359)	This is Remove, a member of class TOutlineNode.
	WriteNode (see page 18359)	This is WriteNode, a member of class TOutlineNode.

Public Methods

	Name	Description
	ChangeLevelBy (see page 18353)	Changes the level of an outline node.
	Collapse (see page 18354)	Collapses an outline item.
	Expand (see page 18354)	Expands an outline item.
	FullExpand (see page 18355)	Expands an outline node and all its descendants.
	GetDisplayWidth (see page 18355)	Returns the distance, in pixels, from the left edge of the outline to the right edge of the outline node.
	getFirstChild (see page 18356)	This is getFirstChild, a member of class TOutlineNode.
	GetLastChild (see page 18356)	Returns the Index (see page 18362) value of the last subnode.
	GetNextChild (see page 18356)	Returns the Index (see page 18362) value of the next child of an outline node.
	GetPrevChild (see page 18357)	Returns the Index (see page 18362) value of the previous child of an outline node.
	MoveTo (see page 18359)	Moves an outline node from one location within the outline to another.

121.1.5.4.1 TOutlineNode.AddNode Method

Pascal

```
function AddNode(Value: TOutlineNode): LongInt;
```

C++

```
__fastcall LongInt AddNode(TOutlineNode Value);
```

Description

This is AddNode, a member of class TOutlineNode.

121.1.5.4.2 TOutlineNode.ChangeLevelBy Method

Changes the level of an outline node.

Pascal

```
procedure ChangeLevelBy(Value: TChangeRange);
```

C++

```
__fastcall ChangeLevelBy(TChangeRange Value);
```

Description

Call ChangeLevelBy to move a node up or down a branch in the outline. Specify a Value parameter value of -1 to move up (toward the root) one level. Specify a Value parameter value of 1 to move down (away from the root) one level.

When moving up one level, an item becomes the next sibling of its former parent. When moving down one level, an item becomes the last child of its former prior sibling.

Nodes at the first level can not be moved up, as they have no parent. The first node in the outline can not be moved down either, as it has no prior sibling.

Note: Before calling ChangeLevelBy, be sure that the node has the appropriate number of parents or that the appropriate prior siblings exist.

See Also

Level (see page 18363)

MoveTo (see page 18359)

Parent (see page 18364)

121.1.5.4.3 TOutlineNode.Collapse Method

Collapses an outline item.

Pascal

```
procedure Collapse;
```

C++

```
__fastcall Collapse();
```

Description

Collapse sets the Expanded (see page 18361) property to false. When an outline node is collapsed, its descendants are hidden and, depending on the OutlineStyle property of the outline in which the node is contained, the plus picture or closed picture is displayed.

Collapse does nothing to nodes that have no descendants.

See Also

Expand (see page 18354)

Expanded (see page 18361)

OutlineStyle (see page 18273)

121.1.5.4.4 TOutlineNode.Expand Method

Expands an outline item.

Pascal

```
procedure Expand;
```

C++

```
__fastcall Expand();
```

Description

Expand sets the Expanded (see page 18361) property to true. When an outline node is expanded, its descendants are displayed and, depending on the OutlineStyle property of the outline in which the node is contained, the minus picture or open picture is displayed.

Expand does nothing to nodes that have no descendants.

Note: Expand only ensures that the immediate children of a node are visible. To expand all descendants of a node, use the FullExpand (see page 18355) method.

See Also

Collapse (see page 18354)

Expanded (see page 18361)

FullExpand (see page 18355)

OutlineStyle (see page 18273)

121.1.5.4.5 TOutlineNode.FullExpand Method

Expands an outline node and all its descendants.

Pascal

```
procedure FullExpand;
```

C++

```
__fastcall FullExpand();
```

Description

Use FullExpand to expand all the nodes in the subtree of the outline rooted at the outline node. FullExpand sets the Expanded (see page 18361) property of the node and all its descendants to true.

See Also

Collapse (see page 18354)

Expand (see page 18354)

Expanded (see page 18361)

121.1.5.4.6 TOutlineNode.GetDataItem Method

Pascal

```
function GetDataItem(Value: Pointer): LongInt;
```

C++

```
__fastcall LongInt GetDataItem(void * Value);
```

Description

This is GetDataItem, a member of class TOutlineNode.

121.1.5.4.7 TOutlineNode.GetDisplayWidth Method

Returns the distance, in pixels, from the left edge of the outline to the right edge of the outline node.

Pascal

```
function GetDisplayWidth: Integer;
```

C++

```
__fastcall int GetDisplayWidth();
```

Description

Call GetDisplayWidth to determine the length of the line in the outline that contains the outline node. GetDisplayWidth adjusts for the lines and bitmaps specified by the OutlineStyle property of the outline.

See Also

GetNodeDisplayWidth (see page 18259)

OutlineStyle (see page 18273)

121.1.5.4.8 TOutlineNode.getFirstChild Method

Pascal

```
function getFirstChild: LongInt;
```

C++

```
__fastcall LongInt getFirstChild();
```

Description

This is getFirstChild, a member of class TOutlineNode.

121.1.5.4.9 TOutlineNode.GetLastChild Method

Returns the Index (see page 18362) value of the last subnode.

Pascal

```
function GetLastChild: LongInt;
```

C++

```
__fastcall LongInt GetLastChild();
```

Description

Call GetLastChild to locate the last subnode of a node. If the node has no descendants, GetLastChild returns -1.

See Also

GetFirstChild (see page 18356)

GetNextChild (see page 18356)

GetPrevChild (see page 18357)

HasItems (see page 18362)

Index (see page 18362)

121.1.5.4.10 TOutlineNode.GetNextChild Method

Returns the Index (see page 18362) value of the next child of an outline node.

Pascal

```
function GetNextChild(Value: LongInt): LongInt;
```

C++

```
__fastcall LongInt GetNextChild(LongInt Value);
```

Description

Call GetNextChild to obtain the index of the next immediate descendant of a node, following the child node with the index given by the Value parameter. If the child node indicated by Value has no next sibling, or if the child node indicated by Value is not an immediate descendant of the outline node, GetNextChild returns -1.

Index (see page 18362) values reflect a depth-first traversal of the outline. Thus, the node returned by GetNextChild is not simply the node with an index of Value+1, unless the node with index Value has no descendants. GetNextChild allows a breadth-first traversal of the outline.

See Also

GetFirstChild (see page 18356)

GetLastChild (see page 18356)

GetPrevChild (see page 18357)

Index (see page 18362)

121.1.5.4.11 TOutlineNode.GetNodeAtIndex Method

Pascal

```
function GetNodeAtIndex(TargetIndex: LongInt): TOutlineNode;
```

C++

```
__fastcall TOutlineNode GetNodeAtIndex(LongInt TargetIndex);
```

Description

This is GetNodeAtIndex, a member of class TOutlineNode.

121.1.5.4.12 TOutlineNode.GetPrevChild Method

Returns the Index (see page 18362) value of the previous child of an outline node.

Pascal

```
function GetPrevChild(Value: LongInt): LongInt;
```

C++

```
__fastcall LongInt GetPrevChild(LongInt Value);
```

Description

Call GetPrevChild to obtain the index of the previous immediate descendant of a node, preceding the child node with the index given by the Value parameter. If the child node indicated by Value has no prior sibling, or if the child node indicated by Value is not an immediate descendant of the outline node, GetPrevChild returns -1.

Index (see page 18362) values reflect a depth-first traversal of the outline. Thus, the previous sibling of the child node indicated by Value is not simply one less than Value, unless that previous sibling has no descendants.

See Also

GetFirstChild (see page 18356)

GetLastChild (see page 18356)

GetNextChild (see page 18356)

Index (see page 18362)

121.1.5.4.13 TOutlineNode.GetRowOfNode Method

Pascal

```
function GetRowOfNode(TargetNode: TOutlineNode; var RowCount: Longint): Boolean;
```

C++

```
__fastcall Boolean GetRowOfNode(TOutlineNode TargetNode, Longint RowCount);
```

Description

This is GetRowOfNode, a member of class TOutlineNode.

121.1.5.4.14 TOutlineNode.GetTextItem Method

Pascal

```
function GetTextItem(const Value: string): LongInt;
```

C++

```
__fastcall LongInt GetTextItem(const AnsiString Value);
```

Description

This is GetTextItem, a member of class TOutlineNode.

121.1.5.4.15 TOutlineNode.GetVisibleNode Method

Pascal

```
function GetVisibleNode(TargetCount: LongInt): TOutlineNode;
```

C++

```
__fastcall TOutlineNode GetVisibleNode(LongInt TargetCount);
```

Description

This is GetVisibleNode, a member of class TOutlineNode.

121.1.5.4.16 TOutlineNode.HasAsParent Method

Pascal

```
function HasAsParent(Value: TOutlineNode): Boolean;
```

C++

```
__fastcall Boolean HasAsParent(TOutlineNode Value);
```

Description

This is HasAsParent, a member of class TOutlineNode.

121.1.5.4.17 TOutlineNode.InsertNode Method

Pascal

```
function InsertNode(Index: LongInt; Value: TOutlineNode): LongInt;
```

C++

```
__fastcall LongInt InsertNode(LongInt Index, TOutlineNode Value);
```

Description

This is InsertNode, a member of class TOutlineNode.

121.1.5.4.18 TOutlineNode.InternalRemove Method

Pascal

```
procedure InternalRemove(Value: TOutlineNode; Index: Integer);
```

C++

```
__fastcall InternalRemove(TOutlineNode Value, int Index);
```

Description

This is InternalRemove, a member of class TOutlineNode.

121.1.5.4.19 TOutlineNode.MoveTo Method

Moves an outline node from one location within the outline to another.

Pascal

```
procedure MoveTo(Destination: LongInt; AttachMode: TAttachMode);
```

C++

```
__fastcall MoveTo(LongInt Destination, TAttachMode AttachMode);
```

Description

Call MoveTo to reposition the node to the position indicated by the Destination parameter. Destination is the Index (see page 18362) value of another outline node. The AttachMode parameter specifies how to attach the moved node to the destination position.

Note: To change only the level of a node, use the ChangeLevelBy (see page 18353) method.

See Also

ChangeLevelBy (see page 18353)

Index (see page 18362)

Add (see page 18250)

AddChild (see page 18250)

Insert (see page 18260)

121.1.5.4.20 TOutlineNode.Remove Method

Pascal

```
procedure Remove(Value: TOutlineNode);
```

C++

```
__fastcall Remove(TOutlineNode Value);
```

Description

This is Remove, a member of class TOutlineNode.

121.1.5.4.21 TOutlineNode.WriteNode Method

Pascal

```
procedure WriteNode(Buffer: PChar; Stream: TStream);
```

C++

```
__fastcall WriteNode(const char * Buffer, TStream Stream);
```

Description

This is WriteNode, a member of class TOutlineNode.

121.1.5.5 TOutlineNode Properties

The properties of the TOutlineNode class are listed here.

Protected Properties

	Name	Description
R	ExpandCount (see page 18361)	This is ExpandCount, a member of class TOutlineNode.
R	Items (see page 18363)	This is Items, a member of class TOutlineNode.
R	List (see page 18364)	This is List, a member of class TOutlineNode.
R	Outline (see page 18364)	This is Outline, a member of class TOutlineNode.

Public Properties

	Name	Description
	Data (see page 18360)	Points to application-specific data associated with the node.
	Expanded (see page 18361)	Determines whether descendants of the outline node are displayed.
R	FullPath (see page 18361)	Describes an outline node, using its text and the text of all ancestor nodes in the outline.
R	HasItems (see page 18362)	Indicates whether the outline node has any subnodes.
R	Index (see page 18362)	Identifies the node in the Items (see page 18363) array maintained by the outline.
R	IsVisible (see page 18363)	Indicates whether the outline node is visible in the outline.
	Level (see page 18363)	Indicates the level of indentation of a node within the outline.
R	Parent (see page 18364)	Identifies the immediate ancestor of the node.
	Text (see page 18364)	Specifies the string that identifies the outline node.
R	TopItem (see page 18365)	Indicates which outline node is the topmost ancestor of a node.

121.1.5.5.1 TOutlineNode.Data Property

Points to application-specific data associated with the node.

Pascal

```
property Data: Pointer;
```

C++

```
__property void * Data;
```

Description

Use Data to associate an arbitrary value with a node in an outline. For example, if the outline represents the table of contents to an electronic document, Data could point to an object that provides jump information for accessing the section of the document named by the node.

The Data property is usually described by the Text (see page 18364) property, which is displayed in the rendered image of the node.

See Also

Text (see page 18364)

121.1.5.5.2 TOutlineNode.ExpandCount Property

Pascal

```
property ExpandCount: LongInt;
```

C++

```
__property LongInt ExpandCount;
```

Description

This is ExpandCount, a member of class TOutlineNode.

121.1.5.5.3 TOutlineNode.Expanded Property

Determines whether descendants of the outline node are displayed.

Pascal

```
property Expanded: Boolean;
```

C++

```
__property Boolean Expanded;
```

Description

Set Expanded to true to display all subnodes of a node. When an outline node is expanded, its descendants are displayed and, depending on the OutlineStyle property of the outline in which the node is contained, the minus picture or open picture is displayed.

Set Expanded to false to collapse the node. When an outline node is collapsed, its descendants are hidden and, depending on the OutlineStyle property of the outline in which the node is contained, the plus picture or closed picture is displayed.

Expanded is not meaningful for nodes that have no descendants.

See Also

HasItems (see page 18362)

OutlineStyle (see page 18273)

121.1.5.5.4 TOutlineNode.FullPath Property

Describes an outline node, using its text and the text of all ancestor nodes in the outline.

Pascal

```
property FullPath: string;
```

C++

```
__property AnsiString FullPath;
```

Description

Read FullPath to obtain a description of the entire branch of the outline leading to the outline node. FullPath consists of the values of the Text (see page 18364) properties of all outline items, starting with the text of the node at the first level and traversing the outline down to the outline node. The string specified as the ItemSeparator property of the outline separates the text for each node in the path.

See Also

Level (see page 18363)

Parent (see page 18364)

ItemSeparator (see page 18271)

Text (see page 18364)

TopItem (see page 18365)

121.1.5.5.5 TOutlineNode.HasItems Property

Indicates whether the outline node has any subnodes.

Pascal

```
property HasItems: Boolean;
```

C++

```
__property Boolean HasItems;
```

Description

Read HasItem to determine whether the outline node is a leaf node. Depending on the OutlineStyle property of the outline that contains the node, leaf nodes may appear with the leaf bitmap. If the node is not a leaf, subnodes appear below and indented from the node when it is expanded.

HasItem returns true if the node has any descendants, even if the node is collapsed and the descendants are not visible. HasItem returns false if the node is a leaf node.

See Also

Expanded (see page 18361)

Parent (see page 18364)

OutlineStyle (see page 18273)

121.1.5.5.6 TOutlineNode.Index Property

Identifies the node in the Items (see page 18363) array maintained by the outline.

Pascal

```
property Index: LongInt;
```

C++

```
__property LongInt Index;
```

Description

Read Index to determine the row of the outline where the node appears. The first node has an Index value of 1, and subsequent nodes are indexed sequentially. If a node has subnodes, the Index value of the parent node is one less than the Index value of its first descendant.

When nodes in the outline are added, moved, or deleted, the values of the Index properties of all subsequent nodes must be recalculated to be valid. Sometimes, the recalculation of indexes is deferred until all changes to the nodes are complete. To ensure that the value of Index is valid, call the SetUpdateState method of the outline that contains the node.

See Also

Items (see page 18271)

SetUpdateState (see page 18267)

121.1.5.5.7 TOutlineNode.IsVisible Property

Indicates whether the outline node is visible in the outline.

Pascal

```
property IsVisible: Boolean;
```

C++

```
__property Boolean IsVisible;
```

Description

Read IsVisible to determine whether the node is part of the current image of the outline. A node is visible if it is on level 1 or if all its parents are expanded.

Note: Do not confuse IsVisible with the Visible property of TControl objects. Outline (see page 18364) nodes are not controls, but merely helper objects that describe nodes for an outline control. IsVisible indicates whether a node is visible because its parents are expanded or collapsed. It does not provide information about whether the node has been scrolled out of view, or even if the outline is not visible.

See Also

Expanded (see page 18361)

Visible (see page 5847)

121.1.5.5.8 TOutlineNode.Items Property

Pascal

```
property Items [Index: LongInt]: TOutlineNode;
```

C++

```
__property TOutlineNode Items[LongInt Index];
```

Description

This is Items, a member of class TOutlineNode.

121.1.5.5.9 TOutlineNode.Level Property

Indicates the level of indentation of a node within the outline.

Pascal

```
property Level: Cardinal;
```

C++

```
__property unsigned Level;
```

Description

The value of Level is 1 for nodes that directly descend from the root of the outline (the top level). The value of Level is 2 for their children, and so on.

See Also

ChangeLevelBy (see page 18353)

Parent (see page 18364)

TopItem (see page 18365)

121.1.5.5.10 TOutlineNode.List Property

Pascal

```
property List: TList;
```

C++

```
__property TList List;
```

Description

This is List, a member of class TOutlineNode.

121.1.5.5.11 TOutlineNode.Outline Property

Pascal

```
property Outline: TCustomOutline;
```

C++

```
__property TCustomOutline Outline;
```

Description

This is Outline, a member of class TOutlineNode.

121.1.5.5.12 TOutlineNode.Parent Property

Identifies the immediate ancestor of the node.

Pascal

```
property Parent: TOutlineNode;
```

C++

```
__property TOutlineNode Parent;
```

Description

Read Parent to find the parent of an outline node. A parent node is one level higher and contains the child node as a subnode.

See Also

Level (see page 18363)

TopItem (see page 18365)

121.1.5.5.13 TOutlineNode.Text Property

Specifies the string that identifies the outline node.

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

Use Text to determine the string that is displayed in the outline to represent the node.

See Also

FullPath (see page 18361)

121.1.5.5.14 TOutlineNode.TopItem Property

Indicates which outline node is the topmost ancestor of a node.

Pascal

```
property TopItem: Longint;
```

C++

```
__property Longint TopItem;
```

Description

Read TopItem to determine the first node in the branch of the outline tree that contains the outline node. The TopItem property specifies the Index (see page 18362) value of the level 1 ancestor of the outline node. For a node on level 1, TopItem has the same value as the Index (see page 18362) property.

See Also

Index (see page 18362)

Level (see page 18363)

Parent (see page 18364)

121.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAttachMode (see page 18366)	TAttachMode indicates how an outline node should be inserted into an outline.
	TOutlineBitmap (see page 18367)	This is record Outline.TOutlineBitmap.
	TOutlineNodeCompare (see page 18367)	This is record Outline.TOutlineNodeCompare.
	TOutlineOption (see page 18368)	TOutlineOptions (see page 18373) represents the style in which an outline is drawn.
	TOutlineStyle (see page 18368)	TOutlineStyle determines how the nodes of an outline are drawn.
	TOutlineType (see page 18369)	TOutlineType indicates whether an outline is owner-drawn.

121.2.1 Outline.TAttachMode Enumeration

TAttachMode indicates how an outline node should be inserted into an outline.

Pascal

```
TAttachMode = (
  oaAdd,
  oaAddChild,
  oaInsert
);
```

C++

```
enum TAttachMode {
  oaAdd,
  oaAddChild,
  oaInsert
};
```

File

Outline

Description

TAttachMode is used by methods that move outline nodes within an outline. It includes the following values:

Value	Description
oaAdd	The item is attached as if added with the Add method. The moved item becomes the last sibling of the item specified by the Destination parameter, sharing the same parent as the Destination item.
oaAddChild	The item is attached as if added with the AddChild method. The moved item becomes the last child of the item specified by the Destination parameter. The Destination item becomes the parent of the moved item.
oaInsert	The item is attached as if inserted with the Insert method. The moved item replaces the Destination item in the outline, while the Destination item and all other following items are moved down one row.

See Also

TCustomOutline

TOutlineNode

121.2.2 Outline.TOutlineBitmap Enumeration

Pascal

```
TOutlineBitmap = (
  obPlus,
  obMinus,
  obOpen,
  obClose,
  obLeaf
);
```

C++

```
enum TOutlineBitmap {
  obPlus,
  obMinus,
  obOpen,
  obClose,
  obLeaf
};
```

File

Outline

Description

This is record Outline.TOutlineBitmap.

121.2.3 Outline.TOutlineNodeCompare Enumeration

Pascal

```
TOutlineNodeCompare = (
  ocLess,
  ocSame,
  ocGreater,
  ocInvalid
);
```

C++

```
enum TOutlineNodeCompare {
  ocLess,
  ocSame,
  ocGreater,
  ocInvalid
};
```

File

Outline

Description

This is record Outline.TOutlineNodeCompare.

121.2.4 Outline.TOutlineOption Enumeration

TOutlineOptions (see page 18373) represents the style in which an outline is drawn.

Pascal

```
TOutlineOption = (
  ooDrawTreeRoot,
  ooDrawFocusRect,
  ooStretchBitmaps
);
```

C++

```
enum TOutlineOption {
  ooDrawTreeRoot,
  ooDrawFocusRect,
  ooStretchBitmaps
};
```

File

Outline

Description

TOutlineOptions (see page 18373) is a set of TOutlineOption values. Together they influence the way an outline is drawn. Each TOutlineOption value is described below:

Value	Meaning
ooDrawTreeRoot	The first item (Index value of 1) is connected to the root item by the outline tree. This means that the tree will extend from the top of the outline to all the first level items. Without ooDrawTreeRoot, all first level items appear leftmost in the outline, not connected by the tree.
ooDrawFocusRect	The outline draws a focus rectangle around the selected item.
ooStretchBitmaps	The outline stretches the standard bitmaps (PictureLeaf, PictureOpen, PictureClosed, PicturePlus, PictureMinus) to fit in the size of the item, determined by the size of the Font of the Text. Without ooStretchBitmaps, the bitmaps will be cropped if larger than the height of the item text, or won't fill up the entire item space if smaller than the text.

See Also

TCustomOutline (see page 18234)

121.2.5 Outline.TOutlineStyle Enumeration

TOutlineStyle determines how the nodes of an outline are drawn.

Pascal

```
TOutlineStyle = (
  osText,
  osPlusMinusText,
  osPictureText,
  osPlusMinusPictureText,
  osTreeText,
  osTreePictureText
);
```

C++

```
enum TOutlineStyle {
  osText,
  osPlusMinusText,
  osPictureText,
  osPlusMinusPictureText,
  osTreeText,
  osTreePictureText
};
```

File

Outline

Description

TOutlineStyle is the type of TCustomOutline (see page 18234).::OutlineStyle. It includes the following values:

Value	Meaning
osText	Only the text of nodes is drawn.
osPlusMinusText	In addition to the text, each node displays plus picture (when it can be expanded) or minus picture (when it can be collapsed).
osPictureText	In addition to the text, each node displays the open picture (when it is expanded), closed picture (when it is collapsed), or leaf picture (when it has no subnodes).
osPlusMinusPictureText	In addition to the text, each node displays the plus picture or minus picture (as in osPlusMinusText), and the open, closed or leaf picture (as in osPictureText).
osTreeText	Visible lines connect the outline nodes.
osTreePictureText	In addition to the text, each node displays the open picture (when it is expanded), closed picture (when it is collapsed), or leaf picture (when it has no subnodes). Visible lines connect the outline nodes.

See Also

TCustomOutline

121.2.6 Outline.TOutlineType Enumeration

TOutlineType indicates whether an outline is owner-drawn.

Pascal

```
TOutlineType = (
  otStandard,
  otOwnerDraw
);
```

C++

```
enum TOutlineType {
  otStandard,
  otOwnerDraw
};
```

File

Outline

Description

TOutlineType determines whether an outline is drawn automatically or by user code in an OnDrawItem event.

Value	Meaning
otStandardItems	Items are drawn automatically.
otOwnerDraw	Items are drawn on the Canvas by application code.

See Also

TCustomOutline

TCustomOutline

121.3 Types

The following table lists types in this documentation.

Types

Name	Description
EOutlineChange (see page 18371)	EOutlineChange is the type of event handlers that respond when the nodes in an outline control change.
TBitmapArray (see page 18371)	This is type Outline.TBitmapArray.
TBitmapArrayRange (see page 18372)	This is type Outline.TBitmapArrayRange.
TChangeRange (see page 18372)	TChangeRange indicates the direction in which a node is moved.
TOutlineBitmaps (see page 18372)	This is type Outline.TOutlineBitmaps.
TOutlineOptions (see page 18373)	TOutlineOptions represents the style in which an outline is drawn.

121.3.1 Outline.EOutlineChange Type

EOutlineChange is the type of event handlers that respond when the nodes in an outline control change.

Pascal

```
EOutlineChange = procedure (Sender: TObject; Index: LongInt) of object;
```

C++

```
(Sender: TObject; Index: LongInt) ( EOutlineChange)();
```

File

Outline

Description

Sender is the outline control whose node just changed.

Index is the index of the node that just changed.

121.3.2 Outline.TBitmapArray Type

Pascal

```
TBitmapArray = array[TBitmapArrayRange] of TBitmap;
```

C++

```
array[TBitmapArrayRange] of TBitmap TBitmapArray;
```

File

Outline

Description

This is type Outline.TBitmapArray.

121.3.3 Outline.TBitmapArrayRange Type

Pascal

```
TBitmapArrayRange = 0..4;
```

C++

```
0..4 TBitmapArrayRange;
```

File

Outline

Description

This is type Outline.TBitmapArrayRange.

121.3.4 Outline.TChangeRange Type

TChangeRange indicates the direction in which a node is moved.

Pascal

```
TChangeRange = -1..1;
```

C++

```
-1..1 TChangeRange;
```

File

Outline

Description

TChangeRange is the type of the Value parameter of an outline node's ChangeLevelBy method. It can take one of the following values:

Value	Meaning
-1	Move the node toward the root by one level.
1	Move the node toward the leaf by one level.

See Also

TOutlineNode (see page 18349)

121.3.5 Outline.TOutlineBitmaps Type

Pascal

```
TOutlineBitmaps = set of TOutlineBitmap;
```

C++

```
TOutlineBitmap TOutlineBitmaps;
```

File

Outline

Description

This is type Outline.TOutlineBitmaps.

121.3.6 Outline.TOutlineOptions Type

TOutlineOptions represents the style in which an outline is drawn.

Pascal

```
TOutlineOptions = set of TOutlineOption;
```

C++

```
TOutlineOption TOutlineOptions;
```

File

Outline

Description

TOutlineOptions is a set of TOutlineOption (see page 18368) values. Together they influence the way an outline is drawn. Each TOutlineOption (see page 18368) value is described below:

Value	Meaning
ooDrawTreeRoot	The first item (Index value of 1) is connected to the root item by the outline tree. This means that the tree will extend from the top of the outline to all the first level items. Without ooDrawTreeRoot, all first level items appear leftmost in the outline, not connected by the tree.
ooDrawFocusRect	The outline draws a focus rectangle around the selected item.
ooStretchBitmaps	The outline stretches the standard bitmaps (PictureLeaf, PictureOpen, PictureClosed, PicturePlus, PictureMinus) to fit in the size of the item, determined by the size of the Font of the Text. Without ooStretchBitmaps, the bitmaps will be cropped if larger than the height of the item text, or won't fill up the entire item space if smaller than the text.

See Also

TCustomOutline

122 PagItems Namespace

This is namespace PagItems.

Topics

Name	Description
Classes (see page 18376)	The following table lists classes in this documentation.

Classes

	Name	Description
	TBasePageItemsProducer (see page 18376)	TBasePageItemsProducer produces a string of HTML commands based on an input template.
	TPageItemsProducer (see page 18383)	TPageItemsProducer maintains a list of Web item components for producing a string of HTML commands based on an input template.

122.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TBasePageItemsProducer (see page 18376)	TBasePageItemsProducer produces a string of HTML commands based on an input template.
	TPageItemsProducer (see page 18383)	TPageItemsProducer maintains a list of Web item components for producing a string of HTML commands based on an input template.

122.1.1 TBasePageItemsProducer Class

TBasePageItemsProducer produces a string of HTML commands based on an input template.

Class Hierarchy

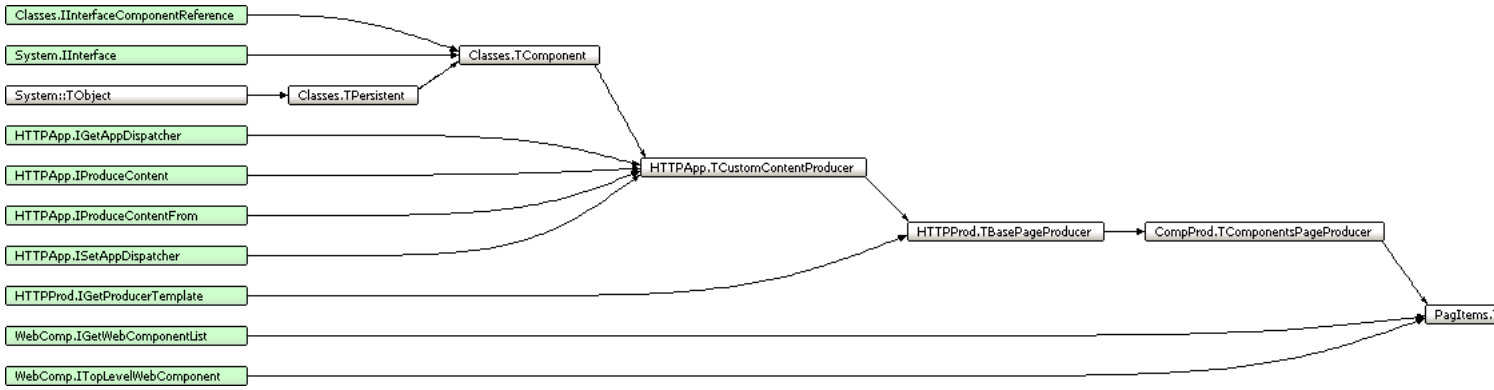

Pascal

```
TBasePageItemsProducer = class(TComponentsPageProducer, IGetWebComponentList,
ITopLevelWebComponent);
```

C++

```
class TBasePageItemsProducer : public TComponentsPageProducer, public IGetWebComponentList,
public ITopLevelWebComponent;
```

File

PagItems

Description

Use TBasePageItemsProducer to convert an HTML template that includes content generated by other components into a string of HTML commands that can be interpreted by a client application such as a Web browser. HTML templates include HTML commands and HTML-transparent tags that are replaced with customized content by the components page producer in an OnHTMLTag event handler.

TBasePageItemsProducer differs from other page producer components in that it replaces special HTML-transparent tags of the format

```
<#COMPONENT Name=WebContentComponent>
```

with the HTML produced by the components named WebContentComponent. In addition, TBasePageItemsProducer stores a default value for WebContentComponent as the value of its WebPageItems (see page 18382) property.

See Also

IWebContent (see page 24792)

TPageItemsProducer (see page 18383)

TInetXPageProducer (see page 17145)

TAdapterPageProducer (see page 19802)

122.1.1.1 TBasePageItemsProducer Members

The following tables list the members exposed by TBasePageItemsProducer.

Protected Events

	Name	Description
	OnHTMLTag (see page 15280)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoTagEvent (see page 5618)	This is DoTagEvent, a member of class TComponentsPageProducer.
	DoTagEvent (see page 15276)	This is DoTagEvent, a member of class TBasePageProducer.
V	FindComponent (see page 5619)	This is FindComponent, a member of class TComponentsPageProducer.
	FindComponentName (see page 5619)	This is FindComponentName, a member of class TComponentsPageProducer.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
V	GetChildren (see page 18381)	This is GetChildren, a member of class TBasePageItemsProducer.
V	GetComponentContent (see page 5619)	This is GetComponentContent, a member of class TComponentsPageProducer.
	GetComponentList (see page 18382)	This is GetComponentList, a member of class TBasePageItemsProducer.
V	GetContentOptions (see page 5619)	This is GetContentOptions, a member of class TComponentsPageProducer.
	GetDefaultComponentList (see page 18382)	This is GetDefaultComponentList, a member of class TBasePageItemsProducer.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetProducerTemplateFileName (see page 15276)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.
	GetProducerTemplateStream (see page 15276)	This is GetProducerTemplateStream, a member of class TBasePageProducer.

	GetScriptEngine (see page 15277)	This is GetScriptEngine, a member of class TBasePageProducer.
	GetTagID (see page 15277)	This is GetTagID, a member of class TBasePageProducer.
	GetTemplateFileName (see page 15277)	This is GetTemplateFileName, a member of class TBasePageProducer.
	GetTemplateStream (see page 15277)	This is GetTemplateStream, a member of class TBasePageProducer.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleScriptTag (see page 15277)	This is HandleScriptTag, a member of class TBasePageProducer.
	HandleTag (see page 5620)	This is HandleTag, a member of class TComponentsPageProducer.
	HandleTag (see page 15278)	This is HandleTag, a member of class TBasePageProducer.
	ImplHandleTag (see page 15278)	This is ImplHandleTag, a member of class TBasePageProducer.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 15108)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ServerScriptFromStream (see page 15278)	This is ServerScriptFromStream, a member of class TBasePageProducer.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 18382)	This is SetChildOrder, a member of class TBasePageItemsProducer.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UseScriptEngine (see page 15278)	This is UseScriptEngine, a member of class TBasePageProducer.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Content (see page 15106)	Returns the content produced by the content producer.
V	Content (see page 15274)	Returns the content produced by the content producer.
V	ContentFromStream (see page 15107)	Returns the content produced in response to input read from a stream.
V	ContentFromStream (see page 15275)	Returns the content produced in response to input read from a stream.
V	ContentFromString (see page 15107)	Returns the content produced in response to an input string.
V	ContentFromString (see page 15275)	Returns the content produced in response to an input string.
V	Create (see page 18380)	Creates an instance of a TBasePageItemsProducer (see page 18376).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 18381)	Destroys the instance of the TBasePageItemsProducer (see page 18376).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProducerTemplateFileName (see page 15320)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.
	GetProducerTemplateStream (see page 15320)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ScriptEngine (see page 15279)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 15279)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebModuleContext (see page 15279)	Provides access to the Web module context object.
R	WebPageItems (see page 18382)	Maintains a list of components that generate HTML content for the producer.

Published Events

	Name	Description
	OnHTMLTag (see page 5620)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

122.1.1.2 TBasePageItemsProducer.Create Constructor

Creates an instance of a TBasePageItemsProducer (see page 18376).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBasePageItemsProducer(TComponent * AOwner);
```

Description

Call Create to instantiate a TBasePageItemsProducer (see page 18376) component at runtime if it was not created at design time.

AOwner is the component, typically the Web module, that becomes the value of this page producer's Owner property. The Owner is responsible for freeing the TBasePageItemsProducer (see page 18376) instance.

See Also

Owner (see page 2807)

122.1.1.3 TBasePageItemsProducer.Destroy Destructor

Destroys the instance of the TBasePageItemsProducer (see page 18376).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBasePageItemsProducer();
```

Description

Do not call Destroy directly. Instead call Free, which checks that the TBasePageItemsProducer (see page 18376) reference is not nil before calling Destroy.

See Also

Create (see page 18380)

Free (see page 22300)

122.1.1.4 TBasePageItemsProducer Methods

The methods of the TBasePageItemsProducer class are listed here.

Protected Methods

	Name	Description
	GetChildren (see page 18381)	This is GetChildren, a member of class TBasePageItemsProducer.
	GetComponentList (see page 18382)	This is GetComponentList, a member of class TBasePageItemsProducer.
	GetDefaultComponentList (see page 18382)	This is GetDefaultComponentList, a member of class TBasePageItemsProducer.
	SetChildOrder (see page 18382)	This is SetChildOrder, a member of class TBasePageItemsProducer.

122.1.1.4.1 TBasePageItemsProducer.GetChildren Method

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

This is GetChildren, a member of class TBasePageItemsProducer.

122.1.1.4.2 TBasePageItemsProducer.GetComponentList Method

Pascal

```
function GetComponentList: TObject;
```

C++

```
__fastcall TObject * GetComponentList();
```

Description

This is GetComponentList, a member of class TBasePageItemsProducer.

122.1.1.4.3 TBasePageItemsProducer.GetDefaultComponentList Method

Pascal

```
function GetDefaultComponentList: TObject;
```

C++

```
__fastcall TObject * GetDefaultComponentList();
```

Description

This is GetDefaultComponentList, a member of class TBasePageItemsProducer.

122.1.1.4.4 TBasePageItemsProducer.SetChildOrder Method

Pascal

```
procedure SetChildOrder(Component: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Component, int Order);
```

Description

This is SetChildOrder, a member of class TBasePageItemsProducer.

122.1.1.5 TBasePageItemsProducer Properties

The properties of the TBasePageItemsProducer class are listed here.

Public Properties

	Name	Description
R	WebPageItems (see page 18382)	Maintains a list of components that generate HTML content for the producer.

122.1.1.5.1 TBasePageItemsProducer.WebPageItems Property

Maintains a list of components that generate HTML content for the producer.

Pascal

```
property WebPageItems: TWebComponentList;
```

C++

```
__property TWebComponentList WebPageItems;
```

Description

WebPageItems lists the components that generate content for the page items producer. Each of these components supports the IWebContent interface, and can contain its own list of Web items that help it generate its content.

The type of items that can be added to the WebPageItems list are restricted by a protected method that is abstract or, in C++ terminology, pure virtual in TBasePageItemsProducer (see page 18376). Descendant classes implement this protected method to indicate what types of items can be added to the list.

At design time, use the Web page editor to add components to the WebPageItems property.

Note: Typically, WebPageItems lists all the components that generate content for the page items producer, in the order that the generated content appears in the final HTML document. However, the page items producer can transform COMPONENT tags in its template using components that are not in this list, as long as they have the same owner as the page items producer..

See Also

IWebContent (see page 24792)

Creating Web Pages with an InternetExpress Page Producer

122.1.2 TPageItemsProducer Class

TPageItemsProducer maintains a list of Web item components for producing a string of HTML commands based on an input template.

Class Hierarchy

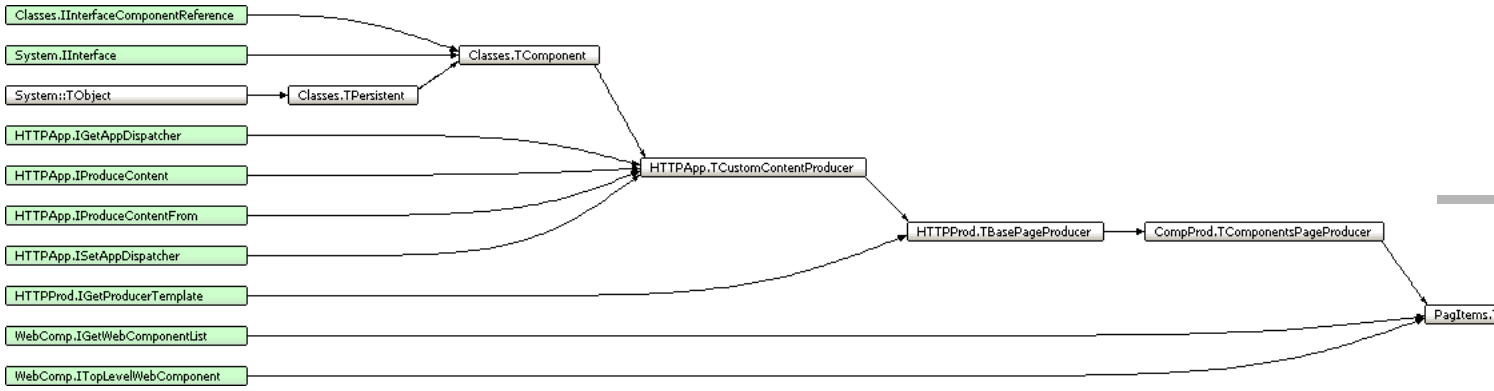

Pascal

```
TPageItemsProducer = class(TBasePageItemsProducer);
```

C++

```
class TPageItemsProducer : public TBasePageItemsProducer;
```

File

PagItems

Description

Use TPageItemsProducer as the base class for components that convert an HTML template that includes content generated by other components into a string of HTML commands that can be interpreted by a client application such as a Web browser. HTML templates include HTML commands and HTML-transparent tags that are replaced with customized content by the page items

producer in an OnHTMLTag event handler.

Page item producers differ from other page producer components in that they replace special HTML-transparent tags of the format

```
<#COMPONENT Name=WebContentComponent>
```

with the HTML produced by the component named WebContentComponent. WebContentComponent must have the same Owner as the page item producer and support the IWebContent interface, which the page item producer uses to obtain the replacement HTML for the COMPONENT tag.

TPageItemsProducer introduces the WebPageItems (see page 18382) property to maintain a hierarchical list of these Web content components, where each component produces content for its parent component.

See Also

IWebContent (see page 24792)

TInetXPageProducer (see page 17145)

122.1.2.1 TPageItemsProducer Members

The following tables list the members exposed by TPageItemsProducer.

Protected Events

	Name	Description
	OnHTMLTag (see page 15280)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoTagEvent (see page 5618)	This is DoTagEvent, a member of class TComponentsPageProducer.
	DoTagEvent (see page 15276)	This is DoTagEvent, a member of class TBasePageProducer.
V	FindComponent (see page 5619)	This is FindComponent, a member of class TComponentsPageProducer.
	FindComponentName (see page 5619)	This is FindComponentName, a member of class TComponentsPageProducer.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 18381)	This is GetChildren, a member of class TBasePageItemsProducer.
V	GetComponentContent (see page 5619)	This is GetComponentContent, a member of class TComponentsPageProducer.
	GetComponentList (see page 18382)	This is GetComponentList, a member of class TBasePageItemsProducer.
V	GetContentOptions (see page 5619)	This is GetContentOptions, a member of class TComponentsPageProducer.
	GetDefaultComponentList (see page 18382)	This is GetDefaultComponentList, a member of class TBasePageItemsProducer.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetProducerTemplateFileName (see page 15276)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.
	GetProducerTemplateStream (see page 15276)	This is GetProducerTemplateStream, a member of class TBasePageProducer.
V	GetScriptEngine (see page 15277)	This is GetScriptEngine, a member of class TBasePageProducer.
	GetTagID (see page 15277)	This is GetTagID, a member of class TBasePageProducer.
V	GetTemplateFileName (see page 15277)	This is GetTemplateFileName, a member of class TBasePageProducer.
V	GetTemplateStream (see page 18388)	This is GetTemplateStream, a member of class TPageItemsProducer.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleScriptTag (see page 15277)	This is HandleScriptTag, a member of class TBasePageProducer.
V	HandleTag (see page 5620)	This is HandleTag, a member of class TComponentsPageProducer.
V	HandleTag (see page 15278)	This is HandleTag, a member of class TBasePageProducer.
	ImplHandleTag (see page 15278)	This is ImplHandleTag, a member of class TBasePageProducer.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 15108)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ServerScriptFromStream (see page 15278)	This is ServerScriptFromStream, a member of class TBasePageProducer.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 18382)	This is SetChildOrder, a member of class TBasePageItemsProducer.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
V	UseScriptEngine (see page 15278)	This is UseScriptEngine, a member of class TBasePageProducer.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.

	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Content (see page 15106)	Returns the content produced by the content producer.
V	Content (see page 15274)	Returns the content produced by the content producer.
V	ContentFromStream (see page 15107)	Returns the content produced in response to input read from a stream.
V	ContentFromStream (see page 15275)	Returns the content produced in response to input read from a stream.
V	ContentFromString (see page 15107)	Returns the content produced in response to an input string.
V	ContentFromString (see page 15275)	Returns the content produced in response to an input string.
V	Create (see page 18388)	Creates and initializes an instance of TPageItemsProducer (see page 18383).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 18388)	Frees an instance of TPageItemsProducer (see page 18383).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProducerTemplateFileName (see page 15320)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.
	GetProducerTemplateStream (see page 15320)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.
	HTMLDoc (see page 18389)	Specifies the HTML template used to generate the value returned by the Content method.
	HTMLFile (see page 18389)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ScriptEngine (see page 15279)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 15279)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebModuleContext (see page 15279)	Provides access to the Web module context object.
R	WebPageItems (see page 18382)	Maintains a list of components that generate HTML content for the producer.

Published Events

	Name	Description
	OnHTMLTag (see page 5620)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

122.1.2.2 TPageItemsProducer.Create Constructor

Creates and initializes an instance of TPageItemsProducer (see page 18383).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TPageItemsProducer(TComponent * AOwner);
```

Description

Call Create to instantiate a page items producer at runtime.

122.1.2.3 TPageItemsProducer.Destroy Destructor

Frees an instance of TPageItemsProducer (see page 18383).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPageItemsProducer();
```

Description

Do not call Destroy directly. Instead, call the Free method, which checks whether the TPageItemsProducer (see page 18383) reference is not nil before calling Destroy.

See Also

Free (see page 22300)

122.1.2.4 TPageItemsProducer Methods

The methods of the TPageItemsProducer class are listed here.

Protected Methods

	Name	Description
	GetTemplateStream (see page 18388)	This is GetTemplateStream, a member of class TPageItemsProducer.

122.1.2.4.1 TPageItemsProducer.GetTemplateStream Method

Pascal

```
function GetTemplateStream(out AOwned: Boolean): TStream; override;
```

C++

```
virtual __fastcall TStream GetTemplateStream(Boolean AOwned);
```

Description

This is GetTemplateStream, a member of class TPageItemsProducer.

122.1.2.5 TPageItemsProducer Properties

The properties of the TPageItemsProducer class are listed here.

Public Properties

	Name	Description
	HTMLDoc (see page 18389)	Specifies the HTML template used to generate the value returned by the Content method.
	HTMLFile (see page 18389)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.

122.1.2.5.1 TPageItemsProducer.HTMLDoc Property

Specifies the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLDoc: TStrings;
```

C++

```
__property TStrings HTMLDoc;
```

Description

Set HTMLDoc to a set of strings that make up the HTML template. Each string is a sequence of one or more HTML commands or HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLDoc clears the HTMLFile (see page 18389) property.

See Also

Content (see page 15274)

HTMLFile (see page 18389)

122.1.2.5.2 TPageItemsProducer.HTMLFile Property

Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLFile: TFileName;
```

C++

```
__property TFileName HTMLFile;
```

Description

Set HTMLFile to the name of the file that contains an HTML template. The HTML template is a sequence of HTML commands and HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLFile clears the HTMLDoc (see page 18389) property.

Note: When distributing applications that contain page producers which use the HTMLFile property, be sure to distribute the file along with the application. Also, be sure that the value of HTMLFile does not include a hard-coded path name that will not be available on the user's disk.

See Also

Content (see page 15274)

HTMLDoc (see page 18389)

123 Printers Namespace

This is namespace Printers.

Topics

Name	Description
Classes (see page 18392)	The following table lists classes in this documentation.
Functions (see page 18412)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 18416)	The following table lists structs, records, enums in this documentation.
Types (see page 18418)	The following table lists types in this documentation.

Classes

	Name	Description
	EPrinter (see page 18392)	EPrinter is the exception class for printing errors.
	TPrinter (see page 18392)	TPrinter encapsulates the Windows printer interface.

Functions

	Name	Description
	AssignPrn (see page 18412)	Assigns a text file variable to the printer.
	Printer (see page 18412)	Returns a global instance of TPrinter (see page 18392) to manage interaction with the printer.
	SetPrinter (see page 18414)	Replaces the global instance of TPrinter (see page 18392) that manages interaction with the printer.

Structs, Records, Enums

	Name	Description
	TPrinterCapability (see page 18416)	TPrinterCapability and TPrinterCapabilities (see page 18418) specify the capabilities of a printer device driver.
	TPrinterOrientation (see page 18416)	TPrinterOrientation indicates the orientation of pages when printing.
	TPrinterState (see page 18417)	This is record Printers.TPrinterState.

Types

Name	Description
TPrinterCapabilities (see page 18418)	This is type Printers.TPrinterCapabilities.

123.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EPrinter (see page 18392)	EPrinter is the exception class for printing errors.
	TPrinter (see page 18392)	TPrinter encapsulates the Windows printer interface.

123.1.1 EPrinter Class

EPrinter is the exception class for printing errors.

Class Hierarchy

Pascal

```
EPrinter = class(Exception);
```

C++

```
class EPrinter : public Exception;
```

File

Printers

Description

EPrinter is raised when a printing error occurs. For example, if an application attempts to use a printer that doesn't exist, or if a job can't be sent to the printer for some other reason, an EPrinter exception is raised.

See Also

TPrinter (see page 18392)

123.1.2 TPrinter Class

TPrinter encapsulates the Windows printer interface.

Class Hierarchy

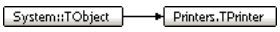

Pascal

```
TPrinter = class(TObject);
```

C++

```
class TPrinter : public TObject;
```

File

Printers

Description

Use TPrinter to manage any printing performed by an application. Obtain an instance of TPrinter by calling the Printer (see page 18412) function in the Printers (see page 18409) unit.

To determine how the printed image of the form appears, use the PrintScale property of a TForm component.

A print job is started by a call to BeginDoc (see page 18397). The application sends commands by rendering through a Text variable or the printer's canvas. You can move to a new page by calling the NewPage (see page 18400) method. The job stays open until the application calls EndDoc (see page 18399). If a problem occurs and you need to terminate a print job that was not sent to the printer successfully, call the Abort (see page 18395) method.

Use the TPrinter properties to configure the print job. For example, the title displayed in the Print Manager (and on network header pages) is determined by the Title (see page 18411) property. Copies (see page 18405) determines the number of copies to print, and Orientation (see page 18406) lets you specify whether to print in portrait or landscape mode.

TPrinter includes several read-only properties as well, that let you determine which page is currently being printed, the fonts available on the printer, the paper size, and so on.

When creating a TPrinter descendant, you must call the SetPrinter (see page 18402) routine in order for the TPrinter descendant object to work correctly.

See Also

Printer (see page 18412)

TCustomForm (see page 13849)

SetPrinter Routine

123.1.2.1 TPrinter Members

The following tables list the members exposed by TPrinter.

Public Methods

	Name	Description
	Abort (see page 18395)	Terminates the printing of a print job, dropping all unprinted data.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDoc (see page 18397)	Sends a print job to the printer.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 18394)	Creates a printer.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 18395)	Destroys an instance of a printer.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndDoc (see page 18399)	Ends the current print job and closes the text file variable.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetPrinter (see page 18400)	Retrieves information about the current printer.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NewPage (see page 18400)	Starts a new page and increments the PageNumber (see page 18408) property.
	Refresh (see page 18401)	Updates the Fonts (see page 18405) and Printers (see page 18409) properties.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetPrinter (see page 18402)	Specifies the current printer.

Public Properties

	Name	Description
R	Aborted (see page 18403)	Determines if the user aborted the print job.
R	Canvas (see page 18404)	Represents the surface of the currently printing page.
R	Capabilities (see page 18404)	Indicates the current settings of a printer device driver.
	Copies (see page 18405)	Specifies the number of copies to print.
R	Fonts (see page 18405)	Lists the fonts supported by the current printer.
R	Handle (see page 18405)	Provides access to the handle of the printer object.
	Orientation (see page 18406)	Determines if the print job prints vertically or horizontally on a page.
R	PageHeight (see page 18407)	Indicates the height (in pixels) of the currently printing page.
R	PageNumber (see page 18408)	Indicates the number of the page currently printing.
R	PageWidth (see page 18408)	Indicates the value of width (in pixels) of the currently printing page.
	PrinterIndex (see page 18409)	Specifies which printer listed in the Printers (see page 18409) property is the currently selected printer.
R	Printers (see page 18409)	Lists all printers installed in Windows.
R	Printing (see page 18409)	Indicates whether a print job is printing.
	Title (see page 18411)	Determines the text that is listed in the Print Manager and on network header pages.

123.1.2.2 TPrinter.Create Constructor

Creates a printer.

Pascal

```
constructor Create;
```

C++

```
__fastcall TPrinter();
```

Description

Never call Create directly. Create is called automatically by the Printer (see page 18412) function. Create allocates memory for a printer and calls the inherited Create. Then it sets the correct driver, device, and port.

See Also

Create (see page 22287)

123.1.2.3 TPrinter.Destroy Destructor

Destroys an instance of a printer.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TPrinter();
```

Description

Do not call Destroy directly. The printer object is automatically created and destroyed.

When Destroy is called, it verifies that any currently printing job has ended, then frees resources such as the handle, fonts, and canvas, closes the printer, and calls the inherited Destroy method.

See Also

Destroy (see page 22287)

Free (see page 22300)

123.1.2.4 TPrinter Methods

The methods of the TPrinter class are listed here.

Public Methods

	Name	Description
	Abort (see page 18395)	Terminates the printing of a print job, dropping all unprinted data.
	BeginDoc (see page 18397)	Sends a print job to the printer.
	EndDoc (see page 18399)	Ends the current print job and closes the text file variable.
	GetPrinter (see page 18400)	Retrieves information about the current printer.
	NewPage (see page 18400)	Starts a new page and increments the PageNumber (see page 18408) property.
	Refresh (see page 18401)	Updates the Fonts (see page 18405) and Printers (see page 18409) properties.
	SetPrinter (see page 18402)	Specifies the current printer.

123.1.2.4.1 TPrinter.Abort Method

Terminates the printing of a print job, dropping all unprinted data.

Pascal

```
procedure Abort;
```

C++

```
__fastcall Abort();
```

Description

Use Abort to terminate the print job before it completes; otherwise, use the EndDoc (see page 18399) method.

When Abort is called, the device is then set for the next print job.

Delphi Examples:

```
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

Aborted (see page 18403)

BeginDoc (see page 18397)

EndDoc (see page 18399)

Printing (see page 18409)

123.1.2.4.2 TPrinter.BeginDoc Method

Sends a print job to the printer.

Pascal

```
procedure BeginDoc;
```

C++

```
__fastcall BeginDoc();
```

Description

Call BeginDoc to initiate a print job. If the print job is sent successfully, the application calls EndDoc (see page 18399) to end the print job. The print job won't actually start printing until EndDoc (see page 18399) is called.

C++ Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PrintDialog1->Options.Clear();
  PrintDialog1->Options << poPageNums << poSelection;
  PrintDialog1->FromPage = 1;
  PrintDialog1->MinPage = 1;
  PrintDialog1->ToPage = PageControl1->PageCount;
  PrintDialog1->MaxPage = PageControl1->PageCount;
  if (PrintDialog1->Execute())
  {
    int Start, Stop;
    // determine the range the user wants to print
    switch (PrintDialog1->PrintRange)
    {
      case prSelection:
        Start = PageControl1->ActivePage->PageIndex;
        Stop = Start;
        break;
      case prPageNums:
        Start = PrintDialog1->FromPage - 1;
        Stop =  PrintDialog1->ToPage - 1;
        break;
      default:  // prAllPages
        Start = PrintDialog1->MinPage - 1;
        Stop = PrintDialog1->MaxPage - 1;
        break;
    }
    // now, print the pages
    Printer()->BeginDoc();
    for (int i = Start; i <= Stop; i++)
    {
      PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
      if (i != Stop)
        Printer()->NewPage();
    }
```

```
    Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
}
procedure TForm1.Button1Click(Sender:TObject);
var
  I, Start, Stop: Integer;
begin
PrintDialog1.Options := [poPageNums, poSelection];
PrintDialog1.FromPage := 1;
PrintDialog1.MinPage := 1;
PrintDialog1.ToPage := PageControl1.PageCount;
PrintDialog1.MaxPage := PageControl1.PageCount;

if PrintDialog1.Execute() then
  begin
    { determine the range the user wants to print }
    with PrintDialog1 do
    begin
      if PrintRange = prAllPages then
        begin
        Start := MinPage - 1;
        Stop := MaxPage - 1;
      end
      else if PrintRange = prSelection then
      begin
        Start := PageControl1.ActivePage.PageIndex;
        Stop := Start;
        end
      else  { PrintRange = prPageNums }
      begin
        Start := FromPage - 1;
        Stop := ToPage - 1;
      end;
    end;
    { now, print the pages }
    with Printer do
    begin
      BeginDoc;
      for I := Start to Stop do
      begin
        PageControl1.Pages[I].PaintTo(Handle, 10, 10);
        if I <> Stop then
          NewPage;
      end;
      EndDoc;
    end;
  end;
end;
```

See Also

Abort (see page 18395)

EndDoc (see page 18399)

Printing (see page 18409)

123.1.2.4.3 TPrinter.EndDoc Method

Ends the current print job and closes the text file variable.

Pascal

```
procedure EndDoc;
```

C++

```
__fastcall EndDoc();
```

Description

EndDoc terminates the print job (and closes the currently open Text). The print job will begin printing on the printer after a call to EndDoc.

After the application calls EndDoc, the printer begins printing. Use EndDoc after successfully sending a print job to the printer. If the print job isn't successful, use the Abort (see page 18395) method.

C++ Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PrintDialog1->Options.Clear();
  PrintDialog1->Options << poPageNums << poSelection;
  PrintDialog1->FromPage = 1;
  PrintDialog1->MinPage = 1;
  PrintDialog1->ToPage = PageControl1->PageCount;
  PrintDialog1->MaxPage = PageControl1->PageCount;
  if (PrintDialog1->Execute())
  {
    int Start, Stop;
    // determine the range the user wants to print
    switch (PrintDialog1->PrintRange)
    {
      case prSelection:
        Start = PageControl1->ActivePage->PageIndex;
        Stop = Start;
        break;
      case prPageNums:
        Start = PrintDialog1->FromPage - 1;
        Stop =  PrintDialog1->ToPage - 1;
        break;
      default:  // prAllPages
        Start = PrintDialog1->MinPage - 1;
        Stop = PrintDialog1->MaxPage - 1;
        break;
    }
    // now, print the pages
    Printer()->BeginDoc();
    for (int i = Start; i <= Stop; i++)
    {
      PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
```

```
        if (i != Stop)
          Printer()->NewPage();
      }
      Printer()->EndDoc();
    }
  }
```

See Also

Abort (see page 18395)

BeginDoc (see page 18397)

Printing (see page 18409)

123.1.2.4.4 TPrinter.GetPrinter Method

Retrieves information about the current printer.

Pascal

```
procedure GetPrinter(ADevice: PChar; ADriver: PChar; APort: PChar; var ADeviceMode: THandle);
```

C++

```
__fastcall GetPrinter(const char * ADevice, const char * ADriver, const char * APort, THandle
ADeviceMode);
```

Description

Call GetPrinter to obtain information about the current printer.

ADevice is a buffer that receives the device name of the current printer. This is the name that appears in the Print Manager (for example, "Epson FX-80"). It is not the printer model name.

ADriver is a buffer that receives the driver name for the current printer. Some versions of Windows do not use printer driver names, in which case this buffer is unused. On versions of Windows that support printer driver names, this is the name of a printer driver (usually 'WINSPOOL').

APort is a buffer that receives the port used by the current printer. It is not used on 32-bit versions of Windows, but is included for backward compatibility.

DeviceMode returns the handle to a memory buffer containing device-specific initialization data for the device driver. Use the GlobalLock function to convert this handle into a pointer to a DeviceMode record (Delphi) or DEVMODE structure (C++).

See Also

Printers (see page 18409)

123.1.2.4.5 TPrinter.NewPage Method

Starts a new page and increments the PageNumber (see page 18408) property.

Pascal

```
procedure NewPage;
```

C++

```
__fastcall NewPage();
```

Description

NewPage forces the current print job to begin printing on a new page in the printer.

NewPage increments the value of the PageNumber (see page 18408) property. It also resets the value of the Pen property of

the Canvas (see page 18404) back to (0, 0).

C++ Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PrintDialog1->Options.Clear();
  PrintDialog1->Options << poPageNums << poSelection;
  PrintDialog1->FromPage = 1;
  PrintDialog1->MinPage = 1;
  PrintDialog1->ToPage = PageControl1->PageCount;
  PrintDialog1->MaxPage = PageControl1->PageCount;
  if (PrintDialog1->Execute())
  {
    int Start, Stop;
    // determine the range the user wants to print
    switch (PrintDialog1->PrintRange)
    {
      case prSelection:
        Start = PageControl1->ActivePage->PageIndex;
        Stop = Start;
        break;
      case prPageNums:
        Start = PrintDialog1->FromPage - 1;
        Stop =  PrintDialog1->ToPage - 1;
        break;
      default:  // prAllPages
        Start = PrintDialog1->MinPage - 1;
        Stop = PrintDialog1->MaxPage - 1;
        break;
    }
    // now, print the pages
    Printer()->BeginDoc();
    for (int i = Start; i <= Stop; i++)
    {
      PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
      if (i != Stop)
        Printer()->NewPage();
    }
    Printer()->EndDoc();
  }
}
```

See Also

PageNumber (see page 18408)

123.1.2.4.6 TPrinter.Refresh Method

Updates the Fonts (see page 18405) and Printers (see page 18409) properties.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh when the installed fonts or printers on the current system may have changed. Refresh clears the internal cache of installed fonts and printers, so that the Fonts (see page 18405) and Printers (see page 18409) properties are updated when they are next read.

See Also

Fonts (see page 18405)

Printers (see page 18409)

123.1.2.4.7 TPrinter.SetPrinter Method

Specifies the current printer.

Pascal

```
procedure SetPrinter(ADevice: PChar; ADriver: PChar; APort: PChar; ADeviceMode: THandle);
```

C++

```
__fastcall SetPrinter(const char * ADevice, const char * ADriver, const char * APort, THandle
ADeviceMode);
```

Description

Call SetPrinter to change the current printer.

ADevice specifies the device name of the new printer. This is the name that appears in the Print Manager (for example, "Epson FX-80"). It is not the printer model name.

ADriver specifies the driver name for the new printer (usually 'WINSPOOL').

APort is included for backward compatibility. It should be set to an empty string.

DeviceMode is the handle to a memory buffer device-specific initialization data for the device driver. This handle can be obtained by calling the GlobalAlloc function to allocate a DeviceMode record (Delphi) or DEVMODE structure (C++).

See Also

Printers (see page 18409)

123.1.2.5 TPrinter Properties

The properties of the TPrinter class are listed here.

Public Properties

	Name	Description
R	Aborted (see page 18403)	Determines if the user aborted the print job.
R	Canvas (see page 18404)	Represents the surface of the currently printing page.
R	Capabilities (see page 18404)	Indicates the current settings of a printer device driver.
	Copies (see page 18405)	Specifies the number of copies to print.
R	Fonts (see page 18405)	Lists the fonts supported by the current printer.
R	Handle (see page 18405)	Provides access to the handle of the printer object.
	Orientation (see page 18406)	Determines if the print job prints vertically or horizontally on a page.
R	PageHeight (see page 18407)	Indicates the height (in pixels) of the currently printing page.

R	PageNumber (see page 18408)	Indicates the number of the page currently printing.
R	PageWidth (see page 18408)	Indicates the value of width (in pixels) of the currently printing page.
	PrinterIndex (see page 18409)	Specifies which printer listed in the Printers (see page 18409) property is the currently selected printer.
R	Printers (see page 18409)	Lists all printers installed in Windows.
R	Printing (see page 18409)	Indicates whether a print job is printing.
	Title (see page 18411)	Determines the text that is listed in the Print Manager and on network header pages.

123.1.2.5.1 TPrinter.Aborted Property

Determines if the user aborted the print job.

Pascal

```
property Aborted: Boolean;
```

C++

```
__property Boolean Aborted;
```

Description

If Aborted is true, the print job was aborted. If it is false, the user did not abort the print job.

The Abort (see page 18395) method sets the Aborted property to true.

Delphi Examples:

```
{
The following code displays a dialog box if the print job
was aborted.  This code aborts a print job if the user
presses Esc. Note that you should set KeyPreview to true
to ensure that the OnKeyDown event handler of Form1 is called.
Also notice that the Button1Click routine continues to run
even after the print job has been aborted.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
```

```
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
    if Printer.Aborted then
      MessageDlg('The print job did not finish printing', mtInformation, [mbOK], 0);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
  //  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

Abort (see page 18395)

123.1.2.5.2 TPrinter.Canvas Property

Represents the surface of the currently printing page.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Canvas represents the printing surface of a page. Use the Brush, Font, and Pen properties of the Canvas object to determine how drawing or text appears on the page.

Note:

See Also

Brush (see page 14484)

CopyRect (see page 14452)

Draw (see page 14454)

Font (see page 14488)

Pen (see page 14492)

StretchDraw (see page 14472)

Handle (see page 14489)

123.1.2.5.3 TPrinter.Capabilities Property

Indicates the current settings of a printer device driver.

Pascal

```
property Capabilities: TPrinterCapabilities;
```

C++

```
__property TPrinterCapabilities Capabilities;
```

Description

Capabilities contains the capabilities for the printer, indicating the orientation, number of copies, and whether or not collated is selected.

123.1.2.5.4 TPrinter.Copies Property

Specifies the number of copies to print.

Pascal

```
property Copies: Integer;
```

C++

```
__property int Copies;
```

Description

Set Copies to indicate the number of copies to print.

Copies uses the device mode to determine the number of copies to print.

123.1.2.5.5 TPrinter.Fonts Property

Lists the fonts supported by the current printer.

Pascal

```
property Fonts: TStrings;
```

C++

```
__property TStrings Fonts;
```

Description

Read Fonts to get a list of the fonts supported by the printer. Each string in Fonts is the name of a typeface.

Fonts contains TrueType fonts even if the printer doesn't support them natively because the Windows Graphics (see page 14387) Device Interface (GDI) can draw TrueType fonts accurately when a print job uses them.

See Also

Name (see page 14510)

123.1.2.5.6 TPrinter.Handle Property

Provides access to the handle of the printer object.

Pascal

```
property Handle: HDC;
```

C++

```
__property HDC Handle;
```

Description

Use Handle to gain access to the handle of the printer object.

C++ Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PrintDialog1->Options.Clear();
  PrintDialog1->Options << poPageNums << poSelection;
  PrintDialog1->FromPage = 1;
  PrintDialog1->MinPage = 1;
  PrintDialog1->ToPage = PageControl1->PageCount;
  PrintDialog1->MaxPage = PageControl1->PageCount;
  if (PrintDialog1->Execute())
  {
    int Start, Stop;
    // determine the range the user wants to print
    switch (PrintDialog1->PrintRange)
    {
      case prSelection:
        Start = PageControl1->ActivePage->PageIndex;
        Stop = Start;
        break;
      case prPageNums:
        Start = PrintDialog1->FromPage - 1;
        Stop =  PrintDialog1->ToPage - 1;
        break;
      default:  // prAllPages
        Start = PrintDialog1->MinPage - 1;
        Stop = PrintDialog1->MaxPage - 1;
        break;
    }
    // now, print the pages
    Printer()->BeginDoc();
    for (int i = Start; i <= Stop; i++)
    {
      PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
      if (i != Stop)
        Printer()->NewPage();
    }
    Printer()->EndDoc();
  }
}
```

123.1.2.5.7 TPrinter.Orientation Property

Determines if the print job prints vertically or horizontally on a page.

Pascal

```
property Orientation: TPrinterOrientation;
```

C++

```
__property TPrinterOrientation Orientation;
```

Description

Use Orientation to determine if a print job prints in landscape or portrait.

123.1.2.5.8 TPrinter.PageHeight Property

Indicates the height (in pixels) of the currently printing page.

Pascal

```
property PageHeight: Integer;
```

C++

```
__property int PageHeight;
```

Description

Use PageHeight to find the height in pixels of the currently printing page.

Delphi Examples:

```
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
```

```
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

PageNumber (see page 18408)

PageWidth (see page 18408)

123.1.2.5.9 TPrinter.PageNumber Property

Indicates the number of the page currently printing.

Pascal

```
property PageNumber: Integer;
```

C++

```
__property int PageNumber;
```

Description

Use PageNumber to find the page number of the currently printing page.

PageNumber is incremented whenever the NewPage (see page 18400) method is called. It is also incremented when a Text variable is written and a CR is encountered on the last line of the page.

See Also

NewPage (see page 18400)

PageHeight (see page 18407)

PageWidth (see page 18408)

123.1.2.5.10 TPrinter.PageWidth Property

Indicates the value of width (in pixels) of the currently printing page.

Pascal

```
property PageWidth: Integer;
```

C++

```
__property int PageWidth;
```

Description

Use PageWidth to find the width in pixels of the currently printing page.

See Also

PageHeight (see page 18407)

PageNumber (see page 18408)

123.1.2.5.11 TPrinter.PrinterIndex Property

Specifies which printer listed in the Printers (see page 18409) property is the currently selected printer.

Pascal

```
property PrinterIndex: Integer;
```

C++

```
__property int PrinterIndex;
```

Description

The value of PrinterIndex is the currently selected printer. If this value is changed, EndDoc (see page 18399) is called automatically. To select the default printer, set the value of PrinterIndex to -1.

The list of installed printers is found in the Printers (see page 18409) property. The list of fonts supported by the current printer is found in the Fonts (see page 18405) property.

See Also

EndDoc (see page 18399)

Fonts (see page 18405)

Printers (see page 18409)

123.1.2.5.12 TPrinter.Printers Property

Lists all printers installed in Windows.

Pascal

```
property Printers: TStrings;
```

C++

```
__property TStrings Printers;
```

Description

The list of installed printers is found in the Printers property. The value of the PrinterIndex (see page 18409) property is the currently selected printer. The list of fonts supported by the current printer is found in the Fonts (see page 18405) property.

See Also

GetPrinter (see page 18400)

123.1.2.5.13 TPrinter.Printing Property

Indicates whether a print job is printing.

Pascal

```
property Printing: Boolean;
```

C++

```
__property Boolean Printing;
```

Description

Use Printing to determine if a job is printing.

Printing is true when the application has called the BeginDoc (see page 18397) method but the EndDoc (see page 18399)

method (or the Abort (see page 18395) method) hasn't yet been called.

Delphi Examples:

```
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

Abort (see page 18395)

BeginDoc (see page 18397)

EndDoc (see page 18399)

123.1.2.5.14 TPrinter.Title Property

Determines the text that is listed in the Print Manager and on network header pages.

Pascal

```
property Title: string;
```

C++

```
__property AnsiString Title;
```

Description

Title is used by Windows in the Print Manager and for network title pages.

123.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AssignPrn (see page 18412)	Assigns a text file variable to the printer.
	Printer (see page 18412)	Returns a global instance of TPrinter (see page 18392) to manage interaction with the printer.
	SetPrinter (see page 18414)	Replaces the global instance of TPrinter (see page 18392) that manages interaction with the printer.

123.2.1 Printers.AssignPrn Function

Assigns a text file variable to the printer.

Pascal

```
procedure AssignPrn(var F: Text);
```

C++

```
AssignPrn(Text F);
```

File

Printers

Description

In Delphi code, Call AssignPrn to cause all information written to the text file variable F to be directed to the printer. After the variable is assigned, the application must call the Rewrite procedure.

See Also

Rewrite

123.2.2 Printers.Printer Function

Returns a global instance of TPrinter (see page 18392) to manage interaction with the printer.

Pascal

```
function Printer: TPrinter;
```

C++

```
TPrinter Printer();
```

File

Printers

Description

Printer creates an instance of a global TPrinter (see page 18392) object the first time it is called. Use the Printer function to

print using the TPrinter (see page 18392) object.

Printer is declared in the Printers (see page 18391) unit. To use the Printer function and the TPrinter (see page 18392) object, add Printers (see page 18391) to the uses clause of your unit (Delphi) or include Printers.hpp in the source file (C++).

C++ Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PrintDialog1->Options.Clear();
  PrintDialog1->Options << poPageNums << poSelection;
  PrintDialog1->FromPage = 1;
  PrintDialog1->MinPage = 1;
  PrintDialog1->ToPage = PageControl1->PageCount;
  PrintDialog1->MaxPage = PageControl1->PageCount;
  if (PrintDialog1->Execute())
  {
    int Start, Stop;
    // determine the range the user wants to print
    switch (PrintDialog1->PrintRange)
    {
      case prSelection:
        Start = PageControl1->ActivePage->PageIndex;
        Stop = Start;
        break;
      case prPageNums:
        Start = PrintDialog1->FromPage - 1;
        Stop =  PrintDialog1->ToPage - 1;
        break;
      default:  // prAllPages
        Start = PrintDialog1->MinPage - 1;
        Stop = PrintDialog1->MaxPage - 1;
        break;
    }
    // now, print the pages
    Printer()->BeginDoc();
    for (int i = Start; i <= Stop; i++)
    {
      PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
      if (i != Stop)
        Printer()->NewPage();
    }
    Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example uses a button, a Page Control, and a Print
dialog box on a form. When the user clicks the button, the
print dialog is displayed.  The user can select any subset
of the pages in the page control for printing.  The selected
pages are then printed.  To run this example successfully,
you must add the Printers unit to the uses clause of your
unit.  Right click on the PageControl to add at least one
page to the PageControl.
```

```
}
procedure TForm1.Button1Click(Sender:TObject);
var
  I, Start, Stop: Integer;
begin
PrintDialog1.Options := [poPageNums, poSelection];
PrintDialog1.FromPage := 1;
PrintDialog1.MinPage := 1;
PrintDialog1.ToPage := PageControl1.PageCount;
PrintDialog1.MaxPage := PageControl1.PageCount;

if PrintDialog1.Execute() then
  begin
    { determine the range the user wants to print }
    with PrintDialog1 do
    begin
      if PrintRange = prAllPages then
        begin
        Start := MinPage - 1;
        Stop := MaxPage - 1;
      end
      else if PrintRange = prSelection then
      begin
        Start := PageControl1.ActivePage.PageIndex;
        Stop := Start;
        end
      else  { PrintRange = prPageNums }
      begin
        Start := FromPage - 1;
        Stop := ToPage - 1;
      end;
    end;
    { now, print the pages }
    with Printer do
    begin
      BeginDoc;
      for I := Start to Stop do
      begin
        PageControl1.Pages[I].PaintTo(Handle, 10, 10);
        if I <> Stop then
          NewPage;
      end;
      EndDoc;
    end;
  end;
end;
```

See Also

SetPrinter (see page 18414)

123.2.3 Printers.SetPrinter Function

Replaces the global instance of TPrinter (see page 18392) that manages interaction with the printer.

Pascal

```
function SetPrinter(NewPrinter: TPrinter): TPrinter;
```

C++

```
TPrinter SetPrinter(TPrinter NewPrinter);
```

File

Printers

Description

SetPrinter replaces the global TPrinter (see page 18392) object with another TPrinter (see page 18392) object. This allows applications to change the way they handle printing by substituting a TPrinter (see page 18392) descendant for the default TPrinter (see page 18392) object.

NewPrinter is the new TPrinter (see page 18392) object that replaces the current global TPrinter (see page 18392) object.

SetPrinter returns the previous global TPrinter (see page 18392) object.

Note: The global TPrinter (see page 18392) object is freed automatically when the application shuts down. After a call to SetPrinter, the printer that is returned is not automatically freed. It is the caller's responsibility to either free the return value, or replace it using another call to SetPrinter and to free the substitute printer that the second SetPrinter call returns.

SetPrinter is declared in the Printers (see page 18391) unit. To use the SetPrinter function, add Printers (see page 18391) to the uses clause of your unit (Delphi) or include Printers.hpp in the source file (C++) .

See Also

Printer (see page 18412)

123.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TPrinterCapability (see page 18416)	TPrinterCapability and TPrinterCapabilities (see page 18418) specify the capabilities of a printer device driver.
	TPrinterOrientation (see page 18416)	TPrinterOrientation indicates the orientation of pages when printing.
	TPrinterState (see page 18417)	This is record Printers.TPrinterState.

123.3.1 Printers.TPrinterCapability Enumeration

TPrinterCapability and TPrinterCapabilities (see page 18418) specify the capabilities of a printer device driver.

Pascal

```
TPrinterCapability = (
  pcCopies,
  pcOrientation,
  pcCollation
);
```

C++

```
enum TPrinterCapability {
  pcCopies,
  pcOrientation,
  pcCollation
};
```

File

Printers

Description

TPrinterCapabilities (see page 18418) is a set of TPrinterCapability values. The possible values of TPrinterCapability are

Value	Meaning
pcCopies	The printer driver allows the user to specify the number of copies.
pcOrientation	The printer driver allows the user to specify portrait or landscape orientation.
pcCollation	The printer driver allows the user to specify whether copies should be collated.

123.3.2 Printers.TPrinterOrientation Enumeration

TPrinterOrientation indicates the orientation of pages when printing.

Pascal

```
TPrinterOrientation = (
```

```
    poPortrait,
    poLandscape
  );
```

C++

```
  enum TPrinterOrientation {
    poPortrait,
    poLandscape
  };
```

File

Printers

Description

These are the possible values for TPrinterOrientation:

Value	Meaning
poPortrait	The job prints vertically on the page.
poLandscape	The job prints horizontally on the page.

123.3.3 Printers.TPrinterState Enumeration

Pascal

```
  TPrinterState = (
    psNoHandle,
    psHandleIC,
    psHandleDC
  );
```

C++

```
  enum TPrinterState {
    psNoHandle,
    psHandleIC,
    psHandleDC
  };
```

File

Printers

Description

This is record Printers.TPrinterState.

123.4 Types

The following table lists types in this documentation.

Types

Name	Description
TPrinterCapabilities (see page 18418)	This is type Printers.TPrinterCapabilities.

123.4.1 Printers.TPrinterCapabilities Type

Pascal

```
TPrinterCapabilities = set of TPrinterCapability;
```

C++

```
TPrinterCapability TPrinterCapabilities;
```

File

Printers

Description

This is type Printers.TPrinterCapabilities.

124 Provider Namespace

This is namespace Provider.

Topics

Name	Description
Classes (see page 18421)	The following table lists classes in this documentation.
Interfaces (see page 18613)	The following table lists interfaces in this documentation.
Functions (see page 18615)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 18617)	The following table lists structs, records, enums in this documentation.
Types (see page 18622)	The following table lists types in this documentation.
Variables (see page 18628)	The following table lists variables in this documentation.
Constants (see page 18629)	The following table lists constants in this documentation.

Classes

	Name	Description
	EDSWriter (see page 18421)	EDSWriter is the exception class for errors that occur when creating data packets to represent the data or metadata in a dataset.
	TBaseProvider (see page 18423)	TBaseProvider is a base class for all providers that represent the data in a dataset.
	TCustomPacketWriter (see page 18445)	This is class Provider.TCustomPacketWriter.
	TCustomProvider (see page 18449)	TCustomProvider is the base class for data providers.
	TCustomResolver (see page 18483)	TCustomResolver is the base class for components that apply updates on behalf of a provider.
	TDataPacketWriter (see page 18497)	This is class Provider.TDataPacketWriter.
	TDataSetProvider (see page 18508)	TDataSetProvider provides data from a dataset and resolves updates to that dataset or its database server.
	TDataSetResolver (see page 18544)	TDataSetResolver applies updates to a dataset on behalf of a provider.
	TLocalAppServer (see page 18553)	This is class Provider.TLocalAppServer.
	TPacketDataSet (see page 18564)	TPacketDataSet represents a data packet or delta packet that is used by a provider.
	TProvider (see page 18589)	TProvider provides data to and applies updates from a client dataset.
	TSQLResolver (see page 18589)	TSQLResolver applies updates to a database server on behalf of a provider.
	TUpdateTree (see page 18600)	TUpdateTree stores information about a delta packet on behalf of a resolver.

Constants

Name	Description
MetaDataOption (see page 18629)	Specifies the options available when retrieving metadata from a target database.

ResetOption (see page 18630)	This is constant Provider.ResetOption.
XMLOption (see page 18630)	This is constant Provider.XMLOption.
XMLUTF8Option (see page 18631)	This is constant Provider.XMLUTF8Option.

Functions

	Name	Description
	GetObjectProperty (see page 18615)	This is function Provider.GetObjectProperty.
	GetStringProperty (see page 18615)	This is function Provider.GetStringProperty.
	VarArrayFromStrings (see page 18615)	Creates a Variant array that contains the strings in a TStrings object.
	VarArrayFromWideStrings (see page 18616)	This is function Provider.VarArrayFromWideStrings.

Interfaces

	Name	Description
	IProviderContainer (see page 18613)	This is class Provider.IProviderContainer.

Structs, Records, Enums

	Name	Description
	TGetRecordOption (see page 18617)	TGetRecordOption and TGetRecordOptions (see page 18624) influence what type of information is added to data packets.
	TProviderOption (see page 18618)	TProviderOption and TProviderOptions (see page 18626) influence what type of information is communicated between client datasets and provider components.
	TPutFieldInfo (see page 18620)	This is record Provider.TPutFieldInfo.
	TResolverResponse (see page 18620)	TResolverResponse indicates the action a resolver component should take after an OnUpdateError event handler handles an update error.

Types

Name	Description
PPutFieldInfo (see page 18622)	This is type Provider.PPutFieldInfo.
TAfterUpdateRecordEvent (see page 18622)	This is type Provider.TAfterUpdateRecordEvent.
TBeforeCommitEvent (see page 18623)	This is type Provider.TBeforeCommitEvent.
TBeforeUpdateRecordEvent (see page 18623)	This is type Provider.TBeforeUpdateRecordEvent.
TDataRequestEvent (see page 18623)	This is type Provider.TDataRequestEvent.
TGetDSProps (see page 18624)	This is type Provider.TGetDSProps.
TGetParamsEvent (see page 18624)	This is type Provider.TGetParamsEvent.
TGetRecordOptions (see page 18624)	This is type Provider.TGetRecordOptions.
TGetTableNameEvent (see page 18625)	This is type Provider.TGetTableNameEvent.
TInfoArray (see page 18625)	This is type Provider.TInfoArray.
TOnValidate (see page 18625)	This is type Provider.TOnValidate.
TProviderDataEvent (see page 18626)	This is type Provider.TProviderDataEvent.
TProviderOptions (see page 18626)	This is type Provider.TProviderOptions.
TPutFieldProc (see page 18626)	This is type Provider.TPutFieldProc.
TResolverErrorEvent (see page 18627)	This is type Provider.TResolverErrorEvent.

Variables

Name	Description
InformixLob (see page 18628)	This is variable Provider.InformixLob.

124.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EDSWriter (see page 18421)	EDSWriter is the exception class for errors that occur when creating data packets to represent the data or metadata in a dataset.
	TBaseProvider (see page 18423)	TBaseProvider is a base class for all providers that represent the data in a dataset.
	TCustomPacketWriter (see page 18445)	This is class Provider.TCustomPacketWriter.
	TCustomProvider (see page 18449)	TCustomProvider is the base class for data providers.
	TCustomResolver (see page 18483)	TCustomResolver is the base class for components that apply updates on behalf of a provider.
	TDataPacketWriter (see page 18497)	This is class Provider.TDataPacketWriter.
	TDataSetProvider (see page 18508)	TDataSetProvider provides data from a dataset and resolves updates to that dataset or its database server.
	TDataSetResolver (see page 18544)	TDataSetResolver applies updates to a dataset on behalf of a provider.
	TLocalAppServer (see page 18553)	This is class Provider.TLocalAppServer.
	TPacketDataSet (see page 18564)	TPacketDataSet represents a data packet or delta packet that is used by a provider.
	TProvider (see page 18589)	TProvider provides data to and applies updates from a client dataset.
	TSQLResolver (see page 18589)	TSQLResolver applies updates to a database server on behalf of a provider.
	TUpdateTree (see page 18600)	TUpdateTree stores information about a delta packet on behalf of a resolver.

124.1.1 EDSWriter Class

EDSWriter is the exception class for errors that occur when creating data packets to represent the data or metadata in a dataset.

Class Hierarchy

Pascal

```
EDSWriter = class(Exception);
```

C++

```
class EDSWriter : public Exception;
```

File

Provider

Description

EDSWriter is raised whenever a provider component encounters an error while trying to create a data packet from a dataset.

See Also

TDataSetProvider (see page 18508)

TCustomProvider (see page 18449)

124.1.1.1 EDSWriter Members

The following tables list the members exposed by EDSWriter.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 18423)	Creates an instance of EDSWriter (see page 18421) using a specified error code and message string.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ErrorCode (see page 18423)	Indicates the error code that this exception represents.
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.

124.1.1.2 EDSWriter.Create Constructor

Creates an instance of EDSWriter (see page 18421) using a specified error code and message string.

Pascal

```
constructor Create(ErrMsg: string; Status: Integer);
```

C++

```
__fastcall EDSWriter(AnsiString ErrMsg, int Status);
```

Description

Create instantiates EDSWriter (see page 18421), setting the Message property to ErrMsg and the ErrorCode (see page 18423) property to Status.

See Also

Create (see page 22752)

ErrorCode (see page 18423)

124.1.1.3 EDSWriter Properties

The properties of the EDSWriter class are listed here.

Public Properties

	Name	Description
R	ErrorCode (see page 18423)	Indicates the error code that this exception represents.

124.1.1.3.1 EDSWriter.ErrorCode Property

Indicates the error code that this exception represents.

Pascal

```
property ErrorCode: Integer;
```

C++

```
__property int ErrorCode;
```

Description

Use ErrorCode to determine what error gave rise to this exception. ErrorCode is the value passed to the canonical EDSWriter (see page 18421) constructor (C++) or Create (see page 18423) method (Delphi).

See Also

Create (see page 18423)

124.1.2 TBaseProvider Class

TBaseProvider is a base class for all providers that represent the data in a dataset.

Class Hierarchy

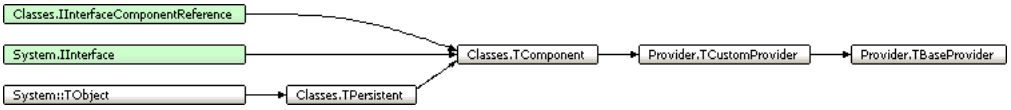

Pascal

```
TBaseProvider = class(TCustomProvider);
```

C++

```
class TBaseProvider : public TCustomProvider;
```

File

Provider

Description

TBaseProvider is the immediate ancestor of TDataSetProvider (see page 18508), which provides data from any dataset and applies updates back to that dataset or directly to an SQL server. Use TBaseProvider to derive other customized data provider objects, such as providers that apply updates directly to a database without using SQL.

Unlike its immediate ancestor (TCustomProvider (see page 18449)) TBaseProvider includes no abstract (Delphi) or pure virtual (C++) methods. However, applications should not create instances of TBaseProvider, as many of the methods are merely stubs that provide a backstop for specialized descendants that do not implement all the provider methods. To add a provider component to a form or data module, use TDataSetProvider (see page 18508) (or TXMLTransformProvider).

A provider derived from TBaseProvider is an intermediate link between a TClientDataSet or TXMLBroker object and a source dataset. The source dataset can reside in the same application as the client dataset or XML broker, or it can be part of a separate application server. The provider packages data from the source dataset and passes it in one or more transportable data packets to the client dataset or XML broker. The client dataset reconstructs the data in the data packet to create a local, in-memory copy for user access. When user access is complete, the client dataset repackages any changed data and sends the updates back to the provider. The provider applies the updates to the database or source dataset.

When the provider supplies data to a client dataset, the client dataset reconstructs the data in the data packet to create a local, in-memory copy for user access. When user access is complete, the client dataset repackages any changed data and sends the updates back to the provider. The provider applies the updates to the database or source dataset.

When the provider supplies data to an XML broker, the XML broker adds the data packet, in XML format, to an HTML document that is downloaded by a Web client. When the XML broker receives updates from the Web client, it sends the updates to the provider, which applies them to the database or source dataset.

Client datasets and XML brokers communicate with a provider through the IAppServer interface. In multi-tiered applications, this is the interface of the remote data module that contains the provider. To enable the remote data module to pass calls to the provider, the provider's Exported (see page 18474) property must be true, and its Owner property must specify the remote data module.

See Also

TClientDataSet (see page 8289)

TDataSetProvider (see page 18508)

IAppServer (see page 16450)

TCustomProvider (see page 18449)

TXMLTransformProvider (see page 26492)

TSoapDataModule (see page 19884)

124.1.2.1 TBaseProvider Members

The following tables list the members exposed by TBaseProvider.

Protected Events

	Name	Description
	AfterApplyUpdates (see page 18475)	Occurs after the provider applies updates.
	AfterExecute (see page 18476)	Occurs after the provider passes an execute command to its dataset.
	AfterGetParams (see page 18477)	This is AfterGetParams, a member of class TCustomProvider.
	AfterGetRecords (see page 18477)	Occurs after the provider creates a data packet to send to the client dataset.
	AfterRowRequest (see page 18478)	Occurs after the provider fetches new information about the current record.
	BeforeApplyUpdates (see page 18479)	Occurs before the provider applies updates from a client dataset.
	BeforeExecute (see page 18479)	This is BeforeExecute, a member of class TCustomProvider.
	BeforeGetParams (see page 18480)	Occurs before the provider creates an OleVariant that encodes the current parameter values and sends them to a client dataset.
	BeforeGetRecords (see page 18480)	Occurs before the provider creates a data packet and sends it to a client dataset.
	BeforeRowRequest (see page 18482)	Occurs before the provider creates a delta packet with information about the current record.
	OnValidate (see page 18483)	This is OnValidate, a member of class TCustomProvider.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckResolver (see page 18433)	Instantiates a resolver for the Resolver (see page 18441) property if it does not already exist.
V	CreateDataPacket (see page 18433)	Generates a data packet for the data associated with the provider.
V	CreateResolver (see page 18434)	Creates a resolver component for handling the application of updates.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterApplyUpdates (see page 18460)	Generates an AfterApplyUpdates (see page 18475) event.
V	DoAfterExecute (see page 18461)	Generates an AfterExecute (see page 18476) event.
V	DoAfterGetParams (see page 18461)	Generates an AfterGetParams (see page 18477) event.
V	DoAfterGetRecords (see page 18462)	Generates an AfterGetRecords (see page 18477) event.
V	DoAfterRowRequest (see page 18462)	Generates an AfterRowRequest (see page 18478) event.
V	DoAfterUpdateRecord (see page 18434)	Generates an AfterUpdateRecord (see page 18442) event.
V	DoApplyUpdates (see page 18463)	This is DoApplyUpdates, a member of class TCustomProvider.
V	DoBeforeApplyUpdates (see page 18463)	Generates a BeforeApplyUpdates (see page 18479) event.
V	DoBeforeCommit (see page 18435)	This is DoBeforeCommit, a member of class TBaseProvider.
V	DoBeforeExecute (see page 18464)	Generates a BeforeExecute (see page 18479) event.
V	DoBeforeGetParams (see page 18464)	Generates a BeforeGetParams (see page 18480) event.
V	DoBeforeGetRecords (see page 18465)	Generates a BeforeGetRecords (see page 18480) event.
V	DoBeforeRowRequest (see page 18465)	Generates a BeforeRowRequest (see page 18482) event.
V	DoBeforeUpdateRecord (see page 18435)	This is DoBeforeUpdateRecord, a member of class TBaseProvider.

	DoGetRecords (see page 18466)	This is DoGetRecords, a member of class TCustomProvider.
	DoOnGetData (see page 18435)	Generates an OnGetData (see page 18443) event.
	DoOnUpdateData (see page 18435)	Generates an OnUpdateData (see page 18443) event.
	FetchDetails (see page 18436)	Fills a delta dataset with all nested dataset field values.
	FreeResolver (see page 18436)	Frees the resolver specified by the Resolver (see page 18441) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InternalApplyUpdates (see page 18436)	Provides the underlying implementation of the ApplyUpdates (see page 18459) method.
	InternalExecute (see page 18470)	Provides the underlying implementation of the Execute (see page 18466) method.
	InternalGetParams (see page 18471)	Returns the requested parameter values as a Variant array.
	InternalGetRecords (see page 18437)	Provides the underlying implementation of the GetRecords (see page 18468) method.
	InternalRowRequest (see page 18438)	Provides the underlying implementation of the RowRequest (see page 18472) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 18439)	Positions a data set on the record that corresponds to the current update in a specified delta packet.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActiveUpdateException (see page 18473)	This is SetActiveUpdateException, a member of class TCustomProvider.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRecord (see page 18439)	Retrieves specified data from a source dataset into a delta packet.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveUpdateException (see page 18474)	This is ActiveUpdateException, a member of class TCustomProvider.
	OwnerData (see page 18475)	This is OwnerData, a member of class TCustomProvider.
	PacketDataSet (see page 18440)	This is PacketDataSet, a member of class TBaseProvider.

Public Events

	Name	Description
	AfterUpdateRecord (see page 18442)	Occurs when the associated resolver has just updated a record.
	BeforeUpdateRecord (see page 18442)	This is BeforeUpdateRecord, a member of class TBaseProvider.
	OnDataRequest (see page 18442)	Occurs when a provider calls its DataRequest method.
	OnGetData (see page 18443)	Occurs after a provider retrieves data but before it is sent to the client.
	OnUpdateData (see page 18443)	Occurs when a provider begins to apply updates.
	OnUpdateError (see page 18444)	Occurs when a provider can't update a record because of an error condition.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	ApplyUpdates (see page 18459)	This is the overview for the ApplyUpdates method overload.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 18429)	Creates and initializes a TBaseProvider (see page 18423) object.
V	DataRequest (see page 18460)	Generates an OnDataRequest (see page 18482) event.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 18430)	Frees an instance of a provider object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Execute (see page 18466)	Executes a query, stored procedure, or custom SQL statement.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParams (see page 18467)	Returns the current parameter values.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetRecords (see page 18468)	This is the overview for the GetRecords method overload.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	RowRequest (see page 18472)	Returns information from a specified record.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Data (see page 18474)	Denotes a data packet created by the provider.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Exported (see page 18474)	Allows client applications to pass calls to the provider when it resides on an application server.
	Options (see page 18440)	Influences what is included in data packets and how the information in data packets is used.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Resolver (see page 18441)	Provides access to the resolver component that applies updates and resolves update errors.
	UpdateMode (see page 18441)	Controls (see page 5711) how the provider locates records when applying updates.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.

	Tag (see page 2808)	Stores an integer value as part of a component.

Strict private Fields

	Name	Description
	FActiveUpdateAbortException (see page 18455)	This is FActiveUpdateAbortException, a member of class TCustomProvider.
	FAfterApplyUpdates (see page 18455)	This is FAfterApplyUpdates, a member of class TCustomProvider.
	FAfterExecute (see page 18456)	This is FAfterExecute, a member of class TCustomProvider.
	FAfterGetParams (see page 18456)	This is FAfterGetParams, a member of class TCustomProvider.
	FAfterGetRecords (see page 18456)	This is FAfterGetRecords, a member of class TCustomProvider.
	FAfterRowRequest (see page 18456)	This is FAfterRowRequest, a member of class TCustomProvider.
	FAfterUpdateRecord (see page 18430)	This is FAfterUpdateRecord, a member of class TBaseProvider.
	FBeforeApplyUpdates (see page 18456)	This is FBeforeApplyUpdates, a member of class TCustomProvider.
	FBeforeCommit (see page 18430)	This is FBeforeCommit, a member of class TBaseProvider.
	FBeforeExecute (see page 18457)	This is FBeforeExecute, a member of class TCustomProvider.
	FBeforeGetParams (see page 18457)	This is FBeforeGetParams, a member of class TCustomProvider.
	FBeforeGetRecords (see page 18457)	This is FBeforeGetRecords, a member of class TCustomProvider.
	FBeforeRowRequest (see page 18457)	This is FBeforeRowRequest, a member of class TCustomProvider.
	FBeforeUpdateRecord (see page 18431)	This is FBeforeUpdateRecord, a member of class TBaseProvider.
	FDataDS (see page 18431)	This is FDataDS, a member of class TBaseProvider.
	FExported (see page 18458)	This is FExported, a member of class TCustomProvider.
	FOnDataRequest (see page 18458)	This is FOnDataRequest, a member of class TCustomProvider.
	FOnGetData (see page 18431)	This is FOnGetData, a member of class TBaseProvider.
	FOnUpdateData (see page 18431)	This is FOnUpdateData, a member of class TBaseProvider.
	FOnUpdateError (see page 18432)	This is FOnUpdateError, a member of class TBaseProvider.
	FOnValidate (see page 18458)	This is FOnValidate, a member of class TCustomProvider.
	FOwnerData (see page 18458)	This is FOwnerData, a member of class TCustomProvider.
	FProviderOptions (see page 18432)	This is FProviderOptions, a member of class TBaseProvider.
	FResolver (see page 18432)	This is FResolver, a member of class TBaseProvider.
	FUpdateMode (see page 18432)	This is FUpdateMode, a member of class TBaseProvider.

Strict private Methods

	Name	Description
	GetData (see page 18467)	This is GetData, a member of class TCustomProvider.

124.1.2.2 TBaseProvider.Create Constructor

Creates and initializes a TBaseProvider (see page 18423) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBaseProvider(TComponent * AOwner);
```

Description

Call Create to instantiate a provider at runtime. Providers placed in forms or data modules at design time are created automatically.

AOwner specifies the component, typically a form or data module, that is responsible for freeing the provider.

See Also

Owner (see page 2807)

124.1.2.3 TBaseProvider.Destroy Destructor

Frees an instance of a provider object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBaseProvider();
```

Description

Do not call Destroy directly. Instead, call Free, which checks that the provider is not nil before calling the Destroy method.

See Also

Free (see page 22300)

Resolver (see page 18441)

124.1.2.4 TBaseProvider Fields

The fields of the TBaseProvider class are listed here.

Strict private Fields

	Name	Description
	FAfterUpdateRecord (see page 18430)	This is FAfterUpdateRecord, a member of class TBaseProvider.
	FBeforeCommit (see page 18430)	This is FBeforeCommit, a member of class TBaseProvider.
	FBeforeUpdateRecord (see page 18431)	This is FBeforeUpdateRecord, a member of class TBaseProvider.
	FDataDS (see page 18431)	This is FDataDS, a member of class TBaseProvider.
	FOnGetData (see page 18431)	This is FOnGetData, a member of class TBaseProvider.
	FOnUpdateData (see page 18431)	This is FOnUpdateData, a member of class TBaseProvider.
	FOnUpdateError (see page 18432)	This is FOnUpdateError, a member of class TBaseProvider.
	FProviderOptions (see page 18432)	This is FProviderOptions, a member of class TBaseProvider.
	FResolver (see page 18432)	This is FResolver, a member of class TBaseProvider.
	FUpdateMode (see page 18432)	This is FUpdateMode, a member of class TBaseProvider.

124.1.2.4.1 TBaseProvider.FAfterUpdateRecord Field

Pascal

```
FAfterUpdateRecord: TAfterUpdateRecordEvent;
```

C++

```
TAfterUpdateRecordEvent FAfterUpdateRecord;
```

Description

This is FAfterUpdateRecord, a member of class TBaseProvider.

124.1.2.4.2 TBaseProvider.FBeforeCommit Field

Pascal

```
FBeforeCommit: TBeforeCommitEvent;
```

C++

```
TBeforeCommitEvent FBeforeCommit;
```

Description

This is FBeforeCommit, a member of class TBaseProvider.

124.1.2.4.3 TBaseProvider.FBeforeUpdateRecord Field

Pascal

```
FBeforeUpdateRecord: TBeforeUpdateRecordEvent;
```

C++

```
TBeforeUpdateRecordEvent FBeforeUpdateRecord;
```

Description

This is FBeforeUpdateRecord, a member of class TBaseProvider.

124.1.2.4.4 TBaseProvider.FDataDS Field

Pascal

```
FDataDS: TPacketDataSet;
```

C++

```
TPacketDataSet FDataDS;
```

Description

This is FDataDS, a member of class TBaseProvider.

124.1.2.4.5 TBaseProvider.FOnGetData Field

Pascal

```
FOnGetData: TProviderDataEvent;
```

C++

```
TProviderDataEvent FOnGetData;
```

Description

This is FOnGetData, a member of class TBaseProvider.

124.1.2.4.6 TBaseProvider.FOnUpdateData Field

Pascal

```
FOnUpdateData: TProviderDataEvent;
```

C++

```
TProviderDataEvent FOnUpdateData;
```

Description

This is FOnUpdateData, a member of class TBaseProvider.

124.1.2.4.7 TBaseProvider.FOnUpdateError Field

Pascal

```
FOnUpdateError: TResolverErrorEvent;
```

C++

```
TResolverErrorEvent FOnUpdateError;
```

Description

This is FOnUpdateError, a member of class TBaseProvider.

124.1.2.4.8 TBaseProvider.FProviderOptions Field

Pascal

```
FProviderOptions: TProviderOptions;
```

C++

```
TProviderOptions FProviderOptions;
```

Description

This is FProviderOptions, a member of class TBaseProvider.

124.1.2.4.9 TBaseProvider.FResolver Field

Pascal

```
FResolver: TCustomResolver;
```

C++

```
TCustomResolver FResolver;
```

Description

This is FResolver, a member of class TBaseProvider.

124.1.2.4.10 TBaseProvider.FUpdateMode Field

Pascal

```
FUpdateMode: TUpdateMode;
```

C++

```
TUpdateMode FUpdateMode;
```

Description

This is FUpdateMode, a member of class TBaseProvider.

124.1.2.5 TBaseProvider Methods

The methods of the TBaseProvider class are listed here.

Protected Methods

	Name	Description
	CheckResolver (see page 18433)	Instantiates a resolver for the Resolver (see page 18441) property if it does not already exist.
V	CreateDataPacket (see page 18433)	Generates a data packet for the data associated with the provider.
V	CreateResolver (see page 18434)	Creates a resolver component for handling the application of updates.
V	DoAfterUpdateRecord (see page 18434)	Generates an AfterUpdateRecord (see page 18442) event.
V	DoBeforeCommit (see page 18435)	This is DoBeforeCommit, a member of class TBaseProvider.
V	DoBeforeUpdateRecord (see page 18435)	This is DoBeforeUpdateRecord, a member of class TBaseProvider.
V	DoOnGetData (see page 18435)	Generates an OnGetData (see page 18443) event.
	DoOnUpdateData (see page 18435)	Generates an OnUpdateData (see page 18443) event.
V	FetchDetails (see page 18436)	Fills a delta dataset with all nested dataset field values.
	FreeResolver (see page 18436)	Frees the resolver specified by the Resolver (see page 18441) property.
V	InternalApplyUpdates (see page 18436)	Provides the underlying implementation of the ApplyUpdates (see page 18459) method.
V	InternalGetRecords (see page 18437)	Provides the underlying implementation of the GetRecords (see page 18468) method.
V	InternalRowRequest (see page 18438)	Provides the underlying implementation of the RowRequest (see page 18472) method.
V	LocateRecord (see page 18439)	Positions a data set on the record that corresponds to the current update in a specified delta packet.
V	UpdateRecord (see page 18439)	Retrieves specified data from a source dataset into a delta packet.

124.1.2.5.1 TBaseProvider.CheckResolver Method

Instantiates a resolver for the Resolver (see page 18441) property if it does not already exist.

Pascal

```
procedure CheckResolver;
```

C++

```
__fastcall CheckResolver();
```

Description

CheckResolver is called internally when the provider needs to use its resolver component. It checks whether Resolver (see page 18441) is nil (Delphi) or NULL (C++), and if so, calls CreateResolver (see page 18434) to assign a value.

See Also

Resolver (see page 18441)

CreateResolver (see page 18434)

124.1.2.5.2 TBaseProvider.CreateDataPacket Method

Generates a data packet for the data associated with the provider.

Pascal

```
procedure CreateDataPacket(PacketOpts: TGetRecordOptions; ProvOpts: TProviderOptions; var
RecsOut: Integer; var Data: OleVariant); virtual;
```

C++

```
virtual __fastcall CreateDataPacket(TGetRecordOptions PacketOpts, TProviderOptions ProvOpts,
int RecsOut, OleVariant Data);
```

Description

CreateDataPacket is used internally to generate data packets. As implemented in TBaseProvider (see page 18423),

CreateDataPacket sets RecsOut to 0 and Data (see page 18474) to NULL. Descendants override this method to create the data packets that are returned by GetRecords (see page 18468).

The PacketOpts parameter indicates what information (other than record values) is included in the data packet. It reflects the Options (see page 18440) passed to the GetRecords (see page 18468) method.

The ProvOpts parameter reflects the current value of the Options (see page 18440) property.

The RecsOut parameter specifies the number of records desired in the packet and returns the actual number of records in the generated data packet.

The Data (see page 18474) parameter returns the generated data packet.

See Also

GetRecords (see page 18468)

Options (see page 18440)

124.1.2.5.3 TBaseProvider.CreateResolver Method

Creates a resolver component for handling the application of updates.

Pascal

```
function CreateResolver: TCustomResolver; virtual;
```

C++

```
virtual __fastcall TCustomResolver CreateResolver();
```

Description

CreateResolver is called automatically when a resolver is needed to implement one of TBaseProvider (see page 18423)'s methods. As implemented in TBaseProvider (see page 18423), CreateResolver always returns nil (Delphi) or NULL (C++). Descendants of TBaseProvider (see page 18423) override this method to create a TCustomResolver (see page 18483) descendant that applies updates in the desired manner.

See Also

Resolver (see page 18441)

FreeResolver (see page 18436)

124.1.2.5.4 TBaseProvider.DoAfterUpdateRecord Method

Generates an AfterUpdateRecord (see page 18442) event.

Pascal

```
procedure DoAfterUpdateRecord(SourceDS: TDataSet; DeltaDS: TCustomClientDataSet; UpdateKind:
TUpdateKind); virtual;
```

C++

```
virtual __fastcall DoAfterUpdateRecord(TDataSet SourceDS, TCustomClientDataSet DeltaDS,
TUpdateKind UpdateKind);
```

Description

DoAfterUpdateRecord is the protected implementation of the AfterUpdateRecord (see page 18442) event. Override this method to perform class-specific actions in response to data requests.

See Also

AfterUpdateRecord (see page 18442)

124.1.2.5.5 TBaseProvider.DoBeforeCommit Method

Pascal

```
procedure DoBeforeCommit(Delta: TPacketDataSet; ErrorCount: Integer; MaxErrors: Integer; const
ReturnVar: OleVariant); virtual;
```

C++

```
virtual __fastcall DoBeforeCommit(TPacketDataSet Delta, int ErrorCount, int MaxErrors, const
OleVariant ReturnVar);
```

Description

This is DoBeforeCommit, a member of class TBaseProvider.

124.1.2.5.6 TBaseProvider.DoBeforeUpdateRecord Method

Pascal

```
procedure DoBeforeUpdateRecord(SourceDS: TDataSet; DeltaDS: TCustomClientDataSet; UpdateKind:
TUpdateKind; var Applied: Boolean); virtual;
```

C++

```
virtual __fastcall DoBeforeUpdateRecord(TDataSet SourceDS, TCustomClientDataSet DeltaDS,
TUpdateKind UpdateKind, Boolean Applied);
```

Description

This is DoBeforeUpdateRecord, a member of class TBaseProvider.

124.1.2.5.7 TBaseProvider.DoOnGetData Method

Generates an OnGetData (see page 18443) event.

Pascal

```
procedure DoOnGetData(var Data: OleVariant); virtual;
```

C++

```
virtual __fastcall DoOnGetData(OleVariant Data);
```

Description

DoOnGetData is the protected implementation of the OnGetData (see page 18443) event. Override this method to perform class-specific actions in response to data requests.

See Also

OnGetData (see page 18443)

124.1.2.5.8 TBaseProvider.DoOnUpdateData Method

Generates an OnUpdateData (see page 18443) event.

Pascal

```
procedure DoOnUpdateData(Delta: TPacketDataSet);
```

C++

```
__fastcall DoOnUpdateData(TPacketDataSet Delta);
```

Description

DoOnUpdateData is the protected implementation of the OnUpdateData (see page 18443) event. The Delta property contains the delta packet of records to update. It is passed to the OnUpdateData (see page 18443) event handler as a client dataset.

Override this method to perform class-specific actions in response to update requests.

See Also

OnUpdateData (see page 18443)

124.1.2.5.9 TBaseProvider.FetchDetails Method

Fills a delta dataset with all nested dataset field values.

Pascal

```
procedure FetchDetails(Source: TDataSet; Delta: TDataSet); virtual;
```

C++

```
virtual __fastcall FetchDetails(TDataSet Source, TDataSet Delta);
```

Description

FetchDetails is used internally to fetch the current values of any nested datasets when the provider is executing its RowRequest (see page 18472) method. As implemented in TBaseProvider (see page 18423), FetchDetails does nothing. Descendants override this method to add nested detail datasets to the delta packet specified by Delta.

Source is a source dataset that has the current values for all fields. Delta is a delta dataset that must be updated so that its nested detail fields match the values in Source.

See Also

RowRequest (see page 18472)

124.1.2.5.10 TBaseProvider.FreeResolver Method

Frees the resolver specified by the Resolver (see page 18441) property.

Pascal

```
procedure FreeResolver;
```

C++

```
__fastcall FreeResolver();
```

Description

Call FreeResolver to safely dispose of the component specified by the Resolver (see page 18441) property. FreeResolver sets the Resolver (see page 18441) property to nil (Delphi) or NULL (C++) after calling the resolver's Free method.

See Also

Resolver (see page 18441)

CreateResolver (see page 18434)

Free (see page 22300)

124.1.2.5.11 TBaseProvider.InternalApplyUpdates Method

Provides the underlying implementation of the ApplyUpdates (see page 18459) method.

Pascal

```
function InternalApplyUpdates(const Delta: OleVariant; MaxErrors: Integer; out ErrorCount:
Integer): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalApplyUpdates(const OleVariant Delta, int MaxErrors, int
ErrorCount);
```

Description

The ApplyUpdates (see page 18459) method calls InternalApplyUpdates to apply the updates contained in a delta packet after generating the BeforeApplyUpdates (see page 18479) event and before generating the AfterApplyUpdates (see page 18475) event.

If the Options (see page 18440) property includes poReadOnly, InternalApplyUpdates raises an exception. Otherwise, it calls the ApplyUpdates (see page 18459) method of its Resolver (see page 18441) component to handle the actual update and resolving process. If the Resolver (see page 18441) property is nil (Delphi) or NULL (C++), ApplyUpdates (see page 18459) calls the CreateResolver (see page 18434) method and assigns the resulting resolver to the Resolver (see page 18441) property.

Delta is the delta packet containing insertions, deletions, and modifications to be applied.

MaxErrors indicates the maximum number of errors that can be applied before stopping the update operation. If MaxErrors is –1, the provider tries to apply all updates.

ErrorCount returns the number of errors encountered during the update operation.

InternalApplyUpdates returns a data packet of updates that could not be applied to the database as an OleVariant.

See Also

Resolver (see page 18441)

ApplyUpdates (see page 18490)

CreateResolver (see page 18434)

Options (see page 18440)

applyupdates (see page 18459)

124.1.2.5.12 TBaseProvider.InternalGetRecords Method

Provides the underlying implementation of the GetRecords (see page 18468) method.

Pascal

```
function InternalGetRecords(Count: Integer; out RecsOut: Integer; Options: TGetRecordOptions;
const CommandText: WideString; var Params: OleVariant): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalGetRecords(int Count, int RecsOut, TGetRecordOptions
Options, const BSTR CommandText, OleVariant Params);
```

Description

The GetRecords (see page 18468) method calls InternalGetRecords to assemble a data packet after calling DoBeforeGetRecords (see page 18465) to apply the CommandText and Params and generate a BeforeGetRecords (see page 18480) event. InternalGetRecords calls CreateDataPacket (see page 18433) to create a data packet that reflects the current property settings of the provider. It then generates an OnGetData (see page 18443) event.

Count indicates the number of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is

retrieved. If Count is greater than 0, only the specified number of records are retrieved.

RecsOut returns the actual number of records retrieved.

Options (see page 18440) indicates what information should be added to the data packet in addition to data. It can include grMetaData (include metadata), grReset (start providing records with the first one in the dataset), and grXML (encode the datapacket in XML rather than as an OleVariant) or grXMLUTF8 (same as grXML except extended characters are encoded using UTF8).

CommandText specifies an optional SQL statement that replaces the SQL of an associated query, or the name of a table or stored procedure that replaces the associated table or stored procedure. This parameter is ignored if the Options (see page 18440) property does not include poAllowCommandText.

Params specifies any parameter values that should be applied to CommandText or to an associated query or stored procedure. If the provider is associated with a table, Params specifies field values that filter the result set. On exit from GetRecords (see page 18468), Params returns any output parameters.

Records are returned as a data packet in a Variant.

See Also

Data (see page 18474)

CreateDataPacket (see page 18433)

GetRecords (see page 18468)

OnGetData (see page 18443)

124.1.2.5.13 TBaseProvider.InternalRowRequest Method

Provides the underlying implementation of the RowRequest (see page 18472) method.

Pascal

```
function InternalRowRequest(const Row: OleVariant; RequestType: TFetchOptions): OleVariant;
override;
```

C++

```
virtual __fastcall OleVariant InternalRowRequest(const OleVariant Row, TFetchOptions
RequestType);
```

Description

The RowRequest (see page 18472) method calls InternalRowRequest to assemble the specified information after generating a BeforeRowRequest (see page 18482) event and before generating an AfterRowRequest (see page 18478) event. InternalRowRequest passes this request on to its associated Resolver (see page 18441).

The Row parameter is an OleVariant that describes the record for which information is requested. The provider uses Row to locate the record before assembling the requested information.

InternalRowRequest returns the requested information in a delta packet.

See Also

RowRequest (see page 18472)

Resolver (see page 18441)

RowRequest (see page 18496)

124.1.2.5.14 TBaseProvider.LocateRecord Method

Positions a data set on the record that corresponds to the current update in a specified delta packet.

Pascal

```
procedure LocateRecord(Source: TDataSet; Delta: TDataSet); virtual;
```

C++

```
virtual __fastcall LocateRecord(TDataSet Source, TDataSet Delta);
```

Description

LocateRecord is called indirectly by the RowRequest (see page 18472) method to locate a record before fetching information about a specified record. The Source parameter is the dataset containing current record values. The Delta parameter is a delta packet that identifies the desired type of information and information identifying the desired record.

As implemented in TBaseProvider (see page 18423), LocateRecord does nothing. Descendants override this method to locate the specified record in Source and navigate to it.

See Also

RowRequest (see page 18472)

124.1.2.5.15 TBaseProvider.UpdateRecord Method

Retrieves specified data from a source dataset into a delta packet.

Pascal

```
procedure UpdateRecord(Source: TDataSet; Delta: TDataSet; BlobsOnly: Boolean; KeyOnly:
Boolean); virtual;
```

C++

```
virtual __fastcall UpdateRecord(TDataSet Source, TDataSet Delta, Boolean BlobsOnly, Boolean
KeyOnly);
```

Description

UpdateRecord is called indirectly by the RowRequest (see page 18472) method to fetch BLOB fields or nested detail sets. The Source parameter is the dataset containing the BLOB fields or the detail dataset. The Delta parameter is a delta packet that returns the requested data. BlobsOnly indicates whether all field values should be provided (BlobsOnly == false) or only Blob field values (BlobsOnly == true).

As implemented in TBaseProvider (see page 18423), UpdateRecord does nothing. Descendants override this method to fill the Delta parameter with the requested information.

See Also

RowRequest (see page 18472)

124.1.2.6 TBaseProvider Properties

The properties of the TBaseProvider class are listed here.

Protected Properties

	Name	Description
	PacketDataSet (see page 18440)	This is PacketDataSet, a member of class TBaseProvider.

Public Properties

	Name	Description
	Options (see page 18440)	Influences what is included in data packets and how the information in data packets is used.
R	Resolver (see page 18441)	Provides access to the resolver component that applies updates and resolves update errors.
	UpdateMode (see page 18441)	Controls (see page 5711) how the provider locates records when applying updates.

124.1.2.6.1 TBaseProvider.Options Property

Influences what is included in data packets and how the information in data packets is used.

Pascal

```
property Options: TProviderOptions;
```

C++

```
__property TProviderOptions Options;
```

Description

Set Options to customize how the provider communicates with client datasets. Options is a set that specifies

Whether nested detail sets and BLOB data are included in data packets or fetched separately.

Whether data packets include field component properties such as formats, display names, and maximum & minimum values.

Whether data packets are provided on a read-only basis, and whether specific types of changes (insertions, edits, or deletions) are disabled.

Whether changes to master table fields should cause corresponding updates to detail tables.

Whether a single update can affect multiple server records.

Whether the client's records are refreshed when it applies updates.

Whether the client can send SQL statements that override the provider's dataset.

Note: Not all options are supported by an InternetExpress client. Specifically, InternetExpress clients don't support poFetchBlobsOnDemand, poFetchDetailsOnDemand, poDisableInserts, poDisableEdits, psDisableDeletes, poPropogateChanges, and poAllowCommandText.

See Also

Data (see page 18474)

applyupdates (see page 18459)

Overriding the Dataset On the Application Server

Setting Options That Influence the Data Packets

124.1.2.6.2 TBaseProvider.PacketDataSet Property

Pascal

```
property PacketDataSet: TPacketDataSet;
```

C++

```
__property TPacketDataSet PacketDataSet;
```

Description

This is PacketDataSet, a member of class TBaseProvider.

124.1.2.6.3 TBaseProvider.Resolver Property

Provides access to the resolver component that applies updates and resolves update errors.

Pascal

```
property Resolver: TCustomResolver;
```

C++

```
__property TCustomResolver Resolver;
```

Description

Provider (see page 18419) components use a resolver to handle the details of applying updates and resolving update errors. Most applications do not need to use the resolver directly.

When any method that uses a resolver is called, the provider automatically creates a resolver component (using the CreateResolver (see page 18434) method) if the value of Resolver is nil (Delphi) or NULL (C++). Thus, this property is nil (Delphi) or NULL (C++) until the first method that requires a resolver is called.

See Also

applyupdates (see page 18459)

CreateResolver (see page 18434)

124.1.2.6.4 TBaseProvider.UpdateMode Property

Controls (see page 5711) how the provider locates records when applying updates.

Pascal

```
property UpdateMode: TUpdateMode;
```

C++

```
__property TUpdateMode UpdateMode;
```

Description

Use UpdateMode to specify the criteria for locating a record in the dataset when applying updates. UpdateMode specifies whether modified records are located based on all columns (fields), on only the key fields, or on the key fields plus the original values of fields that have been modified.

Note: To control how the provider locates records by specifying individual fields, use the field components' ProviderFlags property.

See Also

applyupdates (see page 18459)

ProviderFlags (see page 7176)

Influencing How Updates Are Applied

124.1.2.7 TBaseProvider Events

The events of the TBaseProvider class are listed here.

Public Events

	Name	Description
	AfterUpdateRecord (see page 18442)	Occurs when the associated resolver has just updated a record.
	BeforeUpdateRecord (see page 18442)	This is BeforeUpdateRecord, a member of class TBaseProvider.
	OnDataRequest (see page 18442)	Occurs when a provider calls its DataRequest method.
	OnGetData (see page 18443)	Occurs after a provider retrieves data but before it is sent to the client.
	OnUpdateData (see page 18443)	Occurs when a provider begins to apply updates.
	OnUpdateError (see page 18444)	Occurs when a provider can't update a record because of an error condition.

124.1.2.7.1 TBaseProvider.AfterUpdateRecord Event

Occurs when the associated resolver has just updated a record.

Pascal

```
property AfterUpdateRecord: TAfterUpdateRecordEvent;
```

C++

```
__property TAfterUpdateRecordEvent AfterUpdateRecord;
```

Description

Write an AfterUpdateRecord event handler to provide special processing immediately after a record has been successfully updated. For example, use AfterUpdateRecord to maintain a log of successful updates.

See Also

beforeupdaterecord (see page 18442)

AfterApplyUpdates (see page 18475)

124.1.2.7.2 TBaseProvider.BeforeUpdateRecord Event

Pascal

```
property BeforeUpdateRecord: TBeforeUpdateRecordEvent;
```

C++

```
__property TBeforeUpdateRecordEvent BeforeUpdateRecord;
```

Description

This is BeforeUpdateRecord, a member of class TBaseProvider.

124.1.2.7.3 TBaseProvider.OnDataRequest Event

Occurs when a provider calls its DataRequest method.

Pascal

```
property OnDataRequest: TDataRequestEvent;
```

C++

```
__property TDataRequestEvent OnDataRequest;
```

Description

Write an OnDataRequest event handler to do custom processing of data. OnDataRequest has no built-in use. Instead, it allows developers to augment the IAppServer interface so that clients in a multi-tiered application can communicate directly with an individual provider.

If an application does not provide an OnDataRequest event handler, calls to DataRequest return NULL.

Note: To provide the same sort of custom processing on the client side, use an automation method on the connection component.

See Also

DataRequest (see page 18460)

IAppServer (see page 16450)

Communicating with Providers Using Custom Events

Communicating with the Client Dataset

Responding to Client-generated Events

124.1.2.7.4 TBaseProvider.OnGetData Event

Occurs after a provider retrieves data but before it is sent to the client.

Pascal

```
property OnGetData: TProviderDataEvent;
```

C++

```
__property TProviderDataEvent OnGetData;
```

Description

Write an OnGetData event handler to perform any custom data processing the provider should handle before returning data to a client. The OnGetData event occurs after a provider fetches data from its dataset but before the data is sent on to the client. Using the methods of the DataSet parameter, data can be edited in the OnGetData event handler before sending it to the client dataset or XML broker.

Note: OnGetData occurs when the Data (see page 18474) property is read, as well as when the GetRecords (see page 18468) method is called.

See Also

GetRecords (see page 18468)

OnUpdateData (see page 18443)

BeforeGetRecords (see page 18480)

AfterGetRecords (see page 18477)

Responding to Client Data Requests

124.1.2.7.5 TBaseProvider.OnUpdateData Event

Occurs when a provider begins to apply updates.

Pascal

```
property OnUpdateData: TProviderDataEvent;
```

C++

```
__property TProviderDataEvent OnUpdateData;
```

Description

Write an OnUpdateData event to handle special processing before the provider begins applying updates received from the client dataset or XML broker. Using the methods belonging to the DataSet parameter, an application can

Examine data (for example, for values or data changes that should not permitted), and raise exceptions that cancel applying of updates before they occur.

Change data (for example encrypting or decrypting values) before it is sent on to the source dataset or database server.

See Also

applyupdates (see page 18459)

beforeupdaterecord (see page 18442)

Adding Custom Information to Data Packets

Editing Delta Packets Before Updating the Database

Influencing How Updates Are Applied

124.1.2.7.6 TBaseProvider.OnUpdateError Event

Occurs when a provider can't update a record because of an error condition.

Pascal

```
property OnUpdateError: TResolverErrorEvent;
```

C++

```
__property TResolverErrorEvent OnUpdateError;
```

Description

Write an OnUpdateError event handler to respond to error conditions that arise when the provider applies a client dataset's updates. The OnUpdateError event handler is called for each record that can't be updated.

Warning: The provider can't detect when an update conflicts with another user's changes to a memo field. This situation will not generate an OnUpdateError event.

See Also

applyupdates (see page 18459)

OnReconcileError (see page 8524)

NewValue (see page 7174)

OldValue (see page 7175)

CurValue (see page 7155)

Resolving Update Errors On the Provider

124.1.3 TCustomPacketWriter Class

Class Hierarchy

System::TObject → Provider.TCustomPacketWriter

Pascal

```
TCustomPacketWriter = class(TObject);
```

C++

```
class TCustomPacketWriter : public TObject;
```

File

Provider

Description

This is class Provider.TCustomPacketWriter.

124.1.3.1 TCustomPacketWriter Members

The following tables list the members exposed by TCustomPacketWriter.

Protected Fields

	Name	Description
	FBuffer (see page 18447)	This is FBuffer, a member of class TCustomPacketWriter.

Protected Methods

	Name	Description
V	AddAttribute (see page 18448)	This is AddAttribute, a member of class TCustomPacketWriter.
	Check (see page 18448)	This is Check, a member of class TCustomPacketWriter.
	SetDSWriter (see page 18448)	This is SetDSWriter, a member of class TCustomPacketWriter.

Protected Properties

	Name	Description
R	DSWriter (see page 18449)	This is DSWriter, a member of class TCustomPacketWriter.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 18446)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 18446)	Disposes of an object instance.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Strict private Fields

	Name	Description
	FIDSWriter (see page 18447)	This is FIDSWriter, a member of class TCustomPacketWriter.

124.1.3.2 TCustomPacketWriter.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TCustomPacketWriter();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

124.1.3.3 TCustomPacketWriter.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomPacketWriter();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

124.1.3.4 TCustomPacketWriter Fields

The fields of the TCustomPacketWriter class are listed here.

Protected Fields

	Name	Description
	FBuffer (see page 18447)	This is FBuffer, a member of class TCustomPacketWriter.

Strict private Fields

	Name	Description
	FIDSWriter (see page 18447)	This is FIDSWriter, a member of class TCustomPacketWriter.

124.1.3.4.1 TCustomPacketWriter.FBuffer Field

Pascal

```
FBuffer: TBlobByteData;
```

C++

```
TBlobByteData FBuffer;
```

Description

This is FBuffer, a member of class TCustomPacketWriter.

124.1.3.4.2 TCustomPacketWriter.FIDSWriter Field

Pascal

```
FIDSWriter: IDSWriter;
```

C++

```
IDSWriter FIDSWriter;
```

Description

This is FIDSWriter, a member of class TCustomPacketWriter.

124.1.3.5 TCustomPacketWriter Methods

The methods of the TCustomPacketWriter class are listed here.

Protected Methods

	Name	Description
	AddAttribute (see page 18448)	This is AddAttribute, a member of class TCustomPacketWriter.
	Check (see page 18448)	This is Check, a member of class TCustomPacketWriter.
	SetDSWriter (see page 18448)	This is SetDSWriter, a member of class TCustomPacketWriter.

124.1.3.5.1 TCustomPacketWriter.AddAttribute Method

Pascal

```
procedure AddAttribute(Area: TPcktAttrArea; const ParamName: string; const Value: OleVariant;
IncludeInDelta: Boolean); virtual;
```

C++

```
virtual __fastcall AddAttribute(TPcktAttrArea Area, const AnsiString ParamName, const
OleVariant Value, Boolean IncludeInDelta);
```

Description

This is AddAttribute, a member of class TCustomPacketWriter.

124.1.3.5.2 TCustomPacketWriter.Check Method

Pascal

```
procedure Check(Status: Integer);
```

C++

```
__fastcall Check(int Status);
```

Description

This is Check, a member of class TCustomPacketWriter.

124.1.3.5.3 TCustomPacketWriter.SetDSWriter Method

Pascal

```
procedure SetDSWriter(const Value: IDSWriter);
```

C++

```
__fastcall SetDSWriter(const IDSWriter Value);
```

Description

This is SetDSWriter, a member of class TCustomPacketWriter.

124.1.3.6 TCustomPacketWriter Properties

The properties of the TCustomPacketWriter class are listed here.

Protected Properties

	Name	Description
	DSWriter (see page 18449)	This is DSWriter, a member of class TCustomPacketWriter.

124.1.3.6.1 TCustomPacketWriter.DSWriter Property

Pascal

```
property DSWriter: IDSWriter;
```

C++

```
__property IDSWriter DSWriter;
```

Description

This is DSWriter, a member of class TCustomPacketWriter.

124.1.4 TCustomProvider Class

TCustomProvider is the base class for data providers.

Class Hierarchy

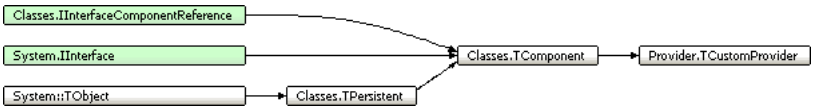

Pascal

```
TCustomProvider = class(TComponent);
```

C++

```
class TCustomProvider : public TComponent;
```

File

Provider

Description

TCustomProvider defines the fundamental properties, methods, and events required of all provider objects, either through abstract (in C++ terminology, pure virutal) method declarations, or through virtual methods that do nothing but provide a backstop.

Do not create instances of TCustomProvider. To add a provider component to a form or data module, use a descendant of TCustomProvider such as TDataSetProvider (see page 18508) or TXMLTransformProvider.

A provider derived from TCustomProvider is an intermediate link between TClientDataSet or TXMLBroker object and a source dataset or source XML document. The source dataset or XML document can reside in the same application as the client dataset or XML broker, or it can be part of a separate application server. The provider packages data from the source dataset or XML document and passes it in one or more transportable data packets to the client dataset or XML broker. The client dataset

reconstructs the data in the data packet to create a local, in-memory copy for user access. When user access is complete, the client dataset repackages any changed data and sends the updates back to the provider. The provider applies the updates to the database or source dataset.

When the provider supplies data to a client dataset, the client dataset reconstructs the data in the data packet to create a local, in-memory copy for user access. When user access is complete, the client dataset repackages any changed data and sends the updates back to the provider. The provider applies the updates to the database, to the source dataset, or to the source XML document.

If the provider can encode data packets in XML, it can supply data to an XML broker, which adds the data packet to an HTML document that is downloaded by a Web client. When the XML broker receives updates from the Web client, it sends the updates to the provider, which applies them to the database, to the source dataset, or to the source XML document.

Client datasets and XML brokers communicate with a provider through the IAppServer interface. In multi-tiered applications, this is the interface of the remote data module that contains the provider. To enable the remote data module to pass calls to the provider, the provider's Exported (see page 18474) property must be true, and its Owner property must specify the remote data module.

See Also

TClientDataSet (see page 8289)

TDataSetProvider (see page 18508)

IAppServer (see page 16450)

TXMLTransformProvider (see page 26492)

124.1.4.1 TCustomProvider Members

The following tables list the members exposed by TCustomProvider.

Protected Events

	Name	Description
	AfterApplyUpdates (see page 18475)	Occurs after the provider applies updates.
	AfterExecute (see page 18476)	Occurs after the provider passes an execute command to its dataset.
	AfterGetParams (see page 18477)	This is AfterGetParams, a member of class TCustomProvider.
	AfterGetRecords (see page 18477)	Occurs after the provider creates a data packet to send to the client dataset.
	AfterRowRequest (see page 18478)	Occurs after the provider fetches new information about the current record.
	BeforeApplyUpdates (see page 18479)	Occurs before the provider applies updates from a client dataset.
	BeforeExecute (see page 18479)	This is BeforeExecute, a member of class TCustomProvider.
	BeforeGetParams (see page 18480)	Occurs before the provider creates an OleVariant that encodes the current parameter values and sends them to a client dataset.
	BeforeGetRecords (see page 18480)	Occurs before the provider creates a data packet and sends it to a client dataset.
	BeforeRowRequest (see page 18482)	Occurs before the provider creates a delta packet with information about the current record.
	OnDataRequest (see page 18482)	Occurs when a provider calls its DataRequest (see page 18460) method.
	OnValidate (see page 18483)	This is OnValidate, a member of class TCustomProvider.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.

	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoAfterApplyUpdates (see page 18460)	Generates an AfterApplyUpdates (see page 18475) event.
	DoAfterExecute (see page 18461)	Generates an AfterExecute (see page 18476) event.
	DoAfterGetParams (see page 18461)	Generates an AfterGetParams (see page 18477) event.
	DoAfterGetRecords (see page 18462)	Generates an AfterGetRecords (see page 18477) event.
	DoAfterRowRequest (see page 18462)	Generates an AfterRowRequest (see page 18478) event.
	DoApplyUpdates (see page 18463)	This is DoApplyUpdates, a member of class TCustomProvider.
	DoBeforeApplyUpdates (see page 18463)	Generates a BeforeApplyUpdates (see page 18479) event.
	DoBeforeExecute (see page 18464)	Generates a BeforeExecute (see page 18479) event.
	DoBeforeGetParams (see page 18464)	Generates a BeforeGetParams (see page 18480) event.
	DoBeforeGetRecords (see page 18465)	Generates a BeforeGetRecords (see page 18480) event.
	DoBeforeRowRequest (see page 18465)	Generates a BeforeRowRequest (see page 18482) event.
	DoGetRecords (see page 18466)	This is DoGetRecords, a member of class TCustomProvider.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InternalApplyUpdates (see page 18469)	Provides the underlying implementation of the ApplyUpdates (see page 18459) method.
	InternalExecute (see page 18470)	Provides the underlying implementation of the Execute (see page 18466) method.
	InternalGetParams (see page 18471)	Returns the requested parameter values as a Variant array.
	InternalGetRecords (see page 18471)	Provides the underlying implementation of the GetRecords (see page 18468) method.
	InternalRowRequest (see page 18472)	Provides the underlying implementation of the RowRequest (see page 18472) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActiveUpdateException (see page 18473)	This is SetActiveUpdateException, a member of class TCustomProvider.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActiveUpdateException (see page 18474)	This is ActiveUpdateException, a member of class TCustomProvider.
	OwnerData (see page 18475)	This is OwnerData, a member of class TCustomProvider.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	ApplyUpdates (see page 18459)	This is the overview for the ApplyUpdates method overload.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 18454)	Creates a new TCustomProvider (see page 18449) object.
	DataRequest (see page 18460)	Generates an OnDataRequest (see page 18482) event.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 18454)	Frees an instance of a provider object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 18466)	Executes a query, stored procedure, or custom SQL statement.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParams (see page 18467)	Returns the current parameter values.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetRecords (see page 18468)	This is the overview for the GetRecords method overload.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	RowRequest (see page 18472)	Returns information from a specified record.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Data (see page 18474)	Denotes a data packet created by the provider.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Exported (see page 18474)	Allows client applications to pass calls to the provider when it resides on an application server.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

Strict private Fields

	Name	Description
	FActiveUpdateAbortException (see page 18455)	This is FActiveUpdateAbortException, a member of class TCustomProvider.
	FAfterApplyUpdates (see page 18455)	This is FAfterApplyUpdates, a member of class TCustomProvider.
	FAfterExecute (see page 18456)	This is FAfterExecute, a member of class TCustomProvider.

	FAfterGetParams (see page 18456)	This is FAfterGetParams, a member of class TCustomProvider.
	FAfterGetRecords (see page 18456)	This is FAfterGetRecords, a member of class TCustomProvider.
	FAfterRowRequest (see page 18456)	This is FAfterRowRequest, a member of class TCustomProvider.
	FBeforeApplyUpdates (see page 18456)	This is FBeforeApplyUpdates, a member of class TCustomProvider.
	FBeforeExecute (see page 18457)	This is FBeforeExecute, a member of class TCustomProvider.
	FBeforeGetParams (see page 18457)	This is FBeforeGetParams, a member of class TCustomProvider.
	FBeforeGetRecords (see page 18457)	This is FBeforeGetRecords, a member of class TCustomProvider.
	FBeforeRowRequest (see page 18457)	This is FBeforeRowRequest, a member of class TCustomProvider.
	FExported (see page 18458)	This is FExported, a member of class TCustomProvider.
	FOnDataRequest (see page 18458)	This is FOnDataRequest, a member of class TCustomProvider.
	FOnValidate (see page 18458)	This is FOnValidate, a member of class TCustomProvider.
	FOwnerData (see page 18458)	This is FOwnerData, a member of class TCustomProvider.

Strict private Methods

	Name	Description
	GetData (see page 18467)	This is GetData, a member of class TCustomProvider.

124.1.4.2 TCustomProvider.Create Constructor

Creates a new TCustomProvider (see page 18449) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomProvider(TComponent * AOwner);
```

Description

Call Create to instantiate a provider at runtime. Providers placed in forms or data modules at design time are created automatically.

The constructor assigns the AOwner parameter as the owner of the provider. This owner is responsible for freeing the provider when it is destroyed. If the owner is a remote data module, the constructor registers the provider with that remote data module so that it is available to client applications through the IAppServer interface.

See Also

IAppServer (see page 16450)

Providers (see page 19908)

registerprovider (see page 19902)

Owner (see page 2807)

124.1.4.3 TCustomProvider.Destroy Destructor

Frees an instance of a provider object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomProvider();
```

Description

Do not call Destroy directly. Instead, call Free, which checks that the provider is not nil before calling its Destroy method. If the provider's Owner is a remote data module, Destroy unregisters the provider from that remote data module.

124.1.4.4 TCustomProvider Fields

The fields of the TCustomProvider class are listed here.

Strict private Fields

	Name	Description
	FActiveUpdateAbortException (see page 18455)	This is FActiveUpdateAbortException, a member of class TCustomProvider.
	FAfterApplyUpdates (see page 18455)	This is FAfterApplyUpdates, a member of class TCustomProvider.
	FAfterExecute (see page 18456)	This is FAfterExecute, a member of class TCustomProvider.
	FAfterGetParams (see page 18456)	This is FAfterGetParams, a member of class TCustomProvider.
	FAfterGetRecords (see page 18456)	This is FAfterGetRecords, a member of class TCustomProvider.
	FAfterRowRequest (see page 18456)	This is FAfterRowRequest, a member of class TCustomProvider.
	FBeforeApplyUpdates (see page 18456)	This is FBeforeApplyUpdates, a member of class TCustomProvider.
	FBeforeExecute (see page 18457)	This is FBeforeExecute, a member of class TCustomProvider.
	FBeforeGetParams (see page 18457)	This is FBeforeGetParams, a member of class TCustomProvider.
	FBeforeGetRecords (see page 18457)	This is FBeforeGetRecords, a member of class TCustomProvider.
	FBeforeRowRequest (see page 18457)	This is FBeforeRowRequest, a member of class TCustomProvider.
	FExported (see page 18458)	This is FExported, a member of class TCustomProvider.
	FOnDataRequest (see page 18458)	This is FOnDataRequest, a member of class TCustomProvider.
	FOnValidate (see page 18458)	This is FOnValidate, a member of class TCustomProvider.
	FOwnerData (see page 18458)	This is FOwnerData, a member of class TCustomProvider.

124.1.4.4.1 TCustomProvider.FActiveUpdateAbortException Field

Pascal

```
FActiveUpdateAbortException: Exception;
```

C++

```
Exception FActiveUpdateAbortException;
```

Description

This is FActiveUpdateAbortException, a member of class TCustomProvider.

124.1.4.4.2 TCustomProvider.FAfterApplyUpdates Field

Pascal

```
FAfterApplyUpdates: TRemoteEvent;
```

C++

```
TRemoteEvent FAfterApplyUpdates;
```

Description

This is FAfterApplyUpdates, a member of class TCustomProvider.

124.1.4.4.3 TCustomProvider.FAfterExecute Field

Pascal

```
FAfterExecute: TRemoteEvent;
```

C++

```
TRemoteEvent FAfterExecute;
```

Description

This is FAfterExecute, a member of class TCustomProvider.

124.1.4.4.4 TCustomProvider.FAfterGetParams Field

Pascal

```
FAfterGetParams: TRemoteEvent;
```

C++

```
TRemoteEvent FAfterGetParams;
```

Description

This is FAfterGetParams, a member of class TCustomProvider.

124.1.4.4.5 TCustomProvider.FAfterGetRecords Field

Pascal

```
FAfterGetRecords: TRemoteEvent;
```

C++

```
TRemoteEvent FAfterGetRecords;
```

Description

This is FAfterGetRecords, a member of class TCustomProvider.

124.1.4.4.6 TCustomProvider.FAfterRowRequest Field

Pascal

```
FAfterRowRequest: TRemoteEvent;
```

C++

```
TRemoteEvent FAfterRowRequest;
```

Description

This is FAfterRowRequest, a member of class TCustomProvider.

124.1.4.4.7 TCustomProvider.FBeforeApplyUpdates Field

Pascal

```
FBeforeApplyUpdates: TRemoteEvent;
```

C++

```
TRemoteEvent FBeforeApplyUpdates;
```

Description

This is FBeforeApplyUpdates, a member of class TCustomProvider.

124.1.4.4.8 TCustomProvider.FBeforeExecute Field

Pascal

```
FBeforeExecute: TRemoteEvent;
```

C++

```
TRemoteEvent FBeforeExecute;
```

Description

This is FBeforeExecute, a member of class TCustomProvider.

124.1.4.4.9 TCustomProvider.FBeforeGetParams Field

Pascal

```
FBeforeGetParams: TRemoteEvent;
```

C++

```
TRemoteEvent FBeforeGetParams;
```

Description

This is FBeforeGetParams, a member of class TCustomProvider.

124.1.4.4.10 TCustomProvider.FBeforeGetRecords Field

Pascal

```
FBeforeGetRecords: TRemoteEvent;
```

C++

```
TRemoteEvent FBeforeGetRecords;
```

Description

This is FBeforeGetRecords, a member of class TCustomProvider.

124.1.4.4.11 TCustomProvider.FBeforeRowRequest Field

Pascal

```
FBeforeRowRequest: TRemoteEvent;
```

C++

```
TRemoteEvent FBeforeRowRequest;
```

Description

This is FBeforeRowRequest, a member of class TCustomProvider.

124.1.4.4.12 TCustomProvider.FExported Field

Pascal

```
FExported: Boolean;
```

C++

```
Boolean FExported;
```

Description

This is FExported, a member of class TCustomProvider.

124.1.4.4.13 TCustomProvider.FOnDataRequest Field

Pascal

```
FOnDataRequest: TDataRequestEvent;
```

C++

```
TDataRequestEvent FOnDataRequest;
```

Description

This is FOnDataRequest, a member of class TCustomProvider.

124.1.4.4.14 TCustomProvider.FOnValidate Field

Pascal

```
FOnValidate: TOnValidate;
```

C++

```
TOnValidate FOnValidate;
```

Description

This is FOnValidate, a member of class TCustomProvider.

124.1.4.4.15 TCustomProvider.FOwnerData Field

Pascal

```
FOwnerData: OleVariant;
```

C++

```
OleVariant FOwnerData;
```

Description

This is FOwnerData, a member of class TCustomProvider.

124.1.4.5 TCustomProvider Methods

The methods of the TCustomProvider class are listed here.

Protected Methods

	Name	Description
	DoAfterApplyUpdates (see page 18460)	Generates an AfterApplyUpdates (see page 18475) event.
	DoAfterExecute (see page 18461)	Generates an AfterExecute (see page 18476) event.
	DoAfterGetParams (see page 18461)	Generates an AfterGetParams (see page 18477) event.
	DoAfterGetRecords (see page 18462)	Generates an AfterGetRecords (see page 18477) event.
	DoAfterRowRequest (see page 18462)	Generates an AfterRowRequest (see page 18478) event.
	DoApplyUpdates (see page 18463)	This is DoApplyUpdates, a member of class TCustomProvider.
	DoBeforeApplyUpdates (see page 18463)	Generates a BeforeApplyUpdates (see page 18479) event.
	DoBeforeExecute (see page 18464)	Generates a BeforeExecute (see page 18479) event.
	DoBeforeGetParams (see page 18464)	Generates a BeforeGetParams (see page 18480) event.
	DoBeforeGetRecords (see page 18465)	Generates a BeforeGetRecords (see page 18480) event.
	DoBeforeRowRequest (see page 18465)	Generates a BeforeRowRequest (see page 18482) event.
	DoGetRecords (see page 18466)	This is DoGetRecords, a member of class TCustomProvider.
	InternalApplyUpdates (see page 18469)	Provides the underlying implementation of the ApplyUpdates (see page 18459) method.
	InternalExecute (see page 18470)	Provides the underlying implementation of the Execute (see page 18466) method.
	InternalGetParams (see page 18471)	Returns the requested parameter values as a Variant array.
	InternalGetRecords (see page 18471)	Provides the underlying implementation of the GetRecords (see page 18468) method.
	InternalRowRequest (see page 18472)	Provides the underlying implementation of the RowRequest (see page 18472) method.
	SetActiveUpdateException (see page 18473)	This is SetActiveUpdateException, a member of class TCustomProvider.

Public Methods

	Name	Description
	ApplyUpdates (see page 18459)	This is the overview for the ApplyUpdates method overload.
	DataRequest (see page 18460)	Generates an OnDataRequest (see page 18482) event.
	Execute (see page 18466)	Executes a query, stored procedure, or custom SQL statement.
	GetParams (see page 18467)	Returns the current parameter values.
	GetRecords (see page 18468)	This is the overview for the GetRecords method overload.
	RowRequest (see page 18472)	Returns information from a specified record.

Strict private Methods

	Name	Description
	GetData (see page 18467)	This is GetData, a member of class TCustomProvider.

124.1.4.5.1 ApplyUpdates Method

This is the overview for the ApplyUpdates method overload.

Overload List

	Name	Description
	TCustomProvider.ApplyUpdates (OleVariant, Integer, Integer) (see page 18459)	This is ApplyUpdates, a member of class TCustomProvider.
	TCustomProvider.ApplyUpdates (OleVariant, Integer, Integer, OleVariant) (see page 18460)	This is ApplyUpdates, a member of class TCustomProvider.

124.1.4.5.1.1 TCustomProvider.ApplyUpdates Method (OleVariant, Integer, Integer)

Pascal

```
function ApplyUpdates(const Delta: OleVariant; MaxErrors: Integer; out ErrorCount: Integer):
```

```
OleVariant; overload;
```

C++

```
__fastcall OleVariant ApplyUpdates(const OleVariant Delta, int MaxErrors, int ErrorCount);
```

Description

This is ApplyUpdates, a member of class TCustomProvider.

124.1.4.5.1.2 TCustomProvider.ApplyUpdates Method (OleVariant, Integer, Integer, OleVariant)

Pascal

```
function ApplyUpdates(const Delta: OleVariant; MaxErrors: Integer; out ErrorCount: Integer;
var OwnerData: OleVariant): OleVariant; overload;
```

C++

```
__fastcall OleVariant ApplyUpdates(const OleVariant Delta, int MaxErrors, int ErrorCount,
OleVariant OwnerData);
```

Description

This is ApplyUpdates, a member of class TCustomProvider.

124.1.4.5.2 TCustomProvider.DataRequest Method

Generates an OnDataRequest (see page 18482) event.

Pascal

```
function DataRequest(Input: OleVariant): OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant DataRequest(OleVariant Input);
```

Description

Write an OnDataRequest (see page 18482) event handler and call the DataRequest function to invoke the event handler at runtime. This function and the event it invokes enable an application to perform custom processing in response to a request for data.

Note: Calling the client dataset's DataRequest method causes a call to the provider's DataRequest method.

See Also

OnDataRequest (see page 18482)

AS_DataRequest (see page 16452)

DataRequest (see page 8404)

Communicating with the Client Dataset

124.1.4.5.3 TCustomProvider.DoAfterApplyUpdates Method

Generates an AfterApplyUpdates (see page 18475) event.

Pascal

```
procedure DoAfterApplyUpdates(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoAfterApplyUpdates(OleVariant OwnerData);
```

Description

The ApplyUpdates (see page 18459) method calls DoAfterApplyUpdates to generate the AfterApplyUpdates (see page 18475) event after applying updates and resolving any update errors that are handled by the provider.

OwnerData (see page 18475) supplies custom information to the AfterApplyUpdates (see page 18475) event handler. This is the same as the outgoing OwnerData (see page 18475) parameter of the BeforeApplyUpdates (see page 18479) event handler. The AfterApplyUpdates (see page 18475) event handler can change this value. The returned value is then passed to the client if it supplied an OwnerData (see page 18475) parameter when calling ApplyUpdates (see page 18459).

See Also

applyupdates (see page 18459)

AfterApplyUpdates (see page 18475)

BeforeApplyUpdates (see page 18479)

124.1.4.5.4 TCustomProvider.DoAfterExecute Method

Generates an AfterExecute (see page 18476) event.

Pascal

```
procedure DoAfterExecute(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoAfterExecute(OleVariant OwnerData);
```

Description

The Execute (see page 18466) method calls DoAfterExecute to generate the AfterExecute (see page 18476) event after executing CommandText or the associated query or stored procedure but before fetching any output parameters.

OwnerData (see page 18475) supplies custom information to the AfterExecute (see page 18476) event handler. This is the same as the outgoing OwnerData (see page 18475) parameter of the BeforeExecute (see page 18479) event handler. The AfterExecute (see page 18476) event handler can change this value. The returned value is then passed to the client.

See Also

Execute (see page 18466)

AfterExecute (see page 18476)

beforeexecute (see page 18479)

124.1.4.5.5 TCustomProvider.DoAfterGetParams Method

Generates an AfterGetParams (see page 18477) event.

Pascal

```
procedure DoAfterGetParams(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoAfterGetParams(OleVariant OwnerData);
```

Description

The GetParams (see page 18467) method calls DoAfterGetParams to generate the AfterGetParams (see page 18477) event after fetching the current parameter values of the associated dataset.

OwnerData (see page 18475) supplies custom information to the AfterGetParams (see page 18477) event handler. This is the same as the outgoing OwnerData (see page 18475) parameter of the BeforeGetParams (see page 18480) event handler. The AfterGetParams (see page 18477) event handler can change this value. The returned value is then passed to the client.

See Also

GetParams (see page 18467)

aftergetparams (see page 18477)

BeforeGetParams (see page 18480)

124.1.4.5.6 TCustomProvider.DoAfterGetRecords Method

Generates an AfterGetRecords (see page 18477) event.

Pascal

```
procedure DoAfterGetRecords(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoAfterGetRecords(OleVariant OwnerData);
```

Description

The GetRecords (see page 18468) method calls DoAfterGetRecords to generate the AfterGetRecords (see page 18477) event after assembling a data packet but before updating the parameters to reflect any output parameters.

OwnerData (see page 18475) supplies custom information to the AfterGetRecords (see page 18477) event handler. This is the same as the outgoing OwnerData (see page 18475) parameter of the BeforeGetRecords (see page 18480) event handler. The AfterGetRecords (see page 18477) event handler can change this value. The returned value is then passed to the client.

See Also

GetRecords (see page 18468)

AfterGetRecords (see page 18477)

BeforeGetRecords (see page 18480)

124.1.4.5.7 TCustomProvider.DoAfterRowRequest Method

Generates an AfterRowRequest (see page 18478) event.

Pascal

```
procedure DoAfterRowRequest(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoAfterRowRequest(OleVariant OwnerData);
```

Description

The RowRequest (see page 18472) method calls DoAfterRowRequest to generate the AfterRowRequest (see page 18478) event after fetching the requested data.

OwnerData (see page 18475) supplies custom information to the AfterRowRequest (see page 18478) event handler. This is the same as the outgoing OwnerData (see page 18475) parameter of the BeforeRowRequest (see page 18482) event handler. The AfterRowRequest (see page 18478) event handler can change this value. The returned value is then passed to the client.

See Also

RowRequest (see page 18472)

AfterRowRequest (see page 18478)

BeforeRowRequest (see page 18482)

124.1.4.5.8 TCustomProvider.DoApplyUpdates Method

Pascal

```
function DoApplyUpdates(const Delta: OleVariant; MaxErrors: Integer; out ErrorCount: Integer;
var OwnerData: OleVariant): OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant DoApplyUpdates(const OleVariant Delta, int MaxErrors, int
ErrorCount, OleVariant OwnerData);
```

Description

This is DoApplyUpdates, a member of class TCustomProvider.

124.1.4.5.9 TCustomProvider.DoBeforeApplyUpdates Method

Generates a BeforeApplyUpdates (see page 18479) event.

Pascal

```
procedure DoBeforeApplyUpdates(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoBeforeApplyUpdates(OleVariant OwnerData);
```

Description

The ApplyUpdates (see page 18459) method calls DoBeforeApplyUpdates to generate the BeforeApplyUpdates (see page 18479) event before it applies the updates in Delta.

Delta is the delta packet containing the updates as received from the client.

MaxErrors is the number of errors tolerated before the update operation should aborted.

OwnerData (see page 18475) supplies custom information to the BeforeApplyUpdates (see page 18479) event handler. The BeforeApplyUpdates (see page 18479) event handler can change this value. The returned value is then passed to the AfterApplyUpdates (see page 18475) event handler after applying the updates in Delta.

As implemented in TCustomProvider (see page 18449), Delta and MaxErrors are ignored.

See Also

applyupdates (see page 18459)

AfterApplyUpdates (see page 18475)

BeforeApplyUpdates (see page 18479)

124.1.4.5.10 TCustomProvider.DoBeforeExecute Method

Generates a BeforeExecute (see page 18479) event.

Pascal

```
procedure DoBeforeExecute(const CommandText: WideString; var Params: OleVariant; var
OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoBeforeExecute(const BSTR CommandText, OleVariant Params, OleVariant
OwnerData);
```

Description

The Execute (see page 18466) method calls DoBeforeExecute to generate the BeforeExecute (see page 18479) event before it executes CommandText or executes an associated query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of an associated query, or the name of a stored procedure that replaces the associated stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

Params contains any parameter values that should be applied to CommandText or to the query or stored procedure before execution.

OwnerData (see page 18475) supplies custom information to the BeforeExecute (see page 18479) event handler. The event handler can change this value. The returned value is then passed to an AfterExecute (see page 18476) event handler and then returned to the caller of the Execute (see page 18466) method.

As implemented in TCustomProvider (see page 18449), CommandText and Params are ignored. Descendant classes apply these values before generating the BeforeExecute (see page 18479) event.

See Also

Execute (see page 18466)

beforeexecute (see page 18479)

AfterExecute (see page 18476)

Options (see page 18440)

124.1.4.5.11 TCustomProvider.DoBeforeGetParams Method

Generates a BeforeGetParams (see page 18480) event.

Pascal

```
procedure DoBeforeGetParams(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoBeforeGetParams(OleVariant OwnerData);
```

Description

The GetParams (see page 18467) method calls DoBeforeGetParams to generate the BeforeGetParams (see page 18480) event before fetching the current parameter values of the associated dataset.

OwnerData (see page 18475) supplies custom information to the BeforeGetParams (see page 18480) event handler. The event handler can change this value. The returned value is then passed to an AfterGetParams (see page 18477) event handler and then returned to the caller of the GetParams (see page 18467) method.

See Also

GetParams (see page 18467)

BeforeGetParams (see page 18480)

aftergetparams (see page 18477)

124.1.4.5.12 TCustomProvider.DoBeforeGetRecords Method

Generates a BeforeGetRecords (see page 18480) event.

Pascal

```
procedure DoBeforeGetRecords(Count: Integer; Options: Integer; const CommandText: WideString;
var Params: OleVariant; var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoBeforeGetRecords(int Count, int Options, const BSTR CommandText,
OleVariant Params, OleVariant OwnerData);
```

Description

The GetRecords (see page 18468) method calls DoBeforeGetRecords to generate the BeforeGetRecords (see page 18480) event before it fetches data and assembles it into a data packet.

Count is the number of records requested. If Count is –1, all records are requested.

Options indicates whether the data packet should include metadata, whether it should start with the first record, and whether the data packet should be encoded in XML. For details, see the GetRecords (see page 18468) method.

CommandText specifies an optional SQL statement that replaces the SQL of an associated query, or the name of a table or stored procedure that replaces the associated table or stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText or if the provider does not have an Options property.

Params contains any parameter values that should be applied to CommandText or to a query or stored procedure before fetching data into a data packet.

OwnerData (see page 18475) supplies custom information to the BeforeGetRecords (see page 18480) event handler. The event handler can change this value. The returned value is then passed to an AfterGetRecords (see page 18477) event handler and then (optionally) returned to the caller of the GetRecords (see page 18468) method.

As implemented in TCustomProvider (see page 18449), Count, Options, CommandText, and Params are ignored. Descendant classes can apply these values before generating the BeforeExecute (see page 18479) event.

See Also

GetRecords (see page 18468)

BeforeGetRecords (see page 18480)

AfterGetRecords (see page 18477)

Options (see page 18440)

124.1.4.5.13 TCustomProvider.DoBeforeRowRequest Method

Generates a BeforeRowRequest (see page 18482) event.

Pascal

```
procedure DoBeforeRowRequest(var OwnerData: OleVariant); virtual;
```

C++

```
virtual __fastcall DoBeforeRowRequest(OleVariant OwnerData);
```

Description

The RowRequest (see page 18472) method calls DoBeforeRowRequest to generate the BeforeRowRequest (see page 18482) event before fetching the requested data.

OwnerData (see page 18475) supplies custom information to the BeforeRowRequest (see page 18482) event handler. The event handler can change this value. The returned value is then passed to an AfterRowRequest (see page 18478) event handler and then returned to the caller of the RowRequest (see page 18472) method.

See Also

RowRequest (see page 18472)

BeforeRowRequest (see page 18482)

AfterRowRequest (see page 18478)

124.1.4.5.14 TCustomProvider.DoGetRecords Method

Pascal

```
function DoGetRecords(Count: Integer; out RecsOut: Integer; Options: Integer; const
CommandText: WideString; var Params: OleVariant; var OwnerData: OleVariant): OleVariant;
virtual;
```

C++

```
virtual __fastcall OleVariant DoGetRecords(int Count, int RecsOut, int Options, const BSTR
CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

This is DoGetRecords, a member of class TCustomProvider.

124.1.4.5.15 TCustomProvider.Execute Method

Executes a query, stored procedure, or custom SQL statement.

Pascal

```
procedure Execute(const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant); virtual;
```

C++

```
virtual __fastcall Execute(const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

Client datasets call Execute through the AS_Execute method of the IAppServer interface. In response, the provider executes an SQL command. If the provider represents a query or stored procedure that does not return a cursor, calling Execute causes the provider to execute the query or stored procedure. If the caller supplies a value for CommandText, Execute may execute that statement instead (see below).

Before executing the appropriate command, Execute generates a BeforeExecute (see page 18479) event. After executing the command, it generates an AfterExecute (see page 18476) event. Finally, it returns any output parameters as the value of the Params parameter.

CommandText is an SQL statement that replaces the SQL of an associated query, or the name of a stored procedure that replaces the associated stored procedure. This parameter is ignored if the Options property does not include

poAllowCommandText.

On entry, Params supplies parameter values for the query, stored procedure, or CommandText string. On exit, Params returns any output parameters.

OwnerData (see page 18475) contains custom information passed to the provider from the client dataset's BeforeExecute (see page 18479) event handler. This information is passed in to the provider's BeforeExecute (see page 18479) event handler and is returned by the provider's AfterExecute (see page 18476) event handler.

See Also

AS_Execute (see page 16453)

beforeexecute (see page 18479)

AfterExecute (see page 18476)

BeforeExecute (see page 8514)

AfterExecute (see page 8504)

Execute (see page 8413)

CommandText (see page 8465)

InternalExecute (see page 18470)

Options (see page 18440)

Communicating with the Client Dataset

124.1.4.5.16 TCustomProvider.GetData Method

Pascal

```
function GetData: OleVariant;
```

C++

```
__fastcall OleVariant GetData();
```

Description

This is GetData, a member of class TCustomProvider.

124.1.4.5.17 TCustomProvider.GetParams Method

Returns the current parameter values.

Pascal

```
function GetParams(var OwnerData: OleVariant): OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant GetParams(OleVariant OwnerData);
```

Description

GetParams returns a variant array of variant arrays that contains the current parameter values for the provider's data source. The first array dimension is an index into all the parameter values. Its value runs from 0 to one less than the number of parameters. The second array dimension has four values: The parameter name (a Delphi string or a C++ AnsiString), the parameter data type (a TFieldType), the parameter type (a TParamType), and the parameter value (a Variant). The values in this array can be applied to a TParams object using the global UnpackParams procedure.

Before fetching the current parameter values, GetParams generates a BeforeGetParams (see page 18480) event, passing the value supplied as the OwnerData (see page 18475) parameter to the event handler. After creating the OleVariant that contains the parameter values, GetParams generates an AfterGetParams (see page 18477) event, passing in the current value of OwnerData (see page 18475). Finally, the value of OwnerData (see page 18475) returned by the AfterGetParams (see page 18477) is returned in the OwnerData (see page 18475) parameter.

See Also

FetchParams (see page 8415)

ParamType (see page 7431)

Name (see page 7430)

DataType (see page 7427)

Value (see page 7433)

AS_GetParams (see page 16454)

UnpackParams (see page 8534)

BeforeGetParams (see page 18480)

aftergetparams (see page 18477)

InternalGetParams (see page 18471)

Communicating with the Client Dataset

124.1.4.5.18 GetRecords Method

This is the overview for the GetRecords method overload.

Overload List

	Name	Description
	TCustomProvider.GetRecords (Integer, Integer, Integer) (see page 18468)	Retrieves a data packet that represents the provider's associated data.
	TCustomProvider.GetRecords (Integer, Integer, Integer, WideString, OleVariant, OleVariant) (see page 18468)	Retrieves a data packet that represents the provider's associated data.

124.1.4.5.18.1 TCustomProvider.GetRecords Method (Integer, Integer, Integer)

Retrieves a data packet that represents the provider's associated data.

Pascal

```
function GetRecords(Count: Integer; out RecsOut: Integer; Options: Integer): OleVariant;
overload;
function GetRecords(Count: Integer; out RecsOut: Integer; Options: Integer; const CommandText:
WideString; var Params: OleVariant; var OwnerData: OleVariant): OleVariant; overload;
```

C++

```
__fastcall OleVariant GetRecords(int Count, int RecsOut, int Options);
__fastcall OleVariant GetRecords(int Count, int RecsOut, int Options, const BSTR CommandText,
OleVariant Params, OleVariant OwnerData);
```

Description

Call GetRecords to retrieve a data packet containing a specified number of records. GetRecords generates a BeforeGetRecords

(see page 18480) event, assembles a data packet, and then generates an AfterGetRecords (see page 18477) event.

Count indicates the number of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only the specified number of records are retrieved.

RecsOut returns the actual number of records in the data packet.

Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants. For example, the following value requests metadata along with a data packet encoded in XML:

```
MetaDataOption + XMLOption
MetaDataOption || XMLOption
```

CommandText specifies an optional SQL statement that replaces the SQL of an associated query, or the name of a table or stored procedure that replaces the associated table or stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText or if the provider does not have an Options property.

Params specifies any parameter values that should be applied to CommandText or to an associated query or stored procedure. If the provider is associated with a TTable, Params specifies field values that filter the result set. On exit from GetRecords, Params returns any output parameters.

OwnerData (see page 18475) contains custom information that is supplied by a client dataset's BeforeGetRecords (see page 18480) event handler. This information is passed to the provider's BeforeGetRecords (see page 18480) event handler and is returned by the provider's AfterGetRecords (see page 18477) event handler.

Records are returned as a data packet in a Variant.

Note: GetRecords always generates the BeforeGetRecords (see page 18480) and AfterGetRecords (see page 18477) events, even if you use the syntax that does not include an OwnerData (see page 18475) parameter. In that case, the value of OwnerData (see page 18475) is nil (Delphi) or NULL (C++) on entry to the BeforeGetRecords (see page 18480) event handler and is discarded on exit from the AfterGetRecords (see page 18477) event handler.

Tip: If the provider is part of a stateless application server, there is no guarantee that the provider will be positioned on the desired record before it creates a data packet. Use a BeforeGetRecords (see page 18480) event handler to position the cursor before the call to GetRecords.

See Also

DataRequest (see page 18460)

Data (see page 18474)

BeforeGetRecords (see page 18480)

AS_GetRecords (see page 16455)

AfterGetRecords (see page 18477)

InternalGetRecords (see page 18471)

BeforeGetRecords (see page 8515)

Communicating with the Client Dataset

Responding to Client Data Requests

124.1.4.5.19 TCustomProvider.InternalApplyUpdates Method

Provides the underlying implementation of the ApplyUpdates (see page 18459) method.

Pascal

```
function InternalApplyUpdates(const Delta: OleVariant; MaxErrors: Integer; out ErrorCount:
Integer): OleVariant; virtual; abstract;
```

C++

```
virtual __fastcall OleVariant InternalApplyUpdates(const OleVariant Delta, int MaxErrors, int
ErrorCount) = 0;
```

Description

The ApplyUpdates (see page 18459) method calls InternalApplyUpdates to apply the updates contained in a delta packet after generating the BeforeApplyUpdates (see page 18479) event and before generating the AfterApplyUpdates (see page 18475) event. The InternalApplyUpdates method of TCustomProvider (see page 18449) is abstract (Delphi) or pure virtual (C++), meaning that it has no implementation. Descendant classes override this method to apply updates, resolve update errors, and return any updates that could not be applied.

Delta is the delta packet containing insertions, deletions, and modifications to be applied.

MaxErrors indicates the maximum number of errors that can be applied before stopping the update operation. If MaxErrors is –1, the provider tries to apply all updates.

ErrorCount returns the number of errors encountered during the update operation.

InternalApplyUpdates returns a data packet of updates that could not be applied to the database as an OleVariant.

See Also

applyupdates (see page 18459)

124.1.4.5.20 TCustomProvider.InternalExecute Method

Provides the underlying implementation of the Execute (see page 18466) method.

Pascal

```
procedure InternalExecute(const CommandText: WideString; var Params: OleVariant); virtual;
```

C++

```
virtual __fastcall InternalExecute(const BSTR CommandText, OleVariant Params);
```

Description

The Execute (see page 18466) method calls InternalExecute to execute the appropriate SQL statement after calling DoBeforeExecute (see page 18464) to generate a BeforeExecute (see page 18479) event and apply the CommandText and Params. As Implemented in TCustomProvider (see page 18449), InternalExecute does nothing. Descendants override this method to pass the execute command on to an associated dataset.

CommandText is an SQL statement that replaces the SQL of an associated query, or the name of a stored procedure that replaces the associated stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

On entry, Params supplies parameter values for the query, stored procedure, or CommandText string. On exit, Params returns any output parameters.

See Also

Execute (see page 18466)

DoBeforeExecute (see page 18464)

124.1.4.5.21 TCustomProvider.InternalGetParams Method

Returns the requested parameter values as a Variant array.

Pascal

```
function InternalGetParams(Types: TParamTypes = AllParamTypes): OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant InternalGetParams(TParamTypes Types = AllParamTypes);
```

Description

TCustomProvider (see page 18449) uses InternalGetParams internally whenever it needs to assemble the current parameter values into an OleVariant. This provides the underlying implementation of the GetParams (see page 18467) method, as well as assembling parameter values for methods that return output parameters (such as Execute (see page 18466) or GetRecords (see page 18468)).

The Types (see page 23293) parameter limits the parameter values that are returned to a subset of all possible parameter types. If this parameter is omitted, all parameter values are returned.

As implemented in TCustomProvider (see page 18449), InternalGetParams always returns NULL. Descendants override this method to return the requested parameter values.

See Also

GetParams (see page 18467)

Execute (see page 18466)

GetRecords (see page 18468)

124.1.4.5.22 TCustomProvider.InternalGetRecords Method

Provides the underlying implementation of the GetRecords (see page 18468) method.

Pascal

```
function InternalGetRecords(Count: Integer; out RecsOut: Integer; Options: TGetRecordOptions;
const CommandText: WideString; var Params: OleVariant): OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant InternalGetRecords(int Count, int RecsOut, TGetRecordOptions
Options, const BSTR CommandText, OleVariant Params);
```

Description

The GetRecords (see page 18468) method calls InternalGetRecords to assemble a data packet after calling DoBeforeGetRecords (see page 18465) to apply the CommandText and Params and generate a BeforeGetRecords (see page 18480) event. As implemented in TCustomProvider (see page 18449), InternalGetRecords always returns NULL. Descendant classes override this method to fetch the requested records and assemble them into a data packet.

Count indicates the number of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only the specified number of records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It can include grMetaData (include metadata), grReset (start providing records with the first one in the dataset), and grXML (encode the datapacket in XML rather than as an OleVariant) or grXMLUTF8 (same as grXML except extended characters are encoded using UTF8).

CommandText specifies an optional SQL statement that replaces the SQL of an associated query, or the name of a table or

stored procedure that replaces the associated table or stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText or if the provider does not have an Options property.

Params specifies any parameter values that should be applied to CommandText or to an associated query or stored procedure. If the provider is associated with a TTable object, Params specifies field values that filter the result set. On exit from GetRecords (see page 18468), Params returns any output parameters.

Records are returned as a data packet in a Variant.

See Also

GetRecords (see page 18468)

DoBeforeGetRecords (see page 18465)

124.1.4.5.23 TCustomProvider.InternalRowRequest Method

Provides the underlying implementation of the RowRequest (see page 18472) method.

Pascal

```
function InternalRowRequest(const Row: OleVariant; RequestType: TFetchOptions): OleVariant;
virtual;
```

C++

```
virtual __fastcall OleVariant InternalRowRequest(const OleVariant Row, TFetchOptions
RequestType);
```

Description

The RowRequest (see page 18472) method calls InternalRowRequest to assemble the specified information after generating a BeforeRowRequest (see page 18482) event and before generating an AfterRowRequest (see page 18478) event. As implemented in TCustomProvider (see page 18449), InternalRowRequest does nothing. Descendant classes override this method to assemble the requested information into an OleVariant.

The Row parameter is an OleVariant that describes the current record on the client dataset. It enables the provider to locate the record of interest.

InternalRowRequest returns the requested information in a delta packet.

See Also

RowRequest (see page 18472)

124.1.4.5.24 TCustomProvider.RowRequest Method

Returns information from a specified record.

Pascal

```
function RowRequest(const Row: OleVariant; RequestType: Integer; var OwnerData: OleVariant):
OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant RowRequest(const OleVariant Row, int RequestType, OleVariant
OwnerData);
```

Description

RowRequest is called automatically in response to the client dataset's FetchBlobs, FetchDetails, or RefreshRecord method. It generates a BeforeRowRequest (see page 18482) event, assembles the requested information, and generates an AfterRowRequest (see page 18478) event.

The Row parameter is an OleVariant that describes the record about which information is requested. It enables the provider to locate the record of interest.

The RequestType parameter indicates the type of information required. It is an integer version of the TFetchOptions type. (Its an integer because the value must be marshaled over an IAppServer interface). In Delphi, to convert the TFetchOptions value to the appropriate integer, use the Ord function. In C++, to combine TFetchOption values, use the bitwise "or" operator. For example:

```
Ord(foBlobs) + Ord(foDetails);
foBlobs || foDetails
```

RowRequest returns the requested information in a delta packet.

As implemented in TCustomProvider (see page 18449), RowRequest always returns NULL. Descendant classes override this method to return the requested data.

See Also

FetchBlobs (see page 8414)

FetchDetails (see page 8414)

RefreshRecord (see page 8442)

BeforeRowRequest (see page 18482)

AfterRowRequest (see page 18478)

AS_RowRequest (see page 16456)

InternalRowRequest (see page 18472)

Communicating with the Client Dataset

124.1.4.5.25 TCustomProvider.SetActiveUpdateException Method

Pascal

```
procedure SetActiveUpdateException(E: Exception);
```

C++

```
__fastcall SetActiveUpdateException(Exception E);
```

Description

This is SetActiveUpdateException, a member of class TCustomProvider.

124.1.4.6 TCustomProvider Properties

The properties of the TCustomProvider class are listed here.

Protected Properties

	Name	Description
R	ActiveUpdateException (see page 18474)	This is ActiveUpdateException, a member of class TCustomProvider.
	OwnerData (see page 18475)	This is OwnerData, a member of class TCustomProvider.

Public Properties

	Name	Description
R	Data (see page 18474)	Denotes a data packet created by the provider.
	Exported (see page 18474)	Allows client applications to pass calls to the provider when it resides on an application server.

124.1.4.6.1 TCustomProvider.ActiveUpdateException Property

Pascal

```
property ActiveUpdateException: Exception;
```

C++

```
__property Exception ActiveUpdateException;
```

Description

This is ActiveUpdateException, a member of class TCustomProvider.

124.1.4.6.2 TCustomProvider.Data Property

Denotes a data packet created by the provider.

Pascal

```
property Data: OleVariant;
```

C++

```
__property OleVariant Data;
```

Description

Data is a read-only property that provides access to a data packet containing all records. Reading Data calls GetRecords (see page 18468) with a parameter of -1.

As implemented in TCustomProvider (see page 18449), GetRecords (see page 18468) returns NULL. Descendants of TCustomProvider (see page 18449) reimplement the GetRecords (see page 18468) method to implement the Data property.

See Also

GetRecords (see page 18468)

124.1.4.6.3 TCustomProvider.Exported Property

Allows client applications to pass calls to the provider when it resides on an application server.

Pascal

```
property Exported: Boolean;
```

C++

```
__property Boolean Exported;
```

Description

Use Exported to indicate whether the provider is made available to clients of a remote data module that owns it. When Exported is true, client applications can specify the provider as a target for calls made using the IAppServer interface. When Exported is false, IAppServer calls that specify the provider cause an exception.

Note: In addition to the Exported property, the provider must be registered with the remote data module for clients to access it using the data module's IAppServer interface. When the provider's Owner is the remote data module, it is automatically registered. If the provider has a different owner, the server application must explicitly register providers using the remote data module's RegisterProvider method.

See Also

IAppServer (see page 16450)

Providers (see page 19908)

registerprovider (see page 19902)

124.1.4.6.4 TCustomProvider.OwnerData Property

Pascal

```
property OwnerData: OleVariant;
```

C++

```
__property OleVariant OwnerData;
```

Description

This is OwnerData, a member of class TCustomProvider.

124.1.4.7 TCustomProvider Events

The events of the TCustomProvider class are listed here.

Protected Events

	Name	Description
	AfterApplyUpdates (see page 18475)	Occurs after the provider applies updates.
	AfterExecute (see page 18476)	Occurs after the provider passes an execute command to its dataset.
	AfterGetParams (see page 18477)	This is AfterGetParams, a member of class TCustomProvider.
	AfterGetRecords (see page 18477)	Occurs after the provider creates a data packet to send to the client dataset.
	AfterRowRequest (see page 18478)	Occurs after the provider fetches new information about the current record.
	BeforeApplyUpdates (see page 18479)	Occurs before the provider applies updates from a client dataset.
	BeforeExecute (see page 18479)	This is BeforeExecute, a member of class TCustomProvider.
	BeforeGetParams (see page 18480)	Occurs before the provider creates an OleVariant that encodes the current parameter values and sends them to a client dataset.
	BeforeGetRecords (see page 18480)	Occurs before the provider creates a data packet and sends it to a client dataset.
	BeforeRowRequest (see page 18482)	Occurs before the provider creates a delta packet with information about the current record.
	OnDataRequest (see page 18482)	Occurs when a provider calls its DataRequest (see page 18460) method.
	OnValidate (see page 18483)	This is OnValidate, a member of class TCustomProvider.

124.1.4.7.1 TCustomProvider.AfterApplyUpdates Event

Occurs after the provider applies updates.

Pascal

```
property AfterApplyUpdates: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterApplyUpdates;
```

Description

Write an AfterApplyUpdates event handler to return custom information to a client dataset after applying updates. AfterApplyUpdates is part of the mechanism by which a client dataset and a provider communicate information about updates. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_ApplyUpdates method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeApplyUpdates (see page 18479) event, where it can respond to or change custom information that the client dataset sends.

2.If it is a dataset provider, the provider receives an OnUpdateData event where it can edit the updates before they are applied.

3. The provider applies the updates, and attempts to resolve problems.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData (see page 18475) parameter or respond to information from the BeforeApplyUpdates (see page 18479) event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider in its AfterApplyUpdates event handler.

Note: If ApplyUpdates (see page 18459) is called without the OwnerData (see page 18475) parameter, on entry to the event handler the OwnerData (see page 18475) parameter contains the value set in the BeforeApplyUpdates (see page 18479) event handler and it is ignored on exit from the AfterApplyUpdates event handler.

See Also

applyupdates (see page 18459)

AS_ApplyUpdates (see page 16451)

ApplyUpdates (see page 8394)

AfterApplyUpdates (see page 8497)

BeforeApplyUpdates (see page 18479)

OnUpdateData (see page 18443)

Responding to Client Update Requests

Communicating with the Client Dataset

124.1.4.7.2 TCustomProvider.AfterExecute Event

Occurs after the provider passes an execute command to its dataset.

Pascal

```
property AfterExecute: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterExecute;
```

Description

Write an AfterExecute event handler to return custom information to a client dataset after executing a query or stored procedure. AfterExecute is part of the mechanism by which a client dataset and a provider communicate information when executing queries or stored procedures that do not return a cursor. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_Execute method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeExecute (see page 18479) event, where it can respond to or change custom information that the client dataset sends.

2.The provider dispatches its Execute (see page 18466) method to execute a query or stored procedure.

3.The provider receives an AfterExecute event, where it can encode custom information into its OwnerData (see page 18475) parameter or respond to information from the BeforeExecute (see page 18479) event handler.

4.The client dataset receives an AfterExecute event, where it can respond to the custom information returned by the provider in its AfterExecute event handler.

See Also

Execute (see page 18466)

AS_Execute (see page 16453)

Execute (see page 8413)

beforeexecute (see page 18479)

DataSet (see page 18529)

AfterExecute (see page 8504)

Communicating with the Client Dataset

124.1.4.7.3 TCustomProvider.AfterGetParams Event

Pascal

```
property AfterGetParams: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterGetParams;
```

Description

This is AfterGetParams, a member of class TCustomProvider.

124.1.4.7.4 TCustomProvider.AfterGetRecords Event

Occurs after the provider creates a data packet to send to the client dataset.

Pascal

```
property AfterGetRecords: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterGetRecords;
```

Description

Write an AfterGetRecords event handler to send custom information to a client dataset when providing a data packet. AfterGetRecords is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_GetRecords method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeGetRecords (see page 18480) event, where it can respond to or change custom information that the client dataset sends.

2.If it is a dataset provider, the provider receives an OnGetData event where it can edit the data before creating a data packet, and then generates the data packet.

3.The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData (see page 18475) parameter or respond to information from the BeforeGetRecords (see page 18480) event handler.

4.The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider in its AfterGetRecords event handler.

See Also

AS_GetRecords (see page 16455)

OnGetData (see page 18443)

GetRecords (see page 18468)

AfterGetRecords (see page 8506)

BeforeGetRecords (see page 18480)

Communicating with the Client Dataset

Responding to Client Data Requests

124.1.4.7.5 TCustomProvider.AfterRowRequest Event

Occurs after the provider fetches new information about the current record.

Pascal

```
property AfterRowRequest: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterRowRequest;
```

Description

Write an AfterRowRequest event handler to send custom information to a client dataset along with information about the current record. AfterRowRequest is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_RowRequest method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeRowRequest (see page 18482) event, where it can respond to or change custom information that the client dataset sends.

2.The provider creates a delta packet that contains the requested information.

3.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData (see page 18475) parameter or respond to information from the BeforeRowRequest (see page 18482) event handler.

4.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider in its AfterRowRequest event handler.

See Also

AS_RowRequest (see page 16456)

RowRequest (see page 18472)

AfterRowRequest (see page 8509)

BeforeRowRequest (see page 18482)

FetchBlobs (see page 8414)

FetchParams (see page 8415)

RefreshRecord (see page 8442)

Communicating with the Client Dataset

124.1.4.7.6 TCustomProvider.BeforeApplyUpdates Event

Occurs before the provider applies updates from a client dataset.

Pascal

```
property BeforeApplyUpdates: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeApplyUpdates;
```

Description

Write a BeforeApplyUpdates event handler to respond to custom information from a client dataset. BeforeApplyUpdates is part of the mechanism by which a provider and a client dataset communicate information about updates. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset's ApplyUpdates (see page 18459) method is called, the following events occur:

1.The client dataset receives a BeforeApplyUpdates event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeApplyUpdates event, where the OleVariant from the client dataset appears as the OwnerData (see page 18475) parameter. The provider can respond to or change that information before applying the updates.

3.If it is a dataset provider, the provider applies the updates, and receives an OnUpdateError event for any problem records.

4.The provider receives an AfterApplyUpdates (see page 18475) event, where it can encode custom information into its OwnerData (see page 18475) parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates (see page 18475) event, where it can respond to the custom information returned by the provider in its AfterApplyUpdates (see page 18475) event handler.

Note: If ApplyUpdates (see page 18459) is called without the OwnerData (see page 18475) parameter, the OwnerData (see page 18475) parameter is nil (Delphi) or NULL (C++) on entry to the event handler and its value (on exit) is only passed to the AfterApplyUpdates (see page 18475) event handler but not returned to the caller.

See Also

applyupdates (see page 18459)

AS_ApplyUpdates (see page 16451)

ApplyUpdates (see page 8394)

BeforeApplyUpdates (see page 8511)

AfterApplyUpdates (see page 18475)

Responding to Client Update Requests

Communicating with the Client Dataset

124.1.4.7.7 TCustomProvider.BeforeExecute Event

Pascal

```
property BeforeExecute: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeExecute;
```

Description

This is BeforeExecute, a member of class TCustomProvider.

124.1.4.7.8 TCustomProvider.BeforeGetParams Event

Occurs before the provider creates an OleVariant that encodes the current parameter values and sends them to a client dataset.

Pascal

```
property BeforeGetParams: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeGetParams;
```

Description

Write a BeforeGetParams event handler to respond to custom information from a client dataset. BeforeGetParams is part of the mechanism by which a provider and a client dataset communicate information during parameter fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset fetches parameter values from the provider, the following events occur:

1.The client dataset receives a BeforeGetParams event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeGetParams event, where the OleVariant from the client dataset appears as the OwnerData (see page 18475) parameter. The provider can respond to or change that information before it fetches and encodes the parameter values.

3.The provider generates an OleVariant for the current parameter values of its dataset, if appropriate.

4.The provider receives an AfterGetParams (see page 18477) event, where it can encode custom information into its OwnerData (see page 18475) parameter or respond to information from the BeforeGetParams event handler.

5.The client dataset receives an AfterGetParams (see page 18477) event, where it can respond to the custom information returned by the provider in its AfterGetParams (see page 18477) event handler.

See Also

AS_GetParams (see page 16454)

GetParams (see page 18467)

BeforeGetParams (see page 8514)

aftergetparams (see page 18477)

Communicating with the Client Dataset

124.1.4.7.9 TCustomProvider.BeforeGetRecords Event

Occurs before the provider creates a data packet and sends it to a client dataset.

Pascal

```
property BeforeGetRecords: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeGetRecords;
```

Description

Write a BeforeGetRecords event handler to respond to custom information from a client dataset. BeforeGetRecords is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset fetches data from the provider, the following events occur:

1.The client dataset receives a BeforeGetRecords event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeGetRecords event, where the OleVariant from the client dataset appears as the OwnerData (see page 18475) parameter. The provider can respond to or change that information before it creates a data packet.

3.The provider generates the data packet.

4.The provider receives an AfterGetRecords (see page 18477) event, where it can encode custom information into its OwnerData (see page 18475) parameter or respond to information from the BeforeGetRecords event handler.

5.The client dataset receives an AfterGetRecords (see page 18477) event, where it can respond to the custom information returned by the provider in its AfterGetRecords (see page 18477) event handler.

Delphi Examples:

```
{
This example shows the implementation of a provider BeforeGetRecords
event handler and a RemoteDataModule UpdateRegistry method.  The
complete project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TPooledRDM.DataSetProvider1BeforeGetRecords(Sender: TObject;
  var OwnerData: OleVariant);
begin
  try
    Query1.Close;
    Query1.SQL.Text := OwnerData[0];
    if not VarIsNull(OwnerData[1]) and not VarIsEmpty(OwnerData[1]) then
    begin
      Query1.Open;
      if not Query1.Locate(Query1.Fields[0].FieldName, OwnerData[1], []) then
        raise Exception.Create('Record not found');
      Query1.Next;
    end;
  finally
    OwnerData := NULL;
  end;
end;
```

See Also

AS_GetRecords (see page 16455)

GetRecords (see page 18468)

BeforeGetRecords (see page 8515)

AfterGetRecords (see page 18477)

Communicating with the Client Dataset

Responding to Client Data Requests

124.1.4.7.10 TCustomProvider.BeforeRowRequest Event

Occurs before the provider creates a delta packet with information about the current record.

Pascal

```
property BeforeRowRequest: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeRowRequest;
```

Description

Write a BeforeRowRequest event handler to respond to custom information from a client dataset. BeforeRowRequest is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset fetches detail information about the current record or refreshes the current record, the following events occur:

1.The client dataset receives a BeforeRowRequest event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeRowRequest event, where the custom information appears as the OwnerData (see page 18475) parameter. The provider can respond to or change that information before it looks up the requested information.

3.The provider creates a delta packet that contains the requested information.

4.The provider receives an AfterRowRequest (see page 18478) event, where it can encode custom information into its OwnerData (see page 18475) parameter or respond to information from the BeforeRowRequest event handler.

5.The client dataset receives an AfterRowRequest (see page 18478) event, where it can respond to the custom information returned by the provider in its AfterRowRequest (see page 18478) event handler.

See Also

AS_RowRequest (see page 16456)

RowRequest (see page 18472)

AfterRowRequest (see page 18478)

BeforeRowRequest (see page 8520)

FetchBlobs (see page 8414)

FetchParams (see page 8415)

RefreshRecord (see page 8442)

Communicating with the Client Dataset

124.1.4.7.11 TCustomProvider.OnDataRequest Event

Occurs when a provider calls its DataRequest (see page 18460) method.

Pascal

```
property OnDataRequest: TDataRequestEvent;
```

C++

```
__property TDataRequestEvent OnDataRequest;
```

Description

Write an OnDataRequest event handler to do custom processing of data. OnDataRequest has no built-in use. Instead, it allows developers to augment the IAppServer interface so that clients in a multi-tiered application can communicate directly with an individual provider.

If an application does not provide an OnDataRequest event handler, calls to DataRequest (see page 18460) return NULL.

Note: To provide the same sort of custom processing on the client side, use an automation method on the connection component.

See Also

DataRequest (see page 18460)

IAppServer (see page 16450)

Communicating with Providers Using Custom Events

Communicating with the Client Dataset

Responding to Client-generated Events

124.1.4.7.12 TCustomProvider.OnValidate Event

Pascal

```
property OnValidate: TOnValidate;
```

C++

```
__property TOnValidate OnValidate;
```

Description

This is OnValidate, a member of class TCustomProvider.

124.1.5 TCustomResolver Class

TCustomResolver is the base class for components that apply updates on behalf of a provider.

Class Hierarchy

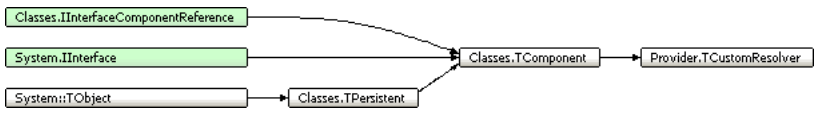

Pascal

```
TCustomResolver = class(TComponent);
```

C++

```
class TCustomResolver : public TComponent;
```

File

Provider

Description

TCustomResolver defines the fundamental properties and methods used by resolver components for applying updates from a client dataset or XML broker and resolving errors that occur as part of that update process.

TCustomResolver has two descendants: TDataSetResolver (see page 18544), which applies updates to a dataset, and TSQLResolver (see page 18589), which applies updates directly to a database server.

Do not create resolver instances. Provider (see page 18496) components create their own TCustomResolver descendants for applying updates.

See Also

TDataSetProvider (see page 18508)

TBaseProvider (see page 18423)

TDataSetResolver (see page 18544)

TSQLResolver (see page 18589)

124.1.5.1 TCustomResolver Members

The following tables list the members exposed by TCustomResolver.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	BeginUpdate (see page 18490)	Provides the interface for a method that performs any initialization before applying updates.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
A	DoDelete (see page 18491)	Defines a method that deletes a record.
A	DoInsert (see page 18491)	Defines a method that inserts a record.
A	DoUpdate (see page 18491)	Defines a method that modifies a record.
V	EndUpdate (see page 18492)	Provides the interface for a method that performs any finalization after applying updates.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleUpdateError (see page 18493)	Manages the resolving process when errors occur while applying updates.

	InitializeConflictBuffer (see page 18493)	Introduces a method for adding information about a failed update that is passed to an OnUpdateError event handler.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	LogUpdateError (see page 18495)	This is LogUpdateError, a member of class TCustomResolver.
	LogUpdateRecord (see page 18495)	Adds records to the error log if their data has been changed by another application.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	ApplyUpdates (see page 18490)	Applies updates received from a provider.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 18487)	Creates and initializes a resolver.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 18487)	Frees an instance of a resolver object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.

	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	FreeTreeData (see page 18492)	This is FreeTreeData, a member of class TCustomResolver.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	InitKeyFields (see page 18494)	Sets the provider flag pfInKey on all key fields in a delta data packet.
V	InitTreeData (see page 18494)	This is InitTreeData, a member of class TCustomResolver.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	InternalBeforeResolve (see page 18494)	Initializes internal data structures once the delta packet has been wrapped in an update tree but before any updates are applied.
V	InternalUpdateRecord (see page 18495)	Manages the updating process once a delta packet is wrapped in a TUpdateTree (see page 18600) object.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	RowRequest (see page 18496)	Returns information from a specified record.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.

R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Provider (see page 18496)	Indicates the provider that uses this resolver to apply updates.
R	UpdateTree (see page 18497)	This is UpdateTree, a member of class TCustomResolver.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

Strict private Fields

	Name	Description
	FErrorCount (see page 18488)	This is FErrorCount, a member of class TCustomResolver.
	FMaxErrors (see page 18488)	This is FMaxErrors, a member of class TCustomResolver.
	FPrevResponse (see page 18488)	This is FPrevResponse, a member of class TCustomResolver.
	FProvider (see page 18489)	This is FProvider, a member of class TCustomResolver.
	FUpdateTree (see page 18489)	This is FUpdateTree, a member of class TCustomResolver.

124.1.5.2 TCustomResolver.Create Constructor

Creates and initializes a resolver.

Pascal

```
constructor Create(AProvider: TBaseProvider); reintroduce;
```

C++

```
__fastcall TCustomResolver(TBaseProvider AProvider);
```

Description

Provider (see page 18496) components call the TCustomResolver (see page 18483) constructor when they need a resolver component for applying updates. The AProvider parameter specifies the provider that creates and uses the resolver.

Do not create resolver instances directly. Resolvers are designed to be created and used by providers, and can not function independently of a provider.

See Also

Resolver (see page 18441)

124.1.5.3 TCustomResolver.Destroy Destructor

Frees an instance of a resolver object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomResolver();
```

Description

Applications rarely have to destroy resolvers explicitly. Once a resolver has been assigned as the Resolver property of a provider, the provider is responsible for freeing the resolver.

See Also

Free (see page 22300)

Resolver (see page 18441)

124.1.5.4 TCustomResolver Fields

The fields of the TCustomResolver class are listed here.

Strict private Fields

	Name	Description
	FErrorCount (see page 18488)	This is FErrorCount, a member of class TCustomResolver.
	FMaxErrors (see page 18488)	This is FMaxErrors, a member of class TCustomResolver.
	FPrevResponse (see page 18488)	This is FPrevResponse, a member of class TCustomResolver.
	FProvider (see page 18489)	This is FProvider, a member of class TCustomResolver.
	FUpdateTree (see page 18489)	This is FUpdateTree, a member of class TCustomResolver.

124.1.5.4.1 TCustomResolver.FErrorCount Field

Pascal

```
FErrorCount: Integer;
```

C++

```
int FErrorCount;
```

Description

This is FErrorCount, a member of class TCustomResolver.

124.1.5.4.2 TCustomResolver.FMaxErrors Field

Pascal

```
FMaxErrors: Integer;
```

C++

```
int FMaxErrors;
```

Description

This is FMaxErrors, a member of class TCustomResolver.

124.1.5.4.3 TCustomResolver.FPrevResponse Field

Pascal

```
FPrevResponse: TResolverResponse;
```

C++

```
TResolverResponse FPrevResponse;
```

Description

This is FPrevResponse, a member of class TCustomResolver.

124.1.5.4.4 TCustomResolver.FProvider Field

Pascal

```
FProvider: TBaseProvider;
```

C++

```
TBaseProvider FProvider;
```

Description

This is FProvider, a member of class TCustomResolver.

124.1.5.4.5 TCustomResolver.FUpdateTree Field

Pascal

```
FUpdateTree: TUpdateTree;
```

C++

```
TUpdateTree FUpdateTree;
```

Description

This is FUpdateTree, a member of class TCustomResolver.

124.1.5.5 TCustomResolver Methods

The methods of the TCustomResolver class are listed here.

Protected Methods

	Name	Description
V	BeginUpdate (see page 18490)	Provides the interface for a method that performs any initialization before applying updates.
A	DoDelete (see page 18491)	Defines a method that deletes a record.
A	DoInsert (see page 18491)	Defines a method that inserts a record.
A	DoUpdate (see page 18491)	Defines a method that modifies a record.
V	EndUpdate (see page 18492)	Provides the interface for a method that performs any finalization after applying updates.
V	HandleUpdateError (see page 18493)	Manages the resolving process when errors occur while applying updates.
A	InitializeConflictBuffer (see page 18493)	Introduces a method for adding information about a failed update that is passed to an OnUpdateError event handler.
V	LogUpdateError (see page 18495)	This is LogUpdateError, a member of class TCustomResolver.
V	LogUpdateRecord (see page 18495)	Adds records to the error log if their data has been changed by another application.

Public Methods

	Name	Description
V	ApplyUpdates (see page 18490)	Applies updates received from a provider.
V	FreeTreeData (see page 18492)	This is FreeTreeData, a member of class TCustomResolver.
V	InitKeyFields (see page 18494)	Sets the provider flag pfInKey on all key fields in a delta data packet.
V	InitTreeData (see page 18494)	This is InitTreeData, a member of class TCustomResolver.
V	InternalBeforeResolve (see page 18494)	Initializes internal data structures once the delta packet has been wrapped in an update tree but before any updates are applied.
V	InternalUpdateRecord (see page 18495)	Manages the updating process once a delta packet is wrapped in a TUpdateTree (see page 18600) object.
V	RowRequest (see page 18496)	Returns information from a specified record.

124.1.5.5.1 TCustomResolver.ApplyUpdates Method

Applies updates received from a provider.

Pascal

```
function ApplyUpdates(const Delta: OleVariant; MaxErrors: Integer; out ErrorCount: Integer):
OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant ApplyUpdates(const OleVariant Delta, int MaxErrors, int
ErrorCount);
```

Description

Providers call ApplyUpdates to apply updates received from a client dataset or XML broker.

ApplyUpdates generates an OnUpdateData event on the associated provider, and then applies the updates passed in as the Delta parameter, returning an OleVariant that contains any updates that could not be applied.

The Delta parameter specifies a delta packet that contains the insertions, deletions, and modifications to be applied. MaxErrors indicates the maximum number of errors to permit before aborting the update operation. ErrorCount returns the actual number of errors encountered during the update operation. If ErrorCount is greater than MaxErrors, (or an OnUpdateError event returns rrAbort), all updates that have been successfully applied so far are rolled back, and the update process is aborted.

If the associated provider has an OnUpdateError event handler, it is called automatically whenever a record in Delta cannot be updated. In the OnUpdateError handler, applications can correct error conditions and try reapplying an update before returning unsuccessfully updated records back to the client dataset or XML broker.

See Also

OnUpdateData (see page 18443)

OnUpdateError (see page 18444)

applyupdates (see page 18459)

124.1.5.5.2 TCustomResolver.BeginUpdate Method

Provides the interface for a method that performs any initialization before applying updates.

Pascal

```
procedure BeginUpdate; virtual;
```

C++

```
virtual __fastcall BeginUpdate();
```

Description

The ApplyUpdates (see page 18490) method calls BeginUpdate before applying updates in a delta packet.

As implemented in TCustomResolver (see page 18483), BeginUpdate does nothing. Descendants that need take specific action before applying updates override this method.

See Also

EndUpdate (see page 18492)

ApplyUpdates (see page 18490)

124.1.5.5.3 TCustomResolver.DoDelete Method

Defines a method that deletes a record.

Pascal

```
procedure DoDelete(Tree: TUpdateTree); virtual; abstract;
```

C++

```
virtual __fastcall DoDelete(TUpdateTree Tree) = 0;
```

Description

DoDelete is one of several helper methods internal to the ApplyUpdates (see page 18490) method. These helper methods perform the actual updates in the delta packet. ApplyUpdates (see page 18490) wraps the delta packet in a TUpdateTree (see page 18600) object. For each record that must be deleted, the resolver calls DoDelete.

TCustomResolver (see page 18483) does not implement the DoDelete method. Descendants override this method to delete records in their own specialized way.

See Also

ApplyUpdates (see page 18490)

DoInsert (see page 18491)

DoUpdate (see page 18491)

124.1.5.5.4 TCustomResolver.DoInsert Method

Defines a method that inserts a record.

Pascal

```
procedure DoInsert(Tree: TUpdateTree); virtual; abstract;
```

C++

```
virtual __fastcall DoInsert(TUpdateTree Tree) = 0;
```

Description

DoInsert is one of several helper methods internal to the ApplyUpdates (see page 18490) method. These helper methods perform the actual updates in the delta packet. ApplyUpdates (see page 18490) wraps the delta packet in a TUpdateTree (see page 18600) object. For each record that must be inserted, the resolver calls DoInsert.

TCustomResolver (see page 18483) does not implement the DoInsert method. Descendants override this method to insert records in their own specialized way.

See Also

ApplyUpdates (see page 18490)

DoDelete (see page 18491)

DoUpdate (see page 18491)

124.1.5.5.5 TCustomResolver.DoUpdate Method

Defines a method that modifies a record.

Pascal

```
procedure DoUpdate(Tree: TUpdateTree); virtual; abstract;
```

C++

```
virtual __fastcall DoUpdate(TUpdateTree Tree) = 0;
```

Description

DoUpdate is one of several helper methods internal to the ApplyUpdates (see page 18490) method. These helper methods perform the actual updates passed in the delta packet. ApplyUpdates (see page 18490) wraps the delta packet in a TUpdateTree (see page 18600) object. For each record that must be modified, the resolver calls DoUpdate.

TCustomResolver (see page 18483) does not implement the DoUpdate method. Descendants override this method to insert records in their own specialized way.

See Also

ApplyUpdates (see page 18490)

DoDelete (see page 18491)

DoInsert (see page 18491)

124.1.5.5.6 TCustomResolver.EndUpdate Method

Provides the interface for a method that performs any finalization after applying updates.

Pascal

```
procedure EndUpdate; virtual;
```

C++

```
virtual __fastcall EndUpdate();
```

Description

The ApplyUpdates (see page 18490) method calls EndUpdate after applying updates.

As implemented in TCustomResolver (see page 18483), EndUpdate does nothing. Descendants that need take specific action after applying updates override this method.

See Also

BeginUpdate (see page 18490)

ApplyUpdates (see page 18490)

124.1.5.5.7 TCustomResolver.FreeTreeData Method

Pascal

```
procedure FreeTreeData(Tree: TUpdateTree); virtual;
```

C++

```
virtual __fastcall FreeTreeData(TUpdateTree Tree);
```

Description

This is FreeTreeData, a member of class TCustomResolver.

124.1.5.5.8 TCustomResolver.HandleUpdateError Method

Manages the resolving process when errors occur while applying updates.

Pascal

```
function HandleUpdateError(Tree: TUpdateTree; E: EUpdateError; var MaxErrors: Integer; var
ErrorCount: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean HandleUpdateError(TUpdateTree Tree, EUpdateError E, int MaxErrors,
int ErrorCount);
```

Description

When a resolver encounters errors in applying updates, it calls HandleUpdateError to determine how to proceed. HandleUpdateError generates an OnUpdateError event on the associated provider so that applications have an opportunity to correct update errors. Depending on the response provided by the OnUpdateError handler (if any), HandleUpdateError logs the unresolved error or signals that the resolver should try to apply the (corrected) record again.

The Tree parameter specifies the TUpdateTree (see page 18600) that wraps all the updates currently being applied. The MaxErrors parameter indicates the maximum number of errors that the resolver will tolerate before the update operation is rolled back. The ErrorCount parameter specifies how many errors have been encountered so far.

HandleUpdateError returns true if the record has been corrected and the resolver should try to reapply the update. It returns false if the record is added to the error log.

See Also

OnUpdateError (see page 18444)

ApplyUpdates (see page 18490)

124.1.5.5.9 TCustomResolver.InitializeConflictBuffer Method

Introduces a method for adding information about a failed update that is passed to an OnUpdateError event handler.

Pascal

```
procedure InitializeConflictBuffer(Tree: TUpdateTree); virtual; abstract;
```

C++

```
virtual __fastcall InitializeConflictBuffer(TUpdateTree Tree) = 0;
```

Description

The HandleUpdateError (see page 18493) method calls InitializeConflictBuffer before generating an OnUpdateError event. This ensures that the client dataset passed to an OnUpdateError event handler represents the update that caused the problem, with appropriate values for field's CurValue property.

Descendants of TCustomResolver (see page 18483) implement this method to obtain information from the dataset or server to which they are applying updates.

See Also

HandleUpdateError (see page 18493)

OnUpdateError (see page 18444)

124.1.5.5.10 TCustomResolver.InitKeyFields Method

Sets the provider flag pfInKey on all key fields in a delta data packet.

Pascal

```
procedure InitKeyFields(Tree: TUpdateTree; ADelta: TPacketDataSet); virtual;
```

C++

```
virtual __fastcall InitKeyFields(TUpdateTree Tree, TPacketDataSet ADelta);
```

Description

InitKeyFields is used internally to mark fields from a delta packet that can be used to uniquely identify records in the underlying dataset.

The Tree parameter is the internal TUpdateTree (see page 18600) that the resolver uses to store information about the delta packet. The ADelta parameter is the delta packet of changes to apply. For every field in this delta packet that acts as a key value to uniquely identify records, InitKeyFields adds pfInKey to its ProviderFlags property.

See Also

ProviderFlags (see page 7176)

124.1.5.5.11 TCustomResolver.InitTreeData Method

Pascal

```
procedure InitTreeData(Tree: TUpdateTree); virtual;
```

C++

```
virtual __fastcall InitTreeData(TUpdateTree Tree);
```

Description

This is InitTreeData, a member of class TCustomResolver.

124.1.5.5.12 TCustomResolver.InternalBeforeResolve Method

Initializes internal data structures once the delta packet has been wrapped in an update tree but before any updates are applied.

Pascal

```
procedure InternalBeforeResolve(Tree: TUpdateTree); virtual;
```

C++

```
virtual __fastcall InternalBeforeResolve(TUpdateTree Tree);
```

Description

The ApplyUpdates (see page 18490) method wraps delta packets in a TUpdateTree (see page 18600) object. Once this is complete, InternalBeforeResolve performs any other initialization before the updates are applied.

As implemented in TCustomResolver (see page 18483), InternalBeforeResolve does nothing. Descendants override this method to perform class-specific initialization.

See Also

ApplyUpdates (see page 18490)

inittreedata (see page 18494)

124.1.5.5.13 TCustomResolver.InternalUpdateRecord Method

Manages the updating process once a delta packet is wrapped in a TUpdateTree (see page 18600) object.

Pascal

```
function InternalUpdateRecord(Tree: TUpdateTree): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean InternalUpdateRecord(TUpdateTree Tree);
```

Description

InternalUpdateRecord orchestrates the updating process, ensuring that updates are applied in the correct order and that errors are resolved or logged. For each record that must be updated, InternalUpdateRecord

1.Generates a BeforeUpdateRecord event.

2.Applies the update.

3.Generates an AfterUpdateRecord event if the update is successful, or calls HandleUpdateError (see page 18493) to generate an OnUpdateError event.

See Also

ApplyUpdates (see page 18490)

beforeupdaterecord (see page 18442)

AfterUpdateRecord (see page 18442)

OnUpdateError (see page 18444)

HandleUpdateError (see page 18493)

124.1.5.5.14 TCustomResolver.LogUpdateError Method

Pascal

```
procedure LogUpdateError(Tree: TUpdateTree; E: EUpdateError; Response: TResolverResponse);
virtual;
```

C++

```
virtual __fastcall LogUpdateError(TUpdateTree Tree, EUpdateError E, TResolverResponse
Response);
```

Description

This is LogUpdateError, a member of class TCustomResolver.

124.1.5.5.15 TCustomResolver.LogUpdateRecord Method

Adds records to the error log if their data has been changed by another application.

Pascal

```
procedure LogUpdateRecord(Tree: TUpdateTree); virtual;
```

C++

```
virtual __fastcall LogUpdateRecord(TUpdateTree Tree);
```

Description

When the InternalUpdateRecord (see page 18495) method detects that a record includes other changes than those specified by an update in the delta packet, it calls LogUpdateRecord to add the changed and updated record to the error log so that the client dataset or XML broker can refresh the updated record. This error log is eventually returned by the ApplyUpdates (see page 18490) method when the update operation ends.

The Tree parameter represents the set of updates being applied.

See Also

logupdateerror (see page 18495)

InternalUpdateRecord (see page 18495)

124.1.5.5.16 TCustomResolver.RowRequest Method

Returns information from a specified record.

Pascal

```
function RowRequest(Row: OleVariant; Options: TFetchOptions): OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant RowRequest(OleVariant Row, TFetchOptions Options);
```

Description

RowRequest is called automatically in response to the associated provider's RowRequest method.

The Row parameter is an OleVariant that describes the record about which information is requested. The resolver uses this to locate the desired record.

The requested information is returned as a delta packet.

See Also

RowRequest (see page 18472)

124.1.5.6 TCustomResolver Properties

The properties of the TCustomResolver class are listed here.

Public Properties

	Name	Description
R	Provider (see page 18496)	Indicates the provider that uses this resolver to apply updates.
R	UpdateTree (see page 18497)	This is UpdateTree, a member of class TCustomResolver.

124.1.5.6.1 TCustomResolver.Provider Property

Indicates the provider that uses this resolver to apply updates.

Pascal

```
property Provider: TBaseProvider;
```

C++

```
__property TBaseProvider Provider;
```

Description

Provider is a read-only property that points to the provider that uses this resolver for applying updates. The Provider property is set in the constructor.

Resolvers use the Provider property internally to obtain information supplied in the provider's properties. They also use it to generate events on the provider at appropriate points in the update process.

See Also

Create (see page 18487)

Resolver (see page 18441)

124.1.5.6.2 TCustomResolver.UpdateTree Property

Pascal

```
property UpdateTree: TUpdateTree;
```

C++

```
__property TUpdateTree UpdateTree;
```

Description

This is UpdateTree, a member of class TCustomResolver.

124.1.6 TDataPacketWriter Class

Class Hierarchy

Pascal

```
TDataPacketWriter = class(TCustomPacketWriter);
```

C++

```
class TDataPacketWriter : public TCustomPacketWriter;
```

File

Provider

Description

This is class Provider.TDataPacketWriter.

124.1.6.1 TDataPacketWriter Members

The following tables list the members exposed by TDataPacketWriter.

Protected Fields

	Name	Description
	FBuffer (see page 18447)	This is FBuffer, a member of class TCustomPacketWriter.

Protected Methods

	Name	Description
V	AddAttribute (see page 18448)	This is AddAttribute, a member of class TCustomPacketWriter.
V	AddColumn (see page 18502)	This is AddColumn, a member of class TDataPacketWriter.
V	AddConstraints (see page 18502)	This is AddConstraints, a member of class TDataPacketWriter.
V	AddDataSetAttributes (see page 18502)	This is AddDataSetAttributes, a member of class TDataPacketWriter.
	AddExtraFieldProps (see page 18502)	This is AddExtraFieldProps, a member of class TDataPacketWriter.
V	AddFieldLinks (see page 18503)	This is AddFieldLinks, a member of class TDataPacketWriter.
V	AddIndexDefs (see page 18503)	This is AddIndexDefs, a member of class TDataPacketWriter.
	Check (see page 18448)	This is Check, a member of class TCustomPacketWriter.
	FreeInfoRecords (see page 18503)	This is FreeInfoRecords, a member of class TDataPacketWriter.
	GetFieldIdx (see page 18503)	This is GetFieldIdx, a member of class TDataPacketWriter.
	InitPutProcs (see page 18504)	This is InitPutProcs, a member of class TDataPacketWriter.
	PutADTField (see page 18504)	This is PutADTField, a member of class TDataPacketWriter.
	PutArrayField (see page 18504)	This is PutArrayField, a member of class TDataPacketWriter.
	PutBlobField (see page 18504)	This is PutBlobField, a member of class TDataPacketWriter.
	PutCalcField (see page 18505)	This is PutCalcField, a member of class TDataPacketWriter.
	PutDataSetField (see page 18505)	This is PutDataSetField, a member of class TDataPacketWriter.
	PutField (see page 18505)	This is PutField, a member of class TDataPacketWriter.
	PutStringField (see page 18505)	This is PutStringField, a member of class TDataPacketWriter.
	PutVarBytesField (see page 18505)	This is PutVarBytesField, a member of class TDataPacketWriter.
	PutWideStringField (see page 18506)	This is PutWideStringField, a member of class TDataPacketWriter.
	RefreshPutProcs (see page 18506)	This is RefreshPutProcs, a member of class TDataPacketWriter.
	SetDSWriter (see page 18448)	This is SetDSWriter, a member of class TCustomPacketWriter.
V	WriteDataSet (see page 18506)	This is WriteDataSet, a member of class TDataPacketWriter.
V	WriteMetaData (see page 18507)	This is WriteMetaData, a member of class TDataPacketWriter.

Protected Properties

	Name	Description
R	DSWriter (see page 18449)	This is DSWriter, a member of class TCustomPacketWriter.

Public Events

	Name	Description
	OnGetParams (see page 18508)	This is OnGetParams, a member of class TDataPacketWriter.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 18446)	Constructs an object and initializes its data before the object is first used.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 18499)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetDataPacket (see page 18503)	This is GetDataPacket, a member of class TDataPacketWriter.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Reset (see page 18506)	This is Reset, a member of class TDataPacketWriter.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Constraints (see page 18507)	This is Constraints, a member of class TDataPacketWriter.
	Options (see page 18507)	This is Options, a member of class TDataPacketWriter.
	PacketOptions (see page 18507)	This is PacketOptions, a member of class TDataPacketWriter.

Strict private Fields

	Name	Description
	FConstraints (see page 18500)	This is FConstraints, a member of class TDataPacketWriter.
	FIDSWriter (see page 18447)	This is FIDSWriter, a member of class TCustomPacketWriter.
	FOnGetParams (see page 18500)	This is FOnGetParams, a member of class TDataPacketWriter.
	FOptions (see page 18500)	This is FOptions, a member of class TDataPacketWriter.
	FPacketOptions (see page 18501)	This is FPacketOptions, a member of class TDataPacketWriter.
	FPutFieldInfo (see page 18501)	This is FPutFieldInfo, a member of class TDataPacketWriter.

124.1.6.2 TDataPacketWriter.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataPacketWriter();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually

allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

124.1.6.3 TDataPacketWriter Fields

The fields of the TDataPacketWriter class are listed here.

Strict private Fields

	Name	Description
	FConstraints (see page 18500)	This is FConstraints, a member of class TDataPacketWriter.
	FOnGetParams (see page 18500)	This is FOnGetParams, a member of class TDataPacketWriter.
	FOptions (see page 18500)	This is FOptions, a member of class TDataPacketWriter.
	FPacketOptions (see page 18501)	This is FPacketOptions, a member of class TDataPacketWriter.
	FPutFieldInfo (see page 18501)	This is FPutFieldInfo, a member of class TDataPacketWriter.

124.1.6.3.1 TDataPacketWriter.FConstraints Field

Pascal

```
FConstraints: Boolean;
```

C++

```
Boolean FConstraints;
```

Description

This is FConstraints, a member of class TDataPacketWriter.

124.1.6.3.2 TDataPacketWriter.FOnGetParams Field

Pascal

```
FOnGetParams: TGetParamsEvent;
```

C++

```
TGetParamsEvent FOnGetParams;
```

Description

This is FOnGetParams, a member of class TDataPacketWriter.

124.1.6.3.3 TDataPacketWriter.FOptions Field

Pascal

```
FOptions: TProviderOptions;
```

C++

```
TProviderOptions FOptions;
```

Description

This is FOptions, a member of class TDataPacketWriter.

124.1.6.3.4 TDataPacketWriter.FPacketOptions Field

Pascal

```
FPacketOptions: TGetRecordOptions;
```

C++

```
TGetRecordOptions FPacketOptions;
```

Description

This is FPacketOptions, a member of class TDataPacketWriter.

124.1.6.3.5 TDataPacketWriter.FPutFieldInfo Field

Pascal

```
FPutFieldInfo: TInfoArray;
```

C++

```
TInfoArray FPutFieldInfo;
```

Description

This is FPutFieldInfo, a member of class TDataPacketWriter.

124.1.6.4 TDataPacketWriter Methods

The methods of the TDataPacketWriter class are listed here.

Protected Methods

	Name	Description
	AddColumn (see page 18502)	This is AddColumn, a member of class TDataPacketWriter.
	AddConstraints (see page 18502)	This is AddConstraints, a member of class TDataPacketWriter.
	AddDataSetAttributes (see page 18502)	This is AddDataSetAttributes, a member of class TDataPacketWriter.
	AddExtraFieldProps (see page 18502)	This is AddExtraFieldProps, a member of class TDataPacketWriter.
	AddFieldLinks (see page 18503)	This is AddFieldLinks, a member of class TDataPacketWriter.
	AddIndexDefs (see page 18503)	This is AddIndexDefs, a member of class TDataPacketWriter.
	FreeInfoRecords (see page 18503)	This is FreeInfoRecords, a member of class TDataPacketWriter.
	GetFieldIdx (see page 18503)	This is GetFieldIdx, a member of class TDataPacketWriter.
	InitPutProcs (see page 18504)	This is InitPutProcs, a member of class TDataPacketWriter.
	PutADTField (see page 18504)	This is PutADTField, a member of class TDataPacketWriter.
	PutArrayField (see page 18504)	This is PutArrayField, a member of class TDataPacketWriter.
	PutBlobField (see page 18504)	This is PutBlobField, a member of class TDataPacketWriter.
	PutCalcField (see page 18505)	This is PutCalcField, a member of class TDataPacketWriter.
	PutDataSetField (see page 18505)	This is PutDataSetField, a member of class TDataPacketWriter.
	PutField (see page 18505)	This is PutField, a member of class TDataPacketWriter.
	PutStringField (see page 18505)	This is PutStringField, a member of class TDataPacketWriter.
	PutVarBytesField (see page 18505)	This is PutVarBytesField, a member of class TDataPacketWriter.
	PutWideStringField (see page 18506)	This is PutWideStringField, a member of class TDataPacketWriter.
	RefreshPutProcs (see page 18506)	This is RefreshPutProcs, a member of class TDataPacketWriter.
	WriteDataSet (see page 18506)	This is WriteDataSet, a member of class TDataPacketWriter.

	WriteMetaData (see page 18507)	This is WriteMetaData, a member of class TDataPacketWriter.

Public Methods

	Name	Description
	GetDataPacket (see page 18503)	This is GetDataPacket, a member of class TDataPacketWriter.
	Reset (see page 18506)	This is Reset, a member of class TDataPacketWriter.

124.1.6.4.1 TDataPacketWriter.AddColumn Method

Pascal

```
procedure AddColumn(const Info: TPutFieldInfo); virtual;
```

C++

```
virtual __fastcall AddColumn(const TPutFieldInfo Info);
```

Description

This is AddColumn, a member of class TDataPacketWriter.

124.1.6.4.2 TDataPacketWriter.AddConstraints Method

Pascal

```
procedure AddConstraints(DataSet: TDataSet); virtual;
```

C++

```
virtual __fastcall AddConstraints(TDataSet DataSet);
```

Description

This is AddConstraints, a member of class TDataPacketWriter.

124.1.6.4.3 TDataPacketWriter.AddDataSetAttributes Method

Pascal

```
procedure AddDataSetAttributes(DataSet: TDataSet); virtual;
```

C++

```
virtual __fastcall AddDataSetAttributes(TDataSet DataSet);
```

Description

This is AddDataSetAttributes, a member of class TDataPacketWriter.

124.1.6.4.4 TDataPacketWriter.AddExtraFieldProps Method

Pascal

```
procedure AddExtraFieldProps(Field: TField);
```

C++

```
__fastcall AddExtraFieldProps(TField Field);
```

Description

This is AddExtraFieldProps, a member of class TDataPacketWriter.

124.1.6.4.5 TDataPacketWriter.AddFieldLinks Method

Pascal

```
procedure AddFieldLinks(const Info: TInfoArray); virtual;
```

C++

```
virtual __fastcall AddFieldLinks(const TInfoArray Info);
```

Description

This is AddFieldLinks, a member of class TDataPacketWriter.

124.1.6.4.6 TDataPacketWriter.AddIndexDefs Method

Pascal

```
procedure AddIndexDefs(DataSet: TDataSet; const Info: TInfoArray); virtual;
```

C++

```
virtual __fastcall AddIndexDefs(TDataSet DataSet, const TInfoArray Info);
```

Description

This is AddIndexDefs, a member of class TDataPacketWriter.

124.1.6.4.7 TDataPacketWriter.FreeInfoRecords Method

Pascal

```
procedure FreeInfoRecords(var Info: TInfoArray);
```

C++

```
__fastcall FreeInfoRecords(TInfoArray Info);
```

Description

This is FreeInfoRecords, a member of class TDataPacketWriter.

124.1.6.4.8 TDataPacketWriter.GetDataPacket Method

Pascal

```
procedure GetDataPacket(DataSet: TDataSet; var RecsOut: Integer; out Data: OleVariant);
virtual;
```

C++

```
virtual __fastcall GetDataPacket(TDataSet DataSet, int RecsOut, OleVariant Data);
```

Description

This is GetDataPacket, a member of class TDataPacketWriter.

124.1.6.4.9 TDataPacketWriter.GetFieldIdx Method

Pascal

```
function GetFieldIdx(const FieldName: WideString; const Info: TInfoArray): Integer;
```

C++

```
__fastcall int GetFieldIdx(const BSTR FieldName, const TInfoArray Info);
```

Description

This is GetFieldIdx, a member of class TDataPacketWriter.

124.1.6.4.10 TDataPacketWriter.InitPutProcs Method

Pascal

```
function InitPutProcs(ADataSet: TDataSet; var GlobalIdx: Integer): TInfoArray;
```

C++

```
__fastcall TInfoArray InitPutProcs(TDataSet ADataSet, int GlobalIdx);
```

Description

This is InitPutProcs, a member of class TDataPacketWriter.

124.1.6.4.11 TDataPacketWriter.PutADTField Method

Pascal

```
procedure PutADTField(Info: PPutFieldInfo);
```

C++

```
__fastcall PutADTField(PPutFieldInfo Info);
```

Description

This is PutADTField, a member of class TDataPacketWriter.

124.1.6.4.12 TDataPacketWriter.PutArrayField Method

Pascal

```
procedure PutArrayField(Info: PPutFieldInfo);
```

C++

```
__fastcall PutArrayField(PPutFieldInfo Info);
```

Description

This is PutArrayField, a member of class TDataPacketWriter.

124.1.6.4.13 TDataPacketWriter.PutBlobField Method

Pascal

```
procedure PutBlobField(Info: PPutFieldInfo);
```

C++

```
__fastcall PutBlobField(PPutFieldInfo Info);
```

Description

This is PutBlobField, a member of class TDataPacketWriter.

124.1.6.4.14 TDataPacketWriter.PutCalcField Method

Pascal

```
procedure PutCalcField(Info: PPutFieldInfo);
```

C++

```
__fastcall PutCalcField(PPutFieldInfo Info);
```

Description

This is PutCalcField, a member of class TDataPacketWriter.

124.1.6.4.15 TDataPacketWriter.PutDataSetField Method

Pascal

```
procedure PutDataSetField(Info: PPutFieldInfo);
```

C++

```
__fastcall PutDataSetField(PPutFieldInfo Info);
```

Description

This is PutDataSetField, a member of class TDataPacketWriter.

124.1.6.4.16 TDataPacketWriter.PutField Method

Pascal

```
procedure PutField(Info: PPutFieldInfo);
```

C++

```
__fastcall PutField(PPutFieldInfo Info);
```

Description

This is PutField, a member of class TDataPacketWriter.

124.1.6.4.17 TDataPacketWriter.PutStringField Method

Pascal

```
procedure PutStringField(Info: PPutFieldInfo);
```

C++

```
__fastcall PutStringField(PPutFieldInfo Info);
```

Description

This is PutStringField, a member of class TDataPacketWriter.

124.1.6.4.18 TDataPacketWriter.PutVarBytesField Method

Pascal

```
procedure PutVarBytesField(Info: PPutFieldInfo);
```

C++

```
__fastcall PutVarBytesField(PPutFieldInfo Info);
```

Description

This is PutVarBytesField, a member of class TDataPacketWriter.

124.1.6.4.19 TDataPacketWriter.PutWideStringField Method

Pascal

```
procedure PutWideStringField(Info: PPutFieldInfo);
```

C++

```
__fastcall PutWideStringField(PPutFieldInfo Info);
```

Description

This is PutWideStringField, a member of class TDataPacketWriter.

124.1.6.4.20 TDataPacketWriter.RefreshPutProcs Method

Pascal

```
procedure RefreshPutProcs(ADataSet: TDataSet; var Info: TInfoArray);
```

C++

```
__fastcall RefreshPutProcs(TDataSet ADataSet, TInfoArray Info);
```

Description

This is RefreshPutProcs, a member of class TDataPacketWriter.

124.1.6.4.21 TDataPacketWriter.Reset Method

Pascal

```
procedure Reset;
```

C++

```
__fastcall Reset();
```

Description

This is Reset, a member of class TDataPacketWriter.

124.1.6.4.22 TDataPacketWriter.WriteDataSet Method

Pascal

```
function WriteDataSet(DataSet: TDataSet; var Info: TInfoArray; RecsOut: Integer): Integer;
virtual;
```

C++

```
virtual __fastcall int WriteDataSet(TDataSet DataSet, TInfoArray Info, int RecsOut);
```

Description

This is WriteDataSet, a member of class TDataPacketWriter.

124.1.6.4.23 TDataPacketWriter.WriteMetaData Method

Pascal

```
procedure WriteMetaData(DataSet: TDataSet; const Info: TInfoArray; IsReference: Boolean =
False); virtual;
```

C++

```
virtual __fastcall WriteMetaData(TDataSet DataSet, const TInfoArray Info, Boolean IsReference
= False);
```

Description

This is WriteMetaData, a member of class TDataPacketWriter.

124.1.6.5 TDataPacketWriter Properties

The properties of the TDataPacketWriter class are listed here.

Public Properties

	Name	Description
	Constraints (see page 18507)	This is Constraints, a member of class TDataPacketWriter.
	Options (see page 18507)	This is Options, a member of class TDataPacketWriter.
	PacketOptions (see page 18507)	This is PacketOptions, a member of class TDataPacketWriter.

124.1.6.5.1 TDataPacketWriter.Constraints Property

Pascal

```
property Constraints: Boolean;
```

C++

```
__property Boolean Constraints;
```

Description

This is Constraints, a member of class TDataPacketWriter.

124.1.6.5.2 TDataPacketWriter.Options Property

Pascal

```
property Options: TProviderOptions;
```

C++

```
__property TProviderOptions Options;
```

Description

This is Options, a member of class TDataPacketWriter.

124.1.6.5.3 TDataPacketWriter.PacketOptions Property

Pascal

```
property PacketOptions: TGetRecordOptions;
```

C++

```
__property TGetRecordOptions PacketOptions;
```

Description

This is PacketOptions, a member of class TDataPacketWriter.

124.1.6.6 TDataPacketWriter Events

The events of the TDataPacketWriter class are listed here.

Public Events

	Name	Description
	OnGetParams (see page 18508)	This is OnGetParams, a member of class TDataPacketWriter.

124.1.6.6.1 TDataPacketWriter.OnGetParams Event

Pascal

```
property OnGetParams: TGetParamsEvent;
```

C++

```
__property TGetParamsEvent OnGetParams;
```

Description

This is OnGetParams, a member of class TDataPacketWriter.

124.1.7 TDataSetProvider Class

TDataSetProvider provides data from a dataset and resolves updates to that dataset or its database server.

Class Hierarchy

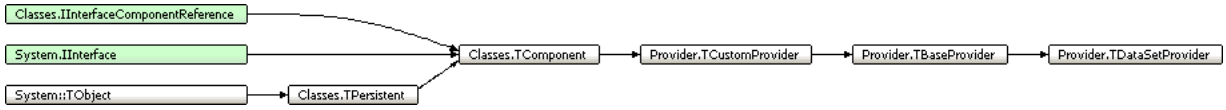

Pascal

```
TDataSetProvider = class(TBaseProvider);
```

C++

```
class TDataSetProvider : public TBaseProvider;
```

File

Provider

Description

Use TDataSetProvider to provide data from a dataset to a client dataset or XML broker and to resolve updates from that client dataset or XML broker back to the dataset or to its underlying database server. TDataSetProvider can be a part of the same application as the client dataset or XML broker, or it can be placed in the application server of a multi-tiered application. It serves as a data broker between a remote database server and a client dataset.

TDataSetProvider packages data from a dataset and passes it in one or more transportable data packets to the client dataset or XML broker. The client dataset reconstructs the data in the data packet to create a local, in-memory copy for user access. When user access is complete, the client dataset repackages any changed data and sends the updates back to the provider. The provider applies the updates to the database or source dataset.

When the provider supplies data to a client dataset, the client dataset reconstructs the data in the data packet to create a local, in-memory copy for user access. When user access is complete, the client dataset repackages any changed data and sends the updates back to the provider. The provider applies the updates to the database or source dataset.

When the provider supplies data to an XML broker, the XML broker adds the data packet, in XML format, to an HTML document that is downloaded by a Web client. When the XML broker receives updates from the Web client, it sends the updates to the provider, which applies them to the database or source dataset.

Client datasets and XML brokers communicate with a provider through the IAppServer interface. In multi-tiered applications, this is the interface of the remote data module that contains the provider. To enable the remote data module to pass calls to the provider, the provider's Exported (see page 18530) property must be true, and its Owner property must specify the remote data module.

See Also

TClientDataSet (see page 8289)

IAppServer (see page 16450)

Determining the Source of Data

Using Provider Components

124.1.7.1 TDataSetProvider Members

The following tables list the members exposed by TDataSetProvider.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckDataSet (see page 18518)	This is CheckDataSet, a member of class TDataSetProvider.
	CheckResolver (see page 18433)	Instantiates a resolver for the Resolver (see page 18441) property if it does not already exist.
V	CreateDataPacket (see page 18518)	Generates a data packet for the data associated with the provider.
V	CreateResolver (see page 18519)	Creates a resolver component for handling the application of updates.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterApplyUpdates (see page 18460)	Generates an AfterApplyUpdates (see page 18475) event.
V	DoAfterExecute (see page 18461)	Generates an AfterExecute (see page 18476) event.
V	DoAfterGetParams (see page 18461)	Generates an AfterGetParams (see page 18477) event.
V	DoAfterGetRecords (see page 18462)	Generates an AfterGetRecords (see page 18477) event.
V	DoAfterRowRequest (see page 18462)	Generates an AfterRowRequest (see page 18478) event.

	DoAfterUpdateRecord (see page 18434)	Generates an AfterUpdateRecord (see page 18442) event.
	DoApplyUpdates (see page 18463)	This is DoApplyUpdates, a member of class TCustomProvider.
	DoBeforeApplyUpdates (see page 18463)	Generates a BeforeApplyUpdates (see page 18479) event.
	DoBeforeCommit (see page 18435)	This is DoBeforeCommit, a member of class TBaseProvider.
	DoBeforeExecute (see page 18519)	Generates a BeforeExecute (see page 18538) event.
	DoBeforeGetParams (see page 18464)	Generates a BeforeGetParams (see page 18480) event.
	DoBeforeGetRecords (see page 18520)	Generates a BeforeGetRecords (see page 18539) event.
	DoBeforeRowRequest (see page 18465)	Generates a BeforeRowRequest (see page 18482) event.
	DoBeforeUpdateRecord (see page 18435)	This is DoBeforeUpdateRecord, a member of class TBaseProvider.
	DoGetProviderAttributes (see page 18521)	Fills a list with information about the dataset that can be used when providing or resolving.
	DoGetRecords (see page 18466)	This is DoGetRecords, a member of class TCustomProvider.
	DoGetTableName (see page 18521)	Generates an OnGetTableName (see page 18542) event.
	DoOnGetData (see page 18435)	Generates an OnGetData (see page 18443) event.
	DoOnUpdateData (see page 18435)	Generates an OnUpdateData (see page 18443) event.
	FetchDetails (see page 18521)	Fills a delta dataset with all nested dataset field values.
	FindRecord (see page 18522)	This is FindRecord, a member of class TDataSetProvider.
	FreeResolver (see page 18436)	Frees the resolver specified by the Resolver (see page 18441) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDataSetFromDelta (see page 18522)	Returns a dataset positioned on the record of the source dataset that is currently being fetched or updated.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InternalApplyUpdates (see page 18522)	Provides the underlying implementation of the ApplyUpdates (see page 18459) method.
	InternalExecute (see page 18523)	Provides the underlying implementation of the Execute method.
	InternalGetParams (see page 18524)	Returns the requested parameter values as a Variant array.
	InternalGetRecords (see page 18524)	Provides the underlying implementation of the GetRecords (see page 18468) method.
	InternalRowRequest (see page 18525)	Provides the underlying implementation of the RowRequest (see page 18472) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 18525)	Positions a data set on the record that corresponds to the current update in a specified delta packet.
	Notification (see page 18526)	Responds to notifications that objects are being created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	Reset (see page 18526)	This is Reset, a member of class TDataSetProvider.
	SetActiveUpdateException (see page 18473)	This is SetActiveUpdateException, a member of class TCustomProvider.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetCommandText (see page 18526)	This is SetCommandText, a member of class TDataSetProvider.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.

	SetParams (see page 18527)	This is SetParams, a member of class TDataSetProvider.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRecord (see page 18527)	Retrieves specified data from a source dataset into a delta packet.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	PacketDataSet (see page 18440)	This is PacketDataSet, a member of class TBaseProvider.
	TransactionStarted (see page 18532)	This is TransactionStarted, a member of class TDataSetProvider.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	ApplyUpdates (see page 18459)	This is the overview for the ApplyUpdates method overload.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 18514)	Creates and initializes a TDataSetProvider (see page 18508) object.
	DataRequest (see page 18460)	Generates an OnDataRequest (see page 18482) event.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 18514)	Frees an instance of a data set provider.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 18466)	Executes a query, stored procedure, or custom SQL statement.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.

	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParams (see page 18467)	Returns the current parameter values.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetRecords (see page 18468)	This is the overview for the GetRecords method overload.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	RowRequest (see page 18472)	Returns information from a specified record.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	ActiveUpdateException (see page 18528)	This is ActiveUpdateException, a member of class TDataSetProvider.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Data (see page 18474)	Denotes a data packet created by the provider.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	OwnerData (see page 18531)	This is OwnerData, a member of class TDataSetProvider.
R	Params (see page 18531)	Stores the parameter values currently in use.
R	Resolver (see page 18441)	Provides access to the resolver component that applies updates and resolves update errors.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterApplyUpdates (see page 18533)	Occurs after the provider applies updates.
	AfterExecute (see page 18534)	Occurs after the provider passes an execute command to its dataset.
	AfterGetParams (see page 18535)	This is AfterGetParams, a member of class TDataSetProvider.
	AfterGetRecords (see page 18535)	Occurs after the provider creates a data packet to send to the client dataset.
	AfterRowRequest (see page 18536)	Occurs after the provider fetches new information about the current record.
	AfterUpdateRecord (see page 18536)	Occurs when the associated resolver has just updated a record.
	BeforeApplyUpdates (see page 18537)	Occurs before the provider applies updates from a client dataset.
	BeforeExecute (see page 18538)	This is BeforeExecute, a member of class TDataSetProvider.
	BeforeGetParams (see page 18538)	Occurs before the provider creates an OleVariant that encodes the current parameter values and sends them to a client dataset.
	BeforeGetRecords (see page 18539)	Occurs before the provider creates a data packet and sends it to a client dataset.
	BeforeRowRequest (see page 18540)	Occurs before the provider creates a delta packet with information about the current record.
	BeforeUpdateRecord (see page 18541)	This is BeforeUpdateRecord, a member of class TDataSetProvider.
	OnDataRequest (see page 18541)	Occurs when a provider calls its DataRequest method.
	OnGetData (see page 18541)	Occurs after a provider retrieves data but before it is sent to the client.
	OnGetDataSetProperties (see page 18542)	Occurs when the provider adds additional information to a data packet it is creating.
	OnGetTableName (see page 18542)	This is OnGetTableName, a member of class TDataSetProvider.
	OnUpdateData (see page 18543)	Occurs when a provider begins to apply updates.
	OnUpdateError (see page 18543)	Occurs when a provider can't update a record because of an error condition.
	OnValidate (see page 18544)	This is OnValidate, a member of class TDataSetProvider.

Published Properties

	Name	Description
	Constraints (see page 18528)	Controls (see page 5711) whether database constraints are passed to the client so that it can check them locally.
	DataSet (see page 18529)	Specifies the dataset from which the provider provides data and to which it applies updates.
	Exported (see page 18530)	Allows client applications to pass calls to the provider when it resides on an application server.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 18530)	Influences what is included in data packets and how the information in data packets is used.
	ResolveToDataSet (see page 18531)	Specifies whether updates should be applied to a dataset or directly to the database server.
	Tag (see page 2808)	Stores an integer value as part of a component.
	UpdateMode (see page 18532)	Controls how the provider locates records when applying updates.

Strict private Fields

	Name	Description
	FActiveUpdateAbortException (see page 18455)	This is FActiveUpdateAbortException, a member of class TCustomProvider.
	FAfterApplyUpdates (see page 18455)	This is FAfterApplyUpdates, a member of class TCustomProvider.
	FAfterExecute (see page 18456)	This is FAfterExecute, a member of class TCustomProvider.
	FAfterGetParams (see page 18456)	This is FAfterGetParams, a member of class TCustomProvider.
	FAfterGetRecords (see page 18456)	This is FAfterGetRecords, a member of class TCustomProvider.
	FAfterRowRequest (see page 18456)	This is FAfterRowRequest, a member of class TCustomProvider.
	FAfterUpdateRecord (see page 18430)	This is FAfterUpdateRecord, a member of class TBaseProvider.
	FBeforeApplyUpdates (see page 18456)	This is FBeforeApplyUpdates, a member of class TCustomProvider.
	FBeforeCommit (see page 18430)	This is FBeforeCommit, a member of class TBaseProvider.
	FBeforeExecute (see page 18457)	This is FBeforeExecute, a member of class TCustomProvider.
	FBeforeGetParams (see page 18457)	This is FBeforeGetParams, a member of class TCustomProvider.
	FBeforeGetRecords (see page 18457)	This is FBeforeGetRecords, a member of class TCustomProvider.
	FBeforeRowRequest (see page 18457)	This is FBeforeRowRequest, a member of class TCustomProvider.

	FBeforeUpdateRecord (see page 18431)	This is FBeforeUpdateRecord, a member of class TBaseProvider.
	FConstraints (see page 18515)	This is FConstraints, a member of class TDataSetProvider.
	FDataDS (see page 18431)	This is FDataDS, a member of class TBaseProvider.
	FDataSet (see page 18515)	This is FDataSet, a member of class TDataSetProvider.
	FDataSetOpened (see page 18516)	This is FDataSetOpened, a member of class TDataSetProvider.
	FDSWriter (see page 18516)	This is FDSWriter, a member of class TDataSetProvider.
	FExported (see page 18458)	This is FExported, a member of class TCustomProvider.
	FGetDSProps (see page 18516)	This is FGetDSProps, a member of class TDataSetProvider.
	FGetTableName (see page 18516)	This is FGetTableName, a member of class TDataSetProvider.
	FOnDataRequest (see page 18458)	This is FOnDataRequest, a member of class TCustomProvider.
	FOnGetData (see page 18431)	This is FOnGetData, a member of class TBaseProvider.
	FOnUpdateData (see page 18431)	This is FOnUpdateData, a member of class TBaseProvider.
	FOnUpdateError (see page 18432)	This is FOnUpdateError, a member of class TBaseProvider.
	FOnValidate (see page 18458)	This is FOnValidate, a member of class TCustomProvider.
	FOwnerData (see page 18458)	This is FOwnerData, a member of class TCustomProvider.
	FParams (see page 18516)	This is FParams, a member of class TDataSetProvider.
	FProviderOptions (see page 18432)	This is FProviderOptions, a member of class TBaseProvider.
	FRecordsSent (see page 18517)	This is FRecordsSent, a member of class TDataSetProvider.
	FResolver (see page 18432)	This is FResolver, a member of class TBaseProvider.
	FResolveToDataSet (see page 18517)	This is FResolveToDataSet, a member of class TDataSetProvider.
	FTransactionStarted (see page 18517)	This is FTransactionStarted, a member of class TDataSetProvider.
	FUpdateMode (see page 18432)	This is FUpdateMode, a member of class TBaseProvider.

Strict private Methods

	Name	Description
	GetData (see page 18467)	This is GetData, a member of class TCustomProvider.
	SetDataSet (see page 18527)	This is SetDataSet, a member of class TDataSetProvider.
	SetResolveToDataSet (see page 18527)	This is SetResolveToDataSet, a member of class TDataSetProvider.

124.1.7.2 TDataSetProvider.Create Constructor

Creates and initializes a TDataSetProvider (see page 18508) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TDataSetProvider(TComponent * AOwner);
```

Description

Call Create to instantiate a provider at runtime. Providers placed in forms or data modules at design time are created automatically.

See Also

Create (see page 18429)

124.1.7.3 TDataSetProvider.Destroy Destructor

Frees an instance of a data set provider.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataSetProvider();
```

Description

Do not call Destroy directly. Instead, call Free, which checks that the reference to the provider is not nil before calling its Destroy method. Destroy frees all helper objects, including the resolver component referenced by the Resolver (see page 18441) property.

See Also

Free (see page 22300)

Resolver (see page 18441)

124.1.7.4 TDataSetProvider Fields

The fields of the TDataSetProvider class are listed here.

Strict private Fields

	Name	Description
	FConstraints (see page 18515)	This is FConstraints, a member of class TDataSetProvider.
	FDataSet (see page 18515)	This is FDataSet, a member of class TDataSetProvider.
	FDataSetOpened (see page 18516)	This is FDataSetOpened, a member of class TDataSetProvider.
	FDSWriter (see page 18516)	This is FDSWriter, a member of class TDataSetProvider.
	FGetDSProps (see page 18516)	This is FGetDSProps, a member of class TDataSetProvider.
	FGetTableName (see page 18516)	This is FGetTableName, a member of class TDataSetProvider.
	FParams (see page 18516)	This is FParams, a member of class TDataSetProvider.
	FRecordsSent (see page 18517)	This is FRecordsSent, a member of class TDataSetProvider.
	FResolveToDataSet (see page 18517)	This is FResolveToDataSet, a member of class TDataSetProvider.
	FTransactionStarted (see page 18517)	This is FTransactionStarted, a member of class TDataSetProvider.

124.1.7.4.1 TDataSetProvider.FConstraints Field

Pascal

```
FConstraints: Boolean;
```

C++

```
Boolean FConstraints;
```

Description

This is FConstraints, a member of class TDataSetProvider.

124.1.7.4.2 TDataSetProvider.FDataSet Field

Pascal

```
FDataSet: TDataSet;
```

C++

```
TDataSet FDataSet;
```

Description

This is FDataSet, a member of class TDataSetProvider.

124.1.7.4.3 TDataSetProvider.FDataSetOpened Field

Pascal

```
FDataSetOpened: Boolean;
```

C++

```
Boolean FDataSetOpened;
```

Description

This is FDataSetOpened, a member of class TDataSetProvider.

124.1.7.4.4 TDataSetProvider.FDSWriter Field

Pascal

```
FDSWriter: TDataPacketWriter;
```

C++

```
TDataPacketWriter FDSWriter;
```

Description

This is FDSWriter, a member of class TDataSetProvider.

124.1.7.4.5 TDataSetProvider.FGetDSProps Field

Pascal

```
FGetDSProps: TGetDSProps;
```

C++

```
TGetDSProps FGetDSProps;
```

Description

This is FGetDSProps, a member of class TDataSetProvider.

124.1.7.4.6 TDataSetProvider.FGetTableName Field

Pascal

```
FGetTableName: TGetTableNameEvent;
```

C++

```
TGetTableNameEvent FGetTableName;
```

Description

This is FGetTableName, a member of class TDataSetProvider.

124.1.7.4.7 TDataSetProvider.FParams Field

Pascal

```
FParams: TParams;
```

C++

```
TParams FParams;
```

Description

This is FParams, a member of class TDataSetProvider.

124.1.7.4.8 TDataSetProvider.FRecordsSent Field

Pascal

```
FRecordsSent: Integer;
```

C++

```
int FRecordsSent;
```

Description

This is FRecordsSent, a member of class TDataSetProvider.

124.1.7.4.9 TDataSetProvider.FResolveToDataSet Field

Pascal

```
FResolveToDataSet: Boolean;
```

C++

```
Boolean FResolveToDataSet;
```

Description

This is FResolveToDataSet, a member of class TDataSetProvider.

124.1.7.4.10 TDataSetProvider.FTransactionStarted Field

Pascal

```
FTransactionStarted: Boolean;
```

C++

```
Boolean FTransactionStarted;
```

Description

This is FTransactionStarted, a member of class TDataSetProvider.

124.1.7.5 TDataSetProvider Methods

The methods of the TDataSetProvider class are listed here.

Protected Methods

	Name	Description
	CheckDataSet (see page 18518)	This is CheckDataSet, a member of class TDataSetProvider.
V	CreateDataPacket (see page 18518)	Generates a data packet for the data associated with the provider.
V	CreateResolver (see page 18519)	Creates a resolver component for handling the application of updates.
V	DoBeforeExecute (see page 18519)	Generates a BeforeExecute (see page 18538) event.
V	DoBeforeGetRecords (see page 18520)	Generates a BeforeGetRecords (see page 18539) event.
V	DoGetProviderAttributes (see page 18521)	Fills a list with information about the dataset that can be used when providing or resolving.
V	DoGetTableName (see page 18521)	Generates an OnGetTableName (see page 18542) event.
V	FetchDetails (see page 18521)	Fills a delta dataset with all nested dataset field values.

	FindRecord (see page 18522)	This is FindRecord, a member of class TDataSetProvider.
	GetDataSetFromDelta (see page 18522)	Returns a dataset positioned on the record of the source dataset that is currently being fetched or updated.
	InternalApplyUpdates (see page 18522)	Provides the underlying implementation of the ApplyUpdates (see page 18459) method.
	InternalExecute (see page 18523)	Provides the underlying implementation of the Execute method.
	InternalGetParams (see page 18524)	Returns the requested parameter values as a Variant array.
	InternalGetRecords (see page 18524)	Provides the underlying implementation of the GetRecords (see page 18468) method.
	InternalRowRequest (see page 18525)	Provides the underlying implementation of the RowRequest (see page 18472) method.
	LocateRecord (see page 18525)	Positions a data set on the record that corresponds to the current update in a specified delta packet.
	Notification (see page 18526)	Responds to notifications that objects are being created or destroyed.
	Reset (see page 18526)	This is Reset, a member of class TDataSetProvider.
	SetCommandText (see page 18526)	This is SetCommandText, a member of class TDataSetProvider.
	SetParams (see page 18527)	This is SetParams, a member of class TDataSetProvider.
	UpdateRecord (see page 18527)	Retrieves specified data from a source dataset into a delta packet.

Strict private Methods

	Name	Description
	SetDataSet (see page 18527)	This is SetDataSet, a member of class TDataSetProvider.
	SetResolveToDataSet (see page 18527)	This is SetResolveToDataSet, a member of class TDataSetProvider.

124.1.7.5.1 TDataSetProvider.CheckDataSet Method

Pascal

```
procedure CheckDataSet;
```

C++

```
__fastcall CheckDataSet();
```

Description

This is CheckDataSet, a member of class TDataSetProvider.

124.1.7.5.2 TDataSetProvider.CreateDataPacket Method

Generates a data packet for the data associated with the provider.

Pascal

```
procedure CreateDataPacket(PacketOpts: TGetRecordOptions; ProvOpts: TProviderOptions; var
RecsOut: Integer; var Data: OleVariant); override;
```

C++

```
virtual __fastcall CreateDataPacket(TGetRecordOptions PacketOpts, TProviderOptions ProvOpts,
int RecsOut, OleVariant Data);
```

Description

CreateDataPacket is used internally to generate the data packets that are returned by GetRecords (see page 18468).

The PacketOpts parameter indicates what information (other than record values) is included in the data packet. It reflects any Options (see page 18530) passed to the GetRecords (see page 18468) method.

The ProvOpts parameter reflects the current value of the Options (see page 18530) property.

The RecsOut parameter specifies the number of records desired in the packet and returns the actual number of records in the generated data packet.

The Data (see page 18474) parameter returns the generated data packet.

See Also

GetRecords (see page 18468)

DataSet (see page 18529)

Options (see page 18440)

124.1.7.5.3 TDataSetProvider.CreateResolver Method

Creates a resolver component for handling the application of updates.

Pascal

```
function CreateResolver: TCustomResolver; override;
```

C++

```
virtual __fastcall TCustomResolver CreateResolver();
```

Description

CreateResolver is called automatically when a resolver is needed to implement one of the provider's methods. CreateResolver returns a newly instantiated resolver object, which is assigned to the Resolver (see page 18441) property. The type of resolver created depends on the ResolveToDataSet (see page 18531) property:

When ResolveToDataSet (see page 18531) is true, CreateResolver returns a TDataSetResolver (see page 18544) component, which applies updates directly to the dataset specified by the DataSet (see page 18529) property.

When ResolveToDataSet (see page 18531) is false, CreateResolver returns a TSQLResolver (see page 18589) component, which applies updates directly to the database server associated with DataSet (see page 18529).

See Also

Resolver (see page 18441)

ResolveToDataSet (see page 18531)

124.1.7.5.4 TDataSetProvider.DoBeforeExecute Method

Generates a BeforeExecute (see page 18538) event.

Pascal

```
procedure DoBeforeExecute(const CommandText: WideString; var Params: OleVariant; var
OwnerData: OleVariant); override;
```

C++

```
virtual __fastcall DoBeforeExecute(const BSTR CommandText, OleVariant Params, OleVariant
OwnerData);
```

Description

The Execute (see page 18466) method calls DoBeforeExecute to generate the BeforeExecute (see page 18538) event before it executes CommandText or executes an associated query or stored procedure.

CommandText is either an optional SQL statement that replaces the SQL of an associated query, or the name of a stored procedure that replaces the associated stored procedure. If Options (see page 18530) includes poAllowCommandText, DoBeforeExecute sends CommandText to the associated dataset so that it can execute the provided statement instead of its query or stored procedure.

Params (see page 18531) contains any parameter values that should be applied to CommandText or to the query or stored

procedure before execution. DoBeforeExecute assigns these parameter values to the dataset (if appropriate).

OwnerData (see page 18531) supplies custom information to the BeforeExecute (see page 18538) event handler. The event handler can change this value. The returned value is then passed to an AfterExecute (see page 18534) event handler and then returned to the caller of the Execute (see page 18466) method.

See Also

Execute (see page 18466)

beforeexecute (see page 18479)

AfterExecute (see page 18476)

DataSet (see page 18529)

124.1.7.5.5 TDataSetProvider.DoBeforeGetRecords Method

Generates a BeforeGetRecords (see page 18539) event.

Pascal

```
procedure DoBeforeGetRecords(Count: Integer; Options: Integer; const CommandText: WideString;
var Params: OleVariant; var OwnerData: OleVariant); override;
```

C++

```
virtual __fastcall DoBeforeGetRecords(int Count, int Options, const BSTR CommandText,
OleVariant Params, OleVariant OwnerData);
```

Description

The GetRecords (see page 18468) method calls DoBeforeGetRecords to generate the BeforeGetRecords (see page 18539) event before it fetches data and assembles it into a data packet.

Count is the number of records requested. TDataSetProvider (see page 18508) does not use this parameter.

Options (see page 18530) indicates whether the data packet should include metadata, whether it should start with the first record, and whether the data packet should be encoded in XML. In TDataSetProvider (see page 18508), DoBeforeGetRecords ignores this parameter.

CommandText specifies either an optional SQL statement that replaces the SQL of an associated query, or the name of a table or stored procedure that replaces the associated table or stored procedure. If Options (see page 18530) includes poAllowCommandText, DoBeforeGetRecords sends CommandText to the associated dataset so that it can generate the set of records.

Params (see page 18531) contains any parameter values that should be applied to CommandText or to a query or stored procedure before fetching data into a data packet. DoBeforeGetRecords assigns these values to the associated dataset, if appropriate.

OwnerData (see page 18531) supplies custom information to the BeforeGetRecords (see page 18539) event handler. The event handler can change this value. The returned value is then passed to an AfterGetRecords (see page 18535) event handler and then (optionally) returned to the caller of the GetRecords (see page 18468) method.

See Also

GetRecords (see page 18468)

BeforeGetRecords (see page 18480)

AfterGetRecords (see page 18477)

DataSet (see page 18529)

124.1.7.5.6 TDataSetProvider.DoGetProviderAttributes Method

Fills a list with information about the dataset that can be used when providing or resolving.

Pascal

```
procedure DoGetProviderAttributes(DataSet: TDataSet; List: TList); virtual;
```

C++

```
virtual __fastcall DoGetProviderAttributes(TDataSet DataSet, TList List);
```

Description

DoGetProviderAttributes is used internally to fetch information about the dataset. This includes both information provided by the dataset (using the IProviderSupport method PSGetAttributes) and custom information supplied by an OnGetDataSetProperties (see page 18542) event handler.

The DataSet (see page 18529) parameter specifies the dataset that stores the information. The List parameter is the list that is filled by DoGetProviderAttributes. DoGetProviderAttributes adds information to the list in Name/Value pairs.

See Also

PSGetAttributes (see page 7579)

OnGetDataSetProperties (see page 18542)

124.1.7.5.7 TDataSetProvider.DoGetTableName Method

Generates an OnGetTableName (see page 18542) event.

Pascal

```
procedure DoGetTableName(DataSet: TDataSet; var TableName: WideString); virtual;
```

C++

```
virtual __fastcall DoGetTableName(TDataSet DataSet, BSTR TableName);
```

Description

The resolver calls DoGetTableName when it initializes the information it needs to apply updates. Override this method to provide an alternate mechanism for determining the name of the dataset that is the target of an update operation.

DataSet (see page 18529) is the dataset that is the target of the update operation.

The name of the target dataset as it should appear in generated SQL statements is returned as the TableName parameter.

See Also

ongettablename (see page 18542)

Resolver (see page 18441)

124.1.7.5.8 TDataSetProvider.FetchDetails Method

Fills a delta dataset with all nested dataset field values.

Pascal

```
procedure FetchDetails(Source: TDataSet; Delta: TDataSet); override;
```

C++

```
virtual __fastcall FetchDetails(TDataSet Source, TDataSet Delta);
```

Description

FetchDetails is used internally to fetch the current values of any nested datasets when the provider is executing its RowRequest (see page 18472) method. Source is a source dataset that has the current values for all fields. Delta is a delta dataset that must be updated so that its nested detail fields match the values in Source.

See Also

RowRequest (see page 18472)

124.1.7.5.9 TDataSetProvider.FindRecord Method

Pascal

```
function FindRecord(Source: TDataSet; Delta: TDataSet; UpdateMode: TUpdateMode): Boolean;
virtual;
```

C++

```
virtual __fastcall Boolean FindRecord(TDataSet Source, TDataSet Delta, TUpdateMode UpdateMode);
```

Description

This is FindRecord, a member of class TDataSetProvider.

124.1.7.5.10 TDataSetProvider.GetDataSetFromDelta Method

Returns a dataset positioned on the record of the source dataset that is currently being fetched or updated.

Pascal

```
function GetDataSetFromDelta(ATree: TUpdateTree; Source: TDataSet; Delta: TDataSet; Mode:
TUpdateMode): TDataSet; virtual;
```

C++

```
virtual __fastcall TDataSet GetDataSetFromDelta(TUpdateTree ATree, TDataSet Source, TDataSet
Delta, TUpdateMode Mode);
```

Description

GetDataSetFromDelta is used internally to fetch a specific record when the source dataset is unidirectional.

ATree is a TUpdateTree (see page 18600) object that represents the update or data fetch operation.

Source is a source dataset. Because it is unidirectional, however, it is not necessarily positioned on the correct record.

Delta is a delta dataset that indicates the desired record.

Mode indicates the criteria to use for locating a record on the server from the dataset that Source uses. The values that follow these criteria come from Delta.

GetDataSetFromDelta returns a unidirectional dataset that corresponds to Source but that contains a single record based on the criteria in Delta and Mode.

See Also

IsUniDirectional (see page 7029)

DataSet (see page 18529)

124.1.7.5.11 TDataSetProvider.InternalApplyUpdates Method

Provides the underlying implementation of the ApplyUpdates (see page 18459) method.

Pascal

```
function InternalApplyUpdates(const Delta: OleVariant; MaxErrors: Integer; out ErrorCount:
Integer): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalApplyUpdates(const OleVariant Delta, int MaxErrors, int
ErrorCount);
```

Description

The ApplyUpdates (see page 18459) method calls InternalApplyUpdates to apply the updates contained in a delta packet after generating the BeforeApplyUpdates (see page 18537) event and before generating the AfterApplyUpdates (see page 18533) event.

If the Options (see page 18530) property includes poReadOnly, InternalApplyUpdates raises an exception. Otherwise, it tells the dataset to start a transaction, calls the ApplyUpdates (see page 18459) method of its Resolver (see page 18441) component to handle the actual update and resolving process, and tells the dataset to complete the transaction.

Delta is the delta packet containing insertions, deletions, and modifications to be applied.

MaxErrors indicates the maximum number of errors that can be applied before stopping the update operation. If MaxErrors is –1, the provider tries to apply all updates.

ErrorCount returns the number of errors encountered during the update operation.

InternalApplyUpdates returns a data packet of updates that could not be applied to the database as an OleVariant.

See Also

Resolver (see page 18441)

ApplyUpdates (see page 18490)

CreateResolver (see page 18519)

Options (see page 18440)

applyupdates (see page 18459)

124.1.7.5.12 TDataSetProvider.InternalExecute Method

Provides the underlying implementation of the Execute method.

Pascal

```
procedure InternalExecute(const CommandText: WideString; var Params: OleVariant); override;
```

C++

```
virtual __fastcall InternalExecute(const BSTR CommandText, OleVariant Params);
```

Description

The Execute method calls InternalExecute to execute the appropriate SQL statement after calling DoBeforeExecute to generate a BeforeExecute event and apply the CommandText and Params. As Implemented in TCustomProvider, InternalExecute does nothing. Descendants override this method to pass the execute command on to an associated dataset.

CommandText is an SQL statement that replaces the SQL of an associated query, or the name of a stored procedure that replaces the associated stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

On entry, Params supplies parameter values for the query, stored procedure, or CommandText string. On exit, Params returns any output parameters.

See Also

Execute (see page 18466)

DoBeforeExecute (see page 18464)

124.1.7.5.13 TDataSetProvider.InternalGetParams Method

Returns the requested parameter values as a Variant array.

Pascal

```
function InternalGetParams(Types: TParamTypes = AllParamTypes): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalGetParams(TParamTypes Types = AllParamTypes);
```

Description

The provider uses InternalGetParams internally whenever it needs to assemble the current parameter values into an OleVariant. This provides the underlying implementation of the GetParams (see page 18467) method, as well as assembling parameter values for methods that return output parameters (such as Execute (see page 18466) or GetRecords (see page 18468)).

The Types (see page 23293) parameter limits the parameter values that are returned to a subset of all possible parameter types. If this parameter is omitted, all parameter values are returned.

See Also

GetParams (see page 18467)

Execute (see page 18466)

GetRecords (see page 18468)

DataSet (see page 18529)

124.1.7.5.14 TDataSetProvider.InternalGetRecords Method

Provides the underlying implementation of the GetRecords (see page 18468) method.

Pascal

```
function InternalGetRecords(Count: Integer; out RecsOut: Integer; Options: TGetRecordOptions;
const CommandText: WideString; var Params: OleVariant): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalGetRecords(int Count, int RecsOut, TGetRecordOptions
Options, const BSTR CommandText, OleVariant Params);
```

Description

The GetRecords (see page 18468) method calls InternalGetRecords to assemble a data packet after calling DoBeforeGetRecords (see page 18520) to apply the CommandText and Params (see page 18531) and generate a BeforeGetRecords (see page 18539) event. InternalGetRecords calls CreateDataPacket (see page 18518) to create a data packet that reflects the current property settings of the provider. It then generates an OnGetData (see page 18541) event.

Count indicates the number of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only the specified number of records are retrieved.

RecsOut returns the actual number of records retrieved.

Options (see page 18530) indicates what information should be added to the data packet in addition to data. It can include

grMetaData (include metadata), grReset (start providing records with the first one in the dataset), and grXML (encode the datapacket in XML rather than as an OleVariant) or grXMLUTF8 (same as grXML but use UTF8 to encode extended characters).

CommandText specifies an optional SQL statement that replaces the SQL of an associated query, or the name of a table or stored procedure that replaces the associated table or stored procedure. This parameter is ignored if the Options (see page 18530) property does not include poAllowCommandText.

Params (see page 18531) specifies any parameter values that should be applied to CommandText or to an associated query or stored procedure. If the provider is associated with a TTable, Params (see page 18531) specifies field values that filter the result set. On exit from GetRecords (see page 18468), Params (see page 18531) returns any output parameters.

Records are returned as a data packet in a Variant.

Note: If the DataSet (see page 18529) property is not set, InternalGetRecords raises an exception.

See Also

Data (see page 18474)

CreateDataPacket (see page 18518)

OnGetData (see page 18443)

BeforeGetRecords (see page 18480)

AfterGetRecords (see page 18477)

DoBeforeGetRecords (see page 18520)

124.1.7.5.15 TDataSetProvider.InternalRowRequest Method

Provides the underlying implementation of the RowRequest (see page 18472) method.

Pascal

```
function InternalRowRequest(const Row: OleVariant; Options: TFetchOptions): OleVariant;
override;
```

C++

```
virtual __fastcall OleVariant InternalRowRequest(const OleVariant Row, TFetchOptions Options);
```

Description

The RowRequest (see page 18472) method calls InternalRowRequest to assemble the specified information after generating a BeforeRowRequest (see page 18540) event and before generating an AfterRowRequest (see page 18536) event. InternalRowRequest passes this request on to its associated Resolver (see page 18441).

The Row parameter is an OleVariant that describes the record about which information is requested.

InternalRowRequest returns the requested information in a delta packet.

See Also

RowRequest (see page 18472)

Resolver (see page 18441)

RowRequest (see page 18496)

124.1.7.5.16 TDataSetProvider.LocateRecord Method

Positions a data set on the record that corresponds to the current update in a specified delta packet.

Pascal

```
procedure LocateRecord(Source: TDataSet; Delta: TDataSet); override;
```

C++

```
virtual __fastcall LocateRecord(TDataSet Source, TDataSet Delta);
```

Description

LocateRecord is called indirectly by the RowRequest (see page 18472) method to locate a record before fetching information about it. The Source parameter is the dataset containing current record values. The Delta parameter is a delta packet that identifies the desired type of information and information identifying the desired record.

LocateRecord uses the criteria specified by UpdateMode (see page 18532) to identify the desired record.

See Also

RowRequest (see page 18472)

UpdateMode (see page 18441)

124.1.7.5.17 TDataSetProvider.Notification Method

Responds to notifications that objects are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification overrides the inherited method to add a check that the dataset specified by the DataSet (see page 18529) property is not being deleted. If DataSet (see page 18529) is about to be destroyed, Notification sets the DataSet (see page 18529) property to nil (Delphi) or NULL (C++).

See Also

OnUpdateError (see page 18444)

Create (see page 6733)

124.1.7.5.18 TDataSetProvider.Reset Method

Pascal

```
procedure Reset;
```

C++

```
__fastcall Reset();
```

Description

This is Reset, a member of class TDataSetProvider.

124.1.7.5.19 TDataSetProvider.SetCommandText Method

Pascal

```
procedure SetCommandText(const CommandText: WideString); virtual;
```

C++

```
virtual __fastcall SetCommandText(const BSTR CommandText);
```

Description

This is SetCommandText, a member of class TDataSetProvider.

124.1.7.5.20 TDataSetProvider.SetDataSet Method

Pascal

```
procedure SetDataSet(ADataSet: TDataSet);
```

C++

```
__fastcall SetDataSet(TDataSet ADataSet);
```

Description

This is SetDataSet, a member of class TDataSetProvider.

124.1.7.5.21 TDataSetProvider.SetParams Method

Pascal

```
procedure SetParams(Values: OleVariant); virtual;
```

C++

```
virtual __fastcall SetParams(OleVariant Values);
```

Description

This is SetParams, a member of class TDataSetProvider.

124.1.7.5.22 TDataSetProvider.SetResolveToDataSet Method

Pascal

```
procedure SetResolveToDataSet(Value: Boolean);
```

C++

```
__fastcall SetResolveToDataSet(Boolean Value);
```

Description

This is SetResolveToDataSet, a member of class TDataSetProvider.

124.1.7.5.23 TDataSetProvider.UpdateRecord Method

Retrieves specified data from a source dataset into a delta packet.

Pascal

```
procedure UpdateRecord(Source: TDataSet; Delta: TDataSet; BlobsOnly: Boolean; KeyOnly:
Boolean); override;
```

C++

```
virtual __fastcall UpdateRecord(TDataSet Source, TDataSet Delta, Boolean BlobsOnly, Boolean
KeyOnly);
```

Description

UpdateRecord is called indirectly by the RowRequest (see page 18472) method to fetch information about a specified record. The Source parameter is the dataset containing indicated record. The Delta parameter is a delta packet that returns the requested data. BlobsOnly indicates whether all field values should be provided (BlobsOnly == false) or only Blob field values (BlobsOnly == true).

See Also

RowRequest (see page 18472)

124.1.7.6 TDataSetProvider Properties

The properties of the TDataSetProvider class are listed here.

Protected Properties

	Name	Description
	TransactionStarted (see page 18532)	This is TransactionStarted, a member of class TDataSetProvider.

Public Properties

	Name	Description
	ActiveUpdateException (see page 18528)	This is ActiveUpdateException, a member of class TDataSetProvider.
	OwnerData (see page 18531)	This is OwnerData, a member of class TDataSetProvider.
R	Params (see page 18531)	Stores the parameter values currently in use.

Published Properties

	Name	Description
	Constraints (see page 18528)	Controls (see page 5711) whether database constraints are passed to the client so that it can check them locally.
	DataSet (see page 18529)	Specifies the dataset from which the provider provides data and to which it applies updates.
	Exported (see page 18530)	Allows client applications to pass calls to the provider when it resides on an application server.
	Options (see page 18530)	Influences what is included in data packets and how the information in data packets is used.
	ResolveToDataSet (see page 18531)	Specifies whether updates should be applied to a dataset or directly to the database server.
	UpdateMode (see page 18532)	Controls how the provider locates records when applying updates.

124.1.7.6.1 TDataSetProvider.ActiveUpdateException Property

Pascal

```
property ActiveUpdateException: Exception;
```

C++

```
__property Exception ActiveUpdateException;
```

Description

This is ActiveUpdateException, a member of class TDataSetProvider.

124.1.7.6.2 TDataSetProvider.Constraints Property

Controls (see page 5711) whether database constraints are passed to the client so that it can check them locally.

Pascal

```
property Constraints: Boolean;
```

C++

```
__property Boolean Constraints;
```

Description

Set Constraints to specify whether data constraints defined for the provider's associated dataset are sent along with the metadata for the dataset. Sending constraints allows client datasets to check for constraint violations locally, so that updates are less likely to fail when sent back to the database server.

When Constraints is true, constraints are sent. To prevent the provider from sending constraint information, set Constraints to false.

Note: Before constraints can be sent to client applications, they must be imported to the data dictionary on the application server. Use SQL explorer to import constraints to the data dictionary.

See Also

Constraints (see page 7013)

DataSet (see page 18529)

124.1.7.6.3 TDataSetProvider.DataSet Property

Specifies the dataset from which the provider provides data and to which it applies updates.

Pascal

```
property DataSet: TDataSet;
```

C++

```
__property TDataSet DataSet;
```

Description

Set DataSet to link the dataset provider to a dataset that it represents. The dataset provider packages the records from (and other information about) DataSet into data packets that can be used by client datasets and XML brokers. DataSet must be defined if the provider is used to create data packets.

If the ResolveToDataSet (see page 18531) property is true, the provider applies updates from the application server directly to this dataset. Otherwise, the provider applies updates directly to the underlying database server.

Note: The provider communicates with DataSet using the IProviderSupport interface. This interface is available in all datasets, although in TDataSet most of the IProviderSupport methods are simple stubs that either do nothing or generate an exception. The dataset classes that ship with the IDE (dbExpress datasets, BDE-enabled datasets, ADO-enabled datasets, InterBase Express datasets,

and client datasets) override these protected methods to implement the IProviderSupport interface in a more useful fashion. If you are using custom dataset components, check with the vendor whether they can work with a dataset provider.

See Also

applyupdates (see page 18459)

Data (see page 18474)

IProviderSupport (see page 7576)

Connecting a Client Dataset to Another Dataset in the Same Application

Determining the Source of Data

124.1.7.6.4 TDataSetProvider.Exported Property

Allows client applications to pass calls to the provider when it resides on an application server.

Pascal

```
property Exported: Boolean;
```

C++

```
__property Boolean Exported;
```

Description

Use Exported to indicate whether the provider is made available to clients of a remote data module that owns it. When Exported is true, client applications can specify the provider as a target for calls made using the IAppServer interface. When Exported is false, IAppServer calls that specify the provider cause an exception.

Note: In addition to the Exported property, the provider must be registered with the remote data module for clients to access it using the data module's IAppServer interface. When the provider's Owner is the remote data module, it is automatically registered. If the provider has a different owner, the server application must explicitly register providers using the remote data module's RegisterProvider method.

See Also

IAppServer (see page 16450)

Providers (see page 19908)

registerprovider (see page 19902)

124.1.7.6.5 TDataSetProvider.Options Property

Influences what is included in data packets and how the information in data packets is used.

Pascal

```
property Options: TProviderOptions;
```

C++

```
__property TProviderOptions Options;
```

Description

Set Options to customize how the provider communicates with client datasets. Options is a set that specifies

Whether nested detail sets and BLOB data are included in data packets or fetched separately.

Whether data packets include field component properties such as formats, display names, and maximum & minimum values.

Whether data packets are provided on a read-only basis, and whether specific types of changes (insertions, edits, or deletions) are disabled.

Whether changes to master table fields should cause corresponding updates to detail tables.

Whether a single update can affect multiple server records.

Whether the client's records are refreshed when it applies updates.

Whether the client can send SQL statements that override the provider's dataset.

Note: Not all options are supported by an InternetExpress client. Specifically, InternetExpress clients don't support poFetchBlobsOnDemand, poFetchDetailsOnDemand, poDisableInserts, poDisableEdits, psDisableDeletes, poPropogateChanges, and poAllowCommandText.

See Also

Data (see page 18474)

applyupdates (see page 18459)

Overriding the Dataset On the Application Server

Setting Options That Influence the Data Packets

124.1.7.6.6 TDataSetProvider.OwnerData Property

Pascal

```
property OwnerData: OleVariant;
```

C++

```
__property OleVariant OwnerData;
```

Description

This is OwnerData, a member of class TDataSetProvider.

124.1.7.6.7 TDataSetProvider.Params Property

Stores the parameter values currently in use.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

This protected property acts as a temporary cache for parameter values from clients. It is set to the parameter values supplied by calls to the Execute (see page 18466) or GetRecords (see page 18468) method.

Note: If the dataset provider is responding to multiple client datasets or XML brokers

, the value of Params can only be used within the context of a single call to SetParams (see page 18527), Execute (see page 18466), or GetRecords (see page 18468). After that, other client calls can change the value of parameters.

See Also

Execute (see page 18466)

GetRecords (see page 18468)

124.1.7.6.8 TDataSetProvider.ResolveToDataSet Property

Specifies whether updates should be applied to a dataset or directly to the database server.

Pascal

```
property ResolveToDataSet: Boolean;
```

C++

```
__property Boolean ResolveToDataSet;
```

Description

Set ResolveToDataSet to specify how updates are applied. When ResolveToDataSet is true, the Resolver (see page 18441) property is set to a TDataSetResolver (see page 18544) component, which applies updates directly to the dataset specified by the DataSet (see page 18529) property. This can be useful if the application uses the events on the dataset component or if the dataset does not represent the data from a database server (for example, a client dataset).

When ResolveToDataSet is false, the Resolver (see page 18441) property is set to a TSQLResolver (see page 18589) component, which applies updates directly to the database server associated with DataSet (see page 18529). This can be more efficient, because it skips the intermediate step of using the dataset. It is the only way to resolve updates if the source dataset is read-only (for example, a unidirectional dataset).

See Also

DataSet (see page 18529)

Resolver (see page 18441)

Choosing How to Apply Updates Using a Datasetm Provider

124.1.7.6.9 TDataSetProvider.TransactionStarted Property

Pascal

```
property TransactionStarted: Boolean;
```

C++

```
__property Boolean TransactionStarted;
```

Description

This is TransactionStarted, a member of class TDataSetProvider.

124.1.7.6.10 TDataSetProvider.UpdateMode Property

Controls how the provider locates records when applying updates.

Pascal

```
property UpdateMode: TUpdateMode;
```

C++

```
__property TUpdateMode UpdateMode;
```

Description

Use UpdateMode to specify the criteria for locating a record in the dataset when applying updates. UpdateMode specifies whether modified records are located based on all columns (fields), on only the key fields, or on the key fields plus the original values of fields that have been modified.

Note: To control how the provider locates records by specifying individual fields, use the field components' ProviderFlags property.

See Also

applyupdates (see page 18459)

ProviderFlags (see page 7176)

Influencing How Updates Are Applied

124.1.7.7 TDataSetProvider Events

The events of the TDataSetProvider class are listed here.

Published Events

	Name	Description
	AfterApplyUpdates (see page 18533)	Occurs after the provider applies updates.
	AfterExecute (see page 18534)	Occurs after the provider passes an execute command to its dataset.
	AfterGetParams (see page 18535)	This is AfterGetParams, a member of class TDataSetProvider.
	AfterGetRecords (see page 18535)	Occurs after the provider creates a data packet to send to the client dataset.
	AfterRowRequest (see page 18536)	Occurs after the provider fetches new information about the current record.
	AfterUpdateRecord (see page 18536)	Occurs when the associated resolver has just updated a record.
	BeforeApplyUpdates (see page 18537)	Occurs before the provider applies updates from a client dataset.
	BeforeExecute (see page 18538)	This is BeforeExecute, a member of class TDataSetProvider.
	BeforeGetParams (see page 18538)	Occurs before the provider creates an OleVariant that encodes the current parameter values and sends them to a client dataset.
	BeforeGetRecords (see page 18539)	Occurs before the provider creates a data packet and sends it to a client dataset.
	BeforeRowRequest (see page 18540)	Occurs before the provider creates a delta packet with information about the current record.
	BeforeUpdateRecord (see page 18541)	This is BeforeUpdateRecord, a member of class TDataSetProvider.
	OnDataRequest (see page 18541)	Occurs when a provider calls its DataRequest method.
	OnGetData (see page 18541)	Occurs after a provider retrieves data but before it is sent to the client.
	OnGetDataSetProperties (see page 18542)	Occurs when the provider adds additional information to a data packet it is creating.
	OnGetTableName (see page 18542)	This is OnGetTableName, a member of class TDataSetProvider.
	OnUpdateData (see page 18543)	Occurs when a provider begins to apply updates.
	OnUpdateError (see page 18543)	Occurs when a provider can't update a record because of an error condition.
	OnValidate (see page 18544)	This is OnValidate, a member of class TDataSetProvider.

124.1.7.7.1 TDataSetProvider.AfterApplyUpdates Event

Occurs after the provider applies updates.

Pascal

```
property AfterApplyUpdates: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterApplyUpdates;
```

Description

Write an AfterApplyUpdates event handler to return custom information to a client dataset after applying updates. AfterApplyUpdates is part of the mechanism by which a client dataset and a provider communicate information about updates. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_ApplyUpdates method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeApplyUpdates event, where it can respond to or change custom information that the client dataset sends.

2.If it is a dataset provider, the provider receives an OnUpdateData event where it can edit the updates before they are applied.

3. The provider applies the updates, and attempts to resolve problems.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider in its AfterApplyUpdates event handler.

Note: If ApplyUpdates is called without the OwnerData parameter, on entry to the event handler the OwnerData parameter contains the value set in the BeforeApplyUpdates event handler and it is ignored on exit from the AfterApplyUpdates event handler.

See Also

applyupdates (see page 18459)

AS_ApplyUpdates (see page 16451)

ApplyUpdates (see page 8394)

AfterApplyUpdates (see page 8497)

BeforeApplyUpdates (see page 18479)

OnUpdateData (see page 18443)

Responding to Client Update Requests

Communicating with the Client Dataset

124.1.7.7.2 TDataSetProvider.AfterExecute Event

Occurs after the provider passes an execute command to its dataset.

Pascal

```
property AfterExecute: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterExecute;
```

Description

Write an AfterExecute event handler to return custom information to a client dataset after executing a query or stored procedure. AfterExecute is part of the mechanism by which a client dataset and a provider communicate information when executing queries or stored procedures that do not return a cursor. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_Execute method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeExecute event, where it can respond to or change custom information that the client dataset sends.

2.The provider dispatches its Execute method to execute a query or stored procedure.

3.The provider receives an AfterExecute event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeExecute event handler.

4.The client dataset receives an AfterExecute event, where it can respond to the custom information returned by the provider in its AfterExecute event handler.

See Also

Execute (see page 18466)

AS_Execute (see page 16453)

Execute (see page 8413)

beforeexecute (see page 18479)

DataSet (see page 18529)

AfterExecute (see page 8504)

Communicating with the Client Dataset

124.1.7.7.3 TDataSetProvider.AfterGetParams Event

Pascal

```
property AfterGetParams: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterGetParams;
```

Description

This is AfterGetParams, a member of class TDataSetProvider.

124.1.7.7.4 TDataSetProvider.AfterGetRecords Event

Occurs after the provider creates a data packet to send to the client dataset.

Pascal

```
property AfterGetRecords: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterGetRecords;
```

Description

Write an AfterGetRecords event handler to send custom information to a client dataset when providing a data packet. AfterGetRecords is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_GetRecords method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeGetRecords event, where it can respond to or change custom information that the client dataset sends.

2.If it is a dataset provider, the provider receives an OnGetData event where it can edit the data before creating a data packet, and then generates the data packet.

3.The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.

4.The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider in its AfterGetRecords event handler.

See Also

AS_GetRecords (see page 16455)

OnGetData (see page 18443)

GetRecords (see page 18468)

AfterGetRecords (see page 8506)

BeforeGetRecords (see page 18480)

Communicating with the Client Dataset

Responding to Client Data Requests

124.1.7.7.5 TDataSetProvider.AfterRowRequest Event

Occurs after the provider fetches new information about the current record.

Pascal

```
property AfterRowRequest: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterRowRequest;
```

Description

Write an AfterRowRequest event handler to send custom information to a client dataset along with information about the current record. AfterRowRequest is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_RowRequest method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeRowRequest event, where it can respond to or change custom information that the client dataset sends.

2.The provider creates a delta packet that contains the requested information.

3.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeRowRequest event handler.

4.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider in its AfterRowRequest event handler.

See Also

AS_RowRequest (see page 16456)

RowRequest (see page 18472)

AfterRowRequest (see page 8509)

BeforeRowRequest (see page 18482)

FetchBlobs (see page 8414)

FetchParams (see page 8415)

RefreshRecord (see page 8442)

Communicating with the Client Dataset

124.1.7.7.6 TDataSetProvider.AfterUpdateRecord Event

Occurs when the associated resolver has just updated a record.

Pascal

```
property AfterUpdateRecord: TAfterUpdateRecordEvent;
```

C++

```
__property TAfterUpdateRecordEvent AfterUpdateRecord;
```

Description

Write an AfterUpdateRecord event handler to provide special processing immediately after a record has been successfully updated. For example, use AfterUpdateRecord to maintain a log of successful updates.

See Also

beforeupdaterecord (see page 18442)

AfterApplyUpdates (see page 18475)

124.1.7.7.7 TDataSetProvider.BeforeApplyUpdates Event

Occurs before the provider applies updates from a client dataset.

Pascal

```
property BeforeApplyUpdates: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeApplyUpdates;
```

Description

Write a BeforeApplyUpdates event handler to respond to custom information from a client dataset. BeforeApplyUpdates is part of the mechanism by which a provider and a client dataset communicate information about updates. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset's ApplyUpdates method is called, the following events occur:

1.The client dataset receives a BeforeApplyUpdates event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeApplyUpdates event, where the OleVariant from the client dataset appears as the OwnerData parameter. The provider can respond to or change that information before applying the updates.

3.If it is a dataset provider, the provider applies the updates, and receives an OnUpdateError event for any problem records.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider in its AfterApplyUpdates event handler.

Note: If ApplyUpdates is called without the OwnerData parameter, the OwnerData parameter is nil (Delphi) or NULL (C++) on entry to the event handler and its value (on exit) is only passed to the AfterApplyUpdates event handler but not returned to the caller.

See Also

applyupdates (see page 18459)

AS_ApplyUpdates (see page 16451)

ApplyUpdates (see page 8394)

BeforeApplyUpdates (see page 8511)

AfterApplyUpdates (see page 18475)

Responding to Client Update Requests

Communicating with the Client Dataset

124.1.7.7.8 TDataSetProvider.BeforeExecute Event

Pascal

```
property BeforeExecute: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeExecute;
```

Description

This is BeforeExecute, a member of class TDataSetProvider.

124.1.7.7.9 TDataSetProvider.BeforeGetParams Event

Occurs before the provider creates an OleVariant that encodes the current parameter values and sends them to a client dataset.

Pascal

```
property BeforeGetParams: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeGetParams;
```

Description

Write a BeforeGetParams event handler to respond to custom information from a client dataset. BeforeGetParams is part of the mechanism by which a provider and a client dataset communicate information during parameter fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset fetches parameter values from the provider, the following events occur:

1.The client dataset receives a BeforeGetParams event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeGetParams event, where the OleVariant from the client dataset appears as the OwnerData parameter. The provider can respond to or change that information before it fetches and encodes the parameter values.

3.The provider generates an OleVariant for the current parameter values of its dataset, if appropriate.

4.The provider receives an AfterGetParams event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetParams event handler.

5.The client dataset receives an AfterGetParams event, where it can respond to the custom information returned by the provider in its AfterGetParams event handler.

See Also

AS_GetParams (see page 16454)

GetParams (see page 18467)

BeforeGetParams (see page 8514)

aftergetparams (see page 18477)

Communicating with the Client Dataset

124.1.7.7.10 TDataSetProvider.BeforeGetRecords Event

Occurs before the provider creates a data packet and sends it to a client dataset.

Pascal

```
property BeforeGetRecords: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeGetRecords;
```

Description

Write a BeforeGetRecords event handler to respond to custom information from a client dataset. BeforeGetRecords is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset fetches data from the provider, the following events occur:

1.The client dataset receives a BeforeGetRecords event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeGetRecords event, where the OleVariant from the client dataset appears as the OwnerData parameter. The provider can respond to or change that information before it creates a data packet.

3.The provider generates the data packet.

4.The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.

5.The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider in its AfterGetRecords event handler.

Delphi Examples:

```
{
This example shows the implementation of a provider BeforeGetRecords
event handler and a RemoteDataModule UpdateRegistry method.  The
complete project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TPooledRDM.DataSetProvider1BeforeGetRecords(Sender: TObject;
  var OwnerData: OleVariant);
begin
  try
    Query1.Close;
    Query1.SQL.Text := OwnerData[0];
    if not VarIsNull(OwnerData[1]) and not VarIsEmpty(OwnerData[1]) then
    begin
      Query1.Open;
      if not Query1.Locate(Query1.Fields[0].FieldName, OwnerData[1], []) then
        raise Exception.Create('Record not found');
      Query1.Next;
    end;
  finally
    OwnerData := NULL;
  end;
end;
```

See Also

AS_GetRecords (see page 16455)

GetRecords (see page 18468)

BeforeGetRecords (see page 8515)

AfterGetRecords (see page 18477)

Communicating with the Client Dataset

Responding to Client Data Requests

124.1.7.7.11 TDataSetProvider.BeforeRowRequest Event

Occurs before the provider creates a delta packet with information about the current record.

Pascal

```
property BeforeRowRequest: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeRowRequest;
```

Description

Write a BeforeRowRequest event handler to respond to custom information from a client dataset. BeforeRowRequest is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset fetches detail information about the current record or refreshes the current record, the following events occur:

1.The client dataset receives a BeforeRowRequest event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeRowRequest event, where the custom information appears as the OwnerData parameter. The provider can respond to or change that information before it looks up the requested information.

3.The provider creates a delta packet that contains the requested information.

4.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeRowRequest event handler.

5.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider in its AfterRowRequest event handler.

See Also

AS_RowRequest (see page 16456)

RowRequest (see page 18472)

AfterRowRequest (see page 18478)

BeforeRowRequest (see page 8520)

FetchBlobs (see page 8414)

FetchParams (see page 8415)

RefreshRecord (see page 8442)

Communicating with the Client Dataset

124.1.7.7.12 TDataSetProvider.BeforeUpdateRecord Event

Pascal

```
property BeforeUpdateRecord: TBeforeUpdateRecordEvent;
```

C++

```
__property TBeforeUpdateRecordEvent BeforeUpdateRecord;
```

Description

This is BeforeUpdateRecord, a member of class TDataSetProvider.

124.1.7.7.13 TDataSetProvider.OnDataRequest Event

Occurs when a provider calls its DataRequest method.

Pascal

```
property OnDataRequest: TDataRequestEvent;
```

C++

```
__property TDataRequestEvent OnDataRequest;
```

Description

Write an OnDataRequest event handler to do custom processing of data. OnDataRequest has no built-in use. Instead, it allows developers to augment the IAppServer interface so that clients in a multi-tiered application can communicate directly with an individual provider.

If an application does not provide an OnDataRequest event handler, calls to DataRequest return NULL.

Note: To provide the same sort of custom processing on the client side, use an automation method on the connection component.

See Also

DataRequest (see page 18460)

IAppServer (see page 16450)

Communicating with Providers Using Custom Events

Communicating with the Client Dataset

Responding to Client-generated Events

124.1.7.7.14 TDataSetProvider.OnGetData Event

Occurs after a provider retrieves data but before it is sent to the client.

Pascal

```
property OnGetData: TProviderDataEvent;
```

C++

```
__property TProviderDataEvent OnGetData;
```

Description

Write an OnGetData event handler to perform any custom data processing the provider should handle before returning data to a

client. The OnGetData event occurs after a provider fetches data from its dataset but before the data is sent on to the client. Using the methods of the DataSet parameter, data can be edited in the OnGetData event handler before sending it to the client dataset or XML broker.

Note: OnGetData occurs when the Data property is read, as well as when the GetRecords method is called.

See Also

GetRecords (see page 18468)

OnUpdateData (see page 18443)

BeforeGetRecords (see page 18480)

AfterGetRecords (see page 18477)

Responding to Client Data Requests

124.1.7.7.15 TDataSetProvider.OnGetDataSetProperties Event

Occurs when the provider adds additional information to a data packet it is creating.

Pascal

```
property OnGetDataSetProperties: TGetDSProps;
```

C++

```
__property TGetDSProps OnGetDataSetProperties;
```

Description

Write an OnGetDataSetProperties event handler to include application-defined information in the data packets. This information can later be returned using the GetOptionalParam method of the client dataset that represents the data in the packet.

Note: The following names are reserved and can't be used as the name for storing custom information:

UNIQUE_KEY	DEFAULT_ORDER	CHANGE_LOG	SERVER_COL
CONSTRAINTS	DATASET_CONTEXT	DATASET_DELTA	LCID
BDERECORD_X	TABLE_NAME	MD_FIELDLINKS	UPDATEMODE

See Also

GetOptionalParam (see page 8425)

SetOptionalParam (see page 8450)

Adding Custom Information to Data Packets

124.1.7.7.16 TDataSetProvider.OnGetTableName Event

Pascal

```
property OnGetTableName: TGetTableNameEvent;
```

C++

```
__property TGetTableNameEvent OnGetTableName;
```

Description

This is OnGetTableName, a member of class TDataSetProvider.

124.1.7.7.17 TDataSetProvider.OnUpdateData Event

Occurs when a provider begins to apply updates.

Pascal

```
property OnUpdateData: TProviderDataEvent;
```

C++

```
__property TProviderDataEvent OnUpdateData;
```

Description

Write an OnUpdateData event to handle special processing before the provider begins applying updates received from the client dataset or XML broker. Using the methods belonging to the DataSet parameter, an application can

Examine data (for example, for values or data changes that should not permitted), and raise exceptions that cancel applying of updates before they occur.

Change data (for example encrypting or decrypting values) before it is sent on to the source dataset or database server.

See Also

applyupdates (see page 18459)

beforeupdaterecord (see page 18442)

Adding Custom Information to Data Packets

Editing Delta Packets Before Updating the Database

Influencing How Updates Are Applied

124.1.7.7.18 TDataSetProvider.OnUpdateError Event

Occurs when a provider can't update a record because of an error condition.

Pascal

```
property OnUpdateError: TResolverErrorEvent;
```

C++

```
__property TResolverErrorEvent OnUpdateError;
```

Description

Write an OnUpdateError event handler to respond to error conditions that arise when the provider applies a client dataset's updates. The OnUpdateError event handler is called for each record that can't be updated.

Warning: The provider can't detect when an update conflicts with another user's changes to a memo field. This situation will not generate an OnUpdateError event.

See Also

applyupdates (see page 18459)

OnReconcileError (see page 8524)

NewValue (see page 7174)

OldValue (see page 7175)

CurValue (see page 7155)

Resolving Update Errors On the Provider

124.1.7.7.19 TDataSetProvider.OnValidate Event

Pascal

```
property OnValidate: TOnValidate;
```

C++

```
__property TOnValidate OnValidate;
```

Description

This is OnValidate, a member of class TDataSetProvider.

124.1.8 TDataSetResolver Class

TDataSetResolver applies updates to a dataset on behalf of a provider.

Class Hierarchy

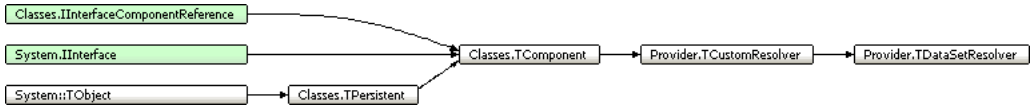

Pascal

```
TDataSetResolver = class(TCustomResolver);
```

C++

```
class TDataSetResolver : public TCustomResolver;
```

File

Provider

Description

TDataSetProvider (see page 18508) components use TDataSetResolver to apply updates to a dataset when their ResolveToDataSet property is true. Otherwise, they use TSQLResolver (see page 18589).

Do not create instances of TDataSetResolver. Provider (see page 18553) components create their own TDataSetResolver objects for applying updates.

See Also

TDataSetProvider (see page 18508)

TBaseProvider (see page 18423)

TSQLResolver (see page 18589)

TDataSetProvider (see page 18508)

124.1.8.1 TDataSetResolver Members

The following tables list the members exposed by TDataSetResolver.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	BeginUpdate (see page 18549)	Provides the interface for a method that performs any initialization before applying updates.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoDelete (see page 18550)	Defines a method that deletes a record.
V	DoInsert (see page 18550)	Defines a method that inserts a record.
V	DoUpdate (see page 18551)	Defines a method that modifies a record.
V	EndUpdate (see page 18551)	Provides the interface for a method that performs any finalization after applying updates.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleUpdateError (see page 18493)	Manages the resolving process when errors occur while applying updates.
V	InitializeConflictBuffer (see page 18552)	Introduces a method for adding information about a failed update that is passed to an OnUpdateError event handler.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	LogUpdateError (see page 18495)	This is LogUpdateError, a member of class TCustomResolver.
V	LogUpdateRecord (see page 18495)	Adds records to the error log if their data has been changed by another application.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Provider (see page 18553)	Indicates the provider that uses this resolver to apply updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	ApplyUpdates (see page 18490)	Applies updates received from a provider.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 18548)	Creates and initializes a dataset resolver.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 18487)	Frees an instance of a resolver object.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	FreeTreeData (see page 18492)	This is FreeTreeData, a member of class TCustomResolver.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	InitKeyFields (see page 18494)	Sets the provider flag pfInKey on all key fields in a delta data packet.
V	InitTreeData (see page 18494)	This is InitTreeData, a member of class TCustomResolver.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	InternalBeforeResolve (see page 18552)	Initializes internal data structures once the delta packet has been wrapped in an update tree but before any updates are applied.
V	InternalUpdateRecord (see page 18495)	Manages the updating process once a delta packet is wrapped in a TUpdateTree (see page 18600) object.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	RowRequest (see page 18496)	Returns information from a specified record.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	UpdateTree (see page 18497)	This is UpdateTree, a member of class TCustomResolver.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

Strict private Fields

	Name	Description
	FBookmark (see page 18548)	This is FBookmark, a member of class TDataSetResolver.
	FErrorCount (see page 18488)	This is FErrorCount, a member of class TCustomResolver.
	FMaxErrors (see page 18488)	This is FMaxErrors, a member of class TCustomResolver.
	FOpened (see page 18549)	This is FOpened, a member of class TDataSetResolver.
	FPrevResponse (see page 18488)	This is FPrevResponse, a member of class TCustomResolver.
	FProvider (see page 18489)	This is FProvider, a member of class TCustomResolver.
	FUpdateTree (see page 18489)	This is FUpdateTree, a member of class TCustomResolver.

Strict private Methods

	Name	Description
	GetProvider (see page 18551)	This is GetProvider, a member of class TDataSetResolver.
	PutRecord (see page 18552)	This is PutRecord, a member of class TDataSetResolver.

124.1.8.2 TDataSetResolver.Create Constructor

Creates and initializes a dataset resolver.

Pascal

```
constructor Create(AProvider: TDataSetProvider); reintroduce;
```

C++

```
__fastcall TDataSetResolver(TDataSetProvider AProvider);
```

Description

Provider (see page 18553) components call Create when they need a resolver component for applying updates to a dataset. The AProvider parameter specifies the provider that creates and uses the resolver.

Do not create resolver instances directly. Resolvers are designed to be created and used by providers, and can not function independently of a provider.

See Also

Resolver (see page 18441)

124.1.8.3 TDataSetResolver Fields

The fields of the TDataSetResolver class are listed here.

Strict private Fields

	Name	Description
	FBookmark (see page 18548)	This is FBookmark, a member of class TDataSetResolver.
	FOpened (see page 18549)	This is FOpened, a member of class TDataSetResolver.

124.1.8.3.1 TDataSetResolver.FBookmark Field

Pascal

```
FBookmark: TBookmarkStr;
```

C++

```
TBookmarkStr FBookmark;
```

Description

This is FBookmark, a member of class TDataSetResolver.

124.1.8.3.2 TDataSetResolver.FOpened Field

Pascal

```
FOpened: Boolean;
```

C++

```
Boolean FOpened;
```

Description

This is FOpened, a member of class TDataSetResolver.

124.1.8.4 TDataSetResolver Methods

The methods of the TDataSetResolver class are listed here.

Protected Methods

	Name	Description
	BeginUpdate (see page 18549)	Provides the interface for a method that performs any initialization before applying updates.
	DoDelete (see page 18550)	Defines a method that deletes a record.
	DoInsert (see page 18550)	Defines a method that inserts a record.
	DoUpdate (see page 18551)	Defines a method that modifies a record.
	EndUpdate (see page 18551)	Provides the interface for a method that performs any finalization after applying updates.
	InitializeConflictBuffer (see page 18552)	Introduces a method for adding information about a failed update that is passed to an OnUpdateError event handler.

Public Methods

	Name	Description
	InternalBeforeResolve (see page 18552)	Initializes internal data structures once the delta packet has been wrapped in an update tree but before any updates are applied.

Strict private Methods

	Name	Description
	GetProvider (see page 18551)	This is GetProvider, a member of class TDataSetResolver.
	PutRecord (see page 18552)	This is PutRecord, a member of class TDataSetResolver.

124.1.8.4.1 TDataSetResolver.BeginUpdate Method

Provides the interface for a method that performs any initialization before applying updates.

Pascal

```
procedure BeginUpdate; override;
```

C++

```
virtual __fastcall BeginUpdate();
```

Description

The ApplyUpdates method calls BeginUpdate before applying updates in a delta packet.

As implemented in TCustomResolver, BeginUpdate does nothing. Descendants that need take specific action before applying updates override this method.

See Also

EndUpdate (see page 18492)

ApplyUpdates (see page 18490)

124.1.8.4.2 TDataSetResolver.DoDelete Method

Defines a method that deletes a record.

Pascal

```
procedure DoDelete(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall DoDelete(TUpdateTree Tree);
```

Description

DoDelete is one of several helper methods internal to the ApplyUpdates method. These helper methods perform the actual updates in the delta packet. ApplyUpdates wraps the delta packet in a TUpdateTree object. For each record that must be deleted, the resolver calls DoDelete.

TCustomResolver does not implement the DoDelete method. Descendants override this method to delete records in their own specialized way.

See Also

ApplyUpdates (see page 18490)

DoInsert (see page 18491)

DoUpdate (see page 18491)

124.1.8.4.3 TDataSetResolver.DoInsert Method

Defines a method that inserts a record.

Pascal

```
procedure DoInsert(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall DoInsert(TUpdateTree Tree);
```

Description

DoInsert is one of several helper methods internal to the ApplyUpdates method. These helper methods perform the actual updates in the delta packet. ApplyUpdates wraps the delta packet in a TUpdateTree object. For each record that must be inserted, the resolver calls DoInsert.

TCustomResolver does not implement the DoInsert method. Descendants override this method to insert records in their own specialized way.

See Also

ApplyUpdates (see page 18490)

DoDelete (see page 18491)

DoUpdate (see page 18491)

124.1.8.4.4 TDataSetResolver.DoUpdate Method

Defines a method that modifies a record.

Pascal

```
procedure DoUpdate(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall DoUpdate(TUpdateTree Tree);
```

Description

DoUpdate is one of several helper methods internal to the ApplyUpdates method. These helper methods perform the actual updates passed in the delta packet. ApplyUpdates wraps the delta packet in a TUpdateTree object. For each record that must be modified, the resolver calls DoUpdate.

TCustomResolver does not implement the DoUpdate method. Descendants override this method to insert records in their own specialized way.

See Also

ApplyUpdates (see page 18490)

DoDelete (see page 18491)

DoInsert (see page 18491)

124.1.8.4.5 TDataSetResolver.EndUpdate Method

Provides the interface for a method that performs any finalization after applying updates.

Pascal

```
procedure EndUpdate; override;
```

C++

```
virtual __fastcall EndUpdate();
```

Description

The ApplyUpdates method calls EndUpdate after applying updates.

As implemented in TCustomResolver, EndUpdate does nothing. Descendants that need take specific action after applying updates override this method.

See Also

BeginUpdate (see page 18490)

ApplyUpdates (see page 18490)

124.1.8.4.6 TDataSetResolver.GetProvider Method

Pascal

```
function GetProvider: TDataSetProvider;
```

C++

```
__fastcall TDataSetProvider GetProvider();
```

Description

This is GetProvider, a member of class TDataSetResolver.

124.1.8.4.7 TDataSetResolver.InitializeConflictBuffer Method

Introduces a method for adding information about a failed update that is passed to an OnUpdateError event handler.

Pascal

```
procedure InitializeConflictBuffer(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall InitializeConflictBuffer(TUpdateTree Tree);
```

Description

The HandleUpdateError method calls InitializeConflictBuffer before generating an OnUpdateError event. This ensures that the client dataset passed to an OnUpdateError event handler represents the update that caused the problem, with appropriate values for field's CurValue property.

Descendants of TCustomResolver implement this method to obtain information from the dataset or server to which they are applying updates.

See Also

HandleUpdateError (see page 18493)

OnUpdateError (see page 18444)

124.1.8.4.8 TDataSetResolver.InternalBeforeResolve Method

Initializes internal data structures once the delta packet has been wrapped in an update tree but before any updates are applied.

Pascal

```
procedure InternalBeforeResolve(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall InternalBeforeResolve(TUpdateTree Tree);
```

Description

The ApplyUpdates method wraps delta packets in a TUpdateTree object. Once this is complete, InternalBeforeResolve performs any other initialization before the updates are applied.

As implemented in TCustomResolver, InternalBeforeResolve does nothing. Descendants override this method to perform class-specific initialization.

See Also

ApplyUpdates (see page 18490)

inittreedata (see page 18494)

124.1.8.4.9 TDataSetResolver.PutRecord Method

Pascal

```
procedure PutRecord(Tree: TUpdateTree);
```

C++

```
__fastcall PutRecord(TUpdateTree Tree);
```

Description

This is PutRecord, a member of class TDataSetResolver.

124.1.8.5 TDataSetResolver Properties

The properties of the TDataSetResolver class are listed here.

Protected Properties

	Name	Description
	Provider (see page 18553)	Indicates the provider that uses this resolver to apply updates.

124.1.8.5.1 TDataSetResolver.Provider Property

Indicates the provider that uses this resolver to apply updates.

Pascal

```
property Provider: TDataSetProvider;
```

C++

```
__property TDataSetProvider Provider;
```

Description

Provider is a read-only property that points to the provider that uses this resolver for applying updates. The Provider property is set in the constructor.

Resolvers use the Provider property internally to obtain information supplied in the provider's properties. They also use it to generate events on the provider at appropriate points in the update process.

See Also

Create (see page 18487)

Resolver (see page 18441)

124.1.9 TLocalAppServer Class

Class Hierarchy

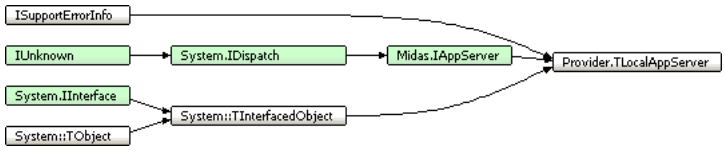

Pascal

```
TLocalAppServer = class(TInterfacedObject, IAppServer, ISupportErrorInfo);
```

C++

```
class TLocalAppServer : public TInterfacedObject, public IAppServer, public ISupportErrorInfo;
```

File

Provider

Description

This is class Provider.TLocalAppServer.

124.1.9.1 TLocalAppServer Members

The following tables list the members exposed by TLocalAppServer.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	AS_ApplyUpdates (see page 18557)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 18558)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 18559)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 18560)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 18560)	Returns a list of all the available providers on a remote data module.
	AS_GetRecords (see page 18561)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 18562)	Returns information from a specified record of the provider's dataset.
	GetIDsOfNames (see page 18563)	This is GetIDsOfNames, a member of class TLocalAppServer.
	GetTypeInfo (see page 18563)	This is GetTypeInfo, a member of class TLocalAppServer.
	GetTypeInfoCount (see page 18563)	This is GetTypeInfoCount, a member of class TLocalAppServer.
	InterfaceSupportsErrorInfo (see page 18563)	This is InterfaceSupportsErrorInfo, a member of class TLocalAppServer.
	Invoke (see page 18564)	This is Invoke, a member of class TLocalAppServer.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 18555)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

	Destroy (see page 18556)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 18564)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

Strict private Fields

	Name	Description
	FProvider (see page 18557)	This is FProvider, a member of class TLocalAppServer.
	FProviderCreated (see page 18557)	This is FProviderCreated, a member of class TLocalAppServer.

124.1.9.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TLocalAppServer.Create (TCustomProvider) (see page 18555)	This is Create, a member of class TLocalAppServer.
	TLocalAppServer.Create (TDataset) (see page 18556)	This is Create, a member of class TLocalAppServer.

124.1.9.2.1 TLocalAppServer.Create Constructor (TCustomProvider)

Pascal

```
constructor Create(AProvider: TCustomProvider); overload;
```

C++

```
__fastcall TLocalAppServer(TCustomProvider AProvider);
```

Description

This is Create, a member of class TLocalAppServer.

124.1.9.2.2 TLocalAppServer.Create Constructor (TDataset)

Pascal

```
constructor Create(ADataset: TDataset); overload;
```

C++

```
__fastcall TLocalAppServer(TDataset ADataset);
```

Description

This is Create, a member of class TLocalAppServer.

124.1.9.3 TLocalAppServer.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TLocalAppServer();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

124.1.9.4 TLocalAppServer Fields

The fields of the TLocalAppServer class are listed here.

Strict private Fields

	Name	Description
	FProvider (see page 18557)	This is FProvider, a member of class TLocalAppServer.
	FProviderCreated (see page 18557)	This is FProviderCreated, a member of class TLocalAppServer.

124.1.9.4.1 TLocalAppServer.FProvider Field

Pascal

```
FProvider: TCustomProvider;
```

C++

```
TCustomProvider FProvider;
```

Description

This is FProvider, a member of class TLocalAppServer.

124.1.9.4.2 TLocalAppServer.FProviderCreated Field

Pascal

```
FProviderCreated: Boolean;
```

C++

```
Boolean FProviderCreated;
```

Description

This is FProviderCreated, a member of class TLocalAppServer.

124.1.9.5 TLocalAppServer Methods

The methods of the TLocalAppServer class are listed here.

Protected Methods

	Name	Description
	AS_ApplyUpdates (see page 18557)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 18558)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 18559)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 18560)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 18560)	Returns a list of all the available providers on a remote data module.
	AS_GetRecords (see page 18561)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 18562)	Returns information from a specified record of the provider's dataset.
	GetIDsOfNames (see page 18563)	This is GetIDsOfNames, a member of class TLocalAppServer.
	GetTypeInfo (see page 18563)	This is GetTypeInfo, a member of class TLocalAppServer.
	GetTypeInfoCount (see page 18563)	This is GetTypeInfoCount, a member of class TLocalAppServer.
	InterfaceSupportsErrorInfo (see page 18563)	This is InterfaceSupportsErrorInfo, a member of class TLocalAppServer.
	Invoke (see page 18564)	This is Invoke, a member of class TLocalAppServer.

Public Methods

	Name	Description
V	SafeCallException (see page 18564)	Handles exceptions in methods declared using the safecall calling convention.

124.1.9.5.1 TLocalAppServer.AS_ApplyUpdates Method

Applies updates received from a client dataset using a specified provider.

Pascal

```
function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
```

```
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_ApplyUpdates(const BSTR ProviderName, OleVariant Delta, int
MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

AS_AS_ApplyUpdates writes client updates to a dataset or database server.

ProviderName specifies the name of the provider component that manages the update operation.

Delta is a Variant containing the updated, inserted, and deleted records to write to the database.

MaxErrors indicates the maximum number of errors to permit before stopping the update operation.

ErrorCount returns the actual number of errors encountered during the update operation.

OwnerData is information supplied by the client application in a BeforeAS_ApplyUpdates event handler and returned to the client application in an AfterAS_ApplyUpdates event handler.

AS_AS_ApplyUpdates (Delphi) or AS_ApplyUpdates_result parameter (C++) returns a Variant that is a data packet containing all records that could not be applied to the database.

See Also

ApplyUpdates (see page 8394)

applyupdates (see page 18459)

BeforeApplyUpdates (see page 18479)

AfterApplyUpdates (see page 18475)

BeforeApplyUpdates (see page 8511)

AfterApplyUpdates (see page 8497)

AS_ApplyUpdates (see page 6590)

as_applyupdates (see page 19892)

Communicating with the Client Dataset

124.1.9.5.2 TLocalAppServer.AS_DataRequest Method

Generates an OnAS_DataRequest event on the specified provider component.

Pascal

```
function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

Use AS_AS_DataRequest to invoke an OnAS_DataRequest event handler of the provider named by ProviderName. This method is provided to allow application developers to customize the communication between a client application and a provider component. There is no predefined meaning for the Data parameter or the return value (Delphi) or the AS_DataRequest_result parameter (C++), which is returned to the caller.

See Also

OnDataRequest (see page 18482)

DataRequest (see page 8404)

Communicating with the Client Dataset

124.1.9.5.3 TLocalAppServer.AS_Execute Method

AS_Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); safecall;
```

C++

```
__fastcall AS_Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant Params,
OleVariant OwnerData);
```

Description

Use AS_AS_Execute to pass an execute command to a remote provider, using a specified set of parameters. AS_AS_Execute calls the AS_Execute method of the specified provider after assigning any parameters. Output parameters that result from executing a query or stored procedure are returned in the Params parameter.

The provider's AS_Execute method generates a BeforeAS_Execute event, passes the command on to its associated dataset, and generates an AfterAS_Execute event. The dataset must represent a query or stored procedure that does not return a cursor.

The ProviderName parameter specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the query, or the name of a stored procedure that replaces the dataset's stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

The Params parameter encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams routine. To decode Params into a TParams object, use the Unpackparams routine.

The OwnerData parameter contains custom information that appears as an argument to the provider's BeforeAS_Execute and AfterAS_Execute event handlers. This information originates in a client dataset's BeforeAS_Execute event handler. The value returned in OwnerData is passed to the client dataset's AfterAS_Execute event handler.

See Also

DataSet (see page 18529)

beforeexecute (see page 18479)

AfterExecute (see page 18476)

Params (see page 8489)

Execute (see page 8413)

Execute (see page 18466)

BeforeExecute (see page 8514)

AfterExecute (see page 8504)

Options (see page 18440)

PackageParams (see page 8534)

UnpackParams (see page 8534)

Communicating with the Client Dataset

124.1.9.5.4 TLocalAppServer.AS_GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

Use AS_AS_GetParams to retrieve the current parameter values from the dataset associated with a provider.

The ProviderName parameter specifies the name of the provider component whose dataset has the parameters.

The OwnerData represents custom information that originates in a client dataset's BeforeAS_GetParams event handler and returns information that is passed to the client dataset's AfterAS_GetParams event handler.

AS_GetParams_result (C++) or AS_AS_GetParams (Delphi) returns the parameters, encoded as a Variant array, where each element is a 2-element Variant array containing a parameter name and a parameter value. These parameters can be converted to a TParams object using the UnpackParams procedure. If the dataset has no parameters, or if the provider does not support parameter fetching, AS_GetParams_result (C++) or AS_AS_GetParams (Delphi) returns a Null Variant.

Note: AS_AS_GetParams should not be used to retrieve output parameters from stateless application servers, because parameter values may be changed by other applications. When writing a stateless application server, retrieve parameter values using the AS_Execute method instead.

See Also

DataSet (see page 18529)

Params (see page 8489)

FetchParams (see page 8415)

GetParams (see page 18467)

BeforeGetParams (see page 18480)

aftergetparams (see page 18477)

AS_Execute (see page 16453)

UnpackParams (see page 8534)

Communicating with the Client Dataset

124.1.9.5.5 TLocalAppServer.AS_GetProviderNames Method

Returns a list of all the available providers on a remote data module.

Pascal

```
function AS_GetProviderNames: OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetProviderNames();
```

Description

Call AS_AS_GetProviderNames for a list of all providers available from a remote data module. These values indicate the possible values for the ProviderName property of a client dataset that uses the IAppServer interface of an application server. When calling any other IAppServer method, pass one of these names to indicate the provider that is the target of the method call.

AS_AS_GetProviderNames returns the names of the providers in a Variant array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

See Also

providers (see page 6599)

ProviderName (see page 8492)

Providers (see page 19908)

Communicating with the Client Dataset

124.1.9.5.6 TLocalAppServer.AS_GetRecords Method

Returns a data packet that contains the specified data.

Pascal

```
function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetRecords(const BSTR ProviderName, int Count, int RecsOut, int
Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

Call AS_AS_GetRecords to retrieve a specified number of records. AS_GetRecords returns the requested records, starting with the current record of the provider's dataset. When working with a stateless remote data module, you may need to reposition the cursor or re-execute a query or stored procedure in the provider's BeforeAS_GetRecords event handler.

ProviderName gives the name of the provider component that provides the records.

Count indicates the number or type of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only Count records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants. For example, the following value requests metadata along with a data packet encoded in XML:

```
MetaDataOption + XMLOption
MetaDataOption || XMLOption
```

CommandText specifies an optional SQL statement that replaces the SQL of a query, or the name of a table or stored procedure that replaces the provider's table or stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

Params is a Variant containing any parameters that should be passed to the provider's dataset before it executes to generate the requested data. It returns any output parameters. You can convert between the Variant format for parameters used by this argument and a TParams object using the global PackageParams and UnpackParams routines.

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_GetRecords event handler. This information is passed to the provider's BeforeAS_GetRecords event handler. OwnerData returns information supplied by the

provider's AfterAS_GetRecords event handler.

Records are returned as a data packet in a Variantparameter, AS_GetRecords_result (C++).

See Also

Data (see page 8467)

Params (see page 8489)

GetNextPacket (see page 8423)

GetRecords (see page 18468)

BeforeGetRecords (see page 8515)

BeforeGetRecords (see page 18480)

AfterGetRecords (see page 18477)

Options (see page 18440)

PackageParams (see page 8534)

UnpackParams (see page 8534)

Communicating with the Client Dataset

124.1.9.5.7 TLocalAppServer.AS_RowRequest Method

Returns information from a specified record of the provider's dataset.

Pascal

```
function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_RowRequest(const BSTR ProviderName, OleVariant Row, int RequestType,
OleVariant OwnerData);
```

Description

Client datasets call AS_AS_RowRequest automatically to implement their FetchBlobs, FetchDetails , or RefreshRecord method.

The ProviderName parameter indicates the provider associated with the dataset from which information should be fetched.

The Row parameter is an OleVariant that describes the current record on the client dataset.

The RequestType parameter indicates the type of information required. It is an integer version of the TFetchOptions type. (An integer because the value may be marshaled to a remote application server). To create a value for RequestType, take the corresponding TFetchOptions value, cast it to a Byte, and cast the result to an Integer:

```
Integer(Byte([foBlobs, foDetails]));
(int) (Byte) (TFetchOptions() << foBlobs << foDetails);
```

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_RowRequest event handler. This information is passed to the provider's BeforeAS_RowRequest event handler. OwnerData returns custom information supplied by the provider's AfterAS_RowRequest event handler.

The requested data is returned as a delta packet in AS_RowRequest_result (C++).

See Also

RowRequest (see page 18472)

BeforeRowRequest (see page 18482)

AfterRowRequest (see page 18478)

FetchBlobs (see page 8414)

FetchDetails (see page 8414)

RefreshRecord (see page 8442)

Communicating with the Client Dataset

124.1.9.5.8 TLocalAppServer.GetIDsOfNames Method

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int NameCount, int
LocaleID, void * DispIDs);
```

Description

This is GetIDsOfNames, a member of class TLocalAppServer.

124.1.9.5.9 TLocalAppServer.GetTypeInfo Method

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

This is GetTypeInfo, a member of class TLocalAppServer.

124.1.9.5.10 TLocalAppServer.GetTypeInfoCount Method

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

This is GetTypeInfoCount, a member of class TLocalAppServer.

124.1.9.5.11 TLocalAppServer.InterfaceSupportsErrorInfo Method

Pascal

```
function InterfaceSupportsErrorInfo(const iid: TGUID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT InterfaceSupportsErrorInfo(const TGUID iid);
```

Description

This is InterfaceSupportsErrorInfo, a member of class TLocalAppServer.

124.1.9.5.12 TLocalAppServer.Invoke Method

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word Flags,
Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is Invoke, a member of class TLocalAppServer.

124.1.9.5.13 TLocalAppServer.SafeCallException Method

Handles exceptions in methods declared using the safecall calling convention.

Pascal

```
function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): HResult; override;
```

C++

```
virtual __fastcall HRESULT SafeCallException(TObject * ExceptObject, void * ExceptAddr);
```

Description

System::TObject::SafeCallException handles exceptions in methods that use the safecall calling convention. Some classes that implement interfaces override this method to handle errors that might occur.

As implemented in System::TObject::TObject, System::TObject::SafeCallException simply returns E_UNEXPECTED. This is the appropriate response for classes that do no support interfaces

124.1.10 TPacketDataSet Class

TPacketDataSet represents a data packet or delta packet that is used by a provider.

Class Hierarchy

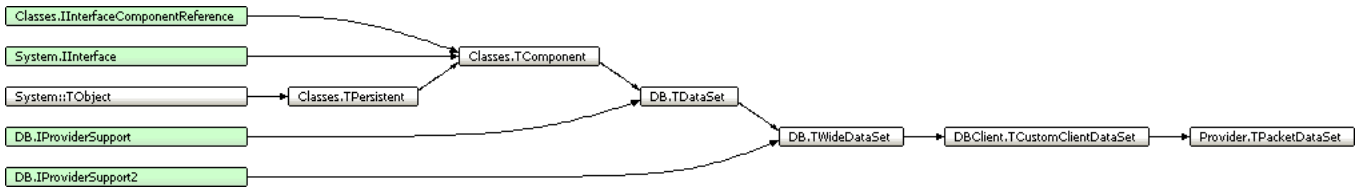

Pascal

```
TPacketDataSet = class(TCustomClientDataSet);
```

C++

```
class TPacketDataSet : public TCustomClientDataSet;
```

File

Provider

Description

TPacketDataSet represents a data packet that a provider creates to represent its dataset's records and metadata or a delta data packet that a provider receives when it should apply updates. Provider (see page 18419) components create instances of TPacketDataSet to represent data packets before they are sent to clients and to represent delta packets after they are received from clients.

TPacketDataSet provides a convenient way for providers and resolvers to work with data packets. As a descendant of TCustomClientDataSet, TPacketDataSet allows providers and resolvers to use any of the standard dataset properties and methods to manipulate the data packet's records. In addition, TPacketDataSet introduces new properties and methods that describe what information goes into the data packet or how updates should be applied.

Applications do not need to work directly with packet datasets. They are mainly of use to component writers who want to create custom provider components.

See Also

TCUstomClientDataSet

TCustomProvider (see page 18449)

124.1.10.1 TPacketDataSet Members

The following tables list the members exposed by TPacketDataSet.

Protected Events

	Name	Description
	AfterApplyUpdates (see page 8497)	Occurs after the client dataset applies updates to the provider but before it attempts to reconcile update errors.
	AfterCancel (see page 8498)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 8500)	Occurs after an application closes a dataset.
	AfterDelete (see page 8501)	Occurs after an application deletes a record.
	AfterEdit (see page 8504)	Occurs after an application starts editing a record.
	AfterExecute (see page 8504)	Occurs after the client dataset receives the results from an execute command.
	AfterGetParams (see page 8505)	Occurs after the client dataset receives parameter values from the provider.
	AfterGetRecords (see page 8506)	Occurs after the client dataset fetches a data packet from the provider.
	AfterInsert (see page 8507)	Occurs after an application inserts a new record.
	AfterOpen (see page 8507)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 8509)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 8509)	Occurs after an application refreshes the data in the dataset.
	AfterRowRequest (see page 8509)	Occurs after the client dataset fetches new information about the current record.
	AfterScroll (see page 8510)	Occurs after an application scrolls from one record to another.
	BeforeApplyUpdates (see page 8511)	Occurs before the client dataset applies updates to the remote server.
	BeforeCancel (see page 8512)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 8512)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 8512)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 8513)	Occurs before an application enters edit mode for the active record.
	BeforeExecute (see page 8514)	Occurs before the client dataset executes the provider's query or stored procedure.
	BeforeGetParams (see page 8514)	Occurs before the client dataset fetches parameter values from the provider.

	BeforeGetRecords (see page 8515)	Occurs before the client dataset fetches a data packet from the provider.
	BeforeInsert (see page 8517)	Occurs before an application enters insert mode.
	BeforeOpen (see page 8518)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 8518)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 8519)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeRowRequest (see page 8520)	Occurs before the client dataset fetches new information about the current record.
	BeforeScroll (see page 8520)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 8521)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 8522)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 8522)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 8523)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 8523)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 8524)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnReconcileError (see page 8524)	Occurs when a client dataset needs to reconcile an update to a record that could not be applied.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	ActivateFilters (see page 8391)	This is ActivateFilters, a member of class TCustomClientDataSet.
V	AddDataPacket (see page 8391)	This is AddDataPacket, a member of class TCustomClientDataSet.
	AddFieldDesc (see page 8391)	This is AddFieldDesc, a member of class TCustomClientDataSet.
	AllocKeyBuffers (see page 8392)	This is AllocKeyBuffers, a member of class TCustomClientDataSet.
V	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
V	AllocRecordBuffer (see page 8393)	This is AllocRecordBuffer, a member of class TCustomClientDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Check (see page 8397)	This is Check, a member of class TCustomClientDataSet.
V	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
V	CheckDetailRecords (see page 8397)	This is CheckDetailRecords, a member of class TCustomClientDataSet.
V	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
V	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
V	CheckProviderEOF (see page 8397)	This is CheckProviderEOF, a member of class TCustomClientDataSet.
	CheckSetKeyMode (see page 8397)	This is CheckSetKeyMode, a member of class TCustomClientDataSet.
V	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
V	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
V	ClearCalcFields (see page 8398)	This is ClearCalcFields, a member of class TCustomClientDataSet.
V	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
V	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
V	CloseCursor (see page 8400)	This is CloseCursor, a member of class TCustomClientDataSet.

	ConstraintsStored (see page 8401)	This is ConstraintsStored, a member of class TCustomClientDataSet.
	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
	DataConvert (see page 8404)	This is DataConvert, a member of class TCustomClientDataSet.
	DataEvent (see page 18583)	This is DataEvent, a member of class TPacketDataSet.
	DeactivateFilters (see page 8405)	This is DeactivateFilters, a member of class TCustomClientDataSet.
	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
	DefChanged (see page 8405)	This is DefChanged, a member of class TCustomClientDataSet.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 8405)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
	DestroyLookupCursor (see page 8406)	This is DestroyLookupCursor, a member of class TCustomClientDataSet.
	DoAfterApplyUpdates (see page 8407)	This is DoAfterApplyUpdates, a member of class TCustomClientDataSet.
	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
	DoAfterExecute (see page 8407)	This is DoAfterExecute, a member of class TCustomClientDataSet.
	DoAfterGetParams (see page 8407)	This is DoAfterGetParams, a member of class TCustomClientDataSet.
	DoAfterGetRecords (see page 8407)	This is DoAfterGetRecords, a member of class TCustomClientDataSet.
	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterRowRequest (see page 8408)	This is DoAfterRowRequest, a member of class TCustomClientDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoApplyUpdates (see page 8408)	This is DoApplyUpdates, a member of class TCustomClientDataSet.
	DoBeforeApplyUpdates (see page 8408)	This is DoBeforeApplyUpdates, a member of class TCustomClientDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeExecute (see page 8408)	This is DoBeforeExecute, a member of class TCustomClientDataSet.
	DoBeforeGetParams (see page 8408)	This is DoBeforeGetParams, a member of class TCustomClientDataSet.
	DoBeforeGetRecords (see page 8409)	This is DoBeforeGetRecords, a member of class TCustomClientDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeInsert (see page 8409)	This is DoBeforeInsert, a member of class TCustomClientDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeRowRequest (see page 8409)	This is DoBeforeRowRequest, a member of class TCustomClientDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoExecute (see page 8409)	This is DoExecute, a member of class TCustomClientDataSet.
	DoGetRecords (see page 8410)	This is DoGetRecords, a member of class TCustomClientDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	DoOnNewRecord (see page 8410)	This is DoOnNewRecord, a member of class TCustomClientDataSet.
	DoRowRequest (see page 8410)	This is DoRowRequest, a member of class TCustomClientDataSet.
	FetchMoreData (see page 8415)	This is FetchMoreData, a member of class TCustomClientDataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.

	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FindRecord (see page 8417)	This is FindRecord, a member of class TCustomClientDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeKeyBuffers (see page 8417)	This is FreeKeyBuffers, a member of class TCustomClientDataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
	FreeRecordBuffer (see page 8417)	This is FreeRecordBuffer, a member of class TCustomClientDataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
	GetAggregateValue (see page 8418)	This is GetAggregateValue, a member of class TCustomClientDataSet.
	GetAppServer (see page 8418)	This is GetAppServer, a member of class TCustomClientDataSet.
	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
	GetBookmarkData (see page 8418)	This is GetBookmarkData, a member of class TCustomClientDataSet.
	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
	GetBookmarkFlag (see page 8418)	This is GetBookmarkFlag, a member of class TCustomClientDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
	GetCanModify (see page 8418)	This is GetCanModify, a member of class TCustomClientDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
	GetDataSource (see page 8419)	This is GetDataSource, a member of class TCustomClientDataSet.
	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIndexField (see page 8421)	This is GetIndexField, a member of class TCustomClientDataSet.
	GetIndexFieldCount (see page 8421)	This is GetIndexFieldCount, a member of class TCustomClientDataSet.
	GetIsClone (see page 8422)	This is GetIsClone, a member of class TCustomClientDataSet.
	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
	GetIsIndexField (see page 8422)	This is GetIsIndexField, a member of class TCustomClientDataSet.
	GetKeyBuffer (see page 8423)	This is GetKeyBuffer, a member of class TCustomClientDataSet.
	GetKeyExclusive (see page 8423)	This is GetKeyExclusive, a member of class TCustomClientDataSet.
	GetKeyFieldCount (see page 8423)	This is GetKeyFieldCount, a member of class TCustomClientDataSet.
	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
	GetRecNo (see page 8425)	This is GetRecNo, a member of class TCustomClientDataSet.
	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
	GetRecord (see page 8425)	This is GetRecord, a member of class TCustomClientDataSet.
	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
	GetRecordCount (see page 8426)	This is GetRecordCount, a member of class TCustomClientDataSet.
	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
	GetRecordSize (see page 8426)	This is GetRecordSize, a member of class TCustomClientDataSet.
	GetRemoteServer (see page 8426)	This is GetRemoteServer, a member of class TCustomClientDataSet.
	GetStateFieldValue (see page 18584)	This is GetStateFieldValue, a member of class TPacketDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
	InitKeyBuffer (see page 8428)	This is InitKeyBuffer, a member of class TCustomClientDataSet.
V	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
V	InitRecord (see page 8428)	This is InitRecord, a member of class TCustomClientDataSet.
V	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
V	InternalAddRecord (see page 8428)	This is InternalAddRecord, a member of class TCustomClientDataSet.
V	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
V	InternalCancel (see page 8429)	This is InternalCancel, a member of class TCustomClientDataSet.
V	InternalClose (see page 18585)	This is InternalClose, a member of class TPacketDataSet.
V	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
V	InternalDelete (see page 8429)	This is InternalDelete, a member of class TCustomClientDataSet.
V	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
V	InternalEdit (see page 8429)	This is InternalEdit, a member of class TCustomClientDataSet.
V	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
V	InternalFirst (see page 8430)	This is InternalFirst, a member of class TCustomClientDataSet.
	InternalGetOptionalParam (see page 8430)	This is InternalGetOptionalParam, a member of class TCustomClientDataSet.
V	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
V	InternalGotoBookmark (see page 8430)	This is InternalGotoBookmark, a member of class TCustomClientDataSet.
A	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
V	InternalHandleException (see page 8430)	This is InternalHandleException, a member of class TCustomClientDataSet.
A	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
V	InternalInitFieldDefs (see page 8430)	This is InternalInitFieldDefs, a member of class TCustomClientDataSet.
V	InternalInitRecord (see page 18586)	This is InternalInitRecord, a member of class TPacketDataSet.
V	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
V	InternalInsert (see page 8431)	This is InternalInsert, a member of class TCustomClientDataSet.
V	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
V	InternalLast (see page 8431)	This is InternalLast, a member of class TCustomClientDataSet.
V	InternalOpen (see page 18586)	This is InternalOpen, a member of class TPacketDataSet.
V	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
V	InternalPost (see page 8432)	This is InternalPost, a member of class TCustomClientDataSet.
V	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
V	InternalRefresh (see page 8432)	This is InternalRefresh, a member of class TCustomClientDataSet.
	InternalSetOptionalParam (see page 8432)	This is InternalSetOptionalParam, a member of class TCustomClientDataSet.
V	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
V	InternalSetToRecord (see page 8432)	This is InternalSetToRecord, a member of class TCustomClientDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
A	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
V	IsCursorOpen (see page 8432)	This is IsCursorOpen, a member of class TCustomClientDataSet.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 8433)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 8436)	This is LocateRecord, a member of class TCustomClientDataSet.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 8437)	Forwards notification messages to all owned components.
V	OpenCursor (see page 6976)	This is OpenCursor, a member of class TDataSet.
V	OpenCursor (see page 8438)	This is OpenCursor, a member of class TCustomClientDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PostKeyBuffer (see page 8439)	This is PostKeyBuffer, a member of class TCustomClientDataSet.
V	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.

	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSExecuteStatement (see page 7535)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 6982)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 7537)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 8439)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandTextW (see page 7537)	This is PSGetCommandTextW, a member of class TWideDataSet.
	PSGetCommandType (see page 6982)	Gets the command type to be executed by PSExecute.
	PSGetCommandType (see page 8440)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 8440)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7538)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetKeyFieldsW (see page 7538)	This is PSGetKeyFieldsW, a member of class TWideDataSet.
	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7538)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetQuoteCharW (see page 7539)	This is PSGetQuoteCharW, a member of class TWideDataSet.
	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7539)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetTableNameW (see page 7539)	This is PSGetTableNameW, a member of class TWideDataSet.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetCommandText (see page 7540)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadDataPacket (see page 8441)	This is ReadDataPacket, a member of class TCustomClientDataSet.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	RefreshInternalCalcFields (see page 8441)	This is RefreshInternalCalcFields, a member of class TCustomClientDataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	ResetAggField (see page 8442)	This is ResetAggField, a member of class TCustomClientDataSet.
	ResetCursorRange (see page 8443)	This is ResetCursorRange, a member of class TCustomClientDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAggregates (see page 8445)	This is SetAggregates, a member of class TCustomClientDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppServer (see page 8446)	This is SetAppServer, a member of class TCustomClientDataSet.

	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkData (see page 8446)	This is SetBookmarkData, a member of class TCustomClientDataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkFlag (see page 8446)	This is SetBookmarkFlag, a member of class TCustomClientDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCommandText (see page 8447)	This is SetCommandText, a member of class TCustomClientDataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetCursorRange (see page 8447)	This is SetCursorRange, a member of class TCustomClientDataSet.
	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDataSetField (see page 8447)	This is SetDataSetField, a member of class TCustomClientDataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldData (see page 18586)	This is SetFieldData, a member of class TPacketDataSet.
	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
	SetFilterData (see page 8448)	This is SetFilterData, a member of class TCustomClientDataSet.
	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
	SetFiltered (see page 8448)	This is SetFiltered, a member of class TCustomClientDataSet.
	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
	SetFilterOptions (see page 8448)	This is SetFilterOptions, a member of class TCustomClientDataSet.
	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
	SetFilterText (see page 8448)	This is SetFilterText, a member of class TCustomClientDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetIndexField (see page 8448)	This is SetIndexField, a member of class TCustomClientDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetKeyBuffer (see page 8449)	This is SetKeyBuffer, a member of class TCustomClientDataSet.
	SetKeyExclusive (see page 8449)	This is SetKeyExclusive, a member of class TCustomClientDataSet.
	SetKeyFieldCount (see page 8450)	This is SetKeyFieldCount, a member of class TCustomClientDataSet.
	SetKeyFields (see page 8450)	This is SetKeyFields, a member of class TCustomClientDataSet.
	SetLinkRanges (see page 8450)	This is SetLinkRanges, a member of class TCustomClientDataSet.
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetOnFilterRecord (see page 8450)	This is SetOnFilterRecord, a member of class TCustomClientDataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetProviderEOF (see page 8452)	This is SetProviderEOF, a member of class TCustomClientDataSet.
	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
	SetRecNo (see page 8454)	This is SetRecNo, a member of class TCustomClientDataSet.
	SetRemoteServer (see page 8454)	This is SetRemoteServer, a member of class TCustomClientDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
	SetStateFieldValue (see page 18586)	This is SetStateFieldValue, a member of class TPacketDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	SwitchToIndex (see page 8455)	This is SwitchToIndex, a member of class TCustomClientDataSet.
	SyncCursors (see page 8455)	This is SyncCursors, a member of class TCustomClientDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.
V	UpdateIndexDefs (see page 8455)	This is UpdateIndexDefs, a member of class TCustomClientDataSet.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteDataPacket (see page 8456)	This is WriteDataPacket, a member of class TCustomClientDataSet.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
	Aggregates (see page 8461)	Lists all aggregates that apply to the client dataset.
	AggregatesActive (see page 8461)	Specifies whether the client dataset calculates and maintains aggregate values.
	AutoCalcFields (see page 8463)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
	CommandText (see page 8465)	Specifies an SQL command to be executed by the database server.
	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
	Constraints (see page 8466)	Specifies record-level constraints that must be met when editing the data.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
	DisableStringTrim (see page 8470)	Specifies whether trailing spaces are trimmed from field values when records are posted.
	DSBase (see page 8471)	Specifies the interface the client dataset uses to communicate with classes in Midas.dll or Midaslib.dcu (on Windows) or midas.so (on Linux).
R	DSCursor (see page 8471)	Provides access to the interface for the client dataset's data cursor.
	FetchOnDemand (see page 8471)	Indicates whether to fetch data packets from the provider on an as-needed basis.
	FieldDefs (see page 8472)	Points to the list of field definitions for the dataset.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
	Filter (see page 8476)	Specifies the text of the current filter.
	Filtered (see page 8478)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 8478)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
	IndexDefs (see page 8480)	Contains information about the indexes for a client dataset.
	IndexName (see page 8483)	Identifies an index for the client dataset.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
R	IsClone (see page 8484)	This is IsClone, a member of class TCustomClientDataSet.
	MasterSource (see page 8486)	Specifies a data source component for the master dataset when establishing a detail-master relationship between this dataset and another one.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
	ObjectView (see page 8487)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.

	Params (see page 8489)	Contains parameter values that are sent to the provider.
	ProviderEOF (see page 8491)	Indicates whether the client dataset has all of its data.
R	Ranged (see page 8492)	This is Ranged, a member of class TCustomClientDataSet.
	ReadOnly (see page 8493)	Specifies whether the client dataset is read-only for this application.
	Reserved (see page 7035)	For internal use only.
	StoreDefs (see page 8495)	Indicates whether field and index definitions are saved with the client dataset.
	WritingCurValues (see page 18589)	This is WritingCurValues, a member of class TPacketDataSet.

Public Events

	Name	Description
	AfterCancel (see page 7038)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 7040)	Occurs after an application closes a dataset.
	AfterDelete (see page 7041)	Occurs after an application deletes a record.
	AfterEdit (see page 7044)	Occurs after an application starts editing a record.
	AfterInsert (see page 7044)	Occurs after an application inserts a new record.
	AfterOpen (see page 7045)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 7046)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 7047)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 7047)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 7048)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 7048)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 7049)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 7049)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 7050)	Occurs before an application enters insert mode.
	BeforeOpen (see page 7051)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 7052)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 7052)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 7053)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 7053)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 7054)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 7054)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 7056)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 7056)	Occurs when an application attempts to modify or insert a record and an exception is raised.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
	AddIndex (see page 8391)	Creates a new index for the client dataset.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendData (see page 8393)	Adds a newly fetched packet of records to those already stored in the client dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
	ApplyRange (see page 8393)	Applies a range to the dataset.
V	ApplyUpdates (see page 8394)	Sends all updated, inserted, and deleted records from the client dataset to the provider for writing to the database.

	Assign (see page 2930)	Copies the contents of another, similar object.
	AssignCurValues (see page 18582)	This is the overview for the AssignCurValues method overload.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
	BookmarkValid (see page 8395)	Tests the validity of a specified bookmark.
	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
	Cancel (see page 8395)	Cancels unposted changes to the current record.
	CancelRange (see page 8396)	Removes any ranges currently in effect for the client dataset.
	CancelUpdates (see page 8396)	Clears all pending edits and clears the change log.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	CloneCursor (see page 8398)	Shares the data belonging to another client dataset.
	Close (see page 6920)	Closes a dataset.
	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
	CompareBookmarks (see page 8400)	Compares two bookmarks.
	ConstraintsDisabled (see page 8400)	Indicates whether constraints are enabled for a client dataset.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
	Create (see page 18580)	Creates and initializes instance of TPacketDataSet (see page 18564).
	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
	CreateBlobStream (see page 8401)	Returns a TClientBlobStream (see page 8285) object for reading or writing the data in a specified blob field.
	CreateDataSet (see page 8401)	Creates a new, empty client dataset.
	CreateFromDelta (see page 18583)	This is CreateFromDelta, a member of class TPacketDataSet.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
	DataRequest (see page 8404)	Generates an OnDataRequest event on the associated provider
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
	DeleteIndex (see page 8406)	Deletes an index for the client dataset.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 6904)	Destroys the TDataSet (see page 6894) instance.
	Destroy (see page 8387)	Destroys the instance of a client dataset component.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableConstraints (see page 8406)	Disables the enforcement of constraints.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EditKey (see page 8410)	Enables modification of the search key buffer.
	EditRangeEnd (see page 8411)	Enables changing the ending value for an existing range.
	EditRangeStart (see page 8411)	Enables changing the starting value for an existing range.

	EmptyDataSet (see page 8412)	Removes all records from the dataset.
	EnableConstraints (see page 8412)	Reenables enforcement of constraints when records are posted.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	Execute (see page 8413)	Executes an SQL command by the provider's dataset.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FetchBlobs (see page 8414)	Retrieves BLOB field values from the provider.
	FetchDetails (see page 8414)	Retrieves nested detail datasets from the provider.
	FetchParams (see page 8415)	Retrieves parameter values from the provider.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindKey (see page 8416)	Searches for a record containing specified field values.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNearest (see page 8416)	Moves the cursor to the record that most closely matches a specified set of key values.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
	GetCurrentRecord (see page 8419)	Retrieves the current record into a buffer.
	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
	GetFieldData (see page 8419)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetGroupState (see page 8420)	Indicates where the current record sits within a specified group of records.
	GetIndexInfo (see page 8421)	Retrieves information about the current index.
	GetIndexNames (see page 8422)	Retrieves a list of available indexes for a client dataset.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNextPacket (see page 8423)	Retrieves the next block of records from the provider.
	GetOptionalParam (see page 8425)	Retrieves a custom parameter from the data packet.
	GetParentComponent (see page 2778)	Returns the containing Component.

	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	GotoCurrent (see page 8427)	Sets the current record in this dataset the same as the current record in another, cloned dataset.
	GotoKey (see page 8427)	Moves the cursor to a record specified by the current key.
	GotoNearest (see page 8427)	Moves the cursor to the record that most closely matches the current key.
	HasCurValues (see page 18584)	Indicates whether the CurValue property is available for fields in the packet dataset.
	HasMergeConflicts (see page 18585)	Indicates whether a current record can be merged with updates in a delta packet.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitAltRecBuffers (see page 18585)	Fetches values for the CurValue, OldValue, and NewValue properties of fields.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
V	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	Last (see page 6968)	Moves to the last record in the dataset.
	LoadFromFile (see page 8433)	Loads a client dataset's data from a file.
	LoadFromStream (see page 8435)	Loads a client dataset's data from a stream.
V	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Locate (see page 8435)	Searches the dataset for a specified record and makes that record current.
V	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
V	Lookup (see page 8436)	Retrieves field values from a record that matches specified search values.
	MergeChangeLog (see page 8437)	Merges any data changes in the change log into the client dataset, overwriting existing values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	Open (see page 6976)	Opens the dataset.
V	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
V	Post (see page 8438)	Writes a modified record to the Data (see page 8467) property or the change log.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSEndTransaction (see page 7589)	Ends a transaction that was started by a call to PSStartTransaction (see page 7597).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecute (see page 7590)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSExecuteStatement (see page 7590)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 7591)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 7579)	Gets the SQL command to be executed by PSExecute (see page 7578).
	PSGetCommandText (see page 7591)	Gets the SQL command to be executed by PSExecute (see page 7590).

	PSGetCommandType (see page 7580)	Gets the command type to be executed by PSExecute (see page 7578).
	PSGetCommandType (see page 7591)	Gets the command type to be executed by PSExecute (see page 7590).
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 7592)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 7592)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7593)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetParams (see page 7593)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7594)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7594)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSGetUpdateException (see page 7594)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 7595)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 7595)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 7596)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 7596)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7585)	Assigns a specified SQL command to be executed by PSExecute (see page 7578).
	PSSetCommandText (see page 7596)	Assigns a specified SQL command to be executed by PSExecute (see page 7590).
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 7597)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).
	PSStartTransaction (see page 7597)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7589).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 7598)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Reconcile (see page 8441)	Clears successfully updated records from the client dataset's cache of changes.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RefreshRecord (see page 8442)	Updates the base values of the current record to match the current values of the provider's dataset.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
	RevertRecord (see page 8443)	Undoes edits to the current record.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.

V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 8443)	Saves a client dataset's data to an external file.
	SaveToStream (see page 8445)	Saves a client dataset's data to a stream.
	SetAltRecBuffers (see page 8446)	Sets record buffers when accessing field values.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetKey (see page 8449)	Enables setting of keys and ranges for a dataset prior to a search.
V	SetOptionalParam (see page 8450)	Saves a custom parameter with the data packet.
	SetProvider (see page 8451)	Associates the client dataset with a provider in the same process space.
	SetRange (see page 8452)	Sets the starting and ending values of a range, and applies it.
	SetRangeEnd (see page 8453)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
	SetRangeStart (see page 8454)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
	UndoLastChange (see page 8455)	Undoes the last edit, insert, or delete operation to a record in the client dataset.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateKind (see page 18587)	Indicates the type of update the current record represents.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.
V	UpdateStatus (see page 8456)	Reports the update status for the current record in the dataset.

Public Properties

	Name	Description
	Active (see page 7004)	Specifies whether or not a dataset is open.
	Active (see page 8458)	Specifies whether the client dataset contains data.
R	ActiveAggs (see page 8460)	Lists the maintained aggregates supported by the current index.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AppServer (see page 8462)	Provides access to the interface the client dataset uses to communicate with a provider.
	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ChangeCount (see page 8463)	Indicates the number of changes in the change log.
R	CloneSource (see page 8464)	Indicates the client dataset (if any) with which this client dataset shares data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	ConnectionBroker (see page 8466)	Specifies a connection broker that handles the connection to an application server.
	Data (see page 8467)	Represents the data of the client dataset in a transportable format.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
	DataSetField (see page 8469)	Indicates the TDataSetField component that represents this client dataset.
R	DataSize (see page 8469)	Indicates the number of bytes required to represent the Data (see page 8467) property.
R	DataSource (see page 7014)	Represents the data source of another dataset that supplies values to the dataset.

R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Delta (see page 8470)	Represents a packet of change log data.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
	FieldDefs (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
	FileName (see page 8475)	Specifies a file that stores the data of the client dataset.
	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
R	GroupingLevel (see page 8479)	Indicates the depth of grouping support provided by the current index.
R	HasAppServer (see page 8480)	Indicates whether the client dataset is connected to a provider.
R	IndexFieldCount (see page 8481)	Indicates the number of fields that make up the current index.
	IndexFieldNames (see page 8482)	Lists the fields to use as an index.
	IndexFields (see page 8482)	Specifies the fields associated with the current index.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
	KeyExclusive (see page 8484)	Specifies the upper and lower boundaries for a range.
	KeyFieldCount (see page 8484)	Specifies the number of fields to use when conducting a partial key search on a multi-field key.
R	KeySize (see page 8485)	Indicates the size of the current key used for the dataset.
	LogChanges (see page 8485)	Indicates whether to maintain a change log separate from the data.
	MasterFields (see page 8486)	Names one or more fields in a master table to link with corresponding fields in this dataset in order to establish a master-detail relationship.
R	Modified (see page 7029)	Indicates whether the active record is modified.
R	NewValuesModified (see page 18588)	Indicates whether the NewValue property of any field in the current record has changed.
	ObjectView (see page 7031)	Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	PacketRecords (see page 8488)	Indicates the number or type of records in a single data packet.
	ProviderName (see page 8492)	Specifies the name of a provider component that supplies data to and applies updates from the client dataset..
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
	RemoteServer (see page 8493)	Specifies the connection component that this client dataset uses to connect to an application server.
	SavePoint (see page 8494)	Represents the current state of edits when logging changes.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	StatusFilter (see page 8495)	Filters the client dataset so that it exposes only those records with a specific update status.
	StreamMetaData (see page 18588)	This is StreamMetaData, a member of class TPacketDataSet.
	UseCurValues (see page 18588)	Indicates that the CurValue property of fields in the dataset should be used instead of the OldValue property.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	XMLData (see page 8496)	Represents the data of the client dataset in XML.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

Strict private Fields

	Name	Description
	FCurRecBuf (see page 18581)	This is FCurRecBuf, a member of class TPacketDataSet.
	FCurValues (see page 18581)	This is FCurValues, a member of class TPacketDataSet.
	FNewValuesModified (see page 18581)	This is FNewValuesModified, a member of class TPacketDataSet.
	FOldRecBuf (see page 18581)	This is FOldRecBuf, a member of class TPacketDataSet.
	FUseCurValues (see page 18581)	This is FUseCurValues, a member of class TPacketDataSet.
	FWritingCurValues (see page 18582)	This is FWritingCurValues, a member of class TPacketDataSet.

Strict private Methods

	Name	Description
	GetStreamMetaData (see page 18584)	This is GetStreamMetaData, a member of class TPacketDataSet.
	SetStreamMetaData (see page 18587)	This is SetStreamMetaData, a member of class TPacketDataSet.
	SetWritingCurValues (see page 18587)	This is SetWritingCurValues, a member of class TPacketDataSet.

124.1.10.2 TPacketDataSet.Create Constructor

Creates and initializes instance of TPacketDataSet (see page 18564).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TPacketDataSet(TComponent * AOwner);
```

Description

Most applications do not need to create packet datasets because they are created by the provider components that use them.Create initializes the properties of the packet dataset.

AOwner is the component that is responsible for freeing the TPacketDataSet (see page 18564) instance. If AOwner is nil (Delphi) or NULL (C++), the application must explicitly free the packet data set object.

See Also

Owner (see page 2807)

124.1.10.3 TPacketDataSet Fields

The fields of the TPacketDataSet class are listed here.

Strict private Fields

	Name	Description
	FCurRecBuf (see page 18581)	This is FCurRecBuf, a member of class TPacketDataSet.
	FCurValues (see page 18581)	This is FCurValues, a member of class TPacketDataSet.
	FNewValuesModified (see page 18581)	This is FNewValuesModified, a member of class TPacketDataSet.
	FOldRecBuf (see page 18581)	This is FOldRecBuf, a member of class TPacketDataSet.
	FUseCurValues (see page 18581)	This is FUseCurValues, a member of class TPacketDataSet.
	FWritingCurValues (see page 18582)	This is FWritingCurValues, a member of class TPacketDataSet.

124.1.10.3.1 TPacketDataSet.FCurRecBuf Field

Pascal

```
FCurRecBuf: PChar;
```

C++

```
const char * FCurRecBuf;
```

Description

This is FCurRecBuf, a member of class TPacketDataSet.

124.1.10.3.2 TPacketDataSet.FCurValues Field

Pascal

```
FCurValues: PChar;
```

C++

```
const char * FCurValues;
```

Description

This is FCurValues, a member of class TPacketDataSet.

124.1.10.3.3 TPacketDataSet.FNewValuesModified Field

Pascal

```
FNewValuesModified: Boolean;
```

C++

```
Boolean FNewValuesModified;
```

Description

This is FNewValuesModified, a member of class TPacketDataSet.

124.1.10.3.4 TPacketDataSet.FOldRecBuf Field

Pascal

```
FOldRecBuf: PChar;
```

C++

```
const char * FOldRecBuf;
```

Description

This is FOldRecBuf, a member of class TPacketDataSet.

124.1.10.3.5 TPacketDataSet.FUseCurValues Field

Pascal

```
FUseCurValues: Boolean;
```

C++

```
Boolean FUseCurValues;
```

Description

This is FUseCurValues, a member of class TPacketDataSet.

124.1.10.3.6 TPacketDataSet.FWritingCurValues Field

Pascal

```
FWritingCurValues: Boolean;
```

C++

```
Boolean FWritingCurValues;
```

Description

This is FWritingCurValues, a member of class TPacketDataSet.

124.1.10.4 TPacketDataSet Methods

The methods of the TPacketDataSet class are listed here.

Protected Methods

	Name	Description
	DataEvent (see page 18583)	This is DataEvent, a member of class TPacketDataSet.
	GetStateFieldValue (see page 18584)	This is GetStateFieldValue, a member of class TPacketDataSet.
	InternalClose (see page 18585)	This is InternalClose, a member of class TPacketDataSet.
	InternalInitRecord (see page 18586)	This is InternalInitRecord, a member of class TPacketDataSet.
	InternalOpen (see page 18586)	This is InternalOpen, a member of class TPacketDataSet.
	SetFieldData (see page 18586)	This is SetFieldData, a member of class TPacketDataSet.
	SetStateFieldValue (see page 18586)	This is SetStateFieldValue, a member of class TPacketDataSet.

Public Methods

	Name	Description
	AssignCurValues (see page 18582)	This is the overview for the AssignCurValues method overload.
	CreateFromDelta (see page 18583)	This is CreateFromDelta, a member of class TPacketDataSet.
	HasCurValues (see page 18584)	Indicates whether the CurValue property is available for fields in the packet dataset.
	HasMergeConflicts (see page 18585)	Indicates whether a current record can be merged with updates in a delta packet.
	InitAltRecBuffers (see page 18585)	Fetches values for the CurValue, OldValue, and NewValue properties of fields.
	UpdateKind (see page 18587)	Indicates the type of update the current record represents.

Strict private Methods

	Name	Description
	GetStreamMetaData (see page 18584)	This is GetStreamMetaData, a member of class TPacketDataSet.
	SetStreamMetaData (see page 18587)	This is SetStreamMetaData, a member of class TPacketDataSet.
	SetWritingCurValues (see page 18587)	This is SetWritingCurValues, a member of class TPacketDataSet.

124.1.10.4.1 AssignCurValues Method

This is the overview for the AssignCurValues method overload.

Overload List

	Name	Description
	TPacketDataSet.AssignCurValues (TDataSet) (see page 18583)	Updates the current field value for each field in the packet dataset.
	TPacketDataSet.AssignCurValues (Variant) (see page 18583)	Updates the current field value for each field in the packet dataset.

124.1.10.4.1.1 TPacketDataSet.AssignCurValues Method (TDataSet)

Updates the current field value for each field in the packet dataset.

Pascal

```
procedure AssignCurValues(Source: TDataSet); overload;
procedure AssignCurValues(const CurValues: Variant); overload;
```

C++

```
__fastcall AssignCurValues(TDataSet Source);
__fastcall AssignCurValues(const Variant CurValues);
```

Description

When a resolver component encounters an update error, it calls AssignCurValues to set the CurValue property of fields before triggering the OnUpdateError event handler.

If AssignCurValues is passed a dataset as the Source parameter, then for each field in the packet dataset, the CurValue property is set to the value of the field with the same name in the source dataset.

If AssignCurValues is passed a Variant array as the CurValues parameter, then each entry in the array is used to assign the CurValue property of a field in the packet dataset. CurValues can have one of two forms: an array of field values, or an array of Variant arrays, where each of those arrays has two elements: a field name and a value. In the first form, field values are assigned by position. In the second form, fields are located based on name and the CurValue property assigned accordingly.

See Also

CurValue (see page 7155)

OnUpdateError (see page 18444)

124.1.10.4.2 TPacketDataSet.CreateFromDelta Method

Pascal

```
procedure CreateFromDelta(Source: TPacketDataSet);
```

C++

```
__fastcall CreateFromDelta(TPacketDataSet Source);
```

Description

This is CreateFromDelta, a member of class TPacketDataSet.

124.1.10.4.3 TPacketDataSet.DataEvent Method

Pascal

```
procedure DataEvent(Event: TDataEvent; Info: Longint); override;
```

C++

```
virtual __fastcall DataEvent(TDataEvent Event, Longint Info);
```

Description

This is DataEvent, a member of class TPacketDataSet.

124.1.10.4.4 TPacketDataSet.GetStateFieldValue Method

Pascal

```
function GetStateFieldValue(State: TDataSetState; Field: TField): Variant; override;
```

C++

```
virtual __fastcall Variant GetStateFieldValue(TDataSetState State, TField Field);
```

Description

This is GetStateFieldValue, a member of class TPacketDataSet.

124.1.10.4.5 TPacketDataSet.GetStreamMetaData Method

Pascal

```
function GetStreamMetaData: Boolean;
```

C++

```
__fastcall Boolean GetStreamMetaData();
```

Description

This is GetStreamMetaData, a member of class TPacketDataSet.

124.1.10.4.6 TPacketDataSet.HasCurValues Method

Indicates whether the CurValue property is available for fields in the packet dataset.

Pascal

```
function HasCurValues: Boolean;
```

C++

```
__fastcall Boolean HasCurValues();
```

Description

When an error occurs while applying an updated record, resolver components attempt to locate the current record in the underlying database (or the dataset to which updates are applied.) This current record can then be used when trying to determine the cause of the update error. If the current record can be located, HasCurValues returns true and the fields in the packet dataset have their CurValue properties set from this record. If the record can't be located (for example, if another user changed a key field value or deleted the record), then HasCurValues returns false and the CurValue property of packet dataset fields can't be used.

See Also

UseCurValues (see page 18588)

CurValue (see page 7155)

124.1.10.4.7 TPacketDataSet.HasMergeConflicts Method

Indicates whether a current record can be merged with updates in a delta packet.

Pascal

```
function HasMergeConflicts: Boolean;
```

C++

```
__fastcall Boolean HasMergeConflicts();
```

Description

When resolving update errors in an OnUpdateError event handler, applications can specify that the updated record should be merged automatically with the current field values of the record. (To do this, set the Response parameter to rrMerge.) However, it is not always possible to merge client edits with the current record. If the updated record includes changes to field values that were also changed by other users, the merge will fail and HasMergeConflicts returns true.

See Also

CurValue (see page 7155)

NewValue (see page 7174)

OnUpdateError (see page 18444)

124.1.10.4.8 TPacketDataSet.InitAltRecBuffers Method

Fetches values for the CurValue, OldValue, and NewValue properties of fields.

Pascal

```
procedure InitAltRecBuffers(CheckModified: Boolean = True);
```

C++

```
__fastcall InitAltRecBuffers(Boolean CheckModified = True);
```

Description

While applying updates, InitAltRecBuffers is called automatically to ensure that fields in the packet dataset have values for the CurValue, OldValue, and NewValue properties. The CheckModified parameter specifies whether InitAltRecBuffers should check if the current record has been modified when it sets values. This check allows the packet dataset to reflect appropriate values when applying updates that are changed by an OnUpdateError event handler.

See Also

OldValue (see page 7175)

NewValue (see page 7174)

CurValue (see page 7155)

OnUpdateError (see page 18444)

124.1.10.4.9 TPacketDataSet.InternalClose Method

Pascal

```
procedure InternalClose; override;
```

C++

```
virtual __fastcall InternalClose();
```

Description

This is InternalClose, a member of class TPacketDataSet.

124.1.10.4.10 TPacketDataSet.InternalInitRecord Method

Pascal

```
procedure InternalInitRecord(Buffer: PChar); override;
```

C++

```
virtual __fastcall InternalInitRecord(const char * Buffer);
```

Description

This is InternalInitRecord, a member of class TPacketDataSet.

124.1.10.4.11 TPacketDataSet.InternalOpen Method

Pascal

```
procedure InternalOpen; override;
```

C++

```
virtual __fastcall InternalOpen();
```

Description

This is InternalOpen, a member of class TPacketDataSet.

124.1.10.4.12 TPacketDataSet.SetFieldData Method

Pascal

```
procedure SetFieldData(Field: TField; Buffer: Pointer); override;
```

C++

```
virtual __fastcall SetFieldData(TField Field, void * Buffer);
```

Description

This is SetFieldData, a member of class TPacketDataSet.

124.1.10.4.13 TPacketDataSet.SetStateFieldValue Method

Pascal

```
procedure SetStateFieldValue(State: TDataSetState; Field: TField; const Value: Variant);
override;
```

C++

```
virtual __fastcall SetStateFieldValue(TDataSetState State, TField Field, const Variant Value);
```

Description

This is SetStateFieldValue, a member of class TPacketDataSet.

124.1.10.4.14 TPacketDataSet.SetStreamMetaData Method

Pascal

```
procedure SetStreamMetaData(Value: Boolean);
```

C++

```
__fastcall SetStreamMetaData(Boolean Value);
```

Description

This is SetStreamMetaData, a member of class TPacketDataSet.

124.1.10.4.15 TPacketDataSet.SetWritingCurValues Method

Pascal

```
procedure SetWritingCurValues(const Value: Boolean);
```

C++

```
__fastcall SetWritingCurValues(const Boolean Value);
```

Description

This is SetWritingCurValues, a member of class TPacketDataSet.

124.1.10.4.16 TPacketDataSet.UpdateKind Method

Indicates the type of update the current record represents.

Pascal

```
function UpdateKind: TUpdateKind;
```

C++

```
__fastcall TUpdateKind UpdateKind();
```

Description

Call UpdateKind while applying updates to determine the type of update the current record represents. UpdateKind returns

ukInsert if the current record is a newly inserted record.

ukDelete if the current record represents a record that should be deleted.

ukModify if the current record represents updates to an existing record.

124.1.10.5 TPacketDataSet Properties

The properties of the TPacketDataSet class are listed here.

Protected Properties

	Name	Description
	WritingCurValues (see page 18589)	This is WritingCurValues, a member of class TPacketDataSet.

Public Properties

	Name	Description
R	NewValuesModified (see page 18588)	Indicates whether the NewValue property of any field in the current record has changed.

	StreamMetaData (see page 18588)	This is StreamMetaData, a member of class TPacketDataSet.
	UseCurValues (see page 18588)	Indicates that the CurValue property of fields in the dataset should be used instead of the OldValue property.

124.1.10.5.1 TPacketDataSet.NewValuesModified Property

Indicates whether the NewValue property of any field in the current record has changed.

Pascal

```
property NewValuesModified: Boolean;
```

C++

```
__property Boolean NewValuesModified;
```

Description

Read NewValuesModified to determine whether any fields in the current record were edited during the resolving process.

See Also

NewValue (see page 7174)

124.1.10.5.2 TPacketDataSet.StreamMetaData Property

Pascal

```
property StreamMetaData: Boolean;
```

C++

```
__property Boolean StreamMetaData;
```

Description

This is StreamMetaData, a member of class TPacketDataSet.

124.1.10.5.3 TPacketDataSet.UseCurValues Property

Indicates that the CurValue property of fields in the dataset should be used instead of the OldValue property.

Pascal

```
property UseCurValues: Boolean;
```

C++

```
__property Boolean UseCurValues;
```

Description

UseCurValues is set during the resolving process when a record is corrected in an OnUpdateError event handler. This allows the resolver to reapply a corrected record rather than using the originally provided values as the basis of an update.

See Also

HasCurValues (see page 18584)

CurValue (see page 7155)

OldValue (see page 7175)

OnUpdateError (see page 18444)

124.1.10.5.4 TPacketDataSet.WritingCurValues Property

Pascal

```
property WritingCurValues: Boolean;
```

C++

```
__property Boolean WritingCurValues;
```

Description

This is WritingCurValues, a member of class TPacketDataSet.

124.1.11 TProvider Class

TProvider provides data to and applies updates from a client dataset.

Class Hierarchy

Pascal

```
TProvider = class(TDataSetProvider);
```

C++

```
class TProvider : public TDataSetProvider;
```

File

Provider

Description

TProvider is provided for backward compatibility. It is identical to TDataSetProvider (see page 18508). New applications should use TDataSetProvider (see page 18508) instead.

See Also

TDataSetProvider (see page 18508)

124.1.12 TSQLResolver Class

TSQLResolver applies updates to a database server on behalf of a provider.

Class Hierarchy

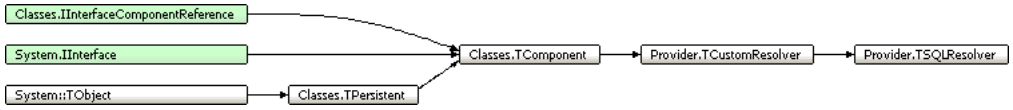

Pascal

```
TSQLResolver = class(TCustomResolver);
```

C++

```
class TSQLResolver : public TCustomResolver;
```

File

Provider

Description

Provider (see page 18600) components use TSQLResolver to apply updates from a client application directly to a database server when their ResolveToDataSet property is false. Otherwise, they use TDataSetResolver (see page 18544).

Do not create instances of TSQLResolver. Provider (see page 18600) components create their own TSQLResolver objects for applying updates.

See Also

TBaseProvider (see page 18423)

TDataSetResolver (see page 18544)

TDataSetProvider (see page 18508)

124.1.12.1 TSQLResolver Members

The following tables list the members exposed by TSQLResolver.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	BeginUpdate (see page 18490)	Provides the interface for a method that performs any initialization before applying updates.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoDelete (see page 18595)	Defines a method that deletes a record.
V	DoExecSQL (see page 18596)	This is DoExecSQL, a member of class TSQLResolver.
V	DoGetValues (see page 18596)	This is DoGetValues, a member of class TSQLResolver.
V	DoInsert (see page 18596)	Defines a method that inserts a record.
V	DoUpdate (see page 18596)	Defines a method that modifies a record.
V	EndUpdate (see page 18492)	Provides the interface for a method that performs any finalization after applying updates.
V	GenDeleteSQL (see page 18597)	This is GenDeleteSQL, a member of class TSQLResolver.
V	GenInsertSQL (see page 18597)	This is GenInsertSQL, a member of class TSQLResolver.
V	GenSelectSQL (see page 18598)	This is GenSelectSQL, a member of class TSQLResolver.
V	GenUpdateSQL (see page 18598)	This is GenUpdateSQL, a member of class TSQLResolver.
V	GenWhereSQL (see page 18598)	This is GenWhereSQL, a member of class TSQLResolver.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.

	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetProvider (see page 18598)	This is GetProvider, a member of class TSQLResolver.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleUpdateError (see page 18493)	Manages the resolving process when errors occur while applying updates.
V	InitializeConflictBuffer (see page 18599)	Introduces a method for adding information about a failed update that is passed to an OnUpdateError event handler.
V	InternalDoUpdate (see page 18599)	This is InternalDoUpdate, a member of class TSQLResolver.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	LogUpdateError (see page 18495)	This is LogUpdateError, a member of class TCustomResolver.
V	LogUpdateRecord (see page 18495)	Adds records to the error log if their data has been changed by another application.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
V	UseFieldInUpdate (see page 18599)	This is UseFieldInUpdate, a member of class TSQLResolver.
V	UseFieldInWhere (see page 18600)	This is UseFieldInWhere, a member of class TSQLResolver.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Provider (see page 18600)	Indicates the provider that uses this resolver to apply updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	ApplyUpdates (see page 18490)	Applies updates received from a provider.

	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 18593)	Creates and initializes an SQL resolver.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 18594)	Destroys an instance of TSQLResolver (see page 18589).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	FreeTreeData (see page 18597)	This is FreeTreeData, a member of class TSQLResolver.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InitKeyFields (see page 18494)	Sets the provider flag pfInKey on all key fields in a delta data packet.
	InitTreeData (see page 18599)	This is InitTreeData, a member of class TSQLResolver.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalBeforeResolve (see page 18494)	Initializes internal data structures once the delta packet has been wrapped in an update tree but before any updates are applied.
	InternalUpdateRecord (see page 18495)	Manages the updating process once a delta packet is wrapped in a TUpdateTree (see page 18600) object.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	RowRequest (see page 18496)	Returns information from a specified record.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	UpdateTree (see page 18497)	This is UpdateTree, a member of class TCustomResolver.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

Strict private Fields

	Name	Description
	FErrorCount (see page 18488)	This is FErrorCount, a member of class TCustomResolver.
	FMaxErrors (see page 18488)	This is FMaxErrors, a member of class TCustomResolver.
	FParams (see page 18594)	This is FParams, a member of class TSQLResolver.
	FPrevResponse (see page 18488)	This is FPrevResponse, a member of class TCustomResolver.
	FProvider (see page 18489)	This is FProvider, a member of class TCustomResolver.
	FSQL (see page 18594)	This is FSQL, a member of class TSQLResolver.
	FUpdateTree (see page 18489)	This is FUpdateTree, a member of class TCustomResolver.

124.1.12.2 TSQLResolver.Create Constructor

Creates and initializes an SQL resolver.

Pascal

```
constructor Create(AProvider: TDataSetProvider); reintroduce;
```

C++

```
__fastcall TSQLResolver(TDataSetProvider AProvider);
```

Description

Provider (see page 18600) components call Create when they need a resolver component for applying updates to a database server. Do not create resolver instances directly. Resolvers are designed to be created and used by providers, and can not function independently of a provider.

The AProvider parameter specifies the provider that creates and uses the resolver.

See Also

Resolver (see page 18441)

124.1.12.3 TSQLResolver.Destroy Destructor

Destroys an instance of TSQLResolver (see page 18589).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSQLResolver();
```

Description

Most applications should not delete instances of TSQLResolver (see page 18589). SQL resolvers are freed automatically by the provider components that use them.

See Also

Free (see page 22300)

Resolver (see page 18441)

124.1.12.4 TSQLResolver Fields

The fields of the TSQLResolver class are listed here.

Strict private Fields

	Name	Description
	FParams (see page 18594)	This is FParams, a member of class TSQLResolver.
	FSQL (see page 18594)	This is FSQL, a member of class TSQLResolver.

124.1.12.4.1 TSQLResolver.FParams Field

Pascal

```
FParams: TParams;
```

C++

```
TParams FParams;
```

Description

This is FParams, a member of class TSQLResolver.

124.1.12.4.2 TSQLResolver.FSQL Field

Pascal

```
FSQL: TWideStringList;
```

C++

```
TWideStringList FSQL;
```

Description

This is FSQL, a member of class TSQLResolver.

124.1.12.5 TSQLResolver Methods

The methods of the TSQLResolver class are listed here.

Protected Methods

	Name	Description
	DoDelete (see page 18595)	Defines a method that deletes a record.
	DoExecSQL (see page 18596)	This is DoExecSQL, a member of class TSQLResolver.
	DoGetValues (see page 18596)	This is DoGetValues, a member of class TSQLResolver.
	DoInsert (see page 18596)	Defines a method that inserts a record.
	DoUpdate (see page 18596)	Defines a method that modifies a record.
	GenDeleteSQL (see page 18597)	This is GenDeleteSQL, a member of class TSQLResolver.
	GenInsertSQL (see page 18597)	This is GenInsertSQL, a member of class TSQLResolver.
	GenSelectSQL (see page 18598)	This is GenSelectSQL, a member of class TSQLResolver.
	GenUpdateSQL (see page 18598)	This is GenUpdateSQL, a member of class TSQLResolver.
	GenWhereSQL (see page 18598)	This is GenWhereSQL, a member of class TSQLResolver.
	GetProvider (see page 18598)	This is GetProvider, a member of class TSQLResolver.
	InitializeConflictBuffer (see page 18599)	Introduces a method for adding information about a failed update that is passed to an OnUpdateError event handler.
	InternalDoUpdate (see page 18599)	This is InternalDoUpdate, a member of class TSQLResolver.
	UseFieldInUpdate (see page 18599)	This is UseFieldInUpdate, a member of class TSQLResolver.
	UseFieldInWhere (see page 18600)	This is UseFieldInWhere, a member of class TSQLResolver.

Public Methods

	Name	Description
	FreeTreeData (see page 18597)	This is FreeTreeData, a member of class TSQLResolver.
	InitTreeData (see page 18599)	This is InitTreeData, a member of class TSQLResolver.

124.1.12.5.1 TSQLResolver.DoDelete Method

Defines a method that deletes a record.

Pascal

```
procedure DoDelete(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall DoDelete(TUpdateTree Tree);
```

Description

DoDelete is one of several helper methods internal to the ApplyUpdates method. These helper methods perform the actual updates in the delta packet. ApplyUpdates wraps the delta packet in a TUpdateTree object. For each record that must be deleted, the resolver calls DoDelete.

TCustomResolver does not implement the DoDelete method. Descendants override this method to delete records in their own specialized way.

See Also

ApplyUpdates (see page 18490)

DoInsert (see page 18491)

DoUpdate (see page 18491)

124.1.12.5.2 TSQLResolver.DoExecSQL Method

Pascal

```
procedure DoExecSQL(SQL: TWideStringList; Params: TParams); virtual;
```

C++

```
virtual __fastcall DoExecSQL(TWideStringList SQL, TParams Params);
```

Description

This is DoExecSQL, a member of class TSQLResolver.

124.1.12.5.3 TSQLResolver.DoGetValues Method

Pascal

```
procedure DoGetValues(SQL: TWideStringList; Params: TParams; DataSet: TDataSet); virtual;
```

C++

```
virtual __fastcall DoGetValues(TWideStringList SQL, TParams Params, TDataSet DataSet);
```

Description

This is DoGetValues, a member of class TSQLResolver.

124.1.12.5.4 TSQLResolver.DoInsert Method

Defines a method that inserts a record.

Pascal

```
procedure DoInsert(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall DoInsert(TUpdateTree Tree);
```

Description

DoInsert is one of several helper methods internal to the ApplyUpdates method. These helper methods perform the actual updates in the delta packet. ApplyUpdates wraps the delta packet in a TUpdateTree object. For each record that must be inserted, the resolver calls DoInsert.

TCustomResolver does not implement the DoInsert method. Descendants override this method to insert records in their own specialized way.

See Also

ApplyUpdates (see page 18490)

DoDelete (see page 18491)

DoUpdate (see page 18491)

124.1.12.5.5 TSQLResolver.DoUpdate Method

Defines a method that modifies a record.

Pascal

```
procedure DoUpdate(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall DoUpdate(TUpdateTree Tree);
```

Description

DoUpdate is one of several helper methods internal to the ApplyUpdates method. These helper methods perform the actual updates passed in the delta packet. ApplyUpdates wraps the delta packet in a TUpdateTree object. For each record that must be modified, the resolver calls DoUpdate.

TCustomResolver does not implement the DoUpdate method. Descendants override this method to insert records in their own specialized way.

See Also

ApplyUpdates (see page 18490)

DoDelete (see page 18491)

DoInsert (see page 18491)

124.1.12.5.6 TSQLResolver.FreeTreeData Method

Pascal

```
procedure FreeTreeData(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall FreeTreeData(TUpdateTree Tree);
```

Description

This is FreeTreeData, a member of class TSQLResolver.

124.1.12.5.7 TSQLResolver.GenDeleteSQL Method

Pascal

```
procedure GenDeleteSQL(Tree: TUpdateTree; SQL: TWideStrings; Params: TParams; Alias: string);
virtual;
```

C++

```
virtual __fastcall GenDeleteSQL(TUpdateTree Tree, TWideStrings SQL, TParams Params, AnsiString
Alias);
```

Description

This is GenDeleteSQL, a member of class TSQLResolver.

124.1.12.5.8 TSQLResolver.GenInsertSQL Method

Pascal

```
procedure GenInsertSQL(Tree: TUpdateTree; SQL: TWideStrings; Params: TParams); virtual;
```

C++

```
virtual __fastcall GenInsertSQL(TUpdateTree Tree, TWideStrings SQL, TParams Params);
```

Description

This is GenInsertSQL, a member of class TSQLResolver.

124.1.12.5.9 TSQLResolver.GenSelectSQL Method

Pascal

```
procedure GenSelectSQL(Tree: TUpdateTree; SQL: TWideStrings; Params: TParams; Alias: string;
Mode: TUpdateMode = upWhereKeyOnly); virtual;
```

C++

```
virtual __fastcall GenSelectSQL(TUpdateTree Tree, TWideStrings SQL, TParams Params, AnsiString
Alias, TUpdateMode Mode = upWhereKeyOnly);
```

Description

This is GenSelectSQL, a member of class TSQLResolver.

124.1.12.5.10 TSQLResolver.GenUpdateSQL Method

Pascal

```
procedure GenUpdateSQL(Tree: TUpdateTree; SQL: TWideStrings; Params: TParams; Alias: string);
virtual;
```

C++

```
virtual __fastcall GenUpdateSQL(TUpdateTree Tree, TWideStrings SQL, TParams Params, AnsiString
Alias);
```

Description

This is GenUpdateSQL, a member of class TSQLResolver.

124.1.12.5.11 TSQLResolver.GenWhereSQL Method

Pascal

```
procedure GenWhereSQL(Tree: TUpdateTree; SQL: TWideStrings; Params: TParams; GenUpdateMode:
TUpdateMode; Alias: string); virtual;
```

C++

```
virtual __fastcall GenWhereSQL(TUpdateTree Tree, TWideStrings SQL, TParams Params, TUpdateMode
GenUpdateMode, AnsiString Alias);
```

Description

This is GenWhereSQL, a member of class TSQLResolver.

124.1.12.5.12 TSQLResolver.GetProvider Method

Pascal

```
function GetProvider: TDataSetProvider; virtual;
```

C++

```
virtual __fastcall TDataSetProvider GetProvider();
```

Description

This is GetProvider, a member of class TSQLResolver.

124.1.12.5.13 TSQLResolver.InitializeConflictBuffer Method

Introduces a method for adding information about a failed update that is passed to an OnUpdateError event handler.

Pascal

```
procedure InitializeConflictBuffer(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall InitializeConflictBuffer(TUpdateTree Tree);
```

Description

The HandleUpdateError method calls InitializeConflictBuffer before generating an OnUpdateError event. This ensures that the client dataset passed to an OnUpdateError event handler represents the update that caused the problem, with appropriate values for field's CurValue property.

Descendants of TCustomResolver implement this method to obtain information from the dataset or server to which they are applying updates.

See Also

HandleUpdateError (see page 18493)

OnUpdateError (see page 18444)

124.1.12.5.14 TSQLResolver.InitTreeData Method

Pascal

```
procedure InitTreeData(Tree: TUpdateTree); override;
```

C++

```
virtual __fastcall InitTreeData(TUpdateTree Tree);
```

Description

This is InitTreeData, a member of class TSQLResolver.

124.1.12.5.15 TSQLResolver.InternalDoUpdate Method

Pascal

```
procedure InternalDoUpdate(Tree: TUpdateTree; UpdateKind: TUpdateKind); virtual;
```

C++

```
virtual __fastcall InternalDoUpdate(TUpdateTree Tree, TUpdateKind UpdateKind);
```

Description

This is InternalDoUpdate, a member of class TSQLResolver.

124.1.12.5.16 TSQLResolver.UseFieldInUpdate Method

Pascal

```
function UseFieldInUpdate(Field: TField): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean UseFieldInUpdate(TField Field);
```

Description

This is UseFieldInUpdate, a member of class TSQLResolver.

124.1.12.5.17 TSQLResolver.UseFieldInWhere Method

Pascal

```
function UseFieldInWhere(Field: TField; Mode: TUpdateMode): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean UseFieldInWhere(TField Field, TUpdateMode Mode);
```

Description

This is UseFieldInWhere, a member of class TSQLResolver.

124.1.12.6 TSQLResolver Properties

The properties of the TSQLResolver class are listed here.

Protected Properties

	Name	Description
	Provider (see page 18600)	Indicates the provider that uses this resolver to apply updates.

124.1.12.6.1 TSQLResolver.Provider Property

Indicates the provider that uses this resolver to apply updates.

Pascal

```
property Provider: TDataSetProvider;
```

C++

```
__property TDataSetProvider Provider;
```

Description

Provider is a read-only property that points to the provider that uses this resolver for applying updates. The Provider property is set in the constructor.

Resolvers use the Provider property internally to obtain information supplied in the provider's properties. They also use it to generate events on the provider at appropriate points in the update process.

See Also

Create (see page 18487)

Resolver (see page 18441)

124.1.13 TUpdateTree Class

TUpdateTree stores information about a delta packet on behalf of a resolver.

Class Hierarchy

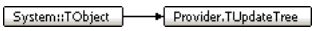

Pascal

```
TUpdateTree = class(TObject);
```

C++

```
class TUpdateTree : public TObject;
```

File

Provider

Description

When a client dataset applies its updates to the source dataset or database server, the provider that represents that dataset or server passes the updates to a resolver component. The resolver takes this delta data packet and builds a TUpdateTree to store the updates and information about them in a convenient form. In particular, the TUpdateTree structure allows a resolver to handle master/detail relationships (represented as nested dataset fields in the delta packet) so that updates are applied in the proper order.

Applications can't use TUpdateTree directly. Most of the structured information about the delta packet is represented in protected properties that are only available to the resolver classes in the Provider (see page 18419) unit.

See Also

TCustomResolver (see page 18483)

TCustomClientDataSet (see page 8373)

TCustomProvider (see page 18449)

124.1.13.1 TUpdateTree Members

The following tables list the members exposed by TUpdateTree.

Protected Properties

	Name	Description
R	DetailList (see page 18610)	This is DetailList, a member of class TUpdateTree.
	Opened (see page 18612)	This is Opened, a member of class TUpdateTree.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 18606)	This is Clear, a member of class TUpdateTree.
	Create (see page 18603)	Creates and initializes an instance of TUpdateTree (see page 18600).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 18603)	Destroys an instance of TUpdateTree (see page 18600).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	DoUpdates (see page 18606)	This is DoUpdates, a member of class TUpdateTree.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitData (see page 18608)	This is InitData, a member of class TUpdateTree.
V	InitDelta (see page 18608)	This is the overview for the InitDelta method overload.
V	InitErrorPacket (see page 18609)	This is InitErrorPacket, a member of class TUpdateTree.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	RefreshData (see page 18609)	This is RefreshData, a member of class TUpdateTree.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Data (see page 18610)	This is Data, a member of class TUpdateTree.
	Delta (see page 18610)	This is Delta, a member of class TUpdateTree.
R	DetailCount (see page 18610)	This is DetailCount, a member of class TUpdateTree.
R	Details (see page 18610)	This is Details, a member of class TUpdateTree.
	ErrorDS (see page 18611)	This is ErrorDS, a member of class TUpdateTree.
R	HasErrors (see page 18611)	This is HasErrors, a member of class TUpdateTree.
R	IsNested (see page 18611)	This is IsNested, a member of class TUpdateTree.
	Name (see page 18611)	This is Name, a member of class TUpdateTree.
R	Parent (see page 18612)	This is Parent, a member of class TUpdateTree.
R	Resolver (see page 18612)	This is Resolver, a member of class TUpdateTree.
	Source (see page 18612)	This is Source, a member of class TUpdateTree.

Strict private Fields

	Name	Description
	FData (see page 18604)	This is FData, a member of class TUpdateTree.
	FDeltaDS (see page 18604)	This is FDeltaDS, a member of class TUpdateTree.
	FDetails (see page 18604)	This is FDetails, a member of class TUpdateTree.
	FErrorDS (see page 18604)	This is FErrorDS, a member of class TUpdateTree.
	FName (see page 18604)	This is FName, a member of class TUpdateTree.
	FOpened (see page 18605)	This is FOpened, a member of class TUpdateTree.
	FParent (see page 18605)	This is FParent, a member of class TUpdateTree.
	FResolver (see page 18605)	This is FResolver, a member of class TUpdateTree.
	FSourceDS (see page 18605)	This is FSourceDS, a member of class TUpdateTree.

Strict private Methods

	Name	Description
	GetDetail (see page 18606)	This is GetDetail, a member of class TUpdateTree.
	GetDetailCount (see page 18607)	This is GetDetailCount, a member of class TUpdateTree.
	GetErrorDS (see page 18607)	This is GetErrorDS, a member of class TUpdateTree.
	GetHasErrors (see page 18607)	This is GetHasErrors, a member of class TUpdateTree.
	GetIsNested (see page 18607)	This is GetIsNested, a member of class TUpdateTree.
	GetTree (see page 18608)	This is GetTree, a member of class TUpdateTree.

124.1.13.2 TUpdateTree.Create Constructor

Creates and initializes an instance of TUpdateTree (see page 18600).

Pascal

```
constructor Create(AParent: TUpdateTree; AResolver: TCustomResolver);
```

C++

```
__fastcall TUpdateTree(TUpdateTree AParent, TCustomResolver AResolver);
```

Description

Applications do not need to call the TUpdateTree (see page 18600) constructor. Update trees are created automatically by the resolver components that use them.

The AParent parameter specifies the TUpdateTree (see page 18600) for the updates to a master table if this update tree represents a detail data set. If the TUpdateTree (see page 18600) does not represent the updates to a detail dataset, AParent is nil (Delphi) or NULL (C++).

The AResolver parameter specifies the resolver component that uses this update tree when applying updates. This parameter must be specified because the update tree calls some of this resolver's methods.

124.1.13.3 TUpdateTree.Destroy Destructor

Destroys an instance of TUpdateTree (see page 18600).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TUpdateTree();
```

Description

Applications do not need to free TUpdateTree (see page 18600) instances. Update trees are freed by the resolver components that use them.

See Also

Free (see page 22300)

124.1.13.4 TUpdateTree Fields

The fields of the TUpdateTree class are listed here.

Strict private Fields

	Name	Description
	FData (see page 18604)	This is FData, a member of class TUpdateTree.
	FDeltaDS (see page 18604)	This is FDeltaDS, a member of class TUpdateTree.
	FDetails (see page 18604)	This is FDetails, a member of class TUpdateTree.
	FErrorDS (see page 18604)	This is FErrorDS, a member of class TUpdateTree.
	FName (see page 18604)	This is FName, a member of class TUpdateTree.
	FOpened (see page 18605)	This is FOpened, a member of class TUpdateTree.
	FParent (see page 18605)	This is FParent, a member of class TUpdateTree.

	FResolver (see page 18605)	This is FResolver, a member of class TUpdateTree.
	FSourceDS (see page 18605)	This is FSourceDS, a member of class TUpdateTree.

124.1.13.4.1 TUpdateTree.FData Field

Pascal

```
FData: Pointer;
```

C++

```
void * FData;
```

Description

This is FData, a member of class TUpdateTree.

124.1.13.4.2 TUpdateTree.FDeltaDS Field

Pascal

```
FDeltaDS: TPacketDataSet;
```

C++

```
TPacketDataSet FDeltaDS;
```

Description

This is FDeltaDS, a member of class TUpdateTree.

124.1.13.4.3 TUpdateTree.FDetails Field

Pascal

```
FDetails: TList;
```

C++

```
TList FDetails;
```

Description

This is FDetails, a member of class TUpdateTree.

124.1.13.4.4 TUpdateTree.FErrorDS Field

Pascal

```
FErrorDS: TPacketDataSet;
```

C++

```
TPacketDataSet FErrorDS;
```

Description

This is FErrorDS, a member of class TUpdateTree.

124.1.13.4.5 TUpdateTree.FName Field

Pascal

```
FName: WideString;
```

C++

```
BSTR FName;
```

Description

This is FName, a member of class TUpdateTree.

124.1.13.4.6 TUpdateTree.FOpened Field

Pascal

```
FOpened: Boolean;
```

C++

```
Boolean FOpened;
```

Description

This is FOpened, a member of class TUpdateTree.

124.1.13.4.7 TUpdateTree.FParent Field

Pascal

```
FParent: TUpdateTree;
```

C++

```
TUpdateTree FParent;
```

Description

This is FParent, a member of class TUpdateTree.

124.1.13.4.8 TUpdateTree.FResolver Field

Pascal

```
FResolver: TCustomResolver;
```

C++

```
TCustomResolver FResolver;
```

Description

This is FResolver, a member of class TUpdateTree.

124.1.13.4.9 TUpdateTree.FSourceDS Field

Pascal

```
FSourceDS: TDataSet;
```

C++

```
TDataSet FSourceDS;
```

Description

This is FSourceDS, a member of class TUpdateTree.

124.1.13.5 TUpdateTree Methods

The methods of the TUpdateTree class are listed here.

Public Methods

	Name	Description
	Clear (see page 18606)	This is Clear, a member of class TUpdateTree.
	DoUpdates (see page 18606)	This is DoUpdates, a member of class TUpdateTree.
	InitData (see page 18608)	This is InitData, a member of class TUpdateTree.
	InitDelta (see page 18608)	This is the overview for the InitDelta method overload.
	InitErrorPacket (see page 18609)	This is InitErrorPacket, a member of class TUpdateTree.
	RefreshData (see page 18609)	This is RefreshData, a member of class TUpdateTree.

Strict private Methods

	Name	Description
	GetDetail (see page 18606)	This is GetDetail, a member of class TUpdateTree.
	GetDetailCount (see page 18607)	This is GetDetailCount, a member of class TUpdateTree.
	GetErrorDS (see page 18607)	This is GetErrorDS, a member of class TUpdateTree.
	GetHasErrors (see page 18607)	This is GetHasErrors, a member of class TUpdateTree.
	GetIsNested (see page 18607)	This is GetIsNested, a member of class TUpdateTree.
	GetTree (see page 18608)	This is GetTree, a member of class TUpdateTree.

124.1.13.5.1 TUpdateTree.Clear Method

Pascal

```
procedure Clear; virtual;
```

C++

```
virtual __fastcall Clear();
```

Description

This is Clear, a member of class TUpdateTree.

124.1.13.5.2 TUpdateTree.DoUpdates Method

Pascal

```
function DoUpdates: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoUpdates();
```

Description

This is DoUpdates, a member of class TUpdateTree.

124.1.13.5.3 TUpdateTree.GetDetail Method

Pascal

```
function GetDetail(Index: Integer): TUpdateTree;
```

C++

```
__fastcall TUpdateTree GetDetail(int Index);
```

Description

This is GetDetail, a member of class TUpdateTree.

124.1.13.5.4 TUpdateTree.GetDetailCount Method

Pascal

```
function GetDetailCount: Integer;
```

C++

```
__fastcall int GetDetailCount();
```

Description

This is GetDetailCount, a member of class TUpdateTree.

124.1.13.5.5 TUpdateTree.GetErrorDS Method

Pascal

```
function GetErrorDS: TPacketDataSet;
```

C++

```
__fastcall TPacketDataSet GetErrorDS();
```

Description

This is GetErrorDS, a member of class TUpdateTree.

124.1.13.5.6 TUpdateTree.GetHasErrors Method

Pascal

```
function GetHasErrors: Boolean;
```

C++

```
__fastcall Boolean GetHasErrors();
```

Description

This is GetHasErrors, a member of class TUpdateTree.

124.1.13.5.7 TUpdateTree.GetIsNested Method

Pascal

```
function GetIsNested: Boolean;
```

C++

```
__fastcall Boolean GetIsNested();
```

Description

This is GetIsNested, a member of class TUpdateTree.

124.1.13.5.8 TUpdateTree.GetTree Method

Pascal

```
function GetTree(const AName: WideString): TUpdateTree;
```

C++

```
__fastcall TUpdateTree GetTree(const BSTR AName);
```

Description

This is GetTree, a member of class TUpdateTree.

124.1.13.5.9 TUpdateTree.InitData Method

Pascal

```
procedure InitData(ASource: TDataSet); virtual;
```

C++

```
virtual __fastcall InitData(TDataSet ASource);
```

Description

This is InitData, a member of class TUpdateTree.

124.1.13.5.10 InitDelta Method

This is the overview for the InitDelta method overload.

Overload List

	Name	Description
	TUpdateTree.InitDelta (OleVariant) (see page 18608)	This is InitDelta, a member of class TUpdateTree.
	TUpdateTree.InitDelta (TPacketDataSet) (see page 18608)	This is InitDelta, a member of class TUpdateTree.

124.1.13.5.10.1 TUpdateTree.InitDelta Method (OleVariant)

Pascal

```
procedure InitDelta(const ADelta: OleVariant); virtual; overload;
```

C++

```
virtual __fastcall InitDelta(const OleVariant ADelta);
```

Description

This is InitDelta, a member of class TUpdateTree.

124.1.13.5.10.2 TUpdateTree.InitDelta Method (TPacketDataSet)

Pascal

```
procedure InitDelta(ADelta: TPacketDataSet); virtual; overload;
```

C++

```
virtual __fastcall InitDelta(TPacketDataSet ADelta);
```

Description

This is InitDelta, a member of class TUpdateTree.

124.1.13.5.11 TUpdateTree.InitErrorPacket Method

Pascal

```
procedure InitErrorPacket(E: EUpdateError; Response: TResolverResponse); virtual;
```

C++

```
virtual __fastcall InitErrorPacket(EUpdateError E, TResolverResponse Response);
```

Description

This is InitErrorPacket, a member of class TUpdateTree.

124.1.13.5.12 TUpdateTree.RefreshData Method

Pascal

```
procedure RefreshData(Options: TFetchOptions); virtual;
```

C++

```
virtual __fastcall RefreshData(TFetchOptions Options);
```

Description

This is RefreshData, a member of class TUpdateTree.

124.1.13.6 TUpdateTree Properties

The properties of the TUpdateTree class are listed here.

Protected Properties

	Name	Description
R	DetailList (see page 18610)	This is DetailList, a member of class TUpdateTree.
	Opened (see page 18612)	This is Opened, a member of class TUpdateTree.

Public Properties

	Name	Description
	Data (see page 18610)	This is Data, a member of class TUpdateTree.
	Delta (see page 18610)	This is Delta, a member of class TUpdateTree.
R	DetailCount (see page 18610)	This is DetailCount, a member of class TUpdateTree.
R	Details (see page 18610)	This is Details, a member of class TUpdateTree.
	ErrorDS (see page 18611)	This is ErrorDS, a member of class TUpdateTree.
R	HasErrors (see page 18611)	This is HasErrors, a member of class TUpdateTree.
R	IsNested (see page 18611)	This is IsNested, a member of class TUpdateTree.
	Name (see page 18611)	This is Name, a member of class TUpdateTree.
R	Parent (see page 18612)	This is Parent, a member of class TUpdateTree.
R	Resolver (see page 18612)	This is Resolver, a member of class TUpdateTree.
	Source (see page 18612)	This is Source, a member of class TUpdateTree.

124.1.13.6.1 TUpdateTree.Data Property

Pascal

```
property Data: Pointer;
```

C++

```
__property void * Data;
```

Description

This is Data, a member of class TUpdateTree.

124.1.13.6.2 TUpdateTree.Delta Property

Pascal

```
property Delta: TPacketDataSet;
```

C++

```
__property TPacketDataSet Delta;
```

Description

This is Delta, a member of class TUpdateTree.

124.1.13.6.3 TUpdateTree.DetailCount Property

Pascal

```
property DetailCount: Integer;
```

C++

```
__property int DetailCount;
```

Description

This is DetailCount, a member of class TUpdateTree.

124.1.13.6.4 TUpdateTree.DetailList Property

Pascal

```
property DetailList: TList;
```

C++

```
__property TList DetailList;
```

Description

This is DetailList, a member of class TUpdateTree.

124.1.13.6.5 TUpdateTree.Details Property

Pascal

```
property Details [Index: Integer]: TUpdateTree;
```

C++

```
__property TUpdateTree Details[int Index];
```

Description

This is Details, a member of class TUpdateTree.

124.1.13.6.6 TUpdateTree.ErrorDS Property

Pascal

```
property ErrorDS: TPacketDataSet;
```

C++

```
__property TPacketDataSet ErrorDS;
```

Description

This is ErrorDS, a member of class TUpdateTree.

124.1.13.6.7 TUpdateTree.HasErrors Property

Pascal

```
property HasErrors: Boolean;
```

C++

```
__property Boolean HasErrors;
```

Description

This is HasErrors, a member of class TUpdateTree.

124.1.13.6.8 TUpdateTree.IsNested Property

Pascal

```
property IsNested: Boolean;
```

C++

```
__property Boolean IsNested;
```

Description

This is IsNested, a member of class TUpdateTree.

124.1.13.6.9 TUpdateTree.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class TUpdateTree.

124.1.13.6.10 TUpdateTree.Opened Property

Pascal

```
property Opened: Boolean;
```

C++

```
__property Boolean Opened;
```

Description

This is Opened, a member of class TUpdateTree.

124.1.13.6.11 TUpdateTree.Parent Property

Pascal

```
property Parent: TUpdateTree;
```

C++

```
__property TUpdateTree Parent;
```

Description

This is Parent, a member of class TUpdateTree.

124.1.13.6.12 TUpdateTree.Resolver Property

Pascal

```
property Resolver: TCustomResolver;
```

C++

```
__property TCustomResolver Resolver;
```

Description

This is Resolver, a member of class TUpdateTree.

124.1.13.6.13 TUpdateTree.Source Property

Pascal

```
property Source: TDataSet;
```

C++

```
__property TDataSet Source;
```

Description

This is Source, a member of class TUpdateTree.

124.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IProviderContainer (see page 18613)	This is class Provider.IProviderContainer.

124.2.1 IProviderContainer Interface

Class Hierarchy

Provider.IProviderContainer

Pascal

```
IProviderContainer = interface;
```

C++

```
__interface IProviderContainer;
```

File

Provider

Description

This is class Provider.IProviderContainer.

124.2.1.1 IProviderContainer Members

The following tables list the members exposed by IProviderContainer.

Public Methods

	Name	Description
	RegisterProvider (see page 18614)	This is RegisterProvider, a member of class IProviderContainer.
	UnRegisterProvider (see page 18614)	This is UnRegisterProvider, a member of class IProviderContainer.

124.2.1.2 IProviderContainer Methods

The methods of the IProviderContainer class are listed here.

Public Methods

	Name	Description
	RegisterProvider (see page 18614)	This is RegisterProvider, a member of class IProviderContainer.
	UnRegisterProvider (see page 18614)	This is UnRegisterProvider, a member of class IProviderContainer.

124.2.1.2.1 IProviderContainer.RegisterProvider Method

Pascal

```
procedure RegisterProvider(Prov: TCustomProvider);
```

C++

```
__fastcall RegisterProvider(TCustomProvider Prov);
```

Description

This is RegisterProvider, a member of class IProviderContainer.

124.2.1.2.2 IProviderContainer.UnRegisterProvider Method

Pascal

```
procedure UnRegisterProvider(Prov: TCustomProvider);
```

C++

```
__fastcall UnRegisterProvider(TCustomProvider Prov);
```

Description

This is UnRegisterProvider, a member of class IProviderContainer.

124.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetObjectProperty (see page 18615)	This is function Provider.GetObjectProperty.
	GetStringProperty (see page 18615)	This is function Provider.GetStringProperty.
	VarArrayFromStrings (see page 18615)	Creates a Variant array that contains the strings in a TStrings object.
	VarArrayFromWideStrings (see page 18616)	This is function Provider.VarArrayFromWideStrings.

124.3.1 Provider.GetObjectProperty Function

Pascal

```
function GetObjectProperty(Instance: TPersistent; const PropName: string): TObject;
```

C++

```
TObject * GetObjectProperty(TPersistent * Instance, const AnsiString PropName);
```

File

Provider

Description

This is function Provider.GetObjectProperty.

124.3.2 Provider.GetStringProperty Function

Pascal

```
function GetStringProperty(Instance: TPersistent; const PropName: string): WideString;
```

C++

```
BSTR GetStringProperty(TPersistent * Instance, const AnsiString PropName);
```

File

Provider

Description

This is function Provider.GetStringProperty.

124.3.3 Provider.VarArrayFromStrings Function

Creates a Variant array that contains the strings in a TStrings object.

Pascal

```
function VarArrayFromStrings(Strings: TStrings): Variant;
```

C++

```
Variant VarArrayFromStrings(TStrings Strings);
```

File

Provider

Description

Call VarArrayFromStrings to create a Variant array to represent the strings in a TStrings object. VarArrayFromStrings creates a new Variant array of WideString values, where each WideString corresponds to a string in the specified TStrings object.

Strings is the TStrings object that supplies the strings for the Variant array.

VarArrayFromStrings returns the new Variant array.

See Also

PackageParams

VarArrayCreate

124.3.4 Provider.VarArrayFromWideStrings Function

Pascal

```
function VarArrayFromWideStrings(Strings: TWideStrings): Variant;
```

C++

```
Variant VarArrayFromWideStrings(TWideStrings Strings);
```

File

Provider

Description

This is function Provider.VarArrayFromWideStrings.

124.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TGetRecordOption (see page 18617)	TGetRecordOption and TGetRecordOptions (see page 18624) influence what type of information is added to data packets.
	TProviderOption (see page 18618)	TProviderOption and TProviderOptions (see page 18626) influence what type of information is communicated between client datasets and provider components.
	TResolverResponse (see page 18620)	TResolverResponse indicates the action a resolver component should take after an OnUpdateError event handler handles an update error.

Records

	Name	Description
	TPutFieldInfo (see page 18620)	This is record Provider.TPutFieldInfo.

124.4.1 Provider.TGetRecordOption Enumeration

TGetRecordOption and TGetRecordOptions (see page 18624) influence what type of information is added to data packets.

Pascal

```
TGetRecordOption = (
  grMetaData,
  grReset,
  grXML,
  grXMLUTF8
);
```

C++

```
enum TGetRecordOption {
  grMetaData,
  grReset,
  grXML,
  grXMLUTF8
};
```

File

Provider

Description

TGetRecordOptions (see page 18624) is a set of TGetRecordOption values. TGetRecordOption values are listed in the following table:

Value	Meaning
grMetaData	The data packet includes metadata as well as record values.
grReset	The records in the data packet start with the first record, regardless of the contents of any previously sent data packets.

grXML	The data packet is transmitted as a string of XML instead of as an OleVariant. Extended characters are expressed using escape sequences.
grXMLUTF8	The data packet is transmitted as a string of XML instead of as an OleVariant. Extended characters are encoded using UTF-8.

See Also

TCustomProvider (see page 18449)

124.4.2 Provider.TProviderOption Enumeration

TProviderOption and TProviderOptions (see page 18626) influence what type of information is communicated between client datasets and provider components.

Pascal

```
TProviderOption = (
  poFetchBlobsOnDemand,
  poFetchDetailsOnDemand,
  poIncFieldProps,
  poCascadeDeletes,
  poCascadeUpdates,
  poReadOnly,
  poAllowMultiRecordUpdates,
  poDisableInserts,
  poDisableEdits,
  poDisableDeletes,
  poNoReset,
  poAutoRefresh,
  poPropogateChanges,
  poAllowCommandText,
  poRetainServerOrder,
  poUseQuoteChar
);
```

C++

```
enum TProviderOption {
  poFetchBlobsOnDemand,
  poFetchDetailsOnDemand,
  poIncFieldProps,
  poCascadeDeletes,
  poCascadeUpdates,
  poReadOnly,
  poAllowMultiRecordUpdates,
  poDisableInserts,
  poDisableEdits,
  poDisableDeletes,
  poNoReset,
  poAutoRefresh,
  poPropogateChanges,
  poAllowCommandText,
  poRetainServerOrder,
  poUseQuoteChar
};
```

File

Provider

Description

TProviderOptions (see page 18626) is a set of TProviderOption values. TProviderOption values are listed in the following table:

Value	Meaning
poFetchBlobsOnDemand	BLOB fields are not included in data packets. Instead, client applications must request these values on an as-needed basis. If the client dataset's FetchOnDemand property is true, the client requests these values automatically. Otherwise, the client application uses the FetchBlobs method to retrieve BLOB data.
poFetchDetailsOnDemand	When the provider represents the master of a master/detail relationship, nested detail values are not included in the data packet. Instead, client applications request these on an as-needed basis. If the client dataset's FetchOnDemand property is true, the client requests these values automatically. Otherwise the client application uses the FetchDetails method to retrieve nested detail sets.
poIncFieldProps	The data packet includes the following field properties (where applicable): Alignment, DisplayLabel, DisplayWidth, Visible, DisplayFormat, EditFormat, MaxValue, MinValue, Currency, EditMask, DisplayValues
poCascadeDeletes	Tells the server to delete detail records automatically when master table records are deleted. To use this option, the provider must represent the master of a master/detail relationship and the database must support cascaded deletes as part of its referential integrity settings.
poCascadeUpdates	Tells the server to update detail records automatically when master table key values are changed. To use this option, the provider must represent the master of a master/detail relationship and the database must support cascaded updates as part of its referential integrity settings.
poReadOnly	The client dataset can't apply updates to the provider.
poAllowMultiRecordUpdates	Enables individual updates that affect multiple records. When poAllowMultiRecordUpdates is not included, updates that change multiple records are automatically aborted.
poDisableInserts	Clients can't add new records. If the client tries to insert a record an exception is raised. (This does not affect the client's ability to edit or delete records)
poDisableEdits	Clients can't modify existing data values. If the client tries to edit a field an exception is raised. (This does not affect the client's ability to insert or delete records)
poDisableDeletes	Clients can't delete records. If the client tries to delete a record an exception is raised. (This does not affect the client's ability to edit or insert records)
poNoReset	Ignores the reset flag in calls to AS_GetRecords (TGetRecordsOption value of grReset).
poAutoRefresh	Refreshes the client dataset with current record values whenever it applies updates.
poPropogateChanges	Changes made in a BeforeUpdateRecord or AfterUpdateRecord event handler are sent back to the client and merged into the client dataset.
poAllowCommandText	Enables the client to override the provider's dataset by replacing its associated SQL text or the name of its stored procedure or table. These overrides are supplied to the GetRecords or Execute method.
poRetainServerOrder	Signals that the client should not attempt to re-establish the default order by sorting the records in the data packet. This prevents the client from changing the record order because, for example, it is unable to duplicate the locale on the server.
poUseQuoteChar	Specifies that the quote character (“) used by the SQL provider will also be used to quote SQL identifiers in the SQL queries created by the provider. This option is True by default; set it False to eliminate the quotes.

See Also

TBaseProvider (see page 18423)

TGetRecordOption (see page 18617)

124.4.3 Provider.TPutFieldInfo Record

Pascal

```
TPutFieldInfo = record
  FieldNo: Integer;
  Field: TField;
  DataSet: TDataSet;
  Size: Integer;
  IsDetail: Boolean;
  Opened: Boolean;
  PutProc: TPutFieldProc;
  LocalFieldIndex: Integer;
  FieldInfos: Pointer;
end;
```

C++

```
struct TPutFieldInfo {
  int FieldNo;
  TField Field;
  TDataSet DataSet;
  int Size;
  Boolean IsDetail;
  Boolean Opened;
  TPutFieldProc PutProc;
  int LocalFieldIndex;
  void * FieldInfos;
};
```

File

Provider

Description

This is record Provider.TPutFieldInfo.

124.4.4 Provider.TResolverResponse Enumeration

TResolverResponse indicates the action a resolver component should take after an OnUpdateError event handler handles an update error.

Pascal

```
TResolverResponse = (
  rrSkip,
  rrAbort,
  rrMerge,
  rrApply,
  rrIgnore
);
```

C++

```
enum TResolverResponse {
  rrSkip,
  rrAbort,
```

```
    rrMerge,
    rrApply,
    rrIgnore
  };
```

File

Provider

Description

The following table lists the possible values for TResolverResponse and what they indicate:

Value	Meaning
rrSkip	Skip updating the record that raised the error condition, and leave the unapplied changes in the cache.
rrAbort	Abort the entire update operation without displaying an error message.
rrMerge	Merge the updates in the delta packet with the new underlying value from the database. This will only succeed if none of the updated fields have been changed by another user.
rrApply	Apply the current record value (as it has been fixed in the OnUpdateError handler) instead of the original update.
rrIgnore	Ignore the error condition: Do not apply the record but do not return the record to the client dataset for resolving either.

See Also

OnUpdateError (see page 18444)

TCustomResolver (see page 18483)

124.5 Types

The following table lists types in this documentation.

Types

Name	Description
PPutFieldInfo (see page 18622)	This is type Provider.PPutFieldInfo.
TAfterUpdateRecordEvent (see page 18622)	This is type Provider.TAfterUpdateRecordEvent.
TBeforeCommitEvent (see page 18623)	This is type Provider.TBeforeCommitEvent.
TBeforeUpdateRecordEvent (see page 18623)	This is type Provider.TBeforeUpdateRecordEvent.
TDataRequestEvent (see page 18623)	This is type Provider.TDataRequestEvent.
TGetDSProps (see page 18624)	This is type Provider.TGetDSProps.
TGetParamsEvent (see page 18624)	This is type Provider.TGetParamsEvent.
TGetRecordOptions (see page 18624)	This is type Provider.TGetRecordOptions.
TGetTableNameEvent (see page 18625)	This is type Provider.TGetTableNameEvent.
TInfoArray (see page 18625)	This is type Provider.TInfoArray.
TOnValidate (see page 18625)	This is type Provider.TOnValidate.
TProviderDataEvent (see page 18626)	This is type Provider.TProviderDataEvent.
TProviderOptions (see page 18626)	This is type Provider.TProviderOptions.
TPutFieldProc (see page 18626)	This is type Provider.TPutFieldProc.
TResolverErrorEvent (see page 18627)	This is type Provider.TResolverErrorEvent.

124.5.1 Provider.PPutFieldInfo Type

Pascal

```
PPutFieldInfo = ^TPutFieldInfo;
```

C++

```
^TPutFieldInfo PPutFieldInfo;
```

File

Provider

Description

This is type Provider.PPutFieldInfo.

124.5.2 Provider.TAfterUpdateRecordEvent Type

Pascal

```
TAfterUpdateRecordEvent = procedure (Sender: TObject; SourceDS: TDataSet; DeltaDS:
TCustomClientDataSet; UpdateKind: TUpdateKind) of object;
```

C++

```
(Sender: TObject; SourceDS: TDataSet; DeltaDS: TCustomClientDataSet; UpdateKind: TUpdateKind)
( TAfterUpdateRecordEvent)();
```

File

Provider

Description

This is type Provider.TAfterUpdateRecordEvent.

124.5.3 Provider.TBeforeCommitEvent Type

Pascal

```
TBeforeCommitEvent = procedure (Sender:TObject; DeltaDS: TCustomClientDataSet; ErrorCount,
MaxErrors: integer; const ResultVar: OleVariant) of object;
```

C++

```
(Sender:TObject; DeltaDS: TCustomClientDataSet; ErrorCount, MaxErrors: integer; const
ResultVar: OleVariant) ( TBeforeCommitEvent)();
```

File

Provider

Description

This is type Provider.TBeforeCommitEvent.

124.5.4 Provider.TBeforeUpdateRecordEvent Type

Pascal

```
TBeforeUpdateRecordEvent = procedure (Sender: TObject; SourceDS: TDataSet; DeltaDS:
TCustomClientDataSet; UpdateKind: TUpdateKind; var Applied: Boolean) of object;
```

C++

```
(Sender: TObject; SourceDS: TDataSet; DeltaDS: TCustomClientDataSet; UpdateKind: TUpdateKind;
var Applied: Boolean) ( TBeforeUpdateRecordEvent)();
```

File

Provider

Description

This is type Provider.TBeforeUpdateRecordEvent.

124.5.5 Provider.TDataRequestEvent Type

Pascal

```
TDataRequestEvent = function (Sender: TObject; Input: OleVariant): OleVariant of object;
```

C++

```
(Sender: TObject; Input: OleVariant): OleVariant ( TDataRequestEvent)();
```

File

Provider

Description

This is type Provider.TDataRequestEvent.

124.5.6 Provider.TGetDSProps Type

Pascal

```
TGetDSProps = procedure (Sender: TObject; DataSet: TDataSet; out Properties: OleVariant) of
object;
```

C++

```
(Sender: TObject; DataSet: TDataSet; out Properties: OleVariant) ( TGetDSProps)();
```

File

Provider

Description

This is type Provider.TGetDSProps.

124.5.7 Provider.TGetParamsEvent Type

Pascal

```
TGetParamsEvent = procedure (DataSet: TDataSet; Params: TList) of object;
```

C++

```
(DataSet: TDataSet; Params: TList) ( TGetParamsEvent)();
```

File

Provider

Description

This is type Provider.TGetParamsEvent.

124.5.8 Provider.TGetRecordOptions Type

Pascal

```
TGetRecordOptions = set of TGetRecordOption;
```

C++

```
TGetRecordOption TGetRecordOptions;
```

File

Provider

Description

This is type Provider.TGetRecordOptions.

124.5.9 Provider.TGetTableNameEvent Type

Pascal

```
TGetTableNameEvent = procedure (Sender: TObject; DataSet: TDataSet; var TableName: WideString)
of object;
```

C++

```
(Sender: TObject; DataSet: TDataSet; var TableName: WideString) ( TGetTableNameEvent)();
```

File

Provider

Description

This is type Provider.TGetTableNameEvent.

124.5.10 Provider.TInfoArray Type

Pascal

```
TInfoArray = array of TPutFieldInfo;
```

C++

```
array of TPutFieldInfo TInfoArray;
```

File

Provider

Description

This is type Provider.TInfoArray.

124.5.11 Provider.TOnValidate Type

Pascal

```
TOnValidate = procedure (const Delta: OleVariant) of object;
```

C++

```
(const Delta: OleVariant) ( TOnValidate)();
```

File

Provider

Description

This is type Provider.TOnValidate.

124.5.12 Provider.TProviderDataEvent Type

Pascal

```
TProviderDataEvent = procedure (Sender: TObject; DataSet: TCustomClientDataSet) of object;
```

C++

```
(Sender: TObject; DataSet: TCustomClientDataSet) ( TProviderDataEvent)();
```

File

Provider

Description

This is type Provider.TProviderDataEvent.

124.5.13 Provider.TProviderOptions Type

Pascal

```
TProviderOptions = set of TProviderOption;
```

C++

```
TProviderOption TProviderOptions;
```

File

Provider

Description

This is type Provider.TProviderOptions.

124.5.14 Provider.TPutFieldProc Type

Pascal

```
TPutFieldProc = procedure (Info: PPutFieldInfo) of object;
```

C++

```
(Info: PPutFieldInfo) ( TPutFieldProc)();
```

File

Provider

Description

This is type Provider.TPutFieldProc.

124.5.15 Provider.TResolverErrorEvent Type

Pascal

```
TResolverErrorEvent = procedure (Sender: TObject; DataSet: TCustomClientDataSet; E:
EUpdateError; UpdateKind: TUpdateKind; var Response: TResolverResponse) of object;
```

C++

```
(Sender: TObject; DataSet: TCustomClientDataSet; E: EUpdateError; UpdateKind: TUpdateKind; var
Response: TResolverResponse) ( TResolverErrorEvent)();
```

File

Provider

Description

This is type Provider.TResolverErrorEvent.

124.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
InformixLob (see page 18628)	This is variable Provider.InformixLob.

124.6.1 Provider.InformixLob Variable

Pascal

```
InformixLob: Boolean;
```

C++

```
Boolean InformixLob;
```

File

Provider

Description

This is variable Provider.InformixLob.

124.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
MetaDataOption (see page 18629)	Specifies the options available when retrieving metadata from a target database.
ResetOption (see page 18630)	This is constant Provider.ResetOption.
XMLOption (see page 18630)	This is constant Provider.XMLOption.
XMLUTF8Option (see page 18631)	This is constant Provider.XMLUTF8Option.

124.7.1 Provider.MetaDataOption Constant

Specifies the options available when retrieving metadata from a target database.

Pascal

```
MetaDataOption: Integer = 1 shl ord(grMetaData);
```

C++

```
int MetaDataOption = 1 shl ord(grMetaData);
```

File

Provider

Description

Each database provides a number of options for data that can be retrieved from and about the database. When you build your own data provider, it can be helpful to use these options when implementing the ISQLMetaData interface. The following table describes the allowable metadata options.

Element	Description
CatalogName	Specifies a system table name.
DatabaseName	Specifies the name of the database.
DatabaseVersion	Specifies the version number of the database.
MaxColumnInSelect	Limits the number of columns that can be specified in a SELECT statement. The column count starts with the first column. Any columns beyond the limit are not returned.
MaxColumnsInTable	Limits the number of columns that can be returned from a table. The column count starts with the first column. Any columns beyond the limit are not returned.
MaxObjectNameLength	Limits the number of characters returned of a database object name. Names exceeding this limit are truncated. If database object names are similar, truncating them might result in duplicate names or meaningless names.
MaxRowSize	Limits the size of any returned rows. This can be useful if you suspect there might be BLOBs in the rows you are selecting, but you want to protect the performance of your query.
MaxSQLLength	Limits the length, in characters, of a SQL statement.
ObjectQuoteChar	Specifies the quote character to use when delimiting object names containing special characters, such as spaces.

PackageName	Specifies the name of a package.
ProcSupportsCursor	Specifies whether or not a stored procedure supports the use of a cursor. Set to **true** if it supports a cursor, otherwise set to **false**.
ProcSupportsCursors	Specifies whether or not a stored procedure supports the use of multiple cursors. Set to **true** if it supports cursors, otherwise set to **false**.
SQLEscapeChar	Specifies the character to use to delimit escape sequences in SQL statements.
SchemaName	Specifies the name of a database schema if different from the DatabaseName.
SupportsTransaction	Specifies whether or not the database can be named in a transaction-based SQL statement. Set to **true** if it supports a transaction, otherwise set to **false**.
SupportsTransactions	Specifies whether or not the database can be named in multiple transaction-based SQL statements. Set to **true** if it supports transactions, otherwise set to **false**.
TransactionIsoLevel	Specifies the transaction isolation level. Isolation levels supported by BDP.NET include ReadCommitted, RepeatableRead, DirtyRead, and Custom.

124.7.2 Provider.ResetOption Constant

Pascal

```
ResetOption: Integer = 1 shl ord(grReset);
```

C++

```
int ResetOption = 1 shl ord(grReset);
```

File

Provider

Description

This is constant Provider.ResetOption.

124.7.3 Provider.XMLOption Constant

Pascal

```
XMLOption: Integer = 1 shl ord(grXML);
```

C++

```
int XMLOption = 1 shl ord(grXML);
```

File

Provider

Description

This is constant Provider.XMLOption.

124.7.4 Provider.XMLUTF8Option Constant

Pascal

```
XMLUTF8Option: Integer = 1 shl ord(grXMLUTF8);
```

C++

```
int XMLUTF8Option = 1 shl ord(grXMLUTF8);
```

File

Provider

Description

This is constant Provider.XMLUTF8Option.

125 Registry Namespace

This is namespace Registry.

Topics

Name	Description
Classes (see page 18634)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 18691)	The following table lists structs, records, enums in this documentation.

Classes

	Name	Description
	ERegistryException (see page 18634)	ERegistryException is the exception class for registry errors.
	TRegIniFile (see page 18635)	TRegIniFile is a low-level wrapper for the Windows 95/NT system registry.
	TRegistry (see page 18645)	TRegistry is a low-level wrapper for the system registry and functions that operate on the registry.
	TRegistryIniFile (see page 18675)	TRegistryIniFile is a wrapper for the Windows system registry.

Structs, Records, Enums

	Name	Description
	TRegDataInfo (see page 18691)	TRegDataInfo is a type used by the GetDataInfo method of TRegistery.
	TRegDataType (see page 18691)	TRegDataType is a type used by the GetDataInfo method of TRegistery.
	TRegKeyInfo (see page 18692)	TRegKeyInfo is a type used by the GetRegKeyInfo method of TRegistery.

125.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ERegistryException (see page 18634)	ERegistryException is the exception class for registry errors.
	TRegIniFile (see page 18635)	TRegIniFile is a low-level wrapper for the Windows 95/NT system registry.
	TRegistry (see page 18645)	TRegistry is a low-level wrapper for the system registry and functions that operate on the registry.
	TRegistryIniFile (see page 18675)	TRegistryIniFile is a wrapper for the Windows system registry.

125.1.1 ERegistryException Class

ERegistryException is the exception class for registry errors.

Class Hierarchy

Pascal

```
ERegistryException = class(Exception);
```

C++

```
class ERegistryException : public Exception;
```

File

Registry

Description

ERegistryException is raised when an application cannot

Create a new key in the registry.

Read a data value associated with a key.

Read a binary data value associated with a key into a local buffer.

Write a data value to a key.

Key creation fails when an application attempts to create a subkey under a key for which it does not have KEY_CREATE_SUB_KEY security access.

Reading a data value associated with a key fails when an application does not have KEY_READ security access, or when the application attempts to read the wrong kind of data from a key (for example, attempting to read string data from an integer data value).

Reading a binary data value into a local buffer fails when an application does not allocate a buffer large enough for the data.

Writing a data value to a key fails because an application does not have KEY_WRITE security access for the current key.

See Also

TRegistry (see page 18645)

TRegIniFile (see page 18635)

TRegistry (see page 18645)

125.1.2 TRegIniFile Class

TRegIniFile is a low-level wrapper for the Windows 95/NT system registry.

Class Hierarchy

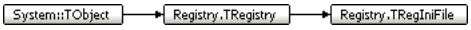

Pascal

```
TRegIniFile = class(TRegistry);
```

C++

```
class TRegIniFile : public TRegistry;
```

File

Registry

Description

TRegIniFile presents a simple interface to the system registry, hiding the need to know about the underlying structure of the registry. TRegIniFile acts as a helper object to TRegistryIniFile (see page 18675), which descends from TCustomIniFile (and is therefore more widely compatible with other INI file classes).

TRegIniFile enables handling the Windows 95/NT system registry as if it were a Windows 3.x INI file. Instead of processing an INI file, however, TRegIniFile reads from and writes to the system registry.

Although TRegIniFile descends from TRegistry (see page 18645), it adds the properties and methods used by a TIniFile object. The FileName (see page 18644) passed to a TRegIniFile object becomes a subkey under the system registry's root key (HKEY_CURRENT_USER by default). What corresponds to a section in an INI file is treated as a key in the system registry, and what corresponds to data entries under a section in an INI file are treated as data values under a key in the system registry.

Note: TRegIniFile is intended to be used primarily as a helper object to TRegistryIniFile (see page 18675), which unifies the interface to INI files and the system registry. However, applications can use TRegIniFile directly.

See Also

TIniFile (see page 15455)

TMemIniFile (see page 15462)

TCustomIniFile (see page 15434)

TRegistryIniFile (see page 18675)

125.1.2.1 TRegIniFile Members

The following tables list the members exposed by TRegIniFile.

Protected Methods

	Name	Description
	ChangeKey (see page 18650)	This is ChangeKey, a member of class TRegistry.
	GetBaseKey (see page 18653)	This is GetBaseKey, a member of class TRegistry.

	GetData (see page 18653)	This is GetData, a member of class TRegistry.
	GetKey (see page 18655)	This is GetKey, a member of class TRegistry.
	PutData (see page 18659)	This is PutData, a member of class TRegistry.
	SetCurrentKey (see page 18666)	This is SetCurrentKey, a member of class TRegistry.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CloseKey (see page 18650)	Writes the current key to the registry and closes the key.
	Create (see page 18638)	This is the overview for the Create constructor overload.
	Create (see page 18647)	This is the overview for the Create constructor overload.
	CreateKey (see page 18651)	Creates a new key in the registry.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DeleteKey (see page 18639)	Erases a data value that is associated with a key.
	DeleteValue (see page 18652)	Erases a data value that is associated with the current key.
V	Destroy (see page 18648)	Closes the current key and destroys the TRegistry (see page 18645) object.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EraseSection (see page 18639)	Deletes a key and its data values in the system registry.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetDataInfo (see page 18653)	Returns information about a specified data value associated with the current key.
	GetDataSize (see page 18654)	Returns the size, in bytes, of a specified data value associated with the current key.
	GetDataType (see page 18654)	Returns the enumerated data type of a specified data value associated with the current key.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetKeyInfo (see page 18655)	Returns information about the current key.
	GetKeyNames (see page 18655)	Returns a string list containing the names of all subkeys belonging to the current key.
	GetValueNames (see page 18656)	Returns a string list containing the names of all data values associated with the current key.
	HasSubKeys (see page 18656)	Determines if the current key has subkeys.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	KeyExists (see page 18656)	Determines if a specified key exists.
	LoadKey (see page 18657)	Creates a subkey under the root key and loads registry information from a file into the newly created subkey.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	MoveKey (see page 18657)	Moves an existing key, its subkeys, and data values to a new location using a new key name.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	OpenKey (see page 18658)	Opens a specified key.
	OpenKeyReadOnly (see page 18658)	Opens a specified key in a read-only manner.
	ReadBinaryData (see page 18659)	Retrieves a binary value from a specified data value associated with the current key.
	ReadBool (see page 18640)	Retrieves a Boolean value from a specified data value associated with a key.
	ReadCurrency (see page 18660)	Retrieves a currency value from a specified data value associated with the current key.
	ReadDate (see page 18661)	Retrieves a date value from a specified data value associated with the current key.
	ReadDateTime (see page 18661)	Retrieves a date and time value from a specified data value associated with the current key.
	ReadFloat (see page 18662)	Retrieves a float value from a specified data value associated with the current key.
	ReadInteger (see page 18640)	Retrieves an integer value from a specified data value (Ident) associated with a key (Section).
	ReadSection (see page 18641)	Retrieves the names of all data values associated with a key, and stores the names in a TStrings object.
	ReadSections (see page 18641)	Retrieves the names of all subkeys associated with the current key.
	ReadSectionValues (see page 18642)	Retrieves the data-values associated with a specified key, and stores the retrieved values in a string object.
	ReadString (see page 18642)	Retrieves a string value from a specified data value (Ident) associated with a key (Section).
	ReadTime (see page 18663)	Retrieves a time value from a specified data value associated with the current key.
	RegistryConnect (see page 18663)	Establishes a connection to a registry on another computer.
	RenameValue (see page 18664)	Changes the name of a data value associated with the current key.
	ReplaceKey (see page 18664)	Replaces the hive file backing a key and all its subkeys with a hive stored in a different file, so that when the system is next started, the key and subkeys will have the values stored in the new file.
	RestoreKey (see page 18665)	Reopens the specified key, and overwrites any information contained in this key and its subkeys keys with registry data stored in a hive file.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveKey (see page 18666)	Opens the specified key with the security access value KEY_ALL_ACCESS and saves the specified key and all of its subkeys and values to a hive file.
	UnLoadKey (see page 18666)	Removes a hive from the registry.
	ValueExists (see page 18667)	Determines if a specified data value exists for the current key.
	WriteBinaryData (see page 18667)	Stores a specifically named data value associated with the current key.
	WriteBool (see page 18643)	Writes a Boolean value to a specified data value associated with a specified key.
	WriteCurrency (see page 18668)	Stores a currency value in a specifically named data value associated with the current key.
	WriteDate (see page 18669)	Stores a date value in a specifically named data value associated with the current key.
	WriteDateTime (see page 18669)	Stores a datetime value in a specifically named data value associated with the current key.
	WriteExpandString (see page 18670)	Stores a string containing unexpanded references to environment variables in a specifically named data value associated with the current key.
	WriteFloat (see page 18670)	Stores a Double value in a specifically named data value associated with the current key.
	WriteInteger (see page 18643)	Writes a long integer value to a specified data value associated with a specified key.
	WriteString (see page 18644)	Writes a string value to a specified data value associated with a specified key.
	WriteTime (see page 18671)	Stores a time value in a specifically named data value associated with the current key.

Public Properties

	Name	Description
	Access (see page 18672)	Specifies the level of access to use when opening keys.

R	CurrentKey (see page 18673)	Specifies the registry key that is currently open.
R	CurrentPath (see page 18673)	Indicates the registry path associated with the current key.
R	FileName (see page 18644)	Specifies the name of the key opened or created in the system registry.
	LazyWrite (see page 18674)	Specifies how keys are written to the registry when calling the CloseKey (see page 18650) procedure.
	RootKey (see page 18674)	Specifies the root key for the TRegistry (see page 18645) object in an application.

125.1.2.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TRegIniFile.Create (string) (see page 18638)	Creates a TRegIniFile (see page 18635) object for an application.
	TRegIniFile.Create (string, LongWord) (see page 18638)	Creates a TRegIniFile (see page 18635) object for an application.

125.1.2.2.1 TRegIniFile.Create Constructor (string)

Creates a TRegIniFile (see page 18635) object for an application.

Pascal

```
constructor Create(const FileName: string); overload;
constructor Create(const FileName: string; AAccess: LongWord); overload;
```

C++

```
__fastcall TRegIniFile(const AnsiString FileName);
__fastcall TRegIniFile(const AnsiString FileName, LongWord AAccess);
```

Description

Use Create to construct a TRegIniFile (see page 18635) object for an application.

FileName (see page 18644) initializes the FileName (see page 18644) property.

AAccess initializes the Access (see page 18672) property, which determines the access rights used when opening keys. If no AAccess parameter is supplied, Access (see page 18672) is initialized to KEY_ALL_ACCESS.

See Also

FileName (see page 18644)

Create (see page 18648)

Access (see page 18672)

125.1.2.3 TRegIniFile Methods

The methods of the TRegIniFile class are listed here.

Public Methods

	Name	Description
	DeleteKey (see page 18639)	Erases a data value that is associated with a key.

	EraseSection (see page 18639)	Deletes a key and its data values in the system registry.
	ReadBool (see page 18640)	Retrieves a Boolean value from a specified data value associated with a key.
	ReadInteger (see page 18640)	Retrieves an integer value from a specified data value (Ident) associated with a key (Section).
	ReadSection (see page 18641)	Retrieves the names of all data values associated with a key, and stores the names in a TStrings object.
	ReadSections (see page 18641)	Retrieves the names of all subkeys associated with the current key.
	ReadSectionValues (see page 18642)	Retrieves the data-values associated with a specified key, and stores the retrieved values in a string object.
	ReadString (see page 18642)	Retrieves a string value from a specified data value (Ident) associated with a key (Section).
	WriteBool (see page 18643)	Writes a Boolean value to a specified data value associated with a specified key.
	WriteInteger (see page 18643)	Writes a long integer value to a specified data value associated with a specified key.
	WriteString (see page 18644)	Writes a string value to a specified data value associated with a specified key.

125.1.2.3.1 TRegIniFile.DeleteKey Method

Erases a data value that is associated with a key.

Pascal

```
procedure DeleteKey(const Section: String; const Ident: String);
```

C++

```
__fastcall DeleteKey(const AnsiString Section, const AnsiString Ident);
```

Description

Call DeleteKey to erase a data value associated with a key. Section is string containing the name of the system registry key containing the value to delete. Ident is a string containing the name of the data value to delete.

Note: DeleteKey for TRegIniFile (see page 18635) obscures the DeleteKey method of TRegistry (see page 18645), from which TRegIniFile (see page 18635) descends. You can access TRegistry (see page 18645)'s DeleteKey method by casting the call to DeleteKey as type TRegistry (see page 18645), or by using the scope resolution operator.

Note: On Windows 95, DeleteKey will delete the key and all its subkeys. On Windows NT, the subkeys must be explicitly deleted by separate calls to DeleteKey.

See Also

DeleteKey (see page 18651)

EraseSection (see page 18639)

125.1.2.3.2 TRegIniFile.EraseSection Method

Deletes a key and its data values in the system registry.

Pascal

```
procedure EraseSection(const Section: string);
```

C++

```
__fastcall EraseSection(const AnsiString Section);
```

Description

Call EraseSection to remove a key and its data values from the system registry. Section identifies the key to delete.

See Also

DeleteKey (see page 18639)

125.1.2.3.3 TRegIniFile.ReadBool Method

Retrieves a Boolean value from a specified data value associated with a key.

Pascal

```
function ReadBool(const Section: string; const Ident: string; Default: Boolean): Boolean;
```

C++

```
__fastcall Boolean ReadBool(const AnsiString Section, const AnsiString Ident, Boolean Default);
```

Description

Call ReadBool to retrieve a Boolean value from a specified data value associated with a key.

Section is a string that identifies the key from which to retrieve a data value. Ident is a string that identifies the name of the data value to return. Default is a Boolean value to use if there is no key corresponding to Section or no data value corresponding to Ident.

Note: This function overrides the ReadBool function of TRegistry (see page 18645), from which TRegIniFile (see page 18635) is derived. To call the ReadBool function of TRegistry (see page 18645) instead of this function, cast the call to ReadBool as TRegistry (see page 18645).

See Also

ReadInteger (see page 18640)

ReadString (see page 18642)

ReadBool (see page 18660)

125.1.2.3.4 TRegIniFile.ReadInteger Method

Retrieves an integer value from a specified data value (Ident) associated with a key (Section).

Pascal

```
function ReadInteger(const Section: string; const Ident: string; Default: Longint): Longint;
```

C++

```
__fastcall Longint ReadInteger(const AnsiString Section, const AnsiString Ident, Longint
Default);
```

Description

Call ReadInteger to retrieve an integer value from a specified data value associated with a key.

Section is a string that identifies the key from which to retrieve a data value. Ident is a string that identifies the name of the data value to return. Default is an integer value to use if there is no key corresponding to Section or no data value corresponding to Ident.

Note: This function overrides the ReadInteger function of TRegistry (see page 18645), from which TRegIniFile (see page 18635) is derived. To call the ReadInteger function of TRegistry (see page 18645) instead of this function, cast the call to ReadInteger as TRegistry (see page 18645).

See Also

ReadBool (see page 18640)

ReadString (see page 18642)

ReadInteger (see page 18662)

125.1.2.3.5 TRegIniFile.ReadSection Method

Retrieves the names of all data values associated with a key, and stores the names in a TStrings object.

Pascal

```
procedure ReadSection(const Section: string; Strings: TStrings);
```

C++

```
__fastcall ReadSection(const AnsiString Section, TStrings Strings);
```

Description

Call ReadSection to retrieve the names of all data values associated with a key into a TStrings object.

Section is a string identifying the key for which to retrieve data-value names.

Strings specifies the string object to hold the retrieved names. Strings can point to a TStrings object such as a string list, or to a component property, such as Items for a TListBox component, that is a TStrings object itself.

See Also

ReadSections (see page 18641)

ReadSectionValues (see page 18642)

TListBox (see page 21598)

Items (see page 21348)

125.1.2.3.6 TRegIniFile.ReadSections Method

Retrieves the names of all subkeys associated with the current key.

Pascal

```
procedure ReadSections(Strings: TStrings);
```

C++

```
__fastcall ReadSections(TStrings Strings);
```

Description

Call ReadSections to retrieve the names of all subkeys associated with the current key.

Strings specifies the string object to hold the retrieved names. Strings can point to a TStrings object, or to a component property, such as Items for a TListBox component, that is a TStrings object itself.

See Also

CurrentKey (see page 18673)

ReadSection (see page 18641)

ReadSectionValues (see page 18642)

TListBox (see page 21598)

Items (see page 21348)

125.1.2.3.7 TRegIniFile.ReadSectionValues Method

Retrieves the data-values associated with a specified key, and stores the retrieved values in a string object.

Pascal

```
procedure ReadSectionValues(const Section: string; Strings: TStrings);
```

C++

```
__fastcall ReadSectionValues(const AnsiString Section, TStrings Strings);
```

Description

Call ReadSectionValues to read all data values associated with a specified key into a string object.

Section is the name of the key from which to retrieve data-values.

Strings specifies the string list object to hold the retrieved values. Strings can point to a TStrings object, or to a component property, such as Items for a TListBox component, that is a TStrings object itself. Use the Values property of the string object to access a specific string in the object.

See Also

ReadSection (see page 18641)

ReadSections (see page 18641)

TListBox (see page 21598)

Items (see page 21348)

Values (see page 3086)

125.1.2.3.8 TRegIniFile.ReadString Method

Retrieves a string value from a specified data value (Ident) associated with a key (Section).

Pascal

```
function ReadString(const Section: string; const Ident: string; const Default: string): string;
```

C++

```
__fastcall AnsiString ReadString(const AnsiString Section, const AnsiString Ident, const
AnsiString Default);
```

Description

Call ReadString to retrieve a string value from a specified data value associated with a key.

Section is a string that identifies the key from which to retrieve a data value. Ident is a string that identifies the name of the data value to return. Default is a string value to use if there is no key corresponding to Section or no data value corresponding to Ident.

Note: This function overrides the ReadString function of TRegistry (see page 18645), from which TRegIniFile (see page 18635) is derived. To call the ReadString function of TRegistry (see page 18645) instead of this function, cast the call to ReadString as TRegistry (see page 18645).

See Also

ReadBool (see page 18640)

ReadInteger (see page 18640)

ReadString (see page 18662)

125.1.2.3.9 TRegIniFile.WriteBool Method

Writes a Boolean value to a specified data value associated with a specified key.

Pascal

```
procedure WriteBool(const Section: string; const Ident: string; Value: Boolean);
```

C++

```
__fastcall WriteBool(const AnsiString Section, const AnsiString Ident, Boolean Value);
```

Description

Call WriteBool to store a Boolean value in a data value associated with a specified key.

Section is a string identifying the key into which to store a data value. Ident is a string identifying the name of the data value into which to write. Value is the Boolean value to write into the data value.

Note: This function overrides the WriteBool procedure of TRegistry (see page 18645), from which TRegIniFile (see page 18635) is derived. To call the WriteBool procedure of TRegistry (see page 18645) instead of this procedure, cast the call to WriteBool as TRegistry (see page 18645).

See Also

WriteInteger (see page 18643)

WriteString (see page 18644)

WriteBool (see page 18668)

125.1.2.3.10 TRegIniFile.WriteInteger Method

Writes a long integer value to a specified data value associated with a specified key.

Pascal

```
procedure WriteInteger(const Section: string; const Ident: string; Value: Longint);
```

C++

```
__fastcall WriteInteger(const AnsiString Section, const AnsiString Ident, Longint Value);
```

Description

Call WriteInteger to store an integer value in a data value associated with a specified key.

Section is a string identifying the key into which to store a data value. Ident is a string identifying the name of the data value into which to write. Value is the integer value to write into the data value.

Note: This function overrides the WriteInteger procedure of TRegistry (see page 18645), from which TRegIniFile (see page 18635) is derived. To call the WriteInteger procedure of TRegistry (see page 18645) instead of this procedure, cast the call to WriteInteger as TRegistry (see page 18645).

See Also

WriteBool (see page 18643)

WriteString (see page 18644)

WriteInteger (see page 18671)

125.1.2.3.11 TRegIniFile.WriteString Method

Writes a string value to a specified data value associated with a specified key.

Pascal

```
procedure WriteString(const Section: String; const Ident: String; const Value: String);
```

C++

```
__fastcall WriteString(const AnsiString Section, const AnsiString Ident, const AnsiString
Value);
```

Description

Call WriteString to store a string value in a data value associated with a specified key.

Section is a string identifying the key into which to store a data value. Ident is a string identifying the name of the data value into which to write. Value is the string value to write into the data value.

Note: This function overrides the WriteString procedure of TRegistry (see page 18645), from which TRegIniFile (see page 18635) is derived. To call the WriteString procedure of TRegistry (see page 18645) instead of this procedure, cast the call to WriteString as TRegistry (see page 18645).

See Also

WriteBool (see page 18643)

WriteInteger (see page 18643)

WriteString (see page 18671)

125.1.2.4 TRegIniFile Properties

The properties of the TRegIniFile class are listed here.

Public Properties

	Name	Description
R	FileName (see page 18644)	Specifies the name of the key opened or created in the system registry.

125.1.2.4.1 TRegIniFile.FileName Property

Specifies the name of the key opened or created in the system registry.

Pascal

```
property FileName: string;
```

C++

```
__property AnsiString FileName;
```

Description

Query FileName to determine the name of the key in the system registry that is the root key for all TRegIniFile (see page 18635) operations.

See Also

CurrentKey (see page 18673)

RootKey (see page 18674)

125.1.3 TRegistry Class

TRegistry is a low-level wrapper for the system registry and functions that operate on the registry.

Class Hierarchy

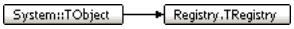

Pascal

```
TRegistry = class(TObject);
```

C++

```
class TRegistry : public TObject;
```

File

Registry

Description

Use TRegistry to encapsulate access to the Windows system registry in an application. The registry is a database that an application can use to store and retrieve configuration information. Configuration information is stored in a hierarchical tree. Each node in the tree is called a key. Every key can contain subkeys and data values that represent part of the configuration information for an application.

All keys that an application creates, opens, reads, or writes are subkeys of predefined root keys. By default, a TRegistry object is created with a root key of HKEY_CURRENT_USER.

Only one key is accessible at a time in a TRegistry object. To determine the key that is currently accessible, read the value of the CurrentKey (see page 18673) property. TRegistry methods enable an application to open, close, save, move, copy, and delete keys.

One or more data values containing actual configuration information can be stored in a key. TRegistry methods enable an application to query a key to see if it contains data, to read data in a key, and to write data to a key.

Note: The TRegistry component is not fully compatable with the Windows NT environment. Functions such as RestoreKey (see page 18665) and SaveKey (see page 18666) may not perform properly without a significant number of workarounds.

125.1.3.1 TRegistry Members

The following tables list the members exposed by TRegistry.

Protected Methods

	Name	Description
	ChangeKey (see page 18650)	This is ChangeKey, a member of class TRegistry.
	GetBaseKey (see page 18653)	This is GetBaseKey, a member of class TRegistry.
	GetData (see page 18653)	This is GetData, a member of class TRegistry.
	GetKey (see page 18655)	This is GetKey, a member of class TRegistry.
	PutData (see page 18659)	This is PutData, a member of class TRegistry.
	SetCurrentKey (see page 18666)	This is SetCurrentKey, a member of class TRegistry.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CloseKey (see page 18650)	Writes the current key to the registry and closes the key.
	Create (see page 18647)	This is the overview for the Create constructor overload.
	CreateKey (see page 18651)	Creates a new key in the registry.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DeleteKey (see page 18651)	Removes a specified key and its associated data from the registry.
	DeleteValue (see page 18652)	Erases a data value that is associated with the current key.
	Destroy (see page 18648)	Closes the current key and destroys the TRegistry (see page 18645) object.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetDataInfo (see page 18653)	Returns information about a specified data value associated with the current key.
	GetDataSize (see page 18654)	Returns the size, in bytes, of a specified data value associated with the current key.
	GetDataType (see page 18654)	Returns the enumerated data type of a specified data value associated with the current key.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetKeyInfo (see page 18655)	Returns information about the current key.
	GetKeyNames (see page 18655)	Returns a string list containing the names of all subkeys belonging to the current key.
	GetValueNames (see page 18656)	Returns a string list containing the names of all data values associated with the current key.
	HasSubKeys (see page 18656)	Determines if the current key has subkeys.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	KeyExists (see page 18656)	Determines if a specified key exists.
	LoadKey (see page 18657)	Creates a subkey under the root key and loads registry information from a file into the newly created subkey.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveKey (see page 18657)	Moves an existing key, its subkeys, and data values to a new location using a new key name.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	OpenKey (see page 18658)	Opens a specified key.
	OpenKeyReadOnly (see page 18658)	Opens a specified key in a read-only manner.
	ReadBinaryData (see page 18659)	Retrieves a binary value from a specified data value associated with the current key.
	ReadBool (see page 18660)	Retrieves a Boolean value from a specified data value associated with the current key.
	ReadCurrency (see page 18660)	Retrieves a currency value from a specified data value associated with the current key.

	ReadDate (see page 18661)	Retrieves a date value from a specified data value associated with the current key.
	ReadDateTime (see page 18661)	Retrieves a date and time value from a specified data value associated with the current key.
	ReadFloat (see page 18662)	Retrieves a float value from a specified data value associated with the current key.
	ReadInteger (see page 18662)	Retrieves an integer value from a specified data value associated with the current key.
	ReadString (see page 18662)	Retrieves a string value from a specified data value associated with the current key.
	ReadTime (see page 18663)	Retrieves a time value from a specified data value associated with the current key.
	RegistryConnect (see page 18663)	Establishes a connection to a registry on another computer.
	RenameValue (see page 18664)	Changes the name of a data value associated with the current key.
	ReplaceKey (see page 18664)	Replaces the hive file backing a key and all its subkeys with a hive stored in a different file, so that when the system is next started, the key and subkeys will have the values stored in the new file.
	RestoreKey (see page 18665)	Reopens the specified key, and overwrites any information contained in this key and its subkeys keys with registry data stored in a hive file.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveKey (see page 18666)	Opens the specified key with the security access value KEY_ALL_ACCESS and saves the specified key and all of its subkeys and values to a hive file.
	UnLoadKey (see page 18666)	Removes a hive from the registry.
	ValueExists (see page 18667)	Determines if a specified data value exists for the current key.
	WriteBinaryData (see page 18667)	Stores a specifically named data value associated with the current key.
	WriteBool (see page 18668)	Stores a Boolean value in a specifically named data value associated with the current key.
	WriteCurrency (see page 18668)	Stores a currency value in a specifically named data value associated with the current key.
	WriteDate (see page 18669)	Stores a date value in a specifically named data value associated with the current key.
	WriteDateTime (see page 18669)	Stores a datetime value in a specifically named data value associated with the current key.
	WriteExpandString (see page 18670)	Stores a string containing unexpanded references to environment variables in a specifically named data value associated with the current key.
	WriteFloat (see page 18670)	Stores a Double value in a specifically named data value associated with the current key.
	WriteInteger (see page 18671)	Stores an integer value in a specifically named data value associated with the current key.
	WriteString (see page 18671)	Stores a string value in a specifically named data value associated with the current key.
	WriteTime (see page 18671)	Stores a time value in a specifically named data value associated with the current key.

Public Properties

	Name	Description
	Access (see page 18672)	Specifies the level of access to use when opening keys.
R	CurrentKey (see page 18673)	Specifies the registry key that is currently open.
R	CurrentPath (see page 18673)	Indicates the registry path associated with the current key.
	LazyWrite (see page 18674)	Specifies how keys are written to the registry when calling the CloseKey (see page 18650) procedure.
	RootKey (see page 18674)	Specifies the root key for the TRegistry (see page 18645) object in an application.

125.1.3.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TRegistry.Create () (see page 18648)	Creates a TRegistry (see page 18645) object for an application.
	TRegistry.Create (LongWord) (see page 18648)	Creates a TRegistry (see page 18645) object for an application.

125.1.3.2.1 TRegistry.Create Constructor ()

Creates a TRegistry (see page 18645) object for an application.

Pascal

```
constructor Create; overload;
constructor Create(AAccess: LongWord); overload;
```

C++

```
__fastcall TRegistry();
__fastcall TRegistry(LongWord AAccess);
```

Description

Create instantiates a TRegistry (see page 18645) object and sets the RootKey (see page 18674) property to HKEY_CURRENT_USER and the LazyWrite (see page 18674) property to true. If an AAccess parameter is supplied, this value is used to initialize the Access (see page 18672) property. Otherwise, Access (see page 18672) is initialized to KEY_ALL_ACCESS.

See Also

LazyWrite (see page 18674)

RootKey (see page 18674)

Destroy (see page 18648)

Access (see page 18672)

125.1.3.3 TRegistry.Destroy Destructor

Closes the current key and destroys the TRegistry (see page 18645) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRegistry();
```

Description

Destroy closes the current key and frees the resources allocated for a registry object when it is no longer needed.

See Also

CurrentKey (see page 18673)

Create (see page 18648)

Free (see page 22300)

125.1.3.4 TRegistry Methods

The methods of the TRegistry class are listed here.

Protected Methods

	Name	Description
	ChangeKey (see page 18650)	This is ChangeKey, a member of class TRegistry.
	GetBaseKey (see page 18653)	This is GetBaseKey, a member of class TRegistry.
	GetData (see page 18653)	This is GetData, a member of class TRegistry.
	GetKey (see page 18655)	This is GetKey, a member of class TRegistry.
	PutData (see page 18659)	This is PutData, a member of class TRegistry.
	SetCurrentKey (see page 18666)	This is SetCurrentKey, a member of class TRegistry.

Public Methods

	Name	Description
	CloseKey (see page 18650)	Writes the current key to the registry and closes the key.
	CreateKey (see page 18651)	Creates a new key in the registry.
	DeleteKey (see page 18651)	Removes a specified key and its associated data from the registry.
	DeleteValue (see page 18652)	Erases a data value that is associated with the current key.
	GetDataInfo (see page 18653)	Returns information about a specified data value associated with the current key.
	GetDataSize (see page 18654)	Returns the size, in bytes, of a specified data value associated with the current key.
	GetDataType (see page 18654)	Returns the enumerated data type of a specified data value associated with the current key.
	GetKeyInfo (see page 18655)	Returns information about the current key.
	GetKeyNames (see page 18655)	Returns a string list containing the names of all subkeys belonging to the current key.
	GetValueNames (see page 18656)	Returns a string list containing the names of all data values associated with the current key.
	HasSubKeys (see page 18656)	Determines if the current key has subkeys.
	KeyExists (see page 18656)	Determines if a specified key exists.
	LoadKey (see page 18657)	Creates a subkey under the root key and loads registry information from a file into the newly created subkey.
	MoveKey (see page 18657)	Moves an existing key, its subkeys, and data values to a new location using a new key name.
	OpenKey (see page 18658)	Opens a specified key.
	OpenKeyReadOnly (see page 18658)	Opens a specified key in a read-only manner.
	ReadBinaryData (see page 18659)	Retrieves a binary value from a specified data value associated with the current key.
	ReadBool (see page 18660)	Retrieves a Boolean value from a specified data value associated with the current key.
	ReadCurrency (see page 18660)	Retrieves a currency value from a specified data value associated with the current key.
	ReadDate (see page 18661)	Retrieves a date value from a specified data value associated with the current key.
	ReadDateTime (see page 18661)	Retrieves a date and time value from a specified data value associated with the current key.
	ReadFloat (see page 18662)	Retrieves a float value from a specified data value associated with the current key.
	ReadInteger (see page 18662)	Retrieves an integer value from a specified data value associated with the current key.
	ReadString (see page 18662)	Retrieves a string value from a specified data value associated with the current key.
	ReadTime (see page 18663)	Retrieves a time value from a specified data value associated with the current key.
	RegistryConnect (see page 18663)	Establishes a connection to a registry on another computer.
	RenameValue (see page 18664)	Changes the name of a data value associated with the current key.

	ReplaceKey (see page 18664)	Replaces the hive file backing a key and all its subkeys with a hive stored in a different file, so that when the system is next started, the key and subkeys will have the values stored in the new file.
	RestoreKey (see page 18665)	Reopens the specified key, and overwrites any information contained in this key and its subkeys keys with registry data stored in a hive file.
	SaveKey (see page 18666)	Opens the specified key with the security access value KEY_ALL_ACCESS and saves the specified key and all of its subkeys and values to a hive file.
	UnLoadKey (see page 18666)	Removes a hive from the registry.
	ValueExists (see page 18667)	Determines if a specified data value exists for the current key.
	WriteBinaryData (see page 18667)	Stores a specifically named data value associated with the current key.
	WriteBool (see page 18668)	Stores a Boolean value in a specifically named data value associated with the current key.
	WriteCurrency (see page 18668)	Stores a currency value in a specifically named data value associated with the current key.
	WriteDate (see page 18669)	Stores a date value in a specifically named data value associated with the current key.
	WriteDateTime (see page 18669)	Stores a datetime value in a specifically named data value associated with the current key.
	WriteExpandString (see page 18670)	Stores a string containing unexpanded references to environment variables in a specifically named data value associated with the current key.
	WriteFloat (see page 18670)	Stores a Double value in a specifically named data value associated with the current key.
	WriteInteger (see page 18671)	Stores an integer value in a specifically named data value associated with the current key.
	WriteString (see page 18671)	Stores a string value in a specifically named data value associated with the current key.
	WriteTime (see page 18671)	Stores a time value in a specifically named data value associated with the current key.

125.1.3.4.1 TRegistry.ChangeKey Method

Pascal

```
procedure ChangeKey(Value: HKey; const Path: string);
```

C++

```
__fastcall ChangeKey(HKey Value, const AnsiString Path);
```

Description

This is ChangeKey, a member of class TRegistry.

125.1.3.4.2 TRegistry.CloseKey Method

Writes the current key to the registry and closes the key.

Pascal

```
procedure CloseKey;
```

C++

```
__fastcall CloseKey();
```

Description

Call CloseKey to write the current key to the registry and close the key. An application should not keep keys open any longer than necessary. Many TRegistry (see page 18645) methods routinely call CloseKey after reading from or writing to the registry. Calling CloseKey when there is no current key has no effect.

Note: When an application closes a key, the key is written to the registry. The LazyWrite (see page 18674) property determines how the write takes place.

See Also

CurrentKey (see page 18673)

LazyWrite (see page 18674)

125.1.3.4.3 TRegistry.CreateKey Method

Creates a new key in the registry.

Pascal

```
function CreateKey(const Key: string): Boolean;
```

C++

```
__fastcall Boolean CreateKey(const AnsiString Key);
```

Description

Use CreateKey to add a new key to the registry. Key is the name of the key to create. Key can be an absolute or relative name. An absolute key begins with a backslash (\) and is a subkey of the root key. A relative key is a subkey of the current key.

CreateKey returns true if key creation is successful. On error, an exception is raised. Attempting to create a key that already exists has no effect.

CreateKey creates only non-volatile keys. If you need to create volatile keys on Windows NT, you must call the Windows API directly.

CreateKey creates keys with a security access of KEY_ALL_ACCESS. If you need to create keys with more restricted access, you must call the Windows API directly.

A key created by CreateKey has no values. To set key values, use the WriteCurrency (see page 18668), WriteBinaryData (see page 18667), WriteBool (see page 18668), WriteDate (see page 18669), WriteDateTime (see page 18669), WriteFloat (see page 18670), WriteInteger (see page 18671), WriteString (see page 18671), or WriteTime (see page 18671) procedures.

See Also

WriteCurrency (see page 18668)

WriteBinaryData (see page 18667)

WriteBool (see page 18668)

WriteDate (see page 18669)

WriteDateTime (see page 18669)

WriteFloat (see page 18670)

WriteInteger (see page 18671)

WriteString (see page 18671)

WriteTime (see page 18671)

Access (see page 18672)

125.1.3.4.4 TRegistry.DeleteKey Method

Removes a specified key and its associated data from the registry.

Pascal

```
function DeleteKey(const Key: string): Boolean;
```

C++

```
__fastcall Boolean DeleteKey(const AnsiString Key);
```

Description

Call DeleteKey to remove a specified key and its associated data, if any, from the registry. Under Windows 95, if the key has subkeys, the subkeys and any associated data are also removed. Under Windows NT, subkeys must be explicitly deleted by separate calls to DeleteKey.

DeleteKey returns true if key deletion is successful. On error, DeleteKey returns false.

See Also

CreateKey (see page 18651)

DeleteValue (see page 18652)

125.1.3.4.5 TRegistry.DeleteValue Method

Erases a data value that is associated with the current key.

Pascal

```
function DeleteValue(const Name: string): Boolean;
```

C++

```
__fastcall Boolean DeleteValue(const AnsiString Name);
```

Description

Call DeleteValue to remove a specific data value associated with the current key. Name is string containing the name of the value to delete. Keys can contain multiple data values, and every value associated with a key has a unique name.

DeleteValue returns true if the deletion is successful, false if the deletion fails.

See Also

DeleteKey (see page 18651)

WriteCurrency (see page 18668)

WriteBinaryData (see page 18667)

WriteBool (see page 18668)

WriteDate (see page 18669)

WriteDateTime (see page 18669)

WriteFloat (see page 18670)

WriteInteger (see page 18671)

WriteString (see page 18671)

WriteTime (see page 18671)

125.1.3.4.6 TRegistry.GetBaseKey Method

Pascal

```
function GetBaseKey(Relative: Boolean): HKey;
```

C++

```
__fastcall HKey GetBaseKey(Boolean Relative);
```

Description

This is GetBaseKey, a member of class TRegistry.

125.1.3.4.7 TRegistry.GetData Method

Pascal

```
function GetData(const Name: string; Buffer: Pointer; BufSize: Integer; var RegData:
TRegDataType): Integer;
```

C++

```
__fastcall int GetData(const AnsiString Name, void * Buffer, int BufSize, TRegDataType
RegData);
```

Description

This is GetData, a member of class TRegistry.

125.1.3.4.8 TRegistry.GetDataInfo Method

Returns information about a specified data value associated with the current key.

Pascal

```
function GetDataInfo(const ValueName: string; var Value: TRegDataInfo): Boolean;
```

C++

```
__fastcall Boolean GetDataInfo(const AnsiString ValueName, TRegDataInfo Value);
```

Description

Call GetDataInfo to get information about a data value associated with the current key. ValueName is a string containing the name of the data value to query. Value is a TRegDataInfo (see page 18691) value in which information about the data value is returned. If the data value is of type rdString or rdExpandString, the size of the data value includes an extra byte for the terminating null character.

GetDataInfo returns true if data value information is returned. If GetDataInfo fails, it returns false, and the Value record is filled with zeros.

See Also

CurrentKey (see page 18673)

GetDataSize (see page 18654)

GetDataType (see page 18654)

GetKeyInfo (see page 18655)

125.1.3.4.9 TRegistry.GetDataSize Method

Returns the size, in bytes, of a specified data value associated with the current key.

Pascal

```
function GetDataSize(const ValueName: string): Integer;
```

C++

```
__fastcall int GetDataSize(const AnsiString ValueName);
```

Description

Call GetDataSize to determine the size, in bytes, of a data value associated with the current key. ValueName is a string containing the name of the data value to query.

On success, GetDataSize returns the size of the data value. On failure, GetDataSize returns -1.

Note: If the data value is a string, GetDataSize returns the size of the data value and one extra byte for the terminating null character.

See Also

CcurrentKey (see page 18673)

GetDataInfo (see page 18653)

GetDataType (see page 18654)

GetKeyInfo (see page 18655)

125.1.3.4.10 TRegistry.GetDataType Method

Returns the enumerated data type of a specified data value associated with the current key.

Pascal

```
function GetDataType(const ValueName: string): TRegDataType;
```

C++

```
__fastcall TRegDataType GetDataType(const AnsiString ValueName);
```

Description

Call GetDataType to determine the data type of a data value associated with the current key. ValueName is a string containing the name of the data value to query.

GetDataType returns a TRegDataType (see page 18691) value that indicates the type of value associated with the current key.

See Also

CurrentKey (see page 18673)

GetDataInfo (see page 18653)

GetDataSize (see page 18654)

GetKeyInfo (see page 18655)

125.1.3.4.11 TRegistry.GetKey Method

Pascal

```
function GetKey(const Key: string): HKEY;
```

C++

```
__fastcall HKEY GetKey(const AnsiString Key);
```

Description

This is GetKey, a member of class TRegistry.

125.1.3.4.12 TRegistry.GetKeyInfo Method

Returns information about the current key.

Pascal

```
function GetKeyInfo(var Value: TRegKeyInfo): Boolean;
```

C++

```
__fastcall Boolean GetKeyInfo(TRegKeyInfo Value);
```

Description

Call GetKeyInfo to return all information about the current key to an application. Value is a variable declared in an application to store the key information returned by this function.

On success GetKeyInfo returns true. On failure it returns false, and the Value record is set to zeros.

See Also

CurrentKey (see page 18673)

GetKeyNames (see page 18655)

GetValueNames (see page 18656)

HasSubKeys (see page 18656)

125.1.3.4.13 TRegistry.GetKeyNames Method

Returns a string list containing the names of all subkeys belonging to the current key.

Pascal

```
procedure GetKeyNames(Strings: TStrings);
```

C++

```
__fastcall GetKeyNames(TStrings Strings);
```

Description

Call GetKeyNames to determine the names of all subkeys belonging to the current key. Determining the names of subkeys is useful in an application that iterates through a set of keys.

Strings is a variable of type TStrings into which to return the list of subkey names.

See Also

CurrentKey (see page 18673)

GetKeyInfo (see page 18655)

GetValueNames (see page 18656)

125.1.3.4.14 TRegistry.GetValueNames Method

Returns a string list containing the names of all data values associated with the current key.

Pascal

```
procedure GetValueNames(Strings: TStrings);
```

C++

```
__fastcall GetValueNames(TStrings Strings);
```

Description

Call GetValueNames to determine the names of all data values associated with the current key. Determining the names of data values is useful in an application that iterates through a set of data values for a key.

Strings is a variable of type TStrings into which to return the list of data value names.

See Also

CurrentKey (see page 18673)

GetDataInfo (see page 18653)

GetKeyNames (see page 18655)

ValueExists (see page 18667)

125.1.3.4.15 TRegistry.HasSubKeys Method

Determines if the current key has subkeys.

Pascal

```
function HasSubKeys: Boolean;
```

C++

```
__fastcall Boolean HasSubKeys();
```

Description

Call HasSubKeys to determine if the current key has any subkeys. HasSubKeys returns true if the current key has at least one subkey. Otherwise it returns false.

See Also

CurrentKey (see page 18673)

KeyExists (see page 18656)

125.1.3.4.16 TRegistry.KeyExists Method

Determines if a specified key exists.

Pascal

```
function KeyExists(const Key: string): Boolean;
```

C++

```
__fastcall Boolean KeyExists(const AnsiString Key);
```

Description

Call KeyExists to determine if a key of a specified name exists. Key is the name of the key for which to search.

See Also

LoadKey (see page 18657)

ValueExists (see page 18667)

125.1.3.4.17 TRegistry.LoadKey Method

Creates a subkey under the root key and loads registry information from a file into the newly created subkey.

Pascal

```
function LoadKey(const Key: string; const FileName: string): Boolean;
```

C++

```
__fastcall Boolean LoadKey(const AnsiString Key, const AnsiString FileName);
```

Description

Call LoadKey to:

1Create (see page 18648) a new subkey under the root key, and

2Load registry information from a file into the subkey. Registry (see page 18633) information can include data values, subkeys, and data values for those subkeys.

LoadKey is intended to simplify creation of a key, its values and subkeys, and the values for those subkeys in a single operation. A key, its subkeys, and all data values of the key and its subkeys is called a hive. Rather than creating each key and value separately, an application can read a hive from a file. This is especially useful for applications that users can reconfigure at run time.

Note: Before an application calls LoadKey, the RootKey (see page 18674) property must be set to HKEY_USERS, HKEY_LOCAL_MACHINE, or to a key returned by a previous call to RegistryConnect (see page 18663).

The Key parameter is the name of the subkey to create. The FileName parameter is the location of the file containing registry information to store in the subkey. The file specified by FileName must be one previously created using the SaveKey (see page 18666) function or the RegSaveKey Windows API function. On systems that use a file allocation table (FAT), FileName cannot include an extension.

See Also

RootKey (see page 18674)

CreateKey (see page 18651)

RegistryConnect (see page 18663)

SaveKey (see page 18666)

UnLoadKey (see page 18666)

125.1.3.4.18 TRegistry.MoveKey Method

Moves an existing key, its subkeys, and data values to a new location using a new key name.

Pascal

```
procedure MoveKey(const OldName: string; const NewName: string; Delete: Boolean);
```

C++

```
__fastcall MoveKey(const AnsiString OldName, const AnsiString NewName, Boolean Delete);
```

Description

Call MoveKey to copy or move an existing key, its subkeys, and data values to a different location. Under Windows 95, copying is recursive. If a key contains subkeys and data values, they too, are copied. When the operation is complete, the new key is closed. Under Windows NT, subkeys must be moved explicitly with separate calls to MoveKey.

OldName specifies the key to copy or move. NewName specifies a name for the duplicate key to create. If the key specified by NewName does not exist, MoveKey creates it. Delete specifies whether to delete the old key after the copy operation. If Delete is true, the key specified by OldName is deleted after it is copied. Otherwise the old key is closed.

See Also

CreateKey (see page 18651)

DeleteKey (see page 18651)

OpenKey (see page 18658)

OpenKey (see page 18658)

125.1.3.4.19 TRegistry.OpenKey Method

Opens a specified key.

Pascal

```
function OpenKey(const Key: string; CanCreate: Boolean): Boolean;
```

C++

```
__fastcall Boolean OpenKey(const AnsiString Key, Boolean CanCreate);
```

Description

Call OpenKey to make a specified key the current key. Key is the name of the key to open. If Key is nil (Delphi) or NULL (C++), the CurrentKey (see page 18673) property is set to the key specified by the RootKey (see page 18674) property.

CanCreate specifies whether to create the specified key if it does not exist. If CanCreate is true, the key is created if necessary.

Key is opened or created with the security access value specified by the Access (see page 18672) property. OpenKey only creates non-volatile keys, A non-volatile key is stored in the registry and is preserved when the system is restarted.

OpenKey returns true if the key is successfully opened or created

See Also

CurrentKey (see page 18673)

RootKey (see page 18674)

CreateKey (see page 18651)

OpenKeyReadOnly (see page 18658)

Access (see page 18672)

125.1.3.4.20 TRegistry.OpenKeyReadOnly Method

Opens a specified key in a read-only manner.

Pascal

```
function OpenKeyReadOnly(const Key: String): Boolean;
```

C++

```
__fastcall Boolean OpenKeyReadOnly(const AnsiString Key);
```

Description

Call OpenKeyReadOnly to make a specified key the current key. Key is the name of the key to open. If Key is nil (Delphi) or NULL (C++), the CurrentKey (see page 18673) property is set to the key specified by the RootKey (see page 18674) property.

Key is opened with the security access value KEY_READ.

OpenKey (see page 18658) returns true if the key is successfully opened and CurrentKey (see page 18673) is set to the key.

Note: After a successful call to OpenKeyReadOnly the Access (see page 18672) property of the registry component is automatically changed to KEY_READ, regardless of what that property was set to prior to the method call.

See Also

CurrentKey (see page 18673)

RootKey (see page 18674)

CreateKey (see page 18651)

OpenKey (see page 18658)

Access (see page 18672)

125.1.3.4.21 TRegistry.PutData Method

Pascal

```
procedure PutData(const Name: string; Buffer: Pointer; BufSize: Integer; RegData:
TRegDataType);
```

C++

```
__fastcall PutData(const AnsiString Name, void * Buffer, int BufSize, TRegDataType RegData);
```

Description

This is PutData, a member of class TRegistry.

125.1.3.4.22 TRegistry.ReadBinaryData Method

Retrieves a binary value from a specified data value associated with the current key.

Pascal

```
function ReadBinaryData(const Name: string; var Buffer; BufSize: Integer): Integer;
```

C++

```
__fastcall int ReadBinaryData(const AnsiString Name,  Buffer, int BufSize);
```

Description

Call ReadBinaryData to read a binary value from a specified data value associated with the current key. Name is the name of the data value to read. Buffer is the application variable into which to read the registry data. Buffer must be large enough to hold all of the data returned. BufSize specifies the size of Buffer.

If successful, ReadBinaryData writes the requested data into Buffer and returns the number of bytes written. If the Registry (see page 18633) entry contains a known type (such as a string), ReadBinaryData raises an exception.

Note: Binary data is typically a complex data structure (a record in Delphi or struct in C++). It might also be an icon or a bitmap although Microsoft recommends against storing graphics objects in the registry for performance reasons.

See Also

CurrentKey (see page 18673)

ReadBool (see page 18660)

ReadCurrency (see page 18660)

ReadDate (see page 18661)

ReadDateTime (see page 18661)

ReadFloat (see page 18662)

ReadInteger (see page 18662)

ReadString (see page 18662)

ReadTime (see page 18663)

125.1.3.4.23 TRegistry.ReadBool Method

Retrieves a Boolean value from a specified data value associated with the current key.

Pascal

```
function ReadBool(const Name: string): Boolean;
```

C++

```
__fastcall Boolean ReadBool(const AnsiString Name);
```

Description

Call ReadBool to read a Boolean value from a specified data value associated with the current key. Name is the name of the data value to read.

If successful, ReadBool returns the requested Boolean value. If the Registry (see page 18633) entry does not contain a boolean or integer value, an exception is raised.

See Also

CurrentKey (see page 18673)

WriteBool (see page 18668)

125.1.3.4.24 TRegistry.ReadCurrency Method

Retrieves a currency value from a specified data value associated with the current key.

Pascal

```
function ReadCurrency(const Name: string): Currency;
```

C++

```
__fastcall Currency ReadCurrency(const AnsiString Name);
```

Description

Call ReadCurrency to read a Currency value from a specified data value associated with the current key. Name is the name of the data value to read.

If successful, ReadCurrency returns a Currency value. If the Registry (see page 18633) entry does not contain a Currency value, an exception is raised.

See Also

CurrentKey (see page 18673)

WriteCurrency (see page 18668)

125.1.3.4.25 TRegistry.ReadDate Method

Retrieves a date value from a specified data value associated with the current key.

Pascal

```
function ReadDate(const Name: string): TDateTime;
```

C++

```
__fastcall TDateTime ReadDate(const AnsiString Name);
```

Description

Call ReadDate to read a date value from a specified data value associated with the current key. Name is the name of the data value to read.

If successful, ReadDate returns a TDateTime value. The integral part of a TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of a TDateTime value is the time of day.

If the Registry (see page 18633) entry does not contain a date or time value, an exception is raised.

Note: Only use ReadDate to read data values previously stored with the WriteDate (see page 18669) procedure.

See Also

CurrentKey (see page 18673)

WriteDate (see page 18669)

ReadDateTime (see page 18661)

125.1.3.4.26 TRegistry.ReadDateTime Method

Retrieves a date and time value from a specified data value associated with the current key.

Pascal

```
function ReadDateTime(const Name: string): TDateTime;
```

C++

```
__fastcall TDateTime ReadDateTime(const AnsiString Name);
```

Description

Call ReadDateTime to read a datetime value from a specified data value associated with the current key. Name is the name of the data value to read.

If successful, ReadDateTime returns a TDateTime value. The integral part of a TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of a TDateTime value is the time of day.

If the Registry (see page 18633) entry does not contain a date or time value, an exception is raised.

See Also

CurrentKey (see page 18673)

WriteDateTime (see page 18669)

ReadDate (see page 18661)

ReadTime (see page 18663)

125.1.3.4.27 TRegistry.ReadFloat Method

Retrieves a float value from a specified data value associated with the current key.

Pascal

```
function ReadFloat(const Name: string): Double;
```

C++

```
__fastcall Double ReadFloat(const AnsiString Name);
```

Description

Call ReadFloat to read a float value from a specified data value associated with the current key. Name is the name of the data value to read.

If the Registry (see page 18633) entry does not contain a floating-point value, an exception is raised.

See Also

CurrentKey (see page 18673)

WriteFloat (see page 18670)

125.1.3.4.28 TRegistry.ReadInteger Method

Retrieves an integer value from a specified data value associated with the current key.

Pascal

```
function ReadInteger(const Name: string): Integer;
```

C++

```
__fastcall int ReadInteger(const AnsiString Name);
```

Description

Call ReadInteger to read an integer value from a specified data value associated with the current key. Name is the name of the data value to read.

If the Registry (see page 18633) entry does not contain an integer, an exception is raised.

See Also

CurrentKey (see page 18673)

WriteInteger (see page 18671)

125.1.3.4.29 TRegistry.ReadString Method

Retrieves a string value from a specified data value associated with the current key.

Pascal

```
function ReadString(const Name: string): string;
```

C++

```
__fastcall AnsiString ReadString(const AnsiString Name);
```

Description

Call ReadString to read a string value from a specified data value associated with the current key. Name is the name of the data value to read.

If there is no Registry (see page 18633) entry with the specified name, ReadString returns an empty string. If the Registry (see page 18633) entry contains something other than a string, an exception is raised.

Note: ReadString retrieves data stored by either the WriteString (see page 18671) or WriteExpandString (see page 18670) method.

See Also

CurrentKey (see page 18673)

WriteString (see page 18671)

WriteExpandString (see page 18670)

125.1.3.4.30 TRegistry.ReadTime Method

Retrieves a time value from a specified data value associated with the current key.

Pascal

```
function ReadTime(const Name: string): TDateTime;
```

C++

```
__fastcall TDateTime ReadTime(const AnsiString Name);
```

Description

Call ReadTime to read a time value from a specified data value associated with the current key. Name is the name of the data value to read.

If successful, ReadTime returns a TDateTime value. The integral part of a TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of a TDateTime value is the time of day.

If the specified Registry (see page 18633) entry does not contain a date or time value, an exception is raised.

Note: Only use ReadTime to read data values previously stored with the WriteTime (see page 18671) method.

See Also

CurrentKey (see page 18673)

WriteTime (see page 18671)

ReadDateTime (see page 18661)

125.1.3.4.31 TRegistry.RegistryConnect Method

Establishes a connection to a registry on another computer.

Pascal

```
function RegistryConnect(const UNCName: string): Boolean;
```

C++

```
__fastcall Boolean RegistryConnect(const AnsiString UNCName);
```

Description

Call RegistryConnect to establish a connection to a registry on another computer running Windows 95 or Windows NT. UNCName is the name of the remote computer, and must take the following form:

```
\\computername
```

where computername is the name of the remote computer. If UNCName is nil (Delphi) or NULL (C++), the local machine name is used.

Note: Before calling RegistryConnect, an application must set the RootKey (see page 18674) property for its registry object to HKEY_USERS or HKEY_LOCAL_MACHINE.

If connection is successful, RegistryConnect sets the RootKey (see page 18674) property to the root key of the remote computer and returns true. If unsuccessful, RegistryConnect returns false, and RootKey (see page 18674) remains unchanged.

See Also

RootKey (see page 18674)

LoadKey (see page 18657)

ReplaceKey (see page 18664)

RestoreKey (see page 18665)

SaveKey (see page 18666)

125.1.3.4.32 TRegistry.RenameValue Method

Changes the name of a data value associated with the current key.

Pascal

```
procedure RenameValue(const OldName: string; const NewName: string);
```

C++

```
__fastcall RenameValue(const AnsiString OldName, const AnsiString NewName);
```

Description

Call RenameValue to change the name of a data value associated with the current key. OldName is a string containing the current name of the data value. NewName is a string containing the replacement name for the data value.

If OldName is the name of an existing data value for the current key, and NewName is not the name of an existing data value for the current key, RenameValue changes the data value name as specified. Otherwise the current name remains unchanged.

See Also

CurrentKey (see page 18673)

GetValueNames (see page 18656)

ValueExists (see page 18667)

MoveKey (see page 18657)

125.1.3.4.33 TRegistry.ReplaceKey Method

Replaces the hive file backing a key and all its subkeys with a hive stored in a different file, so that when the system is next

started, the key and subkeys will have the values stored in the new file.

Pascal

```
function ReplaceKey(const Key: string; const FileName: string; const BackUpFileName: string):
Boolean;
```

C++

```
__fastcall Boolean ReplaceKey(const AnsiString Key, const AnsiString FileName, const
AnsiString BackUpFileName);
```

Description

Call ReplaceKey to replace the hive file backing a key and all its subkeys with a different hive file. A hive is a discrete collection of keys, subkeys, and values that is rooted at the top of the registry hierarchy.

FileName indicates the name of the file containing the new key and subkey information. BackUpFileName is the name of a backup file used to store the current key and subkey information before it is overwritten.

ReplaceKey returns true if it is successful.

Note: The file specified by FileName parameter must be created with the SaveKey (see page 18666) function, or by directly calling the Windows 95/NT RegSaveKey API function. Under the FAT file system FileName cannot have an extension.

See Also

LoadKey (see page 18657)

RestoreKey (see page 18665)

SaveKey (see page 18666)

125.1.3.4.34 TRegistry.RestoreKey Method

Reopens the specified key, and overwrites any information contained in this key and its subkeys keys with registry data stored in a hive file.

Pascal

```
function RestoreKey(const Key: string; const FileName: string): Boolean;
```

C++

```
__fastcall Boolean RestoreKey(const AnsiString Key, const AnsiString FileName);
```

Description

Call RestoreKey to open the specified key with a security access value of KEY_ALL_ACCESS and overwrite any information in the key and its subkeys with registry data stored in a hive file. A hive is a discrete collection of keys, subkeys, and values that is rooted at the top of the registry hierarchy.

Key is the name of the key to open. FileName is the name of the file containing the hive information.

If RestoreKey successfully reads and copies the registry information in FileName to the specified key, the return value of RestoreKey is true and the key is closed.

Note: The file specified by FileName parameter must be created with the SaveKey (see page 18666) function, or by directly calling the Windows 95/NT RegSaveKey API function. Under the FAT file system FileName cannot have an extension.

See Also

LoadKey (see page 18657)

ReplaceKey (see page 18664)

RegistryConnect (see page 18663)

SaveKey (see page 18666)

125.1.3.4.35 TRegistry.SaveKey Method

Opens the specified key with the security access value KEY_ALL_ACCESS and saves the specified key and all of its subkeys and values to a hive file.

Pascal

```
function SaveKey(const Key: string; const FileName: string): Boolean;
```

C++

```
__fastcall Boolean SaveKey(const AnsiString Key, const AnsiString FileName);
```

Description

Call SaveKey to open a key with a security access value of KEY_ALL_ACCESS, and save the key and its subkeys and data values to a hive file. A hive is a discrete collection of keys, subkeys, and values that is rooted at the top of the registry hierarchy.

Key is the key to save. FileName is the file into which to save the key information. It must be the name of a new file that does not already exist. On FAT file systems FileName cannot include an extension.

Files created by SaveKey are passed as parameters to the RestoreKey (see page 18665), ReplaceKey (see page 18664) and LoadKey (see page 18657) functions.

If SaveKey is successful it returns true and closes the key.

See Also

LoadKey (see page 18657)

ReplaceKey (see page 18664)

RestoreKey (see page 18665)

125.1.3.4.36 TRegistry.SetCurrentKey Method

Pascal

```
procedure SetCurrentKey(Value: HKEY);
```

C++

```
__fastcall SetCurrentKey(HKEY Value);
```

Description

This is SetCurrentKey, a member of class TRegistry.

125.1.3.4.37 TRegistry.UnLoadKey Method

Removes a hive from the registry.

Pascal

```
function UnLoadKey(const Key: string): Boolean;
```

C++

```
__fastcall Boolean UnLoadKey(const AnsiString Key);
```

Description

Call UnLoadKey to remove a hive from the registry without modifying its associated hive file, if any. A hive is a discrete collection

of keys, subkeys, and values that is rooted at the top of the registry hierarchy.

Key is the name of the key at the root of the hive to remove.

If successful, UnLoadKey returns true.

Note: Before calling this function, set RootKey (see page 18674) to HKEY_USERS, HKEY_LOCAL_MACHINE, or call RegistryConnect (see page 18663).

See Also

RootKey (see page 18674)

LoadKey (see page 18657)

ReplaceKey (see page 18664)

RestoreKey (see page 18665)

SaveKey (see page 18666)

125.1.3.4.38 TRegistry.ValueExists Method

Determines if a specified data value exists for the current key.

Pascal

```
function ValueExists(const Name: string): Boolean;
```

C++

```
__fastcall Boolean ValueExists(const AnsiString Name);
```

Description

Call ValueExists to determine if a particular data item exists in the registry. Calling ValueExists is especially useful before calling other TRegistry (see page 18645) methods that operate only on existing data items.

Name is the name of the data value for which to check.

ValueExists returns true if a match if found, false otherwise.

See Also

CurrentKey (see page 18673)

KeyExists (see page 18656)

125.1.3.4.39 TRegistry.WriteBinaryData Method

Stores a specifically named data value associated with the current key.

Pascal

```
procedure WriteBinaryData(const Name: string; var Buffer; BufSize: Integer);
```

C++

```
__fastcall WriteBinaryData(const AnsiString Name,  Buffer, int BufSize);
```

Description

Call WriteBinaryData to store a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteBinary. If Name does not exist, it is created.

Buffer is a data buffer containing the data to store in the registry. BufSize indicates the size of Buffer.

Note: Data-value lengths are limited by available memory. Long values (more than 2048 bytes) should be stored as separate files, and those file names should be stored in the registry. Application elements such as icons, bitmaps, and executable files should be stored as separate files rather than as values in the registry.

If WriteBinaryData fails, an exception is raised, and the value is not written to the registry.

See Also

CurrentKey (see page 18673)

ReadBinaryData (see page 18659)

125.1.3.4.40 TRegistry.WriteBool Method

Stores a Boolean value in a specifically named data value associated with the current key.

Pascal

```
procedure WriteBool(const Name: string; Value: Boolean);
```

C++

```
__fastcall WriteBool(const AnsiString Name, Boolean Value);
```

Description

Call WriteBool to store a boolean value in a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteBool. If Name does not exist, it is created.

If WriteBool fails, an exception is raised, and the value is not written to the registry.

See Also

CurrentKey (see page 18673)

ReadBool (see page 18660)

125.1.3.4.41 TRegistry.WriteCurrency Method

Stores a currency value in a specifically named data value associated with the current key.

Pascal

```
procedure WriteCurrency(const Name: string; Value: Currency);
```

C++

```
__fastcall WriteCurrency(const AnsiString Name, Currency Value);
```

Description

Call WriteCurrency to store a Currency value in a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteCurrency. If Name does not exist, it is created.

Value is a Currency value to store in the registry.

If WriteCurrency fails, an exception is raised, and the value is not written to the registry.

See Also

CurrentKey (see page 18673)

ReadCurrency (see page 18660)

125.1.3.4.42 TRegistry.WriteDate Method

Stores a date value in a specifically named data value associated with the current key.

Pascal

```
procedure WriteDate(const Name: string; Value: TDateTime);
```

C++

```
__fastcall WriteDate(const AnsiString Name, TDateTime Value);
```

Description

Call WriteDate to store a datetime value in a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteDate. If Name does not exist, it is created.

Value is a datetime value to store in the registry.

If WriteDate fails, an exception is raised, and the value is not written to the registry.

If successful, WriteDate stores a TDateTime value. The integral part of a TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of a TDateTime value is the time of day.

See Also

CurrentKey (see page 18673)

ReadDate (see page 18661)

WriteDateTime (see page 18669)

125.1.3.4.43 TRegistry.WriteDateTime Method

Stores a datetime value in a specifically named data value associated with the current key.

Pascal

```
procedure WriteDateTime(const Name: string; Value: TDateTime);
```

C++

```
__fastcall WriteDateTime(const AnsiString Name, TDateTime Value);
```

Description

Call WriteDateTime to store a datetime value in a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteDateTime. If Name does not exist, it is created.

Value is a datetime value to store in the registry.

If WriteDateTime fails, an exception is raised, and the value is not written to the registry.

If successful, WriteDateTime stores a TDateTime value. The integral part of a TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of a TDateTime value is the time of day.

See Also

CurrentKey (see page 18673)

ReadDateTime (see page 18661)

WriteDate (see page 18669)

WriteTime (see page 18671)

125.1.3.4.44 TRegistry.WriteExpandString Method

Stores a string containing unexpanded references to environment variables in a specifically named data value associated with the current key.

Pascal

```
procedure WriteExpandString(const Name: string; const Value: string);
```

C++

```
__fastcall WriteExpandString(const AnsiString Name, const AnsiString Value);
```

Description

Call WriteExpandString to store a string that contains unexpanded references to environment variables such as "%PATH%". The string is stored in a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteExpandString. If Name does not exist, it is created.

Value is a string value to store in the registry.

If WriteExpandString fails, an exception is raised.

Note: To write strings that do not contain unexpanded environment variables, use WriteString (see page 18671) instead.

See Also

CurrentKey (see page 18673)

ReadString (see page 18662)

WriteString (see page 18671)

125.1.3.4.45 TRegistry.WriteFloat Method

Stores a Double value in a specifically named data value associated with the current key.

Pascal

```
procedure WriteFloat(const Name: string; Value: Double);
```

C++

```
__fastcall WriteFloat(const AnsiString Name, Double Value);
```

Description

Call WriteFloat to store a float value in a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteFloat. If Name does not exist, it is created.

Value is a float value to store in the registry.

If WriteFloat fails, an exception is raised, and the value is not written to the registry.

See Also

CurrentKey (see page 18673)

ReadFloat (see page 18662)

125.1.3.4.46 TRegistry.WriteInteger Method

Stores an integer value in a specifically named data value associated with the current key.

Pascal

```
procedure WriteInteger(const Name: string; Value: Integer);
```

C++

```
__fastcall WriteInteger(const AnsiString Name, int Value);
```

Description

Call WriteInteger to store an integer value in a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteInteger. If Name does not exist, it is created.

Value is a integer value to store in the registry.

If WriteInteger fails, an exception is raised, and the value is not written to the registry.

See Also

CurrentKey (see page 18673)

ReadInteger (see page 18662)

125.1.3.4.47 TRegistry.WriteString Method

Stores a string value in a specifically named data value associated with the current key.

Pascal

```
procedure WriteString(const Name: string; const Value: string);
```

C++

```
__fastcall WriteString(const AnsiString Name, const AnsiString Value);
```

Description

Call WriteString to store a string value in a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteString. If Name does not exist, it is created.

Value is a string value to store in the registry.

If WriteString fails, an exception is raised, and the value is not written to the registry.

See Also

CurrentKey (see page 18673)

ReadString (see page 18662)

WriteExpandString (see page 18670)

125.1.3.4.48 TRegistry.WriteTime Method

Stores a time value in a specifically named data value associated with the current key.

Pascal

```
procedure WriteTime(const Name: string; Value: TDateTime);
```

C++

```
__fastcall WriteTime(const AnsiString Name, TDateTime Value);
```

Description

Call WriteTime to store a datetime value in a data value associated with the current key.

Name is a string containing the name of the data value in which to store data. If Name already exists, its current value is overwritten by WriteTime. If Name does not exist, it is created.

Value is a datetime value to store in the registry.

If WriteTime fails, an exception is raised, and the value is not written to the registry.

If successful, WriteTime stores a TDateTime value. The integral part of a TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of a TDateTime value is the time of day.

See Also

CurrentKey (see page 18673)

ReadTime (see page 18663)

WriteDateTime (see page 18669)

125.1.3.5 TRegistry Properties

The properties of the TRegistry class are listed here.

Public Properties

	Name	Description
	Access (see page 18672)	Specifies the level of access to use when opening keys.
R	CurrentKey (see page 18673)	Specifies the registry key that is currently open.
R	CurrentPath (see page 18673)	Indicates the registry path associated with the current key.
	LazyWrite (see page 18674)	Specifies how keys are written to the registry when calling the CloseKey (see page 18650) procedure.
	RootKey (see page 18674)	Specifies the root key for the TRegistry (see page 18645) object in an application.

125.1.3.5.1 TRegistry.Access Property

Specifies the level of access to use when opening keys.

Pascal

```
property Access: LongWord;
```

C++

```
__property LongWord Access;
```

Description

Use Access to specify the level of security access to use when opening keys. The OpenKey (see page 18658) method uses the value of Access when opening a registry key. Access is initialized by the TRegistry (see page 18645) constructor, but can be altered before calling OpenKey (see page 18658).

The value of Access is a set of flags or'd together, where each flag indicates a type of permission. Choose a value that grants

sufficient permission to allow the desired actions while not exceeding the access level of the system running the application. The following table lists the flags that can be included:

Flag	Meaning
KEY_ALL_ACCESS	combination of KEY_READ, KEY_WRITE, and KEY_CREATE_LINK.
KEY_READ	combination of KEY_QUERY_VALUE, KEY_ENUMERATE_SUB_KEYS, and KEY_NOTIFY.
KEY_WRITE	combination of KEY_SET_VALUE and KEY_CREATE_SUB_KEY.
KEY_QUERY_VALUE	grants permission to query subkey data.
KEY_ENUMERATE_SUB_KEYS	grants permission to enumerate subkeys.
KEY_NOTIFY	grants ability to receive change notifications.
KEY_SET_VALUE	grants permission to set subkey data.
KEY_CREATE_SUB_KEY	grants permission to create subkeys.
KEY_CREATE_LINK	grants permission to create symbolic links.
KEY_EXECUTE	restricts permission to read-only access.

Note: Access is ignored by the CreateKey (see page 18651) method, which always uses KEY_ALL_ACCESS.

See Also

OpenKey (see page 18658)

Create (see page 18648)

OpenKeyReadOnly (see page 18658)

125.1.3.5.2 TRegistry.CurrentKey Property

Specifies the registry key that is currently open.

Pascal

```
property CurrentKey: HKEY;
```

C++

```
__property HKEY CurrentKey;
```

Description

Use CurrentKey to determine the registry key that is currently open. All operations performed by TRegistry (see page 18645) affect only the current key. To access another key, call the OpenKey (see page 18658) or OpenKeyReadOnly (see page 18658) method.

See Also

CurrentPath (see page 18673)

RootKey (see page 18674)

OpenKey (see page 18658)

OpenKeyReadOnly (see page 18658)

125.1.3.5.3 TRegistry.CurrentPath Property

Indicates the registry path associated with the current key.

Pascal

```
property CurrentPath: string;
```

C++

```
__property AnsiString CurrentPath;
```

Description

Use CurrentPath to determine the registry path of the current key. The system registry is hierarchical, and CurrentPath returns a string that spells out the hierarchy of keys from the root key to the current key, inclusive.

CurrentPath is automatically reset each time a different key becomes the current key.

See Also

CurrentKey (see page 18673)

RootKey (see page 18674)

125.1.3.5.4 TRegistry.LazyWrite Property

Specifies how keys are written to the registry when calling the CloseKey (see page 18650) procedure.

Pascal

```
property LazyWrite: Boolean;
```

C++

```
__property Boolean LazyWrite;
```

Description

Use LazyWrite to specify how keys are written to the registry. LazyWrite is initialized to true when a registry object is first created. When LazyWrite is true, keys are written to the registry when they are closed, but the CloseKey (see page 18650) procedure may return before the write operation takes place.

When LazyWrite is false, keys are written to the registry before CloseKey (see page 18650) returns. Set LazyWrite to false only when absolutely necessary. Setting LazyWrite to false guarantees that the changes an application makes to keys are written to the registry before further application processing takes place, but it uses more system resources when keys are closed. If LazyWrite is false, and an application is displaying or changing many registry entries, there will be a degradation in application performance.

See Also

CloseKey (see page 18650)

125.1.3.5.5 TRegistry.RootKey Property

Specifies the root key for the TRegistry (see page 18645) object in an application.

Pascal

```
property RootKey: HKEY;
```

C++

```
__property HKEY RootKey;
```

Description

Use RootKey to determine the hierarchy of subkeys an application can access, or to specify the root key for the application.

By default, RootKey is set to HKEY_CURRENT_USER when the registry object is created. To change the root key, specify a

valid integer value for the RootKey property.

Note: For some operations, such as LoadKey (see page 18657) or RegistryConnect (see page 18663), an application may need to change the default value of RootKey.

See Also

CurrentKey (see page 18673)

LoadKey (see page 18657)

RegistryConnect (see page 18663)

125.1.4 TRegistryIniFile Class

TRegistryIniFile is a wrapper for the Windows system registry.

Class Hierarchy

Pascal

```
TRegistryIniFile = class(TCustomIniFile);
```

C++

```
class TRegistryIniFile : public TCustomIniFile;
```

File

Registry

Description

TRegistryIniFile presents a simple interface to the system registry and hides the need to know about the underlying structure of the registry. TRegistryIniFile enables handling the Windows system registry as if it were a Windows 3.x INI file. Instead of processing an INI file, however, TRegistryIniFile reads from and writes to the system registry. Because TRegistryIniFile, like TIniFile, descends from TCustomIniFile, you can use it in common code that writes to both INI files and the registry.

In addition, application developers that are switching from the use of INI files to the system registry can use this object to migrate their application with a minimum of coding changes. By finding all references to TIniFile in an application, replacing them with TRegistryIniFile, and recompiling the application, a developer can update an application to use the system registry instead of INI files without having to code any new logic into the application.

Note: Information contained in existing users' INI files is not migrated into the system registry using the method described above. A one-time operation to explicitly copy the INI file data, through TRegistryIniFile, must be used to do this.

TRegistryIniFile adapts the methods inherited from TCustomIniFile to operate on Registry (see page 18633) entries rather than INI file keys. It reinterprets the FileName property as a subkey under the system registry's root key (HKEY_CURRENT_USER by default). What corresponds to a section in an INI file is treated as a key in the system registry, and what corresponds to data entries under a section in an INI file are treated as data values under a key in the system registry.

See Also

TIniFile (see page 15455)

TMemIniFile (see page 15462)

TCustomIniFile (see page 15434)

TRegIniFile (see page 18635)

125.1.4.1 TRegistryIniFile Members

The following tables list the members exposed by TRegistryIniFile.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 18677)	This is the overview for the Create constructor overload.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DeleteKey (see page 18679)	Erases the data value associated with an INI file entry.
	Destroy (see page 18678)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EraseSection (see page 18679)	Erases an entire section of an INI file.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ReadBinaryStream (see page 18679)	Retrieves a binary value from the registry.
	ReadBool (see page 15439)	Retrieves a Boolean value from an ini file.
	ReadDate (see page 18680)	Retrieves a date value from an INI file.
	ReadDateTime (see page 18681)	Retrieves a date-time value from an INI file.
	ReadFloat (see page 18681)	Retrieves a float value from an INI file.
	ReadInteger (see page 18682)	Retrieves an integer value from an INI file.
	ReadSection (see page 18682)	Reads all the key names from a specified section of an INI file into a string list.
	ReadSections (see page 15442)	This is the overview for the ReadSections method overload.
	ReadSections (see page 18683)	This is the overview for the ReadSections method overload.
	ReadSectionValues (see page 18683)	Retrieves the data-values associated with a specified key, and stores the retrieved values in a string object.
	ReadString (see page 18684)	Retrieves a string value from an INI file.
	ReadTime (see page 18685)	Retrieves a time value from an INI file.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SectionExists (see page 15444)	Indicates whether a section exists in the ini file.
	UpdateFile (see page 18685)	Flushes buffered INI file data to disk.

	ValueExists (see page 15445)	Indicates whether a key exists in the ini file.
	WriteBinaryStream (see page 18685)	Writes a binary value to the registry.
	WriteBool (see page 15446)	Writes a Boolean value to an ini file.
	WriteDate (see page 18686)	Writes a date value to an INI file.
	WriteDateTime (see page 18687)	Writes a date-time value to an INI file.
	WriteFloat (see page 18687)	Writes a float value to an INI file.
	WriteInteger (see page 18688)	Writes an integer value to an INI file.
	WriteString (see page 18688)	Writes a string value to an INI file.
	WriteTime (see page 18689)	Writes a time value to an INI file.

Public Properties

	Name	Description
	FileName (see page 15450)	Contains the name of the ini file from which to read and to which to write information.
	RegIniFile (see page 18689)	Contains and internal instance of a TRegIniFile (see page 18635) object.

125.1.4.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TRegistryIniFile.Create (string) (see page 18677)	Creates a TRegistryIniFile (see page 18675) object for an application.
	TRegistryIniFile.Create (string, LongWord) (see page 18677)	Creates a TRegistryIniFile (see page 18675) object for an application.

125.1.4.2.1 TRegistryIniFile.Create Constructor (string)

Creates a TRegistryIniFile (see page 18675) object for an application.

Pascal

```
constructor Create(const FileName: string); overload;
constructor Create(const FileName: string; AAccess: LongWord); overload;
```

C++

```
__fastcall TRegistryIniFile(const AnsiString FileName);
__fastcall TRegistryIniFile(const AnsiString FileName, LongWord AAccess);
```

Description

Create constructs a TRegistryIniFile (see page 18675) and creates an internal TRegIniFile (see page 18635) object used to access the system registry. TRegistryIniFile (see page 18675) uses this TRegIniFile (see page 18635) object as a helper object to provide much of its functionality.

FileName is used as the FileName property for the internal TRegIniFile (see page 18635) object in the RegIniFile (see page 18689) property.

AAccess initializes the Access property of the internal TRegIniFile (see page 18635) object.

See Also

TRegIniFile (see page 18635)

RegIniFile (see page 18689)

Create (see page 22287)

Access (see page 18672)

125.1.4.3 TRegistryIniFile.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRegistryIniFile();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

125.1.4.4 TRegistryIniFile Methods

The methods of the TRegistryIniFile class are listed here.

Public Methods

	Name	Description
	DeleteKey (see page 18679)	Erases the data value associated with an INI file entry.
	EraseSection (see page 18679)	Erases an entire section of an INI file.
	ReadBinaryStream (see page 18679)	Retrieves a binary value from the registry.
	ReadDate (see page 18680)	Retrieves a date value from an INI file.
	ReadDateTime (see page 18681)	Retrieves a date-time value from an INI file.
	ReadFloat (see page 18681)	Retrieves a float value from an INI file.
	ReadInteger (see page 18682)	Retrieves an integer value from an INI file.
	ReadSection (see page 18682)	Reads all the key names from a specified section of an INI file into a string list.
	ReadSections (see page 18683)	This is the overview for the ReadSections method overload.
	ReadSectionValues (see page 18683)	Retrieves the data-values associated with a specified key, and stores the retrieved values in a string object.
	ReadString (see page 18684)	Retrieves a string value from an INI file.

	ReadTime (see page 18685)	Retrieves a time value from an INI file.
	UpdateFile (see page 18685)	Flushes buffered INI file data to disk.
	WriteBinaryStream (see page 18685)	Writes a binary value to the registry.
	WriteDate (see page 18686)	Writes a date value to an INI file.
	WriteDateTime (see page 18687)	Writes a date-time value to an INI file.
	WriteFloat (see page 18687)	Writes a float value to an INI file.
	WriteInteger (see page 18688)	Writes an integer value to an INI file.
	WriteString (see page 18688)	Writes a string value to an INI file.
	WriteTime (see page 18689)	Writes a time value to an INI file.

125.1.4.4.1 TRegistryIniFile.DeleteKey Method

Erases the data value associated with an INI file entry.

Pascal

```
procedure DeleteKey(const Section: String; const Ident: String); override;
```

C++

```
virtual __fastcall DeleteKey(const AnsiString Section, const AnsiString Ident);
```

Description

Call DeleteKey to erase a data value associated with an INI file entry. Section is string containing the name of an INI file section, and Ident is a string containing the name of the key from for which to set a nil (Delphi) or NULL (C++) value.

Note: Attempting to erase a data value in a nonexistent section or attempting to erase data from a nonexistent key are not errors. In these cases, DeleteKey creates the section and key and sets its initial value to an empty value.

See Also

EraseSection (see page 18679)

ValueExists (see page 15445)

125.1.4.4.2 TRegistryIniFile.EraseSection Method

Erases an entire section of an INI file.

Pascal

```
procedure EraseSection(const Section: string); override;
```

C++

```
virtual __fastcall EraseSection(const AnsiString Section);
```

Description

Call EraseSection to remove a section, all its keys, and their data values from an INI file. Section identifies the INI file section to remove. If a section cannot be removed, an exception is raised.

See Also

SectionExists (see page 15444)

DeleteKey (see page 18679)

125.1.4.4.3 TRegistryIniFile.ReadBinaryStream Method

Retrieves a binary value from the registry.

Pascal

```
function ReadBinaryStream(const Section: string; const Name: string; Value: TStream): Integer;
override;
```

C++

```
virtual __fastcall int ReadBinaryStream(const AnsiString Section, const AnsiString Name,
TStream Value);
```

Description

Call ReadBinaryStream to read a binary value into a specified stream. After reading the binary value from the INI file into the current position of the stream, ReadBinaryStream returns the number of bytes written to the stream. The stream is left positioned immediately after the new binary value.

Section is the key for the section that contains the binary data.

Name is the subkey under which the binary data is stored.

Value is a stream to which the binary value is written.

See Also

ReadString (see page 18684)

WriteBinaryStream (see page 18685)

125.1.4.4.4 TRegistryIniFile.ReadDate Method

Retrieves a date value from an INI file.

Pascal

```
function ReadDate(const Section: string; const Name: string; Default: TDateTime): TDateTime;
override;
```

C++

```
virtual __fastcall TDateTime ReadDate(const AnsiString Section, const AnsiString Name,
TDateTime Default);
```

Description

Call ReadDate to read a date value from an INI file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the date value. Default is the date value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 15439)

ReadDateTime (see page 18681)

ReadFloat (see page 18681)

ReadInteger (see page 18682)

ReadString (see page 18684)

ReadTime (see page 18685)

125.1.4.4.5 TRegistryIniFile.ReadDateTime Method

Retrieves a date-time value from an INI file.

Pascal

```
function ReadDateTime(const Section: string; const Name: string; Default: TDateTime):
TDateTime; override;
```

C++

```
virtual __fastcall TDateTime ReadDateTime(const AnsiString Section, const AnsiString Name,
TDateTime Default);
```

Description

Call ReadDateTime to read a date-time value from an INI file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the date-time value. Default is the date-time value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 15439)

ReadDate (see page 18680)

ReadDateTime

ReadFloat (see page 18681)

ReadInteger (see page 18682)

ReadString (see page 18684)

ReadTime (see page 18685)

125.1.4.4.6 TRegistryIniFile.ReadFloat Method

Retrieves a float value from an INI file.

Pascal

```
function ReadFloat(const Section: string; const Name: string; Default: Double): Double;
override;
```

C++

```
virtual __fastcall Double ReadFloat(const AnsiString Section, const AnsiString Name, Double
Default);
```

Description

Call ReadFloat to read a float value from an INI file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the float value. Default is the float value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 15439)

ReadDate (see page 18680)

ReadDateTime (see page 18681)

ReadInteger (see page 18682)

ReadString (see page 18684)

ReadTime (see page 18685)

125.1.4.4.7 TRegistryIniFile.ReadInteger Method

Retrieves an integer value from an INI file.

Pascal

```
function ReadInteger(const Section: string; const Ident: string; Default: Longint): Longint;
override;
```

C++

```
virtual __fastcall Longint ReadInteger(const AnsiString Section, const AnsiString Ident,
Longint Default);
```

Description

Call ReadInteger to read an integer value from an INI file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the value. Default is the integer value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 15439)

ReadDate (see page 18680)

ReadDateTime (see page 18681)

ReadFloat (see page 18681)

ReadString (see page 18684)

ReadTime (see page 18685)

125.1.4.4.8 TRegistryIniFile.ReadSection Method

Reads all the key names from a specified section of an INI file into a string list.

Pascal

```
procedure ReadSection(const Section: string; Strings: TStrings); override;
```

C++

```
virtual __fastcall ReadSection(const AnsiString Section, TStrings Strings);
```

Description

Call ReadSection to retrieve the names of all keys within a specified section of an INI file into a string list object.

Section identifies the section from which to retrieve a list of key names. Strings specifies the string object to hold the retrieved names. Strings can point to a TStrings object such as a string list, or to a component property, such as Items for a TListBox component, that is a TStrings object itself.

See Also

ReadSections (see page 18683)

ReadSectionValues (see page 18683)

SectionExists (see page 15444)

125.1.4.4.9 ReadSections Method

This is the overview for the ReadSections method overload.

Overload List

	Name	Description
	TRegistryIniFile.ReadSections (string, TStrings) (see page 18683)	Reads the names of all sections in an INI file into a string list.
	TRegistryIniFile.ReadSections (TStrings) (see page 18683)	Reads the names of all sections in an INI file into a string list.

125.1.4.4.9.1 TRegistryIniFile.ReadSections Method (TStrings)

Reads the names of all sections in an INI file into a string list.

Pascal

```
procedure ReadSections(Strings: TStrings); override; overload;
procedure ReadSections(const Section: string; Strings: TStrings); override; overload;
```

C++

```
virtual __fastcall ReadSections(TStrings Strings);
virtual __fastcall ReadSections(const AnsiString Section, TStrings Strings);
```

Description

Call ReadSections to retrieve the names of all sections in an INI file into a string list object.

Strings specifies the string object to hold the retrieved names. Strings can point to a TStrings object, or to a component property, such as Items for a TListBox component, that is a TStrings object itself.

See Also

ReadSection (see page 18682)

ReadSectionValues (see page 18683)

125.1.4.4.10 TRegistryIniFile.ReadSectionValues Method

Retrieves the data-values associated with a specified key, and stores the retrieved values in a string object.

Pascal

```
procedure ReadSectionValues(const Section: string; Strings: TStrings); override;
```

C++

```
virtual __fastcall ReadSectionValues(const AnsiString Section, TStrings Strings);
```

Description

Call ReadSectionValues to read all data values associated with a specified key into a string object.

Section is the name of the key from which to retrieve data-values.

Strings specifies the string list object to hold the retrieved values. Strings can point to a TStrings object, or to a component property, such as Items for a TListBox component, that is a TStrings object itself. Use the Values property of the string object to access a specific string in the object.

See Also

ReadSection (see page 18682)

ReadSections (see page 18683)

SectionExists (see page 15444)

125.1.4.4.11 TRegistryIniFile.ReadString Method

Retrieves a string value from an INI file.

Pascal

```
function ReadString(const Section: string; const Ident: string; const Default: string):
string; override;
```

C++

```
virtual __fastcall AnsiString ReadString(const AnsiString Section, const AnsiString Ident,
const AnsiString Default);
```

Description

Call ReadString to read a string value from an INI file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the value. Default is the string value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 15439)

ReadDate (see page 18680)

ReadDateTime (see page 18681)

ReadFloat (see page 18681)

ReadInteger (see page 18682)

ReadTime (see page 18685)

125.1.4.4.12 TRegistryIniFile.ReadTime Method

Retrieves a time value from an INI file.

Pascal

```
function ReadTime(const Section: string; const Name: string; Default: TDateTime): TDateTime;
override;
```

C++

```
virtual __fastcall TDateTime ReadTime(const AnsiString Section, const AnsiString Name,
TDateTime Default);
```

Description

Call ReadTime to read a time value from an INI file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which to retrieve the time value. Default is the time value to return if the:

Section does not exist.

Key does not exist.

Data value for the key is not assigned.

See Also

ReadBool (see page 15439)

ReadDate (see page 18680)

ReadDateTime (see page 18681)

ReadFloat (see page 18681)

ReadInteger (see page 18682)

ReadString (see page 18684)

125.1.4.4.13 TRegistryIniFile.UpdateFile Method

Flushes buffered INI file data to disk.

Pascal

```
procedure UpdateFile; override;
```

C++

```
virtual __fastcall UpdateFile();
```

Description

Call UpdateFile to flush buffered reads from and writes to the INI file to disk. UpdateFile is useful under Windows95, but has no effect under Windows NT as NT does not buffer INI file reads and writes.

See Also

FileName (see page 15450)

125.1.4.4.14 TRegistryIniFile.WriteBinaryStream Method

Writes a binary value to the registry.

Pascal

```
procedure WriteBinaryStream(const Section: string; const Name: string; Value: TStream);
override;
```

C++

```
virtual __fastcall WriteBinaryStream(const AnsiString Section, const AnsiString Name, TStream
Value);
```

Description

Call WriteBinaryStream to write a binary value from a specified stream to a specified key in the registry. WriteBinaryStream writes from the current position in the stream to the end.

Section is the key for the section that contains the subkey where the binary data should be written.

Name is the subkey under which the binary data is stored.

Value is a stream from which the binary value can be read.

Note: Attempting to write a value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteBinaryStream creates the section and key and sets its initial value to the stream's data.

See Also

WriteString (see page 18688)

ReadBinaryStream (see page 18679)

125.1.4.4.15 TRegistryIniFile.WriteDate Method

Writes a date value to an INI file.

Pascal

```
procedure WriteDate(const Section: string; const Name: string; Value: TDateTime); override;
```

C++

```
virtual __fastcall WriteDate(const AnsiString Section, const AnsiString Name, TDateTime Value);
```

Description

Call WriteDate to write a date value to an INI file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the date value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteDate creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 15446)

WriteDateTime (see page 18687)

WriteFloat (see page 18687)

WriteInteger (see page 18688)

WriteString (see page 18688)

WriteTime (see page 18689)

125.1.4.4.16 TRegistryIniFile.WriteDateTime Method

Writes a date-time value to an INI file.

Pascal

```
procedure WriteDateTime(const Section: string; const Name: string; Value: TDateTime); override;
```

C++

```
virtual __fastcall WriteDateTime(const AnsiString Section, const AnsiString Name, TDateTime
Value);
```

Description

Call WriteDateTime to write a date-time value to an INI file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the date-time value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteDateTime creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 15446)

WriteDate (see page 18686)

WriteFloat (see page 18687)

WriteInteger (see page 18688)

WriteString (see page 18688)

WriteTime (see page 18689)

125.1.4.4.17 TRegistryIniFile.WriteFloat Method

Writes a float value to an INI file.

Pascal

```
procedure WriteFloat(const Section: string; const Name: string; Value: Double); override;
```

C++

```
virtual __fastcall WriteFloat(const AnsiString Section, const AnsiString Name, Double Value);
```

Description

Call WriteFloat to write a float value to an INI file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the float value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteFloat creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 15446)

WriteDate (see page 18686)

WriteDateTime (see page 18687)

WriteInteger (see page 18688)

WriteString (see page 18688)

WriteTime (see page 18689)

125.1.4.4.18 TRegistryIniFile.WriteInteger Method

Writes an integer value to an INI file.

Pascal

```
procedure WriteInteger(const Section: string; const Ident: string; Value: Longint); override;
```

C++

```
virtual __fastcall WriteInteger(const AnsiString Section, const AnsiString Ident, Longint
Value);
```

Description

Call WriteInteger to write an integer value to an INI file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the integer value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteInteger creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 15446)

WriteDate (see page 18686)

WriteDateTime (see page 18687)

WriteFloat (see page 18687)

WriteString (see page 18688)

WriteTime (see page 18689)

125.1.4.4.19 TRegistryIniFile.WriteString Method

Writes a string value to an INI file.

Pascal

```
procedure WriteString(const Section: String; const Ident: String; const Value: String);
override;
```

C++

```
virtual __fastcall WriteString(const AnsiString Section, const AnsiString Ident, const
AnsiString Value);
```

Description

Call WriteString to write a string value to an INI file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the string value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteString creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 15446)

WriteDate (see page 18686)

WriteDateTime (see page 18687)

WriteFloat (see page 18687)

WriteInteger (see page 18688)

WriteTime (see page 18689)

125.1.4.4.20 TRegistryIniFile.WriteTime Method

Writes a time value to an INI file.

Pascal

```
procedure WriteTime(const Section: string; const Name: string; Value: TDateTime); override;
```

C++

```
virtual __fastcall WriteTime(const AnsiString Section, const AnsiString Name, TDateTime Value);
```

Description

Call WriteTime to write a time value to an INI file. Section identifies the section in the file that contain the key to which to write. Ident is the name of the key for which to set a value. Value is the time value to write.

Note: Attempting to write a data value to a nonexistent section or attempting to write data to a nonexistent key are not errors. In these cases, WriteTime creates the section and key and sets its initial value to Value.

See Also

WriteBool (see page 15446)

WriteDate (see page 18686)

WriteDateTime (see page 18687)

WriteFloat (see page 18687)

WriteInteger (see page 18688)

WriteString (see page 18688)

125.1.4.5 TRegistryIniFile Properties

The properties of the TRegistryIniFile class are listed here.

Public Properties

	Name	Description
R	RegIniFile (see page 18689)	Contains and internal instance of a TRegIniFile (see page 18635) object.

125.1.4.5.1 TRegistryIniFile.RegIniFile Property

Contains and internal instance of a TRegIniFile (see page 18635) object.

Pascal

```
property RegIniFile: TRegIniFile;
```

C++

```
__property TRegIniFile RegIniFile;
```

Description

RegIniFile is used internally by the TRegistryIniFile (see page 18675) object as the gateway connection between

TRegistryIniFile (see page 18675) methods and the system registry.

See Also

Create (see page 18677)

TRegIniFile (see page 18635)

125.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TRegDataType (see page 18691)	TRegDataType is a type used by the GetDataInfo method of TRegistery.

Records

	Name	Description
	TRegDataInfo (see page 18691)	TRegDataInfo is a type used by the GetDataInfo method of TRegistery.
	TRegKeyInfo (see page 18692)	TRegKeyInfo is a type used by the GetRegKeyInfo method of TRegistery.

125.2.1 Registry.TRegDataInfo Record

TRegDataInfo is a type used by the GetDataInfo method of TRegistery.

Pascal

```
TRegDataInfo = record
  RegData: TRegDataType;
  DataSize: Integer;
end;
```

C++

```
struct TRegDataInfo {
  TRegDataType RegData;
  int DataSize;
};
```

File

Registry

Description

TRegDataInfo is a type used by the GetDataInfo method of TRegistery. Records of type TRegDataInfo contain two fields: a field called RegData, of type TRegDataType (see page 18691), that specifies the data type of the data value; and a field called DataSize that specifies the size, in bytes, of the data value.

125.2.2 Registry.TRegDataType Enumeration

TRegDataType is a type used by the GetDataInfo method of TRegistery.

Pascal

```
TRegDataType = (
  rdUnknown,
  rdString,
  rdExpandString,
  rdInteger,
```

```
  rdBinary
);
```

C++

```
enum TRegDataType {
  rdUnknown,
  rdString,
  rdExpandString,
  rdInteger,
  rdBinary
};
```

File

Registry

Description

TRegDataType is a type used by the GetDataInfo method of TRegistery.It can have the following values:

Value	Meaning
rdUnknown	Data value type is of no defined value type.
rdString	Data value type is a null-terminated string. It is a Unicode or ANSI string depending on whether the application calls the Unicode or ANSI string functions.
rdExpandString	Data value type is a null-terminated string that contains unexpanded references to environment variables (for example, "%PATH%"). It is a Unicode or ANSI string depending on whether the application calls the Unicode or ANSI string functions.
rdInteger	Data value type is a32-bit number.
rdBinary	Data value type is binary data, in any form.

125.2.3 Registry.TRegKeyInfo Record

TRegKeyInfo is a type used by the GetRegKeyInfo method of TRegistery.

Pascal

```
TRegKeyInfo = record
  NumSubKeys: Integer;
  MaxSubKeyLen: Integer;
  NumValues: Integer;
  MaxValueLen: Integer;
  MaxDataLen: Integer;
  FileTime: TFileTime;
end;
```

C++

```
struct TRegKeyInfo {
  int NumSubKeys;
  int MaxSubKeyLen;
  int NumValues;
  int MaxValueLen;
  int MaxDataLen;
  TFileTime FileTime;
};
```

File

Registry

Description

TRegKeyInfo is a type used by the GetRegKeyInfo method of TRegistery to return information about the current key:

Note: Number of subkeys

Note: Longest subkey name length, in characters

Note: Number of data values

Note: Longest data-value name length

Note: Longest data-value length

Note: Time of last write to the key

126 ReqFiles Namespace

This is namespace ReqFiles.

Topics

Name	Description
Classes (see page 18696)	The following table lists classes in this documentation.

Classes

	Name	Description
	TWebRequestFile (see page 18696)	TWebRequestFile describes the content of a file extracted from an HTTP request.
	TWebRequestFiles (see page 18700)	TWebRequestFiles stores a collection of TWebAbstractRequestFile objects.
	TWebRequestFileStream (see page 18704)	This is class ReqFiles.TWebRequestFileStream.

126.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TWebRequestFile (see page 18696)	TWebRequestFile describes the content of a file extracted from an HTTP request.
	TWebRequestFiles (see page 18700)	TWebRequestFiles stores a collection of TWebAbstractRequestFile objects.
	TWebRequestFileStream (see page 18704)	This is class ReqFiles.TWebRequestFileStream.

126.1.1 TWebRequestFile Class

TWebRequestFile describes the content of a file extracted from an HTTP request.

Class Hierarchy

Pascal

```
TWebRequestFile = class(TAbstractWebRequestFile);
```

C++

```
class TWebRequestFile : public TAbstractWebRequestFile;
```

File

ReqFiles

Description

TWebRequestFile describes and provides access to an uploaded file from an HTTP request. The following properties support this access:

FileName, which gives the name of the uploaded file.

FieldName, which gives the name of the adapter field whose value is the uploaded file.

ContentType, which describes how the file data is encoded.

Stream, which is a stream object from which you can read the file content.

Each TWebRequestFile object describes a single uploaded file. A collection of all the uploaded files from the current request is available as the Files property of TWebRequest. In addition, adapter fields that work with uploaded files surface those files using a TUpdateFileList object or IActionFieldValues interface.

See Also

TWebRequestFiles (see page 18700)

IActionFieldValues (see page 925)

TUpdateFileList (see page 24550)

TAdapterFileField (see page 23837)

126.1.1.1 TWebRequestFile Members

The following tables list the members exposed by TWebRequestFile.

Protected Methods

	Name	Description
	GetContentType (see page 18699)	Returns the value of the ContentType property.
	GetFieldName (see page 18699)	Returns the value of the FieldName property.
	GetFileName (see page 18699)	Returns the value of the FileName property.
	GetStream (see page 18700)	Returns the value of the Stream property.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 18698)	Creates an initializes an instance of TWebRequestFile (see page 18696).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 18698)	Frees an instance of TWebRequestFile (see page 18696).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	ContentType (see page 15076)	Describes how the file data is encoded.
	FieldName (see page 15077)	Indicates the name of the adapter field whose value is the contents of the file.
	FileName (see page 15077)	Indicates the name of the uploaded file.
	Stream (see page 15077)	Provides access to the contents of the uploaded file.

126.1.1.2 TWebRequestFile.Create Constructor

Creates an initializes an instance of TWebRequestFile (see page 18696).

Pascal

```
constructor Create(const AFieldName: string; const AFileName: string; const AContentType:
string; AContent: Pointer; AContentLength: Integer);
```

C++

```
__fastcall TWebRequestFile(const AnsiString AFieldName, const AnsiString AFileName, const
AnsiString AContentType, void * AContent, int AContentLength);
```

Description

Typically, applications do not call the TWebRequestFile (see page 18696) constructor. Instead, new Web request file objects are created by calling the Add method of a TWebRequestFiles (see page 18700) collection as the file uploads are extracted from the current Web request.

AFieldName is the name of the adapter field whose value this file represents. It becomes the value of the FieldName property.

AFileName is the name of the file this object represents. It becomes the value of the FileName property.

AContentType describes how the file content is encoded. It becomes the value of the ContentType property.

AContent is an array of bytes that is the content of the file. It is used to initialize the Stream property.

AContentLength is the number of bytes in AContent.

See Also

FieldName

FileName

ContentType

Stream

Add (see page 18703)

126.1.1.3 TWebRequestFile.Destroy Destructor

Frees an instance of TWebRequestFile (see page 18696).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebRequestFile();
```

Description

Do not free instances of TWebRequestFile (see page 18696). TWebRequestFile (see page 18696) objects are created and destroyed by the TWebRequestFiles (see page 18700) object that lists the uploaded files from the current request.

See Also

TWebRequestFiles (see page 18700)

126.1.1.4 TWebRequestFile Methods

The methods of the TWebRequestFile class are listed here.

Protected Methods

	Name	Description
	GetContentType (see page 18699)	Returns the value of the ContentType property.
	GetFieldName (see page 18699)	Returns the value of the FieldName property.
	GetFileName (see page 18699)	Returns the value of the FileName property.
	GetStream (see page 18700)	Returns the value of the Stream property.

126.1.1.4.1 TWebRequestFile.GetContentType Method

Returns the value of the ContentType property.

Pascal

```
function GetContentType: string; override;
```

C++

```
virtual __fastcall AnsiString GetContentType();
```

Description

GetContentType is the protected read implementation for the ContentType property.

In TAbstractWebRequestFile, GetContentType is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the ContentType property.

See Also

ContentType (see page 15076)

126.1.1.4.2 TWebRequestFile.GetFieldName Method

Returns the value of the FieldName property.

Pascal

```
function GetFieldName: string; override;
```

C++

```
virtual __fastcall AnsiString GetFieldName();
```

Description

GetFieldName is the protected read implementation for the FieldName property.

In TAbstractWebRequestFile, GetFieldName is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the FieldName property.

See Also

FieldName (see page 15077)

126.1.1.4.3 TWebRequestFile.GetFileName Method

Returns the value of the FileName property.

Pascal

```
function GetFileName: string; override;
```

C++

```
virtual __fastcall AnsiString GetFileName();
```

Description

GetFileName is the protected read implementation for the FileName property.

In TAbstractWebRequestFile, GetFileName is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the FileName property.

See Also

FileName (see page 15077)

126.1.1.4.4 TWebRequestFile.GetStream Method

Returns the value of the Stream property.

Pascal

```
function GetStream: TStream; override;
```

C++

```
virtual __fastcall TStream GetStream();
```

Description

GetStream is the protected read implementation for the Stream property.

In TAbstractWebRequestFile, GetStream is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Stream property.

See Also

Stream (see page 15077)

126.1.2 TWebRequestFiles Class

TWebRequestFiles stores a collection of TWebAbstractRequestFile objects.

Class Hierarchy

Pascal

```
TWebRequestFiles = class(TAbstractWebRequestFiles);
```

C++

```
class TWebRequestFiles : public TAbstractWebRequestFiles;
```

File

ReqFiles

Description

TWebRequestFiles manages a set of Web request files objects (TWebRequestFile (see page 18696) objects). Each Web

request file object in the collection describes an uploaded file that was extracted from the current HTTP request object.

See Also

TWebRequestFile (see page 18696)

TWebRequest (see page 15161)

126.1.2.1 TWebRequestFiles Members

The following tables list the members exposed by TWebRequestFiles.

Protected Methods

	Name	Description
V	GetCount (see page 18704)	Returns the value of the Count property.
V	GetItem (see page 18704)	Returns the value of the Items property.

Public Methods

	Name	Description
	Add (see page 18702)	This is the overview for the Add method overload.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 18702)	Creates and initializes an instance of TWebRequestFiles (see page 18700).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 18702)	Frees an instance of TWebRequestFiles (see page 18700).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Count (see page 15081)	Indicates the number of uploaded files in the collection.
R	Items (see page 15081)	Lists the uploaded files in the collection.

126.1.2.2 TWebRequestFiles.Create Constructor

Creates and initializes an instance of TWebRequestFiles (see page 18700).

Pascal

```
constructor Create;
```

C++

```
__fastcall TWebRequestFiles();
```

Description

Call Create to instantiate TWebRequestFiles (see page 18700) at runtime. TWebRequestFiles (see page 18700) instances are created automatically by the Web request object to implement its Files property.

See Also

Files (see page 15182)

126.1.2.3 TWebRequestFiles.Destroy Destructor

Frees an instance of TWebRequestFiles (see page 18700).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebRequestFiles();
```

Description

Use the Free method to delete an instance of TWebRequestFiles (see page 18700). Free checks for a nil reference before calling Destroy.

See Also

Free (see page 22300)

126.1.2.4 TWebRequestFiles Methods

The methods of the TWebRequestFiles class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 18704)	Returns the value of the Count property.
	GetItem (see page 18704)	Returns the value of the Items property.

Public Methods

	Name	Description
	Add (see page 18702)	This is the overview for the Add method overload.

126.1.2.4.1 Add Method

This is the overview for the Add method overload.

Overload List

	Name	Description
	TWebRequestFiles.Add (string, string, string, Pointer, Integer) (see page 18703)	Adds a new Web request file object to the collection.
	TWebRequestFiles.Add (TWebRequestFile) (see page 18703)	Adds a new Web request file object to the collection.

126.1.2.4.1.1 TWebRequestFiles.Add Method (TWebRequestFile)

Adds a new Web request file object to the collection.

Pascal

```
procedure Add(const AName: string; const AFileName: string; const AContentType: string;
AContent: Pointer; AContentLength: Integer); overload;
procedure Add(AFile: TWebRequestFile); overload;
```

C++

```
__fastcall Add(const AnsiString AName, const AnsiString AFileName, const AnsiString
AContentType, void * AContent, int AContentLength);
__fastcall Add(TWebRequestFile AFile);
```

Description

Call Add to add a TWebRequestFile (see page 18696) object to the collection. Add inserts the new Web request file object at the end of the Items property array and increments the value of the Count property.

Use the first syntax to cause TWebRequestFiles (see page 18700) to create the Web request file object you are adding. Use the second syntax to add an existing TWebRequestFile (see page 18696) object. Using either syntax, TWebRequestFiles (see page 18700) assumes ownership of the new Web request file object and is responsible for freeing it.

AName is the name of the adapter field whose value the new file represents. It becomes the value of the FieldName property of the new TWebRequestFile (see page 18696) object.

AFileName is the name of the file to add. It becomes the value of the FileName property of the new TWebRequestFile (see page 18696) object.

AContentType describes how the file content is encoded. It becomes the value of the ContentType property of the new TWebRequestFile (see page 18696) object.

AContent is an array of bytes that is the content of the file. It is used to initialize the Stream property of the new TWebRequestFile (see page 18696) object.

AContentLength is the number of bytes in AContent.

AFile is an existing TWebRequestFile (see page 18696) object that describes the file you are adding.

See Also

Count

Items

FieldName

FileName

ContentType

Stream

126.1.2.4.2 TWebRequestFiles.GetCount Method

Returns the value of the Count property.

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCount is the protected read implementation of the Count property.

In TAbstractWebRequestFile, GetCount is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Count property.

See Also

Count (see page 15081)

126.1.2.4.3 TWebRequestFiles.GetItem Method

Returns the value of the Items property.

Pascal

```
function GetItem(I: Integer): TAbstractWebRequestFile; override;
```

C++

```
virtual __fastcall TAbstractWebRequestFile GetItem(int I);
```

Description

GetItem is the protected read implementation of the Items property.

I is the index of the Web request file object to return.

In TAbstractWebRequestFile, GetCount is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Count property.

126.1.3 TWebRequestFileStream Class

Class Hierarchy

Pascal

```
TWebRequestFileStream = class(TCustomMemoryStream);
```

C++

```
class TWebRequestFileStream : public TCustomMemoryStream;
```

File

ReqFiles

Description

This is class ReqFiles.TWebRequestFileStream.

126.1.3.1 TWebRequestFileStream Members

The following tables list the members exposed by TWebRequestFileStream.

Protected Methods

	Name	Description
	GetSize (see page 2982)	This is GetSize, a member of class TStream.
	SetPointer (see page 2823)	Replaces the memory buffer associated with the memory stream.
	SetSize (see page 2989)	This is the overview for the SetSize method overload.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CopyFrom (see page 2979)	Copies a specified number of bytes from one stream to another.
	Create (see page 18706)	This is Create, a member of class TWebRequestFileStream.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FixupResourceHeader (see page 2981)	Patches the resource header for a resource that has been written to the stream.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Read (see page 2817)	Reads up to Count bytes from the memory stream into Buffer and advances the current position of the stream by the number of bytes read.
	Read (see page 2982)	Introduces an abstract or pure virtual method responsible for reading from the stream.
	ReadBuffer (see page 2983)	Reads bytes from the stream into Buffer.
	ReadComponent (see page 2983)	Initiates streaming of components and their properties.
	ReadComponentRes (see page 2987)	Reads components and their properties from a stream in a resource file format.

	ReadResHeader (see page 2987)	Reads a resource-file header from the stream.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 2818)	Writes the entire contents of the memory stream to the file with a given file name.
	SaveToStream (see page 2818)	Writes the entire contents of the memory stream to the stream object specified by Stream.
V	Seek (see page 2819)	Moves the current position of the stream by Offset bytes, relative to the origin specified by Origin.
V	Seek (see page 2988)	This is the overview for the Seek method overload.
V	Write (see page 18706)	Introduces a pure virtual method for writing to the stream.
	WriteBuffer (see page 2990)	Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.
	WriteComponent (see page 2990)	Initiates the writing of components and their properties to a stream.
	WriteComponentRes (see page 2994)	Writes components and their properties to a stream from a resource file format.
	WriteDescendent (see page 2994)	Streams components and their properties in inherited forms.
	WriteDescendentRes (see page 2995)	Streams components and their properties in inherited forms.
	WriteResourceHeader (see page 2995)	Writes a resource-file header to the stream.

Public Properties

	Name	Description
R	Memory (see page 2823)	Provides direct access to the memory pool allocated for the memory stream.
	Position (see page 2996)	Indicates the current offset into the stream for reading and writing.
	Size (see page 2997)	Indicates the size in bytes of the stream.

126.1.3.2 TWebRequestFileStream.Create Constructor

Pascal

```
constructor Create(ABuffer: Pointer; Size: Integer);
```

C++

```
__fastcall TWebRequestFileStream(void * ABuffer, int Size);
```

Description

This is Create, a member of class TWebRequestFileStream.

126.1.3.3 TWebRequestFileStream Methods

The methods of the TWebRequestFileStream class are listed here.

Public Methods

	Name	Description
V	Write (see page 18706)	Introduces a pure virtual method for writing to the stream.

126.1.3.3.1 TWebRequestFileStream.Write Method

Introduces a pure virtual method for writing to the stream.

Pascal

```
function Write(const Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  Buffer, Longint Count);
```

Description

Each descendant stream object defines a Write method that writes data to its particular storage medium (such as memory or a disk file). Write attempts to write up to Count bytes from Buffer, and returns the number of bytes actually written.

All the other data-writing methods of a stream (WriteBuffer, WriteComponent) call Write to do their actual writing.

See Also

WriteBuffer (see page 2990)

WriteComponent (see page 2990)

127 ReqMulti Namespace

This is namespace ReqMulti.

Topics

Name	Description
Classes (see page 18710)	The following table lists classes in this documentation.

Classes

	Name	Description
	TMultipartContentParser (see page 18710)	TMultipartContentParser extracts named value strings and uploaded files from an HTTP request sent by a multipart form.

127.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TMultipartContentParser (see page 18710)	TMultipartContentParser extracts named value strings and uploaded files from an HTTP request sent by a multipart form.

127.1.1 TMultipartContentParser Class

TMultipartContentParser extracts named value strings and uploaded files from an HTTP request sent by a multipart form.

Class Hierarchy

Pascal

```
TMultipartContentParser = class(TAbstractContentParser);
```

C++

```
class TMultipartContentParser : public TAbstractContentParser;
```

File

ReqMulti

Description

Web request objects use TMultipartContentParser object to parse the content of an HTTP request message from a multipart form when that request may contain uploaded files. The Web request uses this content parser to assign values for its ContentFields and Files properties.

TMultipartContentParser is only used when the current HTTP request object has a content type of 'multipart/form-data'. Multipart forms built using a WebSnap application automatically use this content type.

To allow your WebSnap application to use TMultipartContentParser in Delphi, add the ReqMulti (see page 18709) unit to the end of your project uses clause. In C++, include ReqMulti.hpp in the header of your project source file.

See Also

TWebRequest (see page 15161)

TWebRequest (see page 15161)

TWebRequest (see page 15161)

TContentParser (see page 15088)

TWebRequest (see page 15161)

127.1.1.1 TMultipartContentParser Members

The following tables list the members exposed by TMultipartContentParser.

Protected Methods

	Name	Description
	BufferContent (see page 18712)	This is BufferContent, a member of class TMultipartContentParser.
	ExtractContentTypeFields (see page 18713)	This is ExtractContentTypeFields, a member of class TMultipartContentParser.
	ExtractMultipartContentFields (see page 18713)	This is ExtractMultipartContentFields, a member of class TMultipartContentParser.
V	GetContentFields (see page 18714)	Returns the named value strings that are included in the content of the current Web request.
V	GetFiles (see page 18714)	Returns a set of uploaded files that the parser extracts from the current Web request.
	ParseMultiPart (see page 18715)	This is ParseMultiPart, a member of class TMultipartContentParser.
	ParseMultiPartContent (see page 18715)	This is ParseMultiPartContent, a member of class TMultipartContentParser.
	ParseMultipartHeaders (see page 18715)	This is ParseMultipartHeaders, a member of class TMultipartContentParser.

Protected Properties

	Name	Description
R	WebRequest (see page 15043)	Indicates the Web request object that uses this parser to parse its content.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	CanParse (see page 18713)	Indicates whether a specified request message contains information encoded in a way that this parser understands.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 15041)	Creates and initializes a new TAbstractContentParser (see page 15039) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 18712)	Frees the memory associated with the TMultipartContentParser (see page 18710) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

127.1.1.2 TMultipartContentParser.Destroy Destructor

Frees the memory associated with the TMultipartContentParser (see page 18710) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMultipartContentParser();
```

Description

Do not free content parser instances. The Web request object frees the content parser when it no longer needs it.

127.1.1.3 TMultipartContentParser Methods

The methods of the TMultipartContentParser class are listed here.

Protected Methods

	Name	Description
	BufferContent (see page 18712)	This is BufferContent, a member of class TMultipartContentParser.
	ExtractContentTypeFields (see page 18713)	This is ExtractContentTypeFields, a member of class TMultipartContentParser.
	ExtractMultipartContentFields (see page 18713)	This is ExtractMultipartContentFields, a member of class TMultipartContentParser.
V	GetContentFields (see page 18714)	Returns the named value strings that are included in the content of the current Web request.
V	GetFiles (see page 18714)	Returns a set of uploaded files that the parser extracts from the current Web request.
	ParseMultiPart (see page 18715)	This is ParseMultiPart, a member of class TMultipartContentParser.
	ParseMultiPartContent (see page 18715)	This is ParseMultiPartContent, a member of class TMultipartContentParser.
	ParseMultipartHeaders (see page 18715)	This is ParseMultipartHeaders, a member of class TMultipartContentParser.

Public Methods

	Name	Description
V	CanParse (see page 18713)	Indicates whether a specified request message contains information encoded in a way that this parser understands.

127.1.1.3.1 TMultipartContentParser.BufferContent Method

Pascal

```
procedure BufferContent;
```

C++

```
__fastcall BufferContent();
```

Description

This is BufferContent, a member of class TMultipartContentParser.

127.1.1.3.2 TMultipartContentParser.CanParse Method

Indicates whether a specified request message contains information encoded in a way that this parser understands.

Pascal

```
class function CanParse(AWebRequest: TWebRequest): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanParse(TWebRequest AWebRequest);
```

Description

CanParse indicates whether a content parser class can handle parsing a specified request message. CanParse is a class method (Delphi) or static method (C++), which means the Web request does not need an instantiated object to call this method. The Web request object calls the CanParse method of every registered content parser until it finds one that can parse its content.

AWebRequest is the current HTTP request message.

CanParse returns true if this class can parse AWebRequest, false if AWebRequest does not indicate an encoding that this class understands.

In TMultipartContentParser (see page 18710), CanParse returns true if AWebRequest has a content type of 'multipart/form-data'. If the Web request indicates some other content type, the Web request uses a different content parser class such as TContentParser.

See Also

ContentType (see page 15179)

TContentParser (see page 15088)

127.1.1.3.3 TMultipartContentParser.ExtractContentTypeFields Method

Pascal

```
procedure ExtractContentTypeFields(Strings: TStrings);
```

C++

```
__fastcall ExtractContentTypeFields(TStrings Strings);
```

Description

This is ExtractContentTypeFields, a member of class TMultipartContentParser.

127.1.1.3.4 TMultipartContentParser.ExtractMultipartContentFields Method

Pascal

```
procedure ExtractMultipartContentFields;
```

C++

```
__fastcall ExtractMultipartContentFields();
```

Description

This is ExtractMultipartContentFields, a member of class TMultipartContentParser.

127.1.1.3.5 TMultipartContentParser.GetContentFields Method

Returns the named value strings that are included in the content of the current Web request.

Pascal

```
function GetContentFields: TStrings; override;
```

C++

```
virtual __fastcall TStrings GetContentFields();
```

Description

GetContentFields returns a list of strings with the form "Name=Value" that represent any name/value pairs that the parser identifies in the content of the current Web request.

In TAbstractContentParser, GetContentFields is abstract or, in C++ terminology, pure virtual, meaning that this class does not supply an implementation. Descendant classes override this method to extract name/value pairs from the current request and add them to the returned string list.

Note: The content parser owns the string list it returns. The caller should not free the string list.

See Also

ContentFields (see page 15177)

WebRequest (see page 15043)

ExtractContentFields (see page 15166)

127.1.1.3.6 TMultipartContentParser.GetFiles Method

Returns a set of uploaded files that the parser extracts from the current Web request.

Pascal

```
function GetFiles: TAbstractWebRequestFiles; override;
```

C++

```
virtual __fastcall TAbstractWebRequestFiles GetFiles();
```

Description

GetFiles returns a list of Web request file objects, each of which represents an uploaded file from the current Web request. Each Web request file object has properties to describe an uploaded file, including a stream object from which to read the content of the uploaded file.

In TAbstractContentParser, GetFiles is abstract or, in C++ terminology, pure virtual, meaning that this class does not supply an implementation. Descendant classes override this method to extract uploaded files from the current request and add them to the Web request files object.

Note: The content parser owns the list it returns. The caller should not free this list.

See Also

Files (see page 15182)

WebRequest (see page 15043)

TAbstractWebRequestFile (see page 15073)

127.1.1.3.7 TMultipartContentParser.ParseMultiPart Method

Pascal

```
procedure ParseMultiPart(Part: Pointer; Size: Integer);
```

C++

```
__fastcall ParseMultiPart(void * Part, int Size);
```

Description

This is ParseMultiPart, a member of class TMultipartContentParser.

127.1.1.3.8 TMultipartContentParser.ParseMultiPartContent Method

Pascal

```
procedure ParseMultiPartContent;
```

C++

```
__fastcall ParseMultiPartContent();
```

Description

This is ParseMultiPartContent, a member of class TMultipartContentParser.

127.1.1.3.9 TMultipartContentParser.ParseMultipartHeaders Method

Pascal

```
procedure ParseMultipartHeaders(Parser: THTTPParser; AContent: Pointer; AContentLength:
Integer);
```

C++

```
__fastcall ParseMultipartHeaders(THTTPParser Parser, void * AContent, int AContentLength);
```

Description

This is ParseMultipartHeaders, a member of class TMultipartContentParser.

128 Rio Namespace

This is namespace Rio.

Topics

Name	Description
Classes (see page 18718)	The following table lists classes in this documentation.
Interfaces (see page 18733)	The following table lists interfaces in this documentation.
Types (see page 18735)	The following table lists types in this documentation.
Constants (see page 18736)	The following table lists constants in this documentation.

Classes

	Name	Description
	TRIO (see page 18718)	TRIO is the base class for components that generate calls to remote interfaced objects.

Constants

Name	Description
StubSize (see page 18736)	This is constant Rio.StubSize.

Interfaces

	Name	Description
	IRIOAccess (see page 18733)	IRIOAccess is the interface for accessing a remote interfaced object from its invokable interface.

Types

Name	Description
TAfterExecuteEvent (see page 18735)	This is type Rio.TAfterExecuteEvent.
TBeforeExecuteEvent (see page 18735)	This is type Rio.TBeforeExecuteEvent.

128.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TRIO (see page 18718)	TRIO is the base class for components that generate calls to remote interfaced objects.

128.1.1 TRIO Class

TRIO is the base class for components that generate calls to remote interfaced objects.

Class Hierarchy

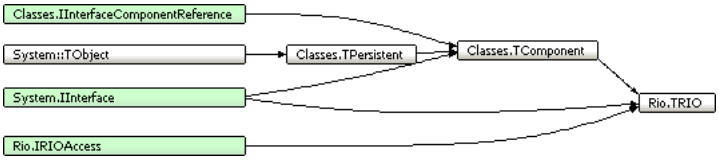

Pascal

```
TRIO = class(TComponent, IInterface, IRIOAccess);
```

C++

```
class TRIO : public TComponent, public IInterface, public IRIOAccess;
```

File

Rio

Description

Use TRIO as a base class for components that generate statically-linked calls to invokable interfaces. When an application casts a TRIO descendant to a registered invokable interface, it dynamically generates an in-memory method table, providing an implementation to that invokable interface. TRIO uses two helper objects when generating this implementation. These helper objects implement the IOPConvert and IWebNode interfaces. IOPConvert handles the marshaling and unmarshaling of calls to methods on the generated method table, and IWebNode receives the marshaled message call, executes it (typically by sending it to a remote server application), and returns the results of executing the call. These interfaces do not assume any particular encoding or transport protocol. Each TRIO descendant uses its own protocol-specific helper objects to implement these interfaces.

Before TRIO can generate a method table for an invokable interface, the interface must be registered with the invocation registry. To register an invokable interface, use the global InvRegistry function to obtain access to the registry, and call its RegisterInterface method.

Do not create instances of TRIO. To add a component that generates calls to a registered invokable interface, use a TRIO descendant such as THTTPRIO.

See Also

THTTPRio

InvRegistry

TInvokableClassRegistry (see page 15526)

IWebNode (see page 25430)

IOPConvert (see page 18122)

IRIOAccess (see page 18733)

128.1.1.1 TRIO Members

The following tables list the members exposed by TRIO.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FConverter (see page 18723)	This is FConverter, a member of class TRIO.
	FIID (see page 18723)	This is FIID, a member of class TRIO.
	FWebNode (see page 18723)	This is FWebNode, a member of class TRIO.
	IntfMD (see page 18723)	This is IntfMD, a member of class TRIO.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecute (see page 18725)	Generates an OnAfterExecute (see page 18730) event.
V	DoBeforeExecute (see page 18725)	Generates an OnBeforeExecute (see page 18730) event.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetResponseStream (see page 18726)	Returns a newly created stream that receives the results of executing a method call.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 18727)	Implements the IInterface QueryInterface method.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 18724)	Updates the reference count after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 18725)	Prevents the destruction of a TRIO (see page 18718) instance if the reference count is greater than zero.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 18722)	Creates an instance of TRIO (see page 18718).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 18722)	Destroys the instance of a TRIO (see page 18718) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.

	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetRIO (see page 18734)	Returns the value of the RIO (see page 18734) property.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 18726)	Allocates memory for each instance of a remote interfaced object.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Converter (see page 18728)	Handles the marshaling and unmarshaling of calls to an invokable interface.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	RefCount (see page 18728)	Maintains a reference count on the use of the TRIO (see page 18718) object's interface.
R	RIO (see page 18734)	Provides access to the remote interfaced object that implements this interface.
R	SOAPHeaders (see page 18728)	Provides access to the headers of SOAP messages that the remote interface object sends or receives.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	WebNode (see page 18729)	Handles the execution of method calls on the invokable interface.

Published Events

	Name	Description
	OnAfterExecute (see page 18730)	Occurs when TRIO (see page 18718) receives the results from executing a method of the invokable interface.
	OnBeforeExecute (see page 18730)	Occurs before TRIO (see page 18718) sends an encoded method call for execution.
	OnGetAttachment (see page 18731)	Occurs when TRIO (see page 18718) reads each attachment included in the results from executing a method of the invokable interface.
	OnSendAttachment (see page 18731)	Occurs when TRIO (see page 18718) sends each attachment that is the value of an argument of the method call on the invokable interface.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

128.1.1.2 TRIO.Create Constructor

Creates an instance of TRIO (see page 18718).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TRIO(TComponent * AOwner);
```

Description

Call Create to instantiate a TRIO (see page 18718) descendant at runtime. TRIO (see page 18718) descendants added to forms or data modules at design time are created automatically.

The AOwner parameter specifies another component (usually a form or data module) that is responsible for freeing the TRIO (see page 18718) instance. It becomes the value of the Owner property. If the TRIO (see page 18718) instance is created with AOwner set to nil (Delphi) or NULL (C++), then it is automatically freed when its reference count drops to zero.

See Also

RefCount (see page 18728)

Owner (see page 2807)

128.1.1.3 TRIO.Destroy Destructor

Destroys the instance of a TRIO (see page 18718) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TRIO();
```

Description

There is no need to explicitly free a TRIO (see page 18718) instance.

If the TRIO (see page 18718) object is created with an Owner, then that Owner handles the freeing of the TRIO (see page 18718) instance. If the TRIO (see page 18718) object is created without an Owner, then it is automatically freed when the reference count on its interface drops to zero.

See Also

Create (see page 18722)

Owner (see page 2807)

RefCount (see page 18728)

128.1.1.4 TRIO Fields

The fields of the TRIO class are listed here.

Protected Fields

	Name	Description
	FConverter (see page 18723)	This is FConverter, a member of class TRIO.
	FIID (see page 18723)	This is FIID, a member of class TRIO.
	FWebNode (see page 18723)	This is FWebNode, a member of class TRIO.
	IntfMD (see page 18723)	This is IntfMD, a member of class TRIO.

128.1.1.4.1 TRIO.FConverter Field

Pascal

```
FConverter: IOPConvert;
```

C++

```
IOPConvert FConverter;
```

Description

This is FConverter, a member of class TRIO.

128.1.1.4.2 TRIO.FIID Field

Pascal

```
FIID: TGUID;
```

C++

```
TGUID FIID;
```

Description

This is FIID, a member of class TRIO.

128.1.1.4.3 TRIO.FWebNode Field

Pascal

```
FWebNode: IWebNode;
```

C++

```
IWebNode FWebNode;
```

Description

This is FWebNode, a member of class TRIO.

128.1.1.4.4 TRIO.IntfMD Field

Pascal

```
IntfMD: TIntfMetaData;
```

C++

```
TIntfMetaData IntfMD;
```

Description

This is IntfMD, a member of class TRIO.

128.1.1.5 TRIO Methods

The methods of the TRIO class are listed here.

Protected Methods

	Name	Description
	DoAfterExecute (see page 18725)	Generates an OnAfterExecute (see page 18730) event.
	DoBeforeExecute (see page 18725)	Generates an OnBeforeExecute (see page 18730) event.
	GetResponseStream (see page 18726)	Returns a newly created stream that receives the results of executing a method call.
	QueryInterface (see page 18727)	Implements the IInterface QueryInterface method.

Public Methods

	Name	Description
	AfterConstruction (see page 18724)	Updates the reference count after the last constructor has executed.
	BeforeDestruction (see page 18725)	Prevents the destruction of a TRIO (see page 18718) instance if the reference count is greater than zero.
	NewInstance (see page 18726)	Allocates memory for each instance of a remote interfaced object.

128.1.1.5.1 TRIO.AfterConstruction Method

Updates the reference count after the last constructor has executed.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called automatically after the TRIO (see page 18718) object's last constructor has executed. Do not call it explicitly in your applications.

TRIO (see page 18718) overrides the AfterConstruction method to update the RefCount (see page 18728) property. TRIO (see page 18718) artificially increments the reference count when it is first created to prevent changes to the reference count from causing the TRIO (see page 18718) instance to be freed before it is fully instantiated. AfterConstruction returns the reference count back to an appropriate level, now that the constructor is finished.

See Also

RefCount (see page 18728)

Create (see page 18722)

NewInstance (see page 18726)

128.1.1.5.2 TRIO.BeforeDestruction Method

Prevents the destruction of a TRIO (see page 18718) instance if the reference count is greater than zero.

Pascal

```
procedure BeforeDestruction; override;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

BeforeDestruction is called automatically before the TRIO (see page 18718) object's first destructor executes. Do not call it explicitly in your applications.

TRIO (see page 18718) overrides the BeforeDestruction method to check the RefCount (see page 18728) property. If RefCount (see page 18728) is greater than zero, then there is still an outstanding reference to the TRIO (see page 18718) object's interface,. and BeforeDestruction raises an exception to prevent the destruction of this object.

See Also

RefCount (see page 18728)

128.1.1.5.3 TRIO.DoAfterExecute Method

Generates an OnAfterExecute (see page 18730) event.

Pascal

```
procedure DoAfterExecute(const MethodName: string; Response: TStream); virtual;
```

C++

```
virtual __fastcall DoAfterExecute(const AnsiString MethodName, TStream Response);
```

Description

TRIO (see page 18718) calls DoAfterExecute internally after it receives the results of a method call but before it unmarshals those results. This method simply generates an OnAfterExecute (see page 18730) event, passing the MethodName and Response parameters to the event handler. Override this method to perform class-specific actions before or after the OnAfterExecute (see page 18730) event.

MethodName is the name of the method that was just executed on the server.

Response is a stream from that contains the results that the server returned after executing the specified method.

See Also

OnAfterExecute (see page 18730)

128.1.1.5.4 TRIO.DoBeforeExecute Method

Generates an OnBeforeExecute (see page 18730) event.

Pascal

```
procedure DoBeforeExecute(const MethodName: string; Request: TStream); virtual;
```

C++

```
virtual __fastcall DoBeforeExecute(const AnsiString MethodName, TStream Request);
```

Description

TRIO (see page 18718) calls DoBeforeExecute internally before it sends a SOAP packet to the server for execution. This method simply generates an OnBeforeExecute (see page 18730) event, passing the MethodName parameter and the string in the Request parameter to the event handler. Override this method to perform class-specific actions before or after the OnBeforeExecute (see page 18730) event.

MethodName is the name of the method that the encoded SOAP request calls.

Request is a stream from which the encoded SOAP request packet can be read.

See Also

OnBeforeExecute (see page 18730)

128.1.1.5.5 TRIO.GetResponseStream Method

Returns a newly created stream that receives the results of executing a method call.

Pascal

```
function GetResponseStream(BindingType: TWebServiceBindingType): TStream; virtual;
```

C++

```
virtual __fastcall TStream GetResponseStream(TWebServiceBindingType BindingType);
```

Description

TRIO (see page 18718) calls GetResponseStream internally when it executes a method call. It passes the returned stream as the Response parameter of the Execute method on the interface that the WebNode (see page 18729) property specifies. After this stream receives the encoded response of a method call, the RIO (see page 18734) component decodes the response and frees the stream.

BindingType indicates the expected coding of the response that the stream will hold. GetResponseStream can use this information to determine what type of stream to create.

See Also

Execute (see page 25432)

WebNode (see page 18729)

128.1.1.5.6 TRIO.NewInstance Method

Allocates memory for each instance of a remote interfaced object.

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
virtual __fastcall TObject * NewInstance();
```

Description

All constructors call NewInstance automatically. NewInstance calls InstanceSize to determine how much memory to allocate from the heap to contain a particular instance. Do not call NewInstance directly.

TRIO (see page 18718) overrides NewInstance to increment the reference count when a new instance is created. This prevents any constructor from accidentally deleting the instance due to a RefCount (see page 18728) of zero. The AfterConstruction (see page 18724) method decrements the value of RefCount (see page 18728) after all constructors have

executed.

See Also

InstanceSize (see page 22306)

NewInstance (see page 22306)

AfterConstruction (see page 18724)

128.1.1.5.7 TRIO.QueryInterface Method

Implements the IInterface QueryInterface method.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface is called through the IInterface interface to obtain an interface pointer for the interface identified by the IID parameter. In Delphi, this call is generated when an application casts TRIO (see page 18718) to an invokable interface using the as operator.

TRIO (see page 18718) can only return a single type of interface pointer. The first time an application calls QueryInterface, TRIO (see page 18718) generates an in-memory method table to support calls to the methods on the requested interface.

IID specifies the invokable interface that the application wants to call using this TRIO (see page 18718) instance. If the TRIO (see page 18718) object has already been typecast to an interface, then IID must identify that interface. If the TRIO (see page 18718) instance has not yet been typecast to an interface, then IID must be a registered invokable interface.

Obj returns an interface pointer for the specified interface, unless TRIO (see page 18718) was already implementing a different interface or IID identified an interface that was not registered.

QueryInterface returns 0 (S_OK) if Obj successfully returns an interface pointer. It returns E_NOINTERFACE otherwise.

If QueryInterface successfully returns an interface pointer, it automatically increments the reference count.

See Also

InvRegistry (see page 15620)

RegisterInterface (see page 15552)

RefCount (see page 18728)

128.1.1.6 TRIO Properties

The properties of the TRIO class are listed here.

Public Properties

	Name	Description
	Converter (see page 18728)	Handles the marshaling and unmarshaling of calls to an invokable interface.
R	RefCount (see page 18728)	Maintains a reference count on the use of the TRIO (see page 18718) object's interface.
R	SOAPHeaders (see page 18728)	Provides access to the headers of SOAP messages that the remote interface object sends or receives.
	WebNode (see page 18729)	Handles the execution of method calls on the invokable interface.

128.1.1.6.1 TRIO.Converter Property

Handles the marshaling and unmarshaling of calls to an invokable interface.

Pascal

```
property Converter: IOPConvert;
```

C++

```
__property IOPConvert Converter;
```

Description

TRIO (see page 18718) uses Converter internally to marshal and unmarshal calls to an invokable interface. That is, when the TRIO (see page 18718) object has been typecast to an invokable interface, it implements each method call by using Converter to marshal the call into an encoding protocol, and to unmarshal any return values after that call is executed.

Each TRIO (see page 18718) descendant uses its own protocol-specific helper object as the value of the Converter property.

See Also

WebNode (see page 18729)

TOPToSoapDomConvert (see page 18137)

128.1.1.6.2 TRIO.RefCount Property

Maintains a reference count on the use of the TRIO (see page 18718) object's interface.

Pascal

```
property RefCount: Integer;
```

C++

```
__property int RefCount;
```

Description

RefCount keeps track of the usage on the TRIO (see page 18718) object's interface. RefCount is incremented by the interface's _AddRef method (Delphi) or AddRef method (C++) and decremented by its _Release method (Delphi) or Release method (C++).

If the TRIO (see page 18718) object is created without an Owner, then when RefCount drops to zero, the TRIO (see page 18718) instance is automatically freed.

See Also

QueryInterface (see page 18727)

Create (see page 18722)

_AddRef (see page 22397)

_Release (see page 22397)

128.1.1.6.3 TRIO.SOAPHeaders Property

Provides access to the headers of SOAP messages that the remote interface object sends or receives.

Pascal

```
property SOAPHeaders: TSOAPHeaders;
```

C++

```
__property TSOAPHeaders SOAPHeaders;
```

Description

SOAPHeaders is a helper object that lets TRIO (see page 18718) support the ISOAPHeaders interface. It has methods to add headers to an outgoing request message and to retrieve the headers from the resulting response message. You can use the SOAPHeaders property directly, or access it through an ISOAPHeaders interface, which is obtained from the invokable interface using the as operator (Delphi) or the Supports method (C++).

Note: Although TRIO (see page 18718) uses the SOAPHeaders property to implement an ISOAPHeaders interface, you can't obtain an ISOAPHeaders interface directly from TRIO (see page 18718). Rather, you must obtain this interface from an invokable interface.

See Also

ISOAPHeaders (see page 15613)

128.1.1.6.4 TRIO.WebNode Property

Handles the execution of method calls on the invokable interface.

Pascal

```
property WebNode: IWebNode;
```

C++

```
__property IWebNode WebNode;
```

Description

TRIO (see page 18718) uses WebNode internally to execute method calls after they have been marshaled by Converter (see page 18728). That is, when the TRIO (see page 18718) object has been typecast to an invokable interface, it implements each method call by using Converter (see page 18728) to marshal the call into an encoding protocol, and then uses WebNode to execute the encoded call.

Each TRIO (see page 18718) descendant uses its own protocol-specific helper object as the value of the WebNode property.

See Also

Converter (see page 18728)

THTTPReqResp (see page 19961)

OnAfterExecute (see page 18730)

OnBeforeExecute (see page 18730)

128.1.1.7 TRIO Events

The events of the TRIO class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 18730)	Occurs when TRIO (see page 18718) receives the results from executing a method of the invokable interface.
	OnBeforeExecute (see page 18730)	Occurs before TRIO (see page 18718) sends an encoded method call for execution.

	OnGetAttachment (see page 18731)	Occurs when TRIO (see page 18718) reads each attachment included in the results from executing a method of the invokable interface.
	OnSendAttachment (see page 18731)	Occurs when TRIO (see page 18718) sends each attachment that is the value of an argument of the method call on the invokable interface.

128.1.1.7.1 TRIO.OnAfterExecute Event

Occurs when TRIO (see page 18718) receives the results from executing a method of the invokable interface.

Pascal

```
property OnAfterExecute: TAfterExecuteEvent;
```

C++

```
__property TAfterExecuteEvent OnAfterExecute;
```

Description

Write an OnAfterExecute event handler to examine or alter the encoded results from executing a method call. OnAfterExecute occurs after TRIO (see page 18718) calls the Execute method of the interface that the WebNode (see page 18729) property specifies, but before the interface that the Converter (see page 18728) property specifies unmarshals those results.

You can use the SOAPResponse stream of the event handler to alter the encoded results from the server before they are unmarshaled. This lets you compensate for known idiosyncrasies of a server or of the Converter (see page 18728) that is the value of the Converter (see page 18728) property. For example, if the Web Service provider publishes interoperability information, you can use this event to adjust SOAP packets to compensate for differences that the Web Service provider reports.

See Also

OnBeforeExecute (see page 18730)

Execute (see page 25432)

WebNode (see page 18729)

Converter (see page 18728)

OnGetAttachment (see page 18731)

128.1.1.7.2 TRIO.OnBeforeExecute Event

Occurs before TRIO (see page 18718) sends an encoded method call for execution.

Pascal

```
property OnBeforeExecute: TBeforeExecuteEvent;
```

C++

```
__property TBeforeExecuteEvent OnBeforeExecute;
```

Description

Write an OnBeforeExecute event handler to examine or alter an encoded method call before it is sent to the server. OnBeforeExecute occurs after the interface that the Converter (see page 18728) property specifies marshals a method call into a SOAP request packet, but before TRIO (see page 18718) calls the Execute method of the interface that the WebNode (see page 18729) property specifies.

You can use the SoapRequest parameter to read or alter the SOAP request before it is passed to the server. For example, if the Web Service provider publishes interoperability information, you can use this parameter to adjust SOAP packets to compensate for differences that the Web Service provider reports.

See Also

OnAfterExecute (see page 18730)

Execute (see page 25432)

WebNode (see page 18729)

Converter (see page 18728)

OnSendAttachment (see page 18731)

128.1.1.7.3 TRIO.OnGetAttachment Event

Occurs when TRIO (see page 18718) reads each attachment included in the results from executing a method of the invokable interface.

Pascal

```
property OnGetAttachment: TOnGetAttachmentEvent;
```

C++

```
__property TOnGetAttachmentEvent OnGetAttachment;
```

Description

Write an OnGetAttachment event handler to examine the encoded attachments included in the results from executing a method call or to redirect them so that they are saved to a specific file. OnGetAttachment occurs once for each attachment received in the results from a call to the Execute method of the interface that the WebNode (see page 18729) property specifies. All OnGetAttachment events occur before the OnAfterExecute (see page 18730) event.

See Also

OnAfterExecute (see page 18730)

Execute (see page 25432)

WebNode (see page 18729)

128.1.1.7.4 TRIO.OnSendAttachment Event

Occurs when TRIO (see page 18718) sends each attachment that is the value of an argument of the method call on the invokable interface.

Pascal

```
property OnSendAttachment: TOnSendAttachmentEvent;
```

C++

```
__property TOnSendAttachmentEvent OnSendAttachment;
```

Description

Write an OnSendAttachment event handler to examine or alter the attachments that are to be included in a request message that represents a method call. OnSendAttachment occurs once for each attachment sent with the request message to the Web Service provider. All OnSendAttachment events occur after the OnBeforeExecute (see page 18730) event.

See Also

OnBeforeExecute (see page 18730)

Execute (see page 25432)

WebNode (see page 18729)

128.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IRIOAccess (see page 18733)	IRIOAccess is the interface for accessing a remote interfaced object from its invokable interface.

128.2.1 IRIOAccess Interface

IRIOAccess is the interface for accessing a remote interfaced object from its invokable interface.

Class Hierarchy

Rio.IRIOAccess

Pascal

```
IRIOAccess = interface;
```

C++

```
__interface IRIOAccess;
```

File

Rio

Description

Remote interfaced objects (TRIO (see page 18718) and its descendants) implement IRIOAccess so that you can access the TRIO (see page 18718) instance given an invokable interface that it implements. This interface defines a single property, RIO (see page 18734), which returns the TRIO (see page 18718) instance that implements it.

Note: C++ method declarations that use IRIOAccess use the _di_IRIOAccess type instead. This type is a DelphiInterface wrapper around the IRIOAccess interface:

Note: typedef System::DelphiInterface (see page 22167)< IRIOAccess > _di_IRIOAccess;

See Also

TRIO (see page 18718)

128.2.1.1 IRIOAccess Members

The following tables list the members exposed by IRIOAccess.

Public Methods

	Name	Description
	GetRIO (see page 18734)	Returns the value of the RIO (see page 18734) property.

Public Properties

	Name	Description
	RIO (see page 18734)	Provides access to the remote interfaced object that implements this interface.

128.2.1.2 IRIOAccess Methods

The methods of the IRIOAccess class are listed here.

Public Methods

	Name	Description
	GetRIO (see page 18734)	Returns the value of the RIO (see page 18734) property.

128.2.1.2.1 IRIOAccess.GetRIO Method

Returns the value of the RIO (see page 18734) property.

Pascal

```
function GetRIO: TRIO;
```

C++

```
__fastcall TRIO GetRIO();
```

Description

GetRIO is the read implementation for the RIO (see page 18734) property.

See Also

RIO (see page 18734)

128.2.1.3 IRIOAccess Properties

The properties of the IRIOAccess class are listed here.

Public Properties

	Name	Description
	RIO (see page 18734)	Provides access to the remote interfaced object that implements this interface.

128.2.1.3.1 IRIOAccess.RIO Property

Provides access to the remote interfaced object that implements this interface.

Pascal

```
property RIO: TRIO;
```

C++

```
__property TRIO RIO;
```

Description

Call GetRIO (see page 18734) to access the underlying remote interfaced object. This method is useful when the client of a Web Service uses the automatically-generated function to obtain an invokable interface rather than explicitly creating a TRIO (see page 18718) instance.

128.3 Types

The following table lists types in this documentation.

Types

Name	Description
TAfterExecuteEvent (see page 18735)	This is type Rio.TAfterExecuteEvent.
TBeforeExecuteEvent (see page 18735)	This is type Rio.TBeforeExecuteEvent.

128.3.1 Rio.TAfterExecuteEvent Type

Pascal

```
TAfterExecuteEvent = procedure (const MethodName: string; SOAPResponse: TStream) of object;
```

C++

```
(const MethodName: string; SOAPResponse: TStream) ( TAfterExecuteEvent)();
```

File

Rio

Description

This is type Rio.TAfterExecuteEvent.

128.3.2 Rio.TBeforeExecuteEvent Type

Pascal

```
TBeforeExecuteEvent = procedure (const MethodName: string; var SOAPRequest: InvString) of
object;
```

C++

```
(const MethodName: string; var SOAPRequest: InvString) ( TBeforeExecuteEvent)();
```

File

Rio

Description

This is type Rio.TBeforeExecuteEvent.

128.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
StubSize (see page 18736)	This is constant Rio.StubSize.

128.4.1 Rio.StubSize Constant

Pascal

```
StubSize = 19;
```

C++

```
StubSize = 19;
```

File

Rio

Description

This is constant Rio.StubSize.

129 ScktComp Namespace

This is namespace ScktComp.

Topics

Name	Description
Classes (see page 18739)	The following table lists classes in this documentation.
Functions (see page 18883)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 18884)	The following table lists structs, records, enums in this documentation.
Types (see page 18889)	The following table lists types in this documentation.
Constants (see page 18893)	The following table lists constants in this documentation.

Classes

	Name	Description
	ESocketError (see page 18739)	ESocketError is the exception class for failures that occur when trying to create, use, or shut down Windows socket objects.
	TAbstractSocket (see page 18740)	TAbstractSocket is the abstract base class for all socket components.
	TClientSocket (see page 18752)	TClientSocket manages socket connections for a TCP/IP client.
	TClientWinSocket (see page 18766)	TClientWinSocket describes the client endpoint of a Windows socket connection.
	TCustomServerSocket (see page 18770)	TCustomServerSocket is the base class for all server socket components.
	TCustomSocket (see page 18789)	TCustomSocket is the base class for socket components.
	TCustomWinSocket (see page 18799)	TCustomWinSocket is the base class for all Windows socket objects.
	TServerAcceptThread (see page 18825)	This is class ScktComp.TServerAcceptThread.
	TServerClientThread (see page 18831)	TServerClientThread is an execution thread used for a single connection to a client socket accepted by a Windows server socket.
	TServerClientWinSocket (see page 18840)	TServerClientWinSocket describes the server endpoint of a socket connection to a client socket.
	TServerSocket (see page 18843)	TServerSocket manages server socket connections for a TCP/IP server.
	TServerWinSocket (see page 18858)	TServerWinSocket describes the endpoint of a listening server socket connection.
	TWinSocketStream (see page 18876)	TWinSocketStream is a stream that provides services which allow applications to read from or write to socket connections.

Constants

Name	Description
CM_DEFERFREE (see page 18893)	This is constant ScktComp.CM_DEFERFREE.
CM_LOOKUPCOMPLETE (see page 18893)	This is constant ScktComp.CM_LOOKUPCOMPLETE.
CM_SOCKETMESSAGE (see page 18893)	This is constant ScktComp.CM_SOCKETMESSAGE.

Functions

	Name	Description
	SetErrorProc (see page 18883)	Replaces the exception handler for error messages that are received from a Windows socket connection.

Structs, Records, Enums

	Name	Description
	TAsyncStyle (see page 18884)	This is record ScktComp.TAsyncStyle.
	TClientType (see page 18885)	This is record ScktComp.TClientType.
	TCMLookupComplete (see page 18885)	This is record ScktComp.TCMLookupComplete.
	TCMSocketMessage (see page 18886)	This is record ScktComp.TCMSocketMessage.
	TErrorEvent (see page 18886)	This is record ScktComp.TErrorEvent.
	TLookupState (see page 18887)	This is record ScktComp.TLookupState.
	TServerType (see page 18887)	This is record ScktComp.TServerType.
	TSocketEvent (see page 18887)	TSocketEvent identifies the operation that a socket is performing.

Types

Name	Description
TAsyncStyles (see page 18889)	This is type ScktComp.TAsyncStyles.
TGetSocketEvent (see page 18889)	This is type ScktComp.TGetSocketEvent.
TGetThreadEvent (see page 18890)	This is type ScktComp.TGetThreadEvent.
TSocketErrorEvent (see page 18890)	TSocketErrorEvent is the type for event handlers that respond when a Windows socket reports an error.
TSocketErrorProc (see page 18891)	This is type ScktComp.TSocketErrorProc.
TSocketEventEvent (see page 18891)	This is type ScktComp.TSocketEventEvent.
TSocketNotifyEvent (see page 18891)	The type for event handlers which respond to events during the expected course of forming or using a socket connection.
TThreadNotifyEvent (see page 18892)	This is type ScktComp.TThreadNotifyEvent.

129.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ESocketError (see page 18739)	ESocketError is the exception class for failures that occur when trying to create, use, or shut down Windows socket objects.
	TAbstractSocket (see page 18740)	TAbstractSocket is the abstract base class for all socket components.
	TClientSocket (see page 18752)	TClientSocket manages socket connections for a TCP/IP client.
	TClientWinSocket (see page 18766)	TClientWinSocket describes the client endpoint of a Windows socket connection.
	TCustomServerSocket (see page 18770)	TCustomServerSocket is the base class for all server socket components.
	TCustomSocket (see page 18789)	TCustomSocket is the base class for socket components.
	TCustomWinSocket (see page 18799)	TCustomWinSocket is the base class for all Windows socket objects.
	TServerAcceptThread (see page 18825)	This is class SocktComp.TServerAcceptThread.
	TServerClientThread (see page 18831)	TServerClientThread is an execution thread used for a single connection to a client socket accepted by a Windows server socket.
	TServerClientWinSocket (see page 18840)	TServerClientWinSocket describes the server endpoint of a socket connection to a client socket.
	TServerSocket (see page 18843)	TServerSocket manages server socket connections for a TCP/IP server.
	TServerWinSocket (see page 18858)	TServerWinSocket describes the endpoint of a listening server socket connection.
	TWinSocketStream (see page 18876)	TWinSocketStream is a stream that provides services which allow applications to read from or write to socket connections.

129.1.1 ESocketError Class

ESocketError is the exception class for failures that occur when trying to create, use, or shut down Windows socket objects.

Class Hierarchy

Pascal

```
ESocketError = class(Exception);
```

C++

```
class ESocketError : public Exception;
```

File

ScktComp

Description

ESocketError is raised if

A socket cannot be created.

A socket cannot be initialized.

A socket cannot be opened.

An application tries to change the properties of an already open socket.

A failure occurs when reading from or writing to a socket.

A socket cannot be properly closed.

There are two ESocketError classes, one in the ScktComp (see page 18737) unit and one in the Sockets (see page 20039) unit. The two exception classes are identically defined and serve the same function for different categories of socket objects. The two definitions occur simply to ensure that ESocketError is defined for all socket objects.

See Also

TCustomWinSocket (see page 18799)

129.1.2 TAbstractSocket Class

TAbstractSocket is the abstract base class for all socket components.

Class Hierarchy

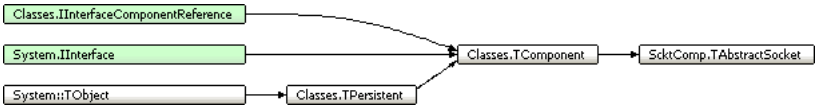

Pascal

```
TAbstractSocket = class(TComponent);
```

C++

```
class TAbstractSocket : public TComponent;
```

File

ScktComp

Description

TAbstractSocket introduces properties and methods to enable an application to work with sockets. A socket encapsulates a set of communication protocols to allow the application to connect to other machines for reading or writing information. Sockets (see page 20039) provide connections based on the TCP/IP protocol. They also allow connections that use the Xerox Network System (see page 22079) (XNS), Digital's DECnet protocol, or Novell's IPX/SPX family. Sockets (see page 20039) allow an application to form connections to other machines without being concerned with the details of the actual networking software.

The properties of TAbstractSocket describe the IP address of the socket and service it provides or seeks. Not all descendants of TAbstractSocket use all of these properties. For example, server sockets do not surface the IP address because it is read implicitly from the system running the application.

Do not create instances of TAbstractSocket. Use a descendant of TAbstractSocket to add the ability to communicate with other machines to an application. To create a socket that initiates the connection with another machine, use TClientSocket (see page 18752). To create a socket that responds to requests for connections from other machines, use TServerSocket (see page 18843).

See Also

TClientSocket (see page 18752)

TCustomWinSocket (see page 18799)

TServerSocket (see page 18843)

TCustomSocket (see page 18789)

129.1.2.1 TAbstractSocket Members

The following tables list the members exposed by TAbstractSocket.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
A	DoActivate (see page 18744)	Provides the prototype of a method to open or close the socket connection.
A	Error (see page 18745)	Provides the prototype of a method that responds to error conditions.
A	Event (see page 18746)	Provides the prototype of a method that responds when specific socket events occur.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InitSocket (see page 18746)	Tells a Windows socket object to call the Error (see page 18745) and Event (see page 18746) methods.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 18747)	Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 18750) property.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActive (see page 18748)	Sets the Active (see page 18750) property.
	SetAddress (see page 18748)	Sets the Address (see page 18750) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHost (see page 18748)	Sets the Host (see page 18751) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetPort (see page 18749)	Sets the Port (see page 18751) property.

	SetService (see page 18749)	Sets the Service (see page 18752) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Active (see page 18750)	Indicates whether the socket connection is open and available for communication with other machines.
	Address (see page 18750)	Specifies the IP address of the server system.
	Host (see page 18751)	Specifies an alias for the IP address of the server system.
	Port (see page 18751)	Specifies the ID number that identifies the server socket connection.
	Service (see page 18752)	Specifies the name of the service for which the socket connection is used.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 18744)	Shuts down the socket connection.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 18747)	Opens the socket connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

129.1.2.2 TAbstractSocket Methods

The methods of the TAbstractSocket class are listed here.

Protected Methods

	Name	Description
A	DoActivate (see page 18744)	Provides the prototype of a method to open or close the socket connection.
A	Error (see page 18745)	Provides the prototype of a method that responds to error conditions.
A	Event (see page 18746)	Provides the prototype of a method that responds when specific socket events occur.
	InitSocket (see page 18746)	Tells a Windows socket object to call the Error (see page 18745) and Event (see page 18746) methods.
V	Loaded (see page 18747)	Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 18750) property.
	SetActive (see page 18748)	Sets the Active (see page 18750) property.
	SetAddress (see page 18748)	Sets the Address (see page 18750) property.
	SetHost (see page 18748)	Sets the Host (see page 18751) property.
	SetPort (see page 18749)	Sets the Port (see page 18751) property.
	SetService (see page 18749)	Sets the Service (see page 18752) property.

Public Methods

	Name	Description
	Close (see page 18744)	Shuts down the socket connection.
	Open (see page 18747)	Opens the socket connection.

129.1.2.2.1 TAbstractSocket.Close Method

Shuts down the socket connection.

Pascal

```
procedure Close;
```

C++

```
__fastcall Close();
```

Description

Call Close to shut down the socket connection. Close sets the Active (see page 18750) property to false.

For client sockets, Close terminates the connection to the server. For server sockets, Close shuts down the connection so that the server socket is no longer listening for client requests.

See Also

Active (see page 18750)

129.1.2.2.2 TAbstractSocket.DoActivate Method

Provides the prototype of a method to open or close the socket connection.

Pascal

```
procedure DoActivate(Value: Boolean); virtual; abstract;
```

C++

```
virtual __fastcall DoActivate(Boolean Value) = 0;
```

Description

Descendants of TAbstractSocket (see page 18740) override DoActivate to provide a method that initiates or terminates the socket connection. The steps that need to be taken to open or close a connection depend on whether the socket is a client socket or a server socket.

See Also

Active (see page 18750)

Open (see page 18747)

Close (see page 18744)

129.1.2.2.3 TAbstractSocket.Error Method

Provides the prototype of a method that responds to error conditions.

Pascal

```
procedure Error(Socket: TCustomWinSocket; ErrorEvent: TErrorEvent; var ErrorCode: Integer);
virtual; abstract;
```

C++

```
virtual __fastcall Error(TCustomWinSocket Socket, TErrorEvent ErrorEvent, int ErrorCode) = 0;
```

Description

Descendants of TAbstractSocket (see page 18740) must override the abstract or, in C++ terminology, pure virtual Error method to provide a response when socket errors occur. Applications cannot call Error. The Windows socket object that encounters the error condition calls Error automatically.

The Socket parameter indicates the Windows socket object that encounters the error condition.

The ErrorEvent parameter indicates what the socket was trying to do when the error occurred. It has one of the following values:

Value	Meaning
eeGeneral	The socket received an error message that does not fit into any of the following categories.
eeSend	An error occurred when trying to write to the socket connection.
eeReceive	An error occurred when trying to read from the socket connection.
eeConnect	For client sockets, this indicates that the client socket can't locate the server, or that a problem on the server prevents the opening of a connection. For server sockets, this indicates that a client connection request that has already been accepted could not be completed.
eeDisconnect	An error occurred when trying to close a connection.
eeAccept	For server sockets only, this indicates that a problem occurred when trying to accept a client connection request.

The ErrorCode parameter is the error code received by the Windows socket object. Setting this value to 0 in the Error method prevents the socket from raising an exception. For information on possible error codes, see the Microsoft documentation on Windows sockets.

See Also

OnErrorEvent (see page 18824)

InitSocket (see page 18746)

129.1.2.2.4 TAbstractSocket.Event Method

Provides the prototype of a method that responds when specific socket events occur.

Pascal

```
procedure Event(Socket: TCustomWinSocket; SocketEvent: TSocketEvent); virtual; abstract;
```

C++

```
virtual __fastcall Event(TCustomWinSocket Socket, TSocketEvent SocketEvent) = 0;
```

Description

Descendants of TAbstractSocket (see page 18740) must override the abstract or, in C++ terminology, pure virtual Event method to provide a response to calls that arise during the normal functioning of the socket. Applications cannot call Event. The Windows socket object calls Event at appropriate times.

The Socket parameter indicates the Windows socket object that calls Event to indicate when specific events occur.

The SocketEvent parameter indicates the event that occurred. It has one of the following values:

Value	Meaning
seLookup	(client sockets only) The Windows socket object is about to locate the server socket. Changing the properties of the client socket has no effect at this point, but changes to the Windows socket object specified by the Socket parameter will affect the attempted connection.
seConnecting	(client sockets only) The server socket has been located but the connection isn't completed. This is the first opportunity to obtain the actual server port and IP address used for the connection, which may differ from those of the listening socket contacted initially.
seConnect	The socket connection has just been established. This is the point when a socket starts reading or writing over the connection.
seListen	(server sockets only) The Windows socket object has been initialized and is about to open a listening connection. Changing the properties of the server socket has no effect at this point, but changes to the Windows socket object specified by the Socket parameter will the listening connection.
seAccept	(server sockets only) A client connection request has just been accepted. This is the first opportunity to obtain the port and IP address of the individual client connection.
seWrite	The socket is ready for information to be written.
seRead	The socket is receiving information from a socket on the other end of the connection.

See Also

OnSocketEvent (see page 18824)

InitSocket (see page 18746)

129.1.2.2.5 TAbstractSocket.InitSocket Method

Tells a Windows socket object to call the Error (see page 18745) and Event (see page 18746) methods.

Pascal

```
procedure InitSocket(Socket: TCustomWinSocket);
```

C++

```
__fastcall InitSocket(TCustomWinSocket Socket);
```

Description

Applications cannot call this protected method. InitSocket is called automatically by the socket's constructor, immediately after the Windows socket object is created. It allows the socket component to respond to events on the Windows socket object by assigning an OnErrorEvent handler that calls Error (see page 18745) and an OnSocketEvent handler that calls Event (see page 18746).

The Socket parameter specifies the Windows socket object that passes events on to the socket component.

See Also

Error (see page 18745)

TAbstractSocket (see page 18740)

OnErrorEvent (see page 18824)

OnSocketEvent (see page 18824)

129.1.2.2.6 TAbstractSocket.Loaded Method

Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 18750) property.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Applications cannot call this protected method. Loaded is called automatically after the socket component has been loaded from a stream. Loaded overrides the default method to implement the value of the Active (see page 18750) property that was saved with the socket component.

See Also

Active (see page 18750)

129.1.2.2.7 TAbstractSocket.Open Method

Opens the socket connection.

Pascal

```
procedure Open;
```

C++

```
__fastcall Open();
```

Description

Call Open to initiate the socket connection. Open sets the Active (see page 18750) property to true.

For client sockets, Open locates and connects to a server. For server sockets, Open opens the socket connection in a listening mode, but does not complete the connection to a client socket.

See Also

Active (see page 18750)

129.1.2.2.8 TAbstractSocket.SetActive Method

Sets the Active (see page 18750) property.

Pascal

```
procedure SetActive(Value: Boolean);
```

C++

```
__fastcall SetActive(Boolean Value);
```

Description

SetActive is the protected implementation of the Active (see page 18750) property. SetActive calls the DoActivate (see page 18744) method to open or close the socket connection.

See Also

Active (see page 18750)

DoActivate (see page 18744)

129.1.2.2.9 TAbstractSocket.SetAddress Method

Sets the Address (see page 18750) property.

Pascal

```
procedure SetAddress(Value: string);
```

C++

```
__fastcall SetAddress(AnsiString Value);
```

Description

SetAddress is the protected implementation of the Address (see page 18750) property. SetAddress raises an ESocketError (see page 18739) exception if an attempt is made to set the Address (see page 18750) property when the socket connection is open.

See Also

Active (see page 18750)

Address (see page 18750)

ESocketError

129.1.2.2.10 TAbstractSocket.SetHost Method

Sets the Host (see page 18751) property.

Pascal

```
procedure SetHost(Value: string);
```

C++

```
__fastcall SetHost(AnsiString Value);
```

Description

SetHost is the protected implementation of the Host (see page 18751) property. SetHost raises an ESocketError (see page 18739) exception if an attempt is made to set the Host (see page 18751) property when the socket connection is open.

See Also

Active (see page 18750)

ESocketError

Host (see page 18751)

129.1.2.2.11 TAbstractSocket.SetPort Method

Sets the Port (see page 18751) property.

Pascal

```
procedure SetPort(Value: Integer);
```

C++

```
__fastcall SetPort(int Value);
```

Description

SetPort is the protected implementation of the Port (see page 18751) property. SetPort raises an ESocketError (see page 18739) exception if an attempt is made to set the Port (see page 18751) property when the socket connection is open.

See Also

Active (see page 18750)

ESocketError

Port (see page 18751)

129.1.2.2.12 TAbstractSocket.SetService Method

Sets the Service (see page 18752) property.

Pascal

```
procedure SetService(Value: string);
```

C++

```
__fastcall SetService(AnsiString Value);
```

Description

SetService is the protected implementation of the Service (see page 18752) property. SetService raises an ESocketError (see page 18739) exception if an attempt is made to set the Service (see page 18752) property when the socket connection is open.

See Also

Active (see page 18750)

ESocketError

Service (see page 18752)

129.1.2.3 TAbstractSocket Properties

The properties of the TAbstractSocket class are listed here.

Protected Properties

	Name	Description
	Active (see page 18750)	Indicates whether the socket connection is open and available for communication with other machines.
	Address (see page 18750)	Specifies the IP address of the server system.
	Host (see page 18751)	Specifies an alias for the IP address of the server system.
	Port (see page 18751)	Specifies the ID number that identifies the server socket connection.
	Service (see page 18752)	Specifies the name of the service for which the socket connection is used.

129.1.2.3.1 TAbstractSocket.Active Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before attempting to use or change the socket connection, read Active to determine whether the connection is open and ready. For client sockets, setting Active opens or shuts down a socket connection to another machine. For server sockets, setting Active opens or shuts down a listening connection that makes the socket available for client requests.

At design time, set Active to true to make the socket open a connection when the application starts running. At runtime, use the Open (see page 18747) or Close (see page 18744) method to open or close the connection.

See Also

Close (see page 18744)

Open (see page 18747)

SetActive (see page 18748)

129.1.2.3.2 TAbstractSocket.Address Property

Specifies the IP address of the server system.

Pascal

```
property Address: string;
```

C++

```
__property AnsiString Address;
```

Description

Address is a string of four numeric (byte) values in the standard Internet dot notation, such as

```
123.197.1.2
```

For client sockets, set Address to the IP address of the server to which the socket should connect. When the connection is opened, the value of Address is bound to the connection.

If the socket specifies the Host (see page 18751) property, the address for the connection is taken from the IP address associated with Host (see page 18751), rather than the value of Address. Specifying the server system by giving the IP address is faster, because the socket doesn't need to search for the IP address associated with the host name before it can locate the server system.

A single system may support more than one IP address.

Note: Trying to change Address when the connection is open will raise an ESocketError (see page 18739) exception.

See Also

ESocketError

Host (see page 18751)

SetAddress (see page 18748)

129.1.2.3.3 TAbstractSocket.Host Property

Specifies an alias for the IP address of the server system.

Pascal

```
property Host: string;
```

C++

```
__property AnsiString Host;
```

Description

Host is a string containing the domain name and service of a particular system, such as

`http://www.wSite.Com`

For client sockets, set Host to the system with which the client socket should form a connection. When the socket opens a connection, it looks up the IP address for the server socket using the value of Host.

Some servers change the system or IP address that is associated with a particular host name. Using a host name allows the client socket to find the abstract site represented by the host name, even when it has moved to a new IP address.

If Host is set, it takes precedence over the Address (see page 18750) property when looking up the address of the server.

Note: Trying to change Host when the connection is open will raise an ESocketError (see page 18739) exception.

See Also

Address (see page 18750)

ESocketError

SetHost (see page 18748)

129.1.2.3.4 TAbstractSocket.Port Property

Specifies the ID number that identifies the server socket connection.

Pascal

```
property Port: Integer;
```

C++

```
__property int Port;
```

Description

Port numbers allow a single system, identified by the Host (see page 18751) or Address (see page 18750) property, to host multiple connections simultaneously. Many values of Port are associated by convention with a particular service such as ftp or http.

For server sockets, Port is the ID of the connection on which the server socket listens for client requests. Server sockets

generally set Port to a predefined value which clients can use to initiate connections.

For client sockets, Port is the ID of the desired server connection. The value of Port is usually associated with the service the client wishes to use on the server application.

Note: Trying to change Port when the connection is open will raise an ESocketError (see page 18739) exception.

See Also

Address (see page 18750)

ESocketError

Host (see page 18751)

Service (see page 18752)

SetPort (see page 18749)

129.1.2.3.5 TAbstractSocket.Service Property

Specifies the name of the service for which the socket connection is used.

Pascal

```
property Service: string;
```

C++

```
__property AnsiString Service;
```

Description

Use Service to identify the use of the connection. Windows provides There are a number of standard service names such as ftp, http, finger, and time. Servers can specify additional services and their associated ports in a SERVICES file. For more information, see the Microsoft documentation for Windows sockets.

Certain port numbers are reserved for specific values of service. Thus, Service provides a more meaningful way to specify the server Port (see page 18751) to use for the socket connection. For server sockets, using Service rather than Port (see page 18751) ensures that the server will listen for TCP/IP requests on the appropriate port.

Note: Trying to change Service when the connection is open will raise an ESocketError (see page 18739) exception.

See Also

ESocketError

Port (see page 18751)

SetService (see page 18749)

LookupService (see page 18810)

129.1.3 TClientSocket Class

TClientSocket manages socket connections for a TCP/IP client.

Class Hierarchy

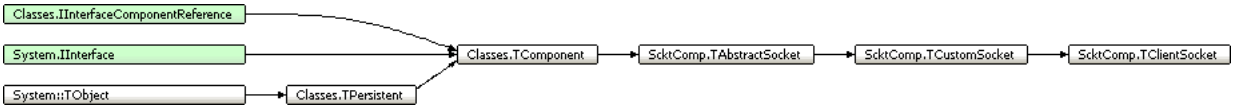

Pascal

```
TClientSocket = class(TCustomSocket);
```

C++

```
class TClientSocket : public TCustomSocket;
```

File

ScktComp

Description

Add a TClientSocket object to a form or data module to turn an application into a TCP/IP client. TClientSocket specifies a desired connection to a TCP/IP server, manages the connection when it is open, and terminates the connection when the application is through.

Note: See the CHAT demo for an example of how to use TClientSocket.

See Also

TClientWinSocket (see page 18766)

TServerSocket (see page 18843)

129.1.3.1 TClientSocket Members

The following tables list the members exposed by TClientSocket.

Protected Events

	Name	Description
	OnAccept (see page 18795)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnListen (see page 18797)	Occurs just before a server socket is opened for listening.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoActivate (see page 18757)	Provides the prototype of a method to open or close the socket connection.
A	Error (see page 18745)	Provides the prototype of a method that responds to error conditions.
V	Error (see page 18793)	Generates an OnError (see page 18797) event.

	Event (see page 18746)	Provides the prototype of a method that responds when specific socket events occur.
	Event (see page 18794)	Dispatches the events that occur in the normal operation of the socket to the appropriate event handlers.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClientType (see page 18758)	This is GetClientType, a member of class TClientSocket.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InitSocket (see page 18746)	Tells a Windows socket object to call the Error (see page 18745) and Event (see page 18746) methods.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 18747)	Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 18750) property.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActive (see page 18748)	Sets the Active (see page 18750) property.
	SetAddress (see page 18748)	Sets the Address (see page 18750) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetClientType (see page 18758)	This is SetClientType, a member of class TClientSocket.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHost (see page 18748)	Sets the Host (see page 18751) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetPort (see page 18749)	Sets the Port (see page 18751) property.
	SetService (see page 18749)	Sets the Service (see page 18752) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 18744)	Shuts down the socket connection.
	Create (see page 18756)	Creates an instance of TClientSocket (see page 18752).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 18757)	Destroys an instance of TClientSocket (see page 18752).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 18747)	Opens the socket connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.

V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Socket (see page 18762)	Describes the client endpoint of the Windows client socket connection.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnConnect (see page 18762)	Occurs on client sockets just after the connection to the server is opened.
	OnConnecting (see page 18763)	Occurs for a client socket after the server socket has been located, but before the connection is established.
	OnDisconnect (see page 18764)	Occurs just before a client socket closes the connection to a server socket.
	OnError (see page 18764)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnLookup (see page 18764)	Occurs when a client socket is about to look up the server socket with which it wants to connect.
	OnRead (see page 18765)	Occurs when a client socket should read information from the socket connection.
	OnWrite (see page 18765)	Occurs when a client socket should write information to the socket connection.

Published Properties

	Name	Description
	Active (see page 18759)	Indicates whether the socket connection is open and available for communication with other machines.
	Address (see page 18759)	Specifies the IP address of the server system.
	ClientType (see page 18760)	Specifies whether the client socket reads and writes information asynchronously over the socket connection.
	Host (see page 18760)	Specifies an alias for the IP address of the server system.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Port (see page 18761)	Specifies the ID number that identifies the server socket connection.
	Service (see page 18761)	Specifies the name of the service for which the socket connection is used.
	Tag (see page 2808)	Stores an integer value as part of a component.

129.1.3.2 TClientSocket.Create Constructor

Creates an instance of TClientSocket (see page 18752).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TClientSocket(TComponent * AOwner);
```

Description

Call Create to create and initialize a client socket component at runtime. TClientSocket (see page 18752) objects placed at design time are created automatically.

After calling the inherited constructor, Create

Allocates the TClientWinSocket (see page 18766) object that describes the client endpoint of the Windows socket connection.

I initializes the event and error dispatchers for the client socket.

See Also

TClientWinSocket (see page 18766)

Create (see page 2766)

129.1.3.3 TClientSocket.Destroy Destructor

Destroys an instance of TClientSocket (see page 18752).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TClientSocket();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the client socket is not nil, and only then calls Destroy.

Destroy frees the TClientWinSocket (see page 18766) object that describes the client endpoint of the Windows socket connection.

See Also

Free (see page 22300)

TClientWinSocket (see page 18766)

129.1.3.4 TClientSocket Methods

The methods of the TClientSocket class are listed here.

Protected Methods

	Name	Description
	DoActivate (see page 18757)	Provides the prototype of a method to open or close the socket connection.
	GetClientType (see page 18758)	This is GetClientType, a member of class TClientSocket.
	SetClientType (see page 18758)	This is SetClientType, a member of class TClientSocket.

129.1.3.4.1 TClientSocket.DoActivate Method

Provides the prototype of a method to open or close the socket connection.

Pascal

```
procedure DoActivate(Value: Boolean); override;
```

C++

```
virtual __fastcall DoActivate(Boolean Value);
```

Description

Descendants of TAbstractSocket override DoActivate to provide a method that initiates or terminates the socket connection. The steps that need to be taken to open or close a connection depend on whether the socket is a client socket or a server socket.

See Also

Active (see page 18750)

Open (see page 18747)

Close (see page 18744)

129.1.3.4.2 TClientSocket.GetClientType Method

Pascal

```
function GetClientType: TClientType;
```

C++

```
__fastcall TClientType GetClientType();
```

Description

This is GetClientType, a member of class TClientSocket.

129.1.3.4.3 TClientSocket.SetClientType Method

Pascal

```
procedure SetClientType(Value: TClientType);
```

C++

```
__fastcall SetClientType(TClientType Value);
```

Description

This is SetClientType, a member of class TClientSocket.

129.1.3.5 TClientSocket Properties

The properties of the TClientSocket class are listed here.

Public Properties

	Name	Description
R	Socket (see page 18762)	Describes the client endpoint of the Windows client socket connection.

Published Properties

	Name	Description
	Active (see page 18759)	Indicates whether the socket connection is open and available for communication with other machines.
	Address (see page 18759)	Specifies the IP address of the server system.
	ClientType (see page 18760)	Specifies whether the client socket reads and writes information asynchronously over the socket connection.
	Host (see page 18760)	Specifies an alias for the IP address of the server system.
	Port (see page 18761)	Specifies the ID number that identifies the server socket connection.

	Service (see page 18761)	Specifies the name of the service for which the socket connection is used.

129.1.3.5.1 TClientSocket.Active Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before attempting to use or change the socket connection, read Active to determine whether the connection is open and ready. For client sockets, setting Active opens or shuts down a socket connection to another machine. For server sockets, setting Active opens or shuts down a listening connection that makes the socket available for client requests.

At design time, set Active to true to make the socket open a connection when the application starts running. At runtime, use the Open or Close method to open or close the connection.

See Also

Close (see page 18744)

Open (see page 18747)

SetActive (see page 18748)

129.1.3.5.2 TClientSocket.Address Property

Specifies the IP address of the server system.

Pascal

```
property Address: string;
```

C++

```
__property AnsiString Address;
```

Description

Address is a string of four numeric (byte) values in the standard Internet dot notation, such as

```
123.197.1.2
```

For client sockets, set Address to the IP address of the server to which the socket should connect. When the connection is opened, the value of Address is bound to the connection.

If the socket specifies the Host property, the address for the connection is taken from the IP address associated with Host, rather than the value of Address. Specifying the server system by giving the IP address is faster, because the socket doesn't need to search for the IP address associated with the host name before it can locate the server system.

A single system may support more than one IP address.

Note: Trying to change Address when the connection is open will raise an ESocketError exception.

See Also

ESocketError

Host (see page 18751)

SetAddress (see page 18748)

129.1.3.5.3 TClientSocket.ClientType Property

Specifies whether the client socket reads and writes information asynchronously over the socket connection.

Pascal

```
property ClientType: TClientType;
```

C++

```
__property TClientType ClientType;
```

Description

Set ClientType to ctNonBlocking to enable the client socket to respond to asynchronous reading and writing events. When ClientType is ctNonBlocking, execution is not blocked by reading and writing over the socket connection. OnRead (see page 18765) or OnWrite (see page 18765) events occur when the socket needs to read or write over the connection.

Set ClientType to ctBlocking to force all reading and writing to occur synchronously. It is a good idea to include the client socket object in a thread if the ClientType is ctBlocking, so that I/O does not block all execution within the client application.

When ClientType is ctBlocking, use a TWinSocketStream (see page 18876) object for reading and writing. TWinSocketStream (see page 18876) prevents the application from hanging indefinitely if a problem occurs while reading or writing. It also can wait for the socket connection to indicate its readiness for reading.

Use a non-blocking socket when the socket needs to synchronize reading and writing with server sockets.

See Also

OnRead (see page 18798)

OnWrite (see page 18799)

TThread (see page 3108)

TWinSocketStream (see page 18876)

129.1.3.5.4 TClientSocket.Host Property

Specifies an alias for the IP address of the server system.

Pascal

```
property Host: string;
```

C++

```
__property AnsiString Host;
```

Description

Host is a string containing the domain name and service of a particular system, such as

`http://www.wSite.Com`

For client sockets, set Host to the system with which the client socket should form a connection. When the socket opens a connection, it looks up the IP address for the server socket using the value of Host.

Some servers change the system or IP address that is associated with a particular host name. Using a host name allows the client socket to find the abstract site represented by the host name, even when it has moved to a new IP address.

If Host is set, it takes precedence over the Address property when looking up the address of the server.

Note: Trying to change Host when the connection is open will raise an ESocketError exception.

See Also

Address (see page 18750)

ESocketError

SetHost (see page 18748)

129.1.3.5.5 TClientSocket.Port Property

Specifies the ID number that identifies the server socket connection.

Pascal

```
property Port: Integer;
```

C++

```
__property int Port;
```

Description

Port numbers allow a single system, identified by the Host or Address property, to host multiple connections simultaneously. Many values of Port are associated by convention with a particular service such as ftp or http.

For server sockets, Port is the ID of the connection on which the server socket listens for client requests. Server sockets generally set Port to a predefined value which clients can use to initiate connections.

For client sockets, Port is the ID of the desired server connection. The value of Port is usually associated with the service the client wishes to use on the server application.

Note: Trying to change Port when the connection is open will raise an ESocketError exception.

See Also

Address (see page 18750)

ESocketError

Host (see page 18751)

Service (see page 18752)

SetPort (see page 18749)

129.1.3.5.6 TClientSocket.Service Property

Specifies the name of the service for which the socket connection is used.

Pascal

```
property Service: string;
```

C++

```
__property AnsiString Service;
```

Description

Use Service to identify the use of the connection. Windows provides There are a number of standard service names such as ftp, http, finger, and time. Servers can specify additional services and their associated ports in a SERVICES file. For more information, see the Microsoft documentation for Windows sockets.

Certain port numbers are reserved for specific values of service. Thus, Service provides a more meaningful way to specify the

server Port to use for the socket connection. For server sockets, using Service rather than Port ensures that the server will listen for TCP/IP requests on the appropriate port.

Note: Trying to change Service when the connection is open will raise an ESocketError exception.

See Also

ESocketError

Port (see page 18751)

SetService (see page 18749)

LookupService (see page 18810)

129.1.3.5.7 TClientSocket.Socket Property

Describes the client endpoint of the Windows client socket connection.

Pascal

```
property Socket: TClientWinSocket;
```

C++

```
__property TClientWinSocket Socket;
```

Description

Use Socket to

Determine the address and port of both the client and server sockets that are bound to the connection.

Read or write information through the socket connection.

Determine what server notifications the client will respond to.

Obtain the Windows socket handle for making Windows socket API calls.

129.1.3.6 TClientSocket Events

The events of the TClientSocket class are listed here.

Published Events

	Name	Description
	OnConnect (see page 18762)	Occurs on client sockets just after the connection to the server is opened.
	OnConnecting (see page 18763)	Occurs for a client socket after the server socket has been located, but before the connection is established.
	OnDisconnect (see page 18764)	Occurs just before a client socket closes the connection to a server socket.
	OnError (see page 18764)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnLookup (see page 18764)	Occurs when a client socket is about to look up the server socket with which it wants to connect.
	OnRead (see page 18765)	Occurs when a client socket should read information from the socket connection.
	OnWrite (see page 18765)	Occurs when a client socket should write information to the socket connection.

129.1.3.6.1 TClientSocket.OnConnect Event

Occurs on client sockets just after the connection to the server is opened.

Pascal

```
property OnConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnect;
```

Description

Write an OnConnect event handler for a client socket to take specific action after the connection to a server socket has been established. Depending on the service, this may be the point when the socket should start reading or writing over the connection.

When a client socket opens a connection, the following events occur:

1An OnLookup event occurs prior to an attempt to locate the server socket.

2The Windows socket is set up, and initialized for event notification.

3An OnConnecting event occurs after the server socket is located.

4The connection request is accepted by the server and completed by the client socket.

5An OnConnect event occurs after the connection is established.

See Also

OnAccept (see page 18795)

OnConnecting (see page 18796)

OnLookup (see page 18797)

129.1.3.6.2 TClientSocket.OnConnecting Event

Occurs for a client socket after the server socket has been located, but before the connection is established.

Pascal

```
property OnConnecting: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnecting;
```

Description

Write an OnConnecting event handler for a client socket to take specific action just before the connection to a server socket is established. This is the first opportunity to obtain the actual port and IP address of the server endpoint of the connection that is about to form.

When a client socket opens a connection, the following events occur:

1An OnLookup event occurs prior to an attempt to locate the server socket.

2The Windows socket is set up, and initialized for event notification.

3An OnConnecting event occurs after the server socket is located.

4The connection request is accepted by the server and completed by the client socket.

5An OnConnect event occurs after the connection is established.

See Also

OnConnect (see page 18795)

OnLookup (see page 18797)

129.1.3.6.3 TClientSocket.OnDisconnect Event

Occurs just before a client socket closes the connection to a server socket.

Pascal

```
property OnDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDisconnect;
```

Description

Write an OnDisconnect event handler to take specific action when the connection to a server socket is about to be terminated. OnDisconnect occurs after the value of the Active property is set to false, but before the actual connection is closed.

See Also

Active (see page 18750)

Close (see page 18744)

OnConnect (see page 18795)

129.1.3.6.4 TClientSocket.OnError Event

Occurs when the socket fails in making, using, or shutting down a connection.

Pascal

```
property OnError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnError;
```

Description

Write an OnError event handler to respond to errors that arise with the socket connection. Set the ErrorCode parameter to 0 if the OnError event handler successfully deals with the error condition, to prevent an ESocketError from being raised.

See Also

ESocketError

129.1.3.6.5 TClientSocket.OnLookup Event

Occurs when a client socket is about to look up the server socket with which it wants to connect.

Pascal

```
property OnLookup: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnLookup;
```

Description

Write an OnLookup event handler for a client socket to take specific action just before attempting to locate a server socket. This is the first opportunity to make Windows API calls that affect the client properties of the socket, such as specifying a particular port number. Use the SocketHandle property of the Socket parameter for Windows API calls.

When a client socket opens a connection, the following events occur:

1An OnLookup event occurs prior to an attempt to locate the server socket.

2The Windows socket is set up, and initialized for event notification.

3An OnConnecting event occurs after the server socket is located.

4The connection request is accepted by the server and completed by the client socket.

5An OnConnect event occurs after the connection is established.

Note: Changing the Address, Host, Port, or Service properties of socket component in an OnLookup event handler has no effect on the address or port used to locate a server socket. These properties must be correct before the Open method is called.

See Also

Address (see page 18750)

Host (see page 18751)

OnConnect (see page 18795)

OnConnecting (see page 18796)

Open (see page 18747)

Port (see page 18751)

Service (see page 18752)

129.1.3.6.6 TClientSocket.OnRead Event

Occurs when a client socket should read information from the socket connection.

Pascal

```
property OnRead: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnRead;
```

Description

Write an OnRead event handler to read from the socket connection. If the socket is a blocking socket, use a TWinSocketStream object to read from the connection. Otherwise, u Use the methods of the Socket parameter to perform the actual reading.

Note: Non-blocking sockets do not always receive an OnRead event for the last bit of data passed over the connection. When using a non-blocking socket, check for any unread data in the OnDisconnect event to make sure that everything is handled.

See Also

receivebuf (see page 18812)

ReceiveText (see page 18812)

TWinSocketStream (see page 18876)

OnDisconnect (see page 18796)

129.1.3.6.7 TClientSocket.OnWrite Event

Occurs when a client socket should write information to the socket connection.

Pascal

```
property OnWrite: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnWrite;
```

Description

Write an OnWrite to write from the socket connection. If the socket is a blocking socket, use a TWinSocketStream object to write to the connection. Otherwise, u Use the methods of the Socket parameter to perform the actual writing.

See Also

SendBuf (see page 18813)

sendstream (see page 18814)

SendStreamThenDrop (see page 18814)

SendText (see page 18814)

TWinSocketStream (see page 18876)

129.1.4 TClientWinSocket Class

TClientWinSocket describes the client endpoint of a Windows socket connection.

Class Hierarchy

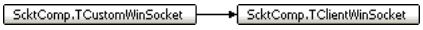

Pascal

```
TClientWinSocket = class(TCustomWinSocket);
```

C++

```
class TClientWinSocket : public TCustomWinSocket;
```

File

ScktComp

Description

Client socket components use TClientWinSocket to manage the Windows socket API calls for a TCP/IP client application. TClientWinSocket differs from its immediate ancestor TCustomWinSocket (see page 18799) only in that it introduces a ClientType (see page 18769) property to distinguish blocking from non-blocking client sockets.

See Also

TClientSocket (see page 18752)

TServerClientWinSocket (see page 18840)

TServerWinSocket (see page 18858)

129.1.4.1 TClientWinSocket Members

The following tables list the members exposed by TClientWinSocket.

Protected Methods

	Name	Description
	AsyncInitSocket (see page 18803)	Looks up information or forms socket connections asynchronously.
	DoListen (see page 18806)	Opens a listening connection.
	DoOpen (see page 18806)	Opens a connection to a remote socket.
	Error (see page 18806)	Generates an OnErrorEvent (see page 18824) event.
	Event (see page 18807)	This is Event, a member of class TCustomWinSocket.
	InitSocket (see page 18807)	Generates a windows Internet socket address structure from a description of the desired socket port.
	SetAsyncStyles (see page 18815)	Sets the AsyncStyles property.
	SetClientType (see page 18769)	This is SetClientType, a member of class TClientWinSocket.

Public Events

	Name	Description
	OnErrorEvent (see page 18824)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnSocketEvent (see page 18824)	Occurs when the Windows socket receives notification of asynchronous events.

Public Methods

	Name	Description
V	Accept (see page 18803)	Provides the interface for a method that accepts a connection to a client socket.
	Close (see page 18804)	Shuts down the socket connection if it is open.
V	Connect (see page 18768)	Responds to connection notifications.
	Create (see page 18801)	Creates an instance of TCustomWinSocket (see page 18799) from a Windows socket handle.
V	DefaultHandler (see page 18805)	Processes message records.
V	Destroy (see page 18802)	Destroys an instance of TCustomWinSocket (see page 18799).
V	Disconnect (see page 18805)	Shuts down the socket connection represented by a Windows socket handle.
	Listen (see page 18808)	Opens a listening connection for a server socket.
	Lock (see page 18809)	Starts a critical section, blocking all other execution threads until the Unlock (see page 18815) method is called.
	LookupName (see page 18809)	Generates the portion of a Windows Internet socket address structure that describes the host.
	LookupService (see page 18810)	Returns the port number associated with a specific service.
	Open (see page 18810)	Opens a connection to a remote socket.
V	Read (see page 18811)	Generates an OnSocketEvent (see page 18824) of type seRead in response to notifications that the socket connection is ready.
	ReceiveBuf (see page 18812)	This is ReceiveBuf, a member of class TCustomWinSocket.
	ReceiveLength (see page 18812)	Returns the number of bytes ready to be sent over the socket connection.
	ReceiveText (see page 18812)	Reads a string from the socket connection.
	SendBuf (see page 18813)	Writes Count bytes to the socket connection from the Buf parameter.
	SendStream (see page 18814)	This is SendStream, a member of class TCustomWinSocket.
	SendStreamThenDrop (see page 18814)	Writes all the information that can be read from the AStream parameter to the socket connection and then terminates the connection.
	SendText (see page 18814)	Writes the string S to the socket connection.
	Unlock (see page 18815)	Reenables execution in other threads after a critical section begun by the Lock (see page 18809) method.
V	Write (see page 18816)	Generates an OnSocketEvent (see page 18824) of type seWrite in response to notifications that the socket connection is ready.

Public Properties

	Name	Description
R	Addr (see page 18817)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the socket port.
	ASyncStyles (see page 18817)	Determines which asynchronous events the socket can receive.
	ClientType (see page 18769)	Specifies whether the client Windows socket reads and writes information asynchronously over the socket connection.

R	Connected (see page 18818)	Indicates whether the socket connection is open and available for communication with other machines.
	Data (see page 18818)	Points to application-specific data associated with the socket.
R	Handle (see page 18819)	Specifies the handle of the window that receives notifications from the Windows socket.
R	LocalAddress (see page 18819)	Specifies the IP address of the local system that was bound to the socket connection.
R	LocalHost (see page 18820)	Specifies the name of the local system.
R	LocalPort (see page 18820)	Specifies the ID number of the port used by the socket connection.
R	LookupState (see page 18821)	Indicates any asynchronous activities occurring while opening a connection.
R	RemoteAddr (see page 18821)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the remote socket port.
R	RemoteAddress (see page 18822)	Specifies the IP address of the remote system that was bound to the socket connection.
R	RemoteHost (see page 18822)	Specifies the name of the remote system.
R	RemotePort (see page 18823)	Specifies the ID number of the port used by the RemoteHost (see page 18822) to form the socket connection.
R	SocketHandle (see page 18823)	Specifies the Windows handle for the socket.

129.1.4.2 TClientWinSocket Methods

The methods of the TClientWinSocket class are listed here.

Protected Methods

	Name	Description
	SetClientType (see page 18769)	This is SetClientType, a member of class TClientWinSocket.

Public Methods

	Name	Description
V	Connect (see page 18768)	Responds to connection notifications.

129.1.4.2.1 TClientWinSocket.Connect Method

Responds to connection notifications.

Pascal

```
procedure Connect(Socket: TSocket); override;
```

C++

```
virtual __fastcall Connect(TSocket Socket);
```

Description

Applications can not call this protected method. Connect is called in response to notifications that the client socket has successfully completed a connection to a server socket. The Socket parameter is the Windows socket handle for the socket that receives the notification.

Connect generates an OnSocketEvent (see page 18824) event with the SocketEvent parameter set to seConnect. Socket components assign an OnSocketEvent (see page 18824) event handler that responds to this event by generating an OnConnect event on the socket component.

Note: Connect is only called if the ASyncStyles (see page 18817) property includes asConnect.

See Also

ASyncStyles (see page 18817)

OnSocketEvent (see page 18824)

OnConnect (see page 18795)

129.1.4.2.2 TClientWinSocket.SetClientType Method

Pascal

```
procedure SetClientType(Value: TClientType);
```

C++

```
__fastcall SetClientType(TClientType Value);
```

Description

This is SetClientType, a member of class TClientWinSocket.

129.1.4.3 TClientWinSocket Properties

The properties of the TClientWinSocket class are listed here.

Public Properties

	Name	Description
	ClientType (see page 18769)	Specifies whether the client Windows socket reads and writes information asynchronously over the socket connection.

129.1.4.3.1 TClientWinSocket.ClientType Property

Specifies whether the client Windows socket reads and writes information asynchronously over the socket connection.

Pascal

```
property ClientType: TClientType;
```

C++

```
__property TClientType ClientType;
```

Description

Client socket components set ClientType to reflect their own ClientType property. When ClientType is ctNonBlocking, the client Windows socket responds to asynchronous reading and writing events, and execution is not blocked by reading and writing over the connection. OnSocketEvent (see page 18824) events occur when the socket needs to read or write over the connection.

When ClientType is ctBlocking, all reading and writing occur synchronously. It is a good idea to perform all reading and writing in a separate thread if ClientType is ctBlocking. Reading and writing in a separate thread ensures that these operations do not block all execution within the client application.

When ClientType is ctBlocking, use a TWinSocketStream (see page 18876) object for reading and writing. TWinSocketStream (see page 18876) times out so that the application does not hang when a problem occurs while reading or writing. TWinSocketStream (see page 18876) also provides a method to wait, before reading, until the socket connection is ready to send information.

Use a non-blocking socket when the socket needs to synchronize reading and writing with server sockets.

See Also

OnSocketEvent (see page 18824)

TThread (see page 3108)

TWinSocketStream (see page 18876)

129.1.5 TCustomServerSocket Class

TCustomServerSocket is the base class for all server socket components.

Class Hierarchy

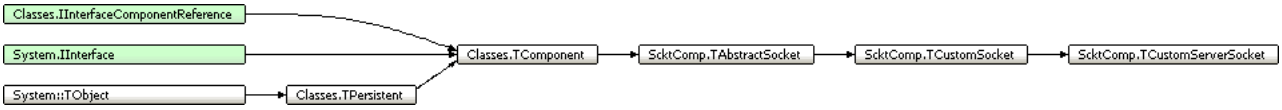

Pascal

```
TCustomServerSocket = class(TCustomSocket);
```

C++

```
class TCustomServerSocket : public TCustomSocket;
```

File

ScktComp

Description

TCustomServerSocket introduces new properties and events specific to the functions of a TCP/IP server.

Do not create instances of TCustomServerSocket. Use TCustomServerSocket as a base class when declaring server socket objects that listen for connection requests from other machines.

See Also

TServerWinSocket (see page 18858)

TServerSocket (see page 18843)

129.1.5.1 TCustomServerSocket Members

The following tables list the members exposed by TCustomServerSocket.

Protected Events

	Name	Description
	OnAccept (see page 18795)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnClientConnect (see page 18784)	Occurs when a client socket completes a connection accepted by the server socket.
	OnClientDisconnect (see page 18785)	Occurs when one of the connections to a client socket is closed.
	OnClientError (see page 18785)	Occurs when there is a failure in establishing, using, or terminating the socket connection to an individual client socket.
	OnClientRead (see page 18786)	Occurs when the server socket should read information from a client socket.
	OnClientWrite (see page 18786)	Occurs when the server socket should write information to a client socket.
	OnConnect (see page 18795)	Occurs on client sockets just after the connection to the server is opened.
	OnConnecting (see page 18796)	Occurs for a client socket after the server socket has been located, but before the connection is established.
	OnDisconnect (see page 18796)	Occurs just before a client socket closes the connection to a server socket.
	OnError (see page 18797)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnGetSocket (see page 18787)	Occurs when the server socket needs to create a new TServerClientWinSocket (see page 18840) object to represent the server endpoint of a connection to a client socket.

	OnGetThread (see page 18787)	Occurs when the server socket needs to create a new execution thread for a connection to a client socket.
	OnListen (see page 18797)	Occurs just before a server socket is opened for listening.
	OnLookup (see page 18797)	Occurs when a client socket is about to look up the server socket with which it wants to connect.
	OnRead (see page 18798)	Occurs when a client socket should read information from the socket connection.
	OnThreadEnd (see page 18788)	Occurs when a client socket connection is terminated and the associated thread finishes execution.
	OnThreadStart (see page 18788)	Occurs when the execution thread for a connection to a client socket starts up.
	OnWrite (see page 18799)	Occurs when a client socket should write information to the socket connection.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FServerSocket (see page 18775)	This is FServerSocket, a member of class TCustomServerSocket.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoActivate (see page 18775)	Opens or Closes the socket connection.
A	Error (see page 18745)	Provides the prototype of a method that responds to error conditions.
V	Error (see page 18793)	Generates an OnError (see page 18797) event.
A	Event (see page 18746)	Provides the prototype of a method that responds when specific socket events occur.
V	Event (see page 18794)	Dispatches the events that occur in the normal operation of the socket to the appropriate event handlers.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetGetSocketEvent (see page 18776)	Returns the OnGetSocket (see page 18787) event handler.
	GetGetThreadEvent (see page 18776)	Returns the OnGetThread (see page 18787) event handler.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOnClientError (see page 18776)	Returns the OnClientError (see page 18785) event handler.
	GetOnClientEvent (see page 18777)	Provides indexed access to the various event handlers that respond to client events.
	GetOnThreadEnd (see page 18777)	Returns the OnThreadEnd (see page 18788) event handler.
	GetOnThreadStart (see page 18778)	Returns the OnThreadStart (see page 18788) event handler.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetServerType (see page 18778)	Returns the ServerType (see page 18782) property.
	GetThreadCacheSize (see page 18778)	Returns the value of the ThreadCacheSize (see page 18783) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InitSocket (see page 18746)	Tells a Windows socket object to call the Error (see page 18745) and Event (see page 18746) methods.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.

	Loaded (see page 18747)	Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 18750) property.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActive (see page 18748)	Sets the Active (see page 18750) property.
	SetAddress (see page 18748)	Sets the Address (see page 18750) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetGetSocketEvent (see page 18779)	Sets the OnGetSocket (see page 18787) event handler.
	SetGetThreadEvent (see page 18779)	Sets the OnGetThread (see page 18787) event handler.
	SetHost (see page 18748)	Sets the Host (see page 18751) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnClientError (see page 18779)	Sets the OnClientError (see page 18785) event handler.
	SetOnClientEvent (see page 18780)	Sets the client event handlers of the associated TServerWinSocket (see page 18858) object.
	SetOnThreadEnd (see page 18780)	Sets the OnThreadEnd (see page 18788) event handler.
	SetOnThreadStart (see page 18781)	Sets the OnThreadStart (see page 18788) event handler.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetPort (see page 18749)	Sets the Port (see page 18751) property.
	SetServerType (see page 18781)	Sets the ServerType (see page 18782) property.
	SetService (see page 18749)	Sets the Service (see page 18752) property.
	SetThreadCacheSize (see page 18781)	Sets the ThreadCacheSize (see page 18783) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Active (see page 18750)	Indicates whether the socket connection is open and available for communication with other machines.
	Address (see page 18750)	Specifies the IP address of the server system.
	Host (see page 18751)	Specifies an alias for the IP address of the server system.
	Port (see page 18751)	Specifies the ID number that identifies the server socket connection.
	ServerType (see page 18782)	Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.
	Service (see page 18752)	Specifies the name of the service for which the socket connection is used.
	ThreadCacheSize (see page 18783)	Specifies the maximum number of threads that can be reused for new client connections.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.

	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 18744)	Shuts down the socket connection.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 18774)	Destroys an instance of TCustomServerSocket (see page 18770).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 18747)	Opens the socket connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

129.1.5.2 TCustomServerSocket.Destroy Destructor

Destroys an instance of TCustomServerSocket (see page 18770).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomServerSocket();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the server socket is not nil, and only then calls Destroy.

Destroy frees the TServerWinSocket (see page 18858) object that describes the server endpoint of the listening connection.

See Also

Free (see page 22300)

TServerWinSocket (see page 18858)

129.1.5.3 TCustomServerSocket Fields

The fields of the TCustomServerSocket class are listed here.

Protected Fields

	Name	Description
	FServerSocket (see page 18775)	This is FServerSocket, a member of class TCustomServerSocket.

129.1.5.3.1 TCustomServerSocket.FServerSocket Field

Pascal

```
FServerSocket: TServerWinSocket;
```

C++

```
TServerWinSocket FServerSocket;
```

Description

This is FServerSocket, a member of class TCustomServerSocket.

129.1.5.4 TCustomServerSocket Methods

The methods of the TCustomServerSocket class are listed here.

Protected Methods

	Name	Description
V	DoActivate (see page 18775)	Opens or Closes the socket connection.
	GetGetSocketEvent (see page 18776)	Returns the OnGetSocket (see page 18787) event handler.
	GetGetThreadEvent (see page 18776)	Returns the OnGetThread (see page 18787) event handler.
	GetOnClientError (see page 18776)	Returns the OnClientError (see page 18785) event handler.
	GetOnClientEvent (see page 18777)	Provides indexed access to the various event handlers that respond to client events.
	GetOnThreadEnd (see page 18777)	Returns the OnThreadEnd (see page 18788) event handler.
	GetOnThreadStart (see page 18778)	Returns the OnThreadStart (see page 18788) event handler.
	GetServerType (see page 18778)	Returns the ServerType (see page 18782) property.
	GetThreadCacheSize (see page 18778)	Returns the value of the ThreadCacheSize (see page 18783) property.
	SetGetSocketEvent (see page 18779)	Sets the OnGetSocket (see page 18787) event handler.
	SetGetThreadEvent (see page 18779)	Sets the OnGetThread (see page 18787) event handler.
	SetOnClientError (see page 18779)	Sets the OnClientError (see page 18785) event handler.
	SetOnClientEvent (see page 18780)	Sets the client event handlers of the associated TServerWinSocket (see page 18858) object.
	SetOnThreadEnd (see page 18780)	Sets the OnThreadEnd (see page 18788) event handler.
	SetOnThreadStart (see page 18781)	Sets the OnThreadStart (see page 18788) event handler.
	SetServerType (see page 18781)	Sets the ServerType (see page 18782) property.
	SetThreadCacheSize (see page 18781)	Sets the ThreadCacheSize (see page 18783) property.

129.1.5.4.1 TCustomServerSocket.DoActivate Method

Opens or Closes the socket connection.

Pascal

```
procedure DoActivate(Value: Boolean); override;
```

C++

```
virtual __fastcall DoActivate(Boolean Value);
```

Description

DoActivate provides the underlying implementation of the Open (see page 18747) and Close (see page 18744) methods.

When Value is true, DoActivate opens a listening connection to accept connection requests. When Value is false, DoActivate terminates the current listening connection and any connections to client sockets.

See Also

Active (see page 18750)

Open (see page 18747)

Close (see page 18744)

129.1.5.4.2 TCustomServerSocket.GetGetSocketEvent Method

Returns the OnGetSocket (see page 18787) event handler.

Pascal

```
function GetGetSocketEvent: TGetSocketEvent;
```

C++

```
__fastcall TGetSocketEvent GetGetSocketEvent();
```

Description

Applications cannot call this protected method. GetGetSocketEvent returns the OnGetSocket (see page 18787) event handler of the TServerWinSocket (see page 18858) object that describes the server endpoint of the listening connection.

See Also

OnGetSocket (see page 18787)

OnGetSocket (see page 18874)

129.1.5.4.3 TCustomServerSocket.GetGetThreadEvent Method

Returns the OnGetThread (see page 18787) event handler.

Pascal

```
function GetGetThreadEvent: TGetThreadEvent;
```

C++

```
__fastcall TGetThreadEvent GetGetThreadEvent();
```

Description

Applications cannot call this protected method. GetGetThreadEvent returns the OnGetThread (see page 18787) event handler of the TServerWinSocket (see page 18858) object that describes the server endpoint of the listening connection.

See Also

OnGetThread (see page 18787)

OnGetThread (see page 18875)

129.1.5.4.4 TCustomServerSocket.GetOnClientError Method

Returns the OnClientError (see page 18785) event handler.

Pascal

```
function GetOnClientError: TSocketErrorEvent;
```

C++

```
__fastcall TSocketErrorEvent GetOnClientError();
```

Description

Applications cannot call this protected method. GetOnClientError returns the OnClientError (see page 18785) event handler of the TServerWinSocket (see page 18858) object that describes the server endpoint of the listening connection.

See Also

OnClientError (see page 18785)

OnClientError (see page 18872)

129.1.5.4.5 TCustomServerSocket.GetOnClientEvent Method

Provides indexed access to the various event handlers that respond to client events.

Pascal

```
function GetOnClientEvent(Index: Integer): TSocketNotifyEvent;
```

C++

```
__fastcall TSocketNotifyEvent GetOnClientEvent(int Index);
```

Description

Applications can't call this protected method. It is used internally to implement the event handlers that respond to client events.

See Also

OnClientRead (see page 18786)

OnClientWrite (see page 18786)

OnClientConnect (see page 18784)

OnClientDisconnect (see page 18785)

OnClientRead (see page 18873)

OnClientWrite (see page 18874)

OnClientConnect (see page 18871)

OnClientDisconnect (see page 18872)

129.1.5.4.6 TCustomServerSocket.GetOnThreadEnd Method

Returns the OnThreadEnd (see page 18788) event handler.

Pascal

```
function GetOnThreadEnd: TThreadNotifyEvent;
```

C++

```
__fastcall TThreadNotifyEvent GetOnThreadEnd();
```

Description

Applications cannot call this protected method. GetOnThreadEnd returns the OnThreadEnd (see page 18788) event handler of the TServerWinSocket (see page 18858) object that describes the server endpoint of the listening connection.

See Also

OnThreadEnd (see page 18788)

OnThreadEnd (see page 18875)

129.1.5.4.7 TCustomServerSocket.GetOnThreadStart Method

Returns the OnThreadStart (see page 18788) event handler.

Pascal

```
function GetOnThreadStart: TThreadNotifyEvent;
```

C++

```
__fastcall TThreadNotifyEvent GetOnThreadStart();
```

Description

Applications cannot call this protected method. GetOnThreadStart returns the OnThreadStart (see page 18788) event handler of the TServerWinSocket (see page 18858) object that describes the server endpoint of the listening connection.

See Also

OnThreadStart (see page 18788)

OnThreadStart (see page 18876)

129.1.5.4.8 TCustomServerSocket.GetServerType Method

Returns the ServerType (see page 18782) property.

Pascal

```
function GetServerType: TServerType;
```

C++

```
__fastcall TServerType GetServerType();
```

Description

Applications cannot call this protected method. GetServerType returns the value of the ServerType (see page 18782) property of the TServerWinSocket (see page 18858) object that describes the server endpoint of the listening connection.

See Also

ServerType (see page 18782)

ServerType (see page 18869)

129.1.5.4.9 TCustomServerSocket.GetThreadCacheSize Method

Returns the value of the ThreadCacheSize (see page 18783) property.

Pascal

```
function GetThreadCacheSize: Integer;
```

C++

```
__fastcall int GetThreadCacheSize();
```

Description

Applications cannot call this protected method. GetThreadCacheSize returns the value of the ThreadCacheSize (see page 18783) property of the TServerWinSocket (see page 18858) object that describes the server endpoint of the listening connection.

See Also

ThreadCacheSize (see page 18783)

ThreadCacheSize (see page 18870)

129.1.5.4.10 TCustomServerSocket.SetGetSocketEvent Method

Sets the OnGetSocket (see page 18787) event handler.

Pascal

```
procedure SetGetSocketEvent(Value: TGetSocketEvent);
```

C++

```
__fastcall SetGetSocketEvent(TGetSocketEvent Value);
```

Description

Applications cannot call this protected method. SetGetSocketEvent sets the OnGetSocket (see page 18787) event handler of the TServerWinSocket (see page 18858) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 18858) to implement their OnGetSocket (see page 18787) event.

See Also

OnGetSocket (see page 18787)

OnGetSocket (see page 18874)

129.1.5.4.11 TCustomServerSocket.SetGetThreadEvent Method

Sets the OnGetThread (see page 18787) event handler.

Pascal

```
procedure SetGetThreadEvent(Value: TGetThreadEvent);
```

C++

```
__fastcall SetGetThreadEvent(TGetThreadEvent Value);
```

Description

Applications cannot call this protected method. SetGetThreadEvent sets the OnGetThread (see page 18787) event handler of the TServerWinSocket (see page 18858) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 18858) to implement their OnGetThread (see page 18787) event.

See Also

OnGetThread (see page 18787)

OnGetThread (see page 18875)

129.1.5.4.12 TCustomServerSocket.SetOnClientError Method

Sets the OnClientError (see page 18785) event handler.

Pascal

```
procedure SetOnClientError(Value: TSocketErrorEvent);
```

C++

```
__fastcall SetOnClientError(TSocketErrorEvent Value);
```

Description

Applications cannot call this protected method. SetOnClientError sets the OnClientError (see page 18785) event handler of the TServerWinSocket (see page 18858) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 18858) to implement their OnClientError (see page 18785) event.

See Also

OnClientError (see page 18785)

OnClientError (see page 18872)

129.1.5.4.13 TCustomServerSocket.SetOnClientEvent Method

Sets the client event handlers of the associated TServerWinSocket (see page 18858) object.

Pascal

```
procedure SetOnClientEvent(Index: Integer; Value: TSocketNotifyEvent);
```

C++

```
__fastcall SetOnClientEvent(int Index, TSocketNotifyEvent Value);
```

Description

Applications cannot call this protected method. SetOnClientEvent sets the various event handlers that respond to events from the server endpoint of a client connection. Server socket components use TServerWinSocket (see page 18858) to implement these events.

See Also

OnClientRead (see page 18786)

OnClientWrite (see page 18786)

OnClientConnect (see page 18784)

OnClientDisconnect (see page 18785)

OnClientRead (see page 18873)

OnClientWrite (see page 18874)

OnClientConnect (see page 18871)

OnClientDisconnect (see page 18872)

129.1.5.4.14 TCustomServerSocket.SetOnThreadEnd Method

Sets the OnThreadEnd (see page 18788) event handler.

Pascal

```
procedure SetOnThreadEnd(Value: TThreadNotifyEvent);
```

C++

```
__fastcall SetOnThreadEnd(TThreadNotifyEvent Value);
```

Description

Applications cannot call this protected method. SetOnThreadEnd sets the OnThreadEnd (see page 18788) event handler of the TServerWinSocket (see page 18858) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 18858) to implement their OnThreadEnd (see page 18788) event.

See Also

OnThreadEnd (see page 18788)

OnThreadEnd (see page 18875)

129.1.5.4.15 TCustomServerSocket.SetOnThreadStart Method

Sets the OnThreadStart (see page 18788) event handler.

Pascal

```
procedure SetOnThreadStart(Value: TThreadNotifyEvent);
```

C++

```
__fastcall SetOnThreadStart(TThreadNotifyEvent Value);
```

Description

Applications cannot call this protected method. SetOnThreadStart sets the OnThreadEnd (see page 18788) event handler of the TServerWinSocket (see page 18858) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 18858) to implement their OnThreadEnd (see page 18788) event.

See Also

OnThreadStart (see page 18788)

OnThreadEnd (see page 18875)

129.1.5.4.16 TCustomServerSocket.SetServerType Method

Sets the ServerType (see page 18782) property.

Pascal

```
procedure SetServerType(Value: TServerType);
```

C++

```
__fastcall SetServerType(TServerType Value);
```

Description

Applications cannot call this protected method. SetServerType sets the ServerType (see page 18782) property of the TServerWinSocket (see page 18858) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 18858) to implement their ServerType (see page 18782) property.

See Also

ServerType (see page 18782)

ServerType (see page 18869)

129.1.5.4.17 TCustomServerSocket.SetThreadCacheSize Method

Sets the ThreadCacheSize (see page 18783) property.

Pascal

```
procedure SetThreadCacheSize(Value: Integer);
```

C++

```
__fastcall SetThreadCacheSize(int Value);
```

Description

Applications cannot call this protected method. SetThreadCacheSize sets the ThreadCacheSize (see page 18783) property of the TServerWinSocket (see page 18858) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 18858) to implement their ThreadCacheSize (see page 18783) property.

See Also

ThreadCacheSize (see page 18783)

ThreadCacheSize (see page 18870)

129.1.5.5 TCustomServerSocket Properties

The properties of the TCustomServerSocket class are listed here.

Protected Properties

	Name	Description
	ServerType (see page 18782)	Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.
	ThreadCacheSize (see page 18783)	Specifies the maximum number of threads that can be reused for new client connections.

129.1.5.5.1 TCustomServerSocket.ServerType Property

Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.

Pascal

```
property ServerType: TServerType;
```

C++

```
__property TServerType ServerType;
```

Description

Set ServerType to stThreadBlocking to automatically spawn a new thread for each socket connection accepted by the server socket. When ServerType is stThreadBlocking, the execution of the connection thread is suspended while reading or writing until all information has been transferred over the connection. The thread for each connection generates OnClientRead (see page 18786) or OnClientWrite (see page 18786) events when the server socket needs to read or write.

Set ServerType to stNonBlocking to handle all reading and writing over the socket connections asynchronously. When ServerType is stNonBlocking, all client connections are handled in a single execution thread by default. OnClientRead (see page 18786) or OnClientWrite (see page 18786) events occur when the client socket on the other end of one of the connections tries to send or receive information over the connection.

Use a non-blocking socket when the socket needs to synchronize reading and writing with client sockets.

Note: When ServerType is stThreadBlocking, it is important that the OnClient... event handlers contain thread-safe code.

See Also

OnClientRead (see page 18786)

OnClientWrite (see page 18786)

TServerClientThread (see page 18831)

129.1.5.5.2 TCustomServerSocket.ThreadCacheSize Property

Specifies the maximum number of threads that can be reused for new client connections.

Pascal

```
property ThreadCacheSize: Integer;
```

C++

```
__property int ThreadCacheSize;
```

Description

When ServerType (see page 18782) is stThreadBlocking, each new connection that is accepted by the server socket is given a separate execution thread. In order to improve performance, server sockets store these threads in a cache rather than freeing them when the connection is closed. New connections can then reuse threads from the cache, rather than requiring the socket server to create a new thread every time a connection is accepted.

Set ThreadCacheSize to specify the number of threads that are cached for reuse. The ideal value for ThreadCacheSize depends on the number and frequency of client socket requests received by the server socket. If ThreadCacheSize is too low, the server socket will spend more time freeing and creating threads when client connections are accepted. If ThreadCacheSize is too high, the server socket may unnecessarily lock up the memory for threads that are never reused.

See Also

ServerType (see page 18782)

TServerClientThread (see page 18831)

ActiveThreads (see page 18867)

IdleThreads (see page 18869)

129.1.5.6 TCustomServerSocket Events

The events of the TCustomServerSocket class are listed here.

Protected Events

	Name	Description
	OnClientConnect (see page 18784)	Occurs when a client socket completes a connection accepted by the server socket.
	OnClientDisconnect (see page 18785)	Occurs when one of the connections to a client socket is closed.
	OnClientError (see page 18785)	Occurs when there is a failure in establishing, using, or terminating the socket connection to an individual client socket.
	OnClientRead (see page 18786)	Occurs when the server socket should read information from a client socket.
	OnClientWrite (see page 18786)	Occurs when the server socket should write information to a client socket.
	OnGetSocket (see page 18787)	Occurs when the server socket needs to create a new TServerClientWinSocket (see page 18840) object to represent the server endpoint of a connection to a client socket.
	OnGetThread (see page 18787)	Occurs when the server socket needs to create a new execution thread for a connection to a client socket.

	OnThreadEnd (see page 18788)	Occurs when a client socket connection is terminated and the associated thread finishes execution.
	OnThreadStart (see page 18788)	Occurs when the execution thread for a connection to a client socket starts up.

129.1.5.6.1 TCustomServerSocket.OnClientConnect Event

Occurs when a client socket completes a connection accepted by the server socket.

Pascal

```
property OnClientConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientConnect;
```

Description

Write an OnClientConnect event handler to take specific action when a client socket completes the socket connection to the server socket. For example, the socket may start reading or writing over the connection in an OnClientConnect event handler.

The order of server socket events leading up to OnClientConnect is as follows:

1An OnListen (see page 18797) event occurs just before the server socket is opened for listening.

2The server socket receives client requests in a listening queue. The server socket accepts one of those requests, and receives a Windows socket handle for the new socket connection.

3The server socket generates an OnGetSocket (see page 18787) event, passing in the Windows socket handle. If a TServerClientWinSocket (see page 18840) object for the server endpoint of the new connection is not created in an OnGetSocket (see page 18787) event handler, the server socket creates one. The TServerWinSocket (see page 18858) object continues to listen for other clients.

4An OnAccept (see page 18795) event occurs, using the new TServerClientWinSocket (see page 18840) object.

5If ServerType (see page 18782) is stThreadBlocking and no thread is available in the cache, an OnGetThread (see page 18787) event occurs. If the OnGetThread (see page 18787) event handler does not create a thread, the server socket creates a TServerClientThread (see page 18831).

6If ServerType (see page 18782) is stThreadBlocking, an OnThreadStart (see page 18788) event occurs as the thread begins execution.

7The client completes the connection to the TServerClientWinSocket (see page 18840) object and an OnClientConnect event occurs.

Note: If ServerType (see page 18782) is stThreadBlocking, make sure that all code in an OnClientConnect event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

Note: The OnClientConnect event handler for TServerSocket (see page 18843) is also set when setting the OnClientConnect event handler of the associated TServerWinSocket (see page 18858).

See Also

OnAccept (see page 18795)

OnGetSocket (see page 18787)

OnGetThread (see page 18787)

OnListen (see page 18797)

OnThreadStart (see page 18788)

ServerType (see page 18782)

TServerClientWinSocket (see page 18840)

TServerWinSocket (see page 18858)

GetClientThread (see page 18865)

129.1.5.6.2 TCustomServerSocket.OnClientDisconnect Event

Occurs when one of the connections to a client socket is closed.

Pascal

```
property OnClientDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientDisconnect;
```

Description

Write an OnClientDisconnect event handler to take specific action when the connection to a client socket is ending. The termination of a client connection does not close the server socket. The server socket remains open and listening for client requests on its listening connection.

The TServerClientWinSocket (see page 18840) that describes the server endpoint of the client connection is freed after OnClientDisconnect. If ServerType (see page 18782) is stThreadBlocking, OnThreadEnd (see page 18788) occurs after OnClientDisconnect.

Note: If ServerType (see page 18782) is stThreadBlocking, make sure that all code in an OnClientDisconnect event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

Note: The OnClientDisconnect event handler for TServerSocket (see page 18843) is also set when setting the OnClientDisconnect event handler of the associated TServerWinSocket (see page 18858).

See Also

OnThreadEnd (see page 18788)

ServerType (see page 18782)

TServerClientWinSocket (see page 18840)

TServerWinSocket (see page 18858)

GetClientThread (see page 18865)

129.1.5.6.3 TCustomServerSocket.OnClientError Event

Occurs when there is a failure in establishing, using, or terminating the socket connection to an individual client socket.

Pascal

```
property OnClientError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnClientError;
```

Description

Write an OnClientError event handler to respond to errors that arise with a connection to an individual client. Set the ErrorCode

parameter to 0 if the OnClientError event handler successfully deals with the error condition to prevent an ESocketError (see page 18739) from being raised.

Note: If ServerType (see page 18782) is stThreadBlocking, make sure that all code in an OnClientError event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

Note: The OnClientError event handler for TServerSocket (see page 18843) is also set when setting the OnClientError event handler of the associated TServerWinSocket (see page 18858).

See Also

ESocketError

ServerType (see page 18782)

TServerClientWinSocket (see page 18840)

GetClientThread (see page 18865)

129.1.5.6.4 TCustomServerSocket.OnClientRead Event

Occurs when the server socket should read information from a client socket.

Pascal

```
property OnClientRead: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientRead;
```

Description

Write an OnClientRead event handler to write to the socket connection. If the ServerType (see page 18782) property is stThreadBlocking, use a TWinSocketStream (see page 18876) object to prevent problems that arise while writing from causing the execution thread to hang indefinitely. Otherwise, uUse the methods of the Socket parameter to perform the actual writing.

Note: If ServerType (see page 18782) is stThreadBlocking, make sure that all code in an OnClientRead event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

See Also

ServerType (see page 18782)

receivebuf (see page 18812)

ReceiveText (see page 18812)

TServerClientWinSocket (see page 18840)

GetClientThread (see page 18865)

TWinSocketStream (see page 18876)

129.1.5.6.5 TCustomServerSocket.OnClientWrite Event

Occurs when the server socket should write information to a client socket.

Pascal

```
property OnClientWrite: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientWrite;
```

Description

Write an OnClientWrite event handler to write to the socket connection. If the ServerType (see page 18782) property is stThreadBlocking, use a TWinSocketStream (see page 18876) object to prevent problems that arise while writing from causing the execution thread to hang indefinitely. Otherwise, uUse the methods of the Socket parameter to perform the actual writing.

Note: If ServerType (see page 18782) is stThreadBlocking, make sure that all code in an OnClientWrite event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

See Also

ServerType (see page 18782)

SendBuf (see page 18813)

sendstream (see page 18814)

SendText (see page 18814)

GetClientThread (see page 18865)

TWinSocketStream (see page 18876)

129.1.5.6.6 TCustomServerSocket.OnGetSocket Event

Occurs when the server socket needs to create a new TServerClientWinSocket (see page 18840) object to represent the server endpoint of a connection to a client socket.

Pascal

```
property OnGetSocket: TGetSocketEvent;
```

C++

```
__property TGetSocketEvent OnGetSocket;
```

Description

Write an OnGetSocket event handler to create a specialized descendant of TServerClientWinSocket (see page 18840) for the server socket component to use. Return the new TServerClientWinSocket (see page 18840) object in the ClientSocket parameter. The Sender parameter is the TServerWinSocket (see page 18858) object that describes the server endpoint of the listening connection.

See Also

TServerClientWinSocket (see page 18840)

Create (see page 18842)

TServerWinSocket (see page 18858)

129.1.5.6.7 TCustomServerSocket.OnGetThread Event

Occurs when the server socket needs to create a new execution thread for a connection to a client socket.

Pascal

```
property OnGetThread: TGetThreadEvent;
```

C++

```
__property TGetThreadEvent OnGetThread;
```

Description

Write an OnGetThread event handler to create a specialized descendant of TServerClientThread (see page 18831) for the connection to the client socket. Return the new TServerClientThread (see page 18831) object in the SocketThread parameter.

Most applications that use thread-blocking server components will want to provide an OnGetThread event handler and implement a TServerClientThread (see page 18831) that handles its own reading and writing in a thread-safe manner rather than relying on the default TServerClientThread (see page 18831) which triggers OnClientRead (see page 18786) and OnClientWrite (see page 18786) events. This is because the OnClientRead (see page 18786) and OnClientWrite (see page 18786) event handlers reside on the server socket, which is in global memory.

The Sender parameter is the TServerWinSocket (see page 18858) object that describes the server endpoint of the listening connection. The ClientSocket parameter is the TServerClientWinSocket (see page 18840) object that describes the server endpoint of the connection to the client socket that is about to be formed.

See Also

ServerType (see page 18782)

TServerClientThread (see page 18831)

TServerClientWinSocket (see page 18840)

TServerWinSocket (see page 18858)

129.1.5.6.8 TCustomServerSocket.OnThreadEnd Event

Occurs when a client socket connection is terminated and the associated thread finishes execution.

Pascal

```
property OnThreadEnd: TThreadNotifyEvent;
```

C++

```
__property TThreadNotifyEvent OnThreadEnd;
```

Description

Write an OnThreadEnd to take specific action when the thread for a client connection finishes execution. OnThreadEnd occurs after the OnClientDisconnect (see page 18785) event.

The Sender parameter is the TServerWinSocket (see page 18858) that describes the server endpoint of the listening connection. The Thread parameter is the thread that is finishing up.

See Also

OnClientDisconnect (see page 18785)

TServerWinSocket (see page 18858)

129.1.5.6.9 TCustomServerSocket.OnThreadStart Event

Occurs when the execution thread for a connection to a client socket starts up.

Pascal

```
property OnThreadStart: TThreadNotifyEvent;
```

C++

```
__property TThreadNotifyEvent OnThreadStart;
```

Description

Write an OnThreadStart to initialize the TServerClientThread (see page 18831) for the connection to a single client socket. The Sender parameter is the TServerWinSocket (see page 18858) that describes the server endpoint of the listening connection. The Thread parameter is the thread that is about to start execution.

See Also

TServerClientThread (see page 18831)

TServerWinSocket (see page 18858)

129.1.6 TCustomSocket Class

TCustomSocket is the base class for socket components.

Class Hierarchy

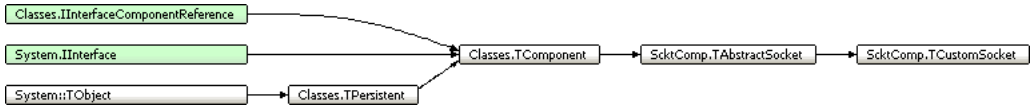

Pascal

```
TCustomSocket = class(TAbstractSocket);
```

C++

```
class TCustomSocket : public TAbstractSocket;
```

File

ScktComp

Description

Override TCustomSocket to create a socket component that can be assigned event handlers for the events and error conditions that occur. In addition to the properties and methods introduced by TAbstractSocket (see page 18740) that enable an application to work with Windows sockets, TCustomSocket introduces several events that can be used by applications.

Do not create instances of TCustomSocket. Use a descendant of TCustomSocket to add the ability to communicate with other machines using TCP/IP. To create a socket that initiates the connection with another machine, use TClientSocket (see page 18752). To create a socket that responds to requests for connections from other machines, use TServerSocket (see page 18843).

See Also

TClientSocket (see page 18752)

TCustomWinSocket (see page 18799)

TServerSocket (see page 18843)

129.1.6.1 TCustomSocket Members

The following tables list the members exposed by TCustomSocket.

Protected Events

	Name	Description
	OnAccept (see page 18795)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnConnect (see page 18795)	Occurs on client sockets just after the connection to the server is opened.
	OnConnecting (see page 18796)	Occurs for a client socket after the server socket has been located, but before the connection is established.
	OnDisconnect (see page 18796)	Occurs just before a client socket closes the connection to a server socket.
	OnError (see page 18797)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnListen (see page 18797)	Occurs just before a server socket is opened for listening.
	OnLookup (see page 18797)	Occurs when a client socket is about to look up the server socket with which it wants to connect.
	OnRead (see page 18798)	Occurs when a client socket should read information from the socket connection.
	OnWrite (see page 18799)	Occurs when a client socket should write information to the socket connection.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
A	DoActivate (see page 18744)	Provides the prototype of a method to open or close the socket connection.
V	Error (see page 18793)	Generates an OnError (see page 18797) event.
V	Event (see page 18794)	Dispatches the events that occur in the normal operation of the socket to the appropriate event handlers.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InitSocket (see page 18746)	Tells a Windows socket object to call the Error (see page 18745) and Event (see page 18746) methods.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 18747)	Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 18750) property.
V	Notification (see page 2783)	Forwards notification messages to all owned components.

	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActive (see page 18748)	Sets the Active (see page 18750) property.
	SetAddress (see page 18748)	Sets the Address (see page 18750) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHost (see page 18748)	Sets the Host (see page 18751) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetPort (see page 18749)	Sets the Port (see page 18751) property.
	SetService (see page 18749)	Sets the Service (see page 18752) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Active (see page 18750)	Indicates whether the socket connection is open and available for communication with other machines.
	Address (see page 18750)	Specifies the IP address of the server system.
	Host (see page 18751)	Specifies an alias for the IP address of the server system.
	Port (see page 18751)	Specifies the ID number that identifies the server socket connection.
	Service (see page 18752)	Specifies the name of the service for which the socket connection is used.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 18744)	Shuts down the socket connection.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.

V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 18747)	Opens the socket connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

129.1.6.2 TCustomSocket Methods

The methods of the TCustomSocket class are listed here.

Protected Methods

	Name	Description
V	Error (see page 18793)	Generates an OnError (see page 18797) event.
V	Event (see page 18794)	Dispatches the events that occur in the normal operation of the socket to the appropriate event handlers.

129.1.6.2.1 TCustomSocket.Error Method

Generates an OnError (see page 18797) event.

Pascal

```
procedure Error(Socket: TCustomWinSocket; ErrorEvent: TErrorEvent; var ErrorCode: Integer);
override;
```

C++

```
virtual __fastcall Error(TCustomWinSocket Socket, TErrorEvent ErrorEvent, int ErrorCode);
```

Description

Applications cannot call Error. It responds to error conditions by calling the OnError (see page 18797) event handler if it is assigned. Override Error to add class-specific error processing or to block the OnError (see page 18797) event.

The Socket parameter indicates the Windows socket object which encounters the error condition.

The ErrorEvent parameter indicates what the socket was trying to do when the error occurred. It has one of the following values:

Value	Meaning
eeGeneral	The socket received an error message that does not fit into any of the following categories.
eeSend	An error occurred when trying to write to the socket connection.
eeReceive	An error occurred when trying to read from the socket connection.
eeConnect	For client sockets, this indicates that the client socket can't locate the server, or that a problem on the server prevents the opening of a connection. For server sockets, this indicates that a client connection request that has already been accepted could not be completed.
eeDisconnect	An error occurred when trying to close a connection.
eeAccept	For server sockets only, this indicates that a problem occurred when trying to accept a client connection request.

The ErrorCode parameter is the error code received by the Windows socket object. Setting this value to 0 in the Error method prevents the socket from raising an exception. For information on possible error codes, see the Microsoft documentation on Windows sockets.

See Also

OnError (see page 18797)

129.1.6.2.2 TCustomSocket.Event Method

Dispatches the events that occur in the normal operation of the socket to the appropriate event handlers.

Pascal

```
procedure Event(Socket: TCustomWinSocket; SocketEvent: TSocketEvent); override;
```

C++

```
virtual __fastcall Event(TCustomWinSocket Socket, TSocketEvent SocketEvent);
```

Description

Applications cannot call Event. It is called automatically when various events occur. Override Event to introduce class-specific responses when certain events occur or to prevent the socket from calling the associated event handler.

The Socket parameter indicates the Windows socket object that calls Event to indicate when specific events occur.

The SocketEvent parameter indicates the event that occurred.

See Also

OnAccept (see page 18795)

OnConnect (see page 18795)

OnConnecting (see page 18796)

OnDisconnect (see page 18796)

OnListen (see page 18797)

OnLookup (see page 18797)

OnRead (see page 18798)

OnWrite (see page 18799)

129.1.6.3 TCustomSocket Events

The events of the TCustomSocket class are listed here.

Protected Events

	Name	Description
	OnAccept (see page 18795)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnConnect (see page 18795)	Occurs on client sockets just after the connection to the server is opened.
	OnConnecting (see page 18796)	Occurs for a client socket after the server socket has been located, but before the connection is established.
	OnDisconnect (see page 18796)	Occurs just before a client socket closes the connection to a server socket.
	OnError (see page 18797)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnListen (see page 18797)	Occurs just before a server socket is opened for listening.
	OnLookup (see page 18797)	Occurs when a client socket is about to look up the server socket with which it wants to connect.

	OnRead (see page 18798)	Occurs when a client socket should read information from the socket connection.
	OnWrite (see page 18799)	Occurs when a client socket should write information to the socket connection.

129.1.6.3.1 TCustomSocket.OnAccept Event

Occurs on server sockets just after the connection to a client socket is accepted.

Pascal

```
property OnAccept: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnAccept;
```

Description

Write an OnAccept event handler for a server socket to take specific action after the connection to a client socket has been accepted. This is the first point when information is available about the local port and IP address that will be used in the server endpoint of the connection. The port number, and possibly the IP address, will differ from those of the listening socket.

Server sockets open a socket connection for listening to establish a queue to hold client requests. After a request from the queue is accepted by the server socket, an OnAccept event occurs. After the server socket accepts the connection, the client socket completes the connection, and an OnConnect (see page 18795) event occurs for the client socket.

See Also

OnConnect (see page 18795)

OnListen (see page 18797)

129.1.6.3.2 TCustomSocket.OnConnect Event

Occurs on client sockets just after the connection to the server is opened.

Pascal

```
property OnConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnect;
```

Description

Write an OnConnect event handler for a client socket to take specific action after the connection to a server socket has been established. Depending on the service, this may be the point when the socket should start reading or writing over the connection.

When a client socket opens a connection, the following events occur:

1An OnLookup (see page 18797) event occurs prior to an attempt to locate the server socket.

2The Windows socket is set up, and initialized for event notification.

3An OnConnecting (see page 18796) event occurs after the server socket is located.

4The connection request is accepted by the server and completed by the client socket.

5An OnConnect event occurs after the connection is established.

See Also

OnAccept (see page 18795)

OnConnecting (see page 18796)

OnLookup (see page 18797)

129.1.6.3.3 TCustomSocket.OnConnecting Event

Occurs for a client socket after the server socket has been located, but before the connection is established.

Pascal

```
property OnConnecting: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnecting;
```

Description

Write an OnConnecting event handler for a client socket to take specific action just before the connection to a server socket is established. This is the first opportunity to obtain the actual port and IP address of the server endpoint of the connection that is about to form.

When a client socket opens a connection, the following events occur:

1An OnLookup (see page 18797) event occurs prior to an attempt to locate the server socket.

2The Windows socket is set up, and initialized for event notification.

3An OnConnecting event occurs after the server socket is located.

4The connection request is accepted by the server and completed by the client socket.

5An OnConnect (see page 18795) event occurs after the connection is established.

See Also

OnConnect (see page 18795)

OnLookup (see page 18797)

129.1.6.3.4 TCustomSocket.OnDisconnect Event

Occurs just before a client socket closes the connection to a server socket.

Pascal

```
property OnDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDisconnect;
```

Description

Write an OnDisconnect event handler to take specific action when the connection to a server socket is about to be terminated. OnDisconnect occurs after the value of the Active (see page 18750) property is set to false, but before the actual connection is closed.

See Also

Active (see page 18750)

Close (see page 18744)

OnConnect (see page 18795)

129.1.6.3.5 TCustomSocket.OnError Event

Occurs when the socket fails in making, using, or shutting down a connection.

Pascal

```
property OnError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnError;
```

Description

Write an OnError event handler to respond to errors that arise with the socket connection. Set the ErrorCode parameter to 0 if the OnError event handler successfully deals with the error condition, to prevent an ESocketError (see page 18739) from being raised.

See Also

ESocketError

129.1.6.3.6 TCustomSocket.OnListen Event

Occurs just before a server socket is opened for listening.

Pascal

```
property OnListen: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnListen;
```

Description

Write an OnListen event handler for a server socket to take specific action just before the socket is opened for listening. OnListen occurs after Address (see page 18750) and Port (see page 18751) have been bound to the socket connection, but before it is opened. This is the last opportunity to make changes to the socket endpoint before it is opened for listening. Use the SocketHandle property of the Socket parameter for Windows API calls that change the socket.

Server sockets open a socket connection for listening to establish a queue to hold client requests. The connection to a client socket is completed when a request from the queue is accepted.

See Also

Accept (see page 18803)

Address (see page 18750)

Port (see page 18751)

129.1.6.3.7 TCustomSocket.OnLookup Event

Occurs when a client socket is about to look up the server socket with which it wants to connect.

Pascal

```
property OnLookup: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnLookup;
```

Description

Write an OnLookup event handler for a client socket to take specific action just before attempting to locate a server socket. This is the first opportunity to make Windows API calls that affect the client properties of the socket, such as specifying a particular port number. Use the SocketHandle property of the Socket parameter for Windows API calls.

When a client socket opens a connection, the following events occur:

1An OnLookup event occurs prior to an attempt to locate the server socket.

2The Windows socket is set up, and initialized for event notification.

3An OnConnecting (see page 18796) event occurs after the server socket is located.

4The connection request is accepted by the server and completed by the client socket.

5An OnConnect (see page 18795) event occurs after the connection is established.

Note: Changing the Address (see page 18750), Host (see page 18751), Port (see page 18751), or Service (see page 18752) properties of socket component in an OnLookup event handler has no effect on the address or port used to locate a server socket. These properties must be correct before the Open (see page 18747) method is called.

See Also

Address (see page 18750)

Host (see page 18751)

OnConnect (see page 18795)

OnConnecting (see page 18796)

Open (see page 18747)

Port (see page 18751)

Service (see page 18752)

129.1.6.3.8 TCustomSocket.OnRead Event

Occurs when a client socket should read information from the socket connection.

Pascal

```
property OnRead: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnRead;
```

Description

Write an OnRead event handler to read from the socket connection. If the socket is a blocking socket, use a TWinSocketStream (see page 18876) object to read from the connection. Otherwise, u Use the methods of the Socket parameter to perform the actual reading.

Note: Non-blocking sockets do not always receive an OnRead event for the last bit of data passed over the connection. When using a non-blocking socket, check for any unread data in the OnDisconnect (see page 18796) event to make sure that everything is handled.

See Also

receivebuf (see page 18812)

ReceiveText (see page 18812)

TWinSocketStream (see page 18876)

OnDisconnect (see page 18796)

129.1.6.3.9 TCustomSocket.OnWrite Event

Occurs when a client socket should write information to the socket connection.

Pascal

```
property OnWrite: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnWrite;
```

Description

Write an OnWrite to write from the socket connection. If the socket is a blocking socket, use a TWinSocketStream (see page 18876) object to write to the connection. Otherwise, u Use the methods of the Socket parameter to perform the actual writing.

See Also

SendBuf (see page 18813)

sendstream (see page 18814)

SendStreamThenDrop (see page 18814)

SendText (see page 18814)

TWinSocketStream (see page 18876)

129.1.7 TCustomWinSocket Class

TCustomWinSocket is the base class for all Windows socket objects.

Class Hierarchy

Pascal

```
TCustomWinSocket = class;
```

C++

```
class TCustomWinSocket;
```

File

ScktComp

Description

TCustomWinSocket introduces properties, events, and methods to describe an endpoint in a Windows socket connection. Descendants of TCustomWinSocket are used by socket components to manage the Windows socket API calls and to store information about a socket communication link.

A Windows socket encapsulates a set of communication protocols to allow the application to connect to other machines for reading or writing information. Windows sockets provide connections based on the TCP/IP protocol. They also allow connections that use the Xerox Network System (see page 22079) (XNS), Digital's DECnet protocol, or Novell's IPX/SPX family. Sockets (see page 20039) allow an application to form connections to other machines without being concerned with the details of the

actual networking software.

Descendants of TCustomWinSocket represent different endpoints in socket connections. TClientWinSocket (see page 18766) represents the client endpoint of a socket communication link to a server socket. TServerWinSocket (see page 18858) represents the server endpoint of a listening connection. TServerClientWinSocket (see page 18840) represents the server endpoint of a socket communication link to a client socket.

See Also

TAbstractSocket (see page 18740)

TClientWinSocket (see page 18766)

TServerClientWinSocket (see page 18840)

TServerWinSocket (see page 18858)

129.1.7.1 TCustomWinSocket Members

The following tables list the members exposed by TCustomWinSocket.

Protected Methods

	Name	Description
	AsyncInitSocket (see page 18803)	Looks up information or forms socket connections asynchronously.
	DoListen (see page 18806)	Opens a listening connection.
	DoOpen (see page 18806)	Opens a connection to a remote socket.
	Error (see page 18806)	Generates an OnErrorEvent (see page 18824) event.
	Event (see page 18807)	This is Event, a member of class TCustomWinSocket.
	InitSocket (see page 18807)	Generates a windows Internet socket address structure from a description of the desired socket port.
	SetAsyncStyles (see page 18815)	Sets the AsyncStyles property.

Public Events

	Name	Description
	OnErrorEvent (see page 18824)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnSocketEvent (see page 18824)	Occurs when the Windows socket receives notification of asynchronous events.

Public Methods

	Name	Description
	Accept (see page 18803)	Provides the interface for a method that accepts a connection to a client socket.
	Close (see page 18804)	Shuts down the socket connection if it is open.
	Connect (see page 18805)	Provides the interface for a method that responds to connection notifications from client sockets.
	Create (see page 18801)	Creates an instance of TCustomWinSocket (see page 18799) from a Windows socket handle.
	DefaultHandler (see page 18805)	Processes message records.
	Destroy (see page 18802)	Destroys an instance of TCustomWinSocket (see page 18799).
	Disconnect (see page 18805)	Shuts down the socket connection represented by a Windows socket handle.
	Listen (see page 18808)	Opens a listening connection for a server socket.
	Lock (see page 18809)	Starts a critical section, blocking all other execution threads until the Unlock (see page 18815) method is called.
	LookupName (see page 18809)	Generates the portion of a Windows Internet socket address structure that describes the host.
	LookupService (see page 18810)	Returns the port number associated with a specific service.
	Open (see page 18810)	Opens a connection to a remote socket.
	Read (see page 18811)	Generates an OnSocketEvent (see page 18824) of type seRead in response to notifications that the socket connection is ready.

	ReceiveBuf (see page 18812)	This is ReceiveBuf, a member of class TCustomWinSocket.
	ReceiveLength (see page 18812)	Returns the number of bytes ready to be sent over the socket connection.
	ReceiveText (see page 18812)	Reads a string from the socket connection.
	SendBuf (see page 18813)	Writes Count bytes to the socket connection from the Buf parameter.
	SendStream (see page 18814)	This is SendStream, a member of class TCustomWinSocket.
	SendStreamThenDrop (see page 18814)	Writes all the information that can be read from the AStream parameter to the socket connection and then terminates the connection.
	SendText (see page 18814)	Writes the string S to the socket connection.
	Unlock (see page 18815)	Reenables execution in other threads after a critical section begun by the Lock (see page 18809) method.
V	Write (see page 18816)	Generates an OnSocketEvent (see page 18824) of type seWrite in response to notifications that the socket connection is ready.

Public Properties

	Name	Description
R	Addr (see page 18817)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the socket port.
	ASyncStyles (see page 18817)	Determines which asynchronous events the socket can receive.
R	Connected (see page 18818)	Indicates whether the socket connection is open and available for communication with other machines.
	Data (see page 18818)	Points to application-specific data associated with the socket.
R	Handle (see page 18819)	Specifies the handle of the window that receives notifications from the Windows socket.
R	LocalAddress (see page 18819)	Specifies the IP address of the local system that was bound to the socket connection.
R	LocalHost (see page 18820)	Specifies the name of the local system.
R	LocalPort (see page 18820)	Specifies the ID number of the port used by the socket connection.
R	LookupState (see page 18821)	Indicates any asynchronous activities occurring while opening a connection.
R	RemoteAddr (see page 18821)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the remote socket port.
R	RemoteAddress (see page 18822)	Specifies the IP address of the remote system that was bound to the socket connection.
R	RemoteHost (see page 18822)	Specifies the name of the remote system.
R	RemotePort (see page 18823)	Specifies the ID number of the port used by the RemoteHost (see page 18822) to form the socket connection.
R	SocketHandle (see page 18823)	Specifies the Windows handle for the socket.

129.1.7.2 TCustomWinSocket.Create Constructor

Creates an instance of TCustomWinSocket (see page 18799) from a Windows socket handle.

Pascal

```
constructor Create(ASocket: TSocket);
```

C++

```
__fastcall TCustomWinSocket(TSocket ASocket);
```

Description

Call Create to create a Windows socket object. Most Windows socket objects are created automatically by the socket component that uses them. Applications may create their own Windows socket objects in an OnGetSocket event handler for a server socket component.

After calling the inherited constructor, Create

Makes sure that Winsock.dll is initialized.

Allocates helper objects to manage multithreading issues.

Initializes ASyncStyles (see page 18817) to [asRead, asWrite, asConnect, asClose]

Initializes the SocketHandle (see page 18823) property to the ASocket parameter and sets the Connected (see page 18818) property to reflect whether ASocket is the handle of a valid open Windows socket.

Initializes the Addr (see page 18817) property.

See Also

Addr (see page 18817)

ASyncStyles (see page 18817)

Connected (see page 18818)

SocketHandle (see page 18823)

OnGetSocket (see page 18787)

129.1.7.3 TCustomWinSocket.Destroy Destructor

Destroys an instance of TCustomWinSocket (see page 18799).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWinSocket();
```

Description

Most applications do not need to explicitly free Windows socket objects. They are freed by the socket components that use them.

See Also

Free (see page 22300)

TCustomSocket (see page 18789)

129.1.7.4 TCustomWinSocket Methods

The methods of the TCustomWinSocket class are listed here.

Protected Methods

	Name	Description
	AsyncInitSocket (see page 18803)	Looks up information or forms socket connections asynchronously.
	DoListen (see page 18806)	Opens a listening connection.
	DoOpen (see page 18806)	Opens a connection to a remote socket.
	Error (see page 18806)	Generates an OnErrorEvent (see page 18824) event.
	Event (see page 18807)	This is Event, a member of class TCustomWinSocket.
	InitSocket (see page 18807)	Generates a windows Internet socket address structure from a description of the desired socket port.
	SetAsyncStyles (see page 18815)	Sets the AsyncStyles property.

Public Methods

	Name	Description
V	Accept (see page 18803)	Provides the interface for a method that accepts a connection to a client socket.
	Close (see page 18804)	Shuts down the socket connection if it is open.

	Connect (see page 18805)	Provides the interface for a method that responds to connection notifications from client sockets.
	DefaultHandler (see page 18805)	Processes message records.
	Disconnect (see page 18805)	Shuts down the socket connection represented by a Windows socket handle.
	Listen (see page 18808)	Opens a listening connection for a server socket.
	Lock (see page 18809)	Starts a critical section, blocking all other execution threads until the Unlock (see page 18815) method is called.
	LookupName (see page 18809)	Generates the portion of a Windows Internet socket address structure that describes the host.
	LookupService (see page 18810)	Returns the port number associated with a specific service.
	Open (see page 18810)	Opens a connection to a remote socket.
	Read (see page 18811)	Generates an OnSocketEvent (see page 18824) of type seRead in response to notifications that the socket connection is ready.
	ReceiveBuf (see page 18812)	This is ReceiveBuf, a member of class TCustomWinSocket.
	ReceiveLength (see page 18812)	Returns the number of bytes ready to be sent over the socket connection.
	ReceiveText (see page 18812)	Reads a string from the socket connection.
	SendBuf (see page 18813)	Writes Count bytes to the socket connection from the Buf parameter.
	SendStream (see page 18814)	This is SendStream, a member of class TCustomWinSocket.
	SendStreamThenDrop (see page 18814)	Writes all the information that can be read from the AStream parameter to the socket connection and then terminates the connection.
	SendText (see page 18814)	Writes the string S to the socket connection.
	Unlock (see page 18815)	Reenables execution in other threads after a critical section begun by the Lock (see page 18809) method.
	Write (see page 18816)	Generates an OnSocketEvent (see page 18824) of type seWrite in response to notifications that the socket connection is ready.

129.1.7.4.1 TCustomWinSocket.Accept Method

Provides the interface for a method that accepts a connection to a client socket.

Pascal

```
procedure Accept(Socket: TSocket); virtual;
```

C++

```
virtual __fastcall Accept(TSocket Socket);
```

Description

Accept is called automatically in response to notifications that a client socket is requesting a connection. The Socket parameter is the Windows socket handle for the socket that receives the notification.

The Accept method for TCustomWinSocket (see page 18799) does nothing with the notification. Descendants of TCustomWinSocket (see page 18799) that represent listening server sockets override this method to accept the connection to the client socket.

Accept is only called if the ASyncStyles (see page 18817) property includes asAccept.

See Also

ASyncStyles (see page 18817)

129.1.7.4.2 TCustomWinSocket.AsyncInitSocket Method

Looks up information or forms socket connections asynchronously.

Pascal

```
procedure AsyncInitSocket(const Name: string; const Address: string; const Service: string;
Port: Word; QueueSize: Integer; Client: Boolean);
```

C++

```
__fastcall AsyncInitSocket(const AnsiString Name, const AnsiString Address, const AnsiString Service, Word Port, int QueueSize, Boolean Client);
```

Description

AsyncInitSocket is used internally to open a socket connection in non-blocking sockets. It is called by the Open (see page 18810) or Listen (see page 18808) method when that method's Block parameter is false.

The Name parameter is the host name for the socket. The Address parameter is the IP address. The Service parameter is the service for which the socket is to used. The Port parameter is the port number for the socket. The QueueSize parameter indicates the number of client requests a listening connection can hold while waiting to form connections. The Client parameter indicates whether the socket address structure is to be used by a client socket to describe the destination of a connection or by a server socket to describe the listening connection.

See Also

ASyncStyles (see page 18817)

ServerType (see page 18869)

ClientType (see page 18769)

Addr (see page 18817)

Listen (see page 18808)

LookupName (see page 18809)

LookupService (see page 18810)

Open (see page 18810)

RemoteAddr (see page 18821)

Address (see page 18750)

Host (see page 18751)

Port (see page 18751)

Service (see page 18752)

129.1.7.4.3 TCustomWinSocket.Close Method

Shuts down the socket connection if it is open.

Pascal

```
procedure Close;
```

C++

```
__fastcall Close();
```

Description

Call Close to terminate an open socket connection. Close triggers an OnSocketEvent (see page 18824) of type seDisconnect before the connection is shut down. If the Windows socket is a listening server socket, Close shuts down all open connections to client sockets before shutting down the listening connection.

See Also

Connected (see page 18818)

OnSocketEvent (see page 18824)

129.1.7.4.4 TCustomWinSocket.Connect Method

Provides the interface for a method that responds to connection notifications from client sockets.

Pascal

```
procedure Connect(Socket: TSocket); virtual;
```

C++

```
virtual __fastcall Connect(TSocket Socket);
```

Description

Connect is called automatically in response to notifications that the client socket has successfully completed a connection to a server socket. The Socket parameter is the Windows socket handle for the socket that receives the notification.

The Connect method for TCustomWinSocket (see page 18799) does nothing with the notification. Descendants of TCustomWinSocket (see page 18799) that represent client sockets override this method to respond to connection notifications.

Connect is only called if the ASyncStyles (see page 18817) property includes asConnect.

See Also

ASyncStyles (see page 18817)

129.1.7.4.5 TCustomWinSocket.DefaultHandler Method

Processes message records.

Pascal

```
procedure DefaultHandler(var Message); override;
```

C++

```
virtual __fastcall DefaultHandler( Message);
```

Description

DefaultHandler overrides the inherited method in order to process all messages that are sent to the Windows socket object. DefaultHandler passes the messages to the Windows CallWindowProc function.

See Also

TMessage

129.1.7.4.6 TCustomWinSocket.Disconnect Method

Shuts down the socket connection represented by a Windows socket handle.

Pascal

```
procedure Disconnect(Socket: TSocket); virtual;
```

C++

```
virtual __fastcall Disconnect(TSocket Socket);
```

Description

Disconnect is called by the Close (see page 18804) method to terminate an individual socket connection.

Disconnect calls the Lock (see page 18809) method to make sure the method is thread-safe. Then, it generates an

OnSocketEvent (see page 18824) of type seDisconnect. After the OnSocketEvent (see page 18824), Disconnect closes the connection. Finally, it calls Unlock (see page 18815) to end the critical section started by Lock (see page 18809).

See Also

Close (see page 18804)

Lock (see page 18809)

OnSocketEvent (see page 18824)

Unlock (see page 18815)

129.1.7.4.7 TCustomWinSocket.DoListen Method

Opens a listening connection.

Pascal

```
procedure DoListen(QueueSize: Integer);
```

C++

```
__fastcall DoListen(int QueueSize);
```

Description

The socket uses DoListen internally to open a listening connection with a queue that can hold QueueSize client connection requests. DoListen handles those tasks that are common to opening a listening connection synchronously or asynchronously.

Do not call DoListen to open a listening connection. Instead, call the public Listen (see page 18808) method, which orchestrates the opening of a connection either synchronously or asynchronously, depending on the socket type.

See Also

Listen (see page 18808)

129.1.7.4.8 TCustomWinSocket.DoOpen Method

Opens a connection to a remote socket.

Pascal

```
procedure DoOpen;
```

C++

```
__fastcall DoOpen();
```

Description

The socket uses DoOpen internally to open a connection to a remote socket. DoOpen handles those tasks that are common to establishing a connection synchronously or asynchronously.

Do not call DoOpen to open a connection. Instead, call the public Open (see page 18810) method, which orchestrates the opening of a connection either synchronously or asynchronously, depending on the socket type.

See Also

Open (see page 18810)

129.1.7.4.9 TCustomWinSocket.Error Method

Generates an OnErrorEvent (see page 18824) event.

Pascal

```
procedure Error(Socket: TCustomWinSocket; ErrorEvent: TErrorEvent; var ErrorCode: Integer);
dynamic;
```

C++

```
__fastcall Error(TCustomWinSocket Socket, TErrorEvent ErrorEvent, int ErrorCode);
```

Description

Error is called automatically when the Windows socket object receives error notifications. It generates the OnErrorEvent (see page 18824) event. When overriding this method, be sure to call the inherited method, because socket components rely on the OnErrorEvent (see page 18824) event for their own error handling.

The Socket parameter indicates the Windows socket object that encounters the error condition.

The ErrorEvent parameter indicates what the socket was trying to do when the error occurred. It has one of the following values:

Value	Meaning
eeGeneral	The socket received an error message that does not fit into any of the following categories.
eeSend	An error occurred when trying to write to the socket connection.
eeReceive	An error occurred when trying to read from the socket connection.
eeConnect	For client sockets, this indicates that the client socket can't locate the server, or that a problem on the server prevents the opening of a connection. For server sockets, this indicates that a client connection request that has already been accepted could not be completed.
eeDisconnect	An error occurred when trying to close a connection.
eeAccept	For server sockets only, this indicates that a problem occurred when trying to accept a client connection request.

The ErrorCode parameter is the error code received by the Windows socket object. Changing this value to 0 in the Error method prevents the socket from raising an exception. For information on possible error codes, see the Microsoft documentation on Windows sockets.

See Also

OnErrorEvent (see page 18824)

129.1.7.4.10 TCustomWinSocket.Event Method

Pascal

```
procedure Event(Socket: TCustomWinSocket; SocketEvent: TSocketEvent); dynamic;
```

C++

```
__fastcall Event(TCustomWinSocket Socket, TSocketEvent SocketEvent);
```

Description

This is Event, a member of class TCustomWinSocket.

129.1.7.4.11 TCustomWinSocket.InitSocket Method

Generates a windows Internet socket address structure from a description of the desired socket port.

Pascal

```
function InitSocket(const Name: string; const Address: string; const Service: string; Port:
Word; Client: Boolean): TSockAddrIn;
```

C++

```
__fastcall TSockAddrIn InitSocket(const AnsiString Name, const AnsiString Address, const
AnsiString Service, Word Port, Boolean Client);
```

Description

Applications can not call InitSocket. It is called by the Open (see page 18810) or Listen (see page 18808) method to obtain a Windows Internet socket address structure for the parameters obtained from a socket component.

The Name parameter is the host name for the socket.The Address parameter is the IP address. The Service parameter is the service for which the socket is to used. The Port parameter is the port number for the socket. The Client parameter indicates whether the socket address structure is to be used by a client socket to describe the destination of a connection or by a server socket to describe the listening connection.

The host portion of the Windows Internet socket address is obtained from the host name if it is given. Otherwise the Address parameter is used. If Client is true, either the Name or the Address parameter must be specified. If Client is false, they may both be left blank to specify a server that listens on multiple addresses.

If port number for the Windows Internet socket address is obtained from the Service parameter if it is given. Otherwise, the Port parameter is used. One of these two parameters must be specified.

See Also

Addr (see page 18817)

Listen (see page 18808)

LookupName (see page 18809)

LookupService (see page 18810)

Open (see page 18810)

RemoteAddr (see page 18821)

Address (see page 18750)

Host (see page 18751)

Port (see page 18751)

Service (see page 18752)

AsyncInitSocket (see page 18803)

129.1.7.4.12 TCustomWinSocket.Listen Method

Opens a listening connection for a server socket.

Pascal

```
procedure Listen(const Name: string; const Address: string; const Service: string; Port: Word;
QueueSize: Integer; Block: Boolean = True);
```

C++

```
__fastcall Listen(const AnsiString Name, const AnsiString Address, const AnsiString Service,
Word Port, int QueueSize, Boolean Block = True);
```

Description

Server socket components call Listen to open a listening connection.

Listen creates the Windows socket that will make the listening connection, and binds the Windows Internet socket address derived from the parameters to the socket. Listen then generates an OnSocketEvent (see page 18824) of type seListen.

Finally, Listen opens the socket as a listening connection with a queue that can hold QueueSize client requests, setting the Connected (see page 18818) property to true.

The Name, Address, Service, and Port parameters are taken from the Host, Address, Service and Port properties of the server socket component.

QueueSize indicates how many client requests the listening connection can hold. TServerSocket (see page 18843) sets the QueueSize to 5.

Block indicates whether the socket can look up the information it needs to establish a connection asynchronously. When Block is true, the socket blocks execution for each lookup instead of retrieving information about the connection using asynchronous notifications.

See Also

Connected (see page 18818)

InitSocket (see page 18807)

OnSocketEvent (see page 18824)

Address (see page 18750)

Host (see page 18751)

Port (see page 18751)

Service (see page 18752)

TServerSocket (see page 18843)

129.1.7.4.13 TCustomWinSocket.Lock Method

Starts a critical section, blocking all other execution threads until the Unlock (see page 18815) method is called.

Pascal

```
procedure Lock;
```

C++

```
__fastcall Lock();
```

Description

Call Lock before beginning a section of code that is not thread-safe. Call Unlock (see page 18815) when the critical section of code is completed. Unless absolutely necessary, do not use Lock to lock lengthy operations such as reading and writing over a blocking connection. Locking lengthy operations can severely impact performance, because all execution threads in the application that use the Windows socket object must wait until Unlock (see page 18815) is called.

See Also

TThread (see page 3108)

Unlock (see page 18815)

129.1.7.4.14 TCustomWinSocket.LookupName Method

Generates the portion of a Windows Internet socket address structure that describes the host.

Pascal

```
function LookupName(const name: string): TInAddr;
```

C++

```
__fastcall TInAddr LookupName(const AnsiString name);
```

Description

Call LookupName to obtain the Internet address associated with the system identified by the name parameter. LookupName is used by InitSocket (see page 18807) to generate the portion of the Windows Internet socket address structure that identifies the host.

See Also

InitSocket (see page 18807)

Host (see page 18751)

129.1.7.4.15 TCustomWinSocket.LookupService Method

Returns the port number associated with a specific service.

Pascal

```
function LookupService(const service: string): Integer;
```

C++

```
__fastcall int LookupService(const AnsiString service);
```

Description

Call LookupService to obtain the standard port number for the service named by the service parameter. Windows provides a number of standard service names such as ftp, http, finger, or time. Each standard service is associated, by convention, with a specific port number. These standard port numbers allow client sockets to indicate the desired service by requesting a specific port number. They allow server sockets to listen on an expected port number for the service offered. Servers can specify additional services and their associated ports in a SERVICES file. For more information on the SERVICES file, see the Microsoft documentation for Windows sockets.

InitSocket (see page 18807) calls LookupService to obtain the port number for the Service specified by a socket component.

See Also

InitSocket (see page 18807)

Service (see page 18752)

129.1.7.4.16 TCustomWinSocket.Open Method

Opens a connection to a remote socket.

Pascal

```
procedure Open(const Name: string; const Address: string; const Service: string; Port: Word;
Block: Boolean = True);
```

C++

```
__fastcall Open(const AnsiString Name, const AnsiString Address, const AnsiString Service,
Word Port, Boolean Block = True);
```

Description

Client socket components call Open to open a connection to a server socket.

Open creates the Windows socket that will connect to the server socket, and generates an OnSocketEvent (see page 18824) of type seLookup. After the OnSocketEvent (see page 18824), Open binds the Windows Internet socket address derived from

the parameters to the socket, locating the server socket. Open then generates an OnSocketEvent (see page 18824) of type seConnecting. When the server socket accepts the connection, Open completes the connection to the server socket. Then it generates an OnSocketEvent (see page 18824) of type seConnect. After the OnSocketEvent (see page 18824), the Connected (see page 18818) property is set to true.

The Name, Address, Service, and Port parameters are taken from the Host, Address, Service and Port properties of the client socket component.

Block indicates whether the socket can look up the information it needs to establish a connection asynchronously. When Block is true, the socket blocks execution for each lookup instead of retrieving information about the connection using asynchronous notifications.

See Also

Connected (see page 18818)

InitSocket (see page 18807)

OnSocketEvent (see page 18824)

Address (see page 18750)

Host (see page 18751)

Port (see page 18751)

Service (see page 18752)

TClientSocket (see page 18752)

129.1.7.4.17 TCustomWinSocket.Read Method

Generates an OnSocketEvent (see page 18824) of type seRead in response to notifications that the socket connection is ready.

Pascal

```
procedure Read(Socket: TSocket); virtual;
```

C++

```
virtual __fastcall Read(TSocket Socket);
```

Description

Read is called in response to asynchronous notifications that the socket connection is ready for reading. The Windows socket can then read from the connection in the OnSocketEvent (see page 18824) event handler or the socket component can read from the connection in an OnRead or OnClientRead event handler. The event handler can use the ReceiveBuf (see page 18812) method or the ReceiveText (see page 18812) method to read the information from the connection.

Note: Read is only called for asynchronous read notifications in a non-blocking socket. Blocking sockets must use a TWinSocketStream (see page 18876) object for reading. The TWinSocketStream (see page 18876) object waits for the remote socket to be ready before transferring information.

See Also

ASyncStyles (see page 18817)

OnSocketEvent (see page 18824)

receivebuf (see page 18812)

ReceiveText (see page 18812)

OnRead (see page 18798)

OnClientRead (see page 18786)

TWinSocketStream (see page 18876)

129.1.7.4.18 TCustomWinSocket.ReceiveBuf Method

Pascal

```
function ReceiveBuf(var Buf; Count: Integer): Integer;
```

C++

```
__fastcall int ReceiveBuf( Buf, int Count);
```

Description

This is ReceiveBuf, a member of class TCustomWinSocket.

129.1.7.4.19 TCustomWinSocket.ReceiveLength Method

Returns the number of bytes ready to be sent over the socket connection.

Pascal

```
function ReceiveLength: Integer;
```

C++

```
__fastcall int ReceiveLength();
```

Description

Call ReceiveLength to determine the amount of information to read over the socket connection in response to an asynchronous read notification.

Note: ReceiveLength is not guaranteed to be accurate for streaming socket connections.

See Also

ASyncStyles (see page 18817)

receivebuf (see page 18812)

129.1.7.4.20 TCustomWinSocket.ReceiveText Method

Reads a string from the socket connection.

Pascal

```
function ReceiveText: string;
```

C++

```
__fastcall AnsiString ReceiveText();
```

Description

Use ReceiveText to read a string from the socket connection in the OnSocketEvent (see page 18824) event handler of a Windows socket object or in the OnRead or OnClientRead event handler of a socket component. ReceiveText returns the string that was read.

ReceiveText only works in response to a read notification to a non-blocking windows socket. Blocking sockets must use a TWinSocketStream (see page 18876) for reading. The TWinSocketStream (see page 18876) object waits for the remote

socket to be ready before transferring information.

See Also

ASyncStyles (see page 18817)

OnSocketEvent (see page 18824)

receivebuf (see page 18812)

OnRead (see page 18798)

OnClientRead (see page 18786)

TWinSocketStream (see page 18876)

129.1.7.4.21 TCustomWinSocket.SendBuf Method

Writes Count bytes to the socket connection from the Buf parameter.

Pascal

```
function SendBuf(var Buf; Count: Integer): Integer;
```

C++

```
__fastcall int SendBuf( Buf, int Count);
```

Description

Use SendBuf to write to the socket connection. Call this method from the OnSocketEvent (see page 18824) event handler of a Windows socket object or in the OnWrite or OnClientWrite event handler of a socket component. Alternately, Use SendBuf to write when a connection is first formed when the socket does not expect notification that the socket on the other end of the connection is expecting to read.

For non-blocking sockets, the data is sent to the WinSock DLL which has it's own internal buffers. If the WinSock can accept additional data, SendBuf returns immediately with the number of bytes queued. If the WinSock internal buffer space is not able to accept the buffer being sent, SendBuf returns -1 and no data is queued at all. In this case, wait a bit for the WinSock to have a chance to send out already-queued data; then try again.

For blocking sockets, SendBuf returns the number of bytes actually written.

If an error occurs while writing to the connection, SendBuf terminates the connection and raises an ESocketError (see page 18739) exception.

See Also

ESocketError

OnSocketEvent (see page 18824)

sendstream (see page 18814)

SendText (see page 18814)

OnWrite (see page 18799)

OnClientWrite (see page 18786)

TWinSocketStream (see page 18876)

129.1.7.4.22 TCustomWinSocket.SendStream Method

Pascal

```
function SendStream(AStream: TStream): Boolean;
```

C++

```
__fastcall Boolean SendStream(TStream AStream);
```

Description

This is SendStream, a member of class TCustomWinSocket.

129.1.7.4.23 TCustomWinSocket.SendStreamThenDrop Method

Writes all the information that can be read from the AStream parameter to the socket connection and then terminates the connection.

Pascal

```
function SendStreamThenDrop(AStream: TStream): Boolean;
```

C++

```
__fastcall Boolean SendStreamThenDrop(TStream AStream);
```

Description

Use SendStreamThenDrop to write to a socket connection when the connection is no longer needed after the writing is complete. SendStream (see page 18814) reads information from the stream indicated by AStream and writes it to the socket connection, shutting down the connection when it is done. The value returned by SendStream (see page 18814) indicates whether any information was successfully written to the connection.

Note: The Stream passed as a parameter to SendStream (see page 18814) becomes "owned" by the windows socket object. The Windows socket object frees the stream when it is finished with it. Do not attempt to free the stream after it has been passed as a parameter.

See Also

Disconnect (see page 18805)

SendBuf (see page 18813)

sendstream (see page 18814)

SendText (see page 18814)

TStream (see page 2973)

TWinSocketStream (see page 18876)

129.1.7.4.24 TCustomWinSocket.SendText Method

Writes the string S to the socket connection.

Pascal

```
function SendText(const S: string): Integer;
```

C++

```
__fastcall int SendText(const AnsiString S);
```

Description

Use SendText to write a string to the socket connection. The writing may occur in the OnSocketEvent (see page 18824) event handler of a Windows socket object or in the OnWrite or OnClientWrite event handler of a socket component. Alternately, SendText may write from a socket that is expected to write to the connection without a notification to signal the connection's readiness to read. If an error occurs while writing to the connection, SendText terminates the connection and raises an ESocketError (see page 18739) exception.

SendText returns the number of bytes sent. Note that this may be less than the length of the string S if the socket is nonblocking.

See Also

ESocketError

OnSocketEvent (see page 18824)

SendBuf (see page 18813)

sendstream (see page 18814)

OnWrite (see page 18799)

OnClientWrite (see page 18786)

TWinSocketStream (see page 18876)

129.1.7.4.25 TCustomWinSocket.SetAsyncStyles Method

Sets the AsyncStyles property.

Pascal

```
procedure SetAsyncStyles(Value: TASyncStyles);
```

C++

```
__fastcall SetAsyncStyles(TASyncStyles Value);
```

Description

SetAsyncStyles implements the AsyncStyles property. Override SetAsyncStyles to limit the value of AsyncStyles to notifications that are appropriate to a particular type of Windows socket.

See Also

ASyncStyles (see page 18817)

129.1.7.4.26 TCustomWinSocket.Unlock Method

Reenables execution in other threads after a critical section begun by the Lock (see page 18809) method.

Pascal

```
procedure Unlock;
```

C++

```
__fastcall Unlock();
```

Description

Call Unlock at the end of a section of code that is not thread-safe. Every call to Unlock should match a call to the Lock (see page 18809) method at the beginning of the critical section of code.

See Also

Lock (see page 18809)

TThread (see page 3108)

129.1.7.4.27 TCustomWinSocket.Write Method

Generates an OnSocketEvent (see page 18824) of type seWrite in response to notifications that the socket connection is ready.

Pascal

```
procedure Write(Socket: TSocket); virtual;
```

C++

```
virtual __fastcall Write(TSocket Socket);
```

Description

Write is called in response to asynchronous notifications that the socket connection is ready for the socket to send information. The Windows socket can write to the connection in the OnSocketEvent (see page 18824) event handler or the socket component can write to the connection in an OnWrite or OnClientWrite event handler. The event handler can use the SendBuf (see page 18813) method, the SendStream (see page 18814) method, or the SendText (see page 18814) method to write the information to the connection.

Note: Write is only called for asynchronous write notifications in a non-blocking socket. Blocking sockets must use their own mechanisms for determining when the connection is ready for the socket to write.

See Also

ASyncStyles (see page 18817)

OnSocketEvent (see page 18824)

SendBuf (see page 18813)

sendstream (see page 18814)

SendText (see page 18814)

OnWrite (see page 18799)

OnClientWrite (see page 18786)

TWinSocketStream (see page 18876)

129.1.7.5 TCustomWinSocket Properties

The properties of the TCustomWinSocket class are listed here.

Public Properties

	Name	Description
R	Addr (see page 18817)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the socket port.
	ASyncStyles (see page 18817)	Determines which asynchronous events the socket can receive.
R	Connected (see page 18818)	Indicates whether the socket connection is open and available for communication with other machines.
	Data (see page 18818)	Points to application-specific data associated with the socket.

R	Handle (see page 18819)	Specifies the handle of the window that receives notifications from the Windows socket.
R	LocalAddress (see page 18819)	Specifies the IP address of the local system that was bound to the socket connection.
R	LocalHost (see page 18820)	Specifies the name of the local system.
R	LocalPort (see page 18820)	Specifies the ID number of the port used by the socket connection.
R	LookupState (see page 18821)	Indicates any asynchronous activities occurring while opening a connection.
R	RemoteAddr (see page 18821)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the remote socket port.
R	RemoteAddress (see page 18822)	Specifies the IP address of the remote system that was bound to the socket connection.
R	RemoteHost (see page 18822)	Specifies the name of the remote system.
R	RemotePort (see page 18823)	Specifies the ID number of the port used by the RemoteHost (see page 18822) to form the socket connection.
R	SocketHandle (see page 18823)	Specifies the Windows handle for the socket.

129.1.7.5.1 TCustomWinSocket.Addr Property

Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the socket port.

Pascal

```
property Addr: TSockAddrIn;
```

C++

```
__property TSockAddrIn Addr;
```

Description

Use Addr to represent the port used by the socket when working directly with Windows socket API calls. For client sockets, the value of Addr represents the target remote port. For server sockets, Addr is determined by the values of the LocalAddress (see page 18819) and LocalPort (see page 18820) properties.

See Also

LocalAddress (see page 18819)

LocalPort (see page 18820)

RemoteAddr (see page 18821)

129.1.7.5.2 TCustomWinSocket.ASyncStyles Property

Determines which asynchronous events the socket can receive.

Pascal

```
property ASyncStyles: TAsyncStyles;
```

C++

```
__property TAsyncStyles ASyncStyles;
```

Description

Use ASyncStyles to determine what sorts of notifications the socket needs to respond to. Set ASyncStyles to change the notifications that the socket receives from the socket connection. ASyncStyles is a set drawn from the following values:

Value	Meaning
asRead	The socket receives notification that the connection is ready for reading.

asWrite	The socket receives notification that the connection is ready for writing.
asOOB	The socket receives notification when out-of-band data arrives.
asAccept	The socket receives notification when another socket requests a connection.
asConnect	The socket receives notification when a communication link to another socket is opened.
asClose	The socket receives notification when a communication link to another socket is terminated.

Any notification specified by ASyncStyles arrives as a window message to the Handle (see page 18819) property.

Note: If the socket is a non-blocking socket, then ASyncStyles should include asRead and asWrite so that the socket will be informed of asynchronous reading and writing events.

See Also

Handle (see page 18819)

ClientType (see page 18769)

ServerType (see page 18869)

129.1.7.5.3 TCustomWinSocket.Connected Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Read (see page 18811) Connected before trying to use or change the Windows socket connection. Connected indicates whether the socket connection has been established. When Connected is true, the Windows socket is open and available for use. When Connected is false, the Windows socket is closed and can be changed.

To open a client Windows socket, use the Open (see page 18810) method. To open a server Windows socket, use the Listen (see page 18808) method. The Windows socket that represents the server endpoint of a connection to a client socket is opened in its constructor.

To close a Windows socket, use the Close (see page 18804) method.

See Also

Open (see page 18810)

Close (see page 18804)

Listen (see page 18808)

Active (see page 18750)

129.1.7.5.4 TCustomWinSocket.Data Property

Points to application-specific data associated with the socket.

Pascal

```
property Data: Pointer;
```

C++

```
__property void * Data;
```

Description

Use Data to associate information specific to the socket connection with the Windows socket object. For example, Data can be used to store access and authentication information used by a server socket to evaluate client connection requests.

Data can store information in a thread-safe manner that would otherwise require the use of global variables. This is useful given the multi-threaded nature of many client-server applications.

129.1.7.5.5 TCustomWinSocket.Handle Property

Specifies the handle of the window that receives notifications from the Windows socket.

Pascal

```
property Handle: HWnd;
```

C++

```
__property HWnd Handle;
```

Description

Use Handle to send messages to the Windows socket or to subclass the Window procedure that receives messages of asynchronous events from the socket connection.

See Also

ASyncStyles (see page 18817)

129.1.7.5.6 TCustomWinSocket.LocalAddress Property

Specifies the IP address of the local system that was bound to the socket connection.

Pascal

```
property LocalAddress: string;
```

C++

```
__property AnsiString LocalAddress;
```

Description

LocalAddress is a string of four numeric (byte) values in the standard Internet dot notation, such as

```
123.197.1.2
```

Each address identifies the system and indicates the types of interfaces supported by the socket. A single system (machine) can support multiple addresses. LocalAddress is the address from among the possible addresses supported by the local system that is bound to the socket connection.

A single LocalAddress can be used simultaneously by multiple socket connections. Each of those connections must have a unique LocalPort (see page 18820) number to distinguish it from other connections to the same LocalAddress.

See Also

LocalHost (see page 18820)

LocalPort (see page 18820)

RemoteAddress (see page 18822)

Address (see page 18750)

129.1.7.5.7 TCustomWinSocket.LocalHost Property

Specifies the name of the local system.

Pascal

```
property LocalHost: string;
```

C++

```
__property AnsiString LocalHost;
```

Description

Use LocalHost to get the name of the system that is running the application that uses the Windows socket. LocalHost is a string containing the domain name and service of the local socket endpoint, such as

`http://www.wSite.Com`

LocalHost provides an alias for the local IP address given in the LocalAddress (see page 18819) property. Most Intranets provide host names for the IP addresses of systems on the net. On Windows 95 and NT machines, if a host name is not available, create one for the local IP address by entering the name into the HOSTS file. See the Microsoft documentation on Windows sockets for more information on the HOSTS file.

See Also

LocalAddress (see page 18819)

RemoteHost (see page 18822)

Host (see page 18751)

129.1.7.5.8 TCustomWinSocket.LocalPort Property

Specifies the ID number of the port used by the socket connection.

Pascal

```
property LocalPort: Integer;
```

C++

```
__property int LocalPort;
```

Description

Use LocalPort to determine the port number that is bound to the socket connection. For server endpoints of listening connections, LocalPort is the ID associated with the service the server socket provides. For client or server endpoints of connections to another socket, LocalPort is usually an arbitrary value picked when the connection was made.

Port numbers allow a single system, identified by the LocalAddress (see page 18819) property, to host multiple connections simultaneously. Each combination of LocalPort and LocalAddress (see page 18819) can only be bound to a single socket connection. This combination is represented by the Addr (see page 18817) property.

Many values of LocalPort are associated by convention with a particular service such as ftp or http. To determine whether the value of LocalPort is associated with a specific service, use the LookupService (see page 18810) method.

See Also

Addr (see page 18817)

LocalAddress (see page 18819)

LookupService (see page 18810)

RemotePort (see page 18823)

129.1.7.5.9 TCustomWinSocket.LookupState Property

Indicates any asynchronous activities occurring while opening a connection.

Pascal

```
property LookupState: TLookupState;
```

C++

```
__property TLookupState LookupState;
```

Description

LookupState is used internally to coordinate asynchronous activities while opening a connection. The AsyncInitSocket (see page 18803) method sets the value of LookupState to indicate when it is looking up various types of information. The following table lists the possible values:

Value	Meaning
lsIdle	The socket is not currently looking up any information asynchronously.
lsLookupAddress	The socket is looking up an IP address asynchronously and has not yet received the result.
lsLookupService	The socket is looking up a Service asynchronously and has not yet received the result.

See Also

AsyncInitSocket (see page 18803)

Open (see page 18810)

Listen (see page 18808)

129.1.7.5.10 TCustomWinSocket.RemoteAddr Property

Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the remote socket port.

Pascal

```
property RemoteAddr: TSockAddrIn;
```

C++

```
__property TSockAddrIn RemoteAddr;
```

Description

When working directly with Windows socket API calls, use RemoteAddr to represent the port used by the socket on the other end of the connection. The value of RemoteAddr is determined by the values of the RemoteAddress (see page 18822) and RemotePort (see page 18823) properties.

See Also

Addr (see page 18817)

RemoteAddress (see page 18822)

RemotePort (see page 18823)

129.1.7.5.11 TCustomWinSocket.RemoteAddress Property

Specifies the IP address of the remote system that was bound to the socket connection.

Pascal

```
property RemoteAddress: string;
```

C++

```
__property AnsiString RemoteAddress;
```

Description

RemoteAddress is a string of four numeric (byte) values in the standard Internet dot notation, such as

`123.197.1.2`

Each address identifies the system on the other end of the socket connection and indicates the types of interfaces supported by the remote socket. A single system can support multiple addresses. RemoteAddress is the address from among the possible addresses supported by the remote system that is bound to the socket connection.

A single RemoteAddress may be participating in multiple socket connections. Each of those connections has a unique RemotePort (see page 18823) number to distinguish it from other connections to the same RemoteAddress. The RemoteAddr (see page 18821) property uniquely identifies the RemoteAddress and RemotePort (see page 18823) combination used by the socket connection.

See Also

LocalAddress (see page 18819)

RemoteAddr (see page 18821)

RemoteHost (see page 18822)

RemotePort (see page 18823)

Address (see page 18750)

129.1.7.5.12 TCustomWinSocket.RemoteHost Property

Specifies the name of the remote system.

Pascal

```
property RemoteHost: string;
```

C++

```
__property AnsiString RemoteHost;
```

Description

Use RemoteHost to get the name of the system on the other end of the socket connection. RemoteHost is a string containing the domain name and service of the remote endpoint of the socket connection, such as

`http://www.wSite.Com`

LocalHost (see page 18820) provides an alias for the local IP address given in the RemoteAddress (see page 18822) property.

See Also

LocalHost (see page 18820)

RemoteAddress (see page 18822)

Address (see page 18750)

Host (see page 18751)

129.1.7.5.13 TCustomWinSocket.RemotePort Property

Specifies the ID number of the port used by the RemoteHost (see page 18822) to form the socket connection.

Pascal

```
property RemotePort: Integer;
```

C++

```
__property int RemotePort;
```

Description

Use RemotePort to determine the port number used by the socket on the other end of the socket connection.

Port numbers allow a single system, identified by the RemoteAddress (see page 18822) or RemoteHost (see page 18822) property, to host multiple connections simultaneously. Each combination of RemotePort and RemoteAddress (see page 18822) can only be bound to a single socket connection. This combination is represented by the RemoteAddr (see page 18821) property.

Many port numbers are associated by convention with a particular service such as ftp or http. To determine whether the value of RemotePort is associated with a specific service, use the LookupService (see page 18810) method.

Note: If a client socket requests a specific port to indicate a desired service, the value of RemotePort for the actual connection may differ from the requested port number. Most server sockets listen on the port associated with a service, but switch to an arbitrary available port number for forming connections.

See Also

LookupState (see page 18821)

RemoteAddr (see page 18821)

RemoteAddress (see page 18822)

RemoteHost (see page 18822)

Port (see page 18751)

129.1.7.5.14 TCustomWinSocket.SocketHandle Property

Specifies the Windows handle for the socket.

Pascal

```
property SocketHandle: TSocket;
```

C++

```
__property TSocket SocketHandle;
```

Description

Use SocketHandle for Windows socket API calls that require the handle to the socket. When the Connected (see page 18818) property is false, SocketHandle is INVALID_SOCKET.

SocketHandle is set to a valid Windows socket handle by the Listen (see page 18808) or Open (see page 18810) method.

See Also

Connected (see page 18818)

Create (see page 18801)

Listen (see page 18808)

Open (see page 18810)

129.1.7.6 TCustomWinSocket Events

The events of the TCustomWinSocket class are listed here.

Public Events

	Name	Description
	OnErrorEvent (see page 18824)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnSocketEvent (see page 18824)	Occurs when the Windows socket receives notification of asynchronous events.

129.1.7.6.1 TCustomWinSocket.OnErrorEvent Event

Occurs when the socket fails in making, using, or shutting down a connection.

Pascal

```
property OnErrorEvent: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnErrorEvent;
```

Description

Socket components provide an OnErrorEvent event handler that converts error notifications into error events on the socket component. Most applications will therefore not provide an event handler for OnErrorEvent.

See Also

ESocketError

InitSocket (see page 18746)

ASyncStyles (see page 18817)

129.1.7.6.2 TCustomWinSocket.OnSocketEvent Event

Occurs when the Windows socket receives notification of asynchronous events.

Pascal

```
property OnSocketEvent: TSocketEventEvent;
```

C++

```
__property TSocketEventEvent OnSocketEvent;
```

Description

Socket components provide an OnSocketEvent event handler that converts Windows socket notifications into events on the socket component. Most applications will therefore not provide an event handler for OnSocketEvent. Replacing this event handler can prevent some expected events from occurring.

The Sender parameter is the Windows socket object that owns the event handler. Usually, this is the same as the Socket parameter.

The Socket parameter indicates the Windows socket that generates or receives notification of the event.

The SocketEvent parameter indicates the event that occurred.

See Also

ASyncStyles (see page 18817)

InitSocket (see page 18746)

129.1.8 TServerAcceptThread Class

Class Hierarchy

Pascal

```
TServerAcceptThread = class(TThread);
```

C++

```
class TServerAcceptThread : public TThread;
```

File

ScktComp

Description

This is class ScktComp.TServerAcceptThread.

129.1.8.1 TServerAcceptThread Members

The following tables list the members exposed by TServerAcceptThread.

Protected Methods

	Name	Description
	CheckThreadError (see page 3114)	This is the overview for the CheckThreadError method overload.
V	DoTerminate (see page 3115)	Generates an OnTerminate (see page 3148) event.

Protected Properties

	Name	Description
	ReturnValue (see page 3141)	Specifies the value returned to other waiting threads when the thread finishes executing.
R	Terminated (see page 3142)	Indicates whether the thread has been asked to terminate.

Public Events

	Name	Description
	OnTerminate (see page 3148)	Occurs after the thread's Execute (see page 3115) method has returned and before the thread is destroyed.

Public Methods

	Name	Description
V	AfterConstruction (see page 3114)	Executes the thread when the thread is created with CreateSuspended set to false.
	Create (see page 18826)	This is Create, a member of class TServerAcceptThread.
V	Destroy (see page 3113)	Destroys the thread object and releases the memory allocated to it.
V	Execute (see page 18827)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.
	Queue (see page 3119)	This is the overview for the Queue method overload.
	RemoveQueuedEvents (see page 3120)	Remove queued method calls.
	Resume (see page 3120)	Restarts the execution of a suspended thread.
	StaticQueue (see page 3124)	Asynchronously executes a method call within the main thread.
	StaticSynchronize (see page 3125)	Executes a method call within the main thread.
	Suspend (see page 3125)	Pauses a running thread.
	Synchronize (see page 3126)	This is the overview for the Synchronize method overload.
	Terminate (see page 3126)	Signals the thread to terminate by setting the Terminated (see page 3142) property to true.
	WaitFor (see page 3132)	Waits for the thread to terminate and then returns the value of the ReturnValue (see page 3141) property.

Public Properties

	Name	Description
R	FatalException (see page 3133)	Determines whether the thread object is automatically destroyed when the thread terminates.
	FreeOnTerminate (see page 3134)	Determines whether the thread object is automatically destroyed when the thread terminates.
R	Handle (see page 3137)	Indicates the thread's handle (Windows only).
	Priority (see page 3138)	Determines the thread's scheduling priority relative to other threads in the process.
R	ServerSocket (see page 18831)	This is ServerSocket, a member of class TServerAcceptThread.
	Suspended (see page 3142)	Indicates whether a thread is suspended.
R	ThreadID (see page 3148)	Identifies the thread throughout the system.

129.1.8.2 TServerAcceptThread.Create Constructor

Pascal

```
constructor Create(CreateSuspended: Boolean; ASocket: TServerWinSocket);
```

C++

```
__fastcall TServerAcceptThread(Boolean CreateSuspended, TServerWinSocket ASocket);
```

Description

This is Create, a member of class TServerAcceptThread.

129.1.8.3 TServerAcceptThread Methods

The methods of the TServerAcceptThread class are listed here.

Public Methods

	Name	Description
V	Execute (see page 18827)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

129.1.8.3.1 TServerAcceptThread.Execute Method

Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Pascal

```
procedure Execute; override;
```

C++

```
virtual __fastcall Execute();
```

Description

Override Execute and insert the code that should be executed when the thread runs. Execute is responsible for checking the value of the Terminated property to determine if the thread needs to exit.

A thread executes when Create is called if CreateSuspended set to false, or when Resume is first called after the thread is created if CreateSuspended set to true.

Note: Do not use the properties and methods of other objects directly in the Execute method of a thread. Instead, separate the use of other objects into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize method.

C++ Examples:

```
/*
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes. To
create the extension choose the
File | New | Other | Thread Object menu.
*/
bool mythreadRunning;
bool youthreadRunning;

class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
}

class TYouThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};

__fastcall TYouThread::TYouThread(bool suspended)
```

```
    : TThread(suspended)
{
}

void __fastcall TMyThread::Execute()
{
  Form1->Memo1->Lines->Add("Process has been running for this many seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo1->Lines->Add("Low priority process " + IntToStr(I));
    Sleep(1000);
  }
  mythreadRunning = FALSE;
}

void __fastcall TYouThread::Execute()
{
  Form1->Memo3->Lines->Add("Second low priority process has been running for this many
seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo3->Lines->Add("Second low priority process " + IntToStr(I));
    Sleep(1000);
  }
  youthreadRunning = FALSE;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMyThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    SecondProcess = new TMyThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form1->Memo2->Lines->Add("Do some work in the main process for 10 seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo2->Lines->Add("Main process " + IntToStr(I));
    Sleep(1000);
  }
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  TYouThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    SecondProcess = new TYouThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
```

```
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  mythreadRunning = FALSE;
  youthreadRunning = FALSE;
}
```

Delphi Examples:

```
{
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes.  The
Execute procedure must be overridden in the extension class
of TThread or an "Abstract Error" will result. To create the
extension choose the File | New | Other | Thread Object menu.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (mythreadRunning = FALSE) then
  begin
    mythreadRunning:= TRUE;
    MyProcess := TMyThread.Create(True); { create suspended - secondprocess does not run yet }
    MyProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    MyProcess.Priority := tpLower;  // set the priority to lower than normal
    MyProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  if mythreadRunning then MyProcess.Terminate();
end;

procedure TMyThread.Execute;
var I: Integer;
begin
  Form1.Memo1.Lines.Add('Process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo1.Lines.Add('Low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
  mythreadRunning:= FALSE;
end;

procedure TYouThread.Execute;
var I: Integer;
begin
  Form1.Memo3.Lines.Add('Second low priority process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo3.Lines.Add('Second low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
    youthreadRunning:= FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
```

```
var   I: Integer;
begin
  Form1.Memo2.Lines.Add('Do some work in the main process for 10 seconds:');
  for I := 0 to 10 do
  begin
    Form1.Memo2.Lines.Add('Main process ' + InttoStr(I));
    Sleep(1000);
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  if (youthreadRunning = FALSE) then
  begin
    youthreadRunning:= TRUE;
    YouProcess := TYouThread.Create(True); { create suspended – secondprocess does not run yet
}
    YouProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    YouProcess.Priority := tpLower;  // set the priority to lower than normal
    YouProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  if youthreadRunning then YouProcess.Terminate();
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  mythreadRunning:= FALSE;
  youthreadRunning:= FALSE;
end;
```

See Also

Create (see page 3110)

Terminated (see page 3142)

Synchronize (see page 3126)

Checking for Termination by Other Threads

Waiting for a Thread to Finish Executing

Writing Clean-up Code

Writing the Thread Function

129.1.8.4 TServerAcceptThread Properties

The properties of the TServerAcceptThread class are listed here.

Public Properties

	Name	Description
R	ServerSocket (see page 18831)	This is ServerSocket, a member of class TServerAcceptThread.

129.1.8.4.1 TServerAcceptThread.ServerSocket Property

Pascal

```
property ServerSocket: TServerWinSocket;
```

C++

```
__property TServerWinSocket ServerSocket;
```

Description

This is ServerSocket, a member of class TServerAcceptThread.

129.1.9 TServerClientThread Class

TServerClientThread is an execution thread used for a single connection to a client socket accepted by a Windows server socket.

Class Hierarchy

Pascal

```
TServerClientThread = class(TThread);
```

C++

```
class TServerClientThread : public TThread;
```

File

ScktComp

Description

TServerWinSocket (see page 18858) objects spawn TServerClientThread objects to handle separate client connections in separate execution threads. This allows server sockets to perform slow reading and writing operations with client sockets without adversely affecting the performance on other connections.

TServerClientThread introduces new properties to

Access the TServerWinSocket (see page 18858) object that spawned the thread and the TServerClientWinSocket (see page 18840) that is connected to the client socket.

Store and access thread-specific data.

Determine if the thread is stored in a thread cache maintained by the TServerWinSocket (see page 18858).

Applications that receive frequent client requests will want to derive a descendant class from TServerClientThread to handle reading and writing to separate clients. This is because TServerClientThread objects use the OnClientRead and OnClientWrite events of the associated server socket. These events are not thread-local.

See Also

TServerClientWinSocket (see page 18840)

TServerWinSocket (see page 18858)

129.1.9.1 TServerClientThread Members

The following tables list the members exposed by TServerClientThread.

Protected Methods

	Name	Description
	CheckThreadError (see page 3114)	This is the overview for the CheckThreadError method overload.
V	ClientExecute (see page 18834)	Simulates OnClientRead and OnClientWrite events for the associated server socket.
V	DoTerminate (see page 18835)	Removes the thread from the list of active threads maintained by the associated server socket.
	EndConnect (see page 18835)	Indicates whether the thread should execute again for another client.
V	Error (see page 18836)	Generates an OnClientError event for the associated server socket.
V	Event (see page 18836)	This is Event, a member of class TServerClientThread.
V	Execute (see page 18836)	Manages the caching of server client threads and calls the ClientExecute (see page 18834) method to run the thread for individual client connections.
V	HandleException (see page 18837)	Displays an error message using the main VCL thread.
	ReActivate (see page 18837)	This is ReActivate, a member of class TServerClientThread.
	StartConnect (see page 18837)	Indicates when the thread is ready to execute.

Protected Properties

	Name	Description
	ReturnValue (see page 3141)	Specifies the value returned to other waiting threads when the thread finishes executing.
R	Terminated (see page 3142)	Indicates whether the thread has been asked to terminate.

Public Events

	Name	Description
	OnTerminate (see page 3148)	Occurs after the thread's Execute (see page 3115) method has returned and before the thread is destroyed.

Public Methods

	Name	Description
V	AfterConstruction (see page 3114)	Executes the thread when the thread is created with CreateSuspended set to false.
	Create (see page 18833)	Creates an instance of TServerClientThread (see page 18831).
V	Destroy (see page 18833)	Destroys the server client thread object and releases the memory allocated to it.
	Queue (see page 3119)	This is the overview for the Queue method overload.
	RemoveQueuedEvents (see page 3120)	Remove queued method calls.
	Resume (see page 3120)	Restarts the execution of a suspended thread.
	StaticQueue (see page 3124)	Asynchronously executes a method call within the main thread.
	StaticSynchronize (see page 3125)	Executes a method call within the main thread.
	Suspend (see page 3125)	Pauses a running thread.
	Synchronize (see page 3126)	This is the overview for the Synchronize method overload.
	Terminate (see page 3126)	Signals the thread to terminate by setting the Terminated (see page 3142) property to true.
	WaitFor (see page 3132)	Waits for the thread to terminate and then returns the value of the ReturnValue (see page 3141) property.

Public Properties

	Name	Description
R	ClientSocket (see page 18838)	Specifies the TServerClientWinSocket (see page 18840) object that uses the thread for all reading and writing over the socket connection.
	Data (see page 18838)	Points to application-specific data associated with the thread.
R	FatalException (see page 3133)	Determines whether the thread object is automatically destroyed when the thread terminates.

	FreeOnTerminate (see page 3134)	Determines whether the thread object is automatically destroyed when the thread terminates.
R	Handle (see page 3137)	Indicates the thread's handle (Windows only).
	KeepInCache (see page 18839)	Determines whether the thread should be kept in the thread cache maintained by ServerSocket (see page 18839).
	Priority (see page 3138)	Determines the thread's scheduling priority relative to other threads in the process.
R	ServerSocket (see page 18839)	Specifies the TServerWinSocket (see page 18858) object that manages all the TServerClientThread (see page 18831) objects for its client connections.
	Suspended (see page 3142)	Indicates whether a thread is suspended.
R	ThreadID (see page 3148)	Identifies the thread throughout the system.

129.1.9.2 TServerClientThread.Create Constructor

Creates an instance of TServerClientThread (see page 18831).

Pascal

```
constructor Create(CreateSuspended: Boolean; ASocket: TServerClientWinSocket);
```

C++

```
__fastcall TServerClientThread(Boolean CreateSuspended, TServerClientWinSocket ASocket);
```

Description

Use Create to create an instance of TServerClientThread (see page 18831) for a connection to a client socket, and add the thread to the cache maintained by the associated listening socket.

Initializes FreeOnTerminate to true.

Initializes Priority to tpHigher.

Initializes ClientSocket (see page 18838) to the ASocket parameter and ServerSocket (see page 18839) to ASocket.ServerWinSocket.

Starts the thread executing.

Adds the thread to the cache maintained by ServerSocket (see page 18839).

Sets the OnErrorEvent and OnSocketEvent event handlers of ClientSocket (see page 18838) so that events are sent to ServerSocket (see page 18839).

Signals when initialization is complete and the thread is ready for reading from or writing to the client connection.

See Also

ClientSocket (see page 18838)

FreeOnTerminate (see page 3134)

Priority (see page 3138)

ServerSocket (see page 18839)

OnErrorEvent (see page 18824)

OnSocketEvent (see page 18824)

129.1.9.3 TServerClientThread.Destroy Destructor

Destroys the server client thread object and releases the memory allocated to it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TServerClientThread();
```

Description

Do not call Destroy directly in an application. ServerClientThread objects are freed by the TServerWinSocket (see page 18858) object that manages them.

Destroy frees the TServerClientWinSocket (see page 18840) object that encapsulates the connection to the client socket before calling the inherited destructor.

See Also

ClientSocket (see page 18838)

ServerSocket (see page 18839)

129.1.9.4 TServerClientThread Methods

The methods of the TServerClientThread class are listed here.

Protected Methods

	Name	Description
	ClientExecute (see page 18834)	Simulates OnClientRead and OnClientWrite events for the associated server socket.
	DoTerminate (see page 18835)	Removes the thread from the list of active threads maintained by the associated server socket.
	EndConnect (see page 18835)	Indicates whether the thread should execute again for another client.
	Error (see page 18836)	Generates an OnClientError event for the associated server socket.
	Event (see page 18836)	This is Event, a member of class TServerClientThread.
	Execute (see page 18836)	Manages the caching of server client threads and calls the ClientExecute (see page 18834) method to run the thread for individual client connections.
	HandleException (see page 18837)	Displays an error message using the main VCL thread.
	ReActivate (see page 18837)	This is ReActivate, a member of class TServerClientThread.
	StartConnect (see page 18837)	Indicates when the thread is ready to execute.

129.1.9.4.1 TServerClientThread.ClientExecute Method

Simulates OnClientRead and OnClientWrite events for the associated server socket.

Pascal

```
procedure ClientExecute; virtual;
```

C++

```
virtual __fastcall ClientExecute();
```

Description

TServerClientThread (see page 18831) calls ClientExecute from its Execute (see page 18836) method. ClientExecute checks whether there is any information to read from the associated socket connection, and if so, simulates an OnClientRead event on the associated server socket. If there is no information sent from the client, ClientExecute simulates an OnClientWrite event.

Because the server socket that receives these events is not local to the thread, the events occur within the main VCL thread using the Synchronize method. This method can be too slow if the server socket has many client connections, because each

thread must wait for any other thread to finish reading or writing. When a server socket receives many client connection requests, override ClientExecute to handle the reading or writing using a thread-local instance of TWinSocketStream (see page 18876). In the ClientExecute method, be sure to check the Terminated property periodically.

Note: Do not use the properties and methods of global objects directly in the ClientExecute method. Instead, separate the use of objects that are not thread-local into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize method.

See Also

Execute (see page 18836)

Synchronize (see page 3126)

TWinSocketStream (see page 18876)

Terminated (see page 3142)

DoTerminate (see page 18835)

129.1.9.4.2 TServerClientThread.DoTerminate Method

Removes the thread from the list of active threads maintained by the associated server socket.

Pascal

```
procedure DoTerminate; override;
```

C++

```
virtual __fastcall DoTerminate();
```

Description

DoTerminate overrides the inherited method to suppress the OnTerminate event. Instead, it removes the thread from the Connections property of the associated TServerWinSocket (see page 18858). The thread is then either added to the server socket's thread cache or destroyed.

See Also

ClientExecute (see page 18834)

Terminated (see page 3142)

Connections (see page 18868)

129.1.9.4.3 TServerClientThread.EndConnect Method

Indicates whether the thread should execute again for another client.

Pascal

```
function EndConnect: Boolean;
```

C++

```
__fastcall Boolean EndConnect();
```

Description

The Execute (see page 18836) method calls EndConnect when the ClientExecute (see page 18834) method finishes. It frees all memory associated with the current client connection and checks whether the Thread is being reused for another client connection. If the thread should be reused, EndConnect returns true to indicate that the ClientExecute (see page 18834) method should be called again.

See Also

ClientExecute (see page 18834)

Execute (see page 18836)

129.1.9.4.4 TServerClientThread.Error Method

Generates an OnClientError event for the associated server socket.

Pascal

```
procedure Error(ErrorEvent: TErrorEvent; var ErrorCode: Integer); virtual;
```

C++

```
virtual __fastcall Error(TErrorEvent ErrorEvent, int ErrorCode);
```

Description

TServerClientThread (see page 18831) assigns the Error method as an event handler for the OnErrorEvent method of the associated TClientWinSocket (see page 18766) object. This allows errors received from the client to generate OnClientError events on the server socket.

Warning: Error is not thread-safe. It should not be called from the server client thread. It is only intended for use by the associated client socket, which is not local to the server client thread.

See Also

OnClientError (see page 18785)

OnErrorEvent (see page 18824)

129.1.9.4.5 TServerClientThread.Event Method

Pascal

```
procedure Event(SocketEvent: TSocketEvent); virtual;
```

C++

```
virtual __fastcall Event(TSocketEvent SocketEvent);
```

Description

This is Event, a member of class TServerClientThread.

129.1.9.4.6 TServerClientThread.Execute Method

Manages the caching of server client threads and calls the ClientExecute (see page 18834) method to run the thread for individual client connections.

Pascal

```
procedure Execute; override;
```

C++

```
virtual __fastcall Execute();
```

Description

Execute is called automatically when the thread runs. A thread runs when Create (see page 18833) is called if CreateSuspended set to false, or when Resume is first called after the thread is created if CreateSuspended set to true.

Do not override the Execute method to implement the thread function of a TServerClientThread (see page 18831) descendant. Instead, override the ClientExecute (see page 18834) method, which is run after the thread is initialized for a particular client connection.

See Also

ClientExecute (see page 18834)

Reactivate (see page 18837)

129.1.9.4.7 TServerClientThread.HandleException Method

Displays an error message using the main VCL thread.

Pascal

```
procedure HandleException; virtual;
```

C++

```
virtual __fastcall HandleException();
```

Description

HandleException obtains the current exception object and calls TApplication.ShowException (for descendants of Exception) or the global ShowException procedure (for other objects) to display an error message. HandleException wraps the call to ShowException in the Synchronize method to ensure that the call is thread-safe.

See Also

ShowException (see page 13698)

ShowException (see page 22941)

Synchronize (see page 3126)

129.1.9.4.8 TServerClientThread.ReActivate Method

Pascal

```
procedure ReActivate(ASocket: TServerClientWinSocket);
```

C++

```
__fastcall ReActivate(TServerClientWinSocket ASocket);
```

Description

This is ReActivate, a member of class TServerClientThread.

129.1.9.4.9 TServerClientThread.StartConnect Method

Indicates when the thread is ready to execute.

Pascal

```
function StartConnect: Boolean;
```

C++

```
__fastcall Boolean StartConnect();
```

Description

The Execute (see page 18836) method calls StartConnect to determine when to call ClientExecute (see page 18834).

StartConnect waits until the Reactivate method has signaled that the thread is properly initialized, and then returns true if the thread is ready to run. If the thread is terminated for any reason before it is fully initialized, StartConnect returns false.

See Also

Reactivate (see page 18837)

ClientExecute (see page 18834)

Execute (see page 18836)

129.1.9.5 TServerClientThread Properties

The properties of the TServerClientThread class are listed here.

Public Properties

	Name	Description
R	ClientSocket (see page 18838)	Specifies the TServerClientWinSocket (see page 18840) object that uses the thread for all reading and writing over the socket connection.
	Data (see page 18838)	Points to application-specific data associated with the thread.
	KeepInCache (see page 18839)	Determines whether the thread should be kept in the thread cache maintained by ServerSocket (see page 18839).
R	ServerSocket (see page 18839)	Specifies the TServerWinSocket (see page 18858) object that manages all the TServerClientThread (see page 18831) objects for its client connections.

129.1.9.5.1 TServerClientThread.ClientSocket Property

Specifies the TServerClientWinSocket (see page 18840) object that uses the thread for all reading and writing over the socket connection.

Pascal

```
property ClientSocket: TServerClientWinSocket;
```

C++

```
__property TServerClientWinSocket ClientSocket;
```

Description

Use ClientSocket to gain direct access to the Windows socket component that uses the thread. If ClientSocket is nil (Delphi) or NULL (C++), the thread is idle and available for reuse by another TServerClientWinSocket (see page 18840) object. TServerClientThread (see page 18831) uses ClientSocket to read from or write to the socket connection.

See Also

TServerClientWinSocket (see page 18840)

129.1.9.5.2 TServerClientThread.Data Property

Points to application-specific data associated with the thread.

Pascal

```
property Data: Pointer;
```

C++

```
__property void * Data;
```

Description

Use Data to associate information with the thread object. Data can store information in a thread-safe manner that would otherwise require the use of global variables.

See Also

Data (see page 18818)

129.1.9.5.3 TServerClientThread.KeepInCache Property

Determines whether the thread should be kept in the thread cache maintained by ServerSocket (see page 18839).

Pascal

```
property KeepInCache: Boolean;
```

C++

```
__property Boolean KeepInCache;
```

Description

The TServerWinSocket (see page 18858) that spawns server client threads for its connections to client sockets maintains a cache of server client threads for reuse. As server client threads are activated, the TServerWinSocket (see page 18858) object sets the KeepInCache property to indicate whether the thread will fit in the thread cache.

When KeepInCache is true, FreeOnTerminate is set to false before the thread terminates, and the thread is left in the thread cache. When KeepInCache is false, the thread is freed when execution terminates.

See Also

FreeOnTerminate (see page 3134)

ServerSocket (see page 18839)

ThreadCacheSize (see page 18870)

129.1.9.5.4 TServerClientThread.ServerSocket Property

Specifies the TServerWinSocket (see page 18858) object that manages all the TServerClientThread (see page 18831) objects for its client connections.

Pascal

```
property ServerSocket: TServerWinSocket;
```

C++

```
__property TServerWinSocket ServerSocket;
```

Description

Use ServerSocket to get direct access to the listening Windows socket object that spawned the thread for a client connection it accepted. TServerClientThread (see page 18831) passes events and errors to the appropriate event handlers assigned to ServerSocket. TServerClientThread (see page 18831) adds itself to the cache of threads maintained by ServerSocket from its constructor, and removes itself when the thread terminates.

See Also

Create (see page 18833)

TServerWinSocket (see page 18858)

129.1.10 TServerClientWinSocket Class

TServerClientWinSocket describes the server endpoint of a socket connection to a client socket.

Class Hierarchy

Pascal

```
TServerClientWinSocket = class(TCustomWinSocket);
```

C++

```
class TServerClientWinSocket : public TCustomWinSocket;
```

File

ScktComp

Description

TServerClientWinSocket is used by server socket components to manage the Windows socket API calls for the socket connection to an individual client socket. TServerClientWinSocket objects are created when a client socket connection is accepted by a listening server socket.

Applications can create and customize the properties of a TServerClientWinSocket object in an OnGetSocket event handler for a server socket component. Component writers can create descendants of TServerClientWinSocket that may be substituted for TServerClientWinSocket in the OnGetSocket event handler.

See Also

TClientWinSocket (see page 18766)

TCustomServerSocket (see page 18770)

TServerSocket (see page 18843)

TServerWinSocket (see page 18858)

129.1.10.1 TServerClientWinSocket Members

The following tables list the members exposed by TServerClientWinSocket.

Protected Methods

	Name	Description
	AsyncInitSocket (see page 18803)	Looks up information or forms socket connections asynchronously.
	DoListen (see page 18806)	Opens a listening connection.
	DoOpen (see page 18806)	Opens a connection to a remote socket.
	Error (see page 18806)	Generates an OnErrorEvent (see page 18824) event.
	Event (see page 18807)	This is Event, a member of class TCustomWinSocket.
	InitSocket (see page 18807)	Generates a windows Internet socket address structure from a description of the desired socket port.
	SetAsyncStyles (see page 18815)	Sets the AsyncStyles property.

Public Events

	Name	Description
	OnErrorEvent (see page 18824)	Occurs when the socket fails in making, using, or shutting down a connection.

	OnSocketEvent (see page 18824)	Occurs when the Windows socket receives notification of asynchronous events.

Public Methods

	Name	Description
V	Accept (see page 18803)	Provides the interface for a method that accepts a connection to a client socket.
	Close (see page 18804)	Shuts down the socket connection if it is open.
V	Connect (see page 18805)	Provides the interface for a method that responds to connection notifications from client sockets.
	Create (see page 18842)	Creates and initializes an instance of TServerClientWinSocket (see page 18840).
V	DefaultHandler (see page 18805)	Processes message records.
V	Destroy (see page 18842)	Destroys an instance of TServerClientWinSocket (see page 18840).
V	Disconnect (see page 18805)	Shuts down the socket connection represented by a Windows socket handle.
	Listen (see page 18808)	Opens a listening connection for a server socket.
	Lock (see page 18809)	Starts a critical section, blocking all other execution threads until the Unlock (see page 18815) method is called.
	LookupName (see page 18809)	Generates the portion of a Windows Internet socket address structure that describes the host.
	LookupService (see page 18810)	Returns the port number associated with a specific service.
	Open (see page 18810)	Opens a connection to a remote socket.
V	Read (see page 18811)	Generates an OnSocketEvent (see page 18824) of type seRead in response to notifications that the socket connection is ready.
	ReceiveBuf (see page 18812)	This is ReceiveBuf, a member of class TCustomWinSocket.
	ReceiveLength (see page 18812)	Returns the number of bytes ready to be sent over the socket connection.
	ReceiveText (see page 18812)	Reads a string from the socket connection.
	SendBuf (see page 18813)	Writes Count bytes to the socket connection from the Buf parameter.
	SendStream (see page 18814)	This is SendStream, a member of class TCustomWinSocket.
	SendStreamThenDrop (see page 18814)	Writes all the information that can be read from the AStream parameter to the socket connection and then terminates the connection.
	SendText (see page 18814)	Writes the string S to the socket connection.
	Unlock (see page 18815)	Reenables execution in other threads after a critical section begun by the Lock (see page 18809) method.
V	Write (see page 18816)	Generates an OnSocketEvent (see page 18824) of type seWrite in response to notifications that the socket connection is ready.

Public Properties

	Name	Description
R	Addr (see page 18817)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the socket port.
	ASyncStyles (see page 18817)	Determines which asynchronous events the socket can receive.
R	Connected (see page 18818)	Indicates whether the socket connection is open and available for communication with other machines.
	Data (see page 18818)	Points to application-specific data associated with the socket.
R	Handle (see page 18819)	Specifies the handle of the window that receives notifications from the Windows socket.
R	LocalAddress (see page 18819)	Specifies the IP address of the local system that was bound to the socket connection.
R	LocalHost (see page 18820)	Specifies the name of the local system.
R	LocalPort (see page 18820)	Specifies the ID number of the port used by the socket connection.
R	LookupState (see page 18821)	Indicates any asynchronous activities occurring while opening a connection.
R	RemoteAddr (see page 18821)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the remote socket port.
R	RemoteAddress (see page 18822)	Specifies the IP address of the remote system that was bound to the socket connection.
R	RemoteHost (see page 18822)	Specifies the name of the remote system.
R	RemotePort (see page 18823)	Specifies the ID number of the port used by the RemoteHost (see page 18822) to form the socket connection.
R	ServerWinSocket (see page 18843)	Describes the endpoint of the listening connection that accepted the connection to the client socket which is managed by this TServerClientWinSocket (see page 18840).

R	SocketHandle (see page 18823)	Specifies the Windows handle for the socket.

129.1.10.2 TServerClientWinSocket.Create Constructor

Creates and initializes an instance of TServerClientWinSocket (see page 18840).

Pascal

```
constructor Create(Socket: TSocket; ServerWinSocket: TServerWinSocket);
```

C++

```
__fastcall TServerClientWinSocket(TSocket Socket, TServerWinSocket ServerWinSocket);
```

Description

Use Create to create an instance of TServerClientWinSocket (see page 18840) in the OnGetSocket event handler of a server socket component. The Socket parameter is the Windows socket handle for the accepted socket connection to a client socket. The ServerWinSocket (see page 18843) parameter is the Windows socket object for the listening connection that accepted the connection to the client socket.

Create adds the new TServerClientWinSocket (see page 18840) object to the list of connections maintained by the listening server socket. It then sets the OnSocketEvent (see page 18824) event handler so that notifications from the connection to the client can be passed on to the server socket. After setting up the connection to the server socket, Create calls the inherited constructor.

See Also

OnSocketEvent (see page 18824)

Create (see page 18801)

Connections (see page 18868)

129.1.10.3 TServerClientWinSocket.Destroy Destructor

Destroys an instance of TServerClientWinSocket (see page 18840).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TServerClientWinSocket();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TServerClientWinSocket (see page 18840) reference is not nil, and only then calls Destroy. Instances of TServerClientWinSocket (see page 18840) are freed automatically when the connection is closed.

Destroy removes the TServerClientWinSocket (see page 18840) from the list of connections maintained by the listening windows socket.

See Also

Free (see page 22300)

Connections (see page 18868)

129.1.10.4 TServerClientWinSocket Properties

The properties of the TServerClientWinSocket class are listed here.

Public Properties

	Name	Description
R	ServerWinSocket (see page 18843)	Describes the endpoint of the listening connection that accepted the connection to the client socket which is managed by this TServerClientWinSocket (see page 18840).

129.1.10.4.1 TServerClientWinSocket.ServerWinSocket Property

Describes the endpoint of the listening connection that accepted the connection to the client socket which is managed by this TServerClientWinSocket (see page 18840).

Pascal

```
property ServerWinSocket: TServerWinSocket;
```

C++

```
__property TServerWinSocket ServerWinSocket;
```

Description

Use ServerWinSocket to gain access to the Windows socket object for the listening connection that spawned this TServerClientWinSocket (see page 18840). Use the ServerWinSocket to trigger events that can be handled by the event handlers of the server socket, to get execution threads from the cache maintained by the Windows server socket, or to get information about the other active connections accepted by the server socket.

See Also

TServerWinSocket (see page 18858)

129.1.11 TServerSocket Class

TServerSocket manages server socket connections for a TCP/IP server.

Class Hierarchy

Pascal

```
TServerSocket = class(TCustomServerSocket);
```

C++

```
class TServerSocket : public TCustomServerSocket;
```

File

ScktComp

Description

Add a TServerSocket object to a form or data module to turn an application into a TCP/IP server. TServerSocket listens for

requests for TCP/IP connections from other machines, and establishes connections when requests are received.

See Also

TClientSocket (see page 18752)

TServerClientWinSocket (see page 18840)

TServerWinSocket (see page 18858)

129.1.11.1 TServerSocket Members

The following tables list the members exposed by TServerSocket.

Protected Events

	Name	Description
	OnConnect (see page 18795)	Occurs on client sockets just after the connection to the server is opened.
	OnConnecting (see page 18796)	Occurs for a client socket after the server socket has been located, but before the connection is established.
	OnDisconnect (see page 18796)	Occurs just before a client socket closes the connection to a server socket.
	OnError (see page 18797)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnLookup (see page 18797)	Occurs when a client socket is about to look up the server socket with which it wants to connect.
	OnRead (see page 18798)	Occurs when a client socket should read information from the socket connection.
	OnWrite (see page 18799)	Occurs when a client socket should write information to the socket connection.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FServerSocket (see page 18775)	This is FServerSocket, a member of class TCustomServerSocket.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
A	DoActivate (see page 18744)	Provides the prototype of a method to open or close the socket connection.
V	DoActivate (see page 18775)	Opens or Closes the socket connection.
A	Error (see page 18745)	Provides the prototype of a method that responds to error conditions.
V	Error (see page 18793)	Generates an OnError (see page 18797) event.
A	Event (see page 18746)	Provides the prototype of a method that responds when specific socket events occur.
V	Event (see page 18794)	Dispatches the events that occur in the normal operation of the socket to the appropriate event handlers.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetGetSocketEvent (see page 18776)	Returns the OnGetSocket (see page 18787) event handler.
	GetGetThreadEvent (see page 18776)	Returns the OnGetThread (see page 18787) event handler.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).

	GetOnClientError (see page 18776)	Returns the OnClientError (see page 18785) event handler.
	GetOnClientEvent (see page 18777)	Provides indexed access to the various event handlers that respond to client events.
	GetOnThreadEnd (see page 18777)	Returns the OnThreadEnd (see page 18788) event handler.
	GetOnThreadStart (see page 18778)	Returns the OnThreadStart (see page 18788) event handler.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetServerType (see page 18778)	Returns the ServerType (see page 18782) property.
	GetThreadCacheSize (see page 18778)	Returns the value of the ThreadCacheSize (see page 18783) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InitSocket (see page 18746)	Tells a Windows socket object to call the Error (see page 18745) and Event (see page 18746) methods.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 18747)	Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 18750) property.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActive (see page 18748)	Sets the Active (see page 18750) property.
	SetAddress (see page 18748)	Sets the Address (see page 18750) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetGetSocketEvent (see page 18779)	Sets the OnGetSocket (see page 18787) event handler.
	SetGetThreadEvent (see page 18779)	Sets the OnGetThread (see page 18787) event handler.
	SetHost (see page 18748)	Sets the Host (see page 18751) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnClientError (see page 18779)	Sets the OnClientError (see page 18785) event handler.
	SetOnClientEvent (see page 18780)	Sets the client event handlers of the associated TServerWinSocket (see page 18858) object.
	SetOnThreadEnd (see page 18780)	Sets the OnThreadEnd (see page 18788) event handler.
	SetOnThreadStart (see page 18781)	Sets the OnThreadStart (see page 18788) event handler.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetPort (see page 18749)	Sets the Port (see page 18751) property.
	SetServerType (see page 18781)	Sets the ServerType (see page 18782) property.
	SetService (see page 18749)	Sets the Service (see page 18752) property.
	SetThreadCacheSize (see page 18781)	Sets the ThreadCacheSize (see page 18783) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Address (see page 18750)	Specifies the IP address of the server system.
	Host (see page 18751)	Specifies an alias for the IP address of the server system.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 18744)	Shuts down the socket connection.
V	Create (see page 18848)	Creates an instance of TServerSocket (see page 18843).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 18774)	Destroys an instance of TCustomServerSocket (see page 18770).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 18747)	Opens the socket connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Socket (see page 18851)	Specifies the TServerWinSocket (see page 18858) object that describes the endpoint of the listening connection.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnAccept (see page 18852)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnClientConnect (see page 18852)	Occurs when a client socket completes a connection accepted by the server socket.
	OnClientDisconnect (see page 18853)	Occurs when one of the connections to a client socket is closed.
	OnClientError (see page 18854)	Occurs when there is a failure in establishing, using, or terminating the socket connection to an individual client socket.
	OnClientRead (see page 18855)	Occurs when the server socket should read information from a client socket.
	OnClientWrite (see page 18855)	Occurs when the server socket should write information to a client socket.
	OnGetSocket (see page 18856)	Occurs when the server socket needs to create a new TServerClientWinSocket object to represent the server endpoint of a connection to a client socket.
	OnGetThread (see page 18856)	Occurs when the server socket needs to create a new execution thread for a connection to a client socket.
	OnListen (see page 18857)	Occurs just before a server socket is opened for listening.
	OnThreadEnd (see page 18857)	Occurs when a client socket connection is terminated and the associated thread finishes execution.
	OnThreadStart (see page 18858)	Occurs when the execution thread for a connection to a client socket starts up.

Published Properties

	Name	Description
	Active (see page 18849)	Indicates whether the socket connection is open and available for communication with other machines.

	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Port (see page 18849)	Specifies the ID number that identifies the server socket connection.
	ServerType (see page 18850)	Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.
	Service (see page 18850)	Specifies the name of the service for which the socket connection is used.
	Tag (see page 2808)	Stores an integer value as part of a component.
	ThreadCacheSize (see page 18851)	Specifies the maximum number of threads that can be reused for new client connections.

129.1.11.2 TServerSocket.Create Constructor

Creates an instance of TServerSocket (see page 18843).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TServerSocket(TComponent * AOwner);
```

Description

Call Create to create and initialize a server socket component at runtime. TServerSocket (see page 18843) objects placed at design time are created automatically.

After calling the inherited constructor, Create

Allocates the TServerWinSocket (see page 18858) object that encapsulates the Windows socket connection for the server socket.

Initializes the event and error dispatchers for the server socket.

Initializes ThreadCacheSize (see page 18851) to 10.

See Also

Create (see page 2766)

ThreadCacheSize (see page 18783)

129.1.11.3 TServerSocket Properties

The properties of the TServerSocket class are listed here.

Public Properties

	Name	Description
R	Socket (see page 18851)	Specifies the TServerWinSocket (see page 18858) object that describes the endpoint of the listening connection.

Published Properties

	Name	Description
	Active (see page 18849)	Indicates whether the socket connection is open and available for communication with other machines.
	Port (see page 18849)	Specifies the ID number that identifies the server socket connection.
	ServerType (see page 18850)	Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.
	Service (see page 18850)	Specifies the name of the service for which the socket connection is used.
	ThreadCacheSize (see page 18851)	Specifies the maximum number of threads that can be reused for new client connections.

129.1.11.3.1 TServerSocket.Active Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before attempting to use or change the socket connection, read Active to determine whether the connection is open and ready. For client sockets, setting Active opens or shuts down a socket connection to another machine. For server sockets, setting Active opens or shuts down a listening connection that makes the socket available for client requests.

At design time, set Active to true to make the socket open a connection when the application starts running. At runtime, use the Open or Close method to open or close the connection.

See Also

Close (see page 18744)

Open (see page 18747)

SetActive (see page 18748)

129.1.11.3.2 TServerSocket.Port Property

Specifies the ID number that identifies the server socket connection.

Pascal

```
property Port: Integer;
```

C++

```
__property int Port;
```

Description

Port numbers allow a single system, identified by the Host or Address property, to host multiple connections simultaneously. Many values of Port are associated by convention with a particular service such as ftp or http.

For server sockets, Port is the ID of the connection on which the server socket listens for client requests. Server sockets generally set Port to a predefined value which clients can use to initiate connections.

For client sockets, Port is the ID of the desired server connection. The value of Port is usually associated with the service the client wishes to use on the server application.

Note: Trying to change Port when the connection is open will raise an ESocketError exception.

See Also

Address (see page 18750)

ESocketError

Host (see page 18751)

Service (see page 18752)

SetPort (see page 18749)

129.1.11.3.3 TServerSocket.ServerType Property

Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.

Pascal

```
property ServerType: TServerType;
```

C++

```
__property TServerType ServerType;
```

Description

Set ServerType to stThreadBlocking to automatically spawn a new thread for each socket connection accepted by the server socket. When ServerType is stThreadBlocking, the execution of the connection thread is suspended while reading or writing until all information has been transferred over the connection. The thread for each connection generates OnClientRead or OnClientWrite events when the server socket needs to read or write.

Set ServerType to stNonBlocking to handle all reading and writing over the socket connections asynchronously. When ServerType is stNonBlocking, all client connections are handled in a single execution thread by default. OnClientRead or OnClientWrite events occur when the client socket on the other end of one of the connections tries to send or receive information over the connection.

Use a non-blocking socket when the socket needs to synchronize reading and writing with client sockets.

Note: When ServerType is stThreadBlocking, it is important that the OnClient... event handlers contain thread-safe code.

See Also

OnClientRead (see page 18786)

OnClientWrite (see page 18786)

TServerClientThread (see page 18831)

129.1.11.3.4 TServerSocket.Service Property

Specifies the name of the service for which the socket connection is used.

Pascal

```
property Service: string;
```

C++

```
__property AnsiString Service;
```

Description

Use Service to identify the use of the connection. Windows provides There are a number of standard service names such as ftp, http, finger, and time. Servers can specify additional services and their associated ports in a SERVICES file. For more information, see the Microsoft documentation for Windows sockets.

Certain port numbers are reserved for specific values of service. Thus, Service provides a more meaningful way to specify the server Port to use for the socket connection. For server sockets, using Service rather than Port ensures that the server will listen for TCP/IP requests on the appropriate port.

Note: Trying to change Service when the connection is open will raise an ESocketError exception.

See Also

ESocketError

Port (see page 18751)

SetService (see page 18749)

LookupService (see page 18810)

129.1.11.3.5 TServerSocket.Socket Property

Specifies the TServerWinSocket (see page 18858) object that describes the endpoint of the listening connection.

Pascal

```
property Socket: TServerWinSocket;
```

C++

```
__property TServerWinSocket Socket;
```

Description

Use Socket to obtain

Information about the connections currently active on the server socket.

Information about the connection threads that are cached for use by the server socket.

Access to the Windows socket handle for the listening connection that is needed for Windows socket API calls.

See Also

ThreadCacheSize (see page 18783)

129.1.11.3.6 TServerSocket.ThreadCacheSize Property

Specifies the maximum number of threads that can be reused for new client connections.

Pascal

```
property ThreadCacheSize: Integer;
```

C++

```
__property int ThreadCacheSize;
```

Description

When ServerType is stThreadBlocking, each new connection that is accepted by the server socket is given a separate execution thread. In order to improve performance, server sockets store these threads in a cache rather than freeing them when the connection is closed. New connections can then reuse threads from the cache, rather than requiring the socket server to create a new thread every time a connection is accepted.

Set ThreadCacheSize to specify the number of threads that are cached for reuse. The ideal value for ThreadCacheSize depends on the number and frequency of client socket requests received by the server socket. If ThreadCacheSize is too low, the server socket will spend more time freeing and creating threads when client connections are accepted. If ThreadCacheSize is too high, the server socket may unnecessarily lock up the memory for threads that are never reused.

See Also

ServerType (see page 18782)

TServerClientThread (see page 18831)

ActiveThreads (see page 18867)

IdleThreads (see page 18869)

129.1.11.4 TServerSocket Events

The events of the TServerSocket class are listed here.

Published Events

	Name	Description
	OnAccept (see page 18852)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnClientConnect (see page 18852)	Occurs when a client socket completes a connection accepted by the server socket.
	OnClientDisconnect (see page 18853)	Occurs when one of the connections to a client socket is closed.
	OnClientError (see page 18854)	Occurs when there is a failure in establishing, using, or terminating the socket connection to an individual client socket.
	OnClientRead (see page 18855)	Occurs when the server socket should read information from a client socket.
	OnClientWrite (see page 18855)	Occurs when the server socket should write information to a client socket.
	OnGetSocket (see page 18856)	Occurs when the server socket needs to create a new TServerClientWinSocket object to represent the server endpoint of a connection to a client socket.
	OnGetThread (see page 18856)	Occurs when the server socket needs to create a new execution thread for a connection to a client socket.
	OnListen (see page 18857)	Occurs just before a server socket is opened for listening.
	OnThreadEnd (see page 18857)	Occurs when a client socket connection is terminated and the associated thread finishes execution.
	OnThreadStart (see page 18858)	Occurs when the execution thread for a connection to a client socket starts up.

129.1.11.4.1 TServerSocket.OnAccept Event

Occurs on server sockets just after the connection to a client socket is accepted.

Pascal

```
property OnAccept: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnAccept;
```

Description

Write an OnAccept event handler for a server socket to take specific action after the connection to a client socket has been accepted. This is the first point when information is available about the local port and IP address that will be used in the server endpoint of the connection. The port number, and possibly the IP address, will differ from those of the listening socket.

Server sockets open a socket connection for listening to establish a queue to hold client requests. After a request from the queue is accepted by the server socket, an OnAccept event occurs. After the server socket accepts the connection, the client socket completes the connection, and an OnConnect event occurs for the client socket.

See Also

OnConnect (see page 18795)

OnListen (see page 18797)

129.1.11.4.2 TServerSocket.OnClientConnect Event

Occurs when a client socket completes a connection accepted by the server socket.

Pascal

```
property OnClientConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientConnect;
```

Description

Write an OnClientConnect event handler to take specific action when a client socket completes the socket connection to the server socket. For example, the socket may start reading or writing over the connection in an OnClientConnect event handler.

The order of server socket events leading up to OnClientConnect is as follows:

1An OnListen event occurs just before the server socket is opened for listening.

2The server socket receives client requests in a listening queue. The server socket accepts one of those requests, and receives a Windows socket handle for the new socket connection.

3The server socket generates an OnGetSocket event, passing in the Windows socket handle. If a TServerClientWinSocket object for the server endpoint of the new connection is not created in an OnGetSocket event handler, the server socket creates one. The TServerWinSocket object continues to listen for other clients.

4An OnAccept event occurs, using the new TServerClientWinSocket object.

5If ServerType is stThreadBlocking and no thread is available in the cache, an OnGetThread event occurs. If the OnGetThread event handler does not create a thread, the server socket creates a TServerClientThread.

6If ServerType is stThreadBlocking, an OnThreadStart event occurs as the thread begins execution.

7The client completes the connection to the TServerClientWinSocket object and an OnClientConnect event occurs.

Note: If ServerType is stThreadBlocking, make sure that all code in an OnClientConnect event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

Note: The OnClientConnect event handler for TServerSocket is also set when setting the OnClientConnect event handler of the associated TServerWinSocket.

See Also

OnAccept (see page 18795)

OnGetSocket (see page 18787)

OnGetThread (see page 18787)

OnListen (see page 18797)

OnThreadStart (see page 18788)

ServerType (see page 18782)

TServerClientWinSocket (see page 18840)

TServerWinSocket (see page 18858)

GetClientThread (see page 18865)

129.1.11.4.3 TServerSocket.OnClientDisconnect Event

Occurs when one of the connections to a client socket is closed.

Pascal

```
property OnClientDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientDisconnect;
```

Description

Write an OnClientDisconnect event handler to take specific action when the connection to a client socket is ending. The termination of a client connection does not close the server socket. The server socket remains open and listening for client requests on its listening connection.

The TServerClientWinSocket that describes the server endpoint of the client connection is freed after OnClientDisconnect. If ServerType is stThreadBlocking, OnThreadEnd occurs after OnClientDisconnect.

Note: If ServerType is stThreadBlocking, make sure that all code in an OnClientDisconnect event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

Note: The OnClientDisconnect event handler for TServerSocket is also set when setting the OnClientDisconnect event handler of the associated TServerWinSocket.

See Also

OnThreadEnd (see page 18788)

ServerType (see page 18782)

TServerClientWinSocket (see page 18840)

TServerWinSocket (see page 18858)

GetClientThread (see page 18865)

129.1.11.4.4 TServerSocket.OnClientError Event

Occurs when there is a failure in establishing, using, or terminating the socket connection to an individual client socket.

Pascal

```
property OnClientError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnClientError;
```

Description

Write an OnClientError event handler to respond to errors that arise with a connection to an individual client. Set the ErrorCode parameter to 0 if the OnClientError event handler successfully deals with the error condition to prevent an ESocketError from being raised.

Note: If ServerType is stThreadBlocking, make sure that all code in an OnClientError event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

Note: The OnClientError event handler for TServerSocket is also set when setting the OnClientError event handler of the associated TServerWinSocket.

See Also

ESocketError

ServerType (see page 18782)

TServerClientWinSocket (see page 18840)

GetClientThread (see page 18865)

129.1.11.4.5 TServerSocket.OnClientRead Event

Occurs when the server socket should read information from a client socket.

Pascal

```
property OnClientRead: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientRead;
```

Description

Write an OnClientRead event handler to write to the socket connection. If the ServerType property is stThreadBlocking, use a TWinSocketStream object to prevent problems that arise while writing from causing the execution thread to hang indefinitely. Otherwise, uUse the methods of the Socket parameter to perform the actual writing.

Note: If ServerType is stThreadBlocking, make sure that all code in an OnClientRead event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

See Also

ServerType (see page 18782)

receivebuf (see page 18812)

ReceiveText (see page 18812)

TServerClientWinSocket (see page 18840)

GetClientThread (see page 18865)

TWinSocketStream (see page 18876)

129.1.11.4.6 TServerSocket.OnClientWrite Event

Occurs when the server socket should write information to a client socket.

Pascal

```
property OnClientWrite: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientWrite;
```

Description

Write an OnClientWrite event handler to write to the socket connection. If the ServerType property is stThreadBlocking, use a TWinSocketStream object to prevent problems that arise while writing from causing the execution thread to hang indefinitely. Otherwise, uUse the methods of the Socket parameter to perform the actual writing.

Note: If ServerType is stThreadBlocking, make sure that all code in an OnClientWrite event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

See Also

ServerType (see page 18782)

SendBuf (see page 18813)

sendstream (see page 18814)

SendText (see page 18814)

GetClientThread (see page 18865)

TWinSocketStream (see page 18876)

129.1.11.4.7 TServerSocket.OnGetSocket Event

Occurs when the server socket needs to create a new TServerClientWinSocket object to represent the server endpoint of a connection to a client socket.

Pascal

```
property OnGetSocket: TGetSocketEvent;
```

C++

```
__property TGetSocketEvent OnGetSocket;
```

Description

Write an OnGetSocket event handler to create a specialized descendant of TServerClientWinSocket for the server socket component to use. Return the new TServerClientWinSocket object in the ClientSocket parameter. The Sender parameter is the TServerWinSocket object that describes the server endpoint of the listening connection.

See Also

TServerClientWinSocket (see page 18840)

Create (see page 18842)

TServerWinSocket (see page 18858)

129.1.11.4.8 TServerSocket.OnGetThread Event

Occurs when the server socket needs to create a new execution thread for a connection to a client socket.

Pascal

```
property OnGetThread: TGetThreadEvent;
```

C++

```
__property TGetThreadEvent OnGetThread;
```

Description

Write an OnGetThread event handler to create a specialized descendant of TServerClientThread for the connection to the client socket. Return the new TServerClientThread object in the SocketThread parameter.

Most applications that use thread-blocking server components will want to provide an OnGetThread event handler and implement a TServerClientThread that handles its own reading and writing in a thread-safe manner rather than relying on the default TServerClientThread which triggers OnClientRead and OnClientWrite events. This is because the OnClientRead and OnClientWrite event handlers reside on the server socket, which is in global memory.

The Sender parameter is the TServerWinSocket object that describes the server endpoint of the listening connection. The ClientSocket parameter is the TServerClientWinSocket object that describes the server endpoint of the connection to the client socket that is about to be formed.

See Also

ServerType (see page 18782)

TServerClientThread (see page 18831)

TServerClientWinSocket (see page 18840)

TServerWinSocket (see page 18858)

129.1.11.4.9 TServerSocket.OnListen Event

Occurs just before a server socket is opened for listening.

Pascal

```
property OnListen: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnListen;
```

Description

Write an OnListen event handler for a server socket to take specific action just before the socket is opened for listening. OnListen occurs after Address and Port have been bound to the socket connection, but before it is opened. This is the last opportunity to make changes to the socket endpoint before it is opened for listening. Use the SocketHandle property of the Socket parameter for Windows API calls that change the socket.

Server sockets open a socket connection for listening to establish a queue to hold client requests. The connection to a client socket is completed when a request from the queue is accepted.

See Also

Accept (see page 18803)

Address (see page 18750)

Port (see page 18751)

129.1.11.4.10 TServerSocket.OnThreadEnd Event

Occurs when a client socket connection is terminated and the associated thread finishes execution.

Pascal

```
property OnThreadEnd: TThreadNotifyEvent;
```

C++

```
__property TThreadNotifyEvent OnThreadEnd;
```

Description

Write an OnThreadEnd to take specific action when the thread for a client connection finishes execution. OnThreadEnd occurs after the OnClientDisconnect event.

The Sender parameter is the TServerWinSocket that describes the server endpoint of the listening connection. The Thread parameter is the thread that is finishing up.

See Also

OnClientDisconnect (see page 18785)

TServerWinSocket (see page 18858)

129.1.11.4.11 TServerSocket.OnThreadStart Event

Occurs when the execution thread for a connection to a client socket starts up.

Pascal

```
property OnThreadStart: TThreadNotifyEvent;
```

C++

```
__property TThreadNotifyEvent OnThreadStart;
```

Description

Write an OnThreadStart to initialize the TServerClientThread for the connection to a single client socket. The Sender parameter is the TServerWinSocket that describes the server endpoint of the listening connection. The Thread parameter is the thread that is about to start execution.

See Also

TServerClientThread (see page 18831)

TServerWinSocket (see page 18858)

129.1.12 TServerWinSocket Class

TServerWinSocket describes the endpoint of a listening server socket connection.

Class Hierarchy

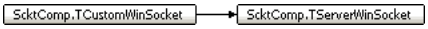

Pascal

```
TServerWinSocket = class(TCustomWinSocket);
```

C++

```
class TServerWinSocket : public TCustomWinSocket;
```

File

ScktComp

Description

TServerWinSocket is used by server socket components to manage the Windows socket API calls for a TCP/IP listening connection. TServerWinSocket introduces new properties, events, and methods to manage multiple socket connections to different client sockets, and to generate or reuse TServerClientThread (see page 18831) objects that allow each client connection to have its own execution thread.

See Also

TClientWinSocket (see page 18766)

TServerClientThread (see page 18831)

TServerClientWinSocket (see page 18840)

TServerSocket (see page 18843)

129.1.12.1 TServerWinSocket Members

The following tables list the members exposed by TServerWinSocket.

Protected Methods

	Name	Description
	AsyncInitSocket (see page 18803)	Looks up information or forms socket connections asynchronously.
	ClientConnect (see page 18863)	This is ClientConnect, a member of class TServerWinSocket.
	ClientDisconnect (see page 18863)	This is ClientDisconnect, a member of class TServerWinSocket.
	ClientErrorEvent (see page 18863)	This is ClientErrorEvent, a member of class TServerWinSocket.
	ClientRead (see page 18863)	This is ClientRead, a member of class TServerWinSocket.
	ClientWrite (see page 18863)	This is ClientWrite, a member of class TServerWinSocket.
V	DoCreateThread (see page 18864)	This is DoCreateThread, a member of class TServerWinSocket.
	DoListen (see page 18806)	Opens a listening connection.
	DoOpen (see page 18806)	Opens a connection to a remote socket.
	Error (see page 18806)	Generates an OnErrorEvent (see page 18824) event.
	Event (see page 18807)	This is Event, a member of class TCustomWinSocket.
	GetClientSocket (see page 18864)	This is GetClientSocket, a member of class TServerWinSocket.
	GetServerThread (see page 18865)	This is GetServerThread, a member of class TServerWinSocket.
	InitSocket (see page 18807)	Generates a windows Internet socket address structure from a description of the desired socket port.
	Listen (see page 18865)	Opens a listening connection for a server socket.
	SetAsyncStyles (see page 18815)	Sets the AsyncStyles property.
	SetServerType (see page 18866)	This is SetServerType, a member of class TServerWinSocket.
	SetThreadCacheSize (see page 18866)	This is SetThreadCacheSize, a member of class TServerWinSocket.
	ThreadEnd (see page 18866)	This is ThreadEnd, a member of class TServerWinSocket.
	ThreadStart (see page 18867)	This is ThreadStart, a member of class TServerWinSocket.

Public Events

	Name	Description
	OnClientConnect (see page 18871)	Occurs when a client socket completes a connection accepted by the listening socket.
	OnClientDisconnect (see page 18872)	Occurs when one of the connections to a client socket is closed.
	OnClientError (see page 18872)	Occurs when a failure is encountered when establishing, using, or terminating the socket connection to an individual client socket.
	OnClientRead (see page 18873)	Occurs when the server socket should read information from a client socket.
	OnClientWrite (see page 18874)	Occurs when the server socket should write information to a client socket.
	OnErrorEvent (see page 18824)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnGetSocket (see page 18874)	Occurs when the server socket needs to create a new TServerClientWinSocket (see page 18840) object to form the connection to a client socket.
	OnGetThread (see page 18875)	Occurs when the server socket needs to create a new execution thread for a connection to a client socket.
	OnSocketEvent (see page 18824)	Occurs when the Windows socket receives notification of asynchronous events.
	OnThreadEnd (see page 18875)	Occurs when a client socket connection is terminated and the associated thread finishes execution.
	OnThreadStart (see page 18876)	Occurs when the execution thread for a connection to a client socket starts up.

Public Methods

	Name	Description
V	Accept (see page 18862)	Accepts a connection to a client socket.
	Close (see page 18804)	Shuts down the socket connection if it is open.
V	Connect (see page 18805)	Provides the interface for a method that responds to connection notifications from client sockets.
	Create (see page 18861)	Creates an instance of TServerWinSocket (see page 18858) from a Windows socket handle.

V	DefaultHandler (see page 18805)	Processes message records.
V	Destroy (see page 18861)	Destroys an instance of TServerWinSocket (see page 18858).
V	Disconnect (see page 18864)	Shuts down the socket connection represented by a Windows socket handle.
	GetClientThread (see page 18865)	Returns the execution thread used by the client connection managed by the ClientSocket parameter.
	Lock (see page 18809)	Starts a critical section, blocking all other execution threads until the Unlock (see page 18815) method is called.
	LookupName (see page 18809)	Generates the portion of a Windows Internet socket address structure that describes the host.
	LookupService (see page 18810)	Returns the port number associated with a specific service.
	Open (see page 18810)	Opens a connection to a remote socket.
V	Read (see page 18811)	Generates an OnSocketEvent (see page 18824) of type seRead in response to notifications that the socket connection is ready.
	ReceiveBuf (see page 18812)	This is ReceiveBuf, a member of class TCustomWinSocket.
	ReceiveLength (see page 18812)	Returns the number of bytes ready to be sent over the socket connection.
	ReceiveText (see page 18812)	Reads a string from the socket connection.
	SendBuf (see page 18813)	Writes Count bytes to the socket connection from the Buf parameter.
	SendStream (see page 18814)	This is SendStream, a member of class TCustomWinSocket.
	SendStreamThenDrop (see page 18814)	Writes all the information that can be read from the AStream parameter to the socket connection and then terminates the connection.
	SendText (see page 18814)	Writes the string S to the socket connection.
	Unlock (see page 18815)	Reenables execution in other threads after a critical section begun by the Lock (see page 18809) method.
V	Write (see page 18816)	Generates an OnSocketEvent (see page 18824) of type seWrite in response to notifications that the socket connection is ready.

Public Properties

	Name	Description
R	ActiveConnections (see page 18867)	Indicates the number of connections to client sockets accepted by this listening socket that are currently open.
R	ActiveThreads (see page 18867)	Indicates the number of TServerClientThread (see page 18831) objects currently in use by the socket connections listed in the Connections (see page 18868) array.
R	Addr (see page 18817)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the socket port.
	ASyncStyles (see page 18817)	Determines which asynchronous events the socket can receive.
R	Connected (see page 18818)	Indicates whether the socket connection is open and available for communication with other machines.
R	Connections (see page 18868)	Lists all TServerClientWinSocket (see page 18840) objects that represent open socket connections to client sockets that were accepted by this listening socket.
	Data (see page 18818)	Points to application-specific data associated with the socket.
R	Handle (see page 18819)	Specifies the handle of the window that receives notifications from the Windows socket.
R	IdleThreads (see page 18869)	Indicates the number of unused TServerClientThread (see page 18831) objects available in the cache.
R	LocalAddress (see page 18819)	Specifies the IP address of the local system that was bound to the socket connection.
R	LocalHost (see page 18820)	Specifies the name of the local system.
R	LocalPort (see page 18820)	Specifies the ID number of the port used by the socket connection.
R	LookupState (see page 18821)	Indicates any asynchronous activities occurring while opening a connection.
R	RemoteAddr (see page 18821)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the remote socket port.
R	RemoteAddress (see page 18822)	Specifies the IP address of the remote system that was bound to the socket connection.
R	RemoteHost (see page 18822)	Specifies the name of the remote system.
R	RemotePort (see page 18823)	Specifies the ID number of the port used by the RemoteHost (see page 18822) to form the socket connection.
	ServerType (see page 18869)	Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.
R	SocketHandle (see page 18823)	Specifies the Windows handle for the socket.

	ThreadCacheSize (see page 18870)	Indicates the number of TServerClientThread (see page 18831) objects cached for reuse by new connections to client sockets.

129.1.12.2 TServerWinSocket.Create Constructor

Creates an instance of TServerWinSocket (see page 18858) from a Windows socket handle.

Pascal

```
constructor Create(ASocket: TSocket);
```

C++

```
__fastcall TServerWinSocket(TSocket ASocket);
```

Description

Call Create to create a Windows listening socket object. Windows listening socket objects are created automatically by the server socket component that uses them.

Create allocates the helper objects that manage the list of active client connections and the thread cache before calling the inherited constructor. Create also initializes the ASyncStyles (see page 18817) property to [asAccept].

See Also

ASyncStyles (see page 18817)

Connections (see page 18868)

Create (see page 18801)

129.1.12.3 TServerWinSocket.Destroy Destructor

Destroys an instance of TServerWinSocket (see page 18858).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TServerWinSocket();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the Windows listening socket object is not nil, and only then calls Destroy. Unless the Windows socket object is not used by a server socket component, it is freed automatically by the socket component.

Destroy frees the helper objects that manage the list of active client connections and the thread cache.

See Also

Free (see page 22300)

TServerSocket (see page 18843)

129.1.12.4 TServerWinSocket Methods

The methods of the TServerWinSocket class are listed here.

Protected Methods

	Name	Description
	ClientConnect (see page 18863)	This is ClientConnect, a member of class TServerWinSocket.
	ClientDisconnect (see page 18863)	This is ClientDisconnect, a member of class TServerWinSocket.
	ClientErrorEvent (see page 18863)	This is ClientErrorEvent, a member of class TServerWinSocket.
	ClientRead (see page 18863)	This is ClientRead, a member of class TServerWinSocket.
	ClientWrite (see page 18863)	This is ClientWrite, a member of class TServerWinSocket.
V	DoCreateThread (see page 18864)	This is DoCreateThread, a member of class TServerWinSocket.
	GetClientSocket (see page 18864)	This is GetClientSocket, a member of class TServerWinSocket.
	GetServerThread (see page 18865)	This is GetServerThread, a member of class TServerWinSocket.
	Listen (see page 18865)	Opens a listening connection for a server socket.
	SetServerType (see page 18866)	This is SetServerType, a member of class TServerWinSocket.
	SetThreadCacheSize (see page 18866)	This is SetThreadCacheSize, a member of class TServerWinSocket.
	ThreadEnd (see page 18866)	This is ThreadEnd, a member of class TServerWinSocket.
	ThreadStart (see page 18867)	This is ThreadStart, a member of class TServerWinSocket.

Public Methods

	Name	Description
V	Accept (see page 18862)	Accepts a connection to a client socket.
V	Disconnect (see page 18864)	Shuts down the socket connection represented by a Windows socket handle.
	GetClientThread (see page 18865)	Returns the execution thread used by the client connection managed by the ClientSocket parameter.

129.1.12.4.1 TServerWinSocket.Accept Method

Accepts a connection to a client socket.

Pascal

```
procedure Accept(Socket: TSocket); override;
```

C++

```
virtual __fastcall Accept(TSocket Socket);
```

Description

Accept is called automatically in response to notifications that a client socket is requesting a connection. The Socket parameter is the Windows socket handle for the listening connection.

Accept accepts the client connection request, obtaining the Windows socket handle for the client connection. It then generates an OnGetSocket (see page 18874) event, passing this socket handle as an argument to any event handler. If a TServerClientWinSocket (see page 18840) is not created by an OnGetSocket (see page 18874) event handler, Accept creates a TServerClientWinSocket (see page 18840) to represent the server endpoint of the accepted connection. Next, Accept obtains a running TServerClientThread (see page 18831), either by restarting a thread in the cache, obtaining one from an OnGetThread (see page 18875) event, or creating a new TServerClientThread (see page 18831) object. This new thread handles the connection to the client socket.

Accept is only called if the ASyncStyles (see page 18817) property includes asAccept.

See Also

ASyncStyles (see page 18817)

OnGetSocket (see page 18874)

OnGetThread (see page 18875)

129.1.12.4.2 TServerWinSocket.ClientConnect Method

Pascal

```
procedure ClientConnect(Socket: TCustomWinSOcket); dynamic;
```

C++

```
__fastcall ClientConnect(TCustomWinSOcket Socket);
```

Description

This is ClientConnect, a member of class TServerWinSocket.

129.1.12.4.3 TServerWinSocket.ClientDisconnect Method

Pascal

```
procedure ClientDisconnect(Socket: TCustomWinSOcket); dynamic;
```

C++

```
__fastcall ClientDisconnect(TCustomWinSOcket Socket);
```

Description

This is ClientDisconnect, a member of class TServerWinSocket.

129.1.12.4.4 TServerWinSocket.ClientErrorEvent Method

Pascal

```
procedure ClientErrorEvent(Socket: TCustomWinSocket; ErrorEvent: TErrorEvent; var ErrorCode:
Integer); dynamic;
```

C++

```
__fastcall ClientErrorEvent(TCustomWinSocket Socket, TErrorEvent ErrorEvent, int ErrorCode);
```

Description

This is ClientErrorEvent, a member of class TServerWinSocket.

129.1.12.4.5 TServerWinSocket.ClientRead Method

Pascal

```
procedure ClientRead(Socket: TCustomWinSocket); dynamic;
```

C++

```
__fastcall ClientRead(TCustomWinSocket Socket);
```

Description

This is ClientRead, a member of class TServerWinSocket.

129.1.12.4.6 TServerWinSocket.ClientWrite Method

Pascal

```
procedure ClientWrite(Socket: TCustomWinSOcket); dynamic;
```

C++

```
__fastcall ClientWrite(TCustomWinSOcket Socket);
```

Description

This is ClientWrite, a member of class TServerWinSocket.

129.1.12.4.7 TServerWinSocket.Disconnect Method

Shuts down the socket connection represented by a Windows socket handle.

Pascal

```
procedure Disconnect(Socket: TSocket); override;
```

C++

```
virtual __fastcall Disconnect(TSocket Socket);
```

Description

Disconnect is called by the Close (see page 18804) method to terminate the socket connection.

Disconnect calls the Lock (see page 18809) method to make sure the method is thread-safe. Then, it terminates all open connections to client sockets. Next it generates an OnSocketEvent (see page 18824) of type seDisconnect. Finally, it closes the listening connection and calls Unlock (see page 18815) to end the critical section started by Lock (see page 18809).

See Also

Close (see page 18804)

Lock (see page 18809)

OnSocketEvent (see page 18824)

Unlock (see page 18815)

129.1.12.4.8 TServerWinSocket.DoCreateThread Method

Pascal

```
function DoCreateThread(ClientSocket: TServerClientWinSocket): TServerClientThread; virtual;
```

C++

```
virtual __fastcall TServerClientThread DoCreateThread(TServerClientWinSocket ClientSocket);
```

Description

This is DoCreateThread, a member of class TServerWinSocket.

129.1.12.4.9 TServerWinSocket.GetClientSocket Method

Pascal

```
function GetClientSocket(Socket: TSocket): TServerClientWinSocket; dynamic;
```

C++

```
__fastcall TServerClientWinSocket GetClientSocket(TSocket Socket);
```

Description

This is GetClientSocket, a member of class TServerWinSocket.

129.1.12.4.10 TServerWinSocket.GetClientThread Method

Returns the execution thread used by the client connection managed by the ClientSocket parameter.

Pascal

```
function GetClientThread(ClientSocket: TServerClientWinSocket): TServerClientThread;
```

C++

```
__fastcall TServerClientThread GetClientThread(TServerClientWinSocket ClientSocket);
```

Description

Use GetClientThread to obtain access to the TServerClientThread (see page 18831) for a particular client connection. GetClientThread allows an OnClient... event handler to access thread-global information saved in the Data (see page 18818) property of the thread.

See Also

OnClientConnect (see page 18871)

OnClientDisconnect (see page 18872)

OnClientError (see page 18872)

OnClientRead (see page 18873)

OnClientWrite (see page 18874)

Data (see page 18838)

129.1.12.4.11 TServerWinSocket.GetServerThread Method

Pascal

```
function GetServerThread(ClientSocket: TServerClientWinSocket): TServerClientThread; dynamic;
```

C++

```
__fastcall TServerClientThread GetServerThread(TServerClientWinSocket ClientSocket);
```

Description

This is GetServerThread, a member of class TServerWinSocket.

129.1.12.4.12 TServerWinSocket.Listen Method

Opens a listening connection for a server socket.

Pascal

```
procedure Listen(var Name: string; var Address: string; var Service: string; Port: Word;
QueueSize: Integer);
```

C++

```
__fastcall Listen(AnsiString Name, AnsiString Address, AnsiString Service, Word Port, int
QueueSize);
```

Description

Server socket components call Listen to open a listening connection.

Listen creates the Windows socket that will make the listening connection, and binds the Windows Internet socket address derived from the parameters to the socket. Listen then generates an OnSocketEvent (see page 18824) of type seListen.

Finally, Listen opens the socket as a listening connection with a queue that can hold QueueSize client requests, setting the Connected (see page 18818) property to true.

The Name, Address, Service, and Port parameters are taken from the Host, Address, Service and Port properties of the server socket component. The QueueSize is set to 5 by TServerSocket (see page 18843).

See Also

Connected (see page 18818)

InitSocket (see page 18807)

OnSocketEvent (see page 18824)

Address (see page 18750)

Host (see page 18751)

Port (see page 18751)

Service (see page 18752)

TServerSocket (see page 18843)

129.1.12.4.13 TServerWinSocket.SetServerType Method

Pascal

```
procedure SetServerType(Value: TServerType);
```

C++

```
__fastcall SetServerType(TServerType Value);
```

Description

This is SetServerType, a member of class TServerWinSocket.

129.1.12.4.14 TServerWinSocket.SetThreadCacheSize Method

Pascal

```
procedure SetThreadCacheSize(Value: Integer);
```

C++

```
__fastcall SetThreadCacheSize(int Value);
```

Description

This is SetThreadCacheSize, a member of class TServerWinSocket.

129.1.12.4.15 TServerWinSocket.ThreadEnd Method

Pascal

```
procedure ThreadEnd(AThread: TServerClientThread); dynamic;
```

C++

```
__fastcall ThreadEnd(TServerClientThread AThread);
```

Description

This is ThreadEnd, a member of class TServerWinSocket.

129.1.12.4.16 TServerWinSocket.ThreadStart Method

Pascal

```
procedure ThreadStart(AThread: TServerClientThread); dynamic;
```

C++

```
__fastcall ThreadStart(TServerClientThread AThread);
```

Description

This is ThreadStart, a member of class TServerWinSocket.

129.1.12.5 TServerWinSocket Properties

The properties of the TServerWinSocket class are listed here.

Public Properties

	Name	Description
R	ActiveConnections (see page 18867)	Indicates the number of connections to client sockets accepted by this listening socket that are currently open.
R	ActiveThreads (see page 18867)	Indicates the number of TServerClientThread (see page 18831) objects currently in use by the socket connections listed in the Connections (see page 18868) array.
R	Connections (see page 18868)	Lists all TServerClientWinSocket (see page 18840) objects that represent open socket connections to client sockets that were accepted by this listening socket.
R	IdleThreads (see page 18869)	Indicates the number of unused TServerClientThread (see page 18831) objects available in the cache.
	ServerType (see page 18869)	Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.
	ThreadCacheSize (see page 18870)	Indicates the number of TServerClientThread (see page 18831) objects cached for reuse by new connections to client sockets.

129.1.12.5.1 TServerWinSocket.ActiveConnections Property

Indicates the number of connections to client sockets accepted by this listening socket that are currently open.

Pascal

```
property ActiveConnections: Integer;
```

C++

```
__property int ActiveConnections;
```

Description

Use ActiveConnections to monitor the use of the server socket. ActiveConnections is the number of entries in the Connections (see page 18868) array.

See Also

Connections (see page 18868)

129.1.12.5.2 TServerWinSocket.ActiveThreads Property

Indicates the number of TServerClientThread (see page 18831) objects currently in use by the socket connections listed in the Connections (see page 18868) array.

Pascal

```
property ActiveThreads: Integer;
```

C++

```
__property int ActiveThreads;
```

Description

Use ActiveThreads to monitor thread usage by the listening socket. If ActiveThreads is usually much higher than ThreadCacheSize (see page 18870), increasing ThreadCacheSize (see page 18870) should improve performance. To determine whether ThreadCacheSize (see page 18870) is too high, use the IdleThreads (see page 18869) property.

When the ServerType (see page 18869) property is stThreadBlocking, each client connection automatically spawns a thread which is added to the cache and counted by ActiveThreads.

See Also

Connections (see page 18868)

GetClientThread (see page 18865)

IdleThreads (see page 18869)

ServerType (see page 18869)

ThreadCacheSize (see page 18870)

129.1.12.5.3 TServerWinSocket.Connections Property

Lists all TServerClientWinSocket (see page 18840) objects that represent open socket connections to client sockets that were accepted by this listening socket.

Pascal

```
property Connections [Index: Integer]: TCustomWinSocket;
```

C++

```
__property TCustomWinSocket Connections[int Index];
```

Description

Use Connections with the ActiveConnections (see page 18867) property to iterate over all open socket connections accepted by the listening socket. Each entry in the Connections array is a TServerClientWinSocket (see page 18840) object, where an Index of 0 gives the first TServerClientWinSocket (see page 18840), an Index of 1 returns the second TServerClientWinSocket (see page 18840), and so on.

If ServerType (see page 18869) is stThreadBlocking, the thread associated with each of the TServerClientWinSocket (see page 18840) objects in the Connections array can be obtained using the GetClientThread (see page 18865) method.

See Also

ActiveConnections (see page 18867)

GetClientThread (see page 18865)

ServerType (see page 18869)

TServerClientWinSocket (see page 18840)

129.1.12.5.4 TServerWinSocket.IdleThreads Property

Indicates the number of unused TServerClientThread (see page 18831) objects available in the cache.

Pascal

```
property IdleThreads: Integer;
```

C++

```
__property int IdleThreads;
```

Description

Use IdleThreads to monitor thread cache usage by the listening socket. If IdleThreads often very high, the thread cache is tying up a lot of unused memory storing extra threads. Monitor the ActiveThreads (see page 18867) property to determine an appropriate value for ThreadCacheSize (see page 18870).

When the ServerType (see page 18869) property is stThreadBlocking, each client connection automatically spawns a thread which is added to the cache. As the threads in the cache complete their execution, they become available for reuse, and are counted by IdleThreads. When there are more active threads than ThreadCacheSize (see page 18870), the extra threads are freed when they finish execution instead.

See Also

ActiveThreads (see page 18867)

ServerType (see page 18869)

ThreadCacheSize (see page 18870)

129.1.12.5.5 TServerWinSocket.ServerType Property

Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.

Pascal

```
property ServerType: TServerType;
```

C++

```
__property TServerType ServerType;
```

Description

Set ServerType to stThreadBlocking to automatically spawn a new thread for each socket connection accepted by the listening socket connection. When ServerType is stThreadBlocking, the TServerClientThread (see page 18831) object for a connection generates OnClientRead (see page 18873) or OnClientWrite (see page 18874) events when the server socket needs to read or write.

Set ServerType to stNonBlocking to handle all reading and writing over socket connections accepted by the listening socket asynchronously. When ServerType is stNonBlocking, all client connections are handled in a single execution thread by default. OnClientRead (see page 18873) or OnClientWrite (see page 18874) events occur when the server socket needs to attend to I/O over one of the connections.

Use a non-blocking socket when reading and writing must be synchronized with client sockets.

Note: When ServerType is stThreadBlocking, it is important that the OnClient...

event handlers contain thread-safe code.

See Also

ASyncStyles (see page 18817)

OnClientRead (see page 18873)

OnClientWrite (see page 18874)

OnGetThread (see page 18875)

TServerClientThread (see page 18831)

129.1.12.5.6 TServerWinSocket.ThreadCacheSize Property

Indicates the number of TServerClientThread (see page 18831) objects cached for reuse by new connections to client sockets.

Pascal

```
property ThreadCacheSize: Integer;
```

C++

```
__property int ThreadCacheSize;
```

Description

When ServerType (see page 18869) is stThreadBlocking, each new connection that is accepted by the server socket is given a separate execution thread. In order to improve performance, server sockets store these threads in a cache rather than freeing them when the connection is closed. New connections can then reuse threads from the cache, rather than requiring the socket server to create a new thread every time a connection is accepted.

Set ThreadCacheSize to specify the number of threads that are cached for reuse. The ideal value for ThreadCacheSize depends on the number and frequency of client socket requests received by the server socket. If ThreadCacheSize is too low, the server socket will spend more time freeing and creating threads when client connections are accepted. If ThreadCacheSize is too high, the server socket may unnecessarily lock up the memory for threads that are never reused.

See Also

ActiveThreads (see page 18867)

IdleThreads (see page 18869)

ServerType (see page 18869)

129.1.12.6 TServerWinSocket Events

The events of the TServerWinSocket class are listed here.

Public Events

	Name	Description
	OnClientConnect (see page 18871)	Occurs when a client socket completes a connection accepted by the listening socket.
	OnClientDisconnect (see page 18872)	Occurs when one of the connections to a client socket is closed.
	OnClientError (see page 18872)	Occurs when a failure is encountered when establishing, using, or terminating the socket connection to an individual client socket.
	OnClientRead (see page 18873)	Occurs when the server socket should read information from a client socket.
	OnClientWrite (see page 18874)	Occurs when the server socket should write information to a client socket.
	OnGetSocket (see page 18874)	Occurs when the server socket needs to create a new TServerClientWinSocket (see page 18840) object to form the connection to a client socket.
	OnGetThread (see page 18875)	Occurs when the server socket needs to create a new execution thread for a connection to a client socket.

	OnThreadEnd (see page 18875)	Occurs when a client socket connection is terminated and the associated thread finishes execution.
	OnThreadStart (see page 18876)	Occurs when the execution thread for a connection to a client socket starts up.

129.1.12.6.1 TServerWinSocket.OnClientConnect Event

Occurs when a client socket completes a connection accepted by the listening socket.

Pascal

```
property OnClientConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientConnect;
```

Description

Write (see page 18816) an OnClientConnect event handler to take specific action when a client socket completes the socket connection to a TServerClientWinSocket (see page 18840) object. Depending on the service, this is the point when the socket should begin reading or writing.

The order of Windows server socket events leading up to OnClientConnect is as follows:

1An OnSocketEvent (see page 18824) event of type seListen occurs just before the Windows socket is opened for listening.

2The listening Windows socket receives client requests in a listening queue. The listening socket accepts one of those requests, and receives a Windows socket handle for the new socket connection.

3The listening socket generates an OnGetSocket (see page 18874) event, passing in the Windows socket handle. If a TServerClientWinSocket (see page 18840) object for the Windows socket handle was not created in the OnGetSocket (see page 18874) event handler, the Windows socket creates one. The Windows socket continues to listen for other clients.

4An OnSocketEvent (see page 18824) event of type seAccept occurs, using the new TServerClientWinSocket (see page 18840) object.

5If ServerType (see page 18869) is stThreadBlocking and no thread is available in the cache, an OnGetThread (see page 18875) event occurs. If the OnGetThread (see page 18875) event handler does not create a thread, the Windows socket object creates a TServerClientThread (see page 18831).

6If ServerType (see page 18869) is stThreadBlocking, an OnThreadStart (see page 18876) event occurs as the thread begins execution.

7The client completes the connection to the TServerClientWinSocket (see page 18840) object and an OnClientConnect event occurs.

The Sender parameter is the TServerWinClient object that listens for client requests. The Socket parameter is the TServerClientWinSocket (see page 18840) object that forms a connection to the client socket.

Note: If ServerType (see page 18869) is stThreadBlocking, make sure that all code in an OnClientConnect event handler is thread-safe. Use the GetClientThread (see page 18865) method to access thread-specific information.

Note: The OnClientConnect event handler for TServerWinSocket (see page 18858) is also set when setting the OnClientConnect event handler of the associated TServerSocket (see page 18843).

See Also

GetClientThread (see page 18865)

OnGetSocket (see page 18874)

OnGetThread (see page 18875)

OnSocketEvent (see page 18824)

OnThreadStart (see page 18876)

ServerType (see page 18869)

TServerClientWinSocket (see page 18840)

OnClientConnect (see page 18784)

129.1.12.6.2 TServerWinSocket.OnClientDisconnect Event

Occurs when one of the connections to a client socket is closed.

Pascal

```
property OnClientDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientDisconnect;
```

Description

Write (see page 18816) an OnClientDisconnect event handler to take specific action when the connection to a client socket is ending. The termination of a client request does not close the listening socket. The listening connection remains open for accepting new client connection requests.

The TServerClientWinSocket (see page 18840) object associated with the client connection is freed after OnClientDisconnect. If ServerType (see page 18869) is stThreadBlocking, OnThreadEnd (see page 18875) occurs after OnClientDisconnect.

The Sender parameter is the TServerWinClient object that received the client request. The Socket parameter is the TServerClientWinSocket (see page 18840) object that is ending a connection with the client socket.

Note: If ServerType (see page 18869) is stThreadBlocking, make sure that all code in an OnClientDisconnect event handler is thread-safe.Use the GetClientThread (see page 18865) method to access thread-specific information.

The OnClientDisconnect event handler for TServerWinSocket (see page 18858) is also set when setting the OnClientDisconnect event handler of the associated TServerSocket (see page 18843).

See Also

GetClientThread (see page 18865)

OnThreadEnd (see page 18875)

ServerType (see page 18869)

TServerClientWinSocket (see page 18840)

OnClientDisconnect (see page 18785)

129.1.12.6.3 TServerWinSocket.OnClientError Event

Occurs when a failure is encountered when establishing, using, or terminating the socket connection to an individual client socket.

Pascal

```
property OnClientError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnClientError;
```

Description

Write (see page 18816) an OnClientError event handler to respond to errors that arise with the connection to a client socket. Set the ErrorCode parameter to 0 if the OnClientError event handler successfully deals with the error condition, to prevent an ESocketError (see page 18739) from being raised.

The Sender parameter is the TServerWinClient object that received the client request. The Socket parameter is the TServerClientWinSocket (see page 18840) object that manages the connection to the client. The ErrorEvent parameter indicates what Socket was attempting to do when the error occurred. The ErrorCode parameter is the error code returned by the Windows socket API call.

Note: If ServerType (see page 18869) is stThreadBlocking, make sure that all code in an OnClientError event handler is thread-safe. Use the GetClientThread (see page 18865) method to access thread-specific information.

Note: The OnClientError event handler for TServerWinSocket (see page 18858) is also set when setting the OnClientError event handler of the associated TServerSocket (see page 18843).

See Also

ESocketError

GetClientThread (see page 18865)

ServerType (see page 18869)

TServerClientWinSocket (see page 18840)

OnClientError (see page 18785)

129.1.12.6.4 TServerWinSocket.OnClientRead Event

Occurs when the server socket should read information from a client socket.

Pascal

```
property OnClientRead: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientRead;
```

Description

Write (see page 18816) an OnClientRead event handler to read from the socket connection. Use a TWinSocketStream (see page 18876) object or the methods of the Socket parameter to do the reading.

The Sender parameter is the TServerWinClient object that received the client request. The Socket parameter is the TServerClientWinSocket (see page 18840) object that encapsulates the Windows socket connection.

Note: If ServerType (see page 18869) is stThreadBlocking, make sure that all code in an OnClientRead event handler is thread-safe. Use the GetClientThread (see page 18865) method to access thread-specific information.

Note: The OnClientRead event handler for TServerWinSocket (see page 18858) is also set when setting the OnClientRead event handler of the associated TServerSocket (see page 18843).

See Also

GetClientThread (see page 18865)

ServerType (see page 18869)

receivebuf (see page 18812)

ReceiveText (see page 18812)

TServerClientWinSocket (see page 18840)

OnClientRead (see page 18786)

TWinSocketStream (see page 18876)

129.1.12.6.5 TServerWinSocket.OnClientWrite Event

Occurs when the server socket should write information to a client socket.

Pascal

```
property OnClientWrite: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientWrite;
```

Description

Write (see page 18816) an OnClientWrite event handler to write to the socket connection. Use a TWinSocketStream (see page 18876) object or the methods of the Socket parameter to do the actual writing.

The Sender parameter is the TServerWinClient object that received the client request. The Socket parameter is the TServerClientWinSocket (see page 18840) object that encapsulates the Windows socket connection.

Note: If ServerType (see page 18869) is stThreadBlocking, make sure that all code in an OnClientWrite event handler is thread-safe. Use the GetClientThread (see page 18865) method to access thread-specific information.

Note: The OnClientWrite event handler for TServerWinSocket (see page 18858) is also set when setting the OnClientWrite event handler of the associated TServerSocket (see page 18843).

See Also

GetClientThread (see page 18865)

ServerType (see page 18869)

SendBuf (see page 18813)

sendstream (see page 18814)

SendText (see page 18814)

OnClientWrite (see page 18786)

129.1.12.6.6 TServerWinSocket.OnGetSocket Event

Occurs when the server socket needs to create a new TServerClientWinSocket (see page 18840) object to form the connection to a client socket.

Pascal

```
property OnGetSocket: TGetSocketEvent;
```

C++

```
__property TGetSocketEvent OnGetSocket;
```

Description

Write (see page 18816) an OnGetSocket event handler to create a specialized TServerClientWinSocket (see page 18840) object to use in the connection represented by the Socket parameter. Return the new TServerClientWinSocket (see page 18840) object in the ClientSocket parameter.

The Sender parameter is the TServerWinSocket (see page 18858) object that received the client request. The Socket parameter is the Windows socket handle for the socket connection to the client socket.

Note: The OnGetSocket event handler for TServerWinSocket (see page 18858) is also set when setting the OnGetSocket event handler of the associated TServerSocket (see page 18843).

See Also

Create (see page 18842)

OnGetSocket (see page 18787)

129.1.12.6.7 TServerWinSocket.OnGetThread Event

Occurs when the server socket needs to create a new execution thread for a connection to a client socket.

Pascal

```
property OnGetThread: TGetThreadEvent;
```

C++

```
__property TGetThreadEvent OnGetThread;
```

Description

Write (see page 18816) an OnGetThread event handler to create a specialized descendant of TServerClientThread (see page 18831) for the connection to the client socket. Create (see page 18861) the new thread with the CreateSuspended parameter set to false, and return it in the SocketThread parameter. OnGetThread only occurs if there are no idle threads in the cache.

Most applications that use thread-blocking connections will want to create a new descendant of TServerClientThread (see page 18831) in an OnGetThread event handler. This is because the default behavior of TServerClientThread (see page 18831) uses the OnClientRead (see page 18873) and OnClientWrite (see page 18874) event handlers for reading and writing. These events occur on the server socket, which is not thread-local.

The Sender parameter is the TServerWinSocket (see page 18858) object that received the client request. The ClientSocket parameter is the TServerClientWinSocket (see page 18840) object that will communicate with the client socket.

Note: The OnGetThread event handler for TServerWinSocket (see page 18858) is also set when setting the OnGetThread event handler of the associated TServerSocket (see page 18843).

See Also

IdleThreads (see page 18869)

ServerType (see page 18869)

OnGetThread (see page 18787)

129.1.12.6.8 TServerWinSocket.OnThreadEnd Event

Occurs when a client socket connection is terminated and the associated thread finishes execution.

Pascal

```
property OnThreadEnd: TThreadNotifyEvent;
```

C++

```
__property TThreadNotifyEvent OnThreadEnd;
```

Description

Write (see page 18816) an OnThreadEnd to take specific action when the thread for a client connection finishes execution. OnThreadEnd occurs after the OnClientDisconnect (see page 18872) event.

The Sender parameter is the TServerWinSocket (see page 18858) that received the client request. The Thread parameter is the thread that is finishing up.

Note: The OnThreadEnd event handler for TServerWinSocket (see page 18858) is also set when setting the OnThreadEnd event handler of the associated TServerSocket (see page 18843).

See Also

OnClientDisconnect (see page 18872)

OnThreadEnd (see page 18788)

129.1.12.6.9 TServerWinSocket.OnThreadStart Event

Occurs when the execution thread for a connection to a client socket starts up.

Pascal

```
property OnThreadStart: TThreadNotifyEvent;
```

C++

```
__property TThreadNotifyEvent OnThreadStart;
```

Description

Write (see page 18816) an OnThreadStart to initialize the TServerClientThread (see page 18831) for the connection to a single client socket. The Sender parameter is the TServerWinSocket (see page 18858) that received the client request. The Thread parameter is the thread that is about to start execution.

Note: The OnThreadStart event handler for TServerWinSocket (see page 18858) is also set when setting the OnThreadStart event handler of the associated TServerSocket (see page 18843).

See Also

OnThreadStart (see page 18788)

129.1.13 TWinSocketStream Class

TWinSocketStream is a stream that provides services which allow applications to read from or write to socket connections.

Class Hierarchy

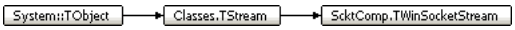

Pascal

```
TWinSocketStream = class(TStream);
```

C++

```
class TWinSocketStream : public TStream;
```

File

ScktComp

Description

Use TWinSocketStream to read or write information over a blocking socket connection. Windows socket objects include methods to read from or write to the socket connection they represent. However, these methods do not provide a mechanism for timing out when the socket connection is dropped or for waiting until the socket connection is ready before reading.

When the socket is a non-blocking socket, this lack of a time-out or waiting mechanism is not a problem, because reading and writing occur asynchronously in response to notifications from the socket connection. For blocking sockets, however, these mechanisms provided by TWinSocketStream are necessary so that the application using the socket does not hang indefinitely.

To use a Windows socket stream, create an instance of TWinSocketStream, use the methods of the stream to read or write the data, and then free the Windows socket stream.

Note: TWinSocketStream does not work with non-blocking sockets.

See Also

TClientSocket (see page 18752)

TClientWinSocket (see page 18766)

TServerClientWinSocket (see page 18840)

TServerSocket (see page 18843)

129.1.13.1 TWinSocketStream Members

The following tables list the members exposed by TWinSocketStream.

Protected Methods

	Name	Description
	GetSize (see page 2982)	This is GetSize, a member of class TStream.
	SetSize (see page 2989)	This is the overview for the SetSize method overload.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CopyFrom (see page 2979)	Copies a specified number of bytes from one stream to another.
	Create (see page 18878)	Creates an instance of TWinSocketStream (see page 18876) for a Windows socket object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 18879)	Destroys an instance of TWinSocketStream (see page 18876).

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FixupResourceHeader (see page 2981)	Patches the resource header for a resource that has been written to the stream.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Read (see page 18880)	Reads up to Count bytes from the socket connection into Buffer.
	ReadBuffer (see page 2983)	Reads bytes from the stream into Buffer.
	ReadComponent (see page 2983)	Initiates streaming of components and their properties.
	ReadComponentRes (see page 2987)	Reads components and their properties from a stream in a resource file format.
	ReadResHeader (see page 2987)	Reads a resource-file header from the stream.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Seek (see page 18880)	Always returns 0.
	Seek (see page 2988)	This is the overview for the Seek method overload.
	WaitForData (see page 18881)	Waits up to TimeOut (see page 18882) milliseconds for the socket connection to be ready to transfer data.
	Write (see page 18881)	Writes Count bytes from Buffer to the socket connection.
	WriteBuffer (see page 2990)	Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.
	WriteComponent (see page 2990)	Initiates the writing of components and their properties to a stream.
	WriteComponentRes (see page 2994)	Writes components and their properties to a stream from a resource file format.
	WriteDescendent (see page 2994)	Streams components and their properties in inherited forms.
	WriteDescendentRes (see page 2995)	Streams components and their properties in inherited forms.
	WriteResourceHeader (see page 2995)	Writes a resource-file header to the stream.

Public Properties

	Name	Description
	Position (see page 2996)	Indicates the current offset into the stream for reading and writing.
	Size (see page 2997)	Indicates the size in bytes of the stream.
	TimeOut (see page 18882)	Specifies the time, in milliseconds, that the socket stream should wait before timing out on a read or write call.

129.1.13.2 TWinSocketStream.Create Constructor

Creates an instance of TWinSocketStream (see page 18876) for a Windows socket object.

Pascal

```
constructor Create(ASocket: TCustomWinSocket; TimeOut: Longint);
```

C++

```
__fastcall TWinSocketStream(TCustomWinSocket ASocket, Longint TimeOut);
```

Description

Call Create to obtain an instance of TWinSocketStream (see page 18876) for reading from or writing to the socket connection represented by the ASocket parameter. ASocket should be the endpoint of an open blocking socket connection to another Windows socket.

Create associates the stream with the socket connection, and allocates helper objects to allow it to determine when the socket connection is ready for reading or writing. The TimeOut (see page 18882) property (the number of milliseconds before a read or write operation aborts) is initialized to the value of the TimeOut (see page 18882) parameter.

If ASocket represents a non-blocking socket connection, Create raises an ESocketError (see page 18739) exception.

See Also

ESocketError

ClientType (see page 18769)

TCustomWinSocket (see page 18799)

ServerType (see page 18869)

TimeOut (see page 18882)

129.1.13.3 TWinSocketStream.Destroy Destructor

Destroys an instance of TWinSocketStream (see page 18876).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWinSocketStream();
```

Description

Do not call Destroy directly in an application. Instead call Free. Free verifies that the TWinSocketStream (see page 18876) reference is not nil and only then calls Destroy.

Free the TWinSocketStream (see page 18876) object when it is no longer needed for reading from or writing to the socket connection.

Destroy frees the helper objects allocated in the Create (see page 18878) method before calling the default destructor.

See Also

Create (see page 18878)

Free (see page 22300)

129.1.13.4 TWinSocketStream Methods

The methods of the TWinSocketStream class are listed here.

Public Methods

	Name	Description
	Read (see page 18880)	Reads up to Count bytes from the socket connection into Buffer.
	Seek (see page 18880)	Always returns 0.

	WaitForData (see page 18881)	Waits up to TimeOut (see page 18882) milliseconds for the socket connection to be ready to transfer data.
	Write (see page 18881)	Writes Count bytes from Buffer to the socket connection.

129.1.13.4.1 TWinSocketStream.Read Method

Reads up to Count bytes from the socket connection into Buffer.

Pascal

```
function Read(var Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Read( Buffer, Longint Count);
```

Description

Use Read to read data from the socket connection when the number of bytes to be transferred is unknown. Buffer must have at least Count bytes allocated to hold the data that is read from the connection. Read returns the number of bytes actually transferred (which may be less than the number requested in Count.)

Read may return 0 if the socket connection is extremely slow and the read operation has not completed after TimeOut (see page 18882) milliseconds. This ensures that the Read method does not hang indefinitely when a problem occurs with the socket connection.

To guard against the Read method timing out because of a slow connection, set Count fairly low, and make several calls to Read, rather than fewer calls with a large value of Count.

To ensure that the socket connection is ready to send data before calling Read, use the WaitForData (see page 18881) method.

Unlike the ReadBuffer method, Read does not raise an exception if Count bytes are not read from the socket connection.

See Also

ReadBuffer (see page 2983)

WaitForData (see page 18881)

Write (see page 18881)

TimeOut (see page 18882)

129.1.13.4.2 TWinSocketStream.Seek Method

Always returns 0.

Pascal

```
function Seek(Offset: Longint; Origin: Word): Longint; override;
```

C++

```
virtual __fastcall Longint Seek(Longint Offset, Word Origin);
```

Description

Seek overrides the inherited method to always set the Position property of the TWinSocketStream (see page 18876) object to 0. Socket connections do not support random access. The Position property of a Windows socket stream can only reflect the position at the beginning of all remaining data.

See Also

Position (see page 2996)

129.1.13.4.3 TWinSocketStream.WaitForData Method

Waits up to TimeOut (see page 18882) milliseconds for the socket connection to be ready to transfer data.

Pascal

```
function WaitForData(Timeout: Longint): Boolean;
```

C++

```
__fastcall Boolean WaitForData(Longint Timeout);
```

Description

Call WaitForData to ensure that the socket connection is ready to read or write information. WaitForData returns true if the socket connection is ready. WaitForData returns false if the socket connection is not ready after TimeOut (see page 18882) milliseconds elapse.

Call WaitForData before reading or writing information over the socket connection. Otherwise, Read (see page 18880) or Write (see page 18881) method calls may time out before any data is transferred.

See Also

Read (see page 18880)

Write (see page 18881)

129.1.13.4.4 TWinSocketStream.Write Method

Writes Count bytes from Buffer to the socket connection.

Pascal

```
function Write(const Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  Buffer, Longint Count);
```

Description

Use Write to write data from a Buffer to the socket connection. Write returns the number of bytes transferred.

Write may return 0 if the socket connection is extremely slow and the write operation has not completed after TimeOut (see page 18882) milliseconds. This ensures that the Write method does not hang indefinitely when a problem occurs with the socket connection.

To guard against the Write method timing out because of a slow connection, set Count fairly low, and make several calls to Write, rather than fewer calls with a large value of Count.

Unlike the WriteBuffer method, Write does not raise an exception if Count bytes are not written. For example, if the connection is too slow to transfer all the data, Write will return 0, where WriteBuffer will raise an exception.

See Also

Read (see page 18880)

WaitForData (see page 18881)

WriteBuffer (see page 2990)

TimeOut (see page 18882)

129.1.13.5 TWinSocketStream Properties

The properties of the TWinSocketStream class are listed here.

Public Properties

	Name	Description
	TimeOut (see page 18882)	Specifies the time, in milliseconds, that the socket stream should wait before timing out on a read or write call.

129.1.13.5.1 TWinSocketStream.TimeOut Property

Specifies the time, in milliseconds, that the socket stream should wait before timing out on a read or write call.

Pascal

```
property TimeOut: Longint;
```

C++

```
__property Longint TimeOut;
```

Description

Set TimeOut to the amount of time that should elapse before a read or write operation aborts under the assumption that the socket connection has failed.

See Also

Read (see page 18880)

Write (see page 18881)

129.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	SetErrorProc (see page 18883)	Replaces the exception handler for error messages that are received from a Windows socket connection.

129.2.1 ScktComp.SetErrorProc Function

Replaces the exception handler for error messages that are received from a Windows socket connection.

Pascal

```
function SetErrorProc(ErrorProc: TSocketErrorProc): TSocketErrorProc;
```

C++

```
TSocketErrorProc SetErrorProc(TSocketErrorProc ErrorProc);
```

File

ScktComp

Description

Call SetErrorProc to assign an exception handler for error messages from Windows socket API calls. By default, socket components have no special error handler and simply raise an ESocketError (see page 18739) exception when they receive error messages. If an exception handler is assigned using SetErrorProc, the socket component does not raise this exception.

The ErrorProc parameter specifies the exception handler. It is passed the Windows socket error code. Inside this exception handler an application can compensate for the error or raise an exception. Passing a value of nil (Delphi) or NULL (C++) for the ErrorProc parameter restores the default behavior, where the socket component raises an ESocketError (see page 18739) exception.

SetErrorProc returns the current exception handler. This value is nil (Delphi) or NULL (C++) when SetErrorProc is called for the first time, and is the value of the ErrorProc parameter from the previous invocation of SetErrorProc thereafter.

See Also

ESocketError (see page 18739)

129.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAsyncStyle (see page 18884)	This is record ScktComp.TAsyncStyle.
	TClientType (see page 18885)	This is record ScktComp.TClientType.
	TErrorEvent (see page 18886)	This is record ScktComp.TErrorEvent.
	TLookupState (see page 18887)	This is record ScktComp.TLookupState.
	TServerType (see page 18887)	This is record ScktComp.TServerType.
	TSocketEvent (see page 18887)	TSocketEvent identifies the operation that a socket is performing.

Records

	Name	Description
	TCMLookupComplete (see page 18885)	This is record ScktComp.TCMLookupComplete.
	TCMSocketMessage (see page 18886)	This is record ScktComp.TCMSocketMessage.

129.3.1 ScktComp.TAsyncStyle Enumeration

Pascal

```
TAsyncStyle = (
  asRead,
  asWrite,
  asOOB,
  asAccept,
  asConnect,
  asClose
);
```

C++

```
enum TAsyncStyle {
  asRead,
  asWrite,
  asOOB,
  asAccept,
  asConnect,
  asClose
};
```

File

ScktComp

Description

This is record ScktComp.TAsyncStyle.

129.3.2 ScktComp.TClientType Enumeration

Pascal

```
TClientType = (
  ctNonBlocking,
  ctBlocking
);
```

C++

```
enum TClientType {
  ctNonBlocking,
  ctBlocking
};
```

File

ScktComp

Description

This is record ScktComp.TClientType.

129.3.3 ScktComp.TCMLookupComplete Record

Pascal

```
TCMLookupComplete = record
  Msg: Cardinal;
  LookupHandle: THandle;
  AsyncBufLen: Word;
  AsyncError: Word;
  Result: Longint;
end;
```

C++

```
struct TCMLookupComplete {
  unsigned Msg;
  THandle LookupHandle;
  Word AsyncBufLen;
  Word AsyncError;
  Longint Result;
};
```

File

ScktComp

Description

This is record ScktComp.TCMLookupComplete.

129.3.4 ScktComp.TCMSocketMessage Record

Pascal

```
TCMSocketMessage = record
  Msg: Cardinal;
  Socket: TSocket;
  SelectEvent: Word;
  SelectError: Word;
  Result: Longint;
end;
```

C++

```
struct TCMSocketMessage {
  unsigned Msg;
  TSocket Socket;
  Word SelectEvent;
  Word SelectError;
  Longint Result;
};
```

File

ScktComp

Description

This is record ScktComp.TCMSocketMessage.

129.3.5 ScktComp.TErrorEvent Enumeration

Pascal

```
TErrorEvent = (
  eeGeneral,
  eeSend,
  eeReceive,
  eeConnect,
  eeDisconnect,
  eeAccept,
  eeLookup
);
```

C++

```
enum TErrorEvent {
  eeGeneral,
  eeSend,
  eeReceive,
  eeConnect,
  eeDisconnect,
  eeAccept,
  eeLookup
};
```

File

ScktComp

Description

This is record ScktComp.TErrorEvent.

129.3.6 ScktComp.TLookupState Enumeration

Pascal

```
TLookupState = (
  lsIdle,
  lsLookupAddress,
  lsLookupService
);
```

C++

```
enum TLookupState {
  lsIdle,
  lsLookupAddress,
  lsLookupService
};
```

File

ScktComp

Description

This is record ScktComp.TLookupState.

129.3.7 ScktComp.TServerType Enumeration

Pascal

```
TServerType = (
  stNonBlocking,
  stThreadBlocking
);
```

C++

```
enum TServerType {
  stNonBlocking,
  stThreadBlocking
};
```

File

ScktComp

Description

This is record ScktComp.TServerType.

129.3.8 ScktComp.TSocketEvent Enumeration

TSocketEvent identifies the operation that a socket is performing.

Pascal

```
TSocketEvent = (
  seLookup,
  seConnecting,
  seConnect,
  seDisconnect,
  seListen,
  seAccept,
  seWrite,
  seRead
);
```

C++

```
enum TSocketEvent {
  seLookup,
  seConnecting,
  seConnect,
  seDisconnect,
  seListen,
  seAccept,
  seWrite,
  seRead
};
```

File

ScktComp

Description

TSocketEvent is used to identify what a socket is doing when various methods or event handlers are called. It has one of the following values:

Value	Meaning
seLookup	(client sockets only) The Windows socket object is about to locate the server socket. Changing the properties of the client socket has no effect at this point, but changes to the Windows socket object specified by the Socket parameter will affect the attempted connection.
seConnecting	(client sockets only) The server socket has been located but the connection isn't completed. This is the first opportunity to obtain the actual server port and IP address used for the connection, which may differ from those of the listening socket contacted initially.
seConnect	The socket connection has just been established. This is the point when a socket starts reading or writing over the connection.
seListen	(server sockets only) The Windows socket object has been initialized and is about to open a listening connection. Changing the properties of the server socket has no effect at this point, but changes to the Windows socket object specified by the Socket parameter will the listening connection.
seAccept	(server sockets only) A client connection request has just been accepted. This is the first opportunity to obtain the port and IP address of the individual client connection.
seWrite	The socket is ready for information to be written.
seRead	The socket is receiving information from a socket on the other end of the connection.

129.4 Types

The following table lists types in this documentation.

Types

Name	Description
TAsyncStyles (see page 18889)	This is type ScktComp.TAsyncStyles.
TGetSocketEvent (see page 18889)	This is type ScktComp.TGetSocketEvent.
TGetThreadEvent (see page 18890)	This is type ScktComp.TGetThreadEvent.
TSocketErrorEvent (see page 18890)	TSocketErrorEvent is the type for event handlers that respond when a Windows socket reports an error.
TSocketErrorProc (see page 18891)	This is type ScktComp.TSocketErrorProc.
TSocketEventEvent (see page 18891)	This is type ScktComp.TSocketEventEvent.
TSocketNotifyEvent (see page 18891)	The type for event handlers which respond to events during the expected course of forming or using a socket connection.
TThreadNotifyEvent (see page 18892)	This is type ScktComp.TThreadNotifyEvent.

129.4.1 ScktComp.TAsyncStyles Type

Pascal

```
TAsyncStyles = set of TAsyncStyle;
```

C++

```
TAsyncStyle TAsyncStyles;
```

File

ScktComp

Description

This is type ScktComp.TAsyncStyles.

129.4.2 ScktComp.TGetSocketEvent Type

Pascal

```
TGetSocketEvent = procedure (Sender: TObject; Socket: TSocket; var ClientSocket:
TServerClientWinSocket) of object;
```

C++

```
(Sender: TObject; Socket: TSocket; var ClientSocket: TServerClientWinSocket) (
TGetSocketEvent)();
```

File

ScktComp

Description

This is type ScktComp.TGetSocketEvent.

129.4.3 ScktComp.TGetThreadEvent Type

Pascal

```
TGetThreadEvent = procedure (Sender: TObject; ClientSocket: TServerClientWinSocket; var
SocketThread: TServerClientThread) of object;
```

C++

```
(Sender: TObject; ClientSocket: TServerClientWinSocket; var SocketThread: TServerClientThread)
( TGetThreadEvent)();
```

File

ScktComp

Description

This is type ScktComp.TGetThreadEvent.

129.4.4 ScktComp.TSocketErrorEvent Type

TSocketErrorEvent is the type for event handlers that respond when a Windows socket reports an error.

Pascal

```
TSocketErrorEvent = procedure (Sender: TObject; Socket: TCustomWinSocket; ErrorEvent:
TErrorEvent; var ErrorCode: Integer) of object;
```

C++

```
(Sender: TObject; Socket: TCustomWinSocket; ErrorEvent: TErrorEvent; var ErrorCode: Integer) (
TSocketErrorEvent)();
```

File

ScktComp

Description

The Sender parameter is the socket component that encountered the error.

The Socket parameter is the TCustomWinSocket (see page 18799) object that receives the error notification.

The ErrorCode parameter is the error code returned by the Windows socket API call. Changing this to 0 within an error handler prevents an exception from being raised.

The ErrorEvent parameter indicates what Socket was attempting to do when the error occurred. It is of type TErrorEvent (see page 18886), which can have any of the following values:

Value	Meaning
eeGeneral	The socket received an error message that does not fit into any of the following categories.
eeSend	An error occurred when trying to write to the socket connection.
eeReceive	An error occurred when trying to read from the socket connection.
eeConnect	A connection request that was already accepted could not be completed.
eeDisconnect	An error occurred when trying to close a connection.

eeAccept	A problem occurred when trying to accept a client connection request.

129.4.5 SocktComp.TSocketErrorProc Type

Pascal

```
TSocketErrorProc = procedure (ErrorCode: Integer);
```

C++

```
(ErrorCode: Integer) ( TSocketErrorProc)();
```

File

SocktComp

Description

This is type SocktComp.TSocketErrorProc.

129.4.6 SocktComp.TSocketEventEvent Type

Pascal

```
TSocketEventEvent = procedure (Sender: TObject; Socket: TCustomWinSocket; SocketEvent:
TSocketEvent) of object;
```

C++

```
(Sender: TObject; Socket: TCustomWinSocket; SocketEvent: TSocketEvent) ( TSocketEventEvent)();
```

File

SocktComp

Description

This is type SocktComp.TSocketEventEvent.

129.4.7 SocktComp.TSocketNotifyEvent Type

The type for event handlers which respond to events during the expected course of forming or using a socket connection.

Pascal

```
TSocketNotifyEvent = procedure (Sender: TObject; Socket: TCustomWinSocket) of object;
```

C++

```
(Sender: TObject; Socket: TCustomWinSocket) ( TSocketNotifyEvent)();
```

File

SocktComp

Description

The Sender parameter is the socket component or Windows socket object to which the event occurred.

The Socket parameter is the TCustomWinSocket (see page 18799) object that represents the endpoint of the connection.

When the event handler belongs to a client Windows socket object, these two parameters are the same. For server Windows socket objects, the Sender parameter represents the listening connection while the Socket parameter represents a client connection.

See Also

TCustomSocket (see page 18789)

129.4.8 ScktComp.TThreadNotifyEvent Type

Pascal

```
TThreadNotifyEvent = procedure (Sender: TObject; Thread: TServerClientThread) of object;
```

C++

```
(Sender: TObject; Thread: TServerClientThread) ( TThreadNotifyEvent)();
```

File

ScktComp

Description

This is type ScktComp.TThreadNotifyEvent.

129.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CM_DEFERFREE (see page 18893)	This is constant ScktComp.CM_DEFERFREE.
CM_LOOKUPCOMPLETE (see page 18893)	This is constant ScktComp.CM_LOOKUPCOMPLETE.
CM_SOCKETMESSAGE (see page 18893)	This is constant ScktComp.CM_SOCKETMESSAGE.

129.5.1 ScktComp.CM_DEFERFREE Constant

Pascal

```
CM_DEFERFREE = WM_USER + $0002;
```

C++

```
CM_DEFERFREE = WM_USER + $0002;
```

File

ScktComp

Description

This is constant ScktComp.CM_DEFERFREE.

129.5.2 ScktComp.CM_LOOKUPCOMPLETE Constant

Pascal

```
CM_LOOKUPCOMPLETE = WM_USER + $0003;
```

C++

```
CM_LOOKUPCOMPLETE = WM_USER + $0003;
```

File

ScktComp

Description

This is constant ScktComp.CM_LOOKUPCOMPLETE.

129.5.3 ScktComp.CM_SOCKETMESSAGE Constant

Pascal

```
CM_SOCKETMESSAGE = WM_USER + $0001;
```

C++

```
CM_SOCKETMESSAGE = WM_USER + $0001;
```

File

ScktComp

Description

This is constant ScktComp.CM_SOCKETMESSAGE.

130 SConnect Namespace

This is namespace SConnect.

Topics

Name	Description
Classes (see page 18897)	The following table lists classes in this documentation.
Interfaces (see page 18983)	The following table lists interfaces in this documentation.
Functions (see page 18999)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 19000)	The following table lists structs, records, enums in this documentation.
Types (see page 19001)	The following table lists types in this documentation.
Variables (see page 19003)	The following table lists variables in this documentation.
Constants (see page 19005)	The following table lists constants in this documentation.

Classes

	Name	Description
	EInterpreterError (see page 18897)	EInterpreterError is the exception class for errors that arise when trying to interpret COM calls that have been marshaled to a data block.
	ESocketConnectionError (see page 18898)	ESocketConnectionError is the exception class for errors that arise when sending or receiving messages using TSocketConnection (see page 18928).
	TCustomDataBlockInterpreter (see page 18898)	TCustomDataBlockInterpreter marshals COM interface calls for streamed connection components.
	TDataBlock (see page 18903)	This is class SConnect.TDataBlock.
	TDataBlockInterpreter (see page 18913)	TDataBlockInterpreter marshals COM interface calls for non COM-based connection components.
	TDataDispatch (see page 18921)	This is class SConnect.TDataDispatch.
	TPacketInterceptFactory (see page 18925)	TPacketInterceptFactory is the type of the class factory used to instantiate a data packet interceptor.
	TSocketConnection (see page 18928)	TSocketConnection uses Windows sockets to manage the connection to an application server.
	TSocketTransport (see page 18939)	This is class SConnect.TSocketTransport.
	TStreamedConnection (see page 18945)	TStreamedConnection is the base class for components that handle their own marshaling of interface calls when connecting to application servers.
	TTransportThread (see page 18961)	This is class SConnect.TTransportThread.
	TWebConnection (see page 18968)	TWebConnection uses HTTP to manage the connection to an application server.

Constants

Name	Description
asCreateObject (see page 19005)	This is constant SConnect.asCreateObject.

asError (see page 19005)	This is constant SConnect.asError.
asFreeObject (see page 19006)	This is constant SConnect.asFreeObject.
asGetAppServers (see page 19006)	This is constant SConnect.asGetAppServers.
asGetGUID (see page 19006)	This is constant SConnect.asGetGUID.
asGetID (see page 19007)	This is constant SConnect.asGetID.
asGetServers (see page 19007)	This is constant SConnect.asGetServers.
asInvoke (see page 19007)	This is constant SConnect.asInvoke.
asMask (see page 19008)	This is constant SConnect.asMask.
asSoapCommand (see page 19008)	This is constant SConnect.asSoapCommand.
CallSig (see page 19008)	This is constant SConnect.CallSig.
ResultSig (see page 19009)	This is constant SConnect.ResultSig.
THREAD_EXCEPTION (see page 19009)	This is constant SConnect.THREAD_EXCEPTION.
THREAD_RECEIVEDSTREAM (see page 19009)	This is constant SConnect.THREAD_RECEIVEDSTREAM.
THREAD_REPLACETRANSPORT (see page 19010)	This is constant SConnect.THREAD_REPLACETRANSPORT.
THREAD_SENDNOTIFY (see page 19010)	This is constant SConnect.THREAD_SENDNOTIFY.
THREAD_SENDSTREAM (see page 19010)	This is constant SConnect.THREAD_SENDSTREAM.

Functions

	Name	Description
	GetPacketInterceptorList (see page 18999)	This is function SConnect.GetPacketInterceptorList.
	LoadWinSock2 (see page 18999)	This is function SConnect.LoadWinSock2.

Interfaces

	Name	Description
	IDataBlock (see page 18983)	IDataBlock is the interface used by data block interpreters to access memory buffers.
	IDataIntercept (see page 18991)	IDataIntercept is the interface used by TSocketConnection (see page 18928) components to transform data that is communicated over the connection.
	ISendDataBlock (see page 18993)	ISendDataBlock is the interface used by data block interpreters to send messages to an application server after they are marshaled.
	ITransport (see page 18995)	ITransport is the interface used by TStreamedConnection (see page 18945) to connect to, read from, and write to a remote application server.

Structs, Records, Enums

	Name	Description
	TVarFlag (see page 19000)	This is record SConnect.TVarFlag.

Types

Name	Description
PIntArray (see page 19001)	This is type SConnect.PIntArray.
PVariantArray (see page 19001)	This is type SConnect.PVariantArray.
TIntArray (see page 19002)	This is type SConnect.TIntArray.
TVarFlags (see page 19002)	This is type SConnect.TVarFlags.
TVariantArray (see page 19002)	This is type SConnect.TVariantArray.

Variables

Name	Description
WSACloseEvent (see page 19003)	This is variable SConnect.WSACloseEvent.
WSACreateEvent (see page 19003)	This is variable SConnect.WSACreateEvent.
WSAEventSelect (see page 19004)	This is variable SConnect.WSAEventSelect.
WSAResetEvent (see page 19004)	This is variable SConnect.WSAResetEvent.

130.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EInterpreterError (see page 18897)	EInterpreterError is the exception class for errors that arise when trying to interpret COM calls that have been marshaled to a data block.
	ESocketConnectionError (see page 18898)	ESocketConnectionError is the exception class for errors that arise when sending or receiving messages using TSocketConnection (see page 18928).
	TCustomDataBlockInterpreter (see page 18898)	TCustomDataBlockInterpreter marshals COM interface calls for streamed connection components.
	TDataBlock (see page 18903)	This is class SConnect.TDataBlock.
	TDataBlockInterpreter (see page 18913)	TDataBlockInterpreter marshals COM interface calls for non COM-based connection components.
	TDataDispatch (see page 18921)	This is class SConnect.TDataDispatch.
	TPacketInterceptFactory (see page 18925)	TPacketInterceptFactory is the type of the class factory used to instantiate a data packet interceptor.
	TSocketConnection (see page 18928)	TSocketConnection uses Windows sockets to manage the connection to an application server.
	TSocketTransport (see page 18939)	This is class SConnect.TSocketTransport.
	TStreamedConnection (see page 18945)	TStreamedConnection is the base class for components that handle their own marshaling of interface calls when connecting to application servers.
	TTransportThread (see page 18961)	This is class SConnect.TTransportThread.
	TWebConnection (see page 18968)	TWebConnection uses HTTP to manage the connection to an application server.

130.1.1 EInterpreterError Class

EInterpreterError is the exception class for errors that arise when trying to interpret COM calls that have been marshaled to a data block.

Class Hierarchy

Pascal

```
EInterpreterError = class(Exception);
```

C++

```
class EInterpreterError : public Exception;
```

File

SConnect

Description

Data block interpreters (TCustomDataBlockInterpreter (see page 18898) descendants) raise EInterpreterError when they can't interpret the information in a data block. Streamed connection components such as TSocketConnection (see page 18928) use data block interpreters to send interface calls to and receive interface calls from a remote application server.

See Also

TCustomDataBlockInterpreter (see page 18898)

TSocketConnection (see page 18928)

TSOAPConnection

TWebConnection (see page 18968)

130.1.2 ESocketConnectionError Class

ESocketConnectionError is the exception class for errors that arise when sending or receiving messages using TSocketConnection (see page 18928).

Class Hierarchy

Pascal

```
ESocketConnectionError = class(Exception);
```

C++

```
class ESocketConnectionError : public Exception;
```

File

SConnect

Description

Socket connection components raise ESocketConnectionError when the socket they use to send and receive messages reports an error, or when the information read from the socket is incomplete.

See Also

TSocketConnection (see page 18928)

130.1.3 TCustomDataBlockInterpreter Class

TCustomDataBlockInterpreter marshals COM interface calls for streamed connection components.

Class Hierarchy

Pascal

```
TCustomDataBlockInterpreter = class;
```

C++

```
class TCustomDataBlockInterpreter;
```

File

SConnect

Description

TCustomDataBlockInterpreter introduces the methods needed for marshaling the messages that pass between a client application and an application server in a multi-tiered database application.

Streamed connection components use data block interpreters to marshal of messages sent to and received from an application server. An object that supports the IDataBlock (see page 18983) interface stores the messages themselves.

See Also

TStreamedConnection (see page 18945)

IDataBlock (see page 18983)

130.1.3.1 TCustomDataBlockInterpreter Members

The following tables list the members exposed by TCustomDataBlockInterpreter.

Protected Methods

	Name	Description
	AddDispatch (see page 18900)	This is AddDispatch, a member of class TCustomDataBlockInterpreter.
	CallCreateObject (see page 18900)	This is CallCreateObject, a member of class TCustomDataBlockInterpreter.
	CallFreeObject (see page 18900)	This is CallFreeObject, a member of class TCustomDataBlockInterpreter.
	CallGetIDsOfNames (see page 18900)	This is CallGetIDsOfNames, a member of class TCustomDataBlockInterpreter.
	CallGetServerList (see page 18901)	This is CallGetServerList, a member of class TCustomDataBlockInterpreter.
	CallInvoke (see page 18901)	This is CallInvoke, a member of class TCustomDataBlockInterpreter.
	CanCreateObject (see page 18901)	This is CanCreateObject, a member of class TCustomDataBlockInterpreter.
	CreateObject (see page 18901)	This is CreateObject, a member of class TCustomDataBlockInterpreter.
	InternalCreateObject (see page 18901)	This is InternalCreateObject, a member of class TCustomDataBlockInterpreter.
	LockObject (see page 18902)	This is LockObject, a member of class TCustomDataBlockInterpreter.
	ReleaseObject (see page 18902)	This is ReleaseObject, a member of class TCustomDataBlockInterpreter.
	RemoveDispatch (see page 18902)	This is RemoveDispatch, a member of class TCustomDataBlockInterpreter.
	StoreObject (see page 18903)	This is StoreObject, a member of class TCustomDataBlockInterpreter.
	UnlockObject (see page 18903)	This is UnlockObject, a member of class TCustomDataBlockInterpreter.

Public Methods

	Name	Description
	InterpretData (see page 18902)	This is InterpretData, a member of class TCustomDataBlockInterpreter.

130.1.3.2 TCustomDataBlockInterpreter Methods

The methods of the TCustomDataBlockInterpreter class are listed here.

Protected Methods

	Name	Description
	AddDispatch (see page 18900)	This is AddDispatch, a member of class TCustomDataBlockInterpreter.
	CallCreateObject (see page 18900)	This is CallCreateObject, a member of class TCustomDataBlockInterpreter.
	CallFreeObject (see page 18900)	This is CallFreeObject, a member of class TCustomDataBlockInterpreter.
	CallGetIDsOfNames (see page 18900)	This is CallGetIDsOfNames, a member of class TCustomDataBlockInterpreter.
	CallGetServerList (see page 18901)	This is CallGetServerList, a member of class TCustomDataBlockInterpreter.
	CallInvoke (see page 18901)	This is CallInvoke, a member of class TCustomDataBlockInterpreter.
	CanCreateObject (see page 18901)	This is CanCreateObject, a member of class TCustomDataBlockInterpreter.
	CreateObject (see page 18901)	This is CreateObject, a member of class TCustomDataBlockInterpreter.
	InternalCreateObject (see page 18901)	This is InternalCreateObject, a member of class TCustomDataBlockInterpreter.
	LockObject (see page 18902)	This is LockObject, a member of class TCustomDataBlockInterpreter.
	ReleaseObject (see page 18902)	This is ReleaseObject, a member of class TCustomDataBlockInterpreter.
	RemoveDispatch (see page 18902)	This is RemoveDispatch, a member of class TCustomDataBlockInterpreter.
	StoreObject (see page 18903)	This is StoreObject, a member of class TCustomDataBlockInterpreter.

	UnlockObject (see page 18903)	This is UnlockObject, a member of class TCustomDataBlockInterpreter.

Public Methods

	Name	Description
	InterpretData (see page 18902)	This is InterpretData, a member of class TCustomDataBlockInterpreter.

130.1.3.2.1 TCustomDataBlockInterpreter.AddDispatch Method

Pascal

```
procedure AddDispatch(Value: TDataDispatch); virtual; abstract;
```

C++

```
virtual __fastcall AddDispatch(TDataDispatch Value) = 0;
```

Description

This is AddDispatch, a member of class TCustomDataBlockInterpreter.

130.1.3.2.2 TCustomDataBlockInterpreter.CallCreateObject Method

Pascal

```
function CallCreateObject(Name: string): OleVariant; virtual; abstract;
```

C++

```
virtual __fastcall OleVariant CallCreateObject(AnsiString Name) = 0;
```

Description

This is CallCreateObject, a member of class TCustomDataBlockInterpreter.

130.1.3.2.3 TCustomDataBlockInterpreter.CallFreeObject Method

Pascal

```
procedure CallFreeObject(DispatchIndex: Integer); virtual; abstract;
```

C++

```
virtual __fastcall CallFreeObject(int DispatchIndex) = 0;
```

Description

This is CallFreeObject, a member of class TCustomDataBlockInterpreter.

130.1.3.2.4 TCustomDataBlockInterpreter.CallGetIDsOfNames Method

Pascal

```
function CallGetIDsOfNames(DispatchIndex: Integer; const IID: TGUID; Names: Pointer;
NameCount: Integer; LocaleID: Integer; DispIDs: Pointer): HResult; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT CallGetIDsOfNames(int DispatchIndex, const TGUID IID,
void * Names, int NameCount, int LocaleID, void * DispIDs) = 0;
```

Description

This is CallGetIDsOfNames, a member of class TCustomDataBlockInterpreter.

130.1.3.2.5 TCustomDataBlockInterpreter.CallGetServerList Method

Pascal

```
function CallGetServerList: OleVariant; virtual; abstract;
```

C++

```
virtual __fastcall OleVariant CallGetServerList() = 0;
```

Description

This is CallGetServerList, a member of class TCustomDataBlockInterpreter.

130.1.3.2.6 TCustomDataBlockInterpreter.CallInvoke Method

Pascal

```
function CallInvoke(DispatchIndex: Integer; DispID: Integer; const IID: TGUID; LocaleID:
Integer; Flags: Word; var Params; VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer):
HResult; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT CallInvoke(int DispatchIndex, int DispID, const TGUID
IID, int LocaleID, Word Flags,  Params, void * VarResult, void * ExcepInfo, void * ArgErr) = 0;
```

Description

This is CallInvoke, a member of class TCustomDataBlockInterpreter.

130.1.3.2.7 TCustomDataBlockInterpreter.CanCreateObject Method

Pascal

```
function CanCreateObject(const ClassID: TGUID): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean CanCreateObject(const TGUID ClassID) = 0;
```

Description

This is CanCreateObject, a member of class TCustomDataBlockInterpreter.

130.1.3.2.8 TCustomDataBlockInterpreter.CreateObject Method

Pascal

```
function CreateObject(const Name: string): OleVariant; virtual; abstract;
```

C++

```
virtual __fastcall OleVariant CreateObject(const AnsiString Name) = 0;
```

Description

This is CreateObject, a member of class TCustomDataBlockInterpreter.

130.1.3.2.9 TCustomDataBlockInterpreter.InternalCreateObject Method

Pascal

```
function InternalCreateObject(const ClassID: TGUID): OleVariant; virtual; abstract;
```

C++

```
virtual __fastcall OleVariant InternalCreateObject(const TGUID ClassID) = 0;
```

Description

This is InternalCreateObject, a member of class TCustomDataBlockInterpreter.

130.1.3.2.10 TCustomDataBlockInterpreter.InterpretData Method

Pascal

```
procedure InterpretData(const Data: IDataBlock); virtual; abstract;
```

C++

```
virtual __fastcall InterpretData(const IDataBlock Data) = 0;
```

Description

This is InterpretData, a member of class TCustomDataBlockInterpreter.

130.1.3.2.11 TCustomDataBlockInterpreter.LockObject Method

Pascal

```
function LockObject(ID: Integer): IDispatch; virtual; abstract;
```

C++

```
virtual __fastcall IDispatch LockObject(int ID) = 0;
```

Description

This is LockObject, a member of class TCustomDataBlockInterpreter.

130.1.3.2.12 TCustomDataBlockInterpreter.ReleaseObject Method

Pascal

```
procedure ReleaseObject(ID: Integer); virtual; abstract;
```

C++

```
virtual __fastcall ReleaseObject(int ID) = 0;
```

Description

This is ReleaseObject, a member of class TCustomDataBlockInterpreter.

130.1.3.2.13 TCustomDataBlockInterpreter.RemoveDispatch Method

Pascal

```
procedure RemoveDispatch(Value: TDataDispatch); virtual; abstract;
```

C++

```
virtual __fastcall RemoveDispatch(TDataDispatch Value) = 0;
```

Description

This is RemoveDispatch, a member of class TCustomDataBlockInterpreter.

130.1.3.2.14 TCustomDataBlockInterpreter.StoreObject Method

Pascal

```
function StoreObject(const Value: OleVariant): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int StoreObject(const OleVariant Value) = 0;
```

Description

This is StoreObject, a member of class TCustomDataBlockInterpreter.

130.1.3.2.15 TCustomDataBlockInterpreter.UnlockObject Method

Pascal

```
procedure UnlockObject(ID: Integer; const Disp: IDispatch); virtual; abstract;
```

C++

```
virtual __fastcall UnlockObject(int ID, const IDispatch Disp) = 0;
```

Description

This is UnlockObject, a member of class TCustomDataBlockInterpreter.

130.1.4 TDataBlock Class

Class Hierarchy

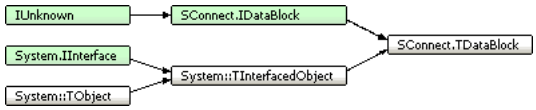

Pascal

```
TDataBlock = class(TInterfacedObject, IDataBlock);
```

C++

```
class TDataBlock : public TInterfacedObject, public IDataBlock;
```

File

SConnect

Description

This is class SConnect.TDataBlock.

130.1.4.1 TDataBlock Members

The following tables list the members exposed by TDataBlock.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	Clear (see page 18906)	Removes all data from Memory except for the reserved bytes.
	GetBytesReserved (see page 18907)	Returns the value of the BytesReserved property.
	GetMemory (see page 18907)	Returns the value of the Memory property.
	GetSignature (see page 18907)	Returns the value of the Signature property.
	GetSize (see page 18907)	Returns the value of the Size property.
	GetStream (see page 18908)	Returns the value of the Stream property.
	IgnoreStream (see page 18908)	Informs the data block when its stream is invalidated.
	InitData (see page 18908)	Adds data to the data block.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	Read (see page 18909)	Reads a specified number of bytes into a buffer.
	SetSignature (see page 18909)	Sets the value of the Signature property.
	SetSize (see page 18910)	Sets the value of the Size property.
	Write (see page 18910)	Writes a specified number of bytes from a buffer.

Protected Properties

	Name	Description
R	BytesReserved (see page 18911)	Indicates the number of bytes from Memory that are reserved for internal use.
R	Memory (see page 18911)	Points to a buffer from which data is read or to which data is written.
	Signature (see page 18911)	Indicates the contents of the data block.
	Size (see page 18912)	Indicates the size of the data in the memory buffer.
R	Stream (see page 18912)	Provides access to an internal stream that contains the data block's memory.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 18905)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 18905)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

130.1.4.2 TDataBlock.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TDataBlock();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

130.1.4.3 TDataBlock.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataBlock();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

130.1.4.4 TDataBlock Methods

The methods of the TDataBlock class are listed here.

Protected Methods

	Name	Description
	Clear (see page 18906)	Removes all data from Memory except for the reserved bytes.
	GetBytesReserved (see page 18907)	Returns the value of the BytesReserved property.
	GetMemory (see page 18907)	Returns the value of the Memory property.
	GetSignature (see page 18907)	Returns the value of the Signature property.
	GetSize (see page 18907)	Returns the value of the Size property.
	GetStream (see page 18908)	Returns the value of the Stream property.
	IgnoreStream (see page 18908)	Informs the data block when its stream is invalidated.
	InitData (see page 18908)	Adds data to the data block.
	Read (see page 18909)	Reads a specified number of bytes into a buffer.
	SetSignature (see page 18909)	Sets the value of the Signature property.
	SetSize (see page 18910)	Sets the value of the Size property.
	Write (see page 18910)	Writes a specified number of bytes from a buffer.

130.1.4.4.1 TDataBlock.Clear Method

Removes all data from Memory except for the reserved bytes.

Pascal

```
procedure Clear; stdcall;
```

C++

```
__fastcall __stdcall Clear();
```

Description

Call Clear when finished reading from Memory or when the information written to Memory has been used. Clear reclaims the resources used to store information that is read from or written to the data block.

See Also

BytesReserved (see page 18989)

Memory (see page 18989)

130.1.4.4.2 TDataBlock.GetBytesReserved Method

Returns the value of the BytesReserved property.

Pascal

```
function GetBytesReserved: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetBytesReserved();
```

Description

GetBytesReserved is the read implementer for the BytesReserved property.

See Also

BytesReserved (see page 18989)

130.1.4.4.3 TDataBlock.GetMemory Method

Returns the value of the Memory property.

Pascal

```
function GetMemory: Pointer; stdcall;
```

C++

```
__fastcall __stdcall void * GetMemory();
```

Description

GetMemory is the read implementer for the Memory property.

See Also

Memory (see page 18989)

130.1.4.4.4 TDataBlock.GetSignature Method

Returns the value of the Signature property.

Pascal

```
function GetSignature: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetSignature();
```

Description

GetSignature is the read implementer for the Signature property.

See Also

Signature (see page 18990)

130.1.4.4.5 TDataBlock.GetSize Method

Returns the value of the Size property.

Pascal

```
function GetSize: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetSize();
```

Description

GetSize is the read implementer for the Size property.

See Also

Size (see page 18990)

130.1.4.4.6 TDataBlock.GetStream Method

Returns the value of the Stream property.

Pascal

```
function GetStream: TStream; stdcall;
```

C++

```
__fastcall __stdcall TStream GetStream();
```

Description

GetStream is the read implementer for the Stream property.

See Also

Stream (see page 18991)

130.1.4.4.7 TDataBlock.IgnoreStream Method

Informs the data block when its stream is invalidated.

Pascal

```
procedure IgnoreStream; stdcall;
```

C++

```
__fastcall __stdcall IgnoreStream();
```

Description

Call IgnoreStream after taking responsibility for freeing the component specified by Stream. After a call to IgnoreStream, the data block will not free the memory for its internal Stream component.

See Also

Stream (see page 18991)

130.1.4.4.8 TDataBlock.InitData Method

Adds data to the data block.

Pascal

```
function InitData(Data: Pointer; DataLen: Integer; CheckLen: Boolean): Integer; stdcall;
```

C++

```
__fastcall __stdcall int InitData(void * Data, int DataLen, Boolean CheckLen);
```

Description

Data is the data to be added to the data block. It must include a valid signature or InitData raises an exception.

DataLen is the number of bytes referenced by Data.

CheckLen specifies whether InitData should verify that the signature indicates the same amount of data as is indicated by the DataLen parameter.

See Also

Signature (see page 18990)

130.1.4.4.9 TDataBlock.Read Method

Reads a specified number of bytes into a buffer.

Pascal

```
function Read(var Buffer; Count: Integer): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Read( Buffer, int Count);
```

Description

Call Read to copy the contents stored in the memory buffer to the buffer supplied by the Buffer parameter. Read reads up to Count bytes and returns the number of bytes actually read.

The first call to Read begins reading from Memory at the first position after the reserved bytes. Subsequent calls begin reading at the point where the last call to Read left off.

Note: The Count parameter should specify a value less than or equal to Size - BytesReserved.

See Also

Write (see page 18988)

Memory (see page 18989)

Size (see page 18990)

BytesReserved (see page 18989)

130.1.4.4.10 TDataBlock.SetSignature Method

Sets the value of the Signature property.

Pascal

```
procedure SetSignature(Value: Integer); stdcall;
```

C++

```
__fastcall __stdcall SetSignature(int Value);
```

Description

SetSignature is the write implementer for the Signature property.

See Also

Signature (see page 18990)

130.1.4.4.11 TDataBlock.SetSize Method

Sets the value of the Size property.

Pascal

```
procedure SetSize(Value: Integer); stdcall;
```

C++

```
__fastcall __stdcall SetSize(int Value);
```

Description

SetSize is the write implementer for the Size property.

See Also

Size (see page 18990)

130.1.4.4.12 TDataBlock.Write Method

Writes a specified number of bytes from a buffer.

Pascal

```
function Write(const Buffer; Count: Integer): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Write(const  Buffer, int Count);
```

Description

Call Write to copy the contents of the buffer supplied as the Buffer parameter to the next position in Memory. Write writes up to Count bytes and returns the number of bytes actually written.

The first call to Write begins writing to Memory at the first position after the reserved bytes. Subsequent calls begin writing at the point where the last call to Write left off.

Note: The Count parameter should specify a value less than or equal to Size - BytesReserved.

See Also

Read (see page 18987)

Memory (see page 18989)

Size (see page 18990)

BytesReserved (see page 18989)

130.1.4.5 TDataBlock Properties

The properties of the TDataBlock class are listed here.

Protected Properties

	Name	Description
R	BytesReserved (see page 18911)	Indicates the number of bytes from Memory that are reserved for internal use.

R	Memory (see page 18911)	Points to a buffer from which data is read or to which data is written.
	Signature (see page 18911)	Indicates the contents of the data block.
	Size (see page 18912)	Indicates the size of the data in the memory buffer.
R	Stream (see page 18912)	Provides access to an internal stream that contains the data block's memory.

130.1.4.5.1 TDataBlock.BytesReserved Property

Indicates the number of bytes from Memory that are reserved for internal use.

Pascal

```
property BytesReserved: Integer;
```

C++

```
__property int BytesReserved;
```

Description

BytesReserved indicates how many bytes of Memory can't be used for reading and writing. When interpreting the value of the Size property, use BytesReserved to determine how much buffer space is available for reading and writing data.

See Also

Memory (see page 18989)

Size (see page 18990)

130.1.4.5.2 TDataBlock.Memory Property

Points to a buffer from which data is read or to which data is written.

Pascal

```
property Memory: Pointer;
```

C++

```
__property void * Memory;
```

Description

Memory provides access to the buffer that contains data that is read or written. Memory includes a reserved area (of size BytesReserved) for internal information such as the value of Signature,

Note: When implementing the Receive and Send methods of the ITransport interface, component writers should read directly from and write directly to Memory, rather than using the Read and Write methods. Otherwise, the information in the reserved area will not be sent to or from the application server.

See Also

Size (see page 18990)

ITransport (see page 18995)

BytesReserved (see page 18989)

Signature (see page 18990)

Stream (see page 18991)

130.1.4.5.3 TDataBlock.Signature Property

Indicates the contents of the data block.

Pascal

```
property Signature: Integer;
```

C++

```
__property int Signature;
```

Description

Write Signature values to indicate what type of data is written to the data block. Read Signature to determine what type of data can be read from the data block.

The value of signature is stored somewhere in the first BytesReserved bytes of Memory.

See Also

BytesReserved (see page 18989)

Memory (see page 18989)

130.1.4.5.4 TDataBlock.Size Property

Indicates the size of the data in the memory buffer.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

Size is the number of bytes available in Memory, excluding the reserved bytes. Set Size to ensure that Memory is large enough to contain the data that the data block will read or write.

See Also

BytesReserved (see page 18989)

Memory (see page 18989)

Signature (see page 18990)

130.1.4.5.5 TDataBlock.Stream Property

Provides access to an internal stream that contains the data block's memory.

Pascal

```
property Stream: TStream;
```

C++

```
__property TStream Stream;
```

Description

Stream allows component writers more convenient access to the data block's memory buffer. This memory buffer contains the data that is read or written.

See Also

Memory (see page 18989)

IgnoreStream (see page 18986)

TMemoryStream (see page 2902)

130.1.5 TDataBlockInterpreter Class

TDataBlockInterpreter marshals COM interface calls for non COM-based connection components.

Class Hierarchy

Pascal

```
TDataBlockInterpreter = class(TCustomDataBlockInterpreter);
```

C++

```
class TDataBlockInterpreter : public TCustomDataBlockInterpreter;
```

File

SConnect

Description

Many streamed connection components use TDataBlockInterpreter to handle the marshaling of messages sent to and received from an application server. An object that supports the IDataBlock (see page 18983) interface stores the messages themselves.

See Also

TStreamedConnection (see page 18945)

IDataBlock (see page 18983)

TSocketConnection (see page 18928)

TWebConnection (see page 18968)

130.1.5.1 TDataBlockInterpreter Members

The following tables list the members exposed by TDataBlockInterpreter.

Protected Methods

	Name	Description
	AddDispatch (see page 18915)	This is AddDispatch, a member of class TDataBlockInterpreter.
	CallFreeObject (see page 18916)	This is CallFreeObject, a member of class TDataBlockInterpreter.
	CallGetIDsOfNames (see page 18916)	This is CallGetIDsOfNames, a member of class TDataBlockInterpreter.
	CallGetServerList (see page 18916)	This is CallGetServerList, a member of class TDataBlockInterpreter.
	CallInvoke (see page 18917)	This is CallInvoke, a member of class TDataBlockInterpreter.
	CanCreateObject (see page 18917)	This is CanCreateObject, a member of class TDataBlockInterpreter.
	CreateObject (see page 18917)	This is CreateObject, a member of class TDataBlockInterpreter.
	DoCreateObject (see page 18917)	This is DoCreateObject, a member of class TDataBlockInterpreter.
	DoCustomAction (see page 18917)	This is DoCustomAction, a member of class TDataBlockInterpreter.
	DoFreeObject (see page 18918)	This is DoFreeObject, a member of class TDataBlockInterpreter.
	DoGetAppServerList (see page 18918)	This is DoGetAppServerList, a member of class TDataBlockInterpreter.
	DoGetIDsOfNames (see page 18918)	This is DoGetIDsOfNames, a member of class TDataBlockInterpreter.
	DoGetServerList (see page 18918)	This is DoGetServerList, a member of class TDataBlockInterpreter.
	DoInvoke (see page 18919)	This is DoInvoke, a member of class TDataBlockInterpreter.

	InternalCreateObject (see page 18919)	This is InternalCreateObject, a member of class TDataBlockInterpreter.
	LockObject (see page 18919)	This is LockObject, a member of class TDataBlockInterpreter.
	ReleaseObject (see page 18920)	This is ReleaseObject, a member of class TDataBlockInterpreter.
	RemoveDispatch (see page 18920)	This is RemoveDispatch, a member of class TDataBlockInterpreter.
	StoreObject (see page 18920)	This is StoreObject, a member of class TDataBlockInterpreter.
	UnlockObject (see page 18920)	This is UnlockObject, a member of class TDataBlockInterpreter.

Public Methods

	Name	Description
	CallCreateObject (see page 18915)	Creates a COM object on a remote application server given its class name.
	Create (see page 18914)	Creates and initializes an instance of TDataBlockInterpreter (see page 18913).
	Destroy (see page 18914)	Destroys an instance of TDataBlockInterpreter (see page 18913).
	InterpretData (see page 18919)	Interprets a call received from a remote application server.

130.1.5.2 TDataBlockInterpreter.Create Constructor

Creates and initializes an instance of TDataBlockInterpreter (see page 18913).

Pascal

```
constructor Create(SendDataBlock: ISendDataBlock; CheckRegValue: string);
```

C++

```
__fastcall TDataBlockInterpreter(ISendDataBlock SendDataBlock, AnsiString CheckRegValue);
```

Description

Applications should not need to instantiate data block interpreters. Typically, they are instantiated by the streamed connection components that use them to marshal data. To access a streamed connection component's data block interpreter instance, use its Interpreter property.

Create takes two arguments:

An ISendDataBlock (see page 18993) interface that the data block interpreter can use to send information in a data block to its destination object (usually the remote data module of an application server).

A string that specifies a registry key value which must appear for application servers that can be accessed using the data block interpreter.

See Also

Interpreter (see page 18960)

130.1.5.3 TDataBlockInterpreter.Destroy Destructor

Destroys an instance of TDataBlockInterpreter (see page 18913).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataBlockInterpreter();
```

Description

Applications have no need to free data block interpreters. They are freed by the streamed connection component that uses them to marshal interface calls.

See Also

Free (see page 22300)

130.1.5.4 TDataBlockInterpreter Methods

The methods of the TDataBlockInterpreter class are listed here.

Protected Methods

	Name	Description
	AddDispatch (see page 18915)	This is AddDispatch, a member of class TDataBlockInterpreter.
	CallFreeObject (see page 18916)	This is CallFreeObject, a member of class TDataBlockInterpreter.
	CallGetIDsOfNames (see page 18916)	This is CallGetIDsOfNames, a member of class TDataBlockInterpreter.
	CallGetServerList (see page 18916)	This is CallGetServerList, a member of class TDataBlockInterpreter.
	CallInvoke (see page 18917)	This is CallInvoke, a member of class TDataBlockInterpreter.
	CanCreateObject (see page 18917)	This is CanCreateObject, a member of class TDataBlockInterpreter.
	CreateObject (see page 18917)	This is CreateObject, a member of class TDataBlockInterpreter.
	DoCreateObject (see page 18917)	This is DoCreateObject, a member of class TDataBlockInterpreter.
	DoCustomAction (see page 18917)	This is DoCustomAction, a member of class TDataBlockInterpreter.
	DoFreeObject (see page 18918)	This is DoFreeObject, a member of class TDataBlockInterpreter.
	DoGetAppServerList (see page 18918)	This is DoGetAppServerList, a member of class TDataBlockInterpreter.
	DoGetIDsOfNames (see page 18918)	This is DoGetIDsOfNames, a member of class TDataBlockInterpreter.
	DoGetServerList (see page 18918)	This is DoGetServerList, a member of class TDataBlockInterpreter.
	DoInvoke (see page 18919)	This is DoInvoke, a member of class TDataBlockInterpreter.
	InternalCreateObject (see page 18919)	This is InternalCreateObject, a member of class TDataBlockInterpreter.
	LockObject (see page 18919)	This is LockObject, a member of class TDataBlockInterpreter.
	ReleaseObject (see page 18920)	This is ReleaseObject, a member of class TDataBlockInterpreter.
	RemoveDispatch (see page 18920)	This is RemoveDispatch, a member of class TDataBlockInterpreter.
	StoreObject (see page 18920)	This is StoreObject, a member of class TDataBlockInterpreter.
	UnlockObject (see page 18920)	This is UnlockObject, a member of class TDataBlockInterpreter.

Public Methods

	Name	Description
	CallCreateObject (see page 18915)	Creates a COM object on a remote application server given its class name.
	InterpretData (see page 18919)	Interprets a call received from a remote application server.

130.1.5.4.1 TDataBlockInterpreter.AddDispatch Method

Pascal

```
procedure AddDispatch(Value: TDataDispatch); override;
```

C++

```
virtual __fastcall AddDispatch(TDataDispatch Value);
```

Description

This is AddDispatch, a member of class TDataBlockInterpreter.

130.1.5.4.2 TDataBlockInterpreter.CallCreateObject Method

Creates a COM object on a remote application server given its class name.

Pascal

```
function CallCreateObject(Name: string): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant CallCreateObject(AnsiString Name);
```

Description

CallCreateObject sends a message to the remote application server, telling it to create an object of the class specified by the Name parameter. Name is the string representation of the Class ID (CLSID). CreateObject (see page 18917) returns a reference to an interface that can be used to communicate with the remote COM object.

See Also

CreateOleObject

130.1.5.4.3 TDataBlockInterpreter.CallFreeObject Method

Pascal

```
procedure CallFreeObject(DispatchIndex: Integer); override;
```

C++

```
virtual __fastcall CallFreeObject(int DispatchIndex);
```

Description

This is CallFreeObject, a member of class TDataBlockInterpreter.

130.1.5.4.4 TDataBlockInterpreter.CallGetIDsOfNames Method

Pascal

```
function CallGetIDsOfNames(DispatchIndex: Integer; const IID: TGUID; Names: Pointer;
NameCount: Integer; LocaleID: Integer; DispIDs: Pointer): HResult; override;
```

C++

```
virtual __fastcall HRESULT CallGetIDsOfNames(int DispatchIndex, const TGUID IID, void * Names,
int NameCount, int LocaleID, void * DispIDs);
```

Description

This is CallGetIDsOfNames, a member of class TDataBlockInterpreter.

130.1.5.4.5 TDataBlockInterpreter.CallGetServerList Method

Pascal

```
function CallGetServerList: OleVariant; override;
```

C++

```
virtual __fastcall OleVariant CallGetServerList();
```

Description

This is CallGetServerList, a member of class TDataBlockInterpreter.

130.1.5.4.6 TDataBlockInterpreter.CallInvoke Method

Pascal

```
function CallInvoke(DispatchIndex: Integer; DispID: Integer; const IID: TGUID; LocaleID:
Integer; Flags: Word; var Params; VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer):
HResult; override;
```

C++

```
virtual __fastcall HRESULT CallInvoke(int DispatchIndex, int DispID, const TGUID IID, int
LocaleID, Word Flags,  Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is CallInvoke, a member of class TDataBlockInterpreter.

130.1.5.4.7 TDataBlockInterpreter.CanCreateObject Method

Pascal

```
function CanCreateObject(const ClassID: TGUID): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanCreateObject(const TGUID ClassID);
```

Description

This is CanCreateObject, a member of class TDataBlockInterpreter.

130.1.5.4.8 TDataBlockInterpreter.CreateObject Method

Pascal

```
function CreateObject(const Name: string): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant CreateObject(const AnsiString Name);
```

Description

This is CreateObject, a member of class TDataBlockInterpreter.

130.1.5.4.9 TDataBlockInterpreter.DoCreateObject Method

Pascal

```
procedure DoCreateObject(const Data: IDataBlock);
```

C++

```
__fastcall DoCreateObject(const IDataBlock Data);
```

Description

This is DoCreateObject, a member of class TDataBlockInterpreter.

130.1.5.4.10 TDataBlockInterpreter.DoCustomAction Method

Pascal

```
function DoCustomAction(Action: Integer; const Data: IDataBlock): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoCustomAction(int Action, const IDataBlock Data);
```

Description

This is DoCustomAction, a member of class TDataBlockInterpreter.

130.1.5.4.11 TDataBlockInterpreter.DoFreeObject Method

Pascal

```
procedure DoFreeObject(const Data: IDataBlock);
```

C++

```
__fastcall DoFreeObject(const IDataBlock Data);
```

Description

This is DoFreeObject, a member of class TDataBlockInterpreter.

130.1.5.4.12 TDataBlockInterpreter.DoGetAppServerList Method

Pascal

```
procedure DoGetAppServerList(const Data: IDataBlock);
```

C++

```
__fastcall DoGetAppServerList(const IDataBlock Data);
```

Description

This is DoGetAppServerList, a member of class TDataBlockInterpreter.

130.1.5.4.13 TDataBlockInterpreter.DoGetIDsOfNames Method

Pascal

```
procedure DoGetIDsOfNames(const Data: IDataBlock);
```

C++

```
__fastcall DoGetIDsOfNames(const IDataBlock Data);
```

Description

This is DoGetIDsOfNames, a member of class TDataBlockInterpreter.

130.1.5.4.14 TDataBlockInterpreter.DoGetServerList Method

Pascal

```
procedure DoGetServerList(const Data: IDataBlock);
```

C++

```
__fastcall DoGetServerList(const IDataBlock Data);
```

Description

This is DoGetServerList, a member of class TDataBlockInterpreter.

130.1.5.4.15 TDataBlockInterpreter.DoInvoke Method

Pascal

```
procedure DoInvoke(const Data: IDataBlock);
```

C++

```
__fastcall DoInvoke(const IDataBlock Data);
```

Description

This is DoInvoke, a member of class TDataBlockInterpreter.

130.1.5.4.16 TDataBlockInterpreter.InternalCreateObject Method

Pascal

```
function InternalCreateObject(const ClassID: TGUID): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalCreateObject(const TGUID ClassID);
```

Description

This is InternalCreateObject, a member of class TDataBlockInterpreter.

130.1.5.4.17 TDataBlockInterpreter.InterpretData Method

Interprets a call received from a remote application server.

Pascal

```
procedure InterpretData(const Data: IDataBlock); override;
```

C++

```
virtual __fastcall InterpretData(const IDataBlock Data);
```

Description

Call InterpretData to respond to a COM interface call that is marshaled to a data block. The Data parameter specifies an IDataBlock (see page 18983) interface that can be used to read the marshaled version of the message.

InterpretData reads the call, interprets it as an IDispatch call, and performs the appropriate actions (such as invoking a local COM interface).

130.1.5.4.18 TDataBlockInterpreter.LockObject Method

Pascal

```
function LockObject(ID: Integer): IDispatch; override;
```

C++

```
virtual __fastcall IDispatch LockObject(int ID);
```

Description

This is LockObject, a member of class TDataBlockInterpreter.

130.1.5.4.19 TDataBlockInterpreter.ReleaseObject Method

Pascal

```
procedure ReleaseObject(ID: Integer); override;
```

C++

```
virtual __fastcall ReleaseObject(int ID);
```

Description

This is ReleaseObject, a member of class TDataBlockInterpreter.

130.1.5.4.20 TDataBlockInterpreter.RemoveDispatch Method

Pascal

```
procedure RemoveDispatch(Value: TDataDispatch); override;
```

C++

```
virtual __fastcall RemoveDispatch(TDataDispatch Value);
```

Description

This is RemoveDispatch, a member of class TDataBlockInterpreter.

130.1.5.4.21 TDataBlockInterpreter.StoreObject Method

Pascal

```
function StoreObject(const Value: OleVariant): Integer; override;
```

C++

```
virtual __fastcall int StoreObject(const OleVariant Value);
```

Description

This is StoreObject, a member of class TDataBlockInterpreter.

130.1.5.4.22 TDataBlockInterpreter.UnlockObject Method

Pascal

```
procedure UnlockObject(ID: Integer; const Disp: IDispatch); override;
```

C++

```
virtual __fastcall UnlockObject(int ID, const IDispatch Disp);
```

Description

This is UnlockObject, a member of class TDataBlockInterpreter.

130.1.6 TDataDispatch Class

Class Hierarchy

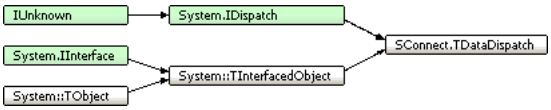

Pascal

```
TDataDispatch = class(TInterfacedObject, IDispatch);
```

C++

```
class TDataDispatch : public TInterfacedObject, public IDispatch;
```

File

SConnect

Description

This is class SConnect.TDataDispatch.

130.1.6.1 TDataDispatch Members

The following tables list the members exposed by TDataDispatch.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetIDsOfNames (see page 18923)	This is GetIDsOfNames, a member of class TDataDispatch.
	GetTypeInfo (see page 18924)	This is GetTypeInfo, a member of class TDataDispatch.
	GetTypeInfoCount (see page 18924)	This is GetTypeInfoCount, a member of class TDataDispatch.
	Invoke (see page 18924)	This is Invoke, a member of class TDataDispatch.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Protected Properties

	Name	Description
R	DispatchIndex (see page 18924)	This is DispatchIndex, a member of class TDataDispatch.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 18922)	This is Create, a member of class TDataDispatch.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 18922)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

130.1.6.2 TDataDispatch.Create Constructor

Pascal

```
constructor Create(Interpreter: TCustomDataBlockInterpreter; DispatchIndex: Integer);
```

C++

```
__fastcall TDataDispatch(TCustomDataBlockInterpreter Interpreter, int DispatchIndex);
```

Description

This is Create, a member of class TDataDispatch.

130.1.6.3 TDataDispatch.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataDispatch();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

130.1.6.4 TDataDispatch Methods

The methods of the TDataDispatch class are listed here.

Protected Methods

	Name	Description
	GetIDsOfNames (see page 18923)	This is GetIDsOfNames, a member of class TDataDispatch.
	GetTypeInfo (see page 18924)	This is GetTypeInfo, a member of class TDataDispatch.
	GetTypeInfoCount (see page 18924)	This is GetTypeInfoCount, a member of class TDataDispatch.
	Invoke (see page 18924)	This is Invoke, a member of class TDataDispatch.

130.1.6.4.1 TDataDispatch.GetIDsOfNames Method

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int NameCount, int
LocaleID, void * DispIDs);
```

Description

This is GetIDsOfNames, a member of class TDataDispatch.

130.1.6.4.2 TDataDispatch.GetTypeInfo Method

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

This is GetTypeInfo, a member of class TDataDispatch.

130.1.6.4.3 TDataDispatch.GetTypeInfoCount Method

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

This is GetTypeInfoCount, a member of class TDataDispatch.

130.1.6.4.4 TDataDispatch.Invoke Method

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word Flags,
Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is Invoke, a member of class TDataDispatch.

130.1.6.5 TDataDispatch Properties

The properties of the TDataDispatch class are listed here.

Protected Properties

	Name	Description
R	DispatchIndex (see page 18924)	This is DispatchIndex, a member of class TDataDispatch.

130.1.6.5.1 TDataDispatch.DispatchIndex Property

Pascal

```
property DispatchIndex: Integer;
```

C++

```
__property int DispatchIndex;
```

Description

This is DispatchIndex, a member of class TDataDispatch.

130.1.7 TPacketInterceptFactory Class

TPacketInterceptFactory is the type of the class factory used to instantiate a data packet interceptor.

Class Hierarchy

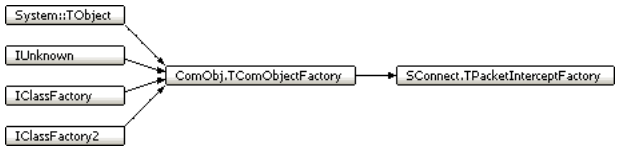

Pascal

```
TPacketInterceptFactory = class(TComObjectFactory);
```

C++

```
class TPacketInterceptFactory : public TComObjectFactory;
```

File

SConnect

Description

In Delphi, use TPacketInterceptFactory as the class factory for COM objects that implement the IDataIntercept (see page 18991) interface. Data packet interceptors implement the IDataIntercept (see page 18991) interface, which they use in multi-tiered database applications to modify the data that passes between a socket connection component in the client application and scktsrvr.exe on the server. TPacketInterceptFactory registers the data packet interceptor so that can appear in the drop-down list of the Object Inspector when setting the socket connection's InterceptName property and in the corresponding list in the properties dialog of scktsrvr.exe.

TPacketInterceptFactory supports the IUnknown, IClassFactory, and IClassFactory2 interfaces. The IClassFactory interface is implemented by class objects associated with class identifiers (CLSIDs) for the classes that they instantiate. IClassFactory must be implemented for every object definition, so that objects of that type can be created. The IClassFactory2 interface is implemented by class factories for classes that need support for licensing.

See Also

ComClassManager

CreateComObject

CreateOleObject

TStreamedConnection (see page 18945)

IDataIntercept (see page 18991)

Specifying a Connection Using Sockets

130.1.7.1 TPacketInterceptFactory Members

The following tables list the members exposed by TPacketInterceptFactory.

Protected Methods

	Name	Description
	_AddRef (see page 5567)	This is _AddRef, a member of class TComObjectFactory.
	_Release (see page 5567)	This is _Release, a member of class TComObjectFactory.
	CreateInstance (see page 5568)	This is CreateInstance, a member of class TComObjectFactory.
	CreateInstanceLic (see page 5568)	This is CreateInstanceLic, a member of class TComObjectFactory.
V	GetLicenseString (see page 5568)	This is GetLicenseString, a member of class TComObjectFactory.
	GetLicInfo (see page 5568)	This is GetLicInfo, a member of class TComObjectFactory.
V	GetProgID (see page 5569)	This is GetProgID, a member of class TComObjectFactory.
V	HasMachineLicense (see page 5569)	This is HasMachineLicense, a member of class TComObjectFactory.
	LockServer (see page 5569)	This is LockServer, a member of class TComObjectFactory.
	QueryInterface (see page 5569)	This is QueryInterface, a member of class TComObjectFactory.
	RequestLicKey (see page 5570)	This is RequestLicKey, a member of class TComObjectFactory.
V	ValidateUserLicense (see page 5571)	This is ValidateUserLicense, a member of class TComObjectFactory.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5565)	Instantiates a class factory object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	CreateComObject (see page 5567)	Instantiates a TComObject (see page 5553) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5566)	Disposes of an instance of a class factory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RegisterClassObject (see page 5569)	Registers the COM class associated with the factory object.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	UpdateRegistry (see page 18927)	Registers or unregisters COM objects created by the factory.

Public Properties

	Name	Description
R	ClassID (see page 5571)	This is ClassID, a member of class TComObjectFactory.
R	ClassName (see page 5572)	Specifies the name of the class that the class factory instantiates.
R	ComClass (see page 5572)	This is ComClass, a member of class TComObjectFactory.
R	ComServer (see page 5572)	This is ComServer, a member of class TComObjectFactory.
R	Description (see page 5572)	Describes the class object.
	ErrorIID (see page 5573)	Indicates the interface of the CoClass in which an error occurred.
R	Instancing (see page 5573)	Indicates the process space of COM object instances.
	LicString (see page 5573)	Contains licensing information about the object.
R	ProgID (see page 5574)	Specifies the name of the class factory.
	ShowErrors (see page 5574)	Indicates whether the COM object factory displays a message box when exceptions are raised.
	SupportsLicensing (see page 5574)	Indicates whether the object being instantiated supports licensing.
R	ThreadingModel (see page 5575)	Indicates the type of thread support provided by objects that the factory creates.

130.1.7.2 TPacketInterceptFactory Methods

The methods of the TPacketInterceptFactory class are listed here.

Public Methods

	Name	Description
V	UpdateRegistry (see page 18927)	Registers or unregisters COM objects created by the factory.

130.1.7.2.1 TPacketInterceptFactory.UpdateRegistry Method

Registers or unregisters COM objects created by the factory.

Pascal

```
procedure UpdateRegistry(Register: Boolean); override;
```

C++

```
virtual __fastcall UpdateRegistry(Boolean Register);
```

Description

UpdateRegistry is called for EXEs when the EXE is run, and for DLLs or EXEs when the routines are called to specifically register or unregister the server.

In addition to the standard COM registration that allows clients to instantiate the data packet interceptor that this factory creates, TPacketInterceptFactory (see page 18925) also registers its associated COM object as a data packet interceptor. This allows the object's ProgID to appear in the drop-down list for a socket connection's InterceptName property, and to appear in the corresponding list on the properties dialog for scktsrvr.exe.

See Also

DLLRegisterServer

DLLUnregisterServer

RegisterClassObject (see page 5569)

InterceptName (see page 18960)

130.1.8 TSocketConnection Class

TSocketConnection uses Windows sockets to manage the connection to an application server.

Class Hierarchy

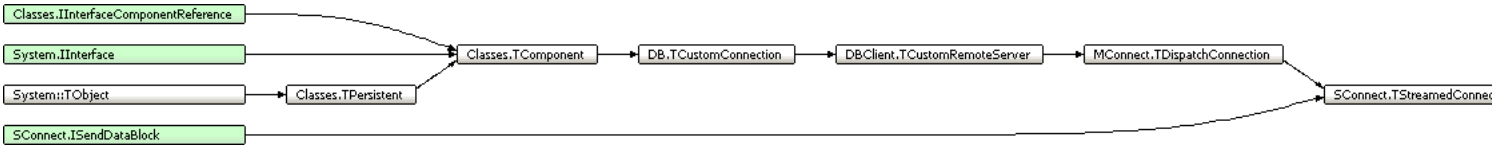

Pascal

```
TSocketConnection = class(TStreamedConnection);
```

C++

```
class TSocketConnection : public TStreamedConnection;
```

File

SConnect

Description

TSocketConnection is used in the client portion of a multi-tiered database application to establish and maintain the connection between the client and a remote application server. The Socket connection object can

Establish an initial connection to a remote application server.

Obtain an IAppServer for the application server.

Call into a separate COM object that encodes and decodes the content of messages.

Obtain a list of providers on the application server.

Drop the connection to the remote application server.

TSocketConnection establishes the initial connection between the client application and a remote application server using TCP/IP. To use TSocketConnection, the application server must be running ScktSrvr.exe.

Once the connection is established, the client registers any or all of its client datasets, and these client datasets use the IAppServer interface from the socket connection component to communicate with providers on the application server, or to otherwise call the application server's data module interface.

To connect to remote application servers using DCOM, use the TDCOMConnection component instead. To connect to remote application servers using HTTP, use the TWebConnection (see page 18968) component instead. To connect using SOAP, use the TSoapConnection component.

See Also

TDataSetProvider (see page 18508)

TClientDataSet (see page 8289)

TDCOMConnection (see page 16008)

TRemoteDataModule (see page 6585)

IAppServer (see page 16450)

TWebConnection (see page 18968)

TSoapConnection (see page 19864)

Specifying a Connection Using Sockets

130.1.8.1 TSocketConnection Members

The following tables list the members exposed by TSocketConnection.

Protected Events

	Name	Description
	OnGetUsername (see page 8533)	This is OnGetUsername, a member of class TCustomRemoteServer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_AddRef (see page 18952)	Increments an internal reference counter.
	_Release (see page 2769)	Called when an application releases a component interface.
	_Release (see page 18952)	Decrements an internal reference counter.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CreateTransport (see page 18934)	Returns an ITransport (see page 18995) interface that can be used to connect to and communicate with the application server.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoConnect (see page 18934)	Connects to a remote application server.
V	DoDisconnect (see page 6864)	Provides the interface for a method that terminates the connection.
V	DoDisconnect (see page 16023)	Drops a connection to an application server.
V	DoDisconnect (see page 18953)	Disconnects from an application server.
V	DoError (see page 18954)	raises an exception for a specified exception object.
V	GetAppServer (see page 8530)	Returns the value of the AppServer (see page 8532) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetConnected (see page 6865)	Returns the value of the Connected (see page 6868) property.
V	GetConnected (see page 16023)	Returns the value of the Connected (see page 16026) property.
V	GetDataSet (see page 6865)	Returns a dataset, given its index in the DataSets (see page 6870) property array.
V	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetInterpreter (see page 18955)	Returns the value of the Interpreter (see page 18960) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetServerCLSID (see page 16024)	Returns the GUID for the application server.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InternalClose (see page 18956)	Terminates a connection to the application server.

V	InternalOpen (see page 18956)	Opens a connection to the application server.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 6866)	Initializes the connection component after all objects in the form or data module have been loaded.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16025)	Responds when other objects are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	QueryInterface (see page 18956)	Implements the IUnknown QueryInterface method.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	RegisterClient (see page 6866)	Registers an object that uses the connection component.
	Send (see page 18957)	Sends an interface call to the application server and returns the response.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAppServer (see page 8531)	Sets the AppServer (see page 8532) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetConnected (see page 6867)	Changes the value of the Connected (see page 6868) property.
V	SetConnected (see page 16025)	Sets the Connected (see page 16026) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	ThreadException (see page 18957)	raises an exception for the thread that communicates with the application server.
	ThreadReceivedStream (see page 18957)	Handles callbacks from the application server.
V	UnRegisterClient (see page 6868)	Removes a registered client so that it is no longer associated with the connection component.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WndProc (see page 18958)	Dispatches messages from the transport layer.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Handle (see page 18959)	Provides access to an internal window handle that can receive messages from the transport layer.
R	Interpreter (see page 18960)	Provides access to the helper object that marshals interface calls for the streamed connection.
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.

Public Events

	Name	Description
	AfterConnect (see page 6874)	Occurs after a connection is established.
	AfterDisconnect (see page 6875)	Occurs after the connection closes.

	BeforeConnect (see page 6875)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 6875)	Occurs immediately before the connection closes.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 6863)	Closes the connection.
V	Create (see page 18933)	Creates and initializes an instance of TSocketConnection (see page 18928).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 6863)	Destroys the instance of a connection component.
V	Destroy (see page 18951)	Disposes of a TStreamedConnection (see page 18945) instance.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetInterceptorList (see page 18954)	Retrieves a list of registered interceptors that can be used to transform messages between the connection component and the application server.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetProviderNames (see page 8530)	Executes a callback for all provider objects on the application server.
V	GetServer (see page 8530)	Returns the IAppServer interface of the application server.
V	GetServer (see page 16023)	Returns the IAppServer interface of the application server.
V	GetServerList (see page 8531)	Returns a list of application servers (remote data modules) to which the component can connect.

V	GetServerList (see page 16024)	Returns a list of remote data modules to which the component can connect.
V	GetServerList (see page 18955)	Retrieves a list of registered data brokers from the remote server system.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 6866)	Opens the connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Send (see page 18994)	Sends information retrieved from an IDataBlock (see page 18983) interface to the application server.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AppServer (see page 8532)	Contains the interface for communicating with the application server.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Connected (see page 6868)	Determines whether a connection has been established to the remote source of data.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 6870)	Lists the datasets currently using this connection component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	LoginPrompt (see page 6872)	Specifies whether a login dialog appears immediately before opening a new connection.
	OnLogin (see page 6873)	Occurs when a communication channel opens to the server.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterConnect (see page 16031)	Occurs after a connection is established.
	AfterDisconnect (see page 16031)	Occurs after the connection closes.
	BeforeConnect (see page 16032)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 16032)	Occurs immediately before the connection closes.
	OnGetUsername (see page 16033)	This is OnGetUsername, a member of class TDispatchConnection.

Published Properties

	Name	Description
	Address (see page 18935)	Specifies the IP address of the server.
	Connected (see page 16026)	Indicates whether the TDispatchConnection (see page 16017) object is connected to an application server.
	Host (see page 18936)	Specifies the host name of the server.
	InterceptGUID (see page 18936)	Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.
	InterceptName (see page 18937)	Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.
	LoginPrompt (see page 16027)	Specifies whether a login dialog appears immediately before opening a new connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectBroker (see page 18937)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	OnLogin (see page 16028)	Occurs when a communication channel opens to the server.
	Port (see page 18938)	Specifies the port number of the socket dispatcher application on the server system.
	ServerGUID (see page 16030)	Specifies the GUID of the application server with which to connect.
	ServerName (see page 16030)	Specifies the name of the application server with which to connect.
	SupportCallbacks (see page 18939)	Determines whether the streamed connection component can handle callbacks in the application server interface.
	Tag (see page 2808)	Stores an integer value as part of a component.

130.1.8.2 TSocketConnection.Create Constructor

Creates and initializes an instance of TSocketConnection (see page 18928).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSocketConnection(TComponent * AOwner);
```

Description

Call Create to instantiate a TSocketConnection (see page 18928) object at runtime. Socket connection components added to forms or data modules at design time are created automatically.

Create overrides the inherited constructor to initialize the Port (see page 18938) property to 211.

See Also

Create (see page 18950)

Port (see page 18938)

130.1.8.3 TSocketConnection Methods

The methods of the TSocketConnection class are listed here.

Protected Methods

	Name	Description
	CreateTransport (see page 18934)	Returns an ITransport (see page 18995) interface that can be used to connect to and communicate with the application server.
	DoConnect (see page 18934)	Connects to a remote application server.

130.1.8.3.1 TSocketConnection.CreateTransport Method

Returns an ITransport (see page 18995) interface that can be used to connect to and communicate with the application server.

Pascal

```
function CreateTransport: ITransport; override;
```

C++

```
virtual __fastcall ITransport CreateTransport();
```

Description

Socket connection components use an ITransport (see page 18995) interface to connect to and communicate with the application server. CreateTransport instantiates the internal helper object that connects to the socket dispatcher on the server and sends or receives messages. CreateTransport passes the values of the Host (see page 18936), Address (see page 18935) and Port (see page 18938) properties to this helper object and returns its ITransport (see page 18995) interface.

See Also

Host (see page 18936)

Address (see page 18935)

Port (see page 18938)

130.1.8.3.2 TSocketConnection.DoConnect Method

Connects to a remote application server.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

DoConnect is used internally to connect to the application server when the Connected property is set to true. If the Host (see page 18936) or Address (see page 18935) and Port (see page 18938) properties are set, DoConnect uses these to locate the socket on the server machine. Otherwise, if the ObjectBroker (see page 18937) property is set, it obtains these values from the object broker.

Once the server machine is located, DoConnect opens a connection.

Note: When ObjectBroker (see page 18937) is nil (Delphi) or NULL (C++), the values of Host (see page 18936), Address (see page 18935), and Port (see page 18938) are saved with the socket connection component. If ObjectBroker (see page 18937) is not nil (Delphi) or NULL (C++), these values are not saved from design time. However, once these values are set (either by values obtained from the Object Broker or by values specified in code at runtime), they are used every time a connection must be re-established. If these values subsequently fail, the socket connection component fetches new values from the Object Broker.

See Also

Connected (see page 16026)

Host (see page 18936)

Address (see page 18935)

Port (see page 18938)

ObjectBroker (see page 16028)

130.1.8.4 TSocketConnection Properties

The properties of the TSocketConnection class are listed here.

Published Properties

	Name	Description
	Address (see page 18935)	Specifies the IP address of the server.
	Host (see page 18936)	Specifies the host name of the server.
	InterceptGUID (see page 18936)	Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.
	InterceptName (see page 18937)	Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.
	ObjectBroker (see page 18937)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	Port (see page 18938)	Specifies the port number of the socket dispatcher application on the server system.
	SupportCallbacks (see page 18939)	Determines whether the streamed connection component can handle callbacks in the application server interface.

130.1.8.4.1 TSocketConnection.Address Property

Specifies the IP address of the server.

Pascal

```
property Address: string;
```

C++

```
__property AnsiString Address;
```

Description

Use Address to specify the IP address of the application server to which the client application connects. In addition to the address, you must provide a value for the Port (see page 18938) property so that the socket connection component can locate the listening socket on the server system.

Sockets (see page 20039) require either an IP address or a Host (see page 18936) name to locate the server system. You can use the Host (see page 18936) property instead of the Address property to specify the server system using a host name. The Address and Host (see page 18936) properties are mutually exclusive: Setting Address causes Host (see page 18936) to be set to an empty string, and vice versa.

Note: If the socket connection component uses an Object Broker, you do not need to specify the Address property. Instead, the Object Broker supplies an Address for a server that is located dynamically at runtime. Explicitly setting the Address property overrides the value supplied by the Object Broker, but such overridden values are not saved with the socket connection component.

See Also

Host (see page 18936)

Port (see page 18938)

ObjectBroker (see page 16028)

Specifying a Connection Using Sockets

130.1.8.4.2 TSocketConnection.Host Property

Specifies the host name of the server.

Pascal

```
property Host: string;
```

C++

```
__property AnsiString Host;
```

Description

Use Host to specify the host name of the server system to which the client application should connect. In addition to the host name, you must provide a value for the Port (see page 18938) property so that the socket connection component can locate the listening socket on the server system.

Sockets (see page 20039) require either a Host name or an IP address to locate the server system. You can use the Address (see page 18935) property instead of the Host property to specify the server system using an IP address. The Host and Address (see page 18935) properties are mutually exclusive: Setting Host causes Address (see page 18935) to be set to an empty string, and vice versa.

Note: If the socket connection component uses an Object Broker, you do not need to specify the Host property. Instead, the Object Broker supplies a host name for a server that is located dynamically at runtime. Explicitly setting the Host property overrides the value supplied by the Object Broker, but such overridden values are not saved with the socket connection component.

See Also

Address (see page 18935)

Port (see page 18938)

ObjectBroker (see page 16028)

Specifying a Connection Using Sockets

130.1.8.4.3 TSocketConnection.InterceptGUID Property

Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.

Pascal

```
property InterceptGUID: string;
```

C++

```
__property AnsiString InterceptGUID;
```

Description

Set InterceptGUID to the GUID of a COM object that intercepts messages between the connection component and an application server before they are sent and after they are received.

The COM object must implement the IDataIntercept interface. Use the IDataIntercept methods to manipulate the data that is communicated using the socket connection. For example, applications can use these methods to encrypt and decrypt data or to compress and decompress data.

Tip: To specify the COM object that intercepts messages using a ProgID instead of a GUID, use the InterceptName property instead.

Note: When using InterceptGUID, the server must use the same COM object to undo the transformation performed by the socket connection component's InterceptGUID. For servers that respond to TSocketConnection, you can set the server's InterceptGUID to the same GUID using the Socket server dialog of ScktSrvr.exe.

See Also

IDataIntercept (see page 18991)

InterceptName (see page 18960)

GetInterceptorList (see page 18954)

Specifying a Connection Using Sockets

130.1.8.4.4 TSocketConnection.InterceptName Property

Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.

Pascal

```
property InterceptName: string;
```

C++

```
__property AnsiString InterceptName;
```

Description

Set InterceptName to the programmatic ID (ProgID) of a COM object that intercepts messages between the connection component and an application server before they are sent and after they are received.

Note: In Delphi, you can choose the ProgID of any COM objects that have been registered to perform this task at Design time, using a drop-down list in the Object Inspector, if they use TPacketInterceptFactory as a class factory.

The COM object must implement the IDataIntercept interface. Use the IDataIntercept methods to manipulate the data that is communicated using the socket connection. For example, applications can use these methods to encrypt and decrypt data or to compress and decompress data.

Tip: To specify the COM object that intercepts messages using a GUID instead of a ProgID, use the InterceptGUID property instead.

Note: When using InterceptName, the server must use the same COM object to undo the transformation performed by the socket connection component's InterceptName. For servers that respond to TSocketConnection, you can set the COM object that the server uses with the Socket server dialog of ScktSrvr.exe.

See Also

IDataIntercept (see page 18991)

InterceptGUID (see page 18959)

GetInterceptorList (see page 18954)

TPacketInterceptFactory (see page 18925)

Specifying a Connection Using Sockets

130.1.8.4.5 TSocketConnection.ObjectBroker Property

Specifies an Object Broker that allows the connection component to choose from several application servers.

Pascal

```
property ObjectBroker: TCustomObjectBroker;
```

C++

```
__property TCustomObjectBroker ObjectBroker;
```

Description

Set ObjectBroker to use a broker that maintains a list of servers that support the application specified by ServerName or ServerGUID. When the connection component needs to open a connection, it requests information to identify a server machine from the Object Broker.

Descendants of TDispatchConnection define properties that specify how to locate server machines. These properties are not saved with the connection component if the ObjectBroker property is set. Instead, the Object Broker provides this information dynamically at runtime from its list of available servers.

Applications can use TSimpleObjectBroker for an object broker, or derive their own object brokers from TCustomObjectBroker.

See Also

DoConnect (see page 6864)

ServerName (see page 16030)

ServerGUID (see page 16030)

TSimpleObjectBroker (see page 17877)

Brokering Connections

Specifying a Connection Using HTTP

Specifying a Connection Using DCOM

Specifying a Connection Using Sockets

Using TSimpleObjectBroker

130.1.8.4.6 TSocketConnection.Port Property

Specifies the port number of the socket dispatcher application on the server system.

Pascal

```
property Port: Integer;
```

C++

```
__property int Port;
```

Description

Use Port to specify the port that the socket dispatcher uses to listen for client connection requests. By default, Port is 211, which is the default port number of ScktSrvr.exe (the socket dispatcher application). Change the value of Port if the socket dispatcher is configured to use a different port.

Note: If the socket connection component uses an Object Broker, you do not need to specify the Port property. Instead, the Object Broker supplies a port number for a server that is located dynamically at runtime. Explicitly setting the Port property overrides the value supplied by the Object Broker, but such overridden values are not saved with the socket connection component.

See Also

Address (see page 18935)

Host (see page 18936)

ObjectBroker (see page 16028)

Specifying a Connection Using Sockets

130.1.8.4.7 TSocketConnection.SupportCallbacks Property

Determines whether the streamed connection component can handle callbacks in the application server interface.

Pascal

```
property SupportCallbacks: Boolean;
```

C++

```
__property Boolean SupportCallbacks;
```

Description

Use SupportCallbacks to indicate whether the connection component can marshal callback functions in the application server's interface. When SupportCallbacks is true, the connection component can marshal calls from the application server to the client application over an interface supplied as a callback.

For TSocketConnection, callbacks are optional. Set SupportCallbacks to true if you are using callbacks as part of the application server's interface. When SupportCallbacks is true, the socket connection component requires WinSock2. Set SupportCallbacks to false if you are not using callback functions and want to eliminate the dependency on WinSock2.

TWebConnection can't support callback functions. It initializes SupportCallbacks to false in the constructor and never changes this value.

See Also

Send (see page 18957)

Extending the Interface of the Application Server

130.1.9 TSocketTransport Class

Class Hierarchy

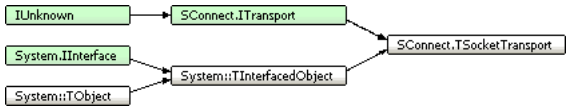

Pascal

```
TSocketTransport = class(TInterfacedObject, ITransport);
```

C++

```
class TSocketTransport : public TInterfacedObject, public ITransport;
```

File

SConnect

Description

This is class SConnect.TSocketTransport.

130.1.9.1 TSocketTransport Members

The following tables list the members exposed by TSocketTransport.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetConnected (see page 18942)	Returns the value of the Connected property.
	GetWaitEvent (see page 18943)	Returns the handle for an event object that is managed by the object that implements ITransport.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	Receive (see page 18943)	Returns an IDataBlock interface that can be used to read information received from the application server.
	Send (see page 18943)	Sends information retrieved from an IDataBlock interface to the application server.
	SetConnected (see page 18944)	Sets the value of the Connected property.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 18941)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 18941)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Address (see page 18944)	This is Address, a member of class TSocketTransport.
	Connected (see page 18998)	Indicates whether the connection to the application server is established.
	Host (see page 18945)	This is Host, a member of class TSocketTransport.
	InterceptGUID (see page 18945)	This is InterceptGUID, a member of class TSocketTransport.
	Port (see page 18945)	This is Port, a member of class TSocketTransport.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Socket (see page 18945)	This is Socket, a member of class TSocketTransport.

130.1.9.2 TSocketTransport.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TSocketTransport();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

130.1.9.3 TSocketTransport.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSocketTransport();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

130.1.9.4 TSocketTransport Methods

The methods of the TSocketTransport class are listed here.

Protected Methods

	Name	Description
	GetConnected (see page 18942)	Returns the value of the Connected property.
	GetWaitEvent (see page 18943)	Returns the handle for an event object that is managed by the object that implements ITransport.
	Receive (see page 18943)	Returns an IDataBlock interface that can be used to read information received from the application server.
	Send (see page 18943)	Sends information retrieved from an IDataBlock interface to the application server.
	SetConnected (see page 18944)	Sets the value of the Connected property.

130.1.9.4.1 TSocketTransport.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; stdcall;
```

C++

```
__fastcall __stdcall Boolean GetConnected();
```

Description

GetConnected is the read property implementer for the Connected property.

See Also

Connected (see page 18998)

130.1.9.4.2 TSocketTransport.GetWaitEvent Method

Returns the handle for an event object that is managed by the object that implements ITransport.

Pascal

```
function GetWaitEvent: THandle; stdcall;
```

C++

```
__fastcall __stdcall THandle GetWaitEvent();
```

Description

Streamed connection components perform all reading from and writing to the application server using a separate execution thread. This limits the impact that communication with the application server has on the speed of the client application, because execution does not need to wait while messages are marshaled and sent or received. The streamed connection component needs to coordinate its use of the communication thread. To do this, it uses an event object that is a member of the object that implements ITransport.

GetWaitEvent returns a handle to this event object. When implementing an object that supports ITransport, create a TEvent object as a member and return its handle as the return value of GetWaitEvent. Do not use the event object for internal purposes or it will interfere with the streamed connection's use of the event.

See Also

TEvent (see page 22063)

130.1.9.4.3 TSocketTransport.Receive Method

Returns an IDataBlock interface that can be used to read information received from the application server.

Pascal

```
function Receive(WaitForInput: Boolean; Context: Integer): IDataBlock; stdcall;
```

C++

```
__fastcall __stdcall IDataBlock Receive(Boolean WaitForInput, int Context);
```

Description

Receive creates a COM object that implements the IDataBlock interface and reads information from the application server into its Memory property.

If WaitForInput is true, Receive halts execution until the application server sends the expected information. If WaitForInput is false, Receive can fail (reading no information) if the application server does not send any information immediately (or after a brief time-out period).

Context is transport-specific data that can be used when implementing a class that supports ITransport. Its value is the return value of the Send method.

See Also

Memory (see page 18989)

Send (see page 18997)

130.1.9.4.4 TSocketTransport.Send Method

Sends information retrieved from an IDataBlock interface to the application server.

Pascal

```
function Send(const Data: IDataBlock): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Send(const IDataBlock Data);
```

Description

Send sends the entire contents of the Data parameter's Memory property to the application server. It returns a transport-specific value that is used by the next call to the Receive method. For example, HTTP transport classes return the handle for the Internet connection.

See Also

Memory (see page 18989)

Receive (see page 18997)

130.1.9.4.5 TSocketTransport.SetConnected Method

Sets the value of the Connected property.

Pascal

```
procedure SetConnected(Value: Boolean); stdcall;
```

C++

```
__fastcall __stdcall SetConnected(Boolean Value);
```

Description

SetConnected is the write implementer for the Connected property.

130.1.9.5 TSocketTransport Properties

The properties of the TSocketTransport class are listed here.

Public Properties

	Name	Description
	Address (see page 18944)	This is Address, a member of class TSocketTransport.
	Host (see page 18945)	This is Host, a member of class TSocketTransport.
	InterceptGUID (see page 18945)	This is InterceptGUID, a member of class TSocketTransport.
	Port (see page 18945)	This is Port, a member of class TSocketTransport.
	Socket (see page 18945)	This is Socket, a member of class TSocketTransport.

130.1.9.5.1 TSocketTransport.Address Property

Pascal

```
property Address: string;
```

C++

```
__property AnsiString Address;
```

Description

This is Address, a member of class TSocketTransport.

130.1.9.5.2 TSocketTransport.Host Property

Pascal

```
property Host: string;
```

C++

```
__property AnsiString Host;
```

Description

This is Host, a member of class TSocketTransport.

130.1.9.5.3 TSocketTransport.InterceptGUID Property

Pascal

```
property InterceptGUID: string;
```

C++

```
__property AnsiString InterceptGUID;
```

Description

This is InterceptGUID, a member of class TSocketTransport.

130.1.9.5.4 TSocketTransport.Port Property

Pascal

```
property Port: Integer;
```

C++

```
__property int Port;
```

Description

This is Port, a member of class TSocketTransport.

130.1.9.5.5 TSocketTransport.Socket Property

Pascal

```
property Socket: TCustomWinSocket;
```

C++

```
__property TCustomWinSocket Socket;
```

Description

This is Socket, a member of class TSocketTransport.

130.1.10 TStreamedConnection Class

TStreamedConnection is the base class for components that handle their own marshaling of interface calls when connecting to application servers.

Class Hierarchy

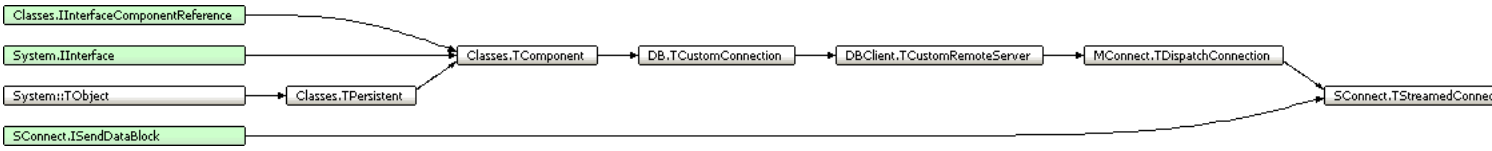

Pascal

```
TStreamedConnection = class(TDispatchConnection, ISendDataBlock);
```

C++

```
class TStreamedConnection : public TDispatchConnection, public ISendDataBlock;
```

File

SConnect

Description

Do not create instances of TStreamedConnection. Instead, instantiate a descendant of TStreamedConnection such as TSocketConnection (see page 18928) or TWebConnection (see page 18968).

TStreamedConnection handles the marshaling of all COM data types so that the streamed connection component can generate, interpret, and respond to remote COM interface calls without using DCOM. It uses an ITransport (see page 18995) interface to connect to application servers and to send and receive messages.

TStreamedConnection can't actually connect to or communicate with an application server. Descendants of TStreamedConnection override the CreateTransport (see page 18952) method to obtain an ITransport (see page 18995) interface that enables these behaviors.

TStreamedConnection implements the ISendDataBlock (see page 18993) interface.

See Also

TClientDataSet (see page 8289)

TSocketConnection (see page 18928)

TWebConnection (see page 18968)

TStream (see page 2973)

ITransport (see page 18995)

TRemoteDataModule (see page 6585)

IAppServer (see page 16450)

130.1.10.1 TStreamedConnection Members

The following tables list the members exposed by TStreamedConnection.

Protected Events

	Name	Description
	OnGetUsername (see page 8533)	This is OnGetUsername, a member of class TCustomRemoteServer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 18952)	Increments an internal reference counter.
	_Release (see page 18952)	Decrements an internal reference counter.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CreateTransport (see page 18952)	Provides the backstop for a method that obtains an ITransport (see page 18995) interface.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoConnect (see page 18953)	Connects to an application server.
V	DoDisconnect (see page 18953)	Disconnects from an application server.
V	DoError (see page 18954)	raises an exception for a specified exception object.
V	GetAppServer (see page 8530)	Returns the value of the AppServer (see page 8532) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetConnected (see page 6865)	Returns the value of the Connected (see page 6868) property.
V	GetConnected (see page 16023)	Returns the value of the Connected (see page 16026) property.
V	GetDataSet (see page 6865)	Returns a dataset, given its index in the DataSets (see page 6870) property array.
V	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetInterpreter (see page 18955)	Returns the value of the Interpreter (see page 18960) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetServerCLSID (see page 16024)	Returns the GUID for the application server.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InternalClose (see page 18956)	Terminates a connection to the application server.
V	InternalOpen (see page 18956)	Opens a connection to the application server.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 6866)	Initializes the connection component after all objects in the form or data module have been loaded.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16025)	Responds when other objects are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 18956)	Implements the IUnknown QueryInterface method.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	RegisterClient (see page 6866)	Registers an object that uses the connection component.
	Send (see page 18957)	Sends an interface call to the application server and returns the response.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAppServer (see page 8531)	Sets the AppServer (see page 8532) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetConnected (see page 6867)	Changes the value of the Connected (see page 6868) property.
V	SetConnected (see page 16025)	Sets the Connected (see page 16026) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	ThreadException (see page 18957)	raises an exception for the thread that communicates with the application server.
	ThreadReceivedStream (see page 18957)	Handles callbacks from the application server.
V	UnRegisterClient (see page 6868)	Removes a registered client so that it is no longer associated with the connection component.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WndProc (see page 18958)	Dispatches messages from the transport layer.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Handle (see page 18959)	Provides access to an internal window handle that can receive messages from the transport layer.
	InterceptGUID (see page 18959)	Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.
	InterceptName (see page 18960)	Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.
R	Interpreter (see page 18960)	Provides access to the helper object that marshals interface calls for the streamed connection.
	ObjectBroker (see page 16028)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.
	SupportCallbacks (see page 18961)	Determines whether the streamed connection component can handle callbacks in the application server interface.

Public Events

	Name	Description
	AfterConnect (see page 6874)	Occurs after a connection is established.
	AfterDisconnect (see page 6875)	Occurs after the connection closes.
	BeforeConnect (see page 6875)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 6875)	Occurs immediately before the connection closes.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 6863)	Closes the connection.
V	Create (see page 18950)	Creates and initializes an instance of TStreamedConnection (see page 18945).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 18951)	Disposes of a TStreamedConnection (see page 18945) instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetInterceptorList (see page 18954)	Retrieves a list of registered interceptors that can be used to transform messages between the connection component and the application server.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetProviderNames (see page 8530)	Executes a callback for all provider objects on the application server.
V	GetServer (see page 8530)	Returns the IAppServer interface of the application server.
V	GetServer (see page 16023)	Returns the IAppServer interface of the application server.
V	GetServerList (see page 18955)	Retrieves a list of registered data brokers from the remote server system.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 6866)	Opens the connection.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AppServer (see page 8532)	Contains the interface for communicating with the application server.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Connected (see page 6868)	Determines whether a connection has been established to the remote source of data.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 6870)	Lists the datasets currently using this connection component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	LoginPrompt (see page 6872)	Specifies whether a login dialog appears immediately before opening a new connection.
	OnLogin (see page 6873)	Occurs when a communication channel opens to the server.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterConnect (see page 16031)	Occurs after a connection is established.
	AfterDisconnect (see page 16031)	Occurs after the connection closes.
	BeforeConnect (see page 16032)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 16032)	Occurs immediately before the connection closes.
	OnGetUsername (see page 16033)	This is OnGetUsername, a member of class TDispatchConnection.

Published Properties

	Name	Description
	Connected (see page 16026)	Indicates whether the TDispatchConnection (see page 16017) object is connected to an application server.
	LoginPrompt (see page 16027)	Specifies whether a login dialog appears immediately before opening a new connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OnLogin (see page 16028)	Occurs when a communication channel opens to the server.
	ServerGUID (see page 16030)	Specifies the GUID of the application server with which to connect.
	ServerName (see page 16030)	Specifies the name of the application server with which to connect.
	Tag (see page 2808)	Stores an integer value as part of a component.

130.1.10.2 TStreamedConnection.Create Constructor

Creates and initializes an instance of TStreamedConnection (see page 18945).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TStreamedConnection(TComponent * AOwner);
```

Description

Call Create to instantiate a streamed connection component at runtime. Connection components placed in forms or data

modules at design time are instantiated automatically.

After calling the inherited constructor, Create instantiates the data block interpreter accessed by the Interpreter (see page 18960) property.

See Also

Interpreter (see page 18960)

Create (see page 8529)

130.1.10.3 TStreamedConnection.Destroy Destructor

Disposes of a TStreamedConnection (see page 18945) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TStreamedConnection();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the streamed connection is not nil, and only then calls Destroy.

Destroy ensures that the connection to the server is closed. It then frees all resources and helper objects before calling the inherited destructor.

See Also

Free (see page 22300)

130.1.10.4 TStreamedConnection Methods

The methods of the TStreamedConnection class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 18952)	Increments an internal reference counter.
	_Release (see page 18952)	Decrements an internal reference counter.
V	CreateTransport (see page 18952)	Provides the backstop for a method that obtains an ITransport (see page 18995) interface.
V	DoConnect (see page 18953)	Connects to an application server.
V	DoDisconnect (see page 18953)	Disconnects from an application server.
V	DoError (see page 18954)	raises an exception for a specified exception object.
V	GetInterpreter (see page 18955)	Returns the value of the Interpreter (see page 18960) property.
V	InternalClose (see page 18956)	Terminates a connection to the application server.
V	InternalOpen (see page 18956)	Opens a connection to the application server.
	QueryInterface (see page 18956)	Implements the IUnknown QueryInterface method.
	Send (see page 18957)	Sends an interface call to the application server and returns the response.
	ThreadException (see page 18957)	raises an exception for the thread that communicates with the application server.
	ThreadReceivedStream (see page 18957)	Handles callbacks from the application server.
	WndProc (see page 18958)	Dispatches messages from the transport layer.

Public Methods

	Name	Description
	GetInterceptorList (see page 18954)	Retrieves a list of registered interceptors that can be used to transform messages between the connection component and the application server.
	GetServerList (see page 18955)	Retrieves a list of registered data brokers from the remote server system.

130.1.10.4.1 TStreamedConnection._AddRef Method

Increments an internal reference counter.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef implements the IUnknown interface method, _AddRef, by incrementing an internal reference counter. The reference count is decreased by calling _Release (see page 18952).

See Also

_Release (see page 18952)

QueryInterface (see page 18956)

130.1.10.4.2 TStreamedConnection._Release Method

Decrements an internal reference counter.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release implements the IUnknown interface method, _Release, by decrementing an internal reference counter. The reference count is increased by calling _AddRef (see page 18952)

See Also

_AddRef (see page 18952)

QueryInterface (see page 18956)

130.1.10.4.3 TStreamedConnection.CreateTransport Method

Provides the backstop for a method that obtains an ITransport (see page 18995) interface.

Pascal

```
function CreateTransport: ITransport; virtual;
```

C++

```
virtual __fastcall ITransport CreateTransport();
```

Description

Streamed connection components use an ITransport (see page 18995) interface to connect to and communicate with the application server. As implemented in TStreamedConnection (see page 18945), CreateTransport always returns nil (Delphi) or NULL (C++).

Descendants of TStreamedConnection (see page 18945) override CreateTransport to obtain an ITransport (see page 18995) interface that connects to and communicates with the application server.

Component writers wanting to create connection components for specialized transport protocols need only override CreateTransport. Once the streamed connection has a working ITransport (see page 18995) interface, all other functions of the connection component are handled automatically.

See Also

Send (see page 18957)

DoConnect (see page 18953)

DoDisconnect (see page 18953)

ThreadReceivedStream (see page 18957)

130.1.10.4.4 TStreamedConnection.DoConnect Method

Connects to an application server.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

DoConnect is called internally when the Connected property is set to true. DoConnect creates a separate thread for managing the connection so that communication with the application server does not block execution in the client application. Using the ITransport (see page 18995) interface, DoConnect connects to the application server and sets the value of the AppServer property to its interface.

Note: Descendants of TStreamedConnection (see page 18945) override CreateTransport (see page 18952) rather than DoConnect to change the way the client application forms a connection to application servers.

See Also

AfterConnect (see page 6874)

BeforeConnect (see page 6875)

Connected (see page 16026)

AppServer (see page 8532)

CreateTransport (see page 18952)

130.1.10.4.5 TStreamedConnection.DoDisconnect Method

Disconnects from an application server.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

DoDisconnect is called internally when the Connected property is set to false. Using the ITransport (see page 18995) interface, DoDisconnect drops the connection to the application server.

See Also

Connected (see page 16026)

CreateTransport (see page 18952)

130.1.10.4.6 TStreamedConnection.DoError Method

raises an exception for a specified exception object.

Pascal

```
procedure DoError(E: Exception); virtual;
```

C++

```
virtual __fastcall DoError(Exception E);
```

Description

TStreamedConnection (see page 18945) uses a separate thread when making interface calls to the application server. When that thread encounters an exception, it sends a message to the Window handle specified by the Handle (see page 18959) property. The connection component uses DoError to raise those exceptions.

Note: DoError is called by the ThreadException (see page 18957) method, which is the message handler for thread exceptions. It is also called when thread exception messages are taken from the message queue before they are received by the internal window procedure.

See Also

ThreadException (see page 18957)

Handle (see page 18959)

EInterpreterError (see page 18897)

130.1.10.4.7 TStreamedConnection.GetInterceptorList Method

Retrieves a list of registered interceptors that can be used to transform messages between the connection component and the application server.

Pascal

```
function GetInterceptorList: OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant GetInterceptorList();
```

Description

Call GetInterceptorList to obtain a list of ProgIDs that can be assigned to the InterceptName (see page 18960) property. GetInterceptorList retrieves a list of all COM objects that have been registered by TPacketInterceptFactory (see page 18925)

as data packet interceptors (objects that support the IDataIntercept (see page 18991) interface that can be used to intercept and transform messages passed between a connection component and an application server.)

GetInterceptorList returns a Variant array of strings. Any of these strings can be used as a valid setting for the InterceptName (see page 18960) property.

At design time, TStreamedConnection (see page 18945) uses the GetInterceptorList method to populate the InterceptName (see page 18960) drop-down in the Object Inspector.

See Also

IDataIntercept (see page 18991)

InterceptName (see page 18960)

InterceptGUID (see page 18959)

TPacketInterceptFactory (see page 18925)

130.1.10.4.8 TStreamedConnection.GetInterpreter Method

Returns the value of the Interpreter (see page 18960) property.

Pascal

```
function GetInterpreter: TCustomDataBlockInterpreter; virtual;
```

C++

```
virtual __fastcall TCustomDataBlockInterpreter GetInterpreter();
```

Description

GetInterpreter is the protected read implementation of the Interpreter (see page 18960) property. As implemented in TStreamedConnection (see page 18945), GetInterpreter instantiates a TDataBlockInterpreter (see page 18913) object.

See Also

Interpreter (see page 18960)

TDataBlockInterpreter (see page 18913)

130.1.10.4.9 TStreamedConnection.GetServerList Method

Retrieves a list of registered data brokers from the remote server system.

Pascal

```
function GetServerList: OleVariant; override;
```

C++

```
virtual __fastcall OleVariant GetServerList();
```

Description

Call GetServerList to obtain a list of servers that are registered on the system to which the streamed connection communicates. GetServerList returns a Variant array of strings. Any of these strings can be used as a valid setting for the ServerName property.

At design time, TStreamedConnection (see page 18945) uses the GetServerList method to populate the ServerName drop-down in the Object Inspector. Note that, unlike COM-based connection components (which populate that list with servers that are registered on the client machine), this results in a list of application servers that are registered on the server machine.

See Also

ServerName (see page 16030)

130.1.10.4.10 TStreamedConnection.InternalClose Method

Terminates a connection to the application server.

Pascal

```
procedure InternalClose; virtual;
```

C++

```
virtual __fastcall InternalClose();
```

Description

Streamed connection components use InternalClose to handle the details of closing a connection to the application server. This method is called by the DoDisconnect (see page 18953) method (which implements the Connected property) and at other times when the connection component must terminate a connection.

See Also

Connected (see page 16026)

DoDisconnect (see page 18953)

130.1.10.4.11 TStreamedConnection.InternalOpen Method

Opens a connection to the application server.

Pascal

```
procedure InternalOpen; virtual;
```

C++

```
virtual __fastcall InternalOpen();
```

Description

Streamed connection components use InternalOpen to handle the details of establishing a connection to the application server. This method is called by the DoConnect (see page 18953) method (which implements the Connected property) and at other times when the connection component must establish a connection.

See Also

Connected (see page 16026)

DoConnect (see page 18953)

130.1.10.4.12 TStreamedConnection.QueryInterface Method

Implements the IUnknown QueryInterface method.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall; reintroduce;
```

C++

```
__fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface returns a reference to the interface specified by the IID parameter as the Obj parameter. It the streamed connection supports the specified interface, QueryInterface returns 0. If the streamed connection does not support the interface, the Obj parameter is nil (Delphi) or NULL (C++) and QueryInterface returns E_NOINTERFACE.

See Also

_AddRef (see page 18952)

ISendDataBlock

130.1.10.4.13 TStreamedConnection.Send Method

Sends an interface call to the application server and returns the response.

Pascal

```
function Send(const Data: IDataBlock; WaitForResult: Boolean): IDataBlock; stdcall;
```

C++

```
__fastcall __stdcall IDataBlock Send(const IDataBlock Data, Boolean WaitForResult);
```

Description

The data block interpreter (specified by the Interpreter (see page 18960) property) calls Send to dispatch interface messages to the application server. It supplies an IDataBlock (see page 18983) interface that encapsulates the interface from the application server.

Communication with the application server occurs on a separate thread. The WaitForResult parameter indicates whether execution on the main thread should halt until the interface call returns a result (WaitForResult == true), or whether the interface call can occur asynchronously (WaitForResult == false). If WaitForResult is true, Send returns an IDataBlock (see page 18983) interface (_di_IDataBlock in C++) that supplies the result.

See Also

Interpreter (see page 18960)

ThreadReceivedStream (see page 18957)

130.1.10.4.14 TStreamedConnection.ThreadException Method

raises an exception for the thread that communicates with the application server.

Pascal

```
procedure ThreadException(var Message: TMessage); message THREAD_EXCEPTION;
```

C++

```
__fastcall ThreadException(TMessage Message);
```

Description

TStreamedConnection (see page 18945) uses a separate thread when making interface calls to the application server. When that thread encounters an exception, it sends a message to the Window handle specified by the Handle (see page 18959) property. ThreadException is the message handler for that Windows message. It raises an exception in the main execution thread.

See Also

Handle (see page 18959)

130.1.10.4.15 TStreamedConnection.ThreadReceivedStream Method

Handles callbacks from the application server.

Pascal

```
procedure ThreadReceivedStream(var Message: TMessage); message THREAD_RECEIVEDSTREAM;
```

C++

```
__fastcall ThreadReceivedStream(TMessage Message);
```

Description

TStreamedConnection (see page 18945) uses a separate thread for interface calls to and from the application server. When the application server calls a client application interface, that thread sends a message to the Window handle specified by the Handle (see page 18959) property. ThreadReceivedStream is the message handler for the Windows message from the application server.

ThreadReceivedStream passes the interface call to the data block interpreter specified by the Interpreter (see page 18960) property.

See Also

Handle (see page 18959)

Interpreter (see page 18960)

130.1.10.4.16 TStreamedConnection.WndProc Method

Dispatches messages from the transport layer.

Pascal

```
procedure WndProc(var Message: TMessage);
```

C++

```
__fastcall WndProc(TMessage Message);
```

Description

TStreamedConnection (see page 18945) uses a separate thread for interface calls to and from the application server. That thread communicates with the streamed connection component by sending Windows messages to the handle specified by the Handle (see page 18959) property. WndProc is the window procedure associated with that Window handle. It dispatches any messages that have message handlers such as ThreadException (see page 18957) or ThreadReceivedStream (see page 18957). All other messages are ignored.

See Also

Handle (see page 18959)

ThreadException (see page 18957)

ThreadReceivedStream (see page 18957)

130.1.10.5 TStreamedConnection Properties

The properties of the TStreamedConnection class are listed here.

Protected Properties

	Name	Description
R	Handle (see page 18959)	Provides access to an internal window handle that can receive messages from the transport layer.
	InterceptGUID (see page 18959)	Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.

	InterceptName (see page 18960)	Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.
R	Interpreter (see page 18960)	Provides access to the helper object that marshals interface calls for the streamed connection.
	SupportCallbacks (see page 18961)	Determines whether the streamed connection component can handle callbacks in the application server interface.

130.1.10.5.1 TStreamedConnection.Handle Property

Provides access to an internal window handle that can receive messages from the transport layer.

Pascal

```
property Handle: THandle;
```

C++

```
__property THandle Handle;
```

Description

Streamed connection components perform all reading from and writing to the application server using a separate execution thread. This limits the impact that communication with the application server has on the speed of the client application, because execution does not need to wait while messages are marshaled and sent or received.

The internal thread that handles these communication tasks uses Windows messages to coordinate events. TStreamedConnection (see page 18945) uses the Handle property to provide this internal thread with a window handle that it can use.

See Also

WndProc (see page 18958)

ThreadReceivedStream (see page 18957)

ThreadException (see page 18957)

130.1.10.5.2 TStreamedConnection.InterceptGUID Property

Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.

Pascal

```
property InterceptGUID: string;
```

C++

```
__property AnsiString InterceptGUID;
```

Description

Set InterceptGUID to the GUID of a COM object that intercepts messages between the connection component and an application server before they are sent and after they are received.

The COM object must implement the IDataIntercept (see page 18991) interface. Use the IDataIntercept (see page 18991) methods to manipulate the data that is communicated using the socket connection. For example, applications can use these methods to encrypt and decrypt data or to compress and decompress data.

Tip: To specify the COM object that intercepts messages using a ProgID instead of a GUID, use the InterceptName (see page 18960) property instead.

Note: When using InterceptGUID, the server must use the same COM object to undo the transformation performed by the

socket connection component's InterceptGUID. For servers that respond to TSocketConnection (see page 18928), you can set the server's InterceptGUID to the same GUID using the Socket server dialog of ScktSrvr.exe.

See Also

IDataIntercept (see page 18991)

InterceptName (see page 18960)

GetInterceptorList (see page 18954)

Specifying a Connection Using Sockets

130.1.10.5.3 TStreamedConnection.InterceptName Property

Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.

Pascal

```
property InterceptName: string;
```

C++

```
__property AnsiString InterceptName;
```

Description

Set InterceptName to the programmatic ID (ProgID) of a COM object that intercepts messages between the connection component and an application server before they are sent and after they are received.

Note: In Delphi, you can choose the ProgID of any COM objects that have been registered to perform this task at Design time, using a drop-down list in the Object Inspector, if they use TPacketInterceptFactory (see page 18925) as a class factory.

The COM object must implement the IDataIntercept (see page 18991) interface. Use the IDataIntercept (see page 18991) methods to manipulate the data that is communicated using the socket connection. For example, applications can use these methods to encrypt and decrypt data or to compress and decompress data.

Tip: To specify the COM object that intercepts messages using a GUID instead of a ProgID, use the InterceptGUID (see page 18959) property instead.

Note: When using InterceptName, the server must use the same COM object to undo the transformation performed by the socket connection component's InterceptName. For servers that respond to TSocketConnection (see page 18928), you can set the COM object that the server uses with the Socket server dialog of ScktSrvr.exe.

See Also

IDataIntercept (see page 18991)

InterceptGUID (see page 18959)

GetInterceptorList (see page 18954)

TPacketInterceptFactory (see page 18925)

Specifying a Connection Using Sockets

130.1.10.5.4 TStreamedConnection.Interpreter Property

Provides access to the helper object that marshals interface calls for the streamed connection.

Pascal

```
property Interpreter: TCustomDataBlockInterpreter;
```

C++

```
__property TCustomDataBlockInterpreter Interpreter;
```

Description

Streamed connection components use a data block interpreter internally to provide marshaling support.

See Also

Send (see page 18957)

130.1.10.5.5 TStreamedConnection.SupportCallbacks Property

Determines whether the streamed connection component can handle callbacks in the application server interface.

Pascal

```
property SupportCallbacks: Boolean;
```

C++

```
__property Boolean SupportCallbacks;
```

Description

Use SupportCallbacks to indicate whether the connection component can marshal callback functions in the application server's interface. When SupportCallbacks is true, the connection component can marshal calls from the application server to the client application over an interface supplied as a callback.

For TSocketConnection (see page 18928), callbacks are optional. Set SupportCallbacks to true if you are using callbacks as part of the application server's interface. When SupportCallbacks is true, the socket connection component requires WinSock2. Set SupportCallbacks to false if you are not using callback functions and want to eliminate the dependency on WinSock2.

TWebConnection (see page 18968) can't support callback functions. It initializes SupportCallbacks to false in the constructor and never changes this value.

See Also

Send (see page 18957)

Extending the Interface of the Application Server

130.1.11 TTransportThread Class

Class Hierarchy

Pascal

```
TTransportThread = class(TThread);
```

C++

```
class TTransportThread : public TThread;
```

File

SConnect

Description

This is class SConnect.TTransportThread.

130.1.11.1 TTransportThread Members

The following tables list the members exposed by TTransportThread.

Protected Methods

	Name	Description
	CheckThreadError (see page 3114)	This is the overview for the CheckThreadError method overload.
V	DoTerminate (see page 3115)	Generates an OnTerminate (see page 3148) event.

Protected Properties

	Name	Description
	ReturnValue (see page 3141)	Specifies the value returned to other waiting threads when the thread finishes executing.
R	Terminated (see page 3142)	Indicates whether the thread has been asked to terminate.

Public Events

	Name	Description
	OnTerminate (see page 3148)	Occurs after the thread's Execute (see page 3115) method has returned and before the thread is destroyed.

Public Methods

	Name	Description
V	AfterConstruction (see page 3114)	Executes the thread when the thread is created with CreateSuspended set to false.
V	Create (see page 18963)	This is Create, a member of class TTransportThread.
V	Destroy (see page 18963)	Destroys the thread object and releases the memory allocated to it.
V	Execute (see page 18963)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.
	Queue (see page 3119)	This is the overview for the Queue method overload.
	RemoveQueuedEvents (see page 3120)	Remove queued method calls.
	Resume (see page 3120)	Restarts the execution of a suspended thread.
	StaticQueue (see page 3124)	Asynchronously executes a method call within the main thread.
	StaticSynchronize (see page 3125)	Executes a method call within the main thread.
	Suspend (see page 3125)	Pauses a running thread.
	Synchronize (see page 3126)	This is the overview for the Synchronize method overload.
	Terminate (see page 3126)	Signals the thread to terminate by setting the Terminated (see page 3142) property to true.
	WaitFor (see page 3132)	Waits for the thread to terminate and then returns the value of the ReturnValue (see page 3141) property.

Public Properties

	Name	Description
R	FatalException (see page 3133)	Determines whether the thread object is automatically destroyed when the thread terminates.
	FreeOnTerminate (see page 3134)	Determines whether the thread object is automatically destroyed when the thread terminates.
R	Handle (see page 3137)	Indicates the thread's handle (Windows only).
	Priority (see page 3138)	Determines the thread's scheduling priority relative to other threads in the process.
R	Semaphore (see page 18967)	This is Semaphore, a member of class TTransportThread.
	Suspended (see page 3142)	Indicates whether a thread is suspended.
R	ThreadID (see page 3148)	Identifies the thread throughout the system.

130.1.11.2 TTransportThread.Create Constructor

Pascal

```
constructor Create(AHandle: THandle; Transport: ITransport); virtual;
```

C++

```
virtual __fastcall TTransportThread(THandle AHandle, ITransport Transport);
```

Description

This is Create, a member of class TTransportThread.

130.1.11.3 TTransportThread.Destroy Destructor

Destroys the thread object and releases the memory allocated to it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTransportThread();
```

Description

Do not call Destroy in an application. Instead, use Free to destroy a thread when FreeOnTerminate is False.

Destroy signals the thread to terminate and then waits for the thread to return before calling the inherited Destroy method.

See Also

Create (see page 3110)

Terminate (see page 3126)

WaitFor (see page 3132)

FreeOnTerminate (see page 3134)

Free (see page 22300)

130.1.11.4 TTransportThread Methods

The methods of the TTransportThread class are listed here.

Public Methods

	Name	Description
	Execute (see page 18963)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

130.1.11.4.1 TTransportThread.Execute Method

Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Pascal

```
procedure Execute; override;
```

C++

```
virtual __fastcall Execute();
```

Description

Override Execute and insert the code that should be executed when the thread runs. Execute is responsible for checking the value of the Terminated property to determine if the thread needs to exit.

A thread executes when Create is called if CreateSuspended set to false, or when Resume is first called after the thread is created if CreateSuspended set to true.

Note: Do not use the properties and methods of other objects directly in the Execute method of a thread. Instead, separate the use of other objects into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize method.

C++ Examples:

```
/*
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes. To
create the extension choose the
File | New | Other | Thread Object menu.
*/
bool mythreadRunning;
bool youthreadRunning;

class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
}

class TYouThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};

__fastcall TYouThread::TYouThread(bool suspended)
    : TThread(suspended)
{
}

void __fastcall TMyThread::Execute()
{
  Form1->Memo1->Lines->Add("Process has been running for this many seconds:");
  for (int I = 0; I <= 10; I++)
  {
```

```
    Form1->Memo1->Lines->Add("Low priority process " + IntToStr(I));
    Sleep(1000);
  }
  mythreadRunning = FALSE;
}

void __fastcall TYouThread::Execute()
{
  Form1->Memo3->Lines->Add("Second low priority process has been running for this many
seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo3->Lines->Add("Second low priority process " + IntToStr(I));
    Sleep(1000);
  }
  youthreadRunning = FALSE;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMyThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    SecondProcess = new TMyThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form1->Memo2->Lines->Add("Do some work in the main process for 10 seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo2->Lines->Add("Main process " + IntToStr(I));
    Sleep(1000);
  }
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  TYouThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    SecondProcess = new TYouThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  mythreadRunning = FALSE;
  youthreadRunning = FALSE;
}
```

Delphi Examples:

```
{
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes.  The
Execute procedure must be overridden in the extension class
of TThread or an "Abstract Error" will result. To create the
extension choose the File | New | Other | Thread Object menu.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (mythreadRunning = FALSE) then
  begin
    mythreadRunning:= TRUE;
    MyProcess := TMyThread.Create(True); { create suspended - secondprocess does not run yet }
    MyProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    MyProcess.Priority := tpLower;  // set the priority to lower than normal
    MyProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  if mythreadRunning then MyProcess.Terminate();
end;

procedure TMyThread.Execute;
var I: Integer;
begin
  Form1.Memo1.Lines.Add('Process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo1.Lines.Add('Low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
  mythreadRunning:= FALSE;
end;

procedure TYouThread.Execute;
var I: Integer;
begin
  Form1.Memo3.Lines.Add('Second low priority process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo3.Lines.Add('Second low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
    youthreadRunning:= FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
var   I: Integer;
begin
  Form1.Memo2.Lines.Add('Do some work in the main process for 10 seconds:');
  for I := 0 to 10 do
  begin
    Form1.Memo2.Lines.Add('Main process ' + InttoStr(I));
    Sleep(1000);
  end;
end;
```

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  if (youthreadRunning = FALSE) then
  begin
    youthreadRunning:= TRUE;
    YouProcess := TYouThread.Create(True); { create suspended – secondprocess does not run yet
}
    YouProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    YouProcess.Priority := tpLower;  // set the priority to lower than normal
    YouProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  if youthreadRunning then YouProcess.Terminate();
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  mythreadRunning:= FALSE;
  youthreadRunning:= FALSE;
end;
```

See Also

Create (see page 3110)

Terminated (see page 3142)

Synchronize (see page 3126)

Checking for Termination by Other Threads

Waiting for a Thread to Finish Executing

Writing Clean-up Code

Writing the Thread Function

130.1.11.5 TTransportThread Properties

The properties of the TTransportThread class are listed here.

Public Properties

	Name	Description
R	Semaphore (see page 18967)	This is Semaphore, a member of class TTransportThread.

130.1.11.5.1 TTransportThread.Semaphore Property

Pascal

```
property Semaphore: THandle;
```

C++

```
__property THandle Semaphore;
```

Description

This is Semaphore, a member of class TTransportThread.

130.1.12 TWebConnection Class

TWebConnection uses HTTP to manage the connection to an application server.

Class Hierarchy

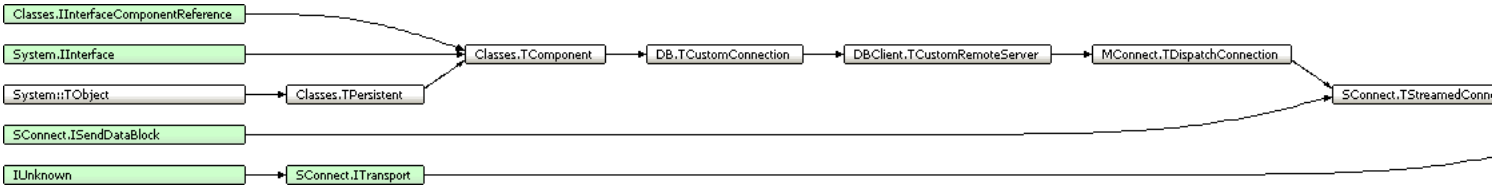

Pascal

```
TWebConnection = class(TStreamedConnection, ITransport);
```

C++

```
class TWebConnection : public TStreamedConnection, public ITransport;
```

File

SConnect

Description

TWebConnection is used in the client portion of a multi-tiered database application to establish and maintain the connection between the client and a remote application server. The Web connection object can

Establish an initial connection to a remote application server.

Obtain an IAppServer for the application server.

Obtain a list of providers on the application server.

Drop the connection to the remote application server.

TWebConnection establishes the initial connection between the client application and a remote application server using HTTP. To use TWebConnection, you must have the following:

Wininet.dll must be installed on the client system. Wininet.dll is found in the Windows system directory if you have IE3 or higher installed.

The server system must have IIS version 4 or better or Netscape Enterprise version 3.6 or better.

HTTPsrvr.dll must be installed with the Web server to which TWebConnection forms a connection. HTTPsrvr.dll launches the application server and marshals all calls to the application server interface.

Once the connection is established, the client registers any or all of its client datasets, and these client datasets use the IAppServer interface from the Web connection component to communicate with providers on the application server, or to otherwise call the application server's data module interface.

To connect to remote application servers using DCOM, use the TDCOMConnection component instead. To connect to remote application servers using TCP/IP, use the TSocketConnection (see page 18928) component instead. To connect to remote application servers using SOAP, use the TSoapConnection component.

See Also

TDataSetProvider (see page 18508)

TClientDataSet (see page 8289)

TDCOMConnection (see page 16008)

TRemoteDataModule (see page 6585)

IAppServer (see page 16450)

TSocketConnection (see page 18928)

TSoapConnection (see page 19864)

Extending the Interface of the Application Server

Specifying a Connection Using HTTP

130.1.12.1 TWebConnection Members

The following tables list the members exposed by TWebConnection.

Protected Events

	Name	Description
	OnGetUsername (see page 8533)	This is OnGetUsername, a member of class TCustomRemoteServer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_AddRef (see page 18952)	Increments an internal reference counter.
	_Release (see page 2769)	Called when an application releases a component interface.
	_Release (see page 18952)	Decrements an internal reference counter.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CreateTransport (see page 18974)	Provides the backstop for a method that obtains an ITransport interface.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoConnect (see page 18975)	Connects to an application server.
V	DoDisconnect (see page 6864)	Provides the interface for a method that terminates the connection.
V	DoDisconnect (see page 16023)	Drops a connection to an application server.
V	DoDisconnect (see page 18953)	Disconnects from an application server.
V	DoError (see page 18954)	raises an exception for a specified exception object.
V	GetAppServer (see page 8530)	Returns the value of the AppServer (see page 8532) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetConnected (see page 6865)	Returns the value of the Connected (see page 6868) property.
V	GetConnected (see page 16023)	Returns the value of the Connected (see page 16026) property.
V	GetDataSet (see page 6865)	Returns a dataset, given its index in the DataSets (see page 6870) property array.
V	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInterpreter (see page 18975)	Returns the value of the Interpreter property.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetServerCLSID (see page 16024)	Returns the GUID for the application server.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWaitEvent (see page 18976)	Returns the handle for an event object that is managed by the object that implements ITransport.
	InternalClose (see page 18956)	Terminates a connection to the application server.
	InternalOpen (see page 18956)	Opens a connection to the application server.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6866)	Initializes the connection component after all objects in the form or data module have been loaded.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 16025)	Responds when other objects are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	QueryInterface (see page 18956)	Implements the IUnknown QueryInterface method.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	Receive (see page 18976)	Returns an IDataBlock interface that can be used to read information received from the application server.
	RegisterClient (see page 6866)	Registers an object that uses the connection component.
	Send (see page 18977)	Sends information retrieved from an IDataBlock interface to the application server.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppServer (see page 8531)	Sets the AppServer (see page 8532) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnected (see page 18977)	Sets the Connected property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	ThreadException (see page 18957)	raises an exception for the thread that communicates with the application server.
	ThreadReceivedStream (see page 18957)	Handles callbacks from the application server.
	Transport_GetConnected (see page 18978)	This is Transport_GetConnected, a member of class TWebConnection.
	Transport_SetConnected (see page 18978)	This is Transport_SetConnected, a member of class TWebConnection.
	UnRegisterClient (see page 6868)	Removes a registered client so that it is no longer associated with the connection component.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WndProc (see page 18958)	Dispatches messages from the transport layer.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Handle (see page 18959)	Provides access to an internal window handle that can receive messages from the transport layer.
	InterceptGUID (see page 18959)	Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.
	InterceptName (see page 18960)	Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.
R	Interpreter (see page 18960)	Provides access to the helper object that marshals interface calls for the streamed connection.
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.
	SupportCallbacks (see page 18981)	Determines whether the streamed connection component can handle callbacks in the application server interface.

Public Events

	Name	Description
	AfterConnect (see page 6874)	Occurs after a connection is established.
	AfterDisconnect (see page 6875)	Occurs after the connection closes.
	BeforeConnect (see page 6875)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 6875)	Occurs immediately before the connection closes.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 6863)	Closes the connection.
V	Create (see page 18973)	Creates and initializes an instance of TWebConnection (see page 18968).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 18974)	Destroys an instance of TWebConnection (see page 18968).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.

	GetConnected (see page 18996)	Returns the value of the Connected (see page 18998) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetInterceptorList (see page 18954)	Retrieves a list of registered interceptors that can be used to transform messages between the connection component and the application server.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetProviderNames (see page 8530)	Executes a callback for all provider objects on the application server.
V	GetServer (see page 8530)	Returns the IAppServer interface of the application server.
V	GetServer (see page 16023)	Returns the IAppServer interface of the application server.
V	GetServerList (see page 8531)	Returns a list of application servers (remote data modules) to which the component can connect.
V	GetServerList (see page 16024)	Returns a list of remote data modules to which the component can connect.
V	GetServerList (see page 18955)	Retrieves a list of registered data brokers from the remote server system.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 6866)	Opens the connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AppServer (see page 8532)	Contains the interface for communicating with the application server.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Connected (see page 6868)	Determines whether a connection has been established to the remote source of data.
	Connected (see page 18998)	Indicates whether the connection to the application server is established.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 6870)	Lists the datasets currently using this connection component.

	DesignInfo (see page 2805)	Contains information used by the Form designer.
	LoginPrompt (see page 6872)	Specifies whether a login dialog appears immediately before opening a new connection.
	OnLogin (see page 6873)	Occurs when a communication channel opens to the server.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterConnect (see page 16031)	Occurs after a connection is established.
	AfterDisconnect (see page 16031)	Occurs after the connection closes.
	BeforeConnect (see page 16032)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 16032)	Occurs immediately before the connection closes.
	OnGetUsername (see page 16033)	This is OnGetUsername, a member of class TDispatchConnection.

Published Properties

	Name	Description
	Agent (see page 18979)	Provides the value of the User-Agent header for HTTP messages to the server.
	Connected (see page 16026)	Indicates whether the TDispatchConnection (see page 16017) object is connected to an application server.
	LoginPrompt (see page 16027)	Specifies whether a login dialog appears immediately before opening a new connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectBroker (see page 18979)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	OnLogin (see page 16028)	Occurs when a communication channel opens to the server.
	Password (see page 18980)	Specifies the password that allows the client application to log on to the application server.
	Proxy (see page 18980)	Lists any proxy servers that are required to make the connection.
	ProxyByPass (see page 18980)	Lists the host names of locally registered servers that do not require a proxy.
	ServerGUID (see page 16030)	Specifies the GUID of the application server with which to connect.
	ServerName (see page 16030)	Specifies the name of the application server with which to connect.
	Tag (see page 2808)	Stores an integer value as part of a component.
	URL (see page 18981)	Specifies the Uniform Resource Locator for httpsrvr.dll on the server system.
	UserName (see page 18982)	Specifies the user name that allows the client application to log on to the application server.

130.1.12.2 TWebConnection.Create Constructor

Creates and initializes an instance of TWebConnection (see page 18968).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebConnection(TComponent * AOwner);
```

Description

Call Create to instantiate a TWebConnection (see page 18968) object at runtime. Web connection components added to forms or data modules at design time are created automatically.

The AOwner parameter specifies the component (usually a form or data module) that is responsible for freeing the Web connection component.

See Also

Create (see page 18950)

Owner (see page 2807)

130.1.12.3 TWebConnection.Destroy Destructor

Destroys an instance of TWebConnection (see page 18968).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebConnection();
```

Description

Do not call Destroy directly in an application. Instead, use the Free method, which checks that the Web connection is not nil before calling Destroy.

130.1.12.4 TWebConnection Methods

The methods of the TWebConnection class are listed here.

Protected Methods

	Name	Description
	CreateTransport (see page 18974)	Provides the backstop for a method that obtains an ITransport interface.
	DoConnect (see page 18975)	Connects to an application server.
	GetInterpreter (see page 18975)	Returns the value of the Interpreter property.
	GetWaitEvent (see page 18976)	Returns the handle for an event object that is managed by the object that implements ITransport.
	Receive (see page 18976)	Returns an IDataBlock interface that can be used to read information received from the application server.
	Send (see page 18977)	Sends information retrieved from an IDataBlock interface to the application server.
	SetConnected (see page 18977)	Sets the Connected property.
	Transport_GetConnected (see page 18978)	This is Transport_GetConnected, a member of class TWebConnection.
	Transport_SetConnected (see page 18978)	This is Transport_SetConnected, a member of class TWebConnection.

130.1.12.4.1 TWebConnection.CreateTransport Method

Provides the backstop for a method that obtains an ITransport interface.

Pascal

```
function CreateTransport: ITransport; override;
```

C++

```
virtual __fastcall ITransport CreateTransport();
```

Description

Streamed connection components use an ITransport interface to connect to and communicate with the application server. As implemented in TStreamedConnection, CreateTransport always returns nil (Delphi) or NULL (C++).

Descendants of TStreamedConnection override CreateTransport to obtain an ITransport interface that connects to and communicates with the application server.

Component writers wanting to create connection components for specialized transport protocols need only override CreateTransport. Once the streamed connection has a working ITransport interface, all other functions of the connection component are handled automatically.

See Also

Send (see page 18957)

DoConnect (see page 18953)

DoDisconnect (see page 18953)

ThreadReceivedStream (see page 18957)

130.1.12.4.2 TWebConnection.DoConnect Method

Connects to an application server.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

DoConnect is called internally when the Connected property is set to true. DoConnect creates a separate thread for managing the connection so that communication with the application server does not block execution in the client application. Using the ITransport interface, DoConnect connects to the application server and sets the value of the AppServer property to its interface.

Note: Descendants of TStreamedConnection override CreateTransport rather than DoConnect to change the way the client application forms a connection to application servers.

See Also

AfterConnect (see page 6874)

BeforeConnect (see page 6875)

Connected (see page 16026)

AppServer (see page 8532)

CreateTransport (see page 18952)

130.1.12.4.3 TWebConnection.GetInterpreter Method

Returns the value of the Interpreter property.

Pascal

```
function GetInterpreter: TCustomDataBlockInterpreter; override;
```

C++

```
virtual __fastcall TCustomDataBlockInterpreter GetInterpreter();
```

Description

GetInterpreter is the protected read implementation of the Interpreter property. As implemented in TStreamedConnection, GetInterpreter instantiates a TDataBlockInterpreter object.

See Also

Interpreter (see page 18960)

TDataBlockInterpreter (see page 18913)

130.1.12.4.4 TWebConnection.GetWaitEvent Method

Returns the handle for an event object that is managed by the object that implements ITransport.

Pascal

```
function GetWaitEvent: THandle; stdcall;
```

C++

```
__fastcall __stdcall THandle GetWaitEvent();
```

Description

Streamed connection components perform all reading from and writing to the application server using a separate execution thread. This limits the impact that communication with the application server has on the speed of the client application, because execution does not need to wait while messages are marshaled and sent or received. The streamed connection component needs to coordinate its use of the communication thread. To do this, it uses an event object that is a member of the object that implements ITransport.

GetWaitEvent returns a handle to this event object. When implementing an object that supports ITransport, create a TEvent object as a member and return its handle as the return value of GetWaitEvent. Do not use the event object for internal purposes or it will interfere with the streamed connection's use of the event.

See Also

TEvent (see page 22063)

130.1.12.4.5 TWebConnection.Receive Method

Returns an IDataBlock interface that can be used to read information received from the application server.

Pascal

```
function Receive(WaitForInput: Boolean; Context: Integer): IDataBlock; stdcall;
```

C++

```
__fastcall __stdcall IDataBlock Receive(Boolean WaitForInput, int Context);
```

Description

Receive creates a COM object that implements the IDataBlock interface and reads information from the application server into its Memory property.

If WaitForInput is true, Receive halts execution until the application server sends the expected information. If WaitForInput is false, Receive can fail (reading no information) if the application server does not send any information immediately (or after a brief time-out period).

Context is transport-specific data that can be used when implementing a class that supports ITransport. Its value is the return value of the Send method.

See Also

Memory (see page 18989)

Send (see page 18997)

130.1.12.4.6 TWebConnection.Send Method

Sends information retrieved from an IDataBlock interface to the application server.

Pascal

```
function Send(const Data: IDataBlock): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Send(const IDataBlock Data);
```

Description

Send sends the entire contents of the Data parameter's Memory property to the application server. It returns a transport-specific value that is used by the next call to the Receive method. For example, HTTP transport classes return the handle for the Internet connection.

See Also

Memory (see page 18989)

Receive (see page 18997)

130.1.12.4.7 TWebConnection.SetConnected Method

Sets the Connected property.

Pascal

```
procedure SetConnected(Value: Boolean); override;
```

C++

```
virtual __fastcall SetConnected(Boolean Value);
```

Description

SetConnected is the protected implementation of the Connected property. It establishes a connection to the Application server if Value is true, and disconnects from the application server if Value is false.

If Value is false, SetConnected generates a BeforeDisconnect event, closes the connection, and generates an AfterDisconnect event.

If Value is true, SetConnected performs the following actions:

1. If LoginPrompt is true, and the connection is not already open, SetConnected generates an OnGetUsername event, and uses that to initialize a login dialog. It displays the login dialog, and aborts the connection if the user cancels.
2. It generates a BeforeConnect event.
3. It opens the connection.
4. It generates an AfterConnect event.
5. If LoginPrompt is true, and the connection opened successfully, SetConnected generates an OnLogin event.

See Also

Connected (see page 16026)

BeforeConnect (see page 6875)

AfterConnect (see page 6874)

BeforeDisconnect (see page 6875)

AfterDisconnect (see page 6875)

LoginPrompt (see page 6872)

OnGetUserName (see page 8533)

OnLogin (see page 6873)

130.1.12.4.8 TWebConnection.Transport_GetConnected Method

Pascal

```
function Transport_GetConnected: Boolean; stdcall;
```

C++

```
__fastcall __stdcall Boolean Transport_GetConnected();
```

Description

This is Transport_GetConnected, a member of class TWebConnection.

130.1.12.4.9 TWebConnection.Transport_SetConnected Method

Pascal

```
procedure Transport_SetConnected(Value: Boolean); stdcall;
```

C++

```
__fastcall __stdcall Transport_SetConnected(Boolean Value);
```

Description

This is Transport_SetConnected, a member of class TWebConnection.

130.1.12.5 TWebConnection Properties

The properties of the TWebConnection class are listed here.

Protected Properties

	Name	Description
	SupportCallbacks (see page 18981)	Determines whether the streamed connection component can handle callbacks in the application server interface.

Published Properties

	Name	Description
	Agent (see page 18979)	Provides the value of the User-Agent header for HTTP messages to the server.
	ObjectBroker (see page 18979)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	Password (see page 18980)	Specifies the password that allows the client application to log on to the application server.
	Proxy (see page 18980)	Lists any proxy servers that are required to make the connection.
	ProxyByPass (see page 18980)	Lists the host names of locally registered servers that do not require a proxy.
	URL (see page 18981)	Specifies the Uniform Resource Locator for httpsrvr.dll on the server system.
	UserName (see page 18982)	Specifies the user name that allows the client application to log on to the application server.

130.1.12.5.1 TWebConnection.Agent Property

Provides the value of the User-Agent header for HTTP messages to the server.

Pascal

```
property Agent: string;
```

C++

```
__property AnsiString Agent;
```

Description

Use Agent to supply HTTP messages generated by the Web connection component with the name of your application. This name is included as a User-Agent header. The User-Agent header is an optional string that describes the application generating the HTTP message.

See Also

Title (see page 13723)

130.1.12.5.2 TWebConnection.ObjectBroker Property

Specifies an Object Broker that allows the connection component to choose from several application servers.

Pascal

```
property ObjectBroker: TCustomObjectBroker;
```

C++

```
__property TCustomObjectBroker ObjectBroker;
```

Description

Set ObjectBroker to use a broker that maintains a list of servers that support the application specified by ServerName or ServerGUID. When the connection component needs to open a connection, it requests information to identify a server machine from the Object Broker.

Descendants of TDispatchConnection define properties that specify how to locate server machines. These properties are not saved with the connection component if the ObjectBroker property is set. Instead, the Object Broker provides this information dynamically at runtime from its list of available servers.

Applications can use TSimpleObjectBroker for an object broker, or derive their own object brokers from TCustomObjectBroker.

See Also

DoConnect (see page 6864)

ServerName (see page 16030)

ServerGUID (see page 16030)

TSimpleObjectBroker (see page 17877)

Brokering Connections

Specifying a Connection Using HTTP

Specifying a Connection Using DCOM

Specifying a Connection Using Sockets

Using TSimpleObjectBroker

130.1.12.5.3 TWebConnection.Password Property

Specifies the password that allows the client application to log on to the application server.

Pascal

```
property Password: string;
```

C++

```
__property AnsiString Password;
```

Description

When the Web connection component connects to the application server, it supplies the values of the UserName (see page 18982) and Password properties so that it can log on to the host or proxy. The value of Password can be an empty string if neither the host nor proxy requires authentication.

See Also

UserName (see page 18982)

Specifying a Connection Using HTTP

130.1.12.5.4 TWebConnection.Proxy Property

Lists any proxy servers that are required to make the connection.

Pascal

```
property Proxy: string;
```

C++

```
__property AnsiString Proxy;
```

Description

Use Proxy if the Web connection component can't resolve the value of Host locally. If the server is registered locally, Proxy can be a null string. Otherwise, Proxy lists the host names of any proxy servers that can forward request messages on to the intended server machine.

See Also

ProxyByPass (see page 18980)

130.1.12.5.5 TWebConnection.ProxyByPass Property

Lists the host names of locally registered servers that do not require a proxy.

Pascal

```
property ProxyByPass: string;
```

C++

```
__property AnsiString ProxyByPass;
```

Description

Use ProxyByPass to indicate any server systems to which the Web connection component can make a direct connection. When the Host property specifies a server in this list, the Web connection component makes a direct connection even if the Proxy (see page 18980) property is specified.

When ProxyByPass is an empty string, the Web connection component checks the registry for any locally known servers and

bypasses the proxy server if Host refers to a registered local server.

See Also

Proxy (see page 18980)

130.1.12.5.6 TWebConnection.SupportCallbacks Property

Determines whether the streamed connection component can handle callbacks in the application server interface.

Pascal

```
property SupportCallbacks: Boolean;
```

C++

```
__property Boolean SupportCallbacks;
```

Description

Use SupportCallbacks to indicate whether the connection component can marshal callback functions in the application server's interface. When SupportCallbacks is true, the connection component can marshal calls from the application server to the client application over an interface supplied as a callback.

For TSocketConnection, callbacks are optional. Set SupportCallbacks to true if you are using callbacks as part of the application server's interface. When SupportCallbacks is true, the socket connection component requires WinSock2. Set SupportCallbacks to false if you are not using callback functions and want to eliminate the dependency on WinSock2.

TWebConnection can't support callback functions. It initializes SupportCallbacks to false in the constructor and never changes this value.

See Also

Send (see page 18957)

Extending the Interface of the Application Server

130.1.12.5.7 TWebConnection.URL Property

Specifies the Uniform Resource Locator for httpsrvr.dll on the server system.

Pascal

```
property URL: string;
```

C++

```
__property AnsiString URL;
```

Description

Use URL to specify the Uniform Resource Locator that the Web connection component uses to locate httpsrvr.dll. This DLL, which ships in the BIN directory, is deployed on the same machine as the Web server so that it can accept HTTP request messages and convert them into COM calls that communicate with the application server.

URL includes the protocol (http or https), host name, and scriptname for httpsrvr.dll. Typically, it has the form

```
http://DataHost.org/scripts/httpsrvr.dll
```

Note: URL is a standard Uniform Resource Locator. As such, you can also use it to override the port associated with http (80) or https (443) if the Web server is configured differently. The port number follows the host name, preceded by a colon.

See Also

Specifying a Connection Using HTTP

130.1.12.5.8 TWebConnection.UserName Property

Specifies the user name that allows the client application to log on to the application server.

Pascal

```
property UserName: string;
```

C++

```
__property AnsiString UserName;
```

Description

When the Web connection component connects to the application server on a local host or to the proxy server, it supplies the values of the UserName and Password (see page 18980) properties so that it can log in. If neither the host nor the proxy server requires authentication, UserName can be an empty string.

See Also

Password (see page 18980)

Specifying a Connection Using HTTP

130.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IDataBlock (see page 18983)	IDataBlock is the interface used by data block interpreters to access memory buffers.
	IDataIntercept (see page 18991)	IDataIntercept is the interface used by TSocketConnection (see page 18928) components to transform data that is communicated over the connection.
	ISendDataBlock (see page 18993)	ISendDataBlock is the interface used by data block interpreters to send messages to an application server after they are marshaled.
	ITransport (see page 18995)	ITransport is the interface used by TStreamedConnection (see page 18945) to connect to, read from, and write to a remote application server.

130.2.1 IDataBlock Interface

IDataBlock is the interface used by data block interpreters to access memory buffers.

Class Hierarchy

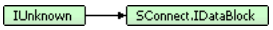

Pascal

```
IDataBlock = interface(IUnknown);
```

C++

```
__interface IDataBlock : public IUnknown;
```

File

SConnect

Description

Data block interpreters uses IDataBlock when marshaling COM interface calls. Streamed connection components send data obtained from IDataBlock to an application server once the data block interpreter has filled its memory by marshaling interface calls. They also read data from the application server into IDataBlock and pass it to a data block interpreter for marshaling.

IDataBlock is an interface, and as such can't be directly instantiated. Instead, objects that implement the IDataBlock properties and methods include IDataBlock in their class definition (Delphi) or be derived from IDataBlock (C++). Such objects can be cast to an IDataBlock so that they can be used by data block interpreters.

Note: C++ method declarations that use IDataBlock use the _di_IDataBlock type instead. This is a DelphiInterface wrapper around the IDataBlock interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IDataBlock >  _di_IDataBlock;
```

See Also

TCustomDataBlockInterpreter (see page 18898)

TStreamedConnection (see page 18945)

130.2.1.1 IDataBlock Members

The following tables list the members exposed by IDataBlock.

Public Methods

	Name	Description
	Clear (see page 18984)	Removes all data from Memory (see page 18989) except for the reserved bytes.
	GetBytesReserved (see page 18985)	Returns the value of the BytesReserved (see page 18989) property.
	GetMemory (see page 18985)	Returns the value of the Memory (see page 18989) property.
	GetSignature (see page 18985)	Returns the value of the Signature (see page 18990) property.
	GetSize (see page 18986)	Returns the value of the Size (see page 18990) property.
	GetStream (see page 18986)	Returns the value of the Stream (see page 18991) property.
	IgnoreStream (see page 18986)	Informs the data block when its stream is invalidated.
	InitData (see page 18987)	Adds data to the data block.
	Read (see page 18987)	Reads a specified number of bytes into a buffer.
	SetSignature (see page 18988)	Sets the value of the Signature (see page 18990) property.
	SetSize (see page 18988)	Sets the value of the Size (see page 18990) property.
	Write (see page 18988)	Writes a specified number of bytes from a buffer.

Public Properties

	Name	Description
R	BytesReserved (see page 18989)	Indicates the number of bytes from Memory (see page 18989) that are reserved for internal use.
R	Memory (see page 18989)	Points to a buffer from which data is read or to which data is written.
	Signature (see page 18990)	Indicates the contents of the data block.
	Size (see page 18990)	Indicates the size of the data in the memory buffer.
R	Stream (see page 18991)	Provides access to an internal stream that contains the data block's memory.

130.2.1.2 IDataBlock Methods

The methods of the IDataBlock class are listed here.

Public Methods

	Name	Description
	Clear (see page 18984)	Removes all data from Memory (see page 18989) except for the reserved bytes.
	GetBytesReserved (see page 18985)	Returns the value of the BytesReserved (see page 18989) property.
	GetMemory (see page 18985)	Returns the value of the Memory (see page 18989) property.
	GetSignature (see page 18985)	Returns the value of the Signature (see page 18990) property.
	GetSize (see page 18986)	Returns the value of the Size (see page 18990) property.
	GetStream (see page 18986)	Returns the value of the Stream (see page 18991) property.
	IgnoreStream (see page 18986)	Informs the data block when its stream is invalidated.
	InitData (see page 18987)	Adds data to the data block.
	Read (see page 18987)	Reads a specified number of bytes into a buffer.
	SetSignature (see page 18988)	Sets the value of the Signature (see page 18990) property.
	SetSize (see page 18988)	Sets the value of the Size (see page 18990) property.
	Write (see page 18988)	Writes a specified number of bytes from a buffer.

130.2.1.2.1 IDataBlock.Clear Method

Removes all data from Memory (see page 18989) except for the reserved bytes.

Pascal

```
procedure Clear; stdcall;
```

C++

```
__fastcall __stdcall Clear();
```

Description

Call Clear when finished reading from Memory (see page 18989) or when the information written to Memory (see page 18989) has been used. Clear reclaims the resources used to store information that is read from or written to the data block.

See Also

BytesReserved (see page 18989)

Memory (see page 18989)

130.2.1.2.2 IDataBlock.GetBytesReserved Method

Returns the value of the BytesReserved (see page 18989) property.

Pascal

```
function GetBytesReserved: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetBytesReserved();
```

Description

GetBytesReserved is the read implementer for the BytesReserved (see page 18989) property.

See Also

BytesReserved (see page 18989)

130.2.1.2.3 IDataBlock.GetMemory Method

Returns the value of the Memory (see page 18989) property.

Pascal

```
function GetMemory: Pointer; stdcall;
```

C++

```
__fastcall __stdcall void * GetMemory();
```

Description

GetMemory is the read implementer for the Memory (see page 18989) property.

See Also

Memory (see page 18989)

130.2.1.2.4 IDataBlock.GetSignature Method

Returns the value of the Signature (see page 18990) property.

Pascal

```
function GetSignature: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetSignature();
```

Description

GetSignature is the read implementer for the Signature (see page 18990) property.

See Also

Signature (see page 18990)

130.2.1.2.5 IDataBlock.GetSize Method

Returns the value of the Size (see page 18990) property.

Pascal

```
function GetSize: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetSize();
```

Description

GetSize is the read implementer for the Size (see page 18990) property.

See Also

Size (see page 18990)

130.2.1.2.6 IDataBlock.GetStream Method

Returns the value of the Stream (see page 18991) property.

Pascal

```
function GetStream: TStream; stdcall;
```

C++

```
__fastcall __stdcall TStream GetStream();
```

Description

GetStream is the read implementer for the Stream (see page 18991) property.

See Also

Stream (see page 18991)

130.2.1.2.7 IDataBlock.IgnoreStream Method

Informs the data block when its stream is invalidated.

Pascal

```
procedure IgnoreStream; stdcall;
```

C++

```
__fastcall __stdcall IgnoreStream();
```

Description

Call IgnoreStream after taking responsibility for freeing the component specified by Stream (see page 18991). After a call to

IgnoreStream, the data block will not free the memory for its internal Stream (see page 18991) component.

See Also

Stream (see page 18991)

130.2.1.2.8 IDataBlock.InitData Method

Adds data to the data block.

Pascal

```
function InitData(Data: Pointer; DataLen: Integer; CheckLen: Boolean): Integer; stdcall;
```

C++

```
__fastcall __stdcall int InitData(void * Data, int DataLen, Boolean CheckLen);
```

Description

Data is the data to be added to the data block. It must include a valid signature or InitData raises an exception.

DataLen is the number of bytes referenced by Data.

CheckLen specifies whether InitData should verify that the signature indicates the same amount of data as is indicated by the DataLen parameter.

See Also

Signature (see page 18990)

130.2.1.2.9 IDataBlock.Read Method

Reads a specified number of bytes into a buffer.

Pascal

```
function Read(var Buffer; Count: Integer): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Read( Buffer, int Count);
```

Description

Call Read to copy the contents stored in the memory buffer to the buffer supplied by the Buffer parameter. Read reads up to Count bytes and returns the number of bytes actually read.

The first call to Read begins reading from Memory (see page 18989) at the first position after the reserved bytes. Subsequent calls begin reading at the point where the last call to Read left off.

Note: The Count parameter should specify a value less than or equal to Size (see page 18990) - BytesReserved (see page 18989).

See Also

Write (see page 18988)

Memory (see page 18989)

Size (see page 18990)

BytesReserved (see page 18989)

130.2.1.2.10 IDataBlock.SetSignature Method

Sets the value of the Signature (see page 18990) property.

Pascal

```
procedure SetSignature(Value: Integer); stdcall;
```

C++

```
__fastcall __stdcall SetSignature(int Value);
```

Description

SetSignature is the write implementer for the Signature (see page 18990) property.

See Also

Signature (see page 18990)

130.2.1.2.11 IDataBlock.SetSize Method

Sets the value of the Size (see page 18990) property.

Pascal

```
procedure SetSize(Value: Integer); stdcall;
```

C++

```
__fastcall __stdcall SetSize(int Value);
```

Description

SetSize is the write implementer for the Size (see page 18990) property.

See Also

Size (see page 18990)

130.2.1.2.12 IDataBlock.Write Method

Writes a specified number of bytes from a buffer.

Pascal

```
function Write(const Buffer; Count: Integer): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Write(const  Buffer, int Count);
```

Description

Call Write to copy the contents of the buffer supplied as the Buffer parameter to the next position in Memory (see page 18989). Write writes up to Count bytes and returns the number of bytes actually written.

The first call to Write begins writing to Memory (see page 18989) at the first position after the reserved bytes. Subsequent calls begin writing at the point where the last call to Write left off.

Note: The Count parameter should specify a value less than or equal to Size (see page 18990) - BytesReserved (see page 18989).

See Also

Read (see page 18987)

Memory (see page 18989)

Size (see page 18990)

BytesReserved (see page 18989)

130.2.1.3 IDataBlock Properties

The properties of the IDataBlock class are listed here.

Public Properties

	Name	Description
R	BytesReserved (see page 18989)	Indicates the number of bytes from Memory (see page 18989) that are reserved for internal use.
R	Memory (see page 18989)	Points to a buffer from which data is read or to which data is written.
	Signature (see page 18990)	Indicates the contents of the data block.
	Size (see page 18990)	Indicates the size of the data in the memory buffer.
R	Stream (see page 18991)	Provides access to an internal stream that contains the data block's memory.

130.2.1.3.1 IDataBlock.BytesReserved Property

Indicates the number of bytes from Memory (see page 18989) that are reserved for internal use.

Pascal

```
property BytesReserved: Integer;
```

C++

```
__property int BytesReserved;
```

Description

BytesReserved indicates how many bytes of Memory (see page 18989) can't be used for reading and writing. When interpreting the value of the Size (see page 18990) property, use BytesReserved to determine how much buffer space is available for reading and writing data.

See Also

Memory (see page 18989)

Size (see page 18990)

130.2.1.3.2 IDataBlock.Memory Property

Points to a buffer from which data is read or to which data is written.

Pascal

```
property Memory: Pointer;
```

C++

```
__property void * Memory;
```

Description

Memory provides access to the buffer that contains data that is read or written. Memory includes a reserved area (of size BytesReserved (see page 18989)) for internal information such as the value of Signature (see page 18990),

Note: When implementing the Receive and Send methods of the ITransport (see page 18995) interface, component writers should read directly from and write directly to Memory, rather than using the Read (see page 18987) and Write (see page

18988) methods. Otherwise, the information in the reserved area will not be sent to or from the application server.

See Also

Size (see page 18990)

ITransport (see page 18995)

BytesReserved (see page 18989)

Signature (see page 18990)

Stream (see page 18991)

130.2.1.3.3 IDataBlock.Signature Property

Indicates the contents of the data block.

Pascal

```
property Signature: Integer;
```

C++

```
__property int Signature;
```

Description

Write (see page 18988) Signature values to indicate what type of data is written to the data block. Read (see page 18987) Signature to determine what type of data can be read from the data block.

The value of signature is stored somewhere in the first BytesReserved (see page 18989) bytes of Memory (see page 18989).

See Also

BytesReserved (see page 18989)

Memory (see page 18989)

130.2.1.3.4 IDataBlock.Size Property

Indicates the size of the data in the memory buffer.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

Size is the number of bytes available in Memory (see page 18989), excluding the reserved bytes. Set Size to ensure that Memory (see page 18989) is large enough to contain the data that the data block will read or write.

See Also

BytesReserved (see page 18989)

Memory (see page 18989)

Signature (see page 18990)

130.2.1.3.5 IDataBlock.Stream Property

Provides access to an internal stream that contains the data block's memory.

Pascal

```
property Stream: TStream;
```

C++

```
__property TStream Stream;
```

Description

Stream allows component writers more convenient access to the data block's memory buffer. This memory buffer contains the data that is read or written.

See Also

Memory (see page 18989)

IgnoreStream (see page 18986)

TMemoryStream (see page 2902)

130.2.2 IDataIntercept Interface

IDataIntercept is the interface used by TSocketConnection (see page 18928) components to transform data that is communicated over the connection.

Class Hierarchy

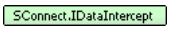

Pascal

```
IDataIntercept = interface;
```

C++

```
__interface IDataIntercept;
```

File

SConnect

Description

IDataIntercept is the interface implemented by data packet interceptors. A data packet interceptor is a COM object that implements IDataIntercept to transform data that is accessed through an IDataBlock (see page 18983) interface. TSocketConnection (see page 18928) can use a data packet interceptor before messages are sent and after responses are received.

Developers can create a data packet interceptor to encrypt and decrypt data or to compress and decompress messages. The transformation must leave the reserved bytes of the data blocks unchanged or the socket connection component won't be able to communicate properly.

Note: When creating a data packet interceptor in Delphi, use TPacketInterceptFactory (see page 18925) as its class factory.

Once you have created and registered a data packet interceptor, Set the InterceptName or InterceptGUID property of a socket connection component to identify the data packet interceptor. This tells the socket connection to instantiate the interceptor and use it to transform messages before they are sent and after they are received. On the server side, right click on the Tray Icon for

scktsrvr.exe and assign the interceptor using the properties dialog.

See Also

IDataBlock (see page 18983)

TStreamedConnection (see page 18945)

TStreamedConnection (see page 18945)

TPacketInterceptFactory (see page 18925)

Specifying a Connection Using Sockets

130.2.2.1 IDataIntercept Members

The following tables list the members exposed by IDataIntercept.

Public Methods

	Name	Description
	DataIn (see page 18992)	Undoes the transformation applied by the DataOut (see page 18993) method.
	DataOut (see page 18993)	Returns the value of the BytesReserved property.

130.2.2.2 IDataIntercept Methods

The methods of the IDataIntercept class are listed here.

Public Methods

	Name	Description
	DataIn (see page 18992)	Undoes the transformation applied by the DataOut (see page 18993) method.
	DataOut (see page 18993)	Returns the value of the BytesReserved property.

130.2.2.2.1 IDataIntercept.DataIn Method

Undoes the transformation applied by the DataOut (see page 18993) method.

Pascal

```
procedure DataIn(const Data: IDataBlock); stdcall;
```

C++

```
__fastcall __stdcall DataIn(const IDataBlock Data);
```

Description

Call DataIn to undo any transformations that were applied to information received by the connection. For example, use this method to decrypt or decompress data.

The Data parameter is an IDataBlock (see page 18983) interface to the data that was received by the socket connection component. Use the properties and methods of IDataBlock (see page 18983) to reverse the transformation that was applied by the DataOut (see page 18993) method.

Warning: Do not transform the reserved bytes in the data block.

See Also

DataOut (see page 18993)

BytesReserved (see page 18989)

130.2.2.2.2 IDataIntercept.DataOut Method

Returns the value of the BytesReserved property.

Pascal

```
procedure DataOut(const Data: IDataBlock); stdcall;
```

C++

```
__fastcall __stdcall DataOut(const IDataBlock Data);
```

Description

Call DataOut to transform data before it is sent using a connection component. For example, use this method to encrypt or compress data.

The Data parameter is an IDataBlock (see page 18983) interface to the data that is about to be sent by the socket connection component. Use the properties and methods of IDataBlock (see page 18983) to transform the data.

Warning: Do not transform the reserved bytes in the data block.

See Also

DataIn (see page 18992)

BytesReserved (see page 18989)

130.2.3 ISendDataBlock Interface

ISendDataBlock is the interface used by data block interpreters to send messages to an application server after they are marshaled.

Class Hierarchy

SConnect.ISendDataBlock

Pascal

```
ISendDataBlock = interface;
```

C++

```
__interface ISendDataBlock;
```

File

SConnect

Description

Streamed connection components implement ISendDataBlock to allow a data block interpreter (which handles marshaling of COM interface calls) to send interface calls to the application server.

ISendDataBlock is an interface, and as such can't be directly instantiated. Instead, streamed connection components (which implement ISendDataBlock) can be cast to an ISendDataBlock (Delphi) or have a conversion operator to obtain this interface (C++).

Note: TDataBlockInterpreter (see page 18913) uses _di_ISendDataBlock in place of ISendDataBlock. This is a DelphiInterface wrapper around the ISendDataBlock interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< ISendDataBlock > _di_ISendDataBlock;
```

See Also

TStreamedConnection (see page 18945)

TCustomDataBlockInterpreter (see page 18898)

IDataBlock (see page 18983)

130.2.3.1 ISendDataBlock Members

The following tables list the members exposed by ISendDataBlock.

Public Methods

	Name	Description
	Send (see page 18994)	Sends information retrieved from an IDataBlock (see page 18983) interface to the application server.

130.2.3.2 ISendDataBlock Methods

The methods of the ISendDataBlock class are listed here.

Public Methods

	Name	Description
	Send (see page 18994)	Sends information retrieved from an IDataBlock (see page 18983) interface to the application server.

130.2.3.2.1 ISendDataBlock.Send Method

Sends information retrieved from an IDataBlock (see page 18983) interface to the application server.

Pascal

```
function Send(const Data: IDataBlock; WaitForResult: Boolean): IDataBlock; stdcall;
```

C++

```
__fastcall __stdcall IDataBlock Send(const IDataBlock Data, Boolean WaitForResult);
```

Description

ISendDataBlock (see page 18993) dispatches information to an application server. The information must be available as the Memory property of an IDataBlock (see page 18983) interface.

The WaitForResult parameter indicates whether the caller requires a response when the information is sent. If WaitForResult is true, Send does not return until a response is received from the application server. This response is returned as an IDataBlock (see page 18983) interface (_di_IDataBlock). If WaitForResult is false, Send may still return a response, but only if the application server responds immediately.

See Also

Memory (see page 18989)

Send (see page 18957)

130.2.4 ITransport Interface

ITransport is the interface used by TStreamedConnection (see page 18945) to connect to, read from, and write to a remote application server.

Class Hierarchy

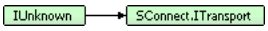

Pascal

```
ITransport = interface(IUnknown);
```

C++

```
__interface ITransport : public IUnknown;
```

File

SConnect

Description

Streamed connection components use ITransport to connect to an application server and to send or receive data over that connection. ITransport is responsible only for managing the communication protocols. It does not provide any support for marshaling or storing information.

ITransport is an interface, and as such can't be directly instantiated. Instead, objects that implement the ITransport properties and methods include ITransport in their class definition (Delphi) or are derived from ITransport (C++). Such objects can be cast to an ITransport so that they can be used by streamed connection components.

Note: C++ method declarations that use ITransport use the _di_ITransport type instead. This is a DelphiInterface wrapper around the ITransport interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< ITransport> _di_ITransport;
```

See Also

TStreamedConnection (see page 18945)

130.2.4.1 ITransport Members

The following tables list the members exposed by ITransport.

Public Methods

	Name	Description
	GetConnected (see page 18996)	Returns the value of the Connected (see page 18998) property.
	GetWaitEvent (see page 18996)	Returns the handle for an event object that is managed by the object that implements ITransport (see page 18995).
	Receive (see page 18997)	Returns an IDataBlock (see page 18983) interface that can be used to read information received from the application server.
	Send (see page 18997)	Sends information retrieved from an IDataBlock (see page 18983) interface to the application server.
	SetConnected (see page 18997)	Sets the value of the Connected (see page 18998) property.

Public Properties

	Name	Description
	Connected (see page 18998)	Indicates whether the connection to the application server is established.

130.2.4.2 ITransport Methods

The methods of the ITransport class are listed here.

Public Methods

	Name	Description
	GetConnected (see page 18996)	Returns the value of the Connected (see page 18998) property.
	GetWaitEvent (see page 18996)	Returns the handle for an event object that is managed by the object that implements ITransport (see page 18995).
	Receive (see page 18997)	Returns an IDataBlock (see page 18983) interface that can be used to read information received from the application server.
	Send (see page 18997)	Sends information retrieved from an IDataBlock (see page 18983) interface to the application server.
	SetConnected (see page 18997)	Sets the value of the Connected (see page 18998) property.

130.2.4.2.1 ITransport.GetConnected Method

Returns the value of the Connected (see page 18998) property.

Pascal

```
function GetConnected: Boolean; stdcall;
```

C++

```
__fastcall __stdcall Boolean GetConnected();
```

Description

GetConnected is the read property implementer for the Connected (see page 18998) property.

See Also

Connected (see page 18998)

130.2.4.2.2 ITransport.GetWaitEvent Method

Returns the handle for an event object that is managed by the object that implements ITransport (see page 18995).

Pascal

```
function GetWaitEvent: THandle; stdcall;
```

C++

```
__fastcall __stdcall THandle GetWaitEvent();
```

Description

Streamed connection components perform all reading from and writing to the application server using a separate execution thread. This limits the impact that communication with the application server has on the speed of the client application, because execution does not need to wait while messages are marshaled and sent or received. The streamed connection component needs to coordinate its use of the communication thread. To do this, it uses an event object that is a member of the object that implements ITransport (see page 18995).

GetWaitEvent returns a handle to this event object. When implementing an object that supports ITransport (see page 18995),

create a TEvent object as a member and return its handle as the return value of GetWaitEvent. Do not use the event object for internal purposes or it will interfere with the streamed connection's use of the event.

See Also

TEvent (see page 22063)

130.2.4.2.3 ITransport.Receive Method

Returns an IDataBlock (see page 18983) interface that can be used to read information received from the application server.

Pascal

```
function Receive(WaitForInput: Boolean; Context: Integer): IDataBlock; stdcall;
```

C++

```
__fastcall __stdcall IDataBlock Receive(Boolean WaitForInput, int Context);
```

Description

Receive creates a COM object that implements the IDataBlock (see page 18983) interface and reads information from the application server into its Memory property.

If WaitForInput is true, Receive halts execution until the application server sends the expected information. If WaitForInput is false, Receive can fail (reading no information) if the application server does not send any information immediately (or after a brief time-out period).

Context is transport-specific data that can be used when implementing a class that supports ITransport (see page 18995). Its value is the return value of the Send (see page 18997) method.

See Also

Memory (see page 18989)

Send (see page 18997)

130.2.4.2.4 ITransport.Send Method

Sends information retrieved from an IDataBlock (see page 18983) interface to the application server.

Pascal

```
function Send(const Data: IDataBlock): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Send(const IDataBlock Data);
```

Description

Send sends the entire contents of the Data parameter's Memory property to the application server. It returns a transport-specific value that is used by the next call to the Receive (see page 18997) method. For example, HTTP transport classes return the handle for the Internet connection.

See Also

Memory (see page 18989)

Receive (see page 18997)

130.2.4.2.5 ITransport.SetConnected Method

Sets the value of the Connected (see page 18998) property.

Pascal

```
procedure SetConnected(Value: Boolean); stdcall;
```

C++

```
__fastcall __stdcall SetConnected(Boolean Value);
```

Description

SetConnected is the write implementer for the Connected (see page 18998) property.

130.2.4.3 ITransport Properties

The properties of the ITransport class are listed here.

Public Properties

	Name	Description
	Connected (see page 18998)	Indicates whether the connection to the application server is established.

130.2.4.3.1 ITransport.Connected Property

Indicates whether the connection to the application server is established.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Set Connected to true to connect to the application server. Set Connected to false to close a connection.

Read the Connected property to verify that the connection is open before reading from or writing to the application server.

See Also

Receive (see page 18997)

Send (see page 18997)

130.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetPacketInterceptorList (see page 18999)	This is function SConnect.GetPacketInterceptorList.
	LoadWinSock2 (see page 18999)	This is function SConnect.LoadWinSock2.

130.3.1 SConnect.GetPacketInterceptorList Function

Pascal

```
procedure GetPacketInterceptorList(List: TStringList);
```

C++

```
GetPacketInterceptorList(TStringList List);
```

File

SConnect

Description

This is function SConnect.GetPacketInterceptorList.

130.3.2 SConnect.LoadWinSock2 Function

Pascal

```
function LoadWinSock2: Boolean;
```

C++

```
Boolean LoadWinSock2();
```

File

SConnect

Description

This is function SConnect.LoadWinSock2.

130.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TVarFlag (see page 19000)	This is record SConnect.TVarFlag.

130.4.1 SConnect.TVarFlag Enumeration

Pascal

```
TVarFlag = (
  vfByRef,
  vfVariant
);
```

C++

```
enum TVarFlag {
  vfByRef,
  vfVariant
};
```

File

SConnect

Description

This is record SConnect.TVarFlag.

130.5 Types

The following table lists types in this documentation.

Types

Name	Description
PIntArray (see page 19001)	This is type SConnect.PIntArray.
PVariantArray (see page 19001)	This is type SConnect.PVariantArray.
TIntArray (see page 19002)	This is type SConnect.TIntArray.
TVarFlags (see page 19002)	This is type SConnect.TVarFlags.
TVariantArray (see page 19002)	This is type SConnect.TVariantArray.

130.5.1 SConnect.PIntArray Type

Pascal

```
PIntArray = ^TIntArray;
```

C++

```
^TIntArray PIntArray;
```

File

SConnect

Description

This is type SConnect.PIntArray.

130.5.2 SConnect.PVariantArray Type

Pascal

```
PVariantArray = ^TVariantArray;
```

C++

```
^TVariantArray PVariantArray;
```

File

SConnect

Description

This is type SConnect.PVariantArray.

130.5.3 SConnect.TIntArray Type

Pascal

```
TIntArray = array[0..0] of Integer;
```

C++

```
array[0..0] of Integer TIntArray;
```

File

SConnect

Description

This is type SConnect.TIntArray.

130.5.4 SConnect.TVarFlags Type

Pascal

```
TVarFlags = set of TVarFlag;
```

C++

```
TVarFlag TVarFlags;
```

File

SConnect

Description

This is type SConnect.TVarFlags.

130.5.5 SConnect.TVariantArray Type

Pascal

```
TVariantArray = array[0..0] of OleVariant;
```

C++

```
array[0..0] of OleVariant TVariantArray;
```

File

SConnect

Description

This is type SConnect.TVariantArray.

130.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
WSACloseEvent (see page 19003)	This is variable SConnect.WSACloseEvent.
WSACreateEvent (see page 19003)	This is variable SConnect.WSACreateEvent.
WSAEventSelect (see page 19004)	This is variable SConnect.WSAEventSelect.
WSAResetEvent (see page 19004)	This is variable SConnect.WSAResetEvent.

130.6.1 SConnect.WSACloseEvent Variable

Pascal

```
WSACloseEvent: function(hEvent: THandle): Boolean stdcall;
```

C++

```
function(hEvent: THandle): Boolean stdcall WSACloseEvent;
```

File

SConnect

Description

This is variable SConnect.WSACloseEvent.

130.6.2 SConnect.WSACreateEvent Variable

Pascal

```
WSACreateEvent: function: THandle stdcall;
```

C++

```
function: THandle stdcall WSACreateEvent;
```

File

SConnect

Description

This is variable SConnect.WSACreateEvent.

130.6.3 SConnect.WSAEventSelect Variable

Pascal

```
WSAEventSelect: function(s: TSocket; hEventObject: THandle; lNetworkEvents: Integer): Integer
stdcall;
```

C++

```
function(s: TSocket; hEventObject: THandle; lNetworkEvents: Integer): Integer stdcall
WSAEventSelect;
```

File

SConnect

Description

This is variable SConnect.WSAEventSelect.

130.6.4 SConnect.WSAResetEvent Variable

Pascal

```
WSAResetEvent: function(hEvent: THandle): Boolean stdcall;
```

C++

```
function(hEvent: THandle): Boolean stdcall WSAResetEvent;
```

File

SConnect

Description

This is variable SConnect.WSAResetEvent.

130.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
asCreateObject (see page 19005)	This is constant SConnect.asCreateObject.
asError (see page 19005)	This is constant SConnect.asError.
asFreeObject (see page 19006)	This is constant SConnect.asFreeObject.
asGetAppServers (see page 19006)	This is constant SConnect.asGetAppServers.
asGetGUID (see page 19006)	This is constant SConnect.asGetGUID.
asGetID (see page 19007)	This is constant SConnect.asGetID.
asGetServers (see page 19007)	This is constant SConnect.asGetServers.
asInvoke (see page 19007)	This is constant SConnect.asInvoke.
asMask (see page 19008)	This is constant SConnect.asMask.
asSoapCommand (see page 19008)	This is constant SConnect.asSoapCommand.
CallSig (see page 19008)	This is constant SConnect.CallSig.
ResultSig (see page 19009)	This is constant SConnect.ResultSig.
THREAD_EXCEPTION (see page 19009)	This is constant SConnect.THREAD_EXCEPTION.
THREAD_RECEIVEDSTREAM (see page 19009)	This is constant SConnect.THREAD_RECEIVEDSTREAM.
THREAD_REPLACETRANSPORT (see page 19010)	This is constant SConnect.THREAD_REPLACETRANSPORT.
THREAD_SENDNOTIFY (see page 19010)	This is constant SConnect.THREAD_SENDNOTIFY.
THREAD_SENDSTREAM (see page 19010)	This is constant SConnect.THREAD_SENDSTREAM.

130.7.1 SConnect.asCreateObject Constant

Pascal

```
asCreateObject = $04;
```

C++

```
asCreateObject = 0x04;
```

File

SConnect

Description

This is constant SConnect.asCreateObject.

130.7.2 SConnect.asError Constant

Pascal

```
asError = $01;
```

C++

```
asError = 0x01;
```

File

SConnect

Description

This is constant SConnect.asError.

130.7.3 SConnect.asFreeObject Constant

Pascal

```
asFreeObject = $05;
```

C++

```
asFreeObject = 0x05;
```

File

SConnect

Description

This is constant SConnect.asFreeObject.

130.7.4 SConnect.asGetAppServers Constant

Pascal

```
asGetAppServers = $12;
```

C++

```
asGetAppServers = 0x12;
```

File

SConnect

Description

This is constant SConnect.asGetAppServers.

130.7.5 SConnect.asGetGUID Constant

Pascal

```
asGetGUID = $11;
```

C++

```
asGetGUID = 0x11;
```

File

SConnect

Description

This is constant SConnect.asGetGUID.

130.7.6 SConnect.asGetID Constant

Pascal

```
asGetID = $03;
```

C++

```
asGetID = 0x03;
```

File

SConnect

Description

This is constant SConnect.asGetID.

130.7.7 SConnect.asGetServers Constant

Pascal

```
asGetServers = $10;
```

C++

```
asGetServers = 0x10;
```

File

SConnect

Description

This is constant SConnect.asGetServers.

130.7.8 SConnect.asInvoke Constant

Pascal

```
asInvoke = $02;
```

C++

```
asInvoke = 0x02;
```

File

SConnect

Description

This is constant SConnect.asInvoke.

130.7.9 SConnect.asMask Constant

Pascal

```
asMask = $FF;
```

C++

```
asMask = 0xFF;
```

File

SConnect

Description

This is constant SConnect.asMask.

130.7.10 SConnect.asSoapCommand Constant

Pascal

```
asSoapCommand = $14;
```

C++

```
asSoapCommand = 0x14;
```

File

SConnect

Description

This is constant SConnect.asSoapCommand.

130.7.11 SConnect.CallSig Constant

Pascal

```
CallSig = $DA00;
```

C++

```
CallSig = 0xDA00;
```

File

SConnect

Description

This is constant SConnect.CallSig.

130.7.12 SConnect.ResultSig Constant

Pascal

```
ResultSig = $DB00;
```

C++

```
ResultSig = 0xDB00;
```

File

SConnect

Description

This is constant SConnect.ResultSig.

130.7.13 SConnect.THREAD_EXCEPTION Constant

Pascal

```
THREAD_EXCEPTION = THREAD_RECEIVEDSTREAM + 1;
```

C++

```
THREAD_EXCEPTION = THREAD_RECEIVEDSTREAM + 1;
```

File

SConnect

Description

This is constant SConnect.THREAD_EXCEPTION.

130.7.14 SConnect.THREAD_RECEIVEDSTREAM Constant

Pascal

```
THREAD_RECEIVEDSTREAM = THREAD_SENDSTREAM + 1;
```

C++

```
THREAD_RECEIVEDSTREAM = THREAD_SENDSTREAM + 1;
```

File

SConnect

Description

This is constant SConnect.THREAD_RECEIVEDSTREAM.

130.7.15 SConnect.THREAD_REPLACETRANSPORT Constant

Pascal

```
THREAD_REPLACETRANSPORT = THREAD_SENDNOTIFY + 1;
```

C++

```
THREAD_REPLACETRANSPORT = THREAD_SENDNOTIFY + 1;
```

File

SConnect

Description

This is constant SConnect.THREAD_REPLACETRANSPORT.

130.7.16 SConnect.THREAD_SENDNOTIFY Constant

Pascal

```
THREAD_SENDNOTIFY = THREAD_EXCEPTION + 1;
```

C++

```
THREAD_SENDNOTIFY = THREAD_EXCEPTION + 1;
```

File

SConnect

Description

This is constant SConnect.THREAD_SENDNOTIFY.

130.7.17 SConnect.THREAD_SENDSTREAM Constant

Pascal

```
THREAD_SENDSTREAM = WM_USER + 1;
```

C++

```
THREAD_SENDSTREAM = WM_USER + 1;
```

File

SConnect

Description

This is constant SConnect.THREAD_SENDSTREAM.

131 SessColn Namespace

This is namespace SessColn.

Topics

Name	Description
Classes (see page 19012)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 19035)	The following table lists structs, records, enums in this documentation.
Types (see page 19036)	The following table lists types in this documentation.
Variables (see page 19037)	The following table lists variables in this documentation.

Classes

	Name	Description
	ESessionsException (see page 19012)	ESessionsException is the exception that is raised when TSessionsService encounters a problem.
	TAbstractSessionIDGen (see page 19012)	This is class SessColn.TAbstractSessionIDGen.
	TAbstractSessionStatusChange (see page 19013)	This is class SessColn.TAbstractSessionStatusChange.
	TSessionIDGen (see page 19015)	This is class SessColn.TSessionIDGen.
	TSessionItem (see page 19016)	This is class SessColn.TSessionItem.
	TSessions (see page 19022)	This is class SessColn.TSessions.

Structs, Records, Enums

	Name	Description
	TSessionStatusFlag (see page 19035)	This is record SessColn.TSessionStatusFlag.

Types

Name	Description
TSessionStatusFlags (see page 19036)	This is type SessColn.TSessionStatusFlags.

Variables

Name	Description
Sessions (see page 19037)	This is variable SessColn.Sessions.

131.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ESessionsException (see page 19012)	ESessionsException is the exception that is raised when TSessionsService encounters a problem.
	TAbstractSessionIDGen (see page 19012)	This is class SessColn.TAbstractSessionIDGen.
	TAbstractSessionStatusChange (see page 19013)	This is class SessColn.TAbstractSessionStatusChange.
	TSessionIDGen (see page 19015)	This is class SessColn.TSessionIDGen.
	TSessionItem (see page 19016)	This is class SessColn.TSessionItem.
	TSessions (see page 19022)	This is class SessColn.TSessions.

131.1.1 ESessionsException Class

ESessionsException is the exception that is raised when TSessionsService encounters a problem.

Class Hierarchy

Pascal

```
ESessionsException = class(EWebBrokerException);
```

C++

```
class ESessionsException : public EWebBrokerException;
```

File

SessColn

Description

ESessionsException represents exceptions that occur when managing the Web sessions in a WebSnap application. For example, this exception is raised when the application tries to create a new session and the sessions service is already storing the maximum allowed number of sessions.

See Also

TSessionsService (see page 25582)

131.1.2 TAbstractSessionIDGen Class

Class Hierarchy

Pascal

```
TAbstractSessionIDGen = class;
```

C++

```
class TAbstractSessionIDGen;
```

File

SessColn

Description

This is class SessColn.TAbstractSessionIDGen.

131.1.2.1 TAbstractSessionIDGen Members

The following tables list the members exposed by TAbstractSessionIDGen.

Public Methods

	Name	Description
	NextID (see page 19013)	This is NextID, a member of class TAbstractSessionIDGen.

131.1.2.2 TAbstractSessionIDGen Methods

The methods of the TAbstractSessionIDGen class are listed here.

Public Methods

	Name	Description
	NextID (see page 19013)	This is NextID, a member of class TAbstractSessionIDGen.

131.1.2.2.1 TAbstractSessionIDGen.NextID Method

Pascal

```
function NextID: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString NextID() = 0;
```

Description

This is NextID, a member of class TAbstractSessionIDGen.

131.1.3 TAbstractSessionStatusChange Class

Class Hierarchy

Pascal

```
TAbstractSessionStatusChange = class;
```

C++

```
class TAbstractSessionStatusChange;
```

File

SessColn

Description

This is class SessColn.TAbstractSessionStatusChange.

131.1.3.1 TAbstractSessionStatusChange Members

The following tables list the members exposed by TAbstractSessionStatusChange.

Protected Methods

	Name	Description
	SessionStarted (see page 19014)	This is SessionStarted, a member of class TAbstractSessionStatusChange.
	SessionTerminated (see page 19014)	This is SessionTerminated, a member of class TAbstractSessionStatusChange.
	SessionTimeout (see page 19015)	This is SessionTimeout, a member of class TAbstractSessionStatusChange.

131.1.3.2 TAbstractSessionStatusChange Methods

The methods of the TAbstractSessionStatusChange class are listed here.

Protected Methods

	Name	Description
	SessionStarted (see page 19014)	This is SessionStarted, a member of class TAbstractSessionStatusChange.
	SessionTerminated (see page 19014)	This is SessionTerminated, a member of class TAbstractSessionStatusChange.
	SessionTimeout (see page 19015)	This is SessionTimeout, a member of class TAbstractSessionStatusChange.

131.1.3.2.1 TAbstractSessionStatusChange.SessionStarted Method

Pascal

```
procedure SessionStarted(AID: TSessionID; var AHandled: Boolean); virtual; abstract;
```

C++

```
virtual __fastcall SessionStarted(TSessionID AID, Boolean AHandled) = 0;
```

Description

This is SessionStarted, a member of class TAbstractSessionStatusChange.

131.1.3.2.2 TAbstractSessionStatusChange.SessionTerminated Method

Pascal

```
procedure SessionTerminated(AID: TSessionID; var AHandled: Boolean); virtual; abstract;
```

C++

```
virtual __fastcall SessionTerminated(TSessionID AID, Boolean AHandled) = 0;
```

Description

This is SessionTerminated, a member of class TAbstractSessionStatusChange.

131.1.3.2.3 TAbstractSessionStatusChange.SessionTimeout Method

Pascal

```
procedure SessionTimeout(AID: TSessionID; var AHandled: Boolean); virtual; abstract;
```

C++

```
virtual __fastcall SessionTimeout(TSessionID AID, Boolean AHandled) = 0;
```

Description

This is SessionTimeout, a member of class TAbstractSessionStatusChange.

131.1.4 TSessionIDGen Class

Class Hierarchy

Pascal

```
TSessionIDGen = class(TAbstractSessionIDGen);
```

C++

```
class TSessionIDGen : public TAbstractSessionIDGen;
```

File

SessColn

Description

This is class SessColn.TSessionIDGen.

131.1.4.1 TSessionIDGen Members

The following tables list the members exposed by TSessionIDGen.

Public Methods

	Name	Description
	Create (see page 19015)	This is Create, a member of class TSessionIDGen.
V	NextID (see page 19016)	This is NextID, a member of class TSessionIDGen.

131.1.4.2 TSessionIDGen.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TSessionIDGen();
```

Description

This is Create, a member of class TSessionIDGen.

131.1.4.3 TSessionIDGen Methods

The methods of the TSessionIDGen class are listed here.

Public Methods

	Name	Description
	NextID (see page 19016)	This is NextID, a member of class TSessionIDGen.

131.1.4.3.1 TSessionIDGen.NextID Method

Pascal

```
function NextID: string; override;
```

C++

```
virtual __fastcall AnsiString NextID();
```

Description

This is NextID, a member of class TSessionIDGen.

131.1.5 TSessionItem Class

Class Hierarchy

Pascal

```
TSessionItem = class(TCollectionItem);
```

C++

```
class TSessionItem : public TCollectionItem;
```

File

SessColn

Description

This is class SessColn.TSessionItem.

131.1.5.1 TSessionItem Members

The following tables list the members exposed by TSessionItem.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.

	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 19019)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 19018)	Creates and initializes a TCollectionItem instance.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 19018)	Destroys the TCollectionItem instance and frees its memory.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActive (see page 19019)	This is IsActive, a member of class TSessionItem.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ExpirationTime (see page 19020)	This is ExpirationTime, a member of class TSessionItem.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	Expired (see page 19020)	This is Expired, a member of class TSessionItem.
	ID (see page 19020)	A unique, permanent index for the item.
	Items (see page 19021)	This is Items, a member of class TSessionItem.

	StartTime (see page 19021)	This is StartTime, a member of class TSessionItem.
	StatusFlags (see page 19021)	This is StatusFlags, a member of class TSessionItem.
	Terminated (see page 19021)	This is Terminated, a member of class TSessionItem.
	TimeoutMinutes (see page 19022)	This is TimeoutMinutes, a member of class TSessionItem.
	TouchTime (see page 19022)	This is TouchTime, a member of class TSessionItem.

131.1.5.2 TSessionItem.Create Constructor

Creates and initializes a TCollectionItem instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TSessionItem(TCollection Collection);
```

Description

Do not call Create to instantiate a collection item. Instead, call the Add method of the collection to which the item should belong.

Collection identifies the TCollection instance to which the new item belongs.

See Also

Add (see page 2736)

Collection (see page 2761)

131.1.5.3 TSessionItem.Destroy Destructor

Destroys the TCollectionItem instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSessionItem();
```

Description

Destroy is called indirectly by TCollection's Clear or Delete method.

See Also

Clear (see page 2741)

Delete (see page 2741)

Free (see page 22300)

131.1.5.4 TSessionItem Methods

The methods of the TSessionItem class are listed here.

Public Methods

	Name	Description
	Assign (see page 19019)	Copies the contents of another, similar object.

	IsActive (see page 19019)	This is IsActive, a member of class TSessionItem.

131.1.5.4.1 TSessionItem.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo (see page 2931)

EConvertError (see page 22724)

131.1.5.4.2 TSessionItem.IsActive Method

Pascal

```
function IsActive: Boolean;
```

C++

```
__fastcall Boolean IsActive();
```

Description

This is IsActive, a member of class TSessionItem.

131.1.5.5 TSessionItem Properties

The properties of the TSessionItem class are listed here.

Public Properties

	Name	Description
R	ExpirationTime (see page 19020)	This is ExpirationTime, a member of class TSessionItem.

Published Properties

	Name	Description
	Expired (see page 19020)	This is Expired, a member of class TSessionItem.
	ID (see page 19020)	A unique, permanent index for the item.
	Items (see page 19021)	This is Items, a member of class TSessionItem.
	StartTime (see page 19021)	This is StartTime, a member of class TSessionItem.
	StatusFlags (see page 19021)	This is StatusFlags, a member of class TSessionItem.
	Terminated (see page 19021)	This is Terminated, a member of class TSessionItem.
	TimeoutMinutes (see page 19022)	This is TimeoutMinutes, a member of class TSessionItem.
	TouchTime (see page 19022)	This is TouchTime, a member of class TSessionItem.

131.1.5.5.1 TSessionItem.ExpirationTime Property

Pascal

```
property ExpirationTime: TDateTime;
```

C++

```
__property TDateTime ExpirationTime;
```

Description

This is ExpirationTime, a member of class TSessionItem.

131.1.5.5.2 TSessionItem.Expired Property

Pascal

```
property Expired: Boolean;
```

C++

```
__property Boolean Expired;
```

Description

This is Expired, a member of class TSessionItem.

131.1.5.5.3 TSessionItem.ID Property

A unique, permanent index for the item.

Pascal

```
property ID: string;
```

C++

```
__property AnsiString ID;
```

Description

In most cases, ID is the value of the item's Index property when it was first added to the collection. But, unlike Index values, ID values are never changed or reassigned. If an item is removed from a collection, its ID not used again.

An item's Index may change when other items are deleted from the collection or when items are rearranged. ID provides a unique identifier for each item that is unchanged for the life of the collection.

See Also

Index (see page 2762)

131.1.5.5.4 TSessionItem.Items Property

Pascal

```
property Items: TAbstractNamedVariants;
```

C++

```
__property TAbstractNamedVariants Items;
```

Description

This is Items, a member of class TSessionItem.

131.1.5.5.5 TSessionItem.StartTime Property

Pascal

```
property StartTime: TDateTime;
```

C++

```
__property TDateTime StartTime;
```

Description

This is StartTime, a member of class TSessionItem.

131.1.5.5.6 TSessionItem.StatusFlags Property

Pascal

```
property StatusFlags: TSessionStatusFlags;
```

C++

```
__property TSessionStatusFlags StatusFlags;
```

Description

This is StatusFlags, a member of class TSessionItem.

131.1.5.5.7 TSessionItem.Terminated Property

Pascal

```
property Terminated: Boolean;
```

C++

```
__property Boolean Terminated;
```

Description

This is Terminated, a member of class TSessionItem.

131.1.5.5.8 TSessionItem.TimeoutMinutes Property

Pascal

```
property TimeoutMinutes: Integer;
```

C++

```
__property int TimeoutMinutes;
```

Description

This is TimeoutMinutes, a member of class TSessionItem.

131.1.5.5.9 TSessionItem.TouchTime Property

Pascal

```
property TouchTime: TDateTime;
```

C++

```
__property TDateTime TouchTime;
```

Description

This is TouchTime, a member of class TSessionItem.

131.1.6 TSessions Class

Class Hierarchy

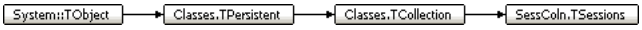

Pascal

```
TSessions = class(TCollection);
```

C++

```
class TSessions : public TCollection;
```

File

SessColn

Description

This is class SessColn.TSessions.

131.1.6.1 TSessions Members

The following tables list the members exposed by TSessions.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 2736)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	CheckStatusChange (see page 19026)	This is CheckStatusChange, a member of class TSessions.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	ConnectActiveSession (see page 19026)	This is ConnectActiveSession, a member of class TSessions.
	ConnectSession (see page 19027)	This is ConnectSession, a member of class TSessions.
	Create (see page 19025)	Constructs an object and initializes its data before the object is first used.
V	CreateSessionIDGen (see page 19027)	This is CreateSessionIDGen, a member of class TSessions.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 19025)	Destroys the collection and each item in it.
	DisconnectSession (see page 19027)	This is DisconnectSession, a member of class TSessions.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.

V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetExpirationTime (see page 19027)	This is GetExpirationTime, a member of class TSessions.
	GetExpired (see page 19027)	This is GetExpired, a member of class TSessions.
	GetIDCount (see page 19028)	This is GetIDCount, a member of class TSessions.
	GetIDs (see page 19028)	This is GetIDs, a member of class TSessions.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsActive (see page 19028)	This is GetIsActive, a member of class TSessions.
	GetItemCount (see page 19028)	This is GetItemCount, a member of class TSessions.
	GetItems (see page 19029)	This is GetItems, a member of class TSessions.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetSession (see page 19029)	This is GetSession, a member of class TSessions.
	GetSessionIDGen (see page 19029)	This is GetSessionIDGen, a member of class TSessions.
	GetStartTime (see page 19029)	This is GetStartTime, a member of class TSessions.
	GetTerminated (see page 19029)	This is GetTerminated, a member of class TSessions.
	GetTimeoutMinutes (see page 19030)	This is GetTimeoutMinutes, a member of class TSessions.
	GetTouchTime (see page 19030)	This is GetTouchTime, a member of class TSessions.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	ItemValue (see page 19030)	This is ItemValue, a member of class TSessions.
	LoadFromFile (see page 19030)	This is LoadFromFile, a member of class TSessions.
	LoadFromStream (see page 19031)	This is LoadFromStream, a member of class TSessions.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NotifyStatusChange (see page 19031)	This is NotifyStatusChange, a member of class TSessions.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 19031)	This is SaveToFile, a member of class TSessions.
	SaveToStream (see page 19031)	This is SaveToStream, a member of class TSessions.
	SessionExists (see page 19031)	This is SessionExists, a member of class TSessions.
	SetItemValue (see page 19032)	This is SetItemValue, a member of class TSessions.
	SetTimeoutMinutes (see page 19032)	This is SetTimeoutMinutes, a member of class TSessions.
	StartSession (see page 19032)	This is StartSession, a member of class TSessions.
	TerminateSession (see page 19032)	This is TerminateSession, a member of class TSessions.
	TouchSession (see page 19033)	This is TouchSession, a member of class TSessions.

Public Properties

	Name	Description
R	ActiveCount (see page 19033)	This is ActiveCount, a member of class TSessions.
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.

	FileName (see page 19033)	This is FileName, a member of class TSessions.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 2751)	Lists the items in the collection.
	MaxSessions (see page 19033)	This is MaxSessions, a member of class TSessions.

131.1.6.2 TSessions.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TSessions();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

131.1.6.3 TSessions.Destroy Destructor

Destroys the collection and each item in it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSessions();
```

Description

Destroy calls the Clear method to free each item referenced in the Items array, then destroys the collection itself.

See Also

Clear (see page 2741)

Free (see page 22300)

131.1.6.4 TSessions Methods

The methods of the TSessions class are listed here.

Public Methods

	Name	Description
	CheckStatusChange (see page 19026)	This is CheckStatusChange, a member of class TSessions.

	ConnectActiveSession (see page 19026)	This is ConnectActiveSession, a member of class TSessions.
	ConnectSession (see page 19027)	This is ConnectSession, a member of class TSessions.
V	CreateSessionIDGen (see page 19027)	This is CreateSessionIDGen, a member of class TSessions.
	DisconnectSession (see page 19027)	This is DisconnectSession, a member of class TSessions.
	GetExpirationTime (see page 19027)	This is GetExpirationTime, a member of class TSessions.
	GetExpired (see page 19027)	This is GetExpired, a member of class TSessions.
	GetIDCount (see page 19028)	This is GetIDCount, a member of class TSessions.
	GetIDs (see page 19028)	This is GetIDs, a member of class TSessions.
	GetIsActive (see page 19028)	This is GetIsActive, a member of class TSessions.
	GetItemCount (see page 19028)	This is GetItemCount, a member of class TSessions.
	GetItems (see page 19029)	This is GetItems, a member of class TSessions.
	GetSession (see page 19029)	This is GetSession, a member of class TSessions.
	GetSessionIDGen (see page 19029)	This is GetSessionIDGen, a member of class TSessions.
	GetStartTime (see page 19029)	This is GetStartTime, a member of class TSessions.
	GetTerminated (see page 19029)	This is GetTerminated, a member of class TSessions.
	GetTimeoutMinutes (see page 19030)	This is GetTimeoutMinutes, a member of class TSessions.
	GetTouchTime (see page 19030)	This is GetTouchTime, a member of class TSessions.
	ItemValue (see page 19030)	This is ItemValue, a member of class TSessions.
	LoadFromFile (see page 19030)	This is LoadFromFile, a member of class TSessions.
	LoadFromStream (see page 19031)	This is LoadFromStream, a member of class TSessions.
	NotifyStatusChange (see page 19031)	This is NotifyStatusChange, a member of class TSessions.
	SaveToFile (see page 19031)	This is SaveToFile, a member of class TSessions.
	SaveToStream (see page 19031)	This is SaveToStream, a member of class TSessions.
	SessionExists (see page 19031)	This is SessionExists, a member of class TSessions.
	SetItemValue (see page 19032)	This is SetItemValue, a member of class TSessions.
	SetTimeoutMinutes (see page 19032)	This is SetTimeoutMinutes, a member of class TSessions.
	StartSession (see page 19032)	This is StartSession, a member of class TSessions.
	TerminateSession (see page 19032)	This is TerminateSession, a member of class TSessions.
	TouchSession (see page 19033)	This is TouchSession, a member of class TSessions.

131.1.6.4.1 TSessions.CheckStatusChange Method

Pascal

```
function CheckStatusChange: Boolean;
```

C++

```
__fastcall Boolean CheckStatusChange();
```

Description

This is CheckStatusChange, a member of class TSessions.

131.1.6.4.2 TSessions.ConnectActiveSession Method

Pascal

```
function ConnectActiveSession(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean ConnectActiveSession(TSessionID AID);
```

Description

This is ConnectActiveSession, a member of class TSessions.

131.1.6.4.3 TSessions.ConnectSession Method

Pascal

```
function ConnectSession(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean ConnectSession(TSessionID AID);
```

Description

This is ConnectSession, a member of class TSessions.

131.1.6.4.4 TSessions.CreateSessionIDGen Method

Pascal

```
function CreateSessionIDGen: TAbstractSessionIDGen; virtual;
```

C++

```
virtual __fastcall TAbstractSessionIDGen CreateSessionIDGen();
```

Description

This is CreateSessionIDGen, a member of class TSessions.

131.1.6.4.5 TSessions.DisconnectSession Method

Pascal

```
procedure DisconnectSession(AID: TSessionID);
```

C++

```
__fastcall DisconnectSession(TSessionID AID);
```

Description

This is DisconnectSession, a member of class TSessions.

131.1.6.4.6 TSessions.GetExpirationTime Method

Pascal

```
function GetExpirationTime(AID: TSessionID): TDateTime;
```

C++

```
__fastcall TDateTime GetExpirationTime(TSessionID AID);
```

Description

This is GetExpirationTime, a member of class TSessions.

131.1.6.4.7 TSessions.GetExpired Method

Pascal

```
function GetExpired(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean GetExpired(TSessionID AID);
```

Description

This is GetExpired, a member of class TSessions.

131.1.6.4.8 TSessions.GetIDCount Method

Pascal

```
function GetIDCount: Integer;
```

C++

```
__fastcall int GetIDCount();
```

Description

This is GetIDCount, a member of class TSessions.

131.1.6.4.9 TSessions.GetIDs Method

Pascal

```
procedure GetIDs(var AList: TList);
```

C++

```
__fastcall GetIDs(TList AList);
```

Description

This is GetIDs, a member of class TSessions.

131.1.6.4.10 TSessions.GetIsActive Method

Pascal

```
function GetIsActive(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean GetIsActive(TSessionID AID);
```

Description

This is GetIsActive, a member of class TSessions.

131.1.6.4.11 TSessions.GetItemCount Method

Pascal

```
function GetItemCount(AID: TSessionID): Integer;
```

C++

```
__fastcall int GetItemCount(TSessionID AID);
```

Description

This is GetItemCount, a member of class TSessions.

131.1.6.4.12 TSessions.GetItems Method

Pascal

```
procedure GetItems(AID: TSessionID; var AItems: TAbstractNamedVariants);
```

C++

```
__fastcall GetItems(TSessionID AID, TAbstractNamedVariants AItems);
```

Description

This is GetItems, a member of class TSessions.

131.1.6.4.13 TSessions.GetSession Method

Pascal

```
function GetSession(AID: TSessionID; var AItem: TSessionItem): Boolean;
```

C++

```
__fastcall Boolean GetSession(TSessionID AID, TSessionItem AItem);
```

Description

This is GetSession, a member of class TSessions.

131.1.6.4.14 TSessions.GetSessionIDGen Method

Pascal

```
function GetSessionIDGen: TAbstractSessionIDGen;
```

C++

```
__fastcall TAbstractSessionIDGen GetSessionIDGen();
```

Description

This is GetSessionIDGen, a member of class TSessions.

131.1.6.4.15 TSessions.GetStartTime Method

Pascal

```
function GetStartTime(AID: TSessionID): TDateTime;
```

C++

```
__fastcall TDateTime GetStartTime(TSessionID AID);
```

Description

This is GetStartTime, a member of class TSessions.

131.1.6.4.16 TSessions.GetTerminated Method

Pascal

```
function GetTerminated(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean GetTerminated(TSessionID AID);
```

Description

This is GetTerminated, a member of class TSessions.

131.1.6.4.17 TSessions.GetTimeoutMinutes Method

Pascal

```
function GetTimeoutMinutes(AID: TSessionID): Integer;
```

C++

```
__fastcall int GetTimeoutMinutes(TSessionID AID);
```

Description

This is GetTimeoutMinutes, a member of class TSessions.

131.1.6.4.18 TSessions.GetTouchTime Method

Pascal

```
function GetTouchTime(AID: TSessionID): TDateTime;
```

C++

```
__fastcall TDateTime GetTouchTime(TSessionID AID);
```

Description

This is GetTouchTime, a member of class TSessions.

131.1.6.4.19 TSessions.ItemValue Method

Pascal

```
function ItemValue(AID: TSessionID; const AName: string): Variant;
```

C++

```
__fastcall Variant ItemValue(TSessionID AID, const AnsiString AName);
```

Description

This is ItemValue, a member of class TSessions.

131.1.6.4.20 TSessions.LoadFromFile Method

Pascal

```
procedure LoadFromFile(const Filename: string);
```

C++

```
__fastcall LoadFromFile(const AnsiString Filename);
```

Description

This is LoadFromFile, a member of class TSessions.

131.1.6.4.21 TSessions.LoadFromStream Method

Pascal

```
procedure LoadFromStream(S: TStream);
```

C++

```
__fastcall LoadFromStream(TStream S);
```

Description

This is LoadFromStream, a member of class TSessions.

131.1.6.4.22 TSessions.NotifyStatusChange Method

Pascal

```
procedure NotifyStatusChange(AStatusChange: TAbstractSessionStatusChange);
```

C++

```
__fastcall NotifyStatusChange(TAbstractSessionStatusChange AStatusChange);
```

Description

This is NotifyStatusChange, a member of class TSessions.

131.1.6.4.23 TSessions.SaveToFile Method

Pascal

```
procedure SaveToFile(const Filename: string);
```

C++

```
__fastcall SaveToFile(const AnsiString Filename);
```

Description

This is SaveToFile, a member of class TSessions.

131.1.6.4.24 TSessions.SaveToStream Method

Pascal

```
procedure SaveToStream(S: TStream);
```

C++

```
__fastcall SaveToStream(TStream S);
```

Description

This is SaveToStream, a member of class TSessions.

131.1.6.4.25 TSessions.SessionExists Method

Pascal

```
function SessionExists(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean SessionExists(TSessionID AID);
```

Description

This is SessionExists, a member of class TSessions.

131.1.6.4.26 TSessions.SetItemValue Method

Pascal

```
procedure SetItemValue(AID: TSessionID; const AName: string; const AValue: Variant);
```

C++

```
__fastcall SetItemValue(TSessionID AID, const AnsiString AName, const Variant AValue);
```

Description

This is SetItemValue, a member of class TSessions.

131.1.6.4.27 TSessions.SetTimeoutMinutes Method

Pascal

```
procedure SetTimeoutMinutes(AID: TSessionID; AValue: Integer);
```

C++

```
__fastcall SetTimeoutMinutes(TSessionID AID, int AValue);
```

Description

This is SetTimeoutMinutes, a member of class TSessions.

131.1.6.4.28 TSessions.StartSession Method

Pascal

```
function StartSession: TSessionID;
```

C++

```
__fastcall TSessionID StartSession();
```

Description

This is StartSession, a member of class TSessions.

131.1.6.4.29 TSessions.TerminateSession Method

Pascal

```
function TerminateSession(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean TerminateSession(TSessionID AID);
```

Description

This is TerminateSession, a member of class TSessions.

131.1.6.4.30 TSessions.TouchSession Method

Pascal

```
function TouchSession(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean TouchSession(TSessionID AID);
```

Description

This is TouchSession, a member of class TSessions.

131.1.6.5 TSessions Properties

The properties of the TSessions class are listed here.

Public Properties

	Name	Description
R	ActiveCount (see page 19033)	This is ActiveCount, a member of class TSessions.
	FileName (see page 19033)	This is FileName, a member of class TSessions.
	MaxSessions (see page 19033)	This is MaxSessions, a member of class TSessions.

131.1.6.5.1 TSessions.ActiveCount Property

Pascal

```
property ActiveCount: Integer;
```

C++

```
__property int ActiveCount;
```

Description

This is ActiveCount, a member of class TSessions.

131.1.6.5.2 TSessions.FileName Property

Pascal

```
property FileName: string;
```

C++

```
__property AnsiString FileName;
```

Description

This is FileName, a member of class TSessions.

131.1.6.5.3 TSessions.MaxSessions Property

Pascal

```
property MaxSessions: Integer;
```

C++

```
__property int MaxSessions;
```

Description

This is MaxSessions, a member of class TSessions.

131.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TSessionStatusFlag (see page 19035)	This is record SessColn.TSessionStatusFlag.

131.2.1 SessColn.TSessionStatusFlag Enumeration

Pascal

```
TSessionStatusFlag = (
  ssStarted,
  ssTerminated,
  ssTimeOut
);
```

C++

```
enum TSessionStatusFlag {
  ssStarted,
  ssTerminated,
  ssTimeOut
};
```

File

SessColn

Description

This is record SessColn.TSessionStatusFlag.

131.3 Types

The following table lists types in this documentation.

Types

Name	Description
TSessionStatusFlags (see page 19036)	This is type SessColn.TSessionStatusFlags.

131.3.1 SessColn.TSessionStatusFlags Type

Pascal

```
TSessionStatusFlags = set of TSessionStatusFlag;
```

C++

```
TSessionStatusFlag TSessionStatusFlags;
```

File

SessColn

Description

This is type SessColn.TSessionStatusFlags.

131.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
Sessions (see page 19037)	This is variable SessColn.Sessions.

131.4.1 SessColn.Sessions Variable

Pascal

```
Sessions: TSessions;
```

C++

```
TSessions Sessions;
```

File

SessColn

Description

This is variable SessColn.Sessions.

132 SHDocVw Namespace

This is namespace SHDocVw.

Topics

Name	Description
Classes (see page 19045)	The following table lists classes in this documentation.
Interfaces (see page 19276)	The following table lists interfaces in this documentation.
Types (see page 19451)	The following table lists types in this documentation.
Constants (see page 19486)	The following table lists constants in this documentation.

Classes

	Name	Description
	CoCScriptErrorList (see page 19045)	This is class SHDocVw.CoCScriptErrorList.
	CoInternetExplorer (see page 19046)	This is class SHDocVw.CoInternetExplorer.
	CoSearchAssistantOC (see page 19047)	This is class SHDocVw.CoSearchAssistantOC.
	CoShellBrowserWindow (see page 19049)	This is class SHDocVw.CoShellBrowserWindow.
	CoShellNameSpace (see page 19050)	This is class SHDocVw.CoShellNameSpace.
	CoShellUIHelper (see page 19051)	This is class SHDocVw.CoShellUIHelper.
	CoShellWindows (see page 19052)	This is class SHDocVw.CoShellWindows.
	TInternetExplorer (see page 19053)	This is class SHDocVw.TInternetExplorer.
	TShellNameSpace (see page 19101)	This is class SHDocVw.TShellNameSpace.
	TShellUIHelper (see page 19121)	This is class SHDocVw.TShellUIHelper.
	TShellWindows (see page 19135)	This is class SHDocVw.TShellWindows.
	TWebBrowser (see page 19147)	TWebBrowser provides access to the Web browser functionality of Microsoft's Shell Doc Object and Control Library (SHDOCVW.DLL).
	TWebBrowser_V1 (see page 19221)	This is class SHDocVw.TWebBrowser_V1.

Constants

Name	Description
CGID_IWebBrowser (see page 19488)	This is constant SHDocVw.CGID_IWebBrowser.
CLASS_CScriptErrorList (see page 19489)	This is constant SHDocVw.CLASS_CScriptErrorList.
CLASS_InternetExplorer (see page 19489)	This is constant SHDocVw.CLASS_InternetExplorer.
CLASS_SearchAssistantOC (see page 19489)	This is constant SHDocVw.CLASS_SearchAssistantOC.
CLASS_ShellBrowserWindow (see page 19490)	This is constant SHDocVw.CLASS_ShellBrowserWindow.
CLASS_ShellNameSpace (see page 19490)	This is constant SHDocVw.CLASS_ShellNameSpace.
CLASS_ShellUIHelper (see page 19490)	This is constant SHDocVw.CLASS_ShellUIHelper.

CLASS_ShellWindows (see page 19491)	This is constant SHDocVw.CLASS_ShellWindows.
CLASS_WebBrowser (see page 19491)	This is constant SHDocVw.CLASS_WebBrowser.
CLASS_WebBrowser_V1 (see page 19491)	This is constant SHDocVw.CLASS_WebBrowser_V1.
CSC_NAVIGATEBACK (see page 19492)	This is constant SHDocVw.CSC_NAVIGATEBACK.
CSC_NAVIGATEFORWARD (see page 19492)	This is constant SHDocVw.CSC_NAVIGATEFORWARD.
CSC_UPDATECOMMANDS (see page 19492)	This is constant SHDocVw.CSC_UPDATECOMMANDS.
DIID__SearchAssistantEvents (see page 19493)	This is constant SHDocVw.DIID__SearchAssistantEvents.
DIID_DShellNameSpaceEvents (see page 19493)	This is constant SHDocVw.DIID_DShellNameSpaceEvents.
DIID_DShellWindowsEvents (see page 19493)	This is constant SHDocVw.DIID_DShellWindowsEvents.
DIID_DWebBrowserEvents (see page 19494)	This is constant SHDocVw.DIID_DWebBrowserEvents.
DIID_DWebBrowserEvents2 (see page 19494)	This is constant SHDocVw.DIID_DWebBrowserEvents2.
dtlOcxPage (see page 19494)	This is constant SHDocVw.dtlOcxPage.
dtlServerPage (see page 19495)	This is constant SHDocVw.dtlServerPage.
HTMLID_FIND (see page 19495)	This is constant SHDocVw.HTMLID_FIND.
HTMLID_OPTIONS (see page 19495)	This is constant SHDocVw.HTMLID_OPTIONS.
HTMLID_VIEWSOURCE (see page 19496)	This is constant SHDocVw.HTMLID_VIEWSOURCE.
IID_IScriptErrorList (see page 19496)	This is constant SHDocVw.IID_IScriptErrorList.
IID_ISearch (see page 19496)	This is constant SHDocVw.IID_ISearch.
IID_ISearchAssistantOC (see page 19497)	This is constant SHDocVw.IID_ISearchAssistantOC.
IID_ISearchAssistantOC2 (see page 19497)	This is constant SHDocVw.IID_ISearchAssistantOC2.
IID_ISearchAssistantOC3 (see page 19497)	This is constant SHDocVw.IID_ISearchAssistantOC3.
IID_ISearches (see page 19498)	This is constant SHDocVw.IID_ISearches.
IID_IShellFavoritesNameSpace (see page 19498)	This is constant SHDocVw.IID_IShellFavoritesNameSpace.
IID_IShellNameSpace (see page 19498)	This is constant SHDocVw.IID_IShellNameSpace.
IID_IShellUIHelper (see page 19499)	This is constant SHDocVw.IID_IShellUIHelper.
IID_IShellWindows (see page 19499)	This is constant SHDocVw.IID_IShellWindows.
IID_IWebBrowser (see page 19499)	This is constant SHDocVw.IID_IWebBrowser.
IID_IWebBrowser2 (see page 19500)	This is constant SHDocVw.IID_IWebBrowser2.
IID_IWebBrowserApp (see page 19500)	This is constant SHDocVw.IID_IWebBrowserApp.
LIBID_SHDocVw (see page 19500)	This is constant SHDocVw.LIBID_SHDocVw.
navAllowAutosearch (see page 19501)	This is constant SHDocVw.navAllowAutosearch.
navBrowserBar (see page 19501)	This is constant SHDocVw.navBrowserBar.
navNoHistory (see page 19501)	This is constant SHDocVw.navNoHistory.
navNoReadFromCache (see page 19502)	This is constant SHDocVw.navNoReadFromCache.
navNoWriteToCache (see page 19502)	This is constant SHDocVw.navNoWriteToCache.
navOpenInNewWindow (see page 19502)	This is constant SHDocVw.navOpenInNewWindow.
OLECMDEXECOPT_DODEFAULT (see page 19503)	This is constant SHDocVw.OLECMDEXECOPT_DODEFAULT.
OLECMDEXECOPT_DONTPROMPTUSER (see page 19503)	This is constant SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER.
OLECMDEXECOPT_PROMPTUSER (see page 19503)	This is constant SHDocVw.OLECMDEXECOPT_PROMPTUSER.
OLECMDEXECOPT_SHOWHELP (see page 19504)	This is constant SHDocVw.OLECMDEXECOPT_SHOWHELP.
OLECMDF_DEFHIDEONCTXTMENU (see page 19504)	This is constant SHDocVw.OLECMDF_DEFHIDEONCTXTMENU.
OLECMDF_ENABLED (see page 19504)	This is constant SHDocVw.OLECMDF_ENABLED.
OLECMDF_INVISIBLE (see page 19505)	This is constant SHDocVw.OLECMDF_INVISIBLE.
OLECMDF_LATCHED (see page 19505)	This is constant SHDocVw.OLECMDF_LATCHED.
OLECMDF_NINCHED (see page 19505)	This is constant SHDocVw.OLECMDF_NINCHED.
OLECMDF_SUPPORTED (see page 19506)	This is constant SHDocVw.OLECMDF_SUPPORTED.
OLECMDID_ALLOWUILESSSAVEAS (see page 19506)	This is constant SHDocVw.OLECMDID_ALLOWUILESSSAVEAS.
OLECMDID_CLEARSELECTION (see page 19506)	This is constant SHDocVw.OLECMDID_CLEARSELECTION.
OLECMDID_CLOSE (see page 19507)	This is constant SHDocVw.OLECMDID_CLOSE.
OLECMDID_COPY (see page 19507)	This is constant SHDocVw.OLECMDID_COPY.
OLECMDID_CUT (see page 19507)	This is constant SHDocVw.OLECMDID_CUT.
OLECMDID_DELETE (see page 19508)	This is constant SHDocVw.OLECMDID_DELETE.
OLECMDID_DONTDOWNLOADCSS (see page 19508)	This is constant SHDocVw.OLECMDID_DONTDOWNLOADCSS.
OLECMDID_ENABLE_INTERACTION (see page 19508)	This is constant SHDocVw.OLECMDID_ENABLE_INTERACTION.

OLECMDID_FIND (see page 19509)	This is constant SHDocVw.OLECMDID_FIND.
OLECMDID_GETPRINTTEMPLATE (see page 19509)	This is constant SHDocVw.OLECMDID_GETPRINTTEMPLATE.
OLECMDID_GETZOOMRANGE (see page 19509)	This is constant SHDocVw.OLECMDID_GETZOOMRANGE.
OLECMDID_HIDETOOLBARS (see page 19510)	This is constant SHDocVw.OLECMDID_HIDETOOLBARS.
OLECMDID_HTTPEQUIV (see page 19510)	This is constant SHDocVw.OLECMDID_HTTPEQUIV.
OLECMDID_HTTPEQUIV_DONE (see page 19510)	This is constant SHDocVw.OLECMDID_HTTPEQUIV_DONE.
OLECMDID_NEW (see page 19511)	This is constant SHDocVw.OLECMDID_NEW.
OLECMDID_ONTOOLBARACTIVATED (see page 19511)	This is constant SHDocVw.OLECMDID_ONTOOLBARACTIVATED.
OLECMDID_ONUNLOAD (see page 19511)	This is constant SHDocVw.OLECMDID_ONUNLOAD.
OLECMDID_OPEN (see page 19512)	This is constant SHDocVw.OLECMDID_OPEN.
OLECMDID_PAGESETUP (see page 19512)	This is constant SHDocVw.OLECMDID_PAGESETUP.
OLECMDID_PASTE (see page 19512)	This is constant SHDocVw.OLECMDID_PASTE.
OLECMDID_PASTESPECIAL (see page 19513)	This is constant SHDocVw.OLECMDID_PASTESPECIAL.
OLECMDID_PREREFRESH (see page 19513)	This is constant SHDocVw.OLECMDID_PREREFRESH.
OLECMDID_PRINT (see page 19513)	This is constant SHDocVw.OLECMDID_PRINT.
OLECMDID_PRINT2 (see page 19514)	This is constant SHDocVw.OLECMDID_PRINT2.
OLECMDID_PRINTPREVIEW (see page 19514)	This is constant SHDocVw.OLECMDID_PRINTPREVIEW.
OLECMDID_PRINTPREVIEW2 (see page 19514)	This is constant SHDocVw.OLECMDID_PRINTPREVIEW2.
OLECMDID_PROPERTIES (see page 19515)	This is constant SHDocVw.OLECMDID_PROPERTIES.
OLECMDID_PROPERTYBAG2 (see page 19515)	This is constant SHDocVw.OLECMDID_PROPERTYBAG2.
OLECMDID_REDO (see page 19515)	This is constant SHDocVw.OLECMDID_REDO.
OLECMDID_REFRESH (see page 19516)	This is constant SHDocVw.OLECMDID_REFRESH.
OLECMDID_SAVE (see page 19516)	This is constant SHDocVw.OLECMDID_SAVE.
OLECMDID_SAVEAS (see page 19516)	This is constant SHDocVw.OLECMDID_SAVEAS.
OLECMDID_SAVECOPYAS (see page 19517)	This is constant SHDocVw.OLECMDID_SAVECOPYAS.
OLECMDID_SELECTALL (see page 19517)	This is constant SHDocVw.OLECMDID_SELECTALL.
OLECMDID_SETDOWNLOADSTATE (see page 19517)	This is constant SHDocVw.OLECMDID_SETDOWNLOADSTATE.
OLECMDID_SETPRINTTEMPLATE (see page 19518)	This is constant SHDocVw.OLECMDID_SETPRINTTEMPLATE.
OLECMDID_SETPROGRESSMAX (see page 19518)	This is constant SHDocVw.OLECMDID_SETPROGRESSMAX.
OLECMDID_SETPROGRESSPOS (see page 19518)	This is constant SHDocVw.OLECMDID_SETPROGRESSPOS.
OLECMDID_SETPROGRESSTEXT (see page 19519)	This is constant SHDocVw.OLECMDID_SETPROGRESSTEXT.
OLECMDID_SETTITLE (see page 19519)	This is constant SHDocVw.OLECMDID_SETTITLE.
OLECMDID_SHOWFIND (see page 19519)	This is constant SHDocVw.OLECMDID_SHOWFIND.
OLECMDID_SHOWMESSAGE (see page 19520)	This is constant SHDocVw.OLECMDID_SHOWMESSAGE.
OLECMDID_SHOWPAGESETUP (see page 19520)	This is constant SHDocVw.OLECMDID_SHOWPAGESETUP.
OLECMDID_SHOWPRINT (see page 19520)	This is constant SHDocVw.OLECMDID_SHOWPRINT.
OLECMDID_SHOWSCRIPTERROR (see page 19521)	This is constant SHDocVw.OLECMDID_SHOWSCRIPTERROR.
OLECMDID_SPELL (see page 19521)	This is constant SHDocVw.OLECMDID_SPELL.
OLECMDID_STOP (see page 19521)	This is constant SHDocVw.OLECMDID_STOP.
OLECMDID_STOPDOWNLOAD (see page 19522)	This is constant SHDocVw.OLECMDID_STOPDOWNLOAD.
OLECMDID_UNDO (see page 19522)	This is constant SHDocVw.OLECMDID_UNDO.
OLECMDID_UPDATECOMMANDS (see page 19522)	This is constant SHDocVw.OLECMDID_UPDATECOMMANDS.
OLECMDID_UPDATEPAGESTATUS (see page 19523)	This is constant SHDocVw.OLECMDID_UPDATEPAGESTATUS.
OLECMDID_ZOOM (see page 19523)	This is constant SHDocVw.OLECMDID_ZOOM.
READYSTATE_COMPLETE (see page 19523)	This is constant SHDocVw.READYSTATE_COMPLETE.
READYSTATE_INTERACTIVE (see page 19524)	This is constant SHDocVw.READYSTATE_INTERACTIVE.
READYSTATE_LOADED (see page 19524)	This is constant SHDocVw.READYSTATE_LOADED.
READYSTATE_LOADING (see page 19524)	This is constant SHDocVw.READYSTATE_LOADING.
READYSTATE_UNINITIALIZED (see page 19525)	This is constant SHDocVw.READYSTATE_UNINITIALIZED.
REFRESH_COMPLETELY (see page 19525)	This is constant SHDocVw.REFRESH_COMPLETELY.
REFRESH_CONTINUE (see page 19525)	This is constant SHDocVw.REFRESH_CONTINUE.
REFRESH_IFEXPIRED (see page 19526)	This is constant SHDocVw.REFRESH_IFEXPIRED.
REFRESH_NORMAL (see page 19526)	This is constant SHDocVw.REFRESH_NORMAL.
secureLockIconMixed (see page 19526)	This is constant SHDocVw.secureLockIconMixed.

secureLockIconSecure128Bit (see page 19527)	This is constant SHDocVw.secureLockIconSecure128Bit.
secureLockIconSecure40Bit (see page 19527)	This is constant SHDocVw.secureLockIconSecure40Bit.
secureLockIconSecure56Bit (see page 19527)	This is constant SHDocVw.secureLockIconSecure56Bit.
secureLockIconSecureFortezza (see page 19528)	This is constant SHDocVw.secureLockIconSecureFortezza.
secureLockIconSecureUnknownBits (see page 19528)	This is constant SHDocVw.secureLockIconSecureUnknownBits.
secureLockIconUnsecure (see page 19528)	This is constant SHDocVw.secureLockIconUnsecure.
SHDocVwMajorVersion (see page 19529)	This is constant SHDocVw.SHDocVwMajorVersion.
SHDocVwMinorVersion (see page 19529)	This is constant SHDocVw.SHDocVwMinorVersion.
SWC_3RDPARTY (see page 19529)	This is constant SHDocVw.SWC_3RDPARTY.
SWC_BROWSER (see page 19530)	This is constant SHDocVw.SWC_BROWSER.
SWC_CALLBACK (see page 19530)	This is constant SHDocVw.SWC_CALLBACK.
SWC_EXPLORER (see page 19530)	This is constant SHDocVw.SWC_EXPLORER.
SWFO_COOKIEPASSED (see page 19531)	This is constant SHDocVw.SWFO_COOKIEPASSED.
SWFO_INCLUDEPENDING (see page 19531)	This is constant SHDocVw.SWFO_INCLUDEPENDING.
SWFO_NEEDDISPATCH (see page 19531)	This is constant SHDocVw.SWFO_NEEDDISPATCH.

Interfaces

	Name	Description
	_SearchAssistantEvents (see page 19276)	This is class SHDocVw._SearchAssistantEvents.
	DShellNameSpaceEvents (see page 19278)	This is class SHDocVw.DShellNameSpaceEvents.
	DShellWindowsEvents (see page 19279)	This is class SHDocVw.DShellWindowsEvents.
	DWebBrowserEvents (see page 19281)	This is class SHDocVw.DWebBrowserEvents.
	DWebBrowserEvents2 (see page 19286)	This is class SHDocVw.DWebBrowserEvents2.
	IScriptErrorList (see page 19295)	This is class SHDocVw.IScriptErrorList.
	IScriptErrorListDisp (see page 19299)	This is class SHDocVw.IScriptErrorListDisp.
	ISearch (see page 19303)	This is class SHDocVw.ISearch.
	ISearchAssistantOC (see page 19306)	This is class SHDocVw.ISearchAssistantOC.
	ISearchAssistantOC2 (see page 19316)	This is class SHDocVw.ISearchAssistantOC2.
	ISearchAssistantOC2Disp (see page 19318)	This is class SHDocVw.ISearchAssistantOC2Disp.
	ISearchAssistantOC3 (see page 19326)	This is class SHDocVw.ISearchAssistantOC3.
	ISearchAssistantOC3Disp (see page 19329)	This is class SHDocVw.ISearchAssistantOC3Disp.
	ISearchAssistantOCDisp (see page 19337)	This is class SHDocVw.ISearchAssistantOCDisp.
	ISearchDisp (see page 19344)	This is class SHDocVw.ISearchDisp.
	ISearches (see page 19346)	This is class SHDocVw.ISearches.
	ISearchesDisp (see page 19348)	This is class SHDocVw.ISearchesDisp.
	IShellFavoritesNameSpace (see page 19350)	This is class SHDocVw.IShellFavoritesNameSpace.
	IShellFavoritesNameSpaceDisp (see page 19355)	This is class SHDocVw.IShellFavoritesNameSpaceDisp.
	IShellNameSpace (see page 19359)	This is class SHDocVw.IShellNameSpace.
	IShellNameSpaceDisp (see page 19368)	This is class SHDocVw.IShellNameSpaceDisp.
	IShellUIHelper (see page 19376)	This is class SHDocVw.IShellUIHelper.
	IShellUIHelperDisp (see page 19380)	This is class SHDocVw.IShellUIHelperDisp.
	IShellWindows (see page 19384)	This is class SHDocVw.IShellWindows.
	IShellWindowsDisp (see page 19388)	This is class SHDocVw.IShellWindowsDisp.
	IWebBrowser (see page 19392)	This is class SHDocVw.IWebBrowser.
	IWebBrowser2 (see page 19403)	This is class SHDocVw.IWebBrowser2.
	IWebBrowser2Disp (see page 19412)	This is class SHDocVw.IWebBrowser2Disp.
	IWebBrowserApp (see page 19425)	This is class SHDocVw.IWebBrowserApp.
	IWebBrowserAppDisp (see page 19434)	This is class SHDocVw.IWebBrowserAppDisp.
	IWebBrowserDisp (see page 19444)	This is class SHDocVw.IWebBrowserDisp.

Types

Name	Description
BrowserNavConstants (see page 19453)	This is type SHDocVw.BrowserNavConstants.
CommandStateChangeConstants (see page 19453)	This is type SHDocVw.CommandStateChangeConstants.
CScriptErrorList (see page 19453)	This is type SHDocVw.CScriptErrorList.

InternetExplorer (see page 19454)	This is type SHDocVw.InternetExplorer.
OLECMDEXECOPT (see page 19454)	OLECMDF (see page 19455) influences the way in which an OLECMDID (see page 19455) command is executed.
OLECMDF (see page 19455)	OLECMDF represents the type of support an Automation object provides for a specific command at a specific time.
OLECMDID (see page 19455)	OLECMDID represents a command that can be executed by an Automation object.
POleVariant1 (see page 19456)	This is type SHDocVw.POleVariant1.
RefreshConstants (see page 19456)	This is type SHDocVw.RefreshConstants.
SearchAssistantOC (see page 19456)	This is type SHDocVw.SearchAssistantOC.
SecureLockIconConstants (see page 19457)	This is type SHDocVw.SecureLockIconConstants.
ShellBrowserWindow (see page 19457)	This is type SHDocVw.ShellBrowserWindow.
ShellNameSpace (see page 19457)	This is type SHDocVw.ShellNameSpace.
ShellUIHelper (see page 19458)	This is type SHDocVw.ShellUIHelper.
ShellWindowFindWindowOptions (see page 19458)	This is type SHDocVw.ShellWindowFindWindowOptions.
ShellWindows (see page 19458)	This is type SHDocVw.ShellWindows.
ShellWindowTypeConstants (see page 19459)	This is type SHDocVw.ShellWindowTypeConstants.
tagREADYSTATE (see page 19459)	This is type SHDocVw.tagREADYSTATE.
TInternetExplorerBeforeNavigate2 (see page 19459)	This is type SHDocVw.TInternetExplorerBeforeNavigate2.
TInternetExplorerClientToHostWindow (see page 19460)	This is type SHDocVw.TInternetExplorerClientToHostWindow.
TInternetExplorerCommandStateChange (see page 19460)	This is type SHDocVw.TInternetExplorerCommandStateChange.
TInternetExplorerDocumentComplete (see page 19460)	This is type SHDocVw.TInternetExplorerDocumentComplete.
TInternetExplorerFileDownload (see page 19461)	This is type SHDocVw.TInternetExplorerFileDownload.
TInternetExplorerNavigateComplete2 (see page 19461)	This is type SHDocVw.TInternetExplorerNavigateComplete2.
TInternetExplorerNavigateError (see page 19461)	This is type SHDocVw.TInternetExplorerNavigateError.
TInternetExplorerNewWindow2 (see page 19462)	This is type SHDocVw.TInternetExplorerNewWindow2.
TInternetExplorerOnFullScreen (see page 19462)	This is type SHDocVw.TInternetExplorerOnFullScreen.
TInternetExplorerOnMenuBar (see page 19462)	This is type SHDocVw.TInternetExplorerOnMenuBar.
TInternetExplorerOnStatusBar (see page 19463)	This is type SHDocVw.TInternetExplorerOnStatusBar.
TInternetExplorerOnTheaterMode (see page 19463)	This is type SHDocVw.TInternetExplorerOnTheaterMode.
TInternetExplorerOnToolBar (see page 19463)	This is type SHDocVw.TInternetExplorerOnToolBar.
TInternetExplorerOnVisible (see page 19464)	This is type SHDocVw.TInternetExplorerOnVisible.
TInternetExplorerPrintTemplateInstantiation (see page 19464)	This is type SHDocVw.TInternetExplorerPrintTemplateInstantiation.
TInternetExplorerPrintTemplateTeardown (see page 19464)	This is type SHDocVw.TInternetExplorerPrintTemplateTeardown.
TInternetExplorerPrivacyImpactedStateChange (see page 19465)	This is type SHDocVw.TInternetExplorerPrivacyImpactedStateChange.
TInternetExplorerProgressChange (see page 19465)	This is type SHDocVw.TInternetExplorerProgressChange.
TInternetExplorerPropertyChange (see page 19465)	This is type SHDocVw.TInternetExplorerPropertyChange.
TInternetExplorerSetSecureLockIcon (see page 19466)	This is type SHDocVw.TInternetExplorerSetSecureLockIcon.
TInternetExplorerStatusTextChange (see page 19466)	This is type SHDocVw.TInternetExplorerStatusTextChange.
TInternetExplorerTitleChange (see page 19466)	This is type SHDocVw.TInternetExplorerTitleChange.
TInternetExplorerUpdatePageStatus (see page 19467)	This is type SHDocVw.TInternetExplorerUpdatePageStatus.
TInternetExplorerWindowClosing (see page 19467)	This is type SHDocVw.TInternetExplorerWindowClosing.
TInternetExplorerWindowSetHeight (see page 19467)	This is type SHDocVw.TInternetExplorerWindowSetHeight.
TInternetExplorerWindowSetLeft (see page 19468)	This is type SHDocVw.TInternetExplorerWindowSetLeft.
TInternetExplorerWindowSetResizable (see page 19468)	This is type SHDocVw.TInternetExplorerWindowSetResizable.
TInternetExplorerWindowSetTop (see page 19468)	This is type SHDocVw.TInternetExplorerWindowSetTop.
TInternetExplorerWindowSetWidth (see page 19469)	This is type SHDocVw.TInternetExplorerWindowSetWidth.
TShellNameSpaceFavoritesSelectionChange (see page 19469)	This is type SHDocVw.TShellNameSpaceFavoritesSelectionChange.
TShellWindowsWindowRegistered (see page 19470)	This is type SHDocVw.TShellWindowsWindowRegistered.
TShellWindowsWindowRevoked (see page 19470)	This is type SHDocVw.TShellWindowsWindowRevoked.
TWebBrowser_V1BeforeNavigate (see page 19470)	This is type SHDocVw.TWebBrowser_V1BeforeNavigate.
TWebBrowser_V1CommandStateChange (see page 19471)	This is type SHDocVw.TWebBrowser_V1CommandStateChange.
TWebBrowser_V1FrameBeforeNavigate (see page 19471)	This is type SHDocVw.TWebBrowser_V1FrameBeforeNavigate.
TWebBrowser_V1FrameNavigateComplete (see page 19471)	This is type SHDocVw.TWebBrowser_V1FrameNavigateComplete.
TWebBrowser_V1FrameNewWindow (see page 19472)	This is type SHDocVw.TWebBrowser_V1FrameNewWindow.

TWebBrowser_V1NavigateComplete (see page 19472)	This is type SHDocVw.TWebBrowser_V1NavigateComplete.
TWebBrowser_V1NewWindow (see page 19472)	This is type SHDocVw.TWebBrowser_V1NewWindow.
TWebBrowser_V1ProgressChange (see page 19473)	This is type SHDocVw.TWebBrowser_V1ProgressChange.
TWebBrowser_V1PropertyChange (see page 19473)	This is type SHDocVw.TWebBrowser_V1PropertyChange.
TWebBrowser_V1Quit (see page 19474)	This is type SHDocVw.TWebBrowser_V1Quit.
TWebBrowser_V1StatusTextChange (see page 19474)	This is type SHDocVw.TWebBrowser_V1StatusTextChange.
TWebBrowser_V1TitleChange (see page 19474)	This is type SHDocVw.TWebBrowser_V1TitleChange.
TWebBrowserBeforeNavigate2 (see page 19475)	This is type SHDocVw.TWebBrowserBeforeNavigate2.
TWebBrowserClientToHostWindow (see page 19475)	This is type SHDocVw.TWebBrowserClientToHostWindow.
TWebBrowserCommandStateChange (see page 19475)	This is type SHDocVw.TWebBrowserCommandStateChange.
TWebBrowserDocumentComplete (see page 19476)	This is type SHDocVw.TWebBrowserDocumentComplete.
TWebBrowserFileDownload (see page 19476)	This is type SHDocVw.TWebBrowserFileDownload.
TWebBrowserNavigateComplete2 (see page 19476)	This is type SHDocVw.TWebBrowserNavigateComplete2.
TWebBrowserNavigateError (see page 19477)	This is type SHDocVw.TWebBrowserNavigateError.
TWebBrowserNewWindow2 (see page 19477)	This is type SHDocVw.TWebBrowserNewWindow2.
TWebBrowserOnFullScreen (see page 19477)	This is type SHDocVw.TWebBrowserOnFullScreen.
TWebBrowserOnMenuBar (see page 19478)	This is type SHDocVw.TWebBrowserOnMenuBar.
TWebBrowserOnStatusBar (see page 19478)	This is type SHDocVw.TWebBrowserOnStatusBar.
TWebBrowserOnTheaterMode (see page 19478)	This is type SHDocVw.TWebBrowserOnTheaterMode.
TWebBrowserOnToolBar (see page 19479)	This is type SHDocVw.TWebBrowserOnToolBar.
TWebBrowserOnVisible (see page 19479)	This is type SHDocVw.TWebBrowserOnVisible.
TWebBrowserPrintTemplateInstantiation (see page 19479)	This is type SHDocVw.TWebBrowserPrintTemplateInstantiation.
TWebBrowserPrintTemplateTeardown (see page 19480)	This is type SHDocVw.TWebBrowserPrintTemplateTeardown.
TWebBrowserPrivacyImpactedStateChange (see page 19480)	This is type SHDocVw.TWebBrowserPrivacyImpactedStateChange.
TWebBrowserProgressChange (see page 19480)	This is type SHDocVw.TWebBrowserProgressChange.
TWebBrowserPropertyChange (see page 19481)	This is type SHDocVw.TWebBrowserPropertyChange.
TWebBrowserSetSecureLockIcon (see page 19481)	This is type SHDocVw.TWebBrowserSetSecureLockIcon.
TWebBrowserStatusTextChange (see page 19481)	This is type SHDocVw.TWebBrowserStatusTextChange.
TWebBrowserTitleChange (see page 19482)	This is type SHDocVw.TWebBrowserTitleChange.
TWebBrowserUpdatePageStatus (see page 19482)	This is type SHDocVw.TWebBrowserUpdatePageStatus.
TWebBrowserWindowClosing (see page 19482)	This is type SHDocVw.TWebBrowserWindowClosing.
TWebBrowserWindowSetHeight (see page 19483)	This is type SHDocVw.TWebBrowserWindowSetHeight.
TWebBrowserWindowSetLeft (see page 19483)	This is type SHDocVw.TWebBrowserWindowSetLeft.
TWebBrowserWindowSetResizable (see page 19483)	This is type SHDocVw.TWebBrowserWindowSetResizable.
TWebBrowserWindowSetTop (see page 19484)	This is type SHDocVw.TWebBrowserWindowSetTop.
TWebBrowserWindowSetWidth (see page 19484)	This is type SHDocVw.TWebBrowserWindowSetWidth.
WebBrowser (see page 19484)	This is type SHDocVw.WebBrowser.
WebBrowser_V1 (see page 19485)	This is type SHDocVw.WebBrowser_V1.

132.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	CoCScriptErrorList (see page 19045)	This is class SHDocVw.CoCScriptErrorList.
	CoInternetExplorer (see page 19046)	This is class SHDocVw.CoInternetExplorer.
	CoSearchAssistantOC (see page 19047)	This is class SHDocVw.CoSearchAssistantOC.
	CoShellBrowserWindow (see page 19049)	This is class SHDocVw.CoShellBrowserWindow.
	CoShellNameSpace (see page 19050)	This is class SHDocVw.CoShellNameSpace.
	CoShellUIHelper (see page 19051)	This is class SHDocVw.CoShellUIHelper.
	CoShellWindows (see page 19052)	This is class SHDocVw.CoShellWindows.
	TInternetExplorer (see page 19053)	This is class SHDocVw.TInternetExplorer.
	TShellNameSpace (see page 19101)	This is class SHDocVw.TShellNameSpace.
	TShellUIHelper (see page 19121)	This is class SHDocVw.TShellUIHelper.
	TShellWindows (see page 19135)	This is class SHDocVw.TShellWindows.
	TWebBrowser (see page 19147)	TWebBrowser provides access to the Web browser functionality of Microsoft's Shell Doc Object and Control Library (SHDOCVW.DLL).
	TWebBrowser_V1 (see page 19221)	This is class SHDocVw.TWebBrowser_V1.

132.1.1 CoCScriptErrorList Class

Class Hierarchy

SHDocVw.CoCScriptErrorList

Pascal

```
CoCScriptErrorList = class;
```

C++

```
class CoCScriptErrorList;
```

File

SHDocVw

Description

This is class SHDocVw.CoCScriptErrorList.

132.1.1.1 CoCScriptErrorList Members

The following tables list the members exposed by CoCScriptErrorList.

Public Methods

	Name	Description
	Create (see page 19046)	This is Create, a member of class CoCScriptErrorList.
	CreateRemote (see page 19046)	This is CreateRemote, a member of class CoCScriptErrorList.

132.1.1.2 CoCScriptErrorList Methods

The methods of the CoCScriptErrorList class are listed here.

Public Methods

	Name	Description
	Create (see page 19046)	This is Create, a member of class CoCScriptErrorList.
	CreateRemote (see page 19046)	This is CreateRemote, a member of class CoCScriptErrorList.

132.1.1.2.1 CoCScriptErrorList.Create Method

Pascal

```
class function Create: IScriptErrorList;
```

C++

```
__fastcall IScriptErrorList Create();
```

Description

This is Create, a member of class CoCScriptErrorList.

132.1.1.2.2 CoCScriptErrorList.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): IScriptErrorList;
```

C++

```
__fastcall IScriptErrorList CreateRemote(const AnsiString MachineName);
```

Description

This is CreateRemote, a member of class CoCScriptErrorList.

132.1.2 CoInternetExplorer Class

Class Hierarchy

Pascal

```
CoInternetExplorer = class;
```

C++

```
class CoInternetExplorer;
```

File

SHDocVw

Description

This is class SHDocVw.CoInternetExplorer.

132.1.2.1 CoInternetExplorer Members

The following tables list the members exposed by CoInternetExplorer.

Public Methods

	Name	Description
	Create (see page 19047)	This is Create, a member of class CoInternetExplorer.
	CreateRemote (see page 19047)	This is CreateRemote, a member of class CoInternetExplorer.

132.1.2.2 CoInternetExplorer Methods

The methods of the CoInternetExplorer class are listed here.

Public Methods

	Name	Description
	Create (see page 19047)	This is Create, a member of class CoInternetExplorer.
	CreateRemote (see page 19047)	This is CreateRemote, a member of class CoInternetExplorer.

132.1.2.2.1 CoInternetExplorer.Create Method

Pascal

```
class function Create: IWebBrowser2;
```

C++

```
__fastcall IWebBrowser2 Create();
```

Description

This is Create, a member of class CoInternetExplorer.

132.1.2.2.2 CoInternetExplorer.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): IWebBrowser2;
```

C++

```
__fastcall IWebBrowser2 CreateRemote(const AnsiString MachineName);
```

Description

This is CreateRemote, a member of class CoInternetExplorer.

132.1.3 CoSearchAssistantOC Class

Class Hierarchy

SHDocVw.CoSearchAssistantOC

Pascal

```
CoSearchAssistantOC = class;
```

C++

```
class CoSearchAssistantOC;
```

File

SHDocVw

Description

This is class SHDocVw.CoSearchAssistantOC.

132.1.3.1 CoSearchAssistantOC Members

The following tables list the members exposed by CoSearchAssistantOC.

Public Methods

	Name	Description
	Create (see page 19048)	This is Create, a member of class CoSearchAssistantOC.
	CreateRemote (see page 19048)	This is CreateRemote, a member of class CoSearchAssistantOC.

132.1.3.2 CoSearchAssistantOC Methods

The methods of the CoSearchAssistantOC class are listed here.

Public Methods

	Name	Description
	Create (see page 19048)	This is Create, a member of class CoSearchAssistantOC.
	CreateRemote (see page 19048)	This is CreateRemote, a member of class CoSearchAssistantOC.

132.1.3.2.1 CoSearchAssistantOC.Create Method

Pascal

```
class function Create: ISearchAssistantOC3;
```

C++

```
__fastcall ISearchAssistantOC3 Create();
```

Description

This is Create, a member of class CoSearchAssistantOC.

132.1.3.2.2 CoSearchAssistantOC.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): ISearchAssistantOC3;
```

C++

```
__fastcall ISearchAssistantOC3 CreateRemote(const AnsiString MachineName);
```

Description

This is CreateRemote, a member of class CoSearchAssistantOC.

132.1.4 CoShellBrowserWindow Class

Class Hierarchy

SHDocVw.CoShellBrowserWindow

Pascal

```
CoShellBrowserWindow = class;
```

C++

```
class CoShellBrowserWindow;
```

File

SHDocVw

Description

This is class SHDocVw.CoShellBrowserWindow.

132.1.4.1 CoShellBrowserWindow Members

The following tables list the members exposed by CoShellBrowserWindow.

Public Methods

	Name	Description
	Create (see page 19049)	This is Create, a member of class CoShellBrowserWindow.
	CreateRemote (see page 19050)	This is CreateRemote, a member of class CoShellBrowserWindow.

132.1.4.2 CoShellBrowserWindow Methods

The methods of the CoShellBrowserWindow class are listed here.

Public Methods

	Name	Description
	Create (see page 19049)	This is Create, a member of class CoShellBrowserWindow.
	CreateRemote (see page 19050)	This is CreateRemote, a member of class CoShellBrowserWindow.

132.1.4.2.1 CoShellBrowserWindow.Create Method

Pascal

```
class function Create: IWebBrowser2;
```

C++

```
__fastcall IWebBrowser2 Create();
```

Description

This is Create, a member of class CoShellBrowserWindow.

132.1.4.2.2 CoShellBrowserWindow.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): IWebBrowser2;
```

C++

```
__fastcall IWebBrowser2 CreateRemote(const AnsiString MachineName);
```

Description

This is CreateRemote, a member of class CoShellBrowserWindow.

132.1.5 CoShellNameSpace Class

Class Hierarchy

Pascal

```
CoShellNameSpace = class;
```

C++

```
class CoShellNameSpace;
```

File

SHDocVw

Description

This is class SHDocVw.CoShellNameSpace.

132.1.5.1 CoShellNameSpace Members

The following tables list the members exposed by CoShellNameSpace.

Public Methods

	Name	Description
	Create (see page 19051)	This is Create, a member of class CoShellNameSpace.
	CreateRemote (see page 19051)	This is CreateRemote, a member of class CoShellNameSpace.

132.1.5.2 CoShellNameSpace Methods

The methods of the CoShellNameSpace class are listed here.

Public Methods

	Name	Description
	Create (see page 19051)	This is Create, a member of class CoShellNameSpace.
	CreateRemote (see page 19051)	This is CreateRemote, a member of class CoShellNameSpace.

132.1.5.2.1 CoShellNameSpace.Create Method

Pascal

```
class function Create: IShellNameSpace;
```

C++

```
__fastcall IShellNameSpace Create();
```

Description

This is Create, a member of class CoShellNameSpace.

132.1.5.2.2 CoShellNameSpace.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): IShellNameSpace;
```

C++

```
__fastcall IShellNameSpace CreateRemote(const AnsiString MachineName);
```

Description

This is CreateRemote, a member of class CoShellNameSpace.

132.1.6 CoShellUIHelper Class

Class Hierarchy

SHDocVw.CoShellUIHelper

Pascal

```
CoShellUIHelper = class;
```

C++

```
class CoShellUIHelper;
```

File

SHDocVw

Description

This is class SHDocVw.CoShellUIHelper.

132.1.6.1 CoShellUIHelper Members

The following tables list the members exposed by CoShellUIHelper.

Public Methods

	Name	Description
	Create (see page 19052)	This is Create, a member of class CoShellUIHelper.
	CreateRemote (see page 19052)	This is CreateRemote, a member of class CoShellUIHelper.

132.1.6.2 CoShellUIHelper Methods

The methods of the CoShellUIHelper class are listed here.

Public Methods

	Name	Description
	Create (see page 19052)	This is Create, a member of class CoShellUIHelper.
	CreateRemote (see page 19052)	This is CreateRemote, a member of class CoShellUIHelper.

132.1.6.2.1 CoShellUIHelper.Create Method

Pascal

```
class function Create: IShellUIHelper;
```

C++

```
__fastcall IShellUIHelper Create();
```

Description

This is Create, a member of class CoShellUIHelper.

132.1.6.2.2 CoShellUIHelper.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): IShellUIHelper;
```

C++

```
__fastcall IShellUIHelper CreateRemote(const AnsiString MachineName);
```

Description

This is CreateRemote, a member of class CoShellUIHelper.

132.1.7 CoShellWindows Class

Class Hierarchy

SHDocVw.CoShellWindows

Pascal

```
CoShellWindows = class;
```

C++

```
class CoShellWindows;
```

File

SHDocVw

Description

This is class SHDocVw.CoShellWindows.

132.1.7.1 CoShellWindows Members

The following tables list the members exposed by CoShellWindows.

Public Methods

	Name	Description
	Create (see page 19053)	This is Create, a member of class CoShellWindows.
	CreateRemote (see page 19053)	This is CreateRemote, a member of class CoShellWindows.

132.1.7.2 CoShellWindows Methods

The methods of the CoShellWindows class are listed here.

Public Methods

	Name	Description
	Create (see page 19053)	This is Create, a member of class CoShellWindows.
	CreateRemote (see page 19053)	This is CreateRemote, a member of class CoShellWindows.

132.1.7.2.1 CoShellWindows.Create Method

Pascal

```
class function Create: IShellWindows;
```

C++

```
__fastcall IShellWindows Create();
```

Description

This is Create, a member of class CoShellWindows.

132.1.7.2.2 CoShellWindows.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): IShellWindows;
```

C++

```
__fastcall IShellWindows CreateRemote(const AnsiString MachineName);
```

Description

This is CreateRemote, a member of class CoShellWindows.

132.1.8 TInternetExplorer Class

Class Hierarchy

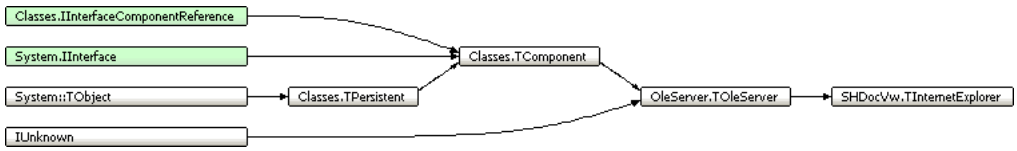

Pascal

```
TInternetExplorer = class(TOleServer);
```

C++

```
class TInternetExplorer : public TOleServer;
```

File

SHDocVw

Description

This is class SHDocVw.TInternetExplorer.

132.1.8.1 TInternetExplorer Members

The following tables list the members exposed by TInternetExplorer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_AddRef (see page 18104)	Increments the reference count for the number of interface references dependent on the OLE server.
	_Release (see page 2769)	Called when an application releases a component interface.
	_Release (see page 18104)	Decrements the reference count for the calling interface on a object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ConnectEvents (see page 18105)	Establishes a connection using the EventDispatch (see page 18111) property as an event sink.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DisconnectEvents (see page 18106)	Terminates a connection that was established using the ConnectEvents (see page 18105) method.
	Get_AddressBar (see page 19065)	This is Get_AddressBar, a member of class TInternetExplorer.
	Get_Application (see page 19065)	This is Get_Application, a member of class TInternetExplorer.
	Get_Busy (see page 19065)	This is Get_Busy, a member of class TInternetExplorer.
	Get_Container (see page 19065)	This is Get_Container, a member of class TInternetExplorer.
	Get_Document (see page 19065)	This is Get_Document, a member of class TInternetExplorer.
	Get_FullName (see page 19066)	This is Get_FullName, a member of class TInternetExplorer.
	Get_FullScreen (see page 19066)	This is Get_FullScreen, a member of class TInternetExplorer.
	Get_Height (see page 19066)	This is Get_Height, a member of class TInternetExplorer.
	Get_HWnd (see page 19066)	This is Get_HWnd, a member of class TInternetExplorer.
	Get_Left (see page 19067)	This is Get_Left, a member of class TInternetExplorer.
	Get_LocationName (see page 19067)	This is Get_LocationName, a member of class TInternetExplorer.
	Get_LocationURL (see page 19067)	This is Get_LocationURL, a member of class TInternetExplorer.
	Get_MenuBar (see page 19067)	This is Get_MenuBar, a member of class TInternetExplorer.
	Get_Name (see page 19067)	This is Get_Name, a member of class TInternetExplorer.
	Get_Offline (see page 19068)	This is Get_Offline, a member of class TInternetExplorer.
	Get_Parent (see page 19068)	This is Get_Parent, a member of class TInternetExplorer.
	Get_Path (see page 19068)	This is Get_Path, a member of class TInternetExplorer.

	Get_ReadyState (see page 19068)	This is Get_ReadyState, a member of class TInternetExplorer.
	Get_RegisterAsBrowser (see page 19069)	This is Get_RegisterAsBrowser, a member of class TInternetExplorer.
	Get_RegisterAsDropTarget (see page 19069)	This is Get_RegisterAsDropTarget, a member of class TInternetExplorer.
	Get_Resizable (see page 19069)	This is Get_Resizable, a member of class TInternetExplorer.
	Get_Silent (see page 19069)	This is Get_Silent, a member of class TInternetExplorer.
	Get_StatusBar (see page 19069)	This is Get_StatusBar, a member of class TInternetExplorer.
	Get_StatusText (see page 19070)	This is Get_StatusText, a member of class TInternetExplorer.
	Get_TheaterMode (see page 19070)	This is Get_TheaterMode, a member of class TInternetExplorer.
	Get_ToolBar (see page 19070)	This is Get_ToolBar, a member of class TInternetExplorer.
	Get_Top (see page 19070)	This is Get_Top, a member of class TInternetExplorer.
	Get_TopLevelContainer (see page 19071)	This is Get_TopLevelContainer, a member of class TInternetExplorer.
	Get_type_ (see page 19071)	This is Get_type_, a member of class TInternetExplorer.
	Get_Visible (see page 19071)	This is Get_Visible, a member of class TInternetExplorer.
	Get_Width (see page 19071)	This is Get_Width, a member of class TInternetExplorer.
	GetAutoConnect (see page 18106)	Returns the value of the AutoConnect (see page 18110) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetConnectKind (see page 18106)	Returns the value of the ConnectKind (see page 18110) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetServer (see page 18107)	Returns an interface for the COM server.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InitServerData (see page 19072)	Initializes the ServerData property to reflect the type library information of the COM server.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	InvokeEvent (see page 19073)	Dispatches an event from the COM server to the appropriate event handler.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 18108)	Connects to the COM server when the Loaded object is first loaded into memory if AutoConnect (see page 18110) is True true.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	QueryInterface (see page 18108)	Returns a reference to a specified interface if the OLE server supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	Set_AddressBar (see page 19078)	This is Set_AddressBar, a member of class TInternetExplorer.
	Set_FullScreen (see page 19078)	This is Set_FullScreen, a member of class TInternetExplorer.
	Set_Height (see page 19079)	This is Set_Height, a member of class TInternetExplorer.
	Set_Left (see page 19079)	This is Set_Left, a member of class TInternetExplorer.
	Set_MenuBar (see page 19079)	This is Set_MenuBar, a member of class TInternetExplorer.
	Set_Offline (see page 19079)	This is Set_Offline, a member of class TInternetExplorer.
	Set_RegisterAsBrowser (see page 19079)	This is Set_RegisterAsBrowser, a member of class TInternetExplorer.
	Set_RegisterAsDropTarget (see page 19080)	This is Set_RegisterAsDropTarget, a member of class TInternetExplorer.
	Set_Resizable (see page 19080)	This is Set_Resizable, a member of class TInternetExplorer.
	Set_Silent (see page 19080)	This is Set_Silent, a member of class TInternetExplorer.
	Set_StatusBar (see page 19080)	This is Set_StatusBar, a member of class TInternetExplorer.
	Set_StatusText (see page 19081)	This is Set_StatusText, a member of class TInternetExplorer.
	Set_TheaterMode (see page 19081)	This is Set_TheaterMode, a member of class TInternetExplorer.
	Set_ToolBar (see page 19081)	This is Set_ToolBar, a member of class TInternetExplorer.

	Set_Top (see page 19081)	This is Set_Top, a member of class TInternetExplorer.
	Set_Visible (see page 19081)	This is Set_Visible, a member of class TInternetExplorer.
	Set_Width (see page 19082)	This is Set_Width, a member of class TInternetExplorer.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoConnect (see page 18109)	Sets the value of the AutoConnect (see page 18110) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnectKind (see page 18109)	Sets the value of the ConnectKind (see page 18110) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	EventDispatch (see page 18111)	This is EventDispatch, a member of class TOleServer.
	ServerData (see page 18111)	Encodes information about the COM server to which ServerData connects.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToWindow (see page 19062)	This is ClientToWindow, a member of class TInternetExplorer.
V	Connect (see page 19062)	Establishes a connection to the COM server.
	ConnectTo (see page 19063)	This is ConnectTo, a member of class TInternetExplorer.
V	Create (see page 19059)	Allocates memory and constructs a safely initialized instance of a component.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 19060)	Disposes of the component and its owned components.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Disconnect (see page 19063)	Terminates the connection to the COM server.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecWB (see page 19064)	This is the overview for the ExecWB method overload.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProperty (see page 19071)	This is GetProperty, a member of class TInternetExplorer.
	GoBack (see page 19072)	This is GoBack, a member of class TInternetExplorer.
	GoForward (see page 19072)	This is GoForward, a member of class TInternetExplorer.
	GoHome (see page 19072)	This is GoHome, a member of class TInternetExplorer.
	GoSearch (see page 19072)	This is GoSearch, a member of class TInternetExplorer.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	Navigate (see page 19073)	This is the overview for the Navigate method overload.
	Navigate2 (see page 19075)	This is the overview for the Navigate2 method overload.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PutProperty (see page 19077)	This is PutProperty, a member of class TInternetExplorer.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	QueryStatusWB (see page 19077)	This is QueryStatusWB, a member of class TInternetExplorer.
	Quit (see page 19077)	This is Quit, a member of class TInternetExplorer.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 19077)	This is Refresh, a member of class TInternetExplorer.
	Refresh2 (see page 19077)	This is the overview for the Refresh2 method overload.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	ShowBrowserBar (see page 19082)	This is the overview for the ShowBrowserBar method overload.
	Stop (see page 19083)	This is Stop, a member of class TInternetExplorer.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	AddressBar (see page 19084)	This is AddressBar, a member of class TInternetExplorer.
R	Application (see page 19084)	This is Application, a member of class TInternetExplorer.
R	Busy (see page 19084)	This is Busy, a member of class TInternetExplorer.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Container (see page 19084)	This is Container, a member of class TInternetExplorer.
R	DefaultInterface (see page 19085)	This is DefaultInterface, a member of class TInternetExplorer.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Document (see page 19085)	This is Document, a member of class TInternetExplorer.
R	FullName (see page 19085)	This is FullName, a member of class TInternetExplorer.
	FullScreen (see page 19085)	This is FullScreen, a member of class TInternetExplorer.
	Height (see page 19086)	This is Height, a member of class TInternetExplorer.
R	HWnd (see page 19086)	This is HWnd, a member of class TInternetExplorer.
	Left (see page 19086)	This is Left, a member of class TInternetExplorer.
R	LocationName (see page 19086)	This is LocationName, a member of class TInternetExplorer.
R	LocationURL (see page 19086)	This is LocationURL, a member of class TInternetExplorer.
	MenuBar (see page 19087)	This is MenuBar, a member of class TInternetExplorer.
R	Name (see page 19087)	Specifies the name of the component as referenced in code.
	Offline (see page 19089)	This is Offline, a member of class TInternetExplorer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Parent (see page 19089)	This is Parent, a member of class TInternetExplorer.
R	Path (see page 19089)	This is Path, a member of class TInternetExplorer.
R	ReadyState (see page 19090)	This is ReadyState, a member of class TInternetExplorer.
	RegisterAsBrowser (see page 19090)	This is RegisterAsBrowser, a member of class TInternetExplorer.
	RegisterAsDropTarget (see page 19090)	This is RegisterAsDropTarget, a member of class TInternetExplorer.
	Resizable (see page 19090)	This is Resizable, a member of class TInternetExplorer.
	Silent (see page 19091)	This is Silent, a member of class TInternetExplorer.
	StatusBar (see page 19091)	This is StatusBar, a member of class TInternetExplorer.
	StatusText (see page 19091)	This is StatusText, a member of class TInternetExplorer.
	TheaterMode (see page 19091)	This is TheaterMode, a member of class TInternetExplorer.
	ToolBar (see page 19091)	This is ToolBar, a member of class TInternetExplorer.
	Top (see page 19092)	This is Top, a member of class TInternetExplorer.
R	TopLevelContainer (see page 19092)	This is TopLevelContainer, a member of class TInternetExplorer.
R	type_ (see page 19092)	This is type_, a member of class TInternetExplorer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 19092)	This is Visible, a member of class TInternetExplorer.
	Width (see page 19093)	This is Width, a member of class TInternetExplorer.

Published Events

	Name	Description
	OnBeforeNavigate2 (see page 19094)	This is OnBeforeNavigate2, a member of class TInternetExplorer.
	OnClientToHostWindow (see page 19094)	This is OnClientToHostWindow, a member of class TInternetExplorer.
	OnCommandStateChange (see page 19094)	This is OnCommandStateChange, a member of class TInternetExplorer.
	OnDocumentComplete (see page 19094)	This is OnDocumentComplete, a member of class TInternetExplorer.
	OnDownloadBegin (see page 19094)	This is OnDownloadBegin, a member of class TInternetExplorer.
	OnDownloadComplete (see page 19095)	This is OnDownloadComplete, a member of class TInternetExplorer.

	OnFileDownload (see page 19095)	This is OnFileDownload, a member of class TInternetExplorer.
	OnFullScreen (see page 19095)	This is OnFullScreen, a member of class TInternetExplorer.
	OnMenuBar (see page 19095)	This is OnMenuBar, a member of class TInternetExplorer.
	OnNavigateComplete2 (see page 19096)	This is OnNavigateComplete2, a member of class TInternetExplorer.
	OnNavigateError (see page 19096)	This is OnNavigateError, a member of class TInternetExplorer.
	OnNewWindow2 (see page 19096)	This is OnNewWindow2, a member of class TInternetExplorer.
	OnPrintTemplateInstantiation (see page 19096)	This is OnPrintTemplateInstantiation, a member of class TInternetExplorer.
	OnPrintTemplateTeardown (see page 19096)	This is OnPrintTemplateTeardown, a member of class TInternetExplorer.
	OnPrivacyImpactedStateChange (see page 19097)	This is OnPrivacyImpactedStateChange, a member of class TInternetExplorer.
	OnProgressChange (see page 19097)	This is OnProgressChange, a member of class TInternetExplorer.
	OnPropertyChange (see page 19097)	This is OnPropertyChange, a member of class TInternetExplorer.
	OnQuit (see page 19097)	This is OnQuit, a member of class TInternetExplorer.
	OnSetSecureLockIcon (see page 19098)	This is OnSetSecureLockIcon, a member of class TInternetExplorer.
	OnStatusBar (see page 19098)	This is OnStatusBar, a member of class TInternetExplorer.
	OnStatusTextChange (see page 19098)	This is OnStatusTextChange, a member of class TInternetExplorer.
	OnTheaterMode (see page 19098)	This is OnTheaterMode, a member of class TInternetExplorer.
	OnTitleChange (see page 19098)	This is OnTitleChange, a member of class TInternetExplorer.
	OnToolBar (see page 19099)	This is OnToolBar, a member of class TInternetExplorer.
	OnUpdatePageStatus (see page 19099)	This is OnUpdatePageStatus, a member of class TInternetExplorer.
	OnVisible (see page 19099)	This is OnVisible, a member of class TInternetExplorer.
	OnWindowClosing (see page 19099)	This is OnWindowClosing, a member of class TInternetExplorer.
	OnWindowSetHeight (see page 19100)	This is OnWindowSetHeight, a member of class TInternetExplorer.
	OnWindowSetLeft (see page 19100)	This is OnWindowSetLeft, a member of class TInternetExplorer.
	OnWindowSetResizable (see page 19100)	This is OnWindowSetResizable, a member of class TInternetExplorer.
	OnWindowSetTop (see page 19100)	This is OnWindowSetTop, a member of class TInternetExplorer.
	OnWindowSetWidth (see page 19100)	This is OnWindowSetWidth, a member of class TInternetExplorer.

Published Properties

	Name	Description
	AutoConnect (see page 18110)	Specifies whether the component automatically connects to the server.
	ConnectKind (see page 18110)	Indicates the type of process to which the ConnectKind component establishes a connection.
	RemoteMachineName (see page 18111)	Specifies the machine running the COM server when ConnectKind (see page 18110) is ckRemote.
	Tag (see page 2808)	Stores an integer value as part of a component.

132.1.8.2 TInternetExplorer.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TInternetExplorer(TComponent * AOwner);
```

Description

All objects have a Create method that constructs the object. TComponent redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner, as indicated by the AOwner parameter

sets the ComponentStyle property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components (see page 2799)

ComponentStyle (see page 2805)

Create (see page 22287)

Destroy (see page 2766)

Free (see page 22300)

Owner (see page 2807)

132.1.8.3 TInternetExplorer.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TInternetExplorer();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

Destroy (see page 22287)

DestroyComponents (see page 2771)

Destroying (see page 2772)

Free (see page 22300)

Notification (see page 2783)

Owner (see page 2807)

Release (see page 13885)

Components (see page 2799)

132.1.8.4 TInternetExplorer Methods

The methods of the TInternetExplorer class are listed here.

Protected Methods

	Name	Description
	Get_AddressBar (see page 19065)	This is Get_AddressBar, a member of class TInternetExplorer.
	Get_Application (see page 19065)	This is Get_Application, a member of class TInternetExplorer.
	Get_Busy (see page 19065)	This is Get_Busy, a member of class TInternetExplorer.
	Get_Container (see page 19065)	This is Get_Container, a member of class TInternetExplorer.
	Get_Document (see page 19065)	This is Get_Document, a member of class TInternetExplorer.
	Get_FullName (see page 19066)	This is Get_FullName, a member of class TInternetExplorer.
	Get_FullScreen (see page 19066)	This is Get_FullScreen, a member of class TInternetExplorer.
	Get_Height (see page 19066)	This is Get_Height, a member of class TInternetExplorer.
	Get_HWnd (see page 19066)	This is Get_HWnd, a member of class TInternetExplorer.
	Get_Left (see page 19067)	This is Get_Left, a member of class TInternetExplorer.
	Get_LocationName (see page 19067)	This is Get_LocationName, a member of class TInternetExplorer.
	Get_LocationURL (see page 19067)	This is Get_LocationURL, a member of class TInternetExplorer.
	Get_MenuBar (see page 19067)	This is Get_MenuBar, a member of class TInternetExplorer.
	Get_Name (see page 19067)	This is Get_Name, a member of class TInternetExplorer.
	Get_Offline (see page 19068)	This is Get_Offline, a member of class TInternetExplorer.
	Get_Parent (see page 19068)	This is Get_Parent, a member of class TInternetExplorer.
	Get_Path (see page 19068)	This is Get_Path, a member of class TInternetExplorer.
	Get_ReadyState (see page 19068)	This is Get_ReadyState, a member of class TInternetExplorer.
	Get_RegisterAsBrowser (see page 19069)	This is Get_RegisterAsBrowser, a member of class TInternetExplorer.
	Get_RegisterAsDropTarget (see page 19069)	This is Get_RegisterAsDropTarget, a member of class TInternetExplorer.
	Get_Resizable (see page 19069)	This is Get_Resizable, a member of class TInternetExplorer.
	Get_Silent (see page 19069)	This is Get_Silent, a member of class TInternetExplorer.
	Get_StatusBar (see page 19069)	This is Get_StatusBar, a member of class TInternetExplorer.
	Get_StatusText (see page 19070)	This is Get_StatusText, a member of class TInternetExplorer.
	Get_TheaterMode (see page 19070)	This is Get_TheaterMode, a member of class TInternetExplorer.
	Get_ToolBar (see page 19070)	This is Get_ToolBar, a member of class TInternetExplorer.
	Get_Top (see page 19070)	This is Get_Top, a member of class TInternetExplorer.
	Get_TopLevelContainer (see page 19071)	This is Get_TopLevelContainer, a member of class TInternetExplorer.
	Get_type_ (see page 19071)	This is Get_type_, a member of class TInternetExplorer.
	Get_Visible (see page 19071)	This is Get_Visible, a member of class TInternetExplorer.
	Get_Width (see page 19071)	This is Get_Width, a member of class TInternetExplorer.
V	InitServerData (see page 19072)	Initializes the ServerData property to reflect the type library information of the COM server.
V	InvokeEvent (see page 19073)	Dispatches an event from the COM server to the appropriate event handler.
	Set_AddressBar (see page 19078)	This is Set_AddressBar, a member of class TInternetExplorer.
	Set_FullScreen (see page 19078)	This is Set_FullScreen, a member of class TInternetExplorer.
	Set_Height (see page 19079)	This is Set_Height, a member of class TInternetExplorer.
	Set_Left (see page 19079)	This is Set_Left, a member of class TInternetExplorer.
	Set_MenuBar (see page 19079)	This is Set_MenuBar, a member of class TInternetExplorer.

	Set_Offline (see page 19079)	This is Set_Offline, a member of class TInternetExplorer.
	Set_RegisterAsBrowser (see page 19079)	This is Set_RegisterAsBrowser, a member of class TInternetExplorer.
	Set_RegisterAsDropTarget (see page 19080)	This is Set_RegisterAsDropTarget, a member of class TInternetExplorer.
	Set_Resizable (see page 19080)	This is Set_Resizable, a member of class TInternetExplorer.
	Set_Silent (see page 19080)	This is Set_Silent, a member of class TInternetExplorer.
	Set_StatusBar (see page 19080)	This is Set_StatusBar, a member of class TInternetExplorer.
	Set_StatusText (see page 19081)	This is Set_StatusText, a member of class TInternetExplorer.
	Set_TheaterMode (see page 19081)	This is Set_TheaterMode, a member of class TInternetExplorer.
	Set_ToolBar (see page 19081)	This is Set_ToolBar, a member of class TInternetExplorer.
	Set_Top (see page 19081)	This is Set_Top, a member of class TInternetExplorer.
	Set_Visible (see page 19081)	This is Set_Visible, a member of class TInternetExplorer.
	Set_Width (see page 19082)	This is Set_Width, a member of class TInternetExplorer.

Public Methods

	Name	Description
	ClientToWindow (see page 19062)	This is ClientToWindow, a member of class TInternetExplorer.
V	Connect (see page 19062)	Establishes a connection to the COM server.
	ConnectTo (see page 19063)	This is ConnectTo, a member of class TInternetExplorer.
V	Disconnect (see page 19063)	Terminates the connection to the COM server.
	ExecWB (see page 19064)	This is the overview for the ExecWB method overload.
	GetProperty (see page 19071)	This is GetProperty, a member of class TInternetExplorer.
	GoBack (see page 19072)	This is GoBack, a member of class TInternetExplorer.
	GoForward (see page 19072)	This is GoForward, a member of class TInternetExplorer.
	GoHome (see page 19072)	This is GoHome, a member of class TInternetExplorer.
	GoSearch (see page 19072)	This is GoSearch, a member of class TInternetExplorer.
	Navigate (see page 19073)	This is the overview for the Navigate method overload.
	Navigate2 (see page 19075)	This is the overview for the Navigate2 method overload.
	PutProperty (see page 19077)	This is PutProperty, a member of class TInternetExplorer.
	QueryStatusWB (see page 19077)	This is QueryStatusWB, a member of class TInternetExplorer.
	Quit (see page 19077)	This is Quit, a member of class TInternetExplorer.
	Refresh (see page 19077)	This is Refresh, a member of class TInternetExplorer.
	Refresh2 (see page 19077)	This is the overview for the Refresh2 method overload.
	ShowBrowserBar (see page 19082)	This is the overview for the ShowBrowserBar method overload.
	Stop (see page 19083)	This is Stop, a member of class TInternetExplorer.

132.1.8.4.1 TInternetExplorer.ClientToWindow Method

Pascal

```
procedure ClientToWindow(var pcx: SYSINT; var pcy: SYSINT);
```

C++

```
__fastcall ClientToWindow(SYSINT pcx, SYSINT pcy);
```

Description

This is ClientToWindow, a member of class TInternetExplorer.

132.1.8.4.2 TInternetExplorer.Connect Method

Establishes a connection to the COM server.

Pascal

```
procedure Connect; override;
```

C++

```
virtual __fastcall Connect();
```

Description

Call Connect to connect the Connect component to the COM server it represents. If the AutoConnect property is True true, the Connect component automatically calls Connect when the application is first run.

Do not use Connect if the value of ConnectKind is ckAttachToInterface. When attaching to an interface, use the ConnectTo method, which is introduced by descendant classes, instead.

If the value of ConnectKind is ckRemote, the RemoteMachineName property must be set before calling Connect.

Connect introduces Connect as an abstract a pure virtual method. Descendant classes must override this method to let the Connect component connect to a specific type of COM server.

See Also

AutoConnect (see page 18110)

ConnectKind (see page 18110)

RemoteMachineName (see page 18111)

Disconnect (see page 18105)

Using Component Wrappers

132.1.8.4.3 TInternetExplorer.ConnectTo Method

Pascal

```
procedure ConnectTo(svrIntf: IWebBrowser2);
```

C++

```
__fastcall ConnectTo(IWebBrowser2 svrIntf);
```

Description

This is ConnectTo, a member of class TInternetExplorer.

132.1.8.4.4 TInternetExplorer.Disconnect Method

Terminates the connection to the COM server.

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

Call Disconnect to terminate a connection formed using the Connect method.

Disconnect introduces Disconnect as an abstract a pure virtual method. Descendant classes must override this method to let the Disconnect component disconnect from a specific type of COM server.

See Also

Connect (see page 18104)

Using Component Wrappers

132.1.8.4.5 ExecWB Method

This is the overview for the ExecWB method overload.

Overload List

	Name	Description
	TInternetExplorer.ExecWB (OLECMDID, OLECMDEXECOPT) (see page 19064)	This is ExecWB, a member of class TInternetExplorer.
	TInternetExplorer.ExecWB (OLECMDID, OLECMDEXECOPT, OleVariant) (see page 19064)	This is ExecWB, a member of class TInternetExplorer.
	TInternetExplorer.ExecWB (OLECMDID, OLECMDEXECOPT, OleVariant, OleVariant) (see page 19064)	This is ExecWB, a member of class TInternetExplorer.

132.1.8.4.5.1 TInternetExplorer.ExecWB Method (OLECMDID, OLECMDEXECOPT)

Pascal

```
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT); overload;
```

C++

```
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt);
```

Description

This is ExecWB, a member of class TInternetExplorer.

132.1.8.4.5.2 TInternetExplorer.ExecWB Method (OLECMDID, OLECMDEXECOPT, OleVariant)

Pascal

```
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT; var pvaIn: OleVariant); overload;
```

C++

```
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, OleVariant pvaIn);
```

Description

This is ExecWB, a member of class TInternetExplorer.

132.1.8.4.5.3 TInternetExplorer.ExecWB Method (OLECMDID, OLECMDEXECOPT, OleVariant, OleVariant)

Pascal

```
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT; var pvaIn: OleVariant; var
pvaOut: OleVariant); overload;
```

C++

```
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, OleVariant pvaIn, OleVariant
pvaOut);
```

Description

This is ExecWB, a member of class TInternetExplorer.

132.1.8.4.6 TInternetExplorer.Get_AddressBar Method

Pascal

```
function Get_AddressBar: WordBool;
```

C++

```
__fastcall WordBool Get_AddressBar();
```

Description

This is Get_AddressBar, a member of class TInternetExplorer.

132.1.8.4.7 TInternetExplorer.Get_Application Method

Pascal

```
function Get_Application: IDispatch;
```

C++

```
__fastcall IDispatch Get_Application();
```

Description

This is Get_Application, a member of class TInternetExplorer.

132.1.8.4.8 TInternetExplorer.Get_Busy Method

Pascal

```
function Get_Busy: WordBool;
```

C++

```
__fastcall WordBool Get_Busy();
```

Description

This is Get_Busy, a member of class TInternetExplorer.

132.1.8.4.9 TInternetExplorer.Get_Container Method

Pascal

```
function Get_Container: IDispatch;
```

C++

```
__fastcall IDispatch Get_Container();
```

Description

This is Get_Container, a member of class TInternetExplorer.

132.1.8.4.10 TInternetExplorer.Get_Document Method

Pascal

```
function Get_Document: IDispatch;
```

C++

```
__fastcall IDispatch Get_Document();
```

Description

This is Get_Document, a member of class TInternetExplorer.

132.1.8.4.11 TInternetExplorer.Get_FullName Method

Pascal

```
function Get_FullName: WideString;
```

C++

```
__fastcall BSTR Get_FullName();
```

Description

This is Get_FullName, a member of class TInternetExplorer.

132.1.8.4.12 TInternetExplorer.Get_FullScreen Method

Pascal

```
function Get_FullScreen: WordBool;
```

C++

```
__fastcall WordBool Get_FullScreen();
```

Description

This is Get_FullScreen, a member of class TInternetExplorer.

132.1.8.4.13 TInternetExplorer.Get_Height Method

Pascal

```
function Get_Height: Integer;
```

C++

```
__fastcall int Get_Height();
```

Description

This is Get_Height, a member of class TInternetExplorer.

132.1.8.4.14 TInternetExplorer.Get_HWnd Method

Pascal

```
function Get_HWnd: Integer;
```

C++

```
__fastcall int Get_HWnd();
```

Description

This is Get_HWnd, a member of class TInternetExplorer.

132.1.8.4.15 TInternetExplorer.Get_Left Method

Pascal

```
function Get_Left: Integer;
```

C++

```
__fastcall int Get_Left();
```

Description

This is Get_Left, a member of class TInternetExplorer.

132.1.8.4.16 TInternetExplorer.Get_LocationName Method

Pascal

```
function Get_LocationName: WideString;
```

C++

```
__fastcall BSTR Get_LocationName();
```

Description

This is Get_LocationName, a member of class TInternetExplorer.

132.1.8.4.17 TInternetExplorer.Get_LocationURL Method

Pascal

```
function Get_LocationURL: WideString;
```

C++

```
__fastcall BSTR Get_LocationURL();
```

Description

This is Get_LocationURL, a member of class TInternetExplorer.

132.1.8.4.18 TInternetExplorer.Get_MenuBar Method

Pascal

```
function Get_MenuBar: WordBool;
```

C++

```
__fastcall WordBool Get_MenuBar();
```

Description

This is Get_MenuBar, a member of class TInternetExplorer.

132.1.8.4.19 TInternetExplorer.Get_Name Method

Pascal

```
function Get_Name: WideString;
```

C++

```
__fastcall BSTR Get_Name();
```

Description

This is Get_Name, a member of class TInternetExplorer.

132.1.8.4.20 TInternetExplorer.Get_Offline Method

Pascal

```
function Get_Offline: WordBool;
```

C++

```
__fastcall WordBool Get_Offline();
```

Description

This is Get_Offline, a member of class TInternetExplorer.

132.1.8.4.21 TInternetExplorer.Get_Parent Method

Pascal

```
function Get_Parent: IDispatch;
```

C++

```
__fastcall IDispatch Get_Parent();
```

Description

This is Get_Parent, a member of class TInternetExplorer.

132.1.8.4.22 TInternetExplorer.Get_Path Method

Pascal

```
function Get_Path: WideString;
```

C++

```
__fastcall BSTR Get_Path();
```

Description

This is Get_Path, a member of class TInternetExplorer.

132.1.8.4.23 TInternetExplorer.Get_ReadyState Method

Pascal

```
function Get_ReadyState: tagREADYSTATE;
```

C++

```
__fastcall tagREADYSTATE Get_ReadyState();
```

Description

This is Get_ReadyState, a member of class TInternetExplorer.

132.1.8.4.24 TInternetExplorer.Get_RegisterAsBrowser Method

Pascal

```
function Get_RegisterAsBrowser: WordBool;
```

C++

```
__fastcall WordBool Get_RegisterAsBrowser();
```

Description

This is Get_RegisterAsBrowser, a member of class TInternetExplorer.

132.1.8.4.25 TInternetExplorer.Get_RegisterAsDropTarget Method

Pascal

```
function Get_RegisterAsDropTarget: WordBool;
```

C++

```
__fastcall WordBool Get_RegisterAsDropTarget();
```

Description

This is Get_RegisterAsDropTarget, a member of class TInternetExplorer.

132.1.8.4.26 TInternetExplorer.Get_Resizable Method

Pascal

```
function Get_Resizable: WordBool;
```

C++

```
__fastcall WordBool Get_Resizable();
```

Description

This is Get_Resizable, a member of class TInternetExplorer.

132.1.8.4.27 TInternetExplorer.Get_Silent Method

Pascal

```
function Get_Silent: WordBool;
```

C++

```
__fastcall WordBool Get_Silent();
```

Description

This is Get_Silent, a member of class TInternetExplorer.

132.1.8.4.28 TInternetExplorer.Get_StatusBar Method

Pascal

```
function Get_StatusBar: WordBool;
```

C++

```
__fastcall WordBool Get_StatusBar();
```

Description

This is Get_StatusBar, a member of class TInternetExplorer.

132.1.8.4.29 TInternetExplorer.Get_StatusText Method

Pascal

```
function Get_StatusText: WideString;
```

C++

```
__fastcall BSTR Get_StatusText();
```

Description

This is Get_StatusText, a member of class TInternetExplorer.

132.1.8.4.30 TInternetExplorer.Get_TheaterMode Method

Pascal

```
function Get_TheaterMode: WordBool;
```

C++

```
__fastcall WordBool Get_TheaterMode();
```

Description

This is Get_TheaterMode, a member of class TInternetExplorer.

132.1.8.4.31 TInternetExplorer.Get_ToolBar Method

Pascal

```
function Get_ToolBar: SYSINT;
```

C++

```
__fastcall SYSINT Get_ToolBar();
```

Description

This is Get_ToolBar, a member of class TInternetExplorer.

132.1.8.4.32 TInternetExplorer.Get_Top Method

Pascal

```
function Get_Top: Integer;
```

C++

```
__fastcall int Get_Top();
```

Description

This is Get_Top, a member of class TInternetExplorer.

132.1.8.4.33 TInternetExplorer.Get_TopLevelContainer Method

Pascal

```
function Get_TopLevelContainer: WordBool;
```

C++

```
__fastcall WordBool Get_TopLevelContainer();
```

Description

This is Get_TopLevelContainer, a member of class TInternetExplorer.

132.1.8.4.34 TInternetExplorer.Get_type_ Method

Pascal

```
function Get_type_: WideString;
```

C++

```
__fastcall BSTR Get_type_();
```

Description

This is Get_type_, a member of class TInternetExplorer.

132.1.8.4.35 TInternetExplorer.Get_Visible Method

Pascal

```
function Get_Visible: WordBool;
```

C++

```
__fastcall WordBool Get_Visible();
```

Description

This is Get_Visible, a member of class TInternetExplorer.

132.1.8.4.36 TInternetExplorer.Get_Width Method

Pascal

```
function Get_Width: Integer;
```

C++

```
__fastcall int Get_Width();
```

Description

This is Get_Width, a member of class TInternetExplorer.

132.1.8.4.37 TInternetExplorer.GetProperty Method

Pascal

```
function GetProperty(const Property_: WideString): OleVariant;
```

C++

```
__fastcall OleVariant GetProperty(const BSTR Property_);
```

Description

This is GetProperty, a member of class TInternetExplorer.

132.1.8.4.38 TInternetExplorer.GoBack Method

Pascal

```
procedure GoBack;
```

C++

```
__fastcall GoBack();
```

Description

This is GoBack, a member of class TInternetExplorer.

132.1.8.4.39 TInternetExplorer.GoForward Method

Pascal

```
procedure GoForward;
```

C++

```
__fastcall GoForward();
```

Description

This is GoForward, a member of class TInternetExplorer.

132.1.8.4.40 TInternetExplorer.GoHome Method

Pascal

```
procedure GoHome;
```

C++

```
__fastcall GoHome();
```

Description

This is GoHome, a member of class TInternetExplorer.

132.1.8.4.41 TInternetExplorer.GoSearch Method

Pascal

```
procedure GoSearch;
```

C++

```
__fastcall GoSearch();
```

Description

This is GoSearch, a member of class TInternetExplorer.

132.1.8.4.42 TInternetExplorer.InitServerData Method

Initializes the ServerData property to reflect the type library information of the COM server.

Pascal

```
procedure InitServerData; override;
```

C++

```
virtual __fastcall InitServerData();
```

Description

The InitServerData constructor calls InitServerData to initialize ServerData with information from the COM server's type library.

InitServerData is an abstract a pure virtual method in InitServerData. Each InitServerData descendant implements this method so that the ServerData property reflects its particular COM server.

See Also

ServerData (see page 18111)

Create (see page 18101)

132.1.8.4.43 TInternetExplorer.InvokeEvent Method

Dispatches an event from the COM server to the appropriate event handler.

Pascal

```
procedure InvokeEvent(DispID: TDispID; var Params: TVariantArray); override;
```

C++

```
virtual __fastcall InvokeEvent(TDispID DispID, TVariantArray Params);
```

Description

InvokeEvent descendants use InvokeEvent internally to link their event handlers to events on the COM server.

DispID is the dispatch ID for the appropriate event handler.

Params passes the parameters of the event handler as an array of OleVariants.

As implemented in InvokeEvent, InvokeEvent does nothing, because InvokeEvent has no events. Descendants that surface events from a COM server override this method so that the event sink specified by EventDispatch can pass events on to the appropriate event handlers.

See Also

EventDispatch (see page 18111)

132.1.8.4.44 Navigate Method

This is the overview for the Navigate method overload.

Overload List

	Name	Description
	TInternetExplorer.Navigate (WideString) (see page 19074)	This is Navigate, a member of class TInternetExplorer.
	TInternetExplorer.Navigate (WideString, OleVariant) (see page 19074)	This is Navigate, a member of class TInternetExplorer.
	TInternetExplorer.Navigate (WideString, OleVariant, OleVariant) (see page 19074)	This is Navigate, a member of class TInternetExplorer.

	TInternetExplorer.Navigate (WideString, OleVariant, OleVariant, OleVariant) (see page 19074)	This is Navigate, a member of class TInternetExplorer.
	TInternetExplorer.Navigate (WideString, OleVariant, OleVariant, OleVariant, OleVariant) (see page 19075)	This is Navigate, a member of class TInternetExplorer.

132.1.8.4.44.1 TInternetExplorer.Navigate Method (WideString)

Pascal

```
procedure Navigate(const URL: WideString); overload;
```

C++

```
__fastcall Navigate(const BSTR URL);
```

Description

This is Navigate, a member of class TInternetExplorer.

132.1.8.4.44.2 TInternetExplorer.Navigate Method (WideString, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags);
```

Description

This is Navigate, a member of class TInternetExplorer.

132.1.8.4.44.3 TInternetExplorer.Navigate Method (WideString, OleVariant, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName);
```

Description

This is Navigate, a member of class TInternetExplorer.

132.1.8.4.44.4 TInternetExplorer.Navigate Method (WideString, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData);
```

Description

This is Navigate, a member of class TInternetExplorer.

132.1.8.4.44.5 TInternetExplorer.Navigate Method (WideString, OleVariant, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate, a member of class TInternetExplorer.

132.1.8.4.45 Navigate2 Method

This is the overview for the Navigate2 method overload.

Overload List

	Name	Description
	TInternetExplorer.Navigate2 (OleVariant) (see page 19075)	This is Navigate2, a member of class TInternetExplorer.
	TInternetExplorer.Navigate2 (OleVariant, OleVariant) (see page 19076)	This is Navigate2, a member of class TInternetExplorer.
	TInternetExplorer.Navigate2 (OleVariant, OleVariant, OleVariant) (see page 19076)	This is Navigate2, a member of class TInternetExplorer.
	TInternetExplorer.Navigate2 (OleVariant, OleVariant, OleVariant, OleVariant) (see page 19076)	This is Navigate2, a member of class TInternetExplorer.
	TInternetExplorer.Navigate2 (OleVariant, OleVariant, OleVariant, OleVariant, OleVariant) (see page 19076)	This is Navigate2, a member of class TInternetExplorer.

132.1.8.4.45.1 TInternetExplorer.Navigate2 Method (OleVariant)

Pascal

```
procedure Navigate2(var URL: OleVariant); overload;
```

C++

```
__fastcall Navigate2(OleVariant URL);
```

Description

This is Navigate2, a member of class TInternetExplorer.

132.1.8.4.45.2 TInternetExplorer.Navigate2 Method (OleVariant, OleVariant)

Pascal

```
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant); overload;
```

C++

```
__fastcall Navigate2(OleVariant URL, OleVariant Flags);
```

Description

This is Navigate2, a member of class TInternetExplorer.

132.1.8.4.45.3 TInternetExplorer.Navigate2 Method (OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant; var TargetFrameName:
OleVariant); overload;
```

C++

```
__fastcall Navigate2(OleVariant URL, OleVariant Flags, OleVariant TargetFrameName);
```

Description

This is Navigate2, a member of class TInternetExplorer.

132.1.8.4.45.4 TInternetExplorer.Navigate2 Method (OleVariant, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant; var TargetFrameName:
OleVariant; var PostData: OleVariant); overload;
```

C++

```
__fastcall Navigate2(OleVariant URL, OleVariant Flags, OleVariant TargetFrameName, OleVariant
PostData);
```

Description

This is Navigate2, a member of class TInternetExplorer.

132.1.8.4.45.5 TInternetExplorer.Navigate2 Method (OleVariant, OleVariant, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant; var TargetFrameName:
OleVariant; var PostData: OleVariant; var Headers: OleVariant); overload;
```

C++

```
__fastcall Navigate2(OleVariant URL, OleVariant Flags, OleVariant TargetFrameName, OleVariant
PostData, OleVariant Headers);
```

Description

This is Navigate2, a member of class TInternetExplorer.

132.1.8.4.46 TInternetExplorer.PutProperty Method

Pascal

```
procedure PutProperty(const Property_: WideString; vtValue: OleVariant);
```

C++

```
__fastcall PutProperty(const BSTR Property_, OleVariant vtValue);
```

Description

This is PutProperty, a member of class TInternetExplorer.

132.1.8.4.47 TInternetExplorer.QueryStatusWB Method

Pascal

```
function QueryStatusWB(cmdID: OLECMDID): OLECMDF;
```

C++

```
__fastcall OLECMDF QueryStatusWB(OLECMDID cmdID);
```

Description

This is QueryStatusWB, a member of class TInternetExplorer.

132.1.8.4.48 TInternetExplorer.Quit Method

Pascal

```
procedure Quit;
```

C++

```
__fastcall Quit();
```

Description

This is Quit, a member of class TInternetExplorer.

132.1.8.4.49 TInternetExplorer.Refresh Method

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

This is Refresh, a member of class TInternetExplorer.

132.1.8.4.50 Refresh2 Method

This is the overview for the Refresh2 method overload.

Overload List

	Name	Description
	TInternetExplorer.Refresh2 () (see page 19078)	This is Refresh2, a member of class TInternetExplorer.
	TInternetExplorer.Refresh2 (OleVariant) (see page 19078)	This is Refresh2, a member of class TInternetExplorer.

132.1.8.4.50.1 TInternetExplorer.Refresh2 Method ()

Pascal

```
procedure Refresh2; overload;
```

C++

```
__fastcall Refresh2();
```

Description

This is Refresh2, a member of class TInternetExplorer.

132.1.8.4.50.2 TInternetExplorer.Refresh2 Method (OleVariant)

Pascal

```
procedure Refresh2(var Level: OleVariant); overload;
```

C++

```
__fastcall Refresh2(OleVariant Level);
```

Description

This is Refresh2, a member of class TInternetExplorer.

132.1.8.4.51 TInternetExplorer.Set_AddressBar Method

Pascal

```
procedure Set_AddressBar(Value: WordBool);
```

C++

```
__fastcall Set_AddressBar(WordBool Value);
```

Description

This is Set_AddressBar, a member of class TInternetExplorer.

132.1.8.4.52 TInternetExplorer.Set_FullScreen Method

Pascal

```
procedure Set_FullScreen(pbFullScreen: WordBool);
```

C++

```
__fastcall Set_FullScreen(WordBool pbFullScreen);
```

Description

This is Set_FullScreen, a member of class TInternetExplorer.

132.1.8.4.53 TInternetExplorer.Set_Height Method

Pascal

```
procedure Set_Height(pl: Integer);
```

C++

```
__fastcall Set_Height(int pl);
```

Description

This is Set_Height, a member of class TInternetExplorer.

132.1.8.4.54 TInternetExplorer.Set_Left Method

Pascal

```
procedure Set_Left(pl: Integer);
```

C++

```
__fastcall Set_Left(int pl);
```

Description

This is Set_Left, a member of class TInternetExplorer.

132.1.8.4.55 TInternetExplorer.Set_MenuBar Method

Pascal

```
procedure Set_MenuBar(Value: WordBool);
```

C++

```
__fastcall Set_MenuBar(WordBool Value);
```

Description

This is Set_MenuBar, a member of class TInternetExplorer.

132.1.8.4.56 TInternetExplorer.Set_Offline Method

Pascal

```
procedure Set_Offline(pbOffline: WordBool);
```

C++

```
__fastcall Set_Offline(WordBool pbOffline);
```

Description

This is Set_Offline, a member of class TInternetExplorer.

132.1.8.4.57 TInternetExplorer.Set_RegisterAsBrowser Method

Pascal

```
procedure Set_RegisterAsBrowser(pbRegister: WordBool);
```

C++

```
__fastcall Set_RegisterAsBrowser(WordBool pbRegister);
```

Description

This is Set_RegisterAsBrowser, a member of class TInternetExplorer.

132.1.8.4.58 TInternetExplorer.Set_RegisterAsDropTarget Method

Pascal

```
procedure Set_RegisterAsDropTarget(pbRegister: WordBool);
```

C++

```
__fastcall Set_RegisterAsDropTarget(WordBool pbRegister);
```

Description

This is Set_RegisterAsDropTarget, a member of class TInternetExplorer.

132.1.8.4.59 TInternetExplorer.Set_Resizable Method

Pascal

```
procedure Set_Resizable(Value: WordBool);
```

C++

```
__fastcall Set_Resizable(WordBool Value);
```

Description

This is Set_Resizable, a member of class TInternetExplorer.

132.1.8.4.60 TInternetExplorer.Set_Silent Method

Pascal

```
procedure Set_Silent(pbSilent: WordBool);
```

C++

```
__fastcall Set_Silent(WordBool pbSilent);
```

Description

This is Set_Silent, a member of class TInternetExplorer.

132.1.8.4.61 TInternetExplorer.Set_StatusBar Method

Pascal

```
procedure Set_StatusBar(pBool: WordBool);
```

C++

```
__fastcall Set_StatusBar(WordBool pBool);
```

Description

This is Set_StatusBar, a member of class TInternetExplorer.

132.1.8.4.62 TInternetExplorer.Set_StatusText Method

Pascal

```
procedure Set_StatusText(const StatusText: WideString);
```

C++

```
__fastcall Set_StatusText(const BSTR StatusText);
```

Description

This is Set_StatusText, a member of class TInternetExplorer.

132.1.8.4.63 TInternetExplorer.Set_TheaterMode Method

Pascal

```
procedure Set_TheaterMode(pbRegister: WordBool);
```

C++

```
__fastcall Set_TheaterMode(WordBool pbRegister);
```

Description

This is Set_TheaterMode, a member of class TInternetExplorer.

132.1.8.4.64 TInternetExplorer.Set_ToolBar Method

Pascal

```
procedure Set_ToolBar(Value: SYSINT);
```

C++

```
__fastcall Set_ToolBar(SYSINT Value);
```

Description

This is Set_ToolBar, a member of class TInternetExplorer.

132.1.8.4.65 TInternetExplorer.Set_Top Method

Pascal

```
procedure Set_Top(pl: Integer);
```

C++

```
__fastcall Set_Top(int pl);
```

Description

This is Set_Top, a member of class TInternetExplorer.

132.1.8.4.66 TInternetExplorer.Set_Visible Method

Pascal

```
procedure Set_Visible(pBool: WordBool);
```

C++

```
__fastcall Set_Visible(WordBool pBool);
```

Description

This is Set_Visible, a member of class TInternetExplorer.

132.1.8.4.67 TInternetExplorer.Set_Width Method

Pascal

```
procedure Set_Width(pl: Integer);
```

C++

```
__fastcall Set_Width(int pl);
```

Description

This is Set_Width, a member of class TInternetExplorer.

132.1.8.4.68 ShowBrowserBar Method

This is the overview for the ShowBrowserBar method overload.

Overload List

	Name	Description
	TInternetExplorer.ShowBrowserBar (OleVariant) (see page 19082)	This is ShowBrowserBar, a member of class TInternetExplorer.
	TInternetExplorer.ShowBrowserBar (OleVariant, OleVariant) (see page 19082)	This is ShowBrowserBar, a member of class TInternetExplorer.
	TInternetExplorer.ShowBrowserBar (OleVariant, OleVariant, OleVariant) (see page 19083)	This is ShowBrowserBar, a member of class TInternetExplorer.

132.1.8.4.68.1 TInternetExplorer.ShowBrowserBar Method (OleVariant)

Pascal

```
procedure ShowBrowserBar(var pvaClsid: OleVariant); overload;
```

C++

```
__fastcall ShowBrowserBar(OleVariant pvaClsid);
```

Description

This is ShowBrowserBar, a member of class TInternetExplorer.

132.1.8.4.68.2 TInternetExplorer.ShowBrowserBar Method (OleVariant, OleVariant)

Pascal

```
procedure ShowBrowserBar(var pvaClsid: OleVariant; var pvarShow: OleVariant); overload;
```

C++

```
__fastcall ShowBrowserBar(OleVariant pvaClsid, OleVariant pvarShow);
```

Description

This is ShowBrowserBar, a member of class TInternetExplorer.

132.1.8.4.68.3 TInternetExplorer.ShowBrowserBar Method (OleVariant, OleVariant, OleVariant)

Pascal

```
procedure ShowBrowserBar(var pvaClsid: OleVariant; var pvarShow: OleVariant; var pvarSize:
OleVariant); overload;
```

C++

```
__fastcall ShowBrowserBar(OleVariant pvaClsid, OleVariant pvarShow, OleVariant pvarSize);
```

Description

This is ShowBrowserBar, a member of class TInternetExplorer.

132.1.8.4.69 TInternetExplorer.Stop Method

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

This is Stop, a member of class TInternetExplorer.

132.1.8.5 TInternetExplorer Properties

The properties of the TInternetExplorer class are listed here.

Public Properties

	Name	Description
	AddressBar (see page 19084)	This is AddressBar, a member of class TInternetExplorer.
R	Application (see page 19084)	This is Application, a member of class TInternetExplorer.
R	Busy (see page 19084)	This is Busy, a member of class TInternetExplorer.
R	Container (see page 19084)	This is Container, a member of class TInternetExplorer.
R	DefaultInterface (see page 19085)	This is DefaultInterface, a member of class TInternetExplorer.
R	Document (see page 19085)	This is Document, a member of class TInternetExplorer.
R	FullName (see page 19085)	This is FullName, a member of class TInternetExplorer.
	FullScreen (see page 19085)	This is FullScreen, a member of class TInternetExplorer.
	Height (see page 19086)	This is Height, a member of class TInternetExplorer.
R	HWnd (see page 19086)	This is HWnd, a member of class TInternetExplorer.
	Left (see page 19086)	This is Left, a member of class TInternetExplorer.
R	LocationName (see page 19086)	This is LocationName, a member of class TInternetExplorer.
R	LocationURL (see page 19086)	This is LocationURL, a member of class TInternetExplorer.
	MenuBar (see page 19087)	This is MenuBar, a member of class TInternetExplorer.
R	Name (see page 19087)	Specifies the name of the component as referenced in code.
	Offline (see page 19089)	This is Offline, a member of class TInternetExplorer.
R	Parent (see page 19089)	This is Parent, a member of class TInternetExplorer.
R	Path (see page 19089)	This is Path, a member of class TInternetExplorer.
R	ReadyState (see page 19090)	This is ReadyState, a member of class TInternetExplorer.
	RegisterAsBrowser (see page 19090)	This is RegisterAsBrowser, a member of class TInternetExplorer.
	RegisterAsDropTarget (see page 19090)	This is RegisterAsDropTarget, a member of class TInternetExplorer.
	Resizable (see page 19090)	This is Resizable, a member of class TInternetExplorer.

	Silent (see page 19091)	This is Silent, a member of class TInternetExplorer.
	StatusBar (see page 19091)	This is StatusBar, a member of class TInternetExplorer.
	StatusText (see page 19091)	This is StatusText, a member of class TInternetExplorer.
	TheaterMode (see page 19091)	This is TheaterMode, a member of class TInternetExplorer.
	ToolBar (see page 19091)	This is ToolBar, a member of class TInternetExplorer.
	Top (see page 19092)	This is Top, a member of class TInternetExplorer.
R	TopLevelContainer (see page 19092)	This is TopLevelContainer, a member of class TInternetExplorer.
R	type_ (see page 19092)	This is type_, a member of class TInternetExplorer.
	Visible (see page 19092)	This is Visible, a member of class TInternetExplorer.
	Width (see page 19093)	This is Width, a member of class TInternetExplorer.

132.1.8.5.1 TInternetExplorer.AddressBar Property

Pascal

```
property AddressBar: WordBool;
```

C++

```
__property WordBool AddressBar;
```

Description

This is AddressBar, a member of class TInternetExplorer.

132.1.8.5.2 TInternetExplorer.Application Property

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

This is Application, a member of class TInternetExplorer.

132.1.8.5.3 TInternetExplorer.Busy Property

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

This is Busy, a member of class TInternetExplorer.

132.1.8.5.4 TInternetExplorer.Container Property

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

This is Container, a member of class TInternetExplorer.

132.1.8.5.5 TInternetExplorer.DefaultInterface Property

Pascal

```
property DefaultInterface: IWebBrowser2;
```

C++

```
__property IWebBrowser2 DefaultInterface;
```

Description

This is DefaultInterface, a member of class TInternetExplorer.

132.1.8.5.6 TInternetExplorer.Document Property

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

This is Document, a member of class TInternetExplorer.

132.1.8.5.7 TInternetExplorer.FullName Property

Pascal

```
property FullName: WideString;
```

C++

```
__property BSTR FullName;
```

Description

This is FullName, a member of class TInternetExplorer.

132.1.8.5.8 TInternetExplorer.FullScreen Property

Pascal

```
property FullScreen: WordBool;
```

C++

```
__property WordBool FullScreen;
```

Description

This is FullScreen, a member of class TInternetExplorer.

132.1.8.5.9 TInternetExplorer.Height Property

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

This is Height, a member of class TInternetExplorer.

132.1.8.5.10 TInternetExplorer.HWnd Property

Pascal

```
property HWnd: Integer;
```

C++

```
__property int HWnd;
```

Description

This is HWnd, a member of class TInternetExplorer.

132.1.8.5.11 TInternetExplorer.Left Property

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

This is Left, a member of class TInternetExplorer.

132.1.8.5.12 TInternetExplorer.LocationName Property

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

This is LocationName, a member of class TInternetExplorer.

132.1.8.5.13 TInternetExplorer.LocationURL Property

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

This is LocationURL, a member of class TInternetExplorer.

132.1.8.5.14 TInternetExplorer.MenuBar Property

Pascal

```
property MenuBar: WordBool;
```

C++

```
__property WordBool MenuBar;
```

Description

This is MenuBar, a member of class TInternetExplorer.

132.1.8.5.15 TInternetExplorer.Name Property

Specifies the name of the component as referenced in code.

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

Use Name to change the name of a component to reflect its purpose in the current application. By default, the IDE assigns sequential names based on the type of the component, such as 'Button1', 'Button2', and so on.

Use Name to refer to the component in code.

Warning: Changing Name at runtime causes any references to the old name to become undefined. Any subsequent code that uses the old name will cause an exception.

C++ Examples:

```
/*
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < ComponentCount; i++)
    ListBox1->Items->InsertObject(0,
                                  Components[i]->Name,
                                  (TObject *)Components[i]);
}
```

```
void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  if (Button == mbRight)
  {
    TClass ClassRef;
    int Index = ListBox1->ItemAtPos(Point(X,Y), true);
    // only components that are controls have a position
    // make sure the component is a control
    for (ClassRef = ListBox1->Items->Objects[Index]->ClassType();
         ClassRef != NULL;
         ClassRef = ClassRef->ClassParent())
      if (String(ClassRef->ClassName()) == "TControl")
      {
        TControl *TheObject = (TControl *)ListBox1->Items->Objects[Index];
        StatusBar1->SimpleText =
          TheObject->Name + " is at (" +
          IntToStr(TheObject->Left) + ", " +
          IntToStr(TheObject->Top) + ")";
         break;
      }
    if (ClassRef == NULL) // if it wasn't a control
      MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to Form1.ComponentCount-1 do
    ListBox1.Items.InsertObject(
      0, Form1.Components[I].Name,
      Form1.Components[I] as TObject);
end;

procedure TForm1.ListBox1MouseUp(
  Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X,Y: integer);
var
  APoint: TPoint;
  Index: integer;
  TheObject: TControl;
begin
  if Button = mbRight then
  begin
  APoint.X := X;
  APoint.Y := Y;
  Index := ListBox1.ItemAtPos(APoint, True);
```

```
    if (ListBox1.Items.Objects[Index] is TControl) then
    begin
      TheObject :=
        (ListBox1.Items.Objects[Index] as TControl);
      StatusBar1.SimpleText := TheObject.Name + ' is at (' +
                               IntToStr(TheObject.Left) + ', ' +
                               IntToStr(TheObject.Top) + ') ';
    end
    else
      Beep;
    end;
  end;
```

See Also

ChangeName (see page 2770)

SetName (see page 2789)

ValidateRename (see page 2792)

Defining New Persistent Fields

Using TSQLQuery

Using TSQLStoredProc

Using TSQLTable

132.1.8.5.16 TInternetExplorer.Offline Property

Pascal

```
property Offline: WordBool;
```

C++

```
__property WordBool Offline;
```

Description

This is Offline, a member of class TInternetExplorer.

132.1.8.5.17 TInternetExplorer.Parent Property

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

This is Parent, a member of class TInternetExplorer.

132.1.8.5.18 TInternetExplorer.Path Property

Pascal

```
property Path: WideString;
```

C++

```
__property BSTR Path;
```

Description

This is Path, a member of class TInternetExplorer.

132.1.8.5.19 TInternetExplorer.ReadyState Property

Pascal

```
property ReadyState: tagREADYSTATE;
```

C++

```
__property tagREADYSTATE ReadyState;
```

Description

This is ReadyState, a member of class TInternetExplorer.

132.1.8.5.20 TInternetExplorer.RegisterAsBrowser Property

Pascal

```
property RegisterAsBrowser: WordBool;
```

C++

```
__property WordBool RegisterAsBrowser;
```

Description

This is RegisterAsBrowser, a member of class TInternetExplorer.

132.1.8.5.21 TInternetExplorer.RegisterAsDropTarget Property

Pascal

```
property RegisterAsDropTarget: WordBool;
```

C++

```
__property WordBool RegisterAsDropTarget;
```

Description

This is RegisterAsDropTarget, a member of class TInternetExplorer.

132.1.8.5.22 TInternetExplorer.Resizable Property

Pascal

```
property Resizable: WordBool;
```

C++

```
__property WordBool Resizable;
```

Description

This is Resizable, a member of class TInternetExplorer.

132.1.8.5.23 TInternetExplorer.Silent Property

Pascal

```
property Silent: WordBool;
```

C++

```
__property WordBool Silent;
```

Description

This is Silent, a member of class TInternetExplorer.

132.1.8.5.24 TInternetExplorer.StatusBar Property

Pascal

```
property StatusBar: WordBool;
```

C++

```
__property WordBool StatusBar;
```

Description

This is StatusBar, a member of class TInternetExplorer.

132.1.8.5.25 TInternetExplorer.StatusText Property

Pascal

```
property StatusText: WideString;
```

C++

```
__property BSTR StatusText;
```

Description

This is StatusText, a member of class TInternetExplorer.

132.1.8.5.26 TInternetExplorer.TheaterMode Property

Pascal

```
property TheaterMode: WordBool;
```

C++

```
__property WordBool TheaterMode;
```

Description

This is TheaterMode, a member of class TInternetExplorer.

132.1.8.5.27 TInternetExplorer.ToolBar Property

Pascal

```
property ToolBar: SYSINT;
```

C++

```
__property SYSINT ToolBar;
```

Description

This is ToolBar, a member of class TInternetExplorer.

132.1.8.5.28 TInternetExplorer.Top Property

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

This is Top, a member of class TInternetExplorer.

132.1.8.5.29 TInternetExplorer.TopLevelContainer Property

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

This is TopLevelContainer, a member of class TInternetExplorer.

132.1.8.5.30 TInternetExplorer.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class TInternetExplorer.

132.1.8.5.31 TInternetExplorer.Visible Property

Pascal

```
property Visible: WordBool;
```

C++

```
__property WordBool Visible;
```

Description

This is Visible, a member of class TInternetExplorer.

132.1.8.5.32 TInternetExplorer.Width Property

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

This is Width, a member of class TInternetExplorer.

132.1.8.6 TInternetExplorer Events

The events of the TInternetExplorer class are listed here.

Published Events

	Name	Description
	OnBeforeNavigate2 (see page 19094)	This is OnBeforeNavigate2, a member of class TInternetExplorer.
	OnClientToHostWindow (see page 19094)	This is OnClientToHostWindow, a member of class TInternetExplorer.
	OnCommandStateChange (see page 19094)	This is OnCommandStateChange, a member of class TInternetExplorer.
	OnDocumentComplete (see page 19094)	This is OnDocumentComplete, a member of class TInternetExplorer.
	OnDownloadBegin (see page 19094)	This is OnDownloadBegin, a member of class TInternetExplorer.
	OnDownloadComplete (see page 19095)	This is OnDownloadComplete, a member of class TInternetExplorer.
	OnFileDownload (see page 19095)	This is OnFileDownload, a member of class TInternetExplorer.
	OnFullScreen (see page 19095)	This is OnFullScreen, a member of class TInternetExplorer.
	OnMenuBar (see page 19095)	This is OnMenuBar, a member of class TInternetExplorer.
	OnNavigateComplete2 (see page 19096)	This is OnNavigateComplete2, a member of class TInternetExplorer.
	OnNavigateError (see page 19096)	This is OnNavigateError, a member of class TInternetExplorer.
	OnNewWindow2 (see page 19096)	This is OnNewWindow2, a member of class TInternetExplorer.
	OnPrintTemplateInstantiation (see page 19096)	This is OnPrintTemplateInstantiation, a member of class TInternetExplorer.
	OnPrintTemplateTeardown (see page 19096)	This is OnPrintTemplateTeardown, a member of class TInternetExplorer.
	OnPrivacyImpactedStateChange (see page 19097)	This is OnPrivacyImpactedStateChange, a member of class TInternetExplorer.
	OnProgressChange (see page 19097)	This is OnProgressChange, a member of class TInternetExplorer.
	OnPropertyChange (see page 19097)	This is OnPropertyChange, a member of class TInternetExplorer.
	OnQuit (see page 19097)	This is OnQuit, a member of class TInternetExplorer.
	OnSetSecureLockIcon (see page 19098)	This is OnSetSecureLockIcon, a member of class TInternetExplorer.
	OnStatusBar (see page 19098)	This is OnStatusBar, a member of class TInternetExplorer.
	OnStatusTextChange (see page 19098)	This is OnStatusTextChange, a member of class TInternetExplorer.
	OnTheaterMode (see page 19098)	This is OnTheaterMode, a member of class TInternetExplorer.
	OnTitleChange (see page 19098)	This is OnTitleChange, a member of class TInternetExplorer.
	OnToolBar (see page 19099)	This is OnToolBar, a member of class TInternetExplorer.
	OnUpdatePageStatus (see page 19099)	This is OnUpdatePageStatus, a member of class TInternetExplorer.
	OnVisible (see page 19099)	This is OnVisible, a member of class TInternetExplorer.
	OnWindowClosing (see page 19099)	This is OnWindowClosing, a member of class TInternetExplorer.
	OnWindowSetHeight (see page 19100)	This is OnWindowSetHeight, a member of class TInternetExplorer.
	OnWindowSetLeft (see page 19100)	This is OnWindowSetLeft, a member of class TInternetExplorer.
	OnWindowSetResizable (see page 19100)	This is OnWindowSetResizable, a member of class TInternetExplorer.
	OnWindowSetTop (see page 19100)	This is OnWindowSetTop, a member of class TInternetExplorer.
	OnWindowSetWidth (see page 19100)	This is OnWindowSetWidth, a member of class TInternetExplorer.

132.1.8.6.1 TInternetExplorer.OnBeforeNavigate2 Event

Pascal

```
property OnBeforeNavigate2: TInternetExplorerBeforeNavigate2;
```

C++

```
__property TInternetExplorerBeforeNavigate2 OnBeforeNavigate2;
```

Description

This is OnBeforeNavigate2, a member of class TInternetExplorer.

132.1.8.6.2 TInternetExplorer.OnClientToHostWindow Event

Pascal

```
property OnClientToHostWindow: TInternetExplorerClientToHostWindow;
```

C++

```
__property TInternetExplorerClientToHostWindow OnClientToHostWindow;
```

Description

This is OnClientToHostWindow, a member of class TInternetExplorer.

132.1.8.6.3 TInternetExplorer.OnCommandStateChange Event

Pascal

```
property OnCommandStateChange: TInternetExplorerCommandStateChange;
```

C++

```
__property TInternetExplorerCommandStateChange OnCommandStateChange;
```

Description

This is OnCommandStateChange, a member of class TInternetExplorer.

132.1.8.6.4 TInternetExplorer.OnDocumentComplete Event

Pascal

```
property OnDocumentComplete: TInternetExplorerDocumentComplete;
```

C++

```
__property TInternetExplorerDocumentComplete OnDocumentComplete;
```

Description

This is OnDocumentComplete, a member of class TInternetExplorer.

132.1.8.6.5 TInternetExplorer.OnDownloadBegin Event

Pascal

```
property OnDownloadBegin: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownloadBegin;
```

Description

This is OnDownloadBegin, a member of class TInternetExplorer.

132.1.8.6.6 TInternetExplorer.OnDownloadComplete Event

Pascal

```
property OnDownloadComplete: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownloadComplete;
```

Description

This is OnDownloadComplete, a member of class TInternetExplorer.

132.1.8.6.7 TInternetExplorer.OnFileDownload Event

Pascal

```
property OnFileDownload: TInternetExplorerFileDownload;
```

C++

```
__property TInternetExplorerFileDownload OnFileDownload;
```

Description

This is OnFileDownload, a member of class TInternetExplorer.

132.1.8.6.8 TInternetExplorer.OnFullScreen Event

Pascal

```
property OnFullScreen: TInternetExplorerOnFullScreen;
```

C++

```
__property TInternetExplorerOnFullScreen OnFullScreen;
```

Description

This is OnFullScreen, a member of class TInternetExplorer.

132.1.8.6.9 TInternetExplorer.OnMenuBar Event

Pascal

```
property OnMenuBar: TInternetExplorerOnMenuBar;
```

C++

```
__property TInternetExplorerOnMenuBar OnMenuBar;
```

Description

This is OnMenuBar, a member of class TInternetExplorer.

132.1.8.6.10 TInternetExplorer.OnNavigateComplete2 Event

Pascal

```
property OnNavigateComplete2: TInternetExplorerNavigateComplete2;
```

C++

```
__property TInternetExplorerNavigateComplete2 OnNavigateComplete2;
```

Description

This is OnNavigateComplete2, a member of class TInternetExplorer.

132.1.8.6.11 TInternetExplorer.OnNavigateError Event

Pascal

```
property OnNavigateError: TInternetExplorerNavigateError;
```

C++

```
__property TInternetExplorerNavigateError OnNavigateError;
```

Description

This is OnNavigateError, a member of class TInternetExplorer.

132.1.8.6.12 TInternetExplorer.OnNewWindow2 Event

Pascal

```
property OnNewWindow2: TInternetExplorerNewWindow2;
```

C++

```
__property TInternetExplorerNewWindow2 OnNewWindow2;
```

Description

This is OnNewWindow2, a member of class TInternetExplorer.

132.1.8.6.13 TInternetExplorer.OnPrintTemplateInstantiation Event

Pascal

```
property OnPrintTemplateInstantiation: TInternetExplorerPrintTemplateInstantiation;
```

C++

```
__property TInternetExplorerPrintTemplateInstantiation OnPrintTemplateInstantiation;
```

Description

This is OnPrintTemplateInstantiation, a member of class TInternetExplorer.

132.1.8.6.14 TInternetExplorer.OnPrintTemplateTeardown Event

Pascal

```
property OnPrintTemplateTeardown: TInternetExplorerPrintTemplateTeardown;
```

C++

```
__property TInternetExplorerPrintTemplateTeardown OnPrintTemplateTeardown;
```

Description

This is OnPrintTemplateTeardown, a member of class TInternetExplorer.

132.1.8.6.15 TInternetExplorer.OnPrivacyImpactedStateChange Event

Pascal

```
property OnPrivacyImpactedStateChange: TInternetExplorerPrivacyImpactedStateChange;
```

C++

```
__property TInternetExplorerPrivacyImpactedStateChange OnPrivacyImpactedStateChange;
```

Description

This is OnPrivacyImpactedStateChange, a member of class TInternetExplorer.

132.1.8.6.16 TInternetExplorer.OnProgressChange Event

Pascal

```
property OnProgressChange: TInternetExplorerProgressChange;
```

C++

```
__property TInternetExplorerProgressChange OnProgressChange;
```

Description

This is OnProgressChange, a member of class TInternetExplorer.

132.1.8.6.17 TInternetExplorer.OnPropertyChange Event

Pascal

```
property OnPropertyChange: TInternetExplorerPropertyChange;
```

C++

```
__property TInternetExplorerPropertyChange OnPropertyChange;
```

Description

This is OnPropertyChange, a member of class TInternetExplorer.

132.1.8.6.18 TInternetExplorer.OnQuit Event

Pascal

```
property OnQuit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnQuit;
```

Description

This is OnQuit, a member of class TInternetExplorer.

132.1.8.6.19 TInternetExplorer.OnSetSecureLockIcon Event

Pascal

```
property OnSetSecureLockIcon: TInternetExplorerSetSecureLockIcon;
```

C++

```
__property TInternetExplorerSetSecureLockIcon OnSetSecureLockIcon;
```

Description

This is OnSetSecureLockIcon, a member of class TInternetExplorer.

132.1.8.6.20 TInternetExplorer.OnStatusBar Event

Pascal

```
property OnStatusBar: TInternetExplorerOnStatusBar;
```

C++

```
__property TInternetExplorerOnStatusBar OnStatusBar;
```

Description

This is OnStatusBar, a member of class TInternetExplorer.

132.1.8.6.21 TInternetExplorer.OnStatusTextChange Event

Pascal

```
property OnStatusTextChange: TInternetExplorerStatusTextChange;
```

C++

```
__property TInternetExplorerStatusTextChange OnStatusTextChange;
```

Description

This is OnStatusTextChange, a member of class TInternetExplorer.

132.1.8.6.22 TInternetExplorer.OnTheaterMode Event

Pascal

```
property OnTheaterMode: TInternetExplorerOnTheaterMode;
```

C++

```
__property TInternetExplorerOnTheaterMode OnTheaterMode;
```

Description

This is OnTheaterMode, a member of class TInternetExplorer.

132.1.8.6.23 TInternetExplorer.OnTitleChange Event

Pascal

```
property OnTitleChange: TInternetExplorerTitleChange;
```

C++

```
__property TInternetExplorerTitleChange OnTitleChange;
```

Description

This is OnTitleChange, a member of class TInternetExplorer.

132.1.8.6.24 TInternetExplorer.OnToolBar Event

Pascal

```
property OnToolBar: TInternetExplorerOnToolBar;
```

C++

```
__property TInternetExplorerOnToolBar OnToolBar;
```

Description

This is OnToolBar, a member of class TInternetExplorer.

132.1.8.6.25 TInternetExplorer.OnUpdatePageStatus Event

Pascal

```
property OnUpdatePageStatus: TInternetExplorerUpdatePageStatus;
```

C++

```
__property TInternetExplorerUpdatePageStatus OnUpdatePageStatus;
```

Description

This is OnUpdatePageStatus, a member of class TInternetExplorer.

132.1.8.6.26 TInternetExplorer.OnVisible Event

Pascal

```
property OnVisible: TInternetExplorerOnVisible;
```

C++

```
__property TInternetExplorerOnVisible OnVisible;
```

Description

This is OnVisible, a member of class TInternetExplorer.

132.1.8.6.27 TInternetExplorer.OnWindowClosing Event

Pascal

```
property OnWindowClosing: TInternetExplorerWindowClosing;
```

C++

```
__property TInternetExplorerWindowClosing OnWindowClosing;
```

Description

This is OnWindowClosing, a member of class TInternetExplorer.

132.1.8.6.28 TInternetExplorer.OnWindowSetHeight Event

Pascal

```
property OnWindowSetHeight: TInternetExplorerWindowSetHeight;
```

C++

```
__property TInternetExplorerWindowSetHeight OnWindowSetHeight;
```

Description

This is OnWindowSetHeight, a member of class TInternetExplorer.

132.1.8.6.29 TInternetExplorer.OnWindowSetLeft Event

Pascal

```
property OnWindowSetLeft: TInternetExplorerWindowSetLeft;
```

C++

```
__property TInternetExplorerWindowSetLeft OnWindowSetLeft;
```

Description

This is OnWindowSetLeft, a member of class TInternetExplorer.

132.1.8.6.30 TInternetExplorer.OnWindowSetResizable Event

Pascal

```
property OnWindowSetResizable: TInternetExplorerWindowSetResizable;
```

C++

```
__property TInternetExplorerWindowSetResizable OnWindowSetResizable;
```

Description

This is OnWindowSetResizable, a member of class TInternetExplorer.

132.1.8.6.31 TInternetExplorer.OnWindowSetTop Event

Pascal

```
property OnWindowSetTop: TInternetExplorerWindowSetTop;
```

C++

```
__property TInternetExplorerWindowSetTop OnWindowSetTop;
```

Description

This is OnWindowSetTop, a member of class TInternetExplorer.

132.1.8.6.32 TInternetExplorer.OnWindowSetWidth Event

Pascal

```
property OnWindowSetWidth: TInternetExplorerWindowSetWidth;
```

C++

```
__property TInternetExplorerWindowSetWidth OnWindowSetWidth;
```

Description

This is OnWindowSetWidth, a member of class TInternetExplorer.

132.1.9 TShellNameSpace Class

Class Hierarchy

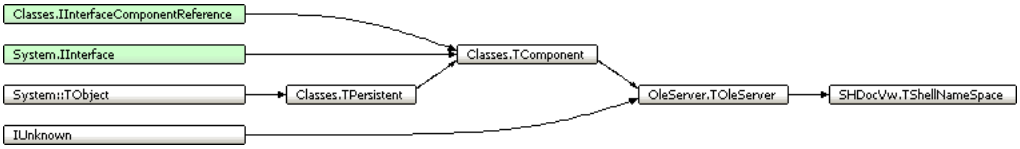

Pascal

```
TShellNameSpace = class(TOleServer);
```

C++

```
class TShellNameSpace : public TOleServer;
```

File

SHDocVw

Description

This is class SHDocVw.TShellNameSpace.

132.1.9.1 TShellNameSpace Members

The following tables list the members exposed by TShellNameSpace.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_AddRef (see page 18104)	Increments the reference count for the number of interface references dependent on the OLE server.
	_Release (see page 2769)	Called when an application releases a component interface.
	_Release (see page 18104)	Decrements the reference count for the calling interface on a object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ConnectEvents (see page 18105)	Establishes a connection using the EventDispatch (see page 18111) property as an event sink.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DisconnectEvents (see page 18106)	Terminates a connection that was established using the ConnectEvents (see page 18105) method.
	Get_Columns (see page 19110)	This is Get_Columns, a member of class TShellNameSpace.

	Get_CountViewTypes (see page 19110)	This is Get_CountViewTypes, a member of class TShellNameSpace.
	Get_Depth (see page 19110)	This is Get_Depth, a member of class TShellNameSpace.
	Get_EnumOptions (see page 19110)	This is Get_EnumOptions, a member of class TShellNameSpace.
	Get_Flags (see page 19110)	This is Get_Flags, a member of class TShellNameSpace.
	Get_Mode (see page 19111)	This is Get_Mode, a member of class TShellNameSpace.
	Get_Root (see page 19111)	This is Get_Root, a member of class TShellNameSpace.
	Get_SelectedItem (see page 19111)	This is Get_SelectedItem, a member of class TShellNameSpace.
	Get_SubscriptionsEnabled (see page 19111)	This is Get_SubscriptionsEnabled, a member of class TShellNameSpace.
	Get_TVFlags (see page 19112)	This is Get_TVFlags, a member of class TShellNameSpace.
	GetAutoConnect (see page 18106)	Returns the value of the AutoConnect (see page 18110) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetConnectKind (see page 18106)	Returns the value of the ConnectKind (see page 18110) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetServer (see page 18107)	Returns an interface for the COM server.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InitServerData (see page 19112)	Initializes the ServerData property to reflect the type library information of the COM server.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	InvokeEvent (see page 19113)	Dispatches an event from the COM server to the appropriate event handler.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 18108)	Connects to the COM server when the Loaded object is first loaded into memory if AutoConnect (see page 18110) is True true.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	QueryInterface (see page 18108)	Returns a reference to a specified interface if the OLE server supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	Set_Columns (see page 19114)	This is Set_Columns, a member of class TShellNameSpace.
	Set_Depth (see page 19115)	This is Set_Depth, a member of class TShellNameSpace.
	Set_EnumOptions (see page 19115)	This is Set_EnumOptions, a member of class TShellNameSpace.
	Set_Flags (see page 19115)	This is Set_Flags, a member of class TShellNameSpace.
	Set_Mode (see page 19115)	This is Set_Mode, a member of class TShellNameSpace.
	Set_Root (see page 19115)	This is Set_Root, a member of class TShellNameSpace.
	Set_SelectedItem (see page 19116)	This is Set_SelectedItem, a member of class TShellNameSpace.
	Set_TVFlags (see page 19116)	This is Set_TVFlags, a member of class TShellNameSpace.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoConnect (see page 18109)	Sets the value of the AutoConnect (see page 18110) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnectKind (see page 18109)	Sets the value of the ConnectKind (see page 18110) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	EventDispatch (see page 18111)	This is EventDispatch, a member of class TOleServer.
	ServerData (see page 18111)	Encodes information about the COM server to which ServerData connects.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Connect (see page 19107)	Establishes a connection to the COM server.
	ConnectTo (see page 19108)	This is ConnectTo, a member of class TShellNameSpace.
	Create (see page 19105)	Allocates memory and constructs a safely initialized instance of a component.
	CreateSubscriptionForSelection (see page 19108)	This is CreateSubscriptionForSelection, a member of class TShellNameSpace.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DeleteSubscriptionForSelection (see page 19108)	This is DeleteSubscriptionForSelection, a member of class TShellNameSpace.
	Destroy (see page 19106)	Disposes of the component and its owned components.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Disconnect (see page 19109)	Terminates the connection to the COM server.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	Expand (see page 19109)	This is Expand, a member of class TShellNameSpace.
	Export (see page 19109)	This is Export, a member of class TShellNameSpace.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.

	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	Import (see page 19112)	This is Import, a member of class TShellNameSpace.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InvokeContextMenuCommand (see page 19112)	This is InvokeContextMenuCommand, a member of class TShellNameSpace.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveSelectionDown (see page 19113)	This is MoveSelectionDown, a member of class TShellNameSpace.
	MoveSelectionTo (see page 19113)	This is MoveSelectionTo, a member of class TShellNameSpace.
	MoveSelectionUp (see page 19113)	This is MoveSelectionUp, a member of class TShellNameSpace.
	NewFolder (see page 19114)	This is NewFolder, a member of class TShellNameSpace.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	ResetSort (see page 19114)	This is ResetSort, a member of class TShellNameSpace.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SelectedItems (see page 19114)	This is SelectedItems, a member of class TShellNameSpace.
	SetRoot (see page 19116)	This is SetRoot, a member of class TShellNameSpace.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetViewType (see page 19116)	This is SetViewType, a member of class TShellNameSpace.
	Synchronize (see page 19117)	This is Synchronize, a member of class TShellNameSpace.
	UnselectAll (see page 19117)	This is UnselectAll, a member of class TShellNameSpace.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Columns (see page 19117)	This is Columns, a member of class TShellNameSpace.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	CountViewTypes (see page 19118)	This is CountViewTypes, a member of class TShellNameSpace.
R	DefaultInterface (see page 19118)	This is DefaultInterface, a member of class TShellNameSpace.
	Depth (see page 19118)	This is Depth, a member of class TShellNameSpace.

	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EnumOptions (see page 19118)	This is EnumOptions, a member of class TShellNameSpace.
	Flags (see page 19118)	This is Flags, a member of class TShellNameSpace.
	Mode (see page 19119)	This is Mode, a member of class TShellNameSpace.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Root (see page 19119)	This is Root, a member of class TShellNameSpace.
	SelectedItem (see page 19119)	This is SelectedItem, a member of class TShellNameSpace.
R	SubscriptionsEnabled (see page 19119)	This is SubscriptionsEnabled, a member of class TShellNameSpace.
	TVFlags (see page 19120)	This is TVFlags, a member of class TShellNameSpace.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnDoubleClick (see page 19120)	This is OnDoubleClick, a member of class TShellNameSpace.
	OnFavoritesSelectionChange (see page 19120)	This is OnFavoritesSelectionChange, a member of class TShellNameSpace.
	OnInitialized (see page 19120)	This is OnInitialized, a member of class TShellNameSpace.
	OnSelectionChange (see page 19121)	This is OnSelectionChange, a member of class TShellNameSpace.

Published Properties

	Name	Description
	AutoConnect (see page 18110)	Specifies whether the component automatically connects to the server.
	ConnectKind (see page 18110)	Indicates the type of process to which the ConnectKind component establishes a connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RemoteMachineName (see page 18111)	Specifies the machine running the COM server when ConnectKind (see page 18110) is ckRemote.
	Tag (see page 2808)	Stores an integer value as part of a component.

132.1.9.2 TShellNameSpace.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TShellNameSpace(TComponent * AOwner);
```

Description

All objects have a Create method that constructs the object. TComponent redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner, as indicated by the AOwner parameter

sets the ComponentStyle property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components (see page 2799)

ComponentStyle (see page 2805)

Create (see page 22287)

Destroy (see page 2766)

Free (see page 22300)

Owner (see page 2807)

132.1.9.3 TShellNameSpace.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TShellNameSpace();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

Destroy (see page 22287)

DestroyComponents (see page 2771)

Destroying (see page 2772)

Free (see page 22300)

Notification (see page 2783)

Owner (see page 2807)

Release (see page 13885)

Components (see page 2799)

132.1.9.4 TShellNameSpace Methods

The methods of the TShellNameSpace class are listed here.

Protected Methods

	Name	Description
	Get_Columns (see page 19110)	This is Get_Columns, a member of class TShellNameSpace.
	Get_CountViewTypes (see page 19110)	This is Get_CountViewTypes, a member of class TShellNameSpace.
	Get_Depth (see page 19110)	This is Get_Depth, a member of class TShellNameSpace.
	Get_EnumOptions (see page 19110)	This is Get_EnumOptions, a member of class TShellNameSpace.
	Get_Flags (see page 19110)	This is Get_Flags, a member of class TShellNameSpace.
	Get_Mode (see page 19111)	This is Get_Mode, a member of class TShellNameSpace.
	Get_Root (see page 19111)	This is Get_Root, a member of class TShellNameSpace.
	Get_SelectedItem (see page 19111)	This is Get_SelectedItem, a member of class TShellNameSpace.
	Get_SubscriptionsEnabled (see page 19111)	This is Get_SubscriptionsEnabled, a member of class TShellNameSpace.
	Get_TVFlags (see page 19112)	This is Get_TVFlags, a member of class TShellNameSpace.
V	InitServerData (see page 19112)	Initializes the ServerData property to reflect the type library information of the COM server.
V	InvokeEvent (see page 19113)	Dispatches an event from the COM server to the appropriate event handler.
	Set_Columns (see page 19114)	This is Set_Columns, a member of class TShellNameSpace.
	Set_Depth (see page 19115)	This is Set_Depth, a member of class TShellNameSpace.
	Set_EnumOptions (see page 19115)	This is Set_EnumOptions, a member of class TShellNameSpace.
	Set_Flags (see page 19115)	This is Set_Flags, a member of class TShellNameSpace.
	Set_Mode (see page 19115)	This is Set_Mode, a member of class TShellNameSpace.
	Set_Root (see page 19115)	This is Set_Root, a member of class TShellNameSpace.
	Set_SelectedItem (see page 19116)	This is Set_SelectedItem, a member of class TShellNameSpace.
	Set_TVFlags (see page 19116)	This is Set_TVFlags, a member of class TShellNameSpace.

Public Methods

	Name	Description
V	Connect (see page 19107)	Establishes a connection to the COM server.
	ConnectTo (see page 19108)	This is ConnectTo, a member of class TShellNameSpace.
	CreateSubscriptionForSelection (see page 19108)	This is CreateSubscriptionForSelection, a member of class TShellNameSpace.
	DeleteSubscriptionForSelection (see page 19108)	This is DeleteSubscriptionForSelection, a member of class TShellNameSpace.
V	Disconnect (see page 19109)	Terminates the connection to the COM server.
	Expand (see page 19109)	This is Expand, a member of class TShellNameSpace.
	Export (see page 19109)	This is Export, a member of class TShellNameSpace.
	Import (see page 19112)	This is Import, a member of class TShellNameSpace.
	InvokeContextMenuCommand (see page 19112)	This is InvokeContextMenuCommand, a member of class TShellNameSpace.
	MoveSelectionDown (see page 19113)	This is MoveSelectionDown, a member of class TShellNameSpace.
	MoveSelectionTo (see page 19113)	This is MoveSelectionTo, a member of class TShellNameSpace.
	MoveSelectionUp (see page 19113)	This is MoveSelectionUp, a member of class TShellNameSpace.
	NewFolder (see page 19114)	This is NewFolder, a member of class TShellNameSpace.
	ResetSort (see page 19114)	This is ResetSort, a member of class TShellNameSpace.
	SelectedItems (see page 19114)	This is SelectedItems, a member of class TShellNameSpace.
	SetRoot (see page 19116)	This is SetRoot, a member of class TShellNameSpace.
	SetViewType (see page 19116)	This is SetViewType, a member of class TShellNameSpace.
	Synchronize (see page 19117)	This is Synchronize, a member of class TShellNameSpace.
	UnselectAll (see page 19117)	This is UnselectAll, a member of class TShellNameSpace.

132.1.9.4.1 TShellNameSpace.Connect Method

Establishes a connection to the COM server.

Pascal

```
procedure Connect; override;
```

C++

```
virtual __fastcall Connect();
```

Description

Call Connect to connect the Connect component to the COM server it represents. If the AutoConnect property is True true, the Connect component automatically calls Connect when the application is first run.

Do not use Connect if the value of ConnectKind is ckAttachToInterface. When attaching to an interface, use the ConnectTo method, which is introduced by descendant classes, instead.

If the value of ConnectKind is ckRemote, the RemoteMachineName property must be set before calling Connect.

Connect introduces Connect as an abstract a pure virtual method. Descendant classes must override this method to let the Connect component connect to a specific type of COM server.

See Also

AutoConnect (see page 18110)

ConnectKind (see page 18110)

RemoteMachineName (see page 18111)

Disconnect (see page 18105)

Using Component Wrappers

132.1.9.4.2 TShellNameSpace.ConnectTo Method

Pascal

```
procedure ConnectTo(svrIntf: IShellNameSpace);
```

C++

```
__fastcall ConnectTo(IShellNameSpace svrIntf);
```

Description

This is ConnectTo, a member of class TShellNameSpace.

132.1.9.4.3 TShellNameSpace.CreateSubscriptionForSelection Method

Pascal

```
function CreateSubscriptionForSelection: WordBool;
```

C++

```
__fastcall WordBool CreateSubscriptionForSelection();
```

Description

This is CreateSubscriptionForSelection, a member of class TShellNameSpace.

132.1.9.4.4 TShellNameSpace.DeleteSubscriptionForSelection Method

Pascal

```
function DeleteSubscriptionForSelection: WordBool;
```

C++

```
__fastcall WordBool DeleteSubscriptionForSelection();
```

Description

This is DeleteSubscriptionForSelection, a member of class TShellNameSpace.

132.1.9.4.5 TShellNameSpace.Disconnect Method

Terminates the connection to the COM server.

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

Call Disconnect to terminate a connection formed using the Connect method.

Disconnect introduces Disconnect as an abstract a pure virtual method. Descendant classes must override this method to let the Disconnect component disconnect from a specific type of COM server.

See Also

Connect (see page 18104)

Using Component Wrappers

132.1.9.4.6 TShellNameSpace.Expand Method

Pascal

```
procedure Expand(var_: OleVariant; iDepth: SYSINT);
```

C++

```
__fastcall Expand(OleVariant var_, SYSINT iDepth);
```

Description

This is Expand, a member of class TShellNameSpace.

132.1.9.4.7 TShellNameSpace.Export Method

Pascal

```
procedure Export;
```

C++

```
__fastcall Export();
```

Description

This is Export, a member of class TShellNameSpace.

132.1.9.4.8 TShellNameSpace.Get_Columns Method

Pascal

```
function Get_Columns: WideString;
```

C++

```
__fastcall BSTR Get_Columns();
```

Description

This is Get_Columns, a member of class TShellNameSpace.

132.1.9.4.9 TShellNameSpace.Get_CountViewTypes Method

Pascal

```
function Get_CountViewTypes: SYSINT;
```

C++

```
__fastcall SYSINT Get_CountViewTypes();
```

Description

This is Get_CountViewTypes, a member of class TShellNameSpace.

132.1.9.4.10 TShellNameSpace.Get_Depth Method

Pascal

```
function Get_Depth: SYSINT;
```

C++

```
__fastcall SYSINT Get_Depth();
```

Description

This is Get_Depth, a member of class TShellNameSpace.

132.1.9.4.11 TShellNameSpace.Get_EnumOptions Method

Pascal

```
function Get_EnumOptions: Integer;
```

C++

```
__fastcall int Get_EnumOptions();
```

Description

This is Get_EnumOptions, a member of class TShellNameSpace.

132.1.9.4.12 TShellNameSpace.Get_Flags Method

Pascal

```
function Get_Flags: LongWord;
```

C++

```
__fastcall LongWord Get_Flags();
```

Description

This is Get_Flags, a member of class TShellNameSpace.

132.1.9.4.13 TShellNameSpace.Get_Mode Method

Pascal

```
function Get_Mode: SYSUINT;
```

C++

```
__fastcall SYSUINT Get_Mode();
```

Description

This is Get_Mode, a member of class TShellNameSpace.

132.1.9.4.14 TShellNameSpace.Get_Root Method

Pascal

```
function Get_Root: OleVariant;
```

C++

```
__fastcall OleVariant Get_Root();
```

Description

This is Get_Root, a member of class TShellNameSpace.

132.1.9.4.15 TShellNameSpace.Get_SelectedItem Method

Pascal

```
function Get_SelectedItem: IDispatch;
```

C++

```
__fastcall IDispatch Get_SelectedItem();
```

Description

This is Get_SelectedItem, a member of class TShellNameSpace.

132.1.9.4.16 TShellNameSpace.Get_SubscriptionsEnabled Method

Pascal

```
function Get_SubscriptionsEnabled: WordBool;
```

C++

```
__fastcall WordBool Get_SubscriptionsEnabled();
```

Description

This is Get_SubscriptionsEnabled, a member of class TShellNameSpace.

132.1.9.4.17 TShellNameSpace.Get_TVFlags Method

Pascal

```
function Get_TVFlags: LongWord;
```

C++

```
__fastcall LongWord Get_TVFlags();
```

Description

This is Get_TVFlags, a member of class TShellNameSpace.

132.1.9.4.18 TShellNameSpace.Import Method

Pascal

```
procedure Import;
```

C++

```
__fastcall Import();
```

Description

This is Import, a member of class TShellNameSpace.

132.1.9.4.19 TShellNameSpace.InitServerData Method

Initializes the ServerData property to reflect the type library information of the COM server.

Pascal

```
procedure InitServerData; override;
```

C++

```
virtual __fastcall InitServerData();
```

Description

The InitServerData constructor calls InitServerData to initialize ServerData with information from the COM server's type library.

InitServerData is an abstract a pure virtual method in InitServerData. Each InitServerData descendant implements this method so that the ServerData property reflects its particular COM server.

See Also

ServerData (see page 18111)

Create (see page 18101)

132.1.9.4.20 TShellNameSpace.InvokeContextMenuCommand Method

Pascal

```
procedure InvokeContextMenuCommand(const strCommand: WideString);
```

C++

```
__fastcall InvokeContextMenuCommand(const BSTR strCommand);
```

Description

This is InvokeContextMenuCommand, a member of class TShellNameSpace.

132.1.9.4.21 TShellNameSpace.InvokeEvent Method

Dispatches an event from the COM server to the appropriate event handler.

Pascal

```
procedure InvokeEvent(DispID: TDispID; var Params: TVariantArray); override;
```

C++

```
virtual __fastcall InvokeEvent(TDispID DispID, TVariantArray Params);
```

Description

InvokeEvent descendants use InvokeEvent internally to link their event handlers to events on the COM server.

DispID is the dispatch ID for the appropriate event handler.

Params passes the parameters of the event handler as an array of OleVariants.

As implemented in InvokeEvent, InvokeEvent does nothing, because InvokeEvent has no events. Descendants that surface events from a COM server override this method so that the event sink specified by EventDispatch can pass events on to the appropriate event handlers.

See Also

EventDispatch (see page 18111)

132.1.9.4.22 TShellNameSpace.MoveSelectionDown Method

Pascal

```
procedure MoveSelectionDown;
```

C++

```
__fastcall MoveSelectionDown();
```

Description

This is MoveSelectionDown, a member of class TShellNameSpace.

132.1.9.4.23 TShellNameSpace.MoveSelectionTo Method

Pascal

```
procedure MoveSelectionTo;
```

C++

```
__fastcall MoveSelectionTo();
```

Description

This is MoveSelectionTo, a member of class TShellNameSpace.

132.1.9.4.24 TShellNameSpace.MoveSelectionUp Method

Pascal

```
procedure MoveSelectionUp;
```

C++

```
__fastcall MoveSelectionUp();
```

Description

This is MoveSelectionUp, a member of class TShellNameSpace.

132.1.9.4.25 TShellNameSpace.NewFolder Method

Pascal

```
procedure NewFolder;
```

C++

```
__fastcall NewFolder();
```

Description

This is NewFolder, a member of class TShellNameSpace.

132.1.9.4.26 TShellNameSpace.ResetSort Method

Pascal

```
procedure ResetSort;
```

C++

```
__fastcall ResetSort();
```

Description

This is ResetSort, a member of class TShellNameSpace.

132.1.9.4.27 TShellNameSpace.SelectedItems Method

Pascal

```
function SelectedItems: IDispatch;
```

C++

```
__fastcall IDispatch SelectedItems();
```

Description

This is SelectedItems, a member of class TShellNameSpace.

132.1.9.4.28 TShellNameSpace.Set_Columns Method

Pascal

```
procedure Set_Columns(const bstrColumns: WideString);
```

C++

```
__fastcall Set_Columns(const BSTR bstrColumns);
```

Description

This is Set_Columns, a member of class TShellNameSpace.

132.1.9.4.29 TShellNameSpace.Set_Depth Method

Pascal

```
procedure Set_Depth(piDepth: SYSINT);
```

C++

```
__fastcall Set_Depth(SYSINT piDepth);
```

Description

This is Set_Depth, a member of class TShellNameSpace.

132.1.9.4.30 TShellNameSpace.Set_EnumOptions Method

Pascal

```
procedure Set_EnumOptions(pgrfEnumFlags: Integer);
```

C++

```
__fastcall Set_EnumOptions(int pgrfEnumFlags);
```

Description

This is Set_EnumOptions, a member of class TShellNameSpace.

132.1.9.4.31 TShellNameSpace.Set_Flags Method

Pascal

```
procedure Set_Flags(pdwFlags: LongWord);
```

C++

```
__fastcall Set_Flags(LongWord pdwFlags);
```

Description

This is Set_Flags, a member of class TShellNameSpace.

132.1.9.4.32 TShellNameSpace.Set_Mode Method

Pascal

```
procedure Set_Mode(puMode: SYSUINT);
```

C++

```
__fastcall Set_Mode(SYSUINT puMode);
```

Description

This is Set_Mode, a member of class TShellNameSpace.

132.1.9.4.33 TShellNameSpace.Set_Root Method

Pascal

```
procedure Set_Root(pvar: OleVariant);
```

C++

```
__fastcall Set_Root(OleVariant pvar);
```

Description

This is Set_Root, a member of class TShellNameSpace.

132.1.9.4.34 TShellNameSpace.Set_SelectedItem Method

Pascal

```
procedure Set_SelectedItem(const pItem: IDispatch);
```

C++

```
__fastcall Set_SelectedItem(const IDispatch pItem);
```

Description

This is Set_SelectedItem, a member of class TShellNameSpace.

132.1.9.4.35 TShellNameSpace.Set_TVFlags Method

Pascal

```
procedure Set_TVFlags(dwFlags: LongWord);
```

C++

```
__fastcall Set_TVFlags(LongWord dwFlags);
```

Description

This is Set_TVFlags, a member of class TShellNameSpace.

132.1.9.4.36 TShellNameSpace.SetRoot Method

Pascal

```
procedure SetRoot(const bstrFullPath: WideString);
```

C++

```
__fastcall SetRoot(const BSTR bstrFullPath);
```

Description

This is SetRoot, a member of class TShellNameSpace.

132.1.9.4.37 TShellNameSpace.SetViewType Method

Pascal

```
procedure SetViewType(iType: SYSINT);
```

C++

```
__fastcall SetViewType(SYSINT iType);
```

Description

This is SetViewType, a member of class TShellNameSpace.

132.1.9.4.38 TShellNameSpace.Synchronize Method

Pascal

```
procedure Synchronize;
```

C++

```
__fastcall Synchronize();
```

Description

This is Synchronize, a member of class TShellNameSpace.

132.1.9.4.39 TShellNameSpace.UnselectAll Method

Pascal

```
procedure UnselectAll;
```

C++

```
__fastcall UnselectAll();
```

Description

This is UnselectAll, a member of class TShellNameSpace.

132.1.9.5 TShellNameSpace Properties

The properties of the TShellNameSpace class are listed here.

Public Properties

	Name	Description
	Columns (see page 19117)	This is Columns, a member of class TShellNameSpace.
R	CountViewTypes (see page 19118)	This is CountViewTypes, a member of class TShellNameSpace.
R	DefaultInterface (see page 19118)	This is DefaultInterface, a member of class TShellNameSpace.
	Depth (see page 19118)	This is Depth, a member of class TShellNameSpace.
	EnumOptions (see page 19118)	This is EnumOptions, a member of class TShellNameSpace.
	Flags (see page 19118)	This is Flags, a member of class TShellNameSpace.
	Mode (see page 19119)	This is Mode, a member of class TShellNameSpace.
	Root (see page 19119)	This is Root, a member of class TShellNameSpace.
	SelectedItem (see page 19119)	This is SelectedItem, a member of class TShellNameSpace.
R	SubscriptionsEnabled (see page 19119)	This is SubscriptionsEnabled, a member of class TShellNameSpace.
	TVFlags (see page 19120)	This is TVFlags, a member of class TShellNameSpace.

132.1.9.5.1 TShellNameSpace.Columns Property

Pascal

```
property Columns: WideString;
```

C++

```
__property BSTR Columns;
```

Description

This is Columns, a member of class TShellNameSpace.

132.1.9.5.2 TShellNameSpace.CountViewTypes Property

Pascal

```
property CountViewTypes: SYSINT;
```

C++

```
__property SYSINT CountViewTypes;
```

Description

This is CountViewTypes, a member of class TShellNameSpace.

132.1.9.5.3 TShellNameSpace.DefaultInterface Property

Pascal

```
property DefaultInterface: IShellNameSpace;
```

C++

```
__property IShellNameSpace DefaultInterface;
```

Description

This is DefaultInterface, a member of class TShellNameSpace.

132.1.9.5.4 TShellNameSpace.Depth Property

Pascal

```
property Depth: SYSINT;
```

C++

```
__property SYSINT Depth;
```

Description

This is Depth, a member of class TShellNameSpace.

132.1.9.5.5 TShellNameSpace.EnumOptions Property

Pascal

```
property EnumOptions: Integer;
```

C++

```
__property int EnumOptions;
```

Description

This is EnumOptions, a member of class TShellNameSpace.

132.1.9.5.6 TShellNameSpace.Flags Property

Pascal

```
property Flags: LongWord;
```

C++

```
__property LongWord Flags;
```

Description

This is Flags, a member of class TShellNameSpace.

132.1.9.5.7 TShellNameSpace.Mode Property

Pascal

```
property Mode: SYSUINT;
```

C++

```
__property SYSUINT Mode;
```

Description

This is Mode, a member of class TShellNameSpace.

132.1.9.5.8 TShellNameSpace.Root Property

Pascal

```
property Root: OleVariant;
```

C++

```
__property OleVariant Root;
```

Description

This is Root, a member of class TShellNameSpace.

132.1.9.5.9 TShellNameSpace.SelectedItem Property

Pascal

```
property SelectedItem: IDispatch;
```

C++

```
__property IDispatch SelectedItem;
```

Description

This is SelectedItem, a member of class TShellNameSpace.

132.1.9.5.10 TShellNameSpace.SubscriptionsEnabled Property

Pascal

```
property SubscriptionsEnabled: WordBool;
```

C++

```
__property WordBool SubscriptionsEnabled;
```

Description

This is SubscriptionsEnabled, a member of class TShellNameSpace.

132.1.9.5.11 TShellNameSpace.TVFlags Property

Pascal

```
property TVFlags: LongWord;
```

C++

```
__property LongWord TVFlags;
```

Description

This is TVFlags, a member of class TShellNameSpace.

132.1.9.6 TShellNameSpace Events

The events of the TShellNameSpace class are listed here.

Published Events

	Name	Description
	OnDoubleClick (see page 19120)	This is OnDoubleClick, a member of class TShellNameSpace.
	OnFavoritesSelectionChange (see page 19120)	This is OnFavoritesSelectionChange, a member of class TShellNameSpace.
	OnInitialized (see page 19120)	This is OnInitialized, a member of class TShellNameSpace.
	OnSelectionChange (see page 19121)	This is OnSelectionChange, a member of class TShellNameSpace.

132.1.9.6.1 TShellNameSpace.OnDoubleClick Event

Pascal

```
property OnDoubleClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDoubleClick;
```

Description

This is OnDoubleClick, a member of class TShellNameSpace.

132.1.9.6.2 TShellNameSpace.OnFavoritesSelectionChange Event

Pascal

```
property OnFavoritesSelectionChange: TShellNameSpaceFavoritesSelectionChange;
```

C++

```
__property TShellNameSpaceFavoritesSelectionChange OnFavoritesSelectionChange;
```

Description

This is OnFavoritesSelectionChange, a member of class TShellNameSpace.

132.1.9.6.3 TShellNameSpace.OnInitialized Event

Pascal

```
property OnInitialized: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnInitialized;
```

Description

This is OnInitialized, a member of class TShellNameSpace.

132.1.9.6.4 TShellNameSpace.OnSelectionChange Event

Pascal

```
property OnSelectionChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSelectionChange;
```

Description

This is OnSelectionChange, a member of class TShellNameSpace.

132.1.10 TShellUIHelper Class

Class Hierarchy

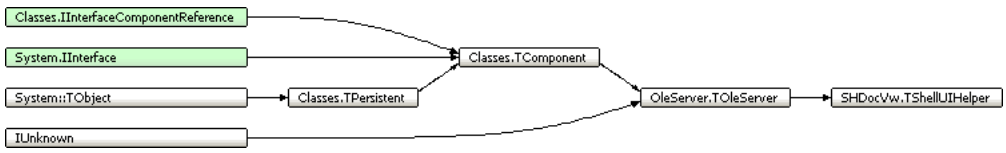

Pascal

```
TShellUIHelper = class(TOleServer);
```

C++

```
class TShellUIHelper : public TOleServer;
```

File

SHDocVw

Description

This is class SHDocVw.TShellUIHelper.

132.1.10.1 TShellUIHelper Members

The following tables list the members exposed by TShellUIHelper.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.

	_AddRef (see page 18104)	Increments the reference count for the number of interface references dependent on the OLE server.
	_Release (see page 2769)	Called when an application releases a component interface.
	_Release (see page 18104)	Decrements the reference count for the calling interface on a object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ConnectEvents (see page 18105)	Establishes a connection using the EventDispatch (see page 18111) property as an event sink.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DisconnectEvents (see page 18106)	Terminates a connection that was established using the ConnectEvents (see page 18105) method.
	GetAutoConnect (see page 18106)	Returns the value of the AutoConnect (see page 18110) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetConnectKind (see page 18106)	Returns the value of the ConnectKind (see page 18110) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetServer (see page 18107)	Returns an interface for the COM server.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InitServerData (see page 19133)	Initializes the ServerData property to reflect the type library information of the COM server.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	InvokeEvent (see page 18107)	Dispatches an event from the COM server to the appropriate event handler.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 18108)	Connects to the COM server when the Loaded object is first loaded into memory if AutoConnect (see page 18110) is True true.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	QueryInterface (see page 18108)	Returns a reference to a specified interface if the OLE server supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoConnect (see page 18109)	Sets the value of the AutoConnect (see page 18110) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnectKind (see page 18109)	Sets the value of the ConnectKind (see page 18110) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	EventDispatch (see page 18111)	This is EventDispatch, a member of class TOleServer.
	ServerData (see page 18111)	Encodes information about the COM server to which ServerData connects.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChannel (see page 19127)	This is AddChannel, a member of class TShellUIHelper.
	AddDesktopComponent (see page 19127)	This is the overview for the AddDesktopComponent method overload.
	AddFavorite (see page 19129)	This is the overview for the AddFavorite method overload.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
	AutoCompleteAttach (see page 19129)	This is the overview for the AutoCompleteAttach method overload.
	AutoCompleteSaveForm (see page 19130)	This is the overview for the AutoCompleteSaveForm method overload.
	AutoScan (see page 19131)	This is the overview for the AutoScan method overload.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Connect (see page 19131)	Establishes a connection to the COM server.
	ConnectTo (see page 19132)	This is ConnectTo, a member of class TShellUIHelper.
V	Create (see page 19125)	Allocates memory and constructs a safely initialized instance of a component.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 19125)	Disposes of the component and its owned components.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Disconnect (see page 19132)	Terminates the connection to the COM server.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	ImportExportFavorites (see page 19132)	This is ImportExportFavorites, a member of class TShellUIHelper.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsSubscribed (see page 19133)	This is IsSubscribed, a member of class TShellUIHelper.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NavigateAndFind (see page 19133)	This is NavigateAndFind, a member of class TShellUIHelper.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RefreshOfflineDesktop (see page 19133)	This is RefreshOfflineDesktop, a member of class TShellUIHelper.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	ResetFirstBootMode (see page 19134)	This is ResetFirstBootMode, a member of class TShellUIHelper.
	ResetSafeMode (see page 19134)	This is ResetSafeMode, a member of class TShellUIHelper.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	ShowBrowserUI (see page 19134)	This is ShowBrowserUI, a member of class TShellUIHelper.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	DefaultInterface (see page 19135)	This is DefaultInterface, a member of class TShellUIHelper.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	AutoConnect (see page 18110)	Specifies whether the component automatically connects to the server.
	ConnectKind (see page 18110)	Indicates the type of process to which the ConnectKind component establishes a connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.

	RemoteMachineName (see page 18111)	Specifies the machine running the COM server when ConnectKind (see page 18110) is ckRemote.
	Tag (see page 2808)	Stores an integer value as part of a component.

132.1.10.2 TShellUIHelper.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TShellUIHelper(TComponent * AOwner);
```

Description

All objects have a Create method that constructs the object. TComponent redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner, as indicated by the AOwner parameter

sets the ComponentStyle property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components (see page 2799)

ComponentStyle (see page 2805)

Create (see page 22287)

Destroy (see page 2766)

Free (see page 22300)

Owner (see page 2807)

132.1.10.3 TShellUIHelper.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TShellUIHelper();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

Destroy (see page 22287)

DestroyComponents (see page 2771)

Destroying (see page 2772)

Free (see page 22300)

Notification (see page 2783)

Owner (see page 2807)

Release (see page 13885)

Components (see page 2799)

132.1.10.4 TShellUIHelper Methods

The methods of the TShellUIHelper class are listed here.

Protected Methods

	Name	Description
	InitServerData (see page 19133)	Initializes the ServerData property to reflect the type library information of the COM server.

Public Methods

	Name	Description
	AddChannel (see page 19127)	This is AddChannel, a member of class TShellUIHelper.
	AddDesktopComponent (see page 19127)	This is the overview for the AddDesktopComponent method overload.
	AddFavorite (see page 19129)	This is the overview for the AddFavorite method overload.
	AutoCompleteAttach (see page 19129)	This is the overview for the AutoCompleteAttach method overload.
	AutoCompleteSaveForm (see page 19130)	This is the overview for the AutoCompleteSaveForm method overload.
	AutoScan (see page 19131)	This is the overview for the AutoScan method overload.
	Connect (see page 19131)	Establishes a connection to the COM server.
	ConnectTo (see page 19132)	This is ConnectTo, a member of class TShellUIHelper.
	Disconnect (see page 19132)	Terminates the connection to the COM server.
	ImportExportFavorites (see page 19132)	This is ImportExportFavorites, a member of class TShellUIHelper.
	IsSubscribed (see page 19133)	This is IsSubscribed, a member of class TShellUIHelper.

	NavigateAndFind (see page 19133)	This is NavigateAndFind, a member of class TShellUIHelper.
	RefreshOfflineDesktop (see page 19133)	This is RefreshOfflineDesktop, a member of class TShellUIHelper.
	ResetFirstBootMode (see page 19134)	This is ResetFirstBootMode, a member of class TShellUIHelper.
	ResetSafeMode (see page 19134)	This is ResetSafeMode, a member of class TShellUIHelper.
	ShowBrowserUI (see page 19134)	This is ShowBrowserUI, a member of class TShellUIHelper.

132.1.10.4.1 TShellUIHelper.AddChannel Method

Pascal

```
procedure AddChannel(const URL: WideString);
```

C++

```
__fastcall AddChannel(const BSTR URL);
```

Description

This is AddChannel, a member of class TShellUIHelper.

132.1.10.4.2 AddDesktopComponent Method

This is the overview for the AddDesktopComponent method overload.

Overload List

	Name	Description
	TShellUIHelper.AddDesktopComponent (WideString, WideString) (see page 19127)	This is AddDesktopComponent, a member of class TShellUIHelper.
	TShellUIHelper.AddDesktopComponent (WideString, WideString, OleVariant) (see page 19128)	This is AddDesktopComponent, a member of class TShellUIHelper.
	TShellUIHelper.AddDesktopComponent (WideString, WideString, OleVariant, OleVariant) (see page 19128)	This is AddDesktopComponent, a member of class TShellUIHelper.
	TShellUIHelper.AddDesktopComponent (WideString, WideString, OleVariant, OleVariant, OleVariant) (see page 19128)	This is AddDesktopComponent, a member of class TShellUIHelper.
	TShellUIHelper.AddDesktopComponent (WideString, WideString, OleVariant, OleVariant, OleVariant, OleVariant) (see page 19128)	This is AddDesktopComponent, a member of class TShellUIHelper.

132.1.10.4.2.1 TShellUIHelper.AddDesktopComponent Method (WideString, WideString)

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString); overload;
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_);
```

Description

This is AddDesktopComponent, a member of class TShellUIHelper.

132.1.10.4.2.2 TShellUIHelper.AddDesktopComponent Method (WideString, WideString, OleVariant)

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString; var Left:
OleVariant); overload;
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_, OleVariant Left);
```

Description

This is AddDesktopComponent, a member of class TShellUIHelper.

132.1.10.4.2.3 TShellUIHelper.AddDesktopComponent Method (WideString, WideString, OleVariant, OleVariant)

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString; var Left:
OleVariant; var Top: OleVariant); overload;
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_, OleVariant Left, OleVariant
Top);
```

Description

This is AddDesktopComponent, a member of class TShellUIHelper.

132.1.10.4.2.4 TShellUIHelper.AddDesktopComponent Method (WideString, WideString, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString; var Left:
OleVariant; var Top: OleVariant; var Width: OleVariant); overload;
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_, OleVariant Left, OleVariant
Top, OleVariant Width);
```

Description

This is AddDesktopComponent, a member of class TShellUIHelper.

132.1.10.4.2.5 TShellUIHelper.AddDesktopComponent Method (WideString, WideString, OleVariant, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString; var Left:
OleVariant; var Top: OleVariant; var Width: OleVariant; var Height: OleVariant); overload;
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_, OleVariant Left, OleVariant
Top, OleVariant Width, OleVariant Height);
```

Description

This is AddDesktopComponent, a member of class TShellUIHelper.

132.1.10.4.3 AddFavorite Method

This is the overview for the AddFavorite method overload.

Overload List

	Name	Description
	TShellUIHelper.AddFavorite (WideString) (see page 19129)	This is AddFavorite, a member of class TShellUIHelper.
	TShellUIHelper.AddFavorite (WideString, OleVariant) (see page 19129)	This is AddFavorite, a member of class TShellUIHelper.

132.1.10.4.3.1 TShellUIHelper.AddFavorite Method (WideString)

Pascal

```
procedure AddFavorite(const URL: WideString); overload;
```

C++

```
__fastcall AddFavorite(const BSTR URL);
```

Description

This is AddFavorite, a member of class TShellUIHelper.

132.1.10.4.3.2 TShellUIHelper.AddFavorite Method (WideString, OleVariant)

Pascal

```
procedure AddFavorite(const URL: WideString; var Title: OleVariant); overload;
```

C++

```
__fastcall AddFavorite(const BSTR URL, OleVariant Title);
```

Description

This is AddFavorite, a member of class TShellUIHelper.

132.1.10.4.4 AutoCompleteAttach Method

This is the overview for the AutoCompleteAttach method overload.

Overload List

	Name	Description
	TShellUIHelper.AutoCompleteAttach () (see page 19129)	This is AutoCompleteAttach, a member of class TShellUIHelper.
	TShellUIHelper.AutoCompleteAttach (OleVariant) (see page 19130)	This is AutoCompleteAttach, a member of class TShellUIHelper.

132.1.10.4.4.1 TShellUIHelper.AutoCompleteAttach Method ()

Pascal

```
procedure AutoCompleteAttach; overload;
```

C++

```
__fastcall AutoCompleteAttach();
```

Description

This is AutoCompleteAttach, a member of class TShellUIHelper.

132.1.10.4.4.2 TShellUIHelper.AutoCompleteAttach Method (OleVariant)

Pascal

```
procedure AutoCompleteAttach(var Reserved: OleVariant); overload;
```

C++

```
__fastcall AutoCompleteAttach(OleVariant Reserved);
```

Description

This is AutoCompleteAttach, a member of class TShellUIHelper.

132.1.10.4.5 AutoCompleteSaveForm Method

This is the overview for the AutoCompleteSaveForm method overload.

Overload List

	Name	Description
	TShellUIHelper.AutoCompleteSaveForm () (see page 19130)	This is AutoCompleteSaveForm, a member of class TShellUIHelper.
	TShellUIHelper.AutoCompleteSaveForm (OleVariant) (see page 19130)	This is AutoCompleteSaveForm, a member of class TShellUIHelper.

132.1.10.4.5.1 TShellUIHelper.AutoCompleteSaveForm Method ()

Pascal

```
procedure AutoCompleteSaveForm; overload;
```

C++

```
__fastcall AutoCompleteSaveForm();
```

Description

This is AutoCompleteSaveForm, a member of class TShellUIHelper.

132.1.10.4.5.2 TShellUIHelper.AutoCompleteSaveForm Method (OleVariant)

Pascal

```
procedure AutoCompleteSaveForm(var Form: OleVariant); overload;
```

C++

```
__fastcall AutoCompleteSaveForm(OleVariant Form);
```

Description

This is AutoCompleteSaveForm, a member of class TShellUIHelper.

132.1.10.4.6 AutoScan Method

This is the overview for the AutoScan method overload.

Overload List

	Name	Description
	TShellUIHelper.AutoScan (WideString, WideString) (see page 19131)	This is AutoScan, a member of class TShellUIHelper.
	TShellUIHelper.AutoScan (WideString, WideString, OleVariant) (see page 19131)	This is AutoScan, a member of class TShellUIHelper.

132.1.10.4.6.1 TShellUIHelper.AutoScan Method (WideString, WideString)

Pascal

```
procedure AutoScan(const strSearch: WideString; const strFailureUrl: WideString); overload;
```

C++

```
__fastcall AutoScan(const BSTR strSearch, const BSTR strFailureUrl);
```

Description

This is AutoScan, a member of class TShellUIHelper.

132.1.10.4.6.2 TShellUIHelper.AutoScan Method (WideString, WideString, OleVariant)

Pascal

```
procedure AutoScan(const strSearch: WideString; const strFailureUrl: WideString; var
pvarTargetFrame: OleVariant); overload;
```

C++

```
__fastcall AutoScan(const BSTR strSearch, const BSTR strFailureUrl, OleVariant
pvarTargetFrame);
```

Description

This is AutoScan, a member of class TShellUIHelper.

132.1.10.4.7 TShellUIHelper.Connect Method

Establishes a connection to the COM server.

Pascal

```
procedure Connect; override;
```

C++

```
virtual __fastcall Connect();
```

Description

Call Connect to connect the Connect component to the COM server it represents. If the AutoConnect property is True true, the Connect component automatically calls Connect when the application is first run.

Do not use Connect if the value of ConnectKind is ckAttachToInterface. When attaching to an interface, use the ConnectTo method, which is introduced by descendant classes, instead.

If the value of ConnectKind is ckRemote, the RemoteMachineName property must be set before calling Connect.

Connect introduces Connect as an abstract a pure virtual method. Descendant classes must override this method to let the Connect component connect to a specific type of COM server.

See Also

AutoConnect (see page 18110)

ConnectKind (see page 18110)

RemoteMachineName (see page 18111)

Disconnect (see page 18105)

Using Component Wrappers

132.1.10.4.8 TShellUIHelper.ConnectTo Method

Pascal

```
procedure ConnectTo(svrIntf: IShellUIHelper);
```

C++

```
__fastcall ConnectTo(IShellUIHelper svrIntf);
```

Description

This is ConnectTo, a member of class TShellUIHelper.

132.1.10.4.9 TShellUIHelper.Disconnect Method

Terminates the connection to the COM server.

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

Call Disconnect to terminate a connection formed using the Connect method.

Disconnect introduces Disconnect as an abstract a pure virtual method. Descendant classes must override this method to let the Disconnect component disconnect from a specific type of COM server.

See Also

Connect (see page 18104)

Using Component Wrappers

132.1.10.4.10 TShellUIHelper.ImportExportFavorites Method

Pascal

```
procedure ImportExportFavorites(fImport: WordBool; const strImpExpPath: WideString);
```

C++

```
__fastcall ImportExportFavorites(WordBool fImport, const BSTR strImpExpPath);
```

Description

This is ImportExportFavorites, a member of class TShellUIHelper.

132.1.10.4.11 TShellUIHelper.InitServerData Method

Initializes the ServerData property to reflect the type library information of the COM server.

Pascal

```
procedure InitServerData; override;
```

C++

```
virtual __fastcall InitServerData();
```

Description

The InitServerData constructor calls InitServerData to initialize ServerData with information from the COM server's type library.

InitServerData is an abstract a pure virtual method in InitServerData. Each InitServerData descendant implements this method so that the ServerData property reflects its particular COM server.

See Also

ServerData (see page 18111)

Create (see page 18101)

132.1.10.4.12 TShellUIHelper.IsSubscribed Method

Pascal

```
function IsSubscribed(const URL: WideString): WordBool;
```

C++

```
__fastcall WordBool IsSubscribed(const BSTR URL);
```

Description

This is IsSubscribed, a member of class TShellUIHelper.

132.1.10.4.13 TShellUIHelper.NavigateAndFind Method

Pascal

```
procedure NavigateAndFind(const URL: WideString; const strQuery: WideString; var
varTargetFrame: OleVariant);
```

C++

```
__fastcall NavigateAndFind(const BSTR URL, const BSTR strQuery, OleVariant varTargetFrame);
```

Description

This is NavigateAndFind, a member of class TShellUIHelper.

132.1.10.4.14 TShellUIHelper.RefreshOfflineDesktop Method

Pascal

```
procedure RefreshOfflineDesktop;
```

C++

```
__fastcall RefreshOfflineDesktop();
```

Description

This is RefreshOfflineDesktop, a member of class TShellUIHelper.

132.1.10.4.15 TShellUIHelper.ResetFirstBootMode Method

Pascal

```
procedure ResetFirstBootMode;
```

C++

```
__fastcall ResetFirstBootMode();
```

Description

This is ResetFirstBootMode, a member of class TShellUIHelper.

132.1.10.4.16 TShellUIHelper.ResetSafeMode Method

Pascal

```
procedure ResetSafeMode;
```

C++

```
__fastcall ResetSafeMode();
```

Description

This is ResetSafeMode, a member of class TShellUIHelper.

132.1.10.4.17 TShellUIHelper.ShowBrowserUI Method

Pascal

```
function ShowBrowserUI(const bstrName: WideString; var pvarIn: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant ShowBrowserUI(const BSTR bstrName, OleVariant pvarIn);
```

Description

This is ShowBrowserUI, a member of class TShellUIHelper.

132.1.10.5 TShellUIHelper Properties

The properties of the TShellUIHelper class are listed here.

Public Properties

	Name	Description
R	DefaultInterface (see page 19135)	This is DefaultInterface, a member of class TShellUIHelper.

132.1.10.5.1 TShellUIHelper.DefaultInterface Property

Pascal

```
property DefaultInterface: IShellUIHelper;
```

C++

```
__property IShellUIHelper DefaultInterface;
```

Description

This is DefaultInterface, a member of class TShellUIHelper.

132.1.11 TShellWindows Class

Class Hierarchy

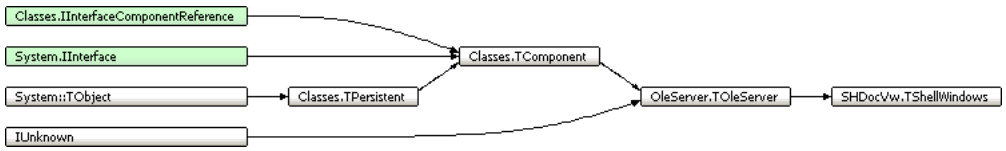

Pascal

```
TShellWindows = class(TOleServer);
```

C++

```
class TShellWindows : public TOleServer;
```

File

SHDocVw

Description

This is class SHDocVw.TShellWindows.

132.1.11.1 TShellWindows Members

The following tables list the members exposed by TShellWindows.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_AddRef (see page 18104)	Increments the reference count for the number of interface references dependent on the OLE server.
	_Release (see page 2769)	Called when an application releases a component interface.
	_Release (see page 18104)	Decrements the reference count for the calling interface on a object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.

	ConnectEvents (see page 18105)	Establishes a connection using the EventDispatch (see page 18111) property as an event sink.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DisconnectEvents (see page 18106)	Terminates a connection that was established using the ConnectEvents (see page 18105) method.
	Get_Count (see page 19142)	This is Get_Count, a member of class TShellWindows.
	GetAutoConnect (see page 18106)	Returns the value of the AutoConnect (see page 18110) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetConnectKind (see page 18106)	Returns the value of the ConnectKind (see page 18110) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetServer (see page 18107)	Returns an interface for the COM server.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InitServerData (see page 19142)	Initializes the ServerData property to reflect the type library information of the COM server.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	InvokeEvent (see page 19143)	Dispatches an event from the COM server to the appropriate event handler.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 18108)	Connects to the COM server when the Loaded object is first loaded into memory if AutoConnect (see page 18110) is True true.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	QueryInterface (see page 18108)	Returns a reference to a specified interface if the OLE server supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoConnect (see page 18109)	Sets the value of the AutoConnect (see page 18110) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnectKind (see page 18109)	Sets the value of the ConnectKind (see page 18110) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	EventDispatch (see page 18111)	This is EventDispatch, a member of class TOleServer.
	ServerData (see page 18111)	Encodes information about the COM server to which ServerData connects.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_NewEnum (see page 19141)	This is _NewEnum, a member of class TShellWindows.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Connect (see page 19141)	Establishes a connection to the COM server.
	ConnectTo (see page 19141)	This is ConnectTo, a member of class TShellWindows.
V	Create (see page 19139)	Allocates memory and constructs a safely initialized instance of a component.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 19139)	Disposes of the component and its owned components.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Disconnect (see page 19142)	Terminates the connection to the COM server.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindWindowSW (see page 19142)	This is FindWindowSW, a member of class TShellWindows.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	Item (see page 19143)	This is the overview for the Item method overload.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	OnActivated (see page 19144)	This is OnActivated, a member of class TShellWindows.
	OnCreated (see page 19144)	This is OnCreated, a member of class TShellWindows.
	OnNavigate (see page 19144)	This is OnNavigate, a member of class TShellWindows.
	ProcessAttachDetach (see page 19145)	This is ProcessAttachDetach, a member of class TShellWindows.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Register (see page 19145)	This is Register, a member of class TShellWindows.
	RegisterPending (see page 19145)	This is RegisterPending, a member of class TShellWindows.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Revoke (see page 19145)	This is Revoke, a member of class TShellWindows.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Count (see page 19146)	This is Count, a member of class TShellWindows.
R	DefaultInterface (see page 19146)	This is DefaultInterface, a member of class TShellWindows.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnWindowRegistered (see page 19146)	This is OnWindowRegistered, a member of class TShellWindows.
	OnWindowRevoked (see page 19147)	This is OnWindowRevoked, a member of class TShellWindows.

Published Properties

	Name	Description
	AutoConnect (see page 18110)	Specifies whether the component automatically connects to the server.
	ConnectKind (see page 18110)	Indicates the type of process to which the ConnectKind component establishes a connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RemoteMachineName (see page 18111)	Specifies the machine running the COM server when ConnectKind (see page 18110) is ckRemote.

	Tag (see page 2808)	Stores an integer value as part of a component.

132.1.11.2 TShellWindows.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TShellWindows(TComponent * AOwner);
```

Description

All objects have a Create method that constructs the object. TComponent redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner, as indicated by the AOwner parameter

sets the ComponentStyle property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components (see page 2799)

ComponentStyle (see page 2805)

Create (see page 22287)

Destroy (see page 2766)

Free (see page 22300)

Owner (see page 2807)

132.1.11.3 TShellWindows.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TShellWindows();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

Destroy (see page 22287)

DestroyComponents (see page 2771)

Destroying (see page 2772)

Free (see page 22300)

Notification (see page 2783)

Owner (see page 2807)

Release (see page 13885)

Components (see page 2799)

132.1.11.4 TShellWindows Methods

The methods of the TShellWindows class are listed here.

Protected Methods

	Name	Description
	Get_Count (see page 19142)	This is Get_Count, a member of class TShellWindows.
V	InitServerData (see page 19142)	Initializes the ServerData property to reflect the type library information of the COM server.
V	InvokeEvent (see page 19143)	Dispatches an event from the COM server to the appropriate event handler.

Public Methods

	Name	Description
	_NewEnum (see page 19141)	This is _NewEnum, a member of class TShellWindows.
V	Connect (see page 19141)	Establishes a connection to the COM server.
	ConnectTo (see page 19141)	This is ConnectTo, a member of class TShellWindows.
V	Disconnect (see page 19142)	Terminates the connection to the COM server.
	FindWindowSW (see page 19142)	This is FindWindowSW, a member of class TShellWindows.
	Item (see page 19143)	This is the overview for the Item method overload.
	OnActivated (see page 19144)	This is OnActivated, a member of class TShellWindows.
	OnCreated (see page 19144)	This is OnCreated, a member of class TShellWindows.
	OnNavigate (see page 19144)	This is OnNavigate, a member of class TShellWindows.

	ProcessAttachDetach (see page 19145)	This is ProcessAttachDetach, a member of class TShellWindows.
	Register (see page 19145)	This is Register, a member of class TShellWindows.
	RegisterPending (see page 19145)	This is RegisterPending, a member of class TShellWindows.
	Revoke (see page 19145)	This is Revoke, a member of class TShellWindows.

132.1.11.4.1 TShellWindows._NewEnum Method

Pascal

```
function _NewEnum: IUnknown;
```

C++

```
__fastcall IUnknown _NewEnum();
```

Description

This is _NewEnum, a member of class TShellWindows.

132.1.11.4.2 TShellWindows.Connect Method

Establishes a connection to the COM server.

Pascal

```
procedure Connect; override;
```

C++

```
virtual __fastcall Connect();
```

Description

Call Connect to connect the Connect component to the COM server it represents. If the AutoConnect property is True true, the Connect component automatically calls Connect when the application is first run.

Do not use Connect if the value of ConnectKind is ckAttachToInterface. When attaching to an interface, use the ConnectTo method, which is introduced by descendant classes, instead.

If the value of ConnectKind is ckRemote, the RemoteMachineName property must be set before calling Connect.

Connect introduces Connect as an abstract a pure virtual method. Descendant classes must override this method to let the Connect component connect to a specific type of COM server.

See Also

AutoConnect (see page 18110)

ConnectKind (see page 18110)

RemoteMachineName (see page 18111)

Disconnect (see page 18105)

Using Component Wrappers

132.1.11.4.3 TShellWindows.ConnectTo Method

Pascal

```
procedure ConnectTo(svrIntf: IShellWindows);
```

C++

```
__fastcall ConnectTo(IShellWindows svrIntf);
```

Description

This is ConnectTo, a member of class TShellWindows.

132.1.11.4.4 TShellWindows.Disconnect Method

Terminates the connection to the COM server.

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

Call Disconnect to terminate a connection formed using the Connect method.

Disconnect introduces Disconnect as an abstract a pure virtual method. Descendant classes must override this method to let the Disconnect component disconnect from a specific type of COM server.

See Also

Connect (see page 18104)

Using Component Wrappers

132.1.11.4.5 TShellWindows.FindWindowSW Method

Pascal

```
function FindWindowSW(var pvarloc: OleVariant; var pvarlocRoot: OleVariant; swClass: SYSINT;
out pHWND: Integer; swfwOptions: SYSINT): IDispatch;
```

C++

```
__fastcall IDispatch FindWindowSW(OleVariant pvarloc, OleVariant pvarlocRoot, SYSINT swClass,
int pHWND, SYSINT swfwOptions);
```

Description

This is FindWindowSW, a member of class TShellWindows.

132.1.11.4.6 TShellWindows.Get_Count Method

Pascal

```
function Get_Count: Integer;
```

C++

```
__fastcall int Get_Count();
```

Description

This is Get_Count, a member of class TShellWindows.

132.1.11.4.7 TShellWindows.InitServerData Method

Initializes the ServerData property to reflect the type library information of the COM server.

Pascal

```
procedure InitServerData; override;
```

C++

```
virtual __fastcall InitServerData();
```

Description

The InitServerData constructor calls InitServerData to initialize ServerData with information from the COM server's type library.

InitServerData is an abstract a pure virtual method in InitServerData. Each InitServerData descendant implements this method so that the ServerData property reflects its particular COM server.

See Also

ServerData (see page 18111)

Create (see page 18101)

132.1.11.4.8 TShellWindows.InvokeEvent Method

Dispatches an event from the COM server to the appropriate event handler.

Pascal

```
procedure InvokeEvent(DispID: TDispID; var Params: TVariantArray); override;
```

C++

```
virtual __fastcall InvokeEvent(TDispID DispID, TVariantArray Params);
```

Description

InvokeEvent descendants use InvokeEvent internally to link their event handlers to events on the COM server.

DispID is the dispatch ID for the appropriate event handler.

Params passes the parameters of the event handler as an array of OleVariants.

As implemented in InvokeEvent, InvokeEvent does nothing, because InvokeEvent has no events. Descendants that surface events from a COM server override this method so that the event sink specified by EventDispatch can pass events on to the appropriate event handlers.

See Also

EventDispatch (see page 18111)

132.1.11.4.9 Item Method

This is the overview for the Item method overload.

Overload List

	Name	Description
	TShellWindows.Item () (see page 19144)	This is Item, a member of class TShellWindows.
	TShellWindows.Item (OleVariant) (see page 19144)	This is Item, a member of class TShellWindows.

132.1.11.4.9.1 TShellWindows.Item Method ()

Pascal

```
function Item: IDispatch; overload;
```

C++

```
__fastcall IDispatch Item();
```

Description

This is Item, a member of class TShellWindows.

132.1.11.4.9.2 TShellWindows.Item Method (OleVariant)

Pascal

```
function Item(index: OleVariant): IDispatch; overload;
```

C++

```
__fastcall IDispatch Item(OleVariant index);
```

Description

This is Item, a member of class TShellWindows.

132.1.11.4.10 TShellWindows.OnActivated Method

Pascal

```
procedure OnActivated(lCookie: Integer; fActive: WordBool);
```

C++

```
__fastcall OnActivated(int lCookie, WordBool fActive);
```

Description

This is OnActivated, a member of class TShellWindows.

132.1.11.4.11 TShellWindows.OnCreated Method

Pascal

```
procedure OnCreated(lCookie: Integer; const punk: IUnknown);
```

C++

```
__fastcall OnCreated(int lCookie, const IUnknown punk);
```

Description

This is OnCreated, a member of class TShellWindows.

132.1.11.4.12 TShellWindows.OnNavigate Method

Pascal

```
procedure OnNavigate(lCookie: Integer; var pvarloc: OleVariant);
```

C++

```
__fastcall OnNavigate(int lCookie, OleVariant pvarloc);
```

Description

This is OnNavigate, a member of class TShellWindows.

132.1.11.4.13 TShellWindows.ProcessAttachDetach Method

Pascal

```
procedure ProcessAttachDetach(fAttach: WordBool);
```

C++

```
__fastcall ProcessAttachDetach(WordBool fAttach);
```

Description

This is ProcessAttachDetach, a member of class TShellWindows.

132.1.11.4.14 TShellWindows.Register Method

Pascal

```
procedure Register(const pid: IDispatch; HWND: Integer; swClass: SYSINT; out plCookie:
Integer);
```

C++

```
__fastcall Register(const IDispatch pid, int HWND, SYSINT swClass, int plCookie);
```

Description

This is Register, a member of class TShellWindows.

132.1.11.4.15 TShellWindows.RegisterPending Method

Pascal

```
procedure RegisterPending(lThreadId: Integer; var pvarloc: OleVariant; var pvarlocRoot:
OleVariant; swClass: SYSINT; out plCookie: Integer);
```

C++

```
__fastcall RegisterPending(int lThreadId, OleVariant pvarloc, OleVariant pvarlocRoot, SYSINT
swClass, int plCookie);
```

Description

This is RegisterPending, a member of class TShellWindows.

132.1.11.4.16 TShellWindows.Revoke Method

Pascal

```
procedure Revoke(lCookie: Integer);
```

C++

```
__fastcall Revoke(int lCookie);
```

Description

This is Revoke, a member of class TShellWindows.

132.1.11.5 TShellWindows Properties

The properties of the TShellWindows class are listed here.

Public Properties

	Name	Description
R	Count (see page 19146)	This is Count, a member of class TShellWindows.
R	DefaultInterface (see page 19146)	This is DefaultInterface, a member of class TShellWindows.

132.1.11.5.1 TShellWindows.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TShellWindows.

132.1.11.5.2 TShellWindows.DefaultInterface Property

Pascal

```
property DefaultInterface: IShellWindows;
```

C++

```
__property IShellWindows DefaultInterface;
```

Description

This is DefaultInterface, a member of class TShellWindows.

132.1.11.6 TShellWindows Events

The events of the TShellWindows class are listed here.

Published Events

	Name	Description
	OnWindowRegistered (see page 19146)	This is OnWindowRegistered, a member of class TShellWindows.
	OnWindowRevoked (see page 19147)	This is OnWindowRevoked, a member of class TShellWindows.

132.1.11.6.1 TShellWindows.OnWindowRegistered Event

Pascal

```
property OnWindowRegistered: TShellWindowsWindowRegistered;
```

C++

```
__property TShellWindowsWindowRegistered OnWindowRegistered;
```

Description

This is OnWindowRegistered, a member of class TShellWindows.

132.1.11.6.2 TShellWindows.OnWindowRevoked Event

Pascal

```
property OnWindowRevoked: TShellWindowsWindowRevoked;
```

C++

```
__property TShellWindowsWindowRevoked OnWindowRevoked;
```

Description

This is OnWindowRevoked, a member of class TShellWindows.

132.1.12 TWebBrowser Class

TWebBrowser provides access to the Web browser functionality of Microsoft's Shell Doc Object and Control Library (SHDOCVW.DLL).

Class Hierarchy

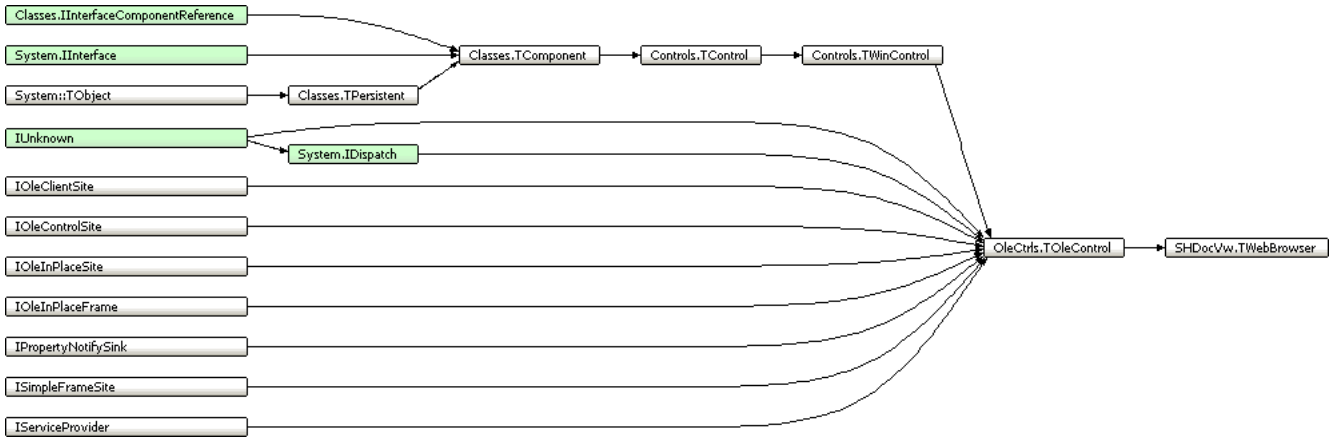

Pascal

```
TWebBrowser = class(TOleControl);
```

C++

```
class TWebBrowser : public TOleControl;
```

File

SHDocVw

Description

TWebBrowser wraps the IWebBrowser2 (see page 19403) interface from Microsoft's Shell Doc Object and Control Library (SHDOCVW.DLL) to allow you to create a customized Web browsing application or to add Internet, file and network browsing, document viewing, and data downloading capabilities to your applications.

Note: Because TWebBrowser wraps the SHDOCVW.DLL, you must have SHDOCVW.DLL installed to use this component. This DLL ships with Internet Explorer 4 and later.

132.1.12.1 TWebBrowser Members

The following tables list the members exposed by TWebBrowser.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.

	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_AddRef (see page 18046)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	_Release (see page 18047)	Called when an application releases a component interface.
	_SetColorProp (see page 18047)	This is _SetColorProp, a member of class TOleControl.
	_SetTColorProp (see page 18048)	This is _SetTColorProp, a member of class TOleControl.
	_SetTFontProp (see page 18048)	This is _SetTFontProp, a member of class TOleControl.
	_SetTOleEnumProp (see page 18048)	This is _SetTOleEnumProp, a member of class TOleControl.
	_SetTPictureProp (see page 18048)	This is _SetTPictureProp, a member of class TOleControl.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanInPlaceActivate (see page 18049)	This is CanInPlaceActivate, a member of class TOleControl.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ContextSensitiveHelp (see page 18049)	This is ContextSensitiveHelp, a member of class TOleControl.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateControl (see page 19163)	This is CreateControl, a member of class TWebBrowser.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CreateWnd (see page 18049)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.

	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DeactivateAndUndo (see page 18050)	This is DeactivateAndUndo, a member of class TOleControl.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
V	DefineProperties (see page 18050)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWindowHandle (see page 18051)	Destroys the window created in the CreateWindowHandle method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DiscardUndoState (see page 18051)	This is DiscardUndoState, a member of class TOleControl.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EnableModeless (see page 18052)	This is EnableModeless, a member of class TOleControl.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	Get_Application (see page 19164)	This is Get_Application, a member of class TWebBrowser.
	Get_Container (see page 19164)	This is Get_Container, a member of class TWebBrowser.
	Get_Document (see page 19164)	This is Get_Document, a member of class TWebBrowser.
	Get_Parent (see page 19165)	This is Get_Parent, a member of class TWebBrowser.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.

	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorder (see page 18052)	This is GetBorder, a member of class TOleControl.
	GetByteProp (see page 18052)	This is GetByteProp, a member of class TOleControl.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetColorProp (see page 18053)	This is GetColorProp, a member of class TOleControl.
	GetCompProp (see page 18053)	This is GetCompProp, a member of class TOleControl.
	GetContainer (see page 18053)	This is GetContainer, a member of class TOleControl.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetCurrencyProp (see page 18053)	This is GetCurrencyProp, a member of class TOleControl.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDoubleProp (see page 18053)	This is GetDoubleProp, a member of class TOleControl.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetExtendedControl (see page 18054)	This is GetExtendedControl, a member of class TOleControl.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDispatchProp (see page 18055)	This is GetIDispatchProp, a member of class TOleControl.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIDsOfNames (see page 18055)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIntegerProp (see page 18055)	This is GetIntegerProp, a member of class TOleControl.
	GetIUnknownProp (see page 18055)	This is GetIUnknownProp, a member of class TOleControl.
	GetMoniker (see page 18056)	This is GetMoniker, a member of class TOleControl.
	GetOleBoolProp (see page 18056)	This is GetOleBoolProp, a member of class TOleControl.
	GetOleDateProp (see page 18056)	This is GetOleDateProp, a member of class TOleControl.
	GetOleEnumProp (see page 18057)	This is GetOleEnumProp, a member of class TOleControl.
	GetOleVariantProp (see page 18057)	This is GetOleVariantProp, a member of class TOleControl.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetShortIntProp (see page 18058)	This is GetShortIntProp, a member of class TOleControl.
	GetSingleProp (see page 18058)	This is GetSingleProp, a member of class TOleControl.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetSmallintProp (see page 18058)	This is GetSmallintProp, a member of class TOleControl.
	GetStringProp (see page 18059)	This is GetStringProp, a member of class TOleControl.
	GetTColorProp (see page 18059)	This is GetTColorProp, a member of class TOleControl.
	GetTDateTimeProp (see page 18059)	This is GetTDateTimeProp, a member of class TOleControl.
	GetTFontProp (see page 18059)	This is GetTFontProp, a member of class TOleControl.
	GetTOleEnumProp (see page 18059)	This is GetTOleEnumProp, a member of class TOleControl.

	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTPictureProp (see page 18060)	This is GetTPictureProp, a member of class TOleControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfo (see page 18060)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetTypeInfoCount (see page 18060)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariantProp (see page 18061)	This is GetVariantProp, a member of class TOleControl.
	GetWideStringProp (see page 18061)	This is GetWideStringProp, a member of class TOleControl.
	GetWindowContext (see page 18061)	This is GetWindowContext, a member of class TOleControl.
	GetWordBoolProp (see page 18061)	This is GetWordBoolProp, a member of class TOleControl.
	GetWordProp (see page 18062)	This is GetWordProp, a member of class TOleControl.
V	InitControlData (see page 19167)	This is InitControlData, a member of class TWebBrowser.
V	InitControlInterface (see page 18062)	This is InitControlInterface, a member of class TOleControl.
	InsertMenus (see page 18062)	This is InsertMenus, a member of class TOleControl.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Invoke (see page 18062)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	InvokeEvent (see page 18063)	This is InvokeEvent, a member of class TOleControl.
	InvokeMethod (see page 18063)	This is InvokeMethod, a member of class TOleControl.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	LockInPlaceActive (see page 18064)	This is LockInPlaceActive, a member of class TOleControl.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	OleControlSite_TranslateAccelerator (see page 18064)	This is OleControlSite_TranslateAccelerator, a member of class TOleControl.
	OleInPlaceFrame_GetWindow (see page 18064)	This is OleInPlaceFrame_GetWindow, a member of class TOleControl.
	OleInPlaceFrame_TranslateAccelerator (see page 18065)	This is OleInPlaceFrame_TranslateAccelerator, a member of class TOleControl.
	OleInPlaceSite_GetWindow (see page 18065)	This is OleInPlaceSite_GetWindow, a member of class TOleControl.
V	OnChanged (see page 18065)	This is OnChanged, a member of class TOleControl.
	OnControlInfoChanged (see page 18065)	This is OnControlInfoChanged, a member of class TOleControl.
	OnFocus (see page 18066)	This is OnFocus, a member of class TOleControl.
	OnInPlaceActivate (see page 18066)	This is OnInPlaceActivate, a member of class TOleControl.
	OnInPlaceDeactivate (see page 18066)	This is OnInPlaceDeactivate, a member of class TOleControl.
	OnPosRectChange (see page 18066)	This is OnPosRectChange, a member of class TOleControl.
V	OnRequestEdit (see page 18066)	This is OnRequestEdit, a member of class TOleControl.
	OnShowWindow (see page 18067)	This is OnShowWindow, a member of class TOleControl.
	OnUIActivate (see page 18067)	This is OnUIActivate, a member of class TOleControl.

	OnUIDeactivate (see page 18067)	This is OnUIDeactivate, a member of class TOleControl.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
V	PaletteChanged (see page 18067)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PictureChanged (see page 18068)	This is PictureChanged, a member of class TOleControl.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	PostMessageFilter (see page 18068)	This is PostMessageFilter, a member of class TOleControl.
	PreMessageFilter (see page 18069)	This is PreMessageFilter, a member of class TOleControl.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	QueryInterface (see page 18069)	Returns a reference to a specified interface if the object supports that interface.
	QueryService (see page 18069)	This is QueryService, a member of class TOleControl.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveMenus (see page 18070)	This is RemoveMenus, a member of class TOleControl.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	RequestBorderSpace (see page 18070)	This is RequestBorderSpace, a member of class TOleControl.
	RequestNewObjectLayout (see page 18070)	This is RequestNewObjectLayout, a member of class TOleControl.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	SaveObject (see page 18070)	This is SaveObject, a member of class TOleControl.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Scroll (see page 18070)	This is Scroll, a member of class TOleControl.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActiveObject (see page 18071)	This is SetActiveObject, a member of class TOleControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetBorderSpace (see page 18071)	This is SetBorderSpace, a member of class TOleControl.
	SetByteProp (see page 18071)	This is SetByteProp, a member of class TOleControl.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetColorProp (see page 18072)	This is SetColorProp, a member of class TOleControl.
	SetCompProp (see page 18072)	This is SetCompProp, a member of class TOleControl.
	SetCurrencyProp (see page 18072)	This is SetCurrencyProp, a member of class TOleControl.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDoubleProp (see page 18072)	This is SetDoubleProp, a member of class TOleControl.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIDispatchProp (see page 18073)	This is SetIDispatchProp, a member of class TOleControl.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetIntegerProp (see page 18073)	This is SetIntegerProp, a member of class TOleControl.
	SetIUnknownProp (see page 18073)	This is SetIUnknownProp, a member of class TOleControl.
	SetMenu (see page 18073)	This is SetMenu, a member of class TOleControl.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetName (see page 18073)	Sets the name of the control.
	SetOleBoolProp (see page 18074)	This is SetOleBoolProp, a member of class TOleControl.
	SetOleDateProp (see page 18074)	This is SetOleDateProp, a member of class TOleControl.
	SetOleEnumProp (see page 18074)	This is SetOleEnumProp, a member of class TOleControl.
	SetOleVariantProp (see page 18075)	This is SetOleVariantProp, a member of class TOleControl.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParent (see page 18075)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetProperty (see page 18076)	This is SetProperty, a member of class TOleControl.
	SetShortIntProp (see page 18076)	This is SetShortIntProp, a member of class TOleControl.
	SetSingleProp (see page 18076)	This is SetSingleProp, a member of class TOleControl.
	SetSmallintProp (see page 18076)	This is SetSmallintProp, a member of class TOleControl.
	SetStatusText (see page 18077)	This is SetStatusText, a member of class TOleControl.
	SetStringProp (see page 18077)	This is SetStringProp, a member of class TOleControl.
	SetTColorProp (see page 18077)	This is SetTColorProp, a member of class TOleControl.
	SetTDateTimeProp (see page 18077)	This is SetTDateTimeProp, a member of class TOleControl.
	SetTFontProp (see page 18077)	This is SetTFontProp, a member of class TOleControl.
	SetTOleEnumProp (see page 18078)	This is SetTOleEnumProp, a member of class TOleControl.
	SetTPictureProp (see page 18078)	This is SetTPictureProp, a member of class TOleControl.
	SetVariantProp (see page 18078)	This is SetVariantProp, a member of class TOleControl.
	SetWideStringProp (see page 18078)	This is SetWideStringProp, a member of class TOleControl.
	SetWordBoolProp (see page 18079)	This is SetWordBoolProp, a member of class TOleControl.
	SetWordProp (see page 18079)	This is SetWordProp, a member of class TOleControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	ShowObject (see page 18079)	This is ShowObject, a member of class TOleControl.
	ShowPropertyFrame (see page 18079)	This is ShowPropertyFrame, a member of class TOleControl.
V	StandardEvent (see page 18080)	This is StandardEvent, a member of class TOleControl.
V	SuppressException (see page 18080)	This is SuppressException, a member of class TOleControl.
	TransformCoords (see page 18080)	This is TransformCoords, a member of class TOleControl.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WndProc (see page 18080)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	ControlData (see page 18086)	This is ControlData, a member of class TOleControl.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	ServiceQuery (see page 18087)	This is ServiceQuery, a member of class TOleControl.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	BrowseProperties (see page 18048)	Notifies the ActiveX control to display its Properties dialog box.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ClientToWindow (see page 19163)	Not used in TWebBrowser (see page 19147).
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 18043)	Creates a new instance of TOleControl (see page 18030).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 18050)	Handles all otherwise-unhandled messages.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 18043)	Frees the memory associated with this instance of TOleControl (see page 18030).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.

	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DoObjectVerb (see page 18051)	Notifies the ActiveX control to perform one of its actions.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecWB (see page 19163)	This is the overview for the ExecWB method overload.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetEnumPropDesc (see page 18054)	Returns descriptions of the ActiveX control's properties.
	GetHelpContext (see page 18054)	Retrieves the Help context of a member of the ActiveX control.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetObjectVerbs (see page 18056)	Retrieves the names of the verbs implemented by the ActiveX control.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetPropDisplayString (see page 18057)	Returns the display string for an interface for the ActiveX control.
	GetPropDisplayStrings (see page 18057)	Returns the display strings for an interface for the ActiveX control.
	GetProperty (see page 19165)	Not used in TWebBrowser (see page 19147).
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	GoBack (see page 19165)	Navigates to the previous item in the history list.
	GoForward (see page 19166)	Navigates to the next item in the history list.
	GoHome (see page 19166)	Navigates to the current start page.
	GoSearch (see page 19167)	Navigates to the current search page.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.

	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	IsCustomProperty (see page 18063)	Indicates whether an interface for the ActiveX control is a custom property.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsPropPageProperty (see page 18064)	Indicates whether an interface for the OCX control is a property page property.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	Navigate (see page 19167)	This is the overview for the Navigate method overload.
	Navigate2 (see page 19169)	This is the overview for the Navigate2 method overload.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	PutProperty (see page 19171)	Not used in TWebBrowser (see page 19147).
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	QueryStatusWB (see page 19171)	Returns information about what commands the Web browser supports.
	Quit (see page 19171)	Not used in TWebBrowser (see page 19147).
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 19172)	Reloads the current document.
	Refresh2 (see page 19172)	This is the overview for the Refresh2 method overload.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.

V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
V	SetBounds (see page 18071)	Sets the boundaries of the ActiveX control.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetPropDisplayString (see page 18075)	Sets the display string for an interface for the ActiveX control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	ShowAboutBox (see page 18079)	Notifies the ActiveX control to display its About box.
	ShowBrowserBar (see page 19173)	This is the overview for the ShowBrowserBar method overload.
	Stop (see page 19174)	Stops the current operation.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
R	Application (see page 19179)	Provides access to the Automation interface for the application that contains the WebBrowser (see page 19484) control, or, if that is unavailable, the interface for the WebBrowser (see page 19484) control.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Busy (see page 19180)	Indicates whether the WebBrowser (see page 19484) control is in the middle of a navigation or downloading operation.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	Container (see page 19180)	Provides access to the automation interface of the Web browser's container.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	ControlInterface (see page 19181)	Provides access to the interface for this Web browser component.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
R	DefaultDispatch (see page 18086)	The default dispatch interface.
R	DefaultInterface (see page 19181)	Provides access to the default interface for the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.

R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
R	Document (see page 19182)	Provides access to the Automation interface of the active document.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	FullName (see page 19184)	Indicates the fully qualified path name for the executable that hosts the Web Browser component.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
R	HWnd (see page 19185)	This is HWnd, a member of class TWebBrowser.
R	LocationName (see page 19185)	Identifies the short (UI-friendly) name of the URL or file currently displayed.
R	LocationURL (see page 19185)	Identifies the URL or path of the document or folder currently displayed.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Name (see page 19186)	Indicates the name for the executable that hosts the Web Browser component.
R	OleObject (see page 18086)	The ActiveX control associated with the TOleControl (see page 18030) object.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
R	Parent (see page 19187)	Provides access to the Automation interface of the Web browser's container.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Path (see page 19188)	Indicates the name for the executable that hosts the Web Browser component.
R	PerPropBrowsing (see page 18086)	Provides access to information about the property page properties of the OleControl.
R	ReadyState (see page 19193)	Indicates the state of the Web browser component.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
R	TopLevelContainer (see page 19198)	Indicates whether the Web browser is a top-level Automation object.
R	type_ (see page 19199)	This is type_, a member of class TWebBrowser.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnBeforeNavigate2 (see page 19200)	Occurs immediately before the Web browser navigates to a new resource.
	OnClientToHostWindow (see page 19200)	This is OnClientToHostWindow, a member of class TWebBrowser.
	OnCommandStateChange (see page 19201)	Occurs when the ability to execute certain TWebBrowser (see page 19147) methods changes.
	OnDocumentComplete (see page 19201)	Occurs when the document being navigated to reaches ReadyState_Complete.
	OnDownloadBegin (see page 19202)	Occurs when the Web browser starts downloading a document.

	OnDownloadComplete (see page 19202)	Occurs when a navigation operation finishes, is halted, or fails.
	OnDragDrop (see page 19203)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 19205)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 19207)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 19208)	Occurs when a control receives the input focus.
	OnExit (see page 19211)	Occurs when the input focus shifts away from one control to another.
	OnFileDownload (see page 19212)	This is OnFileDownload, a member of class TWebBrowser.
	OnFullScreen (see page 19213)	Not used in TWebBrowser (see page 19147).
	OnMenuBar (see page 19213)	Not used in TWebBrowser (see page 19147).
	OnNavigateComplete2 (see page 19213)	Occurs immediately after the Web browser successfully navigates to a new location.
	OnNavigateError (see page 19214)	This is OnNavigateError, a member of class TWebBrowser.
	OnNewWindow2 (see page 19214)	Occurs when a new, hidden, non-navigated Web browser window is needed.
	OnPrintTemplateInstantiation (see page 19214)	This is OnPrintTemplateInstantiation, a member of class TWebBrowser.
	OnPrintTemplateTeardown (see page 19215)	This is OnPrintTemplateTeardown, a member of class TWebBrowser.
	OnPrivacyImpactedStateChange (see page 19215)	This is OnPrivacyImpactedStateChange, a member of class TWebBrowser.
	OnProgressChange (see page 19215)	Occurs when the progress of a download operation is updated.
	OnPropertyChange (see page 19215)	Not used in TWebBrowser (see page 19147).
	OnQuit (see page 19216)	Not used in TWebBrowser (see page 19147).
	OnSetSecureLockIcon (see page 19216)	This is OnSetSecureLockIcon, a member of class TWebBrowser.
	OnStartDrag (see page 19216)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnStatusBar (see page 19217)	Not used in TWebBrowser (see page 19147).
	OnStatusTextChange (see page 19217)	Not used in TWebBrowser (see page 19147).
	OnTheaterMode (see page 19217)	Not used in TWebBrowser (see page 19147).
	OnTitleChange (see page 19218)	Occurs when the title of a document in the WebBrowser (see page 19484) control becomes available or changes.
	OnToolBar (see page 19218)	Not used in TWebBrowser (see page 19147).
	OnUpdatePageStatus (see page 19218)	This is OnUpdatePageStatus, a member of class TWebBrowser.
	OnVisible (see page 19219)	Occurs when the Web browser window is about to be shown or hidden.
	OnWindowClosing (see page 19219)	This is OnWindowClosing, a member of class TWebBrowser.
	OnWindowSetHeight (see page 19219)	This is OnWindowSetHeight, a member of class TWebBrowser.
	OnWindowSetLeft (see page 19219)	This is OnWindowSetLeft, a member of class TWebBrowser.
	OnWindowSetResizable (see page 19220)	This is OnWindowSetResizable, a member of class TWebBrowser.
	OnWindowSetTop (see page 19220)	This is OnWindowSetTop, a member of class TWebBrowser.
	OnWindowSetWidth (see page 19220)	This is OnWindowSetWidth, a member of class TWebBrowser.

Published Properties

	Name	Description
	AddressBar (see page 19175)	Not used in TWebBrowser (see page 19147).
	Align (see page 19175)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 19179)	Specifies how the control is anchored to its parent.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 19182)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 19182)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	FullScreen (see page 19184)	Not used in TWebBrowser (see page 19147).
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.

	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	MenuBar (see page 19186)	Not used in TWebBrowser (see page 19147).
	Offline (see page 19186)	Indicates whether the Web browser gets its document from the local cache.
	ParentShowHint (see page 19187)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 19188)	Identifies the pop-up menu associated with the control.
	RegisterAsBrowser (see page 19193)	Registers the control as a top-level browser for target name resolution.
	RegisterAsDropTarget (see page 19194)	Specifies whether users can load a document using drag-and-drop.
	Resizable (see page 19194)	Not used in TWebBrowser (see page 19147).
	ShowHint (see page 19194)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Silent (see page 19196)	Specifies whether the Web browser can display dialog boxes.
	StatusBar (see page 19196)	Not used in TWebBrowser (see page 19147).
	StatusText (see page 19196)	Not used in TWebBrowser (see page 19147).
	TabOrder (see page 19197)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 19197)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	TheaterMode (see page 19198)	Not used in TWebBrowser (see page 19147).
	ToolBar (see page 19198)	Not used in TWebBrowser (see page 19147).
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 19199)	Specifies whether the Web browser component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

132.1.12.2 TWebBrowser Methods

The methods of the TWebBrowser class are listed here.

Protected Methods

	Name	Description
	CreateControl (see page 19163)	This is CreateControl, a member of class TWebBrowser.
	Get_Application (see page 19164)	This is Get_Application, a member of class TWebBrowser.
	Get_Container (see page 19164)	This is Get_Container, a member of class TWebBrowser.
	Get_Document (see page 19164)	This is Get_Document, a member of class TWebBrowser.
	Get_Parent (see page 19165)	This is Get_Parent, a member of class TWebBrowser.
V	InitControlData (see page 19167)	This is InitControlData, a member of class TWebBrowser.

Public Methods

	Name	Description
	ClientToWindow (see page 19163)	Not used in TWebBrowser (see page 19147).
	ExecWB (see page 19163)	This is the overview for the ExecWB method overload.
	GetProperty (see page 19165)	Not used in TWebBrowser (see page 19147).
	GoBack (see page 19165)	Navigates to the previous item in the history list.
	GoForward (see page 19166)	Navigates to the next item in the history list.
	GoHome (see page 19166)	Navigates to the current start page.
	GoSearch (see page 19167)	Navigates to the current search page.
	Navigate (see page 19167)	This is the overview for the Navigate method overload.
	Navigate2 (see page 19169)	This is the overview for the Navigate2 method overload.
	PutProperty (see page 19171)	Not used in TWebBrowser (see page 19147).
	QueryStatusWB (see page 19171)	Returns information about what commands the Web browser supports.
	Quit (see page 19171)	Not used in TWebBrowser (see page 19147).
	Refresh (see page 19172)	Reloads the current document.
	Refresh2 (see page 19172)	This is the overview for the Refresh2 method overload.

	ShowBrowserBar (see page 19173)	This is the overview for the ShowBrowserBar method overload.
	Stop (see page 19174)	Stops the current operation.

132.1.12.2.1 TWebBrowser.ClientToWindow Method

Not used in TWebBrowser (see page 19147).

Pascal

```
procedure ClientToWindow(var pcx: SYSINT; var pcy: SYSINT);
```

C++

```
__fastcall ClientToWindow(SYSINT pcx, SYSINT pcy);
```

Description

Do not use this method. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. Internet Explorer uses ClientToWindow to convert between client coordinates and window coordinates.

132.1.12.2.2 TWebBrowser.CreateControl Method

Pascal

```
procedure CreateControl;
```

C++

```
__fastcall CreateControl();
```

Description

This is CreateControl, a member of class TWebBrowser.

132.1.12.2.3 ExecWB Method

This is the overview for the ExecWB method overload.

Overload List

	Name	Description
	TWebBrowser.ExecWB (OLECMDID, OLECMDEXECOPT) (see page 19163)	Executes a command.
	TWebBrowser.ExecWB (OLECMDID, OLECMDEXECOPT, OleVariant) (see page 19163)	Executes a command.
	TWebBrowser.ExecWB (OLECMDID, OLECMDEXECOPT, OleVariant, OleVariant) (see page 19163)	Executes a command.

132.1.12.2.3.1 TWebBrowser.ExecWB Method (OLECMDID, OLECMDEXECOPT)

Executes a command.

Pascal

```
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT); overload;
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT; var pvaIn: OleVariant); overload;
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT; var pvaIn: OleVariant; var
```

```
pvaOut: OleVariant); overload;
```

C++

```
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt);
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, OleVariant pvaIn);
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, OleVariant pvaIn, OleVariant
pvaOut);
```

Description

ExecWB lets you use the IOleCommandTarget interface to pass a command to the Web browser control. For more information, see the Microsoft documentation of IOleCommandTarget.

cmdID specifies the command to execute. The Web Browser does not support all of the commands for which Microsoft providesconstants. Use the QueryStatusWB (see page 19171) method to determine what commands can be used.

cmdexecopt modifies how the command is executed.

pvaIn specifies a Variant array that contains any input arguments. This value can be omitted if there are not input arguments.

pvaOutspecifies a Variant that receives the results of the command, if any. This value can be omitted if there are no results.

See Also

QueryStatusWB (see page 19171)

DoObjectVerb (see page 18051)

GetObjectVerbs (see page 18056)

132.1.12.2.4 TWebBrowser.Get_Application Method

Pascal

```
function Get_Application: IDispatch;
```

C++

```
__fastcall IDispatch Get_Application();
```

Description

This is Get_Application, a member of class TWebBrowser.

132.1.12.2.5 TWebBrowser.Get_Container Method

Pascal

```
function Get_Container: IDispatch;
```

C++

```
__fastcall IDispatch Get_Container();
```

Description

This is Get_Container, a member of class TWebBrowser.

132.1.12.2.6 TWebBrowser.Get_Document Method

Pascal

```
function Get_Document: IDispatch;
```

C++

```
__fastcall IDispatch Get_Document();
```

Description

This is Get_Document, a member of class TWebBrowser.

132.1.12.2.7 TWebBrowser.Get_Parent Method

Pascal

```
function Get_Parent: IDispatch;
```

C++

```
__fastcall IDispatch Get_Parent();
```

Description

This is Get_Parent, a member of class TWebBrowser.

132.1.12.2.8 TWebBrowser.GetProperty Method

Not used in TWebBrowser (see page 19147).

Pascal

```
function GetProperty(const Property_: WideString): OleVariant;
```

C++

```
__fastcall OleVariant GetProperty(const BSTR Property_);
```

Description

Do not use this method. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. Internet Explorer uses GetProperty to return a property value.

132.1.12.2.9 TWebBrowser.GoBack Method

Navigates to the previous item in the history list.

Pascal

```
procedure GoBack;
```

C++

```
__fastcall GoBack();
```

Description

Call GoBack to return to the last document in the history list. GoBack has no effect unless the history list contains additional documents.

Note: The Navigate (see page 19168) or Navigate2 (see page 19169) command can change the history list, depending on the specified flags.

See Also

OnCommandStateChange (see page 19201)

GoForward (see page 19166)

GoHome (see page 19166)

GoSearch (see page 19167)

Navigate (see page 19168)

Navigate2 (see page 19169)

132.1.12.2.10 TWebBrowser.GoForward Method

Navigates to the next item in the history list.

Pascal

```
procedure GoForward;
```

C++

```
__fastcall GoForward();
```

Description

Call GoForward to return to the next document in the history list. GoForward reverses the effect of the last GoBack (see page 19165) command.

See Also

OnCommandStateChange (see page 19201)

GoBack (see page 19165)

GoHome (see page 19166)

GoSearch (see page 19167)

132.1.12.2.11 TWebBrowser.GoHome Method

Navigates to the current start page.

Pascal

```
procedure GoHome;
```

C++

```
__fastcall GoHome();
```

Description

Call GoHome to navigate to the registered home or start page. The URL for this page is specified in the Registry (see page 18633) at

`HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main`

under the name Startpage. You can change this value using the TRegistry or TRegistryIniFile object, or using Navigation tab of the Internet Explorer's Options dialog.

See Also

GoBack (see page 19165)

GoForward (see page 19166)

GoSearch (see page 19167)

TRegistry (see page 18645)

TRegistryIniFile (see page 18675)

132.1.12.2.12 TWebBrowser.GoSearch Method

Navigates to the current search page.

Pascal

```
procedure GoSearch;
```

C++

```
__fastcall GoSearch();
```

Description

Call GoSearch to navigate to the registered search page. The URL for this page is specified in the Registry (see page 18633) at

`HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main`

under the name SearchPage. You can change this value using the TRegistry or TRegistryIniFile object, or using Navigation tab of the Internet Explorer's Options dialog.

See Also

GoBack (see page 19165)

GoForward (see page 19166)

GoHome (see page 19166)

TRegistry (see page 18645)

TRegistryIniFile (see page 18675)

132.1.12.2.13 TWebBrowser.InitControlData Method

Pascal

```
procedure InitControlData; override;
```

C++

```
virtual __fastcall InitControlData();
```

Description

This is InitControlData, a member of class TWebBrowser.

132.1.12.2.14 Navigate Method

This is the overview for the Navigate method overload.

Overload List

	Name	Description
	TWebBrowser.Navigate (WideString) (see page 19168)	Navigates to a resource identified by a path name or URL.
	TWebBrowser.Navigate (WideString, OleVariant) (see page 19168)	Navigates to a resource identified by a path name or URL.
	TWebBrowser.Navigate (WideString, OleVariant, OleVariant) (see page 19168)	Navigates to a resource identified by a path name or URL.

	TWebBrowser.Navigate (WideString, OleVariant, OleVariant, OleVariant) (see page 19168)	Navigates to a resource identified by a path name or URL.
	TWebBrowser.Navigate (WideString, OleVariant, OleVariant, OleVariant, OleVariant) (see page 19168)	Navigates to a resource identified by a path name or URL.

132.1.12.2.14.1 TWebBrowser.Navigate Method (WideString)

Navigates to a resource identified by a path name or URL.

Pascal

```
procedure Navigate(const URL: WideString); overload;
procedure Navigate(const URL: WideString; const Flags: OleVariant); overload;
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant); overload;
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant); overload;
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL);
__fastcall Navigate(const BSTR URL, const OleVariant Flags);
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName);
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData);
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

Use Navigate to locate and download a specific resource. Navigate can send an HTTP message to a specified URL and display the results or simply display the results of a specified file.

URL specifies the UNC path name of a file or the Uniform Resource Locator of an Internet resource that the Web browser should display.

Flags is a set of values that specify whether to add the resource to the history list, whether to read from or write to the cache, and whether to display the resource in a new window. It can be a sum of zero or more of the following:

Constant	Value	Meaning
navOpenInNewWindow (see page 19502)	1	Open the resource or file in a new window.
navNoHistory (see page 19501)	2	Do not add the resource or file to the history list. The new page replaces the current page in the list.
navNoReadFromCache (see page 19502)	4	Do not read from the disk cache for this navigation.
navNoWriteToCache (see page 19502)	8	Do not write the results of this navigation to the disk cache.
navAllowAutosearch (see page 19501)	16	If the navigation fails, the Web browser attempts to navigate common root domains (.com, .edu, and so on). If this still fails, the URL is passed to a search engine.

TargetFrameName is the name of the frame in which the resource will be displayed, or NULL if the resource should not be displayed in a named frame.

PostData contains the data sent to the server when using Navigate to generate an HTTP POST message. If PostData is NULL,

Navigate generates an HTTP GET message. PostData is ignored if URL does not specify an HTTP URL.

Headers contains any headers sent to the servers when the URL represents an HTTP URL. HTTP headers specify such things as the intended action required of the server, the type of data, and so on. (See TWebRequest object, whose properties represent many of the more common headers).

Note: To navigate to a location that can't be expressed as an URL (for example, to an item identifier list), use the Navigate2 (see page 19169) method instead.

See Also

Navigate2 (see page 19169)

GoBack (see page 19165)

GoForward (see page 19166)

GoHome (see page 19166)

GoSearch (see page 19167)

OnBeforeNavigate2 (see page 19200)

OnNavigateComplete2 (see page 19213)

TWebRequest (see page 15161)

132.1.12.2.15 Navigate2 Method

This is the overview for the Navigate2 method overload.

Overload List

	Name	Description
	TWebBrowser.Navigate2 (OleVariant) (see page 19169)	Navigates to a specified resource.
	TWebBrowser.Navigate2 (OleVariant, OleVariant) (see page 19169)	Navigates to a specified resource.
	TWebBrowser.Navigate2 (OleVariant, OleVariant, OleVariant) (see page 19169)	Navigates to a specified resource.
	TWebBrowser.Navigate2 (OleVariant, OleVariant, OleVariant, OleVariant) (see page 19169)	Navigates to a specified resource.
	TWebBrowser.Navigate2 (OleVariant, OleVariant, OleVariant, OleVariant, OleVariant) (see page 19169)	Navigates to a specified resource.

132.1.12.2.15.1 TWebBrowser.Navigate2 Method (OleVariant)

Navigates to a specified resource.

Pascal

```
procedure Navigate2(var URL: OleVariant); overload;
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant); overload;
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant; var TargetFrameName:
OleVariant); overload;
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant; var TargetFrameName:
OleVariant; var PostData: OleVariant); overload;
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant; var TargetFrameName:
```

```
OleVariant; var PostData: OleVariant; var Headers: OleVariant); overload;
```

C++

```
__fastcall Navigate2(OleVariant URL);
__fastcall Navigate2(OleVariant URL, OleVariant Flags);
__fastcall Navigate2(OleVariant URL, OleVariant Flags, OleVariant TargetFrameName);
__fastcall Navigate2(OleVariant URL, OleVariant Flags, OleVariant TargetFrameName, OleVariant
PostData);
__fastcall Navigate2(OleVariant URL, OleVariant Flags, OleVariant TargetFrameName, OleVariant
PostData, OleVariant Headers);
```

Description

Use Navigate2 to locate and download a specific resource. Navigate (see page 19168) can send an HTTP message to a specified URL and display the results, display the results of a specified file, or navigate to a resource that can't be expressed as an URL such as an item identifier list.

URL specifies the UNC path name of a file, the Uniform Resource Locator of an Internet resource, or a pointer to an item identifier list (PIDL).

Flags is a set of values that specify whether to add the resource to the history list, whether to read from or write to the cache, and whether to display the resource in a new window. It can be a sum of zero or more of the following:

Constant	Value	Meaning
navOpenInNewWindow (see page 19502)	1	Open the resource or file in a new window.
navNoHistory (see page 19501)	2	Do not add the resource or file to the history list. The new page replaces the current page in the list.
navNoReadFromCache (see page 19502)	4	Do not read from the disk cache for this navigation.
navNoWriteToCache (see page 19502)	8	Do not write the results of this navigation to the disk cache.
navAllowAutosearch (see page 19501)	16	If the navigation fails, the Web browser attempts to navigate common root domains (.com, .edu, and so on). If this still fails, the URL is passed to a search engine.

TargetFrameName is the name of the frame in which the resource will be displayed, or NULL if the resource should not be displayed in a named frame.

PostData contains the data sent to the server when using Navigate (see page 19168) to generate an HTTP POST message. If PostData is NULL, Navigate (see page 19168) generates an HTTP GET message. PostData is ignored if URL does not specify an HTTP URL.

Headers contains any headers sent to the servers when the URL represents an HTTP URL. HTTP headers specify such things as the intended action required of the server, the type of data, and so on. (See TWebRequest object, whose properties represent many of the more common headers).

See Also

Navigate (see page 19168)

GoBack (see page 19165)

GoForward (see page 19166)

GoHome (see page 19166)

GoSearch (see page 19167)

OnBeforeNavigate2 (see page 19200)

OnNavigateComplete2 (see page 19213)

TWebRequest (see page 15161)

132.1.12.2.16 TWebBrowser.PutProperty Method

Not used in TWebBrowser (see page 19147).

Pascal

```
procedure PutProperty(const Property_: WideString; vtValue: OleVariant);
```

C++

```
__fastcall PutProperty(const BSTR Property_, OleVariant vtValue);
```

Description

Do not use this method. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. Internet Explorer uses PutProperty to set a property value.

132.1.12.2.17 TWebBrowser.QueryStatusWB Method

Returns information about what commands the Web browser supports.

Pascal

```
function QueryStatusWB(cmdID: OLECMDID): OLECMDF;
```

C++

```
__fastcall OLECMDF QueryStatusWB(OLECMDID cmdID);
```

Description

Use QueryStatusWB to determine what commands can be executed using the ExecWB (see page 19163) method. QueryStatusWB uses the QueryStatus method of the IOleCommandTarget interface to obtain information about the status of commands. For more information, see the Microsoft documentation of IOleCommandTarget.

cmdID specifies the command about which you want information.

QueryStatusWB returns a value made by adding together flags that indicate the status of the command.

See Also

ExecWB (see page 19163)

132.1.12.2.18 TWebBrowser.Quit Method

Not used in TWebBrowser (see page 19147).

Pascal

```
procedure Quit;
```

C++

```
__fastcall Quit();
```

Description

Do not use this method. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. Internet Explorer responds to Quit by shutting down.

132.1.12.2.19 TWebBrowser.Refresh Method

Reloads the current document.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh to reload the current document. This is useful when the Web browser is displaying the resource at an URL that may have changed.

Refresh always sends a pragma:nocache header, telling the server not to return a cached copy. This can cause problems with some servers. To prevent the pragma:nocache header, or to refresh only expired pages, use the Refresh2 (see page 19172) method instead.

See Also

Refresh2 (see page 19172)

Document (see page 19182)

132.1.12.2.20 Refresh2 Method

This is the overview for the Refresh2 method overload.

Overload List

	Name	Description
	TWebBrowser.Refresh2 () (see page 19172)	Reloads the current document if necessary.
	TWebBrowser.Refresh2 (OleVariant) (see page 19172)	Reloads the current document if necessary.

132.1.12.2.20.1 TWebBrowser.Refresh2 Method ()

Reloads the current document if necessary.

Pascal

```
procedure Refresh2; overload;
procedure Refresh2(var Level: OleVariant); overload;
```

C++

```
__fastcall Refresh2();
__fastcall Refresh2(OleVariant Level);
```

Description

Call Refresh2 to reload the current document. Unlike the Refresh (see page 19172) method, Refresh2 lets you specify what level of refresh to perform.

Level indicates what type of information is refreshed. The following table lists the possible values:

Constant	Value	Meaning
REFRESH_NORMAL (see page 19526)	0	Perform a lightweight refresh that does not include the pragma:nocache header. The pragma:nocache header tells the server not to return a cached copy. This can cause problems with some servers.
REFRESH_IFEXPIRED (see page 19526)	1	Only refresh if the page has expired. Do not include the pragma:nocache header.
REFRESH_CONTINUE (see page 19525)	2	For internal use only.
REFRESH_COMPLETELY (see page 19525)	3	Perform a full refresh, including the pragma:nocache header. Using this option is the same as calling the Refresh (see page 19172) method.

When Level is omitted, a value of REFRESH_COMPLETELY (see page 19525) is assumed.

See Also

Refresh (see page 19172)

132.1.12.2.21 ShowBrowserBar Method

This is the overview for the ShowBrowserBar method overload.

Overload List

	Name	Description
	TWebBrowser.ShowBrowserBar (OleVariant) (see page 19173)	Not used in TWebBrowser (see page 19147)
	TWebBrowser.ShowBrowserBar (OleVariant, OleVariant) (see page 19173)	Not used in TWebBrowser (see page 19147)
	TWebBrowser.ShowBrowserBar (OleVariant, OleVariant, OleVariant) (see page 19173)	Not used in TWebBrowser (see page 19147)

132.1.12.2.21.1 TWebBrowser.ShowBrowserBar Method (OleVariant)

Not used in TWebBrowser (see page 19147)

Pascal

```
procedure ShowBrowserBar(var pvaClsid: OleVariant); overload;
procedure ShowBrowserBar(var pvaClsid: OleVariant; var pvarShow: OleVariant); overload;
procedure ShowBrowserBar(var pvaClsid: OleVariant; var pvarShow: OleVariant; var pvarSize:
OleVariant); overload;
```

C++

```
__fastcall ShowBrowserBar(OleVariant pvaClsid);
__fastcall ShowBrowserBar(OleVariant pvaClsid, OleVariant pvarShow);
__fastcall ShowBrowserBar(OleVariant pvaClsid, OleVariant pvarShow, OleVariant pvarSize);
```

Description

Do not use this method. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. Internet Explorer uses ShowBrowserBar to show or hide a specified browser bar.

132.1.12.2.22 TWebBrowser.Stop Method

Stops the current operation.

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

Stop cancels any pending navigation or download operation and terminates any dynamic page elements, such as background sounds and animations.

See Also

OnCommandStateChange (see page 19201)

GoForward (see page 19166)

GoHome (see page 19166)

GoSearch (see page 19167)

GoBack (see page 19165)

Navigate (see page 19168)

Navigate2 (see page 19169)

132.1.12.3 TWebBrowser Properties

The properties of the TWebBrowser class are listed here.

Public Properties

	Name	Description
R	Application (see page 19179)	Provides access to the Automation interface for the application that contains the WebBrowser (see page 19484) control, or, if that is unavailable, the interface for the WebBrowser (see page 19484) control.
R	Busy (see page 19180)	Indicates whether the WebBrowser (see page 19484) control is in the middle of a navigation or downloading operation.
R	Container (see page 19180)	Provides access to the automation interface of the Web browser's container.
R	ControlInterface (see page 19181)	Provides access to the interface for this Web browser component.
R	DefaultInterface (see page 19181)	Provides access to the default interface for the component.
R	Document (see page 19182)	Provides access to the Automation interface of the active document.
R	FullName (see page 19184)	Indicates the fully qualified path name for the executable that hosts the Web Browser component.
R	HWnd (see page 19185)	This is HWnd, a member of class TWebBrowser.
R	LocationName (see page 19185)	Identifies the short (UI-friendly) name of the URL or file currently displayed.
R	LocationURL (see page 19185)	Identifies the URL or path of the document or folder currently displayed.
R	Name (see page 19186)	Indicates the name for the executable that hosts the Web Browser component.
R	Parent (see page 19187)	Provides access to the Automation interface of the Web browser's container.
R	Path (see page 19188)	Indicates the name for the executable that hosts the Web Browser component.
R	ReadyState (see page 19193)	Indicates the state of the Web browser component.
R	TopLevelContainer (see page 19198)	Indicates whether the Web browser is a top-level Automation object.
R	type_ (see page 19199)	This is type_, a member of class TWebBrowser.

Published Properties

	Name	Description
	AddressBar (see page 19175)	Not used in TWebBrowser (see page 19147).
	Align (see page 19175)	Determines how the control aligns within its container (parent control).
	Anchors (see page 19179)	Specifies how the control is anchored to its parent.
	DragCursor (see page 19182)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 19182)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	FullScreen (see page 19184)	Not used in TWebBrowser (see page 19147).
	MenuBar (see page 19186)	Not used in TWebBrowser (see page 19147).
	Offline (see page 19186)	Indicates whether the Web browser gets its document from the local cache.
	ParentShowHint (see page 19187)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 19188)	Identifies the pop-up menu associated with the control.
	RegisterAsBrowser (see page 19193)	Registers the control as a top-level browser for target name resolution.
	RegisterAsDropTarget (see page 19194)	Specifies whether users can load a document using drag-and-drop.
	Resizable (see page 19194)	Not used in TWebBrowser (see page 19147).
	ShowHint (see page 19194)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Silent (see page 19196)	Specifies whether the Web browser can display dialog boxes.
	StatusBar (see page 19196)	Not used in TWebBrowser (see page 19147).
	StatusText (see page 19196)	Not used in TWebBrowser (see page 19147).
	TabOrder (see page 19197)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 19197)	Determines if the user can tab to a control.
	TheaterMode (see page 19198)	Not used in TWebBrowser (see page 19147).
	ToolBar (see page 19198)	Not used in TWebBrowser (see page 19147).
	Visible (see page 19199)	Specifies whether the Web browser component appears onscreen.

132.1.12.3.1 TWebBrowser.AddressBar Property

Not used in TWebBrowser (see page 19147).

Pascal

```
property AddressBar: WordBool;
```

C++

```
__property WordBool AddressBar;
```

Description

Do not use AddressBar. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. AddressBar shows or hides the URL address bar on Internet Explorer. TWebBrowser (see page 19147) does not have an address bar.

132.1.12.3.2 TWebBrowser.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form,

panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/
```

```
#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
```

```
    Dirs.Parent := Form1;
    Dirs.Align := alLeft;
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
```

```
    end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

132.1.12.3.3 TWebBrowser.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

132.1.12.3.4 TWebBrowser.Application Property

Provides access to the Automation interface for the application that contains the WebBrowser (see page 19484) control, or, if that is unavailable, the interface for the WebBrowser (see page 19484) control.

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

Use Application to access the interface for the application containing the WebBrowser (see page 19484) control. If the application does not have an Automation interface, or if the interface is not accessible, Application provides access to the Automation interface of the Web browser control.

See Also

Container (see page 19180)

Parent (see page 19187)

ControlInterface (see page 19181)

132.1.12.3.5 TWebBrowser.Busy Property

Indicates whether the WebBrowser (see page 19484) control is in the middle of a navigation or downloading operation.

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

Read Busy to determine whether the Web browser component is currently navigating to a new URL or downloading a document. If Busy is true, you can interrupt the current operation using the Stop (see page 19174) method before starting a new operation.

Note: The Web browser receives events at the beginning and end of each operation, as well as at discrete points during a download operation.

See Also

ReadyState (see page 19193)

OnDocumentComplete (see page 19201)

OnBeforeNavigate2 (see page 19200)

OnNavigateComplete2 (see page 19213)

OnDownloadComplete (see page 19202)

OnDownloadBegin (see page 19202)

OnProgressChange (see page 19215)

Stop (see page 19174)

132.1.12.3.6 TWebBrowser.Container Property

Provides access to the automation interface of the Web browser's container.

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

Use Container to access the interface for the Web browser's parent or container. If the Web browser does not have a container with an Automation interface, Container is nil.

See Also

Application (see page 19179)

Parent (see page 19187)

ControlInterface (see page 19181)

TopLevelContainer (see page 19198)

132.1.12.3.7 TWebBrowser.ControlInterface Property

Provides access to the interface for this Web browser component.

Pascal

```
property ControlInterface: IWebBrowser2;
```

C++

```
__property IWebBrowser2 ControlInterface;
```

Description

Use ControlInterface to access the interface for the WebBrowser (see page 19484) component. This interface lets an Automation controller access the properties, methods, and events of the Web browser.

See Also

Application (see page 19179)

Document (see page 19182)

Parent (see page 19187)

DefaultInterface (see page 19181)

132.1.12.3.8 TWebBrowser.DefaultInterface Property

Provides access to the default interface for the component.

Pascal

```
property DefaultInterface: IWebBrowser2;
```

C++

```
__property IWebBrowser2 DefaultInterface;
```

Description

Use DefaultInterface to access the default interface for the component. As implemented in TWebBrowser (see page 19147), this is the same as the ControlInterface (see page 19181) property.

See Also

ControlInterface (see page 19181)

132.1.12.3.9 TWebBrowser.Document Property

Provides access to the Automation interface of the active document.

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

Use Document to access an IDispatch interface for manipulating the contents of the HTML document through the COM object model. If there is no active document, Document is nil. That is, you must first load a document using the Navigate (see page 19168) or Navigate2 (see page 19169) method before you can manipulate it using the Document interface.

For more information about this interface, see the document object section of Microsoft's Dynamic HTML reference.

See Also

ControlInterface (see page 19181)

Navigate (see page 19168)

Navigate2 (see page 19169)

132.1.12.3.10 TWebBrowser.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

132.1.12.3.11 TWebBrowser.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
```

```
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

132.1.12.3.12 TWebBrowser.FullName Property

Indicates the fully qualified path name for the executable that hosts the Web Browser component.

Pascal

```
property FullName: WideString;
```

C++

```
__property BSTR FullName;
```

Description

Read FullName to obtain the name of the executable as a WideString. Unlike the Name (see page 19186) property, FullName specifies a fully qualified path name.

See Also

Application (see page 19179)

ExeName (see page 13709)

Name (see page 19186)

Path (see page 19188)

132.1.12.3.13 TWebBrowser.FullScreen Property

Not used in TWebBrowser (see page 19147).

Pascal

```
property FullScreen: WordBool;
```

C++

```
__property WordBool FullScreen;
```

Description

Do not use FullScreen. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. FullScreen determines whether Internet Explorer is maximized.

132.1.12.3.14 TWebBrowser.HWnd Property

Pascal

```
property HWnd: Integer;
```

C++

```
__property int HWnd;
```

Description

This is HWnd, a member of class TWebBrowser.

132.1.12.3.15 TWebBrowser.LocationName Property

Identifies the short (UI-friendly) name of the URL or file currently displayed.

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

Read LocationName to get the name of the resource currently displayed in the Web browser control. If the resource is an HTML page from the Web, LocationName is the title of that page. If the resource is a folder or file on the local network or on a disk, LocationName is the full UNC name of the folder or file.

See Also

LocationURL (see page 19185)

OnTitleChange (see page 19218)

132.1.12.3.16 TWebBrowser.LocationURL Property

Identifies the URL or path of the document or folder currently displayed.

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

Read LocationURL to get the name of the resource currently displayed in the Web browser control. If the resource is an HTML page from the Web, LocationName (see page 19185) is its URL. If the resource is a folder or file on the local network or on a disk, LocationName (see page 19185) is the full UNC name of the folder or file.

Note: LocationURL is a read-only property. To direct the Web browser to display the resource at a specific URL, folder, or file, use the Navigate (see page 19168) or Navigate2 (see page 19169) method.

See Also

LocationName (see page 19185)

Navigate (see page 19168)

Navigate2 (see page 19169)

132.1.12.3.17 TWebBrowser.MenuBar Property

Not used in TWebBrowser (see page 19147).

Pascal

```
property MenuBar: WordBool;
```

C++

```
__property WordBool MenuBar;
```

Description

Do not use MenuBar. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. MenuBar shows or hides the menu bar in Internet Explorer. TWebBrowser (see page 19147) does not have a menu bar.

132.1.12.3.18 TWebBrowser.Name Property

Indicates the name for the executable that hosts the Web Browser component.

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

Read Name to obtain the name of the executable that hosts the Web browser. Unlike the FullName (see page 19184) property, Name does not include the path.

See Also

Application (see page 19179)

FullName (see page 19184)

Path (see page 19188)

132.1.12.3.19 TWebBrowser.Offline Property

Indicates whether the Web browser gets its document from the local cache.

Pascal

```
property Offline: WordBool;
```

C++

```
__property WordBool Offline;
```

Description

Set Offline to true to work offline with the Web browser. When the Web browser is offline, it can't read HTML pages using an Internet connection. Instead, you must specify a source document as a UNC filename.

Note: You can use Internet Explorer to save HTML pages to a local cache so that they are available for offline work.

See Also

Navigate (see page 19168)

Navigate2 (see page 19169)

132.1.12.3.20 TWebBrowser.Parent Property

Provides access to the Automation interface of the Web browser's container.

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

Use Parent to access the interface for the Web browser's parent or container. If the Web browser does not have a container with an Automation interface, the value of Parent is the same as ControlInterface (see page 19181).

Use the TopLevelContainer (see page 19198) property to determine which interface is the value of Parent.

See Also

Container (see page 19180)

ControlInterface (see page 19181)

Application (see page 19179)

TopLevelContainer (see page 19198)

132.1.12.3.21 TWebBrowser.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

132.1.12.3.22 TWebBrowser.Path Property

Indicates the name for the executable that hosts the Web Browser component.

Pascal

```
property Path: WideString;
```

C++

```
__property BSTR Path;
```

Description

Read Path to obtain the path of the executable that hosts the Web browser.

See Also

FullName (see page 19184)

Name (see page 19186)

132.1.12.3.23 TWebBrowser.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}
```

```
/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
```

```
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
```

```
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

132.1.12.3.24 TWebBrowser.ReadyState Property

Indicates the state of the Web browser component.

Pascal

```
property ReadyState: TOleEnum;
```

C++

```
__property TOleEnum ReadyState;
```

Description

Read ReadyState to obtain state information specific to the Web browser control. Use this information to avoid attempting to use the Web browser control when it is not ready to perform an action.

The following table lists the possible values:

Constant	Value	Meaning
READYSTATE_UNINITIALIZED (see page 19525)	0	Default initialization state. The Control has not begun loading any properties.
READYSTATE_LOADING (see page 19524)	1	The control is loading its properties.
READYSTATE_LOADED (see page 19524)	2	The control has initialized its properties but is not yet interactive.
READYSTATE_INTERACTIVE (see page 19524)	3	The control is interactive, but not all of its data is available. (It is downloading a document or frame).
READYSTATE_COMPLETE (see page 19523)	4	The control is interactive and all of its data is available. (The current document is fully downloaded).

See Also

ComponentState (see page 2804)

ControlState (see page 5813)

Busy (see page 19180)

132.1.12.3.25 TWebBrowser.RegisterAsBrowser Property

Registers the control as a top-level browser for target name resolution.

Pascal

```
property RegisterAsBrowser: WordBool;
```

C++

```
__property WordBool RegisterAsBrowser;
```

Description

Set RegisterAsBrowser to true when using the Web browser control to implement a top-level browser.

132.1.12.3.26 TWebBrowser.RegisterAsDropTarget Property

Specifies whether users can load a document using drag-and-drop.

Pascal

```
property RegisterAsDropTarget: WordBool;
```

C++

```
__property WordBool RegisterAsDropTarget;
```

Description

Set RegisterAsDropTarget to true to allow users to drop HTML documents onto the Web browser. When RegisterAsDropTarget is true, users can load a document by dragging it onto the Web browser. When RegisterAsDropTarget is false, dragging a document onto the Web browser has no effect.

See Also

Navigate (see page 19168)

Navigate2 (see page 19169)

132.1.12.3.27 TWebBrowser.Resizable Property

Not used in TWebBrowser (see page 19147).

Pascal

```
property Resizable: WordBool;
```

C++

```
__property WordBool Resizable;
```

Description

Do not use Resizable. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. Resizable determines whether the Internet Explorer window can be resized.

132.1.12.3.28 TWebBrowser.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

132.1.12.3.29 TWebBrowser.Silent Property

Specifies whether the Web browser can display dialog boxes.

Pascal

```
property Silent: WordBool;
```

C++

```
__property WordBool Silent;
```

Description

Set Silent to true to prevent the Web browser control from displaying dialog boxes.

132.1.12.3.30 TWebBrowser.StatusBar Property

Not used in TWebBrowser (see page 19147).

Pascal

```
property StatusBar: WordBool;
```

C++

```
__property WordBool StatusBar;
```

Description

Do not use StatusBar. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. StatusBar shows or hides the status bar in Internet Explorer. TWebBrowser (see page 19147) does not have a status bar.

132.1.12.3.31 TWebBrowser.StatusText Property

Not used in TWebBrowser (see page 19147).

Pascal

```
property StatusText: WideString;
```

C++

```
__property BSTR StatusText;
```

Description

Do not use StatusText. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. StatusText specifies the text that appears on the status bar in Internet Explorer. TWebBrowser (see page 19147) does not have a status bar.

132.1.12.3.32 TWebBrowser.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

132.1.12.3.33 TWebBrowser.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

132.1.12.3.34 TWebBrowser.TheaterMode Property

Not used in TWebBrowser (see page 19147).

Pascal

```
property TheaterMode: WordBool;
```

C++

```
__property WordBool TheaterMode;
```

Description

Do not use TheaterMode. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. TheaterMode changes the display of Internet Explorer to use theater or normal mode.

132.1.12.3.35 TWebBrowser.ToolBar Property

Not used in TWebBrowser (see page 19147).

Pascal

```
property ToolBar: Integer;
```

C++

```
__property int ToolBar;
```

Description

Do not use ToolBar. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. ToolBar determines which toolbar is displayed in Internet Explorer. TWebBrowser (see page 19147) does not have any tool bars.

132.1.12.3.36 TWebBrowser.TopLevelContainer Property

Indicates whether the Web browser is a top-level Automation object.

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

Read TopLevelContainer to determine whether the Web browser has a parent with an Automation interface. When TopLevelContainer is false, the Container (see page 19180) and Parent (see page 19187) properties provide access to the parent's interface. When TopLevelContainer is true, the value of Container (see page 19180) is nil and the value of Parent (see page 19187) is the Web browser's own interface.

See Also

Container (see page 19180)

Parent (see page 19187)

132.1.12.3.37 TWebBrowser.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class TWebBrowser.

132.1.12.3.38 TWebBrowser.Visible Property

Specifies whether the Web browser component appears onscreen.

Pascal

```
property Visible: WordBool;
```

C++

```
__property WordBool Visible;
```

Description

Use Visible to show or hide the Web browser control.

See Also

Enabled (see page 5821)

132.1.12.4 TWebBrowser Events

The events of the TWebBrowser class are listed here.

Published Events

	Name	Description
	OnBeforeNavigate2 (see page 19200)	Occurs immediately before the Web browser navigates to a new resource.
	OnClientToHostWindow (see page 19200)	This is OnClientToHostWindow, a member of class TWebBrowser.
	OnCommandStateChange (see page 19201)	Occurs when the ability to execute certain TWebBrowser (see page 19147) methods changes.
	OnDocumentComplete (see page 19201)	Occurs when the document being navigated to reaches ReadyState_Complete.
	OnDownloadBegin (see page 19202)	Occurs when the Web browser starts downloading a document.
	OnDownloadComplete (see page 19202)	Occurs when a navigation operation finishes, is halted, or fails.
	OnDragDrop (see page 19203)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 19205)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 19207)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 19208)	Occurs when a control receives the input focus.
	OnExit (see page 19211)	Occurs when the input focus shifts away from one control to another.
	OnFileDownload (see page 19212)	This is OnFileDownload, a member of class TWebBrowser.
	OnFullScreen (see page 19213)	Not used in TWebBrowser (see page 19147).
	OnMenuBar (see page 19213)	Not used in TWebBrowser (see page 19147).
	OnNavigateComplete2 (see page 19213)	Occurs immediately after the Web browser successfully navigates to a new location.
	OnNavigateError (see page 19214)	This is OnNavigateError, a member of class TWebBrowser.
	OnNewWindow2 (see page 19214)	Occurs when a new, hidden, non-navigated Web browser window is needed.

	OnPrintTemplateInstantiation (see page 19214)	This is OnPrintTemplateInstantiation, a member of class TWebBrowser.
	OnPrintTemplateTeardown (see page 19215)	This is OnPrintTemplateTeardown, a member of class TWebBrowser.
	OnPrivacyImpactedStateChange (see page 19215)	This is OnPrivacyImpactedStateChange, a member of class TWebBrowser.
	OnProgressChange (see page 19215)	Occurs when the progress of a download operation is updated.
	OnPropertyChange (see page 19215)	Not used in TWebBrowser (see page 19147).
	OnQuit (see page 19216)	Not used in TWebBrowser (see page 19147).
	OnSetSecureLockIcon (see page 19216)	This is OnSetSecureLockIcon, a member of class TWebBrowser.
	OnStartDrag (see page 19216)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnStatusBar (see page 19217)	Not used in TWebBrowser (see page 19147).
	OnStatusTextChange (see page 19217)	Not used in TWebBrowser (see page 19147).
	OnTheaterMode (see page 19217)	Not used in TWebBrowser (see page 19147).
	OnTitleChange (see page 19218)	Occurs when the title of a document in the WebBrowser (see page 19484) control becomes available or changes.
	OnToolBar (see page 19218)	Not used in TWebBrowser (see page 19147).
	OnUpdatePageStatus (see page 19218)	This is OnUpdatePageStatus, a member of class TWebBrowser.
	OnVisible (see page 19219)	Occurs when the Web browser window is about to be shown or hidden.
	OnWindowClosing (see page 19219)	This is OnWindowClosing, a member of class TWebBrowser.
	OnWindowSetHeight (see page 19219)	This is OnWindowSetHeight, a member of class TWebBrowser.
	OnWindowSetLeft (see page 19219)	This is OnWindowSetLeft, a member of class TWebBrowser.
	OnWindowSetResizable (see page 19220)	This is OnWindowSetResizable, a member of class TWebBrowser.
	OnWindowSetTop (see page 19220)	This is OnWindowSetTop, a member of class TWebBrowser.
	OnWindowSetWidth (see page 19220)	This is OnWindowSetWidth, a member of class TWebBrowser.

132.1.12.4.1 TWebBrowser.OnBeforeNavigate2 Event

Occurs immediately before the Web browser navigates to a new resource.

Pascal

```
property OnBeforeNavigate2: TWebBrowserBeforeNavigate2;
```

C++

```
__property TWebBrowserBeforeNavigate2 OnBeforeNavigate2;
```

Description

Write an OnBeforeNavigate2 event handler to redirect or cancel a change to a different URL. This event may occur as the result of a call to the Navigate (see page 19168) or Navigate2 (see page 19169) method, or the user clicking a link.

See Also

Navigate (see page 19168)

Navigate2 (see page 19169)

OnNavigateComplete2 (see page 19213)

TWebRequest (see page 15161)

132.1.12.4.2 TWebBrowser.OnClientToHostWindow Event

Pascal

```
property OnClientToHostWindow: TWebBrowserClientToHostWindow;
```

C++

```
__property TWebBrowserClientToHostWindow OnClientToHostWindow;
```

Description

This is OnClientToHostWindow, a member of class TWebBrowser.

132.1.12.4.3 TWebBrowser.OnCommandStateChange Event

Occurs when the ability to execute certain TWebBrowser (see page 19147) methods changes.

Pascal

```
property OnCommandStateChange: TWebBrowserCommandStateChange;
```

C++

```
__property TWebBrowserCommandStateChange OnCommandStateChange;
```

Description

Write an OnCommandStateChange to update any controls in the application whose state depends on the ability to execute the Web browser's methods.

See Also

Stop (see page 19174)

GoBack (see page 19165)

GoForward (see page 19166)

Busy (see page 19180)

QueryStatusWB (see page 19171)

132.1.12.4.4 TWebBrowser.OnDocumentComplete Event

Occurs when the document being navigated to reaches ReadyState_Complete.

Pascal

```
property OnDocumentComplete: TWebBrowserDocumentComplete;
```

C++

```
__property TWebBrowserDocumentComplete OnDocumentComplete;
```

Description

Write an OnDocumentComplete event handler to take specific action when a frame or document is fully loaded into the Web browser .For a document without frames, this event occurs once when the document finishes loading. On a document containing multiple frames, this event occurs once for each frame. When the multiple-frame document finishes loading, the Web browser fires the event one final time.

See Also

ReadyState (see page 19193)

OnDownloadBegin (see page 19202)

OnDownloadComplete (see page 19202)

Busy (see page 19180)

Navigate (see page 19168)

Navigate2 (see page 19169)

OnBeforeNavigate2 (see page 19200)

132.1.12.4.5 TWebBrowser.OnDownloadBegin Event

Occurs when the Web browser starts downloading a document.

Pascal

```
property OnDownloadBegin: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownloadBegin;
```

Description

Write an OnDownloadBegin event handler to take specific action after the Web browser has located a document and immediately before it starts downloading the document. For example, use the OnDownloadBegin event to launch an animation control the represents downloading or a progress bar that is updated by an OnProgressChange (see page 19215) event handler. The control can then be stopped in an OnDownloadComplete (see page 19202) event handler.

Note: To take specific action when the Web browser looks up the resource, rather than when it begins downloading, use the OnBeforeNavigate2 (see page 19200) event. OnDownloadBegin occurs shortly after OnBeforeNavigate2 (see page 19200).

See Also

OnDocumentComplete (see page 19201)

OnDownloadComplete (see page 19202)

Navigate (see page 19168)

Navigate2 (see page 19169)

OnBeforeNavigate2 (see page 19200)

TAnimate (see page 3422)

OnNavigateComplete2 (see page 19213)

OnProgressChange (see page 19215)

132.1.12.4.6 TWebBrowser.OnDownloadComplete Event

Occurs when a navigation operation finishes, is halted, or fails.

Pascal

```
property OnDownloadComplete: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownloadComplete;
```

Description

Write an OnDownLoadComplete event handler to take specific action after the Web browser stops a downloading operation. For example, use the OnDownloadComplete event to stop an download indication that is started in an OnDownloadBegin (see page 19202) event handler.

Note: Unlike the OnNavigateComplete2 (see page 19213) event, OnDownloadComplete occurs even if the Web browser does not successfully navigate to an URL.

See Also

OnDocumentComplete (see page 19201)

OnDownloadBegin (see page 19202)

Navigate (see page 19168)

Navigate2 (see page 19169)

OnNavigateComplete2 (see page 19213)

132.1.12.4.7 TWebBrowser.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
```

```
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

132.1.12.4.8 TWebBrowser.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
```

```
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
```

```
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

132.1.12.4.9 TWebBrowser.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

132.1.12.4.10 TWebBrowser.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
```

```
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;
```

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

132.1.12.4.11 TWebBrowser.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
```

```
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

132.1.12.4.12 TWebBrowser.OnFileDownload Event

Pascal

```
property OnFileDownload: TWebBrowserFileDownload;
```

C++

```
__property TWebBrowserFileDownload OnFileDownload;
```

Description

This is OnFileDownload, a member of class TWebBrowser.

132.1.12.4.13 TWebBrowser.OnFullScreen Event

Not used in TWebBrowser (see page 19147).

Pascal

```
property OnFullScreen: TWebBrowserOnFullScreen;
```

C++

```
__property TWebBrowserOnFullScreen OnFullScreen;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. OnFullScreen occurs with Internet Explorer when the application is maximized or restored from a maximized state.

132.1.12.4.14 TWebBrowser.OnMenuBar Event

Not used in TWebBrowser (see page 19147).

Pascal

```
property OnMenuBar: TWebBrowserOnMenuBar;
```

C++

```
__property TWebBrowserOnMenuBar OnMenuBar;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. OnMenuBar occurs with Internet Explorer when the menu bar is hidden or shown.

132.1.12.4.15 TWebBrowser.OnNavigateComplete2 Event

Occurs immediately after the Web browser successfully navigates to a new location.

Pascal

```
property OnNavigateComplete2: TWebBrowserNavigateComplete2;
```

C++

```
__property TWebBrowserNavigateComplete2 OnNavigateComplete2;
```

Description

Write an OnNavigateComplete2 event handler to take specific action when the Web browser successfully navigates to a new resource. The event can occur before the document is fully downloaded, but when it occurs at least part of the document must be received and a viewer for the document created.

Note: Unlike the OnDownloadComplete (see page 19202) event, OnNavigateComplete2 does not occur if the operation is not successful.

See Also

OnDocumentComplete (see page 19201)

OnDownloadBegin (see page 19202)

Navigate (see page 19168)

Navigate2 (see page 19169)

OnDownloadComplete (see page 19202)

OnBeforeNavigate2 (see page 19200)

132.1.12.4.16 TWebBrowser.OnNavigateError Event

Pascal

```
property OnNavigateError: TWebBrowserNavigateError;
```

C++

```
__property TWebBrowserNavigateError OnNavigateError;
```

Description

This is OnNavigateError, a member of class TWebBrowser.

132.1.12.4.17 TWebBrowser.OnNewWindow2 Event

Occurs when a new, hidden, non-navigated Web browser window is needed.

Pascal

```
property OnNewWindow2: TWebBrowserNewWindow2;
```

C++

```
__property TWebBrowserNewWindow2 OnNewWindow2;
```

Description

Write an OnNewWindow2 to take specific action immediately before the Web browser creates a new window for displaying a resource. This window may be needed because the user shift-clicked on a link, the user right-clicked on a link and chose "open in new window", the frame for the target URL does not yet exist, or the Navigate (see page 19168) or Navigate2 (see page 19169) requested that the target document appear in a new window.

See Also

OnBeforeNavigate2 (see page 19200)

132.1.12.4.18 TWebBrowser.OnPrintTemplateInstantiation Event

Pascal

```
property OnPrintTemplateInstantiation: TWebBrowserPrintTemplateInstantiation;
```

C++

```
__property TWebBrowserPrintTemplateInstantiation OnPrintTemplateInstantiation;
```

Description

This is OnPrintTemplateInstantiation, a member of class TWebBrowser.

132.1.12.4.19 TWebBrowser.OnPrintTemplateTeardown Event

Pascal

```
property OnPrintTemplateTeardown: TWebBrowserPrintTemplateTeardown;
```

C++

```
__property TWebBrowserPrintTemplateTeardown OnPrintTemplateTeardown;
```

Description

This is OnPrintTemplateTeardown, a member of class TWebBrowser.

132.1.12.4.20 TWebBrowser.OnPrivacyImpactedStateChange Event

Pascal

```
property OnPrivacyImpactedStateChange: TWebBrowserPrivacyImpactedStateChange;
```

C++

```
__property TWebBrowserPrivacyImpactedStateChange OnPrivacyImpactedStateChange;
```

Description

This is OnPrivacyImpactedStateChange, a member of class TWebBrowser.

132.1.12.4.21 TWebBrowser.OnProgressChange Event

Occurs when the progress of a download operation is updated.

Pascal

```
property OnProgressChange: TWebBrowserProgressChange;
```

C++

```
__property TWebBrowserProgressChange OnProgressChange;
```

Description

Write an OnProgressChange event handler to provide visual feedback about the download process. For example, an OnProgressChange event handler can update a TProgressBar component or display the number of bytes downloaded so far.

See Also

OnDocumentComplete (see page 19201)

OnDownloadBegin (see page 19202)

Navigate (see page 19168)

Navigate2 (see page 19169)

OnDownloadComplete (see page 19202)

TProgressBar (see page 4615)

132.1.12.4.22 TWebBrowser.OnPropertyChange Event

Not used in TWebBrowser (see page 19147).

Pascal

```
property OnPropertyChange: TWebBrowserPropertyChange;
```

C++

```
__property TWebBrowserPropertyChange OnPropertyChange;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. OnPropertyChange occurs with Internet Explorer when a property is modified using the PutProperty (see page 19171) method.

132.1.12.4.23 TWebBrowser.OnQuit Event

Not used in TWebBrowser (see page 19147).

Pascal

```
property OnQuit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnQuit;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. OnQuit occurs when the Internet Explorer is about to shut down.

132.1.12.4.24 TWebBrowser.OnSetSecureLockIcon Event

Pascal

```
property OnSetSecureLockIcon: TWebBrowserSetSecureLockIcon;
```

C++

```
__property TWebBrowserSetSecureLockIcon OnSetSecureLockIcon;
```

Description

This is OnSetSecureLockIcon, a member of class TWebBrowser.

132.1.12.4.25 TWebBrowser.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag

cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

132.1.12.4.26 TWebBrowser.OnStatusBar Event

Not used in TWebBrowser (see page 19147).

Pascal

```
property OnStatusBar: TWebBrowserOnStatusBar;
```

C++

```
__property TWebBrowserOnStatusBar OnStatusBar;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. OnStatusBar occurs when the Internet Explorer first shows or hides its status bar.

132.1.12.4.27 TWebBrowser.OnStatusTextChange Event

Not used in TWebBrowser (see page 19147).

Pascal

```
property OnStatusTextChange: TWebBrowserStatusTextChange;
```

C++

```
__property TWebBrowserStatusTextChange OnStatusTextChange;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. OnStatusTextChange occurs when the text displayed in the Internet Explorer's status bar changes.

132.1.12.4.28 TWebBrowser.OnTheaterMode Event

Not used in TWebBrowser (see page 19147).

Pascal

```
property OnTheaterMode: TWebBrowserOnTheaterMode;
```

C++

```
__property TWebBrowserOnTheaterMode OnTheaterMode;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. OnTheaterMode occurs when the Internet Explorer changes into or out of theater mode.

132.1.12.4.29 TWebBrowser.OnTitleChange Event

Occurs when the title of a document in the WebBrowser (see page 19484) control becomes available or changes.

Pascal

```
property OnTitleChange: TWebBrowserTitleChange;
```

C++

```
__property TWebBrowserTitleChange OnTitleChange;
```

Description

Write an OnTitleChange event handler to respond when the Web browser obtains information about the document title. Because the title can change while an HTML page is downloading, LocationName (see page 19185) is originally set to the URL of the document. After the title specified in the HTML page, if any, becomes available, LocationName (see page 19185) is changed to reflect the actual title.

See Also

LocationName (see page 19185)

132.1.12.4.30 TWebBrowser.OnToolBar Event

Not used in TWebBrowser (see page 19147).

Pascal

```
property OnToolBar: TWebBrowserOnToolBar;
```

C++

```
__property TWebBrowserOnToolBar OnToolBar;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 19147) interface because the IWebBrowser2 (see page 19403) interface is shared with Internet Explorer. OnToolBar occurs when the Internet Explorer changes which tool bars are visible.

132.1.12.4.31 TWebBrowser.OnUpdatePageStatus Event

Pascal

```
property OnUpdatePageStatus: TWebBrowserUpdatePageStatus;
```

C++

```
__property TWebBrowserUpdatePageStatus OnUpdatePageStatus;
```

Description

This is OnUpdatePageStatus, a member of class TWebBrowser.

132.1.12.4.32 TWebBrowser.OnVisible Event

Occurs when the Web browser window is about to be shown or hidden.

Pascal

```
property OnVisible: TWebBrowserOnVisible;
```

C++

```
__property TWebBrowserOnVisible OnVisible;
```

Description

Write an OnVisible event handler to make adjustments to compensate for the appearance or disappearance of the Web browser window.

See Also

Visible (see page 19199)

132.1.12.4.33 TWebBrowser.OnWindowClosing Event

Pascal

```
property OnWindowClosing: TWebBrowserWindowClosing;
```

C++

```
__property TWebBrowserWindowClosing OnWindowClosing;
```

Description

This is OnWindowClosing, a member of class TWebBrowser.

132.1.12.4.34 TWebBrowser.OnWindowSetHeight Event

Pascal

```
property OnWindowSetHeight: TWebBrowserWindowSetHeight;
```

C++

```
__property TWebBrowserWindowSetHeight OnWindowSetHeight;
```

Description

This is OnWindowSetHeight, a member of class TWebBrowser.

132.1.12.4.35 TWebBrowser.OnWindowSetLeft Event

Pascal

```
property OnWindowSetLeft: TWebBrowserWindowSetLeft;
```

C++

```
__property TWebBrowserWindowSetLeft OnWindowSetLeft;
```

Description

This is OnWindowSetLeft, a member of class TWebBrowser.

132.1.12.4.36 TWebBrowser.OnWindowSetResizable Event

Pascal

```
property OnWindowSetResizable: TWebBrowserWindowSetResizable;
```

C++

```
__property TWebBrowserWindowSetResizable OnWindowSetResizable;
```

Description

This is OnWindowSetResizable, a member of class TWebBrowser.

132.1.12.4.37 TWebBrowser.OnWindowSetTop Event

Pascal

```
property OnWindowSetTop: TWebBrowserWindowSetTop;
```

C++

```
__property TWebBrowserWindowSetTop OnWindowSetTop;
```

Description

This is OnWindowSetTop, a member of class TWebBrowser.

132.1.12.4.38 TWebBrowser.OnWindowSetWidth Event

Pascal

```
property OnWindowSetWidth: TWebBrowserWindowSetWidth;
```

C++

```
__property TWebBrowserWindowSetWidth OnWindowSetWidth;
```

Description

This is OnWindowSetWidth, a member of class TWebBrowser.

132.1.13 TWebBrowser_V1 Class

Class Hierarchy

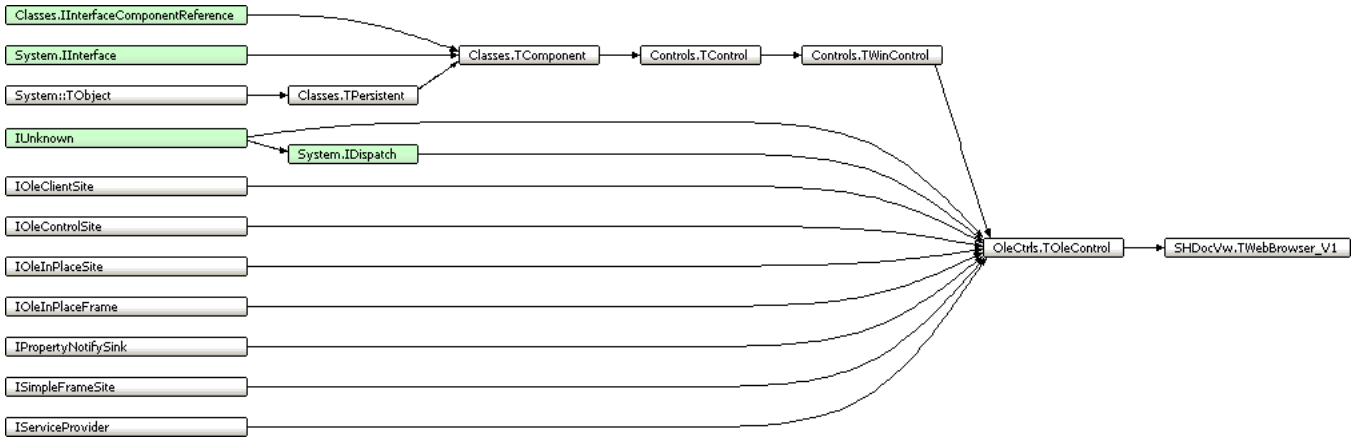

Pascal

```
TWebBrowser_V1 = class(TOleControl);
```

C++

```
class TWebBrowser_V1 : public TOleControl;
```

File

SHDocVw

Description

This is class SHDocVw.TWebBrowser_V1.

132.1.13.1 TWebBrowser_V1 Members

The following tables list the members exposed by TWebBrowser_V1.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.

	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_AddRef (see page 18046)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	_Release (see page 18047)	Called when an application releases a component interface.
	_SetColorProp (see page 18047)	This is _SetColorProp, a member of class TOleControl.
	_SetTColorProp (see page 18048)	This is _SetTColorProp, a member of class TOleControl.
	_SetTFontProp (see page 18048)	This is _SetTFontProp, a member of class TOleControl.
	_SetTOleEnumProp (see page 18048)	This is _SetTOleEnumProp, a member of class TOleControl.
	_SetTPictureProp (see page 18048)	This is _SetTPictureProp, a member of class TOleControl.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.

	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanInPlaceActivate (see page 18049)	This is CanInPlaceActivate, a member of class TOleControl.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ContextSensitiveHelp (see page 18049)	This is ContextSensitiveHelp, a member of class TOleControl.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateControl (see page 19235)	This is CreateControl, a member of class TWebBrowser_V1.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 18049)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DeactivateAndUndo (see page 18050)	This is DeactivateAndUndo, a member of class TOleControl.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DefineProperties (see page 18050)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWindowHandle (see page 18051)	Destroys the window created in the CreateWindowHandle method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DiscardUndoState (see page 18051)	This is DiscardUndoState, a member of class TOleControl.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.

	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EnableModeless (see page 18052)	This is EnableModeless, a member of class TOleControl.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	Get_Application (see page 19236)	This is Get_Application, a member of class TWebBrowser_V1.
	Get_Container (see page 19236)	This is Get_Container, a member of class TWebBrowser_V1.
	Get_Document (see page 19236)	This is Get_Document, a member of class TWebBrowser_V1.
	Get_Parent (see page 19236)	This is Get_Parent, a member of class TWebBrowser_V1.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorder (see page 18052)	This is GetBorder, a member of class TOleControl.
	GetByteProp (see page 18052)	This is GetByteProp, a member of class TOleControl.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetColorProp (see page 18053)	This is GetColorProp, a member of class TOleControl.
	GetCompProp (see page 18053)	This is GetCompProp, a member of class TOleControl.
	GetContainer (see page 18053)	This is GetContainer, a member of class TOleControl.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetCurrencyProp (see page 18053)	This is GetCurrencyProp, a member of class TOleControl.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.

	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDoubleProp (see page 18053)	This is GetDoubleProp, a member of class TOleControl.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetExtendedControl (see page 18054)	This is GetExtendedControl, a member of class TOleControl.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDispatchProp (see page 18055)	This is GetIDispatchProp, a member of class TOleControl.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIDsOfNames (see page 18055)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIntegerProp (see page 18055)	This is GetIntegerProp, a member of class TOleControl.
	GetIUnknownProp (see page 18055)	This is GetIUnknownProp, a member of class TOleControl.
	GetMoniker (see page 18056)	This is GetMoniker, a member of class TOleControl.
	GetOleBoolProp (see page 18056)	This is GetOleBoolProp, a member of class TOleControl.
	GetOleDateProp (see page 18056)	This is GetOleDateProp, a member of class TOleControl.
	GetOleEnumProp (see page 18057)	This is GetOleEnumProp, a member of class TOleControl.
	GetOleVariantProp (see page 18057)	This is GetOleVariantProp, a member of class TOleControl.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetProperty (see page 18058)	This is GetProperty, a member of class TOleControl.
	GetShortIntProp (see page 18058)	This is GetShortIntProp, a member of class TOleControl.
	GetSingleProp (see page 18058)	This is GetSingleProp, a member of class TOleControl.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetSmallintProp (see page 18058)	This is GetSmallintProp, a member of class TOleControl.
	GetStringProp (see page 18059)	This is GetStringProp, a member of class TOleControl.
	GetTColorProp (see page 18059)	This is GetTColorProp, a member of class TOleControl.
	GetTDateTimeProp (see page 18059)	This is GetTDateTimeProp, a member of class TOleControl.
	GetTFontProp (see page 18059)	This is GetTFontProp, a member of class TOleControl.
	GetTOleEnumProp (see page 18059)	This is GetTOleEnumProp, a member of class TOleControl.
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTPictureProp (see page 18060)	This is GetTPictureProp, a member of class TOleControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfo (see page 18060)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetTypeInfoCount (see page 18060)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariantProp (see page 18061)	This is GetVariantProp, a member of class TOleControl.
	GetWideStringProp (see page 18061)	This is GetWideStringProp, a member of class TOleControl.
	GetWindowContext (see page 18061)	This is GetWindowContext, a member of class TOleControl.
	GetWordBoolProp (see page 18061)	This is GetWordBoolProp, a member of class TOleControl.
	GetWordProp (see page 18062)	This is GetWordProp, a member of class TOleControl.
V	InitControlData (see page 19237)	This is InitControlData, a member of class TWebBrowser_V1.
V	InitControlInterface (see page 18062)	This is InitControlInterface, a member of class TOleControl.
	InsertMenus (see page 18062)	This is InsertMenus, a member of class TOleControl.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.

	Invoke (see page 18062)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	InvokeEvent (see page 18063)	This is InvokeEvent, a member of class TOleControl.
	InvokeMethod (see page 18063)	This is InvokeMethod, a member of class TOleControl.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	LockInPlaceActive (see page 18064)	This is LockInPlaceActive, a member of class TOleControl.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	OleControlSite_TranslateAccelerator (see page 18064)	This is OleControlSite_TranslateAccelerator, a member of class TOleControl.
	OleInPlaceFrame_GetWindow (see page 18064)	This is OleInPlaceFrame_GetWindow, a member of class TOleControl.
	OleInPlaceFrame_TranslateAccelerator (see page 18065)	This is OleInPlaceFrame_TranslateAccelerator, a member of class TOleControl.
	OleInPlaceSite_GetWindow (see page 18065)	This is OleInPlaceSite_GetWindow, a member of class TOleControl.
V	OnChanged (see page 18065)	This is OnChanged, a member of class TOleControl.
	OnControlInfoChanged (see page 18065)	This is OnControlInfoChanged, a member of class TOleControl.
	OnFocus (see page 18066)	This is OnFocus, a member of class TOleControl.
	OnInPlaceActivate (see page 18066)	This is OnInPlaceActivate, a member of class TOleControl.
	OnInPlaceDeactivate (see page 18066)	This is OnInPlaceDeactivate, a member of class TOleControl.
	OnPosRectChange (see page 18066)	This is OnPosRectChange, a member of class TOleControl.
V	OnRequestEdit (see page 18066)	This is OnRequestEdit, a member of class TOleControl.
	OnShowWindow (see page 18067)	This is OnShowWindow, a member of class TOleControl.
	OnUIActivate (see page 18067)	This is OnUIActivate, a member of class TOleControl.
	OnUIDeactivate (see page 18067)	This is OnUIDeactivate, a member of class TOleControl.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
V	PaletteChanged (see page 18067)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PictureChanged (see page 18068)	This is PictureChanged, a member of class TOleControl.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	PostMessageFilter (see page 18068)	This is PostMessageFilter, a member of class TOleControl.
	PreMessageFilter (see page 18069)	This is PreMessageFilter, a member of class TOleControl.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	QueryInterface (see page 18069)	Returns a reference to a specified interface if the object supports that interface.
	QueryService (see page 18069)	This is QueryService, a member of class TOleControl.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.

	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveMenus (see page 18070)	This is RemoveMenus, a member of class TOleControl.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	RequestBorderSpace (see page 18070)	This is RequestBorderSpace, a member of class TOleControl.
	RequestNewObjectLayout (see page 18070)	This is RequestNewObjectLayout, a member of class TOleControl.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	SaveObject (see page 18070)	This is SaveObject, a member of class TOleControl.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Scroll (see page 18070)	This is Scroll, a member of class TOleControl.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActiveObject (see page 18071)	This is SetActiveObject, a member of class TOleControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetBorderSpace (see page 18071)	This is SetBorderSpace, a member of class TOleControl.
	SetByteProp (see page 18071)	This is SetByteProp, a member of class TOleControl.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetColorProp (see page 18072)	This is SetColorProp, a member of class TOleControl.
	SetCompProp (see page 18072)	This is SetCompProp, a member of class TOleControl.
	SetCurrencyProp (see page 18072)	This is SetCurrencyProp, a member of class TOleControl.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDoubleProp (see page 18072)	This is SetDoubleProp, a member of class TOleControl.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIDispatchProp (see page 18073)	This is SetIDispatchProp, a member of class TOleControl.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetIntegerProp (see page 18073)	This is SetIntegerProp, a member of class TOleControl.
	SetIUnknownProp (see page 18073)	This is SetIUnknownProp, a member of class TOleControl.
	SetMenu (see page 18073)	This is SetMenu, a member of class TOleControl.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetName (see page 18073)	Sets the name of the control.
	SetOleBoolProp (see page 18074)	This is SetOleBoolProp, a member of class TOleControl.
	SetOleDateProp (see page 18074)	This is SetOleDateProp, a member of class TOleControl.
	SetOleEnumProp (see page 18074)	This is SetOleEnumProp, a member of class TOleControl.
	SetOleVariantProp (see page 18075)	This is SetOleVariantProp, a member of class TOleControl.

	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParent (see page 18075)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetProperty (see page 18076)	This is SetProperty, a member of class TOleControl.
	SetShortIntProp (see page 18076)	This is SetShortIntProp, a member of class TOleControl.
	SetSingleProp (see page 18076)	This is SetSingleProp, a member of class TOleControl.
	SetSmallintProp (see page 18076)	This is SetSmallintProp, a member of class TOleControl.
	SetStatusText (see page 18077)	This is SetStatusText, a member of class TOleControl.
	SetStringProp (see page 18077)	This is SetStringProp, a member of class TOleControl.
	SetTColorProp (see page 18077)	This is SetTColorProp, a member of class TOleControl.
	SetTDateTimeProp (see page 18077)	This is SetTDateTimeProp, a member of class TOleControl.
	SetTFontProp (see page 18077)	This is SetTFontProp, a member of class TOleControl.
	SetTOleEnumProp (see page 18078)	This is SetTOleEnumProp, a member of class TOleControl.
	SetTPictureProp (see page 18078)	This is SetTPictureProp, a member of class TOleControl.
	SetVariantProp (see page 18078)	This is SetVariantProp, a member of class TOleControl.
	SetWideStringProp (see page 18078)	This is SetWideStringProp, a member of class TOleControl.
	SetWordBoolProp (see page 18079)	This is SetWordBoolProp, a member of class TOleControl.
	SetWordProp (see page 18079)	This is SetWordProp, a member of class TOleControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	ShowObject (see page 18079)	This is ShowObject, a member of class TOleControl.
	ShowPropertyFrame (see page 18079)	This is ShowPropertyFrame, a member of class TOleControl.
	StandardEvent (see page 18080)	This is StandardEvent, a member of class TOleControl.
	SuppressException (see page 18080)	This is SuppressException, a member of class TOleControl.
	TransformCoords (see page 18080)	This is TransformCoords, a member of class TOleControl.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WndProc (see page 18080)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	ControlData (see page 18086)	This is ControlData, a member of class TOleControl.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	ServiceQuery (see page 18087)	This is ServiceQuery, a member of class TOleControl.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	BrowseProperties (see page 18048)	Notifies the ActiveX control to display its Properties dialog box.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 18043)	Creates a new instance of TOleControl (see page 18030).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 18050)	Handles all otherwise-unhandled messages.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 18043)	Frees the memory associated with this instance of TOleControl (see page 18030).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DoObjectVerb (see page 18051)	Notifies the ActiveX control to perform one of its actions.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetEnumPropDesc (see page 18054)	Returns descriptions of the ActiveX control's properties.
	GetHelpContext (see page 18054)	Retrieves the Help context of a member of the ActiveX control.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetObjectVerbs (see page 18056)	Retrieves the names of the verbs implemented by the ActiveX control.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetPropDisplayString (see page 18057)	Returns the display string for an interface for the ActiveX control.
	GetPropDisplayStrings (see page 18057)	Returns the display strings for an interface for the ActiveX control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	GoBack (see page 19236)	This is GoBack, a member of class TWebBrowser_V1.
	GoForward (see page 19237)	This is GoForward, a member of class TWebBrowser_V1.
	GoHome (see page 19237)	This is GoHome, a member of class TWebBrowser_V1.
	GoSearch (see page 19237)	This is GoSearch, a member of class TWebBrowser_V1.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	IsCustomProperty (see page 18063)	Indicates whether an interface for the ActiveX control is a custom property.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsPropPageProperty (see page 18064)	Indicates whether an interface for the OCX control is a property page property.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	Navigate (see page 19237)	This is the overview for the Navigate method overload.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 19239)	Repaints the control on the screen.
	Refresh2 (see page 19239)	This is the overview for the Refresh2 method overload.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
V	SetBounds (see page 18071)	Sets the boundaries of the ActiveX control.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetPropDisplayString (see page 18075)	Sets the display string for an interface for the ActiveX control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	ShowAboutBox (see page 18079)	Notifies the ActiveX control to display its About box.
	Stop (see page 19240)	This is Stop, a member of class TWebBrowser_V1.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
R	Application (see page 19245)	This is Application, a member of class TWebBrowser_V1.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.

	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Busy (see page 19245)	This is Busy, a member of class TWebBrowser_V1.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	Container (see page 19246)	This is Container, a member of class TWebBrowser_V1.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	ControlInterface (see page 19246)	This is ControlInterface, a member of class TWebBrowser_V1.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
R	DefaultDispatch (see page 18086)	The default dispatch interface.
R	DefaultInterface (see page 19246)	This is DefaultInterface, a member of class TWebBrowser_V1.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
R	Document (see page 19246)	This is Document, a member of class TWebBrowser_V1.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
R	LocationName (see page 19248)	This is LocationName, a member of class TWebBrowser_V1.
R	LocationURL (see page 19249)	This is LocationURL, a member of class TWebBrowser_V1.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	OleObject (see page 18086)	The ActiveX control associated with the TOleControl (see page 18030) object.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
R	Parent (see page 19249)	Indicates the parent of the control.

	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	PerPropBrowsing (see page 18086)	Provides access to information about the property page properties of the OleControl.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
R	TopLevelContainer (see page 19258)	This is TopLevelContainer, a member of class TWebBrowser_V1.
R	type_ (see page 19258)	This is type_, a member of class TWebBrowser_V1.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnBeforeNavigate (see page 19261)	This is OnBeforeNavigate, a member of class TWebBrowser_V1.
	OnCommandStateChange (see page 19261)	This is OnCommandStateChange, a member of class TWebBrowser_V1.
	OnDownloadBegin (see page 19261)	This is OnDownloadBegin, a member of class TWebBrowser_V1.
	OnDownloadComplete (see page 19261)	This is OnDownloadComplete, a member of class TWebBrowser_V1.
	OnDragDrop (see page 19261)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 19264)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 19266)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 19267)	Occurs when a control receives the input focus.
	OnExit (see page 19270)	Occurs when the input focus shifts away from one control to another.
	OnFrameBeforeNavigate (see page 19271)	This is OnFrameBeforeNavigate, a member of class TWebBrowser_V1.
	OnFrameNavigateComplete (see page 19271)	This is OnFrameNavigateComplete, a member of class TWebBrowser_V1.
	OnFrameNewWindow (see page 19272)	This is OnFrameNewWindow, a member of class TWebBrowser_V1.
	OnNavigateComplete (see page 19272)	This is OnNavigateComplete, a member of class TWebBrowser_V1.
	OnNewWindow (see page 19272)	This is OnNewWindow, a member of class TWebBrowser_V1.
	OnProgressChange (see page 19272)	This is OnProgressChange, a member of class TWebBrowser_V1.
	OnPropertyChange (see page 19272)	This is OnPropertyChange, a member of class TWebBrowser_V1.
	OnQuit (see page 19273)	This is OnQuit, a member of class TWebBrowser_V1.
	OnStartDrag (see page 19273)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnStatusTextChange (see page 19274)	This is OnStatusTextChange, a member of class TWebBrowser_V1.
	OnTitleChange (see page 19274)	This is OnTitleChange, a member of class TWebBrowser_V1.
	OnWindowActivate (see page 19274)	This is OnWindowActivate, a member of class TWebBrowser_V1.
	OnWindowMove (see page 19274)	This is OnWindowMove, a member of class TWebBrowser_V1.
	OnWindowResize (see page 19274)	This is OnWindowResize, a member of class TWebBrowser_V1.

Published Properties

	Name	Description
	Align (see page 19241)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 19245)	Specifies how the control is anchored to its parent.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 19246)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 19247)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.

	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentShowHint (see page 19250)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 19251)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 19256)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 19257)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 19258)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 19258)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

132.1.13.2 TWebBrowser_V1 Methods

The methods of the TWebBrowser_V1 class are listed here.

Protected Methods

	Name	Description
	CreateControl (see page 19235)	This is CreateControl, a member of class TWebBrowser_V1.
	Get_Application (see page 19236)	This is Get_Application, a member of class TWebBrowser_V1.
	Get_Container (see page 19236)	This is Get_Container, a member of class TWebBrowser_V1.
	Get_Document (see page 19236)	This is Get_Document, a member of class TWebBrowser_V1.
	Get_Parent (see page 19236)	This is Get_Parent, a member of class TWebBrowser_V1.
V	InitControlData (see page 19237)	This is InitControlData, a member of class TWebBrowser_V1.

Public Methods

	Name	Description
	GoBack (see page 19236)	This is GoBack, a member of class TWebBrowser_V1.
	GoForward (see page 19237)	This is GoForward, a member of class TWebBrowser_V1.
	GoHome (see page 19237)	This is GoHome, a member of class TWebBrowser_V1.
	GoSearch (see page 19237)	This is GoSearch, a member of class TWebBrowser_V1.
	Navigate (see page 19237)	This is the overview for the Navigate method overload.
	Refresh (see page 19239)	Repaints the control on the screen.
	Refresh2 (see page 19239)	This is the overview for the Refresh2 method overload.
	Stop (see page 19240)	This is Stop, a member of class TWebBrowser_V1.

132.1.13.2.1 TWebBrowser_V1.CreateControl Method

Pascal

```
procedure CreateControl;
```

C++

```
__fastcall CreateControl();
```

Description

This is CreateControl, a member of class TWebBrowser_V1.

132.1.13.2.2 TWebBrowser_V1.Get_Application Method

Pascal

```
function Get_Application: IDispatch;
```

C++

```
__fastcall IDispatch Get_Application();
```

Description

This is Get_Application, a member of class TWebBrowser_V1.

132.1.13.2.3 TWebBrowser_V1.Get_Container Method

Pascal

```
function Get_Container: IDispatch;
```

C++

```
__fastcall IDispatch Get_Container();
```

Description

This is Get_Container, a member of class TWebBrowser_V1.

132.1.13.2.4 TWebBrowser_V1.Get_Document Method

Pascal

```
function Get_Document: IDispatch;
```

C++

```
__fastcall IDispatch Get_Document();
```

Description

This is Get_Document, a member of class TWebBrowser_V1.

132.1.13.2.5 TWebBrowser_V1.Get_Parent Method

Pascal

```
function Get_Parent: IDispatch;
```

C++

```
__fastcall IDispatch Get_Parent();
```

Description

This is Get_Parent, a member of class TWebBrowser_V1.

132.1.13.2.6 TWebBrowser_V1.GoBack Method

Pascal

```
procedure GoBack;
```

C++

```
__fastcall GoBack();
```

Description

This is GoBack, a member of class TWebBrowser_V1.

132.1.13.2.7 TWebBrowser_V1.GoForward Method

Pascal

```
procedure GoForward;
```

C++

```
__fastcall GoForward();
```

Description

This is GoForward, a member of class TWebBrowser_V1.

132.1.13.2.8 TWebBrowser_V1.GoHome Method

Pascal

```
procedure GoHome;
```

C++

```
__fastcall GoHome();
```

Description

This is GoHome, a member of class TWebBrowser_V1.

132.1.13.2.9 TWebBrowser_V1.GoSearch Method

Pascal

```
procedure GoSearch;
```

C++

```
__fastcall GoSearch();
```

Description

This is GoSearch, a member of class TWebBrowser_V1.

132.1.13.2.10 TWebBrowser_V1.InitControlData Method

Pascal

```
procedure InitControlData; override;
```

C++

```
virtual __fastcall InitControlData();
```

Description

This is InitControlData, a member of class TWebBrowser_V1.

132.1.13.2.11 Navigate Method

This is the overview for the Navigate method overload.

Overload List

	Name	Description
	TWebBrowser_V1.Navigate (WideString) (see page 19238)	This is Navigate, a member of class TWebBrowser_V1.
	TWebBrowser_V1.Navigate (WideString, OleVariant) (see page 19238)	This is Navigate, a member of class TWebBrowser_V1.
	TWebBrowser_V1.Navigate (WideString, OleVariant, OleVariant) (see page 19238)	This is Navigate, a member of class TWebBrowser_V1.
	TWebBrowser_V1.Navigate (WideString, OleVariant, OleVariant, OleVariant) (see page 19239)	This is Navigate, a member of class TWebBrowser_V1.
	TWebBrowser_V1.Navigate (WideString, OleVariant, OleVariant, OleVariant, OleVariant) (see page 19239)	This is Navigate, a member of class TWebBrowser_V1.

132.1.13.2.11.1 TWebBrowser_V1.Navigate Method (WideString)

Pascal

```
procedure Navigate(const URL: WideString); overload;
```

C++

```
__fastcall Navigate(const BSTR URL);
```

Description

This is Navigate, a member of class TWebBrowser_V1.

132.1.13.2.11.2 TWebBrowser_V1.Navigate Method (WideString, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags);
```

Description

This is Navigate, a member of class TWebBrowser_V1.

132.1.13.2.11.3 TWebBrowser_V1.Navigate Method (WideString, OleVariant, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName);
```

Description

This is Navigate, a member of class TWebBrowser_V1.

132.1.13.2.11.4 TWebBrowser_V1.Navigate Method (WideString, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData);
```

Description

This is Navigate, a member of class TWebBrowser_V1.

132.1.13.2.11.5 TWebBrowser_V1.Navigate Method (WideString, OleVariant, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate, a member of class TWebBrowser_V1.

132.1.13.2.12 TWebBrowser_V1.Refresh Method

Repaints the control on the screen.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh method to repaint the control immediately. Refresh calls the Repaint method. Use the Refresh and Repaint methods interchangeably.

See Also

Invalidate (see page 5770)

Parent (see page 5832)

Repaint (see page 5779)

Update (see page 5789)

132.1.13.2.13 Refresh2 Method

This is the overview for the Refresh2 method overload.

Overload List

	Name	Description
	TWebBrowser_V1.Refresh2 () (see page 19240)	This is Refresh2, a member of class TWebBrowser_V1.
	TWebBrowser_V1.Refresh2 (OleVariant) (see page 19240)	This is Refresh2, a member of class TWebBrowser_V1.

132.1.13.2.13.1 TWebBrowser_V1.Refresh2 Method ()

Pascal

```
procedure Refresh2; overload;
```

C++

```
__fastcall Refresh2();
```

Description

This is Refresh2, a member of class TWebBrowser_V1.

132.1.13.2.13.2 TWebBrowser_V1.Refresh2 Method (OleVariant)

Pascal

```
procedure Refresh2(var Level: OleVariant); overload;
```

C++

```
__fastcall Refresh2(OleVariant Level);
```

Description

This is Refresh2, a member of class TWebBrowser_V1.

132.1.13.2.14 TWebBrowser_V1.Stop Method

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

This is Stop, a member of class TWebBrowser_V1.

132.1.13.3 TWebBrowser_V1 Properties

The properties of the TWebBrowser_V1 class are listed here.

Public Properties

	Name	Description
R	Application (see page 19245)	This is Application, a member of class TWebBrowser_V1.
R	Busy (see page 19245)	This is Busy, a member of class TWebBrowser_V1.
R	Container (see page 19246)	This is Container, a member of class TWebBrowser_V1.
R	ControlInterface (see page 19246)	This is ControlInterface, a member of class TWebBrowser_V1.

R	DefaultInterface (see page 19246)	This is DefaultInterface, a member of class TWebBrowser_V1.
R	Document (see page 19246)	This is Document, a member of class TWebBrowser_V1.
R	LocationName (see page 19248)	This is LocationName, a member of class TWebBrowser_V1.
R	LocationURL (see page 19249)	This is LocationURL, a member of class TWebBrowser_V1.
R	Parent (see page 19249)	Indicates the parent of the control.
R	TopLevelContainer (see page 19258)	This is TopLevelContainer, a member of class TWebBrowser_V1.
R	type_ (see page 19258)	This is type_, a member of class TWebBrowser_V1.

Published Properties

	Name	Description
	Align (see page 19241)	Determines how the control aligns within its container (parent control).
	Anchors (see page 19245)	Specifies how the control is anchored to its parent.
	DragCursor (see page 19246)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 19247)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	ParentShowHint (see page 19250)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 19251)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 19256)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 19257)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 19258)	Determines if the user can tab to a control.
	Visible (see page 19258)	Determines whether the component appears onscreen.

132.1.13.3.1 TWebBrowser_V1.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

132.1.13.3.2 TWebBrowser_V1.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

132.1.13.3.3 TWebBrowser_V1.Application Property

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

This is Application, a member of class TWebBrowser_V1.

132.1.13.3.4 TWebBrowser_V1.Busy Property

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

This is Busy, a member of class TWebBrowser_V1.

132.1.13.3.5 TWebBrowser_V1.Container Property

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

This is Container, a member of class TWebBrowser_V1.

132.1.13.3.6 TWebBrowser_V1.ControlInterface Property

Pascal

```
property ControlInterface: IWebBrowser;
```

C++

```
__property IWebBrowser ControlInterface;
```

Description

This is ControlInterface, a member of class TWebBrowser_V1.

132.1.13.3.7 TWebBrowser_V1.DefaultInterface Property

Pascal

```
property DefaultInterface: IWebBrowser;
```

C++

```
__property IWebBrowser DefaultInterface;
```

Description

This is DefaultInterface, a member of class TWebBrowser_V1.

132.1.13.3.8 TWebBrowser_V1.Document Property

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

This is Document, a member of class TWebBrowser_V1.

132.1.13.3.9 TWebBrowser_V1.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

132.1.13.3.10 TWebBrowser_V1.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
```

```
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

132.1.13.3.11 TWebBrowser_V1.LocationName Property

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

This is LocationName, a member of class TWebBrowser_V1.

132.1.13.3.12 TWebBrowser_V1.LocationURL Property

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

This is LocationURL, a member of class TWebBrowser_V1.

132.1.13.3.13 TWebBrowser_V1.Parent Property

Indicates the parent of the control.

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

Use the Parent property to get or set the parent of this control. The parent of a control is the control that contains the control. For example, if an application includes three radio buttons in a group box, the group box is the parent of the three radio buttons, and the radio buttons are the child controls of the group box.

To serve as a parent, a control must be an instance of a descendant of TWinControl.

When creating a new control at runtime, assign a Parent property value for the new control. Usually, this is a form, panel, group box, or some control that is designed to contain another. Changing the parent of a control moves the control onscreen so that it is displayed within the new parent. When the parent control moves, the child moves with the parent.

Some controls (such as ActiveX controls) are contained in native windows rather than in a parent VCL control. For these controls, the value of Parent is nil (Delphi) or NULL (C++) and the ParentWindow property specifies the window.

Note: The Parent property declared in TControl is similar to the Owner property declared in TComponent, in that the Parent of a control frees the control just as the Owner of a component frees that Component. However, the Parent of a control is always a windowed control that visually contains the control, and is responsible for writing the control to a stream when the form is saved. The Owner of a component is the component that was passed as a parameter in the constructor and, if assigned, initiates the process of saving all objects (including the control and its parent) when the form is saved.

C++ Examples:

```
/*
This example uses a button placed next to a group box. When the
user clicks the button, the group box becomes the parent of the
button, so the button moves inside the group box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (Button1->Parent != GroupBox1) // second click?
```

```
    {
      RemoveControl(Button1);
      GroupBox1->InsertControl(Button1);
    }
  }

  /*
  Note that it was necessary to remove the button from the Controls
  property of the form before the button actually moves into the
  group box. This code accomplishes the same thing:
  */

  void __fastcall TForm1::Button2Click(TObject *Sender)
  {
    Button2->Parent = GroupBox1;
  }
```

Delphi Examples:

```
  {
  This example uses a button placed next to a group box. When the
  user clicks the button, the group box becomes the parent of the
  button, so the button moves inside the group box.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    if (Button1.Parent <> GroupBox1) then // second click?
    begin
      RemoveControl(Button1);
      GroupBox1.InsertControl(Button1);
    end;
  end;

  {
  Note that it was necessary to remove the button from the Controls
  property of the form before the button actually moves into the
  group box. This code accomplishes the same thing:
  }

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    Button2.Parent := GroupBox1;
  end;
```

See Also

Owner (see page 2807)

Controls (see page 6197)

HostDockSite (see page 5829)

ParentWindow (see page 6210)

132.1.13.3.14 TWebBrowser_V1.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
  property ParentShowHint: Boolean;
```

C++

```
  __property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

132.1.13.3.15 TWebBrowser_V1.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
```

```
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
```

```
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
```

```
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

132.1.13.3.16 TWebBrowser_V1.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

132.1.13.3.17 TWebBrowser_V1.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

132.1.13.3.18 TWebBrowser_V1.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

132.1.13.3.19 TWebBrowser_V1.TopLevelContainer Property

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

This is TopLevelContainer, a member of class TWebBrowser_V1.

132.1.13.3.20 TWebBrowser_V1.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class TWebBrowser_V1.

132.1.13.3.21 TWebBrowser_V1.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

132.1.13.4 TWebBrowser_V1 Events

The events of the TWebBrowser_V1 class are listed here.

Published Events

	Name	Description
	OnBeforeNavigate (see page 19261)	This is OnBeforeNavigate, a member of class TWebBrowser_V1.
	OnCommandStateChange (see page 19261)	This is OnCommandStateChange, a member of class TWebBrowser_V1.
	OnDownloadBegin (see page 19261)	This is OnDownloadBegin, a member of class TWebBrowser_V1.
	OnDownloadComplete (see page 19261)	This is OnDownloadComplete, a member of class TWebBrowser_V1.
	OnDragDrop (see page 19261)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 19264)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 19266)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 19267)	Occurs when a control receives the input focus.
	OnExit (see page 19270)	Occurs when the input focus shifts away from one control to another.
	OnFrameBeforeNavigate (see page 19271)	This is OnFrameBeforeNavigate, a member of class TWebBrowser_V1.
	OnFrameNavigateComplete (see page 19271)	This is OnFrameNavigateComplete, a member of class TWebBrowser_V1.
	OnFrameNewWindow (see page 19272)	This is OnFrameNewWindow, a member of class TWebBrowser_V1.
	OnNavigateComplete (see page 19272)	This is OnNavigateComplete, a member of class TWebBrowser_V1.
	OnNewWindow (see page 19272)	This is OnNewWindow, a member of class TWebBrowser_V1.
	OnProgressChange (see page 19272)	This is OnProgressChange, a member of class TWebBrowser_V1.
	OnPropertyChange (see page 19272)	This is OnPropertyChange, a member of class TWebBrowser_V1.
	OnQuit (see page 19273)	This is OnQuit, a member of class TWebBrowser_V1.
	OnStartDrag (see page 19273)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnStatusTextChange (see page 19274)	This is OnStatusTextChange, a member of class TWebBrowser_V1.
	OnTitleChange (see page 19274)	This is OnTitleChange, a member of class TWebBrowser_V1.
	OnWindowActivate (see page 19274)	This is OnWindowActivate, a member of class TWebBrowser_V1.
	OnWindowMove (see page 19274)	This is OnWindowMove, a member of class TWebBrowser_V1.

	OnWindowResize (see page 19274)	This is OnWindowResize, a member of class TWebBrowser_V1.

132.1.13.4.1 TWebBrowser_V1.OnBeforeNavigate Event

Pascal

```
property OnBeforeNavigate: TWebBrowser_V1BeforeNavigate;
```

C++

```
__property TWebBrowser_V1BeforeNavigate OnBeforeNavigate;
```

Description

This is OnBeforeNavigate, a member of class TWebBrowser_V1.

132.1.13.4.2 TWebBrowser_V1.OnCommandStateChange Event

Pascal

```
property OnCommandStateChange: TWebBrowser_V1CommandStateChange;
```

C++

```
__property TWebBrowser_V1CommandStateChange OnCommandStateChange;
```

Description

This is OnCommandStateChange, a member of class TWebBrowser_V1.

132.1.13.4.3 TWebBrowser_V1.OnDownloadBegin Event

Pascal

```
property OnDownloadBegin: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownloadBegin;
```

Description

This is OnDownloadBegin, a member of class TWebBrowser_V1.

132.1.13.4.4 TWebBrowser_V1.OnDownloadComplete Event

Pascal

```
property OnDownloadComplete: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownloadComplete;
```

Description

This is OnDownloadComplete, a member of class TWebBrowser_V1.

132.1.13.4.5 TWebBrowser_V1.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

132.1.13.4.6 TWebBrowser_V1.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

132.1.13.4.7 TWebBrowser_V1.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
```

```
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

132.1.13.4.8 TWebBrowser_V1.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter

event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

132.1.13.4.9 TWebBrowser_V1.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
```

```
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

132.1.13.4.10 TWebBrowser_V1.OnFrameBeforeNavigate Event

Pascal

```
property OnFrameBeforeNavigate: TWebBrowser_V1FrameBeforeNavigate;
```

C++

```
__property TWebBrowser_V1FrameBeforeNavigate OnFrameBeforeNavigate;
```

Description

This is OnFrameBeforeNavigate, a member of class TWebBrowser_V1.

132.1.13.4.11 TWebBrowser_V1.OnFrameNavigateComplete Event

Pascal

```
property OnFrameNavigateComplete: TWebBrowser_V1FrameNavigateComplete;
```

C++

```
__property TWebBrowser_V1FrameNavigateComplete OnFrameNavigateComplete;
```

Description

This is OnFrameNavigateComplete, a member of class TWebBrowser_V1.

132.1.13.4.12 TWebBrowser_V1.OnFrameNewWindow Event

Pascal

```
property OnFrameNewWindow: TWebBrowser_V1FrameNewWindow;
```

C++

```
__property TWebBrowser_V1FrameNewWindow OnFrameNewWindow;
```

Description

This is OnFrameNewWindow, a member of class TWebBrowser_V1.

132.1.13.4.13 TWebBrowser_V1.OnNavigateComplete Event

Pascal

```
property OnNavigateComplete: TWebBrowser_V1NavigateComplete;
```

C++

```
__property TWebBrowser_V1NavigateComplete OnNavigateComplete;
```

Description

This is OnNavigateComplete, a member of class TWebBrowser_V1.

132.1.13.4.14 TWebBrowser_V1.OnNewWindow Event

Pascal

```
property OnNewWindow: TWebBrowser_V1NewWindow;
```

C++

```
__property TWebBrowser_V1NewWindow OnNewWindow;
```

Description

This is OnNewWindow, a member of class TWebBrowser_V1.

132.1.13.4.15 TWebBrowser_V1.OnProgressChange Event

Pascal

```
property OnProgressChange: TWebBrowser_V1ProgressChange;
```

C++

```
__property TWebBrowser_V1ProgressChange OnProgressChange;
```

Description

This is OnProgressChange, a member of class TWebBrowser_V1.

132.1.13.4.16 TWebBrowser_V1.OnPropertyChange Event

Pascal

```
property OnPropertyChange: TWebBrowser_V1PropertyChange;
```

C++

```
__property TWebBrowser_V1PropertyChange OnPropertyChange;
```

Description

This is OnPropertyChange, a member of class TWebBrowser_V1.

132.1.13.4.17 TWebBrowser_V1.OnQuit Event

Pascal

```
property OnQuit: TWebBrowser_V1Quit;
```

C++

```
__property TWebBrowser_V1Quit OnQuit;
```

Description

This is OnQuit, a member of class TWebBrowser_V1.

132.1.13.4.18 TWebBrowser_V1.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

132.1.13.4.19 TWebBrowser_V1.OnStatusTextChange Event

Pascal

```
property OnStatusTextChange: TWebBrowser_V1StatusTextChange;
```

C++

```
__property TWebBrowser_V1StatusTextChange OnStatusTextChange;
```

Description

This is OnStatusTextChange, a member of class TWebBrowser_V1.

132.1.13.4.20 TWebBrowser_V1.OnTitleChange Event

Pascal

```
property OnTitleChange: TWebBrowser_V1TitleChange;
```

C++

```
__property TWebBrowser_V1TitleChange OnTitleChange;
```

Description

This is OnTitleChange, a member of class TWebBrowser_V1.

132.1.13.4.21 TWebBrowser_V1.OnWindowActivate Event

Pascal

```
property OnWindowActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnWindowActivate;
```

Description

This is OnWindowActivate, a member of class TWebBrowser_V1.

132.1.13.4.22 TWebBrowser_V1.OnWindowMove Event

Pascal

```
property OnWindowMove: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnWindowMove;
```

Description

This is OnWindowMove, a member of class TWebBrowser_V1.

132.1.13.4.23 TWebBrowser_V1.OnWindowResize Event

Pascal

```
property OnWindowResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnWindowResize;
```

Description

This is OnWindowResize, a member of class TWebBrowser_V1.

132.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	_SearchAssistantEvents (see page 19276)	This is class SHDocVw._SearchAssistantEvents.
	DShellNameSpaceEvents (see page 19278)	This is class SHDocVw.DShellNameSpaceEvents.
	DShellWindowsEvents (see page 19279)	This is class SHDocVw.DShellWindowsEvents.
	DWebBrowserEvents (see page 19281)	This is class SHDocVw.DWebBrowserEvents.
	DWebBrowserEvents2 (see page 19286)	This is class SHDocVw.DWebBrowserEvents2.
	IScriptErrorList (see page 19295)	This is class SHDocVw.IScriptErrorList.
	IScriptErrorListDisp (see page 19299)	This is class SHDocVw.IScriptErrorListDisp.
	ISearch (see page 19303)	This is class SHDocVw.ISearch.
	ISearchAssistantOC (see page 19306)	This is class SHDocVw.ISearchAssistantOC.
	ISearchAssistantOC2 (see page 19316)	This is class SHDocVw.ISearchAssistantOC2.
	ISearchAssistantOC2Disp (see page 19318)	This is class SHDocVw.ISearchAssistantOC2Disp.
	ISearchAssistantOC3 (see page 19326)	This is class SHDocVw.ISearchAssistantOC3.
	ISearchAssistantOC3Disp (see page 19329)	This is class SHDocVw.ISearchAssistantOC3Disp.
	ISearchAssistantOCDisp (see page 19337)	This is class SHDocVw.ISearchAssistantOCDisp.
	ISearchDisp (see page 19344)	This is class SHDocVw.ISearchDisp.
	ISearches (see page 19346)	This is class SHDocVw.ISearches.
	ISearchesDisp (see page 19348)	This is class SHDocVw.ISearchesDisp.
	IShellFavoritesNameSpace (see page 19350)	This is class SHDocVw.IShellFavoritesNameSpace.
	IShellFavoritesNameSpaceDisp (see page 19355)	This is class SHDocVw.IShellFavoritesNameSpaceDisp.
	IShellNameSpace (see page 19359)	This is class SHDocVw.IShellNameSpace.
	IShellNameSpaceDisp (see page 19368)	This is class SHDocVw.IShellNameSpaceDisp.
	IShellUIHelper (see page 19376)	This is class SHDocVw.IShellUIHelper.
	IShellUIHelperDisp (see page 19380)	This is class SHDocVw.IShellUIHelperDisp.
	IShellWindows (see page 19384)	This is class SHDocVw.IShellWindows.
	IShellWindowsDisp (see page 19388)	This is class SHDocVw.IShellWindowsDisp.
	IWebBrowser (see page 19392)	This is class SHDocVw.IWebBrowser.
	IWebBrowser2 (see page 19403)	This is class SHDocVw.IWebBrowser2.
	IWebBrowser2Disp (see page 19412)	This is class SHDocVw.IWebBrowser2Disp.
	IWebBrowserApp (see page 19425)	This is class SHDocVw.IWebBrowserApp.
	IWebBrowserAppDisp (see page 19434)	This is class SHDocVw.IWebBrowserAppDisp.
	IWebBrowserDisp (see page 19444)	This is class SHDocVw.IWebBrowserDisp.

132.2.1 _SearchAssistantEvents Interface

Class Hierarchy

SHDocVw._SearchAssistantEvents

Pascal

```
_SearchAssistantEvents = dispinterface;
```

C++

```
class _SearchAssistantEvents;
```

File

SHDocVw

Description

This is class SHDocVw._SearchAssistantEvents.

132.2.1.1 _SearchAssistantEvents Members

The following tables list the members exposed by _SearchAssistantEvents.

Public Methods

	Name	Description
	OnNewSearch (see page 19277)	This is OnNewSearch, a member of class _SearchAssistantEvents.
	OnNextMenuSelect (see page 19277)	This is OnNextMenuSelect, a member of class _SearchAssistantEvents.

132.2.1.2 _SearchAssistantEvents Methods

The methods of the _SearchAssistantEvents class are listed here.

Public Methods

	Name	Description
	OnNewSearch (see page 19277)	This is OnNewSearch, a member of class _SearchAssistantEvents.
	OnNextMenuSelect (see page 19277)	This is OnNextMenuSelect, a member of class _SearchAssistantEvents.

132.2.1.2.1 _SearchAssistantEvents.OnNewSearch Method

Pascal

```
procedure OnNewSearch;
```

C++

```
__fastcall OnNewSearch();
```

Description

This is OnNewSearch, a member of class _SearchAssistantEvents.

132.2.1.2.2 _SearchAssistantEvents.OnNextMenuSelect Method

Pascal

```
procedure OnNextMenuSelect(idItem: Integer);
```

C++

```
__fastcall OnNextMenuSelect(int idItem);
```

Description

This is OnNextMenuSelect, a member of class _SearchAssistantEvents.

132.2.2 DShellNameSpaceEvents Interface

Class Hierarchy

SHDocVw.DShellNameSpaceEvents

Pascal

```
DShellNameSpaceEvents = dispinterface;
```

C++

```
class DShellNameSpaceEvents;
```

File

SHDocVw

Description

This is class SHDocVw.DShellNameSpaceEvents.

132.2.2.1 DShellNameSpaceEvents Members

The following tables list the members exposed by DShellNameSpaceEvents.

Public Methods

	Name	Description
	DoubleClick (see page 19278)	This is DoubleClick, a member of class DShellNameSpaceEvents.
	FavoritesSelectionChange (see page 19279)	This is FavoritesSelectionChange, a member of class DShellNameSpaceEvents.
	Initialized (see page 19279)	This is Initialized, a member of class DShellNameSpaceEvents.
	SelectionChange (see page 19279)	This is SelectionChange, a member of class DShellNameSpaceEvents.

132.2.2.2 DShellNameSpaceEvents Methods

The methods of the DShellNameSpaceEvents class are listed here.

Public Methods

	Name	Description
	DoubleClick (see page 19278)	This is DoubleClick, a member of class DShellNameSpaceEvents.
	FavoritesSelectionChange (see page 19279)	This is FavoritesSelectionChange, a member of class DShellNameSpaceEvents.
	Initialized (see page 19279)	This is Initialized, a member of class DShellNameSpaceEvents.
	SelectionChange (see page 19279)	This is SelectionChange, a member of class DShellNameSpaceEvents.

132.2.2.2.1 DShellNameSpaceEvents.DoubleClick Method

Pascal

```
procedure DoubleClick;
```

C++

```
__fastcall DoubleClick();
```

Description

This is DoubleClick, a member of class DShellNameSpaceEvents.

132.2.2.2.2 DShellNameSpaceEvents.FavoritesSelectionChange Method

Pascal

```
procedure FavoritesSelectionChange(cItems: Integer; hItem: Integer; const strName: WideString;
const strUrl: WideString; cVisits: Integer; const strDate: WideString; fAvailableOffline:
Integer);
```

C++

```
__fastcall FavoritesSelectionChange(int cItems, int hItem, const BSTR strName, const BSTR
strUrl, int cVisits, const BSTR strDate, int fAvailableOffline);
```

Description

This is FavoritesSelectionChange, a member of class DShellNameSpaceEvents.

132.2.2.2.3 DShellNameSpaceEvents.Initialized Method

Pascal

```
procedure Initialized;
```

C++

```
__fastcall Initialized();
```

Description

This is Initialized, a member of class DShellNameSpaceEvents.

132.2.2.2.4 DShellNameSpaceEvents.SelectionChange Method

Pascal

```
procedure SelectionChange;
```

C++

```
__fastcall SelectionChange();
```

Description

This is SelectionChange, a member of class DShellNameSpaceEvents.

132.2.3 DShellWindowsEvents Interface

Class Hierarchy

SHDocVw.DShellWindowsEvents

Pascal

```
DShellWindowsEvents = dispinterface;
```

C++

```
class DShellWindowsEvents;
```

File

SHDocVw

Description

This is class SHDocVw.DShellWindowsEvents.

132.2.3.1 DShellWindowsEvents Members

The following tables list the members exposed by DShellWindowsEvents.

Public Methods

	Name	Description
	WindowRegistered (see page 19280)	This is WindowRegistered, a member of class DShellWindowsEvents.
	WindowRevoked (see page 19280)	This is WindowRevoked, a member of class DShellWindowsEvents.

132.2.3.2 DShellWindowsEvents Methods

The methods of the DShellWindowsEvents class are listed here.

Public Methods

	Name	Description
	WindowRegistered (see page 19280)	This is WindowRegistered, a member of class DShellWindowsEvents.
	WindowRevoked (see page 19280)	This is WindowRevoked, a member of class DShellWindowsEvents.

132.2.3.2.1 DShellWindowsEvents.WindowRegistered Method

Pascal

```
procedure WindowRegistered(lCookie: Integer);
```

C++

```
__fastcall WindowRegistered(int lCookie);
```

Description

This is WindowRegistered, a member of class DShellWindowsEvents.

132.2.3.2.2 DShellWindowsEvents.WindowRevoked Method

Pascal

```
procedure WindowRevoked(lCookie: Integer);
```

C++

```
__fastcall WindowRevoked(int lCookie);
```

Description

This is WindowRevoked, a member of class DShellWindowsEvents.

132.2.4 DWebBrowserEvents Interface

Class Hierarchy

SHDocVw.DWebBrowserEvents

Pascal

```
DWebBrowserEvents = dispinterface;
```

C++

```
class DWebBrowserEvents;
```

File

SHDocVw

Description

This is class SHDocVw.DWebBrowserEvents.

132.2.4.1 DWebBrowserEvents Members

The following tables list the members exposed by DWebBrowserEvents.

Public Methods

	Name	Description
	BeforeNavigate (see page 19282)	This is BeforeNavigate, a member of class DWebBrowserEvents.
	CommandStateChange (see page 19282)	This is CommandStateChange, a member of class DWebBrowserEvents.
	DownloadBegin (see page 19282)	This is DownloadBegin, a member of class DWebBrowserEvents.
	DownloadComplete (see page 19283)	This is DownloadComplete, a member of class DWebBrowserEvents.
	FrameBeforeNavigate (see page 19283)	This is FrameBeforeNavigate, a member of class DWebBrowserEvents.
	FrameNavigateComplete (see page 19283)	This is FrameNavigateComplete, a member of class DWebBrowserEvents.
	FrameNewWindow (see page 19283)	This is FrameNewWindow, a member of class DWebBrowserEvents.
	NavigateComplete (see page 19284)	This is NavigateComplete, a member of class DWebBrowserEvents.
	NewWindow (see page 19284)	This is NewWindow, a member of class DWebBrowserEvents.
	ProgressChange (see page 19284)	This is ProgressChange, a member of class DWebBrowserEvents.
	PropertyChange (see page 19284)	This is PropertyChange, a member of class DWebBrowserEvents.
	Quit (see page 19284)	This is Quit, a member of class DWebBrowserEvents.
	StatusTextChange (see page 19285)	This is StatusTextChange, a member of class DWebBrowserEvents.
	TitleChange (see page 19285)	This is TitleChange, a member of class DWebBrowserEvents.
	WindowActivate (see page 19285)	This is WindowActivate, a member of class DWebBrowserEvents.
	WindowMove (see page 19285)	This is WindowMove, a member of class DWebBrowserEvents.
	WindowResize (see page 19286)	This is WindowResize, a member of class DWebBrowserEvents.

132.2.4.2 DWebBrowserEvents Methods

The methods of the DWebBrowserEvents class are listed here.

Public Methods

	Name	Description
	BeforeNavigate (see page 19282)	This is BeforeNavigate, a member of class DWebBrowserEvents.

	CommandStateChange (see page 19282)	This is CommandStateChange, a member of class DWebBrowserEvents.
	DownloadBegin (see page 19282)	This is DownloadBegin, a member of class DWebBrowserEvents.
	DownloadComplete (see page 19283)	This is DownloadComplete, a member of class DWebBrowserEvents.
	FrameBeforeNavigate (see page 19283)	This is FrameBeforeNavigate, a member of class DWebBrowserEvents.
	FrameNavigateComplete (see page 19283)	This is FrameNavigateComplete, a member of class DWebBrowserEvents.
	FrameNewWindow (see page 19283)	This is FrameNewWindow, a member of class DWebBrowserEvents.
	NavigateComplete (see page 19284)	This is NavigateComplete, a member of class DWebBrowserEvents.
	NewWindow (see page 19284)	This is NewWindow, a member of class DWebBrowserEvents.
	ProgressChange (see page 19284)	This is ProgressChange, a member of class DWebBrowserEvents.
	PropertyChange (see page 19284)	This is PropertyChange, a member of class DWebBrowserEvents.
	Quit (see page 19284)	This is Quit, a member of class DWebBrowserEvents.
	StatusTextChange (see page 19285)	This is StatusTextChange, a member of class DWebBrowserEvents.
	TitleChange (see page 19285)	This is TitleChange, a member of class DWebBrowserEvents.
	WindowActivate (see page 19285)	This is WindowActivate, a member of class DWebBrowserEvents.
	WindowMove (see page 19285)	This is WindowMove, a member of class DWebBrowserEvents.
	WindowResize (see page 19286)	This is WindowResize, a member of class DWebBrowserEvents.

132.2.4.2.1 DWebBrowserEvents.BeforeNavigate Method

Pascal

```
procedure BeforeNavigate(const URL: WideString; Flags: Integer; const TargetFrameName:
WideString; var PostData: OleVariant; const Headers: WideString; var Cancel: WordBool);
```

C++

```
__fastcall BeforeNavigate(const BSTR URL, int Flags, const BSTR TargetFrameName, OleVariant
PostData, const BSTR Headers, WordBool Cancel);
```

Description

This is BeforeNavigate, a member of class DWebBrowserEvents.

132.2.4.2.2 DWebBrowserEvents.CommandStateChange Method

Pascal

```
procedure CommandStateChange(Command: Integer; Enable: WordBool);
```

C++

```
__fastcall CommandStateChange(int Command, WordBool Enable);
```

Description

This is CommandStateChange, a member of class DWebBrowserEvents.

132.2.4.2.3 DWebBrowserEvents.DownloadBegin Method

Pascal

```
procedure DownloadBegin;
```

C++

```
__fastcall DownloadBegin();
```

Description

This is DownloadBegin, a member of class DWebBrowserEvents.

132.2.4.2.4 DWebBrowserEvents.DownloadComplete Method

Pascal

```
procedure DownloadComplete;
```

C++

```
__fastcall DownloadComplete();
```

Description

This is DownloadComplete, a member of class DWebBrowserEvents.

132.2.4.2.5 DWebBrowserEvents.FrameBeforeNavigate Method

Pascal

```
procedure FrameBeforeNavigate(const URL: WideString; Flags: Integer; const TargetFrameName:
WideString; var PostData: OleVariant; const Headers: WideString; var Cancel: WordBool);
```

C++

```
__fastcall FrameBeforeNavigate(const BSTR URL, int Flags, const BSTR TargetFrameName,
OleVariant PostData, const BSTR Headers, WordBool Cancel);
```

Description

This is FrameBeforeNavigate, a member of class DWebBrowserEvents.

132.2.4.2.6 DWebBrowserEvents.FrameNavigateComplete Method

Pascal

```
procedure FrameNavigateComplete(const URL: WideString);
```

C++

```
__fastcall FrameNavigateComplete(const BSTR URL);
```

Description

This is FrameNavigateComplete, a member of class DWebBrowserEvents.

132.2.4.2.7 DWebBrowserEvents.FrameNewWindow Method

Pascal

```
procedure FrameNewWindow(const URL: WideString; Flags: Integer; const TargetFrameName:
WideString; var PostData: OleVariant; const Headers: WideString; var Processed: WordBool);
```

C++

```
__fastcall FrameNewWindow(const BSTR URL, int Flags, const BSTR TargetFrameName, OleVariant
PostData, const BSTR Headers, WordBool Processed);
```

Description

This is FrameNewWindow, a member of class DWebBrowserEvents.

132.2.4.2.8 DWebBrowserEvents.NavigateComplete Method

Pascal

```
procedure NavigateComplete(const URL: WideString);
```

C++

```
__fastcall NavigateComplete(const BSTR URL);
```

Description

This is NavigateComplete, a member of class DWebBrowserEvents.

132.2.4.2.9 DWebBrowserEvents.NewWindow Method

Pascal

```
procedure NewWindow(const URL: WideString; Flags: Integer; const TargetFrameName: WideString;
var PostData: OleVariant; const Headers: WideString; var Processed: WordBool);
```

C++

```
__fastcall NewWindow(const BSTR URL, int Flags, const BSTR TargetFrameName, OleVariant
PostData, const BSTR Headers, WordBool Processed);
```

Description

This is NewWindow, a member of class DWebBrowserEvents.

132.2.4.2.10 DWebBrowserEvents.ProgressChange Method

Pascal

```
procedure ProgressChange(Progress: Integer; ProgressMax: Integer);
```

C++

```
__fastcall ProgressChange(int Progress, int ProgressMax);
```

Description

This is ProgressChange, a member of class DWebBrowserEvents.

132.2.4.2.11 DWebBrowserEvents.PropertyChange Method

Pascal

```
procedure PropertyChange(const Property_: WideString);
```

C++

```
__fastcall PropertyChange(const BSTR Property_);
```

Description

This is PropertyChange, a member of class DWebBrowserEvents.

132.2.4.2.12 DWebBrowserEvents.Quit Method

Pascal

```
procedure Quit(var Cancel: WordBool);
```

C++

```
__fastcall Quit(WordBool Cancel);
```

Description

This is Quit, a member of class DWebBrowserEvents.

132.2.4.2.13 DWebBrowserEvents.StatusTextChange Method

Pascal

```
procedure StatusTextChange(const Text: WideString);
```

C++

```
__fastcall StatusTextChange(const BSTR Text);
```

Description

This is StatusTextChange, a member of class DWebBrowserEvents.

132.2.4.2.14 DWebBrowserEvents.TitleChange Method

Pascal

```
procedure TitleChange(const Text: WideString);
```

C++

```
__fastcall TitleChange(const BSTR Text);
```

Description

This is TitleChange, a member of class DWebBrowserEvents.

132.2.4.2.15 DWebBrowserEvents.WindowActivate Method

Pascal

```
procedure WindowActivate;
```

C++

```
__fastcall WindowActivate();
```

Description

This is WindowActivate, a member of class DWebBrowserEvents.

132.2.4.2.16 DWebBrowserEvents.WindowMove Method

Pascal

```
procedure WindowMove;
```

C++

```
__fastcall WindowMove();
```

Description

This is WindowMove, a member of class DWebBrowserEvents.

132.2.4.2.17 DWebBrowserEvents.WindowResize Method

Pascal

```
procedure WindowResize;
```

C++

```
__fastcall WindowResize();
```

Description

This is WindowResize, a member of class DWebBrowserEvents.

132.2.5 DWebBrowserEvents2 Interface

Class Hierarchy

SHDocVw.DWebBrowserEvents2

Pascal

```
DWebBrowserEvents2 = dispinterface;
```

C++

```
class DWebBrowserEvents2;
```

File

SHDocVw

Description

This is class SHDocVw.DWebBrowserEvents2.

132.2.5.1 DWebBrowserEvents2 Members

The following tables list the members exposed by DWebBrowserEvents2.

Public Methods

	Name	Description
	BeforeNavigate2 (see page 19288)	This is BeforeNavigate2, a member of class DWebBrowserEvents2.
	ClientToHostWindow (see page 19288)	This is ClientToHostWindow, a member of class DWebBrowserEvents2.
	CommandStateChange (see page 19288)	This is CommandStateChange, a member of class DWebBrowserEvents2.
	DocumentComplete (see page 19288)	This is DocumentComplete, a member of class DWebBrowserEvents2.
	DownloadBegin (see page 19289)	This is DownloadBegin, a member of class DWebBrowserEvents2.
	DownloadComplete (see page 19289)	This is DownloadComplete, a member of class DWebBrowserEvents2.
	FileDownload (see page 19289)	This is FileDownload, a member of class DWebBrowserEvents2.
	NavigateComplete2 (see page 19289)	This is NavigateComplete2, a member of class DWebBrowserEvents2.
	NavigateError (see page 19289)	This is NavigateError, a member of class DWebBrowserEvents2.
	NewWindow2 (see page 19290)	This is NewWindow2, a member of class DWebBrowserEvents2.
	OnFullScreen (see page 19290)	This is OnFullScreen, a member of class DWebBrowserEvents2.
	OnMenuBar (see page 19290)	This is OnMenuBar, a member of class DWebBrowserEvents2.
	OnQuit (see page 19290)	This is OnQuit, a member of class DWebBrowserEvents2.
	OnStatusBar (see page 19291)	This is OnStatusBar, a member of class DWebBrowserEvents2.
	OnTheaterMode (see page 19291)	This is OnTheaterMode, a member of class DWebBrowserEvents2.

	OnToolBar (see page 19291)	This is OnToolBar, a member of class DWebBrowserEvents2.
	OnVisible (see page 19291)	This is OnVisible, a member of class DWebBrowserEvents2.
	PrintTemplateInstantiation (see page 19291)	This is PrintTemplateInstantiation, a member of class DWebBrowserEvents2.
	PrintTemplateTeardown (see page 19292)	This is PrintTemplateTeardown, a member of class DWebBrowserEvents2.
	PrivacyImpactedStateChange (see page 19292)	This is PrivacyImpactedStateChange, a member of class DWebBrowserEvents2.
	ProgressChange (see page 19292)	This is ProgressChange, a member of class DWebBrowserEvents2.
	PropertyChange (see page 19292)	This is PropertyChange, a member of class DWebBrowserEvents2.
	SetSecureLockIcon (see page 19293)	This is SetSecureLockIcon, a member of class DWebBrowserEvents2.
	StatusTextChange (see page 19293)	This is StatusTextChange, a member of class DWebBrowserEvents2.
	TitleChange (see page 19293)	This is TitleChange, a member of class DWebBrowserEvents2.
	UpdatePageStatus (see page 19293)	This is UpdatePageStatus, a member of class DWebBrowserEvents2.
	WindowClosing (see page 19293)	This is WindowClosing, a member of class DWebBrowserEvents2.
	WindowSetHeight (see page 19294)	This is WindowSetHeight, a member of class DWebBrowserEvents2.
	WindowSetLeft (see page 19294)	This is WindowSetLeft, a member of class DWebBrowserEvents2.
	WindowSetResizable (see page 19294)	This is WindowSetResizable, a member of class DWebBrowserEvents2.
	WindowSetTop (see page 19294)	This is WindowSetTop, a member of class DWebBrowserEvents2.
	WindowSetWidth (see page 19295)	This is WindowSetWidth, a member of class DWebBrowserEvents2.

132.2.5.2 DWebBrowserEvents2 Methods

The methods of the DWebBrowserEvents2 class are listed here.

Public Methods

	Name	Description
	BeforeNavigate2 (see page 19288)	This is BeforeNavigate2, a member of class DWebBrowserEvents2.
	ClientToHostWindow (see page 19288)	This is ClientToHostWindow, a member of class DWebBrowserEvents2.
	CommandStateChange (see page 19288)	This is CommandStateChange, a member of class DWebBrowserEvents2.
	DocumentComplete (see page 19288)	This is DocumentComplete, a member of class DWebBrowserEvents2.
	DownloadBegin (see page 19289)	This is DownloadBegin, a member of class DWebBrowserEvents2.
	DownloadComplete (see page 19289)	This is DownloadComplete, a member of class DWebBrowserEvents2.
	FileDownload (see page 19289)	This is FileDownload, a member of class DWebBrowserEvents2.
	NavigateComplete2 (see page 19289)	This is NavigateComplete2, a member of class DWebBrowserEvents2.
	NavigateError (see page 19289)	This is NavigateError, a member of class DWebBrowserEvents2.
	NewWindow2 (see page 19290)	This is NewWindow2, a member of class DWebBrowserEvents2.
	OnFullScreen (see page 19290)	This is OnFullScreen, a member of class DWebBrowserEvents2.
	OnMenuBar (see page 19290)	This is OnMenuBar, a member of class DWebBrowserEvents2.
	OnQuit (see page 19290)	This is OnQuit, a member of class DWebBrowserEvents2.
	OnStatusBar (see page 19291)	This is OnStatusBar, a member of class DWebBrowserEvents2.
	OnTheaterMode (see page 19291)	This is OnTheaterMode, a member of class DWebBrowserEvents2.
	OnToolBar (see page 19291)	This is OnToolBar, a member of class DWebBrowserEvents2.
	OnVisible (see page 19291)	This is OnVisible, a member of class DWebBrowserEvents2.
	PrintTemplateInstantiation (see page 19291)	This is PrintTemplateInstantiation, a member of class DWebBrowserEvents2.
	PrintTemplateTeardown (see page 19292)	This is PrintTemplateTeardown, a member of class DWebBrowserEvents2.
	PrivacyImpactedStateChange (see page 19292)	This is PrivacyImpactedStateChange, a member of class DWebBrowserEvents2.
	ProgressChange (see page 19292)	This is ProgressChange, a member of class DWebBrowserEvents2.
	PropertyChange (see page 19292)	This is PropertyChange, a member of class DWebBrowserEvents2.
	SetSecureLockIcon (see page 19293)	This is SetSecureLockIcon, a member of class DWebBrowserEvents2.
	StatusTextChange (see page 19293)	This is StatusTextChange, a member of class DWebBrowserEvents2.
	TitleChange (see page 19293)	This is TitleChange, a member of class DWebBrowserEvents2.
	UpdatePageStatus (see page 19293)	This is UpdatePageStatus, a member of class DWebBrowserEvents2.
	WindowClosing (see page 19293)	This is WindowClosing, a member of class DWebBrowserEvents2.
	WindowSetHeight (see page 19294)	This is WindowSetHeight, a member of class DWebBrowserEvents2.
	WindowSetLeft (see page 19294)	This is WindowSetLeft, a member of class DWebBrowserEvents2.

	WindowSetResizable (see page 19294)	This is WindowSetResizable, a member of class DWebBrowserEvents2.
	WindowSetTop (see page 19294)	This is WindowSetTop, a member of class DWebBrowserEvents2.
	WindowSetWidth (see page 19295)	This is WindowSetWidth, a member of class DWebBrowserEvents2.

132.2.5.2.1 DWebBrowserEvents2.BeforeNavigate2 Method

Pascal

```
procedure BeforeNavigate2(const pDisp: IDispatch; const URL: OleVariant; const Flags:
OleVariant; const TargetFrameName: OleVariant; const PostData: OleVariant; const Headers:
OleVariant; var Cancel: WordBool);
```

C++

```
__fastcall BeforeNavigate2(const IDispatch pDisp, const OleVariant URL, const OleVariant
Flags, const OleVariant TargetFrameName, const OleVariant PostData, const OleVariant Headers,
WordBool Cancel);
```

Description

This is BeforeNavigate2, a member of class DWebBrowserEvents2.

132.2.5.2.2 DWebBrowserEvents2.ClientToHostWindow Method

Pascal

```
procedure ClientToHostWindow(var CX: Integer; var CY: Integer);
```

C++

```
__fastcall ClientToHostWindow(int CX, int CY);
```

Description

This is ClientToHostWindow, a member of class DWebBrowserEvents2.

132.2.5.2.3 DWebBrowserEvents2.CommandStateChange Method

Pascal

```
procedure CommandStateChange(Command: Integer; Enable: WordBool);
```

C++

```
__fastcall CommandStateChange(int Command, WordBool Enable);
```

Description

This is CommandStateChange, a member of class DWebBrowserEvents2.

132.2.5.2.4 DWebBrowserEvents2.DocumentComplete Method

Pascal

```
procedure DocumentComplete(const pDisp: IDispatch; var URL: OleVariant);
```

C++

```
__fastcall DocumentComplete(const IDispatch pDisp, OleVariant URL);
```

Description

This is DocumentComplete, a member of class DWebBrowserEvents2.

132.2.5.2.5 DWebBrowserEvents2.DownloadBegin Method

Pascal

```
procedure DownloadBegin;
```

C++

```
__fastcall DownloadBegin();
```

Description

This is DownloadBegin, a member of class DWebBrowserEvents2.

132.2.5.2.6 DWebBrowserEvents2.DownloadComplete Method

Pascal

```
procedure DownloadComplete;
```

C++

```
__fastcall DownloadComplete();
```

Description

This is DownloadComplete, a member of class DWebBrowserEvents2.

132.2.5.2.7 DWebBrowserEvents2.FileDownload Method

Pascal

```
procedure FileDownload(var Cancel: WordBool);
```

C++

```
__fastcall FileDownload(WordBool Cancel);
```

Description

This is FileDownload, a member of class DWebBrowserEvents2.

132.2.5.2.8 DWebBrowserEvents2.NavigateComplete2 Method

Pascal

```
procedure NavigateComplete2(const pDisp: IDispatch; var URL: OleVariant);
```

C++

```
__fastcall NavigateComplete2(const IDispatch pDisp, OleVariant URL);
```

Description

This is NavigateComplete2, a member of class DWebBrowserEvents2.

132.2.5.2.9 DWebBrowserEvents2.NavigateError Method

Pascal

```
procedure NavigateError(const pDisp: IDispatch; var URL: OleVariant; var Frame: OleVariant;
var StatusCode: OleVariant; var Cancel: WordBool);
```

C++

```
__fastcall NavigateError(const IDispatch pDisp, OleVariant URL, OleVariant Frame, OleVariant
StatusCode, WordBool Cancel);
```

Description

This is NavigateError, a member of class DWebBrowserEvents2.

132.2.5.2.10 DWebBrowserEvents2.NewWindow2 Method

Pascal

```
procedure NewWindow2(var ppDisp: IDispatch; var Cancel: WordBool);
```

C++

```
__fastcall NewWindow2(IDispatch ppDisp, WordBool Cancel);
```

Description

This is NewWindow2, a member of class DWebBrowserEvents2.

132.2.5.2.11 DWebBrowserEvents2.OnFullScreen Method

Pascal

```
procedure OnFullScreen(FullScreen: WordBool);
```

C++

```
__fastcall OnFullScreen(WordBool FullScreen);
```

Description

This is OnFullScreen, a member of class DWebBrowserEvents2.

132.2.5.2.12 DWebBrowserEvents2.OnMenuBar Method

Pascal

```
procedure OnMenuBar(MenuBar: WordBool);
```

C++

```
__fastcall OnMenuBar(WordBool MenuBar);
```

Description

This is OnMenuBar, a member of class DWebBrowserEvents2.

132.2.5.2.13 DWebBrowserEvents2.OnQuit Method

Pascal

```
procedure OnQuit;
```

C++

```
__fastcall OnQuit();
```

Description

This is OnQuit, a member of class DWebBrowserEvents2.

132.2.5.2.14 DWebBrowserEvents2.OnStatusBar Method

Pascal

```
procedure OnStatusBar(StatusBar: WordBool);
```

C++

```
__fastcall OnStatusBar(WordBool StatusBar);
```

Description

This is OnStatusBar, a member of class DWebBrowserEvents2.

132.2.5.2.15 DWebBrowserEvents2.OnTheaterMode Method

Pascal

```
procedure OnTheaterMode(TheaterMode: WordBool);
```

C++

```
__fastcall OnTheaterMode(WordBool TheaterMode);
```

Description

This is OnTheaterMode, a member of class DWebBrowserEvents2.

132.2.5.2.16 DWebBrowserEvents2.OnToolBar Method

Pascal

```
procedure OnToolBar(ToolBar: WordBool);
```

C++

```
__fastcall OnToolBar(WordBool ToolBar);
```

Description

This is OnToolBar, a member of class DWebBrowserEvents2.

132.2.5.2.17 DWebBrowserEvents2.OnVisible Method

Pascal

```
procedure OnVisible(Visible: WordBool);
```

C++

```
__fastcall OnVisible(WordBool Visible);
```

Description

This is OnVisible, a member of class DWebBrowserEvents2.

132.2.5.2.18 DWebBrowserEvents2.PrintTemplateInstantiation Method

Pascal

```
procedure PrintTemplateInstantiation(const pDisp: IDispatch);
```

C++

```
__fastcall PrintTemplateInstantiation(const IDispatch pDisp);
```

Description

This is PrintTemplateInstantiation, a member of class DWebBrowserEvents2.

132.2.5.2.19 DWebBrowserEvents2.PrintTemplateTeardown Method

Pascal

```
procedure PrintTemplateTeardown(const pDisp: IDispatch);
```

C++

```
__fastcall PrintTemplateTeardown(const IDispatch pDisp);
```

Description

This is PrintTemplateTeardown, a member of class DWebBrowserEvents2.

132.2.5.2.20 DWebBrowserEvents2.PrivacyImpactedStateChange Method

Pascal

```
procedure PrivacyImpactedStateChange(bImpacted: WordBool);
```

C++

```
__fastcall PrivacyImpactedStateChange(WordBool bImpacted);
```

Description

This is PrivacyImpactedStateChange, a member of class DWebBrowserEvents2.

132.2.5.2.21 DWebBrowserEvents2.ProgressChange Method

Pascal

```
procedure ProgressChange(Progress: Integer; ProgressMax: Integer);
```

C++

```
__fastcall ProgressChange(int Progress, int ProgressMax);
```

Description

This is ProgressChange, a member of class DWebBrowserEvents2.

132.2.5.2.22 DWebBrowserEvents2.PropertyChange Method

Pascal

```
procedure PropertyChange(const szProperty: WideString);
```

C++

```
__fastcall PropertyChange(const BSTR szProperty);
```

Description

This is PropertyChange, a member of class DWebBrowserEvents2.

132.2.5.2.23 DWebBrowserEvents2.SetSecureLockIcon Method

Pascal

```
procedure SetSecureLockIcon(SecureLockIcon: Integer);
```

C++

```
__fastcall SetSecureLockIcon(int SecureLockIcon);
```

Description

This is SetSecureLockIcon, a member of class DWebBrowserEvents2.

132.2.5.2.24 DWebBrowserEvents2.StatusTextChange Method

Pascal

```
procedure StatusTextChange(const Text: WideString);
```

C++

```
__fastcall StatusTextChange(const BSTR Text);
```

Description

This is StatusTextChange, a member of class DWebBrowserEvents2.

132.2.5.2.25 DWebBrowserEvents2.TitleChange Method

Pascal

```
procedure TitleChange(const Text: WideString);
```

C++

```
__fastcall TitleChange(const BSTR Text);
```

Description

This is TitleChange, a member of class DWebBrowserEvents2.

132.2.5.2.26 DWebBrowserEvents2.UpdatePageStatus Method

Pascal

```
procedure UpdatePageStatus(const pDisp: IDispatch; var nPage: OleVariant; var fDone:
OleVariant);
```

C++

```
__fastcall UpdatePageStatus(const IDispatch pDisp, OleVariant nPage, OleVariant fDone);
```

Description

This is UpdatePageStatus, a member of class DWebBrowserEvents2.

132.2.5.2.27 DWebBrowserEvents2.WindowClosing Method

Pascal

```
procedure WindowClosing(IsChildWindow: WordBool; var Cancel: WordBool);
```

C++

```
__fastcall WindowClosing(WordBool IsChildWindow, WordBool Cancel);
```

Description

This is WindowClosing, a member of class DWebBrowserEvents2.

132.2.5.2.28 DWebBrowserEvents2.WindowSetHeight Method

Pascal

```
procedure WindowSetHeight(Height: Integer);
```

C++

```
__fastcall WindowSetHeight(int Height);
```

Description

This is WindowSetHeight, a member of class DWebBrowserEvents2.

132.2.5.2.29 DWebBrowserEvents2.WindowSetLeft Method

Pascal

```
procedure WindowSetLeft(Left: Integer);
```

C++

```
__fastcall WindowSetLeft(int Left);
```

Description

This is WindowSetLeft, a member of class DWebBrowserEvents2.

132.2.5.2.30 DWebBrowserEvents2.WindowSetResizable Method

Pascal

```
procedure WindowSetResizable(Resizable: WordBool);
```

C++

```
__fastcall WindowSetResizable(WordBool Resizable);
```

Description

This is WindowSetResizable, a member of class DWebBrowserEvents2.

132.2.5.2.31 DWebBrowserEvents2.WindowSetTop Method

Pascal

```
procedure WindowSetTop(Top: Integer);
```

C++

```
__fastcall WindowSetTop(int Top);
```

Description

This is WindowSetTop, a member of class DWebBrowserEvents2.

132.2.5.2.32 DWebBrowserEvents2.WindowSetWidth Method

Pascal

```
procedure WindowSetWidth(Width: Integer);
```

C++

```
__fastcall WindowSetWidth(int Width);
```

Description

This is WindowSetWidth, a member of class DWebBrowserEvents2.

132.2.6 IScriptErrorList Interface

Class Hierarchy

Pascal

```
IScriptErrorList = interface(IDispatch);
```

C++

```
__interface IScriptErrorList : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.IScriptErrorList.

132.2.6.1 IScriptErrorList Members

The following tables list the members exposed by IScriptErrorList.

Public Methods

	Name	Description
	advanceError (see page 19296)	This is advanceError, a member of class IScriptErrorList.
	canAdvanceError (see page 19296)	This is canAdvanceError, a member of class IScriptErrorList.
	canRetreatError (see page 19296)	This is canRetreatError, a member of class IScriptErrorList.
	getAlwaysShowLockState (see page 19297)	This is getAlwaysShowLockState, a member of class IScriptErrorList.
	getDetailsPaneOpen (see page 19297)	This is getDetailsPaneOpen, a member of class IScriptErrorList.
	getErrorChar (see page 19297)	This is getErrorChar, a member of class IScriptErrorList.
	getErrorCode (see page 19297)	This is getErrorCode, a member of class IScriptErrorList.
	getErrorLine (see page 19298)	This is getErrorLine, a member of class IScriptErrorList.
	getErrorMsg (see page 19298)	This is getErrorMsg, a member of class IScriptErrorList.
	getErrorUrl (see page 19298)	This is getErrorUrl, a member of class IScriptErrorList.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	getPerErrorDisplay (see page 19298)	This is getPerErrorDisplay, a member of class IScriptErrorList.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.

	retreatError (see page 19298)	This is retreatError, a member of class IScriptErrorList.
	setDetailsPaneOpen (see page 19299)	This is setDetailsPaneOpen, a member of class IScriptErrorList.
	setPerErrorDisplay (see page 19299)	This is setPerErrorDisplay, a member of class IScriptErrorList.

132.2.6.2 IScriptErrorList Methods

The methods of the IScriptErrorList class are listed here.

Public Methods

	Name	Description
	advanceError (see page 19296)	This is advanceError, a member of class IScriptErrorList.
	canAdvanceError (see page 19296)	This is canAdvanceError, a member of class IScriptErrorList.
	canRetreatError (see page 19296)	This is canRetreatError, a member of class IScriptErrorList.
	getAlwaysShowLockState (see page 19297)	This is getAlwaysShowLockState, a member of class IScriptErrorList.
	getDetailsPaneOpen (see page 19297)	This is getDetailsPaneOpen, a member of class IScriptErrorList.
	getErrorChar (see page 19297)	This is getErrorChar, a member of class IScriptErrorList.
	getErrorCode (see page 19297)	This is getErrorCode, a member of class IScriptErrorList.
	getErrorLine (see page 19298)	This is getErrorLine, a member of class IScriptErrorList.
	getErrorMsg (see page 19298)	This is getErrorMsg, a member of class IScriptErrorList.
	getErrorUrl (see page 19298)	This is getErrorUrl, a member of class IScriptErrorList.
	getPerErrorDisplay (see page 19298)	This is getPerErrorDisplay, a member of class IScriptErrorList.
	retreatError (see page 19298)	This is retreatError, a member of class IScriptErrorList.
	setDetailsPaneOpen (see page 19299)	This is setDetailsPaneOpen, a member of class IScriptErrorList.
	setPerErrorDisplay (see page 19299)	This is setPerErrorDisplay, a member of class IScriptErrorList.

132.2.6.2.1 IScriptErrorList.advanceError Method

Pascal

```
procedure advanceError; safecall;
```

C++

```
__fastcall advanceError();
```

Description

This is advanceError, a member of class IScriptErrorList.

132.2.6.2.2 IScriptErrorList.canAdvanceError Method

Pascal

```
function canAdvanceError: Integer; safecall;
```

C++

```
__fastcall int canAdvanceError();
```

Description

This is canAdvanceError, a member of class IScriptErrorList.

132.2.6.2.3 IScriptErrorList.canRetreatError Method

Pascal

```
function canRetreatError: Integer; safecall;
```

C++

```
__fastcall int canRetreatError();
```

Description

This is canRetreatError, a member of class IScriptErrorList.

132.2.6.2.4 IScriptErrorList.getAlwaysShowLockState Method

Pascal

```
function getAlwaysShowLockState: Integer; safecall;
```

C++

```
__fastcall int getAlwaysShowLockState();
```

Description

This is getAlwaysShowLockState, a member of class IScriptErrorList.

132.2.6.2.5 IScriptErrorList.getDetailsPaneOpen Method

Pascal

```
function getDetailsPaneOpen: Integer; safecall;
```

C++

```
__fastcall int getDetailsPaneOpen();
```

Description

This is getDetailsPaneOpen, a member of class IScriptErrorList.

132.2.6.2.6 IScriptErrorList.getErrorChar Method

Pascal

```
function getErrorChar: Integer; safecall;
```

C++

```
__fastcall int getErrorChar();
```

Description

This is getErrorChar, a member of class IScriptErrorList.

132.2.6.2.7 IScriptErrorList.getErrorCode Method

Pascal

```
function getErrorCode: Integer; safecall;
```

C++

```
__fastcall int getErrorCode();
```

Description

This is getErrorCode, a member of class IScriptErrorList.

132.2.6.2.8 IScriptErrorList.getErrorLine Method

Pascal

```
function getErrorLine: Integer; safecall;
```

C++

```
__fastcall int getErrorLine();
```

Description

This is getErrorLine, a member of class IScriptErrorList.

132.2.6.2.9 IScriptErrorList.getErrorMsg Method

Pascal

```
function getErrorMsg: WideString; safecall;
```

C++

```
__fastcall BSTR getErrorMsg();
```

Description

This is getErrorMsg, a member of class IScriptErrorList.

132.2.6.2.10 IScriptErrorList.getErrorUrl Method

Pascal

```
function getErrorUrl: WideString; safecall;
```

C++

```
__fastcall BSTR getErrorUrl();
```

Description

This is getErrorUrl, a member of class IScriptErrorList.

132.2.6.2.11 IScriptErrorList.getPerErrorDisplay Method

Pascal

```
function getPerErrorDisplay: Integer; safecall;
```

C++

```
__fastcall int getPerErrorDisplay();
```

Description

This is getPerErrorDisplay, a member of class IScriptErrorList.

132.2.6.2.12 IScriptErrorList.retreatError Method

Pascal

```
procedure retreatError; safecall;
```

C++

```
__fastcall retreatError();
```

Description

This is retreatError, a member of class IScriptErrorList.

132.2.6.2.13 IScriptErrorList.setDetailsPaneOpen Method

Pascal

```
procedure setDetailsPaneOpen(fDetailsPaneOpen: Integer); safecall;
```

C++

```
__fastcall setDetailsPaneOpen(int fDetailsPaneOpen);
```

Description

This is setDetailsPaneOpen, a member of class IScriptErrorList.

132.2.6.2.14 IScriptErrorList.setPerErrorDisplay Method

Pascal

```
procedure setPerErrorDisplay(fPerErrorDisplay: Integer); safecall;
```

C++

```
__fastcall setPerErrorDisplay(int fPerErrorDisplay);
```

Description

This is setPerErrorDisplay, a member of class IScriptErrorList.

132.2.7 IScriptErrorListDisp Interface

Class Hierarchy

SHDocVw.IScriptErrorListDisp

Pascal

```
IScriptErrorListDisp = dispinterface;
```

C++

```
class IScriptErrorListDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IScriptErrorListDisp.

132.2.7.1 IScriptErrorListDisp Members

The following tables list the members exposed by IScriptErrorListDisp.

Public Methods

	Name	Description
	advanceError (see page 19300)	This is advanceError, a member of class IScriptErrorListDisp.
	canAdvanceError (see page 19301)	This is canAdvanceError, a member of class IScriptErrorListDisp.
	canRetreatError (see page 19301)	This is canRetreatError, a member of class IScriptErrorListDisp.
	getAlwaysShowLockState (see page 19301)	This is getAlwaysShowLockState, a member of class IScriptErrorListDisp.
	getDetailsPaneOpen (see page 19301)	This is getDetailsPaneOpen, a member of class IScriptErrorListDisp.
	getErrorChar (see page 19301)	This is getErrorChar, a member of class IScriptErrorListDisp.
	getErrorCode (see page 19302)	This is getErrorCode, a member of class IScriptErrorListDisp.
	getErrorLine (see page 19302)	This is getErrorLine, a member of class IScriptErrorListDisp.
	getErrorMsg (see page 19302)	This is getErrorMsg, a member of class IScriptErrorListDisp.
	getErrorUrl (see page 19302)	This is getErrorUrl, a member of class IScriptErrorListDisp.
	getPerErrorDisplay (see page 19303)	This is getPerErrorDisplay, a member of class IScriptErrorListDisp.
	retreatError (see page 19303)	This is retreatError, a member of class IScriptErrorListDisp.
	setDetailsPaneOpen (see page 19303)	This is setDetailsPaneOpen, a member of class IScriptErrorListDisp.
	setPerErrorDisplay (see page 19303)	This is setPerErrorDisplay, a member of class IScriptErrorListDisp.

132.2.7.2 IScriptErrorListDisp Methods

The methods of the IScriptErrorListDisp class are listed here.

Public Methods

	Name	Description
	advanceError (see page 19300)	This is advanceError, a member of class IScriptErrorListDisp.
	canAdvanceError (see page 19301)	This is canAdvanceError, a member of class IScriptErrorListDisp.
	canRetreatError (see page 19301)	This is canRetreatError, a member of class IScriptErrorListDisp.
	getAlwaysShowLockState (see page 19301)	This is getAlwaysShowLockState, a member of class IScriptErrorListDisp.
	getDetailsPaneOpen (see page 19301)	This is getDetailsPaneOpen, a member of class IScriptErrorListDisp.
	getErrorChar (see page 19301)	This is getErrorChar, a member of class IScriptErrorListDisp.
	getErrorCode (see page 19302)	This is getErrorCode, a member of class IScriptErrorListDisp.
	getErrorLine (see page 19302)	This is getErrorLine, a member of class IScriptErrorListDisp.
	getErrorMsg (see page 19302)	This is getErrorMsg, a member of class IScriptErrorListDisp.
	getErrorUrl (see page 19302)	This is getErrorUrl, a member of class IScriptErrorListDisp.
	getPerErrorDisplay (see page 19303)	This is getPerErrorDisplay, a member of class IScriptErrorListDisp.
	retreatError (see page 19303)	This is retreatError, a member of class IScriptErrorListDisp.
	setDetailsPaneOpen (see page 19303)	This is setDetailsPaneOpen, a member of class IScriptErrorListDisp.
	setPerErrorDisplay (see page 19303)	This is setPerErrorDisplay, a member of class IScriptErrorListDisp.

132.2.7.2.1 IScriptErrorListDisp.advanceError Method

Pascal

```
procedure advanceError;
```

C++

```
__fastcall advanceError();
```

Description

This is advanceError, a member of class IScriptErrorListDisp.

132.2.7.2.2 IScriptErrorListDisp.canAdvanceError Method

Pascal

```
function canAdvanceError: Integer;
```

C++

```
__fastcall int canAdvanceError();
```

Description

This is canAdvanceError, a member of class IScriptErrorListDisp.

132.2.7.2.3 IScriptErrorListDisp.canRetreatError Method

Pascal

```
function canRetreatError: Integer;
```

C++

```
__fastcall int canRetreatError();
```

Description

This is canRetreatError, a member of class IScriptErrorListDisp.

132.2.7.2.4 IScriptErrorListDisp.getAlwaysShowLockState Method

Pascal

```
function getAlwaysShowLockState: Integer;
```

C++

```
__fastcall int getAlwaysShowLockState();
```

Description

This is getAlwaysShowLockState, a member of class IScriptErrorListDisp.

132.2.7.2.5 IScriptErrorListDisp.getDetailsPaneOpen Method

Pascal

```
function getDetailsPaneOpen: Integer;
```

C++

```
__fastcall int getDetailsPaneOpen();
```

Description

This is getDetailsPaneOpen, a member of class IScriptErrorListDisp.

132.2.7.2.6 IScriptErrorListDisp.getErrorChar Method

Pascal

```
function getErrorChar: Integer;
```

C++

```
__fastcall int getErrorChar();
```

Description

This is getErrorChar, a member of class IScriptErrorListDisp.

132.2.7.2.7 IScriptErrorListDisp.getErrorCode Method

Pascal

```
function getErrorCode: Integer;
```

C++

```
__fastcall int getErrorCode();
```

Description

This is getErrorCode, a member of class IScriptErrorListDisp.

132.2.7.2.8 IScriptErrorListDisp.getErrorLine Method

Pascal

```
function getErrorLine: Integer;
```

C++

```
__fastcall int getErrorLine();
```

Description

This is getErrorLine, a member of class IScriptErrorListDisp.

132.2.7.2.9 IScriptErrorListDisp.getErrorMsg Method

Pascal

```
function getErrorMsg: WideString;
```

C++

```
__fastcall BSTR getErrorMsg();
```

Description

This is getErrorMsg, a member of class IScriptErrorListDisp.

132.2.7.2.10 IScriptErrorListDisp.getErrorUrl Method

Pascal

```
function getErrorUrl: WideString;
```

C++

```
__fastcall BSTR getErrorUrl();
```

Description

This is getErrorUrl, a member of class IScriptErrorListDisp.

132.2.7.2.11 IScriptErrorListDisp.getPerErrorDisplay Method

Pascal

```
function getPerErrorDisplay: Integer;
```

C++

```
__fastcall int getPerErrorDisplay();
```

Description

This is getPerErrorDisplay, a member of class IScriptErrorListDisp.

132.2.7.2.12 IScriptErrorListDisp.retreatError Method

Pascal

```
procedure retreatError;
```

C++

```
__fastcall retreatError();
```

Description

This is retreatError, a member of class IScriptErrorListDisp.

132.2.7.2.13 IScriptErrorListDisp.setDetailsPaneOpen Method

Pascal

```
procedure setDetailsPaneOpen(fDetailsPaneOpen: Integer);
```

C++

```
__fastcall setDetailsPaneOpen(int fDetailsPaneOpen);
```

Description

This is setDetailsPaneOpen, a member of class IScriptErrorListDisp.

132.2.7.2.14 IScriptErrorListDisp.setPerErrorDisplay Method

Pascal

```
procedure setPerErrorDisplay(fPerErrorDisplay: Integer);
```

C++

```
__fastcall setPerErrorDisplay(int fPerErrorDisplay);
```

Description

This is setPerErrorDisplay, a member of class IScriptErrorListDisp.

132.2.8 ISearch Interface

Class Hierarchy

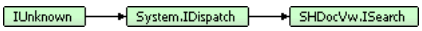

Pascal

```
ISearch = interface(IDispatch);
```

C++

```
__interface ISearch : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.ISearch.

132.2.8.1 ISearch Members

The following tables list the members exposed by ISearch.

Public Methods

	Name	Description
	Get_Id (see page 19304)	This is Get_Id, a member of class ISearch.
	Get_Title (see page 19305)	This is Get_Title, a member of class ISearch.
	Get_URL (see page 19305)	This is Get_URL, a member of class ISearch.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.

Public Properties

	Name	Description
R	Id (see page 19305)	This is Id, a member of class ISearch.
R	Title (see page 19305)	This is Title, a member of class ISearch.
R	URL (see page 19306)	This is URL, a member of class ISearch.

132.2.8.2 ISearch Methods

The methods of the ISearch class are listed here.

Public Methods

	Name	Description
	Get_Id (see page 19304)	This is Get_Id, a member of class ISearch.
	Get_Title (see page 19305)	This is Get_Title, a member of class ISearch.
	Get_URL (see page 19305)	This is Get_URL, a member of class ISearch.

132.2.8.2.1 ISearch.Get_Id Method

Pascal

```
function Get_Id: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Id();
```

Description

This is Get_Id, a member of class ISearch.

132.2.8.2.2 ISearch.Get_Title Method

Pascal

```
function Get_Title: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Title();
```

Description

This is Get_Title, a member of class ISearch.

132.2.8.2.3 ISearch.Get_URL Method

Pascal

```
function Get_URL: WideString; safecall;
```

C++

```
__fastcall BSTR Get_URL();
```

Description

This is Get_URL, a member of class ISearch.

132.2.8.3 ISearch Properties

The properties of the ISearch class are listed here.

Public Properties

	Name	Description
R	Id (see page 19305)	This is Id, a member of class ISearch.
R	Title (see page 19305)	This is Title, a member of class ISearch.
R	URL (see page 19306)	This is URL, a member of class ISearch.

132.2.8.3.1 ISearch.Id Property

Pascal

```
property Id: WideString;
```

C++

```
__property BSTR Id;
```

Description

This is Id, a member of class ISearch.

132.2.8.3.2 ISearch.Title Property

Pascal

```
property Title: WideString;
```

C++

```
__property BSTR Title;
```

Description

This is Title, a member of class ISearch.

132.2.8.3.3 ISearch.URL Property

Pascal

```
property URL: WideString;
```

C++

```
__property BSTR URL;
```

Description

This is URL, a member of class ISearch.

132.2.9 ISearchAssistantOC Interface

Class Hierarchy

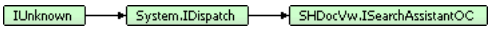

Pascal

```
ISearchAssistantOC = interface(IDispatch);
```

C++

```
__interface ISearchAssistantOC : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchAssistantOC.

132.2.9.1 ISearchAssistantOC Members

The following tables list the members exposed by ISearchAssistantOC.

Public Methods

	Name	Description
	AddNextMenuItem (see page 19308)	This is AddNextMenuItem, a member of class ISearchAssistantOC.
	EncodeString (see page 19308)	This is EncodeString, a member of class ISearchAssistantOC.
	FindComputer (see page 19308)	This is FindComputer, a member of class ISearchAssistantOC.
	FindFilesOrFolders (see page 19309)	This is FindFilesOrFolders, a member of class ISearchAssistantOC.
	FindOnWeb (see page 19309)	This is FindOnWeb, a member of class ISearchAssistantOC.
	FindPeople (see page 19309)	This is FindPeople, a member of class ISearchAssistantOC.
	FindPrinter (see page 19309)	This is FindPrinter, a member of class ISearchAssistantOC.
	Get_ASProvider (see page 19309)	This is Get_ASProvider, a member of class ISearchAssistantOC.
	Get_ASSetting (see page 19310)	This is Get_ASSetting, a member of class ISearchAssistantOC.

	Get_InWebFolder (see page 19310)	This is Get_InWebFolder, a member of class ISearchAssistantOC.
	Get_SearchAssistantDefault (see page 19310)	This is Get_SearchAssistantDefault, a member of class ISearchAssistantOC.
	Get_Searches (see page 19310)	This is Get_Searches, a member of class ISearchAssistantOC.
	Get_ShellFeaturesEnabled (see page 19311)	This is Get_ShellFeaturesEnabled, a member of class ISearchAssistantOC.
	Get_Version (see page 19311)	This is Get_Version, a member of class ISearchAssistantOC.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetProperty (see page 19311)	This is GetProperty, a member of class ISearchAssistantOC.
	GetSearchAssistantURL (see page 19311)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	IsRestricted (see page 19311)	This is IsRestricted, a member of class ISearchAssistantOC.
	NavigateToDefaultSearch (see page 19312)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC.
	NETDetectNextNavigate (see page 19312)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC.
	NotifySearchSettingsChanged (see page 19312)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC.
	PutFindText (see page 19312)	This is PutFindText, a member of class ISearchAssistantOC.
	PutProperty (see page 19313)	This is PutProperty, a member of class ISearchAssistantOC.
	ResetNextMenu (see page 19313)	This is ResetNextMenu, a member of class ISearchAssistantOC.
	Set_ASProvider (see page 19313)	This is Set_ASProvider, a member of class ISearchAssistantOC.
	Set_ASSetting (see page 19313)	This is Set_ASSetting, a member of class ISearchAssistantOC.
	Set_EventHandled (see page 19313)	This is Set_EventHandled, a member of class ISearchAssistantOC.
	SetDefaultSearchUrl (see page 19314)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC.

Public Properties

	Name	Description
	ASProvider (see page 19314)	This is ASProvider, a member of class ISearchAssistantOC.
	ASSetting (see page 19314)	This is ASSetting, a member of class ISearchAssistantOC.
W	EventHandled (see page 19315)	This is EventHandled, a member of class ISearchAssistantOC.
R	InWebFolder (see page 19315)	This is InWebFolder, a member of class ISearchAssistantOC.
R	SearchAssistantDefault (see page 19315)	This is SearchAssistantDefault, a member of class ISearchAssistantOC.
R	Searches (see page 19315)	This is Searches, a member of class ISearchAssistantOC.
R	ShellFeaturesEnabled (see page 19316)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC.
R	Version (see page 19316)	This is Version, a member of class ISearchAssistantOC.

132.2.9.2 ISearchAssistantOC Methods

The methods of the ISearchAssistantOC class are listed here.

Public Methods

	Name	Description
	AddNextMenuItem (see page 19308)	This is AddNextMenuItem, a member of class ISearchAssistantOC.
	EncodeString (see page 19308)	This is EncodeString, a member of class ISearchAssistantOC.
	FindComputer (see page 19308)	This is FindComputer, a member of class ISearchAssistantOC.
	FindFilesOrFolders (see page 19309)	This is FindFilesOrFolders, a member of class ISearchAssistantOC.
	FindOnWeb (see page 19309)	This is FindOnWeb, a member of class ISearchAssistantOC.
	FindPeople (see page 19309)	This is FindPeople, a member of class ISearchAssistantOC.
	FindPrinter (see page 19309)	This is FindPrinter, a member of class ISearchAssistantOC.
	Get_ASProvider (see page 19309)	This is Get_ASProvider, a member of class ISearchAssistantOC.
	Get_ASSetting (see page 19310)	This is Get_ASSetting, a member of class ISearchAssistantOC.
	Get_InWebFolder (see page 19310)	This is Get_InWebFolder, a member of class ISearchAssistantOC.
	Get_SearchAssistantDefault (see page 19310)	This is Get_SearchAssistantDefault, a member of class ISearchAssistantOC.
	Get_Searches (see page 19310)	This is Get_Searches, a member of class ISearchAssistantOC.
	Get_ShellFeaturesEnabled (see page 19311)	This is Get_ShellFeaturesEnabled, a member of class ISearchAssistantOC.

	Get_Version (see page 19311)	This is Get_Version, a member of class ISearchAssistantOC.
	GetProperty (see page 19311)	This is GetProperty, a member of class ISearchAssistantOC.
	GetSearchAssistantURL (see page 19311)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC.
	IsRestricted (see page 19311)	This is IsRestricted, a member of class ISearchAssistantOC.
	NavigateToDefaultSearch (see page 19312)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC.
	NETDetectNextNavigate (see page 19312)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC.
	NotifySearchSettingsChanged (see page 19312)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC.
	PutFindText (see page 19312)	This is PutFindText, a member of class ISearchAssistantOC.
	PutProperty (see page 19313)	This is PutProperty, a member of class ISearchAssistantOC.
	ResetNextMenu (see page 19313)	This is ResetNextMenu, a member of class ISearchAssistantOC.
	Set_ASProvider (see page 19313)	This is Set_ASProvider, a member of class ISearchAssistantOC.
	Set_ASSetting (see page 19313)	This is Set_ASSetting, a member of class ISearchAssistantOC.
	Set_EventHandled (see page 19313)	This is Set_EventHandled, a member of class ISearchAssistantOC.
	SetDefaultSearchUrl (see page 19314)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC.

132.2.9.2.1 ISearchAssistantOC.AddNextMenuItem Method

Pascal

```
procedure AddNextMenuItem(const bstrText: WideString; idItem: Integer); safecall;
```

C++

```
__fastcall AddNextMenuItem(const BSTR bstrText, int idItem);
```

Description

This is AddNextMenuItem, a member of class ISearchAssistantOC.

132.2.9.2.2 ISearchAssistantOC.EncodeString Method

Pascal

```
function EncodeString(const bstrValue: WideString; const bstrCharSet: WideString; bUseUTF8:
WordBool): WideString; safecall;
```

C++

```
__fastcall BSTR EncodeString(const BSTR bstrValue, const BSTR bstrCharSet, WordBool bUseUTF8);
```

Description

This is EncodeString, a member of class ISearchAssistantOC.

132.2.9.2.3 ISearchAssistantOC.FindComputer Method

Pascal

```
procedure FindComputer; safecall;
```

C++

```
__fastcall FindComputer();
```

Description

This is FindComputer, a member of class ISearchAssistantOC.

132.2.9.2.4 ISearchAssistantOC.FindFilesOrFolders Method

Pascal

```
procedure FindFilesOrFolders; safecall;
```

C++

```
__fastcall FindFilesOrFolders();
```

Description

This is FindFilesOrFolders, a member of class ISearchAssistantOC.

132.2.9.2.5 ISearchAssistantOC.FindOnWeb Method

Pascal

```
procedure FindOnWeb; safecall;
```

C++

```
__fastcall FindOnWeb();
```

Description

This is FindOnWeb, a member of class ISearchAssistantOC.

132.2.9.2.6 ISearchAssistantOC.FindPeople Method

Pascal

```
procedure FindPeople; safecall;
```

C++

```
__fastcall FindPeople();
```

Description

This is FindPeople, a member of class ISearchAssistantOC.

132.2.9.2.7 ISearchAssistantOC.FindPrinter Method

Pascal

```
procedure FindPrinter; safecall;
```

C++

```
__fastcall FindPrinter();
```

Description

This is FindPrinter, a member of class ISearchAssistantOC.

132.2.9.2.8 ISearchAssistantOC.Get_ASProvider Method

Pascal

```
function Get_ASProvider: WideString; safecall;
```

C++

```
__fastcall BSTR Get_ASProvider();
```

Description

This is Get_ASProvider, a member of class ISearchAssistantOC.

132.2.9.2.9 ISearchAssistantOC.Get_ASSetting Method

Pascal

```
function Get_ASSetting: SYSINT; safecall;
```

C++

```
__fastcall SYSINT Get_ASSetting();
```

Description

This is Get_ASSetting, a member of class ISearchAssistantOC.

132.2.9.2.10 ISearchAssistantOC.Get_InWebFolder Method

Pascal

```
function Get_InWebFolder: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_InWebFolder();
```

Description

This is Get_InWebFolder, a member of class ISearchAssistantOC.

132.2.9.2.11 ISearchAssistantOC.Get_SearchAssistantDefault Method

Pascal

```
function Get_SearchAssistantDefault: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_SearchAssistantDefault();
```

Description

This is Get_SearchAssistantDefault, a member of class ISearchAssistantOC.

132.2.9.2.12 ISearchAssistantOC.Get_Searches Method

Pascal

```
function Get_Searches: ISearches; safecall;
```

C++

```
__fastcall ISearches Get_Searches();
```

Description

This is Get_Searches, a member of class ISearchAssistantOC.

132.2.9.2.13 ISearchAssistantOC.Get_ShellFeaturesEnabled Method

Pascal

```
function Get_ShellFeaturesEnabled: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_ShellFeaturesEnabled();
```

Description

This is Get_ShellFeaturesEnabled, a member of class ISearchAssistantOC.

132.2.9.2.14 ISearchAssistantOC.Get_Version Method

Pascal

```
function Get_Version: SYSINT; safecall;
```

C++

```
__fastcall SYSINT Get_Version();
```

Description

This is Get_Version, a member of class ISearchAssistantOC.

132.2.9.2.15 ISearchAssistantOC.GetProperty Method

Pascal

```
function GetProperty(bPerLocale: WordBool; const bstrName: WideString): WideString; safecall;
```

C++

```
__fastcall BSTR GetProperty(WordBool bPerLocale, const BSTR bstrName);
```

Description

This is GetProperty, a member of class ISearchAssistantOC.

132.2.9.2.16 ISearchAssistantOC.GetSearchAssistantURL Method

Pascal

```
function GetSearchAssistantURL(bSubstitute: WordBool; bCustomize: WordBool): WideString;
safecall;
```

C++

```
__fastcall BSTR GetSearchAssistantURL(WordBool bSubstitute, WordBool bCustomize);
```

Description

This is GetSearchAssistantURL, a member of class ISearchAssistantOC.

132.2.9.2.17 ISearchAssistantOC.IsRestricted Method

Pascal

```
function IsRestricted(const bstrGuid: WideString): WordBool; safecall;
```

C++

```
__fastcall WordBool IsRestricted(const BSTR bstrGuid);
```

Description

This is IsRestricted, a member of class ISearchAssistantOC.

132.2.9.2.18 ISearchAssistantOC.NavigateToDefaultSearch Method

Pascal

```
procedure NavigateToDefaultSearch; safecall;
```

C++

```
__fastcall NavigateToDefaultSearch();
```

Description

This is NavigateToDefaultSearch, a member of class ISearchAssistantOC.

132.2.9.2.19 ISearchAssistantOC.NETDetectNextNavigate Method

Pascal

```
procedure NETDetectNextNavigate; safecall;
```

C++

```
__fastcall NETDetectNextNavigate();
```

Description

This is NETDetectNextNavigate, a member of class ISearchAssistantOC.

132.2.9.2.20 ISearchAssistantOC.NotifySearchSettingsChanged Method

Pascal

```
procedure NotifySearchSettingsChanged; safecall;
```

C++

```
__fastcall NotifySearchSettingsChanged();
```

Description

This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC.

132.2.9.2.21 ISearchAssistantOC.PutFindText Method

Pascal

```
procedure PutFindText(const FindText: WideString); safecall;
```

C++

```
__fastcall PutFindText(const BSTR FindText);
```

Description

This is PutFindText, a member of class ISearchAssistantOC.

132.2.9.2.22 ISearchAssistantOC.PutProperty Method

Pascal

```
procedure PutProperty(bPerLocale: WordBool; const bstrName: WideString; const bstrValue:
WideString); safecall;
```

C++

```
__fastcall PutProperty(WordBool bPerLocale, const BSTR bstrName, const BSTR bstrValue);
```

Description

This is PutProperty, a member of class ISearchAssistantOC.

132.2.9.2.23 ISearchAssistantOC.ResetNextMenu Method

Pascal

```
procedure ResetNextMenu; safecall;
```

C++

```
__fastcall ResetNextMenu();
```

Description

This is ResetNextMenu, a member of class ISearchAssistantOC.

132.2.9.2.24 ISearchAssistantOC.Set_ASProvider Method

Pascal

```
procedure Set_ASProvider(const pProvider: WideString); safecall;
```

C++

```
__fastcall Set_ASProvider(const BSTR pProvider);
```

Description

This is Set_ASProvider, a member of class ISearchAssistantOC.

132.2.9.2.25 ISearchAssistantOC.Set_ASSetting Method

Pascal

```
procedure Set_ASSetting(pSetting: SYSINT); safecall;
```

C++

```
__fastcall Set_ASSetting(SYSINT pSetting);
```

Description

This is Set_ASSetting, a member of class ISearchAssistantOC.

132.2.9.2.26 ISearchAssistantOC.Set_EventHandled Method

Pascal

```
procedure Set_EventHandled(Param1: WordBool); safecall;
```

C++

```
__fastcall Set_EventHandled(WordBool Param1);
```

Description

This is Set_EventHandled, a member of class ISearchAssistantOC.

132.2.9.2.27 ISearchAssistantOC.SetDefaultSearchUrl Method

Pascal

```
procedure SetDefaultSearchUrl(const bstrUrl: WideString); safecall;
```

C++

```
__fastcall SetDefaultSearchUrl(const BSTR bstrUrl);
```

Description

This is SetDefaultSearchUrl, a member of class ISearchAssistantOC.

132.2.9.3 ISearchAssistantOC Properties

The properties of the ISearchAssistantOC class are listed here.

Public Properties

	Name	Description
	ASProvider (see page 19314)	This is ASProvider, a member of class ISearchAssistantOC.
	ASSetting (see page 19314)	This is ASSetting, a member of class ISearchAssistantOC.
W	EventHandled (see page 19315)	This is EventHandled, a member of class ISearchAssistantOC.
R	InWebFolder (see page 19315)	This is InWebFolder, a member of class ISearchAssistantOC.
R	SearchAssistantDefault (see page 19315)	This is SearchAssistantDefault, a member of class ISearchAssistantOC.
R	Searches (see page 19315)	This is Searches, a member of class ISearchAssistantOC.
R	ShellFeaturesEnabled (see page 19316)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC.
R	Version (see page 19316)	This is Version, a member of class ISearchAssistantOC.

132.2.9.3.1 ISearchAssistantOC.ASProvider Property

Pascal

```
property ASProvider: WideString;
```

C++

```
__property BSTR ASProvider;
```

Description

This is ASProvider, a member of class ISearchAssistantOC.

132.2.9.3.2 ISearchAssistantOC.ASSetting Property

Pascal

```
property ASSetting: SYSINT;
```

C++

```
__property SYSINT ASSetting;
```

Description

This is ASSetting, a member of class ISearchAssistantOC.

132.2.9.3.3 ISearchAssistantOC.EventHandled Property

Pascal

```
property EventHandled: WordBool;
```

C++

```
__property WordBool EventHandled;
```

Description

This is EventHandled, a member of class ISearchAssistantOC.

132.2.9.3.4 ISearchAssistantOC.InWebFolder Property

Pascal

```
property InWebFolder: WordBool;
```

C++

```
__property WordBool InWebFolder;
```

Description

This is InWebFolder, a member of class ISearchAssistantOC.

132.2.9.3.5 ISearchAssistantOC.SearchAssistantDefault Property

Pascal

```
property SearchAssistantDefault: WordBool;
```

C++

```
__property WordBool SearchAssistantDefault;
```

Description

This is SearchAssistantDefault, a member of class ISearchAssistantOC.

132.2.9.3.6 ISearchAssistantOC.Searches Property

Pascal

```
property Searches: ISearches;
```

C++

```
__property ISearches Searches;
```

Description

This is Searches, a member of class ISearchAssistantOC.

132.2.9.3.7 ISearchAssistantOC.ShellFeaturesEnabled Property

Pascal

```
property ShellFeaturesEnabled: WordBool;
```

C++

```
__property WordBool ShellFeaturesEnabled;
```

Description

This is ShellFeaturesEnabled, a member of class ISearchAssistantOC.

132.2.9.3.8 ISearchAssistantOC.Version Property

Pascal

```
property Version: SYSINT;
```

C++

```
__property SYSINT Version;
```

Description

This is Version, a member of class ISearchAssistantOC.

132.2.10 ISearchAssistantOC2 Interface

Class Hierarchy

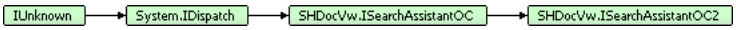

Pascal

```
ISearchAssistantOC2 = interface(ISearchAssistantOC);
```

C++

```
__interface ISearchAssistantOC2 : public ISearchAssistantOC;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchAssistantOC2.

132.2.10.1 ISearchAssistantOC2 Members

The following tables list the members exposed by ISearchAssistantOC2.

Public Methods

	Name	Description
	AddNextMenuItem (see page 19308)	This is AddNextMenuItem, a member of class ISearchAssistantOC.
	EncodeString (see page 19308)	This is EncodeString, a member of class ISearchAssistantOC.
	FindComputer (see page 19308)	This is FindComputer, a member of class ISearchAssistantOC.

	FindFilesOrFolders (see page 19309)	This is FindFilesOrFolders, a member of class ISearchAssistantOC.
	FindOnWeb (see page 19309)	This is FindOnWeb, a member of class ISearchAssistantOC.
	FindPeople (see page 19309)	This is FindPeople, a member of class ISearchAssistantOC.
	FindPrinter (see page 19309)	This is FindPrinter, a member of class ISearchAssistantOC.
	Get_ASProvider (see page 19309)	This is Get_ASProvider, a member of class ISearchAssistantOC.
	Get_ASSetting (see page 19310)	This is Get_ASSetting, a member of class ISearchAssistantOC.
	Get_InWebFolder (see page 19310)	This is Get_InWebFolder, a member of class ISearchAssistantOC.
	Get_SearchAssistantDefault (see page 19310)	This is Get_SearchAssistantDefault, a member of class ISearchAssistantOC.
	Get_Searches (see page 19310)	This is Get_Searches, a member of class ISearchAssistantOC.
	Get_ShellFeaturesEnabled (see page 19311)	This is Get_ShellFeaturesEnabled, a member of class ISearchAssistantOC.
	Get_ShowFindPrinter (see page 19318)	This is Get_ShowFindPrinter, a member of class ISearchAssistantOC2.
	Get_Version (see page 19311)	This is Get_Version, a member of class ISearchAssistantOC.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetProperty (see page 19311)	This is GetProperty, a member of class ISearchAssistantOC.
	GetSearchAssistantURL (see page 19311)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	IsRestricted (see page 19311)	This is IsRestricted, a member of class ISearchAssistantOC.
	NavigateToDefaultSearch (see page 19312)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC.
	NETDetectNextNavigate (see page 19312)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC.
	NotifySearchSettingsChanged (see page 19312)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC.
	PutFindText (see page 19312)	This is PutFindText, a member of class ISearchAssistantOC.
	PutProperty (see page 19313)	This is PutProperty, a member of class ISearchAssistantOC.
	ResetNextMenu (see page 19313)	This is ResetNextMenu, a member of class ISearchAssistantOC.
	Set_ASProvider (see page 19313)	This is Set_ASProvider, a member of class ISearchAssistantOC.
	Set_ASSetting (see page 19313)	This is Set_ASSetting, a member of class ISearchAssistantOC.
	Set_EventHandled (see page 19313)	This is Set_EventHandled, a member of class ISearchAssistantOC.
	SetDefaultSearchUrl (see page 19314)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC.

Public Properties

	Name	Description
	ASProvider (see page 19314)	This is ASProvider, a member of class ISearchAssistantOC.
	ASSetting (see page 19314)	This is ASSetting, a member of class ISearchAssistantOC.
W	EventHandled (see page 19315)	This is EventHandled, a member of class ISearchAssistantOC.
R	InWebFolder (see page 19315)	This is InWebFolder, a member of class ISearchAssistantOC.
R	SearchAssistantDefault (see page 19315)	This is SearchAssistantDefault, a member of class ISearchAssistantOC.
R	Searches (see page 19315)	This is Searches, a member of class ISearchAssistantOC.
R	ShellFeaturesEnabled (see page 19316)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC.
R	ShowFindPrinter (see page 19318)	This is ShowFindPrinter, a member of class ISearchAssistantOC2.
R	Version (see page 19316)	This is Version, a member of class ISearchAssistantOC.

132.2.10.2 ISearchAssistantOC2 Methods

The methods of the ISearchAssistantOC2 class are listed here.

Public Methods

	Name	Description
	Get_ShowFindPrinter (see page 19318)	This is Get_ShowFindPrinter, a member of class ISearchAssistantOC2.

132.2.10.2.1 ISearchAssistantOC2.Get_ShowFindPrinter Method

Pascal

```
function Get_ShowFindPrinter: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_ShowFindPrinter();
```

Description

This is Get_ShowFindPrinter, a member of class ISearchAssistantOC2.

132.2.10.3 ISearchAssistantOC2 Properties

The properties of the ISearchAssistantOC2 class are listed here.

Public Properties

	Name	Description
R	ShowFindPrinter (see page 19318)	This is ShowFindPrinter, a member of class ISearchAssistantOC2.

132.2.10.3.1 ISearchAssistantOC2.ShowFindPrinter Property

Pascal

```
property ShowFindPrinter: WordBool;
```

C++

```
__property WordBool ShowFindPrinter;
```

Description

This is ShowFindPrinter, a member of class ISearchAssistantOC2.

132.2.11 ISearchAssistantOC2Disp Interface

Class Hierarchy

SHDocVw.ISearchAssistantOC2Disp

Pascal

```
ISearchAssistantOC2Disp = dispinterface;
```

C++

```
class ISearchAssistantOC2Disp;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchAssistantOC2Disp.

132.2.11.1 ISearchAssistantOC2Disp Members

The following tables list the members exposed by ISearchAssistantOC2Disp.

Public Methods

	Name	Description
	AddNextMenuItem (see page 19320)	This is AddNextMenuItem, a member of class ISearchAssistantOC2Disp.
	EncodeString (see page 19320)	This is EncodeString, a member of class ISearchAssistantOC2Disp.
	FindComputer (see page 19320)	This is FindComputer, a member of class ISearchAssistantOC2Disp.
	FindFilesOrFolders (see page 19320)	This is FindFilesOrFolders, a member of class ISearchAssistantOC2Disp.
	FindOnWeb (see page 19321)	This is FindOnWeb, a member of class ISearchAssistantOC2Disp.
	FindPeople (see page 19321)	This is FindPeople, a member of class ISearchAssistantOC2Disp.
	FindPrinter (see page 19321)	This is FindPrinter, a member of class ISearchAssistantOC2Disp.
	GetProperty (see page 19321)	This is GetProperty, a member of class ISearchAssistantOC2Disp.
	GetSearchAssistantURL (see page 19322)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC2Disp.
	IsRestricted (see page 19322)	This is IsRestricted, a member of class ISearchAssistantOC2Disp.
	NavigateToDefaultSearch (see page 19322)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC2Disp.
	NETDetectNextNavigate (see page 19322)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC2Disp.
	NotifySearchSettingsChanged (see page 19322)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC2Disp.
	PutFindText (see page 19323)	This is PutFindText, a member of class ISearchAssistantOC2Disp.
	PutProperty (see page 19323)	This is PutProperty, a member of class ISearchAssistantOC2Disp.
	ResetNextMenu (see page 19323)	This is ResetNextMenu, a member of class ISearchAssistantOC2Disp.
	SetDefaultSearchUrl (see page 19323)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC2Disp.

Public Properties

	Name	Description
	ASProvider (see page 19324)	This is ASProvider, a member of class ISearchAssistantOC2Disp.
	ASSetting (see page 19324)	This is ASSetting, a member of class ISearchAssistantOC2Disp.
	EventHandled (see page 19324)	This is EventHandled, a member of class ISearchAssistantOC2Disp.
	InWebFolder (see page 19325)	This is InWebFolder, a member of class ISearchAssistantOC2Disp.
	SearchAssistantDefault (see page 19325)	This is SearchAssistantDefault, a member of class ISearchAssistantOC2Disp.
	Searches (see page 19325)	This is Searches, a member of class ISearchAssistantOC2Disp.
	ShellFeaturesEnabled (see page 19325)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC2Disp.
	ShowFindPrinter (see page 19325)	This is ShowFindPrinter, a member of class ISearchAssistantOC2Disp.
	Version (see page 19326)	This is Version, a member of class ISearchAssistantOC2Disp.

132.2.11.2 ISearchAssistantOC2Disp Methods

The methods of the ISearchAssistantOC2Disp class are listed here.

Public Methods

	Name	Description
	AddNextMenuItem (see page 19320)	This is AddNextMenuItem, a member of class ISearchAssistantOC2Disp.
	EncodeString (see page 19320)	This is EncodeString, a member of class ISearchAssistantOC2Disp.
	FindComputer (see page 19320)	This is FindComputer, a member of class ISearchAssistantOC2Disp.
	FindFilesOrFolders (see page 19320)	This is FindFilesOrFolders, a member of class ISearchAssistantOC2Disp.
	FindOnWeb (see page 19321)	This is FindOnWeb, a member of class ISearchAssistantOC2Disp.
	FindPeople (see page 19321)	This is FindPeople, a member of class ISearchAssistantOC2Disp.
	FindPrinter (see page 19321)	This is FindPrinter, a member of class ISearchAssistantOC2Disp.
	GetProperty (see page 19321)	This is GetProperty, a member of class ISearchAssistantOC2Disp.
	GetSearchAssistantURL (see page 19322)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC2Disp.

	IsRestricted (see page 19322)	This is IsRestricted, a member of class ISearchAssistantOC2Disp.
	NavigateToDefaultSearch (see page 19322)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC2Disp.
	NETDetectNextNavigate (see page 19322)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC2Disp.
	NotifySearchSettingsChanged (see page 19322)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC2Disp.
	PutFindText (see page 19323)	This is PutFindText, a member of class ISearchAssistantOC2Disp.
	PutProperty (see page 19323)	This is PutProperty, a member of class ISearchAssistantOC2Disp.
	ResetNextMenu (see page 19323)	This is ResetNextMenu, a member of class ISearchAssistantOC2Disp.
	SetDefaultSearchUrl (see page 19323)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC2Disp.

132.2.11.2.1 ISearchAssistantOC2Disp.AddNextMenuItem Method

Pascal

```
procedure AddNextMenuItem(const bstrText: WideString; idItem: Integer);
```

C++

```
__fastcall AddNextMenuItem(const BSTR bstrText, int idItem);
```

Description

This is AddNextMenuItem, a member of class ISearchAssistantOC2Disp.

132.2.11.2.2 ISearchAssistantOC2Disp.EncodeString Method

Pascal

```
function EncodeString(const bstrValue: WideString; const bstrCharSet: WideString; bUseUTF8:
WordBool): WideString;
```

C++

```
__fastcall BSTR EncodeString(const BSTR bstrValue, const BSTR bstrCharSet, WordBool bUseUTF8);
```

Description

This is EncodeString, a member of class ISearchAssistantOC2Disp.

132.2.11.2.3 ISearchAssistantOC2Disp.FindComputer Method

Pascal

```
procedure FindComputer;
```

C++

```
__fastcall FindComputer();
```

Description

This is FindComputer, a member of class ISearchAssistantOC2Disp.

132.2.11.2.4 ISearchAssistantOC2Disp.FindFilesOrFolders Method

Pascal

```
procedure FindFilesOrFolders;
```

C++

```
__fastcall FindFilesOrFolders();
```

Description

This is FindFilesOrFolders, a member of class ISearchAssistantOC2Disp.

132.2.11.2.5 ISearchAssistantOC2Disp.FindOnWeb Method

Pascal

```
procedure FindOnWeb;
```

C++

```
__fastcall FindOnWeb();
```

Description

This is FindOnWeb, a member of class ISearchAssistantOC2Disp.

132.2.11.2.6 ISearchAssistantOC2Disp.FindPeople Method

Pascal

```
procedure FindPeople;
```

C++

```
__fastcall FindPeople();
```

Description

This is FindPeople, a member of class ISearchAssistantOC2Disp.

132.2.11.2.7 ISearchAssistantOC2Disp.FindPrinter Method

Pascal

```
procedure FindPrinter;
```

C++

```
__fastcall FindPrinter();
```

Description

This is FindPrinter, a member of class ISearchAssistantOC2Disp.

132.2.11.2.8 ISearchAssistantOC2Disp.GetProperty Method

Pascal

```
function GetProperty(bPerLocale: WordBool; const bstrName: WideString): WideString;
```

C++

```
__fastcall BSTR GetProperty(WordBool bPerLocale, const BSTR bstrName);
```

Description

This is GetProperty, a member of class ISearchAssistantOC2Disp.

132.2.11.2.9 ISearchAssistantOC2Disp.GetSearchAssistantURL Method

Pascal

```
function GetSearchAssistantURL(bSubstitute: WordBool; bCustomize: WordBool): WideString;
```

C++

```
__fastcall BSTR GetSearchAssistantURL(WordBool bSubstitute, WordBool bCustomize);
```

Description

This is GetSearchAssistantURL, a member of class ISearchAssistantOC2Disp.

132.2.11.2.10 ISearchAssistantOC2Disp.IsRestricted Method

Pascal

```
function IsRestricted(const bstrGuid: WideString): WordBool;
```

C++

```
__fastcall WordBool IsRestricted(const BSTR bstrGuid);
```

Description

This is IsRestricted, a member of class ISearchAssistantOC2Disp.

132.2.11.2.11 ISearchAssistantOC2Disp.NavigateToDefaultSearch Method

Pascal

```
procedure NavigateToDefaultSearch;
```

C++

```
__fastcall NavigateToDefaultSearch();
```

Description

This is NavigateToDefaultSearch, a member of class ISearchAssistantOC2Disp.

132.2.11.2.12 ISearchAssistantOC2Disp.NETDetectNextNavigate Method

Pascal

```
procedure NETDetectNextNavigate;
```

C++

```
__fastcall NETDetectNextNavigate();
```

Description

This is NETDetectNextNavigate, a member of class ISearchAssistantOC2Disp.

132.2.11.2.13 ISearchAssistantOC2Disp.NotifySearchSettingsChanged Method

Pascal

```
procedure NotifySearchSettingsChanged;
```

C++

```
__fastcall NotifySearchSettingsChanged();
```

Description

This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC2Disp.

132.2.11.2.14 ISearchAssistantOC2Disp.PutFindText Method

Pascal

```
procedure PutFindText(const FindText: WideString);
```

C++

```
__fastcall PutFindText(const BSTR FindText);
```

Description

This is PutFindText, a member of class ISearchAssistantOC2Disp.

132.2.11.2.15 ISearchAssistantOC2Disp.PutProperty Method

Pascal

```
procedure PutProperty(bPerLocale: WordBool; const bstrName: WideString; const bstrValue:
WideString);
```

C++

```
__fastcall PutProperty(WordBool bPerLocale, const BSTR bstrName, const BSTR bstrValue);
```

Description

This is PutProperty, a member of class ISearchAssistantOC2Disp.

132.2.11.2.16 ISearchAssistantOC2Disp.ResetNextMenu Method

Pascal

```
procedure ResetNextMenu;
```

C++

```
__fastcall ResetNextMenu();
```

Description

This is ResetNextMenu, a member of class ISearchAssistantOC2Disp.

132.2.11.2.17 ISearchAssistantOC2Disp.SetDefaultSearchUrl Method

Pascal

```
procedure SetDefaultSearchUrl(const bstrUrl: WideString);
```

C++

```
__fastcall SetDefaultSearchUrl(const BSTR bstrUrl);
```

Description

This is SetDefaultSearchUrl, a member of class ISearchAssistantOC2Disp.

132.2.11.3 ISearchAssistantOC2Disp Properties

The properties of the ISearchAssistantOC2Disp class are listed here.

Public Properties

	Name	Description
	ASProvider (see page 19324)	This is ASProvider, a member of class ISearchAssistantOC2Disp.
	ASSetting (see page 19324)	This is ASSetting, a member of class ISearchAssistantOC2Disp.
	EventHandled (see page 19324)	This is EventHandled, a member of class ISearchAssistantOC2Disp.
	InWebFolder (see page 19325)	This is InWebFolder, a member of class ISearchAssistantOC2Disp.
	SearchAssistantDefault (see page 19325)	This is SearchAssistantDefault, a member of class ISearchAssistantOC2Disp.
	Searches (see page 19325)	This is Searches, a member of class ISearchAssistantOC2Disp.
	ShellFeaturesEnabled (see page 19325)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC2Disp.
	ShowFindPrinter (see page 19325)	This is ShowFindPrinter, a member of class ISearchAssistantOC2Disp.
	Version (see page 19326)	This is Version, a member of class ISearchAssistantOC2Disp.

132.2.11.3.1 ISearchAssistantOC2Disp.ASProvider Property

Pascal

```
property ASProvider: WideString;
```

C++

```
__property BSTR ASProvider;
```

Description

This is ASProvider, a member of class ISearchAssistantOC2Disp.

132.2.11.3.2 ISearchAssistantOC2Disp.ASSetting Property

Pascal

```
property ASSetting: SYSINT;
```

C++

```
__property SYSINT ASSetting;
```

Description

This is ASSetting, a member of class ISearchAssistantOC2Disp.

132.2.11.3.3 ISearchAssistantOC2Disp.EventHandled Property

Pascal

```
property EventHandled: WordBool;
```

C++

```
__property WordBool EventHandled;
```

Description

This is EventHandled, a member of class ISearchAssistantOC2Disp.

132.2.11.3.4 ISearchAssistantOC2Disp.InWebFolder Property

Pascal

```
property InWebFolder: WordBool;
```

C++

```
__property WordBool InWebFolder;
```

Description

This is InWebFolder, a member of class ISearchAssistantOC2Disp.

132.2.11.3.5 ISearchAssistantOC2Disp.SearchAssistantDefault Property

Pascal

```
property SearchAssistantDefault: WordBool;
```

C++

```
__property WordBool SearchAssistantDefault;
```

Description

This is SearchAssistantDefault, a member of class ISearchAssistantOC2Disp.

132.2.11.3.6 ISearchAssistantOC2Disp.Searches Property

Pascal

```
property Searches: ISearches;
```

C++

```
__property ISearches Searches;
```

Description

This is Searches, a member of class ISearchAssistantOC2Disp.

132.2.11.3.7 ISearchAssistantOC2Disp.ShellFeaturesEnabled Property

Pascal

```
property ShellFeaturesEnabled: WordBool;
```

C++

```
__property WordBool ShellFeaturesEnabled;
```

Description

This is ShellFeaturesEnabled, a member of class ISearchAssistantOC2Disp.

132.2.11.3.8 ISearchAssistantOC2Disp.ShowFindPrinter Property

Pascal

```
property ShowFindPrinter: WordBool;
```

C++

```
__property WordBool ShowFindPrinter;
```

Description

This is ShowFindPrinter, a member of class ISearchAssistantOC2Disp.

132.2.11.3.9 ISearchAssistantOC2Disp.Version Property

Pascal

```
property Version: SYSINT;
```

C++

```
__property SYSINT Version;
```

Description

This is Version, a member of class ISearchAssistantOC2Disp.

132.2.12 ISearchAssistantOC3 Interface

Class Hierarchy

Pascal

```
ISearchAssistantOC3 = interface(ISearchAssistantOC2);
```

C++

```
__interface ISearchAssistantOC3 : public ISearchAssistantOC2;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchAssistantOC3.

132.2.12.1 ISearchAssistantOC3 Members

The following tables list the members exposed by ISearchAssistantOC3.

Public Methods

	Name	Description
	AddNextMenuItem (see page 19308)	This is AddNextMenuItem, a member of class ISearchAssistantOC.
	EncodeString (see page 19308)	This is EncodeString, a member of class ISearchAssistantOC.
	FindComputer (see page 19308)	This is FindComputer, a member of class ISearchAssistantOC.
	FindFilesOrFolders (see page 19309)	This is FindFilesOrFolders, a member of class ISearchAssistantOC.
	FindOnWeb (see page 19309)	This is FindOnWeb, a member of class ISearchAssistantOC.
	FindPeople (see page 19309)	This is FindPeople, a member of class ISearchAssistantOC.
	FindPrinter (see page 19309)	This is FindPrinter, a member of class ISearchAssistantOC.
	Get_ASProvider (see page 19309)	This is Get_ASProvider, a member of class ISearchAssistantOC.
	Get_ASSetting (see page 19310)	This is Get_ASSetting, a member of class ISearchAssistantOC.
	Get_InWebFolder (see page 19310)	This is Get_InWebFolder, a member of class ISearchAssistantOC.
	Get_SearchAssistantDefault (see page 19310)	This is Get_SearchAssistantDefault, a member of class ISearchAssistantOC.

	Get_SearchCompanionAvailable (see page 19328)	This is Get_SearchCompanionAvailable, a member of class ISearchAssistantOC3.
	Get_Searches (see page 19310)	This is Get_Searches, a member of class ISearchAssistantOC.
	Get_ShellFeaturesEnabled (see page 19311)	This is Get_ShellFeaturesEnabled, a member of class ISearchAssistantOC.
	Get_ShowFindPrinter (see page 19318)	This is Get_ShowFindPrinter, a member of class ISearchAssistantOC2.
	Get_UseSearchCompanion (see page 19328)	This is Get_UseSearchCompanion, a member of class ISearchAssistantOC3.
	Get_Version (see page 19311)	This is Get_Version, a member of class ISearchAssistantOC.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetProperty (see page 19311)	This is GetProperty, a member of class ISearchAssistantOC.
	GetSearchAssistantURL (see page 19311)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	IsRestricted (see page 19311)	This is IsRestricted, a member of class ISearchAssistantOC.
	NavigateToDefaultSearch (see page 19312)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC.
	NETDetectNextNavigate (see page 19312)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC.
	NotifySearchSettingsChanged (see page 19312)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC.
	PutFindText (see page 19312)	This is PutFindText, a member of class ISearchAssistantOC.
	PutProperty (see page 19313)	This is PutProperty, a member of class ISearchAssistantOC.
	ResetNextMenu (see page 19313)	This is ResetNextMenu, a member of class ISearchAssistantOC.
	Set_ASProvider (see page 19313)	This is Set_ASProvider, a member of class ISearchAssistantOC.
	Set_ASSetting (see page 19313)	This is Set_ASSetting, a member of class ISearchAssistantOC.
	Set_EventHandled (see page 19313)	This is Set_EventHandled, a member of class ISearchAssistantOC.
	Set_UseSearchCompanion (see page 19328)	This is Set_UseSearchCompanion, a member of class ISearchAssistantOC3.
	SetDefaultSearchUrl (see page 19314)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC.

Public Properties

	Name	Description
	ASProvider (see page 19314)	This is ASProvider, a member of class ISearchAssistantOC.
	ASSetting (see page 19314)	This is ASSetting, a member of class ISearchAssistantOC.
W	EventHandled (see page 19315)	This is EventHandled, a member of class ISearchAssistantOC.
R	InWebFolder (see page 19315)	This is InWebFolder, a member of class ISearchAssistantOC.
R	SearchAssistantDefault (see page 19315)	This is SearchAssistantDefault, a member of class ISearchAssistantOC.
R	SearchCompanionAvailable (see page 19328)	This is SearchCompanionAvailable, a member of class ISearchAssistantOC3.
R	Searches (see page 19315)	This is Searches, a member of class ISearchAssistantOC.
R	ShellFeaturesEnabled (see page 19316)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC.
R	ShowFindPrinter (see page 19318)	This is ShowFindPrinter, a member of class ISearchAssistantOC2.
	UseSearchCompanion (see page 19329)	This is UseSearchCompanion, a member of class ISearchAssistantOC3.
R	Version (see page 19316)	This is Version, a member of class ISearchAssistantOC.

132.2.12.2 ISearchAssistantOC3 Methods

The methods of the ISearchAssistantOC3 class are listed here.

Public Methods

	Name	Description
	Get_SearchCompanionAvailable (see page 19328)	This is Get_SearchCompanionAvailable, a member of class ISearchAssistantOC3.
	Get_UseSearchCompanion (see page 19328)	This is Get_UseSearchCompanion, a member of class ISearchAssistantOC3.
	Set_UseSearchCompanion (see page 19328)	This is Set_UseSearchCompanion, a member of class ISearchAssistantOC3.

132.2.12.2.1 ISearchAssistantOC3.Get_SearchCompanionAvailable Method

Pascal

```
function Get_SearchCompanionAvailable: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_SearchCompanionAvailable();
```

Description

This is Get_SearchCompanionAvailable, a member of class ISearchAssistantOC3.

132.2.12.2.2 ISearchAssistantOC3.Get_UseSearchCompanion Method

Pascal

```
function Get_UseSearchCompanion: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_UseSearchCompanion();
```

Description

This is Get_UseSearchCompanion, a member of class ISearchAssistantOC3.

132.2.12.2.3 ISearchAssistantOC3.Set_UseSearchCompanion Method

Pascal

```
procedure Set_UseSearchCompanion(pbUseSC: WordBool); safecall;
```

C++

```
__fastcall Set_UseSearchCompanion(WordBool pbUseSC);
```

Description

This is Set_UseSearchCompanion, a member of class ISearchAssistantOC3.

132.2.12.3 ISearchAssistantOC3 Properties

The properties of the ISearchAssistantOC3 class are listed here.

Public Properties

	Name	Description
R	SearchCompanionAvailable (see page 19328)	This is SearchCompanionAvailable, a member of class ISearchAssistantOC3.
	UseSearchCompanion (see page 19329)	This is UseSearchCompanion, a member of class ISearchAssistantOC3.

132.2.12.3.1 ISearchAssistantOC3.SearchCompanionAvailable Property

Pascal

```
property SearchCompanionAvailable: WordBool;
```

C++

```
__property WordBool SearchCompanionAvailable;
```

Description

This is SearchCompanionAvailable, a member of class ISearchAssistantOC3.

132.2.12.3.2 ISearchAssistantOC3.UseSearchCompanion Property

Pascal

```
property UseSearchCompanion: WordBool;
```

C++

```
__property WordBool UseSearchCompanion;
```

Description

This is UseSearchCompanion, a member of class ISearchAssistantOC3.

132.2.13 ISearchAssistantOC3Disp Interface

Class Hierarchy

SHDocVw.ISearchAssistantOC3Disp

Pascal

```
ISearchAssistantOC3Disp = dispinterface;
```

C++

```
class ISearchAssistantOC3Disp;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchAssistantOC3Disp.

132.2.13.1 ISearchAssistantOC3Disp Members

The following tables list the members exposed by ISearchAssistantOC3Disp.

Public Methods

	Name	Description
	AddNextMenuItem (see page 19330)	This is AddNextMenuItem, a member of class ISearchAssistantOC3Disp.
	EncodeString (see page 19331)	This is EncodeString, a member of class ISearchAssistantOC3Disp.
	FindComputer (see page 19331)	This is FindComputer, a member of class ISearchAssistantOC3Disp.
	FindFilesOrFolders (see page 19331)	This is FindFilesOrFolders, a member of class ISearchAssistantOC3Disp.
	FindOnWeb (see page 19331)	This is FindOnWeb, a member of class ISearchAssistantOC3Disp.
	FindPeople (see page 19332)	This is FindPeople, a member of class ISearchAssistantOC3Disp.
	FindPrinter (see page 19332)	This is FindPrinter, a member of class ISearchAssistantOC3Disp.
	GetProperty (see page 19332)	This is GetProperty, a member of class ISearchAssistantOC3Disp.
	GetSearchAssistantURL (see page 19332)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC3Disp.
	IsRestricted (see page 19332)	This is IsRestricted, a member of class ISearchAssistantOC3Disp.
	NavigateToDefaultSearch (see page 19333)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC3Disp.

	NETDetectNextNavigate (see page 19333)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC3Disp.
	NotifySearchSettingsChanged (see page 19333)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC3Disp.
	PutFindText (see page 19333)	This is PutFindText, a member of class ISearchAssistantOC3Disp.
	PutProperty (see page 19334)	This is PutProperty, a member of class ISearchAssistantOC3Disp.
	ResetNextMenu (see page 19334)	This is ResetNextMenu, a member of class ISearchAssistantOC3Disp.
	SetDefaultSearchUrl (see page 19334)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC3Disp.

Public Properties

	Name	Description
	ASProvider (see page 19335)	This is ASProvider, a member of class ISearchAssistantOC3Disp.
	ASSetting (see page 19335)	This is ASSetting, a member of class ISearchAssistantOC3Disp.
	EventHandled (see page 19335)	This is EventHandled, a member of class ISearchAssistantOC3Disp.
	InWebFolder (see page 19335)	This is InWebFolder, a member of class ISearchAssistantOC3Disp.
	SearchAssistantDefault (see page 19335)	This is SearchAssistantDefault, a member of class ISearchAssistantOC3Disp.
	SearchCompanionAvailable (see page 19336)	This is SearchCompanionAvailable, a member of class ISearchAssistantOC3Disp.
	Searches (see page 19336)	This is Searches, a member of class ISearchAssistantOC3Disp.
	ShellFeaturesEnabled (see page 19336)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC3Disp.
	ShowFindPrinter (see page 19336)	This is ShowFindPrinter, a member of class ISearchAssistantOC3Disp.
	UseSearchCompanion (see page 19337)	This is UseSearchCompanion, a member of class ISearchAssistantOC3Disp.
	Version (see page 19337)	This is Version, a member of class ISearchAssistantOC3Disp.

132.2.13.2 ISearchAssistantOC3Disp Methods

The methods of the ISearchAssistantOC3Disp class are listed here.

Public Methods

	Name	Description
	AddNextMenuItem (see page 19330)	This is AddNextMenuItem, a member of class ISearchAssistantOC3Disp.
	EncodeString (see page 19331)	This is EncodeString, a member of class ISearchAssistantOC3Disp.
	FindComputer (see page 19331)	This is FindComputer, a member of class ISearchAssistantOC3Disp.
	FindFilesOrFolders (see page 19331)	This is FindFilesOrFolders, a member of class ISearchAssistantOC3Disp.
	FindOnWeb (see page 19331)	This is FindOnWeb, a member of class ISearchAssistantOC3Disp.
	FindPeople (see page 19332)	This is FindPeople, a member of class ISearchAssistantOC3Disp.
	FindPrinter (see page 19332)	This is FindPrinter, a member of class ISearchAssistantOC3Disp.
	GetProperty (see page 19332)	This is GetProperty, a member of class ISearchAssistantOC3Disp.
	GetSearchAssistantURL (see page 19332)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC3Disp.
	IsRestricted (see page 19332)	This is IsRestricted, a member of class ISearchAssistantOC3Disp.
	NavigateToDefaultSearch (see page 19333)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC3Disp.
	NETDetectNextNavigate (see page 19333)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC3Disp.
	NotifySearchSettingsChanged (see page 19333)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC3Disp.
	PutFindText (see page 19333)	This is PutFindText, a member of class ISearchAssistantOC3Disp.
	PutProperty (see page 19334)	This is PutProperty, a member of class ISearchAssistantOC3Disp.
	ResetNextMenu (see page 19334)	This is ResetNextMenu, a member of class ISearchAssistantOC3Disp.
	SetDefaultSearchUrl (see page 19334)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC3Disp.

132.2.13.2.1 ISearchAssistantOC3Disp.AddNextMenuItem Method

Pascal

```
procedure AddNextMenuItem(const bstrText: WideString; idItem: Integer);
```

C++

```
__fastcall AddNextMenuItem(const BSTR bstrText, int idItem);
```

Description

This is AddNextMenuItem, a member of class ISearchAssistantOC3Disp.

132.2.13.2.2 ISearchAssistantOC3Disp.EncodeString Method

Pascal

```
function EncodeString(const bstrValue: WideString; const bstrCharSet: WideString; bUseUTF8:
WordBool): WideString;
```

C++

```
__fastcall BSTR EncodeString(const BSTR bstrValue, const BSTR bstrCharSet, WordBool bUseUTF8);
```

Description

This is EncodeString, a member of class ISearchAssistantOC3Disp.

132.2.13.2.3 ISearchAssistantOC3Disp.FindComputer Method

Pascal

```
procedure FindComputer;
```

C++

```
__fastcall FindComputer();
```

Description

This is FindComputer, a member of class ISearchAssistantOC3Disp.

132.2.13.2.4 ISearchAssistantOC3Disp.FindFilesOrFolders Method

Pascal

```
procedure FindFilesOrFolders;
```

C++

```
__fastcall FindFilesOrFolders();
```

Description

This is FindFilesOrFolders, a member of class ISearchAssistantOC3Disp.

132.2.13.2.5 ISearchAssistantOC3Disp.FindOnWeb Method

Pascal

```
procedure FindOnWeb;
```

C++

```
__fastcall FindOnWeb();
```

Description

This is FindOnWeb, a member of class ISearchAssistantOC3Disp.

132.2.13.2.6 ISearchAssistantOC3Disp.FindPeople Method

Pascal

```
procedure FindPeople;
```

C++

```
__fastcall FindPeople();
```

Description

This is FindPeople, a member of class ISearchAssistantOC3Disp.

132.2.13.2.7 ISearchAssistantOC3Disp.FindPrinter Method

Pascal

```
procedure FindPrinter;
```

C++

```
__fastcall FindPrinter();
```

Description

This is FindPrinter, a member of class ISearchAssistantOC3Disp.

132.2.13.2.8 ISearchAssistantOC3Disp.GetProperty Method

Pascal

```
function GetProperty(bPerLocale: WordBool; const bstrName: WideString): WideString;
```

C++

```
__fastcall BSTR GetProperty(WordBool bPerLocale, const BSTR bstrName);
```

Description

This is GetProperty, a member of class ISearchAssistantOC3Disp.

132.2.13.2.9 ISearchAssistantOC3Disp.GetSearchAssistantURL Method

Pascal

```
function GetSearchAssistantURL(bSubstitute: WordBool; bCustomize: WordBool): WideString;
```

C++

```
__fastcall BSTR GetSearchAssistantURL(WordBool bSubstitute, WordBool bCustomize);
```

Description

This is GetSearchAssistantURL, a member of class ISearchAssistantOC3Disp.

132.2.13.2.10 ISearchAssistantOC3Disp.IsRestricted Method

Pascal

```
function IsRestricted(const bstrGuid: WideString): WordBool;
```

C++

```
__fastcall WordBool IsRestricted(const BSTR bstrGuid);
```

Description

This is IsRestricted, a member of class ISearchAssistantOC3Disp.

132.2.13.2.11 ISearchAssistantOC3Disp.NavigateToDefaultSearch Method

Pascal

```
procedure NavigateToDefaultSearch;
```

C++

```
__fastcall NavigateToDefaultSearch();
```

Description

This is NavigateToDefaultSearch, a member of class ISearchAssistantOC3Disp.

132.2.13.2.12 ISearchAssistantOC3Disp.NETDetectNextNavigate Method

Pascal

```
procedure NETDetectNextNavigate;
```

C++

```
__fastcall NETDetectNextNavigate();
```

Description

This is NETDetectNextNavigate, a member of class ISearchAssistantOC3Disp.

132.2.13.2.13 ISearchAssistantOC3Disp.NotifySearchSettingsChanged Method

Pascal

```
procedure NotifySearchSettingsChanged;
```

C++

```
__fastcall NotifySearchSettingsChanged();
```

Description

This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC3Disp.

132.2.13.2.14 ISearchAssistantOC3Disp.PutFindText Method

Pascal

```
procedure PutFindText(const FindText: WideString);
```

C++

```
__fastcall PutFindText(const BSTR FindText);
```

Description

This is PutFindText, a member of class ISearchAssistantOC3Disp.

132.2.13.2.15 ISearchAssistantOC3Disp.PutProperty Method

Pascal

```
procedure PutProperty(bPerLocale: WordBool; const bstrName: WideString; const bstrValue:
WideString);
```

C++

```
__fastcall PutProperty(WordBool bPerLocale, const BSTR bstrName, const BSTR bstrValue);
```

Description

This is PutProperty, a member of class ISearchAssistantOC3Disp.

132.2.13.2.16 ISearchAssistantOC3Disp.ResetNextMenu Method

Pascal

```
procedure ResetNextMenu;
```

C++

```
__fastcall ResetNextMenu();
```

Description

This is ResetNextMenu, a member of class ISearchAssistantOC3Disp.

132.2.13.2.17 ISearchAssistantOC3Disp.SetDefaultSearchUrl Method

Pascal

```
procedure SetDefaultSearchUrl(const bstrUrl: WideString);
```

C++

```
__fastcall SetDefaultSearchUrl(const BSTR bstrUrl);
```

Description

This is SetDefaultSearchUrl, a member of class ISearchAssistantOC3Disp.

132.2.13.3 ISearchAssistantOC3Disp Properties

The properties of the ISearchAssistantOC3Disp class are listed here.

Public Properties

	Name	Description
	ASProvider (see page 19335)	This is ASProvider, a member of class ISearchAssistantOC3Disp.
	ASSetting (see page 19335)	This is ASSetting, a member of class ISearchAssistantOC3Disp.
	EventHandled (see page 19335)	This is EventHandled, a member of class ISearchAssistantOC3Disp.
	InWebFolder (see page 19335)	This is InWebFolder, a member of class ISearchAssistantOC3Disp.
	SearchAssistantDefault (see page 19335)	This is SearchAssistantDefault, a member of class ISearchAssistantOC3Disp.
	SearchCompanionAvailable (see page 19336)	This is SearchCompanionAvailable, a member of class ISearchAssistantOC3Disp.
	Searches (see page 19336)	This is Searches, a member of class ISearchAssistantOC3Disp.
	ShellFeaturesEnabled (see page 19336)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC3Disp.
	ShowFindPrinter (see page 19336)	This is ShowFindPrinter, a member of class ISearchAssistantOC3Disp.
	UseSearchCompanion (see page 19337)	This is UseSearchCompanion, a member of class ISearchAssistantOC3Disp.

	Version (see page 19337)	This is Version, a member of class ISearchAssistantOC3Disp.

132.2.13.3.1 ISearchAssistantOC3Disp.ASProvider Property

Pascal

```
property ASProvider: WideString;
```

C++

```
__property BSTR ASProvider;
```

Description

This is ASProvider, a member of class ISearchAssistantOC3Disp.

132.2.13.3.2 ISearchAssistantOC3Disp.ASSetting Property

Pascal

```
property ASSetting: SYSINT;
```

C++

```
__property SYSINT ASSetting;
```

Description

This is ASSetting, a member of class ISearchAssistantOC3Disp.

132.2.13.3.3 ISearchAssistantOC3Disp.EventHandled Property

Pascal

```
property EventHandled: WordBool;
```

C++

```
__property WordBool EventHandled;
```

Description

This is EventHandled, a member of class ISearchAssistantOC3Disp.

132.2.13.3.4 ISearchAssistantOC3Disp.InWebFolder Property

Pascal

```
property InWebFolder: WordBool;
```

C++

```
__property WordBool InWebFolder;
```

Description

This is InWebFolder, a member of class ISearchAssistantOC3Disp.

132.2.13.3.5 ISearchAssistantOC3Disp.SearchAssistantDefault Property

Pascal

```
property SearchAssistantDefault: WordBool;
```

C++

```
__property WordBool SearchAssistantDefault;
```

Description

This is SearchAssistantDefault, a member of class ISearchAssistantOC3Disp.

132.2.13.3.6 ISearchAssistantOC3Disp.SearchCompanionAvailable Property

Pascal

```
property SearchCompanionAvailable: WordBool;
```

C++

```
__property WordBool SearchCompanionAvailable;
```

Description

This is SearchCompanionAvailable, a member of class ISearchAssistantOC3Disp.

132.2.13.3.7 ISearchAssistantOC3Disp.Searches Property

Pascal

```
property Searches: ISearches;
```

C++

```
__property ISearches Searches;
```

Description

This is Searches, a member of class ISearchAssistantOC3Disp.

132.2.13.3.8 ISearchAssistantOC3Disp.ShellFeaturesEnabled Property

Pascal

```
property ShellFeaturesEnabled: WordBool;
```

C++

```
__property WordBool ShellFeaturesEnabled;
```

Description

This is ShellFeaturesEnabled, a member of class ISearchAssistantOC3Disp.

132.2.13.3.9 ISearchAssistantOC3Disp.ShowFindPrinter Property

Pascal

```
property ShowFindPrinter: WordBool;
```

C++

```
__property WordBool ShowFindPrinter;
```

Description

This is ShowFindPrinter, a member of class ISearchAssistantOC3Disp.

132.2.13.3.10 ISearchAssistantOC3Disp.UseSearchCompanion Property

Pascal

```
property UseSearchCompanion: WordBool;
```

C++

```
__property WordBool UseSearchCompanion;
```

Description

This is UseSearchCompanion, a member of class ISearchAssistantOC3Disp.

132.2.13.3.11 ISearchAssistantOC3Disp.Version Property

Pascal

```
property Version: SYSINT;
```

C++

```
__property SYSINT Version;
```

Description

This is Version, a member of class ISearchAssistantOC3Disp.

132.2.14 ISearchAssistantOCDisp Interface

Class Hierarchy

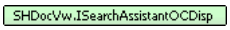

Pascal

```
ISearchAssistantOCDisp = dispinterface;
```

C++

```
class ISearchAssistantOCDisp;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchAssistantOCDisp.

132.2.14.1 ISearchAssistantOCDisp Members

The following tables list the members exposed by ISearchAssistantOCDisp.

Public Methods

	Name	Description
	AddNextMenuItem (see page 19339)	This is AddNextMenuItem, a member of class ISearchAssistantOCDisp.
	EncodeString (see page 19339)	This is EncodeString, a member of class ISearchAssistantOCDisp.
	FindComputer (see page 19339)	This is FindComputer, a member of class ISearchAssistantOCDisp.

	FindFilesOrFolders (see page 19339)	This is FindFilesOrFolders, a member of class ISearchAssistantOCDisp.
	FindOnWeb (see page 19339)	This is FindOnWeb, a member of class ISearchAssistantOCDisp.
	FindPeople (see page 19340)	This is FindPeople, a member of class ISearchAssistantOCDisp.
	FindPrinter (see page 19340)	This is FindPrinter, a member of class ISearchAssistantOCDisp.
	GetProperty (see page 19340)	This is GetProperty, a member of class ISearchAssistantOCDisp.
	GetSearchAssistantURL (see page 19340)	This is GetSearchAssistantURL, a member of class ISearchAssistantOCDisp.
	IsRestricted (see page 19341)	This is IsRestricted, a member of class ISearchAssistantOCDisp.
	NavigateToDefaultSearch (see page 19341)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOCDisp.
	NETDetectNextNavigate (see page 19341)	This is NETDetectNextNavigate, a member of class ISearchAssistantOCDisp.
	NotifySearchSettingsChanged (see page 19341)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOCDisp.
	PutFindText (see page 19341)	This is PutFindText, a member of class ISearchAssistantOCDisp.
	PutProperty (see page 19342)	This is PutProperty, a member of class ISearchAssistantOCDisp.
	ResetNextMenu (see page 19342)	This is ResetNextMenu, a member of class ISearchAssistantOCDisp.
	SetDefaultSearchUrl (see page 19342)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOCDisp.

Public Properties

	Name	Description
	ASProvider (see page 19343)	This is ASProvider, a member of class ISearchAssistantOCDisp.
	ASSetting (see page 19343)	This is ASSetting, a member of class ISearchAssistantOCDisp.
	EventHandled (see page 19343)	This is EventHandled, a member of class ISearchAssistantOCDisp.
	InWebFolder (see page 19343)	This is InWebFolder, a member of class ISearchAssistantOCDisp.
	SearchAssistantDefault (see page 19343)	This is SearchAssistantDefault, a member of class ISearchAssistantOCDisp.
	Searches (see page 19344)	This is Searches, a member of class ISearchAssistantOCDisp.
	ShellFeaturesEnabled (see page 19344)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOCDisp.
	Version (see page 19344)	This is Version, a member of class ISearchAssistantOCDisp.

132.2.14.2 ISearchAssistantOCDisp Methods

The methods of the ISearchAssistantOCDisp class are listed here.

Public Methods

	Name	Description
	AddNextMenuItem (see page 19339)	This is AddNextMenuItem, a member of class ISearchAssistantOCDisp.
	EncodeString (see page 19339)	This is EncodeString, a member of class ISearchAssistantOCDisp.
	FindComputer (see page 19339)	This is FindComputer, a member of class ISearchAssistantOCDisp.
	FindFilesOrFolders (see page 19339)	This is FindFilesOrFolders, a member of class ISearchAssistantOCDisp.
	FindOnWeb (see page 19339)	This is FindOnWeb, a member of class ISearchAssistantOCDisp.
	FindPeople (see page 19340)	This is FindPeople, a member of class ISearchAssistantOCDisp.
	FindPrinter (see page 19340)	This is FindPrinter, a member of class ISearchAssistantOCDisp.
	GetProperty (see page 19340)	This is GetProperty, a member of class ISearchAssistantOCDisp.
	GetSearchAssistantURL (see page 19340)	This is GetSearchAssistantURL, a member of class ISearchAssistantOCDisp.
	IsRestricted (see page 19341)	This is IsRestricted, a member of class ISearchAssistantOCDisp.
	NavigateToDefaultSearch (see page 19341)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOCDisp.
	NETDetectNextNavigate (see page 19341)	This is NETDetectNextNavigate, a member of class ISearchAssistantOCDisp.
	NotifySearchSettingsChanged (see page 19341)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOCDisp.
	PutFindText (see page 19341)	This is PutFindText, a member of class ISearchAssistantOCDisp.
	PutProperty (see page 19342)	This is PutProperty, a member of class ISearchAssistantOCDisp.
	ResetNextMenu (see page 19342)	This is ResetNextMenu, a member of class ISearchAssistantOCDisp.
	SetDefaultSearchUrl (see page 19342)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOCDisp.

132.2.14.2.1 ISearchAssistantOCDisp.AddNextMenuItem Method

Pascal

```
procedure AddNextMenuItem(const bstrText: WideString; idItem: Integer);
```

C++

```
__fastcall AddNextMenuItem(const BSTR bstrText, int idItem);
```

Description

This is AddNextMenuItem, a member of class ISearchAssistantOCDisp.

132.2.14.2.2 ISearchAssistantOCDisp.EncodeString Method

Pascal

```
function EncodeString(const bstrValue: WideString; const bstrCharSet: WideString; bUseUTF8:
WordBool): WideString;
```

C++

```
__fastcall BSTR EncodeString(const BSTR bstrValue, const BSTR bstrCharSet, WordBool bUseUTF8);
```

Description

This is EncodeString, a member of class ISearchAssistantOCDisp.

132.2.14.2.3 ISearchAssistantOCDisp.FindComputer Method

Pascal

```
procedure FindComputer;
```

C++

```
__fastcall FindComputer();
```

Description

This is FindComputer, a member of class ISearchAssistantOCDisp.

132.2.14.2.4 ISearchAssistantOCDisp.FindFilesOrFolders Method

Pascal

```
procedure FindFilesOrFolders;
```

C++

```
__fastcall FindFilesOrFolders();
```

Description

This is FindFilesOrFolders, a member of class ISearchAssistantOCDisp.

132.2.14.2.5 ISearchAssistantOCDisp.FindOnWeb Method

Pascal

```
procedure FindOnWeb;
```

C++

```
__fastcall FindOnWeb();
```

Description

This is FindOnWeb, a member of class ISearchAssistantOCDisp.

132.2.14.2.6 ISearchAssistantOCDisp.FindPeople Method

Pascal

```
procedure FindPeople;
```

C++

```
__fastcall FindPeople();
```

Description

This is FindPeople, a member of class ISearchAssistantOCDisp.

132.2.14.2.7 ISearchAssistantOCDisp.FindPrinter Method

Pascal

```
procedure FindPrinter;
```

C++

```
__fastcall FindPrinter();
```

Description

This is FindPrinter, a member of class ISearchAssistantOCDisp.

132.2.14.2.8 ISearchAssistantOCDisp.GetProperty Method

Pascal

```
function GetProperty(bPerLocale: WordBool; const bstrName: WideString): WideString;
```

C++

```
__fastcall BSTR GetProperty(WordBool bPerLocale, const BSTR bstrName);
```

Description

This is GetProperty, a member of class ISearchAssistantOCDisp.

132.2.14.2.9 ISearchAssistantOCDisp.GetSearchAssistantURL Method

Pascal

```
function GetSearchAssistantURL(bSubstitute: WordBool; bCustomize: WordBool): WideString;
```

C++

```
__fastcall BSTR GetSearchAssistantURL(WordBool bSubstitute, WordBool bCustomize);
```

Description

This is GetSearchAssistantURL, a member of class ISearchAssistantOCDisp.

132.2.14.2.10 ISearchAssistantOCDisp.IsRestricted Method

Pascal

```
function IsRestricted(const bstrGuid: WideString): WordBool;
```

C++

```
__fastcall WordBool IsRestricted(const BSTR bstrGuid);
```

Description

This is IsRestricted, a member of class ISearchAssistantOCDisp.

132.2.14.2.11 ISearchAssistantOCDisp.NavigateToDefaultSearch Method

Pascal

```
procedure NavigateToDefaultSearch;
```

C++

```
__fastcall NavigateToDefaultSearch();
```

Description

This is NavigateToDefaultSearch, a member of class ISearchAssistantOCDisp.

132.2.14.2.12 ISearchAssistantOCDisp.NETDetectNextNavigate Method

Pascal

```
procedure NETDetectNextNavigate;
```

C++

```
__fastcall NETDetectNextNavigate();
```

Description

This is NETDetectNextNavigate, a member of class ISearchAssistantOCDisp.

132.2.14.2.13 ISearchAssistantOCDisp.NotifySearchSettingsChanged Method

Pascal

```
procedure NotifySearchSettingsChanged;
```

C++

```
__fastcall NotifySearchSettingsChanged();
```

Description

This is NotifySearchSettingsChanged, a member of class ISearchAssistantOCDisp.

132.2.14.2.14 ISearchAssistantOCDisp.PutFindText Method

Pascal

```
procedure PutFindText(const FindText: WideString);
```

C++

```
__fastcall PutFindText(const BSTR FindText);
```

Description

This is PutFindText, a member of class ISearchAssistantOCDisp.

132.2.14.2.15 ISearchAssistantOCDisp.PutProperty Method

Pascal

```
procedure PutProperty(bPerLocale: WordBool; const bstrName: WideString; const bstrValue:
WideString);
```

C++

```
__fastcall PutProperty(WordBool bPerLocale, const BSTR bstrName, const BSTR bstrValue);
```

Description

This is PutProperty, a member of class ISearchAssistantOCDisp.

132.2.14.2.16 ISearchAssistantOCDisp.ResetNextMenu Method

Pascal

```
procedure ResetNextMenu;
```

C++

```
__fastcall ResetNextMenu();
```

Description

This is ResetNextMenu, a member of class ISearchAssistantOCDisp.

132.2.14.2.17 ISearchAssistantOCDisp.SetDefaultSearchUrl Method

Pascal

```
procedure SetDefaultSearchUrl(const bstrUrl: WideString);
```

C++

```
__fastcall SetDefaultSearchUrl(const BSTR bstrUrl);
```

Description

This is SetDefaultSearchUrl, a member of class ISearchAssistantOCDisp.

132.2.14.3 ISearchAssistantOCDisp Properties

The properties of the ISearchAssistantOCDisp class are listed here.

Public Properties

	Name	Description
	ASProvider (see page 19343)	This is ASProvider, a member of class ISearchAssistantOCDisp.
	ASSetting (see page 19343)	This is ASSetting, a member of class ISearchAssistantOCDisp.
	EventHandled (see page 19343)	This is EventHandled, a member of class ISearchAssistantOCDisp.
	InWebFolder (see page 19343)	This is InWebFolder, a member of class ISearchAssistantOCDisp.
	SearchAssistantDefault (see page 19343)	This is SearchAssistantDefault, a member of class ISearchAssistantOCDisp.
	Searches (see page 19344)	This is Searches, a member of class ISearchAssistantOCDisp.
	ShellFeaturesEnabled (see page 19344)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOCDisp.

	Version (see page 19344)	This is Version, a member of class ISearchAssistantOCDisp.

132.2.14.3.1 ISearchAssistantOCDisp.ASProvider Property

Pascal

```
property ASProvider: WideString;
```

C++

```
__property BSTR ASProvider;
```

Description

This is ASProvider, a member of class ISearchAssistantOCDisp.

132.2.14.3.2 ISearchAssistantOCDisp.ASSetting Property

Pascal

```
property ASSetting: SYSINT;
```

C++

```
__property SYSINT ASSetting;
```

Description

This is ASSetting, a member of class ISearchAssistantOCDisp.

132.2.14.3.3 ISearchAssistantOCDisp.EventHandled Property

Pascal

```
property EventHandled: WordBool;
```

C++

```
__property WordBool EventHandled;
```

Description

This is EventHandled, a member of class ISearchAssistantOCDisp.

132.2.14.3.4 ISearchAssistantOCDisp.InWebFolder Property

Pascal

```
property InWebFolder: WordBool;
```

C++

```
__property WordBool InWebFolder;
```

Description

This is InWebFolder, a member of class ISearchAssistantOCDisp.

132.2.14.3.5 ISearchAssistantOCDisp.SearchAssistantDefault Property

Pascal

```
property SearchAssistantDefault: WordBool;
```

C++

```
__property WordBool SearchAssistantDefault;
```

Description

This is SearchAssistantDefault, a member of class ISearchAssistantOCDisp.

132.2.14.3.6 ISearchAssistantOCDisp.Searches Property

Pascal

```
property Searches: ISearches;
```

C++

```
__property ISearches Searches;
```

Description

This is Searches, a member of class ISearchAssistantOCDisp.

132.2.14.3.7 ISearchAssistantOCDisp.ShellFeaturesEnabled Property

Pascal

```
property ShellFeaturesEnabled: WordBool;
```

C++

```
__property WordBool ShellFeaturesEnabled;
```

Description

This is ShellFeaturesEnabled, a member of class ISearchAssistantOCDisp.

132.2.14.3.8 ISearchAssistantOCDisp.Version Property

Pascal

```
property Version: SYSINT;
```

C++

```
__property SYSINT Version;
```

Description

This is Version, a member of class ISearchAssistantOCDisp.

132.2.15 ISearchDisp Interface

Class Hierarchy

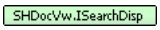

Pascal

```
ISearchDisp = dispinterface;
```

C++

```
class ISearchDisp;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchDisp.

132.2.15.1 ISearchDisp Members

The following tables list the members exposed by ISearchDisp.

Public Properties

	Name	Description
	Id (see page 19345)	This is Id, a member of class ISearchDisp.
	Title (see page 19345)	This is Title, a member of class ISearchDisp.
	URL (see page 19346)	This is URL, a member of class ISearchDisp.

132.2.15.2 ISearchDisp Properties

The properties of the ISearchDisp class are listed here.

Public Properties

	Name	Description
	Id (see page 19345)	This is Id, a member of class ISearchDisp.
	Title (see page 19345)	This is Title, a member of class ISearchDisp.
	URL (see page 19346)	This is URL, a member of class ISearchDisp.

132.2.15.2.1 ISearchDisp.Id Property

Pascal

```
property Id: WideString;
```

C++

```
__property BSTR Id;
```

Description

This is Id, a member of class ISearchDisp.

132.2.15.2.2 ISearchDisp.Title Property

Pascal

```
property Title: WideString;
```

C++

```
__property BSTR Title;
```

Description

This is Title, a member of class ISearchDisp.

132.2.15.2.3 ISearchDisp.URL Property

Pascal

```
property URL: WideString;
```

C++

```
__property BSTR URL;
```

Description

This is URL, a member of class ISearchDisp.

132.2.16 ISearches Interface

Class Hierarchy

Pascal

```
ISearches = interface(IDispatch);
```

C++

```
__interface ISearches : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.ISearches.

132.2.16.1 ISearches Members

The following tables list the members exposed by ISearches.

Public Methods

	Name	Description
	_NewEnum (see page 19347)	This is _NewEnum, a member of class ISearches.
	Get_Count (see page 19347)	This is Get_Count, a member of class ISearches.
	Get_Default (see page 19347)	This is Get_Default, a member of class ISearches.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	Item (see page 19347)	This is Item, a member of class ISearches.

Public Properties

	Name	Description
R	Count (see page 19348)	This is Count, a member of class ISearches.
R	Default (see page 19348)	This is Default, a member of class ISearches.

132.2.16.2 ISearches Methods

The methods of the ISearches class are listed here.

Public Methods

	Name	Description
	_NewEnum (see page 19347)	This is _NewEnum, a member of class ISearches.
	Get_Count (see page 19347)	This is Get_Count, a member of class ISearches.
	Get_Default (see page 19347)	This is Get_Default, a member of class ISearches.
	Item (see page 19347)	This is Item, a member of class ISearches.

132.2.16.2.1 ISearches._NewEnum Method

Pascal

```
function _NewEnum: IUnknown; safecall;
```

C++

```
__fastcall IUnknown _NewEnum();
```

Description

This is _NewEnum, a member of class ISearches.

132.2.16.2.2 ISearches.Get_Count Method

Pascal

```
function Get_Count: Integer; safecall;
```

C++

```
__fastcall int Get_Count();
```

Description

This is Get_Count, a member of class ISearches.

132.2.16.2.3 ISearches.Get_Default Method

Pascal

```
function Get_Default: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Default();
```

Description

This is Get_Default, a member of class ISearches.

132.2.16.2.4 ISearches.Item Method

Pascal

```
function Item(index: OleVariant): ISearch; safecall;
```

C++

```
__fastcall ISearch Item(OleVariant index);
```

Description

This is Item, a member of class ISearches.

132.2.16.3 ISearches Properties

The properties of the ISearches class are listed here.

Public Properties

	Name	Description
R	Count (see page 19348)	This is Count, a member of class ISearches.
R	Default (see page 19348)	This is Default, a member of class ISearches.

132.2.16.3.1 ISearches.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class ISearches.

132.2.16.3.2 ISearches.Default Property

Pascal

```
property Default: WideString;
```

C++

```
__property BSTR Default;
```

Description

This is Default, a member of class ISearches.

132.2.17 ISearchesDisp Interface

Class Hierarchy

SHDocVw.ISearchesDisp

Pascal

```
ISearchesDisp = dispinterface;
```

C++

```
class ISearchesDisp;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchesDisp.

132.2.17.1 ISearchesDisp Members

The following tables list the members exposed by ISearchesDisp.

Public Methods

	Name	Description
	_NewEnum (see page 19349)	This is _NewEnum, a member of class ISearchesDisp.
	Item (see page 19349)	This is Item, a member of class ISearchesDisp.

Public Properties

	Name	Description
	Count (see page 19350)	This is Count, a member of class ISearchesDisp.
	Default (see page 19350)	This is Default, a member of class ISearchesDisp.

132.2.17.2 ISearchesDisp Methods

The methods of the ISearchesDisp class are listed here.

Public Methods

	Name	Description
	_NewEnum (see page 19349)	This is _NewEnum, a member of class ISearchesDisp.
	Item (see page 19349)	This is Item, a member of class ISearchesDisp.

132.2.17.2.1 ISearchesDisp._NewEnum Method

Pascal

```
function _NewEnum: IUnknown;
```

C++

```
__fastcall IUnknown _NewEnum();
```

Description

This is _NewEnum, a member of class ISearchesDisp.

132.2.17.2.2 ISearchesDisp.Item Method

Pascal

```
function Item(index: OleVariant): ISearch;
```

C++

```
__fastcall ISearch Item(OleVariant index);
```

Description

This is Item, a member of class ISearchesDisp.

132.2.17.3 ISearchesDisp Properties

The properties of the ISearchesDisp class are listed here.

Public Properties

	Name	Description
	Count (see page 19350)	This is Count, a member of class ISearchesDisp.
	Default (see page 19350)	This is Default, a member of class ISearchesDisp.

132.2.17.3.1 ISearchesDisp.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class ISearchesDisp.

132.2.17.3.2 ISearchesDisp.Default Property

Pascal

```
property Default: WideString;
```

C++

```
__property BSTR Default;
```

Description

This is Default, a member of class ISearchesDisp.

132.2.18 IShellFavoritesNameSpace Interface

Class Hierarchy

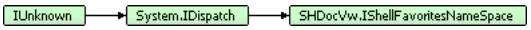

Pascal

```
IShellFavoritesNameSpace = interface(IDispatch);
```

C++

```
__interface IShellFavoritesNameSpace : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.IShellFavoritesNameSpace.

132.2.18.1 IShellFavoritesNameSpace Members

The following tables list the members exposed by IShellFavoritesNameSpace.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 19352)	This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpace.
	DeleteSubscriptionForSelection (see page 19352)	This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpace.
	Export (see page 19352)	This is Export, a member of class IShellFavoritesNameSpace.
	Get_SubscriptionsEnabled (see page 19352)	This is Get_SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	Import (see page 19353)	This is Import, a member of class IShellFavoritesNameSpace.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	InvokeContextMenuCommand (see page 19353)	This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpace.
	MoveSelectionDown (see page 19353)	This is MoveSelectionDown, a member of class IShellFavoritesNameSpace.
	MoveSelectionTo (see page 19353)	This is MoveSelectionTo, a member of class IShellFavoritesNameSpace.
	MoveSelectionUp (see page 19353)	This is MoveSelectionUp, a member of class IShellFavoritesNameSpace.
	NewFolder (see page 19354)	This is NewFolder, a member of class IShellFavoritesNameSpace.
	ResetSort (see page 19354)	This is ResetSort, a member of class IShellFavoritesNameSpace.
	SetRoot (see page 19354)	This is SetRoot, a member of class IShellFavoritesNameSpace.
	Synchronize (see page 19354)	This is Synchronize, a member of class IShellFavoritesNameSpace.

Public Properties

	Name	Description
R	SubscriptionsEnabled (see page 19355)	This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.

132.2.18.2 IShellFavoritesNameSpace Methods

The methods of the IShellFavoritesNameSpace class are listed here.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 19352)	This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpace.
	DeleteSubscriptionForSelection (see page 19352)	This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpace.
	Export (see page 19352)	This is Export, a member of class IShellFavoritesNameSpace.
	Get_SubscriptionsEnabled (see page 19352)	This is Get_SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.
	Import (see page 19353)	This is Import, a member of class IShellFavoritesNameSpace.
	InvokeContextMenuCommand (see page 19353)	This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpace.
	MoveSelectionDown (see page 19353)	This is MoveSelectionDown, a member of class IShellFavoritesNameSpace.
	MoveSelectionTo (see page 19353)	This is MoveSelectionTo, a member of class IShellFavoritesNameSpace.
	MoveSelectionUp (see page 19353)	This is MoveSelectionUp, a member of class IShellFavoritesNameSpace.
	NewFolder (see page 19354)	This is NewFolder, a member of class IShellFavoritesNameSpace.
	ResetSort (see page 19354)	This is ResetSort, a member of class IShellFavoritesNameSpace.
	SetRoot (see page 19354)	This is SetRoot, a member of class IShellFavoritesNameSpace.

	Synchronize (see page 19354)	This is Synchronize, a member of class IShellFavoritesNameSpace.

132.2.18.2.1 IShellFavoritesNameSpace.CreateSubscriptionForSelection Method

Pascal

```
function CreateSubscriptionForSelection: WordBool; safecall;
```

C++

```
__fastcall WordBool CreateSubscriptionForSelection();
```

Description

This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpace.

132.2.18.2.2 IShellFavoritesNameSpace.DeleteSubscriptionForSelection Method

Pascal

```
function DeleteSubscriptionForSelection: WordBool; safecall;
```

C++

```
__fastcall WordBool DeleteSubscriptionForSelection();
```

Description

This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpace.

132.2.18.2.3 IShellFavoritesNameSpace.Export Method

Pascal

```
procedure Export; safecall;
```

C++

```
__fastcall Export();
```

Description

This is Export, a member of class IShellFavoritesNameSpace.

132.2.18.2.4 IShellFavoritesNameSpace.Get_SubscriptionsEnabled Method

Pascal

```
function Get_SubscriptionsEnabled: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_SubscriptionsEnabled();
```

Description

This is Get_SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.

132.2.18.2.5 IShellFavoritesNameSpace.Import Method

Pascal

```
procedure Import; safecall;
```

C++

```
__fastcall Import();
```

Description

This is Import, a member of class IShellFavoritesNameSpace.

132.2.18.2.6 IShellFavoritesNameSpace.InvokeContextMenuCommand Method

Pascal

```
procedure InvokeContextMenuCommand(const strCommand: WideString); safecall;
```

C++

```
__fastcall InvokeContextMenuCommand(const BSTR strCommand);
```

Description

This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpace.

132.2.18.2.7 IShellFavoritesNameSpace.MoveSelectionDown Method

Pascal

```
procedure MoveSelectionDown; safecall;
```

C++

```
__fastcall MoveSelectionDown();
```

Description

This is MoveSelectionDown, a member of class IShellFavoritesNameSpace.

132.2.18.2.8 IShellFavoritesNameSpace.MoveSelectionTo Method

Pascal

```
procedure MoveSelectionTo; safecall;
```

C++

```
__fastcall MoveSelectionTo();
```

Description

This is MoveSelectionTo, a member of class IShellFavoritesNameSpace.

132.2.18.2.9 IShellFavoritesNameSpace.MoveSelectionUp Method

Pascal

```
procedure MoveSelectionUp; safecall;
```

C++

```
__fastcall MoveSelectionUp();
```

Description

This is MoveSelectionUp, a member of class IShellFavoritesNameSpace.

132.2.18.2.10 IShellFavoritesNameSpace.NewFolder Method

Pascal

```
procedure NewFolder; safecall;
```

C++

```
__fastcall NewFolder();
```

Description

This is NewFolder, a member of class IShellFavoritesNameSpace.

132.2.18.2.11 IShellFavoritesNameSpace.ResetSort Method

Pascal

```
procedure ResetSort; safecall;
```

C++

```
__fastcall ResetSort();
```

Description

This is ResetSort, a member of class IShellFavoritesNameSpace.

132.2.18.2.12 IShellFavoritesNameSpace.SetRoot Method

Pascal

```
procedure SetRoot(const bstrFullPath: WideString); safecall;
```

C++

```
__fastcall SetRoot(const BSTR bstrFullPath);
```

Description

This is SetRoot, a member of class IShellFavoritesNameSpace.

132.2.18.2.13 IShellFavoritesNameSpace.Synchronize Method

Pascal

```
procedure Synchronize; safecall;
```

C++

```
__fastcall Synchronize();
```

Description

This is Synchronize, a member of class IShellFavoritesNameSpace.

132.2.18.3 IShellFavoritesNameSpace Properties

The properties of the IShellFavoritesNameSpace class are listed here.

Public Properties

	Name	Description
	SubscriptionsEnabled (see page 19355)	This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.

132.2.18.3.1 IShellFavoritesNameSpace.SubscriptionsEnabled Property

Pascal

```
property SubscriptionsEnabled: WordBool;
```

C++

```
__property WordBool SubscriptionsEnabled;
```

Description

This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.

132.2.19 IShellFavoritesNameSpaceDisp Interface

Class Hierarchy

SHDocVw.IShellFavoritesNameSpaceDisp

Pascal

```
IShellFavoritesNameSpaceDisp = dispinterface;
```

C++

```
class IShellFavoritesNameSpaceDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IShellFavoritesNameSpaceDisp.

132.2.19.1 IShellFavoritesNameSpaceDisp Members

The following tables list the members exposed by IShellFavoritesNameSpaceDisp.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 19356)	This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpaceDisp.
	DeleteSubscriptionForSelection (see page 19356)	This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpaceDisp.
	Export (see page 19357)	This is Export, a member of class IShellFavoritesNameSpaceDisp.
	Import (see page 19357)	This is Import, a member of class IShellFavoritesNameSpaceDisp.
	InvokeContextMenuCommand (see page 19357)	This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpaceDisp.
	MoveSelectionDown (see page 19357)	This is MoveSelectionDown, a member of class IShellFavoritesNameSpaceDisp.
	MoveSelectionTo (see page 19358)	This is MoveSelectionTo, a member of class IShellFavoritesNameSpaceDisp.
	MoveSelectionUp (see page 19358)	This is MoveSelectionUp, a member of class IShellFavoritesNameSpaceDisp.
	NewFolder (see page 19358)	This is NewFolder, a member of class IShellFavoritesNameSpaceDisp.

	ResetSort (see page 19358)	This is ResetSort, a member of class IShellFavoritesNameSpaceDisp.
	SetRoot (see page 19358)	This is SetRoot, a member of class IShellFavoritesNameSpaceDisp.
	Synchronize (see page 19359)	This is Synchronize, a member of class IShellFavoritesNameSpaceDisp.

Public Properties

	Name	Description
	SubscriptionsEnabled (see page 19359)	This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpaceDisp.

132.2.19.2 IShellFavoritesNameSpaceDisp Methods

The methods of the IShellFavoritesNameSpaceDisp class are listed here.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 19356)	This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpaceDisp.
	DeleteSubscriptionForSelection (see page 19356)	This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpaceDisp.
	Export (see page 19357)	This is Export, a member of class IShellFavoritesNameSpaceDisp.
	Import (see page 19357)	This is Import, a member of class IShellFavoritesNameSpaceDisp.
	InvokeContextMenuCommand (see page 19357)	This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpaceDisp.
	MoveSelectionDown (see page 19357)	This is MoveSelectionDown, a member of class IShellFavoritesNameSpaceDisp.
	MoveSelectionTo (see page 19358)	This is MoveSelectionTo, a member of class IShellFavoritesNameSpaceDisp.
	MoveSelectionUp (see page 19358)	This is MoveSelectionUp, a member of class IShellFavoritesNameSpaceDisp.
	NewFolder (see page 19358)	This is NewFolder, a member of class IShellFavoritesNameSpaceDisp.
	ResetSort (see page 19358)	This is ResetSort, a member of class IShellFavoritesNameSpaceDisp.
	SetRoot (see page 19358)	This is SetRoot, a member of class IShellFavoritesNameSpaceDisp.
	Synchronize (see page 19359)	This is Synchronize, a member of class IShellFavoritesNameSpaceDisp.

132.2.19.2.1 IShellFavoritesNameSpaceDisp.CreateSubscriptionForSelection Method

Pascal

```
function CreateSubscriptionForSelection: WordBool;
```

C++

```
__fastcall WordBool CreateSubscriptionForSelection();
```

Description

This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpaceDisp.

132.2.19.2.2 IShellFavoritesNameSpaceDisp.DeleteSubscriptionForSelection Method

Pascal

```
function DeleteSubscriptionForSelection: WordBool;
```

C++

```
__fastcall WordBool DeleteSubscriptionForSelection();
```

Description

This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpaceDisp.

132.2.19.2.3 IShellFavoritesNameSpaceDisp.Export Method

Pascal

```
procedure Export;
```

C++

```
__fastcall Export();
```

Description

This is Export, a member of class IShellFavoritesNameSpaceDisp.

132.2.19.2.4 IShellFavoritesNameSpaceDisp.Import Method

Pascal

```
procedure Import;
```

C++

```
__fastcall Import();
```

Description

This is Import, a member of class IShellFavoritesNameSpaceDisp.

132.2.19.2.5 IShellFavoritesNameSpaceDisp.InvokeContextMenuCommand Method

Pascal

```
procedure InvokeContextMenuCommand(const strCommand: WideString);
```

C++

```
__fastcall InvokeContextMenuCommand(const BSTR strCommand);
```

Description

This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpaceDisp.

132.2.19.2.6 IShellFavoritesNameSpaceDisp.MoveSelectionDown Method

Pascal

```
procedure MoveSelectionDown;
```

C++

```
__fastcall MoveSelectionDown();
```

Description

This is MoveSelectionDown, a member of class IShellFavoritesNameSpaceDisp.

132.2.19.2.7 IShellFavoritesNameSpaceDisp.MoveSelectionTo Method

Pascal

```
procedure MoveSelectionTo;
```

C++

```
__fastcall MoveSelectionTo();
```

Description

This is MoveSelectionTo, a member of class IShellFavoritesNameSpaceDisp.

132.2.19.2.8 IShellFavoritesNameSpaceDisp.MoveSelectionUp Method

Pascal

```
procedure MoveSelectionUp;
```

C++

```
__fastcall MoveSelectionUp();
```

Description

This is MoveSelectionUp, a member of class IShellFavoritesNameSpaceDisp.

132.2.19.2.9 IShellFavoritesNameSpaceDisp.NewFolder Method

Pascal

```
procedure NewFolder;
```

C++

```
__fastcall NewFolder();
```

Description

This is NewFolder, a member of class IShellFavoritesNameSpaceDisp.

132.2.19.2.10 IShellFavoritesNameSpaceDisp.ResetSort Method

Pascal

```
procedure ResetSort;
```

C++

```
__fastcall ResetSort();
```

Description

This is ResetSort, a member of class IShellFavoritesNameSpaceDisp.

132.2.19.2.11 IShellFavoritesNameSpaceDisp.SetRoot Method

Pascal

```
procedure SetRoot(const bstrFullPath: WideString);
```

C++

```
__fastcall SetRoot(const BSTR bstrFullPath);
```

Description

This is SetRoot, a member of class IShellFavoritesNameSpaceDisp.

132.2.19.2.12 IShellFavoritesNameSpaceDisp.Synchronize Method

Pascal

```
procedure Synchronize;
```

C++

```
__fastcall Synchronize();
```

Description

This is Synchronize, a member of class IShellFavoritesNameSpaceDisp.

132.2.19.3 IShellFavoritesNameSpaceDisp Properties

The properties of the IShellFavoritesNameSpaceDisp class are listed here.

Public Properties

	Name	Description
	SubscriptionsEnabled (see page 19359)	This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpaceDisp.

132.2.19.3.1 IShellFavoritesNameSpaceDisp.SubscriptionsEnabled Property

Pascal

```
property SubscriptionsEnabled: WordBool;
```

C++

```
__property WordBool SubscriptionsEnabled;
```

Description

This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpaceDisp.

132.2.20 IShellNameSpace Interface

Class Hierarchy

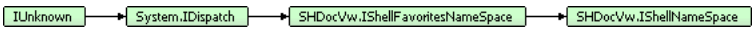

Pascal

```
IShellNameSpace = interface(IShellFavoritesNameSpace);
```

C++

```
__interface IShellNameSpace : public IShellFavoritesNameSpace;
```

File

SHDocVw

Description

This is class SHDocVw.IShellNameSpace.

132.2.20.1 IShellNameSpace Members

The following tables list the members exposed by IShellNameSpace.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 19352)	This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpace.
	DeleteSubscriptionForSelection (see page 19352)	This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpace.
	Expand (see page 19361)	This is Expand, a member of class IShellNameSpace.
	Export (see page 19352)	This is Export, a member of class IShellFavoritesNameSpace.
	Get_Columns (see page 19362)	This is Get_Columns, a member of class IShellNameSpace.
	Get_CountViewTypes (see page 19362)	This is Get_CountViewTypes, a member of class IShellNameSpace.
	Get_Depth (see page 19362)	This is Get_Depth, a member of class IShellNameSpace.
	Get_EnumOptions (see page 19362)	This is Get_EnumOptions, a member of class IShellNameSpace.
	Get_Flags (see page 19362)	This is Get_Flags, a member of class IShellNameSpace.
	Get_Mode (see page 19363)	This is Get_Mode, a member of class IShellNameSpace.
	Get_Root (see page 19363)	This is Get_Root, a member of class IShellNameSpace.
	Get_SelectedItem (see page 19363)	This is Get_SelectedItem, a member of class IShellNameSpace.
	Get_SubscriptionsEnabled (see page 19352)	This is Get_SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.
	Get_TVFlags (see page 19363)	This is Get_TVFlags, a member of class IShellNameSpace.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	Import (see page 19353)	This is Import, a member of class IShellFavoritesNameSpace.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	InvokeContextMenuCommand (see page 19353)	This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpace.
	MoveSelectionDown (see page 19353)	This is MoveSelectionDown, a member of class IShellFavoritesNameSpace.
	MoveSelectionTo (see page 19353)	This is MoveSelectionTo, a member of class IShellFavoritesNameSpace.
	MoveSelectionUp (see page 19353)	This is MoveSelectionUp, a member of class IShellFavoritesNameSpace.
	NewFolder (see page 19354)	This is NewFolder, a member of class IShellFavoritesNameSpace.
	ResetSort (see page 19354)	This is ResetSort, a member of class IShellFavoritesNameSpace.
	SelectedItems (see page 19364)	This is SelectedItems, a member of class IShellNameSpace.
	Set_Columns (see page 19364)	This is Set_Columns, a member of class IShellNameSpace.
	Set_Depth (see page 19364)	This is Set_Depth, a member of class IShellNameSpace.
	Set_EnumOptions (see page 19364)	This is Set_EnumOptions, a member of class IShellNameSpace.
	Set_Flags (see page 19364)	This is Set_Flags, a member of class IShellNameSpace.
	Set_Mode (see page 19365)	This is Set_Mode, a member of class IShellNameSpace.
	Set_Root (see page 19365)	This is Set_Root, a member of class IShellNameSpace.
	Set_SelectedItem (see page 19365)	This is Set_SelectedItem, a member of class IShellNameSpace.
	Set_TVFlags (see page 19365)	This is Set_TVFlags, a member of class IShellNameSpace.
	SetRoot (see page 19354)	This is SetRoot, a member of class IShellFavoritesNameSpace.
	SetViewType (see page 19366)	This is SetViewType, a member of class IShellNameSpace.
	Synchronize (see page 19354)	This is Synchronize, a member of class IShellFavoritesNameSpace.
	UnselectAll (see page 19366)	This is UnselectAll, a member of class IShellNameSpace.

Public Properties

	Name	Description
	Columns (see page 19366)	This is Columns, a member of class IShellNameSpace.
R	CountViewTypes (see page 19367)	This is CountViewTypes, a member of class IShellNameSpace.
	Depth (see page 19367)	This is Depth, a member of class IShellNameSpace.
	EnumOptions (see page 19367)	This is EnumOptions, a member of class IShellNameSpace.
	Flags (see page 19367)	This is Flags, a member of class IShellNameSpace.
	Mode (see page 19367)	This is Mode, a member of class IShellNameSpace.
	Root (see page 19368)	This is Root, a member of class IShellNameSpace.
	SelectedItem (see page 19368)	This is SelectedItem, a member of class IShellNameSpace.
R	SubscriptionsEnabled (see page 19355)	This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.
	TVFlags (see page 19368)	This is TVFlags, a member of class IShellNameSpace.

132.2.20.2 IShellNameSpace Methods

The methods of the IShellNameSpace class are listed here.

Public Methods

	Name	Description
	Expand (see page 19361)	This is Expand, a member of class IShellNameSpace.
	Get_Columns (see page 19362)	This is Get_Columns, a member of class IShellNameSpace.
	Get_CountViewTypes (see page 19362)	This is Get_CountViewTypes, a member of class IShellNameSpace.
	Get_Depth (see page 19362)	This is Get_Depth, a member of class IShellNameSpace.
	Get_EnumOptions (see page 19362)	This is Get_EnumOptions, a member of class IShellNameSpace.
	Get_Flags (see page 19362)	This is Get_Flags, a member of class IShellNameSpace.
	Get_Mode (see page 19363)	This is Get_Mode, a member of class IShellNameSpace.
	Get_Root (see page 19363)	This is Get_Root, a member of class IShellNameSpace.
	Get_SelectedItem (see page 19363)	This is Get_SelectedItem, a member of class IShellNameSpace.
	Get_TVFlags (see page 19363)	This is Get_TVFlags, a member of class IShellNameSpace.
	SelectedItems (see page 19364)	This is SelectedItems, a member of class IShellNameSpace.
	Set_Columns (see page 19364)	This is Set_Columns, a member of class IShellNameSpace.
	Set_Depth (see page 19364)	This is Set_Depth, a member of class IShellNameSpace.
	Set_EnumOptions (see page 19364)	This is Set_EnumOptions, a member of class IShellNameSpace.
	Set_Flags (see page 19364)	This is Set_Flags, a member of class IShellNameSpace.
	Set_Mode (see page 19365)	This is Set_Mode, a member of class IShellNameSpace.
	Set_Root (see page 19365)	This is Set_Root, a member of class IShellNameSpace.
	Set_SelectedItem (see page 19365)	This is Set_SelectedItem, a member of class IShellNameSpace.
	Set_TVFlags (see page 19365)	This is Set_TVFlags, a member of class IShellNameSpace.
	SetViewType (see page 19366)	This is SetViewType, a member of class IShellNameSpace.
	UnselectAll (see page 19366)	This is UnselectAll, a member of class IShellNameSpace.

132.2.20.2.1 IShellNameSpace.Expand Method

Pascal

```
procedure Expand(var_: OleVariant; iDepth: SYSINT); safecall;
```

C++

```
__fastcall Expand(OleVariant var_, SYSINT iDepth);
```

Description

This is Expand, a member of class IShellNameSpace.

132.2.20.2.2 IShellNameSpace.Get_Columns Method

Pascal

```
function Get_Columns: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Columns();
```

Description

This is Get_Columns, a member of class IShellNameSpace.

132.2.20.2.3 IShellNameSpace.Get_CountViewTypes Method

Pascal

```
function Get_CountViewTypes: SYSINT; safecall;
```

C++

```
__fastcall SYSINT Get_CountViewTypes();
```

Description

This is Get_CountViewTypes, a member of class IShellNameSpace.

132.2.20.2.4 IShellNameSpace.Get_Depth Method

Pascal

```
function Get_Depth: SYSINT; safecall;
```

C++

```
__fastcall SYSINT Get_Depth();
```

Description

This is Get_Depth, a member of class IShellNameSpace.

132.2.20.2.5 IShellNameSpace.Get_EnumOptions Method

Pascal

```
function Get_EnumOptions: Integer; safecall;
```

C++

```
__fastcall int Get_EnumOptions();
```

Description

This is Get_EnumOptions, a member of class IShellNameSpace.

132.2.20.2.6 IShellNameSpace.Get_Flags Method

Pascal

```
function Get_Flags: LongWord; safecall;
```

C++

```
__fastcall LongWord Get_Flags();
```

Description

This is Get_Flags, a member of class IShellNameSpace.

132.2.20.2.7 IShellNameSpace.Get_Mode Method

Pascal

```
function Get_Mode: SYSUINT; safecall;
```

C++

```
__fastcall SYSUINT Get_Mode();
```

Description

This is Get_Mode, a member of class IShellNameSpace.

132.2.20.2.8 IShellNameSpace.Get_Root Method

Pascal

```
function Get_Root: OleVariant; safecall;
```

C++

```
__fastcall OleVariant Get_Root();
```

Description

This is Get_Root, a member of class IShellNameSpace.

132.2.20.2.9 IShellNameSpace.Get_SelectedItem Method

Pascal

```
function Get_SelectedItem: IDispatch; safecall;
```

C++

```
__fastcall IDispatch Get_SelectedItem();
```

Description

This is Get_SelectedItem, a member of class IShellNameSpace.

132.2.20.2.10 IShellNameSpace.Get_TVFlags Method

Pascal

```
function Get_TVFlags: LongWord; safecall;
```

C++

```
__fastcall LongWord Get_TVFlags();
```

Description

This is Get_TVFlags, a member of class IShellNameSpace.

132.2.20.2.11 IShellNameSpace.SelectedItems Method

Pascal

```
function SelectedItems: IDispatch; safecall;
```

C++

```
__fastcall IDispatch SelectedItems();
```

Description

This is SelectedItems, a member of class IShellNameSpace.

132.2.20.2.12 IShellNameSpace.Set_Columns Method

Pascal

```
procedure Set_Columns(const bstrColumns: WideString); safecall;
```

C++

```
__fastcall Set_Columns(const BSTR bstrColumns);
```

Description

This is Set_Columns, a member of class IShellNameSpace.

132.2.20.2.13 IShellNameSpace.Set_Depth Method

Pascal

```
procedure Set_Depth(piDepth: SYSINT); safecall;
```

C++

```
__fastcall Set_Depth(SYSINT piDepth);
```

Description

This is Set_Depth, a member of class IShellNameSpace.

132.2.20.2.14 IShellNameSpace.Set_EnumOptions Method

Pascal

```
procedure Set_EnumOptions(pgrfEnumFlags: Integer); safecall;
```

C++

```
__fastcall Set_EnumOptions(int pgrfEnumFlags);
```

Description

This is Set_EnumOptions, a member of class IShellNameSpace.

132.2.20.2.15 IShellNameSpace.Set_Flags Method

Pascal

```
procedure Set_Flags(pdwFlags: LongWord); safecall;
```

C++

```
__fastcall Set_Flags(LongWord pdwFlags);
```

Description

This is Set_Flags, a member of class IShellNameSpace.

132.2.20.2.16 IShellNameSpace.Set_Mode Method

Pascal

```
procedure Set_Mode(puMode: SYSUINT); safecall;
```

C++

```
__fastcall Set_Mode(SYSUINT puMode);
```

Description

This is Set_Mode, a member of class IShellNameSpace.

132.2.20.2.17 IShellNameSpace.Set_Root Method

Pascal

```
procedure Set_Root(pvar: OleVariant); safecall;
```

C++

```
__fastcall Set_Root(OleVariant pvar);
```

Description

This is Set_Root, a member of class IShellNameSpace.

132.2.20.2.18 IShellNameSpace.Set_SelectedItem Method

Pascal

```
procedure Set_SelectedItem(const pItem: IDispatch); safecall;
```

C++

```
__fastcall Set_SelectedItem(const IDispatch pItem);
```

Description

This is Set_SelectedItem, a member of class IShellNameSpace.

132.2.20.2.19 IShellNameSpace.Set_TVFlags Method

Pascal

```
procedure Set_TVFlags(dwFlags: LongWord); safecall;
```

C++

```
__fastcall Set_TVFlags(LongWord dwFlags);
```

Description

This is Set_TVFlags, a member of class IShellNameSpace.

132.2.20.2.20 IShellNameSpace.SetViewType Method

Pascal

```
procedure SetViewType(iType: SYSINT); safecall;
```

C++

```
__fastcall SetViewType(SYSINT iType);
```

Description

This is SetViewType, a member of class IShellNameSpace.

132.2.20.2.21 IShellNameSpace.UnselectAll Method

Pascal

```
procedure UnselectAll; safecall;
```

C++

```
__fastcall UnselectAll();
```

Description

This is UnselectAll, a member of class IShellNameSpace.

132.2.20.3 IShellNameSpace Properties

The properties of the IShellNameSpace class are listed here.

Public Properties

	Name	Description
	Columns (see page 19366)	This is Columns, a member of class IShellNameSpace.
R	CountViewTypes (see page 19367)	This is CountViewTypes, a member of class IShellNameSpace.
	Depth (see page 19367)	This is Depth, a member of class IShellNameSpace.
	EnumOptions (see page 19367)	This is EnumOptions, a member of class IShellNameSpace.
	Flags (see page 19367)	This is Flags, a member of class IShellNameSpace.
	Mode (see page 19367)	This is Mode, a member of class IShellNameSpace.
	Root (see page 19368)	This is Root, a member of class IShellNameSpace.
	SelectedItem (see page 19368)	This is SelectedItem, a member of class IShellNameSpace.
	TVFlags (see page 19368)	This is TVFlags, a member of class IShellNameSpace.

132.2.20.3.1 IShellNameSpace.Columns Property

Pascal

```
property Columns: WideString;
```

C++

```
__property BSTR Columns;
```

Description

This is Columns, a member of class IShellNameSpace.

132.2.20.3.2 IShellNameSpace.CountViewTypes Property

Pascal

```
property CountViewTypes: SYSINT;
```

C++

```
__property SYSINT CountViewTypes;
```

Description

This is CountViewTypes, a member of class IShellNameSpace.

132.2.20.3.3 IShellNameSpace.Depth Property

Pascal

```
property Depth: SYSINT;
```

C++

```
__property SYSINT Depth;
```

Description

This is Depth, a member of class IShellNameSpace.

132.2.20.3.4 IShellNameSpace.EnumOptions Property

Pascal

```
property EnumOptions: Integer;
```

C++

```
__property int EnumOptions;
```

Description

This is EnumOptions, a member of class IShellNameSpace.

132.2.20.3.5 IShellNameSpace.Flags Property

Pascal

```
property Flags: LongWord;
```

C++

```
__property LongWord Flags;
```

Description

This is Flags, a member of class IShellNameSpace.

132.2.20.3.6 IShellNameSpace.Mode Property

Pascal

```
property Mode: SYSUINT;
```

C++

```
__property SYSUINT Mode;
```

Description

This is Mode, a member of class IShellNameSpace.

132.2.20.3.7 IShellNameSpace.Root Property

Pascal

```
property Root: OleVariant;
```

C++

```
__property OleVariant Root;
```

Description

This is Root, a member of class IShellNameSpace.

132.2.20.3.8 IShellNameSpace.SelectedItem Property

Pascal

```
property SelectedItem: IDispatch;
```

C++

```
__property IDispatch SelectedItem;
```

Description

This is SelectedItem, a member of class IShellNameSpace.

132.2.20.3.9 IShellNameSpace.TVFlags Property

Pascal

```
property TVFlags: LongWord;
```

C++

```
__property LongWord TVFlags;
```

Description

This is TVFlags, a member of class IShellNameSpace.

132.2.21 IShellNameSpaceDisp Interface

Class Hierarchy

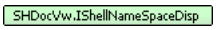

Pascal

```
IShellNameSpaceDisp = dispinterface;
```

C++

```
class IShellNameSpaceDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IShellNameSpaceDisp.

132.2.21.1 IShellNameSpaceDisp Members

The following tables list the members exposed by IShellNameSpaceDisp.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 19370)	This is CreateSubscriptionForSelection, a member of class IShellNameSpaceDisp.
	DeleteSubscriptionForSelection (see page 19370)	This is DeleteSubscriptionForSelection, a member of class IShellNameSpaceDisp.
	Expand (see page 19370)	This is Expand, a member of class IShellNameSpaceDisp.
	Export (see page 19370)	This is Export, a member of class IShellNameSpaceDisp.
	Import (see page 19371)	This is Import, a member of class IShellNameSpaceDisp.
	InvokeContextMenuCommand (see page 19371)	This is InvokeContextMenuCommand, a member of class IShellNameSpaceDisp.
	MoveSelectionDown (see page 19371)	This is MoveSelectionDown, a member of class IShellNameSpaceDisp.
	MoveSelectionTo (see page 19371)	This is MoveSelectionTo, a member of class IShellNameSpaceDisp.
	MoveSelectionUp (see page 19372)	This is MoveSelectionUp, a member of class IShellNameSpaceDisp.
	NewFolder (see page 19372)	This is NewFolder, a member of class IShellNameSpaceDisp.
	ResetSort (see page 19372)	This is ResetSort, a member of class IShellNameSpaceDisp.
	SelectedItems (see page 19372)	This is SelectedItems, a member of class IShellNameSpaceDisp.
	SetRoot (see page 19372)	This is SetRoot, a member of class IShellNameSpaceDisp.
	SetViewType (see page 19373)	This is SetViewType, a member of class IShellNameSpaceDisp.
	Synchronize (see page 19373)	This is Synchronize, a member of class IShellNameSpaceDisp.
	UnselectAll (see page 19373)	This is UnselectAll, a member of class IShellNameSpaceDisp.

Public Properties

	Name	Description
	Columns (see page 19374)	This is Columns, a member of class IShellNameSpaceDisp.
	CountViewTypes (see page 19374)	This is CountViewTypes, a member of class IShellNameSpaceDisp.
	Depth (see page 19374)	This is Depth, a member of class IShellNameSpaceDisp.
	EnumOptions (see page 19374)	This is EnumOptions, a member of class IShellNameSpaceDisp.
	Flags (see page 19374)	This is Flags, a member of class IShellNameSpaceDisp.
	Mode (see page 19375)	This is Mode, a member of class IShellNameSpaceDisp.
	Root (see page 19375)	This is Root, a member of class IShellNameSpaceDisp.
	SelectedItem (see page 19375)	This is SelectedItem, a member of class IShellNameSpaceDisp.
	SubscriptionsEnabled (see page 19375)	This is SubscriptionsEnabled, a member of class IShellNameSpaceDisp.
	TVFlags (see page 19376)	This is TVFlags, a member of class IShellNameSpaceDisp.

132.2.21.2 IShellNameSpaceDisp Methods

The methods of the IShellNameSpaceDisp class are listed here.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 19370)	This is CreateSubscriptionForSelection, a member of class IShellNameSpaceDisp.
	DeleteSubscriptionForSelection (see page 19370)	This is DeleteSubscriptionForSelection, a member of class IShellNameSpaceDisp.
	Expand (see page 19370)	This is Expand, a member of class IShellNameSpaceDisp.

	Export (see page 19370)	This is Export, a member of class IShellNameSpaceDisp.
	Import (see page 19371)	This is Import, a member of class IShellNameSpaceDisp.
	InvokeContextMenuCommand (see page 19371)	This is InvokeContextMenuCommand, a member of class IShellNameSpaceDisp.
	MoveSelectionDown (see page 19371)	This is MoveSelectionDown, a member of class IShellNameSpaceDisp.
	MoveSelectionTo (see page 19371)	This is MoveSelectionTo, a member of class IShellNameSpaceDisp.
	MoveSelectionUp (see page 19372)	This is MoveSelectionUp, a member of class IShellNameSpaceDisp.
	NewFolder (see page 19372)	This is NewFolder, a member of class IShellNameSpaceDisp.
	ResetSort (see page 19372)	This is ResetSort, a member of class IShellNameSpaceDisp.
	SelectedItems (see page 19372)	This is SelectedItems, a member of class IShellNameSpaceDisp.
	SetRoot (see page 19372)	This is SetRoot, a member of class IShellNameSpaceDisp.
	SetViewType (see page 19373)	This is SetViewType, a member of class IShellNameSpaceDisp.
	Synchronize (see page 19373)	This is Synchronize, a member of class IShellNameSpaceDisp.
	UnselectAll (see page 19373)	This is UnselectAll, a member of class IShellNameSpaceDisp.

132.2.21.2.1 IShellNameSpaceDisp.CreateSubscriptionForSelection Method

Pascal

```
function CreateSubscriptionForSelection: WordBool;
```

C++

```
__fastcall WordBool CreateSubscriptionForSelection();
```

Description

This is CreateSubscriptionForSelection, a member of class IShellNameSpaceDisp.

132.2.21.2.2 IShellNameSpaceDisp.DeleteSubscriptionForSelection Method

Pascal

```
function DeleteSubscriptionForSelection: WordBool;
```

C++

```
__fastcall WordBool DeleteSubscriptionForSelection();
```

Description

This is DeleteSubscriptionForSelection, a member of class IShellNameSpaceDisp.

132.2.21.2.3 IShellNameSpaceDisp.Expand Method

Pascal

```
procedure Expand(var_: OleVariant; iDepth: SYSINT);
```

C++

```
__fastcall Expand(OleVariant var_, SYSINT iDepth);
```

Description

This is Expand, a member of class IShellNameSpaceDisp.

132.2.21.2.4 IShellNameSpaceDisp.Export Method

Pascal

```
procedure Export;
```

C++

```
__fastcall Export();
```

Description

This is Export, a member of class IShellNameSpaceDisp.

132.2.21.2.5 IShellNameSpaceDisp.Import Method

Pascal

```
procedure Import;
```

C++

```
__fastcall Import();
```

Description

This is Import, a member of class IShellNameSpaceDisp.

132.2.21.2.6 IShellNameSpaceDisp.InvokeContextMenuCommand Method

Pascal

```
procedure InvokeContextMenuCommand(const strCommand: WideString);
```

C++

```
__fastcall InvokeContextMenuCommand(const BSTR strCommand);
```

Description

This is InvokeContextMenuCommand, a member of class IShellNameSpaceDisp.

132.2.21.2.7 IShellNameSpaceDisp.MoveSelectionDown Method

Pascal

```
procedure MoveSelectionDown;
```

C++

```
__fastcall MoveSelectionDown();
```

Description

This is MoveSelectionDown, a member of class IShellNameSpaceDisp.

132.2.21.2.8 IShellNameSpaceDisp.MoveSelectionTo Method

Pascal

```
procedure MoveSelectionTo;
```

C++

```
__fastcall MoveSelectionTo();
```

Description

This is MoveSelectionTo, a member of class IShellNameSpaceDisp.

132.2.21.2.9 IShellNameSpaceDisp.MoveSelectionUp Method

Pascal

```
procedure MoveSelectionUp;
```

C++

```
__fastcall MoveSelectionUp();
```

Description

This is MoveSelectionUp, a member of class IShellNameSpaceDisp.

132.2.21.2.10 IShellNameSpaceDisp.NewFolder Method

Pascal

```
procedure NewFolder;
```

C++

```
__fastcall NewFolder();
```

Description

This is NewFolder, a member of class IShellNameSpaceDisp.

132.2.21.2.11 IShellNameSpaceDisp.ResetSort Method

Pascal

```
procedure ResetSort;
```

C++

```
__fastcall ResetSort();
```

Description

This is ResetSort, a member of class IShellNameSpaceDisp.

132.2.21.2.12 IShellNameSpaceDisp.SelectedItems Method

Pascal

```
function SelectedItems: IDispatch;
```

C++

```
__fastcall IDispatch SelectedItems();
```

Description

This is SelectedItems, a member of class IShellNameSpaceDisp.

132.2.21.2.13 IShellNameSpaceDisp.SetRoot Method

Pascal

```
procedure SetRoot(const bstrFullPath: WideString);
```

C++

```
__fastcall SetRoot(const BSTR bstrFullPath);
```

Description

This is SetRoot, a member of class IShellNameSpaceDisp.

132.2.21.2.14 IShellNameSpaceDisp.SetViewType Method

Pascal

```
procedure SetViewType(iType: SYSINT);
```

C++

```
__fastcall SetViewType(SYSINT iType);
```

Description

This is SetViewType, a member of class IShellNameSpaceDisp.

132.2.21.2.15 IShellNameSpaceDisp.Synchronize Method

Pascal

```
procedure Synchronize;
```

C++

```
__fastcall Synchronize();
```

Description

This is Synchronize, a member of class IShellNameSpaceDisp.

132.2.21.2.16 IShellNameSpaceDisp.UnselectAll Method

Pascal

```
procedure UnselectAll;
```

C++

```
__fastcall UnselectAll();
```

Description

This is UnselectAll, a member of class IShellNameSpaceDisp.

132.2.21.3 IShellNameSpaceDisp Properties

The properties of the IShellNameSpaceDisp class are listed here.

Public Properties

	Name	Description
	Columns (see page 19374)	This is Columns, a member of class IShellNameSpaceDisp.
	CountViewTypes (see page 19374)	This is CountViewTypes, a member of class IShellNameSpaceDisp.
	Depth (see page 19374)	This is Depth, a member of class IShellNameSpaceDisp.
	EnumOptions (see page 19374)	This is EnumOptions, a member of class IShellNameSpaceDisp.
	Flags (see page 19374)	This is Flags, a member of class IShellNameSpaceDisp.
	Mode (see page 19375)	This is Mode, a member of class IShellNameSpaceDisp.
	Root (see page 19375)	This is Root, a member of class IShellNameSpaceDisp.
	SelectedItem (see page 19375)	This is SelectedItem, a member of class IShellNameSpaceDisp.

	SubscriptionsEnabled (see page 19375)	This is SubscriptionsEnabled, a member of class IShellNameSpaceDisp.
	TVFlags (see page 19376)	This is TVFlags, a member of class IShellNameSpaceDisp.

132.2.21.3.1 IShellNameSpaceDisp.Columns Property

Pascal

```
property Columns: WideString;
```

C++

```
__property BSTR Columns;
```

Description

This is Columns, a member of class IShellNameSpaceDisp.

132.2.21.3.2 IShellNameSpaceDisp.CountViewTypes Property

Pascal

```
property CountViewTypes: SYSINT;
```

C++

```
__property SYSINT CountViewTypes;
```

Description

This is CountViewTypes, a member of class IShellNameSpaceDisp.

132.2.21.3.3 IShellNameSpaceDisp.Depth Property

Pascal

```
property Depth: SYSINT;
```

C++

```
__property SYSINT Depth;
```

Description

This is Depth, a member of class IShellNameSpaceDisp.

132.2.21.3.4 IShellNameSpaceDisp.EnumOptions Property

Pascal

```
property EnumOptions: Integer;
```

C++

```
__property int EnumOptions;
```

Description

This is EnumOptions, a member of class IShellNameSpaceDisp.

132.2.21.3.5 IShellNameSpaceDisp.Flags Property

Pascal

```
property Flags: LongWord;
```

C++

```
__property LongWord Flags;
```

Description

This is Flags, a member of class IShellNameSpaceDisp.

132.2.21.3.6 IShellNameSpaceDisp.Mode Property

Pascal

```
property Mode: SYSUINT;
```

C++

```
__property SYSUINT Mode;
```

Description

This is Mode, a member of class IShellNameSpaceDisp.

132.2.21.3.7 IShellNameSpaceDisp.Root Property

Pascal

```
property Root: OleVariant;
```

C++

```
__property OleVariant Root;
```

Description

This is Root, a member of class IShellNameSpaceDisp.

132.2.21.3.8 IShellNameSpaceDisp.SelectedItem Property

Pascal

```
property SelectedItem: IDispatch;
```

C++

```
__property IDispatch SelectedItem;
```

Description

This is SelectedItem, a member of class IShellNameSpaceDisp.

132.2.21.3.9 IShellNameSpaceDisp.SubscriptionsEnabled Property

Pascal

```
property SubscriptionsEnabled: WordBool;
```

C++

```
__property WordBool SubscriptionsEnabled;
```

Description

This is SubscriptionsEnabled, a member of class IShellNameSpaceDisp.

132.2.21.3.10 IShellNameSpaceDisp.TVFlags Property

Pascal

```
property TVFlags: LongWord;
```

C++

```
__property LongWord TVFlags;
```

Description

This is TVFlags, a member of class IShellNameSpaceDisp.

132.2.22 IShellUIHelper Interface

Class Hierarchy

Pascal

```
IShellUIHelper = interface(IDispatch);
```

C++

```
__interface IShellUIHelper : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.IShellUIHelper.

132.2.22.1 IShellUIHelper Members

The following tables list the members exposed by IShellUIHelper.

Public Methods

	Name	Description
	AddChannel (see page 19377)	This is AddChannel, a member of class IShellUIHelper.
	AddDesktopComponent (see page 19377)	This is AddDesktopComponent, a member of class IShellUIHelper.
	AddFavorite (see page 19377)	This is AddFavorite, a member of class IShellUIHelper.
	AutoCompleteAttach (see page 19378)	This is AutoCompleteAttach, a member of class IShellUIHelper.
	AutoCompleteSaveForm (see page 19378)	This is AutoCompleteSaveForm, a member of class IShellUIHelper.
	AutoScan (see page 19378)	This is AutoScan, a member of class IShellUIHelper.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	ImportExportFavorites (see page 19378)	This is ImportExportFavorites, a member of class IShellUIHelper.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	IsSubscribed (see page 19379)	This is IsSubscribed, a member of class IShellUIHelper.
	NavigateAndFind (see page 19379)	This is NavigateAndFind, a member of class IShellUIHelper.
	RefreshOfflineDesktop (see page 19379)	This is RefreshOfflineDesktop, a member of class IShellUIHelper.
	ResetFirstBootMode (see page 19379)	This is ResetFirstBootMode, a member of class IShellUIHelper.

	ResetSafeMode (see page 19379)	This is ResetSafeMode, a member of class IShellUIHelper.
	ShowBrowserUI (see page 19380)	This is ShowBrowserUI, a member of class IShellUIHelper.

132.2.22.2 IShellUIHelper Methods

The methods of the IShellUIHelper class are listed here.

Public Methods

	Name	Description
	AddChannel (see page 19377)	This is AddChannel, a member of class IShellUIHelper.
	AddDesktopComponent (see page 19377)	This is AddDesktopComponent, a member of class IShellUIHelper.
	AddFavorite (see page 19377)	This is AddFavorite, a member of class IShellUIHelper.
	AutoCompleteAttach (see page 19378)	This is AutoCompleteAttach, a member of class IShellUIHelper.
	AutoCompleteSaveForm (see page 19378)	This is AutoCompleteSaveForm, a member of class IShellUIHelper.
	AutoScan (see page 19378)	This is AutoScan, a member of class IShellUIHelper.
	ImportExportFavorites (see page 19378)	This is ImportExportFavorites, a member of class IShellUIHelper.
	IsSubscribed (see page 19379)	This is IsSubscribed, a member of class IShellUIHelper.
	NavigateAndFind (see page 19379)	This is NavigateAndFind, a member of class IShellUIHelper.
	RefreshOfflineDesktop (see page 19379)	This is RefreshOfflineDesktop, a member of class IShellUIHelper.
	ResetFirstBootMode (see page 19379)	This is ResetFirstBootMode, a member of class IShellUIHelper.
	ResetSafeMode (see page 19379)	This is ResetSafeMode, a member of class IShellUIHelper.
	ShowBrowserUI (see page 19380)	This is ShowBrowserUI, a member of class IShellUIHelper.

132.2.22.2.1 IShellUIHelper.AddChannel Method

Pascal

```
procedure AddChannel(const URL: WideString); safecall;
```

C++

```
__fastcall AddChannel(const BSTR URL);
```

Description

This is AddChannel, a member of class IShellUIHelper.

132.2.22.2.2 IShellUIHelper.AddDesktopComponent Method

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString; var Left:
OleVariant; var Top: OleVariant; var Width: OleVariant; var Height: OleVariant); safecall;
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_, OleVariant Left, OleVariant
Top, OleVariant Width, OleVariant Height);
```

Description

This is AddDesktopComponent, a member of class IShellUIHelper.

132.2.22.2.3 IShellUIHelper.AddFavorite Method

Pascal

```
procedure AddFavorite(const URL: WideString; var Title: OleVariant); safecall;
```

C++

```
__fastcall AddFavorite(const BSTR URL, OleVariant Title);
```

Description

This is AddFavorite, a member of class IShellUIHelper.

132.2.22.2.4 IShellUIHelper.AutoCompleteAttach Method

Pascal

```
procedure AutoCompleteAttach(var Reserved: OleVariant); safecall;
```

C++

```
__fastcall AutoCompleteAttach(OleVariant Reserved);
```

Description

This is AutoCompleteAttach, a member of class IShellUIHelper.

132.2.22.2.5 IShellUIHelper.AutoCompleteSaveForm Method

Pascal

```
procedure AutoCompleteSaveForm(var Form: OleVariant); safecall;
```

C++

```
__fastcall AutoCompleteSaveForm(OleVariant Form);
```

Description

This is AutoCompleteSaveForm, a member of class IShellUIHelper.

132.2.22.2.6 IShellUIHelper.AutoScan Method

Pascal

```
procedure AutoScan(const strSearch: WideString; const strFailureUrl: WideString; var
pvarTargetFrame: OleVariant); safecall;
```

C++

```
__fastcall AutoScan(const BSTR strSearch, const BSTR strFailureUrl, OleVariant
pvarTargetFrame);
```

Description

This is AutoScan, a member of class IShellUIHelper.

132.2.22.2.7 IShellUIHelper.ImportExportFavorites Method

Pascal

```
procedure ImportExportFavorites(fImport: WordBool; const strImpExpPath: WideString); safecall;
```

C++

```
__fastcall ImportExportFavorites(WordBool fImport, const BSTR strImpExpPath);
```

Description

This is ImportExportFavorites, a member of class IShellUIHelper.

132.2.22.2.8 IShellUIHelper.IsSubscribed Method

Pascal

```
function IsSubscribed(const URL: WideString): WordBool; safecall;
```

C++

```
__fastcall WordBool IsSubscribed(const BSTR URL);
```

Description

This is IsSubscribed, a member of class IShellUIHelper.

132.2.22.2.9 IShellUIHelper.NavigateAndFind Method

Pascal

```
procedure NavigateAndFind(const URL: WideString; const strQuery: WideString; var
varTargetFrame: OleVariant); safecall;
```

C++

```
__fastcall NavigateAndFind(const BSTR URL, const BSTR strQuery, OleVariant varTargetFrame);
```

Description

This is NavigateAndFind, a member of class IShellUIHelper.

132.2.22.2.10 IShellUIHelper.RefreshOfflineDesktop Method

Pascal

```
procedure RefreshOfflineDesktop; safecall;
```

C++

```
__fastcall RefreshOfflineDesktop();
```

Description

This is RefreshOfflineDesktop, a member of class IShellUIHelper.

132.2.22.2.11 IShellUIHelper.ResetFirstBootMode Method

Pascal

```
procedure ResetFirstBootMode; safecall;
```

C++

```
__fastcall ResetFirstBootMode();
```

Description

This is ResetFirstBootMode, a member of class IShellUIHelper.

132.2.22.2.12 IShellUIHelper.ResetSafeMode Method

Pascal

```
procedure ResetSafeMode; safecall;
```

C++

```
__fastcall ResetSafeMode();
```

Description

This is ResetSafeMode, a member of class IShellUIHelper.

132.2.22.2.13 IShellUIHelper.ShowBrowserUI Method

Pascal

```
function ShowBrowserUI(const bstrName: WideString; var pvarIn: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant ShowBrowserUI(const BSTR bstrName, OleVariant pvarIn);
```

Description

This is ShowBrowserUI, a member of class IShellUIHelper.

132.2.23 IShellUIHelperDisp Interface

Class Hierarchy

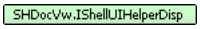

Pascal

```
IShellUIHelperDisp = dispinterface;
```

C++

```
class IShellUIHelperDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IShellUIHelperDisp.

132.2.23.1 IShellUIHelperDisp Members

The following tables list the members exposed by IShellUIHelperDisp.

Public Methods

	Name	Description
	AddChannel (see page 19381)	This is AddChannel, a member of class IShellUIHelperDisp.
	AddDesktopComponent (see page 19381)	This is AddDesktopComponent, a member of class IShellUIHelperDisp.
	AddFavorite (see page 19382)	This is AddFavorite, a member of class IShellUIHelperDisp.
	AutoCompleteAttach (see page 19382)	This is AutoCompleteAttach, a member of class IShellUIHelperDisp.
	AutoCompleteSaveForm (see page 19382)	This is AutoCompleteSaveForm, a member of class IShellUIHelperDisp.
	AutoScan (see page 19382)	This is AutoScan, a member of class IShellUIHelperDisp.
	ImportExportFavorites (see page 19382)	This is ImportExportFavorites, a member of class IShellUIHelperDisp.
	IsSubscribed (see page 19383)	This is IsSubscribed, a member of class IShellUIHelperDisp.

	NavigateAndFind (see page 19383)	This is NavigateAndFind, a member of class IShellUIHelperDisp.
	RefreshOfflineDesktop (see page 19383)	This is RefreshOfflineDesktop, a member of class IShellUIHelperDisp.
	ResetFirstBootMode (see page 19383)	This is ResetFirstBootMode, a member of class IShellUIHelperDisp.
	ResetSafeMode (see page 19384)	This is ResetSafeMode, a member of class IShellUIHelperDisp.
	ShowBrowserUI (see page 19384)	This is ShowBrowserUI, a member of class IShellUIHelperDisp.

132.2.23.2 IShellUIHelperDisp Methods

The methods of the IShellUIHelperDisp class are listed here.

Public Methods

	Name	Description
	AddChannel (see page 19381)	This is AddChannel, a member of class IShellUIHelperDisp.
	AddDesktopComponent (see page 19381)	This is AddDesktopComponent, a member of class IShellUIHelperDisp.
	AddFavorite (see page 19382)	This is AddFavorite, a member of class IShellUIHelperDisp.
	AutoCompleteAttach (see page 19382)	This is AutoCompleteAttach, a member of class IShellUIHelperDisp.
	AutoCompleteSaveForm (see page 19382)	This is AutoCompleteSaveForm, a member of class IShellUIHelperDisp.
	AutoScan (see page 19382)	This is AutoScan, a member of class IShellUIHelperDisp.
	ImportExportFavorites (see page 19382)	This is ImportExportFavorites, a member of class IShellUIHelperDisp.
	IsSubscribed (see page 19383)	This is IsSubscribed, a member of class IShellUIHelperDisp.
	NavigateAndFind (see page 19383)	This is NavigateAndFind, a member of class IShellUIHelperDisp.
	RefreshOfflineDesktop (see page 19383)	This is RefreshOfflineDesktop, a member of class IShellUIHelperDisp.
	ResetFirstBootMode (see page 19383)	This is ResetFirstBootMode, a member of class IShellUIHelperDisp.
	ResetSafeMode (see page 19384)	This is ResetSafeMode, a member of class IShellUIHelperDisp.
	ShowBrowserUI (see page 19384)	This is ShowBrowserUI, a member of class IShellUIHelperDisp.

132.2.23.2.1 IShellUIHelperDisp.AddChannel Method

Pascal

```
procedure AddChannel(const URL: WideString);
```

C++

```
__fastcall AddChannel(const BSTR URL);
```

Description

This is AddChannel, a member of class IShellUIHelperDisp.

132.2.23.2.2 IShellUIHelperDisp.AddDesktopComponent Method

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString; var Left:
OleVariant; var Top: OleVariant; var Width: OleVariant; var Height: OleVariant);
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_, OleVariant Left, OleVariant
Top, OleVariant Width, OleVariant Height);
```

Description

This is AddDesktopComponent, a member of class IShellUIHelperDisp.

132.2.23.2.3 IShellUIHelperDisp.AddFavorite Method

Pascal

```
procedure AddFavorite(const URL: WideString; var Title: OleVariant);
```

C++

```
__fastcall AddFavorite(const BSTR URL, OleVariant Title);
```

Description

This is AddFavorite, a member of class IShellUIHelperDisp.

132.2.23.2.4 IShellUIHelperDisp.AutoCompleteAttach Method

Pascal

```
procedure AutoCompleteAttach(var Reserved: OleVariant);
```

C++

```
__fastcall AutoCompleteAttach(OleVariant Reserved);
```

Description

This is AutoCompleteAttach, a member of class IShellUIHelperDisp.

132.2.23.2.5 IShellUIHelperDisp.AutoCompleteSaveForm Method

Pascal

```
procedure AutoCompleteSaveForm(var Form: OleVariant);
```

C++

```
__fastcall AutoCompleteSaveForm(OleVariant Form);
```

Description

This is AutoCompleteSaveForm, a member of class IShellUIHelperDisp.

132.2.23.2.6 IShellUIHelperDisp.AutoScan Method

Pascal

```
procedure AutoScan(const strSearch: WideString; const strFailureUrl: WideString; var
pvarTargetFrame: OleVariant);
```

C++

```
__fastcall AutoScan(const BSTR strSearch, const BSTR strFailureUrl, OleVariant
pvarTargetFrame);
```

Description

This is AutoScan, a member of class IShellUIHelperDisp.

132.2.23.2.7 IShellUIHelperDisp.ImportExportFavorites Method

Pascal

```
procedure ImportExportFavorites(fImport: WordBool; const strImpExpPath: WideString);
```

C++

```
__fastcall ImportExportFavorites(WordBool fImport, const BSTR strImpExpPath);
```

Description

This is ImportExportFavorites, a member of class IShellUIHelperDisp.

132.2.23.2.8 IShellUIHelperDisp.IsSubscribed Method

Pascal

```
function IsSubscribed(const URL: WideString): WordBool;
```

C++

```
__fastcall WordBool IsSubscribed(const BSTR URL);
```

Description

This is IsSubscribed, a member of class IShellUIHelperDisp.

132.2.23.2.9 IShellUIHelperDisp.NavigateAndFind Method

Pascal

```
procedure NavigateAndFind(const URL: WideString; const strQuery: WideString; var
varTargetFrame: OleVariant);
```

C++

```
__fastcall NavigateAndFind(const BSTR URL, const BSTR strQuery, OleVariant varTargetFrame);
```

Description

This is NavigateAndFind, a member of class IShellUIHelperDisp.

132.2.23.2.10 IShellUIHelperDisp.RefreshOfflineDesktop Method

Pascal

```
procedure RefreshOfflineDesktop;
```

C++

```
__fastcall RefreshOfflineDesktop();
```

Description

This is RefreshOfflineDesktop, a member of class IShellUIHelperDisp.

132.2.23.2.11 IShellUIHelperDisp.ResetFirstBootMode Method

Pascal

```
procedure ResetFirstBootMode;
```

C++

```
__fastcall ResetFirstBootMode();
```

Description

This is ResetFirstBootMode, a member of class IShellUIHelperDisp.

132.2.23.2.12 IShellUIHelperDisp.ResetSafeMode Method

Pascal

```
procedure ResetSafeMode;
```

C++

```
__fastcall ResetSafeMode();
```

Description

This is ResetSafeMode, a member of class IShellUIHelperDisp.

132.2.23.2.13 IShellUIHelperDisp.ShowBrowserUI Method

Pascal

```
function ShowBrowserUI(const bstrName: WideString; var pvarIn: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant ShowBrowserUI(const BSTR bstrName, OleVariant pvarIn);
```

Description

This is ShowBrowserUI, a member of class IShellUIHelperDisp.

132.2.24 IShellWindows Interface

Class Hierarchy

Pascal

```
IShellWindows = interface(IDispatch);
```

C++

```
__interface IShellWindows : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.IShellWindows.

132.2.24.1 IShellWindows Members

The following tables list the members exposed by IShellWindows.

Public Methods

	Name	Description
	_NewEnum (see page 19385)	This is _NewEnum, a member of class IShellWindows.
	FindWindowSW (see page 19385)	This is FindWindowSW, a member of class IShellWindows.
	Get_Count (see page 19386)	This is Get_Count, a member of class IShellWindows.

	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	Item (see page 19386)	This is Item, a member of class IShellWindows.
	OnActivated (see page 19386)	This is OnActivated, a member of class IShellWindows.
	OnCreated (see page 19386)	This is OnCreated, a member of class IShellWindows.
	OnNavigate (see page 19387)	This is OnNavigate, a member of class IShellWindows.
	ProcessAttachDetach (see page 19387)	This is ProcessAttachDetach, a member of class IShellWindows.
	Register (see page 19387)	This is Register, a member of class IShellWindows.
	RegisterPending (see page 19387)	This is RegisterPending, a member of class IShellWindows.
	Revoke (see page 19387)	This is Revoke, a member of class IShellWindows.

Public Properties

	Name	Description
R	Count (see page 19388)	This is Count, a member of class IShellWindows.

132.2.24.2 IShellWindows Methods

The methods of the IShellWindows class are listed here.

Public Methods

	Name	Description
	_NewEnum (see page 19385)	This is _NewEnum, a member of class IShellWindows.
	FindWindowSW (see page 19385)	This is FindWindowSW, a member of class IShellWindows.
	Get_Count (see page 19386)	This is Get_Count, a member of class IShellWindows.
	Item (see page 19386)	This is Item, a member of class IShellWindows.
	OnActivated (see page 19386)	This is OnActivated, a member of class IShellWindows.
	OnCreated (see page 19386)	This is OnCreated, a member of class IShellWindows.
	OnNavigate (see page 19387)	This is OnNavigate, a member of class IShellWindows.
	ProcessAttachDetach (see page 19387)	This is ProcessAttachDetach, a member of class IShellWindows.
	Register (see page 19387)	This is Register, a member of class IShellWindows.
	RegisterPending (see page 19387)	This is RegisterPending, a member of class IShellWindows.
	Revoke (see page 19387)	This is Revoke, a member of class IShellWindows.

132.2.24.2.1 IShellWindows._NewEnum Method

Pascal

```
function _NewEnum: IUnknown; safecall;
```

C++

```
__fastcall IUnknown _NewEnum();
```

Description

This is _NewEnum, a member of class IShellWindows.

132.2.24.2.2 IShellWindows.FindWindowSW Method

Pascal

```
function FindWindowSW(var pvarloc: OleVariant; var pvarlocRoot: OleVariant; swClass: SYSINT;
out pHWND: Integer; swfwOptions: SYSINT): IDispatch; safecall;
```

C++

```
__fastcall IDispatch FindWindowSW(OleVariant pvarloc, OleVariant pvarlocRoot, SYSINT swClass,
int pHWND, SYSINT swfwOptions);
```

Description

This is FindWindowSW, a member of class IShellWindows.

132.2.24.2.3 IShellWindows.Get_Count Method

Pascal

```
function Get_Count: Integer; safecall;
```

C++

```
__fastcall int Get_Count();
```

Description

This is Get_Count, a member of class IShellWindows.

132.2.24.2.4 IShellWindows.Item Method

Pascal

```
function Item(index: OleVariant): IDispatch; safecall;
```

C++

```
__fastcall IDispatch Item(OleVariant index);
```

Description

This is Item, a member of class IShellWindows.

132.2.24.2.5 IShellWindows.OnActivated Method

Pascal

```
procedure OnActivated(lCookie: Integer; fActive: WordBool); safecall;
```

C++

```
__fastcall OnActivated(int lCookie, WordBool fActive);
```

Description

This is OnActivated, a member of class IShellWindows.

132.2.24.2.6 IShellWindows.OnCreated Method

Pascal

```
procedure OnCreated(lCookie: Integer; const punk: IUnknown); safecall;
```

C++

```
__fastcall OnCreated(int lCookie, const IUnknown punk);
```

Description

This is OnCreated, a member of class IShellWindows.

132.2.24.2.7 IShellWindows.OnNavigate Method

Pascal

```
procedure OnNavigate(lCookie: Integer; var pvarloc: OleVariant); safecall;
```

C++

```
__fastcall OnNavigate(int lCookie, OleVariant pvarloc);
```

Description

This is OnNavigate, a member of class IShellWindows.

132.2.24.2.8 IShellWindows.ProcessAttachDetach Method

Pascal

```
procedure ProcessAttachDetach(fAttach: WordBool); safecall;
```

C++

```
__fastcall ProcessAttachDetach(WordBool fAttach);
```

Description

This is ProcessAttachDetach, a member of class IShellWindows.

132.2.24.2.9 IShellWindows.Register Method

Pascal

```
procedure Register(const pid: IDispatch; hWnd: Integer; swClass: SYSINT; out plCookie:
Integer); safecall;
```

C++

```
__fastcall Register(const IDispatch pid, int hWnd, SYSINT swClass, int plCookie);
```

Description

This is Register, a member of class IShellWindows.

132.2.24.2.10 IShellWindows.RegisterPending Method

Pascal

```
procedure RegisterPending(lThreadId: Integer; var pvarloc: OleVariant; var pvarlocRoot:
OleVariant; swClass: SYSINT; out plCookie: Integer); safecall;
```

C++

```
__fastcall RegisterPending(int lThreadId, OleVariant pvarloc, OleVariant pvarlocRoot, SYSINT
swClass, int plCookie);
```

Description

This is RegisterPending, a member of class IShellWindows.

132.2.24.2.11 IShellWindows.Revoke Method

Pascal

```
procedure Revoke(lCookie: Integer); safecall;
```

C++

```
__fastcall Revoke(int lCookie);
```

Description

This is Revoke, a member of class IShellWindows.

132.2.24.3 IShellWindows Properties

The properties of the IShellWindows class are listed here.

Public Properties

	Name	Description
R	Count (see page 19388)	This is Count, a member of class IShellWindows.

132.2.24.3.1 IShellWindows.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class IShellWindows.

132.2.25 IShellWindowsDisp Interface

Class Hierarchy

SHDocVw.IShellWindowsDisp

Pascal

```
IShellWindowsDisp = dispinterface;
```

C++

```
class IShellWindowsDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IShellWindowsDisp.

132.2.25.1 IShellWindowsDisp Members

The following tables list the members exposed by IShellWindowsDisp.

Public Methods

	Name	Description
	_NewEnum (see page 19389)	This is _NewEnum, a member of class IShellWindowsDisp.
	FindWindowSW (see page 19389)	This is FindWindowSW, a member of class IShellWindowsDisp.
	Item (see page 19390)	This is Item, a member of class IShellWindowsDisp.
	OnActivated (see page 19390)	This is OnActivated, a member of class IShellWindowsDisp.
	OnCreated (see page 19390)	This is OnCreated, a member of class IShellWindowsDisp.
	OnNavigate (see page 19390)	This is OnNavigate, a member of class IShellWindowsDisp.
	ProcessAttachDetach (see page 19391)	This is ProcessAttachDetach, a member of class IShellWindowsDisp.
	Register (see page 19391)	This is Register, a member of class IShellWindowsDisp.
	RegisterPending (see page 19391)	This is RegisterPending, a member of class IShellWindowsDisp.
	Revoke (see page 19391)	This is Revoke, a member of class IShellWindowsDisp.

Public Properties

	Name	Description
	Count (see page 19392)	This is Count, a member of class IShellWindowsDisp.

132.2.25.2 IShellWindowsDisp Methods

The methods of the IShellWindowsDisp class are listed here.

Public Methods

	Name	Description
	_NewEnum (see page 19389)	This is _NewEnum, a member of class IShellWindowsDisp.
	FindWindowSW (see page 19389)	This is FindWindowSW, a member of class IShellWindowsDisp.
	Item (see page 19390)	This is Item, a member of class IShellWindowsDisp.
	OnActivated (see page 19390)	This is OnActivated, a member of class IShellWindowsDisp.
	OnCreated (see page 19390)	This is OnCreated, a member of class IShellWindowsDisp.
	OnNavigate (see page 19390)	This is OnNavigate, a member of class IShellWindowsDisp.
	ProcessAttachDetach (see page 19391)	This is ProcessAttachDetach, a member of class IShellWindowsDisp.
	Register (see page 19391)	This is Register, a member of class IShellWindowsDisp.
	RegisterPending (see page 19391)	This is RegisterPending, a member of class IShellWindowsDisp.
	Revoke (see page 19391)	This is Revoke, a member of class IShellWindowsDisp.

132.2.25.2.1 IShellWindowsDisp._NewEnum Method

Pascal

```
function _NewEnum: IUnknown;
```

C++

```
__fastcall IUnknown _NewEnum();
```

Description

This is _NewEnum, a member of class IShellWindowsDisp.

132.2.25.2.2 IShellWindowsDisp.FindWindowSW Method

Pascal

```
function FindWindowSW(var pvarloc: OleVariant; var pvarlocRoot: OleVariant; swClass: SYSINT;
out pHWND: Integer; swfwOptions: SYSINT): IDispatch;
```

C++

```
__fastcall IDispatch FindWindowSW(OleVariant pvarloc, OleVariant pvarlocRoot, SYSINT swClass,
int pHWND, SYSINT swfwOptions);
```

Description

This is FindWindowSW, a member of class IShellWindowsDisp.

132.2.25.2.3 IShellWindowsDisp.Item Method

Pascal

```
function Item(index: OleVariant): IDispatch;
```

C++

```
__fastcall IDispatch Item(OleVariant index);
```

Description

This is Item, a member of class IShellWindowsDisp.

132.2.25.2.4 IShellWindowsDisp.OnActivated Method

Pascal

```
procedure OnActivated(lCookie: Integer; fActive: WordBool);
```

C++

```
__fastcall OnActivated(int lCookie, WordBool fActive);
```

Description

This is OnActivated, a member of class IShellWindowsDisp.

132.2.25.2.5 IShellWindowsDisp.OnCreated Method

Pascal

```
procedure OnCreated(lCookie: Integer; const punk: IUnknown);
```

C++

```
__fastcall OnCreated(int lCookie, const IUnknown punk);
```

Description

This is OnCreated, a member of class IShellWindowsDisp.

132.2.25.2.6 IShellWindowsDisp.OnNavigate Method

Pascal

```
procedure OnNavigate(lCookie: Integer; var pvarloc: OleVariant);
```

C++

```
__fastcall OnNavigate(int lCookie, OleVariant pvarloc);
```

Description

This is OnNavigate, a member of class IShellWindowsDisp.

132.2.25.2.7 IShellWindowsDisp.ProcessAttachDetach Method

Pascal

```
procedure ProcessAttachDetach(fAttach: WordBool);
```

C++

```
__fastcall ProcessAttachDetach(WordBool fAttach);
```

Description

This is ProcessAttachDetach, a member of class IShellWindowsDisp.

132.2.25.2.8 IShellWindowsDisp.Register Method

Pascal

```
procedure Register(const pid: IDispatch; hWnd: Integer; swClass: SYSINT; out plCookie:
Integer);
```

C++

```
__fastcall Register(const IDispatch pid, int hWnd, SYSINT swClass, int plCookie);
```

Description

This is Register, a member of class IShellWindowsDisp.

132.2.25.2.9 IShellWindowsDisp.RegisterPending Method

Pascal

```
procedure RegisterPending(lThreadId: Integer; var pvarloc: OleVariant; var pvarlocRoot:
OleVariant; swClass: SYSINT; out plCookie: Integer);
```

C++

```
__fastcall RegisterPending(int lThreadId, OleVariant pvarloc, OleVariant pvarlocRoot, SYSINT
swClass, int plCookie);
```

Description

This is RegisterPending, a member of class IShellWindowsDisp.

132.2.25.2.10 IShellWindowsDisp.Revoke Method

Pascal

```
procedure Revoke(lCookie: Integer);
```

C++

```
__fastcall Revoke(int lCookie);
```

Description

This is Revoke, a member of class IShellWindowsDisp.

132.2.25.3 IShellWindowsDisp Properties

The properties of the IShellWindowsDisp class are listed here.

Public Properties

	Name	Description
	Count (see page 19392)	This is Count, a member of class IShellWindowsDisp.

132.2.25.3.1 IShellWindowsDisp.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class IShellWindowsDisp.

132.2.26 IWebBrowser Interface

Class Hierarchy

Pascal

```
IWebBrowser = interface(IDispatch);
```

C++

```
__interface IWebBrowser : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.IWebBrowser.

132.2.26.1 IWebBrowser Members

The following tables list the members exposed by IWebBrowser.

Public Methods

	Name	Description
	Get_Application (see page 19394)	This is Get_Application, a member of class IWebBrowser.
	Get_Busy (see page 19394)	This is Get_Busy, a member of class IWebBrowser.
	Get_Container (see page 19394)	This is Get_Container, a member of class IWebBrowser.
	Get_Document (see page 19394)	This is Get_Document, a member of class IWebBrowser.
	Get_Height (see page 19395)	This is Get_Height, a member of class IWebBrowser.
	Get_Left (see page 19395)	This is Get_Left, a member of class IWebBrowser.
	Get_LocationName (see page 19395)	This is Get_LocationName, a member of class IWebBrowser.
	Get_LocationURL (see page 19395)	This is Get_LocationURL, a member of class IWebBrowser.
	Get_Parent (see page 19396)	This is Get_Parent, a member of class IWebBrowser.
	Get_Top (see page 19396)	This is Get_Top, a member of class IWebBrowser.
	Get_TopLevelContainer (see page 19396)	This is Get_TopLevelContainer, a member of class IWebBrowser.

	Get_type_ (see page 19396)	This is Get_type_, a member of class IWebBrowser.
	Get_Width (see page 19396)	This is Get_Width, a member of class IWebBrowser.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	GoBack (see page 19397)	This is GoBack, a member of class IWebBrowser.
	GoForward (see page 19397)	This is GoForward, a member of class IWebBrowser.
	GoHome (see page 19397)	This is GoHome, a member of class IWebBrowser.
	GoSearch (see page 19397)	This is GoSearch, a member of class IWebBrowser.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	Navigate (see page 19398)	This is Navigate, a member of class IWebBrowser.
	Refresh (see page 19398)	This is Refresh, a member of class IWebBrowser.
	Refresh2 (see page 19398)	This is Refresh2, a member of class IWebBrowser.
	Set_Height (see page 19398)	This is Set_Height, a member of class IWebBrowser.
	Set_Left (see page 19398)	This is Set_Left, a member of class IWebBrowser.
	Set_Top (see page 19399)	This is Set_Top, a member of class IWebBrowser.
	Set_Width (see page 19399)	This is Set_Width, a member of class IWebBrowser.
	Stop (see page 19399)	This is Stop, a member of class IWebBrowser.

Public Properties

	Name	Description
R	Application (see page 19400)	This is Application, a member of class IWebBrowser.
R	Busy (see page 19400)	This is Busy, a member of class IWebBrowser.
R	Container (see page 19400)	This is Container, a member of class IWebBrowser.
R	Document (see page 19400)	This is Document, a member of class IWebBrowser.
	Height (see page 19401)	This is Height, a member of class IWebBrowser.
	Left (see page 19401)	This is Left, a member of class IWebBrowser.
R	LocationName (see page 19401)	This is LocationName, a member of class IWebBrowser.
R	LocationURL (see page 19401)	This is LocationURL, a member of class IWebBrowser.
R	Parent (see page 19401)	This is Parent, a member of class IWebBrowser.
	Top (see page 19402)	This is Top, a member of class IWebBrowser.
R	TopLevelContainer (see page 19402)	This is TopLevelContainer, a member of class IWebBrowser.
R	type_ (see page 19402)	This is type_, a member of class IWebBrowser.
	Width (see page 19402)	This is Width, a member of class IWebBrowser.

132.2.26.2 IWebBrowser Methods

The methods of the IWebBrowser class are listed here.

Public Methods

	Name	Description
	Get_Application (see page 19394)	This is Get_Application, a member of class IWebBrowser.
	Get_Busy (see page 19394)	This is Get_Busy, a member of class IWebBrowser.
	Get_Container (see page 19394)	This is Get_Container, a member of class IWebBrowser.
	Get_Document (see page 19394)	This is Get_Document, a member of class IWebBrowser.
	Get_Height (see page 19395)	This is Get_Height, a member of class IWebBrowser.
	Get_Left (see page 19395)	This is Get_Left, a member of class IWebBrowser.
	Get_LocationName (see page 19395)	This is Get_LocationName, a member of class IWebBrowser.
	Get_LocationURL (see page 19395)	This is Get_LocationURL, a member of class IWebBrowser.
	Get_Parent (see page 19396)	This is Get_Parent, a member of class IWebBrowser.
	Get_Top (see page 19396)	This is Get_Top, a member of class IWebBrowser.
	Get_TopLevelContainer (see page 19396)	This is Get_TopLevelContainer, a member of class IWebBrowser.
	Get_type_ (see page 19396)	This is Get_type_, a member of class IWebBrowser.

	Get_Width (see page 19396)	This is Get_Width, a member of class IWebBrowser.
	GoBack (see page 19397)	This is GoBack, a member of class IWebBrowser.
	GoForward (see page 19397)	This is GoForward, a member of class IWebBrowser.
	GoHome (see page 19397)	This is GoHome, a member of class IWebBrowser.
	GoSearch (see page 19397)	This is GoSearch, a member of class IWebBrowser.
	Navigate (see page 19398)	This is Navigate, a member of class IWebBrowser.
	Refresh (see page 19398)	This is Refresh, a member of class IWebBrowser.
	Refresh2 (see page 19398)	This is Refresh2, a member of class IWebBrowser.
	Set_Height (see page 19398)	This is Set_Height, a member of class IWebBrowser.
	Set_Left (see page 19398)	This is Set_Left, a member of class IWebBrowser.
	Set_Top (see page 19399)	This is Set_Top, a member of class IWebBrowser.
	Set_Width (see page 19399)	This is Set_Width, a member of class IWebBrowser.
	Stop (see page 19399)	This is Stop, a member of class IWebBrowser.

132.2.26.2.1 IWebBrowser.Get_Application Method

Pascal

```
function Get_Application: IDispatch; safecall;
```

C++

```
__fastcall IDispatch Get_Application();
```

Description

This is Get_Application, a member of class IWebBrowser.

132.2.26.2.2 IWebBrowser.Get_Busy Method

Pascal

```
function Get_Busy: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_Busy();
```

Description

This is Get_Busy, a member of class IWebBrowser.

132.2.26.2.3 IWebBrowser.Get_Container Method

Pascal

```
function Get_Container: IDispatch; safecall;
```

C++

```
__fastcall IDispatch Get_Container();
```

Description

This is Get_Container, a member of class IWebBrowser.

132.2.26.2.4 IWebBrowser.Get_Document Method

Pascal

```
function Get_Document: IDispatch; safecall;
```

C++

```
__fastcall IDispatch Get_Document();
```

Description

This is Get_Document, a member of class IWebBrowser.

132.2.26.2.5 IWebBrowser.Get_Height Method

Pascal

```
function Get_Height: Integer; safecall;
```

C++

```
__fastcall int Get_Height();
```

Description

This is Get_Height, a member of class IWebBrowser.

132.2.26.2.6 IWebBrowser.Get_Left Method

Pascal

```
function Get_Left: Integer; safecall;
```

C++

```
__fastcall int Get_Left();
```

Description

This is Get_Left, a member of class IWebBrowser.

132.2.26.2.7 IWebBrowser.Get_LocationName Method

Pascal

```
function Get_LocationName: WideString; safecall;
```

C++

```
__fastcall BSTR Get_LocationName();
```

Description

This is Get_LocationName, a member of class IWebBrowser.

132.2.26.2.8 IWebBrowser.Get_LocationURL Method

Pascal

```
function Get_LocationURL: WideString; safecall;
```

C++

```
__fastcall BSTR Get_LocationURL();
```

Description

This is Get_LocationURL, a member of class IWebBrowser.

132.2.26.2.9 IWebBrowser.Get_Parent Method

Pascal

```
function Get_Parent: IDispatch; safecall;
```

C++

```
__fastcall IDispatch Get_Parent();
```

Description

This is Get_Parent, a member of class IWebBrowser.

132.2.26.2.10 IWebBrowser.Get_Top Method

Pascal

```
function Get_Top: Integer; safecall;
```

C++

```
__fastcall int Get_Top();
```

Description

This is Get_Top, a member of class IWebBrowser.

132.2.26.2.11 IWebBrowser.Get_TopLevelContainer Method

Pascal

```
function Get_TopLevelContainer: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_TopLevelContainer();
```

Description

This is Get_TopLevelContainer, a member of class IWebBrowser.

132.2.26.2.12 IWebBrowser.Get_type_ Method

Pascal

```
function Get_type_: WideString; safecall;
```

C++

```
__fastcall BSTR Get_type_();
```

Description

This is Get_type_, a member of class IWebBrowser.

132.2.26.2.13 IWebBrowser.Get_Width Method

Pascal

```
function Get_Width: Integer; safecall;
```

C++

```
__fastcall int Get_Width();
```

Description

This is Get_Width, a member of class IWebBrowser.

132.2.26.2.14 IWebBrowser.GoBack Method

Pascal

```
procedure GoBack; safecall;
```

C++

```
__fastcall GoBack();
```

Description

This is GoBack, a member of class IWebBrowser.

132.2.26.2.15 IWebBrowser.GoForward Method

Pascal

```
procedure GoForward; safecall;
```

C++

```
__fastcall GoForward();
```

Description

This is GoForward, a member of class IWebBrowser.

132.2.26.2.16 IWebBrowser.GoHome Method

Pascal

```
procedure GoHome; safecall;
```

C++

```
__fastcall GoHome();
```

Description

This is GoHome, a member of class IWebBrowser.

132.2.26.2.17 IWebBrowser.GoSearch Method

Pascal

```
procedure GoSearch; safecall;
```

C++

```
__fastcall GoSearch();
```

Description

This is GoSearch, a member of class IWebBrowser.

132.2.26.2.18 IWebBrowser.Navigate Method

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant); safecall;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate, a member of class IWebBrowser.

132.2.26.2.19 IWebBrowser.Refresh Method

Pascal

```
procedure Refresh; safecall;
```

C++

```
__fastcall Refresh();
```

Description

This is Refresh, a member of class IWebBrowser.

132.2.26.2.20 IWebBrowser.Refresh2 Method

Pascal

```
procedure Refresh2(var Level: OleVariant); safecall;
```

C++

```
__fastcall Refresh2(OleVariant Level);
```

Description

This is Refresh2, a member of class IWebBrowser.

132.2.26.2.21 IWebBrowser.Set_Height Method

Pascal

```
procedure Set_Height(pl: Integer); safecall;
```

C++

```
__fastcall Set_Height(int pl);
```

Description

This is Set_Height, a member of class IWebBrowser.

132.2.26.2.22 IWebBrowser.Set_Left Method

Pascal

```
procedure Set_Left(pl: Integer); safecall;
```

C++

```
__fastcall Set_Left(int pl);
```

Description

This is Set_Left, a member of class IWebBrowser.

132.2.26.2.23 IWebBrowser.Set_Top Method

Pascal

```
procedure Set_Top(pl: Integer); safecall;
```

C++

```
__fastcall Set_Top(int pl);
```

Description

This is Set_Top, a member of class IWebBrowser.

132.2.26.2.24 IWebBrowser.Set_Width Method

Pascal

```
procedure Set_Width(pl: Integer); safecall;
```

C++

```
__fastcall Set_Width(int pl);
```

Description

This is Set_Width, a member of class IWebBrowser.

132.2.26.2.25 IWebBrowser.Stop Method

Pascal

```
procedure Stop; safecall;
```

C++

```
__fastcall Stop();
```

Description

This is Stop, a member of class IWebBrowser.

132.2.26.3 IWebBrowser Properties

The properties of the IWebBrowser class are listed here.

Public Properties

	Name	Description
R	Application (see page 19400)	This is Application, a member of class IWebBrowser.
R	Busy (see page 19400)	This is Busy, a member of class IWebBrowser.
R	Container (see page 19400)	This is Container, a member of class IWebBrowser.
R	Document (see page 19400)	This is Document, a member of class IWebBrowser.
	Height (see page 19401)	This is Height, a member of class IWebBrowser.

	Left (see page 19401)	This is Left, a member of class IWebBrowser.
R	LocationName (see page 19401)	This is LocationName, a member of class IWebBrowser.
R	LocationURL (see page 19401)	This is LocationURL, a member of class IWebBrowser.
R	Parent (see page 19401)	This is Parent, a member of class IWebBrowser.
	Top (see page 19402)	This is Top, a member of class IWebBrowser.
R	TopLevelContainer (see page 19402)	This is TopLevelContainer, a member of class IWebBrowser.
R	type_ (see page 19402)	This is type_, a member of class IWebBrowser.
	Width (see page 19402)	This is Width, a member of class IWebBrowser.

132.2.26.3.1 IWebBrowser.Application Property

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

This is Application, a member of class IWebBrowser.

132.2.26.3.2 IWebBrowser.Busy Property

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

This is Busy, a member of class IWebBrowser.

132.2.26.3.3 IWebBrowser.Container Property

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

This is Container, a member of class IWebBrowser.

132.2.26.3.4 IWebBrowser.Document Property

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

This is Document, a member of class IWebBrowser.

132.2.26.3.5 IWebBrowser.Height Property

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

This is Height, a member of class IWebBrowser.

132.2.26.3.6 IWebBrowser.Left Property

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

This is Left, a member of class IWebBrowser.

132.2.26.3.7 IWebBrowser.LocationName Property

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

This is LocationName, a member of class IWebBrowser.

132.2.26.3.8 IWebBrowser.LocationURL Property

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

This is LocationURL, a member of class IWebBrowser.

132.2.26.3.9 IWebBrowser.Parent Property

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

This is Parent, a member of class IWebBrowser.

132.2.26.3.10 IWebBrowser.Top Property

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

This is Top, a member of class IWebBrowser.

132.2.26.3.11 IWebBrowser.TopLevelContainer Property

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

This is TopLevelContainer, a member of class IWebBrowser.

132.2.26.3.12 IWebBrowser.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class IWebBrowser.

132.2.26.3.13 IWebBrowser.Width Property

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

This is Width, a member of class IWebBrowser.

132.2.27 IWebBrowser2 Interface

Class Hierarchy

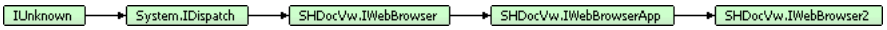

Pascal

```
IWebBrowser2 = interface(IWebBrowserApp);
```

C++

```
__interface IWebBrowser2 : public IWebBrowserApp;
```

File

SHDocVw

Description

This is class SHDocVw.IWebBrowser2.

132.2.27.1 IWebBrowser2 Members

The following tables list the members exposed by IWebBrowser2.

Public Methods

	Name	Description
	ClientToWindow (see page 19427)	This is ClientToWindow, a member of class IWebBrowserApp.
	ExecWB (see page 19405)	This is ExecWB, a member of class IWebBrowser2.
	Get_AddressBar (see page 19406)	This is Get_AddressBar, a member of class IWebBrowser2.
	Get_Application (see page 19394)	This is Get_Application, a member of class IWebBrowser.
	Get_Busy (see page 19394)	This is Get_Busy, a member of class IWebBrowser.
	Get_Container (see page 19394)	This is Get_Container, a member of class IWebBrowser.
	Get_Document (see page 19394)	This is Get_Document, a member of class IWebBrowser.
	Get_FullName (see page 19427)	This is Get_FullName, a member of class IWebBrowserApp.
	Get_FullScreen (see page 19428)	This is Get_FullScreen, a member of class IWebBrowserApp.
	Get_Height (see page 19395)	This is Get_Height, a member of class IWebBrowser.
	Get_HWnd (see page 19428)	This is Get_HWnd, a member of class IWebBrowserApp.
	Get_Left (see page 19395)	This is Get_Left, a member of class IWebBrowser.
	Get_LocationName (see page 19395)	This is Get_LocationName, a member of class IWebBrowser.
	Get_LocationURL (see page 19395)	This is Get_LocationURL, a member of class IWebBrowser.
	Get_MenuBar (see page 19428)	This is Get_MenuBar, a member of class IWebBrowserApp.
	Get_Name (see page 19428)	This is Get_Name, a member of class IWebBrowserApp.
	Get_Offline (see page 19406)	This is Get_Offline, a member of class IWebBrowser2.
	Get_Parent (see page 19396)	This is Get_Parent, a member of class IWebBrowser.
	Get_Path (see page 19429)	This is Get_Path, a member of class IWebBrowserApp.
	Get_ReadyState (see page 19406)	This is Get_ReadyState, a member of class IWebBrowser2.
	Get_RegisterAsBrowser (see page 19406)	This is Get_RegisterAsBrowser, a member of class IWebBrowser2.
	Get_RegisterAsDropTarget (see page 19407)	This is Get_RegisterAsDropTarget, a member of class IWebBrowser2.
	Get_Resizable (see page 19407)	This is Get_Resizable, a member of class IWebBrowser2.
	Get_Silent (see page 19407)	This is Get_Silent, a member of class IWebBrowser2.
	Get_StatusBar (see page 19429)	This is Get_StatusBar, a member of class IWebBrowserApp.
	Get_StatusText (see page 19429)	This is Get_StatusText, a member of class IWebBrowserApp.
	Get_TheaterMode (see page 19407)	This is Get_TheaterMode, a member of class IWebBrowser2.

	Get_ToolBar (see page 19429)	This is Get_ToolBar, a member of class IWebBrowserApp.
	Get_Top (see page 19396)	This is Get_Top, a member of class IWebBrowser.
	Get_TopLevelContainer (see page 19396)	This is Get_TopLevelContainer, a member of class IWebBrowser.
	Get_type_ (see page 19396)	This is Get_type_, a member of class IWebBrowser.
	Get_Visible (see page 19429)	This is Get_Visible, a member of class IWebBrowserApp.
	Get_Width (see page 19396)	This is Get_Width, a member of class IWebBrowser.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetProperty (see page 19430)	This is GetProperty, a member of class IWebBrowserApp.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	GoBack (see page 19397)	This is GoBack, a member of class IWebBrowser.
	GoForward (see page 19397)	This is GoForward, a member of class IWebBrowser.
	GoHome (see page 19397)	This is GoHome, a member of class IWebBrowser.
	GoSearch (see page 19397)	This is GoSearch, a member of class IWebBrowser.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	Navigate (see page 19398)	This is Navigate, a member of class IWebBrowser.
	Navigate2 (see page 19407)	This is Navigate2, a member of class IWebBrowser2.
	PutProperty (see page 19430)	This is PutProperty, a member of class IWebBrowserApp.
	QueryStatusWB (see page 19408)	This is QueryStatusWB, a member of class IWebBrowser2.
	Quit (see page 19430)	This is Quit, a member of class IWebBrowserApp.
	Refresh (see page 19398)	This is Refresh, a member of class IWebBrowser.
	Refresh2 (see page 19398)	This is Refresh2, a member of class IWebBrowser.
	Set_AddressBar (see page 19408)	This is Set_AddressBar, a member of class IWebBrowser2.
	Set_FullScreen (see page 19430)	This is Set_FullScreen, a member of class IWebBrowserApp.
	Set_Height (see page 19398)	This is Set_Height, a member of class IWebBrowser.
	Set_Left (see page 19398)	This is Set_Left, a member of class IWebBrowser.
	Set_MenuBar (see page 19431)	This is Set_MenuBar, a member of class IWebBrowserApp.
	Set_Offline (see page 19408)	This is Set_Offline, a member of class IWebBrowser2.
	Set_RegisterAsBrowser (see page 19408)	This is Set_RegisterAsBrowser, a member of class IWebBrowser2.
	Set_RegisterAsDropTarget (see page 19409)	This is Set_RegisterAsDropTarget, a member of class IWebBrowser2.
	Set_Resizable (see page 19409)	This is Set_Resizable, a member of class IWebBrowser2.
	Set_Silent (see page 19409)	This is Set_Silent, a member of class IWebBrowser2.
	Set_StatusBar (see page 19431)	This is Set_StatusBar, a member of class IWebBrowserApp.
	Set_StatusText (see page 19431)	This is Set_StatusText, a member of class IWebBrowserApp.
	Set_TheaterMode (see page 19409)	This is Set_TheaterMode, a member of class IWebBrowser2.
	Set_ToolBar (see page 19431)	This is Set_ToolBar, a member of class IWebBrowserApp.
	Set_Top (see page 19399)	This is Set_Top, a member of class IWebBrowser.
	Set_Visible (see page 19431)	This is Set_Visible, a member of class IWebBrowserApp.
	Set_Width (see page 19399)	This is Set_Width, a member of class IWebBrowser.
	ShowBrowserBar (see page 19409)	This is ShowBrowserBar, a member of class IWebBrowser2.
	Stop (see page 19399)	This is Stop, a member of class IWebBrowser.

Public Properties

	Name	Description
	AddressBar (see page 19410)	This is AddressBar, a member of class IWebBrowser2.
R	Application (see page 19400)	This is Application, a member of class IWebBrowser.
R	Busy (see page 19400)	This is Busy, a member of class IWebBrowser.
R	Container (see page 19400)	This is Container, a member of class IWebBrowser.
R	Document (see page 19400)	This is Document, a member of class IWebBrowser.
R	FullName (see page 19432)	This is FullName, a member of class IWebBrowserApp.
	FullScreen (see page 19432)	This is FullScreen, a member of class IWebBrowserApp.
	Height (see page 19401)	This is Height, a member of class IWebBrowser.
R	HWnd (see page 19432)	This is HWnd, a member of class IWebBrowserApp.
	Left (see page 19401)	This is Left, a member of class IWebBrowser.

R	LocationName (see page 19401)	This is LocationName, a member of class IWebBrowser.
R	LocationURL (see page 19401)	This is LocationURL, a member of class IWebBrowser.
	MenuBar (see page 19433)	This is MenuBar, a member of class IWebBrowserApp.
R	Name (see page 19433)	This is Name, a member of class IWebBrowserApp.
	Offline (see page 19410)	This is Offline, a member of class IWebBrowser2.
R	Parent (see page 19401)	This is Parent, a member of class IWebBrowser.
R	Path (see page 19433)	This is Path, a member of class IWebBrowserApp.
R	ReadyState (see page 19410)	This is ReadyState, a member of class IWebBrowser2.
	RegisterAsBrowser (see page 19411)	This is RegisterAsBrowser, a member of class IWebBrowser2.
	RegisterAsDropTarget (see page 19411)	This is RegisterAsDropTarget, a member of class IWebBrowser2.
	Resizable (see page 19411)	This is Resizable, a member of class IWebBrowser2.
	Silent (see page 19411)	This is Silent, a member of class IWebBrowser2.
	StatusBar (see page 19433)	This is StatusBar, a member of class IWebBrowserApp.
	StatusText (see page 19434)	This is StatusText, a member of class IWebBrowserApp.
	TheaterMode (see page 19412)	This is TheaterMode, a member of class IWebBrowser2.
	ToolBar (see page 19434)	This is ToolBar, a member of class IWebBrowserApp.
	Top (see page 19402)	This is Top, a member of class IWebBrowser.
R	TopLevelContainer (see page 19402)	This is TopLevelContainer, a member of class IWebBrowser.
R	type_ (see page 19402)	This is type_, a member of class IWebBrowser.
	Visible (see page 19434)	This is Visible, a member of class IWebBrowserApp.
	Width (see page 19402)	This is Width, a member of class IWebBrowser.

132.2.27.2 IWebBrowser2 Methods

The methods of the IWebBrowser2 class are listed here.

Public Methods

	Name	Description
	ExecWB (see page 19405)	This is ExecWB, a member of class IWebBrowser2.
	Get_AddressBar (see page 19406)	This is Get_AddressBar, a member of class IWebBrowser2.
	Get_Offline (see page 19406)	This is Get_Offline, a member of class IWebBrowser2.
	Get_ReadyState (see page 19406)	This is Get_ReadyState, a member of class IWebBrowser2.
	Get_RegisterAsBrowser (see page 19406)	This is Get_RegisterAsBrowser, a member of class IWebBrowser2.
	Get_RegisterAsDropTarget (see page 19407)	This is Get_RegisterAsDropTarget, a member of class IWebBrowser2.
	Get_Resizable (see page 19407)	This is Get_Resizable, a member of class IWebBrowser2.
	Get_Silent (see page 19407)	This is Get_Silent, a member of class IWebBrowser2.
	Get_TheaterMode (see page 19407)	This is Get_TheaterMode, a member of class IWebBrowser2.
	Navigate2 (see page 19407)	This is Navigate2, a member of class IWebBrowser2.
	QueryStatusWB (see page 19408)	This is QueryStatusWB, a member of class IWebBrowser2.
	Set_AddressBar (see page 19408)	This is Set_AddressBar, a member of class IWebBrowser2.
	Set_Offline (see page 19408)	This is Set_Offline, a member of class IWebBrowser2.
	Set_RegisterAsBrowser (see page 19408)	This is Set_RegisterAsBrowser, a member of class IWebBrowser2.
	Set_RegisterAsDropTarget (see page 19409)	This is Set_RegisterAsDropTarget, a member of class IWebBrowser2.
	Set_Resizable (see page 19409)	This is Set_Resizable, a member of class IWebBrowser2.
	Set_Silent (see page 19409)	This is Set_Silent, a member of class IWebBrowser2.
	Set_TheaterMode (see page 19409)	This is Set_TheaterMode, a member of class IWebBrowser2.
	ShowBrowserBar (see page 19409)	This is ShowBrowserBar, a member of class IWebBrowser2.

132.2.27.2.1 IWebBrowser2.ExecWB Method

Pascal

```
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT; var pvaIn: OleVariant; var
pvaOut: OleVariant); safecall;
```

C++

```
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, OleVariant pvaIn, OleVariant
pvaOut);
```

Description

This is ExecWB, a member of class IWebBrowser2.

132.2.27.2.2 IWebBrowser2.Get_AddressBar Method

Pascal

```
function Get_AddressBar: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_AddressBar();
```

Description

This is Get_AddressBar, a member of class IWebBrowser2.

132.2.27.2.3 IWebBrowser2.Get_Offline Method

Pascal

```
function Get_Offline: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_Offline();
```

Description

This is Get_Offline, a member of class IWebBrowser2.

132.2.27.2.4 IWebBrowser2.Get_ReadyState Method

Pascal

```
function Get_ReadyState: tagREADYSTATE; safecall;
```

C++

```
__fastcall tagREADYSTATE Get_ReadyState();
```

Description

This is Get_ReadyState, a member of class IWebBrowser2.

132.2.27.2.5 IWebBrowser2.Get_RegisterAsBrowser Method

Pascal

```
function Get_RegisterAsBrowser: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_RegisterAsBrowser();
```

Description

This is Get_RegisterAsBrowser, a member of class IWebBrowser2.

132.2.27.2.6 IWebBrowser2.Get_RegisterAsDropTarget Method

Pascal

```
function Get_RegisterAsDropTarget: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_RegisterAsDropTarget();
```

Description

This is Get_RegisterAsDropTarget, a member of class IWebBrowser2.

132.2.27.2.7 IWebBrowser2.Get_Resizable Method

Pascal

```
function Get_Resizable: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_Resizable();
```

Description

This is Get_Resizable, a member of class IWebBrowser2.

132.2.27.2.8 IWebBrowser2.Get_Silent Method

Pascal

```
function Get_Silent: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_Silent();
```

Description

This is Get_Silent, a member of class IWebBrowser2.

132.2.27.2.9 IWebBrowser2.Get_TheaterMode Method

Pascal

```
function Get_TheaterMode: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_TheaterMode();
```

Description

This is Get_TheaterMode, a member of class IWebBrowser2.

132.2.27.2.10 IWebBrowser2.Navigate2 Method

Pascal

```
procedure Navigate2(const URL: OleVariant; const Flags: OleVariant; const TargetFrameName:
OleVariant; const PostData: OleVariant; const Headers: OleVariant); safecall;
```

C++

```
__fastcall Navigate2(const OleVariant URL, const OleVariant Flags, const OleVariant
TargetFrameName, const OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate2, a member of class IWebBrowser2.

132.2.27.2.11 IWebBrowser2.QueryStatusWB Method

Pascal

```
function QueryStatusWB(cmdID: OLECMDID): OLECMDF; safecall;
```

C++

```
__fastcall OLECMDF QueryStatusWB(OLECMDID cmdID);
```

Description

This is QueryStatusWB, a member of class IWebBrowser2.

132.2.27.2.12 IWebBrowser2.Set_AddressBar Method

Pascal

```
procedure Set_AddressBar(Value: WordBool); safecall;
```

C++

```
__fastcall Set_AddressBar(WordBool Value);
```

Description

This is Set_AddressBar, a member of class IWebBrowser2.

132.2.27.2.13 IWebBrowser2.Set_Offline Method

Pascal

```
procedure Set_Offline(pbOffline: WordBool); safecall;
```

C++

```
__fastcall Set_Offline(WordBool pbOffline);
```

Description

This is Set_Offline, a member of class IWebBrowser2.

132.2.27.2.14 IWebBrowser2.Set_RegisterAsBrowser Method

Pascal

```
procedure Set_RegisterAsBrowser(pbRegister: WordBool); safecall;
```

C++

```
__fastcall Set_RegisterAsBrowser(WordBool pbRegister);
```

Description

This is Set_RegisterAsBrowser, a member of class IWebBrowser2.

132.2.27.2.15 IWebBrowser2.Set_RegisterAsDropTarget Method

Pascal

```
procedure Set_RegisterAsDropTarget(pbRegister: WordBool); safecall;
```

C++

```
__fastcall Set_RegisterAsDropTarget(WordBool pbRegister);
```

Description

This is Set_RegisterAsDropTarget, a member of class IWebBrowser2.

132.2.27.2.16 IWebBrowser2.Set_Resizable Method

Pascal

```
procedure Set_Resizable(Value: WordBool); safecall;
```

C++

```
__fastcall Set_Resizable(WordBool Value);
```

Description

This is Set_Resizable, a member of class IWebBrowser2.

132.2.27.2.17 IWebBrowser2.Set_Silent Method

Pascal

```
procedure Set_Silent(pbSilent: WordBool); safecall;
```

C++

```
__fastcall Set_Silent(WordBool pbSilent);
```

Description

This is Set_Silent, a member of class IWebBrowser2.

132.2.27.2.18 IWebBrowser2.Set_TheaterMode Method

Pascal

```
procedure Set_TheaterMode(pbRegister: WordBool); safecall;
```

C++

```
__fastcall Set_TheaterMode(WordBool pbRegister);
```

Description

This is Set_TheaterMode, a member of class IWebBrowser2.

132.2.27.2.19 IWebBrowser2.ShowBrowserBar Method

Pascal

```
procedure ShowBrowserBar(var pvaClsid: OleVariant; var pvarShow: OleVariant; var pvarSize:
OleVariant); safecall;
```

C++

```
__fastcall ShowBrowserBar(OleVariant pvaClsid, OleVariant pvarShow, OleVariant pvarSize);
```

Description

This is ShowBrowserBar, a member of class IWebBrowser2.

132.2.27.3 IWebBrowser2 Properties

The properties of the IWebBrowser2 class are listed here.

Public Properties

	Name	Description
	AddressBar (see page 19410)	This is AddressBar, a member of class IWebBrowser2.
	Offline (see page 19410)	This is Offline, a member of class IWebBrowser2.
R	ReadyState (see page 19410)	This is ReadyState, a member of class IWebBrowser2.
	RegisterAsBrowser (see page 19411)	This is RegisterAsBrowser, a member of class IWebBrowser2.
	RegisterAsDropTarget (see page 19411)	This is RegisterAsDropTarget, a member of class IWebBrowser2.
	Resizable (see page 19411)	This is Resizable, a member of class IWebBrowser2.
	Silent (see page 19411)	This is Silent, a member of class IWebBrowser2.
	TheaterMode (see page 19412)	This is TheaterMode, a member of class IWebBrowser2.

132.2.27.3.1 IWebBrowser2.AddressBar Property

Pascal

```
property AddressBar: WordBool;
```

C++

```
__property WordBool AddressBar;
```

Description

This is AddressBar, a member of class IWebBrowser2.

132.2.27.3.2 IWebBrowser2.Offline Property

Pascal

```
property Offline: WordBool;
```

C++

```
__property WordBool Offline;
```

Description

This is Offline, a member of class IWebBrowser2.

132.2.27.3.3 IWebBrowser2.ReadyState Property

Pascal

```
property ReadyState: tagREADYSTATE;
```

C++

```
__property tagREADYSTATE ReadyState;
```

Description

This is ReadyState, a member of class IWebBrowser2.

132.2.27.3.4 IWebBrowser2.RegisterAsBrowser Property

Pascal

```
property RegisterAsBrowser: WordBool;
```

C++

```
__property WordBool RegisterAsBrowser;
```

Description

This is RegisterAsBrowser, a member of class IWebBrowser2.

132.2.27.3.5 IWebBrowser2.RegisterAsDropTarget Property

Pascal

```
property RegisterAsDropTarget: WordBool;
```

C++

```
__property WordBool RegisterAsDropTarget;
```

Description

This is RegisterAsDropTarget, a member of class IWebBrowser2.

132.2.27.3.6 IWebBrowser2.Resizable Property

Pascal

```
property Resizable: WordBool;
```

C++

```
__property WordBool Resizable;
```

Description

This is Resizable, a member of class IWebBrowser2.

132.2.27.3.7 IWebBrowser2.Silent Property

Pascal

```
property Silent: WordBool;
```

C++

```
__property WordBool Silent;
```

Description

This is Silent, a member of class IWebBrowser2.

132.2.27.3.8 IWebBrowser2.TheaterMode Property

Pascal

```
property TheaterMode: WordBool;
```

C++

```
__property WordBool TheaterMode;
```

Description

This is TheaterMode, a member of class IWebBrowser2.

132.2.28 IWebBrowser2Disp Interface

Class Hierarchy

SHDocVw.IWebBrowser2Disp

Pascal

```
IWebBrowser2Disp = dispinterface;
```

C++

```
class IWebBrowser2Disp;
```

File

SHDocVw

Description

This is class SHDocVw.IWebBrowser2Disp.

132.2.28.1 IWebBrowser2Disp Members

The following tables list the members exposed by IWebBrowser2Disp.

Public Methods

	Name	Description
	ClientToWindow (see page 19414)	This is ClientToWindow, a member of class IWebBrowser2Disp.
	ExecWB (see page 19414)	This is ExecWB, a member of class IWebBrowser2Disp.
	GetProperty (see page 19414)	This is GetProperty, a member of class IWebBrowser2Disp.
	GoBack (see page 19414)	This is GoBack, a member of class IWebBrowser2Disp.
	GoForward (see page 19415)	This is GoForward, a member of class IWebBrowser2Disp.
	GoHome (see page 19415)	This is GoHome, a member of class IWebBrowser2Disp.
	GoSearch (see page 19415)	This is GoSearch, a member of class IWebBrowser2Disp.
	Navigate (see page 19415)	This is Navigate, a member of class IWebBrowser2Disp.
	Navigate2 (see page 19415)	This is Navigate2, a member of class IWebBrowser2Disp.
	PutProperty (see page 19416)	This is PutProperty, a member of class IWebBrowser2Disp.
	QueryStatusWB (see page 19416)	This is QueryStatusWB, a member of class IWebBrowser2Disp.
	Quit (see page 19416)	This is Quit, a member of class IWebBrowser2Disp.
	Refresh (see page 19416)	This is Refresh, a member of class IWebBrowser2Disp.
	Refresh2 (see page 19417)	This is Refresh2, a member of class IWebBrowser2Disp.
	ShowBrowserBar (see page 19417)	This is ShowBrowserBar, a member of class IWebBrowser2Disp.

	Stop (see page 19417)	This is Stop, a member of class IWebBrowser2Disp.

Public Properties

	Name	Description
	AddressBar (see page 19418)	This is AddressBar, a member of class IWebBrowser2Disp.
	Application (see page 19418)	This is Application, a member of class IWebBrowser2Disp.
	Busy (see page 19418)	This is Busy, a member of class IWebBrowser2Disp.
	Container (see page 19419)	This is Container, a member of class IWebBrowser2Disp.
	Document (see page 19419)	This is Document, a member of class IWebBrowser2Disp.
	FullName (see page 19419)	This is FullName, a member of class IWebBrowser2Disp.
	FullScreen (see page 19419)	This is FullScreen, a member of class IWebBrowser2Disp.
	Height (see page 19419)	This is Height, a member of class IWebBrowser2Disp.
	HWnd (see page 19420)	This is HWnd, a member of class IWebBrowser2Disp.
	Left (see page 19420)	This is Left, a member of class IWebBrowser2Disp.
	LocationName (see page 19420)	This is LocationName, a member of class IWebBrowser2Disp.
	LocationURL (see page 19420)	This is LocationURL, a member of class IWebBrowser2Disp.
	MenuBar (see page 19421)	This is MenuBar, a member of class IWebBrowser2Disp.
	Name (see page 19421)	This is Name, a member of class IWebBrowser2Disp.
	Offline (see page 19421)	This is Offline, a member of class IWebBrowser2Disp.
	Parent (see page 19421)	This is Parent, a member of class IWebBrowser2Disp.
	Path (see page 19421)	This is Path, a member of class IWebBrowser2Disp.
	ReadyState (see page 19422)	This is ReadyState, a member of class IWebBrowser2Disp.
	RegisterAsBrowser (see page 19422)	This is RegisterAsBrowser, a member of class IWebBrowser2Disp.
	RegisterAsDropTarget (see page 19422)	This is RegisterAsDropTarget, a member of class IWebBrowser2Disp.
	Resizable (see page 19422)	This is Resizable, a member of class IWebBrowser2Disp.
	Silent (see page 19423)	This is Silent, a member of class IWebBrowser2Disp.
	StatusBar (see page 19423)	This is StatusBar, a member of class IWebBrowser2Disp.
	StatusText (see page 19423)	This is StatusText, a member of class IWebBrowser2Disp.
	TheaterMode (see page 19423)	This is TheaterMode, a member of class IWebBrowser2Disp.
	ToolBar (see page 19423)	This is ToolBar, a member of class IWebBrowser2Disp.
	Top (see page 19424)	This is Top, a member of class IWebBrowser2Disp.
	TopLevelContainer (see page 19424)	This is TopLevelContainer, a member of class IWebBrowser2Disp.
	type_ (see page 19424)	This is type_, a member of class IWebBrowser2Disp.
	Visible (see page 19424)	This is Visible, a member of class IWebBrowser2Disp.
	Width (see page 19425)	This is Width, a member of class IWebBrowser2Disp.

132.2.28.2 IWebBrowser2Disp Methods

The methods of the IWebBrowser2Disp class are listed here.

Public Methods

	Name	Description
	ClientToWindow (see page 19414)	This is ClientToWindow, a member of class IWebBrowser2Disp.
	ExecWB (see page 19414)	This is ExecWB, a member of class IWebBrowser2Disp.
	GetProperty (see page 19414)	This is GetProperty, a member of class IWebBrowser2Disp.
	GoBack (see page 19414)	This is GoBack, a member of class IWebBrowser2Disp.
	GoForward (see page 19415)	This is GoForward, a member of class IWebBrowser2Disp.
	GoHome (see page 19415)	This is GoHome, a member of class IWebBrowser2Disp.
	GoSearch (see page 19415)	This is GoSearch, a member of class IWebBrowser2Disp.
	Navigate (see page 19415)	This is Navigate, a member of class IWebBrowser2Disp.
	Navigate2 (see page 19415)	This is Navigate2, a member of class IWebBrowser2Disp.
	PutProperty (see page 19416)	This is PutProperty, a member of class IWebBrowser2Disp.
	QueryStatusWB (see page 19416)	This is QueryStatusWB, a member of class IWebBrowser2Disp.

	Quit (see page 19416)	This is Quit, a member of class IWebBrowser2Disp.
	Refresh (see page 19416)	This is Refresh, a member of class IWebBrowser2Disp.
	Refresh2 (see page 19417)	This is Refresh2, a member of class IWebBrowser2Disp.
	ShowBrowserBar (see page 19417)	This is ShowBrowserBar, a member of class IWebBrowser2Disp.
	Stop (see page 19417)	This is Stop, a member of class IWebBrowser2Disp.

132.2.28.2.1 IWebBrowser2Disp.ClientToWindow Method

Pascal

```
procedure ClientToWindow(var pcx: SYSINT; var pcy: SYSINT);
```

C++

```
__fastcall ClientToWindow(SYSINT pcx, SYSINT pcy);
```

Description

This is ClientToWindow, a member of class IWebBrowser2Disp.

132.2.28.2.2 IWebBrowser2Disp.ExecWB Method

Pascal

```
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT; var pvaIn: OleVariant; var
pvaOut: OleVariant);
```

C++

```
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, OleVariant pvaIn, OleVariant
pvaOut);
```

Description

This is ExecWB, a member of class IWebBrowser2Disp.

132.2.28.2.3 IWebBrowser2Disp.GetProperty Method

Pascal

```
function GetProperty(const Property_: WideString): OleVariant;
```

C++

```
__fastcall OleVariant GetProperty(const BSTR Property_);
```

Description

This is GetProperty, a member of class IWebBrowser2Disp.

132.2.28.2.4 IWebBrowser2Disp.GoBack Method

Pascal

```
procedure GoBack;
```

C++

```
__fastcall GoBack();
```

Description

This is GoBack, a member of class IWebBrowser2Disp.

132.2.28.2.5 IWebBrowser2Disp.GoForward Method

Pascal

```
procedure GoForward;
```

C++

```
__fastcall GoForward();
```

Description

This is GoForward, a member of class IWebBrowser2Disp.

132.2.28.2.6 IWebBrowser2Disp.GoHome Method

Pascal

```
procedure GoHome;
```

C++

```
__fastcall GoHome();
```

Description

This is GoHome, a member of class IWebBrowser2Disp.

132.2.28.2.7 IWebBrowser2Disp.GoSearch Method

Pascal

```
procedure GoSearch;
```

C++

```
__fastcall GoSearch();
```

Description

This is GoSearch, a member of class IWebBrowser2Disp.

132.2.28.2.8 IWebBrowser2Disp.Navigate Method

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant);
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate, a member of class IWebBrowser2Disp.

132.2.28.2.9 IWebBrowser2Disp.Navigate2 Method

Pascal

```
procedure Navigate2(const URL: OleVariant; const Flags: OleVariant; const TargetFrameName:
OleVariant; const PostData: OleVariant; const Headers: OleVariant);
```

C++

```
__fastcall Navigate2(const OleVariant URL, const OleVariant Flags, const OleVariant
TargetFrameName, const OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate2, a member of class IWebBrowser2Disp.

132.2.28.2.10 IWebBrowser2Disp.PutProperty Method

Pascal

```
procedure PutProperty(const Property_: WideString; vtValue: OleVariant);
```

C++

```
__fastcall PutProperty(const BSTR Property_, OleVariant vtValue);
```

Description

This is PutProperty, a member of class IWebBrowser2Disp.

132.2.28.2.11 IWebBrowser2Disp.QueryStatusWB Method

Pascal

```
function QueryStatusWB(cmdID: OLECMDID): OLECMDF;
```

C++

```
__fastcall OLECMDF QueryStatusWB(OLECMDID cmdID);
```

Description

This is QueryStatusWB, a member of class IWebBrowser2Disp.

132.2.28.2.12 IWebBrowser2Disp.Quit Method

Pascal

```
procedure Quit;
```

C++

```
__fastcall Quit();
```

Description

This is Quit, a member of class IWebBrowser2Disp.

132.2.28.2.13 IWebBrowser2Disp.Refresh Method

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

This is Refresh, a member of class IWebBrowser2Disp.

132.2.28.2.14 IWebBrowser2Disp.Refresh2 Method

Pascal

```
procedure Refresh2(var Level: OleVariant);
```

C++

```
__fastcall Refresh2(OleVariant Level);
```

Description

This is Refresh2, a member of class IWebBrowser2Disp.

132.2.28.2.15 IWebBrowser2Disp.ShowBrowserBar Method

Pascal

```
procedure ShowBrowserBar(var pvaClsid: OleVariant; var pvarShow: OleVariant; var pvarSize:
OleVariant);
```

C++

```
__fastcall ShowBrowserBar(OleVariant pvaClsid, OleVariant pvarShow, OleVariant pvarSize);
```

Description

This is ShowBrowserBar, a member of class IWebBrowser2Disp.

132.2.28.2.16 IWebBrowser2Disp.Stop Method

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

This is Stop, a member of class IWebBrowser2Disp.

132.2.28.3 IWebBrowser2Disp Properties

The properties of the IWebBrowser2Disp class are listed here.

Public Properties

	Name	Description
	AddressBar (see page 19418)	This is AddressBar, a member of class IWebBrowser2Disp.
	Application (see page 19418)	This is Application, a member of class IWebBrowser2Disp.
	Busy (see page 19418)	This is Busy, a member of class IWebBrowser2Disp.
	Container (see page 19419)	This is Container, a member of class IWebBrowser2Disp.
	Document (see page 19419)	This is Document, a member of class IWebBrowser2Disp.
	FullName (see page 19419)	This is FullName, a member of class IWebBrowser2Disp.
	FullScreen (see page 19419)	This is FullScreen, a member of class IWebBrowser2Disp.
	Height (see page 19419)	This is Height, a member of class IWebBrowser2Disp.
	HWnd (see page 19420)	This is HWnd, a member of class IWebBrowser2Disp.
	Left (see page 19420)	This is Left, a member of class IWebBrowser2Disp.
	LocationName (see page 19420)	This is LocationName, a member of class IWebBrowser2Disp.

	LocationURL (see page 19420)	This is LocationURL, a member of class IWebBrowser2Disp.
	MenuBar (see page 19421)	This is MenuBar, a member of class IWebBrowser2Disp.
	Name (see page 19421)	This is Name, a member of class IWebBrowser2Disp.
	Offline (see page 19421)	This is Offline, a member of class IWebBrowser2Disp.
	Parent (see page 19421)	This is Parent, a member of class IWebBrowser2Disp.
	Path (see page 19421)	This is Path, a member of class IWebBrowser2Disp.
	ReadyState (see page 19422)	This is ReadyState, a member of class IWebBrowser2Disp.
	RegisterAsBrowser (see page 19422)	This is RegisterAsBrowser, a member of class IWebBrowser2Disp.
	RegisterAsDropTarget (see page 19422)	This is RegisterAsDropTarget, a member of class IWebBrowser2Disp.
	Resizable (see page 19422)	This is Resizable, a member of class IWebBrowser2Disp.
	Silent (see page 19423)	This is Silent, a member of class IWebBrowser2Disp.
	StatusBar (see page 19423)	This is StatusBar, a member of class IWebBrowser2Disp.
	StatusText (see page 19423)	This is StatusText, a member of class IWebBrowser2Disp.
	TheaterMode (see page 19423)	This is TheaterMode, a member of class IWebBrowser2Disp.
	ToolBar (see page 19423)	This is ToolBar, a member of class IWebBrowser2Disp.
	Top (see page 19424)	This is Top, a member of class IWebBrowser2Disp.
	TopLevelContainer (see page 19424)	This is TopLevelContainer, a member of class IWebBrowser2Disp.
	type_ (see page 19424)	This is type_, a member of class IWebBrowser2Disp.
	Visible (see page 19424)	This is Visible, a member of class IWebBrowser2Disp.
	Width (see page 19425)	This is Width, a member of class IWebBrowser2Disp.

132.2.28.3.1 IWebBrowser2Disp.AddressBar Property

Pascal

```
property AddressBar: WordBool;
```

C++

```
__property WordBool AddressBar;
```

Description

This is AddressBar, a member of class IWebBrowser2Disp.

132.2.28.3.2 IWebBrowser2Disp.Application Property

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

This is Application, a member of class IWebBrowser2Disp.

132.2.28.3.3 IWebBrowser2Disp.Busy Property

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

This is Busy, a member of class IWebBrowser2Disp.

132.2.28.3.4 IWebBrowser2Disp.Container Property

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

This is Container, a member of class IWebBrowser2Disp.

132.2.28.3.5 IWebBrowser2Disp.Document Property

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

This is Document, a member of class IWebBrowser2Disp.

132.2.28.3.6 IWebBrowser2Disp.FullName Property

Pascal

```
property FullName: WideString;
```

C++

```
__property BSTR FullName;
```

Description

This is FullName, a member of class IWebBrowser2Disp.

132.2.28.3.7 IWebBrowser2Disp.FullScreen Property

Pascal

```
property FullScreen: WordBool;
```

C++

```
__property WordBool FullScreen;
```

Description

This is FullScreen, a member of class IWebBrowser2Disp.

132.2.28.3.8 IWebBrowser2Disp.Height Property

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

This is Height, a member of class IWebBrowser2Disp.

132.2.28.3.9 IWebBrowser2Disp.HWnd Property

Pascal

```
property HWnd: Integer;
```

C++

```
__property int HWnd;
```

Description

This is HWnd, a member of class IWebBrowser2Disp.

132.2.28.3.10 IWebBrowser2Disp.Left Property

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

This is Left, a member of class IWebBrowser2Disp.

132.2.28.3.11 IWebBrowser2Disp.LocationName Property

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

This is LocationName, a member of class IWebBrowser2Disp.

132.2.28.3.12 IWebBrowser2Disp.LocationURL Property

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

This is LocationURL, a member of class IWebBrowser2Disp.

132.2.28.3.13 IWebBrowser2Disp.MenuBar Property

Pascal

```
property MenuBar: WordBool;
```

C++

```
__property WordBool MenuBar;
```

Description

This is MenuBar, a member of class IWebBrowser2Disp.

132.2.28.3.14 IWebBrowser2Disp.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IWebBrowser2Disp.

132.2.28.3.15 IWebBrowser2Disp.Offline Property

Pascal

```
property Offline: WordBool;
```

C++

```
__property WordBool Offline;
```

Description

This is Offline, a member of class IWebBrowser2Disp.

132.2.28.3.16 IWebBrowser2Disp.Parent Property

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

This is Parent, a member of class IWebBrowser2Disp.

132.2.28.3.17 IWebBrowser2Disp.Path Property

Pascal

```
property Path: WideString;
```

C++

```
__property BSTR Path;
```

Description

This is Path, a member of class IWebBrowser2Disp.

132.2.28.3.18 IWebBrowser2Disp.ReadyState Property

Pascal

```
property ReadyState: tagREADYSTATE;
```

C++

```
__property tagREADYSTATE ReadyState;
```

Description

This is ReadyState, a member of class IWebBrowser2Disp.

132.2.28.3.19 IWebBrowser2Disp.RegisterAsBrowser Property

Pascal

```
property RegisterAsBrowser: WordBool;
```

C++

```
__property WordBool RegisterAsBrowser;
```

Description

This is RegisterAsBrowser, a member of class IWebBrowser2Disp.

132.2.28.3.20 IWebBrowser2Disp.RegisterAsDropTarget Property

Pascal

```
property RegisterAsDropTarget: WordBool;
```

C++

```
__property WordBool RegisterAsDropTarget;
```

Description

This is RegisterAsDropTarget, a member of class IWebBrowser2Disp.

132.2.28.3.21 IWebBrowser2Disp.Resizable Property

Pascal

```
property Resizable: WordBool;
```

C++

```
__property WordBool Resizable;
```

Description

This is Resizable, a member of class IWebBrowser2Disp.

132.2.28.3.22 IWebBrowser2Disp.Silent Property

Pascal

```
property Silent: WordBool;
```

C++

```
__property WordBool Silent;
```

Description

This is Silent, a member of class IWebBrowser2Disp.

132.2.28.3.23 IWebBrowser2Disp.StatusBar Property

Pascal

```
property StatusBar: WordBool;
```

C++

```
__property WordBool StatusBar;
```

Description

This is StatusBar, a member of class IWebBrowser2Disp.

132.2.28.3.24 IWebBrowser2Disp.StatusText Property

Pascal

```
property StatusText: WideString;
```

C++

```
__property BSTR StatusText;
```

Description

This is StatusText, a member of class IWebBrowser2Disp.

132.2.28.3.25 IWebBrowser2Disp.TheaterMode Property

Pascal

```
property TheaterMode: WordBool;
```

C++

```
__property WordBool TheaterMode;
```

Description

This is TheaterMode, a member of class IWebBrowser2Disp.

132.2.28.3.26 IWebBrowser2Disp.ToolBar Property

Pascal

```
property ToolBar: SYSINT;
```

C++

```
__property SYSINT ToolBar;
```

Description

This is ToolBar, a member of class IWebBrowser2Disp.

132.2.28.3.27 IWebBrowser2Disp.Top Property

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

This is Top, a member of class IWebBrowser2Disp.

132.2.28.3.28 IWebBrowser2Disp.TopLevelContainer Property

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

This is TopLevelContainer, a member of class IWebBrowser2Disp.

132.2.28.3.29 IWebBrowser2Disp.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class IWebBrowser2Disp.

132.2.28.3.30 IWebBrowser2Disp.Visible Property

Pascal

```
property Visible: WordBool;
```

C++

```
__property WordBool Visible;
```

Description

This is Visible, a member of class IWebBrowser2Disp.

132.2.28.3.31 IWebBrowser2Disp.Width Property

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

This is Width, a member of class IWebBrowser2Disp.

132.2.29 IWebBrowserApp Interface

Class Hierarchy

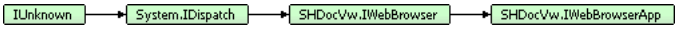

Pascal

```
IWebBrowserApp = interface(IWebBrowser);
```

C++

```
__interface IWebBrowserApp : public IWebBrowser;
```

File

SHDocVw

Description

This is class SHDocVw.IWebBrowserApp.

132.2.29.1 IWebBrowserApp Members

The following tables list the members exposed by IWebBrowserApp.

Public Methods

	Name	Description
	ClientToWindow (see page 19427)	This is ClientToWindow, a member of class IWebBrowserApp.
	Get_Application (see page 19394)	This is Get_Application, a member of class IWebBrowser.
	Get_Busy (see page 19394)	This is Get_Busy, a member of class IWebBrowser.
	Get_Container (see page 19394)	This is Get_Container, a member of class IWebBrowser.
	Get_Document (see page 19394)	This is Get_Document, a member of class IWebBrowser.
	Get_FullName (see page 19427)	This is Get_FullName, a member of class IWebBrowserApp.
	Get_FullScreen (see page 19428)	This is Get_FullScreen, a member of class IWebBrowserApp.
	Get_Height (see page 19395)	This is Get_Height, a member of class IWebBrowser.
	Get_HWnd (see page 19428)	This is Get_HWnd, a member of class IWebBrowserApp.
	Get_Left (see page 19395)	This is Get_Left, a member of class IWebBrowser.
	Get_LocationName (see page 19395)	This is Get_LocationName, a member of class IWebBrowser.
	Get_LocationURL (see page 19395)	This is Get_LocationURL, a member of class IWebBrowser.
	Get_MenuBar (see page 19428)	This is Get_MenuBar, a member of class IWebBrowserApp.
	Get_Name (see page 19428)	This is Get_Name, a member of class IWebBrowserApp.
	Get_Parent (see page 19396)	This is Get_Parent, a member of class IWebBrowser.

	Get_Path (see page 19429)	This is Get_Path, a member of class IWebBrowserApp.
	Get_StatusBar (see page 19429)	This is Get_StatusBar, a member of class IWebBrowserApp.
	Get_StatusText (see page 19429)	This is Get_StatusText, a member of class IWebBrowserApp.
	Get_ToolBar (see page 19429)	This is Get_ToolBar, a member of class IWebBrowserApp.
	Get_Top (see page 19396)	This is Get_Top, a member of class IWebBrowser.
	Get_TopLevelContainer (see page 19396)	This is Get_TopLevelContainer, a member of class IWebBrowser.
	Get_type_ (see page 19396)	This is Get_type_, a member of class IWebBrowser.
	Get_Visible (see page 19429)	This is Get_Visible, a member of class IWebBrowserApp.
	Get_Width (see page 19396)	This is Get_Width, a member of class IWebBrowser.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetProperty (see page 19430)	This is GetProperty, a member of class IWebBrowserApp.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	GoBack (see page 19397)	This is GoBack, a member of class IWebBrowser.
	GoForward (see page 19397)	This is GoForward, a member of class IWebBrowser.
	GoHome (see page 19397)	This is GoHome, a member of class IWebBrowser.
	GoSearch (see page 19397)	This is GoSearch, a member of class IWebBrowser.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	Navigate (see page 19398)	This is Navigate, a member of class IWebBrowser.
	PutProperty (see page 19430)	This is PutProperty, a member of class IWebBrowserApp.
	Quit (see page 19430)	This is Quit, a member of class IWebBrowserApp.
	Refresh (see page 19398)	This is Refresh, a member of class IWebBrowser.
	Refresh2 (see page 19398)	This is Refresh2, a member of class IWebBrowser.
	Set_FullScreen (see page 19430)	This is Set_FullScreen, a member of class IWebBrowserApp.
	Set_Height (see page 19398)	This is Set_Height, a member of class IWebBrowser.
	Set_Left (see page 19398)	This is Set_Left, a member of class IWebBrowser.
	Set_MenuBar (see page 19431)	This is Set_MenuBar, a member of class IWebBrowserApp.
	Set_StatusBar (see page 19431)	This is Set_StatusBar, a member of class IWebBrowserApp.
	Set_StatusText (see page 19431)	This is Set_StatusText, a member of class IWebBrowserApp.
	Set_ToolBar (see page 19431)	This is Set_ToolBar, a member of class IWebBrowserApp.
	Set_Top (see page 19399)	This is Set_Top, a member of class IWebBrowser.
	Set_Visible (see page 19431)	This is Set_Visible, a member of class IWebBrowserApp.
	Set_Width (see page 19399)	This is Set_Width, a member of class IWebBrowser.
	Stop (see page 19399)	This is Stop, a member of class IWebBrowser.

Public Properties

	Name	Description
R	Application (see page 19400)	This is Application, a member of class IWebBrowser.
R	Busy (see page 19400)	This is Busy, a member of class IWebBrowser.
R	Container (see page 19400)	This is Container, a member of class IWebBrowser.
R	Document (see page 19400)	This is Document, a member of class IWebBrowser.
R	FullName (see page 19432)	This is FullName, a member of class IWebBrowserApp.
	FullScreen (see page 19432)	This is FullScreen, a member of class IWebBrowserApp.
	Height (see page 19401)	This is Height, a member of class IWebBrowser.
R	HWnd (see page 19432)	This is HWnd, a member of class IWebBrowserApp.
	Left (see page 19401)	This is Left, a member of class IWebBrowser.
R	LocationName (see page 19401)	This is LocationName, a member of class IWebBrowser.
R	LocationURL (see page 19401)	This is LocationURL, a member of class IWebBrowser.
	MenuBar (see page 19433)	This is MenuBar, a member of class IWebBrowserApp.
R	Name (see page 19433)	This is Name, a member of class IWebBrowserApp.
R	Parent (see page 19401)	This is Parent, a member of class IWebBrowser.
R	Path (see page 19433)	This is Path, a member of class IWebBrowserApp.
	StatusBar (see page 19433)	This is StatusBar, a member of class IWebBrowserApp.
	StatusText (see page 19434)	This is StatusText, a member of class IWebBrowserApp.

	ToolBar (see page 19434)	This is ToolBar, a member of class IWebBrowserApp.
	Top (see page 19402)	This is Top, a member of class IWebBrowser.
R	TopLevelContainer (see page 19402)	This is TopLevelContainer, a member of class IWebBrowser.
R	type_ (see page 19402)	This is type_, a member of class IWebBrowser.
	Visible (see page 19434)	This is Visible, a member of class IWebBrowserApp.
	Width (see page 19402)	This is Width, a member of class IWebBrowser.

132.2.29.2 IWebBrowserApp Methods

The methods of the IWebBrowserApp class are listed here.

Public Methods

	Name	Description
	ClientToWindow (see page 19427)	This is ClientToWindow, a member of class IWebBrowserApp.
	Get_FullName (see page 19427)	This is Get_FullName, a member of class IWebBrowserApp.
	Get_FullScreen (see page 19428)	This is Get_FullScreen, a member of class IWebBrowserApp.
	Get_HWnd (see page 19428)	This is Get_HWnd, a member of class IWebBrowserApp.
	Get_MenuBar (see page 19428)	This is Get_MenuBar, a member of class IWebBrowserApp.
	Get_Name (see page 19428)	This is Get_Name, a member of class IWebBrowserApp.
	Get_Path (see page 19429)	This is Get_Path, a member of class IWebBrowserApp.
	Get_StatusBar (see page 19429)	This is Get_StatusBar, a member of class IWebBrowserApp.
	Get_StatusText (see page 19429)	This is Get_StatusText, a member of class IWebBrowserApp.
	Get_ToolBar (see page 19429)	This is Get_ToolBar, a member of class IWebBrowserApp.
	Get_Visible (see page 19429)	This is Get_Visible, a member of class IWebBrowserApp.
	GetProperty (see page 19430)	This is GetProperty, a member of class IWebBrowserApp.
	PutProperty (see page 19430)	This is PutProperty, a member of class IWebBrowserApp.
	Quit (see page 19430)	This is Quit, a member of class IWebBrowserApp.
	Set_FullScreen (see page 19430)	This is Set_FullScreen, a member of class IWebBrowserApp.
	Set_MenuBar (see page 19431)	This is Set_MenuBar, a member of class IWebBrowserApp.
	Set_StatusBar (see page 19431)	This is Set_StatusBar, a member of class IWebBrowserApp.
	Set_StatusText (see page 19431)	This is Set_StatusText, a member of class IWebBrowserApp.
	Set_ToolBar (see page 19431)	This is Set_ToolBar, a member of class IWebBrowserApp.
	Set_Visible (see page 19431)	This is Set_Visible, a member of class IWebBrowserApp.

132.2.29.2.1 IWebBrowserApp.ClientToWindow Method

Pascal

```
procedure ClientToWindow(var pcx: SYSINT; var pcy: SYSINT); safecall;
```

C++

```
__fastcall ClientToWindow(SYSINT pcx, SYSINT pcy);
```

Description

This is ClientToWindow, a member of class IWebBrowserApp.

132.2.29.2.2 IWebBrowserApp.Get_FullName Method

Pascal

```
function Get_FullName: WideString; safecall;
```

C++

```
__fastcall BSTR Get_FullName();
```

Description

This is Get_FullName, a member of class IWebBrowserApp.

132.2.29.2.3 IWebBrowserApp.Get_FullScreen Method

Pascal

```
function Get_FullScreen: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_FullScreen();
```

Description

This is Get_FullScreen, a member of class IWebBrowserApp.

132.2.29.2.4 IWebBrowserApp.Get_HWnd Method

Pascal

```
function Get_HWnd: Integer; safecall;
```

C++

```
__fastcall int Get_HWnd();
```

Description

This is Get_HWnd, a member of class IWebBrowserApp.

132.2.29.2.5 IWebBrowserApp.Get_MenuBar Method

Pascal

```
function Get_MenuBar: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_MenuBar();
```

Description

This is Get_MenuBar, a member of class IWebBrowserApp.

132.2.29.2.6 IWebBrowserApp.Get_Name Method

Pascal

```
function Get_Name: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Name();
```

Description

This is Get_Name, a member of class IWebBrowserApp.

132.2.29.2.7 IWebBrowserApp.Get_Path Method

Pascal

```
function Get_Path: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Path();
```

Description

This is Get_Path, a member of class IWebBrowserApp.

132.2.29.2.8 IWebBrowserApp.Get_StatusBar Method

Pascal

```
function Get_StatusBar: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_StatusBar();
```

Description

This is Get_StatusBar, a member of class IWebBrowserApp.

132.2.29.2.9 IWebBrowserApp.Get_StatusText Method

Pascal

```
function Get_StatusText: WideString; safecall;
```

C++

```
__fastcall BSTR Get_StatusText();
```

Description

This is Get_StatusText, a member of class IWebBrowserApp.

132.2.29.2.10 IWebBrowserApp.Get_ToolBar Method

Pascal

```
function Get_ToolBar: SYSINT; safecall;
```

C++

```
__fastcall SYSINT Get_ToolBar();
```

Description

This is Get_ToolBar, a member of class IWebBrowserApp.

132.2.29.2.11 IWebBrowserApp.Get_Visible Method

Pascal

```
function Get_Visible: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_Visible();
```

Description

This is Get_Visible, a member of class IWebBrowserApp.

132.2.29.2.12 IWebBrowserApp.GetProperty Method

Pascal

```
function GetProperty(const Property_: WideString): OleVariant; safecall;
```

C++

```
__fastcall OleVariant GetProperty(const BSTR Property_);
```

Description

This is GetProperty, a member of class IWebBrowserApp.

132.2.29.2.13 IWebBrowserApp.PutProperty Method

Pascal

```
procedure PutProperty(const Property_: WideString; vtValue: OleVariant); safecall;
```

C++

```
__fastcall PutProperty(const BSTR Property_, OleVariant vtValue);
```

Description

This is PutProperty, a member of class IWebBrowserApp.

132.2.29.2.14 IWebBrowserApp.Quit Method

Pascal

```
procedure Quit; safecall;
```

C++

```
__fastcall Quit();
```

Description

This is Quit, a member of class IWebBrowserApp.

132.2.29.2.15 IWebBrowserApp.Set_FullScreen Method

Pascal

```
procedure Set_FullScreen(pbFullScreen: WordBool); safecall;
```

C++

```
__fastcall Set_FullScreen(WordBool pbFullScreen);
```

Description

This is Set_FullScreen, a member of class IWebBrowserApp.

132.2.29.2.16 IWebBrowserApp.Set_MenuBar Method

Pascal

```
procedure Set_MenuBar(Value: WordBool); safecall;
```

C++

```
__fastcall Set_MenuBar(WordBool Value);
```

Description

This is Set_MenuBar, a member of class IWebBrowserApp.

132.2.29.2.17 IWebBrowserApp.Set_StatusBar Method

Pascal

```
procedure Set_StatusBar(pBool: WordBool); safecall;
```

C++

```
__fastcall Set_StatusBar(WordBool pBool);
```

Description

This is Set_StatusBar, a member of class IWebBrowserApp.

132.2.29.2.18 IWebBrowserApp.Set_StatusText Method

Pascal

```
procedure Set_StatusText(const StatusText: WideString); safecall;
```

C++

```
__fastcall Set_StatusText(const BSTR StatusText);
```

Description

This is Set_StatusText, a member of class IWebBrowserApp.

132.2.29.2.19 IWebBrowserApp.Set_ToolBar Method

Pascal

```
procedure Set_ToolBar(Value: SYSINT); safecall;
```

C++

```
__fastcall Set_ToolBar(SYSINT Value);
```

Description

This is Set_ToolBar, a member of class IWebBrowserApp.

132.2.29.2.20 IWebBrowserApp.Set_Visible Method

Pascal

```
procedure Set_Visible(pBool: WordBool); safecall;
```

C++

```
__fastcall Set_Visible(WordBool pBool);
```

Description

This is Set_Visible, a member of class IWebBrowserApp.

132.2.29.3 IWebBrowserApp Properties

The properties of the IWebBrowserApp class are listed here.

Public Properties

	Name	Description
R	FullName (see page 19432)	This is FullName, a member of class IWebBrowserApp.
	FullScreen (see page 19432)	This is FullScreen, a member of class IWebBrowserApp.
R	HWnd (see page 19432)	This is HWnd, a member of class IWebBrowserApp.
	MenuBar (see page 19433)	This is MenuBar, a member of class IWebBrowserApp.
R	Name (see page 19433)	This is Name, a member of class IWebBrowserApp.
R	Path (see page 19433)	This is Path, a member of class IWebBrowserApp.
	StatusBar (see page 19433)	This is StatusBar, a member of class IWebBrowserApp.
	StatusText (see page 19434)	This is StatusText, a member of class IWebBrowserApp.
	ToolBar (see page 19434)	This is ToolBar, a member of class IWebBrowserApp.
	Visible (see page 19434)	This is Visible, a member of class IWebBrowserApp.

132.2.29.3.1 IWebBrowserApp.FullName Property

Pascal

```
property FullName: WideString;
```

C++

```
__property BSTR FullName;
```

Description

This is FullName, a member of class IWebBrowserApp.

132.2.29.3.2 IWebBrowserApp.FullScreen Property

Pascal

```
property FullScreen: WordBool;
```

C++

```
__property WordBool FullScreen;
```

Description

This is FullScreen, a member of class IWebBrowserApp.

132.2.29.3.3 IWebBrowserApp.HWnd Property

Pascal

```
property HWnd: Integer;
```

C++

```
__property int HWnd;
```

Description

This is HWnd, a member of class IWebBrowserApp.

132.2.29.3.4 IWebBrowserApp.MenuBar Property

Pascal

```
property MenuBar: WordBool;
```

C++

```
__property WordBool MenuBar;
```

Description

This is MenuBar, a member of class IWebBrowserApp.

132.2.29.3.5 IWebBrowserApp.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IWebBrowserApp.

132.2.29.3.6 IWebBrowserApp.Path Property

Pascal

```
property Path: WideString;
```

C++

```
__property BSTR Path;
```

Description

This is Path, a member of class IWebBrowserApp.

132.2.29.3.7 IWebBrowserApp.StatusBar Property

Pascal

```
property StatusBar: WordBool;
```

C++

```
__property WordBool StatusBar;
```

Description

This is StatusBar, a member of class IWebBrowserApp.

132.2.29.3.8 IWebBrowserApp.StatusText Property

Pascal

```
property StatusText: WideString;
```

C++

```
__property BSTR StatusText;
```

Description

This is StatusText, a member of class IWebBrowserApp.

132.2.29.3.9 IWebBrowserApp.ToolBar Property

Pascal

```
property ToolBar: SYSINT;
```

C++

```
__property SYSINT ToolBar;
```

Description

This is ToolBar, a member of class IWebBrowserApp.

132.2.29.3.10 IWebBrowserApp.Visible Property

Pascal

```
property Visible: WordBool;
```

C++

```
__property WordBool Visible;
```

Description

This is Visible, a member of class IWebBrowserApp.

132.2.30 IWebBrowserAppDisp Interface

Class Hierarchy

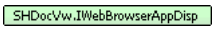

Pascal

```
IWebBrowserAppDisp = dispinterface;
```

C++

```
class IWebBrowserAppDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IWebBrowserAppDisp.

132.2.30.1 IWebBrowserAppDisp Members

The following tables list the members exposed by IWebBrowserAppDisp.

Public Methods

	Name	Description
	ClientToWindow (see page 19436)	This is ClientToWindow, a member of class IWebBrowserAppDisp.
	GetProperty (see page 19436)	This is GetProperty, a member of class IWebBrowserAppDisp.
	GoBack (see page 19436)	This is GoBack, a member of class IWebBrowserAppDisp.
	GoForward (see page 19436)	This is GoForward, a member of class IWebBrowserAppDisp.
	GoHome (see page 19437)	This is GoHome, a member of class IWebBrowserAppDisp.
	GoSearch (see page 19437)	This is GoSearch, a member of class IWebBrowserAppDisp.
	Navigate (see page 19437)	This is Navigate, a member of class IWebBrowserAppDisp.
	PutProperty (see page 19437)	This is PutProperty, a member of class IWebBrowserAppDisp.
	Quit (see page 19438)	This is Quit, a member of class IWebBrowserAppDisp.
	Refresh (see page 19438)	This is Refresh, a member of class IWebBrowserAppDisp.
	Refresh2 (see page 19438)	This is Refresh2, a member of class IWebBrowserAppDisp.
	Stop (see page 19438)	This is Stop, a member of class IWebBrowserAppDisp.

Public Properties

	Name	Description
	Application (see page 19439)	This is Application, a member of class IWebBrowserAppDisp.
	Busy (see page 19439)	This is Busy, a member of class IWebBrowserAppDisp.
	Container (see page 19439)	This is Container, a member of class IWebBrowserAppDisp.
	Document (see page 19440)	This is Document, a member of class IWebBrowserAppDisp.
	FullName (see page 19440)	This is FullName, a member of class IWebBrowserAppDisp.
	FullScreen (see page 19440)	This is FullScreen, a member of class IWebBrowserAppDisp.
	Height (see page 19440)	This is Height, a member of class IWebBrowserAppDisp.
	HWnd (see page 19441)	This is HWnd, a member of class IWebBrowserAppDisp.
	Left (see page 19441)	This is Left, a member of class IWebBrowserAppDisp.
	LocationName (see page 19441)	This is LocationName, a member of class IWebBrowserAppDisp.
	LocationURL (see page 19441)	This is LocationURL, a member of class IWebBrowserAppDisp.
	MenuBar (see page 19441)	This is MenuBar, a member of class IWebBrowserAppDisp.
	Name (see page 19442)	This is Name, a member of class IWebBrowserAppDisp.
	Parent (see page 19442)	This is Parent, a member of class IWebBrowserAppDisp.
	Path (see page 19442)	This is Path, a member of class IWebBrowserAppDisp.
	StatusBar (see page 19442)	This is StatusBar, a member of class IWebBrowserAppDisp.
	StatusText (see page 19443)	This is StatusText, a member of class IWebBrowserAppDisp.
	ToolBar (see page 19443)	This is ToolBar, a member of class IWebBrowserAppDisp.
	Top (see page 19443)	This is Top, a member of class IWebBrowserAppDisp.
	TopLevelContainer (see page 19443)	This is TopLevelContainer, a member of class IWebBrowserAppDisp.
	type_ (see page 19443)	This is type_, a member of class IWebBrowserAppDisp.
	Visible (see page 19444)	This is Visible, a member of class IWebBrowserAppDisp.
	Width (see page 19444)	This is Width, a member of class IWebBrowserAppDisp.

132.2.30.2 IWebBrowserAppDisp Methods

The methods of the IWebBrowserAppDisp class are listed here.

Public Methods

	Name	Description
	ClientToWindow (see page 19436)	This is ClientToWindow, a member of class IWebBrowserAppDisp.

	GetProperty (see page 19436)	This is GetProperty, a member of class IWebBrowserAppDisp.
	GoBack (see page 19436)	This is GoBack, a member of class IWebBrowserAppDisp.
	GoForward (see page 19436)	This is GoForward, a member of class IWebBrowserAppDisp.
	GoHome (see page 19437)	This is GoHome, a member of class IWebBrowserAppDisp.
	GoSearch (see page 19437)	This is GoSearch, a member of class IWebBrowserAppDisp.
	Navigate (see page 19437)	This is Navigate, a member of class IWebBrowserAppDisp.
	PutProperty (see page 19437)	This is PutProperty, a member of class IWebBrowserAppDisp.
	Quit (see page 19438)	This is Quit, a member of class IWebBrowserAppDisp.
	Refresh (see page 19438)	This is Refresh, a member of class IWebBrowserAppDisp.
	Refresh2 (see page 19438)	This is Refresh2, a member of class IWebBrowserAppDisp.
	Stop (see page 19438)	This is Stop, a member of class IWebBrowserAppDisp.

132.2.30.2.1 IWebBrowserAppDisp.ClientToWindow Method

Pascal

```
procedure ClientToWindow(var pcx: SYSINT; var pcy: SYSINT);
```

C++

```
__fastcall ClientToWindow(SYSINT pcx, SYSINT pcy);
```

Description

This is ClientToWindow, a member of class IWebBrowserAppDisp.

132.2.30.2.2 IWebBrowserAppDisp.GetProperty Method

Pascal

```
function GetProperty(const Property_: WideString): OleVariant;
```

C++

```
__fastcall OleVariant GetProperty(const BSTR Property_);
```

Description

This is GetProperty, a member of class IWebBrowserAppDisp.

132.2.30.2.3 IWebBrowserAppDisp.GoBack Method

Pascal

```
procedure GoBack;
```

C++

```
__fastcall GoBack();
```

Description

This is GoBack, a member of class IWebBrowserAppDisp.

132.2.30.2.4 IWebBrowserAppDisp.GoForward Method

Pascal

```
procedure GoForward;
```

C++

```
__fastcall GoForward();
```

Description

This is GoForward, a member of class IWebBrowserAppDisp.

132.2.30.2.5 IWebBrowserAppDisp.GoHome Method

Pascal

```
procedure GoHome;
```

C++

```
__fastcall GoHome();
```

Description

This is GoHome, a member of class IWebBrowserAppDisp.

132.2.30.2.6 IWebBrowserAppDisp.GoSearch Method

Pascal

```
procedure GoSearch;
```

C++

```
__fastcall GoSearch();
```

Description

This is GoSearch, a member of class IWebBrowserAppDisp.

132.2.30.2.7 IWebBrowserAppDisp.Navigate Method

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant);
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate, a member of class IWebBrowserAppDisp.

132.2.30.2.8 IWebBrowserAppDisp.PutProperty Method

Pascal

```
procedure PutProperty(const Property_: WideString; vtValue: OleVariant);
```

C++

```
__fastcall PutProperty(const BSTR Property_, OleVariant vtValue);
```

Description

This is PutProperty, a member of class IWebBrowserAppDisp.

132.2.30.2.9 IWebBrowserAppDisp.Quit Method

Pascal

```
procedure Quit;
```

C++

```
__fastcall Quit();
```

Description

This is Quit, a member of class IWebBrowserAppDisp.

132.2.30.2.10 IWebBrowserAppDisp.Refresh Method

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

This is Refresh, a member of class IWebBrowserAppDisp.

132.2.30.2.11 IWebBrowserAppDisp.Refresh2 Method

Pascal

```
procedure Refresh2(var Level: OleVariant);
```

C++

```
__fastcall Refresh2(OleVariant Level);
```

Description

This is Refresh2, a member of class IWebBrowserAppDisp.

132.2.30.2.12 IWebBrowserAppDisp.Stop Method

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

This is Stop, a member of class IWebBrowserAppDisp.

132.2.30.3 IWebBrowserAppDisp Properties

The properties of the IWebBrowserAppDisp class are listed here.

Public Properties

	Name	Description
	Application (see page 19439)	This is Application, a member of class IWebBrowserAppDisp.
	Busy (see page 19439)	This is Busy, a member of class IWebBrowserAppDisp.
	Container (see page 19439)	This is Container, a member of class IWebBrowserAppDisp.
	Document (see page 19440)	This is Document, a member of class IWebBrowserAppDisp.
	FullName (see page 19440)	This is FullName, a member of class IWebBrowserAppDisp.
	FullScreen (see page 19440)	This is FullScreen, a member of class IWebBrowserAppDisp.
	Height (see page 19440)	This is Height, a member of class IWebBrowserAppDisp.
	HWnd (see page 19441)	This is HWnd, a member of class IWebBrowserAppDisp.
	Left (see page 19441)	This is Left, a member of class IWebBrowserAppDisp.
	LocationName (see page 19441)	This is LocationName, a member of class IWebBrowserAppDisp.
	LocationURL (see page 19441)	This is LocationURL, a member of class IWebBrowserAppDisp.
	MenuBar (see page 19441)	This is MenuBar, a member of class IWebBrowserAppDisp.
	Name (see page 19442)	This is Name, a member of class IWebBrowserAppDisp.
	Parent (see page 19442)	This is Parent, a member of class IWebBrowserAppDisp.
	Path (see page 19442)	This is Path, a member of class IWebBrowserAppDisp.
	StatusBar (see page 19442)	This is StatusBar, a member of class IWebBrowserAppDisp.
	StatusText (see page 19443)	This is StatusText, a member of class IWebBrowserAppDisp.
	ToolBar (see page 19443)	This is ToolBar, a member of class IWebBrowserAppDisp.
	Top (see page 19443)	This is Top, a member of class IWebBrowserAppDisp.
	TopLevelContainer (see page 19443)	This is TopLevelContainer, a member of class IWebBrowserAppDisp.
	type_ (see page 19443)	This is type_, a member of class IWebBrowserAppDisp.
	Visible (see page 19444)	This is Visible, a member of class IWebBrowserAppDisp.
	Width (see page 19444)	This is Width, a member of class IWebBrowserAppDisp.

132.2.30.3.1 IWebBrowserAppDisp.Application Property

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

This is Application, a member of class IWebBrowserAppDisp.

132.2.30.3.2 IWebBrowserAppDisp.Busy Property

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

This is Busy, a member of class IWebBrowserAppDisp.

132.2.30.3.3 IWebBrowserAppDisp.Container Property

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

This is Container, a member of class IWebBrowserAppDisp.

132.2.30.3.4 IWebBrowserAppDisp.Document Property

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

This is Document, a member of class IWebBrowserAppDisp.

132.2.30.3.5 IWebBrowserAppDisp.FullName Property

Pascal

```
property FullName: WideString;
```

C++

```
__property BSTR FullName;
```

Description

This is FullName, a member of class IWebBrowserAppDisp.

132.2.30.3.6 IWebBrowserAppDisp.FullScreen Property

Pascal

```
property FullScreen: WordBool;
```

C++

```
__property WordBool FullScreen;
```

Description

This is FullScreen, a member of class IWebBrowserAppDisp.

132.2.30.3.7 IWebBrowserAppDisp.Height Property

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

This is Height, a member of class IWebBrowserAppDisp.

132.2.30.3.8 IWebBrowserAppDisp.HWnd Property

Pascal

```
property HWnd: Integer;
```

C++

```
__property int HWnd;
```

Description

This is HWnd, a member of class IWebBrowserAppDisp.

132.2.30.3.9 IWebBrowserAppDisp.Left Property

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

This is Left, a member of class IWebBrowserAppDisp.

132.2.30.3.10 IWebBrowserAppDisp.LocationName Property

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

This is LocationName, a member of class IWebBrowserAppDisp.

132.2.30.3.11 IWebBrowserAppDisp.LocationURL Property

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

This is LocationURL, a member of class IWebBrowserAppDisp.

132.2.30.3.12 IWebBrowserAppDisp.MenuBar Property

Pascal

```
property MenuBar: WordBool;
```

C++

```
__property WordBool MenuBar;
```

Description

This is MenuBar, a member of class IWebBrowserAppDisp.

132.2.30.3.13 IWebBrowserAppDisp.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IWebBrowserAppDisp.

132.2.30.3.14 IWebBrowserAppDisp.Parent Property

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

This is Parent, a member of class IWebBrowserAppDisp.

132.2.30.3.15 IWebBrowserAppDisp.Path Property

Pascal

```
property Path: WideString;
```

C++

```
__property BSTR Path;
```

Description

This is Path, a member of class IWebBrowserAppDisp.

132.2.30.3.16 IWebBrowserAppDisp.StatusBar Property

Pascal

```
property StatusBar: WordBool;
```

C++

```
__property WordBool StatusBar;
```

Description

This is StatusBar, a member of class IWebBrowserAppDisp.

132.2.30.3.17 IWebBrowserAppDisp.StatusText Property

Pascal

```
property StatusText: WideString;
```

C++

```
__property BSTR StatusText;
```

Description

This is StatusText, a member of class IWebBrowserAppDisp.

132.2.30.3.18 IWebBrowserAppDisp.ToolBar Property

Pascal

```
property ToolBar: SYSINT;
```

C++

```
__property SYSINT ToolBar;
```

Description

This is ToolBar, a member of class IWebBrowserAppDisp.

132.2.30.3.19 IWebBrowserAppDisp.Top Property

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

This is Top, a member of class IWebBrowserAppDisp.

132.2.30.3.20 IWebBrowserAppDisp.TopLevelContainer Property

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

This is TopLevelContainer, a member of class IWebBrowserAppDisp.

132.2.30.3.21 IWebBrowserAppDisp.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class IWebBrowserAppDisp.

132.2.30.3.22 IWebBrowserAppDisp.Visible Property

Pascal

```
property Visible: WordBool;
```

C++

```
__property WordBool Visible;
```

Description

This is Visible, a member of class IWebBrowserAppDisp.

132.2.30.3.23 IWebBrowserAppDisp.Width Property

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

This is Width, a member of class IWebBrowserAppDisp.

132.2.31 IWebBrowserDisp Interface

Class Hierarchy

SHDocVw.IWebBrowserDisp

Pascal

```
IWebBrowserDisp = dispinterface;
```

C++

```
class IWebBrowserDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IWebBrowserDisp.

132.2.31.1 IWebBrowserDisp Members

The following tables list the members exposed by IWebBrowserDisp.

Public Methods

	Name	Description
	GoBack (see page 19445)	This is GoBack, a member of class IWebBrowserDisp.
	GoForward (see page 19446)	This is GoForward, a member of class IWebBrowserDisp.
	GoHome (see page 19446)	This is GoHome, a member of class IWebBrowserDisp.
	GoSearch (see page 19446)	This is GoSearch, a member of class IWebBrowserDisp.
	Navigate (see page 19446)	This is Navigate, a member of class IWebBrowserDisp.
	Refresh (see page 19446)	This is Refresh, a member of class IWebBrowserDisp.
	Refresh2 (see page 19447)	This is Refresh2, a member of class IWebBrowserDisp.
	Stop (see page 19447)	This is Stop, a member of class IWebBrowserDisp.

Public Properties

	Name	Description
	Application (see page 19448)	This is Application, a member of class IWebBrowserDisp.
	Busy (see page 19448)	This is Busy, a member of class IWebBrowserDisp.
	Container (see page 19448)	This is Container, a member of class IWebBrowserDisp.
	Document (see page 19448)	This is Document, a member of class IWebBrowserDisp.
	Height (see page 19448)	This is Height, a member of class IWebBrowserDisp.
	Left (see page 19449)	This is Left, a member of class IWebBrowserDisp.
	LocationName (see page 19449)	This is LocationName, a member of class IWebBrowserDisp.
	LocationURL (see page 19449)	This is LocationURL, a member of class IWebBrowserDisp.
	Parent (see page 19449)	This is Parent, a member of class IWebBrowserDisp.
	Top (see page 19450)	This is Top, a member of class IWebBrowserDisp.
	TopLevelContainer (see page 19450)	This is TopLevelContainer, a member of class IWebBrowserDisp.
	type_ (see page 19450)	This is type_, a member of class IWebBrowserDisp.
	Width (see page 19450)	This is Width, a member of class IWebBrowserDisp.

132.2.31.2 IWebBrowserDisp Methods

The methods of the IWebBrowserDisp class are listed here.

Public Methods

	Name	Description
	GoBack (see page 19445)	This is GoBack, a member of class IWebBrowserDisp.
	GoForward (see page 19446)	This is GoForward, a member of class IWebBrowserDisp.
	GoHome (see page 19446)	This is GoHome, a member of class IWebBrowserDisp.
	GoSearch (see page 19446)	This is GoSearch, a member of class IWebBrowserDisp.
	Navigate (see page 19446)	This is Navigate, a member of class IWebBrowserDisp.
	Refresh (see page 19446)	This is Refresh, a member of class IWebBrowserDisp.
	Refresh2 (see page 19447)	This is Refresh2, a member of class IWebBrowserDisp.
	Stop (see page 19447)	This is Stop, a member of class IWebBrowserDisp.

132.2.31.2.1 IWebBrowserDisp.GoBack Method

Pascal

```
procedure GoBack;
```

C++

```
__fastcall GoBack();
```

Description

This is GoBack, a member of class IWebBrowserDisp.

132.2.31.2.2 IWebBrowserDisp.GoForward Method

Pascal

```
procedure GoForward;
```

C++

```
__fastcall GoForward();
```

Description

This is GoForward, a member of class IWebBrowserDisp.

132.2.31.2.3 IWebBrowserDisp.GoHome Method

Pascal

```
procedure GoHome;
```

C++

```
__fastcall GoHome();
```

Description

This is GoHome, a member of class IWebBrowserDisp.

132.2.31.2.4 IWebBrowserDisp.GoSearch Method

Pascal

```
procedure GoSearch;
```

C++

```
__fastcall GoSearch();
```

Description

This is GoSearch, a member of class IWebBrowserDisp.

132.2.31.2.5 IWebBrowserDisp.Navigate Method

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant);
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate, a member of class IWebBrowserDisp.

132.2.31.2.6 IWebBrowserDisp.Refresh Method

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

This is Refresh, a member of class IWebBrowserDisp.

132.2.31.2.7 IWebBrowserDisp.Refresh2 Method

Pascal

```
procedure Refresh2(var Level: OleVariant);
```

C++

```
__fastcall Refresh2(OleVariant Level);
```

Description

This is Refresh2, a member of class IWebBrowserDisp.

132.2.31.2.8 IWebBrowserDisp.Stop Method

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

This is Stop, a member of class IWebBrowserDisp.

132.2.31.3 IWebBrowserDisp Properties

The properties of the IWebBrowserDisp class are listed here.

Public Properties

	Name	Description
	Application (see page 19448)	This is Application, a member of class IWebBrowserDisp.
	Busy (see page 19448)	This is Busy, a member of class IWebBrowserDisp.
	Container (see page 19448)	This is Container, a member of class IWebBrowserDisp.
	Document (see page 19448)	This is Document, a member of class IWebBrowserDisp.
	Height (see page 19448)	This is Height, a member of class IWebBrowserDisp.
	Left (see page 19449)	This is Left, a member of class IWebBrowserDisp.
	LocationName (see page 19449)	This is LocationName, a member of class IWebBrowserDisp.
	LocationURL (see page 19449)	This is LocationURL, a member of class IWebBrowserDisp.
	Parent (see page 19449)	This is Parent, a member of class IWebBrowserDisp.
	Top (see page 19450)	This is Top, a member of class IWebBrowserDisp.
	TopLevelContainer (see page 19450)	This is TopLevelContainer, a member of class IWebBrowserDisp.
	type_ (see page 19450)	This is type_, a member of class IWebBrowserDisp.
	Width (see page 19450)	This is Width, a member of class IWebBrowserDisp.

132.2.31.3.1 IWebBrowserDisp.Application Property

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

This is Application, a member of class IWebBrowserDisp.

132.2.31.3.2 IWebBrowserDisp.Busy Property

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

This is Busy, a member of class IWebBrowserDisp.

132.2.31.3.3 IWebBrowserDisp.Container Property

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

This is Container, a member of class IWebBrowserDisp.

132.2.31.3.4 IWebBrowserDisp.Document Property

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

This is Document, a member of class IWebBrowserDisp.

132.2.31.3.5 IWebBrowserDisp.Height Property

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

This is Height, a member of class IWebBrowserDisp.

132.2.31.3.6 IWebBrowserDisp.Left Property

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

This is Left, a member of class IWebBrowserDisp.

132.2.31.3.7 IWebBrowserDisp.LocationName Property

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

This is LocationName, a member of class IWebBrowserDisp.

132.2.31.3.8 IWebBrowserDisp.LocationURL Property

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

This is LocationURL, a member of class IWebBrowserDisp.

132.2.31.3.9 IWebBrowserDisp.Parent Property

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

This is Parent, a member of class IWebBrowserDisp.

132.2.31.3.10 IWebBrowserDisp.Top Property

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

This is Top, a member of class IWebBrowserDisp.

132.2.31.3.11 IWebBrowserDisp.TopLevelContainer Property

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

This is TopLevelContainer, a member of class IWebBrowserDisp.

132.2.31.3.12 IWebBrowserDisp.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class IWebBrowserDisp.

132.2.31.3.13 IWebBrowserDisp.Width Property

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

This is Width, a member of class IWebBrowserDisp.

132.3 Types

The following table lists types in this documentation.

Types

Name	Description
BrowserNavConstants (see page 19453)	This is type SHDocVw.BrowserNavConstants.
CommandStateChangeConstants (see page 19453)	This is type SHDocVw.CommandStateChangeConstants.
CScriptErrorList (see page 19453)	This is type SHDocVw.CScriptErrorList.
InternetExplorer (see page 19454)	This is type SHDocVw.InternetExplorer.
OLECMDEXECOPT (see page 19454)	OLECMDF (see page 19455) influences the way in which an OLECMDID (see page 19455) command is executed.
OLECMDF (see page 19455)	OLECMDF represents the type of support an Automation object provides for a specific command at a specific time.
OLECMDID (see page 19455)	OLECMDID represents a command that can be executed by an Automation object.
POleVariant1 (see page 19456)	This is type SHDocVw.POleVariant1.
RefreshConstants (see page 19456)	This is type SHDocVw.RefreshConstants.
SearchAssistantOC (see page 19456)	This is type SHDocVw.SearchAssistantOC.
SecureLockIconConstants (see page 19457)	This is type SHDocVw.SecureLockIconConstants.
ShellBrowserWindow (see page 19457)	This is type SHDocVw.ShellBrowserWindow.
ShellNameSpace (see page 19457)	This is type SHDocVw.ShellNameSpace.
ShellUIHelper (see page 19458)	This is type SHDocVw.ShellUIHelper.
ShellWindowFindWindowOptions (see page 19458)	This is type SHDocVw.ShellWindowFindWindowOptions.
ShellWindows (see page 19458)	This is type SHDocVw.ShellWindows.
ShellWindowTypeConstants (see page 19459)	This is type SHDocVw.ShellWindowTypeConstants.
tagREADYSTATE (see page 19459)	This is type SHDocVw.tagREADYSTATE.
TInternetExplorerBeforeNavigate2 (see page 19459)	This is type SHDocVw.TInternetExplorerBeforeNavigate2.
TInternetExplorerClientToHostWindow (see page 19460)	This is type SHDocVw.TInternetExplorerClientToHostWindow.
TInternetExplorerCommandStateChange (see page 19460)	This is type SHDocVw.TInternetExplorerCommandStateChange.
TInternetExplorerDocumentComplete (see page 19460)	This is type SHDocVw.TInternetExplorerDocumentComplete.
TInternetExplorerFileDownload (see page 19461)	This is type SHDocVw.TInternetExplorerFileDownload.
TInternetExplorerNavigateComplete2 (see page 19461)	This is type SHDocVw.TInternetExplorerNavigateComplete2.
TInternetExplorerNavigateError (see page 19461)	This is type SHDocVw.TInternetExplorerNavigateError.
TInternetExplorerNewWindow2 (see page 19462)	This is type SHDocVw.TInternetExplorerNewWindow2.
TInternetExplorerOnFullScreen (see page 19462)	This is type SHDocVw.TInternetExplorerOnFullScreen.
TInternetExplorerOnMenuBar (see page 19462)	This is type SHDocVw.TInternetExplorerOnMenuBar.
TInternetExplorerOnStatusBar (see page 19463)	This is type SHDocVw.TInternetExplorerOnStatusBar.
TInternetExplorerOnTheaterMode (see page 19463)	This is type SHDocVw.TInternetExplorerOnTheaterMode.
TInternetExplorerOnToolBar (see page 19463)	This is type SHDocVw.TInternetExplorerOnToolBar.
TInternetExplorerOnVisible (see page 19464)	This is type SHDocVw.TInternetExplorerOnVisible.
TInternetExplorerPrintTemplateInstantiation (see page 19464)	This is type SHDocVw.TInternetExplorerPrintTemplateInstantiation.
TInternetExplorerPrintTemplateTeardown (see page 19464)	This is type SHDocVw.TInternetExplorerPrintTemplateTeardown.
TInternetExplorerPrivacyImpactedStateChange (see page 19465)	This is type SHDocVw.TInternetExplorerPrivacyImpactedStateChange.
TInternetExplorerProgressChange (see page 19465)	This is type SHDocVw.TInternetExplorerProgressChange.
TInternetExplorerPropertyChange (see page 19465)	This is type SHDocVw.TInternetExplorerPropertyChange.
TInternetExplorerSetSecureLockIcon (see page 19466)	This is type SHDocVw.TInternetExplorerSetSecureLockIcon.
TInternetExplorerStatusTextChange (see page 19466)	This is type SHDocVw.TInternetExplorerStatusTextChange.
TInternetExplorerTitleChange (see page 19466)	This is type SHDocVw.TInternetExplorerTitleChange.
TInternetExplorerUpdatePageStatus (see page 19467)	This is type SHDocVw.TInternetExplorerUpdatePageStatus.
TInternetExplorerWindowClosing (see page 19467)	This is type SHDocVw.TInternetExplorerWindowClosing.

TInternetExplorerWindowSetHeight (see page 19467)	This is type SHDocVw.TInternetExplorerWindowSetHeight.
TInternetExplorerWindowSetLeft (see page 19468)	This is type SHDocVw.TInternetExplorerWindowSetLeft.
TInternetExplorerWindowSetResizable (see page 19468)	This is type SHDocVw.TInternetExplorerWindowSetResizable.
TInternetExplorerWindowSetTop (see page 19468)	This is type SHDocVw.TInternetExplorerWindowSetTop.
TInternetExplorerWindowSetWidth (see page 19469)	This is type SHDocVw.TInternetExplorerWindowSetWidth.
TShellNameSpaceFavoritesSelectionChange (see page 19469)	This is type SHDocVw.TShellNameSpaceFavoritesSelectionChange.
TShellWindowsWindowRegistered (see page 19470)	This is type SHDocVw.TShellWindowsWindowRegistered.
TShellWindowsWindowRevoked (see page 19470)	This is type SHDocVw.TShellWindowsWindowRevoked.
TWebBrowser_V1BeforeNavigate (see page 19470)	This is type SHDocVw.TWebBrowser_V1BeforeNavigate.
TWebBrowser_V1CommandStateChange (see page 19471)	This is type SHDocVw.TWebBrowser_V1CommandStateChange.
TWebBrowser_V1FrameBeforeNavigate (see page 19471)	This is type SHDocVw.TWebBrowser_V1FrameBeforeNavigate.
TWebBrowser_V1FrameNavigateComplete (see page 19471)	This is type SHDocVw.TWebBrowser_V1FrameNavigateComplete.
TWebBrowser_V1FrameNewWindow (see page 19472)	This is type SHDocVw.TWebBrowser_V1FrameNewWindow.
TWebBrowser_V1NavigateComplete (see page 19472)	This is type SHDocVw.TWebBrowser_V1NavigateComplete.
TWebBrowser_V1NewWindow (see page 19472)	This is type SHDocVw.TWebBrowser_V1NewWindow.
TWebBrowser_V1ProgressChange (see page 19473)	This is type SHDocVw.TWebBrowser_V1ProgressChange.
TWebBrowser_V1PropertyChange (see page 19473)	This is type SHDocVw.TWebBrowser_V1PropertyChange.
TWebBrowser_V1Quit (see page 19474)	This is type SHDocVw.TWebBrowser_V1Quit.
TWebBrowser_V1StatusTextChange (see page 19474)	This is type SHDocVw.TWebBrowser_V1StatusTextChange.
TWebBrowser_V1TitleChange (see page 19474)	This is type SHDocVw.TWebBrowser_V1TitleChange.
TWebBrowserBeforeNavigate2 (see page 19475)	This is type SHDocVw.TWebBrowserBeforeNavigate2.
TWebBrowserClientToHostWindow (see page 19475)	This is type SHDocVw.TWebBrowserClientToHostWindow.
TWebBrowserCommandStateChange (see page 19475)	This is type SHDocVw.TWebBrowserCommandStateChange.
TWebBrowserDocumentComplete (see page 19476)	This is type SHDocVw.TWebBrowserDocumentComplete.
TWebBrowserFileDownload (see page 19476)	This is type SHDocVw.TWebBrowserFileDownload.
TWebBrowserNavigateComplete2 (see page 19476)	This is type SHDocVw.TWebBrowserNavigateComplete2.
TWebBrowserNavigateError (see page 19477)	This is type SHDocVw.TWebBrowserNavigateError.
TWebBrowserNewWindow2 (see page 19477)	This is type SHDocVw.TWebBrowserNewWindow2.
TWebBrowserOnFullScreen (see page 19477)	This is type SHDocVw.TWebBrowserOnFullScreen.
TWebBrowserOnMenuBar (see page 19478)	This is type SHDocVw.TWebBrowserOnMenuBar.
TWebBrowserOnStatusBar (see page 19478)	This is type SHDocVw.TWebBrowserOnStatusBar.
TWebBrowserOnTheaterMode (see page 19478)	This is type SHDocVw.TWebBrowserOnTheaterMode.
TWebBrowserOnToolBar (see page 19479)	This is type SHDocVw.TWebBrowserOnToolBar.
TWebBrowserOnVisible (see page 19479)	This is type SHDocVw.TWebBrowserOnVisible.
TWebBrowserPrintTemplateInstantiation (see page 19479)	This is type SHDocVw.TWebBrowserPrintTemplateInstantiation.
TWebBrowserPrintTemplateTeardown (see page 19480)	This is type SHDocVw.TWebBrowserPrintTemplateTeardown.
TWebBrowserPrivacyImpactedStateChange (see page 19480)	This is type SHDocVw.TWebBrowserPrivacyImpactedStateChange.
TWebBrowserProgressChange (see page 19480)	This is type SHDocVw.TWebBrowserProgressChange.
TWebBrowserPropertyChange (see page 19481)	This is type SHDocVw.TWebBrowserPropertyChange.
TWebBrowserSetSecureLockIcon (see page 19481)	This is type SHDocVw.TWebBrowserSetSecureLockIcon.
TWebBrowserStatusTextChange (see page 19481)	This is type SHDocVw.TWebBrowserStatusTextChange.
TWebBrowserTitleChange (see page 19482)	This is type SHDocVw.TWebBrowserTitleChange.
TWebBrowserUpdatePageStatus (see page 19482)	This is type SHDocVw.TWebBrowserUpdatePageStatus.
TWebBrowserWindowClosing (see page 19482)	This is type SHDocVw.TWebBrowserWindowClosing.
TWebBrowserWindowSetHeight (see page 19483)	This is type SHDocVw.TWebBrowserWindowSetHeight.
TWebBrowserWindowSetLeft (see page 19483)	This is type SHDocVw.TWebBrowserWindowSetLeft.
TWebBrowserWindowSetResizable (see page 19483)	This is type SHDocVw.TWebBrowserWindowSetResizable.
TWebBrowserWindowSetTop (see page 19484)	This is type SHDocVw.TWebBrowserWindowSetTop.
TWebBrowserWindowSetWidth (see page 19484)	This is type SHDocVw.TWebBrowserWindowSetWidth.
WebBrowser (see page 19484)	This is type SHDocVw.WebBrowser.
WebBrowser_V1 (see page 19485)	This is type SHDocVw.WebBrowser_V1.

132.3.1 SHDocVw.BrowserNavConstants Type

Pascal

```
BrowserNavConstants = TOleEnum;
```

C++

```
TOleEnum BrowserNavConstants;
```

File

SHDocVw

Description

This is type SHDocVw.BrowserNavConstants.

132.3.2 SHDocVw.CommandStateChangeConstants Type

Pascal

```
CommandStateChangeConstants = TOleEnum;
```

C++

```
TOleEnum CommandStateChangeConstants;
```

File

SHDocVw

Description

This is type SHDocVw.CommandStateChangeConstants.

132.3.3 SHDocVw.CScriptErrorList Type

Pascal

```
CScriptErrorList = IScriptErrorList;
```

C++

```
IScriptErrorList CScriptErrorList;
```

File

SHDocVw

Description

This is type SHDocVw.CScriptErrorList.

132.3.4 SHDocVw.InternetExplorer Type

Pascal

```
InternetExplorer = IWebBrowser2;
```

C++

```
IWebBrowser2 InternetExplorer;
```

File

SHDocVw

Description

This is type SHDocVw.InternetExplorer.

132.3.5 SHDocVw.OLECMDEXECOPT Type

OLECMDF (see page 19455) influences the way in which an OLECMDID (see page 19455) command is executed.

Pascal

```
OLECMDEXECOPT = TOleEnum;
```

C++

```
TOleEnum OLECMDEXECOPT;
```

File

SHDocVw

Description

In C++ code imported from the shdocvw type library, OLECMDEXECOPT indicates what an Automation object should do in response to an OLECMDID (see page 19455) command. The following table lists the possible values:

Constant	Value	Meaning
OLECMDEXECOPT_DODEFAULT (see page 19503)	0	Use the default behavior, whether prompting the user for input or not.
OLECMDEXECOPT_PROMPTUSER (see page 19503)	1	Execute the command after obtaining user input.
OLECMDEXECOPT_DONTPROMPTUSER (see page 19503)	2	Execute the command without prompting the user.
OLECMDEXECOPT_SHOWHELP (see page 19504)	3	Show help for the command, but do not execute it.

See Also

OLECMDID (see page 19455)

132.3.6 SHDocVw.OLECMDF Type

OLECMDF represents the type of support an Automation object provides for a specific command at a specific time.

Pascal

```
OLECMDF = TOleEnum;
```

C++

```
TOleEnum OLECMDF;
```

File

SHDocVw

Description

In C++ code imported from the shdocvw type library, OLECMDF is a set of flags that together indicate how a command can be used. These values can be or'd together:

Value	Meaning	
OLECMDF_SUPPORTED (see page 19506)	The Automation object supports the command (but it may not be enabled).	
OLECMDF_ENABLED (see page 19504)	The command is currently enabled.	
OLECMDF_LATCHED (see page 19505)	The command is an on/off toggle that is currently on.	
OLECMDF_NINCHED (see page 19505)	Not currently used.	
OLECMDF_INVISIBLE (see page 19505)	Not currently used.	
OLECMDF_DEFHIDEONCTXTMENU (see page 19504)	Not currently used.	

See Also

OLECMDID (see page 19455)

132.3.7 SHDocVw.OLECMDID Type

OLECMDID represents a command that can be executed by an Automation object.

Pascal

```
OLECMDID = TOleEnum;
```

C++

```
TOleEnum OLECMDID;
```

File

SHDocVw

Description

In C++ code imported from the shdocvw type library, OLECMDID represents a set of standard commands that can be executed by many Automation objects. Typically, each command represents a command in the user interface of the Automation object, such as a menu command. For specific details on how the Automation object interprets a particular command, see the documentation for the Automation object.

See Also

OLECMDEXECOPT (see page 19454)

132.3.8 SHDocVw.POleVariant1 Type

Pascal

```
POleVariant1 = ^OleVariant;
```

C++

```
^OleVariant POleVariant1;
```

File

SHDocVw

Description

This is type SHDocVw.POleVariant1.

132.3.9 SHDocVw.RefreshConstants Type

Pascal

```
RefreshConstants = TOleEnum;
```

C++

```
TOleEnum RefreshConstants;
```

File

SHDocVw

Description

This is type SHDocVw.RefreshConstants.

132.3.10 SHDocVw.SearchAssistantOC Type

Pascal

```
SearchAssistantOC = ISearchAssistantOC3;
```

C++

```
ISearchAssistantOC3 SearchAssistantOC;
```

File

SHDocVw

Description

This is type SHDocVw.SearchAssistantOC.

132.3.11 SHDocVw.SecureLockIconConstants Type

Pascal

```
SecureLockIconConstants = TOleEnum;
```

C++

```
TOleEnum SecureLockIconConstants;
```

File

SHDocVw

Description

This is type SHDocVw.SecureLockIconConstants.

132.3.12 SHDocVw.ShellBrowserWindow Type

Pascal

```
ShellBrowserWindow = IWebBrowser2;
```

C++

```
IWebBrowser2 ShellBrowserWindow;
```

File

SHDocVw

Description

This is type SHDocVw.ShellBrowserWindow.

132.3.13 SHDocVw.ShellNameSpace Type

Pascal

```
ShellNameSpace = IShellNameSpace;
```

C++

```
IShellNameSpace ShellNameSpace;
```

File

SHDocVw

Description

This is type SHDocVw.ShellNameSpace.

132.3.14 SHDocVw.ShellUIHelper Type

Pascal

```
ShellUIHelper = IShellUIHelper;
```

C++

```
IShellUIHelper ShellUIHelper;
```

File

SHDocVw

Description

This is type SHDocVw.ShellUIHelper.

132.3.15 SHDocVw.ShellWindowFindWindowOptions Type

Pascal

```
ShellWindowFindWindowOptions = TOleEnum;
```

C++

```
TOleEnum ShellWindowFindWindowOptions;
```

File

SHDocVw

Description

This is type SHDocVw.ShellWindowFindWindowOptions.

132.3.16 SHDocVw.ShellWindows Type

Pascal

```
ShellWindows = IShellWindows;
```

C++

```
IShellWindows ShellWindows;
```

File

SHDocVw

Description

This is type SHDocVw.ShellWindows.

132.3.17 SHDocVw.ShellWindowTypeConstants Type

Pascal

```
ShellWindowTypeConstants = TOleEnum;
```

C++

```
TOleEnum ShellWindowTypeConstants;
```

File

SHDocVw

Description

This is type SHDocVw.ShellWindowTypeConstants.

132.3.18 SHDocVw.tagREADYSTATE Type

Pascal

```
tagREADYSTATE = TOleEnum;
```

C++

```
TOleEnum tagREADYSTATE;
```

File

SHDocVw

Description

This is type SHDocVw.tagREADYSTATE.

132.3.19 SHDocVw.TInternetExplorerBeforeNavigate2 Type

Pascal

```
TInternetExplorerBeforeNavigate2 = procedure (ASender: TObject; const pDisp: IDispatch; var
URL: OleVariant; var Flags: OleVariant; var TargetFrameName: OleVariant; var PostData:
OleVariant; var Headers: OleVariant; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant; var Flags: OleVariant; var
TargetFrameName: OleVariant; var PostData: OleVariant; var Headers: OleVariant; var Cancel:
WordBool) ( TInternetExplorerBeforeNavigate2)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerBeforeNavigate2.

132.3.20 SHDocVw.TInternetExplorerClientToHostWindow Type

Pascal

```
TInternetExplorerClientToHostWindow = procedure (ASender: TObject; var CX: Integer; var CY:
Integer) of object;
```

C++

```
(ASender: TObject; var CX: Integer; var CY: Integer) ( TInternetExplorerClientToHostWindow)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerClientToHostWindow.

132.3.21 SHDocVw.TInternetExplorerCommandStateChange Type

Pascal

```
TInternetExplorerCommandStateChange = procedure (ASender: TObject; Command: Integer; Enable:
WordBool) of object;
```

C++

```
(ASender: TObject; Command: Integer; Enable: WordBool) (
TInternetExplorerCommandStateChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerCommandStateChange.

132.3.22 SHDocVw.TInternetExplorerDocumentComplete Type

Pascal

```
TInternetExplorerDocumentComplete = procedure (ASender: TObject; const pDisp: IDispatch; var
URL: OleVariant) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant) (
TInternetExplorerDocumentComplete)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerDocumentComplete.

132.3.23 SHDocVw.TInternetExplorerFileDownload Type

Pascal

```
TInternetExplorerFileDownload = procedure (ASender: TObject; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; var Cancel: WordBool) ( TInternetExplorerFileDownload)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerFileDownload.

132.3.24 SHDocVw.TInternetExplorerNavigateComplete2 Type

Pascal

```
TInternetExplorerNavigateComplete2 = procedure (ASender: TObject; const pDisp: IDispatch; var
URL: OleVariant) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant) (
TInternetExplorerNavigateComplete2)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerNavigateComplete2.

132.3.25 SHDocVw.TInternetExplorerNavigateError Type

Pascal

```
TInternetExplorerNavigateError = procedure (ASender: TObject; const pDisp: IDispatch; var URL:
OleVariant; var Frame: OleVariant; var StatusCode: OleVariant; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant; var Frame: OleVariant; var
```

```
StatusCode: OleVariant; var Cancel: WordBool) ( TInternetExplorerNavigateError)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerNavigateError.

132.3.26 SHDocVw.TInternetExplorerNewWindow2 Type

Pascal

```
TInternetExplorerNewWindow2 = procedure (ASender: TObject; var ppDisp: IDispatch; var Cancel:
WordBool) of object;
```

C++

```
(ASender: TObject; var ppDisp: IDispatch; var Cancel: WordBool) (
TInternetExplorerNewWindow2)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerNewWindow2.

132.3.27 SHDocVw.TInternetExplorerOnFullScreen Type

Pascal

```
TInternetExplorerOnFullScreen = procedure (ASender: TObject; FullScreen: WordBool) of object;
```

C++

```
(ASender: TObject; FullScreen: WordBool) ( TInternetExplorerOnFullScreen)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerOnFullScreen.

132.3.28 SHDocVw.TInternetExplorerOnMenuBar Type

Pascal

```
TInternetExplorerOnMenuBar = procedure (ASender: TObject; MenuBar: WordBool) of object;
```

C++

```
(ASender: TObject; MenuBar: WordBool) ( TInternetExplorerOnMenuBar)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerOnMenuBar.

132.3.29 SHDocVw.TInternetExplorerOnStatusBar Type

Pascal

```
TInternetExplorerOnStatusBar = procedure (ASender: TObject; StatusBar: WordBool) of object;
```

C++

```
(ASender: TObject; StatusBar: WordBool) ( TInternetExplorerOnStatusBar)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerOnStatusBar.

132.3.30 SHDocVw.TInternetExplorerOnTheaterMode Type

Pascal

```
TInternetExplorerOnTheaterMode = procedure (ASender: TObject; TheaterMode: WordBool) of object;
```

C++

```
(ASender: TObject; TheaterMode: WordBool) ( TInternetExplorerOnTheaterMode)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerOnTheaterMode.

132.3.31 SHDocVw.TInternetExplorerOnToolBar Type

Pascal

```
TInternetExplorerOnToolBar = procedure (ASender: TObject; ToolBar: WordBool) of object;
```

C++

```
(ASender: TObject; ToolBar: WordBool) ( TInternetExplorerOnToolBar)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerOnToolBar.

132.3.32 SHDocVw.TInternetExplorerOnVisible Type

Pascal

```
TInternetExplorerOnVisible = procedure (ASender: TObject; Visible: WordBool) of object;
```

C++

```
(ASender: TObject; Visible: WordBool) ( TInternetExplorerOnVisible)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerOnVisible.

132.3.33 SHDocVw.TInternetExplorerPrintTemplateInstantiation Type

Pascal

```
TInternetExplorerPrintTemplateInstantiation = procedure (ASender: TObject; const pDisp:
IDispatch) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch) ( TInternetExplorerPrintTemplateInstantiation)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerPrintTemplateInstantiation.

132.3.34 SHDocVw.TInternetExplorerPrintTemplateTeardown Type

Pascal

```
TInternetExplorerPrintTemplateTeardown = procedure (ASender: TObject; const pDisp: IDispatch)
of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch) ( TInternetExplorerPrintTemplateTeardown)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerPrintTemplateTeardown.

132.3.35 SHDocVw.TInternetExplorerPrivacyImpactedStateChange Type

Pascal

```
TInternetExplorerPrivacyImpactedStateChange = procedure (ASender: TObject; bImpacted:
WordBool) of object;
```

C++

```
(ASender: TObject; bImpacted: WordBool) ( TInternetExplorerPrivacyImpactedStateChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerPrivacyImpactedStateChange.

132.3.36 SHDocVw.TInternetExplorerProgressChange Type

Pascal

```
TInternetExplorerProgressChange = procedure (ASender: TObject; Progress: Integer; ProgressMax:
Integer) of object;
```

C++

```
(ASender: TObject; Progress: Integer; ProgressMax: Integer) (
TInternetExplorerProgressChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerProgressChange.

132.3.37 SHDocVw.TInternetExplorerPropertyChange Type

Pascal

```
TInternetExplorerPropertyChange = procedure (ASender: TObject; const szProperty: WideString)
of object;
```

C++

```
(ASender: TObject; const szProperty: WideString) ( TInternetExplorerPropertyChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerPropertyChange.

132.3.38 SHDocVw.TInternetExplorerSetSecureLockIcon Type

Pascal

```
TInternetExplorerSetSecureLockIcon = procedure (ASender: TObject; SecureLockIcon: Integer) of
object;
```

C++

```
(ASender: TObject; SecureLockIcon: Integer) ( TInternetExplorerSetSecureLockIcon)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerSetSecureLockIcon.

132.3.39 SHDocVw.TInternetExplorerStatusTextChange Type

Pascal

```
TInternetExplorerStatusTextChange = procedure (ASender: TObject; const Text: WideString) of
object;
```

C++

```
(ASender: TObject; const Text: WideString) ( TInternetExplorerStatusTextChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerStatusTextChange.

132.3.40 SHDocVw.TInternetExplorerTitleChange Type

Pascal

```
TInternetExplorerTitleChange = procedure (ASender: TObject; const Text: WideString) of object;
```

C++

```
(ASender: TObject; const Text: WideString) ( TInternetExplorerTitleChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerTitleChange.

132.3.41 SHDocVw.TInternetExplorerUpdatePageStatus Type

Pascal

```
TInternetExplorerUpdatePageStatus = procedure (ASender: TObject; const pDisp: IDispatch; var
nPage: OleVariant; var fDone: OleVariant) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var nPage: OleVariant; var fDone: OleVariant) (
TInternetExplorerUpdatePageStatus)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerUpdatePageStatus.

132.3.42 SHDocVw.TInternetExplorerWindowClosing Type

Pascal

```
TInternetExplorerWindowClosing = procedure (ASender: TObject; IsChildWindow: WordBool; var
Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; IsChildWindow: WordBool; var Cancel: WordBool) (
TInternetExplorerWindowClosing)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerWindowClosing.

132.3.43 SHDocVw.TInternetExplorerWindowSetHeight Type

Pascal

```
TInternetExplorerWindowSetHeight = procedure (ASender: TObject; Height: Integer) of object;
```

C++

```
(ASender: TObject; Height: Integer) ( TInternetExplorerWindowSetHeight)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerWindowSetHeight.

132.3.44 SHDocVw.TInternetExplorerWindowSetLeft Type

Pascal

```
TInternetExplorerWindowSetLeft = procedure (ASender: TObject; Left: Integer) of object;
```

C++

```
(ASender: TObject; Left: Integer) ( TInternetExplorerWindowSetLeft)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerWindowSetLeft.

132.3.45 SHDocVw.TInternetExplorerWindowSetResizable Type

Pascal

```
TInternetExplorerWindowSetResizable = procedure (ASender: TObject; Resizable: WordBool) of
object;
```

C++

```
(ASender: TObject; Resizable: WordBool) ( TInternetExplorerWindowSetResizable)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerWindowSetResizable.

132.3.46 SHDocVw.TInternetExplorerWindowSetTop Type

Pascal

```
TInternetExplorerWindowSetTop = procedure (ASender: TObject; Top: Integer) of object;
```

C++

```
(ASender: TObject; Top: Integer) ( TInternetExplorerWindowSetTop)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerWindowSetTop.

132.3.47 SHDocVw.TInternetExplorerWindowSetWidth Type

Pascal

```
TInternetExplorerWindowSetWidth = procedure (ASender: TObject; Width: Integer) of object;
```

C++

```
(ASender: TObject; Width: Integer) ( TInternetExplorerWindowSetWidth)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerWindowSetWidth.

132.3.48 SHDocVw.TShellNameSpaceFavoritesSelectionChange Type

Pascal

```
TShellNameSpaceFavoritesSelectionChange = procedure (ASender: TObject; cItems: Integer; hItem:
Integer; const strName: WideString; const strUrl: WideString; cVisits: Integer; const strDate:
WideString; fAvailableOffline: Integer) of object;
```

C++

```
(ASender: TObject; cItems: Integer; hItem: Integer; const strName: WideString; const strUrl:
WideString; cVisits: Integer; const strDate: WideString; fAvailableOffline: Integer) (
TShellNameSpaceFavoritesSelectionChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TShellNameSpaceFavoritesSelectionChange.

132.3.49 SHDocVw.TShellWindowsWindowRegistered Type

Pascal

```
TShellWindowsWindowRegistered = procedure (ASender: TObject; lCookie: Integer) of object;
```

C++

```
(ASender: TObject; lCookie: Integer) ( TShellWindowsWindowRegistered)();
```

File

SHDocVw

Description

This is type SHDocVw.TShellWindowsWindowRegistered.

132.3.50 SHDocVw.TShellWindowsWindowRevoked Type

Pascal

```
TShellWindowsWindowRevoked = procedure (ASender: TObject; lCookie: Integer) of object;
```

C++

```
(ASender: TObject; lCookie: Integer) ( TShellWindowsWindowRevoked)();
```

File

SHDocVw

Description

This is type SHDocVw.TShellWindowsWindowRevoked.

132.3.51 SHDocVw.TWebBrowser_V1BeforeNavigate Type

Pascal

```
TWebBrowser_V1BeforeNavigate = procedure (ASender: TObject; const URL: WideString; Flags:
Integer; const TargetFrameName: WideString; var PostData: OleVariant; const Headers:
WideString; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; const URL: WideString; Flags: Integer; const TargetFrameName: WideString;
var PostData: OleVariant; const Headers: WideString; var Cancel: WordBool) (
TWebBrowser_V1BeforeNavigate)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1BeforeNavigate.

132.3.52 SHDocVw.TWebBrowser_V1CommandStateChange Type

Pascal

```
TWebBrowser_V1CommandStateChange = procedure (ASender: TObject; Command: Integer; Enable:
WordBool) of object;
```

C++

```
(ASender: TObject; Command: Integer; Enable: WordBool) ( TWebBrowser_V1CommandStateChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1CommandStateChange.

132.3.53 SHDocVw.TWebBrowser_V1FrameBeforeNavigate Type

Pascal

```
TWebBrowser_V1FrameBeforeNavigate = procedure (ASender: TObject; const URL: WideString; Flags:
Integer; const TargetFrameName: WideString; var PostData: OleVariant; const Headers:
WideString; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; const URL: WideString; Flags: Integer; const TargetFrameName: WideString;
var PostData: OleVariant; const Headers: WideString; var Cancel: WordBool) (
TWebBrowser_V1FrameBeforeNavigate)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1FrameBeforeNavigate.

132.3.54 SHDocVw.TWebBrowser_V1FrameNavigateComplete Type

Pascal

```
TWebBrowser_V1FrameNavigateComplete = procedure (ASender: TObject; const URL: WideString) of
object;
```

C++

```
(ASender: TObject; const URL: WideString) ( TWebBrowser_V1FrameNavigateComplete)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1FrameNavigateComplete.

132.3.55 SHDocVw.TWebBrowser_V1FrameNewWindow Type

Pascal

```
TWebBrowser_V1FrameNewWindow = procedure (ASender: TObject; const URL: WideString; Flags:
Integer; const TargetFrameName: WideString; var PostData: OleVariant; const Headers:
WideString; var Processed: WordBool) of object;
```

C++

```
(ASender: TObject; const URL: WideString; Flags: Integer; const TargetFrameName: WideString;
var PostData: OleVariant; const Headers: WideString; var Processed: WordBool) (
TWebBrowser_V1FrameNewWindow)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1FrameNewWindow.

132.3.56 SHDocVw.TWebBrowser_V1NavigateComplete Type

Pascal

```
TWebBrowser_V1NavigateComplete = procedure (ASender: TObject; const URL: WideString) of object;
```

C++

```
(ASender: TObject; const URL: WideString) ( TWebBrowser_V1NavigateComplete)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1NavigateComplete.

132.3.57 SHDocVw.TWebBrowser_V1NewWindow Type

Pascal

```
TWebBrowser_V1NewWindow = procedure (ASender: TObject; const URL: WideString; Flags: Integer;
```

```
const TargetFrameName: WideString; var PostData: OleVariant; const Headers: WideString; var
Processed: WordBool) of object;
```

C++

```
(ASender: TObject; const URL: WideString; Flags: Integer; const TargetFrameName: WideString;
var PostData: OleVariant; const Headers: WideString; var Processed: WordBool) (
TWebBrowser_V1NewWindow)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1NewWindow.

132.3.58 SHDocVw.TWebBrowser_V1ProgressChange Type

Pascal

```
TWebBrowser_V1ProgressChange = procedure (ASender: TObject; Progress: Integer; ProgressMax:
Integer) of object;
```

C++

```
(ASender: TObject; Progress: Integer; ProgressMax: Integer) ( TWebBrowser_V1ProgressChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1ProgressChange.

132.3.59 SHDocVw.TWebBrowser_V1PropertyChange Type

Pascal

```
TWebBrowser_V1PropertyChange = procedure (ASender: TObject; const Property_: WideString) of
object;
```

C++

```
(ASender: TObject; const Property_: WideString) ( TWebBrowser_V1PropertyChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1PropertyChange.

132.3.60 SHDocVw.TWebBrowser_V1Quit Type

Pascal

```
TWebBrowser_V1Quit = procedure (ASender: TObject; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; var Cancel: WordBool) ( TWebBrowser_V1Quit)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1Quit.

132.3.61 SHDocVw.TWebBrowser_V1StatusTextChange Type

Pascal

```
TWebBrowser_V1StatusTextChange = procedure (ASender: TObject; const Text: WideString) of
object;
```

C++

```
(ASender: TObject; const Text: WideString) ( TWebBrowser_V1StatusTextChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1StatusTextChange.

132.3.62 SHDocVw.TWebBrowser_V1TitleChange Type

Pascal

```
TWebBrowser_V1TitleChange = procedure (ASender: TObject; const Text: WideString) of object;
```

C++

```
(ASender: TObject; const Text: WideString) ( TWebBrowser_V1TitleChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1TitleChange.

132.3.63 SHDocVw.TWebBrowserBeforeNavigate2 Type

Pascal

```
TWebBrowserBeforeNavigate2 = procedure (ASender: TObject; const pDisp: IDispatch; var URL:
OleVariant; var Flags: OleVariant; var TargetFrameName: OleVariant; var PostData: OleVariant;
var Headers: OleVariant; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant; var Flags: OleVariant; var
TargetFrameName: OleVariant; var PostData: OleVariant; var Headers: OleVariant; var Cancel:
WordBool) ( TWebBrowserBeforeNavigate2)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserBeforeNavigate2.

132.3.64 SHDocVw.TWebBrowserClientToHostWindow Type

Pascal

```
TWebBrowserClientToHostWindow = procedure (ASender: TObject; var CX: Integer; var CY: Integer)
of object;
```

C++

```
(ASender: TObject; var CX: Integer; var CY: Integer) ( TWebBrowserClientToHostWindow)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserClientToHostWindow.

132.3.65 SHDocVw.TWebBrowserCommandStateChange Type

Pascal

```
TWebBrowserCommandStateChange = procedure (ASender: TObject; Command: Integer; Enable:
WordBool) of object;
```

C++

```
(ASender: TObject; Command: Integer; Enable: WordBool) ( TWebBrowserCommandStateChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserCommandStateChange.

132.3.66 SHDocVw.TWebBrowserDocumentComplete Type

Pascal

```
TWebBrowserDocumentComplete = procedure (ASender: TObject; const pDisp: IDispatch; var URL:
OleVariant) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant) (
TWebBrowserDocumentComplete)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserDocumentComplete.

132.3.67 SHDocVw.TWebBrowserFileDownload Type

Pascal

```
TWebBrowserFileDownload = procedure (ASender: TObject; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; var Cancel: WordBool) ( TWebBrowserFileDownload)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserFileDownload.

132.3.68 SHDocVw.TWebBrowserNavigateComplete2 Type

Pascal

```
TWebBrowserNavigateComplete2 = procedure (ASender: TObject; const pDisp: IDispatch; var URL:
OleVariant) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant) (
TWebBrowserNavigateComplete2)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserNavigateComplete2.

132.3.69 SHDocVw.TWebBrowserNavigateError Type

Pascal

```
TWebBrowserNavigateError = procedure (ASender: TObject; const pDisp: IDispatch; var URL:
OleVariant; var Frame: OleVariant; var StatusCode: OleVariant; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant; var Frame: OleVariant; var
StatusCode: OleVariant; var Cancel: WordBool) ( TWebBrowserNavigateError)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserNavigateError.

132.3.70 SHDocVw.TWebBrowserNewWindow2 Type

Pascal

```
TWebBrowserNewWindow2 = procedure (ASender: TObject; var ppDisp: IDispatch; var Cancel:
WordBool) of object;
```

C++

```
(ASender: TObject; var ppDisp: IDispatch; var Cancel: WordBool) ( TWebBrowserNewWindow2)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserNewWindow2.

132.3.71 SHDocVw.TWebBrowserOnFullScreen Type

Pascal

```
TWebBrowserOnFullScreen = procedure (ASender: TObject; FullScreen: WordBool) of object;
```

C++

```
(ASender: TObject; FullScreen: WordBool) ( TWebBrowserOnFullScreen)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserOnFullScreen.

132.3.72 SHDocVw.TWebBrowserOnMenuBar Type

Pascal

```
TWebBrowserOnMenuBar = procedure (ASender: TObject; MenuBar: WordBool) of object;
```

C++

```
(ASender: TObject; MenuBar: WordBool) ( TWebBrowserOnMenuBar)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserOnMenuBar.

132.3.73 SHDocVw.TWebBrowserOnStatusBar Type

Pascal

```
TWebBrowserOnStatusBar = procedure (ASender: TObject; StatusBar: WordBool) of object;
```

C++

```
(ASender: TObject; StatusBar: WordBool) ( TWebBrowserOnStatusBar)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserOnStatusBar.

132.3.74 SHDocVw.TWebBrowserOnTheaterMode Type

Pascal

```
TWebBrowserOnTheaterMode = procedure (ASender: TObject; TheaterMode: WordBool) of object;
```

C++

```
(ASender: TObject; TheaterMode: WordBool) ( TWebBrowserOnTheaterMode)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserOnTheaterMode.

132.3.75 SHDocVw.TWebBrowserOnToolBar Type

Pascal

```
TWebBrowserOnToolBar = procedure (ASender: TObject; ToolBar: WordBool) of object;
```

C++

```
(ASender: TObject; ToolBar: WordBool) ( TWebBrowserOnToolBar)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserOnToolBar.

132.3.76 SHDocVw.TWebBrowserOnVisible Type

Pascal

```
TWebBrowserOnVisible = procedure (ASender: TObject; Visible: WordBool) of object;
```

C++

```
(ASender: TObject; Visible: WordBool) ( TWebBrowserOnVisible)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserOnVisible.

132.3.77 SHDocVw.TWebBrowserPrintTemplateInstantiation Type

Pascal

```
TWebBrowserPrintTemplateInstantiation = procedure (ASender: TObject; const pDisp: IDispatch)
of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch) ( TWebBrowserPrintTemplateInstantiation)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserPrintTemplateInstantiation.

132.3.78 SHDocVw.TWebBrowserPrintTemplateTeardown Type

Pascal

```
TWebBrowserPrintTemplateTeardown = procedure (ASender: TObject; const pDisp: IDispatch) of
object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch) ( TWebBrowserPrintTemplateTeardown)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserPrintTemplateTeardown.

132.3.79 SHDocVw.TWebBrowserPrivacyImpactedStateChange Type

Pascal

```
TWebBrowserPrivacyImpactedStateChange = procedure (ASender: TObject; bImpacted: WordBool) of
object;
```

C++

```
(ASender: TObject; bImpacted: WordBool) ( TWebBrowserPrivacyImpactedStateChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserPrivacyImpactedStateChange.

132.3.80 SHDocVw.TWebBrowserProgressChange Type

Pascal

```
TWebBrowserProgressChange = procedure (ASender: TObject; Progress: Integer; ProgressMax:
Integer) of object;
```

C++

```
(ASender: TObject; Progress: Integer; ProgressMax: Integer) ( TWebBrowserProgressChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserProgressChange.

132.3.81 SHDocVw.TWebBrowserPropertyChange Type

Pascal

```
TWebBrowserPropertyChange = procedure (ASender: TObject; const szProperty: WideString) of
object;
```

C++

```
(ASender: TObject; const szProperty: WideString) ( TWebBrowserPropertyChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserPropertyChange.

132.3.82 SHDocVw.TWebBrowserSetSecureLockIcon Type

Pascal

```
TWebBrowserSetSecureLockIcon = procedure (ASender: TObject; SecureLockIcon: Integer) of object;
```

C++

```
(ASender: TObject; SecureLockIcon: Integer) ( TWebBrowserSetSecureLockIcon)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserSetSecureLockIcon.

132.3.83 SHDocVw.TWebBrowserStatusTextChange Type

Pascal

```
TWebBrowserStatusTextChange = procedure (ASender: TObject; const Text: WideString) of object;
```

C++

```
(ASender: TObject; const Text: WideString) ( TWebBrowserStatusTextChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserStatusTextChange.

132.3.84 SHDocVw.TWebBrowserTitleChange Type

Pascal

```
TWebBrowserTitleChange = procedure (ASender: TObject; const Text: WideString) of object;
```

C++

```
(ASender: TObject; const Text: WideString) ( TWebBrowserTitleChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserTitleChange.

132.3.85 SHDocVw.TWebBrowserUpdatePageStatus Type

Pascal

```
TWebBrowserUpdatePageStatus = procedure (ASender: TObject; const pDisp: IDispatch; var nPage:
OleVariant; var fDone: OleVariant) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var nPage: OleVariant; var fDone: OleVariant) (
TWebBrowserUpdatePageStatus)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserUpdatePageStatus.

132.3.86 SHDocVw.TWebBrowserWindowClosing Type

Pascal

```
TWebBrowserWindowClosing = procedure (ASender: TObject; IsChildWindow: WordBool; var Cancel:
WordBool) of object;
```

C++

```
(ASender: TObject; IsChildWindow: WordBool; var Cancel: WordBool) (
TWebBrowserWindowClosing)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserWindowClosing.

132.3.87 SHDocVw.TWebBrowserWindowSetHeight Type

Pascal

```
TWebBrowserWindowSetHeight = procedure (ASender: TObject; Height: Integer) of object;
```

C++

```
(ASender: TObject; Height: Integer) ( TWebBrowserWindowSetHeight)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserWindowSetHeight.

132.3.88 SHDocVw.TWebBrowserWindowSetLeft Type

Pascal

```
TWebBrowserWindowSetLeft = procedure (ASender: TObject; Left: Integer) of object;
```

C++

```
(ASender: TObject; Left: Integer) ( TWebBrowserWindowSetLeft)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserWindowSetLeft.

132.3.89 SHDocVw.TWebBrowserWindowSetResizable Type

Pascal

```
TWebBrowserWindowSetResizable = procedure (ASender: TObject; Resizable: WordBool) of object;
```

C++

```
(ASender: TObject; Resizable: WordBool) ( TWebBrowserWindowSetResizable)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserWindowSetResizable.

132.3.90 SHDocVw.TWebBrowserWindowSetTop Type

Pascal

```
TWebBrowserWindowSetTop = procedure (ASender: TObject; Top: Integer) of object;
```

C++

```
(ASender: TObject; Top: Integer) ( TWebBrowserWindowSetTop)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserWindowSetTop.

132.3.91 SHDocVw.TWebBrowserWindowSetWidth Type

Pascal

```
TWebBrowserWindowSetWidth = procedure (ASender: TObject; Width: Integer) of object;
```

C++

```
(ASender: TObject; Width: Integer) ( TWebBrowserWindowSetWidth)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserWindowSetWidth.

132.3.92 SHDocVw.WebBrowser Type

Pascal

```
WebBrowser = IWebBrowser2;
```

C++

```
IWebBrowser2 WebBrowser;
```

File

SHDocVw

Description

This is type SHDocVw.WebBrowser.

132.3.93 SHDocVw.WebBrowser_V1 Type

Pascal

```
WebBrowser_V1 = IWebBrowser;
```

C++

```
IWebBrowser WebBrowser_V1;
```

File

SHDocVw

Description

This is type SHDocVw.WebBrowser_V1.

132.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CGID_IWebBrowser (see page 19488)	This is constant SHDocVw.CGID_IWebBrowser.
CLASS_CScriptErrorList (see page 19489)	This is constant SHDocVw.CLASS_CScriptErrorList.
CLASS_InternetExplorer (see page 19489)	This is constant SHDocVw.CLASS_InternetExplorer.
CLASS_SearchAssistantOC (see page 19489)	This is constant SHDocVw.CLASS_SearchAssistantOC.
CLASS_ShellBrowserWindow (see page 19490)	This is constant SHDocVw.CLASS_ShellBrowserWindow.
CLASS_ShellNameSpace (see page 19490)	This is constant SHDocVw.CLASS_ShellNameSpace.
CLASS_ShellUIHelper (see page 19490)	This is constant SHDocVw.CLASS_ShellUIHelper.
CLASS_ShellWindows (see page 19491)	This is constant SHDocVw.CLASS_ShellWindows.
CLASS_WebBrowser (see page 19491)	This is constant SHDocVw.CLASS_WebBrowser.
CLASS_WebBrowser_V1 (see page 19491)	This is constant SHDocVw.CLASS_WebBrowser_V1.
CSC_NAVIGATEBACK (see page 19492)	This is constant SHDocVw.CSC_NAVIGATEBACK.
CSC_NAVIGATEFORWARD (see page 19492)	This is constant SHDocVw.CSC_NAVIGATEFORWARD.
CSC_UPDATECOMMANDS (see page 19492)	This is constant SHDocVw.CSC_UPDATECOMMANDS.
DIID__SearchAssistantEvents (see page 19493)	This is constant SHDocVw.DIID__SearchAssistantEvents.
DIID_DShellNameSpaceEvents (see page 19493)	This is constant SHDocVw.DIID_DShellNameSpaceEvents.
DIID_DShellWindowsEvents (see page 19493)	This is constant SHDocVw.DIID_DShellWindowsEvents.
DIID_DWebBrowserEvents (see page 19494)	This is constant SHDocVw.DIID_DWebBrowserEvents.
DIID_DWebBrowserEvents2 (see page 19494)	This is constant SHDocVw.DIID_DWebBrowserEvents2.
dtlOcxPage (see page 19494)	This is constant SHDocVw.dtlOcxPage.
dtlServerPage (see page 19495)	This is constant SHDocVw.dtlServerPage.
HTMLID_FIND (see page 19495)	This is constant SHDocVw.HTMLID_FIND.
HTMLID_OPTIONS (see page 19495)	This is constant SHDocVw.HTMLID_OPTIONS.
HTMLID_VIEWSOURCE (see page 19496)	This is constant SHDocVw.HTMLID_VIEWSOURCE.
IID_IScriptErrorList (see page 19496)	This is constant SHDocVw.IID_IScriptErrorList.
IID_ISearch (see page 19496)	This is constant SHDocVw.IID_ISearch.
IID_ISearchAssistantOC (see page 19497)	This is constant SHDocVw.IID_ISearchAssistantOC.
IID_ISearchAssistantOC2 (see page 19497)	This is constant SHDocVw.IID_ISearchAssistantOC2.
IID_ISearchAssistantOC3 (see page 19497)	This is constant SHDocVw.IID_ISearchAssistantOC3.
IID_ISearches (see page 19498)	This is constant SHDocVw.IID_ISearches.
IID_IShellFavoritesNameSpace (see page 19498)	This is constant SHDocVw.IID_IShellFavoritesNameSpace.
IID_IShellNameSpace (see page 19498)	This is constant SHDocVw.IID_IShellNameSpace.
IID_IShellUIHelper (see page 19499)	This is constant SHDocVw.IID_IShellUIHelper.
IID_IShellWindows (see page 19499)	This is constant SHDocVw.IID_IShellWindows.
IID_IWebBrowser (see page 19499)	This is constant SHDocVw.IID_IWebBrowser.
IID_IWebBrowser2 (see page 19500)	This is constant SHDocVw.IID_IWebBrowser2.
IID_IWebBrowserApp (see page 19500)	This is constant SHDocVw.IID_IWebBrowserApp.
LIBID_SHDocVw (see page 19500)	This is constant SHDocVw.LIBID_SHDocVw.
navAllowAutosearch (see page 19501)	This is constant SHDocVw.navAllowAutosearch.
navBrowserBar (see page 19501)	This is constant SHDocVw.navBrowserBar.
navNoHistory (see page 19501)	This is constant SHDocVw.navNoHistory.
navNoReadFromCache (see page 19502)	This is constant SHDocVw.navNoReadFromCache.
navNoWriteToCache (see page 19502)	This is constant SHDocVw.navNoWriteToCache.
navOpenInNewWindow (see page 19502)	This is constant SHDocVw.navOpenInNewWindow.
OLECMDEXECOPT_DODEFAULT (see page 19503)	This is constant SHDocVw.OLECMDEXECOPT_DODEFAULT.

OLECMDEXECOPT_DONTPROMPTUSER (see page 19503)	This is constant SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER.
OLECMDEXECOPT_PROMPTUSER (see page 19503)	This is constant SHDocVw.OLECMDEXECOPT_PROMPTUSER.
OLECMDEXECOPT_SHOWHELP (see page 19504)	This is constant SHDocVw.OLECMDEXECOPT_SHOWHELP.
OLECMDF_DEFHIDEONCTXTMENU (see page 19504)	This is constant SHDocVw.OLECMDF_DEFHIDEONCTXTMENU.
OLECMDF_ENABLED (see page 19504)	This is constant SHDocVw.OLECMDF_ENABLED.
OLECMDF_INVISIBLE (see page 19505)	This is constant SHDocVw.OLECMDF_INVISIBLE.
OLECMDF_LATCHED (see page 19505)	This is constant SHDocVw.OLECMDF_LATCHED.
OLECMDF_NINCHED (see page 19505)	This is constant SHDocVw.OLECMDF_NINCHED.
OLECMDF_SUPPORTED (see page 19506)	This is constant SHDocVw.OLECMDF_SUPPORTED.
OLECMDID_ALLOWUILESSSAVEAS (see page 19506)	This is constant SHDocVw.OLECMDID_ALLOWUILESSSAVEAS.
OLECMDID_CLEARSELECTION (see page 19506)	This is constant SHDocVw.OLECMDID_CLEARSELECTION.
OLECMDID_CLOSE (see page 19507)	This is constant SHDocVw.OLECMDID_CLOSE.
OLECMDID_COPY (see page 19507)	This is constant SHDocVw.OLECMDID_COPY.
OLECMDID_CUT (see page 19507)	This is constant SHDocVw.OLECMDID_CUT.
OLECMDID_DELETE (see page 19508)	This is constant SHDocVw.OLECMDID_DELETE.
OLECMDID_DONTDOWNLOADCSS (see page 19508)	This is constant SHDocVw.OLECMDID_DONTDOWNLOADCSS.
OLECMDID_ENABLE_INTERACTION (see page 19508)	This is constant SHDocVw.OLECMDID_ENABLE_INTERACTION.
OLECMDID_FIND (see page 19509)	This is constant SHDocVw.OLECMDID_FIND.
OLECMDID_GETPRINTTEMPLATE (see page 19509)	This is constant SHDocVw.OLECMDID_GETPRINTTEMPLATE.
OLECMDID_GETZOOMRANGE (see page 19509)	This is constant SHDocVw.OLECMDID_GETZOOMRANGE.
OLECMDID_HIDETOOLBARS (see page 19510)	This is constant SHDocVw.OLECMDID_HIDETOOLBARS.
OLECMDID_HTTPEQUIV (see page 19510)	This is constant SHDocVw.OLECMDID_HTTPEQUIV.
OLECMDID_HTTPEQUIV_DONE (see page 19510)	This is constant SHDocVw.OLECMDID_HTTPEQUIV_DONE.
OLECMDID_NEW (see page 19511)	This is constant SHDocVw.OLECMDID_NEW.
OLECMDID_ONTOOLBARACTIVATED (see page 19511)	This is constant SHDocVw.OLECMDID_ONTOOLBARACTIVATED.
OLECMDID_ONUNLOAD (see page 19511)	This is constant SHDocVw.OLECMDID_ONUNLOAD.
OLECMDID_OPEN (see page 19512)	This is constant SHDocVw.OLECMDID_OPEN.
OLECMDID_PAGESETUP (see page 19512)	This is constant SHDocVw.OLECMDID_PAGESETUP.
OLECMDID_PASTE (see page 19512)	This is constant SHDocVw.OLECMDID_PASTE.
OLECMDID_PASTESPECIAL (see page 19513)	This is constant SHDocVw.OLECMDID_PASTESPECIAL.
OLECMDID_PREREFRESH (see page 19513)	This is constant SHDocVw.OLECMDID_PREREFRESH.
OLECMDID_PRINT (see page 19513)	This is constant SHDocVw.OLECMDID_PRINT.
OLECMDID_PRINT2 (see page 19514)	This is constant SHDocVw.OLECMDID_PRINT2.
OLECMDID_PRINTPREVIEW (see page 19514)	This is constant SHDocVw.OLECMDID_PRINTPREVIEW.
OLECMDID_PRINTPREVIEW2 (see page 19514)	This is constant SHDocVw.OLECMDID_PRINTPREVIEW2.
OLECMDID_PROPERTIES (see page 19515)	This is constant SHDocVw.OLECMDID_PROPERTIES.
OLECMDID_PROPERTYBAG2 (see page 19515)	This is constant SHDocVw.OLECMDID_PROPERTYBAG2.
OLECMDID_REDO (see page 19515)	This is constant SHDocVw.OLECMDID_REDO.
OLECMDID_REFRESH (see page 19516)	This is constant SHDocVw.OLECMDID_REFRESH.
OLECMDID_SAVE (see page 19516)	This is constant SHDocVw.OLECMDID_SAVE.
OLECMDID_SAVEAS (see page 19516)	This is constant SHDocVw.OLECMDID_SAVEAS.
OLECMDID_SAVECOPYAS (see page 19517)	This is constant SHDocVw.OLECMDID_SAVECOPYAS.
OLECMDID_SELECTALL (see page 19517)	This is constant SHDocVw.OLECMDID_SELECTALL.
OLECMDID_SETDOWNLOADSTATE (see page 19517)	This is constant SHDocVw.OLECMDID_SETDOWNLOADSTATE.
OLECMDID_SETPRINTTEMPLATE (see page 19518)	This is constant SHDocVw.OLECMDID_SETPRINTTEMPLATE.
OLECMDID_SETPROGRESSMAX (see page 19518)	This is constant SHDocVw.OLECMDID_SETPROGRESSMAX.
OLECMDID_SETPROGRESSPOS (see page 19518)	This is constant SHDocVw.OLECMDID_SETPROGRESSPOS.
OLECMDID_SETPROGRESSTEXT (see page 19519)	This is constant SHDocVw.OLECMDID_SETPROGRESSTEXT.
OLECMDID_SETTITLE (see page 19519)	This is constant SHDocVw.OLECMDID_SETTITLE.
OLECMDID_SHOWFIND (see page 19519)	This is constant SHDocVw.OLECMDID_SHOWFIND.
OLECMDID_SHOWMESSAGE (see page 19520)	This is constant SHDocVw.OLECMDID_SHOWMESSAGE.
OLECMDID_SHOWPAGESETUP (see page 19520)	This is constant SHDocVw.OLECMDID_SHOWPAGESETUP.
OLECMDID_SHOWPRINT (see page 19520)	This is constant SHDocVw.OLECMDID_SHOWPRINT.
OLECMDID_SHOWSCRIPTERROR (see page 19521)	This is constant SHDocVw.OLECMDID_SHOWSCRIPTERROR.

OLECMDID_SPELL (see page 19521)	This is constant SHDocVw.OLECMDID_SPELL.
OLECMDID_STOP (see page 19521)	This is constant SHDocVw.OLECMDID_STOP.
OLECMDID_STOPDOWNLOAD (see page 19522)	This is constant SHDocVw.OLECMDID_STOPDOWNLOAD.
OLECMDID_UNDO (see page 19522)	This is constant SHDocVw.OLECMDID_UNDO.
OLECMDID_UPDATECOMMANDS (see page 19522)	This is constant SHDocVw.OLECMDID_UPDATECOMMANDS.
OLECMDID_UPDATEPAGESTATUS (see page 19523)	This is constant SHDocVw.OLECMDID_UPDATEPAGESTATUS.
OLECMDID_ZOOM (see page 19523)	This is constant SHDocVw.OLECMDID_ZOOM.
READYSTATE_COMPLETE (see page 19523)	This is constant SHDocVw.READYSTATE_COMPLETE.
READYSTATE_INTERACTIVE (see page 19524)	This is constant SHDocVw.READYSTATE_INTERACTIVE.
READYSTATE_LOADED (see page 19524)	This is constant SHDocVw.READYSTATE_LOADED.
READYSTATE_LOADING (see page 19524)	This is constant SHDocVw.READYSTATE_LOADING.
READYSTATE_UNINITIALIZED (see page 19525)	This is constant SHDocVw.READYSTATE_UNINITIALIZED.
REFRESH_COMPLETELY (see page 19525)	This is constant SHDocVw.REFRESH_COMPLETELY.
REFRESH_CONTINUE (see page 19525)	This is constant SHDocVw.REFRESH_CONTINUE.
REFRESH_IFEXPIRED (see page 19526)	This is constant SHDocVw.REFRESH_IFEXPIRED.
REFRESH_NORMAL (see page 19526)	This is constant SHDocVw.REFRESH_NORMAL.
secureLockIconMixed (see page 19526)	This is constant SHDocVw.secureLockIconMixed.
secureLockIconSecure128Bit (see page 19527)	This is constant SHDocVw.secureLockIconSecure128Bit.
secureLockIconSecure40Bit (see page 19527)	This is constant SHDocVw.secureLockIconSecure40Bit.
secureLockIconSecure56Bit (see page 19527)	This is constant SHDocVw.secureLockIconSecure56Bit.
secureLockIconSecureFortezza (see page 19528)	This is constant SHDocVw.secureLockIconSecureFortezza.
secureLockIconSecureUnknownBits (see page 19528)	This is constant SHDocVw.secureLockIconSecureUnknownBits.
secureLockIconUnsecure (see page 19528)	This is constant SHDocVw.secureLockIconUnsecure.
SHDocVwMajorVersion (see page 19529)	This is constant SHDocVw.SHDocVwMajorVersion.
SHDocVwMinorVersion (see page 19529)	This is constant SHDocVw.SHDocVwMinorVersion.
SWC_3RDPARTY (see page 19529)	This is constant SHDocVw.SWC_3RDPARTY.
SWC_BROWSER (see page 19530)	This is constant SHDocVw.SWC_BROWSER.
SWC_CALLBACK (see page 19530)	This is constant SHDocVw.SWC_CALLBACK.
SWC_EXPLORER (see page 19530)	This is constant SHDocVw.SWC_EXPLORER.
SWFO_COOKIEPASSED (see page 19531)	This is constant SHDocVw.SWFO_COOKIEPASSED.
SWFO_INCLUDEPENDING (see page 19531)	This is constant SHDocVw.SWFO_INCLUDEPENDING.
SWFO_NEEDDISPATCH (see page 19531)	This is constant SHDocVw.SWFO_NEEDDISPATCH.

132.4.1 SHDocVw.CGID_IWebBrowser Constant

Pascal

```
CGID_IWebBrowser: TGUID = '{ED016940-BD5B-11CF-BA4E-00C04FD70816}';
```

C++

```
TGUID CGID_IWebBrowser = '{ED016940-BD5B-11CF-BA4E-00C04FD70816}';
```

File

SHDocVw

Description

This is constant SHDocVw.CGID_IWebBrowser.

132.4.2 SHDocVw.CLASS_CScriptErrorList Constant

Pascal

```
CLASS_CScriptErrorList: TGUID = '{EFD01300-160F-11D2-BB2E-00805FF7EFCA}';
```

C++

```
TGUID CLASS_CScriptErrorList = '{EFD01300-160F-11D2-BB2E-00805FF7EFCA}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_CScriptErrorList.

132.4.3 SHDocVw.CLASS_InternetExplorer Constant

Pascal

```
CLASS_InternetExplorer: TGUID = '{0002DF01-0000-0000-C000-000000000046}';
```

C++

```
TGUID CLASS_InternetExplorer = '{0002DF01-0000-0000-C000-000000000046}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_InternetExplorer.

132.4.4 SHDocVw.CLASS_SearchAssistantOC Constant

Pascal

```
CLASS_SearchAssistantOC: TGUID = '{B45FF030-4447-11D2-85DE-00C04FA35C89}';
```

C++

```
TGUID CLASS_SearchAssistantOC = '{B45FF030-4447-11D2-85DE-00C04FA35C89}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_SearchAssistantOC.

132.4.5 SHDocVw.CLASS_ShellBrowserWindow Constant

Pascal

```
CLASS_ShellBrowserWindow: TGUID = '{C08AFD90-F2A1-11D1-8455-00A0C91F3880}';
```

C++

```
TGUID CLASS_ShellBrowserWindow = '{C08AFD90-F2A1-11D1-8455-00A0C91F3880}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_ShellBrowserWindow.

132.4.6 SHDocVw.CLASS_ShellNameSpace Constant

Pascal

```
CLASS_ShellNameSpace: TGUID = '{55136805-B2DE-11D1-B9F2-00A0C98BC547}';
```

C++

```
TGUID CLASS_ShellNameSpace = '{55136805-B2DE-11D1-B9F2-00A0C98BC547}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_ShellNameSpace.

132.4.7 SHDocVw.CLASS_ShellUIHelper Constant

Pascal

```
CLASS_ShellUIHelper: TGUID = '{64AB4BB7-111E-11D1-8F79-00C04FC2FBE1}';
```

C++

```
TGUID CLASS_ShellUIHelper = '{64AB4BB7-111E-11D1-8F79-00C04FC2FBE1}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_ShellUIHelper.

132.4.8 SHDocVw.CLASS_ShellWindows Constant

Pascal

```
CLASS_ShellWindows: TGUID = '{9BA05972-F6A8-11CF-A442-00A0C90A8F39}';
```

C++

```
TGUID CLASS_ShellWindows = '{9BA05972-F6A8-11CF-A442-00A0C90A8F39}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_ShellWindows.

132.4.9 SHDocVw.CLASS_WebBrowser Constant

Pascal

```
CLASS_WebBrowser: TGUID = '{8856F961-340A-11D0-A96B-00C04FD705A2}';
```

C++

```
TGUID CLASS_WebBrowser = '{8856F961-340A-11D0-A96B-00C04FD705A2}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_WebBrowser.

132.4.10 SHDocVw.CLASS_WebBrowser_V1 Constant

Pascal

```
CLASS_WebBrowser_V1: TGUID = '{EAB22AC3-30C1-11CF-A7EB-0000C05BAE0B}';
```

C++

```
TGUID CLASS_WebBrowser_V1 = '{EAB22AC3-30C1-11CF-A7EB-0000C05BAE0B}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_WebBrowser_V1.

132.4.11 SHDocVw.CSC_NAVIGATEBACK Constant

Pascal

```
CSC_NAVIGATEBACK = $00000002;
```

C++

```
CSC_NAVIGATEBACK = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.CSC_NAVIGATEBACK.

132.4.12 SHDocVw.CSC_NAVIGATEFORWARD Constant

Pascal

```
CSC_NAVIGATEFORWARD = $00000001;
```

C++

```
CSC_NAVIGATEFORWARD = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.CSC_NAVIGATEFORWARD.

132.4.13 SHDocVw.CSC_UPDATECOMMANDS Constant

Pascal

```
CSC_UPDATECOMMANDS = $FFFFFFFF;
```

C++

```
CSC_UPDATECOMMANDS = 0xFFFFFFFF;
```

File

SHDocVw

Description

This is constant SHDocVw.CSC_UPDATECOMMANDS.

132.4.14 SHDocVw.DIID__SearchAssistantEvents Constant

Pascal

```
DIID__SearchAssistantEvents: TGUID = '{1611FDDA-445B-11D2-85DE-00C04FA35C89}';
```

C++

```
TGUID DIID__SearchAssistantEvents = '{1611FDDA-445B-11D2-85DE-00C04FA35C89}';
```

File

SHDocVw

Description

This is constant SHDocVw.DIID__SearchAssistantEvents.

132.4.15 SHDocVw.DIID_DShellNameSpaceEvents Constant

Pascal

```
DIID_DShellNameSpaceEvents: TGUID = '{55136806-B2DE-11D1-B9F2-00A0C98BC547}';
```

C++

```
TGUID DIID_DShellNameSpaceEvents = '{55136806-B2DE-11D1-B9F2-00A0C98BC547}';
```

File

SHDocVw

Description

This is constant SHDocVw.DIID_DShellNameSpaceEvents.

132.4.16 SHDocVw.DIID_DShellWindowsEvents Constant

Pascal

```
DIID_DShellWindowsEvents: TGUID = '{FE4106E0-399A-11D0-A48C-00A0C90A8F39}';
```

C++

```
TGUID DIID_DShellWindowsEvents = '{FE4106E0-399A-11D0-A48C-00A0C90A8F39}';
```

File

SHDocVw

Description

This is constant SHDocVw.DIID_DShellWindowsEvents.

132.4.17 SHDocVw.DIID_DWebBrowserEvents Constant

Pascal

```
DIID_DWebBrowserEvents: TGUID = '{EAB22AC2-30C1-11CF-A7EB-0000C05BAE0B}';
```

C++

```
TGUID DIID_DWebBrowserEvents = '{EAB22AC2-30C1-11CF-A7EB-0000C05BAE0B}';
```

File

SHDocVw

Description

This is constant SHDocVw.DIID_DWebBrowserEvents.

132.4.18 SHDocVw.DIID_DWebBrowserEvents2 Constant

Pascal

```
DIID_DWebBrowserEvents2: TGUID = '{34A715A0-6587-11D0-924A-0020AFC7AC4D}';
```

C++

```
TGUID DIID_DWebBrowserEvents2 = '{34A715A0-6587-11D0-924A-0020AFC7AC4D}';
```

File

SHDocVw

Description

This is constant SHDocVw.DIID_DWebBrowserEvents2.

132.4.19 SHDocVw.dtlOcxPage Constant

Pascal

```
dtlOcxPage = 'ActiveX';
```

C++

```
dtlOcxPage = 'ActiveX';
```

File

SHDocVw

Description

This is constant SHDocVw.dtlOcxPage.

132.4.20 SHDocVw.dtlServerPage Constant

Pascal

```
dtlServerPage = 'Servers';
```

C++

```
dtlServerPage = 'Servers';
```

File

SHDocVw

Description

This is constant SHDocVw.dtlServerPage.

132.4.21 SHDocVw.HTMLID_FIND Constant

Pascal

```
HTMLID_FIND = 1;
```

C++

```
HTMLID_FIND = 1;
```

File

SHDocVw

Description

This is constant SHDocVw.HTMLID_FIND.

132.4.22 SHDocVw.HTMLID_OPTIONS Constant

Pascal

```
HTMLID_OPTIONS = 3;
```

C++

```
HTMLID_OPTIONS = 3;
```

File

SHDocVw

Description

This is constant SHDocVw.HTMLID_OPTIONS.

132.4.23 SHDocVw.HTMLID_VIEWSOURCE Constant

Pascal

```
HTMLID_VIEWSOURCE = 2;
```

C++

```
HTMLID_VIEWSOURCE = 2;
```

File

SHDocVw

Description

This is constant SHDocVw.HTMLID_VIEWSOURCE.

132.4.24 SHDocVw.IID_IScriptErrorList Constant

Pascal

```
IID_IScriptErrorList: TGUID = '{F3470F24-15FD-11D2-BB2E-00805FF7EFCA}';
```

C++

```
TGUID IID_IScriptErrorList = '{F3470F24-15FD-11D2-BB2E-00805FF7EFCA}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IScriptErrorList.

132.4.25 SHDocVw.IID_ISearch Constant

Pascal

```
IID_ISearch: TGUID = '{BA9239A4-3DD5-11D2-BF8B-00C04FB93661}';
```

C++

```
TGUID IID_ISearch = '{BA9239A4-3DD5-11D2-BF8B-00C04FB93661}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_ISearch.

132.4.26 SHDocVw.IID_ISearchAssistantOC Constant

Pascal

```
IID_ISearchAssistantOC: TGUID = '{72423E8F-8011-11D2-BE79-00A0C9A83DA1}';
```

C++

```
TGUID IID_ISearchAssistantOC = '{72423E8F-8011-11D2-BE79-00A0C9A83DA1}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_ISearchAssistantOC.

132.4.27 SHDocVw.IID_ISearchAssistantOC2 Constant

Pascal

```
IID_ISearchAssistantOC2: TGUID = '{72423E8F-8011-11D2-BE79-00A0C9A83DA2}';
```

C++

```
TGUID IID_ISearchAssistantOC2 = '{72423E8F-8011-11D2-BE79-00A0C9A83DA2}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_ISearchAssistantOC2.

132.4.28 SHDocVw.IID_ISearchAssistantOC3 Constant

Pascal

```
IID_ISearchAssistantOC3: TGUID = '{72423E8F-8011-11D2-BE79-00A0C9A83DA3}';
```

C++

```
TGUID IID_ISearchAssistantOC3 = '{72423E8F-8011-11D2-BE79-00A0C9A83DA3}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_ISearchAssistantOC3.

132.4.29 SHDocVw.IID_ISearches Constant

Pascal

```
IID_ISearches: TGUID = '{47C922A2-3DD5-11D2-BF8B-00C04FB93661}';
```

C++

```
TGUID IID_ISearches = '{47C922A2-3DD5-11D2-BF8B-00C04FB93661}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_ISearches.

132.4.30 SHDocVw.IID_IShellFavoritesNameSpace Constant

Pascal

```
IID_IShellFavoritesNameSpace: TGUID = '{55136804-B2DE-11D1-B9F2-00A0C98BC547}';
```

C++

```
TGUID IID_IShellFavoritesNameSpace = '{55136804-B2DE-11D1-B9F2-00A0C98BC547}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IShellFavoritesNameSpace.

132.4.31 SHDocVw.IID_IShellNameSpace Constant

Pascal

```
IID_IShellNameSpace: TGUID = '{E572D3C9-37BE-4AE2-825D-D521763E3108}';
```

C++

```
TGUID IID_IShellNameSpace = '{E572D3C9-37BE-4AE2-825D-D521763E3108}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IShellNameSpace.

132.4.32 SHDocVw.IID_IShellUIHelper Constant

Pascal

```
IID_IShellUIHelper: TGUID = '{729FE2F8-1EA8-11D1-8F85-00C04FC2FBE1}';
```

C++

```
TGUID IID_IShellUIHelper = '{729FE2F8-1EA8-11D1-8F85-00C04FC2FBE1}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IShellUIHelper.

132.4.33 SHDocVw.IID_IShellWindows Constant

Pascal

```
IID_IShellWindows: TGUID = '{85CB6900-4D95-11CF-960C-0080C7F4EE85}';
```

C++

```
TGUID IID_IShellWindows = '{85CB6900-4D95-11CF-960C-0080C7F4EE85}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IShellWindows.

132.4.34 SHDocVw.IID_IWebBrowser Constant

Pascal

```
IID_IWebBrowser: TGUID = '{EAB22AC1-30C1-11CF-A7EB-0000C05BAE0B}';
```

C++

```
TGUID IID_IWebBrowser = '{EAB22AC1-30C1-11CF-A7EB-0000C05BAE0B}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IWebBrowser.

132.4.35 SHDocVw.IID_IWebBrowser2 Constant

Pascal

```
IID_IWebBrowser2: TGUID = '{D30C1661-CDAF-11D0-8A3E-00C04FC9E26E}';
```

C++

```
TGUID IID_IWebBrowser2 = '{D30C1661-CDAF-11D0-8A3E-00C04FC9E26E}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IWebBrowser2.

132.4.36 SHDocVw.IID_IWebBrowserApp Constant

Pascal

```
IID_IWebBrowserApp: TGUID = '{0002DF05-0000-0000-C000-000000000046}';
```

C++

```
TGUID IID_IWebBrowserApp = '{0002DF05-0000-0000-C000-000000000046}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IWebBrowserApp.

132.4.37 SHDocVw.LIBID_SHDocVw Constant

Pascal

```
LIBID_SHDocVw: TGUID = '{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}';
```

C++

```
TGUID LIBID_SHDocVw = '{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}';
```

File

SHDocVw

Description

This is constant SHDocVw.LIBID_SHDocVw.

132.4.38 SHDocVw.navAllowAutosearch Constant

Pascal

```
navAllowAutosearch = $00000010;
```

C++

```
navAllowAutosearch = 0x00000010;
```

File

SHDocVw

Description

This is constant SHDocVw.navAllowAutosearch.

132.4.39 SHDocVw.navBrowserBar Constant

Pascal

```
navBrowserBar = $00000020;
```

C++

```
navBrowserBar = 0x00000020;
```

File

SHDocVw

Description

This is constant SHDocVw.navBrowserBar.

132.4.40 SHDocVw.navNoHistory Constant

Pascal

```
navNoHistory = $00000002;
```

C++

```
navNoHistory = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.navNoHistory.

132.4.41 SHDocVw.navNoReadFromCache Constant

Pascal

```
navNoReadFromCache = $00000004;
```

C++

```
navNoReadFromCache = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.navNoReadFromCache.

132.4.42 SHDocVw.navNoWriteToCache Constant

Pascal

```
navNoWriteToCache = $00000008;
```

C++

```
navNoWriteToCache = 0x00000008;
```

File

SHDocVw

Description

This is constant SHDocVw.navNoWriteToCache.

132.4.43 SHDocVw.navOpenInNewWindow Constant

Pascal

```
navOpenInNewWindow = $00000001;
```

C++

```
navOpenInNewWindow = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.navOpenInNewWindow.

132.4.44 SHDocVw.OLECMDEXECOPT_DODEFAULT Constant

Pascal

```
OLECMDEXECOPT_DODEFAULT = $00000000;
```

C++

```
OLECMDEXECOPT_DODEFAULT = 0x00000000;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDEXECOPT_DODEFAULT.

132.4.45 SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER Constant

Pascal

```
OLECMDEXECOPT_DONTPROMPTUSER = $00000002;
```

C++

```
OLECMDEXECOPT_DONTPROMPTUSER = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER.

132.4.46 SHDocVw.OLECMDEXECOPT_PROMPTUSER Constant

Pascal

```
OLECMDEXECOPT_PROMPTUSER = $00000001;
```

C++

```
OLECMDEXECOPT_PROMPTUSER = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDEXECOPT_PROMPTUSER.

132.4.47 SHDocVw.OLECMDEXECOPT_SHOWHELP Constant

Pascal

```
OLECMDEXECOPT_SHOWHELP = $00000003;
```

C++

```
OLECMDEXECOPT_SHOWHELP = 0x00000003;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDEXECOPT_SHOWHELP.

132.4.48 SHDocVw.OLECMDF_DEFHIDEONCTXTMENU Constant

Pascal

```
OLECMDF_DEFHIDEONCTXTMENU = $00000020;
```

C++

```
OLECMDF_DEFHIDEONCTXTMENU = 0x00000020;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDF_DEFHIDEONCTXTMENU.

132.4.49 SHDocVw.OLECMDF_ENABLED Constant

Pascal

```
OLECMDF_ENABLED = $00000002;
```

C++

```
OLECMDF_ENABLED = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDF_ENABLED.

132.4.50 SHDocVw.OLECMDF_INVISIBLE Constant

Pascal

```
OLECMDF_INVISIBLE = $00000010;
```

C++

```
OLECMDF_INVISIBLE = 0x00000010;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDF_INVISIBLE.

132.4.51 SHDocVw.OLECMDF_LATCHED Constant

Pascal

```
OLECMDF_LATCHED = $00000004;
```

C++

```
OLECMDF_LATCHED = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDF_LATCHED.

132.4.52 SHDocVw.OLECMDF_NINCHED Constant

Pascal

```
OLECMDF_NINCHED = $00000008;
```

C++

```
OLECMDF_NINCHED = 0x00000008;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDF_NINCHED.

132.4.53 SHDocVw.OLECMDF_SUPPORTED Constant

Pascal

```
OLECMDF_SUPPORTED = $00000001;
```

C++

```
OLECMDF_SUPPORTED = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDF_SUPPORTED.

132.4.54 SHDocVw.OLECMDID_ALLOWUILESSSAVEAS Constant

Pascal

```
OLECMDID_ALLOWUILESSSAVEAS = $0000002E;
```

C++

```
OLECMDID_ALLOWUILESSSAVEAS = 0x0000002E;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_ALLOWUILESSSAVEAS.

132.4.55 SHDocVw.OLECMDID_CLEARSELECTION Constant

Pascal

```
OLECMDID_CLEARSELECTION = $00000012;
```

C++

```
OLECMDID_CLEARSELECTION = 0x00000012;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_CLEARSELECTION.

132.4.56 SHDocVw.OLECMDID_CLOSE Constant

Pascal

```
OLECMDID_CLOSE = $0000002D;
```

C++

```
OLECMDID_CLOSE = 0x0000002D;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_CLOSE.

132.4.57 SHDocVw.OLECMDID_COPY Constant

Pascal

```
OLECMDID_COPY = $0000000C;
```

C++

```
OLECMDID_COPY = 0x0000000C;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_COPY.

132.4.58 SHDocVw.OLECMDID_CUT Constant

Pascal

```
OLECMDID_CUT = $0000000B;
```

C++

```
OLECMDID_CUT = 0x0000000B;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_CUT.

132.4.59 SHDocVw.OLECMDID_DELETE Constant

Pascal

```
OLECMDID_DELETE = $00000021;
```

C++

```
OLECMDID_DELETE = 0x00000021;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_DELETE.

132.4.60 SHDocVw.OLECMDID_DONTDOWNLOADCSS Constant

Pascal

```
OLECMDID_DONTDOWNLOADCSS = $0000002F;
```

C++

```
OLECMDID_DONTDOWNLOADCSS = 0x0000002F;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_DONTDOWNLOADCSS.

132.4.61 SHDocVw.OLECMDID_ENABLE_INTERACTION Constant

Pascal

```
OLECMDID_ENABLE_INTERACTION = $00000024;
```

C++

```
OLECMDID_ENABLE_INTERACTION = 0x00000024;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_ENABLE_INTERACTION.

132.4.62 SHDocVw.OLECMDID_FIND Constant

Pascal

```
OLECMDID_FIND = $00000020;
```

C++

```
OLECMDID_FIND = 0x00000020;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_FIND.

132.4.63 SHDocVw.OLECMDID_GETPRINTTEMPLATE Constant

Pascal

```
OLECMDID_GETPRINTTEMPLATE = $00000034;
```

C++

```
OLECMDID_GETPRINTTEMPLATE = 0x00000034;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_GETPRINTTEMPLATE.

132.4.64 SHDocVw.OLECMDID_GETZOOMRANGE Constant

Pascal

```
OLECMDID_GETZOOMRANGE = $00000014;
```

C++

```
OLECMDID_GETZOOMRANGE = 0x00000014;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_GETZOOMRANGE.

132.4.65 SHDocVw.OLECMDID_HIDETOOLBARS Constant

Pascal

```
OLECMDID_HIDETOOLBARS = $00000018;
```

C++

```
OLECMDID_HIDETOOLBARS = 0x00000018;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_HIDETOOLBARS.

132.4.66 SHDocVw.OLECMDID_HTTPEQUIV Constant

Pascal

```
OLECMDID_HTTPEQUIV = $00000022;
```

C++

```
OLECMDID_HTTPEQUIV = 0x00000022;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_HTTPEQUIV.

132.4.67 SHDocVw.OLECMDID_HTTPEQUIV_DONE Constant

Pascal

```
OLECMDID_HTTPEQUIV_DONE = $00000023;
```

C++

```
OLECMDID_HTTPEQUIV_DONE = 0x00000023;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_HTTPEQUIV_DONE.

132.4.68 SHDocVw.OLECMDID_NEW Constant

Pascal

```
OLECMDID_NEW = $00000002;
```

C++

```
OLECMDID_NEW = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_NEW.

132.4.69 SHDocVw.OLECMDID_ONTOOLBARACTIVATED Constant

Pascal

```
OLECMDID_ONTOOLBARACTIVATED = $0000001F;
```

C++

```
OLECMDID_ONTOOLBARACTIVATED = 0x0000001F;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_ONTOOLBARACTIVATED.

132.4.70 SHDocVw.OLECMDID_ONUNLOAD Constant

Pascal

```
OLECMDID_ONUNLOAD = $00000025;
```

C++

```
OLECMDID_ONUNLOAD = 0x00000025;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_ONUNLOAD.

132.4.71 SHDocVw.OLECMDID_OPEN Constant

Pascal

```
OLECMDID_OPEN = $00000001;
```

C++

```
OLECMDID_OPEN = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_OPEN.

132.4.72 SHDocVw.OLECMDID_PAGESETUP Constant

Pascal

```
OLECMDID_PAGESETUP = $00000008;
```

C++

```
OLECMDID_PAGESETUP = 0x00000008;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PAGESETUP.

132.4.73 SHDocVw.OLECMDID_PASTE Constant

Pascal

```
OLECMDID_PASTE = $0000000D;
```

C++

```
OLECMDID_PASTE = 0x0000000D;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PASTE.

132.4.74 SHDocVw.OLECMDID_PASTESPECIAL Constant

Pascal

```
OLECMDID_PASTESPECIAL = $0000000E;
```

C++

```
OLECMDID_PASTESPECIAL = 0x0000000E;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PASTESPECIAL.

132.4.75 SHDocVw.OLECMDID_PREREFRESH Constant

Pascal

```
OLECMDID_PREREFRESH = $00000027;
```

C++

```
OLECMDID_PREREFRESH = 0x00000027;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PREREFRESH.

132.4.76 SHDocVw.OLECMDID_PRINT Constant

Pascal

```
OLECMDID_PRINT = $00000006;
```

C++

```
OLECMDID_PRINT = 0x00000006;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PRINT.

132.4.77 SHDocVw.OLECMDID_PRINT2 Constant

Pascal

```
OLECMDID_PRINT2 = $00000031;
```

C++

```
OLECMDID_PRINT2 = 0x00000031;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PRINT2.

132.4.78 SHDocVw.OLECMDID_PRINTPREVIEW Constant

Pascal

```
OLECMDID_PRINTPREVIEW = $00000007;
```

C++

```
OLECMDID_PRINTPREVIEW = 0x00000007;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PRINTPREVIEW.

132.4.79 SHDocVw.OLECMDID_PRINTPREVIEW2 Constant

Pascal

```
OLECMDID_PRINTPREVIEW2 = $00000032;
```

C++

```
OLECMDID_PRINTPREVIEW2 = 0x00000032;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PRINTPREVIEW2.

132.4.80 SHDocVw.OLECMDID_PROPERTIES Constant

Pascal

```
OLECMDID_PROPERTIES = $0000000A;
```

C++

```
OLECMDID_PROPERTIES = 0x0000000A;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PROPERTIES.

132.4.81 SHDocVw.OLECMDID_PROPERTYBAG2 Constant

Pascal

```
OLECMDID_PROPERTYBAG2 = $00000026;
```

C++

```
OLECMDID_PROPERTYBAG2 = 0x00000026;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PROPERTYBAG2.

132.4.82 SHDocVw.OLECMDID_REDO Constant

Pascal

```
OLECMDID_REDO = $00000010;
```

C++

```
OLECMDID_REDO = 0x00000010;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_REDO.

132.4.83 SHDocVw.OLECMDID_REFRESH Constant

Pascal

```
OLECMDID_REFRESH = $00000016;
```

C++

```
OLECMDID_REFRESH = 0x00000016;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_REFRESH.

132.4.84 SHDocVw.OLECMDID_SAVE Constant

Pascal

```
OLECMDID_SAVE = $00000003;
```

C++

```
OLECMDID_SAVE = 0x00000003;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SAVE.

132.4.85 SHDocVw.OLECMDID_SAVEAS Constant

Pascal

```
OLECMDID_SAVEAS = $00000004;
```

C++

```
OLECMDID_SAVEAS = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SAVEAS.

132.4.86 SHDocVw.OLECMDID_SAVECOPYAS Constant

Pascal

```
OLECMDID_SAVECOPYAS = $00000005;
```

C++

```
OLECMDID_SAVECOPYAS = 0x00000005;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SAVECOPYAS.

132.4.87 SHDocVw.OLECMDID_SELECTALL Constant

Pascal

```
OLECMDID_SELECTALL = $00000011;
```

C++

```
OLECMDID_SELECTALL = 0x00000011;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SELECTALL.

132.4.88 SHDocVw.OLECMDID_SETDOWNLOADSTATE Constant

Pascal

```
OLECMDID_SETDOWNLOADSTATE = $0000001D;
```

C++

```
OLECMDID_SETDOWNLOADSTATE = 0x0000001D;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SETDOWNLOADSTATE.

132.4.89 SHDocVw.OLECMDID_SETPRINTTEMPLATE Constant

Pascal

```
OLECMDID_SETPRINTTEMPLATE = $00000033;
```

C++

```
OLECMDID_SETPRINTTEMPLATE = 0x00000033;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SETPRINTTEMPLATE.

132.4.90 SHDocVw.OLECMDID_SETPROGRESSMAX Constant

Pascal

```
OLECMDID_SETPROGRESSMAX = $00000019;
```

C++

```
OLECMDID_SETPROGRESSMAX = 0x00000019;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SETPROGRESSMAX.

132.4.91 SHDocVw.OLECMDID_SETPROGRESSPOS Constant

Pascal

```
OLECMDID_SETPROGRESSPOS = $0000001A;
```

C++

```
OLECMDID_SETPROGRESSPOS = 0x0000001A;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SETPROGRESSPOS.

132.4.92 SHDocVw.OLECMDID_SETPROGRESSTEXT Constant

Pascal

```
OLECMDID_SETPROGRESSTEXT = $0000001B;
```

C++

```
OLECMDID_SETPROGRESSTEXT = 0x0000001B;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SETPROGRESSTEXT.

132.4.93 SHDocVw.OLECMDID_SETTITLE Constant

Pascal

```
OLECMDID_SETTITLE = $0000001C;
```

C++

```
OLECMDID_SETTITLE = 0x0000001C;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SETTITLE.

132.4.94 SHDocVw.OLECMDID_SHOWFIND Constant

Pascal

```
OLECMDID_SHOWFIND = $0000002A;
```

C++

```
OLECMDID_SHOWFIND = 0x0000002A;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SHOWFIND.

132.4.95 SHDocVw.OLECMDID_SHOWMESSAGE Constant

Pascal

```
OLECMDID_SHOWMESSAGE = $00000029;
```

C++

```
OLECMDID_SHOWMESSAGE = 0x00000029;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SHOWMESSAGE.

132.4.96 SHDocVw.OLECMDID_SHOWPAGESETUP Constant

Pascal

```
OLECMDID_SHOWPAGESETUP = $0000002B;
```

C++

```
OLECMDID_SHOWPAGESETUP = 0x0000002B;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SHOWPAGESETUP.

132.4.97 SHDocVw.OLECMDID_SHOWPRINT Constant

Pascal

```
OLECMDID_SHOWPRINT = $0000002C;
```

C++

```
OLECMDID_SHOWPRINT = 0x0000002C;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SHOWPRINT.

132.4.98 SHDocVw.OLECMDID_SHOWSCRIPTERROR Constant

Pascal

```
OLECMDID_SHOWSCRIPTERROR = $00000028;
```

C++

```
OLECMDID_SHOWSCRIPTERROR = 0x00000028;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SHOWSCRIPTERROR.

132.4.99 SHDocVw.OLECMDID_SPELL Constant

Pascal

```
OLECMDID_SPELL = $00000009;
```

C++

```
OLECMDID_SPELL = 0x00000009;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SPELL.

132.4.100 SHDocVw.OLECMDID_STOP Constant

Pascal

```
OLECMDID_STOP = $00000017;
```

C++

```
OLECMDID_STOP = 0x00000017;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_STOP.

132.4.101 SHDocVw.OLECMDID_STOPDOWNLOAD Constant

Pascal

```
OLECMDID_STOPDOWNLOAD = $0000001E;
```

C++

```
OLECMDID_STOPDOWNLOAD = 0x0000001E;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_STOPDOWNLOAD.

132.4.102 SHDocVw.OLECMDID_UNDO Constant

Pascal

```
OLECMDID_UNDO = $0000000F;
```

C++

```
OLECMDID_UNDO = 0x0000000F;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_UNDO.

132.4.103 SHDocVw.OLECMDID_UPDATECOMMANDS Constant

Pascal

```
OLECMDID_UPDATECOMMANDS = $00000015;
```

C++

```
OLECMDID_UPDATECOMMANDS = 0x00000015;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_UPDATECOMMANDS.

132.4.104 SHDocVw.OLECMDID_UPDATEPAGESTATUS Constant

Pascal

```
OLECMDID_UPDATEPAGESTATUS = $00000030;
```

C++

```
OLECMDID_UPDATEPAGESTATUS = 0x00000030;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_UPDATEPAGESTATUS.

132.4.105 SHDocVw.OLECMDID_ZOOM Constant

Pascal

```
OLECMDID_ZOOM = $00000013;
```

C++

```
OLECMDID_ZOOM = 0x00000013;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_ZOOM.

132.4.106 SHDocVw.READYSTATE_COMPLETE Constant

Pascal

```
READYSTATE_COMPLETE = $00000004;
```

C++

```
READYSTATE_COMPLETE = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.READYSTATE_COMPLETE.

132.4.107 SHDocVw.READYSTATE_INTERACTIVE Constant

Pascal

```
READYSTATE_INTERACTIVE = $00000003;
```

C++

```
READYSTATE_INTERACTIVE = 0x00000003;
```

File

SHDocVw

Description

This is constant SHDocVw.READYSTATE_INTERACTIVE.

132.4.108 SHDocVw.READYSTATE_LOADED Constant

Pascal

```
READYSTATE_LOADED = $00000002;
```

C++

```
READYSTATE_LOADED = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.READYSTATE_LOADED.

132.4.109 SHDocVw.READYSTATE_LOADING Constant

Pascal

```
READYSTATE_LOADING = $00000001;
```

C++

```
READYSTATE_LOADING = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.READYSTATE_LOADING.

132.4.110 SHDocVw.READYSTATE_UNINITIALIZED Constant

Pascal

```
READYSTATE_UNINITIALIZED = $00000000;
```

C++

```
READYSTATE_UNINITIALIZED = 0x00000000;
```

File

SHDocVw

Description

This is constant SHDocVw.READYSTATE_UNINITIALIZED.

132.4.111 SHDocVw.REFRESH_COMPLETELY Constant

Pascal

```
REFRESH_COMPLETELY = 3;
```

C++

```
REFRESH_COMPLETELY = 3;
```

File

SHDocVw

Description

This is constant SHDocVw.REFRESH_COMPLETELY.

132.4.112 SHDocVw.REFRESH_CONTINUE Constant

Pascal

```
REFRESH_CONTINUE = 2;
```

C++

```
REFRESH_CONTINUE = 2;
```

File

SHDocVw

Description

This is constant SHDocVw.REFRESH_CONTINUE.

132.4.113 SHDocVw.REFRESH_IFEXPIRED Constant

Pascal

```
REFRESH_IFEXPIRED = 1;
```

C++

```
REFRESH_IFEXPIRED = 1;
```

File

SHDocVw

Description

This is constant SHDocVw.REFRESH_IFEXPIRED.

132.4.114 SHDocVw.REFRESH_NORMAL Constant

Pascal

```
REFRESH_NORMAL = 0;
```

C++

```
REFRESH_NORMAL = 0;
```

File

SHDocVw

Description

This is constant SHDocVw.REFRESH_NORMAL.

132.4.115 SHDocVw.secureLockIconMixed Constant

Pascal

```
secureLockIconMixed = $00000001;
```

C++

```
secureLockIconMixed = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconMixed.

132.4.116 SHDocVw.secureLockIconSecure128Bit Constant

Pascal

```
secureLockIconSecure128Bit = $00000006;
```

C++

```
secureLockIconSecure128Bit = 0x00000006;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconSecure128Bit.

132.4.117 SHDocVw.secureLockIconSecure40Bit Constant

Pascal

```
secureLockIconSecure40Bit = $00000003;
```

C++

```
secureLockIconSecure40Bit = 0x00000003;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconSecure40Bit.

132.4.118 SHDocVw.secureLockIconSecure56Bit Constant

Pascal

```
secureLockIconSecure56Bit = $00000004;
```

C++

```
secureLockIconSecure56Bit = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconSecure56Bit.

132.4.119 SHDocVw.secureLockIconSecureFortezza Constant

Pascal

```
secureLockIconSecureFortezza = $00000005;
```

C++

```
secureLockIconSecureFortezza = 0x00000005;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconSecureFortezza.

132.4.120 SHDocVw.secureLockIconSecureUnknownBits Constant

Pascal

```
secureLockIconSecureUnknownBits = $00000002;
```

C++

```
secureLockIconSecureUnknownBits = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconSecureUnknownBits.

132.4.121 SHDocVw.secureLockIconUnsecure Constant

Pascal

```
secureLockIconUnsecure = $00000000;
```

C++

```
secureLockIconUnsecure = 0x00000000;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconUnsecure.

132.4.122 SHDocVw.SHDocVwMajorVersion Constant

Pascal

```
SHDocVwMajorVersion = 1;
```

C++

```
SHDocVwMajorVersion = 1;
```

File

SHDocVw

Description

This is constant SHDocVw.SHDocVwMajorVersion.

132.4.123 SHDocVw.SHDocVwMinorVersion Constant

Pascal

```
SHDocVwMinorVersion = 1;
```

C++

```
SHDocVwMinorVersion = 1;
```

File

SHDocVw

Description

This is constant SHDocVw.SHDocVwMinorVersion.

132.4.124 SHDocVw.SWC_3RDPARTY Constant

Pascal

```
SWC_3RDPARTY = $00000002;
```

C++

```
SWC_3RDPARTY = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.SWC_3RDPARTY.

132.4.125 SHDocVw.SWC_BROWSER Constant

Pascal

```
SWC_BROWSER = $00000001;
```

C++

```
SWC_BROWSER = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.SWC_BROWSER.

132.4.126 SHDocVw.SWC_CALLBACK Constant

Pascal

```
SWC_CALLBACK = $00000004;
```

C++

```
SWC_CALLBACK = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.SWC_CALLBACK.

132.4.127 SHDocVw.SWC_EXPLORER Constant

Pascal

```
SWC_EXPLORER = $00000000;
```

C++

```
SWC_EXPLORER = 0x00000000;
```

File

SHDocVw

Description

This is constant SHDocVw.SWC_EXPLORER.

132.4.128 SHDocVw.SWFO_COOKIEPASSED Constant

Pascal

```
SWFO_COOKIEPASSED = $00000004;
```

C++

```
SWFO_COOKIEPASSED = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.SWFO_COOKIEPASSED.

132.4.129 SHDocVw.SWFO_INCLUDEPENDING Constant

Pascal

```
SWFO_INCLUDEPENDING = $00000002;
```

C++

```
SWFO_INCLUDEPENDING = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.SWFO_INCLUDEPENDING.

132.4.130 SHDocVw.SWFO_NEEDDISPATCH Constant

Pascal

```
SWFO_NEEDDISPATCH = $00000001;
```

C++

```
SWFO_NEEDDISPATCH = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.SWFO_NEEDDISPATCH.

133 ShellAnimations Namespace

This is namespace ShellAnimations.

Topics

Name	Description
Classes (see page 19534)	The following table lists classes in this documentation.

Classes

	Name	Description
	TShellResources (see page 19534)	TShellResources provides forwards compability for programs that use system animations in Windows Vista.

133.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TShellResources (see page 19534)	TShellResources provides forwards compability for programs that use system animations in Windows Vista.

133.1.1 TShellResources Class

TShellResources provides forwards compability for programs that use system animations in Windows Vista.

Class Hierarchy

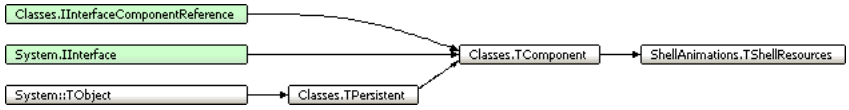

Pascal

```
TShellResources = class(TComponent);
```

C++

```
class TShellResources : public TComponent;
```

File

ShellAnimations

Description

Windows Vista shell animations are no longer accessible outside of the system; programs must now store local copies of these animations instead of linking to them as was done in the past.

Placing TShellResources onto a form copies the following system animation resources into the program:

- CopyFile.res
- CopyFiles.res
- DeleteFile.res
- EmptyRecycle.res
- FindComputer.res
- FindFiles.res
- FindFolder.res
- RecycleFile.res

TShellResources should only be used in programs that are run on Vista and use animations from the common library, otherwise this function adds unneeded overhead.

See Also

TAnimate (see page 3422)

134 SimpleDS Namespace

This is namespace SimpleDS.

Topics

Name	Description
Classes (see page 19538)	The following table lists classes in this documentation.

Classes

	Name	Description
	TInternalSQLDataSet (see page 19538)	This is class SimpleDS.TInternalSQLDataSet.
	TSimpleDataSet (see page 19555)	TSimpleDataSet uses dbExpress to fetch data and cache it internally in memory.

134.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TInternalSQLDataSet (see page 19538)	This is class SimpleDS.TInternalSQLDataSet.
	TSimpleDataSet (see page 19555)	TSimpleDataSet uses dbExpress to fetch data and cache it internally in memory.

134.1.1 TInternalSQLDataSet Class

Class Hierarchy

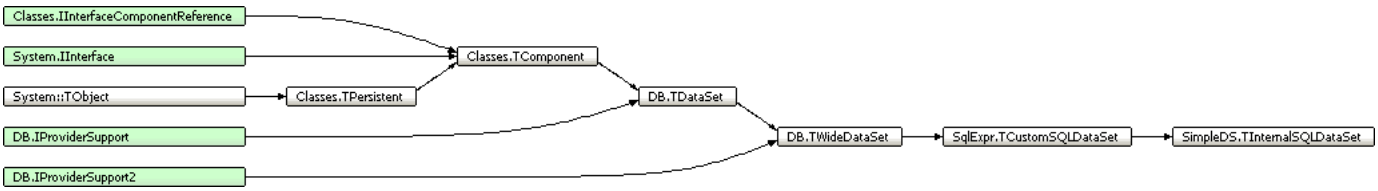

Pascal

```
TInternalSQLDataSet = class(TCustomSQLDataSet);
```

C++

```
class TInternalSQLDataSet : public TCustomSQLDataSet;
```

File

SimpleDS

Description

This is class SimpleDS.TInternalSQLDataSet.

134.1.1.1 TInternalSQLDataSet Members

The following tables list the members exposed by TInternalSQLDataSet.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	AddFieldDesc (see page 20163)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 20163)	This is AddIndexDefs, a member of class TCustomSQLDataSet.

	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
	CheckPrepareError (see page 20164)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
	ClearIndexDefs (see page 20164)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
	CloseCursor (see page 20164)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 20164)	This is CloseStatement, a member of class TCustomSQLDataSet.
	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
	DataEvent (see page 6924)	This is DataEvent, a member of class TDataSet.
	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 20165)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	ExecSQL (see page 20165)	This is ExecSQL, a member of class TCustomSQLDataSet.
	ExecuteStatement (see page 20166)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.

	FreeBuffers (see page 20166)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCommand (see page 20166)	This is FreeCommand, a member of class TCustomSQLDataSet.
V	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeReader (see page 20166)	This is FreeReader, a member of class TCustomSQLDataSet.
V	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
V	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
V	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
V	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
V	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
V	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
V	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
V	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
V	GetCanModify (see page 20167)	This is GetCanModify, a member of class TCustomSQLDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
V	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
V	GetDataSource (see page 20167)	This is GetDataSource, a member of class TCustomSQLDataSet.
V	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
V	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
V	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
V	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
V	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetObjectTypeNames (see page 20169)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 20169)	This is GetOutputParams, a member of class TCustomSQLDataSet.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
V	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
V	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
A	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
V	GetRecord (see page 20170)	This is GetRecord, a member of class TCustomSQLDataSet.
V	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
V	GetRecordCount (see page 20170)	This is GetRecordCount, a member of class TCustomSQLDataSet.
V	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
	GetSortFieldNames (see page 20170)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
V	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
V	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
V	InitRecord (see page 20170)	This is InitRecord, a member of class TCustomSQLDataSet.
V	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
V	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
A	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.
V	InternalClose (see page 20171)	This is InternalClose, a member of class TCustomSQLDataSet.
V	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
V	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.

	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
	InternalHandleException (see page 20171)	This is InternalHandleException, a member of class TCustomSQLDataSet.
	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
	InternalInitFieldDefs (see page 20171)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
	InternalOpen (see page 20171)	This is InternalOpen, a member of class TCustomSQLDataSet.
	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
	InternalRefresh (see page 20171)	This is InternalRefresh, a member of class TCustomSQLDataSet.
	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
	IsCursorOpen (see page 20172)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
	Loaded (see page 20172)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 20173)	This is LocateRecord, a member of class TCustomSQLDataSet.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	OpenCursor (see page 6976)	This is OpenCursor, a member of class TDataSet.
	OpenCursor (see page 20174)	This is OpenCursor, a member of class TCustomSQLDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	OpenSchema (see page 20174)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 20175)	This is PropertyChanged, a member of class TCustomSQLDataSet.
	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
	PSEndTransaction (see page 20175)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
	PSExecute (see page 20176)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 20176)	This is PSExecuteStatement, a member of class TCustomSQLDataSet.
	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSExecuteStatement (see page 7535)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 20176)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 6982)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 7537)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 20177)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandTextW (see page 7537)	This is PSGetCommandTextW, a member of class TWideDataSet.
	PSGetCommandType (see page 6982)	Gets the command type to be executed by PSExecute.
	PSGetCommandType (see page 20177)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 20178)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 20178)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.

	PSGetKeyFields (see page 7538)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetKeyFieldsW (see page 7538)	This is PSGetKeyFieldsW, a member of class TWideDataSet.
	PSGetKeyFieldsW (see page 20179)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetParams (see page 20179)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7538)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetQuoteCharW (see page 7539)	This is PSGetQuoteCharW, a member of class TWideDataSet.
	PSGetQuoteCharW (see page 20179)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7539)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetTableNameW (see page 7539)	This is PSGetTableNameW, a member of class TWideDataSet.
	PSGetTableNameW (see page 20179)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSGetUpdateException (see page 20180)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 20180)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 20180)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 20181)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 20181)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 20182)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetCommandText (see page 7540)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 20182)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSStartTransaction (see page 20182)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 20183)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetBufListSize (see page 20184)	This is SetBufListSize, a member of class TCustomSQLDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCommandText (see page 20184)	This is SetCommandText, a member of class TCustomSQLDataSet.

V	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
V	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFCommandText (see page 20184)	This is SetFCommandText, a member of class TCustomSQLDataSet.
V	SetFieldData (see page 20184)	This is SetFieldData, a member of class TCustomSQLDataSet.
V	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
V	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
V	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
V	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetMaxBlobSize (see page 20184)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
V	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetParamsFromCursor (see page 20185)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
	SetSortFieldNames (see page 20186)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
V	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.
V	UpdateIndexDefs (see page 20186)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
	BlobBuffer (see page 20189)	Reserves a buffer for storing BLOB values.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.

R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
	CurrentBlobSize (see page 20191)	Indicates the size of the most recently read BLOB field.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
R	DataLink (see page 20191)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
R	InternalConnection (see page 20194)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	LastError (see page 20194)	Saves the error message for the last error returned by dbExpress.
	NativeCommand (see page 20195)	Represents the SQL command that is sent to the server.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
R	ParamCount (see page 20197)	Indicates the number of parameters listed by the Params (see page 20197) property.
	ProcParams (see page 20199)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
	Reserved (see page 7035)	For internal use only.
R	RowsAffected (see page 20200)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 20200)	Identifies the metadata with which to populate the dataset.

Public Events

	Name	Description
	AfterCancel (see page 7038)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 7040)	Occurs after an application closes a dataset.
	AfterDelete (see page 7041)	Occurs after an application deletes a record.
	AfterEdit (see page 7044)	Occurs after an application starts editing a record.
	AfterInsert (see page 7044)	Occurs after an application inserts a new record.
	AfterOpen (see page 7045)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 7046)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 7047)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 7047)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 7048)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 7048)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 7049)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 7049)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 7050)	Occurs before an application enters insert mode.
	BeforeOpen (see page 7051)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 7052)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 7052)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 7053)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 7053)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 7054)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 7054)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 7056)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 7056)	Occurs when an application attempts to modify or insert a record and an exception is raised.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
V	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Close (see page 6920)	Closes a dataset.
V	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 6904)	Creates an instance of a TDataSet (see page 6894) component.
V	Create (see page 20160)	Creates an instance of a TCustomSQLDataSet (see page 20150) component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
V	CreateBlobStream (see page 20164)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 6904)	Destroys the TDataSet (see page 6894) instance.
V	Destroy (see page 20161)	Destroys the TCustomSQLDataSet (see page 20150) instance.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.

	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
V	GetBlobFieldData (see page 20166)	Retrieves the current value of a BLOB field into a buffer.
V	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
V	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
V	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
V	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
V	GetDetailLinkFields (see page 20167)	Lists the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
V	GetFieldData (see page 20168)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
V	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetKeyFieldNames (see page 20169)	Fills a list with the names of all fields in all indexes defined for the dataset.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetQuoteChar (see page 20169)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
V	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
V	IsSequenced (see page 20172)	Indicates whether the dataset can use record numbers to indicate the order of records.

	Last (see page 6968)	Moves to the last record in the dataset.
V	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Locate (see page 20173)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
V	Lookup (see page 20174)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	Open (see page 6976)	Opens the dataset.
	ParamByName (see page 20175)	This is ParamByName, a member of class TCustomSQLDataSet.
V	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
V	PrepareStatement (see page 20175)	Performs a prepare on the SQL statement.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSEndTransaction (see page 7589)	Ends a transaction that was started by a call to PSStartTransaction (see page 7597).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecute (see page 7590)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSExecuteStatement (see page 7590)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 7591)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 7579)	Gets the SQL command to be executed by PSExecute (see page 7578).
	PSGetCommandText (see page 7591)	Gets the SQL command to be executed by PSExecute (see page 7590).
	PSGetCommandType (see page 7580)	Gets the command type to be executed by PSExecute (see page 7578).
	PSGetCommandType (see page 7591)	Gets the command type to be executed by PSExecute (see page 7590).
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 7592)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 7592)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7593)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetParams (see page 7593)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7594)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7594)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSGetUpdateException (see page 7594)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.

	PSInTransaction (see page 7595)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 7595)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 7596)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 7596)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7585)	Assigns a specified SQL command to be executed by PSExecute (see page 7578).
	PSSetCommandText (see page 7596)	Assigns a specified SQL command to be executed by PSExecute (see page 7590).
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 7597)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).
	PSStartTransaction (see page 7597)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7589).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 7598)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetSchemaInfo (see page 20185)	Indicates whether the dataset represents metadata from the server, and if so, what kind.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.

Public Properties

	Name	Description
	Active (see page 7004)	Specifies whether or not a dataset is open.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignerData (see page 20192)	Stores custom data.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
	FieldDefs (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
	IndexDefs (see page 20193)	Contains the definitions of all indexes defined on the server for the dataset.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	Modified (see page 7029)	Indicates whether the active record is modified.
	ObjectView (see page 7031)	Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Prepared (see page 20198)	Specifies whether the command is prepared before execution.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
R	RecordCount (see page 20199)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
	SQLConnection (see page 20201)	Specifies the SQL connection component that connects the dataset to a database server.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	TransactionLevel (see page 20202)	Indicates the transaction to which this dataset belongs.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterClose (see page 20203)	Occurs after a data source connection is closed.
	AfterOpen (see page 20203)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 20203)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 20204)	Occurs after a result set is scrolled.
	BeforeClose (see page 20204)	Occurs just before a data connection is closed.
	BeforeOpen (see page 20204)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 20204)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 20204)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 20204)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 20205)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 20189) property.

	ParseInsertSql (see page 20205)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 20189) property.
	ParseSelectSql (see page 20205)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 20189) property.
	ParseUpdateSql (see page 20205)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 20189) property.

Published Properties

	Name	Description
	Active (see page 20187)	Specifies whether or not a dataset is open.
	CommandText (see page 19550)	Specifies the command the dataset executes.
	CommandType (see page 19551)	Indicates the meaning of the CommandText property.
	DataSource (see page 19552)	Links the SQL dataset to another (master) dataset.
	GetMetadata (see page 19552)	Specifies whether the SQL dataset fetches metadata information along with the data.
	MaxBlobSize (see page 19553)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NumericMapping (see page 20195)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 20196)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	ParamCheck (see page 19553)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 19554)	Represents the parameters for a query or stored procedure.
	SchemaName (see page 20201)	Specifies the name of the database schema.
	SortFieldNames (see page 19555)	Indicates the sort order when CommandType is ctTable.
	Tag (see page 2808)	Stores an integer value as part of a component.

134.1.1.2 TInternalSQLDataSet Properties

The properties of the TInternalSQLDataSet class are listed here.

Published Properties

	Name	Description
	CommandText (see page 19550)	Specifies the command the dataset executes.
	CommandType (see page 19551)	Indicates the meaning of the CommandText property.
	DataSource (see page 19552)	Links the SQL dataset to another (master) dataset.
	GetMetadata (see page 19552)	Specifies whether the SQL dataset fetches metadata information along with the data.
	MaxBlobSize (see page 19553)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	ParamCheck (see page 19553)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 19554)	Represents the parameters for a query or stored procedure.
	SortFieldNames (see page 19555)	Indicates the sort order when CommandType is ctTable.

134.1.1.2.1 TInternalSQLDataSet.CommandText Property

Specifies the command the dataset executes.

Pascal

```
property CommandText: WideString;
```

C++

```
__property BSTR CommandText;
```

Description

The value of CommandText depends on the CommandType property:

When CommandType is ctQuery, CommandText is an SQL statement that the dataset executes. If the statement is a SELECT statement, the SQL dataset executes the statement when it is opened. If the statement does not return a result set, the SQL dataset executes the statement when the ExecSQL method is called.

When CommandType is ctStoredProc, CommandText is the name of a stored procedure. The SQL dataset executes the stored procedure when it is opened or when the ExecSQL method is called, depending on whether the stored procedure returns a set of records.

When CommandType is ctTable, CommandText is the name of a table on the database server. The SQL dataset automatically generates a SELECT statement to fetch all the records on all the fields in this table, sorting by the fields listed in the SortFieldNames property. The generated statement is executed when the dataset is opened.

If CommandText is an SQL statement that includes parameters or the name of a stored procedure that has parameters, use the Params property to supply parameter values.

See Also

CommandType (see page 20190)

Params (see page 20197)

Open (see page 6976)

ExecSQL (see page 20278)

SortFieldNames (see page 20201)

Fetching Metadata into a Unidirectional Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Accessing Schema Information

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

134.1.1.2.2 TInternalSQLDataSet.CommandType Property

Indicates the meaning of the CommandText property.

Pascal

```
property CommandType: TSQLCommandType;
```

C++

```
__property TSQLCommandType CommandType;
```

Description

CommandType indicates the type of command that is contained in the CommandText property. TCustomSQLDataSet descendants set this value to the way in which they specify a query or stored procedure.

The following table lists the possible values:

CommandType	Corresponding CommandText
ctQuery	An SQL statement that the dataset executes.
ctStoredProc	The name of a stored procedure.
ctTable	The name of a table on the database server. The SQL dataset automatically generates a SELECT statement to fetch all the records on all the fields in this table.

See Also

CommandText (see page 20189)

Setting Up a Simple Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Specifying What Data to Display

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

134.1.1.2.3 TInternalSQLDataSet.DataSource Property

Links the SQL dataset to another (master) dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;__property TDataSource DataSource;
```

Description

DataSource links this SQL dataset to another (master) dataset that supplies values for this dataset's parameters.

When the SQL dataset's CommandText (or SQL) property specifies a parameterized query but the application does not provide values for the corresponding parameters in the Params property, the SQL dataset fetches parameter values from the dataset identified with the specified data source. When the current record in the dataset associated with DataSource changes, the SQL dataset refreshes its data to reflect the new parameter values.

See Also

CommandText (see page 20189)

SQL (see page 20316)

Params (see page 20197)

MasterSource (see page 20357)

134.1.1.2.4 TInternalSQLDataSet.GetMetadata Property

Specifies whether the SQL dataset fetches metadata information along with the data.

Pascal

```
property GetMetadata: Boolean;
```

C++

```
__property Boolean GetMetadata;
```

Description

Use GetMetadata to turn on or off the fetching of metadata on a database object. Setting GetMetadata to False will improve performance because no additional SQL will execute in order to get index information. However, setting GetMetadata to False might slow any attempts to update data using a client dataset that is linked via a dataset provider because the resolver might not generate optimized SQL to perform the update.

GetMetadata must be True to perform any operations that require an index. These operations include setting up master/detail relationships, updating data from a client dataset when the provider's UpdateMode is not upWhereAll, and delayed fetching of BLOB data.

Set GetMetadata to False if you are fetching a dataset for read-only purposes.

See Also

TClientDataset

TDataSetProvider (see page 18508)

Fetching the Data using TSQLDataSet

134.1.1.2.5 TInternalSQLDataSet.MaxBlobSize Property

Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

MaxBlobSize limits the amount of data that can be fetched from Binary Large Object (BLOB) fields. BLOB fields can hold an indefinite amount of data. However, SQL datasets read that data into a memory buffer. By limiting the amount of data that can be fetched from a BLOB field, you limit the size of the buffer.

If MaxBlobSize is –1, the dataset imposes no maximum BLOB size. If MaxBlobSize is 0, the dataset uses the parameters of the associated SQL connection, if any, to determine the maximum BLOB size.

Note: MaxBlobSize should agree with any parameters on the associated SQL connection component that limit the size of BLOB data.

See Also

GetBlobFieldData (see page 20166)

Params (see page 20253)

SQLConnection (see page 20201)

134.1.1.2.6 TInternalSQLDataSet.ParamCheck Property

Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

By default, the Params property is updated automatically to reflect any parameters that appear in the query specified by the CommandText property (or the SQL property for TSQLQuery) at runtime. That is, whenever the CommandText property changes, the Params property is regenerated to reflect the new set of parameters.

There are times when you want to override this default behavior. For example, when specifying a data definition language (DDL) statement that defines a stored procedure, the resulting SQL command may include parameters. These parameters, however, should not be used as parameters of the SQL dataset, because they are not parameters of the DDL command, but rather, they are parameters of the stored procedure that the command creates.

In order to override the default generation of the Params property in response to changes in CommandText, set ParamCheck to false.

See Also

Params (see page 20197)

CommandText (see page 20189)

SQL (see page 20316)

134.1.1.2.7 TInternalSQLDataSet.Params Property

Represents the parameters for a query or stored procedure.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Params is a collection of TParam objects that represent the parameters of a query or stored procedure specified by the SQL dataset. When you specify a query by setting the CommandText property (or the SQL property in TSQLQuery), the SQL dataset automatically parse the query and fills Params with a TParam object for every parameter in the query. Similarly, when you specify a stored procedure by setting the CommandText property (or the StoredProcName property in TSQLStoredProc), the SQL dataset automatically asks the server for a list of stored procedure parameters and fills Params with a TParam object for every parameter of the stored procedure.

At design-time, use the Params property editor to access the individual TParam objects. For queries, use this property editor to check the DataType property of each parameter and to assign values to the parameters. For stored procedures, you may need to do more, changing the type or number of parameters if the server does not supply complete information about stored procedure parameters.

You can also use Params to assign parameter properties in code at runtime.

Note: If you do not assign values to input parameters before opening the dataset, it fetches values from the dataset identified by the DataSource property. However, this only works if the individual TParam objects have the same names as fields in the master dataset.

See Also

ParamCheck (see page 20196)

SQL (see page 20316)

StoredProcName (see page 20336)

CommandText (see page 20189)

TParam (see page 7386)

Value (see page 7433)

DataType (see page 7427)

DataSource (see page 20192)

Configuring TSQL Connection

Connecting to Databases with TDatabase

Using TQuery

Using TSimpleDataSet

134.1.1.2.8 TInternalSQLDataSet.SortFieldNames Property

Indicates the sort order when CommandType is ctTable.

Pascal

```
property SortFieldNames: WideString;
```

C++

```
__property BSTR SortFieldNames;
```

Description

Use SortFieldNames to specify the fields that should be used to order the records in the dataset. SortFieldNames is only used when CommandType is ctTable.

SortFieldNames is a semicolon-delimited list of field names. These names are used to generate the ORDER BY clause of the SQL statement the dataset creates to fetch data from a server table. Records are sorted by the first named field, and within that sort order, by the second named field, and so on.

See Also

CommandType (see page 20190)

Representing the Records in a Table

Specifying the Data to Display using TSQLDataSet

134.1.2 TSimpleDataSet Class

TSimpleDataSet uses dbExpress to fetch data and cache it internally in memory.

Class Hierarchy

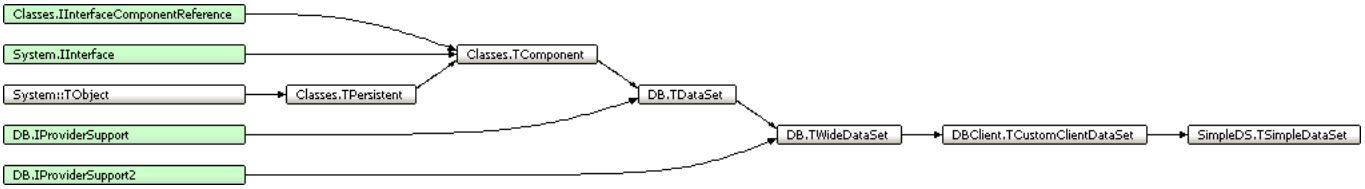

Pascal

```
TSimpleDataSet = class(TCustomClientDataSet);
```

C++

```
class TSimpleDataSet : public TCustomClientDataSet;
```

File

SimpleDS

Description

TSimpleDataSet is a client dataset that uses an internal TSQLDataSet and TDataSetProvider for fetching data and applying updates. It combines the fast access and easy deployment of a unidirectional dataset with a client dataset's ability to edit and navigate data.

TSimpleDataSet uses dbExpress for fast access to database information. As a client dataset, TSimpleDataSet caches that information in memory and saves any updates the application makes. With its internal dataset provider, TSimpleDataSet can apply those updates back to the database server. Thus, TSimpleDataSet works like TClientDataSet when it is connected to a local TSQLDataSet via a provider, except that the source dataset and provider are internal.

TSimpleDataSet surfaces some of the properties and events of TSQLDataSet so that you can specify the database server from which to fetch data, indicate what data to fetch from that server, influence what information is included in data packets, and provide input on the updating process.

In addition to using the source TSQLDataSet, TSimpleDataSet can read from and write to a dedicated file on disk. The internal provider and source dataset are not needed for working with file-based data, making TClientDataSet a more suitable choice if you are writing a purely file-based application. However, the ability to work with a file on disk makes TSimpleDataSet well-suited to briefcase model applications.

See Also

TDataSetProvider (see page 18508)

Choosing the Type of Dataset for Caching Updates

Types of Unidirectional Datasets

134.1.2.1 TSimpleDataSet Members

The following tables list the members exposed by TSimpleDataSet.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	ActivateFilters (see page 8391)	This is ActivateFilters, a member of class TCustomClientDataSet.
	AddDataPacket (see page 8391)	This is AddDataPacket, a member of class TCustomClientDataSet.
	AddFieldDesc (see page 8391)	This is AddFieldDesc, a member of class TCustomClientDataSet.
	AllocConnection (see page 19571)	This is AllocConnection, a member of class TSimpleDataSet.
	AllocDataSet (see page 19571)	This is AllocDataSet, a member of class TSimpleDataSet.
	AllocKeyBuffers (see page 8392)	This is AllocKeyBuffers, a member of class TCustomClientDataSet.
	AllocProvider (see page 19572)	This is AllocProvider, a member of class TSimpleDataSet.
	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
	AllocRecordBuffer (see page 8393)	This is AllocRecordBuffer, a member of class TCustomClientDataSet.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Check (see page 8397)	This is Check, a member of class TCustomClientDataSet.
	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
	CheckDetailRecords (see page 8397)	This is CheckDetailRecords, a member of class TCustomClientDataSet.
	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
	CheckProviderEOF (see page 8397)	This is CheckProviderEOF, a member of class TCustomClientDataSet.
	CheckSetKeyMode (see page 8397)	This is CheckSetKeyMode, a member of class TCustomClientDataSet.
	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
	ClearCalcFields (see page 8398)	This is ClearCalcFields, a member of class TCustomClientDataSet.
	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
	CloseCursor (see page 8400)	This is CloseCursor, a member of class TCustomClientDataSet.
	ConstraintsStored (see page 8401)	This is ConstraintsStored, a member of class TCustomClientDataSet.
	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
	DataConvert (see page 8404)	This is DataConvert, a member of class TCustomClientDataSet.
	DataEvent (see page 6924)	This is DataEvent, a member of class TDataSet.
	DataEvent (see page 8404)	This is DataEvent, a member of class TCustomClientDataSet.
	DeactivateFilters (see page 8405)	This is DeactivateFilters, a member of class TCustomClientDataSet.
	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
	DefChanged (see page 8405)	This is DefChanged, a member of class TCustomClientDataSet.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 8405)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
	DestroyLookupCursor (see page 8406)	This is DestroyLookupCursor, a member of class TCustomClientDataSet.
	DoAfterApplyUpdates (see page 8407)	This is DoAfterApplyUpdates, a member of class TCustomClientDataSet.
	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.

	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
	DoAfterExecute (see page 8407)	This is DoAfterExecute, a member of class TCustomClientDataSet.
	DoAfterGetParams (see page 8407)	This is DoAfterGetParams, a member of class TCustomClientDataSet.
	DoAfterGetRecords (see page 8407)	This is DoAfterGetRecords, a member of class TCustomClientDataSet.
	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterRowRequest (see page 8408)	This is DoAfterRowRequest, a member of class TCustomClientDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoApplyUpdates (see page 8408)	This is DoApplyUpdates, a member of class TCustomClientDataSet.
	DoBeforeApplyUpdates (see page 8408)	This is DoBeforeApplyUpdates, a member of class TCustomClientDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeExecute (see page 8408)	This is DoBeforeExecute, a member of class TCustomClientDataSet.
	DoBeforeGetParams (see page 8408)	This is DoBeforeGetParams, a member of class TCustomClientDataSet.
	DoBeforeGetRecords (see page 8409)	This is DoBeforeGetRecords, a member of class TCustomClientDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeInsert (see page 8409)	This is DoBeforeInsert, a member of class TCustomClientDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeRowRequest (see page 8409)	This is DoBeforeRowRequest, a member of class TCustomClientDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoExecute (see page 8409)	This is DoExecute, a member of class TCustomClientDataSet.
	DoGetRecords (see page 8410)	This is DoGetRecords, a member of class TCustomClientDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	DoOnNewRecord (see page 8410)	This is DoOnNewRecord, a member of class TCustomClientDataSet.
	DoRowRequest (see page 8410)	This is DoRowRequest, a member of class TCustomClientDataSet.
	FetchMoreData (see page 8415)	This is FetchMoreData, a member of class TCustomClientDataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FindRecord (see page 8417)	This is FindRecord, a member of class TCustomClientDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeKeyBuffers (see page 8417)	This is FreeKeyBuffers, a member of class TCustomClientDataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
	FreeRecordBuffer (see page 8417)	This is FreeRecordBuffer, a member of class TCustomClientDataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
	GetAggregateValue (see page 8418)	This is GetAggregateValue, a member of class TCustomClientDataSet.
	GetAppServer (see page 8418)	This is GetAppServer, a member of class TCustomClientDataSet.
	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
	GetBookmarkData (see page 8418)	This is GetBookmarkData, a member of class TCustomClientDataSet.
	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
	GetBookmarkFlag (see page 8418)	This is GetBookmarkFlag, a member of class TCustomClientDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
	GetCanModify (see page 8418)	This is GetCanModify, a member of class TCustomClientDataSet.

	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
V	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
V	GetDataSource (see page 8419)	This is GetDataSource, a member of class TCustomClientDataSet.
V	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
V	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIndexField (see page 8421)	This is GetIndexField, a member of class TCustomClientDataSet.
	GetIndexFieldCount (see page 8421)	This is GetIndexFieldCount, a member of class TCustomClientDataSet.
V	GetIsClone (see page 8422)	This is GetIsClone, a member of class TCustomClientDataSet.
V	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
V	GetIsIndexField (see page 8422)	This is GetIsIndexField, a member of class TCustomClientDataSet.
	GetKeyBuffer (see page 8423)	This is GetKeyBuffer, a member of class TCustomClientDataSet.
	GetKeyExclusive (see page 8423)	This is GetKeyExclusive, a member of class TCustomClientDataSet.
	GetKeyFieldCount (see page 8423)	This is GetKeyFieldCount, a member of class TCustomClientDataSet.
V	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
V	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
V	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
V	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
V	GetRecNo (see page 8425)	This is GetRecNo, a member of class TCustomClientDataSet.
A	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
V	GetRecord (see page 8425)	This is GetRecord, a member of class TCustomClientDataSet.
V	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
V	GetRecordCount (see page 8426)	This is GetRecordCount, a member of class TCustomClientDataSet.
V	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
V	GetRecordSize (see page 8426)	This is GetRecordSize, a member of class TCustomClientDataSet.
V	GetRemoteServer (see page 8426)	This is GetRemoteServer, a member of class TCustomClientDataSet.
V	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
V	GetStateFieldValue (see page 8426)	This is GetStateFieldValue, a member of class TCustomClientDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
	InitKeyBuffer (see page 8428)	This is InitKeyBuffer, a member of class TCustomClientDataSet.
V	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
V	InitRecord (see page 8428)	This is InitRecord, a member of class TCustomClientDataSet.
V	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
V	InternalAddRecord (see page 8428)	This is InternalAddRecord, a member of class TCustomClientDataSet.
V	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
V	InternalCancel (see page 8429)	This is InternalCancel, a member of class TCustomClientDataSet.
A	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.
V	InternalClose (see page 8429)	This is InternalClose, a member of class TCustomClientDataSet.
V	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
V	InternalDelete (see page 8429)	This is InternalDelete, a member of class TCustomClientDataSet.
V	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
V	InternalEdit (see page 8429)	This is InternalEdit, a member of class TCustomClientDataSet.

	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
	InternalFirst (see page 8430)	This is InternalFirst, a member of class TCustomClientDataSet.
	InternalGetOptionalParam (see page 8430)	This is InternalGetOptionalParam, a member of class TCustomClientDataSet.
	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
	InternalGotoBookmark (see page 8430)	This is InternalGotoBookmark, a member of class TCustomClientDataSet.
	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
	InternalHandleException (see page 8430)	This is InternalHandleException, a member of class TCustomClientDataSet.
	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
	InternalInitFieldDefs (see page 8430)	This is InternalInitFieldDefs, a member of class TCustomClientDataSet.
	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
	InternalInitRecord (see page 8431)	This is InternalInitRecord, a member of class TCustomClientDataSet.
	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
	InternalInsert (see page 8431)	This is InternalInsert, a member of class TCustomClientDataSet.
	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
	InternalLast (see page 8431)	This is InternalLast, a member of class TCustomClientDataSet.
	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
	InternalOpen (see page 8431)	This is InternalOpen, a member of class TCustomClientDataSet.
	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
	InternalPost (see page 8432)	This is InternalPost, a member of class TCustomClientDataSet.
	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
	InternalRefresh (see page 8432)	This is InternalRefresh, a member of class TCustomClientDataSet.
	InternalSetOptionalParam (see page 8432)	This is InternalSetOptionalParam, a member of class TCustomClientDataSet.
	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	InternalSetToRecord (see page 8432)	This is InternalSetToRecord, a member of class TCustomClientDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
	IsCursorOpen (see page 8432)	This is IsCursorOpen, a member of class TCustomClientDataSet.
	Loaded (see page 19572)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 8436)	This is LocateRecord, a member of class TCustomClientDataSet.
	Notification (see page 19572)	Forwards notification messages to all owned components.
	OpenCursor (see page 19573)	This is OpenCursor, a member of class TSimpleDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PostKeyBuffer (see page 8439)	This is PostKeyBuffer, a member of class TCustomClientDataSet.
	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSExecuteStatement (see page 7535)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 19573)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandTextW (see page 7537)	This is PSGetCommandTextW, a member of class TWideDataSet.
	PSGetCommandType (see page 6982)	Gets the command type to be executed by PSExecute.
	PSGetCommandType (see page 8440)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 8440)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7538)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetKeyFieldsW (see page 7538)	This is PSGetKeyFieldsW, a member of class TWideDataSet.

	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7538)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetQuoteCharW (see page 7539)	This is PSGetQuoteCharW, a member of class TWideDataSet.
	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7539)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetTableNameW (see page 7539)	This is PSGetTableNameW, a member of class TWideDataSet.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetCommandText (see page 7540)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadDataPacket (see page 8441)	This is ReadDataPacket, a member of class TCustomClientDataSet.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	RefreshInternalCalcFields (see page 8441)	This is RefreshInternalCalcFields, a member of class TCustomClientDataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	ResetAggField (see page 8442)	This is ResetAggField, a member of class TCustomClientDataSet.
	ResetCursorRange (see page 8443)	This is ResetCursorRange, a member of class TCustomClientDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAggregates (see page 8445)	This is SetAggregates, a member of class TCustomClientDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppServer (see page 8446)	This is SetAppServer, a member of class TCustomClientDataSet.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkData (see page 8446)	This is SetBookmarkData, a member of class TCustomClientDataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkFlag (see page 8446)	This is SetBookmarkFlag, a member of class TCustomClientDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCommandText (see page 8447)	This is SetCommandText, a member of class TCustomClientDataSet.
	SetConnection (see page 19574)	This is SetConnection, a member of class TSimpleDataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetCursorRange (see page 8447)	This is SetCursorRange, a member of class TCustomClientDataSet.
	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDataSetField (see page 8447)	This is SetDataSetField, a member of class TCustomClientDataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldData (see page 8447)	This is SetFieldData, a member of class TCustomClientDataSet.
	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.

	SetFilterData (see page 8448)	This is SetFilterData, a member of class TCustomClientDataSet.
V	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
V	SetFiltered (see page 8448)	This is SetFiltered, a member of class TCustomClientDataSet.
V	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
V	SetFilterOptions (see page 8448)	This is SetFilterOptions, a member of class TCustomClientDataSet.
V	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
V	SetFilterText (see page 8448)	This is SetFilterText, a member of class TCustomClientDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetIndexField (see page 8448)	This is SetIndexField, a member of class TCustomClientDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetKeyBuffer (see page 8449)	This is SetKeyBuffer, a member of class TCustomClientDataSet.
	SetKeyExclusive (see page 8449)	This is SetKeyExclusive, a member of class TCustomClientDataSet.
	SetKeyFieldCount (see page 8450)	This is SetKeyFieldCount, a member of class TCustomClientDataSet.
	SetKeyFields (see page 8450)	This is SetKeyFields, a member of class TCustomClientDataSet.
	SetLinkRanges (see page 8450)	This is SetLinkRanges, a member of class TCustomClientDataSet.
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
V	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
V	SetOnFilterRecord (see page 8450)	This is SetOnFilterRecord, a member of class TCustomClientDataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetProviderEOF (see page 8452)	This is SetProviderEOF, a member of class TCustomClientDataSet.
V	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
V	SetRecNo (see page 8454)	This is SetRecNo, a member of class TCustomClientDataSet.
V	SetRemoteServer (see page 8454)	This is SetRemoteServer, a member of class TCustomClientDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
V	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	SwitchToIndex (see page 8455)	This is SwitchToIndex, a member of class TCustomClientDataSet.
	SyncCursors (see page 8455)	This is SyncCursors, a member of class TCustomClientDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.
V	UpdateIndexDefs (see page 8455)	This is UpdateIndexDefs, a member of class TCustomClientDataSet.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteDataPacket (see page 8456)	This is WriteDataPacket, a member of class TCustomClientDataSet.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.

R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
	CommandText (see page 8465)	Specifies an SQL command to be executed by the database server.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
	DSBase (see page 8471)	Specifies the interface the client dataset uses to communicate with classes in Midas.dll or Midaslib.dcu (on Windows) or midas.so (on Linux).
R	DSCursor (see page 8471)	Provides access to the interface for the client dataset's data cursor.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
R	IsClone (see page 8484)	This is IsClone, a member of class TCustomClientDataSet.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
	ProviderEOF (see page 8491)	Indicates whether the client dataset has all of its data.
R	Ranged (see page 8492)	This is Ranged, a member of class TCustomClientDataSet.
	Reserved (see page 7035)	For internal use only.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
	AddIndex (see page 8391)	Creates a new index for the client dataset.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendData (see page 8393)	Adds a newly fetched packet of records to those already stored in the client dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
	ApplyRange (see page 8393)	Applies a range to the dataset.
V	ApplyUpdates (see page 8394)	Sends all updated, inserted, and deleted records from the client dataset to the provider for writing to the database.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
V	BookmarkValid (see page 8395)	Tests the validity of a specified bookmark.
V	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
V	Cancel (see page 8395)	Cancels unposted changes to the current record.
	CancelRange (see page 8396)	Removes any ranges currently in effect for the client dataset.
	CancelUpdates (see page 8396)	Clears all pending edits and clears the change log.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
V	CloneCursor (see page 8398)	Shares the data belonging to another client dataset.
	Close (see page 6920)	Closes a dataset.
V	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.

	CompareBookmarks (see page 8400)	Compares two bookmarks.
	ConstraintsDisabled (see page 8400)	Indicates whether constraints are enabled for a client dataset.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
	Create (see page 19570)	Creates and initializes an instance of TSimpleDataSet (see page 19555).
	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
	CreateBlobStream (see page 8401)	Returns a TClientBlobStream (see page 8285) object for reading or writing the data in a specified blob field.
	CreateDataSet (see page 8401)	Creates a new, empty client dataset.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
	DataRequest (see page 8404)	Generates an OnDataRequest event on the associated provider
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
	DeleteIndex (see page 8406)	Deletes an index for the client dataset.
	Destroy (see page 19571)	Destroys the instance of TSimpleDataSet (see page 19555).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableConstraints (see page 8406)	Disables the enforcement of constraints.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EditKey (see page 8410)	Enables modification of the search key buffer.
	EditRangeEnd (see page 8411)	Enables changing the ending value for an existing range.
	EditRangeStart (see page 8411)	Enables changing the starting value for an existing range.
	EmptyDataSet (see page 8412)	Removes all records from the dataset.
	EnableConstraints (see page 8412)	Reenables enforcement of constraints when records are posted.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	Execute (see page 8413)	Executes an SQL command by the provider's dataset.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FetchBlobs (see page 8414)	Retrieves BLOB field values from the provider.
	FetchDetails (see page 8414)	Retrieves nested detail datasets from the provider.
	FetchParams (see page 8415)	Retrieves parameter values from the provider.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindKey (see page 8416)	Searches for a record containing specified field values.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNearest (see page 8416)	Moves the cursor to the record that most closely matches a specified set of key values.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
V	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
V	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
V	GetCurrentRecord (see page 8419)	Retrieves the current record into a buffer.
V	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
V	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
V	GetFieldData (see page 8419)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
V	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetGroupState (see page 8420)	Indicates where the current record sits within a specified group of records.
	GetIndexInfo (see page 8421)	Retrieves information about the current index.
	GetIndexNames (see page 8422)	Retrieves a list of available indexes for a client dataset.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetNextPacket (see page 8423)	Retrieves the next block of records from the provider.
	GetOptionalParam (see page 8425)	Retrieves a custom parameter from the data packet.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	GotoCurrent (see page 8427)	Sets the current record in this dataset the same as the current record in another, cloned dataset.
	GotoKey (see page 8427)	Moves the cursor to a record specified by the current key.
	GotoNearest (see page 8427)	Moves the cursor to the record that most closely matches the current key.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
V	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	Last (see page 6968)	Moves to the last record in the dataset.
	LoadFromFile (see page 8433)	Loads a client dataset's data from a file.
	LoadFromStream (see page 8435)	Loads a client dataset's data from a stream.
V	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Locate (see page 8435)	Searches the dataset for a specified record and makes that record current.
V	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
V	Lookup (see page 8436)	Retrieves field values from a record that matches specified search values.
	MergeChangeLog (see page 8437)	Merges any data changes in the change log into the client dataset, overwriting existing values.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	Open (see page 6976)	Opens the dataset.
V	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
V	Post (see page 8438)	Writes a modified record to the Data (see page 8467) property or the change log.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSEndTransaction (see page 7589)	Ends a transaction that was started by a call to PSStartTransaction (see page 7597).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecute (see page 7590)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSExecuteStatement (see page 7590)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 7591)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandType (see page 7580)	Gets the command type to be executed by PSExecute (see page 7578).
	PSGetCommandType (see page 7591)	Gets the command type to be executed by PSExecute (see page 7590).
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 7592)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 7592)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7593)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetParams (see page 7593)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7594)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7594)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSGetUpdateException (see page 7594)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 7595)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 7595)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 7596)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 7596)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7585)	Assigns a specified SQL command to be executed by PSExecute (see page 7578).

	PSSetCommandText (see page 7596)	Assigns a specified SQL command to be executed by PSExecute (see page 7590).
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 7597)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).
	PSStartTransaction (see page 7597)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7589).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 7598)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Reconcile (see page 8441)	Clears successfully updated records from the client dataset's cache of changes.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RefreshRecord (see page 8442)	Updates the base values of the current record to match the current values of the provider's dataset.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
	RevertRecord (see page 8443)	Undoes edits to the current record.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 8443)	Saves a client dataset's data to an external file.
	SaveToStream (see page 8445)	Saves a client dataset's data to a stream.
	SetAltRecBuffers (see page 8446)	Sets record buffers when accessing field values.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetKey (see page 8449)	Enables setting of keys and ranges for a dataset prior to a search.
V	SetOptionalParam (see page 8450)	Saves a custom parameter with the data packet.
	SetProvider (see page 8451)	Associates the client dataset with a provider in the same process space.
	SetRange (see page 8452)	Sets the starting and ending values of a range, and applies it.
	SetRangeEnd (see page 8453)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
	SetRangeStart (see page 8454)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
	UndoLastChange (see page 8455)	Undoes the last edit, insert, or delete operation to a record in the client dataset.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.
V	UpdateStatus (see page 8456)	Reports the update status for the current record in the dataset.

Public Properties

	Name	Description
R	ActiveAggs (see page 8460)	Lists the maintained aggregates supported by the current index.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AppServer (see page 8462)	Provides access to the interface the client dataset uses to communicate with a provider.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ChangeCount (see page 8463)	Indicates the number of changes in the change log.

R	CloneSource (see page 8464)	Indicates the client dataset (if any) with which this client dataset shares data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	ConnectionBroker (see page 8466)	Specifies a connection broker that handles the connection to an application server.
	Data (see page 8467)	Represents the data of the client dataset in a transportable format.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
	DataSetField (see page 8469)	Indicates the TDataSetField component that represents this client dataset.
R	DataSize (see page 8469)	Indicates the number of bytes required to represent the Data (see page 8467) property.
R	DataSource (see page 7014)	Represents the data source of another dataset that supplies values to the dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Delta (see page 8470)	Represents a packet of change log data.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
R	GroupingLevel (see page 8479)	Indicates the depth of grouping support provided by the current index.
R	HasAppServer (see page 8480)	Indicates whether the client dataset is connected to a provider.
R	IndexFieldCount (see page 8481)	Indicates the number of fields that make up the current index.
	IndexFields (see page 8482)	Specifies the fields associated with the current index.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
	KeyExclusive (see page 8484)	Specifies the upper and lower boundaries for a range.
	KeyFieldCount (see page 8484)	Specifies the number of fields to use when conducting a partial key search on a multi-field key.
R	KeySize (see page 8485)	Indicates the size of the current key used for the dataset.
	LogChanges (see page 8485)	Indicates whether to maintain a change log separate from the data.
R	Modified (see page 7029)	Indicates whether the active record is modified.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ProviderName (see page 8492)	Specifies the name of a provider component that supplies data to and applies updates from the client dataset..
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
	RemoteServer (see page 8493)	Specifies the connection component that this client dataset uses to connect to an application server.
	SavePoint (see page 8494)	Represents the current state of edits when logging changes.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	StatusFilter (see page 8495)	Filters the client dataset so that it exposes only those records with a specific update status.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

	XMLData (see page 8496)	Represents the data of the client dataset in XML.

Published Events

	Name	Description
	AfterApplyUpdates (see page 19598)	Occurs after the client dataset applies updates to the provider but before it attempts to reconcile update errors.
	AfterCancel (see page 19599)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 19601)	Occurs after an application closes a dataset.
	AfterDelete (see page 19602)	Occurs after an application deletes a record.
	AfterEdit (see page 19605)	Occurs after an application starts editing a record.
	AfterExecute (see page 19606)	Occurs after the client dataset receives the results from an execute command.
	AfterGetParams (see page 19606)	Occurs after the client dataset receives parameter values from the provider.
	AfterGetRecords (see page 19607)	Occurs after the client dataset fetches a data packet from the provider.
	AfterInsert (see page 19608)	Occurs after an application inserts a new record.
	AfterOpen (see page 19609)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 19610)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 19610)	Occurs after an application refreshes the data in the dataset.
	AfterRowRequest (see page 19611)	Occurs after the client dataset fetches new information about the current record.
	AfterScroll (see page 19612)	Occurs after an application scrolls from one record to another.
	BeforeApplyUpdates (see page 19612)	Occurs before the client dataset applies updates to the remote server.
	BeforeCancel (see page 19613)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 19613)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 19614)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 19614)	Occurs before an application enters edit mode for the active record.
	BeforeExecute (see page 19615)	Occurs before the client dataset executes the provider's query or stored procedure.
	BeforeGetParams (see page 19616)	Occurs before the client dataset fetches parameter values from the provider.
	BeforeGetRecords (see page 19616)	Occurs before the client dataset fetches a data packet from the provider.
	BeforeInsert (see page 19618)	Occurs before an application enters insert mode.
	BeforeOpen (see page 19619)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 19620)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 19620)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeRowRequest (see page 19621)	Occurs before the client dataset fetches new information about the current record.
	BeforeScroll (see page 19622)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 19622)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 19623)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 19623)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 19624)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 19625)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 19625)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnReconcileError (see page 19625)	Occurs when a client dataset needs to reconcile an update to a record that could not be applied.

Published Properties

	Name	Description
	Active (see page 19575)	Specifies whether the client dataset contains data.
	Aggregates (see page 19577)	Lists all aggregates that apply to the client dataset.
	AggregatesActive (see page 19578)	Specifies whether the client dataset calculates and maintains aggregate values.

	AutoCalcFields (see page 19578)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	Connection (see page 19579)	Identifies the TSQLConnection component that connects the dataset to a database server.
	Constraints (see page 19579)	Specifies record-level constraints that must be met when editing the data.
R	DataSet (see page 19580)	Provides access to the internal dataset that fetches data from the server.
	DisableStringTrim (see page 19580)	Specifies whether trailing spaces are trimmed from field values when records are posted.
	FetchOnDemand (see page 19581)	Indicates whether to fetch data packets from the provider on an as-needed basis.
	FieldDefs (see page 19582)	Points to the list of field definitions for the dataset.
	FileName (see page 19585)	Specifies a file that stores the data of the client dataset.
	Filter (see page 19585)	Specifies the text of the current filter.
	Filtered (see page 19587)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 19588)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
	IndexDefs (see page 19588)	Contains information about the indexes for a client dataset.
	IndexFieldNames (see page 19589)	Lists the fields to use as an index.
	IndexName (see page 19590)	Identifies an index for the client dataset.
	MasterFields (see page 19591)	Names one or more fields in a master table to link with corresponding fields in this dataset in order to establish a master-detail relationship.
	MasterSource (see page 19592)	Specifies a data source component for the master dataset when establishing a detail-master relationship between this dataset and another one.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ObjectView (see page 19592)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	PacketRecords (see page 19593)	Indicates the number or type of records in a single data packet.
	Params (see page 19594)	Contains parameter values that are sent to the provider.
	ReadOnly (see page 19596)	Specifies whether the client dataset is read-only for this application.
	StoreDefs (see page 19597)	Indicates whether field and index definitions are saved with the client dataset.
	Tag (see page 2808)	Stores an integer value as part of a component.

134.1.2.2 TSimpleDataSet.Create Constructor

Creates and initializes an instance of TSimpleDataSet (see page 19555).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSimpleDataSet(TComponent * AOwner);
```

Description

Call Create to create TSimpleDataSet (see page 19555) at runtime. Datasets added to forms or data modules at design time are created automatically.

AOwner is the component that is responsible for freeing the TSimpleDataSet (see page 19555) instance. If AOwner is nil (Delphi) or NULL (C++), the application must explicitly free the dataset.

Create creates the internal source dataset and provider that fetch data from the database server and apply updates back to the database server.

See Also

Owner (see page 2807)

134.1.2.3 TSimpleDataSet.Destroy Destructor

Destroys the instance of TSimpleDataSet (see page 19555).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSimpleDataSet();
```

Description

Do not call Destroy directly in an application. Usually destruction of datasets is handled automatically. If an application creates its own instances of a dataset without assigning an Owner, however, it should call Free, which verifies that the dataset component is not nil before calling Destroy.

See Also

Free (see page 22300)

134.1.2.4 TSimpleDataSet Methods

The methods of the TSimpleDataSet class are listed here.

Protected Methods

	Name	Description
	AllocConnection (see page 19571)	This is AllocConnection, a member of class TSimpleDataSet.
	AllocDataSet (see page 19571)	This is AllocDataSet, a member of class TSimpleDataSet.
	AllocProvider (see page 19572)	This is AllocProvider, a member of class TSimpleDataSet.
	Loaded (see page 19572)	Initializes the component after the form file has been read into memory.
	Notification (see page 19572)	Forwards notification messages to all owned components.
	OpenCursor (see page 19573)	This is OpenCursor, a member of class TSimpleDataSet.
	PSGetCommandText (see page 19573)	Gets the SQL command to be executed by PSExecute.
	SetConnection (see page 19574)	This is SetConnection, a member of class TSimpleDataSet.

134.1.2.4.1 TSimpleDataSet.AllocConnection Method

Pascal

```
procedure AllocConnection; virtual;
```

C++

```
virtual __fastcall AllocConnection();
```

Description

This is AllocConnection, a member of class TSimpleDataSet.

134.1.2.4.2 TSimpleDataSet.AllocDataSet Method

Pascal

```
procedure AllocDataSet; virtual;
```

C++

```
virtual __fastcall AllocDataSet();
```

Description

This is AllocDataSet, a member of class TSimpleDataSet.

134.1.2.4.3 TSimpleDataSet.AllocProvider Method

Pascal

```
procedure AllocProvider; virtual;
```

C++

```
virtual __fastcall AllocProvider();
```

Description

This is AllocProvider, a member of class TSimpleDataSet.

134.1.2.4.4 TSimpleDataSet.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState (see page 2804)

Create (see page 2766)

TReader (see page 2933)

134.1.2.4.5 TSimpleDataSet.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

134.1.2.4.6 TSimpleDataSet.OpenCursor Method

Pascal

```
procedure OpenCursor(InfoQuery: Boolean); override;
```

C++

```
virtual __fastcall OpenCursor(Boolean InfoQuery);
```

Description

This is OpenCursor, a member of class TSimpleDataSet.

134.1.2.4.7 TSimpleDataSet.PSGetCommandText Method

Gets the SQL command to be executed by PSExecute.

Pascal

```
function PSGetCommandText: string; override;
```

C++

```
virtual __fastcall AnsiString PSGetCommandText();
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

PSSetCommandText (see page 7585)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

134.1.2.4.8 TSimpleDataSet.SetConnection Method

Pascal

```
procedure SetConnection(Value: TSQLConnection); virtual;
```

C++

```
virtual __fastcall SetConnection(TSQLConnection Value);
```

Description

This is SetConnection, a member of class TSimpleDataSet.

134.1.2.5 TSimpleDataSet Properties

The properties of the TSimpleDataSet class are listed here.

Published Properties

	Name	Description
	Active (see page 19575)	Specifies whether the client dataset contains data.
	Aggregates (see page 19577)	Lists all aggregates that apply to the client dataset.
	AggregatesActive (see page 19578)	Specifies whether the client dataset calculates and maintains aggregate values.
	AutoCalcFields (see page 19578)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	Connection (see page 19579)	Identifies the TSQLConnection component that connects the dataset to a database server.
	Constraints (see page 19579)	Specifies record-level constraints that must be met when editing the data.
R	DataSet (see page 19580)	Provides access to the internal dataset that fetches data from the server.
	DisableStringTrim (see page 19580)	Specifies whether trailing spaces are trimmed from field values when records are posted.
	FetchOnDemand (see page 19581)	Indicates whether to fetch data packets from the provider on an as-needed basis.
	FieldDefs (see page 19582)	Points to the list of field definitions for the dataset.
	FileName (see page 19585)	Specifies a file that stores the data of the client dataset.
	Filter (see page 19585)	Specifies the text of the current filter.
	Filtered (see page 19587)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 19588)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
	IndexDefs (see page 19588)	Contains information about the indexes for a client dataset.
	IndexFieldNames (see page 19589)	Lists the fields to use as an index.
	IndexName (see page 19590)	Identifies an index for the client dataset.
	MasterFields (see page 19591)	Names one or more fields in a master table to link with corresponding fields in this dataset in order to establish a master-detail relationship.
	MasterSource (see page 19592)	Specifies a data source component for the master dataset when establishing a detail-master relationship between this dataset and another one.
	ObjectView (see page 19592)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	PacketRecords (see page 19593)	Indicates the number or type of records in a single data packet.
	Params (see page 19594)	Contains parameter values that are sent to the provider.

	ReadOnly (see page 19596)	Specifies whether the client dataset is read-only for this application.
	StoreDefs (see page 19597)	Indicates whether field and index definitions are saved with the client dataset.

134.1.2.5.1 TSimpleDataSet.Active Property

Specifies whether the client dataset contains data.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Use Active to determine whether the client dataset makes its data available. When Active is false, the client dataset is closed; It can't manipulate the data it represents. When Active is true, data can be read or edited using the client dataset.

Setting Active to true:

1Fills the client dataset with data. Depending on the other properties of the client dataset, this data comes from:

The file specified by the FileName property.

The provider specified by the ProviderName property or SetProvider method. (For some TCustomClientDataSet descendants, this is an internal provider component)

The data that was active when the client dataset was deactivated (only if the application has run continuously since the dataset was deactivated).

2Triggers a BeforeOpen event.

3Sets the State property to dsBrowse.

4Opens a cursor into the dataset.

5Triggers an After Open event.

If the client dataset has never been active previously, setting Active to true defines the structure (metadata) of the client dataset. This structure is retrieved with the data packet from the provider or stored on disk, or, if there is no data packet, it is built using the current value of the FieldDefs property or from the persistent field components listed by the Fields property.

If an error occurs during the dataset open, the dataset state is set to dsInactive, and the cursor is closed.

Setting Active to false:

1Triggers a BeforeClose event.

2Sets the State property to dsInactive.

3Closes the cursor, saving the current data to disk if the FileName property is set, and saving the current data packet to a cache so that it can be restored when the client dataset is reopened.

4Triggers an AfterClose event.

Note: Calling the Open method sets Active to true; calling the Close method sets Active to false.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
```

```
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
value of the first field for the last record to the server so
that the server can return the correct records. The complete
project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
var
  LastValue: OleVariant;
  CDSClone: TClientDataSet;
begin
  if ClientDataSet1.Active then
  begin
    CDSClone := TClientDataSet.Create(Form1);
    try
      CDSClone.CloneCursor(ClientDataSet1, True);
      { turn off FetchOnDemand so that the clone only fetches
        the last LOCAL record }
      CDSClone.FetchOnDemand := False;
      CDSClone.Last;
      LastValue := CDSClone.Fields[0].AsString;
      CDSClone.Close;
    finally
      CDSClone.Free;
    end;
  end else
    LastValue := NULL;
  OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
  ClientDataSet1.GetNextPacket;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
```

```
    RemoteServer1.Connected := False;
    with RemoteServer1 do
    begin
      for I := 0 to DataSetCount - 1 do
        DataSets[I].DisableControls;
    end;
  end;

  procedure TForm1.CheckButtonActive(DataSet: TDataSet);
  begin
    Button1.Enabled := not ClientDataSet1.Active;
    Button2.Enabled := ClientDataSet1.Active;
    Button3.Enabled := ClientDataSet1.Active;
  end;
```

See Also

FileName (see page 8475)

Open (see page 6976)

Close (see page 6920)

State (see page 7036)

AfterOpen (see page 7045)

BeforeOpen (see page 7051)

FieldDefs (see page 7019)

Fields (see page 7023)

ProviderName (see page 8492)

134.1.2.5.2 TSimpleDataSet.Aggregates Property

Lists all aggregates that apply to the client dataset.

Pascal

```
property Aggregates: TAggregates;
```

C++

```
__property TAggregates Aggregates;
```

Description

Use Aggregates to define aggregates that summarize the data in the client dataset. Aggregates is a collection of TAggregate objects, each of which defines a formula for calculating an aggregate value from a group of records in the client dataset. The individual aggregates can summarize all the records in the client dataset or subgroups of records that have the same value on a set of fields. Aggregates that summarize a subgroup of records are associated with indexes, and can only be used when the associated index is current.

Do not use Aggregates to obtain summarized values maintained by the client dataset. Aggregates can contain TAggregate objects that are not currently active. Instead, use the ActiveAggs property, which represents only those TAggregate objects that are currently active.

Note: You do not need to add a TAggregate object to the Aggregates collection when you add an aggregate field using the Fields editor. The Fields editor automatically adds TAggregate objects to Aggregates for every aggregate field you define.

See Also

ActiveAggs (see page 8460)

TAggregate (see page 8268)

IndexName (see page 8483)

TAggregateField (see page 6743)

Obtaining Aggregate Values

Specifying Aggregates

Defining an Aggregate Field

134.1.2.5.3 TSimpleDataSet.AggregatesActive Property

Specifies whether the client dataset calculates and maintains aggregate values.

Pascal

```
property AggregatesActive: Boolean;
```

C++

```
__property Boolean AggregatesActive;
```

Description

Use AggregatesActive to indicate whether the client dataset should maintain aggregate values. When AggregatesActive is false (the default), the client dataset does not support maintained aggregates. This allows the client dataset to avoid the overhead of calculating values that the application is not using. When AggregatesActive is true, the client dataset calculates and maintains all aggregate values specified by the Aggregates property that are compatible with the current index.

When aggregates are maintained, the Value method of every active aggregate object returns a value that reflects the current data in the client dataset. When users edit the data in the dataset, these values are recalculated to reflect the user's changes.

Note: To selectively enable and disable aggregates rather than turning them all on or off at once, use the Active property of individual TAggregate objects. These objects are available through the Aggregates property.

See Also

ActiveAggs (see page 8460)

Aggregates (see page 8461)

IndexName (see page 8483)

Value (see page 8273)

Active (see page 8274)

134.1.2.5.4 TSimpleDataSet.AutoCalcFields Property

Determines when the OnCalcFields event is triggered and when lookup field values are calculated.

Pascal

```
property AutoCalcFields: Boolean;
```

C++

```
__property Boolean AutoCalcFields;
```

Description

Set AutoCalcFields to control when the OnCalcFields event is triggered to update calculated fields and when lookup fields are calculated.

A calculated field is one that derives its value from the values of one or more fields in the active record, sometimes with additional processing. Lookup fields are fields whose values come from a secondary dataset or lookup cache.

Note: Unidirectional datasets support calculated fields but not unidirectional fields. For unidirectional datasets, OnCalcFields only controls when calculated fields are updated.

When AutoCalcFields is true (the default), Lookup fields are recalculated and OnCalcFields is triggered when:

The dataset is opened.

The dataset is put into dsEdit state.

Focus moves from one visual control to another, or from one column to another in a data-aware grid and modifications have been made to the record.

When AutoCalcFields is false, Lookup fields are recalculated and the OnCalcFields event occurs only when

The dataset is opened.

The dataset is put into dsEdit state.

A record is retrieved from a database.

If an application permits users to change data, OnCalcFields is frequently triggered. In these cases an application may set AutoCalcFields to false to reduce the frequency with which the OnCalcFields event occurs and with which lookup values are fetched.

See Also

OnCalcFields (see page 7053)

Calculating Fields

134.1.2.5.5 TSimpleDataSet.Connection Property

Identifies the TSQLConnection component that connects the dataset to a database server.

Pascal

```
property Connection: TSQLConnection;
```

C++

```
__property TSQLConnection Connection;
```

Description

Use Connection to specify the component that connects this dataset to a database server. TSimpleDataSet (see page 19555) can use either an external TSQLConnection component or, if you set the ConnectionName property instead, an internally-created component. By using an external TSQLConnection component, you can let multiple datasets share a single database connection.

See Also

Setting Up a Simple Dataset

Connecting to a Data Store Using TADOConnection

134.1.2.5.6 TSimpleDataSet.Constraints Property

Specifies record-level constraints that must be met when editing the data.

Pascal

```
property Constraints: TCheckConstraints;
```

C++

```
__property TCheckConstraints Constraints;
```

Description

Use Constraints to read or add record-level constraints to the dataset. Record-level constraints usually impose relationships between the fields in a single record. Constraints that limit the value of a single field should be added to the individual field components.

See Also

TCheckConstraint (see page 6841)

CustomConstraint (see page 7156)

ImportedConstraint (see page 7167)

Constraining Data Values

134.1.2.5.7 TSimpleDataSet.DataSet Property

Provides access to the internal dataset that fetches data from the server.

Pascal

```
property DataSet: TInternalSQLDataSet;
```

C++

```
__property TInternalSQLDataSet DataSet;
```

Description

Read DataSet to access the internal dataset that TSimpleDataSet (see page 19555) uses to fetch data from the server. TSimpleDataSet (see page 19555) populates its Data property by fetching data from this internal dataset using an internal provider component.

See Also

Data (see page 8467)

134.1.2.5.8 TSimpleDataSet.DisableStringTrim Property

Specifies whether trailing spaces are trimmed from field values when records are posted.

Pascal

```
property DisableStringTrim: Boolean;
```

C++

```
__property Boolean DisableStringTrim;
```

Description

When DisableStringTrim is false, trailing spaces are trimmed from field values when records are posted. When DisableStringTrim is true, field values are posted exactly as typed, including any trailing spaces.

134.1.2.5.9 TSimpleDataSet.FetchOnDemand Property

Indicates whether to fetch data packets from the provider on an as-needed basis.

Pascal

```
property FetchOnDemand: Boolean;
```

C++

```
__property Boolean FetchOnDemand;
```

Description

By default FetchOnDemand is true, meaning that the client dataset fetches additional packets of data as needed (for example, as a user scrolls through data, or conducts a search). If the provider does not automatically include BLOB fields or nested detail sets in data packets, setting FetchOnDemand to true causes the client dataset to fetch these values when needed as well.

Set FetchOnDemand to false only if your application contains code that explicitly fetches data packets as needed using the GetNextPacket method. Similarly, if FetchOnDemand is false and the provider does not include BLOB data by default (Options includes poFetchBlobsOnDemand), the application must fetch BLOB fields explicitly using the FetchBlobs method. If the provider does not include nested detail sets (Options includes poFetchDetailsOnDemand) and FetchOnDemand is false, the application must fetch nested details explicitly using the FetchDetails method.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
value of the first field for the last record to the server so
that the server can return the correct records. The complete
project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
var
  LastValue: OleVariant;
  CDSClone: TClientDataSet;
begin
  if ClientDataSet1.Active then
  begin
    CDSClone := TClientDataSet.Create(Form1);
    try
      CDSClone.CloneCursor(ClientDataSet1, True);
      { turn off FetchOnDemand so that the clone only fetches
        the last LOCAL record }
      CDSClone.FetchOnDemand := False;
      CDSClone.Last;
      LastValue := CDSClone.Fields[0].AsString;
      CDSClone.Close;
    finally
      CDSClone.Free;
    end;
  end else
    LastValue := NULL;
  OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
  ClientDataSet1.GetNextPacket;
end;
```

See Also

GetNextPacket (see page 8423)

Options (see page 18440)

FetchBlobs (see page 8414)

FetchDetails (see page 8414)

Requesting Data from the Source Dataset or Document

134.1.2.5.10 TSimpleDataSet.FieldDefs Property

Points to the list of field definitions for the dataset.

Pascal

```
property FieldDefs: TFieldDefs;
```

C++

```
__property TFieldDefs FieldDefs;
```

Description

FieldDefs lists the field definitions for a dataset. While an application can examine FieldDefs to explore the field definitions for a dataset, it should not change these definitions unless creating a new table with CreateTable or CreateDataSet.

To access fields and field values in a dataset, use the Fields, AggFields, and FieldValues properties, and the FieldsByName method.

Note: If the dataset includes object field descendants, FieldDefs represents a hierarchical view of the data, meaning that the definitions include object field definitions. To determine the definitions in a flattened view, use FieldDefList instead.

C++ Examples:

```
/*
The following code creates and activates a client dataset in
the form's OnCreate event handler.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TFieldDefs *pDefs = CDS2->FieldDefs;
  TFieldDef *pDef = pDefs->AddFieldDef();
  pDef->DataType = ftInteger;
  pDef->Name = "Field1";

  pDef = pDefs->AddFieldDef();
  pDef->DataType = ftString;
  pDef->Size = 10;
  pDef->Name = "Field2";

  TIndexDef *pIDef = CDS2->IndexDefs->AddIndexDef();
  pIDef->Fields = "Field1";
  pIDef->Name = "IntIndex";

  CDS2->CreateDataSet();
}
```

Delphi Examples:

```
{
The following code creates and activates a client dataset in
the form's OnCreate event handler.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  with CDS2 do
  begin
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftInteger;
      Name := 'Field1';
    end;
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftString;
      Size := 10;
      Name := 'Field2';
    end;
    with IndexDefs.AddIndexDef do
    begin
      Fields := 'Field1';
      Name := 'IntIndex';
    end;
    CreateDataSet;
  end;
end;
{ The following example shows how to create a table. }
procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Close;
  Table1.DeleteTable;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
    else
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
```

```
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.Active:= True;
  end;
```

See Also

CreateTable (see page 10189)

Fields (see page 7023)

FieldValues (see page 7025)

ClearFields (see page 6919)

GetFieldList (see page 6953)

GetFieldNames (see page 6954)

FieldByName (see page 6936)

FindField (see page 6938)

SetFields (see page 6997)

ObjectView (see page 7031)

AggFields (see page 7006)

CreateDataSet (see page 8401)

Creating and Deleting Tables

134.1.2.5.11 TSimpleDataSet.FileName Property

Specifies a file that stores the data of the client dataset.

Pascal

```
property FileName: string;
```

C++

```
__property AnsiString FileName;
```

Description

Set FileName if the client dataset always reads its data from and writes its data to a single file. When FileName represents a valid file name, the client dataset automatically writes its data to that file when it is closed. When FileName specifies the name of an existing file, the client dataset automatically reads its data from that file when it is opened.

Changes are not merged into the data when it is saved to the file. This allows an application that employs the "briefcase" model to set FileName without losing updates that are destined for a database server. Single-tiered applications should explicitly merge changes before closing the dataset and saving the data to disk.

Note: When client datasets write their data to a file, they overwrite the contents of the file. Any previous contents of the file are lost.

To read from or write to a different file, use the LoadFromFile and SaveToFile methods instead. These methods can also be used to load and save data at times other than when the Active property changes.

See Also

Open (see page 6976)

Close (see page 6920)

Active (see page 8458)

LoadFromFile (see page 8433)

SaveToFile (see page 8443)

Loading Data from a File or Stream

Saving Data to a File or Stream

Using a Dedicated File on Disk

134.1.2.5.12 TSimpleDataSet.Filter Property

Specifies the text of the current filter.

Pascal

```
property Filter: string;
```

C++

```
__property AnsiString Filter;
```

Description

Use Filter to specify a dataset filter. When filtering is applied to a client dataset, only those records that meet a filter's conditions are available to an application. Filter contains the string that describes the filter condition. Client datasets support the following set of filter operations: See the Limiting what records appear link at the bottom of this topic for a complete list of available filter operations.

Operator or function ComparisonsExampleComment

Operator or function Comparisons	Example	Comment
=	State = 'CA'	
<>	State <> 'CA'	
>=	DateEntered >= '1/1/1998'	
<=	Total <= 100,000	
>	Percentile > 50	
<	Field1 < Field2	
BLANK	State <> 'CA' or State = BLANK	Blank records do not appear unless explicitly included in the filter.
IS NULL	Field1 IS NULL	
IS NOT NULL	Field1 IS NOT NULL	
Logical operators		
and	State = 'CA' and Country = 'US'	
or	State = 'CA' or State = 'MA'	
not	not (State = 'CA')	
Arithmetic operators		
+	Total + 5 > 100	Applies to numbers, strings, or date (time) + number
-	Field1 - 7 <> 10	Applies to numbers, dates, or date (time) - number
*	Discount * 100 > 20	Applies to numbers only
/	Discount > Total / 5	Applies to numbers only
String functions		
Upper	Upper(Field1) = 'ALWAYS'	
Lower	Lower(Field1 + Field2) = 'josp'	
Substring	Substring(DateString,8) = '1998' | Substring(DateString,1,3) = 'JAN'	Value goes from position of second argument to end or number of chars in third arg. First char has position 1.
Trim	Trim(Field1 + Field2) | Trim(Field1, '-')	removes third argument from front and back. If no third argument, trims spaces.
TrimLeft	TrimLeft(StringField) | TrimLeft(Field1, '$') <> ''	See Trim
TrimRight	TrimRight(StringField) | TrimRight(Field1, '.') <> ''	See Trim
DateTime functions		
Year	Year(DateField) = 2000	
Month	Month(DateField) <> 12	
Day	Day(DateField) = 1	
Hour	Hour(DateField) < 16	
Minute	Minute(DateField) = 0	

Second	Second(DateField) = 30	
GetDate	GetDate - DateField > 7	Represents current date and time
Date	DateField = Date(GetDate)	Returns the date portion of a datetime value
Time	TimeField > Time(GetDate)	Returns the time portion of a datetime value
Miscellaneous		
Like	Memo LIKE '%filters%'	Works like SQL-92 without the ESC clause. When applied to BLOB fields, FilterOptions determines whether case is considered.
In	Day(DateField) in (1,7)	Works like SQL-92. Second argument is a list of values all with the same type.
*	State = 'M*'	Wildcard for partial comparisons

See Also

Filtered (see page 7027)

FilterOptions (see page 7027)

Limiting Which Records Appear

FindFirst (see page 6939)

FindLast (see page 6940)

FindNext (see page 6940)

FindPrior (see page 6941)

OnFilterRecord (see page 7055)

Filter (see page 7026)

134.1.2.5.13 TSimpleDataSet.Filtered Property

Specifies whether or not filtering is active for a dataset.

Pascal

```
property Filtered: Boolean;
```

C++

```
__property Boolean Filtered;
```

Description

Check Filtered to determine whether or not dataset filtering is in effect. If Filtered is true, then filtering is active. To apply filter conditions specified in the Filter property or the OnFilterRecord event handler, set Filtered to true. Because unidirectional datasets do not support filters, when you set Filtered to true on a unidirectional dataset, it raises an exception.

Note: When filtering is enabled, user edits to a record may mean that the record no longer meets a filter's test condition. The next time an attempt is made to the read the record from the dataset while the filter is in effect, the record is not visible. If that happens, the next record that passes the filter condition becomes the active record.

See Also

Filter (see page 7026)

FilterOptions (see page 7027)

OnFilterRecord (see page 7055)

Locate (see page 6970)

Lookup (see page 6970)

134.1.2.5.14 TSimpleDataSet.FilterOptions Property

Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.

Pascal

```
property FilterOptions: TFilterOptions;
```

C++

```
__property TFilterOptions FilterOptions;
```

Description

Set FilterOptions to specify whether or not filtering is case insensitive when filtering on string or character fields, and whether or not partial comparisons for matching filter conditions is allowed.

Note: Because unidirectional datasets do not support filters, when you set the FilterOptions property of a unidirectional dataset, it raises an exception.

By default, FilterOptions is set to an empty set. For filters based on string fields, set FilterOptions to foCaseInsensitive to catch all variations on a string regardless of capitalization.

When a string in a filter ends with an asterisk (*), it can be used to match partial strings. To disable matching of partial strings and to treat the asterisk as a literal character in string comparisons, set FilterOptions to include foNoPartialCompare.

See Also

Filter (see page 7026)

Filtered (see page 7027)

OnFilterRecord (see page 7055)

Locate (see page 6970)

Lookup (see page 6970)

134.1.2.5.15 TSimpleDataSet.IndexDefs Property

Contains information about the indexes for a client dataset.

Pascal

```
property IndexDefs: TIndexDefs;
```

C++

```
__property TIndexDefs IndexDefs;
```

Description

Examine IndexDefs for index information. IndexDefs maintains an array of TIndexDef items, each of which describes an available index for the dataset. Set the value of IndexDefs before calling CreateDataSet to create a set of indexes with the client dataset.

Note: The index items in IndexDefs may not always reflect the current indexes available for a dataset. Before examining IndexDefs, call the IndexDefs.Update (Delphi) or IndexDefs->Update (C++) method to refresh the item list.

C++ Examples:

```
/*
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
*/
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2->Active = False;
  // Get the current available indices
  CDS2->IndexDefs->Update();
  // Find a field named "Field2"
  for (int I = 0; I < CDS2->IndexDefs->Count; I++)
    if (CDS2->IndexDefs->Items[I]->Fields == "Field2")
    {
      // set that index as the current index for the dataset}
      CDS2->IndexName = CDS2->IndexDefs->Items[I]->Name;
    }
  CDS2->Active = True;
}
```

Delphi Examples:

```
{
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
}
procedure TForm1.Button3Click(Sender: TObject);
var
  I : Integer;
begin
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2.Active := False;
  { Get the current available indices }
  CDS2.IndexDefs.Update;
  { Find a field named 'Field2' }
  for I := 0 to CDS2.IndexDefs.Count - 1 do
    if CDS2.IndexDefs.Items[I].Fields = 'Field2' then
    begin
      { set that index as the current index for the dataset}
      CDS2.IndexName := CDS2.IndexDefs.Items[I].Name;
    end;
  CDS2.Active := True;
end;
```

See Also

IndexFieldNames (see page 8482)

TIndexDef (see page 7283)

CreateDataSet (see page 8401)

Adding a New Index

134.1.2.5.16 TSimpleDataSet.IndexFieldNames Property

Lists the fields to use as an index.

Pascal

```
property IndexFieldNames: Widestring;
```

C++

```
__property BSTR IndexFieldNames;
```

Description

Use IndexFieldNames as an alternative method of specifying the index to use for a client dataset. Specify the name of each field on which to index the dataset, separating names with semicolons. Ordering of field names is significant.

Note: Indexes added using IndexFieldNames do not support grouping or maintained aggregates.

Tip: Use IndexFieldNames to create sort orders on the fly at runtime.

Note: The IndexFieldNames and IndexName properties are mutually exclusive. Setting one clears the other.

See Also

IndexDefs (see page 8480)

IndexName (see page 8483)

Creating an Index with IndexFieldNames

Adding a New Index

134.1.2.5.17 TSimpleDataSet.IndexName Property

Identifies an index for the client dataset.

Pascal

```
property IndexName: Widestring;
```

C++

```
__property BSTR IndexName;
```

Description

Use IndexName to specify an alternative index for a client dataset.If IndexName is empty, the dataset's sort order is based on the IndexFieldNames property or on its default ordering in the data packet. Default ordering is determined by the predefined index, DEFAULT_ORDER.

If IndexName contains a valid index name, then that index is used to determine sort order of records.

Note: IndexFieldNames and IndexName are mutually exclusive. Setting one clears the other.

C++ Examples:

```
/*
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
*/
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2->Active = False;
```

```
    // Get the current available indices
    CDS2->IndexDefs->Update();
    // Find a field named "Field2"
    for (int I = 0; I < CDS2->IndexDefs->Count; I++)
      if (CDS2->IndexDefs->Items[I]->Fields == "Field2")
      {
        // set that index as the current index for the dataset}
        CDS2->IndexName = CDS2->IndexDefs->Items[I]->Name;
      }
    CDS2->Active = True;
  }
```

Delphi Examples:

```
{
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
}
procedure TForm1.Button3Click(Sender: TObject);
var
  I : Integer;
begin
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2.Active := False;
  { Get the current available indices }
  CDS2.IndexDefs.Update;
  { Find a field named 'Field2' }
  for I := 0 to CDS2.IndexDefs.Count - 1 do
    if CDS2.IndexDefs.Items[I].Fields = 'Field2' then
    begin
      { set that index as the current index for the dataset}
      CDS2.IndexName := CDS2.IndexDefs.Items[I].Name;
    end;
  CDS2.Active := True;
end;
```

See Also

IndexFieldNames (see page 8482)

Deleting and Switching Indexes

134.1.2.5.18 TSimpleDataSet.MasterFields Property

Names one or more fields in a master table to link with corresponding fields in this dataset in order to establish a master-detail relationship.

Pascal

```
property MasterFields: Widestring;
```

C++

```
__property BSTR MasterFields;
```

Description

Use MasterFields after setting the MasterSource property to specify the names of one or more fields to use in establishing a detail-master relationship between this dataset and the one specified in MasterSource.

MasterFields is a string containing one or more field names in the master table. Separate field names with semicolons.

Each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this dataset for display.

Note: At design time, use the Field Link designer to establish the master-detail relationship between two datasets.

See Also

MasterSource (see page 8486)

Making the Table a Detail of Another Dataset

134.1.2.5.19 TSimpleDataSet.MasterSource Property

Specifies a data source component for the master dataset when establishing a detail-master relationship between this dataset and another one.

Pascal

```
property MasterSource: TDataSource;
```

C++

```
__property TDataSource MasterSource;
```

Description

Use MasterSource to specify the name of the data source component whose DataSet property identifies a dataset to use as a master table in establishing a detail-master relationship between this dataset and another one.

At design time choose an available data source from the MasterSource property's drop-down list in the Object Inspector.

After setting the MasterSource property, specify which fields to use in the master table by setting the MasterFields property. At runtime each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this dataset for display.

Note: At design time, use the Field Link designer to establish the master-detail relationship between two datasets.

Tip: MasterSource establishes a master/detail relationship using the traditional linked cursor approach. Alternately, client datasets can participate in master/detail relationships using nested datasets, where the detail dataset is stored with the master table's data. To assign a client dataset's data as the value of a nested detail, use the DataSetField property.

See Also

MasterFields (see page 8486)

Making the Table a Detail of Another Dataset

134.1.2.5.20 TSimpleDataSet.ObjectView Property

Specifies whether fields are stored hierarchically or flattened out in the Fields property.

Pascal

```
property ObjectView: Boolean;
```

C++

```
__property Boolean ObjectView;
```

Description

ObjectView affects the way the Fields property stores object fields and the way data-aware grids display ADT and array fields.

When ObjectView is true, the fields are stored hierarchically in the Fields property, meaning any child fields of an object field are referenced by the object field and don't appear sequentially after the object field in the TFields:: .Fields array. When ObjectView is false, the fields are stored sequentially in the Fields property, meaning any child fields of an object field are stored after the object field as siblings in the Fields array.

When ObjectView is false, object field types, such as TADTField, are not created. This switch is provided for increased compatibility with older data-aware controls, which may not handle object field types properly. The default is false for TBDEDataSet, and true for TClientDataSet.

Note: If the ObjectView property of the dataset provided to a TClientDataSet is false, then the TClientDataSet's ObjectView property always acts as false, since the data received is already flattened.

See Also

FieldDefList (see page 7019)

Fields (see page 7023)

Displaying ADT and Array Fields

Using Nested Detail Tables

Using Object Fields

Working with ADT Fields

Working with Array Fields

134.1.2.5.21 TSimpleDataSet.PacketRecords Property

Indicates the number or type of records in a single data packet.

Pascal

```
property PacketRecords: Integer;
```

C++

```
__property int PacketRecords;
```

Description

Use PacketRecords to specify how many records should be contained in a single packet fetched from a provider, or to indicate that the packet returned should contain only metadata.

When a client dataset is instantiated, PacketRecords is automatically set to -1, meaning that a single packet should contain all records in the dataset.

If PacketRecords is greater than zero, then it specifies the number of records to return in a packet.

To retrieve the metadata for a dataset, set PacketRecords to 0. When PacketRecords is zero, the provider returns only information from its dataset that defines the database's structure, such as table, column, constraint, and domain definitions.

Warning: When PacketRecords is greater than 0, the client dataset must fetch its data incrementally. If the provider is on a stateless application server, the client dataset must inform the provider where to start fetching data. This communication can occur using a BeforeGetRecords event handler.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
```

```
  value of the first field for the last record to the server so
  that the server can return the correct records. The complete
  project using this code example is in
  Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
  }
  procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
  var
    LastValue: OleVariant;
    CDSClone: TClientDataSet;
  begin
    if ClientDataSet1.Active then
    begin
      CDSClone := TClientDataSet.Create(Form1);
      try
        CDSClone.CloneCursor(ClientDataSet1, True);
        { turn off FetchOnDemand so that the clone only fetches
          the last LOCAL record }
        CDSClone.FetchOnDemand := False;
        CDSClone.Last;
        LastValue := CDSClone.Fields[0].AsString;
        CDSClone.Close;
      finally
        CDSClone.Free;
      end;
    end else
      LastValue := NULL;
    OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
    ClientDataSet1.GetNextPacket;
  end;
```

See Also

FetchOnDemand (see page 8471)

GetNextPacket (see page 8423)

BeforeGetRecords (see page 8515)

Requesting Data from the Source Dataset or Document

Using the BDE to Cache Updates

Updating Data in dbExpress Applications

134.1.2.5.22 TSimpleDataSet.Params Property

Contains parameter values that are sent to the provider.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Use Params to specify parameter values that the provider should pass to a source dataset. If the client dataset uses an internal provider, Params represents the values for any parameters used by the SQL statement specified by the CommandText property. If the client dataset uses an external provider that is associated with a dataset that represents a query or stored procedure, these

parameter values supply input parameter values for that query or stored procedure and return any output parameters of a stored procedure. If the provider is associated with a TTable or TSQLTable component, Params limits the records that are sent in data packets. After specifying the parameters (at design time or runtime), they are sent to the provider automatically when the client dataset fetches data or executes a query or stored procedure using its Execute method.

To send parameter values to a query or stored procedure, for every parameter on the query or stored procedure for which the client dataset should send a value, add a parameter object (TParam) to Params. Assign each parameter object the same Name, DataType, and ParamType as the corresponding parameter on the query or stored procedure. These values are automatically encoded and sent to the provider when the client dataset fetches data.

To send parameter values to a TTable or TSQLTable, add a parameter object for each field in the table that is used to limit the values sent in data packets. Assign each parameter object the same Name and DataType as the corresponding field component on the table. These parameter values are automatically sent when the client dataset fetches data, and the records returned in data packets from the provider only contain records that match the assigned value on the corresponding fields. You can achieve the same affect by applying a filter to the field, but using Params is more efficient because the data need not be included in data packets.

Note: To initialize Params so that it reflects the parameters of a provider's associated query or stored procedure, call the FetchParams method.

Warning: The parameters are only applied to the provider when a query or stored procedure is run the first time the client dataset fetches records. To change parameter values and force a query or stored procedure to be rerun with new values, use the Execute method.

Delphi Examples:

```
{
This example fills in the parameters of a client dataset
from the entries of a list box.
}
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
  ListItem: string;
begin
  { Call FetchParams to ensure parameters reflect server metadata }
  ClientDataSet1.FetchParams;
  with ClientDataSet1.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
              Items[I].AsBoolean := True
            else
              Items[I].AsBoolean := False;
          end;
```

```
          ftFloat:
            Items[I].AsFloat := StrToFloat(ListItem);
          ftCurrency:
            Items[I].AsCurrency := StrToFloat(ListItem);
          ftBCD:
            Items[I].AsBCD := StrToCurr(ListItem);
          ftDate:
            Items[I].AsDate := StrToDate(ListItem);
          ftTime:
            Items[I].AsTime := StrToTime(ListItem);
          ftDateTime:
            Items[I].AsDateTime := StrToDateTime(ListItem);
        end;
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Here is a string');
    ListBox1.Items.Add('4');
    ListBox1.Items.Add('3456789');
    ListBox1.Items.Add('34');
    ListBox1.Items.Add('0');
    ListBox1.Items.Add('12.56');
    ListBox1.Items.Add('4.23');
    ListBox1.Items.Add('99.5');
    ListBox1.Items.Add('4/12/53');
    ListBox1.Items.Add('7:45:00');
    ListBox1.Items.Add('4/12/53 7:45:00');

  end;
```

See Also

Params (see page 10087)

Params (see page 10165)

Active (see page 8458)

FetchParams (see page 8415)

Params (see page 20197)

Parameters (see page 1220)

Getting Parameters from the Application Server

Limiting Records with Parameters

Overview of Using Cached Updates

Passing Parameters to the Source Dataset

Sending Query or Stored Procedure Parameters

Setting Up a Simple Dataset

134.1.2.5.23 TSimpleDataSet.ReadOnly Property

Specifies whether the client dataset is read-only for this application.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Use the ReadOnly property to prevent users from updating, inserting, or deleting data in the dataset. By default, ReadOnly is false, meaning users can potentially alter the dataset's data.

To guarantee that users cannot modify or add data to a dataset, set ReadOnly to true.

When ReadOnly is true, the dataset's CanModify property is false.

Note: Even if ReadOnly is false, users may not be able to modify or add data to a client dataset if it gets its data from a provider. The provider's Options property can limit whether data can be edited, and if so, what types of edits are allowed.

See Also

Active (see page 7004)

CanModify

Options (see page 18440)

134.1.2.5.24 TSimpleDataSet.StoreDefs Property

Indicates whether field and index definitions are saved with the client dataset.

Pascal

```
property StoreDefs: Boolean;
```

C++

```
__property Boolean StoreDefs;
```

Description

If StoreDefs is true, the client dataset's index and field definitions are stored with the client dataset when the form or data module that contains it is saved. Setting StoreDefs to true makes the CreateDataSet method into a one-step procedure that creates fields and indexes at runtime.

See Also

FieldDefs (see page 7019)

IndexDefs (see page 8480)

CreateDataSet (see page 8401)

134.1.2.6 TSimpleDataSet Events

The events of the TSimpleDataSet class are listed here.

Published Events

	Name	Description
	AfterApplyUpdates (see page 19598)	Occurs after the client dataset applies updates to the provider but before it attempts to reconcile update errors.
	AfterCancel (see page 19599)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 19601)	Occurs after an application closes a dataset.
	AfterDelete (see page 19602)	Occurs after an application deletes a record.

	AfterEdit (see page 19605)	Occurs after an application starts editing a record.
	AfterExecute (see page 19606)	Occurs after the client dataset receives the results from an execute command.
	AfterGetParams (see page 19606)	Occurs after the client dataset receives parameter values from the provider.
	AfterGetRecords (see page 19607)	Occurs after the client dataset fetches a data packet from the provider.
	AfterInsert (see page 19608)	Occurs after an application inserts a new record.
	AfterOpen (see page 19609)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 19610)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 19610)	Occurs after an application refreshes the data in the dataset.
	AfterRowRequest (see page 19611)	Occurs after the client dataset fetches new information about the current record.
	AfterScroll (see page 19612)	Occurs after an application scrolls from one record to another.
	BeforeApplyUpdates (see page 19612)	Occurs before the client dataset applies updates to the remote server.
	BeforeCancel (see page 19613)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 19613)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 19614)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 19614)	Occurs before an application enters edit mode for the active record.
	BeforeExecute (see page 19615)	Occurs before the client dataset executes the provider's query or stored procedure.
	BeforeGetParams (see page 19616)	Occurs before the client dataset fetches parameter values from the provider.
	BeforeGetRecords (see page 19616)	Occurs before the client dataset fetches a data packet from the provider.
	BeforeInsert (see page 19618)	Occurs before an application enters insert mode.
	BeforeOpen (see page 19619)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 19620)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 19620)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeRowRequest (see page 19621)	Occurs before the client dataset fetches new information about the current record.
	BeforeScroll (see page 19622)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 19622)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 19623)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 19623)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 19624)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 19625)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 19625)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnReconcileError (see page 19625)	Occurs when a client dataset needs to reconcile an update to a record that could not be applied.

134.1.2.6.1 TSimpleDataSet.AfterApplyUpdates Event

Occurs after the client dataset applies updates to the provider but before it attempts to reconcile update errors.

Pascal

```
property AfterApplyUpdates: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterApplyUpdates;
```

Description

Write an AfterApplyUpdates event handler to respond to custom information returned to the client dataset from a provider's AfterApplyUpdates event handler. AfterApplyUpdates is part of the mechanism by which a client dataset and a provider communicate information about updates. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's ApplyUpdates method, the following events occur:

1.The client dataset receives a BeforeApplyUpdates event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeApplyUpdates event, where it can respond to or change that information before it applies the updates.

3.The provider applies the updates, and receives an OnUpdateError event for any problem records.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider in its AfterApplyUpdates event handler.

6.The client dataset generates an OnReconcileError for every record that could not be applied by the provider.

See Also

ApplyUpdates (see page 8394)

AS_ApplyUpdates (see page 16451)

applyupdates (see page 18459)

AfterApplyUpdates (see page 18475)

BeforeApplyUpdates (see page 8511)

OnReconcileError (see page 8524)

Applying Updates

Communicating with the Client Dataset

134.1.2.6.2 TSimpleDataSet.AfterCancel Event

Occurs after an application completes a request to cancel modifications to the active record.

Pascal

```
property AfterCancel: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterCancel;
```

Description

Write an AfterCancel event handler to take specific action after an application cancels changes to the active record. AfterCancel is called by the Cancel method after it updates the current position, releases the lock on the active record if necessary, and sets the dataset state to dsBrowse. If an application requires additional processing before returning control to a user after a Cancel event, code it in the AfterCancel event.

Delphi Examples:

```
{
This example updates the form’s status bar with a message
when an AfterCancel event occurs.  Set the StatusBar
SimplePanel property to True.
}
procedure TForm1.Table1AfterCancel(DataSet: TDataSet);
begin
  StatusBar1.SimpleText := 'Record changes cancelled for ' + DataSet.Name;
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Table1.Cancel;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  Table1.Post
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Name := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
    begin
        Table1.Close;
        Table1.DeleteTable;
    end;
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
```

```
          Options := [ixCaseInsensitive];
        end;
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
      Table1.Active:= True;
      for i := 1 to 20 do
        Table1.AppendRecord([i*111, i*222]);
    end;
  end;
  DS2.DataSet:= Table1;
  DBGrid2.DataSource.DataSet:= Table1;
  Table1.AfterCancel:= Table1AfterCancel;
  Table1.Active:= True;
end;
```

See Also

BeforeCancel (see page 7048)

Cancel (see page 6916)

State (see page 7036)

134.1.2.6.3 TSimpleDataSet.AfterClose Event

Occurs after an application closes a dataset.

Pascal

```
property AfterClose: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterClose;
```

Description

Write an AfterClose event handler to take specific action immediately after an application closes a dataset. For example, as a security measure, an application might clear a PASSWORD entry from the Params property of a TSQLConnection component when the dataset is closed.

AfterClose is called after a dataset is closed and the dataset state is set to dsInactive.

Delphi Examples:

```
{
This example adds an entry to a memo with a message when an
AfterOpen event or an AfterClose event occurs.  Populate the
ClientDataSet events with CheckButtonActive.
}
procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  if (Button1.Enabled) then
    Memo2.Lines.Add('Now that the dataset is open, record information is available.')
  else
    Memo2.Lines.Add('Now that the dataset is closed, record information is not available.');
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
  RemoteServer1.Connected := False;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].DisableControls;
  end;
end;

procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
```

See Also

Active (see page 7004)

State (see page 7036)

BeforeClose (see page 7048)

Close (see page 6920)

Params (see page 20253)

Opening and Closing Datasets

134.1.2.6.4 TSimpleDataSet.AfterDelete Event

Occurs after an application deletes a record.

Pascal

```
property AfterDelete: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterDelete;
```

Description

Write an AfterDelete event handler to take specific action immediately after an application deletes the active record in a dataset. AfterDelete is called by Delete after it deletes the record, sets the dataset state to dsBrowse, and repositions the current record.

C++ Examples:

```
/*
This example displays a message on the form's status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
*/
void __fastcall TForm1::Table1AfterDelete(TDataSet *DataSet)
{
  StatusBar1->SimpleText = Format(
    "There are now %d records in the table",
    ARRAYOFCONST(((int)DataSet->RecordCount)));
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Table1->Delete();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Table1 = new TTable(Form1);
  Table1->Active = false; // The Table component must not be active
  Table1->DatabaseName = "DBDEMOS";
  Table1->TableType = ttParadox;
  Table1->TableName = "CustInfo";
  Table1->Active = False;
  if (!Table1->Exists) // Don't overwrite an existing table
  {
    Table1->Close();
    Table1->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Table1->FieldDefs->Clear();
  TFieldDef *pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Table1->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Table1->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Table1->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Table1->CreateTable();
  Table1->Active = True;
  for (int i = 1; i <= 20; i++)
    Table1->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  DS2->DataSet = Table1;
  DBGrid2->DataSource->DataSet = Table1;
  Table1->AfterDelete = Table1AfterDelete;
  Table1->Active = True;
}
```

Delphi Examples:

```
{
This example displays a message on the form's status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
}
procedure TForm1.Table1AfterDelete(DataSet: TDataSet);
```

```
begin
  StatusBar1.SimpleText := Format('There are now %d records in the table',
[DataSet.RecordCount]);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Delete;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Name := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
    begin
        Table1.Close;
        Table1.DeleteTable;
    end;
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
```

```
        end;
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
      Table1.Active:= True;
      for i := 1 to 20 do
        Table1.AppendRecord([i*111, i*222]);
    end;
  end;
  DS2.DataSet:= Table1;
  DBGrid2.DataSource.DataSet:= Table1;
  Table1.AfterDelete:= Table1AfterDelete;
  Table1.Active:= True;
end;
```

See Also

State (see page 7036)

BeforeDelete (see page 7049)

Delete (see page 6924)

134.1.2.6.5 TSimpleDataSet.AfterEdit Event

Occurs after an application starts editing a record.

Pascal

```
property AfterEdit: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterEdit;
```

Description

Write an AfterEdit event handler to take specific action immediately after dataset enters edit mode. AfterEdit is called by Edit after it enables editing of a record, recalculates calculated fields, and calls the data event handler to process a record change.

Delphi Examples:

```
{
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

BeforeEdit (see page 7049)

Edit (see page 6932)

134.1.2.6.6 TSimpleDataSet.AfterExecute Event

Occurs after the client dataset receives the results from an execute command.

Pascal

```
property AfterExecute: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterExecute;
```

Description

Write an AfterExecute event handler to respond to custom information returned to the client dataset from a provider's AfterExecute event handler. AfterExecute is part of the mechanism by which a client dataset and a provider communicate information when executing an SQL command from the client. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's Execute method, the following events occur:

1.The client dataset receives a BeforeExecute event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeExecute event, where it can respond to or change that information before it executes the query or stored procedure.

3.The provider executes the appropriate command (either the command supplied by the CommandText property, or the command associated with a dataset such as a query or stored procedure).

4.The provider receives an AfterExecute event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeExecute event handler.

5.The client dataset unpacks any output parameters from the provider into the Params property and then receives an AfterExecute event, where it can respond to the custom information returned by the provider's AfterExecute event handler.

See Also

Execute (see page 8413)

AS_Execute (see page 16453)

Execute (see page 18466)

AfterExecute (see page 18476)

BeforeExecute (see page 8514)

CommandText (see page 8465)

Communicating with the Client Dataset

134.1.2.6.7 TSimpleDataSet.AfterGetParams Event

Occurs after the client dataset receives parameter values from the provider.

Pascal

```
property AfterGetParams: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterGetParams;
```

Description

Write an AfterGetParams event handler to respond to custom information returned to the client dataset from a provider's AfterGetParams event handler. AfterGetParams is part of the mechanism by which a client dataset and a provider communicate information when the client application calls FetchParams. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's FetchParams method, the following events occur:

1.The client dataset receives a BeforeGetParams event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeGetParams event, where it can respond to or change that information before it retrieves parameter values.

3.The provider encodes all of its dataset's current parameter values into an OleVariant.

4.The provider receives an AfterGetParams event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetParams event handler.

5.The client dataset unpacks the parameters from the provider into the Params property and then receives an AfterGetParams event, where it can respond to the custom information returned by the provider's AfterGetParams event handler.

See Also

FetchParams (see page 8415)

AS_GetParams (see page 16454)

GetParams (see page 18467)

aftergetparams (see page 18477)

BeforeGetParams (see page 8514)

Communicating with the Client Dataset

134.1.2.6.8 TSimpleDataSet.AfterGetRecords Event

Occurs after the client dataset fetches a data packet from the provider.

Pascal

```
property AfterGetRecords: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterGetRecords;
```

Description

Write an AfterGetRecords event handler to respond to custom information returned to the client dataset from a provider's AfterGetRecords event handler. AfterGetRecords is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches data from the application server, the following events occur:

1. The client dataset receives a BeforeGetRecords event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2. The provider receives a BeforeGetRecords event, where it can respond to or change that information before it creates a data packet.
3. The provider generates an OnGetData event where it can edit the data before creating a data packet, and then generates the data packet.
4. The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.
5. The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider's AfterGetRecords event handler.

See Also

AS_GetRecords (see page 16455)

GetRecords (see page 18468)

AfterGetRecords (see page 18477)

BeforeGetRecords (see page 8515)

Communicating with the Client Dataset

134.1.2.6.9 TSimpleDataSet.AfterInsert Event

Occurs after an application inserts a new record.

Pascal

```
property AfterInsert: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterInsert;
```

Description

Write an AfterInsert event handler to take specific action immediately after an application inserts a record. The Insert and Append methods generate an AfterInsert event after inserting or appending a new record.

Delphi Examples:

```
{
This example updates a memo with a message when an
AfterInsert event or a BeforeInsert event occurs.
}
procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After inserting a record');
end;

procedure TForm1.ClientDataSet1BeforeInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before inserting a record');
end;
```

See Also

BeforeInsert (see page 7050)

Append (see page 6910)

Insert (see page 6959)

134.1.2.6.10 TSimpleDataSet.AfterOpen Event

Occurs after an application completes opening a dataset and before any data access occurs.

Pascal

```
property AfterOpen: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterOpen;
```

Description

Write an AfterOpen event handler to take specific action immediately after an application opens the dataset. AfterOpen is called after the dataset establishes access to its data and the dataset is put into dsBrowse state. For example, an AfterOpen event handler might check an ini file to determine the last record touched in the dataset the previous time the application ran, and position the dataset at that record.

Delphi Examples:

```
{
This example adds an entry to a memo with a message when an
AfterOpen event or an AfterClose event occurs.  Populate the
ClientDataSet events with CheckButtonActive.
}
procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  if (Button1.Enabled) then
    Memo2.Lines.Add('Now that the dataset is open, record information is available.')
  else
    Memo2.Lines.Add('Now that the dataset is closed, record information is not available.');
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
  RemoteServer1.Connected := False;
  with RemoteServer1 do
  begin
```

```
    for I := 0 to DataSetCount - 1 do
      DataSets[I].DisableControls;
  end;
end;

procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
```

See Also

Active (see page 7004)

State (see page 7036)

BeforeOpen (see page 7051)

Open (see page 6976)

Opening and Closing Datasets

134.1.2.6.11 TSimpleDataSet.AfterPost Event

Occurs after an application writes the active record to the database or change log and returns to browse state.

Pascal

```
property AfterPost: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterPost;
```

Description

Write an AfterPost event handler to take specific action immediately after an application posts a change to the active record. AfterPost is called after a modification or insertion is made to a record.

Delphi Examples:

```
{
This example updates the form's status bar with a message
when an AfterPost event occurs.
}
procedure TForm1.ClientDataSet1AfterPost(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before posting a record');
end;
```

See Also

BeforePost (see page 7052)

Post (see page 6977)

134.1.2.6.12 TSimpleDataSet.AfterRefresh Event

Occurs after an application refreshes the data in the dataset.

Pascal

```
property AfterRefresh: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterRefresh;
```

Description

Write an AfterRefresh event handler to take specific action immediately after an application has updated the records in the dataset. AfterRefresh is generated by calls to the Refresh method.

See Also

BeforeRefresh (see page 7052)

Refresh (see page 6991)

134.1.2.6.13 TSimpleDataSet.AfterRowRequest Event

Occurs after the client dataset fetches new information about the current record.

Pascal

```
property AfterRowRequest: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterRowRequest;
```

Description

Write an AfterRowRequest event handler to respond to custom information returned to the client dataset from a provider's AfterRowRequest event handler. AfterRowRequest is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches detail information about the current record or refreshes the current record, the following events occur:

1.The client dataset receives a BeforeRowRequest event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeRowRequest event, where it can respond to or change that information before it looks up the requested information.

3.The provider creates a delta packet that contains the requested information.

4.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeRowRequest event handler.

5.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider's AfterRowRequest event handler.

See Also

AS_RowRequest (see page 16456)

RowRequest (see page 18472)

AfterRowRequest (see page 18478)

BeforeRowRequest (see page 8520)

FetchBlobs (see page 8414)

FetchParams (see page 8415)

RefreshRecord (see page 8442)

Communicating with the Client Dataset

134.1.2.6.14 TSimpleDataSet.AfterScroll Event

Occurs after an application scrolls from one record to another.

Pascal

```
property AfterScroll: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterScroll;
```

Description

Write an AfterScroll event handler to take specific action immediately after an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. AfterScroll is called after all other events triggered by these methods and any other methods that switch from record to record in the dataset.

See Also

BeforeScroll (see page 7053)

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Prior (see page 6979)

Navigating Datasets

134.1.2.6.15 TSimpleDataSet.BeforeApplyUpdates Event

Occurs before the client dataset applies updates to the remote server.

Pascal

```
property BeforeApplyUpdates: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeApplyUpdates;
```

Description

Write a BeforeApplyUpdates event handler to send custom information to the client dataset's provider. BeforeApplyUpdates is part of the mechanism by which a client dataset and a provider communicate information about updates. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's ApplyUpdates method, the following events occur:

1.The client dataset receives a BeforeApplyUpdates event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeApplyUpdates event, where it can respond to or change that information before it applies the updates.

3.The provider applies the updates and receives an OnUpdateError event for any problem records.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider's AfterApplyUpdates event handler.

See Also

ApplyUpdates (see page 8394)

AS_ApplyUpdates (see page 16451)

applyupdates (see page 18459)

BeforeApplyUpdates (see page 18479)

AfterApplyUpdates (see page 8497)

Applying Updates

Communicating with the Client Dataset

134.1.2.6.16 TSimpleDataSet.BeforeCancel Event

Occurs before an application executes a request to cancel changes to the active record.

Pascal

```
property BeforeCancel: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeCancel;
```

Description

Write a BeforeCancel event to take specific action before an application carries out a request to cancel changes. BeforeCancel is called by the Cancel method before it cancels a dataset operation such as Edit, Insert, or Delete.

An application might use the BeforeCancel event to record a user's changes in an undo buffer.

See Also

AfterCancel (see page 7038)

Delete (see page 6924)

Edit (see page 6932)

Insert (see page 6959)

134.1.2.6.17 TSimpleDataSet.BeforeClose Event

Occurs immediately before the dataset closes.

Pascal

```
property BeforeClose: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeClose;
```

Description

Write a BeforeClose event to take specific action before an application closes a dataset. Calling Close or setting the Active property to false results in a call to the BeforeClose event handler.

See Also

Active (see page 7004)

AfterClose (see page 7040)

Close (see page 6920)

Opening and Closing Datasets

Posting Data

134.1.2.6.18 TSimpleDataSet.BeforeDelete Event

Occurs before an application attempts to delete the active record.

Pascal

```
property BeforeDelete: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeDelete;
```

Description

Write a BeforeDelete event handler to take specific action before an application deletes the active record. BeforeDelete is called by Delete before it actually deletes a record.

In Delphi, making use of this event an application might, for example, display a dialog box asking for confirmation before deleting the record. On denial of confirmation, the application could abort the deletion by calling the Abort procedure.

See Also

AfterDelete (see page 7041)

Delete (see page 6924)

Abort (see page 22786)

134.1.2.6.19 TSimpleDataSet.BeforeEdit Event

Occurs before an application enters edit mode for the active record.

Pascal

```
property BeforeEdit: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeEdit;
```

Description

Write a BeforeEdit event handler to take specific action before an application enables editing of the active record. For example, an application that keeps a log of database edits could use the BeforeEdit event to record the edit request, time, and user before entering edit state.

Delphi Examples:

```
{
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
```

```
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

State (see page 7036)

AfterEdit (see page 7044)

Edit (see page 6932)

134.1.2.6.20 TSimpleDataSet.BeforeExecute Event

Occurs before the client dataset executes the provider's query or stored procedure.

Pascal

```
property BeforeExecute: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to send custom information to the client dataset's provider. BeforeExecute is part of the mechanism by which a client dataset and a provider communicate information when explicitly executing an SQL command by the provider's dataset. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's Execute method, the following events occur:

1.The client dataset receives a BeforeExecute event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeExecute event, where it can respond to or change that information before it executes the query or stored procedure.

3.The provider executes the appropriate command (either the command supplied by the CommandText property, or the command associated with a dataset such as a query or stored procedure).

4.The provider receives an AfterExecute event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeExecute event handler.

5.The client dataset unpacks any output parameters from the provider into the Params property and then receives an AfterExecute event, where it can respond to the custom information returned by the provider's AfterExecute event handler.

See Also

Execute (see page 8413)

AS_Execute (see page 16453)

Execute (see page 18466)

beforeexecute (see page 18479)

AfterExecute (see page 8504)

CommandText (see page 8465)

Communicating with the Client Dataset

134.1.2.6.21 TSimpleDataSet.BeforeGetParams Event

Occurs before the client dataset fetches parameter values from the provider.

Pascal

```
property BeforeGetParams: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeGetParams;
```

Description

Write a BeforeGetParams event handler to send custom information to the client dataset's provider. BeforeGetparams is part of the mechanism by which a client dataset and a provider communicate information when passing parameter values from the provider's dataset. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's FetchParams method, the following events occur:

1.The client dataset receives a BeforeGetParams event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeGetParams event, where it can respond to or change that information before it retrieves parameter values.

3.The provider encodes all of its dataset's current parameter values into an OleVariant.

4.The provider receives an AfterGetParams event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetParams event handler.

5.The client dataset unpacks the parameters from the provider into the Params property and then receives an AfterGetParams event, where it can respond to the custom information returned by the provider's AfterGetParams event handler.

See Also

FetchParams (see page 8415)

AS_GetParams (see page 16454)

GetParams (see page 18467)

BeforeGetParams (see page 18480)

AfterGetParams (see page 8505)

Communicating with the Client Dataset

134.1.2.6.22 TSimpleDataSet.BeforeGetRecords Event

Occurs before the client dataset fetches a data packet from the provider.

Pascal

```
property BeforeGetRecords: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeGetRecords;
```

Description

Write a BeforeGetRecords event handler to send custom information to the client dataset's provider. BeforeGetRecords is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches data from the application server, the following events occur:

1.The client dataset receives a BeforeGetRecords event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeGetRecords event, where it can respond to or change that information before it creates a data packet.

3.The provider generates the data packet.

4.The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.

5.The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider's AfterGetRecords event handler.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
value of the first field for the last record to the server so
that the server can return the correct records. The complete
project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
var
  LastValue: OleVariant;
  CDSClone: TClientDataSet;
begin
  if ClientDataSet1.Active then
  begin
    CDSClone := TClientDataSet.Create(Form1);
    try
      CDSClone.CloneCursor(ClientDataSet1, True);
      { turn off FetchOnDemand so that the clone only fetches
        the last LOCAL record }
      CDSClone.FetchOnDemand := False;
      CDSClone.Last;
      LastValue := CDSClone.Fields[0].AsString;
      CDSClone.Close;
    finally
      CDSClone.Free;
    end;
```

```
    end else
      LastValue := NULL;
    OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
    ClientDataSet1.GetNextPacket;
  end;
```

See Also

AS_GetRecords (see page 16455)

GetRecords (see page 18468)

BeforeGetRecords (see page 18480)

AfterGetRecords (see page 8506)

Communicating with the Client Dataset

134.1.2.6.23 TSimpleDataSet.BeforeInsert Event

Occurs before an application enters insert mode.

Pascal

```
property BeforeInsert: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeInsert;
```

Description

Write a BeforeInsert event handler to take specific action before an application inserts or appends a new record. The Insert or Append method generates a BeforeInsert method before it sets the dataset into dsInsert state.

Delphi Examples:

```
{
This example updates a memo with a message when an
AfterInsert event or a BeforeInsert event occurs.
}
procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After inserting a record');
end;

procedure TForm1.ClientDataSet1BeforeInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before inserting a record');
end;
{
This example uses the BeforeInsert event to do data
validation. If the StrToInt function raises an exception,
the edit control's contents are set to a valid value so
that the assignment to the INTEGER field in the dataset
will succeed.  Continuing after the exception causes a
'0' to be placed in the PORTA field.
DataSet design:
This example requires a ClientDataSet, a DBGrid, a
DataSource and a DBNavigator.  Name the ClientDataSet CDS.
Set the DataSet property in the DataSource to CDS and name
that DataSource DS.  Set the DataSource in the DBNavigator
```

```
to DS the DataSource in the DBGrid to DS.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
type to Integer.  The file used to load CDS must have an
attribute name of HOST with an attribute type of string and
an attribute name of PORTA with an attribute type of integer.
Add DB, DBCtrls, Grids, DBGrids and DBClient to the uses clause.
}
{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button3Click(Sender: TObject);
begin
  CDS.Insert;
  CDS.FieldByName('PORTA').AsInteger := StrToInt(Edit1.Text);
  CDS.Post;
end;

procedure TForm1.CDSBeforeInsert(DataSet: TDataSet);
begin
  try
  { Make sure edit field can be converted to integer --
   this will raise an exception if it can't }
    StrToInt(Edit1.Text);
  except
    Edit1.Text := '0';
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;
```

See Also

State (see page 7036)

AfterInsert (see page 7044)

Append (see page 6910)

Insert (see page 6959)

134.1.2.6.24 TSimpleDataSet.BeforeOpen Event

Occurs before an application executes a request to open a dataset.

Pascal

```
property BeforeOpen: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeOpen;
```

Description

Write a BeforeOpen event handler to take specific action before an application opens a dataset for viewing or editing. BeforeOpen is triggered when an application sets the Active property to true for a dataset or an application calls Open.

See Also

Active (see page 7004)

AfterOpen (see page 7045)

Open (see page 6976)

Opening and Closing Datasets

134.1.2.6.25 TSimpleDataSet.BeforePost Event

Occurs before an application posts changes for the active record to the database or change log.

Pascal

```
property BeforePost: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforePost;
```

Description

Write a BeforePost event handler to take specific action before an application posts dataset changes. BeforePost is triggered when an application calls the Post method. Post checks to make sure all required fields are present, then calls BeforePost before posting the record.

An application might use BeforePost to perform validity checks on data changes before committing them. If it encountered a validity problem, it could call Abort to cancel the Post operation (Delphi) or throw an exception (C++).

Delphi Examples:

```
{
This example checks for a valid entry in a TDBEdit control
and calls the Abort procedure if the control is empty; Abort
cancels the post before it happens.
}
procedure TForm1.ClientDataSet1BeforePost(DataSet: TDataSet);
begin
  if DBEdit1.Text = '' then
    Abort;
  DataSet.Fields[2].AsString := DBEdit1.Text;
end;
```

See Also

AfterPost (see page 7046)

Post (see page 6977)

Abort (see page 22786)

Working with Field Component Methods at Runtime

134.1.2.6.26 TSimpleDataSet.BeforeRefresh Event

Occurs immediately before an application refreshes the data in the dataset.

Pascal

```
property BeforeRefresh: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeRefresh;
```

Description

Write a BeforeRefresh event handler to take specific action immediately before an application updates the records in the dataset. BeforeRefresh is generated by calls to the Refresh method.

See Also

AfterRefresh (see page 7047)

Refresh (see page 6991)

134.1.2.6.27 TSimpleDataSet.BeforeRowRequest Event

Occurs before the client dataset fetches new information about the current record.

Pascal

```
property BeforeRowRequest: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeRowRequest;
```

Description

Write a BeforeRowRequest event handler to send custom information to the client dataset's provider. BeforeRowRequest is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches detail information about the current record or refreshes the current record, the following events occur:

1.The client dataset receives a BeforeRowRequest event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider on the application server receives a BeforeRowRequest event, where it can respond to or change that information before it looks up the requested information.

3.The provider creates a delta packet that contains the requested information.

4.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeRowRequest event handler.

5.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider's AfterRowRequest event handler.

See Also

AS_RowRequest (see page 16456)

RowRequest (see page 18472)

BeforeRowRequest (see page 18482)

AfterRowRequest (see page 8509)

FetchBlobs (see page 8414)

FetchParams (see page 8415)

RefreshRecord (see page 8442)

Communicating with the Client Dataset

134.1.2.6.28 TSimpleDataSet.BeforeScroll Event

Occurs before an application scrolls from one record to another.

Pascal

```
property BeforeScroll: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeScroll;
```

Description

Write a BeforeScroll event handler to take specific action immediately before an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. BeforeScroll is called before all other events triggered by these methods and any other methods that switch from record to record in the dataset.

See Also

AfterScroll (see page 7047)

First (see page 6942)

Last (see page 6968)

MoveBy (see page 6971)

Next (see page 6973)

Prior (see page 6979)

Navigating Datasets

134.1.2.6.29 TSimpleDataSet.OnCalcFields Event

Occurs when an application recalculates calculated fields.

Pascal

```
property OnCalcFields: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent OnCalcFields;
```

Description

Write an OnCalcFields event handler to take specific action when an application recalculates calculated fields. A calculated field is one that derives its value from the values in one or more fields in the dataset, sometimes with additional processing.

OnCalcFields is triggered when:

A dataset is opened.

A dataset is put into dsEdit state.

A record is retrieved from a database.

When the AutoCalcFields property is true, OnCalcFields is also triggered when:

Focus moves from one visual control to another, or from one column to another is a data-aware grid control and modifications were made to the record.

Note: When the AutoCalcFields property is true, an OnCalcFields event handler should not modify the dataset (or a linked dataset if it is part of a master-detail relationship), because such modifications retrigger the OnCalcField event, leading to infinite recursion.

If an application permits users to change data, OnCalcFields is frequently triggered. To reduce the frequency with which OnCalcFields occurs, set AutoCalcFields to false.

Warning: When the dataset is the master table of a master-detail relationship, OnCalcFields occurs before detail sets have been synchronized with the master table.

See Also

Active (see page 7004)

AutoCalcFields (see page 7006)

Open (see page 6976)

Post (see page 6977)

Calculating Fields

Using Internally Calculated Fields in Client Datasets

Defining a Calculated Field

134.1.2.6.30 TSimpleDataSet.OnDeleteError Event

Occurs when an application attempts to delete a record and an exception is raised.

Pascal

```
property OnDeleteError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnDeleteError;
```

Description

Write an OnDeleteError event handler to handle exceptions that occur when an attempt to delete a record fails.

See Also

OnEditError (see page 7054)

OnPostError (see page 7056)

Delete (see page 6924)

134.1.2.6.31 TSimpleDataSet.OnEditError Event

Occurs when an application attempts to modify or insert a record and an exception is raised.

Pascal

```
property OnEditError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnEditError;
```

Description

Write an OnEditError event handler to handle exceptions that occur when an attempt to edit a record fails.

See Also

OnDeleteError (see page 7054)

OnPostError (see page 7056)

Edit (see page 6932)

Insert (see page 6959)

134.1.2.6.32 TSimpleDataSet.OnFilterRecord Event

Occurs each time a different record in the dataset becomes the active record and filtering is enabled.

Pascal

```
property OnFilterRecord: TFilterRecordEvent;
```

C++

```
__property TFilterRecordEvent OnFilterRecord;
```

Description

Write an OnFilterRecord event handler to specify for each record in a dataset whether it should be visible to the application. To indicate that a record passes the filter condition, the OnFilterRecord event handler must set the Accept parameter to true. To exclude a record, set the Accept parameter to false. Most TDataSet descendants (such as ADO datasets or BDE-based datasets) initialize Accept to true before calling the OnFilterRecord event handler.

Warning: Do not supply an OnFilterRecord event handler to a unidirectional dataset. Unidirectional datasets do not support filters, and assigning an OnFilterRecord event handler causes the dataset to raise an exception.

Filtering is enabled if the Filtered property is true. When an application is processing a filter, the State property for the dataset is dsFilter.

Use an OnFilterRecord event handler to filter records using a criterion that can't be implemented using the Filter property. For example, using the Filter property, field comparisons are not supported against local tables (Paradox, dBASE, Access, FoxPro), but an OnFilterRecord event handler can implement any criterion at all.

Tip: Be sure that any interactions between the Filter property and the OnFilterRecord event handler do not result in an empty filter set when they are used simultaneously in an application.

See Also

Filter (see page 7026)

Filtered (see page 7027)

FilterOptions (see page 7027)

State (see page 7036)

Creating Filters

Navigating Records in a Filtered Dataset

Displaying and Editing a Subset of Data Using Filters

Writing an OnFilterRecord Event Handler

134.1.2.6.33 TSimpleDataSet.OnNewRecord Event

Occurs when an application inserts or appends a new dataset record.

Pascal

```
property OnNewRecord: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent OnNewRecord;
```

Description

Write an OnNewRecord event handler to take specific actions as an application inserts or appends a new record. OnNewRecord is called as part of the insert or append process. An application might use the OnNewRecord event to set initial values for a record or as a way of implementing cascading insertions in related datasets.

See Also

AfterInsert (see page 7044)

BeforeInsert (see page 7050)

134.1.2.6.34 TSimpleDataSet.OnPostError Event

Occurs when an application attempts to modify or insert a record and an exception is raised.

Pascal

```
property OnPostError: TDataSetErrorEvent;
```

C++

```
__property TDataSetErrorEvent OnPostError;
```

Description

Write an OnPostError event handler to handle exceptions that occur when an attempt to post a record fails.

See Also

OnDeleteError (see page 7054)

OnEditError (see page 7054)

Post (see page 6977)

134.1.2.6.35 TSimpleDataSet.OnReconcileError Event

Occurs when a client dataset needs to reconcile an update to a record that could not be applied.

Pascal

```
property OnReconcileError: TReconcileErrorEvent;
```

C++

```
__property TReconcileErrorEvent OnReconcileError;
```

Description

Write an OnReconcileError event handler to respond to error conditions that arise when the Reconcile method cycles through records that could not be applied to the database. This event handles only those update errors that were not handled in an OnUpdateError event handler.

The OnReconcileError event handler is called once for each record that could not be applied to the dataset or handled by the provider.

Warning: A provider can't detect when an update conflicts with another user's changes to a memo field. This situation does not generate an OnReconcileError event.

See Also

Reconcile (see page 8441)

OldValue (see page 7175)

NewValue (see page 7174)

CurValue (see page 7155)

ApplyUpdates (see page 8394)

OnUpdateError (see page 18444)

OnUpdateError

Applying Updates

Setting Up a Simple Dataset

Reconciling Update Errors

Using the BDE to Cache Updates

Updating Data in dbExpress Applications

135 SiteComp Namespace

This is namespace SiteComp.

Topics

Name	Description
Classes (see page 19631)	The following table lists classes in this documentation.
Interfaces (see page 19645)	The following table lists interfaces in this documentation.
Functions (see page 19788)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 19789)	The following table lists structs, records, enums in this documentation.
Types (see page 19794)	The following table lists types in this documentation.
Variables (see page 19796)	The following table lists variables in this documentation.
Constants (see page 19799)	The following table lists constants in this documentation.

Classes

	Name	Description
	TAbstractDesigntimeWarnings (see page 19631)	TAbstractDesigntimeWarnings assembles design-time warning messages that appear in the HTML view of the form your application builds.
	TAbstractInetFileType (see page 19633)	This is class SiteComp.TAbstractInetFileType.
	TNotifyList (see page 19636)	TNotifyList is a collection of objects that should be notified of changes.
	TScriptComponent (see page 19641)	This is class SiteComp.TScriptComponent.
	TScriptObject (see page 19643)	This is class SiteComp.TScriptObject.

Constants

Name	Description
AdapterActionHTMLElementTypeNames (see page 19799)	This is constant SiteComp.AdapterActionHTMLElementTypeNames.
AdapterDisplayHTMLElementTypeNames (see page 19799)	This is constant SiteComp.AdapterDisplayHTMLElementTypeNames.
AdapterDisplayViewModeTypeNames (see page 19800)	This is constant SiteComp.AdapterDisplayViewModeTypeNames.
AdapterInputHTMLElementTypeNames (see page 19800)	This is constant SiteComp.AdapterInputHTMLElementTypeNames.

Functions

	Name	Description
	FullyQualifiedFieldName (see page 19788)	This is function SiteComp.FullyQualifiedFieldName.
	FullyQualifiedName (see page 19788)	This is function SiteComp.FullyQualifiedName.

Interfaces

	Name	Description
	IActionImageProducer (see page 19646)	IActionImageProducer is the interface for accessing the image or set of images that represent an adapter action.
	IAdapterAccess (see page 19649)	This is class SiteComp.IAdapterAccess.
	IAdapterActionAccess (see page 19650)	This is class SiteComp.IAdapterActionAccess.
	IAdapterActionAttributes (see page 19651)	This is class SiteComp.IAdapterActionAttributes.
	IAdapterDispatcher (see page 19653)	IAdapterDispatcher is the interface for interacting with the adapter dispatcher.
	IAdapterDispatchParams (see page 19655)	IAdapterDispatchParams describes adapter action parameters and adapter field values.
	IAdapterError (see page 19660)	This is class SiteComp.IAdapterError.
	IAdapterErrors (see page 19663)	IAdapterErrors is implemented by adapter errors objects.
	IAdapterFieldAccess (see page 19666)	This is class SiteComp.IAdapterFieldAccess.
	IAdapterFieldAttributes (see page 19667)	This is class SiteComp.IAdapterFieldAttributes.
	IAdapterHiddenField (see page 19669)	This is class SiteComp.IAdapterHiddenField.
	IAdapterHiddenFields (see page 19671)	This is class SiteComp.IAdapterHiddenFields.
	IAdapterItemRequestParams (see page 19673)	IAdapterItemRequestParams is the interface for reading the parameters for an adapter field or action request.
	IAdapterMode (see page 19677)	This is class SiteComp.IAdapterMode.
	IAdapterRequestHandler (see page 19680)	IAdapterRequestHandler handles requests to execute an adapter action or retrieve the value of an image field.
	IAsFormatted (see page 19681)	This is class SiteComp.IAsFormatted.
	IDefaultPageFileName (see page 19682)	This is class SiteComp.IDefaultPageFileName.
	IFieldValuesAdapter (see page 19683)	This is class SiteComp.IFieldValuesAdapter.
	IFindIncludeFile (see page 19684)	This is class SiteComp.IFindIncludeFile.
	IGetAdapterActions (see page 19685)	This is class SiteComp.IGetAdapterActions.
	IGetAdapterDisplayCharacteristics (see page 19686)	This is class SiteComp.IGetAdapterDisplayCharacteristics.
	IGetAdapterErrors (see page 19687)	This is class SiteComp.IGetAdapterErrors.
	IGetAdapterFields (see page 19688)	This is class SiteComp.IGetAdapterFields.
	IGetAdapterHiddenFields (see page 19689)	This is class SiteComp.IGetAdapterHiddenFields.
	IGetAdapterImage (see page 19690)	This is class SiteComp.IGetAdapterImage.
	IGetAdapterItemRequestParams (see page 19691)	This is class SiteComp.IGetAdapterItemRequestParams.
	IGetAdapterRequestDefaultResponse (see page 19692)	This is class SiteComp.IGetAdapterRequestDefaultResponse.
	IGetAdapterValuesList (see page 19694)	This is class SiteComp.IGetAdapterValuesList.
	IGetDefaultAction (see page 19695)	This is class SiteComp.IGetDefaultAction.
	IGetDesigntimeWarnings (see page 19697)	This is class SiteComp.IGetDesigntimeWarnings.
	IGetFieldValuesAdapter (see page 19698)	This is class SiteComp.IGetFieldValuesAdapter.
	IGetHTMLStyle (see page 19699)	IGetHTMLStyle is used to inquire about the preferred display representation of an adapter field or adapter action.
	IGetProducerComponent (see page 19702)	This is class SiteComp.IGetProducerComponent.
	IGetScriptObject (see page 19703)	This is class SiteComp.IGetScriptObject.
	IGetScriptObjectClassKey (see page 19704)	This is class SiteComp.IGetScriptObjectClassKey.
	IGetWebAppComponents (see page 19705)	This is class SiteComp.IGetWebAppComponents.
	IIDEFileManager (see page 19708)	This is class SiteComp.IIDEFileManager.
	IImageProducer (see page 19709)	IImageProducer is the interface for accessing the image that an adapter component supplies.
	IInternetEnvOptions (see page 19710)	This is class SiteComp.IInternetEnvOptions.
	IInternetExecuteScript (see page 19713)	This is class SiteComp.IInternetExecuteScript.
	IInternetFileTypes (see page 19714)	This is class SiteComp.IInternetFileTypes.
	IInternetScriptDebugger (see page 19715)	This is class SiteComp.IInternetScriptDebugger.
	IIsAdapterActionList (see page 19716)	This is class SiteComp.IIsAdapterActionList.
	IIterateIntfSupport (see page 19718)	This is class SiteComp.IIterateIntfSupport.
	IIterateObjectSupport (see page 19719)	This is class SiteComp.IIterateObjectSupport.
	IIteratorIndex (see page 19720)	IIteratorIndex is the interface that indicates where an object is positioned when it is iterating over its subitems.

	IIteratorSupport (see page 19722)	IIteratorSupport is the interface that allows server-side script to iterate over the items of an adapter or one of its child objects.
	INotifyAdapterChange (see page 19724)	INotifyAdapterChange is the interface that allows objects to respond when an adapter changes.
	INotifyList (see page 19725)	INotifyList is the interface that allows objects to register and unregister an interest in changes to a component.
	IPageDispatcher (see page 19727)	IPageDispatcher is the interface for interacting with the page dispatcher.
	IPageResult (see page 19732)	This is class SiteComp.IPageResult.
	ISessionsService (see page 19733)	ISessionsService is the interface for controlling a TSessionsService object.
	ISessionsServiceConnectID (see page 19736)	This is class SiteComp.ISessionsServiceConnectID.
	ISessionsServicePersist (see page 19737)	This is class SiteComp.ISessionsServicePersist.
	ISessionsServiceStoreID (see page 19739)	This is class SiteComp.ISessionsServiceStoreID.
	IValuesListAdapter (see page 19740)	IValuesListAdapter is the interface for supplying name/value pairs.
	IWebActionsList (see page 19743)	This is class SiteComp.IWebActionsList.
	IWebApplicationInfo (see page 19745)	IWebApplicationInfo is the interface for the application adapter.
	IWebContainerSubComponent (see page 19748)	This is class SiteComp.IWebContainerSubComponent.
	IWebDataFields (see page 19749)	This is class SiteComp.IWebDataFields.
	IWebEnabled (see page 19751)	This is class SiteComp.IWebEnabled.
	IWebFormComponentContainer (see page 19752)	This is class SiteComp.IWebFormComponentContainer.
	IWebGetActionName (see page 19754)	This is class SiteComp.IWebGetActionName.
	IWebGetDisplayLabel (see page 19755)	This is class SiteComp.IWebGetDisplayLabel.
	IWebGetDisplayWidth (see page 19756)	This is class SiteComp.IWebGetDisplayWidth.
	IWebGetFieldName (see page 19757)	This is class SiteComp.IWebGetFieldName.
	IWebGetFieldValue (see page 19758)	This is class SiteComp.IWebGetFieldValue.
	IWebGetFieldValues (see page 19760)	IWebGetFieldValues is used to iterate over multiple values associated with an adapter field.
	IWebGetMaxLength (see page 19763)	This is class SiteComp.IWebGetMaxLength.
	IWebGetSessionIDs (see page 19764)	This is class SiteComp.IWebGetSessionIDs.
	IWebImageHREF (see page 19765)	IWebImageHREF is the interface for obtaining an HREF that addresses an image.
	IWebInputName (see page 19766)	This is class SiteComp.IWebInputName.
	IWebIsDefaultAction (see page 19768)	This is class SiteComp.IWebIsDefaultAction.
	IWebIsDefaultField (see page 19769)	This is class SiteComp.IWebIsDefaultField.
	IWebSession (see page 19770)	IWebSession allows an application to terminate a Web session.
	IWebSessionAttributes (see page 19771)	This is class SiteComp.IWebSessionAttributes.
	IWebSessionID (see page 19776)	This is class SiteComp.IWebSessionID.
	IWebSessionIDs (see page 19777)	IWebSessionIDs lists the session IDs (see page 19779) for all sessions that TSessionsService stores in memory.
	IWebSessionValues (see page 19780)	This is class SiteComp.IWebSessionValues.
	IWebSetActionName (see page 19782)	This is class SiteComp.IWebSetActionName.
	IWebSetFieldName (see page 19783)	This is class SiteComp.IWebSetFieldName.
	IWebUserList (see page 19784)	IWebUserList is the interface for validating user logins and access rights.
	IWebValueIsEqual (see page 19786)	This is class SiteComp.IWebValueIsEqual.

Structs, Records, Enums

	Name	Description
	TAdapterActionHTMLElementType (see page 19789)	This is record SiteComp.TAdapterActionHTMLElementType.
	TAdapterDisplayCharacteristic (see page 19789)	This is record SiteComp.TAdapterDisplayCharacteristic.
	TAdapterDisplayHTMLElementType (see page 19790)	This is record SiteComp.TAdapterDisplayHTMLElementType.
	TAdapterDisplayViewModeType (see page 19790)	This is record SiteComp.TAdapterDisplayViewModeType.
	TAdapterInputHTMLElementType (see page 19791)	This is record SiteComp.TAdapterInputHTMLElementType.
	TAdapterItemRequestParamOption (see page 19792)	This is record SiteComp.TAdapterItemRequestParamOption.
	TDispatchPageFlag (see page 19792)	This is record SiteComp.TDispatchPageFlag.
	TQualifyOption (see page 19792)	This is record SiteComp.TQualifyOption.

Types

Name	Description
TAdapterDisplayCharacteristics (see page 19794)	This is type SiteComp.TAdapterDisplayCharacteristics.
TAdapterItemRequestParamOptions (see page 19794)	This is type SiteComp.TAdapterItemRequestParamOptions.
TDispatchPageFlags (see page 19795)	This is type SiteComp.TDispatchPageFlags.
TSessionID (see page 19795)	This is type SiteComp.TSessionID.

Variables

Name	Description
ExecuteScript (see page 19796)	This is variable SiteComp.ExecuteScript.
GetHTMLSampleImage (see page 19796)	This is variable SiteComp.GetHTMLSampleImage.
IDEFileManager (see page 19797)	This is variable SiteComp.IDEFileManager.
InetFileTypes (see page 19797)	This is variable SiteComp.InetFileTypes.
InternetEnvOptions (see page 19797)	This is variable SiteComp.InternetEnvOptions.
ScriptDebugger (see page 19798)	This is variable SiteComp.ScriptDebugger.

135.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TAbstractDesigntimeWarnings (see page 19631)	TAbstractDesigntimeWarnings assembles design-time warning messages that appear in the HTML view of the form your application builds.
	TAbstractInetFileType (see page 19633)	This is class SiteComp.TAbstractInetFileType.
	TNotifyList (see page 19636)	TNotifyList is a collection of objects that should be notified of changes.
	TScriptComponent (see page 19641)	This is class SiteComp.TScriptComponent.
	TScriptObject (see page 19643)	This is class SiteComp.TScriptObject.

135.1.1 TAbstractDesigntimeWarnings Class

TAbstractDesigntimeWarnings assembles design-time warning messages that appear in the HTML view of the form your application builds.

Class Hierarchy

SiteComp.TAbstractDesigntimeWarnings

Pascal

```
TAbstractDesigntimeWarnings = class;
```

C++

```
class TAbstractDesigntimeWarnings;
```

File

SiteComp

Description

TAbstractDesigntimeWarnings is the base class for objects that collect the warning messages for an adapter page producer. At design time, as you add Web items to an adapter page producer, the page producer and the Web items you add to it write warnings to a TAbstractDesigntimeWarnings object describing any problems you can rectify (such as properties that need to be set or components that need to be added.) These warning messages appear in the preview of the emerging HTML form on the HTML view page of the editor.

See Also

TAdapterPageProducer (see page 19802)

GetDesigntimeWarnings

135.1.1.1 TAbstractDesigntimeWarnings Members

The following tables list the members exposed by TAbstractDesigntimeWarnings.

Public Methods

	Name	Description
	AddObject (see page 19632)	Requests any design time warnings from a specified object.
	AddString (see page 19632)	Adds a warning string to the set of warnings that appear at design time.

135.1.1.2 TAbstractDesigntimeWarnings Methods

The methods of the TAbstractDesigntimeWarnings class are listed here.

Public Methods

	Name	Description
	AddObject (see page 19632)	Requests any design time warnings from a specified object.
	AddString (see page 19632)	Adds a warning string to the set of warnings that appear at design time.

135.1.1.2.1 TAbstractDesigntimeWarnings.AddObject Method

Requests any design time warnings from a specified object.

Pascal

```
procedure AddObject(AObject: TObject); virtual; abstract;
```

C++

```
virtual __fastcall AddObject(TObject * AObject) = 0;
```

Description

AddObject collects the design time warnings from the specified object by calling its GetDesigntimeWarnings method (if it has one). If the object has already been added by a previous call to AddObject, AddObject detects this and does nothing.

AObject is the Web item from which to request design-time warnings.

See Also

GetDesigntimeWarnings (see page 25130)

AddString (see page 19632)

135.1.1.2.2 TAbstractDesigntimeWarnings.AddString Method

Adds a warning string to the set of warnings that appear at design time.

Pascal

```
procedure AddString(AWarning: string); virtual; abstract;
```

C++

```
virtual __fastcall AddString(AnsiString AWarning) = 0;
```

Description

AddString adds a string to the design-time warnings object. The adapter page producer displays all such warning strings in the HTML form that appears in the HTML view of the editor.

AWarning is a warning string that describes a problem that the developer can rectify (such as an invalid property value or missing component).

See Also

TAdapterPageProducer (see page 19802)

AddObject (see page 19632)

135.1.2 TAbstractInetFileType Class

Class Hierarchy

Pascal

```
TAbstractInetFileType = class(TObject);
```

C++

```
class TAbstractInetFileType : public TObject;
```

File

SiteComp

Description

This is class SiteComp.TAbstractInetFileType.

135.1.2.1 TAbstractInetFileType Members

The following tables list the members exposed by TAbstractInetFileType.

Protected Methods

	Name	Description
	GetExtCount (see page 19634)	This is GetExtCount, a member of class TAbstractInetFileType.
	GetFileExt (see page 19634)	This is GetFileExt, a member of class TAbstractInetFileType.
	GetTypeID (see page 19635)	This is GetTypeID, a member of class TAbstractInetFileType.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	FileExt (see page 19635)	This is FileExt, a member of class TAbstractInetFileType.
R	FileExtCount (see page 19635)	This is FileExtCount, a member of class TAbstractInetFileType.
R	TypeID (see page 19635)	This is TypeID, a member of class TAbstractInetFileType.

135.1.2.2 TAbstractInetFileType Methods

The methods of the TAbstractInetFileType class are listed here.

Protected Methods

	Name	Description
A	GetExtCount (see page 19634)	This is GetExtCount, a member of class TAbstractInetFileType.
A	GetFileExt (see page 19634)	This is GetFileExt, a member of class TAbstractInetFileType.
A	GetTypeID (see page 19635)	This is GetTypeID, a member of class TAbstractInetFileType.

135.1.2.2.1 TAbstractInetFileType.GetExtCount Method

Pascal

```
function GetExtCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetExtCount() = 0;
```

Description

This is GetExtCount, a member of class TAbstractInetFileType.

135.1.2.2.2 TAbstractInetFileType.GetFileExt Method

Pascal

```
function GetFileExt(I: Integer): string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetFileExt(int I) = 0;
```

Description

This is GetFileExt, a member of class TAbstractInetFileType.

135.1.2.2.3 TAbstractInetFileType.GetTypeID Method

Pascal

```
function GetTypeID: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetTypeID() = 0;
```

Description

This is GetTypeID, a member of class TAbstractInetFileType.

135.1.2.3 TAbstractInetFileType Properties

The properties of the TAbstractInetFileType class are listed here.

Public Properties

	Name	Description
R	FileExt (see page 19635)	This is FileExt, a member of class TAbstractInetFileType.
R	FileExtCount (see page 19635)	This is FileExtCount, a member of class TAbstractInetFileType.
R	TypeID (see page 19635)	This is TypeID, a member of class TAbstractInetFileType.

135.1.2.3.1 TAbstractInetFileType.FileExt Property

Pascal

```
property FileExt [I: Integer]: string;
```

C++

```
__property AnsiString FileExt[int I];
```

Description

This is FileExt, a member of class TAbstractInetFileType.

135.1.2.3.2 TAbstractInetFileType.FileExtCount Property

Pascal

```
property FileExtCount: Integer;
```

C++

```
__property int FileExtCount;
```

Description

This is FileExtCount, a member of class TAbstractInetFileType.

135.1.2.3.3 TAbstractInetFileType.TypeID Property

Pascal

```
property TypeID: string;
```

C++

```
__property AnsiString TypeID;
```

Description

This is TypeID, a member of class TAbstractInetFileType.

135.1.3 TNotifyList Class

TNotifyList is a collection of objects that should be notified of changes.

Class Hierarchy

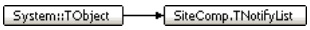

Pascal

```
TNotifyList = class(TObject);
```

C++

```
class TNotifyList : public TObject;
```

File

SiteComp

Description

TNotifyList maintains a list of objects that have registered interest in change notifications. This class is used for a helper object on components that support the INotifyList (see page 19725) interface. When objects use the INotifyList (see page 19725) interface to register interest in change notifications, the component adds them to its TNotifyList instance. When those objects use the INotifyList (see page 19725) interface to unregister their interest, they are removed from the TNotifyList instance.

When the object that uses TNotifyList changes, it sends a notification to every object in the list. For example, when an adapter changes, it obtains an INotifyAdapterChange (see page 19724) interface from every object in the list. It uses this interface to call the object's NotifyAdapterChange method.

See Also

TCustomAdapter

INotifyAdapterChange (see page 19724)

INotifyList (see page 19725)

135.1.3.1 TNotifyList Members

The following tables list the members exposed by TNotifyList.

Public Methods

	Name	Description
	AddNotify (see page 19638)	Adds a new object to the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 19637)	Creates an initializes an instance of TNotifyList (see page 19636).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 19637)	Frees an instance of TNotifyList (see page 19636).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetCount (see page 19638)	Returns the value of the Count (see page 19640) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItem (see page 19639)	Returns the value of the Items (see page 19640) property.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RemoveNotify (see page 19639)	Removes an object from the list.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Count (see page 19640)	Indicates the number of objects in the list.
R	Items (see page 19640)	Lists the objects in this collection.

135.1.3.2 TNotifyList.Create Constructor

Creates an initializes an instance of TNotifyList (see page 19636).

Pascal

```
constructor Create;
```

C++

```
__fastcall TNotifyList();
```

Description

WebSnap applications should not need to create instances of TNotifyList (see page 19636). These objects are created by the component that uses them to keep track of objects that have registered an interest in change notifications.

See Also

NotifyList (see page 24183)

INotifyList (see page 19725)

135.1.3.3 TNotifyList.Destroy Destructor

Frees an instance of TNotifyList (see page 19636).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TNotifyList();
```

Description

WebSnap applications should not need to free instances of TNotifyList (see page 19636), as this is handled by the components that use them. If you must free a TNotifyList (see page 19636) instance, use the Free method.

See Also

Free (see page 22300)

135.1.3.4 TNotifyList Methods

The methods of the TNotifyList class are listed here.

Public Methods

	Name	Description
	AddNotify (see page 19638)	Adds a new object to the list.
	GetCount (see page 19638)	Returns the value of the Count (see page 19640) property.
	GetItem (see page 19639)	Returns the value of the Items (see page 19640) property.
	RemoveNotify (see page 19639)	Removes an object from the list.

135.1.3.4.1 TNotifyList.AddNotify Method

Adds a new object to the list.

Pascal

```
procedure AddNotify(ANotify: TObject);
```

C++

```
__fastcall AddNotify(TObject * ANotify);
```

Description

Call AddNotify to add an object to the end of the Items (see page 19640) property array. This method is called to add objects that want to be informed of changes to the component that uses this list.

ANotify is the object to add. This object should implement an interface that allows the associated component to inform it of changes. For example, objects in the TNotifyList (see page 19636) collection maintained by an adapter support the INotifyAdapterChange (see page 19724) interface.

See Also

Items (see page 19640)

RemoveNotify (see page 19639)

INotifyAdapterChange (see page 19724)

135.1.3.4.2 TNotifyList.GetCount Method

Returns the value of the Count (see page 19640) property.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the read implementation of the Count (see page 19640) property. It returns the number of objects in the list.

See Also

Count (see page 19640)

135.1.3.4.3 TNotifyList.GetItem Method

Returns the value of the Items (see page 19640) property.

Pascal

```
function GetItem(Index: Integer): TObject;
```

C++

```
__fastcall TObject * GetItem(int Index);
```

Description

GetItem is the read implementation of the Items (see page 19640) property.

Index is the zero-offset index of the item to return.

GetItem returns the specified item from the list.

See Also

Items (see page 19640)

135.1.3.4.4 TNotifyList.RemoveNotify Method

Removes an object from the list.

Pascal

```
procedure RemoveNotify(ANotify: TObject);
```

C++

```
__fastcall RemoveNotify(TObject * ANotify);
```

Description

Call RemoveNotify to remove an object from the list. This method is called to remove objects that are no longer interested in changes to the associated component.

ANotify is the object to remove. This object must have been previously added by a call to the AddNotify (see page 19638) method.

See Also

Items (see page 19640)

AddNotify (see page 19638)

135.1.3.5 TNotifyList Properties

The properties of the TNotifyList class are listed here.

Public Properties

	Name	Description
R	Count (see page 19640)	Indicates the number of objects in the list.
R	Items (see page 19640)	Lists the objects in this collection.

135.1.3.5.1 TNotifyList.Count Property

Indicates the number of objects in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Count indicates the number of objects in the Items (see page 19640) property array. Use this value as an upper bound for the index when iterating over all the objects in the list.

See Also

Items (see page 19640)

135.1.3.5.2 TNotifyList.Items Property

Lists the objects in this collection.

Pascal

```
property Items [Index: Integer]: TObject;
```

C++

```
__property TObject * Items[int Index];
```

Description

Use Items to reference an object in the list. Each item is an object that supports an interface that allows the associated component to inform it of changes. For example, objects in the TNotifyList (see page 19636) collection maintained by an adapter support the INotifyAdapterChange (see page 19724) interface.

Index is the index of the object, where 0 specifies the first item, 1 specifies the second item, and so on. The Count (see page 19640) property gives an upper bound for this index.

Note: In Delphi, Items is the default property of TNotifyList (see page 19636). That means the property name may be omitted in code.

See Also

Count (see page 19640)

AddNotify (see page 19638)

RemoveNotify (see page 19639)

INotifyAdapterChange (see page 19724)

operator_sb

135.1.4 TScriptComponent Class

Class Hierarchy

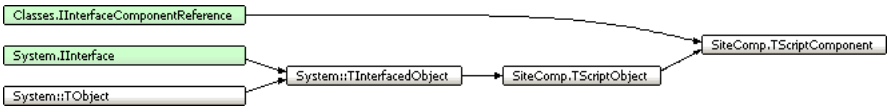

Pascal

```
TScriptComponent = class(TScriptObject, IInterfaceComponentReference);
```

C++

```
class TScriptComponent : public TScriptObject, public IInterfaceComponentReference;
```

File

SiteComp

Description

This is class SiteComp.TScriptComponent.

135.1.4.1 TScriptComponent Members

The following tables list the members exposed by TScriptComponent.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetComponent (see page 19643)	Returns a reference to the component associated by the implementing class.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 19642)	This is Create, a member of class TScriptComponent.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Obj (see page 19643)	This is Obj, a member of class TScriptComponent.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

135.1.4.2 TScriptComponent.Create Constructor

Pascal

```
constructor Create(AObject: TComponent); virtual;
```

C++

```
virtual __fastcall TScriptComponent(TComponent * AObject);
```

Description

This is Create, a member of class TScriptComponent.

135.1.4.3 TScriptComponent Methods

The methods of the TScriptComponent class are listed here.

Protected Methods

	Name	Description
	GetComponent (see page 19643)	Returns a reference to the component associated by the implementing class.

135.1.4.3.1 TScriptComponent.GetComponent Method

Returns a reference to the component associated by the implementing class.

Pascal

```
function GetComponent: TComponent;
```

C++

```
__fastcall TComponent * GetComponent();
```

Description

GetComponent returns a TComponent value that is associated with the implementing class. For example, the TComponent class itself implements the GetComponent to return a pointer to itself.

See Also

TComponent (see page 2762)

135.1.4.4 TScriptComponent Properties

The properties of the TScriptComponent class are listed here.

Public Properties

	Name	Description
R	Obj (see page 19643)	This is Obj, a member of class TScriptComponent.

135.1.4.4.1 TScriptComponent.Obj Property

Pascal

```
property Obj: TComponent;
```

C++

```
__property TComponent * Obj;
```

Description

This is Obj, a member of class TScriptComponent.

135.1.5 TScriptObject Class

Class Hierarchy

Pascal

```
TScriptObject = class(TInterfacedObject);
```

C++

```
class TScriptObject : public TInterfacedObject;
```

File

SiteComp

Description

This is class SiteComp.TScriptObject.

135.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IActionImageProducer (see page 19646)	IActionImageProducer is the interface for accessing the image or set of images that represent an adapter action.
	IAdapterAccess (see page 19649)	This is class SiteComp.IAdapterAccess.
	IAdapterActionAccess (see page 19650)	This is class SiteComp.IAdapterActionAccess.
	IAdapterActionAttributes (see page 19651)	This is class SiteComp.IAdapterActionAttributes.
	IAdapterDispatcher (see page 19653)	IAdapterDispatcher is the interface for interacting with the adapter dispatcher.
	IAdapterDispatchParams (see page 19655)	IAdapterDispatchParams describes adapter action parameters and adapter field values.
	IAdapterError (see page 19660)	This is class SiteComp.IAdapterError.
	IAdapterErrors (see page 19663)	IAdapterErrors is implemented by adapter errors objects.
	IAdapterFieldAccess (see page 19666)	This is class SiteComp.IAdapterFieldAccess.
	IAdapterFieldAttributes (see page 19667)	This is class SiteComp.IAdapterFieldAttributes.
	IAdapterHiddenField (see page 19669)	This is class SiteComp.IAdapterHiddenField.
	IAdapterHiddenFields (see page 19671)	This is class SiteComp.IAdapterHiddenFields.
	IAdapterItemRequestParams (see page 19673)	IAdapterItemRequestParams is the interface for reading the parameters for an adapter field or action request.
	IAdapterMode (see page 19677)	This is class SiteComp.IAdapterMode.
	IAdapterRequestHandler (see page 19680)	IAdapterRequestHandler handles requests to execute an adapter action or retrieve the value of an image field.
	IAsFormatted (see page 19681)	This is class SiteComp.IAsFormatted.
	IDefaultPageFileName (see page 19682)	This is class SiteComp.IDefaultPageFileName.
	IFieldValuesAdapter (see page 19683)	This is class SiteComp.IFieldValuesAdapter.
	IFindIncludeFile (see page 19684)	This is class SiteComp.IFindIncludeFile.
	IGetAdapterActions (see page 19685)	This is class SiteComp.IGetAdapterActions.
	IGetAdapterDisplayCharacteristics (see page 19686)	This is class SiteComp.IGetAdapterDisplayCharacteristics.
	IGetAdapterErrors (see page 19687)	This is class SiteComp.IGetAdapterErrors.
	IGetAdapterFields (see page 19688)	This is class SiteComp.IGetAdapterFields.
	IGetAdapterHiddenFields (see page 19689)	This is class SiteComp.IGetAdapterHiddenFields.
	IGetAdapterImage (see page 19690)	This is class SiteComp.IGetAdapterImage.
	IGetAdapterItemRequestParams (see page 19691)	This is class SiteComp.IGetAdapterItemRequestParams.
	IGetAdapterRequestDefaultResponse (see page 19692)	This is class SiteComp.IGetAdapterRequestDefaultResponse.
	IGetAdapterValuesList (see page 19694)	This is class SiteComp.IGetAdapterValuesList.
	IGetDefaultAction (see page 19695)	This is class SiteComp.IGetDefaultAction.
	IGetDesigntimeWarnings (see page 19697)	This is class SiteComp.IGetDesigntimeWarnings.
	IGetFieldValuesAdapter (see page 19698)	This is class SiteComp.IGetFieldValuesAdapter.
	IGetHTMLStyle (see page 19699)	IGetHTMLStyle is used to inquire about the preferred display representation of an adapter field or adapter action.
	IGetProducerComponent (see page 19702)	This is class SiteComp.IGetProducerComponent.
	IGetScriptObject (see page 19703)	This is class SiteComp.IGetScriptObject.
	IGetScriptObjectClassKey (see page 19704)	This is class SiteComp.IGetScriptObjectClassKey.
	IGetWebAppComponents (see page 19705)	This is class SiteComp.IGetWebAppComponents.
	IIDEFileManager (see page 19708)	This is class SiteComp.IIDEFileManager.
	IImageProducer (see page 19709)	IImageProducer is the interface for accessing the image that an adapter component supplies.
	IInternetEnvOptions (see page 19710)	This is class SiteComp.IInternetEnvOptions.
	IInternetExecuteScript (see page 19713)	This is class SiteComp.IInternetExecuteScript.

	IInternetFileTypes (see page 19714)	This is class SiteComp.IInternetFileTypes.
	IInternetScriptDebugger (see page 19715)	This is class SiteComp.IInternetScriptDebugger.
	IIsAdapterActionList (see page 19716)	This is class SiteComp.IIsAdapterActionList.
	IIterateIntfSupport (see page 19718)	This is class SiteComp.IIterateIntfSupport.
	IIterateObjectSupport (see page 19719)	This is class SiteComp.IIterateObjectSupport.
	IIteratorIndex (see page 19720)	IIteratorIndex is the interface that indicates where an object is positioned when it is iterating over its subitems.
	IIteratorSupport (see page 19722)	IIteratorSupport is the interface that allows server-side script to iterate over the items of an adapter or one of its child objects.
	INotifyAdapterChange (see page 19724)	INotifyAdapterChange is the interface that allows objects to respond when an adapter changes.
	INotifyList (see page 19725)	INotifyList is the interface that allows objects to register and unregister an interest in changes to a component.
	IPageDispatcher (see page 19727)	IPageDispatcher is the interface for interacting with the page dispatcher.
	IPageResult (see page 19732)	This is class SiteComp.IPageResult.
	ISessionsService (see page 19733)	ISessionsService is the interface for controlling a TSessionsService object.
	ISessionsServiceConnectID (see page 19736)	This is class SiteComp.ISessionsServiceConnectID.
	ISessionsServicePersist (see page 19737)	This is class SiteComp.ISessionsServicePersist.
	ISessionsServiceStoreID (see page 19739)	This is class SiteComp.ISessionsServiceStoreID.
	IValuesListAdapter (see page 19740)	IValuesListAdapter is the interface for supplying name/value pairs.
	IWebActionsList (see page 19743)	This is class SiteComp.IWebActionsList.
	IWebApplicationInfo (see page 19745)	IWebApplicationInfo is the interface for the application adapter.
	IWebContainerSubComponent (see page 19748)	This is class SiteComp.IWebContainerSubComponent.
	IWebDataFields (see page 19749)	This is class SiteComp.IWebDataFields.
	IWebEnabled (see page 19751)	This is class SiteComp.IWebEnabled.
	IWebFormComponentContainer (see page 19752)	This is class SiteComp.IWebFormComponentContainer.
	IWebGetActionName (see page 19754)	This is class SiteComp.IWebGetActionName.
	IWebGetDisplayLabel (see page 19755)	This is class SiteComp.IWebGetDisplayLabel.
	IWebGetDisplayWidth (see page 19756)	This is class SiteComp.IWebGetDisplayWidth.
	IWebGetFieldName (see page 19757)	This is class SiteComp.IWebGetFieldName.
	IWebGetFieldValue (see page 19758)	This is class SiteComp.IWebGetFieldValue.
	IWebGetFieldValues (see page 19760)	IWebGetFieldValues is used to iterate over multiple values associated with an adapter field.
	IWebGetMaxLength (see page 19763)	This is class SiteComp.IWebGetMaxLength.
	IWebGetSessionIDs (see page 19764)	This is class SiteComp.IWebGetSessionIDs.
	IWebImageHREF (see page 19765)	IWebImageHREF is the interface for obtaining an HREF that addresses an image.
	IWebInputName (see page 19766)	This is class SiteComp.IWebInputName.
	IWebIsDefaultAction (see page 19768)	This is class SiteComp.IWebIsDefaultAction.
	IWebIsDefaultField (see page 19769)	This is class SiteComp.IWebIsDefaultField.
	IWebSession (see page 19770)	IWebSession allows an application to terminate a Web session.
	IWebSessionAttributes (see page 19771)	This is class SiteComp.IWebSessionAttributes.
	IWebSessionID (see page 19776)	This is class SiteComp.IWebSessionID.
	IWebSessionIDs (see page 19777)	IWebSessionIDs lists the session IDs (see page 19779) for all sessions that TSessionsService stores in memory.
	IWebSessionValues (see page 19780)	This is class SiteComp.IWebSessionValues.
	IWebSetActionName (see page 19782)	This is class SiteComp.IWebSetActionName.
	IWebSetFieldName (see page 19783)	This is class SiteComp.IWebSetFieldName.
	IWebUserList (see page 19784)	IWebUserList is the interface for validating user logins and access rights.
	IWebValueIsEqual (see page 19786)	This is class SiteComp.IWebValueIsEqual.

135.2.1 IActionImageProducer Interface

IActionImageProducer is the interface for accessing the image or set of images that represent an adapter action.

Class Hierarchy

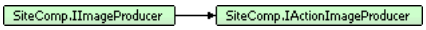

Pascal

```
IActionImageProducer = interface(IImageProducer);
```

C++

```
__interface IActionImageProducer : public IImageProducer;
```

File

SiteComp

Description

IActionImageProducer is the interface used to access the object that supplies images for an adapter action. This interface can supply a single image for the action, or a set of related images that can be used when an HTML control changes its appearance in response to user actions.

IActionImageProducer does not directly supply the HREF for the image or images it represents. Instead, it supplies the interface for another object. That object may generate the image, or it may simply generate an HREF that addresses the object that generates the image. This extra level of indirection allows the object that implements IActionImageProducer to work with multiple adapter fields that produce a variety of images.

Note: Method declarations that use IActionImageProducer use the _di_IActionImageProducer type instead. This type is a DelphiInterface wrapper around the IActionImageProducer interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IActionImageProducer >
_di_IActionImageProducer;
```

See Also

TAdapterActionButton (see page 25034)

TCustomAdapterActionButton (see page 25189)

135.2.1.1 IActionImageProducer Members

The following tables list the members exposed by IActionImageProducer.

Public Methods

	Name	Description
	GetAdapterEventImage (see page 19648)	Returns the object that is responsible for providing a specified image from a set of related images.
	GetAdapterImage (see page 19710)	This is GetAdapterImage, a member of class IImageProducer.
	GetDisplayStyle (see page 19648)	Indicates what type of images this interface can be used to access.

135.2.1.2 IActionImageProducer Methods

The methods of the IActionImageProducer class are listed here.

Public Methods

	Name	Description
	GetAdapterEventImage (see page 19648)	Returns the object that is responsible for providing a specified image from a set of related images.

	GetDisplayStyle (see page 19648)	Indicates what type of images this interface can be used to access.

135.2.1.2.1 IActionImageProducer.GetAdapterEventImage Method

Returns the object that is responsible for providing a specified image from a set of related images.

Pascal

```
function GetAdapterEventImage(const AComponentName: WideString; const AEvent: WideString;
const ACaption: WideString): TObject;
```

C++

```
__fastcall TObject * GetAdapterEventImage(const BSTR AComponentName, const BSTR AEvent, const
BSTR ACaption);
```

Description

GetAdapterEventImage returns the object that supplies an HREF for the specified image. That object can supply the HREF in one of two ways:

It can implement the IWebImageHREF (see page 19765) interface.

It can be the object that produces the image in response to request messages. In this case, the adapter dispatcher's EncodeAdapterItemRequestAsHREF method generates the HREF to address this object.

AComponentName is the name of the component that is requesting the image. GetAdapterEventImage can use this parameter to determine which of several objects is appropriate for supplying the image.

AEvent identifies the user event that the image should reflect. This parameter is used when the GetDisplayStyle (see page 19648) method returns 'EventImages'. If AEvent is an empty string, GetAdapterEventImage returns the object that supplies the base image. If AEvent is not empty, GetAdapterEventImage returns the object that supplies the version of the image to use in response to the associated user action. Possible values are 'onmouseover', 'onmouseout', and 'onmousedown'.

ACaption is a text label that describes the desired image. GetAdapterEventImage can use this parameter to determine which of several objects is appropriate for supplying the image.

See Also

IWebImageHREF (see page 19765)

EncodeAdapterItemRequestAsHREF (see page 24872)

getadapterimage (see page 19710)

GetDisplayStyle (see page 19648)

135.2.1.2.2 IActionImageProducer.GetDisplayStyle Method

Indicates what type of images this interface can be used to access.

Pascal

```
function GetDisplayStyle(Sender: TComponent): string;
```

C++

```
__fastcall AnsiString GetDisplayStyle(TComponent * Sender);
```

Description

GetDisplayStyle returns either 'Image' or 'EventImages', depending on whether it can supply a single image for an adapter action and caption, or multiple images that reflect various user events.

If GetDisplayStyle returns 'Image', use the GetAdapterImage (see page 19710) method to obtain an interface for the image

object. If GetDisplayStyle returns 'EventImages', use the GetAdapterEventImage (see page 19648) method to obtain an interface for the image object.

When a TAdapterActionButton component uses this interface for its ImageProducer property, its DisplayType property should be compatible with the display style this method returns. That is, if this method returns 'Image', the adapter action button should have a DisplayType of ctImage or ctDefault. If this method returns 'EventImages', the adapter action button can have a DisplayType of ctImage, ctEventImages, or ctDefault.

Sender is the component that is requesting the image.

See Also

DisplayType (see page 25202)

GetAdapterEventImage (see page 19648)

getadapterimage (see page 19710)

135.2.2 IAdapterAccess Interface

Class Hierarchy

SiteComp.IAdapterAccess

Pascal

```
IAdapterAccess = interface;
```

C++

```
__interface IAdapterAccess;
```

File

SiteComp

Description

This is class SiteComp.IAdapterAccess.

135.2.2.1 IAdapterAccess Members

The following tables list the members exposed by IAdapterAccess.

Public Methods

	Name	Description
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.

135.2.2.2 IAdapterAccess Methods

The methods of the IAdapterAccess class are listed here.

Public Methods

	Name	Description
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.

135.2.2.2.1 IAdapterAccess.HasExecuteAccess Method

Pascal

```
function HasExecuteAccess: Boolean;
```

C++

```
__fastcall Boolean HasExecuteAccess();
```

Description

This is HasExecuteAccess, a member of class IAdapterAccess.

135.2.2.2.2 IAdapterAccess.HasModifyAccess Method

Pascal

```
function HasModifyAccess: Boolean;
```

C++

```
__fastcall Boolean HasModifyAccess();
```

Description

This is HasModifyAccess, a member of class IAdapterAccess.

135.2.2.2.3 IAdapterAccess.HasViewAccess Method

Pascal

```
function HasViewAccess: Boolean;
```

C++

```
__fastcall Boolean HasViewAccess();
```

Description

This is HasViewAccess, a member of class IAdapterAccess.

135.2.3 IAdapterActionAccess Interface

Class Hierarchy

SiteComp.IAdapterActionAccess

Pascal

```
IAdapterActionAccess = interface;
```

C++

```
__interface IAdapterActionAccess;
```

File

SiteComp

Description

This is class SiteComp.IAdapterActionAccess.

135.2.3.1 IAdapterActionAccess Members

The following tables list the members exposed by IAdapterActionAccess.

Public Methods

	Name	Description
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.

135.2.3.2 IAdapterActionAccess Methods

The methods of the IAdapterActionAccess class are listed here.

Public Methods

	Name	Description
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.

135.2.3.2.1 IAdapterActionAccess.HasExecuteAccess Method

Pascal

```
function HasExecuteAccess: Boolean;
```

C++

```
__fastcall Boolean HasExecuteAccess();
```

Description

This is HasExecuteAccess, a member of class IAdapterActionAccess.

135.2.4 IAdapterActionAttributes Interface

Class Hierarchy

SiteComp.IAdapterActionAttributes

Pascal

```
IAdapterActionAttributes = interface;
```

C++

```
__interface IAdapterActionAttributes;
```

File

SiteComp

Description

This is class SiteComp.IAdapterActionAttributes.

135.2.4.1 IAdapterActionAttributes Members

The following tables list the members exposed by IAdapterActionAttributes.

Public Methods

	Name	Description
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.

Public Properties

	Name	Description
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

135.2.4.2 IAdapterActionAttributes Methods

The methods of the IAdapterActionAttributes class are listed here.

Public Methods

	Name	Description
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.

135.2.4.2.1 IAdapterActionAttributes.GetVisible Method

Pascal

```
function GetVisible: Boolean;
```

C++

```
__fastcall Boolean GetVisible();
```

Description

This is GetVisible, a member of class IAdapterActionAttributes.

135.2.4.3 IAdapterActionAttributes Properties

The properties of the IAdapterActionAttributes class are listed here.

Public Properties

	Name	Description
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

135.2.4.3.1 IAdapterActionAttributes.Visible Property

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

This is Visible, a member of class IAdapterActionAttributes.

135.2.5 IAdapterDispatcher Interface

IAdapterDispatcher is the interface for interacting with the adapter dispatcher.

Class Hierarchy

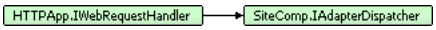

Pascal

```
IAdapterDispatcher = interface(IWebRequestHandler);
```

C++

```
__interface IAdapterDispatcher : public IWebRequestHandler;
```

File

SiteComp

Description

The TAdapterDispatcher component implements IAdapterDispatcher. This interface, which the TWebAppComponents object publishes, allows the WebSnap application to dispatch request messages that represent HTML form submissions and requests for dynamic images that can be resolved using adapter actions and fields. It also encodes the hidden fields and HREF values that are used to make those calls to adapter fields and actions.

Note: In C++, method declarations that use IAdapterDispatcher use the _di_IAdapterDispatcher type instead. This type is a DelphiInterface wrapper around the IAdapterDispatcher interface:

Note: typedef System::DelphiInterface (see page 22167)< IAdapterDispatcher > _di_IAdapterDispatcher;

See Also

TAdapterDispatcher (see page 24840)

TCustomWebAppComponents (see page 24906)

TWebAPpComponents

135.2.5.1 IAdapterDispatcher Members

The following tables list the members exposed by IAdapterDispatcher.

Public Methods

	Name	Description
	EncodeAdapterItemRequestAsFieldValue (see page 19654)	Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.
	EncodeAdapterItemRequestAsHREF (see page 19654)	Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.
	GetAdapterItemRequestParams (see page 19655)	Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.
	HandleRequest (see page 15238)	Processes an HTTP request message.

135.2.5.2 IAdapterDispatcher Methods

The methods of the IAdapterDispatcher class are listed here.

Public Methods

	Name	Description
	EncodeAdapterItemRequestAsFieldValue (see page 19654)	Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.
	EncodeAdapterItemRequestAsHREF (see page 19654)	Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.
	GetAdapterItemRequestParams (see page 19655)	Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.

135.2.5.2.1 IAdapterDispatcher.EncodeAdapterItemRequestAsFieldValue Method

Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.

Pascal

```
function EncodeAdapterItemRequestAsFieldValue(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions): string;
```

C++

```
__fastcall AnsiString EncodeAdapterItemRequestAsFieldValue(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions);
```

Description

Call EncodeAdapterItemRequestAsFieldValue to encode the adapter action or image field parameters as they should appear as field values in form submissions. That is, the resulting parameter string has a form such as '?param1=value1&param2=value2'.

AHandler is the interface of the adapter whose action or image field requires the parameters.

AOptions is a set of flags that indicate what types of parameters to include.

EncodeAdapterItmRequestAsFieldValue returns the parameter string.

See Also

GetAdapterItemRequestParams (see page 19655)

EncodeAdapterItemRequestAsHREF (see page 19654)

135.2.5.2.2 IAdapterDispatcher.EncodeAdapterItemRequestAsHREF Method

Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.

Pascal

```
function EncodeAdapterItemRequestAsHREF(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions): string;
```

C++

```
__fastcall AnsiString EncodeAdapterItemRequestAsHREF(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions);
```

Description

Call EncodeAdapterItemRequestAsHREF to encode the adapter action or image field parameters as they should appear as parameters to an anchor tag in an HTML form.

AHandler is the interface of the adapter whose action or image field requires the parameters.

AOptions is a set of flags that indicate what types of parameters to include.

EncodeAdapterItemRequestAsHREF returns the parameter string.

See Also

GetAdapterItemRequestParams (see page 19655)

EncodeAdapterItemRequestAsFieldValue (see page 19654)

135.2.5.2.3 IAdapterDispatcher.GetAdapterItemRequestParams Method

Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.

Pascal

```
procedure GetAdapterItemRequestParams(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions; AParams: IAdapterItemRequestParams);
```

C++

```
__fastcall GetAdapterItemRequestParams(IUnknown AHandler, TAdapterItemRequestParamOptions
AOptions, IAdapterItemRequestParams AParams);
```

Description

Call GetAdapterItemRequestParams to retrieve the adapter action and image field parameters.

AHandler is the interface of the adapter whose action or field requires the parameters.

AOptions is a set of flags that indicate what types of parameters should be included.

AParams is an interface for reading and writing parameter values. GetAdapterItemRequestParams uses this interface to write the parameter values.

See Also

EncodeAdapterItemRequestAsHREF (see page 19654)

EncodeAdapterItemRequestAsFieldValue (see page 19654)

135.2.6 IAdapterDispatchParams Interface

IAdapterDispatchParams describes adapter action parameters and adapter field values.

Class Hierarchy

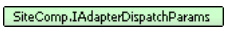

Pascal

```
IAdapterDispatchParams = interface;
```

C++

```
__interface IAdapterDispatchParams;
```

File

SiteComp

Description

Use IAdapterDispatchParams to obtain context information that is needed to execute an adapter action or to obtain the value of an image field from an adapter. When server-side script in a WebSnap application executes an adapter action or requests the value of an image field, an object that implements this interface is created to store information about the adapter that can perform the specified task and the parameters that must be passed to the adapter for it to do so. This interface is then passed to the adapter action's HandleRequest method so that it can look up the adapter and the parameters.

Note: In C++, method declarations that use IAdapterDispatchParams use the _di_IAdapterDispatchParams type instead. This type is a DelphiInterface wrapper around the IAdapterDispatchParams interface:

Note: typedef System::DelphiInterface (see page 22167)< IAdapterDispatchParams > _di_IAdapterDispatchParams;

See Also

IAdapterRequestHandler (see page 19680)

TAdapterDispatchParams (see page 24848)

TBaseAdapterAction (see page 24043)

135.2.6.1 IAdapterDispatchParams Members

The following tables list the members exposed by IAdapterDispatchParams.

Public Methods

	Name	Description
	GetHandler (see page 19657)	Returns the value of the Handler (see page 19658) property.
	GetObjectIdentifier (see page 19657)	Returns the value of the ObjectIdentifier (see page 19658) property.
	GetParams (see page 19657)	Returns the value of the Params (see page 19659) property.
	GetRequestIdentifier (see page 19657)	Returns the value of the RequestIdentifier (see page 19659) property.

Public Properties

	Name	Description
R	Handler (see page 19658)	Indicates the adapter object that can handle the current request.
R	ObjectIdentifier (see page 19658)	Specifies the name of the action to execute or the field to retrieve.
R	Params (see page 19659)	Indicates the parameter values to use when executing the action or looking up the image.
R	RequestIdentifier (see page 19659)	Identifies the type of the request identified by the ObjectIdentifier (see page 19658) property.

135.2.6.2 IAdapterDispatchParams Methods

The methods of the IAdapterDispatchParams class are listed here.

Public Methods

	Name	Description
	GetHandler (see page 19657)	Returns the value of the Handler (see page 19658) property.
	GetObjectIdentifier (see page 19657)	Returns the value of the ObjectIdentifier (see page 19658) property.
	GetParams (see page 19657)	Returns the value of the Params (see page 19659) property.

	GetRequestIdentifier (see page 19657)	Returns the value of the RequestIdentifier (see page 19659) property.

135.2.6.2.1 IAdapterDispatchParams.GetHandler Method

Returns the value of the Handler (see page 19658) property.

Pascal

```
function GetHandler: TObject;
```

C++

```
__fastcall TObject * GetHandler();
```

Description

GetHandler is the read implementation of the Handler (see page 19658) property.

See Also

Handler (see page 19658)

135.2.6.2.2 IAdapterDispatchParams.GetObjectIdentifier Method

Returns the value of the ObjectIdentifier (see page 19658) property.

Pascal

```
function GetObjectIdentifier: string;
```

C++

```
__fastcall AnsiString GetObjectIdentifier();
```

Description

GetObjectIdentifier is the read implementation of the ObjectIdentifier (see page 19658) property.

See Also

ObjectIdentifier (see page 19658)

135.2.6.2.3 IAdapterDispatchParams.GetParams Method

Returns the value of the Params (see page 19659) property.

Pascal

```
function GetParams: TStrings;
```

C++

```
__fastcall TStrings GetParams();
```

Description

GetParams is the read implementation of the Params (see page 19659) property.

See Also

Params (see page 19659)

135.2.6.2.4 IAdapterDispatchParams.GetRequestIdentifier Method

Returns the value of the RequestIdentifier (see page 19659) property.

Pascal

```
function GetRequestIdentifier: string;
```

C++

```
__fastcall AnsiString GetRequestIdentifier();
```

Description

GetRequestIdentifier is the read implementation of the RequestIdentifier (see page 19659) property.

See Also

RequestIdentifier (see page 19659)

135.2.6.3 IAdapterDispatchParams Properties

The properties of the IAdapterDispatchParams class are listed here.

Public Properties

	Name	Description
R	Handler (see page 19658)	Indicates the adapter object that can handle the current request.
R	ObjectIdentifier (see page 19658)	Specifies the name of the action to execute or the field to retrieve.
R	Params (see page 19659)	Indicates the parameter values to use when executing the action or looking up the image.
R	RequestIdentifier (see page 19659)	Identifies the type of the request identified by the ObjectIdentifier (see page 19658) property.

135.2.6.3.1 IAdapterDispatchParams.Handler Property

Indicates the adapter object that can handle the current request.

Pascal

```
property Handler: TObject;
```

C++

```
__property TObject * Handler;
```

Description

Read Handler to access the adapter that can execute the action or supply the image that is requested.

See Also

ObjectIdentifier (see page 19658)

RequestIdentifier (see page 24854)

135.2.6.3.2 IAdapterDispatchParams.ObjectIdentifier Property

Specifies the name of the action to execute or the field to retrieve.

Pascal

```
property ObjectIdentifier: string;
```

C++

```
__property AnsiString ObjectIdentifier;
```

Description

Read ObjectIdentifier to determine the name of the requested action or field. The interpretation of this value is the responsibility of the object specified by the Handler (see page 19658) parameter.

See Also

Handler (see page 19658)

RequestIdentifier (see page 24854)

135.2.6.3.3 IAdapterDispatchParams.Params Property

Indicates the parameter values to use when executing the action or looking up the image.

Pascal

```
property Params: TStrings;
```

C++

```
__property TStrings Params;
```

Description

Read Params to determine the parameters that the adapter action can use to perform its task. Each string in this string list has the form Name=Value, where Name is the name of a parameter and Value is its value.

The interpretation of the parameter values is up to the adapter specified by the Handler (see page 19658) object or to the adapter action object that handles the request.

See Also

Handler (see page 19658)

ObjectIdentifier (see page 24854)

135.2.6.3.4 IAdapterDispatchParams.RequestIdentifier Property

Identifies the type of the request identified by the ObjectIdentifier (see page 19658) property.

Pascal

```
property RequestIdentifier: string;
```

C++

```
__property AnsiString RequestIdentifier;
```

Description

Read RequestIdentifier to determine the type of processing the current request requires. For example, the RequestIdentifier may identify the request as a request for the value of an image field or a request to execute an action.

The interpretation of this value is the responsibility of the object specified by the Handler (see page 19658) parameter.

See Also

Handler (see page 19658)

ObjectIdentifier (see page 24854)

135.2.7 IAdapterError Interface

Class Hierarchy

SiteComp.IAdapterError

Pascal

```
IAdapterError = interface;
```

C++

```
__interface IAdapterError;
```

File

SiteComp

Description

This is class SiteComp.IAdapterError.

135.2.7.1 IAdapterError Members

The following tables list the members exposed by IAdapterError.

Public Methods

	Name	Description
	GetErrorText (see page 19660)	Returns the value of the ErrorText (see page 19662) property.
	GetField (see page 19661)	Returns the value of the Field (see page 19662) property.
	GetID (see page 19661)	Returns the value of the ID (see page 19662) property.

Public Properties

	Name	Description
R	ErrorText (see page 19662)	Returns text describing the error.
R	Field (see page 19662)	Returns text describing the error.
R	ID (see page 19662)	Returns error ID.

135.2.7.2 IAdapterError Methods

The methods of the IAdapterError class are listed here.

Public Methods

	Name	Description
	GetErrorText (see page 19660)	Returns the value of the ErrorText (see page 19662) property.
	GetField (see page 19661)	Returns the value of the Field (see page 19662) property.
	GetID (see page 19661)	Returns the value of the ID (see page 19662) property.

135.2.7.2.1 IAdapterError.GetErrorText Method

Returns the value of the ErrorText (see page 19662) property.

Pascal

```
function GetErrorText: string;
```

C++

```
__fastcall AnsiString GetErrorText();
```

Description

GetErrorText is the read implementation of the ErrorText (see page 19662) property.

See Also

ErrorText (see page 19662)

135.2.7.2.2 IAdapterError.GetField Method

Returns the value of the Field (see page 19662) property.

Pascal

```
function GetField: TComponent;
```

C++

```
__fastcall TComponent * GetField();
```

Description

GetField is the read implementation of the Field (see page 19662) property.

See Also

Field (see page 19662)

135.2.7.2.3 IAdapterError.GetID Method

Returns the value of the ID (see page 19662) property.

Pascal

```
function GetID: integer;
```

C++

```
__fastcall int GetID();
```

Description

GetID is the read implementation of the ID (see page 19662) property.

See Also

ID (see page 19662)

135.2.7.3 IAdapterError Properties

The properties of the IAdapterError class are listed here.

Public Properties

	Name	Description
R	ErrorText (see page 19662)	Returns text describing the error.
R	Field (see page 19662)	Returns text describing the error.

R	ID (see page 19662)	Returns error ID.

135.2.7.3.1 IAdapterError.ErrorText Property

Returns text describing the error.

Pascal

```
property ErrorText: string;
```

C++

```
__property AnsiString ErrorText;
```

Description

Examine ErrorText to obtain text describing the error.

See Also

GetErrorText (see page 19660)

135.2.7.3.2 IAdapterError.Field Property

Returns text describing the error.

Pascal

```
property Field: TComponent;
```

C++

```
__property TComponent * Field;
```

Description

Examine Field to obtain a reference to the field component involved in the error.

See Also

GetField (see page 19661)

135.2.7.3.3 IAdapterError.ID Property

Returns error ID.

Pascal

```
property ID: integer;
```

C++

```
__property int ID;
```

Description

Examine ID to obtain the error ID value.

See Also

GetID (see page 19661)

ID (see page 23698)

135.2.8 IAdapterErrors Interface

IAdapterErrors is implemented by adapter errors objects.

Class Hierarchy

SiteComp.IAdapterErrors

Pascal

```
IAdapterErrors = interface;
```

C++

```
__interface IAdapterErrors;
```

File

SiteComp

Description

Adapter errors objects keep a list of errors detected while executing an adapter action or generating a page using an adapter. Adapter errors objects may be accessed with server-side script using the adapter's Errors (see page 19666) property.

See Also

IAdapterError (see page 19660)

TCustomAdapter

135.2.8.1 IAdapterErrors Members

The following tables list the members exposed by IAdapterErrors.

Public Methods

	Name	Description
	ClearErrors (see page 19664)	Clears the errors list.
	DefineLabel (see page 19664)	Associates a label with all errors associated with a particular adapter field.
	GetError (see page 19664)	Returns a particular error.
	GetErrorCount (see page 19665)	Return the number of errors in the errors list.
	GetObjectErrors (see page 19665)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 19665)	Indicates whether the errors list contains objects associated with a particular adapter field.

Public Properties

	Name	Description
R	ErrorCount (see page 19666)	Returns the number of errors in the errors list.
R	Errors (see page 19666)	Returns an error from the list.

135.2.8.2 IAdapterErrors Methods

The methods of the IAdapterErrors class are listed here.

Public Methods

	Name	Description
	ClearErrors (see page 19664)	Clears the errors list.
	DefineLabel (see page 19664)	Associates a label with all errors associated with a particular adapter field.
	GetError (see page 19664)	Returns a particular error.
	GetErrorCount (see page 19665)	Return the number of errors in the errors list.
	GetObjectErrors (see page 19665)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 19665)	Indicates whether the errors list contains objects associated with a particular adapter field.

135.2.8.2.1 IAdapterErrors.ClearErrors Method

Clears the errors list.

Pascal

```
procedure ClearErrors;
```

C++

```
__fastcall ClearErrors();
```

Description

Clears the errors list.

135.2.8.2.2 IAdapterErrors.DefineLabel Method

Associates a label with all errors associated with a particular adapter field.

Pascal

```
procedure DefineLabel(const AName: string; const ALabel: string);
```

C++

```
__fastcall DefineLabel(const AnsiString AName, const AnsiString ALabel);
```

Description

DefineLabel is responsible for replacing an label text in error messages associated with a particular adapter field with the given label. DefineLabel is used when the label used on the Web page is not the same as the label generated as part of the error message.

135.2.8.2.3 IAdapterErrors.GetError Method

Returns a particular error.

Pascal

```
function GetError(I: Integer): IAdapterError;
```

C++

```
__fastcall IAdapterError GetError(int I);
```

Description

I is a zero-based index.

See Also

IAdapterError (see page 19660)

135.2.8.2.4 IAdapterErrors.GetErrorCount Method

Return the number of errors in the errors list.

Pascal

```
function GetErrorCount: Integer;
```

C++

```
__fastcall int GetErrorCount();
```

Description

Return the number of errors in the errors list.

135.2.8.2.5 IAdapterErrors.GetObjectErrors Method

Returns an object that is used to enumerate all errors associated with an adapter field.

Pascal

```
function GetObjectErrors(const AName: string): TObject;
```

C++

```
__fastcall TObject * GetObjectErrors(const AnsiString AName);
```

Description

AName identifies an adapter field. nil (Delphi) or NULL (C++) is returned if there are no errors associated with the adapter field.

See Also

HasObjectErrors (see page 19665)

135.2.8.2.6 IAdapterErrors.HasObjectErrors Method

Indicates whether the errors list contains objects associated with a particular adapter field.

Pascal

```
function HasObjectErrors(const AName: string): Boolean;
```

C++

```
__fastcall Boolean HasObjectErrors(const AnsiString AName);
```

Description

HasObjectErrors returns true if any errors are associated with the given adapter field. AName identifies an adapter field.

See Also

GetObjectErrors (see page 19665)

135.2.8.3 IAdapterErrors Properties

The properties of the IAdapterErrors class are listed here.

Public Properties

	Name	Description
R	ErrorCount (see page 19666)	Returns the number of errors in the errors list.

	Errors (see page 19666)	Returns an error from the list.

135.2.8.3.1 IAdapterErrors.ErrorCount Property

Returns the number of errors in the errors list.

Pascal

```
property ErrorCount: Integer;
```

C++

```
__property int ErrorCount;
```

Description

ErrorCount returns the result of GetErrorCount (see page 19665).

See Also

GetErrorCount (see page 19665)

135.2.8.3.2 IAdapterErrors.Errors Property

Returns an error from the list.

Pascal

```
property Errors [I: Integer]: IAdapterError;
```

C++

```
__property IAdapterError Errors[int I];
```

Description

I is a zero-based index. Errors returns the result of GetError (see page 19664).

See Also

IAdapterError (see page 19660)

GetError (see page 19664)

135.2.9 IAdapterFieldAccess Interface

Class Hierarchy

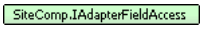

Pascal

```
IAdapterFieldAccess = interface;
```

C++

```
__interface IAdapterFieldAccess;
```

File

SiteComp

Description

This is class SiteComp.IAdapterFieldAccess.

135.2.9.1 IAdapterFieldAccess Members

The following tables list the members exposed by IAdapterFieldAccess.

Public Methods

	Name	Description
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.

135.2.9.2 IAdapterFieldAccess Methods

The methods of the IAdapterFieldAccess class are listed here.

Public Methods

	Name	Description
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.

135.2.9.2.1 IAdapterFieldAccess.HasModifyAccess Method

Pascal

```
function HasModifyAccess: Boolean;
```

C++

```
__fastcall Boolean HasModifyAccess();
```

Description

This is HasModifyAccess, a member of class IAdapterFieldAccess.

135.2.9.2.2 IAdapterFieldAccess.HasViewAccess Method

Pascal

```
function HasViewAccess: Boolean;
```

C++

```
__fastcall Boolean HasViewAccess();
```

Description

This is HasViewAccess, a member of class IAdapterFieldAccess.

135.2.10 IAdapterFieldAttributes Interface

Class Hierarchy

SiteComp.IAdapterFieldAttributes

Pascal

```
IAdapterFieldAttributes = interface;
```

C++

```
__interface IAdapterFieldAttributes;
```

File

SiteComp

Description

This is class SiteComp.IAdapterFieldAttributes.

135.2.10.1 IAdapterFieldAttributes Members

The following tables list the members exposed by IAdapterFieldAttributes.

Public Methods

	Name	Description
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.

Public Properties

	Name	Description
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

135.2.10.2 IAdapterFieldAttributes Methods

The methods of the IAdapterFieldAttributes class are listed here.

Public Methods

	Name	Description
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.

135.2.10.2.1 IAdapterFieldAttributes.GetRequired Method

Pascal

```
function GetRequired: Boolean;
```

C++

```
__fastcall Boolean GetRequired();
```

Description

This is GetRequired, a member of class IAdapterFieldAttributes.

135.2.10.2.2 IAdapterFieldAttributes.GetVisible Method

Pascal

```
function GetVisible: Boolean;
```

C++

```
__fastcall Boolean GetVisible();
```

Description

This is GetVisible, a member of class IAdapterFieldAttributes.

135.2.10.3 IAdapterFieldAttributes Properties

The properties of the IAdapterFieldAttributes class are listed here.

Public Properties

	Name	Description
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

135.2.10.3.1 IAdapterFieldAttributes.Required Property

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;
```

Description

This is Required, a member of class IAdapterFieldAttributes.

135.2.10.3.2 IAdapterFieldAttributes.Visible Property

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

This is Visible, a member of class IAdapterFieldAttributes.

135.2.11 IAdapterHiddenField Interface

Class Hierarchy

SiteComp.IAdapterHiddenField

Pascal

```
IAdapterHiddenField = interface;
```

C++

```
__interface IAdapterHiddenField;
```

File

SiteComp

Description

This is class SiteComp.IAdapterHiddenField.

135.2.11.1 IAdapterHiddenField Members

The following tables list the members exposed by IAdapterHiddenField.

Public Methods

	Name	Description
	GetName (see page 19670)	This is GetName, a member of class IAdapterHiddenField.
	GetValue (see page 19670)	This is GetValue, a member of class IAdapterHiddenField.

Public Properties

	Name	Description
R	Name (see page 19671)	This is Name, a member of class IAdapterHiddenField.
R	Value (see page 19671)	This is Value, a member of class IAdapterHiddenField.

135.2.11.2 IAdapterHiddenField Methods

The methods of the IAdapterHiddenField class are listed here.

Public Methods

	Name	Description
	GetName (see page 19670)	This is GetName, a member of class IAdapterHiddenField.
	GetValue (see page 19670)	This is GetValue, a member of class IAdapterHiddenField.

135.2.11.2.1 IAdapterHiddenField.GetName Method

Pascal

```
function GetName: string;
```

C++

```
__fastcall AnsiString GetName();
```

Description

This is GetName, a member of class IAdapterHiddenField.

135.2.11.2.2 IAdapterHiddenField.GetValue Method

Pascal

```
function GetValue: string;
```

C++

```
__fastcall AnsiString GetValue();
```

Description

This is GetValue, a member of class IAdapterHiddenField.

135.2.11.3 IAdapterHiddenField Properties

The properties of the IAdapterHiddenField class are listed here.

Public Properties

	Name	Description
R	Name (see page 19671)	This is Name, a member of class IAdapterHiddenField.
R	Value (see page 19671)	This is Value, a member of class IAdapterHiddenField.

135.2.11.3.1 IAdapterHiddenField.Name Property

Pascal

```
property Name: string;
```

C++

```
__property AnsiString Name;
```

Description

This is Name, a member of class IAdapterHiddenField.

135.2.11.3.2 IAdapterHiddenField.Value Property

Pascal

```
property Value: string;
```

C++

```
__property AnsiString Value;
```

Description

This is Value, a member of class IAdapterHiddenField.

135.2.12 IAdapterHiddenFields Interface

Class Hierarchy

SiteComp.IAdapterHiddenFields

Pascal

```
IAdapterHiddenFields = interface;
```

C++

```
__interface IAdapterHiddenFields;
```

File

SiteComp

Description

This is class SiteComp.IAdapterHiddenFields.

135.2.12.1 IAdapterHiddenFields Members

The following tables list the members exposed by IAdapterHiddenFields.

Public Methods

	Name	Description
	GetField (see page 19672)	This is GetField, a member of class IAdapterHiddenFields.
	GetFieldCount (see page 19672)	This is GetFieldCount, a member of class IAdapterHiddenFields.

Public Properties

	Name	Description
R	FieldCount (see page 19673)	This is FieldCount, a member of class IAdapterHiddenFields.
R	Fields (see page 19673)	This is Fields, a member of class IAdapterHiddenFields.

135.2.12.2 IAdapterHiddenFields Methods

The methods of the IAdapterHiddenFields class are listed here.

Public Methods

	Name	Description
	GetField (see page 19672)	This is GetField, a member of class IAdapterHiddenFields.
	GetFieldCount (see page 19672)	This is GetFieldCount, a member of class IAdapterHiddenFields.

135.2.12.2.1 IAdapterHiddenFields.GetField Method

Pascal

```
function GetField(I: Integer): IAdapterHiddenField;
```

C++

```
__fastcall IAdapterHiddenField GetField(int I);
```

Description

This is GetField, a member of class IAdapterHiddenFields.

135.2.12.2.2 IAdapterHiddenFields.GetFieldCount Method

Pascal

```
function GetFieldCount: Integer;
```

C++

```
__fastcall int GetFieldCount();
```

Description

This is GetFieldCount, a member of class IAdapterHiddenFields.

135.2.12.3 IAdapterHiddenFields Properties

The properties of the IAdapterHiddenFields class are listed here.

Public Properties

	Name	Description
R	FieldCount (see page 19673)	This is FieldCount, a member of class IAdapterHiddenFields.
R	Fields (see page 19673)	This is Fields, a member of class IAdapterHiddenFields.

135.2.12.3.1 IAdapterHiddenFields.FieldCount Property

Pascal

```
property FieldCount: Integer;
```

C++

```
__property int FieldCount;
```

Description

This is FieldCount, a member of class IAdapterHiddenFields.

135.2.12.3.2 IAdapterHiddenFields.Fields Property

Pascal

```
property Fields [I: Integer]: IAdapterHiddenField;
```

C++

```
__property IAdapterHiddenField Fields[int I];
```

Description

This is Fields, a member of class IAdapterHiddenFields.

135.2.13 IAdapterItemRequestParams Interface

IAdapterItemRequestParams is the interface for reading the parameters for an adapter field or action request.

Class Hierarchy

SiteComp.IAdapterItemRequestParams

Pascal

```
IAdapterItemRequestParams = interface;
```

C++

```
__interface IAdapterItemRequestParams;
```

File

SiteComp

Description

Use IAdapterItemRequestParams to read the parameters that accompany requests to an adapter field or action. The parameters, which can be accessed through the ParamValues (see page 19677) property, are a set of 'Name=Value' strings. In addition to providing access to the parameters, IAdapterItemRequestParams has two properties, SuccessPage (see page 19677) and FailurePage (see page 19676), which identify the Web pages that should be returned to the caller when the adapter request succeeds or fails.

Note: In C++, method declarations that use IAdapterItemRequestParams use the _di_IAdapterItemRequestParams type instead. This type is a DelphiInterface wrapper around the IAdapterItemRequestParams interface:

Note: typedef System::DelphiInterface (see page 22167)< IAdapterItemRequestParams > _di_IAdapterItemRequestParams;

See Also

TPageDispatcher (see page 24949)

TCustomWebAppComponents (see page 24906)

TWebAPpComponents

135.2.13.1 IAdapterItemRequestParams Members

The following tables list the members exposed by IAdapterItemRequestParams.

Public Methods

	Name	Description
	GetFailurePage (see page 19674)	Returns the value of the FailurePage (see page 19676) property.
	GetParamValues (see page 19675)	Returns the value of the ParamValues (see page 19677) property.
	GetSuccessPage (see page 19675)	Returns the value of the SuccessPage (see page 19677) property.
	SetFailurePage (see page 19675)	Sets the value of the FailurePage (see page 19676) property.
	SetSuccessPage (see page 19676)	Sets the value of the SuccessPage (see page 19677) property.

Public Properties

	Name	Description
	FailurePage (see page 19676)	Indicates the Web page module that the application should return if the adapter request fails.
R	ParamValues (see page 19677)	Lists the parameter values that are included in requests to the adapter action or field.
	SuccessPage (see page 19677)	Indicates the Web page module that the application should return if the adapter request succeeds.

135.2.13.2 IAdapterItemRequestParams Methods

The methods of the IAdapterItemRequestParams class are listed here.

Public Methods

	Name	Description
	GetFailurePage (see page 19674)	Returns the value of the FailurePage (see page 19676) property.
	GetParamValues (see page 19675)	Returns the value of the ParamValues (see page 19677) property.
	GetSuccessPage (see page 19675)	Returns the value of the SuccessPage (see page 19677) property.
	SetFailurePage (see page 19675)	Sets the value of the FailurePage (see page 19676) property.
	SetSuccessPage (see page 19676)	Sets the value of the SuccessPage (see page 19677) property.

135.2.13.2.1 IAdapterItemRequestParams.GetFailurePage Method

Returns the value of the FailurePage (see page 19676) property.

Pascal

```
function GetFailurePage: string;
```

C++

```
__fastcall AnsiString GetFailurePage();
```

Description

GetFailurePage is the read implementation of the FailurePage (see page 19676) property.

See Also

FailurePage (see page 19676)

135.2.13.2.2 IAdapterItemRequestParams.GetParamValues Method

Returns the value of the ParamValues (see page 19677) property.

Pascal

```
function GetParamValues: TStrings;
```

C++

```
__fastcall TStrings GetParamValues();
```

Description

GetParamValues is the read implementation of the ParamValues (see page 19677) property.

See Also

ParamValues (see page 19677)

135.2.13.2.3 IAdapterItemRequestParams.GetSuccessPage Method

Returns the value of the SuccessPage (see page 19677) property.

Pascal

```
function GetSuccessPage: string;
```

C++

```
__fastcall AnsiString GetSuccessPage();
```

Description

GetSuccessPage is the read implementation of the SuccessPage (see page 19677) property.

See Also

SuccessPage (see page 19677)

135.2.13.2.4 IAdapterItemRequestParams.SetFailurePage Method

Sets the value of the FailurePage (see page 19676) property.

Pascal

```
procedure SetFailurePage(const AValue: string);
```

C++

```
__fastcall SetFailurePage(const AnsiString AValue);
```

Description

SetFailurePage is the write implementation of the FailurePage (see page 19676) property.

AValue is the name of the Web page that should be returned if the adapter request fails.

See Also

FailurePage (see page 19676)

135.2.13.2.5 IAdapterItemRequestParams.SetSuccessPage Method

Sets the value of the SuccessPage (see page 19677) property.

Pascal

```
procedure SetSuccessPage(const AValue: string);
```

C++

```
__fastcall SetSuccessPage(const AnsiString AValue);
```

Description

SetSuccessPage is the write implementation of the SuccessPage (see page 19677) property.

AValue is the name of the Web page that should be returned when the adapter request succeeds.

See Also

SuccessPage (see page 19677)

135.2.13.3 IAdapterItemRequestParams Properties

The properties of the IAdapterItemRequestParams class are listed here.

Public Properties

	Name	Description
	FailurePage (see page 19676)	Indicates the Web page module that the application should return if the adapter request fails.
R	ParamValues (see page 19677)	Lists the parameter values that are included in requests to the adapter action or field.
	SuccessPage (see page 19677)	Indicates the Web page module that the application should return if the adapter request succeeds.

135.2.13.3.1 IAdapterItemRequestParams.FailurePage Property

Indicates the Web page module that the application should return if the adapter request fails.

Pascal

```
property FailurePage: string;
```

C++

```
__property AnsiString FailurePage;
```

Description

FailurePage is the name of the registered Web page module that should generate the content of the response message if the request to the adapter fails.

See Also

SuccessPage (see page 19677)

135.2.13.3.2 IAdapterItemRequestParams.ParamValues Property

Lists the parameter values that are included in requests to the adapter action or field.

Pascal

```
property ParamValues: TStrings;
```

C++

```
__property TStrings ParamValues;
```

Description

Use ParamValues to access the list of parameter values. Use the Names property of ParamValues to obtain the names of the parameters and the Values property to access the corresponding values.

See Also

Names (see page 3075)

Values (see page 3086)

135.2.13.3.3 IAdapterItemRequestParams.SuccessPage Property

Indicates the Web page module that the application should return if the adapter request succeeds.

Pascal

```
property SuccessPage: string;
```

C++

```
__property AnsiString SuccessPage;
```

Description

SuccessPage is the name of the registered Web page module that should generate the content of the response message when the request to the adapter succeeds.

See Also

SuccessPage

135.2.14 IAdapterMode Interface

Class Hierarchy

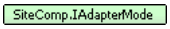

Pascal

```
IAdapterMode = interface;
```

C++

```
__interface IAdapterMode;
```

File

SiteComp

Description

This is class SiteComp.IAdapterMode.

135.2.14.1 IAdapterMode Members

The following tables list the members exposed by IAdapterMode.

Public Methods

	Name	Description
	GetAdapterMode (see page 19678)	This is GetAdapterMode, a member of class IAdapterMode.
	GetNeedAdapterMode (see page 19678)	This is GetNeedAdapterMode, a member of class IAdapterMode.
	RestoreDefaultAdapterMode (see page 19679)	This is RestoreDefaultAdapterMode, a member of class IAdapterMode.
	SetAdapterMode (see page 19679)	This is SetAdapterMode, a member of class IAdapterMode.

Public Properties

	Name	Description
	AdapterMode (see page 19679)	This is AdapterMode, a member of class IAdapterMode.
R	NeedAdapterMode (see page 19679)	This is NeedAdapterMode, a member of class IAdapterMode.

135.2.14.2 IAdapterMode Methods

The methods of the IAdapterMode class are listed here.

Public Methods

	Name	Description
	GetAdapterMode (see page 19678)	This is GetAdapterMode, a member of class IAdapterMode.
	GetNeedAdapterMode (see page 19678)	This is GetNeedAdapterMode, a member of class IAdapterMode.
	RestoreDefaultAdapterMode (see page 19679)	This is RestoreDefaultAdapterMode, a member of class IAdapterMode.
	SetAdapterMode (see page 19679)	This is SetAdapterMode, a member of class IAdapterMode.

135.2.14.2.1 IAdapterMode.GetAdapterMode Method

Pascal

```
function GetAdapterMode: string;
```

C++

```
__fastcall AnsiString GetAdapterMode();
```

Description

This is GetAdapterMode, a member of class IAdapterMode.

135.2.14.2.2 IAdapterMode.GetNeedAdapterMode Method

Pascal

```
function GetNeedAdapterMode: Boolean;
```

C++

```
__fastcall Boolean GetNeedAdapterMode();
```

Description

This is GetNeedAdapterMode, a member of class IAdapterMode.

135.2.14.2.3 IAdapterMode.RestoreDefaultAdapterMode Method

Pascal

```
procedure RestoreDefaultAdapterMode;
```

C++

```
__fastcall RestoreDefaultAdapterMode();
```

Description

This is RestoreDefaultAdapterMode, a member of class IAdapterMode.

135.2.14.2.4 IAdapterMode.SetAdapterMode Method

Pascal

```
procedure SetAdapterMode(const AValue: string);
```

C++

```
__fastcall SetAdapterMode(const AnsiString AValue);
```

Description

This is SetAdapterMode, a member of class IAdapterMode.

135.2.14.3 IAdapterMode Properties

The properties of the IAdapterMode class are listed here.

Public Properties

	Name	Description
	AdapterMode (see page 19679)	This is AdapterMode, a member of class IAdapterMode.
R	NeedAdapterMode (see page 19679)	This is NeedAdapterMode, a member of class IAdapterMode.

135.2.14.3.1 IAdapterMode.AdapterMode Property

Pascal

```
property AdapterMode: string;
```

C++

```
__property AnsiString AdapterMode;
```

Description

This is AdapterMode, a member of class IAdapterMode.

135.2.14.3.2 IAdapterMode.NeedAdapterMode Property

Pascal

```
property NeedAdapterMode: Boolean;
```

C++

```
__property Boolean NeedAdapterMode;
```

Description

This is NeedAdapterMode, a member of class IAdapterMode.

135.2.15 IAdapterRequestHandler Interface

IAdapterRequestHandler handles requests to execute an adapter action or retrieve the value of an image field.

Class Hierarchy

SiteComp.IAdapterRequestHandler

Pascal

```
IAdapterRequestHandler = interface;
```

C++

```
__interface IAdapterRequestHandler;
```

File

SiteComp

Description

IAdapterRequestHandler is implemented by classes that execute an adapter action or look up the value of an image field.

See Also

TBaseAdapterAction (see page 24043)

TCustomAdapterImageField (see page 24244)

135.2.15.1 IAdapterRequestHandler Members

The following tables list the members exposed by IAdapterRequestHandler.

Public Methods

	Name	Description
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.

135.2.15.2 IAdapterRequestHandler Methods

The methods of the IAdapterRequestHandler class are listed here.

Public Methods

	Name	Description
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.

135.2.15.2.1 IAdapterRequestHandler.CreateRequestContext Method

Creates an object that describes the adapter action parameters and adapter field values.

Pascal

```
procedure CreateRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall CreateRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

CreateRequestContext creates a new request context based on the specified parameters.

See Also

AdapterRequest (see page 24655)

TAdapterDispatcher (see page 24840)

CreateRequestContext (see page 24050)

CreateRequestContext (see page 24252)

135.2.15.2.2 IAdapterRequestHandler.HandleRequest Method

Executes the adapter action and generates a response.

Pascal

```
function HandleRequest(DispatchParams: IAdapterDispatchParams): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(IAdapterDispatchParams DispatchParams);
```

Description

The adapter dispatcher calls HandleRequest to execute the adapter action and generate a response.

See Also

HandleRequest (see page 24061)

HandleRequest (see page 24255)

135.2.16 IAsFormatted Interface

Class Hierarchy

SiteComp.IAsFormatted

Pascal

```
IAsFormatted = interface;
```

C++

```
__interface IAsFormatted;
```

File

SiteComp

Description

This is class SiteComp.IAsFormatted.

135.2.16.1 IAsFormatted Members

The following tables list the members exposed by IAsFormatted.

Public Methods

	Name	Description
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.

135.2.16.2 IAsFormatted Methods

The methods of the IAsFormatted class are listed here.

Public Methods

	Name	Description
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.

135.2.16.2.1 IAsFormatted.AsFormatted Method

Pascal

```
function AsFormatted: string;
```

C++

```
__fastcall AnsiString AsFormatted();
```

Description

This is AsFormatted, a member of class IAsFormatted.

135.2.17 IDefaultPageFileName Interface

Class Hierarchy

SiteComp.IDefaultPageFileName

Pascal

```
IDefaultPageFileName = interface;
```

C++

```
__interface IDefaultPageFileName;
```

File

SiteComp

Description

This is class SiteComp.IDefaultPageFileName.

135.2.17.1 IDefaultPageFileName Members

The following tables list the members exposed by IDefaultPageFileName.

Public Methods

	Name	Description
	SetDefaultPageFileName (see page 19683)	This is SetDefaultPageFileName, a member of class IDefaultPageFileName.

135.2.17.2 IDefaultPageFileName Methods

The methods of the IDefaultPageFileName class are listed here.

Public Methods

	Name	Description
	SetDefaultPageFileName (see page 19683)	This is SetDefaultPageFileName, a member of class IDefaultPageFileName.

135.2.17.2.1 IDefaultPageFileName.SetDefaultPageFileName Method

Pascal

```
procedure SetDefaultPageFileName(const APageFile: string);
```

C++

```
__fastcall SetDefaultPageFileName(const AnsiString APageFile);
```

Description

This is SetDefaultPageFileName, a member of class IDefaultPageFileName.

135.2.18 IFieldValuesAdapter Interface

Class Hierarchy

SiteComp.IFieldValuesAdapter

Pascal

```
IFieldValuesAdapter = interface;
```

C++

```
__interface IFieldValuesAdapter;
```

File

SiteComp

Description

This is class SiteComp.IFieldValuesAdapter.

135.2.18.1 IFieldValuesAdapter Members

The following tables list the members exposed by IFieldValuesAdapter.

Public Methods

	Name	Description
	GetField (see page 19684)	This is GetField, a member of class IFieldValuesAdapter.
	GetValueIndex (see page 19684)	This is GetValueIndex, a member of class IFieldValuesAdapter.

135.2.18.2 IFieldValuesAdapter Methods

The methods of the IFieldValuesAdapter class are listed here.

Public Methods

	Name	Description
	GetField (see page 19684)	This is GetField, a member of class IFieldValuesAdapter.
	GetValueIndex (see page 19684)	This is GetValueIndex, a member of class IFieldValuesAdapter.

135.2.18.2.1 IFieldValuesAdapter.GetField Method

Pascal

```
function GetField: TObject;
```

C++

```
__fastcall TObject * GetField();
```

Description

This is GetField, a member of class IFieldValuesAdapter.

135.2.18.2.2 IFieldValuesAdapter.GetValueIndex Method

Pascal

```
function GetValueIndex: Integer;
```

C++

```
__fastcall int GetValueIndex();
```

Description

This is GetValueIndex, a member of class IFieldValuesAdapter.

135.2.19 IFindIncludeFile Interface

Class Hierarchy

SiteComp.IFindIncludeFile

Pascal

```
IFindIncludeFile = interface;
```

C++

```
__interface IFindIncludeFile;
```

File

SiteComp

Description

This is class SiteComp.IFindIncludeFile.

135.2.19.1 IFindIncludeFile Members

The following tables list the members exposed by IFindIncludeFile.

Public Methods

	Name	Description
	FindIncludeFile (see page 19685)	This is FindIncludeFile, a member of class IFindIncludeFile.

135.2.19.2 IFindIncludeFile Methods

The methods of the IFindIncludeFile class are listed here.

Public Methods

	Name	Description
	FindIncludeFile (see page 19685)	This is FindIncludeFile, a member of class IFindIncludeFile.

135.2.19.2.1 IFindIncludeFile.FindIncludeFile Method

Pascal

```
function FindIncludeFile(AComponent: TComponent; const AFileName: string; var AResult:
string): Boolean;
```

C++

```
__fastcall Boolean FindIncludeFile(TComponent * AComponent, const AnsiString AFileName,
AnsiString AResult);
```

Description

This is FindIncludeFile, a member of class IFindIncludeFile.

135.2.20 IGetAdapterActions Interface

Class Hierarchy

SiteComp.IGetAdapterActions

Pascal

```
IGetAdapterActions = interface;
```

C++

```
__interface IGetAdapterActions;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterActions.

135.2.20.1 IGetAdapterActions Members

The following tables list the members exposed by IGetAdapterActions.

Public Methods

	Name	Description
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.

135.2.20.2 IGetAdapterActions Methods

The methods of the IGetAdapterActions class are listed here.

Public Methods

	Name	Description
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.

135.2.20.2.1 IGetAdapterActions.GetAdapterActions Method

Pascal

```
function GetAdapterActions: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterActions();
```

Description

This is GetAdapterActions, a member of class IGetAdapterActions.

135.2.21 IGetAdapterDisplayCharacteristics Interface

Class Hierarchy

SiteComp.IGetAdapterDisplayCharacteristics

Pascal

```
IGetAdapterDisplayCharacteristics = interface;
```

C++

```
__interface IGetAdapterDisplayCharacteristics;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterDisplayCharacteristics.

135.2.21.1 IGetAdapterDisplayCharacteristics Members

The following tables list the members exposed by IGetAdapterDisplayCharacteristics.

Public Methods

	Name	Description
	GetAdapterDisplayCharacteristics (see page 19687)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.

135.2.21.2 IGetAdapterDisplayCharacteristics Methods

The methods of the IGetAdapterDisplayCharacteristics class are listed here.

Public Methods

	Name	Description
	GetAdapterDisplayCharacteristics (see page 19687)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.

135.2.21.2.1 IGetAdapterDisplayCharacteristics.GetAdapterDisplayCharacteristics Method

Pascal

```
function GetAdapterDisplayCharacteristics: TAdapterDisplayCharacteristics;
```

C++

```
__fastcall TAdapterDisplayCharacteristics GetAdapterDisplayCharacteristics();
```

Description

This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.

135.2.22 IGetAdapterErrors Interface

Class Hierarchy

SiteComp.IGetAdapterErrors

Pascal

```
IGetAdapterErrors = interface;
```

C++

```
__interface IGetAdapterErrors;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterErrors.

135.2.22.1 IGetAdapterErrors Members

The following tables list the members exposed by IGetAdapterErrors.

Public Methods

	Name	Description
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.

135.2.22.2 IGetAdapterErrors Methods

The methods of the IGetAdapterErrors class are listed here.

Public Methods

	Name	Description
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.

135.2.22.2.1 IGetAdapterErrors.GetAdapterErrors Method

Pascal

```
function GetAdapterErrors: TObject;
```

C++

```
__fastcall TObject * GetAdapterErrors();
```

Description

This is GetAdapterErrors, a member of class IGetAdapterErrors.

135.2.23 IGetAdapterFields Interface

Class Hierarchy

SiteComp.IGetAdapterFields

Pascal

```
IGetAdapterFields = interface;
```

C++

```
__interface IGetAdapterFields;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterFields.

135.2.23.1 IGetAdapterFields Members

The following tables list the members exposed by IGetAdapterFields.

Public Methods

	Name	Description
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.

135.2.23.2 IGetAdapterFields Methods

The methods of the IGetAdapterFields class are listed here.

Public Methods

	Name	Description
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.

135.2.23.2.1 IGetAdapterFields.GetAdapterFields Method

Pascal

```
function GetAdapterFields: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterFields();
```

Description

This is GetAdapterFields, a member of class IGetAdapterFields.

135.2.24 IGetAdapterHiddenFields Interface

Class Hierarchy

SiteComp.IGetAdapterHiddenFields

Pascal

```
IGetAdapterHiddenFields = interface;
```

C++

```
__interface IGetAdapterHiddenFields;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterHiddenFields.

135.2.24.1 IGetAdapterHiddenFields Members

The following tables list the members exposed by IGetAdapterHiddenFields.

Public Methods

	Name	Description
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.

135.2.24.2 IGetAdapterHiddenFields Methods

The methods of the IGetAdapterHiddenFields class are listed here.

Public Methods

	Name	Description
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.

135.2.24.2.1 IGetAdapterHiddenFields.GetAdapterHiddenFields Method

Pascal

```
function GetAdapterHiddenFields: TObject;
```

C++

```
__fastcall TObject * GetAdapterHiddenFields();
```

Description

This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.

135.2.24.2.2 IGetAdapterHiddenFields.GetAdapterHiddenRecordFields Method

Pascal

```
function GetAdapterHiddenRecordFields: TObject;
```

C++

```
__fastcall TObject * GetAdapterHiddenRecordFields();
```

Description

This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.

135.2.25 IGetAdapterImage Interface

Class Hierarchy

SiteComp.IGetAdapterImage

Pascal

```
IGetAdapterImage = interface;
```

C++

```
__interface IGetAdapterImage;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterImage.

135.2.25.1 IGetAdapterImage Members

The following tables list the members exposed by IGetAdapterImage.

Public Methods

	Name	Description
	GetAdapterImage (see page 19691)	This is GetAdapterImage, a member of class IGetAdapterImage.

135.2.25.2 IGetAdapterImage Methods

The methods of the IGetAdapterImage class are listed here.

Public Methods

	Name	Description
	GetAdapterImage (see page 19691)	This is GetAdapterImage, a member of class IGetAdapterImage.

135.2.25.2.1 IGetAdapterImage.GetAdapterImage Method

Pascal

```
function GetAdapterImage: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterImage();
```

Description

This is GetAdapterImage, a member of class IGetAdapterImage.

135.2.26 IGetAdapterItemRequestParams Interface

Class Hierarchy

SiteComp.IGetAdapterItemRequestParams

Pascal

```
IGetAdapterItemRequestParams = interface;
```

C++

```
__interface IGetAdapterItemRequestParams;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterItemRequestParams.

135.2.26.1 IGetAdapterItemRequestParams Members

The following tables list the members exposed by IGetAdapterItemRequestParams.

Public Methods

	Name	Description
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.

135.2.26.2 IGetAdapterItemRequestParams Methods

The methods of the IGetAdapterItemRequestParams class are listed here.

Public Methods

	Name	Description
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.

135.2.26.2.1 IGetAdapterItemRequestParams.GetAdapterItemRequestParams Method

Pascal

```
procedure GetAdapterItemRequestParams(var AIdentifier: string; AParams:
IAdapterItemRequestParams);
```

C++

```
__fastcall GetAdapterItemRequestParams(AnsiString AIdentifier, IAdapterItemRequestParams
AParams);
```

Description

This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.

135.2.27 IGetAdapterRequestDefaultResponse Interface

Class Hierarchy

SiteComp.IGetAdapterRequestDefaultResponse

Pascal

```
IGetAdapterRequestDefaultResponse = interface;
```

C++

```
__interface IGetAdapterRequestDefaultResponse;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterRequestDefaultResponse.

135.2.27.1 IGetAdapterRequestDefaultResponse Members

The following tables list the members exposed by IGetAdapterRequestDefaultResponse.

Public Methods

	Name	Description
	GetFailurePage (see page 19693)	This is GetFailurePage, a member of class IGetAdapterRequestDefaultResponse.
	GetSuccessPage (see page 19693)	This is GetSuccessPage, a member of class IGetAdapterRequestDefaultResponse.

Public Properties

	Name	Description
R	FailurePage (see page 19694)	This is FailurePage, a member of class IGetAdapterRequestDefaultResponse.
R	SuccessPage (see page 19694)	This is SuccessPage, a member of class IGetAdapterRequestDefaultResponse.

135.2.27.2 IGetAdapterRequestDefaultResponse Methods

The methods of the IGetAdapterRequestDefaultResponse class are listed here.

Public Methods

	Name	Description
	GetFailurePage (see page 19693)	This is GetFailurePage, a member of class IGetAdapterRequestDefaultResponse.
	GetSuccessPage (see page 19693)	This is GetSuccessPage, a member of class IGetAdapterRequestDefaultResponse.

135.2.27.2.1 IGetAdapterRequestDefaultResponse.GetFailurePage Method

Pascal

```
function GetFailurePage: string;
```

C++

```
__fastcall AnsiString GetFailurePage();
```

Description

This is GetFailurePage, a member of class IGetAdapterRequestDefaultResponse.

135.2.27.2.2 IGetAdapterRequestDefaultResponse.GetSuccessPage Method

Pascal

```
function GetSuccessPage: string;
```

C++

```
__fastcall AnsiString GetSuccessPage();
```

Description

This is GetSuccessPage, a member of class IGetAdapterRequestDefaultResponse.

135.2.27.3 IGetAdapterRequestDefaultResponse Properties

The properties of the IGetAdapterRequestDefaultResponse class are listed here.

Public Properties

	Name	Description
R	FailurePage (see page 19694)	This is FailurePage, a member of class IGetAdapterRequestDefaultResponse.
R	SuccessPage (see page 19694)	This is SuccessPage, a member of class IGetAdapterRequestDefaultResponse.

135.2.27.3.1 IGetAdapterRequestDefaultResponse.FailurePage Property

Pascal

```
property FailurePage: string;
```

C++

```
__property AnsiString FailurePage;
```

Description

This is FailurePage, a member of class IGetAdapterRequestDefaultResponse.

135.2.27.3.2 IGetAdapterRequestDefaultResponse.SuccessPage Property

Pascal

```
property SuccessPage: string;
```

C++

```
__property AnsiString SuccessPage;
```

Description

This is SuccessPage, a member of class IGetAdapterRequestDefaultResponse.

135.2.28 IGetAdapterValuesList Interface

Class Hierarchy

SiteComp.IGetAdapterValuesList

Pascal

```
IGetAdapterValuesList = interface;
```

C++

```
__interface IGetAdapterValuesList;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterValuesList.

135.2.28.1 IGetAdapterValuesList Members

The following tables list the members exposed by IGetAdapterValuesList.

Public Methods

	Name	Description
	GetAdapterValuesList (see page 19695)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.

135.2.28.2 IGetAdapterValuesList Methods

The methods of the IGetAdapterValuesList class are listed here.

Public Methods

	Name	Description
	GetAdapterValuesList (see page 19695)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.

135.2.28.2.1 IGetAdapterValuesList.GetAdapterValuesList Method

Pascal

```
function GetAdapterValuesList: IValuesListAdapter;
```

C++

```
__fastcall IValuesListAdapter GetAdapterValuesList();
```

Description

This is GetAdapterValuesList, a member of class IGetAdapterValuesList.

135.2.29 IGetDefaultAction Interface

Class Hierarchy

SiteComp.IGetDefaultAction

Pascal

```
IGetDefaultAction = interface;
```

C++

```
__interface IGetDefaultAction;
```

File

SiteComp

Description

This is class SiteComp.IGetDefaultAction.

135.2.29.1 IGetDefaultAction Members

The following tables list the members exposed by IGetDefaultAction.

Public Methods

	Name	Description
	GetDefaultAction (see page 19696)	This is GetDefaultAction, a member of class IGetDefaultAction.

Public Properties

	Name	Description
R	DefaultAction (see page 19696)	This is DefaultAction, a member of class IGetDefaultAction.

135.2.29.2 IGetDefaultAction Methods

The methods of the IGetDefaultAction class are listed here.

Public Methods

	Name	Description
	GetDefaultAction (see page 19696)	This is GetDefaultAction, a member of class IGetDefaultAction.

135.2.29.2.1 IGetDefaultAction.GetDefaultAction Method

Pascal

```
function GetDefaultAction: TComponent;
```

C++

```
__fastcall TComponent * GetDefaultAction();
```

Description

This is GetDefaultAction, a member of class IGetDefaultAction.

135.2.29.3 IGetDefaultAction Properties

The properties of the IGetDefaultAction class are listed here.

Public Properties

	Name	Description
R	DefaultAction (see page 19696)	This is DefaultAction, a member of class IGetDefaultAction.

135.2.29.3.1 IGetDefaultAction.DefaultAction Property

Pascal

```
property DefaultAction: TComponent;
```

C++

```
__property TComponent * DefaultAction;
```

Description

This is DefaultAction, a member of class IGetDefaultAction.

135.2.30 IGetDesigntimeWarnings Interface

Class Hierarchy

SiteComp.IGetDesigntimeWarnings

Pascal

```
IGetDesigntimeWarnings = interface;
```

C++

```
__interface IGetDesigntimeWarnings;
```

File

SiteComp

Description

This is class SiteComp.IGetDesigntimeWarnings.

135.2.30.1 IGetDesigntimeWarnings Members

The following tables list the members exposed by IGetDesigntimeWarnings.

Public Methods

	Name	Description
	GetDesigntimeWarnings (see page 19697)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.

135.2.30.2 IGetDesigntimeWarnings Methods

The methods of the IGetDesigntimeWarnings class are listed here.

Public Methods

	Name	Description
	GetDesigntimeWarnings (see page 19697)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.

135.2.30.2.1 IGetDesigntimeWarnings.GetDesigntimeWarnings Method

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.

135.2.31 IGetFieldValuesAdapter Interface

Class Hierarchy

SiteComp.IGetFieldValuesAdapter

Pascal

```
IGetFieldValuesAdapter = interface;
```

C++

```
__interface IGetFieldValuesAdapter;
```

File

SiteComp

Description

This is class SiteComp.IGetFieldValuesAdapter.

135.2.31.1 IGetFieldValuesAdapter Members

The following tables list the members exposed by IGetFieldValuesAdapter.

Public Methods

	Name	Description
	GetFieldValuesAdapter (see page 19698)	This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.

135.2.31.2 IGetFieldValuesAdapter Methods

The methods of the IGetFieldValuesAdapter class are listed here.

Public Methods

	Name	Description
	GetFieldValuesAdapter (see page 19698)	This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.

135.2.31.2.1 IGetFieldValuesAdapter.GetFieldValuesAdapter Method

Pascal

```
function GetFieldValuesAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetFieldValuesAdapter();
```

Description

This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.

135.2.32 IGetHTMLStyle Interface

IGetHTMLStyle is used to inquire about the preferred display representation of an adapter field or adapter action.

Class Hierarchy

SiteComp.IGetHTMLStyle

Pascal

```
IGetHTMLStyle = interface;
```

C++

```
__interface IGetHTMLStyle;
```

File

SiteComp

Description

IGetHTMLStyle methods are typically called while generating HTML to display adapter fields and adapter actions. An adapter field implements this interface to indicate, for example, whether it's value should be displayed with an HTML input field or a select list. An adapter action implements this interface to indicate, for example, whether it's value should be displayed with an HTML button or hyperlink.

TAdapterPageProducer child components may call this interface when generating HTML. For example, TCustomAdapterDisplayField calls GetViewMode (see page 19701) when the TCustomAdapterDisplayField's ViewMode property is set to vmDefault. TCustomAdapterActionButton calls GetDisplayStyle (see page 19700) when its DisplayType property is ctDefault.

See Also

TAdapterNamedDisplayField (see page 23913)

TCustomAdapterDisplayColumn (see page 25230)

TCustomAdapterDisplayField (see page 25240)

135.2.32.1 IGetHTMLStyle Members

The following tables list the members exposed by IGetHTMLStyle.

Public Methods

	Name	Description
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.

135.2.32.2 IGetHTMLStyle Methods

The methods of the IGetHTMLStyle class are listed here.

Public Methods

	Name	Description
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.

135.2.32.2.1 IGetHTMLStyle.GetDisplayStyle Method

Returns the preferred display style for an adapter field or adapter action.

Pascal

```
function GetDisplayStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall AnsiString GetDisplayStyle(const AnsiString AAdapterMode);
```

Description

The AAdapterMode parameter can be used to request the style to be used when the adapter is a particular mode (e.g.; Edit).

Adapter fields typically return one of the following values:

Value	Meaning
Text	Display the field as read-only text
Image	Display the field as an image
List	Display the field as a list of read only values
Button	Display the adapter action as a button
Image	Display the adapter action as an image
Anchor	Display the adapter action as a hyperlink
EventImages	Display the adapter action as an image with events that change the image as the mouse is moved.

Adapter actions typically return one of the following values:

Value	Meaning
Button	Display the adapter action as a button
Image	Display the adapter action as an image
Anchor	Display the adapter action as a hyperlink
EventImages	Display the adapter action as an image with events that change the image as the mouse is moved.

User the server-side script DisplayStyle property to get these values from adapter fields and adapter actions.

See Also

DisplayType (see page 25249)

DisplayType (see page 25202)

135.2.32.2.2 IGetHTMLStyle.GetInputStyle Method

Returns the preferred editing style for an adapter field.

Pascal

```
function GetInputStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall AnsiString GetInputStyle(const AnsiString AAdapterMode);
```

Description

The AAdapterMode parameter can be used to request the style to be used when the adapter is a particular mode (e.g.; Browse).

Adapter fields typically return one of the following values:

Value	Meaning
TextInput	Edit the adapter field using a single line input field.
PasswordInput	Edit the adapter field as a password field.
Select	Edit the adapter field using a single selection list.
SelectMultiple	Edit the adapter field using a multiple selection list.
Radio	Edit the adapter field using radio buttons.
CheckBox	Edit the adapter field using check boxes.
TextArea	Edit the adapter field using a multi-line input field.
File	Edit the adapter field using a file upload input field.

Use the server-side script InputStyle property to get these values from adapter fields.

See Also

InputType (see page 25251)

135.2.32.2.3 IGetHTMLStyle.GetViewMode Method

Returns the preferred mode of an adapter field.

Pascal

```
function GetViewMode(const AAdapterMode: string): string;
```

C++

```
__fastcall AnsiString GetViewMode(const AnsiString AAdapterMode);
```

Description

The AAdapterMode parameter can be used to request the style to be used when the adapter is a particular mode (e.g.; Insert).

Adapter fields typically return one of the following values:

Value	Meaning
Input	Display the adapter field as an input field, by default use the style returned by the GetInputStyle (see page 19700) method.
Display	Display the adapter field as a read-only field, by default use the style returned by GetDisplayStyle (see page 19700).
ToggleOnAccess	Display as an input field in the end-user has modify rights, otherwise displays as a read-only field.

Use the server-side script ViewMode property to get these values from adapter fields.

See Also

ViewMode (see page 25252)

135.2.33 IGetProducerComponent Interface

Class Hierarchy

SiteComp.IGetProducerComponent

Pascal

```
IGetProducerComponent = interface;
```

C++

```
__interface IGetProducerComponent;
```

File

SiteComp

Description

This is class SiteComp.IGetProducerComponent.

135.2.33.1 IGetProducerComponent Members

The following tables list the members exposed by IGetProducerComponent.

Public Methods

	Name	Description
	GetProducerComponent (see page 19702)	This is GetProducerComponent, a member of class IGetProducerComponent.

135.2.33.2 IGetProducerComponent Methods

The methods of the IGetProducerComponent class are listed here.

Public Methods

	Name	Description
	GetProducerComponent (see page 19702)	This is GetProducerComponent, a member of class IGetProducerComponent.

135.2.33.2.1 IGetProducerComponent.GetProducerComponent Method

Pascal

```
function GetProducerComponent: TComponent;
```

C++

```
__fastcall TComponent * GetProducerComponent();
```

Description

This is GetProducerComponent, a member of class IGetProducerComponent.

135.2.34 IGetScriptObject Interface

Class Hierarchy

SiteComp.IGetScriptObject

Pascal

```
IGetScriptObject = interface;
```

C++

```
__interface IGetScriptObject;
```

File

SiteComp

Description

This is class SiteComp.IGetScriptObject.

135.2.34.1 IGetScriptObject Members

The following tables list the members exposed by IGetScriptObject.

Public Methods

	Name	Description
	GetScriptObject (see page 19703)	This is GetScriptObject, a member of class IGetScriptObject.

135.2.34.2 IGetScriptObject Methods

The methods of the IGetScriptObject class are listed here.

Public Methods

	Name	Description
	GetScriptObject (see page 19703)	This is GetScriptObject, a member of class IGetScriptObject.

135.2.34.2.1 IGetScriptObject.GetScriptObject Method

Pascal

```
function GetScriptObject: IInterface; deprecated;
```

C++

```
__fastcall IInterface GetScriptObject();
```

Description

This is GetScriptObject, a member of class IGetScriptObject.

135.2.35 IGetScriptObjectClassKey Interface

Class Hierarchy

SiteComp.IGetScriptObjectClassKey

Pascal

```
IGetScriptObjectClassKey = interface;
```

C++

```
__interface IGetScriptObjectClassKey;
```

File

SiteComp

Description

This is class SiteComp.IGetScriptObjectClassKey.

135.2.35.1 IGetScriptObjectClassKey Members

The following tables list the members exposed by IGetScriptObjectClassKey.

Public Methods

	Name	Description
	GetScriptObjectClassKey (see page 19704)	This is GetScriptObjectClassKey, a member of class IGetScriptObjectClassKey.

135.2.35.2 IGetScriptObjectClassKey Methods

The methods of the IGetScriptObjectClassKey class are listed here.

Public Methods

	Name	Description
	GetScriptObjectClassKey (see page 19704)	This is GetScriptObjectClassKey, a member of class IGetScriptObjectClassKey.

135.2.35.2.1 IGetScriptObjectClassKey.GetScriptObjectClassKey Method

Pascal

```
function GetScriptObjectClassKey: TClass; deprecated;
```

C++

```
__fastcall TClass GetScriptObjectClassKey();
```

Description

This is GetScriptObjectClassKey, a member of class IGetScriptObjectClassKey.

135.2.36 IGetWebAppComponents Interface

Class Hierarchy

SiteComp.IGetWebAppComponents

Pascal

```
IGetWebAppComponents = interface;
```

C++

```
__interface IGetWebAppComponents;
```

File

SiteComp

Description

This is class SiteComp.IGetWebAppComponents.

135.2.36.1 IGetWebAppComponents Members

The following tables list the members exposed by IGetWebAppComponents.

Public Methods

	Name	Description
	GetAdapterDispatcher (see page 19706)	This is GetAdapterDispatcher, a member of class IGetWebAppComponents.
	GetLocateFileService (see page 19706)	This is GetLocateFileService, a member of class IGetWebAppComponents.
	GetPageDispatcher (see page 19706)	This is GetPageDispatcher, a member of class IGetWebAppComponents.
	GetWebApplicationInfo (see page 19706)	This is GetWebApplicationInfo, a member of class IGetWebAppComponents.
	GetWebEndUser (see page 19706)	This is GetWebEndUser, a member of class IGetWebAppComponents.
	GetWebUserList (see page 19707)	This is GetWebUserList, a member of class IGetWebAppComponents.

Public Properties

	Name	Description
R	WebApplicationInfo (see page 19707)	This is WebApplicationInfo, a member of class IGetWebAppComponents.
R	WebEndUser (see page 19707)	This is WebEndUser, a member of class IGetWebAppComponents.

135.2.36.2 IGetWebAppComponents Methods

The methods of the IGetWebAppComponents class are listed here.

Public Methods

	Name	Description
	GetAdapterDispatcher (see page 19706)	This is GetAdapterDispatcher, a member of class IGetWebAppComponents.
	GetLocateFileService (see page 19706)	This is GetLocateFileService, a member of class IGetWebAppComponents.
	GetPageDispatcher (see page 19706)	This is GetPageDispatcher, a member of class IGetWebAppComponents.
	GetWebApplicationInfo (see page 19706)	This is GetWebApplicationInfo, a member of class IGetWebAppComponents.
	GetWebEndUser (see page 19706)	This is GetWebEndUser, a member of class IGetWebAppComponents.
	GetWebUserList (see page 19707)	This is GetWebUserList, a member of class IGetWebAppComponents.

135.2.36.2.1 IGetWebAppComponents.GetAdapterDispatcher Method

Pascal

```
function GetAdapterDispatcher: IAdapterDispatcher;
```

C++

```
__fastcall IAdapterDispatcher GetAdapterDispatcher();
```

Description

This is GetAdapterDispatcher, a member of class IGetWebAppComponents.

135.2.36.2.2 IGetWebAppComponents.GetLocateFileService Method

Pascal

```
function GetLocateFileService: ILocateFileService;
```

C++

```
__fastcall ILocateFileService GetLocateFileService();
```

Description

This is GetLocateFileService, a member of class IGetWebAppComponents.

135.2.36.2.3 IGetWebAppComponents.GetPageDispatcher Method

Pascal

```
function GetPageDispatcher: IPageDispatcher;
```

C++

```
__fastcall IPageDispatcher GetPageDispatcher();
```

Description

This is GetPageDispatcher, a member of class IGetWebAppComponents.

135.2.36.2.4 IGetWebAppComponents.GetWebApplicationInfo Method

Pascal

```
function GetWebApplicationInfo: IWebApplicationInfo;
```

C++

```
__fastcall IWebApplicationInfo GetWebApplicationInfo();
```

Description

This is GetWebApplicationInfo, a member of class IGetWebAppComponents.

135.2.36.2.5 IGetWebAppComponents.GetWebEndUser Method

Pascal

```
function GetWebEndUser: IWebEndUser;
```

C++

```
__fastcall IWebEndUser GetWebEndUser();
```

Description

This is GetWebEndUser, a member of class IGetWebAppComponents.

135.2.36.2.6 IGetWebAppComponents.GetWebUserList Method

Pascal

```
function GetWebUserList: IWebUserList;
```

C++

```
__fastcall IWebUserList GetWebUserList();
```

Description

This is GetWebUserList, a member of class IGetWebAppComponents.

135.2.36.3 IGetWebAppComponents Properties

The properties of the IGetWebAppComponents class are listed here.

Public Properties

	Name	Description
	WebApplicationInfo (see page 19707)	This is WebApplicationInfo, a member of class IGetWebAppComponents.
	WebEndUser (see page 19707)	This is WebEndUser, a member of class IGetWebAppComponents.

135.2.36.3.1 IGetWebAppComponents.WebApplicationInfo Property

Pascal

```
property WebApplicationInfo: IWebApplicationInfo;
```

C++

```
__property IWebApplicationInfo WebApplicationInfo;
```

Description

This is WebApplicationInfo, a member of class IGetWebAppComponents.

135.2.36.3.2 IGetWebAppComponents.WebEndUser Property

Pascal

```
property WebEndUser: IWebEndUser;
```

C++

```
__property IWebEndUser WebEndUser;
```

Description

This is WebEndUser, a member of class IGetWebAppComponents.

135.2.37 IIDEFileManager Interface

Class Hierarchy

SiteComp.IIDEFileManager

Pascal

```
IIDEFileManager = interface;
```

C++

```
__interface IIDEFileManager;
```

File

SiteComp

Description

This is class SiteComp.IIDEFileManager.

135.2.37.1 IIDEFileManager Members

The following tables list the members exposed by IIDEFileManager.

Public Methods

	Name	Description
	SearchFile (see page 19708)	This is SearchFile, a member of class IIDEFileManager.

135.2.37.2 IIDEFileManager Methods

The methods of the IIDEFileManager class are listed here.

Public Methods

	Name	Description
	SearchFile (see page 19708)	This is SearchFile, a member of class IIDEFileManager.

135.2.37.2.1 IIDEFileManager.SearchFile Method

Pascal

```
function SearchFile(const AFileName: string): string;
```

C++

```
__fastcall AnsiString SearchFile(const AnsiString AFileName);
```

Description

This is SearchFile, a member of class IIDEFileManager.

135.2.38 IImageProducer Interface

IImageProducer is the interface for accessing the image that an adapter component supplies.

Class Hierarchy

SiteComp.IImageProducer

Pascal

```
IImageProducer = interface;
```

C++

```
__interface IImageProducer;
```

File

SiteComp

Description

IImageProducer is the interface used to access the object that supplies an image to server-side script in a WebSnap application. It defines a single method, GetAdapterImage (see page 19710), which returns the interface of the object that is responsible for providing the image. That object may generate the image, or it may simply generate an HREF that addresses the object that generates the image.

Note that IImageProducer introduces an extra level of indirection when fetching images. IImageProducer is not necessarily the interface of the object that provides the image, it merely retrieves that interface. The reason for this indirection is because the object that implements IImageProducer may work with multiple adapter fields that produce a variety of images, depending on the supplied caption.

Note: In C++, method declarations that use IImageProducer use the _di_IImageProducer type instead. This type is a DelphiInterface wrapper around the IImageProducer interface:

Note: typedef System::DelphiInterface (see page 22167)< IImageProducer > _di_IImageProducer;

See Also

IActionImageProducer (see page 19646)

TDataSetAdapterImageField (see page 8138)

TAdapterImageField (see page 23869)

135.2.38.1 IImageProducer Members

The following tables list the members exposed by IImageProducer.

Public Methods

	Name	Description
	GetAdapterImage (see page 19710)	This is GetAdapterImage, a member of class IImageProducer.

135.2.38.2 IImageProducer Methods

The methods of the IImageProducer class are listed here.

Public Methods

	Name	Description
	GetAdapterImage (see page 19710)	This is GetAdapterImage, a member of class IImageProducer.

135.2.38.2.1 IImageProducer.GetAdapterImage Method

Pascal

```
function GetAdapterImage(const AComponentName: string; const ACaption: WideString): TObject;
```

C++

```
__fastcall TObject * GetAdapterImage(const AnsiString AComponentName, const BSTR ACaption);
```

Description

This is GetAdapterImage, a member of class IImageProducer.

135.2.39 IInternetEnvOptions Interface

Class Hierarchy

SiteComp.IInternetEnvOptions

Pascal

```
IInternetEnvOptions = interface;
```

C++

```
__interface IInternetEnvOptions;
```

File

SiteComp

Description

This is class SiteComp.IInternetEnvOptions.

135.2.39.1 IInternetEnvOptions Members

The following tables list the members exposed by IInternetEnvOptions.

Public Methods

	Name	Description
	GetDisableDebugging (see page 19711)	This is GetDisableDebugging, a member of class IInternetEnvOptions.
	GetHTMLExt (see page 19711)	This is GetHTMLExt, a member of class IInternetEnvOptions.
	GetSampleImageFile (see page 19711)	This is GetSampleImageFile, a member of class IInternetEnvOptions.
	GetScriptTimeoutSecs (see page 19712)	This is GetScriptTimeoutSecs, a member of class IInternetEnvOptions.

Public Properties

	Name	Description
R	DisableDebugger (see page 19712)	This is DisableDebugger, a member of class IInternetEnvOptions.
R	HTMLExt (see page 19712)	This is HTMLExt, a member of class IInternetEnvOptions.
R	SampleImageFile (see page 19712)	This is SampleImageFile, a member of class IInternetEnvOptions.
R	ScriptTimeoutSecs (see page 19713)	This is ScriptTimeoutSecs, a member of class IInternetEnvOptions.

135.2.39.2 IInternetEnvOptions Methods

The methods of the IInternetEnvOptions class are listed here.

Public Methods

	Name	Description
	GetDisableDebugging (see page 19711)	This is GetDisableDebugging, a member of class IInternetEnvOptions.
	GetHTMLExt (see page 19711)	This is GetHTMLExt, a member of class IInternetEnvOptions.
	GetSampleImageFile (see page 19711)	This is GetSampleImageFile, a member of class IInternetEnvOptions.
	GetScriptTimeoutSecs (see page 19712)	This is GetScriptTimeoutSecs, a member of class IInternetEnvOptions.

135.2.39.2.1 IInternetEnvOptions.GetDisableDebugging Method

Pascal

```
function GetDisableDebugging: Boolean;
```

C++

```
__fastcall Boolean GetDisableDebugging();
```

Description

This is GetDisableDebugging, a member of class IInternetEnvOptions.

135.2.39.2.2 IInternetEnvOptions.GetHTMLExt Method

Pascal

```
function GetHTMLExt: string;
```

C++

```
__fastcall AnsiString GetHTMLExt();
```

Description

This is GetHTMLExt, a member of class IInternetEnvOptions.

135.2.39.2.3 IInternetEnvOptions.GetSampleImageFile Method

Pascal

```
function GetSampleImageFile: string;
```

C++

```
__fastcall AnsiString GetSampleImageFile();
```

Description

This is GetSampleImageFile, a member of class IInternetEnvOptions.

135.2.39.2.4 IInternetEnvOptions.GetScriptTimeoutSecs Method

Pascal

```
function GetScriptTimeoutSecs: Integer;
```

C++

```
__fastcall int GetScriptTimeoutSecs();
```

Description

This is GetScriptTimeoutSecs, a member of class IInternetEnvOptions.

135.2.39.3 IInternetEnvOptions Properties

The properties of the IInternetEnvOptions class are listed here.

Public Properties

	Name	Description
R	DisableDebugger (see page 19712)	This is DisableDebugger, a member of class IInternetEnvOptions.
R	HTMLExt (see page 19712)	This is HTMLExt, a member of class IInternetEnvOptions.
R	SampleImageFile (see page 19712)	This is SampleImageFile, a member of class IInternetEnvOptions.
R	ScriptTimeoutSecs (see page 19713)	This is ScriptTimeoutSecs, a member of class IInternetEnvOptions.

135.2.39.3.1 IInternetEnvOptions.DisableDebugger Property

Pascal

```
property DisableDebugger: Boolean;
```

C++

```
__property Boolean DisableDebugger;
```

Description

This is DisableDebugger, a member of class IInternetEnvOptions.

135.2.39.3.2 IInternetEnvOptions.HTMLExt Property

Pascal

```
property HTMLExt: string;
```

C++

```
__property AnsiString HTMLExt;
```

Description

This is HTMLExt, a member of class IInternetEnvOptions.

135.2.39.3.3 IInternetEnvOptions.SampleImageFile Property

Pascal

```
property SampleImageFile: string;
```

C++

```
__property AnsiString SampleImageFile;
```

Description

This is SampleImageFile, a member of class IInternetEnvOptions.

135.2.39.3.4 IInternetEnvOptions.ScriptTimeoutSecs Property

Pascal

```
property ScriptTimeoutSecs: Integer;
```

C++

```
__property int ScriptTimeoutSecs;
```

Description

This is ScriptTimeoutSecs, a member of class IInternetEnvOptions.

135.2.40 IInternetExecuteScript Interface

Class Hierarchy

SiteComp.IInternetExecuteScript

Pascal

```
IInternetExecuteScript = interface;
```

C++

```
__interface IInternetExecuteScript;
```

File

SiteComp

Description

This is class SiteComp.IInternetExecuteScript.

135.2.40.1 IInternetExecuteScript Members

The following tables list the members exposed by IInternetExecuteScript.

Public Methods

	Name	Description
	CanExecuteScript (see page 19714)	This is CanExecuteScript, a member of class IInternetExecuteScript.
	ExecuteScript (see page 19714)	This is ExecuteScript, a member of class IInternetExecuteScript.

135.2.40.2 IInternetExecuteScript Methods

The methods of the IInternetExecuteScript class are listed here.

Public Methods

	Name	Description
	CanExecuteScript (see page 19714)	This is CanExecuteScript, a member of class IInternetExecuteScript.
	ExecuteScript (see page 19714)	This is ExecuteScript, a member of class IInternetExecuteScript.

135.2.40.2.1 IInternetExecuteScript.CanExecuteScript Method

Pascal

```
function CanExecuteScript(AContext: IUnknown): Boolean;
```

C++

```
__fastcall Boolean CanExecuteScript(IUnknown AContext);
```

Description

This is CanExecuteScript, a member of class IInternetExecuteScript.

135.2.40.2.2 IInternetExecuteScript.ExecuteScript Method

Pascal

```
procedure ExecuteScript(AContext: IUnknown);
```

C++

```
__fastcall ExecuteScript(IUnknown AContext);
```

Description

This is ExecuteScript, a member of class IInternetExecuteScript.

135.2.41 IInternetFileTypes Interface

Class Hierarchy

SiteComp.IInternetFileTypes

Pascal

```
IInternetFileTypes = interface;
```

C++

```
__interface IInternetFileTypes;
```

File

SiteComp

Description

This is class SiteComp.IInternetFileTypes.

135.2.41.1 IInternetFileTypes Members

The following tables list the members exposed by IInternetFileTypes.

Public Methods

	Name	Description
	FindFileExt (see page 19715)	This is FindFileExt, a member of class IInternetFileTypes.

135.2.41.2 IInternetFileTypes Methods

The methods of the IInternetFileTypes class are listed here.

Public Methods

	Name	Description
	FindFileExt (see page 19715)	This is FindFileExt, a member of class IInternetFileTypes.

135.2.41.2.1 IInternetFileTypes.FindFileExt Method

Pascal

```
function FindFileExt(const AFileExt: string): TAbstractInetFileType;
```

C++

```
__fastcall TAbstractInetFileType FindFileExt(const AnsiString AFileExt);
```

Description

This is FindFileExt, a member of class IInternetFileTypes.

135.2.42 IInternetScriptDebugger Interface

Class Hierarchy

SiteComp.IInternetScriptDebugger

Pascal

```
IInternetScriptDebugger = interface;
```

C++

```
__interface IInternetScriptDebugger;
```

File

SiteComp

Description

This is class SiteComp.IInternetScriptDebugger.

135.2.42.1 IInternetScriptDebugger Members

The following tables list the members exposed by IInternetScriptDebugger.

Public Methods

	Name	Description
	CanDebugScript (see page 19716)	This is CanDebugScript, a member of class IInternetScriptDebugger.
	DebugScript (see page 19716)	This is DebugScript, a member of class IInternetScriptDebugger.

135.2.42.2 IInternetScriptDebugger Methods

The methods of the IInternetScriptDebugger class are listed here.

Public Methods

	Name	Description
	CanDebugScript (see page 19716)	This is CanDebugScript, a member of class IInternetScriptDebugger.
	DebugScript (see page 19716)	This is DebugScript, a member of class IInternetScriptDebugger.

135.2.42.2.1 IInternetScriptDebugger.CanDebugScript Method

Pascal

```
function CanDebugScript(AContext: IUnknown): Boolean;
```

C++

```
__fastcall Boolean CanDebugScript(IUnknown AContext);
```

Description

This is CanDebugScript, a member of class IInternetScriptDebugger.

135.2.42.2.2 IInternetScriptDebugger.DebugScript Method

Pascal

```
procedure DebugScript(AContext: IUnknown);
```

C++

```
__fastcall DebugScript(IUnknown AContext);
```

Description

This is DebugScript, a member of class IInternetScriptDebugger.

135.2.43 IIsAdapterActionList Interface

Class Hierarchy

SiteComp.IIsAdapterActionList

Pascal

```
IIsAdapterActionList = interface;
```

C++

```
__interface IIsAdapterActionList;
```

File

SiteComp

Description

This is class SiteComp.IIsAdapterActionList.

135.2.43.1 IIsAdapterActionList Members

The following tables list the members exposed by IIsAdapterActionList.

Public Methods

	Name	Description
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.

Public Properties

	Name	Description
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.

135.2.43.2 IIsAdapterActionList Methods

The methods of the IIsAdapterActionList class are listed here.

Public Methods

	Name	Description
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.

135.2.43.2.1 IIsAdapterActionList.GetIsAdapterActionList Method

Pascal

```
function GetIsAdapterActionList: Boolean;
```

C++

```
__fastcall Boolean GetIsAdapterActionList();
```

Description

This is GetIsAdapterActionList, a member of class IIsAdapterActionList.

135.2.43.3 IIsAdapterActionList Properties

The properties of the IIsAdapterActionList class are listed here.

Public Properties

	Name	Description
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.

135.2.43.3.1 IIsAdapterActionList.IsAdapterActionList Property

Pascal

```
property IsAdapterActionList: Boolean;
```

C++

```
__property Boolean IsAdapterActionList;
```

Description

This is IsAdapterActionList, a member of class IIsAdapterActionList.

135.2.44 IIterateIntfSupport Interface

Class Hierarchy

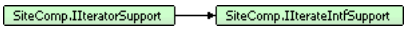

Pascal

```
IIterateIntfSupport = interface(IIteratorSupport);
```

C++

```
__interface IIterateIntfSupport : public IIteratorSupport;
```

File

SiteComp

Description

This is class SiteComp.IIterateIntfSupport.

135.2.44.1 IIterateIntfSupport Members

The following tables list the members exposed by IIterateIntfSupport.

Public Methods

	Name	Description
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	GetCurrentIntf (see page 19718)	This is GetCurrentIntf, a member of class IIterateIntfSupport.
	NextIteration (see page 19723)	Advances to the objects next item.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.

135.2.44.2 IIterateIntfSupport Methods

The methods of the IIterateIntfSupport class are listed here.

Public Methods

	Name	Description
	GetCurrentIntf (see page 19718)	This is GetCurrentIntf, a member of class IIterateIntfSupport.

135.2.44.2.1 IIterateIntfSupport.GetCurrentIntf Method

Pascal

```
function GetCurrentIntf(OwnerData: Pointer): IUnknown;
```

C++

```
__fastcall IUnknown GetCurrentIntf(void * OwnerData);
```

Description

This is GetCurrentIntf, a member of class IIterateIntfSupport.

135.2.45 IIterateObjectSupport Interface

Class Hierarchy

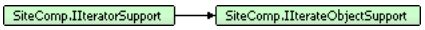

Pascal

```
IIterateObjectSupport = interface(IIteratorSupport);
```

C++

```
__interface IIterateObjectSupport : public IIteratorSupport;
```

File

SiteComp

Description

This is class SiteComp.IIterateObjectSupport.

135.2.45.1 IIterateObjectSupport Members

The following tables list the members exposed by IIterateObjectSupport.

Public Methods

	Name	Description
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	GetCurrentObject (see page 19719)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	NextIteration (see page 19723)	Advances to the objects next item.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.

135.2.45.2 IIterateObjectSupport Methods

The methods of the IIterateObjectSupport class are listed here.

Public Methods

	Name	Description
	GetCurrentObject (see page 19719)	This is GetCurrentObject, a member of class IIterateObjectSupport.

135.2.45.2.1 IIterateObjectSupport.GetCurrentObject Method

Pascal

```
function GetCurrentObject(OwnerData: Pointer): TObject;
```

C++

```
__fastcall TObject * GetCurrentObject(void * OwnerData);
```

Description

This is GetCurrentObject, a member of class IIterateObjectSupport.

135.2.46 IIteratorIndex Interface

IIteratorIndex is the interface that indicates where an object is positioned when it is iterating over its subitems.

Class Hierarchy

SiteComp.IIteratorIndex

Pascal

```
IIteratorIndex = interface;
```

C++

```
__interface IIteratorIndex;
```

File

SiteComp

Description

Server-side script uses the IIteratorIndex interface to detect when an object is in the middle of an enumeration, and if it is, to determine the item on which the object is positioned. This interface is used in conjunction with the IIteratorSupport (see page 19722) interface to enable server-side script to iterate over the items of an adapter or one of its sub-components.

Note: In C++, method declarations that use IIteratorIndex use the _di_IIteratorIndex type instead. This type is a DelphiInterface wrapper around the IIteratorIndex interface:

Note: typedef System::DelphiInterface (see page 22167)< IIteratorIndex > _di_IIteratorIndex;

See Also

IIteratorSupport (see page 19722)

135.2.46.1 IIteratorIndex Members

The following tables list the members exposed by IIteratorIndex.

Public Methods

	Name	Description
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.

Public Properties

	Name	Description
R	IteratorIndex (see page 19721)	Indicates the index of the current item.

135.2.46.2 IIteratorIndex Methods

The methods of the IIteratorIndex class are listed here.

Public Methods

	Name	Description
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.

135.2.46.2.1 IIteratorIndex.GetIteratorIndex Method

Returns the value of the IteratorIndex (see page 19721) property.

Pascal

```
function GetIteratorIndex: Integer;
```

C++

```
__fastcall int GetIteratorIndex();
```

Description

GetIteratorIndex is the read implementation for the IteratorIndex (see page 19721) property.

See Also

IteratorIndex (see page 19721)

135.2.46.2.2 IIteratorIndex.InIterator Method

Indicates whether the object is iterating over its items.

Pascal

```
function InIterator: Boolean;
```

C++

```
__fastcall Boolean InIterator();
```

Description

InIterator returns true if the object is iterating over its items. It returns false if an iteration never started or if an iteration process finished. When InIterator returns false, the current position of the object is indeterminate. In this case, the IteratorIndex (see page 19721) property's value can't be used.

See Also

InIterator (see page 24172)

IIteratorSupport (see page 19722)

135.2.46.3 IIteratorIndex Properties

The properties of the IIteratorIndex class are listed here.

Public Properties

	Name	Description
R	IteratorIndex (see page 19721)	Indicates the index of the current item.

135.2.46.3.1 IIteratorIndex.IteratorIndex Property

Indicates the index of the current item.

Pascal

```
property IteratorIndex: Integer;
```

C++

```
__property int IteratorIndex;
```

Description

IteratorIndex indicates the index of the current item when the object is iterating over its items. When IteratorIndex is 0, the object is positioned on its first item; When IteratorIndex is 1, the object is positioned on its second item, and so on.

When the object is not iterating over its items (when the InIterator (see page 19721) method returns false), the value of IteratorIndex is undefined.

See Also

InIterator (see page 19721)

135.2.47 IIteratorSupport Interface

IIteratorSupport is the interface that allows server-side script to iterate over the items of an adapter or one of its child objects.

Class Hierarchy

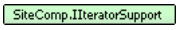

Pascal

```
IIteratorSupport = interface;
```

C++

```
__interface IIteratorSupport;
```

File

SiteComp

Description

Server-side script uses the IIteratorSupport interface to enumerate the items of an adapter or one of its sub-components. When the server-side script includes an enumerator, the WebSnap application automatically generates calls to the IIteratorSupport methods of the object that contains the items to enumerate.

The WebSnap application first calls the StartIterator (see page 19724) method to position the object on its first item. To advance to the next item, the application calls the NextIteration (see page 19723) method. Finally, the application calls the EndIterator (see page 19723) method to signal the end of the iteration.

Each object that implements IIteratorSupport determines the items over which it iterates. For example, if the object is an adapter, these can be records or pages that the adapter uses to group information, or name/value pairs that the adapter represents. If the object is a list of fields or actions, these can be the fields or actions in the list. If the object is an adapter field, these can be the possible values of the field. If the object is an adapter action, these can be the commands the action can execute.

Note: In C++, method declarations that use IIteratorSupport use the _di_IIteratorSupport type instead. This type is a DelphiInterface wrapper around the IIteratorSupport interface:

Note: typedef System::DelphiInterface (see page 22167)< IIteratorSupport > _di_IIteratorSupport;

See Also

IIteratorIndex (see page 19720)

135.2.47.1 IIteratorSupport Members

The following tables list the members exposed by IIteratorSupport.

Public Methods

	Name	Description
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	NextIteration (see page 19723)	Advances to the objects next item.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.

135.2.47.2 IIteratorSupport Methods

The methods of the IIteratorSupport class are listed here.

Public Methods

	Name	Description
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	NextIteration (see page 19723)	Advances to the objects next item.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.

135.2.47.2.1 IIteratorSupport.EndIterator Method

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

This is EndIterator, a member of class IIteratorSupport.

135.2.47.2.2 IIteratorSupport.NextIteration Method

Advances to the objects next item.

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

Server-side script calls NextIteration to move to the next item.

OwnerData is a pointer to information that the object can use to influence the iteration process. Each object stores and uses this information in its own way. This information is first supplied by the initial call to the StartIterator (see page 19724) method. NextIteration can modify the information, after which it is supplied to the next call to the NextIteration or EndIterator (see page 19723) method.

NextIteration returns true if it successfully moves to the next item, false if there were no more items.

See Also

startiterator (see page 19724)

enditerator (see page 19723)

135.2.47.2.3 IIteratorSupport.StartIterator Method

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

This is StartIterator, a member of class IIteratorSupport.

135.2.48 INotifyAdapterChange Interface

INotifyAdapterChange is the interface that allows objects to respond when an adapter changes.

Class Hierarchy

SiteComp.INotifyAdapterChange

Pascal

```
INotifyAdapterChange = interface;
```

C++

```
__interface INotifyAdapterChange;
```

File

SiteComp

Description

Objects that depend on the fields, actions, or properties of an adapter implement the INotifyAdapterChange interface so that the adapter can notify them of changes. They register an interest in such notifications using the adapter's INotifyList (see page 19725) interface. Then, when the adapter changes, it uses this interface to inform them of the change. They can remove themselves from the adapter's notification list using the INotifyList (see page 19725) interface.

See Also

TCustomAdapter

TCustomAdapter

INotifyList (see page 19725)

135.2.48.1 INotifyAdapterChange Members

The following tables list the members exposed by INotifyAdapterChange.

Public Methods

	Name	Description
	NotifyAdapterChange (see page 19725)	Responds when the adapter changes.

135.2.48.2 INotifyAdapterChange Methods

The methods of the INotifyAdapterChange class are listed here.

Public Methods

	Name	Description
	NotifyAdapterChange (see page 19725)	Responds when the adapter changes.

135.2.48.2.1 INotifyAdapterChange.NotifyAdapterChange Method

Responds when the adapter changes.

Pascal

```
procedure NotifyAdapterChange(Sender: TComponent);
```

C++

```
__fastcall NotifyAdapterChange(TComponent * Sender);
```

Description

NotifyAdapterChange is the method objects implement to respond when the associated adapter changes.

Sender is the adapter component that changed.

135.2.49 INotifyList Interface

INotifyList is the interface that allows objects to register and unregister an interest in changes to a component.

Class Hierarchy

SiteComp.INotifyList

Pascal

```
INotifyList = interface;
```

C++

```
__interface INotifyList;
```

File

SiteComp

Description

WebSnap components that let other objects register interest in their changes implement INotifyList. When another object wants to be informed of changes, it calls the AddNotify (see page 19726) method. Then, when the component that implements INotifyList changes, it informs all of the objects that have called AddNotify (see page 19726) of the change. When an object wants to unregister its interest, it calls the RemoveNotify (see page 19726) method.

See Also

TNotifyList (see page 19636)

135.2.49.1 INotifyList Members

The following tables list the members exposed by INotifyList.

Public Methods

	Name	Description
	AddNotify (see page 19726)	Registers an interest in change notifications.
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.

135.2.49.2 INotifyList Methods

The methods of the INotifyList class are listed here.

Public Methods

	Name	Description
	AddNotify (see page 19726)	Registers an interest in change notifications.
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.

135.2.49.2.1 INotifyList.AddNotify Method

Registers an interest in change notifications.

Pascal

```
procedure AddNotify(ANotify: TObject);
```

C++

```
__fastcall AddNotify(TObject * ANotify);
```

Description

Call AddNotify to register an interest in change notifications. After a call to AddNotify, the object is notified every time the component changes, until it unregisters its interest by a call to the RemoveNotify (see page 19726) method.

ANotify is the object to notify of changes. It should implement an interface that allows the notifying component to call it when changes occur. What interface this is depends on the component that implements INotifyList (see page 19725).

See Also

RemoveNotify (see page 19726)

135.2.49.2.2 INotifyList.RemoveNotify Method

Unregisters interest in change notifications.

Pascal

```
procedure RemoveNotify(ANotify: TObject);
```

C++

```
__fastcall RemoveNotify(TObject * ANotify);
```

Description

Call RemoveNotify to unregister interest in change notifications. After a call to RemoveNotify, the object is no longer notified when the component changes.

ANotify is the object to unregister. It must be an object that previously registered interest by calling the AddNotify (see page 19726) method.

See Also

AddNotify (see page 19726)

135.2.50 IPageDispatcher Interface

IPageDispatcher is the interface for interacting with the page dispatcher.

Class Hierarchy

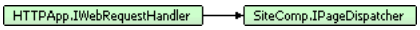

Pascal

```
IPageDispatcher = interface(IWebRequestHandler);
```

C++

```
__interface IPageDispatcher : public IWebRequestHandler;
```

File

SiteComp

Description

The TPageDispatcher component implements IPageDispatcher. This interface, which the TWebAppComponents object publishes, allows the WebSnap application to dispatch request messages. The request messages identify a target Web page using the pathinfo portion of their target URL. If the request is not addressed to an URL with a pathinfo portion, the IPageDispatcher forwards the request to the Web page module identified by the DefaultPageName (see page 19731) property.

In addition to dispatching requests for Web pages, IPageDispatcher includes methods that check whether the current user needs to log in, check whether the current user has rights to view the requested page, and to redirect requests.

Note: In C++, method declarations that use IPageDispatcher use the _di_IPageDispatcher type instead. This type is a DelphiInterface wrapper around the IPageDispatcher interface:

Note: typedef System::DelphiInterface (see page 22167)< IPageDispatcher > _di_IPageDispatcher;

See Also

TPageDispatcher (see page 24949)

TCustomWebAppComponents (see page 24906)

TWebAPpComponents

135.2.50.1 IPageDispatcher Members

The following tables list the members exposed by IPageDispatcher.

Public Methods

	Name	Description
	DispatchPageName (see page 19728)	Checks whether a specified user has a given access rights string.
	GetCanView (see page 19729)	Indicates whether the current user is permitted to view a specified Web page.
	GetDefaultPageName (see page 19729)	Returns the value of the DefaultPageName (see page 19731) property.
	GetLoginRequired (see page 19729)	Indicates whether a user must log in before accessing a specified Web page.
	HandleRequest (see page 15238)	Processes an HTTP request message.
	PageNameOfRequest (see page 19730)	Returns the name of the Web page to which an HTTP request is addressed.
	RedirectToPageName (see page 19730)	Fills out a response message so that it redirects the caller to another page in the application.

Public Properties

	Name	Description
R	DefaultPageName (see page 19731)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

135.2.50.2 IPageDispatcher Methods

The methods of the IPageDispatcher class are listed here.

Public Methods

	Name	Description
	DispatchPageName (see page 19728)	Checks whether a specified user has a given access rights string.
	GetCanView (see page 19729)	Indicates whether the current user is permitted to view a specified Web page.
	GetDefaultPageName (see page 19729)	Returns the value of the DefaultPageName (see page 19731) property.
	GetLoginRequired (see page 19729)	Indicates whether a user must log in before accessing a specified Web page.
	PageNameOfRequest (see page 19730)	Returns the name of the Web page to which an HTTP request is addressed.
	RedirectToPageName (see page 19730)	Fills out a response message so that it redirects the caller to another page in the application.

135.2.50.2.1 IPageDispatcher.DispatchPageName Method

Checks whether a specified user has a given access rights string.

Pascal

```
function DispatchPageName(const APageName: string; AResponse: TWebResponse; AFlags:
TDispatchPageFlags): Boolean;
```

C++

```
__fastcall Boolean DispatchPageName(const AnsiString APageName, TWebResponse AResponse,
TDispatchPageFlags AFlags);
```

Description

Call DispatchPageName to fill out a response message with the Web page that a specified Web page module provides. DispatchPageName locates the Web page module that produces the requested Web page and uses it to fill out the content of a response object.

APageName is the name of the Web page to fetch. If APageName is an empty string, DispatchPageName uses the first Web page module that is registered with the current Web context.

AResponse is the response object whose Content property receives the requested Web page.

AFlags indicates the attributes that must be present on the Web page module. For example, if AFlags includes dpPublished, APageName must specify a published Web page module or the page dispatcher won't fetch the Web page.

DispatchPageName returns true if it successfully fetches the requested page, false if it fails to obtain the requested page.

See Also

RedirectToPageName (see page 19730)

HandleRequest (see page 15238)

Content (see page 15207)

WebContext (see page 24687)

135.2.50.2.2 IPageDispatcher.GetCanView Method

Indicates whether the current user is permitted to view a specified Web page.

Pascal

```
function GetCanView(const APageName: string): Boolean;
```

C++

```
__fastcall Boolean GetCanView(const AnsiString APageName);
```

Description

Call GetCanView to check whether the current user should be allowed to view a specified Web page. Typically, this method is called after the user logs in.

APageName is the name of the Web page the user wants to view.

See Also

GetLoginRequired (see page 19729)

HasViewRights (see page 24676)

OnCanViewPage (see page 24903)

135.2.50.2.3 IPageDispatcher.GetDefaultPageName Method

Returns the value of the DefaultPageName (see page 19731) property.

Pascal

```
function GetDefaultPageName: string;
```

C++

```
__fastcall AnsiString GetDefaultPageName();
```

Description

GetDefaultPageName is the read implementation of the DefaultPageName (see page 19731) property.

See Also

DefaultPageName (see page 19731)

135.2.50.2.4 IPageDispatcher.GetLoginRequired Method

Indicates whether a user must log in before accessing a specified Web page.

Pascal

```
function GetLoginRequired(const APageName: string): Boolean;
```

C++

```
__fastcall Boolean GetLoginRequired(const AnsiString APageName);
```

Description

Call GetLoginRequired to check whether users must log in before accessing a specified Web page. GetLoginRequired can check the Web page information object for the target Web page, but can also use other means (such as an event handler) to obtain this information.

APageName is the name of the Web page to check whether it requires users to first log in.

See Also

GetCanView (see page 19729)

IsLoginRequired

OnIsLoginRequired (see page 24904)

135.2.50.2.5 IPageDispatcher.PageNameOfRequest Method

Returns the name of the Web page to which an HTTP request is addressed.

Pascal

```
function PageNameOfRequest(ARequest: TWebRequest): string;
```

C++

```
__fastcall AnsiString PageNameOfRequest(TWebRequest ARequest);
```

Description

Call PageNameOfRequest to obtain the name of the Web page that a request message is trying to fetch.

ARequest is the request object that describes the HTTP request message.

PageNameOfRequest returns the last string in the path info portion of the request message's target URL.

See Also

DefaultPageName (see page 19731)

135.2.50.2.6 IPageDispatcher.RedirectToPageName Method

Fills out a response message so that it redirects the caller to another page in the application.

Pascal

```
function RedirectToPageName(const APageName: string; AParams: TStrings; AResponse:
TWebResponse; AFlags: TDispatchPageFlags): Boolean;
```

C++

```
__fastcall Boolean RedirectToPageName(const AnsiString APageName, TStrings AParams,
TWebResponse AResponse, TDispatchPageFlags AFlags);
```

Description

Call RedirectToPageName to generate a response that redirects the caller to another Web page in the WebSnap application.

APageName is the name of the Web page to which the request should be redirected. If APageName is an empty string, RedirectToPageName uses the first Web page module that is registered with the current Web context.

AParams is a string list containing application-defined name-value pairs. These name-value pairs supply the query fields of the redirected request message, and are added to the URL that is the target of the SendRedirect call.

AResponse is the response object that the WebSnap uses to make the SendRedirect call.

AFlags indicates the attributes that must be present on the Web page module to which the current request is redirected. For example, if AFlags includes dpPublished, APageName must specify a published Web page module.

RedirectToPageName returns true if it successfully redirects to the specified page.

See Also

DispatchPageName (see page 19728)

HandleRequest (see page 15238)

Content (see page 15207)

WebContext (see page 24687)

135.2.50.3 IPageDispatcher Properties

The properties of the IPageDispatcher class are listed here.

Public Properties

	Name	Description
R	DefaultPageName (see page 19731)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

135.2.50.3.1 IPageDispatcher.DefaultPageName Property

Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

Pascal

```
property DefaultPageName: string;
```

C++

```
__property AnsiString DefaultPageName;
```

Description

DefaultPageName is the name of the registered Web page module that should handle incoming request messages that do not have a pathinfo portion on their target URL. This is the name that is supplied to the Web page module wizard when you created the default Web page module. It forms the basis of that Web page module's class name. For example, if the default Web page module has the name MyDefaultPage, the Web page module class is of type TMyDefaultPage.

If DefaultPageName is an empty string, the page dispatcher uses the Web application module as a default Web page module, as long as it is a TWebAppPageModule object.

See Also

DispatchPageName (see page 19728)

TWebAppPageModule (see page 25394)

135.2.51 IPageResult Interface

Class Hierarchy

SiteComp.IPageResult

Pascal

```
IPageResult = interface;
```

C++

```
__interface IPageResult;
```

File

SiteComp

Description

This is class SiteComp.IPageResult.

135.2.51.1 IPageResult Members

The following tables list the members exposed by IPageResult.

Public Methods

	Name	Description
	DispatchPage (see page 19732)	This is DispatchPage, a member of class IPageResult.
	IncludePage (see page 19733)	This is IncludePage, a member of class IPageResult.
	RedirectToPage (see page 19733)	This is RedirectToPage, a member of class IPageResult.

135.2.51.2 IPageResult Methods

The methods of the IPageResult class are listed here.

Public Methods

	Name	Description
	DispatchPage (see page 19732)	This is DispatchPage, a member of class IPageResult.
	IncludePage (see page 19733)	This is IncludePage, a member of class IPageResult.
	RedirectToPage (see page 19733)	This is RedirectToPage, a member of class IPageResult.

135.2.51.2.1 IPageResult.DispatchPage Method

Pascal

```
function DispatchPage(const APageName: string; AWebResponse: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean DispatchPage(const AnsiString APageName, TWebResponse AWebResponse);
```

Description

This is DispatchPage, a member of class IPageResult.

135.2.51.2.2 IPageResult.IncludePage Method

Pascal

```
function IncludePage(const APageName: string; var AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream IncludePage(const AnsiString APageName, Boolean AOwned);
```

Description

This is IncludePage, a member of class IPageResult.

135.2.51.2.3 IPageResult.RedirectToPage Method

Pascal

```
function RedirectToPage(const APageName: string; AParams: TStrings; AWebResponse:
TWebResponse): Boolean;
```

C++

```
__fastcall Boolean RedirectToPage(const AnsiString APageName, TStrings AParams, TWebResponse
AWebResponse);
```

Description

This is RedirectToPage, a member of class IPageResult.

135.2.52 ISessionsService Interface

ISessionsService is the interface for controlling a TSessionsService object.

Class Hierarchy

SiteComp.ISessionsService

Pascal

```
ISessionsService = interface;
```

C++

```
__interface ISessionsService;
```

File

SiteComp

Description

The TSessionsService component implements ISessionsService. Most applications do not need to use this interface directly. Instead, the TWebAppComponents component uses this interface to drive the orderly management of sessions. In addition, Web session objects (TWebSession) use this interface to update the sessions service when they are terminated.

Note: In C++, method declarations that use ISessionsService use the _di_ISessionsService type instead. This type is a DelphiInterface wrapper around the ISessionsService interface:

Note: typedef System::DelphiInterface (see page 22167)< ISessionsService > _di_ISessionsService;

See Also

TSessionsService (see page 25582)

TWebAppComponents (see page 24958)

TWebSession (see page 24968)

135.2.52.1 ISessionsService Members

The following tables list the members exposed by ISessionsService.

Public Methods

	Name	Description
	CheckStatusChange (see page 19734)	Checks whether any sessions have started or terminated.
	Clear (see page 19734)	Deletes all sessions from memory.
	NewSession (see page 19735)	Starts a new session.
	NotifyStatusChange (see page 19735)	Generates an OnEndSession or OnStartSession event, as approrpriate.

135.2.52.2 ISessionsService Methods

The methods of the ISessionsService class are listed here.

Public Methods

	Name	Description
	CheckStatusChange (see page 19734)	Checks whether any sessions have started or terminated.
	Clear (see page 19734)	Deletes all sessions from memory.
	NewSession (see page 19735)	Starts a new session.
	NotifyStatusChange (see page 19735)	Generates an OnEndSession or OnStartSession event, as approrpriate.

135.2.52.2.1 ISessionsService.CheckStatusChange Method

Checks whether any sessions have started or terminated.

Pascal

```
function CheckStatusChange: Boolean;
```

C++

```
__fastcall Boolean CheckStatusChange();
```

Description

CheckStatusChange examines the sessions currently in memory to see if any sessions have been started or terminated since the last call to NotifyStatusChange (see page 19735). If a session has started or terminated since the last call, CheckStatusChange returns true. Otherwise, it returns false.

See Also

NotifyStatusChange (see page 19735)

135.2.52.2.2 ISessionsService.Clear Method

Deletes all sessions from memory.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the sessions service of all the sessions it has stored in memory. Clear does not terminate active sessions in an orderly fashion (for example, it omits the call to NotifyStatusChange (see page 19735)), it merely frees all memory used to store sessions.

See Also

NotifyStatusChange (see page 19735)

135.2.52.2.3 ISessionsService.NewSession Method

Starts a new session.

Pascal

```
function NewSession: IWebSession;
```

C++

```
__fastcall IWebSession NewSession();
```

Description

Call NewSession to generate a new session. NewSession starts a new session and returns an interface that can be used to terminate that session.

See Also

NotifyStatusChange (see page 19735)

135.2.52.2.4 ISessionsService.NotifyStatusChange Method

Generates an OnEndSession or OnStartSession event, as approrpriate.

Pascal

```
procedure NotifyStatusChange;
```

C++

```
__fastcall NotifyStatusChange();
```

Description

Call NotifyStatusChange to generate the OnStartSession event for any new sessions that have been created since the last call and to generate the OnEndSession event for any sessions that terminated since the last call. For any terminated sessions, NotifyStatusChange frees the memory allocated to the session after the OnEndSession event handler exits.

See Also

OnEndSession (see page 25581)

OnStartSession (see page 25582)

CheckStatusChange (see page 19734)

135.2.53 ISessionsServiceConnectID Interface

Class Hierarchy

SiteComp.ISessionsServiceConnectID

Pascal

```
ISessionsServiceConnectID = interface;
```

C++

```
__interface ISessionsServiceConnectID;
```

File

SiteComp

Description

This is class SiteComp.ISessionsServiceConnectID.

135.2.53.1 ISessionsServiceConnectID Members

The following tables list the members exposed by ISessionsServiceConnectID.

Public Methods

	Name	Description
	Connect (see page 19736)	This is Connect, a member of class ISessionsServiceConnectID.
	ConnectActive (see page 19737)	This is ConnectActive, a member of class ISessionsServiceConnectID.

135.2.53.2 ISessionsServiceConnectID Methods

The methods of the ISessionsServiceConnectID class are listed here.

Public Methods

	Name	Description
	Connect (see page 19736)	This is Connect, a member of class ISessionsServiceConnectID.
	ConnectActive (see page 19737)	This is ConnectActive, a member of class ISessionsServiceConnectID.

135.2.53.2.1 ISessionsServiceConnectID.Connect Method

Pascal

```
function Connect(AID: TSessionID): IWebSession;
```

C++

```
__fastcall IWebSession Connect(TSessionID AID);
```

Description

This is Connect, a member of class ISessionsServiceConnectID.

135.2.53.2.2 ISessionsServiceConnectID.ConnectActive Method

Pascal

```
function ConnectActive(AID: TSessionID): IWebSession;
```

C++

```
__fastcall IWebSession ConnectActive(TSessionID AID);
```

Description

This is ConnectActive, a member of class ISessionsServiceConnectID.

135.2.54 ISessionsServicePersist Interface

Class Hierarchy

SiteComp.ISessionsServicePersist

Pascal

```
ISessionsServicePersist = interface;
```

C++

```
__interface ISessionsServicePersist;
```

File

SiteComp

Description

This is class SiteComp.ISessionsServicePersist.

135.2.54.1 ISessionsServicePersist Members

The following tables list the members exposed by ISessionsServicePersist.

Public Methods

	Name	Description
	LoadFromFile (see page 19738)	This is LoadFromFile, a member of class ISessionsServicePersist.
	LoadFromStream (see page 19738)	This is LoadFromStream, a member of class ISessionsServicePersist.
	SaveToFile (see page 19738)	This is SaveToFile, a member of class ISessionsServicePersist.
	SaveToStream (see page 19738)	This is SaveToStream, a member of class ISessionsServicePersist.

135.2.54.2 ISessionsServicePersist Methods

The methods of the ISessionsServicePersist class are listed here.

Public Methods

	Name	Description
	LoadFromFile (see page 19738)	This is LoadFromFile, a member of class ISessionsServicePersist.
	LoadFromStream (see page 19738)	This is LoadFromStream, a member of class ISessionsServicePersist.
	SaveToFile (see page 19738)	This is SaveToFile, a member of class ISessionsServicePersist.

	SaveToStream (see page 19738)	This is SaveToStream, a member of class ISessionsServicePersist.

135.2.54.2.1 ISessionsServicePersist.LoadFromFile Method

Pascal

```
procedure LoadFromFile(const Filename: string);
```

C++

```
__fastcall LoadFromFile(const AnsiString Filename);
```

Description

This is LoadFromFile, a member of class ISessionsServicePersist.

135.2.54.2.2 ISessionsServicePersist.LoadFromStream Method

Pascal

```
procedure LoadFromStream(S: TStream);
```

C++

```
__fastcall LoadFromStream(TStream S);
```

Description

This is LoadFromStream, a member of class ISessionsServicePersist.

135.2.54.2.3 ISessionsServicePersist.SaveToFile Method

Pascal

```
procedure SaveToFile(const Filename: string);
```

C++

```
__fastcall SaveToFile(const AnsiString Filename);
```

Description

This is SaveToFile, a member of class ISessionsServicePersist.

135.2.54.2.4 ISessionsServicePersist.SaveToStream Method

Pascal

```
procedure SaveToStream(S: TStream);
```

C++

```
__fastcall SaveToStream(TStream S);
```

Description

This is SaveToStream, a member of class ISessionsServicePersist.

135.2.55 ISessionsServiceStoreID Interface

Class Hierarchy

SiteComp.ISessionsServiceStoreID

Pascal

```
ISessionsServiceStoreID = interface;
```

C++

```
__interface ISessionsServiceStoreID;
```

File

SiteComp

Description

This is class SiteComp.ISessionsServiceStoreID.

135.2.55.1 ISessionsServiceStoreID Members

The following tables list the members exposed by ISessionsServiceStoreID.

Public Methods

	Name	Description
	ReadSessionID (see page 19739)	This is ReadSessionID, a member of class ISessionsServiceStoreID.
	WriteSessionID (see page 19740)	This is WriteSessionID, a member of class ISessionsServiceStoreID.

135.2.55.2 ISessionsServiceStoreID Methods

The methods of the ISessionsServiceStoreID class are listed here.

Public Methods

	Name	Description
	ReadSessionID (see page 19739)	This is ReadSessionID, a member of class ISessionsServiceStoreID.
	WriteSessionID (see page 19740)	This is WriteSessionID, a member of class ISessionsServiceStoreID.

135.2.55.2.1 ISessionsServiceStoreID.ReadSessionID Method

Pascal

```
function ReadSessionID(ARequest: TWebRequest; AResponse: TWebResponse; out AID: TSessionID):
Boolean;
```

C++

```
__fastcall Boolean ReadSessionID(TWebRequest ARequest, TWebResponse AResponse, TSessionID AID);
```

Description

This is ReadSessionID, a member of class ISessionsServiceStoreID.

135.2.55.2.2 ISessionsServiceStoreID.WriteSessionID Method

Pascal

```
procedure WriteSessionID(AResponse: TWebResponse; AID: TSessionID);
```

C++

```
__fastcall WriteSessionID(TWebResponse AResponse, TSessionID AID);
```

Description

This is WriteSessionID, a member of class ISessionsServiceStoreID.

135.2.56 IValuesListAdapter Interface

IValuesListAdapter is the interface for supplying name/value pairs.

Class Hierarchy

SiteComp.IValuesListAdapter

Pascal

```
IValuesListAdapter = interface;
```

C++

```
__interface IValuesListAdapter;
```

File

SiteComp

Description

Values list adapters such as TStringsValuesList and TDataSetValuesList implement the IValuesListAdapter interface. This allows adapter fields to use the values list adapter for supplying an associated list of name/value pairs. Typically, this list is used to enumerate the possible values for the adapter field so that they can be added to a selection control such as a list box in an HTML form.

The IValuesListAdapter interface introduces methods to read the current name or value, and to look up the name (or image) associated with a specified value. If the adapter that implements IValuesListAdapter does not also support iteration, server-side script can't change the current name/value pair. However, it can still look up the name or image for a specified value.

Note: In C++, method declarations that use IValuesListAdapter use the _di_IValuesListAdapter type instead. This type is a DelphiInterface wrapper around the IValuesListAdapter interface:

Note: typedef System::DelphiInterface (see page 22167)< IValuesListAdapter > _di_IValuesListAdapter;

See Also

TAdapterUpdateField (see page 24007)

TCustomDataSetAdapterField (see page 7823)

TStringsValuesList (see page 24540)

TDataSetValuesList (see page 8079)

IIteratorSupport (see page 19722)

135.2.56.1 IValuesListAdapter Members

The following tables list the members exposed by IValuesListAdapter.

Public Methods

	Name	Description
	GetListImage (see page 19741)	Returns the image object associated with the current name/value pair.
	GetListImageOfValue (see page 19742)	Returns the image object associated with a specified value.
	GetListName (see page 19742)	Returns the name part of the current name/value pair.
	GetListNameOfValue (see page 19742)	Returns the name associated with a specified value.
	GetListValue (see page 19743)	Returns the value of the current name/value pair.

135.2.56.2 IValuesListAdapter Methods

The methods of the IValuesListAdapter class are listed here.

Public Methods

	Name	Description
	GetListImage (see page 19741)	Returns the image object associated with the current name/value pair.
	GetListImageOfValue (see page 19742)	Returns the image object associated with a specified value.
	GetListName (see page 19742)	Returns the name part of the current name/value pair.
	GetListNameOfValue (see page 19742)	Returns the name associated with a specified value.
	GetListValue (see page 19743)	Returns the value of the current name/value pair.

135.2.56.2.1 IValuesListAdapter.GetListImage Method

Returns the image object associated with the current name/value pair.

Pascal

```
function GetListImage: TComponent;
```

C++

```
__fastcall TComponent * GetListImage();
```

Description

GetListImage returns the object that represents an image associated with the current name/value pair. The image object is the component that responds to request messages which originate from an anchor tag in an HTML form that refers to the image.

If the current name/value pair does not have an associated image, GetListImage returns nil (Delphi) or NULL (C++).

Note: GetListImage provides the support for the Image property of the values list adapter, which can be addressed by server-side script.

See Also

GetListImageOfValue (see page 19742)

GetListName (see page 19742)

GetListValue (see page 19743)

TWebRequestFile (see page 18696)

135.2.56.2.2 IValuesListAdapter.GetListImageOfValue Method

Returns the image object associated with a specified value.

Pascal

```
function GetListImageOfValue(Value: Variant): TComponent;
```

C++

```
__fastcall TComponent * GetListImageOfValue(Variant Value);
```

Description

GetListImageOfValue looks for a name/value pair with the specified value and returns the object that represents an image associated with that value. The image object is the component that responds to request messages which originate from an anchor tag in an HTML form that refers to the image.

Value is the value part of the name/value pair whose image you want.

If Value does not identify a name/value pair with an associated image, GetListImageOfValue returns nil (Delphi) or NULL (C++).

See Also

GetListImage (see page 19741)

GetListNameOfValue (see page 19742)

GetListValue (see page 19743)

TWebRequestFile (see page 18696)

135.2.56.2.3 IValuesListAdapter.GetListName Method

Returns the name part of the current name/value pair.

Pascal

```
function GetListName: string;
```

C++

```
__fastcall AnsiString GetListName();
```

Description

GetListName returns the name part of the current name/value pair. This method provides the support for the Name property of the values list adapter, which can be addressed by server-side script.

If the current name/value pair does not have a name, GetListName returns an empty string.

See Also

GetListNameOfValue (see page 19742)

GetListImage (see page 19741)

GetListValue (see page 19743)

135.2.56.2.4 IValuesListAdapter.GetListNameOfValue Method

Returns the name associated with a specified value.

Pascal

```
function GetListNameOfValue(Value: Variant): string;
```

C++

```
__fastcall AnsiString GetListNameOfValue(Variant Value);
```

Description

GetListNameOfValue looks for a name/value pair with the specified value and returns the name part.

Value is the value part of the name/value pair whose name you want.

If Value does not identify a name/value pair with a name, GetListNameOfValue returns an empty string.

See Also

GetListName (see page 19742)

GetListImageOfValue (see page 19742)

GetListValue (see page 19743)

135.2.56.2.5 IValuesListAdapter.GetListValue Method

Returns the value of the current name/value pair.

Pascal

```
function GetListValue: Variant;
```

C++

```
__fastcall Variant GetListValue();
```

Description

GetListValue returns the value part of the current name/value pair. This method provides the support for the Value property of the values list adapter, which can be addressed by server-side script.

If the current name/value pair does not have a value, GetListValue returns either the name part, or an unassigned Variant, depending on the type of values list adapter.

See Also

GetListName (see page 19742)

GetListImage (see page 19741)

135.2.57 IWebActionsList Interface

Class Hierarchy

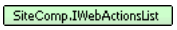

Pascal

```
IWebActionsList = interface;
```

C++

```
__interface IWebActionsList;
```

File

SiteComp

Description

This is class SiteComp.IWebActionsList.

135.2.57.1 IWebActionsList Members

The following tables list the members exposed by IWebActionsList.

Public Methods

	Name	Description
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.

Public Properties

	Name	Description
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.

135.2.57.2 IWebActionsList Methods

The methods of the IWebActionsList class are listed here.

Public Methods

	Name	Description
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.

135.2.57.2.1 IWebActionsList.FindAction Method

Pascal

```
function FindAction(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindAction(const AnsiString AName);
```

Description

This is FindAction, a member of class IWebActionsList.

135.2.57.2.2 IWebActionsList.GetActionsList Method

Pascal

```
procedure GetActionsList(AList: TStrings);
```

C++

```
__fastcall GetActionsList(TStrings AList);
```

Description

This is GetActionsList, a member of class IWebActionsList.

135.2.57.2.3 IWebActionsList.GetVisibleActions Method

Pascal

```
function GetVisibleActions: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleActions();
```

Description

This is GetVisibleActions, a member of class IWebActionsList.

135.2.57.2.4 IWebActionsList.IsActionInUse Method

Pascal

```
function IsActionInUse(const AName: string): Boolean;
```

C++

```
__fastcall Boolean IsActionInUse(const AnsiString AName);
```

Description

This is IsActionInUse, a member of class IWebActionsList.

135.2.57.3 IWebActionsList Properties

The properties of the IWebActionsList class are listed here.

Public Properties

	Name	Description
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.

135.2.57.3.1 IWebActionsList.VisibleActions Property

Pascal

```
property VisibleActions: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleActions;
```

Description

This is VisibleActions, a member of class IWebActionsList.

135.2.58 IWebApplicationInfo Interface

IWebApplicationInfo is the interface for the application adapter.

Class Hierarchy

SiteComp.IWebApplicationInfo

Pascal

```
IWebApplicationInfo = interface;
```

C++

```
__interface IWebApplicationInfo;
```

File

SiteComp

Description

The TApplicationAdapter component implements IWebApplicationInfo. This interface, which the TWebAppComponents object publishes, allows the WebSnap application to obtain application. IWebApplicationInfo declares an ApplicationTitle (see page 19748) property, which allows server-side scripts to use the title of the application. It also provides access to the application adapter object, which may provide additional application-level information.

Note: In C++, method declarations that use IWebApplicationInfo use the _di_IWebApplicationInfo type instead. This type is a DelphiInterface wrapper around the IWebApplicationInfo interface:

Note: typedef System::DelphiInterface (see page 22167)< IWebApplicationInfo > _di_IWebApplicationInfo;

See Also

TApplicationAdapter (see page 24032)

TCustomWebAppComponents (see page 24906)

TWebAPpComponents

135.2.58.1 IWebApplicationInfo Members

The following tables list the members exposed by IWebApplicationInfo.

Public Methods

	Name	Description
	GetApplicationAdapter (see page 19747)	Returns the value of the ApplicationAdapter (see page 19747) property.
	GetApplicationTitle (see page 19747)	Returns the value of the ApplicationTitle (see page 19748) property.

Public Properties

	Name	Description
R	ApplicationAdapter (see page 19747)	Provides access to the adapter that implements the fields and actions of the Application variable in server-side scripts.
R	ApplicationTitle (see page 19748)	Indicates the title of the WebSnap application.

135.2.58.2 IWebApplicationInfo Methods

The methods of the IWebApplicationInfo class are listed here.

Public Methods

	Name	Description
	GetApplicationAdapter (see page 19747)	Returns the value of the ApplicationAdapter (see page 19747) property.
	GetApplicationTitle (see page 19747)	Returns the value of the ApplicationTitle (see page 19748) property.

135.2.58.2.1 IWebApplicationInfo.GetApplicationAdapter Method

Returns the value of the ApplicationAdapter (see page 19747) property.

Pascal

```
function GetApplicationAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetApplicationAdapter();
```

Description

GetApplicationAdapter is the read implementation of the ApplicationAdapter (see page 19747) property.

See Also

ApplicationAdapter (see page 19747)

135.2.58.2.2 IWebApplicationInfo.GetApplicationTitle Method

Returns the value of the ApplicationTitle (see page 19748) property.

Pascal

```
function GetApplicationTitle: string;
```

C++

```
__fastcall AnsiString GetApplicationTitle();
```

Description

GetApplicationTitle is the read implementation of the ApplicationTitle (see page 19748) property.

See Also

ApplicationTitle (see page 19748)

135.2.58.3 IWebApplicationInfo Properties

The properties of the IWebApplicationInfo class are listed here.

Public Properties

	Name	Description
R	ApplicationAdapter (see page 19747)	Provides access to the adapter that implements the fields and actions of the Application variable in server-side scripts.
R	ApplicationTitle (see page 19748)	Indicates the title of the WebSnap application.

135.2.58.3.1 IWebApplicationInfo.ApplicationAdapter Property

Provides access to the adapter that implements the fields and actions of the Application variable in server-side scripts.

Pascal

```
property ApplicationAdapter: TComponent;
```

C++

```
__property TComponent * ApplicationAdapter;
```

Description

Read ApplicationAdapter to access the application adapter component. This is the adapter that implements the fields and actions of the Application variable in server-side scripts. Typically, this is an instance of TApplicationAdapter.

See Also

TApplicationAdapter (see page 24032)

135.2.58.3.2 IWebApplicationInfo.ApplicationTitle Property

Indicates the title of the WebSnap application.

Pascal

```
property ApplicationTitle: string;
```

C++

```
__property AnsiString ApplicationTitle;
```

Description

Read ApplicationTitle to obtain the title of the WebSnap application. This property is the value of the Title field in the Application variable in server-side scripts.

See Also

Title (see page 24624)

135.2.59 IWebContainerSubComponent Interface

Class Hierarchy

SiteComp.IWebContainerSubComponent

Pascal

```
IWebContainerSubComponent = interface;
```

C++

```
__interface IWebContainerSubComponent;
```

File

SiteComp

Description

This is class SiteComp.IWebContainerSubComponent.

135.2.60 IWebDataFields Interface

Class Hierarchy

SiteComp.IWebDataFields

Pascal

```
IWebDataFields = interface;
```

C++

```
__interface IWebDataFields;
```

File

SiteComp

Description

This is class SiteComp.IWebDataFields.

135.2.60.1 IWebDataFields Members

The following tables list the members exposed by IWebDataFields.

Public Methods

	Name	Description
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.

Public Properties

	Name	Description
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

135.2.60.2 IWebDataFields Methods

The methods of the IWebDataFields class are listed here.

Public Methods

	Name	Description
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.

135.2.60.2.1 IWebDataFields.FindField Method

Pascal

```
function FindField(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindField(const AnsiString AName);
```

Description

This is FindField, a member of class IWebDataFields.

135.2.60.2.2 IWebDataFields.GetFieldsList Method

Pascal

```
procedure GetFieldsList(AList: TStrings);
```

C++

```
__fastcall GetFieldsList(TStrings AList);
```

Description

This is GetFieldsList, a member of class IWebDataFields.

135.2.60.2.3 IWebDataFields.GetVisibleFields Method

Pascal

```
function GetVisibleFields: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleFields();
```

Description

This is GetVisibleFields, a member of class IWebDataFields.

135.2.60.2.4 IWebDataFields.IsFieldInUse Method

Pascal

```
function IsFieldInUse(const AName: string): Boolean;
```

C++

```
__fastcall Boolean IsFieldInUse(const AnsiString AName);
```

Description

This is IsFieldInUse, a member of class IWebDataFields.

135.2.60.3 IWebDataFields Properties

The properties of the IWebDataFields class are listed here.

Public Properties

	Name	Description
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

135.2.60.3.1 IWebDataFields.VisibleFields Property

Pascal

```
property VisibleFields: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleFields;
```

Description

This is VisibleFields, a member of class IWebDataFields.

135.2.61 IWebEnabled Interface

Class Hierarchy

SiteComp.IWebEnabled

Pascal

```
IWebEnabled = interface;
```

C++

```
__interface IWebEnabled;
```

File

SiteComp

Description

This is class SiteComp.IWebEnabled.

135.2.61.1 IWebEnabled Members

The following tables list the members exposed by IWebEnabled.

Public Methods

	Name	Description
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

135.2.61.2 IWebEnabled Methods

The methods of the IWebEnabled class are listed here.

Public Methods

	Name	Description
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

135.2.61.2.1 IWebEnabled.WebEnabled Method

Pascal

```
function WebEnabled: Boolean;
```

C++

```
__fastcall Boolean WebEnabled();
```

Description

This is WebEnabled, a member of class IWebEnabled.

135.2.62 IWebFormComponentContainer Interface

Class Hierarchy

SiteComp.IWebFormComponentContainer

Pascal

```
IWebFormComponentContainer = interface;
```

C++

```
__interface IWebFormComponentContainer;
```

File

SiteComp

Description

This is class SiteComp.IWebFormComponentContainer.

135.2.62.1 IWebFormComponentContainer Members

The following tables list the members exposed by IWebFormComponentContainer.

Public Methods

	Name	Description
	GetCount (see page 19753)	This is GetCount, a member of class IWebFormComponentContainer.
	GetItem (see page 19753)	This is GetItem, a member of class IWebFormComponentContainer.

Public Properties

	Name	Description
R	Count (see page 19753)	This is Count, a member of class IWebFormComponentContainer.
R	Items (see page 19754)	This is Items, a member of class IWebFormComponentContainer.

135.2.62.2 IWebFormComponentContainer Methods

The methods of the IWebFormComponentContainer class are listed here.

Public Methods

	Name	Description
	GetCount (see page 19753)	This is GetCount, a member of class IWebFormComponentContainer.
	GetItem (see page 19753)	This is GetItem, a member of class IWebFormComponentContainer.

135.2.62.2.1 IWebFormComponentContainer.GetCount Method

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

This is GetCount, a member of class IWebFormComponentContainer.

135.2.62.2.2 IWebFormComponentContainer.GetItem Method

Pascal

```
function GetItem(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetItem(int I);
```

Description

This is GetItem, a member of class IWebFormComponentContainer.

135.2.62.3 IWebFormComponentContainer Properties

The properties of the IWebFormComponentContainer class are listed here.

Public Properties

	Name	Description
R	Count (see page 19753)	This is Count, a member of class IWebFormComponentContainer.
R	Items (see page 19754)	This is Items, a member of class IWebFormComponentContainer.

135.2.62.3.1 IWebFormComponentContainer.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class IWebFormComponentContainer.

135.2.62.3.2 IWebFormComponentContainer.Items Property

Pascal

```
property Items [I: Integer]: TComponent;
```

C++

```
__property TComponent * Items[int I];
```

Description

This is Items, a member of class IWebFormComponentContainer.

135.2.63 IWebGetActionName Interface

Class Hierarchy

SiteComp.IWebGetActionName

Pascal

```
IWebGetActionName = interface;
```

C++

```
__interface IWebGetActionName;
```

File

SiteComp

Description

This is class SiteComp.IWebGetActionName.

135.2.63.1 IWebGetActionName Members

The following tables list the members exposed by IWebGetActionName.

Public Methods

	Name	Description
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.

Public Properties

	Name	Description
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.

135.2.63.2 IWebGetActionName Methods

The methods of the IWebGetActionName class are listed here.

Public Methods

	Name	Description
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.

135.2.63.2.1 IWebGetActionName.GetActionName Method

Pascal

```
function GetActionName: string;
```

C++

```
__fastcall AnsiString GetActionName();
```

Description

This is GetActionName, a member of class IWebGetActionName.

135.2.63.3 IWebGetActionName Properties

The properties of the IWebGetActionName class are listed here.

Public Properties

	Name	Description
	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.

135.2.63.3.1 IWebGetActionName.ActionName Property

Pascal

```
property ActionName: string;
```

C++

```
__property AnsiString ActionName;
```

Description

This is ActionName, a member of class IWebGetActionName.

135.2.64 IWebGetDisplayLabel Interface

Class Hierarchy

SiteComp.IWebGetDisplayLabel

Pascal

```
IWebGetDisplayLabel = interface;
```

C++

```
__interface IWebGetDisplayLabel;
```

File

SiteComp

Description

This is class SiteComp.IWebGetDisplayLabel.

135.2.64.1 IWebGetDisplayLabel Members

The following tables list the members exposed by IWebGetDisplayLabel.

Public Methods

	Name	Description
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.

135.2.64.2 IWebGetDisplayLabel Methods

The methods of the IWebGetDisplayLabel class are listed here.

Public Methods

	Name	Description
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.

135.2.64.2.1 IWebGetDisplayLabel.GetDisplayLabel Method

Pascal

```
function GetDisplayLabel: string;
```

C++

```
__fastcall AnsiString GetDisplayLabel();
```

Description

This is GetDisplayLabel, a member of class IWebGetDisplayLabel.

135.2.65 IWebGetDisplayWidth Interface

Class Hierarchy

SiteComp.IWebGetDisplayWidth

Pascal

```
IWebGetDisplayWidth = interface;
```

C++

```
__interface IWebGetDisplayWidth;
```

File

SiteComp

Description

This is class SiteComp.IWebGetDisplayWidth.

135.2.65.1 IWebGetDisplayWidth Members

The following tables list the members exposed by IWebGetDisplayWidth.

Public Methods

	Name	Description
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.

135.2.65.2 IWebGetDisplayWidth Methods

The methods of the IWebGetDisplayWidth class are listed here.

Public Methods

	Name	Description
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.

135.2.65.2.1 IWebGetDisplayWidth.GetDisplayWidth Method

Pascal

```
function GetDisplayWidth: Integer;
```

C++

```
__fastcall int GetDisplayWidth();
```

Description

This is GetDisplayWidth, a member of class IWebGetDisplayWidth.

135.2.66 IWebGetFieldName Interface

Class Hierarchy

SiteComp.IWebGetFieldName

Pascal

```
IWebGetFieldName = interface;
```

C++

```
__interface IWebGetFieldName;
```

File

SiteComp

Description

This is class SiteComp.IWebGetFieldName.

135.2.66.1 IWebGetFieldName Members

The following tables list the members exposed by IWebGetFieldName.

Public Methods

	Name	Description
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.

Public Properties

	Name	Description
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.

135.2.66.2 IWebGetFieldName Methods

The methods of the IWebGetFieldName class are listed here.

Public Methods

	Name	Description
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.

135.2.66.2.1 IWebGetFieldName.GetFieldName Method

Pascal

```
function GetFieldName: string;
```

C++

```
__fastcall AnsiString GetFieldName();
```

Description

This is GetFieldName, a member of class IWebGetFieldName.

135.2.66.3 IWebGetFieldName Properties

The properties of the IWebGetFieldName class are listed here.

Public Properties

	Name	Description
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.

135.2.66.3.1 IWebGetFieldName.FieldName Property

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;
```

Description

This is FieldName, a member of class IWebGetFieldName.

135.2.67 IWebGetFieldValue Interface

Class Hierarchy

SiteComp.IWebGetFieldValue

Pascal

```
IWebGetFieldValue = interface;
```

C++

```
__interface IWebGetFieldValue;
```

File

SiteComp

Description

This is class SiteComp.IWebGetFieldValue.

135.2.67.1 IWebGetFieldValue Members

The following tables list the members exposed by IWebGetFieldValue.

Public Methods

	Name	Description
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.

Public Properties

	Name	Description
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.

135.2.67.2 IWebGetFieldValue Methods

The methods of the IWebGetFieldValue class are listed here.

Public Methods

	Name	Description
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.

135.2.67.2.1 IWebGetFieldValue.GetValue Method

Pascal

```
function GetValue: Variant;
```

C++

```
__fastcall Variant GetValue();
```

Description

This is GetValue, a member of class IWebGetFieldValue.

135.2.67.3 IWebGetFieldValue Properties

The properties of the IWebGetFieldValue class are listed here.

Public Properties

	Name	Description
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.

135.2.67.3.1 IWebGetFieldValue.Value Property

Pascal

```
property Value: Variant;
```

C++

```
__property Variant Value;
```

Description

This is Value, a member of class IWebGetFieldValue.

135.2.68 IWebGetFieldValues Interface

IWebGetFieldValues is used to iterate over multiple values associated with an adapter field.

Class Hierarchy

SiteComp.IWebGetFieldValues

Pascal

```
IWebGetFieldValues = interface;
```

C++

```
__interface IWebGetFieldValues;
```

File

SiteComp

Description

IWebGetFieldValues is implemented by adapter fields that may have zero or more associated values. Server-side script accesses these values using the adapter field's Values property.

135.2.68.1 IWebGetFieldValues Members

The following tables list the members exposed by IWebGetFieldValues.

Public Methods

	Name	Description
	EndValueIterator (see page 19761)	Indicate the iteration is finished.
	GetCurrentValue (see page 19761)	Retrieves the current value as a variant.
	HasValue (see page 19761)	Indicates whether the adapter field has a matching value.
	NextValueIteration (see page 19762)	Position the iteration pointer on the next value.
	StartValueIterator (see page 19762)	Reset the iteration pointer to the first item.

135.2.68.2 IWebGetFieldValues Methods

The methods of the IWebGetFieldValues class are listed here.

Public Methods

	Name	Description
	EndValueIterator (see page 19761)	Indicate the iteration is finished.
	GetCurrentValue (see page 19761)	Retrieves the current value as a variant.
	HasValue (see page 19761)	Indicates whether the adapter field has a matching value.
	NextValueIteration (see page 19762)	Position the iteration pointer on the next value.
	StartValueIterator (see page 19762)	Reset the iteration pointer to the first item.

135.2.68.2.1 IWebGetFieldValues.EndValueIterator Method

Indicate the iteration is finished.

Pascal

```
procedure EndValueIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndValueIterator(void * OwnerData);
```

Description

EndValueIterator is used with StartValueIterator (see page 19762) and NextValueIteration (see page 19762) to loop through the values associated with an adapter field. EndValueIterator should be called when iteration is complete. The OwnerData parameter can be used to pass a value between the three iteration functions.

See Also

StartValueIterator (see page 19762)

NextValueIteration (see page 19762)

135.2.68.2.2 IWebGetFieldValues.GetCurrentValue Method

Retrieves the current value as a variant.

Pascal

```
function GetCurrentValue: Variant;
```

C++

```
__fastcall Variant GetCurrentValue();
```

Description

GetCurrentValue retrieves the value associated with internal iteration pointer. The FirstValueIterator and NextValueIteration (see page 19762) are called to position the internal iteration pointer. Server-side script accesses this value using the Values.Value property.

See Also

StartValueIterator (see page 19762)

NextValueIteration (see page 19762)

135.2.68.2.3 IWebGetFieldValues.HasValue Method

Indicates whether the adapter field has a matching value.

Pascal

```
function HasValue(const AValue: Variant): Boolean;
```

C++

```
__fastcall Boolean HasValue(const Variant AValue);
```

Description

Server-side script accesses this value using the Values.HasValue method. Server-side script to generate an HTML select list or check box group can use this method to determine whether an item should be selected or checked.

See Also

ViewMode (see page 25252)

135.2.68.2.4 IWebGetFieldValues.NextValueIteration Method

Position the iteration pointer on the next value.

Pascal

```
function NextValueIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextValueIteration(void * OwnerData);
```

Description

NextValueIteration is used with StartValueIterator (see page 19762) and EndValueIterator (see page 19761) to loop through the values associated with an adapter field. NextValueIteration returns False if there are no more values. The OwnerData parameter can be used to pass a value between the three iteration functions.

See Also

StartValueIterator (see page 19762)

EndValueIterator (see page 19761)

GetCurrentValue (see page 19761)

135.2.68.2.5 IWebGetFieldValues.StartValueIterator Method

Reset the iteration pointer to the first item.

Pascal

```
function StartValueIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartValueIterator(void * OwnerData);
```

Description

StartValueIterator is used with NextValueIteration (see page 19762) and EndValueIterator (see page 19761) to loop through the values associated with an adapter field. StartValueIterator returns false if there are no values. The OwnerData parameter can be used to pass a value between the three iteration functions.

See Also

NextValueIteration (see page 19762)

EndValueIterator (see page 19761)

GetCurrentValue (see page 19761)

135.2.69 IWebGetMaxLength Interface

Class Hierarchy

SiteComp.IWebGetMaxLength

Pascal

```
IWebGetMaxLength = interface;
```

C++

```
__interface IWebGetMaxLength;
```

File

SiteComp

Description

This is class SiteComp.IWebGetMaxLength.

135.2.69.1 IWebGetMaxLength Members

The following tables list the members exposed by IWebGetMaxLength.

Public Methods

	Name	Description
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.

135.2.69.2 IWebGetMaxLength Methods

The methods of the IWebGetMaxLength class are listed here.

Public Methods

	Name	Description
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.

135.2.69.2.1 IWebGetMaxLength.GetMaxLength Method

Pascal

```
function GetMaxLength: integer;
```

C++

```
__fastcall int GetMaxLength();
```

Description

This is GetMaxLength, a member of class IWebGetMaxLength.

135.2.70 IWebGetSessionIDs Interface

Class Hierarchy

SiteComp.IWebGetSessionIDs

Pascal

```
IWebGetSessionIDs = interface;
```

C++

```
__interface IWebGetSessionIDs;
```

File

SiteComp

Description

This is class SiteComp.IWebGetSessionIDs.

135.2.70.1 IWebGetSessionIDs Members

The following tables list the members exposed by IWebGetSessionIDs.

Public Methods

	Name	Description
	GetSessionIDs (see page 19764)	This is GetSessionIDs, a member of class IWebGetSessionIDs.

135.2.70.2 IWebGetSessionIDs Methods

The methods of the IWebGetSessionIDs class are listed here.

Public Methods

	Name	Description
	GetSessionIDs (see page 19764)	This is GetSessionIDs, a member of class IWebGetSessionIDs.

135.2.70.2.1 IWebGetSessionIDs.GetSessionIDs Method

Pascal

```
function GetSessionIDs: IWebSessionIDs;
```

C++

```
__fastcall IWebSessionIDs GetSessionIDs();
```

Description

This is GetSessionIDs, a member of class IWebGetSessionIDs.

135.2.71 IWebImageHREF Interface

IWebImageHREF is the interface for obtaining an HREF that addresses an image.

Class Hierarchy

SiteComp.IWebImageHREF

Pascal

```
IWebImageHREF = interface;
```

C++

```
__interface IWebImageHREF;
```

File

SiteComp

Description

IWebImageHREF defines a single method, WebImageHREF (see page 19766), which supplies a URL for an image that should appear in HTML forms. This interface is used when server-side script addresses the Image.AsHREF property of an adapter.

Adapter field objects that represent images (that is, adapter fields with an Image property) implement IWebImageHREF so that the image reference can be obtained from server-side script. Such adapter fields include TAdapterImageField and TDataSetAdapterImageField.

Note: Method declarations that use IWebImageHREF use the _di_IWebImageHREF type instead. This type is a DelphiInterface wrapper around the IWebImageHREF interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IWebImageHREF >  _di_IWebImageHREF;
```

See Also

TAdapterImageField (see page 23869)

TDataSetAdapterImageField (see page 8138)

135.2.71.1 IWebImageHREF Members

The following tables list the members exposed by IWebImageHREF.

Public Methods

	Name	Description
	WebImageHREF (see page 19766)	Retrieves a URL for an external image file.

135.2.71.2 IWebImageHREF Methods

The methods of the IWebImageHREF class are listed here.

Public Methods

	Name	Description
	WebImageHREF (see page 19766)	Retrieves a URL for an external image file.

135.2.71.2.1 IWebImageHREF.WebImageHREF Method

Retrieves a URL for an external image file.

Pascal

```
function WebImageHREF(var AHREF: string): Boolean;
```

C++

```
__fastcall Boolean WebImageHREF(AnsiString AHREF);
```

Description

WebImageHREF is called when server-side script uses the Image.AsHREF property. WebImageHREF returns a URL that identifies a specific image that is available as a simple file reference. If images must be generated dynamically in response to requests, then WebImageHREF returns false and the adapter dispatcher generates an HREF based on the request parameters supplied by the object that implements this interface.

AHREF returns the URL that identifies the image of this field.

WebImageHREF returns true if it supplies a URL as the value of the AHREF parameter, false if the HTML form must request images using a set of request parameters instead.

See Also

EncodeAdapterItemRequestAsHREF (see page 24872)

135.2.72 IWebInputName Interface

Class Hierarchy

SiteComp.IWebInputName

Pascal

```
IWebInputName = interface;
```

C++

```
__interface IWebInputName;
```

File

SiteComp

Description

This is class SiteComp.IWebInputName.

135.2.72.1 IWebInputName Members

The following tables list the members exposed by IWebInputName.

Public Methods

	Name	Description
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.

Public Properties

	Name	Description
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.

135.2.72.2 IWebInputName Methods

The methods of the IWebInputName class are listed here.

Public Methods

	Name	Description
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.

135.2.72.2.1 IWebInputName.GetInputName Method

Pascal

```
function GetInputName: string;
```

C++

```
__fastcall AnsiString GetInputName();
```

Description

This is GetInputName, a member of class IWebInputName.

135.2.72.3 IWebInputName Properties

The properties of the IWebInputName class are listed here.

Public Properties

	Name	Description
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.

135.2.72.3.1 IWebInputName.InputName Property

Pascal

```
property InputName: string;
```

C++

```
__property AnsiString InputName;
```

Description

This is InputName, a member of class IWebInputName.

135.2.73 IWebIsDefaultAction Interface

Class Hierarchy

SiteComp.IWebIsDefaultAction

Pascal

```
IWebIsDefaultAction = interface;
```

C++

```
__interface IWebIsDefaultAction;
```

File

SiteComp

Description

This is class SiteComp.IWebIsDefaultAction.

135.2.73.1 IWebIsDefaultAction Members

The following tables list the members exposed by IWebIsDefaultAction.

Public Methods

	Name	Description
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.

135.2.73.2 IWebIsDefaultAction Methods

The methods of the IWebIsDefaultAction class are listed here.

Public Methods

	Name	Description
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.

135.2.73.2.1 IWebIsDefaultAction.IsDefaultAction Method

Pascal

```
function IsDefaultAction(ADisplay: TObject): Boolean;
```

C++

```
__fastcall Boolean IsDefaultAction(TObject * ADisplay);
```

Description

This is IsDefaultAction, a member of class IWebIsDefaultAction.

135.2.74 IWebIsDefaultField Interface

Class Hierarchy

SiteComp.IWebIsDefaultField

Pascal

```
IWebIsDefaultField = interface;
```

C++

```
__interface IWebIsDefaultField;
```

File

SiteComp

Description

This is class SiteComp.IWebIsDefaultField.

135.2.74.1 IWebIsDefaultField Members

The following tables list the members exposed by IWebIsDefaultField.

Public Methods

	Name	Description
	IsDefaultField (see page 19769)	This is IsDefaultField, a member of class IWebIsDefaultField.

135.2.74.2 IWebIsDefaultField Methods

The methods of the IWebIsDefaultField class are listed here.

Public Methods

	Name	Description
	IsDefaultField (see page 19769)	This is IsDefaultField, a member of class IWebIsDefaultField.

135.2.74.2.1 IWebIsDefaultField.IsDefaultField Method

Pascal

```
function IsDefaultField(ADisplay: TObject): Boolean;
```

C++

```
__fastcall Boolean IsDefaultField(TObject * ADisplay);
```

Description

This is IsDefaultField, a member of class IWebIsDefaultField.

135.2.75 IWebSession Interface

IWebSession allows an application to terminate a Web session.

Class Hierarchy

SiteComp.IWebSession

Pascal

```
IWebSession = interface;
```

C++

```
__interface IWebSession;
```

File

SiteComp

Description

Use IWebSession to terminate an active Web session. When you add a new session to a TSessionsService component using the ISessionsService (see page 19733) interface, it returns an IWebSession interface for the new session. You can use this interface to terminate the session. Once a session terminates, the application can no longer access its named values.

Note: In C++, method declarations that use IWebSession use the _di_IWebSession type instead. This type is a DelphiInterface wrapper around the IWebSession interface:

Note: typedef System::DelphiInterface (see page 22167)< IWebSession > _di_IWebSession;

See Also

TWebSession (see page 24968)

TSessionsService (see page 25582)

ISessionsService (see page 19733)

135.2.75.1 IWebSession Members

The following tables list the members exposed by IWebSession.

Public Methods

	Name	Description
	Terminate (see page 19771)	Terminates the session.

135.2.75.2 IWebSession Methods

The methods of the IWebSession class are listed here.

Public Methods

	Name	Description
	Terminate (see page 19771)	Terminates the session.

135.2.75.2.1 IWebSession.Terminate Method

Terminates the session.

Pascal

```
procedure Terminate;
```

C++

```
__fastcall Terminate();
```

Description

Call Terminate to terminate a session. Once a session terminates, the application can no longer access its Values.

Sessions are terminated automatically when the application does not receive any requests for the session for TimeOutMinutes minutes.

See Also

TimeoutMinutes

Values

135.2.76 IWebSessionAttributes Interface

Class Hierarchy

SiteComp.IWebSessionAttributes

Pascal

```
IWebSessionAttributes = interface;
```

C++

```
__interface IWebSessionAttributes;
```

File

SiteComp

Description

This is class SiteComp.IWebSessionAttributes.

135.2.76.1 IWebSessionAttributes Members

The following tables list the members exposed by IWebSessionAttributes.

Public Methods

	Name	Description
	GetExpirationTime (see page 19772)	This is GetExpirationTime, a member of class IWebSessionAttributes.
	GetExpired (see page 19772)	This is GetExpired, a member of class IWebSessionAttributes.
	GetIsActive (see page 19773)	This is GetIsActive, a member of class IWebSessionAttributes.
	GetStartTime (see page 19773)	This is GetStartTime, a member of class IWebSessionAttributes.
	GetTerminated (see page 19773)	This is GetTerminated, a member of class IWebSessionAttributes.
	GetTimeoutMinutes (see page 19773)	This is GetTimeoutMinutes, a member of class IWebSessionAttributes.

	GetTouchTime (see page 19773)	This is GetTouchTime, a member of class IWebSessionAttributes.
	SetTimeoutMinutes (see page 19774)	This is SetTimeoutMinutes, a member of class IWebSessionAttributes.

Public Properties

	Name	Description
R	ExpirationTime (see page 19774)	This is ExpirationTime, a member of class IWebSessionAttributes.
R	Expired (see page 19774)	This is Expired, a member of class IWebSessionAttributes.
R	IsActive (see page 19775)	This is IsActive, a member of class IWebSessionAttributes.
R	StartTime (see page 19775)	This is StartTime, a member of class IWebSessionAttributes.
R	Terminated (see page 19775)	This is Terminated, a member of class IWebSessionAttributes.
	TimeoutMinutes (see page 19775)	This is TimeoutMinutes, a member of class IWebSessionAttributes.
R	TouchTime (see page 19775)	This is TouchTime, a member of class IWebSessionAttributes.

135.2.76.2 IWebSessionAttributes Methods

The methods of the IWebSessionAttributes class are listed here.

Public Methods

	Name	Description
	GetExpirationTime (see page 19772)	This is GetExpirationTime, a member of class IWebSessionAttributes.
	GetExpired (see page 19772)	This is GetExpired, a member of class IWebSessionAttributes.
	GetIsActive (see page 19773)	This is GetIsActive, a member of class IWebSessionAttributes.
	GetStartTime (see page 19773)	This is GetStartTime, a member of class IWebSessionAttributes.
	GetTerminated (see page 19773)	This is GetTerminated, a member of class IWebSessionAttributes.
	GetTimeoutMinutes (see page 19773)	This is GetTimeoutMinutes, a member of class IWebSessionAttributes.
	GetTouchTime (see page 19773)	This is GetTouchTime, a member of class IWebSessionAttributes.
	SetTimeoutMinutes (see page 19774)	This is SetTimeoutMinutes, a member of class IWebSessionAttributes.

135.2.76.2.1 IWebSessionAttributes.GetExpirationTime Method

Pascal

```
function GetExpirationTime: TDateTime;
```

C++

```
__fastcall TDateTime GetExpirationTime();
```

Description

This is GetExpirationTime, a member of class IWebSessionAttributes.

135.2.76.2.2 IWebSessionAttributes.GetExpired Method

Pascal

```
function GetExpired: Boolean;
```

C++

```
__fastcall Boolean GetExpired();
```

Description

This is GetExpired, a member of class IWebSessionAttributes.

135.2.76.2.3 IWebSessionAttributes.GetIsActive Method

Pascal

```
function GetIsActive: Boolean;
```

C++

```
__fastcall Boolean GetIsActive();
```

Description

This is GetIsActive, a member of class IWebSessionAttributes.

135.2.76.2.4 IWebSessionAttributes.GetStartTime Method

Pascal

```
function GetStartTime: TDateTime;
```

C++

```
__fastcall TDateTime GetStartTime();
```

Description

This is GetStartTime, a member of class IWebSessionAttributes.

135.2.76.2.5 IWebSessionAttributes.GetTerminated Method

Pascal

```
function GetTerminated: Boolean;
```

C++

```
__fastcall Boolean GetTerminated();
```

Description

This is GetTerminated, a member of class IWebSessionAttributes.

135.2.76.2.6 IWebSessionAttributes.GetTimeoutMinutes Method

Pascal

```
function GetTimeoutMinutes: Integer;
```

C++

```
__fastcall int GetTimeoutMinutes();
```

Description

This is GetTimeoutMinutes, a member of class IWebSessionAttributes.

135.2.76.2.7 IWebSessionAttributes.GetTouchTime Method

Pascal

```
function GetTouchTime: TDateTime;
```

C++

```
__fastcall TDateTime GetTouchTime();
```

Description

This is GetTouchTime, a member of class IWebSessionAttributes.

135.2.76.2.8 IWebSessionAttributes.SetTimeoutMinutes Method

Pascal

```
procedure SetTimeoutMinutes(AValue: Integer);
```

C++

```
__fastcall SetTimeoutMinutes(int AValue);
```

Description

This is SetTimeoutMinutes, a member of class IWebSessionAttributes.

135.2.76.3 IWebSessionAttributes Properties

The properties of the IWebSessionAttributes class are listed here.

Public Properties

	Name	Description
R	ExpirationTime (see page 19774)	This is ExpirationTime, a member of class IWebSessionAttributes.
R	Expired (see page 19774)	This is Expired, a member of class IWebSessionAttributes.
R	IsActive (see page 19775)	This is IsActive, a member of class IWebSessionAttributes.
R	StartTime (see page 19775)	This is StartTime, a member of class IWebSessionAttributes.
R	Terminated (see page 19775)	This is Terminated, a member of class IWebSessionAttributes.
	TimeoutMinutes (see page 19775)	This is TimeoutMinutes, a member of class IWebSessionAttributes.
R	TouchTime (see page 19775)	This is TouchTime, a member of class IWebSessionAttributes.

135.2.76.3.1 IWebSessionAttributes.ExpirationTime Property

Pascal

```
property ExpirationTime: TDateTime;
```

C++

```
__property TDateTime ExpirationTime;
```

Description

This is ExpirationTime, a member of class IWebSessionAttributes.

135.2.76.3.2 IWebSessionAttributes.Expired Property

Pascal

```
property Expired: Boolean;
```

C++

```
__property Boolean Expired;
```

Description

This is Expired, a member of class IWebSessionAttributes.

135.2.76.3.3 IWebSessionAttributes.IsActive Property

Pascal

```
property IsActive: Boolean;
```

C++

```
__property Boolean IsActive;
```

Description

This is IsActive, a member of class IWebSessionAttributes.

135.2.76.3.4 IWebSessionAttributes.StartTime Property

Pascal

```
property StartTime: TDateTime;
```

C++

```
__property TDateTime StartTime;
```

Description

This is StartTime, a member of class IWebSessionAttributes.

135.2.76.3.5 IWebSessionAttributes.Terminated Property

Pascal

```
property Terminated: Boolean;
```

C++

```
__property Boolean Terminated;
```

Description

This is Terminated, a member of class IWebSessionAttributes.

135.2.76.3.6 IWebSessionAttributes.TimeoutMinutes Property

Pascal

```
property TimeoutMinutes: Integer;
```

C++

```
__property int TimeoutMinutes;
```

Description

This is TimeoutMinutes, a member of class IWebSessionAttributes.

135.2.76.3.7 IWebSessionAttributes.TouchTime Property

Pascal

```
property TouchTime: TDateTime;
```

C++

```
__property TDateTime TouchTime;
```

Description

This is TouchTime, a member of class IWebSessionAttributes.

135.2.77 IWebSessionID Interface

Class Hierarchy

SiteComp.IWebSessionID

Pascal

```
IWebSessionID = interface;
```

C++

```
__interface IWebSessionID;
```

File

SiteComp

Description

This is class SiteComp.IWebSessionID.

135.2.77.1 IWebSessionID Members

The following tables list the members exposed by IWebSessionID.

Public Methods

	Name	Description
	GetID (see page 19776)	This is GetID, a member of class IWebSessionID.

Public Properties

	Name	Description
R	ID (see page 19777)	This is ID, a member of class IWebSessionID.

135.2.77.2 IWebSessionID Methods

The methods of the IWebSessionID class are listed here.

Public Methods

	Name	Description
	GetID (see page 19776)	This is GetID, a member of class IWebSessionID.

135.2.77.2.1 IWebSessionID.GetID Method

Pascal

```
function GetID: TSessionID;
```

C++

```
__fastcall TSessionID GetID();
```

Description

This is GetID, a member of class IWebSessionID.

135.2.77.3 IWebSessionID Properties

The properties of the IWebSessionID class are listed here.

Public Properties

	Name	Description
R	ID (see page 19777)	This is ID, a member of class IWebSessionID.

135.2.77.3.1 IWebSessionID.ID Property

Pascal

```
property ID: TSessionID;
```

C++

```
__property TSessionID ID;
```

Description

This is ID, a member of class IWebSessionID.

135.2.78 IWebSessionIDs Interface

IWebSessionIDs lists the session IDs (see page 19779) for all sessions that TSessionsService stores in memory.

Class Hierarchy

SiteComp.IWebSessionIDs

Pascal

```
IWebSessionIDs = interface;
```

C++

```
__interface IWebSessionIDs;
```

File

SiteComp

Description

Use IWebSessionIDs to obtain the session IDs (see page 19779) for all the sessions in a sessions service.

Note: In C++, method declarations that use IWebSessionIDs use the _di_IWebSessionIDs type instead. This type is a DelphiInterface wrapper around the IWebSessionIDs interface:

Note: typedef System::DelphiInterface (see page 22167)< IWebSessionIDs > _di_IWebSessionIDs;

See Also

TSessionsService (see page 25582)

135.2.78.1 IWebSessionIDs Members

The following tables list the members exposed by IWebSessionIDs.

Public Methods

	Name	Description
	GetID (see page 19778)	Returns the value of the IDs (see page 19779) property.
	GetIDCount (see page 19778)	Returns the value of the IDCount (see page 19779) property.

Public Properties

	Name	Description
R	IDCount (see page 19779)	Indicates the number of session IDs (see page 19779) available.
R	IDs (see page 19779)	Lists the session IDs for all the sessions stored by the sessions service.

135.2.78.2 IWebSessionIDs Methods

The methods of the IWebSessionIDs class are listed here.

Public Methods

	Name	Description
	GetID (see page 19778)	Returns the value of the IDs (see page 19779) property.
	GetIDCount (see page 19778)	Returns the value of the IDCount (see page 19779) property.

135.2.78.2.1 IWebSessionIDs.GetID Method

Returns the value of the IDs (see page 19779) property.

Pascal

```
function GetID(I: Integer): TSessionID;
```

C++

```
__fastcall TSessionID GetID(int I);
```

Description

GetID is the read implementation of the IDs (see page 19779) property.

I is the index of the session ID to return.

See Also

IDs (see page 19779)

135.2.78.2.2 IWebSessionIDs.GetIDCount Method

Returns the value of the IDCount (see page 19779) property.

Pascal

```
function GetIDCount: Integer;
```

C++

```
__fastcall int GetIDCount();
```

Description

GetIDCount is the read implementation of the IDCount (see page 19779) property.

See Also

IDCount (see page 19779)

135.2.78.3 IWebSessionIDs Properties

The properties of the IWebSessionIDs class are listed here.

Public Properties

	Name	Description
R	IDCount (see page 19779)	Indicates the number of session IDs (see page 19779) available.
R	IDs (see page 19779)	Lists the session IDs for all the sessions stored by the sessions service.

135.2.78.3.1 IWebSessionIDs.IDCount Property

Indicates the number of session IDs (see page 19779) available.

Pascal

```
property IDCount: Integer;
```

C++

```
__property int IDCount;
```

Description

Read IDCount to obtain an upper bound for the Index parameter of the IDs (see page 19779) property. IDCount is the number of session IDs (see page 19779) (hence, the number of sessions) that the sessions service has stored in memory.

See Also

IDs (see page 19779)

135.2.78.3.2 IWebSessionIDs.IDs Property

Lists the session IDs for all the sessions stored by the sessions service.

Pascal

```
property IDs [I: Integer]: TSessionID;
```

C++

```
__property TSessionID IDs[int I];
```

Description

Use IDs to iterate over the session IDs of all the sessions that the sessions service stores.

I is the index of the session ID. It can be any value from 0 to IDCount (see page 19779) – 1.

See Also

IDCount (see page 19779)

135.2.79 IWebSessionValues Interface

Class Hierarchy

SiteComp.IWebSessionValues

Pascal

```
IWebSessionValues = interface;
```

C++

```
__interface IWebSessionValues;
```

File

SiteComp

Description

This is class SiteComp.IWebSessionValues.

135.2.79.1 IWebSessionValues Members

The following tables list the members exposed by IWebSessionValues.

Public Methods

	Name	Description
	GetItemCount (see page 19780)	This is GetItemCount, a member of class IWebSessionValues.
	GetItems (see page 19781)	This is GetItems, a member of class IWebSessionValues.
	GetValue (see page 19781)	This is GetValue, a member of class IWebSessionValues.
	SetValue (see page 19781)	This is SetValue, a member of class IWebSessionValues.

Public Properties

	Name	Description
	Values (see page 19782)	This is Values, a member of class IWebSessionValues.

135.2.79.2 IWebSessionValues Methods

The methods of the IWebSessionValues class are listed here.

Public Methods

	Name	Description
	GetItemCount (see page 19780)	This is GetItemCount, a member of class IWebSessionValues.
	GetItems (see page 19781)	This is GetItems, a member of class IWebSessionValues.
	GetValue (see page 19781)	This is GetValue, a member of class IWebSessionValues.
	SetValue (see page 19781)	This is SetValue, a member of class IWebSessionValues.

135.2.79.2.1 IWebSessionValues.GetItemCount Method

Pascal

```
function GetItemCount: Integer;
```

C++

```
__fastcall int GetItemCount();
```

Description

This is GetItemCount, a member of class IWebSessionValues.

135.2.79.2.2 IWebSessionValues.GetItems Method

Pascal

```
procedure GetItems(var AItems: TAbstractNamedVariants);
```

C++

```
__fastcall GetItems(TAbstractNamedVariants AItems);
```

Description

This is GetItems, a member of class IWebSessionValues.

135.2.79.2.3 IWebSessionValues.GetValue Method

Pascal

```
function GetValue(const AName: string): Variant;
```

C++

```
__fastcall Variant GetValue(const AnsiString AName);
```

Description

This is GetValue, a member of class IWebSessionValues.

135.2.79.2.4 IWebSessionValues.SetValue Method

Pascal

```
procedure SetValue(const AName: string; const AValue: Variant);
```

C++

```
__fastcall SetValue(const AnsiString AName, const Variant AValue);
```

Description

This is SetValue, a member of class IWebSessionValues.

135.2.79.3 IWebSessionValues Properties

The properties of the IWebSessionValues class are listed here.

Public Properties

	Name	Description
	Values (see page 19782)	This is Values, a member of class IWebSessionValues.

135.2.79.3.1 IWebSessionValues.Values Property

Pascal

```
property Values [const AName: string]: Variant;
```

C++

```
__property Variant Values[AnsiString const AName];
```

Description

This is Values, a member of class IWebSessionValues.

135.2.80 IWebSetActionName Interface

Class Hierarchy

SiteComp.IWebSetActionName

Pascal

```
IWebSetActionName = interface;
```

C++

```
__interface IWebSetActionName;
```

File

SiteComp

Description

This is class SiteComp.IWebSetActionName.

135.2.80.1 IWebSetActionName Members

The following tables list the members exposed by IWebSetActionName.

Public Methods

	Name	Description
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.

135.2.80.2 IWebSetActionName Methods

The methods of the IWebSetActionName class are listed here.

Public Methods

	Name	Description
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.

135.2.80.2.1 IWebSetActionName.SetActionName Method

Pascal

```
procedure SetActionName(const AFieldName: string);
```

C++

```
__fastcall SetActionName(const AnsiString AFieldName);
```

Description

This is SetActionName, a member of class IWebSetActionName.

135.2.81 IWebSetFieldName Interface

Class Hierarchy

SiteComp.IWebSetFieldName

Pascal

```
IWebSetFieldName = interface;
```

C++

```
__interface IWebSetFieldName;
```

File

SiteComp

Description

This is class SiteComp.IWebSetFieldName.

135.2.81.1 IWebSetFieldName Members

The following tables list the members exposed by IWebSetFieldName.

Public Methods

	Name	Description
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.

135.2.81.2 IWebSetFieldName Methods

The methods of the IWebSetFieldName class are listed here.

Public Methods

	Name	Description
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.

135.2.81.2.1 IWebSetFieldName.SetFieldName Method

Pascal

```
procedure SetFieldName(const AFieldName: string);
```

C++

```
__fastcall SetFieldName(const AnsiString AFieldName);
```

Description

This is SetFieldName, a member of class IWebSetFieldName.

135.2.82 IWebUserList Interface

IWebUserList is the interface for validating user logins and access rights.

Class Hierarchy

SiteComp.IWebUserList

Pascal

```
IWebUserList = interface;
```

C++

```
__interface IWebUserList;
```

File

SiteComp

Description

The TWebUserList component implements IWebUserList. This interface, which the TWebAppComponents object publishes, allows the WebSnap application to check whether a user supplied the correct password when logging in and whether the user has rights to perform the operation requested in an HTTP request message.

Note: In C++, method declarations that use IWebUserList use the _di_IWebUserList type instead. This type is a DelphiInterface wrapper around the IWebUserList interface:

Note: typedef System::DelphiInterface (see page 22167)< IWebUserList > _di_IWebUserList;

See Also

TWebUserList (see page 25625)

TCustomWebAppComponents (see page 24906)

TWebAPpComponents

135.2.82.1 IWebUserList Members

The following tables list the members exposed by IWebUserList.

Public Methods

	Name	Description
	CheckAccessRights (see page 19785)	Checks whether a specified user has a given access rights string.
	ValidateUser (see page 19785)	Validates a user login.

135.2.82.2 IWebUserList Methods

The methods of the IWebUserList class are listed here.

Public Methods

	Name	Description
	CheckAccessRights (see page 19785)	Checks whether a specified user has a given access rights string.
	ValidateUser (see page 19785)	Validates a user login.

135.2.82.2.1 IWebUserList.CheckAccessRights Method

Checks whether a specified user has a given access rights string.

Pascal

```
function CheckAccessRights(UserID: Variant; Rights: string): Boolean;
```

C++

```
__fastcall Boolean CheckAccessRights(Variant UserID, AnsiString Rights);
```

Description

Call CheckAccessRights to determine whether the current end user is authorized to view Web pages or adapters whose access rights string is Rights. The end user must be logged in before CheckAccessRights can determine the user's access rights.

UserID is the user ID of the user to check.

Rights is the access rights string that CheckAccessRights must look for. Typically, this is the value of the ViewRights property on a Web page information object that describes the page the user wants to access or on an adapter that the user wants to call.

See Also

HasViewRights (see page 24112)

HasModifyRights (see page 24112)

HasExecuteRights (see page 24111)

ViewAccess (see page 15069)

ViewAccess (see page 24183)

135.2.82.2.2 IWebUserList.ValidateUser Method

Validates a user login.

Pascal

```
function ValidateUser(Strings: TStrings): Variant;
```

C++

```
__fastcall Variant ValidateUser(TStrings Strings);
```

Description

Call ValidateUser to check whether the information a user supplies on login matches a user in the user list.

Strings is a string list that lists the information that the user supplied when logging in. Each entry in the string list is of the form Name=Value. Typically, the names of strings in the list are 'UserName' and 'Password'.

If the login information is valid, ValidateUser returns the UserID of the identified user. Otherwise, ValidateUser returns an unassigned Variant.

See Also

CheckAccessRights (see page 19785)

135.2.83 IWebValueIsEqual Interface

Class Hierarchy

SiteComp.IWebValueIsEqual

Pascal

```
IWebValueIsEqual = interface;
```

C++

```
__interface IWebValueIsEqual;
```

File

SiteComp

Description

This is class SiteComp.IWebValueIsEqual.

135.2.83.1 IWebValueIsEqual Members

The following tables list the members exposed by IWebValueIsEqual.

Public Methods

	Name	Description
	IsEqual (see page 19787)	This is IsEqual, a member of class IWebValueIsEqual.

135.2.83.2 IWebValueIsEqual Methods

The methods of the IWebValueIsEqual class are listed here.

Public Methods

	Name	Description
	IsEqual (see page 19787)	This is IsEqual, a member of class IWebValueIsEqual.

135.2.83.2.1 IWebValueIsEqual.IsEqual Method

Pascal

```
function IsEqual(const Value: Variant): Boolean;
```

C++

```
__fastcall Boolean IsEqual(const Variant Value);
```

Description

This is IsEqual, a member of class IWebValueIsEqual.

135.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FullyQualifiedFieldName (see page 19788)	This is function SiteComp.FullyQualifiedFieldName.
	FullyQualifiedName (see page 19788)	This is function SiteComp.FullyQualifiedName.

135.3.1 SiteComp.FullyQualifiedFieldName Function

Pascal

```
function FullyQualifiedFieldName(AComponent: TComponent): string;
```

C++

```
AnsiString FullyQualifiedFieldName(TComponent * AComponent);
```

File

SiteComp

Description

This is function SiteComp.FullyQualifiedFieldName.

135.3.2 SiteComp.FullyQualifiedName Function

Pascal

```
function FullyQualifiedName(AComponent: TComponent; AOption: TQualifyOption): string;
```

C++

```
AnsiString FullyQualifiedName(TComponent * AComponent, TQualifyOption AOption);
```

File

SiteComp

Description

This is function SiteComp.FullyQualifiedName.

135.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAdapterActionHTMLElementType (see page 19789)	This is record SiteComp.TAdapterActionHTMLElementType.
	TAdapterDisplayCharacteristic (see page 19789)	This is record SiteComp.TAdapterDisplayCharacteristic.
	TAdapterDisplayHTMLElementType (see page 19790)	This is record SiteComp.TAdapterDisplayHTMLElementType.
	TAdapterDisplayViewModeType (see page 19790)	This is record SiteComp.TAdapterDisplayViewModeType.
	TAdapterInputHTMLElementType (see page 19791)	This is record SiteComp.TAdapterInputHTMLElementType.
	TAdapterItemRequestParamOption (see page 19792)	This is record SiteComp.TAdapterItemRequestParamOption.
	TDispatchPageFlag (see page 19792)	This is record SiteComp.TDispatchPageFlag.
	TQualifyOption (see page 19792)	This is record SiteComp.TQualifyOption.

135.4.1 SiteComp.TAdapterActionHTMLElementType Enumeration

Pascal

```
TAdapterActionHTMLElementType = (
  htmlaButton,
  htmlaImage,
  htmlaAnchor,
  htmlaEventImages
);
```

C++

```
enum TAdapterActionHTMLElementType {
  htmlaButton,
  htmlaImage,
  htmlaAnchor,
  htmlaEventImages
};
```

File

SiteComp

Description

This is record SiteComp.TAdapterActionHTMLElementType.

135.4.2 SiteComp.TAdapterDisplayCharacteristic Enumeration

Pascal

```
TAdapterDisplayCharacteristic = (
```

```
    dcMultipleRecordView,
    dcCurrentRecordView,
    dcChangeCurrentRecordView
  );
```

C++

```
  enum TAdapterDisplayCharacteristic {
    dcMultipleRecordView,
    dcCurrentRecordView,
    dcChangeCurrentRecordView
  };
```

File

SiteComp

Description

This is record SiteComp.TAdapterDisplayCharacteristic.

135.4.3 SiteComp.TAdapterDisplayHTMLElementType Enumeration

Pascal

```
  TAdapterDisplayHTMLElementType = (
    htmldText,
    htmldImage,
    htmldList,
    htmldNone
  );
```

C++

```
  enum TAdapterDisplayHTMLElementType {
    htmldText,
    htmldImage,
    htmldList,
    htmldNone
  };
```

File

SiteComp

Description

This is record SiteComp.TAdapterDisplayHTMLElementType.

135.4.4 SiteComp.TAdapterDisplayViewModeType Enumeration

Pascal

```
  TAdapterDisplayViewModeType = (
    htmlvmInput,
    htmlvmDisplay,
    htmlvmToggleOnAccess,
```

```
    htmlvmNone
  );
```

C++

```
  enum TAdapterDisplayViewModeType {
    htmlvmInput,
    htmlvmDisplay,
    htmlvmToggleOnAccess,
    htmlvmNone
  };
```

File

SiteComp

Description

This is record SiteComp.TAdapterDisplayViewModeType.

135.4.5 SiteComp.TAdapterInputHTMLElementType Enumeration

Pascal

```
  TAdapterInputHTMLElementType = (
    htmliNone,
    htmliTextInput,
    htmliPasswordInput,
    htmliSelect,
    htmliSelectMultiple,
    htmliRadio,
    htmliCheckBox,
    htmliTextArea,
    htmliFile
  );
```

C++

```
  enum TAdapterInputHTMLElementType {
    htmliNone,
    htmliTextInput,
    htmliPasswordInput,
    htmliSelect,
    htmliSelectMultiple,
    htmliRadio,
    htmliCheckBox,
    htmliTextArea,
    htmliFile
  };
```

File

SiteComp

Description

This is record SiteComp.TAdapterInputHTMLElementType.

135.4.6 SiteComp.TAdapterItemRequestParamOption Enumeration

Pascal

```
TAdapterItemRequestParamOption = (
  roExcludeID
);
```

C++

```
enum TAdapterItemRequestParamOption {
  roExcludeID
};
```

File

SiteComp

Description

This is record SiteComp.TAdapterItemRequestParamOption.

135.4.7 SiteComp.TDispatchPageFlag Enumeration

Pascal

```
TDispatchPageFlag = (
  dpPublished
);
```

C++

```
enum TDispatchPageFlag {
  dpPublished
};
```

File

SiteComp

Description

This is record SiteComp.TDispatchPageFlag.

135.4.8 SiteComp.TQualifyOption Enumeration

Pascal

```
TQualifyOption = (
  qaUseModulesVar,
  qaNoModulesVar,
  qaNoModuleName
);
```

C++

```
enum TQualifyOption {
  qaUseModulesVar,
  qaNoModulesVar,
  qaNoModuleName
};
```

File

SiteComp

Description

This is record SiteComp.TQualifyOption.

135.5 Types

The following table lists types in this documentation.

Types

Name	Description
TAdapterDisplayCharacteristics (see page 19794)	This is type SiteComp.TAdapterDisplayCharacteristics.
TAdapterItemRequestParamOptions (see page 19794)	This is type SiteComp.TAdapterItemRequestParamOptions.
TDispatchPageFlags (see page 19795)	This is type SiteComp.TDispatchPageFlags.
TSessionID (see page 19795)	This is type SiteComp.TSessionID.

135.5.1 SiteComp.TAdapterDisplayCharacteristics Type

Pascal

```
TAdapterDisplayCharacteristics = set of TAdapterDisplayCharacteristic;
```

C++

```
TAdapterDisplayCharacteristic TAdapterDisplayCharacteristics;
```

File

SiteComp

Description

This is type SiteComp.TAdapterDisplayCharacteristics.

135.5.2 SiteComp.TAdapterItemRequestParamOptions Type

Pascal

```
TAdapterItemRequestParamOptions = set of TAdapterItemRequestParamOption;
```

C++

```
TAdapterItemRequestParamOption TAdapterItemRequestParamOptions;
```

File

SiteComp

Description

This is type SiteComp.TAdapterItemRequestParamOptions.

135.5.3 SiteComp.TDispatchPageFlags Type

Pascal

```
TDispatchPageFlags = set of TDispatchPageFlag;
```

C++

```
TDispatchPageFlag TDispatchPageFlags;
```

File

SiteComp

Description

This is type SiteComp.TDispatchPageFlags.

135.5.4 SiteComp.TSessionID Type

Pascal

```
TSessionID = type string;
```

C++

```
type string TSessionID;
```

File

SiteComp

Description

This is type SiteComp.TSessionID.

135.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
ExecuteScript (see page 19796)	This is variable SiteComp.ExecuteScript.
GetHTMLSampleImage (see page 19796)	This is variable SiteComp.GetHTMLSampleImage.
IDEFileManager (see page 19797)	This is variable SiteComp.IDEFileManager.
InetFileTypes (see page 19797)	This is variable SiteComp.InetFileTypes.
InternetEnvOptions (see page 19797)	This is variable SiteComp.InternetEnvOptions.
ScriptDebugger (see page 19798)	This is variable SiteComp.ScriptDebugger.

135.6.1 SiteComp.ExecuteScript Variable

Pascal

```
ExecuteScript: IInternetExecuteScript = nil;
```

C++

```
IInternetExecuteScript ExecuteScript = nil;
```

File

SiteComp

Description

This is variable SiteComp.ExecuteScript.

135.6.2 SiteComp.GetHTMLSampleImage Variable

Pascal

```
GetHTMLSampleImage: function: string = nil;
```

C++

```
function: string GetHTMLSampleImage = nil;
```

File

SiteComp

Description

This is variable SiteComp.GetHTMLSampleImage.

135.6.3 SiteComp.IDEFileManager Variable

Pascal

```
IDEFileManager: IIDEFileManager = nil;
```

C++

```
IIDEFileManager IDEFileManager = nil;
```

File

SiteComp

Description

This is variable SiteComp.IDEFileManager.

135.6.4 SiteComp.InetFileTypes Variable

Pascal

```
InetFileTypes: IInternetFileTypes = nil;
```

C++

```
IInternetFileTypes InetFileTypes = nil;
```

File

SiteComp

Description

This is variable SiteComp.InetFileTypes.

135.6.5 SiteComp.InternetEnvOptions Variable

Pascal

```
InternetEnvOptions: IInternetEnvOptions = nil;
```

C++

```
IInternetEnvOptions InternetEnvOptions = nil;
```

File

SiteComp

Description

This is variable SiteComp.InternetEnvOptions.

135.6.6 SiteComp.ScriptDebugger Variable

Pascal

```
ScriptDebugger: IInternetScriptDebugger = nil;
```

C++

```
IInternetScriptDebugger ScriptDebugger = nil;
```

File

SiteComp

Description

This is variable SiteComp.ScriptDebugger.

135.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
AdapterActionHTMLElementTypeNames (see page 19799)	This is constant SiteComp.AdapterActionHTMLElementTypeNames.
AdapterDisplayHTMLElementTypeNames (see page 19799)	This is constant SiteComp.AdapterDisplayHTMLElementTypeNames.
AdapterDisplayViewModeTypeNames (see page 19800)	This is constant SiteComp.AdapterDisplayViewModeTypeNames.
AdapterInputHTMLElementTypeNames (see page 19800)	This is constant SiteComp.AdapterInputHTMLElementTypeNames.

135.7.1 SiteComp.AdapterActionHTMLElementTypeNames Constant

Pascal

```
AdapterActionHTMLElementTypeNames: array[TAdapterActionHTMLElementType] of string = ('Button',
'Image', 'Anchor', 'EventImages');
```

C++

```
array[TAdapterActionHTMLElementType] of string AdapterActionHTMLElementTypeNames = ('Button',
'Image', 'Anchor', 'EventImages');
```

File

SiteComp

Description

This is constant SiteComp.AdapterActionHTMLElementTypeNames.

135.7.2 SiteComp.AdapterDisplayHTMLElementTypeNames Constant

Pascal

```
AdapterDisplayHTMLElementTypeNames: array[TAdapterDisplayHTMLElementType] of string = ('Text',
'Image', 'List', '');
```

C++

```
array[TAdapterDisplayHTMLElementType] of string AdapterDisplayHTMLElementTypeNames = ('Text',
'Image', 'List', '');
```

File

SiteComp

Description

This is constant SiteComp.AdapterDisplayHTMLElementTypeNames.

135.7.3 SiteComp.AdapterDisplayViewModeTypeNames Constant

Pascal

```
AdapterDisplayViewModeTypeNames: array[TAdapterDisplayViewModeType] of string = ('Input',
'Display', 'ToggleOnAccess', '');
```

C++

```
array[TAdapterDisplayViewModeType] of string AdapterDisplayViewModeTypeNames = ('Input',
'Display', 'ToggleOnAccess', '');
```

File

SiteComp

Description

This is constant SiteComp.AdapterDisplayViewModeTypeNames.

135.7.4 SiteComp.AdapterInputHTMLElementTypeNames Constant

Pascal

```
AdapterInputHTMLElementTypeNames: array[TAdapterInputHTMLElementType] of string = ('',
'TextInput', 'PasswordInput', 'Select', 'SelectMultiple', 'Radio', 'CheckBox', 'TextArea',
'File');
```

C++

```
array[TAdapterInputHTMLElementType] of string AdapterInputHTMLElementTypeNames = ('',
'TextInput', 'PasswordInput', 'Select', 'SelectMultiple', 'Radio', 'CheckBox', 'TextArea',
'File');
```

File

SiteComp

Description

This is constant SiteComp.AdapterInputHTMLElementTypeNames.

136 SiteProd Namespace

This is namespace SiteProd.

Topics

Name	Description
Classes (see page 19802)	The following table lists classes in this documentation.

Classes

	Name	Description
	TAdapterPageProducer (see page 19802)	TAdapterPageProducer generates the proper HTML and server-side script needed to display adapter fields and execute adapter actions.
	TBaseAdapterPageProducer (see page 19813)	TBaseAdapterPageProducer generates the proper HTML and JavaScript needed to display adapter fields and execute adapter actions.
	TCustomAdapterPageProducer (see page 19832)	tcustomadapterpageproducer generates the proper HTML and JavaScript needed to display adapter fields and execute adapter actions.
	TSitePageElements (see page 19844)	TSitePageElements stores part of the content that page producers generate.

136.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TAdapterPageProducer (see page 19802)	TAdapterPageProducer generates the proper HTML and server-side script needed to display adapter fields and execute adapter actions.
	TBaseAdapterPageProducer (see page 19813)	TBaseAdapterPageProducer generates the proper HTML and JavaScript needed to display adapter fields and execute adapter actions.
	TCustomAdapterPageProducer (see page 19832)	tcustomadapterpageproducer generates the proper HTML and JavaScript needed to display adapter fields and execute adapter actions.
	TSitePageElements (see page 19844)	TSitePageElements stores part of the content that page producers generate.

136.1.1 TAdapterPageProducer Class

TAdapterPageProducer generates the proper HTML and server-side script needed to display adapter fields and execute adapter actions.

Class Hierarchy

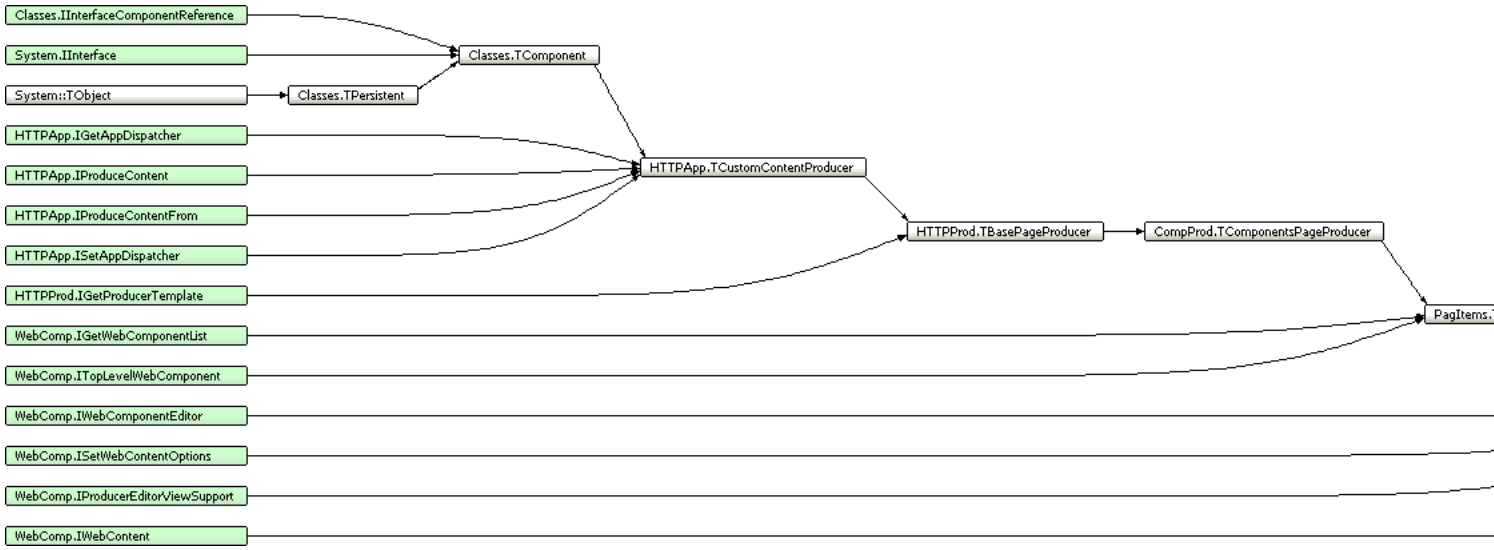

Pascal

```
TAdapterPageProducer = class(TCustomAdapterPageProducer);
```

C++

```
class TAdapterPageProducer : public TCustomAdapterPageProducer;
```

File

SiteProd

Description

TAdapterPageProducer creates an HTML page using content it obtains from a hierarchy of components called Web items. Add components to this hierarchy to define HTML forms and tables. For example, to create an HTML form with a grid, add a TAdapterForm and TAdapterGrid components. The Web items that appear in the TAdapterPageProducer's hierarchy use Adapter components to fetch the information that they display.

Use the Web Page Module wizard to create a web page that uses the TAdapterPageProducer. You can select TAdapterPageProducer from the drop-down list at the top of the Web Page Module wizard, or add a TAdapterPageProducer component to an existing web page modue.

Web page modules that use the TAdapterPageProducer provide an editor view called HTML Script. Use this view to see the HTML Script that TAdapterPageProducer generates. The script generated by TAdapterPageProducer is not editable. However, you can copy the generated script from the HTML Script view to a new Web Page Module. You can then edit the copy of the script in the code editor or with an external HTML editor that supports embedded script tags (for example <% %>).

TAdapterPageProducer uses a template that contains the following special tags:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles (see page 19810) or StylesFile (see page 19811) property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items.

<#SERVERSCRIPT> - This tag is replaced with the generated HTML/JavaScript. The TAdapterPageProducer will not generate any content if this tag is missing.

See Also

TAdapter (see page 23713)

TAdapterForm (see page 25159)

TAdapterGrid (see page 25175)

Page Producers

Server-side Scripting in WebSnap

136.1.1.1 TAdapterPageProducer Members

The following tables list the members exposed by TAdapterPageProducer.

Protected Events

	Name	Description
	OnHTMLTag (see page 15280)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 19820)	Specifies whether a Web item of a specified type can be added to the WebPageItems property list.

	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CreatePageElements (see page 19821)	Creates the object that implements the PageElements (see page 19830) property.
V	DeclareStyles (see page 19821)	Returns an HTML Styles (see page 19830) tag that declares the styles specified by the Styles (see page 19830) or StylesFile (see page 19831) property.
V	DeclareWarnings (see page 19821)	Returns the formatted warnings that appear at design time to indicate the status of content generation.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoAfterGetContent (see page 19822)	Generates an OnAfterGetContent (see page 19832) event.
	DoBeforeGetContent (see page 19822)	Generates an OnBeforeGetContent (see page 19832) event.
V	DoTagEvent (see page 5618)	This is DoTagEvent, a member of class TComponentsPageProducer.
	DoTagEvent (see page 15276)	This is DoTagEvent, a member of class TBasePageProducer.
V	DoTagEvent (see page 19822)	Translates HTML-transparent tags in the template.
V	FindComponent (see page 5619)	This is FindComponent, a member of class TComponentsPageProducer.
	FindComponentName (see page 5619)	This is FindComponentName, a member of class TComponentsPageProducer.
V	GeneratePageElements (see page 19823)	Fills the PageElements (see page 19830) property with information based on the other properties of the adapter Page Producer.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 18381)	This is GetChildren, a member of class TBasePageItemsProducer.
V	GetComponentContent (see page 5619)	This is GetComponentContent, a member of class TComponentsPageProducer.
	GetComponentList (see page 18382)	This is GetComponentList, a member of class TBasePageItemsProducer.
V	GetContentOptions (see page 5619)	This is GetContentOptions, a member of class TComponentsPageProducer.
V	GetContentOptions (see page 19823)	Returns an object that influences how the page producer generates its content.
	GetDefaultComponentList (see page 18382)	This is GetDefaultComponentList, a member of class TBasePageItemsProducer.
V	GetDefaultTemplate (see page 19824)	Returns the default template that the page producer can use to generate content.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetProducerTemplateFileName (see page 15276)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.
	GetProducerTemplateStream (see page 15276)	This is GetProducerTemplateStream, a member of class TBasePageProducer.
V	GetScriptEngine (see page 15277)	This is GetScriptEngine, a member of class TBasePageProducer.
	GetTagID (see page 15277)	This is GetTagID, a member of class TBasePageProducer.
V	GetTemplateFileName (see page 15277)	This is GetTemplateFileName, a member of class TBasePageProducer.
V	GetTemplateFileName (see page 19839)	Returns the value of the HTMLFile (see page 19843) property.
	GetTemplateFileType (see page 19825)	Returns the type of file that can store the templates this object uses to generate its content.
V	GetTemplateStream (see page 15277)	This is GetTemplateStream, a member of class TBasePageProducer.
V	GetTemplateStream (see page 19840)	Returns a stream from which to read the HTML template that this producer uses.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXMLData (see page 19825)	Returns a stream for reading any XML data associated with the page producer.
	GetXSLData (see page 19826)	Returns a stream for reading an XSL template associated with the page producer.
V	HandleScriptTag (see page 15277)	This is HandleScriptTag, a member of class TBasePageProducer.
V	HandleScriptTag (see page 19826)	Translates HTML-transparent tags for an associated server-side script.
V	HandleTag (see page 5620)	This is HandleTag, a member of class TComponentsPageProducer.
V	HandleTag (see page 15278)	This is HandleTag, a member of class TBasePageProducer.

	HasHTMLBrowserView (see page 19827)	Indicates whether the IDE should display an HTML browser view for the producer's content.
	HasScriptView (see page 19827)	Indicates whether the IDE should display an associated server-side script.
	HasXMLBrowserView (see page 19827)	Indicates whether the IDE should display an XML tree view for any XML data associated with the page producer.
	HasXSLBrowserView (see page 19828)	Indicates whether the IDE should display an XSL tree view for an XSL template associated with the page producer.
V	ImplCanAddClass (see page 19828)	Provides the underlying implementation for the CanAddClass (see page 19820) method.
	ImplHandleTag (see page 15278)	This is ImplHandleTag, a member of class TBasePageProducer.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 15108)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ServerScriptFromStream (see page 15278)	This is ServerScriptFromStream, a member of class TBasePageProducer.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 18382)	This is SetChildOrder, a member of class TBasePageItemsProducer.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHTMLDoc (see page 19840)	Sets the HTMLDoc (see page 19842) property.
	SetHTMLFile (see page 19840)	Sets the HTMLFile (see page 19843) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetWebContentOptions (see page 19829)	Sets the options that limit how the page producer creates its generated HTML.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
V	UseScriptEngine (see page 15278)	This is UseScriptEngine, a member of class TBasePageProducer.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebContent (see page 19841)	Returns the HTML document assembled by the Adapter page producer.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	PageElements (see page 19830)	Stores information that the Adapter page producer uses when generating its content.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Content (see page 15106)	Returns the content produced by the content producer.
V	Content (see page 15274)	Returns the content produced by the content producer.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	ContentFromStream (see page 15107)	Returns the content produced in response to input read from a stream.
V	ContentFromStream (see page 15275)	Returns the content produced in response to input read from a stream.
V	ContentFromStream (see page 19820)	Returns the content produced using a template read from a stream.
V	ContentFromString (see page 15107)	Returns the content produced in response to an input string.
V	ContentFromString (see page 15275)	Returns the content produced in response to an input string.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 15274)	Creates an instance of TBasePageProducer (see page 15269).
V	Create (see page 18380)	Creates an instance of a TBasePageItemsProducer (see page 18376).
V	Create (see page 19818)	Creates and initializes an instance of TBaseAdapterPageProducer (see page 19813).
V	Create (see page 19838)	Creates and initializes an instance of TCustomAdapterPageProducer (see page 19832).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 18381)	Destroys the instance of the TBasePageItemsProducer (see page 18376).
V	Destroy (see page 19818)	Frees an instance of TBaseAdapterPageProducer (see page 19813).
V	Destroy (see page 19839)	Frees an instance of TCustomAdapterPageProducer (see page 19832).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.

	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProducerTemplateFileName (see page 15320)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.
	GetProducerTemplateStream (see page 15320)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.
	GetRequiredTags (see page 19824)	Returns the HTML-transparent tags that the adapter page producer requires to have present in its template.
	GetScriptEngines (see page 19825)	Returns a list of script engines that can work with server-side scripts that produce content for the page producer.
	GetTemplateFileType (see page 24780)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 24780)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 24780)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 24780)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasScriptView (see page 24780)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 24781)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 24781)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetStyles (see page 19829)	Assigns a value to the Styles (see page 19830) property.
	SetStylesFile (see page 19829)	Assigns a value to the StylesFile (see page 19831) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetWebContentOptions (see page 24782)	This is SetWebContentOptions, a member of class ISetWebContentOptions.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.

R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ScriptEngine (see page 15279)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 15279)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebModuleContext (see page 15279)	Provides access to the Web module context object.

Published Events

	Name	Description
	OnAfterGetContent (see page 19812)	Occurs after the page producer generates its content.
	OnBeforeGetContent (see page 19812)	Occurs before the page producer generates its content.
	OnHTMLTag (see page 5620)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Published Properties

	Name	Description
	HTMLDoc (see page 19808)	Specifies the HTML template used to generate the value that the Content method returns.
	HTMLFile (see page 19809)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Styles (see page 19810)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 19811)	Specifies a file containing the individual styles for the page producer's style sheet.
	Tag (see page 2808)	Stores an integer value as part of a component.
	WebPageItems (see page 19811)	Maintains a list of components that generate HTML content for the producer.

136.1.1.2 TAdapterPageProducer Properties

The properties of the TAdapterPageProducer class are listed here.

Published Properties

	Name	Description
	HTMLDoc (see page 19808)	Specifies the HTML template used to generate the value that the Content method returns.
	HTMLFile (see page 19809)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	Styles (see page 19810)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 19811)	Specifies a file containing the individual styles for the page producer's style sheet.
	WebPageItems (see page 19811)	Maintains a list of components that generate HTML content for the producer.

136.1.1.2.1 TAdapterPageProducer.HTMLDoc Property

Specifies the HTML template used to generate the value that the Content method returns.

Pascal

```
property HTMLDoc: TStrings;
```

C++

```
__property TStrings HTMLDoc;
```

Description

Set HTMLDoc to a set of strings that make up the HTML template. By default, HTMLDoc is initialized to the following template:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#STYLES> <#WARNINGS> <#SERVERSCRIPT>
</BODY>
</HTML>
```

Each string is a sequence of one or more HTML commands or HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. TCustomAdapterPageProducer automatically handles the conversion of the HTML-transparent tags in the default template as follows:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles or StylesFile property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items.

<#SERVERSCRIPT> - This tag is replaced with generated HTML/JavaScript based on the hierarchy of Web items. TCustomAdapterPageProducer will not generate any content if this tag is missing.

If you set HTMLDoc to include other HTML-transparent tags, you must translate these in an OnHTMLTag event handler.

Note: Setting HTMLDoc clears the HTMLFile property.

See Also

Content (see page 15274)

HTMLFile (see page 19843)

OnHTMLTag (see page 15280)

136.1.1.2.2 TAdapterPageProducer.HTMLFile Property

Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLFile: TFileName;
```

C++

```
__property TFileName HTMLFile;
```

Description

Set HTMLFile to the name of the file that contains an HTML template. The HTML template is a sequence of HTML commands and HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLFile clears the HTMLDoc property.

Note: When distributing applications that contain page producers which use the HTMLFile property, be sure to distribute the file along with the application. Also, be sure that the value of HTMLFile does not include a hard-coded path name that will not be available on the user's disk.

See Also

Content (see page 15274)

HTMLDoc (see page 19842)

OnHTMLTag (see page 15280)

136.1.1.2.3 TAdapterPageProducer.Styles Property

Defines the styles that can be used by the page producer and its Web items.

Pascal

```
property Styles: TStrings;
```

C++

```
__property TStrings Styles;
```

Description

Use Styles to define a style sheet for the HTML document produced by TBaseAdapterPageProducer. Each string in the TStrings object specifies a specific style by giving a style selector followed by a set of attributes in curly braces. These definitions can define styles for standard HTML elements such as

```
H2 B  {color: red}
```

or they can define styles that you name, such as

```
.MyStyle  {font-family: arial; font-weight: bold; font-size: 18px }
```

Note that user-defined style names must begin with a dot.

Styles for standard elements are applied automatically to HTML that uses the tag or tags that form the style selector. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the

selector.

Note: If you are sharing a set of styles between multiple page producers or multiple applications, you can define the styles in a file. To define the styles in a file, use the StylesFile property instead. If StylesFile is defined, it takes precedence over the Styles property.

See Also

StylesFile (see page 19831)

Content (see page 15274)

136.1.1.2.4 TAdapterPageProducer.StylesFile Property

Specifies a file containing the individual styles for the page producer's style sheet.

Pascal

```
property StylesFile: TFileName;
```

C++

```
__property TFileName StylesFile;
```

Description

Set StylesFile to the name of a file containing the style definitions used by the page producer. The page producer uses these style definitions to declare a style sheet in the HTML document it generates.

Styles for standard elements are applied automatically to HTML in the document. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector of a style definition.

Note: If the StylesFile property is not set, you can specify the style definitions using the Styles property instead.

See Also

Styles (see page 19830)

Content (see page 15274)

136.1.1.2.5 TAdapterPageProducer.WebPageItems Property

Maintains a list of components that generate HTML content for the producer.

Pascal

```
property WebPageItems: TWebComponentList;
```

C++

```
__property TWebComponentList WebPageItems;
```

Description

WebPageItems lists the components that generate content for the page items producer. Each of these components supports the IWebContent interface, and can contain its own list of Web items that help it generate its content.

The type of items that can be added to the WebPageItems list are restricted by a protected method that is abstract or, in C++ terminology, pure virtual in TBasePageItemsProducer. Descendant classes implement this protected method to indicate what types of items can be added to the list.

At design time, use the Web page editor to add components to the WebPageItems property.

Note: Typically, WebPageItems lists all the components that generate content for the page items producer, in the order that the generated content appears in the final HTML document. However, the page items producer can transform COMPONENT tags in

its template using components that are not in this list, as long as they have the same owner as the page items producer..

See Also

IWebContent (see page 24792)

Creating Web Pages with an InternetExpress Page Producer

136.1.1.3 TAdapterPageProducer Events

The events of the TAdapterPageProducer class are listed here.

Published Events

	Name	Description
	OnAfterGetContent (see page 19812)	Occurs after the page producer generates its content.
	OnBeforeGetContent (see page 19812)	Occurs before the page producer generates its content.

136.1.1.3.1 TAdapterPageProducer.OnAfterGetContent Event

Occurs after the page producer generates its content.

Pascal

```
property OnAfterGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAfterGetContent;
```

Description

Write an OnAfterGetContent event handler to take specific action immediately after the producer generates its content.

See Also

ContentFromStream (see page 19820)

OnBeforeGetContent (see page 19832)

136.1.1.3.2 TAdapterPageProducer.OnBeforeGetContent Event

Occurs before the page producer generates its content.

Pascal

```
property OnBeforeGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnBeforeGetContent;
```

Description

Write an OnBeforeGetContent event handler to take specific action immediately before the producer generates its content.

See Also

ContentFromStream (see page 19820)

OnAfterGetContent (see page 19832)

136.1.2 TBaseAdapterPageProducer Class

TBaseAdapterPageProducer generates the proper HTML and JavaScript needed to display adapter fields and execute adapter actions.

Class Hierarchy

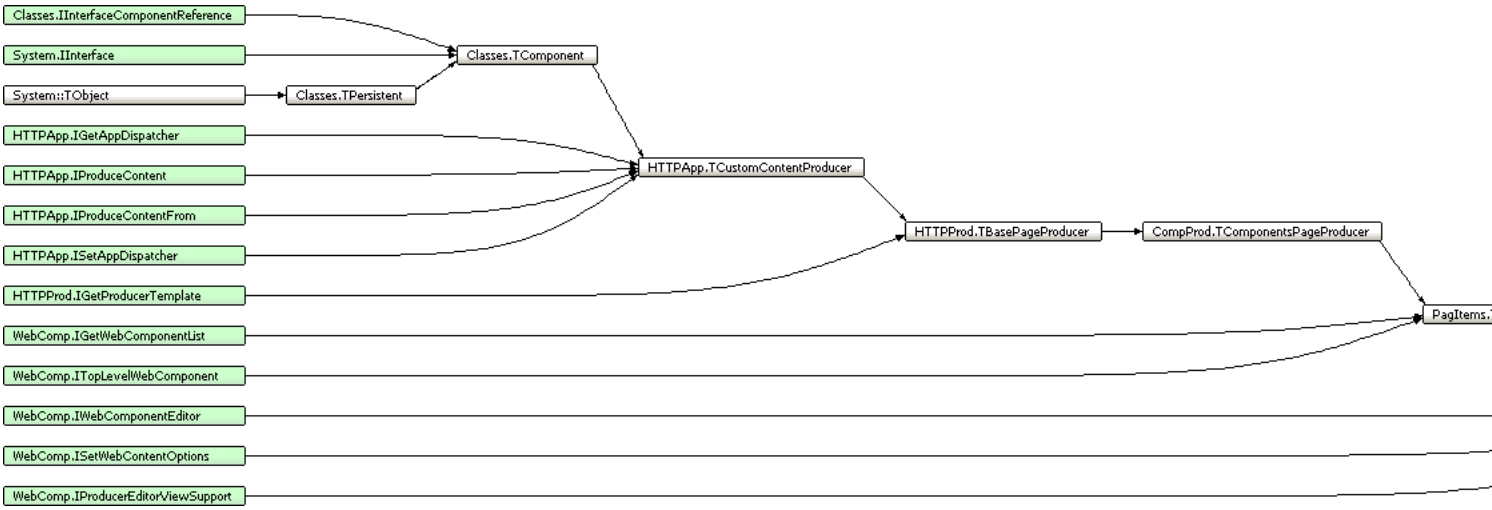

Pascal

```
TBaseAdapterPageProducer = class(TBasePageItemsProducer, IWebComponentEditor,
ISetWebContentOptions, IProducerEditorViewSupport);
```

C++

```
class TBaseAdapterPageProducer : public TBasePageItemsProducer, public IWebComponentEditor,
public ISetWebContentOptions, public IProducerEditorViewSupport;
```

File

SiteProd

Description

TBaseAdapterPageProducer creates an HTML page using content it obtains from a hierarchy of components called Web items. The Web items that appear in the TBaseAdapterPageProducer's hierarchy use Adapter components to fetch the information that they display.

Use TBaseAdapterPageProducer as a base class for page producer components that assemble their content from Web items that use Adapter components. TBaseAdapterPageProducer implements the logic for converting an HTML template into content it assembles from its hierarchy of Web items.

TBaseAdapterPageProducer implements the logic to translate the following special HTML-transparent tags:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles (see page 19830) or StylesFile (see page 19831) property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items.

<#SERVERSCRIPT> - This tag is replaced with generated HTML/JavaScript based on the hierarchy of Web items.

136.1.2.1 TBaseAdapterPageProducer Members

The following tables list the members exposed by TBaseAdapterPageProducer.

Protected Events

	Name	Description
	OnHTMLTag (see page 15280)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 19820)	Specifies whether a Web item of a specified type can be added to the WebPageItems property list.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CreatePageElements (see page 19821)	Creates the object that implements the PageElements (see page 19830) property.
V	DeclareStyles (see page 19821)	Returns an HTML Styles (see page 19830) tag that declares the styles specified by the Styles (see page 19830) or StylesFile (see page 19831) property.
V	DeclareWarnings (see page 19821)	Returns the formatted warnings that appear at design time to indicate the status of content generation.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoAfterGetContent (see page 19822)	Generates an OnAfterGetContent (see page 19832) event.
	DoBeforeGetContent (see page 19822)	Generates an OnBeforeGetContent (see page 19832) event.
V	DoTagEvent (see page 19822)	Translates HTML-transparent tags in the template.
V	FindComponent (see page 5619)	This is FindComponent, a member of class TComponentsPageProducer.
	FindComponentName (see page 5619)	This is FindComponentName, a member of class TComponentsPageProducer.
V	GeneratePageElements (see page 19823)	Fills the PageElements (see page 19830) property with information based on the other properties of the adapter Page Producer.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 18381)	This is GetChildren, a member of class TBasePageItemsProducer.
V	GetComponentContent (see page 5619)	This is GetComponentContent, a member of class TComponentsPageProducer.
	GetComponentList (see page 18382)	This is GetComponentList, a member of class TBasePageItemsProducer.
V	GetContentOptions (see page 19823)	Returns an object that influences how the page producer generates its content.
	GetDefaultComponentList (see page 18382)	This is GetDefaultComponentList, a member of class TBasePageItemsProducer.
V	GetDefaultTemplate (see page 19824)	Returns the default template that the page producer can use to generate content.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetProducerTemplateFileName (see page 15276)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.

	GetProducerTemplateStream (see page 15276)	This is GetProducerTemplateStream, a member of class TBasePageProducer.
	GetScriptEngine (see page 15277)	This is GetScriptEngine, a member of class TBasePageProducer.
	GetTagID (see page 15277)	This is GetTagID, a member of class TBasePageProducer.
	GetTemplateFileName (see page 15277)	This is GetTemplateFileName, a member of class TBasePageProducer.
	GetTemplateFileType (see page 19825)	Returns the type of file that can store the templates this object uses to generate its content.
	GetTemplateStream (see page 15277)	This is GetTemplateStream, a member of class TBasePageProducer.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXMLData (see page 19825)	Returns a stream for reading any XML data associated with the page producer.
	GetXSLData (see page 19826)	Returns a stream for reading an XSL template associated with the page producer.
	HandleScriptTag (see page 19826)	Translates HTML-transparent tags for an associated server-side script.
	HandleTag (see page 5620)	This is HandleTag, a member of class TComponentsPageProducer.
	HandleTag (see page 15278)	This is HandleTag, a member of class TBasePageProducer.
	HasHTMLBrowserView (see page 19827)	Indicates whether the IDE should display an HTML browser view for the producer's content.
	HasScriptView (see page 19827)	Indicates whether the IDE should display an associated server-side script.
	HasXMLBrowserView (see page 19827)	Indicates whether the IDE should display an XML tree view for any XML data associated with the page producer.
	HasXSLBrowserView (see page 19828)	Indicates whether the IDE should display an XSL tree view for an XSL template associated with the page producer.
	ImplCanAddClass (see page 19828)	Provides the underlying implementation for the CanAddClass (see page 19820) method.
	ImplHandleTag (see page 15278)	This is ImplHandleTag, a member of class TBasePageProducer.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 15108)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ServerScriptFromStream (see page 15278)	This is ServerScriptFromStream, a member of class TBasePageProducer.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 18382)	This is SetChildOrder, a member of class TBasePageItemsProducer.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetWebContentOptions (see page 19829)	Sets the options that limit how the page producer creates its generated HTML.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UseScriptEngine (see page 15278)	This is UseScriptEngine, a member of class TBasePageProducer.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.

	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	PageElements (see page 19830)	Stores information that the Adapter page producer uses when generating its content.

Public Events

	Name	Description
	OnAfterGetContent (see page 19832)	Occurs after the page producer generates its content.
	OnBeforeGetContent (see page 19832)	Occurs before the page producer generates its content.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Content (see page 15106)	Returns the content produced by the content producer.
V	Content (see page 15274)	Returns the content produced by the content producer.
V	ContentFromStream (see page 19820)	Returns the content produced using a template read from a stream.
V	ContentFromString (see page 15107)	Returns the content produced in response to an input string.
V	ContentFromString (see page 15275)	Returns the content produced in response to an input string.
V	Create (see page 19818)	Creates and initializes an instance of TBaseAdapterPageProducer (see page 19813).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 19818)	Frees an instance of TBaseAdapterPageProducer (see page 19813).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.

	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProducerTemplateFileName (see page 15320)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.
	GetProducerTemplateStream (see page 15320)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.
	GetRequiredTags (see page 19824)	Returns the HTML-transparent tags that the adapter page producer requires to have present in its template.
	GetScriptEngines (see page 19825)	Returns a list of script engines that can work with server-side scripts that produce content for the page producer.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetStyles (see page 19829)	Assigns a value to the Styles (see page 19830) property.
	SetStylesFile (see page 19829)	Assigns a value to the StylesFile (see page 19831) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.

R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ScriptEngine (see page 15279)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 15279)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.
	Styles (see page 19830)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 19831)	Specifies a file containing the individual styles for the page producer's style sheet.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebModuleContext (see page 15279)	Provides access to the Web module context object.
R	WebPageItems (see page 18382)	Maintains a list of components that generate HTML content for the producer.

Published Events

	Name	Description
	OnHTMLTag (see page 5620)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

136.1.2.2 TBaseAdapterPageProducer.Create Constructor

Creates and initializes an instance of TBaseAdapterPageProducer (see page 19813).

Pascal

```
constructor Create(AComponent: TComponent); override;
```

C++

```
virtual __fastcall TBaseAdapterPageProducer(TComponent * AComponent);
```

Description

Call Create to instantiate an adapter page producer at runtime. Adapter page producers placed in a Web page module at design time (or by the Web page module wizard) are instantiated automatically.

AOwner is a component, typically a Web page module, that becomes the value of the Owner property. The Owner is responsible for freeing the TBaseAdapterPageProducer (see page 19813) instance.

See Also

Owner (see page 2807)

136.1.2.3 TBaseAdapterPageProducer.Destroy Destructor

Frees an instance of TBaseAdapterPageProducer (see page 19813).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBaseAdapterPageProducer();
```

Description

Do not call Destroy directly. Instead, call the Free method, which checks whether the reference to the producer is not nil before calling Destroy.

See Also

Free (see page 22300)

136.1.2.4 TBaseAdapterPageProducer Methods

The methods of the TBaseAdapterPageProducer class are listed here.

Protected Methods

	Name	Description
	CanAddClass (see page 19820)	Specifies whether a Web item of a specified type can be added to the WebPageItems property list.
V	CreatePageElements (see page 19821)	Creates the object that implements the PageElements (see page 19830) property.
V	DeclareStyles (see page 19821)	Returns an HTML Styles (see page 19830) tag that declares the styles specified by the Styles (see page 19830) or StylesFile (see page 19831) property.
V	DeclareWarnings (see page 19821)	Returns the formatted warnings that appear at design time to indicate the status of content generation.
	DoAfterGetContent (see page 19822)	Generates an OnAfterGetContent (see page 19832) event.
	DoBeforeGetContent (see page 19822)	Generates an OnBeforeGetContent (see page 19832) event.
V	DoTagEvent (see page 19822)	Translates HTML-transparent tags in the template.
V	GeneratePageElements (see page 19823)	Fills the PageElements (see page 19830) property with information based on the other properties of the adapter Page Producer.
V	GetContentOptions (see page 19823)	Returns an object that influences how the page producer generates its content.
V	GetDefaultTemplate (see page 19824)	Returns the default template that the page producer can use to generate content.
	GetTemplateFileType (see page 19825)	Returns the type of file that can store the templates this object uses to generate its content.
	GetXMLData (see page 19825)	Returns a stream for reading any XML data associated with the page producer.
	GetXSLData (see page 19826)	Returns a stream for reading an XSL template associated with the page producer.
V	HandleScriptTag (see page 19826)	Translates HTML-transparent tags for an associated server-side script.
	HasHTMLBrowserView (see page 19827)	Indicates whether the IDE should display an HTML browser view for the producer's content.
	HasScriptView (see page 19827)	Indicates whether the IDE should display an associated server-side script.
	HasXMLBrowserView (see page 19827)	Indicates whether the IDE should display an XML tree view for any XML data associated with the page producer.
	HasXSLBrowserView (see page 19828)	Indicates whether the IDE should display an XSL tree view for an XSL template associated with the page producer.
V	ImplCanAddClass (see page 19828)	Provides the underlying implementation for the CanAddClass (see page 19820) method.
	SetWebContentOptions (see page 19829)	Sets the options that limit how the page producer creates its generated HTML.

Public Methods

	Name	Description
V	ContentFromStream (see page 19820)	Returns the content produced using a template read from a stream.
	GetRequiredTags (see page 19824)	Returns the HTML-transparent tags that the adapter page producer requires to have present in its template.
	GetScriptEngines (see page 19825)	Returns a list of script engines that can work with server-side scripts that produce content for the page producer.
	SetStyles (see page 19829)	Assigns a value to the Styles (see page 19830) property.
	SetStylesFile (see page 19829)	Assigns a value to the StylesFile (see page 19831) property.

136.1.2.4.1 TBaseAdapterPageProducer.CanAddClass Method

Specifies whether a Web item of a specified type can be added to the WebPageItems property list.

Pascal

```
function CanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClass(TComponent * AParent, TClass AClass);
```

Description

The IDE calls CanAddClass to determine what Web items can be added to supply content for the TBaseAdapterPageProducer (see page 19813) descendant. CanAddClass calls the virtual ImplCanAddClass (see page 19828) method to determine whether a class of Web item is valid.

AParent is the TBaseAdapterPageProducer (see page 19813) descendant to whose WebPageItems property the Web item will be added.

AClass is the class of the proposed child Web item.

CanAddClass returns true if the TBaseAdapterPageProducer (see page 19813) descendant can use content generated by a component of type AClass, false otherwise.

See Also

ImplCanAddClass (see page 19828)

WebPageItems (see page 18382)

136.1.2.4.2 TBaseAdapterPageProducer.ContentFromStream Method

Returns the content produced using a template read from a stream.

Pascal

```
function ContentFromStream(InStream: TStream): string; override;
```

C++

```
virtual __fastcall AnsiString ContentFromStream(TStream InStream);
```

Description

Call ContentFromStream to retrieve the content produced by the adapter page producer based on the template provided by the Stream parameter. ContentFromStream:

1. Generates an OnBeforeGetContent (see page 19832) event.
2. Translates the HTML template that is the value of Stream, using information produced by the items listed in the WebPageItems property.
3. Generates an OnAfterGetContent (see page 19832) event.
4. Returns the HTML produced in step 2.

See Also

Content (see page 15274)

OnBeforeGetContent (see page 19832)

WebPageItems (see page 18382)

OnAfterGetContent (see page 19832)

136.1.2.4.3 TBaseAdapterPageProducer.CreatePageElements Method

Creates the object that implements the PageElements (see page 19830) property.

Pascal

```
function CreatePageElements: TSitePageElements; virtual;
```

C++

```
virtual __fastcall TSitePageElements CreatePageElements();
```

Description

CreatePageElements creates the object that is used as the value of the PageElements (see page 19830) property. This object holds temporary information that the page producer uses to assemble its content.

See Also

PageElements (see page 19830)

GeneratePageElements (see page 19823)

136.1.2.4.4 TBaseAdapterPageProducer.DeclareStyles Method

Returns an HTML Styles (see page 19830) tag that declares the styles specified by the Styles (see page 19830) or StylesFile (see page 19831) property.

Pascal

```
function DeclareStyles: string; virtual;
```

C++

```
virtual __fastcall AnsiString DeclareStyles();
```

Description

The GeneratePageElements (see page 19823) method calls DeclareStyles to generate the HTML that declares a stylesheet in the generated HTML page. DeclareStyles uses the StylesFile (see page 19831) property to determine the styles it declares. If StylesFile (see page 19831) is an empty string, DeclareStyles uses the Styles (see page 19830) property instead. If both StylesFile (see page 19831) and Styles (see page 19830) are set to empty strings, DeclareStyles returns an empty string.

See Also

Styles (see page 19830)

StylesFile (see page 19831)

GeneratePageElements (see page 19823)

136.1.2.4.5 TBaseAdapterPageProducer.DeclareWarnings Method

Returns the formatted warnings that appear at design time to indicate the status of content generation.

Pascal

```
function DeclareWarnings(Warnings: TStrings): string; virtual;
```

C++

```
virtual __fastcall AnsiString DeclareWarnings(TStrings Warnings);
```

Description

The GeneratePageElements (see page 19823) method calls DeclareWarnings to generate the text that describes any design-time warnings. These warnings indicate possible problems with the hierarchy of web items.

See Also

GeneratePageElements (see page 19823)

WebPageItems (see page 18382)

136.1.2.4.6 TBaseAdapterPageProducer.DoAfterGetContent Method

Generates an OnAfterGetContent (see page 19832) event.

Pascal

```
procedure DoAfterGetContent;
```

C++

```
__fastcall DoAfterGetContent();
```

Description

TBaseAdapterPageProducer (see page 19813) calls DoAfterGetContent internally to generate the OnAfterGetContent (see page 19832) event.

See Also

OnAfterGetContent (see page 19832)

136.1.2.4.7 TBaseAdapterPageProducer.DoBeforeGetContent Method

Generates an OnBeforeGetContent (see page 19832) event.

Pascal

```
procedure DoBeforeGetContent;
```

C++

```
__fastcall DoBeforeGetContent();
```

Description

TBaseAdapterPageProducer (see page 19813) calls DoBeforeGetContent internally to generate the OnBeforeGetContent (see page 19832) event.

See Also

OnBeforeGetContent (see page 19832)

136.1.2.4.8 TBaseAdapterPageProducer.DoTagEvent Method

Translates HTML-transparent tags in the template.

Pascal

```
procedure DoTagEvent(Tag: TTag; const TagString: string; TagParams: TStrings; var ReplaceText:
string); override;
```

C++

```
virtual __fastcall DoTagEvent(TTag Tag, const AnsiString TagString, TStrings TagParams,
```

```
AnsiString ReplaceText);
```

Description

When the page producer encounters an HTML-transparent tag in its template, it calls DoTagEvent to translate that tag into the appropriate value. TBaseAdapterPageProducer (see page 19813) overrides the inherited method to handle the <#STYLES>, <#WARNINGS>, and <#SERVERSCRIPT> tags. For any other tags, DoTagEvent generates an OnHTMLTag event.

Tag is the type of HTML-transparent tag that was encountered in the HTML template.

TagString is tag name of the HTML-transparent tag. This is the first string that follows the pound sign (#).

TagParams gives the parameter portion of the HTML-transparent tag. Each string in the TStrings object has the form ParamName=Value.

ReplaceText returns the translated string that should replace the HTML-transparent tag.

See Also

OnHTMLTag (see page 15280)

136.1.2.4.9 TBaseAdapterPageProducer.GeneratePageElements Method

Fills the PageElements (see page 19830) property with information based on the other properties of the adapter Page Producer.

Pascal

```
procedure GeneratePageElements; virtual;
```

C++

```
virtual __fastcall GeneratePageElements();
```

Description

GeneratePageElements assembles information about the styles, warnings, server script, server functions, and server variables that the adapter page producer uses to generate its content. It writes this information into the members of the object that is the value of the PageElements (see page 19830) property.

See Also

PageElements (see page 19830)

136.1.2.4.10 TBaseAdapterPageProducer.GetContentOptions Method

Returns an object that influences how the page producer generates its content.

Pascal

```
function GetContentOptions(var Owned: Boolean): TWebContentOptions; override;
```

C++

```
virtual __fastcall TWebContentOptions GetContentOptions(Boolean Owned);
```

Description

TBaseAdapterPageProducer (see page 19813) calls GetContentOptions internally to look up the options that limit what can be included in the generated HTML.

Owned returns a value that indicates whether the return value should be freed by the caller. If Owned is true, the caller should free the TWebContentOptions object after it has used it.

See Also

SetWebContentOptions (see page 19829)

136.1.2.4.11 TBaseAdapterPageProducer.GetDefaultTemplate Method

Returns the default template that the page producer can use to generate content.

Pascal

```
function GetDefaultTemplate: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetDefaultTemplate();
```

Description

GetDefaultTemplate returns the default template that the page producer can use to generate its content. This template has the form:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#STYLES> <#WARNINGS> <#SERVERSCRIPT>
</BODY>
</HTML>
```

TBaseAdapterPageProducer (see page 19813) automatically handles the conversion of the HTML-transparent tags in the default template as follows:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles (see page 19830) or StylesFile (see page 19831) property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items.

<#SERVERSCRIPT> - This tag is replaced with generated HTML/JavaScript based on the hierarchy of Web items.

See Also

Styles (see page 19830)

StylesFile (see page 19831)

136.1.2.4.12 TBaseAdapterPageProducer.GetRequiredTags Method

Returns the HTML-transparent tags that the adapter page producer requires to have present in its template.

Pascal

```
class function GetRequiredTags: string;
```

C++

```
__fastcall AnsiString GetRequiredTags();
```

Description

GetRequiredTags returns a string containing the special tags that the page producer expects to find for generating its content. This method returns the string '<#STYLES> <#WARNINGS> <#SERVERSCRIPT>'.

See Also

ContentFromStream (see page 19820)

136.1.2.4.13 TBaseAdapterPageProducer.GetScriptEngines Method

Returns a list of script engines that can work with server-side scripts that produce content for the page producer.

Pascal

```
class procedure GetScriptEngines(S: TStrings);
```

C++

```
__fastcall GetScriptEngines(TStrings S);
```

Description

GetScriptEngines fills an existing string list with the set of all values that can be assigned to the ScriptEngine property. These determine the script engine that works with associated server-side scripts.

This method adds a single string, 'JSCRIPT', to the string list passed as the S parameter.

See Also

ScriptEngine (see page 15279)

136.1.2.4.14 TBaseAdapterPageProducer.GetTemplateFileType Method

Returns the type of file that can store the templates this object uses to generate its content.

Pascal

```
function GetTemplateFileType: string;
```

C++

```
__fastcall AnsiString GetTemplateFileType();
```

Description

GetTemplateFileType returns the string 'HTML', because Adapter page producers generate their contents based on an HTML template that includes HTML-transparent tags.

See Also

GetDefaultTemplate (see page 19824)

136.1.2.4.15 TBaseAdapterPageProducer.GetXMLData Method

Returns a stream for reading any XML data associated with the page producer.

Pascal

```
function GetXMLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXMLData(Boolean Owned);
```

Description

The IDE calls GetXMLData for producers that indicate that they support an XML browser view (that is, if the HasXMLBrowserView (see page 19827) method returns true). In TBaseAdapterPageProducer (see page 19813), GetXMLData always returns nil (Delphi) or NULL (C++), because adapter page producers do not use associated XML data.

Owned returns an indication of whether the caller needs to free the returned stream. In this case, this parameter is irrelevant, because GetXMLData always returns nil (Delphi) or NULL (C++).

See Also

HasXMLBrowserView (see page 19827)

136.1.2.4.16 TBaseAdapterPageProducer.GetXSLData Method

Returns a stream for reading an XSL template associated with the page producer.

Pascal

```
function GetXSLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXSLData(Boolean Owned);
```

Description

The IDE calls GetXSLData for producers that indicate that they support an XSL browser view (that is, if the HasXSLBrowserView (see page 19828) method returns true). In TBaseAdapterPageProducer (see page 19813), GetXSLData always returns nil (Delphi) or NULL (C++), because adapter page producers do not use an associated XSL template.

Owned returns an indication of whether the caller needs to free the returned stream. In this case, this parameter is irrelevant, because GetXSLData always returns nil (Delphi) or NULL (C++).

See Also

HasXSLBrowserView (see page 19828)

136.1.2.4.17 TBaseAdapterPageProducer.HandleScriptTag Method

Translates HTML-transparent tags for an associated server-side script.

Pascal

```
function HandleScriptTag(const TagString: string; TagParams: TStrings; var ReplaceString:
string): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandleScriptTag(const AnsiString TagString, TStrings TagParams,
AnsiString ReplaceString);
```

Description

When a server-side script encounters an HTML-transparent tag, it calls HandleScriptTag to allow the page producer to translate that tag for it. HandleScriptTag translates the <#SERVERSCRIPT> tag, supplying the content produced by the hierarchy of Web items that is the value of the WebPageItems property.

TagString is tag name of the HTML-transparent tag. This is the first string that follows the pound sign (#).

TagParams gives the parameter portion of the HTML-transparent tag. Each string in the TStrings object has the form ParamName=Value.

ReplaceText returns the translated string that should replace the HTML-transparent tag.

HandleScriptTag returns true if it can translate the tag (that is, if TagString is 'SERVERSCRIPT'). It returns false if TagString identifies any other type of tag.

See Also

WebPageItems (see page 18382)

136.1.2.4.18 TBaseAdapterPageProducer.HasHTMLBrowserView Method

Indicates whether the IDE should display an HTML browser view for the producer's content.

Pascal

```
function HasHTMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasHTMLBrowserView();
```

Description

The IDE calls HasHTMLBrowserView to determine whether it should create an HTML tab page in the code editor for displaying the content of the page producer.

HasHTMLBrowserView returns true. As a result, at design time, the value that the Content method returns appears on the HTML tab page of the code editor when the adapter page producer is selected.

See Also

HasScriptView (see page 19827)

HasXMLBrowserView (see page 19827)

HasXSLBrowserView (see page 19828)

Content (see page 15274)

136.1.2.4.19 TBaseAdapterPageProducer.HasScriptView Method

Indicates whether the IDE should display an associated server-side script.

Pascal

```
function HasScriptView: Boolean;
```

C++

```
__fastcall Boolean HasScriptView();
```

Description

The IDE calls HasScriptView to determine whether it should create a script tab page in the code editor for displaying the server-side script associated with the page producer.

HasScriptView returns true. As a result, at design time, the associated server-side script appears on a tab page of the code editor when the adapter page producer is selected.

See Also

HasHTMLBrowserView (see page 19827)

HasXMLBrowserView (see page 19827)

HasXSLBrowserView (see page 19828)

136.1.2.4.20 TBaseAdapterPageProducer.HasXMLBrowserView Method

Indicates whether the IDE should display an XML tree view for any XML data associated with the page producer.

Pascal

```
function HasXMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXMLBrowserView();
```

Description

The IDE calls HasXMLBrowserView to determine whether it should create an XML tab page in the code editor for displaying the XML data that the page producer uses.

HasXMLBrowserView returns false because adapter page producers do not use associated XML data.

See Also

HasScriptView (see page 19827)

HasHTMLBrowserView (see page 19827)

HasXSLBrowserView (see page 19828)

GetXMLData (see page 19825)

136.1.2.4.21 TBaseAdapterPageProducer.HasXSLBrowserView Method

Indicates whether the IDE should display an XSL tree view for an XSL template associated with the page producer.

Pascal

```
function HasXSLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXSLBrowserView();
```

Description

The IDE calls HasXSLBrowserView to determine whether it should create an XSL tab page in the code editor for displaying the XSL template that the page producer uses.

HasXSLBrowserView returns false because adapter page producers do not use associated XSL templates.

See Also

HasScriptView (see page 19827)

HasHTMLBrowserView (see page 19827)

HasXMLBrowserView (see page 19827)

GetXSLData (see page 19826)

136.1.2.4.22 TBaseAdapterPageProducer.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass (see page 19820) method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass (see page 19820) method calls ImpleCanAddClass to determine whether the TBaseAdapterPageProducer (see page 19813) descendant can add a component of type AClass to its WebPageItems property list. AParent is the TBaseAdapterPageProducer (see page 19813) instance to which the component will be added.

In TBaseAdapterPageProducer (see page 19813), ImplCanAddClass returns true if AClass descends from TCustomAdapterForm or TCustomLayoutGroup.

See Also

CanAddClass (see page 19820)

WebPageItems (see page 18382)

TCustomAdapterForm (see page 25276)

TCustomLayoutGroup (see page 16528)

136.1.2.4.23 TBaseAdapterPageProducer.SetStyles Method

Assigns a value to the Styles (see page 19830) property.

Pascal

```
procedure SetStyles(Value: TStrings);
```

C++

```
__fastcall SetStyles(TStrings Value);
```

Description

SetStyles is the write implementation of the Styles (see page 19830) property.

See Also

Styles (see page 19830)

136.1.2.4.24 TBaseAdapterPageProducer.SetStylesFile Method

Assigns a value to the StylesFile (see page 19831) property.

Pascal

```
procedure SetStylesFile(const Value: TFileName);
```

C++

```
__fastcall SetStylesFile(const TFileName Value);
```

Description

SetStylesFile is the write implementation of the StylesFile (see page 19831) property.

See Also

StylesFile (see page 19831)

136.1.2.4.25 TBaseAdapterPageProducer.SetWebContentOptions Method

Sets the options that limit how the page producer creates its generated HTML.

Pascal

```
procedure SetWebContentOptions(AOptions: TWebContentOptions);
```

C++

```
__fastcall SetWebContentOptions(TWebContentOptions AOptions);
```

Description

SetWebContentOptions assigns a set of flags to the adapter page producer that limit what it can include in generated HTML.

AOptions is the TWebContentOptions object that represents the set of options for this page producer. TBaseAdapterPageProducer (see page 19813) takes ownership of the object, so the caller should not free it.

See Also

GetContentOptions (see page 19823)

136.1.2.5 TBaseAdapterPageProducer Properties

The properties of the TBaseAdapterPageProducer class are listed here.

Protected Properties

	Name	Description
R	PageElements (see page 19830)	Stores information that the Adapter page producer uses when generating its content.

Public Properties

	Name	Description
	Styles (see page 19830)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 19831)	Specifies a file containing the individual styles for the page producer's style sheet.

136.1.2.5.1 TBaseAdapterPageProducer.PageElements Property

Stores information that the Adapter page producer uses when generating its content.

Pascal

```
property PageElements: TSitePageElements;
```

C++

```
__property TSitePageElements PageElements;
```

Description

TBaseAdapterPageProducer (see page 19813) uses PageElements internally to store styles, warnings, server script, server functions, and server variables. It uses the properties of PageElements to assemble its content.

See Also

Content (see page 15274)

OnHTMLTag (see page 15280)

136.1.2.5.2 TBaseAdapterPageProducer.Styles Property

Defines the styles that can be used by the page producer and its Web items.

Pascal

```
property Styles: TStrings;
```

C++

```
__property TStrings Styles;
```

Description

Use Styles to define a style sheet for the HTML document produced by TBaseAdapterPageProducer (see page 19813). Each string in the TStrings object specifies a specific style by giving a style selector followed by a set of attributes in curly braces. These definitions can define styles for standard HTML elements such as

```
H2 B  {color: red}
```

or they can define styles that you name, such as

```
.MyStyle  {font-family: arial; font-weight: bold; font-size: 18px }
```

Note that user-defined style names must begin with a dot.

Styles for standard elements are applied automatically to HTML that uses the tag or tags that form the style selector. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector.

Note: If you are sharing a set of styles between multiple page producers or multiple applications, you can define the styles in a file. To define the styles in a file, use the StylesFile (see page 19831) property instead. If StylesFile (see page 19831) is defined, it takes precedence over the Styles property.

See Also

StylesFile (see page 19831)

Content (see page 15274)

136.1.2.5.3 TBaseAdapterPageProducer.StylesFile Property

Specifies a file containing the individual styles for the page producer's style sheet.

Pascal

```
property StylesFile: TFileName;
```

C++

```
__property TFileName StylesFile;
```

Description

Set StylesFile to the name of a file containing the style definitions used by the page producer. The page producer uses these style definitions to declare a style sheet in the HTML document it generates.

Styles (see page 19830) for standard elements are applied automatically to HTML in the document. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector of a style definition.

Note: If the StylesFile property is not set, you can specify the style definitions using the Styles (see page 19830) property instead.

See Also

Styles (see page 19830)

Content (see page 15274)

136.1.2.6 TBaseAdapterPageProducer Events

The events of the TBaseAdapterPageProducer class are listed here.

Public Events

	Name	Description
	OnAfterGetContent (see page 19832)	Occurs after the page producer generates its content.
	OnBeforeGetContent (see page 19832)	Occurs before the page producer generates its content.

136.1.2.6.1 TBaseAdapterPageProducer.OnAfterGetContent Event

Occurs after the page producer generates its content.

Pascal

```
property OnAfterGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAfterGetContent;
```

Description

Write an OnAfterGetContent event handler to take specific action immediately after the producer generates its content.

See Also

ContentFromStream (see page 19820)

OnBeforeGetContent (see page 19832)

136.1.2.6.2 TBaseAdapterPageProducer.OnBeforeGetContent Event

Occurs before the page producer generates its content.

Pascal

```
property OnBeforeGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnBeforeGetContent;
```

Description

Write an OnBeforeGetContent event handler to take specific action immediately before the producer generates its content.

See Also

ContentFromStream (see page 19820)

OnAfterGetContent (see page 19832)

136.1.3 TCustomAdapterPageProducer Class

tcustomadapterpageproducer generates the proper HTML and JavaScript needed to display adapter fields and execute adapter actions.

Class Hierarchy

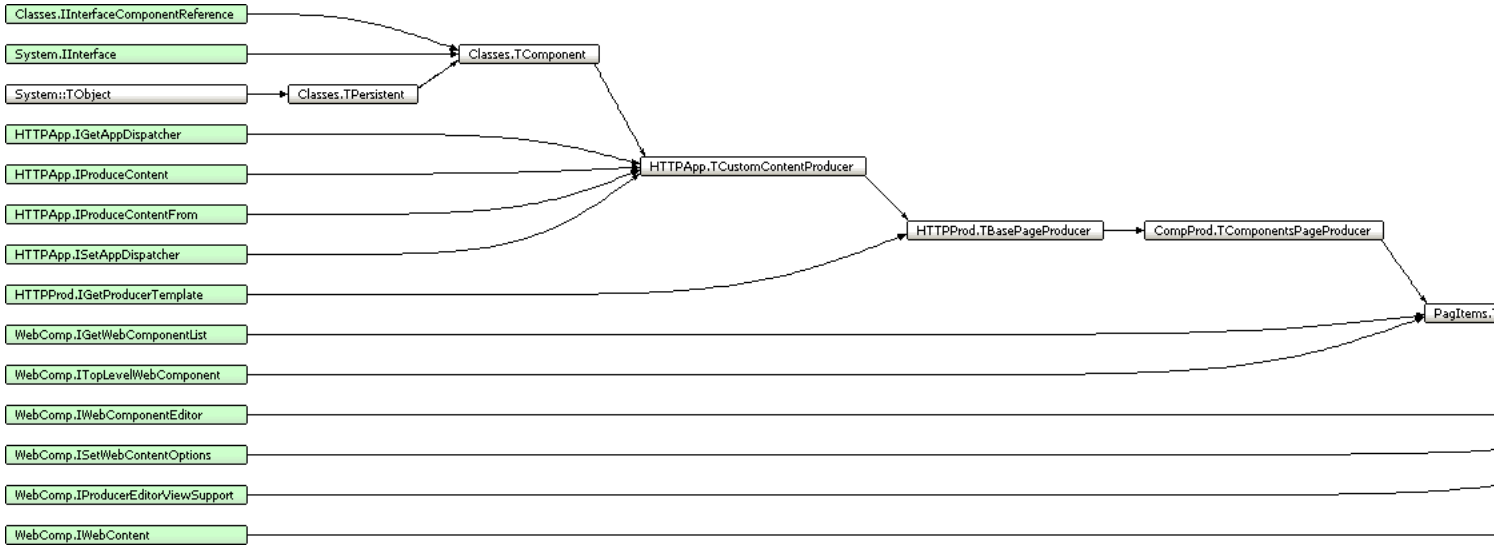

Pascal

```
TCustomAdapterPageProducer = class(TBaseAdapterPageProducer, IWebContent);
```

C++

```
class TCustomAdapterPageProducer : public TBaseAdapterPageProducer, public IWebContent;
```

File

SiteProd

Description

tcustomadapterpageproducer creates an HTML page using content it obtains from a hierarchy of components called Web items. The Web items that appear in the tcustomadapterpageproducer's hierarchy use Adapter components to fetch the information that they display.

Use tcustomadapterpageproducer as a base class for page producer components that assemble their content from Web items that use Adapter components.

tcustomadapterpageproducer introduces two properties, HTMLDoc (see page 19842) and HTMLFile (see page 19843), each of which can specify a template that the page producer uses to generate its content. By default, this template is set to include the following special tags:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles (see page 19830) or StylesFile (see page 19831) property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items.

<#SERVERSCRIPT> - This tag is replaced with generated HTML/JavaScript based on the hierarchy of Web items. tcustomadapterpageproducer will not generate any content if this tag is missing.

Applications can alter the template to further embellish the content that the page producer generates.

136.1.3.1 TCustomAdapterPageProducer Members

The following tables list the members exposed by TCustomAdapterPageProducer.

Protected Events

	Name	Description
	OnHTMLTag (see page 15280)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 19820)	Specifies whether a Web item of a specified type can be added to the WebPageItems property list.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CreatePageElements (see page 19821)	Creates the object that implements the PageElements (see page 19830) property.
V	DeclareStyles (see page 19821)	Returns an HTML Styles (see page 19830) tag that declares the styles specified by the Styles (see page 19830) or StylesFile (see page 19831) property.
V	DeclareWarnings (see page 19821)	Returns the formatted warnings that appear at design time to indicate the status of content generation.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoAfterGetContent (see page 19822)	Generates an OnAfterGetContent (see page 19832) event.
	DoBeforeGetContent (see page 19822)	Generates an OnBeforeGetContent (see page 19832) event.
V	DoTagEvent (see page 5618)	This is DoTagEvent, a member of class TComponentsPageProducer.
	DoTagEvent (see page 15276)	This is DoTagEvent, a member of class TBasePageProducer.
V	DoTagEvent (see page 19822)	Translates HTML-transparent tags in the template.
V	FindComponent (see page 5619)	This is FindComponent, a member of class TComponentsPageProducer.
	FindComponentName (see page 5619)	This is FindComponentName, a member of class TComponentsPageProducer.
V	GeneratePageElements (see page 19823)	Fills the PageElements (see page 19830) property with information based on the other properties of the adapter Page Producer.
	GetAppDispatcher (see page 15108)	This is GetAppDispatcher, a member of class TCustomContentProducer.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 18381)	This is GetChildren, a member of class TBasePageItemsProducer.
V	GetComponentContent (see page 5619)	This is GetComponentContent, a member of class TComponentsPageProducer.
	GetComponentList (see page 18382)	This is GetComponentList, a member of class TBasePageItemsProducer.
V	GetContentOptions (see page 5619)	This is GetContentOptions, a member of class TComponentsPageProducer.
V	GetContentOptions (see page 19823)	Returns an object that influences how the page producer generates its content.
	GetDefaultComponentList (see page 18382)	This is GetDefaultComponentList, a member of class TBasePageItemsProducer.
V	GetDefaultTemplate (see page 19824)	Returns the default template that the page producer can use to generate content.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetProducerTemplateFileName (see page 15276)	This is GetProducerTemplateFileName, a member of class TBasePageProducer.
	GetProducerTemplateStream (see page 15276)	This is GetProducerTemplateStream, a member of class TBasePageProducer.
V	GetScriptEngine (see page 15277)	This is GetScriptEngine, a member of class TBasePageProducer.

	GetTagID (see page 15277)	This is GetTagID, a member of class TBasePageProducer.
	GetTemplateFileName (see page 19839)	Returns the value of the HTMLFile (see page 19843) property.
	GetTemplateFileType (see page 19825)	Returns the type of file that can store the templates this object uses to generate its content.
	GetTemplateStream (see page 19840)	Returns a stream from which to read the HTML template that this producer uses.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXMLData (see page 19825)	Returns a stream for reading any XML data associated with the page producer.
	GetXSLData (see page 19826)	Returns a stream for reading an XSL template associated with the page producer.
	HandleScriptTag (see page 15277)	This is HandleScriptTag, a member of class TBasePageProducer.
	HandleScriptTag (see page 19826)	Translates HTML-transparent tags for an associated server-side script.
	HandleTag (see page 5620)	This is HandleTag, a member of class TComponentsPageProducer.
	HandleTag (see page 15278)	This is HandleTag, a member of class TBasePageProducer.
	HasHTMLBrowserView (see page 19827)	Indicates whether the IDE should display an HTML browser view for the producer's content.
	HasScriptView (see page 19827)	Indicates whether the IDE should display an associated server-side script.
	HasXMLBrowserView (see page 19827)	Indicates whether the IDE should display an XML tree view for any XML data associated with the page producer.
	HasXSLBrowserView (see page 19828)	Indicates whether the IDE should display an XSL tree view for an XSL template associated with the page producer.
	ImplCanAddClass (see page 19828)	Provides the underlying implementation for the CanAddClass (see page 19820) method.
	ImplHandleTag (see page 15278)	This is ImplHandleTag, a member of class TBasePageProducer.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 15108)	Overrides the inherited method to respond to messages that the dispatcher is going away.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 15108)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15109)	This is ProduceContentFromStream, a member of class TCustomContentProducer.
	ProduceContentFromString (see page 15109)	This is ProduceContentFromString, a member of class TCustomContentProducer.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ServerScriptFromStream (see page 15278)	This is ServerScriptFromStream, a member of class TBasePageProducer.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 15109)	This is SetAppDispatcher, a member of class TCustomContentProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 18382)	This is SetChildOrder, a member of class TBasePageItemsProducer.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHTMLDoc (see page 19840)	Sets the HTMLDoc (see page 19842) property.
	SetHTMLFile (see page 19840)	Sets the HTMLFile (see page 19843) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetWebContentOptions (see page 19829)	Sets the options that limit how the page producer creates its generated HTML.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UseScriptEngine (see page 15278)	This is UseScriptEngine, a member of class TBasePageProducer.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebContent (see page 19841)	Returns the HTML document assembled by the Adapter page producer.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	PageElements (see page 19830)	Stores information that the Adapter page producer uses when generating its content.

Public Events

	Name	Description
	OnAfterGetContent (see page 19832)	Occurs after the page producer generates its content.
	OnBeforeGetContent (see page 19832)	Occurs before the page producer generates its content.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Content (see page 15106)	Returns the content produced by the content producer.
V	Content (see page 15274)	Returns the content produced by the content producer.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	ContentFromStream (see page 15107)	Returns the content produced in response to input read from a stream.
V	ContentFromStream (see page 15275)	Returns the content produced in response to input read from a stream.
V	ContentFromStream (see page 19820)	Returns the content produced using a template read from a stream.
V	ContentFromString (see page 15107)	Returns the content produced in response to an input string.
V	ContentFromString (see page 15275)	Returns the content produced in response to an input string.
V	Create (see page 19838)	Creates and initializes an instance of TCustomAdapterPageProducer (see page 19832).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 19839)	Frees an instance of TCustomAdapterPageProducer (see page 19832).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProducerTemplateFileName (see page 15320)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.
	GetProducerTemplateStream (see page 15320)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.
	GetRequiredTags (see page 19824)	Returns the HTML-transparent tags that the adapter page producer requires to have present in its template.
	GetScriptEngines (see page 19825)	Returns a list of script engines that can work with server-side scripts that produce content for the page producer.
	GetTemplateFileType (see page 24780)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 24780)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 24780)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 24780)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasScriptView (see page 24780)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 24781)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 24781)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetStyles (see page 19829)	Assigns a value to the Styles (see page 19830) property.
	SetStylesFile (see page 19829)	Assigns a value to the StylesFile (see page 19831) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetWebContentOptions (see page 24782)	This is SetWebContentOptions, a member of class ISetWebContentOptions.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 15109)	Provides access to the dispatcher for the Web server application that includes the TCustomContentProducer (see page 15102) object.
R	DispatcherComponent (see page 15110)	Provides access to the dispatcher component.
	HTMLDoc (see page 19842)	Specifies the HTML template used to generate the value that the Content method returns.
	HTMLFile (see page 19843)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ScriptEngine (see page 15279)	Names the script engine that generates or interprets script embedded in the documents that this producer creates.
	StripParamQuotes (see page 15279)	Specifies whether to strip quotation marks when extracting the parameters from an HTML-transparent tag.
	Styles (see page 19830)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 19831)	Specifies a file containing the individual styles for the page producer's style sheet.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebModuleContext (see page 15279)	Provides access to the Web module context object.
R	WebPageItems (see page 18382)	Maintains a list of components that generate HTML content for the producer.

Published Events

	Name	Description
	OnHTMLTag (see page 5620)	Occurs when the page producer needs to interpret the value of an HTML-transparent tag in the HTML template.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

136.1.3.2 TCustomAdapterPageProducer.Create Constructor

Creates and initializes an instance of TCustomAdapterPageProducer (see page 19832).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterPageProducer(TComponent * AOwner);
```

Description

Call Create to instantiate an adapter page producer at runtime. Adapter page producers placed in a Web page module at design time (or by the Web page module wizard) are instantiated automatically.

AOwner is a component, typically a Web page module, that becomes the value of the Owner property. The Owner is responsible

for freeing the TCustomAdapterPageProducer (see page 19832) instance.

See Also

Owner (see page 2807)

136.1.3.3 TCustomAdapterPageProducer.Destroy Destructor

Frees an instance of TCustomAdapterPageProducer (see page 19832).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterPageProducer();
```

Description

Do not call Destroy directly. Instead, call the Free method, which checks whether the reference to the producer is not nil before calling Destroy.

See Also

Free (see page 22300)

136.1.3.4 TCustomAdapterPageProducer Methods

The methods of the TCustomAdapterPageProducer class are listed here.

Protected Methods

	Name	Description
V	GetTemplateFileName (see page 19839)	Returns the value of the HTMLFile (see page 19843) property.
V	GetTemplateStream (see page 19840)	Returns a stream from which to read the HTML template that this producer uses.
	SetHTMLDoc (see page 19840)	Sets the HTMLDoc (see page 19842) property.
	SetHTMLFile (see page 19840)	Sets the HTMLFile (see page 19843) property.
	WebContent (see page 19841)	Returns the HTML document assembled by the Adapter page producer.

136.1.3.4.1 TCustomAdapterPageProducer.GetTemplateFileName Method

Returns the value of the HTMLFile (see page 19843) property.

Pascal

```
function GetTemplateFileName: string; override;
```

C++

```
virtual __fastcall AnsiString GetTemplateFileName();
```

Description

TBasePageProducer introduces the protected GetTemplateFileName method to obtain the name of a file that contains a template from which to produce content. Each page producer component overrides GetTemplateFileName to supply this template in a class-specific way. TCustomAdapterPageProducer (see page 19832) returns the value of the HTMLFile (see page 19843) property.

See Also

GetTemplateStream (see page 19840)

HTMLFile (see page 19843)

136.1.3.4.2 TCustomAdapterPageProducer.GetTemplateStream Method

Returns a stream from which to read the HTML template that this producer uses.

Pascal

```
function GetTemplateStream(out AOwned: Boolean): TStream; override;
```

C++

```
virtual __fastcall TStream GetTemplateStream(Boolean AOwned);
```

Description

TBasePageProducer introduces the protected GetTemplateStream method to obtain a stream from which to read the producer component's HTML template. Each page producer component overrides GetTemplateStream to supply this stream in a class-specific way.

AOwned returns a value that indicates whether the caller should free the returned stream after reading the template. When AOwned is true, the caller should free the stream object.

TCustomAdapterPageProducer (see page 19832) returns a stream from which the template specified by the HTMLFile (see page 19843) or HTMLDoc (see page 19842) property can be read.

See Also

GetTemplateFileName (see page 19839)

HTMLFile (see page 19843)

HTMLDoc (see page 19842)

136.1.3.4.3 TCustomAdapterPageProducer.SetHTMLDoc Method

Sets the HTMLDoc (see page 19842) property.

Pascal

```
procedure SetHTMLDoc(const Value: TStrings);
```

C++

```
__fastcall SetHTMLDoc(const TStrings Value);
```

Description

SetHTMLDoc is the protected write implementation of the HTMLDoc (see page 19842) property.

Value is a TStrings descendant that contains the new HTML template. SetHTMLDoc assigns the strings in this object to the TStrings object that implements the HTMLDoc (see page 19842) property. Note that this method does not take ownership of the specified object.

See Also

HTMLDoc (see page 19842)

Assign (see page 3043)

136.1.3.4.4 TCustomAdapterPageProducer.SetHTMLFile Method

Sets the HTMLFile (see page 19843) property.

Pascal

```
procedure SetHTMLFile(const Value: TFileName);
```

C++

```
__fastcall SetHTMLFile(const TFileName Value);
```

Description

SetHTMLFile is the protected write implementation of the HTMLFile (see page 19843) property.

Value is the name of the template that the page producer should use to generate its content.

See Also

HTMLFile (see page 19843)

136.1.3.4.5 TCustomAdapterPageProducer.WebContent Method

Returns the HTML document assembled by the Adapter page producer.

Pascal

```
function WebContent(Options: TWebContentOptions; Layout: TLayout): string;
```

C++

```
__fastcall AnsiString WebContent(TWebContentOptions Options, TLayout Layout);
```

Description

Call WebContent to get the HTML document produced by TCustomAdapterPageProducer (see page 19832). TCustomAdapterPageProducer (see page 19832) uses WebContent to implement the Content method of its IWebContent interface.

Options contains a set of flags that limit what can be included in the generated HTML.

Layout is ignored by the WebContent method. It is included as a parameter for compatibility with other content producers that use a helper object to coordinate the layout of content produced by multiple Web items.

The Adapter page producer creates its content by replacing HTML-transparent tags in the template specified by HTMLDoc (see page 19842) or HTMLFile (see page 19843).

By default, this template has the form

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#STYLES> <#WARNINGS> <#SERVERSCRIPT>
</BODY>
</HTML>
```

Although the template can be changed to include any combination of these tags with custom HTML or tags that the application translates in an OnHTMLTag event handler.

The WebContent method translates the tags listed above as follows:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles (see page 19830) or StylesFile (see page 19831) property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items specified by the WebPageItems property.

<#SERVERSCRIPT> - This tag is replaced with generated HTML/JavaScript based on the hierarchy of Web items.

TCustomAdapterPageProducer (see page 19832) will not generate any content if this tag is missing.

The string returned by WebContent can be used as the Content property of the response to an HTTP request message.

See Also

IWebContent (see page 24792)

Styles (see page 19830)

StylesFile (see page 19831)

OnHTMLTag (see page 15280)

HTMLDoc (see page 19842)

HTMLFile (see page 19843)

WebPageItems (see page 18382)

136.1.3.5 TCustomAdapterPageProducer Properties

The properties of the TCustomAdapterPageProducer class are listed here.

Public Properties

	Name	Description
	HTMLDoc (see page 19842)	Specifies the HTML template used to generate the value that the Content method returns.
	HTMLFile (see page 19843)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.

136.1.3.5.1 TCustomAdapterPageProducer.HTMLDoc Property

Specifies the HTML template used to generate the value that the Content method returns.

Pascal

```
property HTMLDoc: TStrings;
```

C++

```
__property TStrings HTMLDoc;
```

Description

Set HTMLDoc to a set of strings that make up the HTML template. By default, HTMLDoc is initialized to the following template:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#STYLES> <#WARNINGS> <#SERVERSCRIPT>
</BODY>
</HTML>
```

Each string is a sequence of one or more HTML commands or HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white

space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. TCustomAdapterPageProducer (see page 19832) automatically handles the conversion of the HTML-transparent tags in the default template as follows:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles (see page 19830) or StylesFile (see page 19831) property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items.

<#SERVERSCRIPT> - This tag is replaced with generated HTML/JavaScript based on the hierarchy of Web items. TCustomAdapterPageProducer (see page 19832) will not generate any content if this tag is missing.

If you set HTMLDoc to include other HTML-transparent tags, you must translate these in an OnHTMLTag event handler.

Note: Setting HTMLDoc clears the HTMLFile (see page 19843) property.

See Also

Content (see page 15274)

HTMLFile (see page 19843)

OnHTMLTag (see page 15280)

136.1.3.5.2 TCustomAdapterPageProducer.HTMLFile Property

Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLFile: TFileName;
```

C++

```
__property TFileName HTMLFile;
```

Description

Set HTMLFile to the name of the file that contains an HTML template. The HTML template is a sequence of HTML commands and HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLFile clears the HTMLDoc (see page 19842) property.

Note: When distributing applications that contain page producers which use the HTMLFile property, be sure to distribute the file along with the application. Also, be sure that the value of HTMLFile does not include a hard-coded path name that will not be available on the user's disk.

See Also

Content (see page 15274)

HTMLDoc (see page 19842)

OnHTMLTag (see page 15280)

136.1.4 TSitePageElements Class

TSitePageElements stores part of the content that page producers generate.

Class Hierarchy

Pascal

```
TSitePageElements = class(TObject);
```

C++

```
class TSitePageElements : public TObject;
```

File

SiteProd

Description

TSitePageElements stores styles, warnings, server script, server functions, and server variables. These values are stored as class members of type string (not properties). The adapter page producer combines them to make up its generated content.

136.1.4.1 TSitePageElements Members

The following tables list the members exposed by TSitePageElements.

Public Fields

	Name	Description
	ServerFunctionsContent (see page 19845)	This is ServerFunctionsContent, a member of class TSitePageElements.
	ServerScriptContent (see page 19845)	This is ServerScriptContent, a member of class TSitePageElements.
	ServerVariablesContent (see page 19846)	This is ServerVariablesContent, a member of class TSitePageElements.
	StylesContent (see page 19846)	This is StylesContent, a member of class TSitePageElements.
	WarningsContent (see page 19846)	This is WarningsContent, a member of class TSitePageElements.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

136.1.4.2 TSitePageElements Fields

The fields of the TSitePageElements class are listed here.

Public Fields

	Name	Description
	ServerFunctionsContent (see page 19845)	This is ServerFunctionsContent, a member of class TSitePageElements.
	ServerScriptContent (see page 19845)	This is ServerScriptContent, a member of class TSitePageElements.
	ServerVariablesContent (see page 19846)	This is ServerVariablesContent, a member of class TSitePageElements.
	StylesContent (see page 19846)	This is StylesContent, a member of class TSitePageElements.
	WarningsContent (see page 19846)	This is WarningsContent, a member of class TSitePageElements.

136.1.4.2.1 TSitePageElements.ServerFunctionsContent Field

Pascal

```
ServerFunctionsContent: string;
```

C++

```
AnsiString ServerFunctionsContent;
```

Description

This is ServerFunctionsContent, a member of class TSitePageElements.

136.1.4.2.2 TSitePageElements.ServerScriptContent Field

Pascal

```
ServerScriptContent: string;
```

C++

```
AnsiString ServerScriptContent;
```

Description

This is ServerScriptContent, a member of class TSitePageElements.

136.1.4.2.3 TSitePageElements.ServerVariablesContent Field

Pascal

```
ServerVariablesContent: string;
```

C++

```
AnsiString ServerVariablesContent;
```

Description

This is ServerVariablesContent, a member of class TSitePageElements.

136.1.4.2.4 TSitePageElements.StylesContent Field

Pascal

```
StylesContent: string;
```

C++

```
AnsiString StylesContent;
```

Description

This is StylesContent, a member of class TSitePageElements.

136.1.4.2.5 TSitePageElements.WarningsContent Field

Pascal

```
WarningsContent: string;
```

C++

```
AnsiString WarningsContent;
```

Description

This is WarningsContent, a member of class TSitePageElements.

137 SOAPAttachIntf Namespace

This is namespace SOAPAttachIntf.

Topics

Name	Description
Classes (see page 19848)	The following table lists classes in this documentation.
Interfaces (see page 19853)	The following table lists interfaces in this documentation.
Types (see page 19862)	The following table lists types in this documentation.

Classes

	Name	Description
	TSoapDataList (see page 19848)	TSoapDataList maintains a list of SOAP attachments.

Interfaces

	Name	Description
	IMimeAttachmentHandler (see page 19853)	IMimeAttachmentHandler is the interface for interpreting or manipulating the multipart form that represents a Web Service call that includes an attachment.

Types

Name	Description
TOnGetAttachmentEvent (see page 19862)	This is type SOAPAttachIntf.TOnGetAttachmentEvent.
TOnSendAttachmentEvent (see page 19862)	This is type SOAPAttachIntf.TOnSendAttachmentEvent.

137.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TSoapDataList (see page 19848)	TSoapDataList maintains a list of SOAP attachments.

137.1.1 TSoapDataList Class

TSoapDataList maintains a list of SOAP attachments.

Class Hierarchy

System::TObject → Classes.TList → SOAPAttachIntf.TSoapDataList

Pascal

```
TSoapDataList = class(TList);
```

C++

```
class TSoapDataList : public TList;
```

File

SOAPAttachIntf

Description

Use TSoapDataList to store and maintain a list of SOAP attachments (TSOAPAttachment objects).

See Also

TList (see page 2868)

TSoapAttachment

137.1.1.1 TSoapDataList Members

The following tables list the members exposed by TSoapDataList.

Protected Methods

	Name	Description
	Get (see page 2878)	This is Get, a member of class TList.
V	Grow (see page 2879)	This is Grow, a member of class TList.
V	Notify (see page 2884)	Responds when items are added, removed, or extracted from the list.
	Put (see page 2887)	This is Put, a member of class TList.
	SetCapacity (see page 2891)	This is SetCapacity, a member of class TList.
	SetCount (see page 2892)	This is SetCount, a member of class TList.

Public Methods

	Name	Description
	Add (see page 19850)	Adds a new attachment to the end of the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 19850)	Raises an exception.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 19851)	Empties the list and frees the TSOAPAttachment objects.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2874)	Removes the item at the position given by the Index parameter.
V	Destroy (see page 2870)	Destroys an instance of TList (see page 2868).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Error (see page 2875)	This is the overview for the Error method overload.
	Exchange (see page 2876)	Swaps the position of two items in the Items (see page 2898) array.
	Expand (see page 2877)	Increases the Capacity (see page 2895) of the list.
	Extract (see page 2877)	Removes a specified item from the list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	First (see page 2878)	Returns Items (see page 2898)[0].
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2878)	Returns a TList (see page 2868) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	IndexOf (see page 2879)	Returns the index of the first entry in the Items (see page 2898) array with a specified value.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 19851)	Raises an exception.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Last (see page 2882)	Returns Items (see page 2898)[Count (see page 2896) - 1].
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	Move (see page 2884)	Changes the position of an item in the Items (see page 2898) array.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Pack (see page 2885)	Deletes all nil (Delphi) or NULL (C++) items from the Items (see page 2898) array.
	Remove (see page 19852)	Raises an exception.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Sort (see page 2892)	Performs a QuickSort on the list based on the comparison function Compare.

Public Properties

	Name	Description
	Capacity (see page 2895)	Specifies the allocated size of the array of pointers maintained by the TList (see page 2868) object.
	Count (see page 2896)	Indicates the number of entries in the list that are in use.
	Items (see page 2898)	Lists the object references.
R	List (see page 2900)	Specifies the array of pointers that make up the Items (see page 2898) array.

137.1.1.2 TSoapDataList Methods

The methods of the TSoapDataList class are listed here.

Public Methods

	Name	Description
	Add (see page 19850)	Adds a new attachment to the end of the list.
	Assign (see page 19850)	Raises an exception.
V	Clear (see page 19851)	Empties the list and frees the TSOAPAttachment objects.
	Insert (see page 19851)	Raises an exception.
	Remove (see page 19852)	Raises an exception.

137.1.1.2.1 TSoapDataList.Add Method

Adds a new attachment to the end of the list.

Pascal

```
function Add(Item: Pointer): Integer; reintroduce;
```

C++

```
__fastcall int Add(void * Item);
```

Description

Do not call the Add method. This method is used internally to add attachments to the data list when the application first reads an incoming message that includes attachments or when marshaling a message to be sent that includes attachments. Applications should not change the list of attachments, or they will interfere with the marshaling and unmarshaling of attachment data.

Item is a pointer to the TSoapAttachment instance that is added to the list.

Add returns the index of the new attachment.

See Also

Capacity (see page 2895)

Count (see page 2896)

Insert (see page 19851)

Items (see page 2898)

Clear (see page 19851)

137.1.1.2.2 TSoapDataList.Assign Method

Raises an exception.

Pascal

```
procedure Assign(ListA: TList; AOperator: TListAssignOp = laCopy; ListB: TList = Nil);
reintroduce;
```

C++

```
__fastcall Assign(TList ListA, TListAssignOp AOperator = laCopy, TList ListB = Nil);
```

Description

Do not call the Assign method. TSoapDataList (see page 19848) replaces the inherited method to raise an exception if you call this method because applications should not change the items in a TSoapDataList (see page 19848) object.

See Also

Add (see page 19850)

137.1.1.2.3 TSoapDataList.Clear Method

Empties the list and frees the TSOAPAttachment objects.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Clear removes all of the objects from the list and frees each one. Do not call this method. It is used internally when marshaling and unmarshaling messages that contain attachments. Applications should not change the items in a TSoapDataList (see page 19848) object.

See Also

Delete (see page 2874)

Items (see page 2898)

Remove (see page 2887)

137.1.1.2.4 TSoapDataList.Insert Method

Raises an exception.

Pascal

```
procedure Insert(Index: Integer; aClass: TClass); reintroduce;
```

C++

```
__fastcall Insert(int Index, TClass aClass);
```

Description

Do not call the Insert method. TSoapDataList (see page 19848) replaces the inherited method to raise an exception if you call this method because applications should not change the items in a TSoapDataList (see page 19848) object.

See Also

Add (see page 19850)

137.1.1.2.5 TSoapDataList.Remove Method

Raises an exception.

Pascal

```
function Remove(aClass: TClass): Integer; reintroduce;
```

C++

```
__fastcall int Remove(TClass aClass);
```

Description

Do not call the Remove method. TSoapDataList (see page 19848) replaces the inherited method to raise an exception if you call this method because applications should not change the items in a TSoapDataList (see page 19848) object.

See Also

Clear (see page 19851)

137.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IMimeAttachmentHandler (see page 19853)	IMimeAttachmentHandler is the interface for interpreting or manipulating the multipart form that represents a Web Service call that includes an attachment.

137.2.1 IMimeAttachmentHandler Interface

IMimeAttachmentHandler is the interface for interpreting or manipulating the multipart form that represents a Web Service call that includes an attachment.

Class Hierarchy

SOAPAttachIntf.IMimeAttachmentHandler

Pascal

```
IMimeAttachmentHandler = interface;
```

C++

```
__interface IMimeAttachmentHandler;
```

File

SOAPAttachIntf

Description

IMimeAttachmentHandler defines methods for working with multipart forms that represent SOAP requests which include attachements. It is used internally to process the headers of the SOAP envelope part of multipart forms, to generate events when the application sends or receives a multipart form that represents a Web Service call which includes attachments, and to parse or assemble such a multipart form.

Note: C++ method declarations that use IMimeAttachmentHandler use the _di_IMimeAttachmentHandler type instead. This type is a DelphiInterface wrapper around the IMimeAttachmentHandler interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IMimeAttachmentHandler >
_di_IMimeAttachmentHandler;
```

See Also

TSoapAttachment

137.2.1.1 IMimeAttachmentHandler Members

The following tables list the members exposed by IMimeAttachmentHandler.

Public Events

	Name	Description
	OnGetAttachment (see page 19860)	Occurs for each attachment as it is read.
	OnSendAttachment (see page 19860)	Occurs for each attachment that is added to a multipart form to be sent.

Public Methods

	Name	Description
	AddSoapHeader (see page 19854)	Adds a specified header to the SOAP envelope portion of the emerging .
	CreateMimeStream (see page 19855)	Creates and initializes a stream that will eventually contain an assembled multipart form.
	FinalizeStream (see page 19856)	Performs any necessary final steps in creating the emerging multipart form.
	GetMIMEBoundary (see page 19856)	Returns the value of the MIMEBoundary (see page 19859) property.
	GetMIMEStream (see page 19856)	Returns a stream from which to read the assembled multipart form.
	GetOnGetAttachmentEvent (see page 19857)	Returns the OnGetAttachment (see page 19860) event handler.
	GetOnSendAttachmentEvent (see page 19857)	Returns the OnSendAttachment (see page 19860) event handler.
	ProcessMultiPartForm (see page 19857)	Parses a multipart form into the SOAP envelope and a list of attachments.
	SetMIMEBoundary (see page 19858)	Sets the value of the MIMEBoundary (see page 19859) property.
	SetOnGetAttachmentEvent (see page 19858)	Assigns the OnGetAttachment (see page 19860) event handler.
	SetOnSendAttachmentEvent (see page 19859)	Assigns the OnSendAttachment (see page 19860) event handler.

Public Properties

	Name	Description
	MIMEBoundary (see page 19859)	Indicates the MIME boundary string that signals the boundaries between the parts of the multipart form.

137.2.1.2 IMimeAttachmentHandler Methods

The methods of the IMimeAttachmentHandler class are listed here.

Public Methods

	Name	Description
	AddSoapHeader (see page 19854)	Adds a specified header to the SOAP envelope portion of the emerging .
	CreateMimeStream (see page 19855)	Creates and initializes a stream that will eventually contain an assembled multipart form.
	FinalizeStream (see page 19856)	Performs any necessary final steps in creating the emerging multipart form.
	GetMIMEBoundary (see page 19856)	Returns the value of the MIMEBoundary (see page 19859) property.
	GetMIMEStream (see page 19856)	Returns a stream from which to read the assembled multipart form.
	GetOnGetAttachmentEvent (see page 19857)	Returns the OnGetAttachment (see page 19860) event handler.
	GetOnSendAttachmentEvent (see page 19857)	Returns the OnSendAttachment (see page 19860) event handler.
	ProcessMultiPartForm (see page 19857)	Parses a multipart form into the SOAP envelope and a list of attachments.
	SetMIMEBoundary (see page 19858)	Sets the value of the MIMEBoundary (see page 19859) property.
	SetOnGetAttachmentEvent (see page 19858)	Assigns the OnGetAttachment (see page 19860) event handler.
	SetOnSendAttachmentEvent (see page 19859)	Assigns the OnSendAttachment (see page 19860) event handler.

137.2.1.2.1 IMimeAttachmentHandler.AddSoapHeader Method

Adds a specified header to the SOAP envelope portion of the emerging .

Pascal

```
procedure AddSoapHeader(Value: string);
```

C++

```
__fastcall AddSoapHeader(AnsiString Value);
```

Description

Call AddSoapHeader to add an arbitrary header to the SOAP envelope portion of an emerging multipart form. This method should not be called until after a call to the CreateMimeStream (see page 19855) method initializes the emerging multipart form.

Value is the text of the header to add to the SOAP envelope.

Typically, this method is called to add the ContentType header to the SOAP envelope when its value is not known at the point when the application calls CreateMimeStream (see page 19855).

Note: After all calls to AddSoapHeader have been made to add headers to the SOAP envelope, the application must call the FinalizeStream (see page 19856) method to allow the MIME attachment handler to finish generating the multipart form. Only after the call to FinalizeStream (see page 19856) can the application read the content of the multipart form by calling the GetMIMEStream (see page 19856) method.

See Also

CreateMimeStream (see page 19855)

FinalizeStream (see page 19856)

GetMIMEStream (see page 19856)

137.2.1.2.2 IMimeAttachmentHandler.CreateMimeStream Method

Creates and initializes a stream that will eventually contain an assembled multipart form.

Pascal

```
procedure CreateMimeStream(Envelope: TStream; Attachments: TSoapDataList);
```

C++

```
__fastcall CreateMimeStream(TStream Envelope, TSoapDataList Attachments);
```

Description

Call CreateMimeStream as the first step when generating a multipart form to represent a SOAP packet and its associated attachments.

Envelope is a stream from which CreateMimeStream can read the SOAP envelope portion of the multipart form. This must be a complete encoding with the exception of headers, which can be added by subsequent calls to the AddSoapHeader (see page 19854) method.

Attachments is a list of TSoapAttachment objects that represent the attachments that should be included in the multipart form.

CreateMimeStream creates a new stream object and initializes it with the data from Envelope and Attachments. This new stream can then be completed by calling the AddSoapHeader (see page 19854) method to add any headers to the SOAP envelope portion and then calling the FinalizeStream (see page 19856) method. After calling FinalizeStream (see page 19856), the application can access the generated stream by calling the GetMIMEStream (see page 19856) method.

See Also

FinalizeStream (see page 19856)

AddSoapHeader (see page 19854)

GetMIMEStream (see page 19856)

137.2.1.2.3 IMimeAttachmentHandler.FinalizeStream Method

Performs any necessary final steps in creating the emerging multipart form.

Pascal

```
procedure FinalizeStream;
```

C++

```
__fastcall FinalizeStream();
```

Description

Call FinalizeStream as a final step when assembling a multipart form to represent a SOAP packet and its associated attachments. When an application assembles a multipart form to represent a SOAP packet and its associated attachments, it takes the following steps:

1. Call CreateMimeStream (see page 19855) to create the stream object that includes data from the SOAP envelope and the content of any attachments.
2. Call AddSoapHeader (see page 19854) to add any headers to the SOAP envelope portion of the stream.
3. Call FinalizeStream to indicate that all changes are complete and allow the MIME attachment handler to make any final changes.

Only after a call to FinalizeStream can the application read the assembled multipart form by calling GetMIMEStream (see page 19856).

See Also

CreateMimeStream (see page 19855)

AddSoapHeader (see page 19854)

GetMIMEStream (see page 19856)

137.2.1.2.4 IMimeAttachmentHandler.GetMIMEBoundary Method

Returns the value of the MIMEBoundary (see page 19859) property.

Pascal

```
function GetMIMEBoundary: String;
```

C++

```
__fastcall AnsiString GetMIMEBoundary();
```

Description

GetMIMEBoundary is the read implementation for the MIMEBoundary (see page 19859) property.

See Also

MIMEBoundary (see page 19859)

137.2.1.2.5 IMimeAttachmentHandler.GetMIMEStream Method

Returns a stream from which to read the assembled multipart form.

Pascal

```
function GetMIMEStream(Release: Boolean = False): TStream;
```

C++

```
__fastcall TStream GetMIMEStream(Boolean Release = False);
```

Description

Call GetMIMEStream after a call to the FinalizeStream (see page 19856) method to access the stream from which to read a newly-assembled multipart form. This stream should not be used until after the multipart form is fully assembled.

Release indicates whether the caller is responsible for freeing the stream object. When Release is true, the caller takes ownership of the stream and is responsible for freeing it. When Release is false, the caller should not free the stream that is returned, because the object that implements IMimeAttachmentHandler (see page 19853) will free it.

See Also

CreateMimeStream (see page 19855)

AddSoapHeader (see page 19854)

FinalizeStream (see page 19856)

137.2.1.2.6 IMimeAttachmentHandler.GetOnGetAttachmentEvent Method

Returns the OnGetAttachment (see page 19860) event handler.

Pascal

```
function GetOnGetAttachmentEvent: TOnGetAttachmentEvent;
```

C++

```
__fastcall TOnGetAttachmentEvent GetOnGetAttachmentEvent();
```

Description

GetOnGetAttachmentEvent is the read implementation for the OnGetAttachment (see page 19860) event.

See Also

OnGetAttachment (see page 19860)

137.2.1.2.7 IMimeAttachmentHandler.GetOnSendAttachmentEvent Method

Returns the OnSendAttachment (see page 19860) event handler.

Pascal

```
function GetOnSendAttachmentEvent: TOnSendAttachmentEvent;
```

C++

```
__fastcall TOnSendAttachmentEvent GetOnSendAttachmentEvent();
```

Description

GetOnSendAttachmentEvent is the read implementation for the OnSendAttachment (see page 19860) event handler.

See Also

OnSendAttachment (see page 19860)

137.2.1.2.8 IMimeAttachmentHandler.ProcessMultiPartForm Method

Parses a multipart form into the SOAP envelope and a list of attachments.

Pascal

```
procedure ProcessMultiPartForm(const ASourceStream: TStream; const ADestStream: TStream;
AMimeBoundary: string; SoapHeaders: TStrings; Attachments: TSoapDataList; TempDir: string);
```

C++

```
__fastcall ProcessMultiPartForm(const TStream ASourceStream, const TStream ADestStream,
AnsiString AMimeBoundary, TStrings SoapHeaders, TSoapDataList Attachments, AnsiString TempDir);
```

Description

Call ProcessMultiPartForm to interpret a multipart form that is received by the client or server in a Web Service application. This method separates the SOAP envelope portion of the multipart form from the portions that represent attachments, and parses the headers of the SOAP envelope into a convenient string list.

ASourceStream is a stream from which ProcessMultiPartForm can read the incoming multipart form. This is the content of the incoming message.

ADestStream is a stream to which ProcessMultiPartForm writes the SOAP envelope portion of ASourceStream.

AMimeBoundary is the mime boundary string that identifies the boundaries between the parts of the multipart form. Typically, it is obtained from the headers of the incoming HTTP message whose content is the content of ASourceStream.

SoapHeaders is an existing TStrings descendant to which ProcessMultiPartForm writes the headers of the SOAP envelope portion of the multipart form. Each header is added as a string of the form HeaderName=Value.

Attachments is an empty TSoapDataList (see page 19848) object to which ProcessMultiPartForm adds the TSoapAttachment objects that represent the attachments in the multipart form.

TempDir is the name of the directory where ProcessMultiPartForm writes the temporary files it generates for each of the attachments.

137.2.1.2.9 IMimeAttachmentHandler.SetMIMEBoundary Method

Sets the value of the MIMEBoundary (see page 19859) property.

Pascal

```
procedure SetMIMEBoundary(const MimeBndry: String);
```

C++

```
__fastcall SetMIMEBoundary(const AnsiString MimeBndry);
```

Description

SetMIMEBoundary is the write implementation for the MIMEBoundary (see page 19859) property.

MimeBndry is the string that should identify the boundary between parts of the multipart form.

See Also

MIMEBoundary (see page 19859)

137.2.1.2.10 IMimeAttachmentHandler.SetOnGetAttachmentEvent Method

Assigns the OnGetAttachment (see page 19860) event handler.

Pascal

```
procedure SetOnGetAttachmentEvent(OnGetAttachment: TOnGetAttachmentEvent);
```

C++

```
__fastcall SetOnGetAttachmentEvent(TOnGetAttachmentEvent OnGetAttachment);
```

Description

SetOnGetAttachmentEvent is the write implementation for the OnGetAttachment (see page 19860) event.

OnGetAttachment (see page 19860) is the event handler that should to respond to the event.

See Also

OnGetAttachment (see page 19860)

137.2.1.2.11 IMimeAttachmentHandler.SetOnSendAttachmentEvent Method

Assigns the OnSendAttachment (see page 19860) event handler.

Pascal

```
procedure SetOnSendAttachmentEvent(OnSendAttachment: TOnSendAttachmentEvent);
```

C++

```
__fastcall SetOnSendAttachmentEvent(TOnSendAttachmentEvent OnSendAttachment);
```

Description

SetOnSendAttachmentEvent is the write implementation for the OnSendAttachment (see page 19860) event.

OnSendAttachment (see page 19860) is the event handler that should to respond to the event.

See Also

OnSendAttachment (see page 19860)

137.2.1.3 IMimeAttachmentHandler Properties

The properties of the IMimeAttachmentHandler class are listed here.

Public Properties

	Name	Description
	MIMEBoundary (see page 19859)	Indicates the MIME boundary string that signals the boundaries between the parts of the multipart form.

137.2.1.3.1 IMimeAttachmentHandler.MIMEBoundary Property

Indicates the MIME boundary string that signals the boundaries between the parts of the multipart form.

Pascal

```
property MIMEBoundary: string;
```

C++

```
__property AnsiString MIMEBoundary;
```

Description

MIMEBoundary is the string that identifies the boundary between the parts of the multipart form. It is an attribute of the ContentType header for the entire form.

When the MIME attachment handler is processing a received form, this property is not available until after the application has called the ProcessMultiPartForm (see page 19857) method. When the MIME attachment handler is assembling a multipart

form in order to send a message that includes attachments, the handler generates this string for the new form, and the CreateMimeStream (see page 19855) method uses it when assembling the form.

See Also

ProcessMultiPartForm (see page 19857)

CreateMimeStream (see page 19855)

137.2.1.4 IMimeAttachmentHandler Events

The events of the IMimeAttachmentHandler class are listed here.

Public Events

	Name	Description
	OnGetAttachment (see page 19860)	Occurs for each attachment as it is read.
	OnSendAttachment (see page 19860)	Occurs for each attachment that is added to a multipart form to be sent.

137.2.1.4.1 IMimeAttachmentHandler.OnGetAttachment Event

Occurs for each attachment as it is read.

Pascal

```
property OnGetAttachment: TOnGetAttachmentEvent;
```

C++

```
__property TOnGetAttachmentEvent OnGetAttachment;
```

Description

The MIME attachment handler generates this event as it reads each incoming attachment. Classes (see page 2685) that use the MIME attachment handler, such as TRio or TSoapPascalInvoker, assign their own event handlers as the value of this event.

See Also

OnSendAttachment (see page 19860)

OnGetAttachment (see page 18731)

OnGetAttachment (see page 20030)

137.2.1.4.2 IMimeAttachmentHandler.OnSendAttachment Event

Occurs for each attachment that is added to a multipart form to be sent.

Pascal

```
property OnSendAttachment: TOnSendAttachmentEvent;
```

C++

```
__property TOnSendAttachmentEvent OnSendAttachment;
```

Description

The MIME attachment handler generates this event as it adds each outgoing attachment to the multipart form. Classes (see page 2685) that use the MIME attachment handler, such as TRio or TSoapPascalInvoker, assign their own event handlers as the value of this event.

See Also

OnGetAttachment (see page 19860)

OnSendAttachment (see page 18731)

OnSendAttachment (see page 20030)

137.3 Types

The following table lists types in this documentation.

Types

Name	Description
TOnGetAttachmentEvent (see page 19862)	This is type SOAPAttachIntf.TOnGetAttachmentEvent.
TOnSendAttachmentEvent (see page 19862)	This is type SOAPAttachIntf.TOnSendAttachmentEvent.

137.3.1 SOAPAttachIntf.TOnGetAttachmentEvent Type

Pascal

```
TOnGetAttachmentEvent = procedure (AttachmentStream: TStream; Attachment: TSOAPAttachment) of
object;
```

C++

```
(AttachmentStream: TStream; Attachment: TSOAPAttachment) ( TOnGetAttachmentEvent)();
```

File

SOAPAttachIntf

Description

This is type SOAPAttachIntf.TOnGetAttachmentEvent.

137.3.2 SOAPAttachIntf.TOnSendAttachmentEvent Type

Pascal

```
TOnSendAttachmentEvent = procedure (AttachmentStream: TStream; Attachment: TSOAPAttachment) of
object;
```

C++

```
(AttachmentStream: TStream; Attachment: TSOAPAttachment) ( TOnSendAttachmentEvent)();
```

File

SOAPAttachIntf

Description

This is type SOAPAttachIntf.TOnSendAttachmentEvent.

138 SOAPConn Namespace

This is namespace SOAPConn.

Topics

Name	Description
Classes (see page 19864)	The following table lists classes in this documentation.

Classes

	Name	Description
	TSoapConnection (see page 19864)	TSoapConnection connects to an application server that is implemented as a Web Service.

138.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TSoapConnection (see page 19864)	TSoapConnection connects to an application server that is implemented as a Web Service.

138.1.1 TSoapConnection Class

TSoapConnection connects to an application server that is implemented as a Web Service.

Class Hierarchy

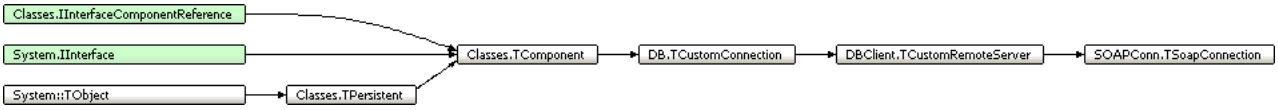

Pascal

```
TSoapConnection = class(TCustomRemoteServer);
```

C++

```
class TSoapConnection : public TCustomRemoteServer;
```

File

SOAPConn

Description

TSoapConnection is used in the client portion of a multi-tiered database application to establish and maintain the connection between the client and a remote application server that is implemented as a Web Service. The SOAP connection object can

Establish an initial connection to a remote application server.

Obtain an interface for the application server.

Obtain a list of providers on the application server.

Drop the connection to the remote application server.

TSoapConnection establishes the initial connection between the client application and a remote application server using SOAP.

Note: To use TSoapConnection on Windows, you must have wininet.dll installed on the client system. wininet.dll is found in the Windows system directory if you have IE3 or higher installed.

TSoapConnection uses an internal THTTPRio object to obtain an interface from the Web Service application. This interface is either IAppServer or IAppServerSOAP (or one of their descendants), depending on the value of the UseSOAPAdapter (see page 19879) property. The recommended approach is to use the IAppServerSOAP interface, which is better adapted to SOAP as a transport protocol. However, some application servers (those created using Delphi 6 or Kylix 2) only support an IAppServer interface. If you are using an application server that does not support IAppServerSOAP, set UseSOAPAdapter (see page 19879) to false.

Regardless of which interface TSoapConnection uses to communicate with the application server, it makes the information available to client datasets in the client application by providing an IAppServer interface. Client datasets use the IAppServer interface from the SOAP connection component to communicate with providers on the application server, or to otherwise call the application server's data module interface.

Note: Unlike when using other DataSnap connection components, you can't use the AppServer property of TSoapConnection to call methods of the application server's interface that are not IAppServer methods. Instead, to communicate with a SOAP data module on the application interface, use a separate THTTPRIO object or specify the interface using the SOAPServerIID (see page 19877) property and then call the GetSOAPServer (see page 19872) method

. To connect to remote application servers using DCOM, use the TDCOMConnection component instead. To connect to remote application servers using TCP/IP, use the TSocketConnection component instead. To connect to remote application servers using HTTP (but not SOAP), use TWebConnection.

See Also

TDataSetProvider (see page 18508)

TClientDataSet (see page 8289)

TDCOMConnection (see page 16008)

TRemoteDataModule (see page 6585)

IAppServer (see page 16450)

TSocketConnection (see page 18928)

TWebConnection (see page 18968)

THTTPRio

IAppServerSoap

Extending the Interface of the Application Server

Specifying a Connection Using SOAP

138.1.1.1 TSoapConnection Members

The following tables list the members exposed by TSoapConnection.

Protected Events

	Name	Description
	OnGetUsername (see page 8533)	This is OnGetUsername, a member of class TCustomRemoteServer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoConnect (see page 19870)	Provides the interface for a method that opens a connection.
	DoDisconnect (see page 19870)	Provides the interface for a method that terminates the connection.
	GetAppServer (see page 8530)	Returns the value of the AppServer (see page 8532) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetConnected (see page 19871)	Returns the value of the Connected property.
	GetDataSet (see page 6865)	Returns a dataset, given its index in the DataSets (see page 6870) property array.
	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSOAPServerIID (see page 19873)	This is GetSOAPServerIID, a member of class TSoapConnection.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6866)	Initializes the connection component after all objects in the form or data module have been loaded.
	Notification (see page 19873)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RegisterClient (see page 6866)	Registers an object that uses the connection component.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppServer (see page 8531)	Sets the AppServer (see page 8532) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnected (see page 6867)	Changes the value of the Connected (see page 6868) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetSOAPServerIID (see page 19874)	This is SetSOAPServerIID, a member of class TSoapConnection.
	SetUseSoapAdapter (see page 19874)	This is SetUseSoapAdapter, a member of class TSoapConnection.
	UnRegisterClient (see page 6868)	Removes a registered client so that it is no longer associated with the connection component.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 6863)	Closes the connection.
V	Create (see page 19869)	Creates and initializes an instance of TSoapConnection (see page 19864).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 19869)	Destroys an instance of TSoapConnection (see page 19864).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetProviderNames (see page 19871)	Executes a callback for all provider objects on the application server.
V	GetServer (see page 19871)	Returns an IAppServer interface for communicating with the application server.
V	GetServerList (see page 19872)	Returns a list of application servers (remote data modules) to which the component can connect.
	GetSOAPServer (see page 19872)	Returns the IAppServerSOAP interface of the application server.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 6866)	Opens the connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AppServer (see page 8532)	Contains the interface for communicating with the application server.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 6870)	Lists the datasets currently using this connection component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	LoginPrompt (see page 6872)	Specifies whether a login dialog appears immediately before opening a new connection.
	OnLogin (see page 6873)	Occurs when a communication channel opens to the server.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	RIO (see page 19876)	Provides access to the internal remote interfaced object that communicates with the application server.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterConnect (see page 19880)	Occurs after a connection is established.
	AfterDisconnect (see page 19880)	Occurs after the connection closes.
	BeforeConnect (see page 19880)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 19881)	Occurs immediately before the connection closes.
	OnAfterExecute (see page 19881)	Occurs when TSoapConnection (see page 19864) receives the results from executing a method on the application server.
	OnBeforeExecute (see page 19882)	Occurs before TSoapConnection (see page 19864) sends a method call to the application server for execution.

Published Properties

	Name	Description
	Agent (see page 19874)	Provides the value of the User-Agent header for HTTP messages to the server.

	Connected (see page 19875)	Indicates whether the SOAP connection component is connected to the application server.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Password (see page 19875)	Specifies the password that allows the client application to log on to a local host or proxy server.
	Proxy (see page 19876)	Lists any proxy servers that are required to make the connection.
	ProxyByPass (see page 19876)	Lists the host names of locally registered servers that do not require a proxy.
	SOAPServerIID (see page 19877)	Identifies the invokable interface that TSoapConnection (see page 19864) uses to communicate with the application server.
	Tag (see page 2808)	Stores an integer value as part of a component.
	URL (see page 19878)	Specifies the Uniform Resource Locator for the THTTPSoapDispatcher on the application server.
	UserName (see page 19878)	Specifies the user name that allows the client application to log on to a local host or proxy server.
	UseSOAPAdapter (see page 19879)	Specifies how the SOAP connection component communicates with the application server.

138.1.1.2 TSoapConnection.Create Constructor

Creates and initializes an instance of TSoapConnection (see page 19864).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSoapConnection(TComponent * AOwner);
```

Description

Call Create to instantiate a TSoapConnection (see page 19864) object at runtime. SOAP connection components added to forms or data modules at design time are created automatically.

The AOwner parameter specifies the component (usually a form or data module) that is responsible for freeing the SOAP connection component.

See Also

Owner (see page 2807)

138.1.1.3 TSoapConnection.Destroy Destructor

Destroys an instance of TSoapConnection (see page 19864).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSoapConnection();
```

Description

Do not call Destroy directly in an application. Instead, use the Free method, which checks that the SOAP connection is not nil before calling Destroy.

138.1.1.4 TSoapConnection Methods

The methods of the TSoapConnection class are listed here.

Protected Methods

	Name	Description
	DoConnect (see page 19870)	Provides the interface for a method that opens a connection.
	DoDisconnect (see page 19870)	Provides the interface for a method that terminates the connection.
	GetConnected (see page 19871)	Returns the value of the Connected property.
	GetSOAPServerIID (see page 19873)	This is GetSOAPServerIID, a member of class TSoapConnection.
	Notification (see page 19873)	Forwards notification messages to all owned components.
	SetSOAPServerIID (see page 19874)	This is SetSOAPServerIID, a member of class TSoapConnection.
	SetUseSoapAdapter (see page 19874)	This is SetUseSoapAdapter, a member of class TSoapConnection.

Public Methods

	Name	Description
	GetProviderNames (see page 19871)	Executes a callback for all provider objects on the application server.
	GetServer (see page 19871)	Returns an IAppServer interface for communicating with the application server.
	GetServerList (see page 19872)	Returns a list of application servers (remote data modules) to which the component can connect.
	GetSOAPServer (see page 19872)	Returns the IAppServerSOAP interface of the application server.

138.1.1.4.1 TSoapConnection.DoConnect Method

Provides the interface for a method that opens a connection.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

The Connected property uses DoConnect to establish a connection. Descendant classes override the DoConnect method to establish their connection as appropriate. As implemented in TCustomConnection, DoConnect does nothing.

See Also

Connected (see page 6868)

DoDisconnect (see page 6864)

138.1.1.4.2 TSoapConnection.DoDisconnect Method

Provides the interface for a method that terminates the connection.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

The Connected property uses DoDisconnect to close a connection. Descendant classes override the DoDisconnect method to

drop a connection. As implemented in TCustomConnection, DoDisconnect does nothing.

See Also

Connected (see page 6868)

DoConnect (see page 6864)

138.1.1.4.3 TSoapConnection.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected read implementation of the Connected property. Descendant classes override this method to return true if a connection is open and false otherwise. As defined in TCustomConnection, GetConnected always returns false.

See Also

Connected (see page 6868)

SetConnected (see page 6867)

138.1.1.4.4 TSoapConnection.GetProviderNames Method

Executes a callback for all provider objects on the application server.

Pascal

```
procedure GetProviderNames(Proc: TGetStrProc); override;
```

C++

```
virtual __fastcall GetProviderNames(TGetStrProc Proc);
```

Description

Call GetProviderNames to execute the procedure specified by the Proc parameter for every provider exposed by the application server. For every provider listed by the remote data module's AS_GetProviderNames method, Proc is called with the name of the provider.

See Also

AS_GetProviderNames (see page 16454)

AppServer (see page 8532)

ProviderName (see page 8492)

138.1.1.4.5 TSoapConnection.GetServer Method

Returns an IAppServer interface for communicating with the application server.

Pascal

```
function GetServer: IAppServer; override;
```

C++

```
virtual __fastcall IAppServer GetServer();
```

Description

Client datasets call GetServer to set the value of their AppServer property. This allows them to access providers on the remote application server.

GetServer ensures that there is a connection and then returns an IAppServer interface that can be used to communicate with providers on the application server. To use other methods of the application server's interface, use a separate THTTPRIO object or call the GetSOAPServer (see page 19872) method after setting the SOAPServerIID (see page 19877) property.

See Also

Connected (see page 19875)

AppServer (see page 8462)

THTTPRIO (see page 19928)

UseSOAPAdapter (see page 19879)

SOAPServerIID (see page 19877)

138.1.1.4.6 TSoapConnection.GetServerList Method

Returns a list of application servers (remote data modules) to which the component can connect.

Pascal

```
function GetServerList: OleVariant; override;
```

C++

```
virtual __fastcall OleVariant GetServerList();
```

Description

Connection components use GetServerList internally to fill the drop down list in the Object Inspector for the ServerName property.

As implemented in TCustomRemoteServer, GetServerList always returns NULL. Descendant classes override this method to return a list of application servers that are on the local machine and registered in the system registry (TDispatchConnection), or fetched from the server machine (TStreamedConnection).

See Also

ServerName (see page 16030)

138.1.1.4.7 TSoapConnection.GetSOAPServer Method

Returns the IAppServerSOAP interface of the application server.

Pascal

```
function GetSOAPServer: IAppServerSOAP;
```

C++

```
__fastcall IAppServerSOAP GetSOAPServer();
```

Description

Call GetSOAPServer to access the IAppServerSOAP interface that the SOAP connection component uses when

UseSOAPAdapter (see page 19879) is true. The internal adapter uses this interface to implement the IAppServer interface that the GetServer (see page 19871) method returns.

Tip: If you have set the SOAPServerIID (see page 19877) property, you can use the as operator (Delphi) or the Supports method (C++) to convert this interface into the application server's interface.

See Also

Connected (see page 19875)

GetServer (see page 19871)

UseSOAPAdapter (see page 19879)

THTTPRio

SOAPServerIID (see page 19877)

138.1.1.4.8 TSoapConnection.GetSOAPServerIID Method

Pascal

```
function GetSOAPServerIID: TGUID;
```

C++

```
__fastcall TGUID GetSOAPServerIID();
```

Description

This is GetSOAPServerIID, a member of class TSoapConnection.

138.1.1.4.9 TSoapConnection.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

138.1.1.4.10 TSoapConnection.SetSOAPServerIID Method

Pascal

```
procedure SetSOAPServerIID(const IID: String);
```

C++

```
__fastcall SetSOAPServerIID(const AnsiString IID);
```

Description

This is SetSOAPServerIID, a member of class TSoapConnection.

138.1.1.4.11 TSoapConnection.SetUseSoapAdapter Method

Pascal

```
procedure SetUseSoapAdapter(Value: Boolean);
```

C++

```
__fastcall SetUseSoapAdapter(Boolean Value);
```

Description

This is SetUseSoapAdapter, a member of class TSoapConnection.

138.1.1.5 TSoapConnection Properties

The properties of the TSoapConnection class are listed here.

Public Properties

	Name	Description
R	RIO (see page 19876)	Provides access to the internal remote interfaced object that communicates with the application server.

Published Properties

	Name	Description
	Agent (see page 19874)	Provides the value of the User-Agent header for HTTP messages to the server.
	Connected (see page 19875)	Indicates whether the SOAP connection component is connected to the application server.
	Password (see page 19875)	Specifies the password that allows the client application to log on to a local host or proxy server.
	Proxy (see page 19876)	Lists any proxy servers that are required to make the connection.
	ProxyByPass (see page 19876)	Lists the host names of locally registered servers that do not require a proxy.
	SOAPServerIID (see page 19877)	Identifies the invokable interface that TSoapConnection (see page 19864) uses to communicate with the application server.
	URL (see page 19878)	Specifies the Uniform Resource Locator for the THTTPSoapDispatcher on the application server.
	UserName (see page 19878)	Specifies the user name that allows the client application to log on to a local host or proxy server.
	UseSOAPAdapter (see page 19879)	Specifies how the SOAP connection component communicates with the application server.

138.1.1.5.1 TSoapConnection.Agent Property

Provides the value of the User-Agent header for HTTP messages to the server.

Pascal

```
property Agent: string;
```

C++

```
__property AnsiString Agent;
```

Description

Use Agent to supply HTTP messages generated by the SOAP connection component with the name of your application. This name is included as a User-Agent header. The User-Agent header is an optional string that describes the application generating the HTTP message.

See Also

Title (see page 13723)

138.1.1.5.2 TSoapConnection.Connected Property

Indicates whether the SOAP connection component is connected to the application server.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Set Connected to true to establish a connection to the application server. Set Connected to false to terminate the connection.

When changing the value of Connected, TSoapConnection (see page 19864) generates a BeforeConnect (see page 19880) and an AfterConnect (see page 19880) event, or a BeforeDisconnect (see page 19881) and an AfterDisconnect (see page 19880) event.

See Also

BeforeConnect (see page 6875)

AfterConnect (see page 6874)

BeforeDisconnect (see page 6875)

AfterDisconnect (see page 6875)

138.1.1.5.3 TSoapConnection.Password Property

Specifies the password that allows the client application to log on to a local host or proxy server.

Pascal

```
property Password: string;
```

C++

```
__property AnsiString Password;
```

Description

When the SOAP connection component connects to the application server, it supplies the values of the UserName (see page 19878) and Password properties so that it can log on to the local host or proxy. The value of Password can be an empty string if neither the local host nor proxy requires authentication.

Note: On Linux, if the proxy does not require authentication, you can use the UserName (see page 19878) and Password properties to log on to the Web Server.

See Also

UserName (see page 19878)

Specifying a Connection Using SOAP

138.1.1.5.4 TSoapConnection.Proxy Property

Lists any proxy servers that are required to make the connection.

Pascal

```
property Proxy: string;
```

C++

```
__property AnsiString Proxy;
```

Description

Use Proxy if the SOAP connection component can't resolve the host portion of URL (see page 19878) locally. If the server is registered locally, Proxy can be a null string. Otherwise, Proxy lists the host names of any proxy servers that can forward request messages on to the intended server machine.

See Also

ProxyByPass (see page 19876)

Specifying a Connection Using SOAP

138.1.1.5.5 TSoapConnection.ProxyByPass Property

Lists the host names of locally registered servers that do not require a proxy.

Pascal

```
property ProxyByPass: string;
```

C++

```
__property AnsiString ProxyByPass;
```

Description

Use ProxyByPass to indicate any server systems to which the SOAP connection component can make a direct connection. When the URL (see page 19878) property identifies a server in this list, the Web connection component makes a direct connection even if the Proxy (see page 19876) property is specified.

When ProxyByPass is an empty string, the Web connection component checks the registry for any locally known servers and bypasses the proxy server if the host portion of URL (see page 19878) refers to a registered local server.

See Also

Proxy (see page 19876)

URL (see page 19878)

138.1.1.5.6 TSoapConnection.RIO Property

Provides access to the internal remote interfaced object that communicates with the application server.

Pascal

```
property RIO: THTTPRIO;
```

C++

```
__property THTTPRIO RIO;
```

Description

TSoapConnection (see page 19864) uses an internal remote interfaced object (THTTPRIO) to communicate with the application server. This internal object obtains a reference to an invokable interface that is specified by the SOAPServerIID (see page 19877) property.

Use RIO to access the internal THTTPRIO instance.

See Also

SOAPServerIID (see page 19877)

138.1.1.5.7 TSoapConnection.SOAPServerIID Property

Identifies the invokable interface that TSoapConnection (see page 19864) uses to communicate with the application server.

Pascal

```
property SOAPServerIID: String;
```

C++

```
__property AnsiString SOAPServerIID;
```

Description

Set SOAPServerIID to identify the interface of the application server. SOAPServerIID is the GUID of the application server's interface. This interface is a descendant of IAppServerSOAP. Once you have formed a connection, you can access this interface by first calling the GetSOAPServer (see page 19872) method and then using the global Supports function or as operator (Delphi) or the Supports method of the interface (C++).

Warning: Use the GUID for the interface that is generated by the WSDL importer when you import the application server's definitions. This may differ from the GUID that the server uses in its implementation.

Note: In Delphi, at runtime, you can use the interface name in code and the compiler automatically converts that to the GUID. At design time (in the Object Inspector) or in C++ code, you must specify the GUID itself, not the interface name.

Note: SOAPServerIID is only used if the UseSOAPAdapter (see page 19879) property is true. In addition, if you set SOAPServerIID to anything other than IAppServerSOAP, you must import the definition of the application server's interface so that it is registered with the invocation registry.

See Also

URL (see page 19878)

IAppServerSOAP (see page 20012)

UseSOAPAdapter (see page 19879)

GetSOAPServer (see page 19872)

Specifying a Connection Using SOAP

138.1.1.5.8 TSoapConnection.URL Property

Specifies the Uniform Resource Locator for the THTTPSoapDispatcher on the application server.

Pascal

```
property URL: string;
```

C++

```
__property AnsiString URL;
```

Description

Use URL to specify the Uniform Resource Locator that the SOAP connection component uses to locate the application server.

URL includes the protocol (http or https), host name, and scriptname for the application server, as well as a path portion that identifies the THTTPSoapDispatcher on the application server. Typically, it has the form

```
http://DataHost.org/scripts/AppServer.dll/SOAP/
http://DataHost.org/scripts/AppServer.so/SOAP/
```

Note: By default, URL does not include the interface name. The name of the interface is assumed to be IAppServerSOAP (or IAppServer). If the server is written in the Delphi language and the target remote data module supports an IAppServerSOAP descendant, Delphi clients can optionally include the target interface name at the end of the URL. This allows clients to identify a SOAP data module when the server contains multiple SOAP data modules, or allows you to access the server's interface by calling the GetSOAPServer (see page 19872) method. If the client or server is not written using Delphi, you can accomplish this same effect using the SOAPServerIID (see page 19877) property.

Note: URL is a standard Uniform Resource Locator. As such, you can also use it to override the port associated with http (80) or https (443) if the Web server is configured differently. The port number follows the host name, preceded by a colon.

See Also

THTTPSoapDispatcher (see page 24628)

SOAPServerIID (see page 19877)

GetSOAPServer (see page 19872)

Specifying a Connection Using SOAP

138.1.1.5.9 TSoapConnection.UserName Property

Specifies the user name that allows the client application to log on to a local host or proxy server.

Pascal

```
property UserName: string;
```

C++

```
__property AnsiString UserName;
```

Description

When the SOAP connection component connects to the application server on a local host or to a proxy server, it supplies the values of the UserName and Password (see page 19875) properties so that it can log in. If neither the host nor the proxy server requires authentication, UserName can be an empty string.

Note: On Linux, if the proxy does not require authentication, you can use the UserName and Password (see page 19875) properties to log on to the Web Server.

See Also

Password (see page 19875)

Proxy (see page 19876)

Specifying a Connection Using SOAP

138.1.1.5.10 TSoapConnection.UseSOAPAdapter Property

Specifies how the SOAP connection component communicates with the application server.

Pascal

```
property UseSOAPAdapter: Boolean;
```

C++

```
__property Boolean UseSOAPAdapter;
```

Description

When UseSOAPAdapter is true, TSOAPConnection communicates with the application server using an IAppServerSOAP interface. It then uses an internal adapter which implements the IAppServer interface for client datasets. In this case, the GetServer (see page 19871) method returns the IAppServer interface that the internal adapter implements.

When UseSOAPAdpater is false, TSOAPConnection uses an IAppServer interface to communicate with the application server and exposes that interface with the GetServer (see page 19871) method. In this case, the GetSOAPServer (see page 19872) method returns nil (Delphi) or NULL (C++).

The recommended setting for UseSOAPAdapter is true. Unlike IAppServer, the IAppServerSOAP interface does not use the safecall calling convention. This makes it easier to marshal interface calls when the client or server is written in C++. IAppServerSOAP is not supported, however, when the application server is built using Delphi 6 (before update patch 2) or Kylix 2. If you are connecting to a server that does not support IAppServerSOAP, set UseSOAPAdapter to false.

See Also

IAppServer (see page 16450)

IAppServerSOAP (see page 20012)

GetServer (see page 19871)

GetSOAPServer (see page 19872)

Specifying a Connection Using SOAP

138.1.1.6 TSoapConnection Events

The events of the TSoapConnection class are listed here.

Published Events

	Name	Description
	AfterConnect (see page 19880)	Occurs after a connection is established.
	AfterDisconnect (see page 19880)	Occurs after the connection closes.
	BeforeConnect (see page 19880)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 19881)	Occurs immediately before the connection closes.
	OnAfterExecute (see page 19881)	Occurs when TSoapConnection (see page 19864) receives the results from executing a method on the application server.

	OnBeforeExecute (see page 19882)	Occurs before TSoapConnection (see page 19864) sends a method call to the application server for execution.

138.1.1.6.1 TSoapConnection.AfterConnect Event

Occurs after a connection is established.

Pascal

```
property AfterConnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterConnect;
```

Description

Write an AfterConnect event handler to take application-specific actions immediately after the connection component opens a connection to the remote source of database information.

See Also

BeforeConnect (see page 6875)

Connected (see page 6868)

Connecting to a Database Server

138.1.1.6.2 TSoapConnection.AfterDisconnect Event

Occurs after the connection closes.

Pascal

```
property AfterDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterDisconnect;
```

Description

Write an AfterDisconnect event handler to take application-specific actions after the connection component drops a connection.

See Also

BeforeDisconnect (see page 6875)

Connected (see page 6868)

Disconnecting from a Database Server

138.1.1.6.3 TSoapConnection.BeforeConnect Event

Occurs immediately before establishing a connection.

Pascal

```
property BeforeConnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeConnect;
```

Description

Write a BeforeConnect event handler to take application-specific actions before the connection component opens a connection

to the remote source of database information.

See Also

AfterConnect (see page 6874)

Connected (see page 6868)

Connecting to a Database Server

138.1.1.6.4 TSoapConnection.BeforeDisconnect Event

Occurs immediately before the connection closes.

Pascal

```
property BeforeDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeDisconnect;
```

Description

Write a BeforeDisconnect event handler to take application-specific actions before dropping a connection.

Delphi Examples:

```
{
This BeforeDisconnect event handler makes sure that the
client dataset has applied all its pending updates before
terminating the connection to the server.
}
procedure TForm1.RemoteServer1BeforeDisconnect(Sender: TObject);
begin
  if (ClientDataSet1.ChangeCount > 0) then
    ClientDataSet1.ApplyUpdates(-1);
end;
```

See Also

AfterDisconnect (see page 6875)

Connected (see page 6868)

Disconnecting from a Database Server

138.1.1.6.5 TSoapConnection.OnAfterExecute Event

Occurs when TSoapConnection (see page 19864) receives the results from executing a method on the application server.

Pascal

```
property OnAfterExecute: TAfterExecuteEvent;
```

C++

```
__property TAfterExecuteEvent OnAfterExecute;
```

Description

Write an OnAfterExecute event handler to examine or alter the encoded results from executing a method on the application server. OnAfterExecute is assigned as the OnAfterExecute event handler of an internal remote interfaced object that the SOAP connection uses to communicate with the application server.

See Also

OnBeforeExecute (see page 19882)

OnAfterExecute (see page 18730)

GetServer (see page 19871)

138.1.1.6.6 TSoapConnection.OnBeforeExecute Event

Occurs before TSoapConnection (see page 19864) sends a method call to the application server for execution.

Pascal

```
property OnBeforeExecute: TBeforeExecuteEvent;
```

C++

```
__property TBeforeExecuteEvent OnBeforeExecute;
```

Description

Write an OnBeforeExecute event handler to examine or alter an encoded method call before it is sent to the server. OnBeforeExecute occurs after the call has been marshaled into a SOAP request packet, but before that SOAP request packet has been sent to the application server.

OnBeforeExecute is assigned as the OnBeforeExecute event handler of an internal remote interfaced object that the SOAP connection uses to communicate with the application server.

See Also

OnAfterExecute (see page 19881)

OnBeforeExecute (see page 18730)

GetServer (see page 19871)

139 SOAPDm Namespace

This is namespace SOAPDm.

Topics

Name	Description
Classes (see page 19884)	The following table lists classes in this documentation.

Classes

	Name	Description
	TSoapDataModule (see page 19884)	TSoapDataModule encapsulates the objects and interfaces of an application server in a SOAP-based multi-tiered database application.

139.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TSoapDataModule (see page 19884)	TSoapDataModule encapsulates the objects and interfaces of an application server in a SOAP-based multi-tiered database application.

139.1.1 TSoapDataModule Class

TSoapDataModule encapsulates the objects and interfaces of an application server in a SOAP-based multi-tiered database application.

Class Hierarchy

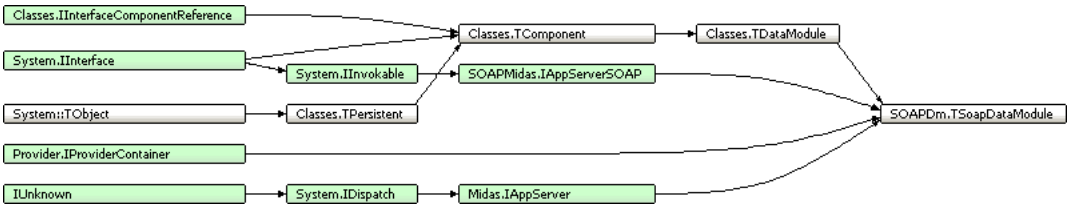

Pascal

```
TSoapDataModule = class(TDataModule, IAppServer, IAppServerSOAP, IProviderContainer);
```

C++

```
class TSoapDataModule : public TDataModule, public IAppServer, public IAppServerSOAP, public
IProviderContainer;
```

File

SOAPDm

Description

Use a TSoapDataModule object as the central repository for all objects in a SOAP-based application server. SOAP data modules can contain any nonvisual components. Typically these are data access components, such as datasets and the provider components that handle communication between these datasets and client applications. They can also contain other nonvisual components, such as TTimer, or TImageList.

At design time a TSoapDataModule object provides a visual container into which a developer can place nonvisual components, set their properties, and write event handlers for them. In the unit file for the SOAP data module a developer can also code any business rules that are to be centralized in the middle tier of a multi-tiered application.

TSoapDataModule implements the IAppServer and IAppServerSOAP interfaces. These are both invokable interfaces, so if you add TSoapDataModule to a Web Service application, the invoker automatically forwards all IAppServer or IAppServerSOAP calls to the SOAP data module. The client application uses a TSoapConnection component to generate these calls.

To create a SOAP data module at design time,

1. Choose File|New|Other, and select Web Services application from the WebServices page of the new items dialog. This generates a Web module that contains a dispatcher and an invoker component. These components work together to interpret and execute incoming SOAP requests.
2. Choose File|New|Other, and select SOAP DataModule from the Web Services page of the new items dialog. This adds a TSoapDataModule descendant to your application. The TSoapDataModule descendant supports an invokable interface that descends from IAppServerSOAP as well as an IAppServer interface.

   Note: Web Service applications should not include more than one TSoapDataModule descendant. When the invoker component that appears in the primary Web module receives a request for the IAppServer or IAppServerSOAP interface, it can't distinguish between multiple implementation classes.

See Also

IAppServer (see page 16450)

TSoapConnection (see page 19864)

IAppServerSOAP (see page 20012)

139.1.1.1 TSoapDataModule Members

The following tables list the members exposed by TSoapDataModule.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	ApplyUpdates (see page 19891)	Applies updates received from a client using a specified provider.
	AS_ApplyUpdates (see page 19892)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 19893)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 19893)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 19894)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 19895)	Returns a list of all the available providers on the SOAP data module.
	AS_GetRecords (see page 19895)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 19897)	Returns information from a specified record of the provider's dataset.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DataRequest (see page 19898)	Generates an OnDataRequest event on the specified provider component.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
V	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
	Execute (see page 19898)	Executes the query or stored procedure bound to a specified provider.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetParams (see page 19899)	Fetches current parameter values from the dataset bound to a specified provider.
V	GetProvider (see page 19899)	Returns a registered provider component given its name.
	GetProviderNames (see page 19900)	Returns a list of all the available providers on the SOAP data module.
	GetRecords (see page 19900)	Returns a data packet that contains the specified data.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.
	RowRequest (see page 19902)	Returns information from a specified record of the provider's dataset.
V	SAS_ApplyUpdates (see page 19903)	This is SAS_ApplyUpdates, a member of class TSoapDataModule.
V	SAS_DataRequest (see page 19903)	Generates an OnSAS_DataRequest event on the specified provider component.
V	SAS_Execute (see page 19904)	SAS_Executes the query or stored procedure bound to a specified provider.
V	SAS_GetParams (see page 19905)	Fetches current parameter values from the dataset bound to a specified provider.
V	SAS_GetProviderNames (see page 19906)	Returns a list of all the available providers on the SOAP data module.
V	SAS_GetRecords (see page 19906)	This is SAS_GetRecords, a member of class TSoapDataModule.
V	SAS_RowRequest (see page 19907)	This is SAS_RowRequest, a member of class TSoapDataModule.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 19890)	Increments the reference count for this data module's interface.
	_Release (see page 19890)	Decrements the reference count for this data module's interface.
V	AfterConstruction (see page 19891)	Generates an OnCreate event.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 19897)	Generates an OnDestroy event.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 19888)	Creates a SOAP data module instance.
V	CreateNew (see page 2828)	Creates an instance of a data module and registers with the global screen object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 19889)	Frees a SOAP data module instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	Lock (see page 19901)	Locks the SOAP data module so that it can't be locked by other threads.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 19901)	Allocates memory for each instance of TSoapDataModule (see page 19884).
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterProvider (see page 19902)	This is RegisterProvider, a member of class TSoapDataModule.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 19903)	Handles exceptions in methods declared using the safecall calling convention.

	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Unlock (see page 19907)	Unlocks the SOAP data module after it was locked by a call to the Lock (see page 19901) method.
	UnRegisterProvider (see page 19907)	Removes a provider from the list returned by AS_GetProviderNames (see page 19895).
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	ProviderCount (see page 19908)	Indicates the number of providers registered with the SOAP data module.
R	Providers (see page 19908)	Lists the registered provider components that are contained in the SOAP data module.
R	RefCount (see page 19909)	Reference counts the data module's interface.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.
	Tag (see page 2808)	Stores an integer value as part of a component.

139.1.1.2 TSoapDataModule.Create Constructor

Creates a SOAP data module instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSoapDataModule(TComponent * AOwner);
```

Description

Most applications do not need to call the constructor for TSoapDataModule (see page 19884). Instead, add a SOAP data module to a Web service application using the SOAP data module wizard on the SOAP page of the new items dialog.

AOwner specifies the component that is responsible for freeing the SOAP data module (typically the application). If you supply an owner, the SOAP data module acts as a component, and is freed by that owner. If you pass nil for the AOwner parameter, the

SOAP data module acts like an interfaced object, and maintains a reference count for every use of its interface. When that reference count goes to zero, the SOAP data module automatically frees itself. Note that is you set AOwner to nil but never obtain a reference to this object's interface, you must explicitly free the data module.

See Also

Create (see page 2827)

139.1.1.3 TSoapDataModule.Destroy Destructor

Frees a SOAP data module instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSoapDataModule();
```

Description

Most applications do not need to call the destructor for TSoapDataModule (see page 19884). The SOAP data module is freed by its owner when the application shuts down.

If you must destroy a SOAP data module at runtime, call the Free method instead.

See Also

Free (see page 22300)

139.1.1.4 TSoapDataModule Methods

The methods of the TSoapDataModule class are listed here.

Protected Methods

	Name	Description
	ApplyUpdates (see page 19891)	Applies updates received from a client using a specified provider.
	AS_ApplyUpdates (see page 19892)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 19893)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 19893)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 19894)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 19895)	Returns a list of all the available providers on the SOAP data module.
	AS_GetRecords (see page 19895)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 19897)	Returns information from a specified record of the provider's dataset.
	DataRequest (see page 19898)	Generates an OnDataRequest event on the specified provider component.
	Execute (see page 19898)	Executes the query or stored procedure bound to a specified provider.
	GetParams (see page 19899)	Fetches current parameter values from the dataset bound to a specified provider.
V	GetProvider (see page 19899)	Returns a registered provider component given its name.
	GetProviderNames (see page 19900)	Returns a list of all the available providers on the SOAP data module.
	GetRecords (see page 19900)	Returns a data packet that contains the specified data.
	RowRequest (see page 19902)	Returns information from a specified record of the provider's dataset.
V	SAS_ApplyUpdates (see page 19903)	This is SAS_ApplyUpdates, a member of class TSoapDataModule.
V	SAS_DataRequest (see page 19903)	Generates an OnSAS_DataRequest event on the specified provider component.
V	SAS_Execute (see page 19904)	SAS_Executes the query or stored procedure bound to a specified provider.
V	SAS_GetParams (see page 19905)	Fetches current parameter values from the dataset bound to a specified provider.
V	SAS_GetProviderNames (see page 19906)	Returns a list of all the available providers on the SOAP data module.

	SAS_GetRecords (see page 19906)	This is SAS_GetRecords, a member of class TSoapDataModule.
	SAS_RowRequest (see page 19907)	This is SAS_RowRequest, a member of class TSoapDataModule.

Public Methods

	Name	Description
	_AddRef (see page 19890)	Increments the reference count for this data module's interface.
	_Release (see page 19890)	Decrements the reference count for this data module's interface.
	AfterConstruction (see page 19891)	Generates an OnCreate event.
	BeforeDestruction (see page 19897)	Generates an OnDestroy event.
	Lock (see page 19901)	Locks the SOAP data module so that it can't be locked by other threads.
	NewInstance (see page 19901)	Allocates memory for each instance of TSoapDataModule (see page 19884).
	RegisterProvider (see page 19902)	This is RegisterProvider, a member of class TSoapDataModule.
	SafeCallException (see page 19903)	Handles exceptions in methods declared using the safecall calling convention.
	Unlock (see page 19907)	Unlocks the SOAP data module after it was locked by a call to the Lock (see page 19901) method.
	UnRegisterProvider (see page 19907)	Removes a provider from the list returned by AS_GetProviderNames (see page 19895).

139.1.1.4.1 TSoapDataModule._AddRef Method

Increments the reference count for this data module's interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

Do not call the __AddRef method directly. This method is called when an application obtains a reference to the SOAP data module's interface pointer. __AddRef increments the RefCount (see page 19909) property.

When the caller is through with the interface, it calls _Release (see page 19890). This allows the data module to free itself when it is no longer used (when the reference count goes to zero).

See Also

_Release (see page 19890)

RefCount (see page 19909)

IAppServer (see page 16450)

139.1.1.4.2 TSoapDataModule._Release Method

Decrements the reference count for this data module's interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

Do not call the __Release method directly. This method is called when an application releases a reference to the SOAP data module's interface pointer. __Release decrements the RefCount (see page 19909) property.

If the SOAP data module does not have an Owner, that is, if the value of the Owner property is nil (Delphi) or NULL (C++), then if the call to __Release changes the value of RefCount (see page 19909) to zero, the SOAP data module automatically frees itself.

See Also

_AddRef (see page 19890)

RefCount (see page 19909)

IAppServer (see page 16450)

Owner (see page 2807)

139.1.1.4.3 TSoapDataModule.AfterConstruction Method

Generates an OnCreate event.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called after the data module's constructor has finished. Do not call it explicitly in applications.

TDataModule overrides AfterConstruction to generate an OnCreate event when OldCreateOrder is false.

See Also

BeforeDestruction (see page 2830)

OnCreate (see page 2833)

OldCreateOrder (see page 2833)

139.1.1.4.4 TSoapDataModule.ApplyUpdates Method

Applies updates received from a client using a specified provider.

Pascal

```
function ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors: Integer;
out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant ApplyUpdates(const BSTR ProviderName, OleVariant Delta, int MaxErrors,
int ErrorCount, OleVariant OwnerData);
```

Description

ApplyUpdates provides the underlying implementation for the AS_ApplyUpates method (on the IAppServer interface) and the SAS_ApplyUpdates (see page 19903) method (on the IAppServerSoap interface). It writes client updates to a dataset or database server.

ProviderName specifies the name of the provider component that manages the update operation.

Delta is a Variant containing the updated, inserted, and deleted records to write to the database.

MaxErrors indicates the maximum number of errors to permit before rolling back the update operation

ErrorCount returns the actual number of errors encountered during the update operation.

OwnerData is information supplied by the client application in a BeforeApplyUpdates event handler and returned to the client application in an AfterApplyUpdates event handler. It is passed to the provider's BeforeApplyUpdates event handler and returned from the provider's AfterApplyUpdates event handler.

ApplyUpdates returns a Variant that is a data packet containing all records that could not be applied.

See Also

as_applyupdates (see page 19892)

sas_applyupdates (see page 19903)

139.1.1.4.5 TSoapDataModule.AS_ApplyUpdates Method

Applies updates received from a client dataset using a specified provider.

Pascal

```
function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_ApplyUpdates(const BSTR ProviderName, OleVariant Delta, int
MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

AS_AS_ApplyUpdates writes client updates to a dataset or database server.

ProviderName specifies the name of the provider component that manages the update operation.

Delta is a Variant containing the updated, inserted, and deleted records to write to the database.

MaxErrors indicates the maximum number of errors to permit before stopping the update operation.

ErrorCount returns the actual number of errors encountered during the update operation.

OwnerData is information supplied by the client application in a BeforeAS_ApplyUpdates event handler and returned to the client application in an AfterAS_ApplyUpdates event handler.

AS_AS_ApplyUpdates (Delphi) or AS_ApplyUpdates_result parameter (C++) returns a Variant that is a data packet containing all records that could not be applied to the database.

See Also

ApplyUpdates (see page 8394)

applyupdates (see page 18459)

BeforeApplyUpdates (see page 18479)

AfterApplyUpdates (see page 18475)

BeforeApplyUpdates (see page 8511)

AfterApplyUpdates (see page 8497)

AS_ApplyUpdates (see page 6590)

as_applyupdates

Communicating with the Client Dataset

139.1.1.4.6 TSoapDataModule.AS_DataRequest Method

Generates an OnAS_DataRequest event on the specified provider component.

Pascal

```
function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

Use AS_AS_DataRequest to invoke an OnAS_DataRequest event handler of the provider named by ProviderName. This method is provided to allow application developers to customize the communication between a client application and a provider component. There is no predefined meaning for the Data parameter or the return value (Delphi) or AS_DataRequest_result parameter (C++).

Note: Applications can only call the protected AS_AS_DataRequest method using the IAppServer interface.

See Also

OnDataRequest (see page 18482)

Providers (see page 19908)

DataRequest (see page 8404)

IAppServer (see page 16450)

139.1.1.4.7 TSoapDataModule.AS_Execute Method

AS_Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); safecall;
```

C++

```
__fastcall AS_Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant Params,
OleVariant OwnerData);
```

Description

Use AS_AS_Execute to pass an execute command to a remote provider, using a specified set of parameters. AS_AS_Execute calls the AS_Execute method of the specified provider after assigning any parameters. Output parameters that result from executing a query or stored procedure are returned in the Params parameter.

Note: AS_AS_Execute should only be called for a provider whose dataset represents a query or stored procedure that does not return a cursor.

The ProviderName parameter specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a stored procedure that replaces the provider's underlying stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

The Params parameter encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams method.

The OwnerData parameter contains custom information that appears as an argument to the provider's BeforeAS_Execute and AfterAS_Execute event handlers. This information originates in a client dataset's BeforeAS_Execute event handler. The value returned in OwnerData is passed to the client dataset's AfterAS_Execute event handler.

Note: Applications can only call the protected AS_AS_Execute method using the IAppServer interface. To execute a dataset in code from the application server, use the dataset component directly.

See Also

DataSet (see page 18529)

beforeexecute (see page 18479)

AfterExecute (see page 18476)

Params (see page 8489)

Execute (see page 8413)

Execute (see page 18466)

Providers (see page 19908)

IAppServer (see page 16450)

BeforeExecute (see page 8514)

AfterExecute (see page 8504)

PackageParams (see page 8534)

Options (see page 18440)

139.1.1.4.8 TSoapDataModule.AS_GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

Use AS_AS_GetParams to retrieve the current parameter values from the dataset associated with a provider. The ProviderName parameter specifies the name of the provider component whose dataset has the parameters. The OwnerData represents custom information that originates in a client dataset's BeforeAS_GetParams event handler and returns information that is passed to the client dataset's AfterAS_GetParams event handler.

Note: In Delphi, AS_AS_GetParams returns the parameters, encoded as a Variant array, or, if the provider does not support parameter fetching, a Null Variant.

Note: In C++, the AS_GetParams_result parameter the Variant array containing the parameters (or the Null Variant).

Use the UnpackParams procedure to convert parameter values returned by AS_AS_GetParams to a TParams object.

Warning: It is not a good idea to call AS_AS_GetParams on a SOAP data module. SOAP data modules are stateless, and any parameter values you fetch may be changed by other applications. The recommended way to obtain output parameter values is to call the AS_Execute (see page 19893) method instead.

Note: Applications can only call the protected AS_AS_GetParams method using the IAppServer interface.

See Also

DataSet (see page 18529)

Params (see page 8489)

FetchParams (see page 8415)

GetParams (see page 18467)

BeforeGetParams (see page 8514)

AfterGetParams (see page 8505)

Providers (see page 19908)

IAppServer (see page 16450)

AS_Execute (see page 19893)

UnpackParams (see page 8534)

139.1.1.4.9 TSoapDataModule.AS_GetProviderNames Method

Returns a list of all the available providers on the SOAP data module.

Pascal

```
function AS_GetProviderNames: OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetProviderNames();
```

Description

Call AS_AS_GetProviderNames for a list of all registered providers available from a SOAP data module. These values indicate the possible values for the ProviderName property of a client dataset that connects to the SOAP data module. When calling any of the SOAP data modules through the IAppServer interface, pass one of these names to indicate the provider that is the target of the method call.

AS_AS_GetProviderNames returns the names of the providers in a Variant array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

Note: Applications can only call the protected AS_AS_GetProviderNames method using the IAppServer interface.

See Also

ProviderName (see page 8492)

Providers (see page 19908)

IAppServer (see page 16450)

Exported (see page 18474)

139.1.1.4.10 TSoapDataModule.AS_GetRecords Method

Returns a data packet that contains the specified data.

Pascal

```
function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetRecords(const BSTR ProviderName, int Count, int RecsOut, int
Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

Call AS_AS_GetRecords to retrieve a specified number of records. AS_AS_GetRecords returns the requested records, starting with the current record of the provider's dataset. Because SOAP data modules are stateless, you may need to reposition the cursor or re-execute a query or stored procedure in the provider's BeforeAS_GetRecords event handler.

ProviderName gives the name of the provider component that provides the records.

Count indicates the number or type of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only Count records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants. For example, the following value requests metadata along with a data packet encoded in XML:

```
MetaDataOption + XMLOption
MetaDataOption || XMLOption
```

CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a table or stored procedure that replaces the provider's underlying database table or stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

Params is a Variant containing any parameters that should be passed to the provider's dataset before it executes to generate the requested data. It returns any output parameters.

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_GetRecords event handler. This information is passed to the provider's BeforeAS_GetRecords event handler. OwnerData returns information supplied by the provider's AfterAS_GetRecords event handler.

Records are returned as a data packet in a Variant (as the return value in Delphi or the AS_GetRecords_result parameter in C++).

Note: Applications can only call the protected AS_AS_GetRecords method using the IAppServer interface. To fetch records directly in code from the application server, use the provider component's AS_GetRecords method.

See Also

Data (see page 8467)

Params (see page 8489)

GetNextPacket (see page 8423)

GetRecords (see page 18468)

BeforeGetRecords (see page 18480)

AfterGetRecords (see page 18477)

Providers (see page 19908)

IAppServer (see page 16450)

139.1.1.4.11 TSoapDataModule.AS_RowRequest Method

Returns information from a specified record of the provider's dataset.

Pascal

```
function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_RowRequest(const BSTR ProviderName, OleVariant Row, int RequestType,
OleVariant OwnerData);
```

Description

Client datasets call AS_AS_RowRequest automatically to implement their FetchBlobs, FetchDetails , or RefreshRecord method.

The ProviderName parameter indicates the provider associated with the dataset from which information should be fetched.

The Row parameter is an OleVariant that describes the current record on the client dataset.

The RequestType parameter indicates the type of information required. It is an integer version of the TFetchOptions type. (An integer because the value may be marshaled to a remote application server). To create a value for RequestType, take the corresponding TFetchOptions value, cast it to a Byte, and cast the result to an Integer:

```
Integer(Byte([foBlobs, foDetails]));
(int) (Byte) (TFetchOptions() << foBlobs << foDetails);
```

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_RowRequest event handler. This information is passed to the provider's BeforeAS_RowRequest event handler. OwnerData returns custom information supplied by the provider's AfterAS_RowRequest event handler.

The requested data is returned as a delta packet.

Note: Applications can only call the protected AS_AS_RowRequest method using the IAppServer interface.

See Also

RowRequest (see page 18472)

BeforeRowRequest (see page 18482)

AfterRowRequest (see page 18478)

FetchBlobs (see page 8414)

FetchDetails (see page 8414)

RefreshRecord (see page 8442)

Providers (see page 19908)

IAppServer (see page 16450)

139.1.1.4.12 TSoapDataModule.BeforeDestruction Method

Generates an OnDestroy event.

Pascal

```
procedure BeforeDestruction; override;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

BeforeDestruction is called immediately before the data module's destructor. Do not call it explicitly in applications.

TDataModule overrides BeforeDestruction to generate an OnDestroy event when OldCreateOrder is false.

See Also

AfterConstruction (see page 2829)

OnDestroy (see page 2834)

OldCreateOrder (see page 2833)

139.1.1.4.13 TSoapDataModule.DataRequest Method

Generates an OnDataRequest event on the specified provider component.

Pascal

```
function DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

DataRequest provides the underlying implementation for the AS_DataRequest (see page 19893) method (on the IAppServer interface) and the SAS_DataRequest (see page 19903) method (on the IAppServerSOAP interface). This method calls any OnDataRequest event handler of the specified provider.

There is no predefined meaning for the Data parameter. It represents custom information supplied by the client and passed to the OnDataRequest event handler.

DataRequest returns the value that is assigned to the OwnerData parameter of the event handler on exit.

See Also

AS_DataRequest (see page 19893)

SAS_DataRequest (see page 19903)

OnDataRequest (see page 18482)

139.1.1.4.14 TSoapDataModule.Execute Method

Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure Execute(const ProviderName: WideString; const CommandText: WideString; var Params:
OleVariant; var OwnerData: OleVariant);
```

C++

```
__fastcall Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant Params,
OleVariant OwnerData);
```

Description

Execute provides the underlying implementation for the AS_Execute (see page 19893) method (on the IAppServer interface) and the SAS_Execute (see page 19904) method (on the IAppServerSOAP interface). It calls the Execute method of the specified provider after assigning any parameters. Output parameters that result from executing a query or stored procedure are returned in Params.

ProviderName specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a stored procedure that replaces the provider's underlying stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

Params encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams method.

OwnerData contains custom information that appears as an argument to the provider's BeforeExecute and AfterExecute event handlers. This information originates in a client dataset's BeforeExecute event handler. The value returned in OwnerData is passed to the client dataset's AfterExecute event handler.

See Also

AS_Execute (see page 19893)

SAS_Execute (see page 19904)

139.1.1.4.15 TSoapDataModule.GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

GetParams provides the underlying implementation of the AS_GetParams (see page 19894) method (on the IAppServer interface) and the SAS_GetParams (see page 19905) method (on the IAppServerSOAP interface).

ProviderName specifies the name of the provider component whose dataset has the parameters to fetch.

OwnerData represents custom information that originates in a client dataset's BeforeGetParams event handler and returns information that is passed to the client dataset's AfterGetParams event handler.

GetParams returns the parameters, encoded as a Variant array, or, if the provider does not support parameter fetching, GetParams returns a Null Variant. Use the UnpackParams procedure to convert parameter values returned by GetParams to a TParams object.

See Also

SAS_GetParams (see page 19905)

AS_GetParams (see page 19894)

139.1.1.4.16 TSoapDataModule.GetProvider Method

Returns a registered provider component given its name.

Pascal

```
function GetProvider(const ProviderName: string): TCustomProvider; virtual;
```

C++

```
virtual __fastcall TCustomProvider GetProvider(const AnsiString ProviderName);
```

Description

GetProvider is the protected read implementation of the Providers (see page 19908) property.

ProviderName specifies the name of a provider that is returned. If ProviderName is not one of the names returned by AS_GetProviderNames (see page 19895) (for example, if the provider's Exported property is false), GetProvider raises an exception.

See Also

Providers (see page 19908)

Exported (see page 18474)

AS_GetProviderNames (see page 19895)

139.1.1.4.17 TSoapDataModule.GetProviderNames Method

Returns a list of all the available providers on the SOAP data module.

Pascal

```
function GetProviderNames: OleVariant;
```

C++

```
__fastcall OleVariant GetProviderNames();
```

Description

GetProviderNames provides the underlying implementation for the AS_GetProviderNames (see page 19895) method (on the IAppServer interface) and the SAS_GetProviderNames (see page 19906) method (on the IAppServerSOAP interface). It returns a list of all registered providers available from a SOAP data module.

GetProviderNames returns the names of the providers in a Variant array.

See Also

AS_GetProviderNames (see page 19895)

SAS_GetProviderNames (see page 19906)

139.1.1.4.18 TSoapDataModule.GetRecords Method

Returns a data packet that contains the specified data.

Pascal

```
function GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant GetRecords(const BSTR ProviderName, int Count, int RecsOut, int Options,
const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

GetRecords provides the underlying implementation for the AS_GetRecords (see page 19895) method (on the IAppServer interface) and the SAS_GetRecords (see page 19906) method (on the IAppServerSOAP interface). GetRecords returns the requested records, starting with the current record of the provider's dataset.

- ProviderName gives the name of the provider component that supplies the records.

- Count indicates the number or type of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only Count records are retrieved.
- RecsOut returns the actual number of records retrieved.
- Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants.:
- CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a table or stored procedure that replaces the provider's underlying database table or stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.
- Params is a Variant containing any parameters that should be passed to the provider's dataset before it executes to generate the requested data. It returns any output parameters.
- OwnerData contains custom information that is supplied by a client dataset's BeforeGetRecords event handler. This information is passed to the provider's BeforeGetRecords event handler. OwnerData returns information supplied by the provider's AfterGetRecords event handler.
- Records are returned as a data packet in a Variant.

See Also

AS_GetRecords (see page 19895)

sas_getrecords (see page 19906)

139.1.1.4.19 TSoapDataModule.Lock Method

Locks the SOAP data module so that it can't be locked by other threads.

Pascal

```
procedure Lock; virtual;
```

C++

```
virtual __fastcall Lock();
```

Description

SOAP data modules use Lock internally to ensure that all IAppServer calls are thread-safe. When implementing a SOAP data module descendant, call Lock before accessing any instance data, including the providers contained in the data module. When finished with the block of code that must be protected from thread conflicts, call the Unlock (see page 19907) method to release the lock.

Lock uses a critical section to guard against thread conflicts. This means that any code that accesses the SOAP data module's properties or contained objects must call Lock or it will introduce thread conflicts.

See Also

Providers (see page 19908)

Unlock (see page 19907)

IAppServer (see page 16450)

TCriticalSection (see page 22058)

139.1.1.4.20 TSoapDataModule.NewInstance Method

Allocates memory for each instance of TSoapDataModule (see page 19884).

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
virtual __fastcall TObject * NewInstance();
```

Description

All constructors call NewInstance automatically. NewInstance calls InstanceSize to determine how much memory to allocate from the heap to contain a particular instance. Do not call NewInstance directly.

TSoapDataModule (see page 19884) overrides NewInstance to increment the reference count when a new instance is created. This prevents any constructor from accidentally deleting the instance due to a reference count of zero. The AfterConstruction (see page 19891) method decrements the reference count after all constructors have executed.

See Also

NewInstance (see page 22306)

InstanceSize (see page 22306)

afterconstruction (see page 19891)

139.1.1.4.21 TSoapDataModule.RegisterProvider Method

Pascal

```
procedure RegisterProvider(Value: TCustomProvider);
```

C++

```
__fastcall RegisterProvider(TCustomProvider Value);
```

Description

This is RegisterProvider, a member of class TSoapDataModule.

139.1.1.4.22 TSoapDataModule.RowRequest Method

Returns information from a specified record of the provider's dataset.

Pascal

```
function RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer; var
OwnerData: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant RowRequest(const BSTR ProviderName, OleVariant Row, int RequestType,
OleVariant OwnerData);
```

Description

RowRequest provides the underlying implementation for the AS_RowRequest (see page 19897) method (on the IAppServer interface) and the SAS_RowRequest (see page 19907) method (on the IAppServerSOAP interface).

ProviderName indicates the provider associated with the dataset from which information should be fetched.

Row is an OleVariant that describes the current record on the client dataset.

RequestType indicates the type of information required. It is an integer version of the TFetchOptions type.

OwnerData contains custom information that is supplied by a client dataset's BeforeRowRequest event handler. This information is passed to the provider's BeforeRowRequest event handler. OwnerData returns custom information supplied by the provider's AfterRowRequest event handler.

The requested data is returned as a delta packet.

See Also

AS_RowRequest (see page 19897)

sas_rowrequest (see page 19907)

139.1.1.4.23 TSoapDataModule.SafeCallException Method

Handles exceptions in methods declared using the safecall calling convention.

Pascal

```
function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): HResult; override;
```

C++

```
virtual __fastcall HRESULT SafeCallException(TObject * ExceptObject, void * ExceptAddr);
```

Description

SafeCallException is used for handling exceptions that are raised by methods such as the AS... methods, which use the safecall calling convention. This method ensures that the SOAP fault packet includes all the necessary error information so that clients can respond appropriately to exceptions in safecall methods.

ExceptObject is the exception that was raised.

ExceptAddr is the address where the exception occurred.

SafeCallException returns an error code that indicates what error occurred.

See Also

SafeCallException (see page 2787)

139.1.1.4.24 TSoapDataModule.SAS_ApplyUpdates Method

Pascal

```
function SAS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall OleVariant SAS_ApplyUpdates(const BSTR ProviderName, OleVariant
Delta, int MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

This is SAS_ApplyUpdates, a member of class TSoapDataModule.

139.1.1.4.25 TSoapDataModule.SAS_DataRequest Method

Generates an OnSAS_DataRequest event on the specified provider component.

Pascal

```
function SAS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall OleVariant SAS_DataRequest(const BSTR ProviderName, OleVariant
Data);
```

Description

Use SAS_SAS_DataRequest to invoke an OnSAS_DataRequest event handler of the provider named by ProviderName. This method is provided to allow application developers to customize the communication between a client application and a provider component.

ProviderName is the provider on which the data module generates an OnSAS_DataRequest event.

There is no predefined meaning for the Data parameter or the return value.

Note: Applications can only call the protected SAS_SAS_DataRequest method using the IAppServerSOAP interface.

See Also

OnDataRequest (see page 18482)

Providers (see page 19908)

DataRequest (see page 8404)

IAppServerSOAP (see page 20012)

139.1.1.4.26 TSoapDataModule.SAS_Execute Method

SAS_Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure SAS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall SAS_Execute(const BSTR ProviderName, const BSTR CommandText,
OleVariant Params, OleVariant OwnerData);
```

Description

Use SAS_SAS_Execute to pass an execute command to a remote provider, using a specified set of parameters. SAS_SAS_Execute calls SAS_Execute, which calls the SAS_Execute method of the specified provider after assigning any parameters. Output parameters that result from executing a query or stored procedure are returned in the Params parameter.

Note: SAS_SAS_Execute should only be called for a provider whose dataset represents a query or stored procedure that does not return a cursor.

ProviderName specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a stored procedure that replaces the provider's underlying stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

Params encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams method.

OwnerData contains custom information that appears as an argument to the provider's BeforeSAS_Execute and AfterSAS_Execute event handlers. This information originates in a client dataset's BeforeSAS_Execute event handler. The value returned in OwnerData is passed to the client dataset's AfterSAS_Execute event handler.

Note: Applications can only call the protected SAS_SAS_Execute method using the IAppServerSOAP interface. To execute a dataset in code from the application server, use the dataset component directly.

See Also

DataSet (see page 18529)

beforeexecute (see page 18479)

AfterExecute (see page 18476)

Params (see page 8489)

Execute (see page 8413)

Execute (see page 18466)

Providers (see page 19908)

IAppServerSOAP (see page 20012)

BeforeExecute (see page 8514)

AfterExecute (see page 8504)

PackageParams (see page 8534)

Options (see page 18440)

139.1.1.4.27 TSoapDataModule.SAS_GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function SAS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall OleVariant SAS_GetParams(const BSTR ProviderName, OleVariant
OwnerData);
```

Description

Use SAS_SAS_GetParams to retrieve the current parameter values from the dataset associated with a provider.

ProviderName specifies the name of the provider component whose dataset has the parameters.

OwnerData represents custom information that originates in a client dataset's BeforeSAS_GetParams event handler and returns information that is passed to the client dataset's AfterSAS_GetParams event handler.

SAS_SAS_GetParams returns the parameters, encoded as a Variant array, or, if the provider does not support parameter fetching, SAS_SAS_GetParams returns a Null Variant. Use the UnpackParams procedure to convert parameter values returned by SAS_SAS_GetParams to a TParams object.

Warning: It is not a good idea to call SAS_SAS_GetParams on a SOAP data module. SOAP data modules are stateless, and any parameter values you fetch may be changed by other applications. The recommended way to obtain output parameter values is to call the SAS_Execute (see page 19904) method instead.

Note: Applications can only call the protected SAS_SAS_GetParams method using the IAppServerSOAP interface.

See Also

DataSet (see page 18529)

Params (see page 8489)

FetchParams (see page 8415)

GetParams (see page 18467)

BeforeGetParams (see page 8514)

AfterGetParams (see page 8505)

Providers (see page 19908)

IAppServerSOAP (see page 20012)

SAS_Execute (see page 19904)

UnpackParams (see page 8534)

139.1.1.4.28 TSoapDataModule.SAS_GetProviderNames Method

Returns a list of all the available providers on the SOAP data module.

Pascal

```
function SAS_GetProviderNames: TWideStringDynArray; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall TWideStringDynArray SAS_GetProviderNames();
```

Description

Call SAS_SAS_GetProviderNames for a list of all registered providers available from a SOAP data module. These values indicate the possible values for the ProviderName property of a client dataset that connects to the SOAP data module. When calling any of the SOAP data modules through the IAppServerSOAP interface, pass one of these names to indicate the provider that is the target of the method call.

SAS_SAS_GetProviderNames returns the names of the providers in a dynamic array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

Note: Applications can only call the protected SAS_SAS_GetProviderNames method using the IAppServerSOAP interface.

See Also

ProviderName (see page 8492)

Providers (see page 19908)

IAppServerSOAP (see page 20012)

Exported (see page 18474)

139.1.1.4.29 TSoapDataModule.SAS_GetRecords Method

Pascal

```
function SAS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall OleVariant SAS_GetRecords(const BSTR ProviderName, int Count, int
RecsOut, int Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

This is SAS_GetRecords, a member of class TSoapDataModule.

139.1.1.4.30 TSoapDataModule.SAS_RowRequest Method

Pascal

```
function SAS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall OleVariant SAS_RowRequest(const BSTR ProviderName, OleVariant
Row, int RequestType, OleVariant OwnerData);
```

Description

This is SAS_RowRequest, a member of class TSoapDataModule.

139.1.1.4.31 TSoapDataModule.Unlock Method

Unlocks the SOAP data module after it was locked by a call to the Lock (see page 19901) method.

Pascal

```
procedure Unlock; virtual;
```

C++

```
virtual __fastcall Unlock();
```

Description

Use Unlock to release the critical section that was entered by a call to the Lock (see page 19901) method. Every call to the Lock (see page 19901) method must be paired with a corresponding call to Unlock.

See Also

Lock (see page 19901)

139.1.1.4.32 TSoapDataModule.UnRegisterProvider Method

Removes a provider from the list returned by AS_GetProviderNames (see page 19895).

Pascal

```
procedure UnRegisterProvider(Value: TCustomProvider);
```

C++

```
__fastcall UnRegisterProvider(TCustomProvider Value);
```

Description

Call UnRegisterProvider to remove a provider from the SOAP data module's list of available providers. Pass a reference to the provider that should be removed as the Value parameter. When a provider is not registered, it is not returned by the AS_GetProviderNames (see page 19895) method, which can be accessed through the IAppServer interface.

To temporarily disable a provider, use the provider's Exported property. Unexported providers can't be accessed through any IAppServer method calls.

Before a registered provider is destroyed, it must be unregistered using the UnRegisterProvider method. Providers (see page 19908) that have the SOAP data module as an Owner are unregistered automatically from their destructors.

See Also

registerprovider (see page 19902)

AS_GetProviderNames (see page 19895)

IAppServer (see page 16450)

Destroy (see page 18454)

Exported (see page 18474)

139.1.1.5 TSoapDataModule Properties

The properties of the TSoapDataModule class are listed here.

Public Properties

	Name	Description
R	ProviderCount (see page 19908)	Indicates the number of providers registered with the SOAP data module.
R	Providers (see page 19908)	Lists the registered provider components that are contained in the SOAP data module.
R	RefCount (see page 19909)	Reference counts the data module's interface.

139.1.1.5.1 TSoapDataModule.ProviderCount Property

Indicates the number of providers registered with the SOAP data module.

Pascal

```
property ProviderCount: integer;
```

C++

```
__property int ProviderCount;
```

Description

ProviderCount is the number of providers that can be accessed using the Providers (see page 19908) property.

See Also

Providers (see page 19908)

registerprovider (see page 19902)

139.1.1.5.2 TSoapDataModule.Providers Property

Lists the registered provider components that are contained in the SOAP data module.

Pascal

```
property Providers [const ProviderName: string]: TCustomProvider;
```

C++

```
__property TCustomProvider Providers[AnsiString const ProviderName];
```

Description

Use Providers to access a provider component from the SOAP data module given its name. The SOAP data module uses Providers to locate the target of method calls made using the IAppServer interface.

If the value passed as is not one of the names returned by AS_GetProviderNames (see page 19895) (for example, if the provider's Exported property is false) reading Providers raises an exception.

See Also

AS_GetProviderNames (see page 19895)

Exported (see page 18474)

registerprovider (see page 19902)

ProviderCount (see page 19908)

139.1.1.5.3 TSoapDataModule.RefCount Property

Reference counts the data module's interface.

Pascal

```
property RefCount: Integer;
```

C++

```
__property int RefCount;
```

Description

RefCount keeps track of the number of times the application takes a reference to the SOAP data module's interface. When application code obtains this data module's interface, a call is generated to the _AddRef (see page 19890) method, which increments RefCount. When application code releases the interface, a call is generated to the _Release (see page 19890) method, which decrements RefCount. When RefCount drops to zero, the SOAP data module frees itself if its Owner property is not nil (Delphi) or NULL (C++).

See Also

_AddRef (see page 19890)

_Release (see page 19890)

Owner (see page 2807)

140 SOAPDomConv Namespace

This is namespace SOAPDomConv.

Topics

Name	Description
Classes (see page 19912)	The following table lists classes in this documentation.
Interfaces (see page 19921)	The following table lists interfaces in this documentation.
Structs, Records, Enums (see page 19924)	The following table lists structs, records, enums in this documentation.
Types (see page 19925)	The following table lists types in this documentation.

Classes

	Name	Description
	TSOAPDOMProcessor (see page 19912)	TSOAPDOMProcessor is the base class for components that process the SOAP headers from SOAP requests.

Interfaces

	Name	Description
	IDOMHeaderProcessor (see page 19921)	IDOMHeaderProcessor is the interface for objects that interpret and create SOAP headers.

Structs, Records, Enums

	Name	Description
	TDOMHeaderProcessorEntry (see page 19924)	This is record SOAPDomConv.TDOMHeaderProcessorEntry.

Types

Name	Description
TDOMHeaderProcessorArray (see page 19925)	This is type SOAPDomConv.TDOMHeaderProcessorArray.

140.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TSOAPDOMProcessor (see page 19912)	TSOAPDOMProcessor is the base class for components that process the SOAP headers from SOAP requests.

140.1.1 TSOAPDOMProcessor Class

TSOAPDOMProcessor is the base class for components that process the SOAP headers from SOAP requests.

Class Hierarchy

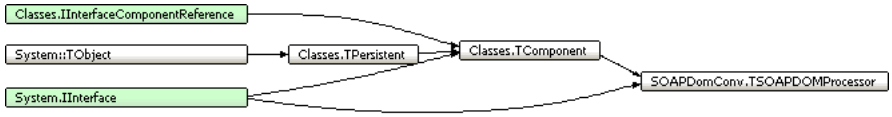

Pascal

```
TSOAPDOMProcessor = class(TComponent, IInterface);
```

C++

```
class TSOAPDOMProcessor : public TComponent, public IInterface;
```

File

SOAPDomConv

Description

TSOAPDOMProcessor is the base class for DOM-based components that interpret and create SOAP method calls. It manages a list of interfaces that it uses to create or interpret specialized headers in the SOAP message. Such headers can include application-specific information such as authentication, compression/decompression, encryption/decryption, and so on.

Do not create instances of TSOAPDOMProcessor. It is intended only as a base class for components that use the information in SOAP headers when interpreting or creating SOAP method calls.

See Also

TOPToSoapDomConvert (see page 18137)

140.1.1.1 TSOAPDOMProcessor Members

The following tables list the members exposed by TSOAPDOMProcessor.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FHeaderProcessors (see page 19915)	This is FHeaderProcessors, a member of class TSOAPDOMProcessor.

Protected Methods

	Name	Description
	_AddRef (see page 19916)	Called when an application uses a component interface.
	_Release (see page 19917)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FindHeaderProcessor (see page 19919)	Locates a processor to interpret or generate a specified SOAP header.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	MakeHeaderNodes (see page 19919)	Generates all the SOAP headers that this object knows how to create.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
V	AddHeaderProcessor (see page 19917)	Adds a processor to the internal list of interfaces to use for interpreting and creating SOAP headers.
V	AfterConstruction (see page 19918)	Responds after the last constructor executes.
V	Assign (see page 2930)	Copies the contents of another, similar object.

	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DefaultProcessHeader (see page 19918)	Handles the interpretation of a SOAP header when no processor can be found.
	Destroy (see page 19915)	Frees the memory associated with the TSOAPDOMProcessor (see page 19912) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 19919)	Allocates memory for each instance of TSOAPDOMProcessor (see page 19912).
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.

V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

140.1.1.2 TSOAPDOMProcessor.Destroy Destructor

Frees the memory associated with the TSOAPDOMProcessor (see page 19912) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSOAPDOMProcessor();
```

Description

Do not call Destroy to free an instance of TSOAPDOMProcessor (see page 19912). Instead, use the Free method, which first checks for a nil reference before calling Destroy.

See Also

Free (see page 22300)

140.1.1.3 TSOAPDOMProcessor Fields

The fields of the TSOAPDOMProcessor class are listed here.

Protected Fields

	Name	Description
	FHeaderProcessors (see page 19915)	This is FHeaderProcessors, a member of class TSOAPDOMProcessor.

140.1.1.3.1 TSOAPDOMProcessor.FHeaderProcessors Field

Pascal

```
FHeaderProcessors: TDOMHeaderProcessorArray;
```

C++

```
TDOMHeaderProcessorArray FHeaderProcessors;
```

Description

This is FHeaderProcessors, a member of class TSOAPDOMProcessor.

140.1.1.4 TSOAPDOMProcessor Methods

The methods of the TSOAPDOMProcessor class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 19916)	Called when an application uses a component interface.
	_Release (see page 19917)	Called when an application releases a component interface.
V	FindHeaderProcessor (see page 19919)	Locates a processor to interpret or generate a specified SOAP header.
	MakeHeaderNodes (see page 19919)	Generates all the SOAP headers that this object knows how to create.

Public Methods

	Name	Description
V	AddHeaderProcessor (see page 19917)	Adds a processor to the internal list of interfaces to use for interpreting and creating SOAP headers.
V	AfterConstruction (see page 19918)	Responds after the last constructor executes.
V	DefaultProcessHeader (see page 19918)	Handles the interpretation of a SOAP header when no processor can be found.
V	NewInstance (see page 19919)	Allocates memory for each instance of TSOAPDOMProcessor (see page 19912).

140.1.1.4.1 TSOAPDOMProcessor._AddRef Method

Called when an application uses a component interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef is a basic implementation of the IInterface method, _AddRef.

Note: If the component is a wrapper for a COM object, __AddRef calls the _AddRef method of that COM object, and returns the resulting reference count.

In all other cases, __AddRef simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __AddRef to implement reference counting.

See Also

Invoke (see page 2781)

QueryInterface (see page 2784)

_Release (see page 2769)

ComObject (see page 2793)

_AddRef (see page 22397)

140.1.1.4.2 TSOAPDOMProcessor._Release Method

Called when an application releases a component interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release is a basic implementation of the IInterface method, _Release.

__Release returns the resulting value of the reference count for the component's interface.

Note:

In all other cases, __Release simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __Release to implement reference counting.

See Also

Invoke (see page 2781)

QueryInterface (see page 2784)

_AddRef (see page 2769)

ComObject (see page 2793)

140.1.1.4.3 TSOAPDOMProcessor.AddHeaderProcessor Method

Adds a processor to the internal list of interfaces to use for interpreting and creating SOAP headers.

Pascal

```
procedure AddHeaderProcessor(Namespace: WideString; HeaderName: WideString; TypeName:
WideString; Processor: IDOMHeaderProcessor); virtual;
```

C++

```
virtual __fastcall AddHeaderProcessor(BSTR Namespace, BSTR HeaderName, BSTR TypeName,
IDOMHeaderProcessor Processor);
```

Description

AddHeaderProcessor stores a specified processor interface and the information necessary to identify the SOAP headers that it handles.

Namespace is the namespace URI in which the SOAP header identified by HeaderName is defined.

HeaderName is the name of the SOAP header that the processor handles.

TypeName is the name of the type that is used for the value of the header.

Processor is the interface for a helper object that creates or interprets SOAP headers that match Namespace, HeaderName, and TypeName.

Note: AddHeaderProcessor is supplied for backward compatibility. Headers are handled differently, using lists of TSOAPHeader descendants that are supplied as parameters to methods in descendant classes.

See Also

FindHeaderProcessor (see page 19919)

140.1.1.4.4 TSOAPDOMProcessor.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of TSOAPDOMProcessor (see page 19912) is allocated, its reference count is incremented to prevent its destruction while processing the constructor. AfterConstruction decrements that reference count to re-establish the correct count after all constructors have executed.

See Also

BeforeDestruction (see page 22289)

NewInstance (see page 19919)

140.1.1.4.5 TSOAPDOMProcessor.DefaultProcessHeader Method

Handles the interpretation of a SOAP header when no processor can be found.

Pascal

```
procedure DefaultProcessHeader(HeaderNode: IXMLNode; var Handled: Boolean; var AbortRequest:
Boolean); virtual;
```

C++

```
virtual __fastcall DefaultProcessHeader(IXMLNode HeaderNode, Boolean Handled, Boolean
AbortRequest);
```

Description

DefaultProcessHeader is called to provide a default interpretation of a SOAP header when it is not processed by one of the processors that FindHeaderProcessor (see page 19919) returns.

HeaderNode is the interface of the DOM representation of the SOAP header to be processed.

Handled returns true if DefaultProcessHeader handles the header node.

AbortRequest returns true if DefaultProcessHeader determines that the SOAP header can't be handled and that this results in an inability to interpret the entire SOAP request.

Note: DefaultProcessHeader is supplied for backward compatibility. Headers are handled differently, using lists of TSOAPHeader descendants that are supplied as parameters to methods in descendant classes.

See Also

FindHeaderProcessor (see page 19919)

140.1.1.4.6 TSOAPDOMProcessor.FindHeaderProcessor Method

Locates a processor to interpret or generate a specified SOAP header.

Pascal

```
function FindHeaderProcessor(Namespace: WideString; HeaderName: WideString; TypeName:
WideString): IDOMHeaderProcessor; virtual;
```

C++

```
virtual __fastcall IDOMHeaderProcessor FindHeaderProcessor(BSTR Namespace, BSTR HeaderName,
BSTR TypeName);
```

Description

FindHeaderProcessor returns the interface of the helper object that can generate or interpret a specified SOAP header. This processor must be previously added to the TSOAPDOMProcessor (see page 19912) object's internal list of processors by a call to the AddHeaderProcessor (see page 19917) method.

Namespace is the namespace URI in which the SOAP header identified by HeaderName is defined.

HeaderName is the name of the SOAP header that the processor handles.

TypeName is the name of the type that is used for the value of the header.

Note: FindHeaderProcessor is supplied for backward compatibility. Headers are handled differently, using lists of TSOAPHeader descendants that are supplied as parameters to methods in descendant classes.

See Also

AddHeaderProcessor (see page 19917)

140.1.1.4.7 TSOAPDOMProcessor.MakeHeaderNodes Method

Generates all the SOAP headers that this object knows how to create.

Pascal

```
function MakeHeaderNodes(HeaderNode: IXMLNode): IXMLNode;
```

C++

```
__fastcall IXMLNode MakeHeaderNodes(IXMLNode HeaderNode);
```

Description

MakeHeaderNodes uses the processors that have been added by the AddHeaderProcessor (see page 19917) method to generate a set of SOAP header nodes. These generated nodes are added as child nodes to the node specified by HeaderNode.

Note: MakeHeaderNodes is supplied for backward compatibility. Headers are handled differently, using lists of TSOAPHeader descendants that are supplied as parameters to methods in descendant classes.

See Also

AddHeaderProcessor (see page 19917)

140.1.1.4.8 TSOAPDOMProcessor.NewInstance Method

Allocates memory for each instance of TSOAPDOMProcessor (see page 19912).

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
virtual __fastcall TObject * NewInstance();
```

Description

All constructors call NewInstance automatically. NewInstance calls InstanceSize to determine how much memory to allocate from the heap to contain a particular instance. Do not call NewInstance directly.

TSOAPDOMProcessor (see page 19912) overrides NewInstance to increment the reference count when a new instance is created. This prevents any constructor from accidentally deleting the instance due to a reference count of zero. The AfterConstruction (see page 19918) method decrements the reference count after all constructors have executed.

See Also

NewInstance (see page 22306)

InstanceSize (see page 22306)

AfterConstruction (see page 19918)

140.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IDOMHeaderProcessor (see page 19921)	IDOMHeaderProcessor is the interface for objects that interpret and create SOAP headers.

140.2.1 IDOMHeaderProcessor Interface

IDOMHeaderProcessor is the interface for objects that interpret and create SOAP headers.

Class Hierarchy

SOAPDomConv.IDOMHeaderProcessor

Pascal

```
IDOMHeaderProcessor = interface;
```

C++

```
__interface IDOMHeaderProcessor;
```

File

SOAPDomConv

Description

IDOMHeaderProcessor provides an interface for working with specialized SOAP headers. Each object that implements IDOMHeaderProcessor handles a specific type of soap header. The TSOAPDOMProcessor (see page 19912) class calls on this interface when it needs to create or interpret the headers of a SOAP packet.

IDOMHeaderProcessor declares two methods, CreateHeader (see page 19922) and ProcessHeader (see page 19922), which create or respond to a given SOAP header, respectively.

Note: C++ method declarations that use IDOMHeaderProcessor use the _di_IDOMHeaderProcessor type instead. This type is a DelphiInterface wrapper around the IDOMHeaderProcessor interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IDOMHeaderProcessor >
_di_IDOMHeaderProcessor;
```

Note: IDOMHeaderProcessor is supplied for backward compatibility. Headers are handled differently, using the ISOAPHeaders interface.

See Also

TSoapDomProcessor

ISOAPHeaders (see page 15613)

140.2.1.1 IDOMHeaderProcessor Members

The following tables list the members exposed by IDOMHeaderProcessor.

Public Methods

	Name	Description
	CreateHeader (see page 19922)	Creates a new SOAP header node.
	ProcessHeader (see page 19922)	Interprets and responds to a SOAP header.

140.2.1.2 IDOMHeaderProcessor Methods

The methods of the IDOMHeaderProcessor class are listed here.

Public Methods

	Name	Description
	CreateHeader (see page 19922)	Creates a new SOAP header node.
	ProcessHeader (see page 19922)	Interprets and responds to a SOAP header.

140.2.1.2.1 IDOMHeaderProcessor.CreateHeader Method

Creates a new SOAP header node.

Pascal

```
function CreateHeader(HeaderNode: IXMLNode): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateHeader(IXMLNode HeaderNode);
```

Description

CreateHeader creates a new header node.

HeaderNode is the node in the emerging SOAP packet that acts as a parent to the new header.

CreateHeader returns the new header node.

See Also

MakeHeaderNodes (see page 19919)

140.2.1.2.2 IDOMHeaderProcessor.ProcessHeader Method

Interprets and responds to a SOAP header.

Pascal

```
procedure ProcessHeader(HeaderNode: IXMLNode; var Handled: Boolean; var AbortRequest: Boolean);
```

C++

```
__fastcall ProcessHeader(IXMLNode HeaderNode, Boolean Handled, Boolean AbortRequest);
```

Description

ProcessHeader provides a response to a given SOAP header node.

HeaderNode is the interface of the DOM representation of the SOAP header to be processed.

Handled returns true if ProcessHeader handles the header node.

AbortRequest returns true if ProcessHeader determines that the SOAP header can't be handled and that this results in an inability to interpret the entire SOAP request. For example, if the header node represents authentication information, ProcessHeader could set AbortRequest to true if the header supplied an invalid password.

140.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TDOMHeaderProcessorEntry (see page 19924)	This is record SOAPDomConv.TDOMHeaderProcessorEntry.

140.3.1 SOAPDomConv.TDOMHeaderProcessorEntry Record

Pascal

```
TDOMHeaderProcessorEntry = record
  Processor: IDOMHeaderProcessor;
  NameSpace: WideString;
  HeaderName: WideString;
  TypeName: WideString;
end;
```

C++

```
struct TDOMHeaderProcessorEntry {
  IDOMHeaderProcessor Processor;
  BSTR NameSpace;
  BSTR HeaderName;
  BSTR TypeName;
};
```

File

SOAPDomConv

Description

This is record SOAPDomConv.TDOMHeaderProcessorEntry.

140.4 Types

The following table lists types in this documentation.

Types

Name	Description
TDOMHeaderProcessorArray (see page 19925)	This is type SOAPDomConv.TDOMHeaderProcessorArray.

140.4.1 SOAPDomConv.TDOMHeaderProcessorArray Type

Pascal

```
TDOMHeaderProcessorArray = array of TDOMHeaderProcessorEntry;
```

C++

```
array of TDOMHeaderProcessorEntry TDOMHeaderProcessorArray;
```

File

SOAPDomConv

Description

This is type SOAPDomConv.TDOMHeaderProcessorArray.

141 SOAPHTTPClient Namespace

This is namespace SOAPHTTPClient.

Topics

Name	Description
Classes (see page 19928)	The following table lists classes in this documentation.

Classes

	Name	Description
	THTTPRIO (see page 19928)	THTTPRIO uses HTTP messages to call remote interfaced objects using SOAP.

141.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	THTTPRIO (see page 19928)	THTTPRIO uses HTTP messages to call remote interfaced objects using SOAP.

141.1.1 THTTPRIO Class

THTTPRIO uses HTTP messages to call remote interfaced objects using SOAP.

Class Hierarchy

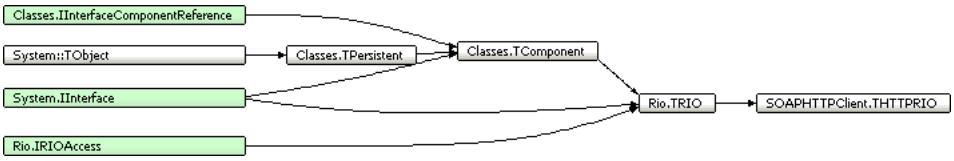

Pascal

```
THTTPRIO = class(TRIO);
```

C++

```
class THTTPRIO : public TRIO;
```

File

SOAPHTTPClient

Description

Use THTTPRIO to generate statically-linked calls to invokable interfaces on a remote Web Service (see page 19936) application. When an application casts THTTPRIO to a registered invokable interface, it dynamically generates an in-memory method table, providing an implementation to that invokable interface. THTTPRIO executes the methods in this method table by encoding the method call as a SOAP request and sending an HTTP request message to the Web Service (see page 19936) application. It unpacks the resulting HTTP response message to obtain the return value and any output parameters, or to raise an exception if the request generated an exception on the server.

Use the published properties of THTTPRIO to indicate how connect to the Web Service (see page 19936) application. There are two ways to identify where the server application is located:

You can use the URL (see page 19937) property to specify the URL (see page 19937) where the server application is located,.

If you want to look up connection information dynamically at runtime from a WSDL document, you can set the WSDLLocation (see page 19938) property. After setting WSDLLocation (see page 19938), select values for the Service (see page 19936) and Port (see page 19936) properties from drop-down lists in the object inspector to fully specify the binding you want to use.

If you need to use a proxy server, or if the server requires authentication, use the properties of the THTTPReqResp object that is the value of the HTTPWebNode (see page 19935) property to provide the necessary information.

Before THTTPRIO can generate a method table for an invokable interface, the interface must be registered with the invocation registry. To register an invokable interface, use the global InvRegistry function to access the registry, and use its RegisterInterface method.

See Also

InvRegistry

TInvokableClassRegistry (see page 15526)

Calling Invokable Interfaces

Understanding Invokable Interfaces

141.1.1.1 THTTPRIO Members

The following tables list the members exposed by THTTPRIO.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FConverter (see page 18723)	This is FConverter, a member of class TRIO.
	FIID (see page 18723)	This is FIID, a member of class TRIO.
	FWebNode (see page 18723)	This is FWebNode, a member of class TRIO.
	IntfMD (see page 18723)	This is IntfMD, a member of class TRIO.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecute (see page 18725)	Generates an OnAfterExecute (see page 18730) event.
V	DoBeforeExecute (see page 18725)	Generates an OnBeforeExecute (see page 18730) event.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetResponseStream (see page 18726)	Returns a newly created stream that receives the results of executing a method call.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 19933)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.

	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 18724)	Updates the reference count after the last constructor has executed.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 18725)	Prevents the destruction of a TRIO (see page 18718) instance if the reference count is greater than zero.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 19932)	Creates an instance of THTTPRIO (see page 19928).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 19932)	Destroys the instance of a THTTPRIO (see page 19928) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.

	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetRIO (see page 18734)	Returns the value of the RIO (see page 18734) property.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 18726)	Allocates memory for each instance of a remote interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	QueryInterface (see page 19934)	Implements the IInterface QueryInterface method.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	RefCount (see page 18728)	Maintains a reference count on the use of the TRIO (see page 18718) object's interface.
R	RIO (see page 18734)	Provides access to the remote interfaced object that implements this interface.
R	SOAPHeaders (see page 18728)	Provides access to the headers of SOAP messages that the remote interface object sends or receives.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	WebNode (see page 18729)	Handles the execution of method calls on the invokable interface.
R	WSDLItems (see page 19937)	Provides access to the definitions in the WSDL file specified by WSDLLocation (see page 19938).

Published Events

	Name	Description
	OnAfterExecute (see page 18730)	Occurs when TRIO (see page 18718) receives the results from executing a method of the invokable interface.
	OnBeforeExecute (see page 18730)	Occurs before TRIO (see page 18718) sends an encoded method call for execution.
	OnGetAttachment (see page 18731)	Occurs when TRIO (see page 18718) reads each attachment included in the results from executing a method of the invokable interface.
	OnSendAttachment (see page 18731)	Occurs when TRIO (see page 18718) sends each attachment that is the value of an argument of the method call on the invokable interface.

Published Properties

	Name	Description
	Converter (see page 19935)	Handles the marshaling and unmarshaling of calls to an invokable interface.
	HTTPWebNode (see page 19935)	Handles the execution of method calls on the invokable interface.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Port (see page 19936)	Specifies the port to use when sending the SOAP message to execute a method call.
	Service (see page 19936)	Specifies the Service to use when sending the SOAP message to execute a method call.
	Tag (see page 2808)	Stores an integer value as part of a component.
	URL (see page 19937)	Specifies the URL that identifies the target Web Service (see page 19936) application.
	WSDLLocation (see page 19938)	Specifies the location of the WSDL document that describes the target Web Service (see page 19936).

141.1.1.2 THTTPRIO.Create Constructor

Creates an instance of THTTPRIO (see page 19928).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall THTTPRIO(TComponent * AOwner);
```

Description

Call Create to instantiate a THTTPRIO (see page 19928) object at runtime. THTTPRIO (see page 19928) components added to forms or data modules at design time are created automatically.

The AOwner parameter specifies another component (usually a form or data module) that is responsible for freeing the THTTPRIO (see page 19928) instance. It becomes the value of the Owner property. If the THTTPRIO (see page 19928) instance is created with AOwner set to nil (Delphi) or NULL (C++), then it is automatically freed when its reference count drops to zero.

See Also

RefCount (see page 18728)

Owner (see page 2807)

141.1.1.3 THTTPRIO.Destroy Destructor

Destroys the instance of a THTTPRIO (see page 19928) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THTTPRIO();
```

Description

There is no need to explicitly free a THTTPRIO (see page 19928) instance.

If the THTTPRIO (see page 19928) object is created with an Owner, then that Owner handles the freeing of the THTTPRIO (see page 19928) instance. If the THTTPRIO (see page 19928) object is created without an Owner, then it is automatically freed when the reference count on its interface drops to zero.

See Also

Create (see page 19932)

Owner (see page 2807)

RefCount (see page 18728)

141.1.1.4 THTTPRIO Methods

The methods of the THTTPRIO class are listed here.

Protected Methods

	Name	Description
	Notification (see page 19933)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
	QueryInterface (see page 19934)	Implements the IInterface QueryInterface method.

141.1.1.4.1 THTTPRIO.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

141.1.1.4.2 THTTPRIO.QueryInterface Method

Implements the IInterface QueryInterface method.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface is called through the IInterface interface to obtain an interface pointer for the interface identified by the IID parameter. In Delphi, this call is generated when an application casts THTTPRIO (see page 19928) to an invokable interface using the as operator.

THTTPRIO (see page 19928) can only return a single type of interface pointer. The first time an application calls QueryInterface, THTTPRIO (see page 19928) generates an in-memory method table to support calls to the methods on the requested interface.

IID specifies the invokable interface that the application wants to call using this THTTPRIO (see page 19928) instance. If the THTTPRIO (see page 19928) object has already been typecast to an interface, then IID must identify that interface. If the THTTPRIO (see page 19928) instance has not yet been typecast to an interface, then IID must be a registered invokable interface.

Obj returns an interface pointer for the specified interface, unless THTTPRIO (see page 19928) was already implementing a different interface or IID identified an interface that was not registered.

QueryInterface returns 0 (S_OK) if Obj successfully returns an interface pointer. It returns E_NOINTERFACE otherwise.

Note: In C++, a second, templatized syntax takes a variable that is a DelphiInterface wrapper for the desired interface type.

If QueryInterface successfully returns an interface pointer, it automatically increments the reference count.

See Also

InvRegistry (see page 15620)

RegisterInterface (see page 15552)

RefCount (see page 18728)

141.1.1.5 THTTPRIO Properties

The properties of the THTTPRIO class are listed here.

Public Properties

	Name	Description
R	WSDLItems (see page 19937)	Provides access to the definitions in the WSDL file specified by WSDLLocation (see page 19938).

Published Properties

	Name	Description
	Converter (see page 19935)	Handles the marshaling and unmarshaling of calls to an invokable interface.
	HTTPWebNode (see page 19935)	Handles the execution of method calls on the invokable interface.
	Port (see page 19936)	Specifies the port to use when sending the SOAP message to execute a method call.
	Service (see page 19936)	Specifies the Service to use when sending the SOAP message to execute a method call.
	URL (see page 19937)	Specifies the URL that identifies the target Web Service (see page 19936) application.
	WSDLLocation (see page 19938)	Specifies the location of the WSDL document that describes the target Web Service (see page 19936).

141.1.1.5.1 THTTPRIO.Converter Property

Handles the marshaling and unmarshaling of calls to an invokable interface.

Pascal

```
property Converter: TOPToSoapDomConvert;
```

C++

```
__property TOPToSoapDomConvert Converter;
```

Description

THTTPRIO (see page 19928) uses Converter internally to marshal and unmarshal calls to an invokable interface. That is, when the THTTPRIO (see page 19928) object has been typecast to an invokable interface, it implements each method call by using Converter to marshal the call from an internal format into an encoding protocol, and to unmarshal any return values after that call is executed.

See Also

HTTPWebNode (see page 19935)

141.1.1.5.2 THTTPRIO.HTTPWebNode Property

Handles the execution of method calls on the invokable interface.

Pascal

```
property HTTPWebNode: THTTPReqResp;
```

C++

```
__property THTTPReqResp HTTPWebNode;
```

Description

THTTPRIO (see page 19928) uses HTTPWebNode internally to execute method calls. It is the implementation of the WebNode (see page 25429) property.

THTTPRio executes a method call by sending a SOAP request as an HTTP message to the Web Service (see page 19936) identified by either the URL (see page 19937) property or the WSDLLocation (see page 19938), Service (see page 19936), and Port (see page 19936) properties. If you need to use a proxy server to access the Web Service (see page 19936) application, or if the server requires authentication, use the properties of the THTTPReqResp object to provide the necessary information.

See Also

WebNode (see page 18729)

URL (see page 19937)

WSDLLocation (see page 19938)

Service (see page 19936)

Port (see page 19936)

141.1.1.5.3 THTTPRIO.Port Property

Specifies the port to use when sending the SOAP message to execute a method call.

Pascal

```
property Port: string;
```

C++

```
__property AnsiString Port;
```

Description

Set Port to indicate the desired port within the service specified by the Service (see page 19936) property. Port is the name of a port defined in the WSDL document specified by the WSDLLocation (see page 19938) property. Once you have set the WSDLLocation (see page 19938) property, you can select a value for Port from a drop-down list in the Object Inspector.

A port defines a binding (which indicates the interface to call and the encoding and transport protocols to use) and the address of the server machine.

See Also

WSDLLocation (see page 19938)

Service (see page 19936)

HTTPWebNode (see page 19935)

141.1.1.5.4 THTTPRIO.Service Property

Specifies the Service to use when sending the SOAP message to execute a method call.

Pascal

```
property Service: string;
```

C++

```
__property AnsiString Service;
```

Description

Set Service to indicate the Service that you want to use by making a remote method call. Service is the name of a Service defined in the WSDL document specified by the WSDLLocation (see page 19938) property. Once you have set the WSDLLocation (see page 19938) property, you can select a value for Service from a drop-down list in the Object Inspector.

A Service is a set of related interfaces provided by a Web Service application. The interface that THTTPRIO (see page 19928) represents must be on one of the interfaces that make up the specified service.

Note: When defining the target Web Service application by referring to a WSDL document, you must set the Port (see page 19936) property as well as the Service and WSDLLocation (see page 19938) properties.

See Also

WSDLLocation (see page 19938)

Port (see page 19936)

HTTPWebNode (see page 19935)

141.1.1.5.5 THTTPRIO.URL Property

Specifies the URL that identifies the target Web Service (see page 19936) application.

Pascal

```
property URL: string;
```

C++

```
__property AnsiString URL;
```

Description

Use URL to identify the target application that implements the invokable interface when there is no need to look up connection information from a WSDL document dynamically at runtime. If the server was written using the VCL, the path portion of this URL must match the target path name of the THTTPSoapDispatcher component on the Web Service (see page 19936) application. The protocol portion must specify http or https. Thus, for servers written in the VCL, URL has the form

`http://MyHost.com/scripts/MyWebService.dllso/SOAP/IMyInterface`

When you use URL to identify the server, THTTPRIO (see page 19928) obtains a namespace for the target interface from the invocation registry. It is necessary, therefore, that the target interface be registered with the same namespace in both client and server applications

Note: In Delphi, this happens automatically if you

Use the same unit to declare and register the target interface in both client and server applications.

Use the same value for the AppNamespacePrefix global variable.

Use the automatically-generated namespace.

If you are using C++, or if one of the above criteria is not met, you can ensure that the target interface is registered with the same namespace by explicitly specifying a namespace in the code that registers the interface.

Note: If you want your application to look up connection information from a WSDL document dynamically at runtime, you must use the WSDLLocation (see page 19938), Service (see page 19936), and Port (see page 19936) properties instead. Using the URL property is mutually exclusive with the use of WSDLLocation (see page 19938), Service (see page 19936) and Port (see page 19936).

See Also

WSDLLocation (see page 19938)

WebDispatch (see page 24634)

HTTPWebNode (see page 19935)

AppNamespacePrefix

141.1.1.5.6 THTTPRIO.WSDLItems Property

Provides access to the definitions in the WSDL file specified by WSDLLocation (see page 19938).

Pascal

```
property WSDLItems: TWSDLItems;
```

C++

```
__property TWSDLItems WSDLItems;
```

Description

WSDLItems is a helper object whose methods return the definitions included in the WSDL file specified by WSDLLocation (see page 19938). Applications should not need to use this object. Rather, THTTPRio uses this property to obtain information about how to connect to the Web Service (see page 19936) application and how to encode request messages.

See Also

WSDLLocation (see page 19938)

141.1.1.5.7 THTTPRIO.WSDLLocation Property

Specifies the location of the WSDL document that describes the target Web Service (see page 19936).

Pascal

```
property WSDLLocation: string;
```

C++

```
__property AnsiString WSDLLocation;
```

Description

Use WSDLLocation to specify a WSDL document that defines the target interface (port type). WSDLLocation can be either a fully qualified path name, or a URL (see page 19937) that indicates where the WSDL document can be found.

When using a WSDL document to indicate the target Web Service (see page 19936), you must also specify values for the Service (see page 19936) and Port (see page 19936) properties. Once WSDLLocation is set, you can use a drop-down list in the Object Inspector to select values for Port (see page 19936) and Service (see page 19936).

Note: If you do not need to look up connection information from the WSDL document at runtime, you can use the URL (see page 19937) property instead. Using the WSDLLocation property is mutually exclusive with the use of the URL (see page 19937) property.

See Also

WSDLLocation

Port (see page 19936)

Service (see page 19936)

URL (see page 19937)

142 SOAPHTTPDisp Namespace

This is namespace SOAPHTTPDisp.

Topics

Name	Description
Classes (see page 19940)	The following table lists classes in this documentation.
Interfaces (see page 19946)	The following table lists interfaces in this documentation.

Classes

	Name	Description
	THTTPSoapDispatchNode (see page 19940)	THTTPSoapDispatchNode forwards SOAP messages to an invoker for interpretation.

Interfaces

	Name	Description
	IHTTPSoapDispatch (see page 19946)	IHTTPSoapDispatch is the interface for objects that interpret SOAP request messages and execute the corresponding invokable interface.

142.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	THTTPSoapDispatchNode (see page 19940)	THTTPSoapDispatchNode forwards SOAP messages to an invoker for interpretation.

142.1.1 THTTPSoapDispatchNode Class

THTTPSoapDispatchNode forwards SOAP messages to an invoker for interpretation.

Class Hierarchy

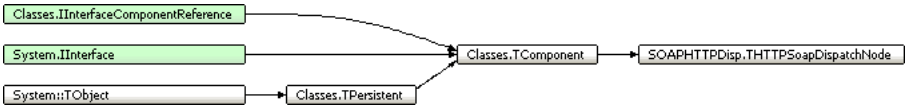

Pascal

```
THTTPSoapDispatchNode = class(TComponent);
```

C++

```
class THTTPSoapDispatchNode : public TComponent;
```

File

SOAPHTTPDisp

Description

Use THTTPSoapDispatchNode as a base class for objects that use an invoker component to interpret and execute SOAP message calls. THTTPSoapDispatchNode maintains a reference to the interface of a component that can interpret and execute SOAP method calls. These calls are requests to execute a method on an invokable interface that is registered in the application. You can register an invokable interface by calling the RegisterInterface method of the TInvokableClassRegistry object that the global InvRegistry function returns.

See Also

THTTPSOAPPascalInvoker

THTTPSOAPCppInvoker

IHTTPSoapDispatch (see page 19946)

InvRegistry

TInvokableClassRegistry (see page 15526)

142.1.1.1 THTTPSoapDispatchNode Members

The following tables list the members exposed by THTTPSoapDispatchNode.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FSoapDispatcher (see page 19943)	This is FSoapDispatcher, a member of class THTTPSoapDispatchNode.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 19944)	Responds when objects are added or removed from the Web module.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.

	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	DispatchSOAP (see page 19944)	Forwards the SOAP message on to the invoker for interpretation and execution.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Dispatcher (see page 19945)	Specifies the interface of the component that interprets SOAP messages.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

142.1.1.2 THTTPSoapDispatchNode Fields

The fields of the THTTPSoapDispatchNode class are listed here.

Protected Fields

	Name	Description
	FSoapDispatcher (see page 19943)	This is FSoapDispatcher, a member of class THTTPSoapDispatchNode.

142.1.1.2.1 THTTPSoapDispatchNode.FSoapDispatcher Field

Pascal

```
FSoapDispatcher: IHTTPSoapDispatch;
```

C++

```
IHTTPSoapDispatch FSoapDispatcher;
```

Description

This is FSoapDispatcher, a member of class THTTPSoapDispatchNode.

142.1.1.3 THTTPSoapDispatchNode Methods

The methods of the THTTPSoapDispatchNode class are listed here.

Protected Methods

	Name	Description
	Notification (see page 19944)	Responds when objects are added or removed from the Web module.

Public Methods

	Name	Description
	DispatchSOAP (see page 19944)	Forwards the SOAP message on to the invoker for interpretation and execution.

142.1.1.3.1 THTTPSoapDispatchNode.DispatchSOAP Method

Forwards the SOAP message on to the invoker for interpretation and execution.

Pascal

```
procedure DispatchSOAP(const Path: WideString; const SoapAction: WideString; const Request:
TStream; Response: TStream); virtual;
```

C++

```
virtual __fastcall DispatchSOAP(const BSTR Path, const BSTR SoapAction, const TStream Request,
TStream Response);
```

Description

DispatchSOAP calls the DispatchSOAP method of the interface specified by the Dispatcher (see page 19945) property. The invoker that implements that interface interprets the message, executes it, and returns the results as a SOAP string.

Path is the path portion of the URL to which the incoming request message was addressed.

SoapAction is the content of the SOAP Action header of the incoming HTTP request message.

Request is a stream from which DispatchSOAP can read the content of the incoming request message. It contains the SOAP encoding of an interface call.

Response is a stream to which DispatchSOAP writes the content for the outgoing response message. DispatchSoap fills it out with the SOAP encoding of any return values.

See Also

DispatchSOAP (see page 19947)

Dispatcher (see page 19945)

142.1.1.3.2 THTTPSoapDispatchNode.Notification Method

Responds when objects are added or removed from the Web module.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. When Operation is opRemove, Notification checks whether AComponent is the implementer of the Dispatcher (see page 19945) property. If so, it sets Dispatcher (see page 19945) to nil (Delphi) or NULL (C++).

See Also

Dispatcher (see page 19945)

142.1.1.4 THTTPSoapDispatchNode Properties

The properties of the THTTPSoapDispatchNode class are listed here.

Published Properties

	Name	Description
	Dispatcher (see page 19945)	Specifies the interface of the component that interprets SOAP messages.

142.1.1.4.1 THTTPSoapDispatchNode.Dispatcher Property

Specifies the interface of the component that interprets SOAP messages.

Pascal

```
property Dispatcher: IHTTPSoapDispatch;
```

C++

```
__property IHTTPSoapDispatch Dispatcher;
```

Description

Set Dispatcher to the interface of a component that interprets SOAP messages and passes them on to the target invokable interface. Dispatcher is the interface of an invoker component, such as THTTPSoapPascalInvoker (Delphi) or THTTPSoapCppInvoker (C++). Invoker components interpret incoming SOAP calls, identify the invokable interface to which they are directed, execute the call, and assemble a response message.

See Also

THTTPSOAPPascalInvoker

THTTPSOAPCppInvoker

142.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IHTTPSoapDispatch (see page 19946)	IHTTPSoapDispatch is the interface for objects that interpret SOAP request messages and execute the corresponding invokable interface.

142.2.1 IHTTPSoapDispatch Interface

IHTTPSoapDispatch is the interface for objects that interpret SOAP request messages and execute the corresponding invokable interface.

Class Hierarchy

SOAPHTTPDisp.IHTTPSoapDispatch

Pascal

```
IHTTPSoapDispatch = interface;
```

C++

```
__interface IHTTPSoapDispatch;
```

File

SOAPHTTPDisp

Description

The THTTPSoapDispatcher component uses this interface to call the object that interprets and responds to the incoming SOAP messages that it receives. By using an interface, THTTPSoapDispatcher can work with any component that interprets and executes SOAP messages.

This interface is implemented by THTTPSoapPascalInvoker and THTTPSoapCppInvoker.

Note: C++ method declarations that use IHTTPSoapDispatch use the _di_IHTTPSoapDispatch type instead. This type is a DelphiInterface wrapper around the IHTTPSoapDispatch interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IHTTPSoapDispatch >
_di_IHTTPSoapDispatch;
```

See Also

THTTPSoapPascalInvoker (see page 19951)

THTTPSoapDispatchNode (see page 19940)

142.2.1.1 IHTTPSoapDispatch Members

The following tables list the members exposed by IHTTPSoapDispatch.

Public Methods

	Name	Description
	DispatchSOAP (see page 19947)	Interprets an incoming SOAP message, calls the target interface, and encodes any return values.

142.2.1.2 IHTTPSoapDispatch Methods

The methods of the IHTTPSoapDispatch class are listed here.

Public Methods

	Name	Description
	DispatchSOAP (see page 19947)	Interprets an incoming SOAP message, calls the target interface, and encodes any return values.

142.2.1.2.1 IHTTPSoapDispatch.DispatchSOAP Method

Interprets an incoming SOAP message, calls the target interface, and encodes any return values.

Pascal

```
procedure DispatchSOAP(const Path: WideString; const SoapAction: WideString; const Request:
TStream; Response: TStream; var BindingType: TWebServiceBindingType);
```

C++

```
__fastcall DispatchSOAP(const BSTR Path, const BSTR SoapAction, const TStream Request, TStream
Response, TWebServiceBindingType BindingType);
```

Description

DispatchSOAP interprets a SOAP message, executes the interface call it represents, and encodes any return values as a SOAP string.

Path is the path portion of the URL to which the incoming request message was addressed.

SoapAction is the content of the SOAP Action header of the incoming HTTP request message.

Request is a stream from which DispatchSOAP can read the content of the incoming request message. It contains the SOAP encoding of an interface call.

Response is a stream to which DispatchSOAP writes the content for the outgoing response message. DispatchSoap fills it out with the SOAP encoding of any return values.

BindingType indicates the format of the Request and Response. It is either btSOAP (to indicate a SOAP packet) or btMime (for a multipart form when the request or response includes attachments). On entry, BindingType indicates the format of the encoded request message. DispatchSOAP changes BindingType to the format of the response message.

See Also

THTTPSoapDispatcher (see page 24628)

143 SOAPHTTPPasInv Namespace

This is namespace SOAPHTTPPasInv.

Topics

Name	Description
Classes (see page 19950)	The following table lists classes in this documentation.

Classes

	Name	Description
	THTTPSoapCppInvoker (see page 19950)	THTTPSoapCppInvoker interprets a SOAP request message and executes the corresponding invokable interface.
	THTTPSoapPascalInvoker (see page 19951)	THTTPSoapPascalInvoker interprets a SOAP request message and executes the corresponding invokable interface.

143.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	THTTPSoapCppInvoker (see page 19950)	THTTPSoapCppInvoker interprets a SOAP request message and executes the corresponding invokable interface.
	THTTPSoapPascalInvoker (see page 19951)	THTTPSoapPascalInvoker interprets a SOAP request message and executes the corresponding invokable interface.

143.1.1 THTTPSoapCppInvoker Class

THTTPSoapCppInvoker interprets a SOAP request message and executes the corresponding invokable interface.

Class Hierarchy

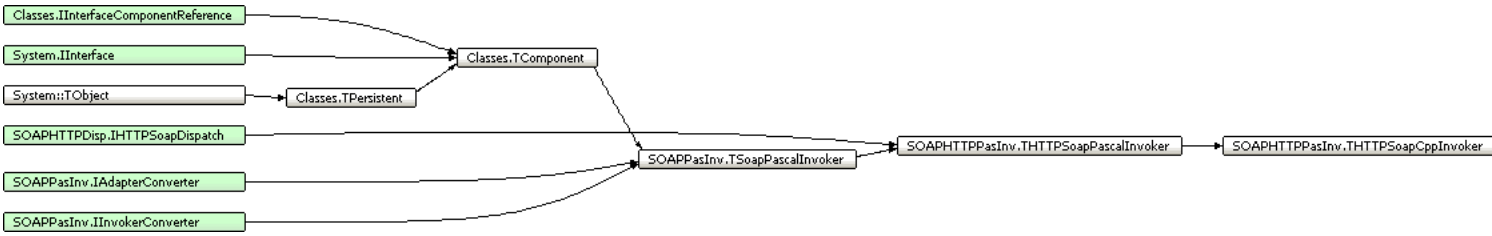

Pascal

```
THTTPSoapCppInvoker = class(THTTPSoapPascalInvoker);
```

C++

```
class THTTPSoapCppInvoker : public THTTPSoapPascalInvoker;
```

File

SOAPHTTPPasInv

Description

Use THTTPSoapCppInvoker in a C++ application that publishes a Web Service using SOAP. THTTPSoapCppInvoker receives an incoming SOAP request from a THTTPSoapDispatcher component, parses it and generates the appropriate call to a registered invokable interface, and encodes the results of that interface call.

THTTPSoapCppInvoker implements the IHTTPSoapDispatch interface, which the dispatcher uses to pass it incoming SOAP requests and to receive the content of the response message. To enable the dispatcher to use this interface, assign it as the value of the THTTPSoapDispatcher component's Dispatcher property.

THTTPSoapCppInvoker uses the invocation registry (InvRegistry) to determine whether the interface call encoded in the incoming SOAP request matches a method on a registered invokable interface. Thus, before the invoker can handle a call, the target interface must be registered using the RegisterInterface method of the invocation registry. In addition, the implementation class for that interface must be registered with the invocation registry.

See Also

IInvokable (see page 22398)

InvRegistry

TInvokableClassRegistry (see page 15526)

THTTPSoapDispatcher (see page 24628)

IHTTPSoapDispatch (see page 19946)

143.1.2 THTTPSoapPascalInvoker Class

THTTPSoapPascalInvoker interprets a SOAP request message and executes the corresponding invokable interface.

Class Hierarchy

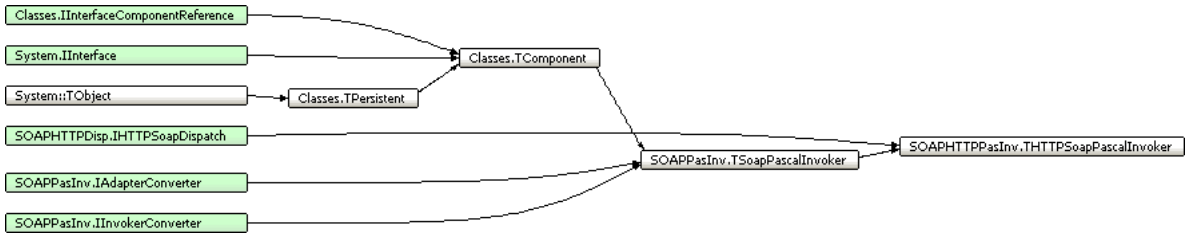

Pascal

```
THTTPSoapPascalInvoker = class(TSoapPascalInvoker, IHTTPSoapDispatch);
```

C++

```
class THTTPSoapPascalInvoker : public TSoapPascalInvoker, public IHTTPSoapDispatch;
```

File

SOAPHTTPPasInv

Description

Use THTTPSoapPascalInvoker in an application that publishes a Web Service using SOAP. THTTPSoapPascalInvoker receives an incoming SOAP request from a THTTPSoapDispatcher component, parses it and generates the appropriate call to a registered invokable interface, and encodes the results of that interface call.

In Delphi, you use THTTPSoapPascalInvoker directly as the invoker in your Web Service application. In C++, you use THTTPSoapCppInvoker (see page 19950), a descendant class, instead.

THTTPSoapPascalInvoker implements the IHTTPSoapDispatch interface, which the dispatcher uses to pass it incoming SOAP requests and to receive the content of the response message. To enable the dispatcher to use this interface, assign it as the value of the THTTPSoapDispatcher component's Dispatcher property.

THTTPSoapPascalInvoker uses the invocation registry (InvRegistry) to determine whether the interface call encoded in the incoming SOAP request matches a method on a registered invokable interface. Thus, before the invoker can handle a call, the target interface must be registered using the RegisterInterface method of the invocation registry. In addition, the implementation class for that interface must be registered with the invocation registry.

See Also

IInvokable (see page 22398)

InvRegistry

TInvokableClassRegistry (see page 15526)

THTTPSoapDispatcher (see page 24628)

IHTTPSoapDispatch (see page 19946)

Understanding Invokable Interfaces

Using the SOAP Application Wizard

Writing Servers that Support Web Services

143.1.2.1 THTTPSoapPascalInvoker Members

The following tables list the members exposed by THTTPSoapPascalInvoker.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FConverter (see page 20025)	This is FConverter, a member of class TSoapPascalInvoker.
	FDomConverter (see page 20025)	This is FDomConverter, a member of class TSoapPascalInvoker.
	FHeadersIn (see page 20025)	This is FHeadersIn, a member of class TSoapPascalInvoker.
	FHeadersOut (see page 20025)	This is FHeadersOut, a member of class TSoapPascalInvoker.
	FSOAPHeaders (see page 20025)	This is FSOAPHeaders, a member of class TSoapPascalInvoker.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDomConverter (see page 20026)	This is GetDomConverter, a member of class TSoapPascalInvoker.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInvoker (see page 20026)	This is GetInvoker, a member of class TSoapPascalInvoker.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20027)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.

	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 20024)	Creates a new TSoapPascalInvoker (see page 20020) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20024)	Frees the memory associated with the TSoapPascalInvoker (see page 20020) object.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DispatchSOAP (see page 19955)	Interprets an incoming SOAP message, calls the target interface, and encodes any return values.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetDomConverter (see page 20033)	This is GetDomConverter, a member of class IAdapterConverter.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetInvoker (see page 20034)	This is GetInvoker, a member of class IInvokerConverter.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invoke (see page 20026)	Interprets a SOAP request message, calls the target interface, and encodes any return values.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterDispatchEvent (see page 20028)	Occurs after the invoker successfully calls an invokable interface to execute a SOAP request.
	BeforeDispatchEvent (see page 20029)	Occurs before the invoker calls an invokable interface to execute a SOAP request.
	BeforeDispatchEvent2 (see page 20029)	Occurs before the invoker calls an invokable interface to execute a SOAP request.
	OnExceptionEvent (see page 20030)	Occurs when the invoker encounters an exception while executing a SOAP request.
	OnGetAttachment (see page 20030)	Occurs when TSoapPascalInvoker (see page 20020) reads each attachment included in an incoming request.
	OnSendAttachment (see page 20030)	Occurs when TSoapPascalInvoker (see page 20020) sends each attachment included in the response to the current request.

Published Properties

	Name	Description
	Converter (see page 20028)	Specifies the interface of the component that handles the marshaling of SOAP messages.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

143.1.2.2 THTTPSoapPascalInvoker Methods

The methods of the THTTPSoapPascalInvoker class are listed here.

Public Methods

	Name	Description
	DispatchSOAP (see page 19955)	Interprets an incoming SOAP message, calls the target interface, and encodes any return values.

143.1.2.2.1 THTTPSoapPascalInvoker.DispatchSOAP Method

Interprets an incoming SOAP message, calls the target interface, and encodes any return values.

Pascal

```
procedure DispatchSOAP(const Path: WideString; const SoapAction: WideString; const Request:
TStream; Response: TStream; var BindingType: TWebServiceBindingType); virtual;
```

C++

```
virtual __fastcall DispatchSOAP(const BSTR Path, const BSTR SoapAction, const TStream Request,
TStream Response, TWebServiceBindingType BindingType);
```

Description

The SOAP dispatcher component (THTTPSoapDispatcher) calls DispatchSOAP when it receives an incoming SOAP message. DispatchSOAP interprets the message, looks up the target interface in the invocation registry, and calls the Invoke method to execute the call and encode any return values as a SOAP string.

Path is the path portion of the URL to which the incoming request message was addressed.

SoapAction is the content of the SOAP Action header of the incoming HTTP request message.

Request is a stream from which DispatchSOAP can read the content of the incoming request message. It contains the SOAP encoding of an interface call.

Response is a stream to which DispatchSOAP writes the SOAP encoding of any return values. THTTPSoapPascalInvoker (see page 19951) uses this as the content for the outgoing response message.

BindingType indicates the format of the Request and Response. It is either btSOAP (to indicate a SOAP packet) or btMime (for a multipart form when the request or response includes attachments). On entry, BindingType indicates the format of the encoded request message. DispatchSOAP changes BindingType to the format of the response message.

See Also

THTTPSoapDispatcher (see page 24628)

IHTTPSoapDispatch (see page 19946)

InvRegistry (see page 15620)

Invoke (see page 20026)

144 SOAPHTTPTrans Namespace

This is namespace SOAPHTTPTrans.

Topics

Name	Description
Classes (see page 19958)	The following table lists classes in this documentation.
Interfaces (see page 19983)	The following table lists interfaces in this documentation.
Structs, Records, Enums (see page 19985)	The following table lists structs, records, enums in this documentation.
Types (see page 19986)	The following table lists types in this documentation.

Classes

	Name	Description
	ESOAPHTTPException (see page 19958)	ESOAPHTTPException is the class for exceptions that occur when sending or receiving an HTTP message that contains a SOAP packet.
	THTTPReqResp (see page 19961)	THTTPReqResp executes a method call on an invokable interface by sending a SOAP message to the server.

Interfaces

	Name	Description
	IHTTPReqResp (see page 19983)	This is class SOAPHTTPTrans.IHTTPReqResp.

Structs, Records, Enums

	Name	Description
	SOAPInvokeOptions (see page 19985)	This is record SOAPHTTPTrans.SOAPInvokeOptions.

Types

Name	Description
TBeforePostEvent (see page 19986)	This is type SOAPHTTPTrans.TBeforePostEvent.
TPostingDataEvent (see page 19986)	This is type SOAPHTTPTrans.TPostingDataEvent.
TReceivingDataEvent (see page 19987)	This is type SOAPHTTPTrans.TReceivingDataEvent.
TSOAPInvokeOptions (see page 19987)	This is type SOAPHTTPTrans.TSOAPInvokeOptions.

144.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ESOAPHTTPException (see page 19958)	ESOAPHTTPException is the class for exceptions that occur when sending or receiving an HTTP message that contains a SOAP packet.
	THTTPReqResp (see page 19961)	THTTPReqResp executes a method call on an invokable interface by sending a SOAP message to the server.

144.1.1 ESOAPHTTPException Class

ESOAPHTTPException is the class for exceptions that occur when sending or receiving an HTTP message that contains a SOAP packet.

Class Hierarchy

Pascal

```
ESOAPHTTPException = class(Exception);
```

C++

```
class ESOAPHTTPException : public Exception;
```

File

SOAPHTTPTrans

Description

ESOAPHTTPException is raised when there is a problem with the HTTP messages that communicate a SOAP request or response in a Web Services application. Such problems include

The application can't send or receive the HTTP message.

The URL that describes the server is missing or badly formed.

The Content-type header of an HTTP message is not consistent with SOAP.

See Also

THTTPReqResp (see page 19961)

144.1.1.1 ESOAPHTTPException Members

The following tables list the members exposed by ESOAPHTTPException.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 19959)	Instantiates an instance of an ESOAPHTTPException (see page 19958) with a simple message string.
	CreateFmt (see page 19960)	Instantiates an instance of an ESOAPHTTPException (see page 19958) with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.
	StatusCode (see page 19961)	Indicates the HTTP status code of the message that gave rise to this exception.

144.1.1.2 ESOAPHTTPException.Create Constructor

Instantiates an instance of an ESOAPHTTPException (see page 19958) with a simple message string.

Pascal

```
constructor Create(const Msg: string; SCode: Integer = 0; Dummy: Integer = 0);
```

C++

```
__fastcall ESOAPHTTPException(const AnsiString Msg, int SCode = 0, int Dummy = 0);
```

Description

Call Create to construct an ESOAPHTTPException (see page 19958) object with a simple message string.

Msg is the string containing the runtime error message to display in the exception dialog box. Msg can be a hard-coded string, or can be a function call that returns a string.

SCode is the HTTP status code, if any, that triggered this exception. It becomes the value of the StatusCode (see page 19961) property.

See Also

CreateFmt (see page 19960)

StatusCode (see page 19961)

Message (see page 22760)

144.1.1.3 ESOAPHTTPException.CreateFmt Constructor

Instantiates an instance of an ESOAPHTTPException (see page 19958) with a formatted message string.

Pascal

```
constructor CreateFmt(const Msg: string; const Args: array of const; SCode: Integer = 0;
Dummy: Integer = 0);
```

C++

```
__fastcall ESOAPHTTPException(const AnsiString Msg, const array of const Args, int SCode = 0,
int Dummy = 0);
```

Description

Call CreateFmt to construct an ESOAPHTTPException (see page 19958) with a formatted message string.

Msg is a string constant containing format specifiers to be replaced with values from Args.

Args is an array of constants containing values to format according to format specifiers in Msg. CreateFmt calls the Format function to transform Msg with the values in Args.

SCode is the HTTP status code, if any, that triggered this exception. It becomes the value of the StatusCode (see page 19961) property.

See Also

Create (see page 19959)

StatusCode (see page 19961)

Message (see page 22760)

144.1.1.4 ESOAPHTTPException Properties

The properties of the ESOAPHTTPException class are listed here.

Public Properties

	Name	Description
	StatusCode (see page 19961)	Indicates the HTTP status code of the message that gave rise to this exception.

144.1.1.4.1 ESOAPHTTPException.StatusCode Property

Indicates the HTTP status code of the message that gave rise to this exception.

Pascal

```
property StatusCode: Integer;
```

C++

```
__property int StatusCode;
```

Description

StatusCode indicates the HTTP status code of the message that caused the application to raise an ESOAPHTTPException (see page 19958) exception. If the exception does not represent an HTTP response with a status code that indicates an error, StatusCode is 0.

The value of StatusCode is set by the ESOAPHTTPException (see page 19958) constructor.

Note: In Delphi code, this is set by the CreateFmt (see page 19960) or Create (see page 19959) constructor.

Note: In C++ code, this is set using one of the two constructor syntaxes that take a status code parameter.

See Also

Create (see page 19959)

CreateFmt (see page 19960)

144.1.2 THTTPReqResp Class

THTTPReqResp executes a method call on an invokable interface by sending a SOAP message to the server.

Class Hierarchy

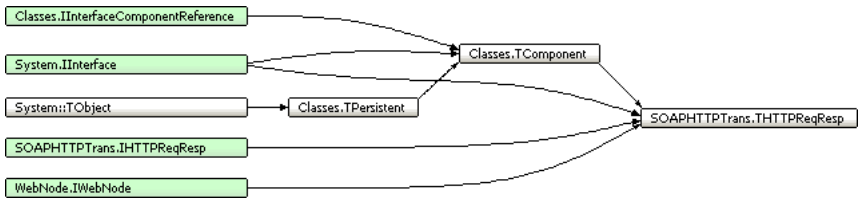

Pascal

```
THTTPReqResp = class(TComponent, IInterface, IWebNode, IHTTPReqResp);
```

C++

```
class THTTPReqResp : public TComponent, public IInterface, public IWebNode, public
IHTTPReqResp;
```

File

SOAPHTTPTrans

Description

THTTPReqResp handles HTTP-based communication with a Web Service provider on behalf of a THTTPRio instance. THTTPRio uses this component to establish a connection to the Web Service provider and perform two tasks:

It can issue a "Get (see page 19971)" request to obtain information from a WSDL document.

It can issue a "Post" request to pass a method call to the server for execution and retrieve the results.

The Web Service that is the target of these HTTP messages can be specified directly, using the SoapAction (see page 19978) and URL (see page 19980) properties, or it can be taken from the SoapAction (see page 19978) in a WSDL document, as specified by the WSDLView (see page 19981) property. Additional properties provide details that are included in the HTTP request message header.

Note: On Windows, THTTPReqResp uses WinInet to establish a connection to the server. Wininet.dll must be installed on the client system. wininet.dll is found in the Windows system directory if you have IE3 or higher installed. WinInet has the advantage that it provides support for secure connections (https). To use WinInet, compile your project without the USE_INDY symbol defined.

Note: On Linux, THTTPReqResp uses an internal instance of the Indy HTTP component (TIdHTTP) to communicate with the server. The Indy component has the advantage that it can be used from an application that acts as both client and server, whereas WinInet only works for pure client code. To use an internal TIdHTTP, compile your project with the USE_INDY symbol defined.

See Also

IWebNode (see page 25430)

THTTPRio

TRIO (see page 18718)

144.1.2.1 THTTPReqResp Members

The following tables list the members exposed by THTTPReqResp.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 19967)	Called when an application uses a component interface.
	_Release (see page 19967)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetMimeBoundary (see page 19972)	Returns the value of the MimeBoundary property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetMimeBoundary (see page 19974)	Sets the value of the MimeBoundary property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
V	AfterConstruction (see page 19968)	Responds after the last constructor executes.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeforeExecute (see page 19968)	Performs transport-specific setup before executing an encoded method call.
	Check (see page 19969)	Raises an exception when an error is detected.
	CheckContentType (see page 19969)	Raises an exception if the response to a SOAP method call has a common invalid content-type header.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Connect (see page 19970)	Establishes or terminates a connection to the Web Service application.
V	Create (see page 19966)	Creates a new THTTPReqResp (see page 19961) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 19966)	Frees the memory associated with the THTTPReqResp (see page 19961) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Execute (see page 19970)	This is the overview for the Execute method overload.
	Execute (see page 25432)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	Get (see page 19971)	Sends a "Get" request to the server and retrieves the results.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHTTPReqResp (see page 19972)	Returns a reference to this component.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 19972)	Allocates memory for each instance of THTTPReqResp (see page 19961).
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	Receive (see page 19973)	Receives the contents of an HTTP response to the message sent by the Send (see page 19973) message.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Send (see page 19973)	Sends the encoded method call as an HTTP-based SOAP message.
V	SendGet (see page 19974)	SendGets sends a "GET" request to the server.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	ConnectTimeout (see page 19976)	Indicates the number of milliseconds that can elapse before aborting an attempt to connect to the server.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	MaxSinglePostSize (see page 19976)	This is MaxSinglePostSize, a member of class THTTPReqResp.
	MimeBoundary (see page 25433)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReceiveTimeout (see page 19978)	Indicates the number of milliseconds that can elapse before canceling a request when no response is received from the server.
	SendTimeout (see page 19978)	Indicates the number of milliseconds that can elapse before timing out the sending of a request.
	SoapAction (see page 19978)	Specifies the SoapAction header to include in the HTTP message that executes the method call.
	URL (see page 19980)	Specifies the URL that identifies the target interface on the target Web Service application.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnBeforePost (see page 19981)	Occurs before THTTPReqResp (see page 19961) sends its message.
	OnPostingData (see page 19982)	Occurs before THTTPReqResp (see page 19961) sends each chunk of data when the content is large.
	OnReceivingData (see page 19982)	Occurs after THTTPReqResp (see page 19961) receives each chunk of data when the content is large.

Published Properties

	Name	Description
	Agent (see page 19975)	Provides the value of the User-Agent header for HTTP messages to the Web Service provider.
	InvokeOptions (see page 19976)	Customizes the way THTTPReqResp (see page 19961) generates or handles headers in the messages it sends to the server.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Password (see page 19976)	Specifies the password that allows the client application to log on when the Web Service provider requires authentication.
	Proxy (see page 19977)	Lists any proxy servers that are required to make the connection.
	ProxyByPass (see page 19977)	Lists the host names of locally registered servers that do not require a proxy.
	Tag (see page 2808)	Stores an integer value as part of a component.
	UDDIBindingKey (see page 19979)	Indicates the binding key of a UDDI entry to use for fail-over support.
	UDDIOperator (see page 19979)	Indicates the URL (see page 19980) of the UDDI registry to use for fail-over support.
	UserName (see page 19980)	Specifies the user name that allows the client application to log on to the server.
	UseUTF8InHeader (see page 19980)	Specifies whether the Content-Type header should specify UTF-8 as a character set.
	WSDLView (see page 19981)	Indicates the target interface method using a WSDL document.

144.1.2.2 THTTPReqResp.Create Constructor

Creates a new THTTPReqResp (see page 19961) object.

Pascal

```
constructor Create(Owner: TComponent); override;
```

C++

```
virtual __fastcall THTTPReqResp(TComponent * Owner);
```

Description

Most applications do not create instances of THTTPReqResp (see page 19961). It is created and used by the THTTPRio class.

The Owner parameter specifies the component that is responsible for freeing the THTTPReqResp (see page 19961) instance.

See Also

Owner (see page 2807)

THTTPRio

144.1.2.3 THTTPReqResp.Destroy Destructor

Frees the memory associated with the THTTPReqResp (see page 19961) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THTTPReqResp();
```

Description

Do not call the Destroy method directly. Instead, use the Free method, which checks for a nil reference before calling Destroy.

The THTTPReqResp (see page 19961) destructor ensures that the connection to the server is closed before the component is freed.

See Also

Free (see page 22300)

144.1.2.4 THTTPReqResp Methods

The methods of the THTTPReqResp class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 19967)	Called when an application uses a component interface.
	_Release (see page 19967)	Called when an application releases a component interface.
	GetMimeBoundary (see page 19972)	Returns the value of the MimeBoundary property.
	SetMimeBoundary (see page 19974)	Sets the value of the MimeBoundary property.

Public Methods

	Name	Description
	AfterConstruction (see page 19968)	Responds after the last constructor executes.
	BeforeExecute (see page 19968)	Performs transport-specific setup before executing an encoded method call.
	Check (see page 19969)	Raises an exception when an error is detected.
	CheckContentType (see page 19969)	Raises an exception if the response to a SOAP method call has a common invalid content-type header.
	Connect (see page 19970)	Establishes or terminates a connection to the Web Service application.
	Execute (see page 19970)	This is the overview for the Execute method overload.
	Get (see page 19971)	Sends a "Get" request to the server and retrieves the results.
	GetHTTPReqResp (see page 19972)	Returns a reference to this component.
	NewInstance (see page 19972)	Allocates memory for each instance of THTTPReqResp (see page 19961).
	Receive (see page 19973)	Receives the contents of an HTTP response to the message sent by the Send (see page 19973) message.
	Send (see page 19973)	Sends the encoded method call as an HTTP-based SOAP message.
	SendGet (see page 19974)	SendGets sends a "GET" request to the server.

144.1.2.4.1 THTTPReqResp._AddRef Method

Called when an application uses a component interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef is a basic implementation of the IInterface method, _AddRef.

Note: If the component is a wrapper for a COM object, __AddRef calls the _AddRef method of that COM object, and returns the resulting reference count.

In all other cases, __AddRef simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __AddRef to implement reference counting.

See Also

Invoke (see page 2781)

QueryInterface (see page 2784)

_Release (see page 2769)

ComObject (see page 2793)

_AddRef (see page 22397)

144.1.2.4.2 THTTPReqResp._Release Method

Called when an application releases a component interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release is a basic implementation of the IInterface method, _Release.

__Release returns the resulting value of the reference count for the component's interface.

Note:

In all other cases, __Release simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __Release to implement reference counting.

See Also

Invoke (see page 2781)

QueryInterface (see page 2784)

_AddRef (see page 2769)

ComObject (see page 2793)

144.1.2.4.3 THTTPReqResp.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of THTTPReqResp (see page 19961) is allocated, its reference count is incremented to prevent its destruction while processing the constructor. AfterConstruction decrements that reference count to re-establish the correct count after all constructors have executed.

See Also

BeforeDestruction (see page 22289)

NewInstance (see page 19972)

144.1.2.4.4 THTTPReqResp.BeforeExecute Method

Performs transport-specific setup before executing an encoded method call.

Pascal

```
procedure BeforeExecute(const IntfMD: TIntfMetaData; const MethMD: TIntfMethEntry;
MethodIndex: Integer; AttachHandler: IMimeAttachmentHandler);
```

C++

```
__fastcall BeforeExecute(const TIntfMetaData IntfMD, const TIntfMethEntry MethMD, int
MethodIndex, IMimeAttachmentHandler AttachHandler);
```

Description

BeforeExecute implements the IWebNode method of the same name. The remote interface object (TRio) calls this method after its OnBeforeExecute event, but before calling the Execute (see page 19971) method to send a request to the server.

BeforeExecute looks up the information needed to call the server from the invocation registry, based on either the URL (see page 19980) that was supplied to the remote interfaced object or on the WSDLLocation, Service, and Port that were supplied to the remote interfaced object. If the method call requires an attachment, BeforeExecute looks up the MIME boundary for the attachment and adds headers to indicate the MIME boundary and content type of the attachment.

IntfMD is the runtime type information (RTTI) of the invokable interface that includes the method to execute.

MethMD is describes the method that is about to be executed.

MethodIndex indicates which overload to use when methMD describes an overloaded method. 0 indicates the first overload, 1 indicates the second overload, and so on.

AttachHandler is an interface to use for handling any attachments that are used as parameters of the method. If AttachHandler is nil (Delphi) or NULL (C++), the method call does not require passing any attachments.

See Also

Execute (see page 19971)

OnBeforeExecute (see page 18730)

BeforeExecute (see page 25431)

URL (see page 19937)

WSDLLocation (see page 19938)

Service (see page 19936)

Port (see page 19936)

144.1.2.4.5 THTTPReqResp.Check Method

Raises an exception when an error is detected.

Pascal

```
procedure Check(Error: Boolean; ShowSOAPAction: Boolean = False);
```

C++

```
__fastcall Check(Boolean Error, Boolean ShowSOAPAction = False);
```

Description

Note: On Windows (when using WinInet), THTTPReqResp (see page 19961) calls Check internally to raise an exception when it encounters an error.

Note: Check is not available on Linux.

Error indicates whether an error condition was detected.

ShowSOAPAction indicates whether the error message should include information about the SOAP Action header.

See Also

Exception (see page 22750)

144.1.2.4.6 THTTPReqResp.CheckContentType Method

Raises an exception if the response to a SOAP method call has a common invalid content-type header.

Pascal

```
procedure CheckContentType;
```

C++

```
__fastcall CheckContentType();
```

Description

CheckContentType checks the Content-type header of the response that THTTPReqResp (see page 19961) receives after sending a request to the Web Service application. If it recognizes the content-type as invalid, CheckContentType raises an ESOAPHTTPException (see page 19958).

CheckContentType does not rule out all invalid content types, only the most common.

See Also

ESOAPHTTPException (see page 19958)

144.1.2.4.7 THTTPReqResp.Connect Method

Establishes or terminates a connection to the Web Service application.

Pascal

```
procedure Connect(Value: Boolean);
```

C++

```
__fastcall Connect(Boolean Value);
```

Description

Note: On Windows, call Connect to open or close a connection to the Web Service application identified by the URL (see page 19980) property. Connect opens a connection using WinInet.

Note: Connect is not available on Linux.

Value indicates whether to open or close the connection. When Value is true, Connect establishes a connection. When Value is false, Connect terminates the connection.

See Also

URL (see page 19980)

144.1.2.4.8 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
	THTTPReqResp.Execute (String, TStream) (see page 19971)	Executes a SOAP method call and returns the encoded results.
	THTTPReqResp.Execute (TStream) (see page 19971)	Executes a SOAP method call and returns the encoded results.
	THTTPReqResp.Execute (TStream, TStream) (see page 19971)	Executes a SOAP method call and returns the encoded results.

144.1.2.4.8.1 THTTPReqResp.Execute Method (String, TStream)

Executes a SOAP method call and returns the encoded results.

Pascal

```
procedure Execute(const DataMsg: String; Resp: TStream); virtual; overload;
procedure Execute(const Request: TStream; Response: TStream); virtual; overload;
function Execute(const Request: TStream): TStream; virtual; overload;
```

C++

```
virtual __fastcall Execute(const AnsiString DataMsg, TStream Resp);
virtual __fastcall Execute(const TStream Request, TStream Response);
virtual __fastcall TStream Execute(const TStream Request);
```

Description

Execute implements the IWebNode method of the same name. THTTPRio uses this method to execute a method on an invokable interface that has been encoded as a SOAP method call.

Execute

1. Connects to the Web Service application identified by the URL (see page 19980) property.
2. Creates an HTTP message with DataMsg or Request as its content and SoapAction (see page 19978) as its SoapAction (see page 19978) header.
3. Sends it to the Web Service application and retrieves the results.
4. Writes the results to the stream specified by Resp or returns a stream from which you can read the results. If using the third syntax, which returns a stream, the caller is responsible for freeing the stream object.

See Also

Execute (see page 25432)

THTTPRio

URL (see page 19980)

Send (see page 19973)

Receive (see page 19973)

SoapAction (see page 19978)

144.1.2.4.9 THTTPReqResp.Get Method

Sends a "Get" request to the server and retrieves the results.

Pascal

```
procedure Get(Resp: TStream); virtual;
```

C++

```
virtual __fastcall Get(TStream Resp);
```

Description

Get sends an HTTP request message to the server with a method type of GET. THTTPRio uses this method to fetch the WSDL document that underlies the WSDLView (see page 19981) property.

Resp is a stream that receives the contents of the response to the "Get" request message.

See Also

THTTPRio

URL (see page 19980)

Send (see page 19973)

Receive (see page 19973)

Execute (see page 19971)

WSDLView (see page 19981)

144.1.2.4.10 THTTPReqResp.GetHTTPReqResp Method

Returns a reference to this component.

Pascal

```
function GetHTTPReqResp: THTTPReqResp;
```

C++

```
__fastcall THTTPReqResp GetHTTPReqResp();
```

Description

GetHTTPReqResp returns this THTTPReqResp (see page 19961) instance (Delphi) or a pointer to this instance (C++).

144.1.2.4.11 THTTPReqResp.GetMimeBoundary Method

Returns the value of the MimeBoundary property.

Pascal

```
function GetMimeBoundary: string;
```

C++

```
__fastcall AnsiString GetMimeBoundary();
```

Description

GetMimeBoundary is the read access method for the MimeBoundary property.

See Also

MimeBoundary (see page 25433)

144.1.2.4.12 THTTPReqResp.NewInstance Method

Allocates memory for each instance of THTTPReqResp (see page 19961).

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
virtual __fastcall TObject * NewInstance();
```

Description

All constructors call NewInstance automatically. NewInstance calls InstanceSize to determine how much memory to allocate from the heap to contain a particular instance. Do not call NewInstance directly.

THTTPReqResp (see page 19961) overrides NewInstance to increment the reference count when a new instance is created. This prevents any constructor from accidentally deleting the instance due to a reference count of zero. The AfterConstruction (see page 19968) method decrements the reference count after all constructors have executed.

See Also

NewInstance (see page 22306)

InstanceSize (see page 22306)

AfterConstruction (see page 19968)

144.1.2.4.13 THTTPReqResp.Receive Method

Receives the contents of an HTTP response to the message sent by the Send (see page 19973) message.

Pascal

```
procedure Receive(Context: Integer; Resp: TStream; IsGet: Boolean = False); virtual;
```

C++

```
virtual __fastcall Receive(int Context, TStream Resp, Boolean IsGet = False);
```

Description

Note: On Windows (when using WinInet), Receive fetches the HTTP response to the message sent by the SendMessage method, and writes its contents to the specified stream.

Note: Receive is not available on Linux.

Context is the context ID that identifies the original request message that the Send (see page 19973) message sent. It is the return value from a previous call to the Send (see page 19973) method.

Resp is a stream to which Receive can write the contents of the response message.

IsGet indicates whether the response stream contains the results from a "Get (see page 19971)" request message.

See Also

Send (see page 19973)

Execute (see page 19971)

144.1.2.4.14 THTTPReqResp.Send Method

Sends the encoded method call as an HTTP-based SOAP message.

Pascal

```
function Send(const ASrc: TStream): Integer; virtual;
```

C++

```
virtual __fastcall int Send(const TStream ASrc);
```

Description

Note: On Windows (when using WinInet), Send sends an HTTP request message with the SoapAction (see page 19978) header specified by SoapAction (see page 19978) to the URL (see page 19980) specified by URL (see page 19980).

Note: Send is not available on Linux.

ASrc is a stream from which to read the contents of a SOAP message for the method call.

Send returns a context ID that identifies this request message. This context ID is passed to the Receive (see page 19973) method to retrieve the corresponding response message from the server.

See Also

Receive (see page 19973)

Execute (see page 19971)

SoapAction (see page 19978)

URL (see page 19980)

144.1.2.4.15 THTTPReqResp.SendGet Method

SendGets sends a "GET" request to the server.

Pascal

```
function SendGet: Integer; virtual;
```

C++

```
virtual __fastcall int SendGet();
```

Description

Note: On Windows (when using WinInet), the Get (see page 19971) method calls SendGet to send the GET request to the URL (see page 19980) specified by URL (see page 19980).

Note: SendGet is not available on Linux.

SendGet returns a context ID that identifies the GET request message. The Get (see page 19971) method uses this context ID to retrieve the corresponding response message from the server.

See Also

Get (see page 19971)

URL (see page 19980)

144.1.2.4.16 THTTPReqResp.SetMimeBoundary Method

Sets the value of the MimeBoundary property.

Pascal

```
procedure SetMimeBoundary(Value: string);
```

C++

```
__fastcall SetMimeBoundary(AnsiString Value);
```

Description

SetMimeBoundary is the write implementation for the MimeBoundary property.

Value is the new value for MimeBoundary.

See Also

MimeBoundary (see page 25433)

144.1.2.5 THTTPReqResp Properties

The properties of the THTTPReqResp class are listed here.

Public Properties

	Name	Description
	ConnectTimeout (see page 19976)	Indicates the number of milliseconds that can elapse before aborting an attempt to connect to the server.
	MaxSinglePostSize (see page 19976)	This is MaxSinglePostSize, a member of class THTTPReqResp.
	ReceiveTimeout (see page 19978)	Indicates the number of milliseconds that can elapse before canceling a request when no response is received from the server.
	SendTimeout (see page 19978)	Indicates the number of milliseconds that can elapse before timing out the sending of a request.
	SoapAction (see page 19978)	Specifies the SoapAction header to include in the HTTP message that executes the method call.
	URL (see page 19980)	Specifies the URL that identifies the target interface on the target Web Service application.

Published Properties

	Name	Description
	Agent (see page 19975)	Provides the value of the User-Agent header for HTTP messages to the Web Service provider.
	InvokeOptions (see page 19976)	Customizes the way THTTPReqResp (see page 19961) generates or handles headers in the messages it sends to the server.
	Password (see page 19976)	Specifies the password that allows the client application to log on when the Web Service provider requires authentication.
	Proxy (see page 19977)	Lists any proxy servers that are required to make the connection.
	ProxyByPass (see page 19977)	Lists the host names of locally registered servers that do not require a proxy.
	UDDIBindingKey (see page 19979)	Indicates the binding key of a UDDI entry to use for fail-over support.
	UDDIOperator (see page 19979)	Indicates the URL (see page 19980) of the UDDI registry to use for fail-over support.
	UserName (see page 19980)	Specifies the user name that allows the client application to log on to the server.
	UseUTF8InHeader (see page 19980)	Specifies whether the Content-Type header should specify UTF-8 as a character set.
	WSDLView (see page 19981)	Indicates the target interface method using a WSDL document.

144.1.2.5.1 THTTPReqResp.Agent Property

Provides the value of the User-Agent header for HTTP messages to the Web Service provider.

Pascal

```
property Agent: string;
```

C++

```
__property AnsiString Agent;
```

Description

Use Agent to supply the SOAP request message with the name of your application. This name is included as a User-Agent header. The User-Agent header is an optional string that describes the application generating the request.

See Also

Title (see page 13723)

144.1.2.5.2 THTTPReqResp.ConnectTimeout Property

Indicates the number of milliseconds that can elapse before aborting an attempt to connect to the server.

Pascal

```
property ConnectTimeout: Integer;
```

C++

```
__property int ConnectTimeout;
```

Description

Use ConnectTimeout to indicate the number of milliseconds that can elapse before THTTPReqResp (see page 19961) abandons the attempt to form a connection.

See Also

SendTimeout (see page 19978)

ReceiveTimeout (see page 19978)

144.1.2.5.3 THTTPReqResp.InvokeOptions Property

Customizes the way THTTPReqResp (see page 19961) generates or handles headers in the messages it sends to the server.

Pascal

```
property InvokeOptions: TSOAPInvokeOptions;
```

C++

```
__property TSOAPInvokeOptions InvokeOptions;
```

Description

Use InvokeOptions to customize the way THTTPReqResp (see page 19961) generates SOAP messages to the invokable interface on the server. InvokeOptions is a set, allowing you to specify that THTTPReqResp (see page 19961) include none of the options.

See Also

SoapAction (see page 19978)

144.1.2.5.4 THTTPReqResp.MaxSinglePostSize Property

Pascal

```
property MaxSinglePostSize: Integer;
```

C++

```
__property int MaxSinglePostSize;
```

Description

This is MaxSinglePostSize, a member of class THTTPReqResp.

144.1.2.5.5 THTTPReqResp.Password Property

Specifies the password that allows the client application to log on when the Web Service provider requires authentication.

Pascal

```
property Password: string;
```

C++

```
__property AnsiString Password;
```

Description

When THTTPReqResp (see page 19961) establishes a connection, it supplies the values of the UserName (see page 19980) and Password properties so that it can log on to the host or proxy. The value of Password can be an empty string if neither the host nor proxy requires authentication.

See Also

UserName (see page 19980)

Connect (see page 19970)

144.1.2.5.6 THTTPReqResp.Proxy Property

Lists any proxy servers that are required to make the connection.

Pascal

```
property Proxy: string;
```

C++

```
__property AnsiString Proxy;
```

Description

Use Proxy if the client application can't resolve the value of host portion of URL (see page 19980) locally. If the server is registered locally, Proxy can be a null string. Otherwise, Proxy lists the host names of any proxy servers that can forward request messages on to the intended server machine.

See Also

ProxyByPass (see page 19977)

URL (see page 19980)

144.1.2.5.7 THTTPReqResp.ProxyByPass Property

Lists the host names of locally registered servers that do not require a proxy.

Pascal

```
property ProxyByPass: string;
```

C++

```
__property AnsiString ProxyByPass;
```

Description

Use ProxyByPass to indicate any server systems to which the client application can make a direct connection. When the URL (see page 19980) property addresses a server in this list, THTTPReqResp (see page 19961) makes a direct connection even if the Proxy (see page 19977) property is specified.

Note: ProxyByPass is only used on Windows, when the connection is formed using WinInet. On Linux, where the connection is formed using Indy, this property has no effect.

Note: When using WinInet, if ProxyByPass is an empty string, THTTPReqResp (see page 19961) checks the registry for any locally known servers and bypasses the proxy server if URL (see page 19980) refers to a registered local server.

See Also

Proxy (see page 19977)

URL (see page 19980)

144.1.2.5.8 THTTPReqResp.ReceiveTimeout Property

Indicates the number of milliseconds that can elapse before canceling a request when no response is received from the server.

Pascal

```
property ReceiveTimeout: Integer;
```

C++

```
__property int ReceiveTimeout;
```

Description

Use ReceiveTimeout to indicate the number of milliseconds that can elapse after sending a request before THTTPReqResp (see page 19961) cancels a request on the assumption that there will be no response forthcoming.

See Also

SendTimeout (see page 19978)

ConnectTimeout (see page 19976)

144.1.2.5.9 THTTPReqResp.SendTimeout Property

Indicates the number of milliseconds that can elapse before timing out the sending of a request.

Pascal

```
property SendTimeout: Integer;
```

C++

```
__property int SendTimeout;
```

Description

Use SendTimeout to indicate the number of milliseconds that can elapse when sending a request before THTTPReqResp (see page 19961) cancels the request.

See Also

ReceiveTimeout (see page 19978)

ConnectTimeout (see page 19976)

144.1.2.5.10 THTTPReqResp.SoapAction Property

Specifies the SoapAction header to include in the HTTP message that executes the method call.

Pascal

```
property SoapAction: string;
```

C++

```
__property AnsiString SoapAction;
```

Description

SoapAction specifies the SoapAction header that appears in the HTTP message that is sent by the Send (see page 19973) method.The value of this header is deduced from information in a WSDL document, as indicated by the WSDLView (see page 19981) property.

See Also

URL (see page 19980)

WSDLView (see page 19981)

Send (see page 19973)

RegisterInterface (see page 15552)

144.1.2.5.11 THTTPReqResp.UDDIBindingKey Property

Indicates the binding key of a UDDI entry to use for fail-over support.

Pascal

```
property UDDIBindingKey: WideString;
```

C++

```
__property BSTR UDDIBindingKey;
```

Description

UDDIBindingKey identifies the binding key for the Web Service TModel the defines the target interface. This key is used to look up the Web Service definition in the UDDI registry that the UDDIOperator (see page 19979) property identifies. UDDIBindingKey and UDDIOperator (see page 19979) are used to implement fail-over support for Web Services that are imported using the UDDI browser.

See Also

UDDIOperator (see page 19979)

144.1.2.5.12 THTTPReqResp.UDDIOperator Property

Indicates the URL (see page 19980) of the UDDI registry to use for fail-over support.

Pascal

```
property UDDIOperator: String;
```

C++

```
__property AnsiString UDDIOperator;
```

Description

UDDIOperator identifies the UDDI registry that was used to import the Web Service definition for the target interface. THTTPReqResp (see page 19961) uses this registry to look up the Web Service definition that the UDDIBindingKey (see page 19979) property identifies. UDDIOperator and UDDIBindingKey (see page 19979) are used to implement fail-over support for Web Services that are imported using the UDDI browser.

See Also

UDDIBindingKey (see page 19979)

144.1.2.5.13 THTTPReqResp.URL Property

Specifies the URL that identifies the target interface on the target Web Service application.

Pascal

```
property URL: string;
```

C++

```
__property AnsiString URL;
```

Description

URL specifies the URL to which to send the message containing the SOAP request.

When communicating with a Web Service based on an imported WSDL document, the value of URL is deduced from information in a WSDL document, as indicated by the WSDLView (see page 19981) property.

Note: In Delphi, you can construct this URL as follows: The path portion must match the target path name of the THTTPSoapDispatcher component on the Web Service application and the protocol portion must specify http or https.

See Also

WSDLView (see page 19981)

WebDispatch (see page 24634)

SoapAction (see page 19978)

Send (see page 19973)

144.1.2.5.14 THTTPReqResp.UserName Property

Specifies the user name that allows the client application to log on to the server.

Pascal

```
property UserName: string;
```

C++

```
__property AnsiString UserName;
```

Description

When THTTPReqResp (see page 19961) establishes a connection, it supplies the values of the UserName and Password (see page 19976) properties so that it can log on to the host or proxy. The value of Password (see page 19976) can be an empty string if neither the host nor proxy requires authentication.

See Also

Password (see page 19976)

Connect (see page 19970)

144.1.2.5.15 THTTPReqResp.UseUTF8InHeader Property

Specifies whether the Content-Type header should specify UTF-8 as a character set.

Pascal

```
property UseUTF8InHeader: Boolean;
```

C++

```
__property Boolean UseUTF8InHeader;
```

Description

Set UseUTF8InHeader to indicate how THTTPReqResp (see page 19961) should generate the Content-Type header of the HTTP request message it sends to the server. When UseUTF8InHeader is true, the Content-Type header indicates that the content of the request is encoded in XML using the UTF-8 character set. Otherwise, the Content-Type header simply indicates that the content is encoded in XML.

See Also

SoapAction (see page 19978)

144.1.2.5.16 THTTPReqResp.WSDLView Property

Indicates the target interface method using a WSDL document.

Pascal

```
property WSDLView: TWSDLView;
```

C++

```
__property TWSDLView WSDLView;
```

Description

WSDLView is a helper object that represents the WSDL file used to define the interfaces and protocols in the target Web Service application, as well as properties that indicate which is the target method call. It is used to generate the URL (see page 19980) and SoapAction (see page 19978) properties when importing definitions from a WSDL document.

See Also

URL (see page 19980)

SoapAction (see page 19978)

144.1.2.6 THTTPReqResp Events

The events of the THTTPReqResp class are listed here.

Published Events

	Name	Description
	OnBeforePost (see page 19981)	Occurs before THTTPReqResp (see page 19961) sends its message.
	OnPostingData (see page 19982)	Occurs before THTTPReqResp (see page 19961) sends each chunk of data when the content is large.
	OnReceivingData (see page 19982)	Occurs after THTTPReqResp (see page 19961) receives each chunk of data when the content is large.

144.1.2.6.1 THTTPReqResp.OnBeforePost Event

Occurs before THTTPReqResp (see page 19961) sends its message.

Pascal

```
property OnBeforePost: TBeforePostEvent;
```

C++

```
__property TBeforePostEvent OnBeforePost;
```

Description

Write an OnBeforePost event handler to make final changes to the HTTP request before it is sent. The Send (see page 19973) method generates OnBeforePost after setting up the content and headers of a request message but before sending it.

See Also

OnPostingData (see page 19982)

Send (see page 19973)

144.1.2.6.2 THTTPReqResp.OnPostingData Event

Occurs before THTTPReqResp (see page 19961) sends each chunk of data when the content is large.

Pascal

```
property OnPostingData: TPostingDataEvent;
```

C++

```
__property TPostingDataEvent OnPostingData;
```

Description

Write an OnPostingData event handler to provide the end-user with feedback of the progress when sending a request that contains a large amount of data. When the content of a request is very large (for example, when it contains large attachments), the Send (see page 19973) method breaks the message into blocks, which it sends separately. After sending each of these blocks, it generates an OnPostingData event. You can use the event handler to update a progress bar or similar control to indicate how much of the message has been sent.

See Also

OnBeforePost (see page 19981)

Send (see page 19973)

OnReceivingData (see page 19982)

144.1.2.6.3 THTTPReqResp.OnReceivingData Event

Occurs after THTTPReqResp (see page 19961) receives each chunk of data when the content is large.

Pascal

```
property OnReceivingData: TReceivingDataEvent;
```

C++

```
__property TReceivingDataEvent OnReceivingData;
```

Description

Write an OnReceivingData event handler to provide the end-user with feedback of the progress when receiving a request that contains a large amount of data. When the content of a response is very large (for example, when it contains large attachments), it is broken into blocks, which are sent separately. After receiving each of these blocks, the Receive (see page 19973) method generates an OnReceivingData event. You can use the event handler to update a progress bar or similar control to indicate how much of the response has been received.

See Also

OnPostingData (see page 19982)

Receive (see page 19973)

144.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IHTTPReqResp (see page 19983)	This is class SOAPHTTPTrans.IHTTPReqResp.

144.2.1 IHTTPReqResp Interface

Class Hierarchy

SOAPHTTPTrans.IHTTPReqResp

Pascal

```
IHTTPReqResp = interface;
```

C++

```
__interface IHTTPReqResp;
```

File

SOAPHTTPTrans

Description

This is class SOAPHTTPTrans.IHTTPReqResp.

144.2.1.1 IHTTPReqResp Members

The following tables list the members exposed by IHTTPReqResp.

Public Methods

	Name	Description
	GetHTTPReqResp (see page 19984)	This is GetHTTPReqResp, a member of class IHTTPReqResp.

144.2.1.2 IHTTPReqResp Methods

The methods of the IHTTPReqResp class are listed here.

Public Methods

	Name	Description
	GetHTTPReqResp (see page 19984)	This is GetHTTPReqResp, a member of class IHTTPReqResp.

144.2.1.2.1 IHTTPReqResp.GetHTTPReqResp Method

Pascal

```
function GetHTTPReqResp: THTTPReqResp;
```

C++

```
__fastcall THTTPReqResp GetHTTPReqResp();
```

Description

This is GetHTTPReqResp, a member of class IHTTPReqResp.

144.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	SOAPInvokeOptions (see page 19985)	This is record SOAPHTTPTrans.SOAPInvokeOptions.

144.3.1 SOAPHTTPTrans.SOAPInvokeOptions Enumeration

Pascal

```
SOAPInvokeOptions = (
  soNoValueForEmptySOAPAction,
  soIgnoreInvalidCerts,
  soNoSOAPActionHeader,
  soAutoCheckAccessPointViaUDDI
);
```

C++

```
enum SOAPInvokeOptions {
  soNoValueForEmptySOAPAction,
  soIgnoreInvalidCerts,
  soNoSOAPActionHeader,
  soAutoCheckAccessPointViaUDDI
};
```

File

SOAPHTTPTrans

Description

This is record SOAPHTTPTrans.SOAPInvokeOptions.

144.4 Types

The following table lists types in this documentation.

Types

Name	Description
TBeforePostEvent (see page 19986)	This is type SOAPHTTPTrans.TBeforePostEvent.
TPostingDataEvent (see page 19986)	This is type SOAPHTTPTrans.TPostingDataEvent.
TReceivingDataEvent (see page 19987)	This is type SOAPHTTPTrans.TReceivingDataEvent.
TSOAPInvokeOptions (see page 19987)	This is type SOAPHTTPTrans.TSOAPInvokeOptions.

144.4.1 SOAPHTTPTrans.TBeforePostEvent Type

Pascal

```
TBeforePostEvent = procedure (const HTTPReqResp: THTTPReqResp; Data: Pointer) of object;
```

C++

```
(const HTTPReqResp: THTTPReqResp; Data: Pointer) ( TBeforePostEvent)();
```

File

SOAPHTTPTrans

Description

This is type SOAPHTTPTrans.TBeforePostEvent.

144.4.2 SOAPHTTPTrans.TPostingDataEvent Type

Pascal

```
TPostingDataEvent = procedure (Sent: Integer; Total: Integer) of object;
```

C++

```
(Sent: Integer; Total: Integer) ( TPostingDataEvent)();
```

File

SOAPHTTPTrans

Description

This is type SOAPHTTPTrans.TPostingDataEvent.

144.4.3 SOAPHTTPTrans.TReceivingDataEvent Type

Pascal

```
TReceivingDataEvent = procedure (Read: Integer; Total: Integer) of object;
```

C++

```
(Read: Integer; Total: Integer) ( TReceivingDataEvent)();
```

File

SOAPHTTPTrans

Description

This is type SOAPHTTPTrans.TReceivingDataEvent.

144.4.4 SOAPHTTPTrans.TSOAPInvokeOptions Type

Pascal

```
TSOAPInvokeOptions = set of SOAPInvokeOptions;
```

C++

```
SOAPInvokeOptions TSOAPInvokeOptions;
```

File

SOAPHTTPTrans

Description

This is type SOAPHTTPTrans.TSOAPInvokeOptions.

145 SOAPLinked Namespace

This is namespace SOAPLinked.

Topics

Name	Description
Classes (see page 19990)	The following table lists classes in this documentation.

Classes

	Name	Description
	TLinkedRIO (see page 19990)	TLinkedRIO makes calls to a registered invokable interface that is implemented in the same application.
	TLinkedWebNode (see page 19997)	TLinkedWebNode executes a method call on an invokable interface that is implemented in the same application.
	TLogLinkedWebNode (see page 20005)	This is class SOAPLinked.TLogLinkedWebNode.

145.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TLinkedRIO (see page 19990)	TLinkedRIO makes calls to a registered invokable interface that is implemented in the same application.
	TLinkedWebNode (see page 19997)	TLinkedWebNode executes a method call on an invokable interface that is implemented in the same application.
	TLogLinkedWebNode (see page 20005)	This is class SOAPLinked.TLogLinkedWebNode.

145.1.1 TLinkedRIO Class

TLinkedRIO makes calls to a registered invokable interface that is implemented in the same application.

Class Hierarchy

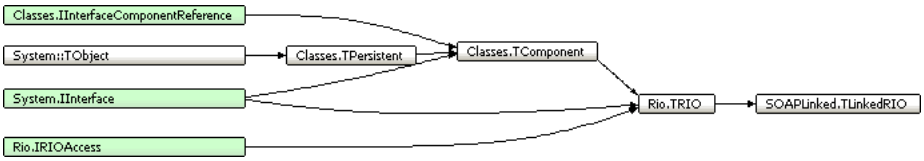

Pascal

```
TLinkedRIO = class(TRIO);
```

C++

```
class TLinkedRIO : public TRIO;
```

File

SOAPLinked

Description

Use TLinkedRIO to generate calls to registered invokable interfaces within a Web Service application. Unlike other remote interfaced objects (TRIO descendants), TLinkedRIO provides no mechanism for connecting to a remote server. Rather, it passes calls directly to a TLinkedWebNode (see page 19997) object, which uses an internal invoker component to look up the target interface and forward calls to the implementation object.

TLinkedRIO is intended primarily for debugging Web Service applications during development. Instead of directly calling the implementation objects in the application, TLinkedRIO marshals the call into its SOAP representation and uses the invocation registry to locate the implementation object and forward the encoded request. If you use the CreateFile (see page 19994) constructor (Delphi) or supply a pair of file names to the constructor (C++), TLinkedRIO saves the encoded value of every method call and the encoded results so that you can track the "messages" that your application receives and sends.

Before TLinkedRIO can call a method table on an invokable interface, the interface must be registered with the invocation registry. To register an invokable interface, use the global InvRegistry function to access the registry, and use its RegisterInterface method.

See Also

InvRegistry

TInvokableClassRegistry (see page 15526)

TSoapPascalInvoker (see page 20020)

TLinkedWebNode (see page 19997)

145.1.1.1 TLinkedRIO Members

The following tables list the members exposed by TLinkedRIO.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FConverter (see page 18723)	This is FConverter, a member of class TRIO.
	FIID (see page 18723)	This is FIID, a member of class TRIO.
	FWebNode (see page 18723)	This is FWebNode, a member of class TRIO.
	IntfMD (see page 18723)	This is IntfMD, a member of class TRIO.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecute (see page 18725)	Generates an OnAfterExecute (see page 18730) event.
V	DoBeforeExecute (see page 18725)	Generates an OnBeforeExecute (see page 18730) event.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetResponseStream (see page 19996)	Returns a newly created stream that receives the results of executing a method call.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 18724)	Updates the reference count after the last constructor has executed.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 18725)	Prevents the destruction of a TRIO (see page 18718) instance if the reference count is greater than zero.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 19994)	Creates an instance of TLinkedRIO (see page 19990).
	CreateFile (see page 19994)	Creates an instance of TLinkedRIO (see page 19990).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 19995)	Destroys the instance of a TLinkedRIO (see page 19990) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetRIO (see page 18734)	Returns the value of the RIO (see page 18734) property.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 18726)	Allocates memory for each instance of a remote interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	QueryInterface (see page 19996)	Implements the IInterface QueryInterface method.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Converter (see page 18728)	Handles the marshaling and unmarshaling of calls to an invokable interface.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	RefCount (see page 18728)	Maintains a reference count on the use of the TRIO (see page 18718) object's interface.
R	RIO (see page 18734)	Provides access to the remote interfaced object that implements this interface.
R	SOAPHeaders (see page 18728)	Provides access to the headers of SOAP messages that the remote interface object sends or receives.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebNode (see page 19997)	Handles the execution of method calls on the invokable interface.

Published Events

	Name	Description
	OnAfterExecute (see page 18730)	Occurs when TRIO (see page 18718) receives the results from executing a method of the invokable interface.

	OnBeforeExecute (see page 18730)	Occurs before TRIO (see page 18718) sends an encoded method call for execution.
	OnGetAttachment (see page 18731)	Occurs when TRIO (see page 18718) reads each attachment included in the results from executing a method of the invokable interface.
	OnSendAttachment (see page 18731)	Occurs when TRIO (see page 18718) sends each attachment that is the value of an argument of the method call on the invokable interface.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

145.1.1.2 TLinkedRIO.Create Constructor

Creates an instance of TLinkedRIO (see page 19990).

Pascal

```
constructor Create(AOwner: TComponent); override; overload;
```

C++

```
virtual __fastcall TLinkedRIO(TComponent * AOwner);
```

Description

Call Create to instantiate a TLinkedRIO (see page 19990) object at runtime.

The AOwner parameter specifies another component (usually a form or data module) that is responsible for freeing the TLinkedRIO (see page 19990) instance. It becomes the value of the Owner property. If the TLinkedRIO (see page 19990) instance is created with AOwner set to nil, then it is automatically freed when its reference count drops to zero.

Tip: To save the encoded method calls that TLinkedRIO (see page 19990) generates and the encoded results that it receives, use the CreateFile (see page 19994) constructor instead.

See Also

RefCount (see page 18728)

Owner (see page 2807)

CreateFile (see page 19994)

145.1.1.3 TLinkedRIO.CreateFile Constructor

Creates an instance of TLinkedRIO (see page 19990).

Pascal

```
constructor CreateFile(AOwner: TComponent; ReqFile: string; RespFile: string); overload;
```

C++

```
__fastcall TLinkedRIO(TComponent * AOwner, AnsiString ReqFile, AnsiString RespFile);
```

Description

Call Create (see page 19994) to instantiate a TLinkedRIO (see page 19990) object when you want to save all of the encoded requests that it generates and the encoded results it receives to a file.

The AOwner parameter specifies another component (usually a form or data module) that is responsible for freeing the TLinkedRIO (see page 19990) instance. It becomes the value of the Owner property. If the TLinkedRIO (see page 19990)

instance is created with AOwner set to nil, then it is automatically freed when its reference count drops to zero.

ReqFile is a file to which TLinkedRIO (see page 19990) writes every encoded request before passing it on to WebNode (see page 19997) for execution. If ReqFile is an empty string, TLinkedRIO (see page 19990) automatically generates a file with the same name as the invoked method. This lets you generate separate log files for each method of the interface.

RespFile is a file to which TLinkedRIO (see page 19990) writes every encoded response it receives from WebNode (see page 19997) after executing a method call. If RespFile is an empty string, TLinkedRIO (see page 19990) automatically generates a file with the name of the method and the string _resp appended. This lets you generate separate log files for responses to each method of the interface.

See Also

RefCount (see page 18728)

Owner (see page 2807)

Create (see page 19994)

WebNode (see page 19997)

145.1.1.4 TLinkedRIO.Destroy Destructor

Destroys the instance of a TLinkedRIO (see page 19990) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TLinkedRIO();
```

Description

There is no need to explicitly free a TLinkedRIO (see page 19990) instance.

If the TLinkedRIO (see page 19990) object is created with an Owner, then that Owner handles the freeing of the TLinkedRIO (see page 19990) instance. If the TLinkedRIO (see page 19990) object is created without an Owner, then it is automatically freed when the reference count on its interface drops to zero.

See Also

Create (see page 19994)

Owner (see page 2807)

RefCount (see page 18728)

145.1.1.5 TLinkedRIO Methods

The methods of the TLinkedRIO class are listed here.

Protected Methods

	Name	Description
	GetResponseStream (see page 19996)	Returns a newly created stream that receives the results of executing a method call.

Public Methods

	Name	Description
	QueryInterface (see page 19996)	Implements the IInterface QueryInterface method.

145.1.1.5.1 TLinkedRIO.GetResponseStream Method

Returns a newly created stream that receives the results of executing a method call.

Pascal

```
function GetResponseStream(BindingType: TWebServiceBindingType): TStream; override;
```

C++

```
virtual __fastcall TStream GetResponseStream(TWebServiceBindingType BindingType);
```

Description

TRIO calls GetResponseStream internally when it executes a method call. It passes the returned stream as the Response parameter of the Execute method on the interface that the WebNode property specifies. After this stream receives the encoded response of a method call, the RIO component decodes the response and frees the stream.

BindingType indicates the expected coding of the response that the stream will hold. GetResponseStream can use this information to determine what type of stream to create.

See Also

Execute (see page 25432)

WebNode (see page 18729)

145.1.1.5.2 TLinkedRIO.QueryInterface Method

Implements the IInterface QueryInterface method.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface is called through the IInterface interface to obtain an interface pointer for the interface identified by the IID parameter. In Delphi, this call is generated when an application casts TLinkedRIO (see page 19990) to an invokable interface using the as operator.

TLinkedRIO (see page 19990) can only return a single type of interface pointer. The first time an application calls QueryInterface, TLinkedRIO (see page 19990) generates an in-memory method table to support calls to the methods on the requested interface.

IID specifies the invokable interface that the application wants to call using this TLinkedRIO (see page 19990) instance. If the TLinkedRIO (see page 19990) object has already been typecast to an interface, then IID must identify that interface. If the TLinkedRIO (see page 19990) instance has not yet been typecast to an interface, then IID must be a registered invokable interface.

Obj returns an interface pointer for the specified interface, unless TLinkedRIO (see page 19990) was already implementing a different interface or IID identified an interface that was not registered.

QueryInterface returns 0 (S_OK) if Obj successfully returns an interface pointer. It returns E_NOINTERFACE otherwise.

Note: In C++, a second, templatized syntax takes a variable that is a DelphiInterface wrapper for the desired interface type.

If QueryInterface successfully returns an interface pointer, it automatically increments the reference count.

See Also

InvRegistry (see page 15620)

RegisterInterface (see page 15552)

RefCount (see page 18728)

145.1.1.6 TLinkedRIO Properties

The properties of the TLinkedRIO class are listed here.

Public Properties

	Name	Description
R	WebNode (see page 19997)	Handles the execution of method calls on the invokable interface.

145.1.1.6.1 TLinkedRIO.WebNode Property

Handles the execution of method calls on the invokable interface.

Pascal

```
property WebNode: TLinkedWebNode;
```

C++

```
__property TLinkedWebNode WebNode;
```

Description

TLinkedRIO (see page 19990) uses WebNode internally to execute method calls. It is the implementation of the IWebNode interface that all TRIO descendants use to send method calls to the Web Service application. In the case of TLinkedRIO (see page 19990), this object can only execute calls on invokable interfaces that are implemented in the same application.

See Also

WebNode (see page 18729)

IWebNode (see page 25430)

145.1.2 TLinkedWebNode Class

TLinkedWebNode executes a method call on an invokable interface that is implemented in the same application.

Class Hierarchy

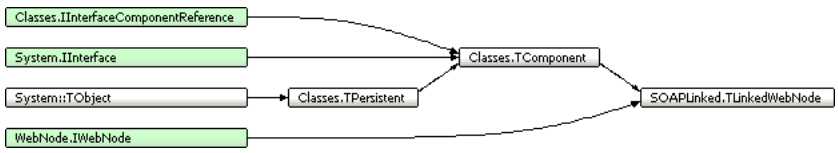

Pascal

```
TLinkedWebNode = class(TComponent, IWebNode);
```

C++

```
class TLinkedWebNode : public TComponent, public IWebNode;
```

File

SOAPLinked

Description

TLinkedWebNode handles the communication between a TLinkedRIO (see page 19990) component and the implementation classes of registered invokable interfaces in the same application. To this end, it implements the IWebNode interface that all TRIO descendants use for executing encoded method calls.

TLinkedRIO (see page 19990) uses an internal invoker (TSoapPascalInvoker) to look up the invokable interface in the invocation registry and forward encoded requests for execution.

See Also

IWebNode (see page 25430)

TLinkedRio

TLinkedRIO (see page 19990)

TSoapPascalInvoker (see page 20020)

145.1.2.1 TLinkedWebNode Members

The following tables list the members exposed by TLinkedWebNode.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetMimeBoundary (see page 20003)	Returns the value of the MimeBoundary (see page 20005) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetResponseStream (see page 20004)	Returns a stream to which the application can write the response to the method call.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetMimeBoundary (see page 20004)	Sets the value of the MimeBoundary (see page 20005) property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeforeExecute (see page 20002)	Caches the RTTI for the method that is about to be executed.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 20001)	Creates a new TLinkedWebNode (see page 19997) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 20001)	Frees the memory associated with the TLinkedWebNode (see page 19997) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 20002)	This is the overview for the Execute method overload.
	Execute (see page 25432)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Invoker (see page 20004)	Executes encoded method calls received from the associated TLinkedRIO (see page 19990) object.
	MimeBoundary (see page 20005)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

145.1.2.2 TLinkedWebNode.Create Constructor

Creates a new TLinkedWebNode (see page 19997) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TLinkedWebNode(TComponent * AOwner);
```

Description

Most applications do not create instances of TLinkedWebNode (see page 19997). It is created and used by the TLinkedRio class.

The Owner parameter specifies the component that is responsible for freeing the TLinkedWebNode (see page 19997) instance.

See Also

Owner (see page 2807)

TLinkedRio

145.1.2.3 TLinkedWebNode.Destroy Destructor

Frees the memory associated with the TLinkedWebNode (see page 19997) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TLinkedWebNode();
```

Description

Do not call the Destroy method directly. Instead, use the Free method, which checks for a nil reference before calling Destroy.

See Also

Free (see page 22300)

145.1.2.4 TLinkedWebNode Methods

The methods of the TLinkedWebNode class are listed here.

Protected Methods

	Name	Description
	GetMimeBoundary (see page 20003)	Returns the value of the MimeBoundary (see page 20005) property.
	GetResponseStream (see page 20004)	Returns a stream to which the application can write the response to the method call.

	SetMimeBoundary (see page 20004)	Sets the value of the MimeBoundary (see page 20005) property.

Public Methods

	Name	Description
	BeforeExecute (see page 20002)	Caches the RTTI for the method that is about to be executed.
	Execute (see page 20002)	This is the overview for the Execute method overload.

145.1.2.4.1 TLinkedWebNode.BeforeExecute Method

Caches the RTTI for the method that is about to be executed.

Pascal

```
procedure BeforeExecute(const IntfMetaData: TIntfMetaData; const MethodMetaData:
TIntfMethEntry; MethodIndex: Integer; AttachHandler: IMimeAttachmentHandler); virtual;
```

C++

```
virtual __fastcall BeforeExecute(const TIntfMetaData IntfMetaData, const TIntfMethEntry
MethodMetaData, int MethodIndex, IMimeAttachmentHandler AttachHandler);
```

Description

BeforeExecute implements the IWebNode method of the same name. The remote interface object (TRio) calls this method after its OnBeforeExecute event, but before calling the Execute (see page 20003) method to send a request to the Web Service application. BeforeExecute allows an IWebNode implementation class to perform any transport-specific tasks prior to establishing a connection. Because TLinkedWebNode (see page 19997) only works with interfaces implemented in the same executable, there is no transport protocol at all, and no transport-specific tasks to perform. BeforeExecute merely caches the RTTI for the method that is about to be executed.

IntfMD is the runtime type information (RTTI) of the invokable interface that includes the method to execute.

MethMD is describes the method that is about to be executed.

MethodIndex indicates which overload to use when methMD describes an overloaded method. 0 indicates the first overload, 1 indicates the second overload, and so on.

AttachHandler is an interface to use for handling any attachments that are used as parameters of the method. If AttachHandler is nil (Delphi) or NULL (C++), the method call does not require passing any attachments.

See Also

Execute (see page 20003)

OnBeforeExecute (see page 18730)

BeforeExecute (see page 25431)

145.1.2.4.2 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
	TLinkedWebNode.Execute (String, TStream) (see page 20003)	Executes a SOAP method call and returns the encoded results.
	TLinkedWebNode.Execute (TStream) (see page 20003)	Executes a SOAP method call and returns the encoded results.

	TLinkedWebNode.Execute (TStream, TStream) (see page 20003)	Executes a SOAP method call and returns the encoded results.

145.1.2.4.2.1 TLinkedWebNode.Execute Method (String, TStream)

Executes a SOAP method call and returns the encoded results.

Pascal

```
procedure Execute(const DataMsg: String; Resp: TStream); virtual; overload;
procedure Execute(const Request: TStream; Response: TStream); virtual; overload;
function Execute(const Request: TStream): TStream; overload;
```

C++

```
virtual __fastcall Execute(const AnsiString DataMsg, TStream Resp);
virtual __fastcall Execute(const TStream Request, TStream Response);
__fastcall TStream Execute(const TStream Request);
```

Description

Execute implements the IWebNode method of the same name. TLinkedRio uses this method to execute a method on an invokable interface that has been encoded as a SOAP method call.

Execute

1. Uses the invocation registry to look up the implementation class for the invokable interface.
2. Uses its internal invoker (the Invoker (see page 20004) property) to unmarshal the request specified by Request or DataMsg and pass it to the implementation class for execution.
3. Returns the encoded results from the invoker in the stream specified by Response or, using the third syntax, in a stream that is the return value of the method. When using the third syntax, the caller is responsible for freeing the returned stream.

See Also

Execute (see page 25432)

TLinkedRio

Invoker (see page 20004)

145.1.2.4.3 TLinkedWebNode.GetMimeBoundary Method

Returns the value of the MimeBoundary (see page 20005) property.

Pascal

```
function GetMimeBoundary: string;
```

C++

```
__fastcall AnsiString GetMimeBoundary();
```

Description

GetMimeBoundary is the read implementation of the MimeBoundary (see page 20005) property.

See Also

MimeBoundary (see page 20005)

145.1.2.4.4 TLinkedWebNode.GetResponseStream Method

Returns a stream to which the application can write the response to the method call.

Pascal

```
function GetResponseStream: TStream; virtual;
```

C++

```
virtual __fastcall TStream GetResponseStream();
```

Description

GetResponseStream creates and returns a memory stream. The linked RIO object uses this stream as its response stream, which receives the response that results from the method call.

See Also

GetResponseStream (see page 18726)

145.1.2.4.5 TLinkedWebNode.SetMimeBoundary Method

Sets the value of the MimeBoundary (see page 20005) property.

Pascal

```
procedure SetMimeBoundary(Value: string);
```

C++

```
__fastcall SetMimeBoundary(AnsiString Value);
```

Description

SetMimeBoundary is the write implementation for the MimeBoundary (see page 20005) property.

Value is the new MIME boundary string.

See Also

MimeBoundary (see page 20005)

145.1.2.5 TLinkedWebNode Properties

The properties of the TLinkedWebNode class are listed here.

Public Properties

	Name	Description
R	Invoker (see page 20004)	Executes encoded method calls received from the associated TLinkedRIO (see page 19990) object.
	MimeBoundary (see page 20005)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

145.1.2.5.1 TLinkedWebNode.Invoker Property

Executes encoded method calls received from the associated TLinkedRIO (see page 19990) object.

Pascal

```
property Invoker: TSoapPascalInvoker;
```

C++

```
__property TSoapPascalInvoker Invoker;
```

Description

Invoker is the internal invoker component that looks up method calls in the invocation registry, forwards encoded requests to the registered implementation class, and returns the encoded results for the associated TLinkedRIO (see page 19990) object.

See Also

TLinkedRIO (see page 19990)

145.1.2.5.2 TLinkedWebNode.MimeBoundary Property

Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

Pascal

```
property MimeBoundary: string;
```

C++

```
__property AnsiString MimeBoundary;
```

Description

When the encoded method call is included in a multipart form, MimeBoundary is the value of the MIME boundary header that uniquely identifies the part of the form that contains the method call. If the method call is not included in a multipart form, MimeBoundary is an empty string.

See Also

MimeBoundary (see page 25433)

145.1.3 TLogLinkedWebNode Class

Class Hierarchy

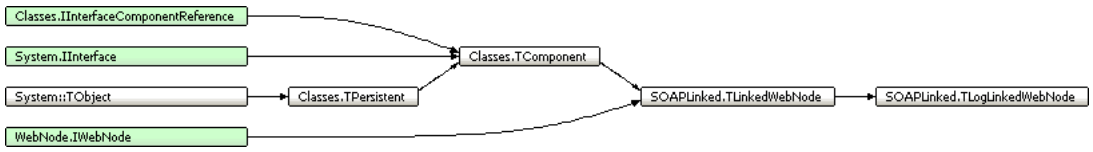

Pascal

```
TLogLinkedWebNode = class(TLinkedWebNode);
```

C++

```
class TLogLinkedWebNode : public TLinkedWebNode;
```

File

SOAPLinked

Description

This is class SOAPLinked.TLogLinkedWebNode.

145.1.3.1 TLogLinkedWebNode Members

The following tables list the members exposed by TLogLinkedWebNode.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetMimeBoundary (see page 20003)	Returns the value of the MimeBoundary (see page 20005) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetResponseStream (see page 20010)	Returns a stream to which the application can write the response to the method call.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetMimeBoundary (see page 20004)	Sets the value of the MimeBoundary (see page 20005) property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.

	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeforeExecute (see page 20002)	Caches the RTTI for the method that is about to be executed.
	BeforeExecute (see page 25431)	Performs any transport-specific setup before executing an encoded method call.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 20008)	Creates a new TLinkedWebNode object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 20009)	Frees the memory associated with the TLinkedWebNode object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 20009)	This is Execute, a member of class TLogLinkedWebNode.
	Execute (see page 20002)	This is the overview for the Execute method overload.
	Execute (see page 25432)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMimeBoundary (see page 25433)	Returns the value of the MimeBoundary (see page 25433) property.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetMimeBoundary (see page 25433)	Sets the value of the MimeBoundary (see page 25433) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Invoker (see page 20004)	Executes encoded method calls received from the associated TLinkedRIO (see page 19990) object.
	MimeBoundary (see page 20005)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.
	MimeBoundary (see page 25433)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ReqFile (see page 20010)	This is ReqFile, a member of class TLogLinkedWebNode.
	RespFile (see page 20010)	This is RespFile, a member of class TLogLinkedWebNode.
	Tag (see page 2808)	Stores an integer value as part of a component.

145.1.3.2 TLogLinkedWebNode.Create Constructor

Creates a new TLinkedWebNode object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TLogLinkedWebNode(TComponent * AOwner);
```

Description

Most applications do not create instances of TLinkedWebNode. It is created and used by the TLinkedRio class.

The Owner parameter specifies the component that is responsible for freeing the TLinkedWebNode instance.

See Also

Owner (see page 2807)

TLinkedRio

145.1.3.3 TLogLinkedWebNode.Destroy Destructor

Frees the memory associated with the TLinkedWebNode object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TLogLinkedWebNode();
```

Description

Do not call the Destroy method directly. Instead, use the Free method, which checks for a nil reference before calling Destroy.

See Also

Free (see page 22300)

145.1.3.4 TLogLinkedWebNode Methods

The methods of the TLogLinkedWebNode class are listed here.

Protected Methods

	Name	Description
	GetResponseStream (see page 20010)	Returns a stream to which the application can write the response to the method call.

Public Methods

	Name	Description
	Execute (see page 20009)	This is Execute, a member of class TLogLinkedWebNode.

145.1.3.4.1 TLogLinkedWebNode.Execute Method

Pascal

```
procedure Execute(const Request: TStream; Response: TStream); override;
```

C++

```
virtual __fastcall Execute(const TStream Request, TStream Response);
```

Description

This is Execute, a member of class TLogLinkedWebNode.

145.1.3.4.2 TLogLinkedWebNode.GetResponseStream Method

Returns a stream to which the application can write the response to the method call.

Pascal

```
function GetResponseStream: TStream; override;
```

C++

```
virtual __fastcall TStream GetResponseStream();
```

Description

GetResponseStream creates and returns a memory stream. The linked RIO object uses this stream as its response stream, which receives the response that results from the method call.

See Also

GetResponseStream (see page 18726)

145.1.3.5 TLogLinkedWebNode Properties

The properties of the TLogLinkedWebNode class are listed here.

Published Properties

	Name	Description
	ReqFile (see page 20010)	This is ReqFile, a member of class TLogLinkedWebNode.
	RespFile (see page 20010)	This is RespFile, a member of class TLogLinkedWebNode.

145.1.3.5.1 TLogLinkedWebNode.ReqFile Property

Pascal

```
property ReqFile: string;
```

C++

```
__property AnsiString ReqFile;
```

Description

This is ReqFile, a member of class TLogLinkedWebNode.

145.1.3.5.2 TLogLinkedWebNode.RespFile Property

Pascal

```
property RespFile: string;
```

C++

```
__property AnsiString RespFile;
```

Description

This is RespFile, a member of class TLogLinkedWebNode.

146 SOAPMidas Namespace

This is namespace SOAPMidas.

Topics

Name	Description
Interfaces (see page 20012)	The following table lists interfaces in this documentation.
Functions (see page 20017)	The following table lists functions in this documentation.

Functions

	Name	Description
	RegDefIAppServerInvClass (see page 20017)	Registers a SOAP data module for use by clients built using Kylix 2 or Delphi 5 prior to update patch 2.

Interfaces

	Name	Description
	IAppServerSOAP (see page 20012)	IAppServerSOAP is the interface for communication with a SOAP data module.

146.1 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAppServerSOAP (see page 20012)	IAppServerSOAP is the interface for communication with a SOAP data module.

146.1.1 IAppServerSOAP Interface

IAppServerSOAP is the interface for communication with a SOAP data module.

Class Hierarchy

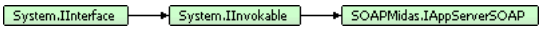

Pascal

```
IAppServerSOAP = interface(IInvokable);
```

C++

```
__interface IAppServerSOAP : public IInvokable;
```

File

SOAPMidas

Description

TSoapConnection and TSoapDataModule use the IAppServerSOAP interface to communicate information in a SOAP-based multi-tiered database application. This is different from other DataSnap components, which use the IAppServer interface. IAppServerSOAP differs from IAppServer primarily in that it uses the stdcall calling convention rather than safecall. This makes it easier for marshalling interface calls when the client or server is written in C++.

Note: C++ method declarations that use IAppServerSOAP use the _di_IAppServerSOAP type instead. This type is a DelphiInterface wrapper around the IAppServerSOAP interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IAppServerSOAP >  _di_IAppServerSOAP;
```

See Also

TSOAPConnection

TSoapDataModule (see page 19884)

IAppServer (see page 16450)

146.1.1.1 IAppServerSOAP Members

The following tables list the members exposed by IAppServerSOAP.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	SAS_ApplyUpdates (see page 20013)	This is SAS_ApplyUpdates, a member of class IAppServerSOAP.
	SAS_DataRequest (see page 20013)	Generates an OnSAS_DataRequest event on the specified provider component.
	SAS_Execute (see page 20014)	SAS_Executes the query or stored procedure bound to a specified provider.
	SAS_GetParams (see page 20015)	This is SAS_GetParams, a member of class IAppServerSOAP.
	SAS_GetProviderNames (see page 20015)	Returns a list of all the available providers on a remote data module.
	SAS_GetRecords (see page 20015)	This is SAS_GetRecords, a member of class IAppServerSOAP.
	SAS_RowRequest (see page 20016)	This is SAS_RowRequest, a member of class IAppServerSOAP.

146.1.1.2 IAppServerSOAP Methods

The methods of the IAppServerSOAP class are listed here.

Public Methods

	Name	Description
	SAS_ApplyUpdates (see page 20013)	This is SAS_ApplyUpdates, a member of class IAppServerSOAP.
	SAS_DataRequest (see page 20013)	Generates an OnSAS_DataRequest event on the specified provider component.
	SAS_Execute (see page 20014)	SAS_Executes the query or stored procedure bound to a specified provider.
	SAS_GetParams (see page 20015)	This is SAS_GetParams, a member of class IAppServerSOAP.
	SAS_GetProviderNames (see page 20015)	Returns a list of all the available providers on a remote data module.
	SAS_GetRecords (see page 20015)	This is SAS_GetRecords, a member of class IAppServerSOAP.
	SAS_RowRequest (see page 20016)	This is SAS_RowRequest, a member of class IAppServerSOAP.

146.1.1.2.1 IAppServerSOAP.SAS_ApplyUpdates Method

Pascal

```
function SAS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; stdcall;
```

C++

```
__fastcall __stdcall OleVariant SAS_ApplyUpdates(const BSTR ProviderName, OleVariant Delta,
int MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

This is SAS_ApplyUpdates, a member of class IAppServerSOAP.

146.1.1.2.2 IAppServerSOAP.SAS_DataRequest Method

Generates an OnSAS_DataRequest event on the specified provider component.

Pascal

```
function SAS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
stdcall;
```

C++

```
__fastcall __stdcall OleVariant SAS_DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

Use SAS_SAS_DataRequest to invoke an OnSAS_DataRequest event handler of the provider named by ProviderName. This

method is provided to allow application developers to customize the communication between a client application and a provider component.

ProviderName is the provider on which the data module generates an OnSAS_DataRequest event.

There is no predefined meaning for the Data parameter or the return value.

See Also

OnDataRequest (see page 18482)

DataRequest (see page 8404)

146.1.1.2.3 IAppServerSOAP.SAS_Execute Method

SAS_Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure SAS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); stdcall;
```

C++

```
__fastcall __stdcall SAS_Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant
Params, OleVariant OwnerData);
```

Description

Use SAS_SAS_Execute to pass an execute command to a remote provider, using a specified set of parameters. SAS_SAS_Execute calls the SAS_Execute method of the specified provider after assigning any parameters. Output parameters that result from executing a query or stored procedure are returned in the Params parameter.

The provider's SAS_Execute method generates a BeforeSAS_Execute event, passes the command on to its associated dataset, and generates an AfterSAS_Execute event. The dataset must represent a query or stored procedure that does not return a cursor.

ProviderName specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the query, or the name of a stored procedure that replaces the dataset's stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

Params encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams method. To decode Params into a TParams object, use the Unpackparams routine.

OwnerData contains custom information that appears as an argument to the provider's BeforeSAS_Execute and AfterSAS_Execute event handlers. This information originates in a client dataset's BeforeSAS_Execute event handler. The value returned in OwnerData is passed to the client dataset's AfterSAS_Execute event handler.

See Also

DataSet (see page 18529)

beforeexecute (see page 18479)

AfterExecute (see page 18476)

Params (see page 8489)

Execute (see page 8413)

Execute (see page 18466)

BeforeExecute (see page 8514)

AfterExecute (see page 8504)

Options (see page 18440)

PackageParams (see page 8534)

UnpackParams (see page 8534)

146.1.1.2.4 IAppServerSOAP.SAS_GetParams Method

Pascal

```
function SAS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
stdcall;
```

C++

```
__fastcall __stdcall OleVariant SAS_GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

This is SAS_GetParams, a member of class IAppServerSOAP.

146.1.1.2.5 IAppServerSOAP.SAS_GetProviderNames Method

Returns a list of all the available providers on a remote data module.

Pascal

```
function SAS_GetProviderNames: TWideStringDynArray; stdcall;
```

C++

```
__fastcall __stdcall TWideStringDynArray SAS_GetProviderNames();
```

Description

Call SAS_SAS_GetProviderNames for a list of all registered providers available from a SOAP data module. These values indicate the possible values for the ProviderName property of a client dataset that connects to the SOAP data module. When calling any of the SOAP data modules through the IAppServerSOAP (see page 20012) interface, pass one of these names to indicate the provider that is the target of the method call.

SAS_SAS_GetProviderNames returns the names of the providers in a dynamic array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

See Also

ProviderName (see page 8492)

Providers (see page 19908)

146.1.1.2.6 IAppServerSOAP.SAS_GetRecords Method

Pascal

```
function SAS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; stdcall;
```

C++

```
__fastcall __stdcall OleVariant SAS_GetRecords(const BSTR ProviderName, int Count, int
RecsOut, int Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

This is SAS_GetRecords, a member of class IAppServerSOAP.

146.1.1.2.7 IAppServerSOAP.SAS_RowRequest Method

Pascal

```
function SAS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; stdcall;
```

C++

```
__fastcall __stdcall OleVariant SAS_RowRequest(const BSTR ProviderName, OleVariant Row, int
RequestType, OleVariant OwnerData);
```

Description

This is SAS_RowRequest, a member of class IAppServerSOAP.

146.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	RegDefIAppServerInvClass (see page 20017)	Registers a SOAP data module for use by clients built using Kylix 2 or Delphi 5 prior to update patch 2.

146.2.1 SOAPMidas.RegDefIAppServerInvClass Function

Registers a SOAP data module for use by clients built using Kylix 2 or Delphi 5 prior to update patch 2.

Pascal

```
procedure RegDefIAppServerInvClass;
```

C++

```
RegDefIAppServerInvClass();
```

File

SOAPMidas

Description

RegDefIAppServerInvClass registers TSoapDataModule as an invokable class that implements IAppServer as a Web Service. C++ applications must add a call to this procedure after the registration of IAppServerSOAP (see page 20012) in the program startup code if they are to be used by clients written using Kylix 2 or Delphi 5 prior to update patch 2. This call is not necessary for other clients of a Web Service-based application server, because they look for the IAppServerSOAP (see page 20012) interface.

Note: Delphi applications do not need to call this procedure.

The following example shows where to place the call to RegDefIAppServerInvClass:

```
void static RegTypes()
{
  InvRegistry()->RegisterInterface(__interfaceTypeinfo(IMyAppServerService));
  InvRegistry()->RegisterInvokableClass(__classid(TMyAppServerService),
MyAppServerServiceFactory);
  RegDefIAppServerInvClass();
}
#pragma statrtup RegTypes 32
```

See Also

TSoapDataModule

147 SOAPPasInv Namespace

This is namespace SOAPPasInv.

Topics

Name	Description
Classes (see page 20020)	The following table lists classes in this documentation.
Interfaces (see page 20032)	The following table lists interfaces in this documentation.
Functions (see page 20035)	The following table lists functions in this documentation.
Types (see page 20036)	The following table lists types in this documentation.

Classes

	Name	Description
	TSoapPascalInvoker (see page 20020)	TSoapPascalInvoker is the base class for components that interpret SOAP requests and execute the corresponding invokable interface.

Functions

	Name	Description
	GetSOAPHeaders (see page 20035)	Returns the interface for accessing the SOAP headers that were received with a SOAP request and supplying the headers for the outgoing response.

Interfaces

	Name	Description
	IAdapterConverter (see page 20032)	This is class SOAPPasInv.IAdapterConverter.
	IInvokerConverter (see page 20033)	This is class SOAPPasInv.IInvokerConverter.

Types

Name	Description
TAfterDispatchEvent (see page 20036)	This is type SOAPPasInv.TAfterDispatchEvent.
TBeforeDispatchEvent (see page 20036)	This is type SOAPPasInv.TBeforeDispatchEvent.
TBeforeDispatchEvent2 (see page 20037)	This is type SOAPPasInv.TBeforeDispatchEvent2.
TOnExceptionEvent (see page 20037)	This is type SOAPPasInv.TOnExceptionEvent.

147.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TSoapPascalInvoker (see page 20020)	TSoapPascalInvoker is the base class for components that interpret SOAP requests and execute the corresponding invokable interface.

147.1.1 TSoapPascalInvoker Class

TSoapPascalInvoker is the base class for components that interpret SOAP requests and execute the corresponding invokable interface.

Class Hierarchy

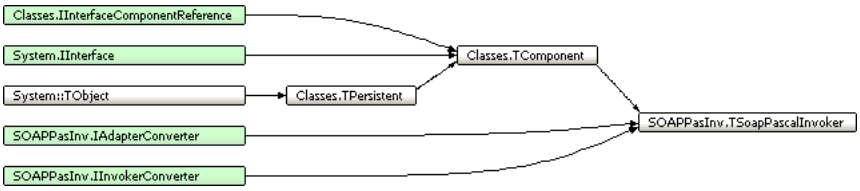

Pascal

```
TSoapPascalInvoker = class(TComponent, IAdapterConverter, IInvokerConverter);
```

C++

```
class TSoapPascalInvoker : public TComponent, public IAdapterConverter, public
IInvokerConverter;
```

File

SOAPPasInv

Description

Use TSoapPascalInvoker as a base class for components that interpret SOAP requests. TSoapPascalInvoker implements the logic that parses a SOAP request, given its target invokable interface, executes the interface call, and encodes the results of that code.

Each TSoapPascalInvoker descendant receives SOAP requests that arrive via a specific transport protocol. The TSoapPascalInvoker descendant must implement the logic to take the incoming SOAP request, identify the target invokable interface, and look up its implementation class and runtime type information (RTTI).

See Also

IInvokable (see page 22398)

THTTPSoapPascalInvoker (see page 19951)

147.1.1.1 TSoapPascalInvoker Members

The following tables list the members exposed by TSoapPascalInvoker.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FConverter (see page 20025)	This is FConverter, a member of class TSoapPascalInvoker.
	FDomConverter (see page 20025)	This is FDomConverter, a member of class TSoapPascalInvoker.
	FHeadersIn (see page 20025)	This is FHeadersIn, a member of class TSoapPascalInvoker.
	FHeadersOut (see page 20025)	This is FHeadersOut, a member of class TSoapPascalInvoker.
	FSOAPHeaders (see page 20025)	This is FSOAPHeaders, a member of class TSoapPascalInvoker.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDomConverter (see page 20026)	This is GetDomConverter, a member of class TSoapPascalInvoker.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInvoker (see page 20026)	This is GetInvoker, a member of class TSoapPascalInvoker.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 20027)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 20024)	Creates a new TSoapPascalInvoker (see page 20020) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 20024)	Frees the memory associated with the TSoapPascalInvoker (see page 20020) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

V	Invoke (see page 20026)	Interprets a SOAP request message, calls the target interface, and encodes any return values.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterDispatchEvent (see page 20028)	Occurs after the invoker successfully calls an invokable interface to execute a SOAP request.
	BeforeDispatchEvent (see page 20029)	Occurs before the invoker calls an invokable interface to execute a SOAP request.
	BeforeDispatchEvent2 (see page 20029)	Occurs before the invoker calls an invokable interface to execute a SOAP request.
	OnExceptionEvent (see page 20030)	Occurs when the invoker encounters an exception while executing a SOAP request.
	OnGetAttachment (see page 20030)	Occurs when TSoapPascalInvoker (see page 20020) reads each attachment included in an incoming request.
	OnSendAttachment (see page 20030)	Occurs when TSoapPascalInvoker (see page 20020) sends each attachment included in the response to the current request.

Published Properties

	Name	Description
	Converter (see page 20028)	Specifies the interface of the component that handles the marshaling of SOAP messages.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

147.1.1.2 TSoapPascalInvoker.Create Constructor

Creates a new TSoapPascalInvoker (see page 20020) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSoapPascalInvoker(TComponent * AOwner);
```

Description

Call Create to instantiate an invoker at runtime. Invoker components added to Web modules at design time (including the one that the Soap Server wizard adds automatically) are created automatically.

The AOwner parameter specifies the component that is responsible for freeing the TSoapPascalInvoker (see page 20020) instance.

See Also

Owner (see page 2807)

147.1.1.3 TSoapPascalInvoker.Destroy Destructor

Frees the memory associated with the TSoapPascalInvoker (see page 20020) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSoapPascalInvoker();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which checks that the invoker component is not nil before calling Destroy.

See Also

Free (see page 22300)

147.1.1.4 TSoapPascalInvoker Fields

The fields of the TSoapPascalInvoker class are listed here.

Protected Fields

	Name	Description
	FConverter (see page 20025)	This is FConverter, a member of class TSoapPascalInvoker.
	FDomConverter (see page 20025)	This is FDomConverter, a member of class TSoapPascalInvoker.
	FHeadersIn (see page 20025)	This is FHeadersIn, a member of class TSoapPascalInvoker.
	FHeadersOut (see page 20025)	This is FHeadersOut, a member of class TSoapPascalInvoker.
	FSOAPHeaders (see page 20025)	This is FSOAPHeaders, a member of class TSoapPascalInvoker.

147.1.1.4.1 TSoapPascalInvoker.FConverter Field

Pascal

```
FConverter: IOPConvert;
```

C++

```
IOPConvert FConverter;
```

Description

This is FConverter, a member of class TSoapPascalInvoker.

147.1.1.4.2 TSoapPascalInvoker.FDomConverter Field

Pascal

```
FDomConverter: TOPToSoapDomConvert;
```

C++

```
TOPToSoapDomConvert FDomConverter;
```

Description

This is FDomConverter, a member of class TSoapPascalInvoker.

147.1.1.4.3 TSoapPascalInvoker.FHeadersIn Field

Pascal

```
FHeadersIn: THeaderList;
```

C++

```
THeaderList FHeadersIn;
```

Description

This is FHeadersIn, a member of class TSoapPascalInvoker.

147.1.1.4.4 TSoapPascalInvoker.FHeadersOut Field

Pascal

```
FHeadersOut: THeaderList;
```

C++

```
THeaderList FHeadersOut;
```

Description

This is FHeadersOut, a member of class TSoapPascalInvoker.

147.1.1.4.5 TSoapPascalInvoker.FSOAPHeaders Field

Pascal

```
FSOAPHeaders: TSOAPHeaders;
```

C++

```
TSOAPHeaders FSOAPHeaders;
```

Description

This is FSOAPHeaders, a member of class TSoapPascalInvoker.

147.1.1.5 TSoapPascalInvoker Methods

The methods of the TSoapPascalInvoker class are listed here.

Protected Methods

	Name	Description
	GetDomConverter (see page 20026)	This is GetDomConverter, a member of class TSoapPascalInvoker.
	GetInvoker (see page 20026)	This is GetInvoker, a member of class TSoapPascalInvoker.
V	Notification (see page 20027)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
V	Invoke (see page 20026)	Interprets a SOAP request message, calls the target interface, and encodes any return values.

147.1.1.5.1 TSoapPascalInvoker.GetDomConverter Method

Pascal

```
function GetDomConverter: TOPToSoapDomConvert;
```

C++

```
__fastcall TOPToSoapDomConvert GetDomConverter();
```

Description

This is GetDomConverter, a member of class TSoapPascalInvoker.

147.1.1.5.2 TSoapPascalInvoker.GetInvoker Method

Pascal

```
function GetInvoker: TSoapPascalInvoker;
```

C++

```
__fastcall TSoapPascalInvoker GetInvoker();
```

Description

This is GetInvoker, a member of class TSoapPascalInvoker.

147.1.1.5.3 TSoapPascalInvoker.Invoke Method

Interprets a SOAP request message, calls the target interface, and encodes any return values.

Pascal

```
procedure Invoke(AClass: TClass; IntfInfo: PTypeInfo; MethName: string; const Request:
TStream; Response: TStream; var BindingType: TWebServiceBindingType); virtual;
```

C++

```
virtual __fastcall Invoke(TClass AClass, PTypeInfo IntfInfo, AnsiString MethName, const
TStream Request, TStream Response, TWebServiceBindingType BindingType);
```

Description

Call Invoke to convert a SOAP request to an interface call on an invokable interface.

AClass is the class that implements the invokable interface. The invoker passes this to the invocation registry so that it can obtain an instance of the implementation class, which it uses to execute the SOAP request.

IntfInfo (see page 15477) is the runtime type information (RTTI) from the invokable interface that is the target of the SOAP request. The invoker uses this to interpret the SOAP request.

MethName is the name of the method to call on the invokable interface.

Request is a stream from which Invoke can read the SOAP request. It contains the SOAP encoding of an interface call.

Response is a stream to which Invoke writes the SOAP encoding of any return values.

BindingType indicates the format of the Request and Response. It is either btSOAP (to indicate a SOAP packet) or btMime (for a multipart form when the request or response includes attachments). On entry, BindingType indicates the format of the encoded request message. Invoke changes BindingType to the format of the response message.

See Also

IInvokable (see page 22398)

InvRegistry (see page 15620)

GetInvokableObjectFromClass (see page 15539)

147.1.1.5.4 TSoapPascalInvoker.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

147.1.1.6 TSoapPascalInvoker Properties

The properties of the TSoapPascalInvoker class are listed here.

Published Properties

	Name	Description
	Converter (see page 20028)	Specifies the interface of the component that handles the marshaling of SOAP messages.

147.1.1.6.1 TSoapPascalInvoker.Converter Property

Specifies the interface of the component that handles the marshaling of SOAP messages.

Pascal

```
property Converter: TOPToSoapDomConvert;
```

C++

```
__property TOPToSoapDomConvert Converter;
```

Description

Converter is the component that marshals and unmarshals method calls for the invoker. The invoker uses this object to interpret incoming method calls and to generate the results it returns.

See Also

TOPToSoapDomConvert (see page 18137)

147.1.1.7 TSoapPascalInvoker Events

The events of the TSoapPascalInvoker class are listed here.

Published Events

	Name	Description
	AfterDispatchEvent (see page 20028)	Occurs after the invoker successfully calls an invokable interface to execute a SOAP request.
	BeforeDispatchEvent (see page 20029)	Occurs before the invoker calls an invokable interface to execute a SOAP request.
	BeforeDispatchEvent2 (see page 20029)	Occurs before the invoker calls an invokable interface to execute a SOAP request.
	OnExceptionEvent (see page 20030)	Occurs when the invoker encounters an exception while executing a SOAP request.
	OnGetAttachment (see page 20030)	Occurs when TSoapPascalInvoker (see page 20020) reads each attachment included in an incoming request.
	OnSendAttachment (see page 20030)	Occurs when TSoapPascalInvoker (see page 20020) sends each attachment included in the response to the current request.

147.1.1.7.1 TSoapPascalInvoker.AfterDispatchEvent Event

Occurs after the invoker successfully calls an invokable interface to execute a SOAP request.

Pascal

```
property AfterDispatchEvent: TAfterDispatchEvent;
```

C++

```
__property TAfterDispatchEvent AfterDispatchEvent;
```

Description

Write a AfterDispatchEvent event handler to examine a SOAP response after the invoker executes a request by calling the target invokable interface. The Invoke (see page 20026) method generates this event after it has executed a requested method and generated a response packet.

Note: AfterDispatchEvent only occurs if the method call is successful. If the invoker encounters an exception when trying to execute the specified method, it generates an OnExceptionEvent (see page 20030) instead.

See Also

BeforeDispatchEvent (see page 20029)

Invoke (see page 20026)

OnExceptionEvent (see page 20030)

147.1.1.7.2 TSoapPascalInvoker.BeforeDispatchEvent Event

Occurs before the invoker calls an invokable interface to execute a SOAP request.

Pascal

```
property BeforeDispatchEvent: TBeforeDispatchEvent;
```

C++

```
__property TBeforeDispatchEvent BeforeDispatchEvent;
```

Description

Write a BeforeDispatchEvent event handler to examine a SOAP request before the invoker calls on the target invokable interface to execute it. The Invoke (see page 20026) method generates this event after it has obtained an interface pointer for the target invokable interface, but before it calls the specified method on that interface.

See Also

AfterDispatchEvent (see page 20028)

Invoke (see page 20026)

BeforeDispatchEvent2 (see page 20029)

147.1.1.7.3 TSoapPascalInvoker.BeforeDispatchEvent2 Event

Occurs before the invoker calls an invokable interface to execute a SOAP request.

Pascal

```
property BeforeDispatchEvent2: TBeforeDispatchEvent2;
```

C++

```
__property TBeforeDispatchEvent2 BeforeDispatchEvent2;
```

Description

Write a BeforeDispatchEvent2 event handler to override the default handling of an incoming request. The Invoke (see page 20026) method generates this event after it has obtained an interface pointer for the target invokable interface, but before it calls the specified method on that interface.

See Also

AfterDispatchEvent (see page 20028)

Invoke (see page 20026)

BeforeDispatchEvent (see page 20029)

147.1.1.7.4 TSoapPascalInvoker.OnExceptionEvent Event

Occurs when the invoker encounters an exception while executing a SOAP request.

Pascal

```
property OnExceptionEvent: TOnExceptionEvent;
```

C++

```
__property TOnExceptionEvent OnExceptionEvent;
```

Description

Write a OnExceptionEvent event handler to examine a SOAP fault packet after the invoker encounters an exception when executing a SOAP request. The Invoke (see page 20026) method generates this event if it encounters an exception, after it generates the SOAP fault packet that describes the problem.

See Also

AfterDispatchEvent (see page 20028)

147.1.1.7.5 TSoapPascalInvoker.OnGetAttachment Event

Occurs when TSoapPascalInvoker (see page 20020) reads each attachment included in an incoming request.

Pascal

```
property OnGetAttachment: TOnGetAttachmentEvent;
```

C++

```
__property TOnGetAttachmentEvent OnGetAttachment;
```

Description

Write an OnGetAttachment event handler to examine the encoded attachments included with an incoming request or to redirect them so that they are saved to a specific file. OnGetAttachment occurs once for each attachment received.

Tip: Use the FileName property of the attachment object to redirect the attachment to a different file.

See Also

OnSendAttachment (see page 20030)

BeforeDispatchEvent (see page 20029)

147.1.1.7.6 TSoapPascalInvoker.OnSendAttachment Event

Occurs when TSoapPascalInvoker (see page 20020) sends each attachment included in the response to the current request.

Pascal

```
property OnSendAttachment: TOnSendAttachmentEvent;
```

C++

```
__property TOnSendAttachmentEvent OnSendAttachment;
```

Description

Write an OnSendAttachment event handler to examine or alter the attachments that are to be included in a message that represents the results of a method call. OnSendAttachment occurs once for each attachment sent with the response message.

See Also

AfterDispatchEvent (see page 20028)

OnGetAttachment (see page 20030)

147.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAdapterConverter (see page 20032)	This is class SOAPPasInv.IAdapterConverter.
	IInvokerConverter (see page 20033)	This is class SOAPPasInv.IInvokerConverter.

147.2.1 IAdapterConverter Interface

Class Hierarchy

SOAPPasInv.IAdapterConverter

Pascal

```
IAdapterConverter = interface;
```

C++

```
__interface IAdapterConverter;
```

File

SOAPPasInv

Description

This is class SOAPPasInv.IAdapterConverter.

147.2.1.1 IAdapterConverter Members

The following tables list the members exposed by IAdapterConverter.

Public Methods

	Name	Description
	GetDomConverter (see page 20033)	This is GetDomConverter, a member of class IAdapterConverter.

147.2.1.2 IAdapterConverter Methods

The methods of the IAdapterConverter class are listed here.

Public Methods

	Name	Description
	GetDomConverter (see page 20033)	This is GetDomConverter, a member of class IAdapterConverter.

147.2.1.2.1 IAdapterConverter.GetDomConverter Method

Pascal

```
function GetDomConverter: TOPToSoapDomConvert;
```

C++

```
__fastcall TOPToSoapDomConvert GetDomConverter();
```

Description

This is GetDomConverter, a member of class IAdapterConverter.

147.2.2 IInvokerConverter Interface

Class Hierarchy

SOAPPasInv.IInvokerConverter

Pascal

```
IInvokerConverter = interface;
```

C++

```
__interface IInvokerConverter;
```

File

SOAPPasInv

Description

This is class SOAPPasInv.IInvokerConverter.

147.2.2.1 IInvokerConverter Members

The following tables list the members exposed by IInvokerConverter.

Public Methods

	Name	Description
	GetInvoker (see page 20034)	This is GetInvoker, a member of class IInvokerConverter.

147.2.2.2 IInvokerConverter Methods

The methods of the IInvokerConverter class are listed here.

Public Methods

	Name	Description
	GetInvoker (see page 20034)	This is GetInvoker, a member of class IInvokerConverter.

147.2.2.2.1 IInvokerConverter.GetInvoker Method

Pascal

```
function GetInvoker: TSoapPascalInvoker;
```

C++

```
__fastcall TSoapPascalInvoker GetInvoker();
```

Description

This is GetInvoker, a member of class IInvokerConverter.

147.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetSOAPHeaders (see page 20035)	Returns the interface for accessing the SOAP headers that were received with a SOAP request and supplying the headers for the outgoing response.

147.3.1 SOAPPasInv.GetSOAPHeaders Function

Returns the interface for accessing the SOAP headers that were received with a SOAP request and supplying the headers for the outgoing response.

Pascal

```
function GetSOAPHeaders: ISOAPHeaders;
```

C++

```
ISOAPHeaders GetSOAPHeaders();
```

File

SOAPPasInv

Description

Call GetSOAPHeaders from an application that implements a Web Service when you do not use TInvokableClass as a base class for the implementation of an invokable interface. GetSOAPHeaders returns an ISOAPHeaders interface, which lets your application examine the SOAP headers in an incoming request message and to add SOAP headers to be sent with the outgoing response.

See Also

TInvokableClass

Defining and Using SOAP Headers

147.4 Types

The following table lists types in this documentation.

Types

Name	Description
TAfterDispatchEvent (see page 20036)	This is type SOAPPasInv.TAfterDispatchEvent.
TBeforeDispatchEvent (see page 20036)	This is type SOAPPasInv.TBeforeDispatchEvent.
TBeforeDispatchEvent2 (see page 20037)	This is type SOAPPasInv.TBeforeDispatchEvent2.
TOnExceptionEvent (see page 20037)	This is type SOAPPasInv.TOnExceptionEvent.

147.4.1 SOAPPasInv.TAfterDispatchEvent Type

Pascal

```
TAfterDispatchEvent = procedure (const MethodName: string; SOAPResponse: TStream) of object;
```

C++

```
(const MethodName: string; SOAPResponse: TStream) ( TAfterDispatchEvent)();
```

File

SOAPPasInv

Description

This is type SOAPPasInv.TAfterDispatchEvent.

147.4.2 SOAPPasInv.TBeforeDispatchEvent Type

Pascal

```
TBeforeDispatchEvent = procedure (const MethodName: string; const Request: TStream) of object;
```

C++

```
(const MethodName: string; const Request: TStream) ( TBeforeDispatchEvent)();
```

File

SOAPPasInv

Description

This is type SOAPPasInv.TBeforeDispatchEvent.

147.4.3 SOAPPasInv.TBeforeDispatchEvent2 Type

Pascal

```
TBeforeDispatchEvent2 = procedure (const MethodName: string; const Request: TStream; Response:
TStream; var BindingType: TWebServiceBindingType; var Handled: Boolean) of object;
```

C++

```
(const MethodName: string; const Request: TStream; Response: TStream; var BindingType:
TWebServiceBindingType; var Handled: Boolean) ( TBeforeDispatchEvent2)();
```

File

SOAPPasInv

Description

This is type SOAPPasInv.TBeforeDispatchEvent2.

147.4.4 SOAPPasInv.TOnExceptionEvent Type

Pascal

```
TOnExceptionEvent = procedure (const MethodName: string; const Request: TStream; const
Response: TStream) of object;
```

C++

```
(const MethodName: string; const Request: TStream; const Response: TStream) (
TOnExceptionEvent)();
```

File

SOAPPasInv

Description

This is type SOAPPasInv.TOnExceptionEvent.

148 Sockets Namespace

This is namespace Sockets.

Topics

Name	Description
Classes (see page 20041)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 20135)	The following table lists structs, records, enums in this documentation.
Types (see page 20140)	The following table lists types in this documentation.
Constants (see page 20144)	The following table lists constants in this documentation.

Classes

	Name	Description
	ESocketError (see page 20041)	Exception class for failures that occur when trying to create, use, or shut down Windows socket objects.
	TBaseSocket (see page 20042)	TBaseSocket encapsulates local operations on file descriptors.
	TClientSocketThread (see page 20059)	Enables creation of separate threads of execution for the client socket in an application.
	TCustomIpClient (see page 20063)	Encapsulates the client network functions.
	TCustomTcpServer (see page 20072)	Ancestor object of all the server socket descendants.
	TIpSocket (see page 20082)	Encapsulates the network support functions.
	TRawSocket (see page 20094)	Encapsulates a raw socket.
	TServerSocketThread (see page 20099)	Enables creation of separate threads of execution for the server socket in an application.
	TTcpClient (see page 20105)	TCP client component.
	TTcpServer (see page 20115)	TTcpServer is the TCP server component.
	TUdpSocket (see page 20123)	UDP component.

Constants

Name	Description
CRLF (see page 20144)	This is constant Sockets.CRLF.

Structs, Records, Enums

	Name	Description
	TIPHeader (see page 20135)	This is record Sockets.TIPHeader.
	TServerSocketBlockMode (see page 20136)	Defines the set of possible socket flags for block mode.
	TSocketDomain (see page 20136)	Set of possible protocol family types used by a socket.
	TSocketType (see page 20138)	The TSocketType type defines the set of possible socket types.

Types

Name	Description
TGetThreadEvent (see page 20140)	This is type Sockets.TGetThreadEvent.
TSocketAcceptEvent (see page 20140)	This is type Sockets.TSocketAcceptEvent.
TSocketBlockMode (see page 20141)	Defines the set of possible flags for blocking mode of the socket.
TSocketDataEvent (see page 20141)	This is type Sockets.TSocketDataEvent.
TSocketErrorEvent (see page 20141)	This is type Sockets.TSocketErrorEvent.
TSocketHost (see page 20142)	This is type Sockets.TSocketHost.
TSocketNotifyEvent (see page 20142)	This is type Sockets.TSocketNotifyEvent.
TSocketPort (see page 20142)	This is type Sockets.TSocketPort.
TSocketProtocol (see page 20143)	This is type Sockets.TSocketProtocol.

148.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ESocketError (see page 20041)	Exception class for failures that occur when trying to create, use, or shut down Windows socket objects.
	TBaseSocket (see page 20042)	TBaseSocket encapsulates local operations on file descriptors.
	TClientSocketThread (see page 20059)	Enables creation of separate threads of execution for the client socket in an application.
	TCustomIpClient (see page 20063)	Encapsulates the client network functions.
	TCustomTcpServer (see page 20072)	Ancestor object of all the server socket descendants.
	TIpSocket (see page 20082)	Encapsulates the network support functions.
	TRawSocket (see page 20094)	Encapsulates a raw socket.
	TServerSocketThread (see page 20099)	Enables creation of separate threads of execution for the server socket in an application.
	TTcpClient (see page 20105)	TCP client component.
	TTcpServer (see page 20115)	TTcpServer is the TCP server component.
	TUdpSocket (see page 20123)	UDP component.

148.1.1 ESocketError Class

Exception class for failures that occur when trying to create, use, or shut down Windows socket objects.

Class Hierarchy

Pascal

```
ESocketError = class(Exception);
```

C++

```
class ESocketError : public Exception;
```

File

Sockets

Description

ESocketError is the exception class for failures that occur when trying to create, use, or shut down Windows socket objects.

ESocketError is thrown if:

A socket cannot be created.

A socket cannot be initialized.

A socket cannot be opened.

An application tries to change the properties of an already open socket.

A failure occurs when reading from or writing to a socket.

A socket cannot be properly closed.

There are two ESocketError classes defined, one in the ScktComp (see page 18737) unit and one in the Sockets (see page 20039) unit. The two exception classes are identically defined and serve the same function for different categories of socket objects. The two definitions occur simply to ensure that ESocketError is defined for all socket objects.

See Also

TCustomWinSocket (see page 18799)

148.1.2 TBaseSocket Class

TBaseSocket encapsulates local operations on file descriptors.

Class Hierarchy

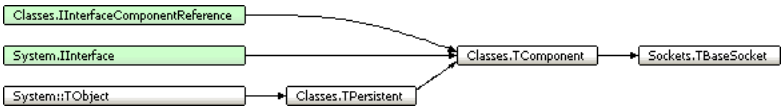

Pascal

```
TBaseSocket = class(TComponent);
```

C++

```
class TBaseSocket : public TComponent;
```

File

Sockets

Description

The TBaseSocket component encapsulates local operations on files descriptors, which are objects that a program uses to read from or write to an open file, or open a network socket or a variety of other devices.

Do not create instances of TBaseSocket. Use TBaseSocket as a base class when declaring non-visual components that can appear on the component palette and be used in the form designer. Properties and methods of TBaseSocket provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TIpSocket (see page 20082)

148.1.2.1 TBaseSocket Members

The following tables list the members exposed by TBaseSocket.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.

	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoCreateHandle (see page 20047)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 20047)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoHandleError (see page 20048)	This is DoHandleError, a member of class TBaseSocket.
	DoReceive (see page 20048)	This is DoReceive, a member of class TBaseSocket.
	DoSend (see page 20048)	This is DoSend, a member of class TBaseSocket.
	ErrorCheck (see page 20048)	This is ErrorCheck, a member of class TBaseSocket.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 20049)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBytesReceived (see page 20053)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 20053)	This is SetBytesSent, a member of class TBaseSocket.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnCreateHandle (see page 20057)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 20057)	Occurs when a socket is deactivated.

	OnError (see page 20058)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 20058)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 20058)	Occurs just before a client socket writes information to the socket connection.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Close (see page 20047)	Shuts down the socket connection.
V	Create (see page 20046)	Creates an instance of TBaseSocket (see page 20042).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 20046)	Frees the memory associated with the TBaseSocket (see page 20042) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MapDomain (see page 20049)	Maps the domain.
	MapSockType (see page 20050)	Maps the socket type.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Open (see page 20050)	Creates the socket connection.
	PeekBuf (see page 20050)	Reads the buffer.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReceiveBuf (see page 20051)	Reads data buffers of to the socket.
	ReceiveIn (see page 20051)	Reads delimited lines of data from the socket.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Select (see page 20051)	Tests for socket activity.
	SendBuf (see page 20052)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 20052)	Writes delimited lines of data to the socket.
	SendStream (see page 20053)	Writes a stream of data to the socket.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WaitForData (see page 20053)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
	Active (see page 20054)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 20054)	Sets or returns the blocking mode of the socket.
R	BytesReceived (see page 20055)	Returns the number of bytes received by the socket.
R	BytesSent (see page 20055)	Returns the number of bytes sent from the socket.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Domain (see page 20055)	Sets or returns the protocol family used by the socket.
R	Handle (see page 20056)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Protocol (see page 20056)	Sets or returns the protocol used by the socket.
	SockType (see page 20056)	Sets or returns the socket type.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

148.1.2.2 TBaseSocket.Create Constructor

Creates an instance of TBaseSocket (see page 20042).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBaseSocket(TComponent * AOwner);
```

Description

For C++, call Create indirectly, using the **new** keyword, to instantiate a TBaseSocket (see page 20042) object.

See Also

TBaseSocket (see page 20042)

148.1.2.3 TBaseSocket.Destroy Destructor

Frees the memory associated with the TBaseSocket (see page 20042) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBaseSocket();
```

Description

For C++, do not call ~TBaseSocket (see page 20042) directly. Instead, use the **delete** keyword on the object, which causes ~TBaseSocket (see page 20042) to be invoked automatically.

See Also

delete

TBaseSocket (see page 20042)

148.1.2.4 TBaseSocket Methods

The methods of the TBaseSocket class are listed here.

Protected Methods

	Name	Description
	DoCreateHandle (see page 20047)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 20047)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoHandleError (see page 20048)	This is DoHandleError, a member of class TBaseSocket.
V	DoReceive (see page 20048)	This is DoReceive, a member of class TBaseSocket.
V	DoSend (see page 20048)	This is DoSend, a member of class TBaseSocket.
V	ErrorCheck (see page 20048)	This is ErrorCheck, a member of class TBaseSocket.
V	Loaded (see page 20049)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 20053)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 20053)	This is SetBytesSent, a member of class TBaseSocket.

Public Methods

	Name	Description
	Close (see page 20047)	Shuts down the socket connection.
	MapDomain (see page 20049)	Maps the domain.
	MapSockType (see page 20050)	Maps the socket type.
	Open (see page 20050)	Creates the socket connection.
	PeekBuf (see page 20050)	Reads the buffer.
	ReceiveBuf (see page 20051)	Reads data buffers of to the socket.
	ReceiveIn (see page 20051)	Reads delimited lines of data from the socket.
	Select (see page 20051)	Tests for socket activity.
	SendBuf (see page 20052)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 20052)	Writes delimited lines of data to the socket.
	SendStream (see page 20053)	Writes a stream of data to the socket.
	WaitForData (see page 20053)	Waits for the socket connection to be ready to transfer data.

148.1.2.4.1 TBaseSocket.Close Method

Shuts down the socket connection.

Pascal

```
procedure Close; virtual;
```

C++

```
virtual __fastcall Close();
```

Description

Call Close to shut down the socket connection. Close sets the Active (see page 20054) property to **false**.

For client sockets, Close terminates the connection to the server. For server sockets, Close shuts down the connection so that the server socket is no longer listening for client requests.

See Also

Active (see page 20054)

Open (see page 20050)

148.1.2.4.2 TBaseSocket.DoCreateHandle Method

Pascal

```
procedure DoCreateHandle; dynamic;
```

C++

```
__fastcall DoCreateHandle();
```

Description

This is DoCreateHandle, a member of class TBaseSocket.

148.1.2.4.3 TBaseSocket.DoDestroyHandle Method

Pascal

```
procedure DoDestroyHandle; dynamic;
```

C++

```
__fastcall DoDestroyHandle();
```

Description

This is DoDestroyHandle, a member of class TBaseSocket.

148.1.2.4.4 TBaseSocket.DoHandleError Method

Pascal

```
procedure DoHandleError; dynamic;
```

C++

```
__fastcall DoHandleError();
```

Description

This is DoHandleError, a member of class TBaseSocket.

148.1.2.4.5 TBaseSocket.DoReceive Method

Pascal

```
procedure DoReceive(Buf: pchar; var DataLen: Integer); virtual;
```

C++

```
virtual __fastcall DoReceive(const char * Buf, int DataLen);
```

Description

This is DoReceive, a member of class TBaseSocket.

148.1.2.4.6 TBaseSocket.DoSend Method

Pascal

```
procedure DoSend(Buf: pchar; var DataLen: Integer); virtual;
```

C++

```
virtual __fastcall DoSend(const char * Buf, int DataLen);
```

Description

This is DoSend, a member of class TBaseSocket.

148.1.2.4.7 TBaseSocket.ErrorCheck Method

Pascal

```
function ErrorCheck(rc: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int ErrorCheck(int rc);
```

Description

This is ErrorCheck, a member of class TBaseSocket.

148.1.2.4.8 TBaseSocket.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState (see page 2804)

Create (see page 2766)

TReader (see page 2933)

148.1.2.4.9 TBaseSocket.MapDomain Method

Maps the domain.

Pascal

```
function MapDomain(sd: TSocketDomain): Integer;
```

C++

```
__fastcall int MapDomain(TSocketDomain sd);
```

Description

MapDomain maps the socket domain to an integer based on the value supplied by the Domain (see page 20055) property.

See Also

Domain (see page 20055)

MapSockType (see page 20050)

148.1.2.4.10 TBaseSocket.MapSockType Method

Maps the socket type.

Pascal

```
function MapSockType(st: TSocketType): Integer;
```

C++

```
__fastcall int MapSockType(TSocketType st);
```

Description

MapSockType maps the socket type to an integer based on the value supplied by the SockType (see page 20056) property.

See Also

SockType (see page 20056)

148.1.2.4.11 TBaseSocket.Open Method

Creates the socket connection.

Pascal

```
procedure Open; virtual;
```

C++

```
virtual __fastcall Open();
```

Description

Call Open to create the socket connection. Descendants of TBaseSocket (see page 20042) use Open to initiate the socket connection.

For client sockets, Open locates and connects to a server. For server sockets, Open opens the socket connection in a listening mode, but does not complete the connection to a client socket.

See Also

TBaseSocket (see page 20042)

Active (see page 20054)

Close (see page 20047)

148.1.2.4.12 TBaseSocket.PeekBuf Method

Reads the buffer.

Pascal

```
function PeekBuf(var Buf; BufSize: Integer): Integer;
```

C++

```
__fastcall int PeekBuf( Buf, int BufSize);
```

Description

Use PeekBuf to make a copy of data waiting in the socket buffer. Use the **Buf** parameter to assign a variable to the buffer, and use **Bufsize** to set the size of the buffer to be copied. PeekBuf only copies, but does not otherwise affect the contents of the buffer.

See Also

ReceiveBuf (see page 20051)

SendBuf (see page 20052)

148.1.2.4.13 TBaseSocket.ReceiveBuf Method

Reads data buffers of to the socket.

Pascal

```
function ReceiveBuf(var Buf; BufSize: Integer; Flags: Integer = 0): Integer;
```

C++

```
__fastcall int ReceiveBuf( Buf, int BufSize, int Flags = 0);
```

Description

ReceiveBuf reads data buffers of size **bufsize** to the socket. Use **buf** to assign a variable to the buffer, and use **bufsize** to set the size of the buffer to be read. The **flags** argument can be any of the flags used by the Linux **recv** function. Refer to recv(2) in the Linux manual pages for more information.

See Also

PeekBuf (see page 20050)

SendBuf (see page 20052)

148.1.2.4.14 TBaseSocket.ReceiveIn Method

Reads delimited lines of data from the socket.

Pascal

```
function ReceiveIn(const eol: string = CRLF): string;
```

C++

```
__fastcall AnsiString ReceiveIn(const AnsiString eol = CRLF);
```

Description

ReceiveIn reads a delimited chunk of data from the socket. The **delimeter** cannot be part of the character set. ReceiveIn fills the buffer until it reaches the **delimeter**.

See Also

SendIn (see page 20052)

OnReceive (see page 20058)

148.1.2.4.15 TBaseSocket.Select Method

Tests for socket activity.

Pascal

```
function Select(ReadReady: PBoolean; WriteReady: PBoolean; ExceptFlag: PBoolean; TimeOut:
Integer = 0): Boolean;
```

C++

```
__fastcall Boolean Select(PBoolean ReadReady, PBoolean WriteReady, PBoolean ExceptFlag, int
```

```
TimeOut = 0);
```

Description

Use Select to test for socket activity. Select indicates whether there is any data on the socket waiting to be sent or whether an error condition exists.

The parameter **TimeOut** is the time-out value. The socket waits for activity up to **TimeOut** milliseconds before returning. If **TimeOut** is not explicitly specified, the default value is zero. If **TimeOut** is any negative number, the socket is blocked. It waits for activity indefinitely, which can be useful for servers.

See Also

WaitForData (see page 20053)

148.1.2.4.16 TBaseSocket.SendBuf Method

Writes data buffers of size bufsize from the socket.

Pascal

```
function SendBuf(var Buf; BufSize: Integer; Flags: Integer = 0): Integer;
```

C++

```
__fastcall int SendBuf( Buf, int BufSize, int Flags = 0);
```

Description

SendBuf sends data buffers of size **bufsize** to the socket. Use **buf** to assign a variable to the buffer, and use **bufsize** to set the size of the buffer to be sent. The flags argument can be any of the flags used by the Linux **send** function. Refer to the send Linux manual page for more information.

See Also

PeekBuf (see page 20050)

ReceiveBuf (see page 20051)

148.1.2.4.17 TBaseSocket.SendIn Method

Writes delimited lines of data to the socket.

Pascal

```
function Sendln(s: string; const eol: string = CRLF): Integer;
```

C++

```
__fastcall int Sendln(AnsiString s, const AnsiString eol = CRLF);
```

Description

SendIn sends a delimited chunk of data to the socket. The **delimeter** cannot be part of the character set. SendIn fills the buffer until it reaches the **delimeter**.

See Also

ReceiveIn (see page 20051)

OnReceive (see page 20058)

148.1.2.4.18 TBaseSocket.SendStream Method

Writes a stream of data to the socket.

Pascal

```
function SendStream(AStream: TStream): Integer;
```

C++

```
__fastcall int SendStream(TStream AStream);
```

Description

SendStream sends a stream of data to the socket.

See Also

OnReceive (see page 20058)

148.1.2.4.19 TBaseSocket.SetBytesReceived Method

Pascal

```
procedure SetBytesReceived(Value: Cardinal);
```

C++

```
__fastcall SetBytesReceived(unsigned Value);
```

Description

This is SetBytesReceived, a member of class TBaseSocket.

148.1.2.4.20 TBaseSocket.SetBytesSent Method

Pascal

```
procedure SetBytesSent(Value: Cardinal);
```

C++

```
__fastcall SetBytesSent(unsigned Value);
```

Description

This is SetBytesSent, a member of class TBaseSocket.

148.1.2.4.21 TBaseSocket.WaitForData Method

Waits for the socket connection to be ready to transfer data.

Pascal

```
function WaitForData(TimeOut: Integer = 0): Boolean;
```

C++

```
__fastcall Boolean WaitForData(int TimeOut = 0);
```

Description

Call WaitForData to ensure that the socket connection is ready to read or write information. WaitForData returns **true** if the socket connection is ready. WaitForData returns **false** if the socket connection is not ready after **TimeOut** milliseconds elapse.

Call WaitForData before reading or writing information over the socket connection. Otherwise, ReceiveBuf (see page 20051),

ReceiveIn (see page 20051), SendBuf (see page 20052), and SendIn (see page 20052) method calls may time out before any data is transferred.

See Also

ReceiveBuf (see page 20051)

ReceiveIn (see page 20051)

SendBuf (see page 20052)

SendIn (see page 20052)

148.1.2.5 TBaseSocket Properties

The properties of the TBaseSocket class are listed here.

Public Properties

	Name	Description
	Active (see page 20054)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 20054)	Sets or returns the blocking mode of the socket.
R	BytesReceived (see page 20055)	Returns the number of bytes received by the socket.
R	BytesSent (see page 20055)	Returns the number of bytes sent from the socket.
	Domain (see page 20055)	Sets or returns the protocol family used by the socket.
R	Handle (see page 20056)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	Protocol (see page 20056)	Sets or returns the protocol used by the socket.
	SockType (see page 20056)	Sets or returns the socket type.

148.1.2.5.1 TBaseSocket.Active Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before attempting to use or change the socket connection, read Active to determine whether the connection is open and ready. For client sockets, setting Active opens or shuts down a socket connection to another machine. For server sockets, setting Active opens or shuts down a listening connection that makes the socket available for client requests.

At design time, set Active to **true** to make the socket open a connection when the application starts running (the default value is false). At runtime, use the Open (see page 20050) or Close (see page 20047) method to open or close the connection.

See Also

Close (see page 20047)

Open (see page 20050)

148.1.2.5.2 TBaseSocket.BlockMode Property

Sets or returns the blocking mode of the socket.

Pascal

```
property BlockMode: TSocketBlockMode;
```

C++

```
__property TSocketBlockMode BlockMode;
```

Description

Use BlockMode to set or return the blocking mode for the socket. By default, the socket is set to bmBlocking.

The TSocketBlockMode (see page 20141) type defines the set of possible socket flags for the BlockMode property.

See Also

TSocketBlockMode (see page 20141)

OnReceive (see page 20058)

OnSend (see page 20058)

148.1.2.5.3 TBaseSocket.BytesReceived Property

Returns the number of bytes received by the socket.

Pascal

```
property BytesReceived: Cardinal;
```

C++

```
__property unsigned BytesReceived;
```

Description

BytesReceived returns the number of bytes received by the socket.

See Also

BytesSent (see page 20055)

148.1.2.5.4 TBaseSocket.BytesSent Property

Returns the number of bytes sent from the socket.

Pascal

```
property BytesSent: Cardinal;
```

C++

```
__property unsigned BytesSent;
```

Description

BytesSent returns the number of bytes sent from the socket.

See Also

BytesReceived (see page 20055)

148.1.2.5.5 TBaseSocket.Domain Property

Sets or returns the protocol family used by the socket.

Pascal

```
property Domain: TSocketDomain;
```

C++

```
__property TSocketDomain Domain;
```

Description

Use Domain to set or return the protocol family used by the socket. The TSocketDomain (see page 20136) type defines the set of possible protocol family types for the Domain property. The default value of this property is pfUnspec.

See Also

MapDomain (see page 20049)

TSocketDomain (see page 20136)

148.1.2.5.6 TBaseSocket.Handle Property

Provides access to the unique-4 byte identifier that represents the instance of the socket object.

Pascal

```
property Handle: TSocket;
```

C++

```
__property TSocket Handle;
```

Description

Use Handle to return a typed pointer to the instance of the socket class. This Handle can be used when making low level function calls into the QT shared libraries, where the function call requires an instance pointer to the object.

148.1.2.5.7 TBaseSocket.Protocol Property

Sets or returns the protocol used by the socket.

Pascal

```
property Protocol: TSocketProtocol;
```

C++

```
__property TSocketProtocol Protocol;
```

Description

Protocol sets or returns the protocol used by the socket. The socket protocol can be IP, TCP, UDP, or other network protocols.

See Also

TTcpServer (see page 20115)

148.1.2.5.8 TBaseSocket.SockType Property

Sets or returns the socket type.

Pascal

```
property SockType: TSocketType;
```

C++

```
__property TSocketType SockType;
```

Description

SockType sets or returns the type of socket connection.

The TSocketType (see page 20138) type defines the set of possible socket types for the SockType property.

See Also

TSocketType (see page 20138)

148.1.2.6 TBaseSocket Events

The events of the TBaseSocket class are listed here.

Public Events

	Name	Description
	OnCreateHandle (see page 20057)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 20057)	Occurs when a socket is deactivated.
	OnError (see page 20058)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 20058)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 20058)	Occurs just before a client socket writes information to the socket connection.

148.1.2.6.1 TBaseSocket.OnCreateHandle Event

Occurs when the handle to a socket becomes active.

Pascal

```
property OnCreateHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnCreateHandle;
```

Description

Write an OnCreateHandle event handler to take specific action when the handle to a socket becomes active.

See Also

Active (see page 20054)

Handle (see page 20056)

148.1.2.6.2 TBaseSocket.OnDestroyHandle Event

Occurs when a socket is deactivated.

Pascal

```
property OnDestroyHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDestroyHandle;
```

Description

Write an OnDestroyHandle event handler to take specific action when the socket is deactivated.

148.1.2.6.3 TBaseSocket.OnError Event

Occurs when the socket fails in making, using, or shutting down a connection.

Pascal

```
property OnError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnError;
```

Description

Write an OnError event handler to respond to errors that arise with the socket connection. Set the **ErrorCode** parameter to 0 if the OnError event handler successfully deals with the error condition, to prevent an ESocketError (see page 20041) exception from being raised.

See Also

ESocketError (see page 20041)

148.1.2.6.4 TBaseSocket.OnReceive Event

Occurs after a socket receives data, but before it passes the data to the application.

Pascal

```
property OnReceive: TSocketDataEvent;
```

C++

```
__property TSocketDataEvent OnReceive;
```

Description

Write an OnReceive event handler to read from the socket connection. Use the OnReceive event to filter data before passing it back to the client application. If the socket is a blocking socket, use a TWinSocketStream object to read from the connection. Otherwise, use the methods of the **Socket** parameter to perform the actual reading.

See Also

TWinSocketStream (see page 18876)

BytesReceived (see page 20055)

ReceiveBuf (see page 20051)

ReceiveIn (see page 20051)

148.1.2.6.5 TBaseSocket.OnSend Event

Occurs just before a client socket writes information to the socket connection.

Pascal

```
property OnSend: TSocketDataEvent;
```

C++

```
__property TSocketDataEvent OnSend;
```

Description

Write an OnSend event handler to perform actions just prior to sending data. Use OnSend to monitor the data stream and

change it (for example, by removing linefeeds). If the socket is a blocking socket, use a TWinSocketStream object to write to the connection. Otherwise, use the methods of the **Socket** parameter to perform the actual writing.

See Also

TWinSocketStream (see page 18876)

SendBuf (see page 20052)

SendIn (see page 20052)

148.1.3 TClientSocketThread Class

Enables creation of separate threads of execution for the client socket in an application.

Class Hierarchy

Pascal

```
TClientSocketThread = class(TThread);
```

C++

```
class TClientSocketThread : public TThread;
```

File

Sockets

Description

Create (see page 20060) a TClientSocketThread (or a descendant of the class) to represent an execution thread for the client socket in a multi-threaded application. Each new instance of TClientSocketThread, a TThread descendant, is a new thread of execution. Using a separate TClientSocketThread allows the client socket to block without blocking the whole application.

See Also

Writing Multi-threaded Applications

TServerSocketThread (see page 20099)

TThread (see page 3108)

148.1.3.1 TClientSocketThread Members

The following tables list the members exposed by TClientSocketThread.

Protected Methods

	Name	Description
	CheckThreadError (see page 3114)	This is the overview for the CheckThreadError method overload.
V	DoTerminate (see page 3115)	Generates an OnTerminate (see page 3148) event.
V	SyncProc (see page 20062)	Method where data synchronization is actually performed.

Protected Properties

	Name	Description
	ReturnValue (see page 3141)	Specifies the value returned to other waiting threads when the thread finishes executing.

R	Terminated (see page 3142)	Indicates whether the thread has been asked to terminate.

Public Events

	Name	Description
	OnTerminate (see page 3148)	Occurs after the thread's Execute (see page 3115) method has returned and before the thread is destroyed.

Public Methods

	Name	Description
V	AfterConstruction (see page 3114)	Executes the thread when the thread is created with CreateSuspended set to false.
	Create (see page 20060)	Creates an instance of a TClientSocketThread (see page 20059) object.
V	Destroy (see page 20061)	Destroys the thread object and releases the memory allocated to it.
V	Execute (see page 20061)	Executes when the thread is run.
	ExecuteSyncProc (see page 20062)	Synchronizes data between threads.
	Queue (see page 3119)	This is the overview for the Queue method overload.
	RemoveQueuedEvents (see page 3120)	Remove queued method calls.
	Resume (see page 3120)	Restarts the execution of a suspended thread.
	StaticQueue (see page 3124)	Asynchronously executes a method call within the main thread.
	StaticSynchronize (see page 3125)	Executes a method call within the main thread.
	Suspend (see page 3125)	Pauses a running thread.
	Synchronize (see page 3126)	This is the overview for the Synchronize method overload.
	Terminate (see page 3126)	Signals the thread to terminate by setting the Terminated (see page 3142) property to true.
	WaitFor (see page 3132)	Waits for the thread to terminate and then returns the value of the ReturnValue (see page 3141) property.

Public Properties

	Name	Description
R	ClientSocket (see page 20063)	Refers to the client socket handling the connection in that thread.
R	FatalException (see page 3133)	Determines whether the thread object is automatically destroyed when the thread terminates.
	FreeOnTerminate (see page 3134)	Determines whether the thread object is automatically destroyed when the thread terminates.
R	Handle (see page 3137)	Indicates the thread's handle (Windows only).
	Priority (see page 3138)	Determines the thread's scheduling priority relative to other threads in the process.
R	ServerSocketThread (see page 20063)	Specifies the server socket's thread.
	Suspended (see page 3142)	Indicates whether a thread is suspended.
R	ThreadID (see page 3148)	Identifies the thread throughout the system.

148.1.3.2 TClientSocketThread.Create Constructor

Creates an instance of a TClientSocketThread (see page 20059) object.

Pascal

```
constructor Create(ServerSocketThread: TServerSocketThread);
```

C++

```
__fastcall TClientSocketThread(TServerSocketThread ServerSocketThread);
```

Description

For C++, call Create indirectly, using the **new** keyword, to create a thread in an application. If the **CreateSuspended** parameter is false, Execute (see page 20061) is called immediately. If **CreateSuspended** is true, Execute (see page 20061) is not called until after Resume is called.

See Also

TClientSocketThread (see page 20059)

Destroy (see page 20061)

Execute (see page 20061)

Resume (see page 3120)

148.1.3.3 TClientSocketThread.Destroy Destructor

Destroys the thread object and releases the memory allocated to it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TClientSocketThread();
```

Description

For C++, do not call Destroy in an application. Instead, use the **delete** keyword to destroy a thread when FreeOnTerminate is false.

See Also

delete

Create (see page 3110)

FreeOnTerminate (see page 3134)

148.1.3.4 TClientSocketThread Methods

The methods of the TClientSocketThread class are listed here.

Protected Methods

	Name	Description
	SyncProc (see page 20062)	Method where data synchronization is actually performed.

Public Methods

	Name	Description
	Execute (see page 20061)	Executes when the thread is run.
	ExecuteSyncProc (see page 20062)	Synchronizes data between threads.

148.1.3.4.1 TClientSocketThread.Execute Method

Executes when the thread is run.

Pascal

```
procedure Execute; override;
```

C++

```
virtual __fastcall Execute();
```

Description

Use Execute to execute the code that should be executed when the thread runs. Execute is responsible for checking the value of the Terminated property to determine if the thread needs to exit.

Note: Do not use the properties and methods of other objects directly in the Execute method of a thread. Instead, separate the use of other objects into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize method.

See Also

Terminated (see page 3142)

Synchronize (see page 3126)

148.1.3.4.2 TClientSocketThread.ExecuteSyncProc Method

Synchronizes data between threads.

Pascal

```
procedure ExecuteSyncProc;
```

C++

```
__fastcall ExecuteSyncProc();
```

Description

Objects in the client socket thread should not directly manipulate data in the main thread. Attempting to do so will cause synchronization problems and, most likely, the application will crash. To avoid synchronization errors, override the SyncProc (see page 20062) method to include instructions that must be executed by the main thread.

See Also

Synchronize (see page 3126)

SyncProc (see page 20062)

148.1.3.4.3 TClientSocketThread.SyncProc Method

Method where data synchronization is actually performed.

Pascal

```
procedure SyncProc; virtual;
```

C++

```
virtual __fastcall SyncProc();
```

Description

In subclasses of TClientSocketThread (see page 20059), override SyncProc to perform synchronization between threads. Put code in it to be executed by the main thread.

Warning: Do not call this protected method directly; instead, call ExecuteSyncProc (see page 20062).

See Also

TClientSocketThread (see page 20059)

ExecuteSyncProc (see page 20062)

Synchronize (see page 3126)

148.1.3.5 TClientSocketThread Properties

The properties of the TClientSocketThread class are listed here.

Public Properties

	Name	Description
R	ClientSocket (see page 20063)	Refers to the client socket handling the connection in that thread.
R	ServerSocketThread (see page 20063)	Specifies the server socket's thread.

148.1.3.5.1 TClientSocketThread.ClientSocket Property

Refers to the client socket handling the connection in that thread.

Pascal

```
property ClientSocket: TCustomIpClient;
```

C++

```
__property TCustomIpClient ClientSocket;
```

Description

Use ClientSocket to obtain access to the client socket object handling the connection in the current thread of execution.

See Also

TCustomIpClient (see page 20063)

148.1.3.5.2 TClientSocketThread.ServerSocketThread Property

Specifies the server socket's thread.

Pascal

```
property ServerSocketThread: TServerSocketThread;
```

C++

```
__property TServerSocketThread ServerSocketThread;
```

Description

Use ServerSocketThread get access to the thread object of the server socket.

See Also

TServerSocketThread (see page 20099)

148.1.4 TCustomIpClient Class

Encapsulates the client network functions.

Class Hierarchy

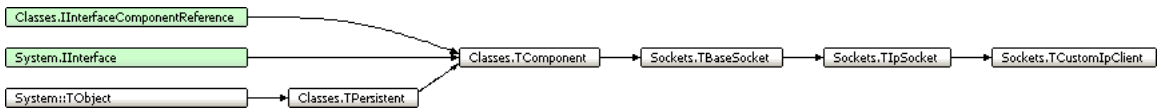

Pascal

```
TCustomIpClient = class(TIpSocket);
```

C++

```
class TCustomIpClient : public TIpSocket;
```

File

Sockets

Description

TCustomIpClient encapsulates the client network functions, linking sockets to network protocols such as TCP/IP and UDP, as well as raw sockets and custom protocols. Do not create instances of TCustomIpClient. Use TCustomIpClient as a base class when declaring non-visual components that can appear on the component palette and be used in the form designer. Properties and methods of TCustomIpClient provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TBaseSocket (see page 20042)

148.1.4.1 TCustomIpClient Members

The following tables list the members exposed by TCustomIpClient.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Bind (see page 20087)	This is Bind, a member of class TIpSocket.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoConnect (see page 20069)	This is DoConnect, a member of class TCustomIpClient.
	DoCreateHandle (see page 20047)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 20047)	This is DoDestroyHandle, a member of class TBaseSocket.
V	DoDisconnect (see page 20070)	This is DoDisconnect, a member of class TCustomIpClient.
	DoHandleError (see page 20048)	This is DoHandleError, a member of class TBaseSocket.
V	DoReceive (see page 20048)	This is DoReceive, a member of class TBaseSocket.
V	DoSend (see page 20048)	This is DoSend, a member of class TBaseSocket.
V	ErrorCheck (see page 20048)	This is ErrorCheck, a member of class TBaseSocket.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.

	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 20049)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBytesReceived (see page 20053)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 20053)	This is SetBytesSent, a member of class TBaseSocket.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnConnect (see page 20071)	Occurs on client sockets just after the connection to the server is opened.
	OnCreateHandle (see page 20057)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 20057)	Occurs when a socket is deactivated.
	OnDisconnect (see page 20072)	Occurs on client sockets just after the connection to the server is closed.
	OnError (see page 20058)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 20058)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 20058)	Occurs just before a client socket writes information to the socket connection.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Close (see page 20068)	Shuts down the socket connection.
	Connect (see page 20069)	Connects the local socket to the remote socket.
V	Create (see page 20068)	Creates an instance of TCustomIpClient (see page 20063).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 20046)	Frees the memory associated with the TBaseSocket (see page 20042) object.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Disconnect (see page 20069)	Disconnects the local socket from the remote socket.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetSocketAddr (see page 20087)	Builds a valid network socket address.
	GetThreadObject (see page 20070)	Returns a reference to the TThread object for the currently executing thread.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LocalDomainName (see page 20088)	Returns the domain name for the local host.
	LocalHostAddr (see page 20088)	Returns the local IP address for the host machine.
	LocalHostName (see page 20088)	Returns the local host name.
	LookupHostAddr (see page 20089)	Looks up the host address based on the host name.
	LookupHostName (see page 20089)	Looks up the host name based on the IP address.
	LookupPort (see page 20089)	Returns the port number.
	LookupProtocol (see page 20090)	Returns the protocol number.
	MapDomain (see page 20049)	Maps the domain.
	MapSockType (see page 20050)	Maps the socket type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Open (see page 20070)	Creates the socket connection.
	PeekBuf (see page 20050)	Reads the buffer.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReceiveBuf (see page 20051)	Reads data buffers of to the socket.
	ReceiveFrom (see page 20090)	Receives a packet of information.
	ReceiveIn (see page 20051)	Reads delimited lines of data from the socket.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Select (see page 20051)	Tests for socket activity.
	SendBuf (see page 20052)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 20052)	Writes delimited lines of data to the socket.
	SendStream (see page 20053)	Writes a stream of data to the socket.
	SendTo (see page 20091)	Sends a packet of information.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WaitForData (see page 20053)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
	Active (see page 20054)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 20054)	Sets or returns the blocking mode of the socket.
R	BytesReceived (see page 20055)	Returns the number of bytes received by the socket.
R	BytesSent (see page 20055)	Returns the number of bytes sent from the socket.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Connected (see page 20071)	Indicates whether the socket connection is open and available for communication with other machines.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Domain (see page 20055)	Sets or returns the protocol family used by the socket.
	Domain (see page 20091)	Sets or returns the protocol family used by the socket.
R	Handle (see page 20056)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	LocalHost (see page 20092)	Specifies the name of the local system.
	LocalPort (see page 20092)	Specifies the ID number of the port used by the socket connection.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Protocol (see page 20056)	Sets or returns the protocol used by the socket.
	RemoteHost (see page 20093)	Specifies the name of the remote system.
	RemotePort (see page 20093)	Specifies the ID number of the port used by the RemoteHost (see page 20093) to form the socket connection.
	SockType (see page 20056)	Sets or returns the socket type.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

148.1.4.2 TCustomIpClient.Create Constructor

Creates an instance of TCustomIpClient (see page 20063).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomIpClient(TComponent * AOwner);
```

Description

For C++, call Create indirectly, using the **new** keyword, to instantiate a TCustomIpClient (see page 20063) object.

See Also

Destroy

148.1.4.3 TCustomIpClient Methods

The methods of the TCustomIpClient class are listed here.

Protected Methods

	Name	Description
	DoConnect (see page 20069)	This is DoConnect, a member of class TCustomIpClient.
	DoDisconnect (see page 20070)	This is DoDisconnect, a member of class TCustomIpClient.

Public Methods

	Name	Description
	Close (see page 20068)	Shuts down the socket connection.
	Connect (see page 20069)	Connects the local socket to the remote socket.
	Disconnect (see page 20069)	Disconnects the local socket from the remote socket.
	GetThreadObject (see page 20070)	Returns a reference to the TThread object for the currently executing thread.
	Open (see page 20070)	Creates the socket connection.

148.1.4.3.1 TCustomIpClient.Close Method

Shuts down the socket connection.

Pascal

```
procedure Close; override;
```

C++

```
virtual __fastcall Close();
```

Description

Call Close to shut down the socket connection. This method performs the same functions as Disconnect (see page 20069).

See Also

Disconnect (see page 20069)

Open (see page 20070)

Active (see page 20054)

148.1.4.3.2 TCustomIpClient.Connect Method

Connects the local socket to the remote socket.

Pascal

```
function Connect: Boolean;
```

C++

```
__fastcall Boolean Connect();
```

Description

Call Connect to connect the local socket to the remote socket.

See Also

Connected (see page 20071)

Close (see page 20068)

Disconnect (see page 20069)

148.1.4.3.3 TCustomIpClient.Disconnect Method

Disconnects the local socket from the remote socket.

Pascal

```
procedure Disconnect;
```

C++

```
__fastcall Disconnect();
```

Description

Call Disconnect to break the connection between the local and remote sockets. This method performs the same function as Close (see page 20068).

See Also

Connected (see page 20071)

Close (see page 20068)

Connect (see page 20069)

148.1.4.3.4 TCustomIpClient.DoConnect Method

Pascal

```
procedure DoConnect; virtual;
```

C++

```
virtual __fastcall DoConnect();
```

Description

This is DoConnect, a member of class TCustomIpClient.

148.1.4.3.5 TCustomIpClient.DoDisconnect Method

Pascal

```
procedure DoDisconnect; virtual;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

This is DoDisconnect, a member of class TCustomIpClient.

148.1.4.3.6 TCustomIpClient.GetThreadObject Method

Returns a reference to the TThread object for the currently executing thread.

Pascal

```
function GetThreadObject: TClientSocketThread;
```

C++

```
__fastcall TClientSocketThread GetThreadObject();
```

Description

Call GetThreadObject to obtain a reference to the currently executing thread. GetThreadObject is only useful when BlockMode (see page 20054) is set to **bmBlockingMode**, which means that the client is executing in its own thread. If this method is called from the main application thread, the value NULL is returned.

See Also

TThread (see page 3108)

BlockMode (see page 20054)

148.1.4.3.7 TCustomIpClient.Open Method

Creates the socket connection.

Pascal

```
procedure Open; override;
```

C++

```
virtual __fastcall Open();
```

Description

Call Open to create and initiate the socket connection.

For client sockets, Open locates and connects to a server. For server sockets, Open opens the socket connection in a listening mode, but does not complete the connection to a client socket.

See Also

Active (see page 20054)

Close (see page 20068)

148.1.4.4 TCustomIpClient Properties

The properties of the TCustomIpClient class are listed here.

Public Properties

	Name	Description
R	Connected (see page 20071)	Indicates whether the socket connection is open and available for communication with other machines.

148.1.4.4.1 TCustomIpClient.Connected Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Read Connected before trying to use or change the socket connection. Connected indicates whether the socket connection has been established. When Connected is **true**, the socket is open and available for use. When Connected is **false**, the socket is closed and can be changed.

To open a socket, use the Open (see page 20070) method. To close a socket, use the Closemethod.

See Also

Close (see page 20068)

Open (see page 20070)

OnConnect (see page 20071)

148.1.4.5 TCustomIpClient Events

The events of the TCustomIpClient class are listed here.

Public Events

	Name	Description
	OnConnect (see page 20071)	Occurs on client sockets just after the connection to the server is opened.
	OnDisconnect (see page 20072)	Occurs on client sockets just after the connection to the server is closed.

148.1.4.5.1 TCustomIpClient.OnConnect Event

Occurs on client sockets just after the connection to the server is opened.

Pascal

```
property OnConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnect;
```

Description

Write an OnConnect event handler for a client socket to take specific action after the connection to a server socket has been established. Depending on the service, this may be the point when the socket should start reading or writing over the connection.

See Also

Connected (see page 20071)

OnDisconnect (see page 20072)

148.1.4.5.2 TCustomIpClient.OnDisconnect Event

Occurs on client sockets just after the connection to the server is closed.

Pascal

```
property OnDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDisconnect;
```

Description

Write an OnDisconnect event handler for a client socket to take specific action just after the connection to a server socket is terminated.

See Also

Connected (see page 20071)

OnConnect (see page 20071)

148.1.5 TCustomTcpServer Class

Ancestor object of all the server socket descendants.

Class Hierarchy

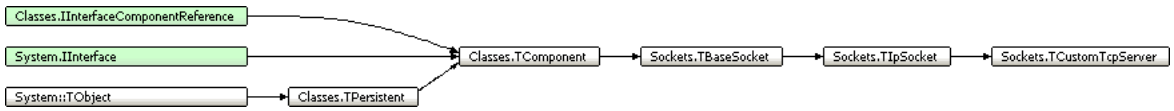

Pascal

```
TCustomTcpServer = class(TIpSocket);
```

C++

```
class TCustomTcpServer : public TIpSocket;
```

File

Sockets

Description

Do not create instances of TCustomTcpServer. Use TCustomTcpServer as a base class when declaring non-visual components that can appear on the component palette and be used in the form designer. Properties and methods of TCustomTcpServer provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

148.1.5.1 TCustomTcpServer Members

The following tables list the members exposed by TCustomTcpServer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Bind (see page 20087)	This is Bind, a member of class TIpSocket.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAccept (see page 20078)	This is DoAccept, a member of class TCustomTcpServer.
	DoCreateHandle (see page 20047)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 20047)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoHandleError (see page 20048)	This is DoHandleError, a member of class TBaseSocket.
V	DoReceive (see page 20048)	This is DoReceive, a member of class TBaseSocket.
V	DoSend (see page 20048)	This is DoSend, a member of class TBaseSocket.
V	ErrorCheck (see page 20048)	This is ErrorCheck, a member of class TBaseSocket.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Listen (see page 20079)	This is Listen, a member of class TCustomTcpServer.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 20049)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBytesReceived (see page 20053)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 20053)	This is SetBytesSent, a member of class TBaseSocket.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property

	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnAccept (see page 20081)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnCreateHandle (see page 20057)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 20057)	Occurs when a socket is deactivated.
	OnError (see page 20058)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnGetThread (see page 20081)	A wrapper for the TServerSocketThread (see page 20099) event of the same name.
	OnListening (see page 20082)	Occurs when a server socket begins listening for connections.
	OnReceive (see page 20058)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 20058)	Occurs just before a client socket writes information to the socket connection.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Accept (see page 20077)	This is the overview for the Accept method overload.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 20078)	Shuts down the socket connection.
	Create (see page 20076)	Creates an instance of TCustomTcpServer (see page 20072).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 20077)	Frees the memory associated with the TCustomTcpServer (see page 20072) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetSocketAddr (see page 20087)	Builds a valid network socket address.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LocalDomainName (see page 20088)	Returns the domain name for the local host.
	LocalHostAddr (see page 20088)	Returns the local IP address for the host machine.
	LocalHostName (see page 20088)	Returns the local host name.
	LookupHostAddr (see page 20089)	Looks up the host address based on the host name.
	LookupHostName (see page 20089)	Looks up the host name based on the IP address.
	LookupPort (see page 20089)	Returns the port number.
	LookupProtocol (see page 20090)	Returns the protocol number.
	MapDomain (see page 20049)	Maps the domain.
	MapSockType (see page 20050)	Maps the socket type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Open (see page 20079)	Opens the socket connection.
	PeekBuf (see page 20050)	Reads the buffer.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReceiveBuf (see page 20051)	Reads data buffers of to the socket.
	ReceiveFrom (see page 20090)	Receives a packet of information.
	ReceiveIn (see page 20051)	Reads delimited lines of data from the socket.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Select (see page 20051)	Tests for socket activity.
	SendBuf (see page 20052)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 20052)	Writes delimited lines of data to the socket.
	SendStream (see page 20053)	Writes a stream of data to the socket.
	SendTo (see page 20091)	Sends a packet of information.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

	WaitForConnection (see page 20079)	Listens for TCP connections.
	WaitForData (see page 20053)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
	Active (see page 20054)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 20080)	Sets or returns the blocking mode of the socket.
R	BytesReceived (see page 20055)	Returns the number of bytes received by the socket.
R	BytesSent (see page 20055)	Returns the number of bytes sent from the socket.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Domain (see page 20055)	Sets or returns the protocol family used by the socket.
	Domain (see page 20091)	Sets or returns the protocol family used by the socket.
R	Handle (see page 20056)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
R	Listening (see page 20080)	Sets the server application to listen for client requests.
	LocalHost (see page 20092)	Specifies the name of the local system.
	LocalPort (see page 20092)	Specifies the ID number of the port used by the socket connection.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Protocol (see page 20056)	Sets or returns the protocol used by the socket.
	RemoteHost (see page 20093)	Specifies the name of the remote system.
	RemotePort (see page 20093)	Specifies the ID number of the port used by the RemoteHost (see page 20093) to form the socket connection.
	ServerSocketThread (see page 20080)	Specifies the TServerSocketThread (see page 20099) in use.
	SockType (see page 20056)	Sets or returns the socket type.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

148.1.5.2 TCustomTcpServer.Create Constructor

Creates an instance of TCustomTcpServer (see page 20072).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomTcpServer(TComponent * AOwner);
```

Description

For C++, call Create indirectly, using the **new** keyword, to instantiate a TCustomTcpServer (see page 20072) object.

See Also

TCustomTcpServer (see page 20072)

Destroy (see page 20077)

148.1.5.3 TCustomTcpServer.Destroy Destructor

Frees the memory associated with the TCustomTcpServer (see page 20072) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomTcpServer();
```

Description

For C++, do not call Destroy directly. Instead, use the **delete** keyword on the object, which causes Destroy to be invoked automatically.

See Also

delete

148.1.5.4 TCustomTcpServer Methods

The methods of the TCustomTcpServer class are listed here.

Protected Methods

	Name	Description
	DoAccept (see page 20078)	This is DoAccept, a member of class TCustomTcpServer.
	Listen (see page 20079)	This is Listen, a member of class TCustomTcpServer.

Public Methods

	Name	Description
	Accept (see page 20077)	This is the overview for the Accept method overload.
	Close (see page 20078)	Shuts down the socket connection.
	Open (see page 20079)	Opens the socket connection.
	WaitForConnection (see page 20079)	Listens for TCP connections.

148.1.5.4.1 Accept Method

This is the overview for the Accept method overload.

Overload List

	Name	Description
	TCustomTcpServer.Accept () (see page 20077)	Accepts a connection to a client socket.
	TCustomTcpServer.Accept (TCustomIpClient) (see page 20077)	Accepts a connection to a client socket.

148.1.5.4.1.1 TCustomTcpServer.Accept Method ()

Accepts a connection to a client socket.

Pascal

```
function Accept: Boolean; overload;
function Accept(var ClientSocket: TCustomIpClient): Boolean; overload;
```

C++

```
__fastcall Boolean Accept();
__fastcall Boolean Accept(TCustomIpClient ClientSocket);
```

Description

Accept is called automatically in response to notifications that a client socket is requesting a connection.

The Accept method for TCustomTcpServerdoes nothing with the notification. Descendants of TCustomTcpServer (see page 20072) that represent listening server sockets override this method to accept the connection to the client socket. In the second declaration of the method, a TCustomIpClient (see page 20063) object is created (as the **ClientSocket** parameter) to handle the connection.

See Also

TCustomIpClient (see page 20063)

TCustomTcpServer (see page 20072)

OnAccept (see page 20081)

148.1.5.4.2 TCustomTcpServer.Close Method

Shuts down the socket connection.

Pascal

```
procedure Close; override;
```

C++

```
virtual __fastcall Close();
```

Description

Call Close to shut down the socket connection. Close sets the Active (see page 20054) property to false.

For client sockets, Close terminates the connection to the server. For server sockets, Close shuts down the connection so that the server socket is no longer listening for client requests.

See Also

Active (see page 20054)

Open (see page 20079)

148.1.5.4.3 TCustomTcpServer.DoAccept Method

Pascal

```
procedure DoAccept(ClientSocket: TCustomIpClient); virtual;
```

C++

```
virtual __fastcall DoAccept(TCustomIpClient ClientSocket);
```

Description

This is DoAccept, a member of class TCustomTcpServer.

148.1.5.4.4 TCustomTcpServer.Listen Method

Pascal

```
function Listen(backlog: Integer = SOMAXCONN): Boolean;
```

C++

```
__fastcall Boolean Listen(int backlog = SOMAXCONN);
```

Description

This is Listen, a member of class TCustomTcpServer.

148.1.5.4.5 TCustomTcpServer.Open Method

Opens the socket connection.

Pascal

```
procedure Open; override;
```

C++

```
virtual __fastcall Open();
```

Description

Call Open to initiate the socket connection. Open sets the Active (see page 20054) property to true.

For client sockets, Open locates and connects to a server. For server sockets, Open opens the socket connection in a listening mode, but does not complete the connection to a client socket.

See Also

Active (see page 20054)

Close (see page 20078)

148.1.5.4.6 TCustomTcpServer.WaitForConnection Method

Listens for TCP connections.

Pascal

```
function WaitForConnection: Boolean;
```

C++

```
__fastcall Boolean WaitForConnection();
```

Description

Call WaitForConnection to listen for TCP connections on the port specified by the LocalPort (see page 20092) property. The thread will block until a connection request is received.

See Also

LocalPort (see page 20092)

Accept (see page 20077)

148.1.5.5 TCustomTcpServer Properties

The properties of the TCustomTcpServer class are listed here.

Public Properties

	Name	Description
	BlockMode (see page 20080)	Sets or returns the blocking mode of the socket.
R	Listening (see page 20080)	Sets the server application to listen for client requests.
	ServerSocketThread (see page 20080)	Specifies the TServerSocketThread (see page 20099) in use.

148.1.5.5.1 TCustomTcpServer.BlockMode Property

Sets or returns the blocking mode of the socket.

Pascal

```
property BlockMode: TServerSocketBlockMode;
```

C++

```
__property TServerSocketBlockMode BlockMode;
```

Description

Use BlockMode to set or return the blocking mode for the socket. By default, the socket is set to bmThreadBlocking.

The TServerSocketBlockMode (see page 20136) type defines the set of possible socket flags for the BlockMode property.

See Also

OnReceive (see page 20058)

OnSend (see page 20058)

TServerSocketBlockMode (see page 20136)

148.1.5.5.2 TCustomTcpServer.Listening Property

Sets the server application to listen for client requests.

Pascal

```
property Listening: Boolean;
```

C++

```
__property Boolean Listening;
```

Description

Listening sets the server application to listen for client requests.

148.1.5.5.3 TCustomTcpServer.ServerSocketThread Property

Specifies the TServerSocketThread (see page 20099) in use.

Pascal

```
property ServerSocketThread: TServerSocketThread;
```

C++

```
__property TServerSocketThread ServerSocketThread;
```

Description

ServerSocketThread is a reference to the server socket thread object.

See Also

TServerSocketThread (see page 20099)

Listening (see page 20080)

148.1.5.6 TCustomTcpServer Events

The events of the TCustomTcpServer class are listed here.

Public Events

	Name	Description
	OnAccept (see page 20081)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnGetThread (see page 20081)	A wrapper for the TServerSocketThread (see page 20099) event of the same name.
	OnListening (see page 20082)	Occurs when a server socket begins listening for connections.

148.1.5.6.1 TCustomTcpServer.OnAccept Event

Occurs on server sockets just after the connection to a client socket is accepted.

Pascal

```
property OnAccept: TSocketAcceptEvent;
```

C++

```
__property TSocketAcceptEvent OnAccept;
```

Description

Write an OnAccept event handler for a server socket to take specific action after the connection to a client socket has been accepted. This is the first point when information is available about the local port and IP address that is used in the server endpoint of the connection. The port number, and possibly the IP address, differs from those of the listening socket.

Server sockets open a socket connection for listening to establish a queue to hold client requests. After a request from the queue is accepted by the server socket, an OnAccept event occurs. After the server socket accepts the connection, the client socket completes the connection, and an OnConnect event occurs for the client socket.

See Also

Active

OnConnect (see page 20071)

OnListening

148.1.5.6.2 TCustomTcpServer.OnGetThread Event

A wrapper for the TServerSocketThread (see page 20099) event of the same name.

Pascal

```
property OnGetThread: TGetThreadEvent;
```

C++

```
__property TGetThreadEvent OnGetThread;
```

Description

This event is exactly the same as the OnGetThread event of the TServerSocketThread (see page 20099) object referenced in ServerSocketThread (see page 20080).

See Also

TServerSocketThread (see page 20099)

ServerSocketThread (see page 20063)

OnGetThread (see page 20105)

148.1.5.6.3 TCustomTcpServer.OnListening Event

Occurs when a server socket begins listening for connections.

Pascal

```
property OnListening: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnListening;
```

Description

Write an OnListening event to take specific action when a server begins listening for a connection.

See Also

Listening (see page 20080)

TNotifyEvent (see page 3298)

148.1.6 TIpSocket Class

Encapsulates the network support functions.

Class Hierarchy

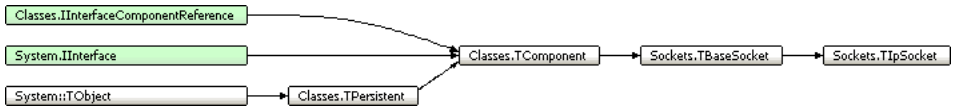

Pascal

```
TIpSocket = class(TBaseSocket);
```

C++

```
class TIpSocket : public TBaseSocket;
```

File

Sockets

Description

TIpSocket encapsulates the network support functions, linking sockets to network protocols such as TCP/IP and UDP, as well as raw sockets and custom protocols.

Do not create instances of TIpSocket. Use TIpSocket as a base class when declaring non-visual components that can appear on

the component palette and be used in the form designer. Properties and methods of TIpSocket provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TBaseSocket (see page 20042)

148.1.6.1 TIpSocket Members

The following tables list the members exposed by TIpSocket.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Bind (see page 20087)	This is Bind, a member of class TIpSocket.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoCreateHandle (see page 20047)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 20047)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoHandleError (see page 20048)	This is DoHandleError, a member of class TBaseSocket.
V	DoReceive (see page 20048)	This is DoReceive, a member of class TBaseSocket.
V	DoSend (see page 20048)	This is DoSend, a member of class TBaseSocket.
V	ErrorCheck (see page 20048)	This is ErrorCheck, a member of class TBaseSocket.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 20049)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBytesReceived (see page 20053)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 20053)	This is SetBytesSent, a member of class TBaseSocket.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnCreateHandle (see page 20057)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 20057)	Occurs when a socket is deactivated.
	OnError (see page 20058)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 20058)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 20058)	Occurs just before a client socket writes information to the socket connection.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 20047)	Shuts down the socket connection.
	Create (see page 20086)	Creates an instance of TIpSocket (see page 20082).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20046)	Frees the memory associated with the TBaseSocket (see page 20042) object.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetSocketAddr (see page 20087)	Builds a valid network socket address.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LocalDomainName (see page 20088)	Returns the domain name for the local host.
	LocalHostAddr (see page 20088)	Returns the local IP address for the host machine.
	LocalHostName (see page 20088)	Returns the local host name.
	LookupHostAddr (see page 20089)	Looks up the host address based on the host name.
	LookupHostName (see page 20089)	Looks up the host name based on the IP address.
	LookupPort (see page 20089)	Returns the port number.
	LookupProtocol (see page 20090)	Returns the protocol number.
	MapDomain (see page 20049)	Maps the domain.
	MapSockType (see page 20050)	Maps the socket type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Open (see page 20050)	Creates the socket connection.
	PeekBuf (see page 20050)	Reads the buffer.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReceiveBuf (see page 20051)	Reads data buffers of to the socket.
	ReceiveFrom (see page 20090)	Receives a packet of information.
	ReceiveIn (see page 20051)	Reads delimited lines of data from the socket.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Select (see page 20051)	Tests for socket activity.
	SendBuf (see page 20052)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 20052)	Writes delimited lines of data to the socket.
	SendStream (see page 20053)	Writes a stream of data to the socket.
	SendTo (see page 20091)	Sends a packet of information.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WaitForData (see page 20053)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
	Active (see page 20054)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 20054)	Sets or returns the blocking mode of the socket.
R	BytesReceived (see page 20055)	Returns the number of bytes received by the socket.
R	BytesSent (see page 20055)	Returns the number of bytes sent from the socket.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Domain (see page 20091)	Sets or returns the protocol family used by the socket.
R	Handle (see page 20056)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	LocalHost (see page 20092)	Specifies the name of the local system.
	LocalPort (see page 20092)	Specifies the ID number of the port used by the socket connection.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Protocol (see page 20056)	Sets or returns the protocol used by the socket.
	RemoteHost (see page 20093)	Specifies the name of the remote system.
	RemotePort (see page 20093)	Specifies the ID number of the port used by the RemoteHost (see page 20093) to form the socket connection.
	SockType (see page 20056)	Sets or returns the socket type.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

148.1.6.2 TIpSocket.Create Constructor

Creates an instance of TIpSocket (see page 20082).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TIpSocket(TComponent * AOwner);
```

Description

For C++, call Create indirectly, using the **new** keyword, to instantiate a TIpSocket (see page 20082) object.

See Also

Destroy

148.1.6.3 TIpSocket Methods

The methods of the TIpSocket class are listed here.

Protected Methods

	Name	Description
	Bind (see page 20087)	This is Bind, a member of class TIpSocket.

Public Methods

	Name	Description
	GetSocketAddr (see page 20087)	Builds a valid network socket address.
	LocalDomainName (see page 20088)	Returns the domain name for the local host.
	LocalHostAddr (see page 20088)	Returns the local IP address for the host machine.
	LocalHostName (see page 20088)	Returns the local host name.
	LookupHostAddr (see page 20089)	Looks up the host address based on the host name.
	LookupHostName (see page 20089)	Looks up the host name based on the IP address.
	LookupPort (see page 20089)	Returns the port number.
	LookupProtocol (see page 20090)	Returns the protocol number.
	ReceiveFrom (see page 20090)	Receives a packet of information.
	SendTo (see page 20091)	Sends a packet of information.

148.1.6.3.1 TIpSocket.Bind Method

Pascal

```
function Bind: Boolean;
```

C++

```
__fastcall Boolean Bind();
```

Description

This is Bind, a member of class TIpSocket.

148.1.6.3.2 TIpSocket.GetSocketAddr Method

Builds a valid network socket address.

Pascal

```
function GetSocketAddr(h: TSocketHost; p: TSocketPort): TSockAddr;
```

C++

```
__fastcall TSockAddr GetSocketAddr(TSocketHost h, TSocketPort p);
```

Description

Use GetSocketAddr to build a valid socket address based on the host name and port number. The **TSocketHost (see page 20142)** and **TSocketPort (see page 20142)** arguments are used to return a TSockAddr structure, which can then be passed to Connect or Bind (see page 20087).

See Also

Bind (see page 20087)

Connect (see page 20069)

LocalHostAddr (see page 20088)

LookupHostAddr (see page 20089)

RemoteHost (see page 20093)

RemotePort (see page 20093)

148.1.6.3.3 TIpSocket.LocalDomainName Method

Returns the domain name for the local host.

Pascal

```
function LocalDomainName: string;
```

C++

```
__fastcall AnsiString LocalDomainName();
```

Description

Use LocalDomainName name to return the domain name for the local host.

See Also

LocalHostAddr (see page 20088)

LocalHostName (see page 20088)

148.1.6.3.4 TIpSocket.LocalHostAddr Method

Returns the local IP address for the host machine.

Pascal

```
function LocalHostAddr: TSocketHost;
```

C++

```
__fastcall TSocketHost LocalHostAddr();
```

Description

Use LocalHostAddr to return the local IP address in string format.

See Also

LocalHost (see page 20092)

LocalDomainName (see page 20088)

LocalHostName (see page 20088)

148.1.6.3.5 TIpSocket.LocalHostName Method

Returns the local host name.

Pascal

```
function LocalHostName: TSocketHost;
```

C++

```
__fastcall TSocketHost LocalHostName();
```

Description

Use LocalHostName to return the host name for the local host. For example:

```
ShowMessage(TcpClient1.LocalHostName);
```

See Also

LocalHost (see page 20092)

LocalDomainName (see page 20088)

LocalHostAddr (see page 20088)

148.1.6.3.6 TIpSocket.LookupHostAddr Method

Looks up the host address based on the host name.

Pascal

```
function LookupHostAddr(const hn: string): TSocketHost;
```

C++

```
__fastcall TSocketHost LookupHostAddr(const AnsiString hn);
```

Description

Use LookupHostAddr to look up the host address based on the machine's host name. For example:

```
StrToInt(TcpClient1.LookupHostAddr("borland.com"));
```

See Also

RemoteHost (see page 20093)

LookupHostName (see page 20089)

LookupPort (see page 20089)

148.1.6.3.7 TIpSocket.LookupHostName Method

Looks up the host name based on the IP address.

Pascal

```
function LookupHostName(const ipaddr: string): TSocketHost;
```

C++

```
__fastcall TSocketHost LookupHostName(const AnsiString ipaddr);
```

Description

Use LookupHostName to look up the host name based on the machine's IP address. For example:

```
TcpClient1.LookupHostName("123.60.192.10");
```

See Also

RemoteHost (see page 20093)

LookupHostAddr (see page 20089)

LookupPort (see page 20089)

148.1.6.3.8 TIpSocket.LookupPort Method

Returns the port number.

Pascal

```
function LookupPort(const sn: string; pn: pchar = nil): word;
```

C++

```
__fastcall word LookupPort(const AnsiString sn, const char * pn = nil);
```

Description

Use LookupPort to return the port number of the IP socket. The port number is related to the service name and protocol number.

For more information, refer to the home page of the Internet Assigned Numbers Authority (www.iana.org), which maintains lists of port numbers and their associated protocols. Also, refer to the documentation for the application's target operating system for information on how port numbers are stored and changed in that OS.

See Also

RemotePort (see page 20093)

LookupProtocol (see page 20090)

148.1.6.3.9 TIpSocket.LookupProtocol Method

Returns the protocol number.

Pascal

```
function LookupProtocol(const pn: string): TSocketProtocol;
```

C++

```
__fastcall TSocketProtocol LookupProtocol(const AnsiString pn);
```

Description

Use LookupProtocol to return the protocol number being used by the IP socket. The protocol number is based on the protocol name.

For more information, refer to the home page of the Internet Assigned Numbers Authority (www.iana.org), which maintains lists of protocol numbers and their associated protocols.

See Also

RemotePort (see page 20093)

LookupPort (see page 20089)

148.1.6.3.10 TIpSocket.ReceiveFrom Method

Receives a packet of information.

Pascal

```
function ReceiveFrom(var buf; bufsize: Integer; ToAddr: TSockAddr; var len: Integer; flags:
Integer = 0): Integer;
```

C++

```
__fastcall int ReceiveFrom( buf, int bufsize, TSockAddr ToAddr, int len, int flags = 0);
```

Description

Use ReceiveFrom to receive a single packet of information. Note that information in the packet is not checked for length, completeness, and such, which occurs at a higher level.

See Also

SendTo (see page 20091)

OnReceive (see page 20058)

148.1.6.3.11 TIpSocket.SendTo Method

Sends a packet of information.

Pascal

```
function SendTo(var buf; bufsize: Integer; ToAddr: TSockAddr; flags: Integer = 0): Integer;
```

C++

```
__fastcall int SendTo( buf, int bufsize, TSockAddr ToAddr, int flags = 0);
```

Description

Use SendTo to send a single packet of information over the network. Note that information in the packet is not checked for length, completeness, and such, which occurs at a higher level.

See Also

ReceiveFrom (see page 20090)

OnSend (see page 20058)

148.1.6.4 TIpSocket Properties

The properties of the TIpSocket class are listed here.

Public Properties

	Name	Description
	Domain (see page 20091)	Sets or returns the protocol family used by the socket.
	LocalHost (see page 20092)	Specifies the name of the local system.
	LocalPort (see page 20092)	Specifies the ID number of the port used by the socket connection.
	RemoteHost (see page 20093)	Specifies the name of the remote system.
	RemotePort (see page 20093)	Specifies the ID number of the port used by the RemoteHost (see page 20093) to form the socket connection.

148.1.6.4.1 TIpSocket.Domain Property

Sets or returns the protocol family used by the socket.

Pascal

```
property Domain: TSocketDomain;
```

C++

```
__property TSocketDomain Domain;
```

Description

Use Domain to set or return the protocol family used by the socket. The TSocketDomain type defines the set of possible protocol family types for the Domain property. The default value of this property is pfUnspec.

See Also

MapDomain (see page 20049)

TSocketDomain (see page 20136)

148.1.6.4.2 TIpSocket.LocalHost Property

Specifies the name of the local system.

Pascal

```
property LocalHost: TSocketHost;
```

C++

```
__property TSocketHost LocalHost;
```

Description

Use LocalHost to get the name of the system that is running the application that uses the socket. LocalHost is a string containing the domain name and service of the local socket endpoint, such as http://www.wSite.Com.

Port numbers allow a single system, identified by the LocalHostAddr (see page 20088) function, to host multiple connections simultaneously. Each combination of LocalPort (see page 20092) and LocalHostAddr (see page 20088) can only be bound to a single socket connection.

LocalHost provides an alias for the local IP address given in the LocalHostAddr (see page 20088) function. Most Intranets provide host names for the IP addresses of systems on the net. For more information on defining a host name for a given system, contact your network administrator.

See Also

LocalPort (see page 20092)

RemoteHost (see page 20093)

LocalHostAddr (see page 20088)

LookupHostName (see page 20089)

LookupPort (see page 20089)

148.1.6.4.3 TIpSocket.LocalPort Property

Specifies the ID number of the port used by the socket connection.

Pascal

```
property LocalPort: TSocketPort;
```

C++

```
__property TSocketPort LocalPort;
```

Description

Use LocalPort to determine the port number that is bound to the socket connection. For server endpoints of listening connections, LocalPort is the ID associated with the service the server socket provides. For client or server endpoints of connections to another socket, LocalPort is usually an arbitrary value picked when the connection was made.

Many values of LocalPort are associated by convention with a particular service such as FTP or HTTP.

See Also

LocalHost (see page 20092)

RemotePort (see page 20093)

LookupPort (see page 20089)

148.1.6.4.4 TIpSocket.RemoteHost Property

Specifies the name of the remote system.

Pascal

```
property RemoteHost: TSocketHost;
```

C++

```
__property TSocketHost RemoteHost;
```

Description

Use RemoteHost to get the name of the system on the other end of the socket connection. RemoteHost is a string containing the domain name and service of the remote endpoint of the socket connection, such as http://www.wSite.Com.

See Also

RemotePort (see page 20093)

LocalHostAddr (see page 20088)

LocalHostName (see page 20088)

LookupHostAddr (see page 20089)

LookupHostName (see page 20089)

148.1.6.4.5 TIpSocket.RemotePort Property

Specifies the ID number of the port used by the RemoteHost (see page 20093) to form the socket connection.

Pascal

```
property RemotePort: TSocketPort;
```

C++

```
__property TSocketPort RemotePort;
```

Description

Use RemotePort to determine the port number used by the socket on the other end of the socket connection.

Port numbers allow a single system, identified by the RemoteHost (see page 20093) property, to host multiple connections simultaneously. Each combination of RemotePort and RemoteHost (see page 20093) can only be bound to a single socket connection.

Many port numbers are associated by convention with a particular service such as ftp or http. To determine whether the value of RemotePort is associated with a specific service, use the LookupProtocol (see page 20090) method.

Note: If a client socket requests a specific port to indicate a desired service, the value of RemotePort for the actual connection may differ from the requested port number. Most server sockets listen on the port associated with a service, but switch to an arbitrary available port number for forming connections.

See Also

RemoteHost (see page 20093)

LookupPort (see page 20089)

LookupProtocol (see page 20090)

148.1.7 TRawSocket Class

Encapsulates a raw socket.

Class Hierarchy

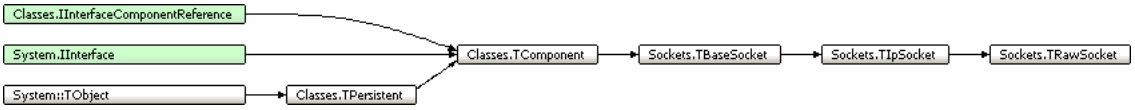

Pascal

```
TRawSocket = class(TIpSocket);
```

C++

```
class TRawSocket : public TIpSocket;
```

File

Sockets

Description

TRawSocket encapsulates a raw socket.

See Also

TBaseSocket (see page 20042)

148.1.7.1 TRawSocket Members

The following tables list the members exposed by TRawSocket.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Bind (see page 20087)	This is Bind, a member of class TIpSocket.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoCreateHandle (see page 20047)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 20047)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoHandleError (see page 20048)	This is DoHandleError, a member of class TBaseSocket.
V	DoReceive (see page 20048)	This is DoReceive, a member of class TBaseSocket.
V	DoSend (see page 20048)	This is DoSend, a member of class TBaseSocket.
V	ErrorCheck (see page 20048)	This is ErrorCheck, a member of class TBaseSocket.

	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 20049)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBytesReceived (see page 20053)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 20053)	This is SetBytesSent, a member of class TBaseSocket.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnCreateHandle (see page 20057)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 20057)	Occurs when a socket is deactivated.
	OnError (see page 20058)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 20058)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 20058)	Occurs just before a client socket writes information to the socket connection.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.

	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 20047)	Shuts down the socket connection.
	Create (see page 20098)	Creates an instance of a TRawSocket (see page 20094) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20046)	Frees the memory associated with the TBaseSocket (see page 20042) object.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetSocketAddr (see page 20087)	Builds a valid network socket address.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LocalDomainName (see page 20088)	Returns the domain name for the local host.
	LocalHostAddr (see page 20088)	Returns the local IP address for the host machine.
	LocalHostName (see page 20088)	Returns the local host name.
	LookupHostAddr (see page 20089)	Looks up the host address based on the host name.
	LookupHostName (see page 20089)	Looks up the host name based on the IP address.
	LookupPort (see page 20089)	Returns the port number.
	LookupProtocol (see page 20090)	Returns the protocol number.
	MapDomain (see page 20049)	Maps the domain.
	MapSockType (see page 20050)	Maps the socket type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Open (see page 20050)	Creates the socket connection.
	PeekBuf (see page 20050)	Reads the buffer.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReceiveBuf (see page 20051)	Reads data buffers of to the socket.
	ReceiveFrom (see page 20090)	Receives a packet of information.
	ReceiveIn (see page 20051)	Reads delimited lines of data from the socket.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Select (see page 20051)	Tests for socket activity.
	SendBuf (see page 20052)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 20052)	Writes delimited lines of data to the socket.
	SendStream (see page 20053)	Writes a stream of data to the socket.
	SendTo (see page 20091)	Sends a packet of information.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WaitForData (see page 20053)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
	Active (see page 20054)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 20054)	Sets or returns the blocking mode of the socket.
R	BytesReceived (see page 20055)	Returns the number of bytes received by the socket.
R	BytesSent (see page 20055)	Returns the number of bytes sent from the socket.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Domain (see page 20055)	Sets or returns the protocol family used by the socket.
	Domain (see page 20091)	Sets or returns the protocol family used by the socket.
R	Handle (see page 20056)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	LocalHost (see page 20092)	Specifies the name of the local system.
	LocalPort (see page 20092)	Specifies the ID number of the port used by the socket connection.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Protocol (see page 20056)	Sets or returns the protocol used by the socket.
	RemoteHost (see page 20093)	Specifies the name of the remote system.
	RemotePort (see page 20093)	Specifies the ID number of the port used by the RemoteHost (see page 20093) to form the socket connection.
	SockType (see page 20098)	Sets or returns the socket type.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

148.1.7.2 TRawSocket.Create Constructor

Creates an instance of a TRawSocket (see page 20094) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TRawSocket(TComponent * AOwner);
```

Description

Call Create indirectly, using the **new** keyword, to instantiate a TRawSocket (see page 20094) object.

See Also

TRawSocket (see page 20094)

Destroy

148.1.7.3 TRawSocket Properties

The properties of the TRawSocket class are listed here.

Public Properties

	Name	Description
	SockType (see page 20098)	Sets or returns the socket type.

148.1.7.3.1 TRawSocket.SockType Property

Sets or returns the socket type.

Pascal

```
property SockType: TSocketType;
```

C++

```
__property TSocketType SockType;
```

Description

SockType sets or returns the type of socket connection.

The TSocketType (see page 20138) type defines the set of possible socket types for the SockType property.

See Also

TSocketType (see page 20138)

148.1.8 TServerSocketThread Class

Enables creation of separate threads of execution for the server socket in an application.

Class Hierarchy

Pascal

```
TServerSocketThread = class(TThread);
```

C++

```
class TServerSocketThread : public TThread;
```

File

Sockets

Description

Create (see page 20100) TServerSocketThread (or a descendant of the class) to represent an execution thread for the server socket in a multi-threaded application. Each new instance of a TServerSocketThread, a TThread descendant, is a new thread of execution. Using a separate TServerSocketThread allows the server socket to block without blocking the whole application.

See Also

Writing Multi-threaded Applications

TClientSocketThread (see page 20059)

TThread (see page 3108)

148.1.8.1 TServerSocketThread Members

The following tables list the members exposed by TServerSocketThread.

Protected Methods

	Name	Description
	AddClientSocketThread (see page 20101)	This is AddClientSocketThread, a member of class TServerSocketThread.
	CheckThreadError (see page 3114)	This is the overview for the CheckThreadError method overload.
V	CreateThread (see page 20102)	This is CreateThread, a member of class TServerSocketThread.
V	DoTerminate (see page 3115)	Generates an OnTerminate (see page 3148) event.
	FetchClientSocketThread (see page 20103)	This is FetchClientSocketThread, a member of class TServerSocketThread.
	RemoveClientSocketThread (see page 20103)	This is RemoveClientSocketThread, a member of class TServerSocketThread.

Protected Properties

	Name	Description
	ReturnValue (see page 3141)	Specifies the value returned to other waiting threads when the thread finishes executing.
R	Terminated (see page 3142)	Indicates whether the thread has been asked to terminate.

Public Events

	Name	Description
	OnGetThread (see page 20105)	Creates and initializes a client socket thread when a connection is requested.

	OnTerminate (see page 3148)	Occurs after the thread's Execute (see page 3115) method has returned and before the thread is destroyed.

Public Methods

	Name	Description
V	AfterConstruction (see page 3114)	Executes the thread when the thread is created with CreateSuspended set to false.
	ClearThreadPool (see page 20102)	Terminates all of the threads in the thread pool.
	Create (see page 20100)	Creates an instance of a TServerSocketThread (see page 20099) object.
V	Destroy (see page 20101)	Destroys the thread object and releases the memory allocated to it.
V	Execute (see page 20102)	Provides the code which executes when the thread is run.
	Queue (see page 3119)	This is the overview for the Queue method overload.
	RemoveQueuedEvents (see page 3120)	Remove queued method calls.
	Resume (see page 3120)	Restarts the execution of a suspended thread.
	StaticQueue (see page 3124)	Asynchronously executes a method call within the main thread.
	StaticSynchronize (see page 3125)	Executes a method call within the main thread.
	Suspend (see page 3125)	Pauses a running thread.
	Synchronize (see page 3126)	This is the overview for the Synchronize method overload.
	Terminate (see page 3126)	Signals the thread to terminate by setting the Terminated (see page 3142) property to true.
	WaitFor (see page 3132)	Waits for the thread to terminate and then returns the value of the ReturnValue (see page 3141) property.

Public Properties

	Name	Description
R	FatalException (see page 3133)	Determines whether the thread object is automatically destroyed when the thread terminates.
	FreeOnTerminate (see page 3134)	Determines whether the thread object is automatically destroyed when the thread terminates.
R	Handle (see page 3137)	Indicates the thread's handle (Windows only).
	Priority (see page 3138)	Determines the thread's scheduling priority relative to other threads in the process.
R	ServerSocket (see page 20103)	Specifies the TCustomTcpServer (see page 20072) object that manages all the TClientSocketThread (see page 20059) objects for its client connections.
	Suspended (see page 3142)	Indicates whether a thread is suspended.
	ThreadCacheSize (see page 20104)	Indicates the number of TServerSocketThread (see page 20099) objects cached for reuse by new connections to client sockets.
R	ThreadID (see page 3148)	Identifies the thread throughout the system.
R	ThreadPool (see page 20104)	A list of all the client socket threads created by the TServerSocketThread (see page 20099) object.

148.1.8.2 TServerSocketThread.Create Constructor

Creates an instance of a TServerSocketThread (see page 20099) object.

Pascal

```
constructor Create(AServerSocket: TCustomTcpServer);
```

C++

```
__fastcall TServerSocketThread(TCustomTcpServer AServerSocket);
```

Description

For C++, call Create indirectly, using the **new** keyword, to create a thread in an application.

See Also

Destroy (see page 20101)

Execute (see page 20102)

148.1.8.3 TServerSocketThread.Destroy Destructor

Destroys the thread object and releases the memory allocated to it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TServerSocketThread();
```

Description

For C++, do not call Destroy in an application. Instead, use the **delete** keyword to destroy a thread when FreeOnTerminate is false.

Destroy signals the thread to terminate and then waits for the thread to return before calling the parent class's destructor.

See Also

delete

FreeOnTerminate (see page 3134)

Create (see page 20100)

148.1.8.4 TServerSocketThread Methods

The methods of the TServerSocketThread class are listed here.

Protected Methods

	Name	Description
	AddClientSocketThread (see page 20101)	This is AddClientSocketThread, a member of class TServerSocketThread.
V	CreateThread (see page 20102)	This is CreateThread, a member of class TServerSocketThread.
	FetchClientSocketThread (see page 20103)	This is FetchClientSocketThread, a member of class TServerSocketThread.
	RemoveClientSocketThread (see page 20103)	This is RemoveClientSocketThread, a member of class TServerSocketThread.

Public Methods

	Name	Description
	ClearThreadPool (see page 20102)	Terminates all of the threads in the thread pool.
V	Execute (see page 20102)	Provides the code which executes when the thread is run.

148.1.8.4.1 TServerSocketThread.AddClientSocketThread Method

Pascal

```
function AddClientSocketThread: TClientSocketThread;
```

C++

```
__fastcall TClientSocketThread AddClientSocketThread();
```

Description

This is AddClientSocketThread, a member of class TServerSocketThread.

148.1.8.4.2 TServerSocketThread.ClearThreadPool Method

Terminates all of the threads in the thread pool.

Pascal

```
procedure ClearThreadPool;
```

C++

```
__fastcall ClearThreadPool();
```

Description

The thread pool is a list of client socket threads that are available to handle connection requests. This list is kept to allow reuse of old threads and improve performance. It is sometimes useful to destroy all of the thread object instances in this list.

Use ClearThreadPool to terminate all of the threads in the thread pool list. Any suspended thread in the pool is resumed so that termination can proceed.

See Also

ThreadPool (see page 20104)

Terminate (see page 3126)

148.1.8.4.3 TServerSocketThread.CreateThread Method

Pascal

```
function CreateThread: TClientSocketThread; virtual;
```

C++

```
virtual __fastcall TClientSocketThread CreateThread();
```

Description

This is CreateThread, a member of class TServerSocketThread.

148.1.8.4.4 TServerSocketThread.Execute Method

Provides the code which executes when the thread is run.

Pascal

```
procedure Execute; override;
```

C++

```
virtual __fastcall Execute();
```

Description

Use Execute to execute the code that should be executed when the thread runs. Execute is responsible for checking the value of the Terminated property to determine if the thread needs to exit.

Note: Do not use the properties and methods of other objects directly in the Execute method of a thread. Instead, separate the use of other objects into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize method.

See Also

Terminated (see page 3142)

Synchronize (see page 3126)

Create (see page 20100)

148.1.8.4.5 TServerSocketThread.FetchClientSocketThread Method

Pascal

```
function FetchClientSocketThread: TClientSocketThread;
```

C++

```
__fastcall TClientSocketThread FetchClientSocketThread();
```

Description

This is FetchClientSocketThread, a member of class TServerSocketThread.

148.1.8.4.6 TServerSocketThread.RemoveClientSocketThread Method

Pascal

```
procedure RemoveClientSocketThread(ClientSocketThread: TClientSocketThread);
```

C++

```
__fastcall RemoveClientSocketThread(TClientSocketThread ClientSocketThread);
```

Description

This is RemoveClientSocketThread, a member of class TServerSocketThread.

148.1.8.5 TServerSocketThread Properties

The properties of the TServerSocketThread class are listed here.

Public Properties

	Name	Description
R	ServerSocket (see page 20103)	Specifies the TCustomTcpServer (see page 20072) object that manages all the TClientSocketThread (see page 20059) objects for its client connections.
	ThreadCacheSize (see page 20104)	Indicates the number of TServerSocketThread (see page 20099) objects cached for reuse by new connections to client sockets.
R	ThreadPool (see page 20104)	A list of all the client socket threads created by the TServerSocketThread (see page 20099) object.

148.1.8.5.1 TServerSocketThread.ServerSocket Property

Specifies the TCustomTcpServer (see page 20072) object that manages all the TClientSocketThread (see page 20059) objects for its client connections.

Pascal

```
property ServerSocket: TCustomTcpServer;
```

C++

```
__property TCustomTcpServer ServerSocket;
```

Description

Use ServerSocket to get direct access to the listening socket object that spawned the thread for a client connection it accepted. TServerSocketThread (see page 20099) passes events and errors to the appropriate event handlers assigned to

ServerSocket. TServerSocketThread (see page 20099) adds itself to the cache of threads maintained by ServerSocket from its constructor, and removes itself when the thread terminates.

See Also

TClientSocketThread (see page 20059)

TServerSocketThread (see page 20099)

148.1.8.5.2 TServerSocketThread.ThreadCacheSize Property

Indicates the number of TServerSocketThread (see page 20099) objects cached for reuse by new connections to client sockets.

Pascal

```
property ThreadCacheSize: Integer;
```

C++

```
__property int ThreadCacheSize;
```

Description

Set ThreadCacheSize to specify the number of threads that are cached for reuse. The ideal value for ThreadCacheSize depends on the number and frequency of client socket requests received by the server socket. If ThreadCacheSize is too low, the server socket will spend more time freeing and creating threads when client connections are accepted. If ThreadCacheSize is too high, the server socket may unnecessarily lock up the memory for threads that are never reused.

148.1.8.5.3 TServerSocketThread.ThreadPool Property

A list of all the client socket threads created by the TServerSocketThread (see page 20099) object.

Pascal

```
property ThreadPool: TList;
```

C++

```
__property TList ThreadPool;
```

Description

TServerSocketThread (see page 20099) objects listen for requests on the network and delegate them to client socket threads. Each client socket thread is added to the list ThreadPool after it is created. Creating and destroying client socket threads can be quite time consuming, so whenever one completes its work on a given connection it becomes dormant. When a new connection request is received, the ThreadPool is checked for dormant client socket threads. If one is available, it is assigned the connection. If not, a new client socket thread created. By reusing previously-created threads, the application's performance is improved.

See Also

TClientSocketThread (see page 20059)

ClearThreadPool (see page 20102)

148.1.8.6 TServerSocketThread Events

The events of the TServerSocketThread class are listed here.

Public Events

	Name	Description
	OnGetThread (see page 20105)	Creates and initializes a client socket thread when a connection is requested.

148.1.8.6.1 TServerSocketThread.OnGetThread Event

Creates and initializes a client socket thread when a connection is requested.

Pascal

```
property OnGetThread: TGetThreadEvent;
```

C++

```
__property TGetThreadEvent OnGetThread;
```

Description

An OnGetThread event occurs when a connection is requested. The event handler creates an instance of TClientSocketThread (see page 20059) and initializes it. Write an event handler on TCustomTcpServer (see page 20072) to create and initialize instances of your own client socket thread class.

See Also

TClientSocketThread (see page 20059)

TCustomTcpServer (see page 20072)

148.1.9 TTcpClient Class

TCP client component.

Class Hierarchy

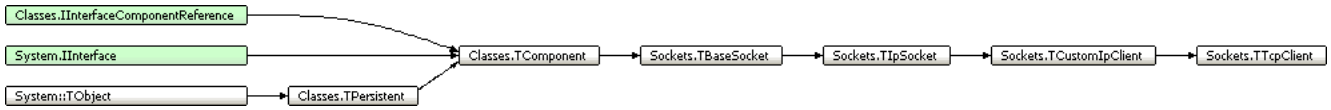

Pascal

```
TTcpClient = class(TCustomIpClient);
```

C++

```
class TTcpClient : public TCustomIpClient;
```

File

Sockets

Description

Use TTcpClient to create TCP client applications.

TTcpClient publishes properties and events from its ancestor socket components.

See Also

TCustomIpClient (see page 20063)

TTcpServer (see page 20115)

148.1.9.1 TTcpClient Members

The following tables list the members exposed by TTcpClient.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Bind (see page 20087)	This is Bind, a member of class TIpSocket.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoConnect (see page 20069)	This is DoConnect, a member of class TCustomIpClient.
	DoCreateHandle (see page 20047)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 20047)	This is DoDestroyHandle, a member of class TBaseSocket.
V	DoDisconnect (see page 20070)	This is DoDisconnect, a member of class TCustomIpClient.
	DoHandleError (see page 20048)	This is DoHandleError, a member of class TBaseSocket.
V	DoReceive (see page 20048)	This is DoReceive, a member of class TBaseSocket.
V	DoSend (see page 20048)	This is DoSend, a member of class TBaseSocket.
V	ErrorCheck (see page 20048)	This is ErrorCheck, a member of class TBaseSocket.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 20049)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBytesReceived (see page 20053)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 20053)	This is SetBytesSent, a member of class TBaseSocket.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property

	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 20047)	Shuts down the socket connection.
	Close (see page 20068)	Shuts down the socket connection.
	Connect (see page 20069)	Connects the local socket to the remote socket.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 20046)	Creates an instance of TBaseSocket (see page 20042).
	Create (see page 20068)	Creates an instance of TCustomIpClient (see page 20063).
	Create (see page 20086)	Creates an instance of TIpSocket (see page 20082).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20046)	Frees the memory associated with the TBaseSocket (see page 20042) object.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Disconnect (see page 20069)	Disconnects the local socket from the remote socket.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetSocketAddr (see page 20087)	Builds a valid network socket address.
	GetThreadObject (see page 20070)	Returns a reference to the TThread object for the currently executing thread.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LocalDomainName (see page 20088)	Returns the domain name for the local host.
	LocalHostAddr (see page 20088)	Returns the local IP address for the host machine.
	LocalHostName (see page 20088)	Returns the local host name.
	LookupHostAddr (see page 20089)	Looks up the host address based on the host name.
	LookupHostName (see page 20089)	Looks up the host name based on the IP address.
	LookupPort (see page 20089)	Returns the port number.
	LookupProtocol (see page 20090)	Returns the protocol number.
	MapDomain (see page 20049)	Maps the domain.
	MapSockType (see page 20050)	Maps the socket type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Open (see page 20050)	Creates the socket connection.
V	Open (see page 20070)	Creates the socket connection.
	PeekBuf (see page 20050)	Reads the buffer.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReceiveBuf (see page 20051)	Reads data buffers of to the socket.
	ReceiveFrom (see page 20090)	Receives a packet of information.
	ReceiveIn (see page 20051)	Reads delimited lines of data from the socket.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Select (see page 20051)	Tests for socket activity.
	SendBuf (see page 20052)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 20052)	Writes delimited lines of data to the socket.
	SendStream (see page 20053)	Writes a stream of data to the socket.
	SendTo (see page 20091)	Sends a packet of information.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WaitForData (see page 20053)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
R	BytesReceived (see page 20055)	Returns the number of bytes received by the socket.
R	BytesSent (see page 20055)	Returns the number of bytes sent from the socket.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Domain (see page 20055)	Sets or returns the protocol family used by the socket.
	Domain (see page 20091)	Sets or returns the protocol family used by the socket.
R	Handle (see page 20056)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	LocalHost (see page 20092)	Specifies the name of the local system.
	LocalPort (see page 20092)	Specifies the ID number of the port used by the socket connection.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Protocol (see page 20056)	Sets or returns the protocol used by the socket.
	SockType (see page 20056)	Sets or returns the socket type.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnConnect (see page 20112)	Occurs on client sockets just after the connection to the server is opened.
	OnCreateHandle (see page 20113)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 20113)	Occurs when a socket is deactivated.
	OnDisconnect (see page 20113)	Occurs on client sockets just after the connection to the server is closed.
	OnError (see page 20114)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 20114)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 20114)	Occurs just before a client socket writes information to the socket connection.

Published Properties

	Name	Description
	Active (see page 20110)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 20110)	Sets or returns the blocking mode of the socket.
	Connected (see page 20111)	Indicates whether the socket connection is open and available for communication with other machines.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RemoteHost (see page 20111)	Specifies the name of the remote system.
	RemotePort (see page 20111)	Specifies the ID number of the port used by the RemoteHost to form the socket connection.
	Tag (see page 2808)	Stores an integer value as part of a component.

148.1.9.2 TTcpClient Properties

The properties of the TTcpClient class are listed here.

Published Properties

	Name	Description
	Active (see page 20110)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 20110)	Sets or returns the blocking mode of the socket.
	Connected (see page 20111)	Indicates whether the socket connection is open and available for communication with other machines.
	RemoteHost (see page 20111)	Specifies the name of the remote system.
	RemotePort (see page 20111)	Specifies the ID number of the port used by the RemoteHost to form the socket connection.

148.1.9.2.1 TTcpClient.Active Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before attempting to use or change the socket connection, read Active to determine whether the connection is open and ready. For client sockets, setting Active opens or shuts down a socket connection to another machine. For server sockets, setting Active opens or shuts down a listening connection that makes the socket available for client requests.

At design time, set Active to **true** to make the socket open a connection when the application starts running (the default value is false). At runtime, use the Open or Close method to open or close the connection.

See Also

Close (see page 20047)

Open (see page 20050)

148.1.9.2.2 TTcpClient.BlockMode Property

Sets or returns the blocking mode of the socket.

Pascal

```
property BlockMode: TSocketBlockMode;
```

C++

```
__property TSocketBlockMode BlockMode;
```

Description

Use BlockMode to set or return the blocking mode for the socket. By default, the socket is set to bmBlocking.

The TSocketBlockMode type defines the set of possible socket flags for the BlockMode property.

See Also

TSocketBlockMode (see page 20141)

OnReceive (see page 20058)

OnSend (see page 20058)

148.1.9.2.3 TTcpClient.Connected Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Read Connected before trying to use or change the socket connection. Connected indicates whether the socket connection has been established. When Connected is **true**, the socket is open and available for use. When Connected is **false**, the socket is closed and can be changed.

To open a socket, use the Open method. To close a socket, use the Closemethod.

See Also

Close (see page 20068)

Open (see page 20070)

OnConnect (see page 20071)

148.1.9.2.4 TTcpClient.RemoteHost Property

Specifies the name of the remote system.

Pascal

```
property RemoteHost: TSocketHost;
```

C++

```
__property TSocketHost RemoteHost;
```

Description

Use RemoteHost to get the name of the system on the other end of the socket connection. RemoteHost is a string containing the domain name and service of the remote endpoint of the socket connection, such as http://www.wSite.Com.

See Also

RemotePort (see page 20093)

LocalHostAddr (see page 20088)

LocalHostName (see page 20088)

LookupHostAddr (see page 20089)

LookupHostName (see page 20089)

148.1.9.2.5 TTcpClient.RemotePort Property

Specifies the ID number of the port used by the RemoteHost to form the socket connection.

Pascal

```
property RemotePort: TSocketPort;
```

C++

```
__property TSocketPort RemotePort;
```

Description

Use RemotePort to determine the port number used by the socket on the other end of the socket connection.

Port numbers allow a single system, identified by the RemoteHost property, to host multiple connections simultaneously. Each combination of RemotePort and RemoteHost can only be bound to a single socket connection.

Many port numbers are associated by convention with a particular service such as ftp or http. To determine whether the value of RemotePort is associated with a specific service, use the LookupProtocol method.

Note: If a client socket requests a specific port to indicate a desired service, the value of RemotePort for the actual connection may differ from the requested port number. Most server sockets listen on the port associated with a service, but switch to an arbitrary available port number for forming connections.

See Also

RemoteHost (see page 20093)

LookupPort (see page 20089)

LookupProtocol (see page 20090)

148.1.9.3 TTcpClient Events

The events of the TTcpClient class are listed here.

Published Events

	Name	Description
	OnConnect (see page 20112)	Occurs on client sockets just after the connection to the server is opened.
	OnCreateHandle (see page 20113)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 20113)	Occurs when a socket is deactivated.
	OnDisconnect (see page 20113)	Occurs on client sockets just after the connection to the server is closed.
	OnError (see page 20114)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 20114)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 20114)	Occurs just before a client socket writes information to the socket connection.

148.1.9.3.1 TTcpClient.OnConnect Event

Occurs on client sockets just after the connection to the server is opened.

Pascal

```
property OnConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnect;
```

Description

Write an OnConnect event handler for a client socket to take specific action after the connection to a server socket has been established. Depending on the service, this may be the point when the socket should start reading or writing over the connection.

See Also

Connected (see page 20071)

OnDisconnect (see page 20072)

148.1.9.3.2 TTcpClient.OnCreateHandle Event

Occurs when the handle to a socket becomes active.

Pascal

```
property OnCreateHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnCreateHandle;
```

Description

Write an OnCreateHandle event handler to take specific action when the handle to a socket becomes active.

See Also

Active (see page 20054)

Handle (see page 20056)

148.1.9.3.3 TTcpClient.OnDestroyHandle Event

Occurs when a socket is deactivated.

Pascal

```
property OnDestroyHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDestroyHandle;
```

Description

Write an OnDestroyHandle event handler to take specific action when the socket is deactivated.

148.1.9.3.4 TTcpClient.OnDisconnect Event

Occurs on client sockets just after the connection to the server is closed.

Pascal

```
property OnDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDisconnect;
```

Description

Write an OnDisconnect event handler for a client socket to take specific action just after the connection to a server socket is terminated.

See Also

Connected (see page 20071)

OnConnect (see page 20071)

148.1.9.3.5 TTcpClient.OnError Event

Occurs when the socket fails in making, using, or shutting down a connection.

Pascal

```
property OnError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnError;
```

Description

Write an OnError event handler to respond to errors that arise with the socket connection. Set the **ErrorCode** parameter to 0 if the OnError event handler successfully deals with the error condition, to prevent an ESocketError exception from being raised.

See Also

ESocketError (see page 20041)

148.1.9.3.6 TTcpClient.OnReceive Event

Occurs after a socket receives data, but before it passes the data to the application.

Pascal

```
property OnReceive: TSocketDataEvent;
```

C++

```
__property TSocketDataEvent OnReceive;
```

Description

Write an OnReceive event handler to read from the socket connection. Use the OnReceive event to filter data before passing it back to the client application. If the socket is a blocking socket, use a TWinSocketStream object to read from the connection. Otherwise, use the methods of the **Socket** parameter to perform the actual reading.

See Also

TWinSocketStream (see page 18876)

BytesReceived (see page 20055)

ReceiveBuf (see page 20051)

ReceiveIn (see page 20051)

148.1.9.3.7 TTcpClient.OnSend Event

Occurs just before a client socket writes information to the socket connection.

Pascal

```
property OnSend: TSocketDataEvent;
```

C++

```
__property TSocketDataEvent OnSend;
```

Description

Write an OnSend event handler to perform actions just prior to sending data. Use OnSend to monitor the data stream and change it (for example, by removing linefeeds). If the socket is a blocking socket, use a TWinSocketStream object to write to the

connection. Otherwise, use the methods of the **Socket** parameter to perform the actual writing.

See Also

TWinSocketStream (see page 18876)

SendBuf (see page 20052)

SendLn (see page 20052)

148.1.10 TTcpServer Class

TTcpServer is the TCP server component.

Class Hierarchy

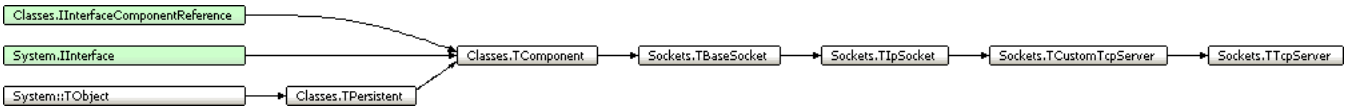

Pascal

```
TTcpServer = class(TCustomTcpServer);
```

C++

```
class TTcpServer : public TCustomTcpServer;
```

File

Sockets

Description

Use TTcpServer to create TCP server applications.

TTcpServer publishes properties and events from its ancestor socket components.

See Also

TCustomTcpServer

TTcpClient

148.1.10.1 TTcpServer Members

The following tables list the members exposed by TTcpServer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Bind (see page 20087)	This is Bind, a member of class TIpSocket.

	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAccept (see page 20078)	This is DoAccept, a member of class TCustomTcpServer.
	DoCreateHandle (see page 20047)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 20047)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoHandleError (see page 20048)	This is DoHandleError, a member of class TBaseSocket.
V	DoReceive (see page 20048)	This is DoReceive, a member of class TBaseSocket.
V	DoSend (see page 20048)	This is DoSend, a member of class TBaseSocket.
V	ErrorCheck (see page 20048)	This is ErrorCheck, a member of class TBaseSocket.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Listen (see page 20079)	This is Listen, a member of class TCustomTcpServer.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 20049)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBytesReceived (see page 20053)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 20053)	This is SetBytesSent, a member of class TBaseSocket.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnError (see page 20058)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 20058)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 20058)	Occurs just before a client socket writes information to the socket connection.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Accept (see page 20077)	This is the overview for the Accept method overload.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Close (see page 20047)	Shuts down the socket connection.
V	Close (see page 20078)	Shuts down the socket connection.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 20046)	Creates an instance of TBaseSocket (see page 20042).
V	Create (see page 20076)	Creates an instance of TCustomTcpServer (see page 20072).
V	Create (see page 20086)	Creates an instance of TIpSocket (see page 20082).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 20046)	Frees the memory associated with the TBaseSocket (see page 20042) object.
V	Destroy (see page 20077)	Frees the memory associated with the TCustomTcpServer (see page 20072) object.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetSocketAddr (see page 20087)	Builds a valid network socket address.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LocalDomainName (see page 20088)	Returns the domain name for the local host.
	LocalHostAddr (see page 20088)	Returns the local IP address for the host machine.
	LocalHostName (see page 20088)	Returns the local host name.
	LookupHostAddr (see page 20089)	Looks up the host address based on the host name.
	LookupHostName (see page 20089)	Looks up the host name based on the IP address.
	LookupPort (see page 20089)	Returns the port number.
	LookupProtocol (see page 20090)	Returns the protocol number.
	MapDomain (see page 20049)	Maps the domain.
	MapSockType (see page 20050)	Maps the socket type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Open (see page 20050)	Creates the socket connection.
V	Open (see page 20079)	Opens the socket connection.
	PeekBuf (see page 20050)	Reads the buffer.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReceiveBuf (see page 20051)	Reads data buffers of to the socket.
	ReceiveFrom (see page 20090)	Receives a packet of information.
	ReceiveIn (see page 20051)	Reads delimited lines of data from the socket.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Select (see page 20051)	Tests for socket activity.
	SendBuf (see page 20052)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 20052)	Writes delimited lines of data to the socket.
	SendStream (see page 20053)	Writes a stream of data to the socket.
	SendTo (see page 20091)	Sends a packet of information.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WaitForConnection (see page 20079)	Listens for TCP connections.
	WaitForData (see page 20053)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
R	BytesReceived (see page 20055)	Returns the number of bytes received by the socket.
R	BytesSent (see page 20055)	Returns the number of bytes sent from the socket.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.

R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Domain (see page 20055)	Sets or returns the protocol family used by the socket.
	Domain (see page 20091)	Sets or returns the protocol family used by the socket.
R	Handle (see page 20056)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
R	Listening (see page 20080)	Sets the server application to listen for client requests.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Protocol (see page 20056)	Sets or returns the protocol used by the socket.
	RemoteHost (see page 20093)	Specifies the name of the remote system.
	RemotePort (see page 20093)	Specifies the ID number of the port used by the RemoteHost (see page 20093) to form the socket connection.
	ServerSocketThread (see page 20080)	Specifies the TServerSocketThread (see page 20099) in use.
	SockType (see page 20056)	Sets or returns the socket type.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnAccept (see page 20122)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnCreateHandle (see page 20122)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 20122)	Occurs when a socket is deactivated.
	OnGetThread (see page 20123)	A wrapper for the TServerSocketThread event of the same name.
	OnListening (see page 20123)	Occurs when a server socket begins listening for connections.

Published Properties

	Name	Description
	Active (see page 20120)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 20120)	Sets or returns the blocking mode of the socket.
	LocalHost (see page 20120)	Specifies the name of the local system.
	LocalPort (see page 20121)	Specifies the ID number of the port used by the socket connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

148.1.10.2 TTcpServer Properties

The properties of the TTcpServer class are listed here.

Published Properties

	Name	Description
	Active (see page 20120)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 20120)	Sets or returns the blocking mode of the socket.
	LocalHost (see page 20120)	Specifies the name of the local system.
	LocalPort (see page 20121)	Specifies the ID number of the port used by the socket connection.

148.1.10.2.1 TTcpServer.Active Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before attempting to use or change the socket connection, read Active to determine whether the connection is open and ready. For client sockets, setting Active opens or shuts down a socket connection to another machine. For server sockets, setting Active opens or shuts down a listening connection that makes the socket available for client requests.

At design time, set Active to **true** to make the socket open a connection when the application starts running (the default value is false). At runtime, use the Open or Close method to open or close the connection.

See Also

Close (see page 20047)

Open (see page 20050)

148.1.10.2.2 TTcpServer.BlockMode Property

Sets or returns the blocking mode of the socket.

Pascal

```
property BlockMode: TSocketBlockMode;
```

C++

```
__property TServerSocketBlockMode BlockMode;__property TSocketBlockMode BlockMode;
```

Description

Use BlockMode to set or return the blocking mode for the socket. By default, the socket is set to bmThreadBlocking.

The TServerSocketBlockMode type defines the set of possible socket flags for the BlockMode property.

See Also

OnReceive (see page 20058)

OnSend (see page 20058)

TServerSocketBlockMode (see page 20136)

148.1.10.2.3 TTcpServer.LocalHost Property

Specifies the name of the local system.

Pascal

```
property LocalHost: TSocketHost;
```

C++

```
__property TSocketHost LocalHost;
```

Description

Use LocalHost to get the name of the system that is running the application that uses the socket. LocalHost is a string containing the domain name and service of the local socket endpoint, such as http://www.wSite.Com.

Port numbers allow a single system, identified by the LocalHostAddr function, to host multiple connections simultaneously. Each combination of LocalPort and LocalHostAddr can only be bound to a single socket connection.

LocalHost provides an alias for the local IP address given in the LocalHostAddr function. Most Intranets provide host names for the IP addresses of systems on the net. For more information on defining a host name for a given system, contact your network administrator.

See Also

LocalPort (see page 20092)

RemoteHost (see page 20093)

LocalHostAddr (see page 20088)

LookupHostName (see page 20089)

LookupPort (see page 20089)

148.1.10.2.4 TTcpServer.LocalPort Property

Specifies the ID number of the port used by the socket connection.

Pascal

```
property LocalPort: TSocketPort;
```

C++

```
__property TSocketPort LocalPort;
```

Description

Use LocalPort to determine the port number that is bound to the socket connection. For server endpoints of listening connections, LocalPort is the ID associated with the service the server socket provides. For client or server endpoints of connections to another socket, LocalPort is usually an arbitrary value picked when the connection was made.

Many values of LocalPort are associated by convention with a particular service such as FTP or HTTP.

See Also

LocalHost (see page 20092)

RemotePort (see page 20093)

LookupPort (see page 20089)

148.1.10.3 TTcpServer Events

The events of the TTcpServer class are listed here.

Published Events

	Name	Description
	OnAccept (see page 20122)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnCreateHandle (see page 20122)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 20122)	Occurs when a socket is deactivated.

	OnGetThread (see page 20123)	A wrapper for the TServerSocketThread event of the same name.
	OnListening (see page 20123)	Occurs when a server socket begins listening for connections.

148.1.10.3.1 TTcpServer.OnAccept Event

Occurs on server sockets just after the connection to a client socket is accepted.

Pascal

```
property OnAccept: TSocketAcceptEvent;
```

C++

```
__property TSocketAcceptEvent OnAccept;
```

Description

Write an OnAccept event handler for a server socket to take specific action after the connection to a client socket has been accepted. This is the first point when information is available about the local port and IP address that is used in the server endpoint of the connection. The port number, and possibly the IP address, differs from those of the listening socket.

Server sockets open a socket connection for listening to establish a queue to hold client requests. After a request from the queue is accepted by the server socket, an OnAccept event occurs. After the server socket accepts the connection, the client socket completes the connection, and an OnConnect event occurs for the client socket.

See Also

Active

OnConnect (see page 20071)

OnListening

148.1.10.3.2 TTcpServer.OnCreateHandle Event

Occurs when the handle to a socket becomes active.

Pascal

```
property OnCreateHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnCreateHandle;
```

Description

Write an OnCreateHandle event handler to take specific action when the handle to a socket becomes active.

See Also

Active (see page 20054)

Handle (see page 20056)

148.1.10.3.3 TTcpServer.OnDestroyHandle Event

Occurs when a socket is deactivated.

Pascal

```
property OnDestroyHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDestroyHandle;
```

Description

Write an OnDestroyHandle event handler to take specific action when the socket is deactivated.

148.1.10.3.4 TTcpServer.OnGetThread Event

A wrapper for the TServerSocketThread event of the same name.

Pascal

```
property OnGetThread: TGetThreadEvent;
```

C++

```
__property TGetThreadEvent OnGetThread;
```

Description

This event is exactly the same as the OnGetThread event of the TServerSocketThread object referenced in ServerSocketThread.

See Also

TServerSocketThread (see page 20099)

ServerSocketThread (see page 20063)

OnGetThread (see page 20105)

148.1.10.3.5 TTcpServer.OnListening Event

Occurs when a server socket begins listening for connections.

Pascal

```
property OnListening: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnListening;
```

Description

Write an OnListening event to take specific action when a server begins listening for a connection.

See Also

Listening (see page 20080)

TNotifyEvent (see page 3298)

148.1.11 TUdpSocket Class

UDP component.

Class Hierarchy

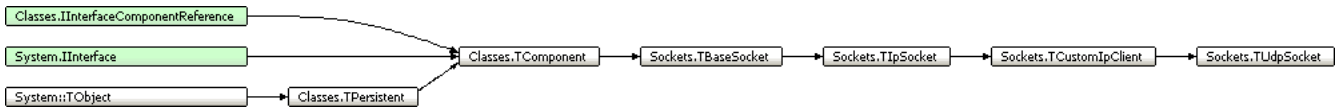

Pascal

```
TUdpSocket = class(TCustomIpClient);
```

C++

```
class TUdpSocket : public TCustomIpClient;
```

File

Sockets

Description

Use TUdpSocket to create UDP-based applications. Add a TUdpSocket object to a form or data module to turn an application into a UDP/IP client and server. TUdpSocket specifies a desired connection to a UDP/IP server, manages the connection when it is open, and terminates the connection when the application is through. It also listens for requests for UDP/IP connections from other machines and establishes connections when requests are received.

148.1.11.1 TUdpSocket Members

The following tables list the members exposed by TUdpSocket.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Bind (see page 20087)	This is Bind, a member of class TIpSocket.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoConnect (see page 20069)	This is DoConnect, a member of class TCustomIpClient.
	DoCreateHandle (see page 20047)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 20047)	This is DoDestroyHandle, a member of class TBaseSocket.
V	DoDisconnect (see page 20070)	This is DoDisconnect, a member of class TCustomIpClient.
	DoHandleError (see page 20048)	This is DoHandleError, a member of class TBaseSocket.
V	DoReceive (see page 20048)	This is DoReceive, a member of class TBaseSocket.
V	DoSend (see page 20048)	This is DoSend, a member of class TBaseSocket.
V	ErrorCheck (see page 20048)	This is ErrorCheck, a member of class TBaseSocket.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 20049)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBytesReceived (see page 20053)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 20053)	This is SetBytesSent, a member of class TBaseSocket.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Close (see page 20047)	Shuts down the socket connection.
V	Close (see page 20068)	Shuts down the socket connection.
	Connect (see page 20069)	Connects the local socket to the remote socket.
V	Create (see page 20128)	Creates an instance of TUdpSocket (see page 20123).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 20046)	Frees the memory associated with the TBaseSocket (see page 20042) object.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Disconnect (see page 20069)	Disconnects the local socket from the remote socket.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetSocketAddr (see page 20087)	Builds a valid network socket address.
	GetThreadObject (see page 20070)	Returns a reference to the TThread object for the currently executing thread.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LocalDomainName (see page 20088)	Returns the domain name for the local host.
	LocalHostAddr (see page 20088)	Returns the local IP address for the host machine.
	LocalHostName (see page 20088)	Returns the local host name.
	LookupHostAddr (see page 20089)	Looks up the host address based on the host name.
	LookupHostName (see page 20089)	Looks up the host name based on the IP address.
	LookupPort (see page 20089)	Returns the port number.
	LookupProtocol (see page 20090)	Returns the protocol number.
	MapDomain (see page 20049)	Maps the domain.
	MapSockType (see page 20050)	Maps the socket type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Open (see page 20050)	Creates the socket connection.
V	Open (see page 20070)	Creates the socket connection.
	PeekBuf (see page 20050)	Reads the buffer.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReceiveBuf (see page 20051)	Reads data buffers of to the socket.
	ReceiveFrom (see page 20090)	Receives a packet of information.
	ReceiveIn (see page 20051)	Reads delimited lines of data from the socket.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Select (see page 20051)	Tests for socket activity.
	SendBuf (see page 20052)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 20052)	Writes delimited lines of data to the socket.
	SendStream (see page 20053)	Writes a stream of data to the socket.
	SendTo (see page 20091)	Sends a packet of information.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WaitForData (see page 20053)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
R	BytesReceived (see page 20055)	Returns the number of bytes received by the socket.
R	BytesSent (see page 20055)	Returns the number of bytes sent from the socket.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Connected (see page 20071)	Indicates whether the socket connection is open and available for communication with other machines.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Domain (see page 20055)	Sets or returns the protocol family used by the socket.
	Domain (see page 20091)	Sets or returns the protocol family used by the socket.
R	Handle (see page 20056)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Protocol (see page 20056)	Sets or returns the protocol used by the socket.
	SockType (see page 20056)	Sets or returns the socket type.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnConnect (see page 20131)	Occurs on client sockets just after the connection to the server is opened.
	OnCreateHandle (see page 20132)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 20132)	Occurs when a socket is deactivated.
	OnDisconnect (see page 20132)	Occurs on client sockets just after the connection to the server is closed.
	OnError (see page 20133)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 20133)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 20133)	Occurs just before a client socket writes information to the socket connection.

Published Properties

	Name	Description
	Active (see page 20128)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 20129)	Sets or returns the blocking mode of the socket.
	LocalHost (see page 20129)	Specifies the name of the local system.
	LocalPort (see page 20130)	Specifies the ID number of the port used by the socket connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.

	RemoteHost (see page 20130)	Specifies the name of the remote system.
	RemotePort (see page 20130)	Specifies the ID number of the port used by the RemoteHost to form the socket connection.
	Tag (see page 2808)	Stores an integer value as part of a component.

148.1.11.2 TUdpSocket.Create Constructor

Creates an instance of TUdpSocket (see page 20123).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TUdpSocket(TComponent * AOwner);
```

Description

For C++, call Create indirectly, using the **new** keyword, to instantiate a TUdpSocket (see page 20123) object.

148.1.11.3 TUdpSocket Properties

The properties of the TUdpSocket class are listed here.

Published Properties

	Name	Description
	Active (see page 20128)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 20129)	Sets or returns the blocking mode of the socket.
	LocalHost (see page 20129)	Specifies the name of the local system.
	LocalPort (see page 20130)	Specifies the ID number of the port used by the socket connection.
	RemoteHost (see page 20130)	Specifies the name of the remote system.
	RemotePort (see page 20130)	Specifies the ID number of the port used by the RemoteHost to form the socket connection.

148.1.11.3.1 TUdpSocket.Active Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before attempting to use or change the socket connection, read Active to determine whether the connection is open and ready. For client sockets, setting Active opens or shuts down a socket connection to another machine. For server sockets, setting Active opens or shuts down a listening connection that makes the socket available for client requests.

At design time, set Active to **true** to make the socket open a connection when the application starts running (the default value is false). At runtime, use the Open or Close method to open or close the connection.

See Also

Close (see page 20047)

Open (see page 20050)

148.1.11.3.2 TUdpSocket.BlockMode Property

Sets or returns the blocking mode of the socket.

Pascal

```
property BlockMode: TSocketBlockMode;
```

C++

```
__property TSocketBlockMode BlockMode;
```

Description

Use BlockMode to set or return the blocking mode for the socket. By default, the socket is set to bmBlocking.

The TSocketBlockMode type defines the set of possible socket flags for the BlockMode property.

See Also

TSocketBlockMode (see page 20141)

OnReceive (see page 20058)

OnSend (see page 20058)

148.1.11.3.3 TUdpSocket.LocalHost Property

Specifies the name of the local system.

Pascal

```
property LocalHost: TSocketHost;
```

C++

```
__property TSocketHost LocalHost;
```

Description

Use LocalHost to get the name of the system that is running the application that uses the socket. LocalHost is a string containing the domain name and service of the local socket endpoint, such as http://www.wSite.Com.

Port numbers allow a single system, identified by the LocalHostAddr function, to host multiple connections simultaneously. Each combination of LocalPort and LocalHostAddr can only be bound to a single socket connection.

LocalHost provides an alias for the local IP address given in the LocalHostAddr function. Most Intranets provide host names for the IP addresses of systems on the net. For more information on defining a host name for a given system, contact your network administrator.

See Also

LocalPort (see page 20092)

RemoteHost (see page 20093)

LocalHostAddr (see page 20088)

LookupHostName (see page 20089)

LookupPort (see page 20089)

148.1.11.3.4 TUdpSocket.LocalPort Property

Specifies the ID number of the port used by the socket connection.

Pascal

```
property LocalPort: TSocketPort;
```

C++

```
__property TSocketPort LocalPort;
```

Description

Use LocalPort to determine the port number that is bound to the socket connection. For server endpoints of listening connections, LocalPort is the ID associated with the service the server socket provides. For client or server endpoints of connections to another socket, LocalPort is usually an arbitrary value picked when the connection was made.

Many values of LocalPort are associated by convention with a particular service such as FTP or HTTP.

See Also

LocalHost (see page 20092)

RemotePort (see page 20093)

LookupPort (see page 20089)

148.1.11.3.5 TUdpSocket.RemoteHost Property

Specifies the name of the remote system.

Pascal

```
property RemoteHost: TSocketHost;
```

C++

```
__property TSocketHost RemoteHost;
```

Description

Use RemoteHost to get the name of the system on the other end of the socket connection. RemoteHost is a string containing the domain name and service of the remote endpoint of the socket connection, such as http://www.wSite.Com.

See Also

RemotePort (see page 20093)

LocalHostAddr (see page 20088)

LocalHostName (see page 20088)

LookupHostAddr (see page 20089)

LookupHostName (see page 20089)

148.1.11.3.6 TUdpSocket.RemotePort Property

Specifies the ID number of the port used by the RemoteHost to form the socket connection.

Pascal

```
property RemotePort: TSocketPort;
```

C++

```
__property TSocketPort RemotePort;
```

Description

Use RemotePort to determine the port number used by the socket on the other end of the socket connection.

Port numbers allow a single system, identified by the RemoteHost property, to host multiple connections simultaneously. Each combination of RemotePort and RemoteHost can only be bound to a single socket connection.

Many port numbers are associated by convention with a particular service such as ftp or http. To determine whether the value of RemotePort is associated with a specific service, use the LookupProtocol method.

Note: If a client socket requests a specific port to indicate a desired service, the value of RemotePort for the actual connection may differ from the requested port number. Most server sockets listen on the port associated with a service, but switch to an arbitrary available port number for forming connections.

See Also

RemoteHost (see page 20093)

LookupPort (see page 20089)

LookupProtocol (see page 20090)

148.1.11.4 TUdpSocket Events

The events of the TUdpSocket class are listed here.

Published Events

	Name	Description
	OnConnect (see page 20131)	Occurs on client sockets just after the connection to the server is opened.
	OnCreateHandle (see page 20132)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 20132)	Occurs when a socket is deactivated.
	OnDisconnect (see page 20132)	Occurs on client sockets just after the connection to the server is closed.
	OnError (see page 20133)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 20133)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 20133)	Occurs just before a client socket writes information to the socket connection.

148.1.11.4.1 TUdpSocket.OnConnect Event

Occurs on client sockets just after the connection to the server is opened.

Pascal

```
property OnConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnect;
```

Description

Write an OnConnect event handler for a client socket to take specific action after the connection to a server socket has been established. Depending on the service, this may be the point when the socket should start reading or writing over the connection.

See Also

Connected (see page 20071)

OnDisconnect (see page 20072)

148.1.11.4.2 TUdpSocket.OnCreateHandle Event

Occurs when the handle to a socket becomes active.

Pascal

```
property OnCreateHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnCreateHandle;
```

Description

Write an OnCreateHandle event handler to take specific action when the handle to a socket becomes active.

See Also

Active (see page 20054)

Handle (see page 20056)

148.1.11.4.3 TUdpSocket.OnDestroyHandle Event

Occurs when a socket is deactivated.

Pascal

```
property OnDestroyHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDestroyHandle;
```

Description

Write an OnDestroyHandle event handler to take specific action when the socket is deactivated.

148.1.11.4.4 TUdpSocket.OnDisconnect Event

Occurs on client sockets just after the connection to the server is closed.

Pascal

```
property OnDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDisconnect;
```

Description

Write an OnDisconnect event handler for a client socket to take specific action just after the connection to a server socket is terminated.

See Also

Connected (see page 20071)

OnConnect (see page 20071)

148.1.11.4.5 TUdpSocket.OnError Event

Occurs when the socket fails in making, using, or shutting down a connection.

Pascal

```
property OnError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnError;
```

Description

Write an OnError event handler to respond to errors that arise with the socket connection. Set the **ErrorCode** parameter to 0 if the OnError event handler successfully deals with the error condition, to prevent an ESocketError exception from being raised.

See Also

ESocketError (see page 20041)

148.1.11.4.6 TUdpSocket.OnReceive Event

Occurs after a socket receives data, but before it passes the data to the application.

Pascal

```
property OnReceive: TSocketDataEvent;
```

C++

```
__property TSocketDataEvent OnReceive;
```

Description

Write an OnReceive event handler to read from the socket connection. Use the OnReceive event to filter data before passing it back to the client application. If the socket is a blocking socket, use a TWinSocketStream object to read from the connection. Otherwise, use the methods of the **Socket** parameter to perform the actual reading.

See Also

TWinSocketStream (see page 18876)

BytesReceived (see page 20055)

ReceiveBuf (see page 20051)

ReceiveIn (see page 20051)

148.1.11.4.7 TUdpSocket.OnSend Event

Occurs just before a client socket writes information to the socket connection.

Pascal

```
property OnSend: TSocketDataEvent;
```

C++

```
__property TSocketDataEvent OnSend;
```

Description

Write an OnSend event handler to perform actions just prior to sending data. Use OnSend to monitor the data stream and change it (for example, by removing linefeeds). If the socket is a blocking socket, use a TWinSocketStream object to write to the

connection. Otherwise, use the methods of the **Socket** parameter to perform the actual writing.

See Also

TWinSocketStream (see page 18876)

SendBuf (see page 20052)

SendIn (see page 20052)

148.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TServerSocketBlockMode (see page 20136)	Defines the set of possible socket flags for block mode.
	TSocketDomain (see page 20136)	Set of possible protocol family types used by a socket.
	TSocketType (see page 20138)	The TSocketType type defines the set of possible socket types.

Records

	Name	Description
	TIPHeader (see page 20135)	This is record Sockets.TIPHeader.

148.2.1 Sockets.TIPHeader Record

Pascal

```
TIPHeader = packed record
  iph_verlen: byte;
  iph_tos: byte;
  iph_length: word;
  iph_id: word;
  iph_offset: word;
  iph_ttl: byte;
  iph_protocol: byte;
  iph_xsum: word;
  iph_src: longword;
  iph_dest: longword;
end;
```

C++

```
struct TIPHeader {
  byte iph_verlen;
  byte iph_tos;
  word iph_length;
  word iph_id;
  word iph_offset;
  byte iph_ttl;
  byte iph_protocol;
  word iph_xsum;
  longword iph_src;
  longword iph_dest;
};
```

File

Sockets

Description

This is record Sockets.TIPHeader.

148.2.2 Sockets.TServerSocketBlockMode Enumeration

Defines the set of possible socket flags for block mode.

Pascal

```
TServerSocketBlockMode = (
  bmBlocking,
  bmNonBlocking,
  bmThreadBlocking
);
```

C++

```
enum TServerSocketBlockMode {
  bmBlocking,
  bmNonBlocking,
  bmThreadBlocking
};
```

File

Sockets

Description

The following table lists the possible values for the TServerSocketBlockMode type and the meaning corresponding to each value:

Value	Socket state
bmBlocking	The socket is set to block mode.
bmNonBlocking	The socket is set to non-blocking mode.
bmThreadBlocking	A separate thread is created so that listening can be performed in the background without blocking the entire server application.

148.2.3 Sockets.TSocketDomain Enumeration

Set of possible protocol family types used by a socket.

Pascal

```
TSocketDomain = (
  pfUnspec,
  pfUnix,
  pfInet,
  pfImpLink,
  pfPup,
  pfChaos,
  pfIpx,
  pfNs,
  pfIso,
  pfOsi,
  pfEcma,
  pfDataKit,
  pfCcitt,
  pfSna,
  pfDecNet,
```

```
    pfDli,
    pfLat,
    pfHylink,
    pfAppleTalk,
    pfVoiceView,
    pfFireFox,
    pfUnknown1,
    pfBan,
    pfMax
  );
```

C++

```
  enum TSocketDomain {
    pfUnspec,
    pfUnix,
    pfInet,
    pfImpLink,
    pfPup,
    pfChaos,
    pfIpx,
    pfNs,
    pfIso,
    pfOsi,
    pfEcma,
    pfDataKit,
    pfCcitt,
    pfSna,
    pfDecNet,
    pfDli,
    pfLat,
    pfHylink,
    pfAppleTalk,
    pfVoiceView,
    pfFireFox,
    pfUnknown1,
    pfBan,
    pfMax
  };
```

File

Sockets

Description

The following table lists the possible values for the TSocketDomain type and the meaning corresponding to each value:

Value	Description
pfAppleTalk	AppleTalk DDP
pfBan	Banyan Vines
pfCcitt	CCIT (International Telegraph and Telephone Consultative)
pfChaos	
pfDataKit	DataKit
pfDecNet	Reserved for DECnet project
pfDli	
pfEcma	EcmaScript
pfFireFox	FireFox
pfHylink	Hyperlink
pfImpLink	Imported link

pfInet	IP protocol family
pfIpx	Novell Interner Protocol
pfIso	International Standards Organization (ISO)
pfLat	
pfMax	Maximum protocol family number
pfNs	Network Services Protocol (used with DECnet)
pfOsi	Open Systems Interconnection (OSI)
pfPup	PUP protocol
pfSna	Linux SNA project
pfUnix	Local to host (old BSD name for local)
pfUnknown1	Unknown
pfUnspec	Unspecified
pfVoiceView	VoiceView

148.2.4 Sockets.TSocketType Enumeration

The TSocketType type defines the set of possible socket types.

Pascal

```
TSocketType = (
  stStream,
  stDgram,
  stRaw,
  stRdm,
  stSeqPacket
);
```

C++

```
enum TSocketType {
  stStream,
  stDgram,
  stRaw,
  stRdm,
  stSeqPacket
};
```

File

Sockets

Description

The following table lists the possible values for the TSocketType type and the meaning corresponding to each value:

Value	Description
stDgram	Single datagram packets, not used by IP. Non-reliable.
stRaw	The socket sends and receives raw datagram packets. Used by IP.
stRdm	Reliably delivered message.
stSeqPacket	Sequenced packet stream sends and receives any type of IP packet.

stStream	Reliable streaming data packets. Used by TCP.

148.3 Types

The following table lists types in this documentation.

Types

Name	Description
TGetThreadEvent (see page 20140)	This is type Sockets.TGetThreadEvent.
TSocketAcceptEvent (see page 20140)	This is type Sockets.TSocketAcceptEvent.
TSocketBlockMode (see page 20141)	Defines the set of possible flags for blocking mode of the socket.
TSocketDataEvent (see page 20141)	This is type Sockets.TSocketDataEvent.
TSocketErrorEvent (see page 20141)	This is type Sockets.TSocketErrorEvent.
TSocketHost (see page 20142)	This is type Sockets.TSocketHost.
TSocketNotifyEvent (see page 20142)	This is type Sockets.TSocketNotifyEvent.
TSocketPort (see page 20142)	This is type Sockets.TSocketPort.
TSocketProtocol (see page 20143)	This is type Sockets.TSocketProtocol.

148.3.1 Sockets.TGetThreadEvent Type

Pascal

```
TGetThreadEvent = procedure (Sender: TObject; var ClientSocketThread: TClientSocketThread) of
object;
```

C++

```
(Sender: TObject; var ClientSocketThread: TClientSocketThread) ( TGetThreadEvent)();
```

File

Sockets

Description

This is type Sockets.TGetThreadEvent.

148.3.2 Sockets.TSocketAcceptEvent Type

Pascal

```
TSocketAcceptEvent = procedure (Sender: TObject; ClientSocket: TCustomIpClient) of object;
```

C++

```
(Sender: TObject; ClientSocket: TCustomIpClient) ( TSocketAcceptEvent)();
```

File

Sockets

Description

This is type Sockets.TSocketAcceptEvent.

148.3.3 Sockets.TSocketBlockMode Type

Defines the set of possible flags for blocking mode of the socket.

Pascal

```
TSocketBlockMode = bmBlocking..bmNonBlocking;
```

C++

```
bmBlocking..bmNonBlocking TSocketBlockMode;
```

File

Sockets

Description

The following table describes TSocketBlockMode values.

Value	Socket state
bmBlocking	The socket is set to block mode.
bmNonBlocking	The socket is set to non-blocking mode.

148.3.4 Sockets.TSocketDataEvent Type

Pascal

```
TSocketDataEvent = procedure (Sender: TObject; Buf: pchar; var DataLen: Integer) of object;
```

C++

```
(Sender: TObject; Buf: pchar; var DataLen: Integer) ( TSocketDataEvent)();
```

File

Sockets

Description

This is type Sockets.TSocketDataEvent.

148.3.5 Sockets.TSocketErrorEvent Type

Pascal

```
TSocketErrorEvent = procedure (Sender: TObject; SocketError: Integer) of object;
```

C++

```
(Sender: TObject; SocketError: Integer) ( TSocketErrorEvent)();
```

File

Sockets

Description

This is type Sockets.TSocketErrorEvent.

148.3.6 Sockets.TSocketHost Type

Pascal

```
TSocketHost = type string;
```

C++

```
type string TSocketHost;
```

File

Sockets

Description

This is type Sockets.TSocketHost.

148.3.7 Sockets.TSocketNotifyEvent Type

Pascal

```
TSocketNotifyEvent = procedure (Sender: TObject) of object;
```

C++

```
(Sender: TObject) ( TSocketNotifyEvent)();
```

File

Sockets

Description

This is type Sockets.TSocketNotifyEvent.

148.3.8 Sockets.TSocketPort Type

Pascal

```
TSocketPort = type string;
```

C++

```
type string TSocketPort;
```

File

Sockets

Description

This is type Sockets.TSocketPort.

148.3.9 Sockets.TSocketProtocol Type

Pascal

```
TSocketProtocol = Word;
```

C++

```
Word TSocketProtocol;
```

File

Sockets

Description

This is type Sockets.TSocketProtocol.

148.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CRLF (see page 20144)	This is constant Sockets.CRLF.

148.4.1 Sockets.CRLF Constant

Pascal

```
CRLF = #13#10;
```

C++

```
CRLF = #13#10;
```

File

Sockets

Description

This is constant Sockets.CRLF.

149 SqlExpr Namespace

This is namespace SqlExpr.

Topics

Name	Description
Classes (see page 20147)	The following table lists classes in this documentation.
Functions (see page 20361)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 20364)	The following table lists structs, records, enums in this documentation.
Types (see page 20371)	The following table lists types in this documentation.
Constants (see page 20375)	The following table lists constants in this documentation.

Classes

	Name	Description
	SPParamDesc (see page 20147)	SPParamDesc describes a parameter to a stored procedure that is accessed using dbExpress.
	TCustomSQLDataSet (see page 20150)	TCustomSQLDataSet is the base class for all dbExpress dataset components.
	TFLDDesc (see page 20205)	This is class SqlExpr.TFLDDesc.
	TSQLBlobStream (see page 20213)	TSQLBlobStream lets applications read from or write to BLOB fields in a client dataset.
	TSQLConnection (see page 20217)	TSQLConnection encapsulates a dbExpress connection to a database server.
	TSQLDataLink (see page 20260)	TSQLDataLink allows a unidirectional dataset act as the detail in a master/detail relationship.
	TSQLDataSet (see page 20265)	TSQLDataSet represents data retrieved using dbExpress.
	TSQLMonitor (see page 20285)	TSQLMonitor intercepts messages that pass between an SQL connection component and a database server and saves them in a string list.
	TSQLQuery (see page 20298)	TSQLQuery represents a query that is executed using dbExpress.
	TSQLStoredProc (see page 20318)	TSQLStoredProc represents a stored procedure that is executed using dbExpress.
	TSQLTable (see page 20337)	TSQLTable represents a database table that is accessed using dbExpress.
	TTransactionItem (see page 20359)	This is class SqlExpr.TTransactionItem.

Constants

Name	Description
BlobTypeMap (see page 20375)	This is constant SqlExpr.BlobTypeMap.
DataTypeMap (see page 20376)	This is constant SqlExpr.DataTypeMap.
DefaultCursor (see page 20376)	This is constant SqlExpr.DefaultCursor.
DefaultMaxBlobSize (see page 20376)	This is constant SqlExpr.DefaultMaxBlobSize.

DefaultRowsetSize (see page 20377)	This is constant SqlExpr.DefaultRowsetSize.
FldSubTypeMap (see page 20377)	This is constant SqlExpr.FldSubTypeMap.
FldTypeMap (see page 20377)	This is constant SqlExpr.FldTypeMap.
HourGlassCursor (see page 20378)	This is constant SqlExpr.HourGlassCursor.
SAnd (see page 20378)	This is constant SqlExpr.SAnd.
SOrderBy (see page 20379)	This is constant SqlExpr.SOrderBy.
SParam (see page 20379)	This is constant SqlExpr.SParam.
SSelect (see page 20379)	This is constant SqlExpr.SSelect.
SSelectSpaces (see page 20380)	This is constant SqlExpr.SSelectSpaces.
SSelectStar (see page 20380)	This is constant SqlExpr.SSelectStar.
SSelectStarFrom (see page 20380)	This is constant SqlExpr.SSelectStarFrom.
SUB_TYPE_MEMO (see page 20381)	This is constant SqlExpr.SUB_TYPE_MEMO.
SWhere (see page 20381)	This is constant SqlExpr.SWhere.
TErrorMessageSize (see page 20381)	This is constant SqlExpr.TErrorMessageSize.

Functions

	Name	Description
	FreeProcParams (see page 20361)	Frees a list of procedure parameters that is allocated by GetProcedureParams.
	GetConnectionNames (see page 20361)	This is function SqlExpr.GetConnectionNames.
	GetConnectionRegistryFile (see page 20362)	This is function SqlExpr.GetConnectionRegistryFile.
	GetDriverNames (see page 20362)	Populates a string list with the names of all available dbExpress drivers.
	GetDriverRegistryFile (see page 20363)	This is function SqlExpr.GetDriverRegistryFile.
	LoadParamListItems (see page 20363)	Fills a TParams object with information from a list of parameter descriptors.

Structs, Records, Enums

	Name	Description
	EConnectFlag (see page 20364)	This is record SqlExpr.EConnectFlag.
	TConnectionState (see page 20365)	TConnectionState describes the state of a TSQLConnection (see page 20217) component.
	TConnectionUserType (see page 20365)	This is record SqlExpr.TConnectionUserType.
	TSchemaType (see page 20366)	TSchemaType and TSchemaInfo describe schema information (metadata) to access using dbXpress.
	TSQLExceptionType (see page 20367)	TSQLExceptionType indicates the type of operation that leads to an ESQLException exception.
	TSQLSchemaInfo (see page 20368)	This is record SqlExpr.TSQLSchemaInfo.
	TTableScope (see page 20368)	TTableScopes (see page 20374) and TTableScopess indicate the types of tables for which to fetch metadata.
	TTransactionDesc (see page 20369)	This is record SqlExpr.TTransactionDesc.
	TTransIsolationLevel (see page 20369)	This enumerated type (in C++) or type (in Delphi) identifies degree to which transactions are isolated from each other.

Types

Name	Description
TFieldDescList (see page 20371)	This is type SqlExpr.TFieldDescList.
TFieldList (see page 20371)	This is type SqlExpr.TFieldList.
TGetDriverFunc (see page 20372)	This is type SqlExpr.TGetDriverFunc.
TLocale (see page 20372)	This is type SqlExpr.TLocale.
TLocaleCode (see page 20372)	This is type SqlExpr.TLocaleCode.
TParseInsertSqlEvent (see page 20373)	This is type SqlExpr.TParseInsertSqlEvent.
TParseSqlEvent (see page 20373)	This is type SqlExpr.TParseSqlEvent.
TSQLConnectionLoginEvent (see page 20373)	This is type SqlExpr.TSQLConnectionLoginEvent.
TTableScopes (see page 20374)	This is type SqlExpr.TTableScopes.
TTraceEvent (see page 20374)	This is type SqlExpr.TTraceEvent.
TTraceLogEvent (see page 20374)	This is type SqlExpr.TTraceLogEvent.

149.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	SPParamDesc (see page 20147)	SPParamDesc describes a parameter to a stored procedure that is accessed using dbExpress.
	TCustomSQLDataSet (see page 20150)	TCustomSQLDataSet is the base class for all dbExpress dataset components.
	TFLDDesc (see page 20205)	This is class SqlExpr.TFLDDesc.
	TSQLBlobStream (see page 20213)	TSQLBlobStream lets applications read from or write to BLOB fields in a client dataset.
	TSQLConnection (see page 20217)	TSQLConnection encapsulates a dbExpress connection to a database server.
	TSQLDataLink (see page 20260)	TSQLDataLink allows a unidirectional dataset act as the detail in a master/detail relationship.
	TSQLDataSet (see page 20265)	TSQLDataSet represents data retrieved using dbExpress.
	TSQLMonitor (see page 20285)	TSQLMonitor intercepts messages that pass between an SQL connection component and a database server and saves them in a string list.
	TSQLQuery (see page 20298)	TSQLQuery represents a query that is executed using dbExpress.
	TSQLStoredProc (see page 20318)	TSQLStoredProc represents a stored procedure that is executed using dbExpress.
	TSQLTable (see page 20337)	TSQLTable represents a database table that is accessed using dbExpress.
	TTransactionItem (see page 20359)	This is class SqlExpr.TTransactionItem.

149.1.1 SPParamDesc Class

SPParamDesc describes a parameter to a stored procedure that is accessed using dbExpress.

Class Hierarchy

SqlExpr.SPParamDesc

Pascal

```
SPParamDesc = class;
```

C++

```
class SPParamDesc;
```

File

SqlExpr

Description

PSPParamDesc is a pointer to a parameter descriptor. SPParamDesc is a parameter descriptor that describes a parameter for a stored procedure. SPParamDesc includes the following members:

Member	Meaning
iParamNum (see page 20149)	The ordinal position of the parameter.
szName (see page 20150)	The name of the parameter

iArgType (see page 20148)	The type of the parameter (input, output, and so on)
iDataType (see page 20149)	The data type of the parameter
iUnits1 (see page 20149)	The size for the datatype (number of characters, digits, etc)
iUnits2 (see page 20150)	Number of decimal places if datatype is BCD
iLen (see page 20149)	The number of bytes needed to store a parameter value

See Also

TSQLConnection

149.1.1.1 SPParamDesc Members

The following tables list the members exposed by SPParamDesc.

Public Fields

	Name	Description
	iArgType (see page 20148)	This is iArgType, a member of class SPParamDesc.
	iDataType (see page 20149)	This is iDataType, a member of class SPParamDesc.
	iLen (see page 20149)	This is iLen, a member of class SPParamDesc.
	iParamNum (see page 20149)	This is iParamNum, a member of class SPParamDesc.
	iUnits1 (see page 20149)	This is iUnits1, a member of class SPParamDesc.
	iUnits2 (see page 20150)	This is iUnits2, a member of class SPParamDesc.
	szName (see page 20150)	This is szName, a member of class SPParamDesc.

149.1.1.2 SPParamDesc Fields

The fields of the SPParamDesc class are listed here.

Public Fields

	Name	Description
	iArgType (see page 20148)	This is iArgType, a member of class SPParamDesc.
	iDataType (see page 20149)	This is iDataType, a member of class SPParamDesc.
	iLen (see page 20149)	This is iLen, a member of class SPParamDesc.
	iParamNum (see page 20149)	This is iParamNum, a member of class SPParamDesc.
	iUnits1 (see page 20149)	This is iUnits1, a member of class SPParamDesc.
	iUnits2 (see page 20150)	This is iUnits2, a member of class SPParamDesc.
	szName (see page 20150)	This is szName, a member of class SPParamDesc.

149.1.1.2.1 SPParamDesc.iArgType Field

Pascal

```
iArgType: TParamType;
```

C++

```
TParamType iArgType;
```

Description

This is iArgType, a member of class SPParamDesc.

149.1.1.2.2 SPParamDesc.iDataType Field

Pascal

```
iDataType: TFieldType;
```

C++

```
TFieldType iDataType;
```

Description

This is iDataType, a member of class SPParamDesc.

149.1.1.2.3 SPParamDesc.iLen Field

Pascal

```
iLen: LongWord;
```

C++

```
LongWord iLen;
```

Description

This is iLen, a member of class SPParamDesc.

149.1.1.2.4 SPParamDesc.iParamNum Field

Pascal

```
iParamNum: Word;
```

C++

```
Word iParamNum;
```

Description

This is iParamNum, a member of class SPParamDesc.

149.1.1.2.5 SPParamDesc.iUnits1 Field

Pascal

```
iUnits1: SmallInt;
```

C++

```
SmallInt iUnits1;
```

Description

This is iUnits1, a member of class SPParamDesc.

149.1.1.2.6 SPParamDesc.iUnits2 Field

Pascal

```
iUnits2: SmallInt;
```

C++

```
SmallInt iUnits2;
```

Description

This is iUnits2, a member of class SPParamDesc.

149.1.1.2.7 SPParamDesc.szName Field

Pascal

```
szName: WideString;
```

C++

```
BSTR szName;
```

Description

This is szName, a member of class SPParamDesc.

149.1.2 TCustomSQLDataSet Class

TCustomSQLDataSet is the base class for all dbExpress dataset components.

Class Hierarchy

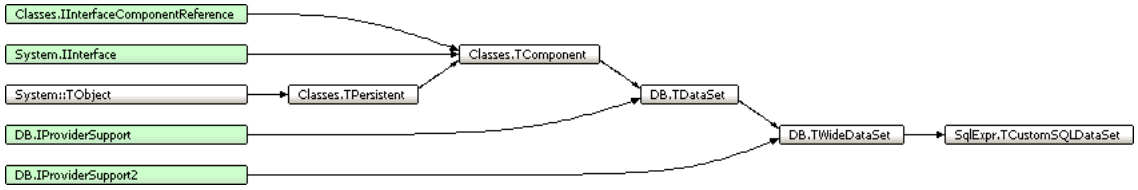

Pascal

```
TCustomSQLDataSet = class(TWideDataSet);
```

C++

```
class TCustomSQLDataSet : public TWideDataSet;
```

File

SqlExpr

Description

TCustomSQLDataSet encapsulates the properties, events, and methods for working with data accessed through dbExpress. Applications should not instantiate TCustomSQLDataSet objects. Instead, instantiate one of the descendant classes: TSQLDataSet (see page 20265), TSQLTable (see page 20337), TSQLQuery (see page 20298), or TSQLStoredProc (see page 20318).

TCustomSQLDataSet and its descendants represent unidirectional datasets. Unlike other datasets, unidirectional datasets do

not buffer multiple records in memory. Because of this, they have the following limitations:

There is no searching or navigation support except for moving to the first record or moving to the next record.

There is no built-in editing support. The data underlying an SQL dataset can only be edited by explicitly creating an SQL UPDATE command or by connecting the SQL dataset to a client dataset using a provider.

There is no support for filtering records or for lookup fields.

See Also

TSQLDataSet (see page 20265)

TSQLQuery (see page 20298)

TSQLStoredProc (see page 20318)

TSQLTable (see page 20337)

TSQLConnection (see page 20217)

TClientDataSEt

TDataSEtProvider

149.1.2.1 TCustomSQLDataSet Members

The following tables list the members exposed by TCustomSQLDataSet.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	AddFieldDesc (see page 20163)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 20163)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
V	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
V	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
V	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
	CheckPrepareError (see page 20164)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
V	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
V	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
	ClearIndexDefs (see page 20164)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
V	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
V	CloseCursor (see page 20164)	This is CloseCursor, a member of class TCustomSQLDataSet.

	CloseStatement (see page 20164)	This is CloseStatement, a member of class TCustomSQLDataSet.
	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
	DataEvent (see page 6924)	This is DataEvent, a member of class TDataSet.
	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
	DefineProperties (see page 20165)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	ExecSQL (see page 20165)	This is ExecSQL, a member of class TCustomSQLDataSet.
	ExecuteStatement (see page 20166)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FreeBuffers (see page 20166)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCommand (see page 20166)	This is FreeCommand, a member of class TCustomSQLDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeReader (see page 20166)	This is FreeReader, a member of class TCustomSQLDataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 20167)	This is GetCanModify, a member of class TCustomSQLDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
	GetDataSource (see page 20167)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).

	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
V	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
V	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
V	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetObjectTypeNames (see page 20169)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 20169)	This is GetOutputParams, a member of class TCustomSQLDataSet.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
V	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
V	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
V	GetRecord (see page 20170)	This is GetRecord, a member of class TCustomSQLDataSet.
V	GetRecordCount (see page 20170)	This is GetRecordCount, a member of class TCustomSQLDataSet.
V	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
	GetSortFieldNames (see page 20170)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
V	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
V	InitRecord (see page 20170)	This is InitRecord, a member of class TCustomSQLDataSet.
V	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
V	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
V	InternalClose (see page 20171)	This is InternalClose, a member of class TCustomSQLDataSet.
V	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
V	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
V	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
V	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
V	InternalHandleException (see page 20171)	This is InternalHandleException, a member of class TCustomSQLDataSet.
V	InternalInitFieldDefs (see page 20171)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
V	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
V	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
V	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
V	InternalOpen (see page 20171)	This is InternalOpen, a member of class TCustomSQLDataSet.
V	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
V	InternalRefresh (see page 20171)	This is InternalRefresh, a member of class TCustomSQLDataSet.
V	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsCursorOpen (see page 20172)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
V	Loaded (see page 20172)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 20173)	This is LocateRecord, a member of class TCustomSQLDataSet.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	OpenCursor (see page 20174)	This is OpenCursor, a member of class TCustomSQLDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
V	OpenSchema (see page 20174)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 20175)	This is PropertyChanged, a member of class TCustomSQLDataSet.
V	PSEndTransaction (see page 20175)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSExecute (see page 20176)	Executes the SQL command associated with the dataset.
V	PSExecuteStatement (see page 20176)	This is PSExecuteStatement, a member of class TCustomSQLDataSet.
V	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
V	PSExecuteStatement (see page 7535)	This is the overview for the PSExecuteStatement method overload.

	PSGetAttributes (see page 20176)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 20177)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandTextW (see page 7537)	This is PSGetCommandTextW, a member of class TWideDataSet.
	PSGetCommandType (see page 20177)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 20178)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 20178)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7538)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 20179)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
	PSGetParams (see page 20179)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7538)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 20179)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7539)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 20179)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
	PSGetUpdateException (see page 20180)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 20180)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 20180)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 20181)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 20181)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 20182)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetCommandText (see page 7540)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 20182)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 20182)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 20183)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 20184)	This is SetBufListSize, a member of class TCustomSQLDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCommandText (see page 20184)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFCommandText (see page 20184)	This is SetFCommandText, a member of class TCustomSQLDataSet.
	SetFieldData (see page 20184)	This is SetFieldData, a member of class TCustomSQLDataSet.

V	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
V	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
V	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
V	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetMaxBlobSize (see page 20184)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
V	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetParamsFromCursor (see page 20185)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
	SetSortFieldNames (see page 20186)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
V	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateIndexDefs (see page 20186)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
	BlobBuffer (see page 20189)	Reserves a buffer for storing BLOB values.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
	CommandType (see page 20190)	Indicates the meaning of the CommandText (see page 20189) property.
	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
	CurrentBlobSize (see page 20191)	Indicates the size of the most recently read BLOB field.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
R	DataLink (see page 20191)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	DataSource (see page 20192)	Links the SQL dataset to another (master) dataset.

	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
R	InternalConnection (see page 20194)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	LastError (see page 20194)	Saves the error message for the last error returned by dbExpress.
	MaxBlobSize (see page 20194)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	NativeCommand (see page 20195)	Represents the SQL command that is sent to the server.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
	ParamCheck (see page 20196)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
R	ParamCount (see page 20197)	Indicates the number of parameters listed by the Params (see page 20197) property.
	Params (see page 20197)	Represents the parameters for a query or stored procedure.
	ProcParams (see page 20199)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
	Reserved (see page 7035)	For internal use only.
R	RowsAffected (see page 20200)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 20200)	Identifies the metadata with which to populate the dataset.
	SortFieldNames (see page 20201)	Indicates the sort order when CommandType (see page 20190) is ctTable.

Public Events

	Name	Description
	AfterCancel (see page 7038)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterDelete (see page 7041)	Occurs after an application deletes a record.
	AfterEdit (see page 7044)	Occurs after an application starts editing a record.
	AfterInsert (see page 7044)	Occurs after an application inserts a new record.
	AfterPost (see page 7046)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	BeforeCancel (see page 7048)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeDelete (see page 7049)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 7049)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 7050)	Occurs before an application enters insert mode.
	BeforePost (see page 7052)	Occurs before an application posts changes for the active record to the database or change log.
	OnDeleteError (see page 7054)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 7054)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 7056)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 7056)	Occurs when an application attempts to modify or insert a record and an exception is raised.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.

	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Close (see page 6920)	Closes a dataset.
	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
	Create (see page 20160)	Creates an instance of a TCustomSQLDataSet (see page 20150) component.
	CreateBlobStream (see page 20164)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
	Destroy (see page 20161)	Destroys the TCustomSQLDataSet (see page 20150) instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetBlobFieldData (see page 20166)	Retrieves the current value of a BLOB field into a buffer.
	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.

	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
	GetDetailLinkFields (see page 20167)	Lists the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
	GetFieldData (see page 20168)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetKeyFieldNames (see page 20169)	Fills a list with the names of all fields in all indexes defined for the dataset.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetQuoteChar (see page 20169)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
	IsSequenced (see page 20172)	Indicates whether the dataset can use record numbers to indicate the order of records.
	Last (see page 6968)	Moves to the last record in the dataset.
	Locate (see page 20173)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Lookup (see page 20174)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	Open (see page 6976)	Opens the dataset.
	ParamByName (see page 20175)	This is ParamByName, a member of class TCustomSQLDataSet.
	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
	PrepareStatement (see page 20175)	Performs a prepare on the SQL statement.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7593)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7594)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.

	PSGetTableName (see page 7594)	Returns the name of the dataset table that appears in generated SQL statements.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetSchemaInfo (see page 20185)	Indicates whether the dataset represents metadata from the server, and if so, what kind.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.

Public Properties

	Name	Description
R	AggFields (see page 7006)	Contains any aggregate fields.
	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
	CommandText (see page 20189)	Specifies the command the dataset executes.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignerData (see page 20192)	Stores custom data.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
	FieldDefs (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.

	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
	IndexDefs (see page 20193)	Contains the definitions of all indexes defined on the server for the dataset.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	Modified (see page 7029)	Indicates whether the active record is modified.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Prepared (see page 20198)	Specifies whether the command is prepared before execution.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 20199)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
	SQLConnection (see page 20201)	Specifies the SQL connection component that connects the dataset to a database server.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	TransactionLevel (see page 20202)	Indicates the transaction to which this dataset belongs.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterClose (see page 20203)	Occurs after a data source connection is closed.
	AfterOpen (see page 20203)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 20203)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 20204)	Occurs after a result set is scrolled.
	BeforeClose (see page 20204)	Occurs just before a data connection is closed.
	BeforeOpen (see page 20204)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 20204)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 20204)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 20204)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 20205)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 20189) property.
	ParseInsertSql (see page 20205)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 20189) property.
	ParseSelectSql (see page 20205)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 20189) property.
	ParseUpdateSql (see page 20205)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 20189) property.

Published Properties

	Name	Description
	Active (see page 20187)	Specifies whether or not a dataset is open.
	GetMetadata (see page 20193)	Specifies whether the SQL dataset fetches metadata information along with the data.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NumericMapping (see page 20195)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 20196)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	SchemaName (see page 20201)	Specifies the name of the database schema.
	Tag (see page 2808)	Stores an integer value as part of a component.

149.1.2.2 TCustomSQLDataSet.Create Constructor

Creates an instance of a TCustomSQLDataSet (see page 20150) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomSQLDataSet(TComponent * AOwner);
```

Description

Call Create to instantiate an SQL dataset component at runtime. SQL datasets placed in forms or data modules at design time are created automatically.

TCustomSQLDataSet (see page 20150) is rarely used directly in applications because many of its properties are protected. Instead, applications instantiate TCustomSQLDataSet (see page 20150) descendants such as TSQLDataSet (see page 20265), TSQLTable (see page 20337), TSQLQuery (see page 20298), or TSQLStoredProc (see page 20318).

AOwner is the component that is responsible for freeing the dataset. It becomes the value of the Owner property.

See Also

Owner (see page 2807)

Create (see page 6904)

TSQLDataSet (see page 20265)

TSQLStoredProc (see page 20318)

TSQLTable (see page 20337)

TSQLQuery (see page 20298)

149.1.2.3 TCustomSQLDataSet.Destroy Destructor

Destroys the TCustomSQLDataSet (see page 20150) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomSQLDataSet();
```

Description

Do not call Destroy directly in an application. Usually destruction of datasets is handled automatically. If an application creates its own instances of a dataset, however, and does not assign an Owner that is responsible for freeing the dataset, then the application should call Free, which verifies that the dataset reference is not nil before calling Destroy.

See Also

Owner (see page 2807)

Create (see page 20160)

Free (see page 22300)

149.1.2.4 TCustomSQLDataSet Methods

The methods of the TCustomSQLDataSet class are listed here.

Protected Methods

	Name	Description
	AddFieldDesc (see page 20163)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 20163)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
	CheckPrepareError (see page 20164)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
	ClearIndexDefs (see page 20164)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
V	CloseCursor (see page 20164)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 20164)	This is CloseStatement, a member of class TCustomSQLDataSet.
V	DefineProperties (see page 20165)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	ExecSQL (see page 20165)	This is ExecSQL, a member of class TCustomSQLDataSet.
	ExecuteStatement (see page 20166)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FreeBuffers (see page 20166)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCommand (see page 20166)	This is FreeCommand, a member of class TCustomSQLDataSet.
	FreeReader (see page 20166)	This is FreeReader, a member of class TCustomSQLDataSet.
V	GetCanModify (see page 20167)	This is GetCanModify, a member of class TCustomSQLDataSet.
V	GetDataSource (see page 20167)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetObjectTypeNames (see page 20169)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 20169)	This is GetOutputParams, a member of class TCustomSQLDataSet.
V	GetRecord (see page 20170)	This is GetRecord, a member of class TCustomSQLDataSet.
V	GetRecordCount (see page 20170)	This is GetRecordCount, a member of class TCustomSQLDataSet.
	GetSortFieldNames (see page 20170)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
V	InitRecord (see page 20170)	This is InitRecord, a member of class TCustomSQLDataSet.
V	InternalClose (see page 20171)	This is InternalClose, a member of class TCustomSQLDataSet.
V	InternalHandleException (see page 20171)	This is InternalHandleException, a member of class TCustomSQLDataSet.
V	InternalInitFieldDefs (see page 20171)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
V	InternalOpen (see page 20171)	This is InternalOpen, a member of class TCustomSQLDataSet.
V	InternalRefresh (see page 20171)	This is InternalRefresh, a member of class TCustomSQLDataSet.
V	IsCursorOpen (see page 20172)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
V	Loaded (see page 20172)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 20173)	This is LocateRecord, a member of class TCustomSQLDataSet.
V	OpenCursor (see page 20174)	This is OpenCursor, a member of class TCustomSQLDataSet.
V	OpenSchema (see page 20174)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PropertyChanged (see page 20175)	This is PropertyChanged, a member of class TCustomSQLDataSet.
V	PSEndTransaction (see page 20175)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSExecute (see page 20176)	Executes the SQL command associated with the dataset.
V	PSExecuteStatement (see page 20176)	This is PSExecuteStatement, a member of class TCustomSQLDataSet.
V	PSGetAttributes (see page 20176)	Returns information that is stored in data packets as Name/Value pairs.
V	PSGetCommandText (see page 20177)	Gets the SQL command to be executed by PSExecute.
V	PSGetCommandType (see page 20177)	Gets the command type to be executed by PSExecute.
V	PSGetDefaultOrder (see page 20178)	Returns the definition of an index that imposes the default order on the data included in data packets.
V	PSGetIndexDefs (see page 20178)	Returns the definitions of all specified indexes defined for the dataset.
V	PSGetKeyFieldsW (see page 20179)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
V	PSGetParams (see page 20179)	Returns the current parameter values of the dataset.
V	PSGetQuoteCharW (see page 20179)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
V	PSGetTableNameW (see page 20179)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
V	PSGetUpdateException (see page 20180)	Generates an EUpdateError object based on another exception object.
V	PSInTransaction (see page 20180)	Indicates whether actions against the dataset occur in the context of a transaction.
V	PSIsSQLBased (see page 20180)	Indicates whether the dataset includes native SQL support.
V	PSIsSQLSupported (see page 20181)	Indicates whether the dataset can execute SQL statements.
V	PSReset (see page 20181)	Resets the dataset so that it is positioned on the first record.
V	PSSetCommandText (see page 20182)	Assigns a specified SQL command to be executed by PSExecute.
V	PSSetParams (see page 20182)	Assigns the specified parameter values to the dataset.

	PSStartTransaction (see page 20182)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 20183)	Applies a single update to the underlying database table or tables.
	SetBufListSize (see page 20184)	This is SetBufListSize, a member of class TCustomSQLDataSet.
	SetCommandText (see page 20184)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetFCommandText (see page 20184)	This is SetFCommandText, a member of class TCustomSQLDataSet.
	SetFieldData (see page 20184)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetMaxBlobSize (see page 20184)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetParamsFromCursor (see page 20185)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetSortFieldNames (see page 20186)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	UpdateIndexDefs (see page 20186)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

Public Methods

	Name	Description
	CreateBlobStream (see page 20164)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	GetBlobFieldData (see page 20166)	Retrieves the current value of a BLOB field into a buffer.
	GetDetailLinkFields (see page 20167)	Lists the field components that link this dataset as a detail of a master dataset.
	GetFieldData (see page 20168)	This is the overview for the GetFieldData method overload.
	GetKeyFieldNames (see page 20169)	Fills a list with the names of all fields in all indexes defined for the dataset.
	GetQuoteChar (see page 20169)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	IsSequenced (see page 20172)	Indicates whether the dataset can use record numbers to indicate the order of records.
	Locate (see page 20173)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Lookup (see page 20174)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	ParamByName (see page 20175)	This is ParamByName, a member of class TCustomSQLDataSet.
	PrepareStatement (see page 20175)	Performs a prepare on the SQL statement.
	SetSchemaInfo (see page 20185)	Indicates whether the dataset represents metadata from the server, and if so, what kind.

149.1.2.4.1 TCustomSQLDataSet.AddFieldDesc Method

Pascal

```
procedure AddFieldDesc(FieldDescs: TFieldDescList; DescNo: Integer; var FieldID: Integer;
RequiredFields: TBits; FieldDefs: TFieldDefs);
```

C++

```
__fastcall AddFieldDesc(TFieldDescList FieldDescs, int DescNo, int FieldID, TBits
RequiredFields, TFieldDefs FieldDefs);
```

Description

This is AddFieldDesc, a member of class TCustomSQLDataSet.

149.1.2.4.2 TCustomSQLDataSet.AddIndexDefs Method

Pascal

```
procedure AddIndexDefs(SourceDS: TCustomSQLDataSet; IndexName: string = '');
```

C++

```
__fastcall AddIndexDefs(TCustomSQLDataSet SourceDS, AnsiString IndexName = '');
```

Description

This is AddIndexDefs, a member of class TCustomSQLDataSet.

149.1.2.4.3 TCustomSQLDataSet.CheckPrepareError Method

Pascal

```
procedure CheckPrepareError;
```

C++

```
__fastcall CheckPrepareError();
```

Description

This is CheckPrepareError, a member of class TCustomSQLDataSet.

149.1.2.4.4 TCustomSQLDataSet.ClearIndexDefs Method

Pascal

```
procedure ClearIndexDefs;
```

C++

```
__fastcall ClearIndexDefs();
```

Description

This is ClearIndexDefs, a member of class TCustomSQLDataSet.

149.1.2.4.5 TCustomSQLDataSet.CloseCursor Method

Pascal

```
procedure CloseCursor; override;
```

C++

```
virtual __fastcall CloseCursor();
```

Description

This is CloseCursor, a member of class TCustomSQLDataSet.

149.1.2.4.6 TCustomSQLDataSet.CloseStatement Method

Pascal

```
procedure CloseStatement;
```

C++

```
__fastcall CloseStatement();
```

Description

This is CloseStatement, a member of class TCustomSQLDataSet.

149.1.2.4.7 TCustomSQLDataSet.CreateBlobStream Method

Creates a blob stream for a Binary large object (BLOB) field in the dataset.

Pascal

```
function CreateBlobStream(Field: TField; Mode: TBlobStreamMode): TStream; override;
```

C++

```
virtual __fastcall TStream CreateBlobStream(TField Field, TBlobStreamMode Mode);
```

Description

Call CreateBlobStream to obtain a stream for reading and writing the value of the field specified by the Field parameter.

Field is a string containing the name of the BLOB field in the dataset. It must be an instance of TBlobField or one of its descendants.

Mode indicates whether the stream will be used for reading the field's value (bmRead), writing the field's value (bmWrite), or both (bmReadWrite).

CreateBlobStream returns the stream you can use for reading or writing data.

See Also

TSQLBlobStream (see page 20213)

TBlobField (see page 6801)

149.1.2.4.8 TCustomSQLDataSet.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor (see page 2839)

DefineBinaryProperty (see page 2837)

DefineProperties (see page 2931)

DefineProperty (see page 2838)

149.1.2.4.9 TCustomSQLDataSet.ExecSQL Method

Pascal

```
function ExecSQL(ExecDirect: Boolean = False): Integer; virtual;
```

C++

```
virtual __fastcall int ExecSQL(Boolean ExecDirect = False);
```

Description

This is ExecSQL, a member of class TCustomSQLDataSet.

149.1.2.4.10 TCustomSQLDataSet.ExecuteStatement Method

Pascal

```
procedure ExecuteStatement;
```

C++

```
__fastcall ExecuteStatement();
```

Description

This is ExecuteStatement, a member of class TCustomSQLDataSet.

149.1.2.4.11 TCustomSQLDataSet.FreeBuffers Method

Pascal

```
procedure FreeBuffers;
```

C++

```
__fastcall FreeBuffers();
```

Description

This is FreeBuffers, a member of class TCustomSQLDataSet.

149.1.2.4.12 TCustomSQLDataSet.FreeCommand Method

Pascal

```
procedure FreeCommand;
```

C++

```
__fastcall FreeCommand();
```

Description

This is FreeCommand, a member of class TCustomSQLDataSet.

149.1.2.4.13 TCustomSQLDataSet.FreeReader Method

Pascal

```
procedure FreeReader;
```

C++

```
__fastcall FreeReader();
```

Description

This is FreeReader, a member of class TCustomSQLDataSet.

149.1.2.4.14 TCustomSQLDataSet.GetBlobFieldData Method

Retrieves the current value of a BLOB field into a buffer.

Pascal

```
function GetBlobFieldData(FieldNo: Integer; var Buffer: TBlobByteData): Integer; override;
```

C++

```
virtual __fastcall int GetBlobFieldData(int FieldNo, TBlobByteData Buffer);
```

Description

This method is used internally to read BLOB data from the field specified by FieldNo into a Buffer. The buffer is a dynamic array of bytes, so that it can grow to accommodate the size of the BLOB data. GetBlobFieldData returns the size of the buffer.

GetBlobFieldData is intended primarily for internal use. Applications should use the BLOB stream returned by the CreateBlobStream (see page 20164) method to access BLOB data.

See Also

GetFieldData (see page 20168)

IsBlob (see page 7136)

CreateBlobStream (see page 20164)

GetBlobSize

149.1.2.4.15 TCustomSQLDataSet.GetCanModify Method

Pascal

```
function GetCanModify: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetCanModify();
```

Description

This is GetCanModify, a member of class TCustomSQLDataSet.

149.1.2.4.16 TCustomSQLDataSet.GetDataSource Method

Pascal

```
function GetDataSource: TDataSource; override;
```

C++

```
virtual __fastcall TDataSource GetDataSource();
```

Description

This is GetDataSource, a member of class TCustomSQLDataSet.

149.1.2.4.17 TCustomSQLDataSet.GetDetailLinkFields Method

Lists the field components that link this dataset as a detail of a master dataset.

Pascal

```
procedure GetDetailLinkFields(MasterFields: TList; DetailFields: TList); override;
```

C++

```
virtual __fastcall GetDetailLinkFields(TList MasterFields, TList DetailFields);
```

Description

GetDetailLInkFIelds fills the two lists with field components that define a master-detail relationship between this dataset and another (master) dataset. The MasterFields list is filled with fields from the master table whose values must equal the values of the fields in the DetailFields list. The DetailFields list is filled with fields from this dataset.

Applications seldom call GetDetailLinkFields. It is used internally by the Field Link designer when setting up master/detail relationships.

See Also

DataSource (see page 20192)

MasterFields (see page 20356)

MasterSource (see page 20357)

149.1.2.4.18 GetFieldData Method

This is the overview for the GetFieldData method overload.

Overload List

	Name	Description
	TCustomSQLDataSet.GetFieldData (Integer, Pointer) (see page 20168)	Retrieves the current value of a field into a buffer.
	TCustomSQLDataSet.GetFieldData (TField, Pointer) (see page 20168)	Retrieves the current value of a field into a buffer.

149.1.2.4.18.1 TCustomSQLDataSet.GetFieldData Method (Integer, Pointer)

Retrieves the current value of a field into a buffer.

Pascal

```
function GetFieldData(FieldNo: Integer; Buffer: Pointer): Boolean; override; overload;
function GetFieldData(Field: TField; Buffer: Pointer): Boolean; override; overload;
```

C++

```
virtual __fastcall Boolean GetFieldData(int FieldNo, void * Buffer);
virtual __fastcall Boolean GetFieldData(TField Field, void * Buffer);
```

Description

Most applications do not need to call GetFieldData. TField objects call this method to implement their GetData method.

The Field or FieldNo parameter indicates the field whose data should be fetched. Field specifies the component itself, while FieldNo indicates its field number.

Buffer is a memory buffer with sufficient space to accept the value of the field as it exists in the database (unformatted and untranslated). GetFieldData returns a value that indicates whether the data was successfully fetched.

GetFieldData returns a value that indicates whether the data was successfully fetched.

See Also

GetBlobFieldData (see page 20166)

IsBlob (see page 7136)

GetData (see page 7131)

149.1.2.4.19 TCustomSQLDataSet.GetKeyFieldNames Method

Fills a list with the names of all fields in all indexes defined for the dataset.

Pascal

```
function GetKeyFieldNames(List: TWideStrings): Integer; overload;
```

C++

```
__fastcall int GetKeyFieldNames(TWideStrings List);
```

Description

Call GetKeyFieldNames to fill a string list with information about the indexes that are available on the database server for this dataset.

For every index defined on the server for the dataset, GetKeyFieldNames adds a string to the string list specified by List. Each string is a semicolon-delimited set of field names that lists the fields in a single index.

TCustomSQLDataSet (see page 20150) does not explicitly use these indexes. GetKeyFieldNames can be useful for composing SQL commands dynamically that need to use an index to access fields.

See Also

IndexDefs (see page 20193)

149.1.2.4.20 TCustomSQLDataSet.GetObjectTypeNames Method

Pascal

```
procedure GetObjectTypeNames(Fields: TFields);
```

C++

```
__fastcall GetObjectTypeNames(TFields Fields);
```

Description

This is GetObjectTypeNames, a member of class TCustomSQLDataSet.

149.1.2.4.21 TCustomSQLDataSet.GetOutputParams Method

Pascal

```
procedure GetOutputParams(AProcParams: TList = nil);
```

C++

```
__fastcall GetOutputParams(TList AProcParams = nil);
```

Description

This is GetOutputParams, a member of class TCustomSQLDataSet.

149.1.2.4.22 TCustomSQLDataSet.GetQuoteChar Method

Returns the character or characters to use in generated SQL statements to enclose quoted strings.

Pascal

```
function GetQuoteChar: WideString; virtual;
```

C++

```
virtual __fastcall BSTR GetQuoteChar();
```

Description

GetQuoteChar returns the character or characters that delimit quoted strings in automatically generated SQL statements. This string appears both before and after quoted strings.

149.1.2.4.23 TCustomSQLDataSet.GetRecord Method

Pascal

```
function GetRecord(Buffer: TRecordBuffer; GetMode: TGetMode; DoCheck: Boolean): TGetResult;
override;
```

C++

```
virtual __fastcall TGetResult GetRecord(TRecordBuffer Buffer, TGetMode GetMode, Boolean
DoCheck);
```

Description

This is GetRecord, a member of class TCustomSQLDataSet.

149.1.2.4.24 TCustomSQLDataSet.GetRecordCount Method

Pascal

```
function GetRecordCount: Integer; override;
```

C++

```
virtual __fastcall int GetRecordCount();
```

Description

This is GetRecordCount, a member of class TCustomSQLDataSet.

149.1.2.4.25 TCustomSQLDataSet.GetSortFieldNames Method

Pascal

```
function GetSortFieldNames: WideString;
```

C++

```
__fastcall BSTR GetSortFieldNames();
```

Description

This is GetSortFieldNames, a member of class TCustomSQLDataSet.

149.1.2.4.26 TCustomSQLDataSet.InitRecord Method

Pascal

```
procedure InitRecord(Buffer: TRecordBuffer); override;
```

C++

```
virtual __fastcall InitRecord(TRecordBuffer Buffer);
```

Description

This is InitRecord, a member of class TCustomSQLDataSet.

149.1.2.4.27 TCustomSQLDataSet.InternalClose Method

Pascal

```
procedure InternalClose; override;
```

C++

```
virtual __fastcall InternalClose();
```

Description

This is InternalClose, a member of class TCustomSQLDataSet.

149.1.2.4.28 TCustomSQLDataSet.InternalHandleException Method

Pascal

```
procedure InternalHandleException; override;
```

C++

```
virtual __fastcall InternalHandleException();
```

Description

This is InternalHandleException, a member of class TCustomSQLDataSet.

149.1.2.4.29 TCustomSQLDataSet.InternalInitFieldDefs Method

Pascal

```
procedure InternalInitFieldDefs; override;
```

C++

```
virtual __fastcall InternalInitFieldDefs();
```

Description

This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.

149.1.2.4.30 TCustomSQLDataSet.InternalOpen Method

Pascal

```
procedure InternalOpen; override;
```

C++

```
virtual __fastcall InternalOpen();
```

Description

This is InternalOpen, a member of class TCustomSQLDataSet.

149.1.2.4.31 TCustomSQLDataSet.InternalRefresh Method

Pascal

```
procedure InternalRefresh; override;
```

C++

```
virtual __fastcall InternalRefresh();
```

Description

This is InternalRefresh, a member of class TCustomSQLDataSet.

149.1.2.4.32 TCustomSQLDataSet.IsCursorOpen Method

Pascal

```
function IsCursorOpen: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCursorOpen();
```

Description

This is IsCursorOpen, a member of class TCustomSQLDataSet.

149.1.2.4.33 TCustomSQLDataSet.IsSequenced Method

Indicates whether the dataset can use record numbers to indicate the order of records.

Pascal

```
function IsSequenced: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsSequenced();
```

Description

Datasets use IsSequenced to indicate whether the RecNo property corresponds to the order of records. When IsSequenced returns true, applications can safely use the RecNo property to navigate to records in the dataset.

Note: For TCustomSQLDataSet (see page 20150), IsSequenced always returns false because TCustomSQLDataSet (see page 20150) descendants do not support record numbers at all.

See Also

RecNo (see page 7031)

RecordCount (see page 20199)

149.1.2.4.34 TCustomSQLDataSet.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the

components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState (see page 2804)

Create (see page 2766)

TReader (see page 2933)

149.1.2.4.35 TCustomSQLDataSet.Locate Method

Implements a virtual method for searching a dataset for a specified record and making it the active record.

Pascal

```
function Locate(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions):
Boolean; override;
```

C++

```
virtual __fastcall Boolean Locate(const AnsiString KeyFields, const Variant KeyValues,
TLocateOptions Options);
```

Description

This function:

Checks whether the dataset is unidirectional, and if so, raises an EDatabaseError exception.

Returns false, indicating that a matching record was not found and the active record was not changed.

Descendant classes that are not unidirectional override this method so that it locates the record where the fields identified by the semicolon-separated list of fields in KeyFields have the values specified by the Variant or Variant array KeyValues. Options indicates whether the search is case insensitive and whether partial matches are supported. Locate returns true if a record is found that matches the specified criteria and that record is now active.

See Also

FindField (see page 6938)

Lookup (see page 6970)

149.1.2.4.36 TCustomSQLDataSet.LocateRecord Method

Pascal

```
function LocateRecord(const KeyFields: string; const KeyValues: Variant; Options:
TLocateOptions; SyncCursor: Boolean): Boolean;
```

C++

```
__fastcall Boolean LocateRecord(const AnsiString KeyFields, const Variant KeyValues,
TLocateOptions Options, Boolean SyncCursor);
```

Description

This is LocateRecord, a member of class TCustomSQLDataSet.

149.1.2.4.37 TCustomSQLDataSet.Lookup Method

Implements a virtual method to retrieve field values from a record that matches specified search values.

Pascal

```
function Lookup(const KeyFields: string; const KeyValues: Variant; const ResultFields:
string): Variant; override;
```

C++

```
virtual __fastcall Variant Lookup(const AnsiString KeyFields, const Variant KeyValues, const
AnsiString ResultFields);
```

Description

This function:

Checks whether the dataset is unidirectional, and if so, raises an EDatabaseError exception.

Returns a Variant with the value false, indicating that a matching record was not found.

Descendant classes that are not unidirectional override this method so that it locates the record where the fields identified by the comma-delimited string KeyFields have the values specified by the Variant or Variant array KeyValues. In classes that implement Lookup, it returns a Variant or Variant array that contains the value or values of the fields specified by the comma-delimited string ResultFields on the specified record.

See Also

Locate (see page 20173)

149.1.2.4.38 TCustomSQLDataSet.OpenCursor Method

Pascal

```
procedure OpenCursor(InfoQuery: Boolean); override;
```

C++

```
virtual __fastcall OpenCursor(Boolean InfoQuery);
```

Description

This is OpenCursor, a member of class TCustomSQLDataSet.

149.1.2.4.39 TCustomSQLDataSet.OpenSchema Method

Pascal

```
procedure OpenSchema; virtual;
```

C++

```
virtual __fastcall OpenSchema();
```

Description

This is OpenSchema, a member of class TCustomSQLDataSet.

149.1.2.4.40 TCustomSQLDataSet.ParamByName Method

Pascal

```
function ParamByName(const Value: string): TParam;
```

C++

```
__fastcall TParam ParamByName(const AnsiString Value);
```

Description

This is ParamByName, a member of class TCustomSQLDataSet.

149.1.2.4.41 TCustomSQLDataSet.PrepareStatement Method

Performs a prepare on the SQL statement.

Pascal

```
procedure PrepareStatement; virtual;
```

C++

```
virtual __fastcall PrepareStatement();
```

Description

The prepare operation is a type of pre-compile that checks the performance of the SQL statement. Some databases support a prepare statement that creates an execution plan that speeds processing.

149.1.2.4.42 TCustomSQLDataSet.PropertyChanged Method

Pascal

```
procedure PropertyChanged;
```

C++

```
__fastcall PropertyChanged();
```

Description

This is PropertyChanged, a member of class TCustomSQLDataSet.

149.1.2.4.43 TCustomSQLDataSet.PSEndTransaction Method

Ends a transaction that was started by a call to PSStartTransaction.

Pascal

```
procedure PSEndTransaction(Commit: Boolean); override;
```

C++

```
virtual __fastcall PSEndTransaction(Boolean Commit);
```

Description

The provider component applies updates within a transaction, if possible. To do so, it uses PSEndTransaction in conjunction with the PSStartTransaction method.

The Commit parameter indicates whether the dataset should commit the current transaction (Commit is true), or roll it back (Commit is false).

Note: TDataSetProvider only starts a transaction (Using PSStartTransaction) if the PSInTransaction method indicates that there is no transaction already established. That is, it does not assume nested transaction support is available. Thus, there is no need to ensure that PSEndTransaction does not terminate a transaction that was not started by the provider.

See Also

PSStartTransaction (see page 7586)

PSInTransaction (see page 7583)

applyupdates (see page 18459)

149.1.2.4.44 TCustomSQLDataSet.PSExecute Method

Executes the SQL command associated with the dataset.

Pascal

```
procedure PSExecute; override;
```

C++

```
virtual __fastcall PSExecute();
```

Description

The provider component calls PSExecute to implement its Execute method. If the dataset represents a query or stored procedure that does not return a cursor, this method executes the query or stored procedure. If the provider Options includes poAllowCommandText and the dataset supports the PSSetCommandText method, the statement specified by PSSetCommandText may be executed instead.

If there is not statement to execute, PSExecute raises an EDatabaseError exception.

See Also

PSSetCommandText (see page 7585)

Execute (see page 18466)

Options (see page 18440)

149.1.2.4.45 TCustomSQLDataSet.PSExecuteStatement Method

Pascal

```
function PSExecuteStatement(const ASQL: WideString; AParams: TParams; ResultSet: TPSResult =
nil): Integer; override;
```

C++

```
virtual __fastcall int PSExecuteStatement(const BSTR ASQL, TParams AParams, TPSResult
ResultSet = nil);
```

Description

This is PSExecuteStatement, a member of class TCustomSQLDataSet.

149.1.2.4.46 TCustomSQLDataSet.PSGetAttributes Method

Returns information that is stored in data packets as Name/Value pairs.

Pascal

```
procedure PSGetAttributes(List: TList); override;
```

C++

```
virtual __fastcall PSGetAttributes(TList List);
```

Description

The provider component calls PSGetAttributes to add information about the dataset to the metadata stored with generated data packets. This information is stored as Name/Value pairs that are added to the list specified by the List parameter.

See Also

DoGetProviderAttributes (see page 18521)

149.1.2.4.47 TCustomSQLDataSet.PSGetCommandText Method

Gets the SQL command to be executed by PSExecute.

Pascal

```
function PSGetCommandText: string; override;
```

C++

```
virtual __fastcall AnsiString PSGetCommandText();
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

PSSetCommandText (see page 7585)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

149.1.2.4.48 TCustomSQLDataSet.PSGetCommandType Method

Gets the command type to be executed by PSExecute.

Pascal

```
function PSGetCommandType: TPSCommandType; override;
```

C++

```
virtual __fastcall TPSCommandType PSGetCommandType();
```

Description

PSGetCommandType gets the command type executed by PSExecute. The type may be a type of SQL command or other command types, such as stored procedures.

See Also

CommandText (see page 8465)

PSExecute (see page 7578)

PSGetCommandText (see page 7579)

Execute (see page 18466)

Options (see page 18440)

149.1.2.4.49 TCustomSQLDataSet.PSGetDefaultOrder Method

Returns the definition of an index that imposes the default order on the data included in data packets.

Pascal

```
function PSGetDefaultOrder: TIndexDef; override;
```

C++

```
virtual __fastcall TIndexDef PSGetDefaultOrder();
```

Description

The provider component calls PSGetDefaultOrder to generate a default index, called DEFAULT_ORDER, for the records that appear in data packets. This index defines the default sort order for records when they appear in the client dataset on the client.

PSGetDefaultOrder creates a TIndexDef object to describe the default index. The provider that calls PSGetDefaultOrder subsequently frees this TIndexDef object.

See Also

CreateDataPacket (see page 18518)

149.1.2.4.50 TCustomSQLDataSet.PSGetIndexDefs Method

Returns the definitions of all specified indexes defined for the dataset.

Pascal

```
function PSGetIndexDefs(IndexTypes: TIndexOptions): TIndexDefs; override;
```

C++

```
virtual __fastcall TIndexDefs PSGetIndexDefs(TIndexOptions IndexTypes);
```

Description

The provider component calls PSGetIndexDefs to locate an index that contains information it needs. For example, it uses PSGetIndexDefs to locate an index that contains the key fields returned by PSGetKeyFields. This information allows the provider to locate the records that need to change when it applies updates. The provider also calls this method to locate indexes that it adds to the metadata of data packets.

IndexTypes indicates the options required of the returned indexes. PSGetIndexDefs returns all indexes whose Options property includes the specified flags.

PSGetIndexDefs returns a TIndexDefs object that contains all the indexes of the dataset that match the IndexTypes parameter.

See Also

Options (see page 7291)

PSGetKeyFields (see page 7581)

PSGetDefaultOrder (see page 7580)

149.1.2.4.51 TCustomSQLDataSet.PSGetKeyFieldsW Method

Pascal

```
function PSGetKeyFieldsW: WideString; override;
```

C++

```
virtual __fastcall BSTR PSGetKeyFieldsW();
```

Description

This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.

149.1.2.4.52 TCustomSQLDataSet.PSGetParams Method

Returns the current parameter values of the dataset.

Pascal

```
function PSGetParams: TParams; override;
```

C++

```
virtual __fastcall TParams PSGetParams();
```

Description

The provider component calls PSGetParams to implement its GetParams method.

PSGetParams returns a TParams object that describes the current values of all parameters of the dataset. If the dataset does not have any parameters, PSGetParams returns nil (Delphi) or NULL (C++).

Note: The PSGetParams method should not create a new instance of TParams for the return value. The provider that calls PSGetParams does not free the returned value.

See Also

GetParams (see page 18467)

149.1.2.4.53 TCustomSQLDataSet.PSGetQuoteCharW Method

Pascal

```
function PSGetQuoteCharW: WideString; override;
```

C++

```
virtual __fastcall BSTR PSGetQuoteCharW();
```

Description

This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.

149.1.2.4.54 TCustomSQLDataSet.PSGetTableNameW Method

Pascal

```
function PSGetTableNameW: WideString; override;
```

C++

```
virtual __fastcall BSTR PSGetTableNameW();
```

Description

This is PSGetTableNameW, a member of class TCustomSQLDataSet.

149.1.2.4.55 TCustomSQLDataSet.PSGetUpdateException Method

Generates an EUpdateError object based on another exception object.

Pascal

```
function PSGetUpdateException(E: Exception; Prev: EUpdateError): EUpdateError; override;
```

C++

```
virtual __fastcall EUpdateError PSGetUpdateException(Exception E, EUpdateError Prev);
```

Description

The provider component calls PSGetUpdateException when the exception E occurs while applying updates. This method allows the provider to create a single EUpdateError object that encapsulates all the exceptions that occur in an update operation.

PSGetUpdateError determines the error message, context string, and error code from the exception specified by E and creates a new EUpdateError exception that includes that information as well as the information about a previous error as specified by Prev. The context string and error code for E may be specific to the Exception class associated with the dataset. By default, TDataSet generates an EUpdateError with an error code of 1 and an empty context string.

See Also

Create (see page 6733)

149.1.2.4.56 TCustomSQLDataSet.PSInTransaction Method

Indicates whether actions against the dataset occur in the context of a transaction.

Pascal

```
function PSInTransaction: Boolean; override;
```

C++

```
virtual __fastcall Boolean PSInTransaction();
```

Description

The provider component applies updates within a transaction, if possible. To do so, it first checks whether a transaction is already underway, and if not, generates one using PSStartTransaction. To determine whether a transaction is already underway, the provider calls PSInTransaction.

PSInTransaction returns true if there is a transaction underway, and false otherwise.

See Also

PSStartTransaction (see page 7586)

PSEndTransaction (see page 7578)

149.1.2.4.57 TCustomSQLDataSet.PSIsSQLBased Method

Indicates whether the dataset includes native SQL support.

Pascal

```
function PSIsSQLBased: Boolean; override;
```

C++

```
virtual __fastcall Boolean PSIsSQLBased();
```

Description

The provider calls PSIsSQLBased to determine whether the dataset supports some form of native SQL. This information is used when generating SQL commands that are executed by PSExecuteStatement.

Even if the dataset is not SQL-based, it may still allow the provider to execute SQL statements for applying updates. For example, BDE-enabled local databases such as Paradox and DBASE provide SQL support via the BDE's local SQL engine, but are not SQL-based. However, table and field names are not specified in the same way for the local SQL engine as they are for native SQL. The provider uses PSIsSQLBased to determine how these names should appear in generated SQL.

Note: The PSIsSQLSupported method indicates whether the dataset can execute SQL commands at all, regardless of whether the dataset is SQL-based.

See Also

PSIsSQLSupported (see page 7584)

PSExecuteStatement (see page 7579)

149.1.2.4.58 TCustomSQLDataSet.PSIsSQLSupported Method

Indicates whether the dataset can execute SQL statements.

Pascal

```
function PSIsSQLSupported: Boolean; override;
```

C++

```
virtual __fastcall Boolean PSIsSQLSupported();
```

Description

PSIsSQLSupported indicates whether the provider can call PSExecuteStatement to execute an SQL statement. When PSIsSQLSupported returns false, calling PSExecuteStatement generates an exception. The provider can still work with a dataset that returns false from PSIsSQLSupported, but it must set the ResolveToDataSet property to true, or provide data on a read-only basis.

See Also

PSIsSQLBased (see page 7584)

PSExecuteStatement (see page 7579)

ResolveToDataSet (see page 18531)

149.1.2.4.59 TCustomSQLDataSet.PSReset Method

Resets the dataset so that it is positioned on the first record.

Pascal

```
procedure PSReset; override;
```

C++

```
virtual __fastcall PSReset();
```

Description

The provider calls PSReset to reposition the dataset on the first record before fetching records.

See Also

GetRecords (see page 18468)

AS_GetRecords (see page 16455)

149.1.2.4.60 TCustomSQLDataSet.PSSetCommandText Method

Assigns a specified SQL command to be executed by PSExecute.

Pascal

```
procedure PSSetCommandText(const ACommandText: WideString); override;
```

C++

```
virtual __fastcall PSSetCommandText(const BSTR ACommandText);
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

The value of the CommandText parameter replaces the dataset's SQL statement (if it has one), or replaces the name of an underlying database table or stored procedure.

See Also

CommandText (see page 8465)

PSExecute (see page 7590)

GetRecords (see page 18468)

Execute (see page 18466)

Options (see page 18440)

149.1.2.4.61 TCustomSQLDataSet.PSSetParams Method

Assigns the specified parameter values to the dataset.

Pascal

```
procedure PSSetParams(AParams: TParams); override;
```

C++

```
virtual __fastcall PSSetParams(TParams AParams);
```

Description

PSSetParams assigns the parameter values specified by AParams to the parameters of the dataset.

See Also

Params (see page 8489)

PSGetParams (see page 7582)

149.1.2.4.62 TCustomSQLDataSet.PSStartTransaction Method

Starts a transaction that encapsulates all activity until a call to PSEndTransaction.

Pascal

```
procedure PSStartTransaction; override;
```

C++

```
virtual __fastcall PSStartTransaction();
```

Description

The provider component applies updates within a transaction, if possible. To do so, it calls PSStartTransaction to start a transaction before applying updates, and calls PSEndTransaction when all updates are applied or too many errors are encountered.

Note: TDataSetProvider only starts a transaction (Using PSStartTransaction) if the PSInTransaction method indicates that there is no transaction already established. That is, it does not assume nested transaction support is available.

See Also

PSEndTransaction (see page 7578)

PSInTransaction (see page 7583)

applyupdates (see page 18459)

149.1.2.4.63 TCustomSQLDataSet.PSUpdateRecord Method

Applies a single update to the underlying database table or tables.

Pascal

```
function PSUpdateRecord(UpdateKind: TUpdateKind; Delta: TDataSet): Boolean; override;
```

C++

```
virtual __fastcall Boolean PSUpdateRecord(TUpdateKind UpdateKind, TDataSet Delta);
```

Description

Before the provider generates SQL statements to apply an update, it calls PSUpdateRecord to let the dataset apply the update in some other fashion. For example, BDE-enabled datasets generate an OnUpdateRecord to allow the application to apply the update and, if the OnUpdateRecord event handler does not apply the update, uses any associated TUpdateSQL object to apply the update.

UpdateKind indicates whether the update is an insertion (ukInsert), deletion (ukDelete), or modification (ukModify).

Delta is the delta datapacket. The current record represents the record to be updated.

PSUpdateRecord returns true if it updates the record, false otherwise. If PSUpdateRecord returns false, the provider generates dynamic SQL to perform the update and calls PSExecuteStatement.

Note: PSUpdateRecord is only called if the provider's ResolveToDataSet property is false.

See Also

OnUpdateRecord (see page 9957)

TUpdateSQL (see page 10229)

applyupdates (see page 18459)

ResolveToDataSet (see page 18531)

PSExecuteStatement (see page 7579)

149.1.2.4.64 TCustomSQLDataSet.SetBufListSize Method

Pascal

```
procedure SetBufListSize(Value: Integer); override;
```

C++

```
virtual __fastcall SetBufListSize(int Value);
```

Description

This is SetBufListSize, a member of class TCustomSQLDataSet.

149.1.2.4.65 TCustomSQLDataSet.SetCommandText Method

Pascal

```
procedure SetCommandText(const Value: WideString); virtual;
```

C++

```
virtual __fastcall SetCommandText(const BSTR Value);
```

Description

This is SetCommandText, a member of class TCustomSQLDataSet.

149.1.2.4.66 TCustomSQLDataSet.SetFCommandText Method

Pascal

```
procedure SetFCommandText(const Value: string);
```

C++

```
__fastcall SetFCommandText(const AnsiString Value);
```

Description

This is SetFCommandText, a member of class TCustomSQLDataSet.

149.1.2.4.67 TCustomSQLDataSet.SetFieldData Method

Pascal

```
procedure SetFieldData(Field: TField; Buffer: Pointer); override;
```

C++

```
virtual __fastcall SetFieldData(TField Field, void * Buffer);
```

Description

This is SetFieldData, a member of class TCustomSQLDataSet.

149.1.2.4.68 TCustomSQLDataSet.SetMaxBlobSize Method

Pascal

```
procedure SetMaxBlobSize(MaxSize: Integer);
```

C++

```
__fastcall SetMaxBlobSize(int MaxSize);
```

Description

This is SetMaxBlobSize, a member of class TCustomSQLDataSet.

149.1.2.4.69 TCustomSQLDataSet.SetParamsFromCursor Method

Pascal

```
procedure SetParamsFromCursor;
```

C++

```
__fastcall SetParamsFromCursor();
```

Description

This is SetParamsFromCursor, a member of class TCustomSQLDataSet.

149.1.2.4.70 TCustomSQLDataSet.SetSchemaInfo Method

Indicates whether the dataset represents metadata from the server, and if so, what kind.

Pascal

```
procedure SetSchemaInfo(SchemaType: TSchemaType; SchemaObjectName: WideString; SchemaPattern:
WideString; PackageName: WideString = '');
```

C++

```
__fastcall SetSchemaInfo(TSchemaType SchemaType, BSTR SchemaObjectName, BSTR SchemaPattern,
BSTR PackageName = '');
```

Description

Call SetSchemaInfo to indicate whether the SQL dataset represents the results of an SQL command (specified by CommandText (see page 20189)) or whether it represents a table of schema information (metadata).

The SchemaType parameter identifies the type of information you want. It can have any of the following values:

Value	Description
stNoSchema	No schema information. The SQL dataset is populated with the results of its query or stored procedure rather than metadata from the server.
stTables	Information about all the data tables on the database server that match the criteria specified by the SQL connection's TableScope property.
stSysTables	Information about all of the system tables on the database server. Not all servers use system tables to store metadata. Requesting a list of system tables from a server that does not use them results in an empty dataset.
stProcedures	Information about all of the stored procedures on the database server.
stColumns	Information about all of the columns (fields) in a specified table.
stProcedureParams	Information about all of the parameters of a specified stored procedure.
stIndexes	Information about all of the indexes defined for a specified table.

SchemaObjectName is the name of the table or stored procedure about whose columns, indexes, or parameters you want information. When SchemaType is stColumns or stIndexes, SchemaObjectName is the name of a table. When SchemaType is stProcedureParams, SchemaObjectName is the name of a stored procedure. When SchemaType is stNoSchema, stTables, stSysTables, or stProcedures, or stPackages, SchemaObjectName is ignored.

SchemaPattern is an SQL pattern mask that filters the resulting dataset. The name of each item (table, stored procedure, field, parameter, or index) must match this pattern mask or it does not appear as a record when the SQL dataset is opened. SchemaPattern uses the wildcards '%' (to match a string of arbitrary characters of any length) and '_' (to match a single arbitrary character). To use a literal percent or underscore in a pattern, the character is doubled (%% or __). If you do not want to use a pattern, set Pattern to nil (Delphi) or NULL (C++).

PackageName is the name of an Oracle package in which to find the stored procedure when SchemaType is stProcedures or stProcedureParams. If the server is not an Oracle server, or if the stored procedure is not defined in a package, PackageName should be an empty string.

After a call to SetSchemaInfo, opening the dataset results in a dataset populated with the specified information. If SchemaType is stNoSchema, the dataset reverts to the command specified by CommandText (see page 20189).

Note: When you change the CommandText (see page 20189) property, the dataset automatically reverts to a SchemaType of stNoSchema.

See Also

Open (see page 6976)

CommandText (see page 20189)

TableScope (see page 20255)

Fetching Metadata into a Unidirectional Dataset

Accessing Schema Information

149.1.2.4.71 TCustomSQLDataSet.SetSortFieldNames Method

Pascal

```
procedure SetSortFieldNames(Value: WideString);
```

C++

```
__fastcall SetSortFieldNames(BSTR Value);
```

Description

This is SetSortFieldNames, a member of class TCustomSQLDataSet.

149.1.2.4.72 TCustomSQLDataSet.UpdateIndexDefs Method

Pascal

```
procedure UpdateIndexDefs; override;
```

C++

```
virtual __fastcall UpdateIndexDefs();
```

Description

This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

149.1.2.5 TCustomSQLDataSet Properties

The properties of the TCustomSQLDataSet class are listed here.

Protected Properties

	Name	Description
	BlobBuffer (see page 20189)	Reserves a buffer for storing BLOB values.
	CommandType (see page 20190)	Indicates the meaning of the CommandText (see page 20189) property.
	CurrentBlobSize (see page 20191)	Indicates the size of the most recently read BLOB field.
R	DataLink (see page 20191)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	DataSource (see page 20192)	Links the SQL dataset to another (master) dataset.
R	InternalConnection (see page 20194)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	LastError (see page 20194)	Saves the error message for the last error returned by dbExpress.
	MaxBlobSize (see page 20194)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	NativeCommand (see page 20195)	Represents the SQL command that is sent to the server.
	ParamCheck (see page 20196)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
R	ParamCount (see page 20197)	Indicates the number of parameters listed by the Params (see page 20197) property.
	Params (see page 20197)	Represents the parameters for a query or stored procedure.
	ProcParams (see page 20199)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
R	RowsAffected (see page 20200)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 20200)	Identifies the metadata with which to populate the dataset.
	SortFieldNames (see page 20201)	Indicates the sort order when CommandType (see page 20190) is ctTable.

Public Properties

	Name	Description
	CommandText (see page 20189)	Specifies the command the dataset executes.
	DesignerData (see page 20192)	Stores custom data.
	IndexDefs (see page 20193)	Contains the definitions of all indexes defined on the server for the dataset.
	Prepared (see page 20198)	Specifies whether the command is prepared before execution.
R	RecordCount (see page 20199)	Indicates the total number of records associated with the dataset.
	SQLConnection (see page 20201)	Specifies the SQL connection component that connects the dataset to a database server.
	TransactionLevel (see page 20202)	Indicates the transaction to which this dataset belongs.

Published Properties

	Name	Description
	Active (see page 20187)	Specifies whether or not a dataset is open.
	GetMetadata (see page 20193)	Specifies whether the SQL dataset fetches metadata information along with the data.
	NumericMapping (see page 20195)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 20196)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	SchemaName (see page 20201)	Specifies the name of the database schema.

149.1.2.5.1 TCustomSQLDataSet.Active Property

Specifies whether or not a dataset is open.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Use Active to determine or set whether a dataset is populated with data. When Active is false, the dataset is closed; the dataset cannot read or write data and data-aware controls can not use it to fetch data or post edits. When Active is true, the dataset can be populated with data. It can read data from a database or other source (such as a provider). Depending on the CanModify property, active datasets can post changes.

Setting Active to true:

1. Generates a BeforeOpen event.
2. Sets the dataset state to dsBrowse.
3. Establishes a way to fetch data (typically by opening a cursor).
4. Generates an AfterOpen event.

If an error occurs while opening the dataset, dataset state is set to dsInactive, and any cursor is closed.

Setting Active to false:

1. Triggers a BeforeClose event.
2. Sets the State property to dsInactive.
3. Closes the cursor.
4. Triggers an AfterClose event.

An application must set Active to false before changing other properties that affect the status of a database or the controls that display data in an application.

Note: Calling the Open method sets Active to true; calling the Close method sets Active to false.

Delphi Examples:

```
{
This example requires a database with the following name or
in the form of a *.DB file in the following path.  The table
should be closed when changing the database.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Customers.Active := False;
  try
  { First try to use an alias }
    Customers.DatabaseName := 'DBDEMOS';
    Customers.Active := True;
  except
    on EDatabaseError do
    begin
      { If that fails, try to use the drive and directory }
      Customers.TableName := 'Flights';
      Customers.DatabaseName :=
        'c:\Documents and Settings\All Users\Documents\RAD
Studio\5.0\Demos\IntraWeb\Win32\FlightInformation\Data';
      Customers.Active := True;
    end;
  end;
end;
```

See Also

DataSource (see page 7014)

State (see page 7036)

AfterOpen (see page 7045)

BeforeOpen (see page 7051)

Close (see page 6920)

Open (see page 6976)

CanModify (see page 7012)

BeforeClose (see page 7048)

AfterClose (see page 7040)

Using Datasets with Decision Support Components

Making the Table a Detail of Another Dataset

Opening and Closing Datasets

Fetching the Data

149.1.2.5.2 TCustomSQLDataSet.BlobBuffer Property

Reserves a buffer for storing BLOB values.

Pascal

```
property BlobBuffer: TBlobByteData;
```

C++

```
__property TBlobByteData BlobBuffer;
```

Description

SQL datasets use the BlobBuffer property to allocate a special buffer for holding BLOB data. Applications do not access this protected property.

See Also

MaxBlobSize (see page 20194)

GetBlobFieldData (see page 6947)

149.1.2.5.3 TCustomSQLDataSet.CommandText Property

Specifies the command the dataset executes.

Pascal

```
property CommandText: WideString;
```

C++

```
__property BSTR CommandText;
```

Description

The value of CommandText depends on the CommandType (see page 20190) property:

When CommandType (see page 20190) is ctQuery, CommandText is an SQL statement that the dataset executes. If the statement is a SELECT statement, the SQL dataset executes the statement when it is opened. If the statement does not return a result set, the SQL dataset executes the statement when the ExecSQL (see page 20165) method is called.

When CommandType (see page 20190) is ctStoredProc, CommandText is the name of a stored procedure. The SQL dataset

executes the stored procedure when it is opened or when the ExecSQL (see page 20165) method is called, depending on whether the stored procedure returns a set of records.

When CommandType (see page 20190) is ctTable, CommandText is the name of a table on the database server. The SQL dataset automatically generates a SELECT statement to fetch all the records on all the fields in this table, sorting by the fields listed in the SortFieldNames (see page 20201) property. The generated statement is executed when the dataset is opened.

If CommandText is an SQL statement that includes parameters or the name of a stored procedure that has parameters, use the Params (see page 20197) property to supply parameter values.

See Also

CommandType (see page 20190)

Params (see page 20197)

Open (see page 6976)

ExecSQL (see page 20278)

SortFieldNames (see page 20201)

Fetching Metadata into a Unidirectional Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Accessing Schema Information

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

149.1.2.5.4 TCustomSQLDataSet.CommandType Property

Indicates the meaning of the CommandText (see page 20189) property.

Pascal

```
property CommandType: TSQLCommandType;
```

C++

```
__property TSQLCommandType CommandType;
```

Description

CommandType indicates the type of command that is contained in the CommandText (see page 20189) property. TCustomSQLDataSet (see page 20150) descendants set this value to the way in which they specify a query or stored procedure.

The following table lists the possible values:

CommandType	Corresponding CommandText (see page 20189)
ctQuery	An SQL statement that the dataset executes.
ctStoredProc	The name of a stored procedure.

ctTable	The name of a table on the database server. The SQL dataset automatically generates a SELECT statement to fetch all the records on all the fields in this table.

See Also

CommandText (see page 20189)

Setting Up a Simple Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Specifying What Data to Display

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

149.1.2.5.5 TCustomSQLDataSet.CurrentBlobSize Property

Indicates the size of the most recently read BLOB field.

Pascal

```
property CurrentBlobSize: Int64;
```

C++

```
__property Int64 CurrentBlobSize;
```

Description

Applications can't use this protected property. TCustomSQLDataSet (see page 20150) uses CurrentBlobSize internally to determine the size needed for BlobBuffer (see page 20189). When the dataset reads a BLOB field, it sets CurrentBlobSize to the number of bytes in the BLOB field, as reported by the database driver.

See Also

MaxBlobSize (see page 20194)

BlobBuffer (see page 20189)

149.1.2.5.6 TCustomSQLDataSet.DataLink Property

Identifies the datalink that manages communication between the SQL dataset and another, master dataset.

Pascal

```
property DataLink: TDataLink;
```

C++

```
__property TDataLink DataLink;
```

Description

The protected DataLink property is used internally when the SQL dataset acts as the detail dataset in a master/detail relationship that is not based on nested datasets. The datalink object manages communication between another (master) dataset and this

SQL dataset. For example, it is the datalink that tells this SQL dataset to refresh its data when the active record of the master dataset changes.

Do not use DataLink to establish the relationship between an SQL dataset and another (master) dataset from which it obtains parameter values. Instead, use the DataSource (see page 20192) property, or, if the dataset is TSQLTable (see page 20337), the MasterSource property.

See Also

DataSource (see page 20192)

TSQLDataLink (see page 20260)

MasterSource (see page 20357)

149.1.2.5.7 TCustomSQLDataSet.DataSource Property

Links the SQL dataset to another (master) dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource links this SQL dataset to another (master) dataset that supplies values for this dataset's parameters.

When the SQL dataset's CommandText (see page 20189) (or SQL) property specifies a parameterized query but the application does not provide values for the corresponding parameters in the Params (see page 20197) property, the SQL dataset fetches parameter values from the dataset identified with the specified data source. When the current record in the dataset associated with DataSource changes, the SQL dataset refreshes its data to reflect the new parameter values.

See Also

CommandText (see page 20189)

SQL (see page 20316)

Params (see page 20197)

MasterSource (see page 20357)

149.1.2.5.8 TCustomSQLDataSet.DesignerData Property

Stores custom data.

Pascal

```
property DesignerData: string;
```

C++

```
__property AnsiString DesignerData;
```

Description

DesignerData has no predefined meaning. Application developers can use the DesignerData property to associate application-specific information with an SQL dataset instance.

See Also

Tag (see page 2808)

149.1.2.5.9 TCustomSQLDataSet.GetMetadata Property

Specifies whether the SQL dataset fetches metadata information along with the data.

Pascal

```
property GetMetadata: Boolean;
```

C++

```
__property Boolean GetMetadata;
```

Description

Use GetMetadata to turn on or off the fetching of metadata on a database object. Setting GetMetadata to False will improve performance because no additional SQL will execute in order to get index information. However, setting GetMetadata to False might slow any attempts to update data using a client dataset that is linked via a dataset provider because the resolver might not generate optimized SQL to perform the update.

GetMetadata must be True to perform any operations that require an index. These operations include setting up master/detail relationships, updating data from a client dataset when the provider's UpdateMode is not upWhereAll, and delayed fetching of BLOB data.

Set GetMetadata to False if you are fetching a dataset for read-only purposes.

See Also

TClientDataset

TDataSetProvider (see page 18508)

Fetching the Data using TSQLDataSet

149.1.2.5.10 TCustomSQLDataSet.IndexDefs Property

Contains the definitions of all indexes defined on the server for the dataset.

Pascal

```
property IndexDefs: TIndexDefs;
```

C++

```
__property TIndexDefs IndexDefs;
```

Description

Read IndexDefs to obtain information about what indexes are available on the database server for this dataset.

TCustomSQLDataSet (see page 20150) does not explicitly use the information in IndexDefs. The order of its records is determined by the ORDER BY clause of its SQL statement.

If the SQL dataset is an instance of TSQLTable (see page 20337), you can use IndexDefs to identify the index to use for the IndexName property. Even with other TCustomSQLDataSet (see page 20150) descendants, IndexDefs can be useful for composing SQL commands dynamically that need to use an index on the server. For example, provider components use the index information to uniquely identify records when composing update commands.

See Also

TIndexDef (see page 7283)

SortFieldNames (see page 20201)

CommandText (see page 20189)

SQL (see page 20316)

TDataSetProvider (see page 18508)

IndexName (see page 20356)

GetKeyFieldNames (see page 20169)

149.1.2.5.11 TCustomSQLDataSet.InternalConnection Property

Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.

Pascal

```
property InternalConnection: TSqlConnection;
```

C++

```
__property TSqlConnection InternalConnection;
```

Description

Applications can't use this protected property. Instead, SQL datasets use InternalConnection internally when they need to access the dbExpress interfaces for the connection to the database server. Unlike the published SQLConnection (see page 20201) property, InternalConnection handles the case when the dataset must use a clone of the primary connection because the server limits the number of active statements that the connection supports.

It is not necessary to use InternalConnection except when using the connection component to reference its dbExpress interfaces.

See Also

CloneConnection (see page 20225)

AutoClone (see page 20243)

SQLConnection (see page 20201)

149.1.2.5.12 TCustomSQLDataSet.LastError Property

Saves the error message for the last error returned by dbExpress.

Pascal

```
property LastError: TDBXError;
```

C++

```
__property TDBXError LastError;
```

Description

When the SQL dataset encounters an error in a call to dbExpress, it caches the error message. This allows it to continue until it is in an appropriate state to raise the associated exception. The message is used to raise an exception with the error message when it is safe to do so.

149.1.2.5.13 TCustomSQLDataSet.MaxBlobSize Property

Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

MaxBlobSize limits the amount of data that can be fetched from Binary Large Object (BLOB) fields. BLOB fields can hold an indefinite amount of data. However, SQL datasets read that data into a memory buffer. By limiting the amount of data that can be fetched from a BLOB field, you limit the size of the buffer.

If MaxBlobSize is –1, the dataset imposes no maximum BLOB size. If MaxBlobSize is 0, the dataset uses the parameters of the associated SQL connection, if any, to determine the maximum BLOB size.

Note: MaxBlobSize should agree with any parameters on the associated SQL connection component that limit the size of BLOB data.

See Also

GetBlobFieldData (see page 20166)

Params (see page 20253)

SQLConnection (see page 20201)

149.1.2.5.14 TCustomSQLDataSet.NativeCommand Property

Represents the SQL command that is sent to the server.

Pascal

```
property NativeCommand: WideString;
```

C++

```
__property BSTR NativeCommand;
```

Description

SQL datasets sometimes need to alter a query before it can be sent to the dbExpress driver. For example, dbExpress does not support named parameters, so the SQL dataset converts named parameters to unnamed parameters before sending off a command.

NativeCommand is the adjusted command that is sent to the dbExpress driver. It is based on the value of CommandText (see page 20189).

See Also

CommandText (see page 20189)

149.1.2.5.15 TCustomSQLDataSet.NumericMapping Property

Maps fldBCD to either ftFMTBCD or ftBCD field types.

Pascal

```
property NumericMapping: Boolean;
```

C++

```
__property Boolean NumericMapping;
```

Description

Use NumericMapping to control how to map fldBCD (dbExpress Core logical type) to eitheir ftFMTBCD or ftBCD field types. The default is False. If True, all dbExpress fldBCD types will map to ftFMTBCD across all dbExpress drivers for field type

compatibility.

See Also

TFieldType (see page 7612)

149.1.2.5.16 TCustomSQLDataSet.ObjectView Property

Specifies whether fields are stored hierarchically or flattened out in the Fields property.

Pascal

```
property ObjectView: Boolean;
```

C++

```
__property Boolean ObjectView;
```

Description

ObjectView affects the way the Fields property stores object fields and the way data-aware grids display ADT and array fields.

When ObjectView is true, the fields are stored hierarchically in the Fields property, meaning any child fields of an object field are referenced by the object field and don't appear sequentially after the object field in the TFields:: .Fields array. When ObjectView is false, the fields are stored sequentially in the Fields property, meaning any child fields of an object field are stored after the object field as siblings in the Fields array.

When ObjectView is false, object field types, such as TADTField, are not created. This switch is provided for increased compatibility with older data-aware controls, which may not handle object field types properly. The default is false for TBDEDataSet, and true for TClientDataSet.

Note: If the ObjectView property of the dataset provided to a TClientDataSet is false, then the TClientDataSet's ObjectView property always acts as false, since the data received is already flattened.

See Also

FieldDefList (see page 7019)

Fields (see page 7023)

Displaying ADT and Array Fields

Using Nested Detail Tables

Using Object Fields

Working with ADT Fields

Working with Array Fields

149.1.2.5.17 TCustomSQLDataSet.ParamCheck Property

Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

By default, the Params (see page 20197) property is updated automatically to reflect any parameters that appear in the query

specified by the CommandText (see page 20189) property (or the SQL property for TSQLQuery (see page 20298)) at runtime. That is, whenever the CommandText (see page 20189) property changes, the Params (see page 20197) property is regenerated to reflect the new set of parameters.

There are times when you want to override this default behavior. For example, when specifying a data definition language (DDL) statement that defines a stored procedure, the resulting SQL command may include parameters. These parameters, however, should not be used as parameters of the SQL dataset, because they are not parameters of the DDL command, but rather, they are parameters of the stored procedure that the command creates.

In order to override the default generation of the Params (see page 20197) property in response to changes in CommandText (see page 20189), set ParamCheck to false.

See Also

Params (see page 20197)

CommandText (see page 20189)

SQL (see page 20316)

149.1.2.5.18 TCustomSQLDataSet.ParamCount Property

Indicates the number of parameters listed by the Params (see page 20197) property.

Pascal

```
property ParamCount: Integer;
```

C++

```
__property int ParamCount;
```

Description

ParamCount is the number of parameters listed in the Params (see page 20197) property. Applications can't use this protected property. It is used internally when preparing SQL statements for execution.

See Also

Params (see page 20197)

Prepared (see page 20198)

149.1.2.5.19 TCustomSQLDataSet.Params Property

Represents the parameters for a query or stored procedure.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Params is a collection of TParam objects that represent the parameters of a query or stored procedure specified by the SQL dataset. When you specify a query by setting the CommandText (see page 20189) property (or the SQL property in TSQLQuery (see page 20298)), the SQL dataset automatically parse the query and fills Params with a TParam object for every parameter in the query. Similarly, when you specify a stored procedure by setting the CommandText (see page 20189) property (or the StoredProcName property in TSQLStoredProc (see page 20318)), the SQL dataset automatically asks the server for a list of stored procedure parameters and fills Params with a TParam object for every parameter of the stored

procedure.

At design-time, use the Params property editor to access the individual TParam objects. For queries, use this property editor to check the DataType property of each parameter and to assign values to the parameters. For stored procedures, you may need to do more, changing the type or number of parameters if the server does not supply complete information about stored procedure parameters.

You can also use Params to assign parameter properties in code at runtime.

Note: If you do not assign values to input parameters before opening the dataset, it fetches values from the dataset identified by the DataSource (see page 20192) property. However, this only works if the individual TParam objects have the same names as fields in the master dataset.

See Also

ParamCheck (see page 20196)

SQL (see page 20316)

StoredProcName (see page 20336)

CommandText (see page 20189)

TParam (see page 7386)

Value (see page 7433)

DataType (see page 7427)

DataSource (see page 20192)

Configuring TSQL Connection

Connecting to Databases with TDatabase

Using TQuery

Using TSimpleDataSet

149.1.2.5.20 TCustomSQLDataSet.Prepared Property

Specifies whether the command is prepared before execution.

Pascal

```
property Prepared: Boolean;
```

C++

```
__property Boolean Prepared;
```

Description

SQL datasets are automatically prepared when you call Open or ExecSQL (see page 20165) (or ExecProc). When a SQL dataset is prepared, dbExpress and the server allocate resources for the SQL statement and its parameters. If CommandType (see page 20190) is ctTable, the dataset generates its SELECT query when it is prepared. Any parameters that are not bound by the server are folded into a query when it is prepared.

If you execute the query or stored procedure more than once, you can improve performance by setting Prepared to true before opening the dataset. Setting Prepared to true prepares the dataset at once, rather than waiting until the dataset is opened.

Conversely, set Prepared to false if you want to ensure that the dataset is re-prepared before it executes (for example, if you change a parameter value or the SortFieldNames (see page 20201) property).

See Also

Params (see page 20197)

ExecSQL (see page 20278)

Open (see page 6976)

Executing the Command

Fetching the Data

149.1.2.5.21 TCustomSQLDataSet.ProcParams Property

ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.

Pascal

```
property ProcParams: TList;
```

C++

```
__property TList ProcParams;
```

Description

When the SQL dataset represents a stored procedure, it fetches information about stored procedure parameters from the server as a list of parameter descriptors. It stores this list as the value of ProcParams.

Applications never use the protected ProcParams directly. Instead, the information from the list is used to initialize the Params (see page 20197) property.

See Also

Params (see page 20197)

GetProcedureParams (see page 20233)

149.1.2.5.22 TCustomSQLDataSet.RecordCount Property

Indicates the total number of records associated with the dataset.

Pascal

```
property RecordCount: Integer;
```

C++

```
__property int RecordCount;
```

Description

Read RecordCount to determine the total number of records in the dataset. An application can read RecordCount even if the dataset is not active.

Reading RecordCount will generate an exception if the dataset can't determine the number of records. Do not read RecordCount if

The dataset represents stored procedure.

The dataset represents a query that contains parameters.

The dataset represents a multi-table join.

See Also

RecNo (see page 7031)

Eof (see page 7015)

149.1.2.5.23 TCustomSQLDataSet.RowsAffected Property

Indicates the number of rows operated on by the latest query execution.

Pascal

```
property RowsAffected: Integer;
```

C++

```
__property int RowsAffected;
```

Description

When the SQL dataset represents a query that does not return a result set, you can't use the RecordCount (see page 20199) property to ascertain the number of records affected by the query. In such cases, read RowsAffected to determine how many rows were updated or deleted by the query when it executed. If no rows were updated or deleted, RowsAffected is zero. RowsAffected is –1 if the SQL statement could not be executed due to an error condition. (This latter situation typically follows the throwing of an exception.)

See Also

ExecSQL (see page 20278)

RecordCount (see page 20199)

ExecProc (see page 20331)

149.1.2.5.24 TCustomSQLDataSet.SchemaInfo Property

Identifies the metadata with which to populate the dataset.

Pascal

```
property SchemaInfo: TSQLSchemaInfo;
```

C++

```
__property TSQLSchemaInfo SchemaInfo;
```

Description

SQL datasets can be used to represent metadata from the database server rather than the results of a query or stored procedure. SchemaInfo controls whether the dataset represents metadata, and if so, what kind.

When SchemaInfo specifies that the dataset represents schema information, the dataset is populated with that information when it is opened. Each record in the resulting dataset represents a single entity (a single table, stored procedure, field, parameter, or index).

Each type of schema information results in a different set of columns (fields), so it is not advisable to use persistent fields when accessing metadata through an SQL dataset.

Applications do not work directly with the protected SchemaInfo property. Instead, they specify the desired schema information by calling the SetSchemaInfo (see page 20185) method.

See Also

SetSchemaInfo (see page 20185)

Open (see page 6976)

149.1.2.5.25 TCustomSQLDataSet.SchemaName Property

Specifies the name of the database schema.

Pascal

```
property SchemaName: string;
```

C++

```
__property AnsiString SchemaName;
```

Description

This name will be available in the appropriate drop down list box after you have connected to a given data source.

149.1.2.5.26 TCustomSQLDataSet.SortFieldNames Property

Indicates the sort order when CommandType (see page 20190) is ctTable.

Pascal

```
property SortFieldNames: WideString;
```

C++

```
__property BSTR SortFieldNames;
```

Description

Use SortFieldNames to specify the fields that should be used to order the records in the dataset. SortFieldNames is only used when CommandType (see page 20190) is ctTable.

SortFieldNames is a semicolon-delimited list of field names. These names are used to generate the ORDER BY clause of the SQL statement the dataset creates to fetch data from a server table. Records are sorted by the first named field, and within that sort order, by the second named field, and so on.

See Also

CommandType (see page 20190)

Representing the Records in a Table

Specifying the Data to Display using TSQLDataSet

149.1.2.5.27 TCustomSQLDataSet.SQLConnection Property

Specifies the SQL connection component that connects the dataset to a database server.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

Use SQLConnection to specify an SQL connection object that connects the dataset to a database server.

At design-time, select from the list of available TSQLConnection (see page 20217) components in the Object Inspector. At runtime, set SQLConnection to reference an existing TSQLConnection (see page 20217) object.

SQLConnection should be the first property you set on the dataset. Only after the SQLConnection property is set can you use the Command Text editor to specify a query for the CommandText (see page 20189) property or populate the dataset with data by setting Active (see page 20187) to true. Similarly, SQLConnection must be set before the dataset can initialize the Params (see page 20197) property when you set CommandText (see page 20189).

See Also

Active (see page 7004)

CommandText (see page 20189)

Params (see page 20197)

Using Query-type Datasets

Using Stored Procedure-type Datasets

Using Table Type Datasets

Connecting Directly to a Database Server

Connecting to the Database Server

149.1.2.5.28 TCustomSQLDataSet.TransactionLevel Property

Indicates the transaction to which this dataset belongs.

Pascal

```
property TransactionLevel: SmallInt;
```

C++

```
__property SmallInt TransactionLevel;
```

Description

Use TransactionLevel to indicate the active transaction to use when opening this dataset or using it to execute commands. TransactionLevel is the same as the transaction identifier that is supplied to the connection's StartTransaction method. If TransactionLevel is unspecified (0), the dataset fetches its records or executes commands in the context of the most recently started transaction.

Note: Most applications do not need to use TransactionLevel because most database servers require that SQL commands belong to the most recently started transaction. However, the InterBase server allows multiple overlapping transactions to be active simultaneously.

See Also

InTransaction (see page 20247)

Commit (see page 20226)

Rollback (see page 20238)

Params (see page 20253)

TransactionsSupported (see page 20256)

StartTransaction

Managing Transactions

149.1.2.6 TCustomSQLDataSet Events

The events of the TCustomSQLDataSet class are listed here.

Published Events

	Name	Description
	AfterClose (see page 20203)	Occurs after a data source connection is closed.
	AfterOpen (see page 20203)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 20203)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 20204)	Occurs after a result set is scrolled.
	BeforeClose (see page 20204)	Occurs just before a data connection is closed.
	BeforeOpen (see page 20204)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 20204)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 20204)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 20204)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 20205)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 20189) property.
	ParseInsertSql (see page 20205)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 20189) property.
	ParseSelectSql (see page 20205)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 20189) property.
	ParseUpdateSql (see page 20205)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 20189) property.

149.1.2.6.1 TCustomSQLDataSet.AfterClose Event

Occurs after a data source connection is closed.

Pascal

```
property AfterClose: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterClose;
```

149.1.2.6.2 TCustomSQLDataSet.AfterOpen Event

Occurs after the connection to a data source is opened.

Pascal

```
property AfterOpen: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterOpen;
```

149.1.2.6.3 TCustomSQLDataSet.AfterRefresh Event

Occurs after a DataSet has been changed and is refreshed.

Pascal

```
property AfterRefresh: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterRefresh;
```

149.1.2.6.4 TCustomSQLDataSet.AfterScroll Event

Occurs after a result set is scrolled.

Pascal

```
property AfterScroll: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent AfterScroll;
```

149.1.2.6.5 TCustomSQLDataSet.BeforeClose Event

Occurs just before a data connection is closed.

Pascal

```
property BeforeClose: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeClose;
```

149.1.2.6.6 TCustomSQLDataSet.BeforeOpen Event

Occurs just before a data connection is opened, but after an open operation has been initiated.

Pascal

```
property BeforeOpen: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeOpen;
```

149.1.2.6.7 TCustomSQLDataSet.BeforeRefresh Event

Occurs before a DataSet refresh occurs.

Pascal

```
property BeforeRefresh: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeRefresh;
```

149.1.2.6.8 TCustomSQLDataSet.BeforeScroll Event

Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.

Pascal

```
property BeforeScroll: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent BeforeScroll;
```

149.1.2.6.9 TCustomSQLDataSet.OnCalcFields Event

Occurs when calculated or derived fields are updated.

Pascal

```
property OnCalcFields: TDataSetNotifyEvent;
```

C++

```
__property TDataSetNotifyEvent OnCalcFields;
```

149.1.2.6.10 TCustomSQLDataSet.ParseDeleteSql Event

Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 20189) property.

Pascal

```
property ParseDeleteSql: TParseSqlEvent;
```

C++

```
__property TParseSqlEvent ParseDeleteSql;
```

149.1.2.6.11 TCustomSQLDataSet.ParseInsertSql Event

Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 20189) property.

Pascal

```
property ParseInsertSql: TParseInsertSqlEvent;
```

C++

```
__property TParseInsertSqlEvent ParseInsertSql;
```

149.1.2.6.12 TCustomSQLDataSet.ParseSelectSql Event

Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 20189) property.

Pascal

```
property ParseSelectSql: TParseSqlEvent;
```

C++

```
__property TParseSqlEvent ParseSelectSql;
```

149.1.2.6.13 TCustomSQLDataSet.ParseUpdateSql Event

Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 20189) property.

Pascal

```
property ParseUpdateSql: TParseSqlEvent;
```

C++

```
__property TParseSqlEvent ParseUpdateSql;
```

149.1.3 TFLDDesc Class

Class Hierarchy

SqlExpr.TFLDDesc

Pascal

```
TFLDDesc = class;
```

C++

```
class TFLDDesc;
```

File

SqlExpr

Description

This is class SqlExpr.TFLDDesc.

149.1.3.1 TFLDDesc Members

The following tables list the members exposed by TFLDDesc.

Public Fields

	Name	Description
	FCalcField (see page 20207)	This is FCalcField, a member of class TFLDDesc.
	FFldNum (see page 20207)	This is FFldNum, a member of class TFLDDesc.
	FFLDRights (see page 20207)	This is FFLDRights, a member of class TFLDDesc.
	FFldType (see page 20208)	This is FFldType, a member of class TFLDDesc.
	FFLDVchk (see page 20208)	This is FFLDVchk, a member of class TFLDDesc.
	FLen (see page 20208)	This is FLen, a member of class TFLDDesc.
	FName (see page 20208)	This is FName, a member of class TFLDDesc.
	FNullOffset (see page 20208)	This is FNullOffset, a member of class TFLDDesc.
	FOffset (see page 20209)	This is FOffset, a member of class TFLDDesc.
	FSubType (see page 20209)	This is FSubType, a member of class TFLDDesc.
	FUnits1 (see page 20209)	This is FUnits1, a member of class TFLDDesc.
	FUnits2 (see page 20209)	This is FUnits2, a member of class TFLDDesc.

Public Properties

	Name	Description
	bCalcField (see page 20210)	This is bCalcField, a member of class TFLDDesc.
	efldrRights (see page 20210)	This is efldrRights, a member of class TFLDDesc.
	efldvVchk (see page 20210)	This is efldvVchk, a member of class TFLDDesc.
	iFldNum (see page 20211)	This is iFldNum, a member of class TFLDDesc.
	iFldType (see page 20211)	This is iFldType, a member of class TFLDDesc.
	iLen (see page 20211)	This is iLen, a member of class TFLDDesc.
	iNullOffset (see page 20211)	This is iNullOffset, a member of class TFLDDesc.
	iOffset (see page 20211)	This is iOffset, a member of class TFLDDesc.
	iSubType (see page 20212)	This is iSubType, a member of class TFLDDesc.
	iUnits1 (see page 20212)	This is iUnits1, a member of class TFLDDesc.
	iUnits2 (see page 20212)	This is iUnits2, a member of class TFLDDesc.
	szName (see page 20212)	This is szName, a member of class TFLDDesc.

149.1.3.2 TFLDDesc Fields

The fields of the TFLDDesc class are listed here.

Public Fields

	Name	Description
	FCalcField (see page 20207)	This is FCalcField, a member of class TFLDDesc.
	FFldNum (see page 20207)	This is FFldNum, a member of class TFLDDesc.
	FFLDRights (see page 20207)	This is FFLDRights, a member of class TFLDDesc.
	FFldType (see page 20208)	This is FFldType, a member of class TFLDDesc.
	FFLDVchk (see page 20208)	This is FFLDVchk, a member of class TFLDDesc.
	FLen (see page 20208)	This is FLen, a member of class TFLDDesc.
	FName (see page 20208)	This is FName, a member of class TFLDDesc.
	FNullOffset (see page 20208)	This is FNullOffset, a member of class TFLDDesc.
	FOffset (see page 20209)	This is FOffset, a member of class TFLDDesc.
	FSubType (see page 20209)	This is FSubType, a member of class TFLDDesc.
	FUnits1 (see page 20209)	This is FUnits1, a member of class TFLDDesc.
	FUnits2 (see page 20209)	This is FUnits2, a member of class TFLDDesc.

149.1.3.2.1 TFLDDesc.FCalcField Field

Pascal

```
FCalcField: WordBool;
```

C++

```
WordBool FCalcField;
```

Description

This is FCalcField, a member of class TFLDDesc.

149.1.3.2.2 TFLDDesc.FFldNum Field

Pascal

```
FFldNum: Word;
```

C++

```
Word FFldNum;
```

Description

This is FFldNum, a member of class TFLDDesc.

149.1.3.2.3 TFLDDesc.FFLDRights Field

Pascal

```
FFLDRights: FLDRights;
```

C++

```
FLDRights FFLDRights;
```

Description

This is FFLDRights, a member of class TFLDDesc.

149.1.3.2.4 TFLDDesc.FFldType Field

Pascal

```
FFldType: Word;
```

C++

```
Word FFldType;
```

Description

This is FFldType, a member of class TFLDDesc.

149.1.3.2.5 TFLDDesc.FFLDVchk Field

Pascal

```
FFLDVchk: FLDVchk;
```

C++

```
FLDVchk FFLDVchk;
```

Description

This is FFLDVchk, a member of class TFLDDesc.

149.1.3.2.6 TFLDDesc.FLen Field

Pascal

```
FLen: LongWord;
```

C++

```
LongWord FLen;
```

Description

This is FLen, a member of class TFLDDesc.

149.1.3.2.7 TFLDDesc.FName Field

Pascal

```
FName: WideString;
```

C++

```
BSTR FName;
```

Description

This is FName, a member of class TFLDDesc.

149.1.3.2.8 TFLDDesc.FNullOffset Field

Pascal

```
FNullOffset: Word;
```

C++

```
Word FNullOffset;
```

Description

This is FNullOffset, a member of class TFLDDesc.

149.1.3.2.9 TFLDDesc.FOffset Field

Pascal

```
FOffset: Word;
```

C++

```
Word FOffset;
```

Description

This is FOffset, a member of class TFLDDesc.

149.1.3.2.10 TFLDDesc.FSubType Field

Pascal

```
FSubType: Word;
```

C++

```
Word FSubType;
```

Description

This is FSubType, a member of class TFLDDesc.

149.1.3.2.11 TFLDDesc.FUnits1 Field

Pascal

```
FUnits1: SmallInt;
```

C++

```
SmallInt FUnits1;
```

Description

This is FUnits1, a member of class TFLDDesc.

149.1.3.2.12 TFLDDesc.FUnits2 Field

Pascal

```
FUnits2: SmallInt;
```

C++

```
SmallInt FUnits2;
```

Description

This is FUnits2, a member of class TFLDDesc.

149.1.3.3 TFLDDesc Properties

The properties of the TFLDDesc class are listed here.

Public Properties

	Name	Description
	bCalcField (see page 20210)	This is bCalcField, a member of class TFLDDesc.
	efldrRights (see page 20210)	This is efldrRights, a member of class TFLDDesc.
	efldvVchk (see page 20210)	This is efldvVchk, a member of class TFLDDesc.
	iFldNum (see page 20211)	This is iFldNum, a member of class TFLDDesc.
	iFldType (see page 20211)	This is iFldType, a member of class TFLDDesc.
	iLen (see page 20211)	This is iLen, a member of class TFLDDesc.
	iNullOffset (see page 20211)	This is iNullOffset, a member of class TFLDDesc.
	iOffset (see page 20211)	This is iOffset, a member of class TFLDDesc.
	iSubType (see page 20212)	This is iSubType, a member of class TFLDDesc.
	iUnits1 (see page 20212)	This is iUnits1, a member of class TFLDDesc.
	iUnits2 (see page 20212)	This is iUnits2, a member of class TFLDDesc.
	szName (see page 20212)	This is szName, a member of class TFLDDesc.

149.1.3.3.1 TFLDDesc.bCalcField Property

Pascal

```
property bCalcField: WordBool;
```

C++

```
__property WordBool bCalcField;
```

Description

This is bCalcField, a member of class TFLDDesc.

149.1.3.3.2 TFLDDesc.efldrRights Property

Pascal

```
property efldrRights: FLDRights;
```

C++

```
__property FLDRights efldrRights;
```

Description

This is efldrRights, a member of class TFLDDesc.

149.1.3.3.3 TFLDDesc.efldvVchk Property

Pascal

```
property efldvVchk: FLDVchk;
```

C++

```
__property FLDVchk efldvVchk;
```

Description

This is efldvVchk, a member of class TFLDDesc.

149.1.3.3.4 TFLDDesc.iFldNum Property

Pascal

```
property iFldNum: Word;
```

C++

```
__property Word iFldNum;
```

Description

This is iFldNum, a member of class TFLDDesc.

149.1.3.3.5 TFLDDesc.iFldType Property

Pascal

```
property iFldType: Word;
```

C++

```
__property Word iFldType;
```

Description

This is iFldType, a member of class TFLDDesc.

149.1.3.3.6 TFLDDesc.iLen Property

Pascal

```
property iLen: LongWord;
```

C++

```
__property LongWord iLen;
```

Description

This is iLen, a member of class TFLDDesc.

149.1.3.3.7 TFLDDesc.iNullOffset Property

Pascal

```
property iNullOffset: Word;
```

C++

```
__property Word iNullOffset;
```

Description

This is iNullOffset, a member of class TFLDDesc.

149.1.3.3.8 TFLDDesc.iOffset Property

Pascal

```
property iOffset: Word;
```

C++

```
__property Word iOffset;
```

Description

This is iOffset, a member of class TFLDDesc.

149.1.3.3.9 TFLDDesc.iSubType Property

Pascal

```
property iSubType: Word;
```

C++

```
__property Word iSubType;
```

Description

This is iSubType, a member of class TFLDDesc.

149.1.3.3.10 TFLDDesc.iUnits1 Property

Pascal

```
property iUnits1: SmallInt;
```

C++

```
__property SmallInt iUnits1;
```

Description

This is iUnits1, a member of class TFLDDesc.

149.1.3.3.11 TFLDDesc.iUnits2 Property

Pascal

```
property iUnits2: SmallInt;
```

C++

```
__property SmallInt iUnits2;
```

Description

This is iUnits2, a member of class TFLDDesc.

149.1.3.3.12 TFLDDesc.szName Property

Pascal

```
property szName: WideString;
```

C++

```
__property BSTR szName;
```

Description

This is szName, a member of class TFLDDesc.

149.1.4 TSQLBlobStream Class

TSQLBlobStream lets applications read from or write to BLOB fields in a client dataset.

Class Hierarchy

Pascal

```
TSQLBlobStream = class(TMemoryStream);
```

C++

```
class TSQLBlobStream : public TMemoryStream;
```

File

SqlExpr

Description

Use TSQLBlobStream to read the value of a Binary Large Object (BLOB) field in a dbExpress dataset (TSQLLDataSet, TSQLTable (see page 20337), TSQLQuery (see page 20298), or TSQLStoredProc (see page 20318)). BLOB fields are TBlobField objects and descendants of TBlobField such as TGraphicField and TMemoField. BLOB fields in client datasets use client BLOB streams to implement many of their data access properties and methods.

TSQLBlobStream allows objects that have no specialized knowledge of how data is stored in a BLOB field to read such data by employing the uniform stream interface.

To use an SQL BLOB stream, create an instance of TSQLBlobStream, use the methods of the stream to read the data, and then free the BLOB stream. If you use the same instance of TSQLBlobStream to access data from more than one record, you must call the ReadBlobData (see page 20216) method to update the stream when the current record changes.

Warning: Although TSQLBlobStream inherits several write methods from TMemoryStream, it can't be used to write data back to the dataset because dbExpress datasets are read-only. Writing to TSQLBlobStream simply alters the in-memory buffer that contains a BLOB field's data.

See Also

TBlobField (see page 6801)

TGraphicField (see page 7257)

TMemoField (see page 7348)

TCustomSQLDataSet (see page 20150)

TSQLDataSet (see page 20265)

TSQLTable (see page 20337)

TSQLQuery (see page 20298)

TSQLStoredProc (see page 20318)

149.1.4.1 TSQLBlobStream Members

The following tables list the members exposed by TSQLBlobStream.

Protected Methods

	Name	Description
	GetSize (see page 2982)	This is GetSize, a member of class TStream.
	Realloc (see page 2910)	This is Realloc, a member of class TMemoryStream.
	SetPointer (see page 2823)	Replaces the memory buffer associated with the memory stream.
	SetSize (see page 2989)	This is the overview for the SetSize method overload.

Protected Properties

	Name	Description
	Capacity (see page 2912)	This is Capacity, a member of class TMemoryStream.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2909)	Frees the memory buffer, discarding all data associated with the memory stream.
	CopyFrom (see page 2979)	Copies a specified number of bytes from one stream to another.
	Create (see page 20215)	Creates and initializes an instance of TSQLBlobStream (see page 20213).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 20216)	Frees an instance of TSQLBlobStream (see page 20213).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FixupResourceHeader (see page 2981)	Patches the resource header for a resource that has been written to the stream.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LoadFromFile (see page 2909)	Loads the entire contents of a file into the memory buffer.
	LoadFromStream (see page 2909)	Loads the entire contents of a stream into the memory buffer.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Read (see page 2817)	Reads up to Count bytes from the memory stream into Buffer and advances the current position of the stream by the number of bytes read.
	Read (see page 2982)	Introduces an abstract or pure virtual method responsible for reading from the stream.
	ReadBlobData (see page 20216)	Fills the BLOB stream's memory buffer with the contents of the associated BLOB field.
	ReadBuffer (see page 2983)	Reads bytes from the stream into Buffer.
	ReadComponent (see page 2983)	Initiates streaming of components and their properties.

	ReadComponentRes (see page 2987)	Reads components and their properties from a stream in a resource file format.
	ReadResHeader (see page 2987)	Reads a resource-file header from the stream.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 2818)	Writes the entire contents of the memory stream to the file with a given file name.
	SaveToStream (see page 2818)	Writes the entire contents of the memory stream to the stream object specified by Stream.
V	Seek (see page 2819)	Moves the current position of the stream by Offset bytes, relative to the origin specified by Origin.
V	Seek (see page 2988)	This is the overview for the Seek method overload.
V	SetSize (see page 2910)	Sets the Size (see page 2997) property of the memory stream.
V	Write (see page 2911)	Writes Count bytes from Buffer to the current position in the memory buffer and updates the current position by Count bytes.
A	Write (see page 2989)	Introduces a pure virtual method for writing to the stream.
	WriteBuffer (see page 2990)	Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.
	WriteComponent (see page 2990)	Initiates the writing of components and their properties to a stream.
	WriteComponentRes (see page 2994)	Writes components and their properties to a stream from a resource file format.
	WriteDescendent (see page 2994)	Streams components and their properties in inherited forms.
	WriteDescendentRes (see page 2995)	Streams components and their properties in inherited forms.
	WriteResourceHeader (see page 2995)	Writes a resource-file header to the stream.

Public Properties

	Name	Description
R	Memory (see page 2823)	Provides direct access to the memory pool allocated for the memory stream.
	Position (see page 2996)	Indicates the current offset into the stream for reading and writing.
	Size (see page 2997)	Indicates the size in bytes of the stream.

149.1.4.2 TSQLBlobStream.Create Constructor

Creates and initializes an instance of TSQLBlobStream (see page 20213).

Pascal

```
constructor Create(Field: TBlobField; Mode: TBlobStreamMode = bmRead);
```

C++

```
__fastcall TSQLBlobStream(TBlobField Field, TBlobStreamMode Mode = bmRead);
```

Description

Call Create to obtain an instance of TSQLBlobStream (see page 20213) for reading from a specific TBlobField object in a TCustomSQLDataSet (see page 20150) descendant.

Field specifies the BLOB field from which to read data.

The Mode parameter is provided for compatibility with other BLOB stream constructors. However, in TSQLBlobStream (see page 20213), the Mode parameter is ignored because dbExpress datasets are read-only. SQL Blob streams can only be used to read BLOB data.

Create links the TSQLBlobStream (see page 20213) to the field object specified by the Field parameter. It then calls ReadBlobData (see page 20216) to read the information from the BLOB field into a memory buffer, where it is available for reading.

Warning: If you write to the SQLBlobStream using inherited methods, you are overwriting the in-memory buffer that is filled by the constructor. This will cause subsequent read operations to provide inaccurate information, because the BLOB stream no longer accurately reflects the contents of the BLOB field.

Instead of calling the TSQLBlobStream (see page 20213) constructor, you can also use the dataset's CreateBlobStream method.

See Also

CreateBlobStream (see page 20164)

ReadBlobData (see page 20216)

149.1.4.3 TSQLBlobStream.Destroy Destructor

Frees an instance of TSQLBlobStream (see page 20213).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSQLBlobStream();
```

Description

Do not call Destroy directly in an application. Instead call Free. Free verifies that the TBlobStream object is not nil and only then calls Destroy.

Destroy the TSQLBlobStream (see page 20213) object by calling Free when it is no longer needed for reading from the BLOB field. BLOB streams should be destroyed when the dataset changes the current record.

See Also

Free (see page 22300)

OnDataChange (see page 9497)

149.1.4.4 TSQLBlobStream Methods

The methods of the TSQLBlobStream class are listed here.

Public Methods

	Name	Description
	ReadBlobData (see page 20216)	Fills the BLOB stream's memory buffer with the contents of the associated BLOB field.

149.1.4.4.1 TSQLBlobStream.ReadBlobData Method

Fills the BLOB stream's memory buffer with the contents of the associated BLOB field.

Pascal

```
procedure ReadBlobData;
```

C++

```
__fastcall ReadBlobData();
```

Description

Call ReadBlobData to refresh the contents of the BLOB stream when the current record in the dataset changes. ReadBlobData reads the contents of a BLOB field into the SQL BLOB stream's in-memory buffer and sets the Size property to the number of bytes of BLOB data. It then sets Position to 0 so that the stream is ready for you to read the BLOB data.

Note: It is not necessary to call ReadBlobData when you first create a SQL BLOB stream, because the constructor already performs this task. You need only call ReadBlobData when the current record changes, so that the stream reflects the BLOB

data for the new record.

See Also

Size (see page 2997)

Position (see page 2996)

Create (see page 20215)

149.1.5 TSQLConnection Class

TSQLConnection encapsulates a dbExpress connection to a database server.

Class Hierarchy

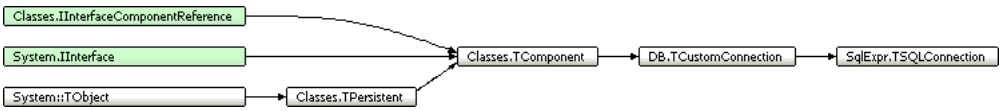

Pascal

```
TSQLConnection = class(TCustomConnection);
```

C++

```
class TSQLConnection : public TCustomConnection;
```

File

SqlExpr

Description

Use TSQLConnection to represent a dbExpress database connection. The connection provided by a single TSQLConnection component can be shared by multiple SQL dataset components through their SQLConnection (see page 20255) properties.

TSQLConnection interacts with a dbExpress driver and two files. The first, dbxdrivers.ini on Windows or dbxdrivers on Linux, lists the installed driver types (Interbase, Oracle, and so on) and, for each driver, lists the libraries (DLLs or shared objects) it requires as well as the default settings for all connection parameters. The second file, dbxconnections.ini on Windows or dbxconnections on Linux, lists named sets of connection configurations. Each configuration represents a set of settings for TSQLConnection and describes a particular database connection.

You can use TSQLConnection to choose one of the named configurations in dbxconnections.ini and use it to connect to a database or to define new configurations and add them to dbxconnections.ini. To perform either of these tasks, double-click on a TSQLConnection component and use the Connection (see page 20244) Builder.

In addition to describing and establishing database connections, TSQLConnection surfaces properties, methods, and events for controlling server login, managing transactions, and fetching metadata from the server.

See Also

TCustomSQLDataSet (see page 20150)

TSQLDataSet (see page 20265)

TSQLTable (see page 20337)

TSQLQuery (see page 20298)

TSQLStoredProc (see page 20318)

Connecting to Databases: Overview

Connecting Directly to a Database Server

Accessing Schema Information

Connecting to the Database Server

Setting Up TSQLConnection

149.1.5.1 TSQLConnection Members

The following tables list the members exposed by TSQLConnection.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckConnection (see page 20225)	This is CheckConnection, a member of class TSQLConnection.
V	CheckDisconnect (see page 20225)	This is CheckDisconnect, a member of class TSQLConnection.
V	ConnectionOptions (see page 20227)	This is ConnectionOptions, a member of class TSQLConnection.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoConnect (see page 20227)	Provides the interface for a method that opens a connection.
V	DoDisconnect (see page 20227)	Provides the interface for a method that terminates the connection.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetConnected (see page 20229)	Returns the value of the Connected property.
	GetDataSet (see page 20229)	Returns a dataset, given its index in the DataSets property array.
V	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 20235)	Initializes the connection component after all objects in the form or data module have been loaded.
V	Notification (see page 20237)	Forwards notification messages to all owned components.
	OpenSchema (see page 20237)	This is the overview for the OpenSchema method overload.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.

V	RegisterClient (see page 20238)	Registers an object that uses the connection component.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetConnected (see page 6867)	Changes the value of the Connected (see page 6868) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SQLError (see page 20240)	This is SQLError, a member of class TSQLConnection.
V	UnRegisterClient (see page 20241)	Removes a registered client so that it is no longer associated with the connection component.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Connection (see page 20244)	This is Connection, a member of class TSQLConnection.
R	ConnectionRegistryFile (see page 20245)	This is ConnectionRegistryFile, a member of class TSQLConnection.
R	DriverRegistryFile (see page 20247)	This is DriverRegistryFile, a member of class TSQLConnection.
	LastError (see page 20248)	This is LastError, a member of class TSQLConnection.
R	QuoteChar (see page 20254)	This is QuoteChar, a member of class TSQLConnection.
	SQLDllHandle (see page 20255)	This is SQLDllHandle, a member of class TSQLConnection.
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.

Public Events

	Name	Description
R	TraceCallbackEvent (see page 20260)	Provides access to the callback function that executes for every SQL command passed to or from the server.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginTransaction (see page 20224)	Initiates a new transaction in the associated database.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CloneConnection (see page 20225)	Returns a copy of the SQL connection object.
	Close (see page 6863)	Closes the connection.
	CloseDataSets (see page 20225)	Closes all datasets associated with the SQL connection object component without disconnecting from the database.
	Commit (see page 20226)	Commits an open transaction.
	CommitAndFree (see page 20226)	This is CommitAndFree, a member of class TSQLConnection.
V	Create (see page 20222)	Creates and initializes an instance TSQLConnection (see page 20217).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 20222)	Destroys the SQL connection object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 20227)	Executes an SQL command on the server.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteDirect (see page 20228)	Executes an SQL command that does not include parameters.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetDefaultSchemaName (see page 20229)	Returns the default schema name of the current object.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldNames (see page 20229)	This is the overview for the GetFieldNames method overload.
	GetIndexNames (see page 20230)	This is the overview for the GetIndexNames method overload.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetLoginUsername (see page 20231)	Returns the username that is logged in on the current connection.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetPackageNames (see page 20231)	Populates a string list with the names of all packages defined on the server.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProcedureNames (see page 20232)	This is the overview for the GetProcedureNames method overload.
	GetProcedureParams (see page 20233)	This is the overview for the GetProcedureParams method overload.
	GetSchemaNames (see page 20234)	Returns the schema names for all objects in the given data source.
	GetTableNames (see page 20234)	This is the overview for the GetTableNames method overload.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LoadParamsFromIniFile (see page 20236)	Sets the DriverName (see page 20246) and Params (see page 20253) properties to the values associated with the ConnectionName (see page 20244) property.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 6866)	Opens the connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Rollback (see page 20238)	Cancels all updates, insertions, and deletions for a specified transaction and ends the transaction.
	RollbackAndFree (see page 20239)	This is RollbackAndFree, a member of class TSQLConnection.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTraceEvent (see page 20239)	Associates the SQL connection with a callback function that executes for every SQL command passed to or from the server.
	StartTransaction (see page 20240)	Initiates a new transaction in the associated database.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActiveStatements (see page 20242)	Indicates the number of statements currently active with the database server.
	AutoClone (see page 20243)	Specifies whether the SQL connection automatically clones database connections when they are needed.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	ConnectionState (see page 20245)	Indicates the current state of the TSQLConnection (see page 20217) object.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 20245)	Lists all active datasets for the connection component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	InTransaction (see page 20247)	Indicates whether a transaction is in progress.
	LocaleCode (see page 20250)	Identifies the locale that is used for sorting items from datasets that use this connection.
R	MaxStmtsPerConn (see page 20251)	Indicates the server limit on the number of statements supported by a single database connection.
R	MetaData (see page 20252)	Provides an interface to the dbExpress object that supplies server metadata.
R	MultipleTransactionsSupported (see page 20253)	Indicates whether the database server supports multiple transactions.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParamsLoaded (see page 20254)	Indicates when parameter values have been loaded from dbxconnections.ini.
	SQLConnection (see page 20255)	This is SQLConnection, a member of class TSQLConnection.
	SQLHourGlass (see page 20255)	Specifies whether the cursor should change to crSQLWait during lengthy operations.
R	TransactionsSupported (see page 20256)	Indicates whether the database server supports transactions.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterConnect (see page 20258)	Occurs after a connection is established.

	AfterDisconnect (see page 20258)	Occurs after the connection closes.
	BeforeConnect (see page 20258)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 20259)	Occurs immediately before the connection closes.
	OnLogin (see page 20259)	Occurs after the BeforeConnect (see page 20258) event and before a connection is established.

Published Properties

	Name	Description
	Connected (see page 20243)	Specifies whether the connection is active.
	ConnectionName (see page 20244)	Names the connection configuration.
	DriverName (see page 20246)	Specifies the database driver associated with the SQL connection.
	GetDriverFunc (see page 20247)	Specifies the function that returns the top-level dbExpress driver.
	KeepConnection (see page 20248)	Specifies whether the connection stays active when no datasets are open.
	LibraryName (see page 20248)	Specifies the dbExpress driver that TSQLConnection (see page 20217) uses to establish a connection.
	LoadParamsOnConnect (see page 20249)	Specifies whether the TSQLConnection (see page 20217) component loads the configuration named by ConnectionName (see page 20244) immediately before connecting to the server.
	LoginPrompt (see page 20250)	Specifies whether a login dialog appears immediately before opening a new connection.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Params (see page 20253)	Lists the connection parameters.
	TableScope (see page 20255)	Indicates what types of tables are returned when fetching schema information about tables.
	Tag (see page 2808)	Stores an integer value as part of a component.
	VendorLib (see page 20257)	Specifies the client software library (DLL or shared object) supplied by the database vendor.

149.1.5.2 TSQLConnection.Create Constructor

Creates and initializes an instance TSQLConnection (see page 20217).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSQLConnection(TComponent * AOwner);
```

Description

Use Create to instantiate TSQLConnection (see page 20217) at runtime. SQL connections added to forms or data modules at design time are created automatically.

The AOwner parameter is the component, typically a form or data module, that is responsible for freeing the SQL connection.

See Also

Destroy (see page 20222)

Owner (see page 2807)

149.1.5.3 TSQLConnection.Destroy Destructor

Destroys the SQL connection object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSQLConnection();
```

Description

To free a TSQLConnection (see page 20217) instance, do not call Destroy. Instead, call Free. Free verifies that the ADO connection is not nil, and only then calls Destroy.

There is no need to explicitly free instances of TSQLConnection (see page 20217) that were created with an Owner.

See Also

Owner (see page 2807)

Create (see page 20222)

delete

Free (see page 22300)

149.1.5.4 TSQLConnection Methods

The methods of the TSQLConnection class are listed here.

Protected Methods

	Name	Description
	CheckConnection (see page 20225)	This is CheckConnection, a member of class TSQLConnection.
V	CheckDisconnect (see page 20225)	This is CheckDisconnect, a member of class TSQLConnection.
V	ConnectionOptions (see page 20227)	This is ConnectionOptions, a member of class TSQLConnection.
V	DoConnect (see page 20227)	Provides the interface for a method that opens a connection.
V	DoDisconnect (see page 20227)	Provides the interface for a method that terminates the connection.
V	GetConnected (see page 20229)	Returns the value of the Connected property.
	GetDataSet (see page 20229)	Returns a dataset, given its index in the DataSets property array.
V	Loaded (see page 20235)	Initializes the connection component after all objects in the form or data module have been loaded.
V	Notification (see page 20237)	Forwards notification messages to all owned components.
	OpenSchema (see page 20237)	This is the overview for the OpenSchema method overload.
V	RegisterClient (see page 20238)	Registers an object that uses the connection component.
	SQLError (see page 20240)	This is SQLError, a member of class TSQLConnection.
V	UnRegisterClient (see page 20241)	Removes a registered client so that it is no longer associated with the connection component.

Public Methods

	Name	Description
	BeginTransaction (see page 20224)	Initiates a new transaction in the associated database.
	CloneConnection (see page 20225)	Returns a copy of the SQL connection object.
	CloseDataSets (see page 20225)	Closes all datasets associated with the SQL connection object component without disconnecting from the database.
	Commit (see page 20226)	Commits an open transaction.
	CommitAndFree (see page 20226)	This is CommitAndFree, a member of class TSQLConnection.
	Execute (see page 20227)	Executes an SQL command on the server.
	ExecuteDirect (see page 20228)	Executes an SQL command that does not include parameters.
	GetDefaultSchemaName (see page 20229)	Returns the default schema name of the current object.
	GetFieldNames (see page 20229)	This is the overview for the GetFieldNames method overload.
	GetIndexNames (see page 20230)	This is the overview for the GetIndexNames method overload.
	GetLoginUsername (see page 20231)	Returns the username that is logged in on the current connection.
	GetPackageNames (see page 20231)	Populates a string list with the names of all packages defined on the server.

	GetProcedureNames (see page 20232)	This is the overview for the GetProcedureNames method overload.
	GetProcedureParams (see page 20233)	This is the overview for the GetProcedureParams method overload.
	GetSchemaNames (see page 20234)	Returns the schema names for all objects in the given data source.
	GetTableNames (see page 20234)	This is the overview for the GetTableNames method overload.
	LoadParamsFromIniFile (see page 20236)	Sets the DriverName (see page 20246) and Params (see page 20253) properties to the values associated with the ConnectionName (see page 20244) property.
	Rollback (see page 20238)	Cancels all updates, insertions, and deletions for a specified transaction and ends the transaction.
	RollbackAndFree (see page 20239)	This is RollbackAndFree, a member of class TSQLConnection.
	SetTraceEvent (see page 20239)	Associates the SQL connection with a callback function that executes for every SQL command passed to or from the server.
	StartTransaction (see page 20240)	Initiates a new transaction in the associated database.

149.1.5.4.1 TSQLConnection.BeginTransaction Method

Initiates a new transaction in the associated database.

Pascal

```
function BeginTransaction: TDBXTransaction;
```

C++

```
__fastcall TDBXTransaction BeginTransaction();
```

Description

Call BeginTransaction to start a new transaction against a database server that supports transactions. (MySQL servers do not support transactions.) You can check the TransactionsSupported (see page 20256) property to determine whether the server supports transactions. If the server does not support transactions, BeginTransaction does nothing.

The method returns a TDBXTransactioninstance.

If the transaction is successfully started, BeginTransaction sets the InTransaction (see page 20247) property to true.

Updates, insertions, and deletions that take place after a call to BeginTransaction are enlisted in that transaction until another overlapping transaction is started or until the application calls CommitFreeAndNil to save the changes or RollbackFreeAndNil to cancel them.

Tip: If the server does not support multiple transactions, check the status of the InTransaction (see page 20247) property to be sure that another transaction is not already in progress.

Tip: When working with overlapped transactions, you can indicate the transaction to which a dataset belongs by setting its TransactionLevel property to the transaction identifier you assign in this method.

See Also

InTransaction (see page 20247)

CommitFreeAndNil

RollbackFreeAndNil

Params (see page 20253)

TransactionsSupported (see page 20256)

TransactionLevel (see page 20202)

149.1.5.4.2 TSQLConnection.CheckConnection Method

Pascal

```
procedure CheckConnection(eFlag: eConnectFlag);
```

C++

```
__fastcall CheckConnection(eConnectFlag eFlag);
```

Description

This is CheckConnection, a member of class TSQLConnection.

149.1.5.4.3 TSQLConnection.CheckDisconnect Method

Pascal

```
procedure CheckDisconnect; virtual;
```

C++

```
virtual __fastcall CheckDisconnect();
```

Description

This is CheckDisconnect, a member of class TSQLConnection.

149.1.5.4.4 TSQLConnection.CloneConnection Method

Returns a copy of the SQL connection object.

Pascal

```
function CloneConnection: TSQLConnection;
```

C++

```
__fastcall TSQLConnection CloneConnection();
```

Description

Call CloneConnection to create a duplicate of this SQLConnection (see page 20255) object. CloneConnection instantiates a new TSQLConnection (see page 20217) object and assigns its properties to match those of this SQLConnection (see page 20255) object.

The returned SQLConnection (see page 20255) is owned by this TSQLConnection (see page 20217). That is, this SQL connection component frees the clone from its own destructor.

See Also

AutoClone (see page 20243)

149.1.5.4.5 TSQLConnection.CloseDataSets Method

Closes all datasets associated with the SQL connection object component without disconnecting from the database.

Pascal

```
procedure CloseDataSets;
```

C++

```
__fastcall CloseDataSets();
```

Description

Call CloseDataSets to close all active datasets without disconnecting from the database server. Ordinarily, when an application calls Close, all datasets are closed, and the connection to the database server is dropped. Calling CloseDataSets instead of Close ensures that an application can close all active datasets without having to reconnect to the database server at a later time.

See Also

Close (see page 6863)

Close (see page 6920)

DataSets (see page 20245)

KeepConnection (see page 20248)

149.1.5.4.6 TSQLConnection.Commit Method

Commits an open transaction.

Pascal

```
procedure Commit(TransDesc: TTransactionDesc); deprecated;
```

C++

```
__fastcall Commit(TTransactionDesc TransDesc);
```

Description

Call Commit to save any changes made during the transaction described by TransDesc and to end that transaction.

TransDesc is the transaction descriptor supplied to the StartTransaction (see page 20240) method when starting the transaction.

If the specified transaction is the only current transaction, Commit sets the InTransaction (see page 20247) property to false after committing the transaction.

Note: This method has been deprecated.

See Also

InTransaction (see page 20247)

StartTransaction (see page 20240)

Rollback (see page 20238)

TransactionsSupported (see page 20256)

TransactionLevel (see page 20202)

149.1.5.4.7 TSQLConnection.CommitAndFree Method

Pascal

```
procedure CommitAndFree(Transaction: TDBXTransaction);
```

C++

```
__fastcall CommitAndFree(TDBXTransaction Transaction);
```

Description

This is CommitAndFree, a member of class TSQLConnection.

149.1.5.4.8 TSQLConnection.ConnectionOptions Method

Pascal

```
procedure ConnectionOptions; virtual;
```

C++

```
virtual __fastcall ConnectionOptions();
```

Description

This is ConnectionOptions, a member of class TSQLConnection.

149.1.5.4.9 TSQLConnection.DoConnect Method

Provides the interface for a method that opens a connection.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

The Connected property uses DoConnect to establish a connection. Descendant classes override the DoConnect method to establish their connection as appropriate. As implemented in TCustomConnection, DoConnect does nothing.

See Also

Connected (see page 6868)

DoDisconnect (see page 6864)

149.1.5.4.10 TSQLConnection.DoDisconnect Method

Provides the interface for a method that terminates the connection.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

The Connected property uses DoDisconnect to close a connection. Descendant classes override the DoDisconnect method to drop a connection. As implemented in TCustomConnection, DoDisconnect does nothing.

See Also

Connected (see page 6868)

DoConnect (see page 6864)

149.1.5.4.11 TSQLConnection.Execute Method

Executes an SQL command on the server.

Pascal

```
function Execute(const SQL: WideString; Params: TParams; ResultSet: TPSResult = nil): Integer;
```

C++

```
__fastcall int Execute(const BSTR SQL, TParams Params, TPSResult ResultSet = nil);
```

Description

Call Execute to execute a single command on the server without the overhead of using an SQL dataset. Execute is especially useful for statements that do not return a result set such as data definition language (DDL) statements.

SQL is the command to execute.

Params (see page 20253) is a TParams object that contains any parameters used by the SQL statement. Parameter binding is by index only (not by name), so the order of parameters is important and the order of the TParam objects in Params (see page 20253) corresponds to the order of the parameters in the SQL statement. Use properties and methods of TParams to create a TParams object, add TParam objects for each parameter, and assign parameter properties such as datatype and value. If the SQL statement does not include any parameters, pass a nil (Delphi) or NULL (C++) value for Params (see page 20253).

ResultSet is a pointer to a variable of type TCustomSQLDataSet (see page 20150)*. If the SQL statement returns a cursor, Execute creates a new TCustomSQLDataSet (see page 20150) instance and populates it with the result set. The new TCustomSQLDataSet (see page 20150) reference is returned as the value to which ResultSet points. If the SQL statement does not return a result set, pass a nil (Delphi) or NULL (C++) value for ResultSet. TPSResult is is set to Pointer for native code and TObject for managed code.

Execute returns the number of records affected by executing the SQL statement.

Tip: If the SQL statement does not include any parameters, you can use the ExecuteDirect (see page 20228) method instead.

See Also

ExecSQL (see page 20278)

ExecuteDirect (see page 20228)

149.1.5.4.12 TSQLConnection.ExecuteDirect Method

Executes an SQL command that does not include parameters.

Pascal

```
function ExecuteDirect(const SQL: WideString): Integer;
```

C++

```
__fastcall int ExecuteDirect(const BSTR SQL);
```

Description

Call ExecuteDirect to execute a single parameterless command on the server without the overhead of using an SQL dataset or preparing the statement before it executes. If the statement ordinarily returns a cursor, ExecuteDirect executes the statement, but does not return a cursor.

SQL is the command to execute.

ExecuteDirect returns 0 if the command was successfully executed. Otherwise, it returns a dbExpress error code.

Note: To execute an SQL statement that includes parameters or that returns a cursor, use the Execute (see page 20227) method instead.

See Also

ExecSQL (see page 20278)

Execute (see page 20227)

149.1.5.4.13 TSQLConnection.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected read implementation of the Connected property. Descendant classes override this method to return true if a connection is open and false otherwise. As defined in TCustomConnection, GetConnected always returns false.

See Also

Connected (see page 6868)

SetConnected (see page 6867)

149.1.5.4.14 TSQLConnection.GetDataSet Method

Returns a dataset, given its index in the DataSets property array.

Pascal

```
function GetDataSet(Index: Integer): TCustomSQLDataSet; reintroduce;
```

C++

```
__fastcall TCustomSQLDataSet GetDataSet(int Index);
```

Description

GetDataSet is the protected read implementation of the DataSets property.

See Also

DataSets (see page 6870)

149.1.5.4.15 TSQLConnection.GetDefaultSchemaName Method

Returns the default schema name of the current object.

Pascal

```
function GetDefaultSchemaName: WideString;
```

C++

```
__fastcall BSTR GetDefaultSchemaName();
```

149.1.5.4.16 GetFieldNames Method

This is the overview for the GetFieldNames method overload.

Overload List

	Name	Description
	TSQLConnection.GetFieldNames (WideString, TWideStrings) (see page 20230)	Populates a string list with the names of fields in a table.
	TSQLConnection.GetFieldNames (WideString, WideString, TWideStrings) (see page 20230)	Populates a string list with the names of fields in a table.

149.1.5.4.16.1 TSQLConnection.GetFieldNames Method (WideString, TWideStrings)

Populates a string list with the names of fields in a table.

Pascal

```
procedure GetFieldNames(const TableName: WideString; SchemaName: WideString; List:
TWideStrings); overload;
procedure GetFieldNames(const TableName: WideString; List: TWideStrings); overload;
```

C++

```
__fastcall GetFieldNames(const BSTR TableName, BSTR SchemaName, TWideStrings List);
__fastcall GetFieldNames(const BSTR TableName, TWideStrings List);
```

Description

Call GetFieldNames to retrieve a list of fields in a table.

TableName names the table whose field names you want added to the list.

List is a TStrings descendant that receives the field names. Any existing strings are deleted from the list before GetFieldNames adds the names of all the fields in TableName.

For example, the following line fills a list box with the names of all fields in the Employee table:

```
SQLConnection1.GetFieldNames('Employee', ListBox1.Items);
SQLConnection1->GetFieldNames("Employee", ListBox1->Items);
```

See Also

GetProcedureNames (see page 20232)

GetPackageNames (see page 20231)

GetTableNames (see page 20235)

GetIndexNames (see page 20231)

GetProcedureParams (see page 20233)

SetSchemaInfo (see page 20185)

149.1.5.4.17 GetIndexNames Method

This is the overview for the GetIndexNames method overload.

Overload List

	Name	Description
	TSQLConnection.GetIndexNames (WideString, TWideStrings) (see page 20231)	Populates a string list with the names of indexes on a table.

	TSQLConnection.GetIndexNames (WideString, WideString, TWideStrings) (see page 20231)	Populates a string list with the names of indexes on a table.

149.1.5.4.17.1 TSQLConnection.GetIndexNames Method (WideString, TWideStrings)

Populates a string list with the names of indexes on a table.

Pascal

```
procedure GetIndexNames(const TableName: WideString; List: TWideStrings); overload;
procedure GetIndexNames(const TableName: WideString; const SchemaName: WideString; List:
TWideStrings); overload;
```

C++

```
__fastcall GetIndexNames(const BSTR TableName, TWideStrings List);
__fastcall GetIndexNames(const BSTR TableName, const BSTR SchemaName, TWideStrings List);
```

Description

Call GetIndexNames to retrieve a list of the indexes defined for a table.

List is a TStrings descendant that receives the index names. Any existing strings are deleted from the list before GetIndexNames adds the names of all the indexes on TableName.

TableName names the table whose indexes you want added to the list.

For example, the following line fills a list box with the names of all indexes on the Employee table:

```
SQLConnection1.GetIndexNames('Employee', ListBox1.Items);
SQLConnection1->GetIndexNames("Employee", ListBox1->Items);
```

See Also

GetProcedureNames (see page 20232)

GetTableNames (see page 20235)

GetPackageNames (see page 20231)

GetFieldNames (see page 20230)

GetProcedureParams (see page 20233)

SetSchemaInfo (see page 20185)

Obtaining Information About Indexes

149.1.5.4.18 TSQLConnection.GetLoginUsername Method

Returns the username that is logged in on the current connection.

Pascal

```
function GetLoginUsername: WideString;
```

C++

```
__fastcall BSTR GetLoginUsername();
```

149.1.5.4.19 TSQLConnection.GetPackageNames Method

Populates a string list with the names of all packages defined on the server.

Pascal

```
procedure GetPackageNames(List: TWideStrings); overload;
```

C++

```
__fastcall GetPackageNames(TWideStrings List);
```

Description

Call GetPackageNames to retrieve a list of packages defined on the associated database. This method is only relevant to Oracle servers.

List is a TStrings descendant that receives the package names. Any existing strings are deleted from the list before GetPackageNames adds the names of all packages that the database defines.

The following example fills a list box with the names of all packages defined by the database:

```
SQLConnection1.GetPackageNames(ListBox1.Items);
SQLConnection1->GetPackageNames(ListBox1->Items);
```

See Also

GetIndexNames (see page 20231)

GetProcedureNames (see page 20232)

GetTableNames (see page 20235)

GetFieldNames (see page 20230)

GetProcedureParams (see page 20233)

SetSchemaInfo (see page 20185)

149.1.5.4.20 GetProcedureNames Method

This is the overview for the GetProcedureNames method overload.

Overload List

	Name	Description
	TSQLConnection.GetProcedureNames (TWideStrings) (see page 20232)	Populates a string list with the names of stored procedures in the database.
	TSQLConnection.GetProcedureNames (Widestring, TWideStrings) (see page 20232)	Populates a string list with the names of stored procedures in the database.
	TSQLConnection.GetProcedureNames (WideString, WideString, TWideStrings) (see page 20232)	Populates a string list with the names of stored procedures in the database.

149.1.5.4.20.1 TSQLConnection.GetProcedureNames Method (TWideStrings)

Populates a string list with the names of stored procedures in the database.

Pascal

```
procedure GetProcedureNames(List: TWideStrings); overload;
procedure GetProcedureNames(const PackageName: Widestring; List: TWideStrings); overload;
procedure GetProcedureNames(const PackageName: WideString; const SchemaName: WideString; List:
TWideStrings); overload;
```

C++

```
__fastcall GetProcedureNames(TWideStrings List);
__fastcall GetProcedureNames(const BSTR PackageName, TWideStrings List);
__fastcall GetProcedureNames(const BSTR PackageName, const BSTR SchemaName, TWideStrings List);
```

Description

Call GetProcedureNames to retrieve a list of stored procedures in the associated database.

List is a TStrings descendant that receives the stored procedure names. Any existing strings are deleted from the list before GetProcedureNames adds the names of all stored procedures in the database.

PackageName is the name of an Oracle package that contains the stored procedures. This syntax should only be used with Oracle servers when you only want the stored procedures from a specific package.

The following example fills a list box with the names of all stored procedures in the database:

```
SQLConnection1.GetProcedureNames(ListBox1.Items);
SQLConnection1->GetProcedureNames(ListBox1->Items);
```

See Also

GetIndexNames (see page 20231)

GetTableNames (see page 20235)

GetPackageNames (see page 20231)

GetFieldNames (see page 20230)

GetProcedureParams (see page 20233)

SetSchemaInfo (see page 20185)

149.1.5.4.21 GetProcedureParams Method

This is the overview for the GetProcedureParams method overload.

Overload List

	Name	Description
	TSQLConnection.GetProcedureParams (WideString, TList) (see page 20233)	Populates a list with parameter descriptors for the parameters in a stored procedure.
	TSQLConnection.GetProcedureParams (WideString, WideString, TList) (see page 20233)	Populates a list with parameter descriptors for the parameters in a stored procedure.
	TSQLConnection.GetProcedureParams (Widestring, Widestring, Widestring, TList) (see page 20233)	Populates a list with parameter descriptors for the parameters in a stored procedure.

149.1.5.4.21.1 TSQLConnection.GetProcedureParams Method (WideString, TList)

Populates a list with parameter descriptors for the parameters in a stored procedure.

Pascal

```
procedure GetProcedureParams(ProcedureName: WideString; List: TList); overload;
procedure GetProcedureParams(ProcedureName: WideString; PackageName: WideString; List: TList);
overload;
procedure GetProcedureParams(ProcedureName: Widestring; PackageName: Widestring; SchemaName:
Widestring; List: TList); overload;
```

C++

```
__fastcall GetProcedureParams(BSTR ProcedureName, TList List);
__fastcall GetProcedureParams(BSTR ProcedureName, BSTR PackageName, TList List);
__fastcall GetProcedureParams(BSTR ProcedureName, BSTR PackageName, BSTR SchemaName, TList
List);
```

Description

Call GetProcedureParams to retrieve information about the parameters of a specified stored procedure.

ProcedureName is the name of the stored procedure for whose parameters you want descriptions.

PackageName is the name of the Oracle package to which the stored procedure belongs. This parameter should only be used if TSQLConnection (see page 20217) connects to an Oracle server.

List is a TList object that receives the parameter descriptions. It should be an empty list on entry. For each parameter, GetProcedureParams adds a reference to an SPParamDesc (see page 20147) class (Delphi) or structure (C++). Each SPParamDesc (see page 20147) entity describes a parameter of a specified stored procedure, including its name, index, parameter type, field type, and so on. You are responsible for freeing the individual SPPParamDesc entities that GetProcedureParams dynamically allocates. (You can use FreeProcParams (see page 20361) to free all the entries and the list object as well).

Note: You can convert the list of SPParamDesc (see page 20147) entities to the more familiar TParams object by calling the global LoadParamListItems (see page 20363) procedure.

See Also

GetIndexNames (see page 20231)

GetTableNames (see page 20235)

GetPackageNames (see page 20231)

GetFieldNames (see page 20230)

GetProcedureNames (see page 20232)

SetSchemaInfo (see page 20185)

LoadParamListItems

FreeProcParams

149.1.5.4.22 TSQLConnection.GetSchemaNames Method

Returns the schema names for all objects in the given data source.

Pascal

```
procedure GetSchemaNames(List: TWideStrings); overload;
```

C++

```
__fastcall GetSchemaNames(TWideStrings List);
```

149.1.5.4.23 GetTableNames Method

This is the overview for the GetTableNames method overload.

Overload List

	Name	Description
	TSQLConnection.GetTableNames (TWideStrings, Boolean) (see page 20235)	Populates a string list with the names of tables in the database.
	TSQLConnection.GetTableNames (TWideStrings, WideString, Boolean) (see page 20235)	Populates a string list with the names of tables in the database.

149.1.5.4.23.1 TSQLConnection.GetTableNames Method (TWideStrings, Boolean)

Populates a string list with the names of tables in the database.

Pascal

```
procedure GetTableNames(List: TWideStrings; SchemaName: WideString; SystemTables: Boolean =
False); overload;
procedure GetTableNames(List: TWideStrings; SystemTables: Boolean = False); overload;
```

C++

```
__fastcall GetTableNames(TWideStrings List, BSTR SchemaName, Boolean SystemTables = False);
__fastcall GetTableNames(TWideStrings List, Boolean SystemTables = False);
```

Description

Call GetTableNames to retrieve a list of tables in the associated database.

List is a TStrings descendant that receives the table names. Any existing strings are deleted from the list before GetTableNames adds the names of all tables in the database.

SystemTables specifies whether the list of table names should include only the database's system tables. If SystemTables is true, only the system tables are added to List. If SystemTables is false, the list is filled with any tables that match the criteria specified by the TableScope (see page 20255) property.

For example, the following line fills a list box with the names of all data tables in the database:

```
SQLConnection1.GetTableNames(ListBox1.Items, False);
SQLConnection1->GetTableNames(ListBox1->Items, false);
```

See Also

GetIndexNames (see page 20231)

GetProcedureParams (see page 20233)

GetFieldNames (see page 20230)

GetProcedureNames (see page 20232)

GetPackageNames (see page 20231)

SetSchemaInfo (see page 20185)

TableScope (see page 20255)

149.1.5.4.24 TSQLConnection.Loaded Method

Initializes the connection component after all objects in the form or data module have been loaded.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. It is called automatically after the form or data module that contains the connection component has loaded all its owned objects. TCustomConnection overrides the inherited method to establish a connection if the Connected property is true.

See Also

StreamedConnected (see page 6874)

Loaded (see page 2782)

149.1.5.4.25 TSQLConnection.LoadParamsFromIniFile Method

Sets the DriverName (see page 20246) and Params (see page 20253) properties to the values associated with the ConnectionName (see page 20244) property.

Pascal

```
procedure LoadParamsFromIniFile(FFileName: WideString = '');
```

C++

```
__fastcall LoadParamsFromIniFile(BSTR FFileName = '');
```

Description

Call LoadParamsFromIniFile to ensure that the DriverName (see page 20246) and Params (see page 20253) properties accurately reflect the connection configuration for the ConnectionName (see page 20244) property that is stored in the file specified by the AFileName parameter. If the AFileName parameter is omitted, LoadParamsFromIniFile uses the dbxconnections.ini file. Otherwise, itis a fully-qualified path name to a file that has the same format as the dbxconnections.ini file. Typically, this file is created by making a copy of the dbxconnections.ini file under a different name.

In most cases, it is unnecessary to load connection configurations at runtime because when you set ConnectionName (see page 20244) at design time, the DriverName (see page 20246) and Params (see page 20253) properties are automatically set as well. However, LoadParamsFromIniFile lets you use a customized set of connection configurations at runtime that differ from those stored in the dbxconnections.ini file, or to temporarily override the DriverName (see page 20246) or connection parameter values at design time and then reestablish the defaults at runtime.

Note: Do not set the LoadParamsOnConnect (see page 20249) property to true if your application calls LoadParamsFromIniFile and specifies an alternate connections file. When LoadParamsOnConnect (see page 20249) is true, it loads the connection configuration from the dbxconnections.ini

file immediately before opening a connection, overriding any values loaded by this method.

See Also

ConnectionName (see page 20244)

LoadParamsOnConnect (see page 20249)

DriverName (see page 20246)

Params (see page 20253)

Setting Up TSQLConnection

Configuring TSQL Connection

149.1.5.4.26 TSQLConnection.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

149.1.5.4.27 OpenSchema Method

This is the overview for the OpenSchema method overload.

Overload List

	Name	Description
	TSQLConnection.OpenSchema (TSchemaType, Widestring, TWideStrings) (see page 20238)	This is OpenSchema, a member of class TSQLConnection.
	TSQLConnection.OpenSchema (TSchemaType, WideString, WideString, TWideStrings) (see page 20237)	This is OpenSchema, a member of class TSQLConnection.

149.1.5.4.27.1 TSQLConnection.OpenSchema Method (TSchemaType, WideString, WideString, TWideStrings)

Pascal

```
procedure OpenSchema(eKind: TSchemaType; sInfo: WideString; SSchemaName: WideString; List:
TWideStrings); overload;
```

C++

```
__fastcall OpenSchema(TSchemaType eKind, BSTR sInfo, BSTR SSchemaName, TWideStrings List);
```

Description

This is OpenSchema, a member of class TSQLConnection.

149.1.5.4.27.2 TSQLConnection.OpenSchema Method (TSchemaType, Widestring, TWideStrings)

Pascal

```
procedure OpenSchema(eKind: TSchemaType; sInfo: Widestring; List: TWideStrings); overload;
```

C++

```
__fastcall OpenSchema(TSchemaType eKind, BSTR sInfo, TWideStrings List);
```

Description

This is OpenSchema, a member of class TSQLConnection.

149.1.5.4.28 TSQLConnection.RegisterClient Method

Registers an object that uses the connection component.

Pascal

```
procedure RegisterClient(Client: TObject; Event: TConnectChangeEvent = nil); override;
```

C++

```
virtual __fastcall RegisterClient(TObject * Client, TConnectChangeEvent Event = nil);
```

Description

RegisterClient is used internally to associate objects with the connection component. Registered components can supply an event handler that responds when the connection is created or terminated.

Client is the object to register. If Client is a dataset, it is added to the DataSets property.

Event is an event handler that is called after the connection is established or before it is terminated.

Datasets call this method when they start using the connection component. For BDE-enabled datasets, ADO datasets, and dbExpress datasets, this is when they are activated. For client datasets, this is when the RemoteServer property is assigned.

See Also

DataSets (see page 6870)

UnRegisterClient (see page 6868)

149.1.5.4.29 TSQLConnection.Rollback Method

Cancels all updates, insertions, and deletions for a specified transaction and ends the transaction.

Pascal

```
procedure Rollback(TransDesc: TTransactionDesc); deprecated;
```

C++

```
__fastcall Rollback(TTransactionDesc TransDesc);
```

Description

Call Rollback to cancel any changes made during the transaction described by TransDesc and to end it.

TransDesc is the transaction descriptor supplied to the StartTransaction (see page 20240) method when starting the transaction.

If the specified transaction is the only current transaction, Rollback sets the InTransaction (see page 20247) property to false after canceling the transaction.

Note: This method has been deprecated.

See Also

InTransaction (see page 20247)

StartTransaction (see page 20240)

Commit (see page 20226)

TransactionsSupported (see page 20256)

TransactionLevel (see page 20202)

149.1.5.4.30 TSQLConnection.RollbackAndFree Method

Pascal

```
procedure RollbackAndFree(Transaction: TDBXTransaction);
```

C++

```
__fastcall RollbackAndFree(TDBXTransaction Transaction);
```

Description

This is RollbackAndFree, a member of class TSQLConnection.

149.1.5.4.31 TSQLConnection.SetTraceEvent Method

Associates the SQL connection with a callback function that executes for every SQL command passed to or from the server.

Pascal

```
procedure SetTraceEvent(Event: TDBXTraceEvent);
```

C++

```
__fastcall SetTraceEvent(TDBXTraceEvent Event);
```

Description

Call SetTraceEvent to assign a callback that responds every time an SQL command passes between the associated dbExpress driver and the database server. The callback function becomes the value of the TraceCallbackEvent (see page 20260) property.

Event is the TDBXTraceEvent event that is called for every message that passes between the dbExpress driver and the SQL server.

Tip: Read the TraceCallbackEvent (see page 20260) property to determine whether a callback is already assigned before calling the SetTraceEvent method. That way, you can avoid interfering with an existing callback when assigning a new one.

See Also

TSQLMonitor (see page 20285)

TraceCallbackEvent (see page 20260)

Debugging dbExpress Applications

Debugging dbExpress Applications using TSQLMonitor

149.1.5.4.32 TSQLConnection.SQLError Method

Pascal

```
procedure SQLError(Error: TDBXError);
```

C++

```
__fastcall SQLError(TDBXError Error);
```

Description

This is SQLError, a member of class TSQLConnection.

149.1.5.4.33 TSQLConnection.StartTransaction Method

Initiates a new transaction in the associated database.

Pascal

```
procedure StartTransaction(TransDesc: TTransactionDesc); deprecated;
```

C++

```
__fastcall StartTransaction(TTransactionDesc TransDesc);
```

Description

Call StartTransaction to start a new transaction against a database server that supports transactions. (MySQL servers do not support transactions.) You can check the TransactionsSupported (see page 20256) property to determine whether the server supports transactions. If the server does not support transactions, StartTransaction does nothing.

TransDesc describes the transaction you are starting. It lets you specify

A user-defined transaction identifier. If the server supports nested or overlapped transactions, you can initiate multiple transactions by calling StartTransaction and providing each transaction with its own unique identifier.

The transaction isolation level. This indicates the degree to which this transaction can "see" changes made by other transactions.

If the transaction is successfully started, StartTransaction sets the InTransaction (see page 20247) property to true.

Updates, insertions, and deletions that take place after a call to StartTransaction are enlisted in that transaction until another overlapping transaction is started or until the application calls Commit (see page 20226) to save the changes or Rollback (see page 20238) to cancel them.

Tip: If the server does not support multiple transactions, check the status of the InTransaction (see page 20247) property to be sure that another transaction is not already in progress.

Tip: When working with overlapped transactions, you can indicate the transaction to which a dataset belongs by setting its TransactionLevel property to the transaction identifier you assign in this method.

Note: This method has been deprecated.

See Also

InTransaction (see page 20247)

Commit (see page 20226)

Rollback (see page 20238)

Params (see page 20253)

TransactionsSupported (see page 20256)

TransactionLevel (see page 20202)

149.1.5.4.34 TSQLConnection.UnRegisterClient Method

Removes a registered client so that it is no longer associated with the connection component.

Pascal

```
procedure UnRegisterClient(Client: TObject); override;
```

C++

```
virtual __fastcall UnRegisterClient(TObject * Client);
```

Description

UnRegisterClient is used internally to remove objects from the internal list of objects that are informed when a connection is created or terminated.

Client is the object that is removed from the list. It should be an object that was previously registered by a call to RegisterClient. If Client is a dataset, it is removed from the DataSets property list.

Datasets call this method when they stop using the connection component. For BDE-enabled datasets, ADO datasets, and dbExpress datasets, this is when they are deactivated. For client datasets, this is when the RemoteServer property is changed.

See Also

DataSets (see page 6870)

RegisterClient (see page 6866)

149.1.5.5 TSQLConnection Properties

The properties of the TSQLConnection class are listed here.

Protected Properties

	Name	Description
R	Connection (see page 20244)	This is Connection, a member of class TSQLConnection.
R	ConnectionRegistryFile (see page 20245)	This is ConnectionRegistryFile, a member of class TSQLConnection.
R	DriverRegistryFile (see page 20247)	This is DriverRegistryFile, a member of class TSQLConnection.
	LastError (see page 20248)	This is LastError, a member of class TSQLConnection.
R	QuoteChar (see page 20254)	This is QuoteChar, a member of class TSQLConnection.
	SQLDllHandle (see page 20255)	This is SQLDllHandle, a member of class TSQLConnection.

Public Properties

	Name	Description
R	ActiveStatements (see page 20242)	Indicates the number of statements currently active with the database server.
	AutoClone (see page 20243)	Specifies whether the SQL connection automatically clones database connections when they are needed.
	ConnectionState (see page 20245)	Indicates the current state of the TSQLConnection (see page 20217) object.
R	DataSets (see page 20245)	Lists all active datasets for the connection component.
R	InTransaction (see page 20247)	Indicates whether a transaction is in progress.
	LocaleCode (see page 20250)	Identifies the locale that is used for sorting items from datasets that use this connection.
R	MaxStmtsPerConn (see page 20251)	Indicates the server limit on the number of statements supported by a single database connection.
R	MetaData (see page 20252)	Provides an interface to the dbExpress object that supplies server metadata.
R	MultipleTransactionsSupported (see page 20253)	Indicates whether the database server supports multiple transactions.
	ParamsLoaded (see page 20254)	Indicates when parameter values have been loaded from dbxconnections.ini.

	SQLConnection (see page 20255)	This is SQLConnection, a member of class TSQLConnection.
	SQLHourGlass (see page 20255)	Specifies whether the cursor should change to crSQLWait during lengthy operations.
R	TransactionsSupported (see page 20256)	Indicates whether the database server supports transactions.

Published Properties

	Name	Description
	Connected (see page 20243)	Specifies whether the connection is active.
	ConnectionName (see page 20244)	Names the connection configuration.
	DriverName (see page 20246)	Specifies the database driver associated with the SQL connection.
	GetDriverFunc (see page 20247)	Specifies the function that returns the top-level dbExpress driver.
	KeepConnection (see page 20248)	Specifies whether the connection stays active when no datasets are open.
	LibraryName (see page 20248)	Specifies the dbExpress driver that TSQLConnection (see page 20217) uses to establish a connection.
	LoadParamsOnConnect (see page 20249)	Specifies whether the TSQLConnection (see page 20217) component loads the configuration named by ConnectionName (see page 20244) immediately before connecting to the server.
	LoginPrompt (see page 20250)	Specifies whether a login dialog appears immediately before opening a new connection.
	Params (see page 20253)	Lists the connection parameters.
	TableScope (see page 20255)	Indicates what types of tables are returned when fetching schema information about tables.
	VendorLib (see page 20257)	Specifies the client software library (DLL or shared object) supplied by the database vendor.

149.1.5.5.1 TSQLConnection.ActiveStatements Property

Indicates the number of statements currently active with the database server.

Pascal

```
property ActiveStatements: LongWord;
```

C++

```
__property LongWord ActiveStatements;
```

Description

Read ActiveStatements to determine how many statements are currently executing using the database connection. While a dataset is prepared, it creates an active statement that lasts until the dataset is unprepared. When you set a dataset's Prepared property to true, the statement remains active until you change the Prepared property to false. When the Prepared property is false, the dataset implicitly prepares its statement when you open the dataset, and the statement remains active until you close the dataset. In addition, executing an SQL statement (for example, using the Execute (see page 20227) method) creates an active statement temporarily, until the statement finishes executing.

Use ActiveStatements to check whether an application has reached the server-imposed limit on the number of active statements allowed to a connection (the MaxStmtsPerConn (see page 20251) property). This is especially important if AutoClone (see page 20243) is false, because if you exceed MaxStmtsPerConn (see page 20251) when AutoClone (see page 20243) is false, the SQL connection raises an exception. To avoid exceeding the server-imposed limit, do not open any datasets or execute SQL statements when ActiveStatements is the same as MaxStmtsPerConn (see page 20251) (unless MaxStmtsPerConn (see page 20251) is 0).

See Also

DataSets (see page 20245)

Execute (see page 20227)

MaxStmtsPerConn (see page 20251)

Prepared (see page 20198)

AutoClone (see page 20243)

149.1.5.5.2 TSQLConnection.AutoClone Property

Specifies whether the SQL connection automatically clones database connections when they are needed.

Pascal

```
property AutoClone: Boolean;
```

C++

```
__property Boolean AutoClone;
```

Description

AutoClone controls whether the TSQLConnection (see page 20217) component automatically creates duplicate database connections when the number of statements allowed by the database server is exceeded. By default, AutoClone is true, meaning that when the ActiveStatements (see page 20242) property is the same as MaxStmtsPerConn (see page 20251), trying to execute another statement (open another dataset or call the Execute (see page 20227) method) causes TSQLConnection (see page 20217) to open a secondary (cloned) connection. The result behaves just like a connection to a server that does not limit the number of statements, except that it consumes more resources (for the additional, cloned connections).

Change AutoClone to false to prevent TSQLConnection (see page 20217) from cloning the database connection when the application attempts to exceed the maximum number of statements. When AutoClone is false, attempts to exceed MaxStmtsPerConn (see page 20251) result in an exception.

Tip: When AutoClone is false, you can check the ActiveStatements (see page 20242) property to ensure that you do not exceed the server-imposed limit on the number of statements. If you are about to exceed that limit, you can explicitly call CloneConnection (see page 20225) to clone the connection yourself.

See Also

MaxStmtsPerConn (see page 20251)

ActiveStatements (see page 20242)

CloneConnection (see page 20225)

Connecting to the Database Server

149.1.5.5.3 TSQLConnection.Connected Property

Specifies whether the connection is active.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Set Connected to trueto establish a connection to the database server. When a dataset that uses the TSQLConnection (see page 20217) object is activated, it automatically sets Connected to true. Set Connected to false to close a connection.

Setting Connected to true generates a BeforeConnect (see page 20258) event, establishes the connection, and generates an AfterConnect (see page 20258) event. In addition, when setting Connected to true, TSQLConnection (see page 20217) may display a login dialog, depending on the value of LoginPrompt (see page 20250).

Setting Connected to false generates a BeforeDisconnect (see page 20259) event, drops the connection, and generates an

AfterDisconnect (see page 20258) event.

See Also

Open (see page 6866)

Close (see page 6863)

ConnectionState (see page 20245)

BeforeConnect (see page 6875)

AfterConnect (see page 6874)

BeforeDisconnect (see page 6875)

AfterDisconnect (see page 6875)

LoginPrompt (see page 6872)

149.1.5.5.4 TSQLConnection.Connection Property

Pascal

```
property Connection: TDBXConnection;
```

C++

```
__property TDBXConnection Connection;
```

Description

This is Connection, a member of class TSQLConnection.

149.1.5.5.5 TSQLConnection.ConnectionName Property

Names the connection configuration.

Pascal

```
property ConnectionName: string;
```

C++

```
__property AnsiString ConnectionName;
```

Description

Set ConnectionName to use a named connection configuration. Setting ConnectionName automatically sets the DriverName (see page 20246) and Params (see page 20253) properties to reflect the driver and connection parameters stored under that name in the dbxconnections.ini file.

Tip: You can define new connection configurations using the Connection (see page 20244) Editor. To launch the Connection (see page 20244) Editor, double-click on the TSQLConnection (see page 20217) component.

If you are using a different dbxconnections.ini file at design time than the one used at runtime, you can force the configuration to reload at runtime by setting the LoadParamsOnConnect (see page 20249) property or calling LoadParamsFromIniFile (see page 20236). In most cases, however, this is unnecessary, because the DriverName (see page 20246) and Params (see page 20253) property settings that are loaded when you set ConnectionName are saved in the form file. In fact, unless you set LoadParamsOnConnect (see page 20249) or call LoadParamsFromIniFile (see page 20236), you do not need to deploy the dbxconnections.ini file with your application even if you are using named connections.

See Also

LoadParamsOnConnect (see page 20249)

LoadParamsFromIniFile (see page 20236)

DriverName (see page 20246)

Params (see page 20253)

Setting Up a Simple Dataset

Setting Up TSQLConnection

Configuring TSQL Connection

149.1.5.5.6 TSQLConnection.ConnectionRegistryFile Property

Pascal

```
property ConnectionRegistryFile: string;
```

C++

```
__property AnsiString ConnectionRegistryFile;
```

Description

This is ConnectionRegistryFile, a member of class TSQLConnection.

149.1.5.5.7 TSQLConnection.ConnectionState Property

Indicates the current state of the TSQLConnection (see page 20217) object.

Pascal

```
property ConnectionState: TConnectionState;
```

C++

```
__property TConnectionState ConnectionState;
```

Description

Read ConnectionState at runtime to determine the activity being performed, if any, by the connection at a specific point in time. Use ConnectionState to coordinate the activities of the connection component, so that you do not start one activity while it is currently engaged in another.

See Also

Open (see page 6866)

Connected (see page 20243)

Close (see page 6863)

Execute (see page 20227)

149.1.5.5.8 TSQLConnection.DataSets Property

Lists all active datasets for the connection component.

Pascal

```
property DataSets [Index: Integer]: TCustomSQLDataSet;
```

C++

```
__property TCustomSQLDataSet DataSets[int Index];
```

Description

Use DataSets to access active datasets associated with a connection component. An active dataset is one that is currently open.

The Index parameter ranges between 0 and DataSetCount.

The value of DataSets changes as the datasets that use this TSQLConnection (see page 20217) component open and close.

Note: If a TSQLClientDataSet uses the connection component to connect to a database server, DataSets lists the internal dataset that the client dataset creates to access data, not the client dataset itself.

See Also

DataSetCount (see page 6869)

Active (see page 7004)

MaxStmtsPerConn (see page 20251)

TSQLClientDataSet

149.1.5.5.9 TSQLConnection.DriverName Property

Specifies the database driver associated with the SQL connection.

Pascal

```
property DriverName: string;
```

C++

```
__property AnsiString DriverName;
```

Description

If you are not using named connections, set DriverName to the name of an installed dbExpress driver such as INTERBASE, MYSQL, INFORMIX, ORACLE, or DB2. Installed drivers are listed in the dbxdrivers.ini file, and their names can be retrieved using the GetDriverNames (see page 20362) function.

If you are using named connections, the DriverName property is automatically set when you set the ConnectionName (see page 20244) property.

When you set DriverName, TSQLConnection (see page 20217) automatically sets the LibraryName (see page 20248) and VendorLib (see page 20257) properties to the names of the libraries (DLLs or shared objects) that implement the specified driver.

See Also

LibraryName (see page 20248)

VendorLib (see page 20257)

ConnectionName (see page 20244)

GetDriverNames (see page 20362)

Setting Up TSQLConnection

Configuring TSQL Connection

Connecting to Databases with TDatabase

149.1.5.5.10 TSQLConnection.DriverRegistryFile Property

Pascal

```
property DriverRegistryFile: string;
```

C++

```
__property AnsiString DriverRegistryFile;
```

Description

This is DriverRegistryFile, a member of class TSQLConnection.

149.1.5.5.11 TSQLConnection.GetDriverFunc Property

Specifies the function that returns the top-level dbExpress driver.

Pascal

```
property GetDriverFunc: string;
```

C++

```
__property AnsiString GetDriverFunc;
```

Description

GetDriverFunc is the name of a function in the dbExpress driver specified by the LibraryName (see page 20248) property. The specified function returns the top-level dbExpress interface. Typically, this property is set to the function name that is stored in the dbxdrivers.ini file.

Note: The GetDriverFunc, LibraryName (see page 20248), and VenderLib properties all reflect the values in the dbxdrivers.ini

file. Because these property values are published (stored with the connection component), it is possible to deploy an application without deploying the dbxdrivers.ini file.

See Also

LibraryName (see page 20248)

VendorLib (see page 20257)

149.1.5.5.12 TSQLConnection.InTransaction Property

Indicates whether a transaction is in progress.

Pascal

```
property InTransaction: Boolean;
```

C++

```
__property Boolean InTransaction;
```

Description

Read InTransaction at runtime to ascertain whether a transaction is currently active for the SQL connection component. If the connection component is currently in a transaction, all communication with the server that passes through the connection is enlisted in that transaction.

InTransaction is true if there is an active transaction. If no transaction is active, InTransaction is false.

A transaction begins when the application calls BeginTransaction (see page 20224) and lasts until the application calls CommitFreeAndNil, RollbackFreeAndNil, or RollbackIncompleteFreeAndNi. If there are multiple transactions, InTransaction

remains true until all transactions are either committed or rolled back.

See Also

BeginTransaction (see page 20224)

CommitFreeAndNil

RollbackFreeAndNil

RollbackIncompleteFreeAndNi

TransactionsSupported (see page 20256)

TransactionLevel (see page 20202)

MultipleTransactionsSupported (see page 20253)

149.1.5.5.13 TSQLConnection.KeepConnection Property

Specifies whether the connection stays active when no datasets are open.

Pascal

```
property KeepConnection: Boolean;
```

C++

```
__property Boolean KeepConnection;
```

Description

Use KeepConnection to specify whether TSQLConnection (see page 20217) maintains an open database connection even if no associated dataset components are currently active. When KeepConnection is true (the default) the connection is maintained. For connections to remote database servers, or in applications that frequently open and close datasets, set KeepConnection to true to reduce network traffic, speed up applications, and avoid logging in to the server each time the connection is reestablished.

When KeepConnection is false a connection is dropped when there are no open datasets. Dropping a connection releases system resources allocated to the connection, but if a dataset is later opened that uses the database, the connection must be reestablished and initialized.

See Also

DataSets (see page 20245)

Connected (see page 20243)

149.1.5.5.14 TSQLConnection.LastError Property

Pascal

```
property LastError: TDBXError;
```

C++

```
__property TDBXError LastError;
```

Description

This is LastError, a member of class TSQLConnection.

149.1.5.5.15 TSQLConnection.LibraryName Property

Specifies the dbExpress driver that TSQLConnection (see page 20217) uses to establish a connection.

Pascal

```
property LibraryName: string;
```

C++

```
__property AnsiString LibraryName;
```

Description

LibraryName is the dbExpress library associated with the driver specified by DriverName (see page 20246). Most applications do not need to use LibraryName directly, because this property is set automatically when the DriverName (see page 20246) property is set.

Note: The association between a dbExpress library and a driver name is stored in the dbxdrivers.ini

file when you install the dbExpress driver. The file named by this property is the only dbExpress file that must be deployed with your application.

Note: The LibraryName, GetDriverFunc (see page 20247), and VenderLib properties all reflect the values in the dbxdrivers.ini

file. Because these property values are published (stored with the connection component), it is possible to deploy an application without deploying the dbxdrivers.ini file.

See Also

DriverName (see page 20246)

VendorLib (see page 20257)

GetDriverFunc (see page 20247)

Setting Up TSQLConnection

Configuring TSQL Connection

149.1.5.5.16 TSQLConnection.LoadParamsOnConnect Property

Specifies whether the TSQLConnection (see page 20217) component loads the configuration named by ConnectionName (see page 20244) immediately before connecting to the server.

Pascal

```
property LoadParamsOnConnect: Boolean;
```

C++

```
__property Boolean LoadParamsOnConnect;
```

Description

Set LoadParamsOnConnect to true if you want the TSQLConnection (see page 20217) object to load the connection configuration (DriverName (see page 20246) and Params (see page 20253) properties) associated with ConnectionName (see page 20244) at runtime. Typically, this is unnecessary because when you set ConnectionName (see page 20244) at design time, the DriverName (see page 20246) and Params (see page 20253) properties are automatically set as well. However, if you use a different dbxconnections.ini file at runtime, or if you temporarily override the DriverName (see page 20246) or any connection parameter values at design time, setting LoadParamsOnConnect to true ensures that at runtime the connection is established using the configuration associated with ConnectionName (see page 20244) in the dbxconnections.ini file.

Note: If you set LoadParamsOnConnect to true, you must be sure to deploy the dbxconnections.ini

file with your application. When you rely on the design-time associations, dbxconnections.ini is not needed at runtime.

Warning: If you set LoadParamsOnConnect to true, you must specify a valid connection name as the value for

ConnectionName (see page 20244).

See Also

ConnectionName (see page 20244)

LoadParamsFromIniFile (see page 20236)

DriverName (see page 20246)

Params (see page 20253)

Setting Up TSQLConnection

Configuring TSQL Connection

149.1.5.5.17 TSQLConnection.LocaleCode Property

Identifies the locale that is used for sorting items from datasets that use this connection.

Pascal

```
property LocaleCode: TLocaleCode;
```

C++

```
__property TLocaleCode LocaleCode;
```

Description

LocaleCode represents the locale that client datasets use to sort strings when they are provided in fields from a dataset associated with this connection. By default, this value is 0, which signals that strings should be sorted without regard to a locale. Changing LocaleCode can affect performance, because sorting is slower when it must take locale into account. Therefore, it is recommended that you only change LocaleCode if the data is known to contain extended characters that will affect the sort order.

Note: Locale codes are treated as connection parameters: they are stored in the dbxconnections.ini

file along with the other parameters, and appear in the Params (see page 20253) property list. Changes to this property change the entry in the Params (see page 20253) property list.

See Also

DataSets (see page 20245)

SQLConnection (see page 20201)

LoadParamsFromIniFile (see page 20236)

Params (see page 20253)

LoadParamsOnConnect (see page 20249)

TClientDataSet (see page 8289)

149.1.5.5.18 TSQLConnection.LoginPrompt Property

Specifies whether a login dialog appears immediately before opening a new connection.

Pascal

```
property LoginPrompt: Boolean;
```

C++

```
__property Boolean LoginPrompt;
```

Description

Set LoginPrompt to true to provide login support when establishing a connection. LoginPrompt controls two things: the occurrence of the OnLogin event, and the appearance of a default login dialog that prompts users for a name and password when you include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi). When the OnLogin event occurs and when the default login dialog appears depend on the type of connection component:

For TDatabase, TIBDatabase, and TSQLConnection, the dialog appears after the BeforeConnect event and before the AfterConnect event, unless you supply an OnLogin event handler. If there is an OnLogin event handler, that event occurs in place of the login dialog, and there is no need to include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi). If correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler, the connection fails. The OnLogin event does not fire unless LoginPrompt is set to true.

For TADOConnection components, the dialog appears after the OnWillConnect event and before the BeforeConnect event. If there is an OnLogin event handler, that event occurs after the login dialog. If you do not include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi), the OnLogin event occurs but there is no default login dialog. If correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler, the connection fails. The OnLogin event does not fire unless LoginPrompt is set to true.

For DataSnap SOAP connection components, the dialog appears after the OnGetUsername event and before the BeforeConnect, AfterConnect, and OnLogin events. If the user cancels from the login dialog, no attempt is made to open a connection.

When LoginPrompt is false, the application must supply user name and password values programmatically:

For TDatabase, the user name and password can be supplied as USER NAME and PASSWORD parameters in the Params property.

For TADOConnection, the user name and password can be supplied as the ConnectionString property.

For TSQLConnection, the user name, password, and database can be supplied as UserName, Password, and Database parameters in the Params property, or provided as connection parameters associated with the connection name.

For DataSnap SOAP connection components, there is no built-in use for the user name and password supplied by the login dialog. (The UserName and Password properties of TWebConnection are unrelated).

Warning: Storing hard-coded user name and password entries as property values or in code for an OnLogin event handler can compromise server security.

See Also

OnLogin (see page 6873)

BeforeConnect (see page 6875)

AfterConnect (see page 6874)

OnGetUserName (see page 8533)

OnLogin (see page 20259)

OnLogin (see page 9994)

Controlling Server Login

149.1.5.5.19 TSQLConnection.MaxStmtsPerConn Property

Indicates the server limit on the number of statements supported by a single database connection.

Pascal

```
property MaxStmtsPerConn: LongWord;
```

C++

```
__property LongWord MaxStmtsPerConn;
```

Description

Read MaxStmtsPerConn when a connection is active to determine whether the database server imposes a limit on the number of statements that can be executed using the current connection. When MaxStmtsPerConn is 0, the server imposes no restrictions: a single database connection can be used to fetch multiple datasets and execute statements while there are active datasets. When MaxStmtsPerConn is greater than 0, the database server limits the number of statements that can execute. The server allows only MaxStmtsPerConn datasets to be opened using a single connection, and when that number of datasets are open, no more statements can be executed (for example, using the Execute (see page 20227) method).

By default, when MaxStmtsPerConn is greater than 0, TSQLConnection (see page 20217) clones the database connection when this server-imposed limit is reached. For example, if MaxStmtsPerConn is 1, then trying to open a second dataset using the same TSQLConnection (see page 20217) component causes TSQLConnection (see page 20217) to establish a second database connection for the second dataset. If AutoClone (see page 20243) is false, however, TSQLConnection (see page 20217) does not clone connections, but rather generates an exception when the application tries to exceed MaxStmtsPerConn.

Note: The value of MaxStmtsPerConn is only valid when the Connected (see page 20243) property is true.

See Also

DataSets (see page 20245)

Execute (see page 20227)

Connected (see page 20243)

AutoClone (see page 20243)

ActiveStatements (see page 20242)

Connecting to the Database Server

149.1.5.5.20 TSQLConnection.MetaData Property

Provides an interface to the dbExpress object that supplies server metadata.

Pascal

```
property MetaData: TDBXMetaData;
```

C++

```
__property TDBXMetaData MetaData;
```

Description

Typically, applications do not use the MetaData property to work directly with the dbExpress MetaData interface. Instead, they use one of methods that assembles schema information into a list (GetFieldNames (see page 20230), GetIndexNames (see page 20231), GetProcedureNames (see page 20232), GetProcedureParams (see page 20233), or GetTableNames (see page 20235)). When more detailed metadata is required, you can also use the SQL dataset's SetSchemaInfo method to populate an SQL dataset with metadata.

See Also

GetFieldNames (see page 20230)

GetIndexNames (see page 20231)

GetProcedureNames (see page 20232)

GetProcedureParams (see page 20233)

GetTableNames (see page 20235)

SetSchemaInfo (see page 20185)

149.1.5.5.21 TSQLConnection.MultipleTransactionsSupported Property

Indicates whether the database server supports multiple transactions.

Pascal

```
property MultipleTransactionsSupported: LongBool;
```

C++

```
__property LongBool MultipleTransactionsSupported;
```

Description

Read MultipleTransactionsSupported at runtime to ascertain whether the database server supports multiple transactions (either nested or overlapping).

Note: To determine whether the server supports transactions at all, use the TransactionsSupported (see page 20256) property. If a server supports only a single transaction at a time, TransactionsSupported (see page 20256) is true, and MultipleTransactionsSupported is false.

See Also

CommitFreeAndNil

BeginTransaction (see page 20224)

RollbackFreeAndNil

InTransaction (see page 20247)

TransactionsSupported (see page 20256)

149.1.5.5.22 TSQLConnection.Params Property

Lists the connection parameters.

Pascal

```
property Params: TWideStrings;
```

C++

```
__property TWideStrings Params;
```

Description

Params is a TStrings object listing the connection parameters. Each entry in the string list has the form

```
Name=Value
```

where Name is the name of the parameter and Value is its value. If you are using named connection configurations (the ConnectionName (see page 20244) property), the values for each parameter are loaded from the dbxconnections.ini file when you set ConnectName.

At design time, you can set parameter values using the string list editor by double clicking on the Params property in the Object Inspector. Any new values you set override the values loaded when you set ConnectionName (see page 20244) unless you

load the connection configuration at runtime using LoadParamsFromIniFile (see page 20236) or LoadParamsOnConnect (see page 20249).

The particular parameters that appear depend on the database server. The Params property is initialized to include a string for each parameter when you set the DriverName (see page 20246) property. One particular parameter, Database, is required for all servers. Its value depends on the server you are using. For example, with InterBase, Database is the name of the .gdb file, while with MySQL, it is the database name that was assigned by the CREATE DATABASE command.

Other typical parameters include HostName (the machine name or IP address of where the server is located), and TransIsolation (the degree to which transactions are isolated from changes made by other users of the database).

See Also

ConnectionName (see page 20244)

LoadParamsFromIniFile (see page 20236)

LoadParamsOnConnect (see page 20249)

DriverName (see page 20246)

LocaleCode (see page 20250)

Setting Up TSQLConnection

149.1.5.5.23 TSQLConnection.ParamsLoaded Property

Indicates when parameter values have been loaded from dbxconnections.ini.

Pascal

```
property ParamsLoaded: Boolean;
```

C++

```
__property Boolean ParamsLoaded;
```

Description

Do not use the ParamsLoaded property. It is used internally by the Connections Editor to keep track of when parameter values have been loaded from the dbxconnections.ini file.

See Also

Params (see page 20253)

LoadParamsFromIniFile (see page 20236)

LoadParamsOnConnect (see page 20249)

149.1.5.5.24 TSQLConnection.QuoteChar Property

Pascal

```
property QuoteChar: WideString;
```

C++

```
__property BSTR QuoteChar;
```

Description

This is QuoteChar, a member of class TSQLConnection.

149.1.5.5.25 TSQLConnection.SQLConnection Property

Pascal

```
property SQLConnection: TDBXConnection;
```

C++

```
__property TDBXConnection SQLConnection;
```

Description

This is SQLConnection, a member of class TSQLConnection.

149.1.5.5.26 TSQLConnection.SQLDllHandle Property

Pascal

```
property SQLDllHandle: THandle;
```

C++

```
__property THandle SQLDllHandle;
```

Description

This is SQLDllHandle, a member of class TSQLConnection.

149.1.5.5.27 TSQLConnection.SQLHourGlass Property

Specifies whether the cursor should change to crSQLWait during lengthy operations.

Pascal

```
property SQLHourGlass: Boolean;
```

C++

```
__property Boolean SQLHourGlass;
```

Description

Set SQLHourGlass to true to specify that the cursor should indicate when the connection is performing a potentially lengthy operation. Such operations include

Establishing a connection to the database server.

Fetching metadata from the database server.

Executing a statement using the Execute (see page 20227) method.

When SQLHourGlass is false, the cursor is not changed when the SQL connection performs such operations.

See Also

Execute (see page 20227)

Open (see page 6866)

TCursorType

149.1.5.5.28 TSQLConnection.TableScope Property

Indicates what types of tables are returned when fetching schema information about tables.

Pascal

```
property TableScope: TTableScopes;
```

C++

```
__property TTableScopes TableScope;
```

Description

Set TableScope to determine the type of tables listed by the GetTableNames (see page 20235) method or that populate an SQL dataset after calling its SetSchemaInfo method with a parameter of stTables. TableScope lets you request information about a combination of table types. By default, TableScope requests only ordinary tables and views (tsTable and tsView).

Note: TableScope only affects GetTableNames (see page 20235) when the SystemTables parameter is false and only affects SetSchemaInfo when the SchemaType parameter is stTables. If an application calls GetTablesNames with SystemTables set to true, or calls SetSchemaInfo with SchemaType set to stSysTables, system tables, and only system tables are returned, regardless of the value of TableScope.

See Also

GetTableNames (see page 20235)

SetSchemaInfo (see page 20185)

Fetching Metadata into a Unidirectional Dataset

149.1.5.5.29 TSQLConnection.TransactionsSupported Property

Indicates whether the database server supports transactions.

Pascal

```
property TransactionsSupported: LongBool;
```

C++

```
__property LongBool TransactionsSupported;
```

Description

Read TransactionsSupported at runtime to ascertain whether the database server supports transactions. If TransactionsSupported is false, then the BeginTransaction (see page 20224), CommitFreeAndNil, RollbackFreeAndNil, and RollbackIncompleteFreeAndNil methods do nothing. If TransactionsSupported is true, you can use the SQL connection component to manage transactions.

Note: Some servers support multiple transactions (either nested or overlapping). When TransactionsSupported is true, you can test whether the server supports multiple transactions by reading the MultipleTransactionsSupported (see page 20253) property.

See Also

BeginTransaction (see page 20224)

CommitFreeAndNil

RollbackFreeAndNil

RollbackIncompleteFreeAndNil

InTransaction (see page 20247)

MultipleTransactionsSupported (see page 20253)

149.1.5.5.30 TSQLConnection.VendorLib Property

Specifies the client software library (DLL or shared object) supplied by the database vendor.

Pascal

```
property VendorLib: string;
```

C++

```
__property AnsiString VendorLib;
```

Description

VendorLib is the library supplied by the database vendor to support client-side use of the database. Most applications do not need to use VendorLib directly, because this property is set automatically when the DriverName (see page 20246) property is set.

Note: The association between a vendor library and a driver name is stored in the dbxdrivers.ini

file when you install the associated dbExpress driver. The file named by this property must be running on the same system as your application when it is deployed. Typically, it is installed when the database is installed rather than with your application.

Note: The VendorLib, LibraryName (see page 20248), and GetDriverFunc (see page 20247) properties all reflect the values in the dbxdrivers.ini

file. Because these property values are published (stored with the connection component), it is possible to deploy an application without deploying the dbxdrivers.ini file.

See Also

DriverName (see page 20246)

VendorLib

GetDriverFunc (see page 20247)

Setting Up TSQLConnection

Configuring TSQL Connection

149.1.5.6 TSQLConnection Events

The events of the TSQLConnection class are listed here.

Public Events

	Name	Description
R	TraceCallbackEvent (see page 20260)	Provides access to the callback function that executes for every SQL command passed to or from the server.

Published Events

	Name	Description
	AfterConnect (see page 20258)	Occurs after a connection is established.
	AfterDisconnect (see page 20258)	Occurs after the connection closes.
	BeforeConnect (see page 20258)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 20259)	Occurs immediately before the connection closes.
	OnLogin (see page 20259)	Occurs after the BeforeConnect (see page 20258) event and before a connection is established.

149.1.5.6.1 TSQLConnection.AfterConnect Event

Occurs after a connection is established.

Pascal

```
property AfterConnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterConnect;
```

Description

Write an AfterConnect event handler to take application-specific actions immediately after the connection component opens a connection to the remote source of database information.

See Also

BeforeConnect (see page 6875)

Connected (see page 6868)

Connecting to a Database Server

149.1.5.6.2 TSQLConnection.AfterDisconnect Event

Occurs after the connection closes.

Pascal

```
property AfterDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterDisconnect;
```

Description

Write an AfterDisconnect event handler to take application-specific actions after the connection component drops a connection.

See Also

BeforeDisconnect (see page 6875)

Connected (see page 6868)

Disconnecting from a Database Server

149.1.5.6.3 TSQLConnection.BeforeConnect Event

Occurs immediately before establishing a connection.

Pascal

```
property BeforeConnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeConnect;
```

Description

Write a BeforeConnect event handler to take application-specific actions before the connection component opens a connection to the remote source of database information.

See Also

AfterConnect (see page 6874)

Connected (see page 6868)

Connecting to a Database Server

149.1.5.6.4 TSQLConnection.BeforeDisconnect Event

Occurs immediately before the connection closes.

Pascal

```
property BeforeDisconnect: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeDisconnect;
```

Description

Write a BeforeDisconnect event handler to take application-specific actions before dropping a connection.

Delphi Examples:

```
{
This BeforeDisconnect event handler makes sure that the
client dataset has applied all its pending updates before
terminating the connection to the server.
}
procedure TForm1.RemoteServer1BeforeDisconnect(Sender: TObject);
begin
  if (ClientDataSet1.ChangeCount > 0) then
    ClientDataSet1.ApplyUpdates(-1);
end;
```

See Also

AfterDisconnect (see page 6875)

Connected (see page 6868)

Disconnecting from a Database Server

149.1.5.6.5 TSQLConnection.OnLogin Event

Occurs after the BeforeConnect (see page 20258) event and before a connection is established.

Pascal

```
property OnLogin: TSQLConnectionLoginEvent;
```

C++

```
__property TSQLConnectionLoginEvent OnLogin;
```

Description

Use the OnLogin event to assign values to the User_Name, Password, and Database parameters immediately before TSQLConnection (see page 20217) attempts to connect to the database server. OnLogin only occurs if the LoginPrompt (see page 20250) property is true. If LoginPrompt (see page 20250) is true but there is no OnLogin event handler, a default login dialog appears in which the user can enter a user name and password. The connection fails if correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler.

See Also

Params (see page 20253)

LoginPrompt (see page 6872)

BeforeConnect (see page 6875)

149.1.5.6.6 TSQLConnection.TraceCallbackEvent Event

Provides access to the callback function that executes for every SQL command passed to or from the server.

Pascal

```
property TraceCallbackEvent: TDBXTraceEvent;
```

C++

```
__property TDBXTraceEvent TraceCallbackEvent;
```

Description

Read TraceCallbackEvent to determine whether the SQL connection is monitoring communication between the application and the SQL server to which it connects. For example, if the application uses a TSQLMonitor (see page 20285) component, that component assigns a value to TraceCallbackEvent, which it uses to monitor SQL commands. Changing the value of the callback would then interfere with the operation of the TSQLMonitor (see page 20285) component.

TraceCallbackEvent is a read-only property. To assign a callback for monitoring SQL commands, use the SetTraceEvent (see page 20239) method.

See Also

SetTraceEvent (see page 20239)

149.1.6 TSQLDataLink Class

TSQLDataLink allows a unidirectional dataset act as the detail in a master/detail relationship.

Class Hierarchy

Pascal

```
TSQLDataLink = class(TDetailDataLink);
```

C++

```
class TSQLDataLink : public TDetailDataLink;
```

File

SqlExpr

Description

SQL datasets use TSQLDataLink to coordinate with another (master) dataset when they represent the detail records for that other dataset. TSQLDataLink passes on data events to the data source that owns it. It is also used to implement the DataSource or MasterSource property of the dataset that owns it.

Do not confuse the dataset that is the value of the TSQLDataLink object's DataSet property with the dataset that uses TSQLDataLink to link to a master dataset. The DataSet property is the master dataset, while the dataset that owns the

TSQLDataLink object is the detail set.

See Also

TSQLDataSet (see page 20265)

TSQLQuery (see page 20298)

TSQLTable (see page 20337)

TMasterDataLink (see page 7339)

TCustomSQLDataSet (see page 20150)

TSQLTable (see page 20337)

149.1.6.1 TSQLDataLink Members

The following tables list the members exposed by TSQLDataLink.

Protected Methods

	Name	Description
	ActiveChanged (see page 20263)	Refreshes the parameter values of the detail dataset and re-executes it.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CheckBrowseMode (see page 20264)	Informs the dataset that owns the TSQLDataLink (see page 20260) object when the current record in the source dataset is about to change.
	DataEvent (see page 6881)	Responds to various events that occur while working with data.
	DataSetChanged (see page 6881)	Responds to changes in the dataset.
	DataSetScrolled (see page 6882)	Allows a response to scrolling the representation of the dataset within the data-aware object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EditingChanged (see page 6883)	Responds to changes in the editing state of the data source.
	FocusControl (see page 6884)	Provides the interface for a method to force focus to the data-aware component associated with the TDataLink (see page 6876).
	GetActiveRecord (see page 6884)	This is GetActiveRecord, a member of class TDataLink.
	GetBOF (see page 6884)	This is GetBOF, a member of class TDataLink.
	GetBufferCount (see page 6884)	This is GetBufferCount, a member of class TDataLink.
	GetDetailDataSet (see page 20264)	Returns the value of the DetailDataSet property.
	GetEOF (see page 6885)	This is GetEOF, a member of class TDataLink.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRecordCount (see page 6885)	This is GetRecordCount, a member of class TDataLink.
	LayoutChanged (see page 6885)	Responds to changes in the representation of the data by the data-aware object.
	MoveBy (see page 6885)	Repositions the dataset to another record relative to its active record.
	RecordChanged (see page 20264)	Refreshes the parameter values of the detail dataset and re-executes it.
	SetActiveRecord (see page 6886)	This is SetActiveRecord, a member of class TDataLink.
	SetBufferCount (see page 6887)	This is SetBufferCount, a member of class TDataLink.
	UpdateData (see page 6887)	Provides an interface for writing edits to a record in the dataset.

Protected Properties

	Name	Description
	VisualControl (see page 6894)	Indicates whether a visual control uses the datalink.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 20263)	Creates an instance of TSQLDataLink (see page 20260).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 6879)	Destroys an instance of TDataLink (see page 6876).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6882)	Attempts to drive the data source into an editing state if it is not already in one.
	ExecuteAction (see page 6883)	Invokes an action with the datasource as its target.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	UpdateAction (see page 6887)	Updates an action to reflect the current state of the TDataLink (see page 6876).
	UpdateRecord (see page 6888)	Responds to notifications that edits should be posted to the dataset.

Public Properties

	Name	Description
R	Active (see page 6888)	Specifies whether the dataset that TDataLink (see page 6876) manages is active.
	ActiveRecord (see page 6889)	Specifies the index of the current record within the internal record buffer maintained by the dataset for the Owner of the TDataLink (see page 6876) object.
R	BOF (see page 6889)	Indicates whether the first record in a dataset is active.
	BufferCount (see page 6890)	Specifies the number of records cached by the dataset for the Owner of the TDataLink (see page 6876) object.
R	DataSet (see page 6890)	Specifies the TDataSet (see page 6894) object that represents the data that this TDataLink (see page 6876) is tracking.
	DataSource (see page 6891)	Specifies the TDataSource (see page 7070) object that the Owner of the TDataLink (see page 6876) uses to connect to the dataset.
	DataSourceFixed (see page 6891)	Indicates whether the DataSource (see page 6891) property can be set.
R	DetailDataSet (see page 7114)	Indicates the detail dataset in the master/detail relationship.
R	Editing (see page 6892)	Indicates whether the DataSource (see page 6891) is in edit mode.
R	Eof (see page 6892)	Indicates whether the last record in a dataset is active.
	ReadOnly (see page 6893)	Indicates whether the dataset for this TDataLink (see page 6876) can be modified.

	RecordCount (see page 6893)	Indicates the number of records in the internal record buffer maintained by the dataset.

149.1.6.2 TSQLDataLink.Create Constructor

Creates an instance of TSQLDataLink (see page 20260).

Pascal

```
constructor Create(ADataSet: TCustomSQLDataSet);
```

C++

```
__fastcall TSQLDataLink(TCustomSQLDataSet ADataSet);
```

Description

Do not create instances of TSQLDataLink (see page 20260). It is instantiated by the SQL dataset that uses it to link to a master dataset. The ADataSet parameter is the dataset that creates and owns this SQL datalink. It becomes the value of the DetailDataSet property.

See Also

DetailDataSet

149.1.6.3 TSQLDataLink Methods

The methods of the TSQLDataLink class are listed here.

Protected Methods

	Name	Description
	ActiveChanged (see page 20263)	Refreshes the parameter values of the detail dataset and re-executes it.
	CheckBrowseMode (see page 20264)	Informs the dataset that owns the TSQLDataLink (see page 20260) object when the current record in the source dataset is about to change.
	GetDetailDataSet (see page 20264)	Returns the value of the DetailDataSet property.
	RecordChanged (see page 20264)	Refreshes the parameter values of the detail dataset and re-executes it.

149.1.6.3.1 TSQLDataLink.ActiveChanged Method

Refreshes the parameter values of the detail dataset and re-executes it.

Pascal

```
procedure ActiveChanged; override;
```

C++

```
virtual __fastcall ActiveChanged();
```

Description

SQL datasets establish master/detail relationships by supplying parameter values from the master dataset. ActiveChanged updates these parameter values on the detail dataset to reflect the current values of the master dataset and then refreshes the dataset so that it is populated with the new detail set.

See Also

Active (see page 6888)

Params (see page 20197)

DetailDataSet

149.1.6.3.2 TSQLDataLink.CheckBrowseMode Method

Informs the dataset that owns the TSQLDataLink (see page 20260) object when the current record in the source dataset is about to change.

Pascal

```
procedure CheckBrowseMode; override;
```

C++

```
virtual __fastcall CheckBrowseMode();
```

Description

Applications cannot call the protected CheckBrowseMode procedure. CheckBrowseMode is called immediately before the current record in the source dataset changes. It calls the CheckBrowseMode method of the dataset that owns the TSQLDataLink (see page 20260) object.

See Also

CheckBrowseMode (see page 6918)

149.1.6.3.3 TSQLDataLink.GetDetailDataSet Method

Returns the value of the DetailDataSet property.

Pascal

```
function GetDetailDataSet: TDataSet; override;
```

C++

```
virtual __fastcall TDataSet GetDetailDataSet();
```

Description

GetDetailDataSet is the protected read implementation of the DetailDataSet property.

See Also

DetailDataSet

149.1.6.3.4 TSQLDataLink.RecordChanged Method

Refreshes the parameter values of the detail dataset and re-executes it.

Pascal

```
procedure RecordChanged(Field: TField); override;
```

C++

```
virtual __fastcall RecordChanged(TField Field);
```

Description

Applications cannot call the protected RecordChanged method. It is called automatically when the contents of the current record change. The Field parameter indicates which field in the current record has changed. If Field is nil (Delphi) or NULL (C++), the entire record has changed.

SQL datasets establish master/detail relationships by supplying parameter values from the master dataset. RecordChanged updates these parameter values on the detail dataset to reflect the current values of the master dataset and then refreshes the dataset so that it is populated with the new detail set.

See Also

Params (see page 20197)

DetailDataSet

149.1.7 TSQLDataSet Class

TSQLDataSet represents data retrieved using dbExpress.

Class Hierarchy

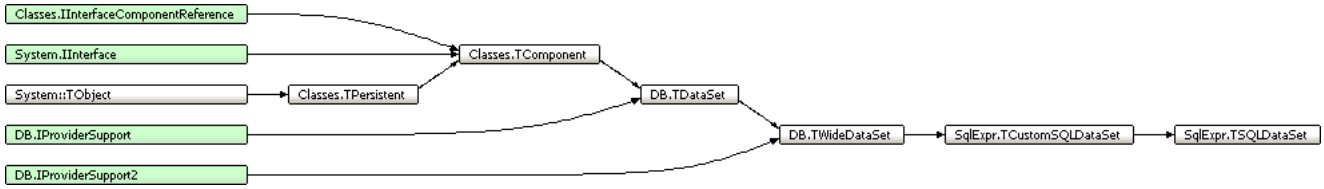

Pascal

```
TSQLDataSet = class(TCustomSQLDataSet);
```

C++

```
class TSQLDataSet : public TCustomSQLDataSet;
```

File

SqlExpr

Description

TSQLDataSet is a general-purpose unidirectional dataset for accessing database information using dbExpress. You can add a TSQLDataSet component to a form or data module at design time, or create one dynamically at runtime.

Use TSQLDataSet to

Represent the records in a database table, the result set of a SELECT query, or the result set returned by a stored procedure.

Execute a query or stored procedure that does not return a result set.

Represent metadata that describes what is available on the database server (tables, stored procedures, fields in a table, and so on).

TSQLDataSet is a unidirectional dataset. Unlike other datasets, unidirectional datasets do not buffer multiple records in memory. Because of this, you can only navigate using the First and Next methods. There is no built-in editing support: you can only edit the data in an SQL dataset by explicitly creating an SQL UPDATE command or by connecting the dataset to a client dataset using a provider. Features that require buffering multiple records, such as filters or lookup fields, are not available.

Before you can use a TSQLDataSet component, it must be connected to the database server. Therefore, the first step to take when working with TSQLDataSet is to set the SQLConnection (see page 20284) property.

Once the dataset is connected to a server, you can use the Command Text editor to set the CommandText (see page 20279) property if you are using the dataset to execute a query. To execute a stored procedure or represent a single database table, set the CommandType (see page 20280) property and then choose the table or stored procedure name from a drop-down list on the CommandText (see page 20279) property. If you want to access metadata with the SQL dataset, you must specify the metadata you want at runtime by calling the SetSchemaInfo (see page 20185) method.

DefaultRowsetSize (see page 20377) is 20 (DefaultRowsetSize (see page 20377) = 20). To use a different RowSetSize for

existing Oracle database connections, manually add the RowSetSize property (for example, RowsetSize = 200) to dbxconnections.ini (Windows) or dbxconnections.conf (Linux). To have the RowSetSize property included in new connections, manually add the RowSetSize property to dbxdrivers.ini (Windows) or dbxdrivers.conf (Linux). RowsetSize can also be modified in code (for example, SQLConnection1.Params.Values['RowsetSize'] := '200'). Currently, only Oracle databases support this feature.

See Also

TSQLQuery (see page 20298)

TSQLStoredProc (see page 20318)

TSQLTable (see page 20337)

TSQLConnection (see page 20217)

TClientDataSEt

TDataSEtProvider

Types of Datasets

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Specifying What Data to Display

Types of Unidirectional Datasets

149.1.7.1 TSQLDataSet Members

The following tables list the members exposed by TSQLDataSet.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	AddFieldDesc (see page 20163)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 20163)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
V	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
V	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
V	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
	CheckPrepareError (see page 20164)	This is CheckPrepareError, a member of class TCustomSQLDataSet.

	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
	ClearIndexDefs (see page 20164)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
	CloseCursor (see page 20164)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 20164)	This is CloseStatement, a member of class TCustomSQLDataSet.
	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
	DataEvent (see page 6924)	This is DataEvent, a member of class TDataSet.
	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 20165)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	ExecuteStatement (see page 20166)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FreeBuffers (see page 20166)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCommand (see page 20166)	This is FreeCommand, a member of class TCustomSQLDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeReader (see page 20166)	This is FreeReader, a member of class TCustomSQLDataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
	GetCanModify (see page 20167)	This is GetCanModify, a member of class TCustomSQLDataSet.

	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
V	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
V	GetDataSource (see page 20167)	This is GetDataSource, a member of class TCustomSQLDataSet.
V	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
V	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
V	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
V	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
V	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetObjectTypeNames (see page 20169)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 20169)	This is GetOutputParams, a member of class TCustomSQLDataSet.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
V	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
V	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
A	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
V	GetRecord (see page 20170)	This is GetRecord, a member of class TCustomSQLDataSet.
V	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
V	GetRecordCount (see page 20170)	This is GetRecordCount, a member of class TCustomSQLDataSet.
V	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
	GetSortFieldNames (see page 20170)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
V	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
V	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
V	InitRecord (see page 20170)	This is InitRecord, a member of class TCustomSQLDataSet.
V	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
V	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
A	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.
V	InternalClose (see page 20171)	This is InternalClose, a member of class TCustomSQLDataSet.
V	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
V	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
V	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
V	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
A	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
V	InternalHandleException (see page 20171)	This is InternalHandleException, a member of class TCustomSQLDataSet.
A	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
V	InternalInitFieldDefs (see page 20171)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
V	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
V	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
V	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
A	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
V	InternalOpen (see page 20171)	This is InternalOpen, a member of class TCustomSQLDataSet.
V	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
V	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.

	InternalRefresh (see page 20171)	This is InternalRefresh, a member of class TCustomSQLDataSet.
	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
	IsCursorOpen (see page 20172)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
	Loaded (see page 20172)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 20173)	This is LocateRecord, a member of class TCustomSQLDataSet.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	OpenCursor (see page 6976)	This is OpenCursor, a member of class TDataSet.
	OpenCursor (see page 20174)	This is OpenCursor, a member of class TCustomSQLDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	OpenSchema (see page 20174)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 20175)	This is PropertyChanged, a member of class TCustomSQLDataSet.
	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
	PSEndTransaction (see page 20175)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
	PSExecute (see page 20176)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 20176)	This is PSExecuteStatement, a member of class TCustomSQLDataSet.
	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSExecuteStatement (see page 7535)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 20176)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 6982)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 7537)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 20177)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandTextW (see page 7537)	This is PSGetCommandTextW, a member of class TWideDataSet.
	PSGetCommandType (see page 6982)	Gets the command type to be executed by PSExecute.
	PSGetCommandType (see page 20177)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 20178)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 20178)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7538)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetKeyFieldsW (see page 7538)	This is PSGetKeyFieldsW, a member of class TWideDataSet.
	PSGetKeyFieldsW (see page 20179)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetParams (see page 20179)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7538)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetQuoteCharW (see page 7539)	This is PSGetQuoteCharW, a member of class TWideDataSet.
	PSGetQuoteCharW (see page 20179)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.

	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7539)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetTableNameW (see page 7539)	This is PSGetTableNameW, a member of class TWideDataSet.
	PSGetTableNameW (see page 20179)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSGetUpdateException (see page 20180)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 20180)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 20180)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 20181)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 20181)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 20182)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetCommandText (see page 7540)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 20182)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSStartTransaction (see page 20182)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 20183)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetBufListSize (see page 20184)	This is SetBufListSize, a member of class TCustomSQLDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCommandText (see page 20184)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFCommandText (see page 20184)	This is SetFCommandText, a member of class TCustomSQLDataSet.
	SetFieldData (see page 20184)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetMaxBlobSize (see page 20184)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
V	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetParamsFromCursor (see page 20185)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
	SetSortFieldNames (see page 20186)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
V	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.
V	UpdateIndexDefs (see page 20186)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
	BlobBuffer (see page 20189)	Reserves a buffer for storing BLOB values.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
	CurrentBlobSize (see page 20191)	Indicates the size of the most recently read BLOB field.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
R	DataLink (see page 20191)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
R	InternalConnection (see page 20194)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	LastError (see page 20194)	Saves the error message for the last error returned by dbExpress.
	NativeCommand (see page 20195)	Represents the SQL command that is sent to the server.

	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
R	ParamCount (see page 20197)	Indicates the number of parameters listed by the Params (see page 20197) property.
	ProcParams (see page 20199)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
	Reserved (see page 7035)	For internal use only.
R	RowsAffected (see page 20200)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 20200)	Identifies the metadata with which to populate the dataset.

Public Events

	Name	Description
	AfterCancel (see page 7038)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 7040)	Occurs after an application closes a dataset.
	AfterDelete (see page 7041)	Occurs after an application deletes a record.
	AfterEdit (see page 7044)	Occurs after an application starts editing a record.
	AfterInsert (see page 7044)	Occurs after an application inserts a new record.
	AfterOpen (see page 7045)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 7046)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 7047)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 7047)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 7048)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 7048)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 7049)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 7049)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 7050)	Occurs before an application enters insert mode.
	BeforeOpen (see page 7051)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 7052)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 7052)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 7053)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 7053)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 7054)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 7054)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 7056)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 7056)	Occurs when an application attempts to modify or insert a record and an exception is raised.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.

	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Close (see page 6920)	Closes a dataset.
	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
	Create (see page 20278)	Creates an instance of a TSQLDataSet (see page 20265) component.
	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
	CreateBlobStream (see page 20164)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 6904)	Destroys the TDataSet (see page 6894) instance.
	Destroy (see page 20161)	Destroys the TCustomSQLDataSet (see page 20150) instance.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	ExecSQL (see page 20278)	Executes a query or stored procedure that does not return a set of records.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
	GetBlobFieldData (see page 20166)	Retrieves the current value of a BLOB field into a buffer.
	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
	GetDetailLinkFields (see page 20167)	Lists the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
	GetFieldData (see page 20168)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetKeyFieldNames (see page 20169)	Fills a list with the names of all fields in all indexes defined for the dataset.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetQuoteChar (see page 20169)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	IsSequenced (see page 20172)	Indicates whether the dataset can use record numbers to indicate the order of records.
	Last (see page 6968)	Moves to the last record in the dataset.
	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Locate (see page 20173)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	Lookup (see page 20174)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	Open (see page 6976)	Opens the dataset.

	ParamByName (see page 20175)	This is ParamByName, a member of class TCustomSQLDataSet.
V	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
V	PrepareStatement (see page 20175)	Performs a prepare on the SQL statement.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSEndTransaction (see page 7589)	Ends a transaction that was started by a call to PSStartTransaction (see page 7597).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecute (see page 7590)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSExecuteStatement (see page 7590)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 7591)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 7579)	Gets the SQL command to be executed by PSExecute (see page 7578).
	PSGetCommandText (see page 7591)	Gets the SQL command to be executed by PSExecute (see page 7590).
	PSGetCommandType (see page 7580)	Gets the command type to be executed by PSExecute (see page 7578).
	PSGetCommandType (see page 7591)	Gets the command type to be executed by PSExecute (see page 7590).
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 7592)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 7592)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7593)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetParams (see page 7593)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7594)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7594)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSGetUpdateException (see page 7594)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 7595)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 7595)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 7596)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 7596)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7585)	Assigns a specified SQL command to be executed by PSExecute (see page 7578).
	PSSetCommandText (see page 7596)	Assigns a specified SQL command to be executed by PSExecute (see page 7590).
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 7597)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).

	PSStartTransaction (see page 7597)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7589).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 7598)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetSchemaInfo (see page 20185)	Indicates whether the dataset represents metadata from the server, and if so, what kind.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.

Public Properties

	Name	Description
	Active (see page 7004)	Specifies whether or not a dataset is open.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignerData (see page 20192)	Stores custom data.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
	FieldDefs (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset

	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
	IndexDefs (see page 20193)	Contains the definitions of all indexes defined on the server for the dataset.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	Modified (see page 7029)	Indicates whether the active record is modified.
	ObjectView (see page 7031)	Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Prepared (see page 20198)	Specifies whether the command is prepared before execution.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
R	RecordCount (see page 20199)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	TransactionLevel (see page 20202)	Indicates the transaction to which this dataset belongs.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterClose (see page 20203)	Occurs after a data source connection is closed.
	AfterOpen (see page 20203)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 20203)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 20204)	Occurs after a result set is scrolled.
	BeforeClose (see page 20204)	Occurs just before a data connection is closed.
	BeforeOpen (see page 20204)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 20204)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 20204)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 20204)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 20205)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 20189) property.
	ParseInsertSql (see page 20205)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 20189) property.
	ParseSelectSql (see page 20205)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 20189) property.
	ParseUpdateSql (see page 20205)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 20189) property.

Published Properties

	Name	Description
	Active (see page 20187)	Specifies whether or not a dataset is open.
	CommandText (see page 20279)	Specifies the command the dataset executes.
	CommandType (see page 20280)	Command type of CommandText (see page 20279) property.
	DataSource (see page 20281)	Links the SQL dataset to another (master) dataset.
	GetMetadata (see page 20193)	Specifies whether the SQL dataset fetches metadata information along with the data.
	MaxBlobSize (see page 20282)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NumericMapping (see page 20195)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 20196)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.

	ParamCheck (see page 20282)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 20283)	Represents the parameters for a query or stored procedure.
	SchemaName (see page 20201)	Specifies the name of the database schema.
	SortFieldNames (see page 20284)	Indicates the sort order when CommandType (see page 20280) is ctTable.
	SQLConnection (see page 20284)	Specifies the SQL connection component that connects the dataset to a database server.
	Tag (see page 2808)	Stores an integer value as part of a component.

149.1.7.2 TSQLDataSet.Create Constructor

Creates an instance of a TSQLDataSet (see page 20265) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSQLDataSet(TComponent * AOwner);
```

Description

Call Create to instantiate an SQL dataset component at runtime. SQL datasets placed in forms or data modules at design time are created automatically.

AOwner is the component, typically the form or data module, that is responsible for freeing the dataset. It becomes the value of the Owner property.

See Also

Owner (see page 2807)

149.1.7.3 TSQLDataSet Methods

The methods of the TSQLDataSet class are listed here.

Public Methods

	Name	Description
	ExecSQL (see page 20278)	Executes a query or stored procedure that does not return a set of records.

149.1.7.3.1 TSQLDataSet.ExecSQL Method

Executes a query or stored procedure that does not return a set of records.

Pascal

```
function ExecSQL(ExecDirect: Boolean = False): Integer; override;
```

C++

```
virtual __fastcall int ExecSQL(Boolean ExecDirect = False);
```

Description

Call ExecSQL to execute an SQL command that does not return a set of records. This command is a query other than a SELECT query, such as an INSERT, UPDATE, DELETE, or CREATE TABLE query.

ExecDirect indicates that the query does not need to be prepared before it is executed. This parameter can be set to true if the query does not include any parameters. When ExecDirect is false, the query will be prepared before it is executed. Even when

ExecDirect is false, you can avoid preparing the query every time ExecSQL is called by setting the Prepared (see page 20198) property to true before calling ExecSQL the first time.

ExecSQL returns the number of rows affected by the executed command. This becomes the value of the RowsAffected (see page 20200) property.

Note: Do not use ExecSQL for commands that return data. These include any use of the ctTable command type, SELECT queries, and stored procedures that return a cursor. When the command returns data, use the Open method or set the Active (see page 20187) property to true.

See Also

CommandText (see page 20189)

RowsAffected (see page 20200)

Open (see page 6976)

Active (see page 7004)

Prepared (see page 20198)

ExecuteDirect (see page 20228)

Executing the Command

149.1.7.4 TSQLDataSet Properties

The properties of the TSQLDataSet class are listed here.

Published Properties

	Name	Description
	CommandText (see page 20279)	Specifies the command the dataset executes.
	CommandType (see page 20280)	Command type of CommandText (see page 20279) property.
	DataSource (see page 20281)	Links the SQL dataset to another (master) dataset.
	MaxBlobSize (see page 20282)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	ParamCheck (see page 20282)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 20283)	Represents the parameters for a query or stored procedure.
	SortFieldNames (see page 20284)	Indicates the sort order when CommandType (see page 20280) is ctTable.
	SQLConnection (see page 20284)	Specifies the SQL connection component that connects the dataset to a database server.

149.1.7.4.1 TSQLDataSet.CommandText Property

Specifies the command the dataset executes.

Pascal

```
property CommandText: WideString;
```

C++

```
__property BSTR CommandText;
```

Description

The value of CommandText depends on the CommandType (see page 20280) property:

When CommandType (see page 20280) is ctQuery, CommandText is an SQL statement that the dataset executes. If the statement is a SELECT statement, the SQL dataset executes the statement when it is opened. If the statement does not return a

result set, the SQL dataset executes the statement when the ExecSQL (see page 20278) method is called.

When CommandType (see page 20280) is ctStoredProc, CommandText is the name of a stored procedure. The SQL dataset executes the stored procedure when it is opened or when the ExecSQL (see page 20278) method is called, depending on whether the stored procedure returns a set of records.

When CommandType (see page 20280) is ctTable, CommandText is the name of a table on the database server. The SQL dataset automatically generates a SELECT statement to fetch all the records on all the fields in this table, sorting by the fields listed in the SortFieldNames (see page 20284) property. The generated statement is executed when the dataset is opened.

If CommandText is an SQL statement that includes parameters or the name of a stored procedure that has parameters, use the Params (see page 20283) property to supply parameter values.

This command is passed on to the TDBXCommand.Text property.

See Also

TDBXCommand (see page 10378)

TDBXCommand.CommandType

Params (see page 20283)

ExecSQL (see page 20278)

Open (see page 6976)

SortFieldNames (see page 20284)

Fetching Metadata into a Unidirectional Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Accessing Schema Information

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

149.1.7.4.2 TSQLDataSet.CommandType Property

Command type of CommandText (see page 20279) property.

Pascal

```
property CommandType: TSQLCommandType;
```

C++

```
__property TSQLCommandType CommandType;
```

Description

CommandType indicates the command type of the CommandText (see page 20279) property.

The following table lists the possible values:

CommandType	Corresponding CommandText (see page 20279)
ctQuery	An SQL statement that the dataset executes.
ctStoredProc	The name of a stored procedure.
ctTable	The name of a table on the database server. The SQL dataset automatically generates a SELECT statement to fetch all the records on all the fields in this table.

Note: Changing CommandType automatically changes the DbxCommandType property and vice versa. The additional settings supported by DbxCommandType are converted to ctQuery for CommandType.

See Also

CommandText (see page 20279)

DbxCommandType

Setting Up a Simple Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Specifying What Data to Display

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

149.1.7.4.3 TSQLDataSet.DataSource Property

Links the SQL dataset to another (master) dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;__property TDataSource DataSource;
```

Description

DataSource links this SQL dataset to another (master) dataset that supplies values for this dataset's parameters.

When the SQL dataset's CommandText (see page 20279) (or SQL) property specifies a parameterized query but the application does not provide values for the corresponding parameters in the Params (see page 20283) property, the SQL dataset fetches parameter values from the dataset identified with the specified data source. When the current record in the dataset associated with DataSource changes, the SQL dataset refreshes its data to reflect the new parameter values.

See Also

CommandText (see page 20279)

SQL (see page 20316)

Params (see page 20283)

MasterSource (see page 20357)

149.1.7.4.4 TSQLDataSet.MaxBlobSize Property

Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

MaxBlobSize limits the amount of data that can be fetched from Binary Large Object (BLOB) fields. BLOB fields can hold an indefinite amount of data. However, SQL datasets read that data into a memory buffer. By limiting the amount of data that can be fetched from a BLOB field, you limit the size of the buffer.

If MaxBlobSize is –1, the dataset imposes no maximum BLOB size. If MaxBlobSize is 0, the dataset uses the parameters of the associated SQL connection, if any, to determine the maximum BLOB size.

Note: MaxBlobSize should agree with any parameters on the associated SQL connection component that limit the size of BLOB data.

See Also

GetBlobFieldData (see page 20166)

Params (see page 20283)

SQLConnection (see page 20284)

149.1.7.4.5 TSQLDataSet.ParamCheck Property

Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

By default, the Params (see page 20283) property is updated automatically to reflect any parameters that appear in the query specified by the CommandText (see page 20279) property (or the SQL property for TSQLQuery (see page 20298)). That is, whenever the CommandText (see page 20279) property changes, the Params (see page 20283) property is regenerated to reflect the new set of parameters.

There are times when you want to override this default behavior. For example, when specifying a data definition language (DDL) statement that defines a stored procedure, the resulting SQL command may include parameters. These parameters, however, should not be used as parameters of the SQL dataset, because they are not parameters of the DDL command, but rather, they are parameters of the stored procedure that the command creates.

In order to override the default generation of the Params (see page 20283) property in response to changes in CommandText (see page 20279), set ParamCheck to false.

See Also

Params (see page 20283)

CommandText (see page 20279)

SQL (see page 20316)

149.1.7.4.6 TSQLDataSet.Params Property

Represents the parameters for a query or stored procedure.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Params is a collection of TParam objects that represent the parameters of a query or stored procedure specified by the SQL dataset. When you specify a query by setting the CommandText (see page 20279) property (or the SQL property in TSQLQuery (see page 20298)), the SQL dataset automatically parse the query and fills Params with a TParam object for every parameter in the query. Similarly, when you specify a stored procedure by setting the CommandText (see page 20279) property (or the StoredProcName property in TSQLStoredProc (see page 20318)), the SQL dataset automatically asks the server for a list of stored procedure parameters and fills Params with a TParam object for every parameter of the stored procedure.

At design-time, use the Params property editor to access the individual TParam objects. For queries, use this property editor to check the DataType property of each parameter and to assign values to the parameters. For stored procedures, you may need to do more, changing the type or number of parameters if the server does not supply complete information about stored procedure parameters.

You can also use Params to assign parameter properties in code at runtime.

Note: If you do not assign values to input parameters before opening the dataset, it fetches values from the dataset identified by the DataSource (see page 20281) property. However, this only works if the individual TParam objects have the same names as fields in the master dataset.

See Also

ParamCheck (see page 20282)

SQL (see page 20316)

StoredProcName (see page 20336)

CommandText (see page 20279)

TParam (see page 7386)

Value (see page 7433)

DataType (see page 7427)

DataSource (see page 20192)

Configuring TSQL Connection

Connecting to Databases with TDatabase

Using TQuery

Using TSimpleDataSet

149.1.7.4.7 TSQLDataSet.SortFieldNames Property

Indicates the sort order when CommandType (see page 20280) is ctTable.

Pascal

```
property SortFieldNames: WideString;
```

C++

```
__property BSTR SortFieldNames;
```

Description

Use SortFieldNames to specify the fields that should be used to order the records in the dataset. SortFieldNames is only used when CommandType (see page 20280) is ctTable.

SortFieldNames is a semicolon-delimited list of field names. These names are used to generate the ORDER BY clause of the SQL statement the dataset creates to fetch data from a server table. Records are sorted by the first named field, and within that sort order, by the second named field, and so on.

See Also

CommandType (see page 20280)

Representing the Records in a Table

Specifying the Data to Display using TSQLDataSet

149.1.7.4.8 TSQLDataSet.SQLConnection Property

Specifies the SQL connection component that connects the dataset to a database server.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

Use SQLConnection to specify an SQL connection object that connects the dataset to a database server.

At design-time, select from the list of available TSQLConnection (see page 20217) components in the Object Inspector. At runtime, set SQLConnection to reference an existing TSQLConnection (see page 20217) object.

SQLConnection should be the first property you set on the dataset. Only after the SQLConnection property is set can you use the Command Text editor to specify a query for the CommandText (see page 20279) property or populate the dataset with data by setting Active (see page 20187) to true. Similarly, SQLConnection must be set before the dataset can initialize the Params (see page 20283) property when you set CommandText (see page 20279).

See Also

Active (see page 7004)

CommandText (see page 20279)

Params (see page 20283)

Using Query-type Datasets

Using Stored Procedure-type Datasets

Using Table Type Datasets

Connecting Directly to a Database Server

Connecting to the Database Server

149.1.8 TSQLMonitor Class

TSQLMonitor intercepts messages that pass between an SQL connection component and a database server and saves them in a string list.

Class Hierarchy

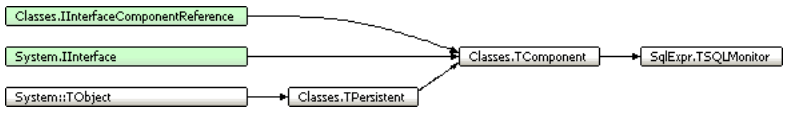

Pascal

```
TSQLMonitor = class(TComponent);
```

C++

```
class TSQLMonitor : public TComponent;
```

File

SqlExpr

Description

Use TSQLMonitor when debugging the communication between your application and a database server. Each TSQLMonitor instance logs the SQL commands from a particular SQL connection component, adding them to a string list. This allows you to see not only the commands you explicitly add to a SQL dataset or SQL connection component, but also any commands that are generated on your behalf, such as those created by a dataset provider. Use the TraceList (see page 20296) property to access the list of SQL commands that has already been logged. The TraceList (see page 20296) property is automatically updated when the connection component passes a message to the database server.

Note: TSQLMonitor uses the TraceCallbackEvent property of the SQL connection component whose messages it monitors. If you assign your own callback to that property, the SQL monitor instance will cease to function.

See Also

TSQLConnection (see page 20217)

TSTringList

TDataSetProvider (see page 18508)

TSQLConnection (see page 20217)

Debugging dbExpress Applications

149.1.8.1 TSQLMonitor Members

The following tables list the members exposed by TSQLMonitor.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	InvokeCallBack (see page 20290)	This is InvokeCallBack, a member of class TSQLMonitor.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActive (see page 20291)	This is SetActive, a member of class TSQLMonitor.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFileName (see page 20291)	This is SetFileName, a member of class TSQLMonitor.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetSQLConnection (see page 20291)	This is SetSQLConnection, a member of class TSQLMonitor.
	SetStreamedActive (see page 20291)	This is SetStreamedActive, a member of class TSQLMonitor.
	SetTraceList (see page 20292)	This is SetTraceList, a member of class TSQLMonitor.
	SwitchConnection (see page 20292)	This is SwitchConnection, a member of class TSQLMonitor.
V	Trace (see page 20292)	This is Trace, a member of class TSQLMonitor.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateTraceCallBack (see page 20292)	This is UpdateTraceCallBack, a member of class TSQLMonitor.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 20288)	Creates and initializes new TSQLMonitor (see page 20285) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 20289)	Frees the memory associated with the TSQLMonitor (see page 20285) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LoadFromFile (see page 20290)	Sets the contents of TraceList (see page 20296) to the strings contained in a file.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 20290)	Saves the contents of TraceList (see page 20296) to a file.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	MaxTraceCount (see page 20295)	Indicates the maximum number of messages that can be logged.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	TraceCount (see page 20295)	Indicates the number of messages currently logged.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnLogTrace (see page 20297)	Occurs immediately after a new message is logged to the list.
	OnTrace (see page 20297)	Occurs when a message is detected but before it is logged to the list.

Published Properties

	Name	Description
	Active (see page 20293)	Starts or stops the monitor from recording SQL commands passed to the database server.
	AutoSave (see page 20294)	Specifies whether the monitored messages are automatically saved to a file.
	FileName (see page 20294)	Specifies the default file to which monitored messages are saved.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SQLConnection (see page 20295)	Specifies the connection components whose messages are to be monitored.
	Tag (see page 2808)	Stores an integer value as part of a component.
	TraceList (see page 20296)	Lists the messages that have passed between SQLConnection (see page 20295) and the database server.

149.1.8.2 TSQLMonitor.Create Constructor

Creates and initializes new TSQLMonitor (see page 20285) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSQLMonitor(TComponent * AOwner);
```

Description

Call Create to create a TSQLMonitor (see page 20285) component at runtime. SQL monitors added to forms or data modules at design time are created automatically.

The Owner parameter specifies the component (usually the form or data module) that is responsible for freeing the SQL monitor.

See Also

Owner (see page 2807)

149.1.8.3 TSQLMonitor.Destroy Destructor

Frees the memory associated with the TSQLMonitor (see page 20285) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSQLMonitor();
```

Description

Do not call Destroy directly. Instead, call Free, which verifies that the SQL monitor instance is not nil before calling Destroy.

See Also

Free (see page 22300)

149.1.8.4 TSQLMonitor Methods

The methods of the TSQLMonitor class are listed here.

Protected Methods

	Name	Description
	InvokeCallBack (see page 20290)	This is InvokeCallBack, a member of class TSQLMonitor.
	SetActive (see page 20291)	This is SetActive, a member of class TSQLMonitor.
	SetFileName (see page 20291)	This is SetFileName, a member of class TSQLMonitor.
	SetSQLConnection (see page 20291)	This is SetSQLConnection, a member of class TSQLMonitor.
	SetStreamedActive (see page 20291)	This is SetStreamedActive, a member of class TSQLMonitor.
	SetTraceList (see page 20292)	This is SetTraceList, a member of class TSQLMonitor.
	SwitchConnection (see page 20292)	This is SwitchConnection, a member of class TSQLMonitor.
V	Trace (see page 20292)	This is Trace, a member of class TSQLMonitor.
	UpdateTraceCallBack (see page 20292)	This is UpdateTraceCallBack, a member of class TSQLMonitor.

Public Methods

	Name	Description
	LoadFromFile (see page 20290)	Sets the contents of TraceList (see page 20296) to the strings contained in a file.
	SaveToFile (see page 20290)	Saves the contents of TraceList (see page 20296) to a file.

149.1.8.4.1 TSQLMonitor.InvokeCallBack Method

Pascal

```
function InvokeCallBack(TraceInfo: TDBXTraceInfo): CBRType;
```

C++

```
__fastcall CBRType InvokeCallBack(TDBXTraceInfo TraceInfo);
```

Description

This is InvokeCallBack, a member of class TSQLMonitor.

149.1.8.4.2 TSQLMonitor.LoadFromFile Method

Sets the contents of TraceList (see page 20296) to the strings contained in a file.

Pascal

```
procedure LoadFromFile(AFileName: string);
```

C++

```
__fastcall LoadFromFile(AnsiString AFileName);
```

Description

Call LoadFromFile to initialize the message log with the contents of a file. This method can be used to start the log with standard header information you have saved in a file, or to pre-load the list with a log you saved previously using the SaveToFile (see page 20290) method.

AFileName is the file from which to load the contents of the message log. If AFileName is an empty string, LoadFromFile uses the value of the FileName (see page 20294) property.

Note: Any information currently in TraceList (see page 20296) is lost when you call LoadFromFile.

See Also

TraceList (see page 20296)

SaveToFile (see page 20290)

FileName (see page 20294)

149.1.8.4.3 TSQLMonitor.SaveToFile Method

Saves the contents of TraceList (see page 20296) to a file.

Pascal

```
procedure SaveToFile(AFileName: string);
```

C++

```
__fastcall SaveToFile(AnsiString AFileName);
```

Description

Call SaveToFile to save all of the currently logged messages to a file.

AFileName specifies the file to which the strings in TraceList (see page 20296) are saved. If AFileName is an empty string, SaveToFile uses the value of the FileName (see page 20294) property.

Note: If the AutoSave (see page 20294) property is true, SaveToFile is called automatically whenever a message is added to

TraceList (see page 20296).

See Also

TraceList (see page 20296)

LoadFromFile (see page 20290)

FileName (see page 20294)

AutoSave (see page 20294)

149.1.8.4.4 TSQLMonitor.SetActive Method

Pascal

```
procedure SetActive(Value: Boolean);
```

C++

```
__fastcall SetActive(Boolean Value);
```

Description

This is SetActive, a member of class TSQLMonitor.

149.1.8.4.5 TSQLMonitor.SetFileName Method

Pascal

```
procedure SetFileName(const Value: String);
```

C++

```
__fastcall SetFileName(const AnsiString Value);
```

Description

This is SetFileName, a member of class TSQLMonitor.

149.1.8.4.6 TSQLMonitor.SetSQLConnection Method

Pascal

```
procedure SetSQLConnection(Value: TSQLConnection);
```

C++

```
__fastcall SetSQLConnection(TSQLConnection Value);
```

Description

This is SetSQLConnection, a member of class TSQLMonitor.

149.1.8.4.7 TSQLMonitor.SetStreamedActive Method

Pascal

```
procedure SetStreamedActive;
```

C++

```
__fastcall SetStreamedActive();
```

Description

This is SetStreamedActive, a member of class TSQLMonitor.

149.1.8.4.8 TSQLMonitor.SetTraceList Method

Pascal

```
procedure SetTraceList(Value: TWideStrings);
```

C++

```
__fastcall SetTraceList(TWideStrings Value);
```

Description

This is SetTraceList, a member of class TSQLMonitor.

149.1.8.4.9 TSQLMonitor.SwitchConnection Method

Pascal

```
procedure SwitchConnection(const Value: TSQLConnection);
```

C++

```
__fastcall SwitchConnection(const TSQLConnection Value);
```

Description

This is SwitchConnection, a member of class TSQLMonitor.

149.1.8.4.10 TSQLMonitor.Trace Method

Pascal

```
procedure Trace(TraceInfo: TDBXTraceInfo; LogTrace: Boolean); virtual;
```

C++

```
virtual __fastcall Trace(TDBXTraceInfo TraceInfo, Boolean LogTrace);
```

Description

This is Trace, a member of class TSQLMonitor.

149.1.8.4.11 TSQLMonitor.UpdateTraceCallBack Method

Pascal

```
procedure UpdateTraceCallBack;
```

C++

```
__fastcall UpdateTraceCallBack();
```

Description

This is UpdateTraceCallBack, a member of class TSQLMonitor.

149.1.8.5 TSQLMonitor Properties

The properties of the TSQLMonitor class are listed here.

Public Properties

	Name	Description
	MaxTraceCount (see page 20295)	Indicates the maximum number of messages that can be logged.
R	TraceCount (see page 20295)	Indicates the number of messages currently logged.

Published Properties

	Name	Description
	Active (see page 20293)	Starts or stops the monitor from recording SQL commands passed to the database server.
	AutoSave (see page 20294)	Specifies whether the monitored messages are automatically saved to a file.
	FileName (see page 20294)	Specifies the default file to which monitored messages are saved.
	SQLConnection (see page 20295)	Specifies the connection components whose messages are to be monitored.
	TraceList (see page 20296)	Lists the messages that have passed between SQLConnection (see page 20295) and the database server.

149.1.8.5.1 TSQLMonitor.Active Property

Starts or stops the monitor from recording SQL commands passed to the database server.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Set Active to true to start monitoring the communication between SQLConnection (see page 20295) and its database server. Set Active to false to stop the monitoring.

When you set Active to true, TSQLMonitor (see page 20285) sets the TraceCallbackEvent property of the associated SQL connection component. This causes all commands sent to the database server that match the criteria established by TraceFlags to be added to the monitor's string list.

When you set Active to false, TSQLMonitor (see page 20285) clears the TraceCallbackEvent property of the associated SQL connection component. No subsequent commands are added to TraceList (see page 20296), but the list is not cleared. That is, any commands currently in the list remain there until you explicitly clear the list.

See Also

TraceCallbackEvent (see page 20260)

SQLConnection (see page 20295)

TraceList (see page 20296)

TraceFlags

Debugging dbExpress Applications

Fetching the Data using TSQLDataSet

Using TSimpleDataSet

Using TSQLQuery

Using TSQLStoredProc

149.1.8.5.2 TSQLMonitor.AutoSave Property

Specifies whether the monitored messages are automatically saved to a file.

Pascal

```
property AutoSave: Boolean;
```

C++

```
__property Boolean AutoSave;
```

Description

Set AutoSave to true to automatically save all messages that match the criteria established by TraceFlags to the file specified by the FileName (see page 20294) property. The FileName (see page 20294) property must be set to a valid file name.

When AutoSave is false, messages that match the criteria specified by TraceFlags are automatically added to TraceList (see page 20296), but are not saved to a file.

See Also

FileName (see page 20294)

TraceList (see page 20296)

TraceFlags

SaveToFile (see page 20290)

Debugging dbExpress Applications

149.1.8.5.3 TSQLMonitor.FileName Property

Specifies the default file to which monitored messages are saved.

Pascal

```
property FileName: string;
```

C++

```
__property AnsiString FileName;
```

Description

Set FileName to the name of a file to which the messages listed in TraceList (see page 20296) are saved. FileName can be a fully-qualified path name.

Once FileName is specified, setting AutoSave (see page 20294) to true causes all monitored messages to be automatically saved to the specified file. Even if AutoSave (see page 20294) is false, FileName specifies the default file name used by the LoadFromFile (see page 20290) and SaveToFile (see page 20290) methods when they are called with an empty string for the AFileName parameter.

See Also

AutoSave (see page 20294)

SaveToFile (see page 20290)

TraceList (see page 20296)

Debugging dbExpress Applications

149.1.8.5.4 TSQLMonitor.MaxTraceCount Property

Indicates the maximum number of messages that can be logged.

Pascal

```
property MaxTraceCount: Integer;
```

C++

```
__property int MaxTraceCount;
```

Description

Set MaxTraceCount to limit the number of messages that can be added to the TraceList (see page 20296) property. When MaxTraceCount is 0 or greater, the SQL monitor logs every message until the limit specified by MaxTraceCount. Once that limit has been reached, the SQL monitor discards messages until the number of messages in TraceList (see page 20296) has been reduced or the value of MaxTraceCount is changed.

If MaxTraceCount is –1, there is no limit on the number of messages that can be added to TraceList (see page 20296).

If MaxTraceCount is 0, the SQL monitor does not log any messages.

Note: Setting MaxTraceCount affects whether messages can be added to TraceList (see page 20296), but does not affect messages that have already been logged. Thus, if you set MaxTraceCount to a value less than the number of strings currently in TraceList (see page 20296), the number of logged messages will exceed MaxTraceCount.

See Also

TraceList (see page 20296)

TraceCount (see page 20295)

149.1.8.5.5 TSQLMonitor.SQLConnection Property

Specifies the connection components whose messages are to be monitored.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

Set SQLConnection to identify the TSQLConnection (see page 20217) component whose communication with a database server you want to monitor.

Note: Changing the value of SQLConnection automatically sets the Active (see page 20293) property to false.

See Also

Active (see page 20293)

Debugging dbExpress Applications

149.1.8.5.6 TSQLMonitor.TraceCount Property

Indicates the number of messages currently logged.

Pascal

```
property TraceCount: Integer;
```

C++

```
__property int TraceCount;
```

Description

TraceCount indicates the number of messages listed by the TraceList (see page 20296) property.

See Also

TraceList (see page 20296)

MaxTraceCount (see page 20295)

149.1.8.5.7 TSQLMonitor.TraceList Property

Lists the messages that have passed between SQLConnection (see page 20295) and the database server.

Pascal

```
property TraceList: TWideStrings;
```

C++

```
__property TWideStrings TraceList;
```

Description

Use the TraceList property to access the list of SQL commands that has already been logged. While the SQL monitor is active, the TraceList property is automatically updated when the connection component passes a message to the database server.

Only messages that meet the criteria specified by the TraceFlags property are added to TraceList.

TraceList is a TStrings descendant. As such, you can use its methods to

Clear the list.

Access the individual commands in the list.

Add your own comments to the list. For example, you may want to indicate when your application initiates certain actions so that you can see clearly what SQL messages correspond to what actions.

See Also

SQLConnection (see page 20295)

Active (see page 20293)

TraceFlags

SaveToFile (see page 20290)

LoadFromFile (see page 20290)

TraceCount (see page 20295)

Debugging dbExpress Applications

149.1.8.6 TSQLMonitor Events

The events of the TSQLMonitor class are listed here.

Published Events

	Name	Description
	OnLogTrace (see page 20297)	Occurs immediately after a new message is logged to the list.
	OnTrace (see page 20297)	Occurs when a message is detected but before it is logged to the list.

149.1.8.6.1 TSQLMonitor.OnLogTrace Event

Occurs immediately after a new message is logged to the list.

Pascal

```
property OnLogTrace: TTraceLogEvent;
```

C++

```
__property TTraceLogEvent OnLogTrace;
```

Description

Write an OnLogTrace event handler to respond immediately after a new string is added to the TraceList (see page 20296) property. For example, you can use this event to periodically save the contents of the list to a file and clear the list so that it does not get prohibitively long. OnLogTrace occurs after the message is added to TraceList (see page 20296) but before the list is saved to a file when AutoSave (see page 20294) is true.

Note: OnLogTrace occurs only after logging messages that meet the criteria specified by TraceFlags.

See Also

TraceList (see page 20296)

OnTrace (see page 20297)

TraceFlags

AutoSave (see page 20294)

Debugging dbExpress Applications

149.1.8.6.2 TSQLMonitor.OnTrace Event

Occurs when a message is detected but before it is logged to the list.

Pascal

```
property OnTrace: TTraceEvent;
```

C++

```
__property TTraceEvent OnTrace;
```

Description

Write an OnTrace event handler to screen or edit messages before they are added to TraceList (see page 20296). OnTrace occurs for every message that passes between the dbExpress driver and the database server, not just those that meet the criteria specified by TraceFlags.

See Also

TraceList (see page 20296)

OnLogTrace (see page 20297)

TraceFlags

149.1.9 TSQLQuery Class

TSQLQuery represents a query that is executed using dbExpress.

Class Hierarchy

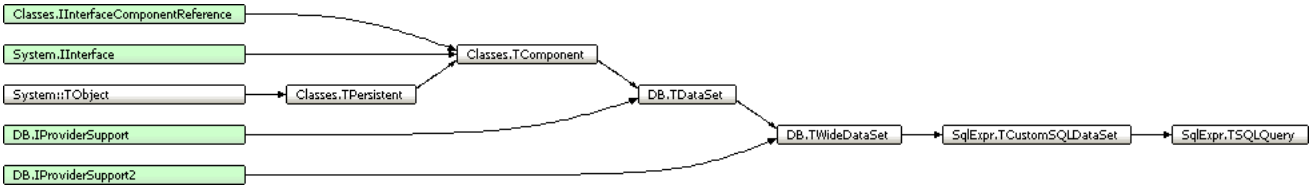

Pascal

```
TSQLQuery = class(TCustomSQLDataSet);
```

C++

```
class TSQLQuery : public TCustomSQLDataSet;
```

File

SqlExpr

Description

Use TSQLQuery to execute an SQL (see page 20316) command on a database server that is accessed via TSQLConnection (see page 20217). TSQLQuery can represent the results of a SELECT statement or perform actions on the database server using statements such as INSERT, DELETE, UPDATE, ALTER TABLE, and so on. You can add a TSQLQuery component to a form or data module at design time, or create one dynamically at runtime.

TSQLQuery is a unidirectional dataset. Unlike other datasets, unidirectional datasets do not buffer multiple records in memory. Because of this, you can only navigate using the First and Next methods. There is no built-in editing support: you can only edit the data in an SQL (see page 20316) query by explicitly creating an SQL (see page 20316) UPDATE command or by connecting the dataset to a client dataset using a provider. Features that require buffering multiple records, such as filters or lookup fields, are not available.

Before you can use a TSQLQuery component, it must be connected to the database server. Therefore, the first step to take when working with TSQLQuery is to set the SQLConnection (see page 20317) property. Once the query is connected to a server, use the SQL (see page 20316) property to specify the command your query executes.

See Also

TSQLDataSet (see page 20265)

TSQLStoredProc (see page 20318)

TSQLTable (see page 20337)

TSQLConnection (see page 20217)

TClientDataSEt

TDataSEtProvider

Types of Datasets

Specifying the Command to Execute

Types of Unidirectional Datasets

149.1.9.1 TSQLQuery Members

The following tables list the members exposed by TSQLQuery.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	AddFieldDesc (see page 20163)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 20163)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
V	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
V	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
V	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
	CheckPrepareError (see page 20164)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
V	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
V	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
	ClearIndexDefs (see page 20164)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
V	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
V	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
V	CloseCursor (see page 20164)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 20164)	This is CloseStatement, a member of class TCustomSQLDataSet.
V	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
V	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
V	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
V	DataEvent (see page 6924)	This is DataEvent, a member of class TDataSet.
V	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 20165)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
V	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
V	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
V	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
V	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
V	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
V	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
V	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.

	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	ExecuteStatement (see page 20166)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FreeBuffers (see page 20166)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCommand (see page 20166)	This is FreeCommand, a member of class TCustomSQLDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeReader (see page 20166)	This is FreeReader, a member of class TCustomSQLDataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
	GetCanModify (see page 20167)	This is GetCanModify, a member of class TCustomSQLDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
	GetDataSource (see page 20167)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetObjectTypeNames (see page 20169)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 20169)	This is GetOutputParams, a member of class TCustomSQLDataSet.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
	GetRecord (see page 20170)	This is GetRecord, a member of class TCustomSQLDataSet.
	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
	GetRecordCount (see page 20170)	This is GetRecordCount, a member of class TCustomSQLDataSet.

V	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
	GetSortFieldNames (see page 20170)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
V	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
V	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
V	InitRecord (see page 20170)	This is InitRecord, a member of class TCustomSQLDataSet.
V	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
V	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
A	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.
V	InternalClose (see page 20171)	This is InternalClose, a member of class TCustomSQLDataSet.
V	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
V	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
V	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
V	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
A	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
V	InternalHandleException (see page 20171)	This is InternalHandleException, a member of class TCustomSQLDataSet.
A	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
V	InternalInitFieldDefs (see page 20171)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
V	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
V	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
V	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
A	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
V	InternalOpen (see page 20171)	This is InternalOpen, a member of class TCustomSQLDataSet.
V	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
V	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
V	InternalRefresh (see page 20171)	This is InternalRefresh, a member of class TCustomSQLDataSet.
V	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
A	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
V	IsCursorOpen (see page 20172)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 20172)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 20173)	This is LocateRecord, a member of class TCustomSQLDataSet.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	OpenCursor (see page 6976)	This is OpenCursor, a member of class TDataSet.
V	OpenCursor (see page 20174)	This is OpenCursor, a member of class TCustomSQLDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
V	OpenSchema (see page 20174)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 20175)	This is PropertyChanged, a member of class TCustomSQLDataSet.
V	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSEndTransaction (see page 20175)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
V	PSExecute (see page 20176)	Executes the SQL command associated with the dataset.
V	PSExecuteStatement (see page 20176)	This is PSExecuteStatement, a member of class TCustomSQLDataSet.
V	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
V	PSExecuteStatement (see page 7535)	This is the overview for the PSExecuteStatement method overload.
V	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.

	PSGetAttributes (see page 20176)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 6982)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 7537)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 20177)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandTextW (see page 7537)	This is PSGetCommandTextW, a member of class TWideDataSet.
	PSGetCommandType (see page 6982)	Gets the command type to be executed by PSExecute.
	PSGetCommandType (see page 20177)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 20178)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 20178)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7538)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetKeyFieldsW (see page 7538)	This is PSGetKeyFieldsW, a member of class TWideDataSet.
	PSGetKeyFieldsW (see page 20179)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetParams (see page 20179)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7538)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetQuoteCharW (see page 7539)	This is PSGetQuoteCharW, a member of class TWideDataSet.
	PSGetQuoteCharW (see page 20179)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7539)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetTableNameW (see page 7539)	This is PSGetTableNameW, a member of class TWideDataSet.
	PSGetTableNameW (see page 20179)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSGetUpdateException (see page 20180)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 20180)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 20180)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 20181)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 20181)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 20182)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetCommandText (see page 7540)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 20182)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSStartTransaction (see page 20182)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 20183)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.

	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetBufListSize (see page 20184)	This is SetBufListSize, a member of class TCustomSQLDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCommandText (see page 20184)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFCommandText (see page 20184)	This is SetFCommandText, a member of class TCustomSQLDataSet.
	SetFieldData (see page 20184)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetMaxBlobSize (see page 20184)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetParamsFromCursor (see page 20185)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
	SetSortFieldNames (see page 20186)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.
	UpdateIndexDefs (see page 20186)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.

	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
	BlobBuffer (see page 20189)	Reserves a buffer for storing BLOB values.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
	CommandType (see page 20190)	Indicates the meaning of the CommandText (see page 20189) property.
	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
	CurrentBlobSize (see page 20191)	Indicates the size of the most recently read BLOB field.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
R	DataLink (see page 20191)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
R	InternalConnection (see page 20194)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	LastError (see page 20194)	Saves the error message for the last error returned by dbExpress.
	NativeCommand (see page 20195)	Represents the SQL command that is sent to the server.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
R	ParamCount (see page 20197)	Indicates the number of parameters listed by the Params (see page 20197) property.
	ProcParams (see page 20199)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
	Reserved (see page 7035)	For internal use only.
	SchemaInfo (see page 20200)	Identifies the metadata with which to populate the dataset.
	SortFieldNames (see page 20201)	Indicates the sort order when CommandType (see page 20190) is ctTable.

Public Events

	Name	Description
	AfterCancel (see page 7038)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 7040)	Occurs after an application closes a dataset.
	AfterDelete (see page 7041)	Occurs after an application deletes a record.
	AfterEdit (see page 7044)	Occurs after an application starts editing a record.
	AfterInsert (see page 7044)	Occurs after an application inserts a new record.
	AfterOpen (see page 7045)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 7046)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 7047)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 7047)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 7048)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 7048)	Occurs immediately before the dataset closes.

	BeforeDelete (see page 7049)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 7049)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 7050)	Occurs before an application enters insert mode.
	BeforeOpen (see page 7051)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 7052)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 7052)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 7053)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 7053)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 7054)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 7054)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 7056)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 7056)	Occurs when an application attempts to modify or insert a record and an exception is raised.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
V	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Close (see page 6920)	Closes a dataset.
V	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
V	Create (see page 20310)	Creates an instance of a TSQLQuery (see page 20298) component.
V	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
V	CreateBlobStream (see page 20164)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
V	Destroy (see page 20311)	Destroys the TSQLQuery (see page 20298) instance.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
V	ExecSQL (see page 20311)	Executes a query that does not return a set of records.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
V	GetBlobFieldData (see page 20166)	Retrieves the current value of a BLOB field into a buffer.
V	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
V	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
V	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
V	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
V	GetDetailLinkFields (see page 20167)	Lists the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
V	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
V	GetFieldData (see page 20168)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
V	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetKeyFieldNames (see page 20169)	Fills a list with the names of all fields in all indexes defined for the dataset.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetQuoteChar (see page 20169)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
V	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
V	IsSequenced (see page 20172)	Indicates whether the dataset can use record numbers to indicate the order of records.
	Last (see page 6968)	Moves to the last record in the dataset.
V	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Locate (see page 20173)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
V	Lookup (see page 20174)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	Open (see page 6976)	Opens the dataset.
	ParamByName (see page 20175)	This is ParamByName, a member of class TCustomSQLDataSet.
V	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
V	PrepareStatement (see page 20312)	Prepares the query for execution.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSEndTransaction (see page 7589)	Ends a transaction that was started by a call to PSStartTransaction (see page 7597).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecute (see page 7590)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSExecuteStatement (see page 7590)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 7591)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 7579)	Gets the SQL command to be executed by PSExecute (see page 7578).
	PSGetCommandText (see page 7591)	Gets the SQL command to be executed by PSExecute (see page 7590).
	PSGetCommandType (see page 7580)	Gets the command type to be executed by PSExecute (see page 7578).
	PSGetCommandType (see page 7591)	Gets the command type to be executed by PSExecute (see page 7590).
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 7592)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 7592)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7593)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.

	PSGetParams (see page 7593)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7594)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7594)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSGetUpdateException (see page 7594)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 7595)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 7595)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 7596)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 7596)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7585)	Assigns a specified SQL command to be executed by PSExecute (see page 7578).
	PSSetCommandText (see page 7596)	Assigns a specified SQL command to be executed by PSExecute (see page 7590).
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 7597)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).
	PSStartTransaction (see page 7597)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7589).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 7598)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetSchemaInfo (see page 20185)	Indicates whether the dataset represents metadata from the server, and if so, what kind.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.

Public Properties

	Name	Description
	Active (see page 7004)	Specifies whether or not a dataset is open.
R	AggFields (see page 7006)	Contains any aggregate fields.

	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
	CommandText (see page 20189)	Specifies the command the dataset executes.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignerData (see page 20192)	Stores custom data.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
	FieldDefs (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
	IndexDefs (see page 20193)	Contains the definitions of all indexes defined on the server for the dataset.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	Modified (see page 7029)	Indicates whether the active record is modified.
	ObjectView (see page 7031)	Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Prepared (see page 20198)	Specifies whether the command is prepared before execution.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
R	RecordCount (see page 20199)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
	RowsAffected (see page 20316)	Indicates the number of rows operated on by the latest query execution.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
R	Text (see page 20318)	Indicates the SQL (see page 20316) statement as a single string.
	TransactionLevel (see page 20202)	Indicates the transaction to which this dataset belongs.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterClose (see page 20203)	Occurs after a data source connection is closed.

	AfterOpen (see page 20203)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 20203)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 20204)	Occurs after a result set is scrolled.
	BeforeClose (see page 20204)	Occurs just before a data connection is closed.
	BeforeOpen (see page 20204)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 20204)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 20204)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 20204)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 20205)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 20189) property.
	ParseInsertSql (see page 20205)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 20189) property.
	ParseSelectSql (see page 20205)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 20189) property.
	ParseUpdateSql (see page 20205)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 20189) property.

Published Properties

	Name	Description
	Active (see page 20187)	Specifies whether or not a dataset is open.
	DataSource (see page 20313)	Links the SQL dataset to another (master) dataset.
	GetMetadata (see page 20313)	Specifies whether the SQL dataset fetches metadata information along with the data.
	MaxBlobSize (see page 20314)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NumericMapping (see page 20195)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 20196)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	ParamCheck (see page 20314)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 20315)	Represents the parameters for a query or stored procedure.
	SchemaName (see page 20201)	Specifies the name of the database schema.
	SQL (see page 20316)	Specifies the SQL statement to execute on the database server.
	SQLConnection (see page 20317)	Specifies the SQL connection component that connects the dataset to a database server.
	Tag (see page 2808)	Stores an integer value as part of a component.

149.1.9.2 TSQLQuery.Create Constructor

Creates an instance of a TSQLQuery (see page 20298) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSQLQuery(TComponent * AOwner);
```

Description

Call Create to instantiate an SQL (see page 20316) query component at runtime. SQL (see page 20316) queries placed in forms or data modules at design time are created automatically.

See Also

Create (see page 6904)

149.1.9.3 TSQLQuery.Destroy Destructor

Destroys the TSQLQuery (see page 20298) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSQLQuery();
```

Description

Do not call Destroy directly in an application. Usually destruction of datasets is handled automatically. If an application creates its own instances of a dataset, however, and does not assign an Owner that is responsible for freeing the dataset, then the application should call Free, which verifies that the dataset reference is not nil before calling Destroy.

See Also

Owner (see page 2807)

Create (see page 20310)

Free (see page 22300)

149.1.9.4 TSQLQuery Methods

The methods of the TSQLQuery class are listed here.

Public Methods

	Name	Description
	ExecSQL (see page 20311)	Executes a query that does not return a set of records.
	PrepareStatement (see page 20312)	Prepares the query for execution.

149.1.9.4.1 TSQLQuery.ExecSQL Method

Executes a query that does not return a set of records.

Pascal

```
function ExecSQL(ExecDirect: Boolean = False): Integer; override;
```

C++

```
virtual __fastcall int ExecSQL(Boolean ExecDirect = False);
```

Description

Call ExecSQL to execute an SQL (see page 20316) command that does not return a set of records. This command is a query other than a SELECT query, such as an INSERT, UPDATE, DELETE, or CREATE TABLE query.

ExecDirect indicates that the query does not need to be prepared before it is executed. This parameter can be set to true if the query does not include any parameters. When ExecDirect is false, the query will be prepared before it is executed. Even when ExecDirect is false, you can avoid preparing the query every time ExecSQL is called by setting the Prepared (see page 20198) property to true before calling ExecSQL the first time.

ExecSQL returns the number of rows affected by the executed command. This becomes the value of the RowsAffected (see page 20316) property.

Note: Do not use ExecSQL for commands such as SELECT queries that return data. When the command returns data, use the Open method or set the Active (see page 20187) property to true.

See Also

SQL (see page 20316)

RowsAffected (see page 20200)

Open (see page 6976)

Active (see page 7004)

Prepared (see page 20198)

ExecuteDirect (see page 20228)

149.1.9.4.2 TSQLQuery.PrepareStatement Method

Prepares the query for execution.

Pascal

```
procedure PrepareStatement; override;
```

C++

```
virtual __fastcall PrepareStatement();
```

Description

PrepareStatement performs most of the necessary tasks for preparing a query to be executed on the server. It ensures the dbExpress and the database server have allocated any resources they need for executing the query and binding its parameters. Any named parameters are temporarily converted to unnamed parameters because dbExpress does not support named parameters.

Do not call PrepareStatement when preparing a query for execution. Instead, set the Prepared (see page 20198) property to true. When you use the Prepared (see page 20198) property instead, the query keeps track of the fact that it has already been prepared and need not repeat the process every time it executes the query. In addition, when you set the Prepared (see page 20198) property, the query initializes internal properties that keep track of the number of rows affected by the query.

See Also

Prepared (see page 20198)

149.1.9.5 TSQLQuery Properties

The properties of the TSQLQuery class are listed here.

Public Properties

	Name	Description
	RowsAffected (see page 20316)	Indicates the number of rows operated on by the latest query execution.
R	Text (see page 20318)	Indicates the SQL (see page 20316) statement as a single string.

Published Properties

	Name	Description
	DataSource (see page 20313)	Links the SQL dataset to another (master) dataset.
	GetMetadata (see page 20313)	Specifies whether the SQL dataset fetches metadata information along with the data.
	MaxBlobSize (see page 20314)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

	ParamCheck (see page 20314)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 20315)	Represents the parameters for a query or stored procedure.
	SQL (see page 20316)	Specifies the SQL statement to execute on the database server.
	SQLConnection (see page 20317)	Specifies the SQL connection component that connects the dataset to a database server.

149.1.9.5.1 TSQLQuery.DataSource Property

Links the SQL dataset to another (master) dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;__property TDataSource DataSource;
```

Description

DataSource links this SQL dataset to another (master) dataset that supplies values for this dataset's parameters.

When the SQL dataset's CommandText (or SQL) property specifies a parameterized query but the application does not provide values for the corresponding parameters in the Params property, the SQL dataset fetches parameter values from the dataset identified with the specified data source. When the current record in the dataset associated with DataSource changes, the SQL dataset refreshes its data to reflect the new parameter values.

See Also

CommandText (see page 20189)

SQL (see page 20316)

Params (see page 20197)

MasterSource (see page 20357)

149.1.9.5.2 TSQLQuery.GetMetadata Property

Specifies whether the SQL dataset fetches metadata information along with the data.

Pascal

```
property GetMetadata: Boolean;
```

C++

```
__property Boolean GetMetadata;
```

Description

Use GetMetadata to turn on or off the fetching of metadata on a database object. Setting GetMetadata to False will improve performance because no additional SQL will execute in order to get index information. However, setting GetMetadata to False might slow any attempts to update data using a client dataset that is linked via a dataset provider because the resolver might not generate optimized SQL to perform the update.

GetMetadata must be True to perform any operations that require an index. These operations include setting up master/detail relationships, updating data from a client dataset when the provider's UpdateMode is not upWhereAll, and delayed fetching of BLOB data.

Set GetMetadata to False if you are fetching a dataset for read-only purposes.

See Also

TClientDataset

TDataSetProvider (see page 18508)

Fetching the Data using TSQLDataSet

149.1.9.5.3 TSQLQuery.MaxBlobSize Property

Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

MaxBlobSize limits the amount of data that can be fetched from Binary Large Object (BLOB) fields. BLOB fields can hold an indefinite amount of data. However, SQL datasets read that data into a memory buffer. By limiting the amount of data that can be fetched from a BLOB field, you limit the size of the buffer.

If MaxBlobSize is –1, the dataset imposes no maximum BLOB size. If MaxBlobSize is 0, the dataset uses the parameters of the associated SQL connection, if any, to determine the maximum BLOB size.

Note: MaxBlobSize should agree with any parameters on the associated SQL connection component that limit the size of BLOB data.

See Also

GetBlobFieldData (see page 20166)

Params (see page 20253)

SQLConnection (see page 20201)

149.1.9.5.4 TSQLQuery.ParamCheck Property

Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

By default, the Params property is updated automatically to reflect any parameters that appear in the query specified by the CommandText property (or the SQL property for TSQLQuery) at runtime. That is, whenever the CommandText property changes, the Params property is regenerated to reflect the new set of parameters.

There are times when you want to override this default behavior. For example, when specifying a data definition language (DDL) statement that defines a stored procedure, the resulting SQL command may include parameters. These parameters, however, should not be used as parameters of the SQL dataset, because they are not parameters of the DDL command, but rather, they are parameters of the stored procedure that the command creates.

In order to override the default generation of the Params property in response to changes in CommandText, set ParamCheck to

false.

See Also

Params (see page 20197)

CommandText (see page 20189)

SQL (see page 20316)

149.1.9.5.5 TSQLQuery.Params Property

Represents the parameters for a query or stored procedure.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Params is a collection of TParam objects that represent the parameters of a query or stored procedure specified by the SQL dataset. When you specify a query by setting the CommandText property (or the SQL property in TSQLQuery), the SQL dataset automatically parse the query and fills Params with a TParam object for every parameter in the query. Similarly, when you specify a stored procedure by setting the CommandText property (or the StoredProcName property in TSQLStoredProc), the SQL dataset automatically asks the server for a list of stored procedure parameters and fills Params with a TParam object for every parameter of the stored procedure.

At design-time, use the Params property editor to access the individual TParam objects. For queries, use this property editor to check the DataType property of each parameter and to assign values to the parameters. For stored procedures, you may need to do more, changing the type or number of parameters if the server does not supply complete information about stored procedure parameters.

You can also use Params to assign parameter properties in code at runtime.

Note: If you do not assign values to input parameters before opening the dataset, it fetches values from the dataset identified by the DataSource property. However, this only works if the individual TParam objects have the same names as fields in the master dataset.

See Also

ParamCheck (see page 20196)

SQL (see page 20316)

StoredProcName (see page 20336)

CommandText (see page 20189)

TParam (see page 7386)

Value (see page 7433)

DataType (see page 7427)

DataSource (see page 20192)

Configuring TSQL Connection

Connecting to Databases with TDatabase

Using TQuery

Using TSimpleDataSet

149.1.9.5.6 TSQLQuery.RowsAffected Property

Indicates the number of rows operated on by the latest query execution.

Pascal

```
property RowsAffected: Integer;
```

C++

```
__property int RowsAffected;
```

Description

When the SQL dataset represents a query that does not return a result set, you can't use the RecordCount property to ascertain the number of records affected by the query. In such cases, read RowsAffected to determine how many rows were updated or deleted by the query when it executed. If no rows were updated or deleted, RowsAffected is zero. RowsAffected is –1 if the SQL statement could not be executed due to an error condition. (This latter situation typically follows the throwing of an exception.)

See Also

ExecSQL (see page 20278)

RecordCount (see page 20199)

ExecProc (see page 20331)

149.1.9.5.7 TSQLQuery.SQL Property

Specifies the SQL statement to execute on the database server.

Pascal

```
property SQL: TWideStrings;
```

C++

```
__property TWideStrings SQL;
```

Description

Use the properties and methods of SQL to specify the query that should be executed on the database server. If SQL is a SELECT statement, the query executes the statement when you call Open or set the Active (see page 20187) property to true. If the statement does not return a result set, call the ExecSQL (see page 20311) method to execute the statement.

When you set the SQL property, TSQLQuery (see page 20298) checks the statement for parameters and initializes the Params (see page 20315) property to reflect any parameters in the statement. You can subsequently use the Params (see page 20315) property to adjust the datatype of any parameters and supply values for the parameters.

At design-time, edit the SQL statement using the property editor that appears when you click the ellipsis button for the property in the Object Inspector. At runtime, use properties and methods of TStrings to clear the current contents, add new contents, or to modify existing contents:

SQLQuery1.SQL.Clear;

```
SQLQuery1.SQL.Add('SELECT ' + Edit1.Text + ' FROM ' + Edit2.Text);
if Length(Edit3.Text) <> 0 then
```

SQLQuery1.SQL.Add('ORDER BY ' + Edit3.Text (see page 20318))

SQLQuery1->SQL->Clear();

```
SQLQuery1->SQL->Add("SELECT " + Edit1->Text + " FROM " + Edit2.Text);
```

```
if (!Edit3->Text->IsEmpty())
SQLQuery1->SQL->Add("ORDER BY " + Edit3.Text (see page 20318));
```

You can also use the methods of SQL to load a query from a file.

Note: The SQL you supply for this property must be valid for the database server to which the TSQLQuery (see page 20298) object is connected.

See Also

Add (see page 3033)

Active (see page 7004)

Open (see page 6976)

ExecSQL (see page 20311)

Params (see page 20197)

Text (see page 20318)

Representing the Results of a Query

Specifying the Command to Execute

149.1.9.5.8 TSQLQuery.SQLConnection Property

Specifies the SQL connection component that connects the dataset to a database server.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

Use SQLConnection to specify an SQL connection object that connects the dataset to a database server.

At design-time, select from the list of available TSQLConnection components in the Object Inspector. At runtime, set SQLConnection to reference an existing TSQLConnection object.

SQLConnection should be the first property you set on the dataset. Only after the SQLConnection property is set can you use the Command Text editor to specify a query for the CommandText property or populate the dataset with data by setting Active to true. Similarly, SQLConnection must be set before the dataset can initialize the Params property when you set CommandText.

See Also

Active (see page 7004)

CommandText (see page 20189)

Params (see page 20197)

Using Query-type Datasets

Using Stored Procedure-type Datasets

Using Table Type Datasets

Connecting Directly to a Database Server

Connecting to the Database Server

149.1.9.5.9 TSQLQuery.Text Property

Indicates the SQL (see page 20316) statement as a single string.

Pascal

```
property Text: string;
```

C++

```
__property AnsiString Text;
```

Description

Use Text as a convenient way to read the entire SQL (see page 20316) query as a single string, rather than using the SQL (see page 20316) property, which may divide the query into substrings. Text is the same as the Text property of the TStrings object that implements the SQL (see page 20316) property.

See Also

SQL (see page 20316)

Text (see page 3084)

149.1.10 TSQLStoredProc Class

TSQLStoredProc represents a stored procedure that is executed using dbExpress.

Class Hierarchy

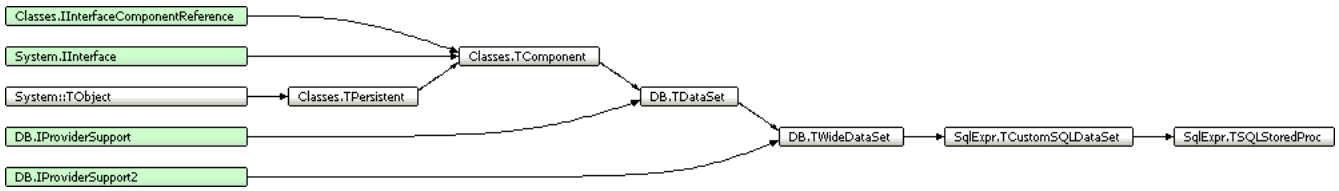

Pascal

```
TSQLStoredProc = class(TCustomSQLDataSet);
```

C++

```
class TSQLStoredProc : public TCustomSQLDataSet;
```

File

SqlExpr

Description

Use TSQLStoredProc to execute a stored procedure on a database server that is accessed via TSQLConnection (see page 20217). TSQLStoredProc can represent the result set if the stored procedure returns a cursor. You can add a TSQLStoredProc component to a form or data module at design time, or create one dynamically at runtime.

TSQLStoredProc is a unidirectional dataset. Unlike other datasets, unidirectional datasets do not buffer multiple records in memory. Because of this, you can only navigate using the First and Next methods. There is no built-in editing support: you can only edit the data in an SQL stored procedure by explicitly creating an SQL UPDATE command or by connecting the dataset to a client dataset using a provider. Features that require buffering multiple records, such as filters or lookup fields, are not available.

Before you can use a TSQLStoredProc component, it must be connected to the database server. Therefore, the first step to take

when working with TSQLStoredProc is to set the SQLConnection (see page 20335) property. Once the dataset is connected to a server, use the StoredProcName (see page 20336) property to specify the stored procedure to execute.

See Also

TSQLDataSet (see page 20265)

TSQLQuery (see page 20298)

TSQLTable (see page 20337)

TSQLConnection (see page 20217)

TClientDataSEt

TDataSEtProvider

Types of Datasets

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Types of Unidirectional Datasets

149.1.10.1 TSQLStoredProc Members

The following tables list the members exposed by TSQLStoredProc.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	AddFieldDesc (see page 20163)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 20163)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
V	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
V	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
V	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
	CheckPrepareError (see page 20164)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
V	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
V	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
	ClearIndexDefs (see page 20164)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
V	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
V	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.

	CloseCursor (see page 20164)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 20164)	This is CloseStatement, a member of class TCustomSQLDataSet.
	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
	DataEvent (see page 6924)	This is DataEvent, a member of class TDataSet.
	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 20165)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.
	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	ExecSQL (see page 20165)	This is ExecSQL, a member of class TCustomSQLDataSet.
	ExecuteStatement (see page 20166)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FreeBuffers (see page 20166)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCommand (see page 20166)	This is FreeCommand, a member of class TCustomSQLDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeReader (see page 20166)	This is FreeReader, a member of class TCustomSQLDataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
	GetCanModify (see page 20167)	This is GetCanModify, a member of class TCustomSQLDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.

	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
	GetDataSource (see page 20167)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetObjectTypeNames (see page 20169)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 20169)	This is GetOutputParams, a member of class TCustomSQLDataSet.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
	GetRecord (see page 20170)	This is GetRecord, a member of class TCustomSQLDataSet.
	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.
	GetRecordCount (see page 20170)	This is GetRecordCount, a member of class TCustomSQLDataSet.
	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
	GetSortFieldNames (see page 20170)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
	InitRecord (see page 20170)	This is InitRecord, a member of class TCustomSQLDataSet.
	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.
	InternalClose (see page 20171)	This is InternalClose, a member of class TCustomSQLDataSet.
	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
	InternalHandleException (see page 20171)	This is InternalHandleException, a member of class TCustomSQLDataSet.
	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
	InternalInitFieldDefs (see page 20171)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
	InternalOpen (see page 20171)	This is InternalOpen, a member of class TCustomSQLDataSet.
	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
	InternalRefresh (see page 20171)	This is InternalRefresh, a member of class TCustomSQLDataSet.
	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.

	IsCursorOpen (see page 20172)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
	Loaded (see page 20172)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 20173)	This is LocateRecord, a member of class TCustomSQLDataSet.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	OpenCursor (see page 6976)	This is OpenCursor, a member of class TDataSet.
	OpenCursor (see page 20174)	This is OpenCursor, a member of class TCustomSQLDataSet.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
	OpenSchema (see page 20174)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 20175)	This is PropertyChanged, a member of class TCustomSQLDataSet.
	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
	PSEndTransaction (see page 20175)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
	PSExecute (see page 20176)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 20176)	This is PSExecuteStatement, a member of class TCustomSQLDataSet.
	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
	PSExecuteStatement (see page 7535)	This is the overview for the PSExecuteStatement method overload.
	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 20176)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 6982)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 7537)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 20177)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandTextW (see page 7537)	This is PSGetCommandTextW, a member of class TWideDataSet.
	PSGetCommandType (see page 6982)	Gets the command type to be executed by PSExecute.
	PSGetCommandType (see page 20177)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 20178)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 20178)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7538)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetKeyFieldsW (see page 7538)	This is PSGetKeyFieldsW, a member of class TWideDataSet.
	PSGetKeyFieldsW (see page 20179)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetParams (see page 20179)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7538)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetQuoteCharW (see page 7539)	This is PSGetQuoteCharW, a member of class TWideDataSet.
	PSGetQuoteCharW (see page 20179)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7539)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetTableNameW (see page 7539)	This is PSGetTableNameW, a member of class TWideDataSet.
	PSGetTableNameW (see page 20179)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.

	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSGetUpdateException (see page 20180)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 20180)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 20180)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 20181)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 20181)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 20182)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetCommandText (see page 7540)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 20182)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSStartTransaction (see page 20182)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 20183)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetBufListSize (see page 20184)	This is SetBufListSize, a member of class TCustomSQLDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCommandText (see page 20184)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFCommandText (see page 20184)	This is SetFCommandText, a member of class TCustomSQLDataSet.
	SetFieldData (see page 20184)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetMaxBlobSize (see page 20184)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 6999)	Sets the value of the Name property.

	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
V	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetParamsFromCursor (see page 20185)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
	SetSortFieldNames (see page 20186)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
V	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.
V	UpdateIndexDefs (see page 20186)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
	BlobBuffer (see page 20189)	Reserves a buffer for storing BLOB values.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
	CommandType (see page 20190)	Indicates the meaning of the CommandText (see page 20189) property.
	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
	CurrentBlobSize (see page 20191)	Indicates the size of the most recently read BLOB field.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
R	DataLink (see page 20191)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	DataSource (see page 20192)	Links the SQL dataset to another (master) dataset.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
R	InternalConnection (see page 20194)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	LastError (see page 20194)	Saves the error message for the last error returned by dbExpress.
	NativeCommand (see page 20195)	Represents the SQL command that is sent to the server.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
R	ParamCount (see page 20197)	Indicates the number of parameters listed by the Params (see page 20197) property.

	ProcParams (see page 20199)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
	Reserved (see page 7035)	For internal use only.
R	RowsAffected (see page 20200)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 20200)	Identifies the metadata with which to populate the dataset.
	SortFieldNames (see page 20201)	Indicates the sort order when CommandType (see page 20190) is ctTable.

Public Events

	Name	Description
	AfterCancel (see page 7038)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 7040)	Occurs after an application closes a dataset.
	AfterDelete (see page 7041)	Occurs after an application deletes a record.
	AfterEdit (see page 7044)	Occurs after an application starts editing a record.
	AfterInsert (see page 7044)	Occurs after an application inserts a new record.
	AfterOpen (see page 7045)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 7046)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 7047)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 7047)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 7048)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 7048)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 7049)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 7049)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 7050)	Occurs before an application enters insert mode.
	BeforeOpen (see page 7051)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 7052)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 7052)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 7053)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 7053)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 7054)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 7054)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 7056)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 7056)	Occurs when an application attempts to modify or insert a record and an exception is raised.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
V	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Close (see page 6920)	Closes a dataset.
V	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
V	Create (see page 20331)	Creates an instance of a TSQLStoredProc (see page 20318) component.
V	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
V	CreateBlobStream (see page 20164)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 6904)	Destroys the TDataSet (see page 6894) instance.
V	Destroy (see page 20161)	Destroys the TCustomSQLDataSet (see page 20150) instance.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
V	ExecProc (see page 20331)	Executes the stored procedure when it does not return a cursor.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
V	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
V	GetBlobFieldData (see page 20166)	Retrieves the current value of a BLOB field into a buffer.

	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
	GetDetailLinkFields (see page 20167)	Lists the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
	GetFieldData (see page 20168)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetKeyFieldNames (see page 20169)	Fills a list with the names of all fields in all indexes defined for the dataset.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetQuoteChar (see page 20169)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	IsSequenced (see page 20172)	Indicates whether the dataset can use record numbers to indicate the order of records.
	Last (see page 6968)	Moves to the last record in the dataset.
	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Locate (see page 20173)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	Lookup (see page 20174)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	NextRecordSet (see page 20332)	Provides access to a secondary recordset.
	Open (see page 6976)	Opens the dataset.
	ParamByName (see page 20175)	This is ParamByName, a member of class TCustomSQLDataSet.

V	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
V	PrepareStatement (see page 20332)	Prepares the stored procedure for execution.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSEndTransaction (see page 7589)	Ends a transaction that was started by a call to PSStartTransaction (see page 7597).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecute (see page 7590)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSExecuteStatement (see page 7590)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 7591)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 7579)	Gets the SQL command to be executed by PSExecute (see page 7578).
	PSGetCommandText (see page 7591)	Gets the SQL command to be executed by PSExecute (see page 7590).
	PSGetCommandType (see page 7580)	Gets the command type to be executed by PSExecute (see page 7578).
	PSGetCommandType (see page 7591)	Gets the command type to be executed by PSExecute (see page 7590).
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 7592)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 7592)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7593)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetParams (see page 7593)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7594)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7594)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSGetUpdateException (see page 7594)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 7595)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 7595)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 7596)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 7596)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7585)	Assigns a specified SQL command to be executed by PSExecute (see page 7578).
	PSSetCommandText (see page 7596)	Assigns a specified SQL command to be executed by PSExecute (see page 7590).
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 7597)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).

	PSStartTransaction (see page 7597)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7589).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 7598)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetSchemaInfo (see page 20185)	Indicates whether the dataset represents metadata from the server, and if so, what kind.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.
	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.

Public Properties

	Name	Description
	Active (see page 7004)	Specifies whether or not a dataset is open.
R	AggFields (see page 7006)	Contains any aggregate fields.
	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
	CommandText (see page 20189)	Specifies the command the dataset executes.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
R	DataSource (see page 7014)	Represents the data source of another dataset that supplies values to the dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignerData (see page 20192)	Stores custom data.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
	FieldDefs (see page 7019)	Points to the list of field definitions for the dataset.

R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
	IndexDefs (see page 20193)	Contains the definitions of all indexes defined on the server for the dataset.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	Modified (see page 7029)	Indicates whether the active record is modified.
	ObjectView (see page 7031)	Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Prepared (see page 20198)	Specifies whether the command is prepared before execution.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
R	RecordCount (see page 20199)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.
	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	TransactionLevel (see page 20202)	Indicates the transaction to which this dataset belongs.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterClose (see page 20203)	Occurs after a data source connection is closed.
	AfterOpen (see page 20203)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 20203)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 20204)	Occurs after a result set is scrolled.
	BeforeClose (see page 20204)	Occurs just before a data connection is closed.
	BeforeOpen (see page 20204)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 20204)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 20204)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 20204)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 20205)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 20189) property.
	ParseInsertSql (see page 20205)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 20189) property.
	ParseSelectSql (see page 20205)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 20189) property.
	ParseUpdateSql (see page 20205)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 20189) property.

Published Properties

	Name	Description
	Active (see page 20187)	Specifies whether or not a dataset is open.
	GetMetadata (see page 20193)	Specifies whether the SQL dataset fetches metadata information along with the data.
	MaxBlobSize (see page 20333)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NumericMapping (see page 20195)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 20196)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.

	PackageName (see page 20334)	Specifies the name of the Oracle package to which the stored procedure belongs.
	ParamCheck (see page 20334)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 20334)	Represents the parameters for a query or stored procedure.
	SchemaName (see page 20201)	Specifies the name of the database schema.
	SQLConnection (see page 20335)	Specifies the SQL connection component that connects the dataset to a database server.
	StoredProcName (see page 20336)	Specifies the SQL statement to execute on the database server.
	Tag (see page 2808)	Stores an integer value as part of a component.

149.1.10.2 TSQLStoredProc.Create Constructor

Creates an instance of a TSQLStoredProc (see page 20318) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSQLStoredProc(TComponent * AOwner);
```

Description

Call Create to instantiate an SQL stored procedure component at runtime. SQL stored procedures placed in forms or data modules at design time are created automatically.

AOwner is the component, typically a form or data module, that is responsible for freeing the TSQLStoredProc (see page 20318) instance. It becomes the value of the Owner property.

See Also

Owner (see page 2807)

Create (see page 6904)

149.1.10.3 TSQLStoredProc Methods

The methods of the TSQLStoredProc class are listed here.

Public Methods

	Name	Description
	ExecProc (see page 20331)	Executes the stored procedure when it does not return a cursor.
	NextRecordSet (see page 20332)	Provides access to a secondary recordset.
	PrepareStatement (see page 20332)	Prepares the stored procedure for execution.

149.1.10.3.1 TSQLStoredProc.ExecProc Method

Executes the stored procedure when it does not return a cursor.

Pascal

```
function ExecProc: Integer; virtual;
```

C++

```
virtual __fastcall int ExecProc();
```

Description

Call ExecProc to execute the stored procedure specified by StoredProcName (see page 20336) if it does not return a cursor.

ExecProc returns the number of rows affected by the stored procedure. This becomes the value of the RowsAffected (see page 20200) property.

To speed performance, an application should ordinarily prepare the stored procedure by setting the Prepared (see page 20198) property to true before calling ExecProc the first time.

Note: Do not use ExecProc for stored procedures that return a cursor. When the stored procedure returns data, use the Open method or set the Active (see page 20187) property to true.

See Also

StoredProcName (see page 20336)

RowsAffected (see page 20200)

Prepared (see page 20198)

Executing the Command

149.1.10.3.2 TSQLStoredProc.NextRecordSet Method

Provides access to a secondary recordset.

Pascal

```
function NextRecordSet: TCustomSQLDataSet;
```

C++

```
__fastcall TCustomSQLDataSet NextRecordSet();
```

Description

Call NextRecordSet to obtain secondary datasets when accessing the results of a stored procedure that returns multiple sets of records. Opening the dataset fetches the first set of records. NextRecordSet provides access to the subsequent sets.

NextRecordSet returns a newly created TCustomSQLDataSet (see page 20150) component that provides access to the next set of records. The first time you call NextRecordSet, it returns a dataset for the second set of records. The next call returns a third dataset, and so on, until there are no more sets of records. When there are no additional datasets, NextRecordSet returns nil (Delphi) or NULL (C++).

See Also

Fetching the Data

Fetching the Data using TSQLDataSet

149.1.10.3.3 TSQLStoredProc.PrepareStatement Method

Prepares the stored procedure for execution.

Pascal

```
procedure PrepareStatement; override;
```

C++

```
virtual __fastcall PrepareStatement();
```

Description

PrepareStatement performs most of the necessary tasks for preparing a stored procedure to be executed on the server. It ensures that dbExpress and the database server have allocated any resources they need for executing the stored procedure and binding its parameters.

Do not call PrepareStatement when preparing a stored procedure for execution. Instead, set the Prepared (see page 20198) property to true. When you use the Prepared (see page 20198) property instead, the stored procedure component keeps track of the fact that it has already been prepared and need not repeat the process every time it executes. In addition, when you set the Prepared (see page 20198) property, the component initializes internal properties that keep track of the number of rows affected by the stored procedure.

See Also

Prepared (see page 20198)

149.1.10.4 TSQLStoredProc Properties

The properties of the TSQLStoredProc class are listed here.

Published Properties

	Name	Description
	MaxBlobSize (see page 20333)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	PackageName (see page 20334)	Specifies the name of the Oracle package to which the stored procedure belongs.
	ParamCheck (see page 20334)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 20334)	Represents the parameters for a query or stored procedure.
	SQLConnection (see page 20335)	Specifies the SQL connection component that connects the dataset to a database server.
	StoredProcName (see page 20336)	Specifies the SQL statement to execute on the database server.

149.1.10.4.1 TSQLStoredProc.MaxBlobSize Property

Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

MaxBlobSize limits the amount of data that can be fetched from Binary Large Object (BLOB) fields. BLOB fields can hold an indefinite amount of data. However, SQL datasets read that data into a memory buffer. By limiting the amount of data that can be fetched from a BLOB field, you limit the size of the buffer.

If MaxBlobSize is –1, the dataset imposes no maximum BLOB size. If MaxBlobSize is 0, the dataset uses the parameters of the associated SQL connection, if any, to determine the maximum BLOB size.

Note: MaxBlobSize should agree with any parameters on the associated SQL connection component that limit the size of BLOB data.

See Also

GetBlobFieldData (see page 20166)

Params (see page 20253)

SQLConnection (see page 20201)

149.1.10.4.2 TSQLStoredProc.PackageName Property

Specifies the name of the Oracle package to which the stored procedure belongs.

Pascal

```
property PackageName: WideString;
```

C++

```
__property BSTR PackageName;
```

Description

Set PackageName to specify the name of the Oracle package that contains the stored procedure which the StoredProcName (see page 20336) property specifies. This property is only used when the stored procedure is defined on an Oracle server.

If the stored procedure does not belong to a package, the value of PackageName is an empty string.

See Also

StoredProcName (see page 20336)

GetPackageNames (see page 20231)

149.1.10.4.3 TSQLStoredProc.ParamCheck Property

Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

By default, the Params property is updated automatically to reflect any parameters that appear in the query specified by the CommandText property (or the SQL property for TSQLQuery) at runtime. That is, whenever the CommandText property changes, the Params property is regenerated to reflect the new set of parameters.

There are times when you want to override this default behavior. For example, when specifying a data definition language (DDL) statement that defines a stored procedure, the resulting SQL command may include parameters. These parameters, however, should not be used as parameters of the SQL dataset, because they are not parameters of the DDL command, but rather, they are parameters of the stored procedure that the command creates.

In order to override the default generation of the Params property in response to changes in CommandText, set ParamCheck to false.

See Also

Params (see page 20197)

CommandText (see page 20189)

SQL (see page 20316)

149.1.10.4.4 TSQLStoredProc.Params Property

Represents the parameters for a query or stored procedure.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Params is a collection of TParam objects that represent the parameters of a query or stored procedure specified by the SQL dataset. When you specify a query by setting the CommandText property (or the SQL property in TSQLQuery), the SQL dataset automatically parse the query and fills Params with a TParam object for every parameter in the query. Similarly, when you specify a stored procedure by setting the CommandText property (or the StoredProcName property in TSQLStoredProc), the SQL dataset automatically asks the server for a list of stored procedure parameters and fills Params with a TParam object for every parameter of the stored procedure.

At design-time, use the Params property editor to access the individual TParam objects. For queries, use this property editor to check the DataType property of each parameter and to assign values to the parameters. For stored procedures, you may need to do more, changing the type or number of parameters if the server does not supply complete information about stored procedure parameters.

You can also use Params to assign parameter properties in code at runtime.

Note: If you do not assign values to input parameters before opening the dataset, it fetches values from the dataset identified by the DataSource property. However, this only works if the individual TParam objects have the same names as fields in the master dataset.

See Also

ParamCheck (see page 20196)

SQL (see page 20316)

StoredProcName (see page 20336)

CommandText (see page 20189)

TParam (see page 7386)

Value (see page 7433)

DataType (see page 7427)

DataSource (see page 20192)

Configuring TSQL Connection

Connecting to Databases with TDatabase

Using TQuery

Using TSimpleDataSet

149.1.10.4.5 TSQLStoredProc.SQLConnection Property

Specifies the SQL connection component that connects the dataset to a database server.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

Use SQLConnection to specify an SQL connection object that connects the dataset to a database server.

At design-time, select from the list of available TSQLConnection components in the Object Inspector. At runtime, set SQLConnection to reference an existing TSQLConnection object.

SQLConnection should be the first property you set on the dataset. Only after the SQLConnection property is set can you use the Command Text editor to specify a query for the CommandText property or populate the dataset with data by setting Active to true. Similarly, SQLConnection must be set before the dataset can initialize the Params property when you set CommandText.

See Also

Active (see page 7004)

CommandText (see page 20189)

Params (see page 20197)

Using Query-type Datasets

Using Stored Procedure-type Datasets

Using Table Type Datasets

Connecting Directly to a Database Server

Connecting to the Database Server

149.1.10.4.6 TSQLStoredProc.StoredProcName Property

Specifies the SQL statement to execute on the database server.

Pascal

```
property StoredProcName: WideString;
```

C++

```
__property BSTR StoredProcName;
```

Description

Set StoredProcName to specify the name of the stored procedure to call on the server. If StoredProcName does not match the name of an existing stored procedure on the server, an exception is raised.

When you set the StoredProcName property, TSQLStoredProc (see page 20318) queries the database server to ascertain the parameters of the specified stored procedure. It uses this information to initialize the Params (see page 20334) property. You can subsequently use the Params (see page 20334) property to adjust the properties of any parameters and supply values for the parameters. Some servers do not supply information about stored procedure parameters. In those cases, you must explicitly add the parameters.

See Also

Params (see page 20197)

PackageName (see page 20334)

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSQLStoredProc

149.1.11 TSQLTable Class

TSQLTable represents a database table that is accessed using dbExpress.

Class Hierarchy

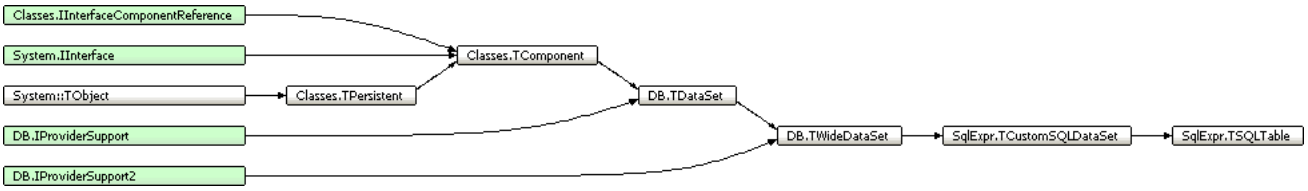

Pascal

```
TSQLTable = class(TCustomSQLDataSet);
```

C++

```
class TSQLTable : public TCustomSQLDataSet;
```

File

SqlExpr

Description

Use TSQLTable to represent a table on a database server that is accessed via TSQLConnection (see page 20217). TSQLTable generates a query to fetch all of the rows and columns in a table you specify. You can add a TSQLTable component to a form or data module at design time, or create one dynamically at runtime.

TSQLTable is a unidirectional dataset. Unlike other datasets, unidirectional datasets do not buffer multiple records in memory. Because of this, you can only navigate using the First and Next methods. There is no built-in editing support: you can only edit the data in an SQL table by explicitly creating an SQL UPDATE command or by connecting the table to a client dataset using a provider. Features that require buffering multiple records, such as filters or lookup fields, are not available.

Before you can use a TSQLTable component, it must be connected to the database server. Therefore, the first step to take when working with TSQLTable is to set the SQLConnection (see page 20358) property. Once the dataset is connected to a server, use the TableName (see page 20359) property to specify the table this component represents.

See Also

TSQLDataSet (see page 20265)

TSQLQuery (see page 20298)

TSQLStoredProc (see page 20318)

TSQLConnection (see page 20217)

TClientDataSEt

TDataSEtProvider

Types of Datasets

Types of Unidirectional Datasets

149.1.11.1 TSQLTable Members

The following tables list the members exposed by TSQLTable.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActivateBuffers (see page 6909)	This is ActivateBuffers, a member of class TDataSet.
	AddFieldDesc (see page 20163)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 20163)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
V	AllocRecordBuffer (see page 6909)	This is AllocRecordBuffer, a member of class TDataSet.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	BindFields (see page 6915)	This is BindFields, a member of class TDataSet.
V	BlockReadNext (see page 6915)	This is BlockReadNext, a member of class TDataSet.
	BookmarkAvailable (see page 6916)	This is BookmarkAvailable, a member of class TDataSet.
V	CalculateFields (see page 6916)	This is CalculateFields, a member of class TDataSet.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CheckActive (see page 6917)	This is CheckActive, a member of class TDataSet.
	CheckBiDirectional (see page 6918)	This is CheckBiDirectional, a member of class TDataSet.
V	CheckFieldCompatibility (see page 6919)	This is CheckFieldCompatibility, a member of class TDataSet.
V	CheckInactive (see page 6919)	This is CheckInactive, a member of class TDataSet.
	CheckPrepareError (see page 20164)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
V	ClearBuffers (see page 6919)	This is ClearBuffers, a member of class TDataSet.
V	ClearCalcFields (see page 6919)	This is ClearCalcFields, a member of class TDataSet.
	ClearIndexDefs (see page 20164)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
V	CloseBlob (see page 6921)	This is CloseBlob, a member of class TDataSet.
V	CloseCursor (see page 6921)	This is CloseCursor, a member of class TDataSet.
V	CloseCursor (see page 20164)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 20164)	This is CloseStatement, a member of class TCustomSQLDataSet.
V	CreateFields (see page 6923)	This is CreateFields, a member of class TDataSet.
V	CreateNestedDataSet (see page 6923)	This is CreateNestedDataSet, a member of class TDataSet.
V	DataConvert (see page 6924)	This is DataConvert, a member of class TDataSet.
V	DataEvent (see page 6924)	This is DataEvent, a member of class TDataSet.
V	DefChanged (see page 6924)	This is DefChanged, a member of class TDataSet.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 20165)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DestroyFields (see page 6925)	This is DestroyFields, a member of class TDataSet.
V	DoAfterCancel (see page 6927)	This is DoAfterCancel, a member of class TDataSet.
V	DoAfterClose (see page 6927)	This is DoAfterClose, a member of class TDataSet.
V	DoAfterDelete (see page 6928)	This is DoAfterDelete, a member of class TDataSet.
V	DoAfterEdit (see page 6928)	This is DoAfterEdit, a member of class TDataSet.
V	DoAfterInsert (see page 6928)	This is DoAfterInsert, a member of class TDataSet.
V	DoAfterOpen (see page 6928)	This is DoAfterOpen, a member of class TDataSet.
V	DoAfterPost (see page 6929)	This is DoAfterPost, a member of class TDataSet.

	DoAfterRefresh (see page 6929)	This is DoAfterRefresh, a member of class TDataSet.
	DoAfterScroll (see page 6929)	This is DoAfterScroll, a member of class TDataSet.
	DoBeforeCancel (see page 6929)	This is DoBeforeCancel, a member of class TDataSet.
	DoBeforeClose (see page 6929)	This is DoBeforeClose, a member of class TDataSet.
	DoBeforeDelete (see page 6930)	This is DoBeforeDelete, a member of class TDataSet.
	DoBeforeEdit (see page 6930)	This is DoBeforeEdit, a member of class TDataSet.
	DoBeforeInsert (see page 6930)	This is DoBeforeInsert, a member of class TDataSet.
	DoBeforeOpen (see page 6930)	This is DoBeforeOpen, a member of class TDataSet.
	DoBeforePost (see page 6931)	This is DoBeforePost, a member of class TDataSet.
	DoBeforeRefresh (see page 6931)	This is DoBeforeRefresh, a member of class TDataSet.
	DoBeforeScroll (see page 6931)	This is DoBeforeScroll, a member of class TDataSet.
	DoOnCalcFields (see page 6931)	This is DoOnCalcFields, a member of class TDataSet.
	DoOnNewRecord (see page 6931)	This is DoOnNewRecord, a member of class TDataSet.
	ExecSQL (see page 20165)	This is ExecSQL, a member of class TCustomSQLDataSet.
	ExecuteStatement (see page 20166)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FieldByNumber (see page 6938)	This is FieldByNumber, a member of class TDataSet.
	FindRecord (see page 6942)	This is FindRecord, a member of class TDataSet.
	FreeBuffers (see page 20166)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCommand (see page 20166)	This is FreeCommand, a member of class TCustomSQLDataSet.
	FreeFieldBuffers (see page 6946)	This is FreeFieldBuffers, a member of class TDataSet.
	FreeReader (see page 20166)	This is FreeReader, a member of class TCustomSQLDataSet.
	FreeRecordBuffer (see page 6947)	This is FreeRecordBuffer, a member of class TDataSet.
	GetAggRecordCount (see page 6947)	This is GetAggRecordCount, a member of class TDataSet.
	GetAggregateValue (see page 6947)	This is GetAggregateValue, a member of class TDataSet.
	GetBookmarkData (see page 6949)	This is GetBookmarkData, a member of class TDataSet.
	GetBookmarkFlag (see page 6949)	This is GetBookmarkFlag, a member of class TDataSet.
	GetBookmarkStr (see page 6949)	This is GetBookmarkStr, a member of class TDataSet.
	GetCalcFields (see page 6950)	This is GetCalcFields, a member of class TDataSet.
	GetCanModify (see page 6950)	This is GetCanModify, a member of class TDataSet.
	GetCanModify (see page 20167)	This is GetCanModify, a member of class TCustomSQLDataSet.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6950)	This is GetChildren, a member of class TDataSet.
	GetDataSource (see page 6951)	This is GetDataSource, a member of class TDataSet.
	GetDataSource (see page 20167)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetFieldClass (see page 6952)	This is GetFieldClass, a member of class TDataSet.
	GetFieldFullName (see page 6953)	This is GetFieldFullName, a member of class TDataSet.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIndexDefs (see page 6954)	This is GetIndexDefs, a member of class TDataSet.
	GetIsIndexField (see page 6954)	This is GetIsIndexField, a member of class TDataSet.
	GetNextRecord (see page 6955)	This is GetNextRecord, a member of class TDataSet.
	GetNextRecords (see page 6955)	This is GetNextRecords, a member of class TDataSet.
	GetObjectTypeNames (see page 20169)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 20169)	This is GetOutputParams, a member of class TCustomSQLDataSet.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPriorRecord (see page 6955)	This is GetPriorRecord, a member of class TDataSet.
	GetPriorRecords (see page 6955)	This is GetPriorRecords, a member of class TDataSet.
	GetRecNo (see page 6956)	This is GetRecNo, a member of class TDataSet.
	GetRecord (see page 6956)	This is GetRecord, a member of class TDataSet.
	GetRecord (see page 20170)	This is GetRecord, a member of class TCustomSQLDataSet.
	GetRecordCount (see page 6956)	This is GetRecordCount, a member of class TDataSet.

V	GetRecordCount (see page 20170)	This is GetRecordCount, a member of class TCustomSQLDataSet.
V	GetRecordSize (see page 6956)	This is GetRecordSize, a member of class TDataSet.
	GetSortFieldNames (see page 20170)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
V	GetStateFieldValue (see page 6956)	This is GetStateFieldValue, a member of class TDataSet.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	InitFieldDefs (see page 6958)	This is InitFieldDefs, a member of class TDataSet.
	InitFieldDefsFromFields (see page 6959)	This is InitFieldDefsFromFields, a member of class TDataSet.
V	InitRecord (see page 6959)	This is InitRecord, a member of class TDataSet.
V	InitRecord (see page 20170)	This is InitRecord, a member of class TCustomSQLDataSet.
V	InternalAddRecord (see page 6963)	This is InternalAddRecord, a member of class TDataSet.
V	InternalCancel (see page 6963)	This is InternalCancel, a member of class TDataSet.
A	InternalClose (see page 6963)	This is InternalClose, a member of class TDataSet.
V	InternalClose (see page 20171)	This is InternalClose, a member of class TCustomSQLDataSet.
V	InternalDelete (see page 6964)	This is InternalDelete, a member of class TDataSet.
V	InternalEdit (see page 6964)	This is InternalEdit, a member of class TDataSet.
V	InternalFirst (see page 6964)	This is InternalFirst, a member of class TDataSet.
V	InternalGotoBookmark (see page 6964)	This is InternalGotoBookmark, a member of class TDataSet.
A	InternalHandleException (see page 6964)	This is InternalHandleException, a member of class TDataSet.
V	InternalHandleException (see page 20171)	This is InternalHandleException, a member of class TCustomSQLDataSet.
A	InternalInitFieldDefs (see page 6965)	This is InternalInitFieldDefs, a member of class TDataSet.
V	InternalInitFieldDefs (see page 20171)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
V	InternalInitRecord (see page 6965)	This is InternalInitRecord, a member of class TDataSet.
V	InternalInsert (see page 6965)	This is InternalInsert, a member of class TDataSet.
V	InternalLast (see page 6965)	This is InternalLast, a member of class TDataSet.
A	InternalOpen (see page 6966)	This is InternalOpen, a member of class TDataSet.
V	InternalOpen (see page 20171)	This is InternalOpen, a member of class TCustomSQLDataSet.
V	InternalPost (see page 6966)	This is InternalPost, a member of class TDataSet.
V	InternalRefresh (see page 6966)	This is InternalRefresh, a member of class TDataSet.
V	InternalRefresh (see page 20171)	This is InternalRefresh, a member of class TCustomSQLDataSet.
V	InternalSetToRecord (see page 6966)	This is InternalSetToRecord, a member of class TDataSet.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
A	IsCursorOpen (see page 6966)	This is IsCursorOpen, a member of class TDataSet.
V	IsCursorOpen (see page 20172)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 6969)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 20172)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 20173)	This is LocateRecord, a member of class TCustomSQLDataSet.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	OpenCursor (see page 20351)	This is OpenCursor, a member of class TSQLTable.
	OpenCursorComplete (see page 6977)	This is OpenCursorComplete, a member of class TDataSet.
	OpenParentDataSet (see page 6977)	This is OpenParentDataSet, a member of class TDataSet.
V	OpenSchema (see page 20174)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PropertyChanged (see page 20175)	This is PropertyChanged, a member of class TCustomSQLDataSet.
V	PSEndTransaction (see page 6979)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSEndTransaction (see page 20175)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSExecute (see page 6980)	Executes the SQL command associated with the dataset.
V	PSExecute (see page 20176)	Executes the SQL command associated with the dataset.
V	PSExecuteStatement (see page 20176)	This is PSExecuteStatement, a member of class TCustomSQLDataSet.
V	PSExecuteStatement (see page 6980)	This is the overview for the PSExecuteStatement method overload.
V	PSExecuteStatement (see page 7535)	This is the overview for the PSExecuteStatement method overload.
V	PSGetAttributes (see page 6981)	Returns information that is stored in data packets as Name/Value pairs.

	PSGetAttributes (see page 20176)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 6982)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 7537)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandText (see page 20177)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandTextW (see page 6982)	This is PSGetCommandTextW, a member of class TDataSet.
	PSGetCommandTextW (see page 7537)	This is PSGetCommandTextW, a member of class TWideDataSet.
	PSGetCommandType (see page 6982)	Gets the command type to be executed by PSExecute.
	PSGetCommandType (see page 20177)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 6983)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 20178)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 6983)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 20178)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 6984)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7538)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFieldsW (see page 6984)	This is PSGetKeyFieldsW, a member of class TDataSet.
	PSGetKeyFieldsW (see page 7538)	This is PSGetKeyFieldsW, a member of class TWideDataSet.
	PSGetKeyFieldsW (see page 20179)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
	PSGetParams (see page 6984)	Returns the current parameter values of the dataset.
	PSGetParams (see page 20179)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 6985)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7538)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteCharW (see page 6985)	This is PSGetQuoteCharW, a member of class TDataSet.
	PSGetQuoteCharW (see page 7539)	This is PSGetQuoteCharW, a member of class TWideDataSet.
	PSGetQuoteCharW (see page 20179)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
	PSGetTableName (see page 6985)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7539)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableNameW (see page 6986)	This is PSGetTableNameW, a member of class TDataSet.
	PSGetTableNameW (see page 7539)	This is PSGetTableNameW, a member of class TWideDataSet.
	PSGetTableNameW (see page 20179)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
	PSGetUpdateException (see page 6986)	Generates an EUpdateError object based on another exception object.
	PSGetUpdateException (see page 20180)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 6987)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 20180)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 6987)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 20180)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 6987)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 20181)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 6988)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 20181)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 20182)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetCommandText (see page 6988)	This is the overview for the PSSetCommandText method overload.
	PSSetCommandText (see page 7540)	This is the overview for the PSSetCommandText method overload.
	PSSetParams (see page 6989)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 20182)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 6989)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSStartTransaction (see page 20182)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 6990)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 20183)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.

	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RefreshInternalCalcFields (see page 6993)	This is RefreshInternalCalcFields, a member of class TDataSet.
	ResetAggField (see page 6993)	This is ResetAggField, a member of class TDataSet.
	RestoreState (see page 6993)	This is RestoreState, a member of class TDataSet.
	SetActive (see page 6994)	This is SetActive, a member of class TDataSet.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetBlockReadSize (see page 6994)	This is SetBlockReadSize, a member of class TDataSet.
	SetBookmarkData (see page 6994)	This is SetBookmarkData, a member of class TDataSet.
	SetBookmarkFlag (see page 6995)	This is SetBookmarkFlag, a member of class TDataSet.
	SetBookmarkStr (see page 6995)	This is SetBookmarkStr, a member of class TDataSet.
	SetBufListSize (see page 6995)	This is SetBufListSize, a member of class TDataSet.
	SetBufListSize (see page 20184)	This is SetBufListSize, a member of class TCustomSQLDataSet.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6995)	This is SetChildOrder, a member of class TDataSet.
	SetCommandText (see page 20184)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetCurrentRecord (see page 6996)	This is SetCurrentRecord, a member of class TDataSet.
	SetDataSetField (see page 6996)	This is SetDataSetField, a member of class TDataSet.
	SetDefaultFields (see page 6996)	This is SetDefaultFields, a member of class TDataSet.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFCommandText (see page 20184)	This is SetFCommandText, a member of class TCustomSQLDataSet.
	SetFieldData (see page 20184)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetFieldData (see page 6996)	This is the overview for the SetFieldData method overload.
	SetFiltered (see page 6997)	This is SetFiltered, a member of class TDataSet.
	SetFilterOptions (see page 6998)	This is SetFilterOptions, a member of class TDataSet.
	SetFilterText (see page 6998)	This is SetFilterText, a member of class TDataSet.
	SetFound (see page 6998)	This is SetFound, a member of class TDataSet.
	SetIndexField (see page 20352)	This is SetIndexField, a member of class TSQLTable.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetMaxBlobSize (see page 20184)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetModified (see page 6998)	This is SetModified, a member of class TDataSet.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 6999)	Sets the value of the Name property.
	SetObjectView (see page 6999)	This is SetObjectView, a member of class TDataSet.
	SetOnFilterRecord (see page 6999)	This is SetOnFilterRecord, a member of class TDataSet.
	SetParamsFromCursor (see page 20185)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetRecNo (see page 6999)	This is SetRecNo, a member of class TDataSet.
	SetSortFieldNames (see page 20186)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	SetState (see page 7000)	This is SetState, a member of class TDataSet.
	SetStateFieldValue (see page 7000)	This is SetStateFieldValue, a member of class TDataSet.
	SetTempState (see page 7000)	This is SetTempState, a member of class TDataSet.
	SetUniDirectional (see page 7000)	This is SetUniDirectional, a member of class TDataSet.
	TempBuffer (see page 7001)	This is TempBuffer, a member of class TDataSet.
	UpdateBufferCount (see page 7001)	This is UpdateBufferCount, a member of class TDataSet.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateIndexDefs (see page 7002)	This is UpdateIndexDefs, a member of class TDataSet.
	UpdateIndexDefs (see page 20186)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.

	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveRecord (see page 7005)	Indicates the index of the active record in the internal cache of records.
	BlobBuffer (see page 20189)	Reserves a buffer for storing BLOB values.
R	BlobFieldCount (see page 7007)	Indicates the number of columns in the dataset that represent Binary large object (BLOB) fields.
	BookmarkSize (see page 7010)	Indicates the number of bytes used to represent a bookmark.
R	BufferCount (see page 7010)	Indicates the number of records in the internal cache.
R	Buffers (see page 7011)	Provides indexed access to the record buffers in the internal cache.
R	CalcBuffer (see page 7011)	Points to the record buffer used during an OnCalcFields (see page 7053) event.
R	CalcFieldsSize (see page 7012)	Indicates the number of bytes in a record buffer dedicated to storing calculated fields and lookup fields.
	CommandType (see page 20190)	Indicates the meaning of the CommandText (see page 20189) property.
	Constraints (see page 7013)	Specifies record-level constraints that must be met when editing the data.
	CurrentBlobSize (see page 20191)	Indicates the size of the most recently read BLOB field.
R	CurrentRecord (see page 7013)	Indicates the index of the current record in the internal cache of record buffers.
R	DataLink (see page 20191)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	DataSource (see page 20192)	Links the SQL dataset to another (master) dataset.
	FieldNoOfs (see page 7022)	Specifies the offset needed to convert indexes in the Fields (see page 7023) property to field numbers.
R	InternalCalcFields (see page 7028)	Indicates whether the dataset includes internally calculated fields.
R	InternalConnection (see page 20194)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	LastError (see page 20194)	Saves the error message for the last error returned by dbExpress.
R	MasterLink (see page 20357)	This is MasterLink, a member of class TSQLTable.
	NativeCommand (see page 20195)	Represents the SQL command that is sent to the server.
	NestedDataSetClass (see page 7030)	Indicates the class type to use when creating nested datasets.
R	NestedDataSets (see page 7030)	Lists all nested datasets.
	ParamCheck (see page 20196)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
R	ParamCount (see page 20197)	Indicates the number of parameters listed by the Params (see page 20197) property.
	Params (see page 20197)	Represents the parameters for a query or stored procedure.
	ProcParams (see page 20199)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
	Reserved (see page 7035)	For internal use only.
R	RowsAffected (see page 20200)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 20200)	Identifies the metadata with which to populate the dataset.
	SortFieldNames (see page 20201)	Indicates the sort order when CommandType (see page 20190) is ctTable.

Public Events

	Name	Description
	AfterCancel (see page 7038)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 7040)	Occurs after an application closes a dataset.
	AfterDelete (see page 7041)	Occurs after an application deletes a record.
	AfterEdit (see page 7044)	Occurs after an application starts editing a record.
	AfterInsert (see page 7044)	Occurs after an application inserts a new record.
	AfterOpen (see page 7045)	Occurs after an application completes opening a dataset and before any data access occurs.

	AfterPost (see page 7046)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 7047)	Occurs after an application refreshes the data in the dataset.
	AfterScroll (see page 7047)	Occurs after an application scrolls from one record to another.
	BeforeCancel (see page 7048)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 7048)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 7049)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 7049)	Occurs before an application enters edit mode for the active record.
	BeforeInsert (see page 7050)	Occurs before an application enters insert mode.
	BeforeOpen (see page 7051)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 7052)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 7052)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeScroll (see page 7053)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 7053)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 7054)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 7054)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 7055)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 7056)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 7056)	Occurs when an application attempts to modify or insert a record and an exception is raised.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	ActiveBuffer (see page 6909)	Returns a pointer to the buffer for the active record.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 6910)	Adds a new, empty record to the end of the dataset.
	AppendRecord (see page 6911)	Adds a new, populated record to the end of the dataset and posts it.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BookmarkValid (see page 6916)	Indicates the validity of a specified bookmark.
V	Cancel (see page 6916)	Cancels modifications to the active record if those changes are not yet posted.
	CheckBrowseMode (see page 6918)	Automatically posts or cancels data changes when the active record changes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearFields (see page 6919)	Clears the contents of all fields for the active record.
	Close (see page 6920)	Closes a dataset.
V	CompareBookmarks (see page 6921)	Implements a virtual function to compare two bookmarks.
	ControlsDisabled (see page 6921)	Indicates whether data-aware controls update their display to reflect changes to the dataset.
	CopyFields (see page 6922)	This is CopyFields, a member of class TDataSet.
V	Create (see page 20349)	Creates an instance of a TSQLTable (see page 20337) component.
V	CreateBlobStream (see page 6922)	Provides the interface for a method that creates a blob stream for a Binary large object (BLOB) field in the dataset.
V	CreateBlobStream (see page 20164)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.

	CursorPosChanged (see page 6923)	Marks the internal cursor position as invalid.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 6924)	Deletes the active record and positions the dataset on the next record.
	DeleteRecords (see page 20350)	Empties the underlying database table.
	Destroy (see page 20350)	Destroys the TSQLTable (see page 20337) instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableControls (see page 6925)	Disables data display in data-aware controls associated with the dataset.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Edit (see page 6932)	Enables editing of data in the dataset.
	EnableControls (see page 6934)	Re-enables data display in data-aware controls associated with the dataset.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FieldByName (see page 6936)	Finds a field based on its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 6938)	Searches for a specified field in the dataset.
	FindFirst (see page 6939)	Implements a virtual method for positioning the dataset on the first record, respecting any filters.
	FindLast (see page 6940)	Implements a virtual method for positioning the dataset on the last record, respecting any filters.
	FindNext (see page 6940)	Implements a virtual method for positioning the dataset on the next record, respecting any filters.
	FindPrior (see page 6941)	Implements a virtual method for positioning the dataset on the previous record, respecting any filters.
	First (see page 6942)	Moves to the first record in the dataset.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeBookmark (see page 6945)	Frees the resources allocated for a specified bookmark.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetBlobFieldData (see page 6947)	Reads BLOB data into a buffer.
	GetBlobFieldData (see page 20166)	Retrieves the current value of a BLOB field into a buffer.
	GetBookmark (see page 6948)	Allocates a bookmark for the active record in the dataset.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentRecord (see page 6950)	Implements a virtual function to retrieve the current record in a buffer.
	GetDetailDataSets (see page 6951)	Fills a list with a dataset for every detail dataset that is not the value of a nested dataset field.
	GetDetailLinkFields (see page 6951)	Introduces a method to list the field components that link this dataset as a detail of a master dataset.
	GetDetailLinkFields (see page 20167)	Lists the field components that link this dataset as a detail of a master dataset.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldData (see page 6952)	This is the overview for the GetFieldData method overload.
	GetFieldData (see page 20168)	This is the overview for the GetFieldData method overload.
	GetFieldList (see page 6953)	Retrieves a specified set of field objects into a list.
	GetFieldNames (see page 6953)	This is the overview for the GetFieldNames method overload.
	GetIndexNames (see page 20351)	Retrieves a list of available indexes defined for the table.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetKeyFieldNames (see page 20169)	Fills a list with the names of all fields in all indexes defined for the dataset.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.

	GetQuoteChar (see page 20169)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	GotoBookmark (see page 6957)	Implements a virtual method to position the dataset on the record to which a specified bookmark points.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 6959)	Inserts a new, empty record in the dataset.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertRecord (see page 6961)	Inserts a new, populated record to the dataset and posts it.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmpty (see page 6967)	Indicates whether the dataset contains no records.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsLinkedTo (see page 6967)	Indicates whether a dataset is linked to a specified data source.
	IsSequenced (see page 6967)	Indicates whether the underlying database table uses record numbers to indicate the order of records.
	IsSequenced (see page 20172)	Indicates whether the dataset can use record numbers to indicate the order of records.
	Last (see page 6968)	Moves to the last record in the dataset.
	Locate (see page 6970)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Locate (see page 20173)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Lookup (see page 6970)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	Lookup (see page 20174)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MoveBy (see page 6971)	Moves to another record relative to the active record in the dataset.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Next (see page 6973)	Moves to the next record in the dataset.
	Open (see page 6976)	Opens the dataset.
	ParamByName (see page 20175)	This is ParamByName, a member of class TCustomSQLDataSet.
	Post (see page 6977)	Implements a virtual method to write a modified record to the database or change log.
	PrepareStatement (see page 20351)	Generates the query that TSQLTable (see page 20337) uses to fetch data from the database server.
	Prior (see page 6979)	Moves to the previous record in the dataset.
	PSEndTransaction (see page 7578)	Ends a transaction that was started by a call to PSStartTransaction (see page 7586).
	PSEndTransaction (see page 7589)	Ends a transaction that was started by a call to PSStartTransaction (see page 7597).
	PSExecute (see page 7578)	Executes the SQL command associated with the dataset.
	PSExecute (see page 7590)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 7579)	Executes a specified SQL command.
	PSExecuteStatement (see page 7590)	Executes a specified SQL command.
	PSGetAttributes (see page 7579)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetAttributes (see page 7591)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 7579)	Gets the SQL command to be executed by PSExecute (see page 7578).
	PSGetCommandText (see page 7591)	Gets the SQL command to be executed by PSExecute (see page 7590).
	PSGetCommandType (see page 7580)	Gets the command type to be executed by PSExecute (see page 7578).
	PSGetCommandType (see page 7591)	Gets the command type to be executed by PSExecute (see page 7590).
	PSGetDefaultOrder (see page 7580)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetDefaultOrder (see page 7592)	Returns the definition of an index that imposes the default order on the data included in data packets.

	PSGetIndexDefs (see page 7581)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetIndexDefs (see page 7592)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFields (see page 7581)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetKeyFields (see page 7593)	Returns the names of all fields required to uniquely identify records in the data packet.
	PSGetParams (see page 7582)	Returns the current parameter values of the dataset.
	PSGetParams (see page 7593)	Returns the current parameter values of the dataset.
	PSGetQuoteChar (see page 7582)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetQuoteChar (see page 7594)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	PSGetTableName (see page 7583)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetTableName (see page 7594)	Returns the name of the dataset table that appears in generated SQL statements.
	PSGetUpdateException (see page 7583)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSGetUpdateException (see page 7594)	Generates an EUpdateError (see page 6732) object based on another exception object.
	PSInTransaction (see page 7583)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSInTransaction (see page 7595)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 7584)	Indicates whether the dataset includes native SQL support.
	PSIsSQLBased (see page 7595)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 7584)	Indicates whether the dataset can execute SQL statements.
	PSIsSQLSupported (see page 7596)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 7585)	Resets the dataset so that it is positioned on the first record.
	PSReset (see page 7596)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 7585)	Assigns a specified SQL command to be executed by PSExecute (see page 7578).
	PSSetCommandText (see page 7596)	Assigns a specified SQL command to be executed by PSExecute (see page 7590).
	PSSetParams (see page 7586)	Assigns the specified parameter values to the dataset.
	PSSetParams (see page 7597)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 7586)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7578).
	PSStartTransaction (see page 7597)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction (see page 7589).
	PSUpdateRecord (see page 7586)	Applies a single update to the underlying database table or tables.
	PSUpdateRecord (see page 7598)	Applies a single update to the underlying database table or tables.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 6991)	Re-fetches data from the database to update a dataset's view of data.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Resync (see page 6994)	Re-fetches the active record and the records that precede and follow it.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetFields (see page 6997)	Sets the values for all fields in a record
	SetSchemaInfo (see page 20185)	Indicates whether the dataset represents metadata from the server, and if so, what kind.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
V	Translate (see page 7001)	Implements a virtual procedure to copy a data string from a source to a destination, optionally performing character translation during the copy.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateCursorPos (see page 7001)	Positions the cursor on the active record.

	UpdateRecord (see page 7002)	Ensures that data-aware controls and detail datasets reflect record updates.
V	UpdateStatus (see page 7002)	Reports the update status for the active record.

Public Properties

	Name	Description
R	AggFields (see page 7006)	Contains any aggregate fields.
	AutoCalcFields (see page 7006)	Determines when the OnCalcFields (see page 7053) event is triggered and when lookup field values are calculated.
	BlockReadSize (see page 7007)	Determines how many record buffers are read in each block.
R	Bof (see page 7008)	Indicates whether the first record in the dataset is active.
	Bookmark (see page 7009)	Specifies the current bookmark in the dataset.
R	CanModify (see page 7012)	Indicates whether the dataset permits write access to data.
	CommandText (see page 20189)	Specifies the command the dataset executes.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DataSetField (see page 7013)	Indicates the persistent TDataSetField (see page 7060) object that owns a nested dataset.
R	DataSource (see page 7014)	Represents the data source of another dataset that supplies values to the dataset.
R	DefaultFields (see page 7014)	Indicates whether a dataset's underlying field components are generated dynamically when the dataset is opened.
R	Designer (see page 7015)	Returns a pointer to the Dataset designer for the dataset.
	DesignerData (see page 20192)	Stores custom data.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Eof (see page 7015)	Indicates whether a dataset is positioned at the last record.
R	FieldCount (see page 7017)	Indicates the number of field components associated with the dataset.
R	FieldDefList (see page 7019)	Points to the list of field definitions for the dataset.
	FieldDefs (see page 7019)	Points to the list of field definitions for the dataset.
R	FieldList (see page 7022)	Lists the field components of a dataset.
R	Fields (see page 7023)	Lists all non-aggregate field components of the dataset.
	FieldValues (see page 7025)	Provides access to the values for all fields in the active record for the dataset
	Filter (see page 7026)	Specifies the text of the current filter for a dataset.
	Filtered (see page 7027)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 7027)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
R	Found (see page 7028)	Indicates whether or not moving to a different record is successful.
	IndexDefs (see page 20193)	Contains the definitions of all indexes defined on the server for the dataset.
R	IndexFieldCount (see page 20354)	Indicates the number of fields that make up the current index.
	IndexFields (see page 20355)	Specifies the fields associated with the current index.
R	IsUniDirectional (see page 7029)	Indicates whether the dataset is unidirectional.
R	Modified (see page 7029)	Indicates whether the active record is modified.
	ObjectView (see page 7031)	Specifies whether fields are stored hierarchically or flattened out in the Fields (see page 7023) property.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Prepared (see page 20198)	Specifies whether the command is prepared before execution.
	RecNo (see page 7031)	Indicates the active record in the dataset.
R	RecordCount (see page 7032)	Indicates the total number of records associated with the dataset.
R	RecordCount (see page 20199)	Indicates the total number of records associated with the dataset.
R	RecordSize (see page 7035)	Indicates the size of a record in the dataset.

	SparseArrays (see page 7036)	Determines whether a unique TField (see page 7115) object is created for each element of an array field.
R	State (see page 7036)	Indicates the current operating mode of the dataset.
	TransactionLevel (see page 20202)	Indicates the transaction to which this dataset belongs.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterClose (see page 20203)	Occurs after a data source connection is closed.
	AfterOpen (see page 20203)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 20203)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 20204)	Occurs after a result set is scrolled.
	BeforeClose (see page 20204)	Occurs just before a data connection is closed.
	BeforeOpen (see page 20204)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 20204)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 20204)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 20204)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 20205)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 20189) property.
	ParseInsertSql (see page 20205)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 20189) property.
	ParseSelectSql (see page 20205)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 20189) property.
	ParseUpdateSql (see page 20205)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 20189) property.

Published Properties

	Name	Description
	Active (see page 20353)	Specifies whether or not a dataset is open.
	GetMetadata (see page 20193)	Specifies whether the SQL dataset fetches metadata information along with the data.
	IndexFieldNames (see page 20355)	Lists the fields to use for sorting records and for linking in master/detail relationships.
	IndexName (see page 20356)	Identifies an index to use for sorting records and for linking the table as the detail of a master/detail relationship.
	MasterFields (see page 20356)	Identifies the fields from the master dataset to use for linking this table as the detail of a master/detail relationship.
	MasterSource (see page 20357)	Identifies the master dataset to use when linking this table as the detail of a master/detail relationship.
	MaxBlobSize (see page 20358)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NumericMapping (see page 20195)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 20196)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	SchemaName (see page 20201)	Specifies the name of the database schema.
	SQLConnection (see page 20358)	Specifies the SQL connection component that connects the dataset to a database server.
	TableName (see page 20359)	Specifies the table on the database server represented by this component.
	Tag (see page 2808)	Stores an integer value as part of a component.

149.1.11.2 TSQLTable.Create Constructor

Creates an instance of a TSQLTable (see page 20337) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSQLTable(TComponent * AOwner);
```

Description

Call Create to instantiate an SQL table at runtime. SQL tables placed in forms or data modules at design time are created automatically.

See Also

Create (see page 6904)

149.1.11.3 TSQLTable.Destroy Destructor

Destroys the TSQLTable (see page 20337) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSQLTable();
```

Description

Usually destruction of datasets is handled automatically by Delphi. If an application creates its own instances of a dataset, however, and does not assign an Owner that is responsible for freeing the dataset, then the application should call Free, which verifies that the dataset reference is not nil before calling Destroy.

See Also

Owner (see page 2807)

Create (see page 20349)

Free (see page 22300)

149.1.11.4 TSQLTable Methods

The methods of the TSQLTable class are listed here.

Protected Methods

	Name	Description
	OpenCursor (see page 20351)	This is OpenCursor, a member of class TSQLTable.
	SetIndexField (see page 20352)	This is SetIndexField, a member of class TSQLTable.

Public Methods

	Name	Description
	DeleteRecords (see page 20350)	Empties the underlying database table.
	GetIndexNames (see page 20351)	Retrieves a list of available indexes defined for the table.
	PrepareStatement (see page 20351)	Generates the query that TSQLTable (see page 20337) uses to fetch data from the database server.

149.1.11.4.1 TSQLTable.DeleteRecords Method

Empties the underlying database table.

Pascal

```
procedure DeleteRecords;
```

C++

```
__fastcall DeleteRecords();
```

Description

Call DeleteRecords to delete all the records in the underlying database table.

See Also

RecordCount (see page 7032)

149.1.11.4.2 TSQLTable.GetIndexNames Method

Retrieves a list of available indexes defined for the table.

Pascal

```
procedure GetIndexNames(List: TWideStrings);
```

C++

```
__fastcall GetIndexNames(TWideStrings List);
```

Description

Call GetIndexNames to retrieve a list of all indexes on the server defined for the table. List is a string list object, created and maintained by the application, into which to retrieve the index names.

See Also

IndexName (see page 20356)

IndexDefs (see page 20193)

Obtaining Information About Indexes

149.1.11.4.3 TSQLTable.OpenCursor Method

Pascal

```
procedure OpenCursor(InfoQuery: Boolean); override;
```

C++

```
virtual __fastcall OpenCursor(Boolean InfoQuery);
```

Description

This is OpenCursor, a member of class TSQLTable.

149.1.11.4.4 TSQLTable.PrepareStatement Method

Generates the query that TSQLTable (see page 20337) uses to fetch data from the database server.

Pascal

```
procedure PrepareStatement; override;
```

C++

```
virtual __fastcall PrepareStatement();
```

Description

PrepareStatement generates a SELECT query to fetch the records from the table specified by TableName (see page 20359), ordering by the fields in IndexFieldNames (see page 20355) or the index in IndexName (see page 20356). If you have specified a master/detail relationship, PrepareStatement sets that up as well.

Do not call PrepareStatement in your applications. Instead, set the Prepared (see page 20198) property to true. When you use the Prepared (see page 20198) property instead, the SQL table keeps track of the fact that it has already been prepared and need not repeat the process every time it executes.

See Also

Prepared (see page 20198)

TableName (see page 20359)

IndexName (see page 20356)

IndexFieldNames (see page 20355)

149.1.11.4.5 TSQLTable.SetIndexField Method

Pascal

```
procedure SetIndexField(Index: Integer; Value: TField);
```

C++

```
__fastcall SetIndexField(int Index, TField Value);
```

Description

This is SetIndexField, a member of class TSQLTable.

149.1.11.5 TSQLTable Properties

The properties of the TSQLTable class are listed here.

Protected Properties

	Name	Description
R	MasterLink (see page 20357)	This is MasterLink, a member of class TSQLTable.

Public Properties

	Name	Description
R	IndexFieldCount (see page 20354)	Indicates the number of fields that make up the current index.
	IndexFields (see page 20355)	Specifies the fields associated with the current index.

Published Properties

	Name	Description
	Active (see page 20353)	Specifies whether or not a dataset is open.
	IndexFieldNames (see page 20355)	Lists the fields to use for sorting records and for linking in master/detail relationships.
	IndexName (see page 20356)	Identifies an index to use for sorting records and for linking the table as the detail of a master/detail relationship.
	MasterFields (see page 20356)	Identifies the fields from the master dataset to use for linking this table as the detail of a master/detail relationship.
	MasterSource (see page 20357)	Identifies the master dataset to use when linking this table as the detail of a master/detail relationship.
	MaxBlobSize (see page 20358)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

	SQLConnection (see page 20358)	Specifies the SQL connection component that connects the dataset to a database server.
	TableName (see page 20359)	Specifies the table on the database server represented by this component.

149.1.11.5.1 TSQLTable.Active Property

Specifies whether or not a dataset is open.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Use Active to determine or set whether a dataset is populated with data. When Active is false, the dataset is closed; the dataset cannot read or write data and data-aware controls can not use it to fetch data or post edits. When Active is true, the dataset can be populated with data. It can read data from a database or other source (such as a provider). Depending on the CanModify property, active datasets can post changes.

Setting Active to true:

1. Generates a BeforeOpen event.
2. Sets the dataset state to dsBrowse.
3. Establishes a way to fetch data (typically by opening a cursor).
4. Generates an AfterOpen event.

If an error occurs while opening the dataset, dataset state is set to dsInactive, and any cursor is closed.

Setting Active to false:

1. Triggers a BeforeClose event.
2. Sets the State property to dsInactive.
3. Closes the cursor.
4. Triggers an AfterClose event.

An application must set Active to false before changing other properties that affect the status of a database or the controls that display data in an application.

Note: Calling the Open method sets Active to true; calling the Close method sets Active to false.

Delphi Examples:

```
{
This example requires a database with the following name or
in the form of a *.DB file in the following path.  The table
should be closed when changing the database.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Customers.Active := False;
  try
  { First try to use an alias }
    Customers.DatabaseName := 'DBDEMOS';
    Customers.Active := True;
  except
    on EDatabaseError do
```

```
      begin
        { If that fails, try to use the drive and directory }
        Customers.TableName := 'Flights';
        Customers.DatabaseName :=
          'c:\Documents and Settings\All Users\Documents\RAD
Studio\5.0\Demos\IntraWeb\Win32\FlightInformation\Data';
        Customers.Active := True;
      end;
    end;
  end;
```

See Also

DataSource (see page 7014)

State (see page 7036)

AfterOpen (see page 7045)

BeforeOpen (see page 7051)

Close (see page 6920)

Open (see page 6976)

CanModify (see page 7012)

BeforeClose (see page 7048)

AfterClose (see page 7040)

Using Datasets with Decision Support Components

Making the Table a Detail of Another Dataset

Opening and Closing Datasets

Fetching the Data

149.1.11.5.2 TSQLTable.IndexFieldCount Property

Indicates the number of fields that make up the current index.

Pascal

```
property IndexFieldCount: Integer;
```

C++

```
__property int IndexFieldCount;
```

Description

Check IndexFieldCount to determine how many fields are in the current index. If the IndexName (see page 20356) property is not set, IndexFieldCount is 0.

See Also

IndexFieldNames (see page 20355)

IndexDefs (see page 20193)

IndexName (see page 20356)

149.1.11.5.3 TSQLTable.IndexFieldNames Property

Lists the fields to use for sorting records and for linking in master/detail relationships.

Pascal

```
property IndexFieldNames: WideString;
```

C++

```
__property BSTR IndexFieldNames;
```

Description

IndexFieldNames is a semicolon-delimited set of field names, where each name specifies a field in the table.

Use IndexFieldNames to identify the fields in the SQL table that are used to sort records in the table. IndexFieldNames are used to generate the ORDER BY clause of the SQL statement automatically generated to fetch records for the dataset.

IndexFieldNames is also used to link this table as the detail in a master/detail relationship. Each of these fields must be matched to a corresponding field from the MasterFields (see page 20356) property.

Instead of explicitly setting the sorting and linking fields using IndexFieldNames, you can indicate them using an existing index. To specify the sorting and linking fields implicitly through an index, use the IndexName (see page 20356) property instead.

Note: The IndexName (see page 20356) and IndexFieldNames properties are mutually exclusive. Setting one automatically clears the other.

See Also

IndexName (see page 20356)

MasterFields (see page 20356)

MasterSource (see page 20357)

Creating an Index with IndexFieldNames

Representing the Records in a Table

149.1.11.5.4 TSQLTable.IndexFields Property

Specifies the fields associated with the current index.

Pascal

```
property IndexFields [Index: Integer]: TField;
```

C++

```
__property TField IndexFields[int Index];
```

Description

IndexFields is a zero-based array of field objects, each of which corresponds to a field in the current index. Index is an ordinal value indicating the position of a field in the index. The first field in the index is IndexFields[0], the second is IndexFields[1], and so on.

Note: Do not set IndexFields directly. Instead use the IndexFieldNames (see page 20355) property to specify the sort order for records in the table.

See Also

IndexDefs (see page 20193)

IndexFieldNames (see page 20355)

IndexName (see page 20356)

149.1.11.5.5 TSQLTable.IndexName Property

Identifies an index to use for sorting records and for linking the table as the detail of a master/detail relationship.

Pascal

```
property IndexName: WideString;
```

C++

```
__property BSTR IndexName;
```

Description

Use IndexName to specify an existing index that is defined for the underlying database table.

IndexName is used to sort records in the table by appearing in the ORDER BY clause of the SQL statement automatically generated to fetch records for the dataset.

IndexName is also used to link this table as the detail in a master/detail relationship. Each of the fields in the index must be matched to a corresponding field from the MasterFields (see page 20356) property.

Note: The IndexName and IndexFieldNames (see page 20355) properties are mutually exclusive. Setting one automatically clears the other.

See Also

IndexFieldNames (see page 20355)

MasterFields (see page 20356)

MasterSource (see page 20357)

GetIndexNames (see page 20351)

Representing the Records in a Table

149.1.11.5.6 TSQLTable.MasterFields Property

Identifies the fields from the master dataset to use for linking this table as the detail of a master/detail relationship.

Pascal

```
property MasterFields: WideString;
```

C++

```
__property BSTR MasterFields;
```

Description

Use MasterFields to identify the fields in another (master) dataset that can be used as link fields in a master/detail relationship. MasterFields is a semicolon-delimited set of field names, where each name identifies a field in the master dataset. The master dataset is the dataset associated with the datasource specified by MasterSource (see page 20357). Each of the fields in MasterFields must be matched to a corresponding field from the IndexFieldNames (see page 20355) property.

At design-time, double-click on the MasterFields property in the Object Inspector to use the Field Link designer to help you define the master/detail relationship. You can only use the Field Link designer if you have already specified the IndexName (see page 20356) or IndexFieldNames (see page 20355) property and the MasterSource (see page 20357) property.

See Also

IndexFieldNames (see page 20355)

IndexName (see page 20356)

MasterSource (see page 20357)

Making the Table a Detail of Another Dataset

Setting Up Master/detail Linked Cursors

149.1.11.5.7 TSQLTable.MasterLink Property

Pascal

```
property MasterLink: TMasterDataLink;
```

C++

```
__property TMasterDataLink MasterLink;
```

Description

This is MasterLink, a member of class TSQLTable.

149.1.11.5.8 TSQLTable.MasterSource Property

Identifies the master dataset to use when linking this table as the detail of a master/detail relationship.

Pascal

```
property MasterSource: TDataSource;
```

C++

```
__property TDataSource MasterSource;
```

Description

Use MasterSource to indicate the dataset that serves as a master table when establishing a master/detail relationship. Specify a TDataSource component whose DataSet property identifies the desired master dataset.

A master/detail relationship is established when you set the MasterSource property to identify a master dataset, the MasterFields (see page 20356) property to identify the fields in that master dataset whose values must be matched by linking fields in this (detail) SQL table, and the IndexName (see page 20356) or IndexFieldNames (see page 20355) property to identify the linking fields in this SQL table.

MasterSource is the same as the DataSource (see page 20192) property, except that it is published.

See Also

IndexFieldNames (see page 20355)

IndexName (see page 20356)

MasterFields (see page 20356)

DataSource (see page 20192)

DataSet (see page 7078)

Making the Table a Detail of Another Dataset

Setting Up Master/detail Linked Cursors

149.1.11.5.9 TSQLTable.MaxBlobSize Property

Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

MaxBlobSize limits the amount of data that can be fetched from Binary Large Object (BLOB) fields. BLOB fields can hold an indefinite amount of data. However, SQL datasets read that data into a memory buffer. By limiting the amount of data that can be fetched from a BLOB field, you limit the size of the buffer.

If MaxBlobSize is –1, the dataset imposes no maximum BLOB size. If MaxBlobSize is 0, the dataset uses the parameters of the associated SQL connection, if any, to determine the maximum BLOB size.

Note: MaxBlobSize should agree with any parameters on the associated SQL connection component that limit the size of BLOB data.

See Also

GetBlobFieldData (see page 20166)

Params (see page 20253)

SQLConnection (see page 20201)

149.1.11.5.10 TSQLTable.SQLConnection Property

Specifies the SQL connection component that connects the dataset to a database server.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

Use SQLConnection to specify an SQL connection object that connects the dataset to a database server.

At design-time, select from the list of available TSQLConnection components in the Object Inspector. At runtime, set SQLConnection to reference an existing TSQLConnection object.

SQLConnection should be the first property you set on the dataset. Only after the SQLConnection property is set can you use the Command Text editor to specify a query for the CommandText property or populate the dataset with data by setting Active to true. Similarly, SQLConnection must be set before the dataset can initialize the Params property when you set CommandText.

See Also

Active (see page 7004)

CommandText (see page 20189)

Params (see page 20197)

Using Query-type Datasets

Using Stored Procedure-type Datasets

Using Table Type Datasets

Connecting Directly to a Database Server

Connecting to the Database Server

149.1.11.5.11 TSQLTable.TableName Property

Specifies the table on the database server represented by this component.

Pascal

```
property TableName: WideString;
```

C++

```
__property BSTR TableName;
```

Description

Set TableName to the name of a table on the database server. TSQLTable (see page 20337) automatically generates an SQL query to fetch all of the records and all of the fields in that table.

Note: In the automatically generated SQL query, the ORDER BY clause is determined by the IndexName (see page 20356) or IndexFieldNames (see page 20355) property.

See Also

IndexFieldNames (see page 20355)

IndexName (see page 20356)

Representing the Records in a Table

149.1.12 TTransactionItem Class

Class Hierarchy

SqlExpr.TTransactionItem

Pascal

```
TTransactionItem = class;
```

C++

```
class TTransactionItem;
```

File

SqlExpr

Description

This is class SqlExpr.TTransactionItem.

149.1.12.1 TTransactionItem Members

The following tables list the members exposed by TTransactionItem.

Public Fields

	Name	Description
	FNext (see page 20360)	This is FNext, a member of class TTransactionItem.
	FTransaction (see page 20360)	This is FTransaction, a member of class TTransactionItem.
	FTransactionDesc (see page 20360)	This is FTransactionDesc, a member of class TTransactionItem.

149.1.12.2 TTransactionItem Fields

The fields of the TTransactionItem class are listed here.

Public Fields

	Name	Description
	FNext (see page 20360)	This is FNext, a member of class TTransactionItem.
	FTransaction (see page 20360)	This is FTransaction, a member of class TTransactionItem.
	FTransactionDesc (see page 20360)	This is FTransactionDesc, a member of class TTransactionItem.

149.1.12.2.1 TTransactionItem.FNext Field

Pascal

```
FNext: TTransactionItem;
```

C++

```
TTransactionItem FNext;
```

Description

This is FNext, a member of class TTransactionItem.

149.1.12.2.2 TTransactionItem.FTransaction Field

Pascal

```
FTransaction: TDBXTransaction;
```

C++

```
TDBXTransaction FTransaction;
```

Description

This is FTransaction, a member of class TTransactionItem.

149.1.12.2.3 TTransactionItem.FTransactionDesc Field

Pascal

```
FTransactionDesc: TTransactionDesc;
```

C++

```
TTransactionDesc FTransactionDesc;
```

Description

This is FTransactionDesc, a member of class TTransactionItem.

149.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FreeProcParams (see page 20361)	Frees a list of procedure parameters that is allocated by GetProcedureParams.
	GetConnectionNames (see page 20361)	This is function SqlExpr.GetConnectionNames.
	GetConnectionRegistryFile (see page 20362)	This is function SqlExpr.GetConnectionRegistryFile.
	GetDriverNames (see page 20362)	Populates a string list with the names of all available dbExpress drivers.
	GetDriverRegistryFile (see page 20363)	This is function SqlExpr.GetDriverRegistryFile.
	LoadParamListItems (see page 20363)	Fills a TParams object with information from a list of parameter descriptors.

149.2.1 SqlExpr.FreeProcParams Function

Frees a list of procedure parameters that is allocated by GetProcedureParams.

Pascal

```
procedure FreeProcParams(var ProcParams: TList);
```

C++

```
FreeProcParams(TList ProcParams);
```

File

SqlExpr

Description

Use FreeProcParams to free a list of stored procedure parameters after using them. FreeProcParams takes a single parameter, ProcParams, which is a list of parameter information that was returned from a call to the GetProcedureParams method of TSQLConnection (see page 20217). It frees all of the parameter structures in the list, and then frees the list itself, changing the parameter that was passed in to nil (Delphi) or NULL (C++).

See Also

TSQLConnection (see page 20217)

LoadParamListItems (see page 20363)

149.2.2 SqlExpr.GetConnectionNames Function

Pascal

```
procedure GetConnectionNames(List: TStrings; Driver: string = ''; DesignMode: Boolean = True);
```

C++

```
GetConnectionNames(TStrings List, AnsiString Driver = '', Boolean DesignMode = True);
```

File

SqlExpr

Description

This is function SqlExpr.GetConnectionNames.

149.2.3 SqlExpr.GetConnectionRegistryFile Function

Pascal

```
function GetConnectionRegistryFile(DesignMode: Boolean = False): string;
```

C++

```
AnsiString GetConnectionRegistryFile(Boolean DesignMode = False);
```

File

SqlExpr

Description

This is function SqlExpr.GetConnectionRegistryFile.

149.2.4 SqlExpr.GetDriverNames Function

Populates a string list with the names of all available dbExpress drivers.

Pascal

```
procedure GetDriverNames(List: TStrings; DesignMode: Boolean = True);
```

C++

```
GetDriverNames(TStrings List, Boolean DesignMode = True);
```

File

SqlExpr

Description

GetDriverNames fills the string list specified by List with the names of all installed dbExpress drivers. GetDriverNames fetches the names of Installed drivers from the drivers.ini file.

List is a string list object, created and maintained by the application, into which to store the driver names.

DesignMode indicates whether the names are fetched at design time (true) or at runtime (false).

See Also

TSQLConnection (see page 20217)

149.2.5 SqlExpr.GetDriverRegistryFile Function

Pascal

```
function GetDriverRegistryFile(DesignMode: Boolean = False): string;
```

C++

```
AnsiString GetDriverRegistryFile(Boolean DesignMode = False);
```

File

SqlExpr

Description

This is function SqlExpr.GetDriverRegistryFile.

149.2.6 SqlExpr.LoadParamListItems Function

Fills a TParams object with information from a list of parameter descriptors.

Pascal

```
procedure LoadParamListItems(Params: TParams; ProcParams: TList);
```

C++

```
LoadParamListItems(TParams Params, TList ProcParams);
```

File

SqlExpr

Description

Call LoadParamListItems to convert the list of parameter descriptors that is returned from the GetProcedureParams method of TSQLConnection (see page 20217) to a collection of TParam objects.

Params is an empty TParams collection that the procedure fills with information from the list of parameter descriptors.

ProcParams is the list of parameter descriptors that GetProcedureParams returns.

See Also

FreeProcParams (see page 20361)

TSQLConnection (see page 20217)

149.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	EConnectFlag (see page 20364)	This is record SqlExpr.EConnectFlag.
	TConnectionState (see page 20365)	TConnectionState describes the state of a TSQLConnection (see page 20217) component.
	TConnectionUserType (see page 20365)	This is record SqlExpr.TConnectionUserType.
	TSchemaType (see page 20366)	TSchemaType and TSchemaInfo describe schema information (metadata) to access using dbXpress.
	TSQLExceptionType (see page 20367)	TSQLExceptionType indicates the type of operation that leads to an ESQLException exception.
	TTableScope (see page 20368)	TTableScopes (see page 20374) and TTableScopess indicate the types of tables for which to fetch metadata.
	TTransIsolationLevel (see page 20369)	This enumerated type (in C++) or type (in Delphi) identifies degree to which transactions are isolated from each other.

Records

	Name	Description
	TSQLSchemaInfo (see page 20368)	This is record SqlExpr.TSQLSchemaInfo.
	TTransactionDesc (see page 20369)	This is record SqlExpr.TTransactionDesc.

149.3.1 SqlExpr.EConnectFlag Enumeration

Pascal

```
EConnectFlag = (
  eConnect,
  eReconnect,
  eDisconnect
);
```

C++

```
enum EConnectFlag {
  eConnect,
  eReconnect,
  eDisconnect
};
```

File

SqlExpr

Description

This is record SqlExpr.EConnectFlag.

149.3.2 SqlExpr.TConnectionState Enumeration

TConnectionState describes the state of a TSQLConnection (see page 20217) component.

Pascal

```
TConnectionState = (
  csStateClosed,
  csStateOpen,
  csStateConnecting,
  csStateExecuting,
  csStateFetching,
  csStateDisconnecting
);
```

C++

```
enum TConnectionState {
  csStateClosed,
  csStateOpen,
  csStateConnecting,
  csStateExecuting,
  csStateFetching,
  csStateDisconnecting
};
```

File

SqlExpr

Description

TConnectionState can have any of the following values:

Value	Meaning
csStateClosed	The connection is closed. (The Connected property is false)
csStateOpen	The connection is open. (The Connected property is true)
csStateConnecting	The component is establishing a connection but has not completed the process.
csStateExecuting	The component has passed an SQL command to the server and it is currently executing.
csStateFetching	The component is fetching database information from the server.
csStateDisconnecting	The component is terminating a connection but has not completed the process.

149.3.3 SqlExpr.TConnectionUserType Enumeration

Pascal

```
TConnectionUserType = (
  eUserMonitor,
  eUserDataSet
);
```

C++

```
enum TConnectionUserType {
  eUserMonitor,
```

```
  eUserDataSet
};
```

File

SqlExpr

Description

This is record SqlExpr.TConnectionUserType.

149.3.4 SqlExpr.TSchemaType Enumeration

TSchemaType and TSchemaInfo describe schema information (metadata) to access using dbXpress.

Pascal

```
TSchemaType = (
  stNoSchema,
  stTables,
  stSysTables,
  stProcedures,
  stColumns,
  stProcedureParams,
  stIndexes,
  stPackages,
  stUserNames
);
```

C++

```
enum TSchemaType {
  stNoSchema,
  stTables,
  stSysTables,
  stProcedures,
  stColumns,
  stProcedureParams,
  stIndexes,
  stPackages,
  stUserNames
};
```

File

SqlExpr

Description

Use TSchemaInfo to describe the metadata you want to fetch from a database server when you are using an SQL dataset. TSchemaInfo has three members: FType, ObjectName, and Pattern, which are described below.

Use TSchemaInfo to describe the metadata you want to fetch from a database server when you are using an SQL dataset. TSchemaInfo has four members: FType, ObjectName, Pattern, and PackageName, which are described below.

FType is of type TSchemaType. It identifies the type of information you want. It can have any of the following values:

Value	Description
stNoSchema	No schema information. When FType is stNoSchema, the SQL dataset is populated with the results of its query or stored procedure rather than metadata from the server.
stTables	Information about all the data tables on the database server that match the criteria specified by the SQL connection's TableScope property.

stSysTables	Information about all of the system tables on the database server. Not all servers use system tables to store metadata. Requesting a list of system tables from a server that does not use them results in an empty dataset.
stProcedures	Information about all of the stored procedures on the database server.
stColumns	Information about all of the columns (fields) in a specified table.
stProcedureParams	Information about all of the parameters of a specified stored procedure.
stIndexes	Information about all of the indexes defined for a specified table.
stPackages	Information about all of the packages defined on the database server (Oracle only).

ObjectName is the name of the table or stored procedure about whose columns, indexes, or parameters you want information. When FType is stColumns or stIndexes, ObjectName is the name of a table. When FType is stProcedureParams, ObjectName is the name of a stored procedure. When FType is stNone, stTables, stSysTables, or stProcedures, ObjectName is ignored.

Pattern is an SQL pattern mask that limits the items returned. The name of each item (table, stored procedure, field, parameter, or index) must match this pattern mask or it does not appear as a record when the SQL dataset is opened. Pattern uses the wildcards '%' (to match a string of arbitrary characters of any length) and '_' (to match a single arbitrary character). To use a literal percent or underscore in a pattern, the character is doubled (%% or __). If you do not want to use a pattern, set Pattern to nil (Delphi) or NULL (C++).

PackageName is the name of an Oracle package that contains the stored procedure about which you are requesting information. This member is used when FType is stProcedures or stProcedureParams. PackageName is ignored if the server is not an Oracle database.

149.3.5 SqlExpr.TSQLExceptionType Enumeration

TSQLExceptionType indicates the type of operation that leads to an ESQLException exception.

Pascal

```
TSQLExceptionType = (
  exceptConnection,
  exceptCommand,
  exceptCursor,
  exceptMetaData,
  exceptUseLast
);
```

C++

```
enum TSQLExceptionType {
  exceptConnection,
  exceptCommand,
  exceptCursor,
  exceptMetaData,
  exceptUseLast
};
```

File

SqlExpr

Description

ESQLException objects use TSQLExceptionType to identify the dbExpress object from which an error code originated. By identifying the underlying dbExpress object, ESQLException identifies that object's interface, which it then uses to obtain an error message.

TSQLExceptionType includes the following values:

Value	Description
exceptConnection	Error arose when establishing a connection to the database server, obtaining an interface for sending a command, fetching server metadata, or controlling a transaction.
exceptCommand	Error occurred while preparing or executing an SQL command.
exceptCursor	Error occurred while fetching data from a dbExpress cursor.
exceptMetaData	Error occurred while trying to fetch metadata (schema information).
exceptUseLast	The error is cached by the TSQLConnection (see page 20217) component.

149.3.6 SqlExpr.TSQLSchemaInfo Record

Pascal

```
TSQLSchemaInfo = record
  FType: TSchemaType;
  ObjectName: WideString;
  Pattern: WideString;
  PackageName: WideString;
end;
```

C++

```
struct TSQLSchemaInfo {
  TSchemaType FType;
  BSTR ObjectName;
  BSTR Pattern;
  BSTR PackageName;
};
```

File

SqlExpr

Description

This is record SqlExpr.TSQLSchemaInfo.

149.3.7 SqlExpr.TTableScope Enumeration

TTableScopes (see page 20374) and TTableScopess indicate the types of tables for which to fetch metadata.

Pascal

```
TTableScope = (
  tsSynonym,
  tsSysTable,
  tsTable,
  tsView
);
```

C++

```
enum TTableScope {
  tsSynonym,
  tsSysTable,
```

```
    tsTable,
    tsView
  };
```

File

SqlExpr

Description

TTableScopess is a set that indicates the types of tables for which to fetch metadata. It is a set of TTableScopes (see page 20374) values. The following table describes the possible values:

Value	Meaning
tsSynonym	synonyms
tsSysTable	system tables
tsTable	ordinary tables
tsView	views

149.3.8 SqlExpr.TTransactionDesc Record

Pascal

```
  TTransactionDesc = packed record
    TransactionID: LongWord;
    GlobalID: LongWord;
    IsolationLevel: TTransIsolationLevel;
    CustomIsolation: LongWord;
  end;
```

C++

```
  struct TTransactionDesc {
    LongWord TransactionID;
    LongWord GlobalID;
    TTransIsolationLevel IsolationLevel;
    LongWord CustomIsolation;
  };
```

File

SqlExpr

Description

This is record SqlExpr.TTransactionDesc.

149.3.9 SqlExpr.TTransIsolationLevel Enumeration

This enumerated type (in C++) or type (in Delphi) identifies degree to which transactions are isolated from each other.

Pascal

```
  TTransIsolationLevel = (
    xilREADCOMMITTED,
    xilREPEATABLEREAD,
    xilDIRTYREAD,
```

```
    xilCUSTOM
  );
```

C++

```
  enum TTransIsolationLevel {
    xilREADCOMMITTED,
    xilREPEATABLEREAD,
    xilDIRTYREAD,
    xilCUSTOM
  };
```

File

SqlExpr

Description

The following table describes the possible values.

Element	Description
xilDIRTYREAD	Current transaction sees all changes made by other transactions, even if they have not yet been committed. This option is not available for the Oracle driver.
xilREADCOMMITTED	Current transaction sees only changes that have been committed, but can receive an inconsistent view of the data if additional changes are committed before the transaction ends.
xilREPEATABLEREAD	Current transaction is guaranteed a consistent view of the data, which includes only changes committed by other transactions at the start of the transaction.
xilCUSTOM	Current transaction uses a database-specific isolation level. When the eConnTxnIsoLevel option is xilCUSTOM, the transaction descriptor gives the actual isolation level. This is not currently supported by any drivers, but allows future drivers the ability to expand the number of isolation levels.

149.4 Types

The following table lists types in this documentation.

Types

Name	Description
TFieldDescList (see page 20371)	This is type SqlExpr.TFieldDescList.
TFieldList (see page 20371)	This is type SqlExpr.TFieldList.
TGetDriverFunc (see page 20372)	This is type SqlExpr.TGetDriverFunc.
TLocale (see page 20372)	This is type SqlExpr.TLocale.
TLocaleCode (see page 20372)	This is type SqlExpr.TLocaleCode.
TParseInsertSqlEvent (see page 20373)	This is type SqlExpr.TParseInsertSqlEvent.
TParseSqlEvent (see page 20373)	This is type SqlExpr.TParseSqlEvent.
TSQLConnectionLoginEvent (see page 20373)	This is type SqlExpr.TSQLConnectionLoginEvent.
TTableScopes (see page 20374)	This is type SqlExpr.TTableScopes.
TTraceEvent (see page 20374)	This is type SqlExpr.TTraceEvent.
TTraceLogEvent (see page 20374)	This is type SqlExpr.TTraceLogEvent.

149.4.1 SqlExpr.TFieldDescList Type

Pascal

```
TFieldDescList = array of TFLDDesc;
```

C++

```
array of TFLDDesc TFieldDescList;
```

File

SqlExpr

Description

This is type SqlExpr.TFieldDescList.

149.4.2 SqlExpr.TFieldList Type

Pascal

```
TFieldList = TList;
```

C++

```
TList TFieldList;
```

File

SqlExpr

Description

This is type SqlExpr.TFieldList.

149.4.3 SqlExpr.TGetDriverFunc Type

Pascal

```
TGetDriverFunc = function (SVendorLib, SResourceFile: PChar; out Obj): TDBXErrorCode;
```

C++

```
(SVendorLib, SResourceFile: PChar; out Obj): TDBXErrorCode ( TGetDriverFunc)();
```

File

SqlExpr

Description

This is type SqlExpr.TGetDriverFunc.

149.4.4 SqlExpr.TLocale Type

Pascal

```
TLocale = Pointer;
```

C++

```
void * TLocale;
```

File

SqlExpr

Description

This is type SqlExpr.TLocale.

149.4.5 SqlExpr.TLocaleCode Type

Pascal

```
TLocaleCode = Integer;
```

C++

```
int TLocaleCode;
```

File

SqlExpr

Description

This is type SqlExpr.TLocaleCode.

149.4.6 SqlExpr.TParseInsertSqlEvent Type

Pascal

```
TParseInsertSqlEvent = procedure (var FieldNames: TWideStrings; SQL: WideString; var
BindAllFields: Boolean; var TableName: WideString) of object;
```

C++

```
(var FieldNames: TWideStrings; SQL: WideString; var BindAllFields: Boolean; var TableName:
WideString) ( TParseInsertSqlEvent)();
```

File

SqlExpr

Description

This is type SqlExpr.TParseInsertSqlEvent.

149.4.7 SqlExpr.TParseSqlEvent Type

Pascal

```
TParseSqlEvent = procedure (var FieldNames: TWideStrings; SQL: WideString; var TableName:
WideString) of object;
```

C++

```
(var FieldNames: TWideStrings; SQL: WideString; var TableName: WideString) ( TParseSqlEvent)();
```

File

SqlExpr

Description

This is type SqlExpr.TParseSqlEvent.

149.4.8 SqlExpr.TSQLConnectionLoginEvent Type

Pascal

```
TSQLConnectionLoginEvent = procedure (Database: TSQLConnection; LoginParams: TWideStrings) of
object;
```

C++

```
(Database: TSQLConnection; LoginParams: TWideStrings) ( TSQLConnectionLoginEvent)();
```

File

SqlExpr

Description

This is type SqlExpr.TSQLConnectionLoginEvent.

149.4.9 SqlExpr.TTableScopes Type

Pascal

```
TTableScopes = set of TTableScope;
```

C++

```
TTableScope TTableScopes;
```

File

SqlExpr

Description

This is type SqlExpr.TTableScopes.

149.4.10 SqlExpr.TTraceEvent Type

Pascal

```
TTraceEvent = procedure (Sender: TObject; TraceInfo: TDBXTraceInfo; var LogTrace: Boolean) of
object;
```

C++

```
(Sender: TObject; TraceInfo: TDBXTraceInfo; var LogTrace: Boolean) ( TTraceEvent)();
```

File

SqlExpr

Description

This is type SqlExpr.TTraceEvent.

149.4.11 SqlExpr.TTraceLogEvent Type

Pascal

```
TTraceLogEvent = procedure (Sender: TObject; TraceInfo: TDBXTraceInfo) of object;
```

C++

```
(Sender: TObject; TraceInfo: TDBXTraceInfo) ( TTraceLogEvent)();
```

File

SqlExpr

Description

This is type SqlExpr.TTraceLogEvent.

149.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
BlobTypeMap (see page 20375)	This is constant SqlExpr.BlobTypeMap.
DataTypeMap (see page 20376)	This is constant SqlExpr.DataTypeMap.
DefaultCursor (see page 20376)	This is constant SqlExpr.DefaultCursor.
DefaultMaxBlobSize (see page 20376)	This is constant SqlExpr.DefaultMaxBlobSize.
DefaultRowsetSize (see page 20377)	This is constant SqlExpr.DefaultRowsetSize.
FldSubTypeMap (see page 20377)	This is constant SqlExpr.FldSubTypeMap.
FldTypeMap (see page 20377)	This is constant SqlExpr.FldTypeMap.
HourGlassCursor (see page 20378)	This is constant SqlExpr.HourGlassCursor.
SAnd (see page 20378)	This is constant SqlExpr.SAnd.
SOrderBy (see page 20379)	This is constant SqlExpr.SOrderBy.
SParam (see page 20379)	This is constant SqlExpr.SParam.
SSelect (see page 20379)	This is constant SqlExpr.SSelect.
SSelectSpaces (see page 20380)	This is constant SqlExpr.SSelectSpaces.
SSelectStar (see page 20380)	This is constant SqlExpr.SSelectStar.
SSelectStarFrom (see page 20380)	This is constant SqlExpr.SSelectStarFrom.
SUB_TYPE_MEMO (see page 20381)	This is constant SqlExpr.SUB_TYPE_MEMO.
SWhere (see page 20381)	This is constant SqlExpr.SWhere.
TErrorMessageSize (see page 20381)	This is constant SqlExpr.TErrorMessageSize.

149.5.1 SqlExpr.BlobTypeMap Constant

Pascal

```
BlobTypeMap: array[SUB_TYPE_MEMO..TDBXTypes.BFileSubType] of TFieldType = ( ftMemo, ftBlob,
ftFmtMemo, ftParadoxOle, ftGraphic, ftDBaseOle, ftTypedBinary, ftBlob, ftBlob, ftBlob,
ftWideMemo, ftOraClob, ftOraBlob, ftBlob, ftBlob);
```

C++

```
array[SUB_TYPE_MEMO..TDBXTypes.BFileSubType] of TFieldType BlobTypeMap = ( ftMemo, ftBlob,
ftFmtMemo, ftParadoxOle, ftGraphic, ftDBaseOle, ftTypedBinary, ftBlob, ftBlob, ftBlob,
ftWideMemo, ftOraClob, ftOraBlob, ftBlob, ftBlob);
```

File

SqlExpr

Description

This is constant SqlExpr.BlobTypeMap.

149.5.2 SqlExpr.DataTypeMap Constant

Pascal

```
DataTypeMap: array[0..TDBXTypes.MaxBaseTypes - 1] of TFieldType = ( ftUnknown, ftString,
ftDate, ftBlob, ftBoolean, ftSmallint, ftInteger, ftFloat, ftFMTBCD, ftBytes, ftTime,
ftDateTime, ftWord, ftInteger, ftUnknown, ftVarBytes, ftUnknown, ftCursor, ftLargeInt,
ftLargeInt, ftADT, ftArray, ftReference, ftDataSet, ftTimeStamp, ftFMTBCD, ftWideString);
```

C++

```
array[0..TDBXTypes.MaxBaseTypes - 1] of TFieldType DataTypeMap = ( ftUnknown, ftString,
ftDate, ftBlob, ftBoolean, ftSmallint, ftInteger, ftFloat, ftFMTBCD, ftBytes, ftTime,
ftDateTime, ftWord, ftInteger, ftUnknown, ftVarBytes, ftUnknown, ftCursor, ftLargeInt,
ftLargeInt, ftADT, ftArray, ftReference, ftDataSet, ftTimeStamp, ftFMTBCD, ftWideString);
```

File

SqlExpr

Description

This is constant SqlExpr.DataTypeMap.

149.5.3 SqlExpr.DefaultCursor Constant

Pascal

```
DefaultCursor = 0;
```

C++

```
DefaultCursor = 0;
```

File

SqlExpr

Description

This is constant SqlExpr.DefaultCursor.

149.5.4 SqlExpr.DefaultMaxBlobSize Constant

Pascal

```
DefaultMaxBlobSize = -1;
```

C++

```
DefaultMaxBlobSize = -1;
```

File

SqlExpr

Description

This is constant SqlExpr.DefaultMaxBlobSize.

149.5.5 SqlExpr.DefaultRowsetSize Constant

Pascal

```
DefaultRowsetSize = 20;
```

C++

```
DefaultRowsetSize = 20;
```

File

SqlExpr

Description

This is constant SqlExpr.DefaultRowsetSize.

149.5.6 SqlExpr.FldSubTypeMap Constant

Pascal

```
FldSubTypeMap: array[TFieldType] of Word = ( 0, 0, 0, 0, 0, 0, 0, TDBXTypes.MoneySubType, 0,
0, 0, 0, 0, 0, TDBXTypes.AutoIncSubType, TDBXTypes.BinarySubType, TDBXTypes.MemoSubType, 0, 0,
0, 0, 0, 0, TDBXTypes.FixedSubType, 0, 0, 0, 0, 0, 0, TDBXTypes.HBinarySubType,
TDBXTypes.HMemoSubType, 0, 0, 0, 0, 0, 0, TDBXTypes.FixedSubType, TDBXTypes.WideMemoSubType,
TDBXTypes.OracleTimeStampSubType, TDBXTypes.OracleIntervalSubType);
```

C++

```
array[TFieldType] of Word FldSubTypeMap = ( 0, 0, 0, 0, 0, 0, 0, TDBXTypes.MoneySubType, 0, 0,
0, 0, 0, 0, TDBXTypes.AutoIncSubType, TDBXTypes.BinarySubType, TDBXTypes.MemoSubType, 0, 0, 0,
0, 0, 0, TDBXTypes.FixedSubType, 0, 0, 0, 0, 0, 0, TDBXTypes.HBinarySubType,
TDBXTypes.HMemoSubType, 0, 0, 0, 0, 0, 0, TDBXTypes.FixedSubType, TDBXTypes.WideMemoSubType,
TDBXTypes.OracleTimeStampSubType, TDBXTypes.OracleIntervalSubType);
```

File

SqlExpr

Description

This is constant SqlExpr.FldSubTypeMap.

149.5.7 SqlExpr.FldTypeMap Constant

Pascal

```
FldTypeMap: TFieldMap = ( TDBXTypes.UnknownType, TDBXTypes.ZStringType, TDBXTypes.Int16Type,
TDBXTypes.Int32Type, TDBXTypes.UInt16Type, TDBXTypes.BoolType, TDBXTypes.FloatType,
TDBXTypes.FloatType, TDBXTypes.BcdType, TDBXTypes.DateType, TDBXTypes.TimeType, 0,
TDBXTypes.BytesType, TDBXTypes.VarBytesType, TDBXTypes.Int32Type, TDBXTypes.BlobType,
TDBXTypes.BlobType, TDBXTypes.BlobType, TDBXTypes.BlobType, TDBXTypes.BlobType,
TDBXTypes.BlobType, TDBXTypes.BlobType, TDBXTypes.CursorType, TDBXTypes.ZStringType,
TDBXTypes.WideStringType, TDBXTypes.Int64Type, TDBXTypes.AdtType, TDBXTypes.ArrayType,
TDBXTypes.RefType, TDBXTypes.TableType, TDBXTypes.BlobType, TDBXTypes.BlobType,
TDBXTypes.UnknownType, TDBXTypes.UnknownType, TDBXTypes.UnknownType, TDBXTypes.ZStringType,
```

```
TDBXTypes.TimeStampType, TDBXTypes.BcdType, TDBXTypes.WideStringType, TDBXTypes.BlobType,
TDBXTypes.TimeStampType, TDBXTypes.ZStringType);
```

C++

```
TFieldMap FldTypeMap = ( TDBXTypes.UnknownType, TDBXTypes.ZStringType, TDBXTypes.Int16Type,
TDBXTypes.Int32Type, TDBXTypes.UInt16Type, TDBXTypes.BoolType, TDBXTypes.FloatType,
TDBXTypes.FloatType, TDBXTypes.BcdType, TDBXTypes.DateType, TDBXTypes.TimeType, 0,
TDBXTypes.BytesType, TDBXTypes.VarBytesType, TDBXTypes.Int32Type, TDBXTypes.BlobType,
TDBXTypes.BlobType, TDBXTypes.BlobType, TDBXTypes.BlobType, TDBXTypes.BlobType,
TDBXTypes.BlobType, TDBXTypes.BlobType, TDBXTypes.CursorType, TDBXTypes.ZStringType,
TDBXTypes.WideStringType, TDBXTypes.Int64Type, TDBXTypes.AdtType, TDBXTypes.ArrayType,
TDBXTypes.RefType, TDBXTypes.TableType, TDBXTypes.BlobType, TDBXTypes.BlobType,
TDBXTypes.UnknownType, TDBXTypes.UnknownType, TDBXTypes.UnknownType, TDBXTypes.ZStringType,
TDBXTypes.TimeStampType, TDBXTypes.BcdType, TDBXTypes.WideStringType, TDBXTypes.BlobType,
TDBXTypes.TimeStampType, TDBXTypes.ZStringType);
```

File

SqlExpr

Description

This is constant SqlExpr.FldTypeMap.

149.5.8 SqlExpr.HourGlassCursor Constant

Pascal

```
HourGlassCursor = -11;
```

C++

```
HourGlassCursor = -11;
```

File

SqlExpr

Description

This is constant SqlExpr.HourGlassCursor.

149.5.9 SqlExpr.SAnd Constant

Pascal

```
SAnd = ' and ';
```

C++

```
SAnd = ' and ';
```

File

SqlExpr

Description

This is constant SqlExpr.SAnd.

149.5.10 SqlExpr.SOrderBy Constant

Pascal

```
SOrderBy = ' order by ';
```

C++

```
SOrderBy = ' order by ';
```

File

SqlExpr

Description

This is constant SqlExpr.SOrderBy.

149.5.11 SqlExpr.SParam Constant

Pascal

```
SParam = '?';
```

C++

```
SParam = '?';
```

File

SqlExpr

Description

This is constant SqlExpr.SParam.

149.5.12 SqlExpr.SSelect Constant

Pascal

```
SSelect = 'select';
```

C++

```
SSelect = 'select';
```

File

SqlExpr

Description

This is constant SqlExpr.SSelect.

149.5.13 SqlExpr.SSelectSpaces Constant

Pascal

```
SSelectSpaces = ' select ';
```

C++

```
SSelectSpaces = ' select ';
```

File

SqlExpr

Description

This is constant SqlExpr.SSelectSpaces.

149.5.14 SqlExpr.SSelectStar Constant

Pascal

```
SSelectStar = ' select * ';
```

C++

```
SSelectStar = ' select * ';
```

File

SqlExpr

Description

This is constant SqlExpr.SSelectStar.

149.5.15 SqlExpr.SSelectStarFrom Constant

Pascal

```
SSelectStarFrom = ' select * from ';
```

C++

```
SSelectStarFrom = ' select * from ';
```

File

SqlExpr

Description

This is constant SqlExpr.SSelectStarFrom.

149.5.16 SqlExpr.SUB_TYPE_MEMO Constant

Pascal

```
SUB_TYPE_MEMO = TDBXTypes.MemoSubType;
```

C++

```
SUB_TYPE_MEMO = TDBXTypes.MemoSubType;
```

File

SqlExpr

Description

This is constant SqlExpr.SUB_TYPE_MEMO.

149.5.17 SqlExpr.SWhere Constant

Pascal

```
SWhere = ' where ';
```

C++

```
SWhere = ' where ';
```

File

SqlExpr

Description

This is constant SqlExpr.SWhere.

149.5.18 SqlExpr.TErrorMessageSize Constant

Pascal

```
TErrorMessageSize = 2048;
```

C++

```
TErrorMessageSize = 2048;
```

File

SqlExpr

Description

This is constant SqlExpr.TErrorMessageSize.

150 SqlTimSt Namespace

This is namespace SqlTimSt.

Topics

Name	Description
Functions (see page 20385)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 20392)	The following table lists structs, records, enums in this documentation.
Types (see page 20394)	The following table lists types in this documentation.
Constants (see page 20395)	The following table lists constants in this documentation.

Constants

Name	Description
NullSQLTimeStamp (see page 20395)	This is constant SqlTimSt.NullSQLTimeStamp.

Functions

	Name	Description
	CheckSqlTimeStamp (see page 20385)	Checks whether a TSQLTimeStamp (see page 20392) value represents a valid date and time.
	DateTimeToSQLTimeStamp (see page 20386)	Converts a TDateTime value to a TSQLTimeStamp (see page 20392) value.
	SQLDayOfWeek (see page 20386)	Indicates the day of the week when a specified TSQLTimeStamp (see page 20392) value occurs.
	SQLTimeStampToDateTime (see page 20387)	Converts a TSQLTimeStamp (see page 20392) value to a TDateTime value.
	SQLTimeStampToStr (see page 20387)	Converts a TSQLTimeStamp (see page 20392) value to a string.
	StrToSQLTimeStamp (see page 20388)	Converts a string to a TSQLTimeStamp (see page 20392) value.
	TryStrToSQLTimeStamp (see page 20388)	Converts a string to a TSQLTimeStamp (see page 20392) value.
	VarIsSQLTimeStamp (see page 20389)	Indicates whether a specified Variant represents a TSQLTimeStamp (see page 20392) value.
	VarSQLTimeStamp (see page 20389)	Returns the TVarType constant that identifies the internal type of a Variant as a TSQLTimeStamp (see page 20392) value.
	VarSQLTimeStampCreate (see page 20390)	Creates a new Variant that stores its value internally as a TSQLTimeStamp (see page 20392) value.
	VarSQLTimeStampCreate (see page 20390)	Creates a new Variant that stores its value internally as a TSQLTimeStamp (see page 20392) value.
	VarSQLTimeStampCreate (see page 20390)	Creates a new Variant that stores its value internally as a TSQLTimeStamp (see page 20392) value.
	VarSQLTimeStampCreate (see page 20390)	Creates a new Variant that stores its value internally as a TSQLTimeStamp (see page 20392) value.

	VarSQLTimeStampCreate (see page 20390)	Creates a new Variant that stores its value internally as a TSQLTimeStamp (see page 20392) value.
	VarToSQLTimeStamp (see page 20391)	Converts a Variant that represents a TSQLTimeStamp (see page 20392) value to the corresponding TSQLTimeStamp (see page 20392).

Structs, Records, Enums

	Name	Description
	TSQLTimeStamp (see page 20392)	TSQLTimeStamp represents time and date values with a high degree of accuracy.

Types

Name	Description
PSQLTimeStamp (see page 20394)	This is type SqlTimSt.PSQLTimeStamp.

150.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CheckSqlTimeStamp (see page 20385)	Checks whether a TSQLTimeStamp (see page 20392) value represents a valid date and time.
	DateTimeToSQLTimeStamp (see page 20386)	Converts a TDateTime value to a TSQLTimeStamp (see page 20392) value.
	SQLDayOfWeek (see page 20386)	Indicates the day of the week when a specified TSQLTimeStamp (see page 20392) value occurs.
	SQLTimeStampToDateTime (see page 20387)	Converts a TSQLTimeStamp (see page 20392) value to a TDateTime value.
	SQLTimeStampToStr (see page 20387)	Converts a TSQLTimeStamp (see page 20392) value to a string.
	StrToSQLTimeStamp (see page 20388)	Converts a string to a TSQLTimeStamp (see page 20392) value.
	TryStrToSQLTimeStamp (see page 20388)	Converts a string to a TSQLTimeStamp (see page 20392) value.
	VarIsSQLTimeStamp (see page 20389)	Indicates whether a specified Variant represents a TSQLTimeStamp (see page 20392) value.
	VarSQLTimeStamp (see page 20389)	Returns the TVarType constant that identifies the internal type of a Variant as a TSQLTimeStamp (see page 20392) value.
	VarSQLTimeStampCreate (see page 20390)	Creates a new Variant that stores its value internally as a TSQLTimeStamp (see page 20392) value.
	VarSQLTimeStampCreate (see page 20390)	Creates a new Variant that stores its value internally as a TSQLTimeStamp (see page 20392) value.
	VarSQLTimeStampCreate (see page 20390)	Creates a new Variant that stores its value internally as a TSQLTimeStamp (see page 20392) value.
	VarSQLTimeStampCreate (see page 20390)	Creates a new Variant that stores its value internally as a TSQLTimeStamp (see page 20392) value.
	VarSQLTimeStampCreate (see page 20390)	Creates a new Variant that stores its value internally as a TSQLTimeStamp (see page 20392) value.
	VarToSQLTimeStamp (see page 20391)	Converts a Variant that represents a TSQLTimeStamp (see page 20392) value to the corresponding TSQLTimeStamp (see page 20392).

150.1.1 SqlTimSt.CheckSqlTimeStamp Function

Checks whether a TSQLTimeStamp (see page 20392) value represents a valid date and time.

Pascal

```
procedure CheckSqlTimeStamp(const ASQLTimeStamp: TSQLTimeStamp);
```

C++

```
CheckSqlTimeStamp(const TSQLTimeStamp ASQLTimeStamp);
```

File

SqlTimSt

Description

CheckSqlTimeStamp checks that the specified TSQLTimeStamp (see page 20392) value represents a valid date and time. If the fields of the ASQLTimeStamp parameter are out of the valid range, CheckSqlTimeStamp raises an EConversionError exception.

See Also

EConvertError

150.1.2 SqlTimSt.DateTimeToSQLTimeStamp Function

Converts a TDateTime value to a TSQLTimeStamp (see page 20392) value.

Pascal

```
function DateTimeToSQLTimeStamp(const DateTime: TDateTime): TSQLTimeStamp;
```

C++

```
TSQLTimeStamp DateTimeToSQLTimeStamp(const TDateTime DateTime);
```

File

SqlTimSt

Description

Use DateTimeToSQLTimeStamp to convert the TDateTime value specified by DateTime to a TSQLTimeStamp (see page 20392) value.

See Also

SQLTimeStampToDateTime (see page 20387)

SQLTimeStampToStr (see page 20387)

StrToSQLTimeStamp (see page 20388)

TSQLTimeStampField

150.1.3 SqlTimSt.SQLDayOfWeek Function

Indicates the day of the week when a specified TSQLTimeStamp (see page 20392) value occurs.

Pascal

```
function SQLDayOfWeek(const DateTime: TSQLTimeStamp): integer;
```

C++

```
int SQLDayOfWeek(const TSQLTimeStamp DateTime);
```

File

SqlTimSt

Description

SQLDayOfWeek returns the day of the week of the specified date as an integer between 1 and 7, where Sunday is the first day of the week and Saturday is the seventh.

See Also

DayOfWeek

150.1.4 SqlTimSt.SQLTimeStampToDateTime Function

Converts a TSQLTimeStamp (see page 20392) value to a TDateTime value.

Pascal

```
function SQLTimeStampToDateTime(const DateTime: TSQLTimeStamp): TDateTime;
```

C++

```
TDateTime SQLTimeStampToDateTime(const TSQLTimeStamp DateTime);
```

File

SqlTimSt

Description

Use SQLTimeStampToDateTime to convert the TSQLTimeStamp (see page 20392) value specified by DateTime to a TDateTime value.

See Also

DateTimeToSQLTimeStamp (see page 20386)

SQLTimeStampToStr (see page 20387)

StrToSQLTimeStamp (see page 20388)

TSQLTimeStampField

150.1.5 SqlTimSt.SQLTimeStampToStr Function

Converts a TSQLTimeStamp (see page 20392) value to a string.

Pascal

```
function SQLTimeStampToStr(const Format: string; DateTime: TSQLTimeStamp): string;
```

C++

```
AnsiString SQLTimeStampToStr(const AnsiString Format, TSQLTimeStamp DateTime);
```

File

SqlTimSt

Description

Use SQLTimeStampToStr to convert a TSQLTimeStamp (see page 20392) value to a string an AnsiString.

DateTimeToStr converts the TSQLTimeStamp (see page 20392) value given by DateTime using the format given by Format and returns the resulting string.

If the string specified by the Format parameter is empty, the TSQLTimeStamp (see page 20392) value is formatted using the ShortDateFormat global variable followed by the LongTimeFormat global variable. The time is not displayed if all time fields in the DateTime parameter are 0.

To create a format string for the Format parameter, use the standard date-time format specifiers:

See Also

DateTimeToSQLTimeStamp (see page 20386)

SQLTimeStampToDateTime (see page 20387)

StrToSQLTimeStamp (see page 20388)

TSQLTimeStampField

CurrencyAndDatetimeFormatting

TryStrToSQLTimeStamp (see page 20388)

150.1.6 SqlTimSt.StrToSQLTimeStamp Function

Converts a string to a TSQLTimeStamp (see page 20392) value.

Pascal

```
function StrToSQLTimeStamp(const S: string): TSQLTimeStamp;
```

C++

```
TSQLTimeStamp StrToSQLTimeStamp(const AnsiString S);
```

File

SqlTimSt

Description

Use StrToSQLTimeStamp to convert the string S to a TSQLTimeStamp (see page 20392) value. If S does not represent a valid date-and-time value, StrToSQLTimeStamp raises an EConvertError exception.

Tip: To convert strings to TSQLTimeStamp (see page 20392) values without raising an exception when the string is not a valid date-and-time, use TryStrToSQLTimeStamp (see page 20388) instead.

See Also

DateTimeToSQLTimeStamp (see page 20386)

SQLTimeStampToStr (see page 20387)

SQLTimeStampToDateTime (see page 20387)

TSQLTimeStampField

EConvertError

TryStrToSQLTimeStamp (see page 20388)

150.1.7 SqlTimSt.TryStrToSQLTimeStamp Function

Converts a string to a TSQLTimeStamp (see page 20392) value.

Pascal

```
function TryStrToSQLTimeStamp(const S: string; var TimeStamp: TSQLTimeStamp): Boolean;
```

C++

```
Boolean TryStrToSQLTimeStamp(const AnsiString S, TSQLTimeStamp TimeStamp);
```

File

SqlTimSt

Description

Use TryStrToSQLTimeStamp to convert the string S to a TSQLTimeStamp (see page 20392) value.

If S can be converted to a TSQLTimeStamp (see page 20392) value, the result is returned as the TimeStamp parameter and TryStrToSQLTimeStamp returns true.

If S can't be converted to a TSQLTimeStamp (see page 20392) value, TimeStamp is set to NullSQLTimeStamp (see page 20395) and TryStrToSQLTimeStamp returns false.

See Also

DateTimeToSQLTimeStamp (see page 20386)

SQLTimeStampToStr (see page 20387)

SQLTimeStampToDateTime (see page 20387)

NullSQLTimeStamp (see page 20395)

StrToSQLTimeStamp (see page 20388)

150.1.8 SqlTimSt.VarIsSQLTimeStamp Function

Indicates whether a specified Variant represents a TSQLTimeStamp (see page 20392) value.

Pascal

```
function VarIsSQLTimeStamp(const aValue: Variant): Boolean; overload;
```

C++

```
Boolean VarIsSQLTimeStamp(const Variant aValue);
```

File

SqlTimSt

Description

VarIsSQLTimeStamp returns true if the Variant aValue stores its value internally as a TSQLTimeStamp (see page 20392) value. VarIsSQLTimeStamp returns false otherwise.

See Also

VarToSQLTimeStamp (see page 20391)

TSQLTimeStamp (see page 20392)

150.1.9 SqlTimSt.VarSQLTimeStamp Function

Returns the TVarType constant that identifies the internal type of a Variant as a TSQLTimeStamp (see page 20392) value.

Pascal

```
function VarSQLTimeStamp: TVarType;
```

C++

```
TVarType VarSQLTimeStamp();
```

File

SqlTimSt

Description

VarSQLTimeStamp returns the TVarType constant that identifies a TSQLTimeStamp (see page 20392)-valued Variant. This is the value that the VarType function returns when the Variant stores its value as a TSQLTimeStamp (see page 20392) value.

The Variant type codes for custom Variants (see page 23567) are dynamically allocated when the application loads the unit that implements them. VarSQLTimeStamp allows the application to identify the Variant type code that is currently used for custom Variants (see page 23567) that represent an SQL time stamp. This value, however, may not match the type code used by another application that uses a different set of custom Variant types.

See Also

VarIsSQLTimeStamp (see page 20389)

VarType

TSQLTimeStamp (see page 20392)

VarToSQLTimeStamp (see page 20391)

150.1.10 SqlTimSt.VarSQLTimeStampCreate Function

Creates a new Variant that stores its value internally as a TSQLTimeStamp (see page 20392) value.

Pascal

```
procedure VarSQLTimeStampCreate(var aDest: Variant; const ASQLTimeStamp: TSQLTimeStamp);
overload;
function VarSQLTimeStampCreate: Variant; overload;
function VarSQLTimeStampCreate(const AValue: string): Variant; overload;
function VarSQLTimeStampCreate(const AValue: TDateTime): Variant; overload;
function VarSQLTimeStampCreate(const ASQLTimeStamp: TSQLTimeStamp): Variant; overload;
```

C++

```
VarSQLTimeStampCreate(Variant aDest, const TSQLTimeStamp ASQLTimeStamp);
Variant VarSQLTimeStampCreate();
Variant VarSQLTimeStampCreate(const AnsiString AValue);
Variant VarSQLTimeStampCreate(const TDateTime AValue);
Variant VarSQLTimeStampCreate(const TSQLTimeStamp ASQLTimeStamp);
```

File

SqlTimSt

Description

Call VarSQLTimeStampCreate to obtain a Variant that stores its value as a TSQLTimeStamp (see page 20392) value.

Using the first syntax, the Variant returned as aDest has the same value as ASQLTimeStamp.

Using the second syntax (passing no arguments), the returned Variant has a value of NullSQLTimeStamp (see page 20395).

Using the third syntax, the returned Variant has the same value as ASQLTimeStamp.

Using the fourth syntax, the returned Variant represents the same date and time as the TDateTime value AValue.

See Also

VarIsSQLTimeStamp (see page 20389)

VarToSQLTimeStamp (see page 20391)

NullSQLTimeStamp (see page 20395)

150.1.11 SqlTimSt.VarToSQLTimeStamp Function

Converts a Variant that represents a TSQLTimeStamp (see page 20392) value to the corresponding TSQLTimeStamp (see page 20392).

Pascal

```
function VarToSQLTimeStamp(const aValue: Variant): TSQLTimeStamp;
```

C++

```
TSQLTimeStamp VarToSQLTimeStamp(const Variant aValue);
```

File

SqlTimSt

Description

Use VarToSQLTimeStamp to obtain a TSQLTimeStamp (see page 20392) from a variant whose value represents a date-and-time value. If the Variant is a Null Variant, VarToSQLTimeStamp returns NullSQLTimeStamp (see page 20395).

See Also

VarIsSQLTimeStamp (see page 20389)

TSQLTimeStamp (see page 20392)

NullSQLTimeStamp (see page 20395)

150.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TSQLTimeStamp (see page 20392)	TSQLTimeStamp represents time and date values with a high degree of accuracy.

150.2.1 SqlTimSt.TSQLTimeStamp Record

TSQLTimeStamp represents time and date values with a high degree of accuracy.

Pascal

```
TSQLTimeStamp = packed record
  Year: SmallInt;
  Month: Word;
  Day: Word;
  Hour: Word;
  Minute: Word;
  Second: Word;
  Fractions: LongWord;
end;
```

C++

```
struct TSQLTimeStamp {
  SmallInt Year;
  Word Month;
  Word Day;
  Word Hour;
  Word Minute;
  Word Second;
  LongWord Fractions;
};
```

File

SqlTimSt

Description

The dbExpress database drivers use TSQLTimeStamp when working with date and time information. Year specifies a year from 1 to 9999. Month specifies a month from 1 to 12. Day specifies a day of the month, from 1 through 28, 29, 30, or 31, depending on the Month value. Hour specifies the hour from 0 to 23. Minute and Second can range from 0 to 59. Fractions specifies milliseconds from 0 to 999.

You can create a Variant that represents a TSQLTimeStamp value using the VarSQLTimeStampCreate (see page 20390) function. In fact, the easiest way to manipulate TSQLTimeStamp values is to create such a Variant and use the built-in operators provided by the Variant type.

See Also

TDateTime

TTimeStamp

TSQLTimeStampField

SQLTimeStampTODateTime

SQLTimeStampToStr

DateTimeToSQLTimeStamp

SQLDayOfWeek

150.3 Types

The following table lists types in this documentation.

Types

Name	Description
PSQLTimeStamp (see page 20394)	This is type SqlTimSt.PSQLTimeStamp.

150.3.1 SqlTimSt.PSQLTimeStamp Type

Pascal

```
PSQLTimeStamp = ^TSQLTimeStamp;
```

C++

```
^TSQLTimeStamp PSQLTimeStamp;
```

File

SqlTimSt

Description

This is type SqlTimSt.PSQLTimeStamp.

150.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
NullSQLTimeStamp (see page 20395)	This is constant SqlTimSt.NullSQLTimeStamp.

150.4.1 SqlTimSt.NullSQLTimeStamp Constant

Pascal

```
NullSQLTimeStamp: TSQLTimeStamp = (Year: 0; Month: 0; Day: 0; Hour: 0; Minute: 0; Second: 0;
Fractions: 0);
```

C++

```
TSQLTimeStamp NullSQLTimeStamp = (Year: 0; Month: 0; Day: 0; Hour: 0; Minute: 0; Second: 0;
Fractions: 0);
```

File

SqlTimSt

Description

This is constant SqlTimSt.NullSQLTimeStamp.

151 StdActnMenus Namespace

This is namespace StdActnMenus.

Topics

Name	Description
Classes (see page 20398)	The following table lists classes in this documentation.
Functions (see page 20504)	The following table lists functions in this documentation.

Classes

	Name	Description
	TStandardAddRemoveItem (see page 20398)	TStandardAddRemoveItem is the ActionBand menu class for customizing action bands.
	TStandardButtonControl (see page 20408)	This is class StdActnMenus.TStandardButtonControl.
	TStandardCustomizePopup (see page 20418)	This is class StdActnMenus.TStandardCustomizePopup.
	TStandardDropDownButton (see page 20434)	This is class StdActnMenus.TStandardDropDownButton.
	TStandardMenuButton (see page 20444)	TStandardMenuButton is the standard ActionBand menu button class.
	TStandardMenuExpandBtn (see page 20455)	This is class StdActnMenus.TStandardMenuExpandBtn.
	TStandardMenuItem (see page 20465)	TStandardMenuItem is the default class for menu items in an action band.
	TStandardMenuPopup (see page 20478)	TStandardMenuPopup is the default menu popup style.
	TStandardToolScrollBtn (see page 20494)	This is class StdActnMenus.TStandardToolScrollBtn.

Functions

	Name	Description
	RegisterStandardMenus (see page 20504)	This is function StdActnMenus.RegisterStandardMenus.

151.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TStandardAddRemoveItem (see page 20398)	TStandardAddRemoveItem is the ActionBand menu class for customizing action bands.
	TStandardButtonControl (see page 20408)	This is class StdActnMenus.TStandardButtonControl.
	TStandardCustomizePopup (see page 20418)	This is class StdActnMenus.TStandardCustomizePopup.
	TStandardDropDownButton (see page 20434)	This is class StdActnMenus.TStandardDropDownButton.
	TStandardMenuButton (see page 20444)	TStandardMenuButton is the standard ActionBand menu button class.
	TStandardMenuExpandBtn (see page 20455)	This is class StdActnMenus.TStandardMenuExpandBtn.
	TStandardMenuItem (see page 20465)	TStandardMenuItem is the default class for menu items in an action band.
	TStandardMenuPopup (see page 20478)	TStandardMenuPopup is the default menu popup style.
	TStandardToolScrollBtn (see page 20494)	This is class StdActnMenus.TStandardToolScrollBtn.

151.1.1 TStandardAddRemoveItem Class

TStandardAddRemoveItem is the ActionBand menu class for customizing action bands.

Class Hierarchy

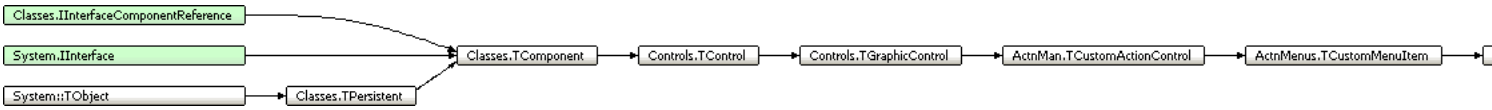

Pascal

```
TStandardAddRemoveItem = class(TCustomAddRemoveItem);
```

C++

```
class TStandardAddRemoveItem : public TCustomAddRemoveItem;
```

File

StdActnMenus

Description

TStandardAddRemoveItem is the standard ActionBand menu for customizing action bands at runtime.

Do not create instances of TStandardAddRemoveItem as this will be done for you automatically if the class is the default customization menu. If you would like to replace the standard customization menu, use the class TCustomAddRemoveItem as a parent class for your replacement class. To make the change for your application, set the value of MenuAddRemoveItemClass (a global variable) to your own subclass of TCustomAddRemoveItem instead of TStandardAddRemoveItem.

See Also

TCustomAddRemoveItem (see page 745)

151.1.1.1 TStandardAddRemoveItem Members

The following tables list the members exposed by TStandardAddRemoveItem.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FNoPrefix (see page 826)	This is FNoPrefix, a member of class TCustomMenuItem.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.

	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	BeginAutoDrag (see page 517)	Begin Automatic Drag.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CalcLayout (see page 518)	This is CalcLayout, a member of class TCustomActionControl.
V	CalcLayout (see page 827)	This is CalcLayout, a member of class TCustomMenuItem.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
	CMDesignHitTest (see page 519)	This is CMDesignHitTest, a member of class TCustomActionControl.
	CMFontChanged (see page 519)	This is CMFontChanged, a member of class TCustomActionControl.
	CMItemDropPoint (see page 519)	This is CMItemDropPoint, a member of class TCustomActionControl.
	CMItemSelected (see page 519)	This is CMItemSelected, a member of class TCustomActionControl.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DesignWndProc (see page 519)	This is DesignWndProc, a member of class TCustomActionControl.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
V	DoDragDrop (see page 520)	This is DoDragDrop, a member of class TCustomActionControl.
V	DoDragDrop (see page 828)	This is DoDragDrop, a member of class TCustomMenuItem.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
V	DoStartDrag (see page 520)	OnStartDrag event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 521)	OnDragOver event dispatcher.
V	DragOver (see page 828)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawBackground (see page 20407)	Draws a background image on the customization menu.
V	DrawDesignFocus (see page 521)	This is DrawDesignFocus, a member of class TCustomActionControl.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawDragDropPoint (see page 522)	This is DrawDragDropPoint, a member of class TCustomActionControl.
V	DrawEdge (see page 829)	This is DrawEdge, a member of class TCustomMenuItem.
V	DrawGlyph (see page 20408)	Draws a symbol on the control.

	DrawLargeGlyph (see page 522)	This is DrawLargeGlyph, a member of class TCustomActionControl.
	DrawSeparator (see page 522)	This is DrawSeparator, a member of class TCustomActionControl.
	DrawShadowedText (see page 523)	This is DrawShadowedText, a member of class TCustomActionControl.
	DrawShadowedText (see page 829)	This is DrawShadowedText, a member of class TCustomMenuItem.
	DrawSubMenuGlyph (see page 829)	This is DrawSubMenuGlyph, a member of class TCustomMenuItem.
	DrawText (see page 523)	Draws text on the control.
	DrawText (see page 829)	This is DrawText, a member of class TCustomMenuItem.
	DrawUnusedEdges (see page 830)	This is DrawUnusedEdges, a member of class TCustomMenuItem.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	GetAction (see page 523)	This is GetAction, a member of class TCustomActionControl.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetImageSize (see page 523)	This is GetImageSize, a member of class TCustomActionControl.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSeparator (see page 524)	This is GetSeparator, a member of class TCustomActionControl.
	GetShowCaption (see page 524)	This is GetShowCaption, a member of class TCustomActionControl.
	GetShowShortCut (see page 524)	This is GetShowShortCut, a member of class TCustomActionControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasGlyph (see page 524)	This is HasGlyph, a member of class TCustomActionControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionVisible (see page 755)	Indicates whether the add/remove item is visible in the menu of the customize dialog.
	IsChecked (see page 525)	This is IsChecked, a member of class TCustomActionControl.
	IsGrouped (see page 525)	This is IsGrouped, a member of class TCustomActionControl.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 755)	Responds when the user releases the mouse button when the mouse cursor is on this menu item.
	MouseUp (see page 830)	OnMouseUp event dispatcher.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.

	Paint (see page 526)	Render control surface.
	Paint (see page 831)	Render control surface.
	Paint (see page 6052)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionChanged (see page 526)	This is PositionChanged, a member of class TCustomActionControl.
	PositionChanged (see page 831)	This is PositionChanged, a member of class TCustomMenuItem.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetUsageData (see page 526)	This is ResetUsageData, a member of class TCustomActionControl.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActionClient (see page 526)	This is SetActionClient, a member of class TCustomActionControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropPoint (see page 527)	This is SetDropPoint, a member of class TCustomActionControl.
	SetEnabled (see page 756)	Sets the value of the Enabled property.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetGlyphLayout (see page 527)	This is SetGlyphLayout, a member of class TCustomActionControl.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelected (see page 527)	This is SetSelected, a member of class TCustomActionControl.
	SetSelected (see page 831)	This is SetSelected, a member of class TCustomMenuItem.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateSelection (see page 527)	This is UpdateSelection, a member of class TCustomActionControl.
	UpdateTextBounds (see page 527)	This is UpdateTextBounds, a member of class TCustomActionControl.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.

	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMSetText (see page 528)	This is WMSetText, a member of class TCustomActionControl.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GlyphPos (see page 532)	This is GlyphPos, a member of class TCustomActionControl.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	Menu (see page 832)	This is Menu, a member of class TCustomMenuItem.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	MouseSelected (see page 832)	This is MouseSelected, a member of class TCustomMenuItem.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ShortCutBounds (see page 832)	This is ShortCutBounds, a member of class TCustomMenuItem.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnClick (see page 535)	Occurs when the user clicks the control.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
V	CalcBounds (see page 518)	Calculates the size of the control.

	CalcBounds (see page 754)	Calculates the size of the add/remove menu item.
	CalcBounds (see page 827)	Calculates the size of the menu item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Click (see page 518)	Executes the control's function in response to a mouse click.
	Click (see page 827)	Executes the control's function.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Create (see page 515)	Creates an instance of TCustomActionControl (see page 507).
	Create (see page 825)	Creates an instance of TCustomMenuItem (see page 816).
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 5731)	Creates an instance of TControl (see page 5724).
	Create (see page 6051)	Creates an instance of TGraphicControl (see page 6044).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 516)	Destroys an instance of TCustomActionControl (see page 507).
	Destroy (see page 825)	Destroys an instance of TCustomMenuItem (see page 816).
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 520)	Performs actions and generates an OnDragDrop event.
	DragDrop (see page 828)	Performs actions and generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 524)	Calls the action client's Update method if the control is associated with an action client.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
V	Keyed (see page 525)	Executes the control's function in response to a keyboard action.
V	Keyed (see page 830)	Executes the menu item's function in response to a user's keyboard command.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.

V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	ActionBar (see page 528)	The ActionBand object which contains the custom action control
	ActionClient (see page 529)	Refers to the control's action client
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Caption (see page 529)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DropPoint (see page 532)	This is DropPoint, a member of class TCustomActionControl.
	Edges (see page 832)	Dictates which edges are drawn when the menu item is drawn.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GlyphLayout (see page 532)	Describes where the image should be on the control
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Margins (see page 532)	Describes the boundaries of the control for drawing purposes.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	Selected (see page 533)	Indicates whether the control is currently selected.

R	Separator (see page 533)	Indicates whether the control is a separator.
R	ShowCaption (see page 533)	Indicates whether the control's caption should be shown.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	ShowShortCut (see page 533)	Indicates whether the control's short cut should be shown.
	SmallIcon (see page 534)	Indicates whether the control's small icon should be shown.
	Spacing (see page 534)	Controls (see page 5711) the separation between the control's icon and caption.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TextBounds (see page 534)	A bounding rectangle for the control caption.
	Transparent (see page 534)	Shows whether the control is transparent or not.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

151.1.1.2 TStandardAddRemoveItem Methods

The methods of the TStandardAddRemoveItem class are listed here.

Protected Methods

	Name	Description
V	DrawBackground (see page 20407)	Draws a background image on the customization menu.
V	DrawGlyph (see page 20408)	Draws a symbol on the control.

151.1.1.2.1 TStandardAddRemoveItem.DrawBackground Method

Draws a background image on the customization menu.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

This method is largely the same as the inherited method, with some modifications to account for the method of selection. See the inherited method for more details.

See Also

DrawBackground (see page 521)

DrawGlyph (see page 20408)

DrawText

151.1.1.2.2 TStandardAddRemoveItem.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

DrawGlyph performs the same function as the method it inherits from, and it does call the inherited method. The override sets many of the draw parameters based on the present state of the menu and how it has been selected.

See Also

DrawGlyph (see page 522)

DrawBackground (see page 20407)

DrawText

151.1.2 TStandardButtonControl Class

Class Hierarchy

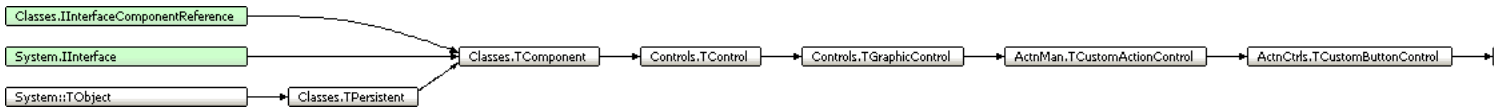

Pascal

```
TStandardButtonControl = class(TCustomButtonControl);
```

C++

```
class TStandardButtonControl : public TCustomButtonControl;
```

File

StdActnMenus

Description

This is class StdActnMenus.TStandardButtonControl.

151.1.2.1 TStandardButtonControl Members

The following tables list the members exposed by TStandardButtonControl.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFlat (see page 178)	This is FFlat, a member of class TCustomButtonControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FState (see page 178)	This is FState, a member of class TCustomButtonControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.

	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	BeginAutoDrag (see page 517)	Begin Automatic Drag.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CalcLayout (see page 518)	This is CalcLayout, a member of class TCustomActionControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
	CMDesignHitTest (see page 519)	This is CMDesignHitTest, a member of class TCustomActionControl.
	CMDialogChar (see page 180)	This is CMDialogChar, a member of class TCustomButtonControl.
	CMEnabledChanged (see page 180)	This is CMEnabledChanged, a member of class TCustomButtonControl.
	CMFontChanged (see page 519)	This is CMFontChanged, a member of class TCustomActionControl.
	CMItemDropPoint (see page 519)	This is CMItemDropPoint, a member of class TCustomActionControl.
	CMItemSelected (see page 519)	This is CMItemSelected, a member of class TCustomActionControl.
	CMMouseEnter (see page 180)	This is CMMouseEnter, a member of class TCustomButtonControl.
	CMMouseLeave (see page 180)	This is CMMouseLeave, a member of class TCustomButtonControl.
	CMTextChanged (see page 180)	This is CMTextChanged, a member of class TCustomButtonControl.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DesignWndProc (see page 519)	This is DesignWndProc, a member of class TCustomActionControl.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
V	DoDragDrop (see page 520)	This is DoDragDrop, a member of class TCustomActionControl.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
V	DoStartDrag (see page 520)	OnStartDrag event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 521)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawBackground (see page 181)	Draws a background image on the control.
V	DrawBackground (see page 521)	Draws a background image on the control.
V	DrawDesignFocus (see page 521)	This is DrawDesignFocus, a member of class TCustomActionControl.

	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawDragDropPoint (see page 522)	This is DrawDragDropPoint, a member of class TCustomActionControl.
V	DrawFrame (see page 20417)	This is DrawFrame, a member of class TStandardButtonControl.
V	DrawGlyph (see page 20418)	Draws a symbol on the control.
V	DrawLargeGlyph (see page 181)	This is DrawLargeGlyph, a member of class TCustomButtonControl.
V	DrawLargeGlyph (see page 522)	This is DrawLargeGlyph, a member of class TCustomActionControl.
V	DrawSeparator (see page 522)	This is DrawSeparator, a member of class TCustomActionControl.
V	DrawShadowedText (see page 523)	This is DrawShadowedText, a member of class TCustomActionControl.
V	DrawText (see page 20418)	This is DrawText, a member of class TStandardButtonControl.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	GetAction (see page 523)	This is GetAction, a member of class TCustomActionControl.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetImageSize (see page 523)	This is GetImageSize, a member of class TCustomActionControl.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
V	GetSeparator (see page 524)	This is GetSeparator, a member of class TCustomActionControl.
V	GetShowCaption (see page 524)	This is GetShowCaption, a member of class TCustomActionControl.
V	GetShowShortCut (see page 181)	This is GetShowShortCut, a member of class TCustomButtonControl.
V	GetShowShortCut (see page 524)	This is GetShowShortCut, a member of class TCustomActionControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasGlyph (see page 524)	This is HasGlyph, a member of class TCustomActionControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsChecked (see page 525)	This is IsChecked, a member of class TCustomActionControl.
V	IsGrouped (see page 525)	This is IsGrouped, a member of class TCustomActionControl.
V	Loaded (see page 182)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 182)	OnMouseDown event dispatcher.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
V	MouseMove (see page 183)	Respond to mouse moving over control area..
	MouseMove (see page 5773)	Respond to mouse moving over control area..
V	MouseUp (see page 183)	OnMouseUp event dispatcher.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.

	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Paint (see page 184)	Render control surface.
	Paint (see page 526)	Render control surface.
	Paint (see page 6052)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionChanged (see page 526)	This is PositionChanged, a member of class TCustomActionControl.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetUsageData (see page 526)	This is ResetUsageData, a member of class TCustomActionControl.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActionClient (see page 184)	This is SetActionClient, a member of class TCustomButtonControl.
	SetActionClient (see page 526)	This is SetActionClient, a member of class TCustomActionControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropPoint (see page 527)	This is SetDropPoint, a member of class TCustomActionControl.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetGlyphLayout (see page 185)	This is SetGlyphLayout, a member of class TCustomButtonControl.
	SetGlyphLayout (see page 527)	This is SetGlyphLayout, a member of class TCustomActionControl.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelected (see page 185)	This is SetSelected, a member of class TCustomButtonControl.
	SetSelected (see page 527)	This is SetSelected, a member of class TCustomActionControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateSelection (see page 527)	This is UpdateSelection, a member of class TCustomActionControl.
	UpdateTextBounds (see page 527)	This is UpdateTextBounds, a member of class TCustomActionControl.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMLButtonDblClk (see page 185)	This is WMLButtonDblClk, a member of class TCustomButtonControl.
	WMSetText (see page 528)	This is WMSetText, a member of class TCustomActionControl.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GlyphPos (see page 532)	This is GlyphPos, a member of class TCustomActionControl.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	MouseInControl (see page 189)	This is MouseInControl, a member of class TCustomButtonControl.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnClick (see page 535)	Occurs when the user clicks the control.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
V	CalcBounds (see page 179)	Calculates the size of the control.

	CalcBounds (see page 518)	Calculates the size of the control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Click (see page 179)	Executes the control's function.
	Click (see page 518)	Executes the control's function in response to a mouse click.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Create (see page 177)	Creates an instance of TCustomButtonControl (see page 168).
	Create (see page 515)	Creates an instance of TCustomActionControl (see page 507).
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 5731)	Creates an instance of TControl (see page 5724).
	Create (see page 6051)	Creates an instance of TGraphicControl (see page 6044).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 516)	Destroys an instance of TCustomActionControl (see page 507).
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 520)	Performs actions and generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 524)	Calls the action client's Update method if the control is associated with an action client.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	Keyed (see page 525)	Executes the control's function in response to a keyboard action.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 184)	Sets the Left, Top, Width, and Height properties all at once.
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.

	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	ActionBar (see page 528)	The ActionBand object which contains the custom action control
	ActionClient (see page 529)	Refers to the control's action client
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	AllowAllUp (see page 186)	Indicates whether all the buttons in a given group can be up at the same time.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Caption (see page 186)	Specifies a text string that identifies the control to the user.
	Caption (see page 529)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Down (see page 189)	Tells whether the button is down (selected) or up (unselected).
	DropPoint (see page 532)	This is DropPoint, a member of class TCustomActionControl.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
	Flat (see page 189)	Dictates whether the button should have a 2D look instead of the usual 3D look.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GlyphLayout (see page 532)	Describes where the image should be on the control
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Margins (see page 532)	Describes the boundaries of the control for drawing purposes.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	Selected (see page 533)	Indicates whether the control is currently selected.

R	Separator (see page 533)	Indicates whether the control is a separator.
	ShowCaption (see page 189)	Indicates whether the control's caption should be shown.
R	ShowCaption (see page 533)	Indicates whether the control's caption should be shown.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
	ShowShortCut (see page 190)	Indicates whether the control's short cut should be shown.
R	ShowShortCut (see page 533)	Indicates whether the control's short cut should be shown.
	SmallIcon (see page 534)	Indicates whether the control's small icon should be shown.
	Spacing (see page 534)	Controls (see page 5711) the separation between the control's icon and caption.
	State (see page 190)	Describes the current state of the button.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TextBounds (see page 534)	A bounding rectangle for the control caption.
	TrackButton (see page 190)	Specifies which mouse button activates the button control.
	Transparent (see page 534)	Shows whether the control is transparent or not.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

151.1.2.2 TStandardButtonControl Methods

The methods of the TStandardButtonControl class are listed here.

Protected Methods

	Name	Description
V	DrawFrame (see page 20417)	This is DrawFrame, a member of class TStandardButtonControl.
V	DrawGlyph (see page 20418)	Draws a symbol on the control.
V	DrawText (see page 20418)	This is DrawText, a member of class TStandardButtonControl.

151.1.2.2.1 TStandardButtonControl.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TStandardButtonControl.

151.1.2.2.2 TStandardButtonControl.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground (see page 521)

DrawText (see page 523)

151.1.2.2.3 TStandardButtonControl.DrawText Method

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: String); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, AnsiString Text);
```

Description

This is DrawText, a member of class TStandardButtonControl.

151.1.3 TStandardCustomizePopup Class

Class Hierarchy

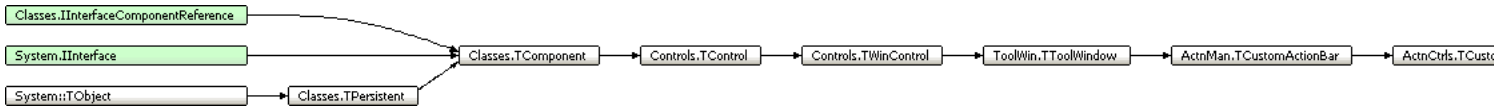

Pascal

```
TStandardCustomizePopup = class(TCustomizeActionToolBar);
```

C++

```
class TStandardCustomizePopup : public TCustomizeActionToolBar;
```

File

StdActnMenus

Description

This is class StdActnMenus.TStandardCustomizePopup.

151.1.3.1 TStandardCustomizePopup Members

The following tables list the members exposed by TStandardCustomizePopup.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.

	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FCachedMenu (see page 699)	This is FCachedMenu, a member of class TCustomActionMenuBar.
	FChildMenu (see page 699)	This is FChildMenu, a member of class TCustomActionMenuBar.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefaultColorMap (see page 460)	This is FDefaultColorMap, a member of class TCustomActionBar.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExpandTimer (see page 700)	This is FExpandTimer, a member of class TCustomActionMenuBar.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FPopupTimer (see page 700)	This is FPopupTimer, a member of class TCustomActionMenuBar.
	FSavedWrapState (see page 460)	This is FSavedWrapState, a member of class TCustomActionBar.
	FSelectedItem (see page 700)	This is FSelectedItem, a member of class TCustomActionMenuBar.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AddEdges (see page 737)	This is AddEdges, a member of class TCustomActionPopupMenu.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 462)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	Animate (see page 701)	This is Animate, a member of class TCustomActionMenuBar.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
V	AutoSizingChanged (see page 462)	This is AutoSizingChanged, a member of class TCustomActionBar.
	BackgroundSize (see page 462)	This is BackgroundSize, a member of class TCustomActionBar.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcInsertPosition (see page 462)	This is CalcInsertPosition, a member of class TCustomActionBar.
V	CanAutoSize (see page 20433)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CleanupStack (see page 701)	This is CleanupStack, a member of class TCustomActionMenuBar.

	Clear (see page 463)	This is Clear, a member of class TCustomActionBar.
	Clear (see page 778)	This is Clear, a member of class TCustomizeActionToolBar.
	ClearSubMenus (see page 702)	This is ClearSubMenus, a member of class TCustomActionMenuBar.
	Click (see page 5743)	Respond to user click.
	CMBorderChanged (see page 23285)	This is CMBorderChanged, a member of class TToolWindow.
	CMCtl3DChanged (see page 23286)	This is CMCtl3DChanged, a member of class TToolWindow.
	CMMouseLeave (see page 738)	This is CMMouseLeave, a member of class TCustomActionPopupMenu.
	CMVisibleChanged (see page 738)	This is CMVisibleChanged, a member of class TCustomActionPopupMenu.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateControl (see page 463)	This is CreateControl, a member of class TCustomActionBar.
	CreateControl (see page 702)	This is CreateControl, a member of class TCustomActionMenuBar.
	CreateControl (see page 738)	This is CreateControl, a member of class TCustomActionPopupMenu.
	CreateControls (see page 463)	This is CreateControls, a member of class TCustomActionBar.
	CreateControls (see page 738)	This is CreateControls, a member of class TCustomActionPopupMenu.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 739)	Initializes a window-creation parameter data structure.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreatePopup (see page 702)	This is CreatePopup, a member of class TCustomActionMenuBar.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 463)	Returns placement order of custom-aligned child control.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 464)	Returns aligned position of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 465)	This is DesignWndProc, a member of class TCustomActionBar.
	DesignWndProc (see page 739)	This is DesignWndProc, a member of class TCustomActionPopupMenu.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DisplayShadow (see page 739)	This is DisplayShadow, a member of class TCustomActionPopupMenu.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoControlCreated (see page 465)	This is DoControlCreated, a member of class TCustomActionBar.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoDropActions (see page 465)	This is DoDropActions, a member of class TCustomActionBar.
	DoDropCategory (see page 465)	This is DoDropCategory, a member of class TCustomActionBar.

	DoDropItem (see page 466)	This is DoDropItem, a member of class TCustomActionBar.
	DoEndDock (see page 146)	OnEndDock event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetControlClass (see page 466)	This is DoGetControlClass, a member of class TCustomActionBar.
	DoGetPopupClass (see page 703)	This is DoGetPopupClass, a member of class TCustomActionMenuBar.
	DoItemClicked (see page 703)	This is DoItemClicked, a member of class TCustomActionMenuBar.
	DoItemClicked (see page 778)	This is DoItemClicked, a member of class TCustomizeActionToolBar.
	DoItemKeyed (see page 703)	This is DoItemKeyed, a member of class TCustomActionMenuBar.
	DoItemSelected (see page 703)	This is DoItemSelected, a member of class TCustomActionMenuBar.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMenuDelay (see page 703)	This is DoMenuDelay, a member of class TCustomActionMenuBar.
	DoMouseIdle (see page 704)	This is DoMouseIdle, a member of class TCustomActionMenuBar.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoneMenuLoop (see page 704)	This is DoneMenuLoop, a member of class TCustomActionMenuBar.
	DoPopup (see page 704)	This is DoPopup, a member of class TCustomActionMenuBar.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 147)	OnStartDock event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 466)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawBackground (see page 467)	This is DrawBackground, a member of class TCustomActionBar.
	DrawBackground (see page 740)	This is DrawBackground, a member of class TCustomActionPopupMenu.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	ExecAction (see page 704)	This is ExecAction, a member of class TCustomActionMenuBar.
	ExecAction (see page 740)	This is ExecAction, a member of class TCustomActionPopupMenu.
	Expand (see page 704)	Causes the menu bar to show all items, including the hidden ones.
	ExpandClick (see page 740)	This is ExpandClick, a member of class TCustomActionPopupMenu.
	ExpandTimer (see page 705)	This is ExpandTimer, a member of class TCustomActionMenuBar.
	FindAccelItem (see page 467)	This is FindAccelItem, a member of class TCustomActionBar.
	FindItemAt (see page 468)	This is FindItemAt, a member of class TCustomActionBar.
	FindItemCaption (see page 468)	This is FindItemCaption, a member of class TCustomActionBar.
	FindLast (see page 468)	This is FindLast, a member of class TCustomActionBar.
	FindLeastUsedItem (see page 469)	This is FindLeastUsedItem, a member of class TCustomActionBar.
	FindNearestControl (see page 469)	This is FindNearestControl, a member of class TCustomActionBar.
	FindNext (see page 470)	This is FindNext, a member of class TCustomActionBar.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FindNextVisibleItem (see page 470)	This is FindNextVisibleItem, a member of class TCustomActionBar.
	FindPrevious (see page 470)	This is FindPrevious, a member of class TCustomActionBar.
	FindPreviousVisibleItem (see page 470)	This is FindPreviousVisibleItem, a member of class TCustomActionBar.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.

	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionControl (see page 471)	This is GetActionControl, a member of class TCustomActionBar.
	GetActionControl (see page 779)	This is GetActionControl, a member of class TCustomizeActionToolBar.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetAddRemoveItemClass (see page 779)	This is GetAddRemoveItemClass, a member of class TCustomizeActionToolBar.
	GetBannerWidth (see page 471)	This is GetBannerWidth, a member of class TCustomActionBar.
	GetBarHeight (see page 471)	This is GetBarHeight, a member of class TCustomActionBar.
	GetBarWidth (see page 471)	This is GetBarWidth, a member of class TCustomActionBar.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 741)	Returns the value of the ClientRect property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlClass (see page 471)	This is GetControlClass, a member of class TCustomActionBar.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDefaultColorMapClass (see page 472)	This is GetDefaultColorMapClass, a member of class TCustomActionBar.
	GetDefaultSounds (see page 705)	This is GetDefaultSounds, a member of class TCustomActionMenuBar.
	GetDesignMode (see page 472)	This is GetDesignMode, a member of class TCustomActionBar.
	GetDesignMode (see page 705)	This is GetDesignMode, a member of class TCustomActionMenuBar.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetExpandBtnClass (see page 741)	This is GetExpandBtnClass, a member of class TCustomActionPopupMenu.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 147)	Returns the class used for the control when it is floating.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetMouseHoverItem (see page 705)	This is GetMouseHoverItem, a member of class TCustomActionMenuBar.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupClass (see page 706)	This is GetPopupClass, a member of class TCustomActionMenuBar.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetPopupMenuClass (see page 472)	This is GetPopupMenuClass, a member of class TCustomActionBar.
	GetSelected (see page 706)	This is GetSelected, a member of class TCustomActionMenuBar.
	GetSiteInfo (see page 147)	Generates an OnGetSiteInfo event
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetSubMenuControlClass (see page 779)	This is GetSubMenuControlClass, a member of class TCustomizeActionToolBar.
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasItems (see page 472)	This is HasItems, a member of class TCustomActionBar.

	HideShadow (see page 741)	This is HideShadow, a member of class TCustomActionPopupMenu.
	Idle (see page 706)	This is Idle, a member of class TCustomActionMenuBar.
	InitMenuLoop (see page 706)	This is InitMenuLoop, a member of class TCustomActionMenuBar.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsDesignMsg (see page 707)	This is IsDesignMsg, a member of class TCustomActionMenuBar.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	ItemCount (see page 473)	This is ItemCount, a member of class TCustomActionBar.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 148)	Initializes the control after it is loaded from a stream.
	Loaded (see page 473)	Initializes the control after it is loaded from a stream.
	Loaded (see page 707)	Initializes the control after it is loaded from a stream.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 707)	This is MouseActivate, a member of class TCustomActionMenuBar.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	NCPaint (see page 20434)	This is NCPaint, a member of class TStandardCustomizePopup.
	NewPopup (see page 707)	This is NewPopup, a member of class TCustomActionMenuBar.
	NewPopup (see page 741)	This is NewPopup, a member of class TCustomActionPopupMenu.
	NewPopup (see page 780)	This is NewPopup, a member of class TCustomizeActionToolBar.
	Notification (see page 473)	Responds to notifications that components are being created or destroyed.
	Notification (see page 708)	Responds to notifications that components are being created or destroyed.
	Notification (see page 742)	Responds to notifications that components are being created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 474)	This is Paint, a member of class TCustomActionBar.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 474)	Renders the image of a windowed control.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	PositionPopup (see page 20434)	This is PositionPopup, a member of class TStandardCustomizePopup.
	ProcessMenuLoop (see page 708)	This is ProcessMenuLoop, a member of class TCustomActionMenuBar.
	ProcessMessage (see page 708)	This is ProcessMessage, a member of class TCustomActionMenuBar.
	ProcessMessages (see page 709)	This is ProcessMessages, a member of class TCustomActionMenuBar.
	ProcessMouseMsg (see page 709)	This is ProcessMouseMsg, a member of class TCustomActionMenuBar.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.

	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Reset (see page 475)	This is Reset, a member of class TCustomActionBar.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Select (see page 709)	This is Select, a member of class TCustomActionMenuBar.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectItem (see page 710)	This is SelectItem, a member of class TCustomActionMenuBar.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActionClient (see page 475)	This is SetActionClient, a member of class TCustomActionBar.
	SetActionClient (see page 780)	This is SetActionClient, a member of class TCustomizeActionToolBar.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetAutoSizing (see page 475)	This is SetAutoSizing, a member of class TCustomActionBar.
	SetBiDiMode (see page 475)	Sets the control's bi-directional mode.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetColorMap (see page 476)	This is SetColorMap, a member of class TCustomActionBar.
	SetColorMap (see page 743)	This is SetColorMap, a member of class TCustomActionPopupMenu.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignMode (see page 476)	This is SetDesignMode, a member of class TCustomActionBar.
	SetDesignMode (see page 710)	Specifies whether the whether the action band is being customized or being used in the IDE.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetOrientation (see page 477)	This is SetOrientation, a member of class TCustomActionBar.
	SetOrientation (see page 743)	This is SetOrientation, a member of class TCustomActionPopupMenu.
	SetParent (see page 710)	Sets the parent of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetPersistentHotKeys (see page 477)	This is SetPersistentHotKeys, a member of class TCustomActionBar.

V	SetPersistentHotKeys (see page 711)	This is SetPersistentHotKeys, a member of class TCustomActionMenuBar.
V	SetSpacing (see page 477)	This is SetSpacing, a member of class TCustomActionBar.
	SetupDefaultColorMap (see page 477)	This is SetupDefaultColorMap, a member of class TCustomActionBar.
	SetupExpandBtn (see page 743)	This is SetupExpandBtn, a member of class TCustomActionPopupMenu.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	VisibleChanging (see page 478)	Provides the interface for a method that responds to changes in the control's visibility.
V	VisibleChanging (see page 743)	Provides the interface for a method that responds to changes in the control's visibility.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMKeyDown (see page 744)	This is WMKeyDown, a member of class TCustomActionPopupMenu.
	WMNCCalcSize (see page 23286)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCPaint (see page 23286)	This is WMNCPaint, a member of class TToolWindow.
	WMPrint (see page 744)	This is WMPrint, a member of class TCustomActionPopupMenu.
	WMPrintClient (see page 744)	This is WMPrintClient, a member of class TCustomActionPopupMenu.
V	WndProc (see page 711)	Provides specific message responses for the control.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
R	AddlItem (see page 780)	This is AddlItem, a member of class TCustomizeActionToolBar.
	AnimateDuration (see page 716)	This is AnimateDuration, a member of class TCustomActionMenuBar.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.

	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DelayItem (see page 717)	This is DelayItem, a member of class TCustomActionMenuBar.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Expandable (see page 717)	This is Expandable, a member of class TCustomActionMenuBar.
	Expanded (see page 718)	This is Expanded, a member of class TCustomActionMenuBar.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
R	HRowCount (see page 488)	Indicates the number of columns of items that the action band displays.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	Items (see page 488)	Lists the action clients that correspond to the child controls of this action band.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	MouseControl (see page 718)	This is MouseControl, a member of class TCustomActionMenuBar.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentMenu (see page 719)	This is ParentMenu, a member of class TCustomActionMenuBar.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
R	PopupStack (see page 719)	This is PopupStack, a member of class TCustomActionMenuBar.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
R	Selected (see page 719)	This is Selected, a member of class TCustomActionMenuBar.
	Text (see page 5842)	Contains a text string associated with the control.
R	VRowCount (see page 490)	Indicates the number of rows of items that the action band displays.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
R	Widest (see page 491)	This is Widest, a member of class TCustomActionBar.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnControlCreated (see page 491)	Occurs when a control is created on the action band.
	OnEnterMenuLoop (see page 720)	Occurs when an ActionBand menu loop has begun.
	OnExitMenuLoop (see page 721)	Occurs when an ActionBand menu loop has ended and the menu is closed.
	OnGetControlClass (see page 492)	Occurs when an item is to be placed on the action band.
	OnGetPopupClass (see page 721)	Occurs prior to an ActionBand popupmenu appearing on screen.
	OnPaint (see page 492)	Occurs when the action band is painted.
	OnPopup (see page 721)	Occurs just before the pop-up menu appears.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.

	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CalcDockedWidth (see page 146)	This is CalcDockedWidth, a member of class TCustomActionDockBar.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	CloseMenu (see page 702)	Closes the menu currently being displayed.
	CloseMenu (see page 737)	Closes the menu currently being displayed.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 145)	Creates and initializes a TCustomActionDockBar (see page 133) instance.
	Create (see page 459)	Creates and initializes a TCustomActionBar (see page 447) instance.
	Create (see page 698)	Creates and initializes a TCustomActionMenuBar (see page 685) instance.
	Create (see page 735)	Creates and initializes a TCustomActionPopupMenu (see page 721) instance.
	Create (see page 777)	Creates and initializes a TCustomActionPopupMenu instance.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 5731)	Creates an instance of TControl (see page 5724).
	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 23285)	Creates and initializes a TToolWindow (see page 23274) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 459)	Destroys an instance of TCustomActionBar (see page 447).
	Destroy (see page 698)	Destroys an instance of TCustomActionMenuBar (see page 685).
	Destroy (see page 736)	Destroys an instance of TCustomActionMenuBar (see page 685).
	Destroy (see page 777)	Destroys an instance of TCustomActionMenuBar.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 146)	Used internally to dock the action band.
	Dock (see page 5749)	Used internally to dock the control.

	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
V	DragDrop (see page 466)	Generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	Expand (see page 740)	Performs the same kinds of functions as the inherited method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
V	FindFirst (see page 467)	Finds the first TActionClientItem (see page 374) in the Items (see page 488) list.
	FindFirstVisibleItem (see page 468)	Finds the first visible TActionClientItem (see page 374) in the Items (see page 488) list.
	FindLastVisibleItem (see page 469)	Finds the last visibleTActionClientItem in the Items (see page 488) list.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.

	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ItemCount (see page 779)	This is ItemCount, a member of class TCustomizeActionToolBar.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	Popup (see page 742)	Causes the popup menu to appear.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	RecreateControls (see page 475)	Destroys and recreates all of the action band's action client controls.
	RecreateControls (see page 709)	Destroys and recreates all of the action band's action client controls.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 476)	Sets the bounds based on whether the popup is expanded or not
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetDesignVisible (see page 6178)	Makes a control visible at design time.
	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Style (see page 477)	Identifies the style object that gives the application its default look and feel.
	Style (see page 711)	Retrieves the style used by the ActionBand menu that started the menu loop so the submenus have a consistent appearance.
	TrackMenu (see page 711)	Executes the action associated with the menu item.
	Update (see page 5789)	Processes any pending paint messages immediately.
	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.

	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	ActionClient (see page 479)	Specifies the action client associated with this action band.
R	ActionControls (see page 479)	Lists the controls on the action band.
	ActionManager (see page 480)	Specifies the action manager component that is associated with the action band.
R	AdditionalItem (see page 780)	This is AdditionalItem, a member of class TCustomizeActionToolBar.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AllowHiding (see page 480)	Determines whether the action band can be hidden.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AnimationStyle (see page 716)	Specifies the type of animation the menu bar will display as it opens.
	AutoFocus (see page 717)	This is AutoFocus, a member of class TCustomActionMenuBar.
	AutoSizing (see page 480)	Specifies whether the action band adjusts its size automatically to fit around its child controls.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 481)	Represents the drawing surface of the action band.
	Caption (see page 481)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 483)	Specifies the background color of the control.
	ColorMap (see page 484)	Indicates the colormap for individual action band components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ContextBar (see page 485)	Indicates whether items shown in this action band represent the items in a popup menu.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designable (see page 485)	Specifies whether the action band can be customized (used internally by the IDE).
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignMode (see page 486)	Specifies whether the action band is being customized or being used in the IDE.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.

	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EdgeBorders (see page 23287)	Determines which sides of the control have visible borders.
	EdgeInner (see page 23287)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 23288)	Determines the border style for the control's outer edge.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
	ExpandBtn (see page 744)	Refers to the popup menu's expand button.
	ExpandDelay (see page 717)	Specifies, in milliseconds, the delay period that a menu action band uses before showing hidden or unused items.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Font (see page 486)	Controls the attributes of text written on or in the control.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HorzMargin (see page 487)	Specifies the width of the horizontal margins in pixels.
	HorzSeparator (see page 487)	Indicates whether a horizontal separator bar appears between item lines.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	InMenuLoop (see page 718)	Indicates whether items in the menu can be selected.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	Orientation (see page 488)	Specifies how the action band lays out its child items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentControl (see page 718)	Refers to the parent object of the TCustomActionBarControl.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PersistentHotKeys (see page 489)	Specifies whether keyboard accelerator shortcuts are always displayed.
	RootMenu (see page 719)	Refers to the root menu of the ActionBand menu.
	Shadow (see page 745)	Refers to the popup menu's shadow.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	Spacing (see page 489)	Determines where the image and text appear on an action band's rendered control that represents the action client.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	UseSystemFont (see page 720)	Specifies whether the default system font for menu items is used.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	VertMargin (see page 490)	Specifies the width of the vertical margins in pixels.
	VertSeparator (see page 490)	Indicates whether a vertical separator bar should appear between item lines.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

151.1.3.2 TStandardCustomizePopup Methods

The methods of the TStandardCustomizePopup class are listed here.

Protected Methods

	Name	Description
	CanAutoSize (see page 20433)	Indicates how the control can be resized.
	NCPaint (see page 20434)	This is NCPaint, a member of class TStandardCustomizePopup.
	PositionPopup (see page 20434)	This is PositionPopup, a member of class TStandardCustomizePopup.

151.1.3.2.1 TStandardCustomizePopup.CanAutoSize Method

Indicates how the control can be resized.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize property is true and an attempt is made to resize the control. It allows controls to implement the AutoSize property before the resize sequence that begins with the OnCanResize and OnConstrainedResize events and ends with the OnResize event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the control. CanAutoSize adjusts these values so that the control's size adjusts to its contents. If the adjustment results in values of NewWidth and NewHeight that differ from the control's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width and Height, CanAutoSize returns false, and the resize sequence is aborted.

As implemented in TWinControl, CanAutoSize returns true, adjusting NewWidth and NewHeight so that all the control's children fit.

See Also

Constraints (see page 5812)

ConstrainedResize (see page 6133)

AdjustSize (see page 6129)

151.1.3.2.2 TStandardCustomizePopup.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TStandardCustomizePopup.

151.1.3.2.3 TStandardCustomizePopup.PositionPopup Method

Pascal

```
procedure PositionPopup(AnOwner: TCustomActionBar; ParentItem: TCustomActionControl); override;
```

C++

```
virtual __fastcall PositionPopup(TCustomActionBar AnOwner, TCustomActionControl ParentItem);
```

Description

This is PositionPopup, a member of class TStandardCustomizePopup.

151.1.4 TStandardDropDownButton Class

Class Hierarchy

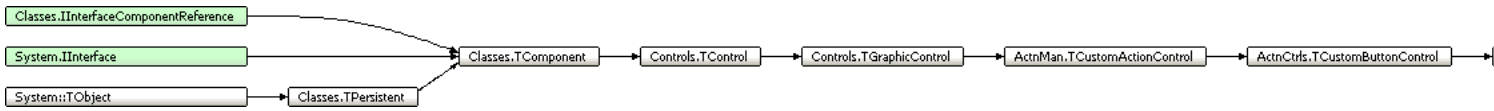

Pascal

```
TStandardDropDownButton = class(TCustomDropDownButton);
```

C++

```
class TStandardDropDownButton : public TCustomDropDownButton;
```

File

StdActnMenus

Description

This is class StdActnMenus.TStandardDropDownButton.

151.1.4.1 TStandardDropDownButton Members

The following tables list the members exposed by TStandardDropDownButton.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFlat (see page 178)	This is FFlat, a member of class TCustomButtonControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FState (see page 178)	This is FState, a member of class TCustomButtonControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.

	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	BeginAutoDrag (see page 517)	Begin Automatic Drag.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CalcLayout (see page 518)	This is CalcLayout, a member of class TCustomActionControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
	CMDesignHitTest (see page 519)	This is CMDesignHitTest, a member of class TCustomActionControl.
	CMDialogChar (see page 180)	This is CMDialogChar, a member of class TCustomButtonControl.
	CMEnabledChanged (see page 180)	This is CMEnabledChanged, a member of class TCustomButtonControl.
	CMFontChanged (see page 519)	This is CMFontChanged, a member of class TCustomActionControl.
	CMItemDropPoint (see page 519)	This is CMItemDropPoint, a member of class TCustomActionControl.
	CMItemSelected (see page 519)	This is CMItemSelected, a member of class TCustomActionControl.
	CMMouseEnter (see page 180)	This is CMMouseEnter, a member of class TCustomButtonControl.
	CMMouseLeave (see page 180)	This is CMMouseLeave, a member of class TCustomButtonControl.
	CMTextChanged (see page 180)	This is CMTextChanged, a member of class TCustomButtonControl.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DesignWndProc (see page 519)	This is DesignWndProc, a member of class TCustomActionControl.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
V	DoDragDrop (see page 520)	This is DoDragDrop, a member of class TCustomActionControl.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
V	DoStartDrag (see page 520)	OnStartDrag event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 521)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawBackground (see page 181)	Draws a background image on the control.
V	DrawBackground (see page 521)	Draws a background image on the control.
V	DrawDesignFocus (see page 521)	This is DrawDesignFocus, a member of class TCustomActionControl.

	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawDragDropPoint (see page 522)	This is DrawDragDropPoint, a member of class TCustomActionControl.
V	DrawFrame (see page 20444)	This is DrawFrame, a member of class TStandardDropDownButton.
V	DrawGlyph (see page 522)	Draws a symbol on the control.
V	DrawLargeGlyph (see page 181)	This is DrawLargeGlyph, a member of class TCustomButtonControl.
V	DrawLargeGlyph (see page 522)	This is DrawLargeGlyph, a member of class TCustomActionControl.
V	DrawSeparator (see page 522)	This is DrawSeparator, a member of class TCustomActionControl.
V	DrawShadowedText (see page 523)	This is DrawShadowedText, a member of class TCustomActionControl.
V	DrawText (see page 523)	Draws text on the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	GetAction (see page 523)	This is GetAction, a member of class TCustomActionControl.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetImageSize (see page 523)	This is GetImageSize, a member of class TCustomActionControl.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
V	GetPopupClass (see page 20444)	This is GetPopupClass, a member of class TStandardDropDownButton.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
V	GetSeparator (see page 524)	This is GetSeparator, a member of class TCustomActionControl.
V	GetShowCaption (see page 524)	This is GetShowCaption, a member of class TCustomActionControl.
V	GetShowShortCut (see page 181)	This is GetShowShortCut, a member of class TCustomButtonControl.
V	GetShowShortCut (see page 524)	This is GetShowShortCut, a member of class TCustomActionControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasGlyph (see page 524)	This is HasGlyph, a member of class TCustomActionControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsChecked (see page 525)	This is IsChecked, a member of class TCustomActionControl.
V	IsGrouped (see page 525)	This is IsGrouped, a member of class TCustomActionControl.
V	Loaded (see page 182)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 182)	OnMouseDown event dispatcher.
V	MouseDown (see page 216)	OnMouseDown event dispatcher.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
V	MouseMove (see page 183)	Respond to mouse moving over control area..
	MouseMove (see page 5773)	Respond to mouse moving over control area..

	MouseOverDropDown (see page 216)	This is MouseOverDropDown, a member of class TCustomDropDownButton.
V	MouseUp (see page 183)	OnMouseUp event dispatcher.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 184)	Render control surface.
V	Paint (see page 216)	Render control surface.
V	Paint (see page 526)	Render control surface.
V	Paint (see page 6052)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	PositionChanged (see page 526)	This is PositionChanged, a member of class TCustomActionControl.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetUsageData (see page 526)	This is ResetUsageData, a member of class TCustomActionControl.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
V	SetActionClient (see page 184)	This is SetActionClient, a member of class TCustomButtonControl.
V	SetActionClient (see page 526)	This is SetActionClient, a member of class TCustomActionControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropPoint (see page 527)	This is SetDropPoint, a member of class TCustomActionControl.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
V	SetGlyphLayout (see page 185)	This is SetGlyphLayout, a member of class TCustomButtonControl.
V	SetGlyphLayout (see page 527)	This is SetGlyphLayout, a member of class TCustomActionControl.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetSelected (see page 185)	This is SetSelected, a member of class TCustomButtonControl.
V	SetSelected (see page 527)	This is SetSelected, a member of class TCustomActionControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	UpdateSelection (see page 527)	This is UpdateSelection, a member of class TCustomActionControl.
	UpdateTextBounds (see page 527)	This is UpdateTextBounds, a member of class TCustomActionControl.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMLButtonDblClk (see page 185)	This is WMLButtonDblClk, a member of class TCustomButtonControl.
	WMSetText (see page 528)	This is WMSetText, a member of class TCustomActionControl.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
R	DroppedDown (see page 217)	This is DroppedDown, a member of class TCustomDropDownButton.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GlyphPos (see page 532)	This is GlyphPos, a member of class TCustomActionControl.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	MouseInControl (see page 189)	This is MouseInControl, a member of class TCustomButtonControl.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnClick (see page 535)	Occurs when the user clicks the control.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.

	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	CalcBounds (see page 179)	Calculates the size of the control.
	CalcBounds (see page 214)	Calculates the size of the control.
	CalcBounds (see page 518)	Calculates the size of the control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Click (see page 179)	Executes the control's function.
	Click (see page 215)	Executes the control's function.
	Click (see page 518)	Executes the control's function in response to a mouse click.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Create (see page 177)	Creates an instance of TCustomButtonControl (see page 168).
	Create (see page 515)	Creates an instance of TCustomActionControl (see page 507).
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 5731)	Creates an instance of TControl (see page 5724).
	Create (see page 6051)	Creates an instance of TGraphicControl (see page 6044).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 516)	Destroys an instance of TCustomActionControl (see page 507).
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 520)	Performs actions and generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	DropDownClick (see page 215)	This is DropDownClick, a member of class TCustomDropDownButton.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 524)	Calls the action client's Update method if the control is associated with an action client.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
V	Keyed (see page 525)	Executes the control's function in response to a keyboard action.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 184)	Sets the Left, Top, Width, and Height properties all at once.

V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	ActionBar (see page 528)	The ActionBand object which contains the custom action control
	ActionClient (see page 529)	Refers to the control's action client
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	AllowAllUp (see page 186)	Indicates whether all the buttons in a given group can be up at the same time.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Caption (see page 186)	Specifies a text string that identifies the control to the user.
	Caption (see page 529)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Down (see page 189)	Tells whether the button is down (selected) or up (unselected).
	DropPoint (see page 532)	This is DropPoint, a member of class TCustomActionControl.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
	Flat (see page 189)	Dictates whether the button should have a 2D look instead of the usual 3D look.
R	Floating (see page 5823)	Indicates whether the control is floating.

	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GlyphLayout (see page 532)	Describes where the image should be on the control
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Margins (see page 532)	Describes the boundaries of the control for drawing purposes.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	Selected (see page 533)	Indicates whether the control is currently selected.
R	Separator (see page 533)	Indicates whether the control is a separator.
	ShowCaption (see page 189)	Indicates whether the control's caption should be shown.
R	ShowCaption (see page 533)	Indicates whether the control's caption should be shown.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
	ShowShortCut (see page 190)	Indicates whether the control's short cut should be shown.
R	ShowShortCut (see page 533)	Indicates whether the control's short cut should be shown.
	SmallIcon (see page 534)	Indicates whether the control's small icon should be shown.
	Spacing (see page 534)	Controls (see page 5711) the separation between the control's icon and caption.
	State (see page 190)	Describes the current state of the button.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TextBounds (see page 534)	A bounding rectangle for the control caption.
	TrackButton (see page 190)	Specifies which mouse button activates the button control.
	Transparent (see page 534)	Shows whether the control is transparent or not.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

151.1.4.2 TStandardDropDownButton Methods

The methods of the TStandardDropDownButton class are listed here.

Protected Methods

	Name	Description
	DrawFrame (see page 20444)	This is DrawFrame, a member of class TStandardDropDownButton.
	GetPopupClass (see page 20444)	This is GetPopupClass, a member of class TStandardDropDownButton.

151.1.4.2.1 TStandardDropDownButton.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TStandardDropDownButton.

151.1.4.2.2 TStandardDropDownButton.GetPopupClass Method

Pascal

```
function GetPopupClass: TCustomActionBarClass; override;
```

C++

```
virtual __fastcall TCustomActionBarClass GetPopupClass();
```

Description

This is GetPopupClass, a member of class TStandardDropDownButton.

151.1.5 TStandardMenuButton Class

TStandardMenuButton is the standard ActionBand menu button class.

Class Hierarchy

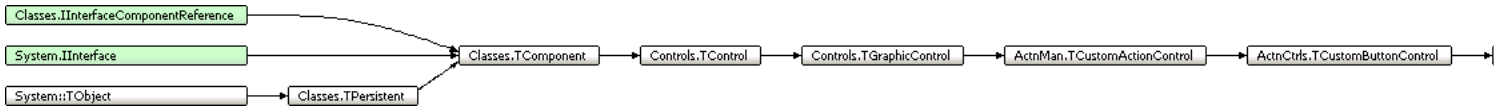

Pascal

```
TStandardMenuButton = class(TCustomMenuButton);
```

C++

```
class TStandardMenuButton : public TCustomMenuButton;
```

File

StdActnMenus

Description

TStandardMenuButton is the standard menu button class. If you would like to replace the standard menu button behaviors, use the class TCustomMenuButton as a parent class for your replacement class. To make the change for your application, set the value of MenuButtonControlClass (a global variable) to your own subclass of TCustomMenuButton instead of

TStandardMenuButton.

See Also

TCustomMenuButton (see page 790)

151.1.5.1 TStandardMenuButton Members

The following tables list the members exposed by TStandardMenuButton.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFlat (see page 178)	This is FFlat, a member of class TCustomButtonControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FState (see page 178)	This is FState, a member of class TCustomButtonControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	BeginAutoDrag (see page 517)	Begin Automatic Drag.
V	BeginAutoDrag (see page 801)	Begin Automatic Drag.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CalcLayout (see page 518)	This is CalcLayout, a member of class TCustomActionControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
	CMDesignHitTest (see page 519)	This is CMDesignHitTest, a member of class TCustomActionControl.
	CMDialogChar (see page 180)	This is CMDialogChar, a member of class TCustomButtonControl.
	CMEnabledChanged (see page 180)	This is CMEnabledChanged, a member of class TCustomButtonControl.
	CMFontChanged (see page 519)	This is CMFontChanged, a member of class TCustomActionControl.
	CMItemDropPoint (see page 519)	This is CMItemDropPoint, a member of class TCustomActionControl.
	CMItemSelected (see page 519)	This is CMItemSelected, a member of class TCustomActionControl.
	CMMouseEnter (see page 180)	This is CMMouseEnter, a member of class TCustomButtonControl.
	CMMouseEnter (see page 801)	This is CMMouseEnter, a member of class TCustomMenuButton.
	CMMouseLeave (see page 180)	This is CMMouseLeave, a member of class TCustomButtonControl.
	CMTextChanged (see page 180)	This is CMTextChanged, a member of class TCustomButtonControl.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DesignWndProc (see page 519)	This is DesignWndProc, a member of class TCustomActionControl.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
V	DoDragDrop (see page 520)	This is DoDragDrop, a member of class TCustomActionControl.
V	DoDragDrop (see page 802)	This is DoDragDrop, a member of class TCustomMenuButton.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
V	DoStartDrag (see page 520)	OnStartDrag event dispatcher.

	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 521)	OnDragOver event dispatcher.
V	DragOver (see page 802)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawBackground (see page 20454)	Draws a background image on the control.
V	DrawDesignFocus (see page 20454)	This is DrawDesignFocus, a member of class TStandardMenuButton.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawDragDropPoint (see page 522)	This is DrawDragDropPoint, a member of class TCustomActionControl.
V	DrawFrame (see page 20454)	This is DrawFrame, a member of class TStandardMenuButton.
V	DrawGlyph (see page 522)	Draws a symbol on the control.
V	DrawLargeGlyph (see page 181)	This is DrawLargeGlyph, a member of class TCustomButtonControl.
V	DrawLargeGlyph (see page 522)	This is DrawLargeGlyph, a member of class TCustomActionControl.
V	DrawSeparator (see page 522)	This is DrawSeparator, a member of class TCustomActionControl.
V	DrawShadowedText (see page 523)	This is DrawShadowedText, a member of class TCustomActionControl.
V	DrawText (see page 20455)	This is DrawText, a member of class TStandardMenuButton.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	GetAction (see page 523)	This is GetAction, a member of class TCustomActionControl.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetImageSize (see page 523)	This is GetImageSize, a member of class TCustomActionControl.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
V	GetSeparator (see page 524)	This is GetSeparator, a member of class TCustomActionControl.
V	GetShowCaption (see page 524)	This is GetShowCaption, a member of class TCustomActionControl.
V	GetShowShortCut (see page 181)	This is GetShowShortCut, a member of class TCustomButtonControl.
V	GetShowShortCut (see page 524)	This is GetShowShortCut, a member of class TCustomActionControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasGlyph (see page 524)	This is HasGlyph, a member of class TCustomActionControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsChecked (see page 525)	This is IsChecked, a member of class TCustomActionControl.
V	IsGrouped (see page 525)	This is IsGrouped, a member of class TCustomActionControl.
V	Loaded (see page 182)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.

	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 182)	OnMouseDown event dispatcher.
V	MouseDown (see page 803)	OnMouseDown event dispatcher.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
V	MouseMove (see page 183)	Respond to mouse moving over control area..
	MouseMove (see page 5773)	Respond to mouse moving over control area..
V	MouseUp (see page 183)	OnMouseUp event dispatcher.
V	MouseUp (see page 804)	OnMouseUp event dispatcher.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 184)	Render control surface.
V	Paint (see page 526)	Render control surface.
V	Paint (see page 804)	Render control surface.
V	Paint (see page 6052)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	PositionChanged (see page 526)	This is PositionChanged, a member of class TCustomActionControl.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetUsageData (see page 526)	This is ResetUsageData, a member of class TCustomActionControl.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
V	SetActionClient (see page 184)	This is SetActionClient, a member of class TCustomButtonControl.
V	SetActionClient (see page 526)	This is SetActionClient, a member of class TCustomActionControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropPoint (see page 527)	This is SetDropPoint, a member of class TCustomActionControl.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
V	SetGlyphLayout (see page 185)	This is SetGlyphLayout, a member of class TCustomButtonControl.
V	SetGlyphLayout (see page 527)	This is SetGlyphLayout, a member of class TCustomActionControl.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetSelected (see page 185)	This is SetSelected, a member of class TCustomButtonControl.
V	SetSelected (see page 527)	This is SetSelected, a member of class TCustomActionControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.

	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateSelection (see page 527)	This is UpdateSelection, a member of class TCustomActionControl.
	UpdateTextBounds (see page 527)	This is UpdateTextBounds, a member of class TCustomActionControl.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMLButtonDblClk (see page 185)	This is WMLButtonDblClk, a member of class TCustomButtonControl.
	WMSetText (see page 528)	This is WMSetText, a member of class TCustomActionControl.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GlyphPos (see page 532)	This is GlyphPos, a member of class TCustomActionControl.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	Menu (see page 805)	This is Menu, a member of class TCustomMenuButton.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	MouseInControl (see page 189)	This is MouseInControl, a member of class TCustomButtonControl.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnClick (see page 535)	Occurs when the user clicks the control.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
V	CalcBounds (see page 179)	Calculates the size of the control.
V	CalcBounds (see page 518)	Calculates the size of the control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Click (see page 179)	Executes the control's function.
V	Click (see page 518)	Executes the control's function in response to a mouse click.
V	Click (see page 801)	Executes the control's function.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 177)	Creates an instance of TCustomButtonControl (see page 168).
V	Create (see page 515)	Creates an instance of TCustomActionControl (see page 507).
V	Create (see page 799)	Creates an instance of TCustomMenuButton (see page 790).
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6051)	Creates an instance of TGraphicControl (see page 6044).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 516)	Destroys an instance of TCustomActionControl (see page 507).
V	Destroy (see page 800)	Destroys an instance of TCustomMenuButton (see page 790).
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
V	DragDrop (see page 520)	Performs actions and generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.

	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 524)	Calls the action client's Update method if the control is associated with an action client.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
V	Keyed (see page 525)	Executes the control's function in response to a keyboard action.
V	Keyed (see page 803)	Executes the button's function in response to a keyboard command.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.

	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 184)	Sets the Left, Top, Width, and Height properties all at once.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	ActionBar (see page 528)	The ActionBand object which contains the custom action control
	ActionClient (see page 529)	Refers to the control's action client
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	AllowAllUp (see page 186)	Indicates whether all the buttons in a given group can be up at the same time.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Caption (see page 186)	Specifies a text string that identifies the control to the user.
	Caption (see page 529)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Down (see page 189)	Tells whether the button is down (selected) or up (unselected).
	DropPoint (see page 532)	This is DropPoint, a member of class TCustomActionControl.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.

R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
	Flat (see page 189)	Dictates whether the button should have a 2D look instead of the usual 3D look.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GlyphLayout (see page 532)	Describes where the image should be on the control
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Margins (see page 532)	Describes the boundaries of the control for drawing purposes.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	Selected (see page 533)	Indicates whether the control is currently selected.
R	Separator (see page 533)	Indicates whether the control is a separator.
	ShowCaption (see page 189)	Indicates whether the control's caption should be shown.
R	ShowCaption (see page 533)	Indicates whether the control's caption should be shown.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
	ShowShortCut (see page 190)	Indicates whether the control's short cut should be shown.
R	ShowShortCut (see page 533)	Indicates whether the control's short cut should be shown.
	SmallIcon (see page 534)	Indicates whether the control's small icon should be shown.
	Spacing (see page 534)	Controls (see page 5711) the separation between the control's icon and caption.
	State (see page 190)	Describes the current state of the button.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TextBounds (see page 534)	A bounding rectangle for the control caption.
	TrackButton (see page 190)	Specifies which mouse button activates the button control.
	Transparent (see page 534)	Shows whether the control is transparent or not.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

151.1.5.2 TStandardMenuButton Methods

The methods of the TStandardMenuButton class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 20454)	Draws a background image on the control.
	DrawDesignFocus (see page 20454)	This is DrawDesignFocus, a member of class TStandardMenuButton.
	DrawFrame (see page 20454)	This is DrawFrame, a member of class TStandardMenuButton.
	DrawText (see page 20455)	This is DrawText, a member of class TStandardMenuButton.

151.1.5.2.1 TStandardMenuButton.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph (see page 522)

DrawText (see page 523)

151.1.5.2.2 TStandardMenuButton.DrawDesignFocus Method

Pascal

```
procedure DrawDesignFocus(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawDesignFocus(TRect PaintRect);
```

Description

This is DrawDesignFocus, a member of class TStandardMenuButton.

151.1.5.2.3 TStandardMenuButton.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TStandardMenuButton.

151.1.5.2.4 TStandardMenuButton.DrawText Method

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: String); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, AnsiString Text);
```

Description

This is DrawText, a member of class TStandardMenuButton.

151.1.6 TStandardMenuExpandBtn Class

Class Hierarchy

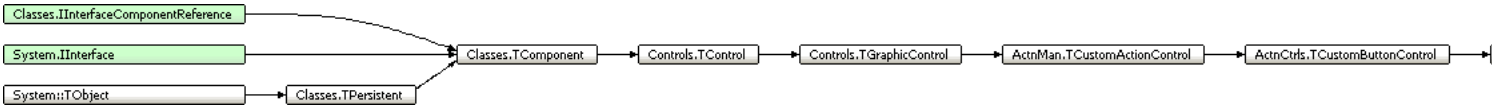

Pascal

```
TStandardMenuExpandBtn = class(TCustomMenuExpandBtn);
```

C++

```
class TStandardMenuExpandBtn : public TCustomMenuExpandBtn;
```

File

StdActnMenus

Description

This is class StdActnMenus.TStandardMenuExpandBtn.

151.1.6.1 TStandardMenuExpandBtn Members

The following tables list the members exposed by TStandardMenuExpandBtn.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFlat (see page 178)	This is FFlat, a member of class TCustomButtonControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FState (see page 178)	This is FState, a member of class TCustomButtonControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	BeginAutoDrag (see page 517)	Begin Automatic Drag.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CalcLayout (see page 518)	This is CalcLayout, a member of class TCustomActionControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
	CMDesignHitTest (see page 519)	This is CMDesignHitTest, a member of class TCustomActionControl.
	CMDialogChar (see page 180)	This is CMDialogChar, a member of class TCustomButtonControl.

	CMEnabledChanged (see page 180)	This is CMEnabledChanged, a member of class TCustomButtonControl.
	CMFontChanged (see page 519)	This is CMFontChanged, a member of class TCustomActionControl.
	CMItemDropPoint (see page 519)	This is CMItemDropPoint, a member of class TCustomActionControl.
	CMItemSelected (see page 519)	This is CMItemSelected, a member of class TCustomActionControl.
	CMMouseEnter (see page 180)	This is CMMouseEnter, a member of class TCustomButtonControl.
	CMMouseEnter (see page 238)	This is CMMouseEnter, a member of class TCustomUtilityButton.
	CMMouseLeave (see page 180)	This is CMMouseLeave, a member of class TCustomButtonControl.
	CMMouseLeave (see page 239)	This is CMMouseLeave, a member of class TCustomUtilityButton.
	CMTextChanged (see page 180)	This is CMTextChanged, a member of class TCustomButtonControl.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DesignWndProc (see page 519)	This is DesignWndProc, a member of class TCustomActionControl.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
V	DoDragDrop (see page 520)	This is DoDragDrop, a member of class TCustomActionControl.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
V	DoStartDrag (see page 520)	OnStartDrag event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 521)	OnDragOver event dispatcher.
V	DragOver (see page 815)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawArrows (see page 239)	This is DrawArrows, a member of class TCustomUtilityButton.
V	DrawBackground (see page 20464)	Draws a background image on the control.
V	DrawDesignFocus (see page 521)	This is DrawDesignFocus, a member of class TCustomActionControl.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawDragDropPoint (see page 522)	This is DrawDragDropPoint, a member of class TCustomActionControl.
V	DrawFrame (see page 20465)	This is DrawFrame, a member of class TStandardMenuExpandBtn.
V	DrawGlyph (see page 522)	Draws a symbol on the control.
V	DrawLargeGlyph (see page 181)	This is DrawLargeGlyph, a member of class TCustomButtonControl.
V	DrawLargeGlyph (see page 522)	This is DrawLargeGlyph, a member of class TCustomActionControl.
V	DrawSeparator (see page 522)	This is DrawSeparator, a member of class TCustomActionControl.
V	DrawShadowedText (see page 523)	This is DrawShadowedText, a member of class TCustomActionControl.
V	DrawText (see page 523)	Draws text on the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	GetAction (see page 523)	This is GetAction, a member of class TCustomActionControl.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.

	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetImageSize (see page 523)	This is GetImageSize, a member of class TCustomActionControl.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSeparator (see page 524)	This is GetSeparator, a member of class TCustomActionControl.
	GetShowCaption (see page 524)	This is GetShowCaption, a member of class TCustomActionControl.
	GetShowShortCut (see page 181)	This is GetShowShortCut, a member of class TCustomButtonControl.
	GetShowShortCut (see page 524)	This is GetShowShortCut, a member of class TCustomActionControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasGlyph (see page 524)	This is HasGlyph, a member of class TCustomActionControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsChecked (see page 525)	This is IsChecked, a member of class TCustomActionControl.
	IsGrouped (see page 525)	This is IsGrouped, a member of class TCustomActionControl.
	Loaded (see page 182)	Initializes the control after it is loaded from a stream.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 182)	OnMouseDown event dispatcher.
	MouseDown (see page 239)	OnMouseDown event dispatcher.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 183)	Respond to mouse moving over control area..
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 183)	OnMouseUp event dispatcher.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	OnDelay (see page 240)	This is OnDelay, a member of class TCustomUtilityButton.
	Paint (see page 184)	Render control surface.
	Paint (see page 240)	Render control surface.
	Paint (see page 526)	Render control surface.
	Paint (see page 6052)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionChanged (see page 526)	This is PositionChanged, a member of class TCustomActionControl.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.

	ResetUsageData (see page 526)	This is ResetUsageData, a member of class TCustomActionControl.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
V	SetActionClient (see page 184)	This is SetActionClient, a member of class TCustomButtonControl.
V	SetActionClient (see page 526)	This is SetActionClient, a member of class TCustomActionControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropPoint (see page 527)	This is SetDropPoint, a member of class TCustomActionControl.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
V	SetGlyphLayout (see page 185)	This is SetGlyphLayout, a member of class TCustomButtonControl.
V	SetGlyphLayout (see page 527)	This is SetGlyphLayout, a member of class TCustomActionControl.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetSelected (see page 185)	This is SetSelected, a member of class TCustomButtonControl.
V	SetSelected (see page 527)	This is SetSelected, a member of class TCustomActionControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateSelection (see page 527)	This is UpdateSelection, a member of class TCustomActionControl.
	UpdateTextBounds (see page 527)	This is UpdateTextBounds, a member of class TCustomActionControl.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMLButtonDblClk (see page 185)	This is WMLButtonDblClk, a member of class TCustomButtonControl.
	WMSetText (see page 528)	This is WMSetText, a member of class TCustomActionControl.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.

	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GlyphPos (see page 532)	This is GlyphPos, a member of class TCustomActionControl.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	Menu (see page 816)	This is Menu, a member of class TCustomMenuExpandBtn.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	MouseInControl (see page 189)	This is MouseInControl, a member of class TCustomButtonControl.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnClick (see page 243)	Occurs when the user clicks the control.
	OnClick (see page 535)	Occurs when the user clicks the control.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
V	CalcBounds (see page 179)	Calculates the size of the control.
V	CalcBounds (see page 518)	Calculates the size of the control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Click (see page 179)	Executes the control's function.
V	Click (see page 238)	Executes the control's function.
V	Click (see page 518)	Executes the control's function in response to a mouse click.

	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 177)	Creates an instance of TCustomButtonControl (see page 168).
V	Create (see page 237)	Creates an instance of TCustomButtonControl.
V	Create (see page 515)	Creates an instance of TCustomActionControl (see page 507).
V	Create (see page 814)	Creates an instance of TCustomButtonControl.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6051)	Creates an instance of TGraphicControl (see page 6044).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 237)	Destroys an instance of TCustomActionControl.
V	Destroy (see page 516)	Destroys an instance of TCustomActionControl (see page 507).
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
V	DragDrop (see page 520)	Performs actions and generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.

	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 524)	Calls the action client's Update method if the control is associated with an action client.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	Keyed (see page 525)	Executes the control's function in response to a keyboard action.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 184)	Sets the Left, Top, Width, and Height properties all at once.
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	ActionBar (see page 528)	The ActionBand object which contains the custom action control

	ActionClient (see page 529)	Refers to the control's action client
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	AllowAllUp (see page 186)	Indicates whether all the buttons in a given group can be up at the same time.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	ArrowSize (see page 240)	This is ArrowSize, a member of class TCustomUtilityButton.
	ArrowType (see page 241)	This is ArrowType, a member of class TCustomUtilityButton.
	AutoScroll (see page 241)	This is AutoScroll, a member of class TCustomUtilityButton.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Caption (see page 186)	Specifies a text string that identifies the control to the user.
	Caption (see page 529)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 241)	Specifies the background color of the control.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Direction (see page 242)	This is Direction, a member of class TCustomUtilityButton.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Down (see page 189)	Tells whether the button is down (selected) or up (unselected).
	DropPoint (see page 532)	This is DropPoint, a member of class TCustomActionControl.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
	Flat (see page 189)	Dictates whether the button should have a 2D look instead of the usual 3D look.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GlyphLayout (see page 532)	Describes where the image should be on the control
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Margins (see page 532)	Describes the boundaries of the control for drawing purposes.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	RepeatRate (see page 242)	This is RepeatRate, a member of class TCustomUtilityButton.
	Selected (see page 533)	Indicates whether the control is currently selected.
R	Separator (see page 533)	Indicates whether the control is a separator.

	ShowCaption (see page 189)	Indicates whether the control's caption should be shown.
R	ShowCaption (see page 533)	Indicates whether the control's caption should be shown.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
	ShowShortCut (see page 190)	Indicates whether the control's short cut should be shown.
R	ShowShortCut (see page 533)	Indicates whether the control's short cut should be shown.
	SmallIcon (see page 534)	Indicates whether the control's small icon should be shown.
	Spacing (see page 534)	Controls (see page 5711) the separation between the control's icon and caption.
	State (see page 190)	Describes the current state of the button.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TextBounds (see page 534)	A bounding rectangle for the control caption.
	TrackButton (see page 190)	Specifies which mouse button activates the button control.
	Transparent (see page 534)	Shows whether the control is transparent or not.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

151.1.6.2 TStandardMenuExpandBtn Methods

The methods of the TStandardMenuExpandBtn class are listed here.

Protected Methods

	Name	Description
V	DrawBackground (see page 20464)	Draws a background image on the control.
V	DrawFrame (see page 20465)	This is DrawFrame, a member of class TStandardMenuExpandBtn.

151.1.6.2.1 TStandardMenuExpandBtn.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph (see page 522)

DrawText (see page 523)

151.1.6.2.2 TStandardMenuExpandBtn.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TStandardMenuExpandBtn.

151.1.7 TStandardMenuItem Class

TStandardMenuItem is the default class for menu items in an action band.

Class Hierarchy

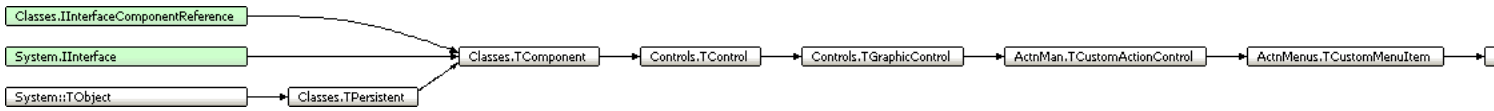

Pascal

```
TStandardMenuItem = class(TCustomMenuItem);
```

C++

```
class TStandardMenuItem : public TCustomMenuItem;
```

File

StdActnMenus

Description

TStandardMenuItem is the class that action bands use to generate child controls that are menu items. Do not create instances of TStandardMenuItem. The action manager automatically handles the creating of TStandardMenuItem instances to represent the actions it contains.

If you want to replace the class used for menu items, derive your own replacement class from TCustomMenuItem. To substitute your replacement class, set the value of the global MenuItemControlClass variable to your TCustomMenuItem descendant.

See Also

TStandardMenuButton (see page 20444)

TStandardMenuPopup (see page 20478)

MenuItemControlClass

151.1.7.1 TStandardMenuItem Members

The following tables list the members exposed by TStandardMenuItem.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FNoPrefix (see page 826)	This is FNoPrefix, a member of class TCustomMenuItem.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	BeginAutoDrag (see page 517)	Begin Automatic Drag.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CalcLayout (see page 518)	This is CalcLayout, a member of class TCustomActionControl.
V	CalcLayout (see page 827)	This is CalcLayout, a member of class TCustomMenuItem.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
	CMDesignHitTest (see page 519)	This is CMDesignHitTest, a member of class TCustomActionControl.
	CMFontChanged (see page 519)	This is CMFontChanged, a member of class TCustomActionControl.
	CMItemDropPoint (see page 519)	This is CMItemDropPoint, a member of class TCustomActionControl.
	CMItemSelected (see page 519)	This is CMItemSelected, a member of class TCustomActionControl.
	CMTextchanged (see page 20475)	Responds when this menu item's caption changes.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DesignWndProc (see page 519)	This is DesignWndProc, a member of class TCustomActionControl.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
V	DoDragDrop (see page 520)	This is DoDragDrop, a member of class TCustomActionControl.
V	DoDragDrop (see page 828)	This is DoDragDrop, a member of class TCustomMenuItem.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
V	DoStartDrag (see page 520)	OnStartDrag event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 521)	OnDragOver event dispatcher.
V	DragOver (see page 828)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.

	DrawBackground (see page 20475)	Draws the background of the menu item.
	DrawDesignFocus (see page 20475)	This is DrawDesignFocus, a member of class TStandardMenuItem.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawDragDropPoint (see page 522)	This is DrawDragDropPoint, a member of class TCustomActionControl.
	DrawEdge (see page 20476)	This is DrawEdge, a member of class TStandardMenuItem.
	DrawGlyph (see page 20476)	Draws a symbol on the menu item.
	DrawGlyphFrame (see page 20476)	This is DrawGlyphFrame, a member of class TStandardMenuItem.
	DrawLargeGlyph (see page 20476)	This is DrawLargeGlyph, a member of class TStandardMenuItem.
	DrawSeparator (see page 20477)	Draws a separator.
	DrawShadowedText (see page 523)	This is DrawShadowedText, a member of class TCustomActionControl.
	DrawShadowedText (see page 829)	This is DrawShadowedText, a member of class TCustomMenuItem.
	DrawSubMenuGlyph (see page 829)	This is DrawSubMenuGlyph, a member of class TCustomMenuItem.
	DrawText (see page 20477)	Draws the menu item's text.
	DrawUnusedEdges (see page 830)	This is DrawUnusedEdges, a member of class TCustomMenuItem.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	GetAction (see page 523)	This is GetAction, a member of class TCustomActionControl.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetImageSize (see page 523)	This is GetImageSize, a member of class TCustomActionControl.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSeparator (see page 524)	This is GetSeparator, a member of class TCustomActionControl.
	GetShowCaption (see page 524)	This is GetShowCaption, a member of class TCustomActionControl.
	GetShowShortCut (see page 524)	This is GetShowShortCut, a member of class TCustomActionControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasGlyph (see page 524)	This is HasGlyph, a member of class TCustomActionControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsChecked (see page 525)	This is IsChecked, a member of class TCustomActionControl.
	IsGrouped (see page 525)	This is IsGrouped, a member of class TCustomActionControl.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 830)	OnMouseUp event dispatcher.

	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 526)	Render control surface.
V	Paint (see page 831)	Render control surface.
V	Paint (see page 6052)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	PositionChanged (see page 526)	This is PositionChanged, a member of class TCustomActionControl.
V	PositionChanged (see page 831)	This is PositionChanged, a member of class TCustomMenuItem.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetUsageData (see page 526)	This is ResetUsageData, a member of class TCustomActionControl.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
V	SetActionClient (see page 526)	This is SetActionClient, a member of class TCustomActionControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropPoint (see page 527)	This is SetDropPoint, a member of class TCustomActionControl.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
V	SetGlyphLayout (see page 527)	This is SetGlyphLayout, a member of class TCustomActionControl.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetSelected (see page 527)	This is SetSelected, a member of class TCustomActionControl.
V	SetSelected (see page 831)	This is SetSelected, a member of class TCustomMenuItem.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateSelection (see page 527)	This is UpdateSelection, a member of class TCustomActionControl.
	UpdateTextBounds (see page 527)	This is UpdateTextBounds, a member of class TCustomActionControl.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMSetText (see page 528)	This is WMSetText, a member of class TCustomActionControl.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GlyphPos (see page 532)	This is GlyphPos, a member of class TCustomActionControl.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	Menu (see page 832)	This is Menu, a member of class TCustomMenuItem.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	MouseSelected (see page 832)	This is MouseSelected, a member of class TCustomMenuItem.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ShortCutBounds (see page 832)	This is ShortCutBounds, a member of class TCustomMenuItem.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnClick (see page 535)	Occurs when the user clicks the control.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.

	CalcBounds (see page 518)	Calculates the size of the control.
	CalcBounds (see page 827)	Calculates the size of the menu item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Click (see page 518)	Executes the control's function in response to a mouse click.
	Click (see page 827)	Executes the control's function.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Create (see page 515)	Creates an instance of TCustomActionControl (see page 507).
	Create (see page 825)	Creates an instance of TCustomMenuItem (see page 816).
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 5731)	Creates an instance of TControl (see page 5724).
	Create (see page 6051)	Creates an instance of TGraphicControl (see page 6044).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 516)	Destroys an instance of TCustomActionControl (see page 507).
	Destroy (see page 825)	Destroys an instance of TCustomMenuItem (see page 816).
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 520)	Performs actions and generates an OnDragDrop event.
	DragDrop (see page 828)	Performs actions and generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 524)	Calls the action client's Update method if the control is associated with an action client.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
V	Keyed (see page 525)	Executes the control's function in response to a keyboard action.
V	Keyed (see page 830)	Executes the menu item's function in response to a user's keyboard command.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.

V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	ActionBar (see page 528)	The ActionBand object which contains the custom action control
	ActionClient (see page 529)	Refers to the control's action client
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Caption (see page 529)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DropPoint (see page 532)	This is DropPoint, a member of class TCustomActionControl.
	Edges (see page 832)	Dictates which edges are drawn when the menu item is drawn.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GlyphLayout (see page 532)	Describes where the image should be on the control
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Margins (see page 532)	Describes the boundaries of the control for drawing purposes.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	Selected (see page 533)	Indicates whether the control is currently selected.

R	Separator (see page 533)	Indicates whether the control is a separator.
R	ShowCaption (see page 533)	Indicates whether the control's caption should be shown.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	ShowShortCut (see page 533)	Indicates whether the control's short cut should be shown.
	SmallIcon (see page 534)	Indicates whether the control's small icon should be shown.
	Spacing (see page 534)	Controls (see page 5711) the separation between the control's icon and caption.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TextBounds (see page 534)	A bounding rectangle for the control caption.
	Transparent (see page 534)	Shows whether the control is transparent or not.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

151.1.7.2 TStandardMenuItem Methods

The methods of the TStandardMenuItem class are listed here.

Protected Methods

	Name	Description
	CMTextchanged (see page 20475)	Responds when this menu item's caption changes.
V	DrawBackground (see page 20475)	Draws the background of the menu item.
V	DrawDesignFocus (see page 20475)	This is DrawDesignFocus, a member of class TStandardMenuItem.
V	DrawEdge (see page 20476)	This is DrawEdge, a member of class TStandardMenuItem.
V	DrawGlyph (see page 20476)	Draws a symbol on the menu item.
	DrawGlyphFrame (see page 20476)	This is DrawGlyphFrame, a member of class TStandardMenuItem.
V	DrawLargeGlyph (see page 20476)	This is DrawLargeGlyph, a member of class TStandardMenuItem.
V	DrawSeparator (see page 20477)	Draws a separator.
V	DrawText (see page 20477)	Draws the menu item's text.

151.1.7.2.1 TStandardMenuItem.CMTextchanged Method

Responds when this menu item's caption changes.

Pascal

```
procedure CMTextchanged(var Message: TMessage); message CM_TEXTCHANGED;
```

C++

```
__fastcall CMTextchanged(TMessage Message);
```

Description

This method is called automatically in response to Windows messages containing a notification that the text of this menu item has changed.

Message represents the Windows message that notifies this menu item of a change to its text.

CMTextchanged recalculates the string it displays when the menu item is not underlining its accelerator key.

See Also

PersistentHotKeys (see page 489)

151.1.7.2.2 TStandardMenuItem.DrawBackground Method

Draws the background of the menu item.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

This method is called automatically when the menu item needs to render its background (everything except the text and glyph).

PaintRect indicates the dimensions of the menu item on its canvas.

DrawBackground paints the background of the menu item so that its color reflects whether the item currently has focus, adjusting PaintRect to accommodate any glyph or checkmark that appears on the menu item.

See Also

DrawBackground (see page 521)

DrawGlyph (see page 20476)

DrawText (see page 20477)

151.1.7.2.3 TStandardMenuItem.DrawDesignFocus Method

Pascal

```
procedure DrawDesignFocus(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawDesignFocus(TRect PaintRect);
```

Description

This is DrawDesignFocus, a member of class TStandardMenuItem.

151.1.7.2.4 TStandardMenuItem.DrawEdge Method

Pascal

```
procedure DrawEdge(Edges: TMenuEdges); override;
```

C++

```
virtual __fastcall DrawEdge(TMenuEdges Edges);
```

Description

This is DrawEdge, a member of class TStandardMenuItem.

151.1.7.2.5 TStandardMenuItem.DrawGlyph Method

Draws a symbol on the menu item.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

DrawGlyph is called automatically when the menu item needs to render its glyph. This occurs if the menu item is checked or if the associated action client item specifies a glyph for this menu item using its ImageIndex property.

Location is the position on this menu item's canvas for the upper left corner of the glyph.

See Also

DrawGlyph (see page 522)

DrawBackground (see page 20475)

DrawText (see page 20477)

ImageIndex (see page 385)

151.1.7.2.6 TStandardMenuItem.DrawGlyphFrame Method

Pascal

```
procedure DrawGlyphFrame(const Location: TPoint);
```

C++

```
__fastcall DrawGlyphFrame(const TPoint Location);
```

Description

This is DrawGlyphFrame, a member of class TStandardMenuItem.

151.1.7.2.7 TStandardMenuItem.DrawLargeGlyph Method

Pascal

```
procedure DrawLargeGlyph(Location: TPoint); override;
```

C++

```
virtual __fastcall DrawLargeGlyph(TPoint Location);
```

Description

This is DrawLargeGlyph, a member of class TStandardMenuItem.

151.1.7.2.8 TStandardMenuItem.DrawSeparator Method

Draws a separator.

Pascal

```
procedure DrawSeparator(const Offset: Integer); override;
```

C++

```
virtual __fastcall DrawSeparator(const int Offset);
```

Description

DrawSeparator is called automatically when the menu item represents a separator bar in the menu and it needs to paint itself. This method paints the separator bar.

Offset indicates the offset of the separator, in pixels, from the top of the menu item. DrawSeparator ignores this value and always uses a fixed value for menu items.

151.1.7.2.9 TStandardMenuItem.DrawText Method

Draws the menu item's text.

Pascal

```
procedure DrawText(var Rect: TRect; var Flags: Cardinal; Text: String); override;
```

C++

```
virtual __fastcall DrawText(TRect Rect, unsigned Flags, AnsiString Text);
```

Description

DrawText is called automatically when the menu item needs to render its text.

ARect indicates the rectangle on the menu item's canvas where the text label should appear.

Flags supplies the value to use for the TextFlags property of the canvas.

Text is the text of the menu item.

DrawText checks whether the menu item should display underlines on accelerator keys, and draws the text accordingly, using a font color that reflects whether the menu item is selected and whether it is enabled.

See Also

DrawText (see page 523)

DrawBackground (see page 20475)

DrawGlyph (see page 20476)

PersistentHotKeys (see page 489)

TextFlags (see page 14495)

151.1.8 TStandardMenuPopup Class

TStandardMenuPopup is the default menu popup style.

Class Hierarchy

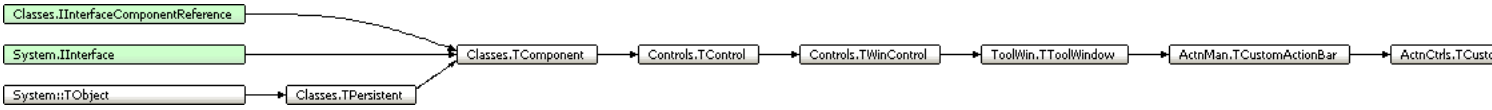

Pascal

```
TStandardMenuPopup = class(TCustomActionPopupMenu);
```

C++

```
class TStandardMenuPopup : public TCustomActionPopupMenu;
```

File

StdActnMenus

Description

TStandardMenuPopup is the default menu popup style. If you would like to replace it with a style of your own, subclass TCustomActionPopupMenu to provide your own behaviors. To make the change for your application, set the value of MenuPopupClass (a global variable) to your own subclass of TCustomActionPopupMenu instead of TStandardMenuPopup.

See Also

TActionMainMenuBar (see page 585)

TStandardMenuButton (see page 20444)

TStandardMenuItem (see page 20465)

151.1.8.1 TStandardMenuPopup Members

The following tables list the members exposed by TStandardMenuPopup.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FCachedMenu (see page 699)	This is FCachedMenu, a member of class TCustomActionMenuBar.
	FChildMenu (see page 699)	This is FChildMenu, a member of class TCustomActionMenuBar.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefaultColorMap (see page 460)	This is FDefaultColorMap, a member of class TCustomActionBar.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExpandTimer (see page 700)	This is FExpandTimer, a member of class TCustomActionMenuBar.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FPopupTimer (see page 700)	This is FPopupTimer, a member of class TCustomActionMenuBar.
	FSavedWrapState (see page 460)	This is FSavedWrapState, a member of class TCustomActionBar.
	FSelectedItem (see page 700)	This is FSelectedItem, a member of class TCustomActionMenuBar.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.

	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AddEdges (see page 737)	This is AddEdges, a member of class TCustomActionPopupMenu.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 462)	Aligns any controls for which the control is the parent within a specified area of the control.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	Animate (see page 701)	This is Animate, a member of class TCustomActionMenuBar.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	AutoSizingChanged (see page 462)	This is AutoSizingChanged, a member of class TCustomActionBar.
	BackgroundSize (see page 462)	This is BackgroundSize, a member of class TCustomActionBar.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcInsertPosition (see page 462)	This is CalcInsertPosition, a member of class TCustomActionBar.
	CanAutoSize (see page 20492)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CleanupStack (see page 701)	This is CleanupStack, a member of class TCustomActionMenuBar.
	Clear (see page 463)	This is Clear, a member of class TCustomActionBar.
	ClearSubMenus (see page 702)	This is ClearSubMenus, a member of class TCustomActionMenuBar.
	Click (see page 5743)	Respond to user click.
	CMBorderChanged (see page 23285)	This is CMBorderChanged, a member of class TToolWindow.
	CMCtl3DChanged (see page 23286)	This is CMCtl3DChanged, a member of class TToolWindow.
	CMMouseLeave (see page 738)	This is CMMouseLeave, a member of class TCustomActionPopupMenu.
	CMVisibleChanged (see page 738)	This is CMVisibleChanged, a member of class TCustomActionPopupMenu.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateControl (see page 463)	This is CreateControl, a member of class TCustomActionBar.
	CreateControl (see page 702)	This is CreateControl, a member of class TCustomActionMenuBar.
	CreateControl (see page 738)	This is CreateControl, a member of class TCustomActionPopupMenu.
	CreateControls (see page 463)	This is CreateControls, a member of class TCustomActionBar.
	CreateControls (see page 738)	This is CreateControls, a member of class TCustomActionPopupMenu.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 739)	Initializes a window-creation parameter data structure.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreatePopup (see page 702)	This is CreatePopup, a member of class TCustomActionMenuBar.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 463)	Returns placement order of custom-aligned child control.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.

	CustomAlignPosition (see page 464)	Returns aligned position of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 465)	This is DesignWndProc, a member of class TCustomActionBar.
	DesignWndProc (see page 739)	This is DesignWndProc, a member of class TCustomActionPopupMenu.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DisplayShadow (see page 739)	This is DisplayShadow, a member of class TCustomActionPopupMenu.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoControlCreated (see page 465)	This is DoControlCreated, a member of class TCustomActionBar.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoDropActions (see page 465)	This is DoDropActions, a member of class TCustomActionBar.
	DoDropCategory (see page 465)	This is DoDropCategory, a member of class TCustomActionBar.
	DoDropItem (see page 466)	This is DoDropItem, a member of class TCustomActionBar.
	DoEndDock (see page 146)	OnEndDock event dispatcher.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetControlClass (see page 466)	This is DoGetControlClass, a member of class TCustomActionBar.
	DoGetPopupClass (see page 703)	This is DoGetPopupClass, a member of class TCustomActionMenuBar.
	DoItemClicked (see page 703)	This is DoItemClicked, a member of class TCustomActionMenuBar.
	DoItemKeyed (see page 703)	This is DoItemKeyed, a member of class TCustomActionMenuBar.
	DoItemSelected (see page 703)	This is DoItemSelected, a member of class TCustomActionMenuBar.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMenuDelay (see page 703)	This is DoMenuDelay, a member of class TCustomActionMenuBar.
	DoMouseIdle (see page 704)	This is DoMouseIdle, a member of class TCustomActionMenuBar.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoneMenuLoop (see page 704)	This is DoneMenuLoop, a member of class TCustomActionMenuBar.
	DoPopup (see page 704)	This is DoPopup, a member of class TCustomActionMenuBar.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 147)	OnStartDock event dispatcher.

	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragOver (see page 466)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
V	DrawBackground (see page 467)	This is DrawBackground, a member of class TCustomActionBar.
V	DrawBackground (see page 740)	This is DrawBackground, a member of class TCustomActionPopupMenu.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
V	ExecAction (see page 704)	This is ExecAction, a member of class TCustomActionMenuBar.
V	ExecAction (see page 740)	This is ExecAction, a member of class TCustomActionPopupMenu.
V	Expand (see page 704)	Causes the menu bar to show all items, including the hidden ones.
	ExpandClick (see page 740)	This is ExpandClick, a member of class TCustomActionPopupMenu.
	ExpandTimer (see page 705)	This is ExpandTimer, a member of class TCustomActionMenuBar.
	FindAccelItem (see page 467)	This is FindAccelItem, a member of class TCustomActionBar.
	FindItemAt (see page 468)	This is FindItemAt, a member of class TCustomActionBar.
	FindItemCaption (see page 468)	This is FindItemCaption, a member of class TCustomActionBar.
V	FindLast (see page 468)	This is FindLast, a member of class TCustomActionBar.
	FindLeastUsedItem (see page 469)	This is FindLeastUsedItem, a member of class TCustomActionBar.
	FindNearestControl (see page 469)	This is FindNearestControl, a member of class TCustomActionBar.
V	FindNext (see page 470)	This is FindNext, a member of class TCustomActionBar.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FindNextVisibleItem (see page 470)	This is FindNextVisibleItem, a member of class TCustomActionBar.
V	FindPrevious (see page 470)	This is FindPrevious, a member of class TCustomActionBar.
	FindPreviousVisibleItem (see page 470)	This is FindPreviousVisibleItem, a member of class TCustomActionBar.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
V	GetActionControl (see page 471)	This is GetActionControl, a member of class TCustomActionBar.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
V	GetBannerWidth (see page 471)	This is GetBannerWidth, a member of class TCustomActionBar.
V	GetBarHeight (see page 471)	This is GetBarHeight, a member of class TCustomActionBar.
V	GetBarWidth (see page 471)	This is GetBarWidth, a member of class TCustomActionBar.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 741)	Returns the value of the ClientRect property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlClass (see page 471)	This is GetControlClass, a member of class TCustomActionBar.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDefaultColorMapClass (see page 472)	This is GetDefaultColorMapClass, a member of class TCustomActionBar.
	GetDefaultSounds (see page 705)	This is GetDefaultSounds, a member of class TCustomActionMenuBar.
V	GetDesignMode (see page 472)	This is GetDesignMode, a member of class TCustomActionBar.
V	GetDesignMode (see page 705)	This is GetDesignMode, a member of class TCustomActionMenuBar.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.

	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetExpandBtnClass (see page 20493)	This is GetExpandBtnClass, a member of class TStandardMenuPopup.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 147)	Returns the class used for the control when it is floating.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetMouseHoverItem (see page 705)	This is GetMouseHoverItem, a member of class TCustomActionMenuBar.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupClass (see page 706)	This is GetPopupClass, a member of class TCustomActionMenuBar.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetPopupMenuClass (see page 472)	This is GetPopupMenuClass, a member of class TCustomActionBar.
	GetSelected (see page 706)	This is GetSelected, a member of class TCustomActionMenuBar.
	GetSiteInfo (see page 147)	Generates an OnGetSiteInfo event
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasItems (see page 472)	This is HasItems, a member of class TCustomActionBar.
	HideShadow (see page 741)	This is HideShadow, a member of class TCustomActionPopupMenu.
	Idle (see page 706)	This is Idle, a member of class TCustomActionMenuBar.
	InitMenuLoop (see page 706)	This is InitMenuLoop, a member of class TCustomActionMenuBar.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsDesignMsg (see page 707)	This is IsDesignMsg, a member of class TCustomActionMenuBar.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	ItemCount (see page 473)	This is ItemCount, a member of class TCustomActionBar.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 148)	Initializes the control after it is loaded from a stream.
	Loaded (see page 473)	Initializes the control after it is loaded from a stream.
	Loaded (see page 707)	Initializes the control after it is loaded from a stream.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 707)	This is MouseActivate, a member of class TCustomActionMenuBar.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	NCPaint (see page 20493)	This is NCPaint, a member of class TStandardMenuPopup.
	NewPopup (see page 707)	This is NewPopup, a member of class TCustomActionMenuBar.
	NewPopup (see page 741)	This is NewPopup, a member of class TCustomActionPopupMenu.
	Notification (see page 473)	Responds to notifications that components are being created or destroyed.

	Notification (see page 708)	Responds to notifications that components are being created or destroyed.
	Notification (see page 742)	Responds to notifications that components are being created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 474)	This is Paint, a member of class TCustomActionBar.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 474)	Renders the image of a windowed control.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	PositionPopup (see page 20493)	This is PositionPopup, a member of class TStandardMenuPopup.
	ProcessMenuLoop (see page 708)	This is ProcessMenuLoop, a member of class TCustomActionMenuBar.
	ProcessMessage (see page 708)	This is ProcessMessage, a member of class TCustomActionMenuBar.
	ProcessMessages (see page 709)	This is ProcessMessages, a member of class TCustomActionMenuBar.
	ProcessMouseMsg (see page 709)	This is ProcessMouseMsg, a member of class TCustomActionMenuBar.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Reset (see page 475)	This is Reset, a member of class TCustomActionBar.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Select (see page 709)	This is Select, a member of class TCustomActionMenuBar.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectItem (see page 710)	This is SelectItem, a member of class TCustomActionMenuBar.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActionClient (see page 475)	This is SetActionClient, a member of class TCustomActionBar.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetAutoSizing (see page 475)	This is SetAutoSizing, a member of class TCustomActionBar.
	SetBiDiMode (see page 475)	Sets the control's bi-directional mode.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetColorMap (see page 476)	This is SetColorMap, a member of class TCustomActionBar.

	SetColorMap (see page 743)	This is SetColorMap, a member of class TCustomActionPopupMenu.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignMode (see page 476)	This is SetDesignMode, a member of class TCustomActionBar.
	SetDesignMode (see page 710)	Specifies whether the whether the action band is being customized or being used in the IDE.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetOrientation (see page 477)	This is SetOrientation, a member of class TCustomActionBar.
	SetOrientation (see page 743)	This is SetOrientation, a member of class TCustomActionPopupMenu.
	SetParent (see page 710)	Sets the parent of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetPersistentHotKeys (see page 477)	This is SetPersistentHotKeys, a member of class TCustomActionBar.
	SetPersistentHotKeys (see page 711)	This is SetPersistentHotKeys, a member of class TCustomActionMenuBar.
	SetSpacing (see page 477)	This is SetSpacing, a member of class TCustomActionBar.
	SetupDefaultColorMap (see page 477)	This is SetupDefaultColorMap, a member of class TCustomActionBar.
	SetupExpandBtn (see page 743)	This is SetupExpandBtn, a member of class TCustomActionPopupMenu.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 478)	Provides the interface for a method that responds to changes in the control's visibility.
	VisibleChanging (see page 743)	Provides the interface for a method that responds to changes in the control's visibility.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.

	WMKeyDown (see page 744)	This is WMKeyDown, a member of class TCustomActionPopupMenu.
	WMNCCalcSize (see page 23286)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCPaint (see page 23286)	This is WMNCPaint, a member of class TToolWindow.
	WMPrint (see page 744)	This is WMPrint, a member of class TCustomActionPopupMenu.
	WMPrintClient (see page 744)	This is WMPrintClient, a member of class TCustomActionPopupMenu.
V	WndProc (see page 711)	Provides specific message responses for the control.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AnimateDuration (see page 716)	This is AnimateDuration, a member of class TCustomActionMenuBar.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DelayItem (see page 717)	This is DelayItem, a member of class TCustomActionMenuBar.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Expandable (see page 717)	This is Expandable, a member of class TCustomActionMenuBar.
	Expanded (see page 718)	This is Expanded, a member of class TCustomActionMenuBar.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
R	HRowCount (see page 488)	Indicates the number of columns of items that the action band displays.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	Items (see page 488)	Lists the action clients that correspond to the child controls of this action band.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	MouseControl (see page 718)	This is MouseControl, a member of class TCustomActionMenuBar.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentMenu (see page 719)	This is ParentMenu, a member of class TCustomActionMenuBar.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
R	PopupStack (see page 719)	This is PopupStack, a member of class TCustomActionMenuBar.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.

R	Selected (see page 719)	This is Selected, a member of class TCustomActionMenuBar.
	Text (see page 5842)	Contains a text string associated with the control.
R	VRowCount (see page 490)	Indicates the number of rows of items that the action band displays.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
R	Widest (see page 491)	This is Widest, a member of class TCustomActionBar.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnControlCreated (see page 491)	Occurs when a control is created on the action band.
	OnEnterMenuLoop (see page 720)	Occurs when an ActionBand menu loop has begun.
	OnExitMenuLoop (see page 721)	Occurs when an ActionBand menu loop has ended and the menu is closed.
	OnGetControlClass (see page 492)	Occurs when an item is to be placed on the action band.
	OnGetPopupClass (see page 721)	Occurs prior to an ActionBand popupmenu appearing on screen.
	OnPaint (see page 492)	Occurs when the action band is painted.
	OnPopup (see page 721)	Occurs just before the pop-up menu appears.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
V	CalcDockedWidth (see page 146)	This is CalcDockedWidth, a member of class TCustomActionDockBar.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	CloseMenu (see page 702)	Closes the menu currently being displayed.
V	CloseMenu (see page 737)	Closes the menu currently being displayed.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 145)	Creates and initializes a TCustomActionDockBar (see page 133) instance.
V	Create (see page 459)	Creates and initializes a TCustomActionBar (see page 447) instance.
V	Create (see page 698)	Creates and initializes a TCustomActionMenuBar (see page 685) instance.
V	Create (see page 735)	Creates and initializes a TCustomActionPopupMenu (see page 721) instance.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 23285)	Creates and initializes a TToolWindow (see page 23274) instance.

	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 459)	Destroys an instance of TCustomActionBar (see page 447).
V	Destroy (see page 698)	Destroys an instance of TCustomActionMenuBar (see page 685).
V	Destroy (see page 736)	Destroys an instance of TCustomActionMenuBar (see page 685).
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Dock (see page 146)	Used internally to dock the action band.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
V	DragDrop (see page 466)	Generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	Expand (see page 740)	Performs the same kinds of functions as the inherited method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
V	FindFirst (see page 467)	Finds the first TActionClientItem (see page 374) in the Items (see page 488) list.
	FindFirstVisibleItem (see page 468)	Finds the first visible TActionClientItem (see page 374) in the Items (see page 488) list.
	FindLastVisibleItem (see page 469)	Finds the last visibleTActionClientItem in the Items (see page 488) list.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	Popup (see page 742)	Causes the popup menu to appear.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	RecreateControls (see page 475)	Destroys and recreates all of the action band's action client controls.
	RecreateControls (see page 709)	Destroys and recreates all of the action band's action client controls.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.

	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 476)	Sets the bounds based on whether the popup is expanded or not
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Style (see page 477)	Identifies the style object that gives the application its default look and feel.
V	Style (see page 711)	Retrieves the style used by the ActionBand menu that started the menu loop so the submenus have a consistent appearance.
V	TrackMenu (see page 711)	Executes the action associated with the menu item.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	ActionClient (see page 479)	Specifies the action client associated with this action band.
R	ActionControls (see page 479)	Lists the controls on the action band.
	ActionManager (see page 480)	Specifies the action manager component that is associated with the action band.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	AllowHiding (see page 480)	Determines whether the action band can be hidden.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AnimationStyle (see page 716)	Specifies the type of animation the menu bar will display as it opens.
	AutoFocus (see page 717)	This is AutoFocus, a member of class TCustomActionMenuBar.
	AutoSizing (see page 480)	Specifies whether the action band adjusts its size automatically to fit around its child controls.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 481)	Represents the drawing surface of the action band.
	Caption (see page 481)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 483)	Specifies the background color of the control.
	ColorMap (see page 484)	Indicates the colormap for individual action band components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ContextBar (see page 485)	Indicates whether items shown in this action band represent the items in a popup menu.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Designable (see page 485)	Specifies whether the action band can be customized (used internally by the IDE).
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignMode (see page 486)	Specifies whether the action band is being customized or being used in the IDE.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EdgeBorders (see page 23287)	Determines which sides of the control have visible borders.
	EdgeInner (see page 23287)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 23288)	Determines the border style for the control's outer edge.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
	ExpandBtn (see page 744)	Refers to the popup menu's expand button.
	ExpandDelay (see page 717)	Specifies, in milliseconds, the delay period that a menu action band uses before showing hidden or unused items.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Font (see page 486)	Controls the attributes of text written on or in the control.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HorzMargin (see page 487)	Specifies the width of the horizontal margins in pixels.
	HorzSeparator (see page 487)	Indicates whether a horizontal separator bar appears between item lines.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	InMenuLoop (see page 718)	Indicates whether items in the menu can be selected.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	Orientation (see page 488)	Specifies how the action band lays out its child items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentControl (see page 718)	Refers to the parent object of the TCustomActionBarControl.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PersistentHotKeys (see page 489)	Specifies whether keyboard accelerator shortcuts are always displayed.
	RootMenu (see page 719)	Refers to the root menu of the ActionBand menu.
	Shadow (see page 745)	Refers to the popup menu's shadow.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.

R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	Spacing (see page 489)	Determines where the image and text appear on an action band's rendered control that represents the action client.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	UseSystemFont (see page 720)	Specifies whether the default system font for menu items is used.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	VertMargin (see page 490)	Specifies the width of the vertical margins in pixels.
	VertSeparator (see page 490)	Indicates whether a vertical separator bar should appear between item lines.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

151.1.8.2 TStandardMenuPopup Methods

The methods of the TStandardMenuPopup class are listed here.

Protected Methods

	Name	Description
V	CanAutoSize (see page 20492)	Indicates how the control can be resized.
V	GetExpandBtnClass (see page 20493)	This is GetExpandBtnClass, a member of class TStandardMenuPopup.
V	NCPaint (see page 20493)	This is NCPaint, a member of class TStandardMenuPopup.
V	PositionPopup (see page 20493)	This is PositionPopup, a member of class TStandardMenuPopup.

151.1.8.2.1 TStandardMenuPopup.CanAutoSize Method

Indicates how the control can be resized.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize property is true and an attempt is made to resize the control. It allows controls to implement the AutoSize property before the resize sequence that begins with the OnCanResize and OnConstrainedResize events and ends with the OnResize event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the control. CanAutoSize adjusts these values so that the control's size adjusts to its contents. If the adjustment results in values of NewWidth and NewHeight that differ from the control's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width and Height, CanAutoSize returns false, and the resize sequence is aborted.

As implemented in TWinControl, CanAutoSize returns true, adjusting NewWidth and NewHeight so that all the control's children fit.

See Also

Constraints (see page 5812)

ConstrainedResize (see page 6133)

AdjustSize (see page 6129)

151.1.8.2.2 TStandardMenuPopup.GetExpandBtnClass Method

Pascal

```
function GetExpandBtnClass: TCustomMenuExpandBtnClass; override;
```

C++

```
virtual __fastcall TCustomMenuExpandBtnClass GetExpandBtnClass();
```

Description

This is GetExpandBtnClass, a member of class TStandardMenuPopup.

151.1.8.2.3 TStandardMenuPopup.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TStandardMenuPopup.

151.1.8.2.4 TStandardMenuPopup.PositionPopup Method

Pascal

```
procedure PositionPopup(AnOwner: TCustomActionBar; ParentItem: TCustomActionControl); override;
```

C++

```
virtual __fastcall PositionPopup(TCustomActionBar AnOwner, TCustomActionControl ParentItem);
```

Description

This is PositionPopup, a member of class TStandardMenuPopup.

151.1.9 TStandardToolScrollBtn Class

Class Hierarchy

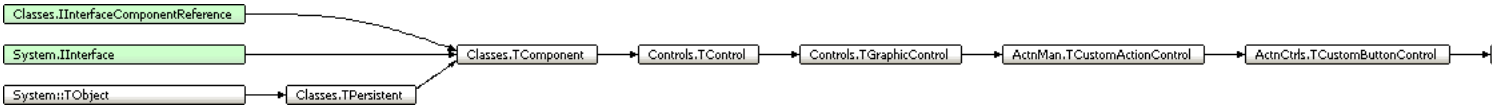

Pascal

```
TStandardToolScrollBtn = class(TCustomToolScrollBtn);
```

C++

```
class TStandardToolScrollBtn : public TCustomToolScrollBtn;
```

File

StdActnMenus

Description

This is class StdActnMenus.TStandardToolScrollBtn.

151.1.9.1 TStandardToolScrollBtn Members

The following tables list the members exposed by TStandardToolScrollBtn.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFlat (see page 178)	This is FFlat, a member of class TCustomButtonControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FState (see page 178)	This is FState, a member of class TCustomButtonControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	BeginAutoDrag (see page 517)	Begin Automatic Drag.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CalcLayout (see page 518)	This is CalcLayout, a member of class TCustomActionControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
	CMDesignHitTest (see page 519)	This is CMDesignHitTest, a member of class TCustomActionControl.
	CMDialogChar (see page 180)	This is CMDialogChar, a member of class TCustomButtonControl.
	CMEnabledChanged (see page 180)	This is CMEnabledChanged, a member of class TCustomButtonControl.
	CMFontChanged (see page 519)	This is CMFontChanged, a member of class TCustomActionControl.
	CMItemDropPoint (see page 519)	This is CMItemDropPoint, a member of class TCustomActionControl.
	CMItemSelected (see page 519)	This is CMItemSelected, a member of class TCustomActionControl.
	CMMouseEnter (see page 180)	This is CMMouseEnter, a member of class TCustomButtonControl.
	CMMouseEnter (see page 238)	This is CMMouseEnter, a member of class TCustomUtilityButton.
	CMMouseLeave (see page 180)	This is CMMouseLeave, a member of class TCustomButtonControl.
	CMMouseLeave (see page 239)	This is CMMouseLeave, a member of class TCustomUtilityButton.
	CMTextChanged (see page 180)	This is CMTextChanged, a member of class TCustomButtonControl.

	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 519)	This is DesignWndProc, a member of class TCustomActionControl.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDragDrop (see page 520)	This is DoDragDrop, a member of class TCustomActionControl.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 520)	OnStartDrag event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 521)	OnDragOver event dispatcher.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawArrows (see page 227)	This is DrawArrows, a member of class TCustomToolScrollBtn.
	DrawArrows (see page 239)	This is DrawArrows, a member of class TCustomUtilityButton.
	DrawBackground (see page 181)	Draws a background image on the control.
	DrawBackground (see page 521)	Draws a background image on the control.
	DrawDesignFocus (see page 521)	This is DrawDesignFocus, a member of class TCustomActionControl.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawDragDropPoint (see page 522)	This is DrawDragDropPoint, a member of class TCustomActionControl.
	DrawFrame (see page 20503)	This is DrawFrame, a member of class TStandardToolScrollBtn.
	DrawGlyph (see page 522)	Draws a symbol on the control.
	DrawLargeGlyph (see page 181)	This is DrawLargeGlyph, a member of class TCustomButtonControl.
	DrawLargeGlyph (see page 522)	This is DrawLargeGlyph, a member of class TCustomActionControl.
	DrawSeparator (see page 522)	This is DrawSeparator, a member of class TCustomActionControl.
	DrawShadowedText (see page 523)	This is DrawShadowedText, a member of class TCustomActionControl.
	DrawText (see page 523)	Draws text on the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	GetAction (see page 523)	This is GetAction, a member of class TCustomActionControl.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.

	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetImageSize (see page 523)	This is GetImageSize, a member of class TCustomActionControl.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSeparator (see page 524)	This is GetSeparator, a member of class TCustomActionControl.
	GetShowCaption (see page 524)	This is GetShowCaption, a member of class TCustomActionControl.
	GetShowShortCut (see page 181)	This is GetShowShortCut, a member of class TCustomButtonControl.
	GetShowShortCut (see page 524)	This is GetShowShortCut, a member of class TCustomActionControl.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HasGlyph (see page 524)	This is HasGlyph, a member of class TCustomActionControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsChecked (see page 525)	This is IsChecked, a member of class TCustomActionControl.
	IsGrouped (see page 525)	This is IsGrouped, a member of class TCustomActionControl.
	Loaded (see page 182)	Initializes the control after it is loaded from a stream.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 182)	OnMouseDown event dispatcher.
	MouseDown (see page 239)	OnMouseDown event dispatcher.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 183)	Respond to mouse moving over control area..
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 183)	OnMouseUp event dispatcher.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	OnDelay (see page 240)	This is OnDelay, a member of class TCustomUtilityButton.
	Paint (see page 184)	Render control surface.
	Paint (see page 240)	Render control surface.
	Paint (see page 526)	Render control surface.
	Paint (see page 6052)	Render control surface.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionChanged (see page 526)	This is PositionChanged, a member of class TCustomActionControl.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetUsageData (see page 526)	This is ResetUsageData, a member of class TCustomActionControl.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetActionClient (see page 184)	This is SetActionClient, a member of class TCustomButtonControl.
	SetActionClient (see page 526)	This is SetActionClient, a member of class TCustomActionControl.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.

V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropPoint (see page 527)	This is SetDropPoint, a member of class TCustomActionControl.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
V	SetGlyphLayout (see page 185)	This is SetGlyphLayout, a member of class TCustomButtonControl.
V	SetGlyphLayout (see page 527)	This is SetGlyphLayout, a member of class TCustomActionControl.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetSelected (see page 185)	This is SetSelected, a member of class TCustomButtonControl.
V	SetSelected (see page 527)	This is SetSelected, a member of class TCustomActionControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateSelection (see page 527)	This is UpdateSelection, a member of class TCustomActionControl.
	UpdateTextBounds (see page 527)	This is UpdateTextBounds, a member of class TCustomActionControl.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMLButtonDblClk (see page 185)	This is WMLButtonDblClk, a member of class TCustomButtonControl.
	WMSetText (see page 528)	This is WMSetText, a member of class TCustomActionControl.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.

	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	GlyphPos (see page 532)	This is GlyphPos, a member of class TCustomActionControl.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
R	MouseInControl (see page 189)	This is MouseInControl, a member of class TCustomButtonControl.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	OnClick (see page 243)	Occurs when the user clicks the control.
	OnClick (see page 535)	Occurs when the user clicks the control.
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
V	CalcBounds (see page 179)	Calculates the size of the control.
V	CalcBounds (see page 518)	Calculates the size of the control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Click (see page 179)	Executes the control's function.
V	Click (see page 238)	Executes the control's function.
V	Click (see page 518)	Executes the control's function in response to a mouse click.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 177)	Creates an instance of TCustomButtonControl (see page 168).
V	Create (see page 227)	Creates an instance of TCustomButtonControl.
V	Create (see page 237)	Creates an instance of TCustomButtonControl.
V	Create (see page 515)	Creates an instance of TCustomActionControl (see page 507).
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6051)	Creates an instance of TGraphicControl (see page 6044).

	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 237)	Destroys an instance of TCustomActionControl.
V	Destroy (see page 516)	Destroys an instance of TCustomActionControl (see page 507).
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
V	DragDrop (see page 520)	Performs actions and generates an OnDragDrop event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 524)	Calls the action client's Update method if the control is associated with an action client.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
V	Keyed (see page 525)	Executes the control's function in response to a keyboard action.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 184)	Sets the Left, Top, Width, and Height properties all at once.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	ActionBar (see page 528)	The ActionBand object which contains the custom action control
	ActionClient (see page 529)	Refers to the control's action client
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	AllowAllUp (see page 186)	Indicates whether all the buttons in a given group can be up at the same time.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	ArrowSize (see page 240)	This is ArrowSize, a member of class TCustomUtilityButton.
	ArrowType (see page 241)	This is ArrowType, a member of class TCustomUtilityButton.
	AutoScroll (see page 241)	This is AutoScroll, a member of class TCustomUtilityButton.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.

	Caption (see page 186)	Specifies a text string that identifies the control to the user.
	Caption (see page 529)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 241)	Specifies the background color of the control.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Direction (see page 242)	This is Direction, a member of class TCustomUtilityButton.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Down (see page 189)	Tells whether the button is down (selected) or up (unselected).
	DropPoint (see page 532)	This is DropPoint, a member of class TCustomActionControl.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
	Flat (see page 189)	Dictates whether the button should have a 2D look instead of the usual 3D look.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GlyphLayout (see page 532)	Describes where the image should be on the control
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Margins (see page 532)	Describes the boundaries of the control for drawing purposes.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	RepeatRate (see page 242)	This is RepeatRate, a member of class TCustomUtilityButton.
	Selected (see page 533)	Indicates whether the control is currently selected.
R	Separator (see page 533)	Indicates whether the control is a separator.
	ShowCaption (see page 189)	Indicates whether the control's caption should be shown.
R	ShowCaption (see page 533)	Indicates whether the control's caption should be shown.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
	ShowShortCut (see page 190)	Indicates whether the control's short cut should be shown.
R	ShowShortCut (see page 533)	Indicates whether the control's short cut should be shown.
	SmallIcon (see page 534)	Indicates whether the control's small icon should be shown.
	Spacing (see page 534)	Controls (see page 5711) the separation between the control's icon and caption.
	State (see page 190)	Describes the current state of the button.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.

	TextBounds (see page 534)	A bounding rectangle for the control caption.
	TrackButton (see page 190)	Specifies which mouse button activates the button control.
	Transparent (see page 534)	Shows whether the control is transparent or not.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

151.1.9.2 TStandardToolScrollBtn Methods

The methods of the TStandardToolScrollBtn class are listed here.

Protected Methods

	Name	Description
	DrawFrame (see page 20503)	This is DrawFrame, a member of class TStandardToolScrollBtn.

151.1.9.2.1 TStandardToolScrollBtn.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TStandardToolScrollBtn.

151.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	RegisterStandardMenus (see page 20504)	This is function StdActnMenus.RegisterStandardMenus.

151.2.1 StdActnMenus.RegisterStandardMenus Function

Pascal

```
procedure RegisterStandardMenus; deprecated;
```

C++

```
RegisterStandardMenus();
```

File

StdActnMenus

Description

This is function StdActnMenus.RegisterStandardMenus.

152 StdActns Namespace

This is namespace StdActns.

Topics

Name	Description
Classes (see page 20507)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 20822)	The following table lists structs, records, enums in this documentation.
Types (see page 20824)	The following table lists types in this documentation.

Classes

	Name	Description
	TBrowseForFolder (see page 20508)	This is class StdActns.TBrowseForFolder.
	TColorSelect (see page 20521)	TColorSelect is the standard action for displaying a color selection dialog.
	TCommonDialogAction (see page 20533)	TCommonDialogAction is the base class for action objects that display a dialog.
	TEditAction (see page 20544)	TEditAction is the base class for action objects that perform standard edits on an edit control.
	TEditCopy (see page 20552)	TEditCopy copies the selected text in a target edit control to the Windows clipboard.
	TEditCut (see page 20557)	TEditCut cuts the selected text in a target edit or memo control to the Windows clipboard.
	TEditDelete (see page 20563)	TEditDelete deletes the selected text in a target edit or memo control.
	TEditPaste (see page 20570)	TEditPaste pastes text from the clipboard into the active edit or memo control.
	TEditSelectAll (see page 20576)	TEditSelectAll selects all text in an active text control.
	TEditUndo (see page 20582)	TEditUndo backs out the last change in an edit control.
	TFileAction (see page 20588)	TFileAction is the base class for action objects that display a file selection dialog.
	TFileExit (see page 20594)	TFileExit is the standard action for shutting down an application.
	TFileOpen (see page 20605)	TFileOpen is the standard action for displaying a file open dialog.
	TFileOpenWith (see page 20619)	TFileOpenWith is the standard action for displaying the dialog that lets users choose what application to user for opening a file.
	TFilePageSetup (see page 20629)	TFilePageSetup is the standard action for displaying a page setup dialog.
	TFilePrintSetup (see page 20642)	TFilePrintSetup is the standard action for displaying a printer setup dialog.
	TFileSaveAs (see page 20654)	TFileSaveAs is the standard action for displaying a file save as dialog.
	TFontEdit (see page 20666)	TFontEdit is the standard action for displaying a font selection dialog.
	THelpAction (see page 20678)	THelpAction is the base class for action objects that interact with the help system.
	THelpContents (see page 20684)	THelpContents brings up the contents for the default help system.
	THelpContextAction (see page 20690)	THelpContextAction brings up the help topic for the active control.
	THelpOnHelp (see page 20696)	THelpOnHelp brings up a help topic on how to use Help.

	THelpTopicSearch (see page 20701)	THelpTopicSearch brings up the help index.
	THintAction (see page 20707)	THintAction is the action that executes when the user pauses the mouse over a control or menu item that has a hint.
	TPrintDlg (see page 20713)	TPrintDlg is the standard action for displaying a print dialog.
	TSearchAction (see page 20725)	TSearchAction is the base class for action objects that display a search or search and replace dialog.
	TSearchFind (see page 20734)	TSearchFind is the standard action for displaying a find dialog.
	TSearchFindFirst (see page 20747)	TSearchFindFirst is the standard action for displaying a find dialog that looks for the first occurrence of a string.
	TSearchFindNext (see page 20753)	TSearchFindNext is the standard action for locating the next instance of a string in an edit control.
	TSearchReplace (see page 20765)	TSearchReplace is the standard action for displaying a find-and-replace dialog.
	TWindowAction (see page 20779)	TWindowAction is the base class for action objects that work with MDI forms.
	TWindowArrange (see page 20786)	TWindowArrange arranges the icons of minimized MDI child forms.
	TWindowCascade (see page 20792)	TWindowCascade cascades the MDI child forms.
	TWindowClose (see page 20798)	TWindowClose closes the active MDI child form.
	TWindowMinimizeAll (see page 20804)	TWindowMinimizeAll minimizes all MDI child forms.
	TWindowTileHorizontal (see page 20809)	TWindowTileHorizontal arranges MDI child forms so that they are all the same size, tiled horizontally.
	TWindowTileVertical (see page 20815)	TWindowTileVertical arranges MDI child forms so that they are all the same size, tiled vertically.

Structs, Records, Enums

	Name	Description
	TBrowseForFolderOption (see page 20822)	This is record StdActns.TBrowseForFolderOption.

Types

Name	Description
TBrowseForFolderCallbackEvent (see page 20824)	This is type StdActns.TBrowseForFolderCallbackEvent.
TBrowseOptions (see page 20824)	This is type StdActns.TBrowseOptions.
TCommonDialogClass (see page 20825)	This is type StdActns.TCommonDialogClass.

152.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TBrowseForFolder (see page 20508)	This is class StdActns.TBrowseForFolder.
	TColorSelect (see page 20521)	TColorSelect is the standard action for displaying a color selection dialog.
	TCommonDialogAction (see page 20533)	TCommonDialogAction is the base class for action objects that display a dialog.
	TEditAction (see page 20544)	TEditAction is the base class for action objects that perform standard edits on an edit control.
	TEditCopy (see page 20552)	TEditCopy copies the selected text in a target edit control to the Windows clipboard.
	TEditCut (see page 20557)	TEditCut cuts the selected text in a target edit or memo control to the Windows clipboard.
	TEditDelete (see page 20563)	TEditDelete deletes the selected text in a target edit or memo control.
	TEditPaste (see page 20570)	TEditPaste pastes text from the clipboard into the active edit or memo control.
	TEditSelectAll (see page 20576)	TEditSelectAll selects all text in an active text control.
	TEditUndo (see page 20582)	TEditUndo backs out the last change in an edit control.
	TFileAction (see page 20588)	TFileAction is the base class for action objects that display a file selection dialog.
	TFileExit (see page 20594)	TFileExit is the standard action for shutting down an application.
	TFileOpen (see page 20605)	TFileOpen is the standard action for displaying a file open dialog.
	TFileOpenWith (see page 20619)	TFileOpenWith is the standard action for displaying the dialog that lets users choose what application to user for opening a file.
	TFilePageSetup (see page 20629)	TFilePageSetup is the standard action for displaying a page setup dialog.
	TFilePrintSetup (see page 20642)	TFilePrintSetup is the standard action for displaying a printer setup dialog.
	TFileSaveAs (see page 20654)	TFileSaveAs is the standard action for displaying a file save as dialog.
	TFontEdit (see page 20666)	TFontEdit is the standard action for displaying a font selection dialog.
	THelpAction (see page 20678)	THelpAction is the base class for action objects that interact with the help system.
	THelpContents (see page 20684)	THelpContents brings up the contents for the default help system.
	THelpContextAction (see page 20690)	THelpContextAction brings up the help topic for the active control.
	THelpOnHelp (see page 20696)	THelpOnHelp brings up a help topic on how to use Help.
	THelpTopicSearch (see page 20701)	THelpTopicSearch brings up the help index.
	THintAction (see page 20707)	THintAction is the action that executes when the user pauses the mouse over a control or menu item that has a hint.
	TPrintDlg (see page 20713)	TPrintDlg is the standard action for displaying a print dialog.
	TSearchAction (see page 20725)	TSearchAction is the base class for action objects that display a search or search and replace dialog.
	TSearchFind (see page 20734)	TSearchFind is the standard action for displaying a find dialog.
	TSearchFindFirst (see page 20747)	TSearchFindFirst is the standard action for displaying a find dialog that looks for the first occurrence of a string.
	TSearchFindNext (see page 20753)	TSearchFindNext is the standard action for locating the next instance of a string in an edit control.
	TSearchReplace (see page 20765)	TSearchReplace is the standard action for displaying a find-and-replace dialog.
	TWindowAction (see page 20779)	TWindowAction is the base class for action objects that work with MDI forms.
	TWindowArrange (see page 20786)	TWindowArrange arranges the icons of minimized MDI child forms.
	TWindowCascade (see page 20792)	TWindowCascade cascades the MDI child forms.
	TWindowClose (see page 20798)	TWindowClose closes the active MDI child form.
	TWindowMinimizeAll (see page 20804)	TWindowMinimizeAll minimizes all MDI child forms.
	TWindowTileHorizontal (see page 20809)	TWindowTileHorizontal arranges MDI child forms so that they are all the same size, tiled horizontally.
	TWindowTileVertical (see page 20815)	TWindowTileVertical arranges MDI child forms so that they are all the same size, tiled vertically.

152.1.1 TBrowseForFolder Class

Class Hierarchy

Pascal

```
TBrowseForFolder = class(TCommonDialogAction);
```

C++

```
class TBrowseForFolder : public TCommonDialogAction;
```

File

StdActns

Description

This is class StdActns.TBrowseForFolder.

152.1.1.1 TBrowseForFolder Members

The following tables list the members exposed by TBrowseForFolder.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDialog (see page 20539)	This is FDialog, a member of class TCommonDialogAction.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAccept (see page 20540)	This is DoAccept, a member of class TCommonDialogAction.
	DoCallback (see page 20513)	This is DoCallback, a member of class TBrowseForFolder.

	DoCancel (see page 20540)	This is DoCancel, a member of class TCommonDialogAction.
	DoValidate (see page 20513)	This is DoValidate, a member of class TBrowseForFolder.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDialogClass (see page 20514)	This is GetDialogClass, a member of class TBrowseForFolder.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 20541)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetupDialog (see page 20542)	This is SetupDialog, a member of class TCommonDialogAction.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 20538)	Creates an instance of TCommonDialogAction.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 20538)	Frees an instance of TCustomAction.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 20513)	Executes the associated dialog and generates an OnAccept or OnCancel event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 20541)	Indicates whether the action can execute its dialog.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
R	ExecuteResult (see page 20542)	Indicates the result of executing the associated dialog.

	Folder (see page 20515)	This is Folder, a member of class TBrowseForFolder.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	BeforeExecute (see page 20519)	Occurs immediately before the dialog appears.
	OnAccept (see page 20520)	This is OnAccept, a member of class TBrowseForFolder.
	OnCallback (see page 20520)	This is OnCallback, a member of class TBrowseForFolder.
	OnCancel (see page 20520)	This is OnCancel, a member of class TBrowseForFolder.
	OnHint (see page 20520)	Occurs when the mouse pauses over a client control or menu item.
	OnValidate (see page 20521)	This is OnValidate, a member of class TBrowseForFolder.

Published Properties

	Name	Description
	BrowseOptions (see page 20514)	This is BrowseOptions, a member of class TBrowseForFolder.
	Caption (see page 20514)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	DialogCaption (see page 20515)	This is DialogCaption, a member of class TBrowseForFolder.
	Enabled (see page 20515)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20516)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20516)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20517)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20517)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20517)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RootDir (see page 20518)	This is RootDir, a member of class TBrowseForFolder.
	SecondaryShortCuts (see page 20518)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20518)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 20519)	Specifies the Visible property for client controls and menu items.

152.1.1.2 TBrowseForFolder Methods

The methods of the TBrowseForFolder class are listed here.

Protected Methods

	Name	Description
	DoCallback (see page 20513)	This is DoCallback, a member of class TBrowseForFolder.
	DoValidate (see page 20513)	This is DoValidate, a member of class TBrowseForFolder.
V	GetDialogClass (see page 20514)	This is GetDialogClass, a member of class TBrowseForFolder.

Public Methods

	Name	Description
V	ExecuteTarget (see page 20513)	Executes the associated dialog and generates an OnAccept or OnCancel event.

152.1.1.2.1 TBrowseForFolder.DoCallback Method

Pascal

```
function DoCallback(Wnd: HWND; uMsg: UINT; lParam: LPARAM; lpData: LPARAM): Integer;
```

C++

```
__fastcall int DoCallback(HWND Wnd, UINT uMsg, LPARAM lParam, LPARAM lpData);
```

Description

This is DoCallback, a member of class TBrowseForFolder.

152.1.1.2.2 TBrowseForFolder.DoValidate Method

Pascal

```
function DoValidate(Wnd: HWND; uMsg: UINT; lParam: LPARAM; lpData: LPARAM): Integer;
```

C++

```
__fastcall int DoValidate(HWND Wnd, UINT uMsg, LPARAM lParam, LPARAM lpData);
```

Description

This is DoValidate, a member of class TBrowseForFolder.

152.1.1.2.3 TBrowseForFolder.ExecuteTarget Method

Executes the associated dialog and generates an OnAccept or OnCancel event.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and

1. Generates a BeforeExecute event, so that the application can initialize the associated dialog according to current conditions.
2. Calls the Execute method of the dialog.
3. Generates an OnAccept event or an OnCancel event, depending on the value returned by the Execute method. You can write code to make use of the results from executing the dialog in these event handlers.

See Also

onaccept (see page 20543)

oncancel (see page 20543)

Execute (see page 11283)

BeforeExecute (see page 20543)

152.1.1.2.4 TBrowseForFolder.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TBrowseForFolder.

152.1.1.3 TBrowseForFolder Properties

The properties of the TBrowseForFolder class are listed here.

Public Properties

	Name	Description
	Folder (see page 20515)	This is Folder, a member of class TBrowseForFolder.

Published Properties

	Name	Description
	BrowseOptions (see page 20514)	This is BrowseOptions, a member of class TBrowseForFolder.
	Caption (see page 20514)	Represents the caption of client controls and menu items.
	DialogCaption (see page 20515)	This is DialogCaption, a member of class TBrowseForFolder.
	Enabled (see page 20515)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20516)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20516)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20517)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20517)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20517)	Indicates the ImageIndex property for client controls and menu items.
	RootDir (see page 20518)	This is RootDir, a member of class TBrowseForFolder.
	SecondaryShortCuts (see page 20518)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20518)	Specifies the ShortCut property for client menu items.
	Visible (see page 20519)	Specifies the Visible property for client controls and menu items.

152.1.1.3.1 TBrowseForFolder.BrowseOptions Property

Pascal

```
property BrowseOptions: TBrowseOptions;
```

C++

```
__property TBrowseOptions BrowseOptions;
```

Description

This is BrowseOptions, a member of class TBrowseForFolder.

152.1.1.3.2 TBrowseForFolder.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

152.1.1.3.3 TBrowseForFolder.DialogCaption Property

Pascal

```
property DialogCaption: string;
```

C++

```
__property AnsiString DialogCaption;
```

Description

This is DialogCaption, a member of class TBrowseForFolder.

152.1.1.3.4 TBrowseForFolder.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

152.1.1.3.5 TBrowseForFolder.Folder Property

Pascal

```
property Folder: string;
```

C++

```
__property AnsiString Folder;
```

Description

This is Folder, a member of class TBrowseForFolder.

152.1.1.3.6 TBrowseForFolder.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

152.1.1.3.7 TBrowseForFolder.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

152.1.1.3.8 TBrowseForFolder.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

152.1.1.3.9 TBrowseForFolder.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

152.1.1.3.10 TBrowseForFolder.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

152.1.1.3.11 TBrowseForFolder.RootDir Property

Pascal

```
property RootDir: string;
```

C++

```
__property AnsiString RootDir;
```

Description

This is RootDir, a member of class TBrowseForFolder.

152.1.1.3.12 TBrowseForFolder.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

152.1.1.3.13 TBrowseForFolder.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

152.1.1.3.14 TBrowseForFolder.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

152.1.1.4 TBrowseForFolder Events

The events of the TBrowseForFolder class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 20519)	Occurs immediately before the dialog appears.
	OnAccept (see page 20520)	This is OnAccept, a member of class TBrowseForFolder.
	OnCallback (see page 20520)	This is OnCallback, a member of class TBrowseForFolder.
	OnCancel (see page 20520)	This is OnCancel, a member of class TBrowseForFolder.
	OnHint (see page 20520)	Occurs when the mouse pauses over a client control or menu item.
	OnValidate (see page 20521)	This is OnValidate, a member of class TBrowseForFolder.

152.1.1.4.1 TBrowseForFolder.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 20540)

Execute (see page 11283)

PropertyName

152.1.1.4.2 TBrowseForFolder.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TBrowseForFolder.

152.1.1.4.3 TBrowseForFolder.OnCallback Event

Pascal

```
property OnCallback: TBrowseForFolderCallbackEvent;
```

C++

```
__property TBrowseForFolderCallbackEvent OnCallback;
```

Description

This is OnCallback, a member of class TBrowseForFolder.

152.1.1.4.4 TBrowseForFolder.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TBrowseForFolder.

152.1.1.4.5 TBrowseForFolder.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The

OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

152.1.1.4.6 TBrowseForFolder.OnValidate Event

Pascal

```
property OnValidate: TBrowseForFolderCallbackEvent;
```

C++

```
__property TBrowseForFolderCallbackEvent OnValidate;
```

Description

This is OnValidate, a member of class TBrowseForFolder.

152.1.2 TColorSelect Class

TColorSelect is the standard action for displaying a color selection dialog.

Class Hierarchy

Pascal

```
TColorSelect = class(TCommonDialogAction);
```

C++

```
class TColorSelect : public TCommonDialogAction;
```

File

StdActns

Description

Add TColorSelect to an action list to add a modal color selection dialog to your application. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to display the color selection dialog when invoked. Use the OnAccept (see page 20531) event to respond after the user selects a color using the dialog or the OnCancel (see page 20532) event to respond if the user cancels from the dialog. In the event handler for these events, you can read the color from the dialog specified by the Dialog (see page 20527) property.

See Also

TColorDialog (see page 11270)

152.1.2.1 TColorSelect Members

The following tables list the members exposed by TColorSelect.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDialog (see page 20539)	This is FDialog, a member of class TCommonDialogAction.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAccept (see page 20540)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 20540)	This is DoCancel, a member of class TCommonDialogAction.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDialogClass (see page 20526)	This is GetDialogClass, a member of class TColorSelect.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20541)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.

	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetupDialog (see page 20542)	This is SetupDialog, a member of class TCommonDialogAction.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 20538)	Creates an instance of TCommonDialogAction.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20538)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
V	ExecuteTarget (see page 20540)	Executes the associated dialog and generates an OnAccept (see page 20543) or OnCancel (see page 20543) event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 20541)	Indicates whether the action can execute its dialog.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.

V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
R	ExecuteResult (see page 20542)	Indicates the result of executing the associated dialog.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	BeforeExecute (see page 20531)	Occurs immediately before the dialog appears.
	OnAccept (see page 20531)	This is OnAccept, a member of class TColorSelect.
	OnCancel (see page 20532)	This is OnCancel, a member of class TColorSelect.
	OnHint (see page 20532)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20532)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 20526)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
R	Dialog (see page 20527)	Provides access the color selection dialog that the action executes.
	Enabled (see page 20527)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20527)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20528)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20528)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20529)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20529)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 20530)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20530)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 20530)	Specifies the Visible property for client controls and menu items.

152.1.2.2 TColorSelect Methods

The methods of the TColorSelect class are listed here.

Protected Methods

	Name	Description
V	GetDialogClass (see page 20526)	This is GetDialogClass, a member of class TColorSelect.

152.1.2.2.1 TColorSelect.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TColorSelect.

152.1.2.3 TColorSelect Properties

The properties of the TColorSelect class are listed here.

Published Properties

	Name	Description
	Caption (see page 20526)	Represents the caption of client controls and menu items.
R	Dialog (see page 20527)	Provides access the color selection dialog that the action executes.
	Enabled (see page 20527)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20527)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20528)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20528)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20529)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20529)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 20530)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20530)	Specifies the ShortCut property for client menu items.
	Visible (see page 20530)	Specifies the Visible property for client controls and menu items.

152.1.2.3.1 TColorSelect.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

152.1.2.3.2 TColorSelect.Dialog Property

Provides access the color selection dialog that the action executes.

Pascal

```
property Dialog: TColorDialog;
```

C++

```
__property TColorDialog Dialog;
```

Description

Use Dialog to configure the color selection dialog before the action fires or to read the selected color after the action executes the dialog. For example, you can set the initially selected color in the dialog at design time using the Object Inspector or read the dialog's Color property in an OnAccept (see page 20531) event handler to make use of the selected color.

When using a color select action, you do not place a TColorDialog component on the form or data module at design time. Instead, you add the TColorSelect (see page 20521) to your action list and then use the Dialog property. The color selection dialog component is created by the TColorSelect (see page 20521) constructor.

See Also

onaccept (see page 20543)

oncancel (see page 20543)

Color (see page 11275)

152.1.2.3.3 TColorSelect.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

152.1.2.3.4 TColorSelect.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

152.1.2.3.5 TColorSelect.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

152.1.2.3.6 TColorSelect.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

152.1.2.3.7 TColorSelect.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

152.1.2.3.8 TColorSelect.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

152.1.2.3.9 TColorSelect.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

152.1.2.3.10 TColorSelect.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

152.1.2.3.11 TColorSelect.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

152.1.2.4 TColorSelect Events

The events of the TColorSelect class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 20531)	Occurs immediately before the dialog appears.
	OnAccept (see page 20531)	This is OnAccept, a member of class TColorSelect.
	OnCancel (see page 20532)	This is OnCancel, a member of class TColorSelect.
	OnHint (see page 20532)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20532)	Occurs when the application is idle or when the action list updates.

152.1.2.4.1 TColorSelect.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 20540)

Execute (see page 11283)

PropertyName

152.1.2.4.2 TColorSelect.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TColorSelect.

152.1.2.4.3 TColorSelect.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TColorSelect.

152.1.2.4.4 TColorSelect.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

152.1.2.4.5 TColorSelect.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
```

```
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

152.1.3 TCommonDialogAction Class

TCommonDialogAction is the base class for action objects that display a dialog.

Class Hierarchy

Pascal

```
TCommonDialogAction = class(TCustomAction);
```

C++

```
class TCommonDialogAction : public TCustomAction;
```

File

StdActns

Description

TCommonDialogAction implements the common behavior for actions that display a dialog when executed. Each descendant class represents a specific dialog. Do not use TCommonDialogAction directly in an application. Instead, use the specific TCommonDialogAction descendant that represents the type of dialog you want to execute. The various TCommonDialogAction descendants are listed in the See also list.

TCommonDialogAction objects do not require a specific type of control to be active: they handle any target. When executed, the dialog action displays its dialog, and then generates an OnAccept (see page 20543) or OnCancel (see page 20543) event, which you can use to process the results from the dialog.

When using TCommonDialogAction desendants, there is no need to place the dialog component in your form or data module. The action object creates its own dialog component.

See Also

TColorSelect (see page 20521)

TFileOpen (see page 20605)

TFilePrintSetup (see page 20642)

TFileSaveAs (see page 20654)

TSearchFind (see page 20734)

TSearchReplace (see page 20765)

TFontEdit (see page 20666)

TOpenPicture (see page 11965)

TSavePicture (see page 12048)

152.1.3.1 TCommonDialogAction Members

The following tables list the members exposed by TCommonDialogAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDialog (see page 20539)	This is FDialog, a member of class TCommonDialogAction.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.

	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoAccept (see page 20540)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 20540)	This is DoCancel, a member of class TCommonDialogAction.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDialogClass (see page 20541)	This is GetDialogClass, a member of class TCommonDialogAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 20541)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetupDialog (see page 20542)	This is SetupDialog, a member of class TCommonDialogAction.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	BeforeExecute (see page 20543)	Occurs immediately before the dialog appears.
	OnAccept (see page 20543)	This is OnAccept, a member of class TCommonDialogAction.
	OnCancel (see page 20543)	This is OnCancel, a member of class TCommonDialogAction.
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 20538)	Creates an instance of TCommonDialogAction.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 20538)	Frees an instance of TCustomAction.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 20540)	Executes the associated dialog and generates an OnAccept (see page 20543) or OnCancel (see page 20543) event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 20541)	Indicates whether the action can execute its dialog.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.

R	ExecuteResult (see page 20542)	Indicates the result of executing the associated dialog.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Properties

	Name	Description
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

152.1.3.2 TCommonDialogAction.Create Constructor

Creates an instance of TCommonDialogAction.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCommonDialogAction(TComponent * AOwner);
```

Description

Most applications do not need to call the TCommonDialogAction constructor because actions are added at design time to an action list or action manager.

The TCommonDialogAction constructor creates an instance of its associated dialog, which descendant classes make available through the Dialog property.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TCommonDialogAction instance.

See Also

Owner (see page 2807)

Actions (see page 325)

152.1.3.3 TCommonDialogAction.Destroy Destructor

Frees an instance of TCustomAction.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCommonDialogAction();
```

Description

Do not call the destructor directly in an application. Instead, call Free. Free ensures that the action is not nil before calling the destructor.

If the action is contained in an action list or action manager, Destroy removes the action from that list.

See Also

Create (see page 301)

Actions (see page 325)

152.1.3.4 TCommonDialogAction Fields

The fields of the TCommonDialogAction class are listed here.

Protected Fields

	Name	Description
	FDialog (see page 20539)	This is FDialog, a member of class TCommonDialogAction.

152.1.3.4.1 TCommonDialogAction.FDialog Field

Pascal

```
FDialog: TCommonDialog;
```

C++

```
TCommonDialog FDialog;
```

Description

This is FDialog, a member of class TCommonDialogAction.

152.1.3.5 TCommonDialogAction Methods

The methods of the TCommonDialogAction class are listed here.

Protected Methods

	Name	Description
	DoAccept (see page 20540)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 20540)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialogClass (see page 20541)	This is GetDialogClass, a member of class TCommonDialogAction.
	Notification (see page 20541)	Responds when components are created or destroyed.
	SetupDialog (see page 20542)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
	ExecuteTarget (see page 20540)	Executes the associated dialog and generates an OnAccept (see page 20543) or OnCancel (see page 20543) event.

	HandlesTarget (see page 20541)	Indicates whether the action can execute its dialog.

152.1.3.5.1 TCommonDialogAction.DoAccept Method

Pascal

```
procedure DoAccept; virtual;
```

C++

```
virtual __fastcall DoAccept();
```

Description

This is DoAccept, a member of class TCommonDialogAction.

152.1.3.5.2 TCommonDialogAction.DoCancel Method

Pascal

```
procedure DoCancel; virtual;
```

C++

```
virtual __fastcall DoCancel();
```

Description

This is DoCancel, a member of class TCommonDialogAction.

152.1.3.5.3 TCommonDialogAction.ExecuteTarget Method

Executes the associated dialog and generates an OnAccept (see page 20543) or OnCancel (see page 20543) event.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and

1. Generates a BeforeExecute (see page 20543) event, so that the application can initialize the associated dialog according to current conditions.
2. Calls the Execute method of the dialog.
3. Generates an OnAccept (see page 20543) event or an OnCancel (see page 20543) event, depending on the value returned by the Execute method. You can write code to make use of the results from executing the dialog in these event handlers.

See Also

onaccept (see page 20543)

oncancel (see page 20543)

Execute (see page 11283)

BeforeExecute (see page 20543)

152.1.3.5.4 TCommonDialogAction.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; virtual;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TCommonDialogAction.

152.1.3.5.5 TCommonDialogAction.HandlesTarget Method

Indicates whether the action can execute its dialog.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method lets the action object indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the currently active object.

HandlesTarget always returns true, because the execution of the dialog does not depend on any target object.

See Also

onaccept (see page 20543)

oncancel (see page 20543)

Execute (see page 11283)

152.1.3.5.6 TCommonDialogAction.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TBasicAction overrides the Notification method to check whether the component that is the value of the ActionComponent

property is about to be freed. If so, it sets the ActionComponent property to nil (Delphi) or NULL (C++).

See Also

ActionComponent (see page 2715)

152.1.3.5.7 TCommonDialogAction.SetupDialog Method

Pascal

```
procedure SetupDialog; virtual;
```

C++

```
virtual __fastcall SetupDialog();
```

Description

This is SetupDialog, a member of class TCommonDialogAction.

152.1.3.6 TCommonDialogAction Properties

The properties of the TCommonDialogAction class are listed here.

Public Properties

	Name	Description
	ExecuteResult (see page 20542)	Indicates the result of executing the associated dialog.

152.1.3.6.1 TCommonDialogAction.ExecuteResult Property

Indicates the result of executing the associated dialog.

Pascal

```
property ExecuteResult: Boolean;
```

C++

```
__property Boolean ExecuteResult;
```

Description

When the dialog action fires, it executes the dialog specified by the CommonDialog property and sets ExecuteResult the value returned by that dialog's Execute method. Thus, ExecuteResult is false if the dialog has not executed or the user canceled when the dialog executed, and true if the user exited the dialog by clicking OK.

See Also

onaccept (see page 20543)

oncancel (see page 20543)

Execute (see page 11283)

152.1.3.7 TCommonDialogAction Events

The events of the TCommonDialogAction class are listed here.

Public Events

	Name	Description
	BeforeExecute (see page 20543)	Occurs immediately before the dialog appears.
	OnAccept (see page 20543)	This is OnAccept, a member of class TCommonDialogAction.
	OnCancel (see page 20543)	This is OnCancel, a member of class TCommonDialogAction.

152.1.3.7.1 TCommonDialogAction.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 20540)

Execute (see page 11283)

PropertyName

152.1.3.7.2 TCommonDialogAction.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TCommonDialogAction.

152.1.3.7.3 TCommonDialogAction.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TCommonDialogAction.

152.1.4 TEditAction Class

TEditAction is the base class for action objects that perform standard edits on an edit control.

Class Hierarchy

Pascal

```
TEditAction = class(TAction);
```

C++

```
class TEditAction : public TAction;
```

File

StdActns

Description

TEditAction is the common base class for actions that work with edit controls. Each descendant class performs a specific type of edit on the target edit control. Do not use TEditAction directly in an application. Instead, use a TEditAction descendant that performs a specific function (for example, Cut, Copy, Paste, Delete, Select All, or Undo). The various TEditAction descendants are listed in the See also list.

See Also

TEditCopy (see page 20552)

TEditCut (see page 20557)

TEditDelete (see page 20563)

TEditPaste (see page 20570)

TEditSelectAll (see page 20576)

TEditUndo (see page 20582)

TEditRedo

TCustomEdit (see page 21232)

TRichEditBold (see page 12017)

TRichEditItalic (see page 12029)

TRichEditUnderline (see page 12042)

TRichEditAlignCenter (see page 11998)

TRIchEditAlignLeft

TRichEditAlignRight (see page 12011)

TRichEditBullets (see page 12023)

TRichEditStrikeOut (see page 12036)

152.1.4.1 TEditAction Members

The following tables list the members exposed by TEditAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetControl (see page 20549)	Returns the target of the edit action.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 20550)	Responds when objects are added or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 20549)	Frees an instance of TCustomAction.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 20550)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
	Update (see page 295)	Indicates whether the action updates itself.
	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateTarget (see page 20551)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Control (see page 20551)	Provides access to the target of the action.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.

	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.4.2 TEditAction.Destroy Destructor

Frees an instance of TCustomAction.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TEditAction();
```

Description

Do not call the destructor directly in an application. Instead, call Free. Free ensures that the action is not nil before calling the destructor.

If the action is contained in an action list or action manager, Destroy removes the action from that list.

See Also

Create (see page 301)

Actions (see page 325)

152.1.4.3 TEditAction Methods

The methods of the TEditAction class are listed here.

Protected Methods

	Name	Description
	GetControl (see page 20549)	Returns the target of the edit action.
	Notification (see page 20550)	Responds when objects are added or removed.

Public Methods

	Name	Description
	HandlesTarget (see page 20550)	Indicates whether the action can execute its function.
	UpdateTarget (see page 20551)	Enables or disables the action, depending on the currently active control in the form.

152.1.4.3.1 TEditAction.GetControl Method

Returns the target of the edit action.

Pascal

```
function GetControl(Target: TObject): TCustomEdit; virtual;
```

C++

```
virtual __fastcall TCustomEdit GetControl(TObject * Target);
```

Description

TEditAction (see page 20544) uses the protected GetControl method internally to obtain the target of the action.

Target is the currently active object, or the value of the Control (see page 20551) property.

GetControl returns the target object as an edit control.

See Also

ExecuteTarget (see page 2712)

Control (see page 20551)

152.1.4.3.2 TEditAction.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the currently active object.

HandlesTarget checks that the target of is a TCustomEdit descendant (or that the EditControl (see page 20551) property is set if Target is nil (Delphi) or NULL (C++)). HandlesTarget returns true if there is an edit control to act as target, false otherwise.

See Also

ExecuteTarget (see page 2712)

TCustomEdit (see page 21232)

EditControl

152.1.4.3.3 TEditAction.Notification Method

Responds when objects are added or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TEditAction (see page 20544) responds to notifications that the target edit control (as specified by the Control (see page 20551) property) is about to be freed by setting Control (see page 20551) to nil (Delphi) or NULL (C++).

See Also

ExecuteTarget (see page 2712)

Control (see page 20551)

152.1.4.3.4 TEditAction.UpdateTarget Method

Enables or disables the action, depending on the currently active control in the form.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the potential target changes so that actions can update themselves accordingly.

In TEditAction (see page 20544), UpdateTarget checks whether the current target object (specified by the Target parameter) is an edit control and, if the action's function requires it, that some of its text is selected. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

152.1.4.4 TEditAction Properties

The properties of the TEditAction class are listed here.

Public Properties

	Name	Description
	Control (see page 20551)	Provides access to the target of the action.

152.1.4.4.1 TEditAction.Control Property

Provides access to the target of the action.

Pascal

```
property Control: TCustomEdit;
```

C++

```
__property TCustomEdit Control;
```

Description

EditControl is the edit control on which the action performs its function. Set the value of Control to limit the action so that it only operates on a specific edit control. If Control is unassigned, the action dynamically locates the target edit control when it executes.

See Also

ExecuteTarget (see page 2712)

152.1.5 TEditCopy Class

TEditCopy copies the selected text in a target edit control to the Windows clipboard.

Class Hierarchy

Pascal

```
TEditCopy = class(TEditAction);
```

C++

```
class TEditCopy : public TEditAction;
```

File

StdActns

Description

Add TEditCopy to an action list to allow users to copy selected text to the clipboard using menu items or toolbar buttons. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to copy the currently selected text in an active text control to the clipboard.

TEditCopy is designed to work with an edit control (TCustomEdit descendant) as a target. It automatically disables itself if the active control in a form is not an edit control or if that edit control does not have any text selected.

See Also

TEditCut (see page 20557)

TEditDelete (see page 20563)

TEditPaste (see page 20570)

TEditSelectAll (see page 20576)

TEditUndo (see page 20582)

TEditRedo

TCustomEdit (see page 21232)

TRichEditBold (see page 12017)

TRichEditItalic (see page 12029)

TRichEditUnderline (see page 12042)

TSearchReplace (see page 20765)

152.1.5.1 TEditCopy Members

The following tables list the members exposed by TEditCopy.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetControl (see page 20549)	Returns the target of the edit action.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20550)	Responds when objects are added or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.

	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20549)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.

	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 20557)	Copies selected text in the target control to the Windows clipboard.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 20550)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.
V	UpdateTarget (see page 20551)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Control (see page 20551)	Provides access to the target of the action.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.

	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.5.2 TEditCopy Methods

The methods of the TEditCopy class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 20557)	Copies selected text in the target control to the Windows clipboard.

152.1.5.2.1 TEditCopy.ExecuteTarget Method

Copies selected text in the target control to the Windows clipboard.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is an edit control (TCustomEdit descendant), and if so, calls its CopyToClipboard method, copying the selected text to the clipboard.

See Also

TCustomEdit (see page 21232)

CopyToClipboard (see page 21246)

152.1.6 TEditCut Class

TEditCut cuts the selected text in a target edit or memo control to the Windows clipboard.

Class Hierarchy

Pascal

```
TEditCut = class(TEditAction);
```

C++

```
class TEditCut : public TEditAction;
```

File

StdActns

Description

Add TEditCut to an action list to allow users to cut selected text to the clipboard using menu items or toolbar buttons. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to cut the currently selected text in an active text control to the clipboard.

TEditCut is designed to work with an edit control (TCustomEdit or TCustomMemo descendant) as a target. It automatically disables itself if the active control in a form is not an edit control or if that edit control does not have a selection to cut.

See Also

TEditCopy (see page 20552)

TEditDelete (see page 20563)

TEditPaste (see page 20570)

TEditSelectAll (see page 20576)

TEditUndo (see page 20582)

TCustomEdit (see page 21232)

TRichEditBold (see page 12017)

TRichEditItalic (see page 12029)

TRichEditUnderline (see page 12042)

TSearchReplace (see page 20765)

152.1.6.1 TEditCut Members

The following tables list the members exposed by TEditCut.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetControl (see page 20549)	Returns the target of the edit action.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20550)	Responds when objects are added or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 20549)	Frees an instance of TCustomAction.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.

	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 20563)	Cuts selected text in the target control to the Windows clipboard.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 20550)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.
V	UpdateTarget (see page 20551)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.

	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Control (see page 20551)	Provides access to the target of the action.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.

	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.6.2 TEditCut Methods

The methods of the TEditCut class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 20563)	Cuts selected text in the target control to the Windows clipboard.

152.1.6.2.1 TEditCut.ExecuteTarget Method

Cuts selected text in the target control to the Windows clipboard.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is an edit control (TCustomEditControl descendant), and if so, calls its CutToClipboard method, cutting the selected text to the clipboard.

See Also

TCustomEdit (see page 21232)

CutToClipboard (see page 21249)

152.1.7 TEditDelete Class

TEditDelete deletes the selected text in a target edit or memo control.

Class Hierarchy

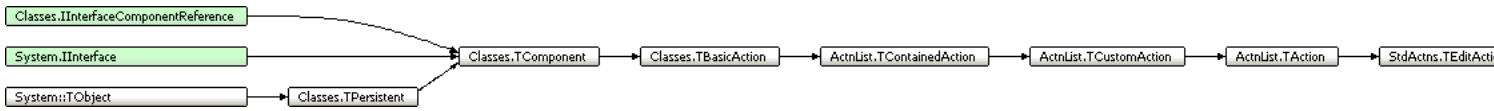

Pascal

```
TEditDelete = class(TEditAction);
```

C++

```
class TEditDelete : public TEditAction;
```

File

StdActns

Description

Add TEditDelete to an action list to let users delete selected text using menu items or toolbar buttons. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to delete the currently selected text in an active text control.

TEditDelete is designed to work with an edit control (TCustomEdit or TCustomMemo descendant) as a target. It automatically disables itself if the active control in a form is not an edit control or if the active edit control does not have any selected text.

See Also

TEditCopy (see page 20552)

TEditCut (see page 20557)

TEditPaste (see page 20570)

TEditSelectAll (see page 20576)

TEditUndo (see page 20582)

TCustomEdit (see page 21232)

TCustomMemo (see page 21362)

TRichEditBold (see page 12017)

TRichEditItalic (see page 12029)

TRichEditUnderline (see page 12042)

TSearchReplace (see page 20765)

152.1.7.1 TEditDelete Members

The following tables list the members exposed by TEditDelete.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.

	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetControl (see page 20549)	Returns the target of the edit action.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 20550)	Responds when objects are added or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 20549)	Frees an instance of TCustomAction.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 20569)	Deletes selected text in the target control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 20550)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 20569)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Control (see page 20551)	Provides access to the target of the action.

	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.7.2 TEditDelete Methods

The methods of the TEditDelete class are listed here.

Public Methods

	Name	Description
V	ExecuteTarget (see page 20569)	Deletes selected text in the target control.

	UpdateTarget (see page 20569)	Enables or disables the action, depending on the currently active control in the form.

152.1.7.2.1 TEditDelete.ExecuteTarget Method

Deletes selected text in the target control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is a TCustomEdit descendant, and if so, calls its ClearSelection method, deleting the selected text.

See Also

TCustomEdit (see page 21232)

EditControl

MemoControl

ClearSelection (see page 21245)

152.1.7.2.2 TEditDelete.UpdateTarget Method

Enables or disables the action, depending on the currently active control in the form.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TEditDelete (see page 20563), UpdateTarget checks whether the current target object (specified by the Target parameter) is an edit control that has some of its text selected. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

152.1.8 TEditPaste Class

TEditPaste pastes text from the clipboard into the active edit or memo control.

Class Hierarchy

Pascal

```
TEditPaste = class(TEditAction);
```

C++

```
class TEditPaste : public TEditAction;
```

File

StdActns

Description

Add TEditPaste to an action list to enable users to paste text from the clipboard using menu items or toolbar buttons. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to paste text from the clipboard into an active text control.

TEditPaste is designed to work with an edit control (TCustomEdit or TCustomMemo descendant) as a target. It automatically disables itself if the clipboard does not contain text.

See Also

TEditCopy (see page 20552)

TEditCut (see page 20557)

TEditDelete (see page 20563)

TEditSelectAll (see page 20576)

TEditUndo (see page 20582)

TCustomEdit (see page 21232)

TRichEditBold (see page 12017)

TRichEditItalic (see page 12029)

TRichEditUnderline (see page 12042)

TSearchReplace (see page 20765)

152.1.8.1 TEditPaste Members

The following tables list the members exposed by TEditPaste.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetControl (see page 20549)	Returns the target of the edit action.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20550)	Responds when objects are added or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.

	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20549)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.

	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 20575)	Pastes text from the clipboard into the target edit control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 20550)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 20575)	Enables or disables the action, depending on the contents of the clipboard.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Control (see page 20551)	Provides access to the target of the action.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.

	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.8.2 TEditPaste Methods

The methods of the TEditPaste class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 20575)	Pastes text from the clipboard into the target edit control.
	UpdateTarget (see page 20575)	Enables or disables the action, depending on the contents of the clipboard.

152.1.8.2.1 TEditPaste.ExecuteTarget Method

Pastes text from the clipboard into the target edit control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is a TCustomEdit descendant, and if so, calls its PasteFromClipboard method, pasting in any text on the clipboard.

See Also

TCustomEdit (see page 21232)

EditControl

MemoControl

PasteFromClipboard (see page 21254)

152.1.8.2.2 TEditPaste.UpdateTarget Method

Enables or disables the action, depending on the contents of the clipboard.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TEditPaste (see page 20570), UpdateTarget checks whether the clipboard contains text. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

152.1.9 TEditSelectAll Class

TEditSelectAll selects all text in an active text control.

Class Hierarchy

Pascal

```
TEditSelectAll = class(TEditAction);
```

C++

```
class TEditSelectAll : public TEditAction;
```

File

StdActns

Description

Add TEditSelectAll to an action list to allow users to select all the text in a text control using menu items or toolbar buttons. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to select all text in an active text control.

TEditSelectAll is designed to work with an edit control (TCustomEdit descendant) as a target. It automatically disables itself if the active control in a form is not an edit control or if the edit control does not have any text.

See Also

TEditCopy (see page 20552)

TEditCut (see page 20557)

TEditDelete (see page 20563)

TEditPaste (see page 20570)

TEditUndo (see page 20582)

TEditRedo

TCustomEdit (see page 21232)

TRichEditBold (see page 12017)

TRichEditItalic (see page 12029)

TRichEditUnderline (see page 12042)

TSearchReplace (see page 20765)

152.1.9.1 TEditSelectAll Members

The following tables list the members exposed by TEditSelectAll.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetControl (see page 20549)	Returns the target of the edit action.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20550)	Responds when objects are added or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.

	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.

	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20549)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 20581)	Selects all text in the target edit control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 20550)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 20582)	Enables or disables the action, depending on the contents of the target edit control.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Control (see page 20551)	Provides access to the target of the action.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.

	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.9.2 TEditSelectAll Methods

The methods of the TEditSelectAll class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 20581)	Selects all text in the target edit control.
	UpdateTarget (see page 20582)	Enables or disables the action, depending on the contents of the target edit control.

152.1.9.2.1 TEditSelectAll.ExecuteTarget Method

Selects all text in the target edit control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is a TCustomEdit descendant, and if so, calls its SelectAll method, selecting all of its text.

See Also

TCustomEdit (see page 21232)

EditControl

MemoControl

SelectAll

152.1.9.2.2 TEditSelectAll.UpdateTarget Method

Enables or disables the action, depending on the contents of the target edit control.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TEditSelectAll (see page 20576), UpdateTarget checks whether Target is a text control that contains text. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

152.1.10 TEditUndo Class

TEditUndo backs out the last change in an edit control.

Class Hierarchy

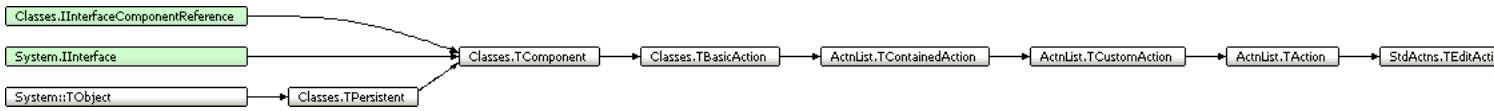

Pascal

```
TEditUndo = class(TEditAction);
```

C++

```
class TEditUndo : public TEditAction;
```

File

StdActns

Description

Add TEditUndo to an action list to allow users to back out edits using menu items or toolbar buttons. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to undo the last edit in an active text control.

TEditUndo is designed to work with an edit control (TCustomEdit descendant) as a target. It automatically disables itself if the active control in a form is not an edit control that can back out the last edit.

See Also

TEditCopy (see page 20552)

TEditCut (see page 20557)

TEditPaste (see page 20570)

TEditSelectAll (see page 20576)

TEditDelete (see page 20563)

TCustomEdit (see page 21232)

TRichEditBold (see page 12017)

TRichEditItalic (see page 12029)

TRichEditUnderline (see page 12042)

TSearchReplace (see page 20765)

152.1.10.1 TEditUndo Members

The following tables list the members exposed by TEditUndo.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetControl (see page 20549)	Returns the target of the edit action.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20550)	Responds when objects are added or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 20549)	Frees an instance of TCustomAction.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 20587)	Backs out the last edit from the target control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 20550)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 20588)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Control (see page 20551)	Provides access to the target of the action.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.

	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.10.2 TEditUndo Methods

The methods of the TEditUndo class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 20587)	Backs out the last edit from the target control.
	UpdateTarget (see page 20588)	Enables or disables the action, depending on the currently active control in the form.

152.1.10.2.1 TEditUndo.ExecuteTarget Method

Backs out the last edit from the target control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is a TCustomEdit descendant, and if so, calls its Undo method, backing out the last change.

See Also

TCustomEdit (see page 21232)

Undo (see page 21259)

152.1.10.2.2 TEditUndo.UpdateTarget Method

Enables or disables the action, depending on the currently active control in the form.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TEditUndo (see page 20582), UpdateTarget checks whether the current target object (specified by the Target parameter) is an edit control that is in a state where it can back out changes. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

CanUndo (see page 21262)

152.1.11 TFileAction Class

TFileAction is the base class for action objects that display a file selection dialog.

Class Hierarchy

Pascal

```
TFileAction = class(TCommonDialogAction);
```

C++

```
class TFileAction : public TCommonDialogAction;
```

File

StdActns

Description

TFileAction implements the FileName (see page 20593) property that is common to action objects that display a file selection object.

See Also

TFileOpen (see page 20605)

TFileSaveAs (see page 20654)

TOpenDialog (see page 11407)

152.1.11.1 TFileAction Members

The following tables list the members exposed by TFileAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDialog (see page 20539)	This is FDialog, a member of class TCommonDialogAction.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAccept (see page 20540)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 20540)	This is DoCancel, a member of class TCommonDialogAction.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDialog (see page 20593)	This is GetDialog, a member of class TFileAction.
V	GetDialogClass (see page 20541)	This is GetDialogClass, a member of class TCommonDialogAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.

	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 20541)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetupDialog (see page 20542)	This is SetupDialog, a member of class TCommonDialogAction.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	FileName (see page 20593)	Specifies the file name for the file selection dialog.
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	BeforeExecute (see page 20543)	Occurs immediately before the dialog appears.
	OnAccept (see page 20543)	This is OnAccept, a member of class TCommonDialogAction.
	OnCancel (see page 20543)	This is OnCancel, a member of class TCommonDialogAction.
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 20538)	Creates an instance of TCommonDialogAction.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20538)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	ExecuteTarget (see page 20540)	Executes the associated dialog and generates an OnAccept (see page 20543) or OnCancel (see page 20543) event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 20541)	Indicates whether the action can execute its dialog.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.

	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
R	ExecuteResult (see page 20542)	Indicates the result of executing the associated dialog.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.

R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Properties

	Name	Description
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

152.1.11.2 TFileAction Methods

The methods of the TFileAction class are listed here.

Protected Methods

	Name	Description
	GetDialog (see page 20593)	This is GetDialog, a member of class TFileAction.

152.1.11.2.1 TFileAction.GetDialog Method

Pascal

```
function GetDialog: TOpenDialog;
```

C++

```
__fastcall TOpenDialog GetDialog();
```

Description

This is GetDialog, a member of class TFileAction.

152.1.11.3 TFileAction Properties

The properties of the TFileAction class are listed here.

Protected Properties

	Name	Description
	FileName (see page 20593)	Specifies the file name for the file selection dialog.

152.1.11.3.1 TFileAction.FileName Property

Specifies the file name for the file selection dialog.

Pascal

```
property FileName: TFileName;
```

C++

```
__property TFileName FileName;
```

Description

Applications can't use the protected FileName property. To access the name of the file selected by the user, use the Dialog property of a TFileAction (see page 20588) descendant.

FileName is the same as the FileName property of the associated dialog.

See Also

FileName (see page 11424)

onaccept (see page 20543)

Dialog (see page 20612)

Dialog (see page 20660)

152.1.12 TFileExit Class

TFileExit is the standard action for shutting down an application.

Class Hierarchy

Pascal

```
TFileExit = class(TCustomAction);
```

C++

```
class TFileExit : public TCustomAction;
```

File

StdActns

Description

Add TFileExit to an action list to let users shut down the application using a menu or tool bar. Controls (see page 5711) linked to this action cause the application to shut down by closing the main form.

See Also

TCustomForm (see page 13849)

TAPplication

152.1.12.1 TFileExit Members

The following tables list the members exposed by TFileExit.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.

	ExecuteTarget (see page 20598)	Shuts down the main form of the application.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 20599)	Indicates whether the action can execute.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
	Update (see page 295)	Indicates whether the action updates itself.
	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.

R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnHint (see page 20604)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20604)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 20600)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Enabled (see page 20600)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20600)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20601)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20601)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20602)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20602)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 20602)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20603)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 20603)	Specifies the Visible property for client controls and menu items.

152.1.12.2 TFileExit Methods

The methods of the TFileExit class are listed here.

Public Methods

	Name	Description
V	ExecuteTarget (see page 20598)	Shuts down the main form of the application.
V	HandlesTarget (see page 20599)	Indicates whether the action can execute.

152.1.12.2.1 TFileExit.ExecuteTarget Method

Shuts down the main form of the application.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and calls the Close method of the application's main form. This causes the application to shut down.

See Also

MainForm (see page 13718)

Close (see page 13867)

152.1.12.2.2 TFileExit.HandlesTarget Method

Indicates whether the action can execute.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the currently active object.

HandlesTarget always returns true, because shutting down the application does not depend on any target object.

152.1.12.3 TFileExit Properties

The properties of the TFileExit class are listed here.

Published Properties

	Name	Description
	Caption (see page 20600)	Represents the caption of client controls and menu items.
	Enabled (see page 20600)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20600)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20601)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20601)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20602)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20602)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 20602)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20603)	Specifies the ShortCut property for client menu items.
	Visible (see page 20603)	Specifies the Visible property for client controls and menu items.

152.1.12.3.1 TFileExit.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

152.1.12.3.2 TFileExit.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

152.1.12.3.3 TFileExit.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

152.1.12.3.4 TFileExit.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

152.1.12.3.5 TFileExit.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

152.1.12.3.6 TFileExit.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

152.1.12.3.7 TFileExit.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

152.1.12.3.8 TFileExit.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key

combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

152.1.12.3.9 TFileExit.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

152.1.12.3.10 TFileExit.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

152.1.12.4 TFileExit Events

The events of the TFileExit class are listed here.

Published Events

	Name	Description
	OnHint (see page 20604)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20604)	Occurs when the application is idle or when the action list updates.

152.1.12.4.1 TFileExit.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

152.1.12.4.2 TFileExit.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
```

```
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

152.1.13 TFileOpen Class

TFileOpen is the standard action for displaying a file open dialog.

Class Hierarchy

Pascal

```
TFileOpen = class(TFileAction);
```

C++

```
class TFileOpen : public TFileAction;
```

File

StdActns

Description

Add TFileOpen to an action list to add a common file open dialog to your application. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to display a file open dialog when invoked. Use the OnAccept (see page 20617) event to respond after the user selects a file name in the file open dialog or the OnCancel (see page 20617) event to respond when the user cancels from the dialog. In the event handler for these events, you can read the file name from the dialog specified by the Dialog (see page 20612) property.

See Also

TFileOpenWith (see page 20619)

TFileSaveAs (see page 20654)

TOpenDialog (see page 11407)

TOpenPicture (see page 11965)

152.1.13.1 TFileOpen Members

The following tables list the members exposed by TFileOpen.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDialog (see page 20539)	This is FDialog, a member of class TCommonDialogAction.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAccept (see page 20540)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 20540)	This is DoCancel, a member of class TCommonDialogAction.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDialog (see page 20593)	This is GetDialog, a member of class TFileAction.
V	GetDialogClass (see page 20611)	This is GetDialogClass, a member of class TFileOpen.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.

	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 20541)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetupDialog (see page 20542)	This is SetupDialog, a member of class TCommonDialogAction.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	FileName (see page 20593)	Specifies the file name for the file selection dialog.
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.

	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 20610)	Creates an instance of TFileOpen (see page 20605).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 20538)	Frees an instance of TCustomAction.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 20610)	Executes the associated dialog and generates an OnAccept or OnCancel event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 20541)	Indicates whether the action can execute its dialog.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
R	ExecuteResult (see page 20542)	Indicates the result of executing the associated dialog.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	BeforeExecute (see page 20617)	Occurs immediately before the dialog appears.
	OnAccept (see page 20617)	This is OnAccept, a member of class TFileOpen.
	OnCancel (see page 20617)	This is OnCancel, a member of class TFileOpen.
	OnHint (see page 20617)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20618)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 20612)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
R	Dialog (see page 20612)	Provides access the open dialog that the action executes.
	Enabled (see page 20612)	Indicates whether client controls and menu items are enabled.

	HelpContext (see page 20613)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20613)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20614)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20614)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20615)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 20615)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20615)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	UseDefaultApp (see page 20616)	Use the default application to open the file.
	Visible (see page 20616)	Specifies the Visible property for client controls and menu items.

152.1.13.2 TFileOpen.Create Constructor

Creates an instance of TFileOpen (see page 20605).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TFileOpen(TComponent * AOwner);
```

Description

Most applications do not need to call the TFileOpen (see page 20605) constructor because actions are added at design time to an action manager or action list.

The TFileOpen (see page 20605) constructor creates an instance of its associated dialog and assigns that component as the value of the Dialog (see page 20612) property.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TFileOpen (see page 20605) instance.

See Also

Dialog (see page 20612)

Owner (see page 2807)

152.1.13.3 TFileOpen Methods

The methods of the TFileOpen class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 20611)	This is GetDialogClass, a member of class TFileOpen.

Public Methods

	Name	Description
	ExecuteTarget (see page 20610)	Executes the associated dialog and generates an OnAccept or OnCancel event.

152.1.13.3.1 TFileOpen.ExecuteTarget Method

Executes the associated dialog and generates an OnAccept or OnCancel event.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and

1. Generates a BeforeExecute event, so that the application can initialize the associated dialog according to current conditions.
2. Calls the Execute method of the dialog.
3. Generates an OnAccept event or an OnCancel event, depending on the value returned by the Execute method. You can write code to make use of the results from executing the dialog in these event handlers.

See Also

onaccept (see page 20543)

oncancel (see page 20543)

Execute (see page 11283)

BeforeExecute (see page 20543)

152.1.13.3.2 TFileOpen.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TFileOpen.

152.1.13.4 TFileOpen Properties

The properties of the TFileOpen class are listed here.

Published Properties

	Name	Description
	Caption (see page 20612)	Represents the caption of client controls and menu items.
R	Dialog (see page 20612)	Provides access the open dialog that the action executes.
	Enabled (see page 20612)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20613)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20613)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20614)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20614)	Indicates the Help hint for client controls and menu items.

	ImageIndex (see page 20615)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 20615)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20615)	Specifies the ShortCut property for client menu items.
	UseDefaultApp (see page 20616)	Use the default application to open the file.
	Visible (see page 20616)	Specifies the Visible property for client controls and menu items.

152.1.13.4.1 TFileOpen.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

152.1.13.4.2 TFileOpen.Dialog Property

Provides access the open dialog that the action executes.

Pascal

```
property Dialog: TOpenDialog;
```

C++

```
__property TOpenDialog Dialog;
```

Description

Use Dialog to configure the open dialog before the action fires. For example, you can set the open dialog's default extension or initially selected file at design time using the Object Inspector. From an OnAccept (see page 20617) event handler, use Dialog to read the selected file name.

When using a file open action, you do not place a TOpenDialog component on the form or data module at design time. Instead, you add the TFileOpen (see page 20605) action to your action list and then use the Dialog property. The open dialog component is created by the TFileOpen (see page 20605) constructor.

See Also

onaccept (see page 20543)

oncancel (see page 20543)

FileName (see page 11424)

152.1.13.4.3 TFileOpen.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

152.1.13.4.4 TFileOpen.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

152.1.13.4.5 TFileOpen.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

152.1.13.4.6 TFileOpen.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

152.1.13.4.7 TFileOpen.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

152.1.13.4.8 TFileOpen.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

152.1.13.4.9 TFileOpen.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

152.1.13.4.10 TFileOpen.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

152.1.13.4.11 TFileOpen.UseDefaultApp Property

Use the default application to open the file.

Pascal

```
property UseDefaultApp: Boolean;
```

C++

```
__property Boolean UseDefaultApp;
```

Description

When set to True, UseDefaultApp automatically uses the default application as specified in the registry to open the file. Otherwise, TFileOpen (see page 20605) displays the FileOpen dialog. The default is False.

See Also

onaccept (see page 20543)

oncancel (see page 20543)

FileName (see page 11424)

152.1.13.4.12 TFileOpen.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

152.1.13.5 TFileOpen Events

The events of the TFileOpen class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 20617)	Occurs immediately before the dialog appears.
	OnAccept (see page 20617)	This is OnAccept, a member of class TFileOpen.
	OnCancel (see page 20617)	This is OnCancel, a member of class TFileOpen.

	OnHint (see page 20617)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20618)	Occurs when the application is idle or when the action list updates.

152.1.13.5.1 TFileOpen.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 20540)

Execute (see page 11283)

PropertyName

152.1.13.5.2 TFileOpen.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TFileOpen.

152.1.13.5.3 TFileOpen.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TFileOpen.

152.1.13.5.4 TFileOpen.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

152.1.13.5.5 TFileOpen.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
```

```
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

152.1.14 TFileOpenWith Class

TFileOpenWith is the standard action for displaying the dialog that lets users choose what application to user for opening a file.

Class Hierarchy

Pascal

```
TFileOpenWith = class(TFileOpen);
```

C++

```
class TFileOpenWith : public TFileOpen;
```

File

StdActns

Description

Add TFileOpenWith to an action list to add the Open With dialog to your application. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to display the Open With dialog and then use the chosen application to open the file specified by the FileName (see page 20625) property. If the FileName (see page 20625) property was not previously set, TFileOpenWith first displays an Open dialog in which the user can select a file name before displaying the Open With dialog for the user to specify what application to use when opening that file.

Note: TFileOpenWith always displays the Open With dialog, even if the specified file already has an extension with a registered association. To let users open a file by letting Windows choose the application based on the extension (or bring up the Open With dialog automatically when there is no association for the extension), use the TFileRun action instead.

You do not need to do anything to launch the selected application. TFileOpenWith automatically opens the file using the application that the user selects.

Note: The Dialog (see page 20612) property of TFileOpenWith represents the Open dialog in which a user selects the file to

open, not the Open With dialog in which they choose the application to user for opening that file. Similarly, the OnAccept (see page 20617) and OnCancel (see page 20617) events do not occur after the action displays the Open With dialog – rather, they occur after the Open dialog that appears when the FileName (see page 20625) property is not preset.

See Also

TFileSaveAs (see page 20654)

TFileOpen (see page 20605)

TOpenPicture (see page 11965)

TFileRun (see page 11872)

152.1.14.1 TFileOpenWith Members

The following tables list the members exposed by TFileOpenWith.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDialog (see page 20539)	This is FDialog, a member of class TCommonDialogAction.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAccept (see page 20540)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 20540)	This is DoCancel, a member of class TCommonDialogAction.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDialog (see page 20593)	This is GetDialog, a member of class TFileAction.
V	GetDialogClass (see page 20541)	This is GetDialogClass, a member of class TCommonDialogAction.
V	GetDialogClass (see page 20611)	This is GetDialogClass, a member of class TFileOpen.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20541)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetupDialog (see page 20542)	This is SetupDialog, a member of class TCommonDialogAction.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnAccept (see page 20543)	This is OnAccept, a member of class TCommonDialogAction.
	OnCancel (see page 20543)	This is OnCancel, a member of class TCommonDialogAction.
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 20538)	Creates an instance of TCommonDialogAction.
	Create (see page 20610)	Creates an instance of TFileOpen (see page 20605).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20538)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 20624)	Displays the Open With dialog and then opens a file with the application the user selects.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.

V	HandlesTarget (see page 20541)	Indicates whether the action can execute its dialog.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
R	ExecuteResult (see page 20542)	Indicates the result of executing the associated dialog.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.

	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	AfterOpen (see page 20627)	Occurs after an application completes opening an Open With dialog.
	BeforeExecute (see page 20628)	Occurs immediately before the dialog appears.
	OnAccept (see page 20617)	This is OnAccept, a member of class TFileOpen.
	OnCancel (see page 20617)	This is OnCancel, a member of class TFileOpen.
	OnHint (see page 20617)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20628)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 20612)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
R	Dialog (see page 20612)	Provides access the open dialog that the action executes.
	Enabled (see page 20612)	Indicates whether client controls and menu items are enabled.
	FileName (see page 20625)	Specifies the file to be opened when the action executes.
	HelpContext (see page 20626)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20626)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20627)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20614)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20615)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 20615)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20615)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	UseDefaultApp (see page 20616)	Use the default application to open the file.
	Visible (see page 20616)	Specifies the Visible property for client controls and menu items.

152.1.14.2 TFileOpenWith Methods

The methods of the TFileOpenWith class are listed here.

Public Methods

	Name	Description
V	ExecuteTarget (see page 20624)	Displays the Open With dialog and then opens a file with the application the user selects.

152.1.14.2.1 TFileOpenWith.ExecuteTarget Method

Displays the Open With dialog and then opens a file with the application the user selects.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and does the following:

1. If the FileName (see page 20625) property is not set, it displays the Open dialog (specified by the Dialog (see page 20612) property) to let the user select a file to open.
2. If it displays the Open dialog, it generates an OnAccept (see page 20617) or OnCancel (see page 20617) event, depending on whether the user exited the Open dialog by clicking OK or Cancel.
3. It displays the Open With dialog for the user to choose an application with which to view the file specified by FileName (see page 20625) or selected in the Open dialog.
4. It launches the selected application, displaying the specified File.

See Also

FileName (see page 20625)

Dialog (see page 20612)

onaccept (see page 20543)

oncancel (see page 20543)

152.1.14.3 TFileOpenWith Properties

The properties of the TFileOpenWith class are listed here.

Published Properties

	Name	Description
	FileName (see page 20625)	Specifies the file to be opened when the action executes.
	HelpContext (see page 20626)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20626)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20627)	Indicates the mechanism for client controls and menu items to use when invoking help.

152.1.14.3.1 TFileOpenWith.FileName Property

Specifies the file to be opened when the action executes.

Pascal

```
property FileName: TFileName;
```

C++

```
__property TFileName FileName;
```

Description

Set FileName to specify the file to open with the application that the user selects in the Open With dialog. If the FileName property is not set, then when the action executes, it displays an Open dialog (specified by the Dialog (see page 20612) property) in which the user can select a file before displaying the Open With dialog.

Note: When the action displays the Open dialog, it uses the selected file to set FileName. Thus, if you want to let the user select

a file the next time the action executes, you must clear the FileName property in an OnAccept (see page 20617) event handler.

See Also

Dialog (see page 20612)

ExecuteTarget (see page 20624)

onaccept (see page 20543)

152.1.14.3.2 TFileOpenWith.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

152.1.14.3.3 TFileOpenWith.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

152.1.14.3.4 TFileOpenWith.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

152.1.14.4 TFileOpenWith Events

The events of the TFileOpenWith class are listed here.

Published Events

	Name	Description
	AfterOpen (see page 20627)	Occurs after an application completes opening an Open With dialog.
	BeforeExecute (see page 20628)	Occurs immediately before the dialog appears.
	OnUpdate (see page 20628)	Occurs when the application is idle or when the action list updates.

152.1.14.4.1 TFileOpenWith.AfterOpen Event

Occurs after an application completes opening an Open With dialog.

Pascal

```
property AfterOpen: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterOpen;
```

Description

Write an AfterOpen event handler to take specific action immediately after an application opens the Open With dialog, but before the user responds to the dialog.

152.1.14.4.2 TFileOpenWith.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 20540)

Execute (see page 11283)

PropertyName

152.1.14.4.3 TFileOpenWith.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
```

```
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

152.1.15 TFilePageSetup Class

TFilePageSetup is the standard action for displaying a page setup dialog.

Class Hierarchy

Pascal

```
TFilePageSetup = class(TCommonDialogAction);
```

C++

```
class TFilePageSetup : public TCommonDialogAction;
```

File

StdActns

Description

Add TFilePageSetup to an action list to add a common page setup dialog to your application. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to display the page setup dialog when invoked. Because the page setup dialog automatically configures the pages as specified when the user clicks OK, there is usually no need to write an OnAccept (see page 20640) or OnCancel (see page 20640) event handler to continue processing after the dialog shuts down.

See Also

TPrinterSetupDialog (see page 11478)

TPrintDlg (see page 20713)

152.1.15.1 TFilePageSetup Members

The following tables list the members exposed by TFilePageSetup.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDialog (see page 20539)	This is FDialog, a member of class TCommonDialogAction.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAccept (see page 20540)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 20540)	This is DoCancel, a member of class TCommonDialogAction.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDialogClass (see page 20634)	This is GetDialogClass, a member of class TFilePageSetup.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20541)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetupDialog (see page 20542)	This is SetupDialog, a member of class TCommonDialogAction.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 20538)	Creates an instance of TCommonDialogAction.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20538)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	ExecuteTarget (see page 20540)	Executes the associated dialog and generates an OnAccept (see page 20543) or OnCancel (see page 20543) event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 20541)	Indicates whether the action can execute its dialog.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.

	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
R	ExecuteResult (see page 20542)	Indicates the result of executing the associated dialog.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	BeforeExecute (see page 20639)	Occurs immediately before the dialog appears.
	OnAccept (see page 20640)	This is OnAccept, a member of class TFilePageSetup.

	OnCancel (see page 20640)	This is OnCancel, a member of class TFilePageSetup.
	OnHint (see page 20640)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20640)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 20635)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
R	Dialog (see page 20635)	Provides access the page setup dialog that the action executes.
	Enabled (see page 20635)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20636)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20636)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20637)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20637)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20638)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 20638)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20638)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 20639)	Specifies the Visible property for client controls and menu items.

152.1.15.2 TFilePageSetup Methods

The methods of the TFilePageSetup class are listed here.

Protected Methods

	Name	Description
V	GetDialogClass (see page 20634)	This is GetDialogClass, a member of class TFilePageSetup.

152.1.15.2.1 TFilePageSetup.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TFilePageSetup.

152.1.15.3 TFilePageSetup Properties

The properties of the TFilePageSetup class are listed here.

Published Properties

	Name	Description
	Caption (see page 20635)	Represents the caption of client controls and menu items.
R	Dialog (see page 20635)	Provides access the page setup dialog that the action executes.
	Enabled (see page 20635)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20636)	Indicates the help context ID for client controls and menu items.

	HelpKeyword (see page 20636)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20637)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20637)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20638)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 20638)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20638)	Specifies the ShortCut property for client menu items.
	Visible (see page 20639)	Specifies the Visible property for client controls and menu items.

152.1.15.3.1 TFilePageSetup.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

152.1.15.3.2 TFilePageSetup.Dialog Property

Provides access the page setup dialog that the action executes.

Pascal

```
property Dialog: TPageSetupDialog;
```

C++

```
__property TPageSetupDialog Dialog;
```

Description

When using a file page setup action, you do not place a TPageSetupDialog component on the form or data module at design time. Instead, you add the TFilePageSetup (see page 20629) action to your action list. If you want to configure the dialog, for example to assign a help context, you can access the dialog component using the Dialog property. The page setup dialog component is created by the TFilePageSetup (see page 20629) constructor.

See Also

Create (see page 20538)

Create

152.1.15.3.3 TFilePageSetup.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

152.1.15.3.4 TFilePageSetup.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

152.1.15.3.5 TFilePageSetup.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

152.1.15.3.6 TFilePageSetup.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

152.1.15.3.7 TFilePageSetup.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

152.1.15.3.8 TFilePageSetup.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

152.1.15.3.9 TFilePageSetup.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

152.1.15.3.10 TFilePageSetup.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

152.1.15.3.11 TFilePageSetup.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

152.1.15.4 TFilePageSetup Events

The events of the TFilePageSetup class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 20639)	Occurs immediately before the dialog appears.
	OnAccept (see page 20640)	This is OnAccept, a member of class TFilePageSetup.
	OnCancel (see page 20640)	This is OnCancel, a member of class TFilePageSetup.
	OnHint (see page 20640)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20640)	Occurs when the application is idle or when the action list updates.

152.1.15.4.1 TFilePageSetup.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 20540)

Execute (see page 11283)

PropertyName

152.1.15.4.2 TFilePageSetup.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TFilePageSetup.

152.1.15.4.3 TFilePageSetup.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TFilePageSetup.

152.1.15.4.4 TFilePageSetup.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

152.1.15.4.5 TFilePageSetup.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

152.1.16 TFilePrintSetup Class

TFilePrintSetup is the standard action for displaying a printer setup dialog.

Class Hierarchy

Pascal

```
TFilePrintSetup = class(TCommonDialogAction);
```

C++

```
class TFilePrintSetup : public TCommonDialogAction;
```

File

StdActns

Description

Add TFilePrintSetup to an action list to add a common printer setup dialog to your application. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to display the printer setup dialog when invoked. Because the printer setup dialog automatically configures the printers as specified when the user clicks OK, there is usually no need to write an OnAccept (see page 20652) or OnCancel (see page 20652) event handler to continue processing after the dialog shuts down.

See Also

TPrinterSetupDialog (see page 11478)

TPrintDlg (see page 20713)

152.1.16.1 TFilePrintSetup Members

The following tables list the members exposed by TFilePrintSetup.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDialog (see page 20539)	This is FDialog, a member of class TCommonDialogAction.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.

	AssignTo (see page 303)	Copies the properties of this action to a destination object.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoAccept (see page 20540)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 20540)	This is DoCancel, a member of class TCommonDialogAction.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDialogClass (see page 20647)	This is GetDialogClass, a member of class TFilePrintSetup.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 20541)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetupDialog (see page 20542)	This is SetupDialog, a member of class TCommonDialogAction.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.

	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 20538)	Creates an instance of TCommonDialogAction.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20538)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	ExecuteTarget (see page 20540)	Executes the associated dialog and generates an OnAccept (see page 20543) or OnCancel (see page 20543) event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 20541)	Indicates whether the action can execute its dialog.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
R	ExecuteResult (see page 20542)	Indicates the result of executing the associated dialog.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	BeforeExecute (see page 20652)	Occurs immediately before the dialog appears.
	OnAccept (see page 20652)	This is OnAccept, a member of class TFilePrintSetup.
	OnCancel (see page 20652)	This is OnCancel, a member of class TFilePrintSetup.
	OnHint (see page 20652)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20653)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 20647)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
R	Dialog (see page 20647)	Provides access the printer setup dialog that the action executes.
	Enabled (see page 20648)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20648)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20649)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20649)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20650)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20650)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 20650)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20651)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 20651)	Specifies the Visible property for client controls and menu items.

152.1.16.2 TFilePrintSetup Methods

The methods of the TFilePrintSetup class are listed here.

Protected Methods

	Name	Description
V	GetDialogClass (see page 20647)	This is GetDialogClass, a member of class TFilePrintSetup.

152.1.16.2.1 TFilePrintSetup.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TFilePrintSetup.

152.1.16.3 TFilePrintSetup Properties

The properties of the TFilePrintSetup class are listed here.

Published Properties

	Name	Description
	Caption (see page 20647)	Represents the caption of client controls and menu items.
R	Dialog (see page 20647)	Provides access the printer setup dialog that the action executes.
	Enabled (see page 20648)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20648)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20649)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20649)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20650)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20650)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 20650)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20651)	Specifies the ShortCut property for client menu items.
	Visible (see page 20651)	Specifies the Visible property for client controls and menu items.

152.1.16.3.1 TFilePrintSetup.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

152.1.16.3.2 TFilePrintSetup.Dialog Property

Provides access the printer setup dialog that the action executes.

Pascal

```
property Dialog: TPrinterSetupDialog;
```

C++

```
__property TPrinterSetupDialog Dialog;
```

Description

When using a file print setup action, you do not place a TPrinterSetupDialog component on the form or data module at design time. Instead, you add the TFilePrintSetup (see page 20642) action to your action list. If you want to configure the dialog, for example to assign a help context, you can access the dialog component using the Dialog property. The printer setup dialog component is created by the TFilePrintSetup (see page 20642) constructor.

See Also

Create (see page 20538)

Create_c

152.1.16.3.3 TFilePrintSetup.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

152.1.16.3.4 TFilePrintSetup.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

152.1.16.3.5 TFilePrintSetup.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

152.1.16.3.6 TFilePrintSetup.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

152.1.16.3.7 TFilePrintSetup.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

152.1.16.3.8 TFilePrintSetup.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

152.1.16.3.9 TFilePrintSetup.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key

combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

152.1.16.3.10 TFilePrintSetup.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

152.1.16.3.11 TFilePrintSetup.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

152.1.16.4 TFilePrintSetup Events

The events of the TFilePrintSetup class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 20652)	Occurs immediately before the dialog appears.
	OnAccept (see page 20652)	This is OnAccept, a member of class TFilePrintSetup.

	OnCancel (see page 20652)	This is OnCancel, a member of class TFilePrintSetup.
	OnHint (see page 20652)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20653)	Occurs when the application is idle or when the action list updates.

152.1.16.4.1 TFilePrintSetup.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 20540)

Execute (see page 11283)

PropertyName

152.1.16.4.2 TFilePrintSetup.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TFilePrintSetup.

152.1.16.4.3 TFilePrintSetup.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TFilePrintSetup.

152.1.16.4.4 TFilePrintSetup.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

152.1.16.4.5 TFilePrintSetup.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
```

```
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

152.1.17 TFileSaveAs Class

TFileSaveAs is the standard action for displaying a file save as dialog.

Class Hierarchy

Pascal

```
TFileSaveAs = class(TFileAction);
```

C++

```
class TFileSaveAs : public TFileAction;
```

File

StdActns

Description

Add TFileSaveAs to an action list to add a common file save dialog to your application. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to display a file save dialog when invoked. Use the OnAccept (see page 20664) event to respond after the user selects a file name in the file save dialog or the OnCancel (see page 20664) event to respond when the user cancels from the dialog. In the event handler for these events, you can read the file name from the dialog specified by the Dialog (see page 20660) property.

See Also

TFileOpen (see page 20605)

TSaveDialog (see page 11489)

TSavePicture (see page 12048)

152.1.17.1 TFileSaveAs Members

The following tables list the members exposed by TFileSaveAs.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDialog (see page 20539)	This is FDialog, a member of class TCommonDialogAction.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAccept (see page 20540)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 20540)	This is DoCancel, a member of class TCommonDialogAction.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDialog (see page 20593)	This is GetDialog, a member of class TFileAction.
V	GetDialogClass (see page 20659)	This is GetDialogClass, a member of class TFileSaveAs.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20541)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.

	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetupDialog (see page 20542)	This is SetupDialog, a member of class TCommonDialogAction.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	FileName (see page 20593)	Specifies the file name for the file selection dialog.
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 20659)	Creates an instance of TFileSaveAs (see page 20654).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).

	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20538)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	ExecuteTarget (see page 20540)	Executes the associated dialog and generates an OnAccept (see page 20543) or OnCancel (see page 20543) event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 20541)	Indicates whether the action can execute its dialog.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
R	ExecuteResult (see page 20542)	Indicates the result of executing the associated dialog.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	BeforeExecute (see page 20663)	Occurs immediately before the dialog appears.
	OnAccept (see page 20664)	This is OnAccept, a member of class TFileSaveAs.
	OnCancel (see page 20664)	This is OnCancel, a member of class TFileSaveAs.
	OnHint (see page 20664)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20665)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 20660)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
R	Dialog (see page 20660)	Provides access the save dialog that the action executes.
	Enabled (see page 20661)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20661)	Indicates the help context ID for client controls and menu items.
	Hint (see page 20661)	Indicates the Help hint for client controls and menu items.

	ImageIndex (see page 20662)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 20662)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20663)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 20663)	Specifies the Visible property for client controls and menu items.

152.1.17.2 TFileSaveAs.Create Constructor

Creates an instance of TFileSaveAs (see page 20654).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TFileSaveAs(TComponent * AOwner);
```

Description

Most applications do not need to call the TFileSaveAs (see page 20654) constructor because actions are added at design time to an action manager or action list.

The TFileSaveAs (see page 20654) constructor creates an instance of its associated dialog and assigns that component as the value of the Dialog (see page 20660) property.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TFileSaveAs (see page 20654) instance.

See Also

Dialog (see page 20660)

Owner (see page 2807)

152.1.17.3 TFileSaveAs Methods

The methods of the TFileSaveAs class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 20659)	This is GetDialogClass, a member of class TFileSaveAs.

152.1.17.3.1 TFileSaveAs.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TFileSaveAs.

152.1.17.4 TFileSaveAs Properties

The properties of the TFileSaveAs class are listed here.

Published Properties

	Name	Description
	Caption (see page 20660)	Represents the caption of client controls and menu items.
R	Dialog (see page 20660)	Provides access the save dialog that the action executes.
	Enabled (see page 20661)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20661)	Indicates the help context ID for client controls and menu items.
	Hint (see page 20661)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20662)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 20662)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20663)	Specifies the ShortCut property for client menu items.
	Visible (see page 20663)	Specifies the Visible property for client controls and menu items.

152.1.17.4.1 TFileSaveAs.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

152.1.17.4.2 TFileSaveAs.Dialog Property

Provides access the save dialog that the action executes.

Pascal

```
property Dialog: TSaveDialog;
```

C++

```
__property TSaveDialog Dialog;
```

Description

Use Dialog to configure the save as dialog before the action fires. For example, you can set the save as dialog's default extension at design time using the Object Inspector. Dialog is also useful for reading the selected file name from an OnAccept (see page 20664) event handler.

When using a file save as action, you do not place a TSaveDialog component on the form or data module at design time. Instead, you add the TFileSaveAs (see page 20654) to your action list and then use the Dialog property. The save dialog component is created by the TFileSaveAs (see page 20654) constructor.

See Also

onaccept (see page 20543)

oncancel (see page 20543)

152.1.17.4.3 TFileSaveAs.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

152.1.17.4.4 TFileSaveAs.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

152.1.17.4.5 TFileSaveAs.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

152.1.17.4.6 TFileSaveAs.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

152.1.17.4.7 TFileSaveAs.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

152.1.17.4.8 TFileSaveAs.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

152.1.17.4.9 TFileSaveAs.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

152.1.17.5 TFileSaveAs Events

The events of the TFileSaveAs class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 20663)	Occurs immediately before the dialog appears.
	OnAccept (see page 20664)	This is OnAccept, a member of class TFileSaveAs.
	OnCancel (see page 20664)	This is OnCancel, a member of class TFileSaveAs.
	OnHint (see page 20664)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20665)	Occurs when the application is idle or when the action list updates.

152.1.17.5.1 TFileSaveAs.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 20540)

Execute (see page 11283)

PropertyName

152.1.17.5.2 TFileSaveAs.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TFileSaveAs.

152.1.17.5.3 TFileSaveAs.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TFileSaveAs.

152.1.17.5.4 TFileSaveAs.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

152.1.17.5.5 TFileSaveAs.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
```

```
  }
  procedure TMainForm.Action1Update(Sender: TObject);
  begin
    if ((Sender as TAction).Checked <> ToolBar1.Visible) then
      lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
      { Indicate whether ToolBar1 is currently visible }
      (Sender as TAction).Checked := ToolBar1.Visible;
  end;
```

See Also

UpdateAction (see page 2790)

152.1.18 TFontEdit Class

TFontEdit is the standard action for displaying a font selection dialog.

Class Hierarchy

Pascal

```
TFontEdit = class(TCommonDialogAction);
```

C++

```
class TFontEdit : public TCommonDialogAction;
```

File

StdActns

Description

Add TFontEdit to an action list to add a modal font selection dialog to your application. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to display a font dialog when invoked. Use the OnAccept (see page 20676) event to respond after the user selects a font in the dialog or the OnCancel (see page 20676) event to respond when the user cancels from the dialog. In the event handler, you can read the selected font from the dialog specified by the Dialog (see page 20672) property.

See Also

TFontDialog (see page 11394)

152.1.18.1 TFontEdit Members

The following tables list the members exposed by TFontEdit.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.

	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDialog (see page 20539)	This is FDialog, a member of class TCommonDialogAction.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAccept (see page 20540)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 20540)	This is DoCancel, a member of class TCommonDialogAction.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDialogClass (see page 20671)	This is GetDialogClass, a member of class TFontEdit.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20541)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetupDialog (see page 20542)	This is SetupDialog, a member of class TCommonDialogAction.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 20538)	Creates an instance of TCommonDialogAction.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 20538)	Frees an instance of TCustomAction.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".

	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	ExecuteTarget (see page 20540)	Executes the associated dialog and generates an OnAccept (see page 20543) or OnCancel (see page 20543) event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 20541)	Indicates whether the action can execute its dialog.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
	Update (see page 295)	Indicates whether the action updates itself.
	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
R	ExecuteResult (see page 20542)	Indicates the result of executing the associated dialog.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	BeforeExecute (see page 20676)	Occurs immediately before the dialog appears.
	OnAccept (see page 20676)	This is OnAccept, a member of class TFontEdit.
	OnCancel (see page 20676)	This is OnCancel, a member of class TFontEdit.
	OnHint (see page 20677)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20677)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 20671)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
R	Dialog (see page 20672)	Provides access the font selection dialog that the action executes.
	Enabled (see page 20672)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20672)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20673)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20673)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20674)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20674)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 20674)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20675)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 20675)	Specifies the Visible property for client controls and menu items.

152.1.18.2 TFontEdit Methods

The methods of the TFontEdit class are listed here.

Protected Methods

	Name	Description
V	GetDialogClass (see page 20671)	This is GetDialogClass, a member of class TFontEdit.

152.1.18.2.1 TFontEdit.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TFontEdit.

152.1.18.3 TFontEdit Properties

The properties of the TFontEdit class are listed here.

Published Properties

	Name	Description
	Caption (see page 20671)	Represents the caption of client controls and menu items.
R	Dialog (see page 20672)	Provides access the font selection dialog that the action executes.
	Enabled (see page 20672)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20672)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20673)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20673)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20674)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20674)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 20674)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20675)	Specifies the ShortCut property for client menu items.
	Visible (see page 20675)	Specifies the Visible property for client controls and menu items.

152.1.18.3.1 TFontEdit.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

152.1.18.3.2 TFontEdit.Dialog Property

Provides access the font selection dialog that the action executes.

Pascal

```
property Dialog: TFontDialog;
```

C++

```
__property TFontDialog Dialog;
```

Description

Use Dialog to configure the font selection dialog before the action fires or to read the selected font after the action executes the dialog. For example, you can set the font dialog's target device or limit the available font sizes at design time using the Object Inspector or read the Font property in an OnAccept (see page 20676) event handler to make use of the font that the user selects.

When using a font edit action, you do not place a TFontDialog component on the form or data module at design time. Instead, you add the TFontEdit (see page 20666) to your action list and then use the Dialog property. The font selection dialog component is created by the TFontEdit (see page 20666) constructor.

See Also

onaccept (see page 20543)

oncancel (see page 20543)

Font (see page 11401)

152.1.18.3.3 TFontEdit.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

152.1.18.3.4 TFontEdit.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

152.1.18.3.5 TFontEdit.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

152.1.18.3.6 TFontEdit.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

152.1.18.3.7 TFontEdit.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

152.1.18.3.8 TFontEdit.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

152.1.18.3.9 TFontEdit.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

152.1.18.3.10 TFontEdit.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

152.1.18.3.11 TFontEdit.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

152.1.18.4 TFontEdit Events

The events of the TFontEdit class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 20676)	Occurs immediately before the dialog appears.
	OnAccept (see page 20676)	This is OnAccept, a member of class TFontEdit.
	OnCancel (see page 20676)	This is OnCancel, a member of class TFontEdit.
	OnHint (see page 20677)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20677)	Occurs when the application is idle or when the action list updates.

152.1.18.4.1 TFontEdit.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 20540)

Execute (see page 11283)

PropertyName

152.1.18.4.2 TFontEdit.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TFontEdit.

152.1.18.4.3 TFontEdit.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TFontEdit.

152.1.18.4.4 TFontEdit.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

152.1.18.4.5 TFontEdit.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
```

```
        lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
      }
      pa->Checked = ToolBar1->Visible;
    }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

152.1.19 THelpAction Class

THelpAction is the base class for action objects that interact with the help system.

Class Hierarchy

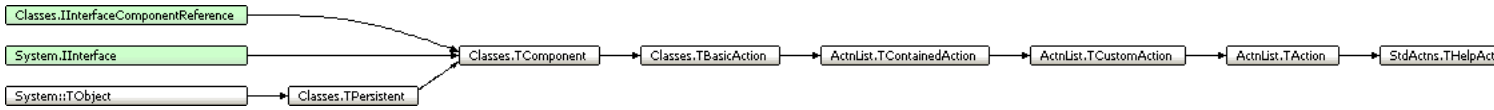

Pascal

```
THelpAction = class(TAction);
```

C++

```
class THelpAction : public TAction;
```

File

StdActns

Description

THelpAction implements the common behavior for actions that work the help system. Each descendant class performs a specific help function. Do not use THelpAction directly in an application. Instead, use a THelpAction descendant such as THelpContents (see page 20684), THelpContextAction (see page 20690), THelpOnHelp (see page 20696), or THelpTopicSearch (see

page 20701).

THelpAction ensures that the action is enabled as long as there is an application object to pass commands on to a registered help system. THelpAction descendants override the ExecuteTarget method to execute their specific help command.

See Also

THelpContents (see page 20684)

THelpContextAction (see page 20690)

THelpOnHelp (see page 20696)

THelpTopicSearch (see page 20701)

TApplication (see page 13660)

152.1.19.1 THelpAction Members

The following tables list the members exposed by THelpAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 20683)	Creates and initializes a TAction object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 20684)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 20684)	Enables or disables the action, depending on the availability of the Application object.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.19.2 THelpAction.Create Constructor

Creates and initializes a TAction object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall THelpAction(TComponent * AOwner);
```

Description

Applications do not usually create action objects explicitly. Actions are created automatically when you add them to an action manager or action list component at design time. When instantiating TAction at runtime, assign a TActionList or TActionManager component to the new action's ActionList property after calling Create.

See Also

ActionList (see page 295)

152.1.19.3 THelpAction Methods

The methods of the THelpAction class are listed here.

Public Methods

	Name	Description
	HandlesTarget (see page 20684)	Indicates whether the action can execute its function.
	UpdateTarget (see page 20684)	Enables or disables the action, depending on the availability of the Application object.

152.1.19.3.1 THelpAction.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

As implemented in THelpAction (see page 20678), HandlesTarget ignores the Target parameter and always returns true, because the help actions do not rely on a specific target object.

152.1.19.3.2 THelpAction.UpdateTarget Method

Enables or disables the action, depending on the availability of the Application object.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In THelpAction (see page 20678), UpdateTarget checks whether the global Application variable is assigned, because this object is required to pass help commands on to WinHelp HyperHelp. If Application is assigned, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

Application (see page 14383)

152.1.20 THelpContents Class

THelpContents brings up the contents for the default help system.

Class Hierarchy

Pascal

```
THelpContents = class(THelpAction);
```

C++

```
class THelpContents : public THelpAction;
```

File

StdActns

Description

Add THelpContents to an action list to allow users to bring up the help contents by using menu items or toolbar buttons. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to send the default help system a command to show the help contents.

On WinHelp, THelpContents displays the Help Topics dialog on the tab (Contents, Index or Find) that was last used.

See Also

THelpContextAction (see page 20690)

THelpOnHelp (see page 20696)

THelpTopicSearch (see page 20701)

152.1.20.1 THelpContents Members

The following tables list the members exposed by THelpContents.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.

	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 20683)	Creates and initializes a TAction object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 20690)	Displays a dialog that lists the help topics.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.

	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 20684)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.
V	UpdateTarget (see page 20684)	Enables or disables the action, depending on the availability of the Application object.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.

	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.20.2 THelpContents Methods

The methods of the THelpContents class are listed here.

Public Methods

	Name	Description
V	ExecuteTarget (see page 20690)	Displays a dialog that lists the help topics.

152.1.20.2.1 THelpContents.ExecuteTarget Method

Displays a dialog that lists the help topics.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget calls the application's HelpCommand method, telling it to list the help topics.

152.1.21 THelpContextAction Class

THelpContextAction brings up the help topic for the active control.

Class Hierarchy

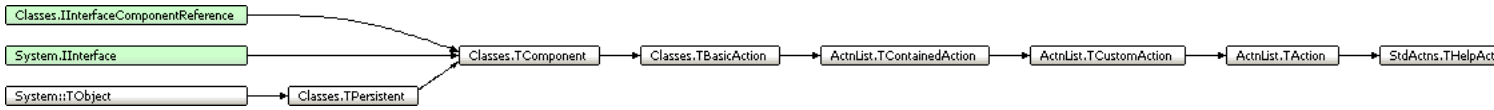

Pascal

```
THelpContextAction = class(THelpAction);
```

C++

```
class THelpContextAction : public THelpAction;
```

File

StdActns

Description

Add THelpContextAction to an action list to allow users to bring up the help for an active control. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to send the help engine a command to display the help topic specified by the active control's HelpContext property.

See Also

THelpContents (see page 20684)

THelpTopicSearch (see page 20701)

THelpOnHelp (see page 20696)

TControl (see page 5724)

TScreen (see page 14197)

152.1.21.1 THelpContextAction Members

The following tables list the members exposed by THelpContextAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 20683)	Creates and initializes a TAction object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.

	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 20695)	Displays the help topic associated with the active control's HelpContext property.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 20684)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 20695)	Enables or disables the action, depending on whether there is an active object.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.

	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.21.2 THelpContextAction Methods

The methods of the THelpContextAction class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 20695)	Displays the help topic associated with the active control's HelpContext property.
	UpdateTarget (see page 20695)	Enables or disables the action, depending on whether there is an active object.

152.1.21.2.1 THelpContextAction.ExecuteTarget Method

Displays the help topic associated with the active control's HelpContext property.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget calls the application's HelpCommand method, telling it to display the topic specified by the active control's HelpContext property. ExecuteTarget gets the active control from the global Screen object rather than using the Target parameter.

See Also

HelpCommand (see page 13678)

ActiveControl (see page 14206)

HelpContext (see page 5826)

152.1.21.2.2 THelpContextAction.UpdateTarget Method

Enables or disables the action, depending on whether there is an active object.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In THelpContextAction (see page 20690), UpdateTarget checks whether the global Screen object's ActiveConrol property is assigned. If there is an active control, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

ActiveControl (see page 14206)

152.1.22 THelpOnHelp Class

THelpOnHelp brings up a help topic on how to use Help.

Class Hierarchy

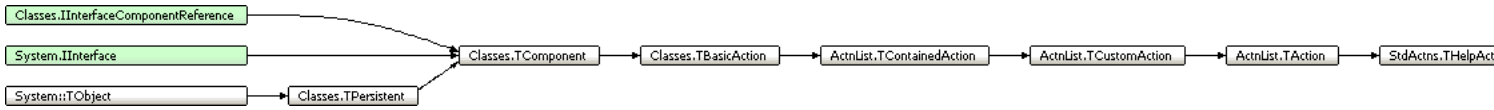

Pascal

```
THelpOnHelp = class(THelpAction);
```

C++

```
class THelpOnHelp : public THelpAction;
```

File

StdActns

Description

Add THelpOnHelp to an action list to allow users to bring up the help system that describes how to use help. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to send the default help system a command to display its help.

Note that on recent versions of Windows with a default help system of WinHelp, this brings up an HTML help file that does not describe the WinHelp system.

See Also

THelpContents (see page 20684)

THelpContextAction (see page 20690)

THelpTopicSearch (see page 20701)

152.1.22.1 THelpOnHelp Members

The following tables list the members exposed by THelpOnHelp.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 20683)	Creates and initializes a TAction object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.

	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 20701)	Brings up the help system's help file.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 20684)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.
V	UpdateTarget (see page 20684)	Enables or disables the action, depending on the availability of the Application object.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.

	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.22.2 THelpOnHelp Methods

The methods of the THelpOnHelp class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 20701)	Brings up the help system's help file.

152.1.22.2.1 THelpOnHelp.ExecuteTarget Method

Brings up the help system's help file.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget calls the application's HelpCommand method, telling it to display the Help system's own help file.

See Also

HelpCommand (see page 13678)

152.1.23 THelpTopicSearch Class

THelpTopicSearch brings up the help index.

Class Hierarchy

Pascal

```
THelpTopicSearch = class(THelpAction);
```

C++

```
class THelpTopicSearch : public THelpAction;
```

File

StdActns

Description

Add THelpTopicSearch to an action list to allow users to bring up the help index. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to send the default help system a command to display the help index.

See Also

THelpTopicSearch

THelpContextAction (see page 20690)

THelpOnHelp (see page 20696)

THelpTopicSearch

152.1.23.1 THelpTopicSearch Members

The following tables list the members exposed by THelpTopicSearch.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.

	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 20683)	Creates and initializes a TAction object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 20707)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 20684)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.
V	UpdateTarget (see page 20684)	Enables or disables the action, depending on the availability of the Application object.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.

	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.23.2 THelpTopicSearch Methods

The methods of the THelpTopicSearch class are listed here.

Public Methods

	Name	Description
V	ExecuteTarget (see page 20707)	Introduces an interface for invoking an action on a target client component or control.

152.1.23.2.1 THelpTopicSearch.ExecuteTarget Method

Introduces an interface for invoking an action on a target client component or control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

ExecuteTarget does nothing in TBasicAction. ExecuteTarget was introduced in TBasicAction so that descendants can override it to initiate the action on the target. For example, an edit action that performs copying might copy the contents of an edit control to the clipboard.

See Also

OnExecute (see page 2716)

HandlesTarget (see page 2713)

152.1.24 THintAction Class

THintAction is the action that executes when the user pauses the mouse over a control or menu item that has a hint.

Class Hierarchy

Pascal

```
THintAction = class(TCustomAction);
```

C++

```
class THintAction : public TCustomAction;
```

File

StdActns

Description

Do not use THintAction in an action manager or action list the way you would use other predefined actions. THintAction is not intended to work with a client control. Rather, it is used by the application object.

When the user pauses the mouse over a control or menu item that defines its Hint (see page 20713) property, the application responds in one of two ways:

If the application has an OnHint event handler, it generates an OnHint event.

If there is no OnHint event handler, the application executes a THintAction object.

When THintAction executes, it calls the ExecuteAction method of potential target controls until a control is found to respond to the action. Some controls, such as TStatusBar, override the ExecuteAction method to respond to THintAction.

Component writers can override the ExecuteAction method of a new component to respond to THintAction by displaying hint

messages. Such an ExecuteAction method might look like the following:

```
function TMyControl.ExecuteAction(Action: TBasicAction): Boolean;
var
  HintText: string;
begin
if Action is THintAction then
begin
    HintText := THintAction(Action).Hint;
    { do something to display the hint text }
    Result := True;
end
else Result := inherited ExecuteAction(Action);
end;
bool __fastcall TMyControl::ExecuteAction(TBasicAction *Action)
{
  THintAction *pHint = dynamic_cast<THintAction *>(Action);
if (pHint)
  {
    // do something to display pHint->Hint
returntrue;
  }
return TComponent::ExecuteAction(Action);
}
```

See Also

TControl (see page 5724)

TMenuItem (see page 16102)

TComponent (see page 2762)

TApplication (see page 13660)

152.1.24.1 THintAction Members

The following tables list the members exposed by THintAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.

	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 20712)	Creates an instance of THintAction (see page 20707).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.

	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Properties

	Name	Description
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Hint (see page 20713)	Indicates the Help hint for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

152.1.24.2 THintAction.Create Constructor

Creates an instance of THintAction (see page 20707).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall THintAction(TComponent * AOwner);
```

Description

Typically, you do not need to create THintAction (see page 20707) instances. THintAction (see page 20707) is created by the application object when it detects the need to display a hint.

You can, however, create a THintAction (see page 20707) object, set its Hint (see page 20713) property, and then call its Execute method to cause other controls in the application (such as status bars) to display a message for some application-generated condition (such as a status message).

AOwner is the component, typically the application object, that is responsible for freeing the THintAction (see page 20707) object.

See Also

Owner (see page 2807)

Hint (see page 312)

Execute (see page 304)

152.1.24.3 THintAction Properties

The properties of the THintAction class are listed here.

Published Properties

	Name	Description
	Hint (see page 20713)	Indicates the Help hint for client controls and menu items.

152.1.24.3.1 THintAction.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

152.1.25 TPrintDlg Class

TPrintDlg is the standard action for displaying a print dialog.

Class Hierarchy

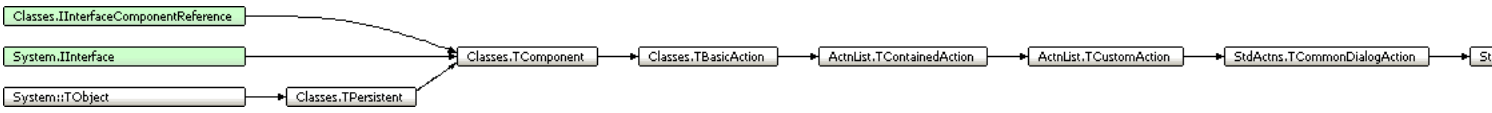

Pascal

```
TPrintDlg = class(TCommonDialogAction);
```

C++

```
class TPrintDlg : public TCommonDialogAction;
```

File

StdActns

Description

Add TPrintDlg to an action list to add a print dialog to your application. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to display the print dialog (TPrintDialog) when invoked. Write an OnAccept (see page 20724) event handler to perform the actual printing when the user clicks OK. You can read details about the user's selections in the dialog from the Dialog (see page 20719) property.

See Also

TPrintDialog (see page 11458)

TFilePrintSetup (see page 20642)

152.1.25.1 TPrintDlg Members

The following tables list the members exposed by TPrintDlg.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDialog (see page 20539)	This is FDialog, a member of class TCommonDialogAction.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAccept (see page 20540)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 20540)	This is DoCancel, a member of class TCommonDialogAction.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDialogClass (see page 20718)	This is GetDialogClass, a member of class TPrintDlg.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20541)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.

	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetupDialog (see page 20542)	This is SetupDialog, a member of class TCommonDialogAction.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 20538)	Creates an instance of TCommonDialogAction.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20538)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	ExecuteTarget (see page 20540)	Executes the associated dialog and generates an OnAccept (see page 20543) or OnCancel (see page 20543) event.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 20541)	Indicates whether the action can execute its dialog.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
R	ExecuteResult (see page 20542)	Indicates the result of executing the associated dialog.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	BeforeExecute (see page 20723)	Occurs immediately before the dialog appears.
	OnAccept (see page 20724)	This is OnAccept, a member of class TPrintDlg.
	OnCancel (see page 20724)	This is OnCancel, a member of class TPrintDlg.
	OnHint (see page 20724)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20724)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 20718)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
R	Dialog (see page 20719)	Provides access the print dialog that the action executes.
	Enabled (see page 20719)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20720)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20720)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20721)	Indicates the mechanism for client controls and menu items to use when invoking help.

	Hint (see page 20721)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20721)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 20722)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20722)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 20723)	Specifies the Visible property for client controls and menu items.

152.1.25.2 TPrintDlg Methods

The methods of the TPrintDlg class are listed here.

Protected Methods

	Name	Description
V	GetDialogClass (see page 20718)	This is GetDialogClass, a member of class TPrintDlg.

152.1.25.2.1 TPrintDlg.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TPrintDlg.

152.1.25.3 TPrintDlg Properties

The properties of the TPrintDlg class are listed here.

Published Properties

	Name	Description
	Caption (see page 20718)	Represents the caption of client controls and menu items.
R	Dialog (see page 20719)	Provides access the print dialog that the action executes.
	Enabled (see page 20719)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20720)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20720)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20721)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20721)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20721)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 20722)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20722)	Specifies the ShortCut property for client menu items.
	Visible (see page 20723)	Specifies the Visible property for client controls and menu items.

152.1.25.3.1 TPrintDlg.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

152.1.25.3.2 TPrintDlg.Dialog Property

Provides access the print dialog that the action executes.

Pascal

```
property Dialog: TPrintDialog;
```

C++

```
__property TPrintDialog Dialog;
```

Description

When using a print dialog action, you do not place a TPrintDialog component on the form or data module at design time. Instead, you add the TPrintDlg (see page 20713) action to your action list. If you want to configure the dialog, for example to set the Options that are available to users, you can access the dialog component using the Dialog property. From an OnAccept (see page 20724) event handler, use Dialog to read the choices that the user made in the print dialog and then perform the printing based on those choices.

The print dialog component is created by the TPrintDlg (see page 20713) constructor.

See Also

Create_d

Create_c

Create

Options (see page 11471)

onaccept (see page 20543)

152.1.25.3.3 TPrintDlg.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

152.1.25.3.4 TPrintDlg.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

152.1.25.3.5 TPrintDlg.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

152.1.25.3.6 TPrintDlg.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

152.1.25.3.7 TPrintDlg.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

152.1.25.3.8 TPrintDlg.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

152.1.25.3.9 TPrintDlg.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

152.1.25.3.10 TPrintDlg.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

152.1.25.3.11 TPrintDlg.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

152.1.25.4 TPrintDlg Events

The events of the TPrintDlg class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 20723)	Occurs immediately before the dialog appears.
	OnAccept (see page 20724)	This is OnAccept, a member of class TPrintDlg.
	OnCancel (see page 20724)	This is OnCancel, a member of class TPrintDlg.
	OnHint (see page 20724)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20724)	Occurs when the application is idle or when the action list updates.

152.1.25.4.1 TPrintDlg.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 20540)

Execute (see page 11283)

PropertyName

152.1.25.4.2 TPrintDlg.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TPrintDlg.

152.1.25.4.3 TPrintDlg.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TPrintDlg.

152.1.25.4.4 TPrintDlg.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

152.1.25.4.5 TPrintDlg.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

152.1.26 TSearchAction Class

TSearchAction is the base class for action objects that display a search or search and replace dialog.

Class Hierarchy

Pascal

```
TSearchAction = class(TCommonDialogAction);
```

C++

```
class TSearchAction : public TCommonDialogAction;
```

File

StdActns

Description

TSearchAction implements the common behavior for actions that display a modeless dialog where the user can enter a search string for searching an edit control. Do not use TSearchAction directly in an application. Instead, use one of the TSearchAction descendants: TSearchFind (see page 20734) or TSearchFindFirst (see page 20747) for a simple locate dialog, TSearchReplace (see page 20765) for a locate-and-replace dialog.

TSearchAction descendants are designed to work with an edit control (TCustomEdit descendant) as a target. They automatically disable themselves if the active control in a form is not an edit control.

Unlike other dialog actions, search actions represent a modeless dialog. As a result, the OnAccept (see page 20543) event occurs not when the user exits the dialog, but when the dialog first appears or comes into focus. When the user directs the dialog to search for or replace a string, the search action automatically performs that search, selecting or replacing text in the target edit control.

See Also

TSearchFind (see page 20734)

TSearchReplace (see page 20765)

TSearchFindNext (see page 20753)

TCustomEdit (see page 21232)

TSearchFindFirst (see page 20747)

152.1.26.1 TSearchAction Members

The following tables list the members exposed by TSearchAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FControl (see page 20731)	This is FControl, a member of class TSearchAction.
	FDialog (see page 20539)	This is FDialog, a member of class TCommonDialogAction.
	FFindFirst (see page 20731)	This is FFindFirst, a member of class TSearchAction.

	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAccept (see page 20540)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 20540)	This is DoCancel, a member of class TCommonDialogAction.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDialogClass (see page 20541)	This is GetDialogClass, a member of class TCommonDialogAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 20733)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetupDialog (see page 20542)	This is SetupDialog, a member of class TCommonDialogAction.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	BeforeExecute (see page 20543)	Occurs immediately before the dialog appears.
	OnAccept (see page 20543)	This is OnAccept, a member of class TCommonDialogAction.
	OnCancel (see page 20543)	This is OnCancel, a member of class TCommonDialogAction.
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 20730)	Creates and initializes an instance of TSearchAction (see page 20725).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 20731)	Frees an instance of TCustomAction.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 20732)	Launches the associated dialog.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 20733)	Indicates whether the action can execute its dialog.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Search (see page 20733)	Selects the next occurrence of the search string in the target edit control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 20734)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.

R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
R	ExecuteResult (see page 20542)	Indicates the result of executing the associated dialog.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Properties

	Name	Description
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

152.1.26.2 TSearchAction.Create Constructor

Creates and initializes an instance of TSearchAction (see page 20725).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSearchAction(TComponent * AOwner);
```

Description

Most applications do not need to call the TSearchAction (see page 20725) constructor because actions are added at design time to action lists or action managers.

The TSearchAction (see page 20725) constructor creates an instance of its associated dialog, which descendant classes surface as the Dialog property. It then assigns the Search (see page 20733) method as an event handler for the dialog's OnFind event, so that the action responds automatically when user's direct the dialog to search for a specified text string.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TSearchAction (see page 20725) instance.

See Also

Dialog (see page 20740)

Dialog (see page 20772)

Owner (see page 2807)

Search (see page 20733)

OnFind (see page 11392)

152.1.26.3 TSearchAction.Destroy Destructor

Frees an instance of TCustomAction.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSearchAction();
```

Description

Do not call the destructor directly in an application. Instead, call Free. Free ensures that the action is not nil before calling the destructor.

If the action is contained in an action list or action manager, Destroy removes the action from that list.

See Also

Create (see page 301)

Actions (see page 325)

152.1.26.4 TSearchAction Fields

The fields of the TSearchAction class are listed here.

Protected Fields

	Name	Description
	FControl (see page 20731)	This is FControl, a member of class TSearchAction.
	FFindFirst (see page 20731)	This is FFindFirst, a member of class TSearchAction.

152.1.26.4.1 TSearchAction.FControl Field

Pascal

```
FControl: TCustomEdit;
```

C++

```
TCustomEdit FControl;
```

Description

This is FControl, a member of class TSearchAction.

152.1.26.4.2 TSearchAction.FFindFirst Field

Pascal

```
FFindFirst: Boolean;
```

C++

```
Boolean FFindFirst;
```

Description

This is FFindFirst, a member of class TSearchAction.

152.1.26.5 TSearchAction Methods

The methods of the TSearchAction class are listed here.

Protected Methods

	Name	Description
	Notification (see page 20733)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	ExecuteTarget (see page 20732)	Launches the associated dialog.
	HandlesTarget (see page 20733)	Indicates whether the action can execute its dialog.
	Search (see page 20733)	Selects the next occurrence of the search string in the target edit control.
	UpdateTarget (see page 20734)	Enables or disables the action, depending on the currently active control in the form.

152.1.26.5.1 TSearchAction.ExecuteTarget Method

Launches the associated dialog.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget first launches its search or search and replace dialog. This is a modeless dialog, so the result of the dialog's Execute method simply indicates whether the dialog successfully appeared or was brought into focus. This result is set as the value of the ExecuteResult (see page 20542) property, and an OnAccept (see page 20543) or OnCancel (see page 20543) event occurs, depending on whether the dialog was brought into focus.

See Also

onaccept (see page 20543)

oncancel (see page 20543)

Execute (see page 11381)

ExecuteResult (see page 20542)

152.1.26.5.2 TSearchAction.HandlesTarget Method

Indicates whether the action can execute its dialog.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method lets the action object indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

HandlesTarget returns true if the target is an edit control that has keyboard focus.

152.1.26.5.3 TSearchAction.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TBasicAction overrides the Notification method to check whether the component that is the value of the ActionComponent property is about to be freed. If so, it sets the ActionComponent property to nil (Delphi) or NULL (C++).

See Also

ActionComponent (see page 2715)

152.1.26.5.4 TSearchAction.Search Method

Selects the next occurrence of the search string in the target edit control.

Pascal

```
procedure Search(Sender: TObject); virtual;
```

C++

```
virtual __fastcall Search(TObject * Sender);
```

Description

Most applications have no need to call the Search method: the TSearchAction (see page 20725) constructor automatically assigns Search as the OnFind event handler of its associated dialog. The dialog's OnFind event occurs when the user directs the dialog to find a specified string.

Search searches the target edit control for the string that is the FindText property of the search dialog, using the options specified by the dialog's Options property. If it finds the specified string, it selects it in the edit control. Otherwise, it displays a

message to the user indicating that the string was not found.

Note: If the search action is an instance of TSearchFind (see page 20734), Search looks for the first occurrence of FindText after the current selection in the edit control. If the search action is an instance of TSearchFindFirst (see page 20747), Search looks for the first occurrence of FindText, regardless of the current selection.

See Also

OnFind (see page 11392)

Options (see page 11386)

FindText (see page 11384)

152.1.26.5.5 TSearchAction.UpdateTarget Method

Enables or disables the action, depending on the currently active control in the form.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TSearchAction (see page 20725), UpdateTarget checks whether the current target object (specified by the Target parameter) is an edit control. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled (see page 310)

152.1.27 TSearchFind Class

TSearchFind is the standard action for displaying a find dialog.

Class Hierarchy

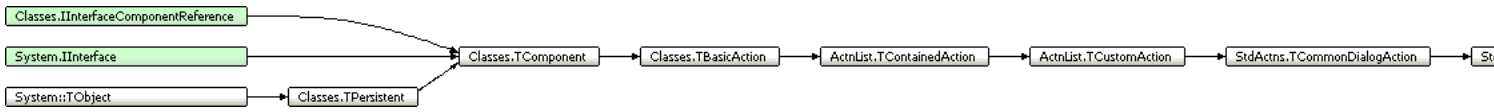

Pascal

```
TSearchFind = class(TSearchAction);
```

C++

```
class TSearchFind : public TSearchAction;
```

File

StdActns

Description

Add TSearchFind to an action list to add a modeless find dialog to your application. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to display the find dialog or bring it to the fore.

TSearchFind is designed to work with an edit control (TCustomEdit descendant) as a target. It automatically disables itself if the active control in a form is not an edit control.

Search (see page 20733) actions represent a modeless dialog. As a result, the OnAccept (see page 20745) event occurs not when the user exits the dialog, but when the dialog first appears or comes into focus. When the user directs the dialog to search for a string, TSearchFind automatically performs that search, selecting the specified text in the target edit control.

See Also

TFindDialog (see page 11376)

TSearchReplace (see page 20765)

TSearchFindNext (see page 20753)

TCustomEdit (see page 21232)

TSearchFindFirst (see page 20747)

152.1.27.1 TSearchFind Members

The following tables list the members exposed by TSearchFind.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FControl (see page 20731)	This is FControl, a member of class TSearchAction.
	FDialog (see page 20539)	This is FDialog, a member of class TCommonDialogAction.
	FFindFirst (see page 20731)	This is FFindFirst, a member of class TSearchAction.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAccept (see page 20540)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 20540)	This is DoCancel, a member of class TCommonDialogAction.

	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDialogClass (see page 20740)	This is GetDialogClass, a member of class TSearchFind.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 20541)	Responds when components are created or destroyed.
	Notification (see page 20733)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetupDialog (see page 20542)	This is SetupDialog, a member of class TCommonDialogAction.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 20538)	Creates an instance of TCommonDialogAction.
V	Create (see page 20730)	Creates and initializes an instance of TSearchAction (see page 20725).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 20538)	Frees an instance of TCustomAction.
V	Destroy (see page 20731)	Frees an instance of TCustomAction.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
V	ExecuteTarget (see page 20540)	Executes the associated dialog and generates an OnAccept (see page 20543) or OnCancel (see page 20543) event.
V	ExecuteTarget (see page 20732)	Launches the associated dialog.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 20541)	Indicates whether the action can execute its dialog.
V	HandlesTarget (see page 20733)	Indicates whether the action can execute its dialog.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Search (see page 20733)	Selects the next occurrence of the search string in the target edit control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.
V	UpdateTarget (see page 20734)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
R	ExecuteResult (see page 20542)	Indicates the result of executing the associated dialog.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	BeforeExecute (see page 20745)	Occurs immediately before the dialog appears.
	OnAccept (see page 20745)	This is OnAccept, a member of class TSearchFind.
	OnCancel (see page 20745)	This is OnCancel, a member of class TSearchFind.
	OnHint (see page 20745)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20746)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 20740)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
R	Dialog (see page 20740)	Provides access the find dialog that the action executes.
	Enabled (see page 20741)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20741)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20742)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20742)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20743)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20743)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 20743)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20744)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 20744)	Specifies the Visible property for client controls and menu items.

152.1.27.2 TSearchFind Methods

The methods of the TSearchFind class are listed here.

Protected Methods

	Name	Description
V	GetDialogClass (see page 20740)	This is GetDialogClass, a member of class TSearchFind.

152.1.27.2.1 TSearchFind.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TSearchFind.

152.1.27.3 TSearchFind Properties

The properties of the TSearchFind class are listed here.

Published Properties

	Name	Description
	Caption (see page 20740)	Represents the caption of client controls and menu items.
R	Dialog (see page 20740)	Provides access the find dialog that the action executes.
	Enabled (see page 20741)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20741)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20742)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20742)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20743)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20743)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 20743)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20744)	Specifies the ShortCut property for client menu items.
	Visible (see page 20744)	Specifies the Visible property for client controls and menu items.

152.1.27.3.1 TSearchFind.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

152.1.27.3.2 TSearchFind.Dialog Property

Provides access the find dialog that the action executes.

Pascal

```
property Dialog: TFindDialog;
```

C++

```
__property TFindDialog Dialog;
```

Description

Use Dialog to configure the find dialog before the action fires or to close the modeless dialog programmatically. For example, you can set the find dialog's options at design time using the Object Inspector or call the dialog's CloseDialog method to close it.

When using a search-find action, you do not place a TFindDialog component on the form or data module at design time. Instead, you add the TSearchFind (see page 20734) action to your action list and then use the Dialog property. The find dialog component is created by the TSearchFind (see page 20734) constructor.

See Also

ExecuteTarget (see page 20732)

CloseDialog (see page 11381)

152.1.27.3.3 TSearchFind.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

152.1.27.3.4 TSearchFind.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

152.1.27.3.5 TSearchFind.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

152.1.27.3.6 TSearchFind.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

152.1.27.3.7 TSearchFind.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

152.1.27.3.8 TSearchFind.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

152.1.27.3.9 TSearchFind.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key

combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

152.1.27.3.10 TSearchFind.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

152.1.27.3.11 TSearchFind.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

152.1.27.4 TSearchFind Events

The events of the TSearchFind class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 20745)	Occurs immediately before the dialog appears.
	OnAccept (see page 20745)	This is OnAccept, a member of class TSearchFind.

	OnCancel (see page 20745)	This is OnCancel, a member of class TSearchFind.
	OnHint (see page 20745)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20746)	Occurs when the application is idle or when the action list updates.

152.1.27.4.1 TSearchFind.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 20540)

Execute (see page 11283)

PropertyName

152.1.27.4.2 TSearchFind.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TSearchFind.

152.1.27.4.3 TSearchFind.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TSearchFind.

152.1.27.4.4 TSearchFind.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

152.1.27.4.5 TSearchFind.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
```

```
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

152.1.28 TSearchFindFirst Class

TSearchFindFirst is the standard action for displaying a find dialog that looks for the first occurrence of a string.

Class Hierarchy

Pascal

```
TSearchFindFirst = class(TSearchFind);
```

C++

```
class TSearchFindFirst : public TSearchFind;
```

File

StdActns

Description

Add TSearchFindFirst to an action list to add a modeless find dialog to your application. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to display the find dialog or bring it to the fore.

TSearchFindFirst is designed to work with an edit control (TCustomEdit descendant) as a target. It automatically disables itself if the active control in a form is not an edit control.

Search (see page 20733) actions represent a modeless dialog. As a result, the OnAccept (see page 20745) event occurs not when the user exits the dialog, but when the dialog first appears or comes into focus. When the user directs the dialog to search for a string, TSearchFindFirst automatically performs that search, selecting the specified text in the target edit control. Unlike the TSearchFind (see page 20734) action, which locates the first occurrence of the search string following the cursor position in the edit control, TSearchFindFirst locates the first occurrence of the search string, regardless of the cursor position.

See Also

TFindDialog (see page 11376)

TSearchReplace (see page 20765)

TSearchFindNext (see page 20753)

TCustomEdit (see page 21232)

TSearchFind (see page 20734)

152.1.28.1 TSearchFindFirst Members

The following tables list the members exposed by TSearchFindFirst.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FControl (see page 20731)	This is FControl, a member of class TSearchAction.
	FDialog (see page 20539)	This is FDialog, a member of class TCommonDialogAction.
	FFindFirst (see page 20731)	This is FFindFirst, a member of class TSearchAction.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAccept (see page 20540)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 20540)	This is DoCancel, a member of class TCommonDialogAction.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDialogClass (see page 20541)	This is GetDialogClass, a member of class TCommonDialogAction.
V	GetDialogClass (see page 20740)	This is GetDialogClass, a member of class TSearchFind.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.

	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 20541)	Responds when components are created or destroyed.
	Notification (see page 20733)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetupDialog (see page 20542)	This is SetupDialog, a member of class TCommonDialogAction.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	BeforeExecute (see page 20543)	Occurs immediately before the dialog appears.
	OnAccept (see page 20543)	This is OnAccept, a member of class TCommonDialogAction.
	OnCancel (see page 20543)	This is OnCancel, a member of class TCommonDialogAction.
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.

	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 20752)	Creates and initializes an instance of TSearchFindFirst (see page 20747).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20538)	Frees an instance of TCustomAction.
	Destroy (see page 20731)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	ExecuteTarget (see page 20540)	Executes the associated dialog and generates an OnAccept (see page 20543) or OnCancel (see page 20543) event.
	ExecuteTarget (see page 20732)	Launches the associated dialog.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 20541)	Indicates whether the action can execute its dialog.
	HandlesTarget (see page 20733)	Indicates whether the action can execute its dialog.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.

	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Search (see page 20733)	Selects the next occurrence of the search string in the target edit control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.
V	UpdateTarget (see page 20734)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
R	ExecuteResult (see page 20542)	Indicates the result of executing the associated dialog.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.

	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	BeforeExecute (see page 20745)	Occurs immediately before the dialog appears.
	OnAccept (see page 20745)	This is OnAccept, a member of class TSearchFind.
	OnCancel (see page 20745)	This is OnCancel, a member of class TSearchFind.
	OnHint (see page 20745)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20746)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 20740)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
R	Dialog (see page 20740)	Provides access the find dialog that the action executes.
	Enabled (see page 20741)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20741)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20742)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20742)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20743)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20743)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 20743)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20744)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 20744)	Specifies the Visible property for client controls and menu items.

152.1.28.2 TSearchFindFirst.Create Constructor

Creates and initializes an instance of TSearchFindFirst (see page 20747).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSearchFindFirst(TComponent * AOwner);
```

Description

Most applications do not need to call the TSearchFindFirst (see page 20747) constructor because actions are added at design time to an action manager or action list.

The TSearchFindFirst (see page 20747) constructor creates an instance of its associated dialog and assigns that component as the value of the Dialog (see page 20740) property. It then assigns the Search (see page 20733) method as an event handler for the dialog's OnFind event, so that the action responds automatically when user's direct the dialog to search for a

specified text string.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TSearchFindFirst (see page 20747) instance.

See Also

Dialog (see page 20740)

Owner (see page 2807)

Search (see page 20733)

OnFind (see page 11392)

152.1.29 TSearchFindNext Class

TSearchFindNext is the standard action for locating the next instance of a string in an edit control.

Class Hierarchy

Pascal

```
TSearchFindNext = class(TCustomAction);
```

C++

```
class TSearchFindNext : public TCustomAction;
```

File

StdActns

Description

Add TSearchFindNext to an action list to add the ability to repeat a search initiated by a TSearchFind (see page 20734) or TSearchFindFirst (see page 20747) action. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to repeat the search initiated by the TSearchFind (see page 20734) action's dialog.

TSearchFindNext objects must be associated with a TSearchFind (see page 20734) action by setting the SearchFind (see page 20762) property.

See Also

TSearchFind (see page 20734)

TSearchFindFirst (see page 20747)

152.1.29.1 TSearchFindNext Members

The following tables list the members exposed by TSearchFindNext.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 20757)	Creates an instance of TSearchFindNext (see page 20753).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 20758)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 20758)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 20759)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnHint (see page 20764)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20764)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 20759)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Enabled (see page 20760)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20760)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20761)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20761)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20762)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20762)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SearchFind (see page 20762)	Associates the TSearchFindNext (see page 20753) action with a TSearchFind (see page 20734) action that initiates the search.
	SecondaryShortCuts (see page 20763)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20763)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 20763)	Specifies the Visible property for client controls and menu items.

152.1.29.2 TSearchFindNext.Create Constructor

Creates an instance of TSearchFindNext (see page 20753).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSearchFindNext(TComponent * AOwner);
```

Description

Most applications do not need to call the TSearchFindNext (see page 20753) constructor because actions are added at design time to an action manager or action list.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TSearchFindNext (see page 20753) instance.

See Also

Owner (see page 2807)

Actions (see page 325)

152.1.29.3 TSearchFindNext Methods

The methods of the TSearchFindNext class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 20758)	Introduces an interface for invoking an action on a target client component or control.
	HandlesTarget (see page 20758)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	UpdateTarget (see page 20759)	Enables or disables the action, depending on the currently active control in the form.

152.1.29.3.1 TSearchFindNext.ExecuteTarget Method

Introduces an interface for invoking an action on a target client component or control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

ExecuteTarget does nothing in TBasicAction. ExecuteTarget was introduced in TBasicAction so that descendants can override it to initiate the action on the target. For example, an edit action that performs copying might copy the contents of an edit control to the clipboard.

See Also

OnExecute (see page 2716)

HandlesTarget (see page 2713)

152.1.29.3.2 TSearchFindNext.HandlesTarget Method

Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

HandlesTarget does nothing in TBasicAction. HandlesTarget was introduced in TBasicAction so that descendants can override it to check the type and state of a target. Handles target can verify any information about a target that is relevant for the action. HandlesTarget returns true if the target meets the specified criteria, false otherwise.

See Also

ExecuteTarget (see page 2712)

152.1.29.3.3 TSearchFindNext.UpdateTarget Method

Enables or disables the action, depending on the currently active control in the form.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the potential target changes so that actions can update themselves accordingly.

In TSearchFindNext (see page 20753), UpdateTarget ignores the current target object (specified by the Target parameter). Instead, it checks the associated TSearchFind (see page 20734) action (the SearchFind (see page 20762) property), and sets Enabled (see page 20760) to true if the search dialog has already specified a search string. If SearchFind (see page 20762) is not set, or if there is no search string in its associated dialog, UpdateTarget sets Enabled (see page 20760) to false.

See Also

Enabled (see page 310)

SearchFind (see page 20762)

FindText (see page 11384)

152.1.29.4 TSearchFindNext Properties

The properties of the TSearchFindNext class are listed here.

Published Properties

	Name	Description
	Caption (see page 20759)	Represents the caption of client controls and menu items.
	Enabled (see page 20760)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20760)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20761)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20761)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20762)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20762)	Indicates the ImageIndex property for client controls and menu items.
	SearchFind (see page 20762)	Associates the TSearchFindNext (see page 20753) action with a TSearchFind (see page 20734) action that initiates the search.
	SecondaryShortCuts (see page 20763)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20763)	Specifies the ShortCut property for client menu items.
	Visible (see page 20763)	Specifies the Visible property for client controls and menu items.

152.1.29.4.1 TSearchFindNext.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

152.1.29.4.2 TSearchFindNext.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

152.1.29.4.3 TSearchFindNext.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

152.1.29.4.4 TSearchFindNext.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

152.1.29.4.5 TSearchFindNext.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

152.1.29.4.6 TSearchFindNext.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

152.1.29.4.7 TSearchFindNext.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

152.1.29.4.8 TSearchFindNext.SearchFind Property

Associates the TSearchFindNext (see page 20753) action with a TSearchFind (see page 20734) action that initiates the search.

Pascal

```
property SearchFind: TSearchFind;
```

C++

```
__property TSearchFind SearchFind;
```

Description

SearchFind is the TSearchFind (see page 20734) action that displays the modeless search dialog whose current search should be repeated. If SearchFind is not set, then the TSearchFindNext (see page 20753) action remains disabled.

TSearchFindNext (see page 20753) uses the SearchFind property to obtain a search string from its associated dialog and to

search its associated control.

See Also

executetarget (see page 20758)

Enabled (see page 310)

Search (see page 20733)

152.1.29.4.9 TSearchFindNext.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

152.1.29.4.10 TSearchFindNext.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

152.1.29.4.11 TSearchFindNext.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

152.1.29.5 TSearchFindNext Events

The events of the TSearchFindNext class are listed here.

Published Events

	Name	Description
	OnHint (see page 20764)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20764)	Occurs when the application is idle or when the action list updates.

152.1.29.5.1 TSearchFindNext.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

152.1.29.5.2 TSearchFindNext.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
```

```
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

152.1.30 TSearchReplace Class

TSearchReplace is the standard action for displaying a find-and-replace dialog.

Class Hierarchy

Pascal

```
TSearchReplace = class(TSearchAction);
```

C++

```
class TSearchReplace : public TSearchAction;
```

File

StdActns

Description

Add TSearchReplace to an action list to add a modeless find-and-replace dialog to your application. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to display the find-and-replace dialog or bring it to the fore.

TSearchReplace is designed to work with an edit control (TCustomEdit descendant) as a target. It automatically disables itself if the active control in a form is not an edit control.

Search (see page 20733)/replace actions represent a modeless dialog. As a result, the OnAccept (see page 20777) event occurs not when the user exits the dialog, but when the dialog first appears or comes into focus. When the user directs the dialog to replace one or all instances of a string, TSearchReplace automatically performs the search and replaces the specified text in the target edit control.

See Also

TReplaceDialog (see page 11482)

TSearchFind (see page 20734)

TCustomEdit (see page 21232)

TSearchFindFirst (see page 20747)

152.1.30.1 TSearchReplace Members

The following tables list the members exposed by TSearchReplace.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FControl (see page 20731)	This is FControl, a member of class TSearchAction.
	FDialog (see page 20539)	This is FDialog, a member of class TCommonDialogAction.
	FFindFirst (see page 20731)	This is FFindFirst, a member of class TSearchAction.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAccept (see page 20540)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 20540)	This is DoCancel, a member of class TCommonDialogAction.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDialogClass (see page 20771)	This is GetDialogClass, a member of class TSearchReplace.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20541)	Responds when components are created or destroyed.
V	Notification (see page 20733)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetupDialog (see page 20542)	This is SetupDialog, a member of class TCommonDialogAction.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 20538)	Creates an instance of TCommonDialogAction.
	Create (see page 20730)	Creates and initializes an instance of TSearchAction (see page 20725).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 20538)	Frees an instance of TCustomAction.
	Destroy (see page 20731)	Frees an instance of TCustomAction.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.

	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 20771)	Launches the associated dialog and assigns an event handler to replace found text.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 20541)	Indicates whether the action can execute its dialog.
	HandlesTarget (see page 20733)	Indicates whether the action can execute its dialog.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Search (see page 20733)	Selects the next occurrence of the search string in the target edit control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
	Update (see page 295)	Indicates whether the action updates itself.
	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.

V	UpdateTarget (see page 20734)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
R	ExecuteResult (see page 20542)	Indicates the result of executing the associated dialog.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	BeforeExecute (see page 20777)	Occurs immediately before the dialog appears.
	OnAccept (see page 20777)	This is OnAccept, a member of class TSearchReplace.
	OnCancel (see page 20777)	This is OnCancel, a member of class TSearchReplace.
	OnHint (see page 20777)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20778)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 20772)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
R	Dialog (see page 20772)	Provides access the find-and-replace dialog that the action executes.
	Enabled (see page 20773)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20773)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20774)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20774)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20775)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20775)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 20775)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20776)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 20776)	Specifies the Visible property for client controls and menu items.

152.1.30.2 TSearchReplace Methods

The methods of the TSearchReplace class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 20771)	This is GetDialogClass, a member of class TSearchReplace.

Public Methods

	Name	Description
	ExecuteTarget (see page 20771)	Launches the associated dialog and assigns an event handler to replace found text.

152.1.30.2.1 TSearchReplace.ExecuteTarget Method

Launches the associated dialog and assigns an event handler to replace found text.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget first launches the search-and-replace dialog specified by Dialog (see page 20772). This is a modeless dialog, so the result of the dialog's Execute method simply indicates whether the dialog successfully appeared or was brought into focus. This result is set as the value of the ExecuteResult (see page 20542) property, and an OnAccept (see page 20777) event occurs if the dialog was brought into focus, or an OnCancel (see page 20777) event occurs if the dialog failed to appear.

Finally, ExecuteTarget sets the OnReplace event handler of the search-and-replace dialog to a private method that performs the replacement specified by the dialog.

See Also

onaccept (see page 20543)

oncancel (see page 20543)

Dialog (see page 20772)

Execute (see page 11381)

ExecuteResult (see page 20542)

OnReplace (see page 11488)

152.1.30.2.2 TSearchReplace.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TSearchReplace.

152.1.30.3 TSearchReplace Properties

The properties of the TSearchReplace class are listed here.

Published Properties

	Name	Description
	Caption (see page 20772)	Represents the caption of client controls and menu items.
R	Dialog (see page 20772)	Provides access the find-and-replace dialog that the action executes.
	Enabled (see page 20773)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 20773)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 20774)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 20774)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 20775)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 20775)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 20775)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 20776)	Specifies the ShortCut property for client menu items.
	Visible (see page 20776)	Specifies the Visible property for client controls and menu items.

152.1.30.3.1 TSearchReplace.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption (see page 5806)

Caption (see page 16136)

152.1.30.3.2 TSearchReplace.Dialog Property

Provides access the find-and-replace dialog that the action executes.

Pascal

```
property Dialog: TReplaceDialog;
```

C++

```
__property TReplaceDialog Dialog;
```

Description

Use Dialog to configure the find-and-replace dialog before the action fires or to close the modeless dialog programmatically. For example, you can set the dialog's options at design time using the Object Inspector or call the dialog's CloseDialog method to shut down the modeless find-and-replace dialog.

When using a search/replace action, you do not place a TReplaceDialog component on the form or data module at design time. Instead, you add the TSearchReplace (see page 20765) action to your action list and then use the Dialog property. The TReplaceDialog component is created by the TSearchReplace (see page 20765) constructor.

See Also

ExecuteTarget (see page 20732)

CloseDialog (see page 11381)

152.1.30.3.3 TSearchReplace.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled (see page 5821)

Enabled (see page 16145)

152.1.30.3.4 TSearchReplace.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext: THelpContext;
```

C++

```
__property THelpContext HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType (see page 312)

HelpKeyword (see page 311)

152.1.30.3.5 TSearchReplace.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword: string;
```

C++

```
__property AnsiString HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword (see page 5826)

HelpContext (see page 311)

HelpType (see page 312)

HelpKey

152.1.30.3.6 TSearchReplace.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType: THelpType;
```

C++

```
__property THelpType HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType (see page 5827)

HelpType

HelpContext (see page 311)

HelpKeyword (see page 311)

152.1.30.3.7 TSearchReplace.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint: string;
```

C++

```
__property AnsiString Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint (see page 5827)

Hint (see page 16148)

152.1.30.3.8 TSearchReplace.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex (see page 5135)

ImageIndex (see page 16148)

Images (see page 325)

152.1.30.3.9 TSearchReplace.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts: TShortCutList;
```

C++

```
__property TShortCutList SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut (see page 314)

Execute (see page 304)

152.1.30.3.10 TSearchReplace.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut: TShortCut;
```

C++

```
__property TShortCut ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut (see page 16152)

SecondaryShortCuts (see page 314)

152.1.30.3.11 TSearchReplace.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible (see page 16153)

Visible (see page 5847)

152.1.30.4 TSearchReplace Events

The events of the TSearchReplace class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 20777)	Occurs immediately before the dialog appears.
	OnAccept (see page 20777)	This is OnAccept, a member of class TSearchReplace.
	OnCancel (see page 20777)	This is OnCancel, a member of class TSearchReplace.
	OnHint (see page 20777)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 20778)	Occurs when the application is idle or when the action list updates.

152.1.30.4.1 TSearchReplace.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 20540)

Execute (see page 11283)

PropertyName

152.1.30.4.2 TSearchReplace.OnAccept Event

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

This is OnAccept, a member of class TSearchReplace.

152.1.30.4.3 TSearchReplace.OnCancel Event

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

This is OnCancel, a member of class TSearchReplace.

152.1.30.4.4 TSearchReplace.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint: THintEvent;
```

C++

```
__property THintEvent OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

152.1.30.4.5 TSearchReplace.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (pa->Checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     pa->Checked = ToolBar1->Visible;
   }
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
```

```
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction (see page 2790)

152.1.31 TWindowAction Class

TWindowAction is the base class for action objects that work with MDI forms.

Class Hierarchy

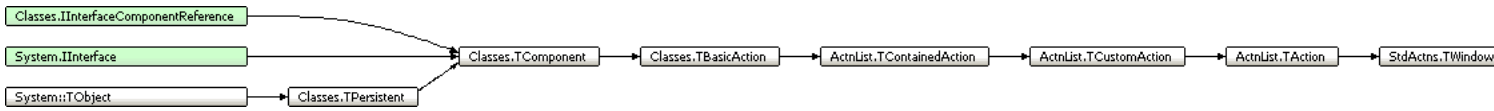

Pascal

```
TWindowAction = class(TAction);
```

C++

```
class TWindowAction : public TAction;
```

File

StdActns

Description

TWindowAction implements the common behavior for actions that work on MDI forms. Each descendant class performs a specific task, such as arranging, cascading, closing, tiling, or minimizing MDI child forms. Do not use TWindowAction directly in an application. Instead, use a TWindowAction descendant such as TWindowClose (see page 20798), TWindowCascade (see page 20792), TWindowMinimizeAll (see page 20804), TWindowArrange (see page 20786), TWindowTileHorizontal (see page 20809), or TWindowTileVertical (see page 20815).

TWindowAction ensures that the action is enabled as long as the target is an MDI parent form. TWindowAction descendants override the ExecuteTarget method to execute their specific function on the targets child forms.

See Also

TWindowArrange (see page 20786)

TWindowCascade (see page 20792)

TWindowClose (see page 20798)

TWindowMinimizeAll (see page 20804)

TWindowTileHorizontal (see page 20809)

TWindowTileVertical (see page 20815)

152.1.31.1 TWindowAction Members

The following tables list the members exposed by TWindowAction.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetForm (see page 20784)	This is GetForm, a member of class TWindowAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 20785)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 20784)	Creates and initializes a TAction object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.

	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 2712)	Introduces an interface for invoking an action on a target client component or control.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 20785)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

V	UpdateTarget (see page 20785)	Enables or disables the action, depending on whether the target form has MDI children.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	Form (see page 20786)	Provides access to the target of the action.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.

	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.31.2 TWindowAction.Create Constructor

Creates and initializes a TAction object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWindowAction(TComponent * AOwner);
```

Description

Applications do not usually create action objects explicitly. Actions are created automatically when you add them to an action manager or action list component at design time. When instantiating TAction at runtime, assign a TActionList or TActionManager component to the new action's ActionList property after calling Create.

See Also

ActionList (see page 295)

152.1.31.3 TWindowAction Methods

The methods of the TWindowAction class are listed here.

Protected Methods

	Name	Description
	GetForm (see page 20784)	This is GetForm, a member of class TWindowAction.
	Notification (see page 20785)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	HandlesTarget (see page 20785)	Indicates whether the action can execute its function.
	UpdateTarget (see page 20785)	Enables or disables the action, depending on whether the target form has MDI children.

152.1.31.3.1 TWindowAction.GetForm Method

Pascal

```
function GetForm(Target: TObject): TForm; virtual;
```

C++

```
virtual __fastcall TForm * GetForm(TObject * Target);
```

Description

This is GetForm, a member of class TWindowAction.

152.1.31.3.2 TWindowAction.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

As implemented in TWindowAction (see page 20779), HandlesTarget makes sure that the target is an MDI parent form. If the target is appropriate, HandlesTarget sets the Form (see page 20786) property.

See Also

Form (see page 20786)

FormStyle (see page 13912)

152.1.31.3.3 TWindowAction.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TBasicAction overrides the Notification method to check whether the component that is the value of the ActionComponent property is about to be freed. If so, it sets the ActionComponent property to nil (Delphi) or NULL (C++).

See Also

ActionComponent (see page 2715)

152.1.31.3.4 TWindowAction.UpdateTarget Method

Enables or disables the action, depending on whether the target form has MDI children.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TWindowAction (see page 20779), UpdateTarget checks that the target object is an MDI parent form with an MDIChildCount property greater than 0. If so, UpdateTarget sets the Enabled property to true. Otherwise, it sets Enabled to false.

See Also

Enabled (see page 310)

MDIChildCount (see page 13921)

152.1.31.4 TWindowAction Properties

The properties of the TWindowAction class are listed here.

Public Properties

	Name	Description
	Form (see page 20786)	Provides access to the target of the action.

152.1.31.4.1 TWindowAction.Form Property

Provides access to the target of the action.

Pascal

```
property Form: TCustomForm;
```

C++

```
__property TCustomForm Form;
```

Description

Form is the MDI parent form on which the action performs its function.

152.1.32 TWindowArrange Class

TWindowArrange arranges the icons of minimized MDI child forms.

Class Hierarchy

Pascal

```
TWindowArrange = class(TWindowAction);
```

C++

```
class TWindowArrange : public TWindowAction;
```

File

StdActns

Description

Add TWindowArrange to an action list to allow users to arrange the icons that represent minimized MDI child forms. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to call the ArrangeIcons method of the MDI parent form.

TWindowArrange is designed to work with an MDI parent form as a target. It automatically disables itself if the active form is not an MDI parent or if it has no MDI children.

See Also

TWindowClose (see page 20798)

TWindowCascade (see page 20792)

TWindowMinimizeAll (see page 20804)

TWindowTileHorizontal (see page 20809)

TWindowTileVertical (see page 20815)

TForm (see page 13992)

152.1.32.1 TWindowArrange Members

The following tables list the members exposed by TWindowArrange.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetForm (see page 20784)	This is GetForm, a member of class TWindowAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).

	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 20785)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.

	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 20784)	Creates and initializes a TAction object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 20791)	Arranges the icons of minimized MDI child forms
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 20785)	Indicates whether the action can execute its function.

V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.
V	UpdateTarget (see page 20785)	Enables or disables the action, depending on whether the target form has MDI children.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	Form (see page 20786)	Provides access to the target of the action.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.

	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.32.2 TWindowArrange Methods

The methods of the TWindowArrange class are listed here.

Public Methods

	Name	Description
V	ExecuteTarget (see page 20791)	Arranges the icons of minimized MDI child forms

152.1.32.2.1 TWindowArrange.ExecuteTarget Method

Arranges the icons of minimized MDI child forms

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active form when the action fires.

ExecuteTarget sets the Form (see page 20786) property to Target, which must be an MDI parent form. It then calls that form's ArrangeIcons method to arrange the icons of minimized MDI children.

See Also

Form (see page 20786)

ArrangeIcons (see page 14006)

152.1.33 TWindowCascade Class

TWindowCascade cascades the MDI child forms.

Class Hierarchy

Pascal

```
TWindowCascade = class(TWindowAction);
```

C++

```
class TWindowCascade : public TWindowAction;
```

File

StdActns

Description

Add TWindowCascade to an action list to allow users to arrange the MDI child forms in an overlapping cascade. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to call the Cascade method of the MDI parent form.

TWindowCascade is designed to work with an MDI parent form as a target. It automatically disables itself if the active form is not an MDI parent or if it has no MDI children.

See Also

TWindowClose (see page 20798)

TWindowArrange (see page 20786)

TWindowMinimizeAll (see page 20804)

TWindowTileHorizontal (see page 20809)

TWindowTileVertical (see page 20815)

TForm (see page 13992)

152.1.33.1 TWindowCascade Members

The following tables list the members exposed by TWindowCascade.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetForm (see page 20784)	This is GetForm, a member of class TWindowAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20785)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 20784)	Creates and initializes a TAction object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.

	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 20797)	Arranges the MDI child forms in an overlapping cascade.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 20785)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.
V	UpdateTarget (see page 20785)	Enables or disables the action, depending on whether the target form has MDI children.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	Form (see page 20786)	Provides access to the target of the action.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.33.2 TWindowCascade Methods

The methods of the TWindowCascade class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 20797)	Arranges the MDI child forms in an overlapping cascade.

152.1.33.2.1 TWindowCascade.ExecuteTarget Method

Arranges the MDI child forms in an overlapping cascade.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active form when the action fires.

ExecuteTarget sets the Form (see page 20786) property to Target, which must be an MDI parent form. It then calls that form's Cascade method to arrange MDI child windows.

See Also

Form (see page 20786)

Cascade (see page 14007)

152.1.34 TWindowClose Class

TWindowClose closes the active MDI child form.

Class Hierarchy

Pascal

```
TWindowClose = class(TWindowAction);
```

C++

```
class TWindowClose : public TWindowAction;
```

File

StdActns

Description

Add TWindowClose to an action list to allow users to close the active MDI child form. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to call the Close method of the active MDI child.

TWindowClose is designed to work with an MDI parent form as a target. It automatically disables itself if the active form is not an MDI parent or if it has no active MDI child.

See Also

TWindowArrange (see page 20786)

TWindowCascade (see page 20792)

TWindowMinimizeAll (see page 20804)

TWindowTileHorizontal (see page 20809)

TWindowTileVertical (see page 20815)

TCustomForm (see page 13849)

152.1.34.1 TWindowClose Members

The following tables list the members exposed by TWindowClose.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetForm (see page 20784)	This is GetForm, a member of class TWindowAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20785)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
V	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.

	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 20784)	Creates and initializes a TAction object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 20803)	Closes the active MDI child.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 20785)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 20803)	Enables or disables the action, depending on whether the target form an active child.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.

	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	Form (see page 20786)	Provides access to the target of the action.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.34.2 TWindowClose Methods

The methods of the TWindowClose class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 20803)	Closes the active MDI child.
	UpdateTarget (see page 20803)	Enables or disables the action, depending on whether the target form an active child.

152.1.34.2.1 TWindowClose.ExecuteTarget Method

Closes the active MDI child.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget sets the Form (see page 20786) property to Target, which must be an MDI parent form. It then obtains a reference to the active MDI child by reading the form's ActiveMDIChild property. It calls this child form's Close method to close the active child form.

See Also

Form (see page 20786)

ActiveMDIChild (see page 13897)

Close (see page 13867)

152.1.34.2.2 TWindowClose.UpdateTarget Method

Enables or disables the action, depending on whether the target form an active child.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TWindowClose (see page 20798), UpdateTarget checks that the target object is an MDI parent form with a non-nil (Delphi) or NULL (C++) ActiveMDIChild property. If so, UpdateTarget sets the Enabled property to true. Otherwise, it sets Enabled to false.

See Also

Enabled (see page 310)

ActiveMDIChild (see page 13897)

152.1.35 TWindowMinimizeAll Class

TWindowMinimizeAll minimizes all MDI child forms.

Class Hierarchy

Pascal

```
TWindowMinimizeAll = class(TWindowAction);
```

C++

```
class TWindowMinimizeAll : public TWindowAction;
```

File

StdActns

Description

Add TWindowMinimizeAll to an action list to allow users to minimize the MDI child forms in an MDI application. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the application to iterate through the MDI child forms, setting their WindowState to wsMinimized.

TWindowMinimizeAll is designed to work with an MDI parent form as a target. It automatically disables itself if the active form is not an MDI parent or if it has no MDI children.

See Also

TWindowClose (see page 20798)

TWindowArrange (see page 20786)

TWindowCascade (see page 20792)

TWindowTileHorizontal (see page 20809)

TWindowTileVertical (see page 20815)

TCustomForm (see page 13849)

TCustomForm (see page 13849)

152.1.35.1 TWindowMinimizeAll Members

The following tables list the members exposed by TWindowMinimizeAll.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetForm (see page 20784)	This is GetForm, a member of class TWindowAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2713)	Responds when components are created or destroyed.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 20785)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
V	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 307)	Sets the value of the Name property.

	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 20784)	Creates and initializes a TAction object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.

	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 20809)	Minimizes all MDI child forms of the target MDI parent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 20785)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.
V	UpdateTarget (see page 20785)	Enables or disables the action, depending on whether the target form has MDI children.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	Form (see page 20786)	Provides access to the target of the action.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.

	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.35.2 TWindowMinimizeAll Methods

The methods of the TWindowMinimizeAll class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 20809)	Minimizes all MDI child forms of the target MDI parent.

152.1.35.2.1 TWindowMinimizeAll.ExecuteTarget Method

Minimizes all MDI child forms of the target MDI parent.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active form when the action fires.

ExecuteTarget sets the Form (see page 20786) property to Target, which must be an MDI parent form. It then iterates through that form's MDI children, setting each child form's WindowState to wsMinimized.

See Also

Form (see page 20786)

MDIChildren (see page 13922)

WindowState (see page 13942)

152.1.36 TWindowTileHorizontal Class

TWindowTileHorizontal arranges MDI child forms so that they are all the same size, tiled horizontally.

Class Hierarchy

Pascal

```
TWindowTileHorizontal = class(TWindowAction);
```

C++

```
class TWindowTileHorizontal : public TWindowAction;
```

File

StdActns

Description

Add TWindowTileHorizontal to an action list to allow users to tile the MDI child forms horizontally. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the MDI parent form to tile its children horizontally.

TWindowTileHorizontal is designed to work with an MDI parent form as a target. It automatically disables itself if the active form is not an MDI parent or if it has no MDI children.

See Also

TWindowClose (see page 20798)

TWindowArrange (see page 20786)

TWindowCascade (see page 20792)

TWindowMinimizeAll (see page 20804)

TWindowTileVertical (see page 20815)

TCustomForm (see page 13849)

152.1.36.1 TWindowTileHorizontal Members

The following tables list the members exposed by TWindowTileHorizontal.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.

	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetForm (see page 20784)	This is GetForm, a member of class TWindowAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 20785)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
V	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
V	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 20784)	Creates and initializes a TAction object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 292)	Destroys the instance of the contained action.
V	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
V	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
V	Execute (see page 293)	Responds when the client control "fires".
V	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	ExecuteTarget (see page 20815)	Tiles the child forms of the target MDI parent horizontally.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
V	HandlesTarget (see page 20785)	Indicates whether the action can execute its function.
V	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.
V	UpdateTarget (see page 20785)	Enables or disables the action, depending on whether the target form has MDI children.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	Form (see page 20786)	Provides access to the target of the action.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.36.2 TWindowTileHorizontal Methods

The methods of the TWindowTileHorizontal class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 20815)	Tiles the child forms of the target MDI parent horizontally.

152.1.36.2.1 TWindowTileHorizontal.ExecuteTarget Method

Tiles the child forms of the target MDI parent horizontally.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active form when the action fires.

ExecuteTarget sets the Form (see page 20786) property to Target, which must be an MDI parent form. It then instructs the form to tile its child forms horizontally.

See Also

Form (see page 20786)

MDIChildren (see page 13922)

152.1.37 TWindowTileVertical Class

TWindowTileVertical arranges MDI child forms so that they are all the same size, tiled vertically.

Class Hierarchy

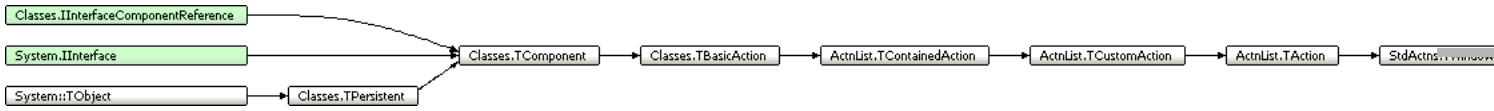

Pascal

```
TWindowTileVertical = class(TWindowAction);
```

C++

```
class TWindowTileVertical : public TWindowAction;
```

File

StdActns

Description

Add TWindowTileVertical to an action list to allow users to tile the MDI child forms vertically. Controls (see page 5711) such as menu items and tool buttons linked to this action cause the MDI parent form to tile its children vertically.

TWindowTileVertical is designed to work with an MDI parent form as a target. It automatically disables itself if the active form is not an MDI parent or if it has no MDI children.

See Also

TWindowClose (see page 20798)

TWindowArrange (see page 20786)

TWindowCascade (see page 20792)

TWindowMinimizeAll (see page 20804)

TWindowTileVertical

TCustomForm (see page 13849)

152.1.37.1 TWindowTileVertical Members

The following tables list the members exposed by TWindowTileVertical.

Protected Events

	Name	Description
	OnChange (see page 2716)	Occurs when one of the action's properties changes.

Protected Fields

	Name	Description
	FClients (see page 2711)	This is FClients, a member of class TBasicAction.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FImage (see page 302)	This is FImage, a member of class TCustomAction.
	FMask (see page 303)	This is FMask, a member of class TCustomAction.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 303)	Copies the properties of this action to a destination object.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	Change (see page 2712)	Generates an OnChange (see page 2716) event.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetForm (see page 20784)	This is GetForm, a member of class TWindowAction.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.

	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleShortCut (see page 306)	Executes the action when the user types on of its short cuts.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2713)	Responds when components are created or destroyed.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 20785)	Responds when components are created or destroyed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 294)	This is ReadState, a member of class TContainedAction.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetHelpContext (see page 306)	This is SetHelpContext, a member of class TCustomAction.
	SetHelpKeyword (see page 306)	This is SetHelpKeyword, a member of class TCustomAction.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 307)	Sets the value of the Name property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetOnExecute (see page 2714)	Assigns an OnExecute (see page 2716) event handler.
	SetParentComponent (see page 294)	This is SetParentComponent, a member of class TContainedAction.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SavedEnabledState (see page 313)	Saves the value of the Enabled (see page 310) property when the action is suspended.

Public Events

	Name	Description
	OnExecute (see page 2716)	Occurs when the client event that is linked to it fires.
	OnHint (see page 315)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 2718)	Occurs when the application is idle or when the action list updates.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 254)	Creates and initializes a TAction (see page 249) object.
	Create (see page 301)	Instantiates and initializes a TCustomAction (see page 297) object.
	Create (see page 2710)	Instantiates and initializes a TBasicAction (see page 2707) object.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 20784)	Creates and initializes a TAction object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 292)	Destroys the instance of the contained action.
	Destroy (see page 302)	Frees an instance of TCustomAction (see page 297).
	Destroy (see page 2711)	Disposes of an instance of a TBasicAction (see page 2707) object.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoHint (see page 303)	Generates an OnHint (see page 315) event.
	Execute (see page 293)	Responds when the client control "fires".
	Execute (see page 304)	Responds when a client control "fires".
	Execute (see page 2712)	Generates an OnExecute (see page 2716) event.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	ExecuteTarget (see page 20820)	Tiles the child forms of the target MDI parent vertically.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 293)	Returns the action list or action manager that contains the action.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandlesTarget (see page 2713)	Introduces an interface for verifying that the type, and state, of a target component or control are appropriate for the action.
	HandlesTarget (see page 20785)	Indicates whether the action can execute its function.
	HasParent (see page 294)	Indicates whether the action is contained in an action list or action manager.

	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RegisterChanges (see page 2713)	Associates the action with an action link.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UnRegisterChanges (see page 2714)	Breaks the association between the action and the action link.
V	Update (see page 295)	Indicates whether the action updates itself.
V	Update (see page 2714)	Provides an opportunity to execute centralized code when an application is idle.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateTarget (see page 2715)	Introduces an interface for a method of notifying a client when the action updates itself.
V	UpdateTarget (see page 20785)	Enables or disables the action, depending on whether the target form has MDI children.

Public Properties

	Name	Description
	ActionComponent (see page 2715)	Indicates the client component that caused this action to execute.
	ActionList (see page 295)	Indicates the action list or action manager that contains the action.
	AutoCheck (see page 308)	Controls (see page 5711) whether the Checked (see page 308) property toggles when the action executes.
	Caption (see page 308)	Represents the caption of client controls and menu items.
	Checked (see page 308)	Indicates whether client controls and menu items appear checked.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisableIfNoHandler (see page 310)	Indicates whether client components should be disabled if no handler is found.
	Enabled (see page 310)	Indicates whether client controls and menu items are enabled.
	Form (see page 20786)	Provides access to the target of the action.
	GroupIndex (see page 311)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 311)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 311)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 312)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 312)	Indicates the Help hint for client controls and menu items.

	ImageIndex (see page 313)	Indicates the ImageIndex property for client controls and menu items.
	Index (see page 296)	Specifies the index of the action in its action list or action manager.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SecondaryShortCuts (see page 314)	Specifies the short cuts (in addition to ShortCut (see page 314)) for triggering clients.
	ShortCut (see page 314)	Specifies the ShortCut property for client menu items.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 314)	Specifies the Visible property for client controls and menu items.

Published Events

	Name	Description
	OnExecute (see page 261)	Occurs when the client event that is linked to it fires.
	OnHint (see page 263)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 263)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	AutoCheck (see page 255)	Controls whether the Checked property toggles when the action executes.
	Caption (see page 255)	Represents the caption of client controls and menu items.
	Category (see page 296)	Specifies the category under which the action appears in its action list or action manager.
	Checked (see page 256)	Indicates whether client controls and menu items appear checked.
	Enabled (see page 257)	Indicates whether client controls and menu items are enabled.
	GroupIndex (see page 257)	Indicates whether the action acts like a radio button with other actions in the same action list.
	HelpContext (see page 258)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 258)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 259)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 259)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 259)	Indicates the ImageIndex property for client controls and menu items.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SecondaryShortCuts (see page 260)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 260)	Specifies the ShortCut property for client menu items.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Visible (see page 261)	Specifies the Visible property for client controls and menu items.

152.1.37.2 TWindowTileVertical Methods

The methods of the TWindowTileVertical class are listed here.

Public Methods

	Name	Description
V	ExecuteTarget (see page 20820)	Tiles the child forms of the target MDI parent vertically.

152.1.37.2.1 TWindowTileVertical.ExecuteTarget Method

Tiles the child forms of the target MDI parent vertically.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active form when the action fires.

ExecuteTarget sets the Form (see page 20786) property to Target, which must be an MDI parent form. It then instructs the form to tile its child forms vertically.

See Also

Form (see page 20786)

MDIChildren (see page 13922)

152.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TBrowseForFolderOption (see page 20822)	This is record StdActns.TBrowseForFolderOption.

152.2.1 StdActns.TBrowseForFolderOption Enumeration

Pascal

```
TBrowseForFolderOption = (
  bifBrowseForComputer,
  bifBrowseForPrinter,
  bifBrowseIncludeFiles,
  bifBrowseIncludeURLS,
  bifDontGoBelowDomain,
  bifEditBox,
  bifNewDialogStyle,
  bifNoNewFolderButton,
  bifNoTranslateTargets,
  bifReturnFSAncestors,
  bifReturnOnlyFSDirs,
  bifShareable,
  bifStatusText,
  bifUAHint,
  bifUseNewUI,
  bifValidate
);
```

C++

```
enum TBrowseForFolderOption {
  bifBrowseForComputer,
  bifBrowseForPrinter,
  bifBrowseIncludeFiles,
  bifBrowseIncludeURLS,
  bifDontGoBelowDomain,
  bifEditBox,
  bifNewDialogStyle,
  bifNoNewFolderButton,
  bifNoTranslateTargets,
  bifReturnFSAncestors,
  bifReturnOnlyFSDirs,
  bifShareable,
  bifStatusText,
  bifUAHint,
  bifUseNewUI,
  bifValidate
};
```

File

StdActns

Description

This is record StdActns.TBrowseForFolderOption.

152.3 Types

The following table lists types in this documentation.

Types

Name	Description
TBrowseForFolderCallbackEvent (see page 20824)	This is type StdActns.TBrowseForFolderCallbackEvent.
TBrowseOptions (see page 20824)	This is type StdActns.TBrowseOptions.
TCommonDialogClass (see page 20825)	This is type StdActns.TCommonDialogClass.

152.3.1 StdActns.TBrowseForFolderCallbackEvent Type

Pascal

```
TBrowseForFolderCallbackEvent = procedure (Sender: TBrowseForFolder; Wnd: HWND; uMsg: UINT;
lParam, lpData: LPARAM; var Result: Integer) of object;
```

C++

```
(Sender: TBrowseForFolder; Wnd: HWND; uMsg: UINT; lParam, lpData: LPARAM; var Result: Integer)
( TBrowseForFolderCallbackEvent)();
```

File

StdActns

Description

This is type StdActns.TBrowseForFolderCallbackEvent.

152.3.2 StdActns.TBrowseOptions Type

Pascal

```
TBrowseOptions = set of TBrowseForFolderOption;
```

C++

```
TBrowseForFolderOption TBrowseOptions;
```

File

StdActns

Description

This is type StdActns.TBrowseOptions.

152.3.3 StdActns.TCommonDialogClass Type

Pascal

```
TCommonDialogClass = class of TCommonDialog;
```

C++

```
TCommonDialog TCommonDialogClass;
```

File

StdActns

Description

This is type StdActns.TCommonDialogClass.

153 StdConvs Namespace

This is namespace StdConvs.

Topics

Name	Description
Functions (see page 20832)	The following table lists functions in this documentation.
Variables (see page 20834)	The following table lists variables in this documentation.
Constants (see page 20883)	The following table lists constants in this documentation.

Constants

Name	Description
CubicMetersPerAcreFoot (see page 20884)	This is constant StdConvs.CubicMetersPerAcreFoot.
CubicMetersPerAcreInch (see page 20884)	This is constant StdConvs.CubicMetersPerAcreInch.
CubicMetersPerCord (see page 20885)	This is constant StdConvs.CubicMetersPerCord.
CubicMetersPerCordFoot (see page 20885)	This is constant StdConvs.CubicMetersPerCordFoot.
CubicMetersPerCubicFoot (see page 20885)	This is constant StdConvs.CubicMetersPerCubicFoot.
CubicMetersPerCubicInch (see page 20886)	This is constant StdConvs.CubicMetersPerCubicInch.
CubicMetersPerCubicMile (see page 20886)	This is constant StdConvs.CubicMetersPerCubicMile.
CubicMetersPerCubicYard (see page 20886)	This is constant StdConvs.CubicMetersPerCubicYard.
CubicMetersPerUKBucket (see page 20887)	This is constant StdConvs.CubicMetersPerUKBucket.
CubicMetersPerUKBushel (see page 20887)	This is constant StdConvs.CubicMetersPerUKBushel.
CubicMetersPerUKGallon (see page 20887)	This is constant StdConvs.CubicMetersPerUKGallon.
CubicMetersPerUKGill (see page 20888)	This is constant StdConvs.CubicMetersPerUKGill.
CubicMetersPerUKOunce (see page 20888)	This is constant StdConvs.CubicMetersPerUKOunce.
CubicMetersPerUKPeck (see page 20888)	This is constant StdConvs.CubicMetersPerUKPeck.
CubicMetersPerUKPint (see page 20889)	This is constant StdConvs.CubicMetersPerUKPint.
CubicMetersPerUKPottle (see page 20889)	This is constant StdConvs.CubicMetersPerUKPottle.
CubicMetersPerUKQuart (see page 20889)	This is constant StdConvs.CubicMetersPerUKQuart.
CubicMetersPerUSDryBucket (see page 20890)	This is constant StdConvs.CubicMetersPerUSDryBucket.
CubicMetersPerUSDryBushel (see page 20890)	This is constant StdConvs.CubicMetersPerUSDryBushel.
CubicMetersPerUSDryGallon (see page 20890)	This is constant StdConvs.CubicMetersPerUSDryGallon.
CubicMetersPerUSDryPeck (see page 20891)	This is constant StdConvs.CubicMetersPerUSDryPeck.
CubicMetersPerUSDryPint (see page 20891)	This is constant StdConvs.CubicMetersPerUSDryPint.
CubicMetersPerUSDryQuart (see page 20891)	This is constant StdConvs.CubicMetersPerUSDryQuart.
CubicMetersPerUSFluidCup (see page 20892)	This is constant StdConvs.CubicMetersPerUSFluidCup.
CubicMetersPerUSFluidGallon (see page 20892)	This is constant StdConvs.CubicMetersPerUSFluidGallon.

CubicMetersPerUSFluidGill (see page 20892)	This is constant StdConvs.CubicMetersPerUSFluidGill.
CubicMetersPerUSFluidOunce (see page 20893)	This is constant StdConvs.CubicMetersPerUSFluidOunce.
CubicMetersPerUSFluidPint (see page 20893)	This is constant StdConvs.CubicMetersPerUSFluidPint.
CubicMetersPerUSFluidQuart (see page 20893)	This is constant StdConvs.CubicMetersPerUSFluidQuart.
CubicMetersPerUSFluidTablespoon (see page 20894)	This is constant StdConvs.CubicMetersPerUSFluidTablespoon.
CubicMetersPerUSFluidTeaspoon (see page 20894)	This is constant StdConvs.CubicMetersPerUSFluidTeaspoon.
GramsPerDrams (see page 20894)	This is constant StdConvs.GramsPerDrams.
GramsPerGrains (see page 20895)	This is constant StdConvs.GramsPerGrains.
GramsPerLongTons (see page 20895)	This is constant StdConvs.GramsPerLongTons.
GramsPerOunces (see page 20895)	This is constant StdConvs.GramsPerOunces.
GramsPerPound (see page 20896)	This is constant StdConvs.GramsPerPound.
GramsPerStones (see page 20896)	This is constant StdConvs.GramsPerStones.
GramsPerTons (see page 20896)	This is constant StdConvs.GramsPerTons.
MetersPerAstronomicalUnit (see page 20897)	This is constant StdConvs.MetersPerAstronomicalUnit.
MetersPerChain (see page 20897)	This is constant StdConvs.MetersPerChain.
MetersPerCubit (see page 20897)	This is constant StdConvs.MetersPerCubit.
MetersPerFathom (see page 20898)	This is constant StdConvs.MetersPerFathom.
MetersPerFoot (see page 20898)	This is constant StdConvs.MetersPerFoot.
MetersPerFurlong (see page 20898)	This is constant StdConvs.MetersPerFurlong.
MetersPerHand (see page 20899)	This is constant StdConvs.MetersPerHand.
MetersPerInch (see page 20899)	This is constant StdConvs.MetersPerInch.
MetersPerLightSecond (see page 20899)	This is constant StdConvs.MetersPerLightSecond.
MetersPerLightYear (see page 20900)	This is constant StdConvs.MetersPerLightYear.
MetersPerLink (see page 20900)	This is constant StdConvs.MetersPerLink.
MetersPerMile (see page 20900)	This is constant StdConvs.MetersPerMile.
MetersPerNauticalMiles (see page 20901)	This is constant StdConvs.MetersPerNauticalMiles.
MetersPerPace (see page 20901)	This is constant StdConvs.MetersPerPace.
MetersPerParsec (see page 20901)	This is constant StdConvs.MetersPerParsec.
MetersPerPica (see page 20902)	This is constant StdConvs.MetersPerPica.
MetersPerPoint (see page 20902)	This is constant StdConvs.MetersPerPoint.
MetersPerRod (see page 20902)	This is constant StdConvs.MetersPerRod.
MetersPerYard (see page 20903)	This is constant StdConvs.MetersPerYard.
SquareMetersPerAcre (see page 20903)	This is constant StdConvs.SquareMetersPerAcre.
SquareMetersPerSquareFoot (see page 20903)	This is constant StdConvs.SquareMetersPerSquareFoot.
SquareMetersPerSquareInch (see page 20904)	This is constant StdConvs.SquareMetersPerSquareInch.
SquareMetersPerSquareMile (see page 20904)	This is constant StdConvs.SquareMetersPerSquareMile.
SquareMetersPerSquareRod (see page 20904)	This is constant StdConvs.SquareMetersPerSquareRod.
SquareMetersPerSquareYard (see page 20905)	This is constant StdConvs.SquareMetersPerSquareYard.

Functions

	Name	Description
	CelsiusToFahrenheit (see page 20832)	Converts a temperature expressed in degrees Celsius to the corresponding temperature in degrees Fahrenheit.
	FahrenheitToCelsius (see page 20832)	Converts a temperature expressed in degrees Fahrenheit to the corresponding temperature in degrees Celsius.
	InitStdConvs (see page 20833)	Initialize conversion variables for C++ program.

Variables

Name	Description
auAcres (see page 20836)	This is variable StdConvs.auAcres.
auAres (see page 20837)	This is variable StdConvs.auAres.
auCentares (see page 20837)	This is variable StdConvs.auCentares.
auHectares (see page 20837)	This is variable StdConvs.auHectares.
auSquareCentimeters (see page 20838)	This is variable StdConvs.auSquareCentimeters.
auSquareDecameters (see page 20838)	This is variable StdConvs.auSquareDecameters.

auSquareDecimeters (see page 20838)	This is variable StdConvs.auSquareDecimeters.
auSquareFeet (see page 20839)	This is variable StdConvs.auSquareFeet.
auSquareHectometers (see page 20839)	This is variable StdConvs.auSquareHectometers.
auSquareInches (see page 20839)	This is variable StdConvs.auSquareInches.
auSquareKilometers (see page 20840)	This is variable StdConvs.auSquareKilometers.
auSquareMeters (see page 20840)	This is variable StdConvs.auSquareMeters.
auSquareMiles (see page 20840)	This is variable StdConvs.auSquareMiles.
auSquareMillimeters (see page 20841)	This is variable StdConvs.auSquareMillimeters.
auSquareRods (see page 20841)	This is variable StdConvs.auSquareRods.
auSquareYards (see page 20841)	This is variable StdConvs.auSquareYards.
cbArea (see page 20842)	This is variable StdConvs.cbArea.
cbDistance (see page 20842)	This is variable StdConvs.cbDistance.
cbMass (see page 20842)	This is variable StdConvs.cbMass.
cbTemperature (see page 20843)	This is variable StdConvs.cbTemperature.
cbTime (see page 20843)	This is variable StdConvs.cbTime.
cbVolume (see page 20843)	This is variable StdConvs.cbVolume.
duAngstroms (see page 20844)	This is variable StdConvs.duAngstroms.
duAstronomicalUnits (see page 20844)	This is variable StdConvs.duAstronomicalUnits.
duCentimeters (see page 20844)	This is variable StdConvs.duCentimeters.
duChains (see page 20845)	This is variable StdConvs.duChains.
duCubits (see page 20845)	This is variable StdConvs.duCubits.
duDecameters (see page 20845)	This is variable StdConvs.duDecameters.
duDecimeters (see page 20846)	This is variable StdConvs.duDecimeters.
duFathoms (see page 20846)	This is variable StdConvs.duFathoms.
duFeet (see page 20846)	This is variable StdConvs.duFeet.
duFurlongs (see page 20847)	This is variable StdConvs.duFurlongs.
duGigameters (see page 20847)	This is variable StdConvs.duGigameters.
duHands (see page 20847)	This is variable StdConvs.duHands.
duHectometers (see page 20848)	This is variable StdConvs.duHectometers.
duInches (see page 20848)	This is variable StdConvs.duInches.
duKilometers (see page 20848)	This is variable StdConvs.duKilometers.
duLightYears (see page 20849)	This is variable StdConvs.duLightYears.
duLinks (see page 20849)	This is variable StdConvs.duLinks.
duMegameters (see page 20849)	This is variable StdConvs.duMegameters.
duMeters (see page 20850)	This is variable StdConvs.duMeters.
duMicromicrons (see page 20850)	This is variable StdConvs.duMicromicrons.
duMicrons (see page 20850)	This is variable StdConvs.duMicrons.
duMiles (see page 20851)	This is variable StdConvs.duMiles.
duMillimeters (see page 20851)	This is variable StdConvs.duMillimeters.
duMillimicrons (see page 20851)	This is variable StdConvs.duMillimicrons.
duNauticalMiles (see page 20852)	This is variable StdConvs.duNauticalMiles.
duPaces (see page 20852)	This is variable StdConvs.duPaces.
duParsecs (see page 20852)	This is variable StdConvs.duParsecs.
duPicas (see page 20853)	This is variable StdConvs.duPicas.
duPoints (see page 20853)	This is variable StdConvs.duPoints.
duRods (see page 20853)	This is variable StdConvs.duRods.
duYards (see page 20854)	This is variable StdConvs.duYards.
muCentigrams (see page 20854)	This is variable StdConvs.muCentigrams.
muDecagrams (see page 20854)	This is variable StdConvs.muDecagrams.
muDecigrams (see page 20855)	This is variable StdConvs.muDecigrams.
muDrams (see page 20855)	This is variable StdConvs.muDrams.
muGrains (see page 20855)	This is variable StdConvs.muGrains.
muGrams (see page 20856)	This is variable StdConvs.muGrams.
muHectograms (see page 20856)	This is variable StdConvs.muHectograms.

muKilograms (see page 20856)	This is variable StdConvs.muKilograms.
muLongTons (see page 20857)	This is variable StdConvs.muLongTons.
muMetricTons (see page 20857)	This is variable StdConvs.muMetricTons.
muMicrograms (see page 20857)	This is variable StdConvs.muMicrograms.
muMilligrams (see page 20858)	This is variable StdConvs.muMilligrams.
muNanograms (see page 20858)	This is variable StdConvs.muNanograms.
muOunces (see page 20858)	This is variable StdConvs.muOunces.
muPounds (see page 20859)	This is variable StdConvs.muPounds.
muStones (see page 20859)	This is variable StdConvs.muStones.
muTons (see page 20859)	This is variable StdConvs.muTons.
tuCelsius (see page 20860)	This is variable StdConvs.tuCelsius.
tuCenturies (see page 20860)	This is variable StdConvs.tuCenturies.
tuDateTime (see page 20860)	This is variable StdConvs.tuDateTime.
tuDays (see page 20861)	This is variable StdConvs.tuDays.
tuDecades (see page 20861)	This is variable StdConvs.tuDecades.
tuFahrenheit (see page 20861)	This is variable StdConvs.tuFahrenheit.
tuFortnights (see page 20862)	This is variable StdConvs.tuFortnights.
tuHours (see page 20862)	This is variable StdConvs.tuHours.
tuJulianDate (see page 20862)	This is variable StdConvs.tuJulianDate.
tuKelvin (see page 20863)	This is variable StdConvs.tuKelvin.
tuMillennia (see page 20863)	This is variable StdConvs.tuMillennia.
tuMilliSeconds (see page 20863)	This is variable StdConvs.tuMilliSeconds.
tuMinutes (see page 20864)	This is variable StdConvs.tuMinutes.
tuModifiedJulianDate (see page 20864)	This is variable StdConvs.tuModifiedJulianDate.
tuMonths (see page 20864)	This is variable StdConvs.tuMonths.
tuRankine (see page 20865)	This is variable StdConvs.tuRankine.
tuReaumur (see page 20865)	This is variable StdConvs.tuReaumur.
tuSeconds (see page 20865)	This is variable StdConvs.tuSeconds.
tuWeeks (see page 20866)	This is variable StdConvs.tuWeeks.
tuYears (see page 20866)	This is variable StdConvs.tuYears.
vuAcreFeet (see page 20866)	This is variable StdConvs.vuAcreFeet.
vuAcreInches (see page 20867)	This is variable StdConvs.vuAcreInches.
vuCentiLiters (see page 20867)	This is variable StdConvs.vuCentiLiters.
vuCordFeet (see page 20867)	This is variable StdConvs.vuCordFeet.
vuCords (see page 20868)	This is variable StdConvs.vuCords.
vuCubicCentimeters (see page 20868)	This is variable StdConvs.vuCubicCentimeters.
vuCubicDecameters (see page 20868)	This is variable StdConvs.vuCubicDecameters.
vuCubicDecimeters (see page 20869)	This is variable StdConvs.vuCubicDecimeters.
vuCubicFeet (see page 20869)	This is variable StdConvs.vuCubicFeet.
vuCubicHectometers (see page 20869)	This is variable StdConvs.vuCubicHectometers.
vuCubicInches (see page 20870)	This is variable StdConvs.vuCubicInches.
vuCubicKilometers (see page 20870)	This is variable StdConvs.vuCubicKilometers.
vuCubicMeters (see page 20870)	This is variable StdConvs.vuCubicMeters.
vuCubicMiles (see page 20871)	This is variable StdConvs.vuCubicMiles.
vuCubicMillimeters (see page 20871)	This is variable StdConvs.vuCubicMillimeters.
vuCubicYards (see page 20871)	This is variable StdConvs.vuCubicYards.
vuDecaLiters (see page 20872)	This is variable StdConvs.vuDecaLiters.
vuDecasteres (see page 20872)	This is variable StdConvs.vuDecasteres.
vuDeciLiters (see page 20872)	This is variable StdConvs.vuDeciLiters.
vuDecisteres (see page 20873)	This is variable StdConvs.vuDecisteres.
vuDryBuckets (see page 20873)	This is variable StdConvs.vuDryBuckets.
vuDryBushels (see page 20873)	This is variable StdConvs.vuDryBushels.
vuDryGallons (see page 20874)	This is variable StdConvs.vuDryGallons.
vuDryPecks (see page 20874)	This is variable StdConvs.vuDryPecks.

vuDryPints (see page 20874)	This is variable StdConvs.vuDryPints.
vuDryQuarts (see page 20875)	This is variable StdConvs.vuDryQuarts.
vuFluidCups (see page 20875)	This is variable StdConvs.vuFluidCups.
vuFluidGallons (see page 20875)	This is variable StdConvs.vuFluidGallons.
vuFluidGills (see page 20876)	This is variable StdConvs.vuFluidGills.
vuFluidOunces (see page 20876)	This is variable StdConvs.vuFluidOunces.
vuFluidPints (see page 20876)	This is variable StdConvs.vuFluidPints.
vuFluidQuarts (see page 20877)	This is variable StdConvs.vuFluidQuarts.
vuFluidTablespoons (see page 20877)	This is variable StdConvs.vuFluidTablespoons.
vuFluidTeaspoons (see page 20877)	This is variable StdConvs.vuFluidTeaspoons.
vuHectoLiters (see page 20878)	This is variable StdConvs.vuHectoLiters.
vuKiloLiters (see page 20878)	This is variable StdConvs.vuKiloLiters.
vuLiters (see page 20878)	This is variable StdConvs.vuLiters.
vuMilliLiters (see page 20879)	This is variable StdConvs.vuMilliLiters.
vuSteres (see page 20879)	This is variable StdConvs.vuSteres.
vuUKBuckets (see page 20879)	This is variable StdConvs.vuUKBuckets.
vuUKBushels (see page 20880)	This is variable StdConvs.vuUKBushels.
vuUKGallons (see page 20880)	This is variable StdConvs.vuUKGallons.
vuUKGills (see page 20880)	This is variable StdConvs.vuUKGills.
vuUKOunces (see page 20881)	This is variable StdConvs.vuUKOunces.
vuUKPecks (see page 20881)	This is variable StdConvs.vuUKPecks.
vuUKPints (see page 20881)	This is variable StdConvs.vuUKPints.
vuUKPottles (see page 20882)	This is variable StdConvs.vuUKPottles.
vuUKQuarts (see page 20882)	This is variable StdConvs.vuUKQuarts.

153.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CelsiusToFahrenheit (see page 20832)	Converts a temperature expressed in degrees Celsius to the corresponding temperature in degrees Fahrenheit.
	FahrenheitToCelsius (see page 20832)	Converts a temperature expressed in degrees Fahrenheit to the corresponding temperature in degrees Celsius.
	InitStdConvs (see page 20833)	Initialize conversion variables for C++ program.

153.1.1 StdConvs.CelsiusToFahrenheit Function

Converts a temperature expressed in degrees Celsius to the corresponding temperature in degrees Fahrenheit.

Pascal

```
function CelsiusToFahrenheit(const AValue: Double): Double;
```

C++

```
Double CelsiusToFahrenheit(const Double AValue);
```

File

StdConvs

Description

Call CelsiusToFahrenheit to convert the temperature specified by AValue, which is in degrees Celsius, to the corresponding temperature in degrees Fahrenheit.

See Also

FahrenheitToCelsius (see page 20832)

Convert

153.1.2 StdConvs.FahrenheitToCelsius Function

Converts a temperature expressed in degrees Fahrenheit to the corresponding temperature in degrees Celsius.

Pascal

```
function FahrenheitToCelsius(const AValue: Double): Double;
```

C++

```
Double FahrenheitToCelsius(const Double AValue);
```

File

StdConvs

Description

Call FahrenheitToCelsius to convert the temperature specified by AValue, which is in degrees Fahrenheit, to the corresponding temperature in degrees Celsius.

See Also

CelsiusToFahrenheit (see page 20832)

Convert

153.1.3 StdConvs.InitStdConvs Function

Initialize conversion variables for C++ program.

Pascal

```
procedure InitStdConvs;
```

C++

```
InitStdConvs();
```

File

StdConvs

Description

Insert InitStdConvs into a C++ program to ensure that StdConvs (see page 20827) global variables are properly initialized. InitStdConvs does nothing and does not need to be called. A reference to InitStdConvs in a C++ program guarantees that the linker will provide a call to the unit's initialization code, even if no other StdConvs (see page 20827) routines are present. This step is not necessary in Delphi code, which always calls the initialization code for referenced units.

See Also

FahrenheitToCelsius (see page 20832)

Convert

153.2 Variables

The following table lists variables in this documentation.

Variables

Name	Description
auAcres (see page 20836)	This is variable StdConvs.auAcres.
auAres (see page 20837)	This is variable StdConvs.auAres.
auCentares (see page 20837)	This is variable StdConvs.auCentares.
auHectares (see page 20837)	This is variable StdConvs.auHectares.
auSquareCentimeters (see page 20838)	This is variable StdConvs.auSquareCentimeters.
auSquareDecameters (see page 20838)	This is variable StdConvs.auSquareDecameters.
auSquareDecimeters (see page 20838)	This is variable StdConvs.auSquareDecimeters.
auSquareFeet (see page 20839)	This is variable StdConvs.auSquareFeet.
auSquareHectometers (see page 20839)	This is variable StdConvs.auSquareHectometers.
auSquareInches (see page 20839)	This is variable StdConvs.auSquareInches.
auSquareKilometers (see page 20840)	This is variable StdConvs.auSquareKilometers.
auSquareMeters (see page 20840)	This is variable StdConvs.auSquareMeters.
auSquareMiles (see page 20840)	This is variable StdConvs.auSquareMiles.
auSquareMillimeters (see page 20841)	This is variable StdConvs.auSquareMillimeters.
auSquareRods (see page 20841)	This is variable StdConvs.auSquareRods.
auSquareYards (see page 20841)	This is variable StdConvs.auSquareYards.
cbArea (see page 20842)	This is variable StdConvs.cbArea.
cbDistance (see page 20842)	This is variable StdConvs.cbDistance.
cbMass (see page 20842)	This is variable StdConvs.cbMass.
cbTemperature (see page 20843)	This is variable StdConvs.cbTemperature.
cbTime (see page 20843)	This is variable StdConvs.cbTime.
cbVolume (see page 20843)	This is variable StdConvs.cbVolume.
duAngstroms (see page 20844)	This is variable StdConvs.duAngstroms.
duAstronomicalUnits (see page 20844)	This is variable StdConvs.duAstronomicalUnits.
duCentimeters (see page 20844)	This is variable StdConvs.duCentimeters.
duChains (see page 20845)	This is variable StdConvs.duChains.
duCubits (see page 20845)	This is variable StdConvs.duCubits.
duDecameters (see page 20845)	This is variable StdConvs.duDecameters.
duDecimeters (see page 20846)	This is variable StdConvs.duDecimeters.
duFathoms (see page 20846)	This is variable StdConvs.duFathoms.
duFeet (see page 20846)	This is variable StdConvs.duFeet.
duFurlongs (see page 20847)	This is variable StdConvs.duFurlongs.
duGigameters (see page 20847)	This is variable StdConvs.duGigameters.
duHands (see page 20847)	This is variable StdConvs.duHands.
duHectometers (see page 20848)	This is variable StdConvs.duHectometers.
duInches (see page 20848)	This is variable StdConvs.duInches.
duKilometers (see page 20848)	This is variable StdConvs.duKilometers.
duLightYears (see page 20849)	This is variable StdConvs.duLightYears.
duLinks (see page 20849)	This is variable StdConvs.duLinks.
duMegameters (see page 20849)	This is variable StdConvs.duMegameters.
duMeters (see page 20850)	This is variable StdConvs.duMeters.
duMicromicrons (see page 20850)	This is variable StdConvs.duMicromicrons.
duMicrons (see page 20850)	This is variable StdConvs.duMicrons.
duMiles (see page 20851)	This is variable StdConvs.duMiles.

duMillimeters (see page 20851)	This is variable StdConvs.duMillimeters.
duMillimicrons (see page 20851)	This is variable StdConvs.duMillimicrons.
duNauticalMiles (see page 20852)	This is variable StdConvs.duNauticalMiles.
duPaces (see page 20852)	This is variable StdConvs.duPaces.
duParsecs (see page 20852)	This is variable StdConvs.duParsecs.
duPicas (see page 20853)	This is variable StdConvs.duPicas.
duPoints (see page 20853)	This is variable StdConvs.duPoints.
duRods (see page 20853)	This is variable StdConvs.duRods.
duYards (see page 20854)	This is variable StdConvs.duYards.
muCentigrams (see page 20854)	This is variable StdConvs.muCentigrams.
muDecagrams (see page 20854)	This is variable StdConvs.muDecagrams.
muDecigrams (see page 20855)	This is variable StdConvs.muDecigrams.
muDrams (see page 20855)	This is variable StdConvs.muDrams.
muGrains (see page 20855)	This is variable StdConvs.muGrains.
muGrams (see page 20856)	This is variable StdConvs.muGrams.
muHectograms (see page 20856)	This is variable StdConvs.muHectograms.
muKilograms (see page 20856)	This is variable StdConvs.muKilograms.
muLongTons (see page 20857)	This is variable StdConvs.muLongTons.
muMetricTons (see page 20857)	This is variable StdConvs.muMetricTons.
muMicrograms (see page 20857)	This is variable StdConvs.muMicrograms.
muMilligrams (see page 20858)	This is variable StdConvs.muMilligrams.
muNanograms (see page 20858)	This is variable StdConvs.muNanograms.
muOunces (see page 20858)	This is variable StdConvs.muOunces.
muPounds (see page 20859)	This is variable StdConvs.muPounds.
muStones (see page 20859)	This is variable StdConvs.muStones.
muTons (see page 20859)	This is variable StdConvs.muTons.
tuCelsius (see page 20860)	This is variable StdConvs.tuCelsius.
tuCenturies (see page 20860)	This is variable StdConvs.tuCenturies.
tuDateTime (see page 20860)	This is variable StdConvs.tuDateTime.
tuDays (see page 20861)	This is variable StdConvs.tuDays.
tuDecades (see page 20861)	This is variable StdConvs.tuDecades.
tuFahrenheit (see page 20861)	This is variable StdConvs.tuFahrenheit.
tuFortnights (see page 20862)	This is variable StdConvs.tuFortnights.
tuHours (see page 20862)	This is variable StdConvs.tuHours.
tuJulianDate (see page 20862)	This is variable StdConvs.tuJulianDate.
tuKelvin (see page 20863)	This is variable StdConvs.tuKelvin.
tuMillennia (see page 20863)	This is variable StdConvs.tuMillennia.
tuMilliSeconds (see page 20863)	This is variable StdConvs.tuMilliSeconds.
tuMinutes (see page 20864)	This is variable StdConvs.tuMinutes.
tuModifiedJulianDate (see page 20864)	This is variable StdConvs.tuModifiedJulianDate.
tuMonths (see page 20864)	This is variable StdConvs.tuMonths.
tuRankine (see page 20865)	This is variable StdConvs.tuRankine.
tuReaumur (see page 20865)	This is variable StdConvs.tuReaumur.
tuSeconds (see page 20865)	This is variable StdConvs.tuSeconds.
tuWeeks (see page 20866)	This is variable StdConvs.tuWeeks.
tuYears (see page 20866)	This is variable StdConvs.tuYears.
vuAcreFeet (see page 20866)	This is variable StdConvs.vuAcreFeet.
vuAcreInches (see page 20867)	This is variable StdConvs.vuAcreInches.
vuCentiLiters (see page 20867)	This is variable StdConvs.vuCentiLiters.
vuCordFeet (see page 20867)	This is variable StdConvs.vuCordFeet.
vuCords (see page 20868)	This is variable StdConvs.vuCords.
vuCubicCentimeters (see page 20868)	This is variable StdConvs.vuCubicCentimeters.
vuCubicDecameters (see page 20868)	This is variable StdConvs.vuCubicDecameters.
vuCubicDecimeters (see page 20869)	This is variable StdConvs.vuCubicDecimeters.

vuCubicFeet (see page 20869)	This is variable StdConvs.vuCubicFeet.
vuCubicHectometers (see page 20869)	This is variable StdConvs.vuCubicHectometers.
vuCubicInches (see page 20870)	This is variable StdConvs.vuCubicInches.
vuCubicKilometers (see page 20870)	This is variable StdConvs.vuCubicKilometers.
vuCubicMeters (see page 20870)	This is variable StdConvs.vuCubicMeters.
vuCubicMiles (see page 20871)	This is variable StdConvs.vuCubicMiles.
vuCubicMillimeters (see page 20871)	This is variable StdConvs.vuCubicMillimeters.
vuCubicYards (see page 20871)	This is variable StdConvs.vuCubicYards.
vuDecaLiters (see page 20872)	This is variable StdConvs.vuDecaLiters.
vuDecasteres (see page 20872)	This is variable StdConvs.vuDecasteres.
vuDeciLiters (see page 20872)	This is variable StdConvs.vuDeciLiters.
vuDecisteres (see page 20873)	This is variable StdConvs.vuDecisteres.
vuDryBuckets (see page 20873)	This is variable StdConvs.vuDryBuckets.
vuDryBushels (see page 20873)	This is variable StdConvs.vuDryBushels.
vuDryGallons (see page 20874)	This is variable StdConvs.vuDryGallons.
vuDryPecks (see page 20874)	This is variable StdConvs.vuDryPecks.
vuDryPints (see page 20874)	This is variable StdConvs.vuDryPints.
vuDryQuarts (see page 20875)	This is variable StdConvs.vuDryQuarts.
vuFluidCups (see page 20875)	This is variable StdConvs.vuFluidCups.
vuFluidGallons (see page 20875)	This is variable StdConvs.vuFluidGallons.
vuFluidGills (see page 20876)	This is variable StdConvs.vuFluidGills.
vuFluidOunces (see page 20876)	This is variable StdConvs.vuFluidOunces.
vuFluidPints (see page 20876)	This is variable StdConvs.vuFluidPints.
vuFluidQuarts (see page 20877)	This is variable StdConvs.vuFluidQuarts.
vuFluidTablespoons (see page 20877)	This is variable StdConvs.vuFluidTablespoons.
vuFluidTeaspoons (see page 20877)	This is variable StdConvs.vuFluidTeaspoons.
vuHectoLiters (see page 20878)	This is variable StdConvs.vuHectoLiters.
vuKiloLiters (see page 20878)	This is variable StdConvs.vuKiloLiters.
vuLiters (see page 20878)	This is variable StdConvs.vuLiters.
vuMilliLiters (see page 20879)	This is variable StdConvs.vuMilliLiters.
vuSteres (see page 20879)	This is variable StdConvs.vuSteres.
vuUKBuckets (see page 20879)	This is variable StdConvs.vuUKBuckets.
vuUKBushels (see page 20880)	This is variable StdConvs.vuUKBushels.
vuUKGallons (see page 20880)	This is variable StdConvs.vuUKGallons.
vuUKGills (see page 20880)	This is variable StdConvs.vuUKGills.
vuUKOunces (see page 20881)	This is variable StdConvs.vuUKOunces.
vuUKPecks (see page 20881)	This is variable StdConvs.vuUKPecks.
vuUKPints (see page 20881)	This is variable StdConvs.vuUKPints.
vuUKPottles (see page 20882)	This is variable StdConvs.vuUKPottles.
vuUKQuarts (see page 20882)	This is variable StdConvs.vuUKQuarts.

153.2.1 StdConvs.auAcres Variable

Pascal

```
auAcres: TConvType;
```

C++

```
TConvType auAcres;
```

File

StdConvs

Description

This is variable StdConvs.auAcres.

153.2.2 StdConvs.auAres Variable

Pascal

```
auAres: TConvType;
```

C++

```
TConvType auAres;
```

File

StdConvs

Description

This is variable StdConvs.auAres.

153.2.3 StdConvs.auCentares Variable

Pascal

```
auCentares: TConvType;
```

C++

```
TConvType auCentares;
```

File

StdConvs

Description

This is variable StdConvs.auCentares.

153.2.4 StdConvs.auHectares Variable

Pascal

```
auHectares: TConvType;
```

C++

```
TConvType auHectares;
```

File

StdConvs

Description

This is variable StdConvs.auHectares.

153.2.5 StdConvs.auSquareCentimeters Variable

Pascal

```
auSquareCentimeters: TConvType;
```

C++

```
TConvType auSquareCentimeters;
```

File

StdConvs

Description

This is variable StdConvs.auSquareCentimeters.

153.2.6 StdConvs.auSquareDecameters Variable

Pascal

```
auSquareDecameters: TConvType;
```

C++

```
TConvType auSquareDecameters;
```

File

StdConvs

Description

This is variable StdConvs.auSquareDecameters.

153.2.7 StdConvs.auSquareDecimeters Variable

Pascal

```
auSquareDecimeters: TConvType;
```

C++

```
TConvType auSquareDecimeters;
```

File

StdConvs

Description

This is variable StdConvs.auSquareDecimeters.

153.2.8 StdConvs.auSquareFeet Variable

Pascal

```
auSquareFeet: TConvType;
```

C++

```
TConvType auSquareFeet;
```

File

StdConvs

Description

This is variable StdConvs.auSquareFeet.

153.2.9 StdConvs.auSquareHectometers Variable

Pascal

```
auSquareHectometers: TConvType;
```

C++

```
TConvType auSquareHectometers;
```

File

StdConvs

Description

This is variable StdConvs.auSquareHectometers.

153.2.10 StdConvs.auSquareInches Variable

Pascal

```
auSquareInches: TConvType;
```

C++

```
TConvType auSquareInches;
```

File

StdConvs

Description

This is variable StdConvs.auSquareInches.

153.2.11 StdConvs.auSquareKilometers Variable

Pascal

```
auSquareKilometers: TConvType;
```

C++

```
TConvType auSquareKilometers;
```

File

StdConvs

Description

This is variable StdConvs.auSquareKilometers.

153.2.12 StdConvs.auSquareMeters Variable

Pascal

```
auSquareMeters: TConvType;
```

C++

```
TConvType auSquareMeters;
```

File

StdConvs

Description

This is variable StdConvs.auSquareMeters.

153.2.13 StdConvs.auSquareMiles Variable

Pascal

```
auSquareMiles: TConvType;
```

C++

```
TConvType auSquareMiles;
```

File

StdConvs

Description

This is variable StdConvs.auSquareMiles.

153.2.14 StdConvs.auSquareMillimeters Variable

Pascal

```
auSquareMillimeters: TConvType;
```

C++

```
TConvType auSquareMillimeters;
```

File

StdConvs

Description

This is variable StdConvs.auSquareMillimeters.

153.2.15 StdConvs.auSquareRods Variable

Pascal

```
auSquareRods: TConvType;
```

C++

```
TConvType auSquareRods;
```

File

StdConvs

Description

This is variable StdConvs.auSquareRods.

153.2.16 StdConvs.auSquareYards Variable

Pascal

```
auSquareYards: TConvType;
```

C++

```
TConvType auSquareYards;
```

File

StdConvs

Description

This is variable StdConvs.auSquareYards.

153.2.17 StdConvs.cbArea Variable

Pascal

```
cbArea: TConvFamily;
```

C++

```
TConvFamily cbArea;
```

File

StdConvs

Description

This is variable StdConvs.cbArea.

153.2.18 StdConvs.cbDistance Variable

Pascal

```
cbDistance: TConvFamily;
```

C++

```
TConvFamily cbDistance;
```

File

StdConvs

Description

This is variable StdConvs.cbDistance.

153.2.19 StdConvs.cbMass Variable

Pascal

```
cbMass: TConvFamily;
```

C++

```
TConvFamily cbMass;
```

File

StdConvs

Description

This is variable StdConvs.cbMass.

153.2.20 StdConvs.cbTemperature Variable

Pascal

```
cbTemperature: TConvFamily;
```

C++

```
TConvFamily cbTemperature;
```

File

StdConvs

Description

This is variable StdConvs.cbTemperature.

153.2.21 StdConvs.cbTime Variable

Pascal

```
cbTime: TConvFamily;
```

C++

```
TConvFamily cbTime;
```

File

StdConvs

Description

This is variable StdConvs.cbTime.

153.2.22 StdConvs.cbVolume Variable

Pascal

```
cbVolume: TConvFamily;
```

C++

```
TConvFamily cbVolume;
```

File

StdConvs

Description

This is variable StdConvs.cbVolume.

153.2.23 StdConvs.duAngstroms Variable

Pascal

```
duAngstroms: TConvType;
```

C++

```
TConvType duAngstroms;
```

File

StdConvs

Description

This is variable StdConvs.duAngstroms.

153.2.24 StdConvs.duAstronomicalUnits Variable

Pascal

```
duAstronomicalUnits: TConvType;
```

C++

```
TConvType duAstronomicalUnits;
```

File

StdConvs

Description

This is variable StdConvs.duAstronomicalUnits.

153.2.25 StdConvs.duCentimeters Variable

Pascal

```
duCentimeters: TConvType;
```

C++

```
TConvType duCentimeters;
```

File

StdConvs

Description

This is variable StdConvs.duCentimeters.

153.2.26 StdConvs.duChains Variable

Pascal

```
duChains: TConvType;
```

C++

```
TConvType duChains;
```

File

StdConvs

Description

This is variable StdConvs.duChains.

153.2.27 StdConvs.duCubits Variable

Pascal

```
duCubits: TConvType;
```

C++

```
TConvType duCubits;
```

File

StdConvs

Description

This is variable StdConvs.duCubits.

153.2.28 StdConvs.duDecameters Variable

Pascal

```
duDecameters: TConvType;
```

C++

```
TConvType duDecameters;
```

File

StdConvs

Description

This is variable StdConvs.duDecameters.

153.2.29 StdConvs.duDecimeters Variable

Pascal

```
duDecimeters: TConvType;
```

C++

```
TConvType duDecimeters;
```

File

StdConvs

Description

This is variable StdConvs.duDecimeters.

153.2.30 StdConvs.duFathoms Variable

Pascal

```
duFathoms: TConvType;
```

C++

```
TConvType duFathoms;
```

File

StdConvs

Description

This is variable StdConvs.duFathoms.

153.2.31 StdConvs.duFeet Variable

Pascal

```
duFeet: TConvType;
```

C++

```
TConvType duFeet;
```

File

StdConvs

Description

This is variable StdConvs.duFeet.

153.2.32 StdConvs.duFurlongs Variable

Pascal

```
duFurlongs: TConvType;
```

C++

```
TConvType duFurlongs;
```

File

StdConvs

Description

This is variable StdConvs.duFurlongs.

153.2.33 StdConvs.duGigameters Variable

Pascal

```
duGigameters: TConvType;
```

C++

```
TConvType duGigameters;
```

File

StdConvs

Description

This is variable StdConvs.duGigameters.

153.2.34 StdConvs.duHands Variable

Pascal

```
duHands: TConvType;
```

C++

```
TConvType duHands;
```

File

StdConvs

Description

This is variable StdConvs.duHands.

153.2.35 StdConvs.duHectometers Variable

Pascal

```
duHectometers: TConvType;
```

C++

```
TConvType duHectometers;
```

File

StdConvs

Description

This is variable StdConvs.duHectometers.

153.2.36 StdConvs.duInches Variable

Pascal

```
duInches: TConvType;
```

C++

```
TConvType duInches;
```

File

StdConvs

Description

This is variable StdConvs.duInches.

153.2.37 StdConvs.duKilometers Variable

Pascal

```
duKilometers: TConvType;
```

C++

```
TConvType duKilometers;
```

File

StdConvs

Description

This is variable StdConvs.duKilometers.

153.2.38 StdConvs.duLightYears Variable

Pascal

```
duLightYears: TConvType;
```

C++

```
TConvType duLightYears;
```

File

StdConvs

Description

This is variable StdConvs.duLightYears.

153.2.39 StdConvs.duLinks Variable

Pascal

```
duLinks: TConvType;
```

C++

```
TConvType duLinks;
```

File

StdConvs

Description

This is variable StdConvs.duLinks.

153.2.40 StdConvs.duMegameters Variable

Pascal

```
duMegameters: TConvType;
```

C++

```
TConvType duMegameters;
```

File

StdConvs

Description

This is variable StdConvs.duMegameters.

153.2.41 StdConvs.duMeters Variable

Pascal

```
duMeters: TConvType;
```

C++

```
TConvType duMeters;
```

File

StdConvs

Description

This is variable StdConvs.duMeters.

153.2.42 StdConvs.duMicromicrons Variable

Pascal

```
duMicromicrons: TConvType;
```

C++

```
TConvType duMicromicrons;
```

File

StdConvs

Description

This is variable StdConvs.duMicromicrons.

153.2.43 StdConvs.duMicrons Variable

Pascal

```
duMicrons: TConvType;
```

C++

```
TConvType duMicrons;
```

File

StdConvs

Description

This is variable StdConvs.duMicrons.

153.2.44 StdConvs.duMiles Variable

Pascal

```
duMiles: TConvType;
```

C++

```
TConvType duMiles;
```

File

StdConvs

Description

This is variable StdConvs.duMiles.

153.2.45 StdConvs.duMillimeters Variable

Pascal

```
duMillimeters: TConvType;
```

C++

```
TConvType duMillimeters;
```

File

StdConvs

Description

This is variable StdConvs.duMillimeters.

153.2.46 StdConvs.duMillimicrons Variable

Pascal

```
duMillimicrons: TConvType;
```

C++

```
TConvType duMillimicrons;
```

File

StdConvs

Description

This is variable StdConvs.duMillimicrons.

153.2.47 StdConvs.duNauticalMiles Variable

Pascal

```
duNauticalMiles: TConvType;
```

C++

```
TConvType duNauticalMiles;
```

File

StdConvs

Description

This is variable StdConvs.duNauticalMiles.

153.2.48 StdConvs.duPaces Variable

Pascal

```
duPaces: TConvType;
```

C++

```
TConvType duPaces;
```

File

StdConvs

Description

This is variable StdConvs.duPaces.

153.2.49 StdConvs.duParsecs Variable

Pascal

```
duParsecs: TConvType;
```

C++

```
TConvType duParsecs;
```

File

StdConvs

Description

This is variable StdConvs.duParsecs.

153.2.50 StdConvs.duPicas Variable

Pascal

```
duPicas: TConvType;
```

C++

```
TConvType duPicas;
```

File

StdConvs

Description

This is variable StdConvs.duPicas.

153.2.51 StdConvs.duPoints Variable

Pascal

```
duPoints: TConvType;
```

C++

```
TConvType duPoints;
```

File

StdConvs

Description

This is variable StdConvs.duPoints.

153.2.52 StdConvs.duRods Variable

Pascal

```
duRods: TConvType;
```

C++

```
TConvType duRods;
```

File

StdConvs

Description

This is variable StdConvs.duRods.

153.2.53 StdConvs.duYards Variable

Pascal

```
duYards: TConvType;
```

C++

```
TConvType duYards;
```

File

StdConvs

Description

This is variable StdConvs.duYards.

153.2.54 StdConvs.muCentigrams Variable

Pascal

```
muCentigrams: TConvType;
```

C++

```
TConvType muCentigrams;
```

File

StdConvs

Description

This is variable StdConvs.muCentigrams.

153.2.55 StdConvs.muDecagrams Variable

Pascal

```
muDecagrams: TConvType;
```

C++

```
TConvType muDecagrams;
```

File

StdConvs

Description

This is variable StdConvs.muDecagrams.

153.2.56 StdConvs.muDecigrams Variable

Pascal

```
muDecigrams: TConvType;
```

C++

```
TConvType muDecigrams;
```

File

StdConvs

Description

This is variable StdConvs.muDecigrams.

153.2.57 StdConvs.muDrams Variable

Pascal

```
muDrams: TConvType;
```

C++

```
TConvType muDrams;
```

File

StdConvs

Description

This is variable StdConvs.muDrams.

153.2.58 StdConvs.muGrains Variable

Pascal

```
muGrains: TConvType;
```

C++

```
TConvType muGrains;
```

File

StdConvs

Description

This is variable StdConvs.muGrains.

153.2.59 StdConvs.muGrams Variable

Pascal

```
muGrams: TConvType;
```

C++

```
TConvType muGrams;
```

File

StdConvs

Description

This is variable StdConvs.muGrams.

153.2.60 StdConvs.muHectograms Variable

Pascal

```
muHectograms: TConvType;
```

C++

```
TConvType muHectograms;
```

File

StdConvs

Description

This is variable StdConvs.muHectograms.

153.2.61 StdConvs.muKilograms Variable

Pascal

```
muKilograms: TConvType;
```

C++

```
TConvType muKilograms;
```

File

StdConvs

Description

This is variable StdConvs.muKilograms.

153.2.62 StdConvs.muLongTons Variable

Pascal

```
muLongTons: TConvType;
```

C++

```
TConvType muLongTons;
```

File

StdConvs

Description

This is variable StdConvs.muLongTons.

153.2.63 StdConvs.muMetricTons Variable

Pascal

```
muMetricTons: TConvType;
```

C++

```
TConvType muMetricTons;
```

File

StdConvs

Description

This is variable StdConvs.muMetricTons.

153.2.64 StdConvs.muMicrograms Variable

Pascal

```
muMicrograms: TConvType;
```

C++

```
TConvType muMicrograms;
```

File

StdConvs

Description

This is variable StdConvs.muMicrograms.

153.2.65 StdConvs.muMilligrams Variable

Pascal

```
muMilligrams: TConvType;
```

C++

```
TConvType muMilligrams;
```

File

StdConvs

Description

This is variable StdConvs.muMilligrams.

153.2.66 StdConvs.muNanograms Variable

Pascal

```
muNanograms: TConvType;
```

C++

```
TConvType muNanograms;
```

File

StdConvs

Description

This is variable StdConvs.muNanograms.

153.2.67 StdConvs.muOunces Variable

Pascal

```
muOunces: TConvType;
```

C++

```
TConvType muOunces;
```

File

StdConvs

Description

This is variable StdConvs.muOunces.

153.2.68 StdConvs.muPounds Variable

Pascal

```
muPounds: TConvType;
```

C++

```
TConvType muPounds;
```

File

StdConvs

Description

This is variable StdConvs.muPounds.

153.2.69 StdConvs.muStones Variable

Pascal

```
muStones: TConvType;
```

C++

```
TConvType muStones;
```

File

StdConvs

Description

This is variable StdConvs.muStones.

153.2.70 StdConvs.muTons Variable

Pascal

```
muTons: TConvType;
```

C++

```
TConvType muTons;
```

File

StdConvs

Description

This is variable StdConvs.muTons.

153.2.71 StdConvs.tuCelsius Variable

Pascal

```
tuCelsius: TConvType;
```

C++

```
TConvType tuCelsius;
```

File

StdConvs

Description

This is variable StdConvs.tuCelsius.

153.2.72 StdConvs.tuCenturies Variable

Pascal

```
tuCenturies: TConvType;
```

C++

```
TConvType tuCenturies;
```

File

StdConvs

Description

This is variable StdConvs.tuCenturies.

153.2.73 StdConvs.tuDateTime Variable

Pascal

```
tuDateTime: TConvType;
```

C++

```
TConvType tuDateTime;
```

File

StdConvs

Description

This is variable StdConvs.tuDateTime.

153.2.74 StdConvs.tuDays Variable

Pascal

```
tuDays: TConvType;
```

C++

```
TConvType tuDays;
```

File

StdConvs

Description

This is variable StdConvs.tuDays.

153.2.75 StdConvs.tuDecades Variable

Pascal

```
tuDecades: TConvType;
```

C++

```
TConvType tuDecades;
```

File

StdConvs

Description

This is variable StdConvs.tuDecades.

153.2.76 StdConvs.tuFahrenheit Variable

Pascal

```
tuFahrenheit: TConvType;
```

C++

```
TConvType tuFahrenheit;
```

File

StdConvs

Description

This is variable StdConvs.tuFahrenheit.

153.2.77 StdConvs.tuFortnights Variable

Pascal

```
tuFortnights: TConvType;
```

C++

```
TConvType tuFortnights;
```

File

StdConvs

Description

This is variable StdConvs.tuFortnights.

153.2.78 StdConvs.tuHours Variable

Pascal

```
tuHours: TConvType;
```

C++

```
TConvType tuHours;
```

File

StdConvs

Description

This is variable StdConvs.tuHours.

153.2.79 StdConvs.tuJulianDate Variable

Pascal

```
tuJulianDate: TConvType;
```

C++

```
TConvType tuJulianDate;
```

File

StdConvs

Description

This is variable StdConvs.tuJulianDate.

153.2.80 StdConvs.tuKelvin Variable

Pascal

```
tuKelvin: TConvType;
```

C++

```
TConvType tuKelvin;
```

File

StdConvs

Description

This is variable StdConvs.tuKelvin.

153.2.81 StdConvs.tuMillennia Variable

Pascal

```
tuMillennia: TConvType;
```

C++

```
TConvType tuMillennia;
```

File

StdConvs

Description

This is variable StdConvs.tuMillennia.

153.2.82 StdConvs.tuMilliSeconds Variable

Pascal

```
tuMilliSeconds: TConvType;
```

C++

```
TConvType tuMilliSeconds;
```

File

StdConvs

Description

This is variable StdConvs.tuMilliSeconds.

153.2.83 StdConvs.tuMinutes Variable

Pascal

```
tuMinutes: TConvType;
```

C++

```
TConvType tuMinutes;
```

File

StdConvs

Description

This is variable StdConvs.tuMinutes.

153.2.84 StdConvs.tuModifiedJulianDate Variable

Pascal

```
tuModifiedJulianDate: TConvType;
```

C++

```
TConvType tuModifiedJulianDate;
```

File

StdConvs

Description

This is variable StdConvs.tuModifiedJulianDate.

153.2.85 StdConvs.tuMonths Variable

Pascal

```
tuMonths: TConvType;
```

C++

```
TConvType tuMonths;
```

File

StdConvs

Description

This is variable StdConvs.tuMonths.

153.2.86 StdConvs.tuRankine Variable

Pascal

```
tuRankine: TConvType;
```

C++

```
TConvType tuRankine;
```

File

StdConvs

Description

This is variable StdConvs.tuRankine.

153.2.87 StdConvs.tuReaumur Variable

Pascal

```
tuReaumur: TConvType;
```

C++

```
TConvType tuReaumur;
```

File

StdConvs

Description

This is variable StdConvs.tuReaumur.

153.2.88 StdConvs.tuSeconds Variable

Pascal

```
tuSeconds: TConvType;
```

C++

```
TConvType tuSeconds;
```

File

StdConvs

Description

This is variable StdConvs.tuSeconds.

153.2.89 StdConvs.tuWeeks Variable

Pascal

```
tuWeeks: TConvType;
```

C++

```
TConvType tuWeeks;
```

File

StdConvs

Description

This is variable StdConvs.tuWeeks.

153.2.90 StdConvs.tuYears Variable

Pascal

```
tuYears: TConvType;
```

C++

```
TConvType tuYears;
```

File

StdConvs

Description

This is variable StdConvs.tuYears.

153.2.91 StdConvs.vuAcreFeet Variable

Pascal

```
vuAcreFeet: TConvType;
```

C++

```
TConvType vuAcreFeet;
```

File

StdConvs

Description

This is variable StdConvs.vuAcreFeet.

153.2.92 StdConvs.vuAcreInches Variable

Pascal

```
vuAcreInches: TConvType;
```

C++

```
TConvType vuAcreInches;
```

File

StdConvs

Description

This is variable StdConvs.vuAcreInches.

153.2.93 StdConvs.vuCentiLiters Variable

Pascal

```
vuCentiLiters: TConvType;
```

C++

```
TConvType vuCentiLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuCentiLiters.

153.2.94 StdConvs.vuCordFeet Variable

Pascal

```
vuCordFeet: TConvType;
```

C++

```
TConvType vuCordFeet;
```

File

StdConvs

Description

This is variable StdConvs.vuCordFeet.

153.2.95 StdConvs.vuCords Variable

Pascal

```
vuCords: TConvType;
```

C++

```
TConvType vuCords;
```

File

StdConvs

Description

This is variable StdConvs.vuCords.

153.2.96 StdConvs.vuCubicCentimeters Variable

Pascal

```
vuCubicCentimeters: TConvType;
```

C++

```
TConvType vuCubicCentimeters;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicCentimeters.

153.2.97 StdConvs.vuCubicDecameters Variable

Pascal

```
vuCubicDecameters: TConvType;
```

C++

```
TConvType vuCubicDecameters;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicDecameters.

153.2.98 StdConvs.vuCubicDecimeters Variable

Pascal

```
vuCubicDecimeters: TConvType;
```

C++

```
TConvType vuCubicDecimeters;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicDecimeters.

153.2.99 StdConvs.vuCubicFeet Variable

Pascal

```
vuCubicFeet: TConvType;
```

C++

```
TConvType vuCubicFeet;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicFeet.

153.2.100 StdConvs.vuCubicHectometers Variable

Pascal

```
vuCubicHectometers: TConvType;
```

C++

```
TConvType vuCubicHectometers;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicHectometers.

153.2.101 StdConvs.vuCubicInches Variable

Pascal

```
vuCubicInches: TConvType;
```

C++

```
TConvType vuCubicInches;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicInches.

153.2.102 StdConvs.vuCubicKilometers Variable

Pascal

```
vuCubicKilometers: TConvType;
```

C++

```
TConvType vuCubicKilometers;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicKilometers.

153.2.103 StdConvs.vuCubicMeters Variable

Pascal

```
vuCubicMeters: TConvType;
```

C++

```
TConvType vuCubicMeters;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicMeters.

153.2.104 StdConvs.vuCubicMiles Variable

Pascal

```
vuCubicMiles: TConvType;
```

C++

```
TConvType vuCubicMiles;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicMiles.

153.2.105 StdConvs.vuCubicMillimeters Variable

Pascal

```
vuCubicMillimeters: TConvType;
```

C++

```
TConvType vuCubicMillimeters;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicMillimeters.

153.2.106 StdConvs.vuCubicYards Variable

Pascal

```
vuCubicYards: TConvType;
```

C++

```
TConvType vuCubicYards;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicYards.

153.2.107 StdConvs.vuDecaLiters Variable

Pascal

```
vuDecaLiters: TConvType;
```

C++

```
TConvType vuDecaLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuDecaLiters.

153.2.108 StdConvs.vuDecasteres Variable

Pascal

```
vuDecasteres: TConvType;
```

C++

```
TConvType vuDecasteres;
```

File

StdConvs

Description

This is variable StdConvs.vuDecasteres.

153.2.109 StdConvs.vuDeciLiters Variable

Pascal

```
vuDeciLiters: TConvType;
```

C++

```
TConvType vuDeciLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuDeciLiters.

153.2.110 StdConvs.vuDecisteres Variable

Pascal

```
vuDecisteres: TConvType;
```

C++

```
TConvType vuDecisteres;
```

File

StdConvs

Description

This is variable StdConvs.vuDecisteres.

153.2.111 StdConvs.vuDryBuckets Variable

Pascal

```
vuDryBuckets: TConvType;
```

C++

```
TConvType vuDryBuckets;
```

File

StdConvs

Description

This is variable StdConvs.vuDryBuckets.

153.2.112 StdConvs.vuDryBushels Variable

Pascal

```
vuDryBushels: TConvType;
```

C++

```
TConvType vuDryBushels;
```

File

StdConvs

Description

This is variable StdConvs.vuDryBushels.

153.2.113 StdConvs.vuDryGallons Variable

Pascal

```
vuDryGallons: TConvType;
```

C++

```
TConvType vuDryGallons;
```

File

StdConvs

Description

This is variable StdConvs.vuDryGallons.

153.2.114 StdConvs.vuDryPecks Variable

Pascal

```
vuDryPecks: TConvType;
```

C++

```
TConvType vuDryPecks;
```

File

StdConvs

Description

This is variable StdConvs.vuDryPecks.

153.2.115 StdConvs.vuDryPints Variable

Pascal

```
vuDryPints: TConvType;
```

C++

```
TConvType vuDryPints;
```

File

StdConvs

Description

This is variable StdConvs.vuDryPints.

153.2.116 StdConvs.vuDryQuarts Variable

Pascal

```
vuDryQuarts: TConvType;
```

C++

```
TConvType vuDryQuarts;
```

File

StdConvs

Description

This is variable StdConvs.vuDryQuarts.

153.2.117 StdConvs.vuFluidCups Variable

Pascal

```
vuFluidCups: TConvType;
```

C++

```
TConvType vuFluidCups;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidCups.

153.2.118 StdConvs.vuFluidGallons Variable

Pascal

```
vuFluidGallons: TConvType;
```

C++

```
TConvType vuFluidGallons;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidGallons.

153.2.119 StdConvs.vuFluidGills Variable

Pascal

```
vuFluidGills: TConvType;
```

C++

```
TConvType vuFluidGills;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidGills.

153.2.120 StdConvs.vuFluidOunces Variable

Pascal

```
vuFluidOunces: TConvType;
```

C++

```
TConvType vuFluidOunces;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidOunces.

153.2.121 StdConvs.vuFluidPints Variable

Pascal

```
vuFluidPints: TConvType;
```

C++

```
TConvType vuFluidPints;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidPints.

153.2.122 StdConvs.vuFluidQuarts Variable

Pascal

```
vuFluidQuarts: TConvType;
```

C++

```
TConvType vuFluidQuarts;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidQuarts.

153.2.123 StdConvs.vuFluidTablespoons Variable

Pascal

```
vuFluidTablespoons: TConvType;
```

C++

```
TConvType vuFluidTablespoons;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidTablespoons.

153.2.124 StdConvs.vuFluidTeaspoons Variable

Pascal

```
vuFluidTeaspoons: TConvType;
```

C++

```
TConvType vuFluidTeaspoons;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidTeaspoons.

153.2.125 StdConvs.vuHectoLiters Variable

Pascal

```
vuHectoLiters: TConvType;
```

C++

```
TConvType vuHectoLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuHectoLiters.

153.2.126 StdConvs.vuKiloLiters Variable

Pascal

```
vuKiloLiters: TConvType;
```

C++

```
TConvType vuKiloLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuKiloLiters.

153.2.127 StdConvs.vuLiters Variable

Pascal

```
vuLiters: TConvType;
```

C++

```
TConvType vuLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuLiters.

153.2.128 StdConvs.vuMilliLiters Variable

Pascal

```
vuMilliLiters: TConvType;
```

C++

```
TConvType vuMilliLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuMilliLiters.

153.2.129 StdConvs.vuSteres Variable

Pascal

```
vuSteres: TConvType;
```

C++

```
TConvType vuSteres;
```

File

StdConvs

Description

This is variable StdConvs.vuSteres.

153.2.130 StdConvs.vuUKBuckets Variable

Pascal

```
vuUKBuckets: TConvType;
```

C++

```
TConvType vuUKBuckets;
```

File

StdConvs

Description

This is variable StdConvs.vuUKBuckets.

153.2.131 StdConvs.vuUKBushels Variable

Pascal

```
vuUKBushels: TConvType;
```

C++

```
TConvType vuUKBushels;
```

File

StdConvs

Description

This is variable StdConvs.vuUKBushels.

153.2.132 StdConvs.vuUKGallons Variable

Pascal

```
vuUKGallons: TConvType;
```

C++

```
TConvType vuUKGallons;
```

File

StdConvs

Description

This is variable StdConvs.vuUKGallons.

153.2.133 StdConvs.vuUKGills Variable

Pascal

```
vuUKGills: TConvType;
```

C++

```
TConvType vuUKGills;
```

File

StdConvs

Description

This is variable StdConvs.vuUKGills.

153.2.134 StdConvs.vuUKOunces Variable

Pascal

```
vuUKOunces: TConvType;
```

C++

```
TConvType vuUKOunces;
```

File

StdConvs

Description

This is variable StdConvs.vuUKOunces.

153.2.135 StdConvs.vuUKPecks Variable

Pascal

```
vuUKPecks: TConvType;
```

C++

```
TConvType vuUKPecks;
```

File

StdConvs

Description

This is variable StdConvs.vuUKPecks.

153.2.136 StdConvs.vuUKPints Variable

Pascal

```
vuUKPints: TConvType;
```

C++

```
TConvType vuUKPints;
```

File

StdConvs

Description

This is variable StdConvs.vuUKPints.

153.2.137 StdConvs.vuUKPottles Variable

Pascal

```
vuUKPottles: TConvType;
```

C++

```
TConvType vuUKPottles;
```

File

StdConvs

Description

This is variable StdConvs.vuUKPottles.

153.2.138 StdConvs.vuUKQuarts Variable

Pascal

```
vuUKQuarts: TConvType;
```

C++

```
TConvType vuUKQuarts;
```

File

StdConvs

Description

This is variable StdConvs.vuUKQuarts.

153.3 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CubicMetersPerAcreFoot (see page 20884)	This is constant StdConvs.CubicMetersPerAcreFoot.
CubicMetersPerAcreInch (see page 20884)	This is constant StdConvs.CubicMetersPerAcreInch.
CubicMetersPerCord (see page 20885)	This is constant StdConvs.CubicMetersPerCord.
CubicMetersPerCordFoot (see page 20885)	This is constant StdConvs.CubicMetersPerCordFoot.
CubicMetersPerCubicFoot (see page 20885)	This is constant StdConvs.CubicMetersPerCubicFoot.
CubicMetersPerCubicInch (see page 20886)	This is constant StdConvs.CubicMetersPerCubicInch.
CubicMetersPerCubicMile (see page 20886)	This is constant StdConvs.CubicMetersPerCubicMile.
CubicMetersPerCubicYard (see page 20886)	This is constant StdConvs.CubicMetersPerCubicYard.
CubicMetersPerUKBucket (see page 20887)	This is constant StdConvs.CubicMetersPerUKBucket.
CubicMetersPerUKBushel (see page 20887)	This is constant StdConvs.CubicMetersPerUKBushel.
CubicMetersPerUKGallon (see page 20887)	This is constant StdConvs.CubicMetersPerUKGallon.
CubicMetersPerUKGill (see page 20888)	This is constant StdConvs.CubicMetersPerUKGill.
CubicMetersPerUKOunce (see page 20888)	This is constant StdConvs.CubicMetersPerUKOunce.
CubicMetersPerUKPeck (see page 20888)	This is constant StdConvs.CubicMetersPerUKPeck.
CubicMetersPerUKPint (see page 20889)	This is constant StdConvs.CubicMetersPerUKPint.
CubicMetersPerUKPottle (see page 20889)	This is constant StdConvs.CubicMetersPerUKPottle.
CubicMetersPerUKQuart (see page 20889)	This is constant StdConvs.CubicMetersPerUKQuart.
CubicMetersPerUSDryBucket (see page 20890)	This is constant StdConvs.CubicMetersPerUSDryBucket.
CubicMetersPerUSDryBushel (see page 20890)	This is constant StdConvs.CubicMetersPerUSDryBushel.
CubicMetersPerUSDryGallon (see page 20890)	This is constant StdConvs.CubicMetersPerUSDryGallon.
CubicMetersPerUSDryPeck (see page 20891)	This is constant StdConvs.CubicMetersPerUSDryPeck.
CubicMetersPerUSDryPint (see page 20891)	This is constant StdConvs.CubicMetersPerUSDryPint.
CubicMetersPerUSDryQuart (see page 20891)	This is constant StdConvs.CubicMetersPerUSDryQuart.
CubicMetersPerUSFluidCup (see page 20892)	This is constant StdConvs.CubicMetersPerUSFluidCup.
CubicMetersPerUSFluidGallon (see page 20892)	This is constant StdConvs.CubicMetersPerUSFluidGallon.
CubicMetersPerUSFluidGill (see page 20892)	This is constant StdConvs.CubicMetersPerUSFluidGill.
CubicMetersPerUSFluidOunce (see page 20893)	This is constant StdConvs.CubicMetersPerUSFluidOunce.
CubicMetersPerUSFluidPint (see page 20893)	This is constant StdConvs.CubicMetersPerUSFluidPint.
CubicMetersPerUSFluidQuart (see page 20893)	This is constant StdConvs.CubicMetersPerUSFluidQuart.
CubicMetersPerUSFluidTablespoon (see page 20894)	This is constant StdConvs.CubicMetersPerUSFluidTablespoon.
CubicMetersPerUSFluidTeaspoon (see page 20894)	This is constant StdConvs.CubicMetersPerUSFluidTeaspoon.
GramsPerDrams (see page 20894)	This is constant StdConvs.GramsPerDrams.
GramsPerGrains (see page 20895)	This is constant StdConvs.GramsPerGrains.
GramsPerLongTons (see page 20895)	This is constant StdConvs.GramsPerLongTons.
GramsPerOunces (see page 20895)	This is constant StdConvs.GramsPerOunces.
GramsPerPound (see page 20896)	This is constant StdConvs.GramsPerPound.
GramsPerStones (see page 20896)	This is constant StdConvs.GramsPerStones.
GramsPerTons (see page 20896)	This is constant StdConvs.GramsPerTons.
MetersPerAstronomicalUnit (see page 20897)	This is constant StdConvs.MetersPerAstronomicalUnit.
MetersPerChain (see page 20897)	This is constant StdConvs.MetersPerChain.
MetersPerCubit (see page 20897)	This is constant StdConvs.MetersPerCubit.
MetersPerFathom (see page 20898)	This is constant StdConvs.MetersPerFathom.
MetersPerFoot (see page 20898)	This is constant StdConvs.MetersPerFoot.
MetersPerFurlong (see page 20898)	This is constant StdConvs.MetersPerFurlong.

MetersPerHand (see page 20899)	This is constant StdConvs.MetersPerHand.
MetersPerInch (see page 20899)	This is constant StdConvs.MetersPerInch.
MetersPerLightSecond (see page 20899)	This is constant StdConvs.MetersPerLightSecond.
MetersPerLightYear (see page 20900)	This is constant StdConvs.MetersPerLightYear.
MetersPerLink (see page 20900)	This is constant StdConvs.MetersPerLink.
MetersPerMile (see page 20900)	This is constant StdConvs.MetersPerMile.
MetersPerNauticalMiles (see page 20901)	This is constant StdConvs.MetersPerNauticalMiles.
MetersPerPace (see page 20901)	This is constant StdConvs.MetersPerPace.
MetersPerParsec (see page 20901)	This is constant StdConvs.MetersPerParsec.
MetersPerPica (see page 20902)	This is constant StdConvs.MetersPerPica.
MetersPerPoint (see page 20902)	This is constant StdConvs.MetersPerPoint.
MetersPerRod (see page 20902)	This is constant StdConvs.MetersPerRod.
MetersPerYard (see page 20903)	This is constant StdConvs.MetersPerYard.
SquareMetersPerAcre (see page 20903)	This is constant StdConvs.SquareMetersPerAcre.
SquareMetersPerSquareFoot (see page 20903)	This is constant StdConvs.SquareMetersPerSquareFoot.
SquareMetersPerSquareInch (see page 20904)	This is constant StdConvs.SquareMetersPerSquareInch.
SquareMetersPerSquareMile (see page 20904)	This is constant StdConvs.SquareMetersPerSquareMile.
SquareMetersPerSquareRod (see page 20904)	This is constant StdConvs.SquareMetersPerSquareRod.
SquareMetersPerSquareYard (see page 20905)	This is constant StdConvs.SquareMetersPerSquareYard.

153.3.1 StdConvs.CubicMetersPerAcreFoot Constant

Pascal

```
CubicMetersPerAcreFoot = SquareMetersPerAcre * MetersPerFoot;
```

C++

```
CubicMetersPerAcreFoot = SquareMetersPerAcre * MetersPerFoot;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerAcreFoot.

153.3.2 StdConvs.CubicMetersPerAcreInch Constant

Pascal

```
CubicMetersPerAcreInch = SquareMetersPerAcre * MetersPerInch;
```

C++

```
CubicMetersPerAcreInch = SquareMetersPerAcre * MetersPerInch;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerAcreInch.

153.3.3 StdConvs.CubicMetersPerCord Constant

Pascal

```
CubicMetersPerCord = CubicMetersPerCubicFoot * 128;
```

C++

```
CubicMetersPerCord = CubicMetersPerCubicFoot * 128;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerCord.

153.3.4 StdConvs.CubicMetersPerCordFoot Constant

Pascal

```
CubicMetersPerCordFoot = CubicMetersPerCubicFoot * 16;
```

C++

```
CubicMetersPerCordFoot = CubicMetersPerCubicFoot * 16;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerCordFoot.

153.3.5 StdConvs.CubicMetersPerCubicFoot Constant

Pascal

```
CubicMetersPerCubicFoot = MetersPerFoot * MetersPerFoot * MetersPerFoot;
```

C++

```
CubicMetersPerCubicFoot = MetersPerFoot * MetersPerFoot * MetersPerFoot;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerCubicFoot.

153.3.6 StdConvs.CubicMetersPerCubicInch Constant

Pascal

```
CubicMetersPerCubicInch = MetersPerInch * MetersPerInch * MetersPerInch;
```

C++

```
CubicMetersPerCubicInch = MetersPerInch * MetersPerInch * MetersPerInch;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerCubicInch.

153.3.7 StdConvs.CubicMetersPerCubicMile Constant

Pascal

```
CubicMetersPerCubicMile = MetersPerMile * MetersPerMile * MetersPerMile;
```

C++

```
CubicMetersPerCubicMile = MetersPerMile * MetersPerMile * MetersPerMile;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerCubicMile.

153.3.8 StdConvs.CubicMetersPerCubicYard Constant

Pascal

```
CubicMetersPerCubicYard = MetersPerYard * MetersPerYard * MetersPerYard;
```

C++

```
CubicMetersPerCubicYard = MetersPerYard * MetersPerYard * MetersPerYard;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerCubicYard.

153.3.9 StdConvs.CubicMetersPerUKBucket Constant

Pascal

```
CubicMetersPerUKBucket = CubicMetersPerUKPeck * 2;
```

C++

```
CubicMetersPerUKBucket = CubicMetersPerUKPeck * 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKBucket.

153.3.10 StdConvs.CubicMetersPerUKBushel Constant

Pascal

```
CubicMetersPerUKBushel = CubicMetersPerUKBucket * 2;
```

C++

```
CubicMetersPerUKBushel = CubicMetersPerUKBucket * 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKBushel.

153.3.11 StdConvs.CubicMetersPerUKGallon Constant

Pascal

```
CubicMetersPerUKGallon = 0.00454609;
```

C++

```
CubicMetersPerUKGallon = 0.00454609;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKGallon.

153.3.12 StdConvs.CubicMetersPerUKGill Constant

Pascal

```
CubicMetersPerUKGill = CubicMetersPerUKPint / 4;
```

C++

```
CubicMetersPerUKGill = CubicMetersPerUKPint / 4;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKGill.

153.3.13 StdConvs.CubicMetersPerUKOunce Constant

Pascal

```
CubicMetersPerUKOunce = CubicMetersPerUKPint / 20;
```

C++

```
CubicMetersPerUKOunce = CubicMetersPerUKPint / 20;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKOunce.

153.3.14 StdConvs.CubicMetersPerUKPeck Constant

Pascal

```
CubicMetersPerUKPeck = CubicMetersPerUKGallon * 2;
```

C++

```
CubicMetersPerUKPeck = CubicMetersPerUKGallon * 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKPeck.

153.3.15 StdConvs.CubicMetersPerUKPint Constant

Pascal

```
CubicMetersPerUKPint = CubicMetersPerUKQuart / 2;
```

C++

```
CubicMetersPerUKPint = CubicMetersPerUKQuart / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKPint.

153.3.16 StdConvs.CubicMetersPerUKPottle Constant

Pascal

```
CubicMetersPerUKPottle = CubicMetersPerUKGallon / 2;
```

C++

```
CubicMetersPerUKPottle = CubicMetersPerUKGallon / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKPottle.

153.3.17 StdConvs.CubicMetersPerUKQuart Constant

Pascal

```
CubicMetersPerUKQuart = CubicMetersPerUKPottle / 2;
```

C++

```
CubicMetersPerUKQuart = CubicMetersPerUKPottle / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKQuart.

153.3.18 StdConvs.CubicMetersPerUSDryBucket Constant

Pascal

```
CubicMetersPerUSDryBucket = CubicMetersPerUSDryPeck * 2;
```

C++

```
CubicMetersPerUSDryBucket = CubicMetersPerUSDryPeck * 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSDryBucket.

153.3.19 StdConvs.CubicMetersPerUSDryBushel Constant

Pascal

```
CubicMetersPerUSDryBushel = CubicMetersPerUSDryBucket * 2;
```

C++

```
CubicMetersPerUSDryBushel = CubicMetersPerUSDryBucket * 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSDryBushel.

153.3.20 StdConvs.CubicMetersPerUSDryGallon Constant

Pascal

```
CubicMetersPerUSDryGallon = CubicMetersPerCubicInch * 268.8025;
```

C++

```
CubicMetersPerUSDryGallon = CubicMetersPerCubicInch * 268.8025;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSDryGallon.

153.3.21 StdConvs.CubicMetersPerUSDryPeck Constant

Pascal

```
CubicMetersPerUSDryPeck = CubicMetersPerUSDryGallon * 2;
```

C++

```
CubicMetersPerUSDryPeck = CubicMetersPerUSDryGallon * 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSDryPeck.

153.3.22 StdConvs.CubicMetersPerUSDryPint Constant

Pascal

```
CubicMetersPerUSDryPint = CubicMetersPerUSDryQuart / 2;
```

C++

```
CubicMetersPerUSDryPint = CubicMetersPerUSDryQuart / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSDryPint.

153.3.23 StdConvs.CubicMetersPerUSDryQuart Constant

Pascal

```
CubicMetersPerUSDryQuart = CubicMetersPerUSDryGallon / 4;
```

C++

```
CubicMetersPerUSDryQuart = CubicMetersPerUSDryGallon / 4;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSDryQuart.

153.3.24 StdConvs.CubicMetersPerUSFluidCup Constant

Pascal

```
CubicMetersPerUSFluidCup = CubicMetersPerUSFluidPint / 2;
```

C++

```
CubicMetersPerUSFluidCup = CubicMetersPerUSFluidPint / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidCup.

153.3.25 StdConvs.CubicMetersPerUSFluidGallon Constant

Pascal

```
CubicMetersPerUSFluidGallon = CubicMetersPerCubicInch * 231;
```

C++

```
CubicMetersPerUSFluidGallon = CubicMetersPerCubicInch * 231;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidGallon.

153.3.26 StdConvs.CubicMetersPerUSFluidGill Constant

Pascal

```
CubicMetersPerUSFluidGill = CubicMetersPerUSFluidCup / 2;
```

C++

```
CubicMetersPerUSFluidGill = CubicMetersPerUSFluidCup / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidGill.

153.3.27 StdConvs.CubicMetersPerUSFluidOunce Constant

Pascal

```
CubicMetersPerUSFluidOunce = CubicMetersPerUSFluidCup / 8;
```

C++

```
CubicMetersPerUSFluidOunce = CubicMetersPerUSFluidCup / 8;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidOunce.

153.3.28 StdConvs.CubicMetersPerUSFluidPint Constant

Pascal

```
CubicMetersPerUSFluidPint = CubicMetersPerUSFluidQuart / 2;
```

C++

```
CubicMetersPerUSFluidPint = CubicMetersPerUSFluidQuart / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidPint.

153.3.29 StdConvs.CubicMetersPerUSFluidQuart Constant

Pascal

```
CubicMetersPerUSFluidQuart = CubicMetersPerUSFluidGallon / 4;
```

C++

```
CubicMetersPerUSFluidQuart = CubicMetersPerUSFluidGallon / 4;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidQuart.

153.3.30 StdConvs.CubicMetersPerUSFluidTablespoon Constant

Pascal

```
CubicMetersPerUSFluidTablespoon = CubicMetersPerUSFluidOunce / 2;
```

C++

```
CubicMetersPerUSFluidTablespoon = CubicMetersPerUSFluidOunce / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidTablespoon.

153.3.31 StdConvs.CubicMetersPerUSFluidTeaspoon Constant

Pascal

```
CubicMetersPerUSFluidTeaspoon = CubicMetersPerUSFluidOunce / 6;
```

C++

```
CubicMetersPerUSFluidTeaspoon = CubicMetersPerUSFluidOunce / 6;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidTeaspoon.

153.3.32 StdConvs.GramsPerDrams Constant

Pascal

```
GramsPerDrams = GramsPerPound / 256;
```

C++

```
GramsPerDrams = GramsPerPound / 256;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerDrams.

153.3.33 StdConvs.GramsPerGrains Constant

Pascal

```
GramsPerGrains = GramsPerPound / 7000;
```

C++

```
GramsPerGrains = GramsPerPound / 7000;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerGrains.

153.3.34 StdConvs.GramsPerLongTons Constant

Pascal

```
GramsPerLongTons = GramsPerPound * 2240;
```

C++

```
GramsPerLongTons = GramsPerPound * 2240;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerLongTons.

153.3.35 StdConvs.GramsPerOunces Constant

Pascal

```
GramsPerOunces = GramsPerPound / 16;
```

C++

```
GramsPerOunces = GramsPerPound / 16;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerOunces.

153.3.36 StdConvs.GramsPerPound Constant

Pascal

```
GramsPerPound = 453.59237;
```

C++

```
GramsPerPound = 453.59237;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerPound.

153.3.37 StdConvs.GramsPerStones Constant

Pascal

```
GramsPerStones = GramsPerPound * 14;
```

C++

```
GramsPerStones = GramsPerPound * 14;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerStones.

153.3.38 StdConvs.GramsPerTons Constant

Pascal

```
GramsPerTons = GramsPerPound * 2000;
```

C++

```
GramsPerTons = GramsPerPound * 2000;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerTons.

153.3.39 StdConvs.MetersPerAstronomicalUnit Constant

Pascal

```
MetersPerAstronomicalUnit = 1.49598E11;
```

C++

```
MetersPerAstronomicalUnit = 1.49598E11;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerAstronomicalUnit.

153.3.40 StdConvs.MetersPerChain Constant

Pascal

```
MetersPerChain = MetersPerRod * 4;
```

C++

```
MetersPerChain = MetersPerRod * 4;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerChain.

153.3.41 StdConvs.MetersPerCubit Constant

Pascal

```
MetersPerCubit = 0.4572;
```

C++

```
MetersPerCubit = 0.4572;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerCubit.

153.3.42 StdConvs.MetersPerFathom Constant

Pascal

```
MetersPerFathom = MetersPerFoot * 6;
```

C++

```
MetersPerFathom = MetersPerFoot * 6;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerFathom.

153.3.43 StdConvs.MetersPerFoot Constant

Pascal

```
MetersPerFoot = MetersPerInch * 12;
```

C++

```
MetersPerFoot = MetersPerInch * 12;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerFoot.

153.3.44 StdConvs.MetersPerFurlong Constant

Pascal

```
MetersPerFurlong = MetersPerYard * 220;
```

C++

```
MetersPerFurlong = MetersPerYard * 220;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerFurlong.

153.3.45 StdConvs.MetersPerHand Constant

Pascal

```
MetersPerHand = MetersPerInch * 4;
```

C++

```
MetersPerHand = MetersPerInch * 4;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerHand.

153.3.46 StdConvs.MetersPerInch Constant

Pascal

```
MetersPerInch = 0.0254;
```

C++

```
MetersPerInch = 0.0254;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerInch.

153.3.47 StdConvs.MetersPerLightSecond Constant

Pascal

```
MetersPerLightSecond = 2.99792458E8;
```

C++

```
MetersPerLightSecond = 2.99792458E8;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerLightSecond.

153.3.48 StdConvs.MetersPerLightYear Constant

Pascal

```
MetersPerLightYear = MetersPerLightSecond * 31556925.9747;
```

C++

```
MetersPerLightYear = MetersPerLightSecond * 31556925.9747;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerLightYear.

153.3.49 StdConvs.MetersPerLink Constant

Pascal

```
MetersPerLink = MetersPerChain / 100;
```

C++

```
MetersPerLink = MetersPerChain / 100;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerLink.

153.3.50 StdConvs.MetersPerMile Constant

Pascal

```
MetersPerMile = MetersPerFoot * 5280;
```

C++

```
MetersPerMile = MetersPerFoot * 5280;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerMile.

153.3.51 StdConvs.MetersPerNauticalMiles Constant

Pascal

```
MetersPerNauticalMiles = 1852;
```

C++

```
MetersPerNauticalMiles = 1852;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerNauticalMiles.

153.3.52 StdConvs.MetersPerPace Constant

Pascal

```
MetersPerPace = MetersPerInch * 30;
```

C++

```
MetersPerPace = MetersPerInch * 30;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerPace.

153.3.53 StdConvs.MetersPerParsec Constant

Pascal

```
MetersPerParsec = MetersPerAstronomicalUnit * 206264.806247096;
```

C++

```
MetersPerParsec = MetersPerAstronomicalUnit * 206264.806247096;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerParsec.

153.3.54 StdConvs.MetersPerPica Constant

Pascal

```
MetersPerPica = MetersPerPoint * 12;
```

C++

```
MetersPerPica = MetersPerPoint * 12;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerPica.

153.3.55 StdConvs.MetersPerPoint Constant

Pascal

```
MetersPerPoint = MetersPerInch * 0.013837;
```

C++

```
MetersPerPoint = MetersPerInch * 0.013837;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerPoint.

153.3.56 StdConvs.MetersPerRod Constant

Pascal

```
MetersPerRod = MetersPerFoot * 16.5;
```

C++

```
MetersPerRod = MetersPerFoot * 16.5;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerRod.

153.3.57 StdConvs.MetersPerYard Constant

Pascal

```
MetersPerYard = MetersPerFoot * 3;
```

C++

```
MetersPerYard = MetersPerFoot * 3;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerYard.

153.3.58 StdConvs.SquareMetersPerAcre Constant

Pascal

```
SquareMetersPerAcre = SquareMetersPerSquareYard * 4840;
```

C++

```
SquareMetersPerAcre = SquareMetersPerSquareYard * 4840;
```

File

StdConvs

Description

This is constant StdConvs.SquareMetersPerAcre.

153.3.59 StdConvs.SquareMetersPerSquareFoot Constant

Pascal

```
SquareMetersPerSquareFoot = MetersPerFoot * MetersPerFoot;
```

C++

```
SquareMetersPerSquareFoot = MetersPerFoot * MetersPerFoot;
```

File

StdConvs

Description

This is constant StdConvs.SquareMetersPerSquareFoot.

153.3.60 StdConvs.SquareMetersPerSquareInch Constant

Pascal

```
SquareMetersPerSquareInch = MetersPerInch * MetersPerInch;
```

C++

```
SquareMetersPerSquareInch = MetersPerInch * MetersPerInch;
```

File

StdConvs

Description

This is constant StdConvs.SquareMetersPerSquareInch.

153.3.61 StdConvs.SquareMetersPerSquareMile Constant

Pascal

```
SquareMetersPerSquareMile = MetersPerMile * MetersPerMile;
```

C++

```
SquareMetersPerSquareMile = MetersPerMile * MetersPerMile;
```

File

StdConvs

Description

This is constant StdConvs.SquareMetersPerSquareMile.

153.3.62 StdConvs.SquareMetersPerSquareRod Constant

Pascal

```
SquareMetersPerSquareRod = MetersPerRod * MetersPerRod;
```

C++

```
SquareMetersPerSquareRod = MetersPerRod * MetersPerRod;
```

File

StdConvs

Description

This is constant StdConvs.SquareMetersPerSquareRod.

153.3.63 StdConvs.SquareMetersPerSquareYard Constant

Pascal

```
SquareMetersPerSquareYard = MetersPerYard * MetersPerYard;
```

C++

```
SquareMetersPerSquareYard = MetersPerYard * MetersPerYard;
```

File

StdConvs

Description

This is constant StdConvs.SquareMetersPerSquareYard.

154 StdCtrls Namespace

This is namespace StdCtrls.

Topics

Name	Description
Classes (see page 20909)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 21945)	The following table lists structs, records, enums in this documentation.
Types (see page 21953)	The following table lists types in this documentation.

Classes

	Name	Description
	TButton (see page 20909)	TButton is a push button control.
	TButtonActionLink (see page 20976)	TButtonActionLink is an action link designed specifically for button control clients.
	TButtonControl (see page 20981)	TButtonControl is the base class for objects that represent Windows button controls.
	TCheckBox (see page 20996)	TCheckBox represents a check box that can be on (checked) or off (unchecked)
	TComboBox (see page 21064)	TComboBox combines an edit box with a scrollable list.
	TCustomCheckBox (see page 21138)	TCustomCheckBox is the ancestor of all check-box components.
	TCustomCombo (see page 21154)	TCustomCombo is the base class for all combo boxes.
	TCustomComboBox (see page 21201)	TCustomComboBox is the base class for combobox components.
	TCustomComboBoxStrings (see page 21225)	TCustomComboBoxStrings represents the strings in the drop-down list of a combo box.
	TCustomEdit (see page 21232)	TCustomEdit is the base class from which all edit-box and memo controls are derived.
	TCustomGroupBox (see page 21275)	TCustomGroupBox is the base class for all group-box components.
	TCustomLabel (see page 21288)	TCustomLabel is the base class for non-windowed controls that display text on a form.
	TCustomListBox (see page 21308)	TCustomListBox is the base class for listbox objects that are wrappers for list box controls.
	TCustomMemo (see page 21362)	TCustomMemo is the base type for memo components, which are multiline edit boxes, including TMemo (see page 21678) and TDBMemo.
	TCustomStaticText (see page 21387)	TCustomStaticText is the base class for TStaticText (see page 21887).
	TEdit (see page 21402)	TEdit is a wrapper for a Windows single-line edit control.
	TGroupBox (see page 21474)	TGroupBox represents a Windows group box.
	TLabel (see page 21544)	TLabel is a nonwindowed control that displays text on a form.
	TListBox (see page 21598)	TListBox displays a collection of items in a scrollable list.
	TMemo (see page 21678)	TMemo is a wrapper for a Windows multiline edit control.
	TRadioButton (see page 21753)	TRadioButton is a wrapper for a system radio button.

	TScrollBar (see page 21821)	TScrollBar is a Windows scroll bar, which is used to scroll the contents of a window, form, or control.
	TStaticText (see page 21887)	TStaticText is a windowed control that displays text on a form.

Structs, Records, Enums

	Name	Description
	TCheckBoxState (see page 21945)	TCheckBoxState indicates the state of a check box control.
	TComboBoxStyle (see page 21946)	TComboBoxStyle is the type for the Style property of combo box controls that are defined in the StdCtrls unit.
	TEditCharCase (see page 21946)	TEditCharCase indicates how a control forces the case of characters, if at all.
	TEllipsisPosition (see page 21947)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	TListBoxStyle (see page 21948)	TListBoxStyle specifies the way a listbox control gets its data and how it is drawn.
	TScrollCode (see page 21949)	TScrollCode indicates what action the user performed on a scroll bar.
	TScrollStyle (see page 21950)	Identifies a set of scrollbars by their orientation.
	TSelection (see page 21950)	TSelection definese the start and position of a selection.
	TStaticBorderStyle (see page 21951)	TStaticBorderStyle determines the appearance of the border on a static text control.
	TTextLayout (see page 21951)	TTextLayout indicates where text appears within the client area of a control.

Types

Name	Description
TButtonActionLinkClass (see page 21953)	This is type StdCtrls.TButtonActionLinkClass.
TCustomComboBoxStringsClass (see page 21953)	TCustomComboBoxStringsClass describes a descendant of TCustomComboBoxStrings (see page 21225).
TDrawItemEvent (see page 21954)	TScrollEvent (see page 21957) is the type of an OnScroll event handler.
TLBFindDataEvent (see page 21954)	TLBFindDataEvent is the type of the list box OnDataFind event handler.
TLBGetDataEvent (see page 21955)	TLBGetDataEvent is the type of the list box OnData event handler.
TLBGetDataObjectEvent (see page 21955)	TLBGetDataObjectEvent is the type of the list box OnDataObject event handler.
TMeasureItemEvent (see page 21955)	TMeasureItemEvent is the type for the OnMeasureItem event handler.
TOwnerDrawState (see page 21956)	Indicates state information that can influence how an item is drawn.
TScrollEvent (see page 21957)	This is type StdCtrls.TScrollEvent.

154.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TButton (see page 20909)	TButton is a push button control.
	TButtonActionLink (see page 20976)	TButtonActionLink is an action link designed specifically for button control clients.
	TButtonControl (see page 20981)	TButtonControl is the base class for objects that represent Windows button controls.
	TCheckBox (see page 20996)	TCheckBox represents a check box that can be on (checked) or off (unchecked)
	TComboBox (see page 21064)	TComboBox combines an edit box with a scrollable list.
	TCustomCheckBox (see page 21138)	TCustomCheckBox is the ancestor of all check-box components.
	TCustomCombo (see page 21154)	TCustomCombo is the base class for all combo boxes.
	TCustomComboBox (see page 21201)	TCustomComboBox is the base class for combobox components.
	TCustomComboBoxStrings (see page 21225)	TCustomComboBoxStrings represents the strings in the drop-down list of a combo box.
	TCustomEdit (see page 21232)	TCustomEdit is the base class from which all edit-box and memo controls are derived.
	TCustomGroupBox (see page 21275)	TCustomGroupBox is the base class for all group-box components.
	TCustomLabel (see page 21288)	TCustomLabel is the base class for non-windowed controls that display text on a form.
	TCustomListBox (see page 21308)	TCustomListBox is the base class for listbox objects that are wrappers for list box controls.
	TCustomMemo (see page 21362)	TCustomMemo is the base type for memo components, which are multiline edit boxes, including TMemo (see page 21678) and TDBMemo.
	TCustomStaticText (see page 21387)	TCustomStaticText is the base class for TStaticText (see page 21887).
	TEdit (see page 21402)	TEdit is a wrapper for a Windows single-line edit control.
	TGroupBox (see page 21474)	TGroupBox represents a Windows group box.
	TLabel (see page 21544)	TLabel is a nonwindowed control that displays text on a form.
	TListBox (see page 21598)	TListBox displays a collection of items in a scrollable list.
	TMemo (see page 21678)	TMemo is a wrapper for a Windows multiline edit control.
	TRadioButton (see page 21753)	TRadioButton is a wrapper for a system radio button.
	TScrollBar (see page 21821)	TScrollBar is a Windows scroll bar, which is used to scroll the contents of a window, form, or control.
	TStaticText (see page 21887)	TStaticText is a windowed control that displays text on a form.

154.1.1 TButton Class

TButton is a push button control.

Class Hierarchy

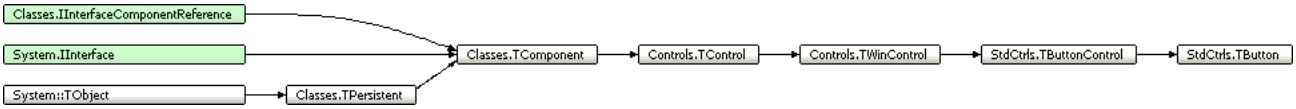

Pascal

```
TButton = class(TButtonControl);
```

C++

```
class TButton : public TButtonControl;
```

File

StdCtrls

Description

Use TButton to put a standard push button on a form. TButton introduces several properties to control its behavior in a dialog box setting. Users choose button controls to initiate actions.

To use a button that displays a bitmap instead of a label, use TBitBtn. To use a button that can remain in a depressed position, use TSpeedButton.

Note: Since the TButton caption is always centered, changing the BiDi alignment has no effect.

See Also

TBitBtn (see page 2290)

TSpeedButton (see page 2335)

Button Controls

Buttons and Similar Controls

154.1.1.1 TButton Members

The following tables list the members exposed by TButton.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.

	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
V	ActionChange (see page 20992)	Updates the button control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 20921)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 20922)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.

	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetActionLinkClass (see page 20993)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 20993)	Returns whether the button control is checked.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.

	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.

	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetButtonStyle (see page 20922)	This is SetButtonStyle, a member of class TButton.
	SetChecked (see page 20994)	Sets the Checked (see page 20995) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WndProc (see page 20994)	Handles Windows messages.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Checked (see page 20995)	Specifies whether the button control is checked.
	ClicksDisabled (see page 20996)	This is ClicksDisabled, a member of class TButtonControl.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Click (see page 20921)	Simulates a mouse click, as if the user had clicked the button.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 20920)	Creates and initializes a TButton (see page 20909) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.

	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
V	UseRightToLeftAlignment (see page 20923)	Specifies whether the control's alignment is in a right-to-left mode.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.

	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 20949)	Occurs when the user clicks the control.
	OnContextPopup (see page 20951)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 20951)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 20954)	Occurs when the user drags an object over a control.
	OnEndDock (see page 20956)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 20956)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 20957)	Occurs when a control receives the input focus.
	OnExit (see page 20960)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 20961)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 20964)	Occurs when a key is pressed.
	OnKeyUp (see page 20965)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 20967)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 20968)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 20970)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 20970)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 20971)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 20973)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 20975)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 20976)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Action (see page 20924)	Designates the action associated with the control.
	Align (see page 20924)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 20927)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 20928)	Specifies the bi-directional mode for the control.
	Cancel (see page 20929)	Determines whether the button's OnClick (see page 20949) event handler executes when the Escape key is pressed.
	Caption (see page 20929)	Specifies a text string that identifies the control to the user.
	Constraints (see page 20932)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Default (see page 20932)	Determines whether the button's OnClick (see page 20949) event handler executes when the Enter key is pressed.
	DragCursor (see page 20933)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 20934)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 20934)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 20936)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 20936)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	ModalResult (see page 20937)	Determines whether and how the button closes its (modal) parent form.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 20938)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 20938)	Determines where a control looks for its font information.
	ParentShowHint (see page 20939)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 20939)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 20944)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 20945)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 20946)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 20946)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	WordWrap (see page 20948)	Specifies whether the button text wraps to fit the width of the control.

154.1.1.2 TButton.Create Constructor

Creates and initializes a TButton (see page 20909) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TButton(TComponent * AOwner);
```

Description

Use Create to programmatically instantiate a button. Buttons (see page 2289) added in the form designer are created automatically.

AOwner is the component that is responsible for freeing the button instance. It becomes the value of the Owner property.

See Also

Owner (see page 2807)

154.1.1.3 TButton Methods

The methods of the TButton class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 20921)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 20922)	Creates the underlying window.
	SetButtonStyle (see page 20922)	This is SetButtonStyle, a member of class TButton.

Public Methods

	Name	Description
	Click (see page 20921)	Simulates a mouse click, as if the user had clicked the button.
	UseRightToLeftAlignment (see page 20923)	Specifies whether the control's alignment is in a right-to-left mode.

154.1.1.3.1 TButton.Click Method

Simulates a mouse click, as if the user had clicked the button.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Calling Click assigns the value of the button's ModalResult (see page 20937) property to the ModalResult (see page 20937) property of its parent form, and then generates an OnClick (see page 20949) event.

See Also

ModalResult (see page 20937)

OnClick (see page 5851)

154.1.1.3.2 TButton.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

154.1.1.3.3 TButton.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams (see page 6135)

CreateWindowHandle (see page 6137)

Perform (see page 5776)

DestroyWnd (see page 6141)

154.1.1.3.4 TButton.SetButtonStyle Method

Pascal

```
procedure SetButtonStyle(ADefault: Boolean); virtual;
```

C++

```
virtual __fastcall SetButtonStyle(Boolean ADefault);
```

Description

This is SetButtonStyle, a member of class TButton.

154.1.1.3.5 TButton.UseRightToLeftAlignment Method

Specifies whether the control's alignment is in a right-to-left mode.

Pascal

```
function UseRightToLeftAlignment: Boolean; override;
```

C++

```
virtual __fastcall Boolean UseRightToLeftAlignment();
```

Description

For TButton (see page 20909), UseRightToLeftAlignment always returns false. This is because the alignment of a button should never be reversed automatically when running in Middle Eastern locales.

See Also

ModalResult (see page 20937)

OnClick (see page 5851)

154.1.1.4 TButton Properties

The properties of the TButton class are listed here.

Published Properties

	Name	Description
	Action (see page 20924)	Designates the action associated with the control.
	Align (see page 20924)	Determines how the control aligns within its container (parent control).
	Anchors (see page 20927)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 20928)	Specifies the bi-directional mode for the control.
	Cancel (see page 20929)	Determines whether the button's OnClick (see page 20949) event handler executes when the Escape key is pressed.
	Caption (see page 20929)	Specifies a text string that identifies the control to the user.
	Constraints (see page 20932)	Specifies the size constraints for the control.
	Default (see page 20932)	Determines whether the button's OnClick (see page 20949) event handler executes when the Enter key is pressed.
	DragCursor (see page 20933)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 20934)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 20934)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 20936)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 20936)	Controls the attributes of text written on or in the control.
	ModalResult (see page 20937)	Determines whether and how the button closes its (modal) parent form.
	ParentBiDiMode (see page 20938)	Specifies whether the control uses its parent's BiDiMode.
	ParentFont (see page 20938)	Determines where a control looks for its font information.
	ParentShowHint (see page 20939)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 20939)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 20944)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 20945)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 20946)	Determines if the user can tab to a control.
	Visible (see page 20946)	Determines whether the component appears onscreen.

	WordWrap (see page 20948)	Specifies whether the button text wraps to fit the width of the control.

154.1.1.4.1 TButton.Action Property

Designates the action associated with the control.

Pascal

```
property Action: TBasicAction;
```

C++

```
__property TBasicAction Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

154.1.1.4.2 TButton.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
```

```
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

154.1.1.4.3 TButton.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

154.1.1.4.4 TButton.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

154.1.1.4.5 TButton.Cancel Property

Determines whether the button's OnClick (see page 20949) event handler executes when the Escape key is pressed.

Pascal

```
property Cancel: Boolean;
```

C++

```
__property Boolean Cancel;
```

Description

If Cancel is true, the button's OnClick (see page 20949) event handler executes when the user presses Esc. Although an application can have more than one Cancel button, the form calls the OnClick (see page 20949) event handler only for the first visible button in the tab order.

See Also

Default (see page 20932)

OnClick (see page 5851)

154.1.1.4.6 TButton.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
```

```
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

154.1.1.4.7 TButton.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

154.1.1.4.8 TButton.Default Property

Determines whether the button's OnClick (see page 20949) event handler executes when the Enter key is pressed.

Pascal

```
property Default: Boolean;
```

C++

```
__property Boolean Default;
```

Description

If Default is true, the button's OnClick (see page 20949) event handler executes when the user presses Enter.

Although an application can have more than one Default button, the form calls the OnClick (see page 20949) event handler only for the first visible button in the tab order. Moreover, any button that has focus becomes the Default button temporarily; hence, if the user selects another button before pressing Enter, the selected button's OnClick (see page 20949) event handler executes instead.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
```

```
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FileListBox1->FileEdit = Edit1;
  FilterComboBox1->FileList = FileListBox1;
  DirectoryListBox1->FileList = FileListBox1;
  DirectoryListBox1->DirLabel = Label1;
  DriveComboBox1->DirList = DirectoryListBox1;
  Button1->Default = true;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FileListBox1->ApplyFilePath(Edit1->Text);
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FileListBox1.ApplyFilePath(Edit1.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileEdit := Edit1;
  FilterComboBox1.FileList := FileListBox1;
  DirectoryListBox1.FileList := FileListBox1;
  DirectoryListBox1.DirLabel := Label1;
  DriveComboBox1.DirList := DirectoryListBox1;
  Button1.Default := True;
end;
```

See Also

Cancel (see page 20929)

OnClick (see page 5851)

154.1.1.4.9 TButton.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

154.1.1.4.10 TButton.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

154.1.1.4.11 TButton.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
```

```
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

154.1.1.4.12 TButton.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

154.1.1.4.13 TButton.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
```

```
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

154.1.1.4.14 TButton.ModalResult Property

Determines whether and how the button closes its (modal) parent form.

Pascal

```
property ModalResult: TModalResult;
```

C++

```
__property TModalResult ModalResult;
```

Description

Setting the TButton (see page 20909) component's ModalResult property is an easy way to make clicking the button close a modal form. When a button is clicked, the ModalResult property of its parent form is set to the same value as the button's ModalResult property.

For example, if a dialog box has OK and Cancel (see page 20929) buttons, their ModalResult properties could be set at design time to mrOk and mrCancel, respectively. At runtime, clicking the OK button then changes the dialog's ModalResult property to mrOk, and clicking the Cancel (see page 20929) button changes the dialog's ModalResult property to mrCancel. Unless further processing is required, no OnClick (see page 20949) event handlers are required for the buttons.

The following constants are defined for the ModalResult property:

Constant	Value	Meaning
mrNone	0	None. Used as a default value before the user exits.
mrOk	idOK	The user exited with OK button.

mrCancel	idCancel	The user exited with the CANCEL button.
mrAbort	idAbort	The user exited with the ABORT button.
mrRetry	idRetry	The user exited with the RETRY button.
mrIgnore	idIgnore	The user exited with the IGNORE button.
mrYes	idYes	The user exited with the YES button.
mrNo	idNo	The user exited with the NO button.
mrAll	mrNo + 1	The user exited with the ALL button.
mrNoToAll	mrAll + 1	The user exited with the NO TO ALL button.
mrYesToAll	mrNoToAll + 1	The user exited with the YES TO ALL button.

154.1.1.4.15 TButton.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

154.1.1.4.16 TButton.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

154.1.1.4.17 TButton.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

154.1.1.4.18 TButton.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
```

```
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}
```

```
void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
```

```
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
```

```
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

154.1.1.4.19 TButton.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
```

```
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

154.1.1.4.20 TButton.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder

value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

154.1.1.4.21 TButton.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

154.1.1.4.22 TButton.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false,

the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
```

```
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

154.1.1.4.23 TButton.WordWrap Property

Specifies whether the button text wraps to fit the width of the control.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the display of multiple lines of text. When WordWrap is true, text that is too wide for the control wraps at the right margin.

Set WordWrap to false to limit the text to a single line. When WordWrap is false, text that is too wide for the control appears truncated.

See Also

ClientWidth (see page 5811)

154.1.1.5 TButton Events

The events of the TButton class are listed here.

Published Events

	Name	Description
	OnClick (see page 20949)	Occurs when the user clicks the control.
	OnContextPopup (see page 20951)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 20951)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 20954)	Occurs when the user drags an object over a control.
	OnEndDock (see page 20956)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 20956)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 20957)	Occurs when a control receives the input focus.
	OnExit (see page 20960)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 20961)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 20964)	Occurs when a key is pressed.

	OnKeyUp (see page 20965)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 20967)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 20968)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 20970)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 20970)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 20971)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 20973)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 20975)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 20976)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

154.1.1.5.1 TButton.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
```

```
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

154.1.1.5.2 TButton.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

154.1.1.5.3 TButton.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y

parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
```

```
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code comes from an application that contains a list box
  and three labels, each with a different font and color.  The
  DragMode property for each of the labels is dmAutomatic.
  The user can select a label and drag it to a list box and
  drop it. When the label is dropped, the items in the list
  box assume the color and font of the dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

154.1.1.5.4 TButton.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
```

```
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.1.5.5 TButton.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.1.5.6 TButton.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

154.1.1.5.7 TButton.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
```

```
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
```

```
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

154.1.1.5.8 TButton.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
```

```
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

154.1.1.5.9 TButton.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
```

```
      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

154.1.1.5.10 TButton.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys

that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

154.1.1.5.11 TButton.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
```

```
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

154.1.1.5.12 TButton.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the

OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.1.5.13 TButton.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
```

```
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
```

```
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.1.5.14 TButton.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

154.1.1.5.15 TButton.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

154.1.1.5.16 TButton.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
```

```
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
```

```
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.1.5.17 TButton.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
```

```
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
```

```
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

154.1.1.5.18 TButton.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

154.1.1.5.19 TButton.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

154.1.2 TButtonActionLink Class

TButtonActionLink is an action link designed specifically for button control clients.

Class Hierarchy

System::TObject → Classes.TBasicActionLink → ActnList.TActionLink → Controls.TControlActionLink → Controls.TWinControlActionLink → StdCtrls.TButtonActionLink

Pascal

```
TButtonActionLink = class(TWinControlActionLink);
```

C++

```
class TButtonActionLink : public TWinControlActionLink;
```

File

StdCtrls

Description

TButtonActionLink is designed to specifically link actions to button control clients. TButtonActionLink overrides methods to determine whether the action and client are linked, and to ensure that action events are appropriate for properties of button control clients.

TButtonActionLink inherits, from its ancestors, support for linking actions with the properties of components, generic controls, and windowed controls. TButtonActionLink adds support for linking the checked property to the appropriate property of a button control.

154.1.2.1 TButtonActionLink Members

The following tables list the members exposed by TButtonActionLink.

Protected Fields

	Name	Description
	FAction (see page 2722)	This is FAction, a member of class TBasicActionLink.
	FClient (see page 20979)	This is FClient, a member of class TButtonActionLink.

Protected Methods

	Name	Description
	AssignClient (see page 20980)	Assigns a client button that is linked to the associated action.
	Change (see page 2723)	This is Change, a member of class TBasicActionLink.
	DoShowHint (see page 5896)	Indicates whether the hint is shown, and modified.
	IsCaptionLinked (see page 268)	Indicates whether the Caption properties for the action and client are linked.
	IsCaptionLinked (see page 5896)	Indicates whether the caption for the action and the client are linked.
	IsCheckedLinked (see page 20980)	Indicates whether the checked state for the action and the client are linked.
	IsDropdownMenuLinked (see page 5896)	This is IsDropdownMenuLinked, a member of class TControlActionLink.
	IsEnabledLinked (see page 269)	Indicates whether the Enabled properties for the action and client are linked.
	IsEnabledLinked (see page 5897)	Indicates whether the enabled state for the action and the client are linked.
	IsEnableDropdownLinked (see page 5897)	This is IsEnableDropdownLinked, a member of class TControlActionLink.
	IsGroupIndexLinked (see page 270)	Indicates whether the GroupIndex property for the action controls client behavior.
	IsHelpContextLinked (see page 270)	Indicates whether the HelpContext properties for the action and client controls are linked.
	IsHelpContextLinked (see page 6236)	Indicates whether the Help context for the action and the client are linked.
	IsHelpLinked (see page 271)	Indicates whether the help properties for the action and client are linked.
	IsHelpLinked (see page 5897)	Indicates whether the help properties for the action and the client are linked.
	IsHintLinked (see page 271)	Indicates whether the Hint properties for the action and client are linked.
	IsHintLinked (see page 5898)	Indicates whether the Hint property for the action and the client are linked.
	IsImageIndexLinked (see page 272)	Indicates whether the image index for the action and the client are linked.
	IsOnExecuteLinked (see page 2723)	This is IsOnExecuteLinked, a member of class TBasicActionLink.

	IsOnExecuteLinked (see page 5898)	Indicates whether the OnExecute event handler for the action and the client are linked.
	IsPopupMenuLinked (see page 5898)	This is IsPopupMenuLinked, a member of class TControlActionLink.
	IsShortCutLinked (see page 272)	Indicates whether the ShortCut properties for the action and client are linked.
	IsVisibleLinked (see page 273)	Indicates whether the Visible properties for the action and client are linked.
	IsVisibleLinked (see page 5899)	Indicates whether the visible state for the action and the client are linked.
	SetAction (see page 2724)	This is SetAction, a member of class TBasicActionLink.
	SetAutoCheck (see page 274)	Introduces an interface for setting the AutoCheck property of a client.
	SetCaption (see page 274)	Introduces an interface for setting the caption on a client.
	SetCaption (see page 5899)	Sets the caption of a control if the client is linked to the action for this event.
	SetChecked (see page 20980)	Sets the checked state of a button control if the client is linked to the action for this property.
	SetDropdownMenu (see page 5899)	This is SetDropdownMenu, a member of class TControlActionLink.
	SetEnabled (see page 275)	Introduces an interface for setting the enabled state of a client.
	SetEnabled (see page 5900)	Sets the enabled state of a control if the client is linked to the action for this event.
	SetEnableDropdown (see page 5900)	This is SetEnableDropdown, a member of class TControlActionLink.
	SetGroupIndex (see page 275)	Introduces an interface for setting the GroupIndex property of a client.
	SetHelpContext (see page 276)	Introduces an interface for setting the help context of a client.
	SetHelpContext (see page 5900)	Sets the HelpContext of a control if the client is linked to the action help properties.
	SetHelpContext (see page 6237)	Sets the Help context of a windowed control if the client is linked to the action for this event.
	SetHelpKeyword (see page 276)	Introduces an interface for setting the help keyword of a client.
	SetHelpKeyword (see page 5901)	Sets the HelpKeyword of a control if the client is linked to the action help properties.
	SetHelpType (see page 277)	Introduces an interface for setting the HelpType property of a client.
	SetHelpType (see page 5901)	Sets the HelpType of a control if the client is linked to the action help properties.
	SetHint (see page 278)	Introduces an interface for setting the hint for a client.
	SetHint (see page 5901)	Sets the hint of a control if the client is linked to the action for this event.
	SetImageIndex (see page 278)	Introduces an interface for setting the image index for a client.
	SetOnExecute (see page 2724)	This is SetOnExecute, a member of class TBasicActionLink.
	SetOnExecute (see page 5902)	Sets the execution event handler for a control if the client is linked to the action for this event.
	SetPopupMenu (see page 5902)	This is SetPopupMenu, a member of class TControlActionLink.
	SetShortCut (see page 279)	Introduces an interface for setting the shortcut for a client.
	SetVisible (see page 279)	Introduces an interface for setting the visible state of a client.
	SetVisible (see page 5902)	Sets the visible state of a control if the client is linked to the action for this event.

Public Events

	Name	Description
	OnChange (see page 2725)	Occurs when an OnChange event is triggered on the action.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2721)	Instantiates and initializes a TBasicActionLink (see page 2720) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

	Destroy (see page 2722)	Destroys an instance of an action link object.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 2723)	Invokes the action's Execute method.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Update (see page 2724)	Invokes the action's Update method.

Public Properties

	Name	Description
	Action (see page 2725)	Indicates which action the link connects to its client.

154.1.2.2 TButtonActionLink Fields

The fields of the TButtonActionLink class are listed here.

Protected Fields

	Name	Description
	FClient (see page 20979)	This is FClient, a member of class TButtonActionLink.

154.1.2.2.1 TButtonActionLink.FClient Field

Pascal

```
FClient: TButtonControl;
```

C++

```
TButtonControl FClient;
```

Description

This is FClient, a member of class TButtonActionLink.

154.1.2.3 TButtonActionLink Methods

The methods of the TButtonActionLink class are listed here.

Protected Methods

	Name	Description
	AssignClient (see page 20980)	Assigns a client button that is linked to the associated action.
	IsCheckedLinked (see page 20980)	Indicates whether the checked state for the action and the client are linked.
	SetChecked (see page 20980)	Sets the checked state of a button control if the client is linked to the action for this property.

154.1.2.3.1 TButtonActionLink.AssignClient Method

Assigns a client button that is linked to the associated action.

Pascal

```
procedure AssignClient(AClient: TObject); override;
```

C++

```
virtual __fastcall AssignClient(TObject * AClient);
```

Description

TButtonActionLink (see page 20976) overrides AssignClient to ensure that the client for the action is a button control.

See Also

Action (see page 5797)

154.1.2.3.2 TButtonActionLink.IsCheckedLinked Method

Indicates whether the checked state for the action and the client are linked.

Pascal

```
function IsCheckedLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCheckedLinked();
```

Description

IsCheckedLinked indicates whether the Checked property is linked to the appropriate client property. IsCheckedLinked returns true if the client requests that the action control this property, false otherwise.

See Also

Checked (see page 308)

Checked (see page 20995)

SetChecked (see page 20980)

154.1.2.3.3 TButtonActionLink.SetChecked Method

Sets the checked state of a button control if the client is linked to the action for this property.

Pascal

```
procedure SetChecked(Value: Boolean); override;
```

C++

```
virtual __fastcall SetChecked(Boolean Value);
```

Description

SetChecked sets the checked state of a button control, and is called only if the client is linked for this property. Value specifies whether the checked state should be set.

See Also

Checked (see page 308)

Checked (see page 20995)

IsCheckedLinked (see page 20980)

154.1.3 TButtonControl Class

TButtonControl is the base class for objects that represent Windows button controls.

Class Hierarchy

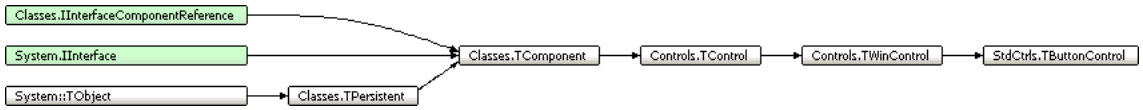

Pascal

```
TButtonControl = class(TWinControl);
```

C++

```
class TButtonControl : public TWinControl;
```

File

StdCtrls

Description

TButtonControl encapsulates behavior common to button controls, check boxes, and radio buttons. Use TButtonControl as a base class when defining a button that is a wrapper for a Windows button. Do not create instances of TButtonControl directly in an application. Instead, use a descendant of TButtonControl to put a push button, check box, or radio button on a form.

Note: Some buttons, such as TSpeedButton, are not wrappers for Windows buttons. These controls do not descend from TButtonControl.

See Also

TButton (see page 20909)

TCustomCheckBox (see page 21138)

TRadioButton (see page 21753)

TSpeedButton (see page 2335)

154.1.3.1 TButtonControl Members

The following tables list the members exposed by TButtonControl.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.

	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	ActionChange (see page 20992)	Updates the button control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 20993)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.

	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 20993)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 20993)	Returns whether the button control is checked.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.

	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.

	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChecked (see page 20994)	Sets the Checked (see page 20995) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 20994)	Handles Windows messages.

	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Checked (see page 20995)	Specifies whether the button control is checked.
	ClicksDisabled (see page 20996)	This is ClicksDisabled, a member of class TButtonControl.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.
	WordWrap (see page 20996)	Specifies whether the button text wraps to fit the width of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 20992)	Creates a button control.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.

V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.

R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

154.1.3.2 TButtonControl.Create Constructor

Creates a button control.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TButtonControl(TComponent * AOwner);
```

Description

AOwner is the component that is responsible for freeing the button. It becomes the value of the Owner property.

154.1.3.3 TButtonControl Methods

The methods of the TButtonControl class are listed here.

Protected Methods

	Name	Description
	ActionChange (see page 20992)	Updates the button control to reflect changes in its associated action.
	CreateParams (see page 20993)	Initializes a window-creation parameter data structure.
	GetActionLinkClass (see page 20993)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 20993)	Returns whether the button control is checked.
	SetChecked (see page 20994)	Sets the Checked (see page 20995) property.
	WndProc (see page 20994)	Handles Windows messages.

154.1.3.3.1 TButtonControl.ActionChange Method

Updates the button control to reflect changes in its associated action.

Pascal

```
procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); override;
```

C++

```
virtual __fastcall ActionChange(TObject * Sender, Boolean CheckDefaults);
```

Description

ActionChange is called automatically when the associated action changes. It updates the button control to reflect the action's current Checked (see page 20995) state. Then it calls the inherited method, which checks the associated properties for that client type.

The Sender parameter is the associated action. The CheckDefaults parameter specifies whether properties and event handlers should be changed if they differ from the default values. When CheckDefaults is true, properties and event handlers are only changed when the current values are the default values (that is, when the values have not been previously changed). When CheckDefaults is false, properties and event handlers are changed, regardless of their current values.

See Also

TCustomAction (see page 297)

154.1.3.3.2 TButtonControl.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

154.1.3.3.3 TButtonControl.GetActionLinkClass Method

Returns the appropriate action link class that is associated with button controls.

Pascal

```
function GetActionLinkClass: TControlActionLinkClass; override;
```

C++

```
virtual __fastcall TControlActionLinkClass GetActionLinkClass();
```

Description

GetActionLinkClass is used to get the appropriate class type for the action link used with button controls. GetActionLinkClass is used internally to create action link objects for clients when a button control has an associated action. The action link is used to link the action to the button control client. Each Action link class type is designed to link specific actions to clients based upon the types of properties the client supports. For button controls, GetActionLinkClass returns the TButtonActionLink (see page 20976) class type.

See Also

TButtonActionLink (see page 20976)

154.1.3.3.4 TButtonControl.GetChecked Method

Returns whether the button control is checked.

Pascal

```
function GetChecked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetChecked();
```

Description

GetChecked is the read implementation of the Checked (see page 20995) property.

See Also

Checked (see page 20995)

154.1.3.3.5 TButtonControl.SetChecked Method

Sets the Checked (see page 20995) property.

Pascal

```
procedure SetChecked(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetChecked(Boolean Value);
```

Description

SetChecked is the write implementer for the Checked (see page 20995) property.

See Also

Checked (see page 20995)

154.1.3.3.6 TButtonControl.WndProc Method

Handles Windows messages.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc enables button objects to receive input focus when WM_LBUTTONDOWN and WM_LBUTTONDBLCLK messages occur.

See Also

WndProc (see page 5791)

154.1.3.4 TButtonControl Properties

The properties of the TButtonControl class are listed here.

Protected Properties

	Name	Description
	Checked (see page 20995)	Specifies whether the button control is checked.

	ClicksDisabled (see page 20996)	This is ClicksDisabled, a member of class TButtonControl.
	WordWrap (see page 20996)	Specifies whether the button text wraps to fit the width of the control.

154.1.3.4.1 TButtonControl.Checked Property

Specifies whether the button control is checked.

Pascal

```
property Checked: Boolean;
```

C++

```
__property Boolean Checked;
```

Description

Use Checked to determine whether a button control is checked.

C++ Examples:

```
/*
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
*/

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->MultiLine = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->Caption = "Stack Tabs?";
  for (int i = 0; i < 20; i++)
  {
    TTabSheet *pPage = new TTabSheet(PageControl1);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
  }
}
```

Delphi Examples:

```
{
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.MultiLine := CheckBox1.Checked;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
```

```
    Checkbox1.Caption := 'Stack Tabs?';
    for i := 0 to 20 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'Tab #' + IntToStr(i);
      end;
  end;
```

See Also

WndProc (see page 5791)

154.1.3.4.2 TButtonControl.ClicksDisabled Property

Pascal

```
property ClicksDisabled: Boolean;
```

C++

```
__property Boolean ClicksDisabled;
```

Description

This is ClicksDisabled, a member of class TButtonControl.

154.1.3.4.3 TButtonControl.WordWrap Property

Specifies whether the button text wraps to fit the width of the control.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the display of multiple lines of text. When WordWrap is true, text that is too wide for the control wraps at the right margin.

Set WordWrap to false to limit the text to a single line. When WordWrap is false, text that is too wide for the control appears truncated.

See Also

ClientWidth (see page 5811)

154.1.4 TCheckBox Class

TCheckBox represents a check box that can be on (checked) or off (unchecked)

Class Hierarchy

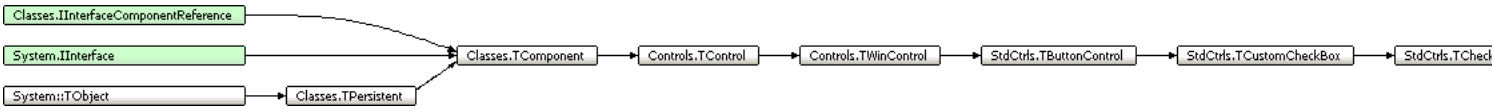

Pascal

```
TCheckBox = class(TCustomCheckBox);
```

C++

```
class TCheckBox : public TCustomCheckBox;
```

File

StdCtrls

Description

A TCheckBox component presents an option for the user. The user can check the box to select the option, or uncheck it to deselect the option.

See Also

TCustomCheckBox (see page 21138)

Buttons and Similar Controls

Check Boxes

154.1.4.1 TCheckBox Members

The following tables list the members exposed by TCheckBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
V	ActionChange (see page 20992)	Updates the button control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	Click (see page 21149)	Click generates an OnClick event.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 20993)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 21149)	Initializes a window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CreateWnd (see page 21150)	Creates a Windows control corresponding to the check box.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.

	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetActionLinkClass (see page 20993)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 20993)	Returns whether the button control is checked.
	GetChecked (see page 21150)	Returns the value of the Checked (see page 20995) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.

	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.

	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
V	SetChecked (see page 20994)	Sets the Checked (see page 20995) property.
V	SetChecked (see page 21151)	Sets the value of the Checked (see page 20995) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
V	Toggle (see page 21151)	Changes the state of the check box.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WndProc (see page 20994)	Handles Windows messages.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	ClicksDisabled (see page 20996)	This is ClicksDisabled, a member of class TButtonControl.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.

	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 5731)	Creates an instance of TControl (see page 5724).
	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 20992)	Creates a button control.
	Create (see page 21148)	Creates and initializes a TCustomCheckBox (see page 21138) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.

	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
V	GetControlsAlignment (see page 21151)	Returns the alignment of the check box caption.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.

R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 21036)	Occurs when the user clicks the control.
	OnContextPopup (see page 21038)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 21039)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21041)	Occurs when the user drags an object over a control.
	OnEndDock (see page 21044)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21044)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 21045)	Occurs when a control receives the input focus.
	OnExit (see page 21048)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 21049)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 21052)	Occurs when a key is pressed.
	OnKeyUp (see page 21053)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 21055)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 21056)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 21058)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21058)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 21058)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

	OnMouseUp (see page 21061)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 21063)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21064)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Action (see page 21008)	Designates the action associated with the control.
	Align (see page 21009)	Determines how the control aligns within its container (parent control).
	Alignment (see page 21012)	Determines whether the checkbox label aligns to the left or right of the tick box.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	AllowGrayed (see page 21013)	Determines whether check box can be in a "grayed" state.
	Anchors (see page 21013)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 21014)	Specifies the bi-directional mode for the control.
	Caption (see page 21014)	Specifies a text string that identifies the control to the user.
	Checked (see page 21017)	Specifies whether the button control is checked.
	Color (see page 21018)	Specifies the background color of the control.
	Constraints (see page 21019)	Specifies the size constraints for the control.
	Ctl3D (see page 21020)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 21020)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21020)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21021)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 21022)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 21023)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 21024)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 21024)	Determines where a control looks for its color information.
	ParentCtl3D (see page 21025)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 21025)	Determines where a control looks for its font information.
	ParentShowHint (see page 21026)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 21026)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 21031)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	State (see page 21032)	Indicates whether the check box is selected, deselected, or grayed.
	TabOrder (see page 21033)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21034)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 21034)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	WordWrap (see page 21035)	Specifies whether the button text wraps to fit the width of the control.

154.1.4.2 TCheckBox Properties

The properties of the TCheckBox class are listed here.

Published Properties

	Name	Description
	Action (see page 21008)	Designates the action associated with the control.
	Align (see page 21009)	Determines how the control aligns within its container (parent control).
	Alignment (see page 21012)	Determines whether the checkbox label aligns to the left or right of the tick box.
	AllowGrayed (see page 21013)	Determines whether check box can be in a "grayed" state.
	Anchors (see page 21013)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 21014)	Specifies the bi-directional mode for the control.
	Caption (see page 21014)	Specifies a text string that identifies the control to the user.
	Checked (see page 21017)	Specifies whether the button control is checked.
	Color (see page 21018)	Specifies the background color of the control.
	Constraints (see page 21019)	Specifies the size constraints for the control.
	Ctl3D (see page 21020)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 21020)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21020)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21021)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 21022)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 21023)	Controls the attributes of text written on or in the control.
	ParentBiDiMode (see page 21024)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 21024)	Determines where a control looks for its color information.
	ParentCtl3D (see page 21025)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 21025)	Determines where a control looks for its font information.
	ParentShowHint (see page 21026)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 21026)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 21031)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	State (see page 21032)	Indicates whether the check box is selected, deselected, or grayed.
	TabOrder (see page 21033)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21034)	Determines if the user can tab to a control.
	Visible (see page 21034)	Determines whether the component appears onscreen.
	WordWrap (see page 21035)	Specifies whether the button text wraps to fit the width of the control.

154.1.4.2.1 TCheckBox.Action Property

Designates the action associated with the control.

Pascal

```
property Action: TBasicAction;
```

C++

```
__property TBasicAction Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

154.1.4.2.2 TCheckBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
```

```
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
```

```
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

154.1.4.2.3 TCheckBox.Alignment Property

Determines whether the checkbox label aligns to the left or right of the tick box.

Pascal

```
property Alignment: TLeftRight;
```

C++

```
__property TLeftRight Alignment;
```

Description

The Alignment property is a little confusing. It implies text justification, but it is not quite that simple. The TLeftRight alignment type may have one of the following values :

Value	Meaning
taLeftJustify	The checkbox label is displayed on the left of the checkbox. The label is aligned to the left of the control, and the checkbox is aligned to the right of the control.
taRightJustify	This is the default value. The checkbox is aligned to the left of the control. The label is aligned to the right side of the checkbox : it is not aligned to the right of the control.

See Also

TLeftRight (see page 3297)

154.1.4.2.4 TCheckBox.AllowGrayed Property

Determines whether check box can be in a "grayed" state.

Pascal

```
property AllowGrayed: Boolean;
```

C++

```
__property Boolean AllowGrayed;
```

Description

If AllowGrayed is set to true, the check box has three possible states: checked, unchecked, and grayed. If AllowGrayed is set to false, the check box has only two possible states: checked and unchecked.

C++ Examples:

```
/*
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->AllowGrayed = true;
  CheckBox1->State = cbChecked;
}
```

Delphi Examples:

```
{
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  CheckBox1.AllowGrayed := True;
  CheckBox1.State := cbChecked;
end;
```

See Also

Checked

State (see page 21153)

154.1.4.2.5 TCheckBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

154.1.4.2.6 TCheckBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

154.1.4.2.7 TCheckBox.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}
```

```
TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
```

```
      begin
        with TToolButton.Create(ToolBar) do
        begin
          Parent := ToolBar;
          Caption := ButtonCaptions[I];
          if (ButtonCaptions[I] = '|') then
            Style := tbsSeparator
          else
            Style := tbsButton;
        end;
      end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ToolBar1 := TToolBar.Create(Self);
    ToolBar1.Parent := Self;
    AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
    ToolBar1.ShowCaptions := True;
    ToolBar1.Height := 40;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I: Integer;
  begin
    for I := ToolBar1.ButtonCount - 1 downto 0 do
      ToolBar1.Buttons[I].Free;
    ToolBar1.Free;
  end;
```

154.1.4.2.8 TCheckBox.Checked Property

Specifies whether the button control is checked.

Pascal

```
property Checked: Boolean;
```

C++

```
__property Boolean Checked;
```

Description

Use Checked to determine whether a button control is checked.

C++ Examples:

```
/*
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
*/

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->MultiLine = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->Caption = "Stack Tabs?";
```

```
  for (int i = 0; i < 20; i++)
  {
    TTabSheet *pPage = new TTabSheet(PageControl1);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
  }
}
```

Delphi Examples:

```
{
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.MultiLine := CheckBox1.Checked;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Checkbox1.Caption := 'Stack Tabs?';
  for i := 0 to 20 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'Tab #' + IntToStr(i);
    end;
end;
```

See Also

WndProc (see page 5791)

154.1.4.2.9 TCheckBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
```

```
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

154.1.4.2.10 TCheckBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

154.1.4.2.11 TCheckBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

154.1.4.2.12 TCheckBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

154.1.4.2.13 TCheckBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

154.1.4.2.14 TCheckBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

154.1.4.2.15 TCheckBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls

appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

154.1.4.2.16 TCheckBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
```

```
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

154.1.4.2.17 TCheckBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

154.1.4.2.18 TCheckBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

154.1.4.2.19 TCheckBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

154.1.4.2.20 TCheckBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

154.1.4.2.21 TCheckBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

154.1.4.2.22 TCheckBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
```

```
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
```

```
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
```

```
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
```

```
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
```

```
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

154.1.4.2.23 TCheckBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
```

```
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

154.1.4.2.24 TCheckBox.State Property

Indicates whether the check box is selected, deselected, or grayed.

Pascal

```
property State: TCheckBoxState;
```

C++

```
__property TCheckBoxState State;
```

Description

Use State to determine whether the check box is checked (cbChecked), unchecked (cbUnchecked), or grayed (cbGrayed).

C++ Examples:

```
/*
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  CheckBox1->AllowGrayed = true;
  CheckBox1->State = cbChecked;
}
```

Delphi Examples:

```
{
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  CheckBox1.AllowGrayed := True;
  CheckBox1.State := cbChecked;
end;
```

See Also

AllowGrayed (see page 21152)

Checked

154.1.4.2.25 TCheckBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

154.1.4.2.26 TCheckBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

154.1.4.2.27 TCheckBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
```

```
    {
      TTabSheet *pPage = new TTabSheet(this);
      pPage->PageControl = PageControl1;
      pPage->Caption = AnsiString("Page") + IntToStr(i);
      TEdit *pEdit = new TEdit(this);
      pEdit->Parent = pPage;
      pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
      pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
    }
    PageControl1Change(Sender);
  }
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

154.1.4.2.28 TCheckBox.WordWrap Property

Specifies whether the button text wraps to fit the width of the control.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the display of multiple lines of text. When WordWrap is true, text that is too wide for the control wraps at the right margin.

Set WordWrap to false to limit the text to a single line. When WordWrap is false, text that is too wide for the control appears truncated.

See Also

ClientWidth (see page 5811)

154.1.4.3 TCheckBox Events

The events of the TCheckBox class are listed here.

Published Events

	Name	Description
	OnClick (see page 21036)	Occurs when the user clicks the control.
	OnContextPopup (see page 21038)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 21039)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21041)	Occurs when the user drags an object over a control.
	OnEndDock (see page 21044)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21044)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 21045)	Occurs when a control receives the input focus.
	OnExit (see page 21048)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 21049)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 21052)	Occurs when a key is pressed.
	OnKeyUp (see page 21053)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 21055)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 21056)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 21058)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21058)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 21058)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 21061)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 21063)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21064)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

154.1.4.3.1 TCheckBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
```

```
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

154.1.4.3.2 TCheckBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

154.1.4.3.3 TCheckBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
```

```
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
```

```
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

154.1.4.3.4 TCheckBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor

property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
```

```
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.4.3.5 TCheckBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.4.3.6 TCheckBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
```

```
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

154.1.4.3.7 TCheckBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter

event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
```

```
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

154.1.4.3.8 TCheckBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

154.1.4.3.9 TCheckBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR

message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
```

```
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

154.1.4.3.10 TCheckBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
```

```
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

154.1.4.3.11 TCheckBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
```

```
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

154.1.4.3.12 TCheckBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.4.3.13 TCheckBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
```

```
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
```

```
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.4.3.14 TCheckBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

154.1.4.3.15 TCheckBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

154.1.4.3.16 TCheckBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
```

```
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.4.3.17 TCheckBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
```

```
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
```

```
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

154.1.4.3.18 TCheckBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

154.1.4.3.19 TCheckBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

154.1.5 TComboBox Class

TComboBox combines an edit box with a scrollable list.

Class Hierarchy

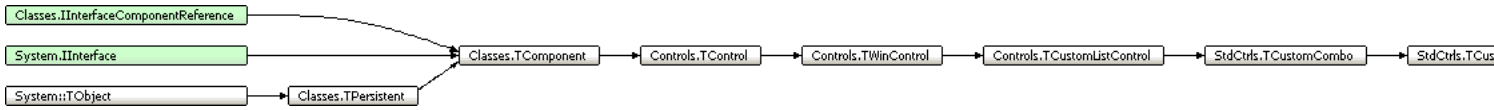

Pascal

```
TComboBox = class(TCustomComboBox);
```

C++

```
class TComboBox : public TCustomComboBox;
```

File

StdCtrls

Description

A TComboBox component is an edit box with a scrollable drop-down list attached to it. Users can select an item from the list or type directly into the edit box.

Note: The width of the button in a TComboBox is equal to the width that Windows uses for scrollbars. This width depends on the color scheme that the user has chosen (by right-clicking the desktop, selecting Properties, and choosing Appearance). On Windows XP, the user can specifically change the width of the scrollbar by clicking the Advanced button and selecting scrollbar as the Item. If you carefully size your combo box so all the text is visible, then a user running under a different color scheme may find that the text is obscured by the button. One solution is to set the scrollbar wide when developing the application. Most color schemes use a scrollbar that is at most 21 pixels wide.

See Also

TCustomComboBox (see page 21201)

TEdit (see page 21402)

TListBox (see page 21598)

Combo Boxes

List Controls

154.1.5.1 TComboBox Members

The following tables list the members exposed by TComboBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefEditProc (see page 21167)	This is FDefEditProc, a member of class TCustomCombo.
	FDefListProc (see page 21167)	This is FDefListProc, a member of class TCustomCombo.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FDropHandle (see page 21168)	This is FDropHandle, a member of class TCustomCombo.
	FDroppingDown (see page 21168)	This is FDroppingDown, a member of class TCustomCombo.
	FEditHandle (see page 21168)	This is FEditHandle, a member of class TCustomCombo.
	FEditInstance (see page 21168)	This is FEditInstance, a member of class TCustomCombo.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFocusChanged (see page 21168)	This is FFocusChanged, a member of class TCustomCombo.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FIsFocused (see page 21169)	This is FIsFocused, a member of class TCustomCombo.
	FListHandle (see page 21169)	This is FListHandle, a member of class TCustomCombo.
	FListInstance (see page 21169)	This is FListInstance, a member of class TCustomCombo.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveIndex (see page 21169)	This is FSaveIndex, a member of class TCustomCombo.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustDropDown (see page 21171)	Adjusts the size and position of the drop down list.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21171)	Generates an OnChange (see page 21197) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.

	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	CloseUp (see page 21172)	Generates an OnCloseUp (see page 21199) event.
V	ComboWndProc (see page 21173)	Responds to Windows messages sent to the edit control or drop-down list.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 21215)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CreateWnd (see page 21174)	Creates the Windows control that represents the combo box.
V	CreateWnd (see page 21215)	Creates the Windows control that represents the combo box.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWindowHandle (see page 21174)	Destroys all of the windows controls that implement the combo box.
V	DestroyWnd (see page 6141)	Destroys the control's window.
V	DestroyWnd (see page 21215)	Destroys the Windows control that represents the combo box.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.

	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawItem (see page 21216)	Generates an OnDrawItem (see page 21224) event.
	DropDown (see page 21175)	Generates an OnDropDown (see page 21199) event.
V	EditWndProc (see page 21175)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetDroppedDown (see page 21176)	Returns the value of the DroppedDown (see page 21185) property.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
A	GetItemCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
V	GetItemCount (see page 21216)	Returns the number of items in the Items (see page 21187) property list.
A	GetItemHt (see page 21177)	Returns the value of the ItemHeight (see page 21186) property.
V	GetItemHt (see page 21216)	Returns the value of the ItemHeight (see page 21186) property.
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
V	GetItemIndex (see page 21177)	Returns the value of the ItemIndex property.
A	GetItemsClass (see page 21177)	Indicates the class that implements the Items (see page 21187) property.
V	GetItemsClass (see page 21217)	Returns the class used to implement the Items (see page 21187) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.

	GetSelLength (see page 21178)	Returns the value of the SelLength (see page 21190) property.
	GetSelStart (see page 21178)	Returns the value of the SelStart (see page 21193) property.
	GetSelText (see page 21217)	Returns the value of the SelText (see page 21222) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
V	KeyPress (see page 21217)	Responds to WM_CHAR messages sent to the combo box.
	KeyUp (see page 6164)	Respond to released key.
	ListWndProc (see page 21178)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 21179)	Initializes the combo box after its form is loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
V	MeasureItem (see page 21218)	Generates an OnMeasureItem (see page 21225) event.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.

	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Select (see page 21179)	Generates an OnSelect (see page 21201) event.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectItem (see page 21218)	Selects a specified item from the drop-down list.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetDropDownCount (see page 21180)	Sets the value of the DropDownCount (see page 21185) property.
	SetDroppedDown (see page 21180)	Sets the value of the DroppedDown (see page 21185) property.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetItemHeight (see page 21181)	Sets the value of the ItemHeight (see page 21186) property.
A	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
V	SetItemIndex (see page 21181)	Sets the value of the ItemIndex property.
V	SetItems (see page 21181)	Sets the value of the Items (see page 21187) property.
	SetMaxLength (see page 21182)	Sets the value of the MaxLength (see page 21189) property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelLength (see page 21182)	Sets the value of the SelLength (see page 21190) property.
	SetSelStart (see page 21182)	Sets the value of the SelStart (see page 21193) property.
V	SetStyle (see page 21219)	Sets the Style (see page 21224) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.

V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WndProc (see page 21183)	Processes messages received by the combo box.
V	WndProc (see page 21219)	Processes messages received by the combo box.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
R	EditHandle (see page 21185)	Returns the window handle for the edit portion of the combo box.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	ItemCount (see page 21186)	Specifies the number of items in the drop-down list.
R	ListHandle (see page 21189)	Provides access to the window handle of the list of items in the combo box.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
A	AddItem (see page 5940)	Adds an item to the list control.
V	AddItem (see page 21170)	Adds an item to the drop-down list of items.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.

	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 5940)	Deletes all of the items from the list control.
V	Clear (see page 21172)	Deletes all text from the edit box and all items from the Items (see page 21187) list.
A	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
V	ClearSelection (see page 21172)	Deselects any selected items in the drop-down list.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
A	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
V	CopySelection (see page 21173)	Copies the selected item in the drop-down list to another list control.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 21166)	Creates and initializes a TCustomCombo (see page 21154) instance.
V	Create (see page 21214)	Creates and initializes a TCustomComboBox (see page 21201) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	DeleteSelected (see page 5941)	Deletes all the selected items.
V	DeleteSelected (see page 21174)	Deletes the selected item from the drop-down list.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 21166)	Destroys the combo box and frees its memory.
V	Destroy (see page 21214)	Destroys the combo box and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.

	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
V	Focused (see page 21175)	Indicates whether the combo box has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.

	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
A	SelectAll (see page 5943)	Selects all items or all text in the selected item.
V	SelectAll (see page 21180)	Selects the entire block of text in the edit box.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 21184)	Provides access to the drawing surface of the combo box.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.

R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DroppedDown (see page 21185)	Indicates whether the drop-down list is currently displayed.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelLength (see page 21190)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 21193)	Specifies the position of the first selected character in the edit portion of the combo box.
	SelText (see page 21222)	Represents the selected text in the edit region.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 21112)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 21114)	Occurs when the user clicks the control.
	OnCloseUp (see page 21116)	Occurs when the drop-down list closes up due to some user action.

	OnContextPopup (see page 21116)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 21117)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 21117)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21120)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 21122)	Occurs when an item in an owner-draw combo box needs to be displayed.
	OnDropDown (see page 21123)	Occurs when the user opens the drop-down list.
	OnEndDock (see page 21124)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21124)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 21125)	Occurs when a control receives the input focus.
	OnExit (see page 21128)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 21129)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 21132)	Occurs when a key is pressed.
	OnKeyUp (see page 21133)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 21135)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.
	OnMouseEnter (see page 21135)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21136)	Occurs when the user moves the mouse outside of a control.
	OnSelect (see page 21136)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 21136)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21137)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 21078)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 21082)	Specifies how the control is anchored to its parent.
	AutoCloseUp (see page 21082)	Specifies whether the drop-down closes up automatically when the user selects an item.
	AutoComplete (see page 21083)	Positions to matching list items as you type.
	AutoCompleteDelay (see page 21083)	Specifies the delay between a key press and an attempt to autocomplete the field.
	AutoDropDown (see page 21083)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 21084)	Specifies which edges of the control are beveled.
	BevelInner (see page 21084)	Specifies the cut of the inner bevel.
	BevelKind (see page 21085)	Specifies the control's bevel style.
	BevelOuter (see page 21085)	Specifies the cut of the outer bevel.
	BiDiMode (see page 21085)	Specifies the bi-directional mode for the control.
	CharCase (see page 21086)	Determines the case of the text in the combo box.
	Color (see page 21086)	Specifies the background color of the control.
	Constraints (see page 21087)	Specifies the size constraints for the control.
	Ctl3D (see page 21088)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 21088)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21089)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21089)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownCount (see page 21091)	Specifies the maximum number of items displayed in the drop-down list.
	Enabled (see page 21091)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 21091)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic

	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 21092)	Determines the behavior of the input method editor (IME).
	ImeName (see page 21093)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	ItemHeight (see page 21094)	Specifies the height, in pixels, of the items in the drop-down list.
	ItemIndex (see page 21094)	Specifies the index of the selected item.
	Items (see page 21095)	Provides access to the list of items (strings) in the list portion of the combo box.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	MaxLength (see page 21097)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 21097)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 21098)	Determines where a control looks for its color information.
	ParentCtl3D (see page 21098)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 21099)	Determines where a control looks for its font information.
	ParentShowHint (see page 21099)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 21100)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 21104)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Sorted (see page 21106)	Determines whether the list portion of the combo box is alphabetized.
	Style (see page 21107)	Determines the display style of the combo box.
	TabOrder (see page 21108)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21108)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Text (see page 21109)	Contains a text string associated with the control.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 21110)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

154.1.5.2 TComboBox Properties

The properties of the TComboBox class are listed here.

Published Properties

	Name	Description
	Align (see page 21078)	Determines how the control aligns within its container (parent control).
	Anchors (see page 21082)	Specifies how the control is anchored to its parent.
	AutoCloseUp (see page 21082)	Specifies whether the drop-down closes up automatically when the user selects an item.
	AutoComplete (see page 21083)	Positions to matching list items as you type.
	AutoCompleteDelay (see page 21083)	Specifies the delay between a key press and an attempt to autocomplete the field.
	AutoDropDown (see page 21083)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 21084)	Specifies which edges of the control are beveled.
	BevelInner (see page 21084)	Specifies the cut of the inner bevel.
	BevelKind (see page 21085)	Specifies the control's bevel style.
	BevelOuter (see page 21085)	Specifies the cut of the outer bevel.
	BiDiMode (see page 21085)	Specifies the bi-directional mode for the control.

	CharCase (see page 21086)	Determines the case of the text in the combo box.
	Color (see page 21086)	Specifies the background color of the control.
	Constraints (see page 21087)	Specifies the size constraints for the control.
	Ctl3D (see page 21088)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 21088)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21089)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21089)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownCount (see page 21091)	Specifies the maximum number of items displayed in the drop-down list.
	Enabled (see page 21091)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 21091)	Controls the attributes of text written on or in the control.
	ImeMode (see page 21092)	Determines the behavior of the input method editor (IME).
	ImeName (see page 21093)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	ItemHeight (see page 21094)	Specifies the height, in pixels, of the items in the drop-down list.
	ItemIndex (see page 21094)	Specifies the index of the selected item.
	Items (see page 21095)	Provides access to the list of items (strings) in the list portion of the combo box.
	MaxLength (see page 21097)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	ParentBiDiMode (see page 21097)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 21098)	Determines where a control looks for its color information.
	ParentCtl3D (see page 21098)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 21099)	Determines where a control looks for its font information.
	ParentShowHint (see page 21099)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 21100)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 21104)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Sorted (see page 21106)	Determines whether the list portion of the combo box is alphabetized.
	Style (see page 21107)	Determines the display style of the combo box.
	TabOrder (see page 21108)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21108)	Determines if the user can tab to a control.
	Text (see page 21109)	Contains a text string associated with the control.
	Visible (see page 21110)	Determines whether the component appears onscreen.

154.1.5.2.1 TComboBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
```

```
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
```

```
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
```

```
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

154.1.5.2.2 TComboBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

154.1.5.2.3 TComboBox.AutoCloseUp Property

Specifies whether the drop-down closes up automatically when the user selects an item.

Pascal

```
property AutoCloseUp: Boolean;
```

C++

```
__property Boolean AutoCloseUp;
```

Description

When AutoCloseUp is true, the drop-down closes up automatically when the user selects an item.

See Also

AutoDropDown (see page 21221)

154.1.5.2.4 TComboBox.AutoComplete Property

Positions to matching list items as you type.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

AutoComplete will position to matching list values as you type, sometimes saving you keystrokes.

As you type, the ItemIndex (see page 21094) property is updated with the list entry that matches the characters you have entered. As you type, different list enrties may match, and ItemIndex (see page 21094) is changed accordingly.

There is a delay of AutoCompleteDelay (see page 21083) milliseconds between the characters typed and the autocomplete operation.

See Also

AutoCompleteDelay (see page 21083)

154.1.5.2.5 TComboBox.AutoCompleteDelay Property

Specifies the delay between a key press and an attempt to autocomplete the field.

Pascal

```
property AutoCompleteDelay: Cardinal;
```

C++

```
__property unsigned AutoCompleteDelay;
```

Description

Specifies the delay in milliseconds between a key press and an attempt to autocomplete the field. Autocomplete will position to matching list values as you type, sometimes saving you keystrokes.

AutoCompleteDelay is only in effect when the AutoComplete (see page 21083) is true.

See Also

AutoComplete (see page 21083)

154.1.5.2.6 TComboBox.AutoDropDown Property

Specifies whether the drop-down list drops down automatically in response to user keystrokes.

Pascal

```
property AutoDropDown: Boolean;
```

C++

```
__property Boolean AutoDropDown;
```

Description

When AutoDropDown is true, the combo box automatically drops down its list when the user starts typing a string while the combo box has focus.

When AutoDropDown is false, the user must explicitly use the drop-down button to drop down the combo box list.

See Also

autocomplete (see page 21220)

OnDropDown (see page 21199)

DroppedDown (see page 21185)

154.1.5.2.7 TComboBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

154.1.5.2.8 TComboBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

154.1.5.2.9 TComboBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

154.1.5.2.10 TComboBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

154.1.5.2.11 TComboBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

154.1.5.2.12 TComboBox.CharCase Property

Determines the case of the text in the combo box.

Pascal

```
property CharCase: TEditCharCase;
```

C++

```
__property TEditCharCase CharCase;
```

Description

Use CharCase to force the contents of the combo box into upper or lower case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types into the combo box. Changing the CharCase property changes the actual text, not just its appearance; case information is lost and cannot be recaptured by changing CharCase back to ecNormal.

See Also

Text (see page 5842)

154.1.5.2.13 TComboBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the

Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

154.1.5.2.14 TComboBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

154.1.5.2.15 TComboBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

154.1.5.2.16 TComboBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

154.1.5.2.17 TComboBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

154.1.5.2.18 TComboBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
```

```
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

154.1.5.2.19 TComboBox.DropDownCount Property

Specifies the maximum number of items displayed in the drop-down list.

Pascal

```
property DropDownCount: Integer;
```

C++

```
__property int DropDownCount;
```

Description

By default, the drop-down list is long enough to contain eight items without requiring the user to scroll to see them all. To make the drop-down list smaller or larger, specify a number larger or smaller than eight as the DropDownCount value.

If the DropDownCount value is larger than the number of items in the Items property, the drop-down list will be just large enough to hold all the possible choices and no more. If the DropDownCount value is smaller then the number of item in the Items property, the drop down list will display a scroll bar.

See Also

Items (see page 21187)

ItemCount (see page 21186)

154.1.5.2.20 TComboBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

154.1.5.2.21 TComboBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

154.1.5.2.22 TComboBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

154.1.5.2.23 TComboBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

154.1.5.2.24 TComboBox.ItemHeight Property

Specifies the height, in pixels, of the items in the drop-down list.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Read ItemHeight to determine the height of the items in the drop-down list.

The effect of setting ItemHeight varies, depending on the type of combo box and the style. For example, TComboBox lets you set the item height only when Style is set to csOwnerDrawFixed or csOwnerDrawVariable. If Style is csOwnerDrawVariable, ItemHeight is the default height for drawing list items, but this value can be overridden by an OnMeasureItem event handler.

TComboBoxEx does not let you change the value of ItemHeight.

Note: Combo boxes that do not support changes to ItemHeight do not raise an exception when you attempt to change ItemHeight; they simply ignore the attempt.

See Also

Items (see page 21187)

OnMeasureItem (see page 21225)

Style (see page 21224)

154.1.5.2.25 TComboBox.ItemIndex Property

Specifies the index of the selected item.

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

Description

Read ItemIndex to determine which item is selected. The first item in the list has index 0, the second item has index 1, and so on. If no item is selected, the value of ItemIndex is -1. If the list control supports multiple selected items, ItemIndex is the index of the selected item that has focus.

Set ItemIndex programmatically to select an item by passing in the index value.

Note: To implement the ItemIndex property in a TCustomListControl descendant, override the protected GetItemIndex and SetItemIndex methods.

See Also

GetItemIndex (see page 5942)

SetItemIndex (see page 5943)

154.1.5.2.26 TComboBox.Items Property

Provides access to the list of items (strings) in the list portion of the combo box.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Read Items to access the list of items that appears in the combo box. Use the methods of Items to add, insert, delete and move items. Set the value of Items to copy the items from another string list.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", (TObject *)vsIcon);
```

```
  ComboBox1->Items->AddObject("vsList", (TObject *)vsList);
  ComboBox1->Items->AddObject("vsReport", (TObject *)vsReport);
  ComboBox1->Items->AddObject("vsSmallIcon", (TObject *)vsSmallIcon);
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := 'Image' + IntToStr(I);
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
    NewColumn.Width := 200;
    // Add View styles and constants to the Combo Box
    ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
    ComboBox1.Items.AddObject('vsList', TObject(vsList));
    ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
    ComboBox1.Items.AddObject(
      'vsSmallIcon', TObject(vsSmallIcon));
    // Display first item in the Combo Box
    ComboBox1.ItemIndex := 0;
```

```
    end;
  end;

  procedure TComboForm.ComboBox1Click(Sender: TObject);
  begin
    with ComboBox1 do
      ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

  end;
```

See Also

ItemIndex

TStrings (see page 3029)

Assign (see page 3043)

154.1.5.2.27 TComboBox.MaxLength Property

Specifies the maximum number of characters the user can type into the edit portion of the combo box.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that the user can enter into the edit portion of the combo box. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

See Also

Text (see page 5842)

154.1.5.2.28 TComboBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

154.1.5.2.29 TComboBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

154.1.5.2.30 TComboBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

154.1.5.2.31 TComboBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

154.1.5.2.32 TComboBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

154.1.5.2.33 TComboBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
```

```
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
```

```
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
```

```
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

154.1.5.2.34 TComboBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

154.1.5.2.35 TComboBox.Sorted Property

Determines whether the list portion of the combo box is alphabetized.

Pascal

```
property Sorted: Boolean;
```

C++

```
__property Boolean Sorted;
```

Description

Set Sorted to true to sort the items in the Items list alphabetically. New items added to the list while Sorted is true are inserted in the correct alphabetical position.

When Sorted is changed from false to true, the original order of the items is lost. Setting Sorted back to false does not restore the original order.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if (((TForm1 *)Screen->CustomForms[i])->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
```

```
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
    end;
    Items.EndUpdate; {reenable painting }
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.Show;
end;
```

See Also

Items

Using TDBListBox and TDBComboBox

154.1.5.2.36 TComboBox.Style Property

Determines the display style of the combo box.

Pascal

```
property Style: TComboBoxStyle;
```

C++

```
__property TComboBoxStyle Style;
```

Description

Set Style to specify how the list and edit region of the combo box should appear.

When Style is csOwnerDrawFixed or csOwnerDrawVariable, the OnDrawItem event occurs when the list items need to be drawn. If Style is csOwnerDrawVariable, an OnMeasureItem event occurs as well.

See Also

Height (see page 5825)

ItemHeight

OnDrawItem (see page 21224)

OnMeasureItem (see page 21225)

154.1.5.2.37 TComboBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

154.1.5.2.38 TComboBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

154.1.5.2.39 TComboBox.Text Property

Contains a text string associated with the control.

Pascal

```
property Text: TCaption;
```

C++

```
__property TCaption Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
  Size++;                        //Add room for null character
  char *Buffer = new char[Size];  //Creates Buffer dynamic variable
  Edit1->GetTextBuf(Buffer,Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer;
  delete [] Buffer;
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
```

```
    Inc(Size);                      {Add room for null character}
    GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
    Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
    Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
    FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
  end;

  {
  Note: the same thing could be accomplished more simply as follows:
  }
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    Edit2.Text := Edit1.Text;
  end;
```

See Also

Caption (see page 5806)

Lines (see page 21380)

154.1.5.2.40 TComboBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
```

```
      pEdit->Parent = pPage;
      pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
      pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
    }
    PageControl1Change(Sender);
  }
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

154.1.5.3 TComboBox Events

The events of the TComboBox class are listed here.

Published Events

	Name	Description
	OnChange (see page 21112)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 21114)	Occurs when the user clicks the control.
	OnCloseUp (see page 21116)	Occurs when the drop-down list closes up due to some user action.

	OnContextPopup (see page 21116)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 21117)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 21117)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21120)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 21122)	Occurs when an item in an owner-draw combo box needs to be displayed.
	OnDropDown (see page 21123)	Occurs when the user opens the drop-down list.
	OnEndDock (see page 21124)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21124)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 21125)	Occurs when a control receives the input focus.
	OnExit (see page 21128)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 21129)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 21132)	Occurs when a key is pressed.
	OnKeyUp (see page 21133)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 21135)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.
	OnMouseEnter (see page 21135)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21136)	Occurs when the user moves the mouse outside of a control.
	OnSelect (see page 21136)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 21136)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21137)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

154.1.5.3.1 TComboBox.OnChange Event

Occurs when the user changes the text displayed in the edit region.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action immediately after the user edits the text in the edit region or selects an item from the list. The Text property gives the new value in the edit region.

Note: OnChange only occurs in response to user actions. Changing the Text property programmatically does not trigger an OnChange event.

C++ Examples:

```
/*
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
```

```
justifying.
*/
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  if (pCB)
    PageControl1->ActivePage = (TTabSheet *)(pCB->Items->Objects[pCB->ItemIndex]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
    ComboBox1->Items->AddObject(PageControl1->Pages[i]->Name, PageControl1->Pages[i]);
    ComboBox1->ItemIndex = 0;
  }
}
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to PageControl1.PageCount - 1 do
    ComboBox1.Items.AddObject(PageControl1.Pages[i].Name,
```

```
      PageControl1.Pages[i]);
    ComboBox1.ItemIndex := 0;
  end;

  procedure TForm1.ComboBox1Change(Sender: TObject);
  begin
    if (Sender is TComboBox) then
      with (Sender as TComboBox) do
        PageControl1.ActivePage := TTabSheet(Items.Objects[ItemIndex]);
  end;
  {
  The following example assumes that a drive combo box, a file
  list box, and a directory list box are on a form. Add this
  code as the OnChange event handler for the drive combo box
  and the OnChange event handler for the directory list box.
  When the user changes the drive using the combo box, the
  directory list box and file list box will update to reflect
  the new drive and the current directory on that drive.  When
  the user double clicks on a directory in the file list box
  will update to reflect the new directory.
  }
  procedure TForm1.DirectoryListBox1Change(Sender: TObject);
  begin
    FileListBox1.Directory := DirectoryListBox1.Directory;
  end;

  procedure TForm1.DriveComboBox1Change(Sender: TObject);
  begin
    DirectoryListBox1.Drive := DriveComboBox1.Drive;
    FileListBox1.Drive := DriveComboBox1.Drive;
    FileListBox1.Directory := DirectoryListBox1.Directory;
  end;
```

See Also

Text (see page 5842)

OnCloseUp (see page 21199)

154.1.5.3.2 TComboBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).

- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
```

```
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

154.1.5.3.3 TComboBox.OnCloseUp Event

Occurs when the drop-down list closes up due to some user action.

Pascal

```
property OnCloseUp: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCloseUp;
```

Description

Write an OnCloseUp event handler to implement special processing that needs to occur when the drop-down list closes up. For example, an OnCloseUp event handler can check whether the user changed the selected item while the list was dropped down and respond accordingly.

See Also

OnDropDown (see page 21199)

DroppedDown (see page 21185)

154.1.5.3.4 TComboBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

154.1.5.3.5 TComboBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

154.1.5.3.6 TComboBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

154.1.5.3.7 TComboBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.
```

```
procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.5.3.8 TComboBox.OnDrawItem Event

Occurs when an item in an owner-draw combo box needs to be displayed.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

Write an OnDrawItem event handler to draw the items in the drop-down list of an owner-draw combo box. An OnDrawItem event handler can add graphic elements to the list items, or replace the list item text by graphics.

Draw the items on the Canvas using the coordinates provided by the Rect parameter. OnDrawItem occurs only if Style is set to csOwnerDrawFixed or csOwnerDrawVariable.

If an OnDrawItem event handler is not provided, the combo box fills the Rect parameter with the current brush and writes the text value of the item specified by the Index parameter.

See Also

Canvas

Items

OnMeasureItem (see page 21225)

Style (see page 21224)

154.1.5.3.9 TComboBox.OnDropDown Event

Occurs when the user opens the drop-down list.

Pascal

```
property OnDropDown: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDropDown;
```

Description

Write an OnDropDown event handler to implement special processing that needs to occur only when the drop-down list is activated.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if (((TForm1 *)Screen->CustomForms[i])->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
```

```
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
    end;
    Items.EndUpdate; {reenable painting }
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.Show;
end;
```

See Also

DroppedDown (see page 21185)

OnCloseUp (see page 21199)

154.1.5.3.10 TComboBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.5.3.11 TComboBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

154.1.5.3.12 TComboBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
```

```
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
```

```
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

154.1.5.3.13 TComboBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
```

```
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

154.1.5.3.14 TComboBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
```

```
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
```

```
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

154.1.5.3.15 TComboBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an

OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

154.1.5.3.16 TComboBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
```

```
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

154.1.5.3.17 TComboBox.OnMeasureItem Event

Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.

Pascal

```
property OnMeasureItem: TMeasureItemEvent;
```

C++

```
__property TMeasureItemEvent OnMeasureItem;
```

Description

When Style is set to csOwnerDrawVariable, the OnMeasureItem event precedes OnDrawItem. Write an OnMeasureItem event handler to specify the height, in pixels, needed to draw an item in the drop-down list.

See Also

ItemHeight

Items

OnDrawItem (see page 21224)

Style (see page 21224)

154.1.5.3.18 TComboBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

154.1.5.3.19 TComboBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

154.1.5.3.20 TComboBox.OnSelect Event

Occurs when the user selects a string in the drop-down list.

Pascal

```
property OnSelect: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSelect;
```

Description

Write an OnSelect event handler to respond when the user changes the selected item in the drop-down list. OnSelect occurs after the selection has changed and the Text property reflects the new item.

See Also

autocomplete (see page 21220)

ItemIndex

Text (see page 5842)

154.1.5.3.21 TComboBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by

dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

154.1.5.3.22 TComboBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

154.1.6 TCustomCheckBox Class

TCustomCheckBox is the ancestor of all check-box components.

Class Hierarchy

Pascal

```
TCustomCheckBox = class(TButtonControl);
```

C++

```
class TCustomCheckBox : public TButtonControl;
```

File

StdCtrls

Description

TCustomCheckBox is an abstract class from which check-box components, including TCheckBox (see page 20996) and TDBCheckBox, descend. Check boxes present the user with options that can be selected (checked) or deselected (unchecked).

See Also

TCheckBox (see page 20996)

TDBCheckBox (see page 8582)

154.1.6.1 TCustomCheckBox Members

The following tables list the members exposed by TCustomCheckBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
V	ActionChange (see page 20992)	Updates the button control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.

	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 21149)	Click generates an OnClick event.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 21149)	Initializes a window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 21150)	Creates a Windows control corresponding to the check box.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus

	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetActionLinkClass (see page 20993)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 21150)	Returns the value of the Checked (see page 20995) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).

	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
V	SetChecked (see page 21151)	Sets the value of the Checked (see page 20995) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	Toggle (see page 21151)	Changes the state of the check box.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WndProc (see page 20994)	Handles Windows messages.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	Alignment (see page 21152)	This is Alignment, a member of class TCustomCheckBox.
	AllowGrayed (see page 21152)	Determines whether check box can be in a "grayed" state.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.

	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Checked (see page 20995)	Specifies whether the button control is checked.
	ClicksDisabled (see page 20996)	This is ClicksDisabled, a member of class TButtonControl.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	State (see page 21153)	Indicates whether the check box is selected, deselected, or grayed.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.
	WordWrap (see page 20996)	Specifies whether the button text wraps to fit the width of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 21148)	Creates and initializes a TCustomCheckBox (see page 21138) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
V	GetControlsAlignment (see page 21151)	Returns the alignment of the check box caption.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.

	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	TabStop (see page 21154)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

154.1.6.2 TCustomCheckBox.Create Constructor

Creates and initializes a TCustomCheckBox (see page 21138) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomCheckBox(TComponent * AOwner);
```

Description

Call Create to generate a check box at runtime. Check boxes placed on forms at design time, Create is called automatically.

AOwner is a component, typically the form, that is responsible for freeing the check box instance.

See Also

Owner (see page 2807)

154.1.6.3 TCustomCheckBox Methods

The methods of the TCustomCheckBox class are listed here.

Protected Methods

	Name	Description
	Click (see page 21149)	Click generates an OnClick event.
	CreateParams (see page 21149)	Initializes a window-creation parameter record.
	CreateWnd (see page 21150)	Creates a Windows control corresponding to the check box.
	GetChecked (see page 21150)	Returns the value of the Checked (see page 20995) property.
	SetChecked (see page 21151)	Sets the value of the Checked (see page 20995) property.
	Toggle (see page 21151)	Changes the state of the check box.

Public Methods

	Name	Description
	GetControlsAlignment (see page 21151)	Returns the alignment of the check box caption.

154.1.6.3.1 TCustomCheckBox.Click Method

Click generates an OnClick event.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

The protected Click method is called automatically when the check box is clicked with the mouse. In addition to generating an OnClick event, it sends a message to the parent control's window to indicate that the state of the check box has changed.

See Also

OnClick (see page 5851)

Parent (see page 5832)

154.1.6.3.2 TCustomCheckBox.CreateParams Method

Initializes a window-creation parameter record.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 21150) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. CreateParams fills in the parameter record that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

In TCustomCheckBox (see page 21138), CreateParams calls the inherited method, then makes specific adjustments for check boxes.

See Also

CreateWnd (see page 6137)

CreateWindowHandle (see page 6137)

154.1.6.3.3 TCustomCheckBox.CreateWnd Method

Creates a Windows control corresponding to the check box.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the check box is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. In TCustomCheckBox (see page 21138), this method calls the inherited method to generate the underlying windows control, then sends a BM_SETCHECK message to set the check state of the control.

See Also

CreateParams (see page 21149)

154.1.6.3.4 TCustomCheckBox.GetChecked Method

Returns the value of the Checked (see page 20995) property.

Pascal

```
function GetChecked: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetChecked();
```

Description

GetChecked is the protected implementation of the Checked (see page 20995) property. Override this method and the SetChecked (see page 21151) method to change the Checked (see page 20995) property in descendant classes.

See Also

Checked

SetChecked (see page 21151)

154.1.6.3.5 TCustomCheckBox.GetControlsAlignment Method

Returns the alignment of the check box caption.

Pascal

```
function GetControlsAlignment: TAlignment; override;
```

C++

```
virtual __fastcall TAlignment GetControlsAlignment();
```

Description

GetControlsAlignment indicates the alignment of the check box caption. Usually, the caption is right-justified, but if the UseRightToLeftAlignment method returns true and the check mark is on the left, then GetControlsAlignment indicates that the caption should be left-justified.

See Also

Caption (see page 5806)

alignment (see page 21152)

UseRightToLeftAlignment (see page 5790)

154.1.6.3.6 TCustomCheckBox.SetChecked Method

Sets the value of the Checked (see page 20995) property.

Pascal

```
procedure SetChecked(Value: Boolean); override;
```

C++

```
virtual __fastcall SetChecked(Boolean Value);
```

Description

SetChecked is the protected implementation of the Checked (see page 20995) property. Override this method and the GetChecked (see page 21150) method to change the Checked (see page 20995) property in descendant classes.

See Also

Checked

GetChecked (see page 21150)

154.1.6.3.7 TCustomCheckBox.Toggle Method

Changes the state of the check box.

Pascal

```
procedure Toggle; virtual;
```

C++

```
virtual __fastcall Toggle();
```

Description

Use the Toggle method to select or deselect the check box programmatically. Toggle switches among the valid check-box

states—from checked to unchecked to grayed, or from checked to unchecked.

See Also

AllowGrayed (see page 21152)

State (see page 21153)

154.1.6.4 TCustomCheckBox Properties

The properties of the TCustomCheckBox class are listed here.

Protected Properties

	Name	Description
	Alignment (see page 21152)	This is Alignment, a member of class TCustomCheckBox.
	AllowGrayed (see page 21152)	Determines whether check box can be in a "grayed" state.
	State (see page 21153)	Indicates whether the check box is selected, deselected, or grayed.

Published Properties

	Name	Description
	TabStop (see page 21154)	Determines if the user can tab to a control.

154.1.6.4.1 TCustomCheckBox.Alignment Property

Pascal

```
property Alignment: TLeftRight;
```

C++

```
__property TLeftRight Alignment;
```

Description

This is Alignment, a member of class TCustomCheckBox.

154.1.6.4.2 TCustomCheckBox.AllowGrayed Property

Determines whether check box can be in a "grayed" state.

Pascal

```
property AllowGrayed: Boolean;
```

C++

```
__property Boolean AllowGrayed;
```

Description

If AllowGrayed is set to true, the check box has three possible states: checked, unchecked, and grayed. If AllowGrayed is set to false, the check box has only two possible states: checked and unchecked.

C++ Examples:

```
/*
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  CheckBox1->AllowGrayed = true;
  CheckBox1->State = cbChecked;
}
```

Delphi Examples:

```
{
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  CheckBox1.AllowGrayed := True;
  CheckBox1.State := cbChecked;
end;
```

See Also

Checked

State (see page 21153)

154.1.6.4.3 TCustomCheckBox.State Property

Indicates whether the check box is selected, deselected, or grayed.

Pascal

```
property State: TCheckBoxState;
```

C++

```
__property TCheckBoxState State;
```

Description

Use State to determine whether the check box is checked (cbChecked), unchecked (cbUnchecked), or grayed (cbGrayed).

C++ Examples:

```
/*
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->AllowGrayed = true;
  CheckBox1->State = cbChecked;
}
```

Delphi Examples:

```
{
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  CheckBox1.AllowGrayed := True;
  CheckBox1.State := cbChecked;
end;
```

See Also

AllowGrayed (see page 21152)

Checked

154.1.6.4.4 TCustomCheckBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

154.1.7 TCustomCombo Class

TCustomCombo is the base class for all combo boxes.

Class Hierarchy

Pascal

```
TCustomCombo = class(TCustomListControl);
```

C++

```
class TCustomCombo : public TCustomListControl;
```

File

StdCtrls

Description

Use TCustomCombo as a base class when defining a control that represents a specialized combo box. A combo box is an edit box with a scrollable list attached to it; users can select an item from the list or type directly into the edit box. Combo boxes can present the list as a drop-down list or as a list box that is always visible.

Typically, combo box classes do not descend directly from TCustomCombo. Rather, they descend from one of TCustomCombo's descendants, TCustomComboBox (see page 21201) or TCustomComboBoxEx. Use TCustomCombo as a base class for a

simple combo box, or TCustomComboBoxEx as a base class for a combo box that supports the extended combo box features such as images on the list entries.

Do not create instances of TCustomCombo. To put a combo box in a form, use a descendant of TCustomCombo, such as TComboBox (see page 21064) or TComboBoxEx.

See Also

TComboBox (see page 21064)

TComboBoxEx (see page 3463)

TDBComboBox (see page 8657)

TColorBox (see page 12182)

TDriveComboBox (see page 13396)

TFilterComboBox (see page 13547)

TCustomComboBox (see page 21201)

TCustomComboBoxEx (see page 3678)

154.1.7.1 TCustomCombo Members

The following tables list the members exposed by TCustomCombo.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnChange (see page 21197)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnCloseUp (see page 21199)	Occurs when the drop-down list closes up due to some user action.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDropDown (see page 21199)	Occurs when the user opens the drop-down list.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnSelect (see page 21201)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefEditProc (see page 21167)	This is FDefEditProc, a member of class TCustomCombo.
	FDefListProc (see page 21167)	This is FDefListProc, a member of class TCustomCombo.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FDropHandle (see page 21168)	This is FDropHandle, a member of class TCustomCombo.
	FDroppingDown (see page 21168)	This is FDroppingDown, a member of class TCustomCombo.
	FEditHandle (see page 21168)	This is FEditHandle, a member of class TCustomCombo.
	FEditInstance (see page 21168)	This is FEditInstance, a member of class TCustomCombo.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFocusChanged (see page 21168)	This is FFocusChanged, a member of class TCustomCombo.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FIsFocused (see page 21169)	This is FIsFocused, a member of class TCustomCombo.
	FListHandle (see page 21169)	This is FListHandle, a member of class TCustomCombo.
	FListInstance (see page 21169)	This is FListInstance, a member of class TCustomCombo.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveIndex (see page 21169)	This is FSaveIndex, a member of class TCustomCombo.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustDropDown (see page 21171)	Adjusts the size and position of the drop down list.

	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21171)	Generates an OnChange (see page 21197) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	CloseUp (see page 21172)	Generates an OnCloseUp (see page 21199) event.
	ComboWndProc (see page 21173)	Responds to Windows messages sent to the edit control or drop-down list.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 21174)	Creates the Windows control that represents the combo box.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 21174)	Destroys all of the windows controls that implement the combo box.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.

	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DropDown (see page 21175)	Generates an OnDropDown (see page 21199) event.
V	EditWndProc (see page 21175)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetDroppedDown (see page 21176)	Returns the value of the DroppedDown (see page 21185) property.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
A	GetItemCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
A	GetItemHt (see page 21177)	Returns the value of the ItemHeight (see page 21186) property.
V	GetItemIndex (see page 21177)	Returns the value of the ItemIndex property.

A	GetItemsClass (see page 21177)	Indicates the class that implements the Items (see page 21187) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelLength (see page 21178)	Returns the value of the SelLength (see page 21190) property.
	GetSelStart (see page 21178)	Returns the value of the SelStart (see page 21193) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	ListWndProc (see page 21178)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
V	Loaded (see page 21179)	Initializes the combo box after its form is loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.

	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Select (see page 21179)	Generates an OnSelect (see page 21201) event.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetDropDownCount (see page 21180)	Sets the value of the DropDownCount (see page 21185) property.
	SetDroppedDown (see page 21180)	Sets the value of the DroppedDown (see page 21185) property.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItemHeight (see page 21181)	Sets the value of the ItemHeight (see page 21186) property.
	SetItemIndex (see page 21181)	Sets the value of the ItemIndex property.
	SetItems (see page 21181)	Sets the value of the Items (see page 21187) property.
	SetMaxLength (see page 21182)	Sets the value of the MaxLength (see page 21189) property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelLength (see page 21182)	Sets the value of the SelLength (see page 21190) property.
	SetSelStart (see page 21182)	Sets the value of the SelStart (see page 21193) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.

V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 21183)	Processes messages received by the combo box.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownCount (see page 21185)	Specifies the maximum number of items displayed in the drop-down list.
R	EditHandle (see page 21185)	Returns the window handle for the edit portion of the combo box.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	ItemCount (see page 21186)	Specifies the number of items in the drop-down list.
	ItemHeight (see page 21186)	Specifies the height, in pixels, of the items in the drop-down list.
R	ListHandle (see page 21189)	Provides access to the window handle of the list of items in the combo box.
	MaxLength (see page 21189)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 21190)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.

	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AddItem (see page 21170)	Adds an item to the drop-down list of items.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 21172)	Deletes all text from the edit box and all items from the Items (see page 21187) list.
V	ClearSelection (see page 21172)	Deselects any selected items in the drop-down list.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	CopySelection (see page 21173)	Copies the selected item in the drop-down list to another list control.
V	Create (see page 21166)	Creates and initializes a TCustomCombo (see page 21154) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	DeleteSelected (see page 21174)	Deletes the selected item from the drop-down list.
V	Destroy (see page 21166)	Destroys the combo box and frees its memory.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.

	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
V	Focused (see page 21175)	Indicates whether the combo box has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
V	SelectAll (see page 21180)	Selects the entire block of text in the edit box.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.

R	Canvas (see page 21184)	Provides access to the drawing surface of the combo box.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DroppedDown (see page 21185)	Indicates whether the drop-down list is currently displayed.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	Items (see page 21187)	Provides access to the list of items (strings) in the list portion of the combo box.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelLength (see page 21190)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 21193)	Specifies the position of the first selected character in the edit portion of the combo box.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.

	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21196)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

154.1.7.2 TCustomCombo.Create Constructor

Creates and initializes a TCustomCombo (see page 21154) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomCombo(TComponent * AOwner);
```

Description

Call Create to generate a combo box at runtime. Combo boxes placed on forms at design time are created automatically.

AOwner is a component (typically a form) that is responsible for freeing the combo box instance.

See Also

Owner (see page 2807)

154.1.7.3 TCustomCombo.Destroy Destructor

Destroys the combo box and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomCombo();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free verifies that the combo box is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

154.1.7.4 TCustomCombo Fields

The fields of the TCustomCombo class are listed here.

Protected Fields

	Name	Description
	FDefEditProc (see page 21167)	This is FDefEditProc, a member of class TCustomCombo.
	FDefListProc (see page 21167)	This is FDefListProc, a member of class TCustomCombo.
	FDropHandle (see page 21168)	This is FDropHandle, a member of class TCustomCombo.
	FDroppingDown (see page 21168)	This is FDroppingDown, a member of class TCustomCombo.
	FEditHandle (see page 21168)	This is FEditHandle, a member of class TCustomCombo.
	FEditInstance (see page 21168)	This is FEditInstance, a member of class TCustomCombo.
	FFocusChanged (see page 21168)	This is FFocusChanged, a member of class TCustomCombo.
	FIsFocused (see page 21169)	This is FIsFocused, a member of class TCustomCombo.
	FListHandle (see page 21169)	This is FListHandle, a member of class TCustomCombo.
	FListInstance (see page 21169)	This is FListInstance, a member of class TCustomCombo.
	FSaveIndex (see page 21169)	This is FSaveIndex, a member of class TCustomCombo.

154.1.7.4.1 TCustomCombo.FDefEditProc Field

Pascal

```
FDefEditProc: Pointer;
```

C++

```
void * FDefEditProc;
```

Description

This is FDefEditProc, a member of class TCustomCombo.

154.1.7.4.2 TCustomCombo.FDefListProc Field

Pascal

```
FDefListProc: Pointer;
```

C++

```
void * FDefListProc;
```

Description

This is FDefListProc, a member of class TCustomCombo.

154.1.7.4.3 TCustomCombo.FDropHandle Field

Pascal

```
FDropHandle: HWnd;
```

C++

```
HWnd FDropHandle;
```

Description

This is FDropHandle, a member of class TCustomCombo.

154.1.7.4.4 TCustomCombo.FDroppingDown Field

Pascal

```
FDroppingDown: Boolean;
```

C++

```
Boolean FDroppingDown;
```

Description

This is FDroppingDown, a member of class TCustomCombo.

154.1.7.4.5 TCustomCombo.FEditHandle Field

Pascal

```
FEditHandle: HWnd;
```

C++

```
HWnd FEditHandle;
```

Description

This is FEditHandle, a member of class TCustomCombo.

154.1.7.4.6 TCustomCombo.FEditInstance Field

Pascal

```
FEditInstance: Pointer;
```

C++

```
void * FEditInstance;
```

Description

This is FEditInstance, a member of class TCustomCombo.

154.1.7.4.7 TCustomCombo.FFocusChanged Field

Pascal

```
FFocusChanged: Boolean;
```

C++

```
Boolean FFocusChanged;
```

Description

This is FFocusChanged, a member of class TCustomCombo.

154.1.7.4.8 TCustomCombo.FIsFocused Field

Pascal

```
FIsFocused: Boolean;
```

C++

```
Boolean FIsFocused;
```

Description

This is FIsFocused, a member of class TCustomCombo.

154.1.7.4.9 TCustomCombo.FListHandle Field

Pascal

```
FListHandle: HWnd;
```

C++

```
HWnd FListHandle;
```

Description

This is FListHandle, a member of class TCustomCombo.

154.1.7.4.10 TCustomCombo.FListInstance Field

Pascal

```
FListInstance: Pointer;
```

C++

```
void * FListInstance;
```

Description

This is FListInstance, a member of class TCustomCombo.

154.1.7.4.11 TCustomCombo.FSaveIndex Field

Pascal

```
FSaveIndex: Integer;
```

C++

```
int FSaveIndex;
```

Description

This is FSaveIndex, a member of class TCustomCombo.

154.1.7.5 TCustomCombo Methods

The methods of the TCustomCombo class are listed here.

Protected Methods

	Name	Description
V	AdjustDropDown (see page 21171)	Adjusts the size and position of the drop down list.
	Change (see page 21171)	Generates an OnChange (see page 21197) event.
	CloseUp (see page 21172)	Generates an OnCloseUp (see page 21199) event.
V	ComboWndProc (see page 21173)	Responds to Windows messages sent to the edit control or drop-down list.
V	CreateWnd (see page 21174)	Creates the Windows control that represents the combo box.
V	DestroyWindowHandle (see page 21174)	Destroys all of the windows controls that implement the combo box.
	DropDown (see page 21175)	Generates an OnDropDown (see page 21199) event.
V	EditWndProc (see page 21175)	Responds to Windows messages that are sent to the edit control portion of the combo box.
V	GetCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
	GetDroppedDown (see page 21176)	Returns the value of the DroppedDown (see page 21185) property.
A	GetItemCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
A	GetItemHt (see page 21177)	Returns the value of the ItemHeight (see page 21186) property.
V	GetItemIndex (see page 21177)	Returns the value of the ItemIndex property.
A	GetItemsClass (see page 21177)	Indicates the class that implements the Items (see page 21187) property.
	GetSelLength (see page 21178)	Returns the value of the SelLength (see page 21190) property.
	GetSelStart (see page 21178)	Returns the value of the SelStart (see page 21193) property.
	ListWndProc (see page 21178)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
V	Loaded (see page 21179)	Initializes the combo box after its form is loaded into memory.
	Select (see page 21179)	Generates an OnSelect (see page 21201) event.
V	SetDropDownCount (see page 21180)	Sets the value of the DropDownCount (see page 21185) property.
	SetDroppedDown (see page 21180)	Sets the value of the DroppedDown (see page 21185) property.
V	SetItemHeight (see page 21181)	Sets the value of the ItemHeight (see page 21186) property.
V	SetItemIndex (see page 21181)	Sets the value of the ItemIndex property.
V	SetItems (see page 21181)	Sets the value of the Items (see page 21187) property.
	SetMaxLength (see page 21182)	Sets the value of the MaxLength (see page 21189) property.
	SetSelLength (see page 21182)	Sets the value of the SelLength (see page 21190) property.
	SetSelStart (see page 21182)	Sets the value of the SelStart (see page 21193) property.
V	WndProc (see page 21183)	Processes messages received by the combo box.

Public Methods

	Name	Description
V	AddItem (see page 21170)	Adds an item to the drop-down list of items.
V	Clear (see page 21172)	Deletes all text from the edit box and all items from the Items (see page 21187) list.
V	ClearSelection (see page 21172)	Deselects any selected items in the drop-down list.
V	CopySelection (see page 21173)	Copies the selected item in the drop-down list to another list control.
V	DeleteSelected (see page 21174)	Deletes the selected item from the drop-down list.
V	Focused (see page 21175)	Indicates whether the combo box has input focus.
V	SelectAll (see page 21180)	Selects the entire block of text in the edit box.

154.1.7.5.1 TCustomCombo.AddItem Method

Adds an item to the drop-down list of items.

Pascal

```
procedure AddItem(Item: String; AObject: TObject); override;
```

C++

```
virtual __fastcall AddItem(AnsiString Item, TObject * AObject);
```

Description

Call AddItem to add a string, with an associated object, to the drop-down list.

Item is the string to add to the drop-down list.

AObject is an object associated with that string. It can be accessed using the Objects property of the TStrings object that implements the Items (see page 21187) property.

See Also

Items (see page 21187)

Objects (see page 3077)

154.1.7.5.2 TCustomCombo.AdjustDropDown Method

Adjusts the size and position of the drop down list.

Pascal

```
procedure AdjustDropDown; virtual;
```

C++

```
virtual __fastcall AdjustDropDown();
```

Description

TCustomCombo (see page 21154) calls AdjustDropDown when it drops down the drop-down list. This method adjusts the size and position of the drop-down list to reflect the values of the ItemHeight (see page 21186) and DropDownCount (see page 21185) properties.

See Also

ItemHeight (see page 21186)

DroppedDown (see page 21185)

DropDownCount (see page 21185)

154.1.7.5.3 TCustomCombo.Change Method

Generates an OnChange (see page 21197) event.

Pascal

```
procedure Change; dynamic;
```

C++

```
__fastcall Change();
```

Description

Change is called automatically in response to notifications that the text has changed. It responds by generating an OnChange (see page 21197) event. Override this method to add additional processing when the text changes, or to block the OnChange (see page 21197) event.

See Also

OnChange (see page 21197)

Changed (see page 5742)

154.1.7.5.4 TCustomCombo.Clear Method

Deletes all text from the edit box and all items from the Items (see page 21187) list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call Clear to empty the combo box at runtime.

See Also

Items (see page 21187)

Text (see page 5842)

154.1.7.5.5 TCustomCombo.ClearSelection Method

Deselects any selected items in the drop-down list.

Pascal

```
procedure ClearSelection; override;
```

C++

```
virtual __fastcall ClearSelection();
```

Description

ClearSelection checks whether an item in the drop-down list is currently selected. If so, it deselects the item (sets ItemIndex to –1), and changes the Text property to an empty string.

See Also

ItemIndex

Text (see page 5842)

154.1.7.5.6 TCustomCombo.CloseUp Method

Generates an OnCloseUp (see page 21199) event.

Pascal

```
procedure CloseUp; dynamic;
```

C++

```
__fastcall CloseUp();
```

Description

CloseUp is called automatically when the DroppedDown (see page 21185) property changes to false. It generates an OnCloseUp (see page 21199) event.

Override CloseUp to implement additional class-specific responses when the drop-down list closes up or to suppress the event.

See Also

DroppedDown (see page 21185)

OnCloseUp (see page 21199)

154.1.7.5.7 TCustomCombo.ComboWndProc Method

Responds to Windows messages sent to the edit control or drop-down list.

Pascal

```
procedure ComboWndProc(var Message: TMessage; ComboWnd: HWnd; ComboProc: Pointer); virtual;
```

C++

```
virtual __fastcall ComboWndProc(TMessage Message, HWnd ComboWnd, void * ComboProc);
```

Description

ComboWndProc handles the common processing of Windows messages sent to the edit region or list portion of the combo box. It is called by the EditWndProc (see page 21175) and ListWndProc (see page 21178) methods to take care of all processing that is not specific to the edit region or drop-down list.

The Message parameter describes the Windows message. The ComboWnd parameter is the windows handle for the edit region or list. The ComboProc parameter is a pointer to the default window procedure for the window that received the message. ComboWndProc provides default responses to focus events, mouse events, and key strokes.

Override ComboWndProc to change the way the combo box responds to Windows messages. When overriding ComboWndProc, be sure to call the parent class's method to process all unhandled messages.

Note: Do not confuse ComboWndProc with the WndProc (see page 21183) method. ComboWndProc responds to messages that are sent to the edit region or drop-down list. WndProc (see page 21183) responds to messages sent to the combo box itself.

See Also

WndProc (see page 21183)

EditWndProc (see page 21175)

ListWndProc (see page 21178)

154.1.7.5.8 TCustomCombo.CopySelection Method

Copies the selected item in the drop-down list to another list control.

Pascal

```
procedure CopySelection(Destination: TCustomListControl); override;
```

C++

```
virtual __fastcall CopySelection(TCustomListControl Destination);
```

Description

CopySelection checks whether an item is selected in the drop-down list, and if so, adds it to the control specified by Destination, along with its associated object.

See Also

Items (see page 21187)

ItemIndex

154.1.7.5.9 TCustomCombo.CreateWnd Method

Creates the Windows control that represents the combo box.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

The CreateHandle method calls CreateWnd to handle the particulars of creating the combo box window. TCustomCombo (see page 21154) overrides the inherited method to ensure that the newly-created window and the TCustomCombo (see page 21154) properties correctly reflect each other.

The Windows control for the combo box can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. Thus, CreateWnd may be called more than once during the lifetime of the TCustomCombo (see page 21154) instance.

See Also

CreateHandle (see page 6135)

154.1.7.5.10 TCustomCombo.DeleteSelected Method

Deletes the selected item from the drop-down list.

Pascal

```
procedure DeleteSelected; override;
```

C++

```
virtual __fastcall DeleteSelected();
```

Description

DeleteSelected deletes the item specified by the ItemIndex property from the drop-down list.

See Also

ItemIndex

Items (see page 21187)

154.1.7.5.11 TCustomCombo.DestroyWindowHandle Method

Destroys all of the windows controls that implement the combo box.

Pascal

```
procedure DestroyWindowHandle; override;
```

C++

```
virtual __fastcall DestroyWindowHandle();
```

Description

The DestroyWnd method calls DestroyWindowHandle to release the underlying Windows control. TCustomCombo (see page 21154) overrides the inherited method to reset the EditHandle (see page 21185) and ListHandle (see page 21189) properties once the underlying Windows controls are gone.

See Also

DestroyWnd (see page 6141)

EditHandle (see page 21185)

ListHandle (see page 21189)

154.1.7.5.12 TCustomCombo.DropDown Method

Generates an OnDropDown (see page 21199) event.

Pascal

```
procedure DropDown; dynamic;
```

C++

```
__fastcall DropDown();
```

Description

DropDown is called automatically when the DroppedDown (see page 21185) property changes to true. It responds by generating an OnDropDown (see page 21199) event. To perform additional processing or block the OnDropDown (see page 21199) event, override DropDown.

See Also

OnDropDown (see page 21199)

DroppedDown (see page 21185)

154.1.7.5.13 TCustomCombo.EditWndProc Method

Responds to Windows messages that are sent to the edit control portion of the combo box.

Pascal

```
procedure EditWndProc(var Message: TMessage); virtual;
```

C++

```
virtual __fastcall EditWndProc(TMessage Message);
```

Description

A combo box is composed of several windows: the combo box itself, the edit control, and the drop-down list. EditWndProc is the window procedure for the edit control portion of the combo box.

See Also

EditHandle (see page 21185)

WndProc (see page 21183)

ListWndProc (see page 21178)

ComboWndProc (see page 21173)

154.1.7.5.14 TCustomCombo.Focused Method

Indicates whether the combo box has input focus.

Pascal

```
function Focused: Boolean; override;
```

C++

```
virtual __fastcall Boolean Focused();
```

Description

Call Focused to determine whether the combo box is currently the recipient of keyboard input. When Focused returns true, the combo box is the active control and keystroke messages are sent to it. When Focused is false, the combo box is not active (although it may be selected).

See Also

ActiveControl (see page 13896)

154.1.7.5.15 TCustomCombo.GetCount Method

Returns the value of the ItemCount (see page 21186) property.

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCount returns the number of items in the drop-down list.

See Also

ItemCount (see page 21186)

154.1.7.5.16 TCustomCombo.GetDroppedDown Method

Returns the value of the DroppedDown (see page 21185) property.

Pascal

```
function GetDroppedDown: Boolean;
```

C++

```
__fastcall Boolean GetDroppedDown();
```

Description

GetDroppedDown is the read implementation of the DroppedDown (see page 21185) property. It returns true if the drop-down list is showing.

See Also

DroppedDown (see page 21185)

154.1.7.5.17 TCustomCombo.GetItemCount Method

Returns the value of the ItemCount (see page 21186) property.

Pascal

```
function GetItemCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetItemCount() = 0;
```

Description

GetItemCount is the protected read access method for the ItemCount (see page 21186) property. In TCustomCombo (see page 21154), this method is abstract (pure virtual), meaning it has no implementation. Descendant classes override this method to return the number of items in the drop-down list.

See Also

ItemCount (see page 21186)

154.1.7.5.18 TCustomCombo.GetItemHt Method

Returns the value of the ItemHeight (see page 21186) property.

Pascal

```
function GetItemHt: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetItemHt() = 0;
```

Description

GetItemHt is the protected read access method for the ItemHeight (see page 21186) property. In TCustomCombo (see page 21154), this method is abstract (pure virtual), meaning it has no implementation. Descendant classes override this method to return the height of items in the drop-down list.

See Also

ItemHeight (see page 21186)

154.1.7.5.19 TCustomCombo.GetItemIndex Method

Returns the value of the ItemIndex property.

Pascal

```
function GetItemIndex: Integer; override;
```

C++

```
virtual __fastcall int GetItemIndex();
```

Description

GetItemIndex is the read implementation of the ItemIndex property. It returns the index of the selected item in the drop-down list. The first item in the list has index 0, the second item has index 1, and so on. If no item is selected, the value of ItemIndex is -1.

See Also

ItemIndex

154.1.7.5.20 TCustomCombo.GetItemsClass Method

Indicates the class that implements the Items (see page 21187) property.

Pascal

```
function GetItemsClass: TCustomComboBoxStringsClass; virtual; abstract;
```

C++

```
virtual __fastcall TCustomComboBoxStringsClass GetItemsClass() = 0;
```

Description

TCustomCombo (see page 21154) uses GetItemsClass internally to determine what class to instantiate when it implements the Items (see page 21187) property.

In TCustomCombo (see page 21154), GetItemsClass is abstract pure virtual, meaning it has no implementation. TCustomCombo (see page 21154) descendants override this method to indicate a TCustomComboBoxStrings (see page 21225) descendant that implements the Items (see page 21187) property.

See Also

Items (see page 21187)

154.1.7.5.21 TCustomCombo.GetSelLength Method

Returns the value of the SelLength (see page 21190) property.

Pascal

```
function GetSelLength: Integer;
```

C++

```
__fastcall int GetSelLength();
```

Description

GetSelLength is the read implementation of the SelLength (see page 21190) property. It returns the number of selected characters.

See Also

SelLength (see page 21190)

154.1.7.5.22 TCustomCombo.GetSelStart Method

Returns the value of the SelStart (see page 21193) property.

Pascal

```
function GetSelStart: Integer;
```

C++

```
__fastcall int GetSelStart();
```

Description

GetSelStart is the read implementation of the SelStart (see page 21193) property. It returns the position of the first selected character, or the position of the cursor if no characters are selected.

See Also

SelStart (see page 21193)

154.1.7.5.23 TCustomCombo.ListWndProc Method

Responds to Windows messages that are sent to the drop-down list portion of the combo box.

Pascal

```
procedure ListWndProc(var Message: TMessage);
```

C++

```
__fastcall ListWndProc(TMessage Message);
```

Description

A combo box is composed of several windows: the combo box itself, the edit control, and the drop-down list. ListWndProc is the window procedure for the drop-down list.

See Also

ListHandle (see page 21189)

WndProc (see page 21183)

EditWndProc (see page 21175)

ComboWndProc (see page 21173)

154.1.7.5.24 TCustomCombo.Loaded Method

Initializes the combo box after its form is loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The component streaming system calls this method after it loads the combo box's form from a stream.

TCustomCombo (see page 21154) overrides the inherited method to select the item specified by the ItemIndex property once the form is fully loaded.

See Also

Loaded (see page 2782)

ItemIndex

154.1.7.5.25 TCustomCombo.Select Method

Generates an OnSelect (see page 21201) event.

Pascal

```
procedure Select; dynamic;
```

C++

```
__fastcall Select();
```

Description

Select is called automatically when the selected item in the drop-down list changes. It responds by generating an OnSelect (see page 21201) event. To perform additional processing or block the OnSelect (see page 21201) event, override Select.

See Also

OnSelect (see page 21201)

ItemIndex

154.1.7.5.26 TCustomCombo.SelectAll Method

Selects the entire block of text in the edit box.

Pascal

```
procedure SelectAll; override;
```

C++

```
virtual __fastcall SelectAll();
```

Description

Use SelectAll to select the entire block of text in the edit box. To select only part of the text, use the SelStart (see page 21193) and SelLength (see page 21190) properties.

See Also

SelStart (see page 21193)

SelLength (see page 21190)

154.1.7.5.27 TCustomCombo.SetDropDownCount Method

Sets the value of the DropDownCount (see page 21185) property.

Pascal

```
procedure SetDropDownCount(const Value: Integer); virtual;
```

C++

```
virtual __fastcall SetDropDownCount(const int Value);
```

Description

SetDropDownCount is the protected write implementation of the DropDownCount (see page 21185) property.

See Also

DropDownCount (see page 21185)

154.1.7.5.28 TCustomCombo.SetDroppedDown Method

Sets the value of the DroppedDown (see page 21185) property.

Pascal

```
procedure SetDroppedDown(Value: Boolean);
```

C++

```
__fastcall SetDroppedDown(Boolean Value);
```

Description

SetDroppedDown is the protected write implementation of the DroppedDown (see page 21185) property.

See Also

DroppedDown (see page 21185)

154.1.7.5.29 TCustomCombo.SetItemHeight Method

Sets the value of the ItemHeight (see page 21186) property.

Pascal

```
procedure SetItemHeight(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetItemHeight(int Value);
```

Description

SetItemHeight is the protected write implementation of the ItemHeight (see page 21186) property. Although SetItemHeight never raises an exception, in some cases it does not change the value of the ItemHeight (see page 21186) property. ItemHeight (see page 21186) can only be changed for some TCustomCombo (see page 21154) descendants, and then only under specific circumstances.

For example, TComboBoxEx never allows the ItemHeight (see page 21186) property to change.

See Also

ItemHeight (see page 21186)

154.1.7.5.30 TCustomCombo.SetItemIndex Method

Sets the value of the ItemIndex property.

Pascal

```
procedure SetItemIndex(const Value: Integer); override;
```

C++

```
virtual __fastcall SetItemIndex(const int Value);
```

Description

SetItemIndex is the protected write implementation of the ItemIndex property. It selects an item in the drop-down list.

Value is the item to select, where 0 indicates the first item, 1 indicates the second item, and so on. A negative Value causes all items to be unselected.

See Also

ItemIndex

154.1.7.5.31 TCustomCombo.SetItems Method

Sets the value of the Items (see page 21187) property.

Pascal

```
procedure SetItems(const Value: TStrings); virtual;
```

C++

```
virtual __fastcall SetItems(const TStrings Value);
```

Description

SetItems is the protected write implementation for the Items (see page 21187) property. It copies the strings (and possibly their associated objects) from Value to the Items (see page 21187) property.

See Also

Items (see page 21187)

Assign (see page 3043)

154.1.7.5.32 TCustomCombo.SetMaxLength Method

Sets the value of the MaxLength (see page 21189) property.

Pascal

```
procedure SetMaxLength(Value: Integer);
```

C++

```
__fastcall SetMaxLength(int Value);
```

Description

SetMaxLength is the protected write implementation of the MaxLength (see page 21189) property.

See Also

MaxLength (see page 21189)

154.1.7.5.33 TCustomCombo.SetSelLength Method

Sets the value of the SelLength (see page 21190) property.

Pascal

```
procedure SetSelLength(Value: Integer);
```

C++

```
__fastcall SetSelLength(int Value);
```

Description

SetSelLength is the protected write implementation of the SelLength (see page 21190) property.

See Also

SelLength (see page 21190)

154.1.7.5.34 TCustomCombo.SetSelStart Method

Sets the value of the SelStart (see page 21193) property.

Pascal

```
procedure SetSelStart(Value: Integer);
```

C++

```
__fastcall SetSelStart(int Value);
```

Description

SetSelStart is the protected write implementation of the SelStart (see page 21193) property.

See Also

SelStart (see page 21193)

154.1.7.5.35 TCustomCombo.WndProc Method

Processes messages received by the combo box.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc is the first method to received Windows messages that are sent to the combo box. Messages (see page 16211) sent to the child windows of the combo box (for the edit region and the list portion) go to the EditWndProc (see page 21175) or ListWndProc (see page 21178) method instead. WndProc handles messages about changes to the system colors defined in the Windows control panel, and closes up the drop-down list when the Enter key or the Escape key is pressed. All other messages are passed to Dispatch for default processing.

See Also

ComboWndProc (see page 21173)

EditWndProc (see page 21175)

ListWndProc (see page 21178)

154.1.7.6 TCustomCombo Properties

The properties of the TCustomCombo class are listed here.

Protected Properties

	Name	Description
	DropDownCount (see page 21185)	Specifies the maximum number of items displayed in the drop-down list.
R	EditHandle (see page 21185)	Returns the window handle for the edit portion of the combo box.
R	ItemCount (see page 21186)	Specifies the number of items in the drop-down list.
	ItemHeight (see page 21186)	Specifies the height, in pixels, of the items in the drop-down list.
R	ListHandle (see page 21189)	Provides access to the window handle of the list of items in the combo box.
	MaxLength (see page 21189)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	ParentColor (see page 21190)	Determines where a control looks for its color information.

Public Properties

	Name	Description
R	Canvas (see page 21184)	Provides access to the drawing surface of the combo box.
	DroppedDown (see page 21185)	Indicates whether the drop-down list is currently displayed.
	Items (see page 21187)	Provides access to the list of items (strings) in the list portion of the combo box.
	SelLength (see page 21190)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 21193)	Specifies the position of the first selected character in the edit portion of the combo box.
	TabStop (see page 21196)	Determines if the user can tab to a control.

154.1.7.6.1 TCustomCombo.Canvas Property

Provides access to the drawing surface of the combo box.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use Canvas as the drawing surface when customizing the way the combo box paints itself. For example, when using a TCustomCombo (see page 21154) descendant that allows applications to draw combo box items in an OnDrawItem even, the OnDrawItem event handler uses Canvas for painting the customized items.

C++ Examples:

```
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TComboBox *myComboBox;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TComboBox")
  {
    myComboBox = (TComboBox *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myComboBox->Canvas->MoveTo(myComboBox->ClientRect.Left, myComboBox->ClientRect.Top);
    myComboBox->Canvas->LineTo(myComboBox->ClientRect.Right, myComboBox->ClientRect.Bottom);
    myComboBox = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TComboBox) then
  begin
    myComboBox:= (ListBox1.Items.Objects[Index] as TComboBox);
    with myComboBox.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myComboBox.Canvas.MoveTo(Left, Top);
      myComboBox.Canvas.LineTo(Right, Bottom);
    end;
  myComboBox:= nil;
  end;
```

See Also

OnDrawItem (see page 21224)

154.1.7.6.2 TCustomCombo.DropDownCount Property

Specifies the maximum number of items displayed in the drop-down list.

Pascal

```
property DropDownCount: Integer;
```

C++

```
__property int DropDownCount;
```

Description

By default, the drop-down list is long enough to contain eight items without requiring the user to scroll to see them all. To make the drop-down list smaller or larger, specify a number larger or smaller than eight as the DropDownCount value.

If the DropDownCount value is larger than the number of items in the Items (see page 21187) property, the drop-down list will be just large enough to hold all the possible choices and no more. If the DropDownCount value is smaller then the number of item in the Items (see page 21187) property, the drop down list will display a scroll bar.

See Also

Items (see page 21187)

ItemCount (see page 21186)

154.1.7.6.3 TCustomCombo.DroppedDown Property

Indicates whether the drop-down list is currently displayed.

Pascal

```
property DroppedDown: Boolean;
```

C++

```
__property Boolean DroppedDown;
```

Description

Read DroppedDown to determine whether the drop-down list is visible. Set DropDown (see page 21175) to drop the list down or close it up. DroppedDown is true when the drop-down list is open, and false when it is closed.

See Also

OnDropDown (see page 21199)

AutoDropDown (see page 21221)

154.1.7.6.4 TCustomCombo.EditHandle Property

Returns the window handle for the edit portion of the combo box.

Pascal

```
property EditHandle: HWnd;
```

C++

```
__property HWnd EditHandle;
```

Description

A combo box is composed of several windows: the combo box itself, the edit control, and the drop-down list. EditHandle is the

window handle of the edit control portion. (The Handle property and the ListHandle (see page 21189) property give the handles for the combo box itself and the drop-down list, respectively.)

Use EditHandle when making Windows API calls addressed to the edit portion of the combo box.

See Also

Handle (see page 6207)

ListHandle (see page 21189)

154.1.7.6.5 TCustomCombo.ItemCount Property

Specifies the number of items in the drop-down list.

Pascal

```
property ItemCount: Integer;
```

C++

```
__property int ItemCount;
```

Description

Read ItemCount to get the number of items in the drop-down list. ItemCount is the number of all items, not just those that are visible at one time.

See Also

Items (see page 21187)

DropDownCount (see page 21185)

154.1.7.6.6 TCustomCombo.ItemHeight Property

Specifies the height, in pixels, of the items in the drop-down list.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Read ItemHeight to determine the height of the items in the drop-down list.

The effect of setting ItemHeight varies, depending on the type of combo box and the style. For example, TComboBox (see page 21064) lets you set the item height only when Style is set to csOwnerDrawFixed or csOwnerDrawVariable. If Style is csOwnerDrawVariable, ItemHeight is the default height for drawing list items, but this value can be overridden by an OnMeasureItem event handler.

TComboBoxEx does not let you change the value of ItemHeight.

Note: Combo boxes that do not support changes to ItemHeight do not raise an exception when you attempt to change ItemHeight; they simply ignore the attempt.

See Also

Items (see page 21187)

OnMeasureItem (see page 21225)

Style (see page 21224)

154.1.7.6.7 TCustomCombo.Items Property

Provides access to the list of items (strings) in the list portion of the combo box.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Read Items to access the list of items that appears in the combo box. Use the methods of Items to add, insert, delete and move items. Set the value of Items to copy the items from another string list.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", (TObject *)vsIcon);
  ComboBox1->Items->AddObject("vsList", (TObject *)vsList);
  ComboBox1->Items->AddObject("vsReport", (TObject *)vsReport);
  ComboBox1->Items->AddObject("vsSmallIcon", (TObject *)vsSmallIcon);
  // Display first item in the Combo Box
```

```
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := 'Image' + IntToStr(I);
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
    NewColumn.Width := 200;
    // Add View styles and constants to the Combo Box
    ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
    ComboBox1.Items.AddObject('vsList', TObject(vsList));
    ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
    ComboBox1.Items.AddObject(
      'vsSmallIcon', TObject(vsSmallIcon));
    // Display first item in the Combo Box
    ComboBox1.ItemIndex := 0;
  end;
end;

procedure TComboForm.ComboBox1Click(Sender: TObject);
```

```
begin
  with ComboBox1 do
    ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

end;
```

See Also

ItemIndex

TStrings (see page 3029)

Assign (see page 3043)

154.1.7.6.8 TCustomCombo.ListHandle Property

Provides access to the window handle of the list of items in the combo box.

Pascal

```
property ListHandle: HWnd;
```

C++

```
__property HWnd ListHandle;
```

Description

A combo box is composed of several windows: the combo box itself, the edit control, and the drop-down list. ListHandle is the window handle of the drop-down list. (The Handle property and the EditHandle (see page 21185) property give the handles for the combo box itself and the edit portion, respectively.)

Use ListHandle when making Windows API calls addressed to the list portion of the combo box.

Note: ListHandle is only available when the combo box has a "simple" style (an edit box with a fixed list below rather than a drop-down list).

See Also

Handle (see page 6207)

EditHandle (see page 21185)

154.1.7.6.9 TCustomCombo.MaxLength Property

Specifies the maximum number of characters the user can type into the edit portion of the combo box.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that the user can enter into the edit portion of the combo box. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that

may be entered into an edit control.

See Also

Text (see page 5842)

154.1.7.6.10 TCustomCombo.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

154.1.7.6.11 TCustomCombo.SelLength Property

Specifies the length, in characters, of the selected text in the edit portion of the combo box.

Pascal

```
property SelLength: Integer;
```

C++

```
__property int SelLength;
```

Description

The selected portion of the text in the edit region of the combo box is the string of SelLength characters that starts at the character specified by the SelStart (see page 21193) property. The selected text appears highlighted in the edit region of the combo box.

To select a section of text in the edit region, first set the SelStart (see page 21193) property to position the cursor, and then set SelLength to the number of characters that should be selected. The SelText property is the substring that is selected.

Note: The SelLength property is only valid when the combo box has focus. When the combo box does not have focus, reading SelLength always yields a value of 0, and setting it has no effect.

C++ Examples:

```
/*
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
*/
void __fastcall TForm1::ComboBox1KeyPress(TObject *Sender, char &Key)
{
  // first, process the keystroke to obtain the current string
  // This code requires all items in list to be uppercase
  if (Key >= 'a' && Key <= 'z')
    Key -= 32; // Force Uppercase only!
  TComboBox *pCB = (TComboBox *)Sender;
  AnsiString TmpStr;
  bool BackSpace = (Key == (char)VK_BACK);
  if (BackSpace && pCB->SelLength)
    TmpStr = pCB->Text.SubString(1,pCB->SelStart)+
             pCB->Text.SubString(pCB->SelLength+pCB->SelStart+1,255);
  else if (BackSpace) // SelLength == 0
    TmpStr = pCB->Text.SubString(1,pCB->SelStart-1)+
             pCB->Text.SubString(pCB->SelStart+1,255);
  else //Key is a visible character
    TmpStr = pCB->Text.SubString(1,pCB->SelStart)+ Key +
             pCB->Text.SubString(pCB->SelLength+pCB->SelStart+1,255);
  if (TmpStr.IsEmpty())
    return;
  // set SelSt to the current insertion point
  int SelSt = pCB->SelStart;
  if (BackSpace && SelSt > 0)
    SelSt--;
  else if (!BackSpace)
    SelSt++;
  Key = 0; // indicate that key was handled
  if (SelSt == 0)
  {
    pCB->Text = "";
    return;
  }
  // Now that TmpStr is the currently typed string, see if we can locate a match
  bool Found = false;
  for (int i = 1; i < pCB->Items->Count; i++)
    if (TmpStr == pCB->Items->Strings[i-1].SubString(1,TmpStr.Length()))
    {
      pCB->Text = pCB->Items->Strings[i-1]; // update to the match that was found
      pCB->ItemIndex = i-1;
      Found = true;
      break;
    }
  if (Found) // select the untyped end of the string
  {
    pCB->SelStart = SelSt;
    pCB->SelLength = pCB->Text.Length() - SelSt;
  }
  else Beep();
}
//---------------------------------------------------------------------------
TStringList *CharSetList;

void __fastcall TForm1::AddCharacterSet(AnsiString S)
{
```

```
  CharSetList->Add(S);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
CharSetList = new TStringList();
GetCharsetValues(AddCharacterSet);
CharSetList->Sort();
ComboBox1->Items = CharSetList;
}
```

Delphi Examples:

```
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
Note: Set the TComboBox AutoComplete to false to turn off
default auto completion.
}
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
var
  Found: boolean;
  i, ct, SelSt: Integer;
  TmpStr: string;
begin
  { first, process the keystroke to obtain the current string }
  { This code requires all items in list to be uppercase}
  if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
  ct := (Sender as TComboBox).Items.Count;
  with (Sender as TComboBox) do
  begin
    SelSt := SelStart;
    if (Key = Chr(vk_Back)) and (SelLength <> 0) then
     TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
    else if Key = Chr(vk_Back) then {SelLength = 0}
     TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
    else {Key in ['A'..'Z', etc]}
     TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
    if TmpStr = '' then Exit;
    { update SelSt to the current insertion point }
    if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
    else if Key <> Chr(vk_Back) then Inc(SelSt);
    Key := #0; { indicate that key was handled }
    if SelSt = 0 then
    begin
      Text:= '';
      Exit;
    end;
   {Now that TmpStr is the currently typed string, see if we can locate a match }
    Found := False;
    for i := 1 to ct do
    begin
      if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
      begin
        Text := Items[i-1]; { update to the match that was found }
        ItemIndex := i-1;
        Found := True;
        Break;
      end;
    end;
    if Found then { select the untyped end of the string }
```

```
      begin
        SelStart := SelSt;
        SelLength := Length(Text) - SelSt;
      end
      else Beep;
    end;
  end;

  var
    CharSetList : TStringList;

  procedure TForm1.AddCharacterSet(const S: string);
  begin
    CharSetList.Add(S);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    CharSetList := TStringList.Create;
    Graphics.GetCharsetValues(AddCharacterSet);
    ComboBox1.Items := CharSetList;
  end;
```

See Also

SelStart (see page 21193)

SelText (see page 21222)

Text (see page 5842)

154.1.7.6.12 TCustomCombo.SelStart Property

Specifies the position of the first selected character in the edit portion of the combo box.

Pascal

```
property SelStart: Integer;
```

C++

```
__property int SelStart;
```

Description

Read SelStart to determine where the selection starts in the Text property. The first character is designated with 0, the second with 1, and so on. When SelLength (see page 21190) is 0, SelStart is the position of the cursor.

Set SelStart to position the cursor in the edit region of the combo box. Setting SelStart to 0 positions the cursor before the first character, setting SelStart to 1 positions the cursor before the second character, and so on.

Note: Setting SelStart changes the SelLength (see page 21190) property to 0. When using SelStart and SelLength (see page 21190) to programmatically select a portion of the text in the edit region, set SelStart before setting SelLength (see page 21190).

Note: The SelStart property is only valid when the combo box has focus. When the combo box does not have focus, reading SelStart always yields a value of 0, and setting it has no effect.

C++ Examples:

```
/*
The following example shows how to complete partial strings
```

```
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
*/
void __fastcall TForm1::ComboBox1KeyPress(TObject *Sender, char &Key)
{
  // first, process the keystroke to obtain the current string
  // This code requires all items in list to be uppercase
  if (Key >= 'a' && Key <= 'z')
    Key -= 32; // Force Uppercase only!
  TComboBox *pCB = (TComboBox *)Sender;
  AnsiString TmpStr;
  bool BackSpace = (Key == (char)VK_BACK);
  if (BackSpace && pCB->SelLength)
    TmpStr = pCB->Text.SubString(1,pCB->SelStart)+
             pCB->Text.SubString(pCB->SelLength+pCB->SelStart+1,255);
  else if (BackSpace) // SelLength == 0
    TmpStr = pCB->Text.SubString(1,pCB->SelStart-1)+
             pCB->Text.SubString(pCB->SelStart+1,255);
  else //Key is a visible character
    TmpStr = pCB->Text.SubString(1,pCB->SelStart)+ Key +
             pCB->Text.SubString(pCB->SelLength+pCB->SelStart+1,255);
  if (TmpStr.IsEmpty())
    return;
  // set SelSt to the current insertion point
  int SelSt = pCB->SelStart;
  if (BackSpace && SelSt > 0)
    SelSt--;
  else if (!BackSpace)
    SelSt++;
  Key = 0; // indicate that key was handled
  if (SelSt == 0)
  {
    pCB->Text = "";
    return;
  }
  // Now that TmpStr is the currently typed string, see if we can locate a match
  bool Found = false;
  for (int i = 1; i < pCB->Items->Count; i++)
    if (TmpStr == pCB->Items->Strings[i-1].SubString(1,TmpStr.Length()))
    {
      pCB->Text = pCB->Items->Strings[i-1]; // update to the match that was found
      pCB->ItemIndex = i-1;
      Found = true;
      break;
    }
  if (Found) // select the untyped end of the string
  {
    pCB->SelStart = SelSt;
    pCB->SelLength = pCB->Text.Length() - SelSt;
  }
  else Beep();
}
//---------------------------------------------------------------------------
TStringList *CharSetList;

void __fastcall TForm1::AddCharacterSet(AnsiString S)
{
  CharSetList->Add(S);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
CharSetList = new TStringList();
GetCharsetValues(AddCharacterSet);
CharSetList->Sort();
ComboBox1->Items = CharSetList;
}
```

Delphi Examples:

```
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
Note: Set the TComboBox AutoComplete to false to turn off
default auto completion.
}
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
var
  Found: boolean;
  i, ct, SelSt: Integer;
  TmpStr: string;
begin
  { first, process the keystroke to obtain the current string }
  { This code requires all items in list to be uppercase}
  if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
  ct := (Sender as TComboBox).Items.Count;
  with (Sender as TComboBox) do
  begin
    SelSt := SelStart;
    if (Key = Chr(vk_Back)) and (SelLength <> 0) then
     TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
    else if Key = Chr(vk_Back) then {SelLength = 0}
     TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
    else {Key in ['A'..'Z', etc]}
     TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
    if TmpStr = '' then Exit;
    { update SelSt to the current insertion point }
    if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
    else if Key <> Chr(vk_Back) then Inc(SelSt);
    Key := #0; { indicate that key was handled }
    if SelSt = 0 then
    begin
      Text:= '';
      Exit;
    end;
   {Now that TmpStr is the currently typed string, see if we can locate a match }
    Found := False;
    for i := 1 to ct do
    begin
      if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
      begin
        Text := Items[i-1]; { update to the match that was found }
        ItemIndex := i-1;
        Found := True;
        Break;
      end;
    end;
    if Found then { select the untyped end of the string }
    begin
      SelStart := SelSt;
      SelLength := Length(Text) - SelSt;
    end
```

```
      else Beep;
    end;
  end;

  var
    CharSetList : TStringList;

  procedure TForm1.AddCharacterSet(const S: string);
  begin
    CharSetList.Add(S);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    CharSetList := TStringList.Create;
    Graphics.GetCharsetValues(AddCharacterSet);
    ComboBox1.Items := CharSetList;
  end;
```

See Also

SelLength (see page 21190)

SelText (see page 21222)

Text (see page 5842)

154.1.7.6.13 TCustomCombo.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

154.1.7.7 TCustomCombo Events

The events of the TCustomCombo class are listed here.

Protected Events

	Name	Description
	OnChange (see page 21197)	Occurs when the user changes the text displayed in the edit region.
	OnCloseUp (see page 21199)	Occurs when the drop-down list closes up due to some user action.
	OnDropDown (see page 21199)	Occurs when the user opens the drop-down list.
	OnSelect (see page 21201)	Occurs when the user selects a string in the drop-down list.

154.1.7.7.1 TCustomCombo.OnChange Event

Occurs when the user changes the text displayed in the edit region.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action immediately after the user edits the text in the edit region or selects an item from the list. The Text property gives the new value in the edit region.

Note: OnChange only occurs in response to user actions. Changing the Text property programmatically does not trigger an OnChange event.

C++ Examples:

```
/*
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
*/
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  if (pCB)
    PageControl1->ActivePage = (TTabSheet *)(pCB->Items->Objects[pCB->ItemIndex]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
    ComboBox1->Items->AddObject(PageControl1->Pages[i]->Name, PageControl1->Pages[i]);
    ComboBox1->ItemIndex = 0;
  }
}
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
```

```
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to PageControl1.PageCount - 1 do
    ComboBox1.Items.AddObject(PageControl1.Pages[i].Name,
      PageControl1.Pages[i]);
  ComboBox1.ItemIndex := 0;
end;

procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  if (Sender is TComboBox) then
    with (Sender as TComboBox) do
      PageControl1.ActivePage := TTabSheet(Items.Objects[ItemIndex]);
end;
{
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
}
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
```

```
    FileListBox1.Drive := DriveComboBox1.Drive;
    FileListBox1.Directory := DirectoryListBox1.Directory;
  end;
```

See Also

Text (see page 5842)

OnCloseUp (see page 21199)

154.1.7.7.2 TCustomCombo.OnCloseUp Event

Occurs when the drop-down list closes up due to some user action.

Pascal

```
property OnCloseUp: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCloseUp;
```

Description

Write an OnCloseUp event handler to implement special processing that needs to occur when the drop-down list closes up. For example, an OnCloseUp event handler can check whether the user changed the selected item while the list was dropped down and respond accordingly.

See Also

OnDropDown (see page 21199)

DroppedDown (see page 21185)

154.1.7.7.3 TCustomCombo.OnDropDown Event

Occurs when the user opens the drop-down list.

Pascal

```
property OnDropDown: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDropDown;
```

Description

Write an OnDropDown event handler to implement special processing that needs to occur only when the drop-down list is activated.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
```

```
    {
      if (((TForm1 *)Screen->CustomForms[i])->Visible)
        ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
    }
    ComboBox1->Items->EndUpdate(); //reenable painting
  }

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    ComboBox1->Sorted = True; // make sure the form names are sorted
  }

  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    Form2->Show();
  }

  void __fastcall TForm1::Button2Click(TObject *Sender)
  {
    Form3->Show();
  }
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
    end;
    Items.EndUpdate; {reenable painting }
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.Show;
end;
```

See Also

DroppedDown (see page 21185)

OnCloseUp (see page 21199)

154.1.7.7.4 TCustomCombo.OnSelect Event

Occurs when the user selects a string in the drop-down list.

Pascal

```
property OnSelect: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSelect;
```

Description

Write an OnSelect event handler to respond when the user changes the selected item in the drop-down list. OnSelect occurs after the selection has changed and the Text property reflects the new item.

See Also

autocomplete (see page 21220)

ItemIndex

Text (see page 5842)

154.1.8 TCustomComboBox Class

TCustomComboBox is the base class for combobox components.

Class Hierarchy

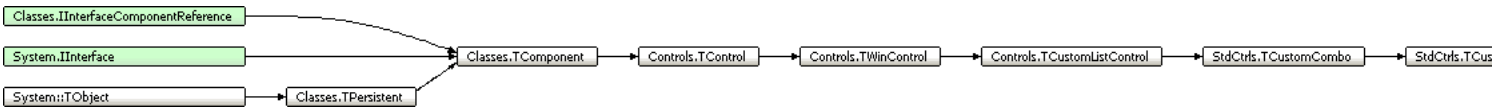

Pascal

```
TCustomComboBox = class(TCustomCombo);
```

C++

```
class TCustomComboBox : public TCustomCombo;
```

File

StdCtrls

Description

Use TCustomComboBox as a base class when defining a control that represents a specialized Windows combo box. A combo box is an edit box with a scrollable list attached to it; users can select an item from the list or type directly into the edit box. Combo boxes can present the list as a drop-down list or as a list box that is always visible.

Note: To create a combo box class that uses the extended combo box features such as images associated with entries, do not use TCustomComboBox. Instead, use the TCustomComboBoxEx class instead.

Do not create instances of TCustomComboBox. To put a combo box in a form, use a descendant of TCustomComboBox, such as TComboBox (see page 21064).

See Also

TComboBox (see page 21064)

TDBComboBox (see page 8657)

TDriveComboBox (see page 13396)

TEdit (see page 21402)

TFilterComboBox (see page 13547)

TListBox (see page 21598)

TColorBox (see page 12182)

TCustomComboBoxEx (see page 3678)

154.1.8.1 TCustomComboBox Members

The following tables list the members exposed by TCustomComboBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnChange (see page 21197)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnCloseUp (see page 21199)	Occurs when the drop-down list closes up due to some user action.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 21224)	Occurs when an item in an owner-draw combo box needs to be displayed.
	OnDropDown (see page 21199)	Occurs when the user opens the drop-down list.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 21225)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnSelect (see page 21201)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDefEditProc (see page 21167)	This is FDefEditProc, a member of class TCustomCombo.
	FDefListProc (see page 21167)	This is FDefListProc, a member of class TCustomCombo.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FDropHandle (see page 21168)	This is FDropHandle, a member of class TCustomCombo.
	FDroppingDown (see page 21168)	This is FDroppingDown, a member of class TCustomCombo.
	FEditHandle (see page 21168)	This is FEditHandle, a member of class TCustomCombo.
	FEditInstance (see page 21168)	This is FEditInstance, a member of class TCustomCombo.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFocusChanged (see page 21168)	This is FFocusChanged, a member of class TCustomCombo.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FIsFocused (see page 21169)	This is FIsFocused, a member of class TCustomCombo.
	FListHandle (see page 21169)	This is FListHandle, a member of class TCustomCombo.
	FListInstance (see page 21169)	This is FListInstance, a member of class TCustomCombo.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveIndex (see page 21169)	This is FSaveIndex, a member of class TCustomCombo.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustDropDown (see page 21171)	Adjusts the size and position of the drop down list.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.

	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21171)	Generates an OnChange (see page 21197) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	CloseUp (see page 21172)	Generates an OnCloseUp (see page 21199) event.
	ComboWndProc (see page 21173)	Responds to Windows messages sent to the edit control or drop-down list.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 21215)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 21215)	Creates the Windows control that represents the combo box.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWindowHandle (see page 21174)	Destroys all of the windows controls that implement the combo box.
	DestroyWnd (see page 21215)	Destroys the Windows control that represents the combo box.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus

	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawItem (see page 21216)	Generates an OnDrawItem (see page 21224) event.
	DropDown (see page 21175)	Generates an OnDropDown (see page 21199) event.
V	EditWndProc (see page 21175)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetCount (see page 21176)	Returns the value of the ItemCount (see page 21186) property.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetDroppedDown (see page 21176)	Returns the value of the DroppedDown (see page 21185) property.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetItemCount (see page 21216)	Returns the number of items in the Items (see page 21187) property list.
V	GetItemHt (see page 21216)	Returns the value of the ItemHeight (see page 21186) property.
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
V	GetItemIndex (see page 21177)	Returns the value of the ItemIndex property.
V	GetItemsClass (see page 21217)	Returns the class used to implement the Items (see page 21187) property.
V	GetOwner (see page 2778)	Returns the owner of a component.

	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelLength (see page 21178)	Returns the value of the SelLength (see page 21190) property.
	GetSelStart (see page 21178)	Returns the value of the SelStart (see page 21193) property.
	GetSelText (see page 21217)	Returns the value of the SelText (see page 21222) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
V	KeyPress (see page 21217)	Responds to WM_CHAR messages sent to the combo box.
	KeyUp (see page 6164)	Respond to released key.
	ListWndProc (see page 21178)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 21179)	Initializes the combo box after its form is loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
V	MeasureItem (see page 21218)	Generates an OnMeasureItem (see page 21225) event.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.

	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Select (see page 21179)	Generates an OnSelect (see page 21201) event.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectItem (see page 21218)	Selects a specified item from the drop-down list.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetDropDownCount (see page 21180)	Sets the value of the DropDownCount (see page 21185) property.
	SetDroppedDown (see page 21180)	Sets the value of the DroppedDown (see page 21185) property.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetItemHeight (see page 21181)	Sets the value of the ItemHeight (see page 21186) property.
A	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
V	SetItemIndex (see page 21181)	Sets the value of the ItemIndex property.
V	SetItems (see page 21181)	Sets the value of the Items (see page 21187) property.
	SetMaxLength (see page 21182)	Sets the value of the MaxLength (see page 21189) property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelLength (see page 21182)	Sets the value of the SelLength (see page 21190) property.
	SetSelStart (see page 21182)	Sets the value of the SelStart (see page 21193) property.
V	SetStyle (see page 21219)	Sets the Style (see page 21224) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 21219)	Processes messages received by the combo box.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownCount (see page 21185)	Specifies the maximum number of items displayed in the drop-down list.
R	EditHandle (see page 21185)	Returns the window handle for the edit portion of the combo box.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	ItemCount (see page 21186)	Specifies the number of items in the drop-down list.
	ItemHeight (see page 21186)	Specifies the height, in pixels, of the items in the drop-down list.
R	ListHandle (see page 21189)	Provides access to the window handle of the list of items in the combo box.
	MaxLength (see page 21189)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 21190)	Determines where a control looks for its color information.

	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Sorted (see page 21222)	Determines whether the list portion of the combo box is alphabetized.
	Style (see page 21224)	Determines the display style of the combo box.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddItem (see page 5940)	Adds an item to the list control.
	AddItem (see page 21170)	Adds an item to the drop-down list of items.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 5940)	Deletes all of the items from the list control.
	Clear (see page 21172)	Deletes all text from the edit box and all items from the Items (see page 21187) list.
	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
	ClearSelection (see page 21172)	Deselects any selected items in the drop-down list.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
	CopySelection (see page 21173)	Copies the selected item in the drop-down list to another list control.
	Create (see page 21214)	Creates and initializes a TCustomComboBox (see page 21201) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.

V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	DeleteSelected (see page 5941)	Deletes all the selected items.
V	DeleteSelected (see page 21174)	Deletes the selected item from the drop-down list.
V	Destroy (see page 21214)	Destroys the combo box and frees its memory.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
V	Focused (see page 21175)	Indicates whether the combo box has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	MoveSelection (see page 5942)	Moves all selected items to another list control.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectAll (see page 5943)	Selects all items or all text in the selected item.
	SelectAll (see page 21180)	Selects the entire block of text in the edit box.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetDesignVisible (see page 6178)	Makes a control visible at design time.
	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Update (see page 5789)	Processes any pending paint messages immediately.
	Update (see page 6183)	Forces the control to update.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AutoCloseUp (see page 21220)	Specifies whether the drop-down closes up automatically when the user selects an item.
	AutoComplete (see page 21220)	Positions to matching list items as you type.
	AutoCompleteDelay (see page 21220)	Specifies the delay between a key press and an attempt to autocomplete the field.
	AutoDropDown (see page 21221)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 21184)	Provides access to the drawing surface of the combo box.
	CharCase (see page 21221)	Determines the case of the text in the combo box.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DroppedDown (see page 21185)	Indicates whether the drop-down list is currently displayed.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.

R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	Items (see page 21187)	Provides access to the list of items (strings) in the list portion of the combo box.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelLength (see page 21190)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 21193)	Specifies the position of the first selected character in the edit portion of the combo box.
	SelText (see page 21222)	Represents the selected text in the edit region.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TabStop (see page 21196)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

154.1.8.2 TCustomComboBox.Create Constructor

Creates and initializes a TCustomComboBox (see page 21201) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomComboBox(TComponent * AOwner);
```

Description

Call Create to generate a combo box at runtime. Combo boxes placed on forms at design time are created automatically.

AOwner is a component, typically the form, that is responsible for freeing the combo box instance.

See Also

Owner (see page 2807)

154.1.8.3 TCustomComboBox.Destroy Destructor

Destroys the combo box and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomComboBox();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free verifies that the combo box is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

154.1.8.4 TCustomComboBox Methods

The methods of the TCustomComboBox class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 21215)	Initializes the window-creation parameter record.
	CreateWnd (see page 21215)	Creates the Windows control that represents the combo box.
	DestroyWnd (see page 21215)	Destroys the Windows control that represents the combo box.
	DrawItem (see page 21216)	Generates an OnDrawItem (see page 21224) event.
	GetItemCount (see page 21216)	Returns the number of items in the Items (see page 21187) property list.
	GetItemHt (see page 21216)	Returns the value of the ItemHeight (see page 21186) property.
	GetItemsClass (see page 21217)	Returns the class used to implement the Items (see page 21187) property.
	GetSelText (see page 21217)	Returns the value of the SelText (see page 21222) property.
	KeyPress (see page 21217)	Responds to WM_CHAR messages sent to the combo box.
	MeasureItem (see page 21218)	Generates an OnMeasureItem (see page 21225) event.

	SelectItem (see page 21218)	Selects a specified item from the drop-down list.
V	SetStyle (see page 21219)	Sets the Style (see page 21224) property.
V	WndProc (see page 21219)	Processes messages received by the combo box.

154.1.8.4.1 TCustomComboBox.CreateParams Method

Initializes the window-creation parameter record.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 21215) method calls CreateParams to initialize the parameters it uses to create the window for the combo box. CreateParams overrides the inherited method to implement the Style (see page 21224) property.

See Also

CreateWnd (see page 21215)

Style (see page 21224)

154.1.8.4.2 TCustomComboBox.CreateWnd Method

Creates the Windows control that represents the combo box.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

The CreateHandle method calls CreateWnd to handle the particulars of creating the combo box window. TCustomCombo overrides the inherited method to ensure that the newly-created window and the TCustomCombo properties correctly reflect each other.

The Windows control for the combo box can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. Thus, CreateWnd may be called more than once during the lifetime of the TCustomCombo instance.

See Also

CreateHandle (see page 6135)

154.1.8.4.3 TCustomComboBox.DestroyWnd Method

Destroys the Windows control that represents the combo box.

Pascal

```
procedure DestroyWnd; override;
```

C++

```
virtual __fastcall DestroyWnd();
```

Description

The Windows control for the combo box can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. TCustomComboBox (see page 21201) calls DestroyWnd to destroy this Windows control when necessary.

DestroyWnd saves the contents of Items (see page 21187) list before calling the inherited DestroyWnd method. The saved list is used if the window is recreated.

See Also

CreateWnd (see page 21215)

154.1.8.4.4 TCustomComboBox.DrawItem Method

Generates an OnDrawItem (see page 21224) event.

Pascal

```
procedure DrawItem(Index: Integer; Rect: TRect; State: TOwnerDrawState); virtual;
```

C++

```
virtual __fastcall DrawItem(int Index, TRect Rect, TOwnerDrawState State);
```

Description

When Style (see page 21224) is csOwnerDrawFixed or csOwnerDrawVariable, DrawItem is called whenever an item in the list needs to be drawn. If an OnDrawItem (see page 21224) event handler is assigned, DrawItem generates an OnDrawItem (see page 21224) event, passing the Index, Rect, and State parameters to the event handler. Otherwise, OnDrawItem (see page 21224) fills the rectangle passed in Rect and draws any text associated with the indexed item, ignoring the State parameter.

Override DrawItem to provide class-specific drawing code for the items in the list.

See Also

OnDrawItem (see page 21224)

Style (see page 21224)

154.1.8.4.5 TCustomComboBox.GetItemCount Method

Returns the number of items in the Items (see page 21187) property list.

Pascal

```
function GetItemCount: Integer; override;
```

C++

```
virtual __fastcall int GetItemCount();
```

Description

GetItemCount is used internally to quickly determine the number of items in the drop-down list of the combo box.

See Also

Items (see page 21187)

154.1.8.4.6 TCustomComboBox.GetItemHt Method

Returns the value of the ItemHeight (see page 21186) property.

Pascal

```
function GetItemHt: Integer; override;
```

C++

```
virtual __fastcall int GetItemHt();
```

Description

GetItemHt is the protected read implementation of the ItemHeight (see page 21186) property.

See Also

ItemHeight

154.1.8.4.7 TCustomComboBox.GetItemsClass Method

Returns the class used to implement the Items (see page 21187) property.

Pascal

```
function GetItemsClass: TCustomComboBoxStringsClass; override;
```

C++

```
virtual __fastcall TCustomComboBoxStringsClass GetItemsClass();
```

Description

Combo box controls use GetItemsClass internally to determine what class to use when implementing the Items (see page 21187) property. This allows the TCustomCombo (see page 21154) base class to implement the Items (see page 21187) property for both TCustomComboBox (see page 21201) descendants and TCustomComboBoxEx descendants.

See Also

Items (see page 21187)

154.1.8.4.8 TCustomComboBox.GetSelText Method

Returns the value of the SelText (see page 21222) property.

Pascal

```
function GetSelText: string;
```

C++

```
__fastcall AnsiString GetSelText();
```

Description

GetSelText is the protected read implementation of the SelText (see page 21222) property.

See Also

SelText (see page 21222)

154.1.8.4.9 TCustomComboBox.KeyPress Method

Responds to WM_CHAR messages sent to the combo box.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress processes standard keystroke messages for the combo box control. If the KeyPress method does not handle the keystroke, it calls the inherited method, which generates an OnKeyPress event.

Key is the character represented by the key that was pressed. If the KeyPress method changes Key to 0, no further processing occurs for the keystroke.

See Also

KeyDown (see page 6163)

KeyUp (see page 6164)

OnKeyPress (see page 6227)

154.1.8.4.10 TCustomComboBox.MeasureItem Method

Generates an OnMeasureItem (see page 21225) event.

Pascal

```
procedure MeasureItem(Index: Integer; var Height: Integer); virtual;
```

C++

```
virtual __fastcall MeasureItem(int Index, int Height);
```

Description

MeasureItem is the protected implementation method for the OnMeasureItem (see page 21225) event. Override MeasureItem to provide class-specific code to specify the size of list items when Style (see page 21224) is csOwnerDrawVariable.

See Also

OnMeasureItem (see page 21225)

Style (see page 21224)

154.1.8.4.11 TCustomComboBox.SelectItem Method

Selects a specified item from the drop-down list.

Pascal

```
function SelectItem(const AnItem: String): Boolean;
```

C++

```
__fastcall Boolean SelectItem(const AnsiString AnItem);
```

Description

SelectItem selects the first item in the drop-down list that matches the string specified by AnItem. If the list does not contain a string that matches AnItem, SelectItem does nothing.

See Also

Items (see page 21187)

ItemIndex

154.1.8.4.12 TCustomComboBox.SetStyle Method

Sets the Style (see page 21224) property.

Pascal

```
procedure SetStyle(Value: TComboBoxStyle); virtual;
```

C++

```
virtual __fastcall SetStyle(TComboBoxStyle Value);
```

Description

SetStyle is the protected implementation method for the Style (see page 21224) property. Override SetStyle to change the implementation of the Style (see page 21224) property.

See Also

Style (see page 21224)

154.1.8.4.13 TCustomComboBox.WndProc Method

Processes messages received by the combo box.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc is the first method to received Windows messages that are sent to the combo box. Messages (see page 16211) sent to the child windows of the combo box (for the edit region and the list portion) go to the EditWndProc (see page 21175) or ListWndProc (see page 21178) method instead. WndProc handles messages about changes to the system colors defined in the Windows control panel, and closes up the drop-down list when the Enter key or the Escape key is pressed. All other messages are passed to Dispatch for default processing.

See Also

ComboWndProc (see page 21173)

Dispatch

WndProc (see page 6187)

EditWndProc (see page 21175)

ListWndProc (see page 21178)

154.1.8.5 TCustomComboBox Properties

The properties of the TCustomComboBox class are listed here.

Protected Properties

	Name	Description
	Sorted (see page 21222)	Determines whether the list portion of the combo box is alphabetized.
	Style (see page 21224)	Determines the display style of the combo box.

Public Properties

	Name	Description
	AutoCloseUp (see page 21220)	Specifies whether the drop-down closes up automatically when the user selects an item.
	AutoComplete (see page 21220)	Positions to matching list items as you type.
	AutoCompleteDelay (see page 21220)	Specifies the delay between a key press and an attempt to autocomplete the field.
	AutoDropDown (see page 21221)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	CharCase (see page 21221)	Determines the case of the text in the combo box.
	SelText (see page 21222)	Represents the selected text in the edit region.

154.1.8.5.1 TCustomComboBox.AutoCloseUp Property

Specifies whether the drop-down closes up automatically when the user selects an item.

Pascal

```
property AutoCloseUp: Boolean;
```

C++

```
__property Boolean AutoCloseUp;
```

Description

When AutoCloseUp is true, the drop-down closes up automatically when the user selects an item.

See Also

AutoDropDown (see page 21221)

154.1.8.5.2 TCustomComboBox.AutoComplete Property

Positions to matching list items as you type.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

AutoComplete will position to matching list values as you type, saving you keystrokes.

As you type, the ItemIndex property is updated with the list entry that matches the characters you have entered. As you type, different list entries may match, and ItemIndex changes accordingly.

There is a delay of AutoCompleteDelay (see page 21220) milliseconds between the characters typed and the autocomplete operation.

See Also

AutoCompleteDelay (see page 21220)

154.1.8.5.3 TCustomComboBox.AutoCompleteDelay Property

Specifies the delay between a key press and an attempt to autocomplete the field.

Pascal

```
property AutoCompleteDelay: Cardinal;
```

C++

```
__property unsigned AutoCompleteDelay;
```

Description

Specifies the delay in milliseconds between a key press and an attempt to autocomplete the field. Autocomplete will position to matching list values as you type, sometimes saving you keystrokes.

AutoCompleteDelay is only in effect when the AutoComplete (see page 21220) is true.

See Also

AutoComplete (see page 21220)

154.1.8.5.4 TCustomComboBox.AutoDropDown Property

Specifies whether the drop-down list drops down automatically in response to user keystrokes.

Pascal

```
property AutoDropDown: Boolean;
```

C++

```
__property Boolean AutoDropDown;
```

Description

When AutoDropDown is true, the combo box automatically drops down its list when the user starts typing a string while the combo box has focus.

When AutoDropDown is false, the user must explicitly use the drop-down button to drop down the combo box list.

See Also

autocomplete (see page 21220)

OnDropDown (see page 21199)

DroppedDown (see page 21185)

154.1.8.5.5 TCustomComboBox.CharCase Property

Determines the case of the text in the combo box.

Pascal

```
property CharCase: TEditCharCase;
```

C++

```
__property TEditCharCase CharCase;
```

Description

Use CharCase to force the contents of the combo box into upper or lower case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types into the combo box. Changing the CharCase property changes the actual text, not just its appearance; case information is lost and cannot be recaptured by changing CharCase back to ecNormal.

See Also

Text (see page 5842)

154.1.8.5.6 TCustomComboBox.SelText Property

Represents the selected text in the edit region.

Pascal

```
property SelText: string;
```

C++

```
__property AnsiString SelText;
```

Description

Read SelText to obtain the text that is selected in the edit region of the combo box. When the combo box is the active control, SelText appears highlighted. Set SelText to replace the current selection with a different substring. If no text is selected when a value is assigned to SelText, the SelText string is inserted in the text at the cursor.

Note: The SelText property is only valid when the combo box has focus. When the combo box does not have focus, reading SelText always yields an empty string and setting it inserts the text at the cursor rather than replacing the selected text.

See Also

SelLength (see page 21190)

SelStart (see page 21193)

154.1.8.5.7 TCustomComboBox.Sorted Property

Determines whether the list portion of the combo box is alphabetized.

Pascal

```
property Sorted: Boolean;
```

C++

```
__property Boolean Sorted;
```

Description

Set Sorted to true to sort the items in the Items (see page 21187) list alphabetically. New items added to the list while Sorted is true are inserted in the correct alphabetical position.

When Sorted is changed from false to true, the original order of the items is lost. Setting Sorted back to false does not restore the original order.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
```

```
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if (((TForm1 *)Screen->CustomForms[i])->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
    end;
    Items.EndUpdate; {reenable painting }
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.Show;
end;
```

See Also

Items

Using TDBListBox and TDBComboBox

154.1.8.5.8 TCustomComboBox.Style Property

Determines the display style of the combo box.

Pascal

```
property Style: TComboBoxStyle;
```

C++

```
__property TComboBoxStyle Style;
```

Description

Set Style to specify how the list and edit region of the combo box should appear.

When Style is csOwnerDrawFixed or csOwnerDrawVariable, the OnDrawItem (see page 21224) event occurs when the list items need to be drawn. If Style is csOwnerDrawVariable, an OnMeasureItem (see page 21225) event occurs as well.

See Also

Height (see page 5825)

ItemHeight

OnDrawItem (see page 21224)

OnMeasureItem (see page 21225)

154.1.8.6 TCustomComboBox Events

The events of the TCustomComboBox class are listed here.

Protected Events

	Name	Description
	OnDrawItem (see page 21224)	Occurs when an item in an owner-draw combo box needs to be displayed.
	OnMeasureItem (see page 21225)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.

154.1.8.6.1 TCustomComboBox.OnDrawItem Event

Occurs when an item in an owner-draw combo box needs to be displayed.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

Write an OnDrawItem event handler to draw the items in the drop-down list of an owner-draw combo box. An OnDrawItem event handler can add graphic elements to the list items, or replace the list item text by graphics.

Draw the items on the Canvas (see page 21184) using the coordinates provided by the Rect parameter. OnDrawItem occurs only if Style (see page 21224) is set to csOwnerDrawFixed or csOwnerDrawVariable.

If an OnDrawItem event handler is not provided, the combo box fills the Rect parameter with the current brush and writes the text value of the item specified by the Index parameter.

See Also

Canvas

Items

OnMeasureItem (see page 21225)

Style (see page 21224)

154.1.8.6.2 TCustomComboBox.OnMeasureItem Event

Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.

Pascal

```
property OnMeasureItem: TMeasureItemEvent;
```

C++

```
__property TMeasureItemEvent OnMeasureItem;
```

Description

When Style (see page 21224) is set to csOwnerDrawVariable, the OnMeasureItem event precedes OnDrawItem (see page 21224). Write an OnMeasureItem event handler to specify the height, in pixels, needed to draw an item in the drop-down list.

See Also

ItemHeight

Items

OnDrawItem (see page 21224)

Style (see page 21224)

154.1.9 TCustomComboBoxStrings Class

TCustomComboBoxStrings represents the strings in the drop-down list of a combo box.

Class Hierarchy

Pascal

```
TCustomComboBoxStrings = class(TStrings);
```

C++

```
class TCustomComboBoxStrings : public TStrings;
```

File

StdCtrls

Description

TCustomComboBoxStrings is the base class for objects that represent the items in the drop-down list of a combo box. Each type of combo box (TComboBox (see page 21064) or TComboBoxEx) uses its own TCustomComboBoxStrings descendant to manage the list of items.

Do not create instances of TCustomComboBoxStrings. This class is designed to work only with a combo box. TCustomCombo

(see page 21154) creates the TCustomComboBoxStrings descendant it uses to implement its Items property.

See Also

TCustomCombo (see page 21154)

TComboBox (see page 21064)

TComboBoxEx (see page 3463)

154.1.9.1 TCustomComboBoxStrings Members

The following tables list the members exposed by TCustomComboBoxStrings.

Protected Methods

	Name	Description
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	CompareStrings (see page 3048)	Compares two strings.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 3049)	Reads and writes the Strings (see page 3080) property as if it were published.
	Error (see page 3052)	This is the overview for the Error method overload.
	ExtractName (see page 3053)	Returns the name portion of a string that is a name value pair.
V	Get (see page 21229)	Returns a string given its index.
V	GetCapacity (see page 3054)	Returns the currently allocated size of the list of strings.
V	GetCount (see page 21230)	Returns the number of strings in the list
V	GetObject (see page 21230)	Returns the object associated with the item at a specified index.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetTextStr (see page 3056)	Returns the value of the Text (see page 3084) property.
V	Put (see page 3064)	Changes the value of the string with a specified index.
V	PutObject (see page 21231)	Changes the object associated with the string at a specified index.
V	SetCapacity (see page 3068)	Changes the amount of memory allocated to hold strings in the list.
V	SetTextStr (see page 3069)	Sets the Text (see page 3084) property.
V	SetUpdateState (see page 21231)	Performs internal adjustments before or after a series of updates.

Protected Properties

	Name	Description
	ComboBox (see page 21232)	Identifies the combo box whose items this object represents.
R	UpdateCount (see page 3085)	Indicates the number of calls to BeginUpdate (see page 3045) that have not been matched by a call to EndUpdate (see page 3050).

Public Methods

	Name	Description
V	Add (see page 3033)	Adds a string at the end of the list.
V	AddObject (see page 3036)	Adds a string to the list, and associates an object with the string.
V	AddStrings (see page 3039)	Adds a group of strings to the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 3041)	Adds a string to the list.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	Assign (see page 3043)	Sets the strings in the list, and possibly associated objects, from a source object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 3045)	Enables the TStrings (see page 3029) object to track when the list of strings is changing.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 21228)	Deletes all the strings from the list.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Delete (see page 21229)	Removes the item specified by the Index parameter.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 3032)	Destroys the TPersistent instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 3050)	Enables the TStrings (see page 3029) object to keep track of when the list of strings has finished changing.
	Equals (see page 3051)	Compares the list of strings to the list from another TStrings (see page 3029) object and returns true if the two lists match.
V	Exchange (see page 3053)	Swaps the position of two strings in the list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 3054)	Returns a TStrings (see page 3029) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetText (see page 3055)	Allocates a text buffer and fills it with the value of the Text (see page 3084) property.
V	IndexOf (see page 21230)	Returns the position of a string in the list.
V	IndexOfName (see page 3057)	Returns the position of the first name-value pair with the specified name.
V	IndexOfObject (see page 3059)	Returns the index of the first string in the list associated with a given object.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
A	Insert (see page 3059)	Introduces abstract (Delphi) or pure virtual (C++) method to insert a string at a specified position.
V	InsertObject (see page 3060)	Inserts a string into the list at the specified position, and associates it with an object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	LoadFromFile (see page 3061)	Fills the list with the lines of text in a specified file.
V	LoadFromStream (see page 3062)	Fills the list with lines of text read from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	Move (see page 3063)	Changes the position of a string in the list.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SaveToFile (see page 3065)	Saves the strings in the list to the specified file.
V	SaveToStream (see page 3067)	Writes the value of the Text (see page 3084) property to a stream object.
V	SetText (see page 3068)	Sets the Text (see page 3084) property.

Public Properties

	Name	Description
	Capacity (see page 3070)	Indicates the number of strings the TStrings (see page 3029) object can hold.

	CommaText (see page 3070)	Lists the strings in the TStrings (see page 3029) object in system data format (SDF).
R	Count (see page 3071)	Introduces an abstract property to represent the number of strings in the list.
	DelimitedText (see page 3074)	Represents all the strings in the TStrings (see page 3029) object as a single delimited string.
	Delimiter (see page 3074)	Specifies the delimiter used by the DelimitedText (see page 3074) property.
	LineBreak (see page 3075)	Defines the line break character(s).
R	Names (see page 3075)	Indicates the name part of strings that are name-value pairs.
	NameValueSeparator (see page 3077)	Indicates the character used to separate names from values.
	Objects (see page 3077)	Represents a set of objects that are associated one with each of the strings in the Strings (see page 3080) property.
	QuoteChar (see page 3080)	Specifies the quote character used by the DelimitedText (see page 3074) property.
	StrictDelimiter (see page 3080)	Determines how the Delimiter (see page 3074) property is used.
	Strings (see page 3080)	References the strings in the list by their positions.
	StringsAdapter (see page 3084)	Implements an IStringsAdapter (see page 3208) interface for the TStrings (see page 3029) object.
	Text (see page 3084)	Lists the strings in the TStrings (see page 3029) object as a single string with the individual strings delimited by carriage returns and line feeds.
	ValueFromIndex (see page 3085)	Represents the value part of a string with a given index, on strings that are name-value pairs.
	Values (see page 3086)	Represents the value part of a string associated with a given name, on strings that are name-value pairs.

154.1.9.2 TCustomComboBoxStrings Methods

The methods of the TCustomComboBoxStrings class are listed here.

Protected Methods

	Name	Description
V	Get (see page 21229)	Returns a string given its index.
V	GetCount (see page 21230)	Returns the number of strings in the list
V	GetObject (see page 21230)	Returns the object associated with the item at a specified index.
V	PutObject (see page 21231)	Changes the object associated with the string at a specified index.
V	SetUpdateState (see page 21231)	Performs internal adjustments before or after a series of updates.

Public Methods

	Name	Description
V	Clear (see page 21228)	Deletes all the strings from the list.
V	Delete (see page 21229)	Removes the item specified by the Index parameter.
V	IndexOf (see page 21230)	Returns the position of a string in the list.

154.1.9.2.1 TCustomComboBoxStrings.Clear Method

Deletes all the strings from the list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call clear to empty the drop-down list of the combo box. All references to associated objects are also removed. However, the objects themselves are not freed.

See Also

Delete (see page 21229)

Objects

Strings

154.1.9.2.2 TCustomComboBoxStrings.Delete Method

Removes the item specified by the Index parameter.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Call Delete to remove a single item from the list. If an object is associated with the string, the reference to the object is removed as well. Index gives the position of the string in the drop-down list, where 0 is the first string, 1 is the second string, and so on.

See Also

Add (see page 3033)

AddObject (see page 3036)

Clear (see page 21228)

IndexOf (see page 21230)

InsertObject (see page 3060)

Objects

Strings

154.1.9.2.3 TCustomComboBoxStrings.Get Method

Returns a string given its index.

Pascal

```
function Get(Index: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString Get(int Index);
```

Description

Get is the protected read implementation of the Strings property. It fetches the specified string from the combo box identified by the ComboBox (see page 21232) property.

Index identifies the string to return, where 0 is the index of the first string, 1 is the index of the second string, and so on.

See Also

Strings

IndexOf (see page 21230)

ComboBox (see page 21232)

154.1.9.2.4 TCustomComboBoxStrings.GetCount Method

Returns the number of strings in the list

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCount is the protected read implementation of the Count property. It fetches the number of items from the combo box identified by the ComboBox (see page 21232) property.

See Also

Count

ComboBox (see page 21232)

154.1.9.2.5 TCustomComboBoxStrings.GetObject Method

Returns the object associated with the item at a specified index.

Pascal

```
function GetObject(Index: Integer): TObject; override;
```

C++

```
virtual __fastcall TObject * GetObject(int Index);
```

Description

GetObject is the protected read implementation of the Objects property. It fetches the associated object for the identified item from the combo box identified by the ComboBox (see page 21232) property.

Index is the index of the item with which the object is associated.

See Also

Objects

ComboBox (see page 21232)

154.1.9.2.6 TCustomComboBoxStrings.IndexOf Method

Returns the position of a string in the list.

Pascal

```
function IndexOf(const S: string): Integer; override;
```

C++

```
virtual __fastcall int IndexOf(const AnsiString S);
```

Description

Call IndexOf to obtain the position of the first occurrence of a string that matches S. The string must be an exact match (S can't be the prefix of a string in the combo box).

IndexOf returns the 0-based index of the string. Thus, if S matches the first string in the list, IndexOf returns 0, if S is the second

string, IndexOf returns 1, and so on. If the string does not have a match in the string list, IndexOf returns -1.

See Also

IndexOfName (see page 3057)

IndexOfObject (see page 3059)

Strings

154.1.9.2.7 TCustomComboBoxStrings.PutObject Method

Changes the object associated with the string at a specified index.

Pascal

```
procedure PutObject(Index: Integer; AObject: TObject); override;
```

C++

```
virtual __fastcall PutObject(int Index, TObject * AObject);
```

Description

PutObject is the protected write implementation of the Objects property. It associates the object specified by AObject with the item identified by Index.

Index identifies an item in the drop-down list of the combo box, where 0 is the index of the first item, 1 is the index of the second item, and so on.

AObject is the object to associated with the specified item.

See Also

Objects

154.1.9.2.8 TCustomComboBoxStrings.SetUpdateState Method

Performs internal adjustments before or after a series of updates.

Pascal

```
procedure SetUpdateState(Updating: Boolean); override;
```

C++

```
virtual __fastcall SetUpdateState(Boolean Updating);
```

Description

SetUpdateState is called at the beginning or end of a series of updates. When the BeginUpdate method is first called and the TCustomComboBoxStrings (see page 21225) object is not already in the middle of an update, SetUpdateState is called with Updating set to true. When the EndUpdate method is called and it cancels out the last unmatched call to BeginUpdate, SetUpdateState is called with Updating set to false.

SetUpdateState refreshes the combo box when Updating is false, so that the combo box reflects the new set of strings.

See Also

BeginUpdate (see page 3045)

EndUpdate (see page 3050)

154.1.9.3 TCustomComboBoxStrings Properties

The properties of the TCustomComboBoxStrings class are listed here.

Protected Properties

	Name	Description
	ComboBox (see page 21232)	Identifies the combo box whose items this object represents.

154.1.9.3.1 TCustomComboBoxStrings.ComboBox Property

Identifies the combo box whose items this object represents.

Pascal

```
property ComboBox: TCustomCombo;
```

C++

```
__property TCustomCombo ComboBox;
```

Description

TCustomComboBoxStrings (see page 21225) uses ComboBox internally so that the strings in the list and the items in the drop-down list of the combo box remain synchronized.

See Also

Items (see page 21187)

154.1.10 TCustomEdit Class

TCustomEdit is the base class from which all edit-box and memo controls are derived.

Class Hierarchy

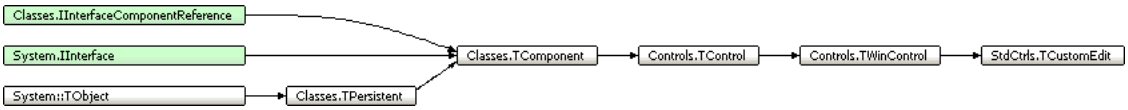

Pascal

```
TCustomEdit = class(TWinControl);
```

C++

```
class TCustomEdit : public TWinControl;
```

File

StdCtrls

Description

TCustomEdit encapsulates the behavior common to all components for editing text by introducing methods and properties that provide:

Basic text editing functions such as selecting text, modifying selected text, and case conversions.

Ability to respond to changes in the contents of the text.

Access control of the text for making it read-only or introducing a password character to hide the actual value.

Do not create instances of TCustomEdit. Use TCustomEdit as a base class when declaring control objects that permit the user to enter or modify text. Properties and methods of TCustomEdit provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TCustomMaskEdit (see page 15798)

TCustomMemo (see page 21362)

TEdit (see page 21402)

154.1.10.1 TCustomEdit Members

The following tables list the members exposed by TCustomEdit.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnChange (see page 21275)	Occurs when the text for the edit control may have changed.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.

	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21245)	Generates an OnChange (see page 21275) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.

	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 21248)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 21248)	Creates a Windows control to represent the control.
V	CreateWnd (see page 21248)	Creates a Windows control corresponding to the edit control.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 21251)	Destroys the Windows control that corresponds to the edit control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
V	DoSetMaxLength (see page 21251)	Sets the MaxLength (see page 21263) property.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.

	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
V	GetSelLength (see page 21252)	Returns the SelLength (see page 21265) property.
V	GetSelStart (see page 21252)	Returns the SelStart (see page 21269) property.
V	GetSelText (see page 21252)	Returns the SelText (see page 21272) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.

	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 21257)	Sets the AutoSize (see page 21260) property.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
V	SetSelLength (see page 21257)	Sets the SelLength (see page 21265) property.
V	SetSelStart (see page 21257)	Sets the SelStart (see page 21269) property.

	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSelect (see page 21260)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 21260)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 21261)	Determines whether the edit control has a single line border around the client area.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	CharCase (see page 21262)	Determines the case of the text within the edit control.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	HideSelection (see page 21262)	Determines whether the visual indication of the selected text remains when focus shifts to another control.

	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MaxLength (see page 21263)	Specifies the maximum number of characters the user can enter into the edit control.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	OEMConvert (see page 21264)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 21264)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PasswordChar (see page 21265)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 21245)	Deletes all text from the edit control.
	ClearSelection (see page 21245)	Removes the selected text from the edit control.
	ClearUndo (see page 21246)	Clears the undo buffer so that no changes to the text can be backed out.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopyToClipboard (see page 21246)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.

	Create (see page 21244)	Creates an instance of TCustomEdit (see page 21232).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CutToClipboard (see page 21249)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
	DefaultHandler (see page 21250)	Provides message handling for all messages that the control does not fully process itself.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetSelTextBuf (see page 21253)	Copies the selected text into a buffer and returns the number of characters copied.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.

	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	PasteFromClipboard (see page 21254)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectAll (see page 21255)	Selects all text in the edit control.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.

V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSelTextBuf (see page 21258)	Replaces the selected text with a null-terminated string.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Undo (see page 21259)	Backs out all changes in the undo buffer.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	CanUndo (see page 21262)	Indicates whether the edit control contains changes that can be backed out.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.

R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Modified (see page 21263)	Indicates whether the user edited the text of the edit control.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ReadOnly (see page 21265)	Determines whether the user can change the text of the edit control.
	SelLength (see page 21265)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 21269)	Specifies the position of the first selected character in the text.
	SelText (see page 21272)	Specifies the selected portion of the edit control's text.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	Text (see page 21273)	Contains a text string associated with the control.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	TabStop (see page 21273)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

154.1.10.2 TCustomEdit.Create Constructor

Creates an instance of TCustomEdit (see page 21232).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomEdit(TComponent * AOwner);
```

Description

Call Create to create and initialize an instance of TCustomEdit (see page 21232).

AOwner is another component, typically the form, that is responsible for freeing the edit control. It becomes the value of the Owner property.

Derived objects that override the Create method should call the inherited Create before performing any additional initialization.

See Also

Owner (see page 2807)

154.1.10.3 TCustomEdit Methods

The methods of the TCustomEdit class are listed here.

Protected Methods

	Name	Description
	Change (see page 21245)	Generates an OnChange (see page 21275) event.
V	CreateParams (see page 21248)	Initializes a window-creation parameter record passed in the Params parameter.
V	CreateWindowHandle (see page 21248)	Creates a Windows control to represent the control.
V	CreateWnd (see page 21248)	Creates a Windows control corresponding to the edit control.
V	DestroyWnd (see page 21251)	Destroys the Windows control that corresponds to the edit control.
V	DoSetMaxLength (see page 21251)	Sets the MaxLength (see page 21263) property.
V	GetSelLength (see page 21252)	Returns the SelLength (see page 21265) property.
V	GetSelStart (see page 21252)	Returns the SelStart (see page 21269) property.
V	GetSelText (see page 21252)	Returns the SelText (see page 21272) property.
V	SetAutoSize (see page 21257)	Sets the AutoSize (see page 21260) property.
V	SetSelLength (see page 21257)	Sets the SelLength (see page 21265) property.
V	SetSelStart (see page 21257)	Sets the SelStart (see page 21269) property.

Public Methods

	Name	Description
V	Clear (see page 21245)	Deletes all text from the edit control.
	ClearSelection (see page 21245)	Removes the selected text from the edit control.
	ClearUndo (see page 21246)	Clears the undo buffer so that no changes to the text can be backed out.
	CopyToClipboard (see page 21246)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
	CutToClipboard (see page 21249)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
V	DefaultHandler (see page 21250)	Provides message handling for all messages that the control does not fully process itself.
V	GetSelTextBuf (see page 21253)	Copies the selected text into a buffer and returns the number of characters copied.
	PasteFromClipboard (see page 21254)	Pastes the contents of the Clipboard into edit control, replacing the current selection.

	SelectAll (see page 21255)	Selects all text in the edit control.
	SetSelTextBuf (see page 21258)	Replaces the selected text with a null-terminated string.
	Undo (see page 21259)	Backs out all changes in the undo buffer.

154.1.10.3.1 TCustomEdit.Change Method

Generates an OnChange (see page 21275) event.

Pascal

```
procedure Change; dynamic;
```

C++

```
__fastcall Change();
```

Description

Applications do not call the Change method directly. It is triggered automatically by Windows events. Change is exposed as a protected method so that derived classes can make additional responses when the text changes beyond invoking the OnChange (see page 21275) event handler. If Derived classes that override the Change method do not call the inherited method, the OnChange (see page 21275) event handler will not be called.

See Also

Handle

OnChange (see page 21275)

Text (see page 5842)

154.1.10.3.2 TCustomEdit.Clear Method

Deletes all text from the edit control.

Pascal

```
procedure Clear; virtual;
```

C++

```
virtual __fastcall Clear();
```

Description

Use Clear to replace the Text (see page 21273) property with an empty string. Clear removes all text, not just the selected text. To remove just the selected text, use ClearSelection (see page 21245).

See Also

ClearSelection (see page 21245)

Text (see page 5842)

154.1.10.3.3 TCustomEdit.ClearSelection Method

Removes the selected text from the edit control.

Pascal

```
procedure ClearSelection;
```

C++

```
__fastcall ClearSelection();
```

Description

Use ClearSelection to delete the selected text from the edit control. If no text is selected, ClearSelection does nothing. If all of the text is selected, ClearSelection clears all text, like the Clear (see page 21245) method.

Note: Calling ClearSelection does the same thing as sending the edit control a WM_CLEAR message.

See Also

Clear (see page 21245)

CutToClipboard (see page 21249)

SelText (see page 21272)

SetSelTextBuf (see page 21258)

154.1.10.3.4 TCustomEdit.ClearUndo Method

Clears the undo buffer so that no changes to the text can be backed out.

Pascal

```
procedure ClearUndo;
```

C++

```
__fastcall ClearUndo();
```

Description

Use ClearUndo to commit changes to the Text (see page 21273) property. After calling ClearUndo, the CanUndo (see page 21262) property is false and the Undo (see page 21259) method does nothing.

See Also

Undo (see page 21259)

CanUndo (see page 21262)

Text (see page 5842)

154.1.10.3.5 TCustomEdit.CopyToClipboard Method

Copies the selected text in the edit control to the Clipboard in CF_TEXT format.

Pascal

```
procedure CopyToClipboard;
```

C++

```
__fastcall CopyToClipboard();
```

Description

Use CopyToClipboard to replace the contents of the Clipboard with the selected text. CopyToClipboard does not clear the Clipboard if no text is selected. If no text is selected, CopyToClipboard does nothing.

Note: Calling CopyToClipboard does the same thing as sending the edit control a WM_COPY message.

C++ Examples:

```
/*
```

```
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, the first
line of text is copied from the rich edit control and pasted
into the edit box.  Note that only the text is pasted. If
the rich edit control includes any formatting information
that is not pasted into the edit control.  If the
destination were a rich edit control, the formatting
information would be copied as well.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->SelectAll();
  RichEdit1->CopyToClipboard();
  Edit1->Clear();
  Edit1->PasteFromClipboard();
  RichEdit1->SetFocus();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("..\\overview.rtf");
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, the first
line of text is copied from the rich edit control and pasted
into the edit box.  Note that only the text is pasted. If
the rich edit control includes any formatting information
that is not pasted into the edit control.  If the
destination were a rich edit control, the formatting
information would be copied as well.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  RichEdit1.SelectAll;
  RichEdit1.CopyToClipboard;
  Edit1.Clear;
  Edit1.PasteFromClipboard;
  RichEdit1.SetFocus;
end;

procedure TForm1.FormCreate(Sender: TObject);
const
  // you may need to change this path to suit your environment
  Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := False;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
end;
```

See Also

CutToClipboard (see page 21249)

GetSelTextBuf (see page 21253)

PasteFromClipboard (see page 21254)

SelText (see page 21272)

Displaying and Editing Text in a Memo Control

154.1.10.3.6 TCustomEdit.CreateParams Method

Initializes a window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 21248) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle (see page 21248). TCustomEdit (see page 21232) overrides the inherited method to specify a scrollable single-line edit control that reflects the current values of the BorderStyle (see page 21261), HideSelection (see page 21262), PasswordChar (see page 21265), and ReadOnly (see page 21265) properties.

See Also

BorderStyle (see page 21261)

CreateWindowHandle (see page 21248)

CreateWnd (see page 21248)

HideSelection (see page 21262)

PasswordChar (see page 21265)

ReadOnly (see page 21265)

154.1.10.3.7 TCustomEdit.CreateWindowHandle Method

Creates a Windows control to represent the control.

Pascal

```
procedure CreateWindowHandle(const Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateWindowHandle(const TCreateParams Params);
```

Description

CreateWindowHandle calls the inherited CreateWindowHandle method after working around a Win95 bug involving setting the read-only style on machines with Far East locales.

See Also

CreateParams (see page 21248)

CreateWnd (see page 21248)

DestroyWnd (see page 21251)

CreateWindowHandle (see page 6137)

154.1.10.3.8 TCustomEdit.CreateWnd Method

Creates a Windows control corresponding to the edit control.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called when the edit control is created, and whenever a change to the edit control requires the window to be recreated. After calling the inherited method, CreateWnd sends the newly created window messages to enforce the MaxLength (see page 21263) and PasswordChar (see page 21265) properties.

See Also

CreateParams (see page 21248)

CreateWindowHandle (see page 6137)

DestroyWnd (see page 21251)

MaxLength (see page 21263)

PasswordChar (see page 21265)

CreateWnd (see page 6137)

154.1.10.3.9 TCustomEdit.CutToClipboard Method

Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.

Pascal

```
procedure CutToClipboard;
```

C++

```
__fastcall CutToClipboard();
```

Description

Use CutToClipboard to replace the contents of the Clipboard with the selected text while clearing the selected text. If no text is selected, CutToClipboard does nothing.

Note: Calling CopyToClipboard (see page 21246) does the same thing as sending the edit control a WM_CUT message.

C++ Examples:

```
/*
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, text is
cut from the rich edit control and pasted into the edit box.
Note that only the text is pasted. If the rich edit control
includes any formatting information that is not pasted into
the edit control.  If the destination where a rich edit
control, the formatting information would be copied as well.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // ensure that all the text is cut, not just the current selection
  RichEdit1->SelectAll();
  RichEdit1->CutToClipboard();
  Edit1->Clear();
  Edit1->PasteFromClipboard();
```

```
  RichEdit1->SetFocus();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const AnsiString Path = "../overview.rtf";
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile(Path);
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, text is
cut from the rich edit control and pasted into the edit box.
Note that only the text is pasted. If the rich edit control
includes any formatting information that is not pasted into
the edit control.  If the destination where a rich edit
control, the formatting information would be copied as well.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  RichEdit1.SelectAll;  { ensure that all the text is cut, not just the current selection }
  RichEdit1.CutToClipboard;
  Edit1.Clear;
  Edit1.PasteFromClipboard;
  RichEdit1.SetFocus;
end;

procedure TForm1.FormCreate(Sender: TObject);
const
  Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := False;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
end;
```

See Also

ClearSelection (see page 21245)

CopyToClipboard (see page 21246)

GetSelTextBuf (see page 21253)

PasteFromClipboard (see page 21254)

SelText (see page 21272)

Displaying and Editing Text in a Memo Control

154.1.10.3.10 TCustomEdit.DefaultHandler Method

Provides message handling for all messages that the control does not fully process itself.

Pascal

```
procedure DefaultHandler(var Message); override;
```

C++

```
virtual __fastcall DefaultHandler( Message);
```

Description

Override DefaultHandler to change the default message handling for the edit control. The Message parameter can be cast to a TMessage type, to obtain the WParam, LParam, and Result of the message. If the Result of the message is non-zero, the message has already been handled. Set the Result field to a non-zero value to prevent further processing of the message by the inherited method.

Note: In Delphi, calling inherited in a message-handling method results in a call to the ancestor's DefaultHandler method if that ancestor does not specify a handler for the message being handled.

See Also

WindowProc (see page 5872)

DefWndProc (see page 6199)

TMessage

154.1.10.3.11 TCustomEdit.DestroyWnd Method

Destroys the Windows control that corresponds to the edit control.

Pascal

```
procedure DestroyWnd; override;
```

C++

```
virtual __fastcall DestroyWnd();
```

Description

The Windows control for the edit box can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. When this happens, DestroyWnd is called.

Before the window is destroyed, DestroyWnd saves a copy of any properties that are needed to recreate the window after it is destroyed.

See Also

CreateWnd (see page 21248)

154.1.10.3.12 TCustomEdit.DoSetMaxLength Method

Sets the MaxLength (see page 21263) property.

Pascal

```
procedure DoSetMaxLength(Value: Integer); virtual;
```

C++

```
virtual __fastcall DoSetMaxLength(int Value);
```

Description

DoSetMaxLength is the property access function for the MaxLength (see page 21263) property. It is exposed as a protected method so that descendants of TCustomEdit (see page 21232) can customize the implementation of the MaxLength (see page 21263) property.

DoSetMaxLength sends an EM_LIMITTEXT message to the window of the edit control.

See Also

MaxLength (see page 21263)

154.1.10.3.13 TCustomEdit.GetSelLength Method

Returns the SelLength (see page 21265) property.

Pascal

```
function GetSelLength: Integer; virtual;
```

C++

```
virtual __fastcall int GetSelLength();
```

Description

GetSelLength is the property access function for the SelLength (see page 21265) property. It is exposed as a protected method so that descendants of TCustomEdit (see page 21232) can customize the implementation of the SelLength (see page 21265) property.

GetSelLength sends an EM_GETSEL message to the window of the edit control and returns the difference between the starting position and the ending position.

See Also

SelLength (see page 21265)

SetSelLength (see page 21257)

154.1.10.3.14 TCustomEdit.GetSelStart Method

Returns the SelStart (see page 21269) property.

Pascal

```
function GetSelStart: Integer; virtual;
```

C++

```
virtual __fastcall int GetSelStart();
```

Description

GetSelStart is the property access method for the SelStart (see page 21269) property. It is exposed as a protected method so that descendants of TCustomEdit (see page 21232) can customize the implementation of the SelStart (see page 21269) property.

GetSelStart sends an EM_GETSEL message to the window of the edit control and returns the starting position.

See Also

SelStart (see page 21269)

SetSelStart (see page 21257)

154.1.10.3.15 TCustomEdit.GetSelText Method

Returns the SelText (see page 21272) property.

Pascal

```
function GetSelText: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetSelText();
```

Description

GetSelText is the property access method for the SelText (see page 21272) property. It is exposed as a protected method so that descendants of TCustomEdit (see page 21232) can customize the implementation of the SelText (see page 21272) property.

See Also

SelText (see page 21272)

154.1.10.3.16 TCustomEdit.GetSelTextBuf Method

Copies the selected text into a buffer and returns the number of characters copied.

Pascal

```
function GetSelTextBuf(Buffer: PChar; BufSize: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int GetSelTextBuf(const char * Buffer, int BufSize);
```

Description

Use GetSelTextBuf to copy the selected text into a character buffer. If there is no selection, the buffer receives an empty string. If the selection contains more than (BufSize - 1) characters, only the first (BufSize - 1) characters are copied. GetSelTextBuf returns the number of characters that were actually copied into the buffer.

C++ Examples:

```
/*
This example copies selected text from one text edit to
another.  This example requires two text edits and two buttons.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->SelLength; //Get length of selected text in Edit1
  Size++;                      //Add room for null character
  char *Buffer = new char[Size];  //Creates Buffer dynamic variable
  Edit1->GetSelTextBuf(Buffer,Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer;
  delete Buffer;
}
//---------------------------------------------------------------------------
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->SelText;
}
```

Delphi Examples:

```
{
This example copies selected text from one text edit to
another.  This example requires two text edits and two buttons.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Integer;
begin
  Size := Edit1.SelLength; {Get length of selected text in Edit1}
  Inc(Size); {Add room for null character}
  GetMem(Buffer, Size); {Creates Buffer dynamic variable}
```

```
  Edit1.GetSelTextBuf(Buffer,Size); {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);{Converts Buffer into Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.SelText;
end;
```

See Also

CopyToClipboard (see page 21246)

GetTextBuf (see page 5766)

SelLength (see page 21265)

SelText (see page 21272)

SetSelTextBuf (see page 21258)

154.1.10.3.17 TCustomEdit.PasteFromClipboard Method

Pastes the contents of the Clipboard into edit control, replacing the current selection.

Pascal

```
procedure PasteFromClipboard;
```

C++

```
__fastcall PasteFromClipboard();
```

Description

Use PasteFromClipboard to replace the selected text with the contents of the Clipboard, or, if no text is selected, to insert the contents of the Clipboard at the cursor. If the Clipboard is empty, or if it does not contain anything in CF_TEXT format, PasteFromClipboard does nothing.

Note: Calling PasteFromClipboard does the same thing as sending the edit control a WM_PASTE message.

C++ Examples:

```
/*
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, the first
line of text is copied from the rich edit control and pasted
into the edit box.  Note that only the text is pasted. If
the rich edit control includes any formatting information
that is not pasted into the edit control.  If the
destination were a rich edit control, the formatting
information would be copied as well.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->SelectAll();
  RichEdit1->CopyToClipboard();
  Edit1->Clear();
  Edit1->PasteFromClipboard();
  RichEdit1->SetFocus();
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("..\\overview.rtf");
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, the first
line of text is copied from the rich edit control and pasted
into the edit box.  Note that only the text is pasted. If
the rich edit control includes any formatting information
that is not pasted into the edit control.  If the
destination were a rich edit control, the formatting
information would be copied as well.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  RichEdit1.SelectAll;
  RichEdit1.CopyToClipboard;
  Edit1.Clear;
  Edit1.PasteFromClipboard;
  RichEdit1.SetFocus;
end;

procedure TForm1.FormCreate(Sender: TObject);
const
  // you may need to change this path to suit your environment
  Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := False;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
end;
```

See Also

CutToClipboard (see page 21249)

SelStart (see page 21269)

SelText (see page 21272)

SetSelTextBuf (see page 21258)

Displaying and Editing Text in a Memo Control

154.1.10.3.18 TCustomEdit.SelectAll Method

Selects all text in the edit control.

Pascal

```
procedure SelectAll;
```

C++

```
__fastcall SelectAll();
```

Description

Use SelectAll to select all text in the edit control. To select only part of the text, use the SelStart (see page 21269) and SelLength (see page 21265) properties.

C++ Examples:

```
/*
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, text is
cut from the rich edit control and pasted into the edit box.
Note that only the text is pasted. If the rich edit control
includes any formatting information that is not pasted into
the edit control.  If the destination where a rich edit
control, the formatting information would be copied as well.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // ensure that all the text is cut, not just the current selection
  RichEdit1->SelectAll();
  RichEdit1->CutToClipboard();
  Edit1->Clear();
  Edit1->PasteFromClipboard();
  RichEdit1->SetFocus();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const AnsiString Path = "../overview.rtf";
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile(Path);
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, text is
cut from the rich edit control and pasted into the edit box.
Note that only the text is pasted. If the rich edit control
includes any formatting information that is not pasted into
the edit control.  If the destination where a rich edit
control, the formatting information would be copied as well.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  RichEdit1.SelectAll;  { ensure that all the text is cut, not just the current selection }
  RichEdit1.CutToClipboard;
  Edit1.Clear;
  Edit1.PasteFromClipboard;
  RichEdit1.SetFocus;
end;

procedure TForm1.FormCreate(Sender: TObject);
const
  Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := False;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
end;
```

See Also

SelLength (see page 21265)

SelStart (see page 21269)

SelText (see page 21272)

Text (see page 5842)

154.1.10.3.19 TCustomEdit.SetAutoSize Method

Sets the AutoSize (see page 21260) property.

Pascal

```
procedure SetAutoSize(Value: Boolean); override;
```

C++

```
virtual __fastcall SetAutoSize(Boolean Value);
```

Description

SetAutoSize is the protected write implementation of the AutoSize (see page 21260) property.

TCustomEdit (see page 21232) overrides the inherited method to update the height of the edit control to reflect the current font when the value of AutoSize (see page 21260) changes to true.

See Also

AutoSize (see page 21260)

154.1.10.3.20 TCustomEdit.SetSelLength Method

Sets the SelLength (see page 21265) property.

Pascal

```
procedure SetSelLength(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetSelLength(int Value);
```

Description

SetSelLength is the property access method for the SelLength (see page 21265) property. It is exposed as a protected method so that descendants of TCustomEdit (see page 21232) can customize the implementation of the SelLength (see page 21265) property.

SetSelLength sends an EM_SETSEL message to the window of the edit control, specifying a selection that begins at SelStart (see page 21269) and ends at SelStart (see page 21269) + Value.

See Also

GetSelLength (see page 21252)

SelLength (see page 21265)

SelStart (see page 21269)

154.1.10.3.21 TCustomEdit.SetSelStart Method

Sets the SelStart (see page 21269) property.

Pascal

```
procedure SetSelStart(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetSelStart(int Value);
```

Description

SetSelStart is the property access method for the SelStart (see page 21269) property. It is exposed as a protected method so that descendants of TCustomEdit (see page 21232) can customize the implementation of the SelStart (see page 21269) property.

SetSelLength (see page 21257) sends an EM_SETSEL message to the window of the edit control to position the cursor on the position indicated by Value.

See Also

GetSelStart (see page 21252)

SelStart (see page 21269)

154.1.10.3.22 TCustomEdit.SetSelTextBuf Method

Replaces the selected text with a null-terminated string.

Pascal

```
procedure SetSelTextBuf(Buffer: PChar);
```

C++

```
__fastcall SetSelTextBuf(const char * Buffer);
```

Description

Use SetSelTextBuf to replace the current selection by the contents of the null-terminated string pointed to by Buffer. If no text is selected, the contents of Buffer are inserted at the cursor. SetSelTextBuf does the same thing as setting the SelText (see page 21272) property, except that it takes a pointer to a char type rather than a string.

C++ Examples:

```
/*
This example inserts the contents of temp_string at the
current location of the cursor.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
   const AnsiString Path = "../readme.txt";
   Memo1->Lines->LoadFromFile(Path);
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char * temp_string = "I am now here";
  Memo1->SetSelTextBuf(temp_string);
}
```

Delphi Examples:

```
{
This example inserts the contents of temp_string at the
current location of the cursor.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  temp_string: array[0..15] of Char;
begin
  temp_string := 'I am now here';
  Memo1.SetSelTextBuf(temp_string);
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
const Path = 'readme.txt';
begin
  Memo1.Lines.LoadFromFile(Path);
end;
```

See Also

GetSelTextBuf (see page 21253)

PasteFromClipboard (see page 21254)

SelText (see page 21272)

SetTextBuf (see page 5787)

154.1.10.3.23 TCustomEdit.Undo Method

Backs out all changes in the undo buffer.

Pascal

```
procedure Undo;
```

C++

```
__fastcall Undo();
```

Description

Call Undo to cancel all changes made to the Text (see page 21273) property since the last call to the ClearUndo (see page 21246) method. If ClearUndo (see page 21246) was never called, Undo rolls back all changes.

To determine whether there are any changes in the undo buffer, check the CanUndo (see page 21262) property.

See Also

ClearUndo (see page 21246)

CanUndo (see page 21262)

Text (see page 5842)

154.1.10.4 TCustomEdit Properties

The properties of the TCustomEdit class are listed here.

Protected Properties

	Name	Description
	AutoSelect (see page 21260)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 21260)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BorderStyle (see page 21261)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 21262)	Determines the case of the text within the edit control.
	HideSelection (see page 21262)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	MaxLength (see page 21263)	Specifies the maximum number of characters the user can enter into the edit control.
	OEMConvert (see page 21264)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentColor (see page 21264)	Determines where a control looks for its color information.

	PasswordChar (see page 21265)	Indicates the character, if any, to display in place of the actual characters typed in the control.

Public Properties

	Name	Description
R	CanUndo (see page 21262)	Indicates whether the edit control contains changes that can be backed out.
	Modified (see page 21263)	Indicates whether the user edited the text of the edit control.
	ReadOnly (see page 21265)	Determines whether the user can change the text of the edit control.
	SelLength (see page 21265)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 21269)	Specifies the position of the first selected character in the text.
	SelText (see page 21272)	Specifies the selected portion of the edit control's text.
	Text (see page 21273)	Contains a text string associated with the control.

Published Properties

	Name	Description
	TabStop (see page 21273)	Determines if the user can tab to a control.

154.1.10.4.1 TCustomEdit.AutoSelect Property

Determines whether all the text in the edit control is automatically selected when the control gets focus.

Pascal

```
property AutoSelect: Boolean;
```

C++

```
__property Boolean AutoSelect;
```

Description

Set AutoSelect to select all the text when the edit control gets focus. AutoSelect only applies to single-line edit controls.

Use AutoSelect when the user is more likely to replace the text in the edit control than to append to it.

See Also

SelectAll (see page 21255)

SelLength (see page 21265)

SelStart (see page 21269)

SelText (see page 21272)

SetSelTextBuf (see page 21258)

154.1.10.4.2 TCustomEdit.AutoSize Property

Determines whether the height of the edit control automatically resizes to accommodate the text.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to make the edit control adjust its size automatically so the client area accommodates the height of the text. When AutoSize is false, the edit control has a fixed height. When AutoSize is true, the size of the control is readjusted whenever a change occurs that could affect the height of the control, such as a change to the font or border style.

Note: AutoSize only adjusts the height of the edit control. Even with AutoSize set to true, text in the edit control may appear truncated if it extends beyond the width of the control.

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit1->AutoSize = true;
  Edit1->Font->Size = 20;
  Edit1->Text = "The edit box is taller now";
}
```

Delphi Examples:

```
{
{
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.AutoSize := True;
  Edit1.Font.Size := 20;
  Edit1.Text := 'The edit box is taller now';
end;
```

See Also

BorderStyle (see page 21261)

ClientHeight (see page 5808)

ControlStyle (see page 5814)

Height (see page 5825)

154.1.10.4.3 TCustomEdit.BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the edit control stands out. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the edit control has a single-line border around the client area. If BorderStyle is bsNone, there will be no border.

154.1.10.4.4 TCustomEdit.CanUndo Property

Indicates whether the edit control contains changes that can be backed out.

Pascal

```
property CanUndo: Boolean;
```

C++

```
__property Boolean CanUndo;
```

Description

Read CanUndo to determine whether the user has made any changes to the text that can be undone by calling the Undo (see page 21259) method. This is useful when enabling or disabling menu items that correspond to these commands.

See Also

Undo (see page 21259)

ClearUndo (see page 21246)

Text (see page 5842)

154.1.10.4.5 TCustomEdit.CharCase Property

Determines the case of the text within the edit control.

Pascal

```
property CharCase: TEditCharCase;
```

C++

```
__property TEditCharCase CharCase;
```

Description

Use CharCase to force the contents of the edit control to assume a particular case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types them into the edit control. Changing the CharCase property to ecLowerCase or ecUpperCase changes the actual contents of the text, not just the appearance. Any case information is lost and can't be recaptured by changing CharCase to ecNormal.

See Also

Text (see page 5842)

154.1.10.4.6 TCustomEdit.HideSelection Property

Determines whether the visual indication of the selected text remains when focus shifts to another control.

Pascal

```
property HideSelection: Boolean;
```

C++

```
__property Boolean HideSelection;
```

Description

Set HideSelection to false to provide visual feedback of the selected portion of the text even when the edit control does not have focus. Set HideSelection to true to show the selection only when the edit control has focus. HideSelection does not affect the

actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms with many edit controls look too busy.

See Also

SelLength (see page 21265)

SelStart (see page 21269)

SelText (see page 21272)

SetSelTextBuf (see page 21258)

154.1.10.4.7 TCustomEdit.MaxLength Property

Specifies the maximum number of characters the user can enter into the edit control.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the edit control. A value of 0 indicates that there is no application-defined limit on the length.

Use MaxLength to limit the length of the text in an edit control if that text will be copied into a fixed-length buffer.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

Note: The unit of MaxLength is characters (not bytes) if the application is using themes.

See Also

Text (see page 5842)

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

154.1.10.4.8 TCustomEdit.Modified Property

Indicates whether the user edited the text of the edit control.

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

Use Modified to determine whether the user changed the Text (see page 21273) property of the edit control. Modified is only reset to **False** when you assign a value to the Text (see page 21273) property. In particular, it is not reset when the control receives focus.

See Also

Change (see page 21245)

OnChange (see page 21275)

SetTextBuf (see page 5787)

Text (see page 5842)

154.1.10.4.9 TCustomEdit.OEMConvert Property

Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.

Pascal

```
property OEMConvert: Boolean;
```

C++

```
__property Boolean OEMConvert;
```

Description

To ensure that any characters typed in the edit control can be unambiguously converted to the OEM character set, set OEMConvert to true. This causes all characters to be converted from ANSI to OEM and then back to ANSI as they are typed. Set OEMConvert to false to avoid the overhead of this conversion when it does not matter whether the text can be unambiguously mapped to an OEM string.

OEMConvert is most useful for edit controls used to enter file names when the application does not use Unicode file names.

See Also

Text (see page 5842)

154.1.10.4.10 TCustomEdit.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

154.1.10.4.11 TCustomEdit.PasswordChar Property

Indicates the character, if any, to display in place of the actual characters typed in the control.

Pascal

```
property PasswordChar: Char;
```

C++

```
__property Char PasswordChar;
```

Description

Use the PasswordChar property to create an edit control that displays a special character in place of any entered text. If PasswordChar is set to the null character (ANSI character zero), the edit control displays its text normally. If PasswordChar is any other character, the edit control displays PasswordChar in place of each character typed. PasswordChar affects the appearance of the edit control only. The value of the Text (see page 21273) property reflects the actual characters that are typed.

See Also

Text (see page 5842)

154.1.10.4.12 TCustomEdit.ReadOnly Property

Determines whether the user can change the text of the edit control.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

Setting ReadOnly to true ensures that the text is not altered, while still allowing the user to select text. The selected text can then be manipulated by the application, or copied to the Clipboard.

See Also

CopyToClipboard (see page 21246)

GetSelTextBuf (see page 21253)

154.1.10.4.13 TCustomEdit.SelLength Property

Specifies the number of characters (bytes) that are selected.

Pascal

```
property SelLength: Integer;
```

C++

```
__property int SelLength;
```

Description

Read SelLength to determine the length, in bytes, of the selected text. This is the same as the number of characters, unless you are using a multi-byte character set. Set SelLength to change the selection to consist of the first SelLength bytes starting at SelStart (see page 21269).

Note: Setting SelLength to a value greater than the number of characters from SelStart (see page 21269) to the end of the text results in the selection of all characters from SelStart (see page 21269) to the end of the text. Reading SelLength immediately after setting it to a value greater than the number of available characters returns the number of characters actually selected, not the value that was just set.

C++ Examples:

```
/*
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FindDialog1->Position = Point(RichEdit1->Left + RichEdit1->Width, RichEdit1->Top);
  FindDialog1->Execute();
}

void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  int FoundAt, StartPos, ToEnd;
  TSearchTypes mySearchTypes;
  // begin the search after the current selection
  // if there is one
  // otherwise, begin at the start of the text
  if (FindDialog1->Options.Contains(frMatchCase))
    mySearchTypes << stMatchCase;
  if (FindDialog1->Options.Contains(frWholeWord))
    mySearchTypes << stWholeWord;
  if (RichEdit1->SelLength)
    StartPos = RichEdit1->SelStart + RichEdit1->SelLength;
  else
    StartPos = 0;
  // ToEnd is the length from StartPos
  // to the end of the text in the rich edit control
  ToEnd = RichEdit1->Text.Length() - StartPos;
  FoundAt = RichEdit1->FindText(FindDialog1->FindText, StartPos, ToEnd, mySearchTypes);
  if (FoundAt != -1)
  {
    RichEdit1->SetFocus();
    RichEdit1->SelStart = FoundAt;
    RichEdit1->SelLength = FindDialog1->FindText.Length();
  }
  else Beep();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("../Overview.RTF");  // starts in Debug
  RichEdit1->ScrollBars = ssVertical;
}
/*
The following OnFind event handler searches a memo component
```

```
for the text specified in the FindText property of a find
dialog component. If found, the first occurrence of the text
in Memo1 is selected. The code uses the Pos function to
compare strings, and stores the number of characters to skip
when determining the selection position in the SkipChars
variable. Because there is no handling of case, whole word,
or search direction in this algorithm, it is assumed that
the Options property of FindDialog1 is set to
[frHideMatchCase, frHideWholeWord, frHideUpDown].
*/
void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  for (int I = 0; I < Memo1->Lines->Count; I++)
  {
    int PosReturn = Memo1->Lines->Strings[I].Pos(FindDialog1->FindText);
    if (PosReturn) //found!
    {
      int Skipchars = 0;
      for (int J = 0; J < I; J++)
        Skipchars += Memo1->Lines->Strings[J].Length();
      Skipchars += I*2; // add CR/LF for all skipped lines
      Skipchars += PosReturn - 1;
      Memo1->SetFocus();
      Memo1->SelStart = Skipchars;
      Memo1->SelLength = FindDialog1->FindText.Length();
      break;
    }
  }
}
```

Delphi Examples:

```
{
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FindDialog1.Position :=
    Point(RichEdit1.Left + RichEdit1.Width, RichEdit1.Top);
  FindDialog1.Execute;
end;

procedure TForm1.FindDialog1Find(Sender: TObject);
var
  FoundAt: LongInt;
  StartPos, ToEnd: Integer;
  mySearchTypes : TSearchTypes;
  myFindOptions : TFindOptions;
begin
  with RichEdit1 do
  begin
    if frMatchCase in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stMatchCase];
    if frWholeWord in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stWholeWord];
    { begin the search after the current selection if there is one }
    { otherwise, begin at the start of the text }
    if SelLength <> 0 then
      StartPos := SelStart + SelLength
    else
      StartPos := 0;
    { ToEnd is the length from StartPos to the end of the
```

```
        text in the rich edit control }
      ToEnd := Length(Text) - StartPos;
      FoundAt :=
        FindText(FindDialog1.FindText, StartPos, ToEnd, mySearchTypes);
      if FoundAt <> -1 then
      begin
        SetFocus;
        SelStart := FoundAt;
        SelLength := Length(FindDialog1.FindText);
      end
      else Beep;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const Path = 'OverView.RTF';
  begin
    RichEdit1.PlainText := False;
    RichEdit1.Lines.LoadFromFile(Path);
    RichEdit1.ScrollBars := ssVertical;
  end;
  {
  The following OnFind event handler searches a memo component
  for the text specified in the FindText property of a find
  dialog component. If found, the first occurrence of the text
  in Memo1 is selected. The code uses the Pos function to
  compare strings, and stores the number of characters to skip
  when determining the selection position in the SkipChars
  variable. Because there is no handling of case, whole word,
  or search direction in this algorithm, it is assumed that
  the Options property of FindDialog1 is set to
  [frHideMatchCase, frHideWholeWord, frHideUpDown].
  }
  procedure TForm1.FindDialog1Find(Sender: TObject);
  var
    I, J, PosReturn, SkipChars: Integer;
  begin
    for I := 0 to Memo1.Lines.Count do
    begin
      PosReturn := Pos(FindDialog1.FindText,Memo1.Lines[I]);
      if PosReturn <> 0 then {found!}
      begin
        SkipChars := 0;
        for J := 0 to I - 1 do
          SkipChars := SkipChars + Length(Memo1.Lines[J]);
        SkipChars := SkipChars + (I*2);
        SkipChars := SkipChars + PosReturn - 1;
        Memo1.SetFocus;
        Memo1.SelStart := SkipChars;
        Memo1.SelLength := Length(FindDialog1.FindText);
        Break;
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const Path = 'country.list';
  begin
    Memo1.Lines.LoadFromFile(Path);
  end;
```

See Also

SelStart (see page 21269)

SelText (see page 21272)

154.1.10.4.14 TCustomEdit.SelStart Property

Specifies the position of the first selected character in the text.

Pascal

```
property SelStart: Integer;
```

C++

```
__property int SelStart;
```

Description

Read SelStart to determine the position of the first selected character, where 0 indicates the first character. If there is no selected text, SelStart indicates the position of the cursor. Set SelStart to remove the current selection and position the cursor just before the indicated character.

To select a particular range of the text, first set SelStart to position the cursor, and then set SelLength (see page 21265) to extend the selection.

C++ Examples:

```
/*
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FindDialog1->Position = Point(RichEdit1->Left + RichEdit1->Width, RichEdit1->Top);
  FindDialog1->Execute();
}

void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  int FoundAt, StartPos, ToEnd;
  TSearchTypes mySearchTypes;
  // begin the search after the current selection
  // if there is one
  // otherwise, begin at the start of the text
  if (FindDialog1->Options.Contains(frMatchCase))
    mySearchTypes << stMatchCase;
  if (FindDialog1->Options.Contains(frWholeWord))
    mySearchTypes << stWholeWord;
  if (RichEdit1->SelLength)
    StartPos = RichEdit1->SelStart + RichEdit1->SelLength;
  else
    StartPos = 0;
  // ToEnd is the length from StartPos
  // to the end of the text in the rich edit control
  ToEnd = RichEdit1->Text.Length() - StartPos;
  FoundAt = RichEdit1->FindText(FindDialog1->FindText, StartPos, ToEnd, mySearchTypes);
  if (FoundAt != -1)
  {
    RichEdit1->SetFocus();
    RichEdit1->SelStart = FoundAt;
    RichEdit1->SelLength = FindDialog1->FindText.Length();
  }
  else Beep();
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("../Overview.RTF");  // starts in Debug
  RichEdit1->ScrollBars = ssVertical;
}
/*
The following OnFind event handler searches a memo component
for the text specified in the FindText property of a find
dialog component. If found, the first occurrence of the text
in Memo1 is selected. The code uses the Pos function to
compare strings, and stores the number of characters to skip
when determining the selection position in the SkipChars
variable. Because there is no handling of case, whole word,
or search direction in this algorithm, it is assumed that
the Options property of FindDialog1 is set to
[frHideMatchCase, frHideWholeWord, frHideUpDown].
*/
void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  for (int I = 0; I < Memo1->Lines->Count; I++)
  {
    int PosReturn = Memo1->Lines->Strings[I].Pos(FindDialog1->FindText);
    if (PosReturn) //found!
    {
      int Skipchars = 0;
      for (int J = 0; J < I; J++)
        Skipchars += Memo1->Lines->Strings[J].Length();
      Skipchars += I*2; // add CR/LF for all skipped lines
      Skipchars += PosReturn - 1;
      Memo1->SetFocus();
      Memo1->SelStart = Skipchars;
      Memo1->SelLength = FindDialog1->FindText.Length();
      break;
    }
  }
}
```

Delphi Examples:

```
{
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FindDialog1.Position :=
    Point(RichEdit1.Left + RichEdit1.Width, RichEdit1.Top);
  FindDialog1.Execute;
end;

procedure TForm1.FindDialog1Find(Sender: TObject);
var
  FoundAt: LongInt;
  StartPos, ToEnd: Integer;
  mySearchTypes : TSearchTypes;
  myFindOptions : TFindOptions;
begin
  with RichEdit1 do
  begin
    if frMatchCase in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stMatchCase];
    if frWholeWord in FindDialog1.Options then
```

```
        mySearchTypes := mySearchTypes + [stWholeWord];
    { begin the search after the current selection if there is one }
    { otherwise, begin at the start of the text }
    if SelLength <> 0 then
      StartPos := SelStart + SelLength
    else
      StartPos := 0;
    { ToEnd is the length from StartPos to the end of the
      text in the rich edit control }
    ToEnd := Length(Text) - StartPos;
    FoundAt :=
      FindText(FindDialog1.FindText, StartPos, ToEnd, mySearchTypes);
    if FoundAt <> -1 then
    begin
      SetFocus;
      SelStart := FoundAt;
      SelLength := Length(FindDialog1.FindText);
    end
    else Beep;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
const Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := False;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
end;
{
The following OnFind event handler searches a memo component
for the text specified in the FindText property of a find
dialog component. If found, the first occurrence of the text
in Memo1 is selected. The code uses the Pos function to
compare strings, and stores the number of characters to skip
when determining the selection position in the SkipChars
variable. Because there is no handling of case, whole word,
or search direction in this algorithm, it is assumed that
the Options property of FindDialog1 is set to
[frHideMatchCase, frHideWholeWord, frHideUpDown].
}
procedure TForm1.FindDialog1Find(Sender: TObject);
var
  I, J, PosReturn, SkipChars: Integer;
begin
  for I := 0 to Memo1.Lines.Count do
  begin
    PosReturn := Pos(FindDialog1.FindText,Memo1.Lines[I]);
    if PosReturn <> 0 then {found!}
    begin
      SkipChars := 0;
      for J := 0 to I - 1 do
        SkipChars := SkipChars + Length(Memo1.Lines[J]);
      SkipChars := SkipChars + (I*2);
      SkipChars := SkipChars + PosReturn - 1;
      Memo1.SetFocus;
      Memo1.SelStart := SkipChars;
      Memo1.SelLength := Length(FindDialog1.FindText);
      Break;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
const Path = 'country.list';
begin
```

```
  Memo1.Lines.LoadFromFile(Path);
end;
```

See Also

SelLength (see page 21265)

SelText (see page 21272)

154.1.10.4.15 TCustomEdit.SelText Property

Specifies the selected portion of the edit control's text.

Pascal

```
property SelText: string;
```

C++

```
__property AnsiString SelText;
```

Description

Read SelText to determine the value of the selected text. Set SelText to replace the selected text with a new string. If there is no selection, but the edit control has focus, set SelText to insert a new string into the text at the cursor.

C++ Examples:

```
/*
This example copies selected text from one text edit to
another.  This example requires two text edits and two buttons.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->SelLength; //Get length of selected text in Edit1
  Size++;                      //Add room for null character
  char *Buffer = new char[Size];  //Creates Buffer dynamic variable
  Edit1->GetSelTextBuf(Buffer,Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer;
  delete Buffer;
}
//---------------------------------------------------------------------------
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->SelText;
}
```

Delphi Examples:

```
{
This example copies selected text from one text edit to
another.  This example requires two text edits and two buttons.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Integer;
begin
  Size := Edit1.SelLength; {Get length of selected text in Edit1}
  Inc(Size); {Add room for null character}
  GetMem(Buffer, Size); {Creates Buffer dynamic variable}
  Edit1.GetSelTextBuf(Buffer,Size); {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);{Converts Buffer into Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
```

```
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.SelText;
end;
```

See Also

ClearSelection (see page 21245)

GetSelTextBuf (see page 21253)

SelLength (see page 21265)

SelStart (see page 21269)

SetSelTextBuf (see page 21258)

Text (see page 5842)

154.1.10.4.16 TCustomEdit.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

154.1.10.4.17 TCustomEdit.Text Property

Contains a text string associated with the control.

Pascal

```
property Text: TCaption;
```

C++

```
__property TCaption Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
  Size++;                        //Add room for null character
  char *Buffer = new char[Size];  //Creates Buffer dynamic variable
  Edit1->GetTextBuf(Buffer,Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer;
  delete [] Buffer;
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
  Inc(Size);                      {Add room for null character}
  GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
  Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
end;
```

See Also

Caption (see page 5806)

Lines (see page 21380)

154.1.10.5 TCustomEdit Events

The events of the TCustomEdit class are listed here.

Protected Events

	Name	Description
	OnChange (see page 21275)	Occurs when the text for the edit control may have changed.

154.1.10.5.1 TCustomEdit.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified (see page 21263) property to see if a change actually occurred. The Text (see page 21273) property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change (see page 21245)

Text (see page 5842)

154.1.11 TCustomGroupBox Class

TCustomGroupBox is the base class for all group-box components.

Class Hierarchy

Pascal

```
TCustomGroupBox = class(TCustomControl);
```

C++

```
class TCustomGroupBox : public TCustomControl;
```

File

StdCtrls

Description

Group boxes are visual components that act as containers for sets of related controls, such as radio buttons. Group boxes may also contain other group boxes. TCustomGroupBox is the class from which all group-box components—including TGroupBox (

see page 21474), TRadioGroup, and TDBRadioGroup—descend.

See Also

TCustomGroupBox

TDBRadioGroup (see page 9283)

TRadioGroup (see page 13115)

TGroupBox (see page 21474)

TCustomControl (see page 5908)

TWidgetControl

154.1.11.1 TCustomGroupBox Members

The following tables list the members exposed by TCustomGroupBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.

	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 21287)	Adjusts the ClientRect property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.

	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 21287)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.

	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 21287)	Renders the onscreen image of the group box.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.

	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.

	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.

	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 21286)	Creates and initializes a TCustomGroupBox (see page 21275) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.

	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.

	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

154.1.11.2 TCustomGroupBox.Create Constructor

Creates and initializes a TCustomGroupBox (see page 21275) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomGroupBox(TComponent * AOwner);
```

Description

Call Create to instantiate a group box at runtime. Group boxes placed on forms at design time are created automatically.

AOwner is the component, typically a form, that is responsible for freeing the group box. It becomes the value of the Owner property.

See Also

Owner (see page 2807)

154.1.11.3 TCustomGroupBox Methods

The methods of the TCustomGroupBox class are listed here.

Protected Methods

	Name	Description
V	AdjustClientRect (see page 21287)	Adjusts the ClientRect property for idiosyncrasies of the window.
V	CreateParams (see page 21287)	Initializes a window-creation parameter record passed in the Params parameter.
V	Paint (see page 21287)	Renders the onscreen image of the group box.

154.1.11.3.1 TCustomGroupBox.AdjustClientRect Method

Adjusts the ClientRect property for idiosyncrasies of the window.

Pascal

```
procedure AdjustClientRect(var Rect: TRect); override;
```

C++

```
virtual __fastcall AdjustClientRect(TRect Rect);
```

Description

AdjustClientRect is called internally when the group box needs accurate information on where to place child controls within the client area. It indents the client rectangle slightly so that child controls are not placed on the frame (or caption) of the group box.

See Also

ClientRect

Caption (see page 5806)

154.1.11.3.2 TCustomGroupBox.CreateParams Method

Initializes a window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a group box creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

Specify the parameter record as the value of the Params parameter. The TCreateParams type is a data structure holding information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TCustomGroupBox (see page 21275) adjusts the window style by turning off the CS_HREDRAW and CS_VREDRAW flags.

See Also

CreateWnd (see page 6137)

CreateWindowHandle (see page 6137)

154.1.11.3.3 TCustomGroupBox.Paint Method

Renders the onscreen image of the group box.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Paint draws a border around the group box and fills in the caption.

154.1.12 TCustomLabel Class

TCustomLabel is the base class for non-windowed controls that display text on a form.

Class Hierarchy

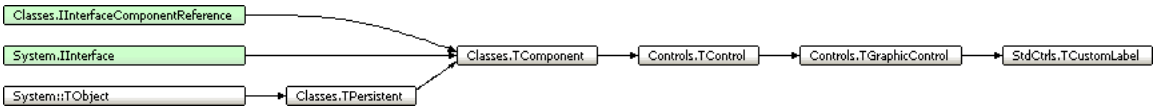

Pascal

```
TCustomLabel = class(TGraphicControl);
```

C++

```
class TCustomLabel : public TGraphicControl;
```

File

StdCtrls

Description

Use TCustomLabel as a base class when defining objects that display text that users can't edit. TCustomLabel introduces several new properties and methods to control the appearance of the text.

TCustomLabel is not a descendant of TWinControl, so it does not have its own window and can't receive direct input from the keyboard. To define an object that responds to direct keyboard input in addition to displaying text, use TCustomStaticText (see page 21387) as a base class.

Do not create instances of TCustomLabel. To put a label on a form, use a TLabel (see page 21544) object.

See Also

TCustomStaticText (see page 21387)

TLabel (see page 21544)

154.1.12.1 TCustomLabel Members

The following tables list the members exposed by TCustomLabel.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.

	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustBounds (see page 21296)	Resizes the label when the text changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.

	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDrawText (see page 21296)	Renders the text in the client area.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetLabelText (see page 21297)	Returns the value of the Caption (see page 21303) property.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 21297)	Adjusts the size of the label when it is first loaded. .
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 21298)	Responds to notifications when objects are about to be created or destroyed.

	Paint (see page 21298)	Draws the image of the label.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 21298)	Sets the AutoSize (see page 21299) property.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	Alignment (see page 21299)	Controls (see page 5711) the horizontal placement of the text within the label.
	AutoSize (see page 21299)	Determines whether the size of the label automatically resizes to accommodate the text.
	Color (see page 5812)	Specifies the background color of the control.

	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EllipsisPosition (see page 21305)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	FocusControl (see page 21306)	Designates a windowed control associated with the label.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	Layout (see page 21306)	Specifies the vertical placement of the text within the label.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ShowAccelChar (see page 21306)	Determines how an ampersand in the label text is displayed.
	Text (see page 5842)	Contains a text string associated with the control.
	Transparent (see page 21307)	Specifies whether controls that sit below the label on a form can be seen through the label.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.
	WordWrap (see page 21308)	Specifies whether the label text wraps when it is too long for the width of the label.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Create (see page 21295)	Creates and initializes an instance of TCustomLabel (see page 21288).
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).

	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Canvas (see page 21300)	Provides the drawing surface used by the graphic control.
	Caption (see page 21303)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.

	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

154.1.12.2 TCustomLabel.Create Constructor

Creates and initializes an instance of TCustomLabel (see page 21288).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomLabel(TComponent * AOwner);
```

Description

Call Create to create an instance of TCustomLabel (see page 21288) at runtime. Labels placed on forms at design time are created automatically.

AOwner is a component, typically the form, that is responsible for freeing the label.

See Also

Owner (see page 2807)

154.1.12.3 TCustomLabel Methods

The methods of the TCustomLabel class are listed here.

Protected Methods

	Name	Description
	AdjustBounds (see page 21296)	Resizes the label when the text changes.
	DoDrawText (see page 21296)	Renders the text in the client area.
V	GetLabelText (see page 21297)	Returns the value of the Caption (see page 21303) property.
V	Loaded (see page 21297)	Adjusts the size of the label when it is first loaded. .
V	Notification (see page 21298)	Responds to notifications when objects are about to be created or destroyed.
V	Paint (see page 21298)	Draws the image of the label.
V	SetAutoSize (see page 21298)	Sets the AutoSize (see page 21299) property.

154.1.12.3.1 TCustomLabel.AdjustBounds Method

Resizes the label when the text changes.

Pascal

```
procedure AdjustBounds; dynamic;
```

C++

```
__fastcall AdjustBounds();
```

Description

When the text in the label or the font changes, AdjustBounds is implicitly called. It redraws the text in the client area, and then resizes the label if the AutoSize (see page 21299) property is true.

When WordWrap (see page 21308) is true, the width of the label is fixed. If AutoSize (see page 21299) is also true, changes to the text cause the label to change in height. When AutoSize (see page 21299) is true and WordWrap (see page 21308) is false, the height of the label is determined by the font, and changes to the text cause the label to change in width.

Override AdjustBounds to make additional adjustments when the text changes.

See Also

DoDrawText (see page 21296)

AutoSize (see page 21299)

154.1.12.3.2 TCustomLabel.DoDrawText Method

Renders the text in the client area.

Pascal

```
procedure DoDrawText(var Rect: TRect; Flags: Longint); dynamic;
```

C++

```
__fastcall DoDrawText(TRect Rect, Longint Flags);
```

Description

When the label needs to draw itself, DoDrawText is implicitly called. DoDrawText uses the Windows API function, DrawText, to render the caption in the client area.

The first parameter, Rect, specifies the boundaries of the text within the client area. The second parameter, Flags, specifies how the text is aligned within those boundaries, how tab characters are handled, etc. See the Microsoft Windows API DrawText function for all the possible values of Flags.

Override DoDrawText to render text differently.

See Also

Caption (see page 5806)

AdjustBounds (see page 21296)

154.1.12.3.3 TCustomLabel.GetLabelText Method

Returns the value of the Caption (see page 21303) property.

Pascal

```
function GetLabelText: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetLabelText();
```

Description

Call GetLabelText to obtain the string that appears as the text of the label.

See Also

Caption (see page 5806)

154.1.12.3.4 TCustomLabel.Loaded Method

Adjusts the size of the label when it is first loaded. .

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

The protected Loaded method is called immediately after a saved version of the form that contains the label is loaded into memory. TCustomLabel (see page 21288) uses this method to adjust the size of the label to the text it contains if the AutoSize (see page 21299) property is true.

See Also

AutoSize (see page 21299)

154.1.12.3.5 TCustomLabel.Notification Method

Responds to notifications when objects are about to be created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are created or destroyed. After calling the inherited method, Notification checks whether the windowed control specified by the FocusControl (see page 21306) property is about to be destroyed. If the windowed control is going away, Notification sets the FocusControl (see page 21306) property to nil (Delphi) or NULL (C++).

See Also

FocusControl (see page 21306)

154.1.12.3.6 TCustomLabel.Paint Method

Draws the image of the label.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Paint is called automatically when the label receives a WM_PAINT message from its Parent. Paint writes the text returned by the GetLabelText (see page 21297) method.

See Also

GetLabelText (see page 21297)

154.1.12.3.7 TCustomLabel.SetAutoSize Method

Sets the AutoSize (see page 21299) property.

Pascal

```
procedure SetAutoSize(Value: Boolean); override;
```

C++

```
virtual __fastcall SetAutoSize(Boolean Value);
```

Description

SetAutoSize is the protected access method for the AutoSize (see page 21299) property. It is exposed as a protected method so descendants can override it to change the implementation of the AutoSize (see page 21299) property.

See Also

AutoSize (see page 21299)

154.1.12.4 TCustomLabel Properties

The properties of the TCustomLabel class are listed here.

Protected Properties

	Name	Description
	Alignment (see page 21299)	Controls (see page 5711) the horizontal placement of the text within the label.
	AutoSize (see page 21299)	Determines whether the size of the label automatically resizes to accommodate the text.
	EllipsisPosition (see page 21305)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	FocusControl (see page 21306)	Designates a windowed control associated with the label.
	Layout (see page 21306)	Specifies the vertical placement of the text within the label.
	ShowAccelChar (see page 21306)	Determines how an ampersand in the label text is displayed.
	Transparent (see page 21307)	Specifies whether controls that sit below the label on a form can be seen through the label.
	WordWrap (see page 21308)	Specifies whether the label text wraps when it is too long for the width of the label.

Public Properties

	Name	Description
	Canvas (see page 21300)	Provides the drawing surface used by the graphic control.
	Caption (see page 21303)	Specifies a text string that identifies the control to the user.

154.1.12.4.1 TCustomLabel.Alignment Property

Controls (see page 5711) the horizontal placement of the text within the label.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Set Alignment to specify how the text of the label is justified within the ClientRect of the label control.

The effect of the Alignment property is more obvious if the WordWrap (see page 21308) property is true and the label includes more than one line of text.

See Also

ClientRect (see page 5810)

GetLabelText (see page 21297)

Layout (see page 21306)

WordWrap (see page 21308)

154.1.12.4.2 TCustomLabel.AutoSize Property

Determines whether the size of the label automatically resizes to accommodate the text.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to make the label adjust its size automatically so the client area accommodates the height and width of the text. When AutoSize is false, the label is fixed in size. When AutoSize is true, the size of the label readjusts whenever the text changes. The size of the label is also readjusts when the Font property changes.

When WordWrap (see page 21308) is true, the width of the label is fixed. If AutoSize is also true, changes to the text cause the label to change in height. When AutoSize is true and WordWrap (see page 21308) is false, the font determines the height of the label, and changes to the text cause the label to change in width.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
```

See Also

BoundsRect (see page 5803)

GetLabelText (see page 21297)

Font (see page 5823)

WordWrap (see page 21308)

Layout (see page 21306)

154.1.12.4.3 TCustomLabel.Canvas Property

Provides the drawing surface used by the graphic control.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control.

Since a TGraphicControl instance does not have its own Windows screen object, it obtains its device context from its parent control.

C++ Examples:

```
/*
When you do custom drawing, such as shown in the code for
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   PaintBox1->Canvas->Brush->Color = clRed;
   PaintBox1->Canvas->FillRect(PaintBox1->Canvas->ClipRect);
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}

/*
In order for the drawing to be persistent, you need to also
include an OnPaint event handler that tells the PaintBox how
to redraw itself when it needs to. In the code below, an
ellipse will be drawn everytime the PaintBox renders itself,
but the PaintBox will only paint the client area of the
control red after the Button is clicked. This client area
will only stay red until it is invalidated, but the ellipse
will persist. This behavior is different from the TImage
component, because the TImage component maintains an
internal bitmap that stores this drawing information for
you.
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
   PaintBox1->Canvas->Brush->Color = clBlue;
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TLabel *myLabel;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TLabel")
  {
    myLabel = (TLabel *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myLabel->Canvas->MoveTo(myLabel->ClientRect.Left, myLabel->ClientRect.Top);
    myLabel->Canvas->LineTo(myLabel->ClientRect.Right, myLabel->ClientRect.Bottom);
    myLabel = NULL;
  }
}
/*
The following code draws the graphic in Picture1 in the
```

```
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PaintBox1->Canvas->Draw(0,0, Image1->Picture->Graphic);
}
```

Delphi Examples:

```
{
When you do custom drawing, such as shown in the code for
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   PaintBox1.Canvas.Brush.Color := clRed;
   PaintBox1.Canvas.FillRect(PaintBox1.Canvas.ClipRect);
   PaintBox1.Canvas.Brush.Color := clBlue;
   PaintBox1.Canvas.Ellipse(0,0,100,50);
end;
{
In order for the drawing to be persistent, you need to also
include an OnPaint event handler that tells the PaintBox how
to redraw itself when it needs to. In the code below, an
ellipse will be drawn everytime the PaintBox renders itself,
but the PaintBox will only paint the client area of the
control red after the Button is clicked. This client area
will only stay red until it is invalidated, but the ellipse
will persist. This behavior is different from the TImage
component, because the TImage component maintains an
internal bitmap that stores this drawing information for
you.
}
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
   PaintBox1.Canvas.Brush.Color := clBlue;
   PaintBox1.Canvas.Ellipse(0,0,100,50);
end;
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TLabel) then
  begin
    myLabel:= (ListBox1.Items.Objects[Index] as TLabel);
    with myLabel.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myLabel.Canvas.MoveTo(Left, Top);
      myLabel.Canvas.LineTo(Right, Bottom);
    end;
  myLabel:= nil;
  end;
{
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  PaintBox1.Canvas.Draw(0,0, Image1.Picture.Graphic);
end;
```

154.1.12.4.4 TCustomLabel.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
```

```
      button->Parent = toolBar;
      button->Caption = captions->Strings[i];
      if (button->Caption == "|")
        button->Style = tbsSeparator;
      else
        button->Style = tbsButton;
    }
  }

  TToolBar *myToolBar;

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    myToolBar = new TToolBar(this);
    myToolBar->Parent = this;
    TStringList *captions = new TStringList();
    captions->Add("New");
    captions->Add("Save");
    captions->Add("|");
    captions->Add("Cut");
    captions->Add("Copy");
    captions->Add("Paste");
    AddButtons(myToolBar, captions);
    delete captions;
    myToolBar->ShowCaptions = True;
    myToolBar->Height = 40;
  }

  void __fastcall TForm1::FormDestroy(TObject *Sender)
  {
    for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
      delete myToolBar->Buttons[I];
    delete myToolBar;
  }
```

Delphi Examples:

```
  {
  The following code keeps the size of the label control
  constant, even though the length of the label's caption
  changes. As a result, the caption of the label is probably
  too long to display in the label when the user clicks the
  button:
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Label1.AutoSize := False;
    Label1.Caption :=
      'This string is too long as the caption of this label';
  end;
  {
  This example requires a TPageControl populated by several
  TabSheets, a TEdit and a TButton. When the button is
  clicked, the caption of the active tab changes to that of
  the TEdit.  These lines of code could also be written:
  PageControl1.ActivePage.Caption := Edit1.Text;
  This would obviate the need for a TTabSheet variable since
  PageControl1.->ActivePage is already of type TTabSheet.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    sheet: TTabSheet;
  begin
    sheet := PageControl1.ActivePage;
    sheet.Caption := Edit1.Text;
  end;
  {
```

```
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

154.1.12.4.5 TCustomLabel.EllipsisPosition Property

Specifies how ellipsis is placed in text not fitting in allocated rectangle.

Pascal

```
property EllipsisPosition: TEllipsisPosition;
```

C++

```
__property TEllipsisPosition EllipsisPosition;
```

Description

Set EllipsisPosition to determine how an ellipsis is placed in text that does not fit in its allocated rectangle. This is a TEllipsisPosition (see page 21947) constant and its value corresponds to constants specifying ellipsis placement in the **dwDTFormat** parameter of the Windows **DrawText** and **DrawTextEx** functions.

See Also

WordWrap (see page 21308)

TypeName

154.1.12.4.6 TCustomLabel.FocusControl Property

Designates a windowed control associated with the label.

Pascal

```
property FocusControl: TWinControl;
```

C++

```
__property TWinControl * FocusControl;
```

Description

Set FocusControl to the windowed control that should receive focus when the user presses the accelerator key specified by the label. Specify an accelerator key by preceding a character in the label text with an ampersand (&) and setting the ShowAccelChar (see page 21306) property to true.

See Also

ShowAccelChar (see page 21306)

GetLabelText (see page 21297)

154.1.12.4.7 TCustomLabel.Layout Property

Specifies the vertical placement of the text within the label.

Pascal

```
property Layout: TTextLayout;
```

C++

```
__property TTextLayout Layout;
```

Description

Set Layout to specify how the text of the label is placed within the ClientRect of the label control. Layout is the vertical analog to the Alignment (see page 21299) property.

See Also

Alignment (see page 21299)

ClientRect (see page 5810)

154.1.12.4.8 TCustomLabel.ShowAccelChar Property

Determines how an ampersand in the label text is displayed.

Pascal

```
property ShowAccelChar: Boolean;
```

C++

```
__property Boolean ShowAccelChar;
```

Description

Set ShowAccelChar to true to allow the label to display an underlined accelerator key value. When ShowAccelChar is true, any character preceded by an ampersand (&) appears underlined. If the FocusControl (see page 21306) property is set, the windowed control specified by the FocusControl (see page 21306) property receives input focus when the user types that underlined character. To display an ampersand when ShowAccelChar is true, use two ampersands (&&) to stand for the single

ampersand that is displayed.

Set ShowAccelChar to false to display the label text with all ampersands appearing as ampersands. When ShowAccelChar is false, the value of the FocusControl (see page 21306) property is not used.

C++ Examples:

```
/*
This example uses two labels on a form. The first label has
a caption with an accelerator character in it. The second
label also includes an ampersand, but it does not appear as
an accelerator character.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Label1->ShowAccelChar = true;
  Label1->Caption = "An &Underlined character appears here";
  Label2->ShowAccelChar = false;
  Label2->Caption = "An ampersand (&) appears here";
}
```

Delphi Examples:

```
{
This example uses two labels on a form. The first label has
a caption with an accelerator character in it. The second
label also includes an ampersand, but it does not appear as
an accelerator character.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Label1.ShowAccelChar := True;
  Label1.Caption := 'An &Underlined character appears here';
  Label2.ShowAccelChar := False;
  Label2.Caption := 'An ampersand (&) appears here';
end;
```

See Also

FocusControl (see page 21306)

GetLabelText (see page 21297)

154.1.12.4.9 TCustomLabel.Transparent Property

Specifies whether controls that sit below the label on a form can be seen through the label.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Set Transparent to true to prevent the label from obscuring other controls on the form. For example, if the label is used to add text to a graphic, set Transparent to true so that the label does not stand out as a separate object.

Writing text so that the background is transparent is slower than writing text when Transparent is false. If the label is not obscuring a complicated image, performance can be improved by setting the background color of the label to match the object beneath it and setting Transparent to false.

See Also

Color (see page 5812)

ParentColor (see page 5834)

154.1.12.4.10 TCustomLabel.WordWrap Property

Specifies whether the label text wraps when it is too long for the width of the label.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the label to display multiple line of text. When WordWrap is true, text that is too wide for the label control wraps at the right margin and continues in additional lines.

Set WordWrap to false to limit the label to a single line. When WordWrap is false, text that is too wide for the label appears truncated.

See Also

ClientWidth (see page 5811)

GetLabelText (see page 21297)

154.1.13 TCustomListBox Class

TCustomListBox is the base class for listbox objects that are wrappers for list box controls.

Class Hierarchy

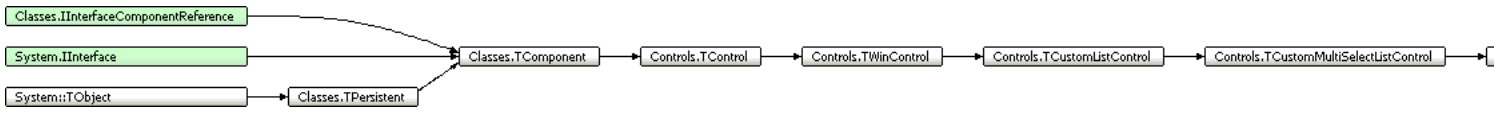

Pascal

```
TCustomListBox = class(TCustomMultiSelectListControl);
```

C++

```
class TCustomListBox : public TCustomMultiSelectListControl;
```

File

StdCtrls

Description

TCustomListBox encapsulates behavior common to all list box controls. List box objects are used to display a scrollable list from which users can select one or more items. Use a list box control to limit the user to selecting from a list of items. Items (see page 21348) can be added at design time or at runtime.

As a wrapper for a list box TCustomListBox introduces methods for:

Displaying, sorting, selecting, manipulating, and querying items in the list.

Creating owner-draw style list boxes that can display graphical images such as icons and bitmaps.

Creating multiple-column and multiple-selection lists.

By default, TCustomListBox is a standard list box that displays items as text strings. These items are contained in a TStrings object, and the methods of the TStrings object are used to add, insert or delete a string. The following code demonstrates adding a new string to the list box:

MyListBox.Items.Add('New item'); //Delphi

MyListBox->Items (see page 21348)->Add("New item"); // C++

Descendants of TCustomListBox inherit the basic functionality of TCustomListBox. Some descendants also have specialized behavior and can be linked to file, directory, drive, or database information. Use these list boxes to easily access and list this type of information.

A combo box is a list box with an edit control and an optional drop down style for the list. Use a combo box to allow users to type their selection into an edit box or to save space on a form by having the list drop down.

For lists of items that require a matrix or expandable tree-like structure, use list views or tree views. These controls allow the grouping of items under column headings or in expandable and contractible rows.

See Also

TStrings (see page 3029)

154.1.13.1 TCustomListBox Members

The following tables list the members exposed by TCustomListBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnData (see page 21357)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 21357)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 21358)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 21358)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.

	OnMeasureItem (see page 21360)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FMoving (see page 21321)	This is FMoving, a member of class TCustomListBox.
	FMultiSelect (see page 5956)	This is FMultiSelect, a member of class TCustomMultiSelectListControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.

	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 21324)	Initializes the window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 21325)	Creates a Windows control corresponding to the list box component.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DeleteString (see page 21326)	Deletes a string from the list box.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 21326)	Saves the list box items in a string list before freeing the underlying Windows control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFindData (see page 21326)	Returns the index of a string in the list box.
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetData (see page 21327)	Returns a string in the list box given its index.
	DoGetDataObject (see page 21327)	Returns the object associated with an item in a virtual list box.

	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
V	DragCanceled (see page 21328)	Sends a mouse message canceling the drag-and-drop operation.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawItem (see page 21328)	Generates an OnDrawItem (see page 21358) event.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetCount (see page 21329)	Returns the value of the Count (see page 21346) property.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetItemData (see page 21329)	Returns a 32-bit value associated with the specified item in a list box.
V	GetItemIndex (see page 21329)	Returns the value of the ItemIndex property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
V	GetSelCount (see page 21330)	Returns the number of selected items.
	GetSelected (see page 21330)	Indicates whether a particular item is selected.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.

	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InternalGetItemData (see page 21330)	Retrieves the data associated with an item in the list box.
	InternalSetItemData (see page 21330)	Associates an integer with a specified item in the list box.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
V	KeyPress (see page 21334)	Responds when the user presses a key.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	LoadRecreateItems (see page 21334)	This is LoadRecreateItems, a member of class TCustomListBox.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
V	MeasureItem (see page 21335)	Generates an OnMeasureItem (see page 21360) event.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetContent (see page 21335)	Removes all of the items from a list box.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
V	SaveRecreateItems (see page 21336)	This is SaveRecreateItems, a member of class TCustomListBox.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.

	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItemData (see page 21336)	Sets the 32-bit value associated with the specified item in a list box.
V	SetItemIndex (see page 21336)	Gives focus to the item with a specified index.
V	SetMultiSelect (see page 21337)	Sets the MultiSelect property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 21337)	Handles Windows messages.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 21343)	Determines whether the list box has a border.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Columns (see page 21346)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	ExtendedSelect (see page 21346)	This is ExtendedSelect, a member of class TCustomListBox.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 21347)	Determines whether the list box displays the partial items.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	ItemHeight (see page 21347)	Specifies the height in pixels of an item in an owner-draw list box.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 21350)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Sorted (see page 21353)	Specifies whether the items in a list box are arranged alphabetically.
	Style (see page 21354)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabWidth (see page 21355)	Specifies the size of the tabs in the list box.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AddItem (see page 21322)	Adds an item to the list box
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 21322)	Deletes all items from the list box.
V	ClearSelection (see page 21323)	Deselects all selected items in the list box, if possible.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	CopySelection (see page 21323)	Copies the selected items to another list-type control
V	Create (see page 21320)	Instantiates a list box control.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	DeleteSelected (see page 21325)	Deletes the selected items from the list box
V	Destroy (see page 21320)	Destroys the instance of a list box control.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ItemAtPos (see page 21331)	Returns the index of the list box item indicated by the coordinates of a point on the control.
	ItemRect (see page 21333)	Returns the rectangle that surrounds the item specified in the Item parameter.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
V	SelectAll (see page 21336)	Selects the entire block of text in the list box.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 21342)	Determines whether the user can give focus to items by typing in the list.
	AutoCompleteDelay (see page 21342)	Specifies the delay between a key press and an attempt to autocomplete the field.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 21343)	Provides a drawing surface when implementing a handler for the OnDrawItem (see page 21358) event of an owner-draw list box.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.

	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Count (see page 21346)	Indicates the number of items in the list box.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	Items (see page 21348)	Contains the strings that appear in the list box.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	MultiSelect (see page 5957)	Specifies whether the user can select more than one item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ScrollWidth (see page 21351)	Specifies the width, in pixels, by which the list box can scroll horizontally.
R	SelCount (see page 5957)	Indicates the number of selected items.
	Selected (see page 21351)	Indicates whether a particular item is selected.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TopIndex (see page 21356)	Specifies the index number of the item that appears at the top of the list box.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.

	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	TabStop (see page 21355)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

154.1.13.2 TCustomListBox.Create Constructor

Instantiates a list box control.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomListBox(TComponent * AOwner);
```

Description

Call Create to instantiate a list box control at runtime. For list boxes created at design time, Create is called automatically.AOwner is the component that is responsible for freeing the list box. It becomes the value of the Owner property. If AOwner is nil, the caller is responsible for explicitly freeing the control.

See Also

Owner (see page 2807)

154.1.13.3 TCustomListBox.Destroy Destructor

Destroys the instance of a list box control.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomListBox();
```

Description

Do not call Destroy directly in an application. Instead, an application should call Free. Free checks that the list box reference is not nil before it calls Destroy.

Destroy frees the memory associated with the objects for the Canvas (see page 21343) and Items (see page 21348) properties. Then it calls the inherited destructor to destroy the list box object.

See Also

Free (see page 22300)

154.1.13.4 TCustomListBox Fields

The fields of the TCustomListBox class are listed here.

Protected Fields

	Name	Description
	FMoving (see page 21321)	This is FMoving, a member of class TCustomListBox.

154.1.13.4.1 TCustomListBox.FMoving Field

Pascal

```
FMoving: Boolean;
```

C++

```
Boolean FMoving;
```

Description

This is FMoving, a member of class TCustomListBox.

154.1.13.5 TCustomListBox Methods

The methods of the TCustomListBox class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 21324)	Initializes the window-creation parameter record passed in the Params parameter.
	CreateWnd (see page 21325)	Creates a Windows control corresponding to the list box component.
	DeleteString (see page 21326)	Deletes a string from the list box.
	DestroyWnd (see page 21326)	Saves the list box items in a string list before freeing the underlying Windows control.
	DoFindData (see page 21326)	Returns the index of a string in the list box.
	DoGetData (see page 21327)	Returns a string in the list box given its index.
	DoGetDataObject (see page 21327)	Returns the object associated with an item in a virtual list box.
	DragCanceled (see page 21328)	Sends a mouse message canceling the drag-and-drop operation.
	DrawItem (see page 21328)	Generates an OnDrawItem (see page 21358) event.
	GetCount (see page 21329)	Returns the value of the Count (see page 21346) property.
	GetItemData (see page 21329)	Returns a 32-bit value associated with the specified item in a list box.
	GetItemIndex (see page 21329)	Returns the value of the ItemIndex property.

	GetSelCount (see page 21330)	Returns the number of selected items.
	GetSelected (see page 21330)	Indicates whether a particular item is selected.
	InternalGetItemData (see page 21330)	Retrieves the data associated with an item in the list box.
	InternalSetItemData (see page 21330)	Associates an integer with a specified item in the list box.
	KeyPress (see page 21334)	Responds when the user presses a key.
	LoadRecreateItems (see page 21334)	This is LoadRecreateItems, a member of class TCustomListBox.
	MeasureItem (see page 21335)	Generates an OnMeasureItem (see page 21360) event.
	ResetContent (see page 21335)	Removes all of the items from a list box.
	SaveRecreateItems (see page 21336)	This is SaveRecreateItems, a member of class TCustomListBox.
	SetItemData (see page 21336)	Sets the 32-bit value associated with the specified item in a list box.
	SetItemIndex (see page 21336)	Gives focus to the item with a specified index.
	SetMultiSelect (see page 21337)	Sets the MultiSelect property.
	WndProc (see page 21337)	Handles Windows messages.

Public Methods

	Name	Description
	AddItem (see page 21322)	Adds an item to the list box
	Clear (see page 21322)	Deletes all items from the list box.
	ClearSelection (see page 21323)	Deselects all selected items in the list box, if possible.
	CopySelection (see page 21323)	Copies the selected items to another list-type control
	DeleteSelected (see page 21325)	Deletes the selected items from the list box
	ItemAtPos (see page 21331)	Returns the index of the list box item indicated by the coordinates of a point on the control.
	ItemRect (see page 21333)	Returns the rectangle that surrounds the item specified in the Item parameter.
	SelectAll (see page 21336)	Selects the entire block of text in the list box.

154.1.13.5.1 TCustomListBox.AddItem Method

Adds an item to the list box

Pascal

```
procedure AddItem(Item: String; AObject: TObject); override;
```

C++

```
virtual __fastcall AddItem(AnsiString Item, TObject * AObject);
```

Description

Call AddItem to add a new item to the end of the list.

Item is the text of the item to add.

AObject is the object associated with the new item.

See Also

AddObject (see page 3036)

Items (see page 21348)

154.1.13.5.2 TCustomListBox.Clear Method

Deletes all items from the list box.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Use Clear to delete all of the items in the list box at once. Although the items in a standard list box are of type TStrings, use the Clear method of the list box rather than calling the Clear method of the Items (see page 21348) property. This allows descendants of TCustomListBox (see page 21308) to perform any other necessary clean-up in the Clear method, in addition to deleting the items from the Items (see page 21348) property. Thus, applications should use

```
ListBox1.Clear;
ListBox1->Clear();
```

rather than

```
ListBox1.Items.Clear;
ListBox1->Items->Clear();
```

See Also

Clear (see page 3047)

Items (see page 21348)

154.1.13.5.3 TCustomListBox.ClearSelection Method

Deselects all selected items in the list box, if possible.

Pascal

```
procedure ClearSelection; override;
```

C++

```
virtual __fastcall ClearSelection();
```

Description

Use ClearSelection to deselect all the currently selected items in the list box. When the list box receives focus, after ClearSelection is called, the last item that was selected is the item that receives the current focus in the list box. Although this item receives focus, the item itself is not selected.

See Also

DeleteSelected (see page 21325)

extendedselect (see page 21346)

MultiSelect

SelCount

Selected (see page 21351)

ItemIndex

154.1.13.5.4 TCustomListBox.CopySelection Method

Copies the selected items to another list-type control

Pascal

```
procedure CopySelection(Destination: TCustomListControl); override;
```

C++

```
virtual __fastcall CopySelection(TCustomListControl Destination);
```

Description

Call CopySelection to copy all the items selected in the list box to a combo box, list view, or another list box. If the destination control is not sorted, the selected items are added to the end of its list.

Destination is the list box, combo box, or list view to which the selected items are copied.

See Also

MoveSelection (see page 5942)

extendedselect (see page 21346)

MultiSelect

SelCount

Selected (see page 21351)

ItemIndex

154.1.13.5.5 TCustomListBox.CreateParams Method

Initializes the window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

CreateParams is used to pass information, in its Params parameter, to CreateWindowHandle when the list box control is created.

TCustomListBox (see page 21308) overrides CreateParams to customize the representation of the window. It uses information from the list box Columns (see page 21346), ExtendedSelect (see page 21346), IntegralHeight (see page 21347), MultiSelect, Sorted (see page 21353) and Style (see page 21354) properties and sets WS_HSCROLL, WS_VSCROLL, LBS_HASSTRINGS and LBS_NOTIFY Windows styles for creation of the list box.

When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

See Also

Columns (see page 21346)

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

extendedselect (see page 21346)

IntegralHeight (see page 21347)

MultiSelect

Sorted (see page 21353)

Style (see page 21354)

154.1.13.5.6 TCustomListBox.CreateWnd Method

Creates a Windows control corresponding to the list box component.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the list box must be displayed. CreateWnd calls the inherited CreateWnd method, which calls

CreateParams (see page 21324) to initialize the window-creation parameters.

CreateWindowHandle to create the window handle for the control.

CreateWnd additionally sets up the list box position, tabs, items, and columns.

See Also

Columns (see page 21346)

CreateHandle (see page 6135)

CreateParams (see page 21324)

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

ItemIndex

Items (see page 21348)

TabWidth (see page 21355)

TopIndex (see page 21356)

154.1.13.5.7 TCustomListBox.DeleteSelected Method

Deletes the selected items from the list box

Pascal

```
procedure DeleteSelected; override;
```

C++

```
virtual __fastcall DeleteSelected();
```

Description

Call DeleteSelected to delete the selected items from the list box.

See Also

ClearSelection (see page 5940)

MoveSelection (see page 5942)

CopySelection (see page 21323)

extendedselect (see page 21346)

MultiSelect

SelCount

Selected (see page 21351)

ItemIndex

154.1.13.5.8 TCustomListBox.DeleteString Method

Deletes a string from the list box.

Pascal

```
procedure DeleteString(Index: Integer); dynamic;
```

C++

```
__fastcall DeleteString(int Index);
```

Description

DeleteString deletes the string, specified by the Index parameter, from the list box control.

154.1.13.5.9 TCustomListBox.DestroyWnd Method

Saves the list box items in a string list before freeing the underlying Windows control.

Pascal

```
procedure DestroyWnd; override;
```

C++

```
virtual __fastcall DestroyWnd();
```

Description

DestroyWnd saves the list box text string items in the Items (see page 21348) property to internal storage in a TStringList object that it creates. It then calls the inherited DestroyWnd method to destroy the windowed control's window handle and free associated device contexts.

See Also

DestroyWnd (see page 6141)

Items (see page 21348)

TStringList (see page 3006)

154.1.13.5.10 TCustomListBox.DoFindData Method

Returns the index of a string in the list box.

Pascal

```
function DoFindData(const Data: String): Integer;
```

C++

```
__fastcall int DoFindData(const AnsiString Data);
```

Description

This protected method is called automatically when the list box needs to indicate the index for an item.

If the list box is virtual (that is, if the value of the Style (see page 21354) property is lbVirtual or lbVirtualOwnerDraw), DoFindData generates an OnDataFind (see page 21357) event so that the index can be provided dynamically. If the list box is not virtual (any other Style (see page 21354)), DoFindData locates the specified string from the Items (see page 21348) property and returns its index.

Data is the text of the item to locate.

See Also

OnDataFind (see page 21357)

Style (see page 21354)

Items (see page 21348)

154.1.13.5.11 TCustomListBox.DoGetData Method

Returns a string in the list box given its index.

Pascal

```
function DoGetData(const Index: Integer): String;
```

C++

```
__fastcall AnsiString DoGetData(const int Index);
```

Description

This protected method is called automatically when a virtual list box needs to supply the text of an item. DoGetData is never called unless the list box is virtual (that is, unless the value of the Style (see page 21354) property is lbVirtual or lbVirtualOwnerDraw).

DoGetData generates an OnData (see page 21357) event and returns the string that is supplied dynamically by the event handler.

See Also

OnData (see page 21357)

Style (see page 21354)

Items (see page 21348)

154.1.13.5.12 TCustomListBox.DoGetDataObject Method

Returns the object associated with an item in a virtual list box.

Pascal

```
function DoGetDataObject(const Index: Integer): TObject;
```

C++

```
__fastcall TObject * DoGetDataObject(const int Index);
```

Description

This protected method is called automatically when a virtual list box needs to supply the object associated with an item. DoGetDataObject is never called unless the list box is virtual (that is, unless the value of the Style (see page 21354) property is lbVirtual or lbVirtualOwnerDraw).

DoGetDataObject generates an OnDataObject (see page 21358) event and returns the object that is supplied dynamically by the event handler.

See Also

OnDataObject (see page 21358)

Style (see page 21354)

Items (see page 21348)

154.1.13.5.13 TCustomListBox.DragCanceled Method

Sends a mouse message canceling the drag-and-drop operation.

Pascal

```
procedure DragCanceled; override;
```

C++

```
virtual __fastcall DragCanceled();
```

Description

DragCanceled is called in response to the cancellation of a drag-and-drop operation that had started. Users can cancel the operation by dropping the item over a component that does not accept the item, or by pressing Esc while dragging.

The OnEndDrag event occurs after the call to DragCanceled.

See Also

OnEndDrag (see page 5860)

154.1.13.5.14 TCustomListBox.DrawItem Method

Generates an OnDrawItem (see page 21358) event.

Pascal

```
procedure DrawItem(Index: Integer; Rect: TRect; State: TOwnerDrawState); virtual;
```

C++

```
virtual __fastcall DrawItem(int Index, TRect Rect, TOwnerDrawState State);
```

Description

DrawItem is called whenever a visual aspect of an owner-draw list box changes. The list box calls DrawItem for each visible item in its list, passing the index of the item in the Index parameter.

By default, the DrawItem method for a list box calls any event handler attached to the component's OnDrawItem (see page 21358) event. If there is no handler attached, DrawItem fills the rectangle passed in Rect and draws any text associated with the indexed item. The default drawing ignores the State parameter, although State is passed along to attached event handlers.

See Also

ItemHeight (see page 21347)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

Style (see page 21354)

154.1.13.5.15 TCustomListBox.GetCount Method

Returns the value of the Count (see page 21346) property.

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCount is the protected read implementation of the Count (see page 21346) property.

See Also

Count (see page 21346)

154.1.13.5.16 TCustomListBox.GetItemData Method

Returns a 32-bit value associated with the specified item in a list box.

Pascal

```
function GetItemData(Index: Integer): LongInt; dynamic;
```

C++

```
__fastcall LongInt GetItemData(int Index);
```

Description

GetItemData retrieves the data associated with a particular item in the list box. This value can be displayed as that list box item, or can be a value (such as a pointer) associated with the list box item.

See Also

SetItemData (see page 21336)

154.1.13.5.17 TCustomListBox.GetItemIndex Method

Returns the value of the ItemIndex property.

Pascal

```
function GetItemIndex: Integer; override;
```

C++

```
virtual __fastcall int GetItemIndex();
```

Description

GetItemIndex is the protected read implementation of the ItemIndex property. It returns the index of the first selected item in the list box.

The first item in the list has index 0, the second item has index 1, and so on. If no item is selected, the value of ItemIndex is -1.

See Also

ItemIndex

154.1.13.5.18 TCustomListBox.GetSelCount Method

Returns the number of selected items.

Pascal

```
function GetSelCount: Integer; override;
```

C++

```
virtual __fastcall int GetSelCount();
```

Description

GetSelCount is the protected read implementation of the SelCount property.

See Also

SelCount

154.1.13.5.19 TCustomListBox.GetSelected Method

Indicates whether a particular item is selected.

Pascal

```
function GetSelected(Index: Integer): Boolean;
```

C++

```
__fastcall Boolean GetSelected(int Index);
```

Description

GetSelected is the protected read implementation of the Selected (see page 21351) property. It returns true if the item specified by Index is selected, false if it is not.

154.1.13.5.20 TCustomListBox.InternalGetItemData Method

Retrieves the data associated with an item in the list box.

Pascal

```
function InternalGetItemData(Index: Integer): Longint; dynamic;
```

C++

```
__fastcall Longint InternalGetItemData(int Index);
```

Description

InternalGetItemData is the low-level protected method for fetching the data associated with an item in the list box. It retrieves a reference pointer to the object associated with the specified item, as an integer.

Index is the index of the item whose data is to be fetched.

See Also

GetItemData (see page 21329)

154.1.13.5.21 TCustomListBox.InternalSetItemData Method

Associates an integer with a specified item in the list box.

Pascal

```
procedure InternalSetItemData(Index: Integer; AData: Longint); dynamic;
```

C++

```
__fastcall InternalSetItemData(int Index, Longint AData);
```

Description

InternalSetItemData is the low-level protected method for associating data with an item in the list box. It associates the item that has the index specified by Index with the integer specified by AData.

See Also

SetItemData (see page 21336)

154.1.13.5.22 TCustomListBox.ItemAtPos Method

Returns the index of the list box item indicated by the coordinates of a point on the control.

Pascal

```
function ItemAtPos(Pos: TPoint; Existing: Boolean): Integer;
```

C++

```
__fastcall int ItemAtPos(TPoint Pos, Boolean Existing);
```

Description

Use ItemAtPos to detect if an item exists at a particular point in the control.

The Pos parameter is the point in the control in window coordinates. If Pos is beyond the last item in the list box, the value of the Existing variable determines the returned value. If Existing is set to true, ItemAtPos returns -1, indicating that no item exists at that point. If Existing is set to false, ItemAtPos returns the index of the last item in the list box plus one.

C++ Examples:

```
/*
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < ComponentCount; i++)
    ListBox1->Items->InsertObject(0,
                                  Components[i]->Name,
                                  (TObject *)Components[i]);
}

void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  if (Button == mbRight)
  {
```

```
    TClass ClassRef;
    int Index = ListBox1->ItemAtPos(Point(X,Y), true);
    // only components that are controls have a position
    // make sure the component is a control
    for (ClassRef = ListBox1->Items->Objects[Index]->ClassType();
         ClassRef != NULL;
         ClassRef = ClassRef->ClassParent())
      if (String(ClassRef->ClassName()) == "TControl")
      {
        TControl *TheObject = (TControl *)ListBox1->Items->Objects[Index];
        StatusBar1->SimpleText =
          TheObject->Name + " is at (" +
          IntToStr(TheObject->Left) + ", " +
          IntToStr(TheObject->Top) + ")";
         break;
      }
    if (ClassRef == NULL) // if it wasn't a control
      MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to Form1.ComponentCount-1 do
    ListBox1.Items.InsertObject(
      0, Form1.Components[I].Name,
      Form1.Components[I] as TObject);
end;

procedure TForm1.ListBox1MouseUp(
  Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X,Y: integer);
var
  APoint: TPoint;
  Index: integer;
  TheObject: TControl;
begin
  if Button = mbRight then
  begin
  APoint.X := X;
  APoint.Y := Y;
  Index := ListBox1.ItemAtPos(APoint, True);
  if (ListBox1.Items.Objects[Index] is TControl) then
  begin
    TheObject :=
      (ListBox1.Items.Objects[Index] as TControl);
    StatusBar1.SimpleText := TheObject.Name + ' is at (' +
                             IntToStr(TheObject.Left) + ', ' +
```

```
                                IntToStr(TheObject.Top) + ') ';
    end
    else
      Beep;
    end;
  end;
```

See Also

ItemIndex

Items (see page 21348)

154.1.13.5.23 TCustomListBox.ItemRect Method

Returns the rectangle that surrounds the item specified in the Item parameter.

Pascal

```
function ItemRect(Index: Integer): TRect;
```

C++

```
__fastcall TRect ItemRect(int Index);
```

Description

Use ItemRect to get the coordinates of a particular item in the list box. For example, ItemRect is used internally by the ItemAtPos (see page 21331) method.

The Item parameter is the ItemIndex of the item whose position is queried.

C++ Examples:

```
/*
This example uses a list box on a form. When the user clicks
one of the items in the list box, a beep occurs if the mouse
is in the first half of the item. A similar technique can be
used to detect "hot" regions in an owner-draw list box:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Hello");
  ListBox1->Items->Add("New");
  ListBox1->Items->Add("World");
}

void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  TRect ItemBounds;
  if (Button == mbLeft)
  {
    ItemBounds = ListBox1->ItemRect(ListBox1->ItemIndex);
    if (Y > ItemBounds.Top &&
        Y < ItemBounds.Bottom &&
        X > ItemBounds.Left &&
        X < (ItemBounds.Left + ItemBounds.Right)/2)
      MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
This example uses a list box on a form. When the user clicks
one of the items in the list box, a beep occurs if the mouse
```

```
is in the first half of the item. A similar technique can be
used to detect "hot" regions in an owner-draw list box:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  with ListBox1 do
  begin
    Items.Add('Hello');
    Items.Add('New');
    Items.Add('World');
  end;
end;

procedure TForm1.ListBox1MouseUp (Sender: TObject; Button: TMouseButton; Shift: TShiftState;
X, Y: Integer);
var
  ListBoxItem: TRect;
begin
  if Button = mbLeft then
  begin
    ListBoxItem := ListBox1.ItemRect(ListBox1.ItemIndex);
    if (Y > ListBoxItem.Top) and
       (Y < ListBoxItem.Bottom) and
       (X > ListBoxItem.Left) and
       (X < (ListBoxItem.Left + ListBoxItem.Right) div 2) then
      Beep;
  end;
 end;
```

See Also

ItemAtPos (see page 21331)

ItemIndex

154.1.13.5.24 TCustomListBox.KeyPress Method

Responds when the user presses a key.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress is called automatically when the list box receives a key-press message (WM_CHAR). It generates an OnKeyPress event, and then, if the AutoComplete (see page 21342) property is true, updates ItemIndex to reflect the next character that the user typed.

The Key parameter is the character represented by the key that is pressed, not a Windows virtual key code.

See Also

OnKeyPress (see page 6227)

154.1.13.5.25 TCustomListBox.LoadRecreateItems Method

Pascal

```
procedure LoadRecreateItems(RecreateItems: TStrings); virtual;
```

C++

```
virtual __fastcall LoadRecreateItems(TStrings RecreateItems);
```

Description

This is LoadRecreateItems, a member of class TCustomListBox.

154.1.13.5.26 TCustomListBox.MeasureItem Method

Generates an OnMeasureItem (see page 21360) event.

Pascal

```
procedure MeasureItem(Index: Integer; var Height: Integer); virtual;
```

C++

```
virtual __fastcall MeasureItem(int Index, int Height);
```

Description

MeasureItem is called whenever a visual aspect of a variable owner-draw list box changes and the value of the Style (see page 21354) property is csOwnerDrawVariable.

Before drawing its items, a variable owner-draw list box calls MeasureItem once for each visible item, passing the index of the item to measure, and the default height of the item. Since Height is passed by reference (a var parameter), MeasureItem can increase or reduce the height of each item as needed.

By default, the MeasureItem method does nothing except call any event handler attached to the OnMeasureItem (see page 21360) event. Override MeasureItem to change or add functionality to the default behavior.

See Also

DrawItem (see page 21328)

ItemHeight (see page 21347)

ItemIndex

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

Style (see page 21354)

154.1.13.5.27 TCustomListBox.ResetContent Method

Removes all of the items from a list box.

Pascal

```
procedure ResetContent; dynamic;
```

C++

```
__fastcall ResetContent();
```

Description

ResetContent is used to remove the items from the list box. ResetContent is used, for example, by the Clear (see page 21322) method.

See Also

Clear (see page 21322)

154.1.13.5.28 TCustomListBox.SaveRecreateItems Method

Pascal

```
procedure SaveRecreateItems(RecreateItems: TStrings); virtual;
```

C++

```
virtual __fastcall SaveRecreateItems(TStrings RecreateItems);
```

Description

This is SaveRecreateItems, a member of class TCustomListBox.

154.1.13.5.29 TCustomListBox.SelectAll Method

Selects the entire block of text in the list box.

Pascal

```
procedure SelectAll; override;
```

C++

```
virtual __fastcall SelectAll();
```

Description

Use SelectAll to select the entire block of text in the list box. To select only part of the text, use the SelStart and SelLength properties.

154.1.13.5.30 TCustomListBox.SetItemData Method

Sets the 32-bit value associated with the specified item in a list box.

Pascal

```
procedure SetItemData(Index: Integer; AData: LongInt); dynamic;
```

C++

```
__fastcall SetItemData(int Index, LongInt AData);
```

Description

Use SetItemData to set the data associated with a particular item in a list box. This value can be displayed as that list box item, or it can be a value (such as a pointer) represented by the list box item.

See Also

GetItemData (see page 21329)

154.1.13.5.31 TCustomListBox.SetItemIndex Method

Gives focus to the item with a specified index.

Pascal

```
procedure SetItemIndex(const Value: Integer); override;
```

C++

```
virtual __fastcall SetItemIndex(const int Value);
```

Description

SetItemIndex is the protected write implementation for the ItemIndex property. Value is the index of the item that should receive focus.

See Also

ItemIndex

154.1.13.5.32 TCustomListBox.SetMultiSelect Method

Sets the MultiSelect property.

Pascal

```
procedure SetMultiSelect(Value: Boolean); override;
```

C++

```
virtual __fastcall SetMultiSelect(Boolean Value);
```

Description

SetMultiSelect is the protected write implementation for the MultiSelect property. Value is true if the list box should allow more than one item to be selected, false if only one item can be selected at a time.

See Also

MultiSelect

154.1.13.5.33 TCustomListBox.WndProc Method

Handles Windows messages.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc overrides the inherited WndProc to provide specialized responses to messages that allows the list box to handle auto drag mode itself. Then it calls the inherited WndProc to handle all other messages.

When overriding the WndProc method, be sure to call the inherited WndProc at the end to dispatch any other messages.

C++ Examples:

```
/*
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
*/
class TMyListBoxDescendant : public TListBox
{
__published:    // IDE-managed Components
    void __fastcall SubClassWndProc(Messages::TMessage &Message);
    void __fastcall ToggleSubClass(bool On);
    void __fastcall OnDrawItemProc(
      TWinControl *Control, int Index, const TRect &Rect,
```

```
      TOwnerDrawState State);
private:    // User declarations
public:     // User declarations
    __fastcall TMyListBoxDescendant(TComponent* Owner);
};

TForm1 *Form1;
TMyListBoxDescendant *MyListBoxDescendant1;
Graphics::TBitmap *bitmap0;

const WM_STYLEMESSAGE = WM_USER + 2000;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

__fastcall TMyListBoxDescendant::TMyListBoxDescendant(TComponent* Owner)
    : TListBox(Owner)
{
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
}

void __fastcall TForm1::SubClassRadioGroup1Click(TObject *Sender)
{
  MyListBoxDescendant1->ToggleSubClass(SubClassRadioGroup1->ItemIndex == 0);
}

void __fastcall TMyListBoxDescendant::SubClassWndProc(Messages::TMessage &Message)
{
  if (Message.Msg == WM_STYLEMESSAGE)
    Style = (TListBoxStyle)Message.WParam;
  else
    WndProc(Message);
}

void __fastcall TMyListBoxDescendant::ToggleSubClass(bool On)
{
  if (On)
    WindowProc = SubClassWndProc;
  else
    WindowProc = WndProc;
}

void __fastcall TMyListBoxDescendant::OnDrawItemProc(TWinControl *Control, int Index,
      const TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *bitmap; // temporary variable for item's bitmap
  int Offset = 2;   // default text offset width
```

```
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *canvas = ((TListBox *)Control)->Canvas;
  canvas->FillRect(Rect); // clear the rectangle
  bitmap = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
  if (bitmap)
  {
    canvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, bitmap->Width, bitmap->Height),
      bitmap, Bounds(0, 0, bitmap->Width, bitmap->Height), clRed); // render bitmap
    Offset += bitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  canvas->TextOut(Rect.Left + Offset, Rect.Top, ((TListBox *)Control)->Items->Strings[Index]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyListBoxDescendant1 = new TMyListBoxDescendant(Form1);
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Parent = Form1;
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Left =
    SubClassRadioGroup1->Left + SubClassRadioGroup1->Width + 30;;
  MyListBoxDescendant1->Top = SubClassRadioGroup1->Top;
  MyListBoxDescendant1->Height = SubClassRadioGroup1->Height;
  MyListBoxDescendant1->OnDrawItem =
    MyListBoxDescendant1->OnDrawItemProc;

  bitmap0 = new Graphics::TBitmap;
  ImageList1->GetBitmap(0, bitmap0);
  MyListBoxDescendant1->Items->AddObject("Butterfly", bitmap0);

  SubClassRadioGroup1->Items->Add("SubClassWndProc");
  SubClassRadioGroup1->Items->Add("WndProc");
  SubClassRadioGroup1->ItemIndex = 2;
}
```

Delphi Examples:

```
{
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
}

type
  TMyListBoxDescendant = class(TlistBox)
    procedure SubClassWndProc(var Message: TMessage);
    procedure ToggleSubClass(On: Boolean);
    procedure OnDrawItemProc(
      Control: TWinControl;
      Index: Integer;
      Rect:TRect;
      State: TOwnerDrawState);
  end;
  TForm1 = class(TForm)
    SubClassRadioGroup1: TRadioGroup;
    Button1: TButton;
    ImageList1: TImageList;
    Button2: TButton;
    procedure SubClassRadioGroup1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
```

```
      procedure Button2Click(Sender: TObject);
    private
      { Private declarations }
    public
      { Public declarations }
    end;

  var
    Form1: TForm1;
    MyListBoxDescendant1: TMyListBoxDescendant;
    bitmap0: TBitmap;

  implementation

  {$R *.dfm}

  const WM_STYLEMESSAGE = WM_USER + 2000;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    PostMessage(
      MyListBoxDescendant1.Handle,
      WM_STYLEMESSAGE,
      Integer(lbOwnerDrawFixed),
      0);
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    PostMessage(
      MyListBoxDescendant1.Handle,
      WM_STYLEMESSAGE,
      Integer(lbStandard),
      0);
  end;

  procedure TMyListBoxDescendant.SubClassWndProc(var Message: TMessage);
  begin
    if (Message.Msg = WM_STYLEMESSAGE) then
      Style:= TListBoxStyle(Message.WParam)
    else
      WndProc(Message);
  end;

  procedure TMyListBoxDescendant.ToggleSubClass(On: Boolean);
  begin
    if On then
      WindowProc := SubClassWndProc
    else
      WindowProc := WndProc;
  end;

  procedure TForm1.SubClassRadioGroup1Click(Sender: TObject);
  begin
    MyListBoxDescendant1.ToggleSubClass(
      SubClassRadioGroup1.ItemIndex = 0);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    MyListBoxDescendant1:= TMyListBoxDescendant.Create(self);
    MyListBoxDescendant1.Visible:= True;
    MyListBoxDescendant1.Parent:= Form1;
    MyListBoxDescendant1.Visible:= True;
    MyListBoxDescendant1.Left:=
      SubClassRadioGroup1.Left + SubClassRadioGroup1.Width + 30;;
```

```
    MyListBoxDescendant1.Top:= SubClassRadioGroup1.Top;
    MyListBoxDescendant1.Height:= SubClassRadioGroup1.Height;
    MyListBoxDescendant1.OnDrawItem:=
      MyListBoxDescendant1.OnDrawItemProc;

    bitmap0 := TBitmap.Create;
    ImageList1.GetBitmap(0, bitmap0);
    MyListBoxDescendant1.Items.AddObject('Butterfly', bitmap0);

    SubClassRadioGroup1.Items.Add('SubClassWndProc');
    SubClassRadioGroup1.Items.Add('WndProc');
    SubClassRadioGroup1.ItemIndex := 2;
  end;

  procedure TMyListBoxDescendant.OnDrawItemProc(
    Control: TWinControl;
    Index: Integer;
    Rect:TRect;
    State: TOwnerDrawState);
  var
    Bitmap: TBitmap;      { temporary variable for the item's bitmap }
    Offset: Integer;      { text offset width }
  begin
    { draw on control canvas, not on the form }
    with (Control as TListBox).Canvas do
    begin
      FillRect(Rect);       { clear the rectangle }
      Offset := 2;          { provide default offset }
      Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
      if Bitmap <> nil then
      begin
        BrushCopy(
          Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
          Bitmap,
          Bounds(0, 0, Bitmap.Width, Bitmap.Height),
          clRed);  {render bitmap}
        Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
      end;
      TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
  text }
    end;
  end;
```

See Also

WndProc (see page 6187)

154.1.13.6 TCustomListBox Properties

The properties of the TCustomListBox class are listed here.

Protected Properties

	Name	Description
	BorderStyle (see page 21343)	Determines whether the list box has a border.
	Columns (see page 21346)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	ExtendedSelect (see page 21346)	This is ExtendedSelect, a member of class TCustomListBox.
	IntegralHeight (see page 21347)	Determines whether the list box displays the partial items.
	ItemHeight (see page 21347)	Specifies the height in pixels of an item in an owner-draw list box.
	ParentColor (see page 21350)	Determines where a control looks for its color information.
	Sorted (see page 21353)	Specifies whether the items in a list box are arranged alphabetically.

	Style (see page 21354)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabWidth (see page 21355)	Specifies the size of the tabs in the list box.

Public Properties

	Name	Description
	AutoComplete (see page 21342)	Determines whether the user can give focus to items by typing in the list.
	AutoCompleteDelay (see page 21342)	Specifies the delay between a key press and an attempt to autocomplete the field.
R	Canvas (see page 21343)	Provides a drawing surface when implementing a handler for the OnDrawItem (see page 21358) event of an owner-draw list box.
	Count (see page 21346)	Indicates the number of items in the list box.
	Items (see page 21348)	Contains the strings that appear in the list box.
	ScrollWidth (see page 21351)	Specifies the width, in pixels, by which the list box can scroll horizontally.
	Selected (see page 21351)	Indicates whether a particular item is selected.
	TopIndex (see page 21356)	Specifies the index number of the item that appears at the top of the list box.

Published Properties

	Name	Description
	TabStop (see page 21355)	Determines if the user can tab to a control.

154.1.13.6.1 TCustomListBox.AutoComplete Property

Determines whether the user can give focus to items by typing in the list.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

Use AutoComplete to specify whether the list box automatically updates the ItemIndex property in response to user keystrokes. When AutoComplete is true, the item with focus changes to the first item that begins with the sequence of characters that the user typed. As the user types more characters, ItemIndex may change to a later item because the string to match becomes longer.

See Also

ItemIndex

154.1.13.6.2 TCustomListBox.AutoCompleteDelay Property

Specifies the delay between a key press and an attempt to autocomplete the field.

Pascal

```
property AutoCompleteDelay: Cardinal;
```

C++

```
__property unsigned AutoCompleteDelay;
```

Description

Specifies the delay in milliseconds between a key press and an attempt to autocomplete the field. Autocomplete will position to matching list values as you type, sometimes saving you keystrokes.

AutoCompleteDelay is only in effect when the AutoComplete (see page 21342) is true.

See Also

AutoComplete (see page 21342)

154.1.13.6.3 TCustomListBox.BorderStyle Property

Determines whether the list box has a border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to change the appearance of the outer boundary of the list box control. A list box can have a single border (bsSingle) or no visible border (bsNone).

See Also

Ctl3D (see page 6198)

154.1.13.6.4 TCustomListBox.Canvas Property

Provides a drawing surface when implementing a handler for the OnDrawItem (see page 21358) event of an owner-draw list box.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Canvas is only useful for applications that control the drawing of list box items, that is, for owner-draw list boxes. Canvas is used to draw graphical images that can be used as items in a list box.

By default a TControlCanvas object is created for the Canvas property in the list box constructor. To create an owner-draw list box that uses a canvas, set the Style (see page 21354) property accordingly.

C++ Examples:

```
/*
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyList = new TStringList;
  bitmap0 = new Graphics::TBitmap;
  bitmap1 = new Graphics::TBitmap;
  bitmap2 = new Graphics::TBitmap;
//  MyList->Add("Animal");
```

```
//  MyList->Add("Flowers");
//  MyList->Add("Butterfly");
//  ListBox1.Items->AddStrings(MyList);

  ImageList1->GetBitmap(0, bitmap0);
  ListBox1->Items->AddObject("Flowers", bitmap0);
  ImageList1->GetBitmap(1, bitmap1);
  ListBox1->Items->AddObject("Animal", bitmap1);
  ImageList1->GetBitmap(2, bitmap2);
  ListBox1->Items->AddObject("Butterfly", bitmap2);
}

void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
      TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBitmap; // temporary variable for item's bitmap
  int     Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *pCanvas = ((TListBox *)Control)->Canvas;
  pCanvas->FillRect(Rect); // clear the rectangle
  pBitmap = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
  if (pBitmap)
  {
    pCanvas->BrushCopy(Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
pBitmap, Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
    Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  pCanvas->TextOut(Rect.Left + Offset, Rect.Top, ((TListBox *)Control)->Items->Strings[Index]);
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TListBox *myListBox;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TListBox")
  {
    myListBox = (TListBox *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myListBox->Canvas->MoveTo(myListBox->ClientRect.Left, myListBox->ClientRect.Top);
    myListBox->Canvas->LineTo(myListBox->ClientRect.Right, myListBox->ClientRect.Bottom);
    myListBox = NULL;
  }
}
```

Delphi Examples:

```
{
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  MyList := TStringList.Create;
```

```
    bitmap0 := TBitmap.Create;
    bitmap1 := TBitmap.Create;
    bitmap2 := TBitmap.Create;
  //  MyList.Add('Animal');
  //  MyList.Add('Flowers');
  //  MyList.Add('Butterfly');
  //  ListBox1.Items.AddStrings(MyList);

    ImageList1.GetBitmap(0, bitmap0);
    ListBox1.Items.AddObject('Flowers', bitmap0);
    ImageList1.GetBitmap(1, bitmap1);
    ListBox1.Items.AddObject('Animal', bitmap1);
    ImageList1.GetBitmap(2, bitmap2);
    ListBox1.Items.AddObject('Butterfly', bitmap2);
  end;

  procedure TForm1.ListBox1DrawItem(Control: TWinControl; Index: Integer; Rect:TRect; State:
  TOwnerDrawState);
  var
    Bitmap: TBitmap;      { temporary variable for the item’s bitmap }
    Offset: Integer;      { text offset width }
  begin
    Bitmap := TBitmap.Create;
    with (Control as TListBox).Canvas do  { draw on control canvas, not on the form }
    begin
      FillRect(Rect);       { clear the rectangle }
      Offset := 2;          { provide default offset }
      Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
      if Bitmap <> nil then
      begin
        BrushCopy(
          Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
          Bitmap,
          Bounds(0, 0, Bitmap.Width, Bitmap.Height),
          clRed);  {render bitmap}
        Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
      end;
      TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
  text }
    end;
  end;
  {
  The following code uses ClientRect to find and draw a line
  from the top left to the bottom right of the current
  control.  Select a TControl in the ListBox list.  This
  example assumes that the current object is a descendent of
  TControl and has a public Canvas property.
  }
    if (ListBox1.Items.Objects[Index] is TListBox) then
    begin
      myListBox:= (ListBox1.Items.Objects[Index] as TListBox);
      with myListBox.ClientRect do
      begin // make sure we are using the TControl's canvas and not the form's!
        myListBox.Canvas.MoveTo(Left, Top);
        myListBox.Canvas.LineTo(Right, Bottom);
      end;
    myListBox:= nil;
    end;
```

See Also

ItemHeight (see page 21347)

OnDrawItem (see page 21358)

Style (see page 21354)

TCanvas (see page 14441)

TControlCanvas (see page 5903)

154.1.13.6.5 TCustomListBox.Columns Property

Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.

Pascal

```
property Columns: Integer;
```

C++

```
__property int Columns;
```

Description

Use Columns to specify the number of columns, in a multi-column list box, that are visible without having to use the horizontal scroll bar.

Multi-column list boxes have a horizontal scroll bar that allows users to view multiple columns as they wrap. The default value for Columns is 0, meaning that the list box is not multi-column. That is, users can scroll only vertically and the list of items will not wrap.

For Columns values greater than 0, multiple columns accommodate the items as they wrap beyond the bottom of the list box. The Columns property specifies the number of columns that are visible without having to horizontally scroll the list box.

The width of each column depends upon both the Width property and the number of Columns.

See Also

ItemHeight (see page 21347)

Width (see page 5849)

154.1.13.6.6 TCustomListBox.Count Property

Indicates the number of items in the list box.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use Count to determine the number of items in the list box. You can only set Count when the Style (see page 21354) property is lbVirtual or lbVirtualOwnerDraw, in which case you must set this property.

See Also

Style (see page 21354)

154.1.13.6.7 TCustomListBox.ExtendedSelect Property

Pascal

```
property ExtendedSelect: Boolean;
```

C++

```
__property Boolean ExtendedSelect;
```

Description

This is ExtendedSelect, a member of class TCustomListBox.

154.1.13.6.8 TCustomListBox.IntegralHeight Property

Determines whether the list box displays the partial items.

Pascal

```
property IntegralHeight: Boolean;
```

C++

```
__property Boolean IntegralHeight;
```

Description

IntegralHeight controls the height of the list box itself.

When IntegralHeight is true, and the Style (see page 21354) property is lsOwnerDrawFixed, the list box height will always be a multiple of the ItemHeight (see page 21347) property value. In this case, items will never be partially displayed.

If IntegralHeight is false, the list box height can be set to any value, but the last item may not be fully displayed.

If the Style (see page 21354) property value of the list box is lbOwnerDrawVariable, setting the IntegralHeight property to true has no effect.

See Also

ItemHeight (see page 21347)

Items (see page 21348)

Style (see page 21354)

Using TDBListBox and TDBComboBox

154.1.13.6.9 TCustomListBox.ItemHeight Property

Specifies the height in pixels of an item in an owner-draw list box.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Use ItemHeight to control the height of an item in a list box when the Style (see page 21354) property is lbOwnerDrawFixed. The ItemHeight property can also affect the height of the list box itself. When the IntegralHeight (see page 21347) property is true, the height of the list box is always an increment of the ItemHeight value.

If the Style (see page 21354) property is lbStandard or lbOwnerDrawVariable, the value of ItemHeight is ignored.

See Also

DrawItem (see page 21328)

IntegralHeight (see page 21347)

Items (see page 21348)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

Style (see page 21354)

Using TDBListBox and TDBComboBox

154.1.13.6.10 TCustomListBox.Items Property

Contains the strings that appear in the list box.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to add, insert, delete and move items. By default, the items in a list box are of type TStrings. Use this item type to access its methods or properties to manipulate the items in the list.

For example, the following code adds the text in the edit box to the list box as an item:

```
ListBox1.Items.Add(Edit1.Text); {Delphi }
ListBox1->Items->Add(Edit1->Text); // C++
```

C++ Examples:

```
/*
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Continue will continue listing
file sizes in a multiple selection that includes the deleted
file.  Break will stop.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
*/
#include <stdio.h>     // for FILE, fopen, fstat, fileno, fread and fclose
#include <sys\stat.h>  // for fstat and the stat struct

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < FileListBox1->Items->Count; i++)
  {
    if (FileListBox1->Selected[i])
    {
      if (!FileExists(FileListBox1->Items->Strings[i]))
      {
        MessageBeep(0);
        if (Application->MessageBox(
          AnsiString("File " + FileListBox1->Items->Strings[i] + "not found").c_str(),
          NULL,
          MB_OKCANCEL) == IDCANCEL)
          break;
        else
          continue;
```

```
        }
        FILE *F = fopen(FileListBox1->Items->Strings[i].c_str(),"r");
        struct stat statbuf;
        fstat(fileno(F), &statbuf);
        ListBox1->Items->Add(
            FileListBox1->Items->Strings[i] + ": " + IntToStr(statbuf.st_size));
        fclose(F);
      }
    }
  }
/*
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
list of strings in the list box. When the application runs
and the button is clicked, the strings in the list box
become uppercase.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int I = 0; I < ListBox1->Items->Count; I++)
    ListBox1->Items->Strings[I] = UpperCase(ListBox1->Items->Strings[I]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");
}
```

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], 1);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode 1.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);
```

```
        AssignFile(F, FileListBox1.Items.Strings[i]);
        Reset(F, 1);
        ListBox1.Items.Add(
          FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
        CloseFile(F);
      end;
    finally
    { do something here }
    end;
  end;
end;
{
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
list of strings in the list box. When the application runs
and the button is clicked, the strings in the list box
become uppercase.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to ListBox1.Items.Count -1 do
    ListBox1.Items[I] := SysUtils.UpperCase(ListBox1.Items[I]);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  Listbox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;
```

See Also

Clear (see page 21322)

ItemIndex

OnDrawItem (see page 21358)

Selected (see page 21351)

TStrings (see page 3029)

154.1.13.6.11 TCustomListBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the

form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

154.1.13.6.12 TCustomListBox.ScrollWidth Property

Specifies the width, in pixels, by which the list box can scroll horizontally.

Pascal

```
property ScrollWidth: Integer;
```

C++

```
__property int ScrollWidth;
```

Description

Use ScrollWidth to get or set the logical width of the list box. When ScrollWidth is greater than the client width of the list box, the list box gets a horizontal scroll bar. When ScrollWidth is less than or equal to ClientWidth, the horizontal scroll bar disappears.

See Also

Columns (see page 21346)

ClientWidth (see page 5811)

154.1.13.6.13 TCustomListBox.Selected Property

Indicates whether a particular item is selected.

Pascal

```
property Selected [Index: Integer]: Boolean;
```

C++

```
__property Boolean Selected[int Index];
```

Description

Use Selected to query the state of a particular item in the list box. If the item specified by the Index parameter is selected in the list box, the value of the Selected property is true. If the specified item is not selected, Selected is false. Set Selected to change the currently selected item.

The Index parameter is the item referenced by its position in the list box, with the first item having an Index value of 0.

An item is selected when the user highlights it. More than one item in the list box can be selected by setting the MultiSelect property to true. In this case, the Selected item is the one that has focus when the list box has input focus.

C++ Examples:

```
/*
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
```

```
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Continue will continue listing
file sizes in a multiple selection that includes the deleted
file.  Break will stop.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
*/
#include <stdio.h>     // for FILE, fopen, fstat, fileno, fread and fclose
#include <sys\stat.h>  // for fstat and the stat struct

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < FileListBox1->Items->Count; i++)
  {
    if (FileListBox1->Selected[i])
    {
      if (!FileExists(FileListBox1->Items->Strings[i]))
      {
        MessageBeep(0);
        if (Application->MessageBox(
          AnsiString("File " + FileListBox1->Items->Strings[i] + "not found").c_str(),
          NULL,
          MB_OKCANCEL) == IDCANCEL)
          break;
        else
          continue;
      }
      FILE *F = fopen(FileListBox1->Items->Strings[i].c_str(),"r");
      struct stat statbuf;
      fstat(fileno(F), &statbuf);
      ListBox1->Items->Add(
          FileListBox1->Items->Strings[i] + ": " + IntToStr(statbuf.st_size));
      fclose(F);
    }
  }
}
```

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], 1);
      if (filehandle = -1) then
```

```
        begin
          MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                     ' cannot be opened with access mode 1.', mtError, [mbOk], 0);
          Continue;
        end
        else
          FileClose(filehandle);

        AssignFile(F, FileListBox1.Items.Strings[i]);
        Reset(F, 1);
        ListBox1.Items.Add(
          FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
        CloseFile(F);
      end;
    finally
    { do something here }
    end;
  end;
end;
```

See Also

extendedselect (see page 21346)

Items (see page 21348)

MultiSelect

SelCount

154.1.13.6.14 TCustomListBox.Sorted Property

Specifies whether the items in a list box are arranged alphabetically.

Pascal

```
property Sorted: Boolean;
```

C++

```
__property Boolean Sorted;
```

Description

Use Sorted to sort the items by setting its value to true. If Sorted is false, the items are unsorted. When Sorted is true, items are automatically added or inserted into the list box in alphabetical order.

Note: Sorted has no effect if Listbox's style is lbVirtual or lbVirtualOwnerDraw

C++ Examples:

```
/*
This example uses an edit box, a list box, and two buttons
on a form. The buttons are named Add and Sort. When the user
clicks the Add button, the text in the edit box is added to
the list in the list box. Before the user clicks the sort
button, any new items are added to the end of the list.
After the user clicks the Sort button, the list in the list
box is sorted and remains sorted, even if additional strings
are added.  If the user clicks the sort button again, the
list box remains sorted, but any new strings are again added
to the end of the list.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");
}

void __fastcall TForm1::AddClick(TObject *Sender)
{
  ListBox1->Items->Add(Edit1->Text);
}

void __fastcall TForm1::SortClick(TObject *Sender)
{
  ListBox1->Sorted = True;
}
```

Delphi Examples:

```
{
This example uses an edit box, a list box, and two buttons
on a form. The buttons are named Add and Sort. When the user
clicks the Add button, the text in the edit box is added to
the list in the list box. Before the user clicks the sort
button, any new items are added to the end of the list.
After the user clicks the Sort button, the list in the list
box is sorted and remains sorted, even if additional strings
are added.  If the user clicks the sort button again, the
list box remains sorted, but any new strings are again added
to the end of the list.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;


procedure TForm1.AddClick(Sender: TObject);
begin
  ListBox1.Items.Add(Edit1.Text);
end;

procedure TForm1.SortClick(Sender: TObject);
begin
  ListBox1.Sorted := True;
end;
```

See Also

Items (see page 21348)

Style (see page 21354)

154.1.13.6.15 TCustomListBox.Style Property

Determines whether the list box is standard or owner-draw and whether it is virtual.

Pascal

```
property Style: TListBoxStyle;
```

C++

```
__property TListBoxStyle Style;
```

Description

Use Style to specify whether the list box is a standard list box that displays text strings, or an owner-draw list box that can display graphical images. Owner-draw list boxes let you display items in some nonstandard way. In this case, you must write the code to paint items in the list box.

In addition, Style controls whether the list box is virtual, which means that items are supplied dynamically using event handlers. By default, Style is lbStandard, meaning that the list box is not virtual and it displays each item as a string.

See Also

DrawItem (see page 21328)

ItemHeight (see page 21347)

Items (see page 21348)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

OnData (see page 21357)

OnDataFind (see page 21357)

OnDataObject (see page 21358)

154.1.13.6.16 TCustomListBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

154.1.13.6.17 TCustomListBox.TabWidth Property

Specifies the size of the tabs in the list box.

Pascal

```
property TabWidth: Integer;
```

C++

```
__property int TabWidth;
```

Description

Set TabWidth to control the number of dialog base units displayed for each tab character that appears in the strings of a list box.

If TabWidth is set to 0, which is the default setting, the tabs default to the width specified by Windows, which is two dialog base units.

154.1.13.6.18 TCustomListBox.TopIndex Property

Specifies the index number of the item that appears at the top of the list box.

Pascal

```
property TopIndex: Integer;
```

C++

```
__property int TopIndex;
```

Description

Use TopIndex property to find or set the first item displayed at the top of the list box. TopIndex can be used, for example, to change the topmost item to a different item in the list.

C++ Examples:

```
/*
The following example uses a list box containing a list of
strings, a button, and an edit box on a form. When the user
runs the application and clicks the button, the third item
in the list becomes the first item, and the index value of
that item appears in the edit box. The index value displayed
is 2, indicating the third item in the list (the first item
in the list has an index value of 0):
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ListBox1->TopIndex = 2;
  Edit1->Text = IntToStr(ListBox1->TopIndex);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int Number = 1; Number <= 20; Number++)
    ListBox1->Items->Add("Item " + IntToStr(Number));
}
```

Delphi Examples:

```
{
The following example uses a list box containing a list of
strings, a button, and an edit box on a form. When the user
runs the application and clicks the button, the third item
in the list becomes the first item, and the index value of
that item appears in the edit box. The index value displayed
is 2, indicating the third item in the list (the first item
in the list has an index value of 0):
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.TopIndex := 2;
  Edit1.Text := IntToStr(ListBox1.TopIndex);
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  Number: Integer;
begin
  for Number := 1 to 20 do
    ListBox1.Items.Add('Item ' + IntToStr(Number));
end;
```

See Also

ItemIndex

Items (see page 21348)

Sorted (see page 21353)

154.1.13.7 TCustomListBox Events

The events of the TCustomListBox class are listed here.

Protected Events

	Name	Description
	OnData (see page 21357)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 21357)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 21358)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDrawItem (see page 21358)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnMeasureItem (see page 21360)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.

154.1.13.7.1 TCustomListBox.OnData Event

Occurs when a virtual list box needs to supply the text of an item.

Pascal

```
property OnData: TLBGetDataEvent;
```

C++

```
__property TLBGetDataEvent OnData;
```

Description

You must supply an OnData event handler to dynamically supply the items in the list box when the Style (see page 21354) property is lbVirtual or lbVirtualOwnerDraw.

See Also

OnDataFind (see page 21357)

OnDataObject (see page 21358)

Style (see page 21354)

Count (see page 21346)

154.1.13.7.2 TCustomListBox.OnDataFind Event

Occurs when a virtual list box needs to identify the index of an item given its text.

Pascal

```
property OnDataFind: TLBFindDataEvent;
```

C++

```
__property TLBFindDataEvent OnDataFind;
```

Description

You must supply an OnDataFind event handler to dynamically provide the mapping from list item to index when the Style (see page 21354) property is lbVirtual or lbVirtualOwnerDraw.

See Also

OnData (see page 21357)

OnDataObject (see page 21358)

Style (see page 21354)

Count (see page 21346)

154.1.13.7.3 TCustomListBox.OnDataObject Event

Occurs when a virtual list box needs to supply the object associated with an item.

Pascal

```
property OnDataObject: TLBGetDataObjectEvent;
```

C++

```
__property TLBGetDataObjectEvent OnDataObject;
```

Description

Supply an OnDataObject event handler to dynamically provide objects associated with the items in a virtual list box.

See Also

OnData (see page 21357)

OnDataFind (see page 21357)

Style (see page 21354)

Count (see page 21346)

154.1.13.7.4 TCustomListBox.OnDrawItem Event

Occurs when an item in an owner-draw list box needs to be redisplayed.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

Use OnDrawItem to write a handler for drawing of the items in list boxes with the Style (see page 21354) values lbOwnerDrawFixed, lbOwnerDrawVariable, or lbVirtualOwnerDraw. OnDrawItem occurs when the list box needs to display an item. OnDrawItem occurs only for owner-draw list boxes.

The size of the rectangle that contains the item is determined either by the ItemHeight (see page 21347) property for fixed

owner-draw list boxes or by the response to the OnMeasureItem (see page 21360) event for variable owner-draw list boxes.

C++ Examples:

```
/*
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyList = new TStringList;
  bitmap0 = new Graphics::TBitmap;
  bitmap1 = new Graphics::TBitmap;
  bitmap2 = new Graphics::TBitmap;
//  MyList->Add("Animal");
//  MyList->Add("Flowers");
//  MyList->Add("Butterfly");
//  ListBox1.Items->AddStrings(MyList);

  ImageList1->GetBitmap(0, bitmap0);
  ListBox1->Items->AddObject("Flowers", bitmap0);
  ImageList1->GetBitmap(1, bitmap1);
  ListBox1->Items->AddObject("Animal", bitmap1);
  ImageList1->GetBitmap(2, bitmap2);
  ListBox1->Items->AddObject("Butterfly", bitmap2);
}

void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
      TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBitmap; // temporary variable for item's bitmap
  int     Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *pCanvas = ((TListBox *)Control)->Canvas;
  pCanvas->FillRect(Rect); // clear the rectangle
  pBitmap = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
  if (pBitmap)
  {
    pCanvas->BrushCopy(Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
pBitmap, Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
    Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  pCanvas->TextOut(Rect.Left + Offset, Rect.Top, ((TListBox *)Control)->Items->Strings[Index]);
}
```

Delphi Examples:

```
{
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  MyList := TStringList.Create;
```

```
    bitmap0 := TBitmap.Create;
    bitmap1 := TBitmap.Create;
    bitmap2 := TBitmap.Create;
  //  MyList.Add('Animal');
  //  MyList.Add('Flowers');
  //  MyList.Add('Butterfly');
  //  ListBox1.Items.AddStrings(MyList);

    ImageList1.GetBitmap(0, bitmap0);
    ListBox1.Items.AddObject('Flowers', bitmap0);
    ImageList1.GetBitmap(1, bitmap1);
    ListBox1.Items.AddObject('Animal', bitmap1);
    ImageList1.GetBitmap(2, bitmap2);
    ListBox1.Items.AddObject('Butterfly', bitmap2);
  end;

  procedure TForm1.ListBox1DrawItem(Control: TWinControl; Index: Integer; Rect:TRect; State:
  TOwnerDrawState);
  var
    Bitmap: TBitmap;      { temporary variable for the item's bitmap }
    Offset: Integer;      { text offset width }
  begin
    Bitmap := TBitmap.Create;
    with (Control as TListBox).Canvas do  { draw on control canvas, not on the form }
    begin
      FillRect(Rect);       { clear the rectangle }
      Offset := 2;          { provide default offset }
      Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
      if Bitmap <> nil then
      begin
        BrushCopy(
          Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
          Bitmap,
          Bounds(0, 0, Bitmap.Width, Bitmap.Height),
          clRed);  {render bitmap}
        Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
      end;
      TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
  text }
    end;
  end;
```

See Also

DrawItem (see page 21328)

ItemHeight (see page 21347)

MeasureItem (see page 21335)

OnMeasureItem (see page 21360)

Style (see page 21354)

154.1.13.7.5 TCustomListBox.OnMeasureItem Event

Occurs when the application needs to redisplay an item in a variable height owner-draw list box.

Pascal

```
property OnMeasureItem: TMeasureItemEvent;
```

C++

```
__property TMeasureItemEvent OnMeasureItem;
```

Description

Use OnMeasureItem to write a handler to measure items to be drawn in a list box with a Style (see page 21354) property value of lbOwnerDrawVariable.

OnMeasureItem is of type TMeasureItemEvent (see page 21955) which contains three parameters describing the item to measure:

Control, specifying the control containing the item.

Index, specifying the index of the item in the control.

Height, specifying the height of the item.

The Index parameter identifies the position of the item in the list box.

The Height parameter should specify the height in pixels that the given item will occupy in the control. The Height parameter is passed by reference (a var parameter), which initially contains the default height of the item or the height of the item text in the control's font. The handler can set Height to a value appropriate to the contents of the item, such as the height of a graphical image to be displayed within the item.

After the OnMeasureItem event occurs, the OnDrawItem (see page 21358) event occurs, rendering the item with the measured size.

C++ Examples:

```
/*
Here is a typical handler for an OnMeasureItem event. The
example assumes that a variable owner-draw list box already
has bitmaps associated with each of its strings. It sets the
height of the item to the height of the associated bitmap if
that height is greater than the default height
(ListBox1.ItemHeight).  Note that TWinControl can also be
used in place of TWidgetControl on Windows.
*/
void __fastcall TForm1::ListBox1MeasureItem(TWinControl *Control, int Index,
      int &Height)
{
  Graphics::TBitmap *bitmap = (Graphics::TBitmap *)((TListBox
*)Control)->Items->Objects[Index];
  if (bitmap && bitmap->Height > Height)
    Height = bitmap->Height;
}

void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
      TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBitmap; // temporary variable for item's bitmap
  int     Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *pCanvas = ((TListBox *)Control)->Canvas;
  pCanvas->FillRect(Rect); // clear the rectangle
  pBitmap = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
  if (pBitmap)
  {
    pCanvas->BrushCopy(Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
pBitmap, Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
    Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  pCanvas->TextOut(Rect.Left + Offset, Rect.Top, ((TListBox *)Control)->Items->Strings[Index]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  Graphics::TBitmap *bitmap0 = new Graphics::TBitmap;
  Graphics::TBitmap *bitmap1 = new Graphics::TBitmap;
  Graphics::TBitmap *bitmap2 = new Graphics::TBitmap;
  bitmap0->LoadFromFile("../littleB_64.bmp");
//  ImageList1->GetBitmap(0, bitmap0);
  ListBox1->Items->AddObject("Butterfly", bitmap0);
  ImageList1->GetBitmap(3, bitmap1);
  ListBox1->Items->AddObject("Animal", bitmap1); // no bitmap, see if it leaves this height at
64.
  ImageList1->GetBitmap(4, bitmap2);
  ListBox1->Items->AddObject("Sunflower", bitmap2); // no bitmap, see if it leaves this height
at 64.

}
```

See Also

DrawItem (see page 21328)

ItemHeight (see page 21347)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

Style (see page 21354)

154.1.14 TCustomMemo Class

TCustomMemo is the base type for memo components, which are multiline edit boxes, including TMemo (see page 21678) and TDBMemo.

Class Hierarchy

Pascal

```
TCustomMemo = class(TCustomEdit);
```

C++

```
class TCustomMemo : public TCustomEdit;
```

File

StdCtrls

Description

TCustomMemo introduces new properties and methods to expand the capabilities of TCustomEdit (see page 21232) for handling multiline edit boxes.

Do not create instances of TCustomMemo. Use TCustomMemo as a base class when declaring control objects that implement a multiline edit control. Properties and methods of TCustomMemo provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TDBMemo (see page 9149)

TMemo (see page 21678)

154.1.14.1 TCustomMemo Members

The following tables list the members exposed by TCustomMemo.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnChange (see page 21275)	Occurs when the text for the edit control may have changed.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.

	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21245)	Generates an OnChange (see page 21275) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 21375)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 21376)	Creates a window based on the window-creation parameter record.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CreateWnd (see page 21248)	Creates a Windows control corresponding to the edit control.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.

	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21251)	Destroys the Windows control that corresponds to the edit control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoSetMaxLength (see page 21251)	Sets the MaxLength (see page 21263) property.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetCaretPos (see page 21376)	Returns the value of the CaretPos (see page 21380) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.

	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelLength (see page 21252)	Returns the SelLength (see page 21265) property.
	GetSelStart (see page 21252)	Returns the SelStart (see page 21269) property.
	GetSelText (see page 21252)	Returns the SelText (see page 21272) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 21377)	Ensures that return characters are not entered into the text when WantReturns (see page 21384) is false.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 21377)	Initializes the memo control when it is first loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.

	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAlignment (see page 21377)	Sets the Alignment (see page 21379) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetAutoSize (see page 21257)	Sets the AutoSize (see page 21260) property.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetCaretPos (see page 21378)	Sets the CaretPos (see page 21380) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetLines (see page 21378)	Sets the Lines (see page 21380) property.
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetScrollBars (see page 21378)	Sets the ScrollBars (see page 21381) property.
	SetSelLength (see page 21257)	Sets the SelLength (see page 21265) property.
	SetSelStart (see page 21257)	Sets the SelStart (see page 21269) property.
	SetWordWrap (see page 21379)	Sets the WordWrap (see page 21384) property.

	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	Alignment (see page 21379)	Determines how the text is aligned within the memo control.
	AutoSelect (see page 21260)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AutoSize (see page 21260)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 21261)	Determines whether the edit control has a single line border around the client area.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	CharCase (see page 21262)	Determines the case of the text within the edit control.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.

	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	HideSelection (see page 21262)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MaxLength (see page 21263)	Specifies the maximum number of characters the user can enter into the edit control.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	OEMConvert (see page 21264)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentColor (see page 21264)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PasswordChar (see page 21265)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ScrollBars (see page 21381)	Determines whether the memo control has scroll bars.
	Text (see page 5842)	Contains a text string associated with the control.
	WantReturns (see page 21384)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 21384)	Determines whether the user can insert tab characters into the text.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.
	WordWrap (see page 21384)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 21245)	Deletes all text from the edit control.
	ClearSelection (see page 21245)	Removes the selected text from the edit control.
	ClearUndo (see page 21246)	Clears the undo buffer so that no changes to the text can be backed out.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopyToClipboard (see page 21246)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
V	Create (see page 21374)	Creates and initializes an instance of TCustomMemo (see page 21362).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CutToClipboard (see page 21249)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 21250)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 21374)	Destroys an instance of TCustomMemo (see page 21362).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
V	GetControlsAlignment (see page 21376)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetSelTextBuf (see page 21253)	Copies the selected text into a buffer and returns the number of characters copied.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	PasteFromClipboard (see page 21254)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.

	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectAll (see page 21255)	Selects all text in the edit control.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSelTextBuf (see page 21258)	Replaces the selected text with a null-terminated string.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Undo (see page 21259)	Backs out all changes in the undo buffer.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	CanUndo (see page 21262)	Indicates whether the edit control contains changes that can be backed out.
	CaretPos (see page 21380)	Indicates the position of the caret in the client area of the memo.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.

R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	Lines (see page 21380)	Contains the individual lines of text in the memo control.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Modified (see page 21263)	Indicates whether the user edited the text of the edit control.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ReadOnly (see page 21265)	Determines whether the user can change the text of the edit control.
	SelLength (see page 21265)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 21269)	Specifies the position of the first selected character in the text.
	SelText (see page 21272)	Specifies the selected portion of the edit control's text.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	Text (see page 21273)	Contains a text string associated with the control.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.

	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	TabStop (see page 21273)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

154.1.14.2 TCustomMemo.Create Constructor

Creates and initializes an instance of TCustomMemo (see page 21362).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomMemo(TComponent * AOwner);
```

Description

Call Create to instantiate a TCustomMemo (see page 21362) object.

AOwner is the component, typically the form, that is responsible for freeing the memo object. This becomes the value of the Owner property.

Derived objects that override the Create method should call the inherited Create before making any additional initializations.

See Also

Owner (see page 2807)

154.1.14.3 TCustomMemo.Destroy Destructor

Destroys an instance of TCustomMemo (see page 21362).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomMemo();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TCustomMemo (see page 21362) reference is not nil, and only then calls Destroy.

Destroy frees the TStrings object that implements the Lines (see page 21380) property before calling the inherited destructor.

See Also

Free (see page 22300)

Lines (see page 21380)

TStrings (see page 3029)

154.1.14.4 TCustomMemo Methods

The methods of the TCustomMemo class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 21375)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateWindowHandle (see page 21376)	Creates a window based on the window-creation parameter record.
	GetCaretPos (see page 21376)	Returns the value of the CaretPos (see page 21380) property.
	KeyPress (see page 21377)	Ensures that return characters are not entered into the text when WantReturns (see page 21384) is false.
	Loaded (see page 21377)	Initializes the memo control when it is first loaded into memory.
	SetAlignment (see page 21377)	Sets the Alignment (see page 21379) property.
	SetCaretPos (see page 21378)	Sets the CaretPos (see page 21380) property.
	SetLines (see page 21378)	Sets the Lines (see page 21380) property.
	SetScrollBars (see page 21378)	Sets the ScrollBars (see page 21381) property.
	SetWordWrap (see page 21379)	Sets the WordWrap (see page 21384) property.

Public Methods

	Name	Description
	GetControlsAlignment (see page 21376)	Indicates how text is aligned within the control.

154.1.14.4.1 TCustomMemo.CreateParams Method

Initializes a window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 21248) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle (see page 21376). TCustomMemo (see page 21362) overrides the inherited method to specify a scrollable multiline edit control. The Params parameter is updated to reflect the current values of the Alignment (see page 21379), ScrollBars (see page 21381), and WordWrap (see page 21384) properties, in addition to the BorderStyle (see page 21261), HideSelection (see page 21262), PasswordChar (see page 21265), and ReadOnly (see page 21265) properties that the inherited method specifies.

See Also

Alignment (see page 21379)

BorderStyle (see page 21261)

CreateWindowHandle (see page 21376)

CreateWnd (see page 21248)

HideSelection (see page 21262)

PasswordChar (see page 21265)

ReadOnly (see page 21265)

ScrollBars (see page 21381)

WordWrap (see page 21384)

154.1.14.4.2 TCustomMemo.CreateWindowHandle Method

Creates a window based on the window-creation parameter record.

Pascal

```
procedure CreateWindowHandle(const Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateWindowHandle(const TCreateParams Params);
```

Description

The CreateWnd (see page 21248) method calls CreateWindowHandle to create the edit control window once it has been specified in the window-creation parameters. CreateWindowHandle creates the window by calling the CreateWindowEx API function, passing parameters from the record passed in the Params parameter. CreateWindowHandle overrides the inherited method to interpret the Caption in Params as the initial text rather than as the window caption.

See Also

CreateParams (see page 21375)

CreateWnd (see page 21248)

154.1.14.4.3 TCustomMemo.GetCaretPos Method

Returns the value of the CaretPos (see page 21380) property.

Pascal

```
function GetCaretPos: TPoint; virtual;
```

C++

```
virtual __fastcall TPoint GetCaretPos();
```

Description

CaretPos (see page 21380) represents the coordinates of the cursor. GetCaretPos is the protected implementation of the CaretPos (see page 21380) property. Override this method to change the way the caret position is computed.

See Also

CaretPos (see page 21380)

154.1.14.4.4 TCustomMemo.GetControlsAlignment Method

Indicates how text is aligned within the control.

Pascal

```
function GetControlsAlignment: TAlignment; override;
```

C++

```
virtual __fastcall TAlignment GetControlsAlignment();
```

Description

Use GetControlsAlignment to determine the alignment of the control. For controls that do not have an Alignment property, GetControlsAlignment returns taLeftJustify.

Note: GetControlsAlignment returns the alignment of text before it is altered by the BiDiMode property. To check if the alignment should be reversed to reflect the BiDiMode property, call UseRightToLeftAlignment.

See Also

BiDiMode (see page 5803)

Text (see page 5842)

Caption (see page 5806)

UseRightToLeftAlignment (see page 5790)

154.1.14.4.5 TCustomMemo.KeyPress Method

Ensures that return characters are not entered into the text when WantReturns (see page 21384) is false.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress is called automatically when the user presses a key. After calling the inherited KeyPress, which calls the OnKeyPress event handler, KeyPress discards return characters when WantReturns (see page 21384) is false.

See Also

WantReturns (see page 21384)

154.1.14.4.6 TCustomMemo.Loaded Method

Initializes the memo control when it is first loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Loaded overrides the inherited method so that the Modified (see page 21263) property is set to false when the form containing the memo is first loaded into memory. Derived classes can override the Loaded method to perform additional initializations after the memo's form is loaded.

See Also

Modified (see page 21263)

154.1.14.4.7 TCustomMemo.SetAlignment Method

Sets the Alignment (see page 21379) property.

Pascal

```
procedure SetAlignment(Value: TAlignment);
```

C++

```
__fastcall SetAlignment(TAlignment Value);
```

Description

SetAlignment is exposed as a protected method so derived classes can customize their implementation of the Alignment (see page 21379) property.

See Also

Alignment (see page 21379)

154.1.14.4.8 TCustomMemo.SetCaretPos Method

Sets the CaretPos (see page 21380) property.

Pascal

```
procedure SetCaretPos(const Value: TPoint); virtual;
```

C++

```
virtual __fastcall SetCaretPos(const TPoint Value);
```

Description

SetCaretPos is the property write access method for the CaretPos (see page 21380) property. The CaretPos (see page 21380) property returns the coordinates of the cursor. SetCaretPos sets the cursor position to Value.

See Also

CaretPos (see page 21380)

154.1.14.4.9 TCustomMemo.SetLines Method

Sets the Lines (see page 21380) property.

Pascal

```
procedure SetLines(Value: TStrings);
```

C++

```
__fastcall SetLines(TStrings Value);
```

Description

SetLines is the protected write implementation for the Lines (see page 21380) property.

SetLines is exposed as a protected method so derived classes can customize their implementation of the Lines (see page 21380) property.

See Also

Lines (see page 21380)

154.1.14.4.10 TCustomMemo.SetScrollBars Method

Sets the ScrollBars (see page 21381) property.

Pascal

```
procedure SetScrollBars(Value: TScrollStyle);
```

C++

```
__fastcall SetScrollBars(TScrollStyle Value);
```

Description

SetScrollBars is the property access method for the ScrollBars (see page 21381) property.

SetScrollBars is exposed as a protected method so derived classes can customize their implementation of the ScrollBars (see page 21381) property.

See Also

ScrollBars (see page 21381)

154.1.14.4.11 TCustomMemo.SetWordWrap Method

Sets the WordWrap (see page 21384) property.

Pascal

```
procedure SetWordWrap(Value: Boolean);
```

C++

```
__fastcall SetWordWrap(Boolean Value);
```

Description

SetWordWrap is the property access method for the WordWrap (see page 21384) property.

See Also

WordWrap (see page 21384)

154.1.14.5 TCustomMemo Properties

The properties of the TCustomMemo class are listed here.

Protected Properties

	Name	Description
	Alignment (see page 21379)	Determines how the text is aligned within the memo control.
	ScrollBars (see page 21381)	Determines whether the memo control has scroll bars.
	WantReturns (see page 21384)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 21384)	Determines whether the user can insert tab characters into the text.
	WordWrap (see page 21384)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

Public Properties

	Name	Description
	CaretPos (see page 21380)	Indicates the position of the caret in the client area of the memo.
	Lines (see page 21380)	Contains the individual lines of text in the memo control.

154.1.14.5.1 TCustomMemo.Alignment Property

Determines how the text is aligned within the memo control.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to change the way the text is formatted by the memo control. Alignment can take one of the following values:

Value	Meaning
taLeftJustify	Align text to the left side of the control
taCenter	Center text horizontally in the control.
taRightJustify	Align text to the right side of the control.

See Also

SetAlignment (see page 21377)

Text (see page 5842)

Displaying and Editing Text in a Memo Control

154.1.14.5.2 TCustomMemo.CaretPos Property

Indicates the position of the caret in the client area of the memo.

Pascal

```
property CaretPos: TPoint;
```

C++

```
__property TPoint CaretPos;
```

Description

Use CaretPos to determine the coordinates of the cursor. CaretPos defines a location onscreen, in characters, with the origin in the top left corner of the memo. X specifies the horizontal coordinate of the point, Y specifies the vertical coordinate.

To determine the position of the caret in terms of characters of text only (rather than X/Y location), use the SelStart (see page 21269) property.

See Also

ClientRect (see page 5810)

ClientOrigin (see page 5809)

SelStart (see page 21269)

154.1.14.5.3 TCustomMemo.Lines Property

Contains the individual lines of text in the memo control.

Pascal

```
property Lines: TStrings;
```

C++

```
__property TStrings Lines;
```

Description

Use Lines to manipulate text in an memo control on a line-by-line basis. Lines is a TStrings object, so the TStrings methods may

be used for Lines to perform manipulations such as counting the lines of text, adding new lines, deleting lines, or replacing lines with new text.

To work with all the text at once, use the Text (see page 21273) property. To manipulate individual lines of text, the Lines property works better.

Note: Although Lines is implemented as a TStrings descendant, it does not implement the support for associating objects with the strings in the list.

C++ Examples:

```
/*
This example uses a button and a memo control on a form.
When the user clicks the button, the content of the specified
file is loaded into the memo, and the fifth line of the file
is written across the top of the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Memo1->Lines->LoadFromFile("../readme.txt");
  Caption = Memo1->Lines->Strings[4];
}
```

Delphi Examples:

```
{
This example uses a button and a memo control on a form.
When the user clicks the button, the content of the specified
file is loaded into the memo, and the fifth line of the file
is written across the top of the form.
}

procedure TForm1.Button1Click(Sender: TObject);
begin
  Memo1.Lines.LoadFromFile('readme.txt');
  Caption:= Memo1.Lines[4];
end;
```

See Also

TStrings (see page 3029)

SelText (see page 21272)

SetLines (see page 21378)

Text (see page 5842)

WordWrap (see page 21384)

154.1.14.5.4 TCustomMemo.ScrollBars Property

Determines whether the memo control has scroll bars.

Pascal

```
property ScrollBars: TScrollStyle;
```

C++

```
__property TScrollStyle ScrollBars;
```

Description

Use ScrollBars to give a multiline edit control horizontal or vertical scroll bars. ScrollBars can take one of the following values:

Value	Meaning
ssNone	The control has no scroll bars.
ssHorizontal	The control has a single scroll bar on the bottom edge.
ssVertical	The control has a single scroll bar on the right edge.
ssBoth	The control has a scroll bar on both the bottom and right edges.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
    ssHorizontal,
    ssNone,
    ssVertical);
begin
  RichEdit1.ScrollBars :=
    ScrollBarA[RadioGroup1.ItemIndex];
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with RadioGroup1, Items do
  begin
    Caption := 'Scroll Bars';
    Clear;
    Add('ssBoth');
    Add('ssHorizontal');
    Add('ssNone');
    Add('ssVertical');
  end;
  with CheckBox1 do
  begin
    Caption := 'HideScrollBars';
    Checked := True;
  end;
  Button1.Caption := 'Loaded';
  RichEdit1.Text := 'Just a little text.';
  RichEdit1.WordWrap := False;
end;
```

See Also

SetScrollBars (see page 21378)

WordWrap (see page 21384)

Displaying and Editing Text in a Memo Control

154.1.14.5.5 TCustomMemo.WantReturns Property

Determines whether the user can insert return characters into the text.

Pascal

```
property WantReturns: Boolean;
```

C++

```
__property Boolean WantReturns;
```

Description

Set WantReturns to true to allow users to enter return characters into the text. Set WantReturns to false to allow the form to handle return characters instead.

For example, in a form with a default button (such as an OK button) and a memo control, if WantReturns is false, pressing Enter chooses the default button. If WantReturns is true, pressing Enter inserts a return character in the text.

Note: If WantReturns is false, users can still enter return characters into the text by pressing Ctrl+Enter.

See Also

Text (see page 5842)

WantTabs (see page 21384)

WordWrap (see page 21384)

154.1.14.5.6 TCustomMemo.WantTabs Property

Determines whether the user can insert tab characters into the text.

Pascal

```
property WantTabs: Boolean;
```

C++

```
__property Boolean WantTabs;
```

Description

Set WantTabs to true to allow users to enter tab characters into the text. Set WantTabs to false if you want the tab character to select the next control on the form instead.

Note: If WantTabs is true, users can tab into the edit control, but they can't tab out.

See Also

Text (see page 5842)

WantReturns (see page 21384)

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

154.1.14.5.7 TCustomMemo.WordWrap Property

Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to make the edit control wrap text at the right margin so it fits in the client area. The wrapping is cosmetic only. The text does not include any return characters that were not explicitly entered. Set WordWrap to false to have the edit control show a separate line only where return characters were explicitly entered into the text.

Note: There should be no use for a horizontal scroll bar if WordWrap is true.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
    ssHorizontal,
    ssNone,
    ssVertical);
begin
  RichEdit1.ScrollBars :=
    ScrollBarA[RadioGroup1.ItemIndex];
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with RadioGroup1, Items do
  begin
    Caption := 'Scroll Bars';
    Clear;
    Add('ssBoth');
    Add('ssHorizontal');
    Add('ssNone');
    Add('ssVertical');
  end;
  with CheckBox1 do
  begin
    Caption := 'HideScrollBars';
    Checked := True;
  end;
  Button1.Caption := 'Loaded';
  RichEdit1.Text := 'Just a little text.';
  RichEdit1.WordWrap := False;
end;
```

See Also

Lines (see page 21380)

ScrollBars (see page 21381)

Text (see page 5842)

WantReturns (see page 21384)

Displaying and Editing Text in a Memo Control

154.1.15 TCustomStaticText Class

TCustomStaticText is the base class for TStaticText (see page 21887).

Class Hierarchy

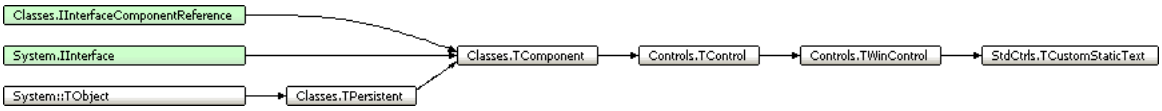

Pascal

```
TCustomStaticText = class(TWinControl);
```

C++

```
class TCustomStaticText : public TWinControl;
```

File

StdCtrls

Description

TCustomStaticText is the base class for TStaticText (see page 21887), a windowed control that displays text on a form. This component functions like TLabel (see page 21544), except that it has a window handle.

TCustomStaticText cannot receive focus or contain other controls.

See Also

TLabel (see page 21544)

TStaticText (see page 21887)

154.1.15.1 TCustomStaticText Members

The following tables list the members exposed by TCustomStaticText.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.

	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.

	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 21398)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.

	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 21399)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 21399)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 21399)	This is SetAutoSize, a member of class TCustomStaticText.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	Alignment (see page 21400)	Determines how text is aligned within the component.
	AutoSize (see page 21400)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderStyle (see page 21401)	Determines the appearance of the control's border.

	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	FocusControl (see page 21401)	Links the component with another windowed control.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ShowAccelChar (see page 21401)	Enables accelerator characters.
	Text (see page 5842)	Contains a text string associated with the control.
	Transparent (see page 21402)	Specifies whether the background of the image obscures objects below the image object.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 21397)	Creates and initializes a TCustomStaticText (see page 21387) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.

	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.

V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.

R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

154.1.15.2 TCustomStaticText.Create Constructor

Creates and initializes a TCustomStaticText (see page 21387) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomStaticText(TComponent * AOwner);
```

Description

Create calls the inherited constructor, then initializes the TCustomStaticText (see page 21387) instance

See Also

autosize (see page 21400)

ControlStyle (see page 5814)

ShowAccelChar (see page 21401)

154.1.15.3 TCustomStaticText Methods

The methods of the TCustomStaticText class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 21398)	Initializes a window-creation parameter data structure.
	Loaded (see page 21399)	Initializes the control after it is loaded from a stream.
	Notification (see page 21399)	Responds to notifications that components are being created or destroyed.
	SetAutoSize (see page 21399)	This is SetAutoSize, a member of class TCustomStaticText.

154.1.15.3.1 TCustomStaticText.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

154.1.15.3.2 TCustomStaticText.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded (see page 2782)

Action (see page 5797)

ActionChange (see page 5737)

154.1.15.3.3 TCustomStaticText.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

154.1.15.3.4 TCustomStaticText.SetAutoSize Method

Pascal

```
procedure SetAutoSize(Value: Boolean); override;
```

C++

```
virtual __fastcall SetAutoSize(Boolean Value);
```

Description

This is SetAutoSize, a member of class TCustomStaticText.

154.1.15.4 TCustomStaticText Properties

The properties of the TCustomStaticText class are listed here.

Protected Properties

	Name	Description
	Alignment (see page 21400)	Determines how text is aligned within the component.
	AutoSize (see page 21400)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BorderStyle (see page 21401)	Determines the appearance of the control's border.
	FocusControl (see page 21401)	Links the component with another windowed control.
	ShowAccelChar (see page 21401)	Enables accelerator characters.
	Transparent (see page 21402)	Specifies whether the background of the image obscures objects below the image object.

154.1.15.4.1 TCustomStaticText.Alignment Property

Determines how text is aligned within the component.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

The possible values of Alignment are

Value	Meaning
taLeftJustify	Text is aligned at the left of the control.
taCenter	Text is centered in the control.
taRightJustify	Text is aligned at the right of the control.

See Also

Font (see page 5823)

Caption (see page 5806)

154.1.15.4.2 TCustomStaticText.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints (see page 5812)

CanAutoSize (see page 5741)

154.1.15.4.3 TCustomStaticText.BorderStyle Property

Determines the appearance of the control's border.

Pascal

```
property BorderStyle: TStaticBorderStyle;
```

C++

```
__property TStaticBorderStyle BorderStyle;
```

Description

Use BorderStyle to specify the appearance of the border, if any, that appears around the text.

154.1.15.4.4 TCustomStaticText.FocusControl Property

Links the component with another windowed control.

Pascal

```
property FocusControl: TWinControl;
```

C++

```
__property TWinControl * FocusControl;
```

Description

Use the FocusControl property to link the TCustomStaticText (see page 21387) instance with another windowed control. If the text of Caption includes an accelerator key, the component specified by FocusControl receives focus when the accelerator is pressed.

See Also

Caption (see page 5806)

FocusControl (see page 21306)

TWinControl (see page 6112)

154.1.15.4.5 TCustomStaticText.ShowAccelChar Property

Enables accelerator characters.

Pascal

```
property ShowAccelChar: Boolean;
```

C++

```
__property Boolean ShowAccelChar;
```

Description

If ShowAccelChar is true, an ampersand in the component's Caption text appears as an underline and makes the character after it into an accelerator key. If ShowAccelChar is false, ampersands appear as ampersands.

See Also

Caption (see page 5806)

154.1.15.4.6 TCustomStaticText.Transparent Property

Specifies whether the background of the image obscures objects below the image object.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Set Transparent to true to allow objects behind the TCustomStaticText (see page 21387) object to show through the background of the image. Set Transparent to false to make the background of the image opaque.

154.1.16 TEdit Class

TEdit is a wrapper for a Windows single-line edit control.

Class Hierarchy

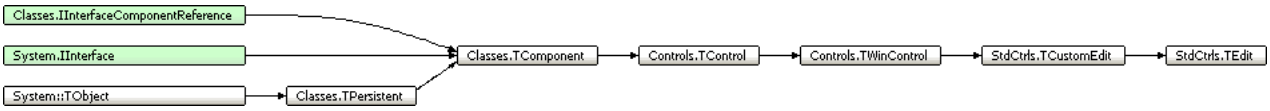

Pascal

```
TEdit = class(TCustomEdit);
```

C++

```
class TEdit : public TCustomEdit;
```

File

StdCtrls

Description

Use a TEdit object to put a standard Windows edit control on a form. Edit controls are used to retrieve text that users type. Edit controls can also display text to the user.

When only displaying text to the user, choose an edit control to allow users to select text and copy it to the Clipboard. Choose a label object if the selection capabilities of an edit control are not needed.

TEdit implements the generic behavior introduced in TCustomEdit (see page 21232). TEdit publishes many of the properties inherited from TCustomEdit (see page 21232), but does not introduce any new behavior. For specialized edit controls, use other descendant classes of TCustomEdit (see page 21232) or derive from it.

See Also

TDBEdit (see page 8735)

TDBMemo (see page 9149)

TInplaceEdit (see page 14938)

TLabel (see page 21544)

TMaskEdit (see page 15824)

TMemo (see page 21678)

TRichEdit (see page 4685)

Displaying Field Component Values in Standard Controls

Edit Controls

154.1.16.1 TEdit Members

The following tables list the members exposed by TEdit.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21245)	Generates an OnChange (see page 21275) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 21248)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWindowHandle (see page 21248)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CreateWnd (see page 21248)	Creates a Windows control corresponding to the edit control.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.

	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
V	DestroyWnd (see page 21251)	Destroys the Windows control that corresponds to the edit control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
V	DoSetMaxLength (see page 21251)	Sets the MaxLength (see page 21263) property.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.

	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelLength (see page 21252)	Returns the SelLength (see page 21265) property.
	GetSelStart (see page 21252)	Returns the SelStart (see page 21269) property.
	GetSelText (see page 21252)	Returns the SelText (see page 21272) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.

	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetAutoSize (see page 21257)	Sets the AutoSize (see page 21260) property.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetSelLength (see page 21257)	Sets the SelLength (see page 21265) property.
	SetSelStart (see page 21257)	Sets the SelStart (see page 21269) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 21245)	Deletes all text from the edit control.
	ClearSelection (see page 21245)	Removes the selected text from the edit control.
	ClearUndo (see page 21246)	Clears the undo buffer so that no changes to the text can be backed out.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.

	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopyToClipboard (see page 21246)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 21244)	Creates an instance of TCustomEdit (see page 21232).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CutToClipboard (see page 21249)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 21250)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
V	GetSelTextBuf (see page 21253)	Copies the selected text into a buffer and returns the number of characters copied.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	PasteFromClipboard (see page 21254)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectAll (see page 21255)	Selects all text in the edit control.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSelTextBuf (see page 21258)	Replaces the selected text with a null-terminated string.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Undo (see page 21259)	Backs out all changes in the undo buffer.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	CanUndo (see page 21262)	Indicates whether the edit control contains changes that can be backed out.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.

	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Modified (see page 21263)	Indicates whether the user edited the text of the edit control.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelLength (see page 21265)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 21269)	Specifies the position of the first selected character in the text.
	SelText (see page 21272)	Specifies the selected portion of the edit control's text.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 21445)	Occurs when the text for the edit control may have changed.
	OnClick (see page 21446)	Occurs when the user clicks the control.
	OnContextPopup (see page 21448)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 21448)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 21449)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21451)	Occurs when the user drags an object over a control.
	OnEndDock (see page 21453)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21454)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 21455)	Occurs when a control receives the input focus.
	OnExit (see page 21457)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 21459)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 21462)	Occurs when a key is pressed.
	OnKeyUp (see page 21463)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 21464)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 21465)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 21468)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21468)	Occurs when the user moves the mouse outside of a control.

	OnMouseMove (see page 21468)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 21470)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 21473)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21473)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 21415)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 21418)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 21419)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 21419)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 21420)	Specifies which edges of the control are beveled.
	BevelInner (see page 21421)	Specifies the cut of the inner bevel.
	BevelKind (see page 21421)	Specifies the control's bevel style.
	BevelOuter (see page 21422)	Specifies the cut of the outer bevel.
	BevelWidth (see page 21422)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 21423)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 21423)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 21423)	Determines the case of the text within the edit control.
	Color (see page 21424)	Specifies the background color of the control.
	Constraints (see page 21425)	Specifies the size constraints for the control.
	Ctl3D (see page 21425)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 21426)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21426)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21426)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 21428)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 21428)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	HideSelection (see page 21429)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 21430)	Determines the behavior of the input method editor (IME).
	ImeName (see page 21430)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	MaxLength (see page 21431)	Specifies the maximum number of characters the user can enter into the edit control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OEMConvert (see page 21432)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBiDiMode (see page 21432)	Specifies whether the control uses its parent's BiDiMode.

	ParentColor (see page 21432)	Determines where a control looks for its color information.
	ParentCtl3D (see page 21433)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 21433)	Determines where a control looks for its font information.
	ParentShowHint (see page 21434)	Determines where a control looks to find out if its Help Hint should be shown.
	PasswordChar (see page 21434)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 21435)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 21439)	Determines whether the user can change the text of the edit control.
	ShowHint (see page 21440)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 21441)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21442)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Text (see page 21442)	Contains a text string associated with the control.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 21443)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

154.1.16.2 TEdit Properties

The properties of the TEdit class are listed here.

Published Properties

	Name	Description
	Align (see page 21415)	Determines how the control aligns within its container (parent control).
	Anchors (see page 21418)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 21419)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 21419)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 21420)	Specifies which edges of the control are beveled.
	BevelInner (see page 21421)	Specifies the cut of the inner bevel.
	BevelKind (see page 21421)	Specifies the control's bevel style.
	BevelOuter (see page 21422)	Specifies the cut of the outer bevel.
	BevelWidth (see page 21422)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 21423)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 21423)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 21423)	Determines the case of the text within the edit control.
	Color (see page 21424)	Specifies the background color of the control.
	Constraints (see page 21425)	Specifies the size constraints for the control.
	Ctl3D (see page 21425)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 21426)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21426)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21426)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 21428)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 21428)	Controls the attributes of text written on or in the control.
	HideSelection (see page 21429)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 21430)	Determines the behavior of the input method editor (IME).
	ImeName (see page 21430)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

	MaxLength (see page 21431)	Specifies the maximum number of characters the user can enter into the edit control.
	OEMConvert (see page 21432)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBiDiMode (see page 21432)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 21432)	Determines where a control looks for its color information.
	ParentCtl3D (see page 21433)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 21433)	Determines where a control looks for its font information.
	ParentShowHint (see page 21434)	Determines where a control looks to find out if its Help Hint should be shown.
	PasswordChar (see page 21434)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 21435)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 21439)	Determines whether the user can change the text of the edit control.
	ShowHint (see page 21440)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 21441)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21442)	Determines if the user can tab to a control.
	Text (see page 21442)	Contains a text string associated with the control.
	Visible (see page 21443)	Determines whether the component appears onscreen.

154.1.16.2.1 TEdit.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
```

```
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
```

```
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

154.1.16.2.2 TEdit.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

154.1.16.2.3 TEdit.AutoSelect Property

Determines whether all the text in the edit control is automatically selected when the control gets focus.

Pascal

```
property AutoSelect: Boolean;
```

C++

```
__property Boolean AutoSelect;
```

Description

Set AutoSelect to select all the text when the edit control gets focus. AutoSelect only applies to single-line edit controls.

Use AutoSelect when the user is more likely to replace the text in the edit control than to append to it.

See Also

SelectAll (see page 21255)

SelLength (see page 21265)

SelStart (see page 21269)

SelText (see page 21272)

SetSelTextBuf (see page 21258)

154.1.16.2.4 TEdit.AutoSize Property

Determines whether the height of the edit control automatically resizes to accommodate the text.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;__property Boolean AutoSize;
```

Description

Use AutoSize to make the edit control adjust its size automatically so the client area accommodates the height of the text. When AutoSize is false, the edit control has a fixed height. When AutoSize is true, the size of the control is readjusted whenever a change occurs that could affect the height of the control, such as a change to the font or border style.

Note: AutoSize only adjusts the height of the edit control. Even with AutoSize set to true, text in the edit control may appear truncated if it extends beyond the width of the control.

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit1->AutoSize = true;
  Edit1->Font->Size = 20;
  Edit1->Text = "The edit box is taller now";
}
```

Delphi Examples:

```
{
{
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.AutoSize := True;
  Edit1.Font.Size := 20;
  Edit1.Text := 'The edit box is taller now';
end;
```

See Also

BorderStyle (see page 21261)

ClientHeight (see page 5808)

ControlStyle (see page 5814)

Height (see page 5825)

154.1.16.2.5 TEdit.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

154.1.16.2.6 TEdit.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

154.1.16.2.7 TEdit.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

154.1.16.2.8 TEdit.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

154.1.16.2.9 TEdit.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

154.1.16.2.10 TEdit.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

154.1.16.2.11 TEdit.BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the edit control stands out. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the edit control has a single-line border around the client area. If BorderStyle is bsNone, there will be no border.

154.1.16.2.12 TEdit.CharCase Property

Determines the case of the text within the edit control.

Pascal

```
property CharCase: TEditCharCase;
```

C++

```
__property TEditCharCase CharCase;
```

Description

Use CharCase to force the contents of the edit control to assume a particular case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types them into the edit control. Changing the CharCase property to ecLowerCase or ecUpperCase changes the actual contents of the text, not just the appearance. Any case information is lost and can't be recaptured by changing CharCase to ecNormal.

See Also

Text (see page 5842)

154.1.16.2.13 TEdit.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

154.1.16.2.14 TEdit.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

154.1.16.2.15 TEdit.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

154.1.16.2.16 TEdit.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

154.1.16.2.17 TEdit.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

154.1.16.2.18 TEdit.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime

by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
```

```
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

154.1.16.2.19 TEdit.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

154.1.16.2.20 TEdit.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

154.1.16.2.21 TEdit.HideSelection Property

Determines whether the visual indication of the selected text remains when focus shifts to another control.

Pascal

```
property HideSelection: Boolean;
```

C++

```
__property Boolean HideSelection;
```

Description

Set HideSelection to false to provide visual feedback of the selected portion of the text even when the edit control does not have focus. Set HideSelection to true to show the selection only when the edit control has focus. HideSelection does not affect the actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms with many edit controls look too busy.

See Also

SelLength (see page 21265)

SelStart (see page 21269)

SelText (see page 21272)

SetSelTextBuf (see page 21258)

154.1.16.2.22 TEdit.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

154.1.16.2.23 TEdit.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

154.1.16.2.24 TEdit.MaxLength Property

Specifies the maximum number of characters the user can enter into the edit control.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the edit control. A value of 0 indicates that there is no application-defined limit on the length.

Use MaxLength to limit the length of the text in an edit control if that text will be copied into a fixed-length buffer.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

Note: The unit of MaxLength is characters (not bytes) if the application is using themes.

See Also

Text (see page 5842)

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

154.1.16.2.25 TEdit.OEMConvert Property

Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.

Pascal

```
property OEMConvert: Boolean;
```

C++

```
__property Boolean OEMConvert;
```

Description

To ensure that any characters typed in the edit control can be unambiguously converted to the OEM character set, set OEMConvert to true. This causes all characters to be converted from ANSI to OEM and then back to ANSI as they are typed. Set OEMConvert to false to avoid the overhead of this conversion when it does not matter whether the text can be unambiguously mapped to an OEM string.

OEMConvert is most useful for edit controls used to enter file names when the application does not use Unicode file names.

See Also

Text (see page 5842)

154.1.16.2.26 TEdit.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

154.1.16.2.27 TEdit.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

154.1.16.2.28 TEdit.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

154.1.16.2.29 TEdit.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

154.1.16.2.30 TEdit.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

154.1.16.2.31 TEdit.PasswordChar Property

Indicates the character, if any, to display in place of the actual characters typed in the control.

Pascal

```
property PasswordChar: Char;
```

C++

```
__property Char PasswordChar;
```

Description

Use the PasswordChar property to create an edit control that displays a special character in place of any entered text. If PasswordChar is set to the null character (ANSI character zero), the edit control displays its text normally. If PasswordChar is any other character, the edit control displays PasswordChar in place of each character typed. PasswordChar affects the appearance of the edit control only. The value of the Text property reflects the actual characters that are typed.

See Also

Text (see page 5842)

154.1.16.2.32 TEdit.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
```

```
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      const int num_items = 4;
      for (int index = 0; index < num_items; ++index)
      {
          TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
          PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
          NewItem->Caption = "Menu Item " + IntToStr(index);
          NewItem->Tag = index;
          NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
          TNotifyEvent();
      }
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
          {
              case 0:
              {
                  ShowMessage("first item clicked");
                  break;
              }
              case 1:
              {
                  ShowMessage("second item clicked");
                  break;
              }
              case 2:
              {
```

```
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
```

```
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
```

```
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

154.1.16.2.33 TEdit.ReadOnly Property

Determines whether the user can change the text of the edit control.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

Setting ReadOnly to true ensures that the text is not altered, while still allowing the user to select text. The selected text can then be manipulated by the application, or copied to the Clipboard.

See Also

CopyToClipboard (see page 21246)

GetSelTextBuf (see page 21253)

154.1.16.2.34 TEdit.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
```

```
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

154.1.16.2.35 TEdit.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore

cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

154.1.16.2.36 TEdit.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

154.1.16.2.37 TEdit.Text Property

Contains a text string associated with the control.

Pascal

```
property Text: TCaption;
```

C++

```
__property TCaption Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
```

```
  Size++;                       //Add room for null character
  char *Buffer = new char[Size];  //Creates Buffer dynamic variable
  Edit1->GetTextBuf(Buffer,Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer;
  delete [] Buffer;
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
  Inc(Size);                      {Add room for null character}
  GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
  Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
end;
```

See Also

Caption (see page 5806)

Lines (see page 21380)

154.1.16.2.38 TEdit.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
```

```
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

154.1.16.3 TEdit Events

The events of the TEdit class are listed here.

Published Events

	Name	Description
	OnChange (see page 21445)	Occurs when the text for the edit control may have changed.
	OnClick (see page 21446)	Occurs when the user clicks the control.
	OnContextPopup (see page 21448)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 21448)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 21449)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21451)	Occurs when the user drags an object over a control.
	OnEndDock (see page 21453)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21454)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 21455)	Occurs when a control receives the input focus.
	OnExit (see page 21457)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 21459)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 21462)	Occurs when a key is pressed.
	OnKeyUp (see page 21463)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 21464)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 21465)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 21468)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21468)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 21468)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 21470)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 21473)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21473)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

154.1.16.3.1 TEdit.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change (see page 21245)

Text (see page 5842)

154.1.16.3.2 TEdit.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
```

```
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

154.1.16.3.3 TEdit.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

154.1.16.3.4 TEdit.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

154.1.16.3.5 TEdit.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
```

```
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

154.1.16.3.6 TEdit.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
```

```
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.16.3.7 TEdit.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.16.3.8 TEdit.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

154.1.16.3.9 TEdit.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
```

```
    ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
```

```
    else if myActiveControl is TRichEdit then
      with myActiveControl as TRichEdit do
        SelAttributes.Assign(TFontDialog(Sender).Font)
    else
      Beep;
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

154.1.16.3.10 TEdit.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
```

```
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

154.1.16.3.11 TEdit.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
```

```
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

```
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

154.1.16.3.12 TEdit.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

154.1.16.3.13 TEdit.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

154.1.16.3.14 TEdit.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.16.3.15 TEdit.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
```

```
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.16.3.16 TEdit.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

154.1.16.3.17 TEdit.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

154.1.16.3.18 TEdit.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  */
  int StartX, StartY; // Declare at the top of the form's unit
  // Use this code as the OnMouseDown event handler of the form:
  void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
  int X, int Y)
  {
    StartX = X;
    StartY = Y;
  }

  // Use this code as the OnMouseUp event handler of the form:
  void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
  int X, int Y)
  {
    Form1->Canvas->Rectangle(StartX, StartY, X, Y);
    StatusBar1->Panels->Items[0]->Text = "";
    StatusBar1->Panels->Items[1]->Text = "";
    StatusBar1->Panels->Items[2]->Text = "";
    StatusBar1->Panels->Items[3]->Text = "";
  }

  // Use this code as the OnMouseMove event handler of the form:
  void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
  {
    if (Shift.Contains(ssLeft)) // make sure button is down
    {
      if (Y > StartY)
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
      }
      else
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
  {
  The following example requires a form with a four-paneled
  status bar.  (Set the Width of the status panels to 150
  before running this example).  When the user presses a mouse
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
```

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.16.3.19 TEdit.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

154.1.16.3.20 TEdit.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

154.1.16.3.21 TEdit.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

154.1.17 TGroupBox Class

TGroupBox represents a Windows group box.

Class Hierarchy

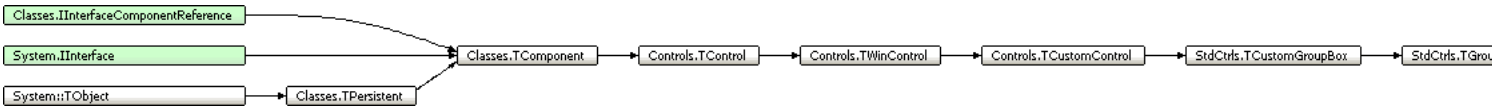

Pascal

```
TGroupBox = class(TCustomGroupBox);
```

C++

```
class TGroupBox : public TCustomGroupBox;
```

File

StdCtrls

Description

The TGroupBox component represents a standard Windows group box, used to group related controls on a form. When another control component is placed within a group box, the group box becomes the parent of that component.

See Also

TRadioGroup (see page 13115)

TDBRadioGroup (see page 9283)

TControl (see page 5724)

Group Boxes and Radio Groups

Grouping Controls

154.1.17.1 TGroupBox Members

The following tables list the members exposed by TGroupBox.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 21287)	Adjusts the ClientRect property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.

	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 21287)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.

	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..

	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 5920)	Renders the image of a custom control.
V	Paint (see page 21287)	Renders the onscreen image of the group box.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.

V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 5731)	Creates an instance of TControl (see page 5724).
	Create (see page 5918)	Creates an instance of TCustomControl (see page 5908).
	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 21286)	Creates and initializes a TCustomGroupBox (see page 21275) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 5919)	Destroys an instance of TCustomControl (see page 5908).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.

	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.

V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.

R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 21511)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 21512)	Occurs when an object with custom alignment is aligned.
	OnClick (see page 21513)	Occurs when the user clicks the control.
	OnContextPopup (see page 21514)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 21515)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 21515)	Occurs when another control is docked to the control.
	OnDockOver (see page 21517)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 21520)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21522)	Occurs when the user drags an object over a control.

	OnEndDock (see page 21524)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21525)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 21526)	Occurs when a control receives the input focus.
	OnExit (see page 21528)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 21530)	Returns the control's docking information.
	OnMouseActivate (see page 21532)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 21533)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 21535)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21535)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 21535)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 21538)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 21540)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21541)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 21541)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 21485)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 21489)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 21490)	Specifies the bi-directional mode for the control.
	Caption (see page 21490)	Specifies a text string that identifies the control to the user.
	Color (see page 21493)	Specifies the background color of the control.
	Constraints (see page 21493)	Specifies the size constraints for the control.
	Ctl3D (see page 21494)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DockSite (see page 21495)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 21496)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21496)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21496)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 21498)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 21498)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Padding (see page 21499)	Specifies the padding of a control.
	ParentBackground (see page 21500)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 21500)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 21500)	Determines where a control looks for its color information.

	ParentCtl3D (see page 21501)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 21501)	Determines where a control looks for its font information.
	ParentShowHint (see page 21502)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 21502)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 21507)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 21508)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21509)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 21509)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

154.1.17.2 TGroupBox Properties

The properties of the TGroupBox class are listed here.

Published Properties

	Name	Description
	Align (see page 21485)	Determines how the control aligns within its container (parent control).
	Anchors (see page 21489)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 21490)	Specifies the bi-directional mode for the control.
	Caption (see page 21490)	Specifies a text string that identifies the control to the user.
	Color (see page 21493)	Specifies the background color of the control.
	Constraints (see page 21493)	Specifies the size constraints for the control.
	Ctl3D (see page 21494)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 21495)	Specifies whether the control can be the target of drag-and-dock operations.
	DragCursor (see page 21496)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21496)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21496)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 21498)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 21498)	Controls the attributes of text written on or in the control.
	Padding (see page 21499)	Specifies the padding of a control.
	ParentBackground (see page 21500)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 21500)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 21500)	Determines where a control looks for its color information.
	ParentCtl3D (see page 21501)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 21501)	Determines where a control looks for its font information.
	ParentShowHint (see page 21502)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 21502)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 21507)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 21508)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21509)	Determines if the user can tab to a control.
	Visible (see page 21509)	Determines whether the component appears onscreen.

154.1.17.2.1 TGroupBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;
```

```
  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
```

```
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
```

```
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

154.1.17.2.2 TGroupBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

154.1.17.2.3 TGroupBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

154.1.17.2.4 TGroupBox.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
```

```
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
```

```
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

154.1.17.2.5 TGroupBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

154.1.17.2.6 TGroupBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

154.1.17.2.7 TGroupBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

154.1.17.2.8 TGroupBox.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite: Boolean;
```

C++

```
__property Boolean DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

154.1.17.2.9 TGroupBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

154.1.17.2.10 TGroupBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

154.1.17.2.11 TGroupBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

154.1.17.2.12 TGroupBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

154.1.17.2.13 TGroupBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

154.1.17.2.14 TGroupBox.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding: TPadding;
```

C++

```
__property TPadding Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding (see page 6103)

154.1.17.2.15 TGroupBox.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground: Boolean;
```

C++

```
__property Boolean ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

154.1.17.2.16 TGroupBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

154.1.17.2.17 TGroupBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

154.1.17.2.18 TGroupBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

154.1.17.2.19 TGroupBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

154.1.17.2.20 TGroupBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

154.1.17.2.21 TGroupBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
```

```
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
```

```
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
```

```
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
```

```
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
```

```
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

154.1.17.2.22 TGroupBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
```

```
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

154.1.17.2.23 TGroupBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

154.1.17.2.24 TGroupBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

154.1.17.2.25 TGroupBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
```

```
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

154.1.17.3 TGroupBox Events

The events of the TGroupBox class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 21511)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 21512)	Occurs when an object with custom alignment is aligned.
	OnClick (see page 21513)	Occurs when the user clicks the control.
	OnContextPopup (see page 21514)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 21515)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 21515)	Occurs when another control is docked to the control.
	OnDockOver (see page 21517)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 21520)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21522)	Occurs when the user drags an object over a control.
	OnEndDock (see page 21524)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21525)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 21526)	Occurs when a control receives the input focus.
	OnExit (see page 21528)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 21530)	Returns the control's docking information.
	OnMouseActivate (see page 21532)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 21533)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 21535)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21535)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 21535)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 21538)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 21540)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21541)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 21541)	Occurs when the application tries to undock a control that is docked to the windowed control.

154.1.17.3.1 TGroupBox.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore: TAlignInsertBeforeEvent;
```

C++

```
__property TAlignInsertBeforeEvent OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore (see page 6137)

Align (see page 5798)

TAlignInsertBeforeEvent (see page 6293)

154.1.17.3.2 TGroupBox.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition: TAlignPositionEvent;
```

C++

```
__property TAlignPositionEvent OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition (see page 6138)

Align (see page 5798)

Anchors (see page 5802)

TAlignInfo (see page 6269)

TAlignPositionEvent (see page 6293)

154.1.17.3.3 TGroupBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
```

```
      pEdit->Parent = pPage;
      pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
      pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
    }
    PageControl1Change(Sender);
  }
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

154.1.17.3.4 TGroupBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

154.1.17.3.5 TGroupBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

154.1.17.3.6 TGroupBox.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop: TDockDropEvent;
```

C++

```
__property TDockDropEvent OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
```

```
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockOver (see page 6216)

154.1.17.3.7 TGroupBox.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver: TDockOverEvent;
```

C++

```
__property TDockOverEvent OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
```

```
      ShowMessage("GetSiteInfo CanDock went bad!");
    }
  }
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnGetSiteInfo (see page 6223)

TDragDockObject (see page 6009)

154.1.17.3.8 TGroupBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

154.1.17.3.9 TGroupBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}
```

```
/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
```

```
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.17.3.10 TGroupBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.17.3.11 TGroupBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

154.1.17.3.12 TGroupBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
```

```
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
```

```
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

154.1.17.3.13 TGroupBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK

button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

154.1.17.3.14 TGroupBox.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo: TGetSiteInfoEvent;
```

C++

```
__property TGetSiteInfoEvent OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
```

```
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

TDragDockObject (see page 6009)

154.1.17.3.15 TGroupBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.17.3.16 TGroupBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
```

```
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
```

```
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.17.3.17 TGroupBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

154.1.17.3.18 TGroupBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

154.1.17.3.19 TGroupBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
```

```
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
```

```
      end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.17.3.20 TGroupBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}
```

```
// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
```

```
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

154.1.17.3.21 TGroupBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

154.1.17.3.22 TGroupBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

154.1.17.3.23 TGroupBox.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock: TUnDockEvent;
```

C++

```
__property TUnDockEvent OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
```

```
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
```

```
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage("GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
```

```
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite (see page 6204)

OnDockDrop (see page 6214)

OnDockOver (see page 6216)

154.1.18 TLabel Class

TLabel is a nonwindowed control that displays text on a form.

Class Hierarchy

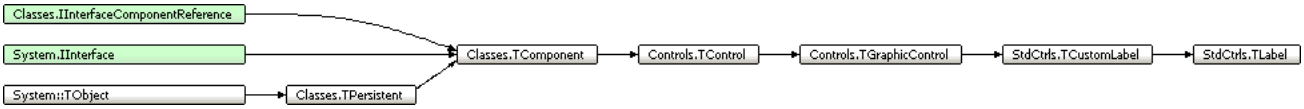

Pascal

```
TLabel = class(TCustomLabel);
```

C++

```
class TLabel : public TCustomLabel;
```

File

StdCtrls

Description

Use TLabel to add text that the user can't edit to a form. This text can be used to label another control, and can set focus to that control when the user types an accelerator key.

Because TLabel is not a descendant of TWinControl, it does not have its own window and can't receive direct input from the keyboard. To add an object to a form that can respond to keyboard input (other than setting focus to another object when an accelerator key is typed) in addition to displaying text, use TStaticText (see page 21887).

To add an object to a form that displays text that a user can scroll or edit, use TEdit (see page 21402).

See Also

TEdit (see page 21402)

TStaticText (see page 21887)

Labels

Display Controls

154.1.18.1 TLabel Members

The following tables list the members exposed by TLabel.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	AdjustBounds (see page 21296)	Resizes the label when the text changes.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDrawText (see page 21296)	Renders the text in the client area.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetLabelText (see page 21297)	Returns the value of the Caption (see page 21303) property.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	Loaded (see page 21297)	Adjusts the size of the label when it is first loaded. .
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 21298)	Responds to notifications when objects are about to be created or destroyed.

V	Paint (see page 6052)	Render control surface.
V	Paint (see page 21298)	Draws the image of the label.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetAutoSize (see page 21298)	Sets the AutoSize (see page 21299) property.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
R	Canvas (see page 6053)	Provides the drawing surface used by the graphic control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.

	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6051)	Creates an instance of TGraphicControl (see page 6044).
V	Create (see page 21295)	Creates and initializes an instance of TCustomLabel (see page 21288).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6052)	Destroys an instance of TGraphicControl (see page 6044).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EndDrag (see page 5757)	Stops a control from being dragged any further.

	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
	Canvas (see page 21300)	Provides the drawing surface used by the graphic control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Parent (see page 5832)	Indicates the parent of the control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnClick (see page 21579)	Occurs when the user clicks the control.
	OnContextPopup (see page 21581)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 21582)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 21582)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21584)	Occurs when the user drags an object over a control.
	OnEndDock (see page 21587)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21587)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 21588)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 21589)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 21592)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21592)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 21592)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 21594)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 21597)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21597)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 21553)	Determines how the control aligns within its container (parent control).
	Alignment (see page 21556)	Controls the horizontal placement of the text within the label.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 21557)	Specifies how the control is anchored to its parent.
	AutoSize (see page 21557)	Determines whether the size of the label automatically resizes to accommodate the text.
	BiDiMode (see page 21558)	Specifies the bi-directional mode for the control.
	Caption (see page 21559)	Specifies a text string that identifies the control to the user.
	Color (see page 21561)	Specifies the background color of the control.
	Constraints (see page 21562)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 21563)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21563)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21563)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EllipsisPosition (see page 21565)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	Enabled (see page 21565)	Controls whether the control responds to mouse, keyboard, and timer events.
	FocusControl (see page 21566)	Designates a windowed control associated with the label.
	Font (see page 21566)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Layout (see page 21567)	Specifies the vertical placement of the text within the label.

	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 21567)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 21568)	Determines where a control looks for its color information.
	ParentFont (see page 21568)	Determines where a control looks for its font information.
	ParentShowHint (see page 21569)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 21569)	Identifies the pop-up menu associated with the control.
	ShowAccelChar (see page 21574)	Determines how an ampersand in the label text is displayed.
	ShowHint (see page 21575)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Transparent (see page 21576)	Specifies whether controls that sit below the label on a form can be seen through the label.
	Visible (see page 21577)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	WordWrap (see page 21578)	Specifies whether the label text wraps when it is too long for the width of the label.

154.1.18.2 TLabel Properties

The properties of the TLabel class are listed here.

Published Properties

	Name	Description
	Align (see page 21553)	Determines how the control aligns within its container (parent control).
	Alignment (see page 21556)	Controls the horizontal placement of the text within the label.
	Anchors (see page 21557)	Specifies how the control is anchored to its parent.
	AutoSize (see page 21557)	Determines whether the size of the label automatically resizes to accommodate the text.
	BiDiMode (see page 21558)	Specifies the bi-directional mode for the control.
	Caption (see page 21559)	Specifies a text string that identifies the control to the user.
	Color (see page 21561)	Specifies the background color of the control.
	Constraints (see page 21562)	Specifies the size constraints for the control.
	DragCursor (see page 21563)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21563)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21563)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EllipsisPosition (see page 21565)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	Enabled (see page 21565)	Controls whether the control responds to mouse, keyboard, and timer events.
	FocusControl (see page 21566)	Designates a windowed control associated with the label.
	Font (see page 21566)	Controls the attributes of text written on or in the control.
	Layout (see page 21567)	Specifies the vertical placement of the text within the label.
	ParentBiDiMode (see page 21567)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 21568)	Determines where a control looks for its color information.
	ParentFont (see page 21568)	Determines where a control looks for its font information.
	ParentShowHint (see page 21569)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 21569)	Identifies the pop-up menu associated with the control.
	ShowAccelChar (see page 21574)	Determines how an ampersand in the label text is displayed.
	ShowHint (see page 21575)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Transparent (see page 21576)	Specifies whether controls that sit below the label on a form can be seen through the label.
	Visible (see page 21577)	Determines whether the component appears onscreen.

	WordWrap (see page 21578)	Specifies whether the label text wraps when it is too long for the width of the label.

154.1.18.2.1 TLabel.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;
```

```
  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
```

```
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
```

```
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

154.1.18.2.2 TLabel.Alignment Property

Controls the horizontal placement of the text within the label.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Set Alignment to specify how the text of the label is justified within the ClientRect of the label control.

The effect of the Alignment property is more obvious if the WordWrap property is true and the label includes more than one line of text.

See Also

ClientRect (see page 5810)

GetLabelText (see page 21297)

Layout (see page 21306)

WordWrap (see page 21308)

154.1.18.2.3 TLabel.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

154.1.18.2.4 TLabel.AutoSize Property

Determines whether the size of the label automatically resizes to accommodate the text.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;__property Boolean AutoSize;
```

Description

Use AutoSize to make the label adjust its size automatically so the client area accommodates the height and width of the text. When AutoSize is false, the label is fixed in size. When AutoSize is true, the size of the label readjusts whenever the text changes. The size of the label is also readjusts when the Font property changes.

When WordWrap is true, the width of the label is fixed. If AutoSize is also true, changes to the text cause the label to change in height. When AutoSize is true and WordWrap is false, the font determines the height of the label, and changes to the text cause the label to change in width.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label’s caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
```

```
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
```

See Also

BoundsRect (see page 5803)

GetLabelText (see page 21297)

Font (see page 5823)

WordWrap (see page 21308)

Layout (see page 21306)

154.1.18.2.5 TLabel.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

154.1.18.2.6 TLabel.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
```

```
      button->Parent = toolBar;
      button->Caption = captions->Strings[i];
      if (button->Caption == "|")
        button->Style = tbsSeparator;
      else
        button->Style = tbsButton;
    }
  }

  TToolBar *myToolBar;

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    myToolBar = new TToolBar(this);
    myToolBar->Parent = this;
    TStringList *captions = new TStringList();
    captions->Add("New");
    captions->Add("Save");
    captions->Add("|");
    captions->Add("Cut");
    captions->Add("Copy");
    captions->Add("Paste");
    AddButtons(myToolBar, captions);
    delete captions;
    myToolBar->ShowCaptions = True;
    myToolBar->Height = 40;
  }

  void __fastcall TForm1::FormDestroy(TObject *Sender)
  {
    for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
      delete myToolBar->Buttons[I];
    delete myToolBar;
  }
```

Delphi Examples:

```
  {
  The following code keeps the size of the label control
  constant, even though the length of the label's caption
  changes. As a result, the caption of the label is probably
  too long to display in the label when the user clicks the
  button:
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Label1.AutoSize := False;
    Label1.Caption :=
      'This string is too long as the caption of this label';
  end;
  {
  This example requires a TPageControl populated by several
  TabSheets, a TEdit and a TButton. When the button is
  clicked, the caption of the active tab changes to that of
  the TEdit.  These lines of code could also be written:
  PageControl1.ActivePage.Caption := Edit1.Text;
  This would obviate the need for a TTabSheet variable since
  PageControl1.->ActivePage is already of type TTabSheet.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    sheet: TTabSheet;
  begin
    sheet := PageControl1.ActivePage;
    sheet.Caption := Edit1.Text;
  end;
  {
```

```
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

154.1.18.2.7 TLabel.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
```

```
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

154.1.18.2.8 TLabel.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

154.1.18.2.9 TLabel.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

154.1.18.2.10 TLabel.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

154.1.18.2.11 TLabel.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

154.1.18.2.12 TLabel.EllipsisPosition Property

Specifies how ellipsis is placed in text not fitting in allocated rectangle.

Pascal

```
property EllipsisPosition: TEllipsisPosition;
```

C++

```
__property TEllipsisPosition EllipsisPosition;
```

Description

Set EllipsisPosition to determine how an ellipsis is placed in text that does not fit in its allocated rectangle. This is a TEllipsisPosition constant and its value corresponds to constants specifying ellipsis placement in the **dwDTFormat** parameter of the Windows **DrawText** and **DrawTextEx** functions.

See Also

WordWrap (see page 21308)

TypeName

154.1.18.2.13 TLabel.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

154.1.18.2.14 TLabel.FocusControl Property

Designates a windowed control associated with the label.

Pascal

```
property FocusControl: TWinControl;
```

C++

```
__property TWinControl * FocusControl;
```

Description

Set FocusControl to the windowed control that should receive focus when the user presses the accelerator key specified by the label. Specify an accelerator key by preceding a character in the label text with an ampersand (&) and setting the ShowAccelChar property to true.

See Also

ShowAccelChar (see page 21306)

GetLabelText (see page 21297)

154.1.18.2.15 TLabel.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
```

```
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

154.1.18.2.16 TLabel.Layout Property

Specifies the vertical placement of the text within the label.

Pascal

```
property Layout: TTextLayout;
```

C++

```
__property TTextLayout Layout;
```

Description

Set Layout to specify how the text of the label is placed within the ClientRect of the label control. Layout is the vertical analog to the Alignment property.

See Also

Alignment (see page 21299)

ClientRect (see page 5810)

154.1.18.2.17 TLabel.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

154.1.18.2.18 TLabel.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

154.1.18.2.19 TLabel.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

154.1.18.2.20 TLabel.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

154.1.18.2.21 TLabel.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
```

```
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
```

```
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
```

```
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
```

```
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

154.1.18.2.22 TLabel.ShowAccelChar Property

Determines how an ampersand in the label text is displayed.

Pascal

```
property ShowAccelChar: Boolean;
```

C++

```
__property Boolean ShowAccelChar;
```

Description

Set ShowAccelChar to true to allow the label to display an underlined accelerator key value. When ShowAccelChar is true, any character preceded by an ampersand (&) appears underlined. If the FocusControl property is set, the windowed control specified by the FocusControl property receives input focus when the user types that underlined character. To display an ampersand when ShowAccelChar is true, use two ampersands (&&) to stand for the single ampersand that is displayed.

Set ShowAccelChar to false to display the label text with all ampersands appearing as ampersands. When ShowAccelChar is false, the value of the FocusControl property is not used.

C++ Examples:

```
/*
This example uses two labels on a form. The first label has
a caption with an accelerator character in it. The second
label also includes an ampersand, but it does not appear as
an accelerator character.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Label1->ShowAccelChar = true;
  Label1->Caption = "An &Underlined character appears here";
  Label2->ShowAccelChar = false;
```

```
  Label2->Caption = "An ampersand (&) appears here";
}
```

Delphi Examples:

```
{
This example uses two labels on a form. The first label has
a caption with an accelerator character in it. The second
label also includes an ampersand, but it does not appear as
an accelerator character.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Label1.ShowAccelChar := True;
  Label1.Caption := 'An &Underlined character appears here';
  Label2.ShowAccelChar := False;
  Label2.Caption := 'An ampersand (&) appears here';
end;
```

See Also

FocusControl (see page 21306)

GetLabelText (see page 21297)

154.1.18.2.23 TLabel.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
```

```
    char index[3];
    for(int i = 1; i <= 10; i++)
    {
      strcpy(string, "Item");
      itoa(i, index, 10);
      strcat(string, index);
      ListBox1->Items->Add(string);
    }
    ListBox1->Hint = "Select an item";
    ListBox1->ShowHint = true;
    Application->ShowHint = true;
  }
  /*
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  */
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

154.1.18.2.24 TLabel.Transparent Property

Specifies whether controls that sit below the label on a form can be seen through the label.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Set Transparent to true to prevent the label from obscuring other controls on the form. For example, if the label is used to add text to a graphic, set Transparent to true so that the label does not stand out as a separate object.

Writing text so that the background is transparent is slower than writing text when Transparent is false. If the label is not obscuring a complicated image, performance can be improved by setting the background color of the label to match the object beneath it and setting Transparent to false.

See Also

Color (see page 5812)

ParentColor (see page 5834)

154.1.18.2.25 TLabel.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
```

```
      pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
      pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
    }
    PageControl1Change(Sender);
  }
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

154.1.18.2.26 TLabel.WordWrap Property

Specifies whether the label text wraps when it is too long for the width of the label.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the label to display multiple line of text. When WordWrap is true, text that is too wide for the label control wraps at the right margin and continues in additional lines.

Set WordWrap to false to limit the label to a single line. When WordWrap is false, text that is too wide for the label appears truncated.

See Also

ClientWidth (see page 5811)

GetLabelText (see page 21297)

154.1.18.3 TLabel Events

The events of the TLabel class are listed here.

Published Events

	Name	Description
	OnClick (see page 21579)	Occurs when the user clicks the control.
	OnContextPopup (see page 21581)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 21582)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 21582)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21584)	Occurs when the user drags an object over a control.
	OnEndDock (see page 21587)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21587)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 21588)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 21589)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 21592)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21592)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 21592)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 21594)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 21597)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21597)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

154.1.18.3.1 TLabel.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an

OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
```

```
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

154.1.18.3.2 TLabel.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the

default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

154.1.18.3.3 TLabel.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

154.1.18.3.4 TLabel.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
```

```
  The user can select a label and drag it to a list box and
  drop it. When the label is dropped, the items in the list
  box assume the color and font of the dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

154.1.18.3.5 TLabel.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.18.3.6 TLabel.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.18.3.7 TLabel.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
```

```
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

154.1.18.3.8 TLabel.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
```

```
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.18.3.9 TLabel.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
```

```
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.18.3.10 TLabel.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

154.1.18.3.11 TLabel.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

154.1.18.3.12 TLabel.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
```

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.18.3.13 TLabel.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

154.1.18.3.14 TLabel.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

154.1.18.3.15 TLabel.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

154.1.19 TListBox Class

TListBox displays a collection of items in a scrollable list.

Class Hierarchy

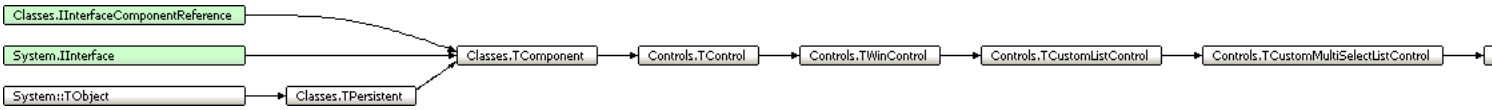

Pascal

```
TListBox = class(TCustomListBox);
```

C++

```
class TListBox : public TCustomListBox;
```

File

StdCtrls

Description

Use TListBox to display a scrollable list of items that users can select, add, or delete. TListBox is a wrapper for the Windows listbox control. For specialized list boxes, use other descendant classes of TCustomListBox (see page 21308) or derive your own class from TCustomListBox (see page 21308).

TListBox implements the generic behavior introduced in TCustomListBox (see page 21308). TListBox publishes many of the properties inherited from TCustomListBox (see page 21308), but does not introduce any new behavior.

See Also

List Controls

List Boxes and Check-list Boxes

154.1.19.1 TListBox Members

The following tables list the members exposed by TListBox.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FMoving (see page 21321)	This is FMoving, a member of class TCustomListBox.
	FMultiSelect (see page 5956)	This is FMultiSelect, a member of class TCustomMultiSelectListControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.

	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 21324)	Initializes the window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 21325)	Creates a Windows control corresponding to the list box component.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DeleteString (see page 21326)	Deletes a string from the list box.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21326)	Saves the list box items in a string list before freeing the underlying Windows control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFindData (see page 21326)	Returns the index of a string in the list box.
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoGetData (see page 21327)	Returns a string in the list box given its index.
	DoGetDataObject (see page 21327)	Returns the object associated with an item in a virtual list box.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.

	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
V	DragCanceled (see page 21328)	Sends a mouse message canceling the drag-and-drop operation.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawItem (see page 21328)	Generates an OnDrawItem (see page 21358) event.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
A	GetCount (see page 5942)	Returns the number of items in the list.
V	GetCount (see page 21329)	Returns the value of the Count (see page 21346) property.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetItemData (see page 21329)	Returns a 32-bit value associated with the specified item in a list box.
A	GetItemIndex (see page 5942)	Returns the value of the ItemIndex (see page 5944) property.
V	GetItemIndex (see page 21329)	Returns the value of the ItemIndex property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
A	GetSelCount (see page 5956)	Returns the value of the SelCount (see page 5957) property.
V	GetSelCount (see page 21330)	Returns the number of selected items.
	GetSelected (see page 21330)	Indicates whether a particular item is selected.

	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InternalGetItemData (see page 21330)	Retrieves the data associated with an item in the list box.
	InternalSetItemData (see page 21330)	Associates an integer with a specified item in the list box.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
V	KeyPress (see page 21334)	Responds when the user presses a key.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	LoadRecreateItems (see page 21334)	This is LoadRecreateItems, a member of class TCustomListBox.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
V	MeasureItem (see page 21335)	Generates an OnMeasureItem (see page 21360) event.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetContent (see page 21335)	Removes all of the items from a list box.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
V	SaveRecreateItems (see page 21336)	This is SaveRecreateItems, a member of class TCustomListBox.

	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetItemData (see page 21336)	Sets the 32-bit value associated with the specified item in a list box.
A	SetItemIndex (see page 5943)	Sets the value of the ItemIndex (see page 5944) property.
V	SetItemIndex (see page 21336)	Gives focus to the item with a specified index.
A	SetMultiSelect (see page 5956)	Sets the value of the MultiSelect (see page 5957) property.
V	SetMultiSelect (see page 21337)	Sets the MultiSelect property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.

	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WndProc (see page 21337)	Handles Windows messages.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddItem (see page 5940)	Adds an item to the list control.
	AddItem (see page 21322)	Adds an item to the list box
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 5940)	Deletes all of the items from the list control.
	Clear (see page 21322)	Deletes all items from the list box.
	ClearSelection (see page 5940)	Removes the selection, leaving all items unselected.
	ClearSelection (see page 21323)	Deselects all selected items in the list box, if possible.

	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
A	CopySelection (see page 5941)	Copies the selected item or items to another TCustomListControl (see page 5928) descendant.
V	CopySelection (see page 21323)	Copies the selected items to another list-type control
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 21320)	Instantiates a list box control.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	DeleteSelected (see page 5941)	Deletes all the selected items.
V	DeleteSelected (see page 21325)	Deletes the selected items from the list box
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 21320)	Destroys the instance of a list box control.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.

	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ItemAtPos (see page 21331)	Returns the index of the list box item indicated by the coordinates of a point on the control.
	ItemRect (see page 21333)	Returns the rectangle that surrounds the item specified in the Item parameter.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	MoveSelection (see page 5942)	Moves all selected items to another list control.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.

	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
A	SelectAll (see page 5943)	Selects all items or all text in the selected item.
V	SelectAll (see page 21336)	Selects the entire block of text in the list box.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 21343)	Provides a drawing surface when implementing a handler for the OnDrawItem (see page 21358) event of an owner-draw list box.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	Count (see page 21346)	Indicates the number of items in the list box.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.

R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemIndex (see page 5944)	Specifies the index of the selected item.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	SelCount (see page 5957)	Indicates the number of selected items.
	Selected (see page 21351)	Indicates whether a particular item is selected.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TopIndex (see page 21356)	Specifies the index number of the item that appears at the top of the list box.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 21645)	Occurs when the user clicks the control.
	OnContextPopup (see page 21647)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnData (see page 21647)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 21648)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 21648)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDblClick (see page 21649)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 21649)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21651)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 21654)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnEndDock (see page 21656)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21656)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 21657)	Occurs when a control receives the input focus.
	OnExit (see page 21660)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 21661)	Occurs when a user presses any key while the control has focus.

	OnKeyPress (see page 21664)	Occurs when a key is pressed.
	OnKeyUp (see page 21665)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 21667)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.
	OnMouseActivate (see page 21668)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 21669)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 21671)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21672)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 21672)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 21674)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 21676)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21677)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 21611)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 21615)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 21615)	Determines whether the user can give focus to items by typing in the list.
	AutoCompleteDelay (see page 21615)	Specifies the delay between a key press and an attempt to autocomplete the field.
	BevelEdges (see page 21616)	Specifies which edges of the control are beveled.
	BevelInner (see page 21616)	Specifies the cut of the inner bevel.
	BevelKind (see page 21617)	Specifies the control's bevel style.
	BevelOuter (see page 21617)	Specifies the cut of the outer bevel.
	BevelWidth (see page 21617)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 21618)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 21618)	Determines whether the list box has a border.
	Color (see page 21619)	Specifies the background color of the control.
	Columns (see page 21620)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	Constraints (see page 21620)	Specifies the size constraints for the control.
	Ctl3D (see page 21621)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 21621)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21621)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21622)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 21623)	Controls whether the control responds to mouse, keyboard, and timer events.
	ExtendedSelect (see page 21624)	Makes list item multi-select operate with Shift and CTRL keys.
	Font (see page 21624)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 21625)	Determines the behavior of the input method editor (IME).
	ImeName (see page 21626)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 21627)	Determines whether the list box displays the partial items.

	ItemHeight (see page 21627)	Specifies the height in pixels of an item in an owner-draw list box.
	Items (see page 21628)	Contains the strings that appear in the list box.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	MultiSelect (see page 21630)	Specifies whether the user can select more than one item.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 21631)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 21631)	Determines where a control looks for its color information.
	ParentCtl3D (see page 21631)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 21632)	Determines where a control looks for its font information.
	ParentShowHint (see page 21632)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 21633)	Identifies the pop-up menu associated with the control.
	ScrollWidth (see page 21637)	Specifies the width, in pixels, by which the list box can scroll horizontally.
	ShowHint (see page 21638)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Sorted (see page 21639)	Specifies whether the items in a list box are arranged alphabetically.
	Style (see page 21641)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabOrder (see page 21641)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21642)	Determines if the user can tab to a control.
	TabWidth (see page 21642)	Specifies the size of the tabs in the list box.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 21643)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

154.1.19.2 TListBox Properties

The properties of the TListBox class are listed here.

Published Properties

	Name	Description
	Align (see page 21611)	Determines how the control aligns within its container (parent control).
	Anchors (see page 21615)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 21615)	Determines whether the user can give focus to items by typing in the list.
	AutoCompleteDelay (see page 21615)	Specifies the delay between a key press and an attempt to autocomplete the field.
	BevelEdges (see page 21616)	Specifies which edges of the control are beveled.
	BevelInner (see page 21616)	Specifies the cut of the inner bevel.
	BevelKind (see page 21617)	Specifies the control's bevel style.
	BevelOuter (see page 21617)	Specifies the cut of the outer bevel.
	BevelWidth (see page 21617)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 21618)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 21618)	Determines whether the list box has a border.
	Color (see page 21619)	Specifies the background color of the control.
	Columns (see page 21620)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	Constraints (see page 21620)	Specifies the size constraints for the control.
	Ctl3D (see page 21621)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 21621)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21621)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21622)	Determines how the control initiates drag-and-drop or drag-and-dock operations.

	Enabled (see page 21623)	Controls whether the control responds to mouse, keyboard, and timer events.
	ExtendedSelect (see page 21624)	Makes list item multi-select operate with Shift and CTRL keys.
	Font (see page 21624)	Controls the attributes of text written on or in the control.
	ImeMode (see page 21625)	Determines the behavior of the input method editor (IME).
	ImeName (see page 21626)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 21627)	Determines whether the list box displays the partial items.
	ItemHeight (see page 21627)	Specifies the height in pixels of an item in an owner-draw list box.
	Items (see page 21628)	Contains the strings that appear in the list box.
	MultiSelect (see page 21630)	Specifies whether the user can select more than one item.
	ParentBiDiMode (see page 21631)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 21631)	Determines where a control looks for its color information.
	ParentCtl3D (see page 21631)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 21632)	Determines where a control looks for its font information.
	ParentShowHint (see page 21632)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 21633)	Identifies the pop-up menu associated with the control.
	ScrollWidth (see page 21637)	Specifies the width, in pixels, by which the list box can scroll horizontally.
	ShowHint (see page 21638)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Sorted (see page 21639)	Specifies whether the items in a list box are arranged alphabetically.
	Style (see page 21641)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabOrder (see page 21641)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21642)	Determines if the user can tab to a control.
	TabWidth (see page 21642)	Specifies the size of the tabs in the list box.
	Visible (see page 21643)	Determines whether the component appears onscreen.

154.1.19.2.1 TListBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of

the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
```

```
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
```

```
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

154.1.19.2.2 TListBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

154.1.19.2.3 TListBox.AutoComplete Property

Determines whether the user can give focus to items by typing in the list.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

Use AutoComplete to specify whether the list box automatically updates the ItemIndex property in response to user keystrokes. When AutoComplete is true, the item with focus changes to the first item that begins with the sequence of characters that the user typed. As the user types more characters, ItemIndex may change to a later item because the string to match becomes longer.

See Also

ItemIndex

154.1.19.2.4 TListBox.AutoCompleteDelay Property

Specifies the delay between a key press and an attempt to autocomplete the field.

Pascal

```
property AutoCompleteDelay: Cardinal;
```

C++

```
__property unsigned AutoCompleteDelay;
```

Description

Specifies the delay in milliseconds between a key press and an attempt to autocomplete the field. Autocomplete will position to matching list values as you type, sometimes saving you keystrokes.

AutoCompleteDelay is only in effect when the AutoComplete (see page 21615) is true.

See Also

AutoComplete (see page 21615)

154.1.19.2.5 TListBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

154.1.19.2.6 TListBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

154.1.19.2.7 TListBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

154.1.19.2.8 TListBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

154.1.19.2.9 TListBox.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth: TBevelWidth;
```

C++

```
__property TBevelWidth BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelOuter (see page 6194)

154.1.19.2.10 TListBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

154.1.19.2.11 TListBox.BorderStyle Property

Determines whether the list box has a border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to change the appearance of the outer boundary of the list box control. A list box can have a single border (bsSingle) or no visible border (bsNone).

See Also

Ctl3D (see page 6198)

154.1.19.2.12 TListBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

154.1.19.2.13 TListBox.Columns Property

Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.

Pascal

```
property Columns: Integer;
```

C++

```
__property int Columns;
```

Description

Use Columns to specify the number of columns, in a multi-column list box, that are visible without having to use the horizontal scroll bar.

Multi-column list boxes have a horizontal scroll bar that allows users to view multiple columns as they wrap. The default value for Columns is 0, meaning that the list box is not multi-column. That is, users can scroll only vertically and the list of items will not wrap.

For Columns values greater than 0, multiple columns accommodate the items as they wrap beyond the bottom of the list box. The Columns property specifies the number of columns that are visible without having to horizontally scroll the list box.

The width of each column depends upon both the Width property and the number of Columns.

See Also

ItemHeight (see page 21347)

Width (see page 5849)

154.1.19.2.14 TListBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

154.1.19.2.15 TListBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

154.1.19.2.16 TListBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

154.1.19.2.17 TListBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

154.1.19.2.18 TListBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
```

```
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

154.1.19.2.19 TListBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

154.1.19.2.20 TListBox.ExtendedSelect Property

Makes list item multi-select operate with Shift and CTRL keys.

Pascal

```
property ExtendedSelect: Boolean;
```

C++

```
__property Boolean ExtendedSelect;
```

Description

ExtendedSelect only has effect when the MultiSelect (see page 21630) is true.

When ExtendedSelect is true, the Shift key (in conjunction with the left mouse button, or arrow keys) is used to select a contiguous sequence of list items. And the Ctrl key is used (in conjunction with the left mouse buttton) to select non-contiguous list items.

When ExtendedSelect is false, Multi-Select operates very differently. The Shift and Ctrl keys are not recognised. List items are always selected individually in any sequence. Reselecting a selected list item will deselect it. Use the arrow and spacebar keys, or the left mouse button to make selections.

See Also

MultiSelect (see page 21630)

154.1.19.2.21 TListBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

154.1.19.2.22 TListBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language

characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

154.1.19.2.23 TListBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

154.1.19.2.24 TListBox.IntegralHeight Property

Determines whether the list box displays the partial items.

Pascal

```
property IntegralHeight: Boolean;
```

C++

```
__property Boolean IntegralHeight;
```

Description

IntegralHeight controls the height of the list box itself.

When IntegralHeight is true, and the Style property is lsOwnerDrawFixed, the list box height will always be a multiple of the ItemHeight property value. In this case, items will never be partially displayed.

If IntegralHeight is false, the list box height can be set to any value, but the last item may not be fully displayed.

If the Style property value of the list box is lbOwnerDrawVariable, setting the IntegralHeight property to true has no effect.

See Also

ItemHeight (see page 21347)

Items (see page 21348)

Style (see page 21354)

Using TDBListBox and TDBComboBox

154.1.19.2.25 TListBox.ItemHeight Property

Specifies the height in pixels of an item in an owner-draw list box.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Use ItemHeight to control the height of an item in a list box when the Style property is lbOwnerDrawFixed. The ItemHeight property can also affect the height of the list box itself. When the IntegralHeight property is true, the height of the list box is always an increment of the ItemHeight value.

If the Style property is lbStandard or lbOwnerDrawVariable, the value of ItemHeight is ignored.

See Also

DrawItem (see page 21328)

IntegralHeight (see page 21347)

Items (see page 21348)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

Style (see page 21354)

Using TDBListBox and TDBComboBox

154.1.19.2.26 TListBox.Items Property

Contains the strings that appear in the list box.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to add, insert, delete and move items. By default, the items in a list box are of type TStrings. Use this item type to access its methods or properties to manipulate the items in the list.

For example, the following code adds the text in the edit box to the list box as an item:

```
ListBox1.Items.Add(Edit1.Text); {Delphi }
ListBox1->Items->Add(Edit1->Text); // C++
```

C++ Examples:

```
/*
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Continue will continue listing
file sizes in a multiple selection that includes the deleted
file.  Break will stop.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
*/
#include <stdio.h>     // for FILE, fopen, fstat, fileno, fread and fclose
#include <sys\stat.h>  // for fstat and the stat struct

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < FileListBox1->Items->Count; i++)
  {
    if (FileListBox1->Selected[i])
    {
      if (!FileExists(FileListBox1->Items->Strings[i]))
      {
        MessageBeep(0);
        if (Application->MessageBox(
          AnsiString("File " + FileListBox1->Items->Strings[i] + "not found").c_str(),
          NULL,
          MB_OKCANCEL) == IDCANCEL)
          break;
        else
          continue;
      }
      FILE *F = fopen(FileListBox1->Items->Strings[i].c_str(),"r");
      struct stat statbuf;
      fstat(fileno(F), &statbuf);
      ListBox1->Items->Add(
          FileListBox1->Items->Strings[i] + ": " + IntToStr(statbuf.st_size));
      fclose(F);
    }
```

```
  }
}
/*
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
list of strings in the list box. When the application runs
and the button is clicked, the strings in the list box
become uppercase.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int I = 0; I < ListBox1->Items->Count; I++)
    ListBox1->Items->Strings[I] = UpperCase(ListBox1->Items->Strings[I]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");
}
```

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], 1);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode 1.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);

      AssignFile(F, FileListBox1.Items.Strings[i]);
      Reset(F, 1);
      ListBox1.Items.Add(
        FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
      CloseFile(F);
    end;
   finally
```

```
    { do something here }
    end;
  end;
end;
{
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
list of strings in the list box. When the application runs
and the button is clicked, the strings in the list box
become uppercase.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to ListBox1.Items.Count -1 do
    ListBox1.Items[I] := SysUtils.UpperCase(ListBox1.Items[I]);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  Listbox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;
```

See Also

Clear (see page 21322)

ItemIndex

OnDrawItem (see page 21358)

Selected (see page 21351)

TStrings (see page 3029)

154.1.19.2.27 TListBox.MultiSelect Property

Specifies whether the user can select more than one item.

Pascal

```
property MultiSelect: Boolean;
```

C++

```
__property Boolean MultiSelect;
```

Description

Use MultiSelect to specify whether the user can select more than one item. When MultiSelect is true, the user can select multiple items in the control, and the SelCount property indicates the number of selected items. When MultiSelect is false, the user can only select one item, which is indicated by the ItemIndex property.

Note: To implement the MultiSelect property in a TCustomMultiSelectListControl descendant, override the protected SetMultiSelect method.

See Also

SelCount (see page 5957)

ItemIndex (see page 5944)

154.1.19.2.28 TListBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

154.1.19.2.29 TListBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

154.1.19.2.30 TListBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

154.1.19.2.31 TListBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

154.1.19.2.32 TListBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

154.1.19.2.33 TListBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
```

```
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Cut1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      const int num_items = 4;
      for (int index = 0; index < num_items; ++index)
      {
          TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
          PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
          NewItem->Caption = "Menu Item " + IntToStr(index);
          NewItem->Tag = index;
          NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
          TNotifyEvent();
      }
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
          {
```

```
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
```

```
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
```

```
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

154.1.19.2.34 TListBox.ScrollWidth Property

Specifies the width, in pixels, by which the list box can scroll horizontally.

Pascal

```
property ScrollWidth: Integer;
```

C++

```
__property int ScrollWidth;
```

Description

Use ScrollWidth to get or set the logical width of the list box. When ScrollWidth is greater than the client width of the list box, the list box gets a horizontal scroll bar. When ScrollWidth is less than or equal to ClientWidth, the horizontal scroll bar disappears.

See Also

Columns (see page 21346)

ClientWidth (see page 5811)

154.1.19.2.35 TListBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
```

```
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

154.1.19.2.36 TListBox.Sorted Property

Specifies whether the items in a list box are arranged alphabetically.

Pascal

```
property Sorted: Boolean;
```

C++

```
__property Boolean Sorted;
```

Description

Use Sorted to sort the items by setting its value to true. If Sorted is false, the items are unsorted. When Sorted is true, items are automatically added or inserted into the list box in alphabetical order.

Note: Sorted has no effect if Listbox's style is lbVirtual or lbVirtualOwnerDraw

C++ Examples:

```
/*
This example uses an edit box, a list box, and two buttons
on a form. The buttons are named Add and Sort. When the user
clicks the Add button, the text in the edit box is added to
the list in the list box. Before the user clicks the sort
button, any new items are added to the end of the list.
After the user clicks the Sort button, the list in the list
box is sorted and remains sorted, even if additional strings
are added.  If the user clicks the sort button again, the
list box remains sorted, but any new strings are again added
to the end of the list.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");
}

void __fastcall TForm1::AddClick(TObject *Sender)
{
  ListBox1->Items->Add(Edit1->Text);
}

void __fastcall TForm1::SortClick(TObject *Sender)
{
  ListBox1->Sorted = True;
}
```

Delphi Examples:

```
{
This example uses an edit box, a list box, and two buttons
on a form. The buttons are named Add and Sort. When the user
clicks the Add button, the text in the edit box is added to
the list in the list box. Before the user clicks the sort
button, any new items are added to the end of the list.
After the user clicks the Sort button, the list in the list
box is sorted and remains sorted, even if additional strings
are added.  If the user clicks the sort button again, the
list box remains sorted, but any new strings are again added
to the end of the list.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;


procedure TForm1.AddClick(Sender: TObject);
begin
  ListBox1.Items.Add(Edit1.Text);
end;

procedure TForm1.SortClick(Sender: TObject);
begin
  ListBox1.Sorted := True;
end;
```

See Also

Items (see page 21348)

Style (see page 21354)

154.1.19.2.37 TListBox.Style Property

Determines whether the list box is standard or owner-draw and whether it is virtual.

Pascal

```
property Style: TListBoxStyle;
```

C++

```
__property TListBoxStyle Style;
```

Description

Use Style to specify whether the list box is a standard list box that displays text strings, or an owner-draw list box that can display graphical images. Owner-draw list boxes let you display items in some nonstandard way. In this case, you must write the code to paint items in the list box.

In addition, Style controls whether the list box is virtual, which means that items are supplied dynamically using event handlers. By default, Style is lbStandard, meaning that the list box is not virtual and it displays each item as a string.

See Also

DrawItem (see page 21328)

ItemHeight (see page 21347)

Items (see page 21348)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

OnMeasureItem (see page 21360)

OnData (see page 21357)

OnDataFind (see page 21357)

OnDataObject (see page 21358)

154.1.19.2.38 TListBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

154.1.19.2.39 TListBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

154.1.19.2.40 TListBox.TabWidth Property

Specifies the size of the tabs in the list box.

Pascal

```
property TabWidth: Integer;
```

C++

```
__property int TabWidth;
```

Description

Set TabWidth to control the number of dialog base units displayed for each tab character that appears in the strings of a list box.

If TabWidth is set to 0, which is the default setting, the tabs default to the width specified by Windows, which is two dialog base units.

154.1.19.2.41 TListBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;
```

```
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

154.1.19.3 TListBox Events

The events of the TListBox class are listed here.

Published Events

	Name	Description
	OnClick (see page 21645)	Occurs when the user clicks the control.
	OnContextPopup (see page 21647)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnData (see page 21647)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 21648)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 21648)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDblClick (see page 21649)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 21649)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21651)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 21654)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnEndDock (see page 21656)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21656)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 21657)	Occurs when a control receives the input focus.
	OnExit (see page 21660)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 21661)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 21664)	Occurs when a key is pressed.
	OnKeyUp (see page 21665)	Occurs when the user releases a key that has been pressed.

	OnMeasureItem (see page 21667)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.
	OnMouseActivate (see page 21668)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 21669)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 21671)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21672)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 21672)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 21674)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 21676)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21677)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

154.1.19.3.1 TListBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
```

```
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

154.1.19.3.2 TListBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

154.1.19.3.3 TListBox.OnData Event

Occurs when a virtual list box needs to supply the text of an item.

Pascal

```
property OnData: TLBGetDataEvent;
```

C++

```
__property TLBGetDataEvent OnData;
```

Description

You must supply an OnData event handler to dynamically supply the items in the list box when the Style property is lbVirtual or lbVirtualOwnerDraw.

See Also

OnDataFind (see page 21357)

OnDataObject (see page 21358)

Style (see page 21354)

Count (see page 21346)

154.1.19.3.4 TListBox.OnDataFind Event

Occurs when a virtual list box needs to identify the index of an item given its text.

Pascal

```
property OnDataFind: TLBFindDataEvent;
```

C++

```
__property TLBFindDataEvent OnDataFind;
```

Description

You must supply an OnDataFind event handler to dynamically provide the mapping from list item to index when the Style property is lbVirtual or lbVirtualOwnerDraw.

See Also

OnData (see page 21357)

OnDataObject (see page 21358)

Style (see page 21354)

Count (see page 21346)

154.1.19.3.5 TListBox.OnDataObject Event

Occurs when a virtual list box needs to supply the object associated with an item.

Pascal

```
property OnDataObject: TLBGetDataObjectEvent;
```

C++

```
__property TLBGetDataObjectEvent OnDataObject;
```

Description

Supply an OnDataObject event handler to dynamically provide objects associated with the items in a virtual list box.

See Also

OnData (see page 21357)

OnDataFind (see page 21357)

Style (see page 21354)

Count (see page 21346)

154.1.19.3.6 TListBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

154.1.19.3.7 TListBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

154.1.19.3.8 TListBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
```

```
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code comes from an application that contains a list box
  and three labels, each with a different font and color.  The
  DragMode property for each of the labels is dmAutomatic.
  The user can select a label and drag it to a list box and
  drop it. When the label is dropped, the items in the list
  box assume the color and font of the dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.19.3.9 TListBox.OnDrawItem Event

Occurs when an item in an owner-draw list box needs to be redisplayed.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

Use OnDrawItem to write a handler for drawing of the items in list boxes with the Style values lbOwnerDrawFixed, lbOwnerDrawVariable, or lbVirtualOwnerDraw. OnDrawItem occurs when the list box needs to display an item. OnDrawItem occurs only for owner-draw list boxes.

The size of the rectangle that contains the item is determined either by the ItemHeight property for fixed owner-draw list boxes or by the response to the OnMeasureItem event for variable owner-draw list boxes.

C++ Examples:

```
/*
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MyList = new TStringList;
  bitmap0 = new Graphics::TBitmap;
  bitmap1 = new Graphics::TBitmap;
  bitmap2 = new Graphics::TBitmap;
//  MyList->Add("Animal");
//  MyList->Add("Flowers");
//  MyList->Add("Butterfly");
//  ListBox1.Items->AddStrings(MyList);

  ImageList1->GetBitmap(0, bitmap0);
  ListBox1->Items->AddObject("Flowers", bitmap0);
  ImageList1->GetBitmap(1, bitmap1);
  ListBox1->Items->AddObject("Animal", bitmap1);
  ImageList1->GetBitmap(2, bitmap2);
  ListBox1->Items->AddObject("Butterfly", bitmap2);
}

void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
      TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBitmap; // temporary variable for item's bitmap
  int     Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *pCanvas = ((TListBox *)Control)->Canvas;
  pCanvas->FillRect(Rect); // clear the rectangle
  pBitmap = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
  if (pBitmap)
  {
    pCanvas->BrushCopy(Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
pBitmap, Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
```

```
    Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  pCanvas->TextOut(Rect.Left + Offset, Rect.Top, ((TListBox *)Control)->Items->Strings[Index]);
}
```

Delphi Examples:

```
{
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  MyList := TStringList.Create;
  bitmap0 := TBitmap.Create;
  bitmap1 := TBitmap.Create;
  bitmap2 := TBitmap.Create;
//  MyList.Add('Animal');
//  MyList.Add('Flowers');
//  MyList.Add('Butterfly');
//  ListBox1.Items.AddStrings(MyList);

  ImageList1.GetBitmap(0, bitmap0);
  ListBox1.Items.AddObject('Flowers', bitmap0);
  ImageList1.GetBitmap(1, bitmap1);
  ListBox1.Items.AddObject('Animal', bitmap1);
  ImageList1.GetBitmap(2, bitmap2);
  ListBox1.Items.AddObject('Butterfly', bitmap2);
end;

procedure TForm1.ListBox1DrawItem(Control: TWinControl; Index: Integer; Rect:TRect; State:
TOwnerDrawState);
var
  Bitmap: TBitmap;      { temporary variable for the item's bitmap }
  Offset: Integer;      { text offset width }
begin
  Bitmap := TBitmap.Create;
  with (Control as TListBox).Canvas do  { draw on control canvas, not on the form }
  begin
    FillRect(Rect);       { clear the rectangle }
    Offset := 2;          { provide default offset }
    Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
    if Bitmap <> nil then
    begin
      BrushCopy(
        Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
        Bitmap,
        Bounds(0, 0, Bitmap.Width, Bitmap.Height),
        clRed);  {render bitmap}
      Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
    end;
    TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
text }
  end;
end;
```

See Also

DrawItem (see page 21328)

ItemHeight (see page 21347)

MeasureItem (see page 21335)

OnMeasureItem (see page 21360)

Style (see page 21354)

154.1.19.3.10 TListBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.19.3.11 TListBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
```

```
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

154.1.19.3.12 TListBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
```

```
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

154.1.19.3.13 TListBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

154.1.19.3.14 TListBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the

OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
```

```
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

154.1.19.3.15 TListBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
```

```
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

154.1.19.3.16 TListBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
```

```
      else
        ComboBox1->Text = "KeyDown";
      Form1->Refresh();
      Memo1->Lines->Add(ComboBox1->Text);
    }
  }

  void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
  {
    if (Key == VK_F1)
    {
      Form1->Color = FormColor;
      if (ComboBox1->Text == "KeyUp")
        ComboBox1->Text = "KeyUp2";
      else
        ComboBox1->Text = "KeyUp";
      Form1->Refresh();
      Memo1->Lines->Add(ComboBox1->Text);
    }
  }
```

Delphi Examples:

```
  {
  This example changes a form’s color to aqua when the user
  presses the F1 key, even when a control on the form has the
  focus. When the user releases the key, the form returns to
  its original color.  Set the KeyPreview on the form to True.
  Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
  event to FormKeyUp.  Set the KeyPreview on the form to True.
  }
  var
    FormColor: TColor;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    KeyPreview := True;
    FormColor := Form1.Color;
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

154.1.19.3.17 TListBox.OnMeasureItem Event

Occurs when the application needs to redisplay an item in a variable height owner-draw list box.

Pascal

```
property OnMeasureItem: TMeasureItemEvent;
```

C++

```
__property TMeasureItemEvent OnMeasureItem;
```

Description

Use OnMeasureItem to write a handler to measure items to be drawn in a list box with a Style property value of lbOwnerDrawVariable.

OnMeasureItem is of type TMeasureItemEvent which contains three parameters describing the item to measure:

Control, specifying the control containing the item.

Index, specifying the index of the item in the control.

Height, specifying the height of the item.

The Index parameter identifies the position of the item in the list box.

The Height parameter should specify the height in pixels that the given item will occupy in the control. The Height parameter is passed by reference (a var parameter), which initially contains the default height of the item or the height of the item text in the control's font. The handler can set Height to a value appropriate to the contents of the item, such as the height of a graphical image to be displayed within the item.

After the OnMeasureItem event occurs, the OnDrawItem event occurs, rendering the item with the measured size.

C++ Examples:

```
/*
Here is a typical handler for an OnMeasureItem event. The
example assumes that a variable owner-draw list box already
has bitmaps associated with each of its strings. It sets the
height of the item to the height of the associated bitmap if
that height is greater than the default height
(ListBox1.ItemHeight).  Note that TWinControl can also be
used in place of TWidgetControl on Windows.
*/
void __fastcall TForm1::ListBox1MeasureItem(TWinControl *Control, int Index,
      int &Height)
{
  Graphics::TBitmap *bitmap = (Graphics::TBitmap *)((TListBox
*)Control)->Items->Objects[Index];
  if (bitmap && bitmap->Height > Height)
    Height = bitmap->Height;
}

void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
      TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBitmap; // temporary variable for item’s bitmap
  int     Offset = 2;   // default text offset width
```

```
    // note that we draw on the listbox's canvas, not on the form
    TCanvas *pCanvas = ((TListBox *)Control)->Canvas;
    pCanvas->FillRect(Rect); // clear the rectangle
    pBitmap = (Graphics::TBitmap *)((TListBox *)Control)->Items->Objects[Index];
    if (pBitmap)
    {
      pCanvas->BrushCopy(Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
  pBitmap, Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
      Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
    }
    // display the text
    pCanvas->TextOut(Rect.Left + Offset, Rect.Top, ((TListBox *)Control)->Items->Strings[Index]);
  }

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    Graphics::TBitmap *bitmap0 = new Graphics::TBitmap;
    Graphics::TBitmap *bitmap1 = new Graphics::TBitmap;
    Graphics::TBitmap *bitmap2 = new Graphics::TBitmap;
    bitmap0->LoadFromFile("../littleB_64.bmp");
  //  ImageList1->GetBitmap(0, bitmap0);
    ListBox1->Items->AddObject("Butterfly", bitmap0);
    ImageList1->GetBitmap(3, bitmap1);
    ListBox1->Items->AddObject("Animal", bitmap1); // no bitmap, see if it leaves this height at
  64.
    ImageList1->GetBitmap(4, bitmap2);
    ListBox1->Items->AddObject("Sunflower", bitmap2); // no bitmap, see if it leaves this height
  at 64.

  }
```

See Also

DrawItem (see page 21328)

ItemHeight (see page 21347)

MeasureItem (see page 21335)

OnDrawItem (see page 21358)

Style (see page 21354)

154.1.19.3.18 TListBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is

over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.19.3.19 TListBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
```

```
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
```

```
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.19.3.20 TListBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

154.1.19.3.21 TListBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

154.1.19.3.22 TListBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
```

```
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.19.3.23 TListBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
```

```
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
```

```
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

154.1.19.3.24 TListBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
  property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

154.1.19.3.25 TListBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

154.1.20 TMemo Class

TMemo is a wrapper for a Windows multiline edit control.

Class Hierarchy

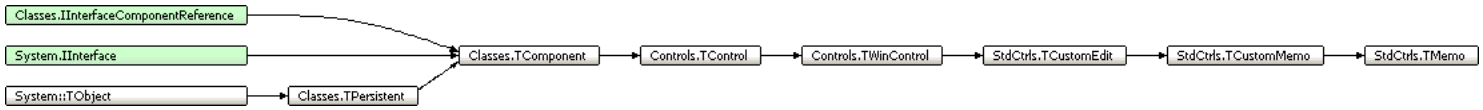

Pascal

```
TMemo = class(TCustomMemo);
```

C++

```
class TMemo : public TCustomMemo;
```

File

StdCtrls

Description

Use TMemo to put a standard Windows multiline edit control on a form. Multiline edit boxes allow the user to enter more than one line of text. They are appropriate for representing lengthy information.

TMemo implements the generic behavior introduced in TCustomMemo (see page 21362). TMemo publishes many of the properties inherited from TCustomMemo (see page 21362), but does not introduce any new behavior.

Note: Under Win 9x, there is a limit of 64k of text for this control.

See Also

TDBMemo (see page 9149)

TEdit (see page 21402)

TRichEdit (see page 4685)

Displaying and Editing Text in a Memo Control

Adding Scroll Bars at Runtime

Setting Text Alignment

Memo and Rich Edit Controls

Edit Controls

154.1.20.1 TMemo Members

The following tables list the members exposed by TMemo.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.

	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21245)	Generates an OnChange (see page 21275) event.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateParams (see page 21248)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateParams (see page 21375)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWindowHandle (see page 21248)	Creates a Windows control to represent the control.
	CreateWindowHandle (see page 21376)	Creates a window based on the window-creation parameter record.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 21248)	Creates a Windows control corresponding to the edit control.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DestroyWnd (see page 21251)	Destroys the Windows control that corresponds to the edit control.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.

	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoSetMaxLength (see page 21251)	Sets the MaxLength (see page 21263) property.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetCaretPos (see page 21376)	Returns the value of the CaretPos (see page 21380) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSelLength (see page 21252)	Returns the SelLength (see page 21265) property.
	GetSelStart (see page 21252)	Returns the SelStart (see page 21269) property.
	GetSelText (see page 21252)	Returns the SelText (see page 21272) property.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event

	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyPress (see page 21377)	Ensures that return characters are not entered into the text when WantReturns (see page 21384) is false.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	Loaded (see page 21377)	Initializes the memo control when it is first loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.

	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAlignment (see page 21377)	Sets the Alignment (see page 21379) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetAutoSize (see page 21257)	Sets the AutoSize (see page 21260) property.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
V	SetCaretPos (see page 21378)	Sets the CaretPos (see page 21380) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetLines (see page 21378)	Sets the Lines (see page 21380) property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetScrollBars (see page 21378)	Sets the ScrollBars (see page 21381) property.
V	SetSelLength (see page 21257)	Sets the SelLength (see page 21265) property.
V	SetSelStart (see page 21257)	Sets the SelStart (see page 21269) property.
	SetWordWrap (see page 21379)	Sets the WordWrap (see page 21384) property.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.

	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSelect (see page 21260)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	AutoSize (see page 21260)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	CharCase (see page 21262)	Determines the case of the text within the edit control.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	PasswordChar (see page 21265)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 21245)	Deletes all text from the edit control.
	ClearSelection (see page 21245)	Removes the selected text from the edit control.

	ClearUndo (see page 21246)	Clears the undo buffer so that no changes to the text can be backed out.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	CopyToClipboard (see page 21246)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 5731)	Creates an instance of TControl (see page 5724).
V	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
V	Create (see page 21244)	Creates an instance of TCustomEdit (see page 21232).
V	Create (see page 21374)	Creates and initializes an instance of TCustomMemo (see page 21362).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	CutToClipboard (see page 21249)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 21250)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 21374)	Destroys an instance of TCustomMemo (see page 21362).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
V	GetControlsAlignment (see page 21376)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
V	GetSelTextBuf (see page 21253)	Copies the selected text into a buffer and returns the number of characters copied.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	PasteFromClipboard (see page 21254)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectAll (see page 21255)	Selects all text in the edit control.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSelTextBuf (see page 21258)	Replaces the selected text with a null-terminated string.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Undo (see page 21259)	Backs out all changes in the undo buffer.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	CanUndo (see page 21262)	Indicates whether the edit control contains changes that can be backed out.
	CaretPos (see page 21380)	Indicates the position of the caret in the client area of the memo.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.

	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Modified (see page 21263)	Indicates whether the user edited the text of the edit control.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	SelLength (see page 21265)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 21269)	Specifies the position of the first selected character in the text.
	SelText (see page 21272)	Specifies the selected portion of the edit control's text.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	Text (see page 21273)	Contains a text string associated with the control.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 21724)	Occurs when the text for the edit control may have changed.
	OnClick (see page 21725)	Occurs when the user clicks the control.
	OnContextPopup (see page 21727)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 21727)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 21728)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21730)	Occurs when the user drags an object over a control.
	OnEndDock (see page 21732)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21733)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 21734)	Occurs when a control receives the input focus.

	OnExit (see page 21736)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 21738)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 21741)	Occurs when a key is pressed.
	OnKeyUp (see page 21742)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 21743)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 21744)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 21747)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21747)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 21747)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 21749)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 21752)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21752)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 21691)	Determines how the control aligns within its container (parent control).
	Alignment (see page 21695)	Determines how the text is aligned within the memo control.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 21695)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 21696)	Specifies which edges of the control are beveled.
	BevelInner (see page 21696)	Specifies the cut of the inner bevel.
	BevelKind (see page 21696)	Specifies the control's bevel style.
	BevelOuter (see page 21697)	Specifies the cut of the outer bevel.
	BiDiMode (see page 21697)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 21698)	Determines whether the edit control has a single line border around the client area.
	Color (see page 21698)	Specifies the background color of the control.
	Constraints (see page 21699)	Specifies the size constraints for the control.
	Ctl3D (see page 21699)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 21700)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21700)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21701)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 21702)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 21703)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	HideSelection (see page 21704)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	ImeMode (see page 21704)	Determines the behavior of the input method editor (IME).
	ImeName (see page 21705)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Lines (see page 21705)	Contains the individual lines of text in the memo control.
	Margins (see page 5831)	Specifies the margins for the control.

	MaxLength (see page 21706)	Specifies the maximum number of characters the user can enter into the edit control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OEMConvert (see page 21707)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBiDiMode (see page 21707)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 21708)	Determines where a control looks for its color information.
	ParentCtl3D (see page 21708)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 21709)	Determines where a control looks for its font information.
	ParentShowHint (see page 21709)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 21710)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 21714)	Determines whether the user can change the text of the edit control.
	ScrollBars (see page 21715)	Determines whether the memo control has scroll bars.
	ShowHint (see page 21717)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 21718)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21719)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 21719)	Determines whether the component appears onscreen.
	WantReturns (see page 21721)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 21721)	Determines whether the user can insert tab characters into the text.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	WordWrap (see page 21722)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

154.1.20.2 TMemo Properties

The properties of the TMemo class are listed here.

Published Properties

	Name	Description
	Align (see page 21691)	Determines how the control aligns within its container (parent control).
	Alignment (see page 21695)	Determines how the text is aligned within the memo control.
	Anchors (see page 21695)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 21696)	Specifies which edges of the control are beveled.
	BevelInner (see page 21696)	Specifies the cut of the inner bevel.
	BevelKind (see page 21696)	Specifies the control's bevel style.
	BevelOuter (see page 21697)	Specifies the cut of the outer bevel.
	BiDiMode (see page 21697)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 21698)	Determines whether the edit control has a single line border around the client area.
	Color (see page 21698)	Specifies the background color of the control.
	Constraints (see page 21699)	Specifies the size constraints for the control.
	Ctl3D (see page 21699)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 21700)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21700)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21701)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 21702)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 21703)	Controls the attributes of text written on or in the control.
	HideSelection (see page 21704)	Determines whether the visual indication of the selected text remains when focus shifts to another control.

	ImeMode (see page 21704)	Determines the behavior of the input method editor (IME).
	ImeName (see page 21705)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Lines (see page 21705)	Contains the individual lines of text in the memo control.
	MaxLength (see page 21706)	Specifies the maximum number of characters the user can enter into the edit control.
	OEMConvert (see page 21707)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBiDiMode (see page 21707)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 21708)	Determines where a control looks for its color information.
	ParentCtl3D (see page 21708)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 21709)	Determines where a control looks for its font information.
	ParentShowHint (see page 21709)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 21710)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 21714)	Determines whether the user can change the text of the edit control.
	ScrollBars (see page 21715)	Determines whether the memo control has scroll bars.
	ShowHint (see page 21717)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 21718)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21719)	Determines if the user can tab to a control.
	Visible (see page 21719)	Determines whether the component appears onscreen.
	WantReturns (see page 21721)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 21721)	Determines whether the user can insert tab characters into the text.
	WordWrap (see page 21722)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

154.1.20.2.1 TMemo.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
```

```
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
```

```
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

154.1.20.2.2 TMemo.Alignment Property

Determines how the text is aligned within the memo control.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to change the way the text is formatted by the memo control. Alignment can take one of the following values:

Value	Meaning
taLeftJustify	Align text to the left side of the control
taCenter	Center text horizontally in the control.
taRightJustify	Align text to the right side of the control.

See Also

SetAlignment (see page 21377)

Text (see page 5842)

Displaying and Editing Text in a Memo Control

154.1.20.2.3 TMemo.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

154.1.20.2.4 TMemo.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

154.1.20.2.5 TMemo.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

154.1.20.2.6 TMemo.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

154.1.20.2.7 TMemo.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

154.1.20.2.8 TMemo.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

154.1.20.2.9 TMemo.BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the edit control stands out. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the edit control has a single-line border around the client area. If BorderStyle is bsNone, there will be no border.

154.1.20.2.10 TMemo.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
```

```
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

154.1.20.2.11 TMemo.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

154.1.20.2.12 TMemo.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

154.1.20.2.13 TMemo.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

154.1.20.2.14 TMemo.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

154.1.20.2.15 TMemo.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
```

```
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

154.1.20.2.16 TMemo.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

154.1.20.2.17 TMemo.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;
```

```
procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

154.1.20.2.18 TMemo.HideSelection Property

Determines whether the visual indication of the selected text remains when focus shifts to another control.

Pascal

```
property HideSelection: Boolean;
```

C++

```
__property Boolean HideSelection;
```

Description

Set HideSelection to false to provide visual feedback of the selected portion of the text even when the edit control does not have focus. Set HideSelection to true to show the selection only when the edit control has focus. HideSelection does not affect the actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms with many edit controls look too busy.

See Also

SelLength (see page 21265)

SelStart (see page 21269)

SelText (see page 21272)

SetSelTextBuf (see page 21258)

154.1.20.2.19 TMemo.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode: TImeMode;
```

C++

```
__property TImeMode ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the

control already has input focus, call the SetIme method.

See Also

ImeName (see page 6208)

ResetIme (see page 6174)

ResetImeComposition (see page 6175)

SetIme (see page 6181)

SetImeCompositionWindow (see page 6181)

154.1.20.2.20 TMemo.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName: TImeName;
```

C++

```
__property TImeName ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode (see page 6207)

ResetIme (see page 6174)

SetIme (see page 6181)

DefaultIme (see page 14216)

Imes (see page 14227)

154.1.20.2.21 TMemo.Lines Property

Contains the individual lines of text in the memo control.

Pascal

```
property Lines: TStrings;
```

C++

```
__property TStrings Lines;
```

Description

Use Lines to manipulate text in an memo control on a line-by-line basis. Lines is a TStrings object, so the TStrings methods may be used for Lines to perform manipulations such as counting the lines of text, adding new lines, deleting lines, or replacing lines with new text.

To work with all the text at once, use the Text property. To manipulate individual lines of text, the Lines property works better.

Note: Although Lines is implemented as a TStrings descendant, it does not implement the support for associating objects with the strings in the list.

C++ Examples:

```
/*
This example uses a button and a memo control on a form.
When the user clicks the button, the content of the specified
file is loaded into the memo, and the fifth line of the file
is written across the top of the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Memo1->Lines->LoadFromFile("../readme.txt");
  Caption = Memo1->Lines->Strings[4];
}
```

Delphi Examples:

```
{
This example uses a button and a memo control on a form.
When the user clicks the button, the content of the specified
file is loaded into the memo, and the fifth line of the file
is written across the top of the form.
}

procedure TForm1.Button1Click(Sender: TObject);
begin
  Memo1.Lines.LoadFromFile('readme.txt');
  Caption:= Memo1.Lines[4];
end;
```

See Also

TStrings (see page 3029)

SelText (see page 21272)

SetLines (see page 21378)

Text (see page 5842)

WordWrap (see page 21384)

154.1.20.2.22 TMemo.MaxLength Property

Specifies the maximum number of characters the user can enter into the edit control.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the edit control. A value of 0 indicates that there is no

application-defined limit on the length.

Use MaxLength to limit the length of the text in an edit control if that text will be copied into a fixed-length buffer.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

Note: The unit of MaxLength is characters (not bytes) if the application is using themes.

See Also

Text (see page 5842)

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

154.1.20.2.23 TMemo.OEMConvert Property

Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.

Pascal

```
property OEMConvert: Boolean;
```

C++

```
__property Boolean OEMConvert;
```

Description

To ensure that any characters typed in the edit control can be unambiguously converted to the OEM character set, set OEMConvert to true. This causes all characters to be converted from ANSI to OEM and then back to ANSI as they are typed. Set OEMConvert to false to avoid the overhead of this conversion when it does not matter whether the text can be unambiguously mapped to an OEM string.

OEMConvert is most useful for edit controls used to enter file names when the application does not use Unicode file names.

See Also

Text (see page 5842)

154.1.20.2.24 TMemo.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

154.1.20.2.25 TMemo.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

154.1.20.2.26 TMemo.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

154.1.20.2.27 TMemo.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

154.1.20.2.28 TMemo.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

154.1.20.2.29 TMemo.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
```

```
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
```

```
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
```

```
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

154.1.20.2.30 TMemo.ReadOnly Property

Determines whether the user can change the text of the edit control.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

Setting ReadOnly to true ensures that the text is not altered, while still allowing the user to select text. The selected text can then be manipulated by the application, or copied to the Clipboard.

See Also

CopyToClipboard (see page 21246)

GetSelTextBuf (see page 21253)

154.1.20.2.31 TMemo.ScrollBars Property

Determines whether the memo control has scroll bars.

Pascal

```
property ScrollBars: TScrollStyle;
```

C++

```
__property TScrollStyle ScrollBars;
```

Description

Use ScrollBars to give a multiline edit control horizontal or vertical scroll bars. ScrollBars can take one of the following values:

Value	Meaning
ssNone	The control has no scroll bars.
ssHorizontal	The control has a single scroll bar on the bottom edge.
ssVertical	The control has a single scroll bar on the right edge.
ssBoth	The control has a scroll bar on both the bottom and right edges.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
```

```
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
    ssHorizontal,
    ssNone,
    ssVertical);
begin
  RichEdit1.ScrollBars :=
    ScrollBarA[RadioGroup1.ItemIndex];
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with RadioGroup1, Items do
  begin
    Caption := 'Scroll Bars';
    Clear;
    Add('ssBoth');
    Add('ssHorizontal');
    Add('ssNone');
    Add('ssVertical');
  end;
  with CheckBox1 do
  begin
    Caption := 'HideScrollBars';
    Checked := True;
  end;
  Button1.Caption := 'Loaded';
  RichEdit1.Text := 'Just a little text.';
```

```
  RichEdit1.WordWrap := False;
end;
```

See Also

SetScrollBars (see page 21378)

WordWrap (see page 21384)

Displaying and Editing Text in a Memo Control

154.1.20.2.32 TMemo.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
```

```
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

154.1.20.2.33 TMemo.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the

control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

154.1.20.2.34 TMemo.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

154.1.20.2.35 TMemo.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
```

```
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

154.1.20.2.36 TMemo.WantReturns Property

Determines whether the user can insert return characters into the text.

Pascal

```
property WantReturns: Boolean;
```

C++

```
__property Boolean WantReturns;
```

Description

Set WantReturns to true to allow users to enter return characters into the text. Set WantReturns to false to allow the form to handle return characters instead.

For example, in a form with a default button (such as an OK button) and a memo control, if WantReturns is false, pressing Enter chooses the default button. If WantReturns is true, pressing Enter inserts a return character in the text.

Note: If WantReturns is false, users can still enter return characters into the text by pressing Ctrl+Enter.

See Also

Text (see page 5842)

WantTabs (see page 21384)

WordWrap (see page 21384)

154.1.20.2.37 TMemo.WantTabs Property

Determines whether the user can insert tab characters into the text.

Pascal

```
property WantTabs: Boolean;
```

C++

```
__property Boolean WantTabs;
```

Description

Set WantTabs to true to allow users to enter tab characters into the text. Set WantTabs to false if you want the tab character to select the next control on the form instead.

Note: If WantTabs is true, users can tab into the edit control, but they can't tab out.

See Also

Text (see page 5842)

WantReturns (see page 21384)

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

154.1.20.2.38 TMemo.WordWrap Property

Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to make the edit control wrap text at the right margin so it fits in the client area. The wrapping is cosmetic only. The text does not include any return characters that were not explicitly entered. Set WordWrap to false to have the edit control show a separate line only where return characters were explicitly entered into the text.

Note: There should be no use for a horizontal scroll bar if WordWrap is true.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
```

```
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
    ssHorizontal,
    ssNone,
    ssVertical);
begin
  RichEdit1.ScrollBars :=
    ScrollBarA[RadioGroup1.ItemIndex];
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with RadioGroup1, Items do
  begin
    Caption := 'Scroll Bars';
    Clear;
    Add('ssBoth');
    Add('ssHorizontal');
    Add('ssNone');
    Add('ssVertical');
  end;
  with CheckBox1 do
  begin
    Caption := 'HideScrollBars';
    Checked := True;
  end;
  Button1.Caption := 'Loaded';
  RichEdit1.Text := 'Just a little text.';
  RichEdit1.WordWrap := False;
```

```
end;
```

See Also

Lines (see page 21380)

ScrollBars (see page 21381)

Text (see page 5842)

WantReturns (see page 21384)

Displaying and Editing Text in a Memo Control

154.1.20.3 TMemo Events

The events of the TMemo class are listed here.

Published Events

	Name	Description
	OnChange (see page 21724)	Occurs when the text for the edit control may have changed.
	OnClick (see page 21725)	Occurs when the user clicks the control.
	OnContextPopup (see page 21727)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 21727)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 21728)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21730)	Occurs when the user drags an object over a control.
	OnEndDock (see page 21732)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21733)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 21734)	Occurs when a control receives the input focus.
	OnExit (see page 21736)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 21738)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 21741)	Occurs when a key is pressed.
	OnKeyUp (see page 21742)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 21743)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 21744)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 21747)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21747)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 21747)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 21749)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 21752)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21752)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

154.1.20.3.1 TMemo.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change (see page 21245)

Text (see page 5842)

154.1.20.3.2 TMemo.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
```

```
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

154.1.20.3.3 TMemo.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

154.1.20.3.4 TMemo.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

154.1.20.3.5 TMemo.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
```

```
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

154.1.20.3.6 TMemo.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
```

```
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.20.3.7 TMemo.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.20.3.8 TMemo.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

154.1.20.3.9 TMemo.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
```

```
    ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
```

```
    else if myActiveControl is TRichEdit then
      with myActiveControl as TRichEdit do
        SelAttributes.Assign(TFontDialog(Sender).Font)
    else
      Beep;
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

154.1.20.3.10 TMemo.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
```

```
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

154.1.20.3.11 TMemo.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
```

```
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

```
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

154.1.20.3.12 TMemo.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

154.1.20.3.13 TMemo.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

154.1.20.3.14 TMemo.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.20.3.15 TMemo.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
```

```
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.20.3.16 TMemo.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

154.1.20.3.17 TMemo.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

154.1.20.3.18 TMemo.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
```

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.20.3.19 TMemo.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

154.1.20.3.20 TMemo.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

154.1.20.3.21 TMemo.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

154.1.21 TRadioButton Class

TRadioButton is a wrapper for a system radio button.

Class Hierarchy

Pascal

```
TRadioButton = class(TButtonControl);
```

C++

```
class TRadioButton : public TButtonControl;
```

File

StdCtrls

Description

Use TRadioButton to add a radio button to a form. Radio buttons present a set of mutually exclusive options to the user- that is, only one radio button in a set can be selected at a time. When the user selects a radio button, the previously selected radio button becomes unselected. Radio buttons are frequently grouped in a radio group box (TGroupBox (see page 21474)). Add the group box to the form first, then get the radio buttons from the Component palette and put them into the group box. Or you can add a TRadioGroup, a group that already has radio buttons in it, so that you do not have to get the buttons from the Tool Palette.

By default, all radio buttons that are directly contained in the same windowed control container, such as a TRadioGroup or TPanel, are grouped. For example, two radio buttons on a form can be checked at the same time only if they are contained in separate containers, such as two different group boxes.

See Also

TCheckBox (see page 20996)

TPanel (see page 13032)

TRadioGroup (see page 13115)

Buttons and Similar Controls

Group Boxes and Radio Groups

Radio Buttons

154.1.21.1 TRadioButton Members

The following tables list the members exposed by TRadioButton.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
V	ActionChange (see page 20992)	Updates the button control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.

	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 21765)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 21765)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus

	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetActionLinkClass (see page 20993)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 21766)	Returns whether the button control is checked.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).

	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
V	SetChecked (see page 21766)	Sets the Checked property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WndProc (see page 20994)	Handles Windows messages.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.

	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	ClicksDisabled (see page 20996)	This is ClicksDisabled, a member of class TButtonControl.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 21764)	Creates an instance of TRadioButton (see page 21753).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).

	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 21766)	Returns the alignment of any text in the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetDesignVisible (see page 6178)	Makes a control visible at design time.
	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Update (see page 5789)	Processes any pending paint messages immediately.
	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	AlignDisabled (see page 6192)	Indicates child control realignment disabled.

	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 21793)	Occurs when the user clicks the control.
	OnContextPopup (see page 21795)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 21796)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 21796)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21798)	Occurs when the user drags an object over a control.
	OnEndDock (see page 21801)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21801)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 21802)	Occurs when a control receives the input focus.
	OnExit (see page 21805)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 21806)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 21809)	Occurs when a key is pressed.
	OnKeyUp (see page 21810)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 21812)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 21813)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 21815)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21815)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 21815)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 21818)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 21820)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21821)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Action (see page 21767)	Designates the action associated with the control.
	Align (see page 21768)	Determines how the control aligns within its container (parent control).
	Alignment (see page 21771)	Determines the position of the caption relative to the radio button.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 21772)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 21772)	Specifies the bi-directional mode for the control.
	Caption (see page 21773)	Specifies a text string that identifies the control to the user.
	Checked (see page 21775)	Determines whether the option represented by the radio button is selected.
	Color (see page 21776)	Specifies the background color of the control.
	Constraints (see page 21776)	Specifies the size constraints for the control.
	Ctl3D (see page 21777)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 21777)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21778)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21778)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 21780)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 21780)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic

	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 21781)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 21782)	Determines where a control looks for its color information.
	ParentCtl3D (see page 21782)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 21783)	Determines where a control looks for its font information.
	ParentShowHint (see page 21783)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 21784)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 21788)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 21790)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21790)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 21791)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.
	WordWrap (see page 21792)	Specifies whether the button text wraps to fit the width of the control.

154.1.21.2 TRadioButton.Create Constructor

Creates an instance of TRadioButton (see page 21753).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TRadioButton(TComponent * AOwner);
```

Description

Call Create to instantiate a radio button at runtime. Radio buttons added to forms at design time are created automatically.

AOwner is the component, typically a form, that becomes the value of the Owner property. The Owner is responsible for freeing the radio button object.

See Also

Owner (see page 2807)

154.1.21.3 TRadioButton Methods

The methods of the TRadioButton class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 21765)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 21765)	Creates the underlying window.
	GetChecked (see page 21766)	Returns whether the button control is checked.

	SetChecked (see page 21766)	Sets the Checked property.

Public Methods

	Name	Description
	GetControlsAlignment (see page 21766)	Returns the alignment of any text in the control.

154.1.21.3.1 TRadioButton.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

154.1.21.3.2 TRadioButton.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams (see page 6135)

CreateWindowHandle (see page 6137)

Perform (see page 5776)

DestroyWnd (see page 6141)

154.1.21.3.3 TRadioButton.GetChecked Method

Returns whether the button control is checked.

Pascal

```
function GetChecked: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetChecked();
```

Description

GetChecked is the read implementation of the Checked property.

See Also

Checked (see page 20995)

154.1.21.3.4 TRadioButton.GetControlsAlignment Method

Returns the alignment of any text in the control.

Pascal

```
function GetControlsAlignment: TAlignment; override;
```

C++

```
virtual __fastcall TAlignment GetControlsAlignment();
```

Description

Use GetControlsAlignment to determine the orientation of text in controls that do not have an Alignment (see page 21771) property.

GetControlsAlignment overrides the inherited method, returning taRightJustify if UseRightToLeftAlignment returns false. If UseRightToLeftAlignment returns true, GetControlsAlignment returns the opposite of the Alignment (see page 21771) property.

154.1.21.3.5 TRadioButton.SetChecked Method

Sets the Checked property.

Pascal

```
procedure SetChecked(Value: Boolean); override;
```

C++

```
virtual __fastcall SetChecked(Boolean Value);
```

Description

SetChecked is the write implementer for the Checked property.

See Also

Checked (see page 20995)

154.1.21.4 TRadioButton Properties

The properties of the TRadioButton class are listed here.

Published Properties

	Name	Description
	Action (see page 21767)	Designates the action associated with the control.
	Align (see page 21768)	Determines how the control aligns within its container (parent control).
	Alignment (see page 21771)	Determines the position of the caption relative to the radio button.
	Anchors (see page 21772)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 21772)	Specifies the bi-directional mode for the control.
	Caption (see page 21773)	Specifies a text string that identifies the control to the user.
	Checked (see page 21775)	Determines whether the option represented by the radio button is selected.
	Color (see page 21776)	Specifies the background color of the control.
	Constraints (see page 21776)	Specifies the size constraints for the control.
	Ctl3D (see page 21777)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 21777)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21778)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21778)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 21780)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 21780)	Controls the attributes of text written on or in the control.
	ParentBiDiMode (see page 21781)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 21782)	Determines where a control looks for its color information.
	ParentCtl3D (see page 21782)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 21783)	Determines where a control looks for its font information.
	ParentShowHint (see page 21783)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 21784)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 21788)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 21790)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21790)	Determines if the user can tab to a control.
	Visible (see page 21791)	Determines whether the component appears onscreen.
	WordWrap (see page 21792)	Specifies whether the button text wraps to fit the width of the control.

154.1.21.4.1 TRadioButton.Action Property

Designates the action associated with the control.

Pascal

```
property Action: TBasicAction;
```

C++

```
__property TBasicAction Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list

editor, they appear in the drop-down list for the Action property in the Object Inspector.

154.1.21.4.2 TRadioButton.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;
```

```
  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
```

```
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
```

```
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

154.1.21.4.3 TRadioButton.Alignment Property

Determines the position of the caption relative to the radio button.

Pascal

```
property Alignment: TLeftRight;
```

C++

```
__property TLeftRight Alignment;
```

Description

Set Alignment to determine whether the caption of the radio button should appear to the left of the radio button (taLeftJustify) or to the right of the radio button (taRightJustify).

Note: Do not confuse the Alignment property with the alignment of the caption in the text area of the radio button. The caption always appears left aligned. Alignment only controls on which side of the button the text area appears.

See Also

Caption (see page 5806)

154.1.21.4.4 TRadioButton.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

154.1.21.4.5 TRadioButton.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

154.1.21.4.6 TRadioButton.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
```

```
    {
      TToolButton *button = new TToolButton(toolBar);
      button->Parent = toolBar;
      button->Caption = captions->Strings[i];
      if (button->Caption == "|")
        button->Style = tbsSeparator;
      else
        button->Style = tbsButton;
    }
  }

  TToolBar *myToolBar;

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    myToolBar = new TToolBar(this);
    myToolBar->Parent = this;
    TStringList *captions = new TStringList();
    captions->Add("New");
    captions->Add("Save");
    captions->Add("|");
    captions->Add("Cut");
    captions->Add("Copy");
    captions->Add("Paste");
    AddButtons(myToolBar, captions);
    delete captions;
    myToolBar->ShowCaptions = True;
    myToolBar->Height = 40;
  }

  void __fastcall TForm1::FormDestroy(TObject *Sender)
  {
    for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
      delete myToolBar->Buttons[I];
    delete myToolBar;
  }
```

Delphi Examples:

```
  {
  The following code keeps the size of the label control
  constant, even though the length of the label's caption
  changes. As a result, the caption of the label is probably
  too long to display in the label when the user clicks the
  button:
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Label1.AutoSize := False;
    Label1.Caption :=
      'This string is too long as the caption of this label';
  end;
  {
  This example requires a TPageControl populated by several
  TabSheets, a TEdit and a TButton. When the button is
  clicked, the caption of the active tab changes to that of
  the TEdit.  These lines of code could also be written:
  PageControl1.ActivePage.Caption := Edit1.Text;
  This would obviate the need for a TTabSheet variable since
  PageControl1.->ActivePage is already of type TTabSheet.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    sheet: TTabSheet;
  begin
    sheet := PageControl1.ActivePage;
    sheet.Caption := Edit1.Text;
```

```
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

154.1.21.4.7 TRadioButton.Checked Property

Determines whether the option represented by the radio button is selected.

Pascal

```
property Checked: Boolean;
```

C++

```
__property Boolean Checked;
```

Description

Read Checked to determine whether the radio button is selected. Set Checked to true to select the radio button and deselect all other radio buttons in the same container. Set Checked to false to deselect the radio button, leaving no radio button in the group selected.

See Also

TRadioGroup (see page 13115)

154.1.21.4.8 TRadioButton.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

154.1.21.4.9 TRadioButton.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

154.1.21.4.10 TRadioButton.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

154.1.21.4.11 TRadioButton.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

154.1.21.4.12 TRadioButton.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

154.1.21.4.13 TRadioButton.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
```

```
  OnDragOver event handler must be implemented to accept the
  drop.
  */
  void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
        int X, int Y)
  {
    if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
      return;
    THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
    TNodeAttachMode AttachMode;
    TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
    if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
      AttachMode = naAddChild;
    else if (HT.Contains(htNowhere))
      AttachMode = naAdd;
    else if (HT.Contains(htOnIndent))
      AttachMode = naInsert;
    else
      return;
    TreeView1->Selected->MoveTo(Item, AttachMode);
  }

  void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
        int X, int Y, TDragState State, bool &Accept)
  {
    Accept = (Source == (TObject *)TreeView1);
  }
```

Delphi Examples:

```
  {
  The following code uses GetNodeAt to add a dragged node as a
  child of the node under the mouse when it is dropped.  This
  example requires a populated TreeView.  Also, the TTreeView
  DragMode property set to dmAutomatic, and the TTreeView
  OnDragOver event handler must be implemented to accept the
  drop.
  }
  procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
  var
    AnItem: TTreeNode;
    AttachMode: TNodeAttachMode;
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

154.1.21.4.14 TRadioButton.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

154.1.21.4.15 TRadioButton.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
```

```
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

154.1.21.4.16 TRadioButton.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

154.1.21.4.17 TRadioButton.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

154.1.21.4.18 TRadioButton.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

154.1.21.4.19 TRadioButton.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

154.1.21.4.20 TRadioButton.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

154.1.21.4.21 TRadioButton.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
```

```
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
```

```
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
```

```
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

154.1.21.4.22 TRadioButton.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

154.1.21.4.23 TRadioButton.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

154.1.21.4.24 TRadioButton.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

154.1.21.4.25 TRadioButton.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
```

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

154.1.21.4.26 TRadioButton.WordWrap Property

Specifies whether the button text wraps to fit the width of the control.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the display of multiple lines of text. When WordWrap is true, text that is too wide for the control wraps at the right margin.

Set WordWrap to false to limit the text to a single line. When WordWrap is false, text that is too wide for the control appears truncated.

See Also

ClientWidth (see page 5811)

154.1.21.5 TRadioButton Events

The events of the TRadioButton class are listed here.

Published Events

	Name	Description
	OnClick (see page 21793)	Occurs when the user clicks the control.
	OnContextPopup (see page 21795)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 21796)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 21796)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21798)	Occurs when the user drags an object over a control.
	OnEndDock (see page 21801)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21801)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 21802)	Occurs when a control receives the input focus.
	OnExit (see page 21805)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 21806)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 21809)	Occurs when a key is pressed.
	OnKeyUp (see page 21810)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 21812)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 21813)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 21815)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21815)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 21815)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 21818)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 21820)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21821)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

154.1.21.5.1 TRadioButton.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.

- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

154.1.21.5.2 TRadioButton.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple

times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

154.1.21.5.3 TRadioButton.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

154.1.21.5.4 TRadioButton.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
```

```
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
```

```
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

154.1.21.5.5 TRadioButton.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor

property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
```

```
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.21.5.6 TRadioButton.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.21.5.7 TRadioButton.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
```

```
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

154.1.21.5.8 TRadioButton.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter

event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
```

```
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

154.1.21.5.9 TRadioButton.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

154.1.21.5.10 TRadioButton.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR

message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
```

```
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
  {
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

154.1.21.5.11 TRadioButton.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
```

```
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

154.1.21.5.12 TRadioButton.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
```

```
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form’s color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

154.1.21.5.13 TRadioButton.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.21.5.14 TRadioButton.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
```

```
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
```

```
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.21.5.15 TRadioButton.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

154.1.21.5.16 TRadioButton.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

154.1.21.5.17 TRadioButton.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
```

```
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.21.5.18 TRadioButton.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
```

```
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
```

```
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

154.1.21.5.19 TRadioButton.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

154.1.21.5.20 TRadioButton.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

154.1.22 TScrollBar Class

TScrollBar is a Windows scroll bar, which is used to scroll the contents of a window, form, or control.

Class Hierarchy

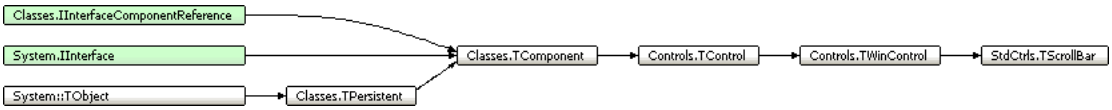

Pascal

```
TScrollBar = class(TWinControl);
```

C++

```
class TScrollBar : public TWinControl;
```

File

StdCtrls

Description

Use TScrollBar to add a free-standing scroll bar to a form. Many controls have properties that add scroll bars which are an integral part of the control (TControlScrollBar objects). TScrollBar allows controls that do not have integrated scroll bars or groupings of controls to be scrolled when the user manipulates the TScrollBar object.

When the bi-directional mode of a scroll bar is changed, its coordinate system is reversed. Position (see page 21858) zero in left-to-right mode is the left side, and position zero in right-to-left mode is the right side.

See Also

TScrollingWinControl (see page 14312)

TTrackBar (see page 5172)

TUpDown (see page 5380)

Scroll Bars

Specialized Input Controls

154.1.22.1 TScrollBar Members

The following tables list the members exposed by TScrollBar.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.

	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Change (see page 21833)	This is Change, a member of class TScrollBar.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 21833)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 21834)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.

	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.

	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.

	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	Scroll (see page 21834)	This is Scroll, a member of class TScrollBar.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.

	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 21832)	Creates and initializes an instance of TScrollBar (see page 21821).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.

	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetParams (see page 21834)	Sets the Position (see page 21858), Max (see page 21847), and Min (see page 21849) property values all at once.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.

	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 21866)	Occurs immediately after the value of Position (see page 21858) changes.
	OnContextPopup (see page 21867)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 21868)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21870)	Occurs when the user drags an object over a control.
	OnEndDock (see page 21872)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21873)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 21874)	Occurs when a control receives the input focus.
	OnExit (see page 21876)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 21878)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 21880)	Occurs when a key is pressed.
	OnKeyUp (see page 21882)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 21883)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21884)	Occurs when the user moves the mouse outside of a control.
	OnScroll (see page 21884)	Occurs when the user scrolls the scroll bar with the mouse or keyboard.
	OnStartDock (see page 21886)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21886)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 21835)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 21839)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 21840)	Specifies the bi-directional mode for the control.

	Constraints (see page 21840)	Specifies the size constraints for the control.
	Ctl3D (see page 21841)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 21841)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21841)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21842)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 21843)	Controls whether the control responds to mouse, keyboard, and timer events.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Kind (see page 21844)	Specifies whether the scroll bar is horizontal or vertical.
	LargeChange (see page 21845)	Determines how much Position (see page 21858) changes when the user clicks the scroll bar on either side of the thumb tab or presses PgUp or PgDn.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Max (see page 21847)	Specifies the maximum position represented by the scroll bar.
	Min (see page 21849)	Specifies the minimum position represented by the scroll bar.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	PageSize (see page 21851)	Specifies the size of the thumb tab.
	ParentBiDiMode (see page 21852)	Specifies whether the control uses its parent's BiDiMode.
	ParentCtl3D (see page 21853)	Determines where a component looks to determine if it should appear three dimensional.
	ParentShowHint (see page 21853)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 21853)	Identifies the pop-up menu associated with the control.
	Position (see page 21858)	Indicates the current position of the scroll bar.
	ShowHint (see page 21860)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SmallChange (see page 21861)	Determines how much Position (see page 21858) changes when the user clicks the arrow buttons on the scroll bar or presses the arrow keys on the keyboard.
	TabOrder (see page 21863)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21864)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 21864)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

154.1.22.2 TScrollBar.Create Constructor

Creates and initializes an instance of TScrollBar (see page 21821).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TScrollBar(TComponent * AOwner);
```

Description

Call Create to create an instance of TScrollBar (see page 21821) at runtime. Scroll (see page 21834) bars placed on a form at design time are created automatically.

AOwner is the component that is responsible for freeing the scroll bar. This becomes the value of the Owner property.

See Also

Owner (see page 2807)

154.1.22.3 TScrollBar Methods

The methods of the TScrollBar class are listed here.

Protected Methods

	Name	Description
	Change (see page 21833)	This is Change, a member of class TScrollBar.
V	CreateParams (see page 21833)	Initializes a window-creation parameter data structure.
V	CreateWnd (see page 21834)	Creates the underlying window.
	Scroll (see page 21834)	This is Scroll, a member of class TScrollBar.

Public Methods

	Name	Description
	SetParams (see page 21834)	Sets the Position (see page 21858), Max (see page 21847), and Min (see page 21849) property values all at once.

154.1.22.3.1 TScrollBar.Change Method

Pascal

```
procedure Change; dynamic;
```

C++

```
__fastcall Change();
```

Description

This is Change, a member of class TScrollBar.

154.1.22.3.2 TScrollBar.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call

to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

154.1.22.3.3 TScrollBar.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams (see page 6135)

CreateWindowHandle (see page 6137)

Perform (see page 5776)

DestroyWnd (see page 6141)

154.1.22.3.4 TScrollBar.Scroll Method

Pascal

```
procedure Scroll(ScrollCode: TScrollCode; var ScrollPos: Integer); dynamic;
```

C++

```
__fastcall Scroll(TScrollCode ScrollCode, int ScrollPos);
```

Description

This is Scroll, a member of class TScrollBar.

154.1.22.3.5 TScrollBar.SetParams Method

Sets the Position (see page 21858), Max (see page 21847), and Min (see page 21849) property values all at once.

Pascal

```
procedure SetParams(APosition: Integer; AMin: Integer; AMax: Integer);
```

C++

```
__fastcall SetParams(int APosition, int AMin, int AMax);
```

Description

Use the SetParams method to set the Position (see page 21858), Min (see page 21849), and Max (see page 21847) property values all at once, preventing repaints that would occur if each property value were set separately.

See Also

Max (see page 21847)

Min (see page 21849)

Position (see page 21858)

154.1.22.4 TScrollBar Properties

The properties of the TScrollBar class are listed here.

Published Properties

	Name	Description
	Align (see page 21835)	Determines how the control aligns within its container (parent control).
	Anchors (see page 21839)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 21840)	Specifies the bi-directional mode for the control.
	Constraints (see page 21840)	Specifies the size constraints for the control.
	Ctl3D (see page 21841)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DragCursor (see page 21841)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21841)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21842)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 21843)	Controls whether the control responds to mouse, keyboard, and timer events.
	Kind (see page 21844)	Specifies whether the scroll bar is horizontal or vertical.
	LargeChange (see page 21845)	Determines how much Position (see page 21858) changes when the user clicks the scroll bar on either side of the thumb tab or presses PgUp or PgDn.
	Max (see page 21847)	Specifies the maximum position represented by the scroll bar.
	Min (see page 21849)	Specifies the minimum position represented by the scroll bar.
	PageSize (see page 21851)	Specifies the size of the thumb tab.
	ParentBiDiMode (see page 21852)	Specifies whether the control uses its parent's BiDiMode.
	ParentCtl3D (see page 21853)	Determines where a component looks to determine if it should appear three dimensional.
	ParentShowHint (see page 21853)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 21853)	Identifies the pop-up menu associated with the control.
	Position (see page 21858)	Indicates the current position of the scroll bar.
	ShowHint (see page 21860)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SmallChange (see page 21861)	Determines how much Position (see page 21858) changes when the user clicks the arrow buttons on the scroll bar or presses the arrow keys on the keyboard.
	TabOrder (see page 21863)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21864)	Determines if the user can tab to a control.
	Visible (see page 21864)	Determines whether the component appears onscreen.

154.1.22.4.1 TScrollBar.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
```

```
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
```

```
    Split: TSplitter;
    Files: TFileListBox;
    Dirs: TDirectoryListBox;
  begin
    { first add a directory list box to the form }
    Dirs := TDirectoryListBox.Create(Form1);
    { line it up on the left of the form }
    Dirs.Parent := Form1;
    Dirs.Align := alLeft;
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
```

```
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
    end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

154.1.22.4.2 TScrollBar.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

154.1.22.4.3 TScrollBar.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

154.1.22.4.4 TScrollBar.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

154.1.22.4.5 TScrollBar.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

154.1.22.4.6 TScrollBar.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

154.1.22.4.7 TScrollBar.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

154.1.22.4.8 TScrollBar.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
```

```
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

154.1.22.4.9 TScrollBar.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

154.1.22.4.10 TScrollBar.Kind Property

Specifies whether the scroll bar is horizontal or vertical.

Pascal

```
property Kind: TScrollBarKind;
```

C++

```
__property TScrollBarKind Kind;
```

Description

Set Kind to indicate the orientation of the scroll bar. These are the possible values:

Value	Meaning
sbHorizontal	Scroll (see page 21834) bar is horizontal
sbVertical	Scroll (see page 21834) bar is vertical

C++ Examples:

```
/*
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Items->Add("Vertical");
  RadioGroup1->Items->Add("Horizontal");
  RadioGroup1->ItemIndex = 2;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Vertical")
    ScrollBar1->Kind = sbVertical;
  else if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Horizontal")
    ScrollBar1->Kind = sbHorizontal;
}
```

Delphi Examples:

```
{
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  RadioGroup1.Items.Add('Vertical');
  RadioGroup1.Items.Add('Horizontal');
  RadioGroup1.ItemIndex := 2;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Vertical' then
    ScrollBar1.Kind := sbVertical;
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Horizontal' then
    ScrollBar1.Kind := sbHorizontal;
end;
```

154.1.22.4.11 TScrollBar.LargeChange Property

Determines how much Position (see page 21858) changes when the user clicks the scroll bar on either side of the thumb tab or presses PgUp or PgDn.

Pascal

```
property LargeChange: TScrollBarInc;
```

C++

```
__property TScrollBarInc LargeChange;
```

Description

Set LargeChange to specify how much to change the value of Position (see page 21858) when the user clicks on the side of the thumb tab (that is, on the thumb tab track, but not on the thumb tab itself). The Max (see page 21847) and Min (see page 21849) properties define the total range over which Position (see page 21858) can vary.

For example, if Max (see page 21847) is 30000 and Min (see page 21849) is 0, the scroll box can assume 30,000 positions. If the LargeChange property setting is 10000 and the Position (see page 21858) property is 0, the user can click the scroll bar track three times after the thumb tab before it moves all the way to the end of the scroll bar (30000 / 10000 = 3).

LargeChange should correspond to the portion of this range that the user sees in one page or screen, expressed in the logical units used by the Max (see page 21847), Min (see page 21849), and Position (see page 21858) properties.

C++ Examples:

```
/*
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
```

```
  // First, initialize the page control
  PageControl1->TabWidth = PageControl1->ClientWidth/4 - 1;
  PageControl1->ActivePage = PageControl1->Pages[0];
  for (int i = 0; i < PageControl1->PageCount; i++)
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
  /* set up the scrollbar to reflect
     the structure of the page control */
  ScrollBar1->Max = PageControl1->PageCount - 1;
  ScrollBar1->Min = 0;
  ScrollBar1->SmallChange = 1;
  // set LargeChange to a single row of tabs
  ScrollBar1->LargeChange = PageControl1->ClientWidth / PageControl1->TabWidth;
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  ScrollBar1->Position = ((TPageControl *)Sender)->ActivePage->PageIndex;
}

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
  PageControl1->ActivePage = PageControl1->Pages[ScrollPos];
}
```

Delphi Examples:

```
{
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
const
  colorpalette: Array[0..11] of TColor = (clRed, clGreen, clYellow, clBlue, clWhite,
clFuchsia, clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  { First, initialize the page control }
  PageControl1.TabWidth := PageControl1.ClientWidth div 4 - 1;
  PageControl1.ActivePage := PageControl1.Pages[0];
  for i := 0 to PageControl1.PageCount - 1 do
    PageControl1.Pages[i].Brush.Color:= colorPalette[i];
  { set up the scrollbar to reflect the structure of the page control}
  with ScrollBar1 do
  begin
    Max := PageControl1.PageCount - 1;
    Min := 0;
    SmallChange := 1;
    { set LargeChange to a single row of tabs }
    LargeChange := PageControl1.ClientWidth div PageControl1.TabWidth;
  end;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  with Sender as TPageControl do
```

```
    ScrollBar1.Position := ActivePage.PageIndex;
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  PageControl1.ActivePage := PageControl1.Pages[ScrollPos];
end;
```

See Also

Max (see page 21847)

Min (see page 21849)

Position (see page 21858)

SmallChange (see page 21861)

PageSize (see page 21851)

154.1.22.4.12 TScrollBar.Max Property

Specifies the maximum position represented by the scroll bar.

Pascal

```
property Max: Integer;
```

C++

```
__property int Max;
```

Description

Set Max to the maximum value the Position (see page 21858) property can take. The Max and Min (see page 21849) properties define the total range over which Position (see page 21858) can vary.

C++ Examples:

```
/*
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  // First, initialize the page control
  PageControl1->TabWidth = PageControl1->ClientWidth/4 - 1;
  PageControl1->ActivePage = PageControl1->Pages[0];
  for (int i = 0; i < PageControl1->PageCount; i++)
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
  /* set up the scrollbar to reflect
     the structure of the page control */
  ScrollBar1->Max = PageControl1->PageCount - 1;
  ScrollBar1->Min = 0;
```

```
  ScrollBar1->SmallChange = 1;
  // set LargeChange to a single row of tabs
  ScrollBar1->LargeChange = PageControl1->ClientWidth / PageControl1->TabWidth;
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  ScrollBar1->Position = ((TPageControl *)Sender)->ActivePage->PageIndex;
}

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
  PageControl1->ActivePage = PageControl1->Pages[ScrollPos];
}
```

Delphi Examples:

```
{
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
const
  colorpalette: Array[0..11] of TColor = (clRed, clGreen, clYellow, clBlue, clWhite,
clFuchsia, clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  { First, initialize the page control }
  PageControl1.TabWidth := PageControl1.ClientWidth div 4 - 1;
  PageControl1.ActivePage := PageControl1.Pages[0];
  for i := 0 to PageControl1.PageCount - 1 do
    PageControl1.Pages[i].Brush.Color:= colorPalette[i];
  { set up the scrollbar to reflect the structure of the page control}
  with ScrollBar1 do
  begin
    Max := PageControl1.PageCount - 1;
    Min := 0;
    SmallChange := 1;
    { set LargeChange to a single row of tabs }
    LargeChange := PageControl1.ClientWidth div PageControl1.TabWidth;
  end;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  with Sender as TPageControl do
    ScrollBar1.Position := ActivePage.PageIndex;
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  PageControl1.ActivePage := PageControl1.Pages[ScrollPos];
end;
```

See Also

LargeChange (see page 21845)

Min (see page 21849)

Position (see page 21858)

SetParams (see page 21834)

SmallChange (see page 21861)

154.1.22.4.13 TScrollBar.Min Property

Specifies the minimum position represented by the scroll bar.

Pascal

```
property Min: Integer;
```

C++

```
__property int Min;
```

Description

Set Min to the minimum value the Position (see page 21858) property can take. The Min and Max (see page 21847) properties define the total range over which Position (see page 21858) can vary.

C++ Examples:

```
/*
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  // First, initialize the page control
  PageControl1->TabWidth = PageControl1->ClientWidth/4 - 1;
  PageControl1->ActivePage = PageControl1->Pages[0];
  for (int i = 0; i < PageControl1->PageCount; i++)
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
  /* set up the scrollbar to reflect
     the structure of the page control */
  ScrollBar1->Max = PageControl1->PageCount - 1;
  ScrollBar1->Min = 0;
  ScrollBar1->SmallChange = 1;
  // set LargeChange to a single row of tabs
  ScrollBar1->LargeChange = PageControl1->ClientWidth / PageControl1->TabWidth;
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  ScrollBar1->Position = ((TPageControl *)Sender)->ActivePage->PageIndex;
```

```
}

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
  PageControl1->ActivePage = PageControl1->Pages[ScrollPos];
}
```

Delphi Examples:

```
{
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
const
  colorpalette: Array[0..11] of TColor = (clRed, clGreen, clYellow, clBlue, clWhite,
clFuchsia, clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  { First, initialize the page control }
  PageControl1.TabWidth := PageControl1.ClientWidth div 4 - 1;
  PageControl1.ActivePage := PageControl1.Pages[0];
  for i := 0 to PageControl1.PageCount - 1 do
    PageControl1.Pages[i].Brush.Color:= colorPalette[i];
  { set up the scrollbar to reflect the structure of the page control}
  with ScrollBar1 do
  begin
    Max := PageControl1.PageCount - 1;
    Min := 0;
    SmallChange := 1;
    { set LargeChange to a single row of tabs }
    LargeChange := PageControl1.ClientWidth div PageControl1.TabWidth;
  end;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  with Sender as TPageControl do
    ScrollBar1.Position := ActivePage.PageIndex;
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  PageControl1.ActivePage := PageControl1.Pages[ScrollPos];
end;
```

See Also

LargeChange (see page 21845)

Max (see page 21847)

Position (see page 21858)

SetParams (see page 21834)

SmallChange (see page 21861)

154.1.22.4.14 TScrollBar.PageSize Property

Specifies the size of the thumb tab.

Pascal

```
property PageSize: Integer;
```

C++

```
__property int PageSize;
```

Description

PageSize is the size of the thumb tab, measured in the same units as Position (see page 21858), Min (see page 21849), and Max (see page 21847) (not pixels).

C++ Examples:

```
/*
The following code sets the thumb tab of a scroll bar so
that it represents the proportion of the scrolling range
that is visible:
*/
void __fastcall SetThumbTab()
{
  // Get the minimum (default) size of the thumb tab
  int MinHeight = GetSystemMetrics(SM_CYVTHUMB);
  // Get the size of the scrollbar track
  int TrackHeight = Form1->ScrollBar1->ClientHeight - 2*GetSystemMetrics(SM_CYVSCROLL);
  // set PageSize to represent the visible portion of the scrolling range:
  Form1->ScrollBar1->PageSize = TrackHeight/(Form1->ScrollBar1->Max - Form1->ScrollBar1->Min +
1);
  // Don't let the thumb tab get too small
  if (Form1->ScrollBar1->PageSize < MinHeight)
  Form1->ScrollBar1->PageSize = MinHeight;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const AnsiString Path = "../overview.rtf";
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile(Path);
  SetThumbTab();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::RichEdit1Change(TObject *Sender)
{
 SetThumbTab();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::ScrollBar1Change(TObject *Sender)
{
 SetThumbTab();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::RichEdit1EndDrag(TObject *Sender, TObject *Target,
      int X, int Y)
{
 SetThumbTab();
}
```

Delphi Examples:

```
{
```

```
The following code sets the thumb tab of a scroll bar so
that it represents the proportion of the scrolling range
that is visible:
}
procedure SetThumbTab(wincontrol: TWinControl);
var
  TrackHeight: Integer; { the size of the scroll bar track }
  MinHeight: Integer; { the default size of the thumb tab }
begin
  MinHeight := GetSystemMetrics(SM_CYVTHUMB); { save default size }
  with Form1.ScrollBar1 do
  begin
    TrackHeight := wincontrol.ClientHeight - 2 * GetSystemMetrics(SM_CYVSCROLL);
    PageSize := TrackHeight div (Max - Min + 1);
    if PageSize < MinHeight then PageSize := MinHeight;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
const
  Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := False;
  RichEdit1.Lines.LoadFromFile(Path);
  SetThumbTab(RichEdit1);
end;
procedure TForm1.RichEdit1Change(Sender: TObject);
begin
  SetThumbTab(RichEdit1);
end;
```

See Also

Min (see page 21849)

Max (see page 21847)

ClientHeight (see page 5808)

LargeChange (see page 21845)

154.1.22.4.15 TScrollBar.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

154.1.22.4.16 TScrollBar.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

154.1.22.4.17 TScrollBar.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

154.1.22.4.18 TScrollBar.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
```

```
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
```

```
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
```

```
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
```

```
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

154.1.22.4.19 TScrollBar.Position Property

Indicates the current position of the scroll bar.

Pascal

```
property Position: Integer;
```

C++

```
__property int Position;
```

Description

Read Position to determine the current position of the thumb tab. This value can be used to determine how to scroll any components controlled by the scroll bar. When the user scrolls the scroll bar, the value of Position changes. Set Position to programmatically move the thumb tab of the scroll bar.

The number of possible positions on the scroll bar is determined by the difference between the Max (see page 21847) property and the Min (see page 21849) property. If Position has the same value as Min (see page 21849), the thumb tab appears at the far left of a horizontal scroll bar or the top of a vertical scroll bar. If Position has the same value as Max (see page 21847), the thumb tab appears at the far right of a horizontal scroll bar or the bottom of a vertical scroll bar.

C++ Examples:

```
/*
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
```

```
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  // First, initialize the page control
  PageControl1->TabWidth = PageControl1->ClientWidth/4 - 1;
  PageControl1->ActivePage = PageControl1->Pages[0];
  for (int i = 0; i < PageControl1->PageCount; i++)
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
  /* set up the scrollbar to reflect
     the structure of the page control */
  ScrollBar1->Max = PageControl1->PageCount - 1;
  ScrollBar1->Min = 0;
  ScrollBar1->SmallChange = 1;
  // set LargeChange to a single row of tabs
  ScrollBar1->LargeChange = PageControl1->ClientWidth / PageControl1->TabWidth;
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  ScrollBar1->Position = ((TPageControl *)Sender)->ActivePage->PageIndex;
}

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
  PageControl1->ActivePage = PageControl1->Pages[ScrollPos];
}
```

Delphi Examples:

```
{
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
const
  colorpalette: Array[0..11] of TColor = (clRed, clGreen, clYellow, clBlue, clWhite,
clFuchsia, clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  { First, initialize the page control }
  PageControl1.TabWidth := PageControl1.ClientWidth div 4 - 1;
  PageControl1.ActivePage := PageControl1.Pages[0];
  for i := 0 to PageControl1.PageCount - 1 do
    PageControl1.Pages[i].Brush.Color:= colorPalette[i];
  { set up the scrollbar to reflect the structure of the page control}
  with ScrollBar1 do
  begin
```

```
      Max := PageControl1.PageCount - 1;
      Min := 0;
      SmallChange := 1;
      { set LargeChange to a single row of tabs }
      LargeChange := PageControl1.ClientWidth div PageControl1.TabWidth;
    end;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    with Sender as TPageControl do
      ScrollBar1.Position := ActivePage.PageIndex;
  end;

  procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
    var ScrollPos: Integer);
  begin
    PageControl1.ActivePage := PageControl1.Pages[ScrollPos];
  end;
```

See Also

Max (see page 21847)

Min (see page 21849)

SetParams (see page 21834)

154.1.22.4.20 TScrollBar.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
```

```
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

154.1.22.4.21 TScrollBar.SmallChange Property

Determines how much Position (see page 21858) changes when the user clicks the arrow buttons on the scroll bar or presses the arrow keys on the keyboard.

Pascal

```
property SmallChange: TScrollBarInc;
```

C++

```
__property TScrollBarInc SmallChange;
```

Description

Set SmallChange to specify how much to change the value of Position (see page 21858) when the user clicks on the arrow buttons. The Max (see page 21847) and Min (see page 21849) properties define the total range over which Position (see page 21858) can vary.

For example, if Max (see page 21847) is 100 and Min (see page 21849) is 0, the scroll box can assume 100 positions. If the SmallChange property setting is 5 and the Position (see page 21858) property is 0, the user can click the down or right arrow button 20 times before the thumb tab is moved all the way to the end of the scroll bar.

SmallChange should correspond to the portion of this range that the user sees as one line or one item, expressed in the logical units used by the Max (see page 21847), Min (see page 21849), and Position (see page 21858) properties.

C++ Examples:

```
/*
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  // First, initialize the page control
  PageControl1->TabWidth = PageControl1->ClientWidth/4 - 1;
  PageControl1->ActivePage = PageControl1->Pages[0];
  for (int i = 0; i < PageControl1->PageCount; i++)
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
  /* set up the scrollbar to reflect
     the structure of the page control */
  ScrollBar1->Max = PageControl1->PageCount - 1;
  ScrollBar1->Min = 0;
  ScrollBar1->SmallChange = 1;
  // set LargeChange to a single row of tabs
  ScrollBar1->LargeChange = PageControl1->ClientWidth / PageControl1->TabWidth;
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  ScrollBar1->Position = ((TPageControl *)Sender)->ActivePage->PageIndex;
}

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
  PageControl1->ActivePage = PageControl1->Pages[ScrollPos];
}
```

Delphi Examples:

```
{
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
const
  colorpalette: Array[0..11] of TColor = (clRed, clGreen, clYellow, clBlue, clWhite,
clFuchsia, clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  { First, initialize the page control }
  PageControl1.TabWidth := PageControl1.ClientWidth div 4 - 1;
  PageControl1.ActivePage := PageControl1.Pages[0];
  for i := 0 to PageControl1.PageCount - 1 do
    PageControl1.Pages[i].Brush.Color:= colorPalette[i];
  { set up the scrollbar to reflect the structure of the page control}
  with ScrollBar1 do
  begin
    Max := PageControl1.PageCount - 1;
    Min := 0;
    SmallChange := 1;
    { set LargeChange to a single row of tabs }
    LargeChange := PageControl1.ClientWidth div PageControl1.TabWidth;
  end;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  with Sender as TPageControl do
    ScrollBar1.Position := ActivePage.PageIndex;
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  PageControl1.ActivePage := PageControl1.Pages[ScrollPos];
end;
```

See Also

LargeChange (see page 21845)

Max (see page 21847)

Min (see page 21849)

Position (see page 21858)

154.1.22.4.22 TScrollBar.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

154.1.22.4.23 TScrollBar.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

154.1.22.4.24 TScrollBar.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

154.1.22.5 TScrollBar Events

The events of the TScrollBar class are listed here.

Published Events

	Name	Description
	OnChange (see page 21866)	Occurs immediately after the value of Position (see page 21858) changes.
	OnContextPopup (see page 21867)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 21868)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21870)	Occurs when the user drags an object over a control.
	OnEndDock (see page 21872)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21873)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 21874)	Occurs when a control receives the input focus.
	OnExit (see page 21876)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 21878)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 21880)	Occurs when a key is pressed.
	OnKeyUp (see page 21882)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 21883)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21884)	Occurs when the user moves the mouse outside of a control.
	OnScroll (see page 21884)	Occurs when the user scrolls the scroll bar with the mouse or keyboard.
	OnStartDock (see page 21886)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21886)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

154.1.22.5.1 TScrollBar.OnChange Event

Occurs immediately after the value of Position (see page 21858) changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to control another object or group of objects using TScrollBar (see page 21821). Adjust the objects controlled by the scroll bar to reflect the new value of Position (see page 21858).

OnChange occurs whether the value of Position (see page 21858) is changed by the user or programmatically. If the value of Position (see page 21858) is changed by the user, OnChange occurs immediately after the OnScroll (see page 21884) event.

See Also

OnScroll (see page 21884)

Position (see page 21858)

154.1.22.5.2 TScrollBar.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

154.1.22.5.3 TScrollBar.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

154.1.22.5.4 TScrollBar.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}
```

```
/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
```

```
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.22.5.5 TScrollBar.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.22.5.6 TScrollBar.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

154.1.22.5.7 TScrollBar.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
```

```
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
```

```
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

154.1.22.5.8 TScrollBar.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK

button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

154.1.22.5.9 TScrollBar.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
```

```
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
```

```
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

154.1.22.5.10 TScrollBar.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

154.1.22.5.11 TScrollBar.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
```

```
  This example changes a form's color to aqua when the user
  presses the F1 key, even when a control on the form has the
  focus. When the user releases the key, the form returns to
  its original color.  Set the KeyPreview on the form to True.
  Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
  event to FormKeyUp.  Set the KeyPreview on the form to True.
  }
  var
    FormColor: TColor;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    KeyPreview := True;
    FormColor := Form1.Color;
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

154.1.22.5.12 TScrollBar.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

154.1.22.5.13 TScrollBar.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

154.1.22.5.14 TScrollBar.OnScroll Event

Occurs when the user scrolls the scroll bar with the mouse or keyboard.

Pascal

```
property OnScroll: TScrollEvent;
```

C++

```
__property TScrollEvent OnScroll;
```

Description

Write an OnScroll to alter the way the scrollbar adjusts its position when the user scrolls the scroll bar.

C++ Examples:

```
/*
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  // First, initialize the page control
  PageControl1->TabWidth = PageControl1->ClientWidth/4 - 1;
  PageControl1->ActivePage = PageControl1->Pages[0];
  for (int i = 0; i < PageControl1->PageCount; i++)
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
  /* set up the scrollbar to reflect
     the structure of the page control */
  ScrollBar1->Max = PageControl1->PageCount - 1;
  ScrollBar1->Min = 0;
  ScrollBar1->SmallChange = 1;
  // set LargeChange to a single row of tabs
  ScrollBar1->LargeChange = PageControl1->ClientWidth / PageControl1->TabWidth;
}
```

```
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  ScrollBar1->Position = ((TPageControl *)Sender)->ActivePage->PageIndex;
}

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
  PageControl1->ActivePage = PageControl1->Pages[ScrollPos];
}
```

Delphi Examples:

```
{
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
const
  colorpalette: Array[0..11] of TColor = (clRed, clGreen, clYellow, clBlue, clWhite,
clFuchsia, clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  { First, initialize the page control }
  PageControl1.TabWidth := PageControl1.ClientWidth div 4 - 1;
  PageControl1.ActivePage := PageControl1.Pages[0];
  for i := 0 to PageControl1.PageCount - 1 do
    PageControl1.Pages[i].Brush.Color:= colorPalette[i];
  { set up the scrollbar to reflect the structure of the page control}
  with ScrollBar1 do
  begin
    Max := PageControl1.PageCount - 1;
    Min := 0;
    SmallChange := 1;
    { set LargeChange to a single row of tabs }
    LargeChange := PageControl1.ClientWidth div PageControl1.TabWidth;
  end;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  with Sender as TPageControl do
    ScrollBar1.Position := ActivePage.PageIndex;
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  PageControl1.ActivePage := PageControl1.Pages[ScrollPos];
end;
```

See Also

LargeChange (see page 21845)

OnChange (see page 21866)

Position (see page 21858)

SmallChange (see page 21861)

154.1.22.5.15 TScrollBar.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

154.1.22.5.16 TScrollBar.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

154.1.23 TStaticText Class

TStaticText is a windowed control that displays text on a form.

Class Hierarchy

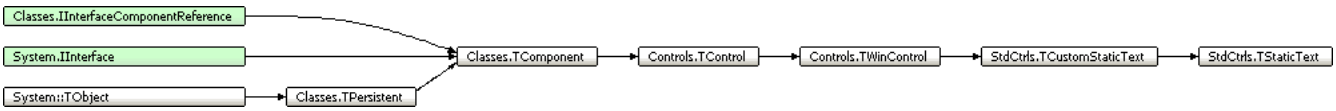

Pascal

```
TStaticText = class(TCustomStaticText);
```

C++

```
class TStaticText : public TCustomStaticText;
```

File

StdCtrls

Description

The TStaticText component functions like TLabel (see page 21544), except that it descends from TWinControl and therefore has a window handle. Use TStaticText instead of TLabel (see page 21544) when the component's accelerator key must belong to a windowed control—for example, on an ActiveX property page.

See Also

TCustomStaticText (see page 21387)

TLabel (see page 21544)

TWinControl (see page 6112)

Labels

Display Controls

154.1.23.1 TStaticText Members

The following tables list the members exposed by TStaticText.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.

	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 21398)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
V	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
V	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.

	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.

	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
V	Loaded (see page 21399)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 21399)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetAutoSize (see page 21399)	This is SetAutoSize, a member of class TCustomStaticText.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.

	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 5731)	Creates an instance of TControl (see page 5724).
	Create (see page 6123)	Creates an instance of TWinControl (see page 6112).
	Create (see page 21397)	Creates and initializes a TCustomStaticText (see page 21387) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.

	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.

R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 21926)	Occurs when the user clicks the control.

	OnContextPopup (see page 21928)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 21928)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 21929)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 21931)	Occurs when the user drags an object over a control.
	OnEndDock (see page 21933)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21934)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 21935)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 21936)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 21938)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21938)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 21939)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 21941)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 21943)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21944)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 21899)	Determines how the control aligns within its container (parent control).
	Alignment (see page 21902)	Determines how text is aligned within the component.
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 21903)	Specifies how the control is anchored to its parent.
	AutoSize (see page 21903)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 21904)	Specifies which edges of the control are beveled.
	BevelInner (see page 21904)	Specifies the cut of the inner bevel.
	BevelKind (see page 21904)	Specifies the control's bevel style.
	BevelOuter (see page 21905)	Specifies the cut of the outer bevel.
	BiDiMode (see page 21905)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 21906)	Determines the appearance of the control's border.
	Caption (see page 21906)	Specifies a text string that identifies the control to the user.
	Color (see page 21909)	Specifies the background color of the control.
	Constraints (see page 21909)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DragCursor (see page 21910)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21910)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21911)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 21912)	Controls whether the control responds to mouse, keyboard, and timer events.
	FocusControl (see page 21913)	Links the component with another windowed control.
	Font (see page 21913)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.

	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBiDiMode (see page 21914)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 21915)	Determines where a control looks for its color information.
	ParentFont (see page 21915)	Determines where a control looks for its font information.
	ParentShowHint (see page 21916)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 21916)	Identifies the pop-up menu associated with the control.
	ShowAccelChar (see page 21921)	Enables accelerator characters.
	ShowHint (see page 21921)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 21923)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21923)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Transparent (see page 21924)	Specifies whether the background of the image obscures objects below the image object.
	Visible (see page 21924)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

154.1.23.2 TStaticText Properties

The properties of the TStaticText class are listed here.

Published Properties

	Name	Description
	Align (see page 21899)	Determines how the control aligns within its container (parent control).
	Alignment (see page 21902)	Determines how text is aligned within the component.
	Anchors (see page 21903)	Specifies how the control is anchored to its parent.
	AutoSize (see page 21903)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 21904)	Specifies which edges of the control are beveled.
	BevelInner (see page 21904)	Specifies the cut of the inner bevel.
	BevelKind (see page 21904)	Specifies the control's bevel style.
	BevelOuter (see page 21905)	Specifies the cut of the outer bevel.
	BiDiMode (see page 21905)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 21906)	Determines the appearance of the control's border.
	Caption (see page 21906)	Specifies a text string that identifies the control to the user.
	Color (see page 21909)	Specifies the background color of the control.
	Constraints (see page 21909)	Specifies the size constraints for the control.
	DragCursor (see page 21910)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 21910)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 21911)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 21912)	Controls whether the control responds to mouse, keyboard, and timer events.
	FocusControl (see page 21913)	Links the component with another windowed control.
	Font (see page 21913)	Controls the attributes of text written on or in the control.
	ParentBiDiMode (see page 21914)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 21915)	Determines where a control looks for its color information.
	ParentFont (see page 21915)	Determines where a control looks for its font information.
	ParentShowHint (see page 21916)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 21916)	Identifies the pop-up menu associated with the control.
	ShowAccelChar (see page 21921)	Enables accelerator characters.
	ShowHint (see page 21921)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

	TabOrder (see page 21923)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 21923)	Determines if the user can tab to a control.
	Transparent (see page 21924)	Specifies whether the background of the image obscures objects below the image object.
	Visible (see page 21924)	Determines whether the component appears onscreen.

154.1.23.2.1 TStaticText.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
```

```
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
```

```
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
```

```
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

154.1.23.2.2 TStaticText.Alignment Property

Determines how text is aligned within the component.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

The possible values of Alignment are

Value	Meaning
taLeftJustify	Text is aligned at the left of the control.
taCenter	Text is centered in the control.
taRightJustify	Text is aligned at the right of the control.

See Also

Font (see page 5823)

Caption (see page 5806)

154.1.23.2.3 TStaticText.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

154.1.23.2.4 TStaticText.AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;__property Boolean AutoSize;
```

Description

Use AutoSize to specify whether the control sizes itself automatically. When AutoSize is true, the control resizes automatically when its contents change.

By default, AutoSize is false.

See Also

Constraints (see page 5812)

CanAutoSize (see page 5741)

154.1.23.2.5 TStaticText.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges: TBevelEdges;
```

C++

```
__property TBevelEdges BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner (see page 6193)

BevelOuter (see page 6194)

BevelKind (see page 6193)

154.1.23.2.6 TStaticText.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner: TBevelCut;
```

C++

```
__property TBevelCut BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelKind (see page 6193)

BevelWidth (see page 6194)

BorderWidth (see page 6195)

154.1.23.2.7 TStaticText.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind: TBevelKind;
```

C++

```
__property TBevelKind BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges (see page 6192)

BevelOuter (see page 6194)

BevelInner (see page 6193)

BevelWidth (see page 6194)

154.1.23.2.8 TStaticText.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter: TBevelCut;
```

C++

```
__property TBevelCut BevelOuter;
```

Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges (see page 6192)

BevelInner (see page 6193)

BevelKind (see page 6193)

BevelWidth (see page 6194)

154.1.23.2.9 TStaticText.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

154.1.23.2.10 TStaticText.BorderStyle Property

Determines the appearance of the control's border.

Pascal

```
property BorderStyle: TStaticBorderStyle;
```

C++

```
__property TStaticBorderStyle BorderStyle;
```

Description

Use BorderStyle to specify the appearance of the border, if any, that appears around the text.

154.1.23.2.11 TStaticText.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
```

```
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

154.1.23.2.12 TStaticText.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

154.1.23.2.13 TStaticText.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

154.1.23.2.14 TStaticText.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

154.1.23.2.15 TStaticText.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

154.1.23.2.16 TStaticText.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
```

```
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

154.1.23.2.17 TStaticText.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

154.1.23.2.18 TStaticText.FocusControl Property

Links the component with another windowed control.

Pascal

```
property FocusControl: TWinControl;
```

C++

```
__property TWinControl * FocusControl;
```

Description

Use the FocusControl property to link the TCustomStaticText instance with another windowed control. If the text of Caption includes an accelerator key, the component specified by FocusControl receives focus when the accelerator is pressed.

See Also

Caption (see page 5806)

FocusControl (see page 21306)

TWinControl (see page 6112)

154.1.23.2.19 TStaticText.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
```

```
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

154.1.23.2.20 TStaticText.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern

locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

154.1.23.2.21 TStaticText.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

154.1.23.2.22 TStaticText.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

154.1.23.2.23 TStaticText.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

154.1.23.2.24 TStaticText.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
```

```
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
```

```
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
```

```
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}
```

```
type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;
```

```
procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

154.1.23.2.25 TStaticText.ShowAccelChar Property

Enables accelerator characters.

Pascal

```
property ShowAccelChar: Boolean;
```

C++

```
__property Boolean ShowAccelChar;
```

Description

If ShowAccelChar is true, an ampersand in the component's Caption text appears as an underline and makes the character after it into an accelerator key. If ShowAccelChar is false, ampersands appear as ampersands.

See Also

Caption (see page 5806)

154.1.23.2.26 TStaticText.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

154.1.23.2.27 TStaticText.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

154.1.23.2.28 TStaticText.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

154.1.23.2.29 TStaticText.Transparent Property

Specifies whether the background of the image obscures objects below the image object.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Set Transparent to true to allow objects behind the TCustomStaticText object to show through the background of the image. Set Transparent to false to make the background of the image opaque.

154.1.23.2.30 TStaticText.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
```

```
    }
    PageControl1Change(Sender);
  }
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

154.1.23.3 TStaticText Events

The events of the TStaticText class are listed here.

Published Events

	Name	Description
	OnClick (see page 21926)	Occurs when the user clicks the control.
	OnContextPopup (see page 21928)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 21928)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 21929)	Occurs when the user drops an object being dragged.

	OnDragOver (see page 21931)	Occurs when the user drags an object over a control.
	OnEndDock (see page 21933)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 21934)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 21935)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 21936)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 21938)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 21938)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 21939)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 21941)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 21943)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 21944)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

154.1.23.3.1 TStaticText.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
```

```
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

154.1.23.3.2 TStaticText.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

154.1.23.3.3 TStaticText.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

154.1.23.3.4 TStaticText.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
```

```
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

154.1.23.3.5 TStaticText.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
```

```
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.23.3.6 TStaticText.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

154.1.23.3.7 TStaticText.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
```

```
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

154.1.23.3.8 TStaticText.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate

event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.23.3.9 TStaticText.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
```

```
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
```

```
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.23.3.10 TStaticText.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

154.1.23.3.11 TStaticText.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

154.1.23.3.12 TStaticText.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

154.1.23.3.13 TStaticText.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
```

```
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
```

```
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

154.1.23.3.14 TStaticText.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

154.1.23.3.15 TStaticText.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

154.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCheckBoxState (see page 21945)	TCheckBoxState indicates the state of a check box control.
	TComboBoxStyle (see page 21946)	TComboBoxStyle is the type for the Style property of combo box controls that are defined in the StdCtrls (see page 20907) unit.
	TEditCharCase (see page 21946)	TEditCharCase indicates how a control forces the case of characters, if at all.
	TEllipsisPosition (see page 21947)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	TListBoxStyle (see page 21948)	TListBoxStyle specifies the way a listbox control gets its data and how it is drawn.
	TScrollCode (see page 21949)	TScrollCode indicates what action the user performed on a scroll bar.
	TScrollStyle (see page 21950)	Identifies a set of scrollbars by their orientation.
	TStaticBorderStyle (see page 21951)	TStaticBorderStyle determines the appearance of the border on a static text control.
	TTextLayout (see page 21951)	TTextLayout indicates where text appears within the client area of a control.

Records

	Name	Description
	TSelection (see page 21950)	TSelection definese the start and position of a selection.

154.2.1 StdCtrls.TCheckBoxState Enumeration

TCheckBoxState indicates the state of a check box control.

Pascal

```
TCheckBoxState = (
  cbUnchecked,
  cbChecked,
  cbGrayed
);
```

C++

```
enum TCheckBoxState {
  cbUnchecked,
  cbChecked,
  cbGrayed
};
```

File

StdCtrls

Description

Following are the possible values of TCheckBoxState:

Value	Meaning
cbUnchecked	The check box has no check mark, indicating that the user has not selected the option.
cbChecked	The check box has a check mark in it, indicating that the user has selected the option.

cbGrayed	The check box has a check mark in it, but it is grayed.

154.2.2 StdCtrls.TComboBoxStyle Enumeration

TComboBoxStyle is the type for the Style property of combo box controls that are defined in the StdCtrls (see page 20907) unit.

Pascal

```
TComboBoxStyle = (
  csDropDown,
  csSimple,
  csDropDownList,
  csOwnerDrawFixed,
  csOwnerDrawVariable
);
```

C++

```
enum TComboBoxStyle {
  csDropDown,
  csSimple,
  csDropDownList,
  csOwnerDrawFixed,
  csOwnerDrawVariable
};
```

File

StdCtrls

Description

These are the values in the TComboBoxStyle type:

Value	Meaning
csDropDown	Creates a drop-down list with an edit box for manually entered text. All items are strings of the same height.
csSimple	Creates an edit box with a fixed list (list box) underneath. The length of the list is determined by the Height of the combo box.
csDropDownList	Creates a drop-down list with no edit box; the user cannot enter text manually. All items are strings of the same height.
csOwnerDrawFixed	Creates an owner-draw drop-down list with no edit box. Each item in the list is the height specified by the ItemHeight property.
csOwnerDrawVariable	Creates an owner-draw drop-down list with no edit box. List items can have varying heights.

154.2.3 StdCtrls.TEditCharCase Enumeration

TEditCharCase indicates how a control forces the case of characters, if at all.

Pascal

```
TEditCharCase = (
```

```
    ecNormal,
    ecUpperCase,
    ecLowerCase
  );
```

C++

```
  enum TEditCharCase {
    ecNormal,
    ecUpperCase,
    ecLowerCase
  };
```

File

StdCtrls

Description

TEditCharCase specifies what case is given to text typed into a control. It can have one of the following values:

Value	Meaning
ecLowerCase	The text is converted to lowercase.
ecNormal	The text appears in mixed case. It is not forced into any case.
ecUpperCase	The text is converted to uppercase.

154.2.4 StdCtrls.TEllipsisPosition Enumeration

Specifies how ellipsis is placed in text not fitting in allocated rectangle.

Pascal

```
  TEllipsisPosition = (
    epNone,
    epPathEllipsis,
    epEndEllipsis,
    epWordEllipsis
  );
```

C++

```
  enum TEllipsisPosition {
    epNone,
    epPathEllipsis,
    epEndEllipsis,
    epWordEllipsis
  };
```

File

StdCtrls

Description

TEllipsisPosition specifies how an ellipsis is placed in text that does not fit in its allocated rectangle. The constants here correspond to constants specifying ellipsis placement in the **dwDTFormat** parameter of the Windows **DrawText** and **DrawTextEx** functions.

The following table lists values for TEllipsisPosition:

Value	Meaning
epNone	An ellipsis is not inserted.
epPathEllipsis	Replaces text in the middle with an ellipsis so that the resulting text fits in the rectangle specified. If the text contains backslash (\) characters, treat the text as a path and attempt to preserve as much text as possible after the last backslash. Corresponds to a value of DT_PATH_ELLIPSIS in the **dwDTFormat** parameter.
epEndEllipsis	Replaces text at the end with an ellipsis so that the resulting text fits in the rectangle specified. Any word not fitting in the rectangle is truncated without adding an ellipsis. Corresponds to a value of DT_END_ELLIPSIS in the **dwDTFormat** parameter.
epWordEllipsis	Truncates any word that doesn't fit in the rectangle and adds ellipses. Corresponds to a value of DT_WORD_ELLIPSIS in the **dwDTFormat** parameter.

154.2.5 StdCtrls.TListBoxStyle Enumeration

TListBoxStyle specifies the way a listbox control gets its data and how it is drawn.

Pascal

```
TListBoxStyle = (
  lbStandard,
  lbOwnerDrawFixed,
  lbOwnerDrawVariable,
  lbVirtual,
  lbVirtualOwnerDraw
);
```

C++

```
enum TListBoxStyle {
  lbStandard,
  lbOwnerDrawFixed,
  lbOwnerDrawVariable,
  lbVirtual,
  lbVirtualOwnerDraw
};
```

File

StdCtrls

Description

The following table lists the values for the TListBoxStyle type:

Value	Meaning
lbStandard	All items are strings, with each item the same height.
lbOwnerDrawFixed	The list box is owner-drawn, but each item in the list box is the height specified by the ItemHeight property. Each time an item is displayed in an lbOwnerDrawFixed list box, the OnDrawItem event occurs. The event handler for OnDrawItem draws the specified item. The ItemHeight property determines the height of each of the items.
lbOwnerDrawVariable	The list box is owner-drawn, and items in the list box can be of varying heights. Each time an item is displayed in an lbOwnerDrawVariable list box, two events occur. The first is the OnMeasureItem event. The code for the OnMeasureItem handler can set the height of each item. Then the OnDrawItem event occurs. The code for the OnDrawItem handler draws each item in the list box using the size specified by the OnMeasureItem handler.

lbVirtual	The list box is virtual, but all items are strings with the same height. You must indicate the number of items in the list box by setting the Count property. You must supply the items in the list box using an OnData event handler. If the items have associated objects, you must supply them using an OnDataObject event handler. In addition, you must supply an OnDataFind event handler to provide the mapping from the strings in the list box to their indexes.
lbVirtualOwnerDraw	The list box is virtual and owner-drawn. All items are the same height. As with lbOwnerDrawFixed, the item height is specified by the ItemHeight property, and you must draw items using an OnDrawItem event handler. As with lbVirtual, you must set the Count property and supply OnData and OnDataFind event handlers to provide the mapping between items and indexes. If items are associated with objects, you must also supply an OnDataObject event handler.

154.2.6 StdCtrls.TScrollCode Enumeration

TScrollCode indicates what action the user performed on a scroll bar.

Pascal

```
TScrollCode = (
  scLineUp,
  scLineDown,
  scPageUp,
  scPageDown,
  scPosition,
  scTrack,
  scTop,
  scBottom,
  scEndScroll
);
```

C++

```
enum TScrollCode {
  scLineUp,
  scLineDown,
  scPageUp,
  scPageDown,
  scPosition,
  scTrack,
  scTop,
  scBottom,
  scEndScroll
};
```

File

StdCtrls

Description

TScrollCode describes the type of scrolling action requested. It can be any of the following values:

Value	Meaning
scLineUp	User clicked the top or left scroll arrow or pressed the Up or Left arrow key.
scLineDown	User clicked the bottom or right scroll arrow or pressed the Down or Right arrow key.
scPageUp	User clicked the area to the left of the thumb tab or pressed the PgUp key.
scPageDown	User clicked the area to the right of the thumb tab or pressed the PgDn key.
scPosition	User positioned the thumb tab and released it.

scTrack	User is moving the thumb tab.
scTop	User moved the thumb tab to the top or far left on the scroll bar.
scBottom	User moved the thumb tab to the bottom or far right on the scroll bar.
scEndScroll	User finished moving the thumb tab on the scroll bar.

154.2.7 StdCtrls.TScrollStyle Enumeration

Identifies a set of scrollbars by their orientation.

Pascal

```
TScrollStyle = (
  ssNone,
  ssHorizontal,
  ssVertical,
  ssBoth
);
```

C++

```
enum TScrollStyle {
  ssNone,
  ssHorizontal,
  ssVertical,
  ssBoth
};
```

File

StdCtrls

Description

TScrollStyle can have one of the following values:

Value	Meaning
ssNone	No scrollbars are indicated.
ssHorizontal	Only a horizontal scrollbar.
ssVertical	Only a vertical scrollbar.
ssBoth	Both a horizontal and a vertical scrollbar.

154.2.8 StdCtrls.TSelection Record

TSelection definese the start and position of a selection.

Pascal

```
TSelection = record
  EndPos: Integer;
  StartPos: Integer;
end;
```

C++

```
struct TSelection {
  int EndPos;
  int StartPos;
};
```

File

StdCtrls

Description

TSelection is used internally in Delphi to hold the start and end position of a selection.

154.2.9 StdCtrls.TStaticBorderStyle Enumeration

TStaticBorderStyle determines the appearance of the border on a static text control.

Pascal

```
TStaticBorderStyle = (
  sbsNone,
  sbsSingle,
  sbsSunken
);
```

C++

```
enum TStaticBorderStyle {
  sbsNone,
  sbsSingle,
  sbsSunken
};
```

File

StdCtrls

Description

These are the values in the TStaticBorderStyle type:

Value	Meaning
sbsNone	No visible border.
sbsSingle	Single-line border.
sbsSunken	Raised border.

154.2.10 StdCtrls.TTextLayout Enumeration

TTextLayout indicates where text appears within the client area of a control.

Pascal

```
TTextLayout = (
  tlTop,
  tlCenter,
  tlBottom
```

```
);
```

C++

```
enum TTextLayout {
  tlTop,
  tlCenter,
  tlBottom
};
```

File

StdCtrls

Description

The following table describes the possible values:

Value	Meaning
tlTop	The text appears at the top of the its control.
tlCenter	The text is vertically centered in the control.
tlBottom	The text appears along the bottom of the control.

154.3 Types

The following table lists types in this documentation.

Types

Name	Description
TButtonActionLinkClass (see page 21953)	This is type StdCtrls.TButtonActionLinkClass.
TCustomComboBoxStringsClass (see page 21953)	TCustomComboBoxStringsClass describes a descendant of TCustomComboBoxStrings (see page 21225).
TDrawItemEvent (see page 21954)	TScrollEvent (see page 21957) is the type of an OnScroll event handler.
TLBFindDataEvent (see page 21954)	TLBFindDataEvent is the type of the list box OnDataFind event handler.
TLBGetDataEvent (see page 21955)	TLBGetDataEvent is the type of the list box OnData event handler.
TLBGetDataObjectEvent (see page 21955)	TLBGetDataObjectEvent is the type of the list box OnDataObject event handler.
TMeasureItemEvent (see page 21955)	TMeasureItemEvent is the type for the OnMeasureItem event handler.
TOwnerDrawState (see page 21956)	Indicates state information that can influence how an item is drawn.
TScrollEvent (see page 21957)	This is type StdCtrls.TScrollEvent.

154.3.1 StdCtrls.TButtonActionLinkClass Type

Pascal

```
TButtonActionLinkClass = class of TButtonActionLink;
```

C++

```
TButtonActionLink TButtonActionLinkClass;
```

File

StdCtrls

Description

This is type StdCtrls.TButtonActionLinkClass.

154.3.2 StdCtrls.TCustomComboBoxStringsClass Type

TCustomComboBoxStringsClass describes a descendant of TCustomComboBoxStrings (see page 21225).

Pascal

```
TCustomComboBoxStringsClass = class of TCustomComboBoxStrings;
```

C++

```
TCustomComboBoxStrings TCustomComboBoxStringsClass;
```

File

StdCtrls

Description

TCustomComboBoxStringsClass is the type for a class reference of TCustomComboBoxStrings (see page 21225) or any of its

descendants. It is a pointer to the runtime type information (RTTI) for the class type.

154.3.3 StdCtrls.TDrawItemEvent Type

TScrollEvent (see page 21957) is the type of an OnScroll event handler.

Pascal

```
TDrawItemEvent = procedure (Control: TWinControl; Index: Integer; Rect: TRect; State:
TOwnerDrawState) of object;
```

C++

```
(Control: TWinControl; Index: Integer; Rect: TRect; State: TOwnerDrawState) (
TDrawItemEvent)();
```

File

StdCtrls

Description

TScrollEvent (see page 21957) defines the type of method procedure that you must define in order to implement the OnScroll event of a visual component such as TScroll.

The OnScroll event occurs when the user scrolls a scroll bar with the mouse or keyboard.

Sender is a reference to the component invoking the OnScroll event.

ScrollCode defines the user scroll action that was performed.

ScrollPos provides the new scroll position.

See Also

OnScroll

TScrollCode (see page 21949)

154.3.4 StdCtrls.TLBFindDataEvent Type

TLBFindDataEvent is the type of the list box OnDataFind event handler.

Pascal

```
TLBFindDataEvent = function (Control: TWinControl; FindString: string): Integer of object;
```

C++

```
(Control: TWinControl; FindString: string): Integer ( TLBFindDataEvent)();
```

File

StdCtrls

Description

Control is the virtual list box whose item index is needed.

FindString is the text of the item whose index is needed.

The event handler returns the index of the item identified by FindString. This return value must be in the range from 0 to one less

than the value of the Count property.

154.3.5 StdCtrls.TLBGetDataEvent Type

TLBGetDataEvent is the type of the list box OnData event handler.

Pascal

```
TLBGetDataEvent = procedure (Control: TWinControl; Index: Integer; var Data: string) of object;
```

C++

```
(Control: TWinControl; Index: Integer; var Data: string) ( TLBGetDataEvent)();
```

File

StdCtrls

Description

Control is the virtual list box whose item is needed.

Index is the index of the desired item. Index must be less than the value of the Count property.

Data returns the string that is the value of the specified item.

154.3.6 StdCtrls.TLBGetDataObjectEvent Type

TLBGetDataObjectEvent is the type of the list box OnDataObject event handler.

Pascal

```
TLBGetDataObjectEvent = procedure (Control: TWinControl; Index: Integer; var DataObject:
TObject) of object;
```

C++

```
(Control: TWinControl; Index: Integer; var DataObject: TObject) ( TLBGetDataObjectEvent)();
```

File

StdCtrls

Description

Control is the virtual list box that contains the item associated with the object.

Index is the index of the desired item. Index must be less than the value of the Count property.

DataObject returns the object associated with the specified item.

154.3.7 StdCtrls.TMeasureItemEvent Type

TMeasureItemEvent is the type for the OnMeasureItem event handler.

Pascal

```
TMeasureItemEvent = procedure (Control: TWinControl; Index: Integer; var Height: Integer) of
```

```
object;
```

C++

```
(Control: TWinControl; Index: Integer; var Height: Integer) ( TMeasureItemEvent)();
```

File

StdCtrls

Description

TMeasureItemEvent is the type for event handlers on custom-drawn controls that allow different items in a list to have different heights.

Control is a reference to the control that contains the item.

Index is the index of the item in the Items property.

Height is the default height of the item, in pixels. The event handler can reset Height to the appropriate value.

154.3.8 StdCtrls.TOwnerDrawState Type

Indicates state information that can influence how an item is drawn.

Pascal

```
TOwnerDrawState = Windows.TOwnerDrawState;
```

C++

```
Windows.TOwnerDrawState TOwnerDrawState;
```

File

StdCtrls

Description

TOwnerDrawState is used by event handlers in owner-draw controls to indicate the state of an item about to be drawn. It is a set that includes zero or more of the following:

Value	Meaning
odSelected	The item is selected.
odGrayed	The item should appear grayed.
odDisabled	The item is disabled.
odChecked	The item should appear checked. (This does not apply to all objects)
odFocused	Keyboard input is directed to the item.
odDefault	The item is the default item (This does not apply to all objects)
odHotLight	The item is hot-lighted because under the mouse (only if the operating system supports hot-lighting)
odInactive	The item and its associated menu are inactive
odNoAccel	Any underline of an accelerator character should be skipped when drawing the control
odNoFocusRect	Do not display visual cues that indicate focus
odReserved1	Not currently used.
odReserved2	Not currently used.

odComboBoxEdit	The drawing takes place in the edit box of a combo box control.

154.3.9 StdCtrls.TScrollEvent Type

Pascal

```
TScrollEvent = procedure (Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer) of
object;
```

C++

```
(Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer) ( TScrollEvent)();
```

File

StdCtrls

Description

This is type StdCtrls.TScrollEvent.

155 StrUtils Namespace

This is namespace StrUtils.

Topics

Name	Description
Functions (see page 21962)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 21992)	The following table lists structs, records, enums in this documentation.
Types (see page 21993)	The following table lists types in this documentation.
Variables (see page 21995)	The following table lists variables in this documentation.
Constants (see page 21997)	The following table lists constants in this documentation.

Constants

Name	Description
WordDelimiters (see page 21997)	This is constant StrUtils.WordDelimiters.

Functions

	Name	Description
	AnsiContainsStr (see page 21963)	Indicates whether one string is a (case-sensitive) substring of another.
	AnsiContainsText (see page 21964)	Indicates whether one string is a (case-insensitive) substring of another.
	AnsiEndsStr (see page 21964)	Indicates whether one string is a (case-sensitive) suffix of another.
	AnsiEndsText (see page 21965)	Indicates whether one string is a (case-insensitive) suffix of another.
	AnsiIndexStr (see page 21965)	Provides the index of a specified string in an array of strings.
	AnsiIndexText (see page 21966)	Provides the index of a specified string in an array of strings.
	AnsiLeftStr (see page 21966)	Returns the substring of a specified length that appears at the start of a string.
	AnsiMatchStr (see page 21967)	Indicates whether an array of strings contains an exact match to a specified string.
	AnsiMatchText (see page 21967)	Indicates whether an array of strings contains a case-insensitive match to a specified string.
	AnsiMidStr (see page 21968)	Returns the substring of a specified length that appears at a specified position in a string.
	AnsiReplaceStr (see page 21969)	Replaces all occurrences of a substring with another string.
	AnsiReplaceText (see page 21969)	Replaces all case-insensitive matches of a substring with another string.
	AnsiResemblesText (see page 21970)	Indicates whether two strings are similar.
	AnsiReverseString (see page 21970)	Returns reversed string.
	AnsiRightStr (see page 21971)	Returns the substring of a specified length that appears at the end of a string.
	AnsiStartsStr (see page 21971)	Indicates whether one string is a (case-sensitive) prefix of another.
	AnsiStartsText (see page 21972)	Indicates whether one string is a (case-insensitive) prefix of another.

	ContainsStr (see page 21972)	Performs a case sensitive search for substring ASubText in string AText. Returns true if one or more instances of ASubText exist in AText.
	ContainsText (see page 21973)	Performs a case insensitive search for substring ASubText in string AText. Returns true if one or more instances of ASubText exist in AText.
	DecodeSoundexInt (see page 21973)	Converts an integer representation of a Soundex (see page 21986) encoding into the corresponding phonetic string.
	DecodeSoundexWord (see page 21974)	Converts a Word representation of a Soundex (see page 21986) encoding into the corresponding phonetic string.
	DupeString (see page 21974)	Returns the concatenation of a string with itself a specified number of repeats.
	EndsStr (see page 21975)	Determines if string AText ends with substring ASubText using a case sensitive string comparison. Returns true if AText ends with ASubText.
	EndsText (see page 21975)	Determines if string AText ends with substring ASubText using a case insensitive string comparison. Returns true if AText ends with ASubText.
	IfThen (see page 21976)	Conditionally returns one of two specified values.
	IndexStr (see page 21976)	IndexStr determines if any of the strings in the array AValues match the string specified by AText using a case sensitive comparison. It returns the zero-based index of the first match in the array, or -1 if no match is found.
	IndexText (see page 21977)	IndexText determines if any of the strings in the array AValues match the string specified by AText using a case insensitive comparison. It returns the zero-based index of the first match in the array, or -1 if no match is found.
	LeftBStr (see page 21977)	Returns the substring of a specified number of bytes that appears at the start of a string.
	LeftStr (see page 21978)	Returns the substring of a specified length that appears at the start of a string.
	LeftStr (see page 21978)	Returns the substring of a specified length that appears at the start of a string.
	MatchStr (see page 21979)	MatchStr determines if any of the strings in the array AValues match the string specified by AText using a case sensitive comparison. It returns true if at least one of the strings in the array match, or false if none of the strings match.
	MatchText (see page 21979)	MatchText determines if any of the strings in the array AValues match the string specified by AText using a case insensitive comparison. It returns true if at least one of the strings in the array match, or false if none of the strings match.
	MidBStr (see page 21980)	Returns the substring of a specified number of bytes that appears at a specified position in a string.
	MidStr (see page 21980)	Returns the substring of a specified length that appears at a specified position in a string.
	MidStr (see page 21980)	Returns the substring of a specified length that appears at a specified position in a string.
	PosEx (see page 21981)	Returns the index value of a substring.
	RandomFrom (see page 21981)	Returns a randomly selected element from an array.
	ReplaceStr (see page 21982)	ReplaceStr replaces all instances of string AFromText to string AToText in the source string AText and returns this value as the result. The replacement is case sensitive.
	ReplaceText (see page 21982)	ReplaceText replaces all instances of string AFromText to string AToText in the source string AText and returns this value as the result. The replacement is case insensitive.
	ResemblesText (see page 21983)	ResemblesText uses a Soundex (see page 21986) method to determine if string AText and string AOther are similar. The result is True if the two strings a similar according to the Soundex (see page 21986) method, and false otherwise.
	ReverseString (see page 21983)	Returns the reverse of a specified string.
	RightBStr (see page 21984)	Returns the substring of a specified number of bytes that appears at the end of a string.
	RightStr (see page 21984)	Returns the substring of a specified length that appears at the end of a string.
	RightStr (see page 21984)	Returns the substring of a specified length that appears at the end of a string.
	SearchBuf (see page 21985)	Locates a substring within a text buffer.
	Soundex (see page 21986)	Converts a string into its Soundex representation.
	SoundexCompare (see page 21986)	Compares the Soundex (see page 21986) representations of two strings.
	SoundexInt (see page 21987)	Converts a string into an integer that represents its phonetic value.
	SoundexProc (see page 21987)	Indicates whether two strings are similar.
	SoundexSimilar (see page 21988)	Indicates whether two strings are similar.
	SoundexWord (see page 21989)	Converts a string into a Word that represents its phonetic value.
	StartsStr (see page 21989)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.

	StartsText (see page 21990)	StartsText determines if the substring ASubText begins the string AText using a case insensitive algorithm. If ASubText matches the begining of AText, the result is true, otherwise it is false.
	StuffString (see page 21990)	Inserts a substring into a specified position of a string, replacing the current characters.

Structs, Records, Enums

	Name	Description
	TStringSeachOption (see page 21992)	TStringSeachOption is an enumeration that describes the behavior of string searches.

Types

Name	Description
TCompareTextProc (see page 21993)	TCompareTextProc is a function pointer which specifies a function to compare two string values, AText, and AOther, and return true if these two values are equivilant.
TSoundexIntLength (see page 21993)	TSoundexIntLength The maximum calculation length for the SoundexInt (see page 21987) function. The integer value can be from 1 to 8.
TSoundexLength (see page 21994)	TSoundexLength The maximum calculation length of the SoundEx function. This value determines how many characters to use when generating the soundex string form the source string.
TStringSearchOptions (see page 21994)	TStringSearchOptions is set of the enumeration TStringSeachOption (see page 21992).

Variables

Name	Description
AnsiResemblesProc (see page 21995)	Controls (see page 5711) the algorithm used by AnsiResemblesText (see page 21970) to determine when two strings are similar.
ResemblesProc (see page 21995)	Controls (see page 5711) the algorithm used by AnsiResemblesText (see page 21970) to determine when two strings are similar. It is the same as AnsiResemblesProc (see page 21995).

155.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AnsiContainsStr (see page 21963)	Indicates whether one string is a (case-sensitive) substring of another.
	AnsiContainsText (see page 21964)	Indicates whether one string is a (case-insensitive) substring of another.
	AnsiEndsStr (see page 21964)	Indicates whether one string is a (case-sensitive) suffix of another.
	AnsiEndsText (see page 21965)	Indicates whether one string is a (case-insensitive) suffix of another.
	AnsiIndexStr (see page 21965)	Provides the index of a specified string in an array of strings.
	AnsiIndexText (see page 21966)	Provides the index of a specified string in an array of strings.
	AnsiLeftStr (see page 21966)	Returns the substring of a specified length that appears at the start of a string.
	AnsiMatchStr (see page 21967)	Indicates whether an array of strings contains an exact match to a specified string.
	AnsiMatchText (see page 21967)	Indicates whether an array of strings contains a case-insensitive match to a specified string.
	AnsiMidStr (see page 21968)	Returns the substring of a specified length that appears at a specified position in a string.
	AnsiReplaceStr (see page 21969)	Replaces all occurrences of a substring with another string.
	AnsiReplaceText (see page 21969)	Replaces all case-insensitive matches of a substring with another string.
	AnsiResemblesText (see page 21970)	Indicates whether two strings are similar.
	AnsiReverseString (see page 21970)	Returns reversed string.
	AnsiRightStr (see page 21971)	Returns the substring of a specified length that appears at the end of a string.
	AnsiStartsStr (see page 21971)	Indicates whether one string is a (case-sensitive) prefix of another.
	AnsiStartsText (see page 21972)	Indicates whether one string is a (case-insensitive) prefix of another.
	ContainsStr (see page 21972)	Performs a case sensitive search for substring ASubText in string AText. Returns true if one or more instances of ASubText exist in AText.
	ContainsText (see page 21973)	Performs a case insensitive search for substring ASubText in string AText. Returns true if one or more instances of ASubText exist in AText.
	DecodeSoundexInt (see page 21973)	Converts an integer representation of a Soundex (see page 21986) encoding into the corresponding phonetic string.
	DecodeSoundexWord (see page 21974)	Converts a Word representation of a Soundex (see page 21986) encoding into the corresponding phonetic string.
	DupeString (see page 21974)	Returns the concatenation of a string with itself a specified number of repeats.
	EndsStr (see page 21975)	Determines if string AText ends with substring ASubText using a case sensitive string comparison. Returns true if AText ends with ASubText.
	EndsText (see page 21975)	Determines if string AText ends with substring ASubText using a case insensitive string comparison. Returns true if AText ends with ASubText.
	IfThen (see page 21976)	Conditionally returns one of two specified values.
	IndexStr (see page 21976)	IndexStr determines if any of the strings in the array AValues match the string specified by AText using a case sensitive comparison. It returns the zero-based index of the first match in the array, or -1 if no match is found.
	IndexText (see page 21977)	IndexText determines if any of the strings in the array AValues match the string specified by AText using a case insensitive comparison. It returns the zero-based index of the first match in the array, or -1 if no match is found.
	LeftBStr (see page 21977)	Returns the substring of a specified number of bytes that appears at the start of a string.
	LeftStr (see page 21978)	Returns the substring of a specified length that appears at the start of a string.
	LeftStr (see page 21978)	Returns the substring of a specified length that appears at the start of a string.
	MatchStr (see page 21979)	MatchStr determines if any of the strings in the array AValues match the string specified by AText using a case sensitive comparison. It returns true if at least one of the strings in the array match, or false if none of the strings match.
	MatchText (see page 21979)	MatchText determines if any of the strings in the array AValues match the string specified by AText using a case insensitive comparison. It returns true if at least one of the strings in the array match, or false if none of the strings match.

	MidBStr (see page 21980)	Returns the substring of a specified number of bytes that appears at a specified position in a string.
	MidStr (see page 21980)	Returns the substring of a specified length that appears at a specified position in a string.
	MidStr (see page 21980)	Returns the substring of a specified length that appears at a specified position in a string.
	PosEx (see page 21981)	Returns the index value of a substring.
	RandomFrom (see page 21981)	Returns a randomly selected element from an array.
	ReplaceStr (see page 21982)	ReplaceStr replaces all instances of string AFromText to string AToText in the source string AText and returns this value as the result. The replacement is case sensitive.
	ReplaceText (see page 21982)	ReplaceText replaces all instances of string AFromText to string AToText in the source string AText and returns this value as the result. The replacement is case insensitive.
	ResemblesText (see page 21983)	ResemblesText uses a Soundex (see page 21986) method to determine if string AText and string AOther are similar. The result is True if the two strings a similar according to the Soundex (see page 21986) method, and false otherwise.
	ReverseString (see page 21983)	Returns the reverse of a specified string.
	RightBStr (see page 21984)	Returns the substring of a specified number of bytes that appears at the end of a string.
	RightStr (see page 21984)	Returns the substring of a specified length that appears at the end of a string.
	RightStr (see page 21984)	Returns the substring of a specified length that appears at the end of a string.
	SearchBuf (see page 21985)	Locates a substring within a text buffer.
	Soundex (see page 21986)	Converts a string into its Soundex representation.
	SoundexCompare (see page 21986)	Compares the Soundex (see page 21986) representations of two strings.
	SoundexInt (see page 21987)	Converts a string into an integer that represents its phonetic value.
	SoundexProc (see page 21987)	Indicates whether two strings are similar.
	SoundexSimilar (see page 21988)	Indicates whether two strings are similar.
	SoundexWord (see page 21989)	Converts a string into a Word that represents its phonetic value.
	StartsStr (see page 21989)	Enter a short sentence, starting with a verb, that states the basic purpose of the property.
	StartsText (see page 21990)	StartsText determines if the substring ASubText begins the string AText using a case insensitive algorithm. If ASubText matches the begining of AText, the result is true, otherwise it is false.
	StuffString (see page 21990)	Inserts a substring into a specified position of a string, replacing the current characters.

155.1.1 StrUtils.AnsiContainsStr Function

Indicates whether one string is a (case-sensitive) substring of another.

Pascal

```
function AnsiContainsStr(const AText: string; const ASubText: string): Boolean;
```

C++

```
Boolean AnsiContainsStr(const AnsiString AText, const AnsiString ASubText);
```

File

StrUtils

Description

AnsiContainsStr returns true if the string specified by ASubText appears as a substring of the string specified by AText. Comparison is performed case-sensitively.

See Also

AnsiResemblesText (see page 21970)

AnsiStartsStr (see page 21971)

AnsiEndsStr (see page 21964)

AnsiCompareStr

Pos

StringReplace

AnsiContainsText (see page 21964)

155.1.2 StrUtils.AnsiContainsText Function

Indicates whether one string is a (case-insensitive) substring of another.

Pascal

```
function AnsiContainsText(const AText: string; const ASubText: string): Boolean;
```

C++

```
Boolean AnsiContainsText(const AnsiString AText, const AnsiString ASubText);
```

File

StrUtils

Description

AnsiContainsText returns true if the string specified by ASubText appears as a substring of the string specified by AText. Comparison is performed case-insensitively and based on the current system locale.

See Also

AnsiResemblesText (see page 21970)

AnsiStartsText (see page 21972)

AnsiEndsText (see page 21965)

AnsiCompareText

Pos

StringReplace

AnsiContainsStr (see page 21963)

155.1.3 StrUtils.AnsiEndsStr Function

Indicates whether one string is a (case-sensitive) suffix of another.

Pascal

```
function AnsiEndsStr(const ASubText: string; const AText: string): Boolean;
```

C++

```
Boolean AnsiEndsStr(const AnsiString ASubText, const AnsiString AText);
```

File

StrUtils

Description

AnsiEndsStr returns true if the string specified by ASubText appears as the trailing part of the string specified by AText. Comparison is performed case-sensitively.

See Also

AnsiStartsStr (see page 21971)

AnsiContainsStr (see page 21963)

Pos

RightStr (see page 21984)

AnsiEndsText (see page 21965)

155.1.4 StrUtils.AnsiEndsText Function

Indicates whether one string is a (case-insensitive) suffix of another.

Pascal

```
function AnsiEndsText(const ASubText: string; const AText: string): Boolean;
```

C++

```
Boolean AnsiEndsText(const AnsiString ASubText, const AnsiString AText);
```

File

StrUtils

Description

AnsiEndsText returns true if the string specified by ASubText appears as the trailing part of the string specified by AText. Comparison is performed case-insensitively and based on the current system locale.

See Also

AnsiStartsText (see page 21972)

AnsiContainsText (see page 21964)

Pos

RightStr (see page 21984)

155.1.5 StrUtils.AnsiIndexStr Function

Provides the index of a specified string in an array of strings.

Pascal

```
function AnsiIndexStr(const AText: string; const AValues: array of string): Integer;
```

C++

```
int AnsiIndexStr(const AnsiString AText, const array of string AValues);
```

File

StrUtils

Description

AnsiIndexStr returns the 0-offset index of the first string it finds in AValues that matches AText case-sensitively. If the string specified by AText does not have an exact match in AValues, AnsiIndexStr returns –1.

See Also

AnsiMatchStr (see page 21967)

RandomFrom (see page 21981)

AnsiIndexText (see page 21966)

155.1.6 StrUtils.AnsiIndexText Function

Provides the index of a specified string in an array of strings.

Pascal

```
function AnsiIndexText(const AText: string; const AValues: array of string): Integer;
```

C++

```
int AnsiIndexText(const AnsiString AText, const array of string AValues);
```

File

StrUtils

Description

AnsiIndexText returns the 0-offset index of the first string it finds in AValues that matches AText case-insensitively. If the string specified by AText does not have a (possibly case-insensitive) match in AValues, AnsiIndexText returns –1. Comparisons are based on the current system locale.

See Also

AnsiMatchText (see page 21967)

RandomFrom (see page 21981)

AnsiIndexStr (see page 21965)

155.1.7 StrUtils.AnsiLeftStr Function

Returns the substring of a specified length that appears at the start of a string.

Pascal

```
function AnsiLeftStr(const AText: AnsiString; const ACount: Integer): AnsiString;
```

C++

```
AnsiString AnsiLeftStr(const AnsiString AText, const int ACount);
```

File

StrUtils

Description

AnsiLeftStr returns the leading characters of AText up to a length of ACount characters.

See Also

Copy

TrimLeft

AnsiStartsText (see page 21972)

AnsiStartsStr (see page 21971)

LeftStr (see page 21978)

LeftBStr (see page 21977)

155.1.8 StrUtils.AnsiMatchStr Function

Indicates whether an array of strings contains an exact match to a specified string.

Pascal

```
function AnsiMatchStr(const AText: string; const AValues: array of string): Boolean;
```

C++

```
Boolean AnsiMatchStr(const AnsiString AText, const array of string AValues);
```

File

StrUtils

Description

Call AnsiMatchStr to determine whether the string specified by AText matches one of the strings in AValues. Comparisons are case-sensitive.

See Also

AnsiIndexStr (see page 21965)

AnsiSameStr

AnsiMatchText (see page 21967)

155.1.9 StrUtils.AnsiMatchText Function

Indicates whether an array of strings contains a case-insensitive match to a specified string.

Pascal

```
function AnsiMatchText(const AText: string; const AValues: array of string): Boolean;
```

C++

```
Boolean AnsiMatchText(const AnsiString AText, const array of string AValues);
```

File

StrUtils

Description

Call AnsiMatchText to determine whether the string specified by AText matches one of the strings in AValues. Comparisons are case-insensitive and use the current system locale.

AText is the string to locate.

AValues is the array of strings to search.

Note: AValues_Size is the index of the last element in AValues (one less than the number of elements).

AnsiMatchText returns true if AText (or a string that matches except for case) appears in AValues, false if it does not appear in the array.

See Also

AnsiIndexText (see page 21966)

AnsiSameText

ANsiMatchStr

155.1.10 StrUtils.AnsiMidStr Function

Returns the substring of a specified length that appears at a specified position in a string.

Pascal

```
function AnsiMidStr(const AText: AnsiString; const AStart: Integer; const ACount: Integer):
AnsiString;
```

C++

```
AnsiString AnsiMidStr(const AnsiString AText, const int AStart, const int ACount);
```

File

StrUtils

Description

AnsiMidStr returns a substring Count characters at AText[AStart].

If AStart is larger than the length of AText, AnsiMidStr returns an empty string.

If Count specifies more characters than are available, only the characters from AText[AStart] to the end of AText are returned.

See Also

Copy

LeftStr (see page 21978)

RightStr (see page 21984)

MidStr (see page 21980)

MidBStr (see page 21980)

155.1.11 StrUtils.AnsiReplaceStr Function

Replaces all occurrences of a substring with another string.

Pascal

```
function AnsiReplaceStr(const AText: string; const AFromText: string; const AToText: string):
string;
```

C++

```
AnsiString AnsiReplaceStr(const AnsiString AText, const AnsiString AFromText, const AnsiString
AToText);
```

File

StrUtils

Description

Call AnsiReplaceStr to scan the string specified by AText and replace all occurrences of the string specified by AFromText with the string specified by AToText. Occurrences of AFromText are identified by a case-sensitive comparison. The identification of substrings is not recursive: if the substitution of AToText results in a new match for AFromText, that match is not replaced.

See Also

AnsiContainsStr (see page 21963)

AnsiReplaceText (see page 21969)

ANsiMatchStr

StringReplace

SearchBuf (see page 21985)

StuffString (see page 21990)

155.1.12 StrUtils.AnsiReplaceText Function

Replaces all case-insensitive matches of a substring with another string.

Pascal

```
function AnsiReplaceText(const AText: string; const AFromText: string; const AToText: string):
string;
```

C++

```
AnsiString AnsiReplaceText(const AnsiString AText, const AnsiString AFromText, const
AnsiString AToText);
```

File

StrUtils

Description

Call AnsiReplaceText to scan the string specified by AText and replace all occurrences of the string specified by AFromText with the string specified by AToText. Occurrences of AFromText are identified by a case-insensitive comparison that uses the current system locale. The identification of substrings is not recursive: if the substitution of AToText results in a new match for

AFromText, that match is not replaced.

See Also

AnsiContainsText (see page 21964)

AnsiReplaceStr (see page 21969)

StringReplace

SearchBuf (see page 21985)

StuffString (see page 21990)

155.1.13 StrUtils.AnsiResemblesText Function

Indicates whether two strings are similar.

Pascal

```
function AnsiResemblesText(const AText: string; const AOther: string): Boolean;
```

C++

```
Boolean AnsiResemblesText(const AnsiString AText, const AnsiString AOther);
```

File

StrUtils

Description

AnsiResemblesText returns true if the string specified by AText is "similar" to the string specified by AOther, or false if the two strings are not sufficiently similar.

The algorithm that controls whether AnsiResemblesText considers two strings similar is specified by the AnsiResemblesProc (see page 21995) variable. By default, AnsiResemblesProc (see page 21995) is set to SoundExProc, which converts both strings to a phonetic representation using a SoundEx algorithm and compares up to 4 characters of the resulting phonetic representation. By changing the function assigned to AnsiResemblesProc (see page 21995), you can change the comparison made by AnsiResemblesText.

See Also

AnsiResemblesProc (see page 21995)

SoundExProc

155.1.14 StrUtils.AnsiReverseString Function

Returns reversed string.

Pascal

```
function AnsiReverseString(const AText: AnsiString): AnsiString;
```

C++

```
AnsiString AnsiReverseString(const AnsiString AText);
```

File

StrUtils

Description

AnsiReverseString returns AText with its characters reversed.

155.1.15 StrUtils.AnsiRightStr Function

Returns the substring of a specified length that appears at the end of a string.

Pascal

```
function AnsiRightStr(const AText: AnsiString; const ACount: Integer): AnsiString;
```

C++

```
AnsiString AnsiRightStr(const AnsiString AText, const int ACount);
```

File

StrUtils

Description

AnsiRightStr returns the trailing characters of AText up to a length of ACount characters. Thus, for example, if AText is the string "Programmer" and ACount is 7, AnsiRightStr returns the string "grammer".

See Also

Copy

TrimRight

AnsiEndsText (see page 21965)

AnsiEndsStr (see page 21964)

RightStr (see page 21984)

RightBStr (see page 21984)

155.1.16 StrUtils.AnsiStartsStr Function

Indicates whether one string is a (case-sensitive) prefix of another.

Pascal

```
function AnsiStartsStr(const ASubText: string; const AText: string): Boolean;
```

C++

```
Boolean AnsiStartsStr(const AnsiString ASubText, const AnsiString AText);
```

File

StrUtils

Description

AnsiStartsStr returns true if the string specified by ASubText appears as the leading part of the string specified by AText.

Comparison is performed case-sensitively.

See Also

AnsiEndsStr (see page 21964)

AnsiContainsStr (see page 21963)

Pos

LeftStr (see page 21978)

AnsiStartsText (see page 21972)

155.1.17 StrUtils.AnsiStartsText Function

Indicates whether one string is a (case-insensitive) prefix of another.

Pascal

```
function AnsiStartsText(const ASubText: string; const AText: string): Boolean;
```

C++

```
Boolean AnsiStartsText(const AnsiString ASubText, const AnsiString AText);
```

File

StrUtils

Description

AnsiStartsText returns true if the string specified by ASubText appears as the leading part of the string specified by AText. Comparison is performed case-insensitively and uses the current system locale.

See Also

AnsiEndsText (see page 21965)

AnsiContainsText (see page 21964)

Pos

LeftStr (see page 21978)

AnsiStartsStr (see page 21971)

155.1.18 StrUtils.ContainsStr Function

Performs a case sensitive search for substring ASubText in string AText. Returns true if one or more instances of ASubText exist in AText.

Pascal

```
function ContainsStr(const AText: string; const ASubText: string): Boolean;
```

C++

```
Boolean ContainsStr(const AnsiString AText, const AnsiString ASubText);
```

File

StrUtils

Description

For a case insensitive search, use the ContainsText (see page 21973) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

ContainsText (see page 21973)

155.1.19 StrUtils.ContainsText Function

Performs a case insensitive search for substring ASubText in string AText. Returns true if one or more instances of ASubText exist in AText.

Pascal

```
function ContainsText(const AText: string; const ASubText: string): Boolean;
```

C++

```
Boolean ContainsText(const AnsiString AText, const AnsiString ASubText);
```

File

StrUtils

Description

For a case sensitive search, use the ContainsStr (see page 21972) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

ContainsStr (see page 21972)

155.1.20 StrUtils.DecodeSoundexInt Function

Converts an integer representation of a Soundex (see page 21986) encoding into the corresponding phonetic string.

Pascal

```
function DecodeSoundexInt(AValue: Integer): string;
```

C++

```
AnsiString DecodeSoundexInt(int AValue);
```

File

StrUtils

Description

Call DecodeSoundexInt to convert the integer representation of a string returned by the SoundexInt (see page 21987) function into the corresponding Soundex (see page 21986) string representation.

A Soundex (see page 21986) representation of a string is a phonetic encoding where each character of the Soundex (see page 21986) representation represents a family of similar phonemes.

See Also

SoundexInt (see page 21987)

DecodeSoundexWord (see page 21974)

Soundex (see page 21986)

155.1.21 StrUtils.DecodeSoundexWord Function

Converts a Word representation of a Soundex (see page 21986) encoding into the corresponding phonetic string.

Pascal

```
function DecodeSoundexWord(AValue: Word): string;
```

C++

```
AnsiString DecodeSoundexWord(Word AValue);
```

File

StrUtils

Description

Call DecodeSoundexWord to convert the unsigned 16-bit representation of a string returned by the SoundexWord (see page 21989) function into the corresponding Soundex (see page 21986) string representation.

A Soundex (see page 21986) representation of a string is a phonetic encoding where each character of the Soundex (see page 21986) representation represents a family of similar phonemes.

See Also

SoundexWord (see page 21989)

DecodeSoundexInt (see page 21973)

Soundex (see page 21986)

155.1.22 StrUtils.DupeString Function

Returns the concatenation of a string with itself a specified number of repeats.

Pascal

```
function DupeString(const AText: string; ACount: Integer): string;
```

C++

```
AnsiString DupeString(const AnsiString AText, int ACount);
```

File

StrUtils

Description

DupeString returns a string containing ACount repeats of the string specified by AText. For example,

S := DupeString('Ha', 5);

sets S to the string 'HaHaHaHaHa'.

See Also

StringOfChar

StringReplace

155.1.23 StrUtils.EndsStr Function

Determines if string AText ends with substring ASubText using a case sensitive string comparison. Returns true if AText ends with ASubText.

Pascal

```
function EndsStr(const ASubText: string; const AText: string): Boolean;
```

C++

```
Boolean EndsStr(const AnsiString ASubText, const AnsiString AText);
```

File

StrUtils

Description

For a case insensitive comparison, use the EndsText (see page 21975) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

EndsText (see page 21975)

155.1.24 StrUtils.EndsText Function

Determines if string AText ends with substring ASubText using a case insensitive string comparison. Returns true if AText ends with ASubText.

Pascal

```
function EndsText(const ASubText: string; const AText: string): Boolean;
```

C++

```
Boolean EndsText(const AnsiString ASubText, const AnsiString AText);
```

File

StrUtils

Description

For a case sensitive comparison, use the EndsStr (see page 21975) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

EndsStr (see page 21975)

155.1.25 StrUtils.IfThen Function

Conditionally returns one of two specified values.

Pascal

```
function IfThen(AValue: Boolean; const ATrue: string; AFalse: string = ''): string; overload;
```

C++

```
AnsiString IfThen(Boolean AValue, const AnsiString ATrue, AnsiString AFalse = '');
```

File

StrUtils

Description

IfThen checks the expression passed as AValue and returns ATrue if it evaluates to true, or AFalse if it evaluates to false. In Delphi, if the AFalse parameter is omitted, IfThen returns 0 or an empty string when AValue evaluates to False.

155.1.26 StrUtils.IndexStr Function

IndexStr determines if any of the strings in the array AValues match the string specified by AText using a case sensitive comparison. It returns the zero-based index of the first match in the array, or -1 if no match is found.

Pascal

```
function IndexStr(const AText: string; const AValues: array of string): Integer;
```

C++

```
int IndexStr(const AnsiString AText, const array of string AValues);
```

File

StrUtils

Description

For a case insensitive comparison, use the IndexText (see page 21977) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

IndexText (see page 21977)

155.1.27 StrUtils.IndexText Function

IndexText determines if any of the strings in the array AValues match the string specified by AText using a case insensitive comparison. It returns the zero-based index of the first match in the array, or -1 if no match is found.

Pascal

```
function IndexText(const AText: string; const AValues: array of string): Integer;
```

C++

```
int IndexText(const AnsiString AText, const array of string AValues);
```

File

StrUtils

Description

For a case sensitive comparison, use the IndexStr (see page 21976) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

IndexStr (see page 21976)

155.1.28 StrUtils.LeftBStr Function

Returns the substring of a specified number of bytes that appears at the start of a string.

Pascal

```
function LeftBStr(const AText: AnsiString; const AByteCount: Integer): AnsiString;
```

C++

```
AnsiString LeftBStr(const AnsiString AText, const int AByteCount);
```

File

StrUtils

Description

LeftBStr returns the leading bytes of AText up to a length of AByteCount bytes.

Note: LeftBStr assumes all characters are one byte long, and does not behave correctly if AText contains multi-byte characters. To handle all strings correctly, use LeftStr (see page 21978).

See Also

Copy

TrimLeft

AnsiStartsText (see page 21972)

AnsiStartsStr (see page 21971)

AnsiLeftStr (see page 21966)

LeftStr (see page 21978)

155.1.29 StrUtils.LeftStr Function

Returns the substring of a specified length that appears at the start of a string.

Pascal

```
function LeftStr(const AText: AnsiString; const ACount: Integer): AnsiString; overload;
function LeftStr(const AText: WideString; const ACount: Integer): WideString; overload;
```

C++

```
AnsiString LeftStr(const AnsiString AText, const int ACount);
BSTR LeftStr(const BSTR AText, const int ACount);
```

File

StrUtils

Description

LeftStr returns the leading characters of AText up to a length of ACount characters.

Note: If AText is an AnsiString and the current locale uses multi-byte characters, LeftStr may return more than ACount bytes. To return a specific number of bytes, use LeftBStr (see page 21977).

See Also

Copy

TrimLeft

AnsiStartsText (see page 21972)

AnsiStartsStr (see page 21971)

AnsiLeftStr (see page 21966)

LeftBStr (see page 21977)

155.1.30 StrUtils.MatchStr Function

MatchStr determines if any of the strings in the array AValues match the string specified by AText using a case sensitive comparison. It returns true if at least one of the strings in the array match, or false if none of the strings match.

Pascal

```
function MatchStr(const AText: string; const AValues: array of string): Boolean;
```

C++

```
Boolean MatchStr(const AnsiString AText, const array of string AValues);
```

File

StrUtils

Description

For a case insensitive match, use the MatchText (see page 21979) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

MatchText (see page 21979)

155.1.31 StrUtils.MatchText Function

MatchText determines if any of the strings in the array AValues match the string specified by AText using a case insensitive comparison. It returns true if at least one of the strings in the array match, or false if none of the strings match.

Pascal

```
function MatchText(const AText: string; const AValues: array of string): Boolean;
```

C++

```
Boolean MatchText(const AnsiString AText, const array of string AValues);
```

File

StrUtils

Description

For a case sensitive comparison, use the MatchStr (see page 21979) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

MatchStr (see page 21979)

155.1.32 StrUtils.MidBStr Function

Returns the substring of a specified number of bytes that appears at a specified position in a string.

Pascal

```
function MidBStr(const AText: AnsiString; const AByteStart: Integer; const AByteCount:
Integer): AnsiString;
```

C++

```
AnsiString MidBStr(const AnsiString AText, const int AByteStart, const int AByteCount);
```

File

StrUtils

Description

MidBStr returns a substring AByteCount bytes beginning at AByteStart bytes.

If AByteStart is larger than the length of AText in bytes, MidBStr returns an empty string.

If AByteCount specifies more bytes than are available, only the available bytes are returned.

Note: MidBStr assumes all characters are one byte long, and does not behave correctly if AText contains multi-byte characters. To handle all strings correctly, use MidStr (see page 21980).

See Also

Copy

LeftStr (see page 21978)

RightStr (see page 21984)

AnsiMidStr (see page 21968)

MidStr (see page 21980)

155.1.33 StrUtils.MidStr Function

Returns the substring of a specified length that appears at a specified position in a string.

Pascal

```
function MidStr(const AText: AnsiString; const AStart: Integer; const ACount: Integer):
AnsiString; overload;
function MidStr(const AText: WideString; const AStart: Integer; const ACount: Integer):
WideString; overload;
```

C++

```
AnsiString MidStr(const AnsiString AText, const int AStart, const int ACount);
BSTR MidStr(const BSTR AText, const int AStart, const int ACount);
```

File

StrUtils

Description

MidStr returns a substring Count characters at AText[AStart].

If AStart is larger than the length of AText, MidStr returns an empty string.

If Count specifies more characters than are available, only the characters from AText[AStart] to the end of AText are returned.

Note: If AText is an AnsiString and the current locale uses multi-byte characters, MidStr may return more than ACount bytes. To return a specific number of bytes, use MidBStr (see page 21980).

See Also

Copy

LeftStr (see page 21978)

RightStr (see page 21984)

AnsiMidStr (see page 21968)

MidBStr (see page 21980)

155.1.34 StrUtils.PosEx Function

Returns the index value of a substring.

Pascal

```
function PosEx(const SubStr: string; const S: string; Offset: Integer = 1): Integer;
```

C++

```
int PosEx(const AnsiString SubStr, const AnsiString S, int Offset = 1);
```

File

StrUtils

Description

PosEx returns the index of SubStr in S, beginning the search at Offset. If Offset is 1 (default), PosEx is equivalent to Pos.

PosEx returns 0 if SubStr is not found, if Offset is greater than the length of S, or if Offset is less than 1.

See Also

Pos

155.1.35 StrUtils.RandomFrom Function

Returns a randomly selected element from an array.

Pascal

```
function RandomFrom(const AValues: array of string): string; overload;
```

C++

```
AnsiString RandomFrom(const array of string AValues);
```

File

StrUtils

Description

RandomFrom returns a randomly selected element from the array passed as the AValues argument. In C++, AValues_Size is the index of the last array element (one less than the number of elements.)

To initialize the random number generator, add a single call Randomize or assign a value to the RandSeed variable before making any calls to RandomFrom.

See Also

Randomize

RandSeed

155.1.36 StrUtils.ReplaceStr Function

ReplaceStr replaces all instances of string AFromText to string AToText in the source string AText and returns this value as the result. The replacement is case sensitive.

Pascal

```
function ReplaceStr(const AText: string; const AFromText: string; const AToText: string):
string;
```

C++

```
AnsiString ReplaceStr(const AnsiString AText, const AnsiString AFromText, const AnsiString
AToText);
```

File

StrUtils

Description

For a case insensitive replacement, use the ReplaceText (see page 21982) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

ReplaceText (see page 21982)

155.1.37 StrUtils.ReplaceText Function

ReplaceText replaces all instances of string AFromText to string AToText in the source string AText and returns this value as the result. The replacement is case insensitive.

Pascal

```
function ReplaceText(const AText: string; const AFromText: string; const AToText: string):
string;
```

C++

```
AnsiString ReplaceText(const AnsiString AText, const AnsiString AFromText, const AnsiString AToText);
```

File

StrUtils

Description

For a case sensitive replacement, use the ReplaceStr (see page 21982) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

ReplaceStr (see page 21982)

155.1.38 StrUtils.ResemblesText Function

ResemblesText uses a Soundex (see page 21986) method to determine if string AText and string AOther are similar. The result is True if the two strings a similar according to the Soundex (see page 21986) method, and false otherwise.

Pascal

```
function ResemblesText(const AText: string; const AOther: string): Boolean;
```

C++

```
Boolean ResemblesText(const AnsiString AText, const AnsiString AOther);
```

File

StrUtils

See Also

Commonly Used Long String Routines

String Types

String Constants

AnsiResemblesText (see page 21970)

155.1.39 StrUtils.ReverseString Function

Returns the reverse of a specified string.

Pascal

```
function ReverseString(const AText: string): string;
```

C++

```
AnsiString ReverseString(const AnsiString AText);
```

File

StrUtils

Description

ReverseString returns the string specified by AText with the characters in reverse order.

See Also

DupeString (see page 21974)

155.1.40 StrUtils.RightBStr Function

Returns the substring of a specified number of bytes that appears at the end of a string.

Pascal

```
function RightBStr(const AText: AnsiString; const AByteCount: Integer): AnsiString;
```

C++

```
AnsiString RightBStr(const AnsiString AText, const int AByteCount);
```

File

StrUtils

Description

RightBStr returns the trailing bytes of AText up to a length of AByteCount bytes.

Note: RightBStr assumes all characters are one byte long, and does not behave correctly if AText contains multi-byte characters. To handle all strings correctly, use RightStr (see page 21984).

See Also

Copy

TrimRight

AnsiEndsText (see page 21965)

AnsiEndsStr (see page 21964)

AnsiRightStr (see page 21971)

155.1.41 StrUtils.RightStr Function

Returns the substring of a specified length that appears at the end of a string.

Pascal

```
function RightStr(const AText: AnsiString; const ACount: Integer): AnsiString; overload;
function RightStr(const AText: WideString; const ACount: Integer): WideString; overload;
```

C++

```
AnsiString RightStr(const AnsiString AText, const int ACount);
BSTR RightStr(const BSTR AText, const int ACount);
```

File

StrUtils

Description

RightStr returns the trailing characters of AText up to a length of ACount characters. Thus, for example, if AText is the string "Programmer" and ACount is 7, RightStr returns the string "grammer".

Note: If AText is an AnsiString and the current locale uses multi-byte characters, RightStr may return more than ACount bytes. To return a specific number of bytes, use RightBStr (see page 21984).

See Also

Copy

TrimRight

AnsiEndsText (see page 21965)

AnsiEndsStr (see page 21964)

AnsiRightStr (see page 21971)

155.1.42 StrUtils.SearchBuf Function

Locates a substring within a text buffer.

Pascal

```
function SearchBuf(Buf: PChar; BufLen: Integer; SelStart: Integer; SelLength: Integer;
SearchString: String; Options: TStringSearchOptions = [soDown]): PChar;
```

C++

```
const char * SearchBuf(const char * Buf, int BufLen, int SelStart, int SelLength, AnsiString
SearchString, TStringSearchOptions Options = [soDown]);
```

File

StrUtils

Description

Call SearchBuf to search for a specified search string within a text buffer.

Buf is the text buffer to search.

BufLen is the length, in bytes, of Buf.

SelStart is the first character of the search when Options indicates a backward search (does not include soDown). The first character in Buf has position 0.

SelLength is the number of characters after SelStart that the search begins when Options indicates a forward search (includes soDown).

SearchString is the string to find in Buf.

Options determines whether the search runs forward (soDown) or backward from SelStart or SelStart+SelLength, whether the search is case sensitive (soMatchCase), and whether the matching string must be a whole word (soWholeWord).

If SearchBuf finds a match, it returns a pointer to the first character of the matching string in Buf. If it does not find a match, SearchBuf returns nil (Delphi) or NULL (C++).

See Also

TFindDialog

155.1.43 StrUtils.Soundex Function

Converts a string into its Soundex representation.

Pascal

```
function Soundex(const AText: string; ALength: TSoundexLength = 4): string;
```

C++

```
AnsiString Soundex(const AnsiString AText, TSoundexLength ALength = 4);
```

File

StrUtils

Description

Soundex converts the string specified by AText into a phonetic representation where each character of the resulting string represents one of six families of similar phonemes. It returns up to the first ALength Soundex codes in the resulting string representation.

See Also

SoundexWord (see page 21989)

SoundexInt (see page 21987)

SoundexSimilar (see page 21988)

SoundexCompare (see page 21986)

SoundexProc (see page 21987)

155.1.44 StrUtils.SoundexCompare Function

Compares the Soundex (see page 21986) representations of two strings.

Pascal

```
function SoundexCompare(const AText: string; const AOther: string; ALength: TSoundexLength =
4): Integer;
```

C++

```
int SoundexCompare(const AnsiString AText, const AnsiString AOther, TSoundexLength ALength =
4);
```

File

StrUtils

Description

SoundexCompare uses a Soundex (see page 21986) algorithm to convert the strings specified by AText and AOther into phonetic representations where each character of the resulting string represents one of six families of similar phonemes. It then compares the first ALength characters of the resulting representations by calling AnsiCompareStr.

SoundexCompare returns 0 if the first ALength characters of the phonetic representations are identical, a value less then 0 if the representation of AText is less than the representation of AOther, and a value greater than 0 if the representation of AText is greater than the representation of AOther.

See Also

SoundexSimilar (see page 21988)

SoundexWord (see page 21989)

SoundexInt (see page 21987)

AnsiCompareStr

Soundex (see page 21986)

SoundexProc (see page 21987)

155.1.45 StrUtils.SoundexInt Function

Converts a string into an integer that represents its phonetic value.

Pascal

```
function SoundexInt(const AText: string; ALength: TSoundexIntLength = 4): Integer;
```

C++

```
int SoundexInt(const AnsiString AText, TSoundexIntLength ALength = 4);
```

File

StrUtils

Description

SoundexInt uses a Soundex (see page 21986) algorithm to convert the string specified by AText into a phonetic representation where each character of the resulting string represents one of six families of similar phonemes. It then returns an integer that uniquely represents the first ALength Soundex (see page 21986) codes in that string representation.

Note: The TSoundexIntLength (see page 21993) type represents an integer between 1 and 8. This is because the size of the integer return value limits the number of Soundex (see page 21986) codes that can be used in the final representation.

See Also

SoundexSimilar (see page 21988)

SoundexWord (see page 21989)

DecodeSoundexInt (see page 21973)

SoundexCompare (see page 21986)

Soundex (see page 21986)

SoundexProc (see page 21987)

155.1.46 StrUtils.SoundexProc Function

Indicates whether two strings are similar.

Pascal

```
function SoundexProc(const AText: string; const AOther: string): Boolean;
```

C++

```
Boolean SoundexProc(const AnsiString AText, const AnsiString AOther);
```

File

StrUtils

Description

SoundexProc returns true if the string specified by AText is "similar" to the string specified by AOther, or false if the two strings are not sufficiently similar. SoundexProc is the default value of the AnsiResemblesProc (see page 21995) variable, which provides the underlying implementation of AnsiResemblesText (see page 21970).

SoundexProc uses a Soundex (see page 21986) algorithm to convert both strings into a phonetic representation where each character of the resulting string represents one of six families of similar phonemes. It then compares up to 4 digits in the resulting representation, returning true if they are all the same and false if they are not.

Note: SoundexProc is the same as the SoundexSimilar (see page 21988) function using an ALength value of 4.

See Also

AnsiResemblesProc (see page 21995)

SoundexSimilar (see page 21988)

SoundexCompare (see page 21986)

SoundexWord (see page 21989)

SOundexInt

Soundex (see page 21986)

AnsiResemblesText (see page 21970)

155.1.47 StrUtils.SoundexSimilar Function

Indicates whether two strings are similar.

Pascal

```
function SoundexSimilar(const AText: string; const AOther: string; ALength: TSoundexLength =
4): Boolean;
```

C++

```
Boolean SoundexSimilar(const AnsiString AText, const AnsiString AOther, TSoundexLength ALength
= 4);
```

File

StrUtils

Description

SoundexSimilar returns true if the string specified by AText is "similar" to the string specified by AOther, or false if the two strings are not sufficiently similar.

SoundexSimilar uses a Soundex (see page 21986) algorithm to convert both strings into a phonetic representation where each character of the resulting string represents one of six families of similar phonemes. It then compares up to ALength digits in the

resulting representation, returning true if they are all the same and false if they are not.

See Also

SOundexCompare

SoundexWord (see page 21989)

SoundexInt (see page 21987)

Soundex (see page 21986)

SoundexProc (see page 21987)

155.1.48 StrUtils.SoundexWord Function

Converts a string into a Word that represents its phonetic value.

Pascal

```
function SoundexWord(const AText: string): Word;
```

C++

```
Word SoundexWord(const AnsiString AText);
```

File

StrUtils

Description

SoundexWord uses a Soundex (see page 21986) algorithm to convert the string specified by AText into a phonetic representation where each character of the resulting string represents one of six families of similar phonemes. It then returns an unsigned 16-bit value that uniquely represents the first 4 Soundex (see page 21986) codes in that string representation.

See Also

SoundexSimilar (see page 21988)

SoundexInt (see page 21987)

DecodeSoundexWord (see page 21974)

SoundexCompare (see page 21986)

Soundex (see page 21986)

SoundexProc (see page 21987)

155.1.49 StrUtils.StartsStr Function

Enter a short sentence, starting with a verb, that states the basic purpose of the property.

Pascal

```
function StartsStr(const ASubText: string; const AText: string): Boolean;
```

C++

```
Boolean StartsStr(const AnsiString ASubText, const AnsiString AText);
```

File

StrUtils

Description

For a case insensitive comparison, use the StartsText (see page 21990) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

StartsText (see page 21990)

155.1.50 StrUtils.StartsText Function

StartsText determines if the substring ASubText begins the string AText using a case insensitive algorithm. If ASubText matches the begining of AText, the result is true, otherwise it is false.

Pascal

```
function StartsText(const ASubText: string; const AText: string): Boolean;
```

C++

```
Boolean StartsText(const AnsiString ASubText, const AnsiString AText);
```

File

StrUtils

Description

For a case sensitive comparison, use the StartsStr (see page 21989) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

StartsStr (see page 21989)

155.1.51 StrUtils.StuffString Function

Inserts a substring into a specified position of a string, replacing the current characters.

Pascal

```
function StuffString(const AText: string; AStart: Cardinal; ALength: Cardinal; const ASubText:
string): string;
```

C++

```
AnsiString StuffString(const AnsiString AText, unsigned AStart, unsigned ALength, const
AnsiString ASubText);
```

File

StrUtils

Description

StuffString returns a new string that is constructed by replacing the characters of AText specified by AStart and ALength with the string specified by ASubText.

AText is the string to which ASubText is added. It is not modified by StuffString.

AStart is the index of the first character in AText to be replaced, where 1 identifies the first character, 2 identifies the second character, and so on.

ALength is the number of characters in AText that are replaced by ASubText. If ALength is 0, ASubText is inserted into AText without removing any characters. If AText is negative, the characters preceding the insertion point are duplicated after the occurrence of ASubText in the result string.

ASubText is the substring to insert into AText, replacing the characters specified by AStart and ALength.

See Also

AnsiReplaceStr (see page 21969)

AnsiReplaceText (see page 21969)

StringReplace

155.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TStringSeachOption (see page 21992)	TStringSeachOption is an enumeration that describes the behavior of string searches.

155.2.1 StrUtils.TStringSeachOption Enumeration

TStringSeachOption is an enumeration that describes the behavior of string searches.

Pascal

```
TStringSeachOption = (
  soDown,
  soMatchCase,
  soWholeWord
);
```

C++

```
enum TStringSeachOption {
  soDown,
  soMatchCase,
  soWholeWord
};
```

File

StrUtils

Description

Note the unusual spelling. TStringSeachOptioncan have one of the following values:

Value	Meaning
soDown	A forward search is performed through the target text from the specified start point to the end of the search buffer. If soDown is not specified a reverse search is performed from the start point to the begining of the search buffer.
soMatchCase	Performs a case sensitive search.
soWholeWord	Match whole words, not pieces of words. Whole words are determined using standard identifier parser rules.

See Also

Commonly Used Long String Routines

String Types

String Constants

TStringSearchOptions (see page 21994)

155.3 Types

The following table lists types in this documentation.

Types

Name	Description
TCompareTextProc (see page 21993)	TCompareTextProc is a function pointer which specifies a function to compare two string values, AText, and AOther, and return true if these two values are equivilant.
TSoundexIntLength (see page 21993)	TSoundexIntLength The maximum calculation length for the SoundexInt (see page 21987) function. The integer value can be from 1 to 8.
TSoundexLength (see page 21994)	TSoundexLength The maximum calculation length of the SoundEx function. This value determines how many characters to use when generating the soundex string form the source string.
TStringSearchOptions (see page 21994)	TStringSearchOptions is set of the enumeration TStringSeachOption (see page 21992).

155.3.1 StrUtils.TCompareTextProc Type

TCompareTextProc is a function pointer which specifies a function to compare two string values, AText, and AOther, and return true if these two values are equivilant.

Pascal

```
TCompareTextProc = function (const AText, AOther: string): Boolean;
```

C++

```
(const AText, AOther: string): Boolean ( TCompareTextProc)();
```

File

StrUtils

See Also

TStringSeachOption (see page 21992)

155.3.2 StrUtils.TSoundexIntLength Type

TSoundexIntLength The maximum calculation length for the SoundexInt (see page 21987) function. The integer value can be from 1 to 8.

Pascal

```
TSoundexIntLength = 1..8;
```

C++

```
1..8 TSoundexIntLength;
```

File

StrUtils

Description

This value determines how many characters to use when generating the soundex string from the source string. There is a maximum of 8 characters due to the maximum capacity of the Integer type.

See Also

SoundexLength

155.3.3 StrUtils.TSoundexLength Type

TSoundexLength The maximum calculation length of the SoundEx function. This value determines how many characters to use when generating the soundex string form the source string.

Pascal

```
TSoundexLength = 1..MaxInt;
```

C++

```
1..MaxInt TSoundexLength;
```

File

StrUtils

See Also

SoundexIntLength

155.3.4 StrUtils.TStringSearchOptions Type

TStringSearchOptions is set of the enumeration TStringSeachOption (see page 21992).

Pascal

```
TStringSearchOptions = set of TStringSeachOption;
```

C++

```
TStringSeachOption TStringSearchOptions;
```

File

StrUtils

Description

Note the unusual spelling of the enumeration.

See Also

TStringSeachOption (see page 21992)

155.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
AnsiResemblesProc (see page 21995)	Controls (see page 5711) the algorithm used by AnsiResemblesText (see page 21970) to determine when two strings are similar.
ResemblesProc (see page 21995)	Controls (see page 5711) the algorithm used by AnsiResemblesText (see page 21970) to determine when two strings are similar. It is the same as AnsiResemblesProc (see page 21995).

155.4.1 StrUtils.AnsiResemblesProc Variable

Controls (see page 5711) the algorithm used by AnsiResemblesText (see page 21970) to determine when two strings are similar.

Pascal

```
AnsiResemblesProc: TCompareTextProc = SoundexProc;
```

C++

```
TCompareTextProc AnsiResemblesProc = SoundexProc;
```

File

StrUtils

Description

The AnsiResemblesText (see page 21970) function uses the function assigned to AnsiResemblesProc to determine when two strings should be considered similar. By default, AnsiResemblesProc is set to SoundExProc.

To make AnsiResemblesText (see page 21970) use a different algorithm, you can change AnsiResemblesProc to another function. For example, you can change the number of phonemes compared by using a function that assigns a different length to SoundExSimilar than then default value of 4. Alternately, you can create a function that uses a different algorithm entirely.

See Also

AnsiResemblesText (see page 21970)

SoundExSimilar

155.4.2 StrUtils.ResemblesProc Variable

Controls (see page 5711) the algorithm used by AnsiResemblesText (see page 21970) to determine when two strings are similar. It is the same as AnsiResemblesProc (see page 21995).

Pascal

```
ResemblesProc: TCompareTextProc = SoundexProc;
```

C++

```
TCompareTextProc ResemblesProc = SoundexProc;
```

File

StrUtils

Description

The AnsiResemblesText (see page 21970) function uses the function assigned to ResemblesProc to determine when two strings should be considered similar. By default, ResemblesProc is set to SoundExProc.

To make AnsiResemblesText (see page 21970) use a different algorithm, you can change ResemblesProc to another function. For example, you can change the number of phonemes compared by using a function that assigns a different length to SoundExSimilar than then default value of 4. Alternately, you can create a function that uses a different algorithm entirely.

See Also

AnsiResemblesText (see page 21970)

AnsiResemblesProc (see page 21995)

SoundExSimilar

155.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
WordDelimiters (see page 21997)	This is constant StrUtils.WordDelimiters.

155.5.1 StrUtils.WordDelimiters Constant

Pascal

```
WordDelimiters: set of Char = [#0..#255] - ['a'..'z','A'..'Z','1'..'9','0'];
```

C++

```
set of Char WordDelimiters = [#0..#255] - ['a'..'z','A'..'Z','1'..'9','0'];
```

File

StrUtils

Description

This is constant StrUtils.WordDelimiters.

156 SvcMgr Namespace

This is namespace SvcMgr.

Topics

Name	Description
Classes (see page 22001)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 22047)	The following table lists structs, records, enums in this documentation.
Types (see page 22051)	The following table lists types in this documentation.
Variables (see page 22054)	The following table lists variables in this documentation.
Constants (see page 22055)	The following table lists constants in this documentation.

Classes

	Name	Description
	TDependencies (see page 22001)	TDependencies is a container for TDependency (see page 22005) objects.
	TDependency (see page 22005)	TDependency object represents a dependent service or load ordering group used by TService (see page 22011) objects.
	TEventLogger (see page 22009)	This is class SvcMgr.TEventLogger.
	TService (see page 22011)	TService encapsulates a Windows NT service.
	TServiceApplication (see page 22032)	TServiceApplication encapsulates a Windows NT service application.
	TServiceThread (see page 22040)	TServiceThread represents the thread associated with a specific service (TService (see page 22011) object).

Constants

Name	Description
CM_SERVICE_CONTROL_CODE (see page 22055)	This is constant SvcMgr.CM_SERVICE_CONTROL_CODE.

Structs, Records, Enums

	Name	Description
	TCurrentStatus (see page 22047)	TCurrentStatus indicates the state of a service.
	TErrorSeverity (see page 22048)	TErrorSeverity indicates the severity of a failure and implies the corresponding response by a service startup program.
	TServiceType (see page 22048)	TServiceType indicates the type of a service.
	TStartType (see page 22049)	TStartType indicates how a service starts up.

Types

Name	Description
TContinueEvent (see page 22051)	This is type SvcMgr.TContinueEvent.

TPauseEvent (see page 22051)	This is type SvcMgr.TPauseEvent.
TServiceController (see page 22052)	This is type SvcMgr.TServiceController.
TServiceEvent (see page 22052)	This is type SvcMgr.TServiceEvent.
TStartEvent (see page 22052)	This is type SvcMgr.TStartEvent.
TStopEvent (see page 22053)	This is type SvcMgr.TStopEvent.

Variables

Name	Description
Application (see page 22054)	This is variable SvcMgr.Application.

156.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TDependencies (see page 22001)	TDependencies is a container for TDependency (see page 22005) objects.
	TDependency (see page 22005)	TDependency object represents a dependent service or load ordering group used by TService (see page 22011) objects.
	TEventLogger (see page 22009)	This is class SvcMgr.TEventLogger.
	TService (see page 22011)	TService encapsulates a Windows NT service.
	TServiceApplication (see page 22032)	TServiceApplication encapsulates a Windows NT service application.
	TServiceThread (see page 22040)	TServiceThread represents the thread associated with a specific service (TService (see page 22011) object).

156.1.1 TDependencies Class

TDependencies is a container for TDependency (see page 22005) objects.

Class Hierarchy

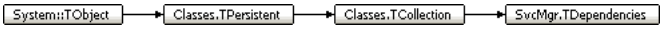

Pascal

```
TDependencies = class(TCollection);
```

C++

```
class TDependencies : public TCollection;
```

File

SvcMgr

Description

TDependencies is a collection of TDependency (see page 22005) objects, each of which represents a dependent service or load ordering group for a service (TService (see page 22011) instance). TDependencies objects are the implementation of the Dependencies property of TService (see page 22011).

See Also

TDependency (see page 22005)

TService (see page 22011)

TService (see page 22011)

TService (see page 22011)

156.1.1.1 TDependencies Members

The following tables list the members exposed by TDependencies.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 22004)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 2736)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 22003)	Creates an instance of a TDependencies (see page 22001) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 22004)	Lists the dependency objects in the dependencies collection.

156.1.1.2 TDependencies.Create Constructor

Creates an instance of a TDependencies (see page 22001) object.

Pascal

```
constructor Create(Owner: TPersistent);
```

C++

```
__fastcall TDependencies(TPersistent * Owner);
```

Description

Applications should not need to call the constructor directly. Instances of TService (see page 22011) create the TDependencies (see page 22001) object from their own constructors. The Owner parameter is the TService (see page 22011) instance that uses the TDependencies (see page 22001) object to implement its Dependencies property.

Create sets the TDependency (see page 22005) owner to the Owner parameter and calls the inherited Create.

See Also

Create (see page 2734)

TService (see page 22011)

156.1.1.3 TDependencies Methods

The methods of the TDependencies class are listed here.

Protected Methods

	Name	Description
	GetOwner (see page 22004)	Returns the owner of an object.

156.1.1.3.1 TDependencies.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

156.1.1.4 TDependencies Properties

The properties of the TDependencies class are listed here.

Public Properties

	Name	Description
	Items (see page 22004)	Lists the dependency objects in the dependencies collection.

156.1.1.4.1 TDependencies.Items Property

Lists the dependency objects in the dependencies collection.

Pascal

```
property Items [Index: Integer]: TDependency;
```

C++

```
__property TDependency Items[int Index];
```

Description

The value of the Index parameter corresponds to the Index property of each TDependency (see page 22005) object in the collection. It represents the position of the dependency object in the dependencies collection.

Note: In Delphi, Items is the default property of TDependencies (see page 22001). This means that you can omit the property

names. For example

Note: Service1.Dependencies.Items[i]

Note: can be written

Note: Service1.Dependencies[i]

Note: In C++, Items can be accessed using the [] operator, to achieve an affect similar to the default property in Delphi.

See Also

Index (see page 2762)

operator_[]

156.1.2 TDependency Class

TDependency object represents a dependent service or load ordering group used by TService (see page 22011) objects.

Class Hierarchy

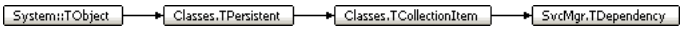

Pascal

```
TDependency = class(TCollectionItem);
```

C++

```
class TDependency : public TCollectionItem;
```

File

SvcMgr

Description

A TDependency object represents a new service or load ordering group that must start before the service represented by a TService (see page 22011) object. Collections of dependency objects are TDependencies (see page 22001) objects. TService (see page 22011) has Dependencies and LoadGroup properties that use dependency collections. Each dependency has a Name (see page 22008) property that identifies the service or load ordering group, and a boolean IsGroup (see page 22008) property to indicate whether the dependency represents a load ordering group.

See Also

TDependencies (see page 22001)

TService (see page 22011)

TService (see page 22011)

Service Name Properties

156.1.2.1 TDependency Members

The following tables list the members exposed by TDependency.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 22007)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2757)	Creates and initializes a TCollectionItem (see page 2755) instance.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2757)	Destroys the TCollectionItem (see page 2755) instance and frees its memory.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).

Published Properties

	Name	Description
	IsGroup (see page 22008)	Indicates whether the dependency represents a load ordering group for a service (TService (see page 22011) object).
	Name (see page 22008)	Identifies the dependent service or load ordering group used by a TService (see page 22011) object.

156.1.2.2 TDependency Methods

The methods of the TDependency class are listed here.

Protected Methods

	Name	Description
	GetDisplayName (see page 22007)	Returns the name of the collection item as it appears in the collection editor.

156.1.2.2.1 TDependency.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName (see page 2761)

GetNamePath (see page 2759)

156.1.2.3 TDependency Properties

The properties of the TDependency class are listed here.

Published Properties

	Name	Description
	IsGroup (see page 22008)	Indicates whether the dependency represents a load ordering group for a service (TService (see page 22011) object).

	Name (see page 22008)	Identifies the dependent service or load ordering group used by a TService (see page 22011) object.

156.1.2.3.1 TDependency.IsGroup Property

Indicates whether the dependency represents a load ordering group for a service (TService (see page 22011) object).

Pascal

```
property IsGroup: Boolean;
```

C++

```
__property Boolean IsGroup;
```

Description

IsGroup indicates whether the dependency represents a load ordering group.

Note: : Because services and service groups share the same name space, group names must be prefixed by the SC_GROUP_IDENTIFIER character to distinguish them from service names. If IsGroup is set to true, SC_GROUP_IDENTIFIER will automatically be prefixed to the Dependency name (Name (see page 22008) property). DisplayName is not modified.

See Also

Name (see page 22008)

LoadGroup (see page 22023)

156.1.2.3.2 TDependency.Name Property

Identifies the dependent service or load ordering group used by a TService (see page 22011) object.

Pascal

```
property Name: String;
```

C++

```
__property AnsiString Name;
```

Description

Name is the name identifying the service or load ordering group. If Name is the name of a service, Name reflects the actual ServiceStartName of a TService (see page 22011) instance.

The property editor for the Dependencies property of TService (see page 22011) technically uses the dependency's DisplayName to identify the dependency service in its collection list. However, Name and DisplayName are nearly always the same.

Dependency objects that are load ordering groups are prefixed by SC_GROUP_IDENTIFIER. Setting the IsGroup (see page 22008) property handles prefixing the Name automatically.

Note: A dependency can be a service. If this is true, do not confuse the various name-related properties of dependency and service objects. For more information, see Service name properties.

See Also

DisplayName

LoadGroup (see page 22023)

TService (see page 22011)

156.1.3 TEventLogger Class

Class Hierarchy

System::TObject → SvcMgr.TEventLogger

Pascal

```
TEventLogger = class(TObject);
```

C++

```
class TEventLogger : public TObject;
```

File

SvcMgr

Description

This is class SvcMgr.TEventLogger.

156.1.3.1 TEventLogger Members

The following tables list the members exposed by TEventLogger.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22010)	This is Create, a member of class TEventLogger.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22010)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LogMessage (see page 22011)	This is LogMessage, a member of class TEventLogger.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

156.1.3.2 TEventLogger.Create Constructor

Pascal

```
constructor Create(Name: String);
```

C++

```
__fastcall TEventLogger(AnsiString Name);
```

Description

This is Create, a member of class TEventLogger.

156.1.3.3 TEventLogger.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TEventLogger();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

156.1.3.4 TEventLogger Methods

The methods of the TEventLogger class are listed here.

Public Methods

	Name	Description
	LogMessage (see page 22011)	This is LogMessage, a member of class TEventLogger.

156.1.3.4.1 TEventLogger.LogMessage Method

Pascal

```
procedure LogMessage(Message: String; EventType: DWord = 1; Category: Word = 0; ID: DWord = 0);
```

C++

```
__fastcall LogMessage(AnsiString Message, DWord EventType = 1, Word Category = 0, DWord ID =
0);
```

Description

This is LogMessage, a member of class TEventLogger.

156.1.4 TService Class

TService encapsulates a Windows NT service.

Class Hierarchy

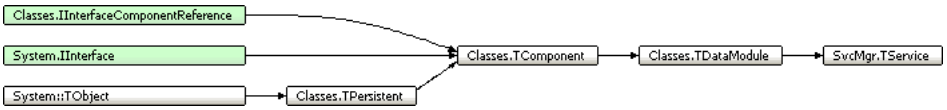

Pascal

```
TService = class(TDataModule);
```

C++

```
class TService : public TDataModule;
```

File

SvcMgr

Description

TService encapsulates a Windows NT service in an NT service application. A Win32 service is accessed via the Service Control Manager and can be started automatically at system boot, by a user through the Services control panel applet, or by a Win32-based application that uses the service functions. Services can execute even when no user is logged on to the system. Windows NT also supports a driver service that does not interact with the SCM.

See Also

TServiceApplication (see page 22032)

Service Applications

Service Name Properties

Service Threads

156.1.4.1 TService Members

The following tables list the members exposed by TService.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Controller (see page 22017)	This is Controller, a member of class TService.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoContinue (see page 22017)	This is DoContinue, a member of class TService.
V	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
V	DoCustomControl (see page 22017)	This is DoCustomControl, a member of class TService.
V	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
V	DoInterrogate (see page 22017)	This is DoInterrogate, a member of class TService.
V	DoPause (see page 22018)	This is DoPause, a member of class TService.
V	DoShutdown (see page 22018)	This is DoShutdown, a member of class TService.
V	DoStart (see page 22018)	This is DoStart, a member of class TService.
V	DoStop (see page 22018)	This is DoStop, a member of class TService.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Main (see page 22019)	This is Main, a member of class TService.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 2829)	Generates an OnCreate (see page 2833) event.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 2830)	Generates an OnDestroy (see page 2834) event.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 2827)	Creates an instance of a data module.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	CreateNew (see page 22015)	Instantiates and initializes a TService (see page 22011) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22016)	Disposes of a TService (see page 22011) instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.

	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
A	GetServiceController (see page 22018)	Returns the service's registered handler function.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LogMessage (see page 22019)	Sends an error message to the event log.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	ReportStatus (see page 22020)	Updates the Service Control Manager's status information for the calling service.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.
	ErrCode (see page 22022)	Contains the service specific error code returned on starting or stopping the service.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Param (see page 22024)	Lists the parameters to the service.
R	ParamCount (see page 22024)	Indicates the number of startup parameters for the service.
R	ServiceThread (see page 22025)	Identifies the dedicated thread for the service.
	Status (see page 22026)	Indicates the current state of the service.
R	Terminated (see page 22027)	Indicates whether the service thread has terminated.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

	Win32ErrCode (see page 22028)	Indicates the Win32 error code reported by the service if a service specific code is not defined.

Published Events

	Name	Description
	AfterInstall (see page 22028)	Occurs immediately after the service is registered with the Windows Service Control manager.
	AfterUninstall (see page 22029)	Occurs immediately after the service is removed from the Service Control manager's database.
	BeforeInstall (see page 22029)	Occurs before the service is first registered with the Windows Service Control manager.
	BeforeUninstall (see page 22030)	Occurs immediately before the service is removed from the Service Control manager's database.
	OnContinue (see page 22030)	Occurs when the Service Control manager resumes the service after it has been paused.
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.
	OnExecute (see page 22030)	Occurs when the thread associated with the service starts up.
	OnPause (see page 22031)	Occurs when the Service Control manager pauses the service temporarily.
	OnShutdown (see page 22031)	Occurs when the system running the service application is about to shut down.
	OnStart (see page 22031)	OnStartup occurs when the service first starts up, before the OnExecute (see page 22030) event.
	OnStop (see page 22032)	Occurs when the Service Control manager stops the service.

Published Properties

	Name	Description
	AllowPause (see page 22021)	Indicates whether clients are allowed to pause the service.
	AllowStop (see page 22021)	Indicates whether clients are allowed to stop the service.
	Dependencies (see page 22021)	Lists the services or load ordering groups on which the service depends.
	DisplayName (see page 22022)	Identifies the service in the Windows Service Control Manager.
	ErrorSeverity (see page 22022)	Indicates the severity of service startup failure.
	Interactive (see page 22023)	Indicates whether the service is interactive with the Windows desktop.
	LoadGroup (see page 22023)	Names of the load ordering group that includes the service.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.
	Password (see page 22024)	Specifies the password, optionally used depending upon the ServiceStartName (see page 22025), for accessing the service.
	ServiceStartName (see page 22025)	ServiceStartName is the name used in the Service Control Manager Startup dialog to start the service.
	ServiceType (see page 22026)	Identifies the type of the service as Win32, device driver, or file system driver.
	StartType (see page 22026)	Indicates when or how the service starts.
	Tag (see page 2808)	Stores an integer value as part of a component.
	TagID (see page 22026)	Specifies a unique tag for the service within its LoadGroup (see page 22023).
	WaitHint (see page 22027)	Specifies the estimated time, in milliseconds, between a call from the Windows Service Control Manager and the service reporting its status.

156.1.4.2 TService.CreateNew Constructor

Instantiates and initializes a TService (see page 22011) object.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer); override;
```

C++

```
virtual __fastcall TService(TComponent * AOwner, int Dummy);
```

Description

Call CreateNew to instantiate a service object at runtime. Services added at design time are created automatically. The AOwner parameter specifies the TServiceApplication (see page 22032) object that is responsible for ultimately freeing the service. The Dummy parameter is not used. Use CreateNew rather than the Create constructor to instantiate a new service object at runtime. CreateNew does not attempt to stream in an associated DFM file that indicates any components added to the service at design time.

See Also

Owner (see page 2807)

Create (see page 22003)

TServiceApplication (see page 22032)

156.1.4.3 TService.Destroy Destructor

Disposes of a TService (see page 22011) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TService();
```

Description

Destroy frees the memory associated with the objects for the Params and Dependencies (see page 22021) properties that were created in the constructor, and for the EventLogger object created in the LogMessage (see page 22019) method. Then it calls the inherited Destroy.

See Also

Free (see page 22300)

LogMessage (see page 22019)

156.1.4.4 TService Methods

The methods of the TService class are listed here.

Protected Methods

	Name	Description
	Controller (see page 22017)	This is Controller, a member of class TService.
V	DoContinue (see page 22017)	This is DoContinue, a member of class TService.
V	DoCustomControl (see page 22017)	This is DoCustomControl, a member of class TService.
V	DoInterrogate (see page 22017)	This is DoInterrogate, a member of class TService.
V	DoPause (see page 22018)	This is DoPause, a member of class TService.
V	DoShutdown (see page 22018)	This is DoShutdown, a member of class TService.
V	DoStart (see page 22018)	This is DoStart, a member of class TService.
V	DoStop (see page 22018)	This is DoStop, a member of class TService.
	Main (see page 22019)	This is Main, a member of class TService.

Public Methods

	Name	Description
	GetServiceController (see page 22018)	Returns the service's registered handler function.
	LogMessage (see page 22019)	Sends an error message to the event log.
	ReportStatus (see page 22020)	Updates the Service Control Manager's status information for the calling service.

156.1.4.4.1 TService.Controller Method

Pascal

```
procedure Controller(CtrlCode: DWord);
```

C++

```
__fastcall Controller(DWord CtrlCode);
```

Description

This is Controller, a member of class TService.

156.1.4.4.2 TService.DoContinue Method

Pascal

```
function DoContinue: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoContinue();
```

Description

This is DoContinue, a member of class TService.

156.1.4.4.3 TService.DoCustomControl Method

Pascal

```
function DoCustomControl(CtrlCode: DWord): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoCustomControl(DWord CtrlCode);
```

Description

This is DoCustomControl, a member of class TService.

156.1.4.4.4 TService.DoInterrogate Method

Pascal

```
procedure DoInterrogate; virtual;
```

C++

```
virtual __fastcall DoInterrogate();
```

Description

This is DoInterrogate, a member of class TService.

156.1.4.4.5 TService.DoPause Method

Pascal

```
function DoPause: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoPause();
```

Description

This is DoPause, a member of class TService.

156.1.4.4.6 TService.DoShutdown Method

Pascal

```
procedure DoShutdown; virtual;
```

C++

```
virtual __fastcall DoShutdown();
```

Description

This is DoShutdown, a member of class TService.

156.1.4.4.7 TService.DoStart Method

Pascal

```
procedure DoStart; virtual;
```

C++

```
virtual __fastcall DoStart();
```

Description

This is DoStart, a member of class TService.

156.1.4.4.8 TService.DoStop Method

Pascal

```
function DoStop: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoStop();
```

Description

This is DoStop, a member of class TService.

156.1.4.4.9 TService.GetServiceController Method

Returns the service's registered handler function.

Pascal

```
function GetServiceController: TServiceController; virtual; abstract;
```

C++

```
virtual __fastcall TServiceController GetServiceController() = 0;
```

Description

GetServiceController returns the handler for the service. You do not need to call this function directly. The service object's main function calls it automatically.

Note: Each service in the service application has a main function for that service. When a request is made to start the service, the main thread of the service calls the RegisterServiceCtrlHandler function to register this control handler function. TService (see page 22011) automatically calls RegisterServiceCtrlHandler and returns this handler function. Because this function cannot be a class method (member function), the Service application wizard generates code that creates a non-member handler, (ServiceController) for your service object. This routine returns the service object's Handler (Controller (see page 22017)).

156.1.4.4.10 TService.LogMessage Method

Sends an error message to the event log.

Pascal

```
procedure LogMessage(Message: String; EventType: DWord = 1; Category: Integer = 0; ID: Integer
= 0);
```

C++

```
__fastcall LogMessage(AnsiString Message, DWord EventType = 1, int Category = 0, int ID = 0);
```

Description

LogMessage sends an error message to the event log when an error or exception occurs. By default the EventType is EVENTLOG_ERROR_TYPE (usually indicating a loss of functionality or data) and the category and ID are zero. For example, if a service cannot be loaded as the system boots, it can log an error event, and the category and ID are zero.

The EventType specifies the type of event being logged. This parameter can be one of the following values:

Value	Meaning
EVENTLOG_WARNING_TYPE	Warning event
EVENTLOG_INFORMATION_TYPE	Information event
EVENTLOG_AUDIT_SUCCESS	Success Audit event
EVENTLOG_AUDIT_FAILURE	Failure Audit event

EVENTLOG_ERROR_TYPEError event

The Category parameter specifies the event category, which is source-specific information and can have any value.

The ID parameter specifies the event identifier, which is the message that goes with this event as an entry in the message file associated with the event source.

156.1.4.4.11 TService.Main Method

Pascal

```
procedure Main(Argc: DWord; Argv: PLPSTR);
```

C++

```
__fastcall Main(DWord Argc, PLPSTR Argv);
```

Description

This is Main, a member of class TService.

156.1.4.4.12 TService.ReportStatus Method

Updates the Service Control Manager's status information for the calling service.

Pascal

```
procedure ReportStatus;
```

C++

```
__fastcall ReportStatus();
```

Description

ReportStatus sends current status information about the calling service to the Service Control Manager.

See Also

LogMessage (see page 22019)

Status (see page 22026)

WaitHint (see page 22027)

156.1.4.5 TService Properties

The properties of the TService class are listed here.

Public Properties

	Name	Description
	ErrCode (see page 22022)	Contains the service specific error code returned on starting or stopping the service.
R	Param (see page 22024)	Lists the parameters to the service.
R	ParamCount (see page 22024)	Indicates the number of startup parameters for the service.
R	ServiceThread (see page 22025)	Identifies the dedicated thread for the service.
	Status (see page 22026)	Indicates the current state of the service.
R	Terminated (see page 22027)	Indicates whether the service thread has terminated.
	Win32ErrCode (see page 22028)	Indicates the Win32 error code reported by the service if a service specific code is not defined.

Published Properties

	Name	Description
	AllowPause (see page 22021)	Indicates whether clients are allowed to pause the service.
	AllowStop (see page 22021)	Indicates whether clients are allowed to stop the service.
	Dependencies (see page 22021)	Lists the services or load ordering groups on which the service depends.
	DisplayName (see page 22022)	Identifies the service in the Windows Service Control Manager.
	ErrorSeverity (see page 22022)	Indicates the severity of service startup failure.
	Interactive (see page 22023)	Indicates whether the service is interactive with the Windows desktop.
	LoadGroup (see page 22023)	Names of the load ordering group that includes the service.
	Password (see page 22024)	Specifies the password, optionally used depending upon the ServiceStartName (see page 22025), for accessing the service.
	ServiceStartName (see page 22025)	ServiceStartName is the name used in the Service Control Manager Startup dialog to start the service.
	ServiceType (see page 22026)	Identifies the type of the service as Win32, device driver, or file system driver.
	StartType (see page 22026)	Indicates when or how the service starts.

	TagID (see page 22026)	Specifies a unique tag for the service within its LoadGroup (see page 22023).
	WaitHint (see page 22027)	Specifies the estimated time, in milliseconds, between a call from the Windows Service Control Manager and the service reporting its status.

156.1.4.5.1 TService.AllowPause Property

Indicates whether clients are allowed to pause the service.

Pascal

```
property AllowPause: Boolean;
```

C++

```
__property Boolean AllowPause;
```

Description

AllowPause indicates whether clients of the service can pause and resume the service by calling the Windows API ControlService function. When AllowPause is true, an OnPause (see page 22031) event occurs before the service pauses, and an OnContinue (see page 22030) event occurs when the service resumes.

See Also

OnPause (see page 22031)

OnContinue (see page 22030)

156.1.4.5.2 TService.AllowStop Property

Indicates whether clients are allowed to stop the service.

Pascal

```
property AllowStop: Boolean;
```

C++

```
__property Boolean AllowStop;
```

Description

AllowStop indicates whether clients of the service can stop it by calling the Windows API ControlService function. When AllowStop is true, an OnStop (see page 22032) event occurs before the service stops.

See Also

OnStop (see page 22032)

156.1.4.5.3 TService.Dependencies Property

Lists the services or load ordering groups on which the service depends.

Pascal

```
property Dependencies: TDependencies;
```

C++

```
__property TDependencies Dependencies;
```

Description

Dependencies is a container for TDependency (see page 22005) objects, which are dependent services or load ordering groups that must start before this service. If the container is empty, the service has no dependencies.

Dependency on a service means that this service can only run if the service it depends on is running. Dependency on a group means that this service can run if at least one member of the group is running after an attempt to start all members of the group.

See Also

TDependency (see page 22005)

LoadGroup (see page 22023)

156.1.4.5.4 TService.DisplayName Property

Identifies the service in the Windows Service Control Manager.

Pascal

```
property DisplayName: String;
```

C++

```
__property AnsiString DisplayName;
```

Description

DisplayName contains a string, with a maximum length of 256 characters, that is the name of the service as displayed in the Service Control Manager. User interface programs use DisplayName to identify the service. DisplayName is case-preserved in the Service Control Manager, however display name comparisons are always case-insensitive.

Note: A dependency can be a service. If this is true, do not confuse the various name-related properties of dependency and service objects. For more information, see Service name properties.

See Also

Name (see page 2805)

ServiceStartName (see page 22025)

156.1.4.5.5 TService.ErrCode Property

Contains the service specific error code returned on starting or stopping the service.

Pascal

```
property ErrCode: DWord;
```

C++

```
__property DWord ErrCode;
```

Description

ErrCode specifies a service specific error code that the service returns when an error occurs while the service is starting or stopping. If ErrCode is not defined, the Win32ErrCode (see page 22028) property is used instead.

See Also

ReportStatus (see page 22020)

Win32ErrCode (see page 22028)

156.1.4.5.6 TService.ErrorSeverity Property

Indicates the severity of service startup failure.

Pascal

```
property ErrorSeverity: TErrorSeverity;
```

C++

```
__property TErrorSeverity ErrorSeverity;
```

Description

ErrorSeverity indicates the severity if this service fails to start during startup of the service application.

See Also

ErrCode (see page 22022)

ReportStatus (see page 22020)

Win32ErrCode (see page 22028)

156.1.4.5.7 TService.Interactive Property

Indicates whether the service is interactive with the Windows desktop.

Pascal

```
property Interactive: Boolean;
```

C++

```
__property Boolean Interactive;
```

Description

Interactive indicates whether the service can interact with the Windows desktop. Interactive only has an effect when ServiceType (see page 22026) is stWin32. For more information about creating interactive NT services, see the Microsoft Help.

See Also

ServiceType (see page 22026)

156.1.4.5.8 TService.LoadGroup Property

Names of the load ordering group that includes the service.

Pascal

```
property LoadGroup: String;
```

C++

```
__property AnsiString LoadGroup;
```

Description

LoadGroup is the name of the load-ordering group that includes the service. This is used by other services that have dependencies on the load ordering group (See Dependencies (see page 22021)). The registry lists load-ordering groups at HKEY_LOCAL_MACHINE\System (see page 22079)\CurrentControlSet\Control\ServiceGroupOrder. The order in which services are loaded depends on the order of the load groups in the Registry (see page 18633) and the dependencies listed in service Dependencies (see page 22021) properties.

Because services and service groups share the same name space, the SC_GROUP_IDENTIFIER character must prefix group names to distinguish them from service names. This is automatically prefixed if the TDependency (see page 22005)'s IsGroup property is true.

Note: A dependency can be a service. If this is true, do not confuse the various name-related properties of dependency and

service objects. For more information, see Service name properties.

See Also

Dependencies (see page 22021)

TagID (see page 22026)

156.1.4.5.9 TService.Param Property

Lists the parameters to the service.

Pascal

```
property Param [Index: Integer]: String;
```

C++

```
__property AnsiString Param[int Index];
```

Description

Param is an indexed property that contains the parameters to the service. These parameters are typed into the Startup Parameters input window in the Service Control Manager.

See Also

ParamCount (see page 22024)

156.1.4.5.10 TService.ParamCount Property

Indicates the number of startup parameters for the service.

Pascal

```
property ParamCount: Integer;
```

C++

```
__property int ParamCount;
```

Description

ParamCount is the number of parameters for the service. These parameters are typed into the Startup Parameters input window in the Service Control Manager.

See Also

Param (see page 22024)

156.1.4.5.11 TService.Password Property

Specifies the password, optionally used depending upon the ServiceStartName (see page 22025), for accessing the service.

Pascal

```
property Password: String;
```

C++

```
__property AnsiString Password;
```

Description

Password is the password for the account name specified by the ServiceStartName (see page 22025) property if the ServiceType (see page 22026) is stWin32.

If Password is nil (Delphi) or NULL (C++), or if it points to an empty string, the service has no password. If ServiceType (see page 22026) is stDevice or stFileSystem, this parameter is ignored.

See Also

ServiceStartName (see page 22025)

156.1.4.5.12 TService.ServiceStartName Property

ServiceStartName is the name used in the Service Control Manager Startup dialog to start the service.

Pascal

```
property ServiceStartName: String;
```

C++

```
__property AnsiString ServiceStartName;
```

Description

ServiceStartName is the name used to start the service.

If ServiceType (see page 22026) is stWin32, ServiceStartName is the account name (in the form "DomainName\Username") which the service process is logged on as when it runs. If the account belongs to the built-in domain, ".\Username" can be specified. If the service application includes more than one service, leave ServiceStartName blank, and leave the Password (see page 22024) property blank.

If the service type is not stWin32, ServiceStartName is the driver object name (\FileSytems\Rdr or \Driver\Xns) which the input and output system uses to load the device driver. With device drivers or file systems drivers, you can leave ServiceStartName blank and the I/O system will create a default name based on the service name.

Note: A dependency can be a service. If this is true, do not confuse the various name-related properties of dependency and service objects. For more information, see Service name properties.

See Also

Password (see page 22024)

156.1.4.5.13 TService.ServiceThread Property

Identifies the dedicated thread for the service.

Pascal

```
property ServiceThread: TServiceThread;
```

C++

```
__property TServiceThread ServiceThread;
```

Description

ServiceThread is the instance of the TServiceThread (see page 22040) class that is the dedicated thread for the service. Each service in the service application has a dedicated thread that contains a loop in its Execute method for processing service requests. Often a service will spawn a new thread, derived from TThread, to free the service thread to continue processing service requests.

See Also

ProcessRequests (see page 22046)

TThread (see page 3108)

156.1.4.5.14 TService.ServiceType Property

Identifies the type of the service as Win32, device driver, or file system driver.

Pascal

```
property ServiceType: TServiceType;
```

C++

```
__property TServiceType ServiceType;
```

Description

ServiceType indicates the type of service. ServiceType can be stWin32 (a Win32 service – the default), stDevice (a device driver), or stFileSystem (a file system driver).

See Also

Interactive (see page 22023)

156.1.4.5.15 TService.StartType Property

Indicates when or how the service starts.

Pascal

```
property StartType: TStartType;
```

C++

```
__property TStartType StartType;
```

Description

StartType specifies how and why the service starts

156.1.4.5.16 TService.Status Property

Indicates the current state of the service.

Pascal

```
property Status: TCurrentStatus;
```

C++

```
__property TCurrentStatus Status;
```

Description

Status indicates the current state of the service. The status information is displayed in the Service Control Manager.

See Also

ReportStatus (see page 22020)

156.1.4.5.17 TService.TagID Property

Specifies a unique tag for the service within its LoadGroup (see page 22023).

Pascal

```
property TagID: DWord;
```

C++

```
__property DWord TagID;
```

Description

TagID is the unique tag value for this service in its LoadGroup (see page 22023). If no tag is requested, this property is 0. Tags are only evaluated when the ServiceType (see page 22026) is not stWin32 and the StartType (see page 22026) is stBoot or stSystem.

See Also

Dependencies (see page 22021)

LoadGroup (see page 22023)

156.1.4.5.18 TService.Terminated Property

Indicates whether the service thread has terminated.

Pascal

```
property Terminated: Boolean;
```

C++

```
__property Boolean Terminated;
```

Description

Terminated indicates whether the dedicated thread that the service is running in has terminated.

See Also

Terminated (see page 3142)

Terminate (see page 3126)

156.1.4.5.19 TService.WaitHint Property

Specifies the estimated time, in milliseconds, between a call from the Windows Service Control Manager and the service reporting its status.

Pascal

```
property WaitHint: Integer;
```

C++

```
__property int WaitHint;
```

Description

WaitHint specifies the amount of time, in milliseconds, that can elapse between a pending start, stop, pause, or continue operation and the service setting its status with a call to the Windows API. Normally this is handled automatically. If the service performs a lengthy operation of more than WaitHint time in one of its event handlers, you should call ReportStatus (see page 22020) method to update to the Service Control Manager.

See Also

ReportStatus (see page 22020)

156.1.4.5.20 TService.Win32ErrCode Property

Indicates the Win32 error code reported by the service if a service specific code is not defined.

Pascal

```
property Win32ErrCode: DWord;
```

C++

```
__property DWord Win32ErrCode;
```

Description

Win32ErrCode specifies a Win32 error code that the service uses to report an error that occurs when it is starting or stopping. If an error code specific to the service is defined, this property is automatically set to ERROR_SERVICE_SPECIFIC_ERROR and the service specific error is reported instead.

See Also

ErrCode (see page 22022)

ReportStatus (see page 22020)

156.1.4.6 TService Events

The events of the TService class are listed here.

Published Events

	Name	Description
	AfterInstall (see page 22028)	Occurs immediately after the service is registered with the Windows Service Control manager.
	AfterUninstall (see page 22029)	Occurs immediately after the service is removed from the Service Control manager's database.
	BeforeInstall (see page 22029)	Occurs before the service is first registered with the Windows Service Control manager.
	BeforeUninstall (see page 22030)	Occurs immediately before the service is removed from the Service Control manager's database.
	OnContinue (see page 22030)	Occurs when the Service Control manager resumes the service after it has been paused.
	OnExecute (see page 22030)	Occurs when the thread associated with the service starts up.
	OnPause (see page 22031)	Occurs when the Service Control manager pauses the service temporarily.
	OnShutdown (see page 22031)	Occurs when the system running the service application is about to shut down.
	OnStart (see page 22031)	OnStartup occurs when the service first starts up, before the OnExecute (see page 22030) event.
	OnStop (see page 22032)	Occurs when the Service Control manager stops the service.

156.1.4.6.1 TService.AfterInstall Event

Occurs immediately after the service is registered with the Windows Service Control manager.

Pascal

```
property AfterInstall: TServiceEvent;
```

C++

```
__property TServiceEvent AfterInstall;
```

Description

Write an AfterInstall event handler to take specific action immediately after the service is registered. Install services by running

the service application with the INSTALL switch.

See Also

AfterUninstall (see page 22029)

OnStart (see page 22031)

OnExecute (see page 22030)

OnCreate (see page 2833)

BeforeInstall (see page 22029)

156.1.4.6.2 TService.AfterUninstall Event

Occurs immediately after the service is removed from the Service Control manager's database.

Pascal

```
property AfterUninstall: TServiceEvent;
```

C++

```
__property TServiceEvent AfterUninstall;
```

Description

Write an AfterUninstall event handler to take specific action immediately after the service is removed. Uninstall services by running the service application with the UNINSTALL switch.

See Also

AfterInstall (see page 22028)

OnStop (see page 22032)

BeforeUninstall (see page 22030)

Destroy (see page 2829)

OnShutdown (see page 22031)

156.1.4.6.3 TService.BeforeInstall Event

Occurs before the service is first registered with the Windows Service Control manager.

Pascal

```
property BeforeInstall: TServiceEvent;
```

C++

```
__property TServiceEvent BeforeInstall;
```

Description

Write a BeforeInstall event handler to take specific action immediately before the service is first registered. Install services by running the service application with the INSTALL switch.

See Also

AfterInstall (see page 22028)

OnStart (see page 22031)

OnExecute (see page 22030)

OnCreate (see page 2833)

BeforeUninstall (see page 22030)

156.1.4.6.4 TService.BeforeUninstall Event

Occurs immediately before the service is removed from the Service Control manager's database.

Pascal

```
property BeforeUninstall: TServiceEvent;
```

C++

```
__property TServiceEvent BeforeUninstall;
```

Description

Write a BeforeUninstall event handler to take specific action immediately before the service is removed. Uninstall services by running the service application with the UNINSTALL switch.

See Also

AfterUninstall (see page 22029)

OnStop (see page 22032)

BeforeInstall (see page 22029)

Destroy (see page 2829)

OnShutdown (see page 22031)

156.1.4.6.5 TService.OnContinue Event

Occurs when the Service Control manager resumes the service after it has been paused.

Pascal

```
property OnContinue: TContinueEvent;
```

C++

```
__property TContinueEvent OnContinue;
```

Description

Write an OnContinue event handler to take specific action when the Service Control manager resumes the service after it has been paused.

See Also

OnPause (see page 22031)

156.1.4.6.6 TService.OnExecute Event

Occurs when the thread associated with the service starts up.

Pascal

```
property OnExecute: TServiceEvent;
```

C++

```
__property TServiceEvent OnExecute;
```

Description

If you are not spawning a new thread to handle individual service requests in an OnStart (see page 22031) event handler, this is where you implement the service. When the OnExecute event handler finishes, the service thread terminates. Most OnExecute event handlers contain a loop that calls the service thread's ProcessRequests method so that other service requests are not locked out.

See Also

ProcessRequests (see page 22046)

OnStart (see page 22031)

156.1.4.6.7 TService.OnPause Event

Occurs when the Service Control manager pauses the service temporarily.

Pascal

```
property OnPause: TPauseEvent;
```

C++

```
__property TPauseEvent OnPause;
```

Description

Write an OnPause event handler to take specific action when the service is temporarily paused.

See Also

OnContinue (see page 22030)

156.1.4.6.8 TService.OnShutdown Event

Occurs when the system running the service application is about to shut down.

Pascal

```
property OnShutdown: TServiceEvent;
```

C++

```
__property TServiceEvent OnShutdown;
```

Description

Write an OnShutdown event handler to take specific action just before the service shuts down. The service application terminates immediately after OnShutdown.

See Also

OnStop (see page 22032)

OnStart (see page 22031)

OnPause (see page 22031)

156.1.4.6.9 TService.OnStart Event

OnStartup occurs when the service first starts up, before the OnExecute (see page 22030) event.

Pascal

```
property OnStart: TStartEvent;
```

C++

```
__property TStartEvent OnStart;
```

Description

This event should be used to initialize the service. For example, if each service request is handled in a separate thread (a good idea if handling the request takes much time) the thread for a request is spawned in an OnStart event handler.

See Also

OnStop (see page 22032)

OnContinue (see page 22030)

OnExecute (see page 22030)

156.1.4.6.10 TService.OnStop Event

Occurs when the Service Control manager stops the service.

Pascal

```
property OnStop: TStopEvent;
```

C++

```
__property TStopEvent OnStop;
```

Description

Write an OnStop event handler to take specific action when the service stops.

See Also

OnShutdown (see page 22031)

OnStart (see page 22031)

OnPause (see page 22031)

156.1.5 TServiceApplication Class

TServiceApplication encapsulates a Windows NT service application.

Class Hierarchy

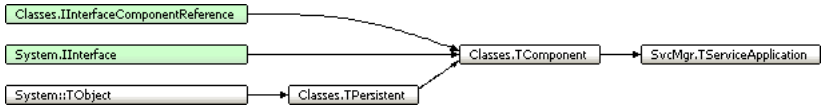

Pascal

```
TServiceApplication = class(TComponent);
```

C++

```
class TServiceApplication : public TComponent;
```

File

SvcMgr

Description

TServiceApplication encapsulates a Windows NT application type known as a service. TServiceApplication provides the fundamental behavior of this type of application. Each Service project automatically declares an Application (see page 22054) variable of type TServiceApplication as the instance of the application. The Application (see page 22054) variable is added to a project by selecting File | New from the menu of the IDE main window and selecting Service Application (see page 22054) in the New Items dialog.

TServiceApplication contains TService (see page 22011) objects, each of which encapsulates a Windows NT service. The service application object provides methods to create service objects and to install, register, dispatch, and uninstall services.

When the service application is run using the /INSTALL switch, it registers the services it contains. When run using the /UNINSTALL switch, it unregisters its services. Unless the application is run with the /SILENT switch, the service application shows a confirmation message after successfully installing or uninstalling its services.

Note: Do not add the Forms (see page 13657) or Httpapp units to the uses list of the project (Delphi) or includ the unit header for these units in the project source (C++). Those units declare conflicting Application (see page 22054) variables of types TApplication and TWebApplication, respectively.

See Also

Application (see page 22054)

TService (see page 22011)

Service Applications

Service Name\ Properties

156.1.5.1 TServiceApplication Members

The following tables list the members exposed by TServiceApplication.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DispatchServiceMain (see page 22037)	This is DispatchServiceMain, a member of class TServiceApplication.
	DoHandleException (see page 22037)	This is DoHandleException, a member of class TServiceApplication.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.

	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Hook (see page 22038)	This is Hook, a member of class TServiceApplication.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RegisterServices (see page 22038)	This is RegisterServices, a member of class TServiceApplication.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 22036)	Instantiates a TServiceApplication (see page 22032) object.
V	CreateForm (see page 22037)	Creates the service object or objects for the service application.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22036)	Disposes of an instance of a TServiceApplication (see page 22032) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.

	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 22038)	Sets the initial state of the application and hides any associated MainForm.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	Installing (see page 22038)	This is Installing, a member of class TServiceApplication.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Run (see page 22039)	Installs, registers, and adds services to its component list, and creates a thread for the service application.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DelayInitialize (see page 22039)	This is DelayInitialize, a member of class TServiceApplication.
	DesignInfo (see page 2805)	Contains information used by the Form designer.

R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	ServiceCount (see page 22039)	Indicates the number of services in the service application.
	Title (see page 22040)	Indicates the name of the title application.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

156.1.5.2 TServiceApplication.Create Constructor

Instantiates a TServiceApplication (see page 22032) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TServiceApplication(TComponent * AOwner);
```

Description

Applications do not need to call the constructor directly. TServiceApplication (see page 22032) is instantiated by selecting File | New from the menu of the IDE main window and selecting Service Application (see page 22054) in the New Items dialog. An Application (see page 22054) variable of type TServiceApplication (see page 22032) is declared and initialized as the instance of the application.

156.1.5.3 TServiceApplication.Destroy Destructor

Disposes of an instance of a TServiceApplication (see page 22032) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TServiceApplication();
```

Description

You do not need to call the destructor directly. TServiceApplication (see page 22032) is destroyed when your application closes.

The destructor frees the memory associated with sub-objects.

See Also

Free (see page 22300)

156.1.5.4 TServiceApplication Methods

The methods of the TServiceApplication class are listed here.

Protected Methods

	Name	Description
	DispatchServiceMain (see page 22037)	This is DispatchServiceMain, a member of class TServiceApplication.
	DoHandleException (see page 22037)	This is DoHandleException, a member of class TServiceApplication.
	Hook (see page 22038)	This is Hook, a member of class TServiceApplication.
	RegisterServices (see page 22038)	This is RegisterServices, a member of class TServiceApplication.

Public Methods

	Name	Description
V	CreateForm (see page 22037)	Creates the service object or objects for the service application.
V	Initialize (see page 22038)	Sets the initial state of the application and hides any associated MainForm.
	Installing (see page 22038)	This is Installing, a member of class TServiceApplication.
V	Run (see page 22039)	Installs, registers, and adds services to its component list, and creates a thread for the service application.

156.1.5.4.1 TServiceApplication.CreateForm Method

Creates the service object or objects for the service application.

Pascal

```
procedure CreateForm(InstanceClass: TComponentClass; var Reference); virtual;
```

C++

```
virtual __fastcall CreateForm(TComponentClass InstanceClass,  Reference);
```

Description

Do not call CreateForm in an application. The module manager calls CreateForm internally to register the service objects of the service application.

CreateForm instantiates a TService (see page 22011) object for each service contained in the service application. CreateForm is called automatically, for the first service, when the service application is first created. Subsequently, CreateForm is called for each new service added to the application. To add a new service, select File | New from the menu of the IDE main window and choose Service in the New Items dialog.

156.1.5.4.2 TServiceApplication.DispatchServiceMain Method

Pascal

```
procedure DispatchServiceMain(Argc: DWord; Argv: PLPSTR);
```

C++

```
__fastcall DispatchServiceMain(DWord Argc, PLPSTR Argv);
```

Description

This is DispatchServiceMain, a member of class TServiceApplication.

156.1.5.4.3 TServiceApplication.DoHandleException Method

Pascal

```
procedure DoHandleException(E: Exception); dynamic;
```

C++

```
__fastcall DoHandleException(Exception E);
```

Description

This is DoHandleException, a member of class TServiceApplication.

156.1.5.4.4 TServiceApplication.Hook Method

Pascal

```
function Hook(var Message: TMessage): Boolean;
```

C++

```
__fastcall Boolean Hook(TMessage Message);
```

Description

This is Hook, a member of class TServiceApplication.

156.1.5.4.5 TServiceApplication.Initialize Method

Sets the initial state of the application and hides any associated MainForm.

Pascal

```
procedure Initialize; virtual;
```

C++

```
virtual __fastcall Initialize();
```

Description

Initialize sets the initial state of the application to hide any main form that may be associated with the application. Service applications are rarely interactive and do not typically have a form associated with them. If a form has been added to the application, Initialize hides it on start. Initialize then follows the standard (default) VCL application initialization process.

156.1.5.4.6 TServiceApplication.Installing Method

Pascal

```
function Installing: Boolean;
```

C++

```
__fastcall Boolean Installing();
```

Description

This is Installing, a member of class TServiceApplication.

156.1.5.4.7 TServiceApplication.RegisterServices Method

Pascal

```
procedure RegisterServices(Install: Boolean; Silent: Boolean);
```

C++

```
__fastcall RegisterServices(Boolean Install, Boolean Silent);
```

Description

This is RegisterServices, a member of class TServiceApplication.

156.1.5.4.8 TServiceApplication.Run Method

Installs, registers, and adds services to its component list, and creates a thread for the service application.

Pascal

```
procedure Run; virtual;
```

C++

```
virtual __fastcall Run();
```

Description

Run handles installing and uninstalling services, depending on the command line switches. When the service application is run using the /INSTALL switch, it registers the services it contains. When run using the /UNINSTALL switch, it unregisters its services. Unless the application is run with the /SILENT switch, the service application shows a confirmation message after successfully installing or uninstalling its services.

If not called with the /INSTALL or /UNINSTALL switch, Run creates a service application thread that responds to Windows messages from clients. Note that this thread is for the application and differs from the dedicated TServiceThread (see page 22040) instances associated with individual services in the application.

156.1.5.5 TServiceApplication Properties

The properties of the TServiceApplication class are listed here.

Public Properties

	Name	Description
	DelayInitialize (see page 22039)	This is DelayInitialize, a member of class TServiceApplication.
R	ServiceCount (see page 22039)	Indicates the number of services in the service application.
	Title (see page 22040)	Indicates the name of the title application.

156.1.5.5.1 TServiceApplication.DelayInitialize Property

Pascal

```
property DelayInitialize: Boolean;
```

C++

```
__property Boolean DelayInitialize;
```

Description

This is DelayInitialize, a member of class TServiceApplication.

156.1.5.5.2 TServiceApplication.ServiceCount Property

Indicates the number of services in the service application.

Pascal

```
property ServiceCount: Integer;
```

C++

```
__property int ServiceCount;
```

Description

ServiceCount is the number of services in the service application. Each service is encapsulated by a TService (see page 22011) instance and has its own dedicated service thread.

Note: If the Service application requires a user name and password, the application must implement only a single service.

See Also

TService (see page 22011)

156.1.5.5.3 TServiceApplication.Title Property

Indicates the name of the title application.

Pascal

```
property Title: string;
```

C++

```
__property AnsiString Title;
```

Description

Title is the name of the service application. This text string appears below the icon of the application when it is minimized.

156.1.6 TServiceThread Class

TServiceThread represents the thread associated with a specific service (TService (see page 22011) object).

Class Hierarchy

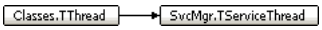

Pascal

```
TServiceThread = class(TThread);
```

C++

```
class TServiceThread : public TThread;
```

File

SvcMgr

Description

TServiceThread is the thread associated with a service. Each service application has one or more service objects, each of which has its own dedicated thread. TServiceThread instances represent those threads.

Note: Service threads.

See Also

TService (see page 22011)

TService (see page 22011)

TServiceApplication (see page 22032)

TServiceThread

IDH

Service Threads

156.1.6.1 TServiceThread Members

The following tables list the members exposed by TServiceThread.

Protected Methods

	Name	Description
	CheckThreadError (see page 3114)	This is the overview for the CheckThreadError method overload.
V	DoTerminate (see page 3115)	Generates an OnTerminate (see page 3148) event.
V	Execute (see page 22042)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Protected Properties

	Name	Description
	ReturnValue (see page 3141)	Specifies the value returned to other waiting threads when the thread finishes executing.
R	Terminated (see page 3142)	Indicates whether the thread has been asked to terminate.

Public Events

	Name	Description
	OnTerminate (see page 3148)	Occurs after the thread's Execute (see page 3115) method has returned and before the thread is destroyed.

Public Methods

	Name	Description
V	AfterConstruction (see page 3114)	Executes the thread when the thread is created with CreateSuspended set to false.
	Create (see page 22042)	Instantiates and initializes a TServiceThread (see page 22040) object.
V	Destroy (see page 3113)	Destroys the thread object and releases the memory allocated to it.
	ProcessRequests (see page 22046)	Allows clients to invoke the service associated with the service thread.
	Queue (see page 3119)	This is the overview for the Queue method overload.
	RemoveQueuedEvents (see page 3120)	Remove queued method calls.
	Resume (see page 3120)	Restarts the execution of a suspended thread.
	StaticQueue (see page 3124)	Asynchronously executes a method call within the main thread.
	StaticSynchronize (see page 3125)	Executes a method call within the main thread.
	Suspend (see page 3125)	Pauses a running thread.
	Synchronize (see page 3126)	This is the overview for the Synchronize method overload.
	Terminate (see page 3126)	Signals the thread to terminate by setting the Terminated (see page 3142) property to true.
	WaitFor (see page 3132)	Waits for the thread to terminate and then returns the value of the ReturnValue (see page 3141) property.

Public Properties

	Name	Description
R	FatalException (see page 3133)	Determines whether the thread object is automatically destroyed when the thread terminates.
	FreeOnTerminate (see page 3134)	Determines whether the thread object is automatically destroyed when the thread terminates.
R	Handle (see page 3137)	Indicates the thread's handle (Windows only).
	Priority (see page 3138)	Determines the thread's scheduling priority relative to other threads in the process.
	Suspended (see page 3142)	Indicates whether a thread is suspended.
R	ThreadID (see page 3148)	Identifies the thread throughout the system.

156.1.6.2 TServiceThread.Create Constructor

Instantiates and initializes a TServiceThread (see page 22040) object.

Pascal

```
constructor Create(Service: TService);
```

C++

```
__fastcall TServiceThread(TService Service);
```

Description

Applications do not need to call the constructor directly. Instances of TService (see page 22011) create the TServiceThread (see page 22040) object on startup. The Service parameter is the TService (see page 22011) instance for which the TServiceThread (see page 22040) is the dedicated thread.

The constructor sets the FreeOnTerminate property to true and then calls the base class constructor so that the thread object is created with its Suspended property set to false.

See Also

FreeOnTerminate (see page 3134)

Suspended (see page 3142)

156.1.6.3 TServiceThread Methods

The methods of the TServiceThread class are listed here.

Protected Methods

	Name	Description
	Execute (see page 22042)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Public Methods

	Name	Description
	ProcessRequests (see page 22046)	Allows clients to invoke the service associated with the service thread.

156.1.6.3.1 TServiceThread.Execute Method

Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Pascal

```
procedure Execute; override;
```

C++

```
virtual __fastcall Execute();
```

Description

Override Execute and insert the code that should be executed when the thread runs. Execute is responsible for checking the value of the Terminated property to determine if the thread needs to exit.

A thread executes when Create is called if CreateSuspended set to false, or when Resume is first called after the thread is created if CreateSuspended set to true.

Note: Do not use the properties and methods of other objects directly in the Execute method of a thread. Instead, separate the

use of other objects into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize method.

C++ Examples:

```
/*
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes. To
create the extension choose the
File | New | Other | Thread Object menu.
*/
bool mythreadRunning;
bool youthreadRunning;

class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
}

class TYouThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};

__fastcall TYouThread::TYouThread(bool suspended)
    : TThread(suspended)
{
}

void __fastcall TMyThread::Execute()
{
  Form1->Memo1->Lines->Add("Process has been running for this many seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo1->Lines->Add("Low priority process " + IntToStr(I));
    Sleep(1000);
  }
  mythreadRunning = FALSE;
}

void __fastcall TYouThread::Execute()
{
  Form1->Memo3->Lines->Add("Second low priority process has been running for this many
seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo3->Lines->Add("Second low priority process " + IntToStr(I));
```

```
    Sleep(1000);
  }
  youthreadRunning = FALSE;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMyThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    SecondProcess = new TMyThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form1->Memo2->Lines->Add("Do some work in the main process for 10 seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo2->Lines->Add("Main process " + IntToStr(I));
    Sleep(1000);
  }
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  TYouThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    SecondProcess = new TYouThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  mythreadRunning = FALSE;
  youthreadRunning = FALSE;
}
```

Delphi Examples:

```
{
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread’s FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes.  The
Execute procedure must be overridden in the extension class
of TThread or an "Abstract Error" will result. To create the
extension choose the File | New | Other | Thread Object menu.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (mythreadRunning = FALSE) then
```

```
  begin
    mythreadRunning:= TRUE;
    MyProcess := TMyThread.Create(True); { create suspended - secondprocess does not run yet }
    MyProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    MyProcess.Priority := tpLower;  // set the priority to lower than normal
    MyProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  if mythreadRunning then MyProcess.Terminate();
end;

procedure TMyThread.Execute;
var I: Integer;
begin
  Form1.Memo1.Lines.Add('Process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo1.Lines.Add('Low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
  mythreadRunning:= FALSE;
end;

procedure TYouThread.Execute;
var I: Integer;
begin
  Form1.Memo3.Lines.Add('Second low priority process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo3.Lines.Add('Second low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
    youthreadRunning:= FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
var   I: Integer;
begin
  Form1.Memo2.Lines.Add('Do some work in the main process for 10 seconds:');
  for I := 0 to 10 do
  begin
    Form1.Memo2.Lines.Add('Main process ' + InttoStr(I));
    Sleep(1000);
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  if (youthreadRunning = FALSE) then
  begin
    youthreadRunning:= TRUE;
    YouProcess := TYouThread.Create(True); { create suspended – secondprocess does not run yet
}
    YouProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    YouProcess.Priority := tpLower;  // set the priority to lower than normal
    YouProcess.Resume; { now run the thread }
  end
  else
```

```
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  if youthreadRunning then YouProcess.Terminate();
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  mythreadRunning:= FALSE;
  youthreadRunning:= FALSE;
end;
```

See Also

Create (see page 3110)

Terminated (see page 3142)

Synchronize (see page 3126)

Checking for Termination by Other Threads

Waiting for a Thread to Finish Executing

Writing Clean-up Code

Writing the Thread Function

156.1.6.3.2 TServiceThread.ProcessRequests Method

Allows clients to invoke the service associated with the service thread.

Pascal

```
procedure ProcessRequests(WaitForMessage: Boolean);
```

C++

```
__fastcall ProcessRequests(Boolean WaitForMessage);
```

Description

TServiceThread (see page 22040)'s ProcessRequests is called in the Execute (see page 22042) method after any associated service's OnExecute event handler has finished executing. For more information about optimizing service execution by spawning individual threads for each service request, see the Developer's Guide.

See Also

OnExecute (see page 22030)

156.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCurrentStatus (see page 22047)	TCurrentStatus indicates the state of a service.
	TErrorSeverity (see page 22048)	TErrorSeverity indicates the severity of a failure and implies the corresponding response by a service startup program.
	TServiceType (see page 22048)	TServiceType indicates the type of a service.
	TStartType (see page 22049)	TStartType indicates how a service starts up.

156.2.1 SvcMgr.TCurrentStatus Enumeration

TCurrentStatus indicates the state of a service.

Pascal

```
TCurrentStatus = (
  csStopped,
  csStartPending,
  csStopPending,
  csRunning,
  csContinuePending,
  csPausePending,
  csPaused
);
```

C++

```
enum TCurrentStatus {
  csStopped,
  csStartPending,
  csStopPending,
  csRunning,
  csContinuePending,
  csPausePending,
  csPaused
};
```

File

SvcMgr

Description

TCurrentStatus can have one of the following values:

Value	Meaning
csStopped	The service is not running.
csStartPending	The service is starting.
csStopPending	The service is stopping.
csRunning	The service is running.

csContinuePending	The service continue is pending.
csPausePending	The service pause is pending.
csPaused	The service is paused

156.2.2 SvcMgr.TErrorSeverity Enumeration

TErrorSeverity indicates the severity of a failure and implies the corresponding response by a service startup program.

Pascal

```
TErrorSeverity = (
  esIgnore,
  esNormal,
  esSevere,
  esCritical
);
```

C++

```
enum TErrorSeverity {
  esIgnore,
  esNormal,
  esSevere,
  esCritical
};
```

File

SvcMgr

Description

TErrorSeverity can have one of the following values:

Value	What the startup program does
esIgnore	Logs the error and continues.
esNormal	Logs the error and displays a message, but continues.
esSevere	Logs the error but continues only if the last-known-good configuration is being started; otherwise the last-known-good configuration is started instead.
esCritical	Logs the error and starts up the last-known-good configuration instead. If this is the last-known-good configuration, the startup operation fails.

156.2.3 SvcMgr.TServiceType Enumeration

TServiceType indicates the type of a service.

Pascal

```
TServiceType = (
  stWin32,
  stDevice,
  stFileSystem
);
```

C++

```
enum TServiceType {
  stWin32,
  stDevice,
  stFileSystem
};
```

File

SvcMgr

Description

TServiceType can have one of the following values:

Value	Meaning
stWin32	a Win32 service.
stDevice	a device driver.
stFileSystem	a file system driver.

156.2.4 SvcMgr.TStartType Enumeration

TStartType indicates how a service starts up.

Pascal

```
TStartType = (
  stBoot,
  stSystem,
  stAuto,
  stManual,
  stDisabled
);
```

C++

```
enum TStartType {
  stBoot,
  stSystem,
  stAuto,
  stManual,
  stDisabled
};
```

File

SvcMgr

Description

The following table lists the possible values of TStartType:

StartType value	Service is started
stBoot(used only when ServiceType is not stWin32)	by the operating system loader
stSystem(used only when ServiceType is not stWin32)	after boot systems are initialized
stAuto	automatically during system startup

stManual	when a process calls the Windows StartService function
stDisabled	the service can't be started except by an administrator

156.3 Types

The following table lists types in this documentation.

Types

Name	Description
TContinueEvent (see page 22051)	This is type SvcMgr.TContinueEvent.
TPauseEvent (see page 22051)	This is type SvcMgr.TPauseEvent.
TServiceController (see page 22052)	This is type SvcMgr.TServiceController.
TServiceEvent (see page 22052)	This is type SvcMgr.TServiceEvent.
TStartEvent (see page 22052)	This is type SvcMgr.TStartEvent.
TStopEvent (see page 22053)	This is type SvcMgr.TStopEvent.

156.3.1 SvcMgr.TContinueEvent Type

Pascal

```
TContinueEvent = procedure (Sender: TService; var Continued: Boolean) of object;
```

C++

```
(Sender: TService; var Continued: Boolean) ( TContinueEvent)();
```

File

SvcMgr

Description

This is type SvcMgr.TContinueEvent.

156.3.2 SvcMgr.TPauseEvent Type

Pascal

```
TPauseEvent = procedure (Sender: TService; var Paused: Boolean) of object;
```

C++

```
(Sender: TService; var Paused: Boolean) ( TPauseEvent)();
```

File

SvcMgr

Description

This is type SvcMgr.TPauseEvent.

156.3.3 SvcMgr.TServiceController Type

Pascal

```
TServiceController = procedure (CtrlCode: DWord);
```

C++

```
(CtrlCode: DWord) ( TServiceController)();
```

File

SvcMgr

Description

This is type SvcMgr.TServiceController.

156.3.4 SvcMgr.TServiceEvent Type

Pascal

```
TServiceEvent = procedure (Sender: TService) of object;
```

C++

```
(Sender: TService) ( TServiceEvent)();
```

File

SvcMgr

Description

This is type SvcMgr.TServiceEvent.

156.3.5 SvcMgr.TStartEvent Type

Pascal

```
TStartEvent = procedure (Sender: TService; var Started: Boolean) of object;
```

C++

```
(Sender: TService; var Started: Boolean) ( TStartEvent)();
```

File

SvcMgr

Description

This is type SvcMgr.TStartEvent.

156.3.6 SvcMgr.TStopEvent Type

Pascal

```
TStopEvent = procedure (Sender: TService; var Stopped: Boolean) of object;
```

C++

```
(Sender: TService; var Stopped: Boolean) ( TStopEvent)();
```

File

SvcMgr

Description

This is type SvcMgr.TStopEvent.

156.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
Application (see page 22054)	This is variable SvcMgr.Application.

156.4.1 SvcMgr.Application Variable

Pascal

```
Application: TServiceApplication = nil;
```

C++

```
TServiceApplication Application = nil;
```

File

SvcMgr

Description

This is variable SvcMgr.Application.

156.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CM_SERVICE_CONTROL_CODE (see page 22055)	This is constant SvcMgr.CM_SERVICE_CONTROL_CODE.

156.5.1 SvcMgr.CM_SERVICE_CONTROL_CODE Constant

Pascal

```
CM_SERVICE_CONTROL_CODE = WM_USER + 1;
```

C++

```
CM_SERVICE_CONTROL_CODE = WM_USER + 1;
```

File

SvcMgr

Description

This is constant SvcMgr.CM_SERVICE_CONTROL_CODE.

157 SyncObjs Namespace

This is namespace SyncObjs.

Topics

Name	Description
Classes (see page 22058)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 22078)	The following table lists structs, records, enums in this documentation.

Classes

	Name	Description
	TCriticalSection (see page 22058)	TCriticalSection allows a thread in a multi-threaded application to temporarily block other threads from accessing a block of code.
	TEvent (see page 22063)	TEvent represents an external event.
	THandleObject (see page 22066)	THandleObject is the base class for all thread synchronization objects that use a handle.
	TMutex (see page 22071)	This is class SyncObjs.TMutex.
	TSimpleEvent (see page 22074)	TSimpleEvent represents an unnamed manual event object.
	TSynchroObject (see page 22075)	TSynchroObject is the base class for all thread synchronization objects.

Structs, Records, Enums

	Name	Description
	TWaitResult (see page 22078)	TWaitResult indicates the reason an event object stopped waiting for its signal.

157.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCriticalSection (see page 22058)	TCriticalSection allows a thread in a multi-threaded application to temporarily block other threads from accessing a block of code.
	TEvent (see page 22063)	TEvent represents an external event.
	THandleObject (see page 22066)	THandleObject is the base class for all thread synchronization objects that use a handle.
	TMutex (see page 22071)	This is class SyncObjs.TMutex.
	TSimpleEvent (see page 22074)	TSimpleEvent represents an unnamed manual event object.
	TSynchroObject (see page 22075)	TSynchroObject is the base class for all thread synchronization objects.

157.1.1 TCriticalSection Class

TCriticalSection allows a thread in a multi-threaded application to temporarily block other threads from accessing a block of code.

Class Hierarchy

Pascal

```
TCriticalSection = class(TSynchroObject);
```

C++

```
class TCriticalSection : public TSynchroObject;
```

File

SyncObjs

Description

Use TCriticalSection to safeguard operations that may be disrupted if another thread executes certain sections of code before the operation is completed. Critical sections work like gates that allow only a single thread to enter at a time. Because they block the execution of other threads, overusing critical sections can seriously interfere with performance.

Critical sections must be global in scope so that they are available to all threads. In every thread, any action that could interfere with the operations safeguarded by the critical section should only take place after calling the Acquire (see page 22061) or Enter (see page 22061) method. Threads that do not use the critical section for relevant operations can introduce bugs.

See Also

TThread (see page 3108)

TCOMCriticalSection

Using Critical Sections

157.1.1.1 TCriticalSection Members

The following tables list the members exposed by TCriticalSection.

Protected Fields

	Name	Description
	FSection (see page 22060)	This is FSection, a member of class TCriticalSection.

Public Methods

	Name	Description
V	Acquire (see page 22061)	Binds the critical section to the calling thread.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22059)	Creates an instance of TCriticalSection (see page 22058).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22060)	Destroys an instance of TCriticalSection (see page 22058).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Enter (see page 22061)	Blocks other threads when the calling thread enters a thread-sensitive section.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Leave (see page 22062)	Allows other threads to use the critical section.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Release (see page 22062)	Allows other threads to use the critical section.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	TryEnter (see page 22062)	This is TryEnter, a member of class TCriticalSection.

157.1.1.2 TCriticalSection.Create Constructor

Creates an instance of TCriticalSection (see page 22058).

Pascal

```
constructor Create;
```

C++

```
__fastcall TCriticalSection();
```

Description

Call Create to create and initialize an instance of TCriticalSection (see page 22058). A TCriticalSection (see page 22058) object can be created for a single object that contains thread-sensitive code, or as a global object to be shared by several threads.

See Also

Create (see page 22287)

157.1.1.3 TCriticalSection.Destroy Destructor

Destroys an instance of TCriticalSection (see page 22058).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCriticalSection();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TCriticalSection (see page 22058) object is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

157.1.1.4 TCriticalSection Fields

The fields of the TCriticalSection class are listed here.

Protected Fields

	Name	Description
	FSection (see page 22060)	This is FSection, a member of class TCriticalSection.

157.1.1.4.1 TCriticalSection.FSection Field

Pascal

```
FSection: TRTLCriticalSection;
```

C++

```
TRTLCriticalSection FSection;
```

Description

This is FSection, a member of class TCriticalSection.

157.1.1.5 TCriticalSection Methods

The methods of the TCriticalSection class are listed here.

Public Methods

	Name	Description
	Acquire (see page 22061)	Binds the critical section to the calling thread.
	Enter (see page 22061)	Blocks other threads when the calling thread enters a thread-sensitive section.
	Leave (see page 22062)	Allows other threads to use the critical section.
	Release (see page 22062)	Allows other threads to use the critical section.
	TryEnter (see page 22062)	This is TryEnter, a member of class TCriticalSection.

157.1.1.5.1 TCriticalSection.Acquire Method

Binds the critical section to the calling thread.

Pascal

```
procedure Acquire; override;
```

C++

```
virtual __fastcall Acquire();
```

Description

Call Acquire to block all other threads from acquiring this critical section until the Release (see page 22062) or Leave (see page 22062) method is called. Acquire does the same thing as the Enter (see page 22061) method.

See Also

Enter (see page 22061)

Leave (see page 22062)

Release (see page 22062)

Using Critical Sections

157.1.1.5.2 TCriticalSection.Enter Method

Blocks other threads when the calling thread enters a thread-sensitive section.

Pascal

```
procedure Enter;
```

C++

```
__fastcall Enter();
```

Description

Call Enter to block all other threads from entering code protected by this critical section until the Leave (see page 22062) or Release (see page 22062) method is called. Enter calls the Acquire (see page 22061) method to bind the critical section to the calling thread.

See Also

Acquire (see page 22061)

Leave (see page 22062)

Release (see page 22062)

157.1.1.5.3 TCriticalSection.Leave Method

Allows other threads to use the critical section.

Pascal

```
procedure Leave;
```

C++

```
__fastcall Leave();
```

Description

Call Leave at the end of a thread-sensitive section of code. Each call to Leave should be balanced by a call to Enter (see page 22061) or Acquire (see page 22061) at the beginning of the thread-sensitive section. Leave calls Release (see page 22062) to allow other threads to acquire the critical section.

See Also

Acquire (see page 22061)

Enter (see page 22061)

Release (see page 22062)

157.1.1.5.4 TCriticalSection.Release Method

Allows other threads to use the critical section.

Pascal

```
procedure Release; override;
```

C++

```
virtual __fastcall Release();
```

Description

Call Release when the thread no longer needs to block other threads from executing code that is protected by this critical section. Each call to Release should be balance by an earlier call to Acquire (see page 22061) or Enter (see page 22061) that blocked other threads from executing. Release does the same thing as the Leave (see page 22062) method.

See Also

Acquire (see page 22061)

Enter (see page 22061)

Leave (see page 22062)

Using Critical Sections

157.1.1.5.5 TCriticalSection.TryEnter Method

Pascal

```
function TryEnter: Boolean;
```

C++

```
__fastcall Boolean TryEnter();
```

Description

This is TryEnter, a member of class TCriticalSection.

157.1.2 TEvent Class

TEvent represents an external event.

Class Hierarchy

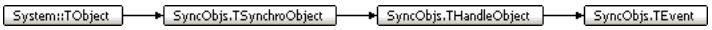

Pascal

```
TEvent = class(THandleObject);
```

C++

```
class TEvent : public THandleObject;
```

File

SyncObjs

Description

Use TEvent to signal that an event has occurred or a state has been reached.

In a multi-threaded application, use TEvent to allow one thread to signal to other threads that an event has occurred.

The handle of a TEvent object can also be used to communicate with other processes, so that an application can coordinate the timing of events with other applications. For example, use the handle of a TEvent object to wait until another process is ready to transfer information.

In a single-threaded application, use TEvent to coordinate between sections of code that respond to asynchronous events such as system events.

See Also

TThread (see page 3108)

Waiting for a Task to Be Completed

157.1.2.1 TEvent Members

The following tables list the members exposed by TEvent.

Protected Fields

	Name	Description
	FHandle (see page 22069)	This is FHandle, a member of class THandleObject.
	FLastError (see page 22069)	This is FLastError, a member of class THandleObject.
	FUseCOMWait (see page 22069)	This is FUseCOMWait, a member of class THandleObject.

Public Methods

	Name	Description
	Acquire (see page 22076)	Provides the interface for a method that devotes the synchronization object to a thread.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22064)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22068)	Destroys an instance of THandleObject (see page 22066).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Release (see page 22076)	Provides the interface for a method that releases the synchronization object for use by other threads.
	ResetEvent (see page 22066)	Turns off the signal of the event object.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEvent (see page 22066)	Turns on the signal of the event object.
V	WaitFor (see page 22070)	This is WaitFor, a member of class THandleObject.

Public Properties

	Name	Description
R	Handle (see page 22070)	Specifies the handle of the Windows object used by THandleObject (see page 22066).
R	LastError (see page 22070)	Indicates the Windows error code obtained from Handle (see page 22070).

157.1.2.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TEvent.Create (Boolean) (see page 22065)	Instantiates a TEvent (see page 22063) object to represent an event object.
	TEvent.Create (PSecurityAttributes, Boolean, Boolean, string, Boolean) (see page 22065)	Instantiates a TEvent (see page 22063) object to represent an event object.

157.1.2.2.1 TEvent.Create Constructor (Boolean)

Instantiates a TEvent (see page 22063) object to represent an event object.

Pascal

```
constructor Create(EventAttributes: PSecurityAttributes; ManualReset: Boolean; InitialState:
Boolean; const Name: string; UseCOMWait: Boolean = False); overload;
constructor Create(UseCOMWait: Boolean = False); overload;
```

C++

```
__fastcall TEvent(PSecurityAttributes EventAttributes, Boolean ManualReset, Boolean
InitialState, const AnsiString Name, Boolean UseCOMWait = False);
__fastcall TEvent(Boolean UseCOMWait = False);
```

Description

Call Create to create a TEvent (see page 22063) object. Create can generate a new event object or can provide access to an existing named event object.

Set the EventAttributes parameter to specify the access rights and inheritance capabilities of the new event object. Calling Create with EventAttributes set to nil (Delphi) or NULL (C++) creates an event object with a default set of security attributes. If the TEvent (see page 22063) object is created to access an existing named event object, only the bInheritHandle field of EventAttributes is used, as the access rights were determined when the event object was created.

Set ManualReset to specify whether the signal of the TEvent (see page 22063) object can only be turned off by a call to the ResetEvent (see page 22066) method, or whether it is automatically reset when a single waiting thread is released. When ManualReset is true, the TEvent (see page 22063) signal stays set until the ResetEvent (see page 22066) method turns it off. When ManualReset is false, the signal is automatically turned off when only a single thread waits on the signal and that thread is released. If the TEvent (see page 22063) object is created to access an existing named event object, the ManualReset parameter is ignored.

Set InitialState to indicate whether the TEvent (see page 22063) object should be created with the signal set, or turned off. When InitialState is true, the TEvent (see page 22063) object is created with the signal set. If the TEvent (see page 22063) object is created to access an existing named event object, the InitialState parameter is ignored.

Set Name to provide a name for a new event object or to specify an existing named event object. If no other thread or process will need to access the event object to wait for its signal, Name can be left blank. Name can be a string of up to 260 characters, not including the backslash character (\). If Name is used to specify an existing event object, the value must match the name of the existing event in a case-sensitive comparison. If Name matches the name of an existing semaphore, mutex, or file-mapping object, the TEvent (see page 22063) object will be created with Handle (see page 22070) set to 0 and all method calls will fail.

See Also

Handle (see page 22070)

ResetEvent (see page 22066)

157.1.2.3 TEvent Methods

The methods of the TEvent class are listed here.

Public Methods

	Name	Description
	ResetEvent (see page 22066)	Turns off the signal of the event object.
	SetEvent (see page 22066)	Turns on the signal of the event object.

157.1.2.3.1 TEvent.ResetEvent Method

Turns off the signal of the event object.

Pascal

```
procedure ResetEvent;
```

C++

```
__fastcall ResetEvent();
```

Description

Call ResetEvent to signal that the event has not occurred or the desired state was not achieved. Other threads or processes that wait for the event will wait until the signal is turned on again by the SetEvent (see page 22066) method, the event object is destroyed, or the waiting process times out.

See Also

SetEvent (see page 22066)

WaitFor

Waiting for a Task to Be Completed

157.1.2.3.2 TEvent.SetEvent Method

Turns on the signal of the event object.

Pascal

```
procedure SetEvent;
```

C++

```
__fastcall SetEvent();
```

Description

Call SetEvent to signal that the event has occurred or the desired state was achieved. Other threads or processes that wait for the event will resume execution with a return value indicating that the signal was set.

See Also

ResetEvent (see page 22066)

WaitFor

Waiting for a Task to Be Completed

157.1.3 THandleObject Class

THandleObject is the base class for all thread synchronization objects that use a handle.

Class Hierarchy

Pascal

```
THandleObject = class(TSynchroObject);
```

C++

```
class THandleObject : public TSynchroObject;
```

File

SyncObjs

Description

Use THandleObject as a base class when defining an object that uses a handle when coordinating the execution of different threads in a multi-threaded application. Handled synchronization objects include event objects that allow threads to signal when critical events have occurred.

Do not create instances of THandleObject. THandleObject is a base class that introduces properties for managing the handle of a synchronization object but does not provide a mechanism for acquiring that handle. Descendants of THandleObject must acquire the appropriate type of handle.

See Also

TThread (see page 3108)

TEvent (see page 22063)

157.1.3.1 THandleObject Members

The following tables list the members exposed by THandleObject.

Protected Fields

	Name	Description
	FHandle (see page 22069)	This is FHandle, a member of class THandleObject.
	FLastError (see page 22069)	This is FLastError, a member of class THandleObject.
	FUseCOMWait (see page 22069)	This is FUseCOMWait, a member of class THandleObject.

Public Methods

	Name	Description
V	Acquire (see page 22076)	Provides the interface for a method that devotes the synchronization object to a thread.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22068)	This is Create, a member of class THandleObject.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22068)	Destroys an instance of THandleObject (see page 22066).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Release (see page 22076)	Provides the interface for a method that releases the synchronization object for use by other threads.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	WaitFor (see page 22070)	This is WaitFor, a member of class THandleObject.

Public Properties

	Name	Description
R	Handle (see page 22070)	Specifies the handle of the Windows object used by THandleObject (see page 22066).
R	LastError (see page 22070)	Indicates the Windows error code obtained from Handle (see page 22070).

157.1.3.2 THandleObject.Create Constructor

Pascal

```
constructor Create(UseCOMWait: Boolean = False);
```

C++

```
__fastcall THandleObject(Boolean UseCOMWait = False);
```

Description

This is Create, a member of class THandleObject.

157.1.3.3 THandleObject.Destroy Destructor

Destroys an instance of THandleObject (see page 22066).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THandleObject();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the THandleObject (see page 22066) reference is not nil and only then calls Destroy.

See Also

Free (see page 22300)

Handle (see page 22070)

157.1.3.4 THandleObject Fields

The fields of the THandleObject class are listed here.

Protected Fields

	Name	Description
	FHandle (see page 22069)	This is FHandle, a member of class THandleObject.
	FLastError (see page 22069)	This is FLastError, a member of class THandleObject.
	FUseCOMWait (see page 22069)	This is FUseCOMWait, a member of class THandleObject.

157.1.3.4.1 THandleObject.FHandle Field

Pascal

```
FHandle: THandle;
```

C++

```
THandle FHandle;
```

Description

This is FHandle, a member of class THandleObject.

157.1.3.4.2 THandleObject.FLastError Field

Pascal

```
FLastError: Integer;
```

C++

```
int FLastError;
```

Description

This is FLastError, a member of class THandleObject.

157.1.3.4.3 THandleObject.FUseCOMWait Field

Pascal

```
FUseCOMWait: Boolean;
```

C++

```
Boolean FUseCOMWait;
```

Description

This is FUseCOMWait, a member of class THandleObject.

157.1.3.5 THandleObject Methods

The methods of the THandleObject class are listed here.

Public Methods

	Name	Description
	WaitFor (see page 22070)	This is WaitFor, a member of class THandleObject.

157.1.3.5.1 THandleObject.WaitFor Method

Pascal

```
function WaitFor(Timeout: LongWord): TWaitResult; virtual;
```

C++

```
virtual __fastcall TWaitResult WaitFor(LongWord Timeout);
```

Description

This is WaitFor, a member of class THandleObject.

157.1.3.6 THandleObject Properties

The properties of the THandleObject class are listed here.

Public Properties

	Name	Description
R	Handle (see page 22070)	Specifies the handle of the Windows object used by THandleObject (see page 22066).
R	LastError (see page 22070)	Indicates the Windows error code obtained from Handle (see page 22070).

157.1.3.6.1 THandleObject.Handle Property

Specifies the handle of the Windows object used by THandleObject (see page 22066).

Pascal

```
property Handle: THandle;
```

C++

```
__property THandle Handle;
```

Description

Read Handle to obtain the Windows handle for the synchronization object. Use Handle when making Windows API calls that require a handle to the synchronization object.

THandleObject (see page 22066) provides no mechanism for setting the Handle property. Descendants of THandleObject (see page 22066) set the Handle property to the handle for whatever Windows synchronization object they represent.

See Also

Destroy (see page 22068)

157.1.3.6.2 THandleObject.LastError Property

Indicates the Windows error code obtained from Handle (see page 22070).

Pascal

```
property LastError: Integer;
```

C++

```
__property int LastError;
```

Description

Read LastError when an operation of the THandleObject (see page 22066) fails. LastError returns the Windows error code

that represents the reason for the failure. If no operation failed, LastError is 0.

See Also

Handle (see page 22070)

157.1.4 TMutex Class

Class Hierarchy

System::TObject → SyncObjs.TSynchroObject → SyncObjs.THandleObject → SyncObjs.TMutex

Pascal

```
TMutex = class(THandleObject);
```

C++

```
class TMutex : public THandleObject;
```

File

SyncObjs

Description

This is class SyncObjs.TMutex.

157.1.4.1 TMutex Members

The following tables list the members exposed by TMutex.

Protected Fields

	Name	Description
	FHandle (see page 22069)	This is FHandle, a member of class THandleObject.
	FLastError (see page 22069)	This is FLastError, a member of class THandleObject.
	FUseCOMWait (see page 22069)	This is FUseCOMWait, a member of class THandleObject.

Public Methods

	Name	Description
V	Acquire (see page 22073)	Provides the interface for a method that devotes the synchronization object to a thread.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22072)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22068)	Destroys an instance of THandleObject (see page 22066).
V	Destroy (see page 22287)	Disposes of an object instance.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Release (see page 22074)	Provides the interface for a method that releases the synchronization object for use by other threads.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	WaitFor (see page 22070)	This is WaitFor, a member of class THandleObject.

Public Properties

	Name	Description
	Handle (see page 22070)	Specifies the handle of the Windows object used by THandleObject (see page 22066).
	LastError (see page 22070)	Indicates the Windows error code obtained from Handle (see page 22070).

157.1.4.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TMutex.Create (Boolean) (see page 22072)	This is Create, a member of class TMutex.
	TMutex.Create (LongWord, Boolean, string, Boolean) (see page 22073)	This is Create, a member of class TMutex.
	TMutex.Create (PSecurityAttributes, Boolean, string, Boolean) (see page 22073)	This is Create, a member of class TMutex.

157.1.4.2.1 TMutex.Create Constructor (Boolean)

Pascal

```
constructor Create(UseCOMWait: Boolean = False); overload;
```

C++

```
__fastcall TMutex(Boolean UseCOMWait = False);
```

Description

This is Create, a member of class TMutex.

157.1.4.2.2 TMutex.Create Constructor (LongWord, Boolean, string, Boolean)

Pascal

```
constructor Create(DesiredAccess: LongWord; InheritHandle: Boolean; const Name: string;
UseCOMWait: Boolean = False); overload;
```

C++

```
__fastcall TMutex(LongWord DesiredAccess, Boolean InheritHandle, const AnsiString Name,
Boolean UseCOMWait = False);
```

Description

This is Create, a member of class TMutex.

157.1.4.2.3 TMutex.Create Constructor (PSecurityAttributes, Boolean, string, Boolean)

Pascal

```
constructor Create(MutexAttributes: PSecurityAttributes; InitialOwner: Boolean; const Name:
string; UseCOMWait: Boolean = False); overload;
```

C++

```
__fastcall TMutex(PSecurityAttributes MutexAttributes, Boolean InitialOwner, const AnsiString
Name, Boolean UseCOMWait = False);
```

Description

This is Create, a member of class TMutex.

157.1.4.3 TMutex Methods

The methods of the TMutex class are listed here.

Public Methods

	Name	Description
	Acquire (see page 22073)	Provides the interface for a method that devotes the synchronization object to a thread.
	Release (see page 22074)	Provides the interface for a method that releases the synchronization object for use by other threads.

157.1.4.3.1 TMutex.Acquire Method

Provides the interface for a method that devotes the synchronization object to a thread.

Pascal

```
procedure Acquire; override;
```

C++

```
virtual __fastcall Acquire();
```

Description

Call Acquire from the thread that needs to use the synchronization object. The Acquire method for TSynchroObject does nothing other than provide the interface for a method that can start the synchronization object working on behalf of the calling thread.

See Also

Release (see page 22076)

157.1.4.3.2 TMutex.Release Method

Provides the interface for a method that releases the synchronization object for use by other threads.

Pascal

```
procedure Release; override;
```

C++

```
virtual __fastcall Release();
```

Description

Call Release from a thread that is finished using the synchronization object. The Release method for TSynchroObject does nothing other than provide the interface for a method that can unbind the synchronization object from the calling thread.

See Also

Acquire (see page 22076)

157.1.5 TSimpleEvent Class

TSimpleEvent represents an unnamed manual event object.

Class Hierarchy

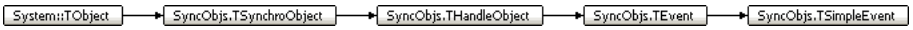

Pascal

```
TSimpleEvent = class(TEvent);
```

C++

```
class TSimpleEvent : public TEvent;
```

File

SyncObjs

Description

Use TSimpleEvent to coordinate between threads that all have access to the same simple event object. TSimpleEvent can also be used in a single-threaded application to coordinate sections of code that respond to different asynchronous events such as system messages or user actions.

Because TSimpleEvent represents an unnamed event object, a single TSimpleEvent object must be within the scope of all code that uses it. To coordinate with code that is out of scope, or with other applications, use a named TEvent (see page 22063) object.

See Also

TThread (see page 3108)

TEvent (see page 22063)

157.1.6 TSynchroObject Class

TSynchroObject is the base class for all thread synchronization objects.

Class Hierarchy

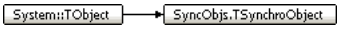

Pascal

```
TSynchroObject = class(TObject);
```

C++

```
class TSynchroObject : public TObject;
```

File

SyncObjs

Description

Use TSynchroObject as a base class when defining an object that coordinates the execution of different threads in a multi-threaded application. Synchronization objects include critical sections that temporarily block the execution of other threads which may interfere with the successful completion of a section of code, and event objects that allow threads to signal when critical events have occurred.

See Also

TCriticalSection (see page 22058)

TEvent (see page 22063)

TThread (see page 3108)

157.1.6.1 TSynchroObject Members

The following tables list the members exposed by TSynchroObject.

Public Methods

	Name	Description
	Acquire (see page 22076)	Provides the interface for a method that devotes the synchronization object to a thread.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Release (see page 22076)	Provides the interface for a method that releases the synchronization object for use by other threads.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

157.1.6.2 TSynchroObject Methods

The methods of the TSynchroObject class are listed here.

Public Methods

	Name	Description
V	Acquire (see page 22076)	Provides the interface for a method that devotes the synchronization object to a thread.
V	Release (see page 22076)	Provides the interface for a method that releases the synchronization object for use by other threads.

157.1.6.2.1 TSynchroObject.Acquire Method

Provides the interface for a method that devotes the synchronization object to a thread.

Pascal

```
procedure Acquire; virtual;
```

C++

```
virtual __fastcall Acquire();
```

Description

Call Acquire from the thread that needs to use the synchronization object. The Acquire method for TSynchroObject (see page 22075) does nothing other than provide the interface for a method that can start the synchronization object working on behalf of the calling thread.

See Also

Release (see page 22076)

157.1.6.2.2 TSynchroObject.Release Method

Provides the interface for a method that releases the synchronization object for use by other threads.

Pascal

```
procedure Release; virtual;
```

C++

```
virtual __fastcall Release();
```

Description

Call Release from a thread that is finished using the synchronization object. The Release method for TSynchroObject (see page 22075) does nothing other than provide the interface for a method that can unbind the synchronization object from the calling thread.

See Also

Acquire (see page 22076)

157.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TWaitResult (see page 22078)	TWaitResult indicates the reason an event object stopped waiting for its signal.

157.2.1 SyncObjs.TWaitResult Enumeration

TWaitResult indicates the reason an event object stopped waiting for its signal.

Pascal

```
TWaitResult = (
  wrSignaled,
  wrTimeout,
  wrAbandoned,
  wrError
);
```

C++

```
enum TWaitResult {
  wrSignaled,
  wrTimeout,
  wrAbandoned,
  wrError
};
```

File

SyncObjs

Description

TWaitResult includes the following values:

Value	Meaning
wrSignaled	The signal of the event object was set.
wrTimeOut	The time specified by the TimeOut parameter elapsed without the signal being set.
wrAbandoned	The event object was destroyed before the TimeOut period elapsed.
wrError	An error occurred while waiting. Check the LastError property for an error code giving more information.

158 System Namespace

This is namespace System.

Topics

Name	Description
Classes (see page 22092)	The following table lists classes in this documentation.
Interfaces (see page 22394)	The following table lists interfaces in this documentation.
Functions (see page 22400)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 22552)	The following table lists structs, records, enums in this documentation.
Types (see page 22579)	The following table lists types in this documentation.
Variables (see page 22629)	The following table lists variables in this documentation.
Constants (see page 22660)	The following table lists constants in this documentation.

Classes

	Name	Description
	AnsiString (see page 22093)	System::AnsiString is the C++ analog for the Delphi long string type.
	AutoCmd (see page 22127)	System::AutoCmd represents a command that can be executed by an Automation interface wrapped by a Variant (see page 22307) class.
	Currency (see page 22145)	System::Currency is the C++ implementation of the Delphi System::Currency data type.
	CurrencyBase (see page 22166)	This is class System::CurrencyBase.
	DelphiInterface (see page 22167)	System::DelphiInterface is a C++ implementation of the Delphi interface.
	DynamicArray (see page 22172)	System::DynamicArray is the C++ analog for the Delphi dynamic array type.
	DynArrayException (see page 22182)	This is class System::DynArrayException.
	DynArrayNullData (see page 22183)	This is class System::DynArrayNullData.
	DynArrayOutOfRange (see page 22183)	This is class System::DynArrayOutOfRange.
	Function (see page 22184)	System::Function represents a method on an Automation interface that returns a value.
	NamedParm (see page 22187)	NamedParm represents a named parameter assigned to a TAutoCmd descendant.
	OleVariant (see page 22193)	System::OleVariant is a C++ implementation of the Delphi intrinsic type System::OleVariant.
	OpenArray (see page 22216)	This is class System::OpenArray.
	OpenArrayCount (see page 22223)	This is class System::OpenArrayCount.
	Procedure (see page 22230)	System::Procedure represents a method on an Automation interface that does not return a value.
	PropertyGet (see page 22233)	PropertyGet represents a property access method on an Automation interface.
	PropertySet (see page 22236)	PropertySet represents a property access method on an Automation interface.

	SmallString (see page 22239)	This is class System::SmallString.
	SmallStringBase (see page 22242)	This is class System::SmallStringBase.
	sPointer (see page 22243)	This is class System::sPointer.
	StaticArray (see page 22245)	This is class System::StaticArray.
	TAggregatedObject (see page 22248)	System::TAggregatedObject provides the functionality for an inner object of an aggregate by implementing the IInterface (see page 22396) methods to delegate to the controlling IInterface (see page 22396).
	TClassHelperBase (see page 22251)	This is class System.TClassHelperBase.
	TContainedObject (see page 22253)	System::TContainedObject implements the IInterface (see page 22396) interface as an inner object in an aggregate.
	TDateTime (see page 22256)	System::TDateTime is the C++ analog for the Delphi System::TDateTime data type.
	TDateTimeBase (see page 22275)	This is class System::TDateTimeBase.
	TInterfacedObject (see page 22276)	System::TInterfacedObject is a base for simple classes that need a basic IInterface (see page 22396) implementation.
	TMetaClass (see page 22281)	System::TMetaClass is the C++ representation of the Delphi class-reference type.
	TObject (see page 22285)	System::TObject is the ultimate ancestor of all objects and components.
	Variant (see page 22307)	System::Variant is a C++ implementation of the Delphi intrinsic type System::Variant.
	VariantError (see page 22375)	This is class System::VariantError.
	WideString (see page 22377)	System::WideString represents strings of 16-bit characters.

Constants

Name	Description
CompilerVersion (see page 22662)	Designates the version number of the Delphi compiler.
CPUi386 (see page 22662)	This is constant System.CPUi386.
CPUi486 (see page 22663)	This is constant System.CPUi486.
CPUPentium (see page 22663)	This is constant System.CPUPentium.
E_NOINTERFACE (see page 22663)	This is constant System.E_NOINTERFACE.
E_NOTIMPL (see page 22664)	This is constant System.E_NOTIMPL.
E_UNEXPECTED (see page 22664)	This is constant System.E_UNEXPECTED.
fmClosed (see page 22664)	This is constant System.fmClosed.
fmInOut (see page 22665)	This is constant System.fmInOut.
fmInput (see page 22665)	This is constant System.fmInput.
fmOutput (see page 22665)	This is constant System.fmOutput.
MaxInt (see page 22666)	The maximum value of the Integer data type.
MaxLongint (see page 22666)	The maximum value of the Longint data type.
NumSmallBlockTypes (see page 22666)	NumSmallBlockTypes is the number of small memory block types employed by the default memory manager.
opAdd (see page 22667)	opAdd defines an addition operation.
opAnd (see page 22667)	opAnd defines a logical and operation.
opCmpEQ (see page 22668)	opCmpEQ defines an equality comparison operator.
opCmpGE (see page 22668)	opCmpGE defines a greater than or equal comparison operator.
opCmpGT (see page 22669)	opCmpGT defines a greater than comparison operator.
opCmpLE (see page 22669)	opCmpLE defines a less than or equal comparison operator.
opCmpLT (see page 22670)	opCmpLT defines a less than comparison operator.
opCmpNE (see page 22670)	opCmpNE defines an inequality comparison operator.
opCompare (see page 22671)	opCompare defines a comparison operation.
opDivide (see page 22671)	opDivide defines a divide operation.
opIntDivide (see page 22672)	opIntDivide defines an integer divide operation.
opModulus (see page 22672)	opModulus defines a modulus operation.
opMultiply (see page 22673)	opMultiply defines a multiply operation.
opNegate (see page 22673)	opNegate defines a negate operation.
opNot (see page 22674)	opNot defines a binary negation operation.
opOr (see page 22674)	opOr defines a logical or operation.
opShiftLeft (see page 22675)	opShiftLeft defines a bitwise shift left operation.
opShiftRight (see page 22675)	opShiftRight defines a bitwise shift right operation.

opSubtract (see page 22676)	opSubtract defines a subtraction operation.
opXor (see page 22676)	opXor defines a logical exclusive or operation.
RTLVersion (see page 22677)	Designates the version number of the Delphi runtime library.
S_FALSE (see page 22677)	This is constant System.S_FALSE.
S_OK (see page 22677)	This is constant System.S_OK.
sLineBreak (see page 22678)	Represents the character(s) used to generate a line feed.
tfCRLF (see page 22678)	This is constant System.tfCRLF.
varAny (see page 22678)	Represents a Variant (see page 22307) that can hold any value.
varArray (see page 22679)	Represents a Variant (see page 22307) that holds dynamic array data.
varBoolean (see page 22680)	Represents a Variant (see page 22307) that holds WordBool data.
varByRef (see page 22680)	Represents a Variant (see page 22307) that points to another variable.
varByte (see page 22681)	Represents a Variant (see page 22307) that holds byte data.
varCurrency (see page 22681)	Represents a Variant (see page 22307) that holds a monetary floating point value.
varDate (see page 22682)	Represents a Variant (see page 22307) that holds a TDateTime (see page 22256) value.
varDispatch (see page 22682)	Represents a Variant (see page 22307) that has been assigned an IDispatch (see page 22394) value.
varDouble (see page 22683)	Represents a Variant (see page 22307) that holds a floating point value.
varEmpty (see page 22683)	Represents a Variant (see page 22307) that has not yet been assigned a value.
varError (see page 22684)	Represents a Variant (see page 22307) that holds a 32 bit error code.
varInt64 (see page 22684)	Represents a Variant (see page 22307) that holds Int64 data.
varInteger (see page 22685)	Represents a Variant (see page 22307) that holds Integer data.
varLongWord (see page 22685)	Represents a Variant (see page 22307) that holds LongWord data.
varNull (see page 22686)	Represents a Variant (see page 22307) that holds a null value.
varOleStr (see page 22686)	Represents a Variant (see page 22307) that has been assigned a COM interface type string value.
varShortInt (see page 22687)	Represents a Variant (see page 22307) that holds ShortInt data.
varSingle (see page 22687)	Represents a Variant (see page 22307) that holds byte data.
varSmallint (see page 22688)	This is constant System.varSmallint.
varStrArg (see page 22688)	Represents a Variant (see page 22307) that has holds a COM compatible string value.
varString (see page 22688)	Represents a Variant (see page 22307) that holds string data.
varTypeMask (see page 22689)	Defines a mask for filtering out the meta data part of a Variant (see page 22307) data type.
varUnknown (see page 22690)	Represents a Variant (see page 22307) that holds an unknown value.
varVariant (see page 22690)	Represents a Variant (see page 22307) data type.
varWord (see page 22691)	Represents a Variant (see page 22307) that holds Word data.
vmtAddRef (see page 22691)	This is constant System.vmtAddRef.
vmtAfterConstruction (see page 22691)	This is constant System.vmtAfterConstruction.
vmtAutoTable (see page 22692)	This is constant System.vmtAutoTable.
vmtBeforeDestruction (see page 22692)	This is constant System.vmtBeforeDestruction.
vmtClassName (see page 22692)	This is constant System.vmtClassName.
vmtCreateObject (see page 22693)	This is constant System.vmtCreateObject.
vmtDefaultHandler (see page 22693)	This is constant System.vmtDefaultHandler.
vmtDestroy (see page 22693)	This is constant System.vmtDestroy.
vmtDispatch (see page 22694)	This is constant System.vmtDispatch.
vmtDynamicTable (see page 22694)	This is constant System.vmtDynamicTable.
vmtFieldTable (see page 22694)	This is constant System.vmtFieldTable.
vmtFreeInstance (see page 22695)	This is constant System.vmtFreeInstance.
vmtInitTable (see page 22695)	This is constant System.vmtInitTable.
vmtInstanceSize (see page 22695)	This is constant System.vmtInstanceSize.
vmtIntfTable (see page 22696)	This is constant System.vmtIntfTable.
vmtMethodTable (see page 22696)	This is constant System.vmtMethodTable.
vmtNewInstance (see page 22696)	This is constant System.vmtNewInstance.

vmtParent (see page 22697)	This is constant System.vmtParent.
vmtQueryInterface (see page 22697)	This is constant System.vmtQueryInterface.
vmtRelease (see page 22697)	This is constant System.vmtRelease.
vmtSafeCallException (see page 22698)	This is constant System.vmtSafeCallException.
vmtSelfPtr (see page 22698)	This is constant System.vmtSelfPtr.
vmtTypeInfo (see page 22698)	This is constant System.vmtTypeInfo.
vtAnsiString (see page 22699)	vtAnsiString defines an AnsiString (see page 22093) TVarData (see page 22571) VType value.
vtBoolean (see page 22699)	vtBoolean defines a Boolean TVarData (see page 22571) VType value.
vtChar (see page 22700)	vtChar defines a Char TVarData (see page 22571) VType value.
vtClass (see page 22700)	vtClass defines a TClass (see page 22611) TVarData (see page 22571) VType value.
vtCurrency (see page 22701)	vtCurrency defines a Currency (see page 22145) TVarData (see page 22571) VType value.
vtExtended (see page 22701)	vtExtended defines an Extended TVarData (see page 22571) VType value.
vtInt64 (see page 22702)	vtInt64 defines an Int64 TVarData (see page 22571) VType value.
vtInteger (see page 22702)	vtInteger defines an Integer TVarData (see page 22571) VType value.
vtInterface (see page 22703)	vtInterface defines an Interface pointer TVarData (see page 22571) VType value.
vtObject (see page 22703)	vtObject defines a TObject (see page 22285) TVarData (see page 22571) VType value.
vtPChar (see page 22704)	vtPChar defines a PChar TVarData (see page 22571) VType value.
vtPointer (see page 22704)	vtPointer defines a Pointer TVarData (see page 22571) VType value.
vtPWideChar (see page 22705)	vtPWideChar defines a PWideChar TVarData (see page 22571) VType value.
vtString (see page 22705)	vtString defines a Variant (see page 22307) TVarData (see page 22571) VType value.
vtVariant (see page 22706)	This is constant System.vtVariant.
vtWideChar (see page 22706)	vtWideChar defines a WideChar TVarData (see page 22571) VType value.
vtWideString (see page 22706)	vtWideString defines a WideString (see page 22377) TVarData (see page 22571) VType value.

Functions

	Name	Description
	!= (see page 22405)	This is function System::!=.
	__currMul (see page 22405)	This is function System::__currMul.
	_CVR_PROBE (see page 22405)	This is function System._CVR_PROBE.
	_CVR_STMTPROBE (see page 22406)	This is function System._CVR_STMTPROBE.
	_roundToInt64 (see page 22406)	This is function System::_roundToInt64.
	+ (see page 22406)	This is function System::+.
	< (see page 22407)	This is function System::<.
	<< (see page 22407)	This is function System::<<.
	<< (see page 22407)	This is function System::<<.
	<< (see page 22408)	This is function System::<<.
	<= (see page 22408)	This is function System::<=.
	== (see page 22408)	This is function System::==.
	> (see page 22409)	This is function System::>.
	>= (see page 22409)	This is function System::>=.
	>> (see page 22409)	This is function System::>>.
	>> (see page 22410)	This is function System::>>.
	Abs (see page 22410)	Returns an absolute value.
	AcquireExceptionObject (see page 22410)	Retains access to exception object.
	AddModuleUnloadProc (see page 22411)	AddModuleUnloadProc adds a call to your defined routine when a package is unloaded.
	AddModuleUnloadProc (see page 22411)	AddModuleUnloadProc adds a call to your defined routine when a package is unloaded.
	Addr (see page 22412)	Returns a pointer to a specified object.
	AllocMem (see page 22413)	AllocMem allocates a memory block and initializes each byte to zero.

	AnsiToUtf8 (see page 22415)	Converts a string encoded in Ansi to UTF-8.
	Append (see page 22415)	Prepares an existing file for adding text to the end.
	ArcTan (see page 22416)	Calculates the arctangent of a given number.
	Assert (see page 22417)	Tests whether a Boolean expression is true.
	Assert (see page 22417)	Tests whether a Boolean expression is true.
	Assigned (see page 22419)	Tests for a nil (unassigned) pointer or procedural variable.
	AssignFile (see page 22420)	Associates the name of an external file with a file variable.
	AttemptToUseSharedMemoryManager (see page 22422)	Attempts to use a shared memory manager.
	BeginThread (see page 22422)	Spawns a separate thread of execution.
	BlockRead (see page 22423)	Reads one or more records from an open file into a variable.
	BlockRead (see page 22423)	Reads one or more records from an open file into a variable.
	BlockWrite (see page 22425)	Writes one or more records from a variable to an open file.
	BlockWrite (see page 22425)	Writes one or more records from a variable to an open file.
	Break (see page 22426)	Causes the flow of control to exit a for, while, or repeat statement.
	ChDir (see page 22429)	Changes the current directory.
	ChDir (see page 22429)	Changes the current directory.
	Chr (see page 22430)	Returns the character for a specified ASCII value.
	Close (see page 22432)	Terminates the association between a file variable and an external file.
	CloseFile (see page 22433)	Terminates the association between file variable and an external disk file.
	CompToCurrency (see page 22435)	Converts a Comp value to a Currency (see page 22145) value.
	CompToDouble (see page 22435)	Converts a Comp value to a double value.
	Concat (see page 22436)	Concatenates two or more strings into one.
	Concat (see page 22436)	Concatenates two or more strings into one.
	Concat (see page 22436)	Concatenates two or more strings into one.
	Continue (see page 22437)	Allows the flow of control to proceed to the next iteration of for, while, or repeat statements.
	Copy (see page 22438)	Returns a substring of a string or a segment of a dynamic array.
	Cos (see page 22440)	Calculates the cosine of an angle.
	CurrencyToComp (see page 22441)	Converts a Currency (see page 22145) value to a Comp value.
	Dec (see page 22441)	Decrements a variable by 1 or N.
	Dec (see page 22441)	Decrements a variable by 1 or N.
	Delete (see page 22443)	Removes a substring from a string.
	Dispose (see page 22444)	Releases memory allocated for a dynamic variable.
	DoubleToComp (see page 22446)	Converts a double value to a Comp.
	DynArrayClear (see page 22446)	This is function System.DynArrayClear.
	DynArrayDim (see page 22446)	This is function System.DynArrayDim.
	DynArraySetLength (see page 22447)	This is function System.DynArraySetLength.
	EndThread (see page 22447)	Terminates the execution of a thread.
	EnumModules (see page 22447)	Executes a callback for all the modules in an application.
	EnumModules (see page 22447)	Executes a callback for all the modules in an application.
	EnumResourceModules (see page 22448)	Executes a callback for all resource modules in an application.
	EnumResourceModules (see page 22448)	Executes a callback for all resource modules in an application.
	Eof (see page 22449)	Tests whether the file position is at the end of a file.
	Eof (see page 22449)	Tests whether the file position is at the end of a file.
	Eof (see page 22449)	Tests whether the file position is at the end of a file.
	Eoln (see page 22450)	Tests whether the file pointer is at the end of a line.
	Eoln (see page 22450)	Tests whether the file pointer is at the end of a line.
	Erase (see page 22451)	Deletes an external file.
	Error (see page 22451)	Error is used to generate a run-time exception.
	ExceptAddr (see page 22452)	Returns the address at which the current exception was raised.
	ExceptObject (see page 22452)	Returns current exception object.
	Exclude (see page 22453)	Removes an element from a Delphi set or removes an event handler in .NET applications.
	Exit (see page 22453)	Exits from the current procedure.
	Exp (see page 22455)	Returns the exponential of X.

	Exp (see page 22455)	Returns the exponential of X.
	FilePos (see page 22456)	Returns the current file position.
	FileSize (see page 22457)	Returns the number of records in a file.
	FillChar (see page 22459)	Fills contiguous bytes with a specified value.
	Finalize (see page 22460)	Uninitializes a dynamically allocated variable.
	Finalize (see page 22460)	Uninitializes a dynamically allocated variable.
	FindClassHInstance (see page 22461)	Returns the instance handle for the module in which a class type is defined.
	FindHInstance (see page 22461)	Returns the instance handle of the module that contains a specified address.
	FindResourceHInstance (see page 22462)	Returns the instance handle of the resource module associated with a specified HInstance.
	Flush (see page 22462)	Empties the buffer of a text file opened for output.
	FPower10 (see page 22463)	This is function System.FPower10.
	Frac (see page 22463)	Returns the fractional part of a real number.
	FreeMem (see page 22464)	FreeMem frees a memory block.
	FreeMem (see page 22464)	FreeMem frees a memory block.
	FreeMemory (see page 22466)	FreeMemory frees a memory block.
	Get8087CW (see page 22466)	Returns the value of the 8087 code word.
	GetDir (see page 22467)	Returns the current directory.
	GetHeapStatus (see page 22468)	Returns the current status of the memory manager.
	GetLastError (see page 22469)	Returns the last error reported by an operating system API call.
	GetMem (see page 22469)	GetMem allocates a memory block.
	GetMemory (see page 22470)	GetMemory allocates a memory block.
	GetMemoryManager (see page 22472)	Returns the entry points of the currently installed memory manager.
	GetMemoryManager (see page 22472)	Returns the entry points of the currently installed memory manager.
	GetMemoryManagerState (see page 22472)	Fetches state information for the Memory Manager.
	GetMemoryMap (see page 22473)	Fetches the memory map.
	GetMinimumBlockAlignment (see page 22473)	Gets the minimum memory block alignment.
	GetModuleFileName (see page 22474)	Returns the fully qualified name for a module, given its handle.
	GetVariantManager (see page 22474)	Returns the entry points of the routines that define variant behavior.
	Halt (see page 22475)	Initiates abnormal termination of a program.
	Halt (see page 22475)	Initiates abnormal termination of a program.
	Hi (see page 22476)	Returns the high-order byte of X as an unsigned value.
	High (see page 22476)	Returns the highest value in the range of an argument.
	Inc (see page 22478)	Increments an ordinal value by one or N.
	Inc (see page 22478)	Increments an ordinal value by one or N.
	Include (see page 22479)	Adds an element to a Delphi set or adds an event handler in .NET applications.
	Initialize (see page 22480)	Initializes a dynamically allocated variable.
	Initialize (see page 22480)	Initializes a dynamically allocated variable.
	Insert (see page 22480)	Inserts a substring into a string beginning at a specified point.
	Int (see page 22481)	Returns the integer part of a real number.
	IOResult (see page 22482)	Returns the status of the last I/O operation performed.
	IsMemoryManagerSet (see page 22483)	Indicates whether the memory manager has been overridden using the SetMemoryManager (see page 22523) procedure.
	IsVariantManagerSet (see page 22483)	Indicates whether variant support is implemented in an application.
	Length (see page 22484)	Returns the number of characters in a string or elements in an array.
	Ln (see page 22485)	Returns the natural log of a real expression.
	Lo (see page 22486)	Returns the low order Byte of argument X.
	LoadResourceModule (see page 22486)	This is function System.LoadResourceModule.
	LoadResString (see page 22487)	This is function System.LoadResString.
	Low (see page 22487)	Returns the lowest value in a range.
	MkDir (see page 22488)	Creates a new subdirectory.
	MkDir (see page 22488)	Creates a new subdirectory.
	Move (see page 22490)	Copies bytes from a source to a destination.
	New (see page 22491)	Creates a new dynamic variable and sets P to point to it.
	Odd (see page 22493)	Returns true if argument is an odd number.

	OleStrToString (see page 22493)	Copies data received from a COM interface to a string.
	OleStrToStrVar (see page 22494)	Copies a string from its COM representation to an existing Delphi string.
	Ord (see page 22494)	Returns the ordinal value of an ordinal-type expression.
	ParamCount (see page 22495)	Returns the number of parameters passed on the command line.
	ParamStr (see page 22496)	Returns a specified parameter from the command-line.
	Pi (see page 22497)	Returns 3.1415926535897932385.
	Pos (see page 22497)	Returns the index value of the first character in a specified substring that occurs in a given string.
	Pos (see page 22497)	Returns the index value of the first character in a specified substring that occurs in a given string.
	Pred (see page 22499)	Returns the predecessor of the argument.
	Ptr (see page 22501)	Converts a specified address to a pointer.
	PUCS4Chars (see page 22501)	Converts a UCS4 string to a null-terminated array of UCS4 characters.
	RaiseList (see page 22502)	This is function System.RaiseList.
	Random (see page 22502)	Generates random numbers within a specified range.
	Random (see page 22502)	Generates random numbers within a specified range.
	Randomize (see page 22503)	Initializes the random number generator with a random value.
	Read (see page 22504)	Read reads data from a file.
	Read (see page 22504)	Read reads data from a file.
	Read (see page 22504)	Read reads data from a file.
	Read (see page 22504)	Read reads data from a file.
	ReadLn (see page 22506)	Reads a line of text from a file.
	ReadLn (see page 22506)	Reads a line of text from a file.
	ReadLn (see page 22506)	Reads a line of text from a file.
	ReadLn (see page 22506)	Reads a line of text from a file.
	ReallocMem (see page 22507)	ReallocMem reallocates a memory block.
	ReallocMemory (see page 22508)	ReallocMemory resizes a memory block.
	RegisterExpectedMemoryLeak (see page 22508)	Registers an expected memory leak.
	RegisterModule (see page 22509)	This is function System.RegisterModule.
	ReleaseExceptionObject (see page 22509)	Releases exception object acquired by AcquireExceptionObject (see page 22410).
	RemoveModuleUnloadProc (see page 22510)	RemoveModuleUnloadProc removes a call to your defined routine when a package is unloaded.
	RemoveModuleUnloadProc (see page 22510)	RemoveModuleUnloadProc removes a call to your defined routine when a package is unloaded.
	Rename (see page 22510)	Changes the name of an external file.
	Rename (see page 22510)	Changes the name of an external file.
	Reset (see page 22511)	Opens an existing file.
	Reset (see page 22511)	Opens an existing file.
	Reset (see page 22511)	Opens an existing file.
	Reset (see page 22511)	Opens an existing file.
	Rewrite (see page 22513)	Creates a new file and opens it.
	Rewrite (see page 22513)	Creates a new file and opens it.
	RmDir (see page 22515)	Deletes an empty subdirectory.
	RmDir (see page 22515)	Deletes an empty subdirectory.
	Round (see page 22516)	Returns the value of X rounded to the nearest whole number.
	RunError (see page 22517)	Stops execution and generates a runtime error.
	RunError (see page 22517)	Stops execution and generates a runtime error.
	Seek (see page 22518)	Moves the current position of a file to a specified component.
	SeekEof (see page 22519)	Returns the end-of-file status of a file, ignoring whitespace.
	SeekEof (see page 22519)	Returns the end-of-file status of a file, ignoring whitespace.
	SeekEoln (see page 22520)	Returns the end-of-line status of a file, ignoring whitespace.
	SeekEoln (see page 22520)	Returns the end-of-line status of a file, ignoring whitespace.
	Set8087CW (see page 22520)	Sets both the control word in the floating point unit and the variable NoErrMsg (see page 22652) declared in the system unit.
	SetInOutRes (see page 22521)	This is function System.SetInOutRes.

	SetLastError (see page 22521)	SetLastError sets the last API error code.
	SetLength (see page 22522)	Sets the length of a string or dynamic-array variable.
	SetLineBreakStyle (see page 22522)	Determines the end-of-line and end-of-file conventions for text file I/O.
	SetMemoryManager (see page 22523)	Sets entry points of the memory manager.
	SetMemoryManager (see page 22523)	Sets entry points of the memory manager.
	SetMinimumBlockAlignment (see page 22524)	Sets the minimum memory block alignment.
	SetMultiByteConversionCodePage (see page 22524)	SetMultiByteConversionCodePage overrides the default multi byte code conversion page.
	SetRaiseList (see page 22525)	This is function System.SetRaiseList.
	SetString (see page 22525)	Sets the contents and length of the given string.
	SetTextBuf (see page 22526)	Assigns an I/O buffer to a text file.
	SetTextBuf (see page 22526)	Assigns an I/O buffer to a text file.
	SetVariantManager (see page 22527)	Sets the entry points of routines that define variant behavior.
	ShareMemoryManager (see page 22527)	Shares the module's memory manager.
	Sin (see page 22528)	Returns the sine of the angle in radians.
	SizeOf (see page 22528)	Returns the number of bytes occupied by a variable or type.
	Slice (see page 22529)	Returns a sub-section of an array.
	Sqr (see page 22529)	Returns the square of a number.
	Sqr (see page 22529)	Returns the square of a number.
	Sqrt (see page 22530)	Returns the square root of X.
	Str (see page 22530)	Formats a string and returns it to a variable.
	Str (see page 22530)	Formats a string and returns it to a variable.
	StringOfChar (see page 22531)	Returns a string with a specified number of repeating characters.
	StringOfChar (see page 22531)	Returns a string with a specified number of repeating characters.
	StringToOleStr (see page 22531)	Allocates memory and copies an AnsiStringa string to the OLE format.
	StringToWideChar (see page 22532)	Returns a UNICODE string from an AnsiString (see page 22093).
	Succ (see page 22532)	Returns the successor of an argument.
	Swap (see page 22533)	Exchanges high order byte with the low order byte of an integer or word.
	SysAllocMem (see page 22533)	This is function System.SysAllocMem.
	SysFreeMem (see page 22534)	Frees the memory pointed to by a specified pointer.
	SysGetMem (see page 22534)	Allocates a specified number of bytes and returns a pointer to them.
	SysReallocMem (see page 22535)	Returns a pointer to a specified number of bytes, preserving the values pointed to by the Pointer parameter.
	SysRegisterExpectedMemoryLeak (see page 22536)	This is function System.SysRegisterExpectedMemoryLeak.
	SysUnregisterExpectedMemoryLeak (see page 22536)	This is function System.SysUnregisterExpectedMemoryLeak.
	TextStart (see page 22536)	This is function System.TextStart.
	Trunc (see page 22537)	Truncates a real number to an integer.
	Truncate (see page 22537)	Deletes all the records after the current file position.
	UCS4StringToWideString (see page 22538)	Converts a string of 4-byte Unicode characters into a WideString (see page 22377) value.
	UnicodeToUtf8 (see page 22538)	Converts a string of Unicode characters into a UTF-8 string.
	UnicodeToUtf8 (see page 22538)	Converts a string of Unicode characters into a UTF-8 string.
	UniqueString (see page 22539)	Ensures that a given string has a reference count of one.
	UniqueString (see page 22539)	Ensures that a given string has a reference count of one.
	UnregisterExpectedMemoryLeak (see page 22539)	Unregisters an expected memory leak.
	UnregisterModule (see page 22540)	This is function System.UnregisterModule.
	UpCase (see page 22540)	Converts a character to uppercase.
	UTF8Decode (see page 22541)	Converts a UTF8 string to a Unicode string (WideString (see page 22377)).
	UTF8Encode (see page 22541)	Converts a WideString (see page 22377) value to a UTF8 string.
	Utf8ToAnsi (see page 22542)	Converts a UTF8 string to a string of Ansi characters.
	Utf8ToUnicode (see page 22542)	Converts a string of UTF-8 characters to a Unicode string.
	Utf8ToUnicode (see page 22542)	Converts a string of UTF-8 characters to a Unicode string.
	Val (see page 22543)	Converts a string to a numeric representation.
	VarArrayRedim (see page 22544)	Resizes a Variant (see page 22307) array.
	VarArrayRedim (see page 22544)	This is function System::VarArrayRedim.

	VarCast (see page 22545)	This is function System::VarCast.
	VarClear (see page 22545)	Empties a Variant (see page 22307), so that it is unassigned.
	VarClear (see page 22546)	This is function System::VarClear.
	WideCharLenToString (see page 22546)	Converts Unicode characters to single or multi byte character data.
	WideCharLenToStrVar (see page 22546)	WideCharLenToStrVar converts Unicode characters to single or multi byte character data.
	WideCharToString (see page 22547)	Converts null-terminated Unicode string to single or multi byte character data.
	WideCharToStrVar (see page 22547)	Converts Unicode string to a single or multi byte character data.
	WideStringToUCS4String (see page 22548)	Converts a WideString (see page 22377) value to a string of 4-byte Unicode characters.
	Write (see page 22548)	Writes to either a typed file or a text file
	Write (see page 22548)	Writes to either a typed file or a text file
	Write (see page 22548)	Writes to either a typed file or a text file
	Write (see page 22548)	Writes to either a typed file or a text file
	WriteLn (see page 22550)	Writes to a text file and adds an end-of-line marker.
	WriteLn (see page 22550)	Writes to a text file and adds an end-of-line marker.
	WriteLn (see page 22550)	Writes to a text file and adds an end-of-line marker.
	WriteLn (see page 22550)	Writes to a text file and adds an end-of-line marker.

Interfaces

	Name	Description
	IDispatch (see page 22394)	This is class System.IDispatch.
	IInterface (see page 22396)	IInterface is the base class for all interfaces defined in Delphi code.
	IInvokable (see page 22398)	IInvokable is the base class for invokable interfaces.

Structs, Records, Enums

	Name	Description
	PackageInfoTable (see page 22553)	This is record System.PackageInfoTable.
	PackageUnitEntry (see page 22553)	This is record System.PackageUnitEntry.
	TCallDesc (see page 22553)	TCallDesc is the dispatch call descriptor for variants, and PCallDesc (see page 22588) is a pointer to this descriptior.
	TChunkStatus (see page 22554)	TChunkStatus enumerates the possible states of a chunk of memory.
	TCVModInfo (see page 22555)	This is record System.TCVModInfo.
	TDispatchMessage (see page 22555)	This is record System.TDispatchMessage.
	TDispDesc (see page 22556)	This is record System.TDispDesc.
	TDynArrayTypeInfo (see page 22556)	This is record System.TDynArrayTypeInfo.
	TFileRec (see page 22557)	TFileRec is the internal format for typed and untyped files in the Delphi language.
	TGUID (see page 22558)	TGUID is a structured form of the value that uniquely identifies an interface.
	THeapStatus (see page 22558)	THeapStatus represents information about the global heap.
	TInitContext (see page 22560)	This is record System.TInitContext.
	TInterfaceEntry (see page 22560)	This is record System::TInterfaceEntry.
	TInterfaceTable (see page 22561)	This is record System::TInterfaceTable.
	TLibModule (see page 22561)	TLibModule describes a module that is compiled with the application.
	TMemoryManager (see page 22562)	TMemoryManager defines the memory block entry points.
	TMemoryManagerEx (see page 22563)	TMemoryManagerEx defines extended memory block entry points.
	TMemoryManagerState (see page 22564)	TMemoryManagerState holds statistics about memory allocations.
	TMethod (see page 22565)	TMethod stores Code and Data fields to represent a method.
	TMinimumBlockAlignment (see page 22566)	TMinimumBlockAlignment defines the mimimum byte alignment for memory block allocation.
	TModuleUnloadRec (see page 22566)	This is record System.TModuleUnloadRec.
	TResStringRec (see page 22567)	This is record System.TResStringRec.
	TRuntimeError (see page 22567)	This is record System.TRuntimeError.
	TSmallBlockTypeState (see page 22568)	TSmallBlockTypeState holds statistics about a small memory block.
	TTextLineBreakStyle (see page 22569)	TTextLineBreakStyle indicates the default set of characters used to divide lines in text.
	TTextRec (see page 22569)	TTextRec is the internal format for text files.

	TVarArray (see page 22570)	This is record System.TVarArray.
	TVarArrayBound (see page 22571)	This is record System.TVarArrayBound.
	TVarData (see page 22571)	TVarData represents Variant (see page 22307) information.
	TVariantManager (see page 22575)	TVariantManager defines entry points for variant conversion routines and other procedures that define variant behavior.
	TVarRec (see page 22577)	This is class System::TVarRec.
	TCallDesc (see page 22578)	This is type System::TCallDesc.
	TDispDesc (see page 22578)	This is type System::TDispDesc.

Types

Name	Description
ArrayOfboolean (see page 22581)	This is type System::ArrayOfboolean.
ArrayOfbyte (see page 22582)	This is type System::ArrayOfbyte.
ArrayOfdouble (see page 22582)	This is type System::ArrayOfdouble.
ArrayOfint (see page 22582)	This is type System::ArrayOfint.
ArrayOflong (see page 22583)	This is type System::ArrayOflong.
ArrayOfshort (see page 22583)	This is type System::ArrayOfshort.
ArrayOfstring (see page 22583)	This is type System::ArrayOfstring.
ArrayOfwidestring (see page 22584)	This is type System::ArrayOfwidestring.
GetPackageInfoTable (see page 22584)	This is type System.GetPackageInfoTable.
HGLOBAL (see page 22584)	This is type System.HGLOBAL.
HINST (see page 22585)	This is type System.HINST.
HMODULE (see page 22585)	This is type System.HMODULE.
HRESULT (see page 22585)	This is type System.HRESULT.
HRSRC (see page 22586)	This is type System.HRSRC.
IntegerArray (see page 22586)	This is type System.IntegerArray.
IUnknown (see page 22586)	IUnknown is a special base interface.
PackageInfo (see page 22587)	This is type System.PackageInfo.
PAnsiString (see page 22587)	PAnsiString is a pointer to an AnsiString (see page 22093) variable.
PBoolean (see page 22587)	PBoolean is a pointer to a Boolean variable.
PByte (see page 22588)	PByte is a pointer to a Byte variable.
PCallDesc (see page 22588)	This is type System::PCallDesc.
PCardinal (see page 22589)	PCardinal is a pointer to a Cardinal variable.
PComp (see page 22589)	PComp is a pointer to a Comp variable.
PCurrency (see page 22589)	PCurrency is a pointer to a Currency (see page 22145) variable.
PCVModInfo (see page 22590)	This is type System.PCVModInfo.
PDate (see page 22590)	PDate is a pointer to a Date variable.
PDateTime (see page 22591)	PDateTime is a pointer to a TDateTime (see page 22256) variable.
PDispatch (see page 22591)	This is type System.PDispatch.
PDispDesc (see page 22591)	This is type System::PDispDesc.
PDouble (see page 22592)	PDouble is a pointer to a Double variable.
PDynArrayTypeInfo (see page 22592)	This is type System.PDynArrayTypeInfo.
PError (see page 22592)	This is type System.PError.
PExtended (see page 22593)	PExtended is a pointer to an Extended variable.
PGUID (see page 22593)	PGUID is a pointer to a TGUID (see page 22558) variable.
PInitContext (see page 22593)	This is type System.PInitContext.
PInt64 (see page 22594)	PInt64 is a pointer to an Int64 variable.
PInteger (see page 22594)	PInteger is a pointer to an Integer variable.
PIntegerArray (see page 22595)	PIntegerArray is a pointer to an IntegerArray (see page 22586) variable.
PInterfaceEntry (see page 22595)	This is type System::PInterfaceEntry.
PInterfaceTable (see page 22595)	This is type System::PInterfaceTable.
PLibModule (see page 22596)	PLibModule is a pointer to a TLibModule (see page 22561) variable.
PLongint (see page 22596)	This is type System.PLongint.
PLongWord (see page 22596)	PLongWord is a pointer to a LongWord variable.

PMemoryManager (see page 22597)	PMemoryManager is a pointer to a TMemoryManager (see page 22562) variable.
PMemoryManagerEx (see page 22597)	PMemoryManagerEx is a pointer to a TMemoryManagerEx (see page 22563) variable.
PModuleUnloadRec (see page 22598)	This is type System.PModuleUnloadRec.
PointerArray (see page 22598)	This is type System.PointerArray.
POleVariant (see page 22598)	This is type System.POleVariant.
PPAnsiChar (see page 22599)	This is type System.PPAnsiChar.
PPChar (see page 22599)	This is type System.PPChar.
PPCharArray (see page 22599)	This is type System.PPCharArray.
PPDispatch (see page 22600)	This is type System.PPDispatch.
PPointer (see page 22600)	PPointer is a pointer to a Pointer variable.
PPointerArray (see page 22600)	PPointerArray is a pointer to a PointerArray (see page 22598) variable.
PPUnknown (see page 22601)	This is type System.PPUnknown.
PPWideChar (see page 22601)	This is type System.PPWideChar.
PResStringRec (see page 22601)	This is type System.PResStringRec.
PShortInt (see page 22602)	PShortInt is a pointer to a ShortInt variable.
PShortString (see page 22602)	PShortString is a pointer to a ShortString variable.
PSingle (see page 22602)	PSingle is a pointer to a Single variable.
PSmallInt (see page 22603)	PSmallInt is a pointer to a SmallInt variable.
PString (see page 22603)	PString is a pointer to a String variable.
PTextBuf (see page 22604)	This is type System.PTextBuf.
PUCS2Char (see page 22604)	This is type System.PUCS2Char.
PUCS4Char (see page 22604)	This is type System.PUCS4Char.
PUCS4CharArray (see page 22605)	This is type System.PUCS4CharArray.
PUnitEntryTable (see page 22605)	This is type System.PUnitEntryTable.
PUnknown (see page 22605)	PUnknown is a pointer to an IUnknown (see page 22586) variable.
PUTF8String (see page 22606)	This is type System.PUTF8String.
PVarArray (see page 22606)	This is type System.PVarArray.
PVarArrayBoundArray (see page 22606)	This is type System.PVarArrayBoundArray.
PVarArrayCoorArray (see page 22607)	This is type System.PVarArrayCoorArray.
PVarData (see page 22607)	This is type System.PVarData.
PVariant (see page 22607)	PVariant is a pointer to a Variant (see page 22307) variable.
PVariantManager (see page 22608)	This is type System.PVariantManager.
PVarRec (see page 22608)	This is type System::PVarRec.
PWideString (see page 22608)	PWideString is a pointer to a WideString (see page 22377) variable.
PWord (see page 22609)	PWord is a pointer to a Word variable.
PWordBool (see page 22609)	PWordBool is a pointer to a WordBool variable.
SCODE (see page 22609)	This is type System::SCODE.
TAssertErrorProc (see page 22610)	This is type System.TAssertErrorProc.
TBooleanDynArray (see page 22610)	This is type System::TBooleanDynArray.
TBoundArray (see page 22610)	This is type System.TBoundArray.
TByteDynArray (see page 22611)	This is type System::TByteDynArray.
TCardinalDynArray (see page 22611)	This is type System::TCardinalDynArray.
TClass (see page 22611)	TClass defines the metaclass for TObject (see page 22285).
TDateTime (see page 22612)	TDateTime represents a date-and-time value in the Delphi language.
TDLLProc (see page 22613)	This is type System.TDLLProc.
TDLLProcEx (see page 22613)	This is type System.TDLLProcEx.
TDoubleDynArray (see page 22614)	This is type System::TDoubleDynArray.
TEnumModuleFunc (see page 22614)	This is type System.TEnumModuleFunc.
TEnumModuleFuncLW (see page 22614)	This is type System.TEnumModuleFuncLW.
TFileIOFunc (see page 22615)	This is type System.TFileIOFunc.
THandle (see page 22615)	THandle is a general purpose data reference (handle).
TInt64DynArray (see page 22615)	This is type System::TInt64DynArray.
TIntegerDynArray (see page 22616)	This is type System::TIntegerDynArray.

TInterfacedClass (see page 22616)	TInterfacedClass defines the metaclass for TInterfacedObject (see page 22276).
TLongWordDynArray (see page 22616)	This is type System::TLongWordDynArray.
TMemoryMap (see page 22617)	TMemoryMap is a map of an application's address space as an array of TChunkStatus (see page 22554) values.
TModuleUnloadProc (see page 22617)	TModuleUnloadProc dehines a procedure that can be called during package unload.
TModuleUnloadProcLW (see page 22618)	This is type System.TModuleUnloadProcLW.
TPCharArray (see page 22618)	This is type System.TPCharArray.
TResourceHandle (see page 22619)	This is type System.TResourceHandle.
TSafeCallErrorProc (see page 22619)	This is type System.TSafeCallErrorProc.
TShortIntDynArray (see page 22619)	This is type System::TShortIntDynArray.
TSingleDynArray (see page 22620)	This is type System::TSingleDynArray.
TSmallBlockTypeStates (see page 22620)	TSmallBlockTypeStates is an array of small memory block information records.
TSmallIntDynArray (see page 22620)	This is type System::TSmallIntDynArray.
TStringDynArray (see page 22621)	This is type System::TStringDynArray.
TSystemThreadEndProc (see page 22621)	This is type System.TSystemThreadEndProc.
TSystemThreadFuncProc (see page 22621)	This is type System.TSystemThreadFuncProc.
TTextBuf (see page 22622)	This is type System.TTextBuf.
TTextIOFunc (see page 22622)	This is type System.TTextIOFunc.
TThreadFunc (see page 22622)	TThreadFunc defines a new thread of execution.
TUCS4CharArray (see page 22623)	This is type System.TUCS4CharArray.
TVarArrayBoundArray (see page 22623)	This is type System.TVarArrayBoundArray.
TVarArrayCoorArray (see page 22623)	This is type System.TVarArrayCoorArray.
TVarOp (see page 22624)	TVarOp defines a Variant (see page 22307) operation.
TVarType (see page 22625)	This is type System.TVarType.
TWideStringDynArray (see page 22625)	This is type System::TWideStringDynArray.
TWordDynArray (see page 22626)	This is type System::TWordDynArray.
UCS2Char (see page 22626)	This is type System.UCS2Char.
UCS4Char (see page 22626)	UCS4Char is used for working with 4–byte Unicode characters.
UCS4String (see page 22627)	UCS4String is used for working with 4–byte Unicode characters.
UnitEntryTable (see page 22627)	This is type System.UnitEntryTable.
UTF8String (see page 22627)	This is type System.UTF8String.

Variables

Name	Description
AbstractErrorProc (see page 22630)	AbstractErrorProc defines a callback routine for abstract method call errors.
AllocMemCount (see page 22630)	Represents the total number of allocated memory blocks in a Delphi application.
AllocMemSize (see page 22631)	Represents the total size of allocated memory blocks.
AssertErrorProc (see page 22633)	Points to the assertion error-handler.
CmdLine (see page 22635)	Points to the command-line arguments specified when the application is invoked.
CmdShow (see page 22635)	Specifies the flags for displaying the application's main window.
DebugHook (see page 22636)	DebugHook is set if an application is running under the IDE debugger.
Default8087CW (see page 22636)	Specifies the Default 8087 control word.
DefaultTextLineBreakStyle (see page 22636)	Specifies the characters that are used by default to separate lines in text.
DispCallByIDProc (see page 22637)	This is variable System.DispCallByIDProc.
DLLProc (see page 22637)	Points to a procedure invoked by a DLL entry point.
envp (see page 22638)	This is variable System.envp.
ErrorAddr (see page 22638)	Contains the address of a statement causing a runtime error.
ErrorProc (see page 22639)	Points to the RTL runtime error handler.
ErrOutput (see page 22640)	ErrOutput specifies a write-only Text file associated with the process's error output file.
ExceptClsProc (see page 22640)	ExceptClsProc defines a pointer to a function called during exception handling.
ExceptionClass (see page 22641)	This is variable System.ExceptionClass.
ExceptObjProc (see page 22641)	ExceptObjProc defines a pointer to a function called during exception handling.
ExceptProc (see page 22641)	Points to the lowest-level exception handler.

ExitCode (see page 22642)	Contains the application's exit code.
ExitProc (see page 22643)	Points to a program's exit procedure (provided for backward compatibility).
ExitProcessProc (see page 22644)	Specifies the last procedure to execute before the application shuts down.
FileMode (see page 22644)	Indicates the access mode on typed and untyped files opened by the Reset (see page 22511) procedure.
HeapAllocFlags (see page 22645)	Indicates how the memory manager obtains memory from the operating system.
HPrevInst (see page 22645)	This is variable System.HPrevInst.
InitProc (see page 22646)	InitProc is the last installed initialization procedure.
Input (see page 22647)	Specifies a read-only Text file associated with the processes standard Input file.
IsConsole (see page 22647)	Indicates whether the module was compiled as a console application.
IsLibrary (see page 22648)	Indicates whether the module is a shared library.
IsMultiThread (see page 22648)	Indicates whether the application spawned additional threads using BeginThread (see page 22422) or TThread objects.
JITEnable (see page 22649)	Controls (see page 5711) when the just-in-time debugger is called.
LibModuleList (see page 22650)	Lists module records for enumeration procedures.
MainInstance (see page 22650)	Indicates the instance handle for the main executable.
MainThreadID (see page 22651)	Indicates the instance handle for the current module's main thread of execution.
MaxDispArgs (see page 22651)	This is variable System::MaxDispArgs.
ModuleUnloadList (see page 22651)	This is variable System.ModuleUnloadList.
NeverSleepOnMMThreadContention (see page 22652)	Dictates memory manager behavior during a thread contention.
NoErrMsg (see page 22652)	Controls (see page 5711) whether the application displays an error message when a runtime error occurs.
Output (see page 22653)	Specifies a write-only Text file associated with the process's standard output file.
RaiseExceptionProc (see page 22653)	Points to RaiseException Windows function.
RandSeed (see page 22654)	RandSeed stores the built-in random number generator's seed.
ReportMemoryLeaksOnShutdown (see page 22654)	Report memory leaks on shutdown.
RTLUnwindProc (see page 22655)	This is variable System.RTLUnwindProc.
SafeCallErrorProc (see page 22655)	SafeCallErrorProc defines a procedure that is called when a safecall routine incurs an error.
SystemThreadEndProc (see page 22656)	This is variable System.SystemThreadEndProc.
SystemThreadFuncProc (see page 22656)	This is variable System.SystemThreadFuncProc.
Test8086 (see page 22656)	This is variable System.Test8086.
Test8087 (see page 22657)	This is variable System.Test8087.
TestFDIV (see page 22657)	This is variable System.TestFDIV.
VarAddRefProc (see page 22657)	This is variable System.VarAddRefProc.
VarClearProc (see page 22658)	This is variable System.VarClearProc.
VarCopyProc (see page 22658)	This is variable System.VarCopyProc.
VarToLStrProc (see page 22658)	This is variable System.VarToLStrProc.
VarToWStrProc (see page 22659)	This is variable System.VarToWStrProc.

158.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	AnsiString (see page 22093)	System::AnsiString is the C++ analog for the Delphi long string type.
	AutoCmd (see page 22127)	System::AutoCmd represents a command that can be executed by an Automation interface wrapped by a Variant (see page 22307) class.
	Currency (see page 22145)	System::Currency is the C++ implementation of the Delphi System::Currency data type.
	CurrencyBase (see page 22166)	This is class System::CurrencyBase.
	DynArrayException (see page 22182)	This is class System::DynArrayException.
	DynArrayNullData (see page 22183)	This is class System::DynArrayNullData.
	DynArrayOutOfRange (see page 22183)	This is class System::DynArrayOutOfRange.
	Function (see page 22184)	System::Function represents a method on an Automation interface that returns a value.
	NamedParm (see page 22187)	NamedParm represents a named parameter assigned to a TAutoCmd descendant.
	Procedure (see page 22230)	System::Procedure represents a method on an Automation interface that does not return a value.
	PropertyGet (see page 22233)	PropertyGet represents a property access method on an Automation interface.
	PropertySet (see page 22236)	PropertySet represents a property access method on an Automation interface.
	TAggregatedObject (see page 22248)	System::TAggregatedObject provides the functionality for an inner object of an aggregate by implementing the IInterface (see page 22396) methods to delegate to the controlling IInterface (see page 22396).
	TClassHelperBase (see page 22251)	This is class System.TClassHelperBase.
	TContainedObject (see page 22253)	System::TContainedObject implements the IInterface (see page 22396) interface as an inner object in an aggregate.
	TDateTime (see page 22256)	System::TDateTime is the C++ analog for the Delphi System::TDateTime data type.
	TDateTimeBase (see page 22275)	This is class System::TDateTimeBase.
	TInterfacedObject (see page 22276)	System::TInterfacedObject is a base for simple classes that need a basic IInterface (see page 22396) implementation.
	TMetaClass (see page 22281)	System::TMetaClass is the C++ representation of the Delphi class-reference type.
	TObject (see page 22285)	System::TObject is the ultimate ancestor of all objects and components.
	Variant (see page 22307)	System::Variant is a C++ implementation of the Delphi intrinsic type System::Variant.
	VariantError (see page 22375)	This is class System::VariantError.
	WideString (see page 22377)	System::WideString represents strings of 16-bit characters.

Structures

	Name	Description
	StaticArray (see page 22245)	This is class System::StaticArray.

Templates

	Name	Description
	DelphiInterface (see page 22167)	System::DelphiInterface is a C++ implementation of the Delphi interface.
	DynamicArray (see page 22172)	System::DynamicArray is the C++ analog for the Delphi dynamic array type.
	OleVariant (see page 22193)	System::OleVariant is a C++ implementation of the Delphi intrinsic type System::OleVariant.
	OpenArray (see page 22216)	This is class System::OpenArray.
	OpenArrayCount (see page 22223)	This is class System::OpenArrayCount.
	SmallString (see page 22239)	This is class System::SmallString.

	SmallStringBase (see page 22242)	This is class System::SmallStringBase.
	sPointer (see page 22243)	This is class System::sPointer.

158.1.1 AnsiString Class

System::AnsiString is the C++ analog for the Delphi long string type.

Class Hierarchy

System::AnsiString

Pascal

```
AnsiString = class;
```

C++

```
class AnsiString;
```

File

dstring.h

Description

Delphi utilizes several string types. The most important is the long string type, more commonly known as System::AnsiString. Support for this type includes the following features:

strings as large as available memory;

efficient use of memory through shared references;

routines and operators that evaluate strings based on the current locale.

Despite its name, System::AnsiString is not restricted to the ANSI character set, and can utilize any character set supported by the current locale definition, including multi-byte character sets.

System::AnsiString variables that have not been assigned an initial value contain a zero-length string.

To utilize the C++ streaming operators (**<<** and **>>**) with System::AnsiString, you must `iostream` before including `system.hpp` and use `AnsiString::c_str (see page 22105)()` to return the internal string representation. The following example demonstrates the usage of AnsiString with stream operators:

```
#include <iostream>
#include <system.hpp>
int main() {
  AnsiString HelloStr = "hello";
  std::cout << HelloStr.c_str();
}
```

Note: Delphi also supports System::AnsiString, but implements it as a primitive type rather than a class. By default, variables declared as type String are System::AnsiString.

See Also

System::WideString (see page 22377)

System::AnsiString::[] (see page 22123)

System::AnsiString::= (see page 22125)

System::AnsiString::== (see page 22126)

System::AnsiString::!= (see page 22123)

System::AnsiString::> (see page 22126)

System::AnsiString::>= (see page 22126)

System::AnsiString::< (see page 22124)

System::AnsiString::<= (see page 22125)

System::AnsiString::+ (see page 22124)

System::AnsiString::+= (see page 22124)

158.1.1.1 AnsiString Members

The following tables list the members exposed by AnsiString.

Enumerations

	Name	Description
	TStringFloatFormat (see page 22101)	This is record System::AnsiString::TStringFloatFormat.
	TStringMbcsByteType (see page 22101)	This is record System::AnsiString::TStringMbcsByteType.

Structures

	Name	Description
	StrRec (see page 22100)	This is record System::AnsiString::StrRec.

Protected Methods

	Name	Description
	GetRec (see page 22112)	This is the overview for the GetRec method overload.
	ThrowIfOutOfRange (see page 22119)	This is ThrowIfOutOfRange, a member of class AnsiString.

Public Friends

Name	Description
AnsiString __fastcall operator +(const char*, const AnsiString& rhs) (see page 22127)	This is friend AnsiString __fastcall operator +(const char*, const AnsiString& rhs).

Public Methods

	Name	Description
	~AnsiString (see page 22096)	This is ~AnsiString, a member of class AnsiString.
	AnsiCompare (see page 22103)	Compares the System::AnsiString::AnsiString (see page 22097) to another specified string (case-sensitively).
	AnsiCompareIC (see page 22104)	Compares the System::AnsiString::AnsiString (see page 22097) to another specified string (case insensitively).
	AnsiLastChar (see page 22104)	Returns a pointer to the last full character in the System::AnsiString::AnsiString (see page 22097).
	AnsiPos (see page 22104)	Returns the index at which the specified substring begins.
	AnsiString (see page 22096)	This is the overview for the AnsiString constructor overload.
	ByteType (see page 22105)	Indicates the kind of byte that occupies a specified position in the System::AnsiString::AnsiString (see page 22097).
	c_str (see page 22105)	Returns the System::AnsiString::AnsiString (see page 22097) converted to null-terminated character array.
	cat_printf (see page 22106)	Appends to the value of the System::AnsiString::AnsiString (see page 22097) given a format string and its arguments.
	cat_sprintf (see page 22106)	Appends the result of a given format string and its arguments to the end of the System::AnsiString::AnsiString (see page 22097).
	cat_vprintf (see page 22107)	Appends the result of a given format string and its arguments to the end of the System::AnsiString::AnsiString (see page 22097).
	CurrToStr (see page 22107)	Converts a currency value to its string representation.

	CurrToStrF (see page 22108)	Converts a currency value to its string representation using the indicated float format convention.
	data (see page 22108)	Returns a pointer to the underlying string System::AnsiString::data.
	Delete (see page 22108)	Removes a specified number of characters from the string.
	FloatToStrF (see page 22109)	Returns the string representation of a floating-point value.
	FmtLoadStr (see page 22110)	Formats output using a resourced format string.
	Format (see page 22110)	System::AnsiString::Formats a string assembled from an System::AnsiString::AnsiString (see page 22097), format, and a series of array arguments, args.
	FormatFloat (see page 22110)	Formats the floating-point value given by value using the format string given by format.
	Insert (see page 22112)	System::AnsiString::Inserts a specified string into the System::AnsiString::AnsiString (see page 22097).
	IntToHex (see page 22113)	Converts a number into a string containing the number's hexadecimal (base 16) representation.
	IsDelimiter (see page 22113)	Returns true if the specified character matches any character in the delimiters string.
	IsEmpty (see page 22113)	Returns true if the System::AnsiString::AnsiString (see page 22097) is empty.
	IsLeadByte (see page 22114)	Returns true if the specified byte is a lead byte.
	IsPathDelimiter (see page 22114)	Returns true if the specified character is '\'.
	IsTrailByte (see page 22114)	Returns true if the specified byte is of type mbTrailByte.
	LastDelimiter (see page 22115)	Returns the index of the rightmost character that matches any character in the delimiters string.
	Length (see page 22115)	Returns the length of the System::AnsiString::AnsiString (see page 22097).
	LoadStr (see page 22115)	This is the overview for the LoadStr method overload.
	LoadString (see page 22116)	Loads a string from the compiled resources of a specified module.
	LowerCase (see page 22116)	Returns a lower-case version of System::AnsiString::AnsiString (see page 22097).
	Pos (see page 22117)	Returns the index at which a specified substring begins.
	printf (see page 22117)	Sets the value of the System::AnsiString::AnsiString (see page 22097) given a format string and its arguments.
	SetLength (see page 22117)	Changes the length of the System::AnsiString::AnsiString (see page 22097).
	sprintf (see page 22118)	Sets the value of the System::AnsiString::AnsiString (see page 22097) given a format string and its arguments.
	StringOfChar (see page 22118)	Returns a string with the specified number of characters.
	SubString (see page 22118)	Returns a specified substring of the System::AnsiString::AnsiString (see page 22097).
	ToDouble (see page 22119)	Converts the string to a floating-point value.
	ToInt (see page 22119)	Converts a string to an integer, throwing an exception if the string is invalid.
	ToIntDef (see page 22120)	Converts the string to an integer, returning a default value if the string is invalid.
	Trim (see page 22120)	Returns a new System::AnsiString::AnsiString (see page 22097) without leading spaces, trailing spaces, or control characters.
	TrimLeft (see page 22120)	Returns a new System::AnsiString::AnsiString (see page 22097) without leading spaces or control characters.
	TrimRight (see page 22120)	Returns a new System::AnsiString::AnsiString (see page 22097) without trailing spaces or control characters.
	Unique (see page 22121)	Makes the System::AnsiString::AnsiString (see page 22097) unique.
	UpperCase (see page 22121)	Returns an upper-case version of this System::AnsiString::AnsiString (see page 22097).
	vprintf (see page 22121)	Sets the value of the System::AnsiString::AnsiString (see page 22097) given a format string and its arguments.
	WideChar (see page 22122)	Converts the System::AnsiString::AnsiString (see page 22097) to a wide-character array.
	WideCharBufSize (see page 22122)	Returns the size of the buffer required by the WideChar (see page 22122) function.

Public Operators

	Name	Description
	!= (see page 22123)	Compares two strings.
	[] (see page 22123)	This is the overview for the [] operator overload.

	+ (see page 22124)	Concatenates two System::AnsiString::AnsiString (see page 22097) values.
	+= (see page 22124)	Append (see page 22415) an System::AnsiString::AnsiString (see page 22097).
	< (see page 22124)	Compares two strings.
	<= (see page 22125)	Compares two strings.
	= (see page 22125)	Assigns the string rhs to this System::AnsiString::AnsiString (see page 22097).
	== (see page 22126)	Compares the System::AnsiString::AnsiString (see page 22097) with a specified string, returning true if they are the same.
	> (see page 22126)	Compares two strings.
	>= (see page 22126)	Compares two strings.

158.1.1.2 AnsiString::~AnsiString Destructor

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~AnsiString();
```

Description

This is ~AnsiString, a member of class AnsiString.

158.1.1.3 AnsiString Constructor

This is the overview for the AnsiString constructor overload.

Overload List

	Name	Description
	AnsiString::AnsiString () (see page 22097)	This is AnsiString, a member of class AnsiString.
	AnsiString::AnsiString (__int64) (see page 22097)	This is AnsiString, a member of class AnsiString.
	AnsiString::AnsiString (AnsiString&) (see page 22097)	This is AnsiString, a member of class AnsiString.
	AnsiString::AnsiString (char) (see page 22098)	This is AnsiString, a member of class AnsiString.
	AnsiString::AnsiString (char*) (see page 22098)	This is AnsiString, a member of class AnsiString.
	AnsiString::AnsiString (char*, unsigned int) (see page 22098)	This is AnsiString, a member of class AnsiString.
	AnsiString::AnsiString (double) (see page 22098)	This is AnsiString, a member of class AnsiString.
	AnsiString::AnsiString (int) (see page 22098)	This is AnsiString, a member of class AnsiString.
	AnsiString::AnsiString (long) (see page 22099)	This is AnsiString, a member of class AnsiString.
	AnsiString::AnsiString (short) (see page 22099)	This is AnsiString, a member of class AnsiString.
	AnsiString::AnsiString (unsigned __int64) (see page 22099)	This is AnsiString, a member of class AnsiString.
	AnsiString::AnsiString (unsigned int) (see page 22099)	This is AnsiString, a member of class AnsiString.
	AnsiString::AnsiString (unsigned long) (see page 22100)	This is AnsiString, a member of class AnsiString.

	AnsiString::AnsiString (unsigned short) (see page 22100)	This is AnsiString, a member of class AnsiString.
	AnsiString::AnsiString (wchar_t*) (see page 22100)	This is AnsiString, a member of class AnsiString.
	AnsiString::AnsiString (WideString &) (see page 22097)	This is AnsiString, a member of class AnsiString.

158.1.1.3.1 AnsiString::AnsiString Constructor ()

Pascal

```
constructor Create;
```

C++

```
__fastcall AnsiString();
```

Description

This is AnsiString, a member of class AnsiString.

158.1.1.3.2 AnsiString::AnsiString Constructor (AnsiString&)

Pascal

```
constructor Create(const src: AnsiString&);
```

C++

```
__fastcall AnsiString(const AnsiString& src);
```

Description

This is AnsiString, a member of class AnsiString.

158.1.1.3.3 AnsiString::AnsiString Constructor (WideString &)

Pascal

```
constructor Create(const src: WideString &);
```

C++

```
__fastcall AnsiString(const WideString & src);
```

Description

This is AnsiString, a member of class AnsiString.

158.1.1.3.4 AnsiString::AnsiString Constructor (__int64)

Pascal

```
constructor Create(: __int64);
```

C++

```
__fastcall AnsiString(__int64);
```

Description

This is AnsiString, a member of class AnsiString.

158.1.1.3.5 AnsiString::AnsiString Constructor (char)

Pascal

```
constructor Create(src: char);
```

C++

```
__fastcall AnsiString(char src);
```

Description

This is AnsiString, a member of class AnsiString.

158.1.1.3.6 AnsiString::AnsiString Constructor (char*)

Pascal

```
constructor Create(const src: char*);
```

C++

```
__fastcall AnsiString(const char* src);
```

Description

This is AnsiString, a member of class AnsiString.

158.1.1.3.7 AnsiString::AnsiString Constructor (char*, unsigned int)

Pascal

```
constructor Create(const src: char*; len: unsigned int);
```

C++

```
__fastcall AnsiString(const char* src, unsigned int len);
```

Description

This is AnsiString, a member of class AnsiString.

158.1.1.3.8 AnsiString::AnsiString Constructor (double)

Pascal

```
constructor Create(src: double);
```

C++

```
__fastcall AnsiString(double src);
```

Description

This is AnsiString, a member of class AnsiString.

158.1.1.3.9 AnsiString::AnsiString Constructor (int)

Pascal

```
constructor Create(src: Integer);
```

C++

```
__fastcall AnsiString(int src);
```

Description

This is AnsiString, a member of class AnsiString.

158.1.1.3.10 AnsiString::AnsiString Constructor (long)

Pascal

```
constructor Create(: long);
```

C++

```
__fastcall AnsiString(long);
```

Description

This is AnsiString, a member of class AnsiString.

158.1.1.3.11 AnsiString::AnsiString Constructor (short)

Pascal

```
constructor Create(: short);
```

C++

```
__fastcall AnsiString(short);
```

Description

This is AnsiString, a member of class AnsiString.

158.1.1.3.12 AnsiString::AnsiString Constructor (unsigned __int64)

Pascal

```
constructor Create(: unsigned __int64);
```

C++

```
__fastcall AnsiString(unsigned __int64);
```

Description

This is AnsiString, a member of class AnsiString.

158.1.1.3.13 AnsiString::AnsiString Constructor (unsigned int)

Pascal

```
constructor Create(: unsigned int);
```

C++

```
__fastcall AnsiString(unsigned int);
```

Description

This is AnsiString, a member of class AnsiString.

158.1.1.3.14 AnsiString::AnsiString Constructor (unsigned long)

Pascal

```
constructor Create(: unsigned long);
```

C++

```
__fastcall AnsiString(unsigned long);
```

Description

This is AnsiString, a member of class AnsiString.

158.1.1.3.15 AnsiString::AnsiString Constructor (unsigned short)

Pascal

```
constructor Create(: unsigned short);
```

C++

```
__fastcall AnsiString(unsigned short);
```

Description

This is AnsiString, a member of class AnsiString.

158.1.1.3.16 AnsiString::AnsiString Constructor (wchar_t*)

Pascal

```
constructor Create(const src: wchar_t*);
```

C++

```
__fastcall AnsiString(const wchar_t* src);
```

Description

This is AnsiString, a member of class AnsiString.

158.1.1.4 AnsiString Structures

The structures of the AnsiString class are listed here.

Structures

	Name	Description
	StrRec (see page 22100)	This is record System::AnsiString::StrRec.

158.1.1.4.1 System::AnsiString::StrRec Structure

Pascal

```
StrRec = record
  allocSiz: Integer;
  refCnt: Integer;
  length: Integer;
end;
```

C++

```
struct StrRec {
  int allocSiz;
  int refCnt;
  int length;
};
```

File

dstring.h

Description

This is record System::AnsiString::StrRec.

158.1.1.5 AnsiString Enumerations

The enumerations of the AnsiString class are listed here.

Enumerations

	Name	Description
	TStringFloatFormat (see page 22101)	This is record System::AnsiString::TStringFloatFormat.
	TStringMbcsByteType (see page 22101)	This is record System::AnsiString::TStringMbcsByteType.

158.1.1.5.1 System::AnsiString::TStringFloatFormat Enumeration

Pascal

```
TStringFloatFormat = (
  sffGeneral,
  sffExponent,
  sffFixed,
  sffNumber,
  sffCurrency
);
```

C++

```
enum TStringFloatFormat {
  sffGeneral,
  sffExponent,
  sffFixed,
  sffNumber,
  sffCurrency
};
```

File

dstring.h

Description

This is record System::AnsiString::TStringFloatFormat.

158.1.1.5.2 System::AnsiString::TStringMbcsByteType Enumeration

Pascal

```
TStringMbcsByteType = (
  mbSingleByte,
  mbLeadByte,
```

```
    mbTrailByte
  );
```

C++

```
enum TStringMbcsByteType {
  mbSingleByte,
  mbLeadByte,
  mbTrailByte
};
```

File

dstring.h

Description

This is record System::AnsiString::TStringMbcsByteType.

158.1.1.6 AnsiString Methods

The methods of the AnsiString class are listed here.

Protected Methods

	Name	Description
	GetRec (see page 22112)	This is the overview for the GetRec method overload.
	ThrowIfOutOfRange (see page 22119)	This is ThrowIfOutOfRange, a member of class AnsiString.

Public Methods

	Name	Description
	AnsiCompare (see page 22103)	Compares the System::AnsiString::AnsiString (see page 22097) to another specified string (case-sensitively).
	AnsiCompareIC (see page 22104)	Compares the System::AnsiString::AnsiString (see page 22097) to another specified string (case insensitively).
	AnsiLastChar (see page 22104)	Returns a pointer to the last full character in the System::AnsiString::AnsiString (see page 22097).
	AnsiPos (see page 22104)	Returns the index at which the specified substring begins.
	ByteType (see page 22105)	Indicates the kind of byte that occupies a specified position in the System::AnsiString::AnsiString (see page 22097).
	c_str (see page 22105)	Returns the System::AnsiString::AnsiString (see page 22097) converted to null-terminated character array.
	cat_printf (see page 22106)	Appends to the value of the System::AnsiString::AnsiString (see page 22097) given a format string and its arguments.
	cat_sprintf (see page 22106)	Appends the result of a given format string and its arguments to the end of the System::AnsiString::AnsiString (see page 22097).
	cat_vprintf (see page 22107)	Appends the result of a given format string and its arguments to the end of the System::AnsiString::AnsiString (see page 22097).
	CurrToStr (see page 22107)	Converts a currency value to its string representation.
	CurrToStrF (see page 22108)	Converts a currency value to its string representation using the indicated float format convention.
	data (see page 22108)	Returns a pointer to the underlying string System::AnsiString::data.
	Delete (see page 22108)	Removes a specified number of characters from the string.
	FloatToStrF (see page 22109)	Returns the string representation of a floating-point value.
	FmtLoadStr (see page 22110)	Formats output using a resourced format string.
	Format (see page 22110)	System::AnsiString::Formats a string assembled from an System::AnsiString::AnsiString (see page 22097), format, and a series of array arguments, args.
	FormatFloat (see page 22110)	Formats the floating-point value given by value using the format string given by format.
	Insert (see page 22112)	System::AnsiString::Inserts a specified string into the System::AnsiString::AnsiString (see page 22097).

	IntToHex (see page 22113)	Converts a number into a string containing the number's hexadecimal (base 16) representation.
	IsDelimiter (see page 22113)	Returns true if the specified character matches any character in the delimiters string.
	IsEmpty (see page 22113)	Returns true if the System::AnsiString::AnsiString (see page 22097) is empty.
	IsLeadByte (see page 22114)	Returns true if the specified byte is a lead byte.
	IsPathDelimiter (see page 22114)	Returns true if the specified character is '\'.
	IsTrailByte (see page 22114)	Returns true if the specified byte is of type mbTrailByte.
	LastDelimiter (see page 22115)	Returns the index of the rightmost character that matches any character in the delimiters string.
	Length (see page 22115)	Returns the length of the System::AnsiString::AnsiString (see page 22097).
	LoadStr (see page 22115)	This is the overview for the LoadStr method overload.
	LoadString (see page 22116)	Loads a string from the compiled resources of a specified module.
	LowerCase (see page 22116)	Returns a lower-case version of System::AnsiString::AnsiString (see page 22097).
	Pos (see page 22117)	Returns the index at which a specified substring begins.
	printf (see page 22117)	Sets the value of the System::AnsiString::AnsiString (see page 22097) given a format string and its arguments.
	SetLength (see page 22117)	Changes the length of the System::AnsiString::AnsiString (see page 22097).
	sprintf (see page 22118)	Sets the value of the System::AnsiString::AnsiString (see page 22097) given a format string and its arguments.
	StringOfChar (see page 22118)	Returns a string with the specified number of characters.
	SubString (see page 22118)	Returns a specified substring of the System::AnsiString::AnsiString (see page 22097).
	ToDouble (see page 22119)	Converts the string to a floating-point value.
	ToInt (see page 22119)	Converts a string to an integer, throwing an exception if the string is invalid.
	ToIntDef (see page 22120)	Converts the string to an integer, returning a default value if the string is invalid.
	Trim (see page 22120)	Returns a new System::AnsiString::AnsiString (see page 22097) without leading spaces, trailing spaces, or control characters.
	TrimLeft (see page 22120)	Returns a new System::AnsiString::AnsiString (see page 22097) without leading spaces or control characters.
	TrimRight (see page 22120)	Returns a new System::AnsiString::AnsiString (see page 22097) without trailing spaces or control characters.
	Unique (see page 22121)	Makes the System::AnsiString::AnsiString (see page 22097) unique.
	UpperCase (see page 22121)	Returns an upper-case version of this System::AnsiString::AnsiString (see page 22097).
	vprintf (see page 22121)	Sets the value of the System::AnsiString::AnsiString (see page 22097) given a format string and its arguments.
	WideChar (see page 22122)	Converts the System::AnsiString::AnsiString (see page 22097) to a wide-character array.
	WideCharBufSize (see page 22122)	Returns the size of the buffer required by the WideChar (see page 22122) function.

158.1.1.6.1 AnsiString::AnsiCompare Method

Compares the System::AnsiString::AnsiString (see page 22097) to another specified string (case-sensitively).

Pascal

```
function AnsiCompare(const rhs: AnsiString&): Integer;
```

C++

```
__fastcall int AnsiCompare(const AnsiString& rhs) const;
```

Description

Compares this System::AnsiString::AnsiString (see page 22097) to rhs, with case sensitivity. The compare operation is controlled by the current Windows locale and is not necessarily the same as using the System::AnsiString::AnsiString (see page 22097) comparison operators. System::AnsiString::AnsiCompare is based on a locale-specific collating order. The return value is shown below:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

158.1.1.6.2 AnsiString::AnsiCompareIC Method

Compares the System::AnsiString::AnsiString (see page 22097) to another specified string (case insensitively).

Pascal

```
function AnsiCompareIC(const rhs: AnsiString&): Integer;
```

C++

```
__fastcall int AnsiCompareIC(const AnsiString& rhs) const;
```

Description

System::AnsiString::AnsiCompareIC compares this System::AnsiString::AnsiString (see page 22097) to rhs, without case sensitivity. The compare operation is controlled by the current Windows locale. The return value is shown below:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

158.1.1.6.3 AnsiString::AnsiLastChar Method

Returns a pointer to the last full character in the System::AnsiString::AnsiString (see page 22097).

Pascal

```
function AnsiLastChar: char*;
```

C++

```
__fastcall char* AnsiLastChar() const;
```

Description

Returns a pointer to the last full character in the System::AnsiString::AnsiString (see page 22097). This function supports multibyte characters.

See Also

System::AnsiString::ByteType (see page 22105)

158.1.1.6.4 AnsiString::AnsiPos Method

Returns the index at which the specified substring begins.

Pascal

```
function AnsiPos(const subStr: AnsiString&): Integer;
```

C++

```
__fastcall int AnsiPos(const AnsiString& subStr) const;
```

Description

Returns the index in the System::AnsiString::AnsiString (see page 22097) at which the substring subStr begins, where 1 is the first character in the string, 2 is the second character, and so on. If the substring is not contained in the System::AnsiString::AnsiString (see page 22097), Pos (see page 22117) returns 0. This member function is the same as Pos (see page 22117) but supports multibyte character strings.

Note: See ByteType (see page 22105) for more information about multibyte characters.

See Also

System::AnsiString::ByteType (see page 22105)

158.1.1.6.5 AnsiString::ByteType Method

Indicates the kind of byte that occupies a specified position in the System::AnsiString::AnsiString (see page 22097).

Pascal

```
function ByteType(index: Integer): TStringMbcsByteType;
```

C++

```
__fastcall TStringMbcsByteType ByteType(int index) const;
```

Description

System::AnsiString::ByteType indicates what kind of byte exists at the byte indicated by index (counting from 0) in the System::AnsiString::AnsiString (see page 22097). These are the possible return values.

Constant	Meaning
mbSingleByte	Single byte character.
mbLeadByte	Lead byte of a multi-byte character.
mbTrailByte	Trailing byte of a multi-byte character.

Some locales, including all Western locales, do not use multi-byte characters. Under those locales, System::AnsiString::ByteType always returns mbSingleByte.

158.1.1.6.6 AnsiString::c_str Method

Returns the System::AnsiString::AnsiString (see page 22097) converted to null-terminated character array.

Pascal

```
function c_str: char*;
```

C++

```
__fastcall char* c_str() const;
```

Description

c_System::AnsiString::c_str returns a pointer a null-terminated character array containing the same characters as the System::AnsiString::AnsiString (see page 22097). If the System::AnsiString::AnsiString (see page 22097) is unassigned, c_System::AnsiString::c_str returns an empty System::AnsiString::c_string (“”).

Usually, the value returned by c_System::AnsiString::c_str points to the internal character array referenced by the data (see page 22108) property. This pointer is valid until the System::AnsiString::c_string is next modified (for example when the SetLength (see page 22117) method is called or the System::AnsiString::AnsiString (see page 22097) goes out of scope). However, if the internal array is NULL, c_System::AnsiString::c_str returns a pointer to a System::AnsiString::c_string literal.

The c_System::AnsiString::c_str method is intended primarily for reading the value of the System::AnsiString::AnsiString (see page 22097). To modify the System::AnsiString::c_string's value, use the [] operator or System::AnsiString::AnsiString (see page 22097) methods such as Insert (see page 22112) and Delete (see page 22108).

158.1.1.6.7 AnsiString::cat_printf Method

Appends to the value of the System::AnsiString::AnsiString (see page 22097) given a format string and its arguments.

Pascal

```
function cat_printf(const format: char*; : ...): Integer; cdecl;
```

C++

```
__cdecl int cat_printf(const char* format, ...);
```

Description

Use cat_System::AnsiString::cat_printf to modify the value of the System::AnsiString::AnsiString (see page 22097) by appending the result of using a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter. This method returns the number of bytes appended to the System::AnsiString::AnsiString (see page 22097).

See Also

System::AnsiString::Format (see page 22110)

System::AnsiString::cat_sprintf (see page 22106)

System::AnsiString::cat_printf

System::AnsiString::cat_vprintf (see page 22107)

158.1.1.6.8 AnsiString::cat_sprintf Method

Appends the result of a given format string and its arguments to the end of the System::AnsiString::AnsiString (see page 22097).

Pascal

```
function cat_sprintf(const format: char*; : ...): AnsiString&; cdecl;
```

C++

```
__cdecl AnsiString& cat_sprintf(const char* format, ...);
```

Description

Use cat_System::AnsiString::cat_sprintf to append to the value of the System::AnsiString::AnsiString (see page 22097) given a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter.

For example, the following code

```
AnsiString Salutation = "Mr."
AnsiString s = "";
s.cat_sprintf("Dear %s ", Salutation);
s.cat_sprintf("%s,", "Smith");
```

results in an System::AnsiString::AnsiString (see page 22097) with the value

```
Dear Mr. Smith,
```

This method returns a reference to the System::AnsiString::AnsiString (see page 22097) (*this) with the resulting value.

See Also

System::AnsiString::Format (see page 22110)

System::AnsiString::cat_sprintf

System::AnsiString::cat_printf (see page 22106)

System::AnsiString::cat_vprintf (see page 22107)

158.1.1.6.9 AnsiString::cat_vprintf Method

Appends the result of a given format string and its arguments to the end of the System::AnsiString::AnsiString (see page 22097).

Pascal

```
function cat_vprintf(const format: char*; : va_list): Integer; cdecl;
```

C++

```
__cdecl int cat_vprintf(const char* format, va_list);
```

Description

Use cat_System::AnsiString::cat_vprintf to append to the value of the System::AnsiString::AnsiString (see page 22097) given a standard C++ format specifier. Pass the values to any arguments in the format specifier as a variable argument list that was created using va_start.

This method returns the number of bytes appended to the System::AnsiString::AnsiString (see page 22097).

See Also

System::AnsiString::Format (see page 22110)

System::AnsiString::cat_vprintf

System::AnsiString::cat_printf (see page 22106)

System::AnsiString::cat_sprintf (see page 22106)

arg

start

158.1.1.6.10 AnsiString::CurrToStr Method

Converts a currency value to its string representation.

Pascal

```
function CurrToStr(value: Currency): String; static;
```

C++

```
static __fastcall AnsiString CurrToStr(Currency value);
```

Description

System::AnsiString::CurrToStr converts the currency value given by value to its string representation. The conversion using the sffGeneral floating-point conversion rules. For further details, see System::AnsiString::CurrToStrF (see page 22108).

See Also

System::AnsiString::CurrToStrF (see page 22108)

158.1.1.6.11 AnsiString::CurrToStrF Method

Converts a currency value to its string representation using the indicated float format convention.

Pascal

```
function CurrToStrF(value: Currency; format: TStringFloatFormat; digits: Integer): String;
static;
```

C++

```
static __fastcall AnsiString CurrToStrF(Currency value, TStringFloatFormat format, int digits);
```

Description

System::AnsiString::CurrToStrF converts the currency value given by value to its string representation using the indicated float format convention. A call to System::AnsiString::CurrToStrF corresponds to a call to FloatToStrF (see page 22109) with an implied precision of 19 digits.

158.1.1.6.12 AnsiString::data Method

Returns a pointer to the underlying string System::AnsiString::data.

Pascal

```
function data: const void*;
```

C++

```
__fastcall const void* data() const;
```

Description

System::AnsiString::data() returns a pointer a null-terminated character array that the System::AnsiString::AnsiString (see page 22097) uses to store its value. This is the same value as the one returned by the c_str (see page 22105)() method, except that System::AnsiString::data() returns a NULL pointer when the value is an empty string.

Note: Because System::AnsiString::data() returns NULL instead of an empty string, it is useful for C++ functions that expect either a valid (non-empty) char* or NULL:

```
AnsiString x;
ShellExecute(0, static_cast<const char*>(x.data()), "File.txt", NULL, NULL, SW_SHOW);
```

See Also

System::AnsiString::c_str (see page 22105)

158.1.1.6.13 AnsiString::Delete Method

Removes a specified number of characters from the string.

Pascal

```
function Delete(index: Integer; count: Integer): AnsiString&;
```

C++

```
__fastcall AnsiString& Delete(int index, int count);
```

Description

System::AnsiString::Delete modifies the System::AnsiString::AnsiString (see page 22097) to remove count characters from the string beginning with the character at index, where 1 is the index of the first character. It returns the resulting modified string

(*this).

If index is larger than the length of the System::AnsiString::AnsiString (see page 22097) or less than 1, no characters are deleted.

If count specifies more characters than remain starting at the index, System::AnsiString::Delete removes the rest of the string. If count is less than 0, no characters are deleted.

158.1.1.6.14 AnsiString::FloatToStrF Method

Returns the string representation of a floating-point value.

Pascal

```
function FloatToStrF(value: long double; format: TStringFloatFormat; precision: Integer;
digits: Integer): String; static;
```

C++

```
static __fastcall AnsiString FloatToStrF(long double value, TStringFloatFormat format, int
precision, int digits);
```

Description

The format parameter controls the format of the resulting string. The precision parameter specifies the precision of the given value. It should be 7 or less for values of type float, 15 or less for values of type double, and 18 or less for values of type Extended. The meaning of the digits parameter depends on the particular format selected.

The possible values of the format parameter, and the meaning of each, are described below.

Value	Meaning
sffGeneral	General number format. The value is converted to the shortest possible decimal string using fixed or scientific format. Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format, and the digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).
sffExponent	Scientific format. The value is converted to a string of the form "-d.ddd...E+dddd". The resulting string starts with a minus sign if the number is negative, and one digit always precedes the decimal point. The total number of digits in the resulting string (including the one before the decimal point) is given by the precision parameter. The "E" exponent character in the resulting string is always followed by a plus or minus sign and up to four digits. The digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).
sffFixed	Fixed point format. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative, and at least one digit always precedes the decimal point. The number of digits after the decimal point is given by the digits parameter--it must be between 0 and 18. If the number of digits to the left of the decimal point is greater than the specified precision, the resulting value will use scientific format.
sffNumber	Number format. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The sffNumber format corresponds to the sffFixed format, except that the resulting string contains thousand separators.
sffCurrency	Currency (see page 22145) format. The value is converted to a string that represents a currency amount. The conversion is controlled by the CurrencyString, CurrencyFormat, NegCurrFormat, ThousandSeparator, and DecimalSeparator global variables, all of which are initialized from the Currency (see page 22145) Format (see page 22110) in the International section of the Windows Control Panels. The number of digits after the decimal point is given by the Digits parameter--it must be between 0 and 18.

For all formats, the actual characters used as decimal and thousand separators are obtained from the DecimalSeparator and ThousandSeparator global variables.

If the given value is a NAN (not-a-number), the resulting string is 'NAN'. If the given value is positive infinity, the resulting string is 'INF'. If the given value is negative infinity, the resulting string is '-INF'.

158.1.1.6.15 AnsiString::FmtLoadStr Method

Formats output using a resourced format string.

Pascal

```
function FmtLoadStr(ident: Integer; const args: TVarRec *; size: Integer): String; static;
```

C++

```
static __fastcall AnsiString FmtLoadStr(int ident, const TVarRec * args, int size);
```

Description

System::AnsiString::FmtLoadStr loads a string from a program's resource string table and uses that string, plus the args array, as a parameter to Format (see page 22110). Ident is the string resource ID of the desired format string.

See Also

System::AnsiString::LoadString (see page 22116)

System::AnsiString::LoadStr (see page 22115)

158.1.1.6.16 AnsiString::Format Method

System::AnsiString::Formats a string assembled from an System::AnsiString::AnsiString (see page 22097), format, and a series of array arguments, args.

Pascal

```
function Format(const format: AnsiString&; const args: TVarRec *; size: Integer): String;
static;
```

C++

```
static __fastcall AnsiString Format(const AnsiString& format, const TVarRec * args, int size);
```

Description

Use System::AnsiString::Format to obtain an System::AnsiString::AnsiString (see page 22097) whose value is the result of applying an array of arguments (args) to a given a format string (format). The last parameter (size) is the index of the last element in args (one less than the number of arguments).

Note that System::AnsiString::Format is a static method. The returned value is a newly allocated System::AnsiString::AnsiString (see page 22097), even if the method is called from an existing System::AnsiString::AnsiString (see page 22097) instance.

For information on how System::AnsiString::Format assembles a final string from its format string and arguments, see System::AnsiString::Format Strings.

See Also

System::AnsiString::printf (see page 22117)

System::AnsiString::sprintf (see page 22118)

158.1.1.6.17 AnsiString::FormatFloat Method

Formats the floating-point value given by value using the format string given by format.

Pascal

```
function FormatFloat(const format: AnsiString&; const value: long double&): String; static;
```

C++

```
static __fastcall AnsiString FormatFloat(const AnsiString& format, const long double& value);
```

Description

The following format specifiers are supported in the format string:

Specifier	Represents
0	Digit place holder. If the value being formatted has a digit in the position where the '0' appears in the format string, then that digit is copied to the output string. Otherwise, a '0' is stored in that position in the output string.
#	Digit placeholder. If the value being formatted has a digit in the position where the '#' appears in the format string, then that digit is copied to the output string. Otherwise, nothing is stored in that position in the output string.
.	Decimal point. The first dot character ('.') in the format string determines the location of the decimal separator in the formatted value; any additional dot characters are ignored. The actual character used as the decimal separator in the output string is determined by the DecimalSeparator global variable, whose default value is specified in the Number Format (see page 22110) of the International section in the Windows Control Panel.
,	Thousand separator. If the format string contains one or more comma characters (','), the output will have thousand separators inserted between each group of three digits to the left of the decimal point. The placement and number of comma characters in the format string does not affect the output, except to indicate that thousand separators are wanted. The actual character used as the thousand separator in the output is determined by the ThousandSeparator global variable, whose default value is specified in the Number Format (see page 22110) of the International section in the Windows Control Panel.
E+	Scientific notation. If any of the strings 'E+', 'E-', 'e+', or 'e-' are contained in the format string, the number is formatted using scientific notation. A group of up to four '0' characters can immediately follow the 'E+', 'E-', 'e+', or 'e-' to determine the minimum number of digits in the exponent. The 'E+' and 'e+' formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The 'E-' and 'e-' formats output a sign character only for negative exponents.
'xx'/"xx"	Characters enclosed in single or double quotes are output as-is, and do not affect formatting.
;	Separates sections for positive, negative, and zero numbers in the format string.

The locations of the leftmost '0' before the decimal point in the format string and the rightmost '0' after the decimal point in the format string determine the range of digits that are always present in the output string.

The number being formatted is always rounded to as many decimal places as there are digit placeholders ('0' or '#') to the right of the decimal point. If the format string contains no decimal point, the value being formatted is rounded to the nearest whole number.

If the number being formatted has more digits to the left of the decimal separator than there are digit placeholders to the left of the '.' character in the format string, the extra digits are output before the first digit placeholder.

To allow different formats for positive, negative, and zero values, the format string can contain from one to three sections separated by semicolons.

One section: The format string applies to all values.

Two sections: The first section applies to positive values and zeros, and the second section applies to negative values.

Three sections: The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If no formatting is specified for positive values, positive values are formatted using general floating-point formatting with 15 significant digits, corresponding to a call to FloatToStrF (see page 22109) with the sffGeneral format. General floating-point formatting is also used if the value has more than 18 digits to the left of the decimal point and the format string does not specify scientific notation.

The following table shows some sample formats and the results produced when the formats are applied to different values:

Format (see page 22110) string-1234-12340.50

Format (see page 22110) string-	1234	-1234	0.5	0
1234	-1234	0.5	0	
0	1234	-1234	1	
0.00	1234.00	-1234.00	0.50	
#	1234	-1234	.5	
#,##0.00	1,234.00	-1,234.00	0.50	
#,##0.00;(#,##0.00)	1,234.00	(1,234.00)	0.50	
#,##0.00;;Zero	1,234.00	-1,234.00	0.50	
0.000E+00	1.234E+03	-1.234E+03	5.000E-01	
#.###E-0	1.234E3	-1.234E3	5E-1	

158.1.1.6.18 GetRec Method

This is the overview for the GetRec method overload.

Overload List

	Name	Description
	AnsiString::GetRec () (see page 22112)	This is GetRec, a member of class AnsiString.
	AnsiString::GetRec () (see page 22112)	This is GetRec, a member of class AnsiString.

158.1.1.6.18.1 AnsiString::GetRec Method ()

Pascal

```
function GetRec: StrRec &;
```

C++

```
StrRec & GetRec();
```

Description

This is GetRec, a member of class AnsiString.

158.1.1.6.18.2 AnsiString::GetRec Method ()

Pascal

```
function GetRec: const StrRec &;
```

C++

```
const StrRec & GetRec() const;
```

Description

This is GetRec, a member of class AnsiString.

158.1.1.6.19 AnsiString::Insert Method

System::AnsiString::Inserts a specified string into the System::AnsiString::AnsiString (see page 22097).

Pascal

```
function Insert(const str: AnsiString&; index: Integer): AnsiString&;
```

C++

```
__fastcall AnsiString& Insert(const AnsiString& str, int index);
```

Description

System::AnsiString::Insert inserts the string str into this System::AnsiString::AnsiString (see page 22097) beginning at the position index, where 1 is the first position in the string. It returns the resulting modified string (*this).

If index is less than 1, it is mapped to a 1. If it is past the end of the string, it is set to Length (see page 22115)(), turning the operation into an append.

If the str parameter is an empty string, System::AnsiString::Insert does nothing.

System::AnsiString::Insert throws an EOutOfMemory exception if the method is unable to allocate enough memory to accommodate the new, longer, string.

Note: The index parameter specifies the character, not the byte, when working with a multi-byte character system.

158.1.1.6.20 AnsiString::IntToHex Method

Converts a number into a string containing the number's hexadecimal (base 16) representation.

Pascal

```
function IntToHex(value: Integer; digits: Integer): String; static;
```

C++

```
static __fastcall AnsiString IntToHex(int value, int digits);
```

Description

Value is the number to convert. Digits indicates the minimum number of hexadecimal digits.

158.1.1.6.21 AnsiString::IsDelimiter Method

Returns true if the specified character matches any character in the delimiters string.

Pascal

```
function IsDelimiter(const delimiters: AnsiString&; index: Integer): bool;
```

C++

```
__fastcall bool IsDelimiter(const AnsiString& delimiters, int index) const;
```

Description

System::AnsiString::IsDelimiter returns true if the byte indicated by index (counting from 0) is a single-byte character contained in delimiters. If the byte is part of a multi-byte character or not in delimiters, System::AnsiString::IsDelimiter returns false.

158.1.1.6.22 AnsiString::IsEmpty Method

Returns true if the System::AnsiString::AnsiString (see page 22097) is empty.

Pascal

```
function IsEmpty: bool;
```

C++

```
__fastcall bool IsEmpty() const;
```

Description

System::AnsiString::IsEmpty returns true if this System::AnsiString::AnsiString (see page 22097) is empty, false otherwise.

158.1.1.6.23 AnsiString::IsLeadByte Method

Returns true if the specified byte is a lead byte.

Pascal

```
function IsLeadByte(index: Integer): bool;
```

C++

```
__fastcall bool IsLeadByte(int index) const;
```

Description

System::AnsiString::IsLeadByte returns true if the byte indicated by index (counting from 0) is the lead byte in a multi-byte character. If the byte is a trailing byte, or not part of a multi-byte character, System::AnsiString::IsLeadByte returns false.

158.1.1.6.24 AnsiString::IsPathDelimiter Method

Returns true if the specified character is '\'.

Pascal

```
function IsPathDelimiter(index: Integer): bool;
```

C++

```
__fastcall bool IsPathDelimiter(int index) const;
```

Description

System::AnsiString::IsPathDelimiter returns true if the byte indicated by index (counting from 0) is a single-byte character containing the path delimiter ('\' on Windows, '/' on Linux).

See Also

System::AnsiString::ByteType (see page 22105)

158.1.1.6.25 AnsiString::IsTrailByte Method

Returns true if the specified byte is of type mbTrailByte.

Pascal

```
function IsTrailByte(index: Integer): bool;
```

C++

```
__fastcall bool IsTrailByte(int index) const;
```

Description

System::AnsiString::IsTrailByte returns true if the byte indicated by index (counting from 0) is the trailing byte in a multi-byte character. If the byte is a lead byte, or not part of a multi-byte character, System::AnsiString::IsTrailByte returns false.

158.1.1.6.26 AnsiString::LastDelimiter Method

Returns the index of the rightmost character that matches any character in the delimiters string.

Pascal

```
function LastDelimiter(const delimiters: AnsiString&): Integer;
```

C++

```
__fastcall int LastDelimiter(const AnsiString& delimiters) const;
```

Description

System::AnsiString::LastDelimiter returns the byte index in this System::AnsiString::AnsiString (see page 22097) of the rightmost whole character that matches any character in delimiters except NULL The System::AnsiString::AnsiString (see page 22097) may contain multibyte characters; delimiters must contain only single byte non-NULL characters.

See Also

System::AnsiString::ByteType (see page 22105)

158.1.1.6.27 AnsiString::Length Method

Returns the length of the System::AnsiString::AnsiString (see page 22097).

Pascal

```
function Length: Integer;
```

C++

```
__fastcall int Length() const;
```

Description

System::AnsiString::Length returns the number of bytes in this System::AnsiString::AnsiString (see page 22097).

158.1.1.6.28 LoadStr Method

This is the overview for the LoadStr method overload.

Overload List

	Name	Description
	AnsiString::LoadStr (HINSTANCE, int) (see page 22115)	Loads a string from the compiled resources of a specified module.
	AnsiString::LoadStr (int) (see page 22115)	Loads a string from the compiled resources of a specified module.

158.1.1.6.28.1 AnsiString::LoadStr Method (HINSTANCE, int)

Loads a string from the compiled resources of a specified module.

Pascal

```
function LoadStr(ident: Integer): String; static;
function LoadStr(hInstance: HINSTANCE; ident: Integer): String; static;
```

C++

```
static __fastcall AnsiString LoadStr(int ident);
```

```
static __fastcall AnsiString LoadStr(HINSTANCE hInstance, int ident);
```

Description

System::AnsiString::LoadStr loads a specified string resource.

ident is the resource identifier of the string.

hInstance is the instance handle of the module that contains the string resource. If hInstance is not given, System::AnsiString::LoadStr loads the string from the current module (executable or shared library)

If the string resource does not exist, System::AnsiString::LoadStr returns an empty string.

Moving display strings into string resources makes an application easier to localize without rewriting the program.

See Also

System::AnsiString::LoadString (see page 22116)

System::AnsiString::FmtLoadStr (see page 22110)

158.1.1.6.29 AnsiString::LoadString Method

Loads a string from the compiled resources of a specified module.

Pascal

```
function LoadString(hInstance: HINSTANCE; ident: Integer): AnsiString&;
```

C++

```
__fastcall AnsiString& LoadString(HINSTANCE hInstance, int ident);
```

Description

System::AnsiString::LoadString loads a specified string resource.

ident is the resource identifier of the string.

hInstance is the instance handle of the module that contains the string resource.

If the string resource does not exist, System::AnsiString::LoadString returns an empty string.

Moving display strings into string resources makes an application easier to localize without rewriting the program.

See Also

System::AnsiString::LoadStr (see page 22115)

System::AnsiString::FmtLoadStr (see page 22110)

158.1.1.6.30 AnsiString::LowerCase Method

Returns a lower-case version of System::AnsiString::AnsiString (see page 22097).

Pascal

```
function LowerCase: String;
```

C++

```
__fastcall AnsiString LowerCase() const;
```

Description

System::AnsiString::LowerCase returns a new System::AnsiString::AnsiString (see page 22097) that contains all characters in this string, converted to lower case. The conversion uses the current locale . The original System::AnsiString::AnsiString (see

page 22097) remains unchanged.

158.1.1.6.31 AnsiString::Pos Method

Returns the index at which a specified substring begins.

Pascal

```
function Pos(const subStr: AnsiString&): Integer;
```

C++

```
__fastcall int Pos(const AnsiString& subStr) const;
```

Description

System::AnsiString::Pos returns the index in the System::AnsiString::AnsiString (see page 22097) at which the substring subStr begins, where 1 is the index of the first character, 2 is the index of the second character, and so on. If the substring is not contained in the System::AnsiString::AnsiString (see page 22097), System::AnsiString::Pos returns 0.

158.1.1.6.32 AnsiString::printf Method

Sets the value of the System::AnsiString::AnsiString (see page 22097) given a format string and its arguments.

Pascal

```
function printf(const format: char*; : ...): Integer; cdecl;
```

C++

```
__cdecl int printf(const char* format, ...);
```

Description

Use System::AnsiString::printf to set the value of the System::AnsiString::AnsiString (see page 22097) given a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter. This method returns the length of the final formatted string.

See Also

System::AnsiString::Format (see page 22110)

System::AnsiString::sprintf (see page 22118)

System::AnsiString::cat_printf (see page 22106)

System::AnsiString::vprintf (see page 22121)

158.1.1.6.33 AnsiString::SetLength Method

Changes the length of the System::AnsiString::AnsiString (see page 22097).

Pascal

```
function SetLength(newLength: Integer): AnsiString&;
```

C++

```
__fastcall AnsiString& SetLength(int newLength);
```

Description

System::AnsiString::SetLength changes the string's length to newLength, truncating if necessary.

System::AnsiString::SetLength reallocates the underlying array of characters that stores the value of the

System::AnsiString::AnsiString (see page 22097). Existing characters in the string are preserved, but the content of newly allocated space is undefined. If there is not enough memory available to reallocate the System::AnsiString::AnsiString (see page 22097), an EOutOfMemory exception is thrown. Following a call to System::AnsiString::SetLength, the System::AnsiString::AnsiString (see page 22097) is guaranteed not to be a shared reference.

System::AnsiString::SetLength returns the modified System::AnsiString::AnsiString (see page 22097) object (*this)

158.1.1.6.34 AnsiString::sprintf Method

Sets the value of the System::AnsiString::AnsiString (see page 22097) given a format string and its arguments.

Pascal

```
function sprintf(const format: char*; : ...): AnsiString&; cdecl;
```

C++

```
__cdecl AnsiString& sprintf(const char* format, ...);
```

Description

Use System::AnsiString::sprintf to set to the value of the System::AnsiString::AnsiString (see page 22097) given a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter.

This method returns a reference to the System::AnsiString::AnsiString (see page 22097) (*this) with the resulting value.

See Also

System::AnsiString::Format (see page 22110)

System::AnsiString::cat_sprintf (see page 22106)

System::AnsiString::printf (see page 22117)

System::AnsiString::vprintf (see page 22121)

158.1.1.6.35 AnsiString::StringOfChar Method

Returns a string with the specified number of characters.

Pascal

```
function StringOfChar(ch: char; count: Integer): String; static;
```

C++

```
static __fastcall AnsiString StringOfChar(char ch, int count);
```

Description

System::AnsiString::StringOfChar returns a string containing count characters with the character value given by ch. For example,

System::AnsiString::AnsiString (see page 22097) s = System::AnsiString::AnsiString::System::AnsiString::StringOfChar('A', 10);

sets s to the string "AAAAAAAAAA".

158.1.1.6.36 AnsiString::SubString Method

Returns a specified substring of the System::AnsiString::AnsiString (see page 22097).

Pascal

```
function SubString(index: Integer; count: Integer): String;
```

C++

```
__fastcall AnsiString SubString(int index, int count) const;
```

Description

System::AnsiString::SubString returns a new System::AnsiString::AnsiString (see page 22097) that is a substring of this System::AnsiString::AnsiString (see page 22097). The substring contains count characters beginning at index.

158.1.1.6.37 AnsiString::ThrowIfOutOfRange Method

Pascal

```
procedure ThrowIfOutOfRange(idx: Integer); cdecl;
```

C++

```
__cdecl void ThrowIfOutOfRange(int idx) const;
```

Description

This is ThrowIfOutOfRange, a member of class AnsiString.

158.1.1.6.38 AnsiString::ToDouble Method

Converts the string to a floating-point value.

Pascal

```
function ToDouble: double;
```

C++

```
__fastcall double ToDouble() const;
```

Description

System::AnsiString::ToDouble converts the string to a floating-point value. The string must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional 'E' or 'e' followed by a signed integer. Leading and trailing blanks in the string are ignored. The DecimalSeparator global variable defines the character that must be used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If the string doesn't contain a valid value, an EConvertError exception is thrown.

158.1.1.6.39 AnsiString::ToInt Method

Converts a string to an integer, throwing an exception if the string is invalid.

Pascal

```
function ToInt: Integer;
```

C++

```
__fastcall int ToInt() const;
```

Description

System::AnsiString::ToInt converts the given string to an integer value and returns the integer. If the string doesn't contain a valid value, an EConvertError exception is thrown.

See Also

System::AnsiString::ToIntDef (see page 22120)

158.1.1.6.40 AnsiString::ToIntDef Method

Converts the string to an integer, returning a default value if the string is invalid.

Pascal

```
function ToIntDef(defaultValue: Integer): Integer;
```

C++

```
__fastcall int ToIntDef(int defaultValue) const;
```

Description

System::AnsiString::ToIntDef converts the given string to an integer value and returns the integer. If the string doesn't contain a valid value, the value given by defaultValue is returned.

See Also

System::AnsiString::ToInt (see page 22119)

158.1.1.6.41 AnsiString::Trim Method

Returns a new System::AnsiString::AnsiString (see page 22097) without leading spaces, trailing spaces, or control characters.

Pascal

```
function Trim: String;
```

C++

```
__fastcall AnsiString Trim() const;
```

Description

System::AnsiString::Trim returns a new System::AnsiString::AnsiString (see page 22097) removing leading and trailing spaces and control characters. Use System::AnsiString::Trim to remove blank space before and after the first printing character.

158.1.1.6.42 AnsiString::TrimLeft Method

Returns a new System::AnsiString::AnsiString (see page 22097) without leading spaces or control characters.

Pascal

```
function TrimLeft: String;
```

C++

```
__fastcall AnsiString TrimLeft() const;
```

Description

System::AnsiString::TrimLeft returns a new System::AnsiString::AnsiString (see page 22097) removing leading spaces and control characters. Use System::AnsiString::TrimLeft to remove blank space before the first printing character.

158.1.1.6.43 AnsiString::TrimRight Method

Returns a new System::AnsiString::AnsiString (see page 22097) without trailing spaces or control characters.

Pascal

```
function TrimRight: String;
```

C++

```
__fastcall AnsiString TrimRight() const;
```

Description

System::AnsiString::TrimRight returns a new System::AnsiString::AnsiString (see page 22097) removing trailing spaces and control characters. Use the System::AnsiString::TrimRight function to remove blank space after the first printing character.

158.1.1.6.44 AnsiString::Unique Method

Makes the System::AnsiString::AnsiString (see page 22097) unique.

Pascal

```
function Unique: AnsiString&;
```

C++

```
__fastcall AnsiString& Unique();
```

Description

Use this function to make a string unique (refcnt == 1). System::AnsiString::Unique sets the reference count on the string for the System::AnsiString::AnsiString (see page 22097) object to 1 so that it is the only object referencing that string.

158.1.1.6.45 AnsiString::UpperCase Method

Returns an upper-case version of this System::AnsiString::AnsiString (see page 22097).

Pascal

```
function UpperCase: String;
```

C++

```
__fastcall AnsiString UpperCase() const;
```

Description

System::AnsiString::UpperCase returns a new System::AnsiString::AnsiString (see page 22097) that contains all characters in this string, converted to upper case. The conversion uses the current locale . The original System::AnsiString::AnsiString (see page 22097) remains unchanged.

158.1.1.6.46 AnsiString::vprintf Method

Sets the value of the System::AnsiString::AnsiString (see page 22097) given a format string and its arguments.

Pascal

```
function vprintf(const format: char*; : va_list): Integer; cdecl;
```

C++

```
__cdecl int vprintf(const char* format, va_list);
```

Description

Use System::AnsiString::vprintf to set to the value of the System::AnsiString::AnsiString (see page 22097) given a standard C++ format specifier. Pass the values to any arguments in the format specifier as a variable argument list that was created using va_start.

This method returns the length of the final formatted string.

See Also

System::AnsiString::Format (see page 22110)

System::AnsiString::sprintf (see page 22118)

System::AnsiString::printf (see page 22117)

System::AnsiString::cat_vprintf (see page 22107)

arg

start

158.1.1.6.47 AnsiString::WideChar Method

Converts the System::AnsiString::AnsiString (see page 22097) to a wide-character array.

Pascal

```
function WideChar(var dest: wchar_t; destSize: Integer): wchar_t*;
```

C++

```
__fastcall wchar_t* WideChar(wchar_t* dest, int destSize) const;
```

Description

System::AnsiString::WideChar converts the System::AnsiString::AnsiString (see page 22097) to an array of wide characters, and returns the resulting wide-character array.

dest is a buffer the caller allocates to hold the return value. Use the System::AnsiString::WideCharBufSize (see page 22122) method to determine the size needed to hold the entire string value.

destSize is the size, in bytes, of dest.

System::AnsiString::WideChar returns a pointer to the wide-character array that is written to dest.

See Also

System::AnsiString::WideCharBufSize (see page 22122)

158.1.1.6.48 AnsiString::WideCharBufSize Method

Returns the size of the buffer required by the WideChar (see page 22122) function.

Pascal

```
function WideCharBufSize: Integer;
```

C++

```
__fastcall int WideCharBufSize() const;
```

Description

System::AnsiString::WideCharBufSize returns an integer value indicating the size of the buffer required by the WideChar (see page 22122) method in order to convert this string.

See Also

System::AnsiString::WideChar (see page 22122)

158.1.1.7 AnsiString Operators

The operators of the AnsiString class are listed here.

Public Operators

	Name	Description
	!= (see page 22123)	Compares two strings.
	[] (see page 22123)	This is the overview for the [] operator overload.
	+ (see page 22124)	Concatenates two System::AnsiString::AnsiString (see page 22097) values.
	+= (see page 22124)	Append (see page 22415) an System::AnsiString::AnsiString (see page 22097).
	< (see page 22124)	Compares two strings.
	<= (see page 22125)	Compares two strings.
	= (see page 22125)	Assigns the string rhs to this System::AnsiString::AnsiString (see page 22097).
	== (see page 22126)	Compares the System::AnsiString::AnsiString (see page 22097) with a specified string, returning true if they are the same.
	> (see page 22126)	Compares two strings.
	>= (see page 22126)	Compares two strings.

158.1.1.7.1 AnsiString::!= Operator

Compares two strings.

Pascal

```
operator !=(const rhs: AnsiString&);
```

C++

```
__fastcall bool operator !=(const AnsiString& rhs) const;
```

Description

Returns true if the strings are not the same, false otherwise.

Note: This System::AnsiString (see page 22093)::!= performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 22103) or AnsiCompareIC (see page 22104) methods.

See Also

System::AnsiString::AnsiCompare (see page 22103)

System::AnsiString::AnsiCompareIC (see page 22104)

158.1.1.7.2 [] Operator

This is the overview for the [] operator overload.

Overload List

	Name	Description
	AnsiString::[] (int) (see page 22123)	Returns the character at a specified index in the string.
	AnsiString::[] (int) (see page 22123)	Returns the character at a specified index in the string.

158.1.1.7.2.1 AnsiString::[] Operator (int)

Returns the character at a specified index in the string.

Pascal

```
operator [](const idx: Integer);
operator [](const idx: Integer);
```

C++

```
__fastcall char operator [](const int idx) const;
__fastcall char& operator [](const int idx);
```

Description

This System::AnsiString (see page 22093)::[] returns the byte in the string at index value idx. The System::AnsiString (see page 22093)::[] assumes a base index of 1.

Note: If the string contains multi-byte characters, the System::AnsiString (see page 22093)::[] may return a byte value that represents the wrong character and/or a partial character. For character-level access, use the SubString (see page 22118) method.

See Also

System::AnsiString::substring

158.1.1.7.3 AnsiString::+ Operator

Concatenates two System::AnsiString::AnsiString (see page 22097) values.

Pascal

```
operator +(const rhs: AnsiString&);
```

C++

```
__fastcall AnsiString operator +(const AnsiString& rhs) const;
```

Description

The friend function returns an System::AnsiString::AnsiString (see page 22097) that is the concatenation of strings lhs and rhs.

The method returns an System::AnsiString::AnsiString (see page 22097) that is the concatenation of this System::AnsiString::AnsiString (see page 22097) and rhs.

158.1.1.7.4 AnsiString::+= Operator

Append (see page 22415) an System::AnsiString::AnsiString (see page 22097).

Pascal

```
operator +=(const rhs: AnsiString&);
```

C++

```
__fastcall AnsiString& operator +=(const AnsiString& rhs);
```

Description

The System::AnsiString (see page 22093)::+= concatenates this System::AnsiString::AnsiString (see page 22097) and rhs, assigning the result to this System::AnsiString::AnsiString (see page 22097).

The System::AnsiString (see page 22093)::+=() is not supported on a property with getter/setter functions.

158.1.1.7.5 AnsiString::< Operator

Compares two strings.

Pascal

```
operator <(const rhs: AnsiString&);
```

C++

```
__fastcall bool operator <(const AnsiString& rhs) const;
```

Description

Returns true if this System::AnsiString::AnsiString (see page 22097) is less than rhs, false otherwise.

Note: This System::AnsiString (see page 22093)::< performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 22103) or AnsiCompareIC (see page 22104) methods.

See Also

System::AnsiString::AnsiCompare (see page 22103)

System::AnsiString::AnsiCompareIC (see page 22104)

158.1.1.7.6 AnsiString::<= Operator

Compares two strings.

Pascal

```
operator <=(const rhs: AnsiString&);
```

C++

```
__fastcall bool operator <=(const AnsiString& rhs) const;
```

Description

Returns true if this System::AnsiString::AnsiString (see page 22097) is less than or equal to rhs, false otherwise.

Note: This System::AnsiString (see page 22093)::<= performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 22103) or AnsiCompareIC (see page 22104) methods.

See Also

System::AnsiString::AnsiCompare (see page 22103)

System::AnsiString::AnsiCompareIC (see page 22104)

158.1.1.7.7 AnsiString::= Operator

Assigns the string rhs to this System::AnsiString::AnsiString (see page 22097).

Pascal

```
operator =(const rhs: AnsiString&);
```

C++

```
__fastcall AnsiString& operator =(const AnsiString& rhs);
```

Description

Use this System::AnsiString (see page 22093)::= to assign the value of the System::AnsiString::AnsiString (see page 22097) from another System::AnsiString::AnsiString (see page 22097).

158.1.1.7.8 AnsiString::== Operator

Compares the System::AnsiString::AnsiString (see page 22097) with a specified string, returning true if they are the same.

Pascal

```
operator ==(const rhs: AnsiString&);
```

C++

```
__fastcall bool operator ==(const AnsiString& rhs) const;
```

Description

Returns true if this System::AnsiString::AnsiString (see page 22097) is the same as rhs, false otherwise.

Note: This System::AnsiString (see page 22093)::== performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 22103) or AnsiCompareIC (see page 22104) methods.

See Also

System::AnsiString::AnsiCompare (see page 22103)

System::AnsiString::AnsiCompareIC (see page 22104)

158.1.1.7.9 AnsiString::> Operator

Compares two strings.

Pascal

```
operator >(const rhs: AnsiString&);
```

C++

```
__fastcall bool operator >(const AnsiString& rhs) const;
```

Description

Returns true if this System::AnsiString::AnsiString (see page 22097) is greater than rhs, false otherwise.

Note: This System::AnsiString (see page 22093)::> performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 22103) or AnsiCompareIC (see page 22104) methods.

See Also

System::AnsiString::AnsiCompare (see page 22103)

System::AnsiString::AnsiCompareIC (see page 22104)

158.1.1.7.10 AnsiString::>= Operator

Compares two strings.

Pascal

```
operator >=(const rhs: AnsiString&);
```

C++

```
__fastcall bool operator >=(const AnsiString& rhs) const;
```

Description

Returns true if this System::AnsiString::AnsiString (see page 22097) is greater than or equal to rhs, false otherwise.

Note: This System::AnsiString (see page 22093)::>= performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 22103) or AnsiCompareIC (see page 22104) methods.

See Also

System::AnsiString::AnsiCompare (see page 22103)

System::AnsiString::AnsiCompareIC (see page 22104)

158.1.1.8 AnsiString Friends

The friends of the AnsiString class are listed here.

Public Friends

Name	Description
AnsiString __fastcall operator +(const char*, const AnsiString& rhs) (see page 22127)	This is friend AnsiString __fastcall operator +(const char*, const AnsiString& rhs).

158.1.1.8.1 friend AnsiString __fastcall operator +(const char*, const AnsiString& rhs) Friend

Pascal

```
AnsiString __fastcall operator +(const char*, const AnsiString& rhs)
```

C++

```
friend AnsiString __fastcall operator +(const char*, const AnsiString& rhs);
```

Description

This is friend AnsiString __fastcall operator +(const char*, const AnsiString& rhs).

158.1.2 AutoCmd Class

System::AutoCmd represents a command that can be executed by an Automation interface wrapped by a Variant (see page 22307) class.

Class Hierarchy

System::AutoCmd

Pascal

```
AutoCmd = class;
```

C++

```
class AutoCmd;
```

File

sysvari.h

Description

The System::AutoCmd class is a friend class to Variant (see page 22307). It represents a single method on an Automation interface that is assigned as the value of a Variant (see page 22307).

Do not use instances of System::AutoCmd, which includes pure virtual methods. Instead, use the System::AutoCmd descendant

that represents the type of the desired method. These descendants are

PropertyGet (see page 22233), for methods that return a property value.

PropertySet (see page 22236), for methods that set a property value.

Procedure (see page 22230), for methods that do not return a value.

Function (see page 22184), for methods that do return a value.

Each System::AutoCmd instance is assigned a name, which is the same as the name of the corresponding method on the interface. In addition, it can be assigned parameters that correspond to the parameters of the method. These parameters can be stored by position or by name.

Once you have assigned parameter values to the System::AutoCmd object, you can execute it using the Variant::Exec method.

See Also

System::Variant (see page 22307)

PropertyGet

PropertySet

System::Function (see page 22184)

System::Procedure (see page 22230)

158.1.2.1 AutoCmd Members

The following tables list the members exposed by AutoCmd.

Protected Data Members

	Name	Description
	Name (see page 22130)	This is Name, a member of class AutoCmd.
	NamedParmNames (see page 22130)	This is NamedParmNames, a member of class AutoCmd.
	NamedParms (see page 22130)	This is NamedParms, a member of class AutoCmd.
	NamedParmTypes (see page 22131)	This is NamedParmTypes, a member of class AutoCmd.
	Parms (see page 22131)	This is Parms, a member of class AutoCmd.
	ParmTypes (see page 22131)	This is ParmTypes, a member of class AutoCmd.

Protected Methods

	Name	Description
	AddElement (see page 22132)	This is AddElement, a member of class AutoCmd.

Public Methods

	Name	Description
	~AutoCmd (see page 22129)	Frees an System::AutoCmd::AutoCmd (see page 22129) object.
	AutoCmd (see page 22129)	This is the overview for the AutoCmd constructor overload.
	Clear (see page 22132)	System::AutoCmd::Clears any assigned parameters and blanks the name of the associated method.
	ClearArgs (see page 22132)	Clears any assigned parameters.
	ClearName (see page 22133)	Blanks the name of the associated method.
	GetArgCount (see page 22133)	Returns the number of positional arguments that have been assigned.
	GetArgType (see page 22134)	Returns the type of a parameter value.
A	GetCallType (see page 22134)	Indicates the type of the interface method represented by the System::AutoCmd::AutoCmd (see page 22129) object.
	GetName (see page 22134)	Returns the name of the interface method this System::AutoCmd::AutoCmd (see page 22129) object represents.

	GetNamedArgCount (see page 22135)	Returns the number of named arguments that have been assigned.
	GetNamedArgType (see page 22135)	Returns the type of a named parameter value.
	GetNamedParm (see page 22136)	Returns the value of a named parameter.
	GetNamedParmName (see page 22136)	Returns the named of a named parameter value.
	GetParm (see page 22137)	Returns the value of a positional parameter.
	RequestResult (see page 22137)	Indicates whether the method represented by the System::AutoCmd::AutoCmd (see page 22129) object has a return value.
	SetName (see page 22138)	Sets the name of the method the System::AutoCmd::AutoCmd (see page 22129) object represents.

Public Operators

	Name	Description
	<< (see page 22138)	This is the overview for the << operator overload.

158.1.2.2 AutoCmd::~AutoCmd Destructor

Frees an System::AutoCmd::AutoCmd (see page 22129) object.

Pascal

```
destructor Destroy;
```

C++

```
~AutoCmd();
```

Description

System::AutoCmd (see page 22127)::~AutoCmd (see page 22127) is the destructor for System::AutoCmd::AutoCmd (see page 22129).

158.1.2.3 AutoCmd Constructor

This is the overview for the AutoCmd constructor overload.

Overload List

	Name	Description
	AutoCmd::AutoCmd (AutoCmd&) (see page 22129)	System::AutoCmd::Creates an instance of System::AutoCmd::AutoCmd (see page 22127) for a specified method.
	AutoCmd::AutoCmd (String&) (see page 22129)	System::AutoCmd::Creates an instance of System::AutoCmd::AutoCmd (see page 22127) for a specified method.

158.1.2.3.1 AutoCmd::AutoCmd Constructor (AutoCmd&)

System::AutoCmd::Creates an instance of System::AutoCmd::AutoCmd (see page 22127) for a specified method.

Pascal

```
constructor Create(const name: String&);
constructor Create(const src: AutoCmd&);
```

C++

```
AutoCmd(const String& name);
AutoCmd(const AutoCmd& src);
```

Description

System::AutoCmd::AutoCmd (see page 22127) has two constructors: a version that takes the name of the method it represents, and a copy constructor. When creating a new System::AutoCmd::AutoCmd (see page 22127) object for a method, the name parameter must be the same as the name of the method as it appears in the interface declaration.

158.1.2.4 AutoCmd Data Members

The data members of the AutoCmd class are listed here.

Protected Data Members

	Name	Description
	Name (see page 22130)	This is Name, a member of class AutoCmd.
	NamedParmNames (see page 22130)	This is NamedParmNames, a member of class AutoCmd.
	NamedParms (see page 22130)	This is NamedParms, a member of class AutoCmd.
	NamedParmTypes (see page 22131)	This is NamedParmTypes, a member of class AutoCmd.
	Parms (see page 22131)	This is Parms, a member of class AutoCmd.
	ParmTypes (see page 22131)	This is ParmTypes, a member of class AutoCmd.

158.1.2.4.1 AutoCmd::Name Data Member

Pascal

```
Name: String;
```

C++

```
String Name;
```

Description

This is Name, a member of class AutoCmd.

158.1.2.4.2 AutoCmd::NamedParmNames Data Member

Pascal

```
NamedParmNames: Variant;
```

C++

```
Variant NamedParmNames;
```

Description

This is NamedParmNames, a member of class AutoCmd.

158.1.2.4.3 AutoCmd::NamedParms Data Member

Pascal

```
NamedParms: Variant;
```

C++

```
Variant NamedParms;
```

Description

This is NamedParms, a member of class AutoCmd.

158.1.2.4.4 AutoCmd::NamedParmTypes Data Member

Pascal

```
NamedParmTypes: Variant;
```

C++

```
Variant NamedParmTypes;
```

Description

This is NamedParmTypes, a member of class AutoCmd.

158.1.2.4.5 AutoCmd::Parms Data Member

Pascal

```
Parms: Variant;
```

C++

```
Variant Parms;
```

Description

This is Parms, a member of class AutoCmd.

158.1.2.4.6 AutoCmd::ParmTypes Data Member

Pascal

```
ParmTypes: Variant;
```

C++

```
Variant ParmTypes;
```

Description

This is ParmTypes, a member of class AutoCmd.

158.1.2.5 AutoCmd Methods

The methods of the AutoCmd class are listed here.

Protected Methods

	Name	Description
	AddElement (see page 22132)	This is AddElement, a member of class AutoCmd.

Public Methods

	Name	Description
	Clear (see page 22132)	System::AutoCmd::Clears any assigned parameters and blanks the name of the associated method.
	ClearArgs (see page 22132)	Clears any assigned parameters.
	ClearName (see page 22133)	Blanks the name of the associated method.
	GetArgCount (see page 22133)	Returns the number of positional arguments that have been assigned.
	GetArgType (see page 22134)	Returns the type of a parameter value.
	GetCallType (see page 22134)	Indicates the type of the interface method represented by the System::AutoCmd::AutoCmd (see page 22129) object.

	GetName (see page 22134)	Returns the name of the interface method this System::AutoCmd::AutoCmd (see page 22129) object represents.
	GetNamedArgCount (see page 22135)	Returns the number of named arguments that have been assigned.
	GetNamedArgType (see page 22135)	Returns the type of a named parameter value.
	GetNamedParm (see page 22136)	Returns the value of a named parameter.
	GetNamedParmName (see page 22136)	Returns the named of a named parameter value.
	GetParm (see page 22137)	Returns the value of a positional parameter.
	RequestResult (see page 22137)	Indicates whether the method represented by the System::AutoCmd::AutoCmd (see page 22129) object has a return value.
	SetName (see page 22138)	Sets the name of the method the System::AutoCmd::AutoCmd (see page 22129) object represents.

158.1.2.5.1 AutoCmd::AddElement Method

Pascal

```
procedure AddElement;
```

C++

```
void AddElement();
```

Description

This is AddElement, a member of class AutoCmd.

158.1.2.5.2 AutoCmd::Clear Method

System::AutoCmd::Clears any assigned parameters and blanks the name of the associated method.

Pascal

```
function Clear: AutoCmd&;
```

C++

```
AutoCmd& Clear();
```

Description

Call System::AutoCmd::Clear to remove the System::AutoCmd::AutoCmd (see page 22129) object's association with an Automation interface method. System::AutoCmd::Clear replaces the method name with an empty string, and frees all arguments that have been assigned. After a call to System::AutoCmd::Clear, the System::AutoCmd::AutoCmd (see page 22129) object can't be executed until a new method name is assigned (using SetName (see page 22138)) and, if necessary, new parameters assigned (using the << operator).

See Also

System::AutoCmd::ClearName (see page 22133)

System::AutoCmd::ClearArgs (see page 22132)

158.1.2.5.3 AutoCmd::ClearArgs Method

Clears any assigned parameters.

Pascal

```
function ClearArgs: AutoCmd&;
```

C++

```
AutoCmd& ClearArgs();
```

Description

Call System::AutoCmd::ClearArgs to remove all assigned parameter values (both named parameter values and values that were supplied by position). Once the existing values have been removed, new parameter values can be assigned using the << operator.

See Also

System::AutoCmd::ClearName (see page 22133)

System::AutoCmd::Clear (see page 22132)

158.1.2.5.4 AutoCmd::ClearName Method

Blanks the name of the associated method.

Pascal

```
function ClearName: AutoCmd&;
```

C++

```
AutoCmd& ClearName();
```

Description

Call System::AutoCmd::ClearName to disassociate the System::AutoCmd::AutoCmd (see page 22129) object from the Automation interface method it represents. System::AutoCmd::ClearName replaces the method name with an empty string, but does not free any parameter values that have been assigned. After a call to System::AutoCmd::ClearName, the System::AutoCmd::AutoCmd (see page 22129) object can't be executed until a new method name is assigned using the SetName (see page 22138) method.

See Also

System::AutoCmd::Clear (see page 22132)

System::AutoCmd::ClearArgs (see page 22132)

158.1.2.5.5 AutoCmd::GetArgCount Method

Returns the number of positional arguments that have been assigned.

Pascal

```
function GetArgCount: Byte;
```

C++

```
Byte GetArgCount() const;
```

Description

Call System::AutoCmd::GetArgCount to determine the number of parameter values that have been assigned using the << operator. The values themselves can be retrieved using the GetParm (see page 22137) method.

See Also

System::AutoCmd::operator_streaming

System::AutoCmd::GetNamedArgCount (see page 22135)

System::AutoCmd::GetParm (see page 22137)

158.1.2.5.6 AutoCmd::GetArgType Method

Returns the type of a parameter value.

Pascal

```
function GetArgType(const idx: Byte): Integer;
```

C++

```
int GetArgType(const Byte idx) const;
```

Description

Call System::AutoCmd::GetArgType to determine the type of an argument that was assigned using the << operator.

The idx parameter is the index of the value in the System::AutoCmd::AutoCmd (see page 22129) object's parameter list, where 0 is the index of the first argument. Use the GetArgCount (see page 22133) method to obtain an upper limit of possible values.

See Also

System::AutoCmd::operator_streaming

System::AutoCmd::GetNamedArgType (see page 22135)

System::AutoCmd::GetParm (see page 22137)

System::AutoCmd::GetArgCount (see page 22133)

158.1.2.5.7 AutoCmd::GetCallType Method

Indicates the type of the interface method represented by the System::AutoCmd::AutoCmd (see page 22129) object.

Pascal

```
function GetCallType: Byte; virtual; abstract;
```

C++

```
virtual Byte GetCallType() const = 0;
```

Description

Call System::AutoCmd::GetCallType to determine the type of flag needed to call the method represented by System::AutoCmd::AutoCmd (see page 22129) using the IDispatch::Invoke method of the Automation interface. System::AutoCmd::GetCallType returns DISPATCH_PROPERTYGET if the System::AutoCmd::AutoCmd (see page 22129) object represents a property getter, DISPATCH_PROPERTYSET if it represents a property setter, and DISPATCH_METHOD if it represents any other method type.

System::AutoCmd::GetCallType is declared as a pure virtual method. Descendant classes, which represent specific types of interface methods, override this method to return the appropriate value.

158.1.2.5.8 AutoCmd::GetName Method

Returns the name of the interface method this System::AutoCmd::AutoCmd (see page 22129) object represents.

Pascal

```
function GetName: String;
```

C++

```
String GetName() const;
```

Description

Call System::AutoCmd::GetName to determine the method on an Automation interface to invoke when executing the System::AutoCmd::AutoCmd (see page 22129) object. This is the name of the method, rather than its dispatch id.

Typically, the System::AutoCmd::AutoCmd (see page 22129) constructor specifies the name of the method, although this can be changed using the SetName (see page 22138) method.

See Also

System::AutoCmd::SetName (see page 22138)

System::AutoCmd::AutoCmd (see page 22127)

158.1.2.5.9 AutoCmd::GetNamedArgCount Method

Returns the number of named arguments that have been assigned.

Pascal

```
function GetNamedArgCount: Byte;
```

C++

```
Byte GetNamedArgCount() const;
```

Description

Call System::AutoCmd::GetNamedArgCount to determine the number of named parameter values that have been assigned using the << operator. The values themselves can be retrieved using the GetNamedParm (see page 22136) method, and the corresponding names using the GetNamedParmName (see page 22136) method.

See Also

System::AutoCmd::operator_streaming

System::AutoCmd::GetArgCount (see page 22133)

System::AutoCmd::GetNamedParm (see page 22136)

System::AutoCmd::GetNamedParmName (see page 22136)

System::AutoCmd::GetNamedArgType (see page 22135)

158.1.2.5.10 AutoCmd::GetNamedArgType Method

Returns the type of a named parameter value.

Pascal

```
function GetNamedArgType(const idx: Byte): Integer;
```

C++

```
int GetNamedArgType(const Byte idx) const;
```

Description

Call System::AutoCmd::GetNamedArgType to determine the type of an argument that was assigned using the << operator.

The idx parameter is the index of the value in the System::AutoCmd::AutoCmd (see page 22129) object's named parameter list, where 0 is the index of the first argument. Use the GetNamedArgCount (see page 22135) method to obtain an upper limit of possible values.

See Also

System::AutoCmd::operator_streaming

System::AutoCmd::GetArgType (see page 22134)

System::AutoCmd::GetNamedParm (see page 22136)

System::AutoCmd::GetnamedArgCount

System::AutoCmd::GetNamedParmName (see page 22136)

158.1.2.5.11 AutoCmd::GetNamedParm Method

Returns the value of a named parameter.

Pascal

```
function GetNamedParm(const idx: Byte): Variant&;
```

C++

```
Variant& GetNamedParm(const Byte idx);
```

Description

Call System::AutoCmd::GetNamedParm to determine the value of named parameter that was assigned using the << operator. The value is returned as a Variant (see page 22307) object. Its underlying type can be obtained using the GetNamedArgType (see page 22135) method.

The idx parameter is the index of the value in the System::AutoCmd::AutoCmd (see page 22129) object's named parameter list, where 0 is the index of the first argument. Use the GetNamedArgCount (see page 22135) method to obtain an upper limit of possible values.

See Also

System::AutoCmd::operator_streaming

System::AutoCmd::GetNamedArgCount (see page 22135)

System::AutoCmd::GetNamedArgType (see page 22135)

System::AutoCmd::GetNamedParmName (see page 22136)

System::Variant (see page 22307)

158.1.2.5.12 AutoCmd::GetNamedParmName Method

Returns the named of a named parameter value.

Pascal

```
function GetNamedParmName(const idx: Byte): String;
```

C++

```
String GetNamedParmName(const Byte idx) const;
```

Description

Call System::AutoCmd::GetNamedParmName to determine the name of an argument that was assigned using the << operator.

The idx parameter is the index of the value in the System::AutoCmd::AutoCmd (see page 22129) object's named parameter list, where 0 is the index of the first argument. Use the GetNamedArgCount (see page 22135) method to obtain an upper limit of possible values.

See Also

System::AutoCmd::operator_streaming

System::AutoCmd::GetNamedArgType (see page 22135)

System::AutoCmd::GetNamedParm (see page 22136)

System::AutoCmd::GetNamedArgCount (see page 22135)

158.1.2.5.13 AutoCmd::GetParm Method

Returns the value of a positional parameter.

Pascal

```
function GetParm(const idx: Byte): Variant&;
```

C++

```
Variant& GetParm(const Byte idx);
```

Description

Call System::AutoCmd::GetParm to determine the value of positional parameter that was assigned using the << operator. The value is returned as a Variant (see page 22307) object. Its underlying type can be obtained using the GetArgType (see page 22134) method.

The idx parameter is the index of the value in the System::AutoCmd::AutoCmd (see page 22129) object's parameter list, where 0 is the index of the first argument. Use the GetArgCount (see page 22133) method to obtain an upper limit of possible values.

See Also

System::AutoCmd::operator_streaming

System::AutoCmd::GetArgCount (see page 22133)

System::AutoCmd::GetArgType (see page 22134)

System::Variant (see page 22307)

158.1.2.5.14 AutoCmd::RequestResult Method

Indicates whether the method represented by the System::AutoCmd::AutoCmd (see page 22129) object has a return value.

Pascal

```
function RequestResult: bool; virtual; abstract;
```

C++

```
virtual bool RequestResult() const = 0;
```

Description

Call System::AutoCmd::RequestResult to determine whether there is a return value after executing the System::AutoCmd::AutoCmd (see page 22129) object using the Variant::Exec method.

System::AutoCmd::AutoCmd (see page 22129) declares System::AutoCmd::RequestResult as a pure virtual method. Descendant classes specialized to different call types return a value depending on what type of method they represent.

See Also

System::AutoCmd::GetCallType (see page 22134)

System::Variant::Exec (see page 22320)

158.1.2.5.15 AutoCmd::SetName Method

Sets the name of the method the System::AutoCmd::AutoCmd (see page 22129) object represents.

Pascal

```
function SetName(const name: String&): AutoCmd&;
```

C++

```
AutoCmd& SetName(const String& name);
```

Description

Use System::AutoCmd::SetName to associate the System::AutoCmd::AutoCmd (see page 22129) object with a specific method on an Automation interface. The name parameter is the name of the interface method.

Typically, the System::AutoCmd::AutoCmd (see page 22129) constructor specifies the name of the method. Use System::AutoCmd::SetName to change the method that the object represents.

See Also

System::AutoCmd::ClearName (see page 22133)

System::AutoCmd::GetName (see page 22134)

System::AutoCmd::AutoCmd (see page 22127)

158.1.2.6 AutoCmd Operators

The operators of the AutoCmd class are listed here.

Public Operators

	Name	Description
	<< (see page 22138)	This is the overview for the << operator overload.

158.1.2.6.1 << Operator

This is the overview for the << operator overload.

Overload List

	Name	Description
	AutoCmd::<< (AnsiString&) (see page 22139)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (AnsiString*) (see page 22139)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (bool) (see page 22142)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (Byte) (see page 22139)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (Byte*) (see page 22140)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (Currency) (see page 22140)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (Currency*) (see page 22140)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (double) (see page 22143)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (double*) (see page 22143)	This is <<, a member of class AutoCmd.

	AutoCmd::<< (float) (see page 22143)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (float*) (see page 22143)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (IDispatch*) (see page 22140)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (IInterface*) (see page 22141)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (int) (see page 22143)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (int*) (see page 22144)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (IUnknown*) (see page 22141)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (NamedParm&) (see page 22141)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (short) (see page 22144)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (short*) (see page 22144)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (TDateTime) (see page 22141)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (TDateTime*) (see page 22141)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (Variant&) (see page 22142)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (Variant*) (see page 22142)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (wchar_t*) (see page 22144)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (wchar_t**) (see page 22144)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (WordBool) (see page 22142)	This is <<, a member of class AutoCmd.
	AutoCmd::<< (WordBool*) (see page 22142)	This is <<, a member of class AutoCmd.

158.1.2.6.1.1 AutoCmd::<< Operator (AnsiString&)

Pascal

```
operator <<(const arg: AnsiString&);
```

C++

```
AutoCmd& operator <<(const AnsiString& arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.2 AutoCmd::<< Operator (AnsiString*)

Pascal

```
operator <<(var arg: String);
```

C++

```
AutoCmd& operator <<(AnsiString* arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.3 AutoCmd::<< Operator (Byte)

Pascal

```
operator <<(const arg: Byte);
```

C++

```
AutoCmd& operator <<(const Byte arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.4 AutoCmd::<< Operator (Byte*)

Pascal

```
operator <<(var arg: Byte);
```

C++

```
AutoCmd& operator <<(Byte* arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.5 AutoCmd::<< Operator (Currency)

Pascal

```
operator <<(const arg: Currency);
```

C++

```
AutoCmd& operator <<(const Currency arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.6 AutoCmd::<< Operator (Currency*)

Pascal

```
operator <<(var arg: Currency);
```

C++

```
AutoCmd& operator <<(Currency* arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.7 AutoCmd::<< Operator (IDispatch*)

Pascal

```
operator <<(var arg: IDispatch);
```

C++

```
AutoCmd& operator <<(IDispatch* arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.8 AutoCmd::<< Operator (IInterface*)

Pascal

```
operator <<(var arg: IInterface);
```

C++

```
AutoCmd& operator <<(IInterface* arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.9 AutoCmd::<< Operator (IUnknown*)

Pascal

```
operator <<(var arg: IUnknown);
```

C++

```
AutoCmd& operator <<(IUnknown* arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.10 AutoCmd::<< Operator (NamedParm&)

Pascal

```
operator <<(const np: NamedParm&);
```

C++

```
AutoCmd& operator <<(const NamedParm& np);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.11 AutoCmd::<< Operator (TDateTime)

Pascal

```
operator <<(const arg: TDateTime);
```

C++

```
AutoCmd& operator <<(const TDateTime arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.12 AutoCmd::<< Operator (TDateTime*)

Pascal

```
operator <<(var arg: TDateTime);
```

C++

```
AutoCmd& operator <<(TDateTime* arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.13 AutoCmd::<< Operator (Variant&)

Pascal

```
operator <<(const arg: Variant&);
```

C++

```
AutoCmd& operator <<(const Variant& arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.14 AutoCmd::<< Operator (Variant*)

Pascal

```
operator <<(var arg: Variant);
```

C++

```
AutoCmd& operator <<(Variant* arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.15 AutoCmd::<< Operator (WordBool)

Pascal

```
operator <<(const arg: WordBool);
```

C++

```
AutoCmd& operator <<(const WordBool arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.16 AutoCmd::<< Operator (WordBool*)

Pascal

```
operator <<(var arg: WordBool);
```

C++

```
AutoCmd& operator <<(WordBool* arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.17 AutoCmd::<< Operator (bool)

Pascal

```
operator <<(const arg: bool);
```

C++

```
AutoCmd& operator <<(const bool arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.18 AutoCmd::<< Operator (double)

Pascal

```
operator <<(const arg: double);
```

C++

```
AutoCmd& operator <<(const double arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.19 AutoCmd::<< Operator (double*)

Pascal

```
operator <<(var arg: double);
```

C++

```
AutoCmd& operator <<(double* arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.20 AutoCmd::<< Operator (float)

Pascal

```
operator <<(const arg: float);
```

C++

```
AutoCmd& operator <<(const float arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.21 AutoCmd::<< Operator (float*)

Pascal

```
operator <<(var arg: float);
```

C++

```
AutoCmd& operator <<(float* arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.22 AutoCmd::<< Operator (int)

Pascal

```
operator <<(const arg: Integer);
```

C++

```
AutoCmd& operator <<(const int arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.23 AutoCmd::<< Operator (int*)

Pascal

```
operator <<(var arg: Integer);
```

C++

```
AutoCmd& operator <<(int* arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.24 AutoCmd::<< Operator (short)

Pascal

```
operator <<(const arg: short);
```

C++

```
AutoCmd& operator <<(const short arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.25 AutoCmd::<< Operator (short*)

Pascal

```
operator <<(var arg: short);
```

C++

```
AutoCmd& operator <<(short* arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.26 AutoCmd::<< Operator (wchar_t*)

Pascal

```
operator <<(var arg: wchar_t);
```

C++

```
AutoCmd& operator <<(wchar_t* arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.2.6.1.27 AutoCmd::<< Operator (wchar_t**)

Pascal

```
operator <<(var arg: wchar_t*);
```

C++

```
AutoCmd& operator <<(wchar_t** arg);
```

Description

This is <<, a member of class AutoCmd.

158.1.3 Currency Class

System::Currency is the C++ implementation of the Delphi System::Currency data type.

Class Hierarchy

Pascal

```
Currency = class(CurrencyBase);
```

C++

```
class Currency : public CurrencyBase;
```

File

syscurr.h

Description

System::Currency is a C++ class that implements a Delphi intrinsic type, also called System::Currency. Use System::Currency to hold monetary values.

Note: A System::Currency object is implemented as a 64-bit signed integer. Avoid operations that might cause integer overflow.

Note: Use only the operators defined by System::Currency. The compiler ignores any operators you overload yourself. Use the System::Currency constructors to create System::Currency objects from pointers to other System::Currency objects, and also from numeric and string values..

Note: To utilize the C++ streaming operators (<< and >>) with System::Currency, you must use #include <iostream> before #include <syscurr.h>. This occurs automatically if you include vcl.h, clx.h, or System.hpp. The following stream operators are defined:

```
ostream& operator << (ostream& os, const Currency& arg);
istream& operator >> (istream& is, Currency& arg);
```

See Also

System::CurrencyComp

System::CurrencyCurrencyBase

158.1.3.1 Currency Members

The following tables list the members exposed by Currency.

Public Data Members

	Name	Description
	Val (see page 22166)	This is Val, a member of class CurrencyBase.

Public Friends

Name	Description
Currency __fastcall operator -(double lhs, const Currency& rhs) (see page 22164)	This is friend Currency __fastcall operator -(double lhs, const Currency& rhs).
Currency __fastcall operator -(int lhs, const Currency& rhs) (see page 22164)	This is friend Currency __fastcall operator -(int lhs, const Currency& rhs).
Currency __fastcall operator *(double lhs, const Currency& rhs) (see page 22164)	This is friend Currency __fastcall operator *(double lhs, const Currency& rhs).
Currency __fastcall operator *(int lhs, const Currency& rhs) (see page 22164)	This is friend Currency __fastcall operator *(int lhs, const Currency& rhs).
Currency __fastcall operator /(double lhs, const Currency& rhs) (see page 22165)	This is friend Currency __fastcall operator /(double lhs, const Currency& rhs).
Currency __fastcall operator /(int lhs, const Currency& rhs) (see page 22165)	This is friend Currency __fastcall operator /(int lhs, const Currency& rhs).
Currency __fastcall operator +(double lhs, const Currency& rhs) (see page 22165)	This is friend Currency __fastcall operator +(double lhs, const Currency& rhs).
Currency __fastcall operator +(int lhs, const Currency& rhs) (see page 22165)	This is friend Currency __fastcall operator +(int lhs, const Currency& rhs).

Public Methods

	Name	Description
	Currency (see page 22147)	This is the overview for the Currency constructor overload.

Public Operators

	Name	Description
	- (see page 22148)	This is the overview for the - operator overload.
	-- (see page 22150)	This is the overview for the -- operator overload.
	! (see page 22150)	Returns a new System::Currency::Currency (see page 22147) object that is the logical negation of this System::Currency::Currency (see page 22147) object.
	!= (see page 22151)	This is the overview for the != operator overload.
	% (see page 22151)	Returns a new System::Currency::Currency (see page 22147) object with the value that is the remainder of this System::Currency::Currency (see page 22147) object divided by a specified integer.
	%= (see page 22152)	Assigns this System::Currency::Currency (see page 22147) object the remainder of its initial value divided by integer rhs.
	* (see page 22152)	This is the overview for the * operator overload.
	*= (see page 22153)	Assigns this System::Currency::Currency (see page 22147) object the result of its initial value multiplied by the value of System::Currency::Currency (see page 22147) object rhs.
	/ (see page 22153)	This is the overview for the / operator overload.
	/= (see page 22154)	Assigns this System::Currency::Currency (see page 22147) object the result of its initial value divided by the value of System::Currency::Currency (see page 22147) object rhs.
	+ (see page 22155)	This is the overview for the + operator overload.
	++ (see page 22156)	This is the overview for the ++ operator overload.
	+= (see page 22156)	Increments the value of this System::Currency::Currency (see page 22147) object by the value of rhs.
	< (see page 22157)	This is the overview for the < operator overload.
	<= (see page 22158)	This is the overview for the <= operator overload.
	-= (see page 22159)	Decrements the value of this System::Currency::Currency (see page 22147) object by the value of rhs.
	= (see page 22158)	This is the overview for the = operator overload.
	== (see page 22160)	This is the overview for the == operator overload.
	> (see page 22160)	This is the overview for the > operator overload.
	>= (see page 22161)	This is the overview for the >= operator overload.
	AnsiString (see page 22162)	Converts the System::Currency::Currency (see page 22147) object to an AnsiString that displays its value.
	double (see page 22162)	Converts the System::Currency::Currency (see page 22147) object to a double value.
	int (see page 22163)	Converts the System::Currency::Currency (see page 22147) object to an integer value

158.1.3.2 Currency Constructor

This is the overview for the Currency constructor overload.

Overload List

	Name	Description
	Currency::Currency () (see page 22147)	Creates and initializes a System::Currency::Currency (see page 22145) object.
	Currency::Currency (AnsiString&) (see page 22147)	Creates and initializes a System::Currency::Currency (see page 22145) object.
	Currency::Currency (Currency&) (see page 22147)	Creates and initializes a System::Currency::Currency (see page 22145) object.
	Currency::Currency (CurrencyBase&) (see page 22147)	Creates and initializes a System::Currency::Currency (see page 22145) object.
	Currency::Currency (double) (see page 22147)	Creates and initializes a System::Currency::Currency (see page 22145) object.
	Currency::Currency (int) (see page 22147)	Creates and initializes a System::Currency::Currency (see page 22145) object.

158.1.3.2.1 Currency::Currency Constructor ()

Creates and initializes a System::Currency::Currency (see page 22145) object.

Pascal

```
constructor Create;
constructor Create(val: double);
constructor Create(val: Integer);
constructor Create(const src: CurrencyBase&);
constructor Create(const src: Currency&);
constructor Create(const src: AnsiString&);
```

C++

```
__fastcall Currency();
__fastcall Currency(double val);
__fastcall Currency(int val);
__fastcall Currency(const CurrencyBase& src);
__fastcall Currency(const Currency& src);
__fastcall Currency(const AnsiString& src);
```

Description

Use System::Currency::Currency (see page 22145) to instantiate a System::Currency::Currency (see page 22145) object.

When called with no parameters, the resulting object has a value of 0.

When called with a System::Currency::Currency (see page 22145) object as the src parameter, the resulting object is a copy of src (the copy constructor).

When called with a System::Currency::CurrencyBase (see page 22166) object as the src parameter, the resulting object has the same value as the src parameter. System::Currency::CurrencyBase (see page 22166) is the base class for the System::Currency::Currency (see page 22145) type. It has a single member, a public Val (see page 22543) field that represents the value of the object.

When called with an AnsiString (see page 22162) object as the src parameter, the value of the resulting object is the parsed

value of the AnsiString (see page 22162). The AnsiString (see page 22162) must represent a numeric value within the range valid for System::Currency::Currency (see page 22145) (-922337203685477.5808 to 922337203685477.5807).

When called with a double (see page 22162) or integer as the val parameter, the value of the resulting object is the value of the val parameter.

158.1.3.3 Currency Operators

The operators of the Currency class are listed here.

Public Operators

	Name	Description
	- (see page 22148)	This is the overview for the - operator overload.
	-- (see page 22150)	This is the overview for the -- operator overload.
	! (see page 22150)	Returns a new System::Currency::Currency (see page 22147) object that is the logical negation of this System::Currency::Currency (see page 22147) object.
	!= (see page 22151)	This is the overview for the != operator overload.
	% (see page 22151)	Returns a new System::Currency::Currency (see page 22147) object with the value that is the remainder of this System::Currency::Currency (see page 22147) object divided by a specified integer.
	%= (see page 22152)	Assigns this System::Currency::Currency (see page 22147) object the remainder of its initial value divided by integer rhs.
	* (see page 22152)	This is the overview for the * operator overload.
	*= (see page 22153)	Assigns this System::Currency::Currency (see page 22147) object the result of its initial value multiplied by the value of System::Currency::Currency (see page 22147) object rhs.
	/ (see page 22153)	This is the overview for the / operator overload.
	/= (see page 22154)	Assigns this System::Currency::Currency (see page 22147) object the result of its initial value divided by the value of System::Currency::Currency (see page 22147) object rhs.
	+ (see page 22155)	This is the overview for the + operator overload.
	++ (see page 22156)	This is the overview for the ++ operator overload.
	+= (see page 22156)	Increments the value of this System::Currency::Currency (see page 22147) object by the value of rhs.
	< (see page 22157)	This is the overview for the < operator overload.
	<= (see page 22158)	This is the overview for the <= operator overload.
	= (see page 22158)	This is the overview for the = operator overload.
	-= (see page 22159)	Decrements the value of this System::Currency::Currency (see page 22147) object by the value of rhs.
	== (see page 22160)	This is the overview for the == operator overload.
	> (see page 22160)	This is the overview for the > operator overload.
	>= (see page 22161)	This is the overview for the >= operator overload.
	AnsiString (see page 22162)	Converts the System::Currency::Currency (see page 22147) object to an AnsiString that displays its value.
	double (see page 22162)	Converts the System::Currency::Currency (see page 22147) object to a double value.
	int (see page 22163)	Converts the System::Currency::Currency (see page 22147) object to an integer value

158.1.3.3.1 - Operator

This is the overview for the - operator overload.

Overload List

	Name	Description
	Currency::- () (see page 22149)	This is -, a member of class Currency.
	Currency::- (Currency&) (see page 22149)	This is -, a member of class Currency.
	Currency::- (double) (see page 22149)	This is -, a member of class Currency.
	Currency::- (int) (see page 22149)	This is -, a member of class Currency.

158.1.3.3.1.1 Currency::- Operator ()

Pascal

```
operator -;
```

C++

```
__fastcall Currency operator -() const;
```

Description

This is -, a member of class Currency.

158.1.3.3.1.2 Currency::- Operator (Currency&)

Pascal

```
operator -(const rhs: Currency&);
```

C++

```
__fastcall Currency operator -(const Currency& rhs) const;
```

Description

This is -, a member of class Currency.

158.1.3.3.1.3 Currency::- Operator (double)

Pascal

```
operator -(rhs: double);
```

C++

```
__fastcall Currency operator -(double rhs) const;
```

Description

This is -, a member of class Currency.

158.1.3.3.1.4 Currency::- Operator (int)

Pascal

```
operator -(rhs: Integer);
```

C++

```
__fastcall Currency operator -(int rhs) const;
```

Description

This is -, a member of class Currency.

158.1.3.3.2 -- Operator

This is the overview for the -- operator overload.

Overload List

	Name	Description
	Currency::-- () (see page 22150)	Decrements the value of the System::Currency::Currency (see page 22147) object by a specified integer value.
	Currency::-- (int) (see page 22150)	Decrements the value of the System::Currency::Currency (see page 22147) object by a specified integer value.

158.1.3.3.2.1 Currency::-- Operator ()

Decrements the value of the System::Currency::Currency (see page 22147) object by a specified integer value.

Pascal

```
operator --;
operator --(: Integer);
```

C++

```
Currency& operator --();
Currency operator --(int);
```

Description

Use the unary System::Currency (see page 22145)::-- to decrement the value of the System::Currency::Currency (see page 22147) object by 1.

Use the binary System::Currency (see page 22145)::-- to decrement the value by the integer specified as an operand.

See Also

System::Currency::++ (see page 22156)

System::Currency::-= (see page 22159)

System::Currency::operator_minus_binary

158.1.3.3.3 Currency::! Operator

Returns a new System::Currency::Currency (see page 22147) object that is the logical negation of this System::Currency::Currency (see page 22147) object.

Pascal

```
operator !;
```

C++

```
__fastcall Currency operator !() const;
```

Description

If the value of the System::Currency::Currency (see page 22147) object is nonzero, the System::Currency (see page 22145)::! returns 0. If the value of the System::Currency::Currency (see page 22147) object is 0, the System::Currency (see page 22145)::! returns 1.

See Also

System::Currency::!= (see page 22151)

System::Currency::operator_minus_unary

158.1.3.3.4 != Operator

This is the overview for the != operator overload.

Overload List

	Name	Description
	Currency::!= (Currency&) (see page 22151)	Tests for inequality between the System::Currency::Currency (see page 22147) object's value and a specified value.
	Currency::!= (double) (see page 22151)	Tests for inequality between the System::Currency::Currency (see page 22147) object's value and a specified value.
	Currency::!= (int) (see page 22151)	Tests for inequality between the System::Currency::Currency (see page 22147) object's value and a specified value.

158.1.3.3.4.1 Currency::!= Operator (Currency&)

Tests for inequality between the System::Currency::Currency (see page 22147) object's value and a specified value.

Pascal

```
operator !=(const rhs: Currency&);
operator !=(rhs: Integer);
operator !=(rhs: double);
```

C++

```
__fastcall bool operator !=(const Currency& rhs) const;
__fastcall bool operator !=(int rhs) const;
__fastcall bool operator !=(double rhs) const;
```

Description

The System::Currency (see page 22145)::!= compares the value of the System::Currency::Currency (see page 22147) object with the value specified by the rhs operand. If the values differ, the System::Currency (see page 22145)::!= returns true. If the values are the same, the System::Currency (see page 22145)::!= returns false.

See Also

System::Currency::= (see page 22159)

System::Currency::> (see page 22161)

System::Currency::>= (see page 22161)

System::Currency::< (see page 22157)

System::Currency::<= (see page 22158)

158.1.3.3.5 Currency::% Operator

Returns a new System::Currency::Currency (see page 22147) object with the value that is the remainder of this System::Currency::Currency (see page 22147) object divided by a specified integer.

Pascal

```
operator %(rhs: Integer);
```

C++

```
__fastcall Currency operator %(int rhs) const;
```

Description

Use the System::Currency (see page 22145)::% to obtain the remainder from dividing the value of this System::Currency::Currency (see page 22147) object by the integer specified by the rhs operand.

See Also

System::Currency::%= (see page 22152)

System::Currency::/ (see page 22154)

158.1.3.3.6 Currency::%= Operator

Assigns this System::Currency::Currency (see page 22147) object the remainder of its initial value divided by integer rhs.

Pascal

```
operator %=(rhs: Integer);
```

C++

```
__fastcall Currency& operator %=(int rhs);
```

Description

Use the System::Currency (see page 22145)::%= to obtain the remainder from dividing this System::Currency::Currency (see page 22147) object's value by the Value of the System::Currency::Currency (see page 22147) object specified by rhs.

See Also

System::Currency::= (see page 22159)

System::Currency::/= (see page 22154)

System::Currency::% (see page 22151)

158.1.3.3.7 * Operator

This is the overview for the * operator overload.

Overload List

	Name	Description
	Currency::* (Currency&) (see page 22152)	Multiplies the value of the System::Currency::Currency (see page 22147) object by a specified value.
	Currency::* (double) (see page 22152)	Multiplies the value of the System::Currency::Currency (see page 22147) object by a specified value.
	Currency::* (int) (see page 22152)	Multiplies the value of the System::Currency::Currency (see page 22147) object by a specified value.

158.1.3.3.7.1 Currency::* Operator (Currency&)

Multiplies the value of the System::Currency::Currency (see page 22147) object by a specified value.

Pascal

```
operator *(const rhs: Currency&);
operator *(rhs: Integer);
operator *(rhs: double);
```

C++

```
__fastcall Currency operator *(const Currency& rhs) const;
__fastcall Currency operator *(int rhs) const;
__fastcall Currency operator *(double rhs) const;
```

Description

Use the System::Currency (see page 22145)::* to multiply the value of the System::Currency::Currency (see page 22147) object by the value specified by the rhs operand.

Note that there are two related friend System::Currency (see page 22145)::*s:

```
friend Currency __fastcall operator *(int lhs, const Currency& rhs);
```

multiples the values of integer lhs and System::Currency::Currency (see page 22147) object rhs and returns a new System::Currency::Currency (see page 22147) object with the resulting value.

```
friend Currency __fastcall operator *(double lhs, const Currency& rhs);
```

multiples the values of double (see page 22162) lhs and System::Currency::Currency (see page 22147) object rhs and returns a new System::Currency::Currency (see page 22147) object with the resulting value.

See Also

System::Currency::*= (see page 22153)

System::Currency::/ (see page 22154)

158.1.3.3.8 Currency::*= Operator

Assigns this System::Currency::Currency (see page 22147) object the result of its initial value multiplied by the value of System::Currency::Currency (see page 22147) object rhs.

Pascal

```
operator *=(const rhs: Currency&);
```

C++

```
__fastcall Currency& operator *=(const Currency& rhs);
```

Description

Use the System::Currency (see page 22145)::*= to multiply this System::Currency::Currency (see page 22147) object's value by the Value of the System::Currency::Currency (see page 22147) object specified by rhs.

See Also

System::Currency::= (see page 22159)

System::Currency::/= (see page 22154)

System::Currency::* (see page 22152)

158.1.3.3.9 / Operator

This is the overview for the / operator overload.

Overload List

	Name	Description
	Currency::/ (Currency&) (see page 22154)	Divides the value of the System::Currency::Currency (see page 22147) object by a specified value.
	Currency::/ (double) (see page 22154)	Divides the value of the System::Currency::Currency (see page 22147) object by a specified value.
	Currency::/ (int) (see page 22154)	Divides the value of the System::Currency::Currency (see page 22147) object by a specified value.

158.1.3.3.9.1 Currency::/ Operator (Currency&)

Divides the value of the System::Currency::Currency (see page 22147) object by a specified value.

Pascal

```
operator /(const rhs: Currency&);
operator /(rhs: Integer);
operator /(rhs: double);
```

C++

```
__fastcall Currency operator /(const Currency& rhs) const;
__fastcall Currency operator /(int rhs) const;
__fastcall Currency operator /(double rhs) const;
```

Description

Use the System::Currency (see page 22145)::/ to divide the value of the System::Currency::Currency (see page 22147) object by the value specified by the rhs operand.

Note that there are two related friend System::Currency::operator_slashs:

```
friend Currency __fastcall operator /(int lhs, const Currency& rhs);
```

divides the value of int (see page 22163) lhs by the value of System::Currency::Currency (see page 22147) object rhs and returns a new System::Currency::Currency (see page 22147) object with the resulting value.

```
friend Currency __fastcalloperator /(double lhs, const Currency& rhs);
```

divides the value of double (see page 22162) lhs by the value of System::Currency::Currency (see page 22147) object rhs and returns a System::Currency::Currency (see page 22147) object with the resulting value.

See Also

System::Currency::/= (see page 22154)

System::Currency::* (see page 22152)

System::Currency::% (see page 22151)

158.1.3.3.10 Currency::/= Operator

Assigns this System::Currency::Currency (see page 22147) object the result of its initial value divided by the value of System::Currency::Currency (see page 22147) object rhs.

Pascal

```
operator /=(const rhs: Currency&);
```

C++

```
__fastcall Currency& operator /=(const Currency& rhs);
```

Description

Use the System::Currency (see page 22145)::/= to divide this System::Currency::Currency (see page 22147) object's value by the Value of the System::Currency::Currency (see page 22147) object specified by rhs.

See Also

System::Currency::= (see page 22159)

System::Currency::*= (see page 22153)

System::Currency::/ (see page 22154)

158.1.3.3.11 + Operator

This is the overview for the + operator overload.

Overload List

	Name	Description
	Currency::+ (Currency&) (see page 22155)	Adds a specified value to the System::Currency::Currency (see page 22147) object.
	Currency::+ (double) (see page 22155)	Adds a specified value to the System::Currency::Currency (see page 22147) object.
	Currency::+ (int) (see page 22155)	Adds a specified value to the System::Currency::Currency (see page 22147) object.

158.1.3.3.11.1 Currency::+ Operator (Currency&)

Adds a specified value to the System::Currency::Currency (see page 22147) object.

Pascal

```
operator +(const rhs: Currency&);
operator +(rhs: Integer);
operator +(rhs: double);
```

C++

```
__fastcall Currency operator +(const Currency& rhs) const;
__fastcall Currency operator +(int rhs) const;
__fastcall Currency operator +(double rhs) const;
```

Description

Use the System::Currency (see page 22145)::+ to add the value specified by the rhs operand to the System::Currency::Currency (see page 22147) object. If the rhs operand is another System::Currency::Currency (see page 22147) object, the System::Currency (see page 22145)::+ adds its value to the System::Currency::Currency (see page 22147) object. If it is an integer or double (see page 22162) value, the System::Currency (see page 22145)::+ adds that value to the value of the System::Currency::Currency (see page 22147) object.

Note that there are two related friend System::Currency (see page 22145)::+s:

```
friend Currency __fastcall operator +(int lhs, const Currency& rhs);
```

returns a new System::Currency::Currency (see page 22147) object that is the sum of int (see page 22163) lhs and System::Currency::Currency (see page 22147) object rhs.

```
friend Currency __fastcall operator +(double lhs, const Currency& rhs);
```

returns a new System::Currency::Currency (see page 22147) object that is the sum of double (see page 22162) lhs and

System::Currency::Currency (see page 22147) object rhs.

See Also

System::Currency::+= (see page 22156)

System::Currency::++ (see page 22156)

System::Currency::operator_minus_binary

158.1.3.3.12 ++ Operator

This is the overview for the ++ operator overload.

Overload List

	Name	Description
	Currency::++ () (see page 22156)	Increments the value of the System::Currency::Currency (see page 22147) object by a specified integer value.
	Currency::++ (int) (see page 22156)	Increments the value of the System::Currency::Currency (see page 22147) object by a specified integer value.

158.1.3.3.12.1 Currency::++ Operator ()

Increments the value of the System::Currency::Currency (see page 22147) object by a specified integer value.

Pascal

```
operator ++;
operator ++(: Integer);
```

C++

```
Currency& operator ++();
Currency operator ++(int);
```

Description

Use the unary System::Currency (see page 22145)::++ to increment the value of the System::Currency::Currency (see page 22147) object by 1.

Use the binary System::Currency (see page 22145)::++ to increment the value by the integer specified as an operand.

See Also

System::Currency::-- (see page 22150)

System::Currency::+= (see page 22156)

System::Currency::+ (see page 22155)

158.1.3.3.13 Currency::+= Operator

Increments the value of this System::Currency::Currency (see page 22147) object by the value of rhs.

Pascal

```
operator +=(const rhs: Currency&);
```

C++

```
__fastcall Currency& operator +=(const Currency& rhs);
```

Description

Use the System::Currency (see page 22145)::+= to add the value of another System::Currency::Currency (see page 22147) object to this one.

See Also

System::Currency::= (see page 22159)

System::Currency::-= (see page 22159)

System::Currency::+ (see page 22155)

System::Currency::++ (see page 22156)

158.1.3.3.14 < Operator

This is the overview for the < operator overload.

Overload List

	Name	Description
	Currency::< (Currency&) (see page 22157)	Tests whether this System::Currency::Currency (see page 22147) value is less than a specified value.
	Currency::< (double) (see page 22157)	Tests whether this System::Currency::Currency (see page 22147) value is less than a specified value.
	Currency::< (int) (see page 22157)	Tests whether this System::Currency::Currency (see page 22147) value is less than a specified value.

158.1.3.3.14.1 Currency::< Operator (Currency&)

Tests whether this System::Currency::Currency (see page 22147) value is less than a specified value.

Pascal

```
operator <(const rhs: Currency&);
operator <(rhs: Integer);
operator <(rhs: double);
```

C++

```
__fastcall bool operator <(const Currency& rhs) const;
__fastcall bool operator <(int rhs) const;
__fastcall bool operator <(double rhs) const;
```

Description

The < System::Currency (see page 22145)::< compares the value of the System::Currency::Currency (see page 22147) object with the value specified by the rhs operand. It returns true if this System::Currency::Currency (see page 22147) object has the lesser value, false otherwise.

See Also

System::Currency::!= (see page 22151)

System::Currency::> (see page 22161)

System::Currency::>= (see page 22161)

System::Currency::== (see page 22160)

System::Currency::<= (see page 22158)

158.1.3.3.15 <= Operator

This is the overview for the <= operator overload.

Overload List

	Name	Description
	Currency::<= (Currency&) (see page 22158)	Tests whether this System::Currency::Currency (see page 22147) value is less than or equal to a specified value.
	Currency::<= (double) (see page 22158)	Tests whether this System::Currency::Currency (see page 22147) value is less than or equal to a specified value.
	Currency::<= (int) (see page 22158)	Tests whether this System::Currency::Currency (see page 22147) value is less than or equal to a specified value.

158.1.3.3.15.1 Currency::<= Operator (Currency&)

Tests whether this System::Currency::Currency (see page 22147) value is less than or equal to a specified value.

Pascal

```
operator <=(const rhs: Currency&);
operator <=(rhs: Integer);
operator <=(rhs: double);
```

C++

```
__fastcall bool operator <=(const Currency& rhs) const;
__fastcall bool operator <=(int rhs) const;
__fastcall bool operator <=(double rhs) const;
```

Description

The <= System::Currency (see page 22145)::<= compares the value of the System::Currency::Currency (see page 22147) object with the value specified by the rhs operand. It returns true if this System::Currency::Currency (see page 22147) object has a value less than or equal to that of rhs, false otherwise.

See Also

System::Currency::!= (see page 22151)

System::Currency::> (see page 22161)

System::Currency::>= (see page 22161)

System::Currency::< (see page 22157)

System::Currency::== (see page 22160)

158.1.3.3.16 = Operator

This is the overview for the = operator overload.

Overload List

	Name	Description
	Currency::= (Currency&) (see page 22159)	Assigns the value of the System::Currency::Currency (see page 22147) object.

	Currency::= (CurrencyBase&) (see page 22159)	Assigns the value of the System::Currency::Currency (see page 22147) object.
	Currency::= (double) (see page 22159)	Assigns the value of the System::Currency::Currency (see page 22147) object.
	Currency::= (int) (see page 22159)	Assigns the value of the System::Currency::Currency (see page 22147) object.

158.1.3.3.16.1 Currency::= Operator (Currency&)

Assigns the value of the System::Currency::Currency (see page 22147) object.

Pascal

```
operator =(rhs: double);
operator =(rhs: Integer);
operator =(const rhs: CurrencyBase&);
operator =(const rhs: Currency&);
```

C++

```
__fastcall Currency& operator =(double rhs);
__fastcall Currency& operator =(int rhs);
__fastcall Currency& operator =(const CurrencyBase& rhs);
__fastcall Currency& operator =(const Currency& rhs);
```

Description

The assignment System::Currency (see page 22145)::= assigns the value of the rhs operand to the System::Currency::Currency (see page 22147) object. The assignment System::Currency (see page 22145)::= can take

a double (see page 22162) or integer value, which is assigned to the System::Currency::Currency (see page 22147) object.

a System::Currency::CurrencyBase (see page 22166) or System::Currency::Currency (see page 22147) object, whose value is assigned to the System::Currency::Currency (see page 22147) object.

158.1.3.3.17 Currency::-= Operator

Decrements the value of this System::Currency::Currency (see page 22147) object by the value of rhs.

Pascal

```
operator -=(const rhs: Currency&);
```

C++

```
__fastcall Currency& operator -=(const Currency& rhs);
```

Description

Use the System::Currency (see page 22145)::-= to subtract the value of another System::Currency::Currency (see page 22147) object from this one.

See Also

System::Currency::= (see page 22159)

System::Currency::+= (see page 22156)

System::Currency::operator_minus_binary

System::Currency::-- (see page 22150)

158.1.3.3.18 == Operator

This is the overview for the == operator overload.

Overload List

	Name	Description
	Currency::== (Currency&) (see page 22160)	Tests for equality between the System::Currency::Currency (see page 22147) object's value and a specified value.
	Currency::== (double) (see page 22160)	Tests for equality between the System::Currency::Currency (see page 22147) object's value and a specified value.
	Currency::== (int) (see page 22160)	Tests for equality between the System::Currency::Currency (see page 22147) object's value and a specified value.

158.1.3.3.18.1 Currency::== Operator (Currency&)

Tests for equality between the System::Currency::Currency (see page 22147) object's value and a specified value.

Pascal

```
operator ==(const rhs: Currency&);
operator ==(rhs: Integer);
operator ==(rhs: double);
```

C++

```
__fastcall bool operator ==(const Currency& rhs) const;
__fastcall bool operator ==(int rhs) const;
__fastcall bool operator ==(double rhs) const;
```

Description

The == System::Currency (see page 22145)::== compares the value of the System::Currency::Currency (see page 22147) object with the value specified by the rhs operand. If the two values are the same, the System::Currency (see page 22145)::== returns true. If the values differ, the System::Currency (see page 22145)::== returns false.

See Also

System::Currency::!= (see page 22151)

System::Currency::> (see page 22161)

System::Currency::>= (see page 22161)

System::Currency::< (see page 22157)

System::Currency::<= (see page 22158)

158.1.3.3.19 > Operator

This is the overview for the > operator overload.

Overload List

	Name	Description
	Currency::> (Currency&) (see page 22161)	Tests whether this System::Currency::Currency (see page 22147) value is greater than a specified value.

	Currency::> (double) (see page 22161)	Tests whether this System::Currency::Currency (see page 22147) value is greater than a specified value.
	Currency::> (int) (see page 22161)	Tests whether this System::Currency::Currency (see page 22147) value is greater than a specified value.

158.1.3.3.19.1 Currency::> Operator (Currency&)

Tests whether this System::Currency::Currency (see page 22147) value is greater than a specified value.

Pascal

```
operator >(const rhs: Currency&);
operator >(rhs: Integer);
operator >(rhs: double);
```

C++

```
__fastcall bool operator >(const Currency& rhs) const;
__fastcall bool operator >(int rhs) const;
__fastcall bool operator >(double rhs) const;
```

Description

The System::Currency (see page 22145)::> compares the value of the System::Currency::Currency (see page 22147) object with the value specified by the rhs operand. It returns true if this System::Currency::Currency (see page 22147) object has a greater value than rhs, false otherwise.

See Also

System::Currency::!= (see page 22151)

System::Currency::== (see page 22160)

System::Currency::>= (see page 22161)

System::Currency::< (see page 22157)

System::Currency::<= (see page 22158)

158.1.3.3.20 >= Operator

This is the overview for the >= operator overload.

Overload List

	Name	Description
	Currency::>= (Currency&) (see page 22161)	Tests whether this System::Currency::Currency (see page 22147) value is greater than or equal to a specified value.
	Currency::>= (double) (see page 22161)	Tests whether this System::Currency::Currency (see page 22147) value is greater than or equal to a specified value.
	Currency::>= (int) (see page 22161)	Tests whether this System::Currency::Currency (see page 22147) value is greater than or equal to a specified value.

158.1.3.3.20.1 Currency::>= Operator (Currency&)

Tests whether this System::Currency::Currency (see page 22147) value is greater than or equal to a specified value.

Pascal

```
operator >=(const rhs: Currency&);
operator >=(rhs: Integer);
operator >=(rhs: double);
```

C++

```
__fastcall bool operator >=(const Currency& rhs) const;
__fastcall bool operator >=(int rhs) const;
__fastcall bool operator >=(double rhs) const;
```

Description

The >= System::Currency (see page 22145)::>= compares the value of the System::Currency::Currency (see page 22147) object with the value specified by the rhs operand. It returns true if this System::Currency::Currency (see page 22147) object's value is greater than or equal to the value of rhs, false otherwise.

See Also

System::Currency::!= (see page 22151)

System::Currency::> (see page 22161)

System::Currency::== (see page 22160)

System::Currency::< (see page 22157)

System::Currency::<= (see page 22158)

158.1.3.3.21 Currency::AnsiString Operator

Converts the System::Currency::Currency (see page 22147) object to an AnsiString that displays its value.

Pascal

```
operator AnsiString;
```

C++

```
__fastcall operator AnsiString() const;
```

Description

AnsiString is a conversion System::Currency::AnsiString that converts the currency value into its string representation. The conversion uses the ffGeneral floating point conversion rules. For further details, see the description of the CurrToStrF and FloatToStrF methods of the AnsiString class.

See Also

System::Currency::CurrToStrF

System::Currency::FloatToStrF

System::Currency::int (see page 22163)

System::Currency::double (see page 22162)

158.1.3.3.22 Currency::double Operator

Converts the System::Currency::Currency (see page 22147) object to a double value.

Pascal

```
operator double;
```

C++

```
__fastcall operator double() const;
```

Description

double is a conversion System::Currency::double that converts a System::Currency::Currency (see page 22147) object to a double.

See Also

System::Currency::int (see page 22163)

System::Currency::AnsiString (see page 22162)

158.1.3.3.23 Currency::int Operator

Converts the System::Currency::Currency (see page 22147) object to an integer value

Pascal

```
operator int;
```

C++

```
__fastcall operator int() const;
```

Description

int is a conversion System::Currency::int that converts a System::Currency::Currency (see page 22147) object to an int.

See Also

System::Currency::double (see page 22162)

System::Currency::AnsiString (see page 22162)

158.1.3.4 Currency Friends

The friends of the Currency class are listed here.

Public Friends

Name	Description
Currency __fastcall operator -(double lhs, const Currency& rhs) (see page 22164)	This is friend Currency __fastcall operator -(double lhs, const Currency& rhs).
Currency __fastcall operator -(int lhs, const Currency& rhs) (see page 22164)	This is friend Currency __fastcall operator -(int lhs, const Currency& rhs).
Currency __fastcall operator *(double lhs, const Currency& rhs) (see page 22164)	This is friend Currency __fastcall operator *(double lhs, const Currency& rhs).
Currency __fastcall operator *(int lhs, const Currency& rhs) (see page 22164)	This is friend Currency __fastcall operator *(int lhs, const Currency& rhs).
Currency __fastcall operator /(double lhs, const Currency& rhs) (see page 22165)	This is friend Currency __fastcall operator /(double lhs, const Currency& rhs).
Currency __fastcall operator /(int lhs, const Currency& rhs) (see page 22165)	This is friend Currency __fastcall operator /(int lhs, const Currency& rhs).
Currency __fastcall operator +(double lhs, const Currency& rhs) (see page 22165)	This is friend Currency __fastcall operator +(double lhs, const Currency& rhs).
Currency __fastcall operator +(int lhs, const Currency& rhs) (see page 22165)	This is friend Currency __fastcall operator +(int lhs, const Currency& rhs).

158.1.3.4.1 friend Currency __fastcall operator -(double lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator -(double lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator -(double lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator -(double lhs, const Currency& rhs).

158.1.3.4.2 friend Currency __fastcall operator -(int lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator -(int lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator -(int lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator -(int lhs, const Currency& rhs).

158.1.3.4.3 friend Currency __fastcall operator *(double lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator *(double lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator *(double lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator *(double lhs, const Currency& rhs).

158.1.3.4.4 friend Currency __fastcall operator *(int lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator *(int lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator *(int lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator *(int lhs, const Currency& rhs).

158.1.3.4.5 friend Currency __fastcall operator /(double lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator /(double lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator /(double lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator /(double lhs, const Currency& rhs).

158.1.3.4.6 friend Currency __fastcall operator /(int lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator /(int lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator /(int lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator /(int lhs, const Currency& rhs).

158.1.3.4.7 friend Currency __fastcall operator +(double lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator +(double lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator +(double lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator +(double lhs, const Currency& rhs).

158.1.3.4.8 friend Currency __fastcall operator +(int lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator +(int lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator +(int lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator +(int lhs, const Currency& rhs).

158.1.4 CurrencyBase Class

Class Hierarchy

System::CurrencyBase

Pascal

```
CurrencyBase = class;
```

C++

```
class CurrencyBase;
```

File

syscurr.h

Description

This is class System::CurrencyBase.

158.1.4.1 CurrencyBase Members

The following tables list the members exposed by CurrencyBase.

Public Data Members

	Name	Description
	Val (see page 22166)	This is Val, a member of class CurrencyBase.

158.1.4.2 CurrencyBase Data Members

The data members of the CurrencyBase class are listed here.

Public Data Members

	Name	Description
	Val (see page 22166)	This is Val, a member of class CurrencyBase.

158.1.4.2.1 CurrencyBase::Val Data Member

Pascal

```
Val: __int64;
```

C++

```
__int64 Val;
```

Description

This is Val, a member of class CurrencyBase.

158.1.5 DelphiInterface Template

System::DelphiInterface is a C++ implementation of the Delphi interface.

Class Hierarchy

System::DelphiInterface

Pascal

```
DelphiInterface = class;
```

C++

```
template <class T>
class DelphiInterface;
```

File

systobj.h

Description

A Delphi interface is an abstract collection of properties, methods, and events with no implementation. It resembles a C++ class with only pure virtual methods. However a Delphi interface type supports built-in reference counting.

System::DelphiInterface is a C++ template that defines types compatible with Delphi interfaces, System::DelphiInterface defines reference counting logic in its initialization constructor, copy constructor, assignment operator, and destructor.

Note: Other operators that provide access to the underlying interface pointer do not handle reference counting. You may need to explicitly call AddRef or Release (see page 22169) to ensure proper reference counting.

System::DelphiInterface is a template:

```
template <class T> class RTL_DELPHIRETURN DelphiInterface;
```

where the T parameter is an underlying interface class or COM interface. In the object framework, the resulting System::DelphiInterface class is usually given a name such as _di_IUnknown, where IUnknown (see page 22586) is the name of the corresponding interface. Such interfaces are documented under the interface name (such as IFormDesigner).

See Also

Variant

TVariant

158.1.5.1 DelphiInterface Members

The following tables list the members exposed by DelphiInterface.

Public Methods

	Name	Description
	~DelphiInterface (see page 22168)	Frees an instance of System::DelphiInterface::DelphiInterface (see page 22168).
	DelphiInterface (see page 22168)	This is the overview for the DelphiInterface constructor overload.
	Release (see page 22169)	This is Release, a member of class DelphiInterface.

Public Operators

	Name	Description
	! (see page 22170)	Indicates whether the System::DelphiInterface::DelphiInterface (see page 22168) object represents a NULL interface.
	& (see page 22170)	Returns a pointer to the underlying interface pointer.
	* (see page 22170)	Returns the underlying interface pointer.
	= (see page 22170)	This is the overview for the = operator overload.
	-> (see page 22171)	Provides access to the properties and methods on the underlying interface.
	T* (see page 22171)	This is T*, a member of class DelphiInterface.

158.1.5.2 DelphiInterface::~DelphiInterface Destructor

Frees an instance of System::DelphiInterface::DelphiInterface (see page 22168).

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~DelphiInterface<T>();
```

Description

Before freeing the System::DelphiInterface::DelphiInterface (see page 22168) instance, System::DelphiInterface (see page 22167)::~DelphiInterface (see page 22167) calls the Release (see page 22169) method of the underlying interface.

158.1.5.3 DelphiInterface Constructor

This is the overview for the DelphiInterface constructor overload.

Overload List

	Name	Description
	DelphiInterface::DelphiInterface () (see page 22168)	Creates an instance of System::DelphiInterface::DelphiInterface (see page 22167).
	DelphiInterface::DelphiInterface (DelphiInterface<ANOTHERINTF> &) (see page 22168)	Creates an instance of System::DelphiInterface::DelphiInterface (see page 22167).
	DelphiInterface::DelphiInterface (DelphiInterface<T> &) (see page 22168)	Creates an instance of System::DelphiInterface::DelphiInterface (see page 22167).
	DelphiInterface::DelphiInterface (T*) (see page 22168)	Creates an instance of System::DelphiInterface::DelphiInterface (see page 22167).

158.1.5.3.1 DelphiInterface::DelphiInterface Constructor ()

Creates an instance of System::DelphiInterface::DelphiInterface (see page 22167).

Pascal

```
constructor Create;
constructor Create(const rhs: DelphiInterface<ANOTHERINTF> &);
constructor Create(const rhs: DelphiInterface<T> &);
constructor Create(var rhs: T);
```

C++

```
__fastcall DelphiInterface<T>();
template <class ANOTHERINTF> __fastcall DelphiInterface<T>(const DelphiInterface<ANOTHERINTF>
& rhs);
__fastcall DelphiInterface<T>(const DelphiInterface<T> & rhs);
__fastcall DelphiInterface<T>(T* rhs);
```

Description

The System::DelphiInterface::DelphiInterface (see page 22167) constructor is overloaded to create a System::DelphiInterface::DelphiInterface (see page 22167) in one of three ways:

When used with no parameters, System::DelphiInterface::DelphiInterface (see page 22167) creates a NULL interface.

When used with another System::DelphiInterface::DelphiInterface (see page 22167) object (the copy constructor), System::DelphiInterface::DelphiInterface (see page 22167) creates a new copy of the interface specified by rhs, incrementing the reference count of the underlying interface.

When used with an interface pointer of the underlying type, System::DelphiInterface::DelphiInterface (see page 22167) wraps the underlying interface specified by rhs and increments its reference count.

158.1.5.4 DelphiInterface Methods

The methods of the DelphiInterface class are listed here.

Public Methods

	Name	Description
	Release (see page 22169)	This is Release, a member of class DelphiInterface.

158.1.5.4.1 DelphiInterface::Release Method

Pascal

```
function Release: Integer;
```

C++

```
int Release();
```

Description

This is Release, a member of class DelphiInterface.

158.1.5.5 DelphiInterface Operators

The operators of the DelphiInterface class are listed here.

Public Operators

	Name	Description
	! (see page 22170)	Indicates whether the System::DelphiInterface::DelphiInterface (see page 22168) object represents a NULL interface.
	& (see page 22170)	Returns a pointer to the underlying interface pointer.
	* (see page 22170)	Returns the underlying interface pointer.
	= (see page 22170)	This is the overview for the = operator overload.
	-> (see page 22171)	Provides access to the properties and methods on the underlying interface.
	T* (see page 22171)	This is T*, a member of class DelphiInterface.

158.1.5.5.1 DelphiInterface::! Operator

Indicates whether the System::DelphiInterface::DelphiInterface (see page 22168) object represents a NULL interface.

Pascal

```
operator !;
```

C++

```
bool operator !() const;
```

Description

The ! System::DelphiInterface (see page 22167)::! returns true if the System::DelphiInterface::DelphiInterface (see page 22168) object represents a NULL interface, and false otherwise.

158.1.5.5.2 DelphiInterface::& Operator

Returns a pointer to the underlying interface pointer.

Pascal

```
operator &;
```

C++

```
__fastcall T** operator &();
```

Description

Use the & System::DelphiInterface (see page 22167)::& to access the underlying interface pointer. Care should be used with this System::DelphiInterface (see page 22167)::&, because it provides no reference counting support. For example, when using this System::DelphiInterface (see page 22167)::& to programmatically replace the underlying interface, you must first call Release (see page 22169) to decrement the reference count of the current interface and, depending on whether you are using this as an additional interface reference, you may need to increment the reference count on the interface being added.

158.1.5.5.3 DelphiInterface::* Operator

Returns the underlying interface pointer.

Pascal

```
operator *;
```

C++

```
__fastcall T& operator *();
```

Description

Use the & System::DelphiInterface (see page 22167)::* to access the underlying interface pointer. Care should be used with this System::DelphiInterface (see page 22167)::*, because it provides no reference counting support. For example, you may need to increment the reference count on the returned value.

158.1.5.5.4 = Operator

This is the overview for the = operator overload.

Overload List

	Name	Description
	DelphiInterface::= (DelphiInterface<ANOTHERINTF> &) (see page 22171)	Assigns a value to the System::DelphiInterface::DelphiInterface (see page 22168).
	DelphiInterface::= (DelphiInterface<T> &) (see page 22171)	Assigns a value to the System::DelphiInterface::DelphiInterface (see page 22168).
	DelphiInterface::= (T *) (see page 22171)	Assigns a value to the System::DelphiInterface::DelphiInterface (see page 22168).

158.1.5.5.4.1 DelphiInterface::= Operator (DelphiInterface<ANOTHERINTF> &)

Assigns a value to the System::DelphiInterface::DelphiInterface (see page 22168).

Pascal

```
operator =(var rhs: DelphiInterface<ANOTHERINTF>);
operator =(var rhs: DelphiInterface<T>);
operator =(var rhs: T);
```

C++

```
template <class ANOTHERINTF> __fastcall DelphiInterface<T>& operator
=(DelphiInterface<ANOTHERINTF> & rhs);
__fastcall DelphiInterface<T>& operator =(DelphiInterface<T> & rhs);
__fastcall DelphiInterface<T>& operator =(T * rhs);
```

Description

The assignment System::DelphiInterface (see page 22167)::= assigns a value to the underlying interface from an interface pointer of the underlying type or from another System::DelphiInterface::DelphiInterface (see page 22168) object that wraps the same interface type.

This System::DelphiInterface (see page 22167)::= manages all reference counting issues. That is, it decrements the reference count on the current interface (if any) before making the assignment and increments the reference count on the newly assigned interface (if any).

158.1.5.5.5 DelphiInterface::-> Operator

Provides access to the properties and methods on the underlying interface.

Pascal

```
operator ->;
```

C++

```
__fastcall T* operator ->() const;
```

Description

Use the System::DelphiInterface (see page 22167)::-> to access the properties and methods of the underlying interface. If the underlying interface is a dual interface, the System::DelphiInterface (see page 22167)::-> makes a vtable call.

158.1.5.5.6 DelphiInterface::T* Operator

Pascal

```
operator T*;
```

C++

```
__fastcall operator T*() const;
```

Description

This is T*, a member of class DelphiInterface.

158.1.6 DynamicArray Template

System::DynamicArray is the C++ analog for the Delphi dynamic array type.

Class Hierarchy

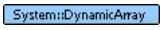

Pascal

```
DynamicArray = class;
```

C++

```
template <class T>
class DynamicArray;
```

File

sysdyn.h

Description

A Dynamic Array is a resizable collection implemented as an array. The System::DynamicArray template is a C++ implementation of Delphi Dynamic Array types. The System::DynamicArray template uses a binary layout compatible with the Delphi Dynamic Array implementation.

```
template <class T> class DELPHIRETURN DynamicArray;
```

You must specify one template parameter:

Parameter	Usage
type	the type of the elements

Length (see page 22181) of dynamic arrays

The size of a System::DynamicArray can be set or obtained from the Length (see page 22181) property:

//C++ example

```
DynamicArray<int> arrayOfInt;
arrayOfInt.Length = 10;
cout << "ArrayLength: " << arrayOfInt.Length << endl;
```

To free a System::DynamicArray, assign it a Length (see page 22181) of zero.

Accessing data

Access elements of a Dynamic Array via the subscript [] operator:

//C++ example

```
void InitArray(DynamicArray<char> &c_array)
{
  c_array[0] = 'A';
  c_array[1] = 'B';
```

```
  cout << "Third char is: " << c_array[2];
}
```

Bounds of dynamic arrays

The Low (see page 22182) and High (see page 22181) properties of System::DynamicArrays return the low and high bounds of the array respectively.

//C++ example

```
int TotalArray(const DynamicArray<int>& arrayOfInt)
{
int total=0;
for (int i=arrayOfInt.Low; i<=arrayOfInt.High; i++)
    total += arrayOfInt[i];
return total;
}
```

Note: The Low (see page 22182) property always returns 0. The High (see page 22181) property returns Length (see page 22181)-1. So you may also traverse a dynamic array via the following construct:

for (int i=0; i<arrayOfInt. Length (see page 22181); i++)

Assigning, comparing and copying dynamic arrays

Dynamic Arrays are reference counted. When a System::DynamicArray is assigned to another, only the reference is assigned (and the reference count adjusted), the content of the source is not copied. Similarly, when two Dynamic Arrays are compared, only the references are compared, not the contents. To copy the contents of a System::DynamicArray, you must use the Copy (see page 22176) (or CopyRange (see page 22177)) methods.

//C++ example

```
void foo(DynamicArray<int> &i_array)
{
  DynamicArray<int> temp = i_array;
  assert(temp == i_array); // temp and i_array point to same chunk
                           // of data..
  i_array[0] = 20;
  assert(temp[0] == 20); // Temp 'sees' changes to i_array..
  temp = i_array.Copy(); // Temp gets its own copy of data
  temp[0] = 10;
  assert(temp[0] != i_array[0]); // Above assignment did not
                                 // modify i_array.
}
```

Multidimensional dynamic arrays

Dynamic arrays can be multidimensional. Multidimensional dynamic arrays don't have to be rectangular. The Length (see page 22181) property can be used to set each dimension. The following example illustrates.

//C++ example

```
typedef DynamicArray< DynamicArray < AnsiString > > T2DStringArray;
void foo(T2DStringArray &s_array)
{
  SetLength(s_array, 10);
for (int i=0; i<s_array.Length; i++)
  { // Set lengths of second dimensions.(NOTE: non-rectangular)
    SetLength(s_array[i], i+1);
for (int j=0; j<s_array[i].Length; j++)
    /* */ s_array[i][j] = itoa(i*10+j);
  }
}
```

C++ & Delphi System::DynamicArrays

A dynamic array declared in C++ code can be used in Delphi code and vice versa. The following examples illustrate passing a

one-dimensional and two-dimensional array between C++ and Delphi.

C++ System::DynamicArray Example

Delphi System::DynamicArray Example

158.1.6.1 DynamicArray Members

The following tables list the members exposed by DynamicArray.

Protected Methods

	Name	Description
	AllocData (see page 22176)	This is AllocData, a member of class DynamicArray.
	DecRefCount (see page 22177)	This is DecRefCount, a member of class DynamicArray.
	FreeData (see page 22178)	This is FreeData, a member of class DynamicArray.
	IncRefCount (see page 22178)	This is IncRefCount, a member of class DynamicArray.
	SetData (see page 22179)	This is SetData, a member of class DynamicArray.

Public Methods

	Name	Description
	~DynamicArray (see page 22174)	Frees a DynamicArray (see page 22172) object.
	Copy (see page 22176)	This is the overview for the Copy method overload.
	CopyRange (see page 22176)	This is the overview for the CopyRange method overload.
	DimCount (see page 22177)	This is DimCount, a member of class DynamicArray.
	DynamicArray (see page 22175)	This is the overview for the DynamicArray constructor overload.
	get_high (see page 22178)	This is get_high, a member of class DynamicArray.
	get_length (see page 22178)	This is get_length, a member of class DynamicArray.
	get_low (see page 22178)	This is get_low, a member of class DynamicArray.
	set_length (see page 22179)	This is set_length, a member of class DynamicArray.

Public Operators

	Name	Description
	[] (see page 22179)	This is the overview for the [] operator overload.
	= (see page 22180)	Assigns data from another Dynamic Array.
	== (see page 22180)	Compares two Dynamic Arrays for equality.

Public Properties

	Name	Description
R	High (see page 22181)	Indicates the high bounds of the array.
	Length (see page 22181)	Represents the number of elements in the DynamicArray (see page 22172).
R	Low (see page 22182)	Indicates the low bound of a dynamic array.

158.1.6.2 DynamicArray::~DynamicArray Destructor

Frees a DynamicArray (see page 22172) object.

Pascal

```
destructor Destroy;
```

C++

```
~DynamicArray();
```

Description

Don't call the dynamic array destructor. To free a dynamically allocated DynamicArray (see page 22172), assign it a Length (

see page 22181) of zero. (DynamicArray (see page 22172) instances on the stack are freed when they go out of scope).

158.1.6.3 DynamicArray Constructor

This is the overview for the DynamicArray constructor overload.

Overload List

	Name	Description
	DynamicArray::DynamicArray () (see page 22175)	Creates and initializes a System::DynamicArray::DynamicArray (see page 22172) object.
	DynamicArray::DynamicArray (DynamicArray<T>&) (see page 22175)	Creates and initializes a System::DynamicArray::DynamicArray (see page 22172) object.

158.1.6.3.1 DynamicArray::DynamicArray Constructor ()

Creates and initializes a System::DynamicArray::DynamicArray (see page 22172) object.

Pascal

```
constructor Create;
constructor Create(const src: DynamicArray<T>&);
```

C++

```
DynamicArray();
DynamicArray(const DynamicArray<T>& src);
```

Description

When called with no parameters, the constructor creates an empty dynamic array. Otherwise, it copies another System::DynamicArray::DynamicArray (see page 22172).

158.1.6.4 DynamicArray Methods

The methods of the DynamicArray class are listed here.

Protected Methods

	Name	Description
	AllocData (see page 22176)	This is AllocData, a member of class DynamicArray.
	DecRefCount (see page 22177)	This is DecRefCount, a member of class DynamicArray.
	FreeData (see page 22178)	This is FreeData, a member of class DynamicArray.
	IncRefCount (see page 22178)	This is IncRefCount, a member of class DynamicArray.
	SetData (see page 22179)	This is SetData, a member of class DynamicArray.

Public Methods

	Name	Description
	Copy (see page 22176)	This is the overview for the Copy method overload.
	CopyRange (see page 22176)	This is the overview for the CopyRange method overload.
	DimCount (see page 22177)	This is DimCount, a member of class DynamicArray.
	get_high (see page 22178)	This is get_high, a member of class DynamicArray.
	get_length (see page 22178)	This is get_length, a member of class DynamicArray.
	get_low (see page 22178)	This is get_low, a member of class DynamicArray.

	set_length (see page 22179)	This is set_length, a member of class DynamicArray.

158.1.6.4.1 DynamicArray::AllocData Method

Pascal

```
function AllocData(count: Integer): T*; static;
```

C++

```
static T* AllocData(int count);
```

Description

This is AllocData, a member of class DynamicArray.

158.1.6.4.2 Copy Method

This is the overview for the Copy method overload.

Overload List

	Name	Description
	DynamicArray::Copy () (see page 22176)	Copies the contents of a DynamicArray (see page 22172) to the specified DynamicArray (see page 22172).
	DynamicArray::Copy (DynamicArray<T> &) (see page 22176)	Copies the contents of a DynamicArray (see page 22172) to the specified DynamicArray (see page 22172).

158.1.6.4.2.1 DynamicArray::Copy Method ()

Copies the contents of a DynamicArray (see page 22172) to the specified DynamicArray (see page 22172).

Pascal

```
function Copy: DynamicArray<T>;
procedure Copy(var dst: DynamicArray<T>);
```

C++

```
DynamicArray<T> Copy() const;
void Copy(DynamicArray<T> & dst) const;
```

Description

Use System::DynamicArray::Copy to make a copy of a dynamic array. Dynamic arrays are reference counted. Therefore assigning one to another simply copies the reference, not the data. The System::DynamicArray::Copy method, on the other hand, makes a separate copy of the dynamic array. The original contents of the destination array are discarded.

The following code copies the contents of i_array into temp; temp and i_array must be of the same DynamicArray (see page 22172) type.

```
temp = i_array.Copy();
```

See Also

CopyRange

158.1.6.4.3 CopyRange Method

This is the overview for the CopyRange method overload.

Overload List

	Name	Description
	DynamicArray::CopyRange (DynamicArray<T> &, int, int) (see page 22177)	Copies a range of the contents of a DynamicArray (see page 22172) within a specified range.
	DynamicArray::CopyRange (int, int) (see page 22177)	Copies a range of the contents of a DynamicArray (see page 22172) within a specified range.

158.1.6.4.3.1 DynamicArray::CopyRange Method (DynamicArray<T> &, int, int)

Copies a range of the contents of a DynamicArray (see page 22172) within a specified range.

Pascal

```
function CopyRange(startIndex: Integer; count: Integer): DynamicArray<T>;
procedure CopyRange(var dst: DynamicArray<T>; startIndex: Integer; count: Integer);
```

C++

```
DynamicArray<T> CopyRange(int startIndex, int count) const;
void CopyRange(DynamicArray<T> & dst, int startIndex, int count) const;
```

Description

Use the System::DynamicArray::CopyRange method to copy a subrange of a dynamic array. The System::DynamicArray::CopyRange method is similar to the Copy (see page 22176) method except that you can specify the startIndex and the number of items to copy from the source object. The size of the destination array will be truncated or expanded to accommodate what has been copied.

newArray = array.System::DynamicArray::CopyRange(3, 5);

See Also

Copy

158.1.6.4.4 DynamicArray::DecRefCount Method

Pascal

```
procedure DecRefCount;
```

C++

```
void DecRefCount();
```

Description

This is DecRefCount, a member of class DynamicArray.

158.1.6.4.5 DynamicArray::DimCount Method

Pascal

```
function DimCount: Integer; static;
```

C++

```
static int DimCount();
```

Description

This is DimCount, a member of class DynamicArray.

158.1.6.4.6 DynamicArray::FreeData Method

Pascal

```
procedure FreeData;
```

C++

```
void FreeData();
```

Description

This is FreeData, a member of class DynamicArray.

158.1.6.4.7 DynamicArray::get_high Method

Pascal

```
function get_high: Integer;
```

C++

```
int get_high() const;
```

Description

This is get_high, a member of class DynamicArray.

158.1.6.4.8 DynamicArray::get_length Method

Pascal

```
function get_length: Integer;
```

C++

```
int get_length() const;
```

Description

This is get_length, a member of class DynamicArray.

158.1.6.4.9 DynamicArray::get_low Method

Pascal

```
function get_low: Integer;
```

C++

```
int get_low() const;
```

Description

This is get_low, a member of class DynamicArray.

158.1.6.4.10 DynamicArray::IncRefCount Method

Pascal

```
procedure IncRefCount;
```

C++

```
void IncRefCount();
```

Description

This is IncRefCount, a member of class DynamicArray.

158.1.6.4.11 DynamicArray::set_length Method

Pascal

```
procedure set_length(l: Integer);
```

C++

```
void set_length(int l);
```

Description

This is set_length, a member of class DynamicArray.

158.1.6.4.12 DynamicArray::SetData Method

Pascal

```
procedure SetData(var t: T);
```

C++

```
void SetData(T* t);
```

Description

This is SetData, a member of class DynamicArray.

158.1.6.5 DynamicArray Operators

The operators of the DynamicArray class are listed here.

Public Operators

	Name	Description
	[] (see page 22179)	This is the overview for the [] operator overload.
	= (see page 22180)	Assigns data from another Dynamic Array.
	== (see page 22180)	Compares two Dynamic Arrays for equality.

158.1.6.5.1 [] Operator

This is the overview for the [] operator overload.

Overload List

	Name	Description
	DynamicArray::[] (int) (see page 22179)	Provides access to the data held by the dynamic array.
	DynamicArray::[] (int) (see page 22179)	Provides access to the data held by the dynamic array.

158.1.6.5.1.1 DynamicArray::[] Operator (int)

Provides access to the data held by the dynamic array.

Pascal

```
operator [](index: Integer);
operator [](index: Integer);
```

C++

```
T& operator [](int index);
T operator [](int index) const;
```

Description

The const version of the System::DynamicArray (see page 22172)::[] returns a copy of the data at the specified subscript. The non-const version returns a reference to the data, allowing you to modify the data.

cout << i_array[0]=10;

158.1.6.5.2 DynamicArray::= Operator

Assigns data from another Dynamic Array.

Pascal

```
operator =(const src: DynamicArray<T>&);
```

C++

```
DynamicArray& operator =(const DynamicArray<T>& src);
```

Description

Dynamic Arrays are reference counted. When a DynamicArray (see page 22172) is assigned to another, only the reference is assigned (and the reference count adjusted), the contents of the source are not copied. Similarly, when two Dynamic Arrays are compared, only the references are compared, not the contents. To create a separate copy of the contents of a DynamicArray (see page 22172), use the Copy (see page 22176) (or CopyRange (see page 22177)) method.

See Also

Copy

158.1.6.5.3 DynamicArray::== Operator

Compares two Dynamic Arrays for equality.

Pascal

```
operator ==(const other: DynamicArray<T> &);
```

C++

```
bool operator ==(const DynamicArray<T> & other) const;
```

Description

The == System::DynamicArray (see page 22172)::== determines whether two dynamic arrays refer to the same memory. It does not compare the contents of the arrays.

158.1.6.6 DynamicArray Properties

The properties of the DynamicArray class are listed here.

Public Properties

	Name	Description
R	High (see page 22181)	Indicates the high bounds of the array.
	Length (see page 22181)	Represents the number of elements in the DynamicArray (see page 22172).
R	Low (see page 22182)	Indicates the low bound of a dynamic array.

158.1.6.6.1 DynamicArray::High Property

Indicates the high bounds of the array.

Pascal

```
property High: Integer;
```

C++

```
__property int High;
```

Description

Reading the High property returns Length (see page 22181)-1.

The following function returns the sum of the contents of an integer type DynamicArray (see page 22172).

int TotalArray(const DynamicArray (see page 22172)<int>& arrayOfInt)

```
{
int total=0;
for (int i=arrayOfInt.Low; i<=arrayOfInt.High; i++)
    total += arrayOfInt[i];
return total;
}
```

See Also

Length

Low

158.1.6.6.2 DynamicArray::Length Property

Represents the number of elements in the DynamicArray (see page 22172).

Pascal

```
property Length: Integer;
```

C++

```
__property int Length;
```

Description

Use the Length property to set or get the length of the dynamic array. The following code sets the length of a DynamicArray (see page 22172) to 10.

DynamicArray (see page 22172)<int> arrayOfInt;

```
arrayOfInt.Length = 10;
cout << "ArrayLength: " << arrayOfInt.Length << endl;
```

SetLength (see page 22522)(arrayOfInt, 10);

To free a dynamic array, simply set its Length to 0:

arrayOfInt.Length = 0;

See Also

High

158.1.6.6.3 DynamicArray::Low Property

Indicates the low bound of a dynamic array.

Pascal

```
property Low: Integer;
```

C++

```
__property int Low;
```

Description

The Low property always returns 0. The following function returns the sum of the contents of an integer type DynamicArray (see page 22172).

int TotalArray(const DynamicArray (see page 22172)<int>& arrayOfInt)

```
{
int total=0;
for (int i=arrayOfInt.Low; i<=arrayOfInt.High; i++)
    total += arrayOfInt[i];
return total;
}
```

See Also

High

158.1.7 DynArrayException Class

Class Hierarchy

System::DynArrayException

Pascal

```
DynArrayException = class;
```

C++

```
class DynArrayException;
```

File

sysdyn.h

Description

This is class System::DynArrayException.

158.1.8 DynArrayNullData Class

Class Hierarchy

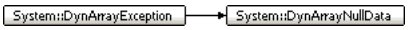

Pascal

```
DynArrayNullData = class(DynArrayException);
```

C++

```
class DynArrayNullData : public DynArrayException;
```

File

sysdyn.h

Description

This is class System::DynArrayNullData.

158.1.9 DynArrayOutOfRange Class

Class Hierarchy

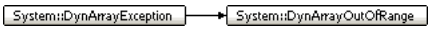

Pascal

```
DynArrayOutOfRange = class(DynArrayException);
```

C++

```
class DynArrayOutOfRange : public DynArrayException;
```

File

sysdyn.h

Description

This is class System::DynArrayOutOfRange.

158.1.9.1 DynArrayOutOfRange Members

The following tables list the members exposed by DynArrayOutOfRange.

Public Data Members

	Name	Description
	m_Count (see page 22184)	This is m_Count, a member of class DynArrayOutOfRange.
	m_Index (see page 22184)	This is m_Index, a member of class DynArrayOutOfRange.

Public Methods

	Name	Description
	DynArrayOutOfRange (see page 22184)	This is DynArrayOutOfRange, a member of class DynArrayOutOfRange.

158.1.9.2 DynArrayOutOfRange::DynArrayOutOfRange Constructor

Pascal

```
constructor Create(index: Integer; count: Integer);
```

C++

```
DynArrayOutOfRange(int index, int count);
```

Description

This is DynArrayOutOfRange, a member of class DynArrayOutOfRange.

158.1.9.3 DynArrayOutOfRange Data Members

The data members of the DynArrayOutOfRange class are listed here.

Public Data Members

	Name	Description
	m_Count (see page 22184)	This is m_Count, a member of class DynArrayOutOfRange.
	m_Index (see page 22184)	This is m_Index, a member of class DynArrayOutOfRange.

158.1.9.3.1 DynArrayOutOfRange::m_Count Data Member

Pascal

```
m_Count: Integer;
```

C++

```
int m_Count;
```

Description

This is m_Count, a member of class DynArrayOutOfRange.

158.1.9.3.2 DynArrayOutOfRange::m_Index Data Member

Pascal

```
m_Index: Integer;
```

C++

```
int m_Index;
```

Description

This is m_Index, a member of class DynArrayOutOfRange.

158.1.10 Function Class

System::Function represents a method on an Automation interface that returns a value.

Class Hierarchy

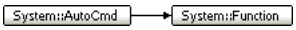

Pascal

```
Function = class(AutoCmd);
```

C++

```
class Function : public AutoCmd;
```

File

sysvari.h

Description

The System::Function class represents a single method on an Automation interface that is assigned as the value of a Variant (see page 22307). It can be executed using the Variant (see page 22307)'s Exec method.

Each System::Function instance is assigned a name, which is the same as the name of the corresponding method on the interface. In addition, it can be assigned parameters that correspond to the parameters of the method. These parameters can be stored by position or by name.

See Also

System::Variant (see page 22307)

System::Procedure (see page 22230)

PropertyGet

PropertySet

158.1.10.1 Function Members

The following tables list the members exposed by Function.

Protected Data Members

	Name	Description
	Name (see page 22130)	This is Name, a member of class AutoCmd.
	NamedParmNames (see page 22130)	This is NamedParmNames, a member of class AutoCmd.
	NamedParms (see page 22130)	This is NamedParms, a member of class AutoCmd.
	NamedParmTypes (see page 22131)	This is NamedParmTypes, a member of class AutoCmd.
	Parms (see page 22131)	This is Parms, a member of class AutoCmd.
	ParmTypes (see page 22131)	This is ParmTypes, a member of class AutoCmd.

Protected Methods

	Name	Description
	AddElement (see page 22132)	This is AddElement, a member of class AutoCmd.

Public Methods

	Name	Description
	~AutoCmd (see page 22129)	Frees an System::AutoCmd::AutoCmd (see page 22129) object.
	AutoCmd (see page 22129)	This is the overview for the AutoCmd constructor overload.
	Clear (see page 22132)	System::AutoCmd::Clears any assigned parameters and blanks the name of the associated method.
	ClearArgs (see page 22132)	Clears any assigned parameters.
	ClearName (see page 22133)	Blanks the name of the associated method.
	Function (see page 22186)	This is the overview for the Function constructor overload.
	GetArgCount (see page 22133)	Returns the number of positional arguments that have been assigned.
	GetArgType (see page 22134)	Returns the type of a parameter value.

	GetCallType (see page 22187)	Indicates the type of the interface method represented by the System::Function::Function (see page 22186) object.
	GetName (see page 22134)	Returns the name of the interface method this System::AutoCmd::AutoCmd (see page 22129) object represents.
	GetNamedArgCount (see page 22135)	Returns the number of named arguments that have been assigned.
	GetNamedArgType (see page 22135)	Returns the type of a named parameter value.
	GetNamedParm (see page 22136)	Returns the value of a named parameter.
	GetNamedParmName (see page 22136)	Returns the named of a named parameter value.
	GetParm (see page 22137)	Returns the value of a positional parameter.
	RequestResult (see page 22187)	Indicates that the interface method returns a value.
	SetName (see page 22138)	Sets the name of the method the System::AutoCmd::AutoCmd (see page 22129) object represents.

Public Operators

	Name	Description
	<< (see page 22138)	This is the overview for the << operator overload.

158.1.10.2 Function Constructor

This is the overview for the Function constructor overload.

Overload List

	Name	Description
	Function::Function (AutoCmd&) (see page 22186)	Creates an instance of System::Function::Create for a specified method.
	Function::Function (String&) (see page 22186)	Creates an instance of System::Function::Create for a specified method.

158.1.10.2.1 Function::Function Constructor (AutoCmd&)

Creates an instance of System::Function::Create for a specified method.

Pascal

```
constructor Create(const name: String&);
constructor Create(const src: AutoCmd&);
```

C++

```
Function(const String& name);
Function(const AutoCmd& src);
```

Description

System::Function::Create has two constructors: a version that takes the name of the method it represents, and a copy constructor. When creating a new System::Function::Create object for a method, the name parameter must be the same as the name of the method as it appears in the interface declaration.

158.1.10.3 Function Methods

The methods of the Function class are listed here.

Public Methods

	Name	Description
	GetCallType (see page 22187)	Indicates the type of the interface method represented by the System::Function::Function (see page 22186) object.
	RequestResult (see page 22187)	Indicates that the interface method returns a value.

158.1.10.3.1 Function::GetCallType Method

Indicates the type of the interface method represented by the System::Function::Function (see page 22186) object.

Pascal

```
function GetCallType: Byte;
```

C++

```
Byte GetCallType() const;
```

Description

Call System::Function::GetCallType to determine the type of flag needed to call the method represented by System::Function::Function (see page 22186) using the IDispatch::Invoke method of the Automation interface. System::Function::GetCallType returns DISPATCH_METHOD, indicating a method that is not a property access method.

158.1.10.3.2 Function::RequestResult Method

Indicates that the interface method returns a value.

Pascal

```
function RequestResult: bool;
```

C++

```
bool RequestResult() const;
```

Description

System::Function::RequestResult always returns true, indicating that there is a return value after executing the System::Function::Function (see page 22186) object using the Variant::Exec method.

See Also

System::Function::GetCallType (see page 22187)

System::Variant::Exec (see page 22320)

158.1.11 NamedParm Class

NamedParm represents a named parameter assigned to a TAutoCmd descendant.

Class Hierarchy

System::NamedParm

Pascal

```
NamedParm = class;
```

C++

```
class NamedParm;
```

File

sysvari.h

Description

The NamedParm class represents a named parameter that can be assigned as an argument to one of the AutoCmd (see page 22127) descendants (PropertyGet (see page 22233), PropertySet (see page 22236), Procedure (see page 22230), or Function (see page 22184)).

NamedParm overloads the constructor to let you assign a variety of types as the value of the parameter. It also defines methods for retrieving the name, value, or type of the parameter.

See Also

PropertyGet

System::Function (see page 22184)

System::Procedure (see page 22230)

PropertySet

System::AutoCmd (see page 22127)

158.1.11.1 NamedParm Members

The following tables list the members exposed by NamedParm.

Protected Data Members

	Name	Description
	Name (see page 22191)	This is Name, a member of class NamedParm.
	Parm (see page 22191)	This is Parm, a member of class NamedParm.
	Type (see page 22192)	This is Type, a member of class NamedParm.

Protected Methods

	Name	Description
	AddElement (see page 22192)	This is AddElement, a member of class NamedParm.

Public Methods

	Name	Description
	GetParm (see page 22192)	Returns the value of the parameter.
	GetParmName (see page 22193)	Returns the name of the parameter.
	GetType (see page 22193)	Returns the type of the parameter's value.
	NamedParm (see page 22188)	This is the overview for the NamedParm constructor overload.

158.1.11.2 NamedParm Constructor

This is the overview for the NamedParm constructor overload.

Overload List

	Name	Description
	NamedParm::NamedParm (String&, AnsiString&) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, bool) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, Byte) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, Byte*) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, char*) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, Currency) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, Currency*) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, double) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, double*) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, float) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, float*) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, IDispatch* const) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, IInterface* const) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, int) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, int*) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, IUnknown* const) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, short) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, short*) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, String*) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, TDateTime) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, TDateTime*) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, Variant&) (see page 22190)	Creates an instance of Create for a specified method.

	NamedParm::NamedParm (String&, Variant*) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, wchar_t*) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, wchar_t**) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, WordBool) (see page 22190)	Creates an instance of Create for a specified method.
	NamedParm::NamedParm (String&, WordBool*) (see page 22190)	Creates an instance of Create for a specified method.

158.1.11.2.1 NamedParm::NamedParm Constructor (String&, AnsiString&)

Creates an instance of Create for a specified method.

Pascal

```
constructor Create(const name: String&; const arg: Variant&);
constructor Create(const name: String&; const arg: short);
constructor Create(const name: String&; const arg: Integer);
constructor Create(const name: String&; const arg: float);
constructor Create(const name: String&; const arg: double);
constructor Create(const name: String&; const arg: Currency);
constructor Create(const name: String&; const arg: TDateTime);
constructor Create(const name: String&; const arg: bool);
constructor Create(const name: String&; const arg: WordBool);
constructor Create(const name: String&; const arg: Byte);
constructor Create(const name: String&; const arg: AnsiString&);
constructor Create(const name: String&; const arg: char*);
constructor Create(const name: String&; var arg: wchar_t);
constructor Create(const name: String&; arg: IDispatch* const);
constructor Create(const name: String&; arg: IUnknown* const);
constructor Create(const name: String&; arg: IInterface* const);
constructor Create(const name: String&; var arg: Variant);
constructor Create(const name: String&; var arg: short);
constructor Create(const name: String&; var arg: Integer);
constructor Create(const name: String&; var arg: float);
constructor Create(const name: String&; var arg: double);
constructor Create(const name: String&; var arg: Currency);
constructor Create(const name: String&; var arg: TDateTime);
constructor Create(const name: String&; var arg: WordBool);
constructor Create(const name: String&; var arg: Byte);
constructor Create(const name: String&; var arg: wchar_t*);
constructor Create(const name: String&; var arg: String);
```

C++

```
NamedParm(const String& name, const Variant& arg);
NamedParm(const String& name, const short arg);
NamedParm(const String& name, const int arg);
NamedParm(const String& name, const float arg);
NamedParm(const String& name, const double arg);
NamedParm(const String& name, const Currency arg);
NamedParm(const String& name, const TDateTime arg);
NamedParm(const String& name, const bool arg);
NamedParm(const String& name, const WordBool arg);
NamedParm(const String& name, const Byte arg);
NamedParm(const String& name, const AnsiString& arg);
NamedParm(const String& name, const char* arg);
NamedParm(const String& name, wchar_t* arg);
NamedParm(const String& name, IDispatch* const arg);
```

```
NamedParm(const String& name, IUnknown* const arg);
NamedParm(const String& name, IInterface* const arg);
NamedParm(const String& name, Variant* arg);
NamedParm(const String& name, short* arg);
NamedParm(const String& name, int* arg);
NamedParm(const String& name, float* arg);
NamedParm(const String& name, double* arg);
NamedParm(const String& name, Currency* arg);
NamedParm(const String& name, TDateTime* arg);
NamedParm(const String& name, WordBool* arg);
NamedParm(const String& name, Byte* arg);
NamedParm(const String& name, wchar_t** arg);
NamedParm(const String& name, String* arg);
```

Description

Pass the Create constructor two arguments: the name of the parameter it represents (name), and the value to assign to that parameter (arg).

158.1.11.3 NamedParm Data Members

The data members of the NamedParm class are listed here.

Protected Data Members

	Name	Description
	Name (see page 22191)	This is Name, a member of class NamedParm.
	Parm (see page 22191)	This is Parm, a member of class NamedParm.
	Type (see page 22192)	This is Type, a member of class NamedParm.

158.1.11.3.1 NamedParm::Name Data Member

Pascal

```
Name: String;
```

C++

```
String Name;
```

Description

This is Name, a member of class NamedParm.

158.1.11.3.2 NamedParm::Parm Data Member

Pascal

```
Parm: Variant;
```

C++

```
Variant Parm;
```

Description

This is Parm, a member of class NamedParm.

158.1.11.3.3 NamedParm::Type Data Member

Pascal

```
Type: Integer;
```

C++

```
int Type;
```

Description

This is Type, a member of class NamedParm.

158.1.11.4 NamedParm Methods

The methods of the NamedParm class are listed here.

Protected Methods

	Name	Description
	AddElement (see page 22192)	This is AddElement, a member of class NamedParm.

Public Methods

	Name	Description
	GetParm (see page 22192)	Returns the value of the parameter.
	GetParmName (see page 22193)	Returns the name of the parameter.
	GetType (see page 22193)	Returns the type of the parameter's value.

158.1.11.4.1 NamedParm::AddElement Method

Pascal

```
procedure AddElement;
```

C++

```
void AddElement();
```

Description

This is AddElement, a member of class NamedParm.

158.1.11.4.2 NamedParm::GetParm Method

Returns the value of the parameter.

Pascal

```
function GetParm: Variant;
```

C++

```
Variant GetParm() const;
```

Description

Call GetParm to obtain the value that was assigned to the named parameter by the constructor. The value is always returned as a Variant (see page 22307). To determine the underlying type of this value, use the GetType (see page 22193) method.

See Also

namedParm

GetType

158.1.11.4.3 NamedParm::GetParmName Method

Returns the name of the parameter.

Pascal

```
function GetParmName: String;
```

C++

```
String GetParmName() const;
```

Description

GetParmName returns the name that was assigned to the parameter by the constructor.

See Also

GetParm

NamedParm

158.1.11.4.4 NamedParm::GetType Method

Returns the type of the parameter's value.

Pascal

```
function GetType: Integer;
```

C++

```
int GetType() const;
```

Description

Call GetType to determine the type of the parameter's value. GetType returns one of the constants declared in the System (see page 22079) unit to represent the types of Variant (see page 22307) values. As such, it returns the value you would obtain by calling VarType on the Variant (see page 22307) returned by GetParm (see page 22192).

Note: A list of the possible return values is included in the topic for VarType.

See Also

GetParm

158.1.12 OleVariant Template

System::OleVariant is a C++ implementation of the Delphi intrinsic type System::OleVariant.

Class Hierarchy

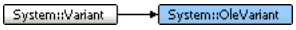

Pascal

```
OleVariant = class(Variant);
```

C++

```
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
```

```
class P9, class P10>
class OleVariant : protected Variant;
```

File

sysvari.h

Description

System::OleVariant is a special descendant of the Variant (see page 22307) class that is used to represent COM interfaces or data that is passed over a COM interface. System::OleVariant inherits from Variant (see page 22307) as a protected class so that it can prevent operations on AnsiString (see page 22093) objects, which are not compatible with COM.

The syntax for using System::OleVariant is the same as that for using the Variant (see page 22307) class. Like the Variant (see page 22307) class, System::OleVariant has the following characteristics:

System::OleVariants can contain integer values, real values, string values, boolean values, date-and-time values, and Automation objects. In addition, System::OleVariants can contain arrays of varying size and dimension with elements of any of these types.

The special System::OleVariant value Unassigned is used to indicate that a System::OleVariant has not yet been assigned a value, and the special System::OleVariant value Null is used to indicate unknown or missing data.

A System::OleVariant can be combined with other System::OleVariants and it can be constructed from any of the following data types. The compiler automatically performs the necessary type conversions.

shortWordBool

intByte

floatVariant

doubleWideString

Currencywchar_t*

TDateTimeOle2::IDispatch (see page 22394)*

boolOle2::IUnknown (see page 22586)*

When an System::OleVariant contains an Automation object, the System::OleVariant can be used to get and set properties of the object, and to invoke methods on the object.

System::OleVariant variables are always initialized to be Unassigned when they are first created. This is true whether a System::OleVariant variable is global, local, or part of a structure such as an array.

Note that while System::OleVariants offer great flexibility, they also consume more memory than regular variables, and operations on System::OleVariants are substantially slower than operations on statically typed values.

Note: Use only the operators declared within System::OleVariant. The compiler will ignore any operators you overload yourself.

Note: For binary operators, if one operand is of type System::OleVariant, the other operand is automatically converted to type System::OleVariant.

See Also

System::Variant (see page 22307)

TVariantT

158.1.12.1 OleVariant Members

The following tables list the members exposed by OleVariant.

Public Friends

Name	Description
class AutoCmd (see page 22375)	This is friend class AutoCmd.

Public Methods

	Name	Description
	~OleVariant (see page 22197)	Frees an instance of System::OleVariant::OleVariant (see page 22200).
	~Variant (see page 22311)	Frees a System::Variant::Variant (see page 22314) object.
	ArrayDimCount (see page 22317)	Returns the number of dimensions of the given variant.
	ArrayHighBound (see page 22317)	Returns the high bound of the given dimension in the given variant array.
	ArrayLock (see page 22317)	Locks the given variant array and returns a pointer to the data stored in the array.
	ArrayLowBound (see page 22318)	Returns the low bound of the given dimension in the given variant array.
	ArrayRedim (see page 22318)	Resizes the given variant array by changing the high bound of the rightmost dimension to the given value.
	ArrayUnlock (see page 22318)	Unlocks a variant array that was previously locked by ArrayLock (see page 22317).
	Assign (see page 22318)	This is Assign, a member of class Variant.
	AsType (see page 22319)	Casts a System::Variant::Variant (see page 22314) object to the type specified by the argument VarType.
	ChangeType (see page 22319)	Changes the data type of a System::Variant::Variant (see page 22314) object once it has been instantiated.
	Clear (see page 22319)	System::Variant::Clears the value of the System::Variant::Variant (see page 22314).
	CreateObject (see page 22320)	Creates an OLE Automation object with the programmatic identifier specified by ProgID and returns it as a System::Variant::Variant (see page 22314).
	Empty (see page 22320)	This is Empty, a member of class Variant.
	Exec (see page 22320)	System::Variant::Executes an OLE procedure, function, or a property Get or Set method.
	GetActiveObject (see page 22321)	Retrieves a reference to an IDispatch (see page 22394) interface of a currently running, registered OLE Automation object, and returns it as a System::Variant::Variant (see page 22314).
	GetBaseVariant (see page 22321)	Returns the Variant (see page 22307) that is assigned as the value of this Variant (see page 22307) when Type (see page 22341) is varVariant (see page 22690).
	GetElement (see page 22322)	This is the overview for the GetElement method overload.
	GetError (see page 22322)	Converts the System::Variant::Variant (see page 22314) to a varError (see page 22684) type and returns the integer value.
	IsArray (see page 22323)	Queries the VType of the base class TVarData (see page 22571) for the varArray (see page 22679) bit.
	IsEmpty (see page 22323)	Queries the VType of the base class TVarData (see page 22571).
	IsNull (see page 22323)	Queries the VType of the base class TVarData (see page 22571) for the varNull (see page 22686) bit.
	NoParam (see page 22324)	This is NoParam, a member of class Variant.
	OleFunction (see page 22324)	This is the overview for the OleFunction method overload.
	OleProcedure (see page 22330)	This is the overview for the OleProcedure method overload.
	OlePropertyGet (see page 22336)	This is the overview for the OlePropertyGet method overload.
	OlePropertySet (see page 22338)	This is the overview for the OlePropertySet method overload.
	OleVariant (see page 22198)	This is the overview for the OleVariant constructor overload.
	PutElement (see page 22340)	This is the overview for the PutElement method overload.
	Sanitized (see page 22340)	This is Sanitized, a member of class Variant.
	SetError (see page 22341)	Converts the System::Variant::Variant (see page 22314) to a varError (see page 22684) type and assigns the integer err as the value of the varError (see page 22684).
	Type (see page 22341)	This is Type, a member of class Variant.

	Variant (see page 22312)	This is the overview for the Variant constructor overload.

Public Operators

	Name	Description
	- (see page 22343)	This is the overview for the - operator overload.
	! (see page 22344)	Returns a new System::Variant::Variant (see page 22314) that is the logical negation of this System::Variant::Variant (see page 22314).
	!= (see page 22209)	This is the overview for the != operator overload.
	!= (see page 22344)	This is the overview for the != operator overload.
	% (see page 22345)	This is the overview for the % operator overload.
	%= (see page 22346)	Returns this System::Variant::Variant (see page 22314) as the remainder from dividing its original value and the System::Variant::Variant (see page 22314) specified by rhs.
	& (see page 22346)	This is the overview for the & operator overload.
	&= (see page 22347)	Returns this System::Variant::Variant (see page 22314) after performing a bitwise and operation.
	* (see page 22347)	This is the overview for the * operator overload.
	*= (see page 22348)	Multiplies this System::Variant::Variant (see page 22314) by the System::Variant::Variant (see page 22314) specified by rhs.
	/ (see page 22348)	This is the overview for the / operator overload.
	/= (see page 22349)	Divides this System::Variant::Variant (see page 22314) by the System::Variant::Variant (see page 22314) rhs.
	^ (see page 22349)	This is the overview for the ^ operator overload.
	^= (see page 22350)	Returns this System::Variant::Variant (see page 22314) after performing a bitwise xor operation.
	__int64 (see page 22350)	This is __int64, a member of class Variant.
	| (see page 22351)	This is the overview for the | operator overload.
	|= (see page 22351)	Returns this System::Variant::Variant (see page 22314) after performing a bitwise or operation.
	+ (see page 22352)	This is the overview for the + operator overload.
	+= (see page 22352)	Adds the System::Variant::Variant (see page 22314) specified by rhs to this System::Variant::Variant (see page 22314) object.
	< (see page 22210)	This is the overview for the < operator overload.
	< (see page 22353)	This is the overview for the < operator overload.
	<< (see page 22353)	This is the overview for the << operator overload.
	<<= (see page 22354)	Performs a bitwise shl (left shift) operation on this System::Variant::Variant (see page 22314).
	<= (see page 22211)	This is the overview for the <= operator overload.
	<= (see page 22355)	This is the overview for the <= operator overload.
	-= (see page 22361)	Subtracts the System::Variant::Variant (see page 22314) specified by rhs from this System::Variant::Variant (see page 22314) object.
	= (see page 22211)	This is the overview for the = operator overload.
	= (see page 22355)	This is the overview for the = operator overload.
	== (see page 22214)	This is the overview for the == operator overload.
	== (see page 22361)	This is the overview for the == operator overload.
	> (see page 22214)	This is the overview for the > operator overload.
	> (see page 22362)	This is the overview for the > operator overload.
	>= (see page 22215)	This is the overview for the >= operator overload.
	>= (see page 22363)	This is the overview for the >= operator overload.
	>> (see page 22363)	This is the overview for the >> operator overload.
	>>= (see page 22365)	Performs a bitwise shr (right shift) operation on this System::Variant::Variant (see page 22314).
	AnsiString (see page 22365)	Converts the System::Variant::Variant (see page 22314) to an AnsiString object.
	bool (see page 22365)	Converts the System::Variant::Variant (see page 22314) to a boolean value.
	char (see page 22365)	This is char, a member of class Variant.
	Currency (see page 22366)	Converts the System::Variant::Variant (see page 22314) to a Currency object.
	CURRENCY (see page 22366)	This is CURRENCY, a member of class Variant.

	Currency* (see page 22366)	This is Currency*, a member of class Variant.
	DECIMAL (see page 22366)	This is DECIMAL, a member of class Variant.
	double (see page 22367)	Converts the System::Variant::Variant (see page 22314) to a double value.
	double* (see page 22367)	This is double*, a member of class Variant.
	float (see page 22367)	Converts the System::Variant::Variant (see page 22314) to a float value.
	float* (see page 22368)	This is float*, a member of class Variant.
	IDispatch* (see page 22368)	This is IDispatch*, a member of class Variant.
	IInterface* (see page 22368)	This is IInterface*, a member of class Variant.
	int (see page 22368)	Converts the System::Variant::Variant (see page 22314) to an integer value.
	int* (see page 22369)	This is int*, a member of class Variant.
	IUnknown* (see page 22369)	This is IUnknown*, a member of class Variant.
	long (see page 22369)	This is long, a member of class Variant.
	long* (see page 22369)	This is long*, a member of class Variant.
	LPSAFEARRAY (see page 22369)	This is LPSAFEARRAY, a member of class Variant.
	short (see page 22370)	Converts the System::Variant::Variant (see page 22314) to a short value.
	short* (see page 22370)	This is short*, a member of class Variant.
	signed char (see page 22370)	This is signed char, a member of class Variant.
	signed char* (see page 22370)	This is signed char*, a member of class Variant.
	TDateTime (see page 22371)	Converts the System::Variant::Variant (see page 22314) to a TDateTime object.
	TDateTime* (see page 22371)	This is TDateTime*, a member of class Variant.
	TVariant (see page 22371)	This is TVariant, a member of class Variant.
	TVariant* (see page 22371)	This is TVariant*, a member of class Variant.
	unsigned char (see page 22372)	This is unsigned char, a member of class Variant.
	unsigned char* (see page 22372)	This is unsigned char*, a member of class Variant.
	unsigned int (see page 22372)	This is unsigned int, a member of class Variant.
	unsigned int* (see page 22372)	This is unsigned int*, a member of class Variant.
	unsigned long (see page 22373)	This is unsigned long, a member of class Variant.
	unsigned long* (see page 22373)	This is unsigned long*, a member of class Variant.
	unsigned short (see page 22373)	This is unsigned short, a member of class Variant.
	unsigned short* (see page 22373)	This is unsigned short*, a member of class Variant.
	VARIANT (see page 22373)	This is VARIANT, a member of class Variant.
	VARIANT* (see page 22374)	This is VARIANT*, a member of class Variant.
	wchar_t* (see page 22374)	Converts the System::Variant::Variant (see page 22314) to the specified type and sets the varByRef (see page 22680) bit, meaning that the variant contains a reference as opposed to a value.
	wchar_t** (see page 22374)	This is wchar_t**, a member of class Variant.
	WideString (see page 22374)	This is WideString, a member of class Variant.

158.1.12.2 OleVariant::~OleVariant Destructor

Frees an instance of System::OleVariant::OleVariant (see page 22200).

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~OleVariant();
```

Description

~System::OleVariant::OleVariant (see page 22200) is the destructor for System::OleVariant::OleVariant (see page 22200).

158.1.12.3 OleVariant Constructor

This is the overview for the OleVariant constructor overload.

Overload List

	Name	Description
	OleVariant::OleVariant () (see page 22200)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (__int64) (see page 22204)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (__int64*) (see page 22204)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (AnsiString&) (see page 22200)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (bool) (see page 22204)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (char) (see page 22204)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (char*) (see page 22204)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (Currency) (see page 22200)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (CURRENCY&) (see page 22200)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (Currency*) (see page 22201)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (CURRENCY*) (see page 22200)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (double) (see page 22205)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (double*) (see page 22205)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (float) (see page 22205)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (float*) (see page 22205)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (IDispatch* const) (see page 22201)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (IInterface* const) (see page 22201)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (int) (see page 22206)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (int*) (see page 22206)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (int*, int, Word) (see page 22206)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (IUnknown* const) (see page 22201)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (long*) (see page 22206)	This is OleVariant, a member of class OleVariant.

	OleVariant::OleVariant (OleVariant&) (see page 22202)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (OleVariant*, int) (see page 22202)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (short) (see page 22206)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (short*) (see page 22207)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (signed char*) (see page 22207)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (TDateTime) (see page 22202)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (TDateTime*) (see page 22202)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (TVariant&) (see page 22202)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (unsigned char) (see page 22207)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (unsigned char*) (see page 22207)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (unsigned int) (see page 22208)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (unsigned int*) (see page 22208)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (unsigned long*) (see page 22208)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (unsigned short) (see page 22208)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (unsigned short*) (see page 22208)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (Variant&) (see page 22203)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (VARIANT&) (see page 22203)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (VARIANT*) (see page 22203)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (wchar_t* const, bool) (see page 22209)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (wchar_t**) (see page 22209)	This is OleVariant, a member of class OleVariant.
	OleVariant::OleVariant (WideString&) (see page 22203)	This is OleVariant, a member of class OleVariant.

158.1.12.3.1 OleVariant::OleVariant Constructor ()

Pascal

```
constructor Create;
```

C++

```
__fastcall OleVariant();
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.2 OleVariant::OleVariant Constructor (AnsiString&)

Pascal

```
constructor Create(const src: AnsiString&);
```

C++

```
__fastcall OleVariant(const AnsiString& src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.3 OleVariant::OleVariant Constructor (CURRENCY&)

Pascal

```
constructor Create(const src: CURRENCY&);
```

C++

```
__fastcall OleVariant(const CURRENCY& src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.4 OleVariant::OleVariant Constructor (CURRENCY*)

Pascal

```
constructor Create(var src: CURRENCY);
```

C++

```
__fastcall OleVariant(CURRENCY* src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.5 OleVariant::OleVariant Constructor (Currency)

Pascal

```
constructor Create(const src: Currency);
```

C++

```
__fastcall OleVariant(const Currency src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.6 OleVariant::OleVariant Constructor (Currency*)

Pascal

```
constructor Create(var src: Currency);
```

C++

```
__fastcall OleVariant(Currency* src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.7 OleVariant::OleVariant Constructor (IDispatch* const)

Pascal

```
constructor Create(src: IDispatch* const);
```

C++

```
__fastcall OleVariant(IDispatch* const src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.8 OleVariant::OleVariant Constructor (IInterface* const)

Pascal

```
constructor Create(src: IInterface* const);
```

C++

```
__fastcall OleVariant(IInterface* const src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.9 OleVariant::OleVariant Constructor (IUnknown* const)

Pascal

```
constructor Create(src: IUnknown* const);
```

C++

```
__fastcall OleVariant(IUnknown* const src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.10 OleVariant::OleVariant Constructor (OleVariant&)

Pascal

```
constructor Create(const src: OleVariant&);
```

C++

```
__fastcall OleVariant(const OleVariant& src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.11 OleVariant::OleVariant Constructor (OleVariant*, int)

Pascal

```
constructor Create(const values: OleVariant*; const valuesSize: Integer);
```

C++

```
__fastcall OleVariant(const OleVariant* values, const int valuesSize);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.12 OleVariant::OleVariant Constructor (TDateTime)

Pascal

```
constructor Create(const src: TDateTime);
```

C++

```
__fastcall OleVariant(const TDateTime src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.13 OleVariant::OleVariant Constructor (TDateTime*)

Pascal

```
constructor Create(var src: TDateTime);
```

C++

```
__fastcall OleVariant(TDateTime* src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.14 OleVariant::OleVariant Constructor (TVariant&)

Pascal

```
constructor Create(const src: TVariant&);
```

C++

```
__fastcall OleVariant(const TVariant& src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.15 OleVariant::OleVariant Constructor (VARIANT&)

Pascal

```
constructor Create(const src: VARIANT&);
```

C++

```
__fastcall OleVariant(const VARIANT& src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.16 OleVariant::OleVariant Constructor (VARIANT*)

Pascal

```
constructor Create(var src: VARIANT);
```

C++

```
__fastcall OleVariant(VARIANT* src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.17 OleVariant::OleVariant Constructor (Variant&)

Pascal

```
constructor Create(const src: Variant&);
```

C++

```
__fastcall OleVariant(const Variant& src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.18 OleVariant::OleVariant Constructor (WideString&)

Pascal

```
constructor Create(const src: WideString&);
```

C++

```
__fastcall OleVariant(const WideString& src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.19 OleVariant::OleVariant Constructor (__int64)

Pascal

```
constructor Create(const src: __int64);
```

C++

```
__fastcall OleVariant(const __int64 src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.20 OleVariant::OleVariant Constructor (__int64*)

Pascal

```
constructor Create(var src: __int64);
```

C++

```
__fastcall OleVariant(__int64* src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.21 OleVariant::OleVariant Constructor (bool)

Pascal

```
constructor Create(const src: bool);
```

C++

```
__fastcall OleVariant(const bool src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.22 OleVariant::OleVariant Constructor (char)

Pascal

```
constructor Create(const src: char);
```

C++

```
__fastcall OleVariant(const char src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.23 OleVariant::OleVariant Constructor (char*)

Pascal

```
constructor Create(const src: char*);
```

C++

```
__fastcall OleVariant(const char* src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.24 OleVariant::OleVariant Constructor (double)

Pascal

```
constructor Create(const src: double);
```

C++

```
__fastcall OleVariant(const double src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.25 OleVariant::OleVariant Constructor (double*)

Pascal

```
constructor Create(var src: double);
```

C++

```
__fastcall OleVariant(double* src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.26 OleVariant::OleVariant Constructor (float)

Pascal

```
constructor Create(const src: float);
```

C++

```
__fastcall OleVariant(const float src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.27 OleVariant::OleVariant Constructor (float*)

Pascal

```
constructor Create(var src: float);
```

C++

```
__fastcall OleVariant(float* src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.28 OleVariant::OleVariant Constructor (int)

Pascal

```
constructor Create(const src: Integer);
```

C++

```
__fastcall OleVariant(const int src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.29 OleVariant::OleVariant Constructor (int*)

Pascal

```
constructor Create(var src: Integer);
```

C++

```
__fastcall OleVariant(int* src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.30 OleVariant::OleVariant Constructor (int*, int, Word)

Pascal

```
constructor Create(const bounds: int*; const boundsSize: Integer; varType: Word);
```

C++

```
__fastcall OleVariant(const int* bounds, const int boundsSize, Word varType);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.31 OleVariant::OleVariant Constructor (long*)

Pascal

```
constructor Create(var src: long);
```

C++

```
__fastcall OleVariant(long* src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.32 OleVariant::OleVariant Constructor (short)

Pascal

```
constructor Create(const src: short);
```

C++

```
__fastcall OleVariant(const short src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.33 OleVariant::OleVariant Constructor (short*)

Pascal

```
constructor Create(var src: short);
```

C++

```
__fastcall OleVariant(short* src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.34 OleVariant::OleVariant Constructor (signed char*)

Pascal

```
constructor Create(var src: signed char);
```

C++

```
__fastcall OleVariant(signed char* src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.35 OleVariant::OleVariant Constructor (unsigned char)

Pascal

```
constructor Create(const src: unsigned char);
```

C++

```
__fastcall OleVariant(const unsigned char src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.36 OleVariant::OleVariant Constructor (unsigned char*)

Pascal

```
constructor Create(var src: unsigned char);
```

C++

```
__fastcall OleVariant(unsigned char* src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.37 OleVariant::OleVariant Constructor (unsigned int)

Pascal

```
constructor Create(const src: unsigned int);
```

C++

```
__fastcall OleVariant(const unsigned int src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.38 OleVariant::OleVariant Constructor (unsigned int*)

Pascal

```
constructor Create(var src: unsigned int);
```

C++

```
__fastcall OleVariant(unsigned int* src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.39 OleVariant::OleVariant Constructor (unsigned long*)

Pascal

```
constructor Create(var src: unsigned long);
```

C++

```
__fastcall OleVariant(unsigned long* src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.40 OleVariant::OleVariant Constructor (unsigned short)

Pascal

```
constructor Create(const src: unsigned short);
```

C++

```
__fastcall OleVariant(const unsigned short src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.41 OleVariant::OleVariant Constructor (unsigned short*)

Pascal

```
constructor Create(var src: unsigned short);
```

C++

```
__fastcall OleVariant(unsigned short* src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.42 OleVariant::OleVariant Constructor (wchar_t* const, bool)

Pascal

```
constructor Create(src: wchar_t* const; isBSTR: bool = true);
```

C++

```
__fastcall OleVariant(wchar_t* const src, bool isBSTR = true);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.3.43 OleVariant::OleVariant Constructor (wchar_t**)

Pascal

```
constructor Create(var src: wchar_t*);
```

C++

```
__fastcall OleVariant(wchar_t** src);
```

Description

This is OleVariant, a member of class OleVariant.

158.1.12.4 OleVariant Operators

The operators of the OleVariant class are listed here.

Public Operators

	Name	Description
	!= (see page 22209)	This is the overview for the != operator overload.
	< (see page 22210)	This is the overview for the < operator overload.
	<= (see page 22211)	This is the overview for the <= operator overload.
	= (see page 22211)	This is the overview for the = operator overload.
	== (see page 22214)	This is the overview for the == operator overload.
	> (see page 22214)	This is the overview for the > operator overload.
	>= (see page 22215)	This is the overview for the >= operator overload.

158.1.12.4.1 != Operator

This is the overview for the != operator overload.

Overload List

	Name	Description
	OleVariant::!= (char*) (see page 22210)	This is !=, a member of class OleVariant.
	OleVariant::!= (OleVariant&) (see page 22210)	This is !=, a member of class OleVariant.

158.1.12.4.1.1 OleVariant::!= Operator (OleVariant&)

Pascal

```
operator !=(const rhs: OleVariant&);
```

C++

```
__fastcall bool operator !=(const OleVariant& rhs) const;
```

Description

This is !=, a member of class OleVariant.

158.1.12.4.1.2 OleVariant::!= Operator (char*)

Pascal

```
operator !=(const rhs: char*);
```

C++

```
__fastcall bool operator !=(const char* rhs) const;
```

Description

This is !=, a member of class OleVariant.

158.1.12.4.2 < Operator

This is the overview for the < operator overload.

Overload List

	Name	Description
	OleVariant::< (char*) (see page 22210)	This is <, a member of class OleVariant.
	OleVariant::< (OleVariant&) (see page 22210)	This is <, a member of class OleVariant.

158.1.12.4.2.1 OleVariant::< Operator (OleVariant&)

Pascal

```
operator <(const rhs: OleVariant&);
```

C++

```
__fastcall bool operator <(const OleVariant& rhs) const;
```

Description

This is <, a member of class OleVariant.

158.1.12.4.2.2 OleVariant::< Operator (char*)

Pascal

```
operator <(const rhs: char*);
```

C++

```
__fastcall bool operator <(const char* rhs) const;
```

Description

This is <, a member of class OleVariant.

158.1.12.4.3 <= Operator

This is the overview for the <= operator overload.

Overload List

	Name	Description
	OleVariant::<= (char*) (see page 22211)	This is <=, a member of class OleVariant.
	OleVariant::<= (OleVariant&) (see page 22211)	This is <=, a member of class OleVariant.

158.1.12.4.3.1 OleVariant::<= Operator (OleVariant&)

Pascal

```
operator <=(const rhs: OleVariant&);
```

C++

```
__fastcall bool operator <=(const OleVariant& rhs) const;
```

Description

This is <=, a member of class OleVariant.

158.1.12.4.3.2 OleVariant::<= Operator (char*)

Pascal

```
operator <=(const rhs: char*);
```

C++

```
__fastcall bool operator <=(const char* rhs) const;
```

Description

This is <=, a member of class OleVariant.

158.1.12.4.4 = Operator

This is the overview for the = operator overload.

Overload List

	Name	Description
	OleVariant::= (__int64) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (AnsiString&) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (bool) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (char) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (char*) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (Currency&) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).

	OleVariant::= (CURRENCY&) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (Currency*) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (CURRENCY*) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (double) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (double*) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (float) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (IDispatch*) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (int) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (IUnknown*) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (long) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (OleVariant&) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (SAFEARRAY**) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (short) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (signed char) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (TDateTime&) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (TDateTime*) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (unsigned char) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (unsigned int) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (unsigned long) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (unsigned short) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (Variant&) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (VARIANT&) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).
	OleVariant::= (VARIANT*) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).

	OleVariant::= (WideString&) (see page 22213)	Copies a value into the System::OleVariant::OleVariant (see page 22200).

158.1.12.4.4.1 OleVariant::= Operator (AnsiString&)

Copies a value into the System::OleVariant::OleVariant (see page 22200).

Pascal

```
operator =(const rhs: OleVariant&);
operator =(const rhs: Variant&);
operator =(var rhs: VARIANT);
operator =(const rhs: AnsiString&);
operator =(const rhs: char*);
operator =(const rhs: WideString&);
operator =(const rhs: bool);
operator =(const rhs: char);
operator =(const rhs: signed char);
operator =(const rhs: unsigned char);
operator =(const rhs: short);
operator =(const rhs: unsigned short);
operator =(const rhs: Integer);
operator =(const rhs: unsigned int);
operator =(const rhs: long);
operator =(const rhs: unsigned long);
operator =(const rhs: float);
operator =(const rhs: double);
operator =(const rhs: __int64);
operator =(const rhs: CURRENCY&);
operator =(const rhs: TDateTime&);
operator =(const rhs: Currency&);
operator =(var rhs: double);
operator =(var rhs: TDateTime);
operator =(var rhs: Currency);
operator =(var rhs: IDispatch);
operator =(var rhs: IUnknown);
operator =(var src: CURRENCY);
operator =(var src: SAFEARRAY*);
operator =(var src: VARIANT);
```

C++

```
__fastcall OleVariant& operator =(const OleVariant& rhs);
__fastcall OleVariant& operator =(const Variant& rhs);
__fastcall OleVariant& operator =(VARIANT& rhs);
__fastcall OleVariant& operator =(const AnsiString& rhs);
__fastcall OleVariant& operator =(const char* rhs);
__fastcall OleVariant& operator =(const WideString& rhs);
__fastcall OleVariant& operator =(const bool rhs);
__fastcall OleVariant& operator =(const char rhs);
__fastcall OleVariant& operator =(const signed char rhs);
__fastcall OleVariant& operator =(const unsigned char rhs);
__fastcall OleVariant& operator =(const short rhs);
__fastcall OleVariant& operator =(const unsigned short rhs);
__fastcall OleVariant& operator =(const int rhs);
__fastcall OleVariant& operator =(const unsigned int rhs);
__fastcall OleVariant& operator =(const long rhs);
__fastcall OleVariant& operator =(const unsigned long rhs);
__fastcall OleVariant& operator =(const float rhs);
__fastcall OleVariant& operator =(const double rhs);
__fastcall OleVariant& operator =(const __int64 rhs);
__fastcall OleVariant& operator =(const CURRENCY& rhs);
__fastcall OleVariant& operator =(const TDateTime& rhs);
__fastcall OleVariant& operator =(const Currency& rhs);
__fastcall OleVariant& operator =(double* rhs);
```

```
__fastcall OleVariant& operator =(TDateTime* rhs);
__fastcall OleVariant& operator =(Currency* rhs);
__fastcall OleVariant& operator =(IDispatch* rhs);
__fastcall OleVariant& operator =(IUnknown* rhs);
__fastcall OleVariant& operator =(CURRENCY* src);
__fastcall OleVariant& operator =(SAFEARRAY** src);
__fastcall OleVariant& operator =(VARIANT* src);
```

Description

The assignment operator converts the value of rhs, as appropriate, and assigns it to the System::OleVariant::OleVariant (see page 22200). If it is not possible to perform the conversion, an EVariantError exception is thrown.

158.1.12.4.5 == Operator

This is the overview for the == operator overload.

Overload List

	Name	Description
	OleVariant::== (char*) (see page 22214)	This is ==, a member of class OleVariant.
	OleVariant::== (OleVariant&) (see page 22214)	This is ==, a member of class OleVariant.

158.1.12.4.5.1 OleVariant::== Operator (OleVariant&)

Pascal

```
operator ==(const rhs: OleVariant&);
```

C++

```
__fastcall bool operator ==(const OleVariant& rhs) const;
```

Description

This is ==, a member of class OleVariant.

158.1.12.4.5.2 OleVariant::== Operator (char*)

Pascal

```
operator ==(const rhs: char*);
```

C++

```
__fastcall bool operator ==(const char* rhs) const;
```

Description

This is ==, a member of class OleVariant.

158.1.12.4.6 > Operator

This is the overview for the > operator overload.

Overload List

	Name	Description
	OleVariant::> (char*) (see page 22215)	This is >, a member of class OleVariant.
	OleVariant::> (OleVariant&) (see page 22215)	This is >, a member of class OleVariant.

158.1.12.4.6.1 OleVariant::> Operator (OleVariant&)

Pascal

```
operator >(const rhs: OleVariant&);
```

C++

```
__fastcall bool operator >(const OleVariant& rhs) const;
```

Description

This is >, a member of class OleVariant.

158.1.12.4.6.2 OleVariant::> Operator (char*)

Pascal

```
operator >(const rhs: char*);
```

C++

```
__fastcall bool operator >(const char* rhs) const;
```

Description

This is >, a member of class OleVariant.

158.1.12.4.7 >= Operator

This is the overview for the >= operator overload.

Overload List

	Name	Description
	OleVariant::>= (char*) (see page 22215)	This is >=, a member of class OleVariant.
	OleVariant::>= (OleVariant&) (see page 22215)	This is >=, a member of class OleVariant.

158.1.12.4.7.1 OleVariant::>= Operator (OleVariant&)

Pascal

```
operator >=(const rhs: OleVariant&);
```

C++

```
__fastcall bool operator >=(const OleVariant& rhs) const;
```

Description

This is >=, a member of class OleVariant.

158.1.12.4.7.2 OleVariant::>= Operator (char*)

Pascal

```
operator >=(const rhs: char*);
```

C++

```
__fastcall bool operator >=(const char* rhs) const;
```

Description

This is >=, a member of class OleVariant.

158.1.13 OpenArray Template

Class Hierarchy

System::OpenArray

Pascal

```
OpenArray = class;
```

C++

```
template <class T>
class OpenArray;
```

File

sysopen.h

Description

This is class System::OpenArray.

158.1.13.1 OpenArray Members

The following tables list the members exposed by OpenArray.

Public Methods

	Name	Description
	~OpenArray (see page 22216)	This is ~OpenArray, a member of class OpenArray.
	GetHigh (see page 22223)	This is GetHigh, a member of class OpenArray.
	OpenArray (see page 22216)	This is the overview for the OpenArray constructor overload.

Public Operators

	Name	Description
	= (see page 22223)	This is =, a member of class OpenArray.
	T* (see page 22223)	This is T*, a member of class OpenArray.

158.1.13.2 OpenArray::~OpenArray Destructor

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~OpenArray();
```

Description

This is ~OpenArray, a member of class OpenArray.

158.1.13.3 OpenArray Constructor

This is the overview for the OpenArray constructor overload.

Overload List

	Name	Description
	OpenArray::OpenArray (OpenArray&) (see page 22217)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T) (see page 22218)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T, T) (see page 22218)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T, T, T) (see page 22218)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T, T, T, T) (see page 22218)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T, T, T, T, T) (see page 22219)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T, T, T, T, T, T) (see page 22219)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T, T, T, T, T, T, T) (see page 22219)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T, T, T, T, T, T, T, T) (see page 22219)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T, T, T, T, T, T, T, T, T) (see page 22219)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T, T, T, T, T, T, T, T, T, T) (see page 22220)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T, T, T, T, T, T, T, T, T, T, T) (see page 22220)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T, T, T, T, T, T, T, T, T, T, T, T) (see page 22220)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T, T, T, T, T, T, T, T, T, T, T, T, T) (see page 22220)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T, T, T, T, T, T, T, T, T, T, T, T, T, T) (see page 22221)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T) (see page 22221)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T) (see page 22221)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T) (see page 22222)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T) (see page 22222)	This is OpenArray, a member of class OpenArray.
	OpenArray::OpenArray (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T) (see page 22222)	This is OpenArray, a member of class OpenArray.

158.1.13.3.1 OpenArray::OpenArray Constructor (OpenArray&)

Pascal

```
constructor Create(const src: OpenArray&);
```

C++

```
__fastcall OpenArray(const OpenArray& src);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.2 OpenArray::OpenArray Constructor (T)

Pascal

```
constructor Create(arg0: T);
```

C++

```
__fastcall OpenArray(T arg0);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.3 OpenArray::OpenArray Constructor (T, T)

Pascal

```
constructor Create(arg0: T; arg1: T);
```

C++

```
__fastcall OpenArray(T arg0, T arg1);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.4 OpenArray::OpenArray Constructor (T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T);
```

C++

```
__fastcall OpenArray(T arg0, T arg1, T arg2);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.5 OpenArray::OpenArray Constructor (T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T);
```

C++

```
__fastcall OpenArray(T arg0, T arg1, T arg2, T arg3);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.6 OpenArray::OpenArray Constructor (T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T);
```

C++

```
__fastcall OpenArray(T arg0, T arg1, T arg2, T arg3, T arg4);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.7 OpenArray::OpenArray Constructor (T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T);
```

C++

```
__fastcall OpenArray(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.8 OpenArray::OpenArray Constructor (T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T);
```

C++

```
__fastcall OpenArray(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.9 OpenArray::OpenArray Constructor (T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T);
```

C++

```
__fastcall OpenArray(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.10 OpenArray::OpenArray Constructor (T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T);
```

C++

```
__fastcall OpenArray(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T arg8);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.11 OpenArray::OpenArray Constructor (T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T; arg9: T);
```

C++

```
__fastcall OpenArray(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T arg8, T
arg9);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.12 OpenArray::OpenArray Constructor (T, T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T; arg9: T; arg10: T);
```

C++

```
__fastcall OpenArray(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T arg8, T
arg9, T arg10);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.13 OpenArray::OpenArray Constructor (T, T, T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T; arg9: T; arg10: T; arg11: T);
```

C++

```
__fastcall OpenArray(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T arg8, T
arg9, T arg10, T arg11);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.14 OpenArray::OpenArray Constructor (T, T, T, T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
```

```
arg8: T; arg9: T; arg10: T; arg11: T; arg12: T);
```

C++

```
__fastcall OpenArray(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T arg8, T
arg9, T arg10, T arg11, T arg12);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.15 OpenArray::OpenArray Constructor (T, T, T, T, T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T; arg9: T; arg10: T; arg11: T; arg12: T; arg13: T);
```

C++

```
__fastcall OpenArray(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T arg8, T
arg9, T arg10, T arg11, T arg12, T arg13);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.16 OpenArray::OpenArray Constructor (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T; arg9: T; arg10: T; arg11: T; arg12: T; arg13: T; arg14: T);
```

C++

```
__fastcall OpenArray(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T arg8, T
arg9, T arg10, T arg11, T arg12, T arg13, T arg14);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.17 OpenArray::OpenArray Constructor (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T; arg9: T; arg10: T; arg11: T; arg12: T; arg13: T; arg14: T; arg15: T);
```

C++

```
__fastcall OpenArray(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T arg8, T
arg9, T arg10, T arg11, T arg12, T arg13, T arg14, T arg15);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.18 OpenArray::OpenArray Constructor (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T; arg9: T; arg10: T; arg11: T; arg12: T; arg13: T; arg14: T; arg15: T; arg16: T);
```

C++

```
__fastcall OpenArray(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T arg8, T
arg9, T arg10, T arg11, T arg12, T arg13, T arg14, T arg15, T arg16);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.19 OpenArray::OpenArray Constructor (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T; arg9: T; arg10: T; arg11: T; arg12: T; arg13: T; arg14: T; arg15: T; arg16: T; arg17:
T);
```

C++

```
__fastcall OpenArray(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T arg8, T
arg9, T arg10, T arg11, T arg12, T arg13, T arg14, T arg15, T arg16, T arg17);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.3.20 OpenArray::OpenArray Constructor (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T; arg9: T; arg10: T; arg11: T; arg12: T; arg13: T; arg14: T; arg15: T; arg16: T; arg17:
T; arg18: T);
```

C++

```
__fastcall OpenArray(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T arg8, T
arg9, T arg10, T arg11, T arg12, T arg13, T arg14, T arg15, T arg16, T arg17, T arg18);
```

Description

This is OpenArray, a member of class OpenArray.

158.1.13.4 OpenArray Methods

The methods of the OpenArray class are listed here.

Public Methods

	Name	Description
	GetHigh (see page 22223)	This is GetHigh, a member of class OpenArray.

158.1.13.4.1 OpenArray::GetHigh Method

Pascal

```
function GetHigh: Integer;
```

C++

```
__fastcall int GetHigh();
```

Description

This is GetHigh, a member of class OpenArray.

158.1.13.5 OpenArray Operators

The operators of the OpenArray class are listed here.

Public Operators

	Name	Description
	= (see page 22223)	This is =, a member of class OpenArray.
	T* (see page 22223)	This is T*, a member of class OpenArray.

158.1.13.5.1 OpenArray::= Operator

Pascal

```
operator =(const rhs: OpenArray&);
```

C++

```
__fastcall OpenArray& operator =(const OpenArray& rhs);
```

Description

This is =, a member of class OpenArray.

158.1.13.5.2 OpenArray::T* Operator

Pascal

```
operator T*;
```

C++

```
__fastcall operator T*();
```

Description

This is T*, a member of class OpenArray.

158.1.14 OpenArrayCount Template

Class Hierarchy

System::OpenArrayCount

Pascal

```
OpenArrayCount = class;
```

C++

```
template <class T>
class OpenArrayCount;
```

File

sysopen.h

Description

This is class System::OpenArrayCount.

158.1.14.1 OpenArrayCount Members

The following tables list the members exposed by OpenArrayCount.

Public Methods

	Name	Description
	GetHigh (see page 22230)	This is GetHigh, a member of class OpenArrayCount.
	OpenArrayCount (see page 22224)	This is the overview for the OpenArrayCount constructor overload.

158.1.14.2 OpenArrayCount Constructor

This is the overview for the OpenArrayCount constructor overload.

Overload List

	Name	Description
	OpenArrayCount::OpenArrayCount (T) (see page 22225)	This is OpenArrayCount, a member of class OpenArrayCount.
	OpenArrayCount::OpenArrayCount (T, T) (see page 22225)	This is OpenArrayCount, a member of class OpenArrayCount.
	OpenArrayCount::OpenArrayCount (T, T, T) (see page 22225)	This is OpenArrayCount, a member of class OpenArrayCount.
	OpenArrayCount::OpenArrayCount (T, T, T, T) (see page 22226)	This is OpenArrayCount, a member of class OpenArrayCount.
	OpenArrayCount::OpenArrayCount (T, T, T, T, T) (see page 22226)	This is OpenArrayCount, a member of class OpenArrayCount.
	OpenArrayCount::OpenArrayCount (T, T, T, T, T, T) (see page 22226)	This is OpenArrayCount, a member of class OpenArrayCount.
	OpenArrayCount::OpenArrayCount (T, T, T, T, T, T, T) (see page 22226)	This is OpenArrayCount, a member of class OpenArrayCount.
	OpenArrayCount::OpenArrayCount (T, T, T, T, T, T, T, T) (see page 22227)	This is OpenArrayCount, a member of class OpenArrayCount.
	OpenArrayCount::OpenArrayCount (T, T, T, T, T, T, T, T, T) (see page 22227)	This is OpenArrayCount, a member of class OpenArrayCount.
	OpenArrayCount::OpenArrayCount (T, T, T, T, T, T, T, T, T, T) (see page 22227)	This is OpenArrayCount, a member of class OpenArrayCount.

	OpenArrayCount::OpenArrayCount (T, T, T, T, T, T, T, T, T, T, T) (see page 22227)	This is OpenArrayCount, a member of class OpenArrayCount.
	OpenArrayCount::OpenArrayCount (T, T, T, T, T, T, T, T, T, T, T, T) (see page 22228)	This is OpenArrayCount, a member of class OpenArrayCount.
	OpenArrayCount::OpenArrayCount (T, T, T, T, T, T, T, T, T, T, T, T, T) (see page 22228)	This is OpenArrayCount, a member of class OpenArrayCount.
	OpenArrayCount::OpenArrayCount (T, T, T, T, T, T, T, T, T, T, T, T, T, T) (see page 22228)	This is OpenArrayCount, a member of class OpenArrayCount.
	OpenArrayCount::OpenArrayCount (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T) (see page 22228)	This is OpenArrayCount, a member of class OpenArrayCount.
	OpenArrayCount::OpenArrayCount (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T) (see page 22229)	This is OpenArrayCount, a member of class OpenArrayCount.
	OpenArrayCount::OpenArrayCount (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T) (see page 22229)	This is OpenArrayCount, a member of class OpenArrayCount.
	OpenArrayCount::OpenArrayCount (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T) (see page 22229)	This is OpenArrayCount, a member of class OpenArrayCount.
	OpenArrayCount::OpenArrayCount (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T) (see page 22230)	This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.1 OpenArrayCount::OpenArrayCount Constructor (T)

Pascal

```
constructor Create(arg0: T);
```

C++

```
__fastcall OpenArrayCount(T arg0);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.2 OpenArrayCount::OpenArrayCount Constructor (T, T)

Pascal

```
constructor Create(arg0: T; arg1: T);
```

C++

```
__fastcall OpenArrayCount(T arg0, T arg1);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.3 OpenArrayCount::OpenArrayCount Constructor (T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T);
```

C++

```
__fastcall OpenArrayCount(T arg0, T arg1, T arg2);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.4 OpenArrayCount::OpenArrayCount Constructor (T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T);
```

C++

```
__fastcall OpenArrayCount(T arg0, T arg1, T arg2, T arg3);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.5 OpenArrayCount::OpenArrayCount Constructor (T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T);
```

C++

```
__fastcall OpenArrayCount(T arg0, T arg1, T arg2, T arg3, T arg4);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.6 OpenArrayCount::OpenArrayCount Constructor (T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T);
```

C++

```
__fastcall OpenArrayCount(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.7 OpenArrayCount::OpenArrayCount Constructor (T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T);
```

C++

```
__fastcall OpenArrayCount(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.8 OpenArrayCount::OpenArrayCount Constructor (T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T);
```

C++

```
__fastcall OpenArrayCount(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.9 OpenArrayCount::OpenArrayCount Constructor (T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T);
```

C++

```
__fastcall OpenArrayCount(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T
arg8);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.10 OpenArrayCount::OpenArrayCount Constructor (T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T; arg9: T);
```

C++

```
__fastcall OpenArrayCount(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T
arg8, T arg9);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.11 OpenArrayCount::OpenArrayCount Constructor (T, T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T; arg9: T; arg10: T);
```

C++

```
__fastcall OpenArrayCount(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T
arg8, T arg9, T arg10);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.12 OpenArrayCount::OpenArrayCount Constructor (T, T, T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T; arg9: T; arg10: T; arg11: T);
```

C++

```
__fastcall OpenArrayCount(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T
arg8, T arg9, T arg10, T arg11);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.13 OpenArrayCount::OpenArrayCount Constructor (T, T, T, T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T; arg9: T; arg10: T; arg11: T; arg12: T);
```

C++

```
__fastcall OpenArrayCount(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T
arg8, T arg9, T arg10, T arg11, T arg12);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.14 OpenArrayCount::OpenArrayCount Constructor (T, T, T, T, T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T; arg9: T; arg10: T; arg11: T; arg12: T; arg13: T);
```

C++

```
__fastcall OpenArrayCount(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T
arg8, T arg9, T arg10, T arg11, T arg12, T arg13);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.15 OpenArrayCount::OpenArrayCount Constructor (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
```

```
arg8: T; arg9: T; arg10: T; arg11: T; arg12: T; arg13: T; arg14: T);
```

C++

```
__fastcall OpenArrayCount(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T
arg8, T arg9, T arg10, T arg11, T arg12, T arg13, T arg14);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.16 OpenArrayCount::OpenArrayCount Constructor (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T; arg9: T; arg10: T; arg11: T; arg12: T; arg13: T; arg14: T; arg15: T);
```

C++

```
__fastcall OpenArrayCount(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T
arg8, T arg9, T arg10, T arg11, T arg12, T arg13, T arg14, T arg15);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.17 OpenArrayCount::OpenArrayCount Constructor (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T; arg9: T; arg10: T; arg11: T; arg12: T; arg13: T; arg14: T; arg15: T; arg16: T);
```

C++

```
__fastcall OpenArrayCount(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T
arg8, T arg9, T arg10, T arg11, T arg12, T arg13, T arg14, T arg15, T arg16);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.18 OpenArrayCount::OpenArrayCount Constructor (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T; arg9: T; arg10: T; arg11: T; arg12: T; arg13: T; arg14: T; arg15: T; arg16: T; arg17:
T);
```

C++

```
__fastcall OpenArrayCount(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T
arg8, T arg9, T arg10, T arg11, T arg12, T arg13, T arg14, T arg15, T arg16, T arg17);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.2.19 OpenArrayCount::OpenArrayCount Constructor (T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T)

Pascal

```
constructor Create(arg0: T; arg1: T; arg2: T; arg3: T; arg4: T; arg5: T; arg6: T; arg7: T;
arg8: T; arg9: T; arg10: T; arg11: T; arg12: T; arg13: T; arg14: T; arg15: T; arg16: T; arg17:
T; arg18: T);
```

C++

```
__fastcall OpenArrayCount(T arg0, T arg1, T arg2, T arg3, T arg4, T arg5, T arg6, T arg7, T
arg8, T arg9, T arg10, T arg11, T arg12, T arg13, T arg14, T arg15, T arg16, T arg17, T arg18);
```

Description

This is OpenArrayCount, a member of class OpenArrayCount.

158.1.14.3 OpenArrayCount Methods

The methods of the OpenArrayCount class are listed here.

Public Methods

	Name	Description
	GetHigh (see page 22230)	This is GetHigh, a member of class OpenArrayCount.

158.1.14.3.1 OpenArrayCount::GetHigh Method

Pascal

```
function GetHigh: Integer;
```

C++

```
__fastcall int GetHigh();
```

Description

This is GetHigh, a member of class OpenArrayCount.

158.1.15 Procedure Class

System::Procedure represents a method on an Automation interface that does not return a value.

Class Hierarchy

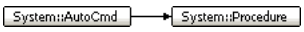

Pascal

```
Procedure = class(AutoCmd);
```

C++

```
class Procedure : public AutoCmd;
```

File

sysvari.h

Description

The System::Procedure class represents a single method on an Automation interface that is assigned as the value of a Variant (see page 22307). It can be executed using the Variant (see page 22307)'s Exec method.

Each System::Procedure instance is assigned a name, which is the same as the name of the corresponding method on the interface. In addition, it can be assigned parameters that correspond to the parameters of the method. These parameters can be stored by position or by name.

See Also

System::Variant (see page 22307)

System::Function (see page 22184)

PropertyGet

PropertySet

158.1.15.1 Procedure Members

The following tables list the members exposed by Procedure.

Protected Data Members

	Name	Description
	Name (see page 22130)	This is Name, a member of class AutoCmd.
	NamedParmNames (see page 22130)	This is NamedParmNames, a member of class AutoCmd.
	NamedParms (see page 22130)	This is NamedParms, a member of class AutoCmd.
	NamedParmTypes (see page 22131)	This is NamedParmTypes, a member of class AutoCmd.
	Parms (see page 22131)	This is Parms, a member of class AutoCmd.
	ParmTypes (see page 22131)	This is ParmTypes, a member of class AutoCmd.

Protected Methods

	Name	Description
	AddElement (see page 22132)	This is AddElement, a member of class AutoCmd.

Public Methods

	Name	Description
	~AutoCmd (see page 22129)	Frees an System::AutoCmd::AutoCmd (see page 22129) object.
	AutoCmd (see page 22129)	This is the overview for the AutoCmd constructor overload.
	Clear (see page 22132)	System::AutoCmd::Clears any assigned parameters and blanks the name of the associated method.
	ClearArgs (see page 22132)	Clears any assigned parameters.
	ClearName (see page 22133)	Blanks the name of the associated method.
	GetArgCount (see page 22133)	Returns the number of positional arguments that have been assigned.
	GetArgType (see page 22134)	Returns the type of a parameter value.
	GetCallType (see page 22232)	Indicates the type of the interface method represented by the System::Procedure::Procedure (see page 22232) object.
	GetName (see page 22134)	Returns the name of the interface method this System::AutoCmd::AutoCmd (see page 22129) object represents.
	GetNamedArgCount (see page 22135)	Returns the number of named arguments that have been assigned.
	GetNamedArgType (see page 22135)	Returns the type of a named parameter value.
	GetNamedParm (see page 22136)	Returns the value of a named parameter.
	GetNamedParmName (see page 22136)	Returns the named of a named parameter value.
	GetParm (see page 22137)	Returns the value of a positional parameter.
	Procedure (see page 22232)	This is the overview for the Procedure constructor overload.
	RequestResult (see page 22233)	Indicates that the interface method does not return a value.

	SetName (see page 22138)	Sets the name of the method the System::AutoCmd::AutoCmd (see page 22129) object represents.

Public Operators

	Name	Description
	<< (see page 22138)	This is the overview for the << operator overload.

158.1.15.2 Procedure Constructor

This is the overview for the Procedure constructor overload.

Overload List

	Name	Description
	Procedure::Procedure (AutoCmd&) (see page 22232)	Creates an instance of System::Procedure::Create for a specified method.
	Procedure::Procedure (String&) (see page 22232)	Creates an instance of System::Procedure::Create for a specified method.

158.1.15.2.1 Procedure::Procedure Constructor (AutoCmd&)

Creates an instance of System::Procedure::Create for a specified method.

Pascal

```
constructor Create(const name: String&);
constructor Create(const src: AutoCmd&);
```

C++

```
Procedure(const String& name);
Procedure(const AutoCmd& src);
```

Description

System::Procedure::Create has two constructors: a version that takes the name of the method it represents, and a copy constructor. When creating a new System::Procedure::Create object for a method, the name parameter must be the same as the name of the method as it appears in the interface declaration.

158.1.15.3 Procedure Methods

The methods of the Procedure class are listed here.

Public Methods

	Name	Description
	GetCallType (see page 22232)	Indicates the type of the interface method represented by the System::Procedure::Procedure (see page 22232) object.
	RequestResult (see page 22233)	Indicates that the interface method does not return a value.

158.1.15.3.1 Procedure::GetCallType Method

Indicates the type of the interface method represented by the System::Procedure::Procedure (see page 22232) object.

Pascal

```
function GetCallType: Byte;
```

C++

```
Byte GetCallType() const;
```

Description

Call System::Procedure::GetCallType to determine the type of flag needed to call the method represented by System::Procedure::Procedure (see page 22232) using the IDispatch::Invoke method of the Automation interface. System::Procedure::GetCallType returns DISPATCH_METHOD, indicating a method that is not a property access method.

158.1.15.3.2 Procedure::RequestResult Method

Indicates that the interface method does not return a value.

Pascal

```
function RequestResult: bool;
```

C++

```
bool RequestResult() const;
```

Description

System::Procedure::RequestResult always returns false, indicating that there is no return value after executing the System::Procedure::Procedure (see page 22232) object using the Variant::Exec method.

See Also

System::Procedure::GetCallType (see page 22232)

System::Variant::Exec (see page 22320)

158.1.16 PropertyGet Class

PropertyGet represents a property access method on an Automation interface.

Class Hierarchy

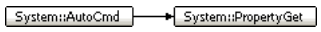

Pascal

```
PropertyGet = class(AutoCmd);
```

C++

```
class PropertyGet : public AutoCmd;
```

File

sysvari.h

Description

The PropertyGet class represents a single method on an Automation interface that returns the value of a property. The interface is assigned as the value of a Variant (see page 22307). The PropertyGet object can be executed using the Variant (see page 22307)'s Exec method.

Each PropertyGet instance is assigned a name, which is the same as the name of the corresponding method on the interface. In addition, it can be assigned parameters that correspond to the parameters of the method. These parameters can be stored by position or by name.

See Also

System::Variant (see page 22307)

System::Function (see page 22184)

System::Procedure (see page 22230)

PropertySet

158.1.16.1 PropertyGet Members

The following tables list the members exposed by PropertyGet.

Protected Data Members

	Name	Description
	Name (see page 22130)	This is Name, a member of class AutoCmd.
	NamedParmNames (see page 22130)	This is NamedParmNames, a member of class AutoCmd.
	NamedParms (see page 22130)	This is NamedParms, a member of class AutoCmd.
	NamedParmTypes (see page 22131)	This is NamedParmTypes, a member of class AutoCmd.
	Parms (see page 22131)	This is Parms, a member of class AutoCmd.
	ParmTypes (see page 22131)	This is ParmTypes, a member of class AutoCmd.

Protected Methods

	Name	Description
	AddElement (see page 22132)	This is AddElement, a member of class AutoCmd.

Public Methods

	Name	Description
	~AutoCmd (see page 22129)	Frees an System::AutoCmd::AutoCmd (see page 22129) object.
	AutoCmd (see page 22129)	This is the overview for the AutoCmd constructor overload.
	Clear (see page 22132)	System::AutoCmd::Clears any assigned parameters and blanks the name of the associated method.
	ClearArgs (see page 22132)	Clears any assigned parameters.
	ClearName (see page 22133)	Blanks the name of the associated method.
	GetArgCount (see page 22133)	Returns the number of positional arguments that have been assigned.
	GetArgType (see page 22134)	Returns the type of a parameter value.
	GetCallType (see page 22235)	Indicates the type of the interface method represented by the PropertyGet (see page 22233) object.
	GetName (see page 22134)	Returns the name of the interface method this System::AutoCmd::AutoCmd (see page 22129) object represents.
	GetNamedArgCount (see page 22135)	Returns the number of named arguments that have been assigned.
	GetNamedArgType (see page 22135)	Returns the type of a named parameter value.
	GetNamedParm (see page 22136)	Returns the value of a named parameter.
	GetNamedParmName (see page 22136)	Returns the named of a named parameter value.
	GetParm (see page 22137)	Returns the value of a positional parameter.
	PropertyGet (see page 22235)	This is the overview for the PropertyGet constructor overload.
	RequestResult (see page 22236)	Indicates that the interface method returns a value.
	SetName (see page 22138)	Sets the name of the method the System::AutoCmd::AutoCmd (see page 22129) object represents.

Public Operators

	Name	Description
	<< (see page 22138)	This is the overview for the << operator overload.

158.1.16.2 PropertyGet Constructor

This is the overview for the PropertyGet constructor overload.

Overload List

	Name	Description
	PropertyGet::PropertyGet (AutoCmd&) (see page 22235)	This is PropertyGet, a member of class PropertyGet.
	PropertyGet::PropertyGet (String&) (see page 22235)	This is PropertyGet, a member of class PropertyGet.

158.1.16.2.1 PropertyGet::PropertyGet Constructor (AutoCmd&)

Pascal

```
constructor Create(const src: AutoCmd&);
```

C++

```
PropertyGet(const AutoCmd& src);
```

Description

This is PropertyGet, a member of class PropertyGet.

158.1.16.2.2 PropertyGet::PropertyGet Constructor (String&)

Pascal

```
constructor Create(const name: String&);
```

C++

```
PropertyGet(const String& name);
```

Description

This is PropertyGet, a member of class PropertyGet.

158.1.16.3 PropertyGet Methods

The methods of the PropertyGet class are listed here.

Public Methods

	Name	Description
	GetCallType (see page 22235)	Indicates the type of the interface method represented by the PropertyGet (see page 22233) object.
	RequestResult (see page 22236)	Indicates that the interface method returns a value.

158.1.16.3.1 PropertyGet::GetCallType Method

Indicates the type of the interface method represented by the PropertyGet (see page 22233) object.

Pascal

```
function GetCallType: Byte;
```

C++

```
Byte GetCallType() const;
```

Description

Call GetCallType to determine the type of flag needed to call the method represented by PropertyGet (see page 22233) using the IDispatch::Invoke method of the Automation interface. GetCallType returns DISPATCH_PROPERTYGET, indicating a method that returns the value of a property.

158.1.16.3.2 PropertyGet::RequestResult Method

Indicates that the interface method returns a value.

Pascal

```
function RequestResult: bool;
```

C++

```
bool RequestResult() const;
```

Description

RequestResult always returns true, indicating that there is a return value (the property value) after executing the PropertyGet (see page 22233) object using the Variant::Exec method.

See Also

GetCallType

System::Variant::Exec (see page 22320)

158.1.17 PropertySet Class

PropertySet represents a property access method on an Automation interface.

Class Hierarchy

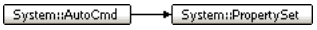

Pascal

```
PropertySet = class(AutoCmd);
```

C++

```
class PropertySet : public AutoCmd;
```

File

sysvari.h

Description

The PropertySet class represents a single method on an Automation interface that sets the value of a property. The interface is assigned as the value of a Variant (see page 22307). The PropertySet object can be executed using the Variant (see page 22307)'s Exec method.

Each PropertySet instance is assigned a name, which is the same as the name of the corresponding method on the interface. In addition, it can be assigned parameters that correspond to the parameters of the method. These parameters can be stored by position or by name.

See Also

System::Variant (see page 22307)

System::Function (see page 22184)

System::Procedure (see page 22230)

PropertySet

158.1.17.1 PropertySet Members

The following tables list the members exposed by PropertySet.

Protected Data Members

	Name	Description
	Name (see page 22130)	This is Name, a member of class AutoCmd.
	NamedParmNames (see page 22130)	This is NamedParmNames, a member of class AutoCmd.
	NamedParms (see page 22130)	This is NamedParms, a member of class AutoCmd.
	NamedParmTypes (see page 22131)	This is NamedParmTypes, a member of class AutoCmd.
	Parms (see page 22131)	This is Parms, a member of class AutoCmd.
	ParmTypes (see page 22131)	This is ParmTypes, a member of class AutoCmd.

Protected Methods

	Name	Description
	AddElement (see page 22132)	This is AddElement, a member of class AutoCmd.

Public Methods

	Name	Description
	~AutoCmd (see page 22129)	Frees an System::AutoCmd::AutoCmd (see page 22129) object.
	AutoCmd (see page 22129)	This is the overview for the AutoCmd constructor overload.
	Clear (see page 22132)	System::AutoCmd::Clears any assigned parameters and blanks the name of the associated method.
	ClearArgs (see page 22132)	Clears any assigned parameters.
	ClearName (see page 22133)	Blanks the name of the associated method.
	GetArgCount (see page 22133)	Returns the number of positional arguments that have been assigned.
	GetArgType (see page 22134)	Returns the type of a parameter value.
	GetCallType (see page 22238)	Indicates the type of the interface method represented by the PropertySet (see page 22236) object.
	GetName (see page 22134)	Returns the name of the interface method this System::AutoCmd::AutoCmd (see page 22129) object represents.
	GetNamedArgCount (see page 22135)	Returns the number of named arguments that have been assigned.
	GetNamedArgType (see page 22135)	Returns the type of a named parameter value.
	GetNamedParm (see page 22136)	Returns the value of a named parameter.
	GetNamedParmName (see page 22136)	Returns the named of a named parameter value.
	GetParm (see page 22137)	Returns the value of a positional parameter.
	PropertySet (see page 22238)	This is the overview for the PropertySet constructor overload.
	RequestResult (see page 22239)	Indicates that the interface method does not return a value.
	SetName (see page 22138)	Sets the name of the method the System::AutoCmd::AutoCmd (see page 22129) object represents.

Public Operators

	Name	Description
	<< (see page 22138)	This is the overview for the << operator overload.

158.1.17.2 PropertySet Constructor

This is the overview for the PropertySet constructor overload.

Overload List

	Name	Description
	PropertySet::PropertySet (AutoCmd&) (see page 22238)	This is PropertySet, a member of class PropertySet.
	PropertySet::PropertySet (String&) (see page 22238)	This is PropertySet, a member of class PropertySet.

158.1.17.2.1 PropertySet::PropertySet Constructor (AutoCmd&)

Pascal

```
constructor Create(const src: AutoCmd&);
```

C++

```
PropertySet(const AutoCmd& src);
```

Description

This is PropertySet, a member of class PropertySet.

158.1.17.2.2 PropertySet::PropertySet Constructor (String&)

Pascal

```
constructor Create(const name: String&);
```

C++

```
PropertySet(const String& name);
```

Description

This is PropertySet, a member of class PropertySet.

158.1.17.3 PropertySet Methods

The methods of the PropertySet class are listed here.

Public Methods

	Name	Description
	GetCallType (see page 22238)	Indicates the type of the interface method represented by the PropertySet (see page 22236) object.
	RequestResult (see page 22239)	Indicates that the interface method does not return a value.

158.1.17.3.1 PropertySet::GetCallType Method

Indicates the type of the interface method represented by the PropertySet (see page 22236) object.

Pascal

```
function GetCallType: Byte;
```

C++

```
Byte GetCallType() const;
```

Description

Call GetCallType to determine the type of flag needed to call the method represented by PropertySet (see page 22236) using the IDispatch::Invoke method of the Automation interface. GetCallType returns DISPATCH_PROPERTYPUT, indicating a method that sets the value of a property.

158.1.17.3.2 PropertySet::RequestResult Method

Indicates that the interface method does not return a value.

Pascal

```
function RequestResult: bool;
```

C++

```
bool RequestResult() const;
```

Description

RequestResult always returns false, indicating that there is no return value after executing the PropertySet (see page 22236) object using the Variant::Exec method.

See Also

GetCallType

System::Variant::Exec (see page 22320)

158.1.18 SmallString Template

Class Hierarchy

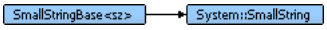

Pascal

```
SmallString = class(SmallStringBase<sz>);
```

C++

```
template <unsigned char sz>
class SmallString : private SmallStringBase<sz>;
```

File

dstring.h

Description

This is class System::SmallString.

158.1.18.1 SmallString Members

The following tables list the members exposed by SmallString.

Public Methods

	Name	Description
	SmallString (see page 22240)	This is the overview for the SmallString constructor overload.

Public Operators

	Name	Description
	[] (see page 22241)	This is [], a member of class SmallString.
	= (see page 22241)	This is =, a member of class SmallString.
	AnsiString (see page 22242)	This is AnsiString, a member of class SmallString.

158.1.18.2 SmallString Constructor

This is the overview for the SmallString constructor overload.

Overload List

	Name	Description
	SmallString::SmallString () (see page 22240)	This is SmallString, a member of class SmallString.
	SmallString::SmallString (AnsiString&) (see page 22240)	This is SmallString, a member of class SmallString.
	SmallString::SmallString (char*) (see page 22241)	This is SmallString, a member of class SmallString.
	SmallString::SmallString (SmallString&) (see page 22241)	This is SmallString, a member of class SmallString.

158.1.18.2.1 SmallString::SmallString Constructor ()

Pascal

```
constructor Create;
```

C++

```
__fastcall SmallString();
```

Description

This is SmallString, a member of class SmallString.

158.1.18.2.2 SmallString::SmallString Constructor (AnsiString&)

Pascal

```
constructor Create(const src: AnsiString&);
```

C++

```
__fastcall SmallString(const AnsiString& src);
```

Description

This is SmallString, a member of class SmallString.

158.1.18.2.3 SmallString::SmallString Constructor (SmallString&)

Pascal

```
constructor Create(const src: SmallString&);
```

C++

```
__fastcall SmallString(const SmallString& src);
```

Description

This is SmallString, a member of class SmallString.

158.1.18.2.4 SmallString::SmallString Constructor (char*)

Pascal

```
constructor Create(const src: char*);
```

C++

```
__fastcall SmallString(const char* src);
```

Description

This is SmallString, a member of class SmallString.

158.1.18.3 SmallString Operators

The operators of the SmallString class are listed here.

Public Operators

	Name	Description
	[] (see page 22241)	This is [], a member of class SmallString.
	= (see page 22241)	This is =, a member of class SmallString.
	AnsiString (see page 22242)	This is AnsiString, a member of class SmallString.

158.1.18.3.1 SmallString::[] Operator

Pascal

```
operator [](const idx: unsigned char);
```

C++

```
__fastcall char& operator [](const unsigned char idx);
```

Description

This is [], a member of class SmallString.

158.1.18.3.2 SmallString::= Operator

Pascal

```
operator =(const rhs: SmallString&);
```

C++

```
__fastcall SmallString& operator =(const SmallString& rhs);
```

Description

This is =, a member of class SmallString.

158.1.18.3.3 SmallString::AnsiString Operator

Pascal

```
operator AnsiString;
```

C++

```
__fastcall operator AnsiString() const;
```

Description

This is AnsiString, a member of class SmallString.

158.1.19 SmallStringBase Template

Class Hierarchy

System::SmallStringBase

Pascal

```
SmallStringBase = class;
```

C++

```
template <unsigned char sz>
class SmallStringBase;
```

File

dstring.h

Description

This is class System::SmallStringBase.

158.1.19.1 SmallStringBase Members

The following tables list the members exposed by SmallStringBase.

Protected Data Members

	Name	Description
	Data (see page 22243)	This is Data, a member of class SmallStringBase.
	Len (see page 22243)	This is Len, a member of class SmallStringBase.

158.1.19.2 SmallStringBase Data Members

The data members of the SmallStringBase class are listed here.

Protected Data Members

	Name	Description
	Data (see page 22243)	This is Data, a member of class SmallStringBase.
	Len (see page 22243)	This is Len, a member of class SmallStringBase.

158.1.19.2.1 SmallStringBase::Data Data Member

Pascal

```
[sz] Data: char;
```

C++

```
char Data[sz];
```

Description

This is Data, a member of class SmallStringBase.

158.1.19.2.2 SmallStringBase::Len Data Member

Pascal

```
Len: unsigned char;
```

C++

```
unsigned char Len;
```

Description

This is Len, a member of class SmallStringBase.

158.1.20 sPointer Template

Class Hierarchy

Pascal

```
sPointer = class;
```

C++

```
template <typename T>
class sPointer;
```

File

sysdyn.h

Description

This is class System::sPointer.

158.1.20.1 sPointer Members

The following tables list the members exposed by sPointer.

Public Methods

	Name	Description
	~sPointer (see page 22244)	This is ~sPointer, a member of class sPointer.
	sPointer (see page 22244)	This is the overview for the sPointer constructor overload.

Public Operators

	Name	Description
	& (see page 22245)	This is &, a member of class sPointer.
	-> (see page 22245)	This is ->, a member of class sPointer.

158.1.20.2 sPointer::~sPointer Destructor

Pascal

```
destructor Destroy;
```

C++

```
~sPointer();
```

Description

This is ~sPointer, a member of class sPointer.

158.1.20.3 sPointer Constructor

This is the overview for the sPointer constructor overload.

Overload List

	Name	Description
	sPointer::sPointer () (see page 22244)	This is sPointer, a member of class sPointer.
	sPointer::sPointer (T*) (see page 22244)	This is sPointer, a member of class sPointer.

158.1.20.3.1 sPointer::sPointer Constructor ()

Pascal

```
constructor Create;
```

C++

```
sPointer();
```

Description

This is sPointer, a member of class sPointer.

158.1.20.3.2 sPointer::sPointer Constructor (T*)

Pascal

```
constructor Create(var p: T);
```

C++

```
sPointer(T* p);
```

Description

This is sPointer, a member of class sPointer.

158.1.20.4 sPointer Operators

The operators of the sPointer class are listed here.

Public Operators

	Name	Description
	& (see page 22245)	This is &, a member of class sPointer.
	-> (see page 22245)	This is ->, a member of class sPointer.

158.1.20.4.1 sPointer::& Operator

Pascal

```
operator &;
```

C++

```
T* operator &();
```

Description

This is &, a member of class sPointer.

158.1.20.4.2 sPointer::-> Operator

Pascal

```
operator ->;
```

C++

```
T* operator ->();
```

Description

This is ->, a member of class sPointer.

158.1.21 StaticArray Structure

Pascal

```
StaticArray = record
  [size] data: T;
end;
```

C++

```
template <typename T, int size>
struct StaticArray {
  T data[size];
};
```

File

sysdyn.h

Description

This is class System::StaticArray.

158.1.21.1 StaticArray Members

The following tables list the members exposed by StaticArray.

Public Data Members

	Name	Description
	data (see page 22246)	This is data, a member of class StaticArray.

Public Operators

	Name	Description
	[] (see page 22246)	This is the overview for the [] operator overload.
	const T* (see page 22247)	This is const T*, a member of class StaticArray.
	T* (see page 22247)	This is T*, a member of class StaticArray.

158.1.21.2 StaticArray Data Members

The data members of the StaticArray class are listed here.

Public Data Members

	Name	Description
	data (see page 22246)	This is data, a member of class StaticArray.

158.1.21.2.1 StaticArray::data Data Member

Pascal

```
[size] data: T;
```

C++

```
T data[size];
```

Description

This is data, a member of class StaticArray.

158.1.21.3 StaticArray Operators

The operators of the StaticArray class are listed here.

Public Operators

	Name	Description
	[] (see page 22246)	This is the overview for the [] operator overload.
	const T* (see page 22247)	This is const T*, a member of class StaticArray.
	T* (see page 22247)	This is T*, a member of class StaticArray.

158.1.21.3.1 [] Operator

This is the overview for the [] operator overload.

Overload List

	Name	Description
	StaticArray::[] (int) (see page 22247)	This is [], a member of class StaticArray.
	StaticArray::[] (int) (see page 22247)	This is [], a member of class StaticArray.

158.1.21.3.1.1 StaticArray::[] Operator (int)

Pascal

```
operator [](index: Integer);
```

C++

```
T& operator [](int index);
```

Description

This is [], a member of class StaticArray.

158.1.21.3.1.2 StaticArray::[] Operator (int)

Pascal

```
operator [](index: Integer);
```

C++

```
const T& operator [](int index) const;
```

Description

This is [], a member of class StaticArray.

158.1.21.3.2 StaticArray::const T* Operator

Pascal

```
operator const T*;
```

C++

```
operator const T*() const;
```

Description

This is const T*, a member of class StaticArray.

158.1.21.3.3 StaticArray::T* Operator

Pascal

```
operator T*;
```

C++

```
operator T*();
```

Description

This is T*, a member of class StaticArray.

158.1.22 TAggregatedObject Class

System::TAggregatedObject provides the functionality for an inner object of an aggregate by implementing the IInterface (see page 22396) methods to delegate to the controlling IInterface (see page 22396).

Class Hierarchy

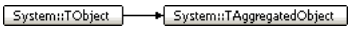

Pascal

```
TAggregatedObject = class(TObject);
```

C++

```
class TAggregatedObject : public TObject;
```

File

System.pas

Description

An aggregated object is an object composed of several interfaced objects. Each object implements its own behavior and interfaces, but all of the objects share the same reference count, which is that of the controller object. In the container pattern, the controller is the container object.

System::TAggregatedObject does not itself support any interfaces. However, as is typical of an aggregate, it does implement the methods of IInterface (see page 22396), which are used by the objects that descend from it. System::TAggregatedObject, therefore, serves as a base for classes that implement interfaces for creating objects that are part of an aggregate.

System::TAggregatedObject is used as a base for classes that create contained objects and connecting objects. Using System::TAggregatedObject ensures that calls to the IInterface (see page 22396) methods delegate to the controlling IInterface (see page 22396) of the aggregate.

The controlling IInterface (see page 22396) is specified in the constructor for System::TAggregatedObject and is indicated by the System::TAggregatedObject::Controller (see page 22251) property.

See Also

System::TContainedObject (see page 22253)

TConnectionPoint (see page 1570)

TConnectionPoints (see page 1575)

158.1.22.1 TAggregatedObject Members

The following tables list the members exposed by TAggregatedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22250)	This is _AddRef, a member of class TAggregatedObject.
	_Release (see page 22250)	This is _Release, a member of class TAggregatedObject.
	QueryInterface (see page 22250)	This is QueryInterface, a member of class TAggregatedObject.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22249)	Instantiates an aggregated object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Controller (see page 22251)	Provides access to the controlling interface of the aggregate.

158.1.22.2 TAggregatedObject::Create Constructor

Instantiates an aggregated object.

Pascal

```
constructor Create(const Controller: IInterface);
```

C++

```
Create(const IInterface Controller);
```

Description

Call System::TAggregatedObject::Create to instantiate an aggregated object at runtime.

System::TAggregatedObject::Create allocates memory for an aggregated object and sets the System::TAggregatedObject::Controller (see page 22251) property to the System::TAggregatedObject::Controller (see page 22251) parameter passed to it. The System::TAggregatedObject::Controller (see page 22251) is the controlling interface of the

aggregate. All other interfaces implemented in the aggregate delegate IInterface (see page 22396) method calls to this System::TAggregatedObject::Controller (see page 22251).

See Also

System::TAggregatedObject::Controller (see page 22251)

158.1.22.3 TAggregatedObject Methods

The methods of the TAggregatedObject class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 22250)	This is _AddRef, a member of class TAggregatedObject.
	_Release (see page 22250)	This is _Release, a member of class TAggregatedObject.
	QueryInterface (see page 22250)	This is QueryInterface, a member of class TAggregatedObject.

158.1.22.3.1 TAggregatedObject::_AddRef Method

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__stdcall Integer _AddRef();
```

Description

This is _AddRef, a member of class TAggregatedObject.

158.1.22.3.2 TAggregatedObject::_Release Method

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__stdcall Integer _Release();
```

Description

This is _Release, a member of class TAggregatedObject.

158.1.22.3.3 TAggregatedObject::QueryInterface Method

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__stdcall HResult QueryInterface(const TGUID IID, out  Obj);
```

Description

This is QueryInterface, a member of class TAggregatedObject.

158.1.22.4 TAggregatedObject Properties

The properties of the TAggregatedObject class are listed here.

Public Properties

	Name	Description
R	Controller (see page 22251)	Provides access to the controlling interface of the aggregate.

158.1.22.4.1 TAggregatedObject::Controller Property

Provides access to the controlling interface of the aggregate.

Pascal

```
property Controller: IInterface;
```

C++

```
__property IInterface Controller;
```

Description

System::TAggregatedObject::Controller is the interface reference of the aggregate. System::TAggregatedObject::TAggregatedObject (see page 22248) is used as part of an aggregate that has a single controlling IInterface (see page 22396). The IInterface (see page 22396) methods that are implemented in aggregated objects are delegated to the System::TAggregatedObject::Controller.

When the aggregated object is created, its constructor is passed a controlling IInterface (see page 22396). System::TAggregatedObject::Controller is initialized to this IInterface (see page 22396) in the constructor.

See Also

System::TAggregatedObject::Create (see page 22249)

158.1.23 TClassHelperBase Class

Class Hierarchy

Pascal

```
TClassHelperBase = class(TInterfacedObject, IInterface);
```

C++

```
class TClassHelperBase : public TInterfacedObject, public IInterface;
```

File

System

Description

This is class System.TClassHelperBase.

158.1.23.1 TClassHelperBase Members

The following tables list the members exposed by TClassHelperBase.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Fields

	Name	Description
	FInstance (see page 22253)	This is FInstance, a member of class TClassHelperBase.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Create (see page 22253)	This is _Create, a member of class TClassHelperBase.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

158.1.23.2 TClassHelperBase._Create Constructor

Pascal

```
constructor _Create(Instance: TObject);
```

C++

```
__fastcall TClassHelperBase(TObject * Instance);
```

Description

This is _Create, a member of class TClassHelperBase.

158.1.23.3 TClassHelperBase Fields

The fields of the TClassHelperBase class are listed here.

Protected Fields

	Name	Description
	FInstance (see page 22253)	This is FInstance, a member of class TClassHelperBase.

158.1.23.3.1 TClassHelperBase.FInstance Field

Pascal

```
FInstance: TObject;
```

C++

```
TObject * FInstance;
```

Description

This is FInstance, a member of class TClassHelperBase.

158.1.24 TContainedObject Class

System::TContainedObject implements the IInterface (see page 22396) interface as an inner object in an aggregate.

Class Hierarchy

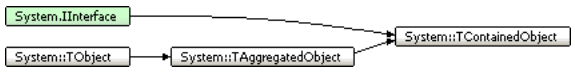

Pascal

```
TContainedObject = class(TAggregatedObject, IInterface);
```

C++

```
class TContainedObject : public TAggregatedObject, public IInterface;
```

File

System.pas

Description

System::TContainedObject can be used as a base for classes that create inner, or contained, objects of an aggregate. In Delphi, use TComObject or its descendants to instantiate an outer object that supports aggregation.

- The following implementation rules apply to aggregatable (inner or contained) objects, and are supported by System::TContainedObject:
- When an object of its type is created, the creation succeeds only for a requested interface of type IInterface (see page 22396).
- The IInterface (see page 22396) methods _AddRef (see page 22250) and _Release (see page 22250) delegate to the controlling IInterface (see page 22396).
- QueryInterface (see page 22255) is implemented in System::TContainedObject only for interfaces implemented in the contained object.
- The reference count on the inner object is incremented in the containing object.

Note: In Delphi, System::TContainedObject is used as a base class for a connection point that must be an inner object in an aggregate.

Note: An aggregated object is an object that is composed of several interfaced objects. Each object implements its own behavior and interfaces, but all of the objects share the same reference count, which is that of the controller object. In the container pattern, the controller is the container object.

See Also

TComObject (see page 5553)

TConnectionPoint (see page 1570)

Automation Servers

158.1.24.1 TContainedObject Members

The following tables list the members exposed by TContainedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22250)	This is _AddRef, a member of class TAggregatedObject.
	_Release (see page 22250)	This is _Release, a member of class TAggregatedObject.
V	QueryInterface (see page 22255)	Returns a reference to a specified interface if the object supports that interface.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22249)	Instantiates an aggregated object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Controller (see page 22251)	Provides access to the controlling interface of the aggregate.

158.1.24.2 TContainedObject Methods

The methods of the TContainedObject class are listed here.

Protected Methods

	Name	Description
V	QueryInterface (see page 22255)	Returns a reference to a specified interface if the object supports that interface.

158.1.24.2.1 TContainedObject::QueryInterface Method

Returns a reference to a specified interface if the object supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __stdcall HResult QueryInterface(const TGUID IID, out  Obj);
```

Description

QueryInterface checks whether the object that implements this interface supports the interface specified by IID. If so, QueryInterface

increments the reference count.

sets the Obj parameter so that it points to an instance of the specified interface.

returns 0 to indicate success.

If the object does not support the interface, QueryInterface returns a nonzero error code such as E_NoInterface.

See Also

_AddRef (see page 22397)

158.1.25 TDateTime Class

System::TDateTime is the C++ analog for the Delphi System::TDateTime data type.

Class Hierarchy

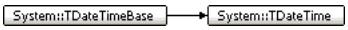

Pascal

```
TDateTime = class(TDateTimeBase);
```

C++

```
class TDateTime : public TDateTimeBase;
```

File

systdate.h

Description

System::TDateTime implements the Delphi System::TDateTime data type and the date/time runtime library routines that use the System::TDateTime data type.

The System::TDateTime class inherits a val data member declared as a double (see page 22274) that holds the date-time value. The integral part of a System::TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of a System::TDateTime value is the time of day.

Following are some examples of System::TDateTime values and their corresponding dates and times:

0 12/30/1899 12:00 am

2.75 1/1/1900 6:00 pm

-1.25 12/29/1899 6:00 am

35065 1/1/1996 12:00 am

To find the fractional number of days between two dates, simply subtract the two values, unless one of the System::TDateTime values is negative. Similarly, to increment a date and time value by a certain fractional number of days, add the fractional number to the date and time value if the System::TDateTime value is positive.

When working with negative System::TDateTime values, computations must handle time portion separately. The fractional part reflects the fraction of a 24-hour day without regard to the sign of the System::TDateTime value. For example, 6:00 am on

12/29/1899 is –1.25, not –1 + 0.25, which would be –0.75. There are no System::TDateTime values between –1 and 0.

Note: Use only the operators declared by System::TDateTime. The compiler ignores any operators you overload yourself.

Warning: It is not possible to use System::TDateTime as a base class for inheriting. To add functionality to System::TDateTime, you must write a wrapper class.

Note: To utilize the C++ streaming operators (<< and >>) with Currency (see page 22145), you must use #include <iostream> before #include <systdate.h>. This occurs automatically if you include vcl.h, clx.h, or System.hpp. The following stream operators are defined:

ostream& operator << (ostream& os, const System::TDateTime& arg);

istream& operator >> (istream& is, System::TDateTime& arg);

158.1.25.1 TDateTime Members

The following tables list the members exposed by TDateTime.

Enumerations

	Name	Description
	TDateTimeFlag (see page 22260)	This is record System::TDateTime::TDateTimeFlag.

Public Data Members

	Name	Description
	Val (see page 22275)	This is Val, a member of class TDateTimeBase.

Public Methods

	Name	Description
	CurrentDate (see page 22261)	Returns the current date as a System::TDateTime::TDateTime (see page 22259) value.
	CurrentDateTime (see page 22261)	Returns the current date and time as a System::TDateTime::TDateTime (see page 22259) value.
	CurrentTime (see page 22262)	Returns the current time as a System::TDateTime::TDateTime (see page 22259) value.
	DateString (see page 22262)	Converts the date of the System::TDateTime::TDateTime (see page 22259) value to a string.
	DateTimeString (see page 22262)	Converts the System::TDateTime::TDateTime (see page 22259) value to a string.
	DayOfWeek (see page 22263)	Returns the day of the week of the date portion of the System::TDateTime::TDateTime (see page 22259)
	DecodeDate (see page 22263)	Separates the System::TDateTime::TDateTime (see page 22259) value into year, month, and day values and stores these values in the year, month, and day parameters, respectively.
	DecodeTime (see page 22263)	Separates the System::TDateTime::TDateTime (see page 22259) value into hour, minute, second, and millisecond values and stores these values in the hour, min, sec, and msec parameters, respectively.
	FileDate (see page 22264)	Converts the date/time value to an OS date/time stamp.
	FileDateToDateTime (see page 22264)	Converts an OS timestamp value to System::TDateTime::TDateTime (see page 22259) value.
	FormatString (see page 22265)	Formats the System::TDateTime::TDateTime (see page 22259) object using the format given by format.
	TDateTime (see page 22258)	This is the overview for the TDateTime constructor overload.
	TimeString (see page 22266)	Converts the System::TDateTime::TDateTime (see page 22259) object to an AnsiString (see page 22273).

Public Operators

	Name	Description
	- (see page 22267)	This is the overview for the - operator overload.
	-- (see page 22268)	This is the overview for the -- operator overload.
	!= (see page 22268)	Compares two System::TDateTime::TDateTime (see page 22259) objects for inequality.
	+ (see page 22268)	This is the overview for the + operator overload.
	++ (see page 22269)	This is the overview for the ++ operator overload.
	+= (see page 22270)	This is the overview for the += operator overload.
	< (see page 22270)	Tests whether this System::TDateTime::TDateTime (see page 22259) value is less than another specified System::TDateTime::TDateTime (see page 22259) value.
	<= (see page 22271)	Tests whether this System::TDateTime::TDateTime (see page 22259) value is less than or equal to another specified System::TDateTime::TDateTime (see page 22259) value.
	= (see page 22271)	This is the overview for the = operator overload.
	-= (see page 22272)	This is the overview for the -= operator overload.
	== (see page 22273)	Compares two System::TDateTime::TDateTime (see page 22259) objects for equality.
	> (see page 22273)	Tests whether this System::TDateTime::TDateTime (see page 22259) value is greater than another specified System::TDateTime::TDateTime (see page 22259) value.
	>= (see page 22273)	Tests whether this System::TDateTime::TDateTime (see page 22259) value is greater than or equal to another specified System::TDateTime::TDateTime (see page 22259) value.
	AnsiString (see page 22273)	Converts the System::TDateTime::TDateTime (see page 22259) object to an AnsiString.
	double (see page 22274)	Converts the System::TDateTime::TDateTime (see page 22259) object to a double value.
	int (see page 22274)	Converts the System::TDateTime::TDateTime (see page 22259) object to an integer value.

158.1.25.2 TDateTime Constructor

This is the overview for the TDateTime constructor overload.

Overload List

	Name	Description
	TDateTime::TDateTime () (see page 22259)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 22256).
	TDateTime::TDateTime (AnsiString&, TDateTimeFlag) (see page 22259)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 22256).
	TDateTime::TDateTime (double) (see page 22259)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 22256).
	TDateTime::TDateTime (int) (see page 22259)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 22256).
	TDateTime::TDateTime (TDateTime&) (see page 22259)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 22256).
	TDateTime::TDateTime (TDateTimeBase&) (see page 22259)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 22256).
	TDateTime::TDateTime (unsigned short, unsigned short, unsigned short) (see page 22259)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 22256).

	TDateTime::TDateTime (unsigned short, unsigned short, unsigned short, unsigned short) (see page 22259)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 22256).
	TDateTime::TDateTime (unsigned short, unsigned short, unsigned short, unsigned short, unsigned short, unsigned short, unsigned short) (see page 22259)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 22256).

158.1.25.2.1 TDateTime::TDateTime Constructor ()

Creates and initializes an instance of System::TDateTime::TDateTime (see page 22256).

Pascal

```
constructor Create;
constructor Create(const src: TDateTimeBase&);
constructor Create(const src: TDateTime&);
constructor Create(const src: double);
constructor Create(const src: Integer);
constructor Create(const src: AnsiString&; flag: TDateTimeFlag = DateTime);
constructor Create(year: unsigned short; month: unsigned short; day: unsigned short);
constructor Create(hour: unsigned short; min: unsigned short; sec: unsigned short; msec:
unsigned short);
constructor Create(year: unsigned short; month: unsigned short; day: unsigned short; hour:
unsigned short; min: unsigned short; sec: unsigned short; msec: unsigned short);
```

C++

```
__fastcall TDateTime();
__fastcall TDateTime(const TDateTimeBase& src);
__fastcall TDateTime(const TDateTime& src);
__fastcall TDateTime(const double src);
__fastcall TDateTime(const int src);
__fastcall TDateTime(const AnsiString& src, TDateTimeFlag flag = DateTime);
__fastcall TDateTime(unsigned short year, unsigned short month, unsigned short day);
__fastcall TDateTime(unsigned short hour, unsigned short min, unsigned short sec, unsigned
short msec);
__fastcall TDateTime(unsigned short year, unsigned short month, unsigned short day, unsigned
short hour, unsigned short min, unsigned short sec, unsigned short msec);
```

Description

Use System::TDateTime::TDateTime (see page 22256) to instantiate a System::TDateTime::TDateTime (see page 22256) object.

When called with no arguments, the resulting System::TDateTime::TDateTime (see page 22256) object has a Val (see page 22543) data member of zero. This date and time corresponds to 12/30/1899 12:00 am.

When called with a System::TDateTime::TDateTime (see page 22256) object as the src parameter, the resulting object is a copy of src (the copy constructor).

When called with a System::TDateTime::TDateTimeBase (see page 22275) object as the src parameter, the resulting object has the same value as the src parameter. System::TDateTime::TDateTimeBase (see page 22275) is the base class for the System::TDateTime::TDateTime (see page 22256) type. It has a single member, a public Val (see page 22543) field that represents the value of the object.

When called with a double (see page 22274) or integer as the src parameter, the resulting object has the same value as the src parameter. The integral part of src is the number of days that have passed since 12/30/1899. The fractional part is the time of day, represented as a fraction of 24 hours.

When called with an AnsiString (see page 22273) as the src parameter and a System::TDateTime::TDateTimeFlag (see

page 22260) as the flag parameter, the resulting object's value is converted from the AnsiString (see page 22273). The flag parameter indicates whether the string contains a date, a time, or both. The default for the flag is DateTime.

When called with three unsigned shorts (as the year, month, and day parameters), the resulting object's value is assembled from the specified year, month, and day, with a time portion of 0 (midnight). The year must be between 1 and 9999. Valid month values are 1 through 12. Valid day values are 1 through 28, 29, 30, or 31, depending on the month value. For example, the possible day values for month 2 (February) are 1 through 28 or 1 through 29, depending on whether or not the year value specifies a leap year.

When called with four unsigned shorts (as the hour, min, sec, and msec parameters), the resulting object's value represents a time based on the specified number of hours, minutes, seconds, and milliseconds past midnight. The date portion of the resulting value is 0 (12/30/1899). The hour must be between 0 and 23, the minute must be between 0 and 59, the second must be between 0 and 59, and the millisecond must be between 0 and 999.

Note: When the constructor calls for a conversion that can't be performed (an AnsiString (see page 22273) that does not represent a date or time, or invalid year, month, day, hour, min, sec, or msec value), an EConvertError exception is thrown.

158.1.25.3 TDateTime Enumerations

The enumerations of the TDateTime class are listed here.

Enumerations

	Name	Description
	TDateTimeFlag (see page 22260)	This is record System::TDateTime::TDateTimeFlag.

158.1.25.3.1 System::TDateTime::TDateTimeFlag Enumeration

Pascal

```
TDateTimeFlag = (
  Date,
  Time,
  DateTime
);
```

C++

```
enum TDateTimeFlag {
  Date,
  Time,
  DateTime
};
```

File

systdate.h

Description

This is record System::TDateTime::TDateTimeFlag.

158.1.25.4 TDateTime Methods

The methods of the TDateTime class are listed here.

Public Methods

	Name	Description
	CurrentDate (see page 22261)	Returns the current date as a System::TDateTime::TDateTime (see page 22259) value.
	CurrentDateTime (see page 22261)	Returns the current date and time as a System::TDateTime::TDateTime (see page 22259) value.
	CurrentTime (see page 22262)	Returns the current time as a System::TDateTime::TDateTime (see page 22259) value.
	DateString (see page 22262)	Converts the date of the System::TDateTime::TDateTime (see page 22259) value to a string.
	DateTimeString (see page 22262)	Converts the System::TDateTime::TDateTime (see page 22259) value to a string.
	DayOfWeek (see page 22263)	Returns the day of the week of the date portion of the System::TDateTime::TDateTime (see page 22259)
	DecodeDate (see page 22263)	Separates the System::TDateTime::TDateTime (see page 22259) value into year, month, and day values and stores these values in the year, month, and day parameters, respectively.
	DecodeTime (see page 22263)	Separates the System::TDateTime::TDateTime (see page 22259) value into hour, minute, second, and millisecond values and stores these values in the hour, min, sec, and msec parameters, respectively.
	FileDate (see page 22264)	Converts the date/time value to an OS date/time stamp.
	FileDateToDateTime (see page 22264)	Converts an OS timestamp value to System::TDateTime::TDateTime (see page 22259) value.
	FormatString (see page 22265)	Formats the System::TDateTime::TDateTime (see page 22259) object using the format given by format.
	TimeString (see page 22266)	Converts the System::TDateTime::TDateTime (see page 22259) object to an AnsiString (see page 22273).

158.1.25.4.1 TDateTime::CurrentDate Method

Returns the current date as a System::TDateTime::TDateTime (see page 22259) value.

Pascal

```
function CurrentDate: TDateTime; static;
```

C++

```
static __fastcall TDateTime CurrentDate();
```

Description

System::TDateTime::CurrentDate retrieves the current date and returns it as a new System::TDateTime::TDateTime (see page 22259) object with a 0 time value.

See Also

System::TDateTime::CurrentDateTime (see page 22261)

System::TDateTime::CurrentTime (see page 22262)

158.1.25.4.2 TDateTime::CurrentDateTime Method

Returns the current date and time as a System::TDateTime::TDateTime (see page 22259) value.

Pascal

```
function CurrentDateTime: TDateTime; static;
```

C++

```
static __fastcall TDateTime CurrentDateTime();
```

Description

System::TDateTime::CurrentDateTime retrieves the current date and time, and returns it as a new System::TDateTime::TDateTime (see page 22259) object.

See Also

System::TDateTime::CurrentTime (see page 22262)

System::TDateTime::CurrentDate (see page 22261)

158.1.25.4.3 TDateTime::CurrentTime Method

Returns the current time as a System::TDateTime::TDateTime (see page 22259) value.

Pascal

```
function CurrentTime: TDateTime; static;
```

C++

```
static __fastcall TDateTime CurrentTime();
```

Description

System::TDateTime::CurrentTime retrieves the current time and returns it as a new System::TDateTime::TDateTime (see page 22259) object with a 0 date value.

See Also

System::TDateTime::CurrentDateTime (see page 22261)

System::TDateTime::CurrentDate (see page 22261)

158.1.25.4.4 TDateTime::DateString Method

Converts the date of the System::TDateTime::TDateTime (see page 22259) value to a string.

Pascal

```
function DateString: String;
```

C++

```
__fastcall AnsiString DateString() const;
```

Description

System::TDateTime::DateString returns the value of the System::TDateTime::TDateTime (see page 22259) object as an AnsiString (see page 22273). The conversion uses the format specified by the ShortDateFormat global variable.

See Also

System::TDateTime::DateTimeString (see page 22262)

System::TDateTime::TimeString (see page 22266)

variables

158.1.25.4.5 TDateTime::DateTimeString Method

Converts the System::TDateTime::TDateTime (see page 22259) value to a string.

Pascal

```
function DateTimeString: String;
```

C++

```
__fastcall AnsiString DateTimeString() const;
```

Description

System::TDateTime::DateTimeString returns an AnsiString (see page 22273) object containing the date and time value (the Val (see page 22543) data member) of the System::TDateTime::TDateTime (see page 22259) object to a string. System::TDateTime::DateTimeString converts the System::TDateTime::TDateTime (see page 22259) value using the format given by the ShortDateFormat global variable, followed by the time using the format given by the LongTimeFormat global variable. The time is not displayed if the fractional part of the DateTime value is zero.

See Also

System::TDateTime::DateString (see page 22262)

System::TDateTime::TimeString (see page 22266)

variables

158.1.25.4.6 TDateTime::DayOfWeek Method

Returns the day of the week of the date portion of the System::TDateTime::TDateTime (see page 22259)

Pascal

```
function DayOfWeek: Integer;
```

C++

```
__fastcall int DayOfWeek() const;
```

Description

System::TDateTime::DayOfWeek returns an integer between 1 and 7. Sunday is treated as the first day of the week and Saturday as the seventh.

158.1.25.4.7 TDateTime::DecodeDate Method

Separates the System::TDateTime::TDateTime (see page 22259) value into year, month, and day values and stores these values in the year, month, and day parameters, respectively.

Pascal

```
procedure DecodeDate(var year: unsigned short; var month: unsigned short; var day: unsigned
short);
```

C++

```
__fastcall void DecodeDate(unsigned short* year, unsigned short* month, unsigned short* day)
const;
```

Description

Use System::TDateTime::DecodeDate() when you need to access the year, month, or day of a System::TDateTime::TDateTime (see page 22259) object's date portion.

158.1.25.4.8 TDateTime::DecodeTime Method

Separates the System::TDateTime::TDateTime (see page 22259) value into hour, minute, second, and millisecond values and

stores these values in the hour, min, sec, and msec parameters, respectively.

Pascal

```
procedure DecodeTime(var hour: unsigned short; var min: unsigned short; var sec: unsigned
short; var msec: unsigned short);
```

C++

```
__fastcall void DecodeTime(unsigned short* hour, unsigned short* min, unsigned short* sec,
unsigned short* msec) const;
```

Description

Use System::TDateTime::DecodeTime() to access the hour, minute, second, or millisecond values of the System::TDateTime::TDateTime (see page 22259) object's time portion.

158.1.25.4.9 TDateTime::FileDate Method

Converts the date/time value to an OS date/time stamp.

Pascal

```
function FileDate: Integer;
```

C++

```
__fastcall int FileDate() const;
```

Description

Use System::TDateTime::FileDate to convert a System::TDateTime::TDateTime (see page 22259) value to an OS date/time stamp. For example, use System::TDateTime::FileDate to convert a System::TDateTime::TDateTime (see page 22259) value to a format that can be used as the Age parameter of FileSetDate. You may also need to use System::TDateTime::FileDate to convert a System::TDateTime::TDateTime (see page 22259) so that it is compatible with the Time field of the TSearchRec structure used by the FindFirst and FindNext functions.

See Also

System::TDateTime::FileDateToDateTime (see page 22264)

FileSetDate (see page 22878)

TSearchRec (see page 23005)

158.1.25.4.10 TDateTime::FileDateToDateTime Method

Converts an OS timestamp value to System::TDateTime::TDateTime (see page 22259) value.

Pascal

```
function FileDateToDateTime(fileDate: Integer): TDateTime; static;
```

C++

```
static __fastcall TDateTime FileDateToDateTime(int fileDate);
```

Description

A timestamp is a signed 32-bit integer, used by the OS to record information such as the date and time a file was modified. The precise format of a timestamp depends on the OS. Use System::TDateTime::FileDateToDateTime to convert a timestamp to a System::TDateTime::TDateTime (see page 22259) value.

For example, use System::TDateTime::FileDateToDateTime to convert the file timestamp returned by FileGetDate or FileAge. You may also need to use System::TDateTime::FileDateToDateTime to convert the Time field of the TSearchRec struct used by the FindFirst and FindNext functions.

See Also

FileGetDate (see page 22866)

FileAge (see page 22850)

TSearchRec (see page 23005)

158.1.25.4.11 TDateTime::FormatString Method

Formats the System::TDateTime::TDateTime (see page 22259) object using the format given by format.

Pascal

```
function FormatString(const format: AnsiString&): String;
```

C++

```
__fastcall AnsiString FormatString(const AnsiString& format) const;
```

Description

The following format specifiers are supported:

Specifier	Displays
c	Displays the date using the format given by the ShortDateFormat global variable, followed by the time using the format given by the LongTimeFormat global variable. The time is not displayed if the fractional part of the DateTime value is zero.
d	Displays the day as a number without a leading zero (1-31).
dd	Displays the day as a number with a leading zero (01-31).
ddd	Displays the day as an abbreviation (Sun-Sat) using the strings given by the ShortDayNames global variable.
dddd	Displays the day as a full name (Sunday-Saturday) using the strings given by the LongDayNames global variable.
ddddd	Displays the date using the format given by the ShortDateFormat global variable.
dddddd	Displays the date using the format given by the LongDateFormat global variable.
m	Displays the month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mm	Displays the month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mmm	Displays the month as an abbreviation (Jan-Dec (see page 22441)) using the strings given by the ShortMonthNames global variable.
mmmm	Displays the month as a full name (January-December) using the strings given by the LongMonthNames global variable.
yy	Displays the year as a two-digit number (00-99).
yyyy	Displays the year as a four-digit number (0000-9999).
h	Displays the hour without a leading zero (0-23).
hh	Displays the hour with a leading zero (00-23).
n	Displays the minute without a leading zero (0-59).
nn	Displays the minute with a leading zero (00-59).
s	Displays the second without a leading zero (0-59).
ss	Displays the second with a leading zero (00-59).
t	Displays the time using the format given by the ShortTimeFormat global variable.

tt	Displays the time using the format given by the LongTimeFormat global variable.
am/pm	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'am' for any hour before noon, and 'pm' for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'a' for any hour before noon, and 'p' for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm	Uses the 12-hour clock for the preceding h or hh specifier, and displays the contents of the TimeAMString global variable for any hour before noon, and the contents of the TimePMString global variable for any hour after noon.
/	Displays the date separator character given by the DateSeparator global variable.
:	Displays the time separator character given by the TimeSeparator global variable.
'xx'/"xx"	Characters enclosed in single or double (see page 22274) quotes are displayed as is, and do not affect formatting.

If the string given by the format parameter is empty, the System::TDateTime::TDateTime (see page 22259) value is formatted as if a c format specifier had been given.

158.1.25.4.12 TDateTime::TimeString Method

Converts the System::TDateTime::TDateTime (see page 22259) object to an AnsiString (see page 22273).

Pascal

```
function TimeString: String;
```

C++

```
__fastcall AnsiString TimeString() const;
```

Description

System::TDateTime::TimeString returns an AnsiString (see page 22273) object containing the time value (the fractional portion of the Val (see page 22543) data member) of the System::TDateTime::TDateTime (see page 22259) object to a string. The conversion uses the format specified by the LongTimeFormat variable. Change the format of how the string is displayed by changing the value of one or more of the date/time formatting variables.

See Also

System::TDateTime::DateString (see page 22262)

System::TDateTime::DateTimeString (see page 22262)

variables

158.1.25.5 TDateTime Operators

The operators of the TDateTime class are listed here.

Public Operators

	Name	Description
	- (see page 22267)	This is the overview for the - operator overload.
	-- (see page 22268)	This is the overview for the -- operator overload.
	!= (see page 22268)	Compares two System::TDateTime::TDateTime (see page 22259) objects for inequality.
	+ (see page 22268)	This is the overview for the + operator overload.
	++ (see page 22269)	This is the overview for the ++ operator overload.
	+= (see page 22270)	This is the overview for the += operator overload.

	< (see page 22270)	Tests whether this System::TDateTime::TDateTime (see page 22259) value is less than another specified System::TDateTime::TDateTime (see page 22259) value.
	<= (see page 22271)	Tests whether this System::TDateTime::TDateTime (see page 22259) value is less than or equal to another specified System::TDateTime::TDateTime (see page 22259) value.
	= (see page 22271)	This is the overview for the = operator overload.
	-= (see page 22272)	This is the overview for the -= operator overload.
	== (see page 22273)	Compares two System::TDateTime::TDateTime (see page 22259) objects for equality.
	> (see page 22273)	Tests whether this System::TDateTime::TDateTime (see page 22259) value is greater than another specified System::TDateTime::TDateTime (see page 22259) value.
	>= (see page 22273)	Tests whether this System::TDateTime::TDateTime (see page 22259) value is greater than or equal to another specified System::TDateTime::TDateTime (see page 22259) value.
	AnsiString (see page 22273)	Converts the System::TDateTime::TDateTime (see page 22259) object to an AnsiString.
	double (see page 22274)	Converts the System::TDateTime::TDateTime (see page 22259) object to a double value.
	int (see page 22274)	Converts the System::TDateTime::TDateTime (see page 22259) object to an integer value.

158.1.25.5.1 - Operator

This is the overview for the - operator overload.

Overload List

	Name	Description
	TDateTime::- (double) (see page 22267)	Subtracts a specified value from the System::TDateTime::TDateTime (see page 22259) object.
	TDateTime::- (int) (see page 22267)	Subtracts a specified value from the System::TDateTime::TDateTime (see page 22259) object.
	TDateTime::- (TDateTime&) (see page 22267)	Subtracts a specified value from the System::TDateTime::TDateTime (see page 22259) object.
	TDateTime::- (TDateTimeBase&) (see page 22267)	Subtracts a specified value from the System::TDateTime::TDateTime (see page 22259) object.

158.1.25.5.1.1 TDateTime::- Operator (TDateTime&)

Subtracts a specified value from the System::TDateTime::TDateTime (see page 22259) object.

Pascal

```
operator -(const rhs: TDateTimeBase&);
operator -(const rhs: TDateTime&);
operator -(const rhs: double);
operator -(const rhs: Integer);
```

C++

```
__fastcall TDateTime operator -(const TDateTimeBase& rhs) const;
__fastcall TDateTime operator -(const TDateTime& rhs) const;
__fastcall TDateTime operator -(const double rhs) const;
__fastcall TDateTime operator -(const int rhs) const;
```

Description

Use the System::TDateTime (see page 22256)::- to subtract the value specified by the rhs operand from the System::TDateTime::TDateTime (see page 22259) object. The rhs operand represents the number of days (including

fractional days) to subtract.

158.1.25.5.2 -- Operator

This is the overview for the -- operator overload.

Overload List

	Name	Description
	TDateTime::-- () (see page 22268)	Decrements the value of the System::TDateTime::TDateTime (see page 22259) object by a specified number of days.
	TDateTime::-- (int) (see page 22268)	Decrements the value of the System::TDateTime::TDateTime (see page 22259) object by a specified number of days.

158.1.25.5.2.1 TDateTime::-- Operator ()

Decrements the value of the System::TDateTime::TDateTime (see page 22259) object by a specified number of days.

Pascal

```
operator --;
operator --(: Integer);
```

C++

```
TDateTime& operator --();
TDateTime operator --(int);
```

Description

Use the unary System::TDateTime::operator_MinusMinus to decrement the value of the System::TDateTime::TDateTime (see page 22259) object by 1 day.

Use the binary System::TDateTime::operator_MinusMinus to decrement the value by the number of days specified by the integer operand.

158.1.25.5.3 TDateTime::!= Operator

Compares two System::TDateTime::TDateTime (see page 22259) objects for inequality.

Pascal

```
operator !=(const rhs: TDateTime&);
```

C++

```
__fastcall bool operator !=(const TDateTime& rhs) const;
```

Description

The != System::TDateTime (see page 22256)::!= returns true if the two objects do not have the same value, false otherwise.

158.1.25.5.4 + Operator

This is the overview for the + operator overload.

Overload List

	Name	Description
	TDateTime::+ (double) (see page 22269)	Adds a specified value to the System::TDateTime::TDateTime (see page 22259) object.
	TDateTime::+ (int) (see page 22269)	Adds a specified value to the System::TDateTime::TDateTime (see page 22259) object.
	TDateTime::+ (TDateTime&) (see page 22269)	Adds a specified value to the System::TDateTime::TDateTime (see page 22259) object.
	TDateTime::+ (TDateTimeBase&) (see page 22269)	Adds a specified value to the System::TDateTime::TDateTime (see page 22259) object.

158.1.25.5.4.1 TDateTime::+ Operator (TDateTime&)

Adds a specified value to the System::TDateTime::TDateTime (see page 22259) object.

Pascal

```
operator +(const rhs: TDateTimeBase&);
operator +(const rhs: TDateTime&);
operator +(const rhs: double);
operator +(const rhs: Integer);
```

C++

```
__fastcall TDateTime operator +(const TDateTimeBase& rhs) const;
__fastcall TDateTime operator +(const TDateTime& rhs) const;
__fastcall TDateTime operator +(const double rhs) const;
__fastcall TDateTime operator +(const int rhs) const;
```

Description

Use the + System::TDateTime (see page 22256)::+ to add the value specified by the rhs operand to the System::TDateTime::TDateTime (see page 22259) object. The rhs operand represents the number of days (including fractional days) to add.

158.1.25.5.5 ++ Operator

This is the overview for the ++ operator overload.

Overload List

	Name	Description
	TDateTime::++ () (see page 22269)	Increments the value of the System::TDateTime::TDateTime (see page 22259) object by a specified number of days.
	TDateTime::++ (int) (see page 22269)	Increments the value of the System::TDateTime::TDateTime (see page 22259) object by a specified number of days.

158.1.25.5.5.1 TDateTime::++ Operator ()

Increments the value of the System::TDateTime::TDateTime (see page 22259) object by a specified number of days.

Pascal

```
operator ++;
operator ++(: Integer);
```

C++

```
TDateTime& operator ++();
TDateTime operator ++(int);
```

Description

Use the unary ++ System::TDateTime (see page 22256)::+Plus to increment the value of the System::TDateTime::TDateTime (see page 22259) object by 1 day.

Use the binary ++ System::TDateTime (see page 22256)::+Plus to increment the value by the number of days specified by the integer operand.

158.1.25.5.6 += Operator

This is the overview for the += operator overload.

Overload List

	Name	Description
	TDateTime::+= (double) (see page 22270)	Increments the value of this System::TDateTime::TDateTime (see page 22259) object by a specified value.
	TDateTime::+= (int) (see page 22270)	Increments the value of this System::TDateTime::TDateTime (see page 22259) object by a specified value.
	TDateTime::+= (TDateTime&) (see page 22270)	Increments the value of this System::TDateTime::TDateTime (see page 22259) object by a specified value.
	TDateTime::+= (TDateTimeBase&) (see page 22270)	Increments the value of this System::TDateTime::TDateTime (see page 22259) object by a specified value.

158.1.25.5.6.1 TDateTime::+= Operator (TDateTime&)

Increments the value of this System::TDateTime::TDateTime (see page 22259) object by a specified value.

Pascal

```
operator +=(const rhs: TDateTimeBase&);
operator +=(const rhs: TDateTime&);
operator +=(const rhs: double);
operator +=(const rhs: Integer);
```

C++

```
__fastcall TDateTime& operator +=(const TDateTimeBase& rhs);
__fastcall TDateTime& operator +=(const TDateTime& rhs);
__fastcall TDateTime& operator +=(const double rhs);
__fastcall TDateTime& operator +=(const int rhs);
```

Description

Use the += System::TDateTime (see page 22256)::+= to add to the value specified by rhs to this System::TDateTime::TDateTime (see page 22259) value. The rhs operand represents the number of days (including fractional days) to add.

158.1.25.5.7 TDateTime::< Operator

Tests whether this System::TDateTime::TDateTime (see page 22259) value is less than another specified System::TDateTime::TDateTime (see page 22259) value.

Pascal

```
operator <(const rhs: TDateTime&);
```

C++

```
__fastcall bool operator <(const TDateTime& rhs) const;
```

Description

The < System::TDateTime (see page 22256)::< returns true if this System::TDateTime::TDateTime (see page 22259) object has a value (Val (see page 22543) data member) less than that of the object specified by rhs, and false otherwise.

158.1.25.5.8 TDateTime::<= Operator

Tests whether this System::TDateTime::TDateTime (see page 22259) value is less than or equal to another specified System::TDateTime::TDateTime (see page 22259) value.

Pascal

```
operator <=(const rhs: TDateTime&);
```

C++

```
__fastcall bool operator <=(const TDateTime& rhs) const;
```

Description

The <= System::TDateTime (see page 22256)::<= returns true if this System::TDateTime::TDateTime (see page 22259) object has a value (Val (see page 22543) data member) less than or equal to that of the object specified by rhs, and false otherwise.

158.1.25.5.9 = Operator

This is the overview for the = operator overload.

Overload List

	Name	Description
	TDateTime::= (double) (see page 22271)	Assigns the value of the System::TDateTime::TDateTime (see page 22259) object.
	TDateTime::= (int) (see page 22271)	Assigns the value of the System::TDateTime::TDateTime (see page 22259) object.
	TDateTime::= (TDateTime&) (see page 22271)	Assigns the value of the System::TDateTime::TDateTime (see page 22259) object.
	TDateTime::= (TDateTimeBase&) (see page 22271)	Assigns the value of the System::TDateTime::TDateTime (see page 22259) object.

158.1.25.5.9.1 TDateTime::= Operator (TDateTime&)

Assigns the value of the System::TDateTime::TDateTime (see page 22259) object.

Pascal

```
operator =(const rhs: TDateTimeBase&);
operator =(const rhs: TDateTime&);
operator =(const rhs: double);
operator =(const rhs: Integer);
```

C++

```
__fastcall TDateTime& operator =(const TDateTimeBase& rhs);
__fastcall TDateTime& operator =(const TDateTime& rhs);
__fastcall TDateTime& operator =(const double rhs);
__fastcall TDateTime& operator =(const int rhs);
```

Description

The assignment System::TDateTime (see page 22256)::= assigns the value of the rhs operand to the System::TDateTime::TDateTime (see page 22259) object. The assignment System::TDateTime (see page 22256)::= can take

a double (see page 22274) or integer value, which is assigned to the System::TDateTime::TDateTime (see page 22259) object such that the integral part of rhs is the number of days that have passed since 12/30/1899 and the fractional part is the time of day, represented as a fraction of 24 hours.

a System::TDateTime::TDateTimeBase (see page 22275) or System::TDateTime::TDateTime (see page 22259) object, whose value is assigned to the System::TDateTime::TDateTime (see page 22259) object. (Note that System::TDateTime::TDateTimeBase (see page 22275) is the base class for the System::TDateTime::TDateTime (see page 22259) class. It has a single member, a public Val (see page 22543) field that represents the value of the object).

158.1.25.5.10 -= Operator

This is the overview for the -= operator overload.

Overload List

	Name	Description
	TDateTime::-= (double) (see page 22272)	Decrements the value of this System::TDateTime::TDateTime (see page 22259) object by a specified value.
	TDateTime::-= (int) (see page 22272)	Decrements the value of this System::TDateTime::TDateTime (see page 22259) object by a specified value.
	TDateTime::-= (TDateTime&) (see page 22272)	Decrements the value of this System::TDateTime::TDateTime (see page 22259) object by a specified value.
	TDateTime::-= (TDateTimeBase&) (see page 22272)	Decrements the value of this System::TDateTime::TDateTime (see page 22259) object by a specified value.

158.1.25.5.10.1 TDateTime::-= Operator (TDateTime&)

Decrements the value of this System::TDateTime::TDateTime (see page 22259) object by a specified value.

Pascal

```
operator -=(const rhs: TDateTimeBase&);
operator -=(const rhs: TDateTime&);
operator -=(const rhs: double);
operator -=(const rhs: Integer);
```

C++

```
__fastcall TDateTime& operator -=(const TDateTimeBase& rhs);
__fastcall TDateTime& operator -=(const TDateTime& rhs);
__fastcall TDateTime& operator -=(const double rhs);
__fastcall TDateTime& operator -=(const int rhs);
```

Description

Use the -= System::TDateTime (see page 22256)::-= to subtract the value specified by rhs from this

System::TDateTime::TDateTime (see page 22259) value. The rhs operand represents the number of days (including fractional days) to subtract.

158.1.25.5.11 TDateTime::== Operator

Compares two System::TDateTime::TDateTime (see page 22259) objects for equality.

Pascal

```
operator ==(const rhs: TDateTime&);
```

C++

```
__fastcall bool operator ==(const TDateTime& rhs) const;
```

Description

The == System::TDateTime (see page 22256)::== returns true if the two objects have the same value, false otherwise.

158.1.25.5.12 TDateTime::> Operator

Tests whether this System::TDateTime::TDateTime (see page 22259) value is greater than another specified System::TDateTime::TDateTime (see page 22259) value.

Pascal

```
operator >(const rhs: TDateTime&);
```

C++

```
__fastcall bool operator >(const TDateTime& rhs) const;
```

Description

The > System::TDateTime (see page 22256)::> returns true if this System::TDateTime::TDateTime (see page 22259) object has a value (Val (see page 22543) data member) greater than that of the object specified by rhs, and false otherwise.

158.1.25.5.13 TDateTime::>= Operator

Tests whether this System::TDateTime::TDateTime (see page 22259) value is greater than or equal to another specified System::TDateTime::TDateTime (see page 22259) value.

Pascal

```
operator >=(const rhs: TDateTime&);
```

C++

```
__fastcall bool operator >=(const TDateTime& rhs) const;
```

Description

The >= System::TDateTime (see page 22256)::>= returns true if this System::TDateTime::TDateTime (see page 22259) object has a value (Val (see page 22543) data member) greater than or equal to that of the object specified by rhs, and false otherwise.

158.1.25.5.14 TDateTime::AnsiString Operator

Converts the System::TDateTime::TDateTime (see page 22259) object to an AnsiString.

Pascal

```
operator AnsiString;
```

C++

```
__fastcall operator AnsiString() const;
```

Description

The AnsiString conversion System::TDateTime::AnsiString returns an AnsiString object that is the string conversion of this System::TDateTime::TDateTime (see page 22259) object. The conversion uses the format specified by the LongTimeFormat global variable of the System::TDateTime::TDateTime (see page 22259) formatting variables. Change the format of the string display by changing the values of some of the date and time variables.

See Also

System::TDateTime::DateTimeString (see page 22262)

variables

158.1.25.5.15 TDateTime::double Operator

Converts the System::TDateTime::TDateTime (see page 22259) object to a double value.

Pascal

```
operator double;
```

C++

```
__fastcall operator double() const;
```

Description

The double conversion System::TDateTime::double returns a double that represents this System::TDateTime::TDateTime (see page 22259) object. The integral part of the double is the number of days that have passed since 12/30/1899. The fractional part is the time of day, represented as a fraction of 24 hours.

See Also

System::TDateTime::TDateTime (see page 22259)

158.1.25.5.16 TDateTime::int Operator

Converts the System::TDateTime::TDateTime (see page 22259) object to an integer value.

Pascal

```
operator int;
```

C++

```
__fastcall operator int() const;
```

Description

The int conversion System::TDateTime::int returns an int that is the integer conversion of the date portion of this System::TDateTime::TDateTime (see page 22259) object. The resulting integer is the number of days that have passed since 12/30/1899.

See Also

System::TDateTime::TDateTime (see page 22259)

158.1.26 TDateTimeBase Class

Class Hierarchy

System::TDateTimeBase

Pascal

```
TDateTimeBase = class;
```

C++

```
class TDateTimeBase;
```

File

systdate.h

Description

This is class System::TDateTimeBase.

158.1.26.1 TDateTimeBase Members

The following tables list the members exposed by TDateTimeBase.

Public Data Members

	Name	Description
	Val (see page 22275)	This is Val, a member of class TDateTimeBase.

158.1.26.2 TDateTimeBase Data Members

The data members of the TDateTimeBase class are listed here.

Public Data Members

	Name	Description
	Val (see page 22275)	This is Val, a member of class TDateTimeBase.

158.1.26.2.1 TDateTimeBase::Val Data Member

Pascal

```
Val: double;
```

C++

```
double Val;
```

Description

This is Val, a member of class TDateTimeBase.

158.1.27 TInterfacedObject Class

System::TInterfacedObject is a base for simple classes that need a basic IInterface (see page 22396) implementation.

Class Hierarchy

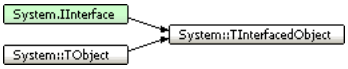

Pascal

```
TInterfacedObject = class(TObject, IInterface);
```

C++

```
class TInterfacedObject : public TObject, public IInterface;
```

File

System.pas

Description

System::TInterfacedObject is a thread-safe base class for Delphi classes that support interfaces. System::TInterfacedObject provides basic reference-counting functionality that makes its descendant classes useful in both Delphi and C++ code.

Note: In Delphi, variables declared to be of type System::TInterfacedObject have dynamic-binding, meaning that an instance of a descendant class can be assigned to that variable at runtime. This is because System::TInterfacedObject implements System::TInterfacedObject::QueryInterface (see page 22280), which is called at runtime to query whether the instantiated object supports the requested interface. If it does, then the descendant object is successfully assigned to the variable. Without System::TInterfacedObject::QueryInterface (see page 22280) support, such an assignment would produce a compile-time error.

Note: In Delphi, a descendant of System::TInterfacedObject does not have a class factories or a CLSID. Thus it cannot be instantiated except by calling a constructor.

Note: System::TInterfacedObject does not implement the methods of the C++ version of IUnknown (see page 22586) or IInterface (see page 22396). This is not a problem when using C++ wrappers for System::TInterfacedObject descendants that are written in Delphi. However, if you are creating C++ descendants of System::TInterfacedObject, you must provide your own implementation of IUnknown (see page 22586). For information on how to do this, see Creating classes that support IUnknown (see page 22586) (supplied with C++ products only).

158.1.27.1 TInterfacedObject Members

The following tables list the members exposed by TInterfacedObject.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

158.1.27.2 TInterfacedObject Data Members

The data members of the TInterfacedObject class are listed here.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

158.1.27.2.1 TInterfacedObject::FRefCount Data Member

Pascal

```
FRefCount: Integer;
```

C++

```
Integer FRefCount;
```

Description

This is FRefCount, a member of class TInterfacedObject.

158.1.27.3 TInterfacedObject Methods

The methods of the TInterfacedObject class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.

158.1.27.3.1 TInterfacedObject::_AddRef Method

Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__stdcall Integer _AddRef();
```

Description

Do not call the protected System::TInterfacedObject::_AddRef method directly. System::TInterfacedObject::_AddRef is called through the interface to indicate that another object is using the interfaced object's interface pointer. When the other object is through with the interface, it calls System::TInterfacedObject::_Release (see page 22279) through the same interface. This allows the interfaced object to free itself when it is no longer used (when the reference count goes to zero).

System::TInterfacedObject::_AddRef increments the System::TInterfacedObject::RefCount (see page 22281) property.

See Also

System::TInterfacedObject::RefCount (see page 22281)

System::TInterfacedObject::_Release (see page 22279)

158.1.27.3.2 TInterfacedObject::_Release Method

Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__stdcall Integer _Release();
```

Description

Do not call the protected _System::TInterfacedObject::_Release method directly. _System::TInterfacedObject::_Release is called through the IInterface (see page 22396) interface to indicate that another object is through with the interfaced object's interface pointer. This allows the interfaced object to free itself when it is no longer used (when the reference count goes to zero).

_System::TInterfacedObject::_Release decrements the System::TInterfacedObject::RefCount (see page 22281) property.

See Also

System::TInterfacedObject::RefCount (see page 22281)

System::TInterfacedObject::_AddRef (see page 22278)

System::TInterfacedObject::BeforeDestruction (see page 22279)

158.1.27.3.3 TInterfacedObject::AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; virtual; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject (see page 22276) is allocated, System::TInterfacedObject::RefCount (see page 22281) is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount (see page 22281) to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 22279)

System::TInterfacedObject::NewInstance (see page 22280)

System::TInterfacedObject::RefCount (see page 22281)

158.1.27.3.4 TInterfacedObject::BeforeDestruction Method

Checks that the reference count is zero before allowing the interfaced object to be destroyed.

Pascal

```
procedure BeforeDestruction; virtual; override;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

System::TInterfacedObject::BeforeDestruction is called automatically immediately before the interfaced object's destructor. It checks that the reference count is zero, and, if it is not, raises an exception.

See Also

System::TInterfacedObject::RefCount (see page 22281)

System::TObject::free

158.1.27.3.5 TInterfacedObject::NewInstance Method

Allocates memory for each instance of an interfaced object.

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
TObject NewInstance();
```

Description

All constructors call System::TInterfacedObject::NewInstance automatically. System::TInterfacedObject::NewInstance calls System::TObject::InstanceSize (see page 22306) to determine how much memory to allocate from the heap to contain a particular instance. Do not call System::TInterfacedObject::NewInstance directly.

System::TInterfacedObject::TInterfacedObject (see page 22276) overrides System::TInterfacedObject::NewInstance to increment the reference count when a new instance is created. This prevents any constructor from accidentally deleting the instance due to a System::TInterfacedObject::RefCount (see page 22281) of zero. The System::TInterfacedObject::AfterConstruction (see page 22279) method decrements the value of System::TInterfacedObject::RefCount (see page 22281) after all constructors have executed.

See Also

System::TInterfacedObject::RefCount (see page 22281)

System::TObject::NewInstance (see page 22306)

System::TObject::InstanceSize (see page 22306)

System::TInterfacedObject::AfterConstruction (see page 22279)

158.1.27.3.6 TInterfacedObject::QueryInterface Method

Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__stdcall HResult QueryInterface(const TGUID IID, out  Obj);
```

Description

Do not call the protected System::TInterfacedObject::QueryInterface method directly. System::TInterfacedObject::QueryInterface is called through IInterface (see page 22396) to obtain an interface pointer for the interface identified by the IID parameter. If the object supports the requested interface, it is returned in the Obj parameter and System::TInterfacedObject::QueryInterface returns S_OK (see page 22677). If the object does not support the interface, System::TInterfacedObject::QueryInterface returns E_NOINTERFACE (see page 22663).

If System::TInterfacedObject::QueryInterface successfully returns an interface pointer, it automatically increments the reference count.

See Also

System::TInterfacedObject::_AddRef (see page 22278)

158.1.27.4 TInterfacedObject Properties

The properties of the TInterfacedObject class are listed here.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

158.1.27.4.1 TInterfacedObject::RefCount Property

Indicates the number of interface pointers currently dependent upon the object.

Pascal

```
property RefCount: Integer;
```

C++

```
__property Integer RefCount;
```

Description

System::TInterfacedObject::RefCount defines the lifetime of the object. When System::TInterfacedObject::RefCount is zero, the object is destroyed.

System::TInterfacedObject::RefCount is incremented by calls to the _AddRef (see page 22278) method and decremented by the _Release (see page 22279) method.

Note: If the destructor is called on a System::TInterfacedObject::TInterfacedObject (see page 22276) descendant that is not fully constructed (an exception was raised in the constructor), then System::TInterfacedObject::RefCount will have a value of 1. Otherwise, System::TInterfacedObject::RefCount will be zero.

158.1.28 TMetaClass Class

System::TMetaClass is the C++ representation of the Delphi class-reference type.

Class Hierarchy

System::TMetaClass

Pascal

```
TMetaClass = class;
```

C++

```
class TMetaClass;
```

File

systobj.h

Description

System::TMetaClass allows operations to be performed directly on classes. This contrasts with class types, which can be instantiated as objects and which allow operations to be performed on those instances. The System::TMetaClass for a given class can be acquired by using the __classid keyword extension.

A variable of type System::TMetaClass can be NULL, which indicates that the variable does not currently reference a class.

Every class inherits a method function called ClassType, which returns a reference to the class of an object. The type of the value returned by ClassType is TClass (see page 22611) (System::TMetaClass*), which is a pointer to a System::TMetaClass.

The methods of System::TMetaClass correspond to the static methods of TObject (see page 22285). They have the same functionality but differ in parameters. Whereas the TObject (see page 22285) methods take a TClass (see page 22611) (System::TMetaClass*) as the first parameter, this parameter is implicit in the System::TMetaClass methods.

C++ Examples:

```
/*
The following example allows users to type the name of a
graphics class into an edit control and when a button is
pressed, the name of the default extension for that class is
displayed in another edit control.  In the form's OnCreate
event handler (or a similar place) you must register the
graphics classes so that GetClass can find them.
*/
TMetaClass *MetaClass;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClassFinder *myClassFinder = new TClassFinder(MetaClass, False);
  TGraphicClass mygraphclass = (TGraphicClass) (myClassFinder->GetClass(Edit2->Text));
//  TGraphicClass mygraphclass = (TGraphicClass) GetClass(Edit2->Text);
  Edit1->Text = GraphicExtension(mygraphclass);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MetaClass = __classid(TIcon); // ico
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(Graphics::TBitmap); // bmp
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TMetafile); // emf
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TShape);  // nographic extension
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TImage); // nographic extension
  RegisterClasses(&MetaClass, 0);
}
```

See Also

TObject

158.1.28.1 TMetaClass Members

The following tables list the members exposed by TMetaClass.

Public Methods

	Name	Description
	ClassInfo (see page 22283)	This is ClassInfo, a member of class TMetaClass.
	ClassName (see page 22283)	This is ClassName, a member of class TMetaClass.
	ClassNameIs (see page 22284)	This is ClassNameIs, a member of class TMetaClass.
	ClassParent (see page 22284)	This is ClassParent, a member of class TMetaClass.
	InheritsFrom (see page 22284)	This is InheritsFrom, a member of class TMetaClass.
	InitInstance (see page 22284)	This is InitInstance, a member of class TMetaClass.
	InstanceSize (see page 22285)	This is InstanceSize, a member of class TMetaClass.
	MethodAddress (see page 22285)	This is MethodAddress, a member of class TMetaClass.
	MethodName (see page 22285)	This is MethodName, a member of class TMetaClass.

158.1.28.2 TMetaClass Methods

The methods of the TMetaClass class are listed here.

Public Methods

	Name	Description
	ClassInfo (see page 22283)	This is ClassInfo, a member of class TMetaClass.
	ClassName (see page 22283)	This is ClassName, a member of class TMetaClass.
	ClassNameIs (see page 22284)	This is ClassNameIs, a member of class TMetaClass.
	ClassParent (see page 22284)	This is ClassParent, a member of class TMetaClass.
	InheritsFrom (see page 22284)	This is InheritsFrom, a member of class TMetaClass.
	InitInstance (see page 22284)	This is InitInstance, a member of class TMetaClass.
	InstanceSize (see page 22285)	This is InstanceSize, a member of class TMetaClass.
	MethodAddress (see page 22285)	This is MethodAddress, a member of class TMetaClass.
	MethodName (see page 22285)	This is MethodName, a member of class TMetaClass.

158.1.28.2.1 TMetaClass::ClassInfo Method

Pascal

```
function ClassInfo: Pointer;
```

C++

```
__fastcall void * ClassInfo();
```

Description

This is ClassInfo, a member of class TMetaClass.

158.1.28.2.2 TMetaClass::ClassName Method

Pascal

```
function ClassName: ShortString;
```

C++

```
__fastcall ShortString ClassName();
```

Description

This is ClassName, a member of class TMetaClass.

158.1.28.2.3 TMetaClass::ClassNameIs Method

Pascal

```
function ClassNameIs(const string: AnsiString &): bool;
```

C++

```
__fastcall bool ClassNameIs(const AnsiString & string);
```

Description

This is ClassNameIs, a member of class TMetaClass.

158.1.28.2.4 TMetaClass::ClassParent Method

Pascal

```
function ClassParent: TClass;
```

C++

```
__fastcall TClass ClassParent();
```

Description

This is ClassParent, a member of class TMetaClass.

158.1.28.2.5 TMetaClass::InheritsFrom Method

Pascal

```
function InheritsFrom(aClass: TClass): bool;
```

C++

```
__fastcall bool InheritsFrom(TClass aClass);
```

Description

This is InheritsFrom, a member of class TMetaClass.

158.1.28.2.6 TMetaClass::InitInstance Method

Pascal

```
function InitInstance(var instance): TObject;
```

C++

```
__fastcall TObject * InitInstance(void * instance);
```

Description

This is InitInstance, a member of class TMetaClass.

158.1.28.2.7 TMetaClass::InstanceSize Method

Pascal

```
function InstanceSize: long;
```

C++

```
__fastcall long InstanceSize();
```

Description

This is InstanceSize, a member of class TMetaClass.

158.1.28.2.8 TMetaClass::MethodAddress Method

Pascal

```
function MethodAddress(const Name: ShortString &): Pointer;
```

C++

```
__fastcall void * MethodAddress(const ShortString & Name);
```

Description

This is MethodAddress, a member of class TMetaClass.

158.1.28.2.9 TMetaClass::MethodName Method

Pascal

```
function MethodName(var Address): ShortString;
```

C++

```
__fastcall ShortString MethodName(void * Address);
```

Description

This is MethodName, a member of class TMetaClass.

158.1.29 TObject Class

System::TObject is the ultimate ancestor of all objects and components.

Class Hierarchy

System::TObject

Pascal

```
TObject = class;
```

C++

```
class TObject;
```

File

System.pas

Description

System::TObject encapsulates fundamental behavior common to objects by introducing methods that

create, maintain and destroy instances of the object by allocating, initializing, and freeing required memory.

respond when object instances are created or destroyed.

return class-type and instance information on an object and runtime type information (RTTI) about its published properties.

support message handling.

support interfaces that the object implements.

Use System::TObject as an immediate base class when declaring simple objects that do not need to persist (are not saved and reloaded) and that do not need to be assigned to other objects.

Much of the capability of objects is established by methods that System::TObject introduces. Many of these methods are used internally by IDEs and are not intended for users to call directly. Others are overridden in descendant objects that have more complex behavior.

Although System::TObject is the based object of a component framework, not all objects are components. All component classes are descended from TComponent.

Note: System::TObject is never directly instantiated. Although it does not use programming language features that prevent instantiation, System::TObject is an abstract class.

See Also

TComponent (see page 2762)

TPersistent (see page 2928)

TObject Branch

TWinControl/TWidgetControl Branch

VCL Overview

158.1.29.1 TObject Members

The following tables list the members exposed by TObject.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

158.1.29.2 TObject::Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
Create();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject (see page 22285) allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

158.1.29.3 TObject::Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; virtual;
```

C++

```
virtual Destroy();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free (see page 22300) instead. Free (see page 22300) verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject (see page 22285) deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free (see page 22300) on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free (see page 22300) provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

158.1.29.4 TObject Methods

The methods of the TObject class are listed here.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

158.1.29.4.1 TObject::AfterConstruction Method

Responds after the last constructor has executed.

Pascal

```
procedure AfterConstruction; virtual;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

The System::TObject::AfterConstruction method implemented in System::TObject::TObject (see page 22285) does nothing. Override this method when creating a class that takes some action after the object is created. For example, TCustomForm overrides System::TObject::AfterConstruction to generate an OnCreate event.

See Also

System::TObject::BeforeDestruction (see page 22289)

OnCreate (see page 13951)

System::TObject::Create (see page 22287)

158.1.29.4.2 TObject::BeforeDestruction Method

Responds before the first destructor executes.

Pascal

```
procedure BeforeDestruction; virtual;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

System::TObject::BeforeDestruction is called automatically before the object's first destructor executes. Do not call it explicitly in your applications.

The System::TObject::BeforeDestruction method implemented in System::TObject::TObject (see page 22285) does nothing. Override this method when creating a class that takes some action before the object is destroyed. For example, TCustomForm overrides System::TObject::BeforeDestruction to generate an OnDestroy event.

Note: System::TObject::BeforeDestruction is not called when the object is destroyed before it is fully constructed. That is, if the object's constructor raises an exception, the destructor is called to dispose of the object, but System::TObject::BeforeDestruction is not called.

See Also

System::TObject::AfterConstruction (see page 22289)

OnDestroy (see page 13953)

System::TObject::Destroy (see page 22287)

158.1.29.4.3 TObject::ClassInfo Method

Returns a pointer to the runtime type information (RTTI) table for the object type.

Pascal

```
class function ClassInfo: Pointer;
```

C++

```
Pointer ClassInfo();
```

Description

System::TObject::ClassInfo provides access to the RTTI table for a given object type.

Note: The format of the RTTI table is subject to change. Applications should use the other RTTI methods provided by System::TObject::TObject (see page 22285); these provide easy, consistent access to RTTI data.

Note: Some classes do not provide runtime type information. For these classes, System::TObject::ClassInfo returns nil (Delphi) or NULL (C++). All classes descended from TPersistent do provide runtime type information.

Note: The Delphi compiler does not provide RTTI by default. Most classes implemented in Delphi that need to provide RTTI are descended from TPersistent. If your Delphi class needs RTTI but not the full functionality of TPersistent, compile the code with the $M switch.

Note: The C++ compiler provides RTTI by default. This feature can be disabled as project option and selectively re-enabled using the __rtti keyword.

See Also

System::TObject::ClassType (see page 22295)

System::TObject::ClassName (see page 22290)

System::TObject::ClassParent (see page 22293)

System::TObject::FieldAddress (see page 22299)

System::TObject::MethodAddress (see page 22306)

System::TObject::InstanceSize (see page 22306)

System::TObject::MethodName (see page 22306)

System::TObject::InheritsFrom (see page 22305)

_typeinfo_Macro

158.1.29.4.4 TObject::ClassName Method

Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

Pascal

```
class function ClassName: ShortString;
```

C++

```
ShortString ClassName();
```

Description

Use System::TObject::ClassName to obtain the class name from an object instance or class reference. This is useful for differentiating object instances that are assigned to a variable that has the type of an ancestor class.

Note: In C++ code, call System::TObject::ClassName as a method to obtain an object's class name. Use the global static function to obtain the class name from a metaclass object.

C++ Examples:

```
/*
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button's class and the names of its
parent classes are added to the list box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClass ClassRef;
  ListBox1->Clear();
  ClassRef = Sender->ClassType();
  while (ClassRef != NULL)
  {
    ListBox1->Items->Add(ClassName(ClassRef));
    ClassRef = ClassParent(ClassRef);
  };
}
/*
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
*/
/*
To run this example, add the example code to a new project.
The example code dynamically creates a TCoolbar and three
TCoolBand objects populated with a windowed control in each
TCoolBand.
*/
void AddBands(TCoolBar *CoolBar, TList *Objects)
{
  TControl *pCurrent;
  for (int i = 0; i < Objects->Count; i++)
  {
    pCurrent = (TControl *)Objects->Items[i];
    // only add windowed controls to the coolbar
    if (dynamic_cast<TWinControl *>(pCurrent) != NULL)
    {
      // Placing the control onto the CoolBar
      // causes the TCoolBar object to create a TCoolBand
      // and place the control within the band.
      pCurrent->Parent = CoolBar;   // This statement increments CoolBar->Bands->Count.
      // Get the reference of the TCoolBand just created and assign text.
      String S = pCurrent->ClassName(pCurrent->ClassType());
      CoolBar->Bands->Items[CoolBar->Bands->Count - 1]->Text = S;
```

```
      }
    }
  }

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    TCoolBar *CoolBar = new TCoolBar(this);
    TList *List;
    CoolBar->Parent = this;
    CoolBar->Align = alTop;
    List = new TList();
    try
    {
      TControl *Control;
      Control = new TButton(CoolBar);
      List->Add(Control);
      Control = new TCheckBox(CoolBar);
      List->Add(Control);
      Control = new TEdit(CoolBar);
      List->Add(Control);
      AddBands(CoolBar, List);
    }
    catch (...)
    {
      ShowMessage("Some objects could not be added to Coolband");
    }
    delete List;
  }
```

Delphi Examples:

```
  {
  This example shows how to obtain the ancestry of a component
  using the ClassType and ClassParent properties. It uses a
  button and a list box on a form. When the user clicks the
  button, the name of the button's class and the names of its
  parent classes are added to the list box.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    ClassRef: TClass;
  begin
    ListBox1.Clear;
    ClassRef := Sender.ClassType;
    while ClassRef <> nil do
    begin
      ListBox1.Items.Add(ClassRef.ClassName);
      ClassRef := ClassRef.ClassParent;
    end;
  end;
  {
  The list box contains the following strings after the user clicks the button:
      TButton
      TButtonControl
      TWinControl or TWinControl
      TControl
      TComponent
      TPersistent
      TObject
  }
```

See Also

System::TObject::ClassNameIs (see page 22293)

System::TObject::ClassParent (see page 22293)

System::TObject::ClassType (see page 22295)

158.1.29.4.5 TObject::ClassNameIs Method

Determines whether an object is of a specific type.

Pascal

```
class function ClassNameIs(const Name: string): Boolean;
```

C++

```
Boolean ClassNameIs(const string Name);
```

Description

System::TObject::ClassNameIs determines whether an object instance or class reference has a class name that matches a specified string. This is useful to query objects across modules or shared libraries.

Note: In C++ code, call System::TObject::ClassNameIs as a method to compare an object's class name. Use the global static function to compare the class name from a metaclass object.

See Also

System::TObject::ClassName (see page 22290)

System::TObject::InheritsFrom (see page 22305)

158.1.29.4.6 TObject::ClassParent Method

Returns the type of the immediate ancestor of a class.

Pascal

```
class function ClassParent: TClass;
```

C++

```
TClass ClassParent();
```

Description

System::TObject::ClassParent returns the name of the parent class for an object instance or class reference. For System::TObject::TObject (see page 22285), System::TObject::ClassParent returns nil (Delphi) or NULL (C++).

Avoid using System::TObject::ClassParent in application code.

Note: In Delphi code, use the is or as operators instead of System::TObject::ClassParent.

Note: In C++ code, use a dynamic cast or the System::TObject::InheritsFrom (see page 22305) method instead of System::TObject::ClassParent.

C++ Examples:

```
/*
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button’s class and the names of its
parent classes are added to the list box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
  TClass ClassRef;
  ListBox1->Clear();
  ClassRef = Sender->ClassType();
  while (ClassRef != NULL)
  {
    ListBox1->Items->Add(ClassName(ClassRef));
    ClassRef = ClassParent(ClassRef);
  };
}
/*
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
*/
/*
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component’s coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < ComponentCount; i++)
    ListBox1->Items->InsertObject(0,
                                  Components[i]->Name,
                                  (TObject *)Components[i]);
}

void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  if (Button == mbRight)
  {
    TClass ClassRef;
    int Index = ListBox1->ItemAtPos(Point(X,Y), true);
    // only components that are controls have a position
    // make sure the component is a control
    for (ClassRef = ListBox1->Items->Objects[Index]->ClassType();
         ClassRef != NULL;
         ClassRef = ClassRef->ClassParent())
      if (String(ClassRef->ClassName()) == "TControl")
      {
        TControl *TheObject = (TControl *)ListBox1->Items->Objects[Index];
        StatusBar1->SimpleText =
          TheObject->Name + " is at (" +
          IntToStr(TheObject->Left) + ", " +
          IntToStr(TheObject->Top) + ")";
         break;
      }
    if (ClassRef == NULL) // if it wasn't a control
      MessageBeep(0);
```

```
  }
}
```

Delphi Examples:

```
{
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button's class and the names of its
parent classes are added to the list box.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ClassRef: TClass;
begin
  ListBox1.Clear;
  ClassRef := Sender.ClassType;
  while ClassRef <> nil do
  begin
    ListBox1.Items.Add(ClassRef.ClassName);
    ClassRef := ClassRef.ClassParent;
  end;
end;
{
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
}
```

See Also

System::TObject::ClassInfo (see page 22290)

System::TObject::ClassType (see page 22295)

System::TObject::InheritsFrom (see page 22305)

158.1.29.4.7 TObject::ClassType Method

Returns the class reference for the object's class.

Pascal

```
function ClassType: TClass;
```

C++

```
__fastcall TClass ClassType();
```

Description

Note: System::TObject::ClassType dynamically determines the type of an object and returns its class reference, or metaclass.

Avoid using System::TObject::ClassType in application code.

Note: In Delphi code, use the is or as operators instead of System::TObject::ClassType.

Note: In C++ code, use a dynamic cast or the System::TObject::InheritsFrom (see page 22305) method instead of System::TObject::ClassType.

C++ Examples:

```
/*
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button's class and the names of its
parent classes are added to the list box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClass ClassRef;
  ListBox1->Clear();
  ClassRef = Sender->ClassType();
  while (ClassRef != NULL)
  {
    ListBox1->Items->Add(ClassName(ClassRef));
    ClassRef = ClassParent(ClassRef);
  };
}
/*
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
*/
/*
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < ComponentCount; i++)
    ListBox1->Items->InsertObject(0,
                                  Components[i]->Name,
                                  (TObject *)Components[i]);
}

void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  if (Button == mbRight)
  {
    TClass ClassRef;
    int Index = ListBox1->ItemAtPos(Point(X,Y), true);
    // only components that are controls have a position
    // make sure the component is a control
    for (ClassRef = ListBox1->Items->Objects[Index]->ClassType();
         ClassRef != NULL;
```

```
          ClassRef = ClassRef->ClassParent())
        if (String(ClassRef->ClassName()) == "TControl")
        {
          TControl *TheObject = (TControl *)ListBox1->Items->Objects[Index];
          StatusBar1->SimpleText =
            TheObject->Name + " is at (" +
            IntToStr(TheObject->Left) + ", " +
            IntToStr(TheObject->Top) + ")";
          break;
        }
      if (ClassRef == NULL) // if it wasn't a control
        MessageBeep(0);
    }
  }
  /*
  To run this example, add the example code to a new project.
  The example code dynamically creates a TCoolbar and three
  TCoolBand objects populated with a windowed control in each
  TCoolBand.
  */
  void AddBands(TCoolBar *CoolBar, TList *Objects)
  {
    TControl *pCurrent;
    for (int i = 0; i < Objects->Count; i++)
    {
      pCurrent = (TControl *)Objects->Items[i];
      // only add windowed controls to the coolbar
      if (dynamic_cast<TWinControl *>(pCurrent) != NULL)
      {
        // Placing the control onto the CoolBar
        // causes the TCoolBar object to create a TCoolBand
        // and place the control within the band.
        pCurrent->Parent = CoolBar;   // This statement increments CoolBar->Bands->Count.
        // Get the reference of the TCoolBand just created and assign text.
        String S = pCurrent->ClassName(pCurrent->ClassType());
        CoolBar->Bands->Items[CoolBar->Bands->Count - 1]->Text = S;
      }
    }
  }

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    TCoolBar *CoolBar = new TCoolBar(this);
    TList *List;
    CoolBar->Parent = this;
    CoolBar->Align = alTop;
    List = new TList();
    try
    {
      TControl *Control;
      Control = new TButton(CoolBar);
      List->Add(Control);
      Control = new TCheckBox(CoolBar);
      List->Add(Control);
      Control = new TEdit(CoolBar);
      List->Add(Control);
      AddBands(CoolBar, List);
    }
    catch (...)
    {
      ShowMessage("Some objects could not be added to Coolband");
    }
    delete List;
  }
```

Delphi Examples:

```
{
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button's class and the names of its
parent classes are added to the list box.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ClassRef: TClass;
begin
  ListBox1.Clear;
  ClassRef := Sender.ClassType;
  while ClassRef <> nil do
  begin
    ListBox1.Items.Add(ClassRef.ClassName);
    ClassRef := ClassRef.ClassParent;
  end;
end;
{
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
}
```

See Also

System::TObject::ClassInfo (see page 22290)

System::TObject::ClassName (see page 22290)

System::TObject::ClassParent (see page 22293)

_classid

158.1.29.4.8 TObject::CleanupInstance Method

Performs finalization on long strings, variants, and interface variables within a class.

Pascal

```
procedure CleanupInstance;
```

C++

```
__fastcall CleanupInstance();
```

Description

Do not call System::TObject::CleanupInstance directly. System::TObject::CleanupInstance is called automatically when the object instance is destroyed.

System::TObject::CleanupInstance releases all long strings and variants. It sets long strings to empty and variants to Unassigned.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

delete

158.1.29.4.9 TObject::DefaultHandler Method

Provides the interface for a method that processes message records.

Pascal

```
procedure DefaultHandler(var Message); virtual;
```

C++

```
virtual DefaultHandler(var  Message);
```

Description

System::TObject::DefaultHandler is called by System::TObject::Dispatch (see page 22299) when it cannot find a method for a particular message. System::TObject::DefaultHandler provides message handling for all messages for which an object does not have specific handlers. Descendant classes that process messages override System::TObject::DefaultHandler according to the types of messages they handle.

Note: In a Delphi message-handling method, calling inherited results in a call to the ancestor's System::TObject::DefaultHandler method only if that ancestor does not specify a message method for the particular message being handled. Otherwise, calling inherited results in a call to the specific handler for that type of message.

See Also

System::TObject::Dispatch (see page 22299)

158.1.29.4.10 TObject::Dispatch Method

Calls message-handling methods for the object, based on the contents of the Message parameter.

Pascal

```
procedure Dispatch(var Message); virtual;
```

C++

```
virtual Dispatch(var  Message);
```

Description

Call System::TObject::Dispatch to automatically pass messages to the appropriate message handler.

System::TObject::Dispatch determines whether a message is in the list of message handlers declared for the object. If the object does not handle the message, System::TObject::Dispatch then examines the message-handler list of the ancestor class, and continues checking ancestors until it either finds a specific handler or runs out of ancestors, in which case it calls System::TObject::DefaultHandler (see page 22299).

The only assumption System::TObject::Dispatch makes about the data in Message is that the first two bytes contain a message ID—that is, an integer that determines which message handler System::TObject::Dispatch calls. Although any kind of data can be passed to System::TObject::Dispatch, most System::TObject::TObject (see page 22285) descendants expect a message record such as TMessage or a specific data structure type.

See Also

System::TObject::DefaultHandler (see page 22299)

TMessage

158.1.29.4.11 TObject::FieldAddress Method

Returns the address of a published object field.

Pascal

```
function FieldAddress(const Name: ShortString): Pointer;
```

C++

```
Pointer FieldAddress(const ShortString Name);
```

Description

System::TObject::FieldAddress is used internally by the component streaming system to access a specified published field of an object. System::TObject::FieldAddress returns a pointer to the field if it exists. If the object has no published field by that name, System::TObject::FieldAddress returns nil (Delphi) or NULL (C++).

Programs should access and manipulate fields by using properties instead of System::TObject::FieldAddress.

158.1.29.4.12 TObject::Free Method

Destroys an object and frees its associated memory, if necessary.

Pascal

```
procedure Free;
```

C++

```
__fastcall Free();
```

Description

Use System::TObject::Free to destroy an object. System::TObject::Free automatically calls the destructor if the object reference is not nil. Any object instantiated at runtime that does not have an owner should be destroyed by a call to System::TObject::Free so that it can be properly disposed of and its memory released. Unlike System::TObject::Destroy (see page 22287), System::TObject::Free is successful even if the object is nil; so if the object was never initialized, System::TObject::Free won't result in an error.

When you call System::TObject::Free for a component, it calls System::TObject::Free for all components that it owns—that is, all components in its component list. Since a form owns all the controls and other components that are created on it in design mode, those components are automatically freed when the form is freed. By default, all forms are owned by the Application object; when the application terminates, it frees the Application object, which frees all forms. For objects that are not components, or for components created with a nil owner, be sure to call System::TObject::Free after you are finished with them; otherwise the allocated memory will not be usable until after the application terminates.

Warning: Never explicitly free a component within one of its own event handlers or the event handler of a component it owns or contains. For example, don't free a button, or the form that owns the button, in its OnClick event handler.

To free a form, call its Release method, which destroys the form and releases the memory allocated for it after all its event handlers and those of the components it contains are through executing.

Note: In C++ code, do not use System::TObject::Free to destroy an object. Use the delete keyword.

C++ Examples:

```
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
```

```
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
    NewItem :=
      TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;
```

```
{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;
```

See Also

System::TObject::Destroy (see page 22287)

Release (see page 13885)

Owner (see page 2807)

FreeAndNil (see page 22907)

158.1.29.4.13 TObject::FreeInstance Method

Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

Pascal

```
procedure FreeInstance; virtual;
```

C++

```
virtual __fastcall FreeInstance();
```

Description

All destructors call System::TObject::FreeInstance automatically to deallocate memory that was allocated by overriding System::TObject::NewInstance (see page 22306).

Do not call System::TObject::FreeInstance directly. System::TObject::FreeInstance should be overridden if System::TObject::NewInstance (see page 22306) was overridden to change the way the object's instance data was allocated.

Like System::TObject::NewInstance (see page 22306), System::TObject::FreeInstance uses the value returned from System::TObject::InstanceSize (see page 22306) to deallocate the object's memory.

See Also

System::TObject::InstanceSize (see page 22306)

System::TObject::NewInstance (see page 22306)

158.1.29.4.14 TObject::GetInterface Method

Retrieves a specified interface.

Pascal

```
function GetInterface(const IID: TGUID; out Obj): Boolean;
```

C++

```
Boolean GetInterface(const TGUID IID, out  Obj);
```

Description

System::TObject::GetInterface retrieves the interface designated by a GUID or type name. The basic implementation of System::TObject::GetInterface uses the GUID specified in the IID parameter. If the specified interface is supported by the class, it is returned in the Obj parameter, and System::TObject::GetInterface has a return value of true. Otherwise Obj contains nil (Delphi) or NULL (C++), and System::TObject::GetInterface returns false.

Note: In Delphi code, IID can be an interface name. The compiler automatically translates this name into the corresponding GUID.

Note: In C++ code, use the templated version of System::TObject::GetInterface to obtain an interface from a DelphiInterface (see page 22167) object.

System::TObject::GetInterface is equivalent to the as operator (Delphi) and dynamic casts (C++), except that System::TObject::GetInterface does not raise an exception if the interface is not supported.

See Also

System::TInterfacedObject::QueryInterface (see page 22280)

158.1.29.4.15 TObject::GetInterfaceEntry Method

Returns the entry for a specific interface implemented in a class.

Pascal

```
class function GetInterfaceEntry(const IID: TGUID): PInterfaceEntry;
```

C++

```
PInterfaceEntry GetInterfaceEntry(const TGUID IID);
```

Description

System::TObject::GetInterfaceEntry returns the class entry for the interface specified by the IID parameter.

Note: In Delphi Code, IID can be an interface name. The compiler replaces this name with the actual GUID.

Note: COM objects can use System::TObject::GetInterfaceEntry to automate dispatch calls to a dual-IDispatch (see page 22394) interface.

158.1.29.4.16 TObject::GetInterfaceTable Method

Returns a pointer to a structure containing all of the interfaces implemented by a given class.

Pascal

```
class function GetInterfaceTable: PInterfaceTable;
```

C++

```
PInterfaceTable GetInterfaceTable();
```

Description

System::TObject::GetInterfaceTable returns the interface entries for the class. This list contains only interfaces implemented by this class, not its ancestors. To find the ancestor list, iteratively call System::TObject::ClassParent (see page 22293) and then call System::TObject::GetInterfaceTable on the value it returns. To find the entry for a specific interface, use the System::TObject::GetInterfaceEntry (see page 22304) method instead.

See Also

System::TObject::GetInterfaceEntry (see page 22304)

System::TObject::ClassParent (see page 22293)

158.1.29.4.17 TObject::InheritsFrom Method

Determines the relationship of two object types.

Pascal

```
class function InheritsFrom(AClass: TClass): Boolean;
```

C++

```
Boolean InheritsFrom(TClass AClass);
```

Description

Use System::TObject::InheritsFrom to determine if a particular class type or object is an instance of a class or one of its descendants. System::TObject::InheritsFrom returns true if the object type specified in the aClass parameter is an ancestor of the object type or the type of the object itself. Otherwise, it returns false.

Note: System::TObject::InheritsFrom is similar to the Delphi is operator, but applies to class references.

Note: In C++ code, a non-static version of System::TObject::InheritsFrom is provided. This call is useful in determining whether a descendant class method or property can be used, given a variable of a base class. For example, use System::TObject::InheritsFrom to determine whether the Sender parameter in an event handler is of a particular class type or one of its descendants.

See Also

System::TObject::ClassParent (see page 22293)

System::TObject::ClassNameIs (see page 22293)

158.1.29.4.18 TObject::InitInstance Method

Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

Pascal

```
class function InitInstance(Instance: Pointer): TObject;
```

C++

```
TObject InitInstance(Pointer Instance);
```

Description

There should be no reason to call System::TObject::InitInstance directly. System::TObject::InitInstance is called by System::TObject::NewInstance (see page 22306) when an object is created. When overriding System::TObject::NewInstance (see page 22306), be sure to call System::TObject::InitInstance as the last statement.

System::TObject::InitInstance is not virtual, so you cannot override it: Instead, initialize any data for an object in the constructor.

See Also

System::TObject::Create (see page 22287)

System::TObject::NewInstance (see page 22306)

158.1.29.4.19 TObject::InstanceSize Method

Returns the size in bytes of each instance of the object type.

Pascal

```
class function InstanceSize: Longint;
```

C++

```
Longint InstanceSize();
```

Description

System::TObject::InstanceSize indicates how many bytes of memory are required for a class's instance data. System::TObject::InstanceSize is called from methods that allocate and deallocate memory. System::TObject::InstanceSize is not a virtual method, so it cannot be overridden. System::TObject::InstanceSize should be called only when implementing a custom version of System::TObject::NewInstance (see page 22306).

See Also

System::TObject::FreeInstance (see page 22303)

System::TObject::NewInstance (see page 22306)

158.1.29.4.20 TObject::MethodAddress Method

Pascal

```
class function MethodAddress(const Name: ShortString): Pointer;
```

C++

```
Pointer MethodAddress(const ShortString Name);
```

Description

This is MethodAddress, a member of class TObject.

158.1.29.4.21 TObject::MethodName Method

Pascal

```
class function MethodName(Address: Pointer): ShortString;
```

C++

```
ShortString MethodName(Pointer Address);
```

Description

This is MethodName, a member of class TObject.

158.1.29.4.22 TObject::NewInstance Method

Allocates memory for an instance of an object type and returns a pointer to that new instance.

Pascal

```
class function NewInstance: TObject; virtual;
```

C++

```
virtual TObject NewInstance();
```

Description

All constructors call System::TObject::NewInstance automatically. System::TObject::NewInstance calls System::TObject::InstanceSize (see page 22306) to determine how much memory to allocate from the heap to contain a particular instance. Do not call System::TObject::NewInstance directly.

Override System::TObject::NewInstance only for special memory allocation requirements. For example, when allocating a large number of identical objects that all need to be in memory at the same time, you could allocate a single block of memory for the entire group, then override System::TObject::NewInstance to use part of that larger block for each instance.

If you override System::TObject::NewInstance to allocate memory, you may need to override System::TObject::FreeInstance (see page 22303) to deallocate the memory.

Note: By default System::TObject::NewInstance calls System::TObject::InitInstance (see page 22305).

See Also

System::TObject::FreeInstance (see page 22303)

System::TObject::InstanceSize (see page 22306)

System::TObject::InitInstance (see page 22305)

158.1.29.4.23 TObject::SafeCallException Method

Handles exceptions in methods declared using the safecall calling convention.

Pascal

```
function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): HResult; virtual;
```

C++

```
virtual HResult SafeCallException(TObject ExceptObject, Pointer ExceptAddr);
```

Description

System::TObject::SafeCallException handles exceptions in methods that use the safecall calling convention. Some classes that implement interfaces override this method to handle errors that might occur.

As implemented in System::TObject::TObject (see page 22285), System::TObject::SafeCallException simply returns E_UNEXPECTED (see page 22664). This is the appropriate response for classes that do no support interfaces

158.1.30 Variant Class

System::Variant is a C++ implementation of the Delphi intrinsic type System::Variant.

Class Hierarchy

Pascal

```
Variant = class(TVarData);
```

C++

```
class Variant : public TVarData;
```

File

sysvari.h

Description

The System::Variant type is capable of representing values that change type dynamically. Whereas a variable of any other type is statically bound to that type, a variable of the System::Variant type can assume values of differing types at run-time. The System::Variant type is most commonly used in situations where the actual type to be operated upon varies or is unknown at compile-time. While variants offer great flexibility, they also consume more memory than regular variables, and operations on variants are substantially slower than operations on statically typed values.

System::Variants (see page 23567) can contain various numeric, string, Boolean, and date-time values. If COM is available (Windows only) variants can also contain OLE Automation objects. Finally, variants can contain arrays of any of the above types. The special System::Variant value Unassigned is used to indicate that a variant has not yet been assigned a value, and the special variant value Null is used to indicate unknown or missing data.

A variant can be constructed from the following data types:

short (see page 22370)

int (see page 22368)

float (see page 22367)

double (see page 22367)

Currency (see page 22366)

TDateTime (see page 22371)

bool (see page 22365)

WordBool

Byte

AnsiString (see page 22365)&

char (see page 22365) *

wchar_t * const

Ole2::IDispatch (see page 22394)* const (Windows only)

Ole2::IUnknown (see page 22586)* const (Windows only)

Note: When a System::Variant contains an OLE Automation object, the variant can be used to get and set properties of the object, and to invoke methods on the object.

System::Variant variables are always initialized to be Unassigned when they are first created. This is true whether a variant variable is global, local, or part of a structure such as an array.

In C++ code, variant arrays are usually created using a System::Variant constructor in conjunction with the OPENARRAY macro:

```
Variant V(OPENARRAY(int,(0,HighVal,0,HighVal)),varInteger);
```

You can also use the VarArrayCreate function, but that function is designed for use in Delphi code, and makes for complex C++ code.

Note: Use only the operators declared within System::Variant. The compiler ignores any operators you overload yourself.

Note: For binary operators, if one operand is of type System::Variant, the other operand is automatically converted to type System::Variant.

See Also

Autocmd

TVarRec (see page 22577)

TVarData (see page 22571)

Openarray

158.1.30.1 Variant Members

The following tables list the members exposed by Variant.

Public Friends

Name	Description
class AutoCmd (see page 22375)	This is friend class AutoCmd.

Public Methods

	Name	Description
	~Variant (see page 22311)	Frees a System::Variant::Variant (see page 22314) object.
	ArrayDimCount (see page 22317)	Returns the number of dimensions of the given variant.
	ArrayHighBound (see page 22317)	Returns the high bound of the given dimension in the given variant array.
	ArrayLock (see page 22317)	Locks the given variant array and returns a pointer to the data stored in the array.
	ArrayLowBound (see page 22318)	Returns the low bound of the given dimension in the given variant array.
	ArrayRedim (see page 22318)	Resizes the given variant array by changing the high bound of the rightmost dimension to the given value.
	ArrayUnlock (see page 22318)	Unlocks a variant array that was previously locked by ArrayLock (see page 22317).
	Assign (see page 22318)	This is Assign, a member of class Variant.
	AsType (see page 22319)	Casts a System::Variant::Variant (see page 22314) object to the type specified by the argument VarType.
	ChangeType (see page 22319)	Changes the data type of a System::Variant::Variant (see page 22314) object once it has been instantiated.
	Clear (see page 22319)	System::Variant::Clears the value of the System::Variant::Variant (see page 22314).
	CreateObject (see page 22320)	Creates an OLE Automation object with the programmatic identifier specified by ProgID and returns it as a System::Variant::Variant (see page 22314).
	Empty (see page 22320)	This is Empty, a member of class Variant.
	Exec (see page 22320)	System::Variant::Executes an OLE procedure, function, or a property Get or Set method.
	GetActiveObject (see page 22321)	Retrieves a reference to an IDispatch (see page 22394) interface of a currently running, registered OLE Automation object, and returns it as a System::Variant::Variant (see page 22314).
	GetBaseVariant (see page 22321)	Returns the Variant (see page 22307) that is assigned as the value of this Variant (see page 22307) when Type (see page 22341) is varVariant (see page 22690).
	GetElement (see page 22322)	This is the overview for the GetElement method overload.
	GetError (see page 22322)	Converts the System::Variant::Variant (see page 22314) to a varError (see page 22684) type and returns the integer value.
	IsArray (see page 22323)	Queries the VType of the base class TVarData (see page 22571) for the varArray (see page 22679) bit.
	IsEmpty (see page 22323)	Queries the VType of the base class TVarData (see page 22571).
	IsNull (see page 22323)	Queries the VType of the base class TVarData (see page 22571) for the varNull (see page 22686) bit.

	NoParam (see page 22324)	This is NoParam, a member of class Variant.
	OleFunction (see page 22324)	This is the overview for the OleFunction method overload.
	OleProcedure (see page 22330)	This is the overview for the OleProcedure method overload.
	OlePropertyGet (see page 22336)	This is the overview for the OlePropertyGet method overload.
	OlePropertySet (see page 22338)	This is the overview for the OlePropertySet method overload.
	PutElement (see page 22340)	This is the overview for the PutElement method overload.
	Sanitized (see page 22340)	This is Sanitized, a member of class Variant.
	SetError (see page 22341)	Converts the System::Variant::Variant (see page 22314) to a varError (see page 22684) type and assigns the integer err as the value of the varError (see page 22684).
	Type (see page 22341)	This is Type, a member of class Variant.
	Variant (see page 22312)	This is the overview for the Variant constructor overload.

Public Operators

	Name	Description
	- (see page 22343)	This is the overview for the - operator overload.
	! (see page 22344)	Returns a new System::Variant::Variant (see page 22314) that is the logical negation of this System::Variant::Variant (see page 22314).
	!= (see page 22344)	This is the overview for the != operator overload.
	% (see page 22345)	This is the overview for the % operator overload.
	%= (see page 22346)	Returns this System::Variant::Variant (see page 22314) as the remainder from dividing its original value and the System::Variant::Variant (see page 22314) specified by rhs.
	& (see page 22346)	This is the overview for the & operator overload.
	&= (see page 22347)	Returns this System::Variant::Variant (see page 22314) after performing a bitwise and operation.
	* (see page 22347)	This is the overview for the * operator overload.
	*= (see page 22348)	Multiplies this System::Variant::Variant (see page 22314) by the System::Variant::Variant (see page 22314) specified by rhs.
	/ (see page 22348)	This is the overview for the / operator overload.
	/= (see page 22349)	Divides this System::Variant::Variant (see page 22314) by the System::Variant::Variant (see page 22314) rhs.
	^ (see page 22349)	This is the overview for the ^ operator overload.
	^= (see page 22350)	Returns this System::Variant::Variant (see page 22314) after performing a bitwise xor operation.
	__int64 (see page 22350)	This is __int64, a member of class Variant.
	| (see page 22351)	This is the overview for the | operator overload.
	|= (see page 22351)	Returns this System::Variant::Variant (see page 22314) after performing a bitwise or operation.
	+ (see page 22352)	This is the overview for the + operator overload.
	+= (see page 22352)	Adds the System::Variant::Variant (see page 22314) specified by rhs to this System::Variant::Variant (see page 22314) object.
	< (see page 22353)	This is the overview for the < operator overload.
	<< (see page 22353)	This is the overview for the << operator overload.
	<<= (see page 22354)	Performs a bitwise shl (left shift) operation on this System::Variant::Variant (see page 22314).
	<= (see page 22355)	This is the overview for the <= operator overload.
	-= (see page 22361)	Subtracts the System::Variant::Variant (see page 22314) specified by rhs from this System::Variant::Variant (see page 22314) object.
	= (see page 22355)	This is the overview for the = operator overload.
	== (see page 22361)	This is the overview for the == operator overload.
	> (see page 22362)	This is the overview for the > operator overload.
	>= (see page 22363)	This is the overview for the >= operator overload.
	>> (see page 22363)	This is the overview for the >> operator overload.
	>>= (see page 22365)	Performs a bitwise shr (right shift) operation on this System::Variant::Variant (see page 22314).
	AnsiString (see page 22365)	Converts the System::Variant::Variant (see page 22314) to an AnsiString object.
	bool (see page 22365)	Converts the System::Variant::Variant (see page 22314) to a boolean value.

	char (see page 22365)	This is char, a member of class Variant.
	Currency (see page 22366)	Converts the System::Variant::Variant (see page 22314) to a Currency object.
	CURRENCY (see page 22366)	This is CURRENCY, a member of class Variant.
	Currency* (see page 22366)	This is Currency*, a member of class Variant.
	DECIMAL (see page 22366)	This is DECIMAL, a member of class Variant.
	double (see page 22367)	Converts the System::Variant::Variant (see page 22314) to a double value.
	double* (see page 22367)	This is double*, a member of class Variant.
	float (see page 22367)	Converts the System::Variant::Variant (see page 22314) to a float value.
	float* (see page 22368)	This is float*, a member of class Variant.
	IDispatch* (see page 22368)	This is IDispatch*, a member of class Variant.
	IInterface* (see page 22368)	This is IInterface*, a member of class Variant.
	int (see page 22368)	Converts the System::Variant::Variant (see page 22314) to an integer value.
	int* (see page 22369)	This is int*, a member of class Variant.
	IUnknown* (see page 22369)	This is IUnknown*, a member of class Variant.
	long (see page 22369)	This is long, a member of class Variant.
	long* (see page 22369)	This is long*, a member of class Variant.
	LPSAFEARRAY (see page 22369)	This is LPSAFEARRAY, a member of class Variant.
	short (see page 22370)	Converts the System::Variant::Variant (see page 22314) to a short value.
	short* (see page 22370)	This is short*, a member of class Variant.
	signed char (see page 22370)	This is signed char, a member of class Variant.
	signed char* (see page 22370)	This is signed char*, a member of class Variant.
	TDateTime (see page 22371)	Converts the System::Variant::Variant (see page 22314) to a TDateTime object.
	TDateTime* (see page 22371)	This is TDateTime*, a member of class Variant.
	TVariant (see page 22371)	This is TVariant, a member of class Variant.
	TVariant* (see page 22371)	This is TVariant*, a member of class Variant.
	unsigned char (see page 22372)	This is unsigned char, a member of class Variant.
	unsigned char* (see page 22372)	This is unsigned char*, a member of class Variant.
	unsigned int (see page 22372)	This is unsigned int, a member of class Variant.
	unsigned int* (see page 22372)	This is unsigned int*, a member of class Variant.
	unsigned long (see page 22373)	This is unsigned long, a member of class Variant.
	unsigned long* (see page 22373)	This is unsigned long*, a member of class Variant.
	unsigned short (see page 22373)	This is unsigned short, a member of class Variant.
	unsigned short* (see page 22373)	This is unsigned short*, a member of class Variant.
	VARIANT (see page 22373)	This is VARIANT, a member of class Variant.
	VARIANT* (see page 22374)	This is VARIANT*, a member of class Variant.
	wchar_t* (see page 22374)	Converts the System::Variant::Variant (see page 22314) to the specified type and sets the varByRef (see page 22680) bit, meaning that the variant contains a reference as opposed to a value.
	wchar_t** (see page 22374)	This is wchar_t**, a member of class Variant.
	WideString (see page 22374)	This is WideString, a member of class Variant.

158.1.30.2 Variant::~Variant Destructor

Frees a System::Variant::Variant (see page 22314) object.

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~Variant();
```

Description

System::Variant::Destroy is the destructor for System::Variant::Variant (see page 22314).

158.1.30.3 Variant Constructor

This is the overview for the Variant constructor overload.

Overload List

	Name	Description
	Variant::Variant () (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (__int64) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (__int64*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (AnsiString&) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (bool) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (char) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (char*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (Currency&) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (CURRENCY&) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (Currency*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (CURRENCY*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (DECIMAL&) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (DECIMAL*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (double) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (double*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (float) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (float*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (IDispatch*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (IInterface*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (int) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (int*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (int*, int, Word) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (IUnknown*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (long) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (long double) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (long*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (SAFEARRAY&) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (SAFEARRAY*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (SAFEARRAY**) (see page 22314)	Creates an instance of System::Variant::Create.

	Variant::Variant (short) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (short*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (signed char) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (signed char*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (System::DelphiInterface<IDispatch>&) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (TDateTime&) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (TDateTime*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (TVarData&) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (TVariant&) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (unsigned char) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (unsigned char*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (unsigned int) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (unsigned int*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (unsigned long) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (unsigned long*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (unsigned short) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (unsigned short*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (Variant&) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (VARIANT&) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (Variant*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (VARIANT*) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (Variant*, int) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (VariantError&) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (wchar_t**) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (wchar_t*, bool) (see page 22314)	Creates an instance of System::Variant::Create.
	Variant::Variant (WideString&) (see page 22314)	Creates an instance of System::Variant::Create.

158.1.30.3.1 Variant::Variant Constructor ()

Creates an instance of System::Variant::Create.

Pascal

```
constructor Create;
constructor Create(const src: Variant&);
constructor Create(const src: TVarData&);
constructor Create(const src: bool);
constructor Create(const src: char);
constructor Create(const src: signed char);
constructor Create(const src: unsigned char);
constructor Create(const src: short);
constructor Create(const src: unsigned short);
constructor Create(const src: Integer);
constructor Create(const src: unsigned int);
constructor Create(const src: long);
constructor Create(const src: unsigned long);
constructor Create(const src: float);
constructor Create(const src: double);
constructor Create(const src: long double);
constructor Create(const src: __int64);
constructor Create(const src: CURRENCY&);
constructor Create(const src: DECIMAL&);
constructor Create(const src: SAFEARRAY&);
constructor Create(const src: VARIANT&);
constructor Create(var src: SAFEARRAY);
constructor Create(var src: VARIANT);
constructor Create(const src: TVariant&);
constructor Create(const src: AnsiString&);
constructor Create(const src: Currency&);
constructor Create(const src: TDateTime&);
constructor Create(const src: VariantError&);
constructor Create(const src: WideString&);
constructor Create(const src: char*);
constructor Create(var src: wchar_t; makeCopy: bool = true);
constructor Create(var src: IUnknown);
constructor Create(var src: IInterface);
constructor Create(var src: IDispatch);
constructor Create(const src: System::DelphiInterface<IDispatch>&);
constructor Create(var src: signed char);
constructor Create(var src: unsigned char);
constructor Create(var src: short);
constructor Create(var src: unsigned short);
constructor Create(var src: Integer);
constructor Create(var src: unsigned int);
constructor Create(var src: long);
constructor Create(var src: unsigned long);
constructor Create(var src: float);
constructor Create(var src: double);
constructor Create(var src: __int64);
constructor Create(var src: CURRENCY);
constructor Create(var src: DECIMAL);
constructor Create(var src: SAFEARRAY*);
constructor Create(var src: wchar_t*);
constructor Create(var src: Currency);
constructor Create(var src: TDateTime);
constructor Create(var src: Variant);
constructor Create(const bounds: int*; const boundsSize: Integer; varType: Word);
constructor Create(const values: Variant*; const valuesSize: Integer);
```

C++

```
__fastcall Variant();
```

```
__fastcall Variant(const Variant& src);
__fastcall Variant(const TVarData& src);
__fastcall Variant(const bool src);
__fastcall Variant(const char src);
__fastcall Variant(const signed char src);
__fastcall Variant(const unsigned char src);
__fastcall Variant(const short src);
__fastcall Variant(const unsigned short src);
__fastcall Variant(const int src);
__fastcall Variant(const unsigned int src);
__fastcall Variant(const long src);
__fastcall Variant(const unsigned long src);
__fastcall Variant(const float src);
__fastcall Variant(const double src);
__fastcall Variant(const long double src);
__fastcall Variant(const __int64 src);
__fastcall Variant(const CURRENCY& src);
__fastcall Variant(const DECIMAL& src);
__fastcall Variant(const SAFEARRAY& src);
__fastcall Variant(const VARIANT& src);
__fastcall Variant(SAFEARRAY* src);
__fastcall Variant(VARIANT* src);
__fastcall Variant(const TVariant& src);
__fastcall Variant(const AnsiString& src);
__fastcall Variant(const Currency& src);
__fastcall Variant(const TDateTime& src);
__fastcall Variant(const VariantError& src);
__fastcall Variant(const WideString& src);
__fastcall Variant(const char* src);
__fastcall Variant(wchar_t* src, bool makeCopy = true);
__fastcall Variant(IUnknown* src);
__fastcall Variant(IInterface* src);
__fastcall Variant(IDispatch* src);
__fastcall Variant(const System::DelphiInterface<IDispatch>& src);
__fastcall Variant(signed char* src);
__fastcall Variant(unsigned char* src);
__fastcall Variant(short* src);
__fastcall Variant(unsigned short* src);
__fastcall Variant(int* src);
__fastcall Variant(unsigned int* src);
__fastcall Variant(long* src);
__fastcall Variant(unsigned long* src);
__fastcall Variant(float* src);
__fastcall Variant(double* src);
__fastcall Variant(__int64* src);
__fastcall Variant(CURRENCY* src);
__fastcall Variant(DECIMAL* src);
__fastcall Variant(SAFEARRAY** src);
__fastcall Variant(wchar_t** src);
__fastcall Variant(Currency* src);
__fastcall Variant(TDateTime* src);
__fastcall Variant(Variant* src);
__fastcall Variant(const int* bounds, const int boundsSize, Word varType);
__fastcall Variant(const Variant* values, const int valuesSize);
```

Description

The first System::Variant::Create constructor above creates a variant instance with the value Unassigned.

The second System::Variant::Create constructor creates a System::Variant::Create array with the bounds given by bounds and the element type given by varType. The bounds parameter must contain an even number of values, and each pair of values must specify the upper and lower bounds of one dimension of the array. The boundsSize parameter is the index of the last element in bounds. The element type of the array, given by the varType parameter, must be one of the TVarData (see page 22571) type codes, and cannot include the varArray (see page 22679) or varByRef (see page 22680) bits. The element type cannot be varString (see page 22688). To create an array of strings, use the varOleStr (see page 22686) type code. If the

element type is varSystem::Variant::Create, the elements of the array are themselves System::Variant::Creates and can, in turn, contain System::Variant::Create arrays.

The third System::Variant::Create constructor creates a one-dimensional System::Variant::Create array with the elements given by the values parameter. The low bound of the returned array is zero, the high bound is given by valuesSize (the index of the last value). The element type of the instantiated array is System::Variant::Create.

The remaining System::Variant::Create constructors create a System::Variant::Create with the same value as the src argument, converting src as appropriate. Constructors that declare src as a const parameter copy src into the variant's internal representation. Constructors that do not declare src as a const create a reference to src and set the variant's varByRef (see page 22680) bit.

See Also

Unassigned (see page 23643)

TVarData (see page 22571)

158.1.30.4 Variant Methods

The methods of the Variant class are listed here.

Public Methods

	Name	Description
	ArrayDimCount (see page 22317)	Returns the number of dimensions of the given variant.
	ArrayHighBound (see page 22317)	Returns the high bound of the given dimension in the given variant array.
	ArrayLock (see page 22317)	Locks the given variant array and returns a pointer to the data stored in the array.
	ArrayLowBound (see page 22318)	Returns the low bound of the given dimension in the given variant array.
	ArrayRedim (see page 22318)	Resizes the given variant array by changing the high bound of the rightmost dimension to the given value.
	ArrayUnlock (see page 22318)	Unlocks a variant array that was previously locked by ArrayLock (see page 22317).
	Assign (see page 22318)	This is Assign, a member of class Variant.
	AsType (see page 22319)	Casts a System::Variant::Variant (see page 22314) object to the type specified by the argument VarType.
	ChangeType (see page 22319)	Changes the data type of a System::Variant::Variant (see page 22314) object once it has been instantiated.
	Clear (see page 22319)	System::Variant::Clears the value of the System::Variant::Variant (see page 22314).
	CreateObject (see page 22320)	Creates an OLE Automation object with the programmatic identifier specified by ProgID and returns it as a System::Variant::Variant (see page 22314).
	Empty (see page 22320)	This is Empty, a member of class Variant.
	Exec (see page 22320)	System::Variant::Executes an OLE procedure, function, or a property Get or Set method.
	GetActiveObject (see page 22321)	Retrieves a reference to an IDispatch (see page 22394) interface of a currently running, registered OLE Automation object, and returns it as a System::Variant::Variant (see page 22314).
	GetBaseVariant (see page 22321)	Returns the Variant (see page 22307) that is assigned as the value of this Variant (see page 22307) when Type (see page 22341) is varVariant (see page 22690).
	GetElement (see page 22322)	This is the overview for the GetElement method overload.
	GetError (see page 22322)	Converts the System::Variant::Variant (see page 22314) to a varError (see page 22684) type and returns the integer value.
	IsArray (see page 22323)	Queries the VType of the base class TVarData (see page 22571) for the varArray (see page 22679) bit.
	IsEmpty (see page 22323)	Queries the VType of the base class TVarData (see page 22571).
	IsNull (see page 22323)	Queries the VType of the base class TVarData (see page 22571) for the varNull (see page 22686) bit.
	NoParam (see page 22324)	This is NoParam, a member of class Variant.

	OleFunction (see page 22324)	This is the overview for the OleFunction method overload.
	OleProcedure (see page 22330)	This is the overview for the OleProcedure method overload.
	OlePropertyGet (see page 22336)	This is the overview for the OlePropertyGet method overload.
	OlePropertySet (see page 22338)	This is the overview for the OlePropertySet method overload.
	PutElement (see page 22340)	This is the overview for the PutElement method overload.
	Sanitized (see page 22340)	This is Sanitized, a member of class Variant.
	SetError (see page 22341)	Converts the System::Variant::Variant (see page 22314) to a varError (see page 22684) type and assigns the integer err as the value of the varError (see page 22684).
	Type (see page 22341)	This is Type, a member of class Variant.

158.1.30.4.1 Variant::ArrayDimCount Method

Returns the number of dimensions of the given variant.

Pascal

```
function ArrayDimCount: Integer;
```

C++

```
__fastcall int ArrayDimCount() const;
```

Description

The resulting value is zero if the given variant is not a System::Variant::Variant (see page 22314) array.

158.1.30.4.2 Variant::ArrayHighBound Method

Returns the high bound of the given dimension in the given variant array.

Pascal

```
function ArrayHighBound(const dim: Integer = 1): Integer;
```

C++

```
__fastcall int ArrayHighBound(const int dim = 1) const;
```

Description

The dim parameter should be 1 for the leftmost dimension, 2 for the second leftmost dimension, and so on. An ESystem::Variant::VariantError (see page 22375) exception is raised if the variant given by A is not an array, or if the dimension specified by dim does not exist.

158.1.30.4.3 Variant::ArrayLock Method

Locks the given variant array and returns a pointer to the data stored in the array.

Pascal

```
function ArrayLock: Pointer;
```

C++

```
__fastcall Pointer ArrayLock();
```

Description

While locked, a variant array can not be resized and any calls to ArrayRedim (see page 22318) will fail. Once locked, a variant array must later be unlocked using ArrayUnlock (see page 22318). An ESystem::Variant::VariantError (see page 22375) exception is raised if the variant given by A is not an array.

The pointer returned by System::Variant::ArrayLock points to an array of elements in which the leftmost dimension increases

first. In other words, the dimensions of the returned array pointer are reversed from the dimensions of the variant array.

Once it is verified that a variant array has the correct bounds and dimensions, System::Variant::ArrayLock can be used to gain direct access to the array data which results in much improved performance.

158.1.30.4.4 Variant::ArrayLowBound Method

Returns the low bound of the given dimension in the given variant array.

Pascal

```
function ArrayLowBound(const dim: Integer = 1): Integer;
```

C++

```
__fastcall int ArrayLowBound(const int dim = 1) const;
```

Description

The dim parameter should be 1 for the leftmost dimension, 2 for the second leftmost dimension, and so on. An ESystem::Variant::VariantError (see page 22375) exception is raised if the variant given by A is not an array, or if the dimension specified by dim does not exist.

158.1.30.4.5 Variant::ArrayRedim Method

Resizes the given variant array by changing the high bound of the rightmost dimension to the given value.

Pascal

```
procedure ArrayRedim(highBound: Integer);
```

C++

```
__fastcall void ArrayRedim(int highBound);
```

Description

Existing elements of the array are preserved, and new elements are set to zero or empty. An ESystem::Variant::VariantError (see page 22375) exception is raised if the variant given by A is not an array, or if there is not enough memory to resize the array.

158.1.30.4.6 Variant::ArrayUnlock Method

Unlocks a variant array that was previously locked by ArrayLock (see page 22317).

Pascal

```
procedure ArrayUnlock;
```

C++

```
__fastcall void ArrayUnlock();
```

Description

System::Variant::ArrayUnlock invalidates the pointer returned by a call to the ArrayLock (see page 22317) method. An ESystem::Variant::VariantError (see page 22375) exception is raised if the System::Variant::Variant (see page 22314) is not an array.

158.1.30.4.7 Variant::Assign Method

Pascal

```
function Assign(var src: wchar_t; makeCopy: bool): Variant&;
```

C++

```
__fastcall Variant& Assign(wchar_t* src, bool makeCopy);
```

Description

This is Assign, a member of class Variant.

158.1.30.4.8 Variant::AsType Method

Casts a System::Variant::Variant (see page 22314) object to the type specified by the argument VarType.

Pascal

```
function AsType(VarType: Integer): Variant;
```

C++

```
__fastcall Variant AsType(int VarType) const;
```

Description

System::Variant::AsType returns a new System::Variant::Variant (see page 22314) with the same value as this System::Variant::Variant (see page 22314) but with the type specified by VarType. After the call to System::Variant::AsType there are two System::Variant::Variant (see page 22314) objects of the same type.

158.1.30.4.9 Variant::ChangeType Method

Changes the data type of a System::Variant::Variant (see page 22314) object once it has been instantiated.

Pascal

```
function ChangeType(VarType: Integer): Variant&;
```

C++

```
__fastcall Variant& ChangeType(int VarType);
```

Description

System::Variant::ChangeType converts the System::Variant::Variant (see page 22314) to the type specified by VarType.

See Also

System::Variant::Type2

158.1.30.4.10 Variant::Clear Method

System::Variant::Clears the value of the System::Variant::Variant (see page 22314).

Pascal

```
procedure Clear;
```

C++

```
__fastcall void Clear();
```

Description

System::Variant::Clear assigns the value Unassigned to this System::Variant::Variant (see page 22314).

See Also

Unassigned (see page 23643)

158.1.30.4.11 Variant::CreateObject Method

Creates an OLE Automation object with the programmatic identifier specified by ProgID and returns it as a System::Variant::Variant (see page 22314).

Pascal

```
function CreateObject(const ProgID: String&): Variant; static;
```

C++

```
static __fastcall Variant CreateObject(const String& ProgID);
```

Description

System::Variant::CreateObject creates a single uninitialized object of the class associated with the ProgID specified by the ProgID parameter. System::Variant::CreateObject is used to create an object of a specified type when the ProgID is known, and when the object is on a local or in-proc server. Only objects that are not part of an aggregate are created using System::Variant::CreateObject.

System::Variant::CreateObject is called once to create each new single instance of a class. To create multiple instance of the same class, it is recommended to use a class factory.

System::Variant::CreateObject returns a reference to the identifier of the interface to be used to communicate with the object. For System::Variant::CreateObject, this interface is of type IDispatch (see page 22394).

If ProgID is invalid, System::Variant::CreateObject throws an EOleSysError exception.

158.1.30.4.12 Variant::Empty Method

Pascal

```
function Empty: Variant&; static;
```

C++

```
static Variant& Empty();
```

Description

This is Empty, a member of class Variant.

158.1.30.4.13 Variant::Exec Method

System::Variant::Executes an OLE procedure, function, or a property Get or Set method.

Pascal

```
function Exec(var cmd: AutoCmd; lcid: Integer = LOCALE_SYSTEM_DEFAULT): Variant;
```

C++

```
__fastcall Variant Exec(AutoCmd& cmd, Integer lcid = LOCALE_SYSTEM_DEFAULT) const;
```

Description

Use System::Variant::Exec to call Automation methods.

The cmd parameter is the wrapper class for the method on an interface that is assigned as the System::Variant::Variant (see page 22314)'s value. Use this parameter to identify the method you want to call and to assign any parameter values.

The lcid parameter is ignored by System::Variant::Variant (see page 22314). The locale identifier used to interpret parameters is always LOCALE_SYSTEM_DEFAULT.

See Also

System::Variant::OleFunction (see page 22326)

System::Variant::OleProcedure (see page 22332)

System::Variant::OlePropertyGet (see page 22337)

System::Variant::OlePropertySet (see page 22338)

158.1.30.4.14 Variant::GetActiveObject Method

Retrieves a reference to an IDispatch (see page 22394) interface of a currently running, registered OLE Automation object, and returns it as a System::Variant::Variant (see page 22314).

Pascal

```
function GetActiveObject(const ProgID: String&): Variant; static;
```

C++

```
static __fastcall Variant GetActiveObject(const String& ProgID);
```

Description

System::Variant::GetActiveObject returns a reference to the IDispatch (see page 22394) interface of the active object for the given class indicated by ProgID. ProgID is the programmatic ID for the class of the active object from the OLE registration database.

If either the class object or the IDispatch (see page 22394) interface is not successfully returned, an EOleSysError exception is thrown.

Tip: If the Automation object is not running or not registered, use the CreateObject (see page 22320) member of System::Variant::Variant (see page 22314) to run it.

See Also

System::Variant::CreateObject (see page 22320)

158.1.30.4.15 Variant::GetBaseVariant Method

Returns the Variant (see page 22307) that is assigned as the value of this Variant (see page 22307) when Type (see page 22341) is varVariant (see page 22690).

Pascal

```
function GetBaseVariant: Variant*;
```

C++

```
__fastcall Variant* GetBaseVariant();
```

Description

Call GetBaseVariant to remove a level of indirection by accessing a Variant (see page 22307) that has been assigned as the value of this Variant (see page 22307). Check the Type (see page 22341) method to ensure that the variant is of type var Variant (see page 22307) before calling GetBaseVariant.

See Also

System::Variant (see page 22307)

158.1.30.4.16 GetElement Method

This is the overview for the GetElement method overload.

Overload List

	Name	Description
	Variant::GetElement (int) (see page 22322)	Returns the value of a specified element in a System::Variant::Variant (see page 22314) array.
	Variant::GetElement (int, int) (see page 22322)	Returns the value of a specified element in a System::Variant::Variant (see page 22314) array.
	Variant::GetElement (int, int, int) (see page 22322)	Returns the value of a specified element in a System::Variant::Variant (see page 22314) array.
	Variant::GetElement (int, int, int, int) (see page 22322)	Returns the value of a specified element in a System::Variant::Variant (see page 22314) array.
	Variant::GetElement (int, int, int, int, int) (see page 22322)	Returns the value of a specified element in a System::Variant::Variant (see page 22314) array.

158.1.30.4.16.1 Variant::GetElement Method (int)

Returns the value of a specified element in a System::Variant::Variant (see page 22314) array.

Pascal

```
function GetElement(const i1: Integer): Variant;
function GetElement(const i1: Integer; const i2: Integer): Variant;
function GetElement(const i1: Integer; const i2: Integer; const i3: Integer): Variant;
function GetElement(const i1: Integer; const i2: Integer; const i3: Integer; const i4:
Integer): Variant;
function GetElement(const i1: Integer; const i2: Integer; const i3: Integer; const i4:
Integer; const i5: Integer): Variant;
```

C++

```
__fastcall Variant GetElement(const int i1) const;
__fastcall Variant GetElement(const int i1, const int i2) const;
__fastcall Variant GetElement(const int i1, const int i2, const int i3) const;
__fastcall Variant GetElement(const int i1, const int i2, const int i3, const int i4) const;
__fastcall Variant GetElement(const int i1, const int i2, const int i3, const int i4, const
int i5) const;
```

Description

Use System::Variant::GetElement to access one-dimensional or multi-dimensional System::Variant::Variant (see page 22314) arrays of up to 5 dimensions. The parameters specify the indexes of the desired element.

See Also

System::Variant::PutElement (see page 22340)

158.1.30.4.17 Variant::GetError Method

Converts the System::Variant::Variant (see page 22314) to a varError (see page 22684) type and returns the integer value.

Pascal

```
function GetError: HResult;
```

C++

```
__fastcall HResult GetError() const;
```

Description

See the base class TVarData (see page 22571) for more information about varError (see page 22684).

158.1.30.4.18 Variant::IsArray Method

Queries the VType of the base class TVarData (see page 22571) for the varArray (see page 22679) bit.

Pascal

```
function IsArray: bool;
```

C++

```
__fastcall bool IsArray() const;
```

Description

System::Variant::IsArray returns true if the given variant is an array, false otherwise.

See Also

TVarData (see page 22571)

158.1.30.4.19 Variant::IsEmpty Method

Queries the VType of the base class TVarData (see page 22571).

Pascal

```
function IsEmpty: bool;
```

C++

```
__fastcall bool IsEmpty() const;
```

Description

System::Variant::IsEmpty returns true if the System::Variant::Variant (see page 22314) type code is varEmpty (see page 22683), false otherwise.

See Also

TVarData (see page 22571)

158.1.30.4.20 Variant::IsNull Method

Queries the VType of the base class TVarData (see page 22571) for the varNull (see page 22686) bit.

Pascal

```
function IsNull: bool;
```

C++

```
__fastcall bool IsNull() const;
```

Description

System::Variant::IsNull returns true if the System::Variant::Variant (see page 22314) type code is varNull (see page 22686), false otherwise.

See Also

TVarData (see page 22571)

158.1.30.4.21 Variant::NoParam Method

Pascal

```
function NoParam: Variant&; static;
```

C++

```
static Variant& NoParam();
```

Description

This is NoParam, a member of class Variant.

158.1.30.4.22 OleFunction Method

This is the overview for the OleFunction method overload.

Overload List

	Name	Description
	Variant::OleFunction (String&, P1) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().

	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().

	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27, P28) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27, P28, P29) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27, P28, P29, P30) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27, P28, P29, P30, P31) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27, P28, P29, P30, P31, P32) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().
	Variant::OleFunction (String&, TAutoArgsBase*) (see page 22326)	Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().

158.1.30.4.22.1 Variant::OleFunction Method (String&, P1)

Provides an alternate syntax for OLE Automation calls to Exec (see page 22320)().

Pascal

```
function OleFunction(const name: String&; var args: TAutoArgsBase = 0): Variant;
function OleFunction(const name: String&; p1: P1): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6):
Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
```

```
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16):
Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16; p17:
P17): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16; p17:
P17; p18: P18): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16; p17:
P17; p18: P18; p19: P19): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16; p17:
P17; p18: P18; p19: P19; p20: P20): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16; p17:
P17; p18: P18; p19: P19; p20: P20; p21: P21): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16; p17:
P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16; p17:
P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16; p17:
P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23; p24: P24): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16; p17:
P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23; p24: P24; p25: P25): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16; p17:
P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23; p24: P24; p25: P25; p26:
P26): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16; p17:
P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23; p24: P24; p25: P25; p26: P26;
p27: P27): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16; p17:
P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23; p24: P24; p25: P25; p26: P26;
p27: P27; p28: P28): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16; p17:
P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23; p24: P24; p25: P25; p26: P26;
p27: P27; p28: P28; p29: P29): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16; p17:
P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23; p24: P24; p25: P25; p26: P26;
p27: P27; p28: P28; p29: P29; p30: P30): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16; p17:
P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23; p24: P24; p25: P25; p26: P26;
p27: P27; p28: P28; p29: P29; p30: P30; p31: P31): Variant;
function OleFunction(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6; p7:
P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16; p17:
P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23; p24: P24; p25: P25; p26: P26;
p27: P27; p28: P28; p29: P29; p30: P30; p31: P31; p32: P32): Variant;
```

C++

```
Variant OleFunction(const String& name, TAutoArgsBase* args = 0);
```

```
template <class P1> Variant OleFunction(const String& name, P1 p1);
template <class P1, class P2> Variant OleFunction(const String& name, P1 p1, P2 p2);
template <class P1, class P2, class P3> Variant OleFunction(const String& name, P1 p1, P2 p2,
P3 p3);
template <class P1, class P2, class P3, class P4> Variant OleFunction(const String& name, P1
p1, P2 p2, P3 p3, P4 p4);
template <class P1, class P2, class P3, class P4, class P5> Variant OleFunction(const String&
name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5);
template <class P1, class P2, class P3, class P4, class P5, class P6> Variant
OleFunction(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7> Variant
OleFunction(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8>
Variant OleFunction(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8
p8);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9> Variant OleFunction(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7
p7, P8 p8, P9 p9);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10> Variant OleFunction(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5
p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11> Variant OleFunction(const String& name, P1 p1, P2 p2, P3 p3,
P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12> Variant OleFunction(const String& name, P1 p1, P2
p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13> Variant OleFunction(const String& name,
P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13
p13);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14> Variant OleFunction(const
String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11,
P12 p12, P13 p13, P14 p14);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15> Variant
OleFunction(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9,
P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16> Variant
OleFunction(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9,
P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16 p16);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17> Variant OleFunction(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7,
P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16 p16, P17 p17);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18> Variant OleFunction(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6
p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16 p16, P17
p17, P18 p18);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19> Variant OleFunction(const String& name, P1 p1, P2 p2, P3 p3, P4 p4,
P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16
p16, P17 p17, P18 p18, P19 p19);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20> Variant OleFunction(const String& name, P1 p1, P2 p2, P3
p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15
p15, P16 p16, P17 p17, P18 p18, P19 p19, P20 p20);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21> Variant OleFunction(const String& name, P1
p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13
```

```
p13, P14 p14, P15 p15, P16 p16, P17 p17, P18 p18, P19 p19, P20 p20, P21 p21);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22> Variant OleFunction(const String&
name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12
p12, P13 p13, P14 p14, P15 p15, P16 p16, P17 p17, P18 p18, P19 p19, P20 p20, P21 p21, P22 p22);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23> Variant
OleFunction(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9,
P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16 p16, P17 p17, P18 p18, P19 p19, P20
p20, P21 p21, P22 p22, P23 p23);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23, class P24> Variant
OleFunction(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9,
P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16 p16, P17 p17, P18 p18, P19 p19, P20
p20, P21 p21, P22 p22, P23 p23, P24 p24);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23, class P24, class P25>
Variant OleFunction(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8
p8, P9 p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16 p16, P17 p17, P18 p18,
P19 p19, P20 p20, P21 p21, P22 p22, P23 p23, P24 p24, P25 p25);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23, class P24, class P25,
class P26> Variant OleFunction(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6,
P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16 p16, P17 p17,
P18 p18, P19 p19, P20 p20, P21 p21, P22 p22, P23 p23, P24 p24, P25 p25, P26 p26);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23, class P24, class P25,
class P26, class P27> Variant OleFunction(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5
p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16 p16,
P17 p17, P18 p18, P19 p19, P20 p20, P21 p21, P22 p22, P23 p23, P24 p24, P25 p25, P26 p26, P27
p27);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23, class P24, class P25,
class P26, class P27, class P28> Variant OleFunction(const String& name, P1 p1, P2 p2, P3 p3,
P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15
p15, P16 p16, P17 p17, P18 p18, P19 p19, P20 p20, P21 p21, P22 p22, P23 p23, P24 p24, P25 p25,
P26 p26, P27 p27, P28 p28);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23, class P24, class P25,
class P26, class P27, class P28, class P29> Variant OleFunction(const String& name, P1 p1, P2
p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13 p13, P14
p14, P15 p15, P16 p16, P17 p17, P18 p18, P19 p19, P20 p20, P21 p21, P22 p22, P23 p23, P24 p24,
P25 p25, P26 p26, P27 p27, P28 p28, P29 p29);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23, class P24, class P25,
class P26, class P27, class P28, class P29, class P30> Variant OleFunction(const String& name,
P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13
p13, P14 p14, P15 p15, P16 p16, P17 p17, P18 p18, P19 p19, P20 p20, P21 p21, P22 p22, P23 p23,
P24 p24, P25 p25, P26 p26, P27 p27, P28 p28, P29 p29, P30 p30);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23, class P24, class P25,
class P26, class P27, class P28, class P29, class P30, class P31> Variant OleFunction(const
String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11,
P12 p12, P13 p13, P14 p14, P15 p15, P16 p16, P17 p17, P18 p18, P19 p19, P20 p20, P21 p21, P22
p22, P23 p23, P24 p24, P25 p25, P26 p26, P27 p27, P28 p28, P29 p29, P30 p30, P31 p31);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
```

```
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23, class P24, class P25,
class P26, class P27, class P28, class P29, class P30, class P31, class P32> Variant
OleFunction(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9,
P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16 p16, P17 p17, P18 p18, P19 p19, P20
p20, P21 p21, P22 p22, P23 p23, P24 p24, P25 p25, P26 p26, P27 p27, P28 p28, P29 p29, P30 p30,
P31 p31, P32 p32);
```

Description

System::Variant::OleFunction is a wrapper for calling Exec (see page 22320) for OLE Automation object functions. The System::Variant::Variant (see page 22314) object's value must be a COM interface.

name specifies the name of the function to call. Note that this parameter's value may be different for localized versions of the Automation server. As a result, it is a good idea to use a string resource for this value if the application is intended for multiple locales.

args is a TAutoArgsBase object that contains the parameter values. Typically, this is an instance of TAutoArgs. The args parameter can be omitted if the function has no parameters.

As an alternate syntax, there is a templatized version that can be called with up to 32 arguments, where the template parameters specify the types of each argument.

System::Variant::OleFunction returns the value returned by the COM object's function as a System::Variant::Variant (see page 22314).

Note: Do not use COM features in cross-platform applications. COM is only available under Windows.

See Also

System::Variant::Exec (see page 22320)

TAutoArgs

158.1.30.4.23 OleProcedure Method

This is the overview for the OleProcedure method overload.

Overload List

	Name	Description
	Variant::OleProcedure (String&, P1) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().

	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().

	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27, P28) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27, P28, P29) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27, P28, P29, P30) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27, P28, P29, P30, P31) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27, P28, P29, P30, P31, P32) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().
	Variant::OleProcedure (String&, TAutoArgsBase*) (see page 22332)	Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().

158.1.30.4.23.1 Variant::OleProcedure Method (String&, P1)

Provides alternative syntax to OLE Automation calls to Exec (see page 22320)().

Pascal

```
procedure OleProcedure(const name: String&; var args: TAutoArgsBase = 0);
procedure OleProcedure(const name: String&; p1: P1);
procedure OleProcedure(const name: String&; p1: P1; p2: P2);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8);
```

```
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16;
p17: P17);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16;
p17: P17; p18: P18);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16;
p17: P17; p18: P18; p19: P19);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16;
p17: P17; p18: P18; p19: P19; p20: P20);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16;
p17: P17; p18: P18; p19: P19; p20: P20; p21: P21);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16;
p17: P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16;
p17: P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16;
p17: P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23; p24: P24);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16;
p17: P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23; p24: P24; p25: P25);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16;
p17: P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23; p24: P24; p25: P25; p26:
P26);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16;
p17: P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23; p24: P24; p25: P25; p26:
P26; p27: P27);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16;
p17: P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23; p24: P24; p25: P25; p26:
P26; p27: P27; p28: P28);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16;
p17: P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23; p24: P24; p25: P25; p26:
P26; p27: P27; p28: P28; p29: P29);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16;
p17: P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23; p24: P24; p25: P25; p26:
P26; p27: P27; p28: P28; p29: P29; p30: P30);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16;
p17: P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23; p24: P24; p25: P25; p26:
```

```
P26; p27: P27; p28: P28; p29: P29; p30: P30; p31: P31);
procedure OleProcedure(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10; p11: P11; p12: P12; p13: P13; p14: P14; p15: P15; p16: P16;
p17: P17; p18: P18; p19: P19; p20: P20; p21: P21; p22: P22; p23: P23; p24: P24; p25: P25; p26:
P26; p27: P27; p28: P28; p29: P29; p30: P30; p31: P31; p32: P32);
```

C++

```
void OleProcedure(const String& name, TAutoArgsBase* args = 0);
template <class P1> void OleProcedure(const String& name, P1 p1);
template <class P1, class P2> void OleProcedure(const String& name, P1 p1, P2 p2);
template <class P1, class P2, class P3> void OleProcedure(const String& name, P1 p1, P2 p2, P3
p3);
template <class P1, class P2, class P3, class P4> void OleProcedure(const String& name, P1 p1,
P2 p2, P3 p3, P4 p4);
template <class P1, class P2, class P3, class P4, class P5> void OleProcedure(const String&
name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5);
template <class P1, class P2, class P3, class P4, class P5, class P6> void OleProcedure(const
String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7> void
OleProcedure(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8> void
OleProcedure(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9> void OleProcedure(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7
p7, P8 p8, P9 p9);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10> void OleProcedure(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5,
P6 p6, P7 p7, P8 p8, P9 p9, P10 p10);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11> void OleProcedure(const String& name, P1 p1, P2 p2, P3 p3, P4
p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12> void OleProcedure(const String& name, P1 p1, P2 p2,
P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13> void OleProcedure(const String& name, P1
p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13
p13);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14> void OleProcedure(const
String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11,
P12 p12, P13 p13, P14 p14);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15> void
OleProcedure(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9
p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16> void
OleProcedure(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9
p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16 p16);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17> void OleProcedure(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8
p8, P9 p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16 p16, P17 p17);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18> void OleProcedure(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6
p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16 p16, P17
p17, P18 p18);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19> void OleProcedure(const String& name, P1 p1, P2 p2, P3 p3, P4 p4,
P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16
p16, P17 p17, P18 p18, P19 p19);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
```

```
P17, class P18, class P19, class P20> void OleProcedure(const String& name, P1 p1, P2 p2, P3
p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15
p15, P16 p16, P17 p17, P18 p18, P19 p19, P20 p20);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21> void OleProcedure(const String& name, P1 p1,
P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13 p13,
P14 p14, P15 p15, P16 p16, P17 p17, P18 p18, P19 p19, P20 p20, P21 p21);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22> void OleProcedure(const String&
name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12
p12, P13 p13, P14 p14, P15 p15, P16 p16, P17 p17, P18 p18, P19 p19, P20 p20, P21 p21, P22 p22);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23> void OleProcedure(const
String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11,
P12 p12, P13 p13, P14 p14, P15 p15, P16 p16, P17 p17, P18 p18, P19 p19, P20 p20, P21 p21, P22
p22, P23 p23);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23, class P24> void
OleProcedure(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9
p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16 p16, P17 p17, P18 p18, P19 p19,
P20 p20, P21 p21, P22 p22, P23 p23, P24 p24);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23, class P24, class P25>
void OleProcedure(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8,
P9 p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16 p16, P17 p17, P18 p18, P19
p19, P20 p20, P21 p21, P22 p22, P23 p23, P24 p24, P25 p25);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23, class P24, class P25,
class P26> void OleProcedure(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7
p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16 p16, P17 p17, P18
p18, P19 p19, P20 p20, P21 p21, P22 p22, P23 p23, P24 p24, P25 p25, P26 p26);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23, class P24, class P25,
class P26, class P27> void OleProcedure(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5,
P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16 p16, P17
p17, P18 p18, P19 p19, P20 p20, P21 p21, P22 p22, P23 p23, P24 p24, P25 p25, P26 p26, P27 p27);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23, class P24, class P25,
class P26, class P27, class P28> void OleProcedure(const String& name, P1 p1, P2 p2, P3 p3, P4
p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15,
P16 p16, P17 p17, P18 p18, P19 p19, P20 p20, P21 p21, P22 p22, P23 p23, P24 p24, P25 p25, P26
p26, P27 p27, P28 p28);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23, class P24, class P25,
class P26, class P27, class P28, class P29> void OleProcedure(const String& name, P1 p1, P2
p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13 p13, P14
p14, P15 p15, P16 p16, P17 p17, P18 p18, P19 p19, P20 p20, P21 p21, P22 p22, P23 p23, P24 p24,
P25 p25, P26 p26, P27 p27, P28 p28, P29 p29);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23, class P24, class P25,
class P26, class P27, class P28, class P29, class P30> void OleProcedure(const String& name,
P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11, P12 p12, P13
p13, P14 p14, P15 p15, P16 p16, P17 p17, P18 p18, P19 p19, P20 p20, P21 p21, P22 p22, P23 p23,
P24 p24, P25 p25, P26 p26, P27 p27, P28 p28, P29 p29, P30 p30);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
```

```
P17, class P18, class P19, class P20, class P21, class P22, class P23, class P24, class P25,
class P26, class P27, class P28, class P29, class P30, class P31> void OleProcedure(const
String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10, P11 p11,
P12 p12, P13 p13, P14 p14, P15 p15, P16 p16, P17 p17, P18 p18, P19 p19, P20 p20, P21 p21, P22
p22, P23 p23, P24 p24, P25 p25, P26 p26, P27 p27, P28 p28, P29 p29, P30 p30, P31 p31);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10, class P11, class P12, class P13, class P14, class P15, class P16, class
P17, class P18, class P19, class P20, class P21, class P22, class P23, class P24, class P25,
class P26, class P27, class P28, class P29, class P30, class P31, class P32> void
OleProcedure(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8, P9
p9, P10 p10, P11 p11, P12 p12, P13 p13, P14 p14, P15 p15, P16 p16, P17 p17, P18 p18, P19 p19,
P20 p20, P21 p21, P22 p22, P23 p23, P24 p24, P25 p25, P26 p26, P27 p27, P28 p28, P29 p29, P30
p30, P31 p31, P32 p32);
```

Description

System::Variant::OleProcedure is a wrapper for calling Exec (see page 22320) for OLE procedures (functions that do not return a value). The System::Variant::Variant (see page 22314) object's value must be a COM interface.

name specifies the name of the procedure to call. Note that this parameter's value may be different for localized versions of the Automation server. As a result, it is a good idea to use a string resource for this value if the application is intended for multiple locales.

args is a TAutoArgsBase object that contains the parameter values. Typically, this is an instance of TAutoArgs. The args parameter can be omitted if the procedure has no parameters.

As an alternate syntax, there is a templatized version that can be called with up to 32 arguments, where the template parameters specify the types of each argument.

Note: Do not use COM features in cross-platform applications. COM is only available under Windows.

See Also

System::Variant::Exec (see page 22320)

TAutoArgs

158.1.30.4.24 OlePropertyGet Method

This is the overview for the OlePropertyGet method overload.

Overload List

	Name	Description
	Variant::OlePropertyGet (String&, P1) (see page 22337)	Returns the value of a property on a COM interface.
	Variant::OlePropertyGet (String&, P1, P2) (see page 22337)	Returns the value of a property on a COM interface.
	Variant::OlePropertyGet (String&, P1, P2, P3) (see page 22337)	Returns the value of a property on a COM interface.
	Variant::OlePropertyGet (String&, P1, P2, P3, P4) (see page 22337)	Returns the value of a property on a COM interface.
	Variant::OlePropertyGet (String&, P1, P2, P3, P4, P5) (see page 22337)	Returns the value of a property on a COM interface.
	Variant::OlePropertyGet (String&, P1, P2, P3, P4, P5, P6) (see page 22337)	Returns the value of a property on a COM interface.
	Variant::OlePropertyGet (String&, P1, P2, P3, P4, P5, P6, P7) (see page 22337)	Returns the value of a property on a COM interface.

	Variant::OlePropertyGet (String&, P1, P2, P3, P4, P5, P6, P7, P8) (see page 22337)	Returns the value of a property on a COM interface.
	Variant::OlePropertyGet (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9) (see page 22337)	Returns the value of a property on a COM interface.
	Variant::OlePropertyGet (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10) (see page 22337)	Returns the value of a property on a COM interface.
	Variant::OlePropertyGet (String&, TAutoArgsBase*) (see page 22337)	Returns the value of a property on a COM interface.

158.1.30.4.24.1 Variant::OlePropertyGet Method (String&, P1)

Returns the value of a property on a COM interface.

Pascal

```
function OlePropertyGet(const name: String&; var args: TAutoArgsBase = 0): Variant;
function OlePropertyGet(const name: String&; p1: P1): Variant;
function OlePropertyGet(const name: String&; p1: P1; p2: P2): Variant;
function OlePropertyGet(const name: String&; p1: P1; p2: P2; p3: P3): Variant;
function OlePropertyGet(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4): Variant;
function OlePropertyGet(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5): Variant;
function OlePropertyGet(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6):
Variant;
function OlePropertyGet(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7): Variant;
function OlePropertyGet(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8): Variant;
function OlePropertyGet(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9): Variant;
function OlePropertyGet(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10): Variant;
```

C++

```
Variant OlePropertyGet(const String& name, TAutoArgsBase* args = 0);
template <class P1> Variant OlePropertyGet(const String& name, P1 p1);
template <class P1, class P2> Variant OlePropertyGet(const String& name, P1 p1, P2 p2);
template <class P1, class P2, class P3> Variant OlePropertyGet(const String& name, P1 p1, P2
p2, P3 p3);
template <class P1, class P2, class P3, class P4> Variant OlePropertyGet(const String& name,
P1 p1, P2 p2, P3 p3, P4 p4);
template <class P1, class P2, class P3, class P4, class P5> Variant OlePropertyGet(const
String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5);
template <class P1, class P2, class P3, class P4, class P5, class P6> Variant
OlePropertyGet(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7> Variant
OlePropertyGet(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8>
Variant OlePropertyGet(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8
p8);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9> Variant OlePropertyGet(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6,
P7 p7, P8 p8, P9 p9);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10> Variant OlePropertyGet(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5
p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10);
```

Description

System::Variant::OlePropertyGet is a wrapper for calling Exec (see page 22320)(). System::Variant::OlePropertyGet returns a property value on the interface that is the System::Variant::Variant (see page 22314) object's value.

name is the name of the property whose value is returned. Note that this parameter's value may be different for localized

versions of the Automation server. As a result, it is a good idea to use a string resource for this value if the application is intended for multiple locales.

args is a TAutoArgsBase object that contains the parameter values for the property's read access method. Typically, this is an instance of TAutoArgs. The args parameter can be omitted if the access method has no parameters.

As an alternate syntax, there is a templatized version that can be called with up to 10 arguments, where the template parameters specify the types of each argument.

Note: Do not use COM features in cross-platform applications. COM is only available under Windows.

See Also

System::Variant::Exec (see page 22320)

TAutoArgs

158.1.30.4.25 OlePropertySet Method

This is the overview for the OlePropertySet method overload.

Overload List

	Name	Description
	Variant::OlePropertySet (String&, P1) (see page 22338)	Sets the value of a property on a COM interface.
	Variant::OlePropertySet (String&, P1, P2) (see page 22338)	Sets the value of a property on a COM interface.
	Variant::OlePropertySet (String&, P1, P2, P3) (see page 22338)	Sets the value of a property on a COM interface.
	Variant::OlePropertySet (String&, P1, P2, P3, P4) (see page 22338)	Sets the value of a property on a COM interface.
	Variant::OlePropertySet (String&, P1, P2, P3, P4, P5) (see page 22338)	Sets the value of a property on a COM interface.
	Variant::OlePropertySet (String&, P1, P2, P3, P4, P5, P6) (see page 22338)	Sets the value of a property on a COM interface.
	Variant::OlePropertySet (String&, P1, P2, P3, P4, P5, P6, P7) (see page 22338)	Sets the value of a property on a COM interface.
	Variant::OlePropertySet (String&, P1, P2, P3, P4, P5, P6, P7, P8) (see page 22338)	Sets the value of a property on a COM interface.
	Variant::OlePropertySet (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9) (see page 22338)	Sets the value of a property on a COM interface.
	Variant::OlePropertySet (String&, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10) (see page 22338)	Sets the value of a property on a COM interface.
	Variant::OlePropertySet (String&, TAutoArgsBase&) (see page 22338)	Sets the value of a property on a COM interface.

158.1.30.4.25.1 Variant::OlePropertySet Method (String&, P1)

Sets the value of a property on a COM interface.

Pascal

```
procedure OlePropertySet(const name: String&; var args: TAutoArgsBase);
```

```
procedure OlePropertySet(const name: String&; p1: P1);
procedure OlePropertySet(const name: String&; p1: P1; p2: P2);
procedure OlePropertySet(const name: String&; p1: P1; p2: P2; p3: P3);
procedure OlePropertySet(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4);
procedure OlePropertySet(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5);
procedure OlePropertySet(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6);
procedure OlePropertySet(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7);
procedure OlePropertySet(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8);
procedure OlePropertySet(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9);
procedure OlePropertySet(const name: String&; p1: P1; p2: P2; p3: P3; p4: P4; p5: P5; p6: P6;
p7: P7; p8: P8; p9: P9; p10: P10);
```

C++

```
void OlePropertySet(const String& name, TAutoArgsBase& args);
template <class P1> void OlePropertySet(const String& name, P1 p1);
template <class P1, class P2> void OlePropertySet(const String& name, P1 p1, P2 p2);
template <class P1, class P2, class P3> void OlePropertySet(const String& name, P1 p1, P2 p2,
P3 p3);
template <class P1, class P2, class P3, class P4> void OlePropertySet(const String& name, P1
p1, P2 p2, P3 p3, P4 p4);
template <class P1, class P2, class P3, class P4, class P5> void OlePropertySet(const String&
name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5);
template <class P1, class P2, class P3, class P4, class P5, class P6> void
OlePropertySet(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7> void
OlePropertySet(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8> void
OlePropertySet(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7 p7, P8 p8);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9> void OlePropertySet(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5 p5, P6 p6, P7
p7, P8 p8, P9 p9);
template <class P1, class P2, class P3, class P4, class P5, class P6, class P7, class P8,
class P9, class P10> void OlePropertySet(const String& name, P1 p1, P2 p2, P3 p3, P4 p4, P5
p5, P6 p6, P7 p7, P8 p8, P9 p9, P10 p10);
```

Description

System::Variant::OlePropertySet is a wrapper for calling Exec (see page 22320). You can use System::Variant::OlePropertySet to set the value of a property on an interface that is the System::Variant::Variant (see page 22314) object's value.

name is the name of the property whose value is set. Note that this parameter's value may be different for localized versions of the Automation server. As a result, it is a good idea to use a string resource for this value if the application is intended for multiple locales.

args is a TAutoArgsBase object that contains the parameter values of the property's write access method. Typically, this is an instance of TAutoArgs with one parameter (the value to set).

As an alternate syntax, there is a templatized version that can be called with up to 10 arguments, where the template parameters specify the types of each argument.

Note: Do not use COM features in cross-platform applications. COM is only available under Windows.

See Also

System::Variant::Exec (see page 22320)

TAutoArgs

158.1.30.4.26 PutElement Method

This is the overview for the PutElement method overload.

Overload List

	Name	Description
	Variant::PutElement (Variant&, int) (see page 22340)	Assigns an element as the value of an entry in a System::Variant::Variant (see page 22314) array.
	Variant::PutElement (Variant&, int, int) (see page 22340)	Assigns an element as the value of an entry in a System::Variant::Variant (see page 22314) array.
	Variant::PutElement (Variant&, int, int, int) (see page 22340)	Assigns an element as the value of an entry in a System::Variant::Variant (see page 22314) array.
	Variant::PutElement (Variant&, int, int, int, int) (see page 22340)	Assigns an element as the value of an entry in a System::Variant::Variant (see page 22314) array.
	Variant::PutElement (Variant&, int, int, int, int, int) (see page 22340)	Assigns an element as the value of an entry in a System::Variant::Variant (see page 22314) array.

158.1.30.4.26.1 Variant::PutElement Method (Variant&, int)

Assigns an element as the value of an entry in a System::Variant::Variant (see page 22314) array.

Pascal

```
procedure PutElement(const data: Variant&; const i1: Integer);
procedure PutElement(const data: Variant&; const i1: Integer; const i2: Integer);
procedure PutElement(const data: Variant&; const i1: Integer; const i2: Integer; const i3:
Integer);
procedure PutElement(const data: Variant&; const i1: Integer; const i2: Integer; const i3:
Integer; const i4: Integer);
procedure PutElement(const data: Variant&; const i1: Integer; const i2: Integer; const i3:
Integer; const i4: Integer; const i5: Integer);
```

C++

```
__fastcall void PutElement(const Variant& data, const int i1);
__fastcall void PutElement(const Variant& data, const int i1, const int i2);
__fastcall void PutElement(const Variant& data, const int i1, const int i2, const int i3);
__fastcall void PutElement(const Variant& data, const int i1, const int i2, const int i3,
const int i4);
__fastcall void PutElement(const Variant& data, const int i1, const int i2, const int i3,
const int i4, const int i5);
```

Description

Use System::Variant::PutElement to access one-dimensional and multi-dimensional System::Variant::Variant (see page 22314) arrays of up to 5 dimensions. The data parameter is the value to assign. The remaining parameters provide indexes into the dimensions of the array.

See Also

System::Variant::GetElement (see page 22322)

158.1.30.4.27 Variant::Sanitized Method

Pascal

```
function Sanitized: Variant;
```

C++

```
__fastcall Variant Sanitized() const;
```

Description

This is Sanitized, a member of class Variant.

158.1.30.4.28 Variant::SetError Method

Converts the System::Variant::Variant (see page 22314) to a varError (see page 22684) type and assigns the integer err as the value of the varError (see page 22684).

Pascal

```
procedure SetError(const err: HResult);
```

C++

```
__fastcall void SetError(const HResult err);
```

Description

See the base class TVarData (see page 22571) for more information about varError (see page 22684).

See Also

TVarData (see page 22571)

158.1.30.4.29 Variant::Type Method

Pascal

```
function Type: Integer;
```

C++

```
__fastcall int Type() const;
```

Description

This is Type, a member of class Variant.

158.1.30.5 Variant Operators

The operators of the Variant class are listed here.

Public Operators

	Name	Description
	- (see page 22343)	This is the overview for the - operator overload.
	! (see page 22344)	Returns a new System::Variant::Variant (see page 22314) that is the logical negation of this System::Variant::Variant (see page 22314).
	!= (see page 22344)	This is the overview for the != operator overload.
	% (see page 22345)	This is the overview for the % operator overload.
	%= (see page 22346)	Returns this System::Variant::Variant (see page 22314) as the remainder from dividing its original value and the System::Variant::Variant (see page 22314) specified by rhs.
	& (see page 22346)	This is the overview for the & operator overload.
	&= (see page 22347)	Returns this System::Variant::Variant (see page 22314) after performing a bitwise and operation.
	* (see page 22347)	This is the overview for the * operator overload.

	*= (see page 22348)	Multiplies this System::Variant::Variant (see page 22314) by the System::Variant::Variant (see page 22314) specified by rhs.
	/ (see page 22348)	This is the overview for the / operator overload.
	/= (see page 22349)	Divides this System::Variant::Variant (see page 22314) by the System::Variant::Variant (see page 22314) rhs.
	^ (see page 22349)	This is the overview for the ^ operator overload.
	^= (see page 22350)	Returns this System::Variant::Variant (see page 22314) after performing a bitwise xor operation.
	__int64 (see page 22350)	This is __int64, a member of class Variant.
	| (see page 22351)	This is the overview for the | operator overload.
	|= (see page 22351)	Returns this System::Variant::Variant (see page 22314) after performing a bitwise or operation.
	+ (see page 22352)	This is the overview for the + operator overload.
	+= (see page 22352)	Adds the System::Variant::Variant (see page 22314) specified by rhs to this System::Variant::Variant (see page 22314) object.
	< (see page 22353)	This is the overview for the < operator overload.
	<< (see page 22353)	This is the overview for the << operator overload.
	<<= (see page 22354)	Performs a bitwise shl (left shift) operation on this System::Variant::Variant (see page 22314).
	<= (see page 22355)	This is the overview for the <= operator overload.
	= (see page 22355)	This is the overview for the = operator overload.
	-= (see page 22361)	Subtracts the System::Variant::Variant (see page 22314) specified by rhs from this System::Variant::Variant (see page 22314) object.
	== (see page 22361)	This is the overview for the == operator overload.
	> (see page 22362)	This is the overview for the > operator overload.
	>= (see page 22363)	This is the overview for the >= operator overload.
	>> (see page 22363)	This is the overview for the >> operator overload.
	>>= (see page 22365)	Performs a bitwise shr (right shift) operation on this System::Variant::Variant (see page 22314).
	AnsiString (see page 22365)	Converts the System::Variant::Variant (see page 22314) to an AnsiString object.
	bool (see page 22365)	Converts the System::Variant::Variant (see page 22314) to a boolean value.
	char (see page 22365)	This is char, a member of class Variant.
	Currency (see page 22366)	Converts the System::Variant::Variant (see page 22314) to a Currency object.
	CURRENCY (see page 22366)	This is CURRENCY, a member of class Variant.
	Currency* (see page 22366)	This is Currency*, a member of class Variant.
	DECIMAL (see page 22366)	This is DECIMAL, a member of class Variant.
	double (see page 22367)	Converts the System::Variant::Variant (see page 22314) to a double value.
	double* (see page 22367)	This is double*, a member of class Variant.
	float (see page 22367)	Converts the System::Variant::Variant (see page 22314) to a float value.
	float* (see page 22368)	This is float*, a member of class Variant.
	IDispatch* (see page 22368)	This is IDispatch*, a member of class Variant.
	IInterface* (see page 22368)	This is IInterface*, a member of class Variant.
	int (see page 22368)	Converts the System::Variant::Variant (see page 22314) to an integer value.
	int* (see page 22369)	This is int*, a member of class Variant.
	IUnknown* (see page 22369)	This is IUnknown*, a member of class Variant.
	long (see page 22369)	This is long, a member of class Variant.
	long* (see page 22369)	This is long*, a member of class Variant.
	LPSAFEARRAY (see page 22369)	This is LPSAFEARRAY, a member of class Variant.
	short (see page 22370)	Converts the System::Variant::Variant (see page 22314) to a short value.
	short* (see page 22370)	This is short*, a member of class Variant.
	signed char (see page 22370)	This is signed char, a member of class Variant.
	signed char* (see page 22370)	This is signed char*, a member of class Variant.
	TDateTime (see page 22371)	Converts the System::Variant::Variant (see page 22314) to a TDateTime object.
	TDateTime* (see page 22371)	This is TDateTime*, a member of class Variant.
	TVariant (see page 22371)	This is TVariant, a member of class Variant.

	TVariant* (see page 22371)	This is TVariant*, a member of class Variant.
	unsigned char (see page 22372)	This is unsigned char, a member of class Variant.
	unsigned char* (see page 22372)	This is unsigned char*, a member of class Variant.
	unsigned int (see page 22372)	This is unsigned int, a member of class Variant.
	unsigned int* (see page 22372)	This is unsigned int*, a member of class Variant.
	unsigned long (see page 22373)	This is unsigned long, a member of class Variant.
	unsigned long* (see page 22373)	This is unsigned long*, a member of class Variant.
	unsigned short (see page 22373)	This is unsigned short, a member of class Variant.
	unsigned short* (see page 22373)	This is unsigned short*, a member of class Variant.
	VARIANT (see page 22373)	This is VARIANT, a member of class Variant.
	VARIANT* (see page 22374)	This is VARIANT*, a member of class Variant.
	wchar_t* (see page 22374)	Converts the System::Variant::Variant (see page 22314) to the specified type and sets the varByRef (see page 22680) bit, meaning that the variant contains a reference as opposed to a value.
	wchar_t** (see page 22374)	This is wchar_t**, a member of class Variant.
	WideString (see page 22374)	This is WideString, a member of class Variant.

158.1.30.5.1 - Operator

This is the overview for the - operator overload.

Overload List

	Name	Description
	Variant::- (AnsiString&) (see page 22343)	This is -, a member of class Variant.
	Variant::- () (see page 22343)	This is -, a member of class Variant.
	Variant::- (double) (see page 22344)	This is -, a member of class Variant.
	Variant::- (int) (see page 22344)	This is -, a member of class Variant.
	Variant::- (Variant&) (see page 22344)	This is -, a member of class Variant.

158.1.30.5.1.1 Variant::- Operator ()

Pascal

```
operator -;
```

C++

```
__fastcall Variant operator -() const;
```

Description

This is -, a member of class Variant.

158.1.30.5.1.2 Variant::- Operator (AnsiString&)

Pascal

```
operator -(const rhs: AnsiString&);
```

C++

```
__fastcall Variant operator -(const AnsiString& rhs) const;
```

Description

This is -, a member of class Variant.

158.1.30.5.1.3 Variant::- Operator (Variant&)

Pascal

```
operator -(const rhs: Variant&);
```

C++

```
__fastcall Variant operator -(const Variant& rhs) const;
```

Description

This is -, a member of class Variant.

158.1.30.5.1.4 Variant::- Operator (double)

Pascal

```
operator -(rhs: double);
```

C++

```
__fastcall Variant operator -(double rhs) const;
```

Description

This is -, a member of class Variant.

158.1.30.5.1.5 Variant::- Operator (int)

Pascal

```
operator -(rhs: Integer);
```

C++

```
__fastcall Variant operator -(int rhs) const;
```

Description

This is -, a member of class Variant.

158.1.30.5.2 Variant::! Operator

Returns a new System::Variant::Variant (see page 22314) that is the logical negation of this System::Variant::Variant (see page 22314).

Pascal

```
operator !;
```

C++

```
__fastcall Variant operator !() const;
```

Description

The System::Variant (see page 22307)::! returns 0 if the System::Variant::Variant (see page 22314) has a nonzero value and 1 if the value of the System::Variant::Variant (see page 22314) is 0 or NULL.

158.1.30.5.3 != Operator

This is the overview for the != operator overload.

Overload List

	Name	Description
	Variant::!= (char*) (see page 22345)	Compares the System::Variant::Variant (see page 22314) with another value, testing for inequality.
	Variant::!= (double) (see page 22345)	Compares the System::Variant::Variant (see page 22314) with another value, testing for inequality.
	Variant::!= (int) (see page 22345)	Compares the System::Variant::Variant (see page 22314) with another value, testing for inequality.
	Variant::!= (TDateTime) (see page 22345)	Compares the System::Variant::Variant (see page 22314) with another value, testing for inequality.
	Variant::!= (Variant&) (see page 22345)	Compares the System::Variant::Variant (see page 22314) with another value, testing for inequality.

158.1.30.5.3.1 Variant::!= Operator (TDateTime)

Compares the System::Variant::Variant (see page 22314) with another value, testing for inequality.

Pascal

```
operator !=(const rhs: Variant&);
operator !=(const rhs: char*);
operator !=(rhs: Integer);
operator !=(rhs: double);
operator !=(rhs: TDateTime);
```

C++

```
__fastcall bool operator !=(const Variant& rhs) const;
__fastcall bool operator !=(const char* rhs) const;
__fastcall bool operator !=(int rhs) const;
__fastcall bool operator !=(double rhs) const;
__fastcall bool operator !=(TDateTime rhs) const;
```

Description

The != System::Variant (see page 22307)::!= returns true if the System::Variant::Variant (see page 22314)'s value is not the same as the value specified by rhs, false otherwise. If the comparison is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.4 % Operator

This is the overview for the % operator overload.

Overload List

	Name	Description
	Variant::% (AnsiString&) (see page 22346)	Returns the remainder from dividing the System::Variant::Variant (see page 22314)'s value by a specified value.
	Variant::% (double) (see page 22346)	Returns the remainder from dividing the System::Variant::Variant (see page 22314)'s value by a specified value.
	Variant::% (int) (see page 22346)	Returns the remainder from dividing the System::Variant::Variant (see page 22314)'s value by a specified value.

	Variant::% (Variant&) (see page 22346)	Returns the remainder from dividing the System::Variant::Variant (see page 22314)'s value by a specified value.

158.1.30.5.4.1 Variant::% Operator (AnsiString&)

Returns the remainder from dividing the System::Variant::Variant (see page 22314)'s value by a specified value.

Pascal

```
operator %(const rhs: Variant&);
operator %(const rhs: AnsiString&);
operator %(rhs: Integer);
operator %(rhs: double);
```

C++

```
__fastcall Variant operator %(const Variant& rhs) const;
__fastcall Variant operator %(const AnsiString& rhs) const;
__fastcall Variant operator %(int rhs) const;
__fastcall Variant operator %(double rhs) const;
```

Description

The System::Variant (see page 22307)::% returns a new System::Variant::Variant (see page 22314) that is the modulus of this System::Variant::Variant (see page 22314) and rhs.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.5 Variant::%= Operator

Returns this System::Variant::Variant (see page 22314) as the remainder from dividing its original value and the System::Variant::Variant (see page 22314) specified by rhs.

Pascal

```
operator %=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator %=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 23567) determine the meaning of this operation. If the System::Variant::Variants (see page 23567) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.6 & Operator

This is the overview for the & operator overload.

Overload List

	Name	Description
	Variant::& (AnsiString&) (see page 22347)	Performs a bitwise and operation with a specified value.
	Variant::& (double) (see page 22347)	Performs a bitwise and operation with a specified value.

	Variant::& (int) (see page 22347)	Performs a bitwise and operation with a specified value.
	Variant::& (Variant&) (see page 22347)	Performs a bitwise and operation with a specified value.

158.1.30.5.6.1 Variant::& Operator (AnsiString&)

Performs a bitwise and operation with a specified value.

Pascal

```
operator &(const rhs: Variant&);
operator &(const rhs: AnsiString&);
operator &(rhs: Integer);
operator &(rhs: double);
```

C++

```
__fastcall Variant operator &(const Variant& rhs) const;
__fastcall Variant operator &(const AnsiString& rhs) const;
__fastcall Variant operator &(int rhs) const;
__fastcall Variant operator &(double rhs) const;
```

Description

The System::Variant (see page 22307)::& returns a new System::Variant::Variant (see page 22314) that is the result of a bitwise and operation on this System::Variant::Variant (see page 22314) using rhs as the rvalue.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.7 Variant::&= Operator

Returns this System::Variant::Variant (see page 22314) after performing a bitwise and operation.

Pascal

```
operator &=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator &=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 23567) determine the meaning of this operation. If the System::Variant::Variants (see page 23567) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.8 * Operator

This is the overview for the * operator overload.

Overload List

	Name	Description
	Variant::* (AnsiString&) (see page 22348)	Multiplies the System::Variant::Variant (see page 22314)'s value by a specified value.
	Variant::* (double) (see page 22348)	Multiplies the System::Variant::Variant (see page 22314)'s value by a specified value.

	Variant::* (int) (see page 22348)	Multiplies the System::Variant::Variant (see page 22314)'s value by a specified value.
	Variant::* (Variant&) (see page 22348)	Multiplies the System::Variant::Variant (see page 22314)'s value by a specified value.

158.1.30.5.8.1 Variant::* Operator (AnsiString&)

Multiplies the System::Variant::Variant (see page 22314)'s value by a specified value.

Pascal

```
operator *(const rhs: Variant&);
operator *(const rhs: AnsiString&);
operator *(rhs: Integer);
operator *(rhs: double);
```

C++

```
__fastcall Variant operator *(const Variant& rhs) const;
__fastcall Variant operator *(const AnsiString& rhs) const;
__fastcall Variant operator *(int rhs) const;
__fastcall Variant operator *(double rhs) const;
```

Description

The System::Variant (see page 22307)::* returns a new System::Variant::Variant (see page 22314) that is the result of multiplying this System::Variant::Variant (see page 22314) with rhs.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.9 Variant::*= Operator

Multiplies this System::Variant::Variant (see page 22314) by the System::Variant::Variant (see page 22314) specified by rhs.

Pascal

```
operator *=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator *=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 23567) determine the meaning of this operation. If the System::Variant::Variants (see page 23567) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.10 / Operator

This is the overview for the / operator overload.

Overload List

	Name	Description
	Variant::/ (AnsiString&) (see page 22349)	Divides the System::Variant::Variant (see page 22314)'s value by a specified value.

	Variant::/ (double) (see page 22349)	Divides the System::Variant::Variant (see page 22314)'s value by a specified value.
	Variant::/ (int) (see page 22349)	Divides the System::Variant::Variant (see page 22314)'s value by a specified value.
	Variant::/ (Variant&) (see page 22349)	Divides the System::Variant::Variant (see page 22314)'s value by a specified value.

158.1.30.5.10.1 Variant::/ Operator (AnsiString&)

Divides the System::Variant::Variant (see page 22314)'s value by a specified value.

Pascal

```
operator /(const rhs: Variant&);
operator /(const rhs: AnsiString&);
operator /(rhs: Integer);
operator /(rhs: double);
```

C++

```
__fastcall Variant operator /(const Variant& rhs) const;
__fastcall Variant operator /(const AnsiString& rhs) const;
__fastcall Variant operator /(int rhs) const;
__fastcall Variant operator /(double rhs) const;
```

Description

The System::Variant (see page 22307)::/ returns a new System::Variant::Variant (see page 22314) that is the result of dividing this System::Variant::Variant (see page 22314) by rhs.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.11 Variant::/= Operator

Divides this System::Variant::Variant (see page 22314) by the System::Variant::Variant (see page 22314) rhs.

Pascal

```
operator /=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator /=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 23567) determine the meaning of this operation. If the System::Variant::Variants (see page 23567) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.12 ^ Operator

This is the overview for the ^ operator overload.

Overload List

	Name	Description
	Variant::^ (AnsiString&) (see page 22350)	Performs a bitwise xor operation with a specified value.
	Variant::^ (double) (see page 22350)	Performs a bitwise xor operation with a specified value.
	Variant::^ (int) (see page 22350)	Performs a bitwise xor operation with a specified value.
	Variant::^ (Variant&) (see page 22350)	Performs a bitwise xor operation with a specified value.

158.1.30.5.12.1 Variant::^ Operator (AnsiString&)

Performs a bitwise xor operation with a specified value.

Pascal

```
operator ^(const rhs: Variant&);
operator ^(const rhs: AnsiString&);
operator ^(rhs: Integer);
operator ^(rhs: double);
```

C++

```
__fastcall Variant operator ^(const Variant& rhs) const;
__fastcall Variant operator ^(const AnsiString& rhs) const;
__fastcall Variant operator ^(int rhs) const;
__fastcall Variant operator ^(double rhs) const;
```

Description

The ^ System::Variant (see page 22307)::^ returns a new System::Variant::Variant (see page 22314) that is the result of a bitwise xor operation on this System::Variant::Variant (see page 22314) using rhs as the rvalue.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.13 Variant::^= Operator

Returns this System::Variant::Variant (see page 22314) after performing a bitwise xor operation.

Pascal

```
operator ^=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator ^=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 23567) determine the meaning of this operation. If the System::Variant::Variants (see page 23567) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.14 Variant::__int64 Operator

Pascal

```
operator __int64;
```

C++

```
__fastcall operator __int64() const;
```

Description

This is __int64, a member of class Variant.

158.1.30.5.15 | Operator

This is the overview for the | operator overload.

Overload List

	Name	Description
	Variant::| (AnsiString&) (see page 22351)	Performs a bitwise or operation with a specified value.
	Variant::| (double) (see page 22351)	Performs a bitwise or operation with a specified value.
	Variant::| (int) (see page 22351)	Performs a bitwise or operation with a specified value.
	Variant::| (Variant&) (see page 22351)	Performs a bitwise or operation with a specified value.

158.1.30.5.15.1 Variant::| Operator (AnsiString&)

Performs a bitwise or operation with a specified value.

Pascal

```
operator |(const rhs: Variant&);
operator |(const rhs: AnsiString&);
operator |(rhs: Integer);
operator |(rhs: double);
```

C++

```
__fastcall Variant operator |(const Variant& rhs) const;
__fastcall Variant operator |(const AnsiString& rhs) const;
__fastcall Variant operator |(int rhs) const;
__fastcall Variant operator |(double rhs) const;
```

Description

The | System::Variant (see page 22307)::| returns a new System::Variant::Variant (see page 22314) that is the result of a bitwise or operation on this System::Variant::Variant (see page 22314) using rhs as the rvalue.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.16 Variant::|= Operator

Returns this System::Variant::Variant (see page 22314) after performing a bitwise or operation.

Pascal

```
operator |=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator |=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 23567) determine the meaning of this operation. If the

System::Variant::Variants (see page 23567) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.17 + Operator

This is the overview for the + operator overload.

Overload List

	Name	Description
	Variant::+ (double) (see page 22352)	Adds a specified value to the System::Variant::Variant (see page 22314).
	Variant::+ (int) (see page 22352)	Adds a specified value to the System::Variant::Variant (see page 22314).
	Variant::+ (Variant&) (see page 22352)	Adds a specified value to the System::Variant::Variant (see page 22314).

158.1.30.5.17.1 Variant::+ Operator (Variant&)

Adds a specified value to the System::Variant::Variant (see page 22314).

Pascal

```
operator +(const rhs: Variant&);
operator +(rhs: Integer);
operator +(rhs: double);
```

C++

```
__fastcall Variant operator +(const Variant& rhs) const;
__fastcall Variant operator +(int rhs) const;
__fastcall Variant operator +(double rhs) const;
```

Description

The + System::Variant (see page 22307)::+ returns a new System::Variant::Variant (see page 22314) that is the sum of this System::Variant::Variant (see page 22314) and rhs.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.18 Variant::+= Operator

Adds the System::Variant::Variant (see page 22314) specified by rhs to this System::Variant::Variant (see page 22314) object.

Pascal

```
operator +=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator +=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 23567) determine the meaning of this operation. If the System::Variant::Variants (see page 23567) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an

ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.19 < Operator

This is the overview for the < operator overload.

Overload List

	Name	Description
	Variant::< (char*) (see page 22353)	Tests whether the System::Variant::Variant (see page 22314)'s value is less than a specified value.
	Variant::< (double) (see page 22353)	Tests whether the System::Variant::Variant (see page 22314)'s value is less than a specified value.
	Variant::< (int) (see page 22353)	Tests whether the System::Variant::Variant (see page 22314)'s value is less than a specified value.
	Variant::< (TDateTime) (see page 22353)	Tests whether the System::Variant::Variant (see page 22314)'s value is less than a specified value.
	Variant::< (Variant&) (see page 22353)	Tests whether the System::Variant::Variant (see page 22314)'s value is less than a specified value.

158.1.30.5.19.1 Variant::< Operator (TDateTime)

Tests whether the System::Variant::Variant (see page 22314)'s value is less than a specified value.

Pascal

```
operator <(const rhs: Variant&);
operator <(const rhs: char*);
operator <(rhs: Integer);
operator <(rhs: double);
operator <(rhs: TDateTime);
```

C++

```
__fastcall bool operator <(const Variant& rhs) const;
__fastcall bool operator <(const char* rhs) const;
__fastcall bool operator <(int rhs) const;
__fastcall bool operator <(double rhs) const;
__fastcall bool operator <(TDateTime rhs) const;
```

Description

The < System::Variant (see page 22307)::< returns true if the System::Variant::Variant (see page 22314)'s value is less than the value specified by rhs, false otherwise. If the comparison is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.20 << Operator

This is the overview for the << operator overload.

Overload List

	Name	Description
	Variant::<< (AnsiString&) (see page 22354)	This is <<, a member of class Variant.
	Variant::<< (double) (see page 22354)	This is <<, a member of class Variant.
	Variant::<< (int) (see page 22354)	This is <<, a member of class Variant.

	Variant::<< (Variant&) (see page 22354)	This is <<, a member of class Variant.

158.1.30.5.20.1 Variant::<< Operator (AnsiString&)

Pascal

```
operator <<(const rhs: AnsiString&);
```

C++

```
__fastcall Variant operator <<(const AnsiString& rhs) const;
```

Description

This is <<, a member of class Variant.

158.1.30.5.20.2 Variant::<< Operator (Variant&)

Pascal

```
operator <<(const rhs: Variant&);
```

C++

```
__fastcall Variant operator <<(const Variant& rhs) const;
```

Description

This is <<, a member of class Variant.

158.1.30.5.20.3 Variant::<< Operator (double)

Pascal

```
operator <<(rhs: double);
```

C++

```
__fastcall Variant operator <<(double rhs) const;
```

Description

This is <<, a member of class Variant.

158.1.30.5.20.4 Variant::<< Operator (int)

Pascal

```
operator <<(rhs: Integer);
```

C++

```
__fastcall Variant operator <<(int rhs) const;
```

Description

This is <<, a member of class Variant.

158.1.30.5.21 Variant::<<= Operator

Performs a bitwise shl (left shift) operation on this System::Variant::Variant (see page 22314).

Pascal

```
operator <<=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator <<=(const Variant& rhs);
```

Description

If the System::Variant::Variants (see page 23567) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.22 <= Operator

This is the overview for the <= operator overload.

Overload List

	Name	Description
	Variant::<= (char*) (see page 22355)	Tests whether the System::Variant::Variant (see page 22314)'s value is less than or equal to a specified value.
	Variant::<= (double) (see page 22355)	Tests whether the System::Variant::Variant (see page 22314)'s value is less than or equal to a specified value.
	Variant::<= (int) (see page 22355)	Tests whether the System::Variant::Variant (see page 22314)'s value is less than or equal to a specified value.
	Variant::<= (TDateTime) (see page 22355)	Tests whether the System::Variant::Variant (see page 22314)'s value is less than or equal to a specified value.
	Variant::<= (Variant&) (see page 22355)	Tests whether the System::Variant::Variant (see page 22314)'s value is less than or equal to a specified value.

158.1.30.5.22.1 Variant::<= Operator (TDateTime)

Tests whether the System::Variant::Variant (see page 22314)'s value is less than or equal to a specified value.

Pascal

```
operator <=(const rhs: Variant&);
operator <=(const rhs: char*);
operator <=(rhs: Integer);
operator <=(rhs: double);
operator <=(rhs: TDateTime);
```

C++

```
__fastcall bool operator <=(const Variant& rhs) const;
__fastcall bool operator <=(const char* rhs) const;
__fastcall bool operator <=(int rhs) const;
__fastcall bool operator <=(double rhs) const;
__fastcall bool operator <=(TDateTime rhs) const;
```

Description

The <= System::Variant (see page 22307)::<= returns true if the System::Variant::Variant (see page 22314)'s value is less than or equal to the value specified by rhs, false otherwise. If the comparison is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.23 = Operator

This is the overview for the = operator overload.

Overload List

	Name	Description
	Variant::= (__int64) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (__int64*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (AnsiString&) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (bool) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (char) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (char*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (Currency&) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (CURRENCY&) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (Currency*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (CURRENCY*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (DECIMAL&) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (DECIMAL*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (double) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (double*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (float) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (float*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).

	Variant::= (IDispatch*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (IDispatch**) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (IInterface*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (IInterface**) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (int) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (int*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (IUnknown*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (IUnknown**) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (long) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (long*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (SAFEARRAY&) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (SAFEARRAY*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (SAFEARRAY**) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (short) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (short*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (signed char) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (signed char*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).

	Variant::= (System::DelphiInterface<IDispatch>&) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (TDateTime&) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (TDateTime*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (TVariant&) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (TVariant*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (unsigned char) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (unsigned char*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (unsigned int) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (unsigned int*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (unsigned long) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (unsigned long*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (unsigned short) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (unsigned short*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (Variant&) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (VARIANT&) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (Variant*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (VARIANT*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).

	Variant::= (VariantError&) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (wchar_t*) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (wchar_t**) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).
	Variant::= (WideString&) (see page 22359)	Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).

158.1.30.5.23.1 Variant::= Operator (AnsiString&)

Copies the System::Variant::Variant (see page 22314) specified by rhs into this System::Variant::Variant (see page 22314).

Pascal

```
operator =(const rhs: Variant&);
operator =(const src: bool);
operator =(const src: char);
operator =(const src: signed char);
operator =(const src: unsigned char);
operator =(const src: short);
operator =(const src: unsigned short);
operator =(const src: Integer);
operator =(const src: unsigned int);
operator =(const src: long);
operator =(const src: unsigned long);
operator =(const src: float);
operator =(const src: double);
operator =(const src: __int64);
operator =(const src: CURRENCY&);
operator =(const src: DECIMAL&);
operator =(const src: SAFEARRAY&);
operator =(const rhs: VARIANT&);
operator =(var src: SAFEARRAY);
operator =(var rhs: VARIANT);
operator =(const rhs: TVariant&);
operator =(var rhs: TVariant);
operator =(const src: AnsiString&);
operator =(const src: Currency&);
operator =(const src: TDateTime&);
operator =(const src: VariantError&);
operator =(const src: WideString&);
operator =(const src: System::DelphiInterface<IDispatch>&);
operator =(var src: wchar_t);
operator =(var src: IUnknown);
operator =(var src: IInterface);
operator =(var src: IDispatch);
operator =(const src: char*);
operator =(var src: signed char);
operator =(var src: unsigned char);
operator =(var src: short);
operator =(var src: unsigned short);
operator =(var src: Integer);
operator =(var src: unsigned int);
operator =(var src: long);
operator =(var src: unsigned long);
operator =(var src: float);
```

```
operator =(var src: double);
operator =(var src: __int64);
operator =(var src: CURRENCY);
operator =(var src: SAFEARRAY*);
operator =(var src: DECIMAL);
operator =(var src: wchar_t*);
operator =(var src: IUnknown*);
operator =(var src: IInterface*);
operator =(var src: IDispatch*);
operator =(var src: Currency);
operator =(var src: TDateTime);
operator =(var src: Variant);
```

C++

```
__fastcall Variant& operator =(const Variant& rhs);
__fastcall Variant& operator =(const bool src);
__fastcall Variant& operator =(const char src);
__fastcall Variant& operator =(const signed char src);
__fastcall Variant& operator =(const unsigned char src);
__fastcall Variant& operator =(const short src);
__fastcall Variant& operator =(const unsigned short src);
__fastcall Variant& operator =(const int src);
__fastcall Variant& operator =(const unsigned int src);
__fastcall Variant& operator =(const long src);
__fastcall Variant& operator =(const unsigned long src);
__fastcall Variant& operator =(const float src);
__fastcall Variant& operator =(const double src);
__fastcall Variant& operator =(const __int64 src);
__fastcall Variant& operator =(const CURRENCY& src);
__fastcall Variant& operator =(const DECIMAL& src);
__fastcall Variant& operator =(const SAFEARRAY& src);
__fastcall Variant& operator =(const VARIANT& rhs);
__fastcall Variant& operator =(SAFEARRAY* src);
__fastcall Variant& operator =(VARIANT* rhs);
__fastcall Variant& operator =(const TVariant& rhs);
__fastcall Variant& operator =(TVariant* rhs);
__fastcall Variant& operator =(const AnsiString& src);
__fastcall Variant& operator =(const Currency& src);
__fastcall Variant& operator =(const TDateTime& src);
__fastcall Variant& operator =(const VariantError& src);
__fastcall Variant& operator =(const WideString& src);
__fastcall Variant& operator =(const System::DelphiInterface<IDispatch>& src);
__fastcall Variant& operator =(wchar_t* src);
__fastcall Variant& operator =(IUnknown* src);
__fastcall Variant& operator =(IInterface* src);
__fastcall Variant& operator =(IDispatch* src);
__fastcall Variant& operator =(const char* src);
__fastcall Variant& operator =(signed char* src);
__fastcall Variant& operator =(unsigned char* src);
__fastcall Variant& operator =(short* src);
__fastcall Variant& operator =(unsigned short* src);
__fastcall Variant& operator =(int* src);
__fastcall Variant& operator =(unsigned int* src);
__fastcall Variant& operator =(long* src);
__fastcall Variant& operator =(unsigned long* src);
__fastcall Variant& operator =(float* src);
__fastcall Variant& operator =(double* src);
__fastcall Variant& operator =(__int64* src);
__fastcall Variant& operator =(CURRENCY* src);
__fastcall Variant& operator =(SAFEARRAY** src);
__fastcall Variant& operator =(DECIMAL* src);
__fastcall Variant& operator =(wchar_t** src);
__fastcall Variant& operator =(IUnknown** src);
__fastcall Variant& operator =(IInterface** src);
__fastcall Variant& operator =(IDispatch** src);
__fastcall Variant& operator =(Currency* src);
```

```
__fastcall Variant& operator =(TDateTime* src);
__fastcall Variant& operator =(Variant* src);
```

Description

The actual types of the System::Variant::Variants (see page 23567) determine the meaning of this operation. If the System::Variant::Variants (see page 23567) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.24 Variant::-= Operator

Subtracts the System::Variant::Variant (see page 22314) specified by rhs from this System::Variant::Variant (see page 22314) object.

Pascal

```
operator -=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator -=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 23567) determine the meaning of this operation. If the System::Variant::Variants (see page 23567) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.25 == Operator

This is the overview for the == operator overload.

Overload List

	Name	Description
	Variant::== (AnsiString&) (see page 22361)	Compares the System::Variant::Variant (see page 22314) with another value, testing for equality.
	Variant::== (char*) (see page 22361)	Compares the System::Variant::Variant (see page 22314) with another value, testing for equality.
	Variant::== (double) (see page 22361)	Compares the System::Variant::Variant (see page 22314) with another value, testing for equality.
	Variant::== (int) (see page 22361)	Compares the System::Variant::Variant (see page 22314) with another value, testing for equality.
	Variant::== (TDateTime) (see page 22361)	Compares the System::Variant::Variant (see page 22314) with another value, testing for equality.
	Variant::== (Variant&) (see page 22361)	Compares the System::Variant::Variant (see page 22314) with another value, testing for equality.

158.1.30.5.25.1 Variant::== Operator (AnsiString&)

Compares the System::Variant::Variant (see page 22314) with another value, testing for equality.

Pascal

```
operator ==(const rhs: Variant&);
```

```
operator ==(const rhs: char*);
operator ==(rhs: Integer);
operator ==(const rhs: AnsiString&);
operator ==(rhs: double);
operator ==(rhs: TDateTime);
```

C++

```
__fastcall bool operator ==(const Variant& rhs) const;
__fastcall bool operator ==(const char* rhs) const;
__fastcall bool operator ==(int rhs) const;
__fastcall bool operator ==(const AnsiString& rhs) const;
__fastcall bool operator ==(double rhs) const;
__fastcall bool operator ==(TDateTime rhs) const;
```

Description

The == System::Variant (see page 22307)::== returns true if the System::Variant::Variant (see page 22314)'s value is the same as the value specified by rhs, false otherwise. If the comparison is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.26 > Operator

This is the overview for the > operator overload.

Overload List

	Name	Description
	Variant::> (char*) (see page 22362)	Tests whether the System::Variant::Variant (see page 22314)'s value is greater than a specified value.
	Variant::> (double) (see page 22362)	Tests whether the System::Variant::Variant (see page 22314)'s value is greater than a specified value.
	Variant::> (int) (see page 22362)	Tests whether the System::Variant::Variant (see page 22314)'s value is greater than a specified value.
	Variant::> (TDateTime) (see page 22362)	Tests whether the System::Variant::Variant (see page 22314)'s value is greater than a specified value.
	Variant::> (Variant&) (see page 22362)	Tests whether the System::Variant::Variant (see page 22314)'s value is greater than a specified value.

158.1.30.5.26.1 Variant::> Operator (TDateTime)

Tests whether the System::Variant::Variant (see page 22314)'s value is greater than a specified value.

Pascal

```
operator >(const rhs: Variant&);
operator >(const rhs: char*);
operator >(rhs: Integer);
operator >(rhs: double);
operator >(rhs: TDateTime);
```

C++

```
__fastcall bool operator >(const Variant& rhs) const;
__fastcall bool operator >(const char* rhs) const;
__fastcall bool operator >(int rhs) const;
__fastcall bool operator >(double rhs) const;
__fastcall bool operator >(TDateTime rhs) const;
```

Description

The > System::Variant (see page 22307)::> returns true if the System::Variant::Variant (see page 22314)'s value is greater than the value specified by rhs, false otherwise. If the comparison is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.27 >= Operator

This is the overview for the >= operator overload.

Overload List

	Name	Description
	Variant::>= (char*) (see page 22363)	Tests whether the System::Variant::Variant (see page 22314)'s value is greater than or equal to a specified value.
	Variant::>= (double) (see page 22363)	Tests whether the System::Variant::Variant (see page 22314)'s value is greater than or equal to a specified value.
	Variant::>= (int) (see page 22363)	Tests whether the System::Variant::Variant (see page 22314)'s value is greater than or equal to a specified value.
	Variant::>= (TDateTime) (see page 22363)	Tests whether the System::Variant::Variant (see page 22314)'s value is greater than or equal to a specified value.
	Variant::>= (Variant&) (see page 22363)	Tests whether the System::Variant::Variant (see page 22314)'s value is greater than or equal to a specified value.

158.1.30.5.27.1 Variant::>= Operator (TDateTime)

Tests whether the System::Variant::Variant (see page 22314)'s value is greater than or equal to a specified value.

Pascal

```
operator >=(const rhs: Variant&);
operator >=(const rhs: char*);
operator >=(rhs: Integer);
operator >=(rhs: double);
operator >=(rhs: TDateTime);
```

C++

```
__fastcall bool operator >=(const Variant& rhs) const;
__fastcall bool operator >=(const char* rhs) const;
__fastcall bool operator >=(int rhs) const;
__fastcall bool operator >=(double rhs) const;
__fastcall bool operator >=(TDateTime rhs) const;
```

Description

The >= System::Variant (see page 22307)::>= returns true if the System::Variant::Variant (see page 22314)'s value is greater than or equal to the value specified by rhs, false otherwise. If the comparison is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.28 >> Operator

This is the overview for the >> operator overload.

Overload List

	Name	Description
	Variant::>> (AnsiString&) (see page 22364)	This is >>, a member of class Variant.
	Variant::>> (double) (see page 22364)	This is >>, a member of class Variant.
	Variant::>> (int) (see page 22364)	This is >>, a member of class Variant.
	Variant::>> (Variant&) (see page 22364)	This is >>, a member of class Variant.

158.1.30.5.28.1 Variant::>> Operator (AnsiString&)

Pascal

```
operator >>(const rhs: AnsiString&);
```

C++

```
__fastcall Variant operator >>(const AnsiString& rhs) const;
```

Description

This is >>, a member of class Variant.

158.1.30.5.28.2 Variant::>> Operator (Variant&)

Pascal

```
operator >>(const rhs: Variant&);
```

C++

```
__fastcall Variant operator >>(const Variant& rhs) const;
```

Description

This is >>, a member of class Variant.

158.1.30.5.28.3 Variant::>> Operator (double)

Pascal

```
operator >>(rhs: double);
```

C++

```
__fastcall Variant operator >>(double rhs) const;
```

Description

This is >>, a member of class Variant.

158.1.30.5.28.4 Variant::>> Operator (int)

Pascal

```
operator >>(rhs: Integer);
```

C++

```
__fastcall Variant operator >>(int rhs) const;
```

Description

This is >>, a member of class Variant.

158.1.30.5.29 Variant::>>= Operator

Performs a bitwise shr (right shift) operation on this System::Variant::Variant (see page 22314).

Pascal

```
operator >>=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator >>=(const Variant& rhs);
```

Description

If the System::Variant::Variants (see page 23567) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.30 Variant::AnsiString Operator

Converts the System::Variant::Variant (see page 22314) to an AnsiString object.

Pascal

```
operator AnsiString;
```

C++

```
__fastcall operator AnsiString() const;
```

Description

This conversion System::Variant::AnsiString returns an AnsiString object that is a conversion of this System::Variant::Variant (see page 22314).

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.31 Variant::bool Operator

Converts the System::Variant::Variant (see page 22314) to a boolean value.

Pascal

```
operator bool;
```

C++

```
__fastcall operator bool() const;
```

Description

This conversion System::Variant::bool returns a bool that is a conversion of this System::Variant::Variant (see page 22314).

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.32 Variant::char Operator

Pascal

```
operator char;
```

C++

```
__fastcall operator char() const;
```

Description

This is char, a member of class Variant.

158.1.30.5.33 Variant::Currency Operator

Converts the System::Variant::Variant (see page 22314) to a Currency object.

Pascal

```
operator Currency;
```

C++

```
__fastcall operator Currency() const;
```

Description

The Currency conversion System::Variant::Currency returns a Currency or a reference to a Currency that is a conversion of this System::Variant::Variant (see page 22314). If the Currency* conversion is used, the System::Variant::Currency sets the varByRef (see page 22680) bit, meaning that the System::Variant::Variant (see page 22314) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.34 Variant::CURRENCY Operator

Pascal

```
operator CURRENCY;
```

C++

```
__fastcall operator CURRENCY() const;
```

Description

This is CURRENCY, a member of class Variant.

158.1.30.5.35 Variant::Currency* Operator

Pascal

```
operator Currency*;
```

C++

```
__fastcall operator Currency*();
```

Description

This is Currency*, a member of class Variant.

158.1.30.5.36 Variant::DECIMAL Operator

Pascal

```
operator DECIMAL;
```

C++

```
__fastcall operator DECIMAL() const;
```

Description

This is DECIMAL, a member of class Variant.

158.1.30.5.37 Variant::double Operator

Converts the System::Variant::Variant (see page 22314) to a double value.

Pascal

```
operator double;
```

C++

```
__fastcall operator double() const;
```

Description

The double conversion System::Variant::double returns a double or a reference to a double that is a conversion of this System::Variant::Variant (see page 22314). If the double* conversion is used, the System::Variant::double sets the varByRef (see page 22680) bit, meaning that the System::Variant::Variant (see page 22314) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.38 Variant::double* Operator

Pascal

```
operator double*;
```

C++

```
__fastcall operator double*();
```

Description

This is double*, a member of class Variant.

158.1.30.5.39 Variant::float Operator

Converts the System::Variant::Variant (see page 22314) to a float value.

Pascal

```
operator float;
```

C++

```
__fastcall operator float() const;
```

Description

The float conversion System::Variant::float returns a float or a reference to a float that is a conversion of this System::Variant::Variant (see page 22314). If the float* conversion is used, the System::Variant::float sets the varByRef (see page 22680) bit, meaning that the System::Variant::Variant (see page 22314) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.40 Variant::float* Operator

Pascal

```
operator float*;
```

C++

```
__fastcall operator float*();
```

Description

This is float*, a member of class Variant.

158.1.30.5.41 Variant::IDispatch* Operator

Pascal

```
operator IDispatch*;
```

C++

```
__fastcall operator IDispatch*();
```

Description

This is IDispatch*, a member of class Variant.

158.1.30.5.42 Variant::IInterface* Operator

Pascal

```
operator IInterface*;
```

C++

```
__fastcall operator IInterface*();
```

Description

This is IInterface*, a member of class Variant.

158.1.30.5.43 Variant::int Operator

Converts the System::Variant::Variant (see page 22314) to an integer value.

Pascal

```
operator int;
```

C++

```
__fastcall operator int() const;
```

Description

The int conversion System::Variant::int returns an integer or a reference to an integer that is a conversion of this System::Variant::Variant (see page 22314). If the int* conversion is used, the System::Variant::int sets the varByRef (see page 22680) bit, meaning that the System::Variant::Variant (see page 22314) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.44 Variant::int* Operator

Pascal

```
operator int*;
```

C++

```
__fastcall operator int*();
```

Description

This is int*, a member of class Variant.

158.1.30.5.45 Variant::IUnknown* Operator

Pascal

```
operator IUnknown*;
```

C++

```
__fastcall operator IUnknown*();
```

Description

This is IUnknown*, a member of class Variant.

158.1.30.5.46 Variant::long Operator

Pascal

```
operator long;
```

C++

```
__fastcall operator long() const;
```

Description

This is long, a member of class Variant.

158.1.30.5.47 Variant::long* Operator

Pascal

```
operator long*;
```

C++

```
__fastcall operator long*();
```

Description

This is long*, a member of class Variant.

158.1.30.5.48 Variant::LPSAFEARRAY Operator

Pascal

```
operator LPSAFEARRAY;
```

C++

```
__fastcall operator LPSAFEARRAY();
```

Description

This is LPSAFEARRAY, a member of class Variant.

158.1.30.5.49 Variant::short Operator

Converts the System::Variant::Variant (see page 22314) to a short value.

Pascal

```
operator short;
```

C++

```
__fastcall operator short() const;
```

Description

The short conversion System::Variant::short returns a short or a reference to a short that is a conversion of this System::Variant::Variant (see page 22314). If the short* conversion is used, the System::Variant::short sets the varByRef (see page 22680) bit, meaning that the System::Variant::Variant (see page 22314) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.50 Variant::short* Operator

Pascal

```
operator short*;
```

C++

```
__fastcall operator short*();
```

Description

This is short*, a member of class Variant.

158.1.30.5.51 Variant::signed char Operator

Pascal

```
operator signed char;
```

C++

```
__fastcall operator signed char() const;
```

Description

This is signed char, a member of class Variant.

158.1.30.5.52 Variant::signed char* Operator

Pascal

```
operator signed char*;
```

C++

```
__fastcall operator signed char*();
```

Description

This is signed char*, a member of class Variant.

158.1.30.5.53 Variant::TDateTime Operator

Converts the System::Variant::Variant (see page 22314) to a TDateTime object.

Pascal

```
operator TDateTime;
```

C++

```
__fastcall operator TDateTime() const;
```

Description

The TDateTime conversion System::Variant::TDateTime returns a TDateTime or a reference to a TDateTime that is a conversion of this System::Variant::Variant (see page 22314). If the TDateTime* conversion is used, the System::Variant::TDateTime sets the varByRef (see page 22680) bit, meaning that the System::Variant::Variant (see page 22314) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.54 Variant::TDateTime* Operator

Pascal

```
operator TDateTime*;
```

C++

```
__fastcall operator TDateTime*();
```

Description

This is TDateTime*, a member of class Variant.

158.1.30.5.55 Variant::TVariant Operator

Pascal

```
operator TVariant;
```

C++

```
__fastcall operator TVariant();
```

Description

This is TVariant, a member of class Variant.

158.1.30.5.56 Variant::TVariant* Operator

Pascal

```
operator TVariant*;
```

C++

```
__fastcall operator TVariant*();
```

Description

This is TVariant*, a member of class Variant.

158.1.30.5.57 Variant::unsigned char Operator

Pascal

```
operator unsigned char;
```

C++

```
__fastcall operator unsigned char() const;
```

Description

This is unsigned char, a member of class Variant.

158.1.30.5.58 Variant::unsigned char* Operator

Pascal

```
operator unsigned char*;
```

C++

```
__fastcall operator unsigned char*();
```

Description

This is unsigned char*, a member of class Variant.

158.1.30.5.59 Variant::unsigned int Operator

Pascal

```
operator unsigned int;
```

C++

```
__fastcall operator unsigned int() const;
```

Description

This is unsigned int, a member of class Variant.

158.1.30.5.60 Variant::unsigned int* Operator

Pascal

```
operator unsigned int*;
```

C++

```
__fastcall operator unsigned int*();
```

Description

This is unsigned int*, a member of class Variant.

158.1.30.5.61 Variant::unsigned long Operator

Pascal

```
operator unsigned long;
```

C++

```
__fastcall operator unsigned long() const;
```

Description

This is unsigned long, a member of class Variant.

158.1.30.5.62 Variant::unsigned long* Operator

Pascal

```
operator unsigned long*;
```

C++

```
__fastcall operator unsigned long*();
```

Description

This is unsigned long*, a member of class Variant.

158.1.30.5.63 Variant::unsigned short Operator

Pascal

```
operator unsigned short;
```

C++

```
__fastcall operator unsigned short() const;
```

Description

This is unsigned short, a member of class Variant.

158.1.30.5.64 Variant::unsigned short* Operator

Pascal

```
operator unsigned short*;
```

C++

```
__fastcall operator unsigned short*();
```

Description

This is unsigned short*, a member of class Variant.

158.1.30.5.65 Variant::VARIANT Operator

Pascal

```
operator VARIANT;
```

C++

```
__fastcall operator VARIANT();
```

Description

This is VARIANT, a member of class Variant.

158.1.30.5.66 Variant::VARIANT* Operator

Pascal

```
operator VARIANT*;
```

C++

```
__fastcall operator VARIANT*();
```

Description

This is VARIANT*, a member of class Variant.

158.1.30.5.67 Variant::wchar_t* Operator

Converts the System::Variant::Variant (see page 22314) to the specified type and sets the varByRef (see page 22680) bit, meaning that the variant contains a reference as opposed to a value.

Pascal

```
operator wchar_t*;
```

C++

```
__fastcall operator wchar_t*() const;
```

Description

This conversion System::Variant::wchar_t* returns a reference to a wide character array that is a conversion of this System::Variant::Variant (see page 22314).

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError (see page 22375) exception is thrown.

158.1.30.5.68 Variant::wchar_t** Operator

Pascal

```
operator wchar_t**;
```

C++

```
__fastcall operator wchar_t**();
```

Description

This is wchar_t**, a member of class Variant.

158.1.30.5.69 Variant::WideString Operator

Pascal

```
operator WideString;
```

C++

```
__fastcall operator WideString() const;
```

Description

This is WideString, a member of class Variant.

158.1.30.6 Variant Friends

The friends of the Variant class are listed here.

Public Friends

Name	Description
class AutoCmd (see page 22375)	This is friend class AutoCmd.

158.1.30.6.1 friend class AutoCmd Friend

Pascal

```
class AutoCmd
```

C++

```
friend class AutoCmd;
```

Description

This is friend class AutoCmd.

158.1.31 VariantError Class

Class Hierarchy

System::VariantError

Pascal

```
VariantError = class;
```

C++

```
class VariantError;
```

File

sysvari.h

Description

This is class System::VariantError.

158.1.31.1 VariantError Members

The following tables list the members exposed by VariantError.

Protected Data Members

	Name	Description
	m_scode (see page 22376)	This is m_scode, a member of class VariantError.

Public Methods

	Name	Description
	VariantError (see page 22376)	This is VariantError, a member of class VariantError.

Public Operators

	Name	Description
	= (see page 22377)	This is =, a member of class VariantError.
	SCODE (see page 22377)	This is SCODE, a member of class VariantError.

158.1.31.2 VariantError::VariantError Constructor

Pascal

```
constructor Create(err: SCODE = DISP_E_PARAMNOTFOUND);
```

C++

```
VariantError(SCODE err = DISP_E_PARAMNOTFOUND);
```

Description

This is VariantError, a member of class VariantError.

158.1.31.3 VariantError Data Members

The data members of the VariantError class are listed here.

Protected Data Members

	Name	Description
	m_scode (see page 22376)	This is m_scode, a member of class VariantError.

158.1.31.3.1 VariantError::m_scode Data Member

Pascal

```
m_scode: SCODE;
```

C++

```
SCODE m_scode;
```

Description

This is m_scode, a member of class VariantError.

158.1.31.4 VariantError Operators

The operators of the VariantError class are listed here.

Public Operators

	Name	Description
	= (see page 22377)	This is =, a member of class VariantError.
	SCODE (see page 22377)	This is SCODE, a member of class VariantError.

158.1.31.4.1 VariantError::= Operator

Pascal

```
operator =(err: SCODE);
```

C++

```
VariantError& operator =(SCODE err);
```

Description

This is =, a member of class VariantError.

158.1.31.4.2 VariantError::SCODE Operator

Pascal

```
operator SCODE;
```

C++

```
operator SCODE() const;
```

Description

This is SCODE, a member of class VariantError.

158.1.32 WideString Class

System::WideString represents strings of 16-bit characters.

Class Hierarchy

System::WideString

Pascal

```
WideString = class;
```

C++

```
class WideString;
```

File

wstring.h

Description

System::WideString represents strings of 16-bit characters. Each character is stored as a wchar_t value. The C++ class System::WideString is functionally equivalent to a Delphi intrinsic type, also called System::WideString.

Note: Although System::WideString is a platform-independent data (see page 22383) type and often appears in cross-platform code, System::WideString is also useful for working with Microsoft's Component Object Model (COM) System::WideString values are easily converted to or from the BSTR (see page 22393) values used in COM.

158.1.32.1 WideString Members

The following tables list the members exposed by WideString.

Public Methods

	Name	Description
	~WideString (see page 22378)	Frees an instance ofSystem::WideString::WideString (see page 22377).
	Attach (see page 22381)	Uses an existing BSTR (see page 22393) for the underlying implementation.
	c_bstr (see page 22382)	Returns the underlying BSTR (see page 22393) that implements theSystem::WideString::WideString (see page 22377).
	Copy (see page 22382)	This is the overview for the Copy method overload.
	data (see page 22383)	This is data, a member of class WideString.
	Delete (see page 22383)	System::WideString::Deletes a substring from the System::WideString::WideString (see page 22380).
	Detach (see page 22383)	System::WideString::Detaches theSystem::WideString::WideString (see page 22377) from a BSTR (see page 22393) that was previously attached using the Attach (see page 22381) method.
	Empty (see page 22384)	Sets the value of theSystem::WideString::WideString (see page 22377) to an empty string.
	Insert (see page 22384)	System::WideString::Inserts a specified string into theSystem::WideString::WideString (see page 22377).
	IsEmpty (see page 22385)	Returns true if the System::WideString::WideString (see page 22380) is empty.
	Length (see page 22385)	Returns the length of theSystem::WideString::WideString (see page 22377).
	Pos (see page 22385)	Returns the index at which a specified substring begins.
	SetLength (see page 22385)	Changes the length of theSystem::WideString::WideString (see page 22377).
	SubString (see page 22386)	Returns a specified substring of theSystem::WideString::WideString (see page 22377).
	WideString (see page 22379)	This is the overview for the WideString constructor overload.

Public Operators

	Name	Description
	!= (see page 22387)	This is the overview for the != operator overload.
	& (see page 22387)	Returns a pointer to the underlying BSTR (see page 22393).
	[] (see page 22388)	This is the overview for the [] operator overload.
	+ (see page 22388)	Concatenates twoSystem::WideString::WideStrings.
	+= (see page 22389)	Concatenates this System::WideString::WideString (see page 22380) and rhs, and assigns the result to this System::WideString::WideString (see page 22380).
	< (see page 22389)	This is the overview for the < operator overload.
	<= (see page 22390)	This is the overview for the <= operator overload.
	= (see page 22390)	This is the overview for the = operator overload.
	== (see page 22391)	This is the overview for the == operator overload.
	> (see page 22392)	This is the overview for the > operator overload.
	>= (see page 22392)	This is the overview for the >= operator overload.
	BSTR (see page 22393)	Converts the wide string to a System::WideString::BSTR.

158.1.32.2 WideString::~WideString Destructor

Frees an instance ofSystem::WideString::WideString (see page 22377).

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~WideString();
```

Description

System::WideString (see page 22377)::~WideString (see page 22377) is the destructor forSystem::WideString::WideString (see page 22377).

158.1.32.3 WideString Constructor

This is the overview for the WideString constructor overload.

Overload List

	Name	Description
	WideString::WideString () (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (__int64) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (AnsiString&) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (char) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (char*) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (double) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (float) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (int) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (long) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (long double) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (short) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (unsigned __int64) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (unsigned int) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (unsigned long) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (unsigned short) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (wchar_t) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (wchar_t*) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (wchar_t*, int) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (WideChar) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (WideChar*) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).

	WideString::WideString (WideChar*, int) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).
	WideString::WideString (WideString&) (see page 22380)	Creates an instance of System::WideString::WideString (see page 22377).

158.1.32.3.1 WideString::WideString Constructor ()

Creates an instance of System::WideString::WideString (see page 22377).

Pascal

```
constructor Create;
constructor Create(const src: char*);
constructor Create(const src: WideString&);
constructor Create(const src: AnsiString&);
constructor Create(const src: WideChar*; len: Integer);
constructor Create(const src: WideChar*);
constructor Create(const src: WideChar);
constructor Create(const src: wchar_t*; len: Integer);
constructor Create(const src: wchar_t*);
constructor Create(const src: wchar_t);
constructor Create(src: char);
constructor Create(src: short);
constructor Create(: unsigned short);
constructor Create(src: Integer);
constructor Create(: unsigned int);
constructor Create(: long);
constructor Create(: unsigned long);
constructor Create(: __int64);
constructor Create(: unsigned __int64);
constructor Create(src: float);
constructor Create(src: double);
constructor Create(src: long double);
```

C++

```
__fastcall WideString();
__fastcall WideString(const char* src);
__fastcall WideString(const WideString& src);
__fastcall WideString(const AnsiString& src);
__fastcall WideString(const WideChar* src, int len);
__fastcall WideString(const WideChar* src);
__fastcall explicit WideString(const WideChar src);
__fastcall WideString(const wchar_t* src, int len);
__fastcall WideString(const wchar_t* src);
__fastcall explicit WideString(const wchar_t src);
__fastcall explicit WideString(char src);
__fastcall explicit WideString(short src);
__fastcall explicit WideString(unsigned short);
__fastcall explicit WideString(int src);
__fastcall explicit WideString(unsigned int);
__fastcall explicit WideString(long);
__fastcall explicit WideString(unsigned long);
__fastcall explicit WideString(__int64);
__fastcall explicit WideString(unsigned __int64);
__fastcall explicit WideString(float src);
__fastcall explicit WideString(double src);
__fastcall explicit WideString(long double src);
```

Description

Call System::WideString::WideString (see page 22377) to create and initialize a wide string. If the constructor is given no arguments, the resulting System::WideString::WideString (see page 22377) is an empty string. If the constructor is passed a

single wide character, the resulting System::WideString::WideString (see page 22377) has length one, and contains that character. All other versions of the constructor copy the string passed as an argument, converting characters to wchar_t if necessary. When constructing a System::WideString::WideString (see page 22377) from an array of char or wchar_t, the src string should be null terminated, unless a second argument is given that indicates the number of characters in the array.

158.1.32.4 WideString Methods

The methods of the WideString class are listed here.

Public Methods

	Name	Description
	Attach (see page 22381)	Uses an existing BSTR (see page 22393) for the underlying implementation.
	c_bstr (see page 22382)	Returns the underlying BSTR (see page 22393) that implements theSystem::WideString::WideString (see page 22377).
	Copy (see page 22382)	This is the overview for the Copy method overload.
	data (see page 22383)	This is data, a member of class WideString.
	Delete (see page 22383)	System::WideString::Deletes a substring from the System::WideString::WideString (see page 22380).
	Detach (see page 22383)	System::WideString::Detaches theSystem::WideString::WideString (see page 22377) from a BSTR (see page 22393) that was previously attached using the Attach (see page 22381) method.
	Empty (see page 22384)	Sets the value of theSystem::WideString::WideString (see page 22377) to an empty string.
	Insert (see page 22384)	System::WideString::Inserts a specified string into theSystem::WideString::WideString (see page 22377).
	IsEmpty (see page 22385)	Returns true if the System::WideString::WideString (see page 22380) is empty.
	Length (see page 22385)	Returns the length of theSystem::WideString::WideString (see page 22377).
	Pos (see page 22385)	Returns the index at which a specified substring begins.
	SetLength (see page 22385)	Changes the length of theSystem::WideString::WideString (see page 22377).
	SubString (see page 22386)	Returns a specified substring of theSystem::WideString::WideString (see page 22377).

158.1.32.4.1 WideString::Attach Method

Uses an existing BSTR (see page 22393) for the underlying implementation.

Pascal

```
procedure Attach(src: WideString);
```

C++

```
__fastcall void Attach(BSTR src);
```

Description

System::WideString::Attach uses the BSTR (see page 22393) passed as an argument as the value of theSystem::WideString::WideString (see page 22377). While the source BSTR (see page 22393) is attached, you can access the memory for the BSTR (see page 22393) using either theSystem::WideString::WideString (see page 22377) or the source BSTR (see page 22393).

To subsequently detach from src, use the Detach (see page 22383) method.

Warning: Do not delete the source BSTR (see page 22393) while it is attached to theSystem::WideString::WideString (see page 22377). This will leave theSystem::WideString::WideString (see page 22377) with a dangling reference.

Note: BSTR (see page 22393) values are used with COM objects. Do not use COM objects in platform-independent

applications, since COM is only available under Windows.

See Also

System::WideString::Detach (see page 22383)

System::WideString::c_bstr (see page 22382)

158.1.32.4.2 WideString::c_bstr Method

Returns the underlying BSTR (see page 22393) that implements theSystem::WideString::WideString (see page 22377).

Pascal

```
function c_bstr: WideChar*;
```

C++

```
__fastcall WideChar* c_bstr() const;
```

Description

Use c_System::WideString::c_bstr to access the BSTR (see page 22393) that implements theSystem::WideString::WideString (see page 22377).

Note: BSTR (see page 22393) values are used with COM objects. Do not use COM objects in platform-independent applications, since COM is only available under Window.

See Also

System::WideString::c_bstr

System::WideString::Copy (see page 22382)

System::WideString::& (see page 22387)

158.1.32.4.3 Copy Method

This is the overview for the Copy method overload.

Overload List

	Name	Description
	WideString::Copy () (see page 22382)	Returns a copy of theSystem::WideString::WideString (see page 22377).
	WideString::Copy (wchar_t *, int) (see page 22382)	Returns a copy of theSystem::WideString::WideString (see page 22377).
	WideString::Copy (wchar_t*) (see page 22382)	Returns a copy of theSystem::WideString::WideString (see page 22377).

158.1.32.4.3.1 WideString::Copy Method ()

Returns a copy of theSystem::WideString::WideString (see page 22377).

Pascal

```
function Copy(var src: wchar_t): wchar_t*; static;
function Copy: wchar_t*;
function Copy(var : wchar_t; len: Integer): wchar_t*;
```

C++

```
static __fastcall wchar_t* Copy(wchar_t* src);
__fastcall wchar_t* Copy() const;
__fastcall wchar_t* Copy(wchar_t *, int len) const;
```

Description

Use System::WideString::Copy to copy theSystem::WideString::WideString (see page 22377) into an array of wchar_t. Use the Length (see page 22385) method to ensure that the destination buffer is large enough to hold the contents of theSystem::WideString::WideString (see page 22377).

WideString (see page 22377) also supports a static copy method that copies an arbitrary source string.

See Also

System::WideString::bstr

System::WideString::c_bstr (see page 22382)

System::WideString::& (see page 22387)

System::WideString::Length (see page 22385)

158.1.32.4.4 WideString::data Method

Pascal

```
function data: WideChar*;
```

C++

```
__fastcall WideChar* data();
```

Description

This is data, a member of class WideString.

158.1.32.4.5 WideString::Delete Method

System::WideString::Deletes a substring from the System::WideString::WideString (see page 22380).

Pascal

```
procedure Delete(index: Integer; count: Integer);
```

C++

```
__fastcall void Delete(int index, int count);
```

Description

System::WideString::Delete removes count characters from the System::WideString::WideString (see page 22380), beginning with the character at position index.

See Also

System::WideString::Insert (see page 22384)

System::WideString::[] (see page 22388)

158.1.32.4.6 WideString::Detach Method

System::WideString::Detaches theSystem::WideString::WideString (see page 22377) from a BSTR (see page 22393) that was previously attached using the Attach (see page 22381) method.

Pascal

```
function Detach: WideString;
```

C++

```
__fastcall BSTR Detach();
```

Description

Call System::WideString::Detach to undo the association set up by a call to the Attach (see page 22381) method. System::WideString::Detach returns the BSTR (see page 22393) that is no longer associated with theSystem::WideString::WideString (see page 22377).

Note: BSTR (see page 22393) values are used with COM objects. Do not use COM objects in platform-independent applications, since COM is only available under Windows.

See Also

System::WideString::Attach (see page 22381)

158.1.32.4.7 WideString::Empty Method

Sets the value of theSystem::WideString::WideString (see page 22377) to an empty string.

Pascal

```
procedure Empty;
```

C++

```
__fastcall void Empty();
```

Description

Call System::WideString::Empty to replace the contents of theSystem::WideString::WideString (see page 22377) with an empty string.

See Also

System::WideString::IsEmpty (see page 22385)

158.1.32.4.8 WideString::Insert Method

System::WideString::Inserts a specified string into theSystem::WideString::WideString (see page 22377).

Pascal

```
procedure Insert(const str: WideString&; index: Integer);
```

C++

```
__fastcall void Insert(const WideString& str, int index);
```

Description

Call System::WideString::Insert to inserts the string str into thisSystem::WideString::WideString (see page 22377) beginning at the position index.

See Also

System::WideString::Delete (see page 22383)

System::WideString::[] (see page 22388)

158.1.32.4.9 WideString::IsEmpty Method

Returns true if the System::WideString::WideString (see page 22380) is empty.

Pascal

```
function IsEmpty: bool;
```

C++

```
__fastcall bool IsEmpty() const;
```

Description

Call System::WideString::IsEmpty to determine whether the System::WideString::WideString (see page 22380) represents an empty string.

See Also

System::WideString::Empty (see page 22384)

158.1.32.4.10 WideString::Length Method

Returns the length of theSystem::WideString::WideString (see page 22377).

Pascal

```
function Length: Integer;
```

C++

```
__fastcall int Length() const;
```

Description

System::WideString::Length returns the number of characters in thisSystem::WideString::WideString (see page 22377).

158.1.32.4.11 WideString::Pos Method

Returns the index at which a specified substring begins.

Pascal

```
function Pos(const subStr: WideString&): Integer;
```

C++

```
__fastcall int Pos(const WideString& subStr) const;
```

Description

Use System::WideString::Pos to locate a substring of theSystem::WideString::WideString (see page 22377). System::WideString::Pos returns the index of the first character in subStr. If subStr is not contained in theSystem::WideString::WideString (see page 22377), System::WideString::Pos returns 0.

See Also

System::WideString::Length (see page 22385)

System::WideString::SubString (see page 22386)

158.1.32.4.12 WideString::SetLength Method

Changes the length of theSystem::WideString::WideString (see page 22377).

Pascal

```
procedure SetLength(newLength: Integer);
```

C++

```
__fastcall void SetLength(int newLength);
```

Description

System::WideString::SetLength changes theSystem::WideString::WideString (see page 22377)'s length to newLength, truncating if necessary.

System::WideString::SetLength reallocates the underlying array of characters that stores the value of theSystem::WideString::WideString (see page 22377). Existing characters in the string are preserved, but the content of newly allocated space is undefined. If there is not enough memory available to reallocate theSystem::WideString::WideString (see page 22377), an EOutOfMemory exception is thrown. Following a call to System::WideString::SetLength, theSystem::WideString::WideString (see page 22377) is guaranteed to reference a unique string — that is a string with a reference count of one.

See Also

System::WideString::Length (see page 22385)

158.1.32.4.13 WideString::SubString Method

Returns a specified substring of theSystem::WideString::WideString (see page 22377).

Pascal

```
function SubString(index: Integer; count: Integer): WideString;
```

C++

```
__fastcall WideString SubString(int index, int count) const;
```

Description

System::WideString::SubString returns a newSystem::WideString::WideString (see page 22377) that is a substring of thisSystem::WideString::WideString (see page 22377). The substring contains count characters beginning at index.

See Also

System::WideString::Length (see page 22385)

System::WideString::pos

158.1.32.5 WideString Operators

The operators of the WideString class are listed here.

Public Operators

	Name	Description
	!= (see page 22387)	This is the overview for the != operator overload.
	& (see page 22387)	Returns a pointer to the underlying BSTR (see page 22393).
	[] (see page 22388)	This is the overview for the [] operator overload.
	+ (see page 22388)	Concatenates twoSystem::WideString::WideStrings.
	+= (see page 22389)	Concatenates this System::WideString::WideString (see page 22380) and rhs, and assigns the result to this System::WideString::WideString (see page 22380).
	< (see page 22389)	This is the overview for the < operator overload.

	<= (see page 22390)	This is the overview for the <= operator overload.
	= (see page 22390)	This is the overview for the = operator overload.
	== (see page 22391)	This is the overview for the == operator overload.
	> (see page 22392)	This is the overview for the > operator overload.
	>= (see page 22392)	This is the overview for the >= operator overload.
	BSTR (see page 22393)	Converts the wide string to a System::WideString::BSTR.

158.1.32.5.1 != Operator

This is the overview for the != operator overload.

Overload List

	Name	Description
	WideString::!= (BSTR) (see page 22387)	Tests whether theSystem::WideString::WideString (see page 22377) is different from anotherSystem::WideString::WideString (see page 22377).
	WideString::!= (char*) (see page 22387)	Tests whether theSystem::WideString::WideString (see page 22377) is different from anotherSystem::WideString::WideString (see page 22377).
	WideString::!= (WideString&) (see page 22387)	Tests whether theSystem::WideString::WideString (see page 22377) is different from anotherSystem::WideString::WideString (see page 22377).

158.1.32.5.1.1 WideString::!= Operator (BSTR)

Tests whether theSystem::WideString::WideString (see page 22377) is different from anotherSystem::WideString::WideString (see page 22377).

Pascal

```
operator !=(const rhs: WideString&);
operator !=(const w: WideString);
operator !=(const s: char*);
```

C++

```
__fastcall bool operator !=(const WideString& rhs) const;
__fastcall bool operator !=(const BSTR w) const;
__fastcall bool operator !=(const char* s) const;
```

Description

The != System::WideString (see page 22377)::!= returns true if the strings are not the same, false otherwise.

See Also

System::WideString::== (see page 22391)

158.1.32.5.2 WideString::& Operator

Returns a pointer to the underlying BSTR (see page 22393).

Pascal

```
operator &;
```

C++

```
__fastcall BSTR* operator &();
```

Description

Use the & System::WideString (see page 22377)::& to obtain a pointer to the internal BSTR (see page 22393) that is used to implement theSystem::WideString::WideString (see page 22377).

Note: BSTR (see page 22393) values are used with COM objects. Do not use COM objects in platform-independent applications, since COM is only available under Window.

See Also

System::WideString::bstr

System::WideString::Copy (see page 22382)

System::WideString::c_bstr (see page 22382)

158.1.32.5.3 [] Operator

This is the overview for the [] operator overload.

Overload List

	Name	Description
	WideString::[] (int) (see page 22388)	Returns the character at a specified index in the string.
	WideString::[] (int) (see page 22388)	Returns the character at a specified index in the string.

158.1.32.5.3.1 WideString::[] Operator (int)

Returns the character at a specified index in the string.

Pascal

```
operator [](const idx: Integer);
operator [](const idx: Integer);
```

C++

```
__fastcall WideChar & operator [](const int idx);
__fastcall const WideChar& operator [](const int idx) const;
```

Description

The System::WideString (see page 22377)::[] returns the character in the string at index value idx. The System::WideString (see page 22377)::[] assumes a base index of 1. For example, given the string “Hello” and an index value of 2, the character returned would be ‘e’.

158.1.32.5.4 WideString::+ Operator

Concatenates twoSystem::WideString::WideStrings.

Pascal

```
operator +(const rhs: WideString&);
```

C++

```
__fastcall WideString operator +(const WideString& rhs) const;
```

Description

The + System::WideString (see page 22377)::+ returns aSystem::WideString::WideString (see page 22377) that has the value passed as rhs concatenated to the end. A related friend System::WideString (see page 22377)::+ takes two arguments

and returns the concatenation of lhs with rhs.

158.1.32.5.5 WideString::+= Operator

Concatenates this System::WideString::WideString (see page 22380) and rhs, and assigns the result to this System::WideString::WideString (see page 22380).

Pascal

```
operator +=(const rhs: WideString&);
```

C++

```
__fastcall WideString& operator +=(const WideString& rhs);
```

Description

Use += to concatenate rhs to the end of the System::WideString::WideString (see page 22380). This System::WideString (see page 22377)::+= is not supported when the System::WideString::WideString (see page 22380) is a property value.

158.1.32.5.6 < Operator

This is the overview for the < operator overload.

Overload List

	Name	Description
	WideString::< (BSTR) (see page 22389)	Compares two strings.
	WideString::< (char*) (see page 22389)	Compares two strings.
	WideString::< (WideString&) (see page 22389)	Compares two strings.

158.1.32.5.6.1 WideString::< Operator (BSTR)

Compares two strings.

Pascal

```
operator <(const rhs: WideString&);
operator <(const w: WideString);
operator <(const s: char*);
```

C++

```
__fastcall bool operator <(const WideString& rhs) const;
__fastcall bool operator <(const BSTR w) const;
__fastcall bool operator <(const char* s) const;
```

Description

The < System::WideString (see page 22377)::< returns true if thisSystem::WideString::WideString (see page 22377) is less than rhs, false otherwise. The strings are compared using the numerical value of each character, with case sensitivity.

See Also

System::WideString::<= (see page 22390)

System::WideString::> (see page 22392)

158.1.32.5.7 <= Operator

This is the overview for the <= operator overload.

Overload List

	Name	Description
	WideString::<= (BSTR) (see page 22390)	Compares two strings.
	WideString::<= (char*) (see page 22390)	Compares two strings.
	WideString::<= (WideString&) (see page 22390)	Compares two strings.

158.1.32.5.7.1 WideString::<= Operator (BSTR)

Compares two strings.

Pascal

```
operator <=(const rhs: WideString&);
operator <=(const w: WideString);
operator <=(const s: char*);
```

C++

```
__fastcall bool operator <=(const WideString& rhs) const;
__fastcall bool operator <=(const BSTR w) const;
__fastcall bool operator <=(const char* s) const;
```

Description

The <= System::WideString (see page 22377)::<= returns true if thisSystem::WideString::WideString (see page 22377) is less than or equal to rhs, false otherwise. The strings are compared using their numerical value of each character, with case sensitivity.

See Also

System::WideString::>= (see page 22392)

System::WideString::< (see page 22389)

158.1.32.5.8 = Operator

This is the overview for the = operator overload.

Overload List

	Name	Description
	WideString::= (BSTR) (see page 22390)	Assigns the value of theSystem::WideString::WideString (see page 22377) to equal a source string.
	WideString::= (WideString&) (see page 22390)	Assigns the value of theSystem::WideString::WideString (see page 22377) to equal a source string.

158.1.32.5.8.1 WideString::= Operator (BSTR)

Assigns the value of theSystem::WideString::WideString (see page 22377) to equal a source string.

Pascal

```
operator =(const rhs: WideString&);
operator =(rhs: WideString);
```

C++

```
__fastcall WideString& operator =(const WideString& rhs);
__fastcall WideString& operator =(BSTR rhs);
```

Description

Use the assignment System::WideString (see page 22377)::= to set the value of theSystem::WideString::WideString (see page 22377) to match the string given by rhs.

Note: BSTR (see page 22393) values are used with COM objects. Do not use COM objects in platform-independent applications, since COM is only available under Windows.

158.1.32.5.9 == Operator

This is the overview for the == operator overload.

Overload List

	Name	Description
	WideString::== (BSTR) (see page 22391)	Compares theSystem::WideString::WideString (see page 22377) with a specified string, returning true if they are the same.
	WideString::== (char*) (see page 22391)	Compares theSystem::WideString::WideString (see page 22377) with a specified string, returning true if they are the same.
	WideString::== (WideString&) (see page 22391)	Compares theSystem::WideString::WideString (see page 22377) with a specified string, returning true if they are the same.

158.1.32.5.9.1 WideString::== Operator (BSTR)

Compares theSystem::WideString::WideString (see page 22377) with a specified string, returning true if they are the same.

Pascal

```
operator ==(const rhs: WideString&);
operator ==(const w: WideString);
operator ==(const s: char*);
```

C++

```
__fastcall bool operator ==(const WideString& rhs) const;
__fastcall bool operator ==(const BSTR w) const;
__fastcall bool operator ==(const char* s) const;
```

Description

The equality System::WideString (see page 22377)::== returns true if thisSystem::WideString::WideString (see page 22377) is the same as rhs, false otherwise.

See Also

System::WideString::!= (see page 22387)

158.1.32.5.10 > Operator

This is the overview for the > operator overload.

Overload List

	Name	Description
	WideString::> (BSTR) (see page 22392)	Compares two strings.
	WideString::> (char*) (see page 22392)	Compares two strings.
	WideString::> (WideString&) (see page 22392)	Compares two strings.

158.1.32.5.10.1 WideString::> Operator (BSTR)

Compares two strings.

Pascal

```
operator >(const rhs: WideString&);
operator >(const w: WideString);
operator >(const s: char*);
```

C++

```
__fastcall bool operator >(const WideString& rhs) const;
__fastcall bool operator >(const BSTR w) const;
__fastcall bool operator >(const char* s) const;
```

Description

The System::WideString (see page 22377)::> returns true if thisSystem::WideString::WideString (see page 22377) is greater than rhs, false otherwise. The strings are compared using the numerical value of each character, with case sensitivity.

See Also

System::WideString::>= (see page 22392)

System::WideString::< (see page 22389)

158.1.32.5.11 >= Operator

This is the overview for the >= operator overload.

Overload List

	Name	Description
	WideString::>= (BSTR) (see page 22392)	Compares two strings.
	WideString::>= (char*) (see page 22392)	Compares two strings.
	WideString::>= (WideString&) (see page 22392)	Compares two strings.

158.1.32.5.11.1 WideString::>= Operator (BSTR)

Compares two strings.

Pascal

```
operator >=(const rhs: WideString&);
operator >=(const w: WideString);
```

```
operator >=(const s: char*);
```

C++

```
__fastcall bool operator >=(const WideString& rhs) const;
__fastcall bool operator >=(const BSTR w) const;
__fastcall bool operator >=(const char* s) const;
```

Description

The System::WideString (see page 22377)::>= returns true if thisSystem::WideString::WideString (see page 22377) is greater than or equal to rhs, false otherwise. The strings are compared using the numerical value of each character, with case sensitivity.

See Also

System::WideString::<= (see page 22390)

System::WideString::> (see page 22392)

158.1.32.5.12 WideString::BSTR Operator

Converts the wide string to a System::WideString::BSTR.

Pascal

```
operator BSTR;
```

C++

```
operator BSTR() const;
```

Description

Use the System::WideString::BSTR conversion operator to cast aSystem::WideString::WideString (see page 22377) value to a System::WideString::BSTR.

Note: System::WideString::BSTR values are used with COM objects. Do not use COM objects in platform-independent applications, since COM is only available under Window.

See Also

System::WideString::c_bstr (see page 22382)

System::WideString::Copy (see page 22382)

System::WideString::& (see page 22387)

158.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IDispatch (see page 22394)	This is class System.IDispatch.
	IInterface (see page 22396)	IInterface is the base class for all interfaces defined in Delphi code.
	IInvokable (see page 22398)	IInvokable is the base class for invokable interfaces.

158.2.1 IDispatch Interface

Class Hierarchy

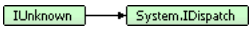

Pascal

```
IDispatch = interface(IUnknown);
```

C++

```
__interface IDispatch : public IUnknown;
```

File

System

Description

This is class System.IDispatch.

158.2.1.1 IDispatch Members

The following tables list the members exposed by IDispatch.

Public Methods

	Name	Description
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.

158.2.1.2 IDispatch Methods

The methods of the IDispatch class are listed here.

Public Methods

	Name	Description
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.

	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.

158.2.1.2.1 IDispatch.GetIDsOfNames Method

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int NameCount, int
LocaleID, void * DispIDs);
```

Description

This is GetIDsOfNames, a member of class IDispatch.

158.2.1.2.2 IDispatch.GetTypeInfo Method

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

This is GetTypeInfo, a member of class IDispatch.

158.2.1.2.3 IDispatch.GetTypeInfoCount Method

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

This is GetTypeInfoCount, a member of class IDispatch.

158.2.1.2.4 IDispatch.Invoke Method

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word Flags,
Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is Invoke, a member of class IDispatch.

158.2.2 IInterface Interface

IInterface is the base class for all interfaces defined in Delphi code.

Class Hierarchy

System.IInterface

Pascal

```
IInterface = interface;
```

C++

```
__interface IInterface;
```

File

System

Description

IInterface is the base interface for interfaces defined in Delphi code. When used in C++ code, IInterface is considered a child interface of IUnknown (see page 22586) and IUnknown (see page 22586) is considered the base interface for all interfaces.

Note: IInterface introduces the QueryInterface (see page 22398) method, which is useful for discovering and using other interfaces implemented by the same object. IInterface also introduces the reference counting methods, _AddRef (see page 22397) and _Release (see page 22397). The Delphi compiler automatically provides calls to these methods when interfaces are used. The System::TInterfacedObject (see page 22276) class is provided as a base class for objects that implement interfaces. Use of System::TInterfacedObject (see page 22276) is not mandatory, but will often be easier than designing an implementer class from scratch.

Note: C++ method declarations that use IInterface refer to the _di_IInterface type instead. This is a DelphiInterface (see page 22167) wrapper around the IInterface interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IInterface > _di_IInterface;
```

See Also

System::TInterfacedObject (see page 22276)

IInvokable (see page 22398)

IUnknown (see page 22586)

System::DelphiInterface (see page 22167)

158.2.2.1 IInterface Members

The following tables list the members exposed by IInterface.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

158.2.2.2 IInterface Methods

The methods of the IInterface class are listed here.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

158.2.2.2.1 IInterface._AddRef Method

Increments the reference count for this interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

The Delphi compiler automatically generates a call to __AddRef for every reference to this interface. For most implementations, __AddRef increments the reference count for the interface and returns the new reference count. When the caller is through with the interface, the compiler generates a call to the _Release (see page 22397) method, which decrements the reference count. When the reference count drops to zero, the object is automatically freed.

Note: Implementation objects can implement __AddRef and _Release (see page 22397) so that they do not perform any reference counting. For such objects, the reference count never drops to zero and the object is not automatically freed. In these cases, it is the application's responsibility to free the object.

See Also

_Release (see page 22397)

158.2.2.2.2 IInterface._Release Method

Decrements the reference count for this interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

The Delphi compiler automatically generates a call to __Release whenever a reference to this interface goes out of scope. For most implementations, __Release decrements the reference count for the interface and returns the new reference count. When the reference count drops to zero, the object is automatically freed.

Note: Implementation objects can implement __Release so that it does not perform any reference counting. For such objects, the reference count never drops to zero and the object is not automatically freed. In these cases, it is the application's responsibility to free the object.

See Also

_AddRef (see page 22397)

158.2.2.2.3 IInterface.QueryInterface Method

Returns a reference to a specified interface if the object supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the object that implements this interface supports the interface specified by IID. If so, QueryInterface

increments the reference count.

sets the Obj parameter so that it points to an instance of the specified interface.

returns 0 to indicate success.

If the object does not support the interface, QueryInterface returns a nonzero error code such as E_NoInterface.

See Also

_AddRef (see page 22397)

158.2.3 IInvokable Interface

IInvokable is the base class for invokable interfaces.

Class Hierarchy

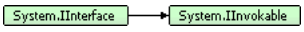

Pascal

```
IInvokable = interface(IInterface);
```

C++

```
__interface IInvokable : public IInterface;
```

File

System

Description

Use IInvokable as a base interface when defining invokable interfaces. IInvokable does not add any properties or methods to the methods it inherits from IInterface (see page 22396). It exists, rather, to ensure that the interface an all its descendants are compiled with runtime type information (RTTI). This RTTI can be used by Web Service applications or the clients of Web Service applications for translating between the code that implements a method and the symbolic encoding of that method call that is sent between client and server applications.

See Also

InvRegistry

TInvokableClassRegistry (see page 15526)

TInvokableClass (see page 15520)

Understanding Invokable Interfaces

158.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	!= (see page 22405)	This is function System::!=.
	__currMul (see page 22405)	This is function System::__currMul.
	_CVR_PROBE (see page 22405)	This is function System._CVR_PROBE.
	_CVR_STMTPROBE (see page 22406)	This is function System._CVR_STMTPROBE.
	_roundToInt64 (see page 22406)	This is function System::_roundToInt64.
	+ (see page 22406)	This is function System::+.
	< (see page 22407)	This is function System::<.
	<< (see page 22407)	This is function System::<<.
	<< (see page 22407)	This is function System::<<.
	<< (see page 22408)	This is function System::<<.
	<= (see page 22408)	This is function System::<=.
	== (see page 22408)	This is function System::==.
	> (see page 22409)	This is function System::>.
	>= (see page 22409)	This is function System::>=.
	>> (see page 22409)	This is function System::>>.
	>> (see page 22410)	This is function System::>>.
	Abs (see page 22410)	Returns an absolute value.
	AcquireExceptionObject (see page 22410)	Retains access to exception object.
	AddModuleUnloadProc (see page 22411)	AddModuleUnloadProc adds a call to your defined routine when a package is unloaded.
	AddModuleUnloadProc (see page 22411)	AddModuleUnloadProc adds a call to your defined routine when a package is unloaded.
	Addr (see page 22412)	Returns a pointer to a specified object.
	AllocMem (see page 22413)	AllocMem allocates a memory block and initializes each byte to zero.
	AnsiToUtf8 (see page 22415)	Converts a string encoded in Ansi to UTF-8.
	Append (see page 22415)	Prepares an existing file for adding text to the end.
	ArcTan (see page 22416)	Calculates the arctangent of a given number.
	Assert (see page 22417)	Tests whether a Boolean expression is true.
	Assert (see page 22417)	Tests whether a Boolean expression is true.
	Assigned (see page 22419)	Tests for a nil (unassigned) pointer or procedural variable.
	AssignFile (see page 22420)	Associates the name of an external file with a file variable.
	AttemptToUseSharedMemoryManager (see page 22422)	Attempts to use a shared memory manager.
	BeginThread (see page 22422)	Spawns a separate thread of execution.
	BlockRead (see page 22423)	Reads one or more records from an open file into a variable.
	BlockRead (see page 22423)	Reads one or more records from an open file into a variable.
	BlockWrite (see page 22425)	Writes one or more records from a variable to an open file.
	BlockWrite (see page 22425)	Writes one or more records from a variable to an open file.
	Break (see page 22426)	Causes the flow of control to exit a for, while, or repeat statement.
	ChDir (see page 22429)	Changes the current directory.
	ChDir (see page 22429)	Changes the current directory.
	Chr (see page 22430)	Returns the character for a specified ASCII value.
	Close (see page 22432)	Terminates the association between a file variable and an external file.
	CloseFile (see page 22433)	Terminates the association between file variable and an external disk file.
	CompToCurrency (see page 22435)	Converts a Comp value to a Currency (see page 22145) value.
	CompToDouble (see page 22435)	Converts a Comp value to a double value.

	Concat (see page 22436)	Concatenates two or more strings into one.
	Concat (see page 22436)	Concatenates two or more strings into one.
	Concat (see page 22436)	Concatenates two or more strings into one.
	Continue (see page 22437)	Allows the flow of control to proceed to the next iteration of for, while, or repeat statements.
	Copy (see page 22438)	Returns a substring of a string or a segment of a dynamic array.
	Cos (see page 22440)	Calculates the cosine of an angle.
	CurrencyToComp (see page 22441)	Converts a Currency (see page 22145) value to a Comp value.
	Dec (see page 22441)	Decrements a variable by 1 or N.
	Dec (see page 22441)	Decrements a variable by 1 or N.
	Delete (see page 22443)	Removes a substring from a string.
	Dispose (see page 22444)	Releases memory allocated for a dynamic variable.
	DoubleToComp (see page 22446)	Converts a double value to a Comp.
	DynArrayClear (see page 22446)	This is function System.DynArrayClear.
	DynArrayDim (see page 22446)	This is function System.DynArrayDim.
	DynArraySetLength (see page 22447)	This is function System.DynArraySetLength.
	EndThread (see page 22447)	Terminates the execution of a thread.
	EnumModules (see page 22447)	Executes a callback for all the modules in an application.
	EnumModules (see page 22447)	Executes a callback for all the modules in an application.
	EnumResourceModules (see page 22448)	Executes a callback for all resource modules in an application.
	EnumResourceModules (see page 22448)	Executes a callback for all resource modules in an application.
	Eof (see page 22449)	Tests whether the file position is at the end of a file.
	Eof (see page 22449)	Tests whether the file position is at the end of a file.
	Eof (see page 22449)	Tests whether the file position is at the end of a file.
	Eoln (see page 22450)	Tests whether the file pointer is at the end of a line.
	Eoln (see page 22450)	Tests whether the file pointer is at the end of a line.
	Erase (see page 22451)	Deletes an external file.
	Error (see page 22451)	Error is used to generate a run-time exception.
	ExceptAddr (see page 22452)	Returns the address at which the current exception was raised.
	ExceptObject (see page 22452)	Returns current exception object.
	Exclude (see page 22453)	Removes an element from a Delphi set or removes an event handler in .NET applications.
	Exit (see page 22453)	Exits from the current procedure.
	Exp (see page 22455)	Returns the exponential of X.
	Exp (see page 22455)	Returns the exponential of X.
	FilePos (see page 22456)	Returns the current file position.
	FileSize (see page 22457)	Returns the number of records in a file.
	FillChar (see page 22459)	Fills contiguous bytes with a specified value.
	Finalize (see page 22460)	Uninitializes a dynamically allocated variable.
	Finalize (see page 22460)	Uninitializes a dynamically allocated variable.
	FindClassHInstance (see page 22461)	Returns the instance handle for the module in which a class type is defined.
	FindHInstance (see page 22461)	Returns the instance handle of the module that contains a specified address.
	FindResourceHInstance (see page 22462)	Returns the instance handle of the resource module associated with a specified HInstance.
	Flush (see page 22462)	Empties the buffer of a text file opened for output.
	FPower10 (see page 22463)	This is function System.FPower10.
	Frac (see page 22463)	Returns the fractional part of a real number.
	FreeMem (see page 22464)	FreeMem frees a memory block.
	FreeMem (see page 22464)	FreeMem frees a memory block.
	FreeMemory (see page 22466)	FreeMemory frees a memory block.
	Get8087CW (see page 22466)	Returns the value of the 8087 code word.
	GetDir (see page 22467)	Returns the current directory.
	GetHeapStatus (see page 22468)	Returns the current status of the memory manager.
	GetLastError (see page 22469)	Returns the last error reported by an operating system API call.
	GetMem (see page 22469)	GetMem allocates a memory block.

	GetMemory (see page 22470)	GetMemory allocates a memory block.
	GetMemoryManager (see page 22472)	Returns the entry points of the currently installed memory manager.
	GetMemoryManager (see page 22472)	Returns the entry points of the currently installed memory manager.
	GetMemoryManagerState (see page 22472)	Fetches state information for the Memory Manager.
	GetMemoryMap (see page 22473)	Fetches the memory map.
	GetMinimumBlockAlignment (see page 22473)	Gets the minimum memory block alignment.
	GetModuleFileName (see page 22474)	Returns the fully qualified name for a module, given its handle.
	GetVariantManager (see page 22474)	Returns the entry points of the routines that define variant behavior.
	Halt (see page 22475)	Initiates abnormal termination of a program.
	Halt (see page 22475)	Initiates abnormal termination of a program.
	Hi (see page 22476)	Returns the high-order byte of X as an unsigned value.
	High (see page 22476)	Returns the highest value in the range of an argument.
	Inc (see page 22478)	Increments an ordinal value by one or N.
	Inc (see page 22478)	Increments an ordinal value by one or N.
	Include (see page 22479)	Adds an element to a Delphi set or adds an event handler in .NET applications.
	Initialize (see page 22480)	Initializes a dynamically allocated variable.
	Initialize (see page 22480)	Initializes a dynamically allocated variable.
	Insert (see page 22480)	Inserts a substring into a string beginning at a specified point.
	Int (see page 22481)	Returns the integer part of a real number.
	IOResult (see page 22482)	Returns the status of the last I/O operation performed.
	IsMemoryManagerSet (see page 22483)	Indicates whether the memory manager has been overridden using the SetMemoryManager (see page 22523) procedure.
	IsVariantManagerSet (see page 22483)	Indicates whether variant support is implemented in an application.
	Length (see page 22484)	Returns the number of characters in a string or elements in an array.
	Ln (see page 22485)	Returns the natural log of a real expression.
	Lo (see page 22486)	Returns the low order Byte of argument X.
	LoadResourceModule (see page 22486)	This is function System.LoadResourceModule.
	LoadResString (see page 22487)	This is function System.LoadResString.
	Low (see page 22487)	Returns the lowest value in a range.
	MkDir (see page 22488)	Creates a new subdirectory.
	MkDir (see page 22488)	Creates a new subdirectory.
	Move (see page 22490)	Copies bytes from a source to a destination.
	New (see page 22491)	Creates a new dynamic variable and sets P to point to it.
	Odd (see page 22493)	Returns true if argument is an odd number.
	OleStrToString (see page 22493)	Copies data received from a COM interface to a string.
	OleStrToStrVar (see page 22494)	Copies a string from its COM representation to an existing Delphi string.
	Ord (see page 22494)	Returns the ordinal value of an ordinal-type expression.
	ParamCount (see page 22495)	Returns the number of parameters passed on the command line.
	ParamStr (see page 22496)	Returns a specified parameter from the command-line.
	Pi (see page 22497)	Returns 3.1415926535897932385.
	Pos (see page 22497)	Returns the index value of the first character in a specified substring that occurs in a given string.
	Pos (see page 22497)	Returns the index value of the first character in a specified substring that occurs in a given string.
	Pred (see page 22499)	Returns the predecessor of the argument.
	Ptr (see page 22501)	Converts a specified address to a pointer.
	PUCS4Chars (see page 22501)	Converts a UCS4 string to a null-terminated array of UCS4 characters.
	RaiseList (see page 22502)	This is function System.RaiseList.
	Random (see page 22502)	Generates random numbers within a specified range.
	Random (see page 22502)	Generates random numbers within a specified range.
	Randomize (see page 22503)	Initializes the random number generator with a random value.
	Read (see page 22504)	Read reads data from a file.
	Read (see page 22504)	Read reads data from a file.
	Read (see page 22504)	Read reads data from a file.
	Read (see page 22504)	Read reads data from a file.

	ReadLn (see page 22506)	Reads a line of text from a file.
	ReadLn (see page 22506)	Reads a line of text from a file.
	ReadLn (see page 22506)	Reads a line of text from a file.
	ReadLn (see page 22506)	Reads a line of text from a file.
	ReallocMem (see page 22507)	ReallocMem reallocates a memory block.
	ReallocMemory (see page 22508)	ReallocMemory resizes a memory block.
	RegisterExpectedMemoryLeak (see page 22508)	Registers an expected memory leak.
	RegisterModule (see page 22509)	This is function System.RegisterModule.
	ReleaseExceptionObject (see page 22509)	Releases exception object acquired by AcquireExceptionObject (see page 22410).
	RemoveModuleUnloadProc (see page 22510)	RemoveModuleUnloadProc removes a call to your defined routine when a package is unloaded.
	RemoveModuleUnloadProc (see page 22510)	RemoveModuleUnloadProc removes a call to your defined routine when a package is unloaded.
	Rename (see page 22510)	Changes the name of an external file.
	Rename (see page 22510)	Changes the name of an external file.
	Reset (see page 22511)	Opens an existing file.
	Reset (see page 22511)	Opens an existing file.
	Reset (see page 22511)	Opens an existing file.
	Reset (see page 22511)	Opens an existing file.
	Rewrite (see page 22513)	Creates a new file and opens it.
	Rewrite (see page 22513)	Creates a new file and opens it.
	RmDir (see page 22515)	Deletes an empty subdirectory.
	RmDir (see page 22515)	Deletes an empty subdirectory.
	Round (see page 22516)	Returns the value of X rounded to the nearest whole number.
	RunError (see page 22517)	Stops execution and generates a runtime error.
	RunError (see page 22517)	Stops execution and generates a runtime error.
	Seek (see page 22518)	Moves the current position of a file to a specified component.
	SeekEof (see page 22519)	Returns the end-of-file status of a file, ignoring whitespace.
	SeekEof (see page 22519)	Returns the end-of-file status of a file, ignoring whitespace.
	SeekEoln (see page 22520)	Returns the end-of-line status of a file, ignoring whitespace.
	SeekEoln (see page 22520)	Returns the end-of-line status of a file, ignoring whitespace.
	Set8087CW (see page 22520)	Sets both the control word in the floating point unit and the variable NoErrMsg (see page 22652) declared in the system unit.
	SetInOutRes (see page 22521)	This is function System.SetInOutRes.
	SetLastError (see page 22521)	SetLastError sets the last API error code.
	SetLength (see page 22522)	Sets the length of a string or dynamic-array variable.
	SetLineBreakStyle (see page 22522)	Determines the end-of-line and end-of-file conventions for text file I/O.
	SetMemoryManager (see page 22523)	Sets entry points of the memory manager.
	SetMemoryManager (see page 22523)	Sets entry points of the memory manager.
	SetMinimumBlockAlignment (see page 22524)	Sets the minimum memory block alignment.
	SetMultiByteConversionCodePage (see page 22524)	SetMultiByteConversionCodePage overrides the default multi byte code conversion page.
	SetRaiseList (see page 22525)	This is function System.SetRaiseList.
	SetString (see page 22525)	Sets the contents and length of the given string.
	SetTextBuf (see page 22526)	Assigns an I/O buffer to a text file.
	SetTextBuf (see page 22526)	Assigns an I/O buffer to a text file.
	SetVariantManager (see page 22527)	Sets the entry points of routines that define variant behavior.
	ShareMemoryManager (see page 22527)	Shares the module's memory manager.
	Sin (see page 22528)	Returns the sine of the angle in radians.
	SizeOf (see page 22528)	Returns the number of bytes occupied by a variable or type.
	Slice (see page 22529)	Returns a sub-section of an array.
	Sqr (see page 22529)	Returns the square of a number.
	Sqr (see page 22529)	Returns the square of a number.
	Sqrt (see page 22530)	Returns the square root of X.
	Str (see page 22530)	Formats a string and returns it to a variable.

	Str (see page 22530)	Formats a string and returns it to a variable.
	StringOfChar (see page 22531)	Returns a string with a specified number of repeating characters.
	StringOfChar (see page 22531)	Returns a string with a specified number of repeating characters.
	StringToOleStr (see page 22531)	Allocates memory and copies an AnsiStringa string to the OLE format.
	StringToWideChar (see page 22532)	Returns a UNICODE string from an AnsiString (see page 22093).
	Succ (see page 22532)	Returns the successor of an argument.
	Swap (see page 22533)	Exchanges high order byte with the low order byte of an integer or word.
	SysAllocMem (see page 22533)	This is function System.SysAllocMem.
	SysFreeMem (see page 22534)	Frees the memory pointed to by a specified pointer.
	SysGetMem (see page 22534)	Allocates a specified number of bytes and returns a pointer to them.
	SysReallocMem (see page 22535)	Returns a pointer to a specified number of bytes, preserving the values pointed to by the Pointer parameter.
	SysRegisterExpectedMemoryLeak (see page 22536)	This is function System.SysRegisterExpectedMemoryLeak.
	SysUnregisterExpectedMemoryLeak (see page 22536)	This is function System.SysUnregisterExpectedMemoryLeak.
	TextStart (see page 22536)	This is function System.TextStart.
	Trunc (see page 22537)	Truncates a real number to an integer.
	Truncate (see page 22537)	Deletes all the records after the current file position.
	UCS4StringToWideString (see page 22538)	Converts a string of 4-byte Unicode characters into a WideString (see page 22377) value.
	UnicodeToUtf8 (see page 22538)	Converts a string of Unicode characters into a UTF-8 string.
	UnicodeToUtf8 (see page 22538)	Converts a string of Unicode characters into a UTF-8 string.
	UniqueString (see page 22539)	Ensures that a given string has a reference count of one.
	UniqueString (see page 22539)	Ensures that a given string has a reference count of one.
	UnregisterExpectedMemoryLeak (see page 22539)	Unregisters an expected memory leak.
	UnregisterModule (see page 22540)	This is function System.UnregisterModule.
	UpCase (see page 22540)	Converts a character to uppercase.
	UTF8Decode (see page 22541)	Converts a UTF8 string to a Unicode string (WideString (see page 22377)).
	UTF8Encode (see page 22541)	Converts a WideString (see page 22377) value to a UTF8 string.
	Utf8ToAnsi (see page 22542)	Converts a UTF8 string to a string of Ansi characters.
	Utf8ToUnicode (see page 22542)	Converts a string of UTF-8 characters to a Unicode string.
	Utf8ToUnicode (see page 22542)	Converts a string of UTF-8 characters to a Unicode string.
	Val (see page 22543)	Converts a string to a numeric representation.
	VarArrayRedim (see page 22544)	Resizes a Variant (see page 22307) array.
	VarArrayRedim (see page 22544)	This is function System::VarArrayRedim.
	VarCast (see page 22545)	This is function System::VarCast.
	VarClear (see page 22545)	Empties a Variant (see page 22307), so that it is unassigned.
	VarClear (see page 22546)	This is function System::VarClear.
	WideCharLenToString (see page 22546)	Converts Unicode characters to single or multi byte character data.
	WideCharLenToStrVar (see page 22546)	WideCharLenToStrVar converts Unicode characters to single or multi byte character data.
	WideCharToString (see page 22547)	Converts null-terminated Unicode string to single or multi byte character data.
	WideCharToStrVar (see page 22547)	Converts Unicode string to a single or multi byte character data.
	WideStringToUCS4String (see page 22548)	Converts a WideString (see page 22377) value to a string of 4-byte Unicode characters.
	Write (see page 22548)	Writes to either a typed file or a text file
	Write (see page 22548)	Writes to either a typed file or a text file
	Write (see page 22548)	Writes to either a typed file or a text file
	Write (see page 22548)	Writes to either a typed file or a text file
	WriteLn (see page 22550)	Writes to a text file and adds an end-of-line marker.
	WriteLn (see page 22550)	Writes to a text file and adds an end-of-line marker.
	WriteLn (see page 22550)	Writes to a text file and adds an end-of-line marker.
	WriteLn (see page 22550)	Writes to a text file and adds an end-of-line marker.

158.3.1 System::!= Function

Pascal

```
operator !=(const w: WideString; const rhs: WideString&);
```

C++

```
__fastcall bool operator !=(const BSTR w, const WideString& rhs);
```

File

wstring.h

Description

This is function System::!=.

158.3.2 System::__currMul Function

Pascal

```
procedure __currMul(const c1: System::Currency; var c2: System::Currency);
```

C++

```
__fastcall void __currMul(const System::Currency c1, System::Currency & c2);
```

File

syscurr.h

Description

This is function System::__currMul.

158.3.3 System._CVR_PROBE Function

Pascal

```
procedure _CVR_PROBE(mi: PCVModInfo; probeNum: Cardinal); cdecl;
```

C++

```
__cdecl _CVR_PROBE(PCVModInfo mi, unsigned probeNum);
```

File

System

Description

This is function System._CVR_PROBE.

158.3.4 System._CVR_STMTPROBE Function

Pascal

```
function _CVR_STMTPROBE(mi: PCVModInfo; probeNum: Cardinal; TrueFalse: Cardinal): Boolean;
cdecl;
```

C++

```
__cdecl Boolean _CVR_STMTPROBE(PCVModInfo mi, unsigned probeNum, unsigned TrueFalse);
```

File

System

Description

This is function System._CVR_STMTPROBE.

158.3.5 System::_roundToInt64 Function

Pascal

```
function _roundToInt64(: long double): __int64; cdecl;
```

C++

```
__cdecl __int64 _roundToInt64(long double);
```

File

syscurr.h

Description

This is function System::_roundToInt64.

158.3.6 System::+ Function

Pascal

```
operator +(const : char*; const : AnsiString&);
```

C++

```
__fastcall AnsiString operator +(const char*, const AnsiString&);
```

File

dstring.h

Description

This is function System::+.

158.3.7 System::< Function

Pascal

```
operator <(const w: WideString; const rhs: WideString&);
```

C++

```
__fastcall bool operator <(const BSTR w, const WideString& rhs);
```

File

wstring.h

Description

This is function System::<.

158.3.8 System::<< Function

Pascal

```
operator <<(var os: std::ostream; const arg: Currency&);
```

C++

```
std::ostream& operator <<(std::ostream& os, const Currency& arg);
```

File

syscurr.h

Description

This is function System::<<.

158.3.9 System::<< Function

Pascal

```
operator <<(var os: std::ostream; const arg: TDateTime&);
```

C++

```
std::ostream& operator <<(std::ostream& os, const TDateTime& arg);
```

File

systdate.h

Description

This is function System::<<.

158.3.10 System::<< Function

Pascal

```
operator <<(var os: std::ostream; const arg: Variant&);
```

C++

```
std::ostream& operator <<(std::ostream& os, const Variant& arg);
```

File

sysvari.h

Description

This is function System::<<.

158.3.11 System::<= Function

Pascal

```
operator <=(const w: WideString; const rhs: WideString&);
```

C++

```
__fastcall bool operator <=(const BSTR w, const WideString& rhs);
```

File

wstring.h

Description

This is function System::<=.

158.3.12 System::== Function

Pascal

```
operator ==(const w: WideString; const rhs: WideString&);
```

C++

```
__fastcall bool operator ==(const BSTR w, const WideString& rhs);
```

File

wstring.h

Description

This is function System::==.

158.3.13 System::> Function

Pascal

```
operator >(const w: WideString; const rhs: WideString&);
```

C++

```
__fastcall bool operator >(const BSTR w, const WideString& rhs);
```

File

wstring.h

Description

This is function System::>.

158.3.14 System::>= Function

Pascal

```
operator >=(const w: WideString; const rhs: WideString&);
```

C++

```
__fastcall bool operator >=(const BSTR w, const WideString& rhs);
```

File

wstring.h

Description

This is function System::>=.

158.3.15 System::>> Function

Pascal

```
operator >>(var is: std::istream; var arg: Currency);
```

C++

```
std::istream& operator >>(std::istream& is, Currency& arg);
```

File

syscurr.h

Description

This is function System::>>.

158.3.16 System::>> Function

Pascal

```
operator >>(var is: std::istream; var arg: TDateTime);
```

C++

```
std::istream& operator >>(std::istream& is, TDateTime& arg);
```

File

systdate.h

Description

This is function System::>>.

158.3.17 System.Abs Function

Returns an absolute value.

Pascal

```
procedure Abs(X);
```

C++

```
Abs( X);
```

File

System

Description

Abs returns the absolute value of the argument, X.

X is an integer-type or real-type expression.

Delphi Examples:

```
{
This example converts negative floating point numbers and
integers to postive.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.Text := '-2.3';
  Edit3.Text := FloatToStr(Abs(StrToFloat(Edit1.Text)));    { 2.3 }
  Edit2.Text := '-157';
  Edit4.Text := IntToStr(Abs(StrToInt(Edit2.Text)));    { 157 }
end;
```

158.3.18 System.AcquireExceptionObject Function

Retains access to exception object.

Pascal

```
function AcquireExceptionObject: Pointer;
```

C++

```
void * AcquireExceptionObject();
```

File

System

Description

AcquireExceptionObject returns a pointer to the current exception object and prevents the exception object from being deallocated when the current exception handler exits.

Note: AcquireExceptionObject increments the exception object's reference count. Make sure that the reference count is decremented when the exception object is no longer needed. This happens automatically if you use the exception object to re-raise the exception. In all other cases, every call to AcquireExceptionObject must have a matching call to ReleaseExceptionObject (see page 22509). AcquireExceptionObject/ReleaseExceptionObject (see page 22509) sequences can be nested.

See Also

ReleaseExceptionObject (see page 22509)

ExceptObject (see page 22452)

on

catch

158.3.19 System.AddModuleUnloadProc Function

AddModuleUnloadProc adds a call to your defined routine when a package is unloaded.

Pascal

```
procedure AddModuleUnloadProc(Proc: TModuleUnloadProc); overload;
procedure AddModuleUnloadProc(Proc: TModuleUnloadProcLW); overload;
```

C++

```
AddModuleUnloadProc(TModuleUnloadProc Proc);
AddModuleUnloadProc(TModuleUnloadProcLW Proc);
```

File

System

Description

Before Delphi unloads a package that was used by your application, it can optionally call your **Proc** routine. In essence, AddModuleUnloadProc adds your callback routine to the head of the list of internal routines that Delphi invokes to perform the unload.

Use RemoveModuleUnloadProc (see page 22510) to reverse the effect of AddModuleUnloadProc.

See Also

RemoveModuleUnloadProc (see page 22510)

TModuleUnloadProc (see page 22617)

158.3.20 System.Addr Function

Returns a pointer to a specified object.

Pascal

```
function Addr(var X): Pointer;
```

C++

```
void * Addr( X);
```

File

System

Description

The Addr function returns the address of a specified object. X is any variable, procedure or function identifier. The result is a pointer to X.

The result of Addr is of the predefined type Pointer, which means that it is assignment-compatible with all pointer types but can't be dereferenced directly without a typecast.

Note: Addr is equivalent to the @ operator except that it is unaffected by the $T compiler directive.

Delphi Examples:

```
{
This example uses the Addr function to get the address of a
numeric value that is stored as custom data in a tree node.
Once the assignment has been made, changes to the entries in
the NodeNumbers array can be read by dereferencing the
pointers that are stored in the Data property of the Tree
nodes.  This example requires a populated TreeView.
}

var
  NodeNumbers: array [0 .. 200] of String;

procedure TForm1.Button1Click(Sender: TObject);
var
  I : Integer;
 begin
  with TreeView1 do
  begin
    for I := 0 to Items.Count - 1 do
    begin
      NodeNumbers[I] := Items[I].Text;
      Items[I].Data := Addr(NodeNumbers[I]);
      StringGrid1.Cells[0, I + 1] := IntToStr(I);
      StringGrid1.Cells[1, I + 1] := IntToStr(Integer(Items[I].Data));
      StringGrid1.Cells[2, I + 1] := NodeNumbers[I];
    end;
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  I : Integer;
  addr : Pointer;
begin
```

```
    I := StrToInt(Edit2.Text);
    addr := TreeView1.Items[I].Data;
    string(addr^) := Edit1.Text;  // NodeNumbers has been altered
    TreeView1.Items[I].Text := NodeNumbers[I]; // Update TreeView
    StringGrid1.Cells[2, I + 1] := NodeNumbers[I];  // Update StringGrid
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    StringGrid1.Cells[0, 0] := 'Node index';
    StringGrid1.Cells[1, 0] := 'Node addr';
    StringGrid1.Cells[2, 0] := 'Node text';
  end;
```

See Also

Ptr (see page 22501)

IDH

158.3.21 System.AllocMem Function

AllocMem allocates a memory block and initializes each byte to zero.

Pascal

```
function AllocMem(Size: Cardinal): Pointer;
```

C++

```
void * AllocMem(unsigned Size);
```

File

System

Description

AllocMem allocates a block of the given **Size** on the heap, and returns the address of this memory. Each byte in the allocated buffer is set to zero. To dispose of the buffer, use FreeMem (see page 22464). If there isn't enough memory available to allocate the block, an EOutOfMemory exception is raised.

Note: If the memory does not need to be zero-initialized, it is more efficient to use GetMem (see page 22469) instead.

C++ Examples:

```
/*
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
  char *pszBuffer;
```

```
  if (OpenDialog1->Execute())
  {
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      pszBuffer = PChar(AllocMem(iFileLength+1));
//    pszBuffer = new char[iFileLength+1];
      iBytesRead = FileRead(iFileHandle, pszBuffer, iFileLength);
      FileClose(iFileHandle);
      for (int i=0;i<iBytesRead;i++)
      {
        StringGrid1->RowCount += 1;
        StringGrid1->Cells[1][i+1] = pszBuffer[i];
        StringGrid1->Cells[2][i+1] = IntToStr((int)pszBuffer[i]);
      }
      delete [] pszBuffer;
    }
    catch(...)
    {
      Application->MessageBox("Can't perform one of the following file operations: Open, Seek,
Read, Close.", "File Error", IDOK);
    }
  }
}
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  iFileHandle: Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PChar;
  i: Integer;
begin
  if OpenDialog1.Execute then
  begin
    try
      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      Buffer := PChar(System.AllocMem(iFileLength + 1));
      iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
      FileClose(iFileHandle);
      for i := 0 to iBytesRead-1 do
      begin
        StringGrid1.RowCount := StringGrid1.RowCount + 1;
        StringGrid1.Cells[1,i+1] := Buffer[i];
        StringGrid1.Cells[2,i+1] := IntToStr(Integer(Buffer[i]));
      end;
    finally
      FreeMem(Buffer);
    end;
  end;
end;
```

See Also

FreeMem (see page 22464)

FreeMemory (see page 22466)

GetMem (see page 22469)

GetMemory (see page 22470)

EOutOfMemory (see page 22742)

ReallocMem (see page 22507)

ReallocMemory (see page 22508)

158.3.22 System.AnsiToUtf8 Function

Converts a string encoded in Ansi to UTF-8.

Pascal

```
function AnsiToUtf8(const S: string): UTF8String;
```

C++

```
UTF8String AnsiToUtf8(const AnsiString S);
```

File

System

Description

AnsiToUtf8 converts the string specified by S, which uses the ANSI encoding system, to UTF-8.

See Also

Utf8ToAnsi (see page 22542)

Utf8ToUnicode (see page 22542)

WideStringToUCS4String (see page 22548)

158.3.23 System.Append Function

Prepares an existing file for adding text to the end.

Pascal

```
procedure Append(var F: Text);
```

C++

```
Append(Text F);
```

File

System

Description

Call Append to ensure that a file is opened with write-only access with the file pointer positioned at the end of the file. F is a text file variable and must be associated with an external file using AssignFile (see page 22420). If no external file of the given name exists, an error occurs. If F is already open, it is closed, then reopened. The current file position is set to the end of the file.

Note: If a Ctrl+Z (ASCII 26) is present in the last 128-byte block of the file, the current file position is set so that the next character added to the file overwrites the first Ctrl+Z in the block. In this way, text can be appended to a file that terminates with a Ctrl+Z.

If F was not assigned a name, then, after the call to Append, F refers to the standard output file.

Delphi Examples:

```
{
Append, Flush example
}

procedure TForm1.Button1Click(Sender: TObject);
var
  f: TextFile;
begin
  if OpenDialog1.Execute then
  begin                    { open a text file }
    AssignFile(f, OpenDialog1.FileName);
    Append(f);
    Writeln(f, 'I am appending some stuff to the end of the file.');
    { insert code here that would require a Flush before closing the file }
    Flush(f);  { ensures that the text was actually written to file }
    CloseFile(f);
    MessageDlg(OpenDialog1.FileName + ' has been altered, please validate.', mtInformation,
[mbOK], 0)
  end;
end;
```

See Also

AssignFile (see page 22420)

FileClose

Reset (see page 22511)

Rewrite (see page 22513)

IDH

158.3.24 System.ArcTan Function

Calculates the arctangent of a given number.

Pascal

```
function ArcTan(const X: Extended): Extended;
```

C++

```
Extended ArcTan(const Extended X);
```

File

System

Description

In Delphi code, ArcTan returns the arctangent of X.

X is a real-type expression that gives an angle in radians.

Delphi Examples:

```
{
This example shows how to use the ArcTan and FloatToStr
functions.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ex : Extended;
  mystring : String;
begin
  ex := 16 * ArcTan(1/5) - 4 * ArcTan(1/239); // Machin's formula
  mystring := FloatToStr(ex);
  Edit2.Text := mystring;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  ex : Extended;
  myString : String;
begin
  ex := Pi;
  mystring := FloatToStr(ex);
  Edit1.Text := mystring;
end;
```

See Also

Cos (see page 22440)

Sin (see page 22528)

ArcTan2

Tan

ArcSin

ArcCos

158.3.25 System.Assert Function

Tests whether a Boolean expression is true.

Pascal

```
procedure Assert(expr: Boolean); overload;
procedure Assert(expr: Boolean; const msg: string); overload;
```

C++

```
Assert(Boolean expr);
Assert(Boolean expr, const AnsiString msg);
```

File

System

Description

In Delphi code, use Assert as a debugging tool to test that conditions assumed to be true are never violated. Assert provides an opportunity to intercept an unexpected condition and halt a program rather than allow execution to continue under unanticipated conditions.

Assert takes a Boolean expression and an optional message string as parameters. If the Boolean test fails, Assert raises an EAssertionFailed exception. If a message string was passed to Assert, the exception object is created with that string. Otherwise it is created with a default string indicating that the assertion failed. The message is displayed along with the complete path, filename, and the line number on which Assert failed.

The SysUtils (see page 22709) unit causes runtime errors to be turned into exceptions. If SysUtils (see page 22709) is not used anywhere in your application, you will get a runtime error 227 rather than an EAssertionFailed exception. This runtime error will halt the program.

Because assertions are not usually used in shipping versions of a product, compiler directives are provided to disable the generation of assertion code:

$ASSERTIONS ON/OFF(long form)

$C +/-(short form)

These are global switches that affect the entire source file where they occur, regardless of their position in the file. It is not possible to enable and disable assertions for something smaller than a source file. Assertions are on by default.

Delphi Examples:

```
{
This example exercises the System Assert function.  The first
call passes and the second call fails.
}

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TStorage = class(TObject)
    FData: string;
    property Data: string read FData write FData;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure ModifyStorage(AStorage: TStorage; const s: string);
begin
  Assert(AStorage <> nil, '');
  AStorage.Data := s;
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Storage: TStorage;
begin
  Storage := TStorage.Create;
  try
    ModifyStorage(Storage, 'Hello world');
  finally
    Storage.Free;
  end;

  // The following call is buggy and triggers the Assert
  ModifyStorage(nil, 'Ooops');
end;
```

See Also

EAssertionFailed

158.3.26 System.Assigned Function

Tests for a nil (unassigned) pointer or procedural variable.

Pascal

```
function Assigned(const P): Boolean;
```

C++

```
Boolean Assigned(const  P);
```

File

System

Description

Use Assigned to determine whether the pointer or procedure referenced by P is nil. P must be a variable reference of a pointer or procedural type. Assigned(P) corresponds to the test P<> nil for a pointer variable, and @P <> nil for a procedural variable.

Assigned returns false if P is nil, true otherwise.

Note: Assigned can't detect a dangling pointer--that is, one that isn't nil but no longer points to valid data. For example, in the code example for Assigned, Assigned won't detect the fact that P isn't valid.

Delphi Examples:

```
{
This example demonstrates the System Assigned function succeeds.
}
procedure TForm1.Button1Click(Sender: TObject);
var P: Pointer;
begin
  P := nil;
  if Assigned (P) then
    MessageDlg('You won''t see this', mtInformation, [mbOK], 0);
  GetMem(P, 1024);  {P valid}
  FreeMem(P, 1024); {P no longer valid and still not nil}
  if Assigned (P) then
    MessageDlg('You''ll see this', mtInformation, [mbOK], 0);
end;
```

158.3.27 System.AssignFile Function

Associates the name of an external file with a file variable.

Pascal

```
procedure AssignFile(var F; FileName: string);
```

C++

```
AssignFile( F, AnsiString FileName);
```

File

System

Description

Call AssignFile to initialize a file variable in Delphi code. F is a file variable of any file type. FileName is a string-type expression or an expression of type PChar if extended syntax is enabled.

After calling AssignFile, F is associated with the external file until F is closed. All further operations on the file variable F operate on the external file named by FileName.

When the FileName parameter is empty, AssignFile associates F with the standard input or standard output file. If assigned an empty name, after a call to Reset (see page 22511) (F), F refers to the standard input file, and after a call to Rewrite (see page 22513) (F), F refers to the standard output file.

Do not use AssignFile on a file variable that is already open.

Note: To avoid scope conflicts, AssignFile replaces the Assign procedure that was available in early versions of the Delphi product. However, for backward compatibility Assign is still available.

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], 1);
```

```
        if (filehandle = -1) then
        begin
          MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                     ' cannot be opened with access mode 1.', mtError, [mbOk], 0);
          Continue;
        end
        else
          FileClose(filehandle);

        AssignFile(F, FileListBox1.Items.Strings[i]);
        Reset(F, 1);
        ListBox1.Items.Add(
          FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
        CloseFile(F);
      end;
     finally
     { do something here }
     end;
    end;
  end;
  {
  Use a TOpenDialog to select a file.  Place the first line of
  the file into a TEdit.
  }
  procedure TForm1.Edit1Click(Sender: TObject);
  var
    F: TextFile;
    S: string;
  begin
    if OpenDialog1.Execute then            { Display Open dialog box }
    begin
      AssignFile(F, OpenDialog1.FileName); { File selected in dialog }
      Reset(F);
      Readln(F, S);                        { Read first line of file }
      Edit1.Text := S;                     { Put string in a TEdit control }
      CloseFile(F);
    end;
  end;
  {
  Click the button to open a TOpenDialog, then select a file
  to delete.
  }

  procedure TForm1.Button1Click(Sender: TObject);
  var
    F: Textfile;
  begin
    OpenDialog1.Title := 'Delete File';
    if OpenDialog1.Execute then
    begin
      AssignFile(F, OpenDialog1.FileName);
      try
        Reset(F);
        if MessageDlg('Erase ' + OpenDialog1.FileName + '?',
          mtConfirmation, [mbYes, mbNo], 0) = mrYes then
        begin
          CloseFile(F);
          Erase(F);
        end;
      except
        on EInOutError do
          MessageDlg('File I/O error.', mtError, [mbOk], 0);
      end;
    end;
  end;
```

See Also

Append (see page 22415)

CloseFile (see page 22433)

Reset (see page 22511)

Rewrite (see page 22513)

IDH

158.3.28 System.AttemptToUseSharedMemoryManager Function

Attempts to use a shared memory manager.

Pascal

```
function AttemptToUseSharedMemoryManager: Boolean;
```

C++

```
Boolean AttemptToUseSharedMemoryManager();
```

File

System

Description

AttemptToUseSharedMemoryManager searches the current process for a shared memory manager. If no memory has been allocated using this module's memory manager, it will switch to using the shared memory manager. It will return **True** if it found a shared memory manager to which it could switch. It will return **False** otherwise.

Note: The memory manager cannot be changed once memory has been allocated, since the new memory manager cannot free the pointers allocated through a different memory manager. This function is not thread safe.

See Also

Memory Management

Configuring the Memory Manager

Sharing Memory

158.3.29 System.BeginThread Function

Spawns a separate thread of execution.

Pascal

```
function BeginThread(SecurityAttributes: Pointer; StackSize: LongWord; ThreadFunc:
TThreadFunc; Parameter: Pointer; CreationFlags: LongWord; var ThreadId: LongWord): Integer;
```

C++

```
int BeginThread(void * SecurityAttributes, LongWord StackSize, TThreadFunc ThreadFunc, void *
Parameter, LongWord CreationFlags, LongWord ThreadId);
```

File

System

Description

Use this routine or a TThread object to spawn separate threads of execution. BeginThread spawns a new thread of execution and sets the global IsMultiThread (see page 22648) variable, thereby making the heap thread-safe.

ThreadFunc is the thread function. This is the code that executes in the new thread of execution. ThreadFunc should handle all of its own exceptions. However, because BeginThread sets up an execution frame, the system's default exception handler will catch any exceptions that escape ThreadFunc.

Parameter points to a single 32-bit parameter that is passed, uninterpreted, to ThreadFunc.

ThreadId returns the unique identifier for the new thread. This ID appears in the Thread Status box of the debugger.

SecurityAttributes is a pointer to a Windows SecurityAttributes record (structure). For details on security attributes, see the Microsoft documentation.

StackSize is the size of the stack reserved for the new thread when CreationFlags includes STACK_SIZE_IS_A_RESERVATION. Otherwise, this parameter is ignored.

CreationFlags is a set of bits or'ed together that are drawn from the following:

Value	Meaning
CREATE_SUSPENDED	The thread is created in a suspended state and does not run until the Windows API ResumeThread function is called. If this flag is not included, the new thread starts running as soon as it is created.
STACK_SIZE_PARAM_IS_A_RESERVATION	The StackSize parameter specifies the stack size for the thread. If this flag is not included, the new thread is assigned a default stack size by the linker.

On Windows, BeginThread returns the Windows thread handle. A return value of 0 indicates failure.

Attribute is a structure containing information about the thread such as its priority, its schedule policy, and its stack size.

On Linux, Begin Thread returns 0 if the thread is successfully spawned, an error code otherwise.

See Also

EndThread (see page 22447)

IsMultiThread (see page 22648)

Defining Thread Objects

158.3.30 System.BlockRead Function

Reads one or more records from an open file into a variable.

Pascal

```
procedure BlockRead(var F: File; var Buf; Count: Integer); overload;
procedure BlockRead(var F: File; var Buf; Count: Integer; var AmtTransferred: Integer);
overload;
```

C++

```
BlockRead(File F,  Buf, int Count);
BlockRead(File F,  Buf, int Count, int AmtTransferred);
```

File

System

Description

F is an untyped file variable, Buf is any variable, Count is an expression of type Integer, and AmtTransferred is an optional variable of type Integer.

BlockRead reads Count or fewer records from the file F into memory, starting at the first byte occupied by Buf. The actual number of complete records read (less than or equal to Count) is returned in AmtTransferred.

The entire transferred block occupies at most Count * RecSize bytes. RecSize is the record size specified when the file was opened (or 128 if the record size was not specified).

If the entire block was transferred, AmtTransferred is equal to Count.

If AmtTransferred is less than Count, ReadBlock reached the end of the file before the transfer was complete. If the file's record size is greater than 1, AmtTransferred returns the number of complete records read.

If AmtTransferred isn't specified, an I/O error occurs if the number of records read isn't equal to Count. If the $I+ compiler directive is in effect, errors raise an EInOutError exception.

Delphi Examples:

```
{
This example reads an entire file into a buffer with one
command and then writes it into a saved file.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FromF, ToF: file;
  NumRead, NumWritten: Integer;
  Buf: array[1..2048] of Char;
begin
  if OpenDialog1.Execute then     { Display Open dialog box }
  begin
    AssignFile(FromF, OpenDialog1.FileName);
    Reset(FromF, 1);    { Record size = 1 }
    if SaveDialog1.Execute then      { Display Save dialog box}
    begin
      AssignFile(ToF, SaveDialog1.FileName);    { Open output file }
      Rewrite(ToF, 1);  { Record size = 1 }
      Canvas.TextOut(10, 10, 'Copying ' + IntToStr(FileSize(FromF))
        + ' bytes...');
      repeat
        System.BlockRead(FromF, Buf, SizeOf(Buf), NumRead);
        BlockWrite(ToF, Buf, NumRead, NumWritten);
      until (NumRead = 0) or (NumWritten <> NumRead);
      CloseFile(FromF);
      CloseFile(ToF);
    end;
  end;
end;
{
The following example opens a file of your choice and reads
the entire file into a dynamically allocated buffer. The
buffer and the size of the file are then passed to a routine
that processes the text, and finally the dynamically
allocated buffer is freed and the file is closed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: file;
  Size: Integer;
```

```
    Buffer: PChar;
  begin
    if OpenDialog1.Execute then
    begin
      AssignFile(F, OpenDialog1.FileName);
      Reset(F, 1);
      try
        Size := FileSize(F);
        GetMem(Buffer, Size);
        try
          BlockRead(F, Buffer^, Size);
          Memo1.Lines.Add(Buffer);
        finally
          FreeMem(Buffer);
        end;
      finally
        CloseFile(F);
      end;
    end;
  end;
```

See Also

TFileStream

EInOutError

158.3.31 System.BlockWrite Function

Writes one or more records from a variable to an open file.

Pascal

```
procedure BlockWrite(var f: File; var Buf; Count: Integer); overload;
procedure BlockWrite(var f: File; var Buf; Count: Integer; var AmtTransferred: Integer);
overload;
```

C++

```
BlockWrite(File f,  Buf, int Count);
BlockWrite(File f,  Buf, int Count, int AmtTransferred);
```

File

System

Description

F is an untyped file variable, Buf is any variable, Count is an expression of type Integer, and AmtTransferred is an optional variable of type Integer.

BlockWrite writes Count or fewer records to the file F from memory, starting at the first byte occupied by Buf. The actual number of complete records written (less than or equal to Count) is returned in AmtTransferred.

The entire block transferred occupies at most Count * RecSize bytes. RecSize is the record size specified when the file was opened (or 128 if the record size was unspecified).

If the entire block is transferred, AmtTransferred is equal to Count on return.

If AmtTransferred is less than Count, the disk became full before the transfer was complete. In this case, if the file's record size is greater than 1, AmtTransferred returns the number of complete records written.

BlockWrite advances the current file position by AmtTransferred records.

If AmtTransferred isn't specified, an I/O error occurs if the number written isn't equal to Count. If the $I+ compiler directive is in effect, errors raise an EInOutError exception.

Delphi Examples:

```
{
This example reads an entire file into a buffer with one
command and then writes it into a saved file.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FromF, ToF: file;
  NumRead, NumWritten: Integer;
  Buf: array[1..2048] of Char;
begin
  if OpenDialog1.Execute then     { Display Open dialog box }
  begin
    AssignFile(FromF, OpenDialog1.FileName);
    Reset(FromF, 1);    { Record size = 1 }
    if SaveDialog1.Execute then      { Display Save dialog box}
    begin
      AssignFile(ToF, SaveDialog1.FileName);    { Open output file }
      Rewrite(ToF, 1);  { Record size = 1 }
      Canvas.TextOut(10, 10, 'Copying ' + IntToStr(FileSize(FromF))
        + ' bytes...');
      repeat
        System.BlockRead(FromF, Buf, SizeOf(Buf), NumRead);
        BlockWrite(ToF, Buf, NumRead, NumWritten);
      until (NumRead = 0) or (NumWritten <> NumRead);
      CloseFile(FromF);
      CloseFile(ToF);
    end;
  end;
end;
```

See Also

TFileStream

EInOutError

158.3.32 System.Break Function

Causes the flow of control to exit a for, while, or repeat statement.

Pascal

```
procedure Break;
```

C++

```
Break();
```

File

System

Description

The Break procedure causes the flow of control in Delphi code to exit a for, while, or repeat statement and continue at the next statement following the loop statement.

A call to Break must be contained in a for, while, or repeat statement, or the compiler reports an error.

Note: Break does not violate the flow of control dictated by a try..finally construct. If a break occurs inside the try clause, the finally clause is entered.

Delphi Examples:

```
{
The following OnFind event handler searches a memo component
for the text specified in the FindText property of a find
dialog component. If found, the first occurrence of the text
in Memo1 is selected. The code uses the Pos function to
compare strings, and stores the number of characters to skip
when determining the selection position in the SkipChars
variable. Because there is no handling of case, whole word,
or search direction in this algorithm, it is assumed that
the Options property of FindDialog1 is set to
[frHideMatchCase, frHideWholeWord, frHideUpDown].
}
procedure TForm1.FindDialog1Find(Sender: TObject);
var
  I, J, PosReturn, SkipChars: Integer;
begin
  for I := 0 to Memo1.Lines.Count do
  begin
    PosReturn := Pos(FindDialog1.FindText,Memo1.Lines[I]);
    if PosReturn <> 0 then {found!}
    begin
      SkipChars := 0;
      for J := 0 to I - 1 do
        SkipChars := SkipChars + Length(Memo1.Lines[J]);
      SkipChars := SkipChars + (I*2);
      SkipChars := SkipChars + PosReturn - 1;
      Memo1.SetFocus;
      Memo1.SelStart := SkipChars;
      Memo1.SelLength := Length(FindDialog1.FindText);
      Break;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
const Path = 'country.list';
begin
  Memo1.Lines.LoadFromFile(Path);
end;
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
Note: Set the TComboBox AutoComplete to false to turn off
default auto completion.
}
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
var
  Found: boolean;
  i, ct, SelSt: Integer;
  TmpStr: string;
begin
  { first, process the keystroke to obtain the current string }
  { This code requires all items in list to be uppercase}
```

```
    if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
    ct := (Sender as TComboBox).Items.Count;
    with (Sender as TComboBox) do
    begin
      SelSt := SelStart;
      if (Key = Chr(vk_Back)) and (SelLength <> 0) then
       TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
      else if Key = Chr(vk_Back) then {SelLength = 0}
       TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
      else {Key in ['A'..'Z', etc]}
       TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
      if TmpStr = '' then Exit;
      { update SelSt to the current insertion point }
      if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
      else if Key <> Chr(vk_Back) then Inc(SelSt);
      Key := #0; { indicate that key was handled }
      if SelSt = 0 then
      begin
        Text:= '';
        Exit;
      end;
     {Now that TmpStr is the currently typed string, see if we can locate a match }
      Found := False;
      for i := 1 to ct do
      begin
        if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
        begin
          Text := Items[i-1]; { update to the match that was found }
          ItemIndex := i-1;
          Found := True;
          Break;
        end;
      end;
      if Found then { select the untyped end of the string }
      begin
        SelStart := SelSt;
        SelLength := Length(Text) - SelSt;
      end
      else Beep;
    end;
  end;

  var
    CharSetList : TStringList;

  procedure TForm1.AddCharacterSet(const S: string);
  begin
    CharSetList.Add(S);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    CharSetList := TStringList.Create;
    Graphics.GetCharsetValues(AddCharacterSet);
    ComboBox1.Items := CharSetList;
  end;
```

See Also

Halt (see page 22475)

158.3.33 System.ChDir Function

Changes the current directory.

Pascal

```
procedure ChDir(const S: string); overload;
procedure ChDir(P: PChar); overload;
```

C++

```
ChDir(const AnsiString S);
ChDir(const char * P);
```

File

System

Description

Note: ChDir changes the current directory to the path specified by S or P. If this operation fails, EInOutError is raised.

Note: On Windows, the path can include a drive specifier, which causes the current drive to be changed as well.

Note: In Delphi, {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 22482) to check for an I/O error.

Delphi Examples:

```
{
This example shows how to change and display the current
directory through text.  This example requires that a drive
combo box, a file list box, and a directory list box are on
a form. Add this code as the OnChange event handler for the
drive combo box and the OnChange event handler for the
directory list box.  When the user changes the drive using
the combo box, the directory list box and file list box will
update to reflect the new drive and the current directory on
that drive.  When the user double clicks on a directory in
the file list box will update to reflect the new directory.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  System.ChDir(Edit1.Text);
  if IOResult <> 0 then
    MessageDlg('Cannot find directory', mtWarning, [mbOk], 0);
  Edit2.Text := GetCurrentDir;
  FileListBox1.Directory := GetCurrentDir;
  DirectoryListBox1.Directory := GetCurrentDir;
end;

procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
  Edit2.Text := GetCurrentDir;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
```

```
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
  Edit2.Text := GetCurrentDir;
end;
```

See Also

GetDir (see page 22467)

MkDir (see page 22488)

RmDir (see page 22515)

SetCurrentDir

EInOutError

158.3.34 System.Chr Function

Returns the character for a specified ASCII value.

Pascal

```
function Chr(X: Byte): Char;
```

C++

```
Char Chr(Byte X);
```

File

System

Description

Chr returns the character with the ordinal value (ASCII value) of the byte-type expression, X.

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
```

```
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
    ssHorizontal,
    ssNone,
    ssVertical);
begin
  RichEdit1.ScrollBars :=
    ScrollBarA[RadioGroup1.ItemIndex];
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with RadioGroup1, Items do
  begin
    Caption := 'Scroll Bars';
    Clear;
    Add('ssBoth');
    Add('ssHorizontal');
    Add('ssNone');
    Add('ssVertical');
  end;
  with CheckBox1 do
  begin
    Caption := 'HideScrollBars';
    Checked := True;
  end;
  Button1.Caption := 'Loaded';
  RichEdit1.Text := 'Just a little text.';
  RichEdit1.WordWrap := False;
end;
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
Note: Set the TComboBox AutoComplete to false to turn off
default auto completion.
}
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
var
  Found: boolean;
  i, ct, SelSt: Integer;
  TmpStr: string;
begin
  { first, process the keystroke to obtain the current string }
  { This code requires all items in list to be uppercase}
  if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
  ct := (Sender as TComboBox).Items.Count;
  with (Sender as TComboBox) do
  begin
    SelSt := SelStart;
    if (Key = Chr(vk_Back)) and (SelLength <> 0) then
     TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
    else if Key = Chr(vk_Back) then {SelLength = 0}
     TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
    else {Key in ['A'..'Z', etc]}
```

```
      TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
     if TmpStr = '' then Exit;
     { update SelSt to the current insertion point }
     if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
     else if Key <> Chr(vk_Back) then Inc(SelSt);
     Key := #0; { indicate that key was handled }
     if SelSt = 0 then
     begin
       Text:= '';
       Exit;
     end;
    {Now that TmpStr is the currently typed string, see if we can locate a match }
     Found := False;
     for i := 1 to ct do
     begin
       if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
       begin
         Text := Items[i-1]; { update to the match that was found }
         ItemIndex := i-1;
         Found := True;
         Break;
       end;
     end;
     if Found then { select the untyped end of the string }
     begin
       SelStart := SelSt;
       SelLength := Length(Text) - SelSt;
     end
     else Beep;
   end;
end;

var
  CharSetList : TStringList;

procedure TForm1.AddCharacterSet(const S: string);
begin
  CharSetList.Add(S);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  CharSetList := TStringList.Create;
  Graphics.GetCharsetValues(AddCharacterSet);
  ComboBox1.Items := CharSetList;
end;
```

See Also

Ord (see page 22494)

158.3.35 System.Close Function

Terminates the association between a file variable and an external file.

Pascal

```
procedure Close(var F);
```

C++

```
Close( F);
```

File

System

Description

Close is provided for backward compatibility with existing code. When writing new applications in Delphi, use CloseFile (see page 22433) instead.

F is a file variable of any file type opened using Reset (see page 22511), Rewrite (see page 22513), or Append (see page 22415). The external file associated with F is completely updated and then closed, freeing the file handle for reuse.

See Also

Append (see page 22415)

AssignFile (see page 22420)

Reset (see page 22511)

Rewrite (see page 22513)

CloseFile (see page 22433)

158.3.36 System.CloseFile Function

Terminates the association between file variable and an external disk file.

Pascal

```
procedure CloseFile(var F);
```

C++

```
CloseFile( F);
```

File

System

Description

Due to naming conflicts, CloseFile replaces the Close (see page 22432) procedure. Use the CloseFile procedure instead of Close (see page 22432) to terminate the association between a file variable and an external disk file.

F is a file variable of any file type opened using Reset (see page 22511), Rewrite (see page 22513), or Append (see page 22415). The external file associated with F is completely updated and then closed, freeing the file handle for reuse.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 22482) to check for I/O errors.

Delphi Examples:

```
{
This example reads a file one character at a time and writes
it into a file of your chosing.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F1, F2: TextFile;
  Ch: Char;
begin
  if OpenDialog1.Execute then begin
```

```
      AssignFile(F1, OpenDialog1.Filename);
      Reset(F1);
      if SaveDialog1.Execute then begin
        AssignFile(F2, SaveDialog1.Filename);
        Rewrite(F2);
        while not Eof(F1) do
        begin
          Read(F1, Ch);
          Write(F2, Ch);
        end;
        CloseFile(F2);
      end;
      CloseFile(F1);
    end;
  end;
  {
  Use a TOpenDialog to select a file.  Place the first line of
  the file into a TEdit.
  }
  procedure TForm1.Edit1Click(Sender: TObject);
  var
    F: TextFile;
    S: string;
  begin
    if OpenDialog1.Execute then            { Display Open dialog box }
    begin
      AssignFile(F, OpenDialog1.FileName); { File selected in dialog }
      Reset(F);
      Readln(F, S);                        { Read first line of file }
      Edit1.Text := S;                     { Put string in a TEdit control }
      CloseFile(F);
    end;
  end;
  {
  Click the button to open a TOpenDialog, then select a file
  to delete.
  }

  procedure TForm1.Button1Click(Sender: TObject);
  var
    F: Textfile;
  begin
    OpenDialog1.Title := 'Delete File';
    if OpenDialog1.Execute then
    begin
      AssignFile(F, OpenDialog1.FileName);
      try
        Reset(F);
        if MessageDlg('Erase ' + OpenDialog1.FileName + '?',
          mtConfirmation, [mbYes, mbNo], 0) = mrYes then
        begin
          CloseFile(F);
          Erase(F);
        end;
      except
        on EInOutError do
          MessageDlg('File I/O error.', mtError, [mbOk], 0);
      end;
    end;
  end;
```

See Also

AssignFile (see page 22420)

158.3.37 System.CompToCurrency Function

Converts a Comp value to a Currency (see page 22145) value.

Pascal

```
function CompToCurrency(Value: Comp): Currency; cdecl;
```

C++

```
__cdecl Currency CompToCurrency(Comp Value);
```

File

System

Description

CompToCurrency converts a Comp value into a Currency (see page 22145) value containing the decimal representation of that number.

See Also

CompToDouble (see page 22435)

CurrencyToComp (see page 22441)

DoubleToComp (see page 22446)

158.3.38 System.CompToDouble Function

Converts a Comp value to a double value.

Pascal

```
function CompToDouble(Value: Comp): Double; cdecl;
```

C++

```
__cdecl Double CompToDouble(Comp Value);
```

File

System

Description

CompToDouble converts a Comp value into a double value containing the decimal representation of that number.

See Also

CompToCurrency (see page 22435)

CurrencyToComp (see page 22441)

DoubleToComp (see page 22446)

158.3.39 System.Concat Function

Concatenates two or more strings into one.

Pascal

```
function Concat(s1: string): string; overload;
function Concat(s1: string; s2: string): string; overload;
function Concat(s1: string; s2: string; s3: string): string; overload;
```

C++

```
AnsiString Concat(AnsiString s1);
AnsiString Concat(AnsiString s1, AnsiString s2);
AnsiString Concat(AnsiString s1, AnsiString s2, AnsiString s3);
```

File

System

Description

In Delphi code, use Concat to concatenate an arbitrary number of strings. Each parameter is a string-type expression. The result is the concatenation of all the string parameters.

The syntax shown here for Concat demonstrates that the procedure can take a variable number of arguments.

Using the plus (+) operator has the same effect on two strings as using the Concat function:

S := 'ABC' + 'DEF';

Tip: The plus operator is faster than Concat.

Delphi Examples:

```
{
This example requires a button and three text fields.
Select the button to concatenate the two texts into
the third.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S: string;
begin
  Edit3.Text := Concat(Edit1.Text, Edit2.Text);     { 'ABCDEF' }
end;
```

See Also

Copy (see page 22438)

Delete (see page 22443)

Insert (see page 22480)

Length (see page 22484)

Pos (see page 22497)

158.3.40 System.Continue Function

Allows the flow of control to proceed to the next iteration of for, while, or repeat statements.

Pascal

```
procedure Continue;
```

C++

```
Continue();
```

File

System

Description

In Delphi code, the Continue procedure causes the flow of control to proceed to the next iteration of the enclosing for, while, or repeat statement.

The compiler reports an error if a call to Continue isn't enclosed by a for, while, or repeat statement.

Note: Continue does not violate the flow of control dictated by a try..finally construct. If a call to Continue causes control to leave the try clause, the finally clause is entered.

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], 1);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode 1.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);
```

```
        AssignFile(F, FileListBox1.Items.Strings[i]);
        Reset(F, 1);
        ListBox1.Items.Add(
          FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
        CloseFile(F);
      end;
    finally
    { do something here }
    end;
  end;
end;
```

See Also

Break (see page 22426)

Exit (see page 22453)

Halt (see page 22475)

Abort

158.3.41 System.Copy Function

Returns a substring of a string or a segment of a dynamic array.

Pascal

```
function Copy(S; Index: Integer; Count: Integer): array; overload;
```

C++

```
array Copy( S, int Index, int Count);
```

File

System

Description

S is an expression of a string or dynamic-array type. Index and Count are integer-type expressions. Copy returns a substring or subarray containing Count characters or elements starting at S[Index]. The substring or subarray is a unique copy (that is, it does not share memory with S, although if the elements of the array are pointers or objects, these are not copied as well.)

If Index is larger than the length of S, Copy returns an empty string or array.

If Count specifies more characters or array elements than are available, only the characters or elements from S[Index] to the end of S are returned.

Note: When S is a dynamic array, you can omit the Index and Count parameters and Copy copies the entire array.

Delphi Examples:

```
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
```

```
  caught in the OnKeyDown event handler instead.
  Note: Set the TComboBox AutoComplete to false to turn off
  default auto completion.
  }
  procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
  var
    Found: boolean;
    i, ct, SelSt: Integer;
    TmpStr: string;
  begin
    { first, process the keystroke to obtain the current string }
    { This code requires all items in list to be uppercase}
    if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
    ct := (Sender as TComboBox).Items.Count;
    with (Sender as TComboBox) do
    begin
      SelSt := SelStart;
      if (Key = Chr(vk_Back)) and (SelLength <> 0) then
       TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
      else if Key = Chr(vk_Back) then {SelLength = 0}
       TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
      else {Key in ['A'..'Z', etc]}
       TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
      if TmpStr = '' then Exit;
      { update SelSt to the current insertion point }
      if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
      else if Key <> Chr(vk_Back) then Inc(SelSt);
      Key := #0; { indicate that key was handled }
      if SelSt = 0 then
      begin
        Text:= '';
        Exit;
      end;
     {Now that TmpStr is the currently typed string, see if we can locate a match }
      Found := False;
      for i := 1 to ct do
      begin
        if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
        begin
          Text := Items[i-1]; { update to the match that was found }
          ItemIndex := i-1;
          Found := True;
          Break;
        end;
      end;
      if Found then { select the untyped end of the string }
      begin
        SelStart := SelSt;
        SelLength := Length(Text) - SelSt;
      end
      else Beep;
    end;
  end;

  var
    CharSetList : TStringList;

  procedure TForm1.AddCharacterSet(const S: string);
  begin
    CharSetList.Add(S);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    CharSetList := TStringList.Create;
    Graphics.GetCharsetValues(AddCharacterSet);
```

```
  ComboBox1.Items := CharSetList;
end;
```

See Also

Concat (see page 22436)

Delete (see page 22443)

Insert (see page 22480)

Length (see page 22484)

Pos (see page 22497)

Slice (see page 22529)

LeftStr

RightStr

Parameters

158.3.42 System.Cos Function

Calculates the cosine of an angle.

Pascal

```
function Cos(const X: Extended): Extended;
```

C++

```
Extended Cos(const Extended X);
```

File

System

Description

In Delphi code, Cos returns the cosine of the angle X.

X is a real-type expression that represents an angle in radians.

Delphi Examples:

```
{
This example displays the Cosine of any value between 0 and
12.  Since the scrollbar position is an integer between the
scrollbar  min and max, multiply the range and divide the
position to fine tune it.  With the scrollbar selected use
the left and right buttons to change the position by 1E-6.
Or use the Dec function to decrement (or increment) by a
specific amount.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  new : Integer;
begin
  new := ScrollBar1.Position;
  System.Dec(new, Round(StrToFloat(Edit3.Text)*1000000));
  ScrollBar1.Position := new;
  Edit1.Text := FloatToStr(ScrollBar1.Position/1000000);
  Edit2.Text := FloatToStr(Cos(ScrollBar1.Position/1000000));
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ScrollBar1.Max := Round(Pi * 4000000);   // 4 * Pi
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  Edit1.Text := FloatToStr(ScrollPos/1000000);
  Edit2.Text := FloatToStr(Cos(ScrollPos/1000000));
end;
```

See Also

ArcCos

Sin (see page 22528)

Tan

Cosh

158.3.43 System.CurrencyToComp Function

Converts a Currency (see page 22145) value to a Comp value.

Pascal

```
procedure CurrencyToComp(Value: Currency; var Result: Comp); cdecl;
```

C++

```
__cdecl CurrencyToComp(Currency Value, Comp Result);
```

File

System

Description

CurrencyToComp converts a Currency (see page 22145) value into a Comp.

See Also

CompToDouble (see page 22435)

CompToCurrency (see page 22435)

DoubleToComp (see page 22446)

158.3.44 System.Dec Function

Decrements a variable by 1 or N.

Pascal

```
procedure Dec(var X); overload;
procedure Dec(var X; N: Longint); overload;
```

C++

```
Dec( X);
Dec( X, Longint N);
```

File

System

Description

In Delphi code, Dec subtracts one or N from a variable.

X is a variable of an ordinal type (including Int64), or a pointer type if the extended syntax is enabled.

N is an integer-type expression.

X decrements by 1, or by N if N is specified; that is, Dec(X) corresponds to the statement X := X - 1, and Dec(X, N) corresponds to the statement X := X - N. On some platforms, Dec may generate optimized code, especially useful in tight loops.

If X is a pointer type, it decrements X by N times the size of the type pointed to. Thus, given

```
type
  PMyType = ^TMyType;
```

and

```
var
  P: PMyType;
```

the statement `Dec(P)` decrements `P` by `SizeOf (` see page 22528`)(TMyType)`.

Warning: Do not use Dec on ordinal-type properties if the property uses a write procedure.

Delphi Examples:

```
{
This example displays the Cosine of any value between 0 and
12.  Since the scrollbar position is an integer between the
scrollbar  min and max, multiply the range and divide the
position to fine tune it.  With the scrollbar selected use
the left and right buttons to change the position by 1E-6.
Or use the Dec function to decrement (or increment) by a
specific amount.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  new : Integer;
begin
  new := ScrollBar1.Position;
  System.Dec(new, Round(StrToFloat(Edit3.Text)*1000000));
  ScrollBar1.Position := new;
  Edit1.Text := FloatToStr(ScrollBar1.Position/1000000);
  Edit2.Text := FloatToStr(Cos(ScrollBar1.Position/1000000));
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ScrollBar1.Max := Round(Pi * 4000000);   // 4 * Pi
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  Edit1.Text := FloatToStr(ScrollPos/1000000);
  Edit2.Text := FloatToStr(Cos(ScrollPos/1000000));
end;
```

See Also

Inc (see page 22478)

Pred (see page 22499)

Succ (see page 22532)

158.3.45 System.Delete Function

Removes a substring from a string.

Pascal

```
procedure Delete(var S: string; Index: Integer; Count: Integer);
```

C++

```
Delete(AnsiString S, int Index, int Count);
```

File

System

Description

In Delphi code, Delete removes a substring of Count characters from string S starting with S[Index]. S is a string-type variable. Index and Count are integer-type expressions.

If index is larger than the length of the string or less than 1, no characters are deleted.

If count specifies more characters than remain starting at the index, Delete removes the rest of the string. If count is less than or equal to 0, no characters are deleted.

Delphi Examples:

```
{
This example uses the Delete function to edit a string.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  s: string;
begin
  s := 'Honest Abe Lincoln';
  Canvas.TextOut(10, 10, s);
  Delete(s,8,4);
  Canvas.TextOut(10, 40, s);    { 'Honest Lincoln' }
end;
```

See Also

Concat (see page 22436)

Copy (see page 22438)

Insert (see page 22480)

Length (see page 22484)

Pos (see page 22497)

158.3.46 System.Dispose Function

Releases memory allocated for a dynamic variable.

Pascal

```
procedure Dispose(var P: Pointer);
```

C++

```
Dispose(void * P);
```

File

System

Description

Use Dispose in Delphi code to free the memory which a pointer addresses. After a call to Dispose, the value of P is undefined and it is an error to reference P.

Note: If compiling with {$I+}, use exceptions to handle this error. When an invalid pointer is passed to Dispose it raises an EInvalidPointer exception.

Delphi Examples:

```
{
The following code explicitly allocates memory for a pointer
in the OnCreate event of Form1, then releases the memory in
the OnDestroy event. Assume that MyPtr is a Pointer type
field of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  New(MyPtr);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Dispose(MyPtr);
end;
{
The following code adds an object to MyList if it isn't
already in the list.
}
type
  PMyList = ^AList;
  AList = record
    I: Integer;
    C: Char;
  end;
var
  Form1: TForm1;
  MyList: TList;
  ARecord1, ARecord2: PMyList;

implementation

{$R *.dfm}

procedure DisplayTList(TheList: TList);
var
```

```
    ARecord: PMyList;
    B: Byte;
    Y: Word;
  begin
      { Now paint the items onto the paintbox}
      Y := 10;             {Variable used in TextOut function}
      for B := 0 to (TheList.Count - 1) do
      begin
        ARecord := TheList.Items[B];
        Form1.PaintBox1.Canvas.TextOut(10, Y, IntToStr(ARecord^.I)); {Display I}
        Y := Y + 30;  {Increment Y Value again}
        Form1.PaintBox1.Canvas.TextOut(10, Y, ARecord^.C);  {Display C}
        Y := Y + 30;  {Increment Y Value}
      end;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    if (MyList.IndexOf(ARecord1) = -1) then MyList.Add(ARecord1);
    DisplayTList(MyList);
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    if (MyList.IndexOf(ARecord2) = -1) then MyList.Add(ARecord2);
    DisplayTList(MyList);
  end;

  procedure TForm1.Button3Click(Sender: TObject);
  var
    ARecord: PMyList;
    B: Integer;
  begin
    if (MyList.Count <> 0) then
    begin
      for B := (MyList.Count - 1) downto 0 do
      begin
        ARecord := MyList.Items[B];
        MyList.Remove(ARecord);
      end;
    end;
    Form1.PaintBox1.Repaint;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    MyList := TList.Create;
    New(ARecord1);
    ARecord1^.I := 100;
    ARecord1^.C := 'Z';
    New(ARecord2);
    ARecord2^.I := 200;
    ARecord2^.C := 'X';
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    Dispose(ARecord1);
    Dispose(ARecord2);
    MyList.Free;
  end;
```

See Also

EInvalidPointer

FreeMem (see page 22464)

GetMem (see page 22469)

New (see page 22491)

158.3.47 System.DoubleToComp Function

Converts a double value to a Comp.

Pascal

```
procedure DoubleToComp(Value: Double; var Result: Comp); cdecl;
```

C++

```
__cdecl DoubleToComp(Double Value, Comp Result);
```

File

System

Description

DoubleToComp converts a Double value into a Comp.

See Also

CompToDouble (see page 22435)

CompToCurrency (see page 22435)

CurrencyToComp (see page 22441)

158.3.48 System.DynArrayClear Function

Pascal

```
procedure DynArrayClear(var a: Pointer; typeInfo: Pointer);
```

C++

```
DynArrayClear(void * a, void * typeInfo);
```

File

System

Description

This is function System.DynArrayClear.

158.3.49 System.DynArrayDim Function

Pascal

```
function DynArrayDim(typeInfo: PDynArrayTypeInfo): Integer;
```

C++

```
int DynArrayDim(PDynArrayTypeInfo typeInfo);
```

File

System

Description

This is function System.DynArrayDim.

158.3.50 System.DynArraySetLength Function

Pascal

```
procedure DynArraySetLength(var a: Pointer; typeInfo: Pointer; dimCnt: Longint; lengthVec:
PLongint);
```

C++

```
DynArraySetLength(void * a, void * typeInfo, Longint dimCnt, PLongint lengthVec);
```

File

System

Description

This is function System.DynArraySetLength.

158.3.51 System.EndThread Function

Terminates the execution of a thread.

Pascal

```
procedure EndThread(ExitCode: Integer);
```

C++

```
EndThread(int ExitCode);
```

File

System

Description

EndThread terminates execution of a thread with the specified ExitCode (see page 22642).

Note: On Windows, EndThread calls the ExitThread WinAPI with the specified ExitCode (see page 22642).

See Also

BeginThread (see page 22422)

TThreadFunc (see page 22622)

158.3.52 System.EnumModules Function

Executes a callback for all the modules in an application.

Pascal

```
procedure EnumModules(Func: TEnumModuleFunc; Data: Pointer); overload;
procedure EnumModules(Func: TEnumModuleFuncLW; Data: Pointer); overload;
```

C++

```
EnumModules(TEnumModuleFunc Func, void * Data);
EnumModules(TEnumModuleFuncLW Func, void * Data);
```

File

System

Description

EnumModules enumerates the executable and all packages in an application by passing the handle of the executable and then of each package, in turn, to a user-defined callback function. EnumModules continues until the last module in the application is enumerated, or until the callback function returns false. The parameters to EnumModules areas follows:

Parameter	Meaning
Func	The procedure-instance address of the callback function.
Data	A 32-bit user-defined value that is passed to the callback function.

See Also

EnumResourceModules (see page 22448)

LibModuleList (see page 22650)

Hinstance

158.3.53 System.EnumResourceModules Function

Executes a callback for all resource modules in an application.

Pascal

```
procedure EnumResourceModules(Func: TEnumModuleFunc; Data: Pointer); overload;
procedure EnumResourceModules(Func: TEnumModuleFuncLW; Data: Pointer); overload;
```

C++

```
EnumResourceModules(TEnumModuleFunc Func, void * Data);
EnumResourceModules(TEnumModuleFuncLW Func, void * Data);
```

File

System

Description

EnumResourceModules executes the callback specified by the Func parameter for all of the instance handles associated with resources of the current program. The instance handle (HInstance) of each resource is passed to Func. EnumResourceModules continues until the last resource module in the application is enumerated, or until the callback returns false. The parameters to EnumResourceModules are as follows:

Parameter	Meaning
Func	The procedure-instance address of the callback function.
Data	A 32-bit user-defined value that is passed to the callback function.

See Also

EnumModules (see page 22447)

LibModuleList (see page 22650)

Hinstance

158.3.54 System.Eof Function

Tests whether the file position is at the end of a file.

Pascal

```
function Eof: Boolean; overload;
function Eof(var F): Boolean; overload;
function Eof(var F: Text): Boolean; overload;
```

C++

```
Boolean Eof();
Boolean Eof( F);
Boolean Eof(Text F);
```

File

System

Description

In Delphi code, Eof tests whether the current file position is the end-of-file. F is a file variable that has been opened for reading. If F is omitted, the standard file variable Input (see page 22647) is assumed.

Eof(F) returns true if the current file position is beyond the last character of the file or if the file is empty; otherwise, Eof(F) returns false.

Note: Eof fails if the file F has been opened in write-only mode. For example, you can't use Eof with files opened using the Append (see page 22415) or Rewrite (see page 22513), which open a file in write-only mode.

Delphi Examples:

```
{
This example reads a file one character at a time and writes
it into a file of your chosing.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F1, F2: TextFile;
  Ch: Char;
begin
  if OpenDialog1.Execute then begin
    AssignFile(F1, OpenDialog1.Filename);
    Reset(F1);
    if SaveDialog1.Execute then begin
      AssignFile(F2, SaveDialog1.Filename);
      Rewrite(F2);
      while not Eof(F1) do
      begin
        Read(F1, Ch);
        Write(F2, Ch);
      end;
```

```
      CloseFile(F2);
    end;
    CloseFile(F1);
  end;
end;
```

See Also

DeleteFile

Eoln (see page 22450)

SeekEof (see page 22519)

SetLineBreakStyle (see page 22522)

158.3.55 System.Eoln Function

Tests whether the file pointer is at the end of a line.

Pascal

```
function Eoln: Boolean; overload;
function Eoln(var F: Text): Boolean; overload;
```

C++

```
Boolean Eoln();
Boolean Eoln(Text F);
```

File

System

Description

In Delphi code, Eoln tests whether the current file position is the end-of-line of a text file. F, if specified, is a text file variable. If F is omitted, the standard file variable Input (see page 22647) is assumed.

Eoln(F) returns true if the current file position is at an end-of-line or if Eof (see page 22449)(F) is true; otherwise, Eoln(F) returns false.

Delphi Examples:

```
{
This example waits for keyboard input before continuing.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   WriteLn('Type something on the console and hit return');
   Eoln;
   WriteLn('Write this after you type something');
   Readln;
end;
```

See Also

Eof (see page 22449)

SeekEoln (see page 22520)

SetLineBreakStyle (see page 22522)

158.3.56 System.Erase Function

Deletes an external file.

Pascal

```
procedure Erase(var F);
```

C++

```
Erase( F);
```

File

System

Description

In Delphi code, Erase deletes the external file associated with F. F is a file variable of any file type.

Always close a file before erasing it.

Delphi Examples:

```
{
Click the button to open a TOpenDialog, then select a file
to delete.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  F: Textfile;
begin
  OpenDialog1.Title := 'Delete File';
  if OpenDialog1.Execute then
  begin
    AssignFile(F, OpenDialog1.FileName);
    try
      Reset(F);
      if MessageDlg('Erase ' + OpenDialog1.FileName + '?',
        mtConfirmation, [mbYes, mbNo], 0) = mrYes then
      begin
        CloseFile(F);
        Erase(F);
      end;
    except
      on EInOutError do
        MessageDlg('File I/O error.', mtError, [mbOk], 0);
    end;
  end;
end;
```

See Also

Rename (see page 22510)

158.3.57 System.Error Function

Error is used to generate a run-time exception.

Pascal

```
procedure Error(errorCode: TRuntimeError);
```

C++

```
Error(TRuntimeError errorCode);
```

File

System

Description

Error should be invoked when you want to generate an exception. **errorCode** defines the type of error to be reported.

See Also

TRunTimeError

158.3.58 System.ExceptAddr Function

Returns the address at which the current exception was raised.

Pascal

```
function ExceptAddr: Pointer;
```

C++

```
void * ExceptAddr();
```

File

System

Description

Use ExceptAddr to obtain the address at which an exception was raised. If there is no current exception, ExceptAddr returns nil (Delphi) or NULL (C++).

See Also

ExceptionErrorMessage

ExceptObject (see page 22452)

158.3.59 System.ExceptObject Function

Returns current exception object.

Pascal

```
function ExceptObject: TObject;
```

C++

```
TObject * ExceptObject();
```

File

System

Description

ExceptObject returns the current exception object. This is the Exception object that represents the exception currently being handled. If no exception is currently being handled, ExceptObject returns nil (Delphi) or NULL (C++).

ExceptObject is useful when the exception variable (declared in the Delphi on block or C++ catch statement) is not accessible. This can occur when the exception handler calls a procedure, putting the exception variable is out of scope. This is also true in the Delphi else exception handler, which cannot declare an exception variable.

Note: The object returned by ExceptObject may not exist after the exception handler finishes. Thus you cannot use ExceptObject to retain a reference to the exception object, or to re-raise the exception. If you need to do either of these things, call AcquireExceptionObject (see page 22410).

See Also

ExceptionErrorMessage

Exception

on

catch

158.3.60 System.Exclude Function

Removes an element from a Delphi set or removes an event handler in .NET applications.

Pascal

```
procedure Exclude(var S: set of T; I: T);
```

C++

```
Exclude(set of T S, T I);
```

File

System

Description

When the first argument of Include (see page 22479) is a set type variable, the Exclude procedure removes an element from the set. If S is a set type variable, and I is an expression of a type compatible with the base type of S. then `Exclude(S,I)` corresponds to `S := S - [I]` but the Exclude procedure generates more efficient code.

On the .NET platform, Exclude is overloaded. When the first argument of Exclude is a multicast event and the second argument is an event handler, Include (see page 22479) removes the event handler from the event's listeners.

See Also

Include (see page 22479)

158.3.61 System.Exit Function

Exits from the current procedure.

Pascal

```
procedure Exit;
```

C++

```
Exit();
```

File

System

Description

In Delphi, the Exit procedure immediately passes control away from the current procedure. If the current procedure is the main program, Exit causes the program to terminate.

Exit will cause the calling procedure to continue with the statement after the point at which the procedure was called.

Note: Exit passes control away from the current procedure, not merely the current block. But Exit does not violate the flow of control dictated by a try..finally construct; if Exit is called inside the try clause, the finally clause is still executed.

Delphi Examples:

```
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
Note: Set the TComboBox AutoComplete to false to turn off
default auto completion.
}
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
var
  Found: boolean;
  i, ct, SelSt: Integer;
  TmpStr: string;
begin
  { first, process the keystroke to obtain the current string }
  { This code requires all items in list to be uppercase}
  if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
  ct := (Sender as TComboBox).Items.Count;
  with (Sender as TComboBox) do
  begin
    SelSt := SelStart;
    if (Key = Chr(vk_Back)) and (SelLength <> 0) then
     TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
    else if Key = Chr(vk_Back) then {SelLength = 0}
     TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
    else {Key in ['A'..'Z', etc]}
     TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
    if TmpStr = '' then Exit;
    { update SelSt to the current insertion point }
    if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
    else if Key <> Chr(vk_Back) then Inc(SelSt);
    Key := #0; { indicate that key was handled }
    if SelSt = 0 then
    begin
      Text:= '';
      Exit;
    end;
```

```
    {Now that TmpStr is the currently typed string, see if we can locate a match }
     Found := False;
     for i := 1 to ct do
     begin
       if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
       begin
         Text := Items[i-1]; { update to the match that was found }
         ItemIndex := i-1;
         Found := True;
         Break;
       end;
     end;
     if Found then { select the untyped end of the string }
     begin
       SelStart := SelSt;
       SelLength := Length(Text) - SelSt;
     end
     else Beep;
   end;
 end;

 var
   CharSetList : TStringList;

 procedure TForm1.AddCharacterSet(const S: string);
 begin
   CharSetList.Add(S);
 end;

 procedure TForm1.FormCreate(Sender: TObject);
 begin
   CharSetList := TStringList.Create;
   Graphics.GetCharsetValues(AddCharacterSet);
   ComboBox1.Items := CharSetList;
 end;
```

See Also

Break (see page 22426)

Halt (see page 22475)

Continue (see page 22437)

158.3.62 System.Exp Function

Returns the exponential of X.

Pascal

```
function Exp(X: Real): Real;
function Exp(const X: Extended): Extended;
```

C++

```
Real Exp(Real X);
Extended Exp(const Extended X);
```

File

System

Description

In Delphi code, Exp returns the value of e raised to the power of X, where e is the base of the natural logarithms.

Delphi Examples:

```
{
This example calculated the exponential of 1.0 and then calculates
the natural log of that quantity to show that the natural log
function is the inverse of the exponential function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   e : real;
   S : string;
begin
   e := Exp(1.0);
   Str(ln(e):3:2, S);
   S := 'e = ' + FloatToStr(e) + '; ln(e) = ' + S;
   Canvas.TextOut(10, 10, S);
end;
```

See Also

Ln (see page 22485)

158.3.63 System.FilePos Function

Returns the current file position.

Pascal

```
function FilePos(var F): Longint;
```

C++

```
Longint FilePos( F);
```

File

System

Description

In Delphi code, use FilePos on the file variable of an open file to determine the current position in the file. If the current position in the file is at the very beginning, FilePos returns 0. Otherwise, FilePos returns the byte offset from the beginning of the current position.

FilePos will not work on a file that is not open.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 22482) to check for I/O errors.

Delphi Examples:

```
{
FileSize, Seek, FilePos example
This example displays a open dialog when the user clicks a button. When the user selects the
file, it reports the size of the file, seeks to halfway through the file, and reports the file
position at that point.
Note:   This example assumes the compiler flag $IOCHECKS ON ($I+). That is, it expects errors
to generate exceptions rather than checking IOResult to be sure that I/O routines succeed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   f: file of Byte;
   size: Longint;
```

```
    S: string;
    y: Integer;
begin
  if OpenDialog1.Execute then
  begin
    AssignFile(f, OpenDialog1.FileName);
    Reset(f);
    try
      size := FileSize(f);
      S := 'File size in bytes: ' + IntToStr(size);
      y := 10;
      Canvas.TextOut(5, y, S);
      y := y + Canvas.TextHeight(S) + 5;
      S := 'Seeking halfway into file...';
      Canvas.TextOut(5, y, S);
      y := y + Canvas.TextHeight(S) + 5;
      Seek(f, size div 2);
      S := 'Position is now ' + IntToStr(FilePos(f));
      Canvas.TextOut(5, y, S);
    finally
      CloseFile(f);
    end;
  end;
end;
```

See Also

FileSize (see page 22457)

Seek (see page 22518)

158.3.64 System.FileSize Function

Returns the number of records in a file.

Pascal

```
function FileSize(var F): Integer;
```

C++

```
int FileSize( F);
```

File

System

Description

In Delphi code, call FileSize to determine the size of the file specified by the file variable F. The size is expressed as the number of records in a record file. Thus:

If the file is declared as a file of byte, then the record size defaults to one byte, and FileSize returns the number of bytes in the file.

The Reset (see page 22511) procedure can set the record size (in bytes) when it opens the file. In this case, FileSize returns the number of records in the file.

Note: If the file is declared as an untyped file and you don't specify a record size when you call Reset (see page 22511), then FileSize assumes a record size of 128. That is, FileSize gives the number of bytes divided by 128.

To use FileSize, the file must be open. If the file is empty, FileSize(F) returns 0.

Delphi Examples:

```
{
FileSize, Seek, FilePos example
This example displays a open dialog when the user clicks a button. When the user selects the
file, it reports the size of the file, seeks to halfway through the file, and reports the file
position at that point.
Note:   This example assumes the compiler flag $IOCHECKS ON ($I+). That is, it expects errors
to generate exceptions rather than checking IOResult to be sure that I/O routines succeed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   f: file of Byte;
   size: Longint;
   S: string;
   y: Integer;
begin
  if OpenDialog1.Execute then
  begin
    AssignFile(f, OpenDialog1.FileName);
    Reset(f);
    try
      size := FileSize(f);
      S := 'File size in bytes: ' + IntToStr(size);
      y := 10;
      Canvas.TextOut(5, y, S);
      y := y + Canvas.TextHeight(S) + 5;
      S := 'Seeking halfway into file...';
      Canvas.TextOut(5, y, S);
      y := y + Canvas.TextHeight(S) + 5;
      Seek(f, size div 2);
      S := 'Position is now ' + IntToStr(FilePos(f));
      Canvas.TextOut(5, y, S);
    finally
      CloseFile(f);
    end;
  end;
end;
{
The following example opens a file of your choice and reads
the entire file into a dynamically allocated buffer. The
buffer and the size of the file are then passed to a routine
that processes the text, and finally the dynamically
allocated buffer is freed and the file is closed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: file;
  Size: Integer;
  Buffer: PChar;
begin
  if OpenDialog1.Execute then
  begin
    AssignFile(F, OpenDialog1.FileName);
    Reset(F, 1);
    try
      Size := FileSize(F);
      GetMem(Buffer, Size);
      try
        BlockRead(F, Buffer^, Size);
        Memo1.Lines.Add(Buffer);
      finally
        FreeMem(Buffer);
      end;
    finally
      CloseFile(F);
    end;
  end;
```

```
end;
```

See Also

FilePos (see page 22456)

Reset (see page 22511)

158.3.65 System.FillChar Function

Fills contiguous bytes with a specified value.

Pascal

```
procedure FillChar(var X; Count: Integer; Value: Byte);
```

C++

```
FillChar( X, int Count, Byte Value);
```

File

System

Description

In Delphi, FillChar fills Count contiguous bytes (referenced by X) with the value specified by Value (Value can be type Byte or Char).

Warning: This function does not perform any range checking.

Delphi Examples:

```
{
This example creates a rotated font and displays it on the
form by assigning the handle of the rotated to the Form's
Canvas Font via the Font's Handle property.  Rotating fonts
is a straightforward process, so long as the Windows Font
Mapper can supply a rotated font based on the font you
request. If you are using a TrueType font there is virtually
no reason for failure.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  lf: LOGFONT; // Windows native font structure
begin
  Canvas.Brush.Style := bsClear; // set the brush style to transparent
  FillChar(lf, SizeOf(lf), Byte(0));
  lf.lfHeight := 20;
  lf.lfEscapement := 10 * 45; // degrees to rotate
  lf.lfOrientation := 10 * 45;
  lf.lfCharSet := DEFAULT_CHARSET;
  StrCopy(lf.lfFaceName, 'Tahoma');

  Canvas.Font.Handle := CreateFontIndirect(lf);
  Canvas.TextOut(10, 100, 'Rotated text'); // output the rotated font
end;
{
This example fills a char array with characters and then
assigns the array to a text edit.
}
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
  S: array[0..79] of char;
begin
  { Set to all Xs }
  FillChar(S, SizeOf(S), Ord('X'));
  Edit1.Text := S;
end;
```

See Also

Move (see page 22490)

StringOfChar (see page 22531)

158.3.66 System.Finalize Function

Uninitializes a dynamically allocated variable.

Pascal

```
procedure Finalize(var V); overload;
procedure Finalize(var V; Count: Integer); overload;
```

C++

```
Finalize( V);
Finalize( V, int Count);
```

File

System

Description

Finalize should be used only in Delphi code where a dynamically allocated variable is deallocated by other means than the Dispose (see page 22444) procedure. Dynamic arrays can never be deallocated using the Dispose (see page 22444) procedure, but can be freed by passing them to Finalize.

For global variables, local variables, objects, and dynamic variables deallocated using Dispose (see page 22444), the compiler generates code that finalizes all long strings, variants, and interfaces contained by the variable when the instance is destroyed.

If a dynamic variable meets the following two conditions, a call to Finalize is required to finalize the variable before it can be deallocated.

The variable is deallocated by other means than the Dispose (see page 22444) standard procedure (for example using FreeMem (see page 22464)).

The variable contains long strings, variants, or interfaces, not all of which are empty or Unassigned.

Finalize simply sets all long strings to empty and all variants and interfaces to Unassigned, thus properly releasing any memory that was referenced by the long strings and variants.

In cases where several variables are deallocated in a contiguous memory block such as a dynamically allocated array of strings, the additional Count parameter can be specified to finalize all variables in one operation.

If the variable specified in a call to Finalize contains no long strings, variants, or interfaces, the compiler eliminates the call and generates no code for it.

See Also

Initialize (see page 22480)

158.3.67 System.FindClassHInstance Function

Returns the instance handle for the module in which a class type is defined.

Pascal

```
function FindClassHInstance(ClassType: TClass): LongWord;
```

C++

```
LongWord FindClassHInstance(TClass ClassType);
```

File

System

Description

FindClassHInstance finds the HInstance of the executable or library that "owns", or contains, the runtime type information (RTTI) and the Virtual Method Table (VMT) for the specified class passed in as the Class Type parameter. When reading a component (which is a type), this information is needed in order to be able to load the form file (resources) associated with that component. The component streaming system uses FindClassHInstance for this purpose.

FindClassHInstance returns 0 if the instance is not found. Otherwise it returns the class owner's HInstance.

See Also

FindResourceHInstance (see page 22462)

HInstance

158.3.68 System.FindHInstance Function

Returns the instance handle of the module that contains a specified address.

Pascal

```
function FindHInstance(Address: Pointer): LongWord;
```

C++

```
LongWord FindHInstance(void * Address);
```

File

System

Description

Use FindHInstance to obtain the HInstance of the module that "owns", or contains, the address specified by the Address parameter. If no HInstance is found for that address, FindHInstance returns 0.

See Also

FindClassHInstance (see page 22461)

FindResourceHInstance (see page 22462)

MainInstance (see page 22650)

HInstance

158.3.69 System.FindResourceHInstance Function

Returns the instance handle of the resource module associated with a specified HInstance.

Pascal

```
function FindResourceHInstance(Instance: LongWord): LongWord;
```

C++

```
LongWord FindResourceHInstance(LongWord Instance);
```

File

System

Description

FindResourceHInstance finds and returns the instance handle of the resource module associated with the executable, library, or package identified by the Instance parameter.

FindResourceHInstance returns 0 upon failure.

Note: The Instance parameter must be a valid instance handle.

See Also

FindClassHInstance (see page 22461)

FindHInstance (see page 22461)

158.3.70 System.Flush Function

Empties the buffer of a text file opened for output.

Pascal

```
function Flush(var t: Text): Integer;
```

C++

```
int Flush(Text t);
```

File

System

Description

F is a text file variable.

When a text file is opened for output using Rewrite (see page 22513) or Append (see page 22415), Flush empties the file's buffer. This guarantees that all characters written to the file at that time have actually been written to the external file. Flush has no effect on files opened for input.

Flush returns 0 if the operation succeeded. Otherwise, it returns the error code. When compiled using the {$I-} flag, the IOResult (see page 22482) method returns this value.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 22482) to check for I/O errors.

Delphi Examples:

```
{
Append, Flush example
}

procedure TForm1.Button1Click(Sender: TObject);
var
  f: TextFile;
begin
  if OpenDialog1.Execute then
  begin                    { open a text file }
    AssignFile(f, OpenDialog1.FileName);
    Append(f);
    Writeln(f, 'I am appending some stuff to the end of the file.');
    { insert code here that would require a Flush before closing the file }
    Flush(f);  { ensures that the text was actually written to file }
    CloseFile(f);
    MessageDlg(OpenDialog1.FileName + ' has been altered, please validate.', mtInformation,
[mbOK], 0)
  end;
end;
```

See Also

Append (see page 22415)

Rewrite (see page 22513)

158.3.71 System.FPower10 Function

Pascal

```
procedure FPower10;
```

C++

```
FPower10();
```

File

System

Description

This is function System.FPower10.

158.3.72 System.Frac Function

Returns the fractional part of a real number.

Pascal

```
function Frac(const X: Extended): Extended;
```

C++

```
Extended Frac(const Extended X);
```

File

System

Description

In Delphi code, the Frac function returns the fractional part of the argument X.

X is a real-type expression. The result is the fractional part of X; that is, Frac(X) = X - Int (see page 22481)(X).

Delphi Examples:

```
{
This example illustrates the use of the System Frac function.
The number in the text edit can be a negative or positive
real number.
Notice how the type passed (R) matters.
}
procedure TForm1.Button1Click(Sender: TObject);
var
 R: Extended;
begin
  R := StrToFloat(Edit1.Text);
  ListBox1.Items.Add(FloatToStr(Frac(R)));
end;

procedure TForm1.Button2Click(Sender: TObject);
var
 R: Real;
begin
  R := StrToFloat(Edit1.Text);
  ListBox1.Items.Add(FloatToStr(Frac(R)));
end;
```

See Also

Int (see page 22481)

158.3.73 System.FreeMem Function

FreeMem frees a memory block.

Pascal

```
procedure FreeMem(var P: Pointer); overload;
procedure FreeMem(var P: Pointer; Size: Integer); overload;
```

C++

```
FreeMem(void * P);
FreeMem(void * P, int Size);
```

File

System

Description

GetMem (see page 22469) allocates a block on the heap. To dispose of the buffer, use FreeMem.

Note: FreeMemory (see page 22466) is the C++ compatible version of FreeMem.

Delphi Examples:

```
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
```

```
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  iFileHandle: Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PChar;
  i: Integer;
begin
  if OpenDialog1.Execute then
  begin
    try
      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      Buffer := PChar(System.AllocMem(iFileLength + 1));
      iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
      FileClose(iFileHandle);
      for i := 0 to iBytesRead-1 do
      begin
        StringGrid1.RowCount := StringGrid1.RowCount + 1;
        StringGrid1.Cells[1,i+1] := Buffer[i];
        StringGrid1.Cells[2,i+1] := IntToStr(Integer(Buffer[i]));
      end;
    finally
      FreeMem(Buffer);
    end;
  end;
end;
{
The following example opens a file of your choice and reads
the entire file into a dynamically allocated buffer. The
buffer and the size of the file are then passed to a routine
that processes the text, and finally the dynamically
allocated buffer is freed and the file is closed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: file;
  Size: Integer;
  Buffer: PChar;
begin
  if OpenDialog1.Execute then
  begin
    AssignFile(F, OpenDialog1.FileName);
    Reset(F, 1);
    try
      Size := FileSize(F);
      GetMem(Buffer, Size);
      try
        BlockRead(F, Buffer^, Size);
        Memo1.Lines.Add(Buffer);
      finally
        FreeMem(Buffer);
      end;
    finally
      CloseFile(F);
    end;
  end;
end;
```

See Also

AllocMem (see page 22413)

FreeMemory (see page 22466)

GetMem (see page 22469)

GetMemory (see page 22470)

EOutOfMemory (see page 22742)

ReallocMem (see page 22507)

ReallocMemory (see page 22508)

158.3.74 System.FreeMemory Function

FreeMemory frees a memory block.

Pascal

```
function FreeMemory(P: Pointer): Integer; cdecl;
```

C++

```
__cdecl int FreeMemory(void * P);
```

File

System

Description

GetMemory (see page 22470) allocates a block on the heap. To dispose of the buffer, use FreeMemory.

Note: FreeMemory is the C++ compatible version of FreeMem (see page 22464).

See Also

AllocMem (see page 22413)

FreeMem (see page 22464)

GetMem (see page 22469)

GetMemory (see page 22470)

EOutOfMemory (see page 22742)

ReallocMem (see page 22507)

ReallocMemory (see page 22508)

158.3.75 System.Get8087CW Function

Returns the value of the 8087 code word.

Pascal

```
function Get8087CW: Word;
```

C++

```
Word Get8087CW();
```

File

System

Description

Use Get8087CW to determine the current value of the floating-point unit control word. This control word controls the precision of floating point calculations, the rounding mode, and whether certain floating-point operations trigger exceptions. See Intel's processor documentation for details.

See Also

NoErrMsg (see page 22652)

SetExceptionMask

SetPrecisionMode

SetRoundMode

Set8087CW (see page 22520)

158.3.76 System.GetDir Function

Returns the current directory.

Pascal

```
procedure GetDir(D: Byte; var S: string);
```

C++

```
GetDir(Byte D, AnsiString S);
```

File

System

Description

Note: On Windows, GetDir sets the S parameter to the current directory on the drive specified by the D parameter. D can be set to any of the following values:

Value	Drive
0	Default
1	A
2	B
3	C

Note: GetDir performs no error checking. For example, if the invalid value 5 (J) were specified, S would return J:\ as if it were the root directory of the invalid drive.

Note: On Linux, GetDir returns the current working directory using the S parameter. The first parameter is ignored.

Delphi Examples:

```
{
This example reports the current directory and drive.  Set
Help on GetDir for reporting information on other drives.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  s : string;
begin
  GetDir(0,s); { 0 = Current drive }
  MessageDlg('Current drive and directory: ' + s, mtInformation, [mbOk] , 0);
end;
```

See Also

ChDir (see page 22429)

MkDir (see page 22488)

RmDir (see page 22515)

DirectoryExists

CreateDir

GetCurrentDir

158.3.77 System.GetHeapStatus Function

Returns the current status of the memory manager.

Pascal

```
function GetHeapStatus: THeapStatus; deprecated; platform;
```

C++

```
THeapStatus GetHeapStatus();
```

File

System

Description

GetHeapStatus returns the current status of the memory manager in a THeapStatus (see page 22558) type. If the ShareMem unit is not included in the current project, GetHeapStatus returns information about the global heap. If the ShareMem unit is included, GetHeapStatus returns information about memory that can be shared between between the current project and other binary modules (such as DLLs) that also make use of the ShareMem unit. GetHeapStatus can only return useful information if the built-in memory manager is used. For example, when using CodeGuard, the memory manager is replaced and GetHeapStatus always returns nil (Delphi) or NULL (C++).

If you want information about the memory manager that is shared between modules, make sure to call the function GetHeapStatus provided in the ShareMem unit. The function GetHeapStatus in the System (see page 22079) unit returns no values for a project that uses shared memory.

Warning: ShareMem must be the first unit included in the project file to prevent mixing of memory management models.

158.3.78 System.GetLastError Function

Returns the last error reported by an operating system API call.

Pascal

```
function GetLastError: Integer; stdcall;
```

C++

```
__stdcall int GetLastError();
```

File

System

Description

GetLastError returns the last error reported by an API call into the operating system. Calling this function usually resets the operating system error state.

See Also

Win32Check

158.3.79 System.GetMem Function

GetMem allocates a memory block.

Pascal

```
procedure GetMem(var P: Pointer; Size: Integer);
```

C++

```
GetMem(void * P, int Size);
```

File

System

Description

GetMem allocates a block of the given **Size** on the heap, and returns the address of this memory in parameter **P**. The bytes of the allocated buffer are not set to zero. To dispose of the buffer, use FreeMem (see page 22464). If there isn't enough memory available to allocate the block, an EOutOfMemory exception is raised.

Note: If the memory needs to be zero-initialized, use AllocMem (see page 22413) instead.

Delphi Examples:

```
{
The following example opens a file of your choice and reads
the entire file into a dynamically allocated buffer. The
buffer and the size of the file are then passed to a routine
that processes the text, and finally the dynamically
allocated buffer is freed and the file is closed.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  F: file;
  Size: Integer;
  Buffer: PChar;
begin
  if OpenDialog1.Execute then
  begin
    AssignFile(F, OpenDialog1.FileName);
    Reset(F, 1);
    try
      Size := FileSize(F);
      GetMem(Buffer, Size);
      try
        BlockRead(F, Buffer^, Size);
        Memo1.Lines.Add(Buffer);
      finally
        FreeMem(Buffer);
      end;
    finally
      CloseFile(F);
    end;
  end;
end;
```

See Also

AllocMem (see page 22413)

FreeMem (see page 22464)

FreeMemory (see page 22466)

GetMemory (see page 22470)

EOutOfMemory (see page 22742)

ReallocMem (see page 22507)

ReallocMemory (see page 22508)

158.3.80 System.GetMemory Function

GetMemory allocates a memory block.

Pascal

```
function GetMemory(Size: Integer): Pointer; cdecl;
```

C++

```
__cdecl void * GetMemory(int Size);
```

File

System

Description

GetMemory allocates a block of the given **Size** on the heap, and returns the address of this memory. The bytes of the allocated buffer are not set to zero. To dispose of the buffer, use FreeMem (see page 22464). If there isn't enough memory available to allocate the block, an EOutOfMemory exception is raised.

Note: If the memory needs to be zero-initialized, use AllocMem (see page 22413) instead.

Note: GetMemory is the C++ compatible version of GetMem (see page 22469).

Delphi Examples:

```
{
This example requires a button, a test edit, and a populated
ClientDataSet.  Pipe the ClientDataSet through a DataSource
to a DGGrid or DBNavigator to control the current field.
Cast the data correctly according to the field type when
assigning to the test edit.
}

{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button1Click(Sender: TObject);
var MyBuffer: Pointer;
begin
{ Retrieve the "raw" data from Field1 }
with CDS.Fields[0] do
  begin
    if not IsBlob then { this does not work for BLOB fields }
    begin
      { Allocate space }
      MyBuffer:= GetMemory(DataSize);
      try
        if not GetData(MyBuffer) then
          MessageDlg(DisplayName + ' is NULL', mtInformation, [mbOK], 0)
        else
          { Do something with the data };
          Edit1.Text:= string(MyBuffer); // for a stringfield
      finally
        { Free the space }
        FreeMem(MyBuffer, DataSize);
      end;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;
```

See Also

AllocMem (see page 22413)

FreeMem (see page 22464)

FreeMemory (see page 22466)

GetMem (see page 22469)

EOutOfMemory (see page 22742)

ReallocMem (see page 22507)

ReallocMemory (see page 22508)

158.3.81 System.GetMemoryManager Function

Returns the entry points of the currently installed memory manager.

Pascal

```
procedure GetMemoryManager(var MemMgr: TMemoryManager); overload; deprecated;
procedure GetMemoryManager(var MemMgrEx: TMemoryManagerEx); overload;
```

C++

```
GetMemoryManager(TMemoryManager MemMgr);
GetMemoryManager(TMemoryManagerEx MemMgrEx);
```

File

System

Description

GetMemoryManager returns the entry points of the currently installed memory manager in a TMemoryManager (see page 22562) record.

See Also

SetMemoryManager (see page 22523)

IsMemoryManagerSet (see page 22483)

158.3.82 System.GetMemoryManagerState Function

Fetches state information for the Memory Manager.

Pascal

```
procedure GetMemoryManagerState(var AMemoryManagerState: TMemoryManagerState);
```

C++

```
GetMemoryManagerState(TMemoryManagerState AMemoryManagerState);
```

File

System

Description

GetMemoryManagerState returns a structure containing memory usage statistics for each block type - small, medium and large blocks. In the current implementation, small blocks are blocks that are smaller than roughly 2.5K, large blocks are blocks that are larger than roughly 256K, and medium blocks are the sizes in between. It provides additional detail for the small block type (which is usually the most frequently used block type) down to each individual block size.

The structure returned by this procedure makes a distinction between the reserved address space and the allocated address space for each block type. The allocated address space is that part of the address space that is currently in use by the application, i.e. memory that was allocated through GetMem (see page 22469) and not yet freed. The reserved address space is the amount of address space that was obtained from the operating system for current and future use by the application, i.e. the allocated address space plus all overhead.

See Also

Memory Management

Configuring the Memory Manager

Monitoring the Memory Manager

158.3.83 System.GetMemoryMap Function

Fetches the memory map.

Pascal

```
procedure GetMemoryMap(var AMemoryMap: TMemoryMap);
```

C++

```
GetMemoryMap(TMemoryMap AMemoryMap);
```

File

System

Description

GetMemoryMap returns a map of the address space of the process. For the purposes of the map, the entire 4GB address space is broken down into 64K chunks. There are 65536 such chunks. GetMemoryMap obtains a status value for each of them in the TMemoryMap (see page 22617) array. The sum of the number of blocks with the status of **csUnallocated** multiplied by the chunk size (64K) gives an indication of how much more memory the application can allocate before it will run out of address space.

Status Value	Description
csUnallocated	Free
csAllocated	In use by the process
csReserved	Reserved for future use by the process
csSysAllocated	In use by the operating system
csSysReserved	Reserved for future use by the operating system

See Also

Memory Management

Configuring the Memory Manager

Monitoring the Memory Manager

158.3.84 System.GetMinimumBlockAlignment Function

Gets the minimum memory block alignment.

Pascal

```
function GetMinimumBlockAlignment: TMinimumBlockAlignment;
```

C++

```
TMinimumBlockAlignment GetMinimumBlockAlignment();
```

File

System

Description

GetMinimumBlockAlignment fetches the Memory Manager's minimum block alignment.

See Also

Memory Management

Configuring the Memory Manager

158.3.85 System.GetModuleFileName Function

Returns the fully qualified name for a module, given its handle.

Pascal

```
function GetModuleFileName(Module: HMODULE; Buffer: PChar; BufLen: Integer): Integer;
```

C++

```
int GetModuleFileName(HMODULE Module, const char * Buffer, int BufLen);
```

File

System

Description

Call GetModuleFileName to obtain the fully qualified name of a module, given its handle. The version of GetModuleFileName defined in the System (see page 22079) unit is available only on Linux. On Windows, use the Windows API with the same name instead.

Module is the handle of the module whose file name is required.

Buffer is a buffer that receives the file name.

BufLen is the number of bytes in Buffer.

GetModuleFileName returns the number of bytes written to Buffer.

Warning: GetModuleFileName is not always successful. Depending on the way the application is called, the presence of the /proc file system, and other factors, there are times when GetModuleFileName may return a name that is not fully qualified or in some cases, no file name at all. Be sure to check the results of this function before using the returned value.

See Also

GetModuleName

158.3.86 System.GetVariantManager Function

Returns the entry points of the routines that define variant behavior.

Pascal

```
procedure GetVariantManager(var VarMgr: TVariantManager); deprecated;
```

C++

```
GetVariantManager(TVariantManager VarMgr);
```

File

System

Description

GetVariantManager returns the entry points of the routines that perform variant type conversion and define other variant behavior.

Note: Although GetVariantManager is defined in System.hpp, do not use this procedure in C++ code. Working directly with the Variant (see page 22307) manger is not guaranteed to work in C++.

See Also

SetVariantManager (see page 22527)

IsVariantManagerSet (see page 22483)

158.3.87 System.Halt Function

Initiates abnormal termination of a program.

Pascal

```
procedure Halt; overload;
procedure Halt(Exitcode: Integer); overload;
```

C++

```
Halt();
Halt(int Exitcode);
```

File

System

Description

Halt performs an abnormal termination of a program and returns to the operating system.

To perform a normal termination of a Delphi application, call the Terminate method on the global Application object. If the application does not use a unit that provides an Application object, call the Exit (see page 22453) procedure from the main Program block.

Exitcode is an optional expression that specifies an exit code for the program.

Delphi Examples:

```
{
This example halts before the TextOut and terminates the
application.  Do not expect to one step this procedure.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   if 1 = 1 then
     begin
       if 2 = 2 then
```

```
          begin
            if 3 = 3 then
              begin
                Halt(1); { Halt right here! }
              end;
          end;
      end;
    Canvas.TextOut(10, 10, 'This will not be executed');
  end;
```

158.3.88 System.Hi Function

Returns the high-order byte of X as an unsigned value.

Pascal

```
function Hi(X): Byte;
```

C++

```
Byte Hi( X);
```

File

System

Description

In Delphi code, use Hi to obtain the high-order byte of an expression of type Word or Integer.

Note: Hi treats Integer as a 16-bit value, even if it is 32-bit.

Delphi Examples:

```
{
This example demonstrate use of the System Hi function to
store the high byte of a word or integer into a byte.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  B : Byte;
begin
  B := Hi($1234);
  ShowMessage(Format('High byte of $1234 = $%x', [B]));
end;
```

See Also

Lo (see page 22486)

158.3.89 System.High Function

Returns the highest value in the range of an argument.

Pascal

```
procedure High(X);
```

C++

```
High( X);
```

File

System

Description

In Delphi code, call High to obtain the upper limit of an Ordinal, Array, or ShortString value. The result type is X, or the index type of X.

X is either a type identifier or a variable reference. The type denoted by X, or the type of the variable denoted by X, must be one of the following:

For this type	High returns
Ordinal type (includes Int64)	The highest value in the range of the type
Array type	The highest value within the range of the index type of the array. For empty arrays, High returns –1.
short string type	The declared size of the string
Open array	The value, of type Integer, giving the number of elements in the actual parameter minus one
short string parameter	The value, of type Integer, giving the number of elements in the actual parameter minus one

Delphi Examples:

```
{
This example requires a button and two list boxes.  Two lists
of floating point numbers are generated initially and
displayed in the list boxes. Clicking the button sums the lists.
Notice that the range of the list arrays can be any integers,
as long as Low is less than High.
}
var
  List1: array[0..3] of Double;
  List2: array[5..17] of Double;

function Sum( var X: array of Double): Double;
var
  I: Word;
  S: Real;
begin
  S := 0; { Note that open array index range is always zero-based. }
  for I := 0 to High(X) do S := S + X[I];
  Sum := S;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  S, TempStr: string;
begin
  Str(Sum(List1):4:2, S);
  S := 'Sum of List1: ' + S + #13#10;
  S := S + 'Sum of List2: ';
  Str(Sum(List2):4:2, TempStr);
  S := S + TempStr;
  MessageDlg(S, mtInformation, [mbOk], 0);

end;

procedure TForm1.FormCreate(Sender: TObject);
var
  X: Word;
begin
  for X := Low(List1) to High(List1) do
```

```
    begin
        List1[X] := X * 3.4;
        ListBox1.Items.Add(FloatToStr(List1[X]));
    end;
    for X := Low(List2) to High(List2) do
    begin
        List2[X] := X * 0.0123;
        ListBox2.Items.Add(FloatToStr(List2[X]));
    end;
  end;
```

See Also

Low (see page 22487)

VarArrayHighBound

158.3.90 System.Inc Function

Increments an ordinal value by one or N.

Pascal

```
procedure Inc(var X); overload;
procedure Inc(var X; N: Longint); overload;
```

C++

```
Inc( X);
Inc( X, Longint N);
```

File

System

Description

In Delphi code, Inc adds one or N to the variable X.

X is a variable of an ordinal type (including Int64), or a pointer type if the extended syntax is enabled.

N is an integer-type expression.

X increments by 1, or by N if N is specified; that is, Inc(X) corresponds to the statement X := X + 1, and Inc(X, N) corresponds to the statement X := X + N. On some platforms, Inc may generate optimized code, especially useful in tight loops.

If X is a pointer type, it increments X by N times the size of the type pointed to. Thus, given

```
type
  PMyType = ^TMyType;
```

and

```
var
  P: PMyType;
```

the statement `Inc(P)` increments `P` by `SizeOf (``see page 22528``)(TMyType)`.

Warning: You can't use Inc on properties because it modifies the parameter.

Note: Inc(S, I) where S is a ShortInt and I is a number greater than 127 will cause an EIntOverFlow exception to be raised if range and overflow checking are on. In Delphi 1.0, this did not raise an exception.

Delphi Examples:

```
{
```

```
This example requires two buttons and two text fields.  The
text fields can be set to any integer and long integer value.
Click the buttons to execute Inc.
Inc example
}
procedure TForm1.Button1Click(Sender: TObject);
var
  IntVar: Integer;
begin
  IntVar := StrToInt(Edit1.Text);
  Inc(IntVar);
  Edit1.Text := IntToStr(IntVar);       { IntVar := IntVar + 1 }
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  LongintVar: Longint;
begin
  LongintVar := StrToInt(Edit2.Text);
  Inc(LongintVar, 5);
  Edit2.Text := IntToStr(LongintVar);   { LongintVar := LongintVar + 5 }
end;
```

See Also

Dec (see page 22441)

Pred (see page 22499)

Succ (see page 22532)

158.3.91 System.Include Function

Adds an element to a Delphi set or adds an event handler in .NET applications.

Pascal

```
procedure Include(var S: set of T; I: T);
```

C++

```
Include(set of T S, T I);
```

File

System

Description

When the first argument of Include is a set type variable, Include adds an element to the set. The expression `Include(S,I)` corresponds to `S := S + [I]`, where S is a set type variable, and I is an expression of a type compatible with the base type of S. The Include procedure generates more efficient code than using the + operator.

On the .NET platform, Include is overloaded. When the first argument of Include is a multicast event, and the second argument is an event handler, Include adds the event handler as a listener to the event.

See Also

Exclude (see page 22453)

158.3.92 System.Initialize Function

Initializes a dynamically allocated variable.

Pascal

```
procedure Initialize(var V); overload;
procedure Initialize(var V; Count: Integer); overload;
```

C++

```
Initialize( V);
Initialize( V, int Count);
```

File

System

Description

Initialize should be used only in Delphi code where a variable is dynamically allocated by other means than the New (see page 22491) standard procedure.

For global variables, local variables, objects, and dynamic variables allocated using New (see page 22491), the compiler generates code that initializes all long strings and variants contained by a variable upon creation of the variable. A call to Initialize is required to initialize a variable before it can be used if:

- the dynamic variable is created by other means than the New (see page 22491) standard function (for example using GetMem (see page 22469) or ReallocMem (see page 22507)).
- the variable contains long strings, variants, or interfaces.
- the dynamic variable is created by other means than the New (see page 22491) standard function (for example using GetMem (see page 22469) or ReallocMem (see page 22507)).
- the variable contains long strings, variants, or interfaces.
- the memory allocated for the variable is not initialized to zeros.

Initialize simply zeros out the memory occupied by long strings, variants, and interfaces, causing long strings to be empty and variants and interfaces to be Unassigned.

In cases where several variables are allocated in a contiguous memory block, the additional Count parameter can be specified to initialize all variables in one operation.

If the variable specified in a call to Initialize contains no long strings, variants, or interfaces, the compiler eliminates the call and generates no code for it.

See Also

Finalize (see page 22460)

New (see page 22491)

158.3.93 System.Insert Function

Inserts a substring into a string beginning at a specified point.

Pascal

```
procedure Insert(Source: string; var S: string; Index: Integer);
```

C++

```
Insert(AnsiString Source, AnsiString S, int Index);
```

File

System

Description

In Delphi code, Insert merges Source into S at the position S[index].

Source is a string-type expression. S is a string-type variable of any length. Index is an integer-type expression. It is a character index and not a byte index.

If Index is less than 1, it is mapped to a 1. If it is past the end of the string, it is set to the length of the string, turning the operation into an append.

If the Source parameter is an empty string, Insert does nothing.

Insert throws an EOutOfMemory exception if it is unable to allocate enough memory to accomodate the new returned string.

Delphi Examples:

```
{
This example uses the Insert function to edit a string.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S: string;
begin
  S := 'Honest Lincoln';
  Canvas.TextOut(10, 10, S);
  Insert('Abe ', S, 8);                    { 'Honest Abe Lincoln' }
  Canvas.TextOut(10, 40, S);
end;
```

See Also

Concat (see page 22436)

Copy (see page 22438)

Delete (see page 22443)

Length (see page 22484)

Pos (see page 22497)

158.3.94 System.Int Function

Returns the integer part of a real number.

Pascal

```
function Int(const X: Extended): Extended;
```

C++

```
Extended Int(const Extended X);
```

File

System

Description

In Delphi code, Int returns the integer part of X; that is, X rounded toward zero. X is a real-type expression.

Delphi Examples:

```
{
Click the button to display the integer part of any positive or
negative real number.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  R: Real;
begin
  R := StrToFloat(Edit1.Text);     { 123.456 }
  R := Int(R);                     { 123.0 }
  Edit2.Text := FloatToStr(R);
end;
```

See Also

Frac (see page 22463)

Round (see page 22516)

Trunc (see page 22537)

158.3.95 System.IOResult Function

Returns the status of the last I/O operation performed.

Pascal

```
function IOResult: Integer;
```

C++

```
int IOResult();
```

File

System

Description

IOResult returns the result of the last Delphi I/O routine when I/O-checking is off (that is, when using the {$I-} compiler option). If IOResult returns 0, the last I/O operation succeeded. Otherwise, IOResult returns an error code.

If an I/O error occurs and I/O-checking is off, all subsequent I/O operations are ignored until the internal error flag is cleared. Calling IOResult clears the internal error flag.

An alternative way to handle I/O errors is to use exception handling with the {$I+} compiler option.

Delphi Examples:

```
{
IOResult example
This example illustrates how to use IOResult to check the success of the Reset
method. If the call to Reset is successful, a message dialog displays the size
of the file that the user selects in an Open dialog. If the call to Reset fails,
a message warns the user that there was a file access error. To test the failure
```

```
  part of this example, just misspell a file name in the open dialog.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    F: File of Byte;
  begin
    if OpenDialog1.Execute then
    begin
      AssignFile(F, OpenDialog1.FileName);
      {$I-}
      Reset(F);
      {$I+}
      if IOResult = 0 then
      begin
        MessageDlg('File size in bytes: ' + IntToStr(FileSize(F)),
          mtInformation, [mbOk], 0);
        CloseFile(F);
      end
      else
        MessageDlg('File access error', mtWarning, [mbOk], 0);
    end;
  end;
```

158.3.96 System.IsMemoryManagerSet Function

Indicates whether the memory manager has been overridden using the SetMemoryManager (see page 22523) procedure.

Pascal

```
function IsMemoryManagerSet: Boolean;
```

C++

```
Boolean IsMemoryManagerSet();
```

File

System

Description

Call IsMemoryManagerSet to determine whether the default system memory management has been changed by a call to SetMemoryManager (see page 22523). IsMemoryManagerSet returns false if the default system memory management is in place, true otherwise.

See Also

GetMemoryManager (see page 22472)

SetMemoryManager (see page 22523)

158.3.97 System.IsVariantManagerSet Function

Indicates whether variant support is implemented in an application.

Pascal

```
function IsVariantManagerSet: Boolean; deprecated;
```

C++

```
Boolean IsVariantManagerSet();
```

File

System

Description

Use IsVariantManagerSet in Delphi code to determine whether the compiler knows the locations of functions that define variant behavior. In any program or unit that uses the Variants (see page 23567) unit (directly or indirectly), IsVariantManagerSet should return true.

Note: Although IsVariantManagerSet is defined in System.hpp, do not use this function in C++ code. Working directly with the Variant (see page 22307) manger is not guaranteed to work in C++.

See Also

GetVariantManager (see page 22474)

SetVariantManager (see page 22527)

158.3.98 System.Length Function

Returns the number of characters in a string or elements in an array.

Pascal

```
function Length(S): Integer;
```

C++

```
int Length( S);
```

File

System

Description

In Delphi code, Length returns the number of characters actually used in the string or the number of elements in the array. In C++ code, use the method of the same name on the AnsiString (see page 22093) or DynamicArray (see page 22172) class.

For single-byte and multibyte strings, Length returns the number of bytes used by the string. For Unicode (WideString (see page 22377)) strings, Length returns the number of bytes divided by two.

S is a string- or array-valued expression.

Delphi Examples:

```
{
The following example uses an edit control, a button, and a
label on a form. When the button is clicked, the hexadecimal
value of each character in the edit control is displayed in
the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
begin
  Label1.Caption := '';
  for i := 1 to Length(Edit1.Text) do
  begin
```

```
      try
        Label1.Caption :=
          Label1.Caption +
          SysUtils.IntToHex(Byte(Edit1.Text[i]),2) + ' ';
      except
        Beep;
      end;
    end;
  end;
```

See Also

Concat (see page 22436)

Copy (see page 22438)

Delete (see page 22443)

Insert (see page 22480)

Pos (see page 22497)

158.3.99 System.Ln Function

Returns the natural log of a real expression.

Pascal

```
function Ln(const X: Extended): Extended;
```

C++

```
Extended Ln(const Extended X);
```

File

System

Description

In Delphi code, Ln returns the natural logarithm (Ln(e) = 1) of the real-type expression X.

Delphi Examples:

```
{
This example calculated the exponential of 1.0 and then calculates
the natural log of that quantity to show that the natural log
function is the inverse of the exponential function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   e : real;
   S : string;
begin
   e := Exp(1.0);
   Str(ln(e):3:2, S);
   S := 'e = ' + FloatToStr(e) + '; ln(e) = ' + S;
   Canvas.TextOut(10, 10, S);
end;
```

See Also

Exp (see page 22455)

LnXP1

158.3.100 System.Lo Function

Returns the low order Byte of argument X.

Pascal

```
function Lo(X): Byte;
```

C++

```
Byte Lo( X);
```

File

System

Description

In Delphi code, Lo returns the low-order Byte of the argument X as an unsigned value. X is an expression of type Integer.

Delphi Examples:

```
{
This example demonstrate use of the System Lo function to
store the low byte of a word or integer into a byte.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  B : Byte;
begin
  B := Lo($1234);
  ShowMessage(Format('Low byte of $1234 = $%x', [B]));
end;
```

See Also

Hi (see page 22476)

158.3.101 System.LoadResourceModule Function

Pascal

```
function LoadResourceModule(ModuleName: PChar; CheckOwner: Boolean = True): LongWord;
```

C++

```
LongWord LoadResourceModule(const char * ModuleName, Boolean CheckOwner = True);
```

File

System

Description

This is function System.LoadResourceModule.

158.3.102 System.LoadResString Function

Pascal

```
function LoadResString(ResStringRec: PResStringRec): string;
```

C++

```
AnsiString LoadResString(PResStringRec ResStringRec);
```

File

System

Description

This is function System.LoadResString.

158.3.103 System.Low Function

Returns the lowest value in a range.

Pascal

```
procedure Low(X);
```

C++

```
Low( X);
```

File

System

Description

In Delphi code, call Low to obtain the lowest value or first element of an Ordinal, Array or ShortString. Result type is X, or the index type of X where X is either a type identifier or a variable reference.

Type	Low returns
Ordinal type (includes Int64)	The lowest value in the range of the type
Array type	The lowest value within the range of the index type of the array
short string type	Returns 0
Open array	Returns 0
short string parameter	Returns 0

Delphi Examples:

```
{
This example requires a button and two list boxes.  Two lists
of floating point numbers are generated initially and
displayed in the list boxes. Clicking the button sums the lists.
Notice that the range of the list arrays can be any integers,
as long as Low is less than High.
}
var
  List1: array[0..3] of Double;
```

```
    List2: array[5..17] of Double;

  function Sum( var X: array of Double): Double;
  var
    I: Word;
    S: Real;
  begin
    S := 0; { Note that open array index range is always zero-based. }
    for I := 0 to High(X) do S := S + X[I];
    Sum := S;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    S, TempStr: string;
  begin
    Str(Sum(List1):4:2, S);
    S := 'Sum of List1: ' + S + #13#10;
    S := S + 'Sum of List2: ';
    Str(Sum(List2):4:2, TempStr);
    S := S + TempStr;
    MessageDlg(S, mtInformation, [mbOk], 0);

  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    X: Word;
  begin
    for X := Low(List1) to High(List1) do
    begin
        List1[X] := X * 3.4;
        ListBox1.Items.Add(FloatToStr(List1[X]));
    end;
    for X := Low(List2) to High(List2) do
    begin
        List2[X] := X * 0.0123;
        ListBox2.Items.Add(FloatToStr(List2[X]));
    end;
  end;
```

See Also

High (see page 22476)

VarArrayLowBound

158.3.104 System.MkDir Function

Creates a new subdirectory.

Pascal

```
procedure MkDir(const S: string); overload;
procedure MkDir(P: PChar); overload;
```

C++

```
MkDir(const AnsiString S);
MkDir(const char * P);
```

File

System

Description

MkDir creates a new subdirectory with the path specified by string S or P. The last item in the path cannot be an existing file name.

Note: In Delphi, {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 22482) to check for an I/O error.

Delphi Examples:

```
{
The following sample code sets the directory of
DirectoryListBox1 to C:\WINDOWS when the form is created.
When Button1 is pressed, a subdirectory called
mytemp is added to C:\WINDOWS, but note that it is not
updated in DirectoryListBox1 until Button2 is pressed
and Update is called.
Note: A EInOutError will occur if you add a directory that
already exists.  Also, you must add the new directory to
your current path to see the magic happen.
}

procedure TForm1.Button1Click(Sender: TObject);
begin
  MkDir(Edit1.Text);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  DirectoryListBox1.Update;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  DirectoryListBox1.Directory := 'c:\WINDOWS';
end;
{
This example creates a directory using the path and directory name
entered in the text edit.  The path is relative to the project
directory.  Type in the name of an existing directory to induce the
error.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  {$I-}
  { Get directory name from TEdit control }
  MkDir(Edit1.Text);
  if IOResult <> 0 then
    MessageDlg('Cannot create directory', mtWarning, [mbOk], 0, mbOk)
  else
    MessageDlg('New directory created', mtInformation, [mbOk], 0, mbOk);
end;
```

See Also

ChDir (see page 22429)

GetDir (see page 22467)

RmDir (see page 22515)

ForceDirectories

158.3.105 System.Move Function

Copies bytes from a source to a destination.

Pascal

```
procedure Move(const Source; var Dest; Count: Integer);
```

C++

```
Move(const  Source,  Dest, int Count);
```

File

System

Description

Move copies Count bytes from Source to Dest. No range checking is performed. Move compensates for overlaps between the source and destination blocks.

Whenever possible, use the global SizeOf (see page 22528) function (Delphi) or the sizeof operator (C++) to determine the count.

Delphi Examples:

```
{
This example moves characters from a Char array into an integer.
Displaying the integer as a hex, you can see that 'W' (0x57) is
stored in the least significant byte of the integer.
}
var
  A: array[1..4] of Char;
  B: Integer;

procedure DisplayAB;
begin
  Form1.ListBox1.Items[0] := A[1];
  Form1.ListBox1.Items[1] := A[2];
  Form1.ListBox1.Items[2] := A[3];
  Form1.ListBox1.Items[3] := A[4];
  Form1.Edit1.Text := IntToHex(B, 1);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Move(A, B, SizeOf(B));  { SizeOf = safety! }
  DisplayAB;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  I : Integer;
begin
  A[1] := 'W';
  A[2] := 'H';
  A[3] := 'A';
  A[4] := 'T';
  B := 5;
  DisplayAB;
end;
```

158.3.106 System.New Function

Creates a new dynamic variable and sets P to point to it.

Pascal

```
procedure New(var P: Pointer);
```

C++

```
New(void * P);
```

File

System

Description

In Delphi code, the New procedure creates a new dynamic variable and sets a pointer variable to point to it. P is a variable of any pointer type. The size of the allocated memory block corresponds to the size of the type that P points to. The newly created variable can be referenced as P^. If there isn't enough memory available to allocate the dynamic variable, an EOutOfMemory exception is raised.

When an application is finished using a dynamic variable created with New, it should dispose of the memory allocated for the variable using the Dispose (see page 22444) standard procedure.

Delphi Examples:

```
{
The following code explicitly allocates memory for a pointer
in the OnCreate event of Form1, then releases the memory in
the OnDestroy event. Assume that MyPtr is a Pointer type
field of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  New(MyPtr);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Dispose(MyPtr);
end;
{
This example requires three buttons, four TEdits, two labels
and a populated TreeView.  The following code defines a
record type of TMyRec and a record pointer type of PMyRec.
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Edit1: TEdit;
    TreeView1: TTreeView;
    Label2: TLabel;
    Label3: TLabel;
    Button3: TButton;
    Edit4: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
```

```
    private
      { Private declarations }
    public
      { Public declarations }
    end;
    type
    PMyRec = ^TMyRec;
    TMyRec = record
      FName: string;
      LName: string;
    end;


  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  {
  Assuming these types are used, the following code adds a
  node to TreeView1 as the last sibling of the selected node.
  A TMyRec record is associated with the added item. The FName
  and LName fields are obtained from edit boxes Edit1 and
  Edit2.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    MyRecPtr: PMyRec;
    TreeViewIndex: LongInt;
  begin
    New(MyRecPtr);
    MyRecPtr^.FName := Edit1.Text;
    MyRecPtr^.LName := Edit2.Text;
    with TreeView1 do
    begin
      TreeViewIndex := Selected.AbsoluteIndex;
      if Items.Count = 0 then
        Items.AddObject(nil, 'Item' + IntToStr(TreeViewIndex), MyRecPtr)
      else if (TreeViewIndex < Items.Count) and (TreeViewIndex >= 0) then
        Items.AddObject(Items[TreeViewIndex], 'Item' + IntToStr(TreeViewIndex), MyRecPtr);
    end;
  end;
  {
  After an item containing a TMyRec record has been added, the
  following code retrieves the FName and LName values
  associated with the item and displays the values in a label.
  }
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    if (TreeView1.Selected.Data <> nil) then { query only works on new nodes }
      Edit3.Text := PMyRec(TreeView1.Selected.Data)^.FName + ' ' +
                    PMyRec(TreeView1.Selected.Data)^.LName;
  end;
  procedure TForm1.Button3Click(Sender: TObject);
  begin
    Edit4.Text := IntToStr(TreeView1.Selected.AbsoluteIndex);
  end;
```

See Also

Dispose (see page 22444)

FreeMem (see page 22464)

GetMem (see page 22469)

158.3.107 System.Odd Function

Returns true if argument is an odd number.

Pascal

```
function Odd(X: Integer): Boolean;
```

C++

```
Boolean Odd(int X);
```

File

System

Description

In Delphi code, Odd tests if the argument is an odd number. It returns true if X is an odd number, false if X is even.

Delphi Examples:

```
{
This example requires a text edit and a button with some clear
canvas in the top left ocrner of the form.  Enter the number to be
tested in the text edit and click to button to test if the entry
is odd.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   Canvas.TextOut(10, 10, '                                        ');
   if Odd(StrToInt(Edit1.Text)) then
     Canvas.TextOut(10, 10, Edit1.Text + ' is odd.')
   else
     Canvas.TextOut(10, 10, Edit1.Text + ' is even.');
end;
```

158.3.108 System.OleStrToString Function

Copies data received from a COM interface to a string.

Pascal

```
function OleStrToString(Source: PWideChar): string;
```

C++

```
AnsiString OleStrToString(PWideChar Source);
```

File

System

Description

OleStrToString copies data from the format used by COM into a Delphi string.

See Also

OleStrToStrVar (see page 22494)

StringToOleStr (see page 22531)

158.3.109 System.OleStrToStrVar Function

Copies a string from its COM representation to an existing Delphi string.

Pascal

```
procedure OleStrToStrVar(Source: PWideChar; var Dest: string);
```

C++

```
OleStrToStrVar(PWideChar Source, AnsiString Dest);
```

File

System

Description

OleStrToStrVar converts a wide string (the format used by COM interfaces) into an existing string variable.

See Also

OleStrToString (see page 22493)

StringToOleStr (see page 22531)

158.3.110 System.Ord Function

Returns the ordinal value of an ordinal-type expression.

Pascal

```
procedure Ord(X);
```

C++

```
Ord( X);
```

File

System

Description

X is a Delphi ordinal-type expression. The result is the ordinal position of X; its type is the smallest standard integer type that can hold all values of X's type.

Ord cannot operate on Int64 values.

Delphi Examples:

```
{
This example displays the ordinal value for different enumerations.
}
procedure TForm1.Button1Click(Sender: TObject);
type
   Colors = (RED,BLUE,GREEN);
var
  S: string;
begin
  S := 'BLUE has an ordinal value of ' + IntToStr(Ord(BLUE)) + #13#10;
  S := S + 'The ASCII code for "c" is ' + IntToStr(Ord('c')) +  ' decimal';
```

```
  MessageDlg(S, mtInformation, [mbOk], 0, mbOK);
end;
```

See Also

Chr (see page 22430)

158.3.111 System.ParamCount Function

Returns the number of parameters passed on the command line.

Pascal

```
function ParamCount: Integer;
```

C++

```
int ParamCount();
```

File

System

Description

ParamCount returns the number of parameters passed to the program on the command line. Separate parameters with spaces or tabs. Use double quotes to wrap multiple words as one parameter (such as long file names containing spaces).

Delphi Examples:

```
{
The following example beeps once for each "beep" passed in
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//     'System ParamCount = ' + IntToStr(System.ParamCount),
//     mtConfirmation, [mbOK], 0);
  for i := 1 to System.ParamCount do
  begin
    if LowerCase(ParamStr(i)) = 'beep' then
      Beep
    else if LowerCase(ParamStr(i)) = 'exit' then
      Application.Terminate;
    Sleep(250);
  end;
end;
```

See Also

ParamStr (see page 22496)

CmdLine (see page 22635)

158.3.112 System.ParamStr Function

Returns a specified parameter from the command-line.

Pascal

```
function ParamStr(Index: Integer): string;
```

C++

```
AnsiString ParamStr(int Index);
```

File

System

Description

ParamStr returns the parameter from the command line that corresponds to Index, or an empty string if Index is greater than ParamCount (see page 22495). For example, an Index value of 2 returns the second command-line parameter.

Note: On Windows, ParamStr(0) returns the path and file name of the executing program (for example, C:\TEST\MYPROG.EXE).

Note: On Linux, ParamStr(0) returns the command used to execute the program, without parameters (for example, ./myprogram). This behavior is dependent on information returned by the shell program and may not be consistent among all shells.

Note: Use double quotes to wrap multiple words as one parameter (such as long file names containing spaces).

Delphi Examples:

```
{
The following example beeps once for each "beep" passed in
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//     'System ParamCount = ' + IntToStr(System.ParamCount),
//    mtConfirmation, [mbOK], 0);
  for i := 1 to System.ParamCount do
  begin
    if LowerCase(ParamStr(i)) = 'beep' then
      Beep
    else if LowerCase(ParamStr(i)) = 'exit' then
      Application.Terminate;
    Sleep(250);
  end;
end;
```

See Also

ParamCount (see page 22495)

CmdLine (see page 22635)

158.3.113 System.Pi Function

Returns 3.1415926535897932385.

Pascal

```
function Pi: Extended;
```

C++

```
Extended Pi();
```

File

System

Description

Use Pi in Delphi code to represent the mathematical value pi, the ratio of a circle's circumference to its diameter. Pi is approximated as 3.1415926535897932385.

Delphi Examples:

```
{
This example shows how to use the ArcTan and FloatToStr
functions.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ex : Extended;
  mystring : String;
begin
  ex := 16 * ArcTan(1/5) - 4 * ArcTan(1/239); // Machin's formula
  mystring := FloatToStr(ex);
  Edit2.Text := mystring;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  ex : Extended;
  myString : String;
begin
  ex := Pi;
  mystring := FloatToStr(ex);
  Edit1.Text := mystring;
end;
```

158.3.114 System.Pos Function

Returns the index value of the first character in a specified substring that occurs in a given string.

Pascal

```
function Pos(const substr: AnsiString; const str: AnsiString): Integer; overload;
function Pos(const substr: WideString; const str: WideString): Integer; overload;
```

C++

```
int Pos(const AnsiString substr, const AnsiString str);
int Pos(const BSTR substr, const BSTR str);
```

File

System

Description

In Delphi, Pos searches for a substring, Substr, in a string, S. Substr and S are string-type expressions.

Pos searches for Substr within S and returns an integer value that is the index of the first character of Substr within S. Pos is case-sensitive. If Substr is not found, Pos returns zero.

The PosEx function is similar to Pos, but provides additional features and can be used in C++ code.

Delphi Examples:

```
{
The following example updates the strings in a list box
given the strings contained in another list box. If a string
in the source list box has the form Name=Value and the
destination list box contains a string with the same Name
part, the Value part in the destination list box will be
replaced by the source's value.  TListBox objects that are
not name-value pairs are not assigned a Name and IndexOfName
will return -1.
}
procedure MergeStrings(Dest, Source: TStrings);
var
  I, DI: Integer;
  begin
  for I := 0 to Source.Count - 1 do
  begin
    if Pos ('=', Source[I]) > 1 then
    begin
      DI := Dest.IndexOfName(Source.Names[I]);
      if DI > -1 then Dest[DI] := Source[I];
    end;
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  MergeStrings(ListBox1.Items, ListBox2.Items);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Plants = 10');
  ListBox1.Items.Add('Animals = 20');
  ListBox1.Items.Add('Minerals = 15');
  ListBox2.Items.Add('Animals = 4');
  ListBox2.Items.Add('Plants = 3');
  ListBox2.Items.Add('Minerals = 78');
end;
{
This example demonstrates how to edit particular elements of a string.
}
procedure TForm1.Button1Click(Sender: TObject);
var S: string;
begin
  S := Edit1.Text;
  { Convert spaces to zeros }
  while Pos(' ', S) > 0 do
```

```
    S[Pos(' ', S)] := '0';
  Edit1.Text := S;
end;
```

See Also

Concat (see page 22436)

Copy (see page 22438)

Delete (see page 22443)

Insert (see page 22480)

Length (see page 22484)

158.3.115 System.Pred Function

Returns the predecessor of the argument.

Pascal

```
procedure Pred(X);
```

C++

```
Pred( X);
```

File

System

Description

X is an expression of a Delphi ordinal type (including Int64). The result, of the same type as X, is the predecessor of X.

Delphi Examples:

```
{
AssertErrorProc example
The following code is a self-contained unit that you can add to a project
that logs failed assertions to the event log rather than raising an
exception. If /Debug is passed to the application as a command-line
parameter, all failed Assert calls cause a call to OutputDebugString,
which sends a messages to the Event Log window if run from Delphi, or
does nothing if run outside the IDE (OutputDebugString is a no-op if not
run from within a debugger). A nice side benefit is that all the line
numbers and unit names can be stripped from the EXE by turning off
assertions and rebuilding the project.
Note: AssertErrorHandler should do as little as possible.  This routine
does no cleanup, so no variables, no compiler generated variables.
Procedure calls are OK.
Note:   This example was originally provided by Brian Long.
}
type
  TForm1 = class(TForm)
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TStorage = class(TObject)
    FData: string;
```

```
    property Data: string read FData write FData;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  var
  Form1: TForm1;

procedure AssertErrorHandler(const Message, Filename: string;  LineNumber: Integer; ErrorAddr:
Pointer);

implementation

{$R *.dfm}

{$ASSERTIONS ON}

type
  EMyAssert = class(Exception);


procedure DoMessage(Message, Filename: String; LineNumber: Integer; ErrorAddr: Pointer);
var
  S: String;
begin
  S := Format('%s (%s, line %d, address $%x)',
    [Message, Filename, LineNumber, Pred(Integer(ErrorAddr))]);
  OutputDebugString(PChar(S));
end;

procedure AssertErrorHandler(const Message, Filename: string;  LineNumber: Integer; ErrorAddr:
Pointer);
{ No local variables.  Not compiler generated temporary variables. }
{ Using the call stack here will cause Access Violation errors. }
begin
  DoMessage(Message, Filename, LineNumber, ErrorAddr);
  raise EMyAssert.Create('Boom!');
end;

procedure AssertErrorNoHandler(const Message, Filename: string; LineNumber: Integer;
ErrorAddr: Pointer);
begin
end;

procedure ModifyStorage(AStorage: TStorage; const s: string);
begin
  Assert(AStorage <> nil, 'Value');
  AStorage.Data := s;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  ModifyStorage(nil, 'Oops');
end;

initialization
  if FindCmdLineSwitch('Debug', ['/', '-'], True) then
  begin
    MessageDlg('Command line switch found!', mtInformation, [mbOK], 0);
//    OldProc := System.AssertErrorProc;
    System.AssertErrorProc := @AssertErrorHandler
  end
  else
    System.AssertErrorProc := @AssertErrorNoHandler;
```

```
  ModifyStorage(nil, 'Ooops');
{
This example demonstrates how Pred and Succ can be used to
manage enumerations.
}
procedure TForm1.Button1Click(Sender: TObject);
type
  Colors = (RED,BLUE,GREEN);
var
  S: string;
begin
  S := 'The predecessor of 5 is ' + IntToStr(Pred(5)) + #13#10;
  S := S + 'The successor of 10 is ' + IntToStr(Succ(10)) + #13#10;
  if Succ(RED) = BLUE then
    S := S + 'In the type Colors, RED is the predecessor of BLUE.';
  MessageDlg(S, mtInformation, [mbOk], 0, mbOk);
end;
```

See Also

Succ (see page 22532)

158.3.116 System.Ptr Function

Converts a specified address to a pointer.

Pascal

```
function Ptr(Address: Integer): Pointer;
```

C++

```
void * Ptr(int Address);
```

File

System

Description

In Delphi code, Ptr converts the given address to a pointer. A call to this function generates no code, but simply treats the 32-bit value given by Address as a pointer. Like nil, the result of Ptr is assignment compatible with all pointer types.

See Also

Addr (see page 22412)

158.3.117 System.PUCS4Chars Function

Converts a UCS4 string to a null-terminated array of UCS4 characters.

Pascal

```
function PUCS4Chars(const S: UCS4String): PUCS4Char;
```

C++

```
PUCS4Char PUCS4Chars(const UCS4String S);
```

File

System

Description

PUCS4Chars converts the UCS-4 string specified by S to a null-terminated sequence of UCS4Char (see page 22626) values.

See Also

UCS4StringToWideString (see page 22538)

Utf8ToUnicode (see page 22542)

WideStringToUCS4String (see page 22548)

158.3.118 System.RaiseList Function

Pascal

```
function RaiseList: Pointer; deprecated;
```

C++

```
void * RaiseList();
```

File

System

Description

This is function System.RaiseList.

158.3.119 System.Random Function

Generates random numbers within a specified range.

Pascal

```
function Random(const ARange: Integer): Integer; overload;
function Random: Extended; overload;
```

C++

```
int Random(const int ARange);
Extended Random();
```

File

System

Description

In Delphi code, Random returns a random number within the range 0 <= X < Range. If Range is not specified, the result is a real-type random number within the range

`0 <= X < 1.`

To initialize the random number generator, add a single call Randomize (see page 22503) or assign a value to the RandSeed (see page 22654) variable before making any calls to Random.

Note: Because the implementation of the Random function may change between compiler versions, we do not recommend using Random for encryption or other purposes that require reproducible sequences of pseudo-random numbers.

Delphi Examples:

```
{
This example draws many rectangles of various sizes and
colors on a form maximized to fill the entire screen. To run
the code, drop a TTimer component on the form and use the
object inspector to create the OnTimer and OnActivate event
handlers.
}
var
  X, Y: Integer;
procedure TForm1.FormActivate(Sender: TObject);
begin
  WindowState := wsMaximized;
  Timer1.Interval := 50;
  Randomize;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  X := Random(Screen.Width - 10);
  Y := Random(Screen.Height - 10);
  Canvas.Pen.Color := Random(65535);
  case Random(5) of
    0: Canvas.Pen.Style := psSolid;
    1: Canvas.Pen.Style := psDash;
    2: Canvas.Pen.Style := psDot;
    3: Canvas.Pen.Style := psDashDot;
    4: Canvas.Pen.Style := psDashDotDot;
  end;
  Canvas.Rectangle(X, Y, X + Random(400), Y + Random(400));
end;
```

See Also

RandSeed (see page 22654)

158.3.120 System.Randomize Function

Initializes the random number generator with a random value.

Pascal

```
procedure Randomize;
```

C++

```
Randomize();
```

File

System

Description

Randomize initializes the built-in random number generator with a random value (obtained from the system clock). The random number generator should be initialized by making a call to Randomize, or by assigning a value to RandSeed (see page 22654).

Do not combine the call to Randomize in a loop with calls to the Random (see page 22502) function. Typically, Randomize is

called only once, before all calls to Random (see page 22502).

Delphi Examples:

```
{
This example draws many rectangles of various sizes and
colors on a form maximized to fill the entire screen. To run
the code, drop a TTimer component on the form and use the
object inspector to create the OnTimer and OnActivate event
handlers.
}
var
  X, Y: Integer;
procedure TForm1.FormActivate(Sender: TObject);
begin
  WindowState := wsMaximized;
  Timer1.Interval := 50;
  Randomize;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  X := Random(Screen.Width - 10);
  Y := Random(Screen.Height - 10);
  Canvas.Pen.Color := Random(65535);
  case Random(5) of
    0: Canvas.Pen.Style := psSolid;
    1: Canvas.Pen.Style := psDash;
    2: Canvas.Pen.Style := psDot;
    3: Canvas.Pen.Style := psDashDot;
    4: Canvas.Pen.Style := psDashDotDot;
  end;
  Canvas.Rectangle(X, Y, X + Random(400), Y + Random(400));
end;
```

See Also

RandG

RandSeed (see page 22654)

Random (see page 22502)

158.3.121 System.Read Function

Read reads data from a file.

Pascal

```
procedure Read(F; V1); overload;
procedure Read(F; V1; Vn); overload;
procedure Read(F: Text; V1); overload;
procedure Read(F: Text; V1; Vn); overload;
```

C++

```
Read( F,  V1);
Read( F,  V1,  Vn);
Read(Text F,  V1);
Read(Text F,  V1,  Vn);
```

File

System

Description

The Read procedure can be used in Delphi code in the following ways.

For typed files, it reads a file component into a variable.

For text files, it reads one or more values into one or more variables.

The syntax shown here for Read demonstrates that the procedure can take a variable number of arguments.

Read reads all characters up to, but not including, the next end-of-line marker or until Eof (see page 22449)(F) becomes true; it does not skip to the next line after reading. If the resulting string is longer than the maximum length of the string variable, it is truncated.

After the first Read, each subsequent Read sees the end-of-line marker and returns a zero-length string.

Use multiple Readln calls to read successive string values.

When the extended syntax is enabled, Read can read null-terminated strings into zero-based character arrays.

Read reads one character from the file and assigns it to the variable. If CRLF mode is enabled and Eof (see page 22449)(F) was true before Read was executed, the value Chr (see page 22430)(26) (a Ctrl-Z character) is assigned to the variable. (To enable CRLF mode, use SetLineBreakStyle (see page 22522).)

Read skips any blanks, tabs, or end-of-line markers preceding the numeric string.

If the numeric string does not conform to the expected format, an I/O error occurs; otherwise, the value is assigned to the variable.

The next Read starts with the blank, tab, or end-of-line marker that terminated the numeric string.

Delphi Examples:

```
{
This example reads a file one character at a time and writes
it into a file of your chosing.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F1, F2: TextFile;
  Ch: Char;
begin
  if OpenDialog1.Execute then begin
    AssignFile(F1, OpenDialog1.Filename);
    Reset(F1);
    if SaveDialog1.Execute then begin
      AssignFile(F2, SaveDialog1.Filename);
      Rewrite(F2);
      while not Eof(F1) do
      begin
        Read(F1, Ch);
        Write(F2, Ch);
      end;
      CloseFile(F2);
    end;
    CloseFile(F1);
  end;
end;
```

See Also

Eof (see page 22449)

Readln

Write (see page 22548)

Writeln

SetLineBreakStyle (see page 22522)

158.3.122 System.ReadLn Function

Reads a line of text from a file.

Pascal

```
procedure ReadLn(V1); overload;
procedure ReadLn(V1; Vn); overload;
procedure ReadLn(var F: Text; V1); overload;
procedure ReadLn(var F: Text; V1; Vn); overload;
```

C++

```
ReadLn( V1);
ReadLn( V1,  Vn);
ReadLn(Text F,  V1);
ReadLn(Text F,  V1,  Vn);
```

File

System

Description

In Delphi code, the ReadLn procedure reads a line of text and then skips to the next line of the file.

The syntax shown here for ReadLn demonstrates that the procedure can take a variable number of arguments.

ReadLn(F) with no parameters causes the current file position to advance to the beginning of the next line if there is one; otherwise, it goes to the end of the file.

If F is omitted, the global variable Input (see page 22647) is used, which accesses the processes standard input file. Use of Input (see page 22647) in GUI applications raises special issues.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 22482) to check for I/O errors.

Delphi Examples:

```
{
This example assumes you are compiling as a console
application.  Notice how the form deactivates when the button
click procedure exits.
}

{$APPTYPE CONSOLE}

procedure TForm1.Button1Click(Sender: TObject);
var
  s : string;
begin
  Write('Enter a line of text: ');
  Readln(s);
  Writeln('You typed: ',s);
  Writeln('Hit <Enter> to exit');
  Readln;
end;
{
Use a TOpenDialog to select a file.  Place the first line of
the file into a TEdit.
}
```

```
procedure TForm1.Edit1Click(Sender: TObject);
var
  F: TextFile;
  S: string;
begin
  if OpenDialog1.Execute then            { Display Open dialog box }
  begin
    AssignFile(F, OpenDialog1.FileName); { File selected in dialog }
    Reset(F);
    Readln(F, S);                        { Read first line of file }
    Edit1.Text := S;                     { Put string in a TEdit control }
    CloseFile(F);
  end;
end;
```

See Also

Read (see page 22504)

Writeln

SetLineBreakStyle (see page 22522)

158.3.123 System.ReallocMem Function

ReallocMem reallocates a memory block.

Pascal

```
procedure ReallocMem(var P: Pointer; Size: Integer);
```

C++

```
ReallocMem(void * P, int Size);
```

File

System

Description

GetMem (see page 22469) allocates a block of the given **Size** on the heap. If you need to change the size of this memory block, call ReallocMem, passing the existing memory block pointer in **P**, and the revised block size in **Size**. If ReallocMem can't expand the memory block pointed to by **P**, it frees the referenced memory and copies the values to the newly allocated memory that is returned.

If there isn't enough memory available to extend the memory block to the desired size, an EOutOfMemory exception is raised.

Note: ReallocMemory (see page 22508) is the C++ compatible version of ReallocMem.

See Also

AllocMem (see page 22413)

FreeMem (see page 22464)

FreeMemory (see page 22466)

GetMem (see page 22469)

GetMemory (see page 22470)

EOutOfMemory (see page 22742)

ReallocMemory (see page 22508)

158.3.124 System.ReallocMemory Function

ReallocMemory resizes a memory block.

Pascal

```
function ReallocMemory(P: Pointer; Size: Integer): Pointer; cdecl;
```

C++

```
__cdecl void * ReallocMemory(void * P, int Size);
```

File

System

Description

GetMem (see page 22469) allocates a block of the given **Size** on the heap. If you need to change the size of this memory block, call ReallocMemory, passing the existing memory block pointer in **P**, and the revised block size in **Size**. A pointer to the resized memory block is returned. If ReallocMemory can't expand the memory block pointed to by **P**, it frees the referenced memory and copies the values to the newly allocated memory that is returned.

If there isn't enough memory available to extend the memory block to the desired size, an EOutOfMemory exception is raised.

Note: ReallocMemory is the C++ compatible version of ReallocMem (see page 22507).

See Also

AllocMem (see page 22413)

FreeMem (see page 22464)

FreeMemory (see page 22466)

GetMem (see page 22469)

GetMemory (see page 22470)

EOutOfMemory (see page 22742)

ReallocMem (see page 22507)

158.3.125 System.RegisterExpectedMemoryLeak Function

Registers an expected memory leak.

Pascal

```
function RegisterExpectedMemoryLeak(P: Pointer): boolean;
```

C++

```
boolean RegisterExpectedMemoryLeak(void * P);
```

File

System

Description

RegisterExpectedMemoryLeak registers a memory location that an application has allocated and does not expect to free.

See Also

Memory Management

Configuring the Memory Manager

Registering Memory Leaks

UnregisterExpectedMemoryLeak (see page 22539)

ReportMemoryLeakOnShutdown

158.3.126 System.RegisterModule Function

Pascal

```
procedure RegisterModule(LibModule: PLibModule);
```

C++

```
RegisterModule(PLibModule LibModule);
```

File

System

Description

This is function System.RegisterModule.

158.3.127 System.ReleaseExceptionObject Function

Releases exception object acquired by AcquireExceptionObject (see page 22410).

Pascal

```
procedure ReleaseExceptionObject;
```

C++

```
ReleaseExceptionObject();
```

File

System

Description

ReleaseExceptionObject releases an exception object previously acquired by AcquireExceptionObject (see page 22410).

See Also

AcquireExceptionObject (see page 22410)

ExceptObject (see page 22452)

158.3.128 System.RemoveModuleUnloadProc Function

RemoveModuleUnloadProc removes a call to your defined routine when a package is unloaded.

Pascal

```
procedure RemoveModuleUnloadProc(Proc: TModuleUnloadProc); overload;
procedure RemoveModuleUnloadProc(Proc: TModuleUnloadProcLW); overload;
```

C++

```
RemoveModuleUnloadProc(TModuleUnloadProc Proc);
RemoveModuleUnloadProc(TModuleUnloadProcLW Proc);
```

File

System

Description

Before Delphi unloads a package that was used by your application, it can optionally call your **Proc** routine. In essence, AddModuleUnloadProc (see page 22411) adds your routine to the head of the list of internal routines that Delphi invokes to perform the unload. RemoveModuleUnloadProc reverses the effect of AddModuleUnloadProc (see page 22411).

See Also

AddModuleUnloadProc (see page 22411)

TModuleUnloadProc (see page 22617)

158.3.129 System.Rename Function

Changes the name of an external file.

Pascal

```
procedure Rename(var F; Newname: string); overload;
procedure Rename(var F; Newname: PChar); overload;
```

C++

```
Rename( F, AnsiString Newname);
Rename( F, const char * Newname);
```

File

System

Description

The external file associated with F is renamed Newname. Further operations on F operate on the external file with the new name.

F is a variable of any file type. Newname is a string-type expression or an expression of type PChar if the extended syntax is enabled.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 22482) to check for I/O errors.

Delphi Examples:

```
{
This example requires a button, an open dialog and a save
```

```
dialog.  Click the button to rename a file.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  f : file;
begin
  OpenDialog1.Title := 'Choose a file... ';
  if OpenDialog1.Execute then
  begin
    SaveDialog1.Title := 'Rename to...';
    if SaveDialog1.Execute then
    begin
      AssignFile(f, OpenDialog1.FileName);
      Canvas.TextOut(5, 10, 'Renaming ' + OpenDialog1.FileName +
                     ' to ' + SaveDialog1.FileName);
      Rename(f, SaveDialog1.FileName);
    end;
  end;
end;
```

See Also

Rename File

158.3.130 System.Reset Function

Opens an existing file.

Pascal

```
procedure Reset(var F); overload;
procedure Reset(var F; RecSize: Word); overload;
procedure Reset(var F: File); overload;
procedure Reset(var F: File; RecSize: Word); overload;
```

C++

```
Reset( F);
Reset( F, Word RecSize);
Reset(File F);
Reset(File F, Word RecSize);
```

File

System

Description

In Delphi code, Reset opens the existing external file with the name assigned to F using the mode specified by the global FileMode (see page 22644) variable. An error results if no existing external file of the given name exists or if the file can't be opened with the current file mode. If F is already open, it is first closed and then reopened. The current file position is set to the beginning of the file.

Warning: The default value of FileMode (see page 22644) is 2 (Read (see page 22504)/Write (see page 22548) access). If this is not changed to a read-only file mode before calling Reset, attempts to open read-only files will fail.

F is a variable of any file type associated with an external file using AssignFile (see page 22420). RecSize is an optional expression, which can be specified only if F is an untyped file. If F is an untyped file, RecSize specifies the record size to be used in data transfers. If RecSize is omitted, a default record size of 128 bytes is assumed.

If F is assigned an empty name, such as AssignFile (see page 22420)(F, ''), then after the call to Reset, F refers to the standard input file.

If F is a text file, F becomes read-only.

After a call to Reset, Eof (see page 22449)(F) is true if the file is empty; otherwise, Eof (see page 22449)(F) is false.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 22482) to check for I/O errors.

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], 1);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode 1.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);

      AssignFile(F, FileListBox1.Items.Strings[i]);
      Reset(F, 1);
      ListBox1.Items.Add(
        FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
      CloseFile(F);
    end;
   finally
   { do something here }
   end;
  end;
end;
{
This example should be executed from a command line
formatted like this: "SystemReset_proj foo.txt".  The file
path is relative to the project directory.
}
function FileIsThere(FileName: string): Boolean;
{ Boolean function that returns True if the file exists; otherwise,
  it returns False. Closes the file if it exists. }
 var
  F: file;
begin
```

```
    {$I-}
    AssignFile(F, FileName);
    FileMode := 0;  {Set file access to read only }
    Reset(F);
    CloseFile(F);
    {$I+}
    FileIsThere := (IOResult = 0) and (FileName <> '');
  end;  { FileIsThere }

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    if FileIsThere(ParamStr(1)) then {Get file name from command line}
      Canvas.TextOut(10, 10, 'File exists')
    else
      Canvas.TextOut(10, 10, 'File not found');
  end;
  {
  Click the button to open a TOpenDialog, then select a file
  to delete.
  }

  procedure TForm1.Button1Click(Sender: TObject);
  var
    F: Textfile;
  begin
    OpenDialog1.Title := 'Delete File';
    if OpenDialog1.Execute then
    begin
      AssignFile(F, OpenDialog1.FileName);
      try
        Reset(F);
        if MessageDlg('Erase ' + OpenDialog1.FileName + '?',
          mtConfirmation, [mbYes, mbNo], 0) = mrYes then
        begin
          CloseFile(F);
          Erase(F);
        end;
      except
        on EInOutError do
          MessageDlg('File I/O error.', mtError, [mbOk], 0);
      end;
    end;
  end;
```

See Also

Append (see page 22415)

AssignFile (see page 22420)

CloseFile (see page 22433)

Rewrite (see page 22513)

Truncate (see page 22537)

IDH

FileMode (see page 22644)

158.3.131 System.Rewrite Function

Creates a new file and opens it.

Pascal

```
procedure Rewrite(var F: File); overload;
procedure Rewrite(var F: File; Recsize: Word); overload;
```

C++

```
Rewrite(File F);
Rewrite(File F, Word Recsize);
```

File

System

Description

In Delphi code, Rewrite creates a new external file with the name assigned to F.

F is a variable of any file type associated with an external file using AssignFile (see page 22420). RecSize is an optional expression, which can be specified only if F is an untyped file. If F is an untyped file, RecSize specifies the record size to be used in data transfers. If RecSize is omitted, a default record size of 128 bytes is assumed.

If an external file with the same name already exists, it is deleted and a new empty file is created in its place.

If F is already open, it is first closed and then re-created. The current file position is set to the beginning of the empty file.

If F was assigned an empty name, such as AssignFile (see page 22420)(F,''), then after the call to Rewrite, F refers to the standard output file.

If F is a text file, F becomes write-only.

After calling Rewrite, Eof (see page 22449)(F) is always true.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 22482) to check for I/O errors.

Delphi Examples:

```
{
This example reads an entire file into a buffer with one
command and then writes it into a saved file.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FromF, ToF: file;
  NumRead, NumWritten: Integer;
  Buf: array[1..2048] of Char;
begin
  if OpenDialog1.Execute then     { Display Open dialog box }
  begin
    AssignFile(FromF, OpenDialog1.FileName);
    Reset(FromF, 1);    { Record size = 1 }
    if SaveDialog1.Execute then      { Display Save dialog box}
    begin
      AssignFile(ToF, SaveDialog1.FileName);    { Open output file }
      Rewrite(ToF, 1);  { Record size = 1 }
      Canvas.TextOut(10, 10, 'Copying ' + IntToStr(FileSize(FromF))
        + ' bytes...');
      repeat
        System.BlockRead(FromF, Buf, SizeOf(Buf), NumRead);
        BlockWrite(ToF, Buf, NumRead, NumWritten);
      until (NumRead = 0) or (NumWritten <> NumRead);
      CloseFile(FromF);
      CloseFile(ToF);
    end;
  end;
end;
```

```
{
Rewrite example
}

procedure TForm1.Button1Click(Sender: TObject);
var F: TextFile;
begin
  AssignFile(F, 'NEWFILE.$$$');
  Rewrite(F);  // default record size is 128 bytes
  Writeln(F, 'Just created file with this text in it...');
  CloseFile(F);
  MessageDlg('NEWFILE.$$$ has been created in the ' + GetCurrentDir + ' directory.',
    mtInformation, [mbOk], 0, mbOK);
end;
```

See Also

Append (see page 22415)

AssignFile (see page 22420)

Reset (see page 22511)

Truncate (see page 22537)

158.3.132 System.RmDir Function

Deletes an empty subdirectory.

Pascal

```
procedure RmDir(const S: string); overload;
procedure RmDir(P: PChar); overload;
```

C++

```
RmDir(const AnsiString S);
RmDir(const char * P);
```

File

System

Description

RmDir removes the subdirectory with the path specified by S or P. If the path does not exist, is non-empty, or is the currently logged directory, an I/O error occurs.

Note: In Delphi, {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 22482) to check for I/O errors.

Delphi Examples:

```
{
This example requires a text edit and a button.  Enter the
folder to remove in the text edit and click the button.  The
path is relative to the project folder.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   {$I-}
   { Get directory name from TEdit control }
   RmDir(Edit1.Text);
```

```
    if IOResult <> 0 then
      MessageDlg('Cannot remove directory', mtWarning, [mbOk], 0, mbOk)
    else
      MessageDlg('Directory removed', mtInformation, [mbOk], 0, mbOk);
end;
```

See Also

ChDir (see page 22429)

GetDir (see page 22467)

MkDir (see page 22488)

RemoveDir

158.3.133 System.Round Function

Returns the value of X rounded to the nearest whole number.

Pascal

```
function Round(X: Extended): Int64;
```

C++

```
Int64 Round(Extended X);
```

File

System

Description

In Delphi, the Round function rounds a real-type value to an integer-type value.

X is a real-type expression. Round returns an Int64 value that is the value of X rounded to the nearest whole number. If X is exactly halfway between two whole numbers, the result is always the even number. This method of rounding is often called "Banker's Rounding".

If the rounded value of X is not within the Int64 range, a runtime error is generated, which can be handled using the EInvalidOp exception.

Note: The behavior of Round can be affected by the Set8087CW (see page 22520) procedure or SetRoundMode function.

Delphi Examples:

```
{
This example displays the Cosine of any value between 0 and
12.  Since the scrollbar position is an integer between the
scrollbar  min and max, multiply the range and divide the
position to fine tune it.  With the scrollbar selected use
the left and right buttons to change the position by 1E-6.
Or use the Dec function to decrement (or increment) by a
specific amount.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  new : Integer;
begin
  new := ScrollBar1.Position;
  System.Dec(new, Round(StrToFloat(Edit3.Text)*1000000));
```

```
    ScrollBar1.Position := new;
    Edit1.Text := FloatToStr(ScrollBar1.Position/1000000);
    Edit2.Text := FloatToStr(Cos(ScrollBar1.Position/1000000));
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ScrollBar1.Max := Round(Pi * 4000000);   // 4 * Pi
  end;

  procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
    var ScrollPos: Integer);
  begin
    Edit1.Text := FloatToStr(ScrollPos/1000000);
    Edit2.Text := FloatToStr(Cos(ScrollPos/1000000));
  end;
  {
  This example illustrates the rules for rounding positive and
  negative real numbers to integers.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
     S, T: string;
  begin
     Str(1.4:2:1, T);
     S := T + ' rounds to ' + IntToStr(Round(1.4)) + #13#10;
     Str(1.5:2:1, T);
     S := S + T + ' rounds to ' + IntToStr(Round(1.5)) + #13#10;
     Str(-1.4:2:1, T);
     S := S + T + ' rounds to ' + IntToStr(Round(-1.4)) + #13#10;
     Str(-1.5:2:1, T);
     S := S + T + ' rounds to ' + IntToStr(Round(-1.5));
     MessageDlg(S, mtInformation, [mbOk], 0, mbOk);
  end;
```

See Also

Int (see page 22481)

Trunc (see page 22537)

Set8087CW (see page 22520)

SetRoundMode (see page 15972)

RoundTo (see page 15968)

SimpleRoundTo (see page 15973)

158.3.134 System.RunError Function

Stops execution and generates a runtime error.

Pascal

```
procedure RunError; overload;
procedure RunError(Errorcode: Byte); overload;
```

C++

```
RunError();
RunError(Byte Errorcode);
```

File

System

Description

In Delphi, the RunError procedure stops program execution by generating a runtime error with the given number at the current statement. Errorcode is the runtime error number (0 if omitted).

Delphi Examples:

```
{
This example demonstrates how to set a run error and the behavior of
execution and the message that follows.
}
var P: Pointer;

procedure TForm1.Button1Click(Sender: TObject);
begin
{$IFDEF Debug}
  if P = nil then
    RunError(204);
{$ENDIF}
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  P:= nil;
end;
```

See Also

Halt (see page 22475)

158.3.135 System.Seek Function

Moves the current position of a file to a specified component.

Pascal

```
procedure Seek(var F; N: Longint);
```

C++

```
Seek( F, Longint N);
```

File

System

Description

Use Seek in Delphi code to move to a specified position in open typed or untyped files. The current file position of F moves to component number N, where the number of the first component of a file is 0.

In the above syntax, F is a typed or untyped file variable. The file specified by F must be open. N is an expression of type Longint

To expand a file, seek one component beyond the last component; that is, the statement Seek(F, FileSize (see page 22457)(F)) moves the current file position to the end of the file.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 22482) to check for I/O errors.

Delphi Examples:

```
{
FileSize, Seek, FilePos example
This example displays a open dialog when the user clicks a button. When the user selects the
file, it reports the size of the file, seeks to halfway through the file, and reports the file
position at that point.
Note:   This example assumes the compiler flag $IOCHECKS ON ($I+). That is, it expects errors
to generate exceptions rather than checking IOResult to be sure that I/O routines succeed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   f: file of Byte;
   size: Longint;
   S: string;
   y: Integer;
begin
  if OpenDialog1.Execute then
  begin
    AssignFile(f, OpenDialog1.FileName);
    Reset(f);
    try
      size := FileSize(f);
      S := 'File size in bytes: ' + IntToStr(size);
      y := 10;
      Canvas.TextOut(5, y, S);
      y := y + Canvas.TextHeight(S) + 5;
      S := 'Seeking halfway into file...';
      Canvas.TextOut(5, y, S);
      y := y + Canvas.TextHeight(S) + 5;
      Seek(f, size div 2);
      S := 'Position is now ' + IntToStr(FilePos(f));
      Canvas.TextOut(5, y, S);
    finally
      CloseFile(f);
    end;
  end;
end;
```

See Also

FilePos (see page 22456)

158.3.136 System.SeekEof Function

Returns the end-of-file status of a file, ignoring whitespace.

Pascal

```
function SeekEof: Boolean; overload;
function SeekEof(var F: Text): Boolean; overload;
```

C++

```
Boolean SeekEof();
Boolean SeekEof(Text F);
```

File

System

Description

In Delphi code, call SeekEof to determine whether there is only whitespace between the file pointer and the end of a file.

SeekEof moves the file pointer past any whitespace, leaving it positioned at the end of a file or on the next non-whitespace character, whichever comes first. If it leaves the file pointer at the end of the file, SeekEof returns true. Otherwise, it returns false.

SeekEof can only be used on open text files.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 22482) to check for I/O errors.

See Also

Eof (see page 22449)

SeekEoln (see page 22520)

SetLineBreakStyle (see page 22522)

158.3.137 System.SeekEoln Function

Returns the end-of-line status of a file, ignoring whitespace.

Pascal

```
function SeekEoln: Boolean; overload;
function SeekEoln(var F: Text): Boolean; overload;
```

C++

```
Boolean SeekEoln();
Boolean SeekEoln(Text F);
```

File

System

Description

In Delphi code, call SeekEoln to determine whether there is only whitespace between the file pointer and the end of the current line. SeekEoln moves the file pointer past any whitespace, leaving it positioned at the end of the current line, the end of the file, or on the next non-whitespace character (whichever comes first). If SeekEoln leaves the file pointer at the end of the current line or at the end of the file, it returns true. Otherwise, it returns false.

SeekEoln can be used only on open text files.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 22482) to check for I/O errors.

See Also

Eoln (see page 22450)

SeekEof (see page 22519)

SetLineBreakStyle (see page 22522)

158.3.138 System.Set8087CW Function

Sets both the control word in the floating point unit and the variable NoErrMsg (see page 22652) declared in the system unit.

Pascal

```
procedure Set8087CW(NewCW: Word);
```

C++

```
Set8087CW(Word NewCW);
```

File

System

Description

The floating-point unit control word controls the precision of floating point calculations, the rounding mode, and whether certain floating-point operations trigger exceptions. See Intel's processor documentation for details.

This routine allows the programmer to have direct access to the CW. Be aware that using this routine to change the value of the 8087CW will change the behavior of the program's FP calculations. It is the programmer's responsibility to reset it.

It is recommended that you disable all floating-point exceptions when using OpenGL to render 3D graphics. To do this, call Set8087CW(0x133f) in your main form's OnCreate event before calling any OpenGL functions.

See Also

NoErrMsg (see page 22652)

SetExceptionMask

SetPrecisionMode

SetRoundMode

158.3.139 System.SetInOutRes Function

Pascal

```
procedure SetInOutRes(NewValue: Integer);
```

C++

```
SetInOutRes(int NewValue);
```

File

System

Description

This is function System.SetInOutRes.

158.3.140 System.SetLastError Function

SetLastError sets the last API error code.

Pascal

```
procedure SetLastError(ErrorCode: Integer); stdcall;
```

C++

```
__stdcall SetLastError(int ErrorCode);
```

File

System

Description

GetLastError (see page 22469) returns the last API error code encountered. It then resets it to 0. SetLastError forces the API error code to the specified **ErrorCode** value. A call to GetLastError (see page 22469) will then return this value before again resetting the error code to 0.

See Also

GetLastError (see page 22469)

158.3.141 System.SetLength Function

Sets the length of a string or dynamic-array variable.

Pascal

```
procedure SetLength(var S; NewLength: Integer);
```

C++

```
SetLength( S, int NewLength);
```

File

System

Description

S is a Delphi string or dynamic-array variable.

NewLength is the new number of characters or elements in S.

For a short-string variable, SetLength simply sets the length-indicator character (the character at S[0]) to the given value. In this case, NewLength must be a value between 0 and 255.

For a long-string or dynamic-array variable, SetLength reallocates the string or array referenced by S to the given length. Existing characters in the string or elements in the array are preserved, but the content of newly allocated space is undefined. The one exception is when increasing the length of a dynamic array in which the elements are types that must be initialized (strings, Variants (see page 23567), Variant (see page 22307) arrays, or records that contain such types). When S is a dynamic array of types that must be initialized, newly allocated space is set to 0 or nil.

For dynamic arrays, SetLength may take more than one length parameter (up to the number of array dimensions). Each parameter specifies the number of elements along a particular dimension.

Following a call to SetLength, S is guaranteed to reference a unique string or array—that is, a string or array with a reference count of one. If there is not enough memory available to reallocate the variable, SetLength raises an EOutOfMemory exception.

See Also

EOutOfMemory

string

IDH

158.3.142 System.SetLineBreakStyle Function

Determines the end-of-line and end-of-file conventions for text file I/O.

Pascal

```
procedure SetLineBreakStyle(var T: Text; Style: TTextLineBreakStyle);
```

C++

```
SetLineBreakStyle(Text T, TTextLineBreakStyle Style);
```

File

System

Description

In Delphi, call SetLineBreakStyle to determine how text file routines find or mark the end of a line and the end of a file. Affected routines include Read (see page 22504), Write (see page 22548), Readln, Writeln, Eof (see page 22449), Eoln (see page 22450), SeekEof (see page 22519), and SeekEoln (see page 22520).

T is the text file whose line break style is altered. It must be closed, or SetLineBreakStyle generates a runtime error.

Style indicates the style to use. When Style is tlbsLF, end-of-line is marked using only a linefeed character. When Style is tlbsCRLF, end-of-line is marked using a carriage return and linefeed.

See Also

Readln

Writeln

Read (see page 22504)

Write (see page 22548)

Eof (see page 22449)

Eoln (see page 22450)

SeekEof (see page 22519)

SeekEoln (see page 22520)

Text

DefaultTextLineBreakStyle (see page 22636)

158.3.143 System.SetMemoryManager Function

Sets entry points of the memory manager.

Pascal

```
procedure SetMemoryManager(const MemMgr: TMemoryManager); overload; deprecated;
procedure SetMemoryManager(const MemMgrEx: TMemoryManagerEx); overload;
```

C++

```
SetMemoryManager(const TMemoryManager MemMgr);
SetMemoryManager(const TMemoryManagerEx MemMgrEx);
```

File

System

Description

SetMemoryManager sets the entry points of the memory manager to the three functions encapsulated by the MemMgr

parameter. Use SetMemoryManager to change the memory allocation routines that are used for:

Allocation and deallocation of dynamic strings and dynamic arrays.

The GetMem (see page 22469), ReallocMem (see page 22507), and FreeMem (see page 22464) standard procedures (Delphi only)

The New (see page 22491) and Dispose (see page 22444) standard procedures (Delphi only).

See Also

GetMemoryManager (see page 22472)

GetMem (see page 22469)

ReallocMem (see page 22507)

FreeMem (see page 22464)

New (see page 22491)

Dispose (see page 22444)

IsMemorymanagerSet

158.3.144 System.SetMinimumBlockAlignment Function

Sets the minimum memory block alignment.

Pascal

```
procedure SetMinimumBlockAlignment(AMinimumBlockAlignment: TMinimumBlockAlignment);
```

C++

```
SetMinimumBlockAlignment(TMinimumBlockAlignment AMinimumBlockAlignment);
```

File

System

Description

SetMinimumBlockAlignment sets the Memory Manager's minimum block alignment. Existing allocations are not affected if this setting is changed. The memory manager will enforce the new setting for all allocations made after the change.

See Also

Memory Management

Configuring the Memory Manager

158.3.145 System.SetMultiByteConversionCodePage Function

SetMultiByteConversionCodePage overrides the default multi byte code conversion page.

Pascal

```
procedure SetMultiByteConversionCodePage(CodePage: Integer);
```

C++

```
SetMultiByteConversionCodePage(int CodePage);
```

File

System

Description

RAD Studion textual elements such as captions are stored as WideStrings. When running under 9x, they are automatically converted to AnsiStrings. This conversion uses, by default, the System (see page 22079) Code Page. SetMultiByteConversionCodePage allows you to specify the **CodePage** that is used in such conversions.

See Also

GetLastError (see page 22469)

158.3.146 System.SetRaiseList Function

Pascal

```
function SetRaiseList(NewPtr: Pointer): Pointer; deprecated;
```

C++

```
void * SetRaiseList(void * NewPtr);
```

File

System

Description

This is function System.SetRaiseList.

158.3.147 System.SetString Function

Sets the contents and length of the given string.

Pascal

```
procedure SetString(var s: string; buffer: PChar; len: Integer);
```

C++

```
SetString(AnsiString s, const char * buffer, int len);
```

File

System

Description

In Delphi code, SetString sets the contents and length of the given string variable to the block of characters given by the Buffer and Len parameters.

For a short string variable, SetString sets the length indicator character (the character at S[0]) to the value given by Len and then, if the Buffer parameter is not nil, copies Len characters from Buffer into the string starting at S[1]. For a short string variable, the Len parameter must be a value between 0 and 255.

For a long string variable, SetString sets S to reference a newly allocated string of the given length. If the Buffer parameter is not nil, SetString then copies Len characters from Buffer into the string; otherwise, the contents of the new string is left uninitialized. If there is not enough memory available to create the string, an EOutOfMemory exception is raised. Following a call to SetString, S is guaranteed to reference a unique string, that is a string with a reference count of one.

See Also

EOutOfMemory

string

158.3.148 System.SetTextBuf Function

Assigns an I/O buffer to a text file.

Pascal

```
procedure SetTextBuf(var F: Text; var Buf); overload;
procedure SetTextBuf(var F: Text; var Buf; Size: Integer); overload;
```

C++

```
SetTextBuf(Text F,  Buf);
SetTextBuf(Text F,  Buf, int Size);
```

File

System

Description

In Delphi code, SetTextBuf changes the text file F to use the buffer specified by Buf instead of F's internal buffer. F is a text file variable, Buf is any variable, and Size is an optional expression.

Each Text file variable has an internal 128-byte buffer that buffers Read (see page 22504) and Write (see page 22548) operations. This buffer is adequate for most operations. However, heavily I/O-bound programs benefit from a larger buffer to reduce disk head movement and file system overhead.

Size specifies the size of the buffer in bytes. If Size is omitted, SizeOf (see page 22528)(Buf) is assumed. The new buffer remains in effect until F is next passed to AssignFile (see page 22420).

SetTextBuf can be called immediately after Reset (see page 22511), Rewrite (see page 22513), and Append (see page 22415), but never apply it to an open file.

When SetTextBuf is called on an open file once I/O operations have taken place, data could be lost because of the change of buffer.

The Delphi runtime library does not ensure that the buffer exists for the entire duration of I/O operations on the file. A common error is to install a local variable as a buffer, then use the file outside the procedure that declared the buffer.

See Also

Append (see page 22415)

AssignFile (see page 22420)

Read (see page 22504)

Reset (see page 22511)

Rewrite (see page 22513)

SizeOf (see page 22528)

Write (see page 22548)

158.3.149 System.SetVariantManager Function

Sets the entry points of routines that define variant behavior.

Pascal

```
procedure SetVariantManager(const VarMgr: TVariantManager); deprecated;
```

C++

```
SetVariantManager(const TVariantManager VarMgr);
```

File

System

Description

SetVariantManager sets the entry points of the routines that perform variant type conversion and define other variant behavior.

Note: Although System.hpp defines the SetVariantManager procedure, do not use this routine in C++ code. Customized Variant (see page 22307) support is not fully supported in C++.

See Also

GetVariantManager (see page 22474)

IsVariantManagerSet (see page 22483)

158.3.150 System.ShareMemoryManager Function

Shares the module's memory manager.

Pascal

```
function ShareMemoryManager: Boolean;
```

C++

```
Boolean ShareMemoryManager();
```

File

System

Description

ShareMemoryManager makes the module's memory manager available for sharing to other modules in the same process. Only one memory manager may be shared per process. It returns **False** if another module in the process is already sharing its memory manager.

Note: This function is not thread safe.

See Also

Memory Management

Configuring the Memory Manager

Sharing Memory

158.3.151 System.Sin Function

Returns the sine of the angle in radians.

Pascal

```
function Sin(const X: Extended): Extended;
```

C++

```
Extended Sin(const Extended X);
```

File

System

Description

In Delphi code, the Sin function returns the sine of the argument.

X is a real-type expression. Sin returns the sine of the angle X in radians.

See Also

ArcTan (see page 22416)

Cos (see page 22440)

158.3.152 System.SizeOf Function

Returns the number of bytes occupied by a variable or type.

Pascal

```
function SizeOf(X): Integer;
```

C++

```
int SizeOf( X);
```

File

System

Description

Pass a Delphi variable reference to SizeOf to determine the number of bytes used to represent the variable. Pass a type identifier to SizeOf to determine the number of bytes used to represent instances of that type. SizeOf is useful for determining the amount of memory to specify for the FillChar (see page 22459), Move (see page 22490), or GetMem (see page 22469) procedures.

SizeOf returns 0 when the argument is an untyped variable.

Delphi Examples:

```
{
This example fills a char array with characters and then
assigns the array to a text edit.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  S: array[0..79] of char;
begin
  { Set to all Xs }
  FillChar(S, SizeOf(S), Ord('X'));
  Edit1.Text := S;
end;
```

See Also

FillChar (see page 22459)

GetMem (see page 22469)

Move (see page 22490)

158.3.153 System.Slice Function

Returns a sub-section of an array.

Pascal

```
function Slice(var A: array; Count: Integer): array;
```

C++

```
array Slice(array A, int Count);
```

File

System

Description

In Delphi code, use Slice to select a sub-section (slice) of an array to be passed as an open array parameter. Slice is only allowed as a parameter in a call to a procedure or function that expects an open array parameter. Slice selects the first Count elements of the array given by A to be passed as an open array parameter. A can be an open array parameter itself, allowing dynamically allocated arrays to be passed to open array parameters.

See Also

SetLength (see page 22522)

IDH

158.3.154 System.Sqr Function

Returns the square of a number.

Pascal

```
function Sqr(X: Extended): Extended; overload;
function Sqr(X: Integer): Integer; overload;
```

C++

```
Extended Sqr(Extended X);
int Sqr(int X);
```

File

System

Description

In Delphi code, the Sqr function returns the square of the argument.

X is a floating-point expression. The result, of the same type as X, is the square of X, or X*X.

See Also

Sqrt (see page 22530)

Power

158.3.155 System.Sqrt Function

Returns the square root of X.

Pascal

```
function Sqrt(const X: Extended): Extended;
```

C++

```
Extended Sqrt(const Extended X);
```

File

System

Description

X is a Delphi floating-point expression. The result is the square root of X.

See Also

Sqr (see page 22529)

158.3.156 System.Str Function

Formats a string and returns it to a variable.

Pascal

```
procedure Str(X: Width; var S); overload;
procedure Str(X: Decimals; var S); overload;
```

C++

```
Str(Width X,  S);
Str(Decimals X,  S);
```

File

System

Description

In Delphi code, Str converts X to a string representation according to the Width and Decimals formatting parameters. The effect is like a call to Write (see page 22548) except the resulting string is stored in S instead of being written to a text file.

X is an integer-type or real-type expression. Width and Decimals are integer-type expressions. S is a string-type variable or a zero-based character array variable if extended syntax is enabled.

See Also

Write (see page 22548)

158.3.157 System.StringOfChar Function

Returns a string with a specified number of repeating characters.

Pascal

```
function StringOfChar(ch: AnsiChar; Count: Integer): AnsiString; overload;
function StringOfChar(ch: WideChar; Count: Integer): WideString; overload;
```

C++

```
AnsiString StringOfChar(AnsiChar ch, int Count);
BSTR StringOfChar(WideChar ch, int Count);
```

File

System

Description

In Delphi code, StringOfChar returns a string containing Count characters with the character value given by Ch. For example,

S := StringOfChar('A', 10);

sets S to the string 'AAAAAAAAAA'.

See Also

FillChar (see page 22459)

DupeString

158.3.158 System.StringToOleStr Function

Allocates memory and copies an AnsiStringa string to the OLE format.

Pascal

```
function StringToOleStr(const Source: string): PWideChar;
```

C++

```
PWideChar StringToOleStr(const AnsiString Source);
```

File

System

Description

StringToOleStr allocates memory and copies string data from a native AnsiString (see page 22093) into a format that can be passed to OLE functions.

See Also

OleStrToString (see page 22493)

OleStrToStrVar (see page 22494)

StringToWideChar (see page 22532)

158.3.159 System.StringToWideChar Function

Returns a UNICODE string from an AnsiString (see page 22093).

Pascal

```
function StringToWideChar(const Source: string; Dest: PWideChar; DestSize: Integer): PWideChar;
```

C++

```
PWideChar StringToWideChar(const AnsiString Source, PWideChar Dest, int DestSize);
```

File

System

Description

StringToWideChar converts a string from ANSI to UNICODE and stores the result in a specified buffer.

Source is the string to convert. It must include at most DestSize – 1 characters.

Dest is the buffer into which StringToWideChar writes the corresponding UNICODE string. The caller allocates this memory to include at least DestSize wide characters. Following the call, Dest contains at most DestSize - 1 characters, terminated by a NULL wide character.

DestSize is the size of the buffer Dest.

StringToWideChar returns a pointer to Dest.

See Also

WideCharToString (see page 22547)

WideCharLenToString (see page 22546)

WideCharToStrVar (see page 22547)

WideCharlenToStrVar

158.3.160 System.Succ Function

Returns the successor of an argument.

Pascal

```
procedure Succ(X);
```

C++

```
Succ( X);
```

File

System

Description

In Delphi code, Succ returns the successor of the argument, X.

X is an expression of an ordinal type (including Int64). The result, of the same type as X, is the successor of X.

Delphi Examples:

```
{
This example demonstrates how Pred and Succ can be used to
manage enumerations.
}
procedure TForm1.Button1Click(Sender: TObject);
type
  Colors = (RED,BLUE,GREEN);
var
  S: string;
begin
  S := 'The predecessor of 5 is ' + IntToStr(Pred(5)) + #13#10;
  S := S + 'The successor of 10 is ' + IntToStr(Succ(10)) + #13#10;
  if Succ(RED) = BLUE then
    S := S + 'In the type Colors, RED is the predecessor of BLUE.';
  MessageDlg(S, mtInformation, [mbOk], 0, mbOk);
end;
```

See Also

Pred (see page 22499)

158.3.161 System.Swap Function

Exchanges high order byte with the low order byte of an integer or word.

Pascal

```
procedure Swap(X);
```

C++

```
Swap( X);
```

File

System

Description

In Delphi code, Swap exchanges the high-order bytes with the low-order bytes of the argument. X is an expression of type SmallInt, as a 16-bit value, or Word. This is provided for backward compatibility only.

See Also

Hi (see page 22476)

Lo (see page 22486)

158.3.162 System.SysAllocMem Function

Pascal

```
function SysAllocMem(Size: Cardinal): Pointer;
```

C++

```
void * SysAllocMem(unsigned Size);
```

File

System

Description

This is function System.SysAllocMem.

158.3.163 System.SysFreeMem Function

Frees the memory pointed to by a specified pointer.

Pascal

```
function SysFreeMem(P: Pointer): Integer;
```

C++

```
int SysFreeMem(void * P);
```

File

System

Description

When implementing a custom memory manager, use SysFreeMem to free the memory pointed to by the P parameter.

Note: On Windows, the System (see page 22079) unit provides a limited implementation of SysFreeMem. If the custom memory manager needs to support reference counting and objects that can be shared between processes, use the implementation of SysFreeMem provided by the ShareMem unit. When used, ShareMem must be the first unit in the project file, to enforce a consistent memory management model.

Note: On Linux, the System (see page 22079) unit provides an implementation of SysFreeMem that supports reference counting and shared objects. The ShareMem unit is not provided for Linux.

See Also

GetHeapStatus (see page 22468)

SysGetMem (see page 22534)

SetMemoryManager (see page 22523)

158.3.164 System.SysGetMem Function

Allocates a specified number of bytes and returns a pointer to them.

Pascal

```
function SysGetMem(Size: Integer): Pointer;
```

C++

```
void * SysGetMem(int Size);
```

File

System

Description

When implementing a custom memory manager, use SysGetMem to dynamically allocate Size bytes of memory.

Note: On Windows, the System (see page 22079) unit provides a limited implementation of SysGetMem. If the custom memory manager needs to support reference counting and objects that can be shared between processes, use the implementation of SysGetMem provided by the ShareMem unit. When used, ShareMem must be the first unit in the project file, to enforce a consistent memory management model.

Note: On Linux, the System (see page 22079) unit provides an implementation of SysGetMem that supports reference counting and shared objects. The ShareMem unit is not provided for Linux.

See Also

SysFreeMem (see page 22534)

SysReallocMem (see page 22535)

SetMemoryManager (see page 22523)

158.3.165 System.SysReallocMem Function

Returns a pointer to a specified number of bytes, preserving the values pointed to by the Pointer parameter.

Pascal

```
function SysReallocMem(P: Pointer; Size: Integer): Pointer;
```

C++

```
void * SysReallocMem(void * P, int Size);
```

File

System

Description

When implementing a custom memory manager, use SysReallocMem to change the size of a dynamically allocated chunk of memory. If SysReallocMem can't expand the memory block point to by P, it frees the referenced memory and copies the values to the newly allocated memory that is returned. If Size is less than the number of bytes pointed to by P, only the first Size bytes are preserved. Otherwise, the entire value of the memory pointed to by P is preserved in the newly allocated memory.

Note: On Windows, the System (see page 22079) unit provides a limited implementation of SysReallocMem. If the custom memory manager needs to support reference counting and objects that can be shared between processes, use the implementation of SysReallocMem provided by the ShareMem unit. When used, ShareMem must be the first unit in the project file, to enforce a consistent memory management model.

Note: On Linux, the System (see page 22079) unit provides an implementation of SysReallocMem that supports reference counting and shared objects. The ShareMem unit is not provided for Linux.

See Also

SysFreeMem (see page 22534)

SysGetMem (see page 22534)

SetMemoryManager (see page 22523)

158.3.166 System.SysRegisterExpectedMemoryLeak Function

Pascal

```
function SysRegisterExpectedMemoryLeak(P: Pointer): Boolean;
```

C++

```
Boolean SysRegisterExpectedMemoryLeak(void * P);
```

File

System

Description

This is function System.SysRegisterExpectedMemoryLeak.

158.3.167 System.SysUnregisterExpectedMemoryLeak Function

Pascal

```
function SysUnregisterExpectedMemoryLeak(P: Pointer): Boolean;
```

C++

```
Boolean SysUnregisterExpectedMemoryLeak(void * P);
```

File

System

Description

This is function System.SysUnregisterExpectedMemoryLeak.

158.3.168 System.TextStart Function

Pascal

```
procedure TextStart; deprecated;
```

C++

```
TextStart();
```

File

System

Description

This is function System.TextStart.

158.3.169 System.Trunc Function

Truncates a real number to an integer.

Pascal

```
function Trunc(X: Extended): Int64;
```

C++

```
Int64 Trunc(Extended X);
```

File

System

Description

In Delphi code, the Trunc function truncates a real-type value to an integer-type value. X is a real-type expression. Trunc returns an Int64 value that is the value of X rounded toward zero.

If the truncated value of X is not within the Int64 range, an EInvalidOp exception is raised.

See Also

Int (see page 22481)

Round (see page 22516)

Floor (see page 15945)

Ceil (see page 15937)

158.3.170 System.Truncate Function

Deletes all the records after the current file position.

Pascal

```
procedure Truncate(var F);
```

C++

```
Truncate( F);
```

File

System

Description

In Delphi code, Call Truncate to cause the current file position to become end-of-file (Eof (see page 22449)(F) is true).

F is a file variable of any type except a text file. Truncate does not work on text files. F must be open.

See Also

Reset (see page 22511)

Rewrite (see page 22513)

Seek (see page 22518)

158.3.171 System.UCS4StringToWideString Function

Converts a string of 4-byte Unicode characters into a WideString (see page 22377) value.

Pascal

```
function UCS4StringToWideString(const S: UCS4String): WideString;
```

C++

```
BSTR UCS4StringToWideString(const UCS4String S);
```

File

System

Description

Call UCS4StringToWideString to convert a string of UCS4 (4-byte Unicode) characters into a WideString (see page 22377) value. UCS4 strings are returned by some Linux operating system calls.

See Also

WideStringToUCS4String (see page 22548)

PUCS4Chars (see page 22501)

UnicodeToUtf8 (see page 22538)

Utf8ToUnicode (see page 22542)

158.3.172 System.UnicodeToUtf8 Function

Converts a string of Unicode characters into a UTF-8 string.

Pascal

```
function UnicodeToUtf8(Dest: PChar; Source: PWideChar; MaxBytes: Integer): Integer; overload;
deprecated;
function UnicodeToUtf8(Dest: PChar; MaxDestBytes: Cardinal; Source: PWideChar; SourceChars:
Cardinal): Cardinal; overload;
```

C++

```
int UnicodeToUtf8(const char * Dest, PWideChar Source, int MaxBytes);
unsigned UnicodeToUtf8(const char * Dest, unsigned MaxDestBytes, PWideChar Source, unsigned
SourceChars);
```

File

System

Description

Call UnicodeToUtf8 to convert a Unicode string to a null-terminated sequence of UTF-8 characters. Use the second syntax. The deprecated syntax is included only for backward compatibility.

Dest receives the resulting null-terminated array of UTF-8 characters.

MaxDestBytes or MaxBytes indicates the number of bytes available in the buffer specified by Dest, including the byte for the null terminator.

Source is an array of Unicode characters.

SourceChars is the number of characters in Source. If SourceChars is not specified, Source must be null-terminated.

UnicodeToUtf8 returns the number of bytes written to Dest.

See Also

UCS4StringToWideString (see page 22538)

Utf8ToUnicode (see page 22542)

Utf8Encode

Utf8Decode

UTF8String (see page 22627)

158.3.173 System.UniqueString Function

Ensures that a given string has a reference count of one.

Pascal

```
procedure UniqueString(var str: AnsiString); overload;
procedure UniqueString(var str: WideString); overload;
```

C++

```
UniqueString(AnsiString str);
UniqueString(BSTR str);
```

File

System

Description

UniqueString forces the reference count on an AnsiString (see page 22093) to one, copying the string in memory if necessary. For normal string handling, there is no need to call UniqueString. UniqueString is used only in cases where an application modifies the contents of a string after

Note: casting the string to a PChar or PWideChar in Delphi.

Note: using the c_str() method of AnsiString (see page 22093) or the c_bstr() method of WideString (see page 22377) in C++.

See Also

SetString (see page 22525)

SetLength (see page 22522)

158.3.174 System.UnregisterExpectedMemoryLeak Function

Unregisters an expected memory leak.

Pascal

```
function UnregisterExpectedMemoryLeak(P: Pointer): boolean;
```

C++

```
boolean UnregisterExpectedMemoryLeak(void * P);
```

File

System

Description

UnregisterExpectedMemoryLeak removes a memory location from the Memory Manager's list of expected memory leaks.

See Also

Memory Management

Configuring the Memory Manager

Registering Memory Leaks

RegisterExpectedMemoryLeak (see page 22508)

ReportMemoryLeakOnShutdown

158.3.175 System.UnregisterModule Function

Pascal

```
procedure UnregisterModule(LibModule: PLibModule);
```

C++

```
UnregisterModule(PLibModule LibModule);
```

File

System

Description

This is function System.UnregisterModule.

158.3.176 System.UpCase Function

Converts a character to uppercase.

Pascal

```
function UpCase(Ch: Char): Char;
```

C++

```
Char UpCase(Char Ch);
```

File

System

Description

UpCase converts a character to uppercase. Ch is an expression of type Char. Character values not in the range a..z are unaffected.

See Also

AnsiUpperCase

StrUpper

UpperCase

158.3.177 System.UTF8Decode Function

Converts a UTF8 string to a Unicode string (WideString (see page 22377)).

Pascal

```
function UTF8Decode(const S: UTF8String): WideString;
```

C++

```
BSTR UTF8Decode(const UTF8String S);
```

File

System

Description

Call UTF8Decode to convert a UTF-8 string to Unicode. S is a string encoded in UTF-8. UTF8Decode returns the corresponding WideString (see page 22377) value that represents the string in Unicode.

See Also

UnicodeToUtf8 (see page 22538)

UTF8Encode (see page 22541)

Utf8ToUnicode (see page 22542)

Utf8ToAnsi (see page 22542)

AnsiToUtf8 (see page 22415)

158.3.178 System.UTF8Encode Function

Converts a WideString (see page 22377) value to a UTF8 string.

Pascal

```
function UTF8Encode(const WS: WideString): UTF8String;
```

C++

```
UTF8String UTF8Encode(const BSTR WS);
```

File

System

Description

Call Utf8Encode to convert a Unicode string to UTF-8. WS is the Unicode string to convert. Utf8Encode returns the corresponding UTF-8 string.

See Also

UnicodeToUtf8 (see page 22538)

UTF8Decode (see page 22541)

Utf8ToUnicode (see page 22542)

158.3.179 System.Utf8ToAnsi Function

Converts a UTF8 string to a string of Ansi characters.

Pascal

```
function Utf8ToAnsi(const S: UTF8String): string;
```

C++

```
AnsiString Utf8ToAnsi(const UTF8String S);
```

File

System

Description

Call Utf8ToAnsi to convert a UTF-8 string to Ansi. S is a string encoded in UTF-8. Utf8ToAnsi returns the corresponding string that uses the Ansi character set.

See Also

UnicodeToUtf8 (see page 22538)

Utf8Encode

Utf8ToUnicode (see page 22542)

Utf8Decode

AnsiToUtf8 (see page 22415)

158.3.180 System.Utf8ToUnicode Function

Converts a string of UTF-8 characters to a Unicode string.

Pascal

```
function Utf8ToUnicode(Dest: PWideChar; Source: PChar; MaxChars: Integer): Integer; overload;
deprecated;
function Utf8ToUnicode(Dest: PWideChar; MaxDestChars: Cardinal; Source: PChar; SourceBytes:
Cardinal): Cardinal; overload;
```

C++

```
int Utf8ToUnicode(PWideChar Dest, const char * Source, int MaxChars);
unsigned Utf8ToUnicode(PWideChar Dest, unsigned MaxDestChars, const char * Source, unsigned
```

```
SourceBytes);
```

File

System

Description

Call Utf8ToUnicode to convert a UTF-8 string to Unicode. Use the second syntax. The deprecated syntax is provided for backward compatibility only.

Dest receives the resulting null-terminated array of Unicode characters.

MaxDestChars or MaxChars indicates the maximum number of Unicode characters that the buffer specified by Dest can hold, not counting the null terminator.

Source is an array of UTF-8 characters.

SourceBytes is the number of bytes in Source. If SourceBytes is not specified, Source must be null-terminated.

Utf8ToUnicode returns the number of characters written to Dest.

In case of errors, the Utf8ToUnicode ... : Cardinal variant returns `Cardinal(-1) == $FFFFFFFF` and the Utf8ToUnicode ... : Integer returns `-1`.

See Also

UCS4StringToWideString (see page 22538)

UnicodeToUtf8 (see page 22538)

Utf8Encode

Utf8Decode

UTF8String (see page 22627)

158.3.181 System.Val Function

Converts a string to a numeric representation.

Pascal

```
procedure Val(S; var V; var Code: Integer);
```

C++

```
Val( S,  V, int Code);
```

File

System

Description

In Delphi code, Val converts the string value S to its numeric representation, as if it were read from a text file with Read (see page 22504).

S is a string-type expression; it must be a sequence of characters that form a signed real number.

V is an integer-type or real-type variable. If V is an integer-type variable, S must form a whole number.

Code is a variable of type Integer.

If the string is invalid, the index of the offending character is stored in Code; otherwise, Code is set to zero. For a null-terminated

string, the error position returned in Code is one larger than the actual zero-based index of the character in error.

See Also

Str (see page 22530)

158.3.182 System.VarArrayRedim Function

Resizes a Variant (see page 22307) array.

Pascal

```
procedure VarArrayRedim(A: Variant; HighBound: Integer);
```

C++

```
VarArrayRedim(Variant A, int HighBound);
```

File

System

Description

VarArrayRedim resizes the given Variant (see page 22307) array by changing the high bound of the rightmost dimension to the given value. The A parameter can be a Variant (see page 22307) or an OleVariant (see page 22193), and it must be possible to assign a value to A (A must be an lvalue).

Existing elements of the array are preserved, and new elements are set to zero or empty. An EVariantArrayCreateError exception is thrown if the Variant (see page 22307) given by A is not an array, or if there is not enough memory to resize the array.

Note: A Variant (see page 22307) array can't be resized while it is locked.

See Also

VarArrayCreate

VarArrayLock

EVariantArrayCreateError

158.3.183 System::VarArrayRedim Function

Pascal

```
procedure VarArrayRedim(var a: Variant; high: Integer);
```

C++

```
__fastcall void VarArrayRedim(Variant& a, int high);
```

File

sysvari.h

Description

This is function System::VarArrayRedim.

158.3.184 System::VarCast Function

Pascal

```
procedure VarCast(var dest: Variant; const source: Variant&; type: Integer);
```

C++

```
__fastcall void VarCast(Variant & dest, const Variant& source, int type);
```

File

sysvari.h

Description

This is function System::VarCast.

158.3.185 System.VarClear Function

Empties a Variant (see page 22307), so that it is unassigned.

Pascal

```
procedure VarClear(V: Variant);
```

C++

```
VarClear(Variant V);
```

File

System

Description

Calling VarClear is equivalent to assigning the Unassigned constant to the Variant (see page 22307). V can be either a Variant (see page 22307) or an OleVariant (see page 22193), but it must be possible to assign a value to it (it must be an lvalue).

After calling VarClear, the VarIsEmpty function returns true, and the VarType function returns varEmpty (see page 22683). Using an unassigned variant in an expression causes an exception to be thrown. Likewise, if you attempt to convert an unassigned Variant (see page 22307) to another type (using VarAsType), an exception is thrown.

Note: Do not confuse clearing a Variant (see page 22307), which leaves it unassigned, with assigning a Null value. A Null Variant (see page 22307) is still assigned, but has the value Null. Unlike unassigned Variants (see page 23567), Null Variants (see page 23567) can be used in expressions and can be converted to other types of Variants (see page 23567).

See Also

Unassigned

VarIsEmpty

VarAsType

VarType

158.3.186 System::VarClear Function

Pascal

```
procedure VarClear(var v: Variant);
```

C++

```
__fastcall void VarClear(Variant& v);
```

File

sysvari.h

Description

This is function System::VarClear.

158.3.187 System.WideCharLenToString Function

Converts Unicode characters to single or multi byte character data.

Pascal

```
function WideCharLenToString(Source: PWideChar; SourceLen: Integer): string;
```

C++

```
AnsiString WideCharLenToString(PWideChar Source, int SourceLen);
```

File

System

Description

WideCharLenToString converts SourceLen Unicode characters from the buffer given by Source to a string made up of single or multi byte characters.

See Also

StringToWideChar (see page 22532)

WideCharToString (see page 22547)

158.3.188 System.WideCharLenToStrVar Function

WideCharLenToStrVar converts Unicode characters to single or multi byte character data.

Pascal

```
procedure WideCharLenToStrVar(Source: PWideChar; SourceLen: Integer; var Dest: string);
```

C++

```
WideCharLenToStrVar(PWideChar Source, int SourceLen, AnsiString Dest);
```

File

System

Description

The WideCharToStrVar (see page 22547) procedure converts SourceLen WideChar (Unicode) characters from Source into single or multi byte character data, and copies the result into Dest.

See Also

WideCharToString (see page 22547)

158.3.189 System.WideCharToString Function

Converts null-terminated Unicode string to single or multi byte character data.

Pascal

```
function WideCharToString(Source: PWideChar): string;
```

C++

```
AnsiString WideCharToString(PWideChar Source);
```

File

System

Description

WideCharToString converts the given null-terminated Unicode string to a string made up of single- or multi-byte characters.

See Also

StringToWideChar (see page 22532)

WideCharLenToString (see page 22546)

158.3.190 System.WideCharToStrVar Function

Converts Unicode string to a single or multi byte character data.

Pascal

```
procedure WideCharToStrVar(Source: PWideChar; var Dest: string);
```

C++

```
WideCharToStrVar(PWideChar Source, AnsiString Dest);
```

File

System

Description

WideCharToStrVar converts UNICODE string Source into single or multi byte character data, and copies the result into Dest. The source must be null-terminated.

See Also

WideCharToString (see page 22547)

158.3.191 System.WideStringToUCS4String Function

Converts a WideString (see page 22377) value to a string of 4-byte Unicode characters.

Pascal

```
function WideStringToUCS4String(const S: WideString): UCS4String;
```

C++

```
UCS4String WideStringToUCS4String(const BSTR S);
```

File

System

Description

Call WideStringToUCS4String to convert a WideString (see page 22377) value to a string of UCS4 (4-byte Unicode) characters. UCS4 strings are used by some Linux operating system calls.

158.3.192 System.Write Function

Writes to either a typed file or a text file

Pascal

```
procedure Write(P1); overload;
procedure Write(P1; Pn); overload;
procedure Write(var F: Text; P1); overload;
procedure Write(var F: Text; P1; Pn); overload;
```

C++

```
Write( P1);
Write( P1,  Pn);
Write(Text F,  P1);
Write(Text F,  P1,  Pn);
```

File

System

Description

The syntax shown here for the Write procedure is an example that illustrates that Write can take a variable number of arguments.

Write procedure for typed files:

In Delphi code, Write writes a file to a file component. F is a file variable, and each V is a variable of the same type as the component type of F. For each variable written, the current file position is advanced to the next component. If the current file position is at the end of the file (that is, if Eof (see page 22449)(F) is true), the file is expanded.

Write procedure for text files:

In Delphi code, Write writes one or more values to a text file. F, if specified, is a text file variable. If F is omitted, the standard file

variable Output (see page 22653) is assumed. Each P is a write parameter. Each write parameter includes an output expression whose value is to be written to the file. A write parameter can also contain the specifications of a field width and a number of decimal places. Each output expression must be of a type Char, one of the Integer types (Byte, Shortint, Word, Longint, Cardinal), one of the floating-point types (Single, Real, Double, Extended, Currency (see page 22145)), one of the string types (PChar, AnsiString (see page 22093), ShortString), a packed string, or one of the Boolean types (Boolean, Bool).

A write parameter has the form:

```
OutExpr [: MinWidth [: DecPlaces ] ]
```

where OutExpr is an output expression. MinWidth and DecPlaces are type integer expressions.

MinWidth specifies the minimum field width, which must be greater than 0. Exactly MinWidth characters are written (using leading blanks if necessary) except when OutExpr has a value that must be represented in more than MinWidth characters. In that case, enough characters are written to represent the value of OutExpr. Likewise, if MinWidth is omitted, then the necessary number of characters is written to represent the value of OutExpr.

DecPlaces specifies the number of decimal places in a fixed-point representation of one of the Real types. It can be specified only if OutExpr is one of the Real types, and if MinWidth is also specified. When MinWidth is specified, it must be greater than or equal to 0.

Write with a character-type value: If MinWidth is omitted, the character value of OutExpr is written to the file. Otherwise, MinWidth - 1 blanks followed by the character value of OutExpr is written.

Write with one of the integer type values: If MinWidth is omitted, the decimal representation of OutExpr is written to the file with no preceding blanks. If MinWidth is specified and its value is larger than the length of the decimal string, enough blanks are written before the decimal string to make the field width MinWidth.

Write with one of the real type values: If OutExpr has one of the real type values, its decimal representation is written to the file. The format of the representation depends on the presence or absence of DecPlaces. If DecPlaces is omitted (or if it's present but has a negative value), a floating-point decimal string is written. If MinWidth is also omitted, a default MinWidth of 17 is assumed; otherwise, if MinWidth is less than 8, it's assumed to be 8. The format of the floating-point string is:

```
[   | - ] <digit> . <decimals> E [ + | - ] <exponent>
```

The following table lists the components of the output string.

Component	Meaning
[| —]	" “ or “-”l, according to the sign of OutExpr
<digit>	Single digit, “0” only if OutExpr is 0
<decimals>	Digit string of MinWidth-7 (but at most 10) digits
E	Uppercase [E] character
[+ | —]	According to the sign of the exponent
<exponent>	Two-digit decimal exponent

If DecPlaces is present, a fixed-point decimal string is written. If DecPlaces is larger than 11, it's assumed to be 11. The format of the fixed-point string follows:

```
 [<blanks>] [ - ] <digits> [.<decimals>]
```

The following table lists the components of the fixed-point string:

Component	Meaning
[<blanks>	Blanks to satisfy MinWidth
[-]	If OutExpr is negative
<digits>	At least one digit, but no leading zeros

[.<decimals>]	Decimals if DecPlaces > 0

Write with one of the string-type values: If MinWidth is omitted, the string value of OutExpr is written to the file with no leading blanks. If MinWidth is specified, and its value is larger than the length of OutExpr, enough blanks are written before the decimal string to make the field width MinWidth.

Write with a packed string-type value: If OutExpr is of packed string type, the effect is the same as writing a string whose length is the number of elements in the packed string type.

Write with one of the Boolean type values: If OutExpr is of type Boolean, the effect is the same as writing the strings true or false, depending on the value of OutExpr.

Note: When using Write the file must be open for output.

Delphi Examples:

```
{
This example reads a file one character at a time and writes
it into a file of your chosing.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F1, F2: TextFile;
  Ch: Char;
begin
  if OpenDialog1.Execute then begin
    AssignFile(F1, OpenDialog1.Filename);
    Reset(F1);
    if SaveDialog1.Execute then begin
      AssignFile(F2, SaveDialog1.Filename);
      Rewrite(F2);
      while not Eof(F1) do
      begin
        Read(F1, Ch);
        Write(F2, Ch);
      end;
      CloseFile(F2);
    end;
    CloseFile(F1);
  end;
end;
```

See Also

WriteLn (see page 22550)

158.3.193 System.WriteLn Function

Writes to a text file and adds an end-of-line marker.

Pascal

```
procedure WriteLn(P1); overload;
procedure WriteLn(P1; Pn); overload;
procedure WriteLn(var F: Text; P1); overload;
procedure WriteLn(var F: Text; P1; Pn); overload;
```

C++

```
WriteLn( P1);
WriteLn( P1,  Pn);
```

```
WriteLn(Text F,  P1);
WriteLn(Text F,  P1,  Pn);
```

File

System

Description

WriteLn is an extension to the Write (see page 22548) procedure, as it is defined for text files.

The syntax shown here for the WriteLn procedure is an example that illustrates that WriteLn can take a variable number of arguments.

After executing Write (see page 22548), WriteLn writes an end-of-line marker (line feed or carriage return/line feed) to the file.

If F is omitted, the global variable Output (see page 22653) is used, which accesses the processes standard input file. Use of Output (see page 22653) in GUI applications raises special issues.

Delphi Examples:

```
{
This example assumes you are compiling as a console
application.  Notice how the form deactivates when the button
click procedure exits.
}

{$APPTYPE CONSOLE}

procedure TForm1.Button1Click(Sender: TObject);
var
  s : string;
begin
  Write('Enter a line of text: ');
  Readln(s);
  Writeln('You typed: ',s);
  Writeln('Hit <Enter> to exit');
  Readln;
end;
```

See Also

Write (see page 22548)

SetLineBreakStyle (see page 22522)

Output (see page 22653)

158.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TChunkStatus (see page 22554)	TChunkStatus enumerates the possible states of a chunk of memory.
	TMinimumBlockAlignment (see page 22566)	TMinimumBlockAlignment defines the mimimum byte alignment for memory block allocation.
	TRuntimeError (see page 22567)	This is record System.TRuntimeError.
	TTextLineBreakStyle (see page 22569)	TTextLineBreakStyle indicates the default set of characters used to divide lines in text.

Records

	Name	Description
	PackageInfoTable (see page 22553)	This is record System.PackageInfoTable.
	PackageUnitEntry (see page 22553)	This is record System.PackageUnitEntry.
	TCallDesc (see page 22553)	TCallDesc is the dispatch call descriptor for variants, and PCallDesc (see page 22588) is a pointer to this descriptior.
	TCVModInfo (see page 22555)	This is record System.TCVModInfo.
	TDispatchMessage (see page 22555)	This is record System.TDispatchMessage.
	TDispDesc (see page 22556)	This is record System.TDispDesc.
	TDynArrayTypeInfo (see page 22556)	This is record System.TDynArrayTypeInfo.
	TFileRec (see page 22557)	TFileRec is the internal format for typed and untyped files in the Delphi language.
	TGUID (see page 22558)	TGUID is a structured form of the value that uniquely identifies an interface.
	THeapStatus (see page 22558)	THeapStatus represents information about the global heap.
	TInitContext (see page 22560)	This is record System.TInitContext.
	TInterfaceEntry (see page 22560)	This is record System::TInterfaceEntry.
	TInterfaceTable (see page 22561)	This is record System::TInterfaceTable.
	TLibModule (see page 22561)	TLibModule describes a module that is compiled with the application.
	TMemoryManager (see page 22562)	TMemoryManager defines the memory block entry points.
	TMemoryManagerEx (see page 22563)	TMemoryManagerEx defines extended memory block entry points.
	TMemoryManagerState (see page 22564)	TMemoryManagerState holds statistics about memory allocations.
	TMethod (see page 22565)	TMethod stores Code and Data fields to represent a method.
	TModuleUnloadRec (see page 22566)	This is record System.TModuleUnloadRec.
	TResStringRec (see page 22567)	This is record System.TResStringRec.
	TSmallBlockTypeState (see page 22568)	TSmallBlockTypeState holds statistics about a small memory block.
	TTextRec (see page 22569)	TTextRec is the internal format for text files.
	TVarArray (see page 22570)	This is record System.TVarArray.
	TVarArrayBound (see page 22571)	This is record System.TVarArrayBound.
	TVarData (see page 22571)	TVarData represents Variant (see page 22307) information.
	TVariantManager (see page 22575)	TVariantManager defines entry points for variant conversion routines and other procedures that define variant behavior.
	TVarRec (see page 22577)	This is class System::TVarRec.

Structures

Name	Description
TCallDesc (see page 22578)	This is type System::TCallDesc.
TDispDesc (see page 22578)	This is type System::TDispDesc.

158.4.1 System.PackageInfoTable Record

Pascal

```
PackageInfoTable = packed record
  UnitCount: Integer;
  UnitInfo: PUnitEntryTable;
end;
```

C++

```
struct PackageInfoTable {
  int UnitCount;
  PUnitEntryTable UnitInfo;
};
```

File

System

Description

This is record System.PackageInfoTable.

158.4.2 System.PackageUnitEntry Record

Pascal

```
PackageUnitEntry = packed record
  FInit: Pointer;
  Init: Pointer;
end;
```

C++

```
struct PackageUnitEntry {
  void * FInit;
  void * Init;
};
```

File

System

Description

This is record System.PackageUnitEntry.

158.4.3 System.TCallDesc Record

TCallDesc is the dispatch call descriptor for variants, and PCallDesc (see page 22588) is a pointer to this descriptior.

Pascal

```
TCallDesc = packed record
  CallType: Byte;
  ArgCount: Byte;
```

```
    NamedArgCount: Byte;
    ArgTypes: array[0..255] of Byte;
  end;
```

C++

```
  struct TCallDesc {
    Byte CallType;
    Byte ArgCount;
    Byte NamedArgCount;
    array[0..255] of Byte ArgTypes;
  };
```

File

System

Description

TCallDesc identifies a method call that was made using a Variant (see page 22307).

CallType identifies the method that was called.

ArgCount indicates the total number of arguments passed to the method.

NamedArgCount

ArgTypes is an array giving the type codes for each argument.

See Also

PCallDesc (see page 22588)

158.4.4 System.TChunkStatus Enumeration

TChunkStatus enumerates the possible states of a chunk of memory.

Pascal

```
  TChunkStatus = (
    csUnallocated,
    csAllocated,
    csReserved,
    csSysAllocated,
    csSysReserved
  );
```

C++

```
  enum TChunkStatus {
    csUnallocated,
    csAllocated,
    csReserved,
    csSysAllocated,
    csSysReserved
  };
```

File

System

Description

GetMemoryMap (see page 22473) returns a map of the address space of the current process. For the purposes of the map, the entire 4GB address space is broken down into 64K chunks. There are 65536 such chunks. TMemoryMap (see page 22617) obtains a TChunkStatus status value for each of them in the TMemoryMap (see page 22617) array. TChunkStatus

defines the following values :

Status value	Meaning
csUnallocated	Free
csAllocated	In use by the process
csReserved	Reserved for future use by the process
csSysAllocated	In use by the operating system
csSysReserved	Reserved for future use by the operating system

See Also

GetMemoryMap (see page 22473)

TMemoryMap (see page 22617)

158.4.5 System.TCVModInfo Record

Pascal

```
TCVModInfo = packed record
  ModName: PChar;
  LibName: PChar;
  UserData: Pointer;
end;
```

C++

```
struct TCVModInfo {
  const char * ModName;
  const char * LibName;
  void * UserData;
};
```

File

System

Description

This is record System.TCVModInfo.

158.4.6 System.TDispatchMessage Record

Pascal

```
TDispatchMessage = record
  MsgID: Word;
end;
```

C++

```
struct TDispatchMessage {
  Word MsgID;
};
```

File

System

Description

This is record System.TDispatchMessage.

158.4.7 System.TDispDesc Record

Pascal

```
TDispDesc = packed record
  DispID: Integer;
  ResType: Byte;
  CallDesc: TCallDesc;
end;
```

C++

```
struct TDispDesc {
  int DispID;
  Byte ResType;
  TCallDesc CallDesc;
};
```

File

System

Description

This is record System.TDispDesc.

158.4.8 System.TDynArrayTypeInfo Record

Pascal

```
TDynArrayTypeInfo = packed record
  kind: Byte;
  name: string[0];
  elSize: Longint;
  elType: ^PDynArrayTypeInfo;
  varType: Integer;
end;
```

C++

```
struct TDynArrayTypeInfo {
  Byte kind;
  string[0] name;
  Longint elSize;
  ^PDynArrayTypeInfo elType;
  int varType;
};
```

File

System

Description

This is record System.TDynArrayTypeInfo.

158.4.9 System.TFileRec Record

TFileRec is the internal format for typed and untyped files in the Delphi language.

Pascal

```
TFileRec = packed record
  Handle: Integer;
  Mode: Word;
  Flags: Word;
  case Byte of
    0: (RecSize: Cardinal;);
    1: (BufSize: Cardinal;
        BufPos: Cardinal;
        BufEnd: Cardinal;
        BufPtr: PChar;
        OpenFunc: Pointer;
        InOutFunc: Pointer;
        FlushFunc: Pointer;
        CloseFunc: Pointer;
        UserData: array[1..32] of Byte;
        Name: array[0..259] of Char;);
end;
```

C++

```
struct TFileRec {
  int Handle;
  Word Mode;
  Word Flags;
  unsigned RecSize;
  unsigned BufSize;
  unsigned BufPos;
  unsigned BufEnd;
  const char * BufPtr;
  void * OpenFunc;
  void * InOutFunc;
  void * FlushFunc;
  void * CloseFunc;
  array[1..32] of Byte UserData;
  array[0..259] of Char Name;
};
```

File

System

Description

TFileRec is the internal format for typed and untyped files. TFileRec enables you to typecast a Delphi file variable to access its internal fields.

Note: You would normally never declare a variable of this type.

158.4.10 System.TGUID Record

TGUID is a structured form of the value that uniquely identifies an interface.

Pascal

```
TGUID = packed record
  D1: LongWord;
  D2: Word;
  D3: Word;
  D4: array[0..7] of Byte;
end;
```

C++

```
struct TGUID {
  LongWord D1;
  Word D2;
  Word D3;
  array[0..7] of Byte D4;
};
```

File

System

Description

An interface declaration can specify a globally unique identifier (GUID), represented by a string literal enclosed in brackets immediately preceding the member list. The GUID part of the declaration must have the form :

['{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}']

where each x is a hexadecimal digit (0 through 9 or A through F). On Windows, the Type Library editor automatically generates GUIDs for new interfaces. You can also generate GUIDs by pressing Ctrl+Shift+G in the Code editor.

TGUID provides a structured access to this GUID :

D1 maps to the first 8 hexadecimal digits.

D2 maps to the next 4 hexadecimal digits.

D3 maps to the next 4 hexadecimal digits.

D4 maps to the final 8 hexadecimal digits.

158.4.11 System.THeapStatus Record

THeapStatus represents information about the global heap.

Pascal

```
THeapStatus = record
  TotalAddrSpace: Cardinal;
  TotalUncommitted: Cardinal;
  TotalCommitted: Cardinal;
  TotalAllocated: Cardinal;
  TotalFree: Cardinal;
  FreeSmall: Cardinal;
  FreeBig: Cardinal;
```

```
    Unused: Cardinal;
    Overhead: Cardinal;
    HeapErrorCode: Cardinal;
  end;
```

C++

```
  struct THeapStatus {
    unsigned TotalAddrSpace;
    unsigned TotalUncommitted;
    unsigned TotalCommitted;
    unsigned TotalAllocated;
    unsigned TotalFree;
    unsigned FreeSmall;
    unsigned FreeBig;
    unsigned Unused;
    unsigned Overhead;
    unsigned HeapErrorCode;
  };
```

File

System

Description

THeapStatus represents information about the current memory manager. These are the meanings of the values in each field:

Field	Meaning
TotalAddrSpace	The (current) total address space available to your program, in bytes. This will grow as your program's dynamic memory usage grows.
TotalUncommitted	The total number of bytes (of TotalAddrSpace) for which space has not been allocated in the swap file.
TotalCommitted	The total number of bytes (of TotalAddrSpace) for which space has been allocated in the swap file. Note : TotalUncommitted + TotalCommitted = TotalAddrSpace
TotalAllocated	The total number of bytes dynamically allocated by your program.
TotalFree	The total number of free bytes available in the (current) address space for allocation by your program. If this number is exceeded, and enough virtual memory is available, more address space will be allocated from the OS; TotalAddrSpace will be incremented accordingly.
FreeSmall	Total bytes of small memory blocks which are not currently allocated by your program.
FreeBig	Total bytes of big memory blocks which are not currently allocated by your program. Large free blocks can be created by coalescing smaller, contiguous, free blocks or by freeing a large dynamic allocation. (The exact size of the blocks is immaterial).
Unused	Total bytes which have never been allocated by your program. Note : Unused + FreeBig + FreeSmall = TotalFree These three fields (Unused, FreeBig, and FreeSmall) refer to dynamic allocation by the user program.
Overhead	The total number of bytes required by the heap manager to manage all the blocks dynamically allocated by your program.
HeapErrorCode	Indicates the current status of the heap, as internally determined.

Note: TotalAddrSpace, TotalUncommitted and TotalCommitted refer to OS memory used by the program, whereas TotalAllocated and TotalFree refer to the heap memory used within the program by dynamic allocations. Therefore, to monitor dynamic memory used in your program use TotalAllocated and TotalFree.

158.4.12 System.TInitContext Record

Pascal

```
TInitContext = record
  OuterContext: PInitContext;
  ExcFrame: Pointer;
  InitTable: PackageInfo;
  InitCount: Integer;
  Module: PLibModule;
  DLLSaveEBP: Pointer;
  DLLSaveEBX: Pointer;
  DLLSaveESI: Pointer;
  DLLSaveEDI: Pointer;
  ExitProcessTLS: procedure;
  DLLInitState: Byte;
end;
```

C++

```
struct TInitContext {
  PInitContext OuterContext;
  void * ExcFrame;
  PackageInfo InitTable;
  int InitCount;
  PLibModule Module;
  void * DLLSaveEBP;
  void * DLLSaveEBX;
  void * DLLSaveESI;
  void * DLLSaveEDI;
  procedure ExitProcessTLS;
  Byte DLLInitState;
};
```

File

System

Description

This is record System.TInitContext.

158.4.13 System::TInterfaceEntry Record

Pascal

```
TInterfaceEntry = packed record
  IID: TGUID;
  VTable: Pointer;
  IOffset: Integer;
  ImplGetter: Integer;
end;
```

C++

```
struct TInterfaceEntry {
  TGUID IID;
  Pointer VTable;
  Integer IOffset;
  Integer ImplGetter;
```

```
};
```

File

System.pas

Description

This is record System::TInterfaceEntry.

158.4.14 System::TInterfaceTable Record

Pascal

```
TInterfaceTable = packed record
  EntryCount: Integer;
  Entries: array[0..9999] of TInterfaceEntry;
end;
```

C++

```
struct TInterfaceTable {
  Integer EntryCount;
  array[0..9999] of TInterfaceEntry Entries;
};
```

File

System.pas

Description

This is record System::TInterfaceTable.

158.4.15 System.TLibModule Record

TLibModule describes a module that is compiled with the application.

Pascal

```
TLibModule = record
  Next: PLibModule;
  Instance: LongWord;
  CodeInstance: LongWord;
  DataInstance: LongWord;
  ResInstance: LongWord;
  Reserved: Integer;
end;
```

C++

```
struct TLibModule {
  PLibModule Next;
  LongWord Instance;
  LongWord CodeInstance;
  LongWord DataInstance;
  LongWord ResInstance;
  int Reserved;
};
```

File

System

Description

TLibModule is an entry in a linked list of data structures that describe the modules in an application.

Next is a pointer to the next entry in the list.

The **Instance** field represents the instance handle of the executable or package and the instance handles of the text, data, and resource associated with that executable instance.

The additional fields on Linux are used internally to access information in the module header.

See Also

PLibModule (see page 22596)

158.4.16 System.TMemoryManager Record

TMemoryManager defines the memory block entry points.

Pascal

```
TMemoryManager = record
  GetMem: function(Size: Integer): Pointer;
  FreeMem: function(P: Pointer): Integer;
  ReallocMem: function(P: Pointer; Size: Integer): Pointer;
end;
```

C++

```
struct TMemoryManager {
  function(Size: Integer): Pointer GetMem;
  function(P: Pointer): Integer FreeMem;
  function(P: Pointer; Size: Integer): Pointer ReallocMem;
};
```

File

System

Description

The TMemoryManager type is used by the GetMemoryManager (see page 22472) and SetMemoryManager (see page 22523) procedures. It defines the routines that allocate and free memory.

The GetMem field (Delphi) or method (C++) must specify a function that allocates the given number of bytes and returns a pointer to the newly allocated block, as invoked by the GetMemory (see page 22470) routine.

The Size parameter passed to the GetMem function will never be zero. If the GetMem function cannot allocate a block of the given size, it should return nil (Delphi) or NULL (C++).

The FreeMem field (Delphi) or method (C++) must specify a function that deallocates the given block. The pointer parameter passed to the FreeMem function will never be nil (Delphi) or NULL (C++). If the FreeMem function successfully deallocates the given block, it should return zero. Otherwise, it should return a non-zero value. As invoked by the FreeMemory (see page 22466) routine.

The ReallocMem field (Delphi) or method (C++) must specify a function that reallocates the given block to the given new size. The pointer parameter passed to the ReallocMem function will never be nil (Delphi) or NULL (C++), and the Size parameter will never be zero. The ReallocMem function must reallocate the given block to the given new size, possibly moving the block if it

cannot be resized in place. Any existing contents of the block must be preserved, but newly allocated space can be uninitialized. The ReallocMem function must return a pointer to the reallocated block, or nil (Delphi) or NULL (C++) if the block cannot be reallocated. As invoked by the ReallocMem routine.

See Also

AllocMem (see page 22413)

FreeMemory (see page 22466)

GetMemoryManager (see page 22472)

ReallocMem (see page 22507)

SetMemoryManager (see page 22523)

158.4.17 System.TMemoryManagerEx Record

TMemoryManagerEx defines extended memory block entry points.

Pascal

```
TMemoryManagerEx = record
  GetMem: function(Size: Integer): Pointer;
  FreeMem: function(P: Pointer): Integer;
  ReallocMem: function(P: Pointer; Size: Integer): Pointer;
  AllocMem: function(Size: Cardinal): Pointer;
  RegisterExpectedMemoryLeak: function(P: Pointer): Boolean;
  UnregisterExpectedMemoryLeak: function(P: Pointer): Boolean;
end;
```

C++

```
struct TMemoryManagerEx {
  function(Size: Integer): Pointer GetMem;
  function(P: Pointer): Integer FreeMem;
  function(P: Pointer; Size: Integer): Pointer ReallocMem;
  function(Size: Cardinal): Pointer AllocMem;
  function(P: Pointer): Boolean RegisterExpectedMemoryLeak;
  function(P: Pointer): Boolean UnregisterExpectedMemoryLeak;
};
```

File

System

Description

The RegisterExpectedMemoryLeak type is used by the GetMemoryManager (see page 22472) and SetMemoryManager (see page 22523) procedures. It defines the routines that allocate and free memory.

The GetMem field (Delphi) or method (C++) must specify a function that allocates the given number of bytes and returns a pointer to the newly allocated block, as invoked by the GetMemory (see page 22470) routine.

The Size parameter passed to the GetMem function will never be zero. If the GetMem function cannot allocate a block of the given size, it should return nil (Delphi) or NULL (C++).

The FreeMem field (Delphi) or method (C++) must specify a function that deallocates the given block. The pointer parameter passed to the FreeMem function will never be nil (Delphi) or NULL (C++). If the FreeMem function successfully deallocates the given block, it should return zero. Otherwise, it should return a non-zero value. As invoked by the FreeMemory (see page 22466) routine.

The ReallocMem field (Delphi) or method (C++) must specify a function that reallocates the given block to the given new size. The pointer parameter passed to the ReallocMem function will never be nil (Delphi) or NULL (C++), and the Size parameter will never be zero. The ReallocMem function must reallocate the given block to the given new size, possibly moving the block if it cannot be resized in place. Any existing contents of the block must be preserved, but newly allocated space can be uninitialized. The ReallocMem function must return a pointer to the reallocated block, or nil (Delphi) or NULL (C++) if the block cannot be reallocated. As invoked by the ReallocMem routine.

The AllocMem field (Delphi) or method (C++) must specify a function that allocates the given number of bytes and returns a pointer to the newly allocated block, as invoked by the AllocMem routine.

The RegisterExpectedMemoryLeak field (Delphi) or method (C++) must specify a function that registers a memory location that an application has allocated and does not expect to free, as invoked by the RegisterExpectedMemoryLeak routine.

The UnregisterExpectedMemoryLeak field (Delphi) or method (C++) must specify a function that removes a memory location from the Memory Manager's list of expected memory leaks, as invoked by the UnregisterExpectedMemoryLeak routine.

See Also

AllocMem (see page 22413)

FreeMemory (see page 22466)

GetMemory (see page 22470)

GetMemoryManager (see page 22472)

ReallocMem (see page 22507)

RegisterExpectedMemoryLeak (see page 22508)

SetMemoryManager (see page 22523)

UnregisterExpectedMemoryLeak (see page 22539)

158.4.18 System.TMemoryManagerState Record

TMemoryManagerState holds statistics about memory allocations.

Pascal

```
TMemoryManagerState = packed record
  SmallBlockTypeStates: TSmallBlockTypeStates;
  AllocatedMediumBlockCount: Cardinal;
  TotalAllocatedMediumBlockSize: Cardinal;
  ReservedMediumBlockAddressSpace: Cardinal;
  AllocatedLargeBlockCount: Cardinal;
  TotalAllocatedLargeBlockSize: Cardinal;
  ReservedLargeBlockAddressSpace: Cardinal;
end;
```

C++

```
struct TMemoryManagerState {
  TSmallBlockTypeStates SmallBlockTypeStates;
  unsigned AllocatedMediumBlockCount;
  unsigned TotalAllocatedMediumBlockSize;
  unsigned ReservedMediumBlockAddressSpace;
  unsigned AllocatedLargeBlockCount;
  unsigned TotalAllocatedLargeBlockSize;
  unsigned ReservedLargeBlockAddressSpace;
};
```

File

System

Description

GetMemoryManagerState (see page 22472) returns a structure containing memory usage statistics for each block type - small, medium and large blocks. It returns this data in a TMemoryManagerState record. AllocMem (see page 22413) and GetMemory (see page 22470) can be used to allocate memory blocks. The record holds the following information :

Field	Meaning
SmallBlockTypeStates	An array of small memory block information records.
AllocatedMediumBlockCount	The number of medium sized memory blocks (blocks less than 264,749 bytes in size).
TotalAllocatedMediumBlockSize	The total number of bytes oiccupied by the medium memory blocks.
ReservedMediumBlockAddressSpace	The amount of memory reserved for medium memory blocks.
AllocatedLargeBlockCount	The number of large sized memory blocks (blocks greater than 264,748 bytes in size).
TotalAllocatedLargeBlockSize	The total number of bytes oiccupied by the large memory blocks.
ReservedLargeBlockAddressSpace	The amount of memory reserved for large memory blocks.

See Also

AllocMem (see page 22413)

GetMemory (see page 22470)

GetMemoryManagerState (see page 22472)

TSmallBlockTypeStates (see page 22620)

158.4.19 System.TMethod Record

TMethod stores Code and Data fields to represent a method.

Pascal

```
TMethod = record
  Code: Pointer;
  Data: Pointer;
end;
```

C++

```
struct TMethod {
  void * Code;
  void * Data;
};
```

File

System

Description

The TMethod stores the Code and Data pointers for a method. This type can be used in a type cast of a method pointer to access the code and data parts of the method pointer.

You can also furnish a TMethod variable by assigning Data a pointer to an object, and assigning Code using MethodAddress, specifying the method name as a string parameter to that method.

See Also

MethodAddress (see page 22306)

158.4.20 System.TMinimumBlockAlignment Enumeration

TMinimumBlockAlignment defines the mimimum byte alignment for memory block allocation.

Pascal

```
TMinimumBlockAlignment = (
  mba8Byte,
  mba16Byte
);
```

C++

```
enum TMinimumBlockAlignment {
  mba8Byte,
  mba16Byte
};
```

File

System

Description

The TMinimumBlockAlignment type defines either 8 byte (**mba8Byte**) or 16 byte (**mba16Byte**) block alignments. Namely, that the memory address is divisible exactly by 8 or 16 respectively.

See Also

AllocMem (see page 22413)

GetMemory (see page 22470)

158.4.21 System.TModuleUnloadRec Record

Pascal

```
TModuleUnloadRec = record
  Next: PModuleUnloadRec;
  Proc: TModuleUnloadProcLW;
end;
```

C++

```
struct TModuleUnloadRec {
  PModuleUnloadRec Next;
  TModuleUnloadProcLW Proc;
};
```

File

System

Description

This is record System.TModuleUnloadRec.

158.4.22 System.TResStringRec Record

Pascal

```
TResStringRec = packed record
  Module: ^Cardinal;
  Identifier: Integer;
end;
```

C++

```
struct TResStringRec {
  ^Cardinal Module;
  int Identifier;
};
```

File

System

Description

This is record System.TResStringRec.

158.4.23 System.TRuntimeError Enumeration

Pascal

```
TRuntimeError = (
  reNone,
  reOutOfMemory,
  reInvalidPtr,
  reDivByZero,
  reRangeError,
  reIntOverflow,
  reInvalidOp,
  reZeroDivide,
  reOverflow,
  reUnderflow,
  reInvalidCast,
  reAccessViolation,
  rePrivInstruction,
  reControlBreak,
  reStackOverflow,
  reVarTypeCast,
  reVarInvalidOp,
  reVarDispatch,
  reVarArrayCreate,
  reVarNotArray,
  reVarArrayBounds,
  reAssertionFailed,
  reExternalException,
  reIntfCastError,
  reSafeCallError
);
```

C++

```
enum TRuntimeError {
  reNone,
```

```
    reOutOfMemory,
    reInvalidPtr,
    reDivByZero,
    reRangeError,
    reIntOverflow,
    reInvalidOp,
    reZeroDivide,
    reOverflow,
    reUnderflow,
    reInvalidCast,
    reAccessViolation,
    rePrivInstruction,
    reControlBreak,
    reStackOverflow,
    reVarTypeCast,
    reVarInvalidOp,
    reVarDispatch,
    reVarArrayCreate,
    reVarNotArray,
    reVarArrayBounds,
    reAssertionFailed,
    reExternalException,
    reIntfCastError,
    reSafeCallError
  };
```

File

System

Description

This is record System.TRuntimeError.

158.4.24 System.TSmallBlockTypeState Record

TSmallBlockTypeState holds statistics about a small memory block.

Pascal

```
  TSmallBlockTypeState = packed record
    InternalBlockSize: Cardinal;
    UseableBlockSize: Cardinal;
    AllocatedBlockCount: Cardinal;
    ReservedAddressSpace: Cardinal;
  end;
```

C++

```
  struct TSmallBlockTypeState {
    unsigned InternalBlockSize;
    unsigned UseableBlockSize;
    unsigned AllocatedBlockCount;
    unsigned ReservedAddressSpace;
  };
```

File

System

Description

GetMemoryManagerState (see page 22472) returns a structure containing memory usage statistics for each block type - small, medium and large blocks. It returns this data in a TMemoryManagerState (see page 22564) record. Within that record is

held a TSmallBlockTypeStates (see page 22620) field - an array of TSmallBlockTypeState records. Each TSmallBlockTypeState record holds the following information :

Field	Meaning
InternalBlockSize	Actual memory allocated to each small memory block - at least 4 bytes of which are not available in the UseableBlockSize.
UseableBlockSize	The useable size of each allocated small memory block.
AllocatedBlockCount	The number of small memory blocks allocated of the given InternalBlockSize.
ReservedAddressSpace	The amount memory reserved for this group of small memory blocks.

See Also

TSmallBlockTypeStates (see page 22620)

GetMemoryManagerState (see page 22472)

158.4.25 System.TTextLineBreakStyle Enumeration

TTextLineBreakStyle indicates the default set of characters used to divide lines in text.

Pascal

```
TTextLineBreakStyle = (
  tlbsLF,
  tlbsCRLF
);
```

C++

```
enum TTextLineBreakStyle {
  tlbsLF,
  tlbsCRLF
};
```

File

System

Description

TTextLineBreakStyle can have one of the following values:

Value	Meaning
tlbsLF	Linefeed only
tlbsCRLF	Carriage return followed by Linefeed

158.4.26 System.TTextRec Record

TTextRec is the internal format for text files.

Pascal

```
TTextRec = packed record
  Handle: Integer;
```

```
    Mode: Word;
    Flags: Word;
    BufSize: Cardinal;
    BufPos: Cardinal;
    BufEnd: Cardinal;
    BufPtr: PChar;
    OpenFunc: Pointer;
    InOutFunc: Pointer;
    FlushFunc: Pointer;
    CloseFunc: Pointer;
    UserData: array[1..32] of Byte;
    Name: array[0..259] of Char;
    Buffer: TTextBuf;
  end;
```

C++

```
  struct TTextRec {
    int Handle;
    Word Mode;
    Word Flags;
    unsigned BufSize;
    unsigned BufPos;
    unsigned BufEnd;
    const char * BufPtr;
    void * OpenFunc;
    void * InOutFunc;
    void * FlushFunc;
    void * CloseFunc;
    array[1..32] of Byte UserData;
    array[0..259] of Char Name;
    TTextBuf Buffer;
  };
```

File

System

Description

TTextRec is the internal format of a Delphi variable of type text. You would never declare a variable of this type. However, you can use TTextRec to typecast a text file variable to access the internal data fields, such as the file name.

TextBuf is the type of the Buffer entry in the TTextRec record, and indicates the size of the buffer used.

Note: Do not use this unless you are familiar with writing text file device drivers for the Delphi language.

158.4.27 System.TVarArray Record

Pascal

```
  TVarArray = packed record
    DimCount: Word;
    Flags: Word;
    ElementSize: Integer;
    LockCount: Integer;
    Data: Pointer;
    Bounds: TVarArrayBoundArray;
  end;
```

C++

```
  struct TVarArray {
    Word DimCount;
```

```
    Word Flags;
    int ElementSize;
    int LockCount;
    void * Data;
    TVarArrayBoundArray Bounds;
  };
```

File

System

Description

This is record System.TVarArray.

158.4.28 System.TVarArrayBound Record

Pascal

```
  TVarArrayBound = packed record
    ElementCount: Integer;
    LowBound: Integer;
  end;
```

C++

```
  struct TVarArrayBound {
    int ElementCount;
    int LowBound;
  };
```

File

System

Description

This is record System.TVarArrayBound.

158.4.29 System.TVarData Record

TVarData represents Variant (see page 22307) information.

Pascal

```
  TVarData = packed record
    case Integer of
      0: (VType: TVarType;
          Reserved1: Word;
          Reserved2: Word;
          Reserved3: Word;);
      1: (VLongs: array[0..2] of LongInt);
          RawData: array [0..3] of LongInt););
      2: (VWords: array [0..6] of Word);
          VBytes: array [0..13] of Byte););
      varAny: (VAny: Pointer;);
      varArray: (VArray: PVarArray;);
      varBoolean: (VBoolean: WordBool;);
      varByRef: (VPointer: Pointer;);
      varByte: (VByte: Byte;);
      varCurrency: (VCurrency: Currency;);
```

```
    varDate: (VDate: TDateTime;);
    varDispatch: (VDispatch: Pointer;);
    varDouble: (VDouble: Double;);
    varError: (VError: HRESULT;);
    varInt64: (VInt64: Int64;);
    varInteger: (VInteger: Integer;);
    varLongWord: (VLongWord: LongWord;);
    varOleStr: (VOleStr: PWideChar;);
    varShortInt: (VShortInt: ShortInt;);
    varSingle: (VSingle: Single;);
    varSmallInt: (VSmallInt: SmallInt;);
    varString: (VString: Pointer;);
    varUnknown: (VUnknown: Pointer;);
    varWord: (VWord: Word;);
end;
```

C++

```
struct TVarData {
  TVarType VType;
  Word Reserved1;
  Word Reserved2;
  Word Reserved3;
  SmallInt VSmallInt;
  int VInteger;
  Single VSingle;
  Double VDouble;
  Currency VCurrency;
  TDateTime VDate;
  PWideChar VOleStr;
  void * VDispatch;
  HRESULT VError;
  WordBool VBoolean;
  void * VUnknown;
  ShortInt VShortInt;
  Byte VByte;
  Word VWord;
  LongWord VLongWord;
  Int64 VInt64;
  void * VString;
  void * VAny;
  PVarArray VArray;
  void * VPointer;
  array[0..2] of LongInt) VLongs;
  array [0..6] of Word) VWords;
  array [0..13] of Byte) VBytes;
  array [0..3] of LongInt) RawData;
};
```

File

System

Description

TVarData is the base class for Variant (see page 22307).

Note: The internal structure of a Variant (see page 22307) as designated by TVarData is identical to the Variant (see page 22307) type used by OLE and the Win32 APIs.

struct TVarData

```
  {
union
    {
      Word     VType;    // Delphi-compatible - Variant Type member
      VARTYPE  vt;       // tagVARIANT compatible member
    };
```

```
    Word Reserved1;
    Word Reserved2;
    Word Reserved3;
union
    {
      // Delphi-compatible Variant members
      Smallint      VSmallint;    //  iVal
      Integer       VInteger;     //  lVal
      Single        VSingle;      //  fltVal
      Double        VDouble;      //  dblVal
      CurrencyBase  VCurrency;    //  cyVal
      TDateTimeBase VDate;        //  date
      PWideChar     VOleStr;      //  bstrVal
      IDispatch*    VDispatch;    //  pdispVal
      HResult       VError;       //  scode
      WordBool      VBoolean;     //  boolVal
      IUnknown*     VUnknown;     //  punkVal
      Byte          VByte;        //  bVal
      Pointer       VString;      //  ??????
      PVarArray     VArray;       //  parray
      Pointer       VPointer;     //  byref
      // tagVARIANT compatible members (from OAIDL.H)
      // Allowing all types marked as [V] (may appear in a VARIANT to be initialized
      LONG          lVal;
      BYTE          bVal;
      SHORT         iVal;
      FLOAT         fltVal;
      DOUBLE        dblVal;
      VARIANT_BOOL  boolVal;
      SCODE         scode;
      CY            cyVal;
      DATE          date;
      BSTR          bstrVal;
      IUnknown     *punkVal;
      IDispatch    *pdispVal;
      SAFEARRAY    *parray;
      BYTE         *pbVal;SHORT        *piVal;
      LONG         *plVal;
      FLOAT        *pfltVal;
      DOUBLE       *pdblVal;
      VARIANT_BOOL *pboolVal;
      SCODE        *pscode;
      CY           *pcyVal;
      DATE         *pdate;
      BSTR         *pbstrVal;
      IUnknown    **ppunkVal;
      IDispatch   **ppdispVal;
      SAFEARRAY   **pparray;
      VARIANT      *pvarVal;
      PVOID         byref;
      CHAR          cVal;
      USHORT        uiVal;
      ULONG         ulVal;
      INT           intVal;
      UINT          uintVal;
      DECIMAL      *pdecVal;
      CHAR         *pcVal;
      USHORT       *puiVal;
      ULONG        *pulVal;
      INT          *pintVal;
      UINT         *puintVal;
    };
  };
```

The VType of TVarData contains the type code of the variant in the lower twelve bits (the bits defined by varTypeMask (see page 22689)).

The following values are used for VType:

#define varEmpty (see page 22683) (Byte)(0)

```
#define varNull (ByteByte_Type)(1)
#define varSmallint (ByteByte_Type)(2)
#define varInteger (ByteByte_Type)(3)
#define varSingle (ByteByte_Type)(4)
#define varDouble (ByteByte_Type)(5)
#define varCurrency (ByteByte_Type)(6)
#define varDate (ByteByte_Type)(7)
#define varOleStr (ByteByte_Type)(8)
#define varDispatch (ByteByte_Type)(9)
#define varError (ByteByte_Type)(10)
#define varBoolean (ByteByte_Type)(11)
#define varVariant (ByteByte_Type)(12)
#define varUnknown (ByteByte_Type)(13)
#define varByte (ByteByte_Type)(17)
#define varString (WordWord_Type)(256)
#define varTypeMask (WordWord_Type)(4095)
#define varArray (WordWord_Type)(8192)
#define varByRef (WordWord_Type)(16384)
#define varStrArg (ByteByte_Type)(72)
#define vtInteger (ByteByte_Type)(0)
#define vtBoolean (ByteByte_Type)(1)
#define vtChar (ByteByte_Type)(2)
#define vtExtended (ByteByte_Type)(3)
#define vtString (ByteByte_Type)(4)
#define vtPointer (ByteByte_Type)(5)
#define vtPChar (ByteByte_Type)(6)
#define vtObject (ByteByte_Type)(7)
#define vtClass (ByteByte_Type)(8)
#define vtWideChar (ByteByte_Type)(9)
#define vtPWideChar (ByteByte_Type)(10)
#define vtAnsiString (ByteByte_Type)(11)
#define vtCurrency (ByteByte_Type)(12)
#define vtVariant (ByteByte_Type)(13)
```

In addition, the varArray (see page 22679) bit may be set to indicate that the variant is an array, and the varByRef (see page 22680) bit may be set to indicate that the variant contains a reference as opposed to a value.

Word Reserved1;

```
    Word Reserved2;
    Word Reserved3;
```

The Reserved1, Reserved2, and Reserved3 members of TVarData are unused.

union { Smallint VSmallint;

```
      Integer VInteger;
      Single VSingle;
      Double VDouble;
      CurrencyBase VCurrency;
      TDateTimeBase VDate;
      PWideChar VOleStr;
      Ole2::IDispatch* VDispatch;
      Integer VError;
      WordBool VBoolean;
      Ole2::IUnknown* VUnknown;
      Byte VByte;
      Pointer VString;
      PVarArray VArray;
      Pointer VPointer;
    };
};
```

The contents of the remaining eight bytes of TVarData depend on the VType field. If neither the varArray (see page 22679) nor

the varByRef (see page 22680) bits are set, the variant contains a value of the given type.

If the varArray (see page 22679) bit is set, the variant contains a pointer to a TVarArray (see page 22570) structure that defines the specifics of the array. The type of each array element is given by the varTypeMask (see page 22689) bits in VType.

If the varByRef (see page 22680) bit is set, the variant contains a reference to a value of the type given by the varTypeMask (see page 22689) and varArray (see page 22679) bits in VType.

158.4.30 System.TVariantManager Record

TVariantManager defines entry points for variant conversion routines and other procedures that define variant behavior.

Pascal

```
TVariantManager = record
  VarClear: procedure(var V : Variant);
  VarCopy: procedure(var Dest: Variant; const Source: Variant);
  VarCopyNoInd: procedure;
  VarCast: procedure(var Dest: Variant; const Source: Variant; VarType: Integer);
  VarCastOle: procedure(var Dest: Variant; const Source: Variant; VarType: Integer);
  VarToInt: function(const V: Variant): Integer;
  VarToInt64: function(const V: Variant): Int64;
  VarToBool: function(const V: Variant): Boolean;
  VarToReal: function(const V: Variant): Extended;
  VarToCurr: function(const V: Variant): Currency;
  VarToPStr: procedure(var S; const V: Variant);
  VarToLStr: procedure(var S: string; const V: Variant);
  VarToWStr: procedure(var S: WideString; const V: Variant);
  VarToIntf: procedure(var Unknown: IInterface; const V: Variant);
  VarToDisp: procedure(var Dispatch: IDispatch; const V: Variant);
  VarToDynArray: procedure(var DynArray: Pointer; const V: Variant; TypeInfo: Pointer);
  VarFromInt: procedure(var V: Variant; const Value: Integer; const Range: ShortInt);
  VarFromInt64: procedure(var V: Variant; const Value: Int64);
  VarFromBool: procedure(var V: Variant; const Value: Boolean);
  VarFromReal: procedure;
  VarFromTDateTime: procedure;
  VarFromCurr: procedure;
  VarFromPStr: procedure(var V: Variant; const Value: ShortString);
  VarFromLStr: procedure(var V: Variant; const Value: string);
  VarFromWStr: procedure(var V: Variant; const Value: WideString);
  VarFromIntf: procedure(var V: Variant; const Value: IInterface);
  VarFromDisp: procedure(var V: Variant; const Value: IDispatch);
  VarFromDynArray: procedure(var V: Variant; const DynArray: Pointer; TypeInfo: Pointer);
  OleVarFromPStr: procedure(var V: OleVariant; const Value: ShortString);
  OleVarFromLStr: procedure(var V: OleVariant; const Value: string);
  OleVarFromVar: procedure(var V: OleVariant; const Value: Variant);
  OleVarFromInt: procedure(var V: OleVariant; const Value: Integer; const Range: ShortInt);
  OleVarFromInt64: procedure(var V: OleVariant; const Value: Int64);
  VarOp: procedure(var Left: Variant; const Right: Variant; OpCode: TVarOp);
  VarCmp: procedure(const Left, Right: TVarData; const OpCode: TVarOp);
  VarNeg: procedure(var V: Variant);
  VarNot: procedure(var V: Variant);
  DispInvoke: procedure(Dest: PVarData; const Source: TVarData; CallDesc: PCallDesc; Params:
Pointer);
  VarAddRef: procedure(var V: Variant);
  VarArrayRedim: procedure(var A : Variant; HighBound: Integer);
  VarArrayGet: function(var A: Variant; IndexCount: Integer; Indices: Integer): Variant;
  VarArrayPut: procedure(var A: Variant; const Value: Variant; IndexCount: Integer; Indices:
Integer);
  WriteVariant: function(var T: Text; const V: Variant; Width: Integer): Pointer;
  Write0Variant: function(var T: Text; const V: Variant): Pointer;
end;
```

C++

```
struct TVariantManager {
  procedure(var V : Variant) VarClear;
  procedure(var Dest: Variant; const Source: Variant) VarCopy;
  procedure VarCopyNoInd;
  procedure(var Dest: Variant; const Source: Variant; VarType: Integer) VarCast;
  procedure(var Dest: Variant; const Source: Variant; VarType: Integer) VarCastOle;
  function(const V: Variant): Integer VarToInt;
  function(const V: Variant): Int64 VarToInt64;
  function(const V: Variant): Boolean VarToBool;
  function(const V: Variant): Extended VarToReal;
  function(const V: Variant): Currency VarToCurr;
  procedure(var S; const V: Variant) VarToPStr;
  procedure(var S: string; const V: Variant) VarToLStr;
  procedure(var S: WideString; const V: Variant) VarToWStr;
  procedure(var Unknown: IInterface; const V: Variant) VarToIntf;
  procedure(var Dispatch: IDispatch; const V: Variant) VarToDisp;
  procedure(var DynArray: Pointer; const V: Variant; TypeInfo: Pointer) VarToDynArray;
  procedure(var V: Variant; const Value: Integer; const Range: ShortInt) VarFromInt;
  procedure(var V: Variant; const Value: Int64) VarFromInt64;
  procedure(var V: Variant; const Value: Boolean) VarFromBool;
  procedure VarFromReal;
  procedure VarFromTDateTime;
  procedure VarFromCurr;
  procedure(var V: Variant; const Value: ShortString) VarFromPStr;
  procedure(var V: Variant; const Value: string) VarFromLStr;
  procedure(var V: Variant; const Value: WideString) VarFromWStr;
  procedure(var V: Variant; const Value: IInterface) VarFromIntf;
  procedure(var V: Variant; const Value: IDispatch) VarFromDisp;
  procedure(var V: Variant; const DynArray: Pointer; TypeInfo: Pointer) VarFromDynArray;
  procedure(var V: OleVariant; const Value: ShortString) OleVarFromPStr;
  procedure(var V: OleVariant; const Value: string) OleVarFromLStr;
  procedure(var V: OleVariant; const Value: Variant) OleVarFromVar;
  procedure(var V: OleVariant; const Value: Integer; const Range: ShortInt) OleVarFromInt;
  procedure(var V: OleVariant; const Value: Int64) OleVarFromInt64;
  procedure(var Left: Variant; const Right: Variant; OpCode: TVarOp) VarOp;
  procedure(const Left, Right: TVarData; const OpCode: TVarOp) VarCmp;
  procedure(var V: Variant) VarNeg;
  procedure(var V: Variant) VarNot;
  procedure(Dest: PVarData; const Source: TVarData; CallDesc: PCallDesc; Params: Pointer)
DispInvoke;
  procedure(var V: Variant) VarAddRef;
  procedure(var A : Variant; HighBound: Integer) VarArrayRedim;
  function(var A: Variant; IndexCount: Integer; Indices: Integer): Variant VarArrayGet;
  procedure(var A: Variant; const Value: Variant; IndexCount: Integer; Indices: Integer)
VarArrayPut;
  function(var T: Text; const V: Variant; Width: Integer): Pointer WriteVariant;
  function(var T: Text; const V: Variant): Pointer Write0Variant;
};
```

File

System

Description

The TVariantManager type is used by the GetVariantManager (see page 22474) and SetVariantManager (see page 22527) procedures. It defines routines that perform Variant (see page 22307) type conversions and other tasks that implement Variant (see page 22307) behavior.

Note: Although a version of TVariantManager is declared in sysclass.h, it should not be used in C++ code. The use of a Variant (see page 22307) manager to customize the behavior of Variants (see page 23567) is not fully supported in C++.

See Also

GetVariantManager (see page 22474)

SetVariantManager (see page 22527)

158.4.31 System::TVarRec Record

Pascal

```
TVarRec = record
  case Byte of
    vtAnsiString: (VAnsiString: Pointer;);
    vtBoolean: (VBoolean: Boolean;);
    vtChar: (VChar: Char;);
    vtClass: (VClass: TClass;);
    vtCurrency: (VCurrency: PCurrency;);
    vtExtended: (VExtended: PExtended;);
    vtInt64: (VInt64: PInt64;);
    vtInteger: (VInteger: Integer;
                VType: Byte;);
    vtInterface: (VInterface: Pointer;);
    vtObject: (VObject: TObject;);
    vtPChar: (VPChar: PChar;);
    vtPointer: (VPointer: Pointer;);
    vtPWideChar: (VPWideChar: PWideChar;);
    vtString: (VString: PShortString;);
    vtVariant: (VVariant: PVariant;);
    vtWideChar: (VWideChar: WideChar;);
    vtWideString: (VWideString: Pointer;);
end;
```

C++

```
struct TVarRec {
  Integer VInteger;
  Byte VType;
  Boolean VBoolean;
  Char VChar;
  PExtended VExtended;
  PShortString VString;
  Pointer VPointer;
  PChar VPChar;
  TObject VObject;
  TClass VClass;
  WideChar VWideChar;
  PWideChar VPWideChar;
  Pointer VAnsiString;
  PCurrency VCurrency;
  PVariant VVariant;
  Pointer VInterface;
  Pointer VWideString;
  PInt64 VInt64;
};
```

File

System.pas

Description

This is class System::TVarRec.

158.4.32 System::TCallDesc Structure

Pascal

```
TCallDesc = struct TCallDesc@1;
```

C++

```
typedef struct {
  unsigned char CallType;
  unsigned char ArgCount;
  unsigned char NamedArgCount;
  unsigned char ArgTypes[256];
} TCallDesc, * PCallDesc;
```

File

systvar.h

Description

This is type System::TCallDesc.

158.4.33 System::TDispDesc Structure

Pascal

```
TDispDesc = struct TDispDesc@1;
```

C++

```
typedef struct {
  int DispID;
  unsigned char ResType;
  TCallDesc CallDesc;
} TDispDesc, * PDispDesc;
```

File

systvar.h

Description

This is type System::TDispDesc.

158.5 Types

The following table lists types in this documentation.

Types

Name	Description
ArrayOfboolean (see page 22581)	This is type System::ArrayOfboolean.
ArrayOfbyte (see page 22582)	This is type System::ArrayOfbyte.
ArrayOfdouble (see page 22582)	This is type System::ArrayOfdouble.
ArrayOfint (see page 22582)	This is type System::ArrayOfint.
ArrayOflong (see page 22583)	This is type System::ArrayOflong.
ArrayOfshort (see page 22583)	This is type System::ArrayOfshort.
ArrayOfstring (see page 22583)	This is type System::ArrayOfstring.
ArrayOfwidestring (see page 22584)	This is type System::ArrayOfwidestring.
GetPackageInfoTable (see page 22584)	This is type System.GetPackageInfoTable.
HGLOBAL (see page 22584)	This is type System.HGLOBAL.
HINST (see page 22585)	This is type System.HINST.
HMODULE (see page 22585)	This is type System.HMODULE.
HRESULT (see page 22585)	This is type System.HRESULT.
HRSRC (see page 22586)	This is type System.HRSRC.
IntegerArray (see page 22586)	This is type System.IntegerArray.
IUnknown (see page 22586)	IUnknown is a special base interface.
PackageInfo (see page 22587)	This is type System.PackageInfo.
PAnsiString (see page 22587)	PAnsiString is a pointer to an AnsiString (see page 22093) variable.
PBoolean (see page 22587)	PBoolean is a pointer to a Boolean variable.
PByte (see page 22588)	PByte is a pointer to a Byte variable.
PCallDesc (see page 22588)	This is type System::PCallDesc.
PCardinal (see page 22589)	PCardinal is a pointer to a Cardinal variable.
PComp (see page 22589)	PComp is a pointer to a Comp variable.
PCurrency (see page 22589)	PCurrency is a pointer to a Currency (see page 22145) variable.
PCVModInfo (see page 22590)	This is type System.PCVModInfo.
PDate (see page 22590)	PDate is a pointer to a Date variable.
PDateTime (see page 22591)	PDateTime is a pointer to a TDateTime (see page 22256) variable.
PDispatch (see page 22591)	This is type System.PDispatch.
PDispDesc (see page 22591)	This is type System::PDispDesc.
PDouble (see page 22592)	PDouble is a pointer to a Double variable.
PDynArrayTypeInfo (see page 22592)	This is type System.PDynArrayTypeInfo.
PError (see page 22592)	This is type System.PError.
PExtended (see page 22593)	PExtended is a pointer to an Extended variable.
PGUID (see page 22593)	PGUID is a pointer to a TGUID (see page 22558) variable.
PInitContext (see page 22593)	This is type System.PInitContext.
PInt64 (see page 22594)	PInt64 is a pointer to an Int64 variable.
PInteger (see page 22594)	PInteger is a pointer to an Integer variable.
PIntegerArray (see page 22595)	PIntegerArray is a pointer to an IntegerArray (see page 22586) variable.
PInterfaceEntry (see page 22595)	This is type System::PInterfaceEntry.
PInterfaceTable (see page 22595)	This is type System::PInterfaceTable.
PLibModule (see page 22596)	PLibModule is a pointer to a TLibModule (see page 22561) variable.
PLongint (see page 22596)	This is type System.PLongint.
PLongWord (see page 22596)	PLongWord is a pointer to a LongWord variable.
PMemoryManager (see page 22597)	PMemoryManager is a pointer to a TMemoryManager (see page 22562) variable.

PMemoryManagerEx (see page 22597)	PMemoryManagerEx is a pointer to a TMemoryManagerEx (see page 22563) variable.
PModuleUnloadRec (see page 22598)	This is type System.PModuleUnloadRec.
PointerArray (see page 22598)	This is type System.PointerArray.
POleVariant (see page 22598)	This is type System.POleVariant.
PPAnsiChar (see page 22599)	This is type System.PPAnsiChar.
PPChar (see page 22599)	This is type System.PPChar.
PPCharArray (see page 22599)	This is type System.PPCharArray.
PPDispatch (see page 22600)	This is type System.PPDispatch.
PPointer (see page 22600)	PPointer is a pointer to a Pointer variable.
PPointerArray (see page 22600)	PPointerArray is a pointer to a PointerArray (see page 22598) variable.
PPUnknown (see page 22601)	This is type System.PPUnknown.
PPWideChar (see page 22601)	This is type System.PPWideChar.
PResStringRec (see page 22601)	This is type System.PResStringRec.
PShortInt (see page 22602)	PShortInt is a pointer to a ShortInt variable.
PShortString (see page 22602)	PShortString is a pointer to a ShortString variable.
PSingle (see page 22602)	PSingle is a pointer to a Single variable.
PSmallInt (see page 22603)	PSmallInt is a pointer to a SmallInt variable.
PString (see page 22603)	PString is a pointer to a String variable.
PTextBuf (see page 22604)	This is type System.PTextBuf.
PUCS2Char (see page 22604)	This is type System.PUCS2Char.
PUCS4Char (see page 22604)	This is type System.PUCS4Char.
PUCS4CharArray (see page 22605)	This is type System.PUCS4CharArray.
PUnitEntryTable (see page 22605)	This is type System.PUnitEntryTable.
PUnknown (see page 22605)	PUnknown is a pointer to an IUnknown (see page 22586) variable.
PUTF8String (see page 22606)	This is type System.PUTF8String.
PVarArray (see page 22606)	This is type System.PVarArray.
PVarArrayBoundArray (see page 22606)	This is type System.PVarArrayBoundArray.
PVarArrayCoorArray (see page 22607)	This is type System.PVarArrayCoorArray.
PVarData (see page 22607)	This is type System.PVarData.
PVariant (see page 22607)	PVariant is a pointer to a Variant (see page 22307) variable.
PVariantManager (see page 22608)	This is type System.PVariantManager.
PVarRec (see page 22608)	This is type System::PVarRec.
PWideString (see page 22608)	PWideString is a pointer to a WideString (see page 22377) variable.
PWord (see page 22609)	PWord is a pointer to a Word variable.
PWordBool (see page 22609)	PWordBool is a pointer to a WordBool variable.
SCODE (see page 22609)	This is type System::SCODE.
TAssertErrorProc (see page 22610)	This is type System.TAssertErrorProc.
TBooleanDynArray (see page 22610)	This is type System::TBooleanDynArray.
TBoundArray (see page 22610)	This is type System.TBoundArray.
TByteDynArray (see page 22611)	This is type System::TByteDynArray.
TCardinalDynArray (see page 22611)	This is type System::TCardinalDynArray.
TClass (see page 22611)	TClass defines the metaclass for TObject (see page 22285).
TDateTime (see page 22612)	TDateTime represents a date-and-time value in the Delphi language.
TDLLProc (see page 22613)	This is type System.TDLLProc.
TDLLProcEx (see page 22613)	This is type System.TDLLProcEx.
TDoubleDynArray (see page 22614)	This is type System::TDoubleDynArray.
TEnumModuleFunc (see page 22614)	This is type System.TEnumModuleFunc.
TEnumModuleFuncLW (see page 22614)	This is type System.TEnumModuleFuncLW.
TFileIOFunc (see page 22615)	This is type System.TFileIOFunc.
THandle (see page 22615)	THandle is a general purpose data reference (handle).
TInt64DynArray (see page 22615)	This is type System::TInt64DynArray.
TIntegerDynArray (see page 22616)	This is type System::TIntegerDynArray.
TInterfacedClass (see page 22616)	TInterfacedClass defines the metaclass for TInterfacedObject (see page 22276).

TLongWordDynArray (see page 22616)	This is type System::TLongWordDynArray.
TMemoryMap (see page 22617)	TMemoryMap is a map of an application's address space as an array of TChunkStatus (see page 22554) values.
TModuleUnloadProc (see page 22617)	TModuleUnloadProc dehines a procedure that can be called during package unload.
TModuleUnloadProcLW (see page 22618)	This is type System.TModuleUnloadProcLW.
TPCharArray (see page 22618)	This is type System.TPCharArray.
TResourceHandle (see page 22619)	This is type System.TResourceHandle.
TSafeCallErrorProc (see page 22619)	This is type System.TSafeCallErrorProc.
TShortIntDynArray (see page 22619)	This is type System::TShortIntDynArray.
TSingleDynArray (see page 22620)	This is type System::TSingleDynArray.
TSmallBlockTypeStates (see page 22620)	TSmallBlockTypeStates is an array of small memory block information records.
TSmallIntDynArray (see page 22620)	This is type System::TSmallIntDynArray.
TStringDynArray (see page 22621)	This is type System::TStringDynArray.
TSystemThreadEndProc (see page 22621)	This is type System.TSystemThreadEndProc.
TSystemThreadFuncProc (see page 22621)	This is type System.TSystemThreadFuncProc.
TTextBuf (see page 22622)	This is type System.TTextBuf.
TTextIOFunc (see page 22622)	This is type System.TTextIOFunc.
TThreadFunc (see page 22622)	TThreadFunc defines a new thread of execution.
TUCS4CharArray (see page 22623)	This is type System.TUCS4CharArray.
TVarArrayBoundArray (see page 22623)	This is type System.TVarArrayBoundArray.
TVarArrayCoorArray (see page 22623)	This is type System.TVarArrayCoorArray.
TVarOp (see page 22624)	TVarOp defines a Variant (see page 22307) operation.
TVarType (see page 22625)	This is type System.TVarType.
TWideStringDynArray (see page 22625)	This is type System::TWideStringDynArray.
TWordDynArray (see page 22626)	This is type System::TWordDynArray.
UCS2Char (see page 22626)	This is type System.UCS2Char.
UCS4Char (see page 22626)	UCS4Char is used for working with 4–byte Unicode characters.
UCS4String (see page 22627)	UCS4String is used for working with 4–byte Unicode characters.
UnitEntryTable (see page 22627)	This is type System.UnitEntryTable.
UTF8String (see page 22627)	This is type System.UTF8String.

158.5.1 System::ArrayOfboolean Type

Pascal

```
ArrayOfboolean = DynamicArray<bool>;
```

C++

```
typedef DynamicArray<bool> ArrayOfboolean;
```

File

sysdyn.h

Description

This is type System::ArrayOfboolean.

158.5.2 System::ArrayOfbyte Type

Pascal

```
ArrayOfbyte = DynamicArray<Byte>;
```

C++

```
typedef DynamicArray<Byte> ArrayOfbyte;
```

File

sysdyn.h

Description

This is type System::ArrayOfbyte.

158.5.3 System::ArrayOfdouble Type

Pascal

```
ArrayOfdouble = DynamicArray<double>;
```

C++

```
typedef DynamicArray<double> ArrayOfdouble;
```

File

sysdyn.h

Description

This is type System::ArrayOfdouble.

158.5.4 System::ArrayOfint Type

Pascal

```
ArrayOfint = DynamicArray<int>;
```

C++

```
typedef DynamicArray<int> ArrayOfint;
```

File

sysdyn.h

Description

This is type System::ArrayOfint.

158.5.5 System::ArrayOflong Type

Pascal

```
ArrayOflong = DynamicArray<long>;
```

C++

```
typedef DynamicArray<long> ArrayOflong;
```

File

sysdyn.h

Description

This is type System::ArrayOflong.

158.5.6 System::ArrayOfshort Type

Pascal

```
ArrayOfshort = DynamicArray<short>;
```

C++

```
typedef DynamicArray<short> ArrayOfshort;
```

File

sysdyn.h

Description

This is type System::ArrayOfshort.

158.5.7 System::ArrayOfstring Type

Pascal

```
ArrayOfstring = DynamicArray<AnsiString>;
```

C++

```
typedef DynamicArray<AnsiString> ArrayOfstring;
```

File

sysdyn.h

Description

This is type System::ArrayOfstring.

158.5.8 System::ArrayOfwidestring Type

Pascal

```
ArrayOfwidestring = DynamicArray<WideString>;
```

C++

```
typedef DynamicArray<WideString> ArrayOfwidestring;
```

File

sysdyn.h

Description

This is type System::ArrayOfwidestring.

158.5.9 System.GetPackageInfoTable Type

Pascal

```
GetPackageInfoTable = function : PackageInfo;
```

C++

```
: PackageInfo ( GetPackageInfoTable)();
```

File

System

Description

This is type System.GetPackageInfoTable.

158.5.10 System.HGLOBAL Type

Pascal

```
HGLOBAL = THandle;
```

C++

```
THandle HGLOBAL;
```

File

System

Description

This is type System.HGLOBAL.

158.5.11 System.HINST Type

Pascal

```
HINST = THandle;
```

C++

```
THandle HINST;
```

File

System

Description

This is type System.HINST.

158.5.12 System.HMODULE Type

Pascal

```
HMODULE = HINST;
```

C++

```
HINST HMODULE;
```

File

System

Description

This is type System.HMODULE.

158.5.13 System.HRESULT Type

Pascal

```
HRESULT = type Longint;
```

C++

```
type Longint HRESULT;
```

File

System

Description

This is type System.HRESULT.

158.5.14 System.HRSRC Type

Pascal

```
HRSRC = THandle;
```

C++

```
THandle HRSRC;
```

File

System

Description

This is type System.HRSRC.

158.5.15 System.IntegerArray Type

Pascal

```
IntegerArray = array[0..$efffff] of Integer;
```

C++

```
array[0..$efffff] of Integer IntegerArray;
```

File

System

Description

This is type System.IntegerArray.

158.5.16 System.IUnknown Type

IUnknown is a special base interface.

Pascal

```
IUnknown = IInterface;
```

C++

```
IInterface IUnknown;
```

File

System

Description

IUnknown is a base interface with features that are important to C++ programmers and also programmers who utilize COM objects.

Note: In Delphi code, IUnknown is simply another name for IInterface (see page 22396). Descending from IUnknown instead of IInterface (see page 22396) informs the Delphi compiler that the interface must be compatible with COM objects.

Cross-platform applications can use interfaces descended from IUnknown, but must not use them to access COM objects, which are only available under Windows.

See Also

IInterface (see page 22396)

System::DelphiInterface (see page 22167)

158.5.17 System.PackageInfo Type

Pascal

```
PackageInfo = ^PackageInfoTable;
```

C++

```
^PackageInfoTable PackageInfo;
```

File

System

Description

This is type System.PackageInfo.

158.5.18 System.PAnsiString Type

PAnsiString is a pointer to an AnsiString (see page 22093) variable.

Pascal

```
PAnsiString = ^AnsiString;
```

C++

```
^AnsiString PAnsiString;
```

File

System

Description

PAnsiString is a pointer to an AnsiString (see page 22093) variable.

See Also

AnsiString (see page 22093)

158.5.19 System.PBoolean Type

PBoolean is a pointer to a Boolean variable.

Pascal

```
PBoolean = ^Boolean;
```

C++

```
^Boolean PBoolean;
```

File

System

Description

PBoolean is a pointer to a Boolean variable.

See Also

Boolean

158.5.20 System.PByte Type

PByte is a pointer to a Byte variable.

Pascal

```
PByte = ^Byte;
```

C++

```
^Byte PByte;
```

File

System

Description

PByte is a pointer to a Byte variable.

See Also

Byte

158.5.21 System::PCallDesc Type

Pascal

```
PCallDesc = ^TCallDesc;
```

C++

```
^TCallDesc PCallDesc;
```

File

System.pas

Description

This is type System::PCallDesc.

158.5.22 System.PCardinal Type

PCardinal is a pointer to a Cardinal variable.

Pascal

```
PCardinal = ^Cardinal;
```

C++

```
^Cardinal PCardinal;
```

File

System

Description

PCardinal is a pointer to a Cardinal variable.

See Also

Cardinal

158.5.23 System.PComp Type

PComp is a pointer to a Comp variable.

Pascal

```
PComp = ^Comp;
```

C++

```
^Comp PComp;
```

File

System

Description

PComp is a pointer to a Comp variable.

Note: The Comp data type is deprecated.

See Also

Comp

158.5.24 System.PCurrency Type

PCurrency is a pointer to a Currency (see page 22145) variable.

Pascal

```
PCurrency = ^Currency;
```

C++

```
^Currency PCurrency;
```

File

System

Description

PCurrency is a pointer to a Currency (see page 22145) variable.

See Also

Currency (see page 22145)

158.5.25 System.PCVModInfo Type

Pascal

```
PCVModInfo = ^TCVModInfo;
```

C++

```
^TCVModInfo PCVModInfo;
```

File

System

Description

This is type System.PCVModInfo.

158.5.26 System.PDate Type

PDate is a pointer to a Date variable.

Pascal

```
PDate = ^Double;
```

C++

```
^Double PDate;
```

File

System

Description

PDate is a pointer to a Date variable.

See Also

Date

158.5.27 System.PDateTime Type

PDateTime is a pointer to a TDateTime (see page 22256) variable.

Pascal

```
PDateTime = ^TDateTime;
```

C++

```
^TDateTime PDateTime;
```

File

System

Description

PDateTime is a pointer to a TDateTime (see page 22256) variable.

See Also

TDateTime (see page 22612)

158.5.28 System.PDispatch Type

Pascal

```
PDispatch = ^IDispatch;
```

C++

```
^IDispatch PDispatch;
```

File

System

Description

This is type System.PDispatch.

158.5.29 System::PDispDesc Type

Pascal

```
PDispDesc = ^TDispDesc;
```

C++

```
^TDispDesc PDispDesc;
```

File

System.pas

Description

This is type System::PDispDesc.

158.5.30 System.PDouble Type

PDouble is a pointer to a Double variable.

Pascal

```
PDouble = ^Double;
```

C++

```
^Double PDouble;
```

File

System

Description

PDouble is a pointer to a Double variable.

See Also

Double

158.5.31 System.PDynArrayTypeInfo Type

Pascal

```
PDynArrayTypeInfo = ^TDynArrayTypeInfo;
```

C++

```
^TDynArrayTypeInfo PDynArrayTypeInfo;
```

File

System

Description

This is type System.PDynArrayTypeInfo.

158.5.32 System.PError Type

Pascal

```
PError = ^LongWord;
```

C++

```
^LongWord PError;
```

File

System

Description

This is type System.PError.

158.5.33 System.PExtended Type

PExtended is a pointer to an Extended variable.

Pascal

```
PExtended = ^Extended;
```

C++

```
^Extended PExtended;
```

File

System

Description

PExtended is a pointer to an Extended variable.

See Also

Extended

158.5.34 System.PGUID Type

PGUID is a pointer to a TGUID (see page 22558) variable.

Pascal

```
PGUID = ^TGUID;
```

C++

```
^TGUID PGUID;
```

File

System

Description

PGUID is a pointer to a TGUID (see page 22558) variable.

See Also

TGUID (see page 22558)

158.5.35 System.PInitContext Type

Pascal

```
PInitContext = ^TInitContext;
```

C++

```
^TInitContext PInitContext;
```

File

System

Description

This is type System.PInitContext.

158.5.36 System.PInt64 Type

PInt64 is a pointer to an Int64 variable.

Pascal

```
PInt64 = ^Int64;
```

C++

```
^Int64 PInt64;
```

File

System

Description

PInt64 is a pointer to an Int64 variable.

See Also

Int64

158.5.37 System.PInteger Type

PInteger is a pointer to an Integer variable.

Pascal

```
PInteger = ^Integer;
```

C++

```
^Integer PInteger;
```

File

System

Description

PInteger is a pointer to an Integer variable.

See Also

Integer

158.5.38 System.PIntegerArray Type

PIntegerArray is a pointer to an IntegerArray (see page 22586) variable.

Pascal

```
PIntegerArray = ^IntegerArray;
```

C++

```
^IntegerArray PIntegerArray;
```

File

System

Description

PIntegerArray is a pointer to an IntegerArray (see page 22586) variable.

See Also

IntegerArray (see page 22586)

158.5.39 System::PInterfaceEntry Type

Pascal

```
PInterfaceEntry = ^TInterfaceEntry;
```

C++

```
^TInterfaceEntry PInterfaceEntry;
```

File

System.pas

Description

This is type System::PInterfaceEntry.

158.5.40 System::PInterfaceTable Type

Pascal

```
PInterfaceTable = ^TInterfaceTable;
```

C++

```
^TInterfaceTable PInterfaceTable;
```

File

System.pas

Description

This is type System::PInterfaceTable.

158.5.41 System.PLibModule Type

PLibModule is a pointer to a TLibModule (see page 22561) variable.

Pascal

```
PLibModule = ^TLibModule;
```

C++

```
^TLibModule PLibModule;
```

File

System

Description

PLibModule is a pointer to a TLibModule (see page 22561) variable.

See Also

TLibModule (see page 22561)

158.5.42 System.PLongint Type

Pascal

```
PLongint = ^Longint;
```

C++

```
^Longint PLongint;
```

File

System

Description

This is type System.PLongint.

158.5.43 System.PLongWord Type

PLongWord is a pointer to a LongWord variable.

Pascal

```
PLongWord = ^LongWord;
```

C++

```
^LongWord PLongWord;
```

File

System

Description

PLongWord is a pointer to a LongWord variable.

See Also

LongWord

158.5.44 System.PMemoryManager Type

PMemoryManager is a pointer to a TMemoryManager (see page 22562) variable.

Pascal

```
PMemoryManager = ^TMemoryManager;
```

C++

```
^TMemoryManager PMemoryManager;
```

File

System

Description

PMemoryManager is a pointer to a TMemoryManager (see page 22562) variable.

See Also

TMemoryManager (see page 22562)

158.5.45 System.PMemoryManagerEx Type

PMemoryManagerEx is a pointer to a TMemoryManagerEx (see page 22563) variable.

Pascal

```
PMemoryManagerEx = ^TMemoryManagerEx;
```

C++

```
^TMemoryManagerEx PMemoryManagerEx;
```

File

System

Description

PMemoryManagerEx is a pointer to a TMemoryManagerEx (see page 22563) variable.

See Also

TMemoryManagerEx (see page 22563)

158.5.46 System.PModuleUnloadRec Type

Pascal

```
PModuleUnloadRec = ^TModuleUnloadRec;
```

C++

```
^TModuleUnloadRec PModuleUnloadRec;
```

File

System

Description

This is type System.PModuleUnloadRec.

158.5.47 System.PointerArray Type

Pascal

```
PointerArray = array [0..512*1024*1024 - 2] of Pointer;
```

C++

```
array [0..512*1024*1024 - 2] of Pointer PointerArray;
```

File

System

Description

This is type System.PointerArray.

158.5.48 System.POleVariant Type

Pascal

```
POleVariant = ^OleVariant;
```

C++

```
^OleVariant POleVariant;
```

File

System

Description

This is type System.POleVariant.

158.5.49 System.PPAnsiChar Type

Pascal

```
PPAnsiChar = PPChar;
```

C++

```
PPChar PPAnsiChar;
```

File

System

Description

This is type System.PPAnsiChar.

158.5.50 System.PPChar Type

Pascal

```
PPChar = ^PChar;
```

C++

```
^PChar PPChar;
```

File

System

Description

This is type System.PPChar.

158.5.51 System.PPCharArray Type

Pascal

```
PPCharArray = ^TPCharArray;
```

C++

```
^TPCharArray PPCharArray;
```

File

System

Description

This is type System.PPCharArray.

158.5.52 System.PPDispatch Type

Pascal

```
PPDispatch = ^PDispatch;
```

C++

```
^PDispatch PPDispatch;
```

File

System

Description

This is type System.PPDispatch.

158.5.53 System.PPointer Type

PPointer is a pointer to a Pointer variable.

Pascal

```
PPointer = ^Pointer;
```

C++

```
^Pointer PPointer;
```

File

System

Description

PPointer is a pointer to a Pointer variable.

See Also

Pointer

158.5.54 System.PPointerArray Type

PPointerArray is a pointer to a PointerArray (see page 22598) variable.

Pascal

```
PPointerArray = ^PointerArray;
```

C++

```
^PointerArray PPointerArray;
```

File

System

Description

PPointerArray is a pointer to a PointerArray (see page 22598) variable.

See Also

PointerArray (see page 22598)

158.5.55 System.PPUnknown Type

Pascal

```
PPUnknown = ^PUnknown;
```

C++

```
^PUnknown PPUnknown;
```

File

System

Description

This is type System.PPUnknown.

158.5.56 System.PPWideChar Type

Pascal

```
PPWideChar = ^PWideChar;
```

C++

```
^PWideChar PPWideChar;
```

File

System

Description

This is type System.PPWideChar.

158.5.57 System.PResStringRec Type

Pascal

```
PResStringRec = ^TResStringRec;
```

C++

```
^TResStringRec PResStringRec;
```

File

System

Description

This is type System.PResStringRec.

158.5.58 System.PShortInt Type

PShortInt is a pointer to a ShortInt variable.

Pascal

```
PShortInt = ^ShortInt;
```

C++

```
^ShortInt PShortInt;
```

File

System

Description

PShortInt is a pointer to a ShortInt variable.

See Also

ShortInt

158.5.59 System.PShortString Type

PShortString is a pointer to a ShortString variable.

Pascal

```
PShortString = ^ShortString;
```

C++

```
^ShortString PShortString;
```

File

System

Description

PShortString is a pointer to a ShortString variable.

See Also

ShortString

158.5.60 System.PSingle Type

PSingle is a pointer to a Single variable.

Pascal

```
PSingle = ^Single;
```

C++

```
^Single PSingle;
```

File

System

Description

PSingle is a pointer to a Single variable.

See Also

Single

158.5.61 System.PSmallInt Type

PSmallInt is a pointer to a SmallInt variable.

Pascal

```
PSmallInt = ^SmallInt;
```

C++

```
^SmallInt PSmallInt;
```

File

System

Description

PSmallInt is a pointer to a SmallInt variable.

See Also

SmallInt

158.5.62 System.PString Type

PString is a pointer to a String variable.

Pascal

```
PString = PAnsiString;
```

C++

```
PAnsiString PString;
```

File

System

Description

PString is a pointer to a String variable.

See Also

String

158.5.63 System.PTextBuf Type

Pascal

```
PTextBuf = ^TTextBuf;
```

C++

```
^TTextBuf PTextBuf;
```

File

System

Description

This is type System.PTextBuf.

158.5.64 System.PUCS2Char Type

Pascal

```
PUCS2Char = PWideChar;
```

C++

```
PWideChar PUCS2Char;
```

File

System

Description

This is type System.PUCS2Char.

158.5.65 System.PUCS4Char Type

Pascal

```
PUCS4Char = ^UCS4Char;
```

C++

```
^UCS4Char PUCS4Char;
```

File

System

Description

This is type System.PUCS4Char.

158.5.66 System.PUCS4CharArray Type

Pascal

```
PUCS4CharArray = ^TUCS4CharArray;
```

C++

```
^TUCS4CharArray PUCS4CharArray;
```

File

System

Description

This is type System.PUCS4CharArray.

158.5.67 System.PUnitEntryTable Type

Pascal

```
PUnitEntryTable = ^UnitEntryTable;
```

C++

```
^UnitEntryTable PUnitEntryTable;
```

File

System

Description

This is type System.PUnitEntryTable.

158.5.68 System.PUnknown Type

PUnknown is a pointer to an IUnknown (see page 22586) variable.

Pascal

```
PUnknown = ^IUnknown;
```

C++

```
^IUnknown PUnknown;
```

File

System

Description

PUnknown is a pointer to an IUnknown (see page 22586) variable.

See Also

IUnknown (see page 22586)

158.5.69 System.PUTF8String Type

Pascal

```
PUTF8String = ^UTF8String;
```

C++

```
^UTF8String PUTF8String;
```

File

System

Description

This is type System.PUTF8String.

158.5.70 System.PVarArray Type

Pascal

```
PVarArray = ^TVarArray;
```

C++

```
^TVarArray PVarArray;
```

File

System

Description

This is type System.PVarArray.

158.5.71 System.PVarArrayBoundArray Type

Pascal

```
PVarArrayBoundArray = ^TVarArrayBoundArray;
```

C++

```
^TVarArrayBoundArray PVarArrayBoundArray;
```

File

System

Description

This is type System.PVarArrayBoundArray.

158.5.72 System.PVarArrayCoorArray Type

Pascal

```
PVarArrayCoorArray = ^TVarArrayCoorArray;
```

C++

```
^TVarArrayCoorArray PVarArrayCoorArray;
```

File

System

Description

This is type System.PVarArrayCoorArray.

158.5.73 System.PVarData Type

Pascal

```
PVarData = ^TVarData;
```

C++

```
^TVarData PVarData;
```

File

System

Description

This is type System.PVarData.

158.5.74 System.PVariant Type

PVariant is a pointer to a Variant (see page 22307) variable.

Pascal

```
PVariant = ^Variant;
```

C++

```
^Variant PVariant;
```

File

System

Description

PVariant is a pointer to a Variant (see page 22307) variable.

See Also

Variant (see page 22307)

158.5.75 System.PVariantManager Type

Pascal

```
PVariantManager = ^TVariantManager;
```

C++

```
^TVariantManager PVariantManager;
```

File

System

Description

This is type System.PVariantManager.

158.5.76 System::PVarRec Type

Pascal

```
PVarRec = ^TVarRec;
```

C++

```
^TVarRec PVarRec;
```

File

System.pas

Description

This is type System::PVarRec.

158.5.77 System.PWideString Type

PWideString is a pointer to a WideString (see page 22377) variable.

Pascal

```
PWideString = ^WideString;
```

C++

```
^WideString PWideString;
```

File

System

Description

PWideString is a pointer to a WideString (see page 22377) variable.

See Also

WideString (see page 22377)

158.5.78 System.PWord Type

PWord is a pointer to a Word variable.

Pascal

```
PWord = ^Word;
```

C++

```
^Word PWord;
```

File

System

Description

PWord is a pointer to a Word variable.

See Also

Word

158.5.79 System.PWordBool Type

PWordBool is a pointer to a WordBool variable.

Pascal

```
PWordBool = ^WordBool;
```

C++

```
^WordBool PWordBool;
```

File

System

Description

PWordBool is a pointer to a WordBool variable.

See Also

WordBool

158.5.80 System::SCODE Type

Pascal

```
SCODE = uint32_t;
```

C++

```
typedef uint32_t SCODE;
```

File

sysvari.h

Description

This is type System::SCODE.

158.5.81 System.TAssertErrorProc Type

Pascal

```
TAssertErrorProc = procedure (const Message, Filename: string; LineNumber: Integer; ErrorAddr:
Pointer);
```

C++

```
(const Message, Filename: string; LineNumber: Integer; ErrorAddr: Pointer) (
TAssertErrorProc)();
```

File

System

Description

This is type System.TAssertErrorProc.

158.5.82 System::TBooleanDynArray Type

Pascal

```
TBooleanDynArray = DynamicArray<Boolean>;
```

C++

```
typedef DynamicArray<Boolean> TBooleanDynArray;
```

File

sysdyn.h

Description

This is type System::TBooleanDynArray.

158.5.83 System.TBoundArray Type

Pascal

```
TBoundArray = array of Integer;
```

C++

```
array of Integer TBoundArray;
```

File

System

Description

This is type System.TBoundArray.

158.5.84 System::TByteDynArray Type

Pascal

```
TByteDynArray = DynamicArray<Byte>;
```

C++

```
typedef DynamicArray<Byte> TByteDynArray;
```

File

sysdyn.h

Description

This is type System::TByteDynArray.

158.5.85 System::TCardinalDynArray Type

Pascal

```
TCardinalDynArray = DynamicArray<Cardinal>;
```

C++

```
typedef DynamicArray<Cardinal> TCardinalDynArray;
```

File

sysdyn.h

Description

This is type System::TCardinalDynArray.

158.5.86 System.TClass Type

TClass defines the metaclass for TObject (see page 22285).

Pascal

```
TClass = class of TObject;
```

C++

```
TObject * TClass;
```

File

System

Description

TClass is the metaclass for TObject (see page 22285). Its value is the class reference for TObject (see page 22285) or for

one of its descendants.

Delphi Examples:

```
{
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button's class and the names of its
parent classes are added to the list box.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ClassRef: TClass;
begin
  ListBox1.Clear;
  ClassRef := Sender.ClassType;
  while ClassRef <> nil do
  begin
    ListBox1.Items.Add(ClassRef.ClassName);
    ClassRef := ClassRef.ClassParent;
  end;
end;
{
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
}
```

See Also

TObject (see page 22285)

158.5.87 System.TDateTime Type

TDateTime represents a date-and-time value in the Delphi language.

Pascal

```
TDateTime = type Double;
```

C++

```
type Double TDateTime;
```

File

System

Description

In Delphi, TDateTime is a type that maps to a Double. In C++, the TDateTime class corresponds to the Delphi TDateTime type.

The integral part of a Delphi TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of the TDateTime value is fraction of a 24 hour day that has elapsed.

Following are some examples of TDateTime values and their corresponding dates and times:

0	12/30/1899 12:00 am
2.75	1/1/1900 6:00 pm
-1.25	12/29/1899 6:00 am
35065	1/1/1996 12:00 am

To find the fractional number of days between two dates, simply subtract the two values, unless one of the TDateTime values is negative. Similarly, to increment a date and time value by a certain fractional number of days, add the fractional number to the date and time value if the TDateTime value is positive.

When working with negative TDateTime values, computations must handle time portion separately. The fractional part reflects the fraction of a 24-hour day without regard to the sign of the TDateTime value. For example, 6:00 am on 12/29/1899 is –1.25, not –1 + 0.25, which would be –0.75. There are no TDateTime values between –1 and 0.

Note: Delphi 1.0 calculated the date from year 1 instead of from 1899. To convert a Delphi 1.0 date to a TDateTime value in later versions of the Delphi language, subtract 693594.0 from the Delphi 1.0 date.

See Also

TTimeStamp

TDate

TSQLTimeStamp

158.5.88 System.TDLLProc Type

Pascal

```
TDLLProc = procedure (Reason: Integer);
```

C++

```
(Reason: Integer) ( TDLLProc)();
```

File

System

Description

This is type System.TDLLProc.

158.5.89 System.TDLLProcEx Type

Pascal

```
TDLLProcEx = procedure (Reason: Integer; Reserved: Integer);
```

C++

```
(Reason: Integer; Reserved: Integer) ( TDLLProcEx)();
```

File

System

Description

This is type System.TDLLProcEx.

158.5.90 System::TDoubleDynArray Type

Pascal

```
TDoubleDynArray = DynamicArray<Double>;
```

C++

```
typedef DynamicArray<Double> TDoubleDynArray;
```

File

sysdyn.h

Description

This is type System::TDoubleDynArray.

158.5.91 System.TEnumModuleFunc Type

Pascal

```
TEnumModuleFunc = function (HInstance: Integer; Data: Pointer): Boolean;
```

C++

```
(HInstance: Integer; Data: Pointer): Boolean ( TEnumModuleFunc)();
```

File

System

Description

This is type System.TEnumModuleFunc.

158.5.92 System.TEnumModuleFuncLW Type

Pascal

```
TEnumModuleFuncLW = function (HInstance: LongWord; Data: Pointer): Boolean;
```

C++

```
(HInstance: LongWord; Data: Pointer): Boolean ( TEnumModuleFuncLW)();
```

File

System

Description

This is type System.TEnumModuleFuncLW.

158.5.93 System.TFileIOFunc Type

Pascal

```
TFileIOFunc = function (var F: TFileRec): Integer;
```

C++

```
(var F: TFileRec): Integer ( TFileIOFunc)();
```

File

System

Description

This is type System.TFileIOFunc.

158.5.94 System.THandle Type

THandle is a general purpose data reference (handle).

Pascal

```
THandle = LongWord;
```

C++

```
LongWord THandle;
```

File

System

Description

For example, the SaveToClipboardFormat uses a THandle variable to hold a handle to the data of a bit map. SetAsHandle can then be used to store the bit map data, via its handle, to the clipboard.

Handles are used extensively in communication with Windows APIs.

See Also

SaveToClipboardFormat (see page 14415)

SetAsHandle (see page 3343)

158.5.95 System::TInt64DynArray Type

Pascal

```
TInt64DynArray = DynamicArray<__int64>;
```

C++

```
typedef DynamicArray<__int64> TInt64DynArray;
```

File

sysdyn.h

Description

This is type System::TInt64DynArray.

158.5.96 System::TIntegerDynArray Type

Pascal

```
TIntegerDynArray = DynamicArray<Integer>;
```

C++

```
typedef DynamicArray<Integer> TIntegerDynArray;
```

File

sysdyn.h

Description

This is type System::TIntegerDynArray.

158.5.97 System.TInterfacedClass Type

TInterfacedClass defines the metaclass for TInterfacedObject (see page 22276).

Pascal

```
TInterfacedClass = class of TInterfacedObject;
```

C++

```
TInterfacedObject TInterfacedClass;
```

File

System

Description

TInterfacedClass is the metaclass for TInterfacedObject (see page 22276). Its value is the class reference for TInterfacedObject (see page 22276) or for one of its descendants.

See Also

TInterfacedObject (see page 22276)

158.5.98 System::TLongWordDynArray Type

Pascal

```
TLongWordDynArray = DynamicArray<unsigned>;
```

C++

```
typedef DynamicArray<unsigned> TLongWordDynArray;
```

File

sysdyn.h

Description

This is type System::TLongWordDynArray.

158.5.99 System.TMemoryMap Type

TMemoryMap is a map of an application's address space as an array of TChunkStatus (see page 22554) values.

Pascal

```
TMemoryMap = array[0..65535] of TChunkStatus;
```

C++

```
array[0..65535] of TChunkStatus TMemoryMap;
```

File

System

Description

GetMemoryMap (see page 22473) returns a map of the address space of the current process. For the purposes of the map, the entire 4GB address space is broken down into 64K chunks. There are 65536 such chunks. TMemoryMap obtains a status value for each of them in the TMemoryMap array. Each TMemoryMap array element may have the following value :

Status value	Meaning
csUnallocated	Free
csAllocated	In use by the process
csReserved	Reserved for future use by the process
csSysAllocated	In use by the operating system
csSysReserved	Reserved for future use by the operating system

See Also

GetMemoryMap (see page 22473)

158.5.100 System.TModuleUnloadProc Type

TModuleUnloadProc dehines a procedure that can be called during package unload.

Pascal

```
TModuleUnloadProc = procedure (HInstance: Integer);
```

C++

```
(HInstance: Integer) ( TModuleUnloadProc)();
```

File

System

Description

Before Delphi unloads a package that was used by your application, it can optionally call your TModuleUnloadProcLW (see page 22618) routine. In essence, AddModuleUnloadProc (see page 22411) adds your callback routine to the head of the list of internal routines that Delphi invokes to perform the unload.

Proc is the Handle to the DLL Package instance being unloaded.

The LW part of the name refers to the Long Word type of package handle. This is in line with Windows handles, and the THandle (see page 22615) type.

See Also

AddModuleUnloadProcLW

TModuleUnloadProcLW (see page 22618)

158.5.101 System.TModuleUnloadProcLW Type

Pascal

```
TModuleUnloadProcLW = procedure (HInstance: LongWord);
```

C++

```
(HInstance: LongWord) ( TModuleUnloadProcLW)();
```

File

System

Description

This is type System.TModuleUnloadProcLW.

158.5.102 System.TPCharArray Type

Pascal

```
TPCharArray = array[0..(MaxLongint div SizeOf(PChar))-1] of PChar;
```

C++

```
array[0..(MaxLongint div SizeOf(PChar))-1] of PChar TPCharArray;
```

File

System

Description

This is type System.TPCharArray.

158.5.103 System.TResourceHandle Type

Pascal

```
TResourceHandle = HRSRC;
```

C++

```
HRSRC TResourceHandle;
```

File

System

Description

This is type System.TResourceHandle.

158.5.104 System.TSafeCallErrorProc Type

Pascal

```
TSafeCallErrorProc = procedure (ErrorCode: HResult; ErrorAddr: Pointer);
```

C++

```
(ErrorCode: HResult; ErrorAddr: Pointer) ( TSafeCallErrorProc)();
```

File

System

Description

This is type System.TSafeCallErrorProc.

158.5.105 System::TShortIntDynArray Type

Pascal

```
TShortIntDynArray = DynamicArray<Shortint>;
```

C++

```
typedef DynamicArray<Shortint> TShortIntDynArray;
```

File

sysdyn.h

Description

This is type System::TShortIntDynArray.

158.5.106 System::TSingleDynArray Type

Pascal

```
TSingleDynArray = DynamicArray<Single>;
```

C++

```
typedef DynamicArray<Single> TSingleDynArray;
```

File

sysdyn.h

Description

This is type System::TSingleDynArray.

158.5.107 System.TSmallBlockTypeStates Type

TSmallBlockTypeStates is an array of small memory block information records.

Pascal

```
TSmallBlockTypeStates = array[0..NumSmallBlockTypes - 1] of TSmallBlockTypeState;
```

C++

```
array[0..NumSmallBlockTypes - 1] of TSmallBlockTypeState TSmallBlockTypeStates;
```

File

System

Description

GetMemoryManagerState (see page 22472) returns a structure containing memory usage statistics for each block type - small, medium and large blocks. It returns this data in a TMemoryManagerState (see page 22564) record. Within that record is held a TSmallBlockTypeStates field - an array of TSmallBlockTypeState (see page 22568) records. There are NumSmallBlockTypes (see page 22666) (55) different small memory block sizes, and therefore 55 TSmallBlockTypeState (see page 22568) records in this array.

See Also

GetMemoryManagerState (see page 22472)

NumSmallBlockTypes (see page 22666)

TSmallBlockTypeState (see page 22568)

158.5.108 System::TSmallIntDynArray Type

Pascal

```
TSmallIntDynArray = DynamicArray<Smallint>;
```

C++

```
typedef DynamicArray<Smallint> TSmallIntDynArray;
```

File

sysdyn.h

Description

This is type System::TSmallIntDynArray.

158.5.109 System::TStringDynArray Type

Pascal

```
TStringDynArray = DynamicArray<String>;
```

C++

```
typedef DynamicArray<String> TStringDynArray;
```

File

sysdyn.h

Description

This is type System::TStringDynArray.

158.5.110 System.TSystemThreadEndProc Type

Pascal

```
TSystemThreadEndProc = procedure (ExitCode: Integer);
```

C++

```
(ExitCode: Integer) ( TSystemThreadEndProc)();
```

File

System

Description

This is type System.TSystemThreadEndProc.

158.5.111 System.TSystemThreadFuncProc Type

Pascal

```
TSystemThreadFuncProc = function (ThreadFunc: TThreadFunc; Parameter: Pointer): Pointer;
```

C++

```
(ThreadFunc: TThreadFunc; Parameter: Pointer): Pointer ( TSystemThreadFuncProc)();
```

File

System

Description

This is type System.TSystemThreadFuncProc.

158.5.112 System.TTextBuf Type

Pascal

```
TTextBuf = array[0..127] of Char;
```

C++

```
array[0..127] of Char TTextBuf;
```

File

System

Description

This is type System.TTextBuf.

158.5.113 System.TTextIOFunc Type

Pascal

```
TTextIOFunc = function (var F: TTextRec): Integer;
```

C++

```
(var F: TTextRec): Integer ( TTextIOFunc)();
```

File

System

Description

This is type System.TTextIOFunc.

158.5.114 System.TThreadFunc Type

TThreadFunc defines a new thread of execution.

Pascal

```
TThreadFunc = function (Parameter: Pointer): Integer;
```

C++

```
(Parameter: Pointer): Integer ( TThreadFunc)();
```

File

System

Description

Use BeginThread (see page 22422) or a TThread object to spawn separate threads of execution. BeginThread (see page 22422) spawns a new thread of execution via the TThreadFunc defined function.

Parameter is a pointer to any data you want to pass to the new thread.

When the thread completes, it returns a status code.

See Also

BeginThread (see page 22422)

TThread

158.5.115 System.TUCS4CharArray Type

Pascal

```
TUCS4CharArray = array [0..$effffff] of UCS4Char;
```

C++

```
array [0..$effffff] of UCS4Char TUCS4CharArray;
```

File

System

Description

This is type System.TUCS4CharArray.

158.5.116 System.TVarArrayBoundArray Type

Pascal

```
TVarArrayBoundArray = array [0..0] of TVarArrayBound;
```

C++

```
array [0..0] of TVarArrayBound TVarArrayBoundArray;
```

File

System

Description

This is type System.TVarArrayBoundArray.

158.5.117 System.TVarArrayCoorArray Type

Pascal

```
TVarArrayCoorArray = array [0..0] of Integer;
```

C++

```
array [0..0] of Integer TVarArrayCoorArray;
```

File

System

Description

This is type System.TVarArrayCoorArray.

158.5.118 System.TVarOp Type

TVarOp defines a Variant (see page 22307) operation.

Pascal

```
TVarOp = Integer;
```

C++

```
int TVarOp;
```

File

System

Description

TVarOp is used to define an Variant (see page 22307) operation. There are three operation categories :

Type	Meaning
Two operand processing	A logical operation of left and right operands. Such as BinaryOp.
Two operand comparison	A comparison of left and right operands. Such as CompareOp.
One operand processing	Processing of a single operand. Such as UnaryOp.

TVarOp may hold one of the following values :

Value	Meaning
opAdd (see page 22667)	Addition.
opAnd (see page 22667)	Bitwise and.
opCmpEQ (see page 22668)	Compare for equality.
opCmpGE (see page 22668)	Compare left greater than or equal to right.
opCmpGT (see page 22669)	Compare left greater than right.
opCmpLE (see page 22669)	Compare left less than or equal to right.
opCmpLT (see page 22670)	Compare left less than right.
opCmpNE (see page 22670)	Compare for inequality.
opCompare (see page 22671)	Generalised comparison.
opDivide (see page 22671)	Division of left by right.
opIntDivide (see page 22672)	Integer division of left by right.
opModulus (see page 22672)	Remainder of left divide by right.

opMultiply (see page 22673)	Left multiplied by right.
opNegate (see page 22673)	Negation of a single operand value.
opNot (see page 22674)	Bitwise negation of a single operand.
opOr (see page 22674)	Bitwise or.
opShiftLeft (see page 22675)	Bitwise shift left.
opShiftRight (see page 22675)	Bitwise shift right.
opSubtract (see page 22676)	Right subtracted from left.
opXor (see page 22676)	Bitwise exclusive or.

See Also

BinaryOp

CompareOp (see page 23590)

UnaryOp

158.5.119 System.TVarType Type

Pascal

```
TVarType = Word;
```

C++

```
Word TVarType;
```

File

System

Description

This is type System.TVarType.

158.5.120 System::TWideStringDynArray Type

Pascal

```
TWideStringDynArray = DynamicArray<WideString>;
```

C++

```
typedef DynamicArray<WideString> TWideStringDynArray;
```

File

sysdyn.h

Description

This is type System::TWideStringDynArray.

158.5.121 System::TWordDynArray Type

Pascal

```
TWordDynArray = DynamicArray<Word>;
```

C++

```
typedef DynamicArray<Word> TWordDynArray;
```

File

sysdyn.h

Description

This is type System::TWordDynArray.

158.5.122 System.UCS2Char Type

Pascal

```
UCS2Char = WideChar;
```

C++

```
WideChar UCS2Char;
```

File

System

Description

This is type System.UCS2Char.

158.5.123 System.UCS4Char Type

UCS4Char is used for working with 4–byte Unicode characters.

Pascal

```
UCS4Char = type LongWord;
```

C++

```
type LongWord UCS4Char;
```

File

System

Description

UCS4Char represents a 4–byte Unicode character. UCS4String (see page 22627) is a dynamic array of UCS4Char.

See Also

UCS4String (see page 22627)

UCS4StringToWideString

158.5.124 System.UCS4String Type

UCS4String is used for working with 4–byte Unicode characters.

Pascal

```
UCS4String = array of UCS4Char;
```

C++

```
array of UCS4Char UCS4String;
```

File

System

Description

UCS4Char (see page 22626) represents a 4–byte Unicode character. UCS4String, which is a dynamic array of UCS4Char (see page 22626), represents a string encoded using UCS4 (4–byte Unicode).

See Also

UCS4Char (see page 22626)

UCS4StringToWideString

158.5.125 System.UnitEntryTable Type

Pascal

```
UnitEntryTable = array [0..9999999] of PackageUnitEntry;
```

C++

```
array [0..9999999] of PackageUnitEntry UnitEntryTable;
```

File

System

Description

This is type System.UnitEntryTable.

158.5.126 System.UTF8String Type

Pascal

```
UTF8String = type string;
```

C++

```
type string UTF8String;
```

File

System

Description

This is type System.UTF8String.

158.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
AbstractErrorProc (see page 22630)	AbstractErrorProc defines a callback routine for abstract method call errors.
AllocMemCount (see page 22630)	Represents the total number of allocated memory blocks in a Delphi application.
AllocMemSize (see page 22631)	Represents the total size of allocated memory blocks.
AssertErrorProc (see page 22633)	Points to the assertion error-handler.
CmdLine (see page 22635)	Points to the command-line arguments specified when the application is invoked.
CmdShow (see page 22635)	Specifies the flags for displaying the application's main window.
DebugHook (see page 22636)	DebugHook is set if an application is running under the IDE debugger.
Default8087CW (see page 22636)	Specifies the Default 8087 control word.
DefaultTextLineBreakStyle (see page 22636)	Specifies the characters that are used by default to separate lines in text.
DispCallByIDProc (see page 22637)	This is variable System.DispCallByIDProc.
DLLProc (see page 22637)	Points to a procedure invoked by a DLL entry point.
envp (see page 22638)	This is variable System.envp.
ErrorAddr (see page 22638)	Contains the address of a statement causing a runtime error.
ErrorProc (see page 22639)	Points to the RTL runtime error handler.
ErrOutput (see page 22640)	ErrOutput specifies a write-only Text file associated with the process's error output file.
ExceptClsProc (see page 22640)	ExceptClsProc defines a pointer to a function called during exception handling.
ExceptionClass (see page 22641)	This is variable System.ExceptionClass.
ExceptObjProc (see page 22641)	ExceptObjProc defines a pointer to a function called during exception handling.
ExceptProc (see page 22641)	Points to the lowest-level exception handler.
ExitCode (see page 22642)	Contains the application's exit code.
ExitProc (see page 22643)	Points to a program's exit procedure (provided for backward compatibility).
ExitProcessProc (see page 22644)	Specifies the last procedure to execute before the application shuts down.
FileMode (see page 22644)	Indicates the access mode on typed and untyped files opened by the Reset (see page 22511) procedure.
HeapAllocFlags (see page 22645)	Indicates how the memory manager obtains memory from the operating system.
HPrevInst (see page 22645)	This is variable System.HPrevInst.
InitProc (see page 22646)	InitProc is the last installed initialization procedure.
Input (see page 22647)	Specifies a read-only Text file associated with the processes standard Input file.
IsConsole (see page 22647)	Indicates whether the module was compiled as a console application.
IsLibrary (see page 22648)	Indicates whether the module is a shared library.
IsMultiThread (see page 22648)	Indicates whether the application spawned additional threads using BeginThread (see page 22422) or TThread objects.
JITEnable (see page 22649)	Controls (see page 5711) when the just-in-time debugger is called.
LibModuleList (see page 22650)	Lists module records for enumeration procedures.
MainInstance (see page 22650)	Indicates the instance handle for the main executable.
MainThreadID (see page 22651)	Indicates the instance handle for the current module's main thread of execution.
MaxDispArgs (see page 22651)	This is variable System::MaxDispArgs.
ModuleUnloadList (see page 22651)	This is variable System.ModuleUnloadList.
NeverSleepOnMMThreadContention (see page 22652)	Dictates memory manager behavior during a thread contention.
NoErrMsg (see page 22652)	Controls (see page 5711) whether the application displays an error message when a runtime error occurs.
Output (see page 22653)	Specifies a write-only Text file associated with the process's standard output file.
RaiseExceptionProc (see page 22653)	Points to RaiseException Windows function.
RandSeed (see page 22654)	RandSeed stores the built-in random number generator's seed.
ReportMemoryLeaksOnShutdown (see page 22654)	Report memory leaks on shutdown.

RTLUnwindProc (see page 22655)	This is variable System.RTLUnwindProc.
SafeCallErrorProc (see page 22655)	SafeCallErrorProc defines a procedure that is called when a safecall routine incurs an error.
SystemThreadEndProc (see page 22656)	This is variable System.SystemThreadEndProc.
SystemThreadFuncProc (see page 22656)	This is variable System.SystemThreadFuncProc.
Test8086 (see page 22656)	This is variable System.Test8086.
Test8087 (see page 22657)	This is variable System.Test8087.
TestFDIV (see page 22657)	This is variable System.TestFDIV.
VarAddRefProc (see page 22657)	This is variable System.VarAddRefProc.
VarClearProc (see page 22658)	This is variable System.VarClearProc.
VarCopyProc (see page 22658)	This is variable System.VarCopyProc.
VarToLStrProc (see page 22658)	This is variable System.VarToLStrProc.
VarToWStrProc (see page 22659)	This is variable System.VarToWStrProc.

158.6.1 System.AbstractErrorProc Variable

AbstractErrorProc defines a callback routine for abstract method call errors.

Pascal

```
AbstractErrorProc: procedure;
```

C++

```
procedure AbstractErrorProc;
```

File

System

Description

An abstract method is a virtual or dynamic method that has no implementation in the class where it is declared. Its implementation is deferred to a descendant class. An EAbstractError exception will be thrown if a call is attempted to an abstact method.

You can define an AbstractErrorProc routine that will be called prior to the exception throw.

See Also

EAbstractError (see page 22722)

158.6.2 System.AllocMemCount Variable

Represents the total number of allocated memory blocks in a Delphi application.

Pascal

```
AllocMemCount: Integer deprecated;
```

C++

```
Integer deprecated AllocMemCount;
```

File

System

Description

Note: On Windows, AllocMemCount is the number of currently allocated blocks of memory that were requested by the user. AllocMemCount is incremented each time a block of memory is allocated and is decremented each time a block of memory is freed. Use AllocMemCount to find out how many unfreed blocks of memory remain.

Note: On Linux, the AllocMemCount variable is not maintained by default. Unless it is updated by a third-party memory manager, this variable always has a value of zero.

Warning: Caution:Because AllocMemCount is a global variable declared in the System (see page 22079) unit, it is not always safe to access it directly. In situations where modules are statically linked to the RTL, each module will have a different instance of AllocMemCount. The following table is a summary. Note that applications that use runtime packages are dynamically linked to the RTL, whereas applications that do not are statically linked.

Application type	Accessibility of AllocMemCount
EXE	Applications that do not use packages or Delphi DLLs can safely access this global variable directly because there is only one instance of it.
EXE with packages and no Delphi DLLs	Applications that use runtime packages and do not use any statically linked libraries can safely access AllocMemCount directly. In this situation all modules are dynamically linked to the RTL and there is only one instance of the variable. Unlike standard DLLs, packages allow access to global variables.
EXE with statically linked DLLs	If an application is statically linked to the RTL and also uses libraries that are statically linked to the RTL, then the AllocMemCount variable should never be accessed directly, since the EXE and each library will have its own instance of it. Instead, use the GetAllocMemCount function, implemented in BorlandMM, which returns the instance of the AllocMemCount global variable in the BorlandMM DLL. This DLL manages the memory allocation for all modules that include the ShareMem unit as the first unit in the application or library's uses clause. It is, therefore, this instance of the variable that contains the correct value. A function is used to access AllocMemCount in this situation because global variables in a DLL are private to the DLL.
EXE with packages and statically linked DLLs	It is not advisable to create applications that mix runtime packages with statically linked libraries. In this circumstance, use caution when allocating, freeing, and accessing memory. Be aware that each version of the RTL contains an instance of AllocMemCount that references memory that is allocated and freed only by that particular module.

See Also

AllocMemSize (see page 22631)

AllocMem (see page 22413)

158.6.3 System.AllocMemSize Variable

Represents the total size of allocated memory blocks.

Pascal

```
AllocMemSize: Integer deprecated;
```

C++

```
Integer deprecated AllocMemSize;
```

File

System

Description

Note: On Windows, AllocMemSize is the total size, in bytes, of all currently allocated blocks of memory in use by an application. Use AllocMemSize to find out how many bytes of memory an application is currently using.

Note: On Linux, the AllocMemSize variable is not maintained by default. Unless it is updated by a third-party memory manager, this variable always has a value of zero.

Warning: Caution:Because AllocMemSize is a global variable declared in the System (see page 22079) unit, it is not always safe to access it directly. In situations where modules are statically linked to the RTL, each module will have a different instance of AllocMemSize. The following table is a summary. Note that applications that use runtime packages are dynamically linked to the RTL, whereas applications that do not are statically linked.

Application type	Accessibility of AllocMemSize
EXE	Applications that do not use packages or Delphi DLLs can safely access this global variable directly because there is only one instance of it.
EXE with packages and no Delphi DLLs	Applications that use runtime packages and do not use any statically linked DLLs can safely access AllocMemSize directly. In this situation all modules are dynamically linked to the RTL and there is only one instance of the variable. Unlike standard DLLs, packages allow access to global variables.
EXE with statically linked DLLs	If an application is statically linked to the RTL and if that application is also using DLLs that are statically linked to the RTL then the AllocMemSize variable should never be accessed directly, since the EXE and each DLL will have its own instance of it. Instead use the GetAllocMemSize function, implemented in BorlandMM, which returns the instance of the AllocMemSize global variable in the BorlandMM DLL. This DLL manages the memory allocation for all modules that include the ShareMem unit as the first unit in the application or library's uses clause. It is, therefore, this instance of the variable that contains the correct value. A function is used to access AllocMemSize in this situation because global variables in a DLL are private to the DLL.
EXE with packages and statically linked DLLS	It is not advisable to create applications that mix runtime packages with statically linked DLLs. In this circumstance use caution when allocating, freeing, and accessing memory. Be aware that each version of the RTL contains an instance of AllocMemSize that references memory that is allocated and freed only by that particular module.

See Also

AllocMem (see page 22413)

AllocMemCount (see page 22630)

158.6.4 System.AssertErrorProc Variable

Points to the assertion error-handler.

Pascal

```
AssertErrorProc: TAssertErrorProc;
```

C++

```
TAssertErrorProc AssertErrorProc;
```

File

System

Description

AssertErrorProc points to a routine that produces runtime error 227 in Delphi code. The SysUtils (see page 22709) unit replaces it to cause an EAssertionFailed exception. If you do not use assertions, you can replace this with a method that that does something else, such as call OutputDebugString to add messages to the event log (see the example).

The value of AssertErrorProc must be a procedure with the following signature

procedure AssertErrorHandler(const Message, Filename: string; LineNumber: Integer; ErrorAddr (see page 22638): Pointer);

Delphi Examples:

```
{
AssertErrorProc example
The following code is a self-contained unit that you can add to a project
that logs failed assertions to the event log rather than raising an
exception. If /Debug is passed to the application as a command-line
parameter, all failed Assert calls cause a call to OutputDebugString,
which sends a messages to the Event Log window if run from Delphi, or
does nothing if run outside the IDE (OutputDebugString is a no-op if not
run from within a debugger). A nice side benefit is that all the line
numbers and unit names can be stripped from the EXE by turning off
assertions and rebuilding the project.
Note: AssertErrorHandler should do as little as possible.  This routine
does no cleanup, so no variables, no compiler generated variables.
Procedure calls are OK.
Note:   This example was originally provided by Brian Long.
}
type
  TForm1 = class(TForm)
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TStorage = class(TObject)
    FData: string;
    property Data: string read FData write FData;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  var
  Form1: TForm1;
```

```
procedure AssertErrorHandler(const Message, Filename: string;  LineNumber: Integer; ErrorAddr:
Pointer);

implementation

{$R *.dfm}

{$ASSERTIONS ON}

type
  EMyAssert = class(Exception);


procedure DoMessage(Message, Filename: String; LineNumber: Integer; ErrorAddr: Pointer);
var
  S: String;
begin
  S := Format('%s (%s, line %d, address $%x)',
    [Message, Filename, LineNumber, Pred(Integer(ErrorAddr))]);
  OutputDebugString(PChar(S));
end;

procedure AssertErrorHandler(const Message, Filename: string;  LineNumber: Integer; ErrorAddr:
Pointer);
{ No local variables.  Not compiler generated temporary variables. }
{ Using the call stack here will cause Access Violation errors. }
begin
  DoMessage(Message, Filename, LineNumber, ErrorAddr);
  raise EMyAssert.Create('Boom!');
end;

procedure AssertErrorNoHandler(const Message, Filename: string; LineNumber: Integer;
ErrorAddr: Pointer);
begin
end;

procedure ModifyStorage(AStorage: TStorage; const s: string);
begin
  Assert(AStorage <> nil, 'Value');
  AStorage.Data := s;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  ModifyStorage(nil, 'Oops');
end;

initialization
  if FindCmdLineSwitch('Debug', ['/', '-'], True) then
  begin
    MessageDlg('Command line switch found!', mtInformation, [mbOK], 0);
//    OldProc := System.AssertErrorProc;
    System.AssertErrorProc := @AssertErrorHandler
  end
  else
    System.AssertErrorProc := @AssertErrorNoHandler;

  ModifyStorage(nil, 'Ooops');
```

See Also

EAssertionFailed

Assert (see page 22417)

ErrorProc (see page 22639)

158.6.5 System.CmdLine Variable

Points to the command-line arguments specified when the application is invoked.

Pascal

```
CmdLine: PChar platform;
```

C++

```
PChar platform CmdLine;
```

File

System

Description

Note: On Windows, the CmdLine variable points to a null-terminated string that contains the command-line arguments specified when the application was started. Use ParamStr (see page 22496) to access individual arguments. In a library (DLL), CmdLine is nil (Delphi) or NULL (C++).

Note: On Linux, CmdLine is always nil (Delphi) or NULL (C++).

See Also

ParamStr (see page 22496)

158.6.6 System.CmdShow Variable

Specifies the flags for displaying the application's main window.

Pascal

```
CmdShow: Integer platform;
```

C++

```
Integer platform CmdShow;
```

File

System

Description

Note: On Windows, CmdShow specifies the parameter that should be passed to the Windows API ShowWindow when displaying the main window of the application. In a library (DLL), CmdShow is always 0.

Note: On Linux, CmdShow is always 0. It exists only for compatibility with Windows code in cross-platform applications.

158.6.7 System.DebugHook Variable

DebugHook is set if an application is running under the IDE debugger.

Pascal

```
DebugHook: Byte platform = 0;
```

C++

```
Byte platform DebugHook = 0;
```

File

System

Description

DebugHook is 0 when an application operates outside the IDE debugger. DebugHook is 1 when an application operates inside the IDE debugger.

158.6.8 System.Default8087CW Variable

Specifies the Default 8087 control word.

Pascal

```
Default8087CW: Word = $1332;
```

C++

```
Word Default8087CW = 0x1332;
```

File

System

Description

Default8087CW is the value to which the Floating Point Unit (FPU) control register is set. The FPU control word includes bits that control the FPU's precision, rounding mode, and whether exceptions generate signals if they occur. See Intel's processor documentation for details.

The Set8087CW (see page 22520) procedure allows the programmer to have direct access to the Default8087CW. Setting the floating-point precision and rounding mechanism can be useful if you are reusing old code that is sensitive to the floating-point precision standard used. However, be aware that using this procedure to change the value of the 8087CW will change the behavior of the program's FP calculations. It is the programmer's responsibility to reset it.

See Also

Set8087CW (see page 22520)

158.6.9 System.DefaultTextLineBreakStyle Variable

Specifies the characters that are used by default to separate lines in text.

Pascal

```
DefaultTextLineBreakStyle: TTextLineBreakStyle = tlbsCRLF;
```

C++

```
TTextLineBreakStyle DefaultTextLineBreakStyle = tlbsCRLF;
```

File

System

Description

DefaultTextLineBreakStyle controls how the code editor represents the separation between lines of text in new files. For existing files, it maintains the line break style that the file currently uses.

See Also

SetLineBreakStyle (see page 22522)

158.6.10 System.DispCallByIDProc Variable

Pascal

```
DispCallByIDProc: Pointer;
```

C++

```
void * DispCallByIDProc;
```

File

System

Description

This is variable System.DispCallByIDProc.

158.6.11 System.DLLProc Variable

Points to a procedure invoked by a DLL entry point.

Pascal

```
DLLProc: Pointer;
```

C++

```
void * DLLProc;
```

File

System

Description

DLLProc is used to specify a procedure that is invoked every time a DLL's entry point is called. A procedure assigned to DLLProc must take one parameter of type Integer or int. For example,

- procedure LibraryProc(Reason: Integer); { Delphi }
- void LibraryProc(int Reason); // C++

- When the procedure is invoked, this single parameter contains a value between 0 and 3 as defined by the following group of constants (from the Windows unit):
- const
- DLL_PROCESS_DETACH = 0;
- DLL_PROCESS_ATTACH = 1;
- DLL_THREAD_ATTACH = 2;
- DLL_THREAD_DETACH = 3;
- For further details on the meaning of these constants, refer to the description of the DllEntryPoint function in the Win32 API online help.

Note: DLL_PROCESS_ATTACH is passed to the procedure only if the DLL's initialization code calls the procedure and specifies DLL_PROCESS_ATTACH as a parameter.

Note: When the Reason parameter is DLL_PROCESS_ATTACH, setting ExitCode (see page 22642) to a nonzero value causes the entry point to return false.

158.6.12 System.envp Variable

Pascal

```
envp: PPChar platform;
```

C++

```
PPChar platform envp;
```

File

System

Description

This is variable System.envp.

158.6.13 System.ErrorAddr Variable

Contains the address of a statement causing a runtime error.

Pascal

```
ErrorAddr: Pointer = nil;
```

C++

```
void * ErrorAddr = nil;
```

File

System

Description

If a program terminates normally or stops due to a call to Halt (see page 22475), ErrorAddr is nil (Delphi) or NULL (C++). If a program ends because of a runtime error, ErrorAddr contains the address of the statement in error.

See Also

ErrorProc (see page 22639)

ExceptProc (see page 22641)

AcquireExceptionObject (see page 22410)

ExceptAddr (see page 22452)

ExceptionErrorMessage

ExceptObject (see page 22452)

GetLastError (see page 22469)

158.6.14 System.ErrorProc Variable

Points to the RTL runtime error handler.

Pascal

```
ErrorProc: procedure (ErrorCode: Byte; ErrorAddr: Pointer);
```

C++

```
procedure (ErrorCode: Byte; ErrorAddr: Pointer) ErrorProc;
```

File

System

Description

In Delphi, ErrorProc is a procedure variable pointing to the runtime error handler. The standard ErrorProc procedure reports the runtime error and terminates the program.

The ErrorProc variable is retained primarily for compatibility with older code. Newer applications should not specify an alternate runtime error procedure using ErrorProc as SysUtils (see page 22709) uses this variable to convert runtime errors into exceptions.

If you use SysUtils (see page 22709) in your program, it will force ErrorProc to its own routine and convert the runtime error into an exception. When a runtime library error occurs, the ErrorProc is called. The ErrorCode parameter contains the runtime error number and the ErrorAddr (see page 22638) parameter contains the machine code address that caused the runtime error.

See Also

ErrorAddr (see page 22638)

ExceptAddr (see page 22452)

ExceptionErrorMessage

ExceptObject (see page 22452)

GetLastError (see page 22469)

ExceptProc (see page 22641)

GetLastError (see page 22469)

AcquireExceptionObject (see page 22410)

158.6.15 System.ErrOutput Variable

ErrOutput specifies a write-only Text file associated with the process's error output file.

Pascal

```
ErrOutput: Text;
```

C++

```
Text ErrOutput;
```

File

System

Description

In Delphi, the ErrOutput variable is a write-only Text file associated with the process's error output file.

Note: On Windows, most processes don't have an error output file, and writing to ErrOutput raises an error. Delphi programs have a standard output file if they are linked as console applications.

Note: On Linux, all processes have an error output file. There are various ways to view the error output. One simple method is to run the program from a shell window.

See Also

Input (see page 22647)

Output (see page 22653)

TextFile

158.6.16 System.ExceptClsProc Variable

ExceptClsProc defines a pointer to a function called during exception handling.

Pascal

```
ExceptClsProc: Pointer;
```

C++

```
void * ExceptClsProc;
```

File

System

Description

By default, ExceptClsProc is defined internally to point to a function that handles exceptions on your behalf. It returns the exception class. If you override ExceptClsProc to point to your function, that function will be called when an exception occurs in a try-except block. It will be passed an updateable TExceptionRecord value that describes the exception.

Internally, the class returned is used to determine which exception handler to use before ExceptObjProc (see page 22641) is called to create the exception instance.

See Also

ExceptObjProc (see page 22641)

TExceptionRecord

158.6.17 System.ExceptionClass Variable

Pascal

```
ExceptionClass: TClass;
```

C++

```
TClass ExceptionClass;
```

File

System

Description

This is variable System.ExceptionClass.

158.6.18 System.ExceptObjProc Variable

ExceptObjProc defines a pointer to a function called during exception handling.

Pascal

```
ExceptObjProc: Pointer;
```

C++

```
void * ExceptObjProc;
```

File

System

Description

ExceptClsProc (see page 22640) defines the function that returns a class that is used to determine which exception handler to use before an exception object is created.

You can override the default exception object creation by defining your own function that returns a TObject (see page 22285). You point ExceptObjProc to that function. It is passed the an updateable version of the TExceptionRecord.

See Also

ExceptClsProc (see page 22640)

TExceptionRecord

158.6.19 System.ExceptProc Variable

Points to the lowest-level exception handler.

Pascal

```
ExceptProc: Pointer;
```

C++

```
void * ExceptProc;
```

File

System

Description

In Delphi code, ExceptProc handles unhandled exceptions (exceptions that have not been caught in the except portion of a try..except statement.) If your application has a TApplication object, it automatically catches exceptions and generates an OnException event (to which you can respond using either the application's or a TApplicationEvents object's OnException event handler). Thus, because the TApplication object catches the exceptions, they never reach the ExceptProc handler. However, even if you have a TApplication object, it does not catch exceptions generated during initialization and finalization. These are handled by ExceptProc.

You can hook into ExceptProc to change how unhandled exceptions are reported. To do so, save the contexts of ExceptProc before changing it to the address of your own exception handler. The first statement of your new exception handler must then reinstall the saved value of ExceptProc. In this way, the ExceptProc procedure becomes part of a chain of exception procedures, each of which is executed in reverse order of installation.

The procedure assigned to ExceptProc must have a signature such as the following:

procedure ExceptHandler(ExceptObject (see page 22452): TObject (see page 22285); ExceptAddr (see page 22452): Pointer);

Note: :On Linux, ExceptProc may be called when most of the runtime has already shut down. This happens if an application terminates with an unhandled exception.

See Also

ErrorAddr (see page 22638)

ErrorProc (see page 22639)

OnException

TApplication

ExceptObject (see page 22452)

158.6.20 System.ExitCode Variable

Contains the application's exit code.

Pascal

```
ExitCode: Integer = 0;
```

C++

```
int ExitCode = 0;
```

File

System

Description

In Delphi code, the ExitCode variable contains the application's exit code. An exit procedure can learn the cause of termination by examining ExitCode. If the program terminates normally, ExitCode is zero. If the program terminates due to a call to Halt (see page 22475), ExitCode contains the value passed to Halt (see page 22475). If the program terminates due to a runtime error, ExitCode contains the error code.

Note: On Windows, setting the ExitCode variable to a value other than 0 in the initialization section of a unit will make loading of a DLL fail (DllMain will return false). You can also set ExitCode in code called from DllProc or DllProcEx to signal failure of loading.

Note: On Linux, although the ExitCode variable is accessible from shared object code, its value does not affect the loader of the shared object. Thus, using ExitCode in a shared object has no effect.

See Also

AddExitProc

AddTerminateProc

CallTerminateProcs

ExitProc (see page 22643)

158.6.21 System.ExitProc Variable

Points to a program's exit procedure (provided for backward compatibility).

Pascal

```
ExitProc: Pointer;
```

C++

```
void * ExitProc;
```

File

System

Description

In Delphi code, ExitProc points to an exit procedure in an application. The exit procedure always gets called as part of a program's termination. ExitProc is only used when generating executable files.

An exit procedure takes no parameters and must be compiled with a far procedure directive to force it to use the far call model.

Do not use ExitProc in a dynamically loaded package. This procedure is called when the process halts, not when a package or DLL is unloaded. If you set ExitProc from a package or DLL, it may interfere with the ExitProc of the host executable. Because ExitProc is not compatible with packages, it is recommended that you add code to the finalization section of a unit instead. Inside a DLL, you can use the DLLProc (see page 22637) variable instead.

See Also

AddExitProc

ExitCode (see page 22642)

AddTerminateProc

CallTerminateProcs

DLLProc (see page 22637)

158.6.22 System.ExitProcessProc Variable

Specifies the last procedure to execute before the application shuts down.

Pascal

```
ExitProcessProc: procedure;
```

C++

```
procedure ExitProcessProc;
```

File

System

Description

Assign a value to ExitProcessProc to specify a procedure that you want to execute when the application shuts down. The value of ExitProcessProc is a procedure with no parameters.

See Also

InitProc (see page 22646)

158.6.23 System.FileMode Variable

Indicates the access mode on typed and untyped files opened by the Reset (see page 22511) procedure.

Pascal

```
FileMode: Byte = 2;
```

C++

```
Byte FileMode = 2;
```

File

System

Description

In Delphi code, FileMode determines the access mode to use when typed and untyped files (not text files) are opened using the Reset (see page 22511) procedure.

The default FileMode is 2 (Read (see page 22504)/Write (see page 22548) access). Assigning another value to FileMode causes all subsequent Resets to use that mode. The SysUtils (see page 22709) unit declares a number of file open mode constants that can be assigned to the FileMode variable.

Note: Be sure to reset FileMode before calling Reset (see page 22511) with a read-only file. Even if you are only reading from the file, the default FileMode value (Read (see page 22504)/Write (see page 22548) access) will cause an attempt to open a read-only file to fail.

Warning: FileMode is declared as a variable, not as a threadvar. Therefore this variable is not thread-safe.

See Also

Reset (see page 22511)

Rewrite (see page 22513)

TFileStream

158.6.24 System.HeapAllocFlags Variable

Indicates how the memory manager obtains memory from the operating system.

Pascal

```
HeapAllocFlags: Word platform = 2;
```

C++

```
Word platform HeapAllocFlags = 2;
```

File

System

Description

On Windows, use HeapAllocFlags to customize the flags that the system memory manager uses when requesting blocks of memory from the operating system. By default, the value is GMEM_MOVEABLE.

HeapAllocFlags can be a combination of the following values:

GMEM_FIXED	Allocates fixed memory. The operating system can't move blocks, so there is no need to lock memory when obtaining a pointer. (can't be combined with GMEM_MOVEABLE).
GMEM_MOVEABLE	Allocates movable memory. In Win32, memory blocks are never moved in physical memory, but they can be moved within the default heap.
GMEM_ZEROINIT	Initializes memory contents to zero.
GMEM_MODIFY	Used to modify the attributes of an already-allocated block of memory.
GMEM_DDESHARE GMEM_SHARE	This flag is primarily for compatibility with 16-bit Window, but. can be used to enhance the performance of DDE operations. It should only be specified if the memory is to be used for DDE.
GPTR	Combines the GMEM_FIXED and GMEM_ZEROINIT flags.
GHND	Combines the GMEM_MOVEABLE and GMEM_ZEROINIT flags.

See Also

SetMemoryManager (see page 22523)

158.6.25 System.HPrevInst Variable

Pascal

```
HPrevInst: LongWord deprecated;
```

C++

```
LongWord deprecated HPrevInst;
```

File

System

Description

This is variable System.HPrevInst.

158.6.26 System.InitProc Variable

InitProc is the last installed initialization procedure.

Pascal

```
InitProc: Pointer;
```

C++

```
void * InitProc;
```

File

System

Description

Assign a value to InitProc to specify a procedure that you want to execute when the application starts up. The value of InitProc is a procedure with no parameters. For example:

In Delphi, the assignment can be made in the initialization section of a unit:

```
procedure MyInitProcedure;
...
initialization
  InitProc = @MyInitProcedure;
begin
end;
```

In C++, the assignment is most easily made in the project source file. This example is for Windows. On Linux applications, the code is similar.

```
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
  try
  {
    // Note that call must precede call to Application->Initialize();
    System.InitProc = MyInitProcedure;
    Application->Initialize();
    Application->CreateForm(__classid(TForm1), &Form1);
    Application->Run();
  }
  catch (Exception &exception)
  {
    Application->ShowException(&exception);
  }
  catch (...)
  {
    try
    {
      throw Exception("");
    }
```

```
    catch (Exception &exception)
    {
      Application->ShowException(&exception);
    }
  }
  return 0;
}
```

Only one initialization procedure can be assigned to the InitProc variable. If your application defines more than one initialization procedure, only the last assignment to InitProc executes. To allow other initialization procedures to execute, you must "chain" the procedures together, calling the previously assigned value of InitProc from the new value.

See Also

TApplication

ExitProcessProc (see page 22644)

158.6.27 System.Input Variable

Specifies a read-only Text file associated with the processes standard Input file.

Pascal

```
Input: Text;
```

C++

```
Text Input;
```

File

System

Description

In Delphi, the Input variable is a read-only Text file associated with the process's standard Input file.

Note: On Windows, most processes don't have a standard Input file, and reading from Input raises an error. Delphi programs have a standard Input file if they are linked as console applications.

Note: On Linux, all processes have a standard Input file. There are various ways to use the standard Input. One simple method is to run the program from a shell window.

See Also

Output (see page 22653)

TextFile

158.6.28 System.IsConsole Variable

Indicates whether the module was compiled as a console application.

Pascal

```
IsConsole: Boolean;
```

C++

```
Boolean IsConsole;
```

File

System

Description

The IsConsole variable is true if the module was linked as a console application. This applies both to an executable and a shared library.

Note: On Linux, IsConsole is always true.

See Also

IsLibrary (see page 22648)

IsMultiThread (see page 22648)

Input (see page 22647)

Output (see page 22653)

ErrOutput (see page 22640)

158.6.29 System.IsLibrary Variable

Indicates whether the module is a shared library.

Pascal

```
IsLibrary: Boolean;
```

C++

```
Boolean IsLibrary;
```

File

System

Description

The IsLibrary variable is true if the module is a dynamic link library (Windows) or shared object (Linux).

See Also

IsConsole (see page 22647)

IsMultiThread (see page 22648)

158.6.30 System.IsMultiThread Variable

Indicates whether the application spawned additional threads using BeginThread (see page 22422) or TThread objects.

Pascal

```
IsMultiThread: Boolean;
```

C++

```
Boolean IsMultiThread;
```

File

System

Description

IsMultiThread is set to true to indicate that the memory manager should support multiple threads. IsMultiThread is set to true by BeginThread (see page 22422) and class factories.

See Also

BeginThread (see page 22422)

IsConsole (see page 22647)

IsLibrary (see page 22648)

CoInitFlags

TComObjectFactory

158.6.31 System.JITEnable Variable

Controls (see page 5711) when the just-in-time debugger is called.

Pascal

```
JITEnable: Byte platform = 0;
```

C++

```
Byte platform JITEnable = 0;
```

File

System

Description

Use JITEnable to indicate what types of exceptions trigger the just-in-time debugger. JITEnable only has an effect on applications that run under Windows.

When JITEnable is 0 (the default), all exceptions are handled using the application's exception handling mechanism. The just-in-time debugger is only called if the application provides no mechanism to catch exceptions. Note that any application that includes the Forms (see page 13657) unit provides exception handling (via the Application object) for all but the initialization and finalization sections in Delphi units and does not trigger the just-in-time debugger.

When JITEnable is 1, any non-native exceptions (Exceptions raised by code that was compiled using other products) trigger the just-in-time debugger. Native exceptions are handled by the application's exception handling mechanism.

When JITEnable is greater than 1, all exceptions trigger the just-in-time debugger.

Note: When you install the IDE, it is registered as the just-in-time debugger. If you install other tools with debuggers, they may register themselves, overriding the registration of the IDE.

158.6.32 System.LibModuleList Variable

Lists module records for enumeration procedures.

Pascal

```
LibModuleList: PLibModule = nil;
```

C++

```
PLibModule LibModuleList = nil;
```

File

System

Description

LibModuleList implements a linked list of records for executables and packages in an application. LibModuleList is used by the EnumModules (see page 22447) and the EnumResourceModules (see page 22448) procedures.

See Also

EnumModules (see page 22447)

EnumResourceModules (see page 22448)

158.6.33 System.MainInstance Variable

Indicates the instance handle for the main executable.

Pascal

```
MainInstance: LongWord;
```

C++

```
LongWord MainInstance;
```

File

System

Description

Use MainInstance to obtain the instance handle for the main executable of an application. This is useful in applications that use runtime libraries or packages, when you need the handle for the executable rather than for the library.

Note:

See Also

HInstance

FindHInstance (see page 22461)

158.6.34 System.MainThreadID Variable

Indicates the instance handle for the current module's main thread of execution.

Pascal

```
MainThreadID: LongWord;
```

C++

```
LongWord MainThreadID;
```

File

System

Description

Use MainThreadID to obtain the thread ID for the main execution thread (the current thread when the module was initialized).

During debugging, this ID appears in the thread status box for the main thread.

Note: On Windows, the value of MainThreadID is compatible with the thread ID required by some Win32 API calls.

See Also

TThread

158.6.35 System::MaxDispArgs Variable

Pascal

```
MaxDispArgs: const short int = $20;
```

C++

```
const short int MaxDispArgs = 0x20;
```

File

sysvari.h

Description

This is variable System::MaxDispArgs.

158.6.36 System.ModuleUnloadList Variable

Pascal

```
ModuleUnloadList: PModuleUnloadRec = nil;
```

C++

```
PModuleUnloadRec ModuleUnloadList = nil;
```

File

System

Description

This is variable System.ModuleUnloadList.

158.6.37 System.NeverSleepOnMMThreadContention Variable

Dictates memory manager behavior during a thread contention.

Pascal

```
NeverSleepOnMMThreadContention: Boolean;
```

C++

```
Boolean NeverSleepOnMMThreadContention;
```

File

System

Description

The Memory Manager is a shared resource, and when many threads in the application attempt to perform a memory operation at the same time, one or more threads may have to wait for a pending operation in another thread to complete before it can continue. This situation is called a thread contention. When a thread contention occurs inside the Memory Manager the default behavior is to first relinquish the remaining time in the thread's time slice. If the resource is still not available when the thread enters its next time slice, the Memory Manager calls the OS procedure Sleep before trying again.

This behavior works well on machines with single or dual core CPUs, and also when the ratio of the number of running threads to number of CPU cores is relatively high (greater than 2:1). In other situations, better performance can be obtained by entering a busy waiting loop until the resource becomes available. If NeverSleepOnMMThreadContention is **True**, the Memory Manager will use this alternate strategy when a thread contention occurs. The default value for NeverSleepOnMMThreadContention is **False**.

See Also

Memory Management

Configuring the Memory Manager

158.6.38 System.NoErrMsg Variable

Controls (see page 5711) whether the application displays an error message when a runtime error occurs.

Pascal

```
NoErrMsg: Boolean platform = False;
```

C++

```
Boolean platform NoErrMsg = False;
```

File

System

Description

Note: On Windows, set NoErrMsg to control whether a message appears to indicate that a runtime error has occurred. When NoErrMsg is false (the default), runtime errors cause the application to display a message box indicating the type of error that occurred. When NoErrMsg is true, these messages are suppressed.

On Linux, NoErrMsg only no effect.

Note: The SysUtils (see page 22709) unit turns most runtime errors into exceptions. If your application includes SysUtils (see page 22709), the runtime error message box is unlikely to appear, even if NoErrMsg is false.

158.6.39 System.Output Variable

Specifies a write-only Text file associated with the process's standard output file.

Pascal

```
Output: Text;
```

C++

```
Text Output;
```

File

System

Description

In Delphi, the Output variable is a write-only Text file associated with the process's standard output file.

Note: On Windows, most processes don't have a standard output file, and writing to Output raises an error. Delphi programs have a standard output file if they are linked as console applications.

Note: On Linux, all processes have a standard output file. There are various ways to view the standard error output. One simple method is to run the program from a shell window.

See Also

Input (see page 22647)

TextFile

158.6.40 System.RaiseExceptionProc Variable

Points to RaiseException Windows function.

Pascal

```
RaiseExceptionProc: Pointer;
```

C++

```
void * RaiseExceptionProc;
```

File

System

Description

RaiseExceptionProc points to the RaiseException Windows API function. This can be used to invoke the function from assembly language.

See Also

RaiseException Function (MSDN)

158.6.41 System.RandSeed Variable

RandSeed stores the built-in random number generator's seed.

Pascal

```
RandSeed: Longint = 0;
```

C++

```
Longint RandSeed = 0;
```

File

System

Description

By assigning a specific value to RandSeed, you can use the Random (see page 22502) function to repetitively generate a specific sequence of random numbers. This is useful for applications that deal with data statistics and simulations. For applications that deal with encryption see the note under the Random (see page 22502) function.

See Also

Random (see page 22502)

Randomize (see page 22503)

158.6.42 System.ReportMemoryLeaksOnShutdown Variable

Report memory leaks on shutdown.

Pascal

```
ReportMemoryLeaksOnShutdown: Boolean;
```

C++

```
Boolean ReportMemoryLeaksOnShutdown;
```

File

System

Description

Set ReportMemoryLeaksOnShutdown to report memory leaks on shutdown.

The memory manager can report memory that was allocated but not freed by the time the memory manager shuts down. Such memory blocks are called memory leaks and are often the result of programming errors. When this global variable is set to **True**, the Memory Manager will scan the memory pool when it shuts down and report all unregistered memory leaks in a message

dialog. The default value for ReportMemoryLeaksOnShutdown is **False**.

Expected memory leaks can be registered so the Memory Manager will ignore them during its checks for memory leaks. Use the procedures RegisterExpectedMemoryLeak (see page 22508) and UnregisterExpectedMemoryLeak (see page 22539) to control the memory areas that the Memory Manager checks for leaks.

See Also

Memory Management

Configuring the Memory Manager

Registering Memory Leaks

RegisterExpectedMemoryLeak

UnregisterExpectedMemoryLeak

158.6.43 System.RTLUnwindProc Variable

Pascal

```
RTLUnwindProc: Pointer;
```

C++

```
void * RTLUnwindProc;
```

File

System

Description

This is variable System.RTLUnwindProc.

158.6.44 System.SafeCallErrorProc Variable

SafeCallErrorProc defines a procedure that is called when a safecall routine incurs an error.

Pascal

```
SafeCallErrorProc: TSafeCallErrorProc;
```

C++

```
TSafeCallErrorProc SafeCallErrorProc;
```

File

System

Description

A function or procedure defined with the safecall directive is invoked using a safe calling convention. When such a routine incurs an error, the SafeCallException is invoked. However, you can override this call by defining your own handling routine, and setting SafeCallErrorProc to point to it.

SafeCallErrorProc should point to a procedure that takes two parameters :

ErrorCode is an integer holding the error code.

ErrorAddr (see page 22638) is the code address at the time of error.

If SafeCallErrorProc returns normally, or is set to nil, then the ESafeCallException is raised.

See Also

ESafeCallException

SafeCallException

158.6.45 System.SystemThreadEndProc Variable

Pascal

```
SystemThreadEndProc: TSystemThreadEndProc = nil;
```

C++

```
TSystemThreadEndProc SystemThreadEndProc = nil;
```

File

System

Description

This is variable System.SystemThreadEndProc.

158.6.46 System.SystemThreadFuncProc Variable

Pascal

```
SystemThreadFuncProc: TSystemThreadFuncProc = nil;
```

C++

```
TSystemThreadFuncProc SystemThreadFuncProc = nil;
```

File

System

Description

This is variable System.SystemThreadFuncProc.

158.6.47 System.Test8086 Variable

Pascal

```
Test8086: Byte;
```

C++

```
Byte Test8086;
```

File

System

Description

This is variable System.Test8086.

158.6.48 System.Test8087 Variable

Pascal

```
Test8087: Byte = 3;
```

C++

```
Byte Test8087 = 3;
```

File

System

Description

This is variable System.Test8087.

158.6.49 System.TestFDIV Variable

Pascal

```
TestFDIV: Shortint;
```

C++

```
Shortint TestFDIV;
```

File

System

Description

This is variable System.TestFDIV.

158.6.50 System.VarAddRefProc Variable

Pascal

```
VarAddRefProc: procedure (var v: TVarData) = nil;
```

C++

```
procedure (var v: TVarData) VarAddRefProc = nil;
```

File

System

Description

This is variable System.VarAddRefProc.

158.6.51 System.VarClearProc Variable

Pascal

```
VarClearProc: procedure (var v: TVarData) = nil;
```

C++

```
procedure (var v: TVarData) VarClearProc = nil;
```

File

System

Description

This is variable System.VarClearProc.

158.6.52 System.VarCopyProc Variable

Pascal

```
VarCopyProc: procedure (var Dest: TVarData; const Source: TVarData) = nil;
```

C++

```
procedure (var Dest: TVarData; const Source: TVarData) VarCopyProc = nil;
```

File

System

Description

This is variable System.VarCopyProc.

158.6.53 System.VarToLStrProc Variable

Pascal

```
VarToLStrProc: procedure (var Dest: AnsiString; const Source: TVarData) = nil;
```

C++

```
procedure (var Dest: AnsiString; const Source: TVarData) VarToLStrProc = nil;
```

File

System

Description

This is variable System.VarToLStrProc.

158.6.54 System.VarToWStrProc Variable

Pascal

```
VarToWStrProc: procedure (var Dest: WideString; const Source: TVarData) = nil;
```

C++

```
procedure (var Dest: WideString; const Source: TVarData) VarToWStrProc = nil;
```

File

System

Description

This is variable System.VarToWStrProc.

158.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CompilerVersion (see page 22662)	Designates the version number of the Delphi compiler.
CPUi386 (see page 22662)	This is constant System.CPUi386.
CPUi486 (see page 22663)	This is constant System.CPUi486.
CPUPentium (see page 22663)	This is constant System.CPUPentium.
E_NOINTERFACE (see page 22663)	This is constant System.E_NOINTERFACE.
E_NOTIMPL (see page 22664)	This is constant System.E_NOTIMPL.
E_UNEXPECTED (see page 22664)	This is constant System.E_UNEXPECTED.
fmClosed (see page 22664)	This is constant System.fmClosed.
fmInOut (see page 22665)	This is constant System.fmInOut.
fmInput (see page 22665)	This is constant System.fmInput.
fmOutput (see page 22665)	This is constant System.fmOutput.
MaxInt (see page 22666)	The maximum value of the Integer data type.
MaxLongint (see page 22666)	The maximum value of the Longint data type.
NumSmallBlockTypes (see page 22666)	NumSmallBlockTypes is the number of small memory block types employed by the default memory manager.
opAdd (see page 22667)	opAdd defines an addition operation.
opAnd (see page 22667)	opAnd defines a logical and operation.
opCmpEQ (see page 22668)	opCmpEQ defines an equality comparison operator.
opCmpGE (see page 22668)	opCmpGE defines a greater than or equal comparison operator.
opCmpGT (see page 22669)	opCmpGT defines a greater than comparison operator.
opCmpLE (see page 22669)	opCmpLE defines a less than or equal comparison operator.
opCmpLT (see page 22670)	opCmpLT defines a less than comparison operator.
opCmpNE (see page 22670)	opCmpNE defines an inequality comparison operator.
opCompare (see page 22671)	opCompare defines a comparison operation.
opDivide (see page 22671)	opDivide defines a divide operation.
opIntDivide (see page 22672)	opIntDivide defines an integer divide operation.
opModulus (see page 22672)	opModulus defines a modulus operation.
opMultiply (see page 22673)	opMultiply defines a multiply operation.
opNegate (see page 22673)	opNegate defines a negate operation.
opNot (see page 22674)	opNot defines a binary negation operation.
opOr (see page 22674)	opOr defines a logical or operation.
opShiftLeft (see page 22675)	opShiftLeft defines a bitwise shift left operation.
opShiftRight (see page 22675)	opShiftRight defines a bitwise shift right operation.
opSubtract (see page 22676)	opSubtract defines a subtraction operation.
opXor (see page 22676)	opXor defines a logical exclusive or operation.
RTLVersion (see page 22677)	Designates the version number of the Delphi runtime library.
S_FALSE (see page 22677)	This is constant System.S_FALSE.
S_OK (see page 22677)	This is constant System.S_OK.
sLineBreak (see page 22678)	Represents the character(s) used to generate a line feed.
tfCRLF (see page 22678)	This is constant System.tfCRLF.
varAny (see page 22678)	Represents a Variant (see page 22307) that can hold any value.
varArray (see page 22679)	Represents a Variant (see page 22307) that holds dynamic array data.
varBoolean (see page 22680)	Represents a Variant (see page 22307) that holds WordBool data.
varByRef (see page 22680)	Represents a Variant (see page 22307) that points to another variable.
varByte (see page 22681)	Represents a Variant (see page 22307) that holds byte data.

varCurrency (see page 22681)	Represents a Variant (see page 22307) that holds a monetary floating point value.
varDate (see page 22682)	Represents a Variant (see page 22307) that holds a TDateTime (see page 22256) value.
varDispatch (see page 22682)	Represents a Variant (see page 22307) that has been assigned an IDispatch (see page 22394) value.
varDouble (see page 22683)	Represents a Variant (see page 22307) that holds a floating point value.
varEmpty (see page 22683)	Represents a Variant (see page 22307) that has not yet been assigned a value.
varError (see page 22684)	Represents a Variant (see page 22307) that holds a 32 bit error code.
varInt64 (see page 22684)	Represents a Variant (see page 22307) that holds Int64 data.
varInteger (see page 22685)	Represents a Variant (see page 22307) that holds Integer data.
varLongWord (see page 22685)	Represents a Variant (see page 22307) that holds LongWord data.
varNull (see page 22686)	Represents a Variant (see page 22307) that holds a null value.
varOleStr (see page 22686)	Represents a Variant (see page 22307) that has been assigned a COM interface type string value.
varShortInt (see page 22687)	Represents a Variant (see page 22307) that holds ShortInt data.
varSingle (see page 22687)	Represents a Variant (see page 22307) that holds byte data.
varSmallint (see page 22688)	This is constant System.varSmallint.
varStrArg (see page 22688)	Represents a Variant (see page 22307) that has holds a COM compatible string value.
varString (see page 22688)	Represents a Variant (see page 22307) that holds string data.
varTypeMask (see page 22689)	Defines a mask for filtering out the meta data part of a Variant (see page 22307) data type.
varUnknown (see page 22690)	Represents a Variant (see page 22307) that holds an unknown value.
varVariant (see page 22690)	Represents a Variant (see page 22307) data type.
varWord (see page 22691)	Represents a Variant (see page 22307) that holds Word data.
vmtAddRef (see page 22691)	This is constant System.vmtAddRef.
vmtAfterConstruction (see page 22691)	This is constant System.vmtAfterConstruction.
vmtAutoTable (see page 22692)	This is constant System.vmtAutoTable.
vmtBeforeDestruction (see page 22692)	This is constant System.vmtBeforeDestruction.
vmtClassName (see page 22692)	This is constant System.vmtClassName.
vmtCreateObject (see page 22693)	This is constant System.vmtCreateObject.
vmtDefaultHandler (see page 22693)	This is constant System.vmtDefaultHandler.
vmtDestroy (see page 22693)	This is constant System.vmtDestroy.
vmtDispatch (see page 22694)	This is constant System.vmtDispatch.
vmtDynamicTable (see page 22694)	This is constant System.vmtDynamicTable.
vmtFieldTable (see page 22694)	This is constant System.vmtFieldTable.
vmtFreeInstance (see page 22695)	This is constant System.vmtFreeInstance.
vmtInitTable (see page 22695)	This is constant System.vmtInitTable.
vmtInstanceSize (see page 22695)	This is constant System.vmtInstanceSize.
vmtIntfTable (see page 22696)	This is constant System.vmtIntfTable.
vmtMethodTable (see page 22696)	This is constant System.vmtMethodTable.
vmtNewInstance (see page 22696)	This is constant System.vmtNewInstance.
vmtParent (see page 22697)	This is constant System.vmtParent.
vmtQueryInterface (see page 22697)	This is constant System.vmtQueryInterface.
vmtRelease (see page 22697)	This is constant System.vmtRelease.
vmtSafeCallException (see page 22698)	This is constant System.vmtSafeCallException.
vmtSelfPtr (see page 22698)	This is constant System.vmtSelfPtr.
vmtTypeInfo (see page 22698)	This is constant System.vmtTypeInfo.
vtAnsiString (see page 22699)	vtAnsiString defines an AnsiString (see page 22093) TVarData (see page 22571) VType value.
vtBoolean (see page 22699)	vtBoolean defines a Boolean TVarData (see page 22571) VType value.
vtChar (see page 22700)	vtChar defines a Char TVarData (see page 22571) VType value.
vtClass (see page 22700)	vtClass defines a TClass (see page 22611) TVarData (see page 22571) VType value.

vtCurrency (see page 22701)	vtCurrency defines a Currency (see page 22145) TVarData (see page 22571) VType value.
vtExtended (see page 22701)	vtExtended defines an Extended TVarData (see page 22571) VType value.
vtInt64 (see page 22702)	vtInt64 defines an Int64 TVarData (see page 22571) VType value.
vtInteger (see page 22702)	vtInteger defines an Integer TVarData (see page 22571) VType value.
vtInterface (see page 22703)	vtInterface defines an Interface pointer TVarData (see page 22571) VType value.
vtObject (see page 22703)	vtObject defines a TObject (see page 22285) TVarData (see page 22571) VType value.
vtPChar (see page 22704)	vtPChar defines a PChar TVarData (see page 22571) VType value.
vtPointer (see page 22704)	vtPointer defines a Pointer TVarData (see page 22571) VType value.
vtPWideChar (see page 22705)	vtPWideChar defines a PWideChar TVarData (see page 22571) VType value.
vtString (see page 22705)	vtString defines a Variant (see page 22307) TVarData (see page 22571) VType value.
vtVariant (see page 22706)	This is constant System.vtVariant.
vtWideChar (see page 22706)	vtWideChar defines a WideChar TVarData (see page 22571) VType value.
vtWideString (see page 22706)	vtWideString defines a WideString (see page 22377) TVarData (see page 22571) VType value.

158.7.1 System.CompilerVersion Constant

Designates the version number of the Delphi compiler.

Pascal

```
CompilerVersion = 18.0;
```

C++

```
CompilerVersion = 18.0;
```

File

System

Description

Use CompilerVersion to test the version level of the Delphi compiler. This is useful if you want to conditionalize code that takes advantage of newer features in the Delphi language.

See Also

RTLVersion (see page 22677)

158.7.2 System.CPUi386 Constant

Pascal

```
CPUi386 = 2;
```

C++

```
CPUi386 = 2;
```

File

System

Description

This is constant System.CPUi386.

158.7.3 System.CPUi486 Constant

Pascal

```
CPUi486 = 3;
```

C++

```
CPUi486 = 3;
```

File

System

Description

This is constant System.CPUi486.

158.7.4 System.CPUPentium Constant

Pascal

```
CPUPentium = 4;
```

C++

```
CPUPentium = 4;
```

File

System

Description

This is constant System.CPUPentium.

158.7.5 System.E_NOINTERFACE Constant

Pascal

```
E_NOINTERFACE = HRESULT($80004002);
```

C++

```
E_NOINTERFACE = HRESULT($80004002);
```

File

System

Description

This is constant System.E_NOINTERFACE.

158.7.6 System.E_NOTIMPL Constant

Pascal

```
E_NOTIMPL = HRESULT($80004001);
```

C++

```
E_NOTIMPL = HRESULT($80004001);
```

File

System

Description

This is constant System.E_NOTIMPL.

158.7.7 System.E_UNEXPECTED Constant

Pascal

```
E_UNEXPECTED = HRESULT($8000FFFF);
```

C++

```
E_UNEXPECTED = HRESULT($8000FFFF);
```

File

System

Description

This is constant System.E_UNEXPECTED.

158.7.8 System.fmClosed Constant

Pascal

```
fmClosed = $D7B0;
```

C++

```
fmClosed = 0xD7B0;
```

File

System

Description

This is constant System.fmClosed.

158.7.9 System.fmInOut Constant

Pascal

```
fmInOut = $D7B3;
```

C++

```
fmInOut = 0xD7B3;
```

File

System

Description

This is constant System.fmInOut.

158.7.10 System.fmInput Constant

Pascal

```
fmInput = $D7B1;
```

C++

```
fmInput = 0xD7B1;
```

File

System

Description

This is constant System.fmInput.

158.7.11 System.fmOutput Constant

Pascal

```
fmOutput = $D7B2;
```

C++

```
fmOutput = 0xD7B2;
```

File

System

Description

This is constant System.fmOutput.

158.7.12 System.MaxInt Constant

The maximum value of the Integer data type.

Pascal

```
MaxInt = 2147483647;
```

C++

```
MaxInt = 2147483647;
```

File

System

Description

MaxInt represents the highest value in the range of the integer data type (2147483647).

See Also

High (see page 22476)

Integer

158.7.13 System.MaxLongint Constant

The maximum value of the Longint data type.

Pascal

```
MaxLongint = 2147483647;
```

C++

```
MaxLongint = 2147483647;
```

File

System

Description

MaxLongint represents the highest value in the range of the Longint (Delphi) or int (C++) data type (2147483647).

See Also

High (see page 22476)

Longint

158.7.14 System.NumSmallBlockTypes Constant

NumSmallBlockTypes is the number of small memory block types employed by the default memory manager.

Pascal

```
NumSmallBlockTypes = 55;
```

C++

```
NumSmallBlockTypes = 55;
```

File

System

Description

GetMemoryManagerState (see page 22472) returns a structure containing memory usage statistics for each block type - small, medium and large blocks. It returns this data in a TMemoryManagerState (see page 22564) record. Within that record is held a TSmallBlockTypeStates (see page 22620) field - an array of TSmallBlockTypeState (see page 22568) records. There are NumSmallBlockTypes (55) different small memory block sizes, and therefore 55 TSmallBlockTypeState (see page 22568) records in this array.

See Also

GetMemoryManagerState (see page 22472)

TMemoryManagerState (see page 22564)

TSmallBlockTypeState (see page 22568)

TSmallBlockTypeStates (see page 22620)

158.7.15 System.opAdd Constant

opAdd defines an addition operation.

Pascal

```
opAdd = 0;
```

C++

```
opAdd = 0;
```

File

System

Description

opAdd is one of the operation types that TVarOp (see page 22624) may be set to when performing binary operations such as BinaryOp.

opAdd represents an addition of two operands.

See Also

BinaryOp

BinaryOp

TVarOp (see page 22624)

158.7.16 System.opAnd Constant

opAnd defines a logical and operation.

Pascal

```
opAnd = 8;
```

C++

```
opAnd = 8;
```

File

System

Description

opAnd is one of the operation types that TVarOp (see page 22624) may be set to when performing binary operations such as BinaryOp.

opAnd represents a logical and of two operands.

See Also

BinaryOp

BinaryOp

TVarOp (see page 22624)

158.7.17 System.opCmpEQ Constant

opCmpEQ defines an equality comparison operator.

Pascal

```
opCmpEQ = 14;
```

C++

```
opCmpEQ = 14;
```

File

System

Description

opCmpEQ is one of the opCmp* comparison values that TVarOp (see page 22624) may be set to when performing left/right operand comparisons, such as CompareOp.

opCmpEQ represents a comparison that two operands are equal in value.

See Also

CompareOp (see page 23590)

CompareOp

TVarOp (see page 22624)

158.7.18 System.opCmpGE Constant

opCmpGE defines a greater than or equal comparison operator.

Pascal

```
opCmpGE = 19;
```

C++

```
opCmpGE = 19;
```

File

System

Description

opCmpGE is one of the opCmp* comparison values that TVarOp (see page 22624) may be set to when performing left/right operand comparisons, such as CompareOp.

opCmpGE represents the comparison : left operand greater than or equal to the right operand.

See Also

CompareOp (see page 23590)

CompareOp

TVarOp (see page 22624)

158.7.19 System.opCmpGT Constant

opCmpGT defines a greater than comparison operator.

Pascal

```
opCmpGT = 18;
```

C++

```
opCmpGT = 18;
```

File

System

Description

opCmpGT is one of the opCmp* comparison values that TVarOp (see page 22624) may be set to when performing left/right operand comparisons, such as CompareOp.

opCmpGT represents the comparison : left operand greater than the right operand.

See Also

CompareOp (see page 23590)

CompareOp

TVarOp (see page 22624)

158.7.20 System.opCmpLE Constant

opCmpLE defines a less than or equal comparison operator.

Pascal

```
opCmpLE = 17;
```

C++

```
opCmpLE = 17;
```

File

System

Description

opCmpLE is one of the opCmp* comparison values that TVarOp (see page 22624) may be set to when performing left/right operand comparisons, such as CompareOp.

opCmpLE represents the comparison : left operand less than or equal to the right operand.

See Also

CompareOp (see page 23590)

CompareOp

TVarOp (see page 22624)

158.7.21 System.opCmpLT Constant

opCmpLT defines a less than comparison operator.

Pascal

```
opCmpLT = 16;
```

C++

```
opCmpLT = 16;
```

File

System

Description

opCmpLT is one of the opCmp* comparison values that TVarOp (see page 22624) may be set to when performing left/right operand comparisons, such as CompareOp.

opCmpLT represents the comparison : left operand less than the right operand.

See Also

CompareOp (see page 23590)

CompareOp

TVarOp (see page 22624)

158.7.22 System.opCmpNE Constant

opCmpNE defines an inequality comparison operator.

Pascal

```
opCmpNE = 15;
```

C++

```
opCmpNE = 15;
```

File

System

Description

opCmpNE is one of the opCmp* comparison values that TVarOp (see page 22624) may be set to when performing left/right operand comparisons, such as CompareOp.

opCmpNE represents a comparison that two operands are not equal in value.

See Also

CompareOp (see page 23590)

CompareOp

TVarOp (see page 22624)

158.7.23 System.opCompare Constant

opCompare defines a comparison operation.

Pascal

```
opCompare = 11;
```

C++

```
opCompare = 11;
```

File

System

Description

opCompare defines a comparison operation.

See Also

TVarOp (see page 22624)

158.7.24 System.opDivide Constant

opDivide defines a divide operation.

Pascal

```
opDivide = 3;
```

C++

```
opDivide = 3;
```

File

System

Description

opDivide is one of the operation types that TVarOp (see page 22624) may be set to when performing binary operations such as BinaryOp.

opDivide represents the division of the left operand by the right operation.

See Also

BinaryOp

BinaryOp

TVarOp (see page 22624)

158.7.25 System.opIntDivide Constant

opIntDivide defines an integer divide operation.

Pascal

```
opIntDivide = 4;
```

C++

```
opIntDivide = 4;
```

File

System

Description

opIntDivide is one of the operation types that TVarOp (see page 22624) may be set to when performing binary operations such as BinaryOp.

opIntDivide represents the integer division of the left operand by the right operation.

See Also

BinaryOp

BinaryOp

TVarOp (see page 22624)

158.7.26 System.opModulus Constant

opModulus defines a modulus operation.

Pascal

```
opModulus = 5;
```

C++

```
opModulus = 5;
```

File

System

Description

opModulus is one of the operation types that TVarOp (see page 22624) may be set to when performing binary operations such as BinaryOp.

opModulus represents the remainder of the division of the left operand by the right operand.

See Also

BinaryOp

BinaryOp

TVarOp (see page 22624)

158.7.27 System.opMultiply Constant

opMultiply defines a multiply operation.

Pascal

```
opMultiply = 2;
```

C++

```
opMultiply = 2;
```

File

System

Description

opMultiply is one of the operation types that TVarOp (see page 22624) may be set to when performing binary operations such as BinaryOp.

opMultiply represents the multiplication of the left operand by the right operation.

See Also

BinaryOp

BinaryOp

TVarOp (see page 22624)

158.7.28 System.opNegate Constant

opNegate defines a negate operation.

Pascal

```
opNegate = 12;
```

C++

```
opNegate = 12;
```

File

System

Description

opNegate is one of the operation types that TVarOp (see page 22624) may be set to when performing unary operations such as UnaryOp.

opNegate represents the inversion of an operand value.

See Also

UnaryOp

TVarOp (see page 22624)

158.7.29 System.opNot Constant

opNot defines a binary negation operation.

Pascal

```
opNot = 13;
```

C++

```
opNot = 13;
```

File

System

Description

opNegate (see page 22673) represents the binary (bitwise) inversion of an operand value.

See Also

TVarOp (see page 22624)

158.7.30 System.opOr Constant

opOr defines a logical or operation.

Pascal

```
opOr = 9;
```

C++

```
opOr = 9;
```

File

System

Description

opOr is one of the operation types that TVarOp (see page 22624) may be set to when performing binary operations such as BinaryOp.

opOr represents a logical or of two operands.

See Also

BinaryOp

BinaryOp

TVarOp (see page 22624)

158.7.31 System.opShiftLeft Constant

opShiftLeft defines a bitwise shift left operation.

Pascal

```
opShiftLeft = 6;
```

C++

```
opShiftLeft = 6;
```

File

System

Description

opShiftLeft is one of the operation types that TVarOp (see page 22624) may be set to when performing binary operations such as BinaryOp.

opShiftLeft represents the left operand shifted left N times, where N is the numeric value of the right operand.

See Also

BinaryOp

BinaryOp

TVarOp (see page 22624)

158.7.32 System.opShiftRight Constant

opShiftRight defines a bitwise shift right operation.

Pascal

```
opShiftRight = 7;
```

C++

```
opShiftRight = 7;
```

File

System

Description

opShiftRight is one of the operation types that TVarOp (see page 22624) may be set to when performing binary operations such as BinaryOp.

opShiftRight represents the left operand shifted right N times, where N is the numeric value of the right operand.

See Also

BinaryOp

BinaryOp

TVarOp (see page 22624)

158.7.33 System.opSubtract Constant

opSubtract defines a subtraction operation.

Pascal

```
opSubtract = 1;
```

C++

```
opSubtract = 1;
```

File

System

Description

opSubtract is one of the operation types that TVarOp (see page 22624) may be set to when performing binary operations such as BinaryOp.

opSubtract represents the right operand subtracted from the left operand.

See Also

BinaryOp

BinaryOp

TVarOp (see page 22624)

158.7.34 System.opXor Constant

opXor defines a logical exclusive or operation.

Pascal

```
opXor = 10;
```

C++

```
opXor = 10;
```

File

System

Description

opXor is one of the operation types that TVarOp (see page 22624) may be set to when performing binary operations such as BinaryOp.

opXor represents a logical exclusive or of two operands.

See Also

BinaryOp

BinaryOp

TVarOp (see page 22624)

158.7.35 System.RTLVersion Constant

Designates the version number of the Delphi runtime library.

Pascal

```
RTLVersion = 18.00;
```

C++

```
RTLVersion = 18.00;
```

File

System

Description

Use RTLVersion to test the Delphi runtime library version level independently of the compiler version level.

See Also

CompilerVersion (see page 22662)

158.7.36 System.S_FALSE Constant

Pascal

```
S_FALSE = $00000001;
```

C++

```
S_FALSE = 0x00000001;
```

File

System

Description

This is constant System.S_FALSE.

158.7.37 System.S_OK Constant

Pascal

```
S_OK = 0;
```

C++

```
S_OK = 0;
```

File

System

Description

This is constant System.S_OK.

158.7.38 System.sLineBreak Constant

Represents the character(s) used to generate a line feed.

Pascal

```
sLineBreak = #13#10;
```

C++

```
sLineBreak = #13#10;
```

File

System

Description

Windows systems define sLineBreak as a combination of carriage return (#13) and line feed (#10).

Linux systems define sLineBreak as simply a line feed (#10).

It is preferable to use this constabnt rather than hard code #13#10 or #10 (as appropriate) to move to a new line in a text output.

158.7.39 System.tfCRLF Constant

Pascal

```
tfCRLF = $1;
```

C++

```
tfCRLF = 0x1;
```

File

System

Description

This is constant System.tfCRLF.

158.7.40 System.varAny Constant

Represents a Variant (see page 22307) that can hold any value.

Pascal

```
varAny = $0101;
```

C++

```
varAny = 0x0101;
```

File

System

Description

A variant with a varAny type is used as a placeholder for any variant value, such as when passed as a paremeter to a routine. A varAny variant cannot be directly processed - use ChangeAnyProc to convert it to a variant type that can be processed.

See Also

TVarType

ChangeAnyProc (see page 23677)

VarType (see page 23667)

158.7.41 System.varArray Constant

Represents a Variant (see page 22307) that holds dynamic array data.

Pascal

```
varArray = $2000;
```

C++

```
varArray = 0x2000;
```

File

System

Description

The VarType routine returns a TVarType (see page 22625) value that describes the current data type used to hold the contents of a Variant (see page 22307) data type. The data type is a 16 bit value, split into a high order 4 bits and a low order 12 bits. The low order bits determine the basic data type, such as varDouble (see page 22683), varSmallInt.

The high order bits represent meta data types varArray and varByRef (see page 22680).

Logically And varArray with a Variant (see page 22307) to determine if a Variant (see page 22307) holds array data.

Use VarArrayCreate to generate an array Variant (see page 22307);

See Also

TVarType

VarArrayCreate (see page 23644)

VarType (see page 23667)

158.7.42 System.varBoolean Constant

Represents a Variant (see page 22307) that holds WordBool data.

Pascal

```
varBoolean = $000B;
```

C++

```
varBoolean = 0x000B;
```

File

System

Description

varBoolean represents a Variant (see page 22307) variable that currently holds a 16-bit boolean (type WordBool) value.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 23650)

VarType (see page 23667)

158.7.43 System.varByRef Constant

Represents a Variant (see page 22307) that points to another variable.

Pascal

```
varByRef = $4000;
```

C++

```
varByRef = 0x4000;
```

File

System

Description

The VarType routine returns a TVarType (see page 22625) value that describes the current data type used to hold the contents of a Variant (see page 22307) data type. The data type is a 16 bit value, split into a high order 4 bits and a low order 12 bits. The low order bits determine the basic data type, such as varDouble (see page 22683), varSmallInt.

The high order bits represent meta data types varArray (see page 22679) and varByRef.

In essence, a varByRef variant is a pointer. The data type of the variable it points to is determined in the low order bits.

Logically And varByRef with a Variant (see page 22307) to determine if a Variant (see page 22307) is a pointer.

See Also

TVarType

VarType (see page 23667)

158.7.44 System.varByte Constant

Represents a Variant (see page 22307) that holds byte data.

Pascal

```
varByte = $0011;
```

C++

```
varByte = 0x0011;
```

File

System

Description

When a Variant (see page 22307) is assigned an integer value less than 256, it will be set to a varByte Variant (see page 22307) type. varByte represents a Variant (see page 22307) variable that currently holds a byte data value.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 23650)

VarType (see page 23667)

158.7.45 System.varCurrency Constant

Represents a Variant (see page 22307) that holds a monetary floating point value.

Pascal

```
varCurrency = $0006;
```

C++

```
varCurrency = 0x0006;
```

File

System

Description

When a Variant (see page 22307) is assigned a floating point value with less than 5 decimal places, it is set to the varCurrency type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 23650)

VarType (see page 23667)

158.7.46 System.varDate Constant

Represents a Variant (see page 22307) that holds a TDateTime (see page 22256) value.

Pascal

```
varDate = $0007;
```

C++

```
varDate = 0x0007;
```

File

System

Description

When a Variant (see page 22307) is assignned a TDateTime (see page 22256) value, it is set to the varDate type.

Use VarType to determine the variant type.

See Also

TDateTime (see page 22612)

TVarType

VarAsType (see page 23650)

VarType (see page 23667)

158.7.47 System.varDispatch Constant

Represents a Variant (see page 22307) that has been assigned an IDispatch (see page 22394) value.

Pascal

```
varDispatch = $0009;
```

C++

```
varDispatch = 0x0009;
```

File

System

Description

When a Variant (see page 22307) is assigned an IDispatch (see page 22394) value, it is set to the varDispatch type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 23650)

VarType (see page 23667)

158.7.48 System.varDouble Constant

Represents a Variant (see page 22307) that holds a floating point value.

Pascal

```
varDouble = $0005;
```

C++

```
varDouble = 0x0005;
```

File

System

Description

When a Variant (see page 22307) is assigned a floating point value with more than 4 decimal places, it is set to the varDouble type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 23650)

VarType (see page 23667)

158.7.49 System.varEmpty Constant

Represents a Variant (see page 22307) that has not yet been assigned a value.

Pascal

```
varEmpty = $0000;
```

C++

```
varEmpty = 0x0000;
```

File

System

Description

When a Variant (see page 22307) is first created, it is set to the varEmpty type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 23650)

VarIsEmpty (see page 23657)

VarType (see page 23667)

158.7.50 System.varError Constant

Represents a Variant (see page 22307) that holds a 32 bit error code.

Pascal

```
varError = $000A;
```

C++

```
varError = 0x000A;
```

File

System

Description

When a Variant (see page 22307) is assigned a 32 bit error code, it is set to the varError type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 23650)

VarIsEmpty (see page 23657)

VarType (see page 23667)

158.7.51 System.varInt64 Constant

Represents a Variant (see page 22307) that holds Int64 data.

Pascal

```
varInt64 = $0014;
```

C++

```
varInt64 = 0x0014;
```

File

System

Description

When a Variant (see page 22307) is assigned an integer value greater in value than an LongWord type can hold, it will be set to a varInt64 Variant (see page 22307) type. varInt64 represents a Variant (see page 22307) variable that currently holds an Int64 data value.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 23650)

VarType (see page 23667)

158.7.52 System.varInteger Constant

Represents a Variant (see page 22307) that holds Integer data.

Pascal

```
varInteger = $0003;
```

C++

```
varInteger = 0x0003;
```

File

System

Description

To define a Variant (see page 22307) to be a varInteger type, you should use VarAsType to create one. By default, a Variant (see page 22307) with an integer value greater than the Word type capacity, but less than the Int64 type capacity will normally be assigned a varLongWord (see page 22685) type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 23650)

VarType (see page 23667)

158.7.53 System.varLongWord Constant

Represents a Variant (see page 22307) that holds LongWord data.

Pascal

```
varLongWord = $0013;
```

C++

```
varLongWord = 0x0013;
```

File

System

Description

When a Variant (see page 22307) is assigned an integer value greater in value than a Word type can hold, but less than an Int64 type can hold, it will be set to a varLongWord Variant (see page 22307) type. varLongWord represents a Variant (see page 22307) variable that currently holds a LongWord data value.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 23650)

VarType (see page 23667)

158.7.54 System.varNull Constant

Represents a Variant (see page 22307) that holds a null value.

Pascal

```
varNull = $0001;
```

C++

```
varNull = 0x0001;
```

File

System

Description

When a Variant (see page 22307) is assigned a null value, it is set to the varNull type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 23650)

VarIsNull (see page 23660)

VarType (see page 23667)

158.7.55 System.varOleStr Constant

Represents a Variant (see page 22307) that has been assigned a COM interface type string value.

Pascal

```
varOleStr = $0008;
```

C++

```
varOleStr = 0x0008;
```

File

System

Description

When a Variant (see page 22307) is assigned a string value via a COM interface, it is set to the varOleStr type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 23650)

VarType (see page 23667)

158.7.56 System.varShortInt Constant

Represents a Variant (see page 22307) that holds ShortInt data.

Pascal

```
varShortInt = $0010;
```

C++

```
varShortInt = 0x0010;
```

File

System

Description

To define a Variant (see page 22307) to be a varShortInt type, you should use VarAsType to create one. By default, a Variant (see page 22307) with an integer value less than the Word type capacity will normally be assigned a varByte (see page 22681) type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 23650)

VarType (see page 23667)

158.7.57 System.varSingle Constant

Represents a Variant (see page 22307) that holds byte data.

Pascal

```
varSingle = $0004;
```

C++

```
varSingle = 0x0004;
```

File

System

Description

To define a Variant (see page 22307) to be a varSingle type, you should use VarAsType to create one. By default, a Variant (see page 22307) with a floating point value with less than 4 decimal places will be assigned the varCurrency (see page 22681) type. With greater than 4 decimal places, it will be assigned by default the varDouble (see page 22683) type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 23650)

VarType (see page 23667)

158.7.58 System.varSmallint Constant

Pascal

```
varSmallint = $0002;
```

C++

```
varSmallint = 0x0002;
```

File

System

Description

This is constant System.varSmallint.

158.7.59 System.varStrArg Constant

Represents a Variant (see page 22307) that has holds a COM compatible string value.

Pascal

```
varStrArg = $0048;
```

C++

```
varStrArg = 0x0048;
```

File

System

Description

In order to use a Variant (see page 22307) with a COM interface, you should create a varStrArg type using VarAsType.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 23650)

VarType (see page 23667)

158.7.60 System.varString Constant

Represents a Variant (see page 22307) that holds string data.

Pascal

```
varString = $0100;
```

C++

```
varString = 0x0100;
```

File

System

Description

When a Variant (see page 22307) is assigned a string value, it will be set to a varString Variant (see page 22307) type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 23650)

VarType (see page 23667)

158.7.61 System.varTypeMask Constant

Defines a mask for filtering out the meta data part of a Variant (see page 22307) data type.

Pascal

```
varTypeMask = $0FFF;
```

C++

```
varTypeMask = 0x0FFF;
```

File

System

Description

The VarType routine returns a TVarType (see page 22625) value that describes the current data type used to hold the contents of a Variant (see page 22307) data type. The data type is a 16 bit value, split into a high order 4 bits and a low order 12 bits. The high order bits give a meta data type (varArray (see page 22679) and varByRef (see page 22680)). The low order bits determine the basic data type, such as varDouble (see page 22683), varSmallInt. Use varTypeMask to mask out the meta data so as to be able to determine the basic data type of a Variant (see page 22307).

See Also

TVarType

varArray (see page 22679)

VarAsType (see page 23650)

varByRef (see page 22680)

VarType (see page 23667)

158.7.62 System.varUnknown Constant

Represents a Variant (see page 22307) that holds an unknown value.

Pascal

```
varUnknown = $000D;
```

C++

```
varUnknown = 0x000D;
```

File

System

Description

When a Variant (see page 22307) is assignned a value that is not recognised, it is set to the varUnknown type.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 23650)

VarType (see page 23667)

158.7.63 System.varVariant Constant

Represents a Variant (see page 22307) data type.

Pascal

```
varVariant = $000C;
```

C++

```
varVariant = 0x000C;
```

File

System

Description

varVariant is principally used when using VarArrayCreate to create an array of Variants (see page 23567). It determines that such an arrray has unspecified Variant (see page 22307) types. Use such an array when element contents may differ.

See Also

TVarType

VarArrayCreate (see page 23644)

VarType (see page 23667)

158.7.64 System.varWord Constant

Represents a Variant (see page 22307) that holds Word data.

Pascal

```
varWord = $0012;
```

C++

```
varWord = 0x0012;
```

File

System

Description

When a Variant (see page 22307) is assigned an integer value greater in value than a Byte type can hold, but less than an LongWord type can hold, it will be set to a varWord Variant (see page 22307) type. varWord represents a Variant (see page 22307) variable that currently holds a Word data value.

Use VarType to determine the variant type.

See Also

TVarType

VarAsType (see page 23650)

VarType (see page 23667)

158.7.65 System.vmtAddRef Constant

Pascal

```
vmtAddRef = 4 deprecated;
```

C++

```
vmtAddRef = 4 deprecated;
```

File

System

Description

This is constant System.vmtAddRef.

158.7.66 System.vmtAfterConstruction Constant

Pascal

```
vmtAfterConstruction = -28 deprecated;
```

C++

```
vmtAfterConstruction = -28 deprecated;
```

File

System

Description

This is constant System.vmtAfterConstruction.

158.7.67 System.vmtAutoTable Constant

Pascal

```
vmtAutoTable = -68;
```

C++

```
vmtAutoTable = -68;
```

File

System

Description

This is constant System.vmtAutoTable.

158.7.68 System.vmtBeforeDestruction Constant

Pascal

```
vmtBeforeDestruction = -24 deprecated;
```

C++

```
vmtBeforeDestruction = -24 deprecated;
```

File

System

Description

This is constant System.vmtBeforeDestruction.

158.7.69 System.vmtClassName Constant

Pascal

```
vmtClassName = -44;
```

C++

```
vmtClassName = -44;
```

File

System

Description

This is constant System.vmtClassName.

158.7.70 System.vmtCreateObject Constant

Pascal

```
vmtCreateObject = 12 deprecated;
```

C++

```
vmtCreateObject = 12 deprecated;
```

File

System

Description

This is constant System.vmtCreateObject.

158.7.71 System.vmtDefaultHandler Constant

Pascal

```
vmtDefaultHandler = -16 deprecated;
```

C++

```
vmtDefaultHandler = -16 deprecated;
```

File

System

Description

This is constant System.vmtDefaultHandler.

158.7.72 System.vmtDestroy Constant

Pascal

```
vmtDestroy = -4 deprecated;
```

C++

```
vmtDestroy = -4 deprecated;
```

File

System

Description

This is constant System.vmtDestroy.

158.7.73 System.vmtDispatch Constant

Pascal

```
vmtDispatch = -20 deprecated;
```

C++

```
vmtDispatch = -20 deprecated;
```

File

System

Description

This is constant System.vmtDispatch.

158.7.74 System.vmtDynamicTable Constant

Pascal

```
vmtDynamicTable = -48;
```

C++

```
vmtDynamicTable = -48;
```

File

System

Description

This is constant System.vmtDynamicTable.

158.7.75 System.vmtFieldTable Constant

Pascal

```
vmtFieldTable = -56;
```

C++

```
vmtFieldTable = -56;
```

File

System

Description

This is constant System.vmtFieldTable.

158.7.76 System.vmtFreeInstance Constant

Pascal

```
vmtFreeInstance = -8 deprecated;
```

C++

```
vmtFreeInstance = -8 deprecated;
```

File

System

Description

This is constant System.vmtFreeInstance.

158.7.77 System.vmtInitTable Constant

Pascal

```
vmtInitTable = -64;
```

C++

```
vmtInitTable = -64;
```

File

System

Description

This is constant System.vmtInitTable.

158.7.78 System.vmtInstanceSize Constant

Pascal

```
vmtInstanceSize = -40;
```

C++

```
vmtInstanceSize = -40;
```

File

System

Description

This is constant System.vmtInstanceSize.

158.7.79 System.vmtIntfTable Constant

Pascal

```
vmtIntfTable = -72;
```

C++

```
vmtIntfTable = -72;
```

File

System

Description

This is constant System.vmtIntfTable.

158.7.80 System.vmtMethodTable Constant

Pascal

```
vmtMethodTable = -52;
```

C++

```
vmtMethodTable = -52;
```

File

System

Description

This is constant System.vmtMethodTable.

158.7.81 System.vmtNewInstance Constant

Pascal

```
vmtNewInstance = -12 deprecated;
```

C++

```
vmtNewInstance = -12 deprecated;
```

File

System

Description

This is constant System.vmtNewInstance.

158.7.82 System.vmtParent Constant

Pascal

```
vmtParent = -36;
```

C++

```
vmtParent = -36;
```

File

System

Description

This is constant System.vmtParent.

158.7.83 System.vmtQueryInterface Constant

Pascal

```
vmtQueryInterface = 0 deprecated;
```

C++

```
vmtQueryInterface = 0 deprecated;
```

File

System

Description

This is constant System.vmtQueryInterface.

158.7.84 System.vmtRelease Constant

Pascal

```
vmtRelease = 8 deprecated;
```

C++

```
vmtRelease = 8 deprecated;
```

File

System

Description

This is constant System.vmtRelease.

158.7.85 System.vmtSafeCallException Constant

Pascal

```
vmtSafeCallException = -32 deprecated;
```

C++

```
vmtSafeCallException = -32 deprecated;
```

File

System

Description

This is constant System.vmtSafeCallException.

158.7.86 System.vmtSelfPtr Constant

Pascal

```
vmtSelfPtr = -76;
```

C++

```
vmtSelfPtr = -76;
```

File

System

Description

This is constant System.vmtSelfPtr.

158.7.87 System.vmtTypeInfo Constant

Pascal

```
vmtTypeInfo = -60;
```

C++

```
vmtTypeInfo = -60;
```

File

System

Description

This is constant System.vmtTypeInfo.

158.7.88 System.vtAnsiString Constant

vtAnsiString defines an AnsiString (see page 22093) TVarData (see page 22571) VType value.

Pascal

```
vtAnsiString = 11;
```

C++

```
vtAnsiString = 11;
```

File

System

Description

The TVarData (see page 22571) type represents the internal structure of a Variant (see page 22307) variable (on Windows, this is identical to the Variant (see page 22307) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 22307) it holds.

See Also

TVarData (see page 22571)

TVarRec (see page 22577)

158.7.89 System.vtBoolean Constant

vtBoolean defines a Boolean TVarData (see page 22571) VType value.

Pascal

```
vtBoolean = 1;
```

C++

```
vtBoolean = 1;
```

File

System

Description

The TVarData (see page 22571) type represents the internal structure of a Variant (see page 22307) variable (on Windows, this is identical to the Variant (see page 22307) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 22307) it holds.

See Also

TVarData (see page 22571)

TVarRec (see page 22577)

158.7.90 System.vtChar Constant

vtChar defines a Char TVarData (see page 22571) VType value.

Pascal

```
vtChar = 2;
```

C++

```
vtChar = 2;
```

File

System

Description

The TVarData (see page 22571) type represents the internal structure of a Variant (see page 22307) variable (on Windows, this is identical to the Variant (see page 22307) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 22307) it holds.

See Also

TVarData (see page 22571)

TVarRec (see page 22577)

158.7.91 System.vtClass Constant

vtClass defines a TClass (see page 22611) TVarData (see page 22571) VType value.

Pascal

```
vtClass = 8;
```

C++

```
vtClass = 8;
```

File

System

Description

The TVarData (see page 22571) type represents the internal structure of a Variant (see page 22307) variable (on Windows, this is identical to the Variant (see page 22307) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 22307) it holds.

See Also

TVarData (see page 22571)

TVarRec (see page 22577)

158.7.92 System.vtCurrency Constant

vtCurrency defines a Currency (see page 22145) TVarData (see page 22571) VType value.

Pascal

```
vtCurrency = 12;
```

C++

```
vtCurrency = 12;
```

File

System

Description

The TVarData (see page 22571) type represents the internal structure of a Variant (see page 22307) variable (on Windows, this is identical to the Variant (see page 22307) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 22307) it holds.

See Also

TVarData (see page 22571)

TVarRec (see page 22577)

158.7.93 System.vtExtended Constant

vtExtended defines an Extended TVarData (see page 22571) VType value.

Pascal

```
vtExtended = 3;
```

C++

```
vtExtended = 3;
```

File

System

Description

The TVarData (see page 22571) type represents the internal structure of a Variant (see page 22307) variable (on Windows, this is identical to the Variant (see page 22307) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 22307) it holds.

See Also

TVarData (see page 22571)

TVarRec (see page 22577)

158.7.94 System.vtInt64 Constant

vtInt64 defines an Int64 TVarData (see page 22571) VType value.

Pascal

```
vtInt64 = 16;
```

C++

```
vtInt64 = 16;
```

File

System

Description

The TVarData (see page 22571) type represents the internal structure of a Variant (see page 22307) variable (on Windows, this is identical to the Variant (see page 22307) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 22307) it holds.

See Also

TVarData (see page 22571)

TVarRec (see page 22577)

158.7.95 System.vtInteger Constant

vtInteger defines an Integer TVarData (see page 22571) VType value.

Pascal

```
vtInteger = 0;
```

C++

```
vtInteger = 0;
```

File

System

Description

The TVarData (see page 22571) type represents the internal structure of a Variant (see page 22307) variable (on Windows, this is identical to the Variant (see page 22307) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 22307) it holds.

See Also

TVarData (see page 22571)

TVarRec (see page 22577)

158.7.96 System.vtInterface Constant

vtInterface defines an Interface pointer TVarData (see page 22571) VType value.

Pascal

```
vtInterface = 14;
```

C++

```
vtInterface = 14;
```

File

System

Description

The TVarData (see page 22571) type represents the internal structure of a Variant (see page 22307) variable (on Windows, this is identical to the Variant (see page 22307) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 22307) it holds.

See Also

TVarData (see page 22571)

TVarRec (see page 22577)

158.7.97 System.vtObject Constant

vtObject defines a TObject (see page 22285) TVarData (see page 22571) VType value.

Pascal

```
vtObject = 7;
```

C++

```
vtObject = 7;
```

File

System

Description

The TVarData (see page 22571) type represents the internal structure of a Variant (see page 22307) variable (on Windows, this is identical to the Variant (see page 22307) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 22307) it holds.

See Also

TVarData (see page 22571)

TVarRec (see page 22577)

158.7.98 System.vtPChar Constant

vtPChar defines a PChar TVarData (see page 22571) VType value.

Pascal

```
vtPChar = 6;
```

C++

```
vtPChar = 6;
```

File

System

Description

The TVarData (see page 22571) type represents the internal structure of a Variant (see page 22307) variable (on Windows, this is identical to the Variant (see page 22307) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 22307) it holds.

See Also

TVarData (see page 22571)

TVarRec (see page 22577)

158.7.99 System.vtPointer Constant

vtPointer defines a Pointer TVarData (see page 22571) VType value.

Pascal

```
vtPointer = 5;
```

C++

```
vtPointer = 5;
```

File

System

Description

The TVarData (see page 22571) type represents the internal structure of a Variant (see page 22307) variable (on Windows, this is identical to the Variant (see page 22307) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 22307) it holds.

See Also

TVarData (see page 22571)

TVarRec (see page 22577)

158.7.100 System.vtPWideChar Constant

vtPWideChar defines a PWideChar TVarData (see page 22571) VType value.

Pascal

```
vtPWideChar = 10;
```

C++

```
vtPWideChar = 10;
```

File

System

Description

The TVarData (see page 22571) type represents the internal structure of a Variant (see page 22307) variable (on Windows, this is identical to the Variant (see page 22307) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 22307) it holds.

See Also

TVarData (see page 22571)

TVarRec (see page 22577)

158.7.101 System.vtString Constant

vtString defines a Variant (see page 22307) TVarData (see page 22571) VType value.

Pascal

```
vtString = 4;
```

C++

```
vtString = 4;
```

File

System

Description

The TVarData (see page 22571) type represents the internal structure of a Variant (see page 22307) variable (on Windows, this is identical to the Variant (see page 22307) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 22307) it holds.

See Also

TVarData (see page 22571)

TVarRec (see page 22577)

158.7.102 System.vtVariant Constant

Pascal

```
vtVariant = 13;
```

C++

```
vtVariant = 13;
```

File

System

Description

This is constant System.vtVariant.

158.7.103 System.vtWideChar Constant

vtWideChar defines a WideChar TVarData (see page 22571) VType value.

Pascal

```
vtWideChar = 9;
```

C++

```
vtWideChar = 9;
```

File

System

Description

The TVarData (see page 22571) type represents the internal structure of a Variant (see page 22307) variable (on Windows, this is identical to the Variant (see page 22307) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 22307) it holds.

See Also

TVarData (see page 22571)

TVarRec (see page 22577)

158.7.104 System.vtWideString Constant

vtWideString defines a WideString (see page 22377) TVarData (see page 22571) VType value.

Pascal

```
vtWideString = 15;
```

C++

```
vtWideString = 15;
```

File

System

Description

The TVarData (see page 22571) type represents the internal structure of a Variant (see page 22307) variable (on Windows, this is identical to the Variant (see page 22307) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 22307) it holds.

See Also

TVarData (see page 22571)

TVarRec (see page 22577)

159 SysUtils Namespace

This is namespace SysUtils.

Topics

Name	Description
Classes (see page 22721)	The following table lists classes in this documentation.
Interfaces (see page 22778)	The following table lists interfaces in this documentation.
Functions (see page 22780)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 22996)	The following table lists structs, records, enums in this documentation.
Types (see page 23012)	The following table lists types in this documentation.
Variables (see page 23021)	The following table lists variables in this documentation.
Constants (see page 23038)	The following table lists constants in this documentation.

Classes

	Name	Description
	EAbort (see page 22722)	EAbort is the exception class for errors that should not display an error message dialog box.
	EAbstractError (see page 22722)	EAbstractError is the exception class for attempts to call an abstract method.
	EAccessViolation (see page 22723)	EAccessViolation is the exception class for invalid memory access errors.
	EAssertionFailed (see page 22723)	EAssertionFailed is the exception class for Delphi assertion failure.
	EControlC (see page 22724)	EControlC is the exception class for user interrupts.
	EConvertError (see page 22724)	EConvertError is the exception class for string and object conversion errors.
	EDivByZero (see page 22726)	EDivByZero is the exception class for integer divide-by-zero errors.
	EExternal (see page 22727)	EExternal is the exception class that captures external error information.
	EExternalException (see page 22729)	EExternalException is the exception class for invalid exception codes.
	EHeapException (see page 22730)	EHeapException is the exception class for errors related to heap-allocated memory.
	EInOutError (see page 22732)	EInOutError is the exception class for file input/output errors.
	EIntError (see page 22735)	EIntError is the base class for integer math error exceptions.
	EIntfCastError (see page 22736)	EIntfCastError is the exception class for interface casting errors.
	EIntOverflow (see page 22736)	EIntOverflow is the exception class for integer calculations whose results are too large to fit in the allocated register.
	EInvalidCast (see page 22737)	EInvalidCast is an exception class for typecasting errors.
	EInvalidContainer (see page 22738)	This is class SysUtils.EInvalidContainer.
	EInvalidInsert (see page 22738)	This is class SysUtils.EInvalidInsert.
	EInvalidOp (see page 22738)	EInvalidOp is the exception class for undefined floating-point operations.
	EInvalidPointer (see page 22739)	EInvalidPointer is the exception class for invalid pointer operations.

	EMathError (see page 22739)	EMathError is the base exception class for floating-point math errors.
	EOSError (see page 22740)	EOSError is the exception class for operating system errors.
	EOutOfMemory (see page 22742)	EOutOfMemory is the exception class for unsuccessful attempts to allocate memory.
	EOverflow (see page 22743)	EOverflow is the exception class for floating-point register overflows.
	EPackageError (see page 22744)	EPackageError is the exception class for package-related errors.
	EPrivilege (see page 22744)	EPrivilege is the exception class for processor privilege violations.
	EPropReadOnly (see page 22745)	EPropReadOnly is the exception class for invalid attempts to write to a property using OLE automation.
	EPropWriteOnly (see page 22745)	EPropWriteOnly is the exception class for invalid attempts to read a property using OLE automation.
	ERangeError (see page 22745)	ERangeError is the exception for Delphi range errors.
	ESafecallException (see page 22747)	ESafecallException is the exception for problems with the safecall calling convention.
	EStackOverflow (see page 22747)	EStackOverflow is the exception class for stack overflows.
	EUnderflow (see page 22748)	EUnderflow is the exception class for values that are too small to be represented with a floating-point variable.
	EVariantError (see page 22748)	EVariantError is the exception class for errors involving Variant data types.
	EWin32Error (see page 22749)	EWin32Error is the exception class for Windows errors.
	Exception (see page 22750)	Exception is the base class for all runtime exceptions.
	EZeroDivide (see page 22761)	EZeroDivide is the exception class for floating-point divide-by-zero errors.
	TLanguages (see page 22762)	TLanguages lists all available Windows locales.
	TMultiReadExclusiveWriteSynchronizer (see page 22766)	TMultiReadExclusiveWriteSynchronizer protects memory in a multi-threaded application.
	TSimpleRWSync (see page 22772)	This is class SysUtils.TSimpleRWSync.
	TThreadLocalCounter (see page 22775)	This is class SysUtils.TThreadLocalCounter.

Constants

Name	Description
DateDelta (see page 23039)	Specifies the difference between TDateTime and TTimeStamp (see page 23009) starting values.
DefaultFalseBoolStr (see page 23039)	Specifies the default string representation of the boolean false value.
DefaultTrueBoolStr (see page 23040)	Specifies the default string representation of the boolean true value.
DriveDelim (see page 23041)	Specifies the character used to separate the Windows drive from the path in a file name.
faAnyFile (see page 23041)	Specifies any file type.
faArchive (see page 23042)	Specifies only Windows archived file types.
faDirectory (see page 23042)	Specifies only directory 'file' types.
faHidden (see page 23043)	Specifies only hidden file types.
faReadOnly (see page 23044)	Specifies only read only file types.
faSymLink (see page 23044)	Specifies only symbolic link file types.
faSysFile (see page 23045)	Specifies only system file types.
faVolumeID (see page 23045)	Specifies Volume ID file types.
fmOpenRead (see page 23046)	Specifies a file should be opened for reading only.
fmOpenReadWrite (see page 23047)	Specifies a file should be opened for reading and writing.
fmOpenWrite (see page 23047)	Specifies a file should be opened for writing only.
fmShareCompat (see page 23048)	This is constant SysUtils.fmShareCompat.
fmShareDenyNone (see page 23048)	This is constant SysUtils.fmShareDenyNone.
fmShareDenyRead (see page 23049)	This is constant SysUtils.fmShareDenyRead.
fmShareDenyWrite (see page 23049)	This is constant SysUtils.fmShareDenyWrite.
fmShareExclusive (see page 23049)	This is constant SysUtils.fmShareExclusive.
HoursPerDay (see page 23050)	Specifies the number of hours in a day.
MaxCurrency (see page 23050)	Defines the largest value supported for Currency data types.
MaxDateTime (see page 23051)	Specifies the highest supported TDateTime value.
MaxEraCount (see page 23051)	This is constant SysUtils.MaxEraCount.
MinCurrency (see page 23051)	Defines the largest negative value supported for Currency data types.
MinDateTime (see page 23052)	Specifies the earliest supported TDateTime value.

MinsPerDay (see page 23052)	Specifies the number of minutes in a day.
MinsPerHour (see page 23053)	Specifies the number of minutes in an hour.
MonthDays (see page 23054)	An array specifying how many days are in a month.
MSecsPerDay (see page 23054)	Specifies the number of milliseconds in a day.
MSecsPerSec (see page 23055)	Specifies the number of milliseconds in a second.
PathDelim (see page 23055)	Specifies a path delimiter character.
PathSep (see page 23055)	Specifies the character used to separate a sequence of file or directory names.
pfBCB4Produced (see page 23056)	Indicates that a package was produced by via C++.
pfConsumerBCB (see page 23056)	Indicates that C++ can be a consumer of a package.
pfConsumerCompat (see page 23057)	This is constant SysUtils.pfConsumerCompat.
pfConsumerDelphi (see page 23057)	Indicates that Delphi can be a consumer of a package.
pfConsumerMask (see page 23058)	Indicates the comsumer of a package.
pfDelphi4Produced (see page 23058)	Indicates that a package was produced by Delphi 4 or later.
pfDesignOnly (see page 23059)	Indicates that a package was compiled with {$DESIGNONLY ON}.
pfExeModule (see page 23059)	Indicates that a module is an executable.
pfIgnoreDupUnits (see page 23060)	Indicates that a package was compiled with {$DUPLICATEENABLE ON}.
pfLibraryModule (see page 23060)	Indicates that a module is a library.
pfModuleTypeMask (see page 23060)	Indicates that a package is a module.
pfNeverBuild (see page 23061)	Indicates that a package was compiled with the -z flag.
pfPackageModule (see page 23061)	Indicates that a module is a package.
pfProducerMask (see page 23062)	Indicates the producer of a package.
pfProducerUndefined (see page 23062)	Indicates that a package was produced by an unknown developmemt environment.
pfRunOnly (see page 23063)	Indicates that a package was compiled with {$RUNONLY ON}.
pfV3Produced (see page 23063)	Indicates that a package was produced by Delphi 3 or later.
SecsPerDay (see page 23064)	Specifies the number of seconds in a day.
SecsPerMin (see page 23064)	Specifies the number of seconds in a minute.
SwitchChars (see page 23065)	Specifies the prefix character(s) for program command line arguments.
ufImplicitUnit (see page 23065)	Indicates that a unit was implicitly imported into a package.
ufMainUnit (see page 23066)	Indicates that a unit is the main one in a project.
ufOrgWeakUnit (see page 23066)	This is constant SysUtils.ufOrgWeakUnit.
ufPackageUnit (see page 23067)	Indicates that a unit is a package.
ufWeakPackageUnit (see page 23067)	This is constant SysUtils.ufWeakPackageUnit.
ufWeakUnit (see page 23068)	Indicates that a unit is a weak unit.
UnixDateDelta (see page 23068)	Specifies the difference between TDateTime and Lunix date/time origins.

Functions

	Name	Description
	Abort (see page 22786)	Raises a silent exception.
	AddExitProc (see page 22787)	Adds a procedure to the exit list.
	AddTerminateProc (see page 22787)	Adds a terminate procedure to the system list of termination procedures.
	AdjustLineBreaks (see page 22788)	Adjusts line break characters to reflect Windows or Linux style.
	AnsiCompareFileName (see page 22788)	Compares file names based on the current locale.
	AnsiCompareStr (see page 22789)	Compares strings based on the current locale with case sensitivity.
	AnsiCompareText (see page 22790)	Compares strings based on the current locale without case sensitivity.
	AnsiDequotedStr (see page 22790)	Converts a quoted string into an unquoted string
	AnsiExtractQuotedStr (see page 22791)	Converts a quoted string into an unquoted string.
	AnsiLastChar (see page 22791)	Returns a pointer to the last character in a string.
	AnsiLowerCase (see page 22792)	Returns a string that is a copy of the given string converted to lower case.
	AnsiLowerCaseFileName (see page 22792)	Converts a file name to lower case.
	AnsiPos (see page 22793)	Locates the position of a substring.
	AnsiQuotedStr (see page 22793)	Returns the quoted version of a string.
	AnsiSameStr (see page 22794)	Compares strings based on the current locale with case sensitivity.
	AnsiSameText (see page 22794)	Compares strings based on the current locale without case sensitivity.
	AnsiStrComp (see page 22795)	Compares null-terminated character strings.

	AnsiStrIComp (see page 22795)	Compares null terminated character strings case insensitively.
	AnsiStrLastChar (see page 22796)	Returns a pointer to the last character in a null-terminated string.
	AnsiStrLComp (see page 22796)	Compares the first MaxLen bytes of two null-terminated strings, case-sensitively.
	AnsiStrLIComp (see page 22797)	Compares two strings, case-insensitively, up to the first MaxLen bytes.
	AnsiStrLower (see page 22798)	Converts all characters in a null-terminated string to lower case.
	AnsiStrPos (see page 22798)	Returns a pointer to the first occurrence of a substring.
	AnsiStrRScan (see page 22799)	Returns a pointer to the last occurrence of a specified character in a specified string.
	AnsiStrScan (see page 22799)	Returns a pointer to first occurrence of a character in a string.
	AnsiStrUpper (see page 22800)	Converts all characters in a null-terminated string to upper case.
	AnsiUpperCase (see page 22800)	Converts a string to upper case.
	AnsiUpperCaseFileName (see page 22801)	Converts a file name to upper case.
	AppendStr (see page 22801)	Appends a dynamically allocated string to an existing string.
	AssignStr (see page 22802)	Assigns a new dynamically allocated string to a specified pointer.
	Beep (see page 22802)	Emits a message beep.
	BoolToStr (see page 22804)	Converts a Boolean value to a string.
	ByteToCharIndex (see page 22806)	Returns the position of the character that contains a specified byte in a string.
	ByteToCharLen (see page 22807)	Returns the number of characters fully contained in the first MaxLen bytes of a string.
	ByteType (see page 22807)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	CallTerminateProcs (see page 22808)	Calls all of the functions in the termination procedure list.
	ChangeFileExt (see page 22808)	Changes the extension of a file name.
	ChangeFilePath (see page 22810)	Changes the path of a file name.
	CharLength (see page 22810)	Returns number of bytes used by a character.
	CharToByteIndex (see page 22811)	Returns the 1-based index of the first byte of a specified character in a string.
	CharToByteLen (see page 22811)	Returns the length, in bytes, of the first MaxLen characters of a string.
	CheckWin32Version (see page 22812)	Check to see if you are running on a specific level (or higher) of the Windows 32 bit Operating System (see page 22079).
	CompareMem (see page 22812)	Performs a binary comparison of two memory images.
	CompareStr (see page 22813)	Compares two strings case sensitively.
	CompareStr (see page 22813)	Compares two strings case sensitively.
	CompareText (see page 22814)	Compares two strings by ordinal value without case sensitivity.
	CompareText (see page 22814)	Compares two strings by ordinal value without case sensitivity.
	CreateDir (see page 22817)	Creates a new directory.
	CreateGUID (see page 22819)	Creates a globally unique identifier.
	CurrentYear (see page 22819)	Returns the current year.
	CurrToStr (see page 22819)	Formats a Currency value as a string.
	CurrToStr (see page 22819)	Formats a Currency value as a string.
	CurrToStrF (see page 22820)	Converts a Currency value to a string, using a specified format.
	CurrToStrF (see page 22820)	Converts a Currency value to a string, using a specified format.
	Date (see page 22820)	Returns the current date.
	DateTimeToFileDate (see page 22821)	Converts a TDateTime value to an OS timestamp value.
	DateTimeToStr (see page 22822)	Converts a TDateTime value to a string.
	DateTimeToStr (see page 22822)	Converts a TDateTime value to a string.
	DateTimeToString (see page 22823)	Converts a TDateTime value to a string using a specified Format (see page 22897).
	DateTimeToString (see page 22823)	Converts a TDateTime value to a string using a specified Format (see page 22897).
	DateTimeToSystemTime (see page 22825)	Converts a TDateTime value into the Win32 API's system time type.
	DateTimeToTimeStamp (see page 22825)	Converts a TDateTime value into the corresponding TTimeStamp (see page 23009) value.
	DateToStr (see page 22825)	Converts a TDateTime value to a string.
	DateToStr (see page 22825)	Converts a TDateTime value to a string.
	DayOfWeek (see page 22827)	Returns the day of the week for a specified date.
	DecodeDate (see page 22828)	Returns Year, Month, and Day values for a TDateTime value.

	DecodeDateFully (see page 22829)	Returns Year, Month, and Day, and Day-of-Week values for a TDateTime value.
	DecodeTime (see page 22830)	Breaks a TDateTime value into hours, minutes, seconds, and milliseconds.
	DeleteFile (see page 22831)	Deletes a file from disk.
	DirectoryExists (see page 22832)	Determines whether a specified directory exists.
	DiskFree (see page 22833)	Returns the number of free bytes on a specified drive.
	DiskSize (see page 22835)	Returns the size, in bytes, of a specified drive.
	DisposeStr (see page 22836)	Disposes of a string pointer that was allocated using NewStr (see page 22929).
	EncodeDate (see page 22836)	Returns a TDateTime value that represents a specified Year, Month, and Day.
	EncodeTime (see page 22837)	Returns a TDateTime value for a specified Hour, Min, Sec, and MSec.
	ExceptionErrorMessage (see page 22839)	Formats a standard error message.
	ExcludeTrailingBackslash (see page 22839)	Returns a path name without a trailing delimiter.
	ExcludeTrailingPathDelimiter (see page 22840)	Returns a path name without a trailing delimiter.
	ExpandFileName (see page 22840)	Returns the full path name for a relative file name.
	ExpandFileNameCase (see page 22841)	Returns the full path name for a relative file name on a case-sensitive file system.
	ExpandUNCFileName (see page 22842)	Returns the full path of a file name in UNC format, if appropriate.
	ExtractFileDir (see page 22843)	Extracts the drive and directory parts from FileName.
	ExtractFileDrive (see page 22843)	Returns the drive portion of a file name.
	ExtractFileExt (see page 22844)	Returns the extension portions of a file name.
	ExtractFileName (see page 22844)	Extracts the name and extension parts of a file name.
	ExtractFilePath (see page 22846)	Returns the drive and directory portions of a file name.
	ExtractRelativePath (see page 22848)	Returns a relative path name, relative to a specific base directory.
	ExtractShortPathName (see page 22849)	Converts a file name to the short 8.3 form.
	FileAge (see page 22850)	Returns the OS timestamp of a file.
	FileAge (see page 22850)	Returns the OS timestamp of a file.
	FileClose (see page 22852)	Closes a specified file.
	FileCreate (see page 22855)	Creates a new file.
	FileCreate (see page 22855)	Creates a new file.
	FileDateToDateTime (see page 22858)	Converts an OS timestamp value to TDateTime value.
	FileExists (see page 22860)	Tests if a specified file exists.
	FileGetAttr (see page 22863)	Returns the file attributes of FileName.
	FileGetDate (see page 22866)	Returns an OS timestamp for a specified file.
	FileIsReadOnly (see page 22866)	Reports whether a file is read-only.
	FileOpen (see page 22867)	Opens a specified file using a specified access mode.
	FileRead (see page 22870)	Reads a specified number of bytes from a file.
	FileSearch (see page 22872)	Searches a specified directory path for a file.
	FileSeek (see page 22873)	Repositions read/write point.
	FileSeek (see page 22873)	Repositions read/write point.
	FileSetAttr (see page 22875)	Sets the file attributes of a specified file.
	FileSetDate (see page 22878)	Sets the OS time stamp for a specified file.
	FileSetDate (see page 22878)	Sets the OS time stamp for a specified file.
	FileSetReadOnly (see page 22878)	Sets the file permissions to read-only.
	FileWrite (see page 22879)	Writes the contents of a buffer to the current position in a file.
	FinalizePackage (see page 22882)	Finalizes the specified package (.bpl file).
	FindClose (see page 22882)	Releases memory allocated by FindFirst (see page 22884).
	FindCmdLineSwitch (see page 22884)	Determines whether a string was passed as a command line argument to the application.
	FindCmdLineSwitch (see page 22884)	Determines whether a string was passed as a command line argument to the application.
	FindCmdLineSwitch (see page 22884)	Determines whether a string was passed as a command line argument to the application.
	FindFirst (see page 22884)	Searches for the first instance of a file name with a given set of attributes in a specified directory.
	FindNext (see page 22887)	Returns the next entry matching the name and attributes specified in a previous call to FindFirst (see page 22884).
	FloatToCurr (see page 22889)	Converts a floating-point value to a Currency value.
	FloatToDateTime (see page 22889)	Converts a floating-point value to a TDateTime value.

	FloatToDecimal (see page 22890)	Converts a floating-point value to a decimal representation.
	FloatToStr (see page 22890)	Converts a floating point value to a string.
	FloatToStr (see page 22890)	Converts a floating point value to a string.
	FloatToStrF (see page 22891)	Converts a floating point value to a string, using a specified Format (see page 22897), Precision, and Digits.
	FloatToStrF (see page 22891)	Converts a floating point value to a string, using a specified Format (see page 22897), Precision, and Digits.
	FloatToText (see page 22892)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 22897), Precision and Digits.
	FloatToText (see page 22892)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 22897), Precision and Digits.
	FloatToTextFmt (see page 22893)	Converts a floating-point value to an unterminated character string, using a specified format.
	FloatToTextFmt (see page 22893)	Converts a floating-point value to an unterminated character string, using a specified format.
	FmtLoadStr (see page 22893)	Returns formatted output using a resourced format string.
	FmtStr (see page 22894)	Assembles a formatted string using a format string and an array of arguments.
	FmtStr (see page 22894)	Assembles a formatted string using a format string and an array of arguments.
	ForceDirectories (see page 22895)	Creates a new directory, including the creation of parent directories as needed.
	Format (see page 22897)	Returns a formatted string assembled from a format string and an array of arguments.
	Format (see page 22897)	Returns a formatted string assembled from a format string and an array of arguments.
	FormatBuf (see page 22900)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 22900)	Formats the arguments from an array, placing the result in a buffer.
	FormatCurr (see page 22901)	Formats a Currency object.
	FormatCurr (see page 22901)	Formats a Currency object.
	FormatDateTime (see page 22902)	Formats a TDateTime value.
	FormatDateTime (see page 22902)	Formats a TDateTime value.
	FormatFloat (see page 22904)	Formats a floating point value.
	FormatFloat (see page 22904)	Formats a floating point value.
	FreeAndNil (see page 22907)	Frees an object reference and replaces the reference with nil (Delphi) or NULL (C++).
	GDAL (see page 22909)	Get Delphi Access Licence.
	GetCurrentDir (see page 22910)	Returns the name of the current directory.
	GetEnvironmentVariable (see page 22911)	Retrieves an environment variable value.
	GetFileVersion (see page 22912)	Returns file version.
	GetFormatSettings (see page 22912)	Resets the date and number format parameters to initial values.
	GetLocaleChar (see page 22913)	Retrieves the first character of a localisation value.
	GetLocaleFormatSettings (see page 22913)	Populates a TFormatSettings (see page 23001) data structure.
	GetLocaleStr (see page 22913)	Retrieves a localisation value.
	GetModuleName (see page 22914)	Returns the fully qualified name for a module, given its handle.
	GetPackageDescription (see page 22915)	Returns the description resource from a package.
	GetPackageInfo (see page 22915)	Enumerates all the units and required packages for a package.
	GetTime (see page 22916)	Returns the current time.
	GUIDToString (see page 22916)	Converts a class identifier GUID to a string.
	IncAMonth (see page 22917)	Increments date data by one month.
	IncludeTrailingBackslash (see page 22918)	Ensures path name ends with delimiter
	IncludeTrailingPathDelimiter (see page 22918)	Ensures path name ends with delimiter.
	IncMonth (see page 22919)	Returns a date shifted by a specified number of months.
	InitializePackage (see page 22920)	Validates and initializes a given package.
	InitializePackage (see page 22920)	Validates and initializes a given package.
	IntToHex (see page 22920)	Returns the hex representation of an integer.
	IntToHex (see page 22920)	Returns the hex representation of an integer.
	IntToStr (see page 22921)	Converts an integer to a string.
	IntToStr (see page 22921)	Converts an integer to a string.
	IsAssembly (see page 22922)	Determines whether a dll is an Assembly.

	IsDelimiter (see page 22923)	Indicates whether a specified character in a string matches one of a set of delimiters.
	IsEqualGUID (see page 22923)	Indicates whether two TGUID values are the same.
	IsLeapYear (see page 22924)	Indicates whether a specified year is a leap year.
	IsPathDelimiter (see page 22925)	Indicates whether the byte at a specified position in a string is the path delimiter.
	IsValidIdent (see page 22925)	Determines whether a string conforms to Delphi identifier rules.
	Languages (see page 22926)	Lists the locales for which support is available.
	LastDelimiter (see page 22926)	Returns the byte index of the last character that matches any character in a specified set of delimiters.
	LoadPackage (see page 22927)	Loads a specified package.
	LoadPackage (see page 22927)	Loads a specified package.
	LoadStr (see page 22927)	Loads a string resource from the application's executable file.
	LowerCase (see page 22928)	Converts an ASCII string to lowercase.
	LowerCase (see page 22928)	Converts an ASCII string to lowercase.
	MSecsToTimeStamp (see page 22929)	Converts a specified number of milliseconds into a TTimeStamp (see page 23009) value.
	NewStr (see page 22929)	Allocates a string on the heap.
	NextCharIndex (see page 22930)	Returns the byte index of the next character.
	Now (see page 22930)	Returns the current date and time.
	OutOfMemoryError (see page 22931)	Raises an EOutOfMemory (see page 22742) exception.
	QuotedStr (see page 22931)	Returns the quoted version of a string.
	RaiseLastOSError (see page 22932)	Raises an exception for the last occurring OS or system library error.
	RaiseLastOSError (see page 22932)	Raises an exception for the last occurring OS or system library error.
	RaiseLastWin32Error (see page 22932)	Raises an exception for the last occurring Win32 error.
	RCS (see page 22933)	Check for Delphi Client/Server Licence.
	RemoveDir (see page 22933)	Deletes an existing empty directory.
	RenameFile (see page 22934)	Changes a file name.
	ReplaceDate (see page 22936)	Replaces the date portion of a TDateTime value with a specified date.
	ReplaceTime (see page 22937)	Replaces the time portion of a TDateTime value with a specified time.
	RPR (see page 22938)	Check for Delphi Professional Licence.
	SafeLoadLibrary (see page 22938)	Safe and simple library loading.
	SameFileName (see page 22939)	Compares file names based on the current locale.
	SameStr (see page 22939)	Compares two strings by ordinal value with case sensitivity.
	SameStr (see page 22939)	Compares two strings by ordinal value with case sensitivity.
	SameText (see page 22940)	Compares two strings by ordinal value without case sensitivity.
	SameText (see page 22940)	Compares two strings by ordinal value without case sensitivity.
	SetCurrentDir (see page 22940)	Sets the current directory.
	ShowException (see page 22941)	Displays an exception message with its physical address
	Sleep (see page 22941)	Delay program execution for a specified number of milliseconds.
	StrAlloc (see page 22942)	Allocates a buffer for a null-terminated string and returns a pointer to its first character.
	StrBufSize (see page 22942)	Returns the maximum number of characters that can be stored in a buffer allocated by StrAlloc (see page 22942).
	StrByteType (see page 22943)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	StrCat (see page 22943)	Appends a copy of Source to the end of Dest and returns the concatenated string.
	StrCharLength (see page 22944)	Returns the length, in bytes, of a character.
	StrComp (see page 22945)	Compares two strings with case sensitivity.
	StrCopy (see page 22946)	Copies a null-terminated string.
	StrDispose (see page 22947)	Disposes of a string.
	StrECopy (see page 22948)	Copies null-terminated string.
	StrEnd (see page 22949)	Returns a pointer to the end of a null terminated string.
	StrFmt (see page 22950)	Formats entries in an array.
	StrFmt (see page 22950)	Formats entries in an array.
	StrIComp (see page 22951)	Compares two strings without case sensitivity.
	StringReplace (see page 22951)	Returns a string with occurrences of one substring replaced by another substring.

	StringToGUID (see page 22952)	Converts a string to a GUID.
	StrLCat (see page 22952)	Appends up to a specified maximum number of characters to string.
	StrLComp (see page 22953)	Compares up to a specified maximum number of characters in two strings.
	StrLCopy (see page 22953)	Copies up to a specified maximum number of characters from Source to Dest.
	StrLen (see page 22954)	Returns number of characters in a string excluding the null terminator.
	StrLFmt (see page 22954)	Formats a series of arguments from a specified open array into a buffer.
	StrLFmt (see page 22954)	Formats a series of arguments from a specified open array into a buffer.
	StrLIComp (see page 22955)	Compares strings up to a specified maximum number of characters, without case sensitivity.
	StrLower (see page 22956)	Converts a string to lowercase.
	StrMove (see page 22956)	Copies specified number of characters to string.
	StrNew (see page 22957)	Allocates space on and copies a string to the heap; returning a pointer to the string.
	StrNextChar (see page 22958)	Returns pointer to following character.
	StrPas (see page 22958)	Converts null-terminated string to an AnsiString (long string).
	StrPCopy (see page 22958)	Copies an AnsiString (long string) to a null-terminated string.
	StrPLCopy (see page 22959)	Copies characters from an AnsiString (long string) into a null-terminated string.
	StrPos (see page 22959)	Returns a pointer to the first occurrence of STR2 in STR1.
	StrRScan (see page 22960)	Returns a pointer to the last occurrence of a specified character in a string.
	StrScan (see page 22960)	Returns a pointer to first occurrence of a specified character in a string.
	StrToBool (see page 22961)	Converts a string to a Boolean value.
	StrToBoolDef (see page 22961)	Converts a string to a Boolean value, with error default.
	StrToCurr (see page 22962)	Converts a string to a Currency value.
	StrToCurr (see page 22962)	Converts a string to a Currency value.
	StrToCurrDef (see page 22963)	Converts a string to a Currency value, with error default.
	StrToCurrDef (see page 22963)	Converts a string to a Currency value, with error default.
	StrToDate (see page 22963)	Converts a string to a TDateTime value.
	StrToDate (see page 22963)	Converts a string to a TDateTime value.
	StrToDateDef (see page 22965)	Converts a string to a TDateTime value, with error default.
	StrToDateDef (see page 22965)	Converts a string to a TDateTime value, with error default.
	StrToDateTime (see page 22965)	Converts a string to a TDateTime value.
	StrToDateTime (see page 22965)	Converts a string to a TDateTime value.
	StrToDateTimeDef (see page 22966)	Converts a string to a TDateTime value with error default.
	StrToDateTimeDef (see page 22966)	Converts a string to a TDateTime value with error default.
	StrToFloat (see page 22967)	Converts a given string to a floating-point value.
	StrToFloat (see page 22967)	Converts a given string to a floating-point value.
	StrToFloatDef (see page 22969)	Converts a given string to a floating-point value with error default.
	StrToFloatDef (see page 22969)	Converts a given string to a floating-point value with error default.
	StrToInt (see page 22970)	Converts a string that represents an integer (decimal or hex notation) to a number.
	StrToInt64 (see page 22970)	Converts a string that represents an integer (decimal or hex notation) to a number.
	StrToInt64Def (see page 22971)	Converts a string that represents an integer (decimal or hex notation) to a number.
	StrToIntDef (see page 22971)	Converts a string that represents an integer (decimal or hex notation) to a number with error default.
	StrToTime (see page 22972)	Converts a string to a TDateTime value.
	StrToTime (see page 22972)	Converts a string to a TDateTime value.
	StrToTimeDef (see page 22972)	Converts a string to a TDateTime value with error default.
	StrToTimeDef (see page 22972)	Converts a string to a TDateTime value with error default.
	StrUpper (see page 22973)	Returns a string in upper case.
	Supports (see page 22973)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 22973)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 22973)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 22973)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 22973)	Indicates whether a given object or interface supports a specified interface.

	SysErrorMessage (see page 22974)	Converts OS error codes into strings.
	SystemTimeToDateTime (see page 22974)	Converts a system time value into a TDateTime value.
	TextToFloat (see page 22975)	Converts a null-terminated string to a floating-point value.
	TextToFloat (see page 22975)	Converts a null-terminated string to a floating-point value.
	Time (see page 22976)	Returns the current time.
	TimeStampToDateTime (see page 22976)	Converts a TTimeStamp (see page 23009) value into the corresponding TDateTime value.
	TimeStampToMSecs (see page 22977)	Converts a TTimeStamp (see page 23009) value into an absolute number of milliseconds.
	TimeToStr (see page 22977)	Returns a string that represents a TDateTime value.
	TimeToStr (see page 22977)	Returns a string that represents a TDateTime value.
	Trim (see page 22978)	Trims leading and trailing spaces and control characters from a string.
	Trim (see page 22978)	Trims leading and trailing spaces and control characters from a string.
	TrimLeft (see page 22979)	Trims leading spaces and control characters from a string.
	TrimLeft (see page 22979)	Trims leading spaces and control characters from a string.
	TrimRight (see page 22979)	Trims trailing spaces and control characters from a string.
	TrimRight (see page 22979)	Trims trailing spaces and control characters from a string.
	TryEncodeDate (see page 22980)	Creates a TDateTime value that represents a specified Year, Month, and Day.
	TryEncodeTime (see page 22980)	Creates a TDateTime value for a specified Hour, Min, Sec, and MSec.
	TryFloatToCurr (see page 22981)	Converts a floating-point value to a Currency value.
	TryFloatToDateTime (see page 22982)	Converts a floating-point value to a TDateTime value.
	TryStrToBool (see page 22982)	Converts a string to a Boolean value, with a Boolean success code.
	TryStrToCurr (see page 22983)	Converts a string to a Currency value, with a Boolean success code.
	TryStrToCurr (see page 22983)	Converts a string to a Currency value, with a Boolean success code.
	TryStrToDate (see page 22983)	Converts a string to a TDateTime value, with a Boolean success code.
	TryStrToDate (see page 22983)	Converts a string to a TDateTime value, with a Boolean success code.
	TryStrToDateTime (see page 22984)	Converts a string to a TDateTime value with a Boolean success code.
	TryStrToDateTime (see page 22984)	Converts a string to a TDateTime value with a Boolean success code.
	TryStrToFloat (see page 22985)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 22985)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 22985)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 22985)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 22985)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 22985)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToInt (see page 22986)	Converts a string that represents an integer (decimal or hex notation) to a number with Boolean success code.
	TryStrToInt64 (see page 22986)	Converts a string that represents an integer (decimal or hex notation) to a number.
	TryStrToTime (see page 22987)	Converts a string to a TDateTime value with an error default.
	TryStrToTime (see page 22987)	Converts a string to a TDateTime value with an error default.
	UnloadPackage (see page 22987)	Unloads a package.
	UpperCase (see page 22988)	Returns a copy of a string in uppercase.
	UpperCase (see page 22988)	Returns a copy of a string in uppercase.
	WideCompareStr (see page 22989)	Compares Unicode strings based on the current locale with case sensitivity.
	WideCompareText (see page 22990)	Compares Unicode strings based on the current locale without case sensitivity.
	WideFmtStr (see page 22990)	Assembles a formatted Unicode string using a format string and an array of arguments.
	WideFmtStr (see page 22990)	Assembles a formatted Unicode string using a format string and an array of arguments.
	WideFormat (see page 22991)	Returns a formatted Unicode string assembled from a format string and an array of arguments.
	WideFormat (see page 22991)	Returns a formatted Unicode string assembled from a format string and an array of arguments.
	WideFormatBuf (see page 22991)	Formats the arguments from an array, placing the result in a buffer.
	WideFormatBuf (see page 22991)	Formats the arguments from an array, placing the result in a buffer.
	WideLowerCase (see page 22992)	Returns Unicode string converted to lower case.
	WideSameStr (see page 22992)	Compares Unicode strings based on the current locale with case sensitivity.

	WideSameText (see page 22993)	Compares Unicode strings based on the current locale without case sensitivity.
	WideUpperCase (see page 22993)	Returns Unicode string converted to upper case.
	Win32Check (see page 22994)	Checks the return value of a Windows API call and raises an appropriate exception when it indicates failure.
	WrapText (see page 22994)	Splits a string into multiple lines as its length approaches a specified size.
	WrapText (see page 22994)	Splits a string into multiple lines as its length approaches a specified size.

Interfaces

	Name	Description
	IReadWriteSync (see page 22778)	This is class SysUtils.IReadWriteSync.

Structs, Records, Enums

	Name	Description
	Int64Rec (see page 22996)	Int64Rec declares a utility record to provide access to the bytes of an Int64 value.
	LongRec (see page 22997)	LongRec declares a utility record to provide access to the bytes of a LongWord value.
	TExceptionRecord (see page 22997)	TExceptionRecord holds information about an exception in an application.
	TFilenameCaseMatch (see page 22998)	TFilenameCaseMatch indicates how close a match was found to a specified file name.
	TFloatFormat (see page 22999)	TFloatFormat is an enumerated list of formatting codes for float functions.
	TFloatRec (see page 23000)	TFloatRec declares a utility record that provides access to the attributes of a floating point number.
	TFloatValue (see page 23001)	This is record SysUtils.TFloatValue.
	TFormatSettings (see page 23001)	TFormatSettings defines a thread-safe string formatting context.
	TLangRec (see page 23002)	TLangRec declares a record used to describe a Windows locale.
	TLocaleOptions (see page 23003)	TLocaleOptions defines a choice of dependent and independent locale options.
	TMbcsByteType (see page 23004)	TMbcsByteType represents the use of a single byte in a string that uses a multi-byte character set (MBCS).
	TNameType (see page 23005)	This is record SysUtils.TNameType.
	TSearchRec (see page 23005)	TSearchRec defines file information searched for by FindFirst (see page 22884) or FindNext (see page 22887).
	TSysLocale (see page 23008)	TSysLocale represents locale information.
	TThreadInfo (see page 23009)	This is record SysUtils.TThreadInfo.
	TTimeStamp (see page 23009)	TTimeStamp represents time and date values.
	WordRec (see page 23010)	WordRec declares a utility record to store high and low order bytes of a variable.

Types

Name	Description
ExceptClass (see page 23012)	ExceptClass declares an Exception (see page 22750) class.
PByteArray (see page 23013)	PByteArray declares a pointer to an array of Bytes.
PDayTable (see page 23013)	PDayTable is a pointer to a TDayTable (see page 23015) type.
PExceptionRecord (see page 23014)	PExceptionRecord is a pointer to a TExceptionRecord (see page 22997) type.
PThreadInfo (see page 23014)	This is type SysUtils.PThreadInfo.
PWordArray (see page 23014)	PWordArray is a pointer to a TWordArray (see page 23020) type.
TByteArray (see page 23015)	TByteArray declares an array of Bytes.
TBytes (see page 23015)	TBytes declares an array of Bytes.
TDayTable (see page 23015)	TDayTable is used to hold the number of days of each month of a year.
TFileName (see page 23016)	TFileName is a string that represents a file name.
TIntegerSet (see page 23016)	TIntegerSet is used to treat an integer as a set of bits.
TMREWSync (see page 23017)	TMREWSync is a short name equivalent to TMultiReadExclusiveWriteSynchronizer (see page 22766).
TPackageInfoProc (see page 23017)	TPackageInfoProc is a call-back procedure invoked by GetPackageInfo (see page 22915).
TProcedure (see page 23018)	TProcedure declares a procedure.
TReplaceFlags (see page 23018)	TReplaceFlags indicates how to perform a search-and-replace operation
TSysCharSet (see page 23019)	TSysCharSet is a set containing special characters that can be used to parse a string. The elements of the set vary, depending on the type of parsing performed.

TTerminateProc (see page 23019)	TValidatePackageProc (see page 23019) is a call-back function invoked by LoadPackage (see page 22927).
TValidatePackageProc (see page 23019)	This is type SysUtils.TValidatePackageProc.
TWordArray (see page 23020)	TWordArray declares an array of Words.

Variables

Name	Description
CurrencyDecimals (see page 23022)	CurrencyDecimals is the number of digits to the right of the decimal point in a currency amount.
CurrencyFormat (see page 23022)	CurrencyFormat defines the currency symbol placement and separation used in floating-point–to–decimal conversions. Possible values are: 0 = $1 \| 1 = 1$ | 2 = $ 1 | 3 = 1 $
CurrencyString (see page 23023)	CurrencyString defines the currency symbol (or characters) used in floating-point to decimal conversions.
DateSeparator (see page 23023)	DateSeparator is the character used to separate the year, month, and day parts of a date value.
DecimalSeparator (see page 23023)	DecimalSeparator is the character used to separate the integer part from the fractional part of a number. This variable must have a value other than #0.
EmptyStr (see page 23024)	An empty string constant.
EmptyWideStr (see page 23024)	An empty string version of the WideString constant.
EraNames (see page 23024)	Defines the names of calendar eras for East Asian locales.
EraYearOffsets (see page 23025)	Controls (see page 5711) the DateTimeToStr (see page 22822) function (for East Asian locales).
FalseBoolStrs (see page 23025)	Lists strings that can represent the boolean value false.
GetDiskFreeSpaceEx (see page 23026)	This is variable SysUtils.GetDiskFreeSpaceEx.
HexDisplayPrefix (see page 23026)	HexDisplayPrefix contains the prefix to display on hexadecimal values - `$` for Pascal (Delphi) syntax, `0x` for C++ syntax.
LeadBytes (see page 23026)	Lists all values that can be the first byte of a double byte character on the local system.
ListSeparator (see page 23027)	The character used to separate items in a list.
LongDateFormat (see page 23027)	LongDateFormat is the format string used to convert a date value to a long string suitable for display but not for editing.
LongDayNames (see page 23027)	LongDayNames is the array of strings containing long day names. The dddd format specifier in a format string passed to FormatDateTime (see page 22902) causes a long day name to be substituted.
LongMonthNames (see page 23028)	LongMonthNames is the array of strings containing long month names. The mmmm format specifier in a format string passed to FormatDateTime (see page 22902) causes a long month name to be substituted.
LongTimeFormat (see page 23028)	LongTimeFormat is the format string used to convert a time value to a string with hours, minutes, and seconds.
NegCurrFormat (see page 23029)	NegCurrFormat defines the currency format used in floating-point to decimal conversions of negative numbers.
NullStr (see page 23029)	A pointer to an empty string.
NullWideStr (see page 23030)	A pointer to an empty wide string.
ShortDateFormat (see page 23030)	ShortDateFormat is the format string used to convert a date value to a short string suitable for editing.
ShortDayNames (see page 23031)	ShortDayNames is the array of strings containing short day names. The ddd format specifier in a format string passed to FormatDateTime (see page 22902) causes a short day name to be substituted.
ShortMonthNames (see page 23031)	ShortMonthNames is the array of strings containing short month names. The mmm format specifier in a format string passed to FormatDateTime (see page 22902) causes a short month name to be substituted.
ShortTimeFormat (see page 23031)	ShortTimeFormat is the format string used to convert a time value to a short string with only hours and minutes.
SysLocale (see page 23032)	SysLocale holds locale information.
ThousandSeparator (see page 23032)	ThousandSeparator is the character used to separate thousands in numbers with more than three digits to the left of the decimal separator.
TimeAMString (see page 23033)	TimeAMString is the suffix string used for time values between 00:00 and 11:59 in 12-hour clock format.
TimePMString (see page 23033)	TimePMString is the suffix string used for time values between 12:00 and 23:59 in 12-hour clock format.

TimeSeparator (see page 23033)	TimeSeparator is the character used to separate the hour, minute, and second parts of a time value.
TrueBoolStrs (see page 23034)	Lists strings that can represent the boolean value false.
TwoDigitYearCenturyWindow (see page 23034)	Determines what century is added to two-digit years when string dates are converted to numeric dates.
Win32BuildNumber (see page 23035)	Specifies which the build level of the Windows Operating System (see page 22079) operating.
Win32CSDVersion (see page 23035)	Specifies the Win32 latest installed service pack.
Win32MajorVersion (see page 23036)	Specifies which major Windows Operating System (see page 22079) version is operating.
Win32MinorVersion (see page 23036)	Specifies which minor Windows Operating System (see page 22079) version is operating.
Win32Platform (see page 23037)	Specifies the Windows Operating System (see page 22079) platform identifier.

159.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EAbort (see page 22722)	EAbort is the exception class for errors that should not display an error message dialog box.
	EAbstractError (see page 22722)	EAbstractError is the exception class for attempts to call an abstract method.
	EAccessViolation (see page 22723)	EAccessViolation is the exception class for invalid memory access errors.
	EAssertionFailed (see page 22723)	EAssertionFailed is the exception class for Delphi assertion failure.
	EControlC (see page 22724)	EControlC is the exception class for user interrupts.
	EConvertError (see page 22724)	EConvertError is the exception class for string and object conversion errors.
	EDivByZero (see page 22726)	EDivByZero is the exception class for integer divide-by-zero errors.
	EExternal (see page 22727)	EExternal is the exception class that captures external error information.
	EExternalException (see page 22729)	EExternalException is the exception class for invalid exception codes.
	EHeapException (see page 22730)	EHeapException is the exception class for errors related to heap-allocated memory.
	EInOutError (see page 22732)	EInOutError is the exception class for file input/output errors.
	EIntError (see page 22735)	EIntError is the base class for integer math error exceptions.
	EIntfCastError (see page 22736)	EIntfCastError is the exception class for interface casting errors.
	EIntOverflow (see page 22736)	EIntOverflow is the exception class for integer calculations whose results are too large to fit in the allocated register.
	EInvalidCast (see page 22737)	EInvalidCast is an exception class for typecasting errors.
	EInvalidContainer (see page 22738)	This is class SysUtils.EInvalidContainer.
	EInvalidInsert (see page 22738)	This is class SysUtils.EInvalidInsert.
	EInvalidOp (see page 22738)	EInvalidOp is the exception class for undefined floating-point operations.
	EInvalidPointer (see page 22739)	EInvalidPointer is the exception class for invalid pointer operations.
	EMathError (see page 22739)	EMathError is the base exception class for floating-point math errors.
	EOSError (see page 22740)	EOSError is the exception class for operating system errors.
	EOutOfMemory (see page 22742)	EOutOfMemory is the exception class for unsuccessful attempts to allocate memory.
	EOverflow (see page 22743)	EOverflow is the exception class for floating-point register overflows.
	EPackageError (see page 22744)	EPackageError is the exception class for package-related errors.
	EPrivilege (see page 22744)	EPrivilege is the exception class for processor privilege violations.
	EPropReadOnly (see page 22745)	EPropReadOnly is the exception class for invalid attempts to write to a property using OLE automation.
	EPropWriteOnly (see page 22745)	EPropWriteOnly is the exception class for invalid attempts to read a property using OLE automation.
	ERangeError (see page 22745)	ERangeError is the exception for Delphi range errors.
	ESafecallException (see page 22747)	ESafecallException is the exception for problems with the safecall calling convention.
	EStackOverflow (see page 22747)	EStackOverflow is the exception class for stack overflows.
	EUnderflow (see page 22748)	EUnderflow is the exception class for values that are too small to be represented with a floating-point variable.
	EVariantError (see page 22748)	EVariantError is the exception class for errors involving Variant data types.
	EWin32Error (see page 22749)	EWin32Error is the exception class for Windows errors.
	Exception (see page 22750)	Exception is the base class for all runtime exceptions.
	EZeroDivide (see page 22761)	EZeroDivide is the exception class for floating-point divide-by-zero errors.
	TLanguages (see page 22762)	TLanguages lists all available Windows locales.
	TMultiReadExclusiveWriteSynchronizer (see page 22766)	TMultiReadExclusiveWriteSynchronizer protects memory in a multi-threaded application.
	TSimpleRWSync (see page 22772)	This is class SysUtils.TSimpleRWSync.

	TThreadLocalCounter (see page 22775)	This is class SysUtils.TThreadLocalCounter.

159.1.1 EAbort Class

EAbort is the exception class for errors that should not display an error message dialog box.

Class Hierarchy

Pascal

```
EAbort = class(Exception);
```

C++

```
class EAbort : public Exception;
```

File

SysUtils

Description

Use EAbort to raise an exception without displaying an error message in a dialog box. If applications do not trap such "silent" exceptions, the EAbort exception is passed to the HandleException method of the Application object.

The Abort (see page 22786) procedure provides a simple, standard way to raise EAbort.

See Also

Abort (see page 22786)

TApplication (see page 13660)

Application

159.1.2 EAbstractError Class

EAbstractError is the exception class for attempts to call an abstract method.

Class Hierarchy

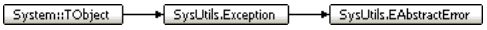

Pascal

```
EAbstractError = class(Exception);
```

C++

```
class EAbstractError : public Exception;
```

File

SysUtils

Description

EAbstractError is raised when an application tries to call a Delphi abstract method. It is also raised at design time when a component with Delphi abstract method is placed on a form.

Delphi abstract methods are declared with the abstract directive and must be overridden in descendant classes before an instance object can be created.

C++ code can catch this error, but never throws it.

159.1.3 EAccessViolation Class

EAccessViolation is the exception class for invalid memory access errors.

Class Hierarchy

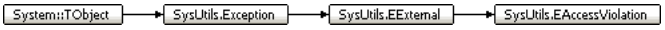

Pascal

```
EAccessViolation = class(EExternal);
```

C++

```
class EAccessViolation : public EExternal;
```

File

SysUtils

Description

EAccessViolation is raised when an application

Dereferences a nil (Delphi) or NULL (C++) pointer.

Writes to memory reserved for executable code.

Attempts to access a memory address for which there is no virtual memory allocated to the application.

Run-time exception information is saved in fields provided by EExternal (see page 22727).

Note: Applications should not raise EAccessViolation directly, but should instead rely on the runtime to raise this exception.

See Also

EExternal (see page 22727)

EOutOfMemory (see page 22742)

EOutOfResources (see page 2702)

159.1.4 EAssertionFailed Class

EAssertionFailed is the exception class for Delphi assertion failure.

Class Hierarchy

Pascal

```
EAssertionFailed = class(Exception);
```

C++

```
class EAssertionFailed : public Exception;
```

File

SysUtils

Description

EAssertionFailed is raised when an assertion fails in Delphi code.

Note: To raise this exception, use the Assert procedure. The exception is raised only in code compiled with the $ASSERTIONS ON directive.

See Also

Assert

159.1.5 EControlC Class

EControlC is the exception class for user interrupts.

Class Hierarchy

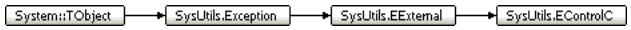

Pascal

```
EControlC = class(EExternal);
```

C++

```
class EControlC : public EExternal;
```

File

SysUtils

Description

EControlC is raised when the application is interrupted by the user.

Note: On Windows, a user interrupt can only occur in console applications, when the user presses Ctrl+C.

Note: On Linux, EControlC is raised when the program receives a TERM (15) signal. The TERM signal can be generated by the user typing the interrupt character (normally Ctrl+C) or by another program using the signal system call.

Run-time exception information is saved in fields provided by EExternal (see page 22727).

See Also

EExternal (see page 22727)

159.1.6 EConvertError Class

EConvertError is the exception class for string and object conversion errors.

Class Hierarchy

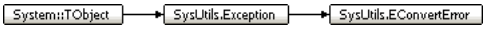

Pascal

```
EConvertError = class(Exception);
```

C++

```
class EConvertError : public Exception;
```

File

SysUtils

Description

EConvertError is raised when

An application makes an unsuccessful attempt to convert an integer, float, date, or time to a string, or to convert a string to one of these other types.

An application passes an invalid argument to a formatting routine.

An application attempts to assign one type of component derived from TPersistent to another component derived from TPersistent when such an assignment is not possible. For example, EConvertError is raised by the attempted assignment of a TButton control to a TEdit control.

In the examples below, an EConvertError exception is raised on the attempt to convert a String to a TDateTime and the date in the String is invalid. The type of exception and the error message are displayed.

//Delphi example

```
const
  LF = #10;
var
  TempDate: TDateTime;
begin
try
    TempDate := StrToDateTime('99/99/1998');
except
on E: EConvertError do
      ShowMessage(E.ClassName + LF + E.Message);
```

end;

//C++ example

```
  TDateTime TempDate;
try
  {
    TempDate = StrToDateTime("99/99/1998");
  }
catch (EConvertError &E)
  {
    ShowMessage(AnsiString(E.ClassName()) + "\n" + AnsiString(E.Message));
  }
```

See Also

DateTimeToStr (see page 22822)

DateTimeToString (see page 22823)

DateToStr (see page 22825)

EncodeDate (see page 22836)

EncodeTime (see page 22837)

FmtLoadStr (see page 22893)

FmtStr (see page 22894)

Format (see page 22897)

FormatBuf (see page 22900)

FormatCurr (see page 22901)

FormatDateTime (see page 22902)

FormatFloat (see page 22904)

IntToStr (see page 22921)

StrFmt (see page 22950)

StrLFmt (see page 22954)

StrToCurr (see page 22962)

StrToDate (see page 22963)

StrToDateTime (see page 22965)

StrToFloat (see page 22967)

StrToInt (see page 22970)

StrToTime (see page 22972)

TimeToStr (see page 22977)

TPersistent (see page 2928)

159.1.7 EDivByZero Class

EDivByZero is the exception class for integer divide-by-zero errors.

Class Hierarchy

Pascal

```
EDivByZero = class(EIntError);
```

C++

```
class EDivByZero : public EIntError;
```

File

SysUtils

Description

EDivByZero is raised when an application tries to divide an integer by zero.

Run-time exception information is saved in fields provided by EExternal (see page 22727).

Note: Floating-point divide-by-zero errors raise the EZeroDivide (see page 22761) exception.

See Also

EExternal (see page 22727)

EIntOverflow (see page 22736)

ERangeError (see page 22745)

EZeroDivide (see page 22761)

159.1.8 EExternal Class

EExternal is the exception class that captures external error information.

Class Hierarchy

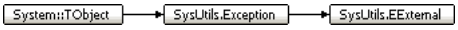

Pascal

```
EExternal = class(Exception);
```

C++

```
class EExternal : public Exception;
```

File

SysUtils

Description

EExternal is the ancestor of all exception classes that represent external run time errors.

Note: Under Windows, EExternal classes represent Win32 exception codes. The ExceptionRecord (see page 22729) member points to a Win32 exception record:

//Delphi syntax

```
type
PExceptionRecord = ^TExceptionRecord;
TExceptionRecord = record
  ExceptionCode: Cardinal;
  ExceptionFlags: Cardinal;
  ExceptionRecord: PExceptionRecord;
  ExceptionAddress: Pointer;
  NumberParameters: Cardinal;
  ExceptionInformation: array[0..14] of Cardinal;
end;
```

ExceptionRecord (see page 22729): PExceptionRecord (see page 23014);

//C++ syntax

```
struct TExceptionRecord;
typedef TExceptionRecord *PExceptionRecord;
struct TExceptionRecord
{
unsigned ExceptionCode;
unsigned ExceptionFlags;
  TExceptionRecord *ExceptionRecord;
void *ExceptionAddress;
unsigned NumberParameters;
```

```
unsigned ExceptionInformation[15];
} ;
```

Note: Under Linux, EExternal classes represent various runtime errors. EExternal has three fields with exception information:

ExceptionAddress, which contains the address of the instruction that caused the error;

AccessAddress, the address that was being accessed when the error occurred;

SignalNumber, for exceptions raised by Linux signals.

See Also

EAccessViolation (see page 22723)

EIntError (see page 22735)

EMathError (see page 22739)

EPrivilege (see page 22744)

EStackOverflow (see page 22747)

159.1.8.1 EExternal Members

The following tables list the members exposed by EExternal.

Public Fields

	Name	Description
	ExceptionRecord (see page 22729)	This is ExceptionRecord, a member of class EExternal.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 22752)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.

159.1.8.2 EExternal Fields

The fields of the EExternal class are listed here.

Public Fields

	Name	Description
	ExceptionRecord (see page 22729)	This is ExceptionRecord, a member of class EExternal.

159.1.8.2.1 EExternal.ExceptionRecord Field

Pascal

```
ExceptionRecord: PExceptionRecord platform;
```

C++

```
PExceptionRecord platform ExceptionRecord;
```

Description

This is ExceptionRecord, a member of class EExternal.

159.1.9 EExternalException Class

EExternalException is the exception class for invalid exception codes.

Class Hierarchy

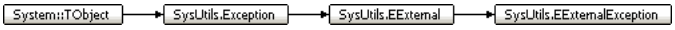

Pascal

```
EExternalException = class(EExternal);
```

C++

```
class EExternalException : public EExternal;
```

File

SysUtils

Description

EExternalException is raised when the run time library detects an unrecognized exception code. The run time library recognizes the following exception codes:

STATUS_INTEGER_DIVIDE_BY_ZERO

STATUS_ARRAY_BOUNDS_EXCEEDED

STATUS_FLOAT_OVERFLOW

STATUS_FLOAT_INEXACT_RESULT

STATUS_FLOAT_INVALID_OPERATION

STATUS_FLOAT_STACK_CHECK

STATUS_FLOAT_DIVIDE_BY_ZERO

STATUS_INTEGER_OVERFLOW

STATUS_FLOAT_UNDERFLOW

STATUS_FLOAT_DENORMAL_OPERAND

STATUS_ACCESS_VIOLATION

STATUS_PRIVILEGED_INSTRUCTION

STATUS_CONTROL_C_EXIT

STATUS_STACK_OVERFLOW

Any exception code that is not on the list above raises an EExternalException.

Note: As a descendant of EExternal (see page 22727), EExternalException saves the exception record in its ExceptionRecord (see page 22729) data structure. An exception record is only generated under Windows.

See Also

EExternal (see page 22727)

159.1.10 EHeapException Class

EHeapException is the exception class for errors related to heap-allocated memory.

Class Hierarchy

Pascal

```
EHeapException = class(Exception);
```

C++

```
class EHeapException : public Exception;
```

File

SysUtils

Description

EHeapException's descendants—EOutOfMemory (see page 22742) and EInvalidPointer (see page 22739)—are used to handle failed allocations of dynamic memory and invalid pointer operations.

Note: Memory for these exceptions is pre-allocated whenever an application starts and remains allocated as long as the application is running. Never raise EHeapException or its descendants directly.

See Also

EOutOfMemory (see page 22742)

EInvalidPointer (see page 22739)

159.1.10.1 EHeapException Members

The following tables list the members exposed by EHeapException.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 22752)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22732)	Deallocates the memory occupied by an EHeapException (see page 22730) instance.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.

159.1.10.2 EHeapException Methods

The methods of the EHeapException class are listed here.

Public Methods

	Name	Description
V	FreeInstance (see page 22732)	Deallocates the memory occupied by an EHeapException (see page 22730) instance.

159.1.10.2.1 EHeapException.FreeInstance Method

Deallocates the memory occupied by an EHeapException (see page 22730) instance.

Pascal

```
procedure FreeInstance; override;
```

C++

```
virtual __fastcall FreeInstance();
```

Description

FreeInstance is called internally when an application shuts down. Do not call this method directly or attempt to create or destroy instances of EHeapException (see page 22730).

159.1.11 EInOutError Class

EInOutError is the exception class for file input/output errors.

Class Hierarchy

Pascal

```
EInOutError = class(Exception);
```

C++

```
class EInOutError : public Exception;
```

File

SysUtils

Description

EInOutError is raised when an file input/output error occurs, provided I/O checking is enabled.

Note: In Delphi code, use the $I+ directive to enable I/O checking. If an I/O error occurs while this directive is disabled, the application must call IOResult to clear the error.

Note: In C++ programs, I/O checking is a project option.

The error code is available in the ErrorCode (see page 22735) class member. Error codes are come in ranges: 0-99 (native OS errors)

Error codes in the range 0-99 represent OS error conditions, which are different for Windows and Linux. Refer to the OS documentation for complete error summaries. The SysErrorMessage (see page 22974) function returns descriptive text for OS errors.

Here are some common OS I/O errors, arranged by rough equivalents in Linux and Windows:

LinuxWindows

Linux		**Windows**	
Error Code	Description	Error Code	Description
2	No such file or directory	2	File not found
3	Path not found		
5	I/O Error		
13	Permission denied	5	Access denied
20	Not a directory		
21	Is a directory		
32	Sharing violation		
Error codes in the range 100 to 149 represent error conditions raised by CLX. Here are the CLX I/O error codes:			

Error code	**Meaning**
100	End of File
101	Disk Full
102	File variable not assigned
103	File not open
104	File not open for input
105	File not open for output
106	Error in formatted input
107	File already open

See Also

IOResult

EStreamError (see page 2704)

SysErrorMessage (see page 22974)

159.1.11.1 EInOutError Members

The following tables list the members exposed by EInOutError.

Public Fields

	Name	Description
	ErrorCode (see page 22735)	This is ErrorCode, a member of class EInOutError.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 22752)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.

159.1.11.2 EInOutError Fields

The fields of the EInOutError class are listed here.

Public Fields

	Name	Description
	ErrorCode (see page 22735)	This is ErrorCode, a member of class EInOutError.

159.1.11.2.1 EInOutError.ErrorCode Field

Pascal

```
ErrorCode: Integer;
```

C++

```
int ErrorCode;
```

Description

This is ErrorCode, a member of class EInOutError.

159.1.12 EIntError Class

EIntError is the base class for integer math error exceptions.

Class Hierarchy

System::TObject → SysUtils.Exception → SysUtils.EExternal → SysUtils.EIntError

Pascal

```
EIntError = class(EExternal);
```

C++

```
class EIntError : public EExternal;
```

File

SysUtils

Description

EIntError is a generic integer math exception. Although it is never raised in the runtime library, it is the base from which other integer math exceptions descend. These are the integer math exceptions:

Exception (see page 22750)	Meaning
EDivByZero (see page 22726)	An attempt was made to divide by zero
ERangeError (see page 22745)	Number or expression out of range
EIntOverflow (see page 22736)	Integer operation overflowed

Run-time exception information is saved in fields provided by EExternal (see page 22727).

See Also

EDivByZero (see page 22726)

EExternal (see page 22727)

ERangeError (see page 22745)

EIntOverflow (see page 22736)

159.1.13 EIntfCastError Class

EIntfCastError is the exception class for interface casting errors.

Class Hierarchy

Pascal

```
EIntfCastError = class(Exception);
```

C++

```
class EIntfCastError : public Exception;
```

File

SysUtils

Description

EIntfCastError is raised when an illegal cast is attempted on an interface.

See Also

EInvalidCast (see page 22737)

159.1.14 EIntOverflow Class

EIntOverflow is the exception class for integer calculations whose results are too large to fit in the allocated register.

Class Hierarchy

Pascal

```
EIntOverflow = class(EIntError);
```

C++

```
class EIntOverflow : public EIntError;
```

File

SysUtils

Description

EIntOverflow is a occurs when data is lost because an integer result is too large to retain.

Note: In Delphi Code, EIntOverflow is raised only if overflow checking is turned on. To turn on overflow checking, include the $Q+ directive in project source code, or select Project|Options, choose the Compiler tab, and check the Overflow-checking option in the dialog box.

Run-time exception information is saved in fields provided by EExternal (see page 22727).

Note: Floating-point overflows raise the EOverflow (see page 22743) exception.

See Also

EExternal (see page 22727)

ERangeError (see page 22745)

EDivByZero (see page 22726)

EOverflow (see page 22743)

159.1.15 EInvalidCast Class

EInvalidCast is an exception class for typecasting errors.

Class Hierarchy

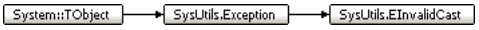

Pascal

```
EInvalidCast = class(Exception);
```

C++

```
class EInvalidCast : public Exception;
```

File

SysUtils

Description

EInvalidCast is raised when an illegal typecast is attempted.

See Also

EIntfCastError (see page 22736)

Bad

159.1.16 EInvalidContainer Class

Class Hierarchy

Pascal

```
EInvalidContainer = class(Exception);
```

C++

```
class EInvalidContainer : public Exception;
```

File

SysUtils

Description

This is class SysUtils.EInvalidContainer.

159.1.17 EInvalidInsert Class

Class Hierarchy

Pascal

```
EInvalidInsert = class(Exception);
```

C++

```
class EInvalidInsert : public Exception;
```

File

SysUtils

Description

This is class SysUtils.EInvalidInsert.

159.1.18 EInvalidOp Class

EInvalidOp is the exception class for undefined floating-point operations.

Class Hierarchy

Pascal

```
EInvalidOp = class(EMathError);
```

C++

```
class EInvalidOp : public EMathError;
```

File

SysUtils

Description

EInvalidOp is raised when the CPU encounters an undefined instruction, invalid operation, or floating-point stack overflow.

Run-time exception information is saved in fields provided by EExternal (see page 22727).

Note: Do not confuse EInvalidOp with EInvalidOperation, the exception class for invalid component operations.

See Also

EExternal (see page 22727)

EInvalidOperation (see page 2700)

EInvalidArgument (see page 15927)

EOverflow (see page 22743)

EUnderflow (see page 22748)

EZeroDivide (see page 22761)

159.1.19 EInvalidPointer Class

EInvalidPointer is the exception class for invalid pointer operations.

Class Hierarchy

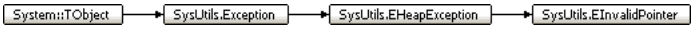

Pascal

```
EInvalidPointer = class(EHeapException);
```

C++

```
class EInvalidPointer : public EHeapException;
```

File

SysUtils

Description

EInvalidPointer is raised when an application attempts an invalid pointer operation. For example, it can occur if an application tries to dispose of the same pointer twice, or refers to a pointer which has already been disposed of.

Note: Never raise an EInvalidPointer exception directly. EInvalidPointer is raised internally by the memory manager.

159.1.20 EMathError Class

EMathError is the base exception class for floating-point math errors.

Class Hierarchy

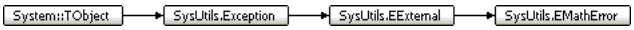

Pascal

```
EMathError = class(EExternal);
```

C++

```
class EMathError : public EExternal;
```

File

SysUtils

Description

EMathError is the base class for floating-point math exception classes. EMathError itself is never raised.

The following exceptions descend from EMathError:

Exception (see page 22750)	Meaning
EInvalidArgument	Parameter out of range
EInvalidOp (see page 22738)	Processor encountered an undefined instruction
EOverflow (see page 22743)	Floating-point operation produced result too large to store
EUnderflow (see page 22748)	Floating-point operation produced result with no precision
EZeroDivide (see page 22761)	Attempt to divide by zero

Run-time exception information is saved in fields provided by EExternal (see page 22727).

See Also

EExternal (see page 22727)

EInvalidArgument (see page 15927)

EInvalidOp (see page 22738)

EOverflow (see page 22743)

EUnderflow (see page 22748)

EZeroDivide (see page 22761)

159.1.21 EOSError Class

EOSError is the exception class for operating system errors.

Class Hierarchy

Pascal

```
EOSError = class(Exception);
```

C++

```
class EOSError : public Exception;
```

File

SysUtils

Description

EOSError is raised by the RaiseLastOSError (see page 22932) procedure if the operating system returns an error code. The exception dialog contains the error number and the message associated with the error.

Note: EOSError replaces EWin32Error (see page 22749), which is now deprecated.

See Also

RaiseLastOSError (see page 22932)

159.1.21.1 EOSError Members

The following tables list the members exposed by EOSError.

Public Fields

	Name	Description
	ErrorCode (see page 22742)	This is ErrorCode, a member of class EOSError.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 22752)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.

159.1.21.2 EOSError Fields

The fields of the EOSError class are listed here.

Public Fields

	Name	Description
	ErrorCode (see page 22742)	This is ErrorCode, a member of class EOSError.

159.1.21.2.1 EOSError.ErrorCode Field

Pascal

```
ErrorCode: DWORD;
```

C++

```
DWORD ErrorCode;
```

Description

This is ErrorCode, a member of class EOSError.

159.1.22 EOutOfMemory Class

EOutOfMemory is the exception class for unsuccessful attempts to allocate memory.

Class Hierarchy

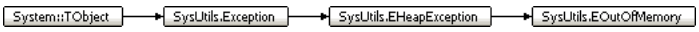

Pascal

```
EOutOfMemory = class(EHeapException);
```

C++

```
class EOutOfMemory : public EHeapException;
```

File

SysUtils

Description

EOutOfMemory occurs when an application attempts to allocate dynamic memory, but there is not enough free memory in the system to meet the request.

Memory for the EOutOfMemory exception is pre-allocated whenever an application starts and remains allocated as long as the application is running.

Note: Never raise EOutOfMemory directly. Instead, call the global OutOfMemoryError (see page 22931) procedure.

See Also

EOutOfResources (see page 2702)

OutOfMemoryError (see page 22931)

EHeapException (see page 22730)

159.1.23 EOverflow Class

EOverflow is the exception class for floating-point register overflows.

Class Hierarchy

Pascal

```
EOverflow = class(EMathError);
```

C++

```
class EOverflow : public EMathError;
```

File

SysUtils

Description

EOverflow is raised when a calculated result is too large to fit in the floating-point register allocated for it and data is therefore lost.

Run-time exception information is saved in fields provided by EExternal (see page 22727).

Note: Integer register overflows raise the EIntOverflow (see page 22736) exception.

See Also

EExternal (see page 22727)

EInvalidArgument (see page 15927)

EInvalidOp (see page 22738)

EUnderflow (see page 22748)

EZeroDivide (see page 22761)

EIntOverflow (see page 22736)

159.1.24 EPackageError Class

EPackageError is the exception class for package-related errors.

Class Hierarchy

Pascal

```
EPackageError = class(Exception);
```

C++

```
class EPackageError : public Exception;
```

File

SysUtils

Description

EPackageError is raised at design time only, when an error occurs loading or utilizing a package.

159.1.25 EPrivilege Class

EPrivilege is the exception class for processor privilege violations.

Class Hierarchy

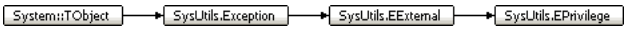

Pascal

```
EPrivilege = class(EExternal);
```

C++

```
class EPrivilege : public EExternal;
```

File

SysUtils

Description

EPrivilege is raised when an application tries to execute a processor instruction that is invalid for the current processor privilege level.

Run-time exception information is saved in fields provided by EExternal (see page 22727).

See Also

EExternal (see page 22727)

159.1.26 EPropReadOnly Class

EPropReadOnly is the exception class for invalid attempts to write to a property using OLE automation.

Class Hierarchy

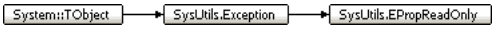

Pascal

```
EPropReadOnly = class(Exception);
```

C++

```
class EPropReadOnly : public Exception;
```

File

SysUtils

Description

EPropReadOnly is raised when an application tries to write to a read-only property using OLE automation.

159.1.27 EPropWriteOnly Class

EPropWriteOnly is the exception class for invalid attempts to read a property using OLE automation.

Class Hierarchy

Pascal

```
EPropWriteOnly = class(Exception);
```

C++

```
class EPropWriteOnly : public Exception;
```

File

SysUtils

Description

EPropWriteOnly is raised when an application tries to read a write-only property using OLE automation.

159.1.28 ERangeError Class

ERangeError is the exception for Delphi range errors.

Class Hierarchy

Pascal

```
ERangeError = class(EIntError);
```

C++

```
class ERangeError : public EIntError;
```

File

SysUtils

Description

ERangeError occurs in Delphi code when range checking is enabled and an ordinal values goes outside its declared range.

Note: Range checking is not a C++ feature. However, ERangeError can occur when C++ code calls Delphi code.

Note: By default, the Delphi compiler disables range checking. The compiler can always detect obvious range errors, which prevents the following code from compiling.

//Delphi subrange example

```
var
  SmallValue: 1..3;
  SevenBits: byte;
begin;
  SmallValue := 4;  //Won't compile: constant expression out of bounds
  SevenBits := 256; //Won't compile: constant expression out of bounds
end;
```

Note: However, the compiler can't anticipate out-of-range values provided at run time:

//Delphi subrange example

```
var
  SmallValue: 1..3;
  SevenBits: byte;
  ThirtyTwoBits: Integer
begin;
  ThirtyTwoBits := 4;
  SmallValue := ThirtyTwoBits; //Assigns 4 to SmallValue (SmallValue is stored as a Byte)
  ThirtyTwoBits := 256;
  SevenBits := ThirtyTwoBits;  //Assigns 0 to ThirtyTwoBits (high-order bits discarded)
end;
```

Note: If range checking is enabled, both out-of-bounds assignments throw ERangeError exceptions. To enable range checking, use the Project option in the IDE or the $R+ directive.

Note: Range checking is considered a debugging feature. It produces bigger and slower executables.

Note: Do not confuse ERangeError with ERangeException, which relates to errors using graduated controls in VisualCLX.

See Also

EExternal (see page 22727)

EIntOverflow (see page 22736)

EDivByZero (see page 22726)

ERangeException

159.1.29 ESafecallException Class

ESafecallException is the exception for problems with the safecall calling convention.

Class Hierarchy

Pascal

```
ESafecallException = class(Exception);
```

C++

```
class ESafecallException : public Exception;
```

File

SysUtils

Description

ESafecallException is raised when the safecall error handler has not been set up and a safecall routine returns a non-0 HResult, or if the safecall error handler does not raise an exception.

If this exception occurs, the Comobj unit is probably missing from the application's uses list (Delphi) or not included in the project source file (C++). You may want to consider removing the safecall calling convention from the routine that gave rise to the exception.

See Also

TComponent (see page 2762)

159.1.30 EStackOverflow Class

EStackOverflow is the exception class for stack overflows.

Class Hierarchy

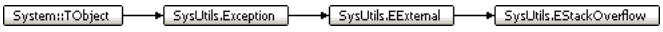

Pascal

```
EStackOverflow = class(EExternal);
```

C++

```
class EStackOverflow : public EExternal;
```

File

SysUtils

Description

EStackOverflow is raised when the system is unable to allocate necessary stack space. This can happen because of extremely large local variables, deeply recursive routines, or invalid machine-language code.

Run-time exception information is saved in fields provided by EExternal (see page 22727).

See Also

EExternal (see page 22727)

159.1.31 EUnderflow Class

EUnderflow is the exception class for values that are too small to be represented with a floating-point variable.

Class Hierarchy

Pascal

```
EUnderflow = class(EMathError);
```

C++

```
class EUnderflow : public EMathError;
```

File

SysUtils

Description

When a floating-point math operation produces a value that is too small to be represented in a floating-point variable, the result ordinarily becomes zero. However, an application can change the numeric coprocessor's control word to unmask underflow hardware exceptions. In this case, EUnderflow is raised when the processor reports an underflow hardware exception. The control word is reset after the exception.

Run-time exception information is saved in fields provided by EExternal (see page 22727).

See Also

EExternal (see page 22727)

EInvalidArgument (see page 15927)

EInvalidOp (see page 22738)

EOverflow (see page 22743)

EZeroDivide (see page 22761)

159.1.32 EVariantError Class

EVariantError is the exception class for errors involving Variant data types.

Class Hierarchy

Pascal

```
EVariantError = class(Exception);
```

C++

```
class EVariantError : public Exception;
```

File

SysUtils

Description

EVariantError is raised when a variant operation fails. EVariantError has a number of descendant classes which provide more specific information about the error.

See Also

EVariantArrayCreateError (see page 23572)

EVariantArrayLockedError (see page 23573)

EVariantBadIndexError (see page 23574)

EVariantBadVarTypeError (see page 23574)

EVariantDispatchError (see page 23575)

EVariantInvalidArgError (see page 23575)

EVariantInvalidOpError (see page 23577)

EVariantNotImplError (see page 23578)

EVariantOutOfMemoryError (see page 23578)

EVariantOverflowError (see page 23579)

EVariantTypeCastError (see page 23580)

EVariantUnexpectedError (see page 23581)

159.1.33 EWin32Error Class

EWin32Error is the exception class for Windows errors.

Class Hierarchy

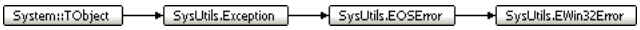

Pascal

```
EWin32Error = class(EOSError);
```

C++

```
class EWin32Error : public EOSError;
```

File

SysUtils

Description

EWin32Error is now deprecated in favor of EOSError (see page 22740).

See Also

RaiseLastWin32Error (see page 22932)

EOSError (see page 22740)

159.1.34 Exception Class

Exception is the base class for all runtime exceptions.

Class Hierarchy

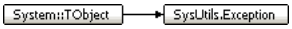

Pascal

```
Exception = class(TObject);
```

C++

```
class Exception : public TObject;
```

File

SysUtils

Description

Exception encapsulates the fundamental properties and methods for all exceptions. Use Exception as the base class for creating custom exceptions.

All methods introduced for the Exception object are constructors that provide alternative ways to create exception messages. Some constructors also create help context IDs. Typically, an application calls these constructors dynamically when an exception occurs.

Exception messages can be hard-coded strings, formatted strings, or strings (including formatted strings) loaded from an application resource.

Exceptions are raised when a runtime error occurs in an application, such as attempting to divide by zero. When an exception is raised, typically an exception instance displays a dialog box describing the error condition. If an application does not handle the exception condition, then the default exception handler is called. This handler also displays a dialog box with an OK button that usually permits an application to continue processing when the user clicks OK.

Note: Standard practice is to derive all exception classes from Exception, and specify a name beginning with "E", followed by an abbreviated, descriptive name.

C++ Examples:

```
/*
In addition to displaying the exception message, which
happens by default, the following code shuts down the
application when an exception is not caught and handled.
AppException should be declared a method of TForm1.
*/

#include <System.hpp>
#include <SysUtils.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnException = AppException;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::AppException(TObject *Sender, Exception *E)
{
  Application->ShowException(E);
```

```
  Application->Terminate();
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  throw(Exception("Hardware error: Divide by 0"));
}
```

159.1.34.1 Exception Members

The following tables list the members exposed by Exception.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22752)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.

159.1.34.2 Exception.Create Constructor

Instantiates an instance of an exception with a simple message string.

Pascal

```
constructor Create(const Msg: string);
```

C++

```
__fastcall Exception(const AnsiString Msg);
```

Description

Call Create to construct an exception object with a simple message string.

Msg is the string containing the runtime error message to display in the exception dialog box. Msg can be a hard-coded string, or can be a function call that returns a string.

C++ Examples:

```
/*
The following example creates a directory 'C:\temp' if it
does not already exist.
}
*/
#include <Filectrl.hpp>
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (!DirectoryExists("c:\\temp"))
  {
    if (!CreateDir("C:\\temp"))
      throw Exception("Cannot create c:\\temp directory.");
  }
  if (!CreateDir("C:\\temp"))
    throw Exception("The c:\\temp directory already exists.");
}
```

Delphi Examples:

```
{
The following example creates a directory 'C:\temp' if it
does not already exist.
}
uses FileCtrl;
procedure TForm1.Button1Click(Sender: TObject);
begin
  if not SysUtils.DirectoryExists('C:\temp') then
    if not CreateDir('C:\temp') then
      raise Exception.Create('Cannot create C:\temp');
  if not CreateDir('C:\temp') then
    raise Exception.Create('Cannot create C:\temp again!');
end;
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
```

```
  file. Additional information is also written to the file so
  that it can be read easily with the FileRead function.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    BackupName: string;
    FileHandle: Integer;
    StringLen: Integer;
    X, Y, I: Integer;
    colCountLength, rowCountLength: Integer;
    Buffer: PChar;
    cellString: string;
  begin
    if SaveDialog1.Execute then
    begin
      if FileExists(SaveDialog1.FileName) then
      begin
        BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
        BackupName := ChangeFileExt(BackupName, '.BAK');
        if not RenameFile(SaveDialog1.FileName, BackupName) then
          raise Exception.Create('Unable to create backup file.');
      end;
      FileHandle := FileCreate(SaveDialog1.FileName);
      { Write out the number of rows and columns in the grid. }
      colCountLength := SizeOf(StringGrid1.ColCount);
      FileWrite(FileHandle,
        Pchar(StringGrid1.ColCount), colCountLength);
      rowCountLength := SizeOf(StringGrid1.RowCount);
      FileWrite(FileHandle,
        PChar(StringGrid1.RowCount), rowCountLength);
      for X := 0 to StringGrid1.ColCount - 1 do
      begin
        for Y := 0 to StringGrid1.RowCount - 1 do
        begin
          try
            { Write out the length of each string, followed by the string itself. }
            StringLen := Length(StringGrid1.Cells[X,Y]);
            FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
            GetMem(Buffer, StringLen); { allocate the buffer }
            cellString := StringGrid1.Cells[X,Y];
            for I := 1 to StringLen do
               Buffer[I - 1] := cellString[I];
            FileWrite(FileHandle, Buffer^, StringLen);
          finally
            FreeMem(Buffer, StringLen);
          end;
        end;
      end;
      FileClose(FileHandle);
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with StringGrid1 do
    begin
      Cells[1,0] := 'Column 1';
      Cells[2,0] := 'Column 2';
      Cells[3,0] := 'Column 3';
      Cells[4,0] := 'Column 4';
      Cells[0,1] := 'Row 1';
      Cells[1,1] := 'Object';
      Cells[2,1] := 'Pascal';
      Cells[3,1] := 'is';
      Cells[4,1] := 'excellent';
      Cells[0,2] := 'Row 2';
```

```
      Cells[1,2] := 'Delphi';
      Cells[2,2] := 'is';
      Cells[4,2] := 'RAD';
    end;
  end;
  {
  In addition to displaying the exception message, which
  happens by default, the following code shuts down the
  application when an exception is not caught and handled.
  AppException should be declared a method of TForm1.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Application.OnException := AppException;
  end;
  procedure TForm1.AppException(Sender: TObject; E: Exception);
  begin
    Application.ShowException(E);
    Application.Terminate;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    raise EPasswordInvalid.Create('Incorrect password entered');
  end;
```

See Also

CreateFmt (see page 22754)

CreateRes (see page 22756)

CreateResFmt (see page 22757)

Message (see page 22760)

159.1.34.3 Exception.CreateFmt Constructor

Instantiates an instance of an exception with a formatted message string.

Pascal

```
constructor CreateFmt(const Msg: string; const Args: array of const);
```

C++

```
__fastcall Exception(const AnsiString Msg, const array of const Args);
```

Description

Call CreateFmt to construct an exception with a formatted message string. Msg is a string constant containing format specifiers to be replaced with values from Args. Args is an array of constants containing values to format according to format specifiers in Msg, and insert into Msg. CreateFmt calls the Format (see page 22897) function to transform Msg with the values in Arg.

See Also

CreateResFmt (see page 22757)

CreateResFmtHelp (see page 22758)

Format (see page 22897)

Message (see page 22760)

159.1.34.4 Exception.CreateFmtHelp Constructor

Instantiates an instance of an exception with a formatted message string and a help-context ID.

Pascal

```
constructor CreateFmtHelp(const Msg: string; const Args: array of const; AHelpContext:
Integer);
```

C++

```
__fastcall Exception(const AnsiString Msg, const array of const Args, int AHelpContext);
```

Description

Call CreateFmtHelp to construct an exception with a formatted message string and a help context ID. Msg is a string constant containing format specifiers to be replaced with values from Args. Args is an array of constants containing values to format according to format specifiers in Msg, and insert into Msg. CreateFmt (see page 22754) calls the Format (see page 22897) function to transform Msg with the values in Arg.

AHelpContext is an integer that specifies the context-sensitive help ID for the exception.

See Also

CreateHelp (see page 22755)

CreateResFmtHelp (see page 22758)

Format (see page 22897)

HelpContext (see page 22760)

Message (see page 22760)

159.1.34.5 Exception.CreateHelp Constructor

Creates an instance of an exception with a simple message string and a help- context ID.

Pascal

```
constructor CreateHelp(const Msg: string; AHelpContext: Integer);
```

C++

```
__fastcall Exception(const AnsiString Msg, int AHelpContext);
```

Description

Call CreateHelp to construct an exception with a simple message string and a help-context ID. Msg is the string containing the runtime error message to display in the exception dialog box. Msg can be a hard-coded string, or can be a function call that returns a string.

AHelpContext is an integer that specifies the context-sensitive help ID for the exception.

See Also

HelpContext (see page 22760)

Message (see page 22760)

CreateFmtHelp (see page 22755)

CreateResFmtHelp (see page 22758)

Format (see page 22897)

159.1.34.6 CreateRes Constructor

This is the overview for the CreateRes constructor overload.

Overload List

	Name	Description
	Exception.CreateRes (Integer) (see page 22756)	Creates an instance of an exception with a simple message string that is loaded from the application's resources.
	Exception.CreateRes (PResStringRec) (see page 22756)	Creates an instance of an exception with a simple message string that is loaded from the application's resources.

159.1.34.6.1 Exception.CreateRes Constructor (Integer)

Creates an instance of an exception with a simple message string that is loaded from the application's resources.

Pascal

```
constructor CreateRes(Ident: Integer); overload;
constructor CreateRes(ResStringRec: PResStringRec); overload;
```

C++

```
__fastcall Exception(int Ident);
__fastcall Exception(PResStringRec ResStringRec);
```

Description

Call CreateRes to construct an exception with a simple message string loaded from an application's resources. Resources are bound into the application executable at compile time, but at design time they exist in a separate resource file.

Ident is the unique ID of the resource as specified in the resource file. If Ident is not a valid resource ID, CreateRes creates an empty message string for the exception.

ResStringRec is a pointer to a resource string. This syntax appears as follows:

```
resourcestring sMyNewErrorMessage = 'You shouldn''t have done that';
...
Exception.Create(@sMyNewErrorMessage);
```

See Also

Message (see page 22760)

CreateResFmt (see page 22757)

CreateResFmtHelp (see page 22758)

159.1.34.7 CreateResFmt Constructor

This is the overview for the CreateResFmt constructor overload.

Overload List

	Name	Description
	Exception.CreateResFmt (Integer, array of const) (see page 22757)	Creates an instance of an exception with a message string that is loaded from the application's resources and then formatted.
	Exception.CreateResFmt (PResStringRec, array of const) (see page 22757)	Creates an instance of an exception with a message string that is loaded from the application's resources and then formatted.

159.1.34.7.1 Exception.CreateResFmt Constructor (Integer, array of const)

Creates an instance of an exception with a message string that is loaded from the application's resources and then formatted.

Pascal

```
constructor CreateResFmt(Ident: Integer; const Args: array of const); overload;
constructor CreateResFmt(ResStringRec: PResStringRec; const Args: array of const); overload;
```

C++

```
__fastcall Exception(int Ident, const array of const Args);
__fastcall Exception(PResStringRec ResStringRec, const array of const Args);
```

Description

Call CreateResFmt to construct an exception with a message string loaded from an application's resources and then formatted with additional information. Resources are bound into the application executable at compile time, but at design time they exist in a separate resource file.

Ident is the unique ID of the resource as specified in the resource file. If Ident is not a valid resource ID, CreateResFmt creates an empty message string for the exception.

ResStringRec is a pointer to a resource string. This syntax appears as follows:

```
resourcestring sMyNewErrorMessage = 'Illegal value: %s';
...
Exception.CreateResFmt(@sMyNewErrorMessage, [-1]);
```

Args is an array of constants containing values to:

1Format (see page 22897) according to format specifiers embedded in the string, and

2Insert into the message string.

CreateResFmt calls the Format (see page 22897) function to transform the message string with the values in Arg.

See Also

Message (see page 22760)

CreateRes (see page 22756)

CreateResFmtHelp (see page 22758)

Format (see page 22897)

159.1.34.8 CreateResFmtHelp Constructor

This is the overview for the CreateResFmtHelp constructor overload.

Overload List

	Name	Description
	Exception.CreateResFmtHelp (Integer, array of const, Integer) (see page 22758)	Creates an instance of an exception with a help-context ID and a formatted message string loaded from the application resources
	Exception.CreateResFmtHelp (PResStringRec, array of const, Integer) (see page 22758)	Creates an instance of an exception with a help-context ID and a formatted message string loaded from the application resources

159.1.34.8.1 Exception.CreateResFmtHelp Constructor (Integer, array of const, Integer)

Creates an instance of an exception with a help-context ID and a formatted message string loaded from the application resources

Pascal

```
constructor CreateResFmtHelp(ResStringRec: PResStringRec; const Args: array of const;
AHelpContext: Integer); overload;
constructor CreateResFmtHelp(Ident: Integer; const Args: array of const; AHelpContext:
Integer); overload;
```

C++

```
__fastcall Exception(PResStringRec ResStringRec, const array of const Args, int AHelpContext);
__fastcall Exception(int Ident, const array of const Args, int AHelpContext);
```

Description

Call CreateResFmtHelp to construct an exception with a:

1Message (see page 22760) string loaded from an application's resources and formatted with additional information.

2Help-context ID.

Resources are bound into the application executable at compile time, but at design time they exist in a separate resource file. Ident is the unique ID of the resource as specified in the resource file. If Ident is not a valid resource ID, CreateResFmt (see page 22757) creates an empty message string.

ResStringRec is a pointer to a resource string. This syntax appears as follows:

```
resourcestring sMyNewErrorMessage = 'Illegal value: %s';
const IDH_MYNEWERROR: Integer = 30000;
...
Exception.CreateResFmt(@sMyNewErrorMessage, [-1], IDH_MYNEWERROR);
```

Args is an array of constants containing values to:

1Format (see page 22897) according to format specifiers embedded in the string, and

2Insert into the message string.

CreateResFmtHelp calls the Format (see page 22897) function to transform the message string with the values in Arg.

AHelpContext is an integer that specifies the context-sensitive help ID for the exception.

See Also

HelpContext (see page 22760)

Message (see page 22760)

CreateHelp (see page 22755)

CreateRes (see page 22756)

CreateResHelp (see page 22759)

Format (see page 22897)

159.1.34.9 CreateResHelp Constructor

This is the overview for the CreateResHelp constructor overload.

Overload List

	Name	Description
	Exception.CreateResHelp (Integer, Integer) (see page 22759)	Creates an instance of an exception with a help-context ID and a simple message string that is loaded from the application's resources.
	Exception.CreateResHelp (PResStringRec, Integer) (see page 22759)	Creates an instance of an exception with a help-context ID and a simple message string that is loaded from the application's resources.

159.1.34.9.1 Exception.CreateResHelp Constructor (Integer, Integer)

Creates an instance of an exception with a help-context ID and a simple message string that is loaded from the application's resources.

Pascal

```
constructor CreateResHelp(Ident: Integer; AHelpContext: Integer); overload;
constructor CreateResHelp(ResStringRec: PResStringRec; AHelpContext: Integer); overload;
```

C++

```
__fastcall Exception(int Ident, int AHelpContext);
__fastcall Exception(PResStringRec ResStringRec, int AHelpContext);
```

Description

Call CreateResHelp to construct an exception with a help-context ID and a simple message string loaded from an application's resources. Resources are bound into the application executable at compile time, but at design time they exist in a separate resource file.

Ident is the unique ID of the resource as specified in the resource file. If Ident is not a valid resource ID, CreateResHelp creates an empty message string for the exception.

AHelpContext is an integer that specifies the context-sensitive help ID for the exception.

ResStringRec is a pointer to a resource string. Its syntax appears as follows:

```
resourcestring sMyNewErrorMessage = 'You shouldn''t have done that';
const IDH_MYNEWERROR: Integer = 30000;
...
Exception.Create(@sMyNewErrorMessage, IDH_MYNEWERROR);
```

See Also

HelpContext (see page 22760)

Message (see page 22760)

CreateHelp (see page 22755)

CreateRes (see page 22756)

CreateResFmtHelp (see page 22758)

Format (see page 22897)

159.1.34.10 Exception Properties

The properties of the Exception class are listed here.

Public Properties

	Name	Description
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.

159.1.34.10.1 Exception.HelpContext Property

Indicates the context ID number for context-sensitive online help associated with the exception.

Pascal

```
property HelpContext: Integer;
```

C++

```
__property int HelpContext;
```

Description

HelpContext provides storage for the help context ID number associated with the exception object. The context ID number determines the contents of the help screen that appears when a user presses F1 in response to an exception error message.

Note: To initialize HelpContext when the Exception (see page 22750) object is instantiated, use the appropriate constructor.

Note: In Delphi code, instantiate the Exception (see page 22750) object with CreateFmtHelp (see page 22755), CreateHelp (see page 22755), CreateResFmtHelp (see page 22758), or CreateResHelp (see page 22759).

Note: In C++ code, use a form of the constructor that takes a help context parameter.

See Also

Create (see page 22752)

CreateFmtHelp (see page 22755)

CreateHelp (see page 22755)

CreateResFmtHelp (see page 22758)

CreateResHelp (see page 22759)

159.1.34.10.2 Exception.Message Property

Message contains the text string to display in the exception dialog box when the exception is raised.

Pascal

```
property Message: string;
```

C++

```
__property AnsiString Message;
```

Description

Message stores the error-message string to display when the exception is raised. All Exception (see page 22750) constructors expect a string parameter to store in Message. Message text can be hard-coded as a parameter to an Exception (see page 22750) constructor, created as dynamic parameter, or loaded from a resource file as a static or dynamically formatted parameter.

See Also

Create (see page 22752)

CreateFmt (see page 22754)

CreateFmtHelp (see page 22755)

CreateHelp (see page 22755)

CreateRes (see page 22756)

CreateResFmt (see page 22757)

CreateResFmtHelp (see page 22758)

CreateResHelp (see page 22759)

159.1.35 EZeroDivide Class

EZeroDivide is the exception class for floating-point divide-by-zero errors.

Class Hierarchy

Pascal

```
EZeroDivide = class(EMathError);
```

C++

```
class EZeroDivide : public EMathError;
```

File

SysUtils

Description

EZeroDivide exception is raised when an application attempts to divide a floating-point value by zero.

Run-time exception information is saved in fields provided by EExternal (see page 22727).

Note: Integer divide-by-zero errors raise the EDivByZero (see page 22726) exception.

See Also

EExternal (see page 22727)

EInvalidArgument (see page 15927)

EInvalidOp (see page 22738)

EOverflow (see page 22743)

EUnderflow (see page 22748)

EDivByZero (see page 22726)

159.1.36 TLanguages Class

TLanguages lists all available Windows locales.

Class Hierarchy

SysUtils.TLanguages

Pascal

```
TLanguages = class;
```

C++

```
class TLanguages;
```

File

SysUtils

Description

The Languages (see page 22926) function returns a TLanguages instance that lists the Windows locales supported by the current system. For each locale, TLanguages indicates the locale ID (see page 22764) (LCID), standard name, and recommended file-name extension. This information is obtained from the operating system; all TLanguages properties are read-only.

See Also

Languages (see page 22926)

159.1.36.1 TLanguages Members

The following tables list the members exposed by TLanguages.

Public Methods

	Name	Description
	Create (see page 22763)	This is Create, a member of class TLanguages.
	IndexOf (see page 22763)	Returns the index of the first locale in the array with the specified LCID.

Public Properties

	Name	Description
R	Count (see page 22764)	Contains the number of locales listed in the TLanguages (see page 22762) object.
R	Ext (see page 22764)	Contains the standard file-name extension at the specified position in the array.
R	ID (see page 22764)	Contains a locale ID as a string.
R	LocaleID (see page 22765)	Contains a locale ID (see page 22764) as an integer.
R	Name (see page 22765)	Contains the locale name at the specified position in the array.
R	NameFromLCID (see page 22766)	Contains the locale name specified by a string representation of a locale ID (see page 22764).

R	NameFromLocaleID (see page 22766)	Contains the locale name specified by an integer locale ID (see page 22764).

159.1.36.2 TLanguages.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TLanguages();
```

Description

This is Create, a member of class TLanguages.

159.1.36.3 TLanguages Methods

The methods of the TLanguages class are listed here.

Public Methods

	Name	Description
	IndexOf (see page 22763)	Returns the index of the first locale in the array with the specified LCID.

159.1.36.3.1 TLanguages.IndexOf Method

Returns the index of the first locale in the array with the specified LCID.

Pascal

```
function IndexOf(ID: LCID): Integer;
```

C++

```
__fastcall int IndexOf(LCID ID);
```

Description

Call IndexOf to get the position in the array of a specified locale ID (see page 22764). For example, if U.S. English is the first locale listed in TLanguages (see page 22762), then IndexOf($0409) returns 0.

If the specified LCID is not found, IndexOf returns -1.

See Also

Ext (see page 22764)

ID (see page 22764)

LocaleID (see page 22765)

Name (see page 22765)

159.1.36.4 TLanguages Properties

The properties of the TLanguages class are listed here.

Public Properties

	Name	Description
R	Count (see page 22764)	Contains the number of locales listed in the TLanguages (see page 22762) object.
R	Ext (see page 22764)	Contains the standard file-name extension at the specified position in the array.
R	ID (see page 22764)	Contains a locale ID as a string.
R	LocaleID (see page 22765)	Contains a locale ID (see page 22764) as an integer.
R	Name (see page 22765)	Contains the locale name at the specified position in the array.
R	NameFromLCID (see page 22766)	Contains the locale name specified by a string representation of a locale ID (see page 22764).
R	NameFromLocaleID (see page 22766)	Contains the locale name specified by an integer locale ID (see page 22764).

159.1.36.4.1 TLanguages.Count Property

Contains the number of locales listed in the TLanguages (see page 22762) object.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read Count to determine the number of locales supported by the current system.

See Also

ID (see page 22764)

LocaleID (see page 22765)

IndexOf (see page 22763)

159.1.36.4.2 TLanguages.Ext Property

Contains the standard file-name extension at the specified position in the array.

Pascal

```
property Ext [Index: Integer]: string;
```

C++

```
__property AnsiString Ext[int Index];
```

Description

Read Ext to get the standard three-letter file-name extension for a locale. For example, Ext[IndexOf (see page 22763)($0409)] returns the string ENU.

Ext is a zero-based array.

See Also

IndexOf (see page 22763)

159.1.36.4.3 TLanguages.ID Property

Contains a locale ID as a string.

Pascal

```
property ID [Index: Integer]: string;
```

C++

```
__property AnsiString ID[int Index];
```

Description

Read ID to get the locale ID at a specified position in the array as a string. (To get the same ID as an integer, use the LocaleID (see page 22765) property.) For example, if U.S. English is the first locale listed in TLanguages (see page 22762), then ID[0] returns the string $00000409. You can iterate through the ID array (from 0 to Count (see page 22764) - 1) to determine whether a specific locale is in the array.

See Also

Count (see page 22764)

LocaleID (see page 22765)

159.1.36.4.4 TLanguages.LocaleID Property

Contains a locale ID (see page 22764) as an integer.

Pascal

```
property LocaleID [Index: Integer]: LCID;
```

C++

```
__property LCID LocaleID[int Index];
```

Description

Read LocaleID to get the locale ID (see page 22764) at a specified position in the array as an integer. (To get the same ID (see page 22764) as a string, use the ID (see page 22764) property.) For example, if U.S. English is the first locale listed in TLanguages (see page 22762), then LocaleID[0] returns the integer $409. You can iterate through the LocaleID array (from 0 to Count (see page 22764) - 1) to determine whether a specific locale is in the array.

See Also

Count (see page 22764)

ID (see page 22764)

159.1.36.4.5 TLanguages.Name Property

Contains the locale name at the specified position in the array.

Pascal

```
property Name [Index: Integer]: string;
```

C++

```
__property AnsiString Name[int Index];
```

Description

Read Name to get the standard Windows name for a locale. For example, Name[IndexOf (see page 22763)($0409)] returns the string English (United States).

Name is a zero-based array.

See Also

IndexOf (see page 22763)

NameFromLCID (see page 22766)

NameFromLocaleID (see page 22766)

159.1.36.4.6 TLanguages.NameFromLCID Property

Contains the locale name specified by a string representation of a locale ID (see page 22764).

Pascal

```
property NameFromLCID [const ID: string]: string;
```

C++

```
__property AnsiString NameFromLCID[AnsiString const ID];
```

Description

Read NameFromLCID to get the standard Windows name for a locale using a locale ID (see page 22764) represented by a string.

See Also

Name (see page 22765)

NameFromLocaleID (see page 22766)

159.1.36.4.7 TLanguages.NameFromLocaleID Property

Contains the locale name specified by an integer locale ID (see page 22764).

Pascal

```
property NameFromLocaleID [ID: LCID]: string;
```

C++

```
__property AnsiString NameFromLocaleID[LCID ID];
```

Description

Read NameFromLocaleID to get the standard Windows name for a locale using an integer locale ID (see page 22764). For example, NameFromLCID (see page 22766)[$0409] returns the string English (United States).

See Also

NameFromLCID (see page 22766)

Name (see page 22765)

159.1.37 TMultiReadExclusiveWriteSynchronizer Class

TMultiReadExclusiveWriteSynchronizer protects memory in a multi-threaded application.

Class Hierarchy

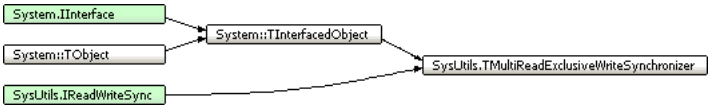

Pascal

```
TMultiReadExclusiveWriteSynchronizer = class(TInterfacedObject, IReadWriteSync);
```

C++

```
class TMultiReadExclusiveWriteSynchronizer : public TInterfacedObject, public IReadWriteSync;
```

File

SysUtils

Description

Use TMultiReadExclusiveWriteSynchronizer to guard access to memory in a multi-threaded application. Unlike a critical section, which blocks all other threads from reading or writing its associated memory, TMultiReadExclusiveWriteSynchronizer allows multiple threads to read from the protected memory simultaneously, while ensuring that any thread writing to the memory has exclusive access.

In applications where threads read from an object or variable frequently and only write to it occasionally, using the multi-read exclusive-write synchronizer instead of a critical section can result in considerable performance improvement.

All access to the protected memory must be bracketed by calls to the BeginRead (see page 22769) and EndRead (see page 22770) or BeginWrite (see page 22770) and EndWrite (see page 22771) methods. Any thread that reads from or writes to this memory without using these calls can introduce thread conflicts.

See Also

TCriticalSection (see page 22058)

TThread (see page 3108)

Using the Multi-read Exclusive-write Synchronizer

159.1.37.1 TMultiReadExclusiveWriteSynchronizer Members

The following tables list the members exposed by TMultiReadExclusiveWriteSynchronizer.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginRead (see page 22769)	Allows a thread to read from the protected memory when it is safe to do so.
	BeginWrite (see page 22770)	Allows a thread to write to the protected memory when it is safe to do so.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22769)	Creates and initializes an instance of TMultiReadExclusiveWriteSynchronizer (see page 22766).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22769)	Destroys an instance of TMultiReadExclusiveWriteSynchronizer (see page 22766).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndRead (see page 22770)	Removes a thread from the internal list of threads that are accessing the protected memory.
	EndWrite (see page 22771)	Allows other threads to access the associated memory.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	RevisionLevel (see page 22771)	Changes revision level of protected memory.

159.1.37.2 TMultiReadExclusiveWriteSynchronizer.Create Constructor

Creates and initializes an instance of TMultiReadExclusiveWriteSynchronizer (see page 22766).

Pascal

```
constructor Create;
```

C++

```
__fastcall TMultiReadExclusiveWriteSynchronizer();
```

Description

Call Create to instantiate TMultiReadExclusiveWriteSynchronizer (see page 22766). The synchronizer must be created in the same thread as the memory it protects, so that it is accessible to all threads that need to access that memory.

159.1.37.3 TMultiReadExclusiveWriteSynchronizer.Destroy Destructor

Destroys an instance of TMultiReadExclusiveWriteSynchronizer (see page 22766).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMultiReadExclusiveWriteSynchronizer();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TMultiReadExclusiveWriteSynchronizer (see page 22766) reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

159.1.37.4 TMultiReadExclusiveWriteSynchronizer Methods

The methods of the TMultiReadExclusiveWriteSynchronizer class are listed here.

Public Methods

	Name	Description
	BeginRead (see page 22769)	Allows a thread to read from the protected memory when it is safe to do so.
	BeginWrite (see page 22770)	Allows a thread to write to the protected memory when it is safe to do so.
	EndRead (see page 22770)	Removes a thread from the internal list of threads that are accessing the protected memory.
	EndWrite (see page 22771)	Allows other threads to access the associated memory.

159.1.37.4.1 TMultiReadExclusiveWriteSynchronizer.BeginRead Method

Allows a thread to read from the protected memory when it is safe to do so.

Pascal

```
procedure BeginRead;
```

C++

```
__fastcall BeginRead();
```

Description

Call BeginRead before reading from the memory protected by the multi-read exclusive-write synchronizer. BeginRead does not return until there are no other threads writing to the memory. After calling BeginRead, other threads can be granted read access to the protected memory by the BeginRead method. However, any thread asking for write permission by calling BeginWrite (see page 22770) must wait until all reading threads have called EndRead (see page 22770).

See Also

EndRead (see page 22770)

BeginWrite (see page 22770)

Using the Multi-read Exclusive-write Synchronizer

159.1.37.4.2 TMultiReadExclusiveWriteSynchronizer.BeginWrite Method

Allows a thread to write to the protected memory when it is safe to do so.

Pascal

```
function BeginWrite: Boolean;
```

C++

```
__fastcall Boolean BeginWrite();
```

Description

BeginWrite obtains an exclusive write lock on the memory protected by the synchronizer object. Call BeginWrite before writing to this memory. BeginWrite does not return until no other threads are reading from or writing to the memory. After a thread calls BeginWrite, any other thread asking to access that memory by calling BeginRead (see page 22769) or BeginWrite must wait until the writing thread calls EndWrite (see page 22771).

Note that a write lock operation is not atomic. Data in protected memory will always be in a consistent state when BeginWrite returns, but is not necessarily in the same state as when BeginWrite was called. As a rule, a thread should always discard previous samples from protected memory after promoting a read lock to a write lock. However, the calling thread can determine whether a state change has occurred by examining return value of BeginWrite: true if protected memory has not be written to by another thread, false if another thread may have modified protected memory.

See Also

EndWrite (see page 22771)

BeginRead (see page 22769)

Using the Multi-read Exclusive-write Synchronizer

159.1.37.4.3 TMultiReadExclusiveWriteSynchronizer.EndRead Method

Removes a thread from the internal list of threads that are accessing the protected memory.

Pascal

```
procedure EndRead;
```

C++

```
__fastcall EndRead();
```

Description

Call EndRead after reading from the memory protected by the multi-read exclusive-write synchronizer. Each call to BeginRead (see page 22769) must be matched by a call to EndRead, or other threads can never return from the BeginWrite (see page 22770) method.

See Also

BeginRead (see page 22769)

BeginWrite (see page 22770)

Using the Multi-read Exclusive-write Synchronizer

159.1.37.4.4 TMultiReadExclusiveWriteSynchronizer.EndWrite Method

Allows other threads to access the associated memory.

Pascal

```
procedure EndWrite;
```

C++

```
__fastcall EndWrite();
```

Description

Call EndWrite after writing to the memory that is protected by the multi-read exclusive-write synchronizer. Each call to BeginWrite (see page 22770) must be matched by a call to EndWrite, or other threads can never return from the BeginRead (see page 22769) or BeginWrite (see page 22770) methods.

See Also

BeginWrite (see page 22770)

BeginRead (see page 22769)

Using the Multi-read Exclusive-write Synchronizer

159.1.37.5 TMultiReadExclusiveWriteSynchronizer Properties

The properties of the TMultiReadExclusiveWriteSynchronizer class are listed here.

Public Properties

	Name	Description
R	RevisionLevel (see page 22771)	Changes revision level of protected memory.

159.1.37.5.1 TMultiReadExclusiveWriteSynchronizer.RevisionLevel Property

Changes revision level of protected memory.

Pascal

```
property RevisionLevel: Cardinal;
```

C++

```
__property unsigned RevisionLevel;
```

Description

RevisionLevel changes each time a write lock is granted. This property should be used for identity comparisons only, not for determining revision sequence.

159.1.38 TSimpleRWSync Class

Class Hierarchy

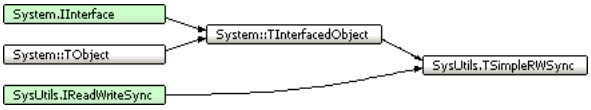

Pascal

```
TSimpleRWSync = class(TInterfacedObject, IReadWriteSync);
```

C++

```
class TSimpleRWSync : public TInterfacedObject, public IReadWriteSync;
```

File

SysUtils

Description

This is class SysUtils.TSimpleRWSync.

159.1.38.1 TSimpleRWSync Members

The following tables list the members exposed by TSimpleRWSync.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginRead (see page 22774)	This is BeginRead, a member of class TSimpleRWSync.

	BeginWrite (see page 22775)	This is BeginWrite, a member of class TSimpleRWSync.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22773)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22774)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndRead (see page 22775)	This is EndRead, a member of class TSimpleRWSync.
	EndWrite (see page 22775)	This is EndWrite, a member of class TSimpleRWSync.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

159.1.38.2 TSimpleRWSync.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TSimpleRWSync();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

159.1.38.3 TSimpleRWSync.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSimpleRWSync();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

159.1.38.4 TSimpleRWSync Methods

The methods of the TSimpleRWSync class are listed here.

Public Methods

	Name	Description
	BeginRead (see page 22774)	This is BeginRead, a member of class TSimpleRWSync.
	BeginWrite (see page 22775)	This is BeginWrite, a member of class TSimpleRWSync.
	EndRead (see page 22775)	This is EndRead, a member of class TSimpleRWSync.
	EndWrite (see page 22775)	This is EndWrite, a member of class TSimpleRWSync.

159.1.38.4.1 TSimpleRWSync.BeginRead Method

Pascal

```
procedure BeginRead;
```

C++

```
__fastcall BeginRead();
```

Description

This is BeginRead, a member of class TSimpleRWSync.

159.1.38.4.2 TSimpleRWSync.BeginWrite Method

Pascal

```
function BeginWrite: Boolean;
```

C++

```
__fastcall Boolean BeginWrite();
```

Description

This is BeginWrite, a member of class TSimpleRWSync.

159.1.38.4.3 TSimpleRWSync.EndRead Method

Pascal

```
procedure EndRead;
```

C++

```
__fastcall EndRead();
```

Description

This is EndRead, a member of class TSimpleRWSync.

159.1.38.4.4 TSimpleRWSync.EndWrite Method

Pascal

```
procedure EndWrite;
```

C++

```
__fastcall EndWrite();
```

Description

This is EndWrite, a member of class TSimpleRWSync.

159.1.39 TThreadLocalCounter Class

Class Hierarchy

SysUtils.TThreadLocalCounter

Pascal

```
TThreadLocalCounter = class;
```

C++

```
class TThreadLocalCounter;
```

File

SysUtils

Description

This is class SysUtils.TThreadLocalCounter.

159.1.39.1 TThreadLocalCounter Members

The following tables list the members exposed by TThreadLocalCounter.

Public Methods

	Name	Description
	Close (see page 22776)	This is Close, a member of class TThreadLocalCounter.
	Delete (see page 22777)	This is Delete, a member of class TThreadLocalCounter.
V	Destroy (see page 22776)	This is Destroy, a member of class TThreadLocalCounter.
	Open (see page 22777)	This is Open, a member of class TThreadLocalCounter.

159.1.39.2 TThreadLocalCounter.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TThreadLocalCounter();
```

Description

This is Destroy, a member of class TThreadLocalCounter.

159.1.39.3 TThreadLocalCounter Methods

The methods of the TThreadLocalCounter class are listed here.

Public Methods

	Name	Description
	Close (see page 22776)	This is Close, a member of class TThreadLocalCounter.
	Delete (see page 22777)	This is Delete, a member of class TThreadLocalCounter.
	Open (see page 22777)	This is Open, a member of class TThreadLocalCounter.

159.1.39.3.1 TThreadLocalCounter.Close Method

Pascal

```
procedure Close(var Thread: PThreadInfo);
```

C++

```
__fastcall Close(PThreadInfo Thread);
```

Description

This is Close, a member of class TThreadLocalCounter.

159.1.39.3.2 TThreadLocalCounter.Delete Method

Pascal

```
procedure Delete(var Thread: PThreadInfo);
```

C++

```
__fastcall Delete(PThreadInfo Thread);
```

Description

This is Delete, a member of class TThreadLocalCounter.

159.1.39.3.3 TThreadLocalCounter.Open Method

Pascal

```
procedure Open(var Thread: PThreadInfo);
```

C++

```
__fastcall Open(PThreadInfo Thread);
```

Description

This is Open, a member of class TThreadLocalCounter.

159.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IReadWriteSync (see page 22778)	This is class SysUtils.IReadWriteSync.

159.2.1 IReadWriteSync Interface

Class Hierarchy

SysUtils.IReadWriteSync

Pascal

```
IReadWriteSync = interface;
```

C++

```
__interface IReadWriteSync;
```

File

SysUtils

Description

This is class SysUtils.IReadWriteSync.

159.2.1.1 IReadWriteSync Members

The following tables list the members exposed by IReadWriteSync.

Public Methods

	Name	Description
	BeginRead (see page 22779)	This is BeginRead, a member of class IReadWriteSync.
	BeginWrite (see page 22779)	This is BeginWrite, a member of class IReadWriteSync.
	EndRead (see page 22779)	This is EndRead, a member of class IReadWriteSync.
	EndWrite (see page 22779)	This is EndWrite, a member of class IReadWriteSync.

159.2.1.2 IReadWriteSync Methods

The methods of the IReadWriteSync class are listed here.

Public Methods

	Name	Description
	BeginRead (see page 22779)	This is BeginRead, a member of class IReadWriteSync.
	BeginWrite (see page 22779)	This is BeginWrite, a member of class IReadWriteSync.
	EndRead (see page 22779)	This is EndRead, a member of class IReadWriteSync.

	EndWrite (see page 22779)	This is EndWrite, a member of class IReadWriteSync.

159.2.1.2.1 IReadWriteSync.BeginRead Method

Pascal

```
procedure BeginRead;
```

C++

```
__fastcall BeginRead();
```

Description

This is BeginRead, a member of class IReadWriteSync.

159.2.1.2.2 IReadWriteSync.BeginWrite Method

Pascal

```
function BeginWrite: Boolean;
```

C++

```
__fastcall Boolean BeginWrite();
```

Description

This is BeginWrite, a member of class IReadWriteSync.

159.2.1.2.3 IReadWriteSync.EndRead Method

Pascal

```
procedure EndRead;
```

C++

```
__fastcall EndRead();
```

Description

This is EndRead, a member of class IReadWriteSync.

159.2.1.2.4 IReadWriteSync.EndWrite Method

Pascal

```
procedure EndWrite;
```

C++

```
__fastcall EndWrite();
```

Description

This is EndWrite, a member of class IReadWriteSync.

159.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	Abort (see page 22786)	Raises a silent exception.
	AddExitProc (see page 22787)	Adds a procedure to the exit list.
	AddTerminateProc (see page 22787)	Adds a terminate procedure to the system list of termination procedures.
	AdjustLineBreaks (see page 22788)	Adjusts line break characters to reflect Windows or Linux style.
	AnsiCompareFileName (see page 22788)	Compares file names based on the current locale.
	AnsiCompareStr (see page 22789)	Compares strings based on the current locale with case sensitivity.
	AnsiCompareText (see page 22790)	Compares strings based on the current locale without case sensitivity.
	AnsiDequotedStr (see page 22790)	Converts a quoted string into an unquoted string
	AnsiExtractQuotedStr (see page 22791)	Converts a quoted string into an unquoted string.
	AnsiLastChar (see page 22791)	Returns a pointer to the last character in a string.
	AnsiLowerCase (see page 22792)	Returns a string that is a copy of the given string converted to lower case.
	AnsiLowerCaseFileName (see page 22792)	Converts a file name to lower case.
	AnsiPos (see page 22793)	Locates the position of a substring.
	AnsiQuotedStr (see page 22793)	Returns the quoted version of a string.
	AnsiSameStr (see page 22794)	Compares strings based on the current locale with case sensitivity.
	AnsiSameText (see page 22794)	Compares strings based on the current locale without case sensitivity.
	AnsiStrComp (see page 22795)	Compares null-terminated character strings.
	AnsiStrIComp (see page 22795)	Compares null terminated character strings case insensitively.
	AnsiStrLastChar (see page 22796)	Returns a pointer to the last character in a null-terminated string.
	AnsiStrLComp (see page 22796)	Compares the first MaxLen bytes of two null-terminated strings, case-sensitively.
	AnsiStrLIComp (see page 22797)	Compares two strings, case-insensitively, up to the first MaxLen bytes.
	AnsiStrLower (see page 22798)	Converts all characters in a null-terminated string to lower case.
	AnsiStrPos (see page 22798)	Returns a pointer to the first occurrence of a substring.
	AnsiStrRScan (see page 22799)	Returns a pointer to the last occurrence of a specified character in a specified string.
	AnsiStrScan (see page 22799)	Returns a pointer to first occurrence of a character in a string.
	AnsiStrUpper (see page 22800)	Converts all characters in a null-terminated string to upper case.
	AnsiUpperCase (see page 22800)	Converts a string to upper case.
	AnsiUpperCaseFileName (see page 22801)	Converts a file name to upper case.
	AppendStr (see page 22801)	Appends a dynamically allocated string to an existing string.
	AssignStr (see page 22802)	Assigns a new dynamically allocated string to a specified pointer.
	Beep (see page 22802)	Emits a message beep.
	BoolToStr (see page 22804)	Converts a Boolean value to a string.
	ByteToCharIndex (see page 22806)	Returns the position of the character that contains a specified byte in a string.
	ByteToCharLen (see page 22807)	Returns the number of characters fully contained in the first MaxLen bytes of a string.
	ByteType (see page 22807)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	CallTerminateProcs (see page 22808)	Calls all of the functions in the termination procedure list.
	ChangeFileExt (see page 22808)	Changes the extension of a file name.
	ChangeFilePath (see page 22810)	Changes the path of a file name.
	CharLength (see page 22810)	Returns number of bytes used by a character.
	CharToByteIndex (see page 22811)	Returns the 1-based index of the first byte of a specified character in a string.
	CharToByteLen (see page 22811)	Returns the length, in bytes, of the first MaxLen characters of a string.
	CheckWin32Version (see page 22812)	Check to see if you are running on a specific level (or higher) of the Windows 32 bit Operating System (see page 22079).

	CompareMem (see page 22812)	Performs a binary comparison of two memory images.
	CompareStr (see page 22813)	Compares two strings case sensitively.
	CompareStr (see page 22813)	Compares two strings case sensitively.
	CompareText (see page 22814)	Compares two strings by ordinal value without case sensitivity.
	CompareText (see page 22814)	Compares two strings by ordinal value without case sensitivity.
	CreateDir (see page 22817)	Creates a new directory.
	CreateGUID (see page 22819)	Creates a globally unique identifier.
	CurrentYear (see page 22819)	Returns the current year.
	CurrToStr (see page 22819)	Formats a Currency value as a string.
	CurrToStr (see page 22819)	Formats a Currency value as a string.
	CurrToStrF (see page 22820)	Converts a Currency value to a string, using a specified format.
	CurrToStrF (see page 22820)	Converts a Currency value to a string, using a specified format.
	Date (see page 22820)	Returns the current date.
	DateTimeToFileDate (see page 22821)	Converts a TDateTime value to an OS timestamp value.
	DateTimeToStr (see page 22822)	Converts a TDateTime value to a string.
	DateTimeToStr (see page 22822)	Converts a TDateTime value to a string.
	DateTimeToString (see page 22823)	Converts a TDateTime value to a string using a specified Format (see page 22897).
	DateTimeToString (see page 22823)	Converts a TDateTime value to a string using a specified Format (see page 22897).
	DateTimeToSystemTime (see page 22825)	Converts a TDateTime value into the Win32 API's system time type.
	DateTimeToTimeStamp (see page 22825)	Converts a TDateTime value into the corresponding TTimeStamp (see page 23009) value.
	DateToStr (see page 22825)	Converts a TDateTime value to a string.
	DateToStr (see page 22825)	Converts a TDateTime value to a string.
	DayOfWeek (see page 22827)	Returns the day of the week for a specified date.
	DecodeDate (see page 22828)	Returns Year, Month, and Day values for a TDateTime value.
	DecodeDateFully (see page 22829)	Returns Year, Month, and Day, and Day-of-Week values for a TDateTime value.
	DecodeTime (see page 22830)	Breaks a TDateTime value into hours, minutes, seconds, and milliseconds.
	DeleteFile (see page 22831)	Deletes a file from disk.
	DirectoryExists (see page 22832)	Determines whether a specified directory exists.
	DiskFree (see page 22833)	Returns the number of free bytes on a specified drive.
	DiskSize (see page 22835)	Returns the size, in bytes, of a specified drive.
	DisposeStr (see page 22836)	Disposes of a string pointer that was allocated using NewStr (see page 22929).
	EncodeDate (see page 22836)	Returns a TDateTime value that represents a specified Year, Month, and Day.
	EncodeTime (see page 22837)	Returns a TDateTime value for a specified Hour, Min, Sec, and MSec.
	ExceptionErrorMessage (see page 22839)	Formats a standard error message.
	ExcludeTrailingBackslash (see page 22839)	Returns a path name without a trailing delimiter.
	ExcludeTrailingPathDelimiter (see page 22840)	Returns a path name without a trailing delimiter.
	ExpandFileName (see page 22840)	Returns the full path name for a relative file name.
	ExpandFileNameCase (see page 22841)	Returns the full path name for a relative file name on a case-sensitive file system.
	ExpandUNCFileName (see page 22842)	Returns the full path of a file name in UNC format, if appropriate.
	ExtractFileDir (see page 22843)	Extracts the drive and directory parts from FileName.
	ExtractFileDrive (see page 22843)	Returns the drive portion of a file name.
	ExtractFileExt (see page 22844)	Returns the extension portions of a file name.
	ExtractFileName (see page 22844)	Extracts the name and extension parts of a file name.
	ExtractFilePath (see page 22846)	Returns the drive and directory portions of a file name.
	ExtractRelativePath (see page 22848)	Returns a relative path name, relative to a specific base directory.
	ExtractShortPathName (see page 22849)	Converts a file name to the short 8.3 form.
	FileAge (see page 22850)	Returns the OS timestamp of a file.
	FileAge (see page 22850)	Returns the OS timestamp of a file.
	FileClose (see page 22852)	Closes a specified file.
	FileCreate (see page 22855)	Creates a new file.
	FileCreate (see page 22855)	Creates a new file.
	FileDateToDateTime (see page 22858)	Converts an OS timestamp value to TDateTime value.

	FileExists (see page 22860)	Tests if a specified file exists.
	FileGetAttr (see page 22863)	Returns the file attributes of FileName.
	FileGetDate (see page 22866)	Returns an OS timestamp for a specified file.
	FileIsReadOnly (see page 22866)	Reports whether a file is read-only.
	FileOpen (see page 22867)	Opens a specified file using a specified access mode.
	FileRead (see page 22870)	Reads a specified number of bytes from a file.
	FileSearch (see page 22872)	Searches a specified directory path for a file.
	FileSeek (see page 22873)	Repositions read/write point.
	FileSeek (see page 22873)	Repositions read/write point.
	FileSetAttr (see page 22875)	Sets the file attributes of a specified file.
	FileSetDate (see page 22878)	Sets the OS time stamp for a specified file.
	FileSetDate (see page 22878)	Sets the OS time stamp for a specified file.
	FileSetReadOnly (see page 22878)	Sets the file permissions to read-only.
	FileWrite (see page 22879)	Writes the contents of a buffer to the current position in a file.
	FinalizePackage (see page 22882)	Finalizes the specified package (.bpl file).
	FindClose (see page 22882)	Releases memory allocated by FindFirst (see page 22884).
	FindCmdLineSwitch (see page 22884)	Determines whether a string was passed as a command line argument to the application.
	FindCmdLineSwitch (see page 22884)	Determines whether a string was passed as a command line argument to the application.
	FindCmdLineSwitch (see page 22884)	Determines whether a string was passed as a command line argument to the application.
	FindFirst (see page 22884)	Searches for the first instance of a file name with a given set of attributes in a specified directory.
	FindNext (see page 22887)	Returns the next entry matching the name and attributes specified in a previous call to FindFirst (see page 22884).
	FloatToCurr (see page 22889)	Converts a floating-point value to a Currency value.
	FloatToDateTime (see page 22889)	Converts a floating-point value to a TDateTime value.
	FloatToDecimal (see page 22890)	Converts a floating-point value to a decimal representation.
	FloatToStr (see page 22890)	Converts a floating point value to a string.
	FloatToStr (see page 22890)	Converts a floating point value to a string.
	FloatToStrF (see page 22891)	Converts a floating point value to a string, using a specified Format (see page 22897), Precision, and Digits.
	FloatToStrF (see page 22891)	Converts a floating point value to a string, using a specified Format (see page 22897), Precision, and Digits.
	FloatToText (see page 22892)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 22897), Precision and Digits.
	FloatToText (see page 22892)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 22897), Precision and Digits.
	FloatToTextFmt (see page 22893)	Converts a floating-point value to an unterminated character string, using a specified format.
	FloatToTextFmt (see page 22893)	Converts a floating-point value to an unterminated character string, using a specified format.
	FmtLoadStr (see page 22893)	Returns formatted output using a resourced format string.
	FmtStr (see page 22894)	Assembles a formatted string using a format string and an array of arguments.
	FmtStr (see page 22894)	Assembles a formatted string using a format string and an array of arguments.
	ForceDirectories (see page 22895)	Creates a new directory, including the creation of parent directories as needed.
	Format (see page 22897)	Returns a formatted string assembled from a format string and an array of arguments.
	Format (see page 22897)	Returns a formatted string assembled from a format string and an array of arguments.
	FormatBuf (see page 22900)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 22900)	Formats the arguments from an array, placing the result in a buffer.
	FormatCurr (see page 22901)	Formats a Currency object.
	FormatCurr (see page 22901)	Formats a Currency object.
	FormatDateTime (see page 22902)	Formats a TDateTime value.
	FormatDateTime (see page 22902)	Formats a TDateTime value.
	FormatFloat (see page 22904)	Formats a floating point value.

	FormatFloat (see page 22904)	Formats a floating point value.
	FreeAndNil (see page 22907)	Frees an object reference and replaces the reference with nil (Delphi) or NULL (C++).
	GDAL (see page 22909)	Get Delphi Access Licence.
	GetCurrentDir (see page 22910)	Returns the name of the current directory.
	GetEnvironmentVariable (see page 22911)	Retrieves an environment variable value.
	GetFileVersion (see page 22912)	Returns file version.
	GetFormatSettings (see page 22912)	Resets the date and number format parameters to initial values.
	GetLocaleChar (see page 22913)	Retrieves the first character of a localisation value.
	GetLocaleFormatSettings (see page 22913)	Populates a TFormatSettings (see page 23001) data structure.
	GetLocaleStr (see page 22913)	Retrieves a localisation value.
	GetModuleName (see page 22914)	Returns the fully qualified name for a module, given its handle.
	GetPackageDescription (see page 22915)	Returns the description resource from a package.
	GetPackageInfo (see page 22915)	Enumerates all the units and required packages for a package.
	GetTime (see page 22916)	Returns the current time.
	GUIDToString (see page 22916)	Converts a class identifier GUID to a string.
	IncAMonth (see page 22917)	Increments date data by one month.
	IncludeTrailingBackslash (see page 22918)	Ensures path name ends with delimiter
	IncludeTrailingPathDelimiter (see page 22918)	Ensures path name ends with delimiter.
	IncMonth (see page 22919)	Returns a date shifted by a specified number of months.
	InitializePackage (see page 22920)	Validates and initializes a given package.
	InitializePackage (see page 22920)	Validates and initializes a given package.
	IntToHex (see page 22920)	Returns the hex representation of an integer.
	IntToHex (see page 22920)	Returns the hex representation of an integer.
	IntToStr (see page 22921)	Converts an integer to a string.
	IntToStr (see page 22921)	Converts an integer to a string.
	IsAssembly (see page 22922)	Determines whether a dll is an Assembly.
	IsDelimiter (see page 22923)	Indicates whether a specified character in a string matches one of a set of delimiters.
	IsEqualGUID (see page 22923)	Indicates whether two TGUID values are the same.
	IsLeapYear (see page 22924)	Indicates whether a specified year is a leap year.
	IsPathDelimiter (see page 22925)	Indicates whether the byte at a specified position in a string is the path delimiter.
	IsValidIdent (see page 22925)	Determines whether a string conforms to Delphi identifier rules.
	Languages (see page 22926)	Lists the locales for which support is available.
	LastDelimiter (see page 22926)	Returns the byte index of the last character that matches any character in a specified set of delimiters.
	LoadPackage (see page 22927)	Loads a specified package.
	LoadPackage (see page 22927)	Loads a specified package.
	LoadStr (see page 22927)	Loads a string resource from the application's executable file.
	LowerCase (see page 22928)	Converts an ASCII string to lowercase.
	LowerCase (see page 22928)	Converts an ASCII string to lowercase.
	MSecsToTimeStamp (see page 22929)	Converts a specified number of milliseconds into a TTimeStamp (see page 23009) value.
	NewStr (see page 22929)	Allocates a string on the heap.
	NextCharIndex (see page 22930)	Returns the byte index of the next character.
	Now (see page 22930)	Returns the current date and time.
	OutOfMemoryError (see page 22931)	Raises an EOutOfMemory (see page 22742) exception.
	QuotedStr (see page 22931)	Returns the quoted version of a string.
	RaiseLastOSError (see page 22932)	Raises an exception for the last occurring OS or system library error.
	RaiseLastOSError (see page 22932)	Raises an exception for the last occurring OS or system library error.
	RaiseLastWin32Error (see page 22932)	Raises an exception for the last occurring Win32 error.
	RCS (see page 22933)	Check for Delphi Client/Server Licence.
	RemoveDir (see page 22933)	Deletes an existing empty directory.
	RenameFile (see page 22934)	Changes a file name.
	ReplaceDate (see page 22936)	Replaces the date portion of a TDateTime value with a specified date.
	ReplaceTime (see page 22937)	Replaces the time portion of a TDateTime value with a specified time.

	RPR (see page 22938)	Check for Delphi Professional Licence.
	SafeLoadLibrary (see page 22938)	Safe and simple library loading.
	SameFileName (see page 22939)	Compares file names based on the current locale.
	SameStr (see page 22939)	Compares two strings by ordinal value with case sensitivity.
	SameStr (see page 22939)	Compares two strings by ordinal value with case sensitivity.
	SameText (see page 22940)	Compares two strings by ordinal value without case sensitivity.
	SameText (see page 22940)	Compares two strings by ordinal value without case sensitivity.
	SetCurrentDir (see page 22940)	Sets the current directory.
	ShowException (see page 22941)	Displays an exception message with its physical address
	Sleep (see page 22941)	Delay program execution for a specified number of milliseconds.
	StrAlloc (see page 22942)	Allocates a buffer for a null-terminated string and returns a pointer to its first character.
	StrBufSize (see page 22942)	Returns the maximum number of characters that can be stored in a buffer allocated by StrAlloc (see page 22942).
	StrByteType (see page 22943)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	StrCat (see page 22943)	Appends a copy of Source to the end of Dest and returns the concatenated string.
	StrCharLength (see page 22944)	Returns the length, in bytes, of a character.
	StrComp (see page 22945)	Compares two strings with case sensitivity.
	StrCopy (see page 22946)	Copies a null-terminated string.
	StrDispose (see page 22947)	Disposes of a string.
	StrECopy (see page 22948)	Copies null-terminated string.
	StrEnd (see page 22949)	Returns a pointer to the end of a null terminated string.
	StrFmt (see page 22950)	Formats entries in an array.
	StrFmt (see page 22950)	Formats entries in an array.
	StrIComp (see page 22951)	Compares two strings without case sensitivity.
	StringReplace (see page 22951)	Returns a string with occurrences of one substring replaced by another substring.
	StringToGUID (see page 22952)	Converts a string to a GUID.
	StrLCat (see page 22952)	Appends up to a specified maximum number of characters to string.
	StrLComp (see page 22953)	Compares up to a specified maximum number of characters in two strings.
	StrLCopy (see page 22953)	Copies up to a specified maximum number of characters from Source to Dest.
	StrLen (see page 22954)	Returns number of characters in a string excluding the null terminator.
	StrLFmt (see page 22954)	Formats a series of arguments from a specified open array into a buffer.
	StrLFmt (see page 22954)	Formats a series of arguments from a specified open array into a buffer.
	StrLIComp (see page 22955)	Compares strings up to a specified maximum number of characters, without case sensitivity.
	StrLower (see page 22956)	Converts a string to lowercase.
	StrMove (see page 22956)	Copies specified number of characters to string.
	StrNew (see page 22957)	Allocates space on and copies a string to the heap; returning a pointer to the string.
	StrNextChar (see page 22958)	Returns pointer to following character.
	StrPas (see page 22958)	Converts null-terminated string to an AnsiString (long string).
	StrPCopy (see page 22958)	Copies an AnsiString (long string) to a null-terminated string.
	StrPLCopy (see page 22959)	Copies characters from an AnsiString (long string) into a null-terminated string.
	StrPos (see page 22959)	Returns a pointer to the first occurrence of STR2 in STR1.
	StrRScan (see page 22960)	Returns a pointer to the last occurrence of a specified character in a string.
	StrScan (see page 22960)	Returns a pointer to first occurrence of a specified character in a string.
	StrToBool (see page 22961)	Converts a string to a Boolean value.
	StrToBoolDef (see page 22961)	Converts a string to a Boolean value, with error default.
	StrToCurr (see page 22962)	Converts a string to a Currency value.
	StrToCurr (see page 22962)	Converts a string to a Currency value.
	StrToCurrDef (see page 22963)	Converts a string to a Currency value, with error default.
	StrToCurrDef (see page 22963)	Converts a string to a Currency value, with error default.
	StrToDate (see page 22963)	Converts a string to a TDateTime value.
	StrToDate (see page 22963)	Converts a string to a TDateTime value.
	StrToDateDef (see page 22965)	Converts a string to a TDateTime value, with error default.

	StrToDateDef (see page 22965)	Converts a string to a TDateTime value, with error default.
	StrToDateTime (see page 22965)	Converts a string to a TDateTime value.
	StrToDateTime (see page 22965)	Converts a string to a TDateTime value.
	StrToDateTimeDef (see page 22966)	Converts a string to a TDateTime value with error default.
	StrToDateTimeDef (see page 22966)	Converts a string to a TDateTime value with error default.
	StrToFloat (see page 22967)	Converts a given string to a floating-point value.
	StrToFloat (see page 22967)	Converts a given string to a floating-point value.
	StrToFloatDef (see page 22969)	Converts a given string to a floating-point value with error default.
	StrToFloatDef (see page 22969)	Converts a given string to a floating-point value with error default.
	StrToInt (see page 22970)	Converts a string that represents an integer (decimal or hex notation) to a number.
	StrToInt64 (see page 22970)	Converts a string that represents an integer (decimal or hex notation) to a number.
	StrToInt64Def (see page 22971)	Converts a string that represents an integer (decimal or hex notation) to a number.
	StrToIntDef (see page 22971)	Converts a string that represents an integer (decimal or hex notation) to a number with error default.
	StrToTime (see page 22972)	Converts a string to a TDateTime value.
	StrToTime (see page 22972)	Converts a string to a TDateTime value.
	StrToTimeDef (see page 22972)	Converts a string to a TDateTime value with error default.
	StrToTimeDef (see page 22972)	Converts a string to a TDateTime value with error default.
	StrUpper (see page 22973)	Returns a string in upper case.
	Supports (see page 22973)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 22973)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 22973)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 22973)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 22973)	Indicates whether a given object or interface supports a specified interface.
	SysErrorMessage (see page 22974)	Converts OS error codes into strings.
	SystemTimeToDateTime (see page 22974)	Converts a system time value into a TDateTime value.
	TextToFloat (see page 22975)	Converts a null-terminated string to a floating-point value.
	TextToFloat (see page 22975)	Converts a null-terminated string to a floating-point value.
	Time (see page 22976)	Returns the current time.
	TimeStampToDateTime (see page 22976)	Converts a TTimeStamp (see page 23009) value into the corresponding TDateTime value.
	TimeStampToMSecs (see page 22977)	Converts a TTimeStamp (see page 23009) value into an absolute number of milliseconds.
	TimeToStr (see page 22977)	Returns a string that represents a TDateTime value.
	TimeToStr (see page 22977)	Returns a string that represents a TDateTime value.
	Trim (see page 22978)	Trims leading and trailing spaces and control characters from a string.
	Trim (see page 22978)	Trims leading and trailing spaces and control characters from a string.
	TrimLeft (see page 22979)	Trims leading spaces and control characters from a string.
	TrimLeft (see page 22979)	Trims leading spaces and control characters from a string.
	TrimRight (see page 22979)	Trims trailing spaces and control characters from a string.
	TrimRight (see page 22979)	Trims trailing spaces and control characters from a string.
	TryEncodeDate (see page 22980)	Creates a TDateTime value that represents a specified Year, Month, and Day.
	TryEncodeTime (see page 22980)	Creates a TDateTime value for a specified Hour, Min, Sec, and MSec.
	TryFloatToCurr (see page 22981)	Converts a floating-point value to a Currency value.
	TryFloatToDateTime (see page 22982)	Converts a floating-point value to a TDateTime value.
	TryStrToBool (see page 22982)	Converts a string to a Boolean value, with a Boolean success code.
	TryStrToCurr (see page 22983)	Converts a string to a Currency value, with a Boolean success code.
	TryStrToCurr (see page 22983)	Converts a string to a Currency value, with a Boolean success code.
	TryStrToDate (see page 22983)	Converts a string to a TDateTime value, with a Boolean success code.
	TryStrToDate (see page 22983)	Converts a string to a TDateTime value, with a Boolean success code.
	TryStrToDateTime (see page 22984)	Converts a string to a TDateTime value with a Boolean success code.
	TryStrToDateTime (see page 22984)	Converts a string to a TDateTime value with a Boolean success code.
	TryStrToFloat (see page 22985)	Converts a given string to a floating-point value with a boolean success code.

	TryStrToFloat (see page 22985)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 22985)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 22985)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 22985)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 22985)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToInt (see page 22986)	Converts a string that represents an integer (decimal or hex notation) to a number with Boolean success code.
	TryStrToInt64 (see page 22986)	Converts a string that represents an integer (decimal or hex notation) to a number.
	TryStrToTime (see page 22987)	Converts a string to a TDateTime value with an error default.
	TryStrToTime (see page 22987)	Converts a string to a TDateTime value with an error default.
	UnloadPackage (see page 22987)	Unloads a package.
	UpperCase (see page 22988)	Returns a copy of a string in uppercase.
	UpperCase (see page 22988)	Returns a copy of a string in uppercase.
	WideCompareStr (see page 22989)	Compares Unicode strings based on the current locale with case sensitivity.
	WideCompareText (see page 22990)	Compares Unicode strings based on the current locale without case sensitivity.
	WideFmtStr (see page 22990)	Assembles a formatted Unicode string using a format string and an array of arguments.
	WideFmtStr (see page 22990)	Assembles a formatted Unicode string using a format string and an array of arguments.
	WideFormat (see page 22991)	Returns a formatted Unicode string assembled from a format string and an array of arguments.
	WideFormat (see page 22991)	Returns a formatted Unicode string assembled from a format string and an array of arguments.
	WideFormatBuf (see page 22991)	Formats the arguments from an array, placing the result in a buffer.
	WideFormatBuf (see page 22991)	Formats the arguments from an array, placing the result in a buffer.
	WideLowerCase (see page 22992)	Returns Unicode string converted to lower case.
	WideSameStr (see page 22992)	Compares Unicode strings based on the current locale with case sensitivity.
	WideSameText (see page 22993)	Compares Unicode strings based on the current locale without case sensitivity.
	WideUpperCase (see page 22993)	Returns Unicode string converted to upper case.
	Win32Check (see page 22994)	Checks the return value of a Windows API call and raises an appropriate exception when it indicates failure.
	WrapText (see page 22994)	Splits a string into multiple lines as its length approaches a specified size.
	WrapText (see page 22994)	Splits a string into multiple lines as its length approaches a specified size.

159.3.1 SysUtils.Abort Function

Raises a silent exception.

Pascal

```
procedure Abort;
```

C++

```
Abort();
```

File

SysUtils

Description

Use Abort to escape from an execution path without reporting an error.

Abort raises a special "silent exception" (EAbort (see page 22722)), which operates like any other exception, but does not display an error message to the end user. Abort redirects execution to the end of the last exception block.

Delphi Examples:

```
{
This example checks for a valid entry in a TDBEdit control
and calls the Abort procedure if the control is empty; Abort
cancels the post before it happens.
}
procedure TForm1.ClientDataSet1BeforePost(DataSet: TDataSet);
begin
  if DBEdit1.Text = '' then
    Abort;
  DataSet.Fields[2].AsString := DBEdit1.Text;
end;
```

See Also

EAbort (see page 22722)

159.3.2 SysUtils.AddExitProc Function

Adds a procedure to the exit list.

Pascal

```
procedure AddExitProc(Proc: TProcedure);
```

C++

```
AddExitProc(TProcedure Proc);
```

File

SysUtils

Description

AddExitProc adds Proc to the Delphi exit procedure list, so that Proc is automatically executed when the application terminates. AddExitProc is not compatible with CLX package support and is provided for backward compatibility only. Do not use AddExitProc in new applications.

159.3.3 SysUtils.AddTerminateProc Function

Adds a terminate procedure to the system list of termination procedures.

Pascal

```
procedure AddTerminateProc(TermProc: TTerminateProc);
```

C++

```
AddTerminateProc(TTerminateProc TermProc);
```

File

SysUtils

Description

Call AddTerminateProc in Delphi code to add a procedure that is called when the application is about to terminate. The run time library calls all termination procedures before the application terminates. The supplied termination procedure (TermProc) should return true if the application can safely terminate or false if the application cannot safely terminate. If any termination procedure returns false, the application does not terminate.

See Also

CallTerminateProcs (see page 22808)

159.3.4 SysUtils.AdjustLineBreaks Function

Adjusts line break characters to reflect Windows or Linux style.

Pascal

```
function AdjustLineBreaks(const S: string; Style: TTextLineBreakStyle = tlbsCRLF): string;
```

C++

```
AnsiString AdjustLineBreaks(const AnsiString S, TTextLineBreakStyle Style = tlbsCRLF);
```

File

SysUtils

Description

AdjustLineBreaks returns a text string with its line breaks adjusted to fit a specified style.

S is the string to convert.

Style indicates whether line breaks should all be Windows-style line breaks with a carriage return and linefeed (tlbsCRLF), or Linux-style breaks with a single linefeed character (tlbsLF). If Style is omitted, the style for the compile target is used.

See Also

TrimLeft (see page 22979)

TrimRight (see page 22979)

159.3.5 SysUtils.AnsiCompareFileName Function

Compares file names based on the current locale.

Pascal

```
function AnsiCompareFileName(const S1: string; const S2: string): Integer;
```

C++

```
int AnsiCompareFileName(const AnsiString S1, const AnsiString S2);
```

File

SysUtils

Description

AnsiCompareFileName compares two file names. The current locale controls the compare operation. Comparison is case-insensitive under Windows, case-sensitive under Linux.

The return value is:

Condition	Return Value
S1 > S2	> 0

S1 < S2	< 0
S1 = S2	= 0

AnsiCompareFileName supports the idiosyncrasies of MS-DOS file names that use Zenkaku Moji, a character encoding used in some Asian locales. Under Windows locales that do not use MBCS, AnsiCompareFileName is identical to AnsiCompareText (see page 22790). Under Linux, AnsiCompareFileName is identical to AnsiCompareStr (see page 22789).

See Also

SameFileName (see page 22939)

AnsiCompareText (see page 22790)

159.3.6 SysUtils.AnsiCompareStr Function

Compares strings based on the current locale with case sensitivity.

Pascal

```
function AnsiCompareStr(const S1: string; const S2: string): Integer;
```

C++

```
int AnsiCompareStr(const AnsiString S1, const AnsiString S2);
```

File

SysUtils

Description

AnsiCompareStr compares S1 to S2, with case sensitivity. The compare operation is controlled by the current locale. The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

Note: Most locales consider lowercase characters to be less than the corresponding uppercase characters. This is in contrast to ASCII order, in which lowercase characters are greater than uppercase characters. Thus, setting S1 to 'a' and S2 to 'A' causees AnsiCompareStr to return a value less than zero, while CompareStr (see page 22813), with the same arguments, returns a value greater than zero.

See Also

AnsiCompareText (see page 22790)

AnsiStrComp (see page 22795)

AnsiSameStr (see page 22794)

159.3.7 SysUtils.AnsiCompareText Function

Compares strings based on the current locale without case sensitivity.

Pascal

```
function AnsiCompareText(const S1: string; const S2: string): Integer;
```

C++

```
int AnsiCompareText(const AnsiString S1, const AnsiString S2);
```

File

SysUtils

Description

AnsiCompareText compares S1 to S2, without case sensitivity. The compare operation is controlled by the current locale. AnsiCompareText returns a value less than 0 if S1 < S2, a value greater than 0 if S1 > S2, and returns 0 if S1 = S2.

See Also

AnsiCompareStr (see page 22789)

AnsiStrIComp (see page 22795)

AnsiSameText (see page 22794)

159.3.8 SysUtils.AnsiDequotedStr Function

Converts a quoted string into an unquoted string

Pascal

```
function AnsiDequotedStr(const S: string; AQuote: Char): string;
```

C++

```
AnsiString AnsiDequotedStr(const AnsiString S, Char AQuote);
```

File

SysUtils

Description

AnsiDequotedStr removes the quote characters from the beginning and end of a quoted string, and reduces pairs of quote characters within the string to a single quote character. The AQuote parameter defines what character to use as a quote character. If the S parameter does not begin and end with the quote character, AnsiDequotedStr returns S unchanged.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiQuotedStr (see page 22793)

AnsiExtractQuotedStr (see page 22791)

159.3.9 SysUtils.AnsiExtractQuotedStr Function

Converts a quoted string into an unquoted string.

Pascal

```
function AnsiExtractQuotedStr(var Src: PChar; Quote: Char): string;
```

C++

```
AnsiString AnsiExtractQuotedStr(const char * Src, Char Quote);
```

File

SysUtils

Description

AnsiExtractQuotedStr removes the quote characters from the beginning and end of a quoted string, and reduces pairs of quote characters within the string to a single quote character. The Quote parameter defines what character to use as a quote character. If the first character in Src is not the value of the Quote parameter, AnsiExtractQuotedStr returns an empty string.

The function copies characters from Src to the result string until the second solitary quote character or the first null character in Src. The Src parameter is updated to point to the first character following the quoted string. If Src does not contain a matching end quote character, the Src parameter is updated to point to the terminating null character.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiQuotedStr (see page 22793)

159.3.10 SysUtils.AnsiLastChar Function

Returns a pointer to the last character in a string.

Pascal

```
function AnsiLastChar(const S: string): PChar;
```

C++

```
const char * AnsiLastChar(const AnsiString S);
```

File

SysUtils

Description

Call AnsiLastChar to obtain the last character in a string. This function supports multi-byte character sets (MBCS).

See Also

AnsiStrLastChar (see page 22796)

AnsiStrRScan (see page 22799)

159.3.11 SysUtils.AnsiLowerCase Function

Returns a string that is a copy of the given string converted to lower case.

Pascal

```
function AnsiLowerCase(const S: string): string;
```

C++

```
AnsiString AnsiLowerCase(const AnsiString S);
```

File

SysUtils

Description

AnsiLowerCase returns a string that is a copy of the given string converted to lower case. The conversion uses the current locale. This function supports multi-byte character sets (MBCS).

See Also

AnsiLowerCaseFileName (see page 22792)

AnsiUpperCase (see page 22800)

LowerCase (see page 22928)

159.3.12 SysUtils.AnsiLowerCaseFileName Function

Converts a file name to lower case.

Pascal

```
function AnsiLowerCaseFileName(const S: string): string;
```

C++

```
AnsiString AnsiLowerCaseFileName(const AnsiString S);
```

File

SysUtils

Description

AnsiLowerCaseFileName returns a string that is a copy of S converted to lower case. The conversion uses the current locale.

Note: On Windows, AnsiLowerCaseFileName supports lowercase conversion idiosyncrasies of file names in Asian locales (Zenkaku).

On Linux, or in locales that do not use multi byte character systems (MBCS), AnsiLowerCaseFileName is identical to AnsiLowerCase (see page 22792).

See Also

AnsiLowerCase (see page 22792)

AnsiUpperCaseFileName (see page 22801)

159.3.13 SysUtils.AnsiPos Function

Locates the position of a substring.

Pascal

```
function AnsiPos(const Substr: string; const S: string): Integer;
```

C++

```
int AnsiPos(const AnsiString Substr, const AnsiString S);
```

File

SysUtils

Description

Call AnsiPos to obtain the byte offset of the Substr parameter, as it appears in the string S. For example, if Substr is the string "AB", and S is the string "ABCDE", AnsiPos returns 1. If Substr does not appear in S, AnsiPos returns 0.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiStrPos (see page 22798)

159.3.14 SysUtils.AnsiQuotedStr Function

Returns the quoted version of a string.

Pascal

```
function AnsiQuotedStr(const S: string; Quote: Char): string;
```

C++

```
AnsiString AnsiQuotedStr(const AnsiString S, Char Quote);
```

File

SysUtils

Description

Use AnsiQuotedStr to convert a string (S) to a quoted string, using the provided Quote character. A Quote character is inserted at the beginning and end of S, and each Quote character in the string is doubled.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiExtractQuotedStr (see page 22791)

QuotedStr (see page 22931)

159.3.15 SysUtils.AnsiSameStr Function

Compares strings based on the current locale with case sensitivity.

Pascal

```
function AnsiSameStr(const S1: string; const S2: string): Boolean;
```

C++

```
Boolean AnsiSameStr(const AnsiString S1, const AnsiString S2);
```

File

SysUtils

Description

AnsiSameStr compares S1 to S2, with case sensitivity. The compare operation is controlled by the current locale. The return value is true if the two strings have the same value, false otherwise.

See Also

AnsiSameText (see page 22794)

AnsiCompareStr (see page 22789)

AnsiSameCaption (see page 16186)

159.3.16 SysUtils.AnsiSameText Function

Compares strings based on the current locale without case sensitivity.

Pascal

```
function AnsiSameText(const S1: string; const S2: string): Boolean;
```

C++

```
Boolean AnsiSameText(const AnsiString S1, const AnsiString S2);
```

File

SysUtils

Description

AnsiSameText compares S1 to S2, without case sensitivity. The compare operation is controlled by the current locale. The return value is true if the two strings have the same value, false otherwise.

See Also

AnsiSameStr (see page 22794)

AnsiCompareText (see page 22790)

AnsiSameCaption (see page 16186)

159.3.17 SysUtils.AnsiStrComp Function

Compares null-terminated character strings.

Pascal

```
function AnsiStrComp(S1: PChar; S2: PChar): Integer;
```

C++

```
int AnsiStrComp(const char * S1, const char * S2);
```

File

SysUtils

Description

AnsiStrComp compares S1 to S2, with case sensitivity. The compare operation is controlled by the current locale. The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

Note: Most locales consider lowercase characters to be less than the corresponding uppercase characters. This is in contrast to ASCII order, in which lowercase characters are greater than uppercase characters.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiCompareStr (see page 22789)

AnsiStrIComp (see page 22795)

AnsiStrLComp (see page 22796)

159.3.18 SysUtils.AnsiStrIComp Function

Compares null terminated character strings case insensitively.

Pascal

```
function AnsiStrIComp(S1: PChar; S2: PChar): Integer;
```

C++

```
int AnsiStrIComp(const char * S1, const char * S2);
```

File

SysUtils

Description

AnsiStrIComp compares S1 to S2, without case sensitivity. The compare operation is controlled by the current locale. The return value is the same as for CompareStr (see page 22813).

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiCompareText (see page 22790)

AnsiStrComp (see page 22795)

159.3.19 SysUtils.AnsiStrLastChar Function

Returns a pointer to the last character in a null-terminated string.

Pascal

```
function AnsiStrLastChar(P: PChar): PChar;
```

C++

```
const char * AnsiStrLastChar(const char * P);
```

File

SysUtils

Description

Call AnsiStrLastChar to obtain a pointer to the last full character in a null-terminated sequence. When working with multi-byte character sets (MBCS), the last character will not be the same as the last char value if the last two bytes represent a single double byte character.

See Also

AnsiLastChar (see page 22791)

159.3.20 SysUtils.AnsiStrLComp Function

Compares the first MaxLen bytes of two null-terminated strings, case-sensitively.

Pascal

```
function AnsiStrLComp(S1: PChar; S2: PChar; MaxLen: Cardinal): Integer;
```

C++

```
int AnsiStrLComp(const char * S1, const char * S2, unsigned MaxLen);
```

File

SysUtils

Description

AnsiStrLComp compares S1 to S2, with case sensitivity. If S1 or S2 is longer than MaxLen bytes, AnsiStrLComp only compares the first MaxLen bytes. The compare operation is controlled by the current locale. The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2 (up to MaxLen characters)	= 0

Note: This function supports multi-byte character sets (MBCS).

Note: AnsiStrLComp can continue the comparison beyond a null terminating character if S1 and S2 are equal up to a null character and the null character occurs before MaxLen bytes.

Note: Most locales consider lowercase characters to come before corresponding uppercase characters. This is in contrast to ASCII order, in which lowercase characters come after uppercase characters.

See Also

AnsiStrComp (see page 22795)

AnsiStrLIComp (see page 22797)

StrLComp (see page 22953)

159.3.21 SysUtils.AnsiStrLIComp Function

Compares two strings, case-insensitively, up to the first MaxLen bytes.

Pascal

```
function AnsiStrLIComp(S1: PChar; S2: PChar; MaxLen: Cardinal): Integer;
```

C++

```
int AnsiStrLIComp(const char * S1, const char * S2, unsigned MaxLen);
```

File

SysUtils

Description

AnsiStrLIComp compares S1 to S2, without case sensitivity. If S1 or S2 is longer than MaxLen characters, AnsiStrLIComp only compares up to the first MaxLen characters. The compare operation is controlled by the current locale. The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2 (up to MaxLen characters)	= 0

Note: This function supports multi-byte character sets (MBCS).

Note: AnsiStrLIComp can continue the comparison beyond a null terminating character if S1 and S2 are equal up to a null character and the null character occurs before MaxLen bytes.

See Also

AnsiStrIComp (see page 22795)

AnsiStrLComp (see page 22796)

StrLIComp (see page 22955)

159.3.22 SysUtils.AnsiStrLower Function

Converts all characters in a null-terminated string to lower case.

Pascal

```
function AnsiStrLower(Str: PChar): PChar;
```

C++

```
const char * AnsiStrLower(const char * Str);
```

File

SysUtils

Description

AnsiStrLower returns a null-terminated character sequence where all characters are lower case. The conversion uses the current locale.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiLowerCase (see page 22792)

AnsiLowerCaseFileName (see page 22792)

AnsiStrUpper (see page 22800)

159.3.23 SysUtils.AnsiStrPos Function

Returns a pointer to the first occurrence of a substring.

Pascal

```
function AnsiStrPos(Str: PChar; SubStr: PChar): PChar;
```

C++

```
const char * AnsiStrPos(const char * Str, const char * SubStr);
```

File

SysUtils

Description

Call AnsiStrPos to locate the first occurrence of SubStr in Str. If SubStr does not occur in Str, AnsiStrPos returns nil (Delphi) or NULL (C++).

Unlike the StrPos (see page 22959) function, AnsiStrPos works with multi-byte characters sets (MBCS).

See Also

AnsiPos (see page 22793)

StrPos (see page 22959)

159.3.24 SysUtils.AnsiStrRScan Function

Returns a pointer to the last occurrence of a specified character in a specified string.

Pascal

```
function AnsiStrRScan(Str: PChar; Chr: Char): PChar;
```

C++

```
const char * AnsiStrRScan(const char * Str, Char Chr);
```

File

SysUtils

Description

Call AnsiStrRScan to obtain a pointer to the last occurrence of Chr in Str. If Chr does not occur in Str, AnsiStrRScan returns nil (Delphi) or NULL (C++). The null terminator is considered to be part of the string.

Unlike the StrRScan (see page 22960) function, AnsiStrRScan works with multi-byte character systems (MBCS).

See Also

AnsiStrScan (see page 22799)

StrRScan (see page 22960)

159.3.25 SysUtils.AnsiStrScan Function

Returns a pointer to first occurrence of a character in a string.

Pascal

```
function AnsiStrScan(Str: PChar; Chr: Char): PChar;
```

C++

```
const char * AnsiStrScan(const char * Str, Char Chr);
```

File

SysUtils

Description

Call AnsiStrScan to obtain a pointer to the first occurrence of Chr in Str. If Chr does not occur in Str, AnsiStrScan returns nil (Delphi) or NULL (C++). The null terminator is considered to be part of the string.

Unlike the StrScan (see page 22960) function, AnsiStrScan works with multi-byte character systems (MBCS).

See Also

AnsiStrRScan (see page 22799)

StrScan (see page 22960)

159.3.26 SysUtils.AnsiStrUpper Function

Converts all characters in a null-terminated string to upper case.

Pascal

```
function AnsiStrUpper(Str: PChar): PChar;
```

C++

```
const char * AnsiStrUpper(const char * Str);
```

File

SysUtils

Description

AnsiStrUpper returns a null-terminated string where all characters are upper case. The conversion uses the current locale.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiUpperCase (see page 22800)

AnsiUpperCaseFileName (see page 22801)

AnsiStrLower (see page 22798)

159.3.27 SysUtils.AnsiUpperCase Function

Converts a string to upper case.

Pascal

```
function AnsiUpperCase(const S: string): string;
```

C++

```
AnsiString AnsiUpperCase(const AnsiString S);
```

File

SysUtils

Description

AnsiUpperCase returns a string that is a copy of S, converted to upper case. The conversion uses the current locale.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiLowerCase (see page 22792)

UpperCase (see page 22988)

159.3.28 SysUtils.AnsiUpperCaseFileName Function

Converts a file name to upper case.

Pascal

```
function AnsiUpperCaseFileName(const S: string): string;
```

C++

```
AnsiString AnsiUpperCaseFileName(const AnsiString S);
```

File

SysUtils

Description

AnsiUpperCaseFileName returns a string that is a copy of S converted to upper case. The conversion uses the current locale.

Note: On Windows, AnsiUpperCaseFileName supports uppercase conversion idiosyncrasies of file names in Asian locales (Zenkaku).

On Linux or in locales that do not use multi byte character systems (MBCS), AnsiUpperCaseFileName is identical to AnsiUpperCase (see page 22800).

See Also

AnsiLowerCaseFileName (see page 22792)

AnsiUpperCase (see page 22800)

159.3.29 SysUtils.AppendStr Function

Appends a dynamically allocated string to an existing string.

Pascal

```
procedure AppendStr(var Dest: string; const S: string); deprecated;
```

C++

```
AppendStr(AnsiString Dest, const AnsiString S);
```

File

SysUtils

Description

AppendStr appends S to the end of Dest. It is provided for backwards compatibility only. Use the + operator instead:

Dest := Dest + S;

Dest = Dest + S;

159.3.30 SysUtils.AssignStr Function

Assigns a new dynamically allocated string to a specified pointer.

Pascal

```
procedure AssignStr(var P: PString; const S: string); deprecated;
```

C++

```
AssignStr(PString P, const AnsiString S);
```

File

SysUtils

Description

Do not use AssignStr in new applications. It is provided for backward compatibility only.

See Also

DisposeStr (see page 22836)

159.3.31 SysUtils.Beep Function

Emits a message beep.

Pascal

```
procedure Beep;
```

C++

```
Beep();
```

File

SysUtils

Description

Beep generates a conventional message beep.

Note: On Windows, Beep calls the Windows API MessageBeep.

Note: On Linux, to minimize library dependencies in the run time library, the SysUtils (see page 22709) version of Beep has limited Linux functionality. For cross-platform GUI applications, use the QControls version of Beep instead.

C++ Examples:

```
/*
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
```

```
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FindDialog1->Position = Point(RichEdit1->Left + RichEdit1->Width, RichEdit1->Top);
  FindDialog1->Execute();
}

void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  int FoundAt, StartPos, ToEnd;
  TSearchTypes mySearchTypes;
  // begin the search after the current selection
  // if there is one
  // otherwise, begin at the start of the text
  if (FindDialog1->Options.Contains(frMatchCase))
    mySearchTypes << stMatchCase;
  if (FindDialog1->Options.Contains(frWholeWord))
    mySearchTypes << stWholeWord;
  if (RichEdit1->SelLength)
    StartPos = RichEdit1->SelStart + RichEdit1->SelLength;
  else
    StartPos = 0;
  // ToEnd is the length from StartPos
  // to the end of the text in the rich edit control
  ToEnd = RichEdit1->Text.Length() - StartPos;
  FoundAt = RichEdit1->FindText(FindDialog1->FindText, StartPos, ToEnd, mySearchTypes);
  if (FoundAt != -1)
  {
    RichEdit1->SetFocus();
    RichEdit1->SelStart = FoundAt;
    RichEdit1->SelLength = FindDialog1->FindText.Length();
  }
  else Beep();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("../Overview.RTF");  // starts in Debug
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FindDialog1.Position :=
    Point(RichEdit1.Left + RichEdit1.Width, RichEdit1.Top);
  FindDialog1.Execute;
end;

procedure TForm1.FindDialog1Find(Sender: TObject);
var
  FoundAt: LongInt;
  StartPos, ToEnd: Integer;
  mySearchTypes : TSearchTypes;
  myFindOptions : TFindOptions;
begin
  with RichEdit1 do
```

```
    begin
      if frMatchCase in FindDialog1.Options then
         mySearchTypes := mySearchTypes + [stMatchCase];
      if frWholeWord in FindDialog1.Options then
         mySearchTypes := mySearchTypes + [stWholeWord];
      { begin the search after the current selection if there is one }
      { otherwise, begin at the start of the text }
      if SelLength <> 0 then
        StartPos := SelStart + SelLength
      else
        StartPos := 0;
      { ToEnd is the length from StartPos to the end of the
        text in the rich edit control }
      ToEnd := Length(Text) - StartPos;
      FoundAt :=
        FindText(FindDialog1.FindText, StartPos, ToEnd, mySearchTypes);
      if FoundAt <> -1 then
      begin
        SetFocus;
        SelStart := FoundAt;
        SelLength := Length(FindDialog1.FindText);
      end
      else Beep;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const Path = 'OverView.RTF';
  begin
    RichEdit1.PlainText := False;
    RichEdit1.Lines.LoadFromFile(Path);
    RichEdit1.ScrollBars := ssVertical;
  end;
  {
  The following example beeps once for each "beep" passed in
  on the command line.  The example terminates the application
  if  "exit" is passed in on the command line.  Build the
  project to generate a .exe file and then execute that file
  from a command line: "ParamCount_proj beep beep exit".
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
  //  Use print statements since you can't use the debugger.
  //  MessageDlg(
  //    'System ParamCount = ' + IntToStr(System.ParamCount),
  //    mtConfirmation, [mbOK], 0);
    for i := 1 to System.ParamCount do
    begin
      if LowerCase(ParamStr(i)) = 'beep' then
        Beep
      else if LowerCase(ParamStr(i)) = 'exit' then
        Application.Terminate;
      Sleep(250);
    end;
  end;
```

159.3.32 SysUtils.BoolToStr Function

Converts a Boolean value to a string.

Pascal

```
function BoolToStr(B: Boolean; UseBoolStrs: Boolean = False): string;
```

C++

```
AnsiString BoolToStr(Boolean B, Boolean UseBoolStrs = False);
```

File

SysUtils

Description

BoolToStr converts the given Boolean value to a string as follows:

Value of B	Value of UseBoolStrs	Value of returned string
true	false	'-1'
true	true	The first string in TrueBoolStrs (see page 23034) (default 'True');
false	false	'0'
false	true	The first string in FalseBoolStrs (see page 23025) (default 'False');

Note: The TrueBoolStrs (see page 23034) and FalseBoolStrs (see page 23025) variables are undefined when an application starts. Any use of StrToBool (see page 22961), BoolToStr or TryStrToBool (see page 22982) will cause these variables to be initialised with the default values above.

C++ Examples:

```
/*
This example requires that a memo and a TApplicationEvent be
added to the form.  Change the Windows Locale by using the
Regional and Language Options in the Windows Control Panel.
*/
void __fastcall TForm1::ApplicationEvents1SettingChange(TObject *Sender, int Flag, const
AnsiString Section, int &Result)
{
  Memo1->Lines->BeginUpdate();
  try
  {
//    char buffer[64];
    AnsiString buffer;
    buffer.sprintf("Section = %s", Section.c_str());
    Memo1->Lines->Add(buffer);
    buffer.sprintf("Flag = %8x", Flag);
    Memo1->Lines->Add(buffer);
    if (strcmp(Section.c_str(), "intl") == 0)
    {
      buffer.sprintf("DefaultLCID = %8x", SysLocale.DefaultLCID);
      Memo1->Lines->Add(buffer);
      buffer.sprintf("PriLangID = %8x", SysLocale.PriLangID);
      Memo1->Lines->Add(buffer);
      buffer.sprintf("SubLangID = %8x", SysLocale.SubLangID);
      Memo1->Lines->Add(buffer);
      buffer.sprintf("FarEast = %s", BoolToStr(SysLocale.FarEast, true));
      Memo1->Lines->Add(buffer);
      buffer.sprintf("MiddleEast = %s", BoolToStr(SysLocale.MiddleEast, true));
      Memo1->Lines->Add(buffer);
      Memo1->Lines->Add("");
    }
  }
  __finally
  {
```

```
    Memo1->Lines->EndUpdate();
  }
}
```

Delphi Examples:

```
{
This example requires that a memo and a TApplicationEvent be
added to the form.  Change the Windows Locale by using the
Regional and Language Options in the Windows Control Panel.
}
procedure TForm1.ApplicationEvents1SettingChange(Sender: TObject; Flag: Integer; const
Section: string; var Result: Integer);
begin
  Memo1.Lines.BeginUpdate;
  try
    Memo1.Lines.Add(Format('Section  = %s', [Section]));
    Memo1.Lines.Add(Format('Flags  = %.8x', [Flag]));
    if AnsiSameStr(Section, 'intl') then
    with SysLocale do
    begin
      Memo1.Lines.Add(Format('DefaultLCID  = %.8x', [DefaultLCID]));
      Memo1.Lines.Add(Format('PriLangID  = %.8x', [PriLangID]));
      Memo1.Lines.Add(Format('SubLangID  = %.8x', [SubLangID]));
      Memo1.Lines.Add(Format('FarEast  = %s', [BoolToStr(FarEast, True)]));
      Memo1.Lines.Add(Format('MiddleEast  = %s', [BoolToStr(MiddleEast, True)]));
    end;
    Memo1.Lines.Add('');
  finally
    Memo1.Lines.EndUpdate;
  end;
//  Result:= 0;
end;
```

See Also

StrToBool (see page 22961)

TryStrToBool (see page 22982)

TrueBoolStrs (see page 23034)

FalseBoolStrs (see page 23025)

159.3.33 SysUtils.ByteToCharIndex Function

Returns the position of the character that contains a specified byte in a string.

Pascal

```
function ByteToCharIndex(const S: string; Index: Integer): Integer;
```

C++

```
int ByteToCharIndex(const AnsiString S, int Index);
```

File

SysUtils

Description

Call ByteToCharIndex to determine which character contains a particular byte in a string, where 1 specifies the first byte, 2 specifies the second byte, and so on. If the Index parameter specifies a byte that is not in the AnsiString (Index < 0 or Index > Length(S)), ByteToCharIndex returns 0.

If the system does not use a multi-byte character system (MBCS), ByteToCharIndex simply returns the value of Index, because there is a 1:1 correspondence between bytes and characters. Otherwise, ByteToCharIndex returns the position of the character, where 1 specifies the first character, 2 specifies the second character, and so on.

For example, if an AnsiString contains two characters, a single-byte character followed by a double byte character, then ByteToCharIndex returns 1 when Index is 1, it returns 2 when Index is 2 or 3, and it returns 0 for any other value of Index.

See Also

ByteToCharLen (see page 22807)

CharToByteIndex (see page 22811)

159.3.34 SysUtils.ByteToCharLen Function

Returns the number of characters fully contained in the first MaxLen bytes of a string.

Pascal

```
function ByteToCharLen(const S: string; MaxLen: Integer): Integer;
```

C++

```
int ByteToCharLen(const AnsiString S, int MaxLen);
```

File

SysUtils

Description

Call ByteToCharLen to determine the number of characters represented by a portion of a string. ByteToCharLen counts characters until it reaches MaxLen bytes or the end of the string, whichever comes first.

When working with multi-byte character systems (MBCS), the value returned by ByteToCharLen may be less than MaxLen or the length of the string.

Note: ByteToCharLen will count a partial character if it appears as the last of the first MaxLen bytes.

See Also

ByteToCharIndex (see page 22806)

CharToByteIndex (see page 22811)

CharToByteLen (see page 22811)

159.3.35 SysUtils.ByteType Function

Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.

Pascal

```
function ByteType(const S: string; Index: Integer): TMbcsByteType;
```

C++

```
TMbcsByteType ByteType(const AnsiString S, int Index);
```

File

SysUtils

Description

Call ByteType to determine whether a specified byte in a string is a single-byte character, the first byte of a multibyte character, or one of the trailing bytes.

AnsiString is the string that contains the byte in question.

Index identifies the byte for which you want to know the byte type. Bytes are numbered from 1.

If the system is not using a multi-byte character system (MBCS), ByteType always returns mbSingleByte. Otherwise, ByteType returns mbSingleByte if the indicated byte represents a complete character in S, mbLeadByte if it represents the first byte of a double byte character, and mbTrailByte if it represents a subsequent byte of a multibyte character.

Note: No checking is done to ensure that Index is less than the length of S. It is the caller's responsibility to ensure that Index is not out of bounds.

See Also

LeadBytes (see page 23026)

StrByteType (see page 22943)

159.3.36 SysUtils.CallTerminateProcs Function

Calls all of the functions in the termination procedure list.

Pascal

```
function CallTerminateProcs: Boolean;
```

C++

```
Boolean CallTerminateProcs();
```

File

SysUtils

Description

CallTerminateProcs is called internally when an application is about to terminate. It returns true only if all of the functions in the application's terminate procedure list return true. Do not call this function directly. Its use is automatic in Delphi application.

See Also

AddTerminateProc (see page 22787)

159.3.37 SysUtils.ChangeFileExt Function

Changes the extension of a file name.

Pascal

```
function ChangeFileExt(const FileName: string; const Extension: string): string;
```

C++

```
AnsiString ChangeFileExt(const AnsiString FileName, const AnsiString Extension);
```

File

SysUtils

Description

ChangeFileExt takes the file name passed in FileName and changes the extension of the file name to the extension passed in Extension. Extension specifies the new extension, including the initial dot character.

ChangeFileExt does not rename the actual file, it just creates a new file name string.

Note: This function works with multi-byte character sets (MBCS).

C++ Examples:

```
/*
This example converts a specified icon to a bitmap. To run
this example, add an image, a button, and an Open dialog to
a form. Name the button ConvertIcon2Bitmap, and add the
following code as its OnClick event handler.
*/
void __fastcall TForm1::ConvertIcon2BitmapClick(TObject *Sender)
{
  OpenDialog1->DefaultExt = ".ICO";
  OpenDialog1->Filter = "icons (*.ico)|*.ICO";
  OpenDialog1->Options << ofOverwritePrompt << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TIcon *pi = new TIcon();
    try
    {
      AnsiString as;
      pi->LoadFromFile(OpenDialog1->FileName);
      as = ChangeFileExt(OpenDialog1->FileName,".BMP");
      Image1->Width = pi->Width;
      Image1->Height = pi->Height;
      Image1->Canvas->Draw(0,0,pi);
      Image1->Picture->SaveToFile(as);
      ShowMessage(OpenDialog1->FileName + " Saved to " + as);
    }
    __finally
    {
      delete pi;
    }
  }
}
```

Delphi Examples:

```
{
This example converts a specified icon to a bitmap. To run
this example, add an image, a button, and an Open dialog to
a form. Name the button ConvertIcon2Bitmap, and add the
following code as its OnClick event handler.
}
procedure TForm1.ConvertIcon2BitmapClick(Sender: TObject);
var
  s : string;
  Icon: TIcon;
begin
  OpenDialog1.DefaultExt := '.ICO';
```

```
    OpenDialog1.Filter := 'icons (*.ico)|*.ICO';
    OpenDialog1.Options := [ofOverwritePrompt, ofFileMustExist, ofHideReadOnly ];
    if OpenDialog1.Execute then
    begin
      Icon := TIcon.Create;
      try
        Icon.Loadfromfile(OpenDialog1.FileName);
        s:= ChangeFileExt(OpenDialog1.FileName,'.BMP');
        Image1.Width := Icon.Width;
        Image1.Height := Icon.Height;
        Image1.Canvas.Draw(0,0,Icon);
        Image1.Picture.SaveToFile(s);
        ShowMessage(OpenDialog1.FileName + ' Saved to ' + s);
      finally
        Icon.Free;
      end;
    end;
  end;
```

See Also

ExtractFileExt (see page 22844)

159.3.38 SysUtils.ChangeFilePath Function

Changes the path of a file name.

Pascal

```
function ChangeFilePath(const FileName: string; const Path: string): string;
```

C++

```
AnsiString ChangeFilePath(const AnsiString FileName, const AnsiString Path);
```

File

SysUtils

Description

ChangeFilePath replaces the drive and directory parts of the passed FileName with the passed Path value. If Path is not blank, it must specify a drive (terminated by a colon) and/or a directory (terminated by a backslash).

ChangeFilePath does not rename the actual file, it just creates a new file name string.

Note: This function works with multi-byte character sets (MBCS).

See Also

ExtractFilePath (see page 22846)

159.3.39 SysUtils.CharLength Function

Returns number of bytes used by a character.

Pascal

```
function CharLength(const S: String; Index: Integer): Integer;
```

C++

```
int CharLength(const AnsiString S, int Index);
```

File

SysUtils

Description

Call CharLength to determine the size in bytes of the character located in S at Index.

If the system is not using a multi-byte character system (MBCS), CharLength always returns 1.

159.3.40 SysUtils.CharToByteIndex Function

Returns the 1-based index of the first byte of a specified character in a string.

Pascal

```
function CharToByteIndex(const S: string; Index: Integer): Integer;
```

C++

```
int CharToByteIndex(const AnsiString S, int Index);
```

File

SysUtils

Description

Call CharToByteIndex to obtain the byte offset for the character at a specified position in the string S, where 1 specifies the first character, 2 specifies the second character, and so on. CharToByteIndex returns zero if Index is out of range (Index <= 0 or S contains fewer than Index characters). If the Index parameter indicates a double-byte character, CharToByteIndex returns the offset of the first byte.

If the system is not using a multi-byte character system (MBCS), CharToByteIndex returns the value of Index, because there is a 1:1 correspondence between bytes and characters.

See Also

ByteToCharIndex (see page 22806)

CharToByteLen (see page 22811)

159.3.41 SysUtils.CharToByteLen Function

Returns the length, in bytes, of the first MaxLen characters of a string.

Pascal

```
function CharToByteLen(const S: string; MaxLen: Integer): Integer;
```

C++

```
int CharToByteLen(const AnsiString S, int MaxLen);
```

File

SysUtils

Description

Call CharToByteLen to convert the length of a string in characters to the corresponding length in bytes. CharToByteLen returns the number of bytes required by the first MaxLen characters of S. If S contains fewer than MaxLen characters, CharToByteLen returns the byte length of S.

If the system is not using a multi-byte character system (MBCS), CharToByteLen returns the length of S or MaxLen, whichever is smaller.

See Also

ByteToCharLen (see page 22807)

CharToByteIndex (see page 22811)

159.3.42 SysUtils.CheckWin32Version Function

Check to see if you are running on a specific level (or higher) of the Windows 32 bit Operating System (see page 22079).

Pascal

```
function CheckWin32Version(AMajor: Integer; AMinor: Integer = 0): Boolean;
```

C++

```
Boolean CheckWin32Version(int AMajor, int AMinor = 0);
```

File

SysUtils

Description

The Windows Operating System (see page 22079) Major.Minor version number is compared with the passed AMajor and AMinor values. AMinor defaults to 0 if not supplied.

CheckWin32Version Returns true if the Windows OS Major.Minor version number is >= the passed AMajor.AMinor value.

Note: Prior to Delphi 7, this routine returned a different result.

See Also

Win32MajorVersion (see page 23036)Win32MinorVersion (see page 23036)

159.3.43 SysUtils.CompareMem Function

Performs a binary comparison of two memory images.

Pascal

```
function CompareMem(P1: Pointer; P2: Pointer; Length: Integer): Boolean;
```

C++

```
Boolean CompareMem(void * P1, void * P2, int Length);
```

File

SysUtils

Description

CompareMem performs a binary compare of Length bytes of memory referenced by P1 to that of P2. CompareMem returns true if the memory referenced by P1 is identical to that of P2.

See Also

CompareStr (see page 22813)

CompareText (see page 22814)

CollectionsEqual (see page 3220)

159.3.44 SysUtils.CompareStr Function

Compares two strings case sensitively.

Pascal

```
function CompareStr(const S1: string; const S2: string): Integer; overload;
function CompareStr(const S1: string; const S2: string; LocaleOptions: TLocaleOptions):
Integer; overload;
```

C++

```
int CompareStr(const AnsiString S1, const AnsiString S2);
int CompareStr(const AnsiString S1, const AnsiString S2, TLocaleOptions LocaleOptions);
```

File

SysUtils

Description

CompareStr compares S1 to S2, with case-sensitivity. The return value is less than 0 if S1 is less than S2, 0 if S1 equals S2, or greater than 0 if S1 is greater than S2. The compare operation is based on the 8-bit ordinal value of each character and is not affected by the current locale.

C++ Examples:

```
/*
The following code compares String1, 'STEVE', to String2,
'STEVe'. Note that CompareStr returns a number less than 0
because the value of 'e' is greater than the value of 'E'.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString String1 = "STEVE";
  AnsiString String2 = "Steve";
  int I = CompareStr(String1, String2);  // the value of I is < 0
  Edit1->Text = IntToStr(I);
  if (I != 0)
    ShowMessage ("The strings are not equal");
}
```

Delphi Examples:

```
{
The following code compares String1, 'STEVE', to String2,
'STEVe'. Note that CompareStr returns a number less than 0
because the value of 'e' is greater than the value of 'E'.
}
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
    String1, String2 : string;
    I : Integer;
  begin
    String1 := 'STEVE';
    String2 := 'STEVe';
    I := SysUtils.CompareStr(String1, String2);  { the value of I is < 0 }
    Edit1.Text := InttoStr(I);
    if I <> 0 then
      MessageDlg('The strings are not equal', mtWarning, [mbOK], 0)
  end;
```

See Also

CompareText (see page 22814)

AnsiCompareStr (see page 22789)

159.3.45 SysUtils.CompareText Function

Compares two strings by ordinal value without case sensitivity.

Pascal

```
function CompareText(const S1: string; const S2: string): Integer; overload;
function CompareText(const S1: string; const S2: string; LocaleOptions: TLocaleOptions):
Integer; overload;
```

C++

```
int CompareText(const AnsiString S1, const AnsiString S2);
int CompareText(const AnsiString S1, const AnsiString S2, TLocaleOptions LocaleOptions);
```

File

SysUtils

Description

CompareText compares S1 and S2 and returns 0 if they are equal. If S1 is greater than S2, CompareText returns an integer greater than 0. If S1 is less than S2, CompareText returns an integer less than 0. CompareText is not case sensitive and is not affected by the current locale.

C++ Examples:

```
/*
The following code orders a list view in reverse
alphabetical order on the click of a button. The callback
function CustomSortProc calls the global CompareText
function and negates its return value.
*/
int __stdcall CustomSortProc(long Item1, long Item2, long ParamSort)
{
  return -CompareText(((TListItem *)Item1)->Caption,
                      ((TListItem *)Item2)->Caption);
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ListView1->CustomSort(CustomSortProc, 0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *ListItem;
```

```
  ListView1->ViewStyle = vsList;
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Apples";
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Oranges";
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Pears";
}
/*
This example shows how to use the OnColumnClick and
OnCompare events of a list view to let users sort the
columns in a report-style list view by clicking on the
column headers. This requires a global variable to keep
track of the column that was clicked.  The OnColumnClick
event handler sets the global variable to indicate the
column to sort and calls AlphaSort:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {
    "C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {
    "Documentation for the C++ icon.",
    "Borland icon.", "Delphi icon."
    };
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  ListView1->ViewStyle = vsReport;
}

int ColumnToSort = 0;

/*
The OnColumnClick event handler sets the global variable to
indicate the column to sort and calls AlphaSort:
*/

void __fastcall TForm1::ListView1ColumnClick(TObject *Sender,
      TListColumn *Column)
{
  ColumnToSort = Column->Index;
  ((TCustomListView *)Sender)->AlphaSort();
}

/*
The OnCompare event handler causes the list view to sort on
the selected column:
*/
```

```
void __fastcall TForm1::ListView1Compare(TObject *Sender, TListItem *Item1,
      TListItem *Item2, int Data, int &Compare)
{
  if (ColumnToSort == 0)
    Compare = CompareText(Item1->Caption,Item2->Caption);
  else
  {
    int ix = ColumnToSort - 1;
    Compare =
      CompareText(Item1->SubItems->Strings[ix], Item2->SubItems->Strings[ix]);
  }
}
```

Delphi Examples:

```
{
The following code orders a list view in reverse
alphabetical order on the click of a button. The callback
function CustomSortProc calls the global CompareText
function and negates its return value.
}
function CustomSortProc(Item1, Item2: TListItem; ParamSort: integer): integer; stdcall;
begin
  Result := -CompareText(Item1.Caption,Item2.Caption);
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListView1.CustomSort(@CustomSortProc, 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
var ListItem : TListItem;
begin
  ListView1.ViewStyle := vsList;
  ListItem := ListView1.Items.Add;
  ListItem.Caption := 'Apples';
  ListItem := ListView1.Items.Add;
  ListItem.Caption := 'Oranges';
  ListItem := ListView1.Items.Add;
  ListItem.Caption := 'Pears';
end;
{
This example shows how to use the OnColumnClick and
OnCompare events of a list view to let users sort the
columns in a report-style list view by clicking on the
column headers. This requires a global variable to keep
track of the column that was clicked.  The OnColumnClick
event handler sets the global variable to indicate the
column to sort and calls AlphaSort:
}
var ColumnToSort: Integer;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
const
  imagenames: Array[0..2] of string = ('C++ Image', 'Borland Image', 'Delphi Image');
begin
  Col2Array[0]:= 'Documentation for the C++ icon.';
  Col2Array[1]:= 'Borland icon.';
  Col2Array[2]:= 'Delphi icon.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
```

```
      LargeImages := ImageList1;
      for I := 0 to ImageList1.Count - 1 do
      begin
        ListItem := Items.Add;
        Listitem.Caption := imagenames[I];
        ListItem.ImageIndex := I;
        Listitem.SubItems.Add(Col2Array[I]);
      end;
      // Create two columns to show during viewing as vsReport
      NewColumn := Columns.Add;
      NewColumn.Caption := 'Column 1';
      NewColumn.Width := 200;
      NewColumn := Columns.Add;
      NewColumn.Caption := 'Column 2';
      NewColumn.Width := 200;
      ListView1.ViewStyle := vsReport;
    end;
  end;

  procedure TForm1.ListView1ColumnClick(Sender: TObject; Column: TListColumn);
  begin
    ColumnToSort := Column.Index;
    (Sender as TCustomListView).AlphaSort;
  end;

  {
  The OnCompare event handler causes the list view to sort on
  the selected column:
  }
  procedure TForm1.ListView1Compare(Sender: TObject; Item1, Item2: TListItem; Data: Integer; var
  Compare: Integer);
  var
    ix: Integer;
  begin
    if ColumnToSort = 0 then
      Compare := CompareText(Item1.Caption,Item2.Caption)
    else begin
     ix := ColumnToSort - 1;
     Compare := CompareText(Item1.SubItems[ix],Item2.SubItems[ix]);
    end;
  end;
  {
  Note:   This OnCompare event handler uses the global
  CompareText function. An application may want to use
  AnsiCompareText, CompareStr, or AnsiCompareStr instead,
  depending on whether the comparison should be case-sensitive
  and whether the locale should be considered.
  }
```

See Also

CompareStr (see page 22813)

AnsiCompareText (see page 22790)

CompareMem (see page 22812)

SameText (see page 22940)

159.3.46 SysUtils.CreateDir Function

Creates a new directory.

Pascal

```
function CreateDir(const Dir: string): Boolean;
```

C++

```
Boolean CreateDir(const AnsiString Dir);
```

File

SysUtils

Description

CreateDir creates a new directory. The return value is true if a new directory was successfully created, or false if an error occurred.

C++ Examples:

```
/*
The following example creates a directory 'C:\temp' if it
does not already exist.
}
*/
#include <Filectrl.hpp>
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (!DirectoryExists("c:\\temp"))
  {
    if (!CreateDir("C:\\temp"))
      throw Exception("Cannot create c:\\temp directory.");
  }
  if (!CreateDir("C:\\temp"))
    throw Exception("The c:\\temp directory already exists.");
}
```

Delphi Examples:

```
{
The following example creates a directory 'C:\temp' if it
does not already exist.
}
uses FileCtrl;
procedure TForm1.Button1Click(Sender: TObject);
begin
  if not SysUtils.DirectoryExists('C:\temp') then
    if not CreateDir('C:\temp') then
      raise Exception.Create('Cannot create C:\temp');
  if not CreateDir('C:\temp') then
    raise Exception.Create('Cannot create C:\temp again!');
end;
```

See Also

ForceDirectories (see page 22895)

ExtractFileDir (see page 22843)

GetCurrentDir (see page 22910)

RemoveDir (see page 22933)

SetCurrentDir (see page 22940)

ForceDirectories (see page 22895)

159.3.47 SysUtils.CreateGUID Function

Creates a globally unique identifier.

Pascal

```
function CreateGUID(out Guid: TGUID): HResult; stdcall;
```

C++

```
__stdcall HRESULT CreateGUID(TGUID Guid);
```

File

SysUtils

Description

CreateGUID sets Guid to a new-created Globally Unique Identifier.

Note: Under Windows, CreateGUID calls the Windows API CoCreateGUID.

Note: Under Linux, CreateGUID calls uuid_generate_time.

159.3.48 SysUtils.CurrentYear Function

Returns the current year.

Pascal

```
function CurrentYear: Word;
```

C++

```
Word CurrentYear();
```

File

SysUtils

Description

CurrentYear returns the current year on the Gregorian calendar. The value returned is the complete four-digit year.

159.3.49 SysUtils.CurrToStr Function

Formats a Currency value as a string.

Pascal

```
function CurrToStr(Value: Currency): string; overload;
function CurrToStr(Value: Currency; const FormatSettings: TFormatSettings): string; overload;
```

C++

```
AnsiString CurrToStr(Currency Value);
AnsiString CurrToStr(Currency Value, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

CurrToStr converts the Currency value given by Value to its string representation. The conversion uses the ffGeneral floating point conversion rules.

The first form of CurrToStr is not thread-safe, because it uses localization information contained in global variables. The second form of CurrToStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of CurrToStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

159.3.50 SysUtils.CurrToStrF Function

Converts a Currency value to a string, using a specified format.

Pascal

```
function CurrToStrF(Value: Currency; Format: TFloatFormat; Digits: Integer): string; overload;
function CurrToStrF(Value: Currency; Format: TFloatFormat; Digits: Integer; const
FormatSettings: TFormatSettings): string; overload;
```

C++

```
AnsiString CurrToStrF(Currency Value, TFloatFormat Format, int Digits);
AnsiString CurrToStrF(Currency Value, TFloatFormat Format, int Digits, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

CurrToStrF converts the Currency value given by Value to its string representation using the indicated float format convention. A call to CurrToStrF corresponds to a call to FloatToStrF (see page 22891) with an implied precision of 19 digits.

The first form of CurrToStrF is not thread-safe, because it uses localization information contained in global variables. The second form of CurrToStrF, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of CurrToStrF, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

159.3.51 SysUtils.Date Function

Returns the current date.

Pascal

```
function Date: TDateTime;
```

C++

```
TDateTime Date();
```

File

SysUtils

Description

Use Date to obtain the current local date as a TDateTime value. The time portion of the value is 0 (midnight).

C++ Examples:

```
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date is displayed in the
caption of the label:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = DateToStr(Date());
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, the current date is displayed in the
caption of the label:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := 'Today is  ' + SysUtils.DateToStr(Date);
end;
```

See Also

DateToStr (see page 22825)

DecodeDate (see page 22828)

Now (see page 22930)

Time (see page 22976)

Today (see page 6694)

Tomorrow (see page 6695)

Yesterday (see page 6716)

IsToday (see page 6651)

159.3.52 SysUtils.DateTimeToFileDate Function

Converts a TDateTime value to an OS timestamp value.

Pascal

```
function DateTimeToFileDate(DateTime: TDateTime): Integer;
```

C++

```
int DateTimeToFileDate(TDateTime DateTime);
```

File

SysUtils

Description

A timestamp is a signed 32-bit integer, used by the Operating System (see page 22079) to record information such as the date

and time a file was modified. The precise format of a timestamp depends on the OS. Use DateTimeToFileDate to convert a TDateTime value to an OS timestamp value.

DateTimeToFileDate is particularly useful for converting the Time (see page 22976) field of the TSearchRec (see page 23005) type used by the FindFirst (see page 22884) and FindNext (see page 22887) functions.

See Also

FileDateToDateTime (see page 22858)

FindFirst (see page 22884)

FindNext (see page 22887)

TSearchRec (see page 23005)

159.3.53 SysUtils.DateTimeToStr Function

Converts a TDateTime value to a string.

Pascal

```
function DateTimeToStr(const DateTime: TDateTime): string; overload;
function DateTimeToStr(const DateTime: TDateTime; const FormatSettings: TFormatSettings):
string; overload;
```

C++

```
AnsiString DateTimeToStr(const TDateTime DateTime);
AnsiString DateTimeToStr(const TDateTime DateTime, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

The first form of DateTimeToStr is not thread-safe, because it uses localization information contained in global variables. The second form of DateTimeToStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of DateTimeToStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

C++ Examples:

```
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date and time is
converted to a string and displayed as the caption of the
label:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = DateTimeToStr(Now());
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, the current date and time is
converted to a string and displayed as the caption of the
label:
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  Label1.Caption := SysUtils.DateTimeToStr(Now);
end;
```

159.3.54 SysUtils.DateTimeToString Function

Converts a TDateTime value to a string using a specified Format (see page 22897).

Pascal

```
procedure DateTimeToString(var Result: string; const Format: string; DateTime: TDateTime);
overload;
procedure DateTimeToString(var Result: string; const Format: string; DateTime: TDateTime;
const FormatSettings: TFormatSettings); overload;
```

C++

```
DateTimeToString(AnsiString Result, const AnsiString Format, TDateTime DateTime);
DateTimeToString(AnsiString Result, const AnsiString Format, TDateTime DateTime, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

DateTimeToString converts the TDateTime value given by DateTime using the format string given by Format (see page 22897) into the string variable given by Result. See the table below for information about the supported format strings.

The first form of DateTimeToString is not thread-safe, because it uses localization information contained in global variables. The second form of DateTimeToString, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of DateTimeToString, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

Date (see page 22820) Time (see page 22976) Format (see page 22897) Strings are composed from specifiers that represent values to be inserted into the formatted string. Some specifiers (such as "d"), simply format numbers or strings. Other specifiers (such as "/") refer to locale-specific strings from global variables.

In the following table, specifiers are given in lower case. Case is ignored in formats, except for the "am/pm" and "a/p" specifiers.

Specifier	Displays
c	Displays the date using the format given by the ShortDateFormat (see page 23030) global variable, followed by the time using the format given by the LongTimeFormat (see page 23028) global variable. The time is not displayed if the date-time value indicates midnight precisely.
d	Displays the day as a number without a leading zero (1-31).
dd	Displays the day as a number with a leading zero (01-31).
ddd	Displays the day as an abbreviation (Sun-Sat) using the strings given by the ShortDayNames (see page 23031) global variable.
dddd	Displays the day as a full name (Sunday-Saturday) using the strings given by the LongDayNames (see page 23027) global variable.
ddddd	Displays the date using the format given by the ShortDateFormat (see page 23030) global variable.
dddddd	Displays the date using the format given by the LongDateFormat (see page 23027) global variable.
e	(Windows only) Displays the year in the current period/era as a number without a leading zero (Japanese, Korean and Taiwanese locales only).

ee	(Windows only) Displays the year in the current period/era as a number with a leading zero (Japanese, Korean and Taiwanese locales only).
g	(Windows only) Displays the period/era as an abbreviation (Japanese and Taiwanese locales only).
gg	(Windows only) Displays the period/era as a full name. (Japanese and Taiwanese locales only).
m	Displays the month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mm	Displays the month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mmm	Displays the month as an abbreviation (Jan-Dec) using the strings given by the ShortMonthNames (see page 23031) global variable.
mmmm	Displays the month as a full name (January-December) using the strings given by the LongMonthNames (see page 23028) global variable.
yy	Displays the year as a two-digit number (00-99).
yyyy	Displays the year as a four-digit number (0000-9999).
h	Displays the hour without a leading zero (0-23).
hh	Displays the hour with a leading zero (00-23).
n	Displays the minute without a leading zero (0-59).
nn	Displays the minute with a leading zero (00-59).
s	Displays the second without a leading zero (0-59).
ss	Displays the second with a leading zero (00-59).
z	Displays the millisecond without a leading zero (0-999).
zzz	Displays the millisecond with a leading zero (000-999).
t	Displays the time using the format given by the ShortTimeFormat (see page 23031) global variable.
tt\	Displays the time using the format given by the LongTimeFormat (see page 23028) global variable.
am/pm	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'am' for any hour before noon, and 'pm' for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'a' for any hour before noon, and 'p' for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm	Uses the 12-hour clock for the preceding h or hh specifier, and displays the contents of the TimeAMString (see page 23033) global variable for any hour before noon, and the contents of the TimePMString (see page 23033) global variable for any hour after noon.
/	Displays the date separator character given by the DateSeparator (see page 23023) global variable.
:	Displays the time separator character given by the TimeSeparator (see page 23033) global variable.
'xx'/"xx"	Characters enclosed in single or double quotes are displayed as-is, and do not affect formatting.

See Also

DateTimeToStr (see page 22822)

GetLocaleFormatSettings (see page 22913)

159.3.55 SysUtils.DateTimeToSystemTime Function

Converts a TDateTime value into the Win32 API's system time type.

Pascal

```
procedure DateTimeToSystemTime(const DateTime: TDateTime; var SystemTime: TSystemTime);
```

C++

```
DateTimeToSystemTime(const TDateTime DateTime, TSystemTime SystemTime);
```

File

SysUtils

Description

Use DateTimeToSystemTime to convert a TDateTime value into a TSystemTime (Delphi) or _SYSTEMTIME (C++) value. This procedure is primarily used to convert TDateTime values for making a Windows API call.

See Also

SystemTimeToDateTime (see page 22974)

159.3.56 SysUtils.DateTimeToTimeStamp Function

Converts a TDateTime value into the corresponding TTimeStamp (see page 23009) value.

Pascal

```
function DateTimeToTimeStamp(DateTime: TDateTime): TTimeStamp;
```

C++

```
TTimeStamp DateTimeToTimeStamp(TDateTime DateTime);
```

File

SysUtils

Description

Call DateTimeToTimeStamp to convert a TDateTime value into a TTimeStamp (see page 23009) value.

See Also

TimeStampToDateTime (see page 22976)

159.3.57 SysUtils.DateToStr Function

Converts a TDateTime value to a string.

Pascal

```
function DateToStr(const DateTime: TDateTime): string; overload;
function DateToStr(const DateTime: TDateTime; const FormatSettings: TFormatSettings): string;
overload;
```

C++

```
AnsiString DateToStr(const TDateTime DateTime);
AnsiString DateToStr(const TDateTime DateTime, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Use DateToStr to obtain a string representation of a date value that can be used for display purposes.

The first form of DateToStr is not thread-safe, because it uses localization information contained in global variables. The second form of DateToStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of DateToStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

C++ Examples:

```
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date is displayed in the
caption of the label:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = DateToStr(Date());
}
/*
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TDateTime dt;
  try
  {
    dt = EncodeDate((Word)StrToInt(Edit1->Text), (Word)StrToInt(Edit2->Text),
(Word)StrToInt(Edit3->Text));
    Label1->Caption = DateToStr(dt);
  }
  catch(...)
  {
    ShowMessage("Invalid day, month, or year.");
  }
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, the current date is displayed in the
caption of the label:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := 'Today is  ' + SysUtils.DateToStr(Date);
end;
{
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
```

```
AnsiString that is displayed in the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  MyDate: TDateTime;
begin
  MyDate := SysUtils.EncodeDate(StrToInt(Edit1.Text), StrToInt(Edit2.Text),
StrToInt(Edit3.Text));
  Label1.Caption := DateToStr(MyDate);
end;
```

159.3.58 SysUtils.DayOfWeek Function

Returns the day of the week for a specified date.

Pascal

```
function DayOfWeek(const DateTime: TDateTime): Word;
```

C++

```
Word DayOfWeek(const TDateTime DateTime);
```

File

SysUtils

Description

DayOfWeek returns the day of the week of the specified date as an integer between 1 and 7, where Sunday is the first day of the week and Saturday is the seventh.

Note: DayOfWeek is not compliant with the ISO 8601 standard, which defines Monday as the first day of the week. For an ISO 8601 compliant version, use the DayOfTheWeek function instead.

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user enters a date in the edit box in the format
associated with the current locale (for example MM/DD/YY
format in the US), the string entered is converted to a
TDateTime value. This value is used to indicate the day of
the week the date represents.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char days[7][10] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday" };
  TDateTime dtDate = StrToDate(Edit1->Text);
  ShowMessage(Edit1->Text + AnsiString(" is a ") + days[dtDate.DayOfWeek() - 1]);
}
```

Delphi Examples:

```
{
This example uses an edit box and a button on a form. When
the user enters a date in the edit box in the format
associated with the current locale (for example MM/DD/YY
format in the US), the string entered is converted to a
TDateTime value. This value is used to indicate the day of
the week the date represents.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ADate: TDateTime;
  days: array[1..7] of string;
begin
  days[1] := 'Sunday';
  days[2] := 'Monday';
  days[3] := 'Tuesday';
  days[4] := 'Wednesday';
  days[5] := 'Thursday';
  days[6] := 'Friday';
  days[7] := 'Saturday';
  ADate := StrToDate(Edit1.Text);
  ShowMessage(Edit1.Text + ' is a ' + days[SysUtils.DayOfWeek(ADate)]);
end;
```

See Also

Date (see page 22820)

EncodeDate (see page 22836)

DayOfTheMonth (see page 6618)

DayOfTheYear (see page 6620)

DayOfTheWeek (see page 6619)

159.3.59 SysUtils.DecodeDate Function

Returns Year, Month, and Day values for a TDateTime value.

Pascal

```
procedure DecodeDate(const DateTime: TDateTime; var Year: Word; var Month: Word; var Day:
Word);
```

C++

```
DecodeDate(const TDateTime DateTime, Word Year, Word Month, Word Day);
```

File

SysUtils

Description

The DecodeDate procedure breaks the value specified as the Date (see page 22820) parameter into Year, Month, and Day values. If the given TDateTime value has a negative (BC) year, the year, month, and day return parameters are all set to zero.

C++ Examples:

```
/*
This example uses a button and two labels on a form. When
the user clicks the button, the current date and time are
reported in the captions of the two labels.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Word Year, Month, Day, Hour, Min, Sec, MSec;
  TDateTime dtPresent = Now();
  DecodeDate(dtPresent, Year, Month, Day);
  Label1->Caption = AnsiString("Today is Day ") + IntToStr(Day) + AnsiString(" of Month ") +
IntToStr(Month) + AnsiString(" of Year ") + IntToStr(Year);
  DecodeTime(dtPresent, Hour, Min, Sec, MSec);
```

```
  Label2->Caption = AnsiString("The time is Minute ") + IntToStr(Min) + AnsiString(" of Hour
") + IntToStr(Hour);
}
```

Delphi Examples:

```
{
This example uses a button and two labels on a form. When
the user clicks the button, the current date and time are
reported in the captions of the two labels.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Present: TDateTime;
  Year, Month, Day, Hour, Min, Sec, MSec: Word;
 begin
  Present:= Now;
  SysUtils.DecodeDate(Present, Year, Month, Day);
  Label1.Caption := 'Today is Day ' + IntToStr(Day) + ' of Month '
    + IntToStr(Month) + ' of Year ' + IntToStr(Year);
  SysUtils.DecodeTime(Present, Hour, Min, Sec, MSec);
  Label2.Caption := 'The time is Minute ' + IntToStr(Min) + ' of Hour '
    + IntToStr(Hour);
end;
```

See Also

DecodeDateFully (see page 22829)

DecodeTime (see page 22830)

DayOf (see page 6618)

YearOf (see page 6714)

MonthOf (see page 6670)

DecodeDateTime (see page 6626)

159.3.60 SysUtils.DecodeDateFully Function

Returns Year, Month, and Day, and Day-of-Week values for a TDateTime value.

Pascal

```
function DecodeDateFully(const DateTime: TDateTime; var Year: Word; var Month: Word; var Day:
Word; var DOW: Word): Boolean;
```

C++

```
Boolean DecodeDateFully(const TDateTime DateTime, Word Year, Word Month, Word Day, Word DOW);
```

File

SysUtils

Description

DecodeDateFully procedure breaks the value specified as the Date (see page 22820) parameter into Year, Month, Day, and DOW values. If Year is a leap year, the function return value is True; otherwise the function return value is false. If the given TDateTime value has a negative (BC) year, the return parameters are all set to zero.

See Also

DecodeDate (see page 22828)

DecodeTime (see page 22830)

DayOf (see page 6618)

YearOf (see page 6714)

MonthOf (see page 6670)

DecodeDateTime (see page 6626)

159.3.61 SysUtils.DecodeTime Function

Breaks a TDateTime value into hours, minutes, seconds, and milliseconds.

Pascal

```
procedure DecodeTime(const DateTime: TDateTime; var Hour: Word; var Min: Word; var Sec: Word;
var MSec: Word);
```

C++

```
DecodeTime(const TDateTime DateTime, Word Hour, Word Min, Word Sec, Word MSec);
```

File

SysUtils

Description

DecodeTime breaks the object specified as the Time (see page 22976) parameter into hours, minutes, seconds, and milliseconds.

C++ Examples:

```
/*
This example uses a button and two labels on a form. When
the user clicks the button, the current date and time are
reported in the captions of the two labels.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Word Year, Month, Day, Hour, Min, Sec, MSec;
  TDateTime dtPresent = Now();
  DecodeDate(dtPresent, Year, Month, Day);
  Label1->Caption = AnsiString("Today is Day ") + IntToStr(Day) + AnsiString(" of Month ") +
IntToStr(Month) + AnsiString(" of Year ") + IntToStr(Year);
  DecodeTime(dtPresent, Hour, Min, Sec, MSec);
  Label2->Caption = AnsiString("The time is Minute ") + IntToStr(Min) + AnsiString(" of Hour
") + IntToStr(Hour);
}
```

Delphi Examples:

```
{
This example uses a button and two labels on a form. When
the user clicks the button, the current date and time are
reported in the captions of the two labels.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Present: TDateTime;
  Year, Month, Day, Hour, Min, Sec, MSec: Word;
 begin
  Present:= Now;
  SysUtils.DecodeDate(Present, Year, Month, Day);
```

```
  Label1.Caption := 'Today is Day ' + IntToStr(Day) + ' of Month '
    + IntToStr(Month) + ' of Year ' + IntToStr(Year);
  SysUtils.DecodeTime(Present, Hour, Min, Sec, MSec);
  Label2.Caption := 'The time is Minute ' + IntToStr(Min) + ' of Hour '
    + IntToStr(Hour);
end;
```

See Also

DecodeDate (see page 22828)

EncodeTime (see page 22837)

Time (see page 22976)

HourOf (see page 6638)

MinuteOf (see page 6663)

SecondOf (see page 6682)

MilliSecondOf (see page 6656)

DecodeDateTime (see page 6626)

159.3.62 SysUtils.DeleteFile Function

Deletes a file from disk.

Pascal

```
function DeleteFile(const FileName: string): Boolean;
```

C++

```
Boolean DeleteFile(const AnsiString FileName);
```

File

SysUtils

Description

DeleteFile deletes the file named by FileName from the disk. If the file cannot be deleted or does not exist, the function returns false.

C++ Examples:

```
/*
The following code prompts for confirmation before deleting
a file.  Create a file and then click on the button to select
the file with a TOpenDialog.  Click Yes to delete it.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenDialog1->Title = "Delete File";
  if (OpenDialog1->Execute())
  {
    if (FileExists(OpenDialog1->FileName))
    {
      if (MessageDlg(
        ("Do you really want to delete " + ExtractFileName(OpenDialog1->FileName) + "?"),
        mtConfirmation,
        TMsgDlgButtons() << mbYes << mbNo,
        0,
```

```
        mbNo) == mrYes)
        DeleteFile(OpenDialog1->FileName);
    }
    else
      MessageDlg(("File " + ExtractFileName(OpenDialog1->FileName) + " does not exist."),
mtConfirmation, TMsgDlgButtons() << mbOK, 0);
  }
}
/*
Click the button to open a TOpenDialog, then select a file
to delete.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenDialog1->Title = "Delete File";
  if (OpenDialog1->Execute())
  {
    if (FileExists(OpenDialog1->FileName))
      DeleteFile(OpenDialog1->FileName);
  }
}
```

Delphi Examples:

```
{
The following code prompts for confirmation before deleting
a file.  Create a file and then enter the path and file name
in the textfields.  Click the button to delete it.
}
procedure TForm1.Button1Click(Sender: TObject);
var FileName: string;
begin
  Filename:= Edit1.Text;
  if SysUtils.FileExists(FileName) then
  begin
    if MessageDlg(('Do you really want to delete ' + ExtractFileName(FileName) + '?'),
mtConfirmation, [mbYes, mbNo], 0, mbNo) = IDYes then
      DeleteFile(FileName);
  end
  else
    MessageDlg(('File ' + ExtractFileName(FileName) + ' does not exist.'), mtConfirmation,
[mbOK], 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit2.Text:= ExtractFilePath(Application.ExeName);
end;
```

159.3.63 SysUtils.DirectoryExists Function

Determines whether a specified directory exists.

Pascal

```
function DirectoryExists(const Directory: string): Boolean;
```

C++

```
Boolean DirectoryExists(const AnsiString Directory);
```

File

SysUtils

Description

Call DirectoryExists to determine whether the directory specified by the **Directory** parameter exists. If the directory exists, the function returns true. If the directory does not exist, the function returns false.

If a full path name is entered, DirectoryExists searches for the directory along the designated path. Otherwise, the **Directory** parameter is interpreted as a relative path name from the current directory.

Note: The FileCtrl (see page 13311) unit (Windows only) also contains a DirectoryExists function. However, the FileCtrl (see page 13311) version is deprecated, and the SysUtils (see page 22709) version preferred, even if the code does not need to be cross-platform.

C++ Examples:

```
/*
The following example creates a directory 'C:\temp' if it
does not already exist.
}
*/
#include <Filectrl.hpp>
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (!DirectoryExists("c:\\temp"))
  {
    if (!CreateDir("C:\\temp"))
      throw Exception("Cannot create c:\\temp directory.");
  }
  if (!CreateDir("C:\\temp"))
    throw Exception("The c:\\temp directory already exists.");
}
```

Delphi Examples:

```
{
The following example creates a directory 'C:\temp' if it
does not already exist.
}
uses FileCtrl;
procedure TForm1.Button1Click(Sender: TObject);
begin
  if not SysUtils.DirectoryExists('C:\temp') then
    if not CreateDir('C:\temp') then
      raise Exception.Create('Cannot create C:\temp');
  if not CreateDir('C:\temp') then
    raise Exception.Create('Cannot create C:\temp again!');
end;
```

See Also

ForceDirectories (see page 22895)

SelectDirectory

FileExists

159.3.64 SysUtils.DiskFree Function

Returns the number of free bytes on a specified drive.

Pascal

```
function DiskFree(Drive: Byte): Int64;
```

C++

```
Int64 DiskFree(Byte Drive);
```

File

SysUtils

Description

DiskFree returns the number of free bytes on the specified drive, where 0 = Current, 1 = A, 2 = B, and so on.

DiskFree returns -1 if the drive number is invalid.

Note: DiskFree is only available on Windows.

C++ Examples:

```
/*
This example uses a form with a label on it. When the
following code executes, it displays a message in the label
indicating the number of KB free, and what percentage of the
entire disk space that represents.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  __int64 AmtFree = DiskFree(0);
  __int64 Total = DiskSize(0);
  AnsiString S;
  S.sprintf("%I64d percent of the space on drive 0 is free: %I64d KB", AmtFree*100/Total,
AmtFree/1024 );
  Label1->Caption = S;
}
```

Delphi Examples:

```
{
This example uses a form with a label on it. When the
following code executes, it displays a message in the label
indicating the number of KB free, and what percentage of the
entire disk space that represents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S: string;
  AmtFree: Int64;
  Total:   Int64;
begin
  AmtFree := SysUtils.DiskFree(0);
  Total := DiskSize(0);
  S := IntToStr((AmtFree  * 100) div Total) + ' percent of the space on drive 0 is free: ' +
IntToStr(AmtFree div 1024) + ' Kbytes free. ';
  Label1.Caption := S;
end;
{
The following example displays in a dialog the disk free
space of the current drive with commas as thousand separators.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage(FormatFloat('#,##0 bytes', DiskFree(0)));
end;
```

See Also

DiskSize (see page 22835)

159.3.65 SysUtils.DiskSize Function

Returns the size, in bytes, of a specified drive.

Pascal

```
function DiskSize(Drive: Byte): Int64;
```

C++

```
Int64 DiskSize(Byte Drive);
```

File

SysUtils

Description

DiskSize returns the size in bytes of the specified drive, where 0 = Current, 1 = A, 2 = B, etc. DiskSize returns -1 if the drive number is invalid.

Note: DiskSize is only available on Windows.

C++ Examples:

```
/*
This example uses a form with a label on it. When the
following code executes, it displays a message in the label
indicating the number of KB free, and what percentage of the
entire disk space that represents.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  __int64 AmtFree = DiskFree(0);
  __int64 Total = DiskSize(0);
  AnsiString S;
  S.sprintf("%I64d percent of the space on drive 0 is free: %I64d KB", AmtFree*100/Total,
AmtFree/1024 );
  Label1->Caption = S;
}
```

Delphi Examples:

```
{
This example uses a form with a label on it. When the
following code executes, it displays a message in the label
indicating the number of KB free, and what percentage of the
entire disk space that represents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S: string;
  AmtFree: Int64;
  Total:   Int64;
begin
  AmtFree := SysUtils.DiskFree(0);
  Total := DiskSize(0);
  S := IntToStr((AmtFree  * 100) div Total) + ' percent of the space on drive 0 is free: ' +
IntToStr(AmtFree div 1024) + ' Kbytes free. ';
  Label1.Caption := S;
end;
```

See Also

DiskFree (see page 22833)

159.3.66 SysUtils.DisposeStr Function

Disposes of a string pointer that was allocated using NewStr (see page 22929).

Pascal

```
procedure DisposeStr(P: PString); deprecated;
```

C++

```
DisposeStr(PString P);
```

File

SysUtils

Description

DisposeStr is only provided for backward compatibility with existing code that allocated a string using NewStr (see page 22929). Both DisposeStr and NewStr (see page 22929) are obsolete.

See Also

NewStr (see page 22929)

159.3.67 SysUtils.EncodeDate Function

Returns a TDateTime value that represents a specified Year, Month, and Day.

Pascal

```
function EncodeDate(Year: Word; Month: Word; Day: Word): TDateTime;
```

C++

```
TDateTime EncodeDate(Word Year, Word Month, Word Day);
```

File

SysUtils

Description

EncodeDate returns a TDateTime value from the values specified as the Year, Month, and Day parameters.

The year must be between 1 and 9999.

Valid Month values are 1 through 12.

Valid Day values are 1 through 28, 29, 30, or 31, depending on the Month value. For example, the possible Day values for month 2 (February) are 1 through 28 or 1 through 29, depending on whether or not the Year value specifies a leap year.

If the specified values are not within range, EncodeDate raises an EConvertError (see page 22724) exception.

C++ Examples:

```
/*
This example uses three edit controls, a label, and a button
```

```
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TDateTime dt;
  try
  {
    dt = EncodeDate((Word)StrToInt(Edit1->Text), (Word)StrToInt(Edit2->Text),
(Word)StrToInt(Edit3->Text));
    Label1->Caption = DateToStr(dt);
  }
  catch(...)
  {
    ShowMessage("Invalid day, month, or year.");
  }
}
```

Delphi Examples:

```
{
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  MyDate: TDateTime;
begin
  MyDate := SysUtils.EncodeDate(StrToInt(Edit1.Text), StrToInt(Edit2.Text),
StrToInt(Edit3.Text));
  Label1.Caption := DateToStr(MyDate);
end;
```

See Also

DateToStr (see page 22825)

DecodeDate (see page 22828)

EncodeTime (see page 22837)

EncodeDateTime (see page 6630)

TryEncodeDate (see page 22980)

159.3.68 SysUtils.EncodeTime Function

Returns a TDateTime value for a specified Hour, Min, Sec, and MSec.

Pascal

```
function EncodeTime(Hour: Word; Min: Word; Sec: Word; MSec: Word): TDateTime;
```

C++

```
TDateTime EncodeTime(Word Hour, Word Min, Word Sec, Word MSec);
```

File

SysUtils

Description

EncodeTime encodes the given hour, minute, second, and millisecond into a TDateTime value.

Valid Hour values are 0 through 23.

Valid Min and Sec values are 0 through 59.

Valid MSec values are 0 through 999.

If the specified values are not within range, EncodeTime raises an EConvertError (see page 22724) exception.

The resulting value is a number between 0 and 1 (inclusive) that indicates the fractional part of a day given by the specified time or (if 1.0) midnight on the following day. The value 0 corresponds to midnight, 0.5 corresponds to noon, 0.75 corresponds to 6:00 pm, and so on.

C++ Examples:

```
/*
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TDateTime dt;
  try
  {
    dt = EncodeTime((Word)StrToInt(Edit1->Text), (Word)StrToInt(Edit2->Text),
(Word)StrToInt(Edit3->Text), (Word)StrToInt(Edit4->Text));
    Label1->Caption = TimeToStr(dt);
  }
  catch(...)
  {
    ShowMessage("Invalid hour, minute, second, or millisecond.");
  }
}
```

Delphi Examples:

```
{
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  MyDate: TDateTime;
begin
  MyDate := SysUtils.EncodeTime(StrToInt(Edit1.Text), StrToInt(Edit2.Text),
StrToInt(Edit3.Text), StrToInt(Edit4.Text));
  Label1.Caption := TimeToStr(MyDate);
end;
```

See Also

DecodeTime (see page 22830)

EncodeDate (see page 22836)

EConvertError (see page 22724)

EncodeDateTime (see page 6630)

159.3.69 SysUtils.ExceptionErrorMessage Function

Formats a standard error message.

Pascal

```
function ExceptionErrorMessage(ExceptObject: TObject; ExceptAddr: Pointer; Buffer: PChar;
Size: Integer): Integer;
```

C++

```
int ExceptionErrorMessage(TObject * ExceptObject, void * ExceptAddr, const char * Buffer, int
Size);
```

File

SysUtils

Description

Call ExceptionErrorMessage to assemble an error message that indicates the object associated with the current exception and the location at which the exception was raised. The assembled message is written into the buffer provided by the Buffer parameter. Specify the size of this buffer using the Size parameter.

ExceptionErrorMessage returns the number of bytes in the assembled error message.

See Also

ExceptAddr (see page 22452)

ExceptObject (see page 22452)

ShowException (see page 22941)

159.3.70 SysUtils.ExcludeTrailingBackslash Function

Returns a path name without a trailing delimiter.

Pascal

```
function ExcludeTrailingBackslash(const S: string): string; platform;
```

C++

```
AnsiString ExcludeTrailingBackslash(const AnsiString S);
```

File

SysUtils

Description

ExcludeTrailingBackslash returns S with any trailing path delimiter ('\' on Windows, '/' on Linux) removed. This function is included for backward compatibility only. New programs should use ExcludeTrailingPathDelimiter (see page 22840).

Note: ExcludeTrailingBackslash works with multibyte character sets.

See Also

IncludeTrailingBackslash (see page 22918)

IsPathDelimiter (see page 22925)

ExcludeTrailingPathDelimiter (see page 22840)

159.3.71 SysUtils.ExcludeTrailingPathDelimiter Function

Returns a path name without a trailing delimiter.

Pascal

```
function ExcludeTrailingPathDelimiter(const S: string): string;
```

C++

```
AnsiString ExcludeTrailingPathDelimiter(const AnsiString S);
```

File

SysUtils

Description

ExcludeTrailingPathDelimiter returns S with any trailing path delimiter ('\' on Windows, '/' on Linux) removed. If the last character in S is a delimiter, the return value is S with the last character removed. Otherwise, S is returned unchanged.

Note: ExcludeTrailingPathDelimiter works with multibyte character sets.

See Also

IncludeTrailingPathDelimiter (see page 22918)

IsPathDelimiter (see page 22925)

159.3.72 SysUtils.ExpandFileName Function

Returns the full path name for a relative file name.

Pascal

```
function ExpandFileName(const FileName: string): string;
```

C++

```
AnsiString ExpandFileName(const AnsiString FileName);
```

File

SysUtils

Description

ExpandFileName converts the relative file name into a fully qualified path name. ExpandFileName does not verify that the resulting fully qualified path name refers to an existing file, or even that the resulting path exists.

Note: ExpandFileName does not support UNC format. To expand to a UNC file name, use ExpandUNCFileName (see page 22842).

C++ Examples:

```
/*
The following example uses a listbox, an edit control, and a
button on a form. When the button is clicked, the filename
```

```
specified in the edit control is expanded to a fully
qualified filename and added to the list box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ListBox1->Items->Add(ExpandFileName(Edit1->Text));
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Label2->Caption = ExtractFilePath(Application->ExeName);
}
```

Delphi Examples:

```
{
The following example uses a listbox, an edit control, and a
button on a form. When the button is clicked, the filename
specified in the edit control is expanded to a fully
qualified filename and added to the list box.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.Items.Add(SysUtils.ExpandFileName(Edit1.Text));
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Label2.Caption := ExtractFilePath(Application.ExeName);
end;
```

See Also

ExtractFileName (see page 22844)

ExpandUNCFileName (see page 22842)

ExpandFileNameCase (see page 22841)

159.3.73 SysUtils.ExpandFileNameCase Function

Returns the full path name for a relative file name on a case-sensitive file system.

Pascal

```
function ExpandFileNameCase(const FileName: string; out MatchFound: TFilenameCaseMatch):
string;
```

C++

```
AnsiString ExpandFileNameCase(const AnsiString FileName, TFilenameCaseMatch MatchFound);
```

File

SysUtils

Description

ExpandFileNameCase expands a relative file name into a fully qualified path name, then attempts to find the file or a close case-insensitive match. This is useful when converting file name data from case-insensitive conventions (Windows) to case-sensitive conventions (Linux). It is also useful for validating file names entered by users unfamiliar with Linux conventions.

FileName is the relative file name to expand.

MatchFound indicates what match, if any, was found for FileName.

The return value is the fully qualified path name generated by ExpandFileNameCase. Its relationship to FileName depends on the value of MatchFound, as indicated in the following table:

MatchFound	Return value
mkNone	The fully qualified path name, as returned by ExpandFileName (see page 22840).
mkExactMatch	The fully qualified path name for the single case-sensitive match to FileName. This is the same value as the one returned when MatchFound is mkNone.
mkSingleMatch	(Linux only.) The fully qualified path name for the single file that matches FileName case-insensitively.
mkAmbiguous	(Linux only.) The fully qualified path name for one of the case-insensitve matches to FileName. This path name is not the same as the specified file, and not necessarily the closest match.

Note: ExpandFileNameCase is much slower than ExpandFileName (see page 22840), particularly when the specified file name is ambiguous or does not match an actual file.

See Also

ExpandFileName (see page 22840)

159.3.74 SysUtils.ExpandUNCFileName Function

Returns the full path of a file name in UNC format, if appropriate.

Pascal

```
function ExpandUNCFileName(const FileName: string): string;
```

C++

```
AnsiString ExpandUNCFileName(const AnsiString FileName);
```

File

SysUtils

Description

ExpandUNCFileName returns the fully-qualified file name for a specified file name.

Note: On Windows, ExpandUNCFileName returns a fully qualified path name using Universal Naming Convention for network files. For files on local disks, ExpandUNCFileName is equivalent to ExpandFileName (see page 22840). In a UNC file name, the drive prefix is replaced with the name of the network resource to which the drive is mapped. For example, if the working directory is J:\ourdir and drive J is mapped to \\ourserver\ourshare, ExpandUNCFileName('ourfile') returns "\\ourserver\ourshare\ourdir\ourfile".

Note: On Linux, ExpandUNCFileName is always equivalent to ExpandFileName (see page 22840).

See Also

ExpandFileName (see page 22840)

ExtractFileDrive (see page 22843)

159.3.75 SysUtils.ExtractFileDir Function

Extracts the drive and directory parts from FileName.

Pascal

```
function ExtractFileDir(const FileName: string): string;
```

C++

```
AnsiString ExtractFileDir(const AnsiString FileName);
```

File

SysUtils

Description

The resulting string is a directory name suitable for passing to the CreateDir (see page 22817), GetCurrentDir (see page 22910), RemoveDir (see page 22933), and SetCurrentDir (see page 22940) functions. This string is empty if FileName contains no drive and directory parts.

Note: This function works for multi-byte character systems (MBCS).

See Also

CreateDir (see page 22817)

GetCurrentDir (see page 22910)

RemoveDir (see page 22933)

SetCurrentDir (see page 22940)

159.3.76 SysUtils.ExtractFileDrive Function

Returns the drive portion of a file name.

Pascal

```
function ExtractFileDrive(const FileName: string): string;
```

C++

```
AnsiString ExtractFileDrive(const AnsiString FileName);
```

File

SysUtils

Description

Note: On Windows, ExtractFileDrive returns a string containing the drive portion of a fully qualified path name for the file passed in the FileName. For file names with drive letters, the result is in the form '"<drive>"'. For file names with a UNC path the result is in the form '"\\<servername>\<sharename>"'. If the given path contains neither style of path prefix, the result is an empty string.

Note: On Linux, ExtractFileDrive always returns a zero-length string.

See Also

ExpandFileName (see page 22840)

ExpandUNCFileName (see page 22842)

159.3.77 SysUtils.ExtractFileExt Function

Returns the extension portions of a file name.

Pascal

```
function ExtractFileExt(const FileName: string): string;
```

C++

```
AnsiString ExtractFileExt(const AnsiString FileName);
```

File

SysUtils

Description

Use ExtractFileExt to obtain the extension from a file name. For example, the following code returns the extension of the file name specified by a variable named MyFileName:

MyFilesExtension := ExtractFileExt(MyFileName); // Delphi

MyFilesExtension = ExtractFileExt(MyFileName); // C++

The resulting string includes the period character that separates the name and extension parts. This string is empty if the given file name has no extension.

Note: This function works for multi-byte character systems (MBCS).

See Also

ExtractFileName (see page 22844)

159.3.78 SysUtils.ExtractFileName Function

Extracts the name and extension parts of a file name.

Pascal

```
function ExtractFileName(const FileName: string): string;
```

C++

```
AnsiString ExtractFileName(const AnsiString FileName);
```

File

SysUtils

Description

The resulting string is the rightmost characters of FileName, starting with the first character after the colon or backslash that separates the path information from the name and extension. The resulting string is equal to FileName if FileName contains no drive and directory parts.

Note: This function works for multi-byte character systems (MBCS).

Delphi Examples:

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      Pchar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          { Write out the length of each string, followed by the string itself. }
          StringLen := Length(StringGrid1.Cells[X,Y]);
          FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
          GetMem(Buffer, StringLen); { allocate the buffer }
          cellString := StringGrid1.Cells[X,Y];
          for I := 1 to StringLen do
             Buffer[I - 1] := cellString[I];
          FileWrite(FileHandle, Buffer^, StringLen);
        finally
          FreeMem(Buffer, StringLen);
        end;
      end;
    end;
    FileClose(FileHandle);
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
```

```
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
    Cells[1,1] := 'Object';
    Cells[2,1] := 'Pascal';
    Cells[3,1] := 'is';
    Cells[4,1] := 'excellent';
    Cells[0,2] := 'Row 2';
    Cells[1,2] := 'Delphi';
    Cells[2,2] := 'is';
    Cells[4,2] := 'RAD';
  end;
end;
```

See Also

ExpandFileName (see page 22840)

ExtractFilePath (see page 22846)

159.3.79 SysUtils.ExtractFilePath Function

Returns the drive and directory portions of a file name.

Pascal

```
function ExtractFilePath(const FileName: string): string;
```

C++

```
AnsiString ExtractFilePath(const AnsiString FileName);
```

File

SysUtils

Description

The resulting string is the leftmost characters of FileName, up to and including the colon or backslash that separates the path information from the name and extension. The resulting string is empty if FileName contains no drive and directory parts.

This function works for multi-byte character systems (MBCS).

C++ Examples:

```
/*
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Files
can be saved to and from the app directory by using relative
paths.
*/
void __fastcall TForm1::Save1Click(TObject *Sender)
{
  AnsiString NewFileName =
    ExtractFilePath(Application->ExeName) +
    ExtractFileName(Edit1->Text);
  AnsiString OldFileName =
    ExtractFilePath(Application->ExeName) +
    ExtractFileName(Edit2->Text);
  AnsiString Msg = Format("Copy %s to %s", ARRAYOFCONST((OldFileName, NewFileName)));
  if (MessageDlg(Msg, mtCustom, TMsgDlgButtons() << mbOK << mbCancel, 0) == mrOk)
```

```
  {
    TFileStream *OldFile = new TFileStream(OldFileName, fmOpenRead);
    try
    {
      TFileStream *NewFile = new TFileStream(NewFileName, fmCreate);
      try
      {
        NewFile->CopyFrom(OldFile, OldFile->Size);
      }
      __finally
      {
        FreeAndNil(&NewFile);
      }
    }
    __finally
    {
      FreeAndNil(&OldFile);
    }
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Label3->Caption = "Current directory: " + ExtractFilePath(Application->ExeName);
}
/*
The following code prompts for confirmation before deleting
a file.  Create a file and then click on the button to select
the file with a TOpenDialog.  Click Yes to delete it.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenDialog1->Title = "Delete File";
  if (OpenDialog1->Execute())
  {
    if (FileExists(OpenDialog1->FileName))
    {
      if (MessageDlg(
        ("Do you really want to delete " + ExtractFileName(OpenDialog1->FileName) + "?"),
        mtConfirmation,
        TMsgDlgButtons() << mbYes << mbNo,
        0,
        mbNo) == mrYes)
        DeleteFile(OpenDialog1->FileName);
    }
    else
      MessageDlg(("File " + ExtractFileName(OpenDialog1->FileName) + " does not exist."),
mtConfirmation, TMsgDlgButtons() << mbOK, 0);
  }
}
```

Delphi Examples:

```
{
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Files
can be saved to and from the app directory by using relative
paths.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Label3.Caption:= 'Current directory: ' + ExtractFilePath(Application.ExeName);
end;

procedure TForm1.Save1Click(Sender: TObject);
var
```

```
    NewFileName, OldFileName: string;
    Msg: string;
    NewFile: TFileStream;
    OldFile: TFileStream;
  begin
    NewFileName :=
      ExtractFilePath(Application.ExeName) +
      ExtractFileName(Edit1.Text);
    OldFileName :=
      ExtractFilePath(Application.ExeName) +
      ExtractFileName(Edit2.Text);
    Msg := Format('Copy %s to %s?', [OldFileName, NewFileName]);
    if MessageDlg(Msg, mtCustom, mbOKCancel, 0) = mrOK then
    begin
      OldFile := TFileStream.Create(
        OldFileName, fmOpenRead or fmShareDenyWrite);
      try
        NewFile := TFileStream.Create(
          NewFileName, fmCreate or fmShareDenyRead);
        try
          NewFile.CopyFrom(OldFile, OldFile.Size);
        finally
          FreeAndNil(NewFile);
        end;
      finally
        FreeAndNil(OldFile);
      end;
    end;
  end;
  {
  The following code prompts for confirmation before deleting
  a file.  Create a file and then enter the path and file name
  in the textfields.  Click the button to delete it.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var FileName: string;
  begin
    Filename:= Edit1.Text;
    if SysUtils.FileExists(FileName) then
    begin
      if MessageDlg(('Do you really want to delete ' + ExtractFileName(FileName) + '?'),
  mtConfirmation, [mbYes, mbNo], 0, mbNo) = IDYes then
        DeleteFile(FileName);
    end
    else
      MessageDlg(('File ' + ExtractFileName(FileName) + ' does not exist.'), mtConfirmation,
  [mbOK], 0);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Edit2.Text:= ExtractFilePath(Application.ExeName);
  end;
```

See Also

ExtractFileName (see page 22844)

159.3.80 SysUtils.ExtractRelativePath Function

Returns a relative path name, relative to a specific base directory.

Pascal

```
function ExtractRelativePath(const BaseName: string; const DestName: string): string;
```

C++

```
AnsiString ExtractRelativePath(const AnsiString BaseName, const AnsiString DestName);
```

File

SysUtils

Description

Call ExtractRelativePath to convert a fully qualified path name into a relative path name. The `DestName` parameter specifies file name (including path) to be converted. `BaseName` is the fully qualified name of the base directory to which the returned path name should be relative. `BaseName` may or may not include a file name, but it must include the final path delimiter.

Note: If you send it what should be a directory without the trailing backslash (c:\foo

instead of `c:\foo\`), the function treats `foo` as a file and returns undesirable results. You should correctly call it like this:

```
ExtractRelativePath('c:\foo\', 'c:\bar\')
```

so that it would return

```
'..\bar\'
```

.

```
ExtractRelativePath('c:\foo\', 'c:\bar')
```

would return the incorrect result, probably like

```
'..'
```

.

See Also

ExpandFileName (see page 22840)

159.3.81 SysUtils.ExtractShortPathName Function

Converts a file name to the short 8.3 form.

Pascal

```
function ExtractShortPathName(const FileName: string): string;
```

C++

```
AnsiString ExtractShortPathName(const AnsiString FileName);
```

File

SysUtils

Description

ExtractShortPathName converts the file name, FileName, to the short 8.3 form. ExtractShortPathName returns an empty string if the file or directory does not exist.

The first parameter, FileName, can be a full path name. For example, ExtractShortPathName would convert the full path name:

C:\Program Files\MyCompany\MyApp\MyApp.exe

to the short form:

C:\Progra~1\MyComp~1\MyApp\MyApp.exe

Note: ExtractShortPathName is only available on Windows.

See Also

ExpandFileName (see page 22840)

ExpandUNCFileName (see page 22842)

159.3.82 SysUtils.FileAge Function

Returns the OS timestamp of a file.

Pascal

```
function FileAge(const FileName: string): Integer; overload; deprecated;
function FileAge(const FileName: string; out FileDateTime: TDateTime): Boolean; overload;
```

C++

```
int FileAge(const AnsiString FileName);
Boolean FileAge(const AnsiString FileName, TDateTime FileDateTime);
```

File

SysUtils

Description

Call FileAge to obtain the OS timestamp of the file specified by FileName. The return value can be converted to a TDateTime object using the FileDateToDateTime (see page 22858) function. The return value is -1 if the file does not exist.

Note: :Under Linux, -1 is a valid timestamp. Use FileExists (see page 22860) to verify that the file does not exist.

C++ Examples:

```
/*
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings:
*/
#include "fattrdlg.h"

void __fastcall TForm1::FileListMouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{

  unsigned short  Attributes;
  unsigned short  NewAttributes;

  FileAttrDlg->FileDirName->Caption = FileList->Items->Strings[FileList->ItemIndex];
  FileAttrDlg->FilePathName->Caption = FileList->Directory;
  FileAttrDlg->ChangeDate->Caption =
DateTimeToStr(FileDateToDateTime(FileAge(FileList->FileName)));
  Attributes = FileGetAttr(FileList->Items->Strings[FileList->ItemIndex]);
  FileAttrDlg->ReadOnly->Checked = Attributes & faReadOnly;
  FileAttrDlg->Archive->Checked = Attributes & faArchive;
```

```
  FileAttrDlg->System->Checked = Attributes & faSysFile;
  FileAttrDlg->Hidden->Checked = Attributes & faHidden;
  if (FileAttrDlg->ShowModal()!= mrCancel){
     NewAttributes = Attributes;
     if (FileAttrDlg->ReadOnly->Checked)
       NewAttributes = NewAttributes | faReadOnly;
     else
        NewAttributes = NewAttributes & ~faReadOnly;

     if (FileAttrDlg->Archive->Checked)
       NewAttributes = NewAttributes | faArchive;
     else
       NewAttributes = NewAttributes & ~faArchive;

     if (FileAttrDlg->System->Checked)
       NewAttributes = NewAttributes | faSysFile;
     else
       NewAttributes = NewAttributes & ~faSysFile;

     if (FileAttrDlg->Hidden->Checked)
       NewAttributes = NewAttributes | faHidden;
     else
       NewAttributes = NewAttributes  & ~faHidden;
     if (NewAttributes != Attributes)
       FileSetAttr(FileAttrDlg->FileDirName->Caption, NewAttributes);
  }
}
```

Delphi Examples:

```
{
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings:
}
procedure TForm1.FileListMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  Attributes, NewAttributes: Word;
begin
  with FileAttrForm do
  begin
    FileDirName.Caption := FileList.Items[FileList.ItemIndex];
    { set box caption }
    FilePathName.Caption := FileList.Directory;
    { show directory name }
    ChangeDate.Caption := DateTimeToStr(FileDateToDateTime(FileAge(FileList.FileName)));
    Attributes := SysUtils.FileGetAttr(FileDirName.Caption);
    { read file attributes }
    ReadOnly.Checked := (Attributes and SysUtils.faReadOnly) = faReadOnly;
    Archive.Checked := (Attributes and faArchive) = faArchive;
    System.Checked := (Attributes and faSysFile) = faSysFile;
    Hidden.Checked := (Attributes and faHidden) = faHidden;
    if ShowModal <> id_Cancel then    { execute dialog box }
    begin
      NewAttributes := Attributes;
      { start with original attributes }
      if ReadOnly.Checked then
        NewAttributes := NewAttributes or SysUtils.faReadOnly
      else
        NewAttributes := NewAttributes and not SysUtils.faReadOnly;
      if Archive.Checked then
        NewAttributes := NewAttributes or faArchive
      else
```

```
          NewAttributes := NewAttributes and not faArchive;
        if System.Checked then
          NewAttributes := NewAttributes or faSysFile
        else
          NewAttributes := NewAttributes and not faSysFile;
        if Hidden.Checked then
          NewAttributes := NewAttributes or faHidden
        else
          NewAttributes := NewAttributes and not faHidden;
        if NewAttributes <> Attributes then { if anything changed... }
          FileSetAttr(FileDirName.Caption, NewAttributes);
           { ...write the new values }
      end;
    end;
  end;
```

See Also

FileDateToDateTime (see page 22858)

FileGetDate (see page 22866)

FileExists (see page 22860)

159.3.83 SysUtils.FileClose Function

Closes a specified file.

Pascal

```
procedure FileClose(Handle: Integer);
```

C++

```
FileClose(int Handle);
```

File

SysUtils

Description

FileClose closes a file given its handle. The handle is obtained when the file is opened using FileOpen (see page 22867) or FileCreate (see page 22855).

Note: : When working with Delphi language file variables, use the CloseFile procedure instead.

C++ Examples:

```
/*
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
*/

#include <dir.h>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
```

```
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
    if (FileExists(SaveDialog1->FileName))
    {
      fnsplit(SaveDialog1->FileName.c_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, (char*)&(StringGrid1->ColCount), sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, (char*)&(StringGrid1->RowCount), sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
    {
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(iFileHandle, (char*)&iLength, sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].c_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
    FileClose(iFileHandle);
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "Column 1";
  StringGrid1->Cells[2][0] = "Column 2";
  StringGrid1->Cells[3][0] = "Column 3";
  StringGrid1->Cells[4][0] = "Column 4";
  StringGrid1->Cells[0][1] = "Row 1";
  StringGrid1->Cells[1][1] = "Object";
  StringGrid1->Cells[2][1] = "Pascal";
  StringGrid1->Cells[3][1] = "is";
  StringGrid1->Cells[4][1] = "excellent";
  StringGrid1->Cells[0][2] = "Row 2";
  StringGrid1->Cells[1][2] = "Delphi";
  StringGrid1->Cells[2][2] = "is";
  StringGrid1->Cells[4][2] = "RAD";
};
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
```

```
    begin
      if FileExists(SaveDialog1.FileName) then
      begin
        BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
        BackupName := ChangeFileExt(BackupName, '.BAK');
        if not RenameFile(SaveDialog1.FileName, BackupName) then
          raise Exception.Create('Unable to create backup file.');
      end;
      FileHandle := FileCreate(SaveDialog1.FileName);
      { Write out the number of rows and columns in the grid. }
      colCountLength := SizeOf(StringGrid1.ColCount);
      FileWrite(FileHandle,
        Pchar(StringGrid1.ColCount), colCountLength);
      rowCountLength := SizeOf(StringGrid1.RowCount);
      FileWrite(FileHandle,
        PChar(StringGrid1.RowCount), rowCountLength);
      for X := 0 to StringGrid1.ColCount - 1 do
      begin
        for Y := 0 to StringGrid1.RowCount - 1 do
        begin
          try
            { Write out the length of each string, followed by the string itself. }
            StringLen := Length(StringGrid1.Cells[X,Y]);
            FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
            GetMem(Buffer, StringLen); { allocate the buffer }
            cellString := StringGrid1.Cells[X,Y];
            for I := 1 to StringLen do
               Buffer[I - 1] := cellString[I];
            FileWrite(FileHandle, Buffer^, StringLen);
          finally
            FreeMem(Buffer, StringLen);
          end;
        end;
      end;
      FileClose(FileHandle);
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with StringGrid1 do
    begin
      Cells[1,0] := 'Column 1';
      Cells[2,0] := 'Column 2';
      Cells[3,0] := 'Column 3';
      Cells[4,0] := 'Column 4';
      Cells[0,1] := 'Row 1';
      Cells[1,1] := 'Object';
      Cells[2,1] := 'Pascal';
      Cells[3,1] := 'is';
      Cells[4,1] := 'excellent';
      Cells[0,2] := 'Row 2';
      Cells[1,2] := 'Delphi';
      Cells[2,2] := 'is';
      Cells[4,2] := 'RAD';
    end;
  end;
  {
  This example uses a file listbox and a regular listbox on a
  form. The following routine scans through the files listed
  in the file listbox and lists the sizes of any selected
  files to the regular list box.  To exercise the error
  condition create a file in the Debug directory, start this
  application and then delete the file.  Now try to list the
  size of the deleted file.  Set the MultiSelect and
  ExtendedSelect properties on the FileListBox.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], 1);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode 1.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);

      AssignFile(F, FileListBox1.Items.Strings[i]);
      Reset(F, 1);
      ListBox1.Items.Add(
        FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
      CloseFile(F);
    end;
   finally
   { do something here }
   end;
  end;
end;
```

See Also

FileCreate (see page 22855)

FileOpen (see page 22867)

CloseFile (see page 22433)

159.3.84 SysUtils.FileCreate Function

Creates a new file.

Pascal

```
function FileCreate(const FileName: string): Integer; overload;
function FileCreate(const FileName: string; Rights: Integer): Integer; overload;
```

C++

```
int FileCreate(const AnsiString FileName);
int FileCreate(const AnsiString FileName, int Rights);
```

File

SysUtils

Description

FileCreate creates a new file with the specified name. If the return value is positive, the function was successful and the value is the file handle of the new file. A return value of -1 indicates that an error occurred.

Note: On Windows, the FileAccessRights variable and Rights parameter are ignored.

Note: On Linux, a file has a set of permission bits controlling access to the file. The first form of FileCreate takes the permission bits from the global FileAccessRights variable. The second form takes the permission bits from the Rights parameter. In either case, the permission bits are masked by the current process's umask value.

C++ Examples:

```
/*
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
*/

#include <dir.h>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
    if (FileExists(SaveDialog1->FileName))
    {
      fnsplit(SaveDialog1->FileName.c_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, (char*)&(StringGrid1->ColCount), sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, (char*)&(StringGrid1->RowCount), sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
    {
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(iFileHandle, (char*)&iLength, sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].c_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
    FileClose(iFileHandle);
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "Column 1";
  StringGrid1->Cells[2][0] = "Column 2";
  StringGrid1->Cells[3][0] = "Column 3";
```

```
  StringGrid1->Cells[4][0] = "Column 4";
  StringGrid1->Cells[0][1] = "Row 1";
  StringGrid1->Cells[1][1] = "Object";
  StringGrid1->Cells[2][1] = "Pascal";
  StringGrid1->Cells[3][1] = "is";
  StringGrid1->Cells[4][1] = "excellent";
  StringGrid1->Cells[0][2] = "Row 2";
  StringGrid1->Cells[1][2] = "Delphi";
  StringGrid1->Cells[2][2] = "is";
  StringGrid1->Cells[4][2] = "RAD";
};
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      Pchar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          { Write out the length of each string, followed by the string itself. }
          StringLen := Length(StringGrid1.Cells[X,Y]);
          FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
          GetMem(Buffer, StringLen); { allocate the buffer }
          cellString := StringGrid1.Cells[X,Y];
          for I := 1 to StringLen do
             Buffer[I - 1] := cellString[I];
          FileWrite(FileHandle, Buffer^, StringLen);
        finally
          FreeMem(Buffer, StringLen);
        end;
      end;
    end;
```

```
    FileClose(FileHandle);
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
    Cells[1,1] := 'Object';
    Cells[2,1] := 'Pascal';
    Cells[3,1] := 'is';
    Cells[4,1] := 'excellent';
    Cells[0,2] := 'Row 2';
    Cells[1,2] := 'Delphi';
    Cells[2,2] := 'is';
    Cells[4,2] := 'RAD';
  end;
end;
```

See Also

FileClose (see page 22852)

FileOpen (see page 22867)

AssignFile (see page 22420)

159.3.85 SysUtils.FileDateToDateTime Function

Converts an OS timestamp value to TDateTime value.

Pascal

```
function FileDateToDateTime(FileDate: Integer): TDateTime;
```

C++

```
TDateTime FileDateToDateTime(int FileDate);
```

File

SysUtils

Description

A timestamp is a signed 32-bit integer, used by the OS to record information such as the date and time a file was modified. The precise format of a timestamp depends on the OS. Use FileDateToDateTime to convert a timestamp to a TDateTime value.

You may need to use FileDateToDateTime to convert the Time (see page 22976) field of the TSearchRec (see page 23005) type used by the FindFirst (see page 22884) and FindNext (see page 22887) functions.

C++ Examples:

```
/*
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
```

```
changed settings:
*/
#include "fattrdlg.h"

void __fastcall TForm1::FileListMouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{

  unsigned short  Attributes;
  unsigned short  NewAttributes;

  FileAttrDlg->FileDirName->Caption = FileList->Items->Strings[FileList->ItemIndex];
  FileAttrDlg->FilePathName->Caption = FileList->Directory;
  FileAttrDlg->ChangeDate->Caption =
DateTimeToStr(FileDateToDateTime(FileAge(FileList->FileName)));
  Attributes = FileGetAttr(FileList->Items->Strings[FileList->ItemIndex]);
  FileAttrDlg->ReadOnly->Checked = Attributes & faReadOnly;
  FileAttrDlg->Archive->Checked = Attributes & faArchive;
  FileAttrDlg->System->Checked = Attributes & faSysFile;
  FileAttrDlg->Hidden->Checked = Attributes & faHidden;
  if (FileAttrDlg->ShowModal()!= mrCancel){
     NewAttributes = Attributes;
     if (FileAttrDlg->ReadOnly->Checked)
       NewAttributes = NewAttributes | faReadOnly;
     else
        NewAttributes = NewAttributes & ~faReadOnly;

     if (FileAttrDlg->Archive->Checked)
       NewAttributes = NewAttributes | faArchive;
     else
       NewAttributes = NewAttributes & ~faArchive;

     if (FileAttrDlg->System->Checked)
       NewAttributes = NewAttributes | faSysFile;
     else
       NewAttributes = NewAttributes & ~faSysFile;

     if (FileAttrDlg->Hidden->Checked)
       NewAttributes = NewAttributes | faHidden;
     else
       NewAttributes = NewAttributes  & ~faHidden;
     if (NewAttributes != Attributes)
       FileSetAttr(FileAttrDlg->FileDirName->Caption, NewAttributes);
  }
}
```

Delphi Examples:

```
{
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings:
}
procedure TForm1.FileListMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  Attributes, NewAttributes: Word;
begin
  with FileAttrForm do
  begin
    FileDirName.Caption := FileList.Items[FileList.ItemIndex];
    { set box caption }
    FilePathName.Caption := FileList.Directory;
    { show directory name }
```

```
      ChangeDate.Caption := DateTimeToStr(FileDateToDateTime(FileAge(FileList.FileName)));
      Attributes := SysUtils.FileGetAttr(FileDirName.Caption);
      { read file attributes }
      ReadOnly.Checked := (Attributes and SysUtils.faReadOnly) = faReadOnly;
      Archive.Checked := (Attributes and faArchive) = faArchive;
      System.Checked := (Attributes and faSysFile) = faSysFile;
      Hidden.Checked := (Attributes and faHidden) = faHidden;
      if ShowModal <> id_Cancel then    { execute dialog box }
      begin
        NewAttributes := Attributes;
        { start with original attributes }
        if ReadOnly.Checked then
          NewAttributes := NewAttributes or SysUtils.faReadOnly
        else
          NewAttributes := NewAttributes and not SysUtils.faReadOnly;
        if Archive.Checked then
          NewAttributes := NewAttributes or faArchive
        else
          NewAttributes := NewAttributes and not faArchive;
        if System.Checked then
          NewAttributes := NewAttributes or faSysFile
        else
          NewAttributes := NewAttributes and not faSysFile;
        if Hidden.Checked then
          NewAttributes := NewAttributes or faHidden
        else
          NewAttributes := NewAttributes and not faHidden;
        if NewAttributes <> Attributes then { if anything changed... }
          FileSetAttr(FileDirName.Caption, NewAttributes);
           { ...write the new values }
      end;
    end;
  end;
```

See Also

DateTimeToFileDate (see page 22821)

FindFirst (see page 22884)

FindNext (see page 22887)

159.3.86 SysUtils.FileExists Function

Tests if a specified file exists.

Pascal

```
function FileExists(const FileName: string): Boolean;
```

C++

```
Boolean FileExists(const AnsiString FileName);
```

File

SysUtils

Description

FileExists returns true if the file specified by FileName exists. If the file does not exist, FileExists returns false.

C++ Examples:

```
/*
```

```
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
*/

#include <dir.h>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
    if (FileExists(SaveDialog1->FileName))
    {
      fnsplit(SaveDialog1->FileName.c_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, (char*)&(StringGrid1->ColCount), sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, (char*)&(StringGrid1->RowCount), sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
    {
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(iFileHandle, (char*)&iLength, sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].c_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
    FileClose(iFileHandle);
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "Column 1";
  StringGrid1->Cells[2][0] = "Column 2";
  StringGrid1->Cells[3][0] = "Column 3";
  StringGrid1->Cells[4][0] = "Column 4";
  StringGrid1->Cells[0][1] = "Row 1";
  StringGrid1->Cells[1][1] = "Object";
  StringGrid1->Cells[2][1] = "Pascal";
  StringGrid1->Cells[3][1] = "is";
  StringGrid1->Cells[4][1] = "excellent";
  StringGrid1->Cells[0][2] = "Row 2";
  StringGrid1->Cells[1][2] = "Delphi";
  StringGrid1->Cells[2][2] = "is";
  StringGrid1->Cells[4][2] = "RAD";
};
/*
The following code prompts for confirmation before deleting
a file.  Create a file and then click on the button to select
the file with a TOpenDialog.  Click Yes to delete it.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenDialog1->Title = "Delete File";
```

```
  if (OpenDialog1->Execute())
  {
    if (FileExists(OpenDialog1->FileName))
    {
      if (MessageDlg(
        ("Do you really want to delete " + ExtractFileName(OpenDialog1->FileName) + "?"),
        mtConfirmation,
        TMsgDlgButtons() << mbYes << mbNo,
        0,
        mbNo) == mrYes)
        DeleteFile(OpenDialog1->FileName);
    }
    else
      MessageDlg(("File " + ExtractFileName(OpenDialog1->FileName) + " does not exist."),
mtConfirmation, TMsgDlgButtons() << mbOK, 0);
  }
}
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      Pchar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          { Write out the length of each string, followed by the string itself. }
          StringLen := Length(StringGrid1.Cells[X,Y]);
          FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
          GetMem(Buffer, StringLen); { allocate the buffer }
          cellString := StringGrid1.Cells[X,Y];
          for I := 1 to StringLen do
             Buffer[I - 1] := cellString[I];
```

```
          FileWrite(FileHandle, Buffer^, StringLen);
        finally
          FreeMem(Buffer, StringLen);
        end;
      end;
    end;
    FileClose(FileHandle);
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
    Cells[1,1] := 'Object';
    Cells[2,1] := 'Pascal';
    Cells[3,1] := 'is';
    Cells[4,1] := 'excellent';
    Cells[0,2] := 'Row 2';
    Cells[1,2] := 'Delphi';
    Cells[2,2] := 'is';
    Cells[4,2] := 'RAD';
  end;
end;
{
The following code prompts for confirmation before deleting
a file.  Create a file and then enter the path and file name
in the textfields.  Click the button to delete it.
}
procedure TForm1.Button1Click(Sender: TObject);
var FileName: string;
begin
  Filename:= Edit1.Text;
  if SysUtils.FileExists(FileName) then
  begin
    if MessageDlg(('Do you really want to delete ' + ExtractFileName(FileName) + '?'),
mtConfirmation, [mbYes, mbNo], 0, mbNo) = IDYes then
      DeleteFile(FileName);
  end
  else
    MessageDlg(('File ' + ExtractFileName(FileName) + ' does not exist.'), mtConfirmation,
[mbOK], 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit2.Text:= ExtractFilePath(Application.ExeName);
end;
```

See Also

FileSearch (see page 22872)

DirectoryExists (see page 22832)

159.3.87 SysUtils.FileGetAttr Function

Returns the file attributes of FileName.

Pascal

```
function FileGetAttr(const FileName: string): Integer; platform;
```

C++

```
int FileGetAttr(const AnsiString FileName);
```

File

SysUtils

Description

FileGetAttr returns the attributes of the file as a string of bits. This value is the same as the Attr field of a TSearchRec (see page 23005) type. Check for individual attributes with code such as the following:

Attrs := FileGetAttr('MyFile.sys');

```
if Attrs and faHidden <> 0 then
  FileSetAttr('MyFile.sys', Attrs – faHidden);
```

int Attrs = FileGetAttr("MyFile.sys");

```
if (Attrs & faHidden)
  FileSetAttr("MyFile.sys", Attrs & !faHidden);
```

A return value of -1 indicates that an error occurred.

Note: See TSearchRec (see page 23005) for a description of the individual attribute constants.

Note: FileGetAttr is Windows-only.

C++ Examples:

```
/*
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings:
*/
#include "fattrdlg.h"

void __fastcall TForm1::FileListMouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{

  unsigned short  Attributes;
  unsigned short  NewAttributes;

  FileAttrDlg->FileDirName->Caption = FileList->Items->Strings[FileList->ItemIndex];
  FileAttrDlg->FilePathName->Caption = FileList->Directory;
  FileAttrDlg->ChangeDate->Caption =
DateTimeToStr(FileDateToDateTime(FileAge(FileList->FileName)));
  Attributes = FileGetAttr(FileList->Items->Strings[FileList->ItemIndex]);
  FileAttrDlg->ReadOnly->Checked = Attributes & faReadOnly;
  FileAttrDlg->Archive->Checked = Attributes & faArchive;
  FileAttrDlg->System->Checked = Attributes & faSysFile;
  FileAttrDlg->Hidden->Checked = Attributes & faHidden;
  if (FileAttrDlg->ShowModal()!= mrCancel){
     NewAttributes = Attributes;
     if (FileAttrDlg->ReadOnly->Checked)
       NewAttributes = NewAttributes | faReadOnly;
```

```
      else
         NewAttributes = NewAttributes & ~faReadOnly;

      if (FileAttrDlg->Archive->Checked)
        NewAttributes = NewAttributes | faArchive;
      else
        NewAttributes = NewAttributes & ~faArchive;

      if (FileAttrDlg->System->Checked)
        NewAttributes = NewAttributes | faSysFile;
      else
        NewAttributes = NewAttributes & ~faSysFile;

      if (FileAttrDlg->Hidden->Checked)
        NewAttributes = NewAttributes | faHidden;
      else
        NewAttributes = NewAttributes  & ~faHidden;
      if (NewAttributes != Attributes)
        FileSetAttr(FileAttrDlg->FileDirName->Caption, NewAttributes);
  }
}
```

Delphi Examples:

```
{
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings:
}
procedure TForm1.FileListMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  Attributes, NewAttributes: Word;
begin
  with FileAttrForm do
  begin
    FileDirName.Caption := FileList.Items[FileList.ItemIndex];
    { set box caption }
    FilePathName.Caption := FileList.Directory;
    { show directory name }
    ChangeDate.Caption := DateTimeToStr(FileDateToDateTime(FileAge(FileList.FileName)));
    Attributes := SysUtils.FileGetAttr(FileDirName.Caption);
    { read file attributes }
    ReadOnly.Checked := (Attributes and SysUtils.faReadOnly) = faReadOnly;
    Archive.Checked := (Attributes and faArchive) = faArchive;
    System.Checked := (Attributes and faSysFile) = faSysFile;
    Hidden.Checked := (Attributes and faHidden) = faHidden;
    if ShowModal <> id_Cancel then    { execute dialog box }
    begin
      NewAttributes := Attributes;
      { start with original attributes }
      if ReadOnly.Checked then
        NewAttributes := NewAttributes or SysUtils.faReadOnly
      else
        NewAttributes := NewAttributes and not SysUtils.faReadOnly;
      if Archive.Checked then
        NewAttributes := NewAttributes or faArchive
      else
        NewAttributes := NewAttributes and not faArchive;
      if System.Checked then
        NewAttributes := NewAttributes or faSysFile
      else
        NewAttributes := NewAttributes and not faSysFile;
      if Hidden.Checked then
```

```
          NewAttributes := NewAttributes or faHidden
        else
          NewAttributes := NewAttributes and not faHidden;
        if NewAttributes <> Attributes then { if anything changed... }
          FileSetAttr(FileDirName.Caption, NewAttributes);
           { ...write the new values }
      end;
    end;
  end;
```

See Also

FileSetAttr (see page 22875)

159.3.88 SysUtils.FileGetDate Function

Returns an OS timestamp for a specified file.

Pascal

```
function FileGetDate(Handle: Integer): Integer;
```

C++

```
int FileGetDate(int Handle);
```

File

SysUtils

Description

Use FileGetDate to determine the timestamp of a file given its handle. The return value is -1 if the handle is invalid. The FileDateToDateTime (see page 22858) function can be used to convert the returned value to a TDateTime value.

Note: :FileGetDate uses a file handle rather than a native Delphi file variable. To obtain the timestamp of a file that has not been opened using the FileOpen (see page 22867) or FileCreate (see page 22855) method, use the FileAge (see page 22850) function instead.

Note: :Under Linux, -1 is a valid timestamp. Use FileExists (see page 22860) to verify that the file does not exist.

See Also

FileSetDate (see page 22878)

FileDateToDateTime (see page 22858)

FileAge (see page 22850)

FileExists (see page 22860)

159.3.89 SysUtils.FileIsReadOnly Function

Reports whether a file is read-only.

Pascal

```
function FileIsReadOnly(const FileName: string): Boolean;
```

C++

```
Boolean FileIsReadOnly(const AnsiString FileName);
```

File

SysUtils

Description

FileIsReadOnly returns **True** if FileName can be found and opened, but the current process does not have permission to write to it.

Note: On Windows XP, FileIsReadOnly returns **True** if the file does **not** exist.

159.3.90 SysUtils.FileOpen Function

Opens a specified file using a specified access mode.

Pascal

```
function FileOpen(const FileName: string; Mode: LongWord): Integer;
```

C++

```
int FileOpen(const AnsiString FileName, LongWord Mode);
```

File

SysUtils

Description

Use FileOpen to open a file and obtain a file handle. The access mode value is constructed by or-ing one of the fmOpen constants with one of the fmShare constants defined in File open mode constants. If the return value is 0 or greater, the function was successful and the value is the file handle of the opened file. A return value of -1 indicates that an error occurred.

Note: Use of the non-native Delphi language file handlers such as FileOpen is not encouraged. These routines map to system routines and return OS file handles, not normal Delphi file variables. These are low-level file access routines. For normal file operations use AssignFile, Rewrite, and Reset instead.

C++ Examples:

```
/*
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
  char *pszBuffer;
  if (OpenDialog1->Execute())
  {
```

```
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      pszBuffer = PChar(AllocMem(iFileLength+1));
//    pszBuffer = new char[iFileLength+1];
      iBytesRead = FileRead(iFileHandle, pszBuffer, iFileLength);
      FileClose(iFileHandle);
      for (int i=0;i<iBytesRead;i++)
      {
        StringGrid1->RowCount += 1;
        StringGrid1->Cells[1][i+1] = pszBuffer[i];
        StringGrid1->Cells[2][i+1] = IntToStr((int)pszBuffer[i]);
      }
      delete [] pszBuffer;
    }
    catch(...)
    {
      Application->MessageBox("Can't perform one of the following file operations: Open, Seek,
Read, Close.", "File Error", IDOK);
    }
  }
}
```

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], 1);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode 1.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);

      AssignFile(F, FileListBox1.Items.Strings[i]);
      Reset(F, 1);
      ListBox1.Items.Add(
        FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
      CloseFile(F);
```

```
    end;
   finally
   { do something here }
   end;
  end;
end;
{
This example requires a TEdit, a button and a string grid.
Enter the file name in the TEdit and click the button to open
that file.  Notice that the file cannot be read.  Remove the
fmOpenWrite and the fmShareDenyNone access and the string
grid loads fine.
}
procedure OpenForShare(const FileName: String);
var
  FileHandle : Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PChar;
  i: Integer;
begin
  FileHandle := SysUtils.FileOpen(FileName, fmOpenWrite or fmShareDenyNone or fmOpenRead);
  if FileHandle > 0 then
  begin
    try
      MessageDlg('Valid file handle.', mtInformation, [mbOk], 0);
      iFileLength := SysUtils.FileSeek(FileHandle,0,2);
      FileSeek(FileHandle, 0, 0);
      Buffer := PChar(System.AllocMem(iFileLength + 1));
      iBytesRead := SysUtils.FileRead(FileHandle, Buffer^, iFileLength);
      for i := 0 to iBytesRead-1 do
      begin
        Form1.StringGrid1.RowCount := Form1.StringGrid1.RowCount + 1;
        Form1.StringGrid1.Cells[1, i + 1] := Buffer[i];
        Form1.StringGrid1.Cells[2, i + 1] := IntToStr(Integer(Buffer[i]));
      end;
    finally
      FreeMem(Buffer);
    end;
  end
  else
    MessageDlg('Open error: FileHandle = negative DOS error code.', mtInformation, [mbOk], 0);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  OpenForShare(Edit1.Text);
end;
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  iFileHandle: Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PChar;
  i: Integer;
begin
```

```
    if OpenDialog1.Execute then
    begin
      try
        iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
        iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
        FileSeek(iFileHandle,0,0);
        Buffer := PChar(System.AllocMem(iFileLength + 1));
        iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
        FileClose(iFileHandle);
        for i := 0 to iBytesRead-1 do
        begin
          StringGrid1.RowCount := StringGrid1.RowCount + 1;
          StringGrid1.Cells[1,i+1] := Buffer[i];
          StringGrid1.Cells[2,i+1] := IntToStr(Integer(Buffer[i]));
        end;
      finally
        FreeMem(Buffer);
      end;
    end;
  end;
```

See Also

FileClose (see page 22852)

FileCreate (see page 22855)

AssignFile (see page 22420)

Rewrite (see page 22513)

Reset (see page 22511)

159.3.91 SysUtils.FileRead Function

Reads a specified number of bytes from a file.

Pascal

```
function FileRead(Handle: Integer; var Buffer; Count: LongWord): Integer;
```

C++

```
int FileRead(int Handle,  Buffer, LongWord Count);
```

File

SysUtils

Description

FileRead reads Count bytes from the file specified by Handle into the buffer. The Count parameter indicates the size, in bytes, of the buffer. The function result is the actual number of bytes read, which may be less than Count. The Handle that is passed to FileRead must be opened with FileOpen (see page 22867) or FileCreate (see page 22855).

Note: :In Delphi, do not mix routines that take or return file handles with those that use Delphi language file variables (typically seen as var F). To read from a file specified by a Delphi file variable, use the BlockRead procedure or the Read procedure instead.

C++ Examples:

```
/*
```

```
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
  char *pszBuffer;
  if (OpenDialog1->Execute())
  {
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      pszBuffer = PChar(AllocMem(iFileLength+1));
//    pszBuffer = new char[iFileLength+1];
      iBytesRead = FileRead(iFileHandle, pszBuffer, iFileLength);
      FileClose(iFileHandle);
      for (int i=0;i<iBytesRead;i++)
      {
        StringGrid1->RowCount += 1;
        StringGrid1->Cells[1][i+1] = pszBuffer[i];
        StringGrid1->Cells[2][i+1] = IntToStr((int)pszBuffer[i]);
      }
      delete [] pszBuffer;
    }
    catch(...)
    {
      Application->MessageBox("Can't perform one of the following file operations: Open, Seek,
Read, Close.", "File Error", IDOK);
    }
  }
}
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  iFileHandle: Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PChar;
  i: Integer;
begin
  if OpenDialog1.Execute then
  begin
    try
      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
```

```
        FileSeek(iFileHandle,0,0);
        Buffer := PChar(System.AllocMem(iFileLength + 1));
        iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
        FileClose(iFileHandle);
        for i := 0 to iBytesRead-1 do
        begin
          StringGrid1.RowCount := StringGrid1.RowCount + 1;
          StringGrid1.Cells[1,i+1] := Buffer[i];
          StringGrid1.Cells[2,i+1] := IntToStr(Integer(Buffer[i]));
        end;
      finally
        FreeMem(Buffer);
      end;
    end;
  end;
```

See Also

BlockRead (see page 22423)

FileOpen (see page 22867)

FileCreate (see page 22855)

FileSeek (see page 22873)

FileWrite (see page 22879)

Read (see page 22504)

Write (see page 22548)

159.3.92 SysUtils.FileSearch Function

Searches a specified directory path for a file.

Pascal

```
function FileSearch(const Name: string; const DirList: string): string;
```

C++

```
AnsiString FileSearch(const AnsiString Name, const AnsiString DirList);
```

File

SysUtils

Description

FileSearch searches through the directories passed in DirList for a file named Name. DirList is a list of path names delimited by semicolons on Windows, colons on Linux). If FileSearch locates a file matching Name, it returns a string specifying a path name for that file. If no matching file exists, FileSearch returns an empty string.

C++ Examples:

```
/*
The following example uses an edit control and a button on a
form. When the button is clicked, the current directory is
searched for the filename specified in the edit control. A
message box indicates whether the file is found.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString asFileName = FileSearch(Edit1->Text, GetCurrentDir());
```

```
  if (asFileName.IsEmpty())
    ShowMessage(AnsiString("Couldn't find ") + Edit1->Text + ".");
  else
    ShowMessage(AnsiString("Found ") + asFileName + ".");
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Label2->Caption = GetCurrentDir();
}
```

Delphi Examples:

```
{
The following example uses an edit control and a button on a
form. When the button is clicked, the current directory is
searched for the filename specified in the edit control. A
message box indicates whether the file is found.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FileToFind: string;
begin
  FileToFind := SysUtils.FileSearch(Edit1.Text, GetCurrentDir);
  if FileToFind = '' then
    ShowMessage('Couldn''t find ' + Edit1.Text + '.')
  else
    ShowMessage('Found ' + FileToFind + '.');
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Label2.Caption := GetCurrentDir;
end;
```

See Also

FileExists (see page 22860)

159.3.93 SysUtils.FileSeek Function

Repositions read/write point.

Pascal

```
function FileSeek(Handle: Integer; Offset: Integer; Origin: Integer): Integer; overload;
function FileSeek(Handle: Integer; const Offset: Int64; Origin: Integer): Int64; overload;
```

C++

```
int FileSeek(int Handle, int Offset, int Origin);
Int64 FileSeek(int Handle, const Int64 Offset, int Origin);
```

File

SysUtils

Description

Use FileSeek to reposition the read/write point in a file that was opened with FileOpen (see page 22867) or FileCreate (see page 22855). Handle is the file handle that was returned by FileOpen (see page 22867) or FileCreate (see page 22855).

Offset specifies the number of bytes from Origin where the file pointer should be positioned. Origin is a code with three possible values, denoting the beginning of the file, the end of the file, and the current position of the file pointer.

Origin	Action
0	The file pointer is positioned Offset bytes from the beginning of the file.
1	The file pointer is positioned Offset bytes from its current position.
2	The file pointer is positioned Offset bytes from the end of the file.

If FileSeek is successful, it returns the new position of the file pointer; otherwise, it returns -1.

Note: :Do not mix routines that take or return file handles with those that use Delphi language file variables (typically seen as var F). To move the file pointer in a file specified by a Delphi file variable, use the Seek procedure instead.

C++ Examples:

```
/*
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
  char *pszBuffer;
  if (OpenDialog1->Execute())
  {
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      pszBuffer = PChar(AllocMem(iFileLength+1));
//    pszBuffer = new char[iFileLength+1];
      iBytesRead = FileRead(iFileHandle, pszBuffer, iFileLength);
      FileClose(iFileHandle);
      for (int i=0;i<iBytesRead;i++)
      {
        StringGrid1->RowCount += 1;
        StringGrid1->Cells[1][i+1] = pszBuffer[i];
        StringGrid1->Cells[2][i+1] = IntToStr((int)pszBuffer[i]);
      }
      delete [] pszBuffer;
    }
    catch(...)
    {
      Application->MessageBox("Can't perform one of the following file operations: Open, Seek,
Read, Close.", "File Error", IDOK);
    }
  }
}
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
```

```
  the specified file is opened, read into a buffer, and closed.
  Then the buffer is displayed in two columns of the string
  grid. The first column contains the character values in the
  buffer. The second column contains the numeric values of the
  characters in the buffer.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    iFileHandle: Integer;
    iFileLength: Integer;
    iBytesRead: Integer;
    Buffer: PChar;
    i: Integer;
  begin
    if OpenDialog1.Execute then
    begin
      try
        iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
        iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
        FileSeek(iFileHandle,0,0);
        Buffer := PChar(System.AllocMem(iFileLength + 1));
        iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
        FileClose(iFileHandle);
        for i := 0 to iBytesRead-1 do
        begin
          StringGrid1.RowCount := StringGrid1.RowCount + 1;
          StringGrid1.Cells[1,i+1] := Buffer[i];
          StringGrid1.Cells[2,i+1] := IntToStr(Integer(Buffer[i]));
        end;
      finally
        FreeMem(Buffer);
      end;
    end;
  end;
```

See Also

FileOpen (see page 22867)

FileCreate (see page 22855)

FileRead (see page 22870)

FileWrite (see page 22879)

Seek (see page 22518)

159.3.94 SysUtils.FileSetAttr Function

Sets the file attributes of a specified file.

Pascal

```
function FileSetAttr(const FileName: string; Attr: Integer): Integer; platform;
```

C++

```
int FileSetAttr(const AnsiString FileName, int Attr);
```

File

SysUtils

Description

FileSetAttr sets the file attributes of the file given by FileName to the value given by Attr. The value of Attr is formed by combining the appropriate file attribute constants, as in the following:

```
FileSetAttr('MyFile.sys', faReadOnly or faSysFile); // Delphi
FileSetAttr("MyFile.sys", faReadOnly | faSysFile); // C++
```

FileSetAttr returns zero if the function was successful. Otherwise the return value is an error code.

Note: See TSearchRec (see page 23005) for a description of the file attribute constants.

Note: FileSetAttr is only available on Windows.

C++ Examples:

```
/*
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings:
*/
#include "fattrdlg.h"

void __fastcall TForm1::FileListMouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{

  unsigned short  Attributes;
  unsigned short  NewAttributes;

  FileAttrDlg->FileDirName->Caption = FileList->Items->Strings[FileList->ItemIndex];
  FileAttrDlg->FilePathName->Caption = FileList->Directory;
  FileAttrDlg->ChangeDate->Caption =
DateTimeToStr(FileDateToDateTime(FileAge(FileList->FileName)));
  Attributes = FileGetAttr(FileList->Items->Strings[FileList->ItemIndex]);
  FileAttrDlg->ReadOnly->Checked = Attributes & faReadOnly;
  FileAttrDlg->Archive->Checked = Attributes & faArchive;
  FileAttrDlg->System->Checked = Attributes & faSysFile;
  FileAttrDlg->Hidden->Checked = Attributes & faHidden;
  if (FileAttrDlg->ShowModal()!= mrCancel){
     NewAttributes = Attributes;
     if (FileAttrDlg->ReadOnly->Checked)
       NewAttributes = NewAttributes | faReadOnly;
     else
        NewAttributes = NewAttributes & ~faReadOnly;

     if (FileAttrDlg->Archive->Checked)
       NewAttributes = NewAttributes | faArchive;
     else
       NewAttributes = NewAttributes & ~faArchive;

     if (FileAttrDlg->System->Checked)
       NewAttributes = NewAttributes | faSysFile;
     else
       NewAttributes = NewAttributes & ~faSysFile;

     if (FileAttrDlg->Hidden->Checked)
       NewAttributes = NewAttributes | faHidden;
     else
       NewAttributes = NewAttributes  & ~faHidden;
```

```
      if (NewAttributes != Attributes)
        FileSetAttr(FileAttrDlg->FileDirName->Caption, NewAttributes);
    }
  }
```

Delphi Examples:

```
{
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings:
}
procedure TForm1.FileListMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  Attributes, NewAttributes: Word;
begin
  with FileAttrForm do
  begin
    FileDirName.Caption := FileList.Items[FileList.ItemIndex];
    { set box caption }
    FilePathName.Caption := FileList.Directory;
    { show directory name }
    ChangeDate.Caption := DateTimeToStr(FileDateToDateTime(FileAge(FileList.FileName)));
    Attributes := SysUtils.FileGetAttr(FileDirName.Caption);
    { read file attributes }
    ReadOnly.Checked := (Attributes and SysUtils.faReadOnly) = faReadOnly;
    Archive.Checked := (Attributes and faArchive) = faArchive;
    System.Checked := (Attributes and faSysFile) = faSysFile;
    Hidden.Checked := (Attributes and faHidden) = faHidden;
    if ShowModal <> id_Cancel then    { execute dialog box }
    begin
      NewAttributes := Attributes;
      { start with original attributes }
      if ReadOnly.Checked then
        NewAttributes := NewAttributes or SysUtils.faReadOnly
      else
        NewAttributes := NewAttributes and not SysUtils.faReadOnly;
      if Archive.Checked then
        NewAttributes := NewAttributes or faArchive
      else
        NewAttributes := NewAttributes and not faArchive;
      if System.Checked then
        NewAttributes := NewAttributes or faSysFile
      else
        NewAttributes := NewAttributes and not faSysFile;
      if Hidden.Checked then
        NewAttributes := NewAttributes or faHidden
      else
        NewAttributes := NewAttributes and not faHidden;
      if NewAttributes <> Attributes then { if anything changed... }
        FileSetAttr(FileDirName.Caption, NewAttributes);
         { ...write the new values }
    end;
  end;
end;
```

See Also

FileGetAttr (see page 22863)

TSearchRec (see page 23005)

159.3.95 SysUtils.FileSetDate Function

Sets the OS time stamp for a specified file.

Pascal

```
function FileSetDate(const FileName: string; Age: Integer): Integer; overload;
function FileSetDate(Handle: Integer; Age: Integer): Integer; overload; platform;
```

C++

```
int FileSetDate(const AnsiString FileName, int Age);
int FileSetDate(int Handle, int Age);
```

File

SysUtils

Description

FileSetDate sets the OS timestamp of the specified file to the value given by Age. The DateTimeToFileDate (see page 22821) function can be used to convert a TDateTime value to a OS timestamp.

Handle is the Windows file handle of the file to alter. This syntax is only available on Windows.

FileName is the name of the file to alter.

Age is the timestamp to apply to the specified file.

The return value is zero if the function was successful. Otherwise the return value is an error code.

See Also

FileCreate (see page 22855)

FileGetDate (see page 22866)

FileOpen (see page 22867)

DateTimeToFileDate (see page 22821)

159.3.96 SysUtils.FileSetReadOnly Function

Sets the file permissions to read-only.

Pascal

```
function FileSetReadOnly(const FileName: string; ReadOnly: Boolean): Boolean;
```

C++

```
Boolean FileSetReadOnly(const AnsiString FileName, Boolean ReadOnly);
```

File

SysUtils

Description

FileSetReadOnly attempts to set the read-only status of the specified file.

FileName is the name of the file to alter.

ReadOnly indicates the status to assign to the file.

FileSetReadOnly returns true if the operation was successful and false if an error occurred.

Note: On Windows, the ReadOnly parameter is assigned to the file's read-only flag.

Note: On Linux, the inverse of the ReadOnly parameter is assigned to the file's user write, group write, and other write permission bits. The active user must have permission to perform these operations.

FileSetReadOnly is designed to support cross-platform code. To manipulate permissions more specifically than FileSetReadOnly allows, use platform-specific functions.

159.3.97 SysUtils.FileWrite Function

Writes the contents of a buffer to the current position in a file.

Pascal

```
function FileWrite(Handle: Integer; const Buffer; Count: LongWord): Integer;
```

C++

```
int FileWrite(int Handle, const  Buffer, LongWord Count);
```

File

SysUtils

Description

FileWrite writes Count bytes to the file given by Handle from the buffer specified by Buffer. Handle is a file handle returned by the FileOpen (see page 22867) or FileCreate (see page 22855) method.

The return value is the number of bytes actually written, or -1 if an error occurred.

Note: :Do not mix routines that take or return file handles with those that use Delphi language file variables (typically seen as var F). To write to a file specified by a Delphi file variable, use Write, Writeln, or BlockWrite instead.

C++ Examples:

```
/*
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
*/

#include <dir.h>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
    if (FileExists(SaveDialog1->FileName))
```

```
    {
      fnsplit(SaveDialog1->FileName.c_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, (char*)&(StringGrid1->ColCount), sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, (char*)&(StringGrid1->RowCount), sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
    {
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(iFileHandle, (char*)&iLength, sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].c_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
    FileClose(iFileHandle);
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "Column 1";
  StringGrid1->Cells[2][0] = "Column 2";
  StringGrid1->Cells[3][0] = "Column 3";
  StringGrid1->Cells[4][0] = "Column 4";
  StringGrid1->Cells[0][1] = "Row 1";
  StringGrid1->Cells[1][1] = "Object";
  StringGrid1->Cells[2][1] = "Pascal";
  StringGrid1->Cells[3][1] = "is";
  StringGrid1->Cells[4][1] = "excellent";
  StringGrid1->Cells[0][2] = "Row 2";
  StringGrid1->Cells[1][2] = "Delphi";
  StringGrid1->Cells[2][2] = "is";
  StringGrid1->Cells[4][2] = "RAD";
};
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
```

```
        if not RenameFile(SaveDialog1.FileName, BackupName) then
          raise Exception.Create('Unable to create backup file.');
      end;
      FileHandle := FileCreate(SaveDialog1.FileName);
      { Write out the number of rows and columns in the grid. }
      colCountLength := SizeOf(StringGrid1.ColCount);
      FileWrite(FileHandle,
        Pchar(StringGrid1.ColCount), colCountLength);
      rowCountLength := SizeOf(StringGrid1.RowCount);
      FileWrite(FileHandle,
        PChar(StringGrid1.RowCount), rowCountLength);
      for X := 0 to StringGrid1.ColCount - 1 do
      begin
        for Y := 0 to StringGrid1.RowCount - 1 do
        begin
          try
            { Write out the length of each string, followed by the string itself. }
            StringLen := Length(StringGrid1.Cells[X,Y]);
            FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
            GetMem(Buffer, StringLen); { allocate the buffer }
            cellString := StringGrid1.Cells[X,Y];
            for I := 1 to StringLen do
               Buffer[I - 1] := cellString[I];
            FileWrite(FileHandle, Buffer^, StringLen);
          finally
            FreeMem(Buffer, StringLen);
          end;
        end;
      end;
      FileClose(FileHandle);
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with StringGrid1 do
    begin
      Cells[1,0] := 'Column 1';
      Cells[2,0] := 'Column 2';
      Cells[3,0] := 'Column 3';
      Cells[4,0] := 'Column 4';
      Cells[0,1] := 'Row 1';
      Cells[1,1] := 'Object';
      Cells[2,1] := 'Pascal';
      Cells[3,1] := 'is';
      Cells[4,1] := 'excellent';
      Cells[0,2] := 'Row 2';
      Cells[1,2] := 'Delphi';
      Cells[2,2] := 'is';
      Cells[4,2] := 'RAD';
    end;
  end;
```

See Also

FileRead (see page 22870)

FileSeek (see page 22873)

Writeln

Write (see page 22548)

BlockWrite (see page 22425)

159.3.98 SysUtils.FinalizePackage Function

Finalizes the specified package (.bpl file).

Pascal

```
procedure FinalizePackage(Module: HMODULE);
```

C++

```
FinalizePackage(HMODULE Module);
```

File

SysUtils

Description

Most applications do not need to call FinalizePackage, because UnloadPackage (see page 22987) calls FinalizePackage internally. FinalizePackage executes any finalization code in the specified package.

See Also

UnloadPackage (see page 22987)

159.3.99 SysUtils.FindClose Function

Releases memory allocated by FindFirst (see page 22884).

Pascal

```
procedure FindClose(var F: TSearchRec);
```

C++

```
FindClose(TSearchRec F);
```

File

SysUtils

Description

FindClose terminates a FindFirst (see page 22884)/FindNext (see page 22887) sequence.

C++ Examples:

```
/*
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory).
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TSearchRec sr;
```

```
  int iAttributes = 0;
  StringGrid1->RowCount = 1;
  iAttributes |= faReadOnly * CheckBox1->Checked;
  iAttributes |= faHidden * CheckBox2->Checked;
  iAttributes |= faSysFile * CheckBox3->Checked;
  iAttributes |= faVolumeID * CheckBox4->Checked;
  iAttributes |= faDirectory * CheckBox5->Checked;
  iAttributes |= faArchive * CheckBox6->Checked;
  iAttributes |= faAnyFile * CheckBox7->Checked;
  StringGrid1->RowCount = 0;
  if (FindFirst(Edit1->Text, iAttributes, sr) == 0)
  {
    do
    {
      if ((sr.Attr & iAttributes) == sr.Attr)
      {
        StringGrid1->RowCount = StringGrid1->RowCount + 1;
        StringGrid1->Cells[1][StringGrid1->RowCount-1] = sr.Name;
        StringGrid1->Cells[2][StringGrid1->RowCount-1] = IntToStr(sr.Size);
      }
    } while (FindNext(sr) == 0);
    FindClose(sr);
  }
}
```

Delphi Examples:

```
{
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory).
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sr: TSearchRec;
  FileAttrs: Integer;
begin
  StringGrid1.RowCount := 1;
  if CheckBox1.Checked then
    FileAttrs := faReadOnly
  else
    FileAttrs := 0;
  if CheckBox2.Checked then
    FileAttrs := FileAttrs + faHidden;
  if CheckBox3.Checked then
    FileAttrs := FileAttrs + faSysFile;
  if CheckBox4.Checked then
    FileAttrs := FileAttrs + faVolumeID;
  if CheckBox5.Checked then
    FileAttrs := FileAttrs + faDirectory;
  if CheckBox6.Checked then
    FileAttrs := FileAttrs + faArchive;
  if CheckBox7.Checked then
    FileAttrs := FileAttrs + faAnyFile;
  with StringGrid1 do
  begin
    RowCount := 0;
    if SysUtils.FindFirst(Edit1.Text, FileAttrs, sr) = 0 then
    begin
      repeat
```

```
          if (sr.Attr and FileAttrs) = sr.Attr then
          begin
          RowCount := RowCount + 1;
          Cells[1,RowCount-1] := sr.Name;
          Cells[2,RowCount-1] := IntToStr(sr.Size);
          end;
        until FindNext(sr) <> 0;
        FindClose(sr);
      end;
    end;
  end;
```

See Also

FindFirst (see page 22884)

FindNext (see page 22887)

159.3.100 SysUtils.FindCmdLineSwitch Function

Determines whether a string was passed as a command line argument to the application.

Pascal

```
function FindCmdLineSwitch(const Switch: string; const Chars: TSysCharSet; IgnoreCase:
Boolean): Boolean; overload;
function FindCmdLineSwitch(const Switch: string): Boolean; overload;
function FindCmdLineSwitch(const Switch: string; IgnoreCase: Boolean): Boolean; overload;
```

C++

```
Boolean FindCmdLineSwitch(const AnsiString Switch, const TSysCharSet Chars, Boolean
IgnoreCase);
Boolean FindCmdLineSwitch(const AnsiString Switch);
Boolean FindCmdLineSwitch(const AnsiString Switch, Boolean IgnoreCase);
```

File

SysUtils

Description

FindCmdLineSwitch determines whether a string represents a command-line switch, based on the first character of the string.

Switch is a command line parameter, typically provided by the ParamStr function.

Chars is the set of characters that distinguish switches from other parameters. If SwitchChars (see page 23065) is omitted, it defaults to '/' and '-' on Windows, and to '-' on Linux.

IgnoreCase determines whether case-insensitive character matching is used. The default is true on Windows, false on Linux.

See Also

ParamCount (see page 22495)

ParamStr (see page 22496)

159.3.101 SysUtils.FindFirst Function

Searches for the first instance of a file name with a given set of attributes in a specified directory.

Pascal

```
function FindFirst(const Path: string; Attr: Integer; var F: TSearchRec): Integer;
```

C++

```
int FindFirst(const AnsiString Path, int Attr, TSearchRec F);
```

File

SysUtils

Description

FindFirst searches the directory specified by Path for the first file that matches the file name implied by Path and the attributes specified by the Attr parameter. The result is returned in the F parameter. Use the fields of this search record to extract the information needed. FindFirst returns 0 if a file was successfully located, otherwise, it returns an error code.

The Path constant parameter is the directory and file name mask, including wildcard characters. For example, '.\test*.*' specifies all files in the test subdirectory.

The Attr parameter specifies the special files to include in addition to all normal files. Choose from these file attribute constants when specifying the Attr parameter:

Constant	Description
faReadOnly (see page 23044)	Read-only files
faHidden (see page 23043)	Hidden files
faSysFile (see page 23045)	System (see page 22079) files
faVolumeID (see page 23045)	Volume ID files
faDirectory (see page 23042)	Directory files
faArchive (see page 23042)	Archive files
faAnyFile (see page 23041)	Any file

Attributes can be combined by adding (Delphi) or or-ing (C++) their constants or values. For example, to search for read-only and hidden files in addition to normal files, pass (faReadOnly (see page 23044) + faHidden (see page 23043)) in Delphi or (faReadOnly (see page 23044) | faHidden (see page 23043)) in C++ as the Attr parameter. To include only normal files, pass zero for the Attr parameter.

Note: FindFirst allocates resources (memory) which must be released by calling FindClose (see page 22882).

Note: Some of the file attribute constants are not valid on all platforms. For example, faVolumeID (see page 23045) and faArchive (see page 23042) will not work on Linux.

C++ Examples:

```
/*
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory).
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TSearchRec sr;
```

```
  int iAttributes = 0;
  StringGrid1->RowCount = 1;
  iAttributes |= faReadOnly * CheckBox1->Checked;
  iAttributes |= faHidden * CheckBox2->Checked;
  iAttributes |= faSysFile * CheckBox3->Checked;
  iAttributes |= faVolumeID * CheckBox4->Checked;
  iAttributes |= faDirectory * CheckBox5->Checked;
  iAttributes |= faArchive * CheckBox6->Checked;
  iAttributes |= faAnyFile * CheckBox7->Checked;
  StringGrid1->RowCount = 0;
  if (FindFirst(Edit1->Text, iAttributes, sr) == 0)
  {
    do
    {
      if ((sr.Attr & iAttributes) == sr.Attr)
      {
        StringGrid1->RowCount = StringGrid1->RowCount + 1;
        StringGrid1->Cells[1][StringGrid1->RowCount-1] = sr.Name;
        StringGrid1->Cells[2][StringGrid1->RowCount-1] = IntToStr(sr.Size);
      }
    } while (FindNext(sr) == 0);
    FindClose(sr);
  }
}
```

Delphi Examples:

```
{
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory).
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sr: TSearchRec;
  FileAttrs: Integer;
begin
  StringGrid1.RowCount := 1;
  if CheckBox1.Checked then
    FileAttrs := faReadOnly
  else
    FileAttrs := 0;
  if CheckBox2.Checked then
    FileAttrs := FileAttrs + faHidden;
  if CheckBox3.Checked then
    FileAttrs := FileAttrs + faSysFile;
  if CheckBox4.Checked then
    FileAttrs := FileAttrs + faVolumeID;
  if CheckBox5.Checked then
    FileAttrs := FileAttrs + faDirectory;
  if CheckBox6.Checked then
    FileAttrs := FileAttrs + faArchive;
  if CheckBox7.Checked then
    FileAttrs := FileAttrs + faAnyFile;
  with StringGrid1 do
  begin
    RowCount := 0;
    if SysUtils.FindFirst(Edit1.Text, FileAttrs, sr) = 0 then
    begin
      repeat
```

```
        if (sr.Attr and FileAttrs) = sr.Attr then
        begin
        RowCount := RowCount + 1;
        Cells[1,RowCount-1] := sr.Name;
        Cells[2,RowCount-1] := IntToStr(sr.Size);
        end;
      until FindNext(sr) <> 0;
      FindClose(sr);
    end;
  end;
end;
```

See Also

FindClose (see page 22882)

FindNext (see page 22887)

159.3.102 SysUtils.FindNext Function

Returns the next entry matching the name and attributes specified in a previous call to FindFirst (see page 22884).

Pascal

```
function FindNext(var F: TSearchRec): Integer;
```

C++

```
int FindNext(TSearchRec F);
```

File

SysUtils

Description

FindNext returns the next entry that matches the name and attributes specified in a previous call to FindFirst (see page 22884). The search record must be one that was passed to FindFirst (see page 22884). The return value is zero if the function was successful. Otherwise the return value is an error code.

C++ Examples:

```
/*
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory).
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TSearchRec sr;
  int iAttributes = 0;
  StringGrid1->RowCount = 1;
  iAttributes |= faReadOnly * CheckBox1->Checked;
  iAttributes |= faHidden * CheckBox2->Checked;
  iAttributes |= faSysFile * CheckBox3->Checked;
  iAttributes |= faVolumeID * CheckBox4->Checked;
  iAttributes |= faDirectory * CheckBox5->Checked;
```

```
  iAttributes |= faArchive * CheckBox6->Checked;
  iAttributes |= faAnyFile * CheckBox7->Checked;
  StringGrid1->RowCount = 0;
  if (FindFirst(Edit1->Text, iAttributes, sr) == 0)
  {
    do
    {
      if ((sr.Attr & iAttributes) == sr.Attr)
      {
        StringGrid1->RowCount = StringGrid1->RowCount + 1;
        StringGrid1->Cells[1][StringGrid1->RowCount-1] = sr.Name;
        StringGrid1->Cells[2][StringGrid1->RowCount-1] = IntToStr(sr.Size);
      }
    } while (FindNext(sr) == 0);
    FindClose(sr);
  }
}
```

Delphi Examples:

```
{
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory).
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sr: TSearchRec;
  FileAttrs: Integer;
begin
  StringGrid1.RowCount := 1;
  if CheckBox1.Checked then
    FileAttrs := faReadOnly
  else
    FileAttrs := 0;
  if CheckBox2.Checked then
    FileAttrs := FileAttrs + faHidden;
  if CheckBox3.Checked then
    FileAttrs := FileAttrs + faSysFile;
  if CheckBox4.Checked then
    FileAttrs := FileAttrs + faVolumeID;
  if CheckBox5.Checked then
    FileAttrs := FileAttrs + faDirectory;
  if CheckBox6.Checked then
    FileAttrs := FileAttrs + faArchive;
  if CheckBox7.Checked then
    FileAttrs := FileAttrs + faAnyFile;
  with StringGrid1 do
  begin
    RowCount := 0;
    if SysUtils.FindFirst(Edit1.Text, FileAttrs, sr) = 0 then
    begin
      repeat
        if (sr.Attr and FileAttrs) = sr.Attr then
        begin
        RowCount := RowCount + 1;
        Cells[1,RowCount-1] := sr.Name;
        Cells[2,RowCount-1] := IntToStr(sr.Size);
        end;
      until FindNext(sr) <> 0;
```

```
      FindClose(sr);
    end;
  end;
end;
```

See Also

FindFirst (see page 22884)

159.3.103 SysUtils.FloatToCurr Function

Converts a floating-point value to a Currency value.

Pascal

```
function FloatToCurr(const Value: Extended): Currency;
```

C++

```
Currency FloatToCurr(const Extended Value);
```

File

SysUtils

Description

FloatToCurr returns the Currency value equivalent to Value. If Value is not within the valid range for a Currency value, FloatToCurr raises an EConvertError (see page 22724) exception.

See Also

TryFloatToCurr (see page 22981)

Double

159.3.104 SysUtils.FloatToDateTime Function

Converts a floating-point value to a TDateTime value.

Pascal

```
function FloatToDateTime(const Value: Extended): TDateTime;
```

C++

```
TDateTime FloatToDateTime(const Extended Value);
```

File

SysUtils

Description

FloatToDateTime returns the TDateTime value equivalent to Value. If Value is not within the valid range for TDateTime, FloatToDateTime raises an EConvertError (see page 22724) exception.

See Also

TryFloatToDateTime (see page 22982)

MinDateTime (see page 23052)

EConvertError (see page 22724)

159.3.105 SysUtils.FloatToDecimal Function

Converts a floating-point value to a decimal representation.

Pascal

```
procedure FloatToDecimal(var Result: TFloatRec; const Value; ValueType: TFloatValue;
Precision: Integer; Decimals: Integer);
```

C++

```
FloatToDecimal(TFloatRec Result, const  Value, TFloatValue ValueType, int Precision, int
Decimals);
```

File

SysUtils

Description

FloatToDecimal converts a floating-point value to a decimal representation that is suited for further formatting.

The Value parameter must be a variable of type Extended or Currency, as indicated by the ValueType parameter.

For values of type Extended, the Precision parameter specifies the requested number of significant digits in the result--the allowed range is 1..18. For values of type Currency, the Precision parameter is ignored, and the implied precision of the conversion is 19 digits.

The Decimals parameter specifies the requested maximum number of digits to the left of the decimal point in the result. Precision and Decimals together control how the result is rounded. To produce a result that always has a given number of significant digits regardless of the magnitude of the number, specify 9999 for the Decimals parameter.

159.3.106 SysUtils.FloatToStr Function

Converts a floating point value to a string.

Pascal

```
function FloatToStr(Value: Extended): string; overload;
function FloatToStr(Value: Extended; const FormatSettings: TFormatSettings): string; overload;
```

C++

```
AnsiString FloatToStr(Extended Value);
AnsiString FloatToStr(Extended Value, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

FloatToStr converts the floating-point value given by Value to its string representation. The conversion uses general number format with 15 significant digits.

For greater control over the formatting of the string, use the FloatToStrF (see page 22891) function.

The first form of FloatToStr is not thread-safe, because it uses localization information contained in global variables. The second

form of FloatToStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FloatToStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

Delphi Examples:

```
{
This example displays the compiler version number in the
text field.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit1.Text := FloatToStr(System.CompilerVersion);
end;
{
This example shows how to use the ArcTan and FloatToStr
functions.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ex : Extended;
  mystring : String;
begin
  ex := 16 * ArcTan(1/5) - 4 * ArcTan(1/239); // Machin's formula
  mystring := FloatToStr(ex);
  Edit2.Text := mystring;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  ex : Extended;
  myString : String;
begin
  ex := Pi;
  mystring := FloatToStr(ex);
  Edit1.Text := mystring;
end;
```

See Also

FloatToStrF (see page 22891)

FloatToText (see page 22892)

GetLocaleFormatSettings (see page 22913)

159.3.107 SysUtils.FloatToStrF Function

Converts a floating point value to a string, using a specified Format (see page 22897), Precision, and Digits.

Pascal

```
function FloatToStrF(Value: Extended; Format: TFloatFormat; Precision: Integer; Digits:
Integer): string; overload;
function FloatToStrF(Value: Extended; Format: TFloatFormat; Precision: Integer; Digits:
Integer; const FormatSettings: TFormatSettings): string; overload;
```

C++

```
AnsiString FloatToStrF(Extended Value, TFloatFormat Format, int Precision, int Digits);
AnsiString FloatToStrF(Extended Value, TFloatFormat Format, int Precision, int Digits, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

FloatToStrF converts the floating-point value given by Value to its string representation.

The Value parameter is the value to convert.

The Precision parameter specifies the precision of the given value. It should be 7 or less for values of type Single, 15 or less for values of type Double, and 18 or less for values of type Extended.

The Digits and Format (see page 22897) parameters together control how the value is formatted into a string. For details, see the description of TFloatFormat (see page 22999).

If the given value is a NAN (not-a-number), the resulting string is 'NAN'. If the given value is positive infinity, the resulting string is 'INF'. If the given value is negative infinity, the resulting string is '-INF'.

The first form of FloatToStrF is not thread-safe, because it uses localization information contained in global variables. The second form of FloatToStrF, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FloatToStrF, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

159.3.108 SysUtils.FloatToText Function

Converts a floating-point value to an unterminated character string, using a specified Format (see page 22897), Precision and Digits.

Pascal

```
function FloatToText(BufferArg: PChar; const Value; ValueType: TFloatValue; Format:
TFloatFormat; Precision: Integer; Digits: Integer): Integer; overload;
function FloatToText(BufferArg: PChar; const Value; ValueType: TFloatValue; Format:
TFloatFormat; Precision: Integer; Digits: Integer; const FormatSettings: TFormatSettings):
Integer; overload;
```

C++

```
int FloatToText(const char * BufferArg, const  Value, TFloatValue ValueType, TFloatFormat
Format, int Precision, int Digits);
int FloatToText(const char * BufferArg, const  Value, TFloatValue ValueType, TFloatFormat
Format, int Precision, int Digits, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

FloatToText converts the given floating-point value to its decimal representation using the specified format, precision, and digits. The Value parameter must be a variable of type Extended or Currency, as indicated by the ValueType parameter. The resulting string of characters is stored in the given buffer, and the returned value is the number of characters stored. The resulting string is not null-terminated.

The first form of FloatToText is not thread-safe, because it uses localization information contained in global variables. The second form of FloatToText, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FloatToText, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

See Also

FloatToStrF (see page 22891)

GetLocaleFormatSettings (see page 22913)

159.3.109 SysUtils.FloatToTextFmt Function

Converts a floating-point value to an unterminated character string, using a specified format.

Pascal

```
function FloatToTextFmt(Buf: PChar; const Value; ValueType: TFloatValue; Format: PChar):
Integer; overload;
function FloatToTextFmt(Buf: PChar; const Value; ValueType: TFloatValue; Format: PChar; const
FormatSettings: TFormatSettings): Integer; overload;
```

C++

```
int FloatToTextFmt(const char * Buf, const  Value, TFloatValue ValueType, const char * Format);
int FloatToTextFmt(const char * Buf, const  Value, TFloatValue ValueType, const char * Format,
const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Use FloatToTextFmt to format a floating-point value using a format string rather than using a predefined format. To use a predefined format, use the FloatToText (see page 22892) or FloatToStrF (see page 22891) function instead.

The Value parameter must be a variable of type Extended or Currency, as indicated by the ValueType parameter. The resulting string of characters is stored in the given buffer, and the returned value is the number of characters stored. The resulting string is not null-terminated.

Formats supported are described as part of the FormatFloat (see page 22904) function.

The first form of FloatToTextFmt is not thread-safe, because it uses localization information contained in global variables. The second form of FloatToTextFmt, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FloatToTextFmt, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

See Also

FormatFloat (see page 22904)

FloatToStrF (see page 22891)

FloatToText (see page 22892)

GetLocaleFormatSettings (see page 22913)

159.3.110 SysUtils.FmtLoadStr Function

Returns formatted output using a resourced format string.

Pascal

```
function FmtLoadStr(Ident: Integer; const Args: array of const): string;
```

C++

```
AnsiString FmtLoadStr(int Ident, const array of const Args);
```

File

SysUtils

Description

FmtLoadStr loads a string from a program's resource string table and uses that string, plus an array of arguments, as a parameter to Format (see page 22897).

Ident is the string resource ID of the desired format string.

Args is an array of arguments that replace the format specifiers in the string Ident designates.

Note: Args_Size is the index of the last element in the Args array (one less than the number of arguments).

FmtLoadStr returns the resourced string with the arguments applied. This is the same value that would result from calling Format (see page 22897), using the resourced string and the specified arguments.

See Also

Format (see page 22897)

159.3.111 SysUtils.FmtStr Function

Assembles a formatted string using a format string and an array of arguments.

Pascal

```
procedure FmtStr(var Result: string; const Format: string; const Args: array of const);
overload;
procedure FmtStr(var Result: string; const Format: string; const Args: array of const; const
FormatSettings: TFormatSettings); overload;
```

C++

```
FmtStr(AnsiString Result, const AnsiString Format, const array of const Args);
FmtStr(AnsiString Result, const AnsiString Format, const array of const Args, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

This function formats the series of arguments in the specified open (untyped) array.

StrResult returns the results of the operation.

Format (see page 22897) is the format string that determines how to generate StrResult. For information on the format strings, see Format (see page 22897) Strings.

Args is an array of arguments that replace the format specifiers in Format (see page 22897).

Note: Args_Size specifies the index of the last element of Args (one less than the number of elements).

The first form of FmtStr is not thread-safe, because it uses localization information contained in global variables. The second

form of FmtStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FmtStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

See Also

FormatBuf (see page 22900)

StrFmt (see page 22950)

StrLFmt (see page 22954)

Format (see page 22897)

FmtLoadStr (see page 22893)

GetLocaleFormatSettings (see page 22913)

159.3.112 SysUtils.ForceDirectories Function

Creates a new directory, including the creation of parent directories as needed.

Pascal

```
function ForceDirectories(Dir: string): Boolean;
```

C++

```
Boolean ForceDirectories(AnsiString Dir);
```

File

SysUtils

Description

ForceDirectories creates a new directory as specified in Dir, which must be a fully-qualified path name. If the directories given in the path do not yet exist, ForceDirectories attempts to create them.

ForceDirectories returns true if it successfully creates all necessary directories, false if it could not create a needed directory.

Warning: Important:Do not call ForceDirectories with an empty string. Doing so causes ForceDirectories to raise an exception.

Note: The FileCtrl (see page 13311) unit (Windows only) also contains a ForceDirectories function. However, the FileCtrl (see page 13311) version is deprecated, and the SysUtils (see page 22709) version preferred, even if the code does not need to be cross-platform.

C++ Examples:

```
/*
The following sample code sets the directory of
DirectoryListBox1 to c:\\WINDOWS when the form is created.
When Button1 is pressed, a subdirectory called
c:\\WINDOWS\\mytemp is added, but note that it is not updated
in DirectoryListBox1 until Button2 is pressed and Update is
called.
Note: A EInOutError will occur if you add a directory that
already exists.  Also, you must add the new directory to
your current directory to see the magic happen.  Also,
capitalization matters in C++ and the file spearators must
```

```
be "\\".
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ForceDirectories(Edit1->Text);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  DirectoryListBox1->Update();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  DirectoryListBox1->Directory = "c:\\WINDOWS";
}
/*
This example uses a label and a button on a form. When the
user clicks the button, all the directories along the
specified path that don't exist are created. The results are
reported in the caption of the label:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString Dir = "C:\\Apps\\Sales\\Local";
  if (ForceDirectories(Dir))
    Label1->Caption = Dir + " was created";
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, all the directories along the
specified path that don't exist are created. The results are
reported in the caption of the label:
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Dir: string;
begin
  Dir := 'C:\APPS\SALES\LOCAL';
  if SysUtils.ForceDirectories(Dir) then
    Label1.Caption := Dir + ' was created'
end;
```

See Also

ExtractFileDir (see page 22843)

GetCurrentDir (see page 22910)

RemoveDir (see page 22933)

SetCurrentDir (see page 22940)

CreateDir (see page 22817)

DirectoryExists (see page 22832)

SelectDirectory (see page 13619)

159.3.113 SysUtils.Format Function

Returns a formatted string assembled from a format string and an array of arguments.

Pascal

```
function Format(const Format: string; const Args: array of const): string; overload;
function Format(const Format: string; const Args: array of const; const FormatSettings:
TFormatSettings): string; overload;
```

C++

```
AnsiString Format(const AnsiString Format, const array of const Args);
AnsiString Format(const AnsiString Format, const array of const Args, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

The Format function formats the series of arguments in an open (untyped) array.

Format is the format string. For information on the format strings, see Format Strings, described in this topic.

Args is an array of arguments to apply to the format specifiers in Format. For C++, Args_Size specifies the index of the last element of Args (one less than the number of elements).

Format returns the results of applying the arguments in Args to the format string Format.

Two syntax forms are given for Format. The first form of Format is not thread-safe, because it uses localization information contained in global variables. The second form of Format, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of Format, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

Format Strings

Format strings specify required formats to general-purpose formatting routines. Format strings passed to the string formatting routines contain two types of objects -- literal characters and format specifiers. Literal characters are copied verbatim to the resulting string. Format specifiers fetch arguments from the argument list and apply formatting to them.

Format specifiers have the following form:

```
"%" [index ":"] ["-"] [width] ["." prec] type
```

A format specifier begins with a % character. After the % come the following elements, in this order:

An optional argument zero-offset index specifier (that is, the first item has index 0), `[index ":"]`

An optional left justification indicator, `["-"]`

An optional width specifier, `[width]`

An optional precision specifier, `["." prec]`

The conversion type character, `type`

The following table summarizes the possible values for type:

Value	Meaning
d	Decimal. The argument must be an integer value. The value is converted to a string of decimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has less digits, the resulting string is left-padded with zeros.
u	Unsigned decimal. Similar to 'd' but no sign is output.
e	Scientific. The argument must be a floating-point value. The value is converted to a string of the form "-d.ddd...E+ddd". The resulting string starts with a minus sign if the number is negative. One digit always precedes the decimal point. The total number of digits in the resulting string (including the one before the decimal point) is given by the precision specifier in the format string—a default precision of 15 is assumed if no precision specifier is present. The "E" exponent character in the resulting string is always followed by a plus or minus sign and at least three digits.
f	Fixed. The argument must be a floating-point value. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative. The number of digits after the decimal point is given by the precision specifier in the format string—a default of 2 decimal digits is assumed if no precision specifier is present.
g	General. The argument must be a floating-point value. The value is converted to the shortest possible decimal string using fixed or scientific format. The number of significant digits in the resulting string is given by the precision specifier in the format string—a default precision of 15 is assumed if no precision specifier is present. Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format.
n	Number. The argument must be a floating-point value. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The "n" format corresponds to the "f" format, except that the resulting string contains thousand separators.
m	Money. The argument must be a floating-point value. The value is converted to a string that represents a currency amount. The conversion is controlled by the CurrencyString (see page 23023), CurrencyFormat (see page 23022), NegCurrFormat (see page 23029), ThousandSeparator (see page 23032), DecimalSeparator (see page 23023), and CurrencyDecimals (see page 23022) global variables or their equivalent in a TFormatSettings (see page 23001) data structure. If the format string contains a precision specifier, it overrides the value given by the CurrencyDecimals (see page 23022) global variable or its TFormatSettings (see page 23001) equivalent.
p	Pointer. The argument must be a pointer value. The value is converted to an 8 character string that represents the pointers value in hexadecimal.
s	String. The argument must be a character, a string, or a PChar value. The string or character is inserted in place of the format specifier. The precision specifier, if present in the format string, specifies the maximum length of the resulting string. If the argument is a string that is longer than this maximum, the string is truncated.
x	Hexadecimal. The argument must be an integer value. The value is converted to a string of hexadecimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has fewer digits, the resulting string is left-padded with zeros.

Conversion characters may be specified in uppercase as well as in lowercase—both produce the same results.

For all floating-point formats, the actual characters used as decimal and thousand separators are obtained from the DecimalSeparator (see page 23023) and ThousandSeparator (see page 32032) global variables or their TFormatSettings (see page 23001) equivalent.

Index, width, and precision specifiers can be specified directly using decimal digit string (for example "%10d"), or indirectly using an asterisk character (for example "%*.*f"). When using an asterisk, the next argument in the argument list (which must be an integer value) becomes the value that is actually used. For example,

```
Format ('%*.*f', [8, 2, 123.456]);
```

is equivalent to

```
Format ('%8.2f', [123.456]);
```

Similarly:

```
TVarRec args[3] = {8,2,123.456};
Format ("%*.*f", args, 2);
```

is equivalent to

```
TVarRec args[1] = {123.456};
Format ("%8.2f", args, 0);
```

A width specifier sets the minimum field width for a conversion. If the resulting string is shorter than the minimum field width, it is padded with blanks to increase the field width. The default is to right-justify the result by adding blanks in front of the value, but if the format specifier contains a left-justification indicator (a "-" character preceding the width specifier), the result is left-justified by adding blanks after the value.

An index specifier sets the current argument list index to the specified value. The index of the first argument in the argument list is 0. Using index specifiers, it is possible to format the same argument multiple times. For example "Format('%d %d %0:d %1:d', [10, 20])" produces the string '10 20 10 20'.

Note: Setting the index specifier affects all subsequent formatting. That is, Format('%d %d %d %0:d %d', [1, 2, 3, 4]) returns '1 2 3 1 2', not '1 2 3 1 4'. To get the latter result, you must use Format('%d %d %d %0:d %3:d', [1, 2, 3, 4])

C++ Examples:

```
/*
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Files
can be saved to and from the app directory by using relative
paths.
*/
void __fastcall TForm1::Save1Click(TObject *Sender)
{
  AnsiString NewFileName =
    ExtractFilePath(Application->ExeName) +
    ExtractFileName(Edit1->Text);
  AnsiString OldFileName =
    ExtractFilePath(Application->ExeName) +
    ExtractFileName(Edit2->Text);
  AnsiString Msg = Format("Copy %s to %s", ARRAYOFCONST((OldFileName, NewFileName)));
  if (MessageDlg(Msg, mtCustom, TMsgDlgButtons() << mbOK << mbCancel, 0) == mrOk)
  {
    TFileStream *OldFile = new TFileStream(OldFileName, fmOpenRead);
    try
    {
      TFileStream *NewFile = new TFileStream(NewFileName, fmCreate);
      try
      {
        NewFile->CopyFrom(OldFile, OldFile->Size);
      }
      __finally
      {
        FreeAndNil(&NewFile);
      }
    }
    __finally
    {
      FreeAndNil(&OldFile);
    }
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Label3->Caption = "Current directory: " + ExtractFilePath(Application->ExeName);
}
```

Delphi Examples:

```
{
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Files
can be saved to and from the app directory by using relative
paths.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Label3.Caption:= 'Current directory: ' + ExtractFilePath(Application.ExeName);
end;

procedure TForm1.Save1Click(Sender: TObject);
var
  NewFileName, OldFileName: string;
  Msg: string;
  NewFile: TFileStream;
  OldFile: TFileStream;
begin
  NewFileName :=
    ExtractFilePath(Application.ExeName) +
    ExtractFileName(Edit1.Text);
  OldFileName :=
    ExtractFilePath(Application.ExeName) +
    ExtractFileName(Edit2.Text);
  Msg := Format('Copy %s to %s?', [OldFileName, NewFileName]);
  if MessageDlg(Msg, mtCustom, mbOKCancel, 0) = mrOK then
  begin
    OldFile := TFileStream.Create(
      OldFileName, fmOpenRead or fmShareDenyWrite);
    try
      NewFile := TFileStream.Create(
        NewFileName, fmCreate or fmShareDenyRead);
      try
        NewFile.CopyFrom(OldFile, OldFile.Size);
      finally
        FreeAndNil(NewFile);
      end;
    finally
      FreeAndNil(OldFile);
    end;
  end;
end;
```

See Also

Date (see page 22820)

159.3.114 SysUtils.FormatBuf Function

Formats the arguments from an array, placing the result in a buffer.

Pascal

```
function FormatBuf(var Buffer; BufLen: Cardinal; const Format; FmtLen: Cardinal; const Args:
array of const): Cardinal; overload;
function FormatBuf(var Buffer; BufLen: Cardinal; const Format; FmtLen: Cardinal; const Args:
array of const; const FormatSettings: TFormatSettings): Cardinal; overload;
```

C++

```
unsigned FormatBuf( Buffer, unsigned BufLen, const  Format, unsigned FmtLen, const array of
const Args);
unsigned FormatBuf( Buffer, unsigned BufLen, const  Format, unsigned FmtLen, const array of
```

```
const Args, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

This function formats the series of arguments in the specified open array into a text buffer.

Buffer is the text buffer that receives the results.

BufLen is the number of bytes in Buffer.

Format (see page 22897) is the format string that determines how arguments are formatted. For information on the format strings, see Format (see page 22897) Strings.

FmtLen is the number of bytes in Format (see page 22897).

Args are the arguments that replace the format specifiers in Format (see page 22897).

Note: Args_Size is the index of the last element in Args (one less than the number of arguments).

FormatBuf function returns the length, in bytes, of the resulting string.

The first form of FormatBuf is not thread-safe, because it uses localization information contained in global variables. The second form of FormatBuf, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FormatBuf, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

See Also

GetLocaleFormatSettings (see page 22913)

159.3.115 SysUtils.FormatCurr Function

Formats a Currency object.

Pascal

```
function FormatCurr(const Format: string; Value: Currency): string; overload;
function FormatCurr(const Format: string; Value: Currency; const FormatSettings:
TFormatSettings): string; overload;
```

C++

```
AnsiString FormatCurr(const AnsiString Format, Currency Value);
AnsiString FormatCurr(const AnsiString Format, Currency Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

FormatCurr formats the Currency object given by Value using the format string given by Format (see page 22897).

For information on the format strings, see Format (see page 22897) Strings.

The first form of FormatCurr is not thread-safe, because it uses localization information contained in global variables. The second form of FormatCurr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FormatCurr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

See Also

CurrToStr (see page 22819)

CurrToStrF (see page 22820)

StrToCurr (see page 22962)

FormatFloat (see page 22904)

GetLocaleFormatSettings (see page 22913)

159.3.116 SysUtils.FormatDateTime Function

Formats a TDateTime value.

Pascal

```
function FormatDateTime(const Format: string; DateTime: TDateTime): string; overload;
function FormatDateTime(const Format: string; DateTime: TDateTime; const FormatSettings:
TFormatSettings): string; overload;
```

C++

```
AnsiString FormatDateTime(const AnsiString Format, TDateTime DateTime);
AnsiString FormatDateTime(const AnsiString Format, TDateTime DateTime, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

FormatDateTime formats the TDateTime value given by DateTime using the format given by Format (see page 22897). See the table below for information about the supported format strings.

If the string specified by the Format (see page 22897) parameter is empty, the TDateTime value is formatted as if a 'c' format specifier had been given.

The first form of FormatDateTime is not thread-safe, because it uses localization information contained in global variables. The second form of FormatDateTime, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FormatDateTime, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

Date (see page 22820) Time (see page 22976) Format (see page 22897) Strings are composed from specifiers that represent values to be inserted into the formatted string. Some specifiers (such as "d"), simply format numbers or strings. Other specifiers (such as "/") refer to locale-specific strings from global variables.

In the following table, specifiers are given in lower case. Case is ignored in formats, except for the "am/pm" and "a/p" specifiers.

Specifier	Displays
c	Displays the date using the format given by the ShortDateFormat (see page 23030) global variable, followed by the time using the format given by the LongTimeFormat (see page 23028) global variable. The time is not displayed if the date-time value indicates midnight precisely.
d	Displays the day as a number without a leading zero (1-31).
dd	Displays the day as a number with a leading zero (01-31).

ddd	Displays the day as an abbreviation (Sun-Sat) using the strings given by the ShortDayNames (see page 23031) global variable.
dddd	Displays the day as a full name (Sunday-Saturday) using the strings given by the LongDayNames (see page 23027) global variable.
ddddd	Displays the date using the format given by the ShortDateFormat (see page 23030) global variable.
dddddd	Displays the date using the format given by the LongDateFormat (see page 23027) global variable.
e	(Windows only) Displays the year in the current period/era as a number without a leading zero (Japanese, Korean and Taiwanese locales only).
ee	(Windows only) Displays the year in the current period/era as a number with a leading zero (Japanese, Korean and Taiwanese locales only).
g	(Windows only) Displays the period/era as an abbreviation (Japanese and Taiwanese locales only).
gg	(Windows only) Displays the period/era as a full name. (Japanese and Taiwanese locales only).
m	Displays the month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mm	Displays the month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mmm	Displays the month as an abbreviation (Jan-Dec) using the strings given by the ShortMonthNames (see page 23031) global variable.
mmmm	Displays the month as a full name (January-December) using the strings given by the LongMonthNames (see page 23028) global variable.
yy	Displays the year as a two-digit number (00-99).
yyyy	Displays the year as a four-digit number (0000-9999).
h	Displays the hour without a leading zero (0-23).
hh	Displays the hour with a leading zero (00-23).
n	Displays the minute without a leading zero (0-59).
nn	Displays the minute with a leading zero (00-59).
s	Displays the second without a leading zero (0-59).
ss	Displays the second with a leading zero (00-59).
z	Displays the millisecond without a leading zero (0-999).
zzz	Displays the millisecond with a leading zero (000-999).
t	Displays the time using the format given by the ShortTimeFormat (see page 23031) global variable.
tt\	Displays the time using the format given by the LongTimeFormat (see page 23028) global variable.
am/pm	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'am' for any hour before noon, and 'pm' for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'a' for any hour before noon, and 'p' for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm	Uses the 12-hour clock for the preceding h or hh specifier, and displays the contents of the TimeAMString (see page 23033) global variable for any hour before noon, and the contents of the TimePMString (see page 23033) global variable for any hour after noon.
/	Displays the date separator character given by the DateSeparator (see page 23023) global variable.
:	Displays the time separator character given by the TimeSeparator (see page 23033) global variable.
'xx'/"xx"	Characters enclosed in single or double quotes are displayed as-is, and do not affect formatting.

C++ Examples:

```
/*
The following example uses FormatDateTime to set the string
variable S to a sentence indicating a meeting time in 3
hours. The sentence has the form "The meeting is on
Wednesday, February 15, 1995 at 2:30 PM."
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString S = "The meeting is on ";
  S = S + FormatDateTime("dddd, mmmm d, yyyy ' at ' hh:mm AM/PM", Now() + 0.125);
  Label1->Caption = S;
}
```

Delphi Examples:

```
{
The following example uses FormatDateTime to set the string
variable S to a sentence indicating a meeting time in 3
hours. The sentence has the form "The meeting is on
Wednesday, February 15, 1995 at 2:30 PM."
}
procedure TForm1.Button1Click(Sender: TObject);
var S : string;
begin
  S := SysUtils.FormatDateTime(
    '"The meeting is on " dddd, mmmm d, yyyy, " at " hh:mm AM/PM', Now + 0.125);
  Label1.Caption := S;
end;
```

159.3.117 SysUtils.FormatFloat Function

Formats a floating point value.

Pascal

```
function FormatFloat(const Format: string; Value: Extended): string; overload;
function FormatFloat(const Format: string; Value: Extended; const FormatSettings:
TFormatSettings): string; overload;
```

C++

```
AnsiString FormatFloat(const AnsiString Format, Extended Value);
AnsiString FormatFloat(const AnsiString Format, Extended Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

FormatFloat formats the floating-point value given by Value using the format string given by Format (see page 22897). The following format specifiers are supported in the format string:

Specifier	Represents
0	Digit place holder. If the value being formatted has a digit in the position where the '0' appears in the format string, then that digit is copied to the output string. Otherwise, a '0' is stored in that position in the output string.
#	Digit placeholder. If the value being formatted has a digit in the position where the '#' appears in the format string, then that digit is copied to the output string. Otherwise, nothing is stored in that position in the output string.

.	Decimal point. The first '.' character in the format string determines the location of the decimal separator in the formatted value; any additional '.' characters are ignored. The actual character used as a the decimal separator in the output string is determined by the DecimalSeparator (see page 23023) global variable or its TFormatSettings (see page 23001) equivalent.
,	Thousand separator. If the format string contains one or more ',' characters, the output will have thousand separators inserted between each group of three digits to the left of the decimal point. The placement and number of ',' characters in the format string does not affect the output, except to indicate that thousand separators are wanted. The actual character used as a the thousand separator in the output is determined by the ThousandSeparator (see page 23032) global variable or its TFormatSettings (see page 23001) equivalent.
E+	Scientific notation. If any of the strings 'E+', 'E-', 'e+', or 'e-' are contained in the format string, the number is formatted using scientific notation. A group of up to four '0' characters can immediately follow the 'E+', 'E-', 'e+', or 'e-' to determine the minimum number of digits in the exponent. The 'E+' and 'e+' formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The 'E-' and 'e-' formats output a sign character only for negative exponents.
'xx'/"xx"	Characters enclosed in single or double quotes are output as-is, and do not affect formatting.
;	Separates sections for positive, negative, and zero numbers in the format string.

The locations of the leftmost '0' before the decimal point in the format string and the rightmost '0' after the decimal point in the format string determine the range of digits that are always present in the output string.

The number being formatted is always rounded to as many decimal places as there are digit placeholders ('0' or '#') to the right of the decimal point. If the format string contains no decimal point, the value being formatted is rounded to the nearest whole number.

If the number being formatted has more digits to the left of the decimal separator than there are digit placeholders to the left of the '.' character in the format string, the extra digits are output before the first digit placeholder.

To allow different formats for positive, negative, and zero values, the format string can contain between one and three sections separated by semicolons.

One section: The format string applies to all values.

Two sections: The first section applies to positive values and zeros, and the second section applies to negative values.

Three sections: The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If the section for negative values or the section for zero values is empty, that is if there is nothing between the semicolons that delimit the section, the section for positive values is used instead.

If the section for positive values is empty, or if the entire format string is empty, the value is formatted using general floating-point formatting with 15 significant digits, corresponding to a call to FloatToStrF (see page 22891) with the ffGeneral format. General floating-point formatting is also used if the value has more than 18 digits to the left of the decimal point and the format string does not specify scientific notation.

The first form of FormatFloat is not thread-safe, because it uses localization information contained in global variables. The second form of FormatFloat, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FormatFloat, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

C++ Examples:

```
/*
The following example uses two buttons, a string grid, and
text edit on a form. When the conversion button is clicked,
the values across the top of the string grid are converted
using the formats along the left side.  Select a cell and
use the Alter Cell button and text edit to change the
values converted and the conversion formats.
```

```
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int x = 1; x < StringGrid1->ColCount; x++)
  {
    for (int y = 1; y < StringGrid1->RowCount; y++)
    {
      StringGrid1->Cells[x][y] = FormatFloat(
      StringGrid1->Cells[0][y], StrToFloat(StringGrid1->Cells[x][0]));
    }
  }
}

int currCellCol, currCellRow;

void __fastcall TForm1::StringGrid1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  StringGrid1->MouseToCell(X, Y, currCellCol, currCellRow);
}

// Alter Cell
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  StringGrid1->Cells[currCellCol][currCellRow] = Edit1->Text;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "1234";
  StringGrid1->Cells[2][0] = "-1234";
  StringGrid1->Cells[3][0] = "0.5";
  StringGrid1->Cells[4][0] = "0";
  StringGrid1->Cells[0][1] = "";
  StringGrid1->Cells[0][2] = "0";
  StringGrid1->Cells[0][3] = "0.00";
  StringGrid1->Cells[0][4] = "#.##";
  StringGrid1->Cells[0][5] = "#,##0.00";
  StringGrid1->Cells[0][6] = "#,##0.00;(#,##0.00)";
  StringGrid1->Cells[0][7] = "#,##0.00;;Zero";
  StringGrid1->Cells[0][8] = "0.000E+00";
  StringGrid1->Cells[0][9] = "#.###E-0";
}
```

Delphi Examples:

```
{
The following example uses two buttons, a string grid, and
text edit on a form. When the conversion button is clicked,
the values across the top of the string grid are converted
using the formats along the left side.  Select a cell and
use the Alter Cell button and text edit to change the
values converted and the conversion formats.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  X, Y, I: Integer;
begin
  for X := 1 to StringGrid1.ColCount - 1 do
  begin
    for Y := 0 to StringGrid1.RowCount - 1 do
    begin
      StringGrid1.Cells[X,Y] := SysUtils.FormatFloat(
        StringGrid1.Cells[0,Y], StrToFloat(StringGrid1.Cells[X,0]));
    end;
  end;
```

```
end;

var currCellCol, currCellRow: Integer;

// Alter Cell
procedure TForm1.Button2Click(Sender: TObject);
begin
  StringGrid1.Cells[currCellCol,currCellRow] := Edit1.Text;
end;

procedure TForm1.StringGrid1MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  StringGrid1.MouseToCell(X, Y, &currCellCol, &currCellRow);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  StringGrid1.Cells[1,0] := '1234';
  StringGrid1.Cells[2,0] := '-1234';
  StringGrid1.Cells[3,0] := '0.5';
  StringGrid1.Cells[4,0] := '0';
  StringGrid1.Cells[0,1] := '';
  StringGrid1.Cells[0,2] := '0';
  StringGrid1.Cells[0,3] := '0.00';
  StringGrid1.Cells[0,4] := '#.##';
  StringGrid1.Cells[0,5] := '#,##0.00';
  StringGrid1.Cells[0,6] := '#,##0.00;(#,##0.00)';
  StringGrid1.Cells[0,7] := '#,##0.00;;Zero';
  StringGrid1.Cells[0,8] := '0.000E+00';
  StringGrid1.Cells[0,9] := '#.###E-0';
end;
{
The following example displays in a dialog the disk free
space of the current drive with commas as thousand separators.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage(FormatFloat('#,##0 bytes', DiskFree(0)));
end;
```

See Also

FloatToStrF (see page 22891)

FloatToText (see page 22892)

FloatToTextFmt (see page 22893)

GetLocaleFormatSettings (see page 22913)

159.3.118 SysUtils.FreeAndNil Function

Frees an object reference and replaces the reference with nil (Delphi) or NULL (C++).

Pascal

```
procedure FreeAndNil(var Obj);
```

C++

```
FreeAndNil( Obj);
```

File

SysUtils

Description

Use FreeAndNil to ensure that a variable is nil (Delphi) or NULL (C++) after you free the object it references. Pass any variable that represents an object as the Obj parameter.

Warning: Obj must be an instance of a TObject descendant.

C++ Examples:

```
/*
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString NewFileName, OldFileName;
  TFileStream *NewFile, *OldFile;
  SavePictureDialog1->DefaultExt = GraphicExtension(__classid(Graphics::TBitmap));
  SavePictureDialog1->Filter = GraphicFilter(__classid(Graphics::TBitmap));
  OldFileName = "factory.bmp";
  if (SavePictureDialog1->Execute())
  {
    NewFileName = SavePictureDialog1->FileName;
    OldFile = new TFileStream(OldFileName, (fmOpenRead | fmShareDenyWrite));
    try
    {
      NewFile = new TFileStream(NewFileName, (fmCreate | fmShareDenyRead));
      try
      {
        NewFile->CopyFrom(OldFile, OldFile->Size);
      }
      __finally
      {
        FreeAndNil(&NewFile);
      };
    }
    __finally
    {
      FreeAndNil(&OldFile);
    };
  };
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Image1->Picture->LoadFromFile("../factory.bmp");
}
```

Delphi Examples:

```
{
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
var NewFileName, OldFileName: string;
    NewFile, OldFile: TFileStream;
begin
  SavePictureDialog1.DefaultExt := GraphicExtension(TBitmap);
  SavePictureDialog1.Filter := GraphicFilter(TBitmap);
  OldFileName := 'factory.bmp';
  if SavePictureDialog1.Execute then
  begin
    NewFileName := SavePictureDialog1.FileName;
    OldFile := TFileStream.Create(
      OldFileName, fmOpenRead or fmShareDenyWrite);
    try
      NewFile := TFileStream.Create(
        NewFileName, fmCreate or fmShareDenyRead);
      try
        NewFile.CopyFrom(OldFile, OldFile.Size);
      finally
        FreeAndNil(NewFile);
      end;
    finally
      FreeAndNil(OldFile);
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Picture.LoadFromFile('factory.bmp');
end;
```

See Also

TObject

159.3.119 SysUtils.GDAL Function

Get Delphi Access Licence.

Pascal

```
function GDAL: LongWord;
```

C++

```
LongWord GDAL();
```

File

SysUtils

Description

GDAL retreives the access licences resource. It checks if it is valid, if not an Access Licence Violation exception with the message 'Application is not licenced to use this feature' is raised.

The returned value is the decrypted first Access Licence (AL1).

See Also

RCS (see page 22933)

RPR (see page 22938)

159.3.120 SysUtils.GetCurrentDir Function

Returns the name of the current directory.

Pascal

```
function GetCurrentDir: string;
```

C++

```
AnsiString GetCurrentDir();
```

File

SysUtils

Description

GetCurrentDir returns the fully qualified name of the current directory.

Delphi Examples:

```
{
This example shows how to change and display the current
directory through text.  This example requires that a drive
combo box, a file list box, and a directory list box are on
a form. Add this code as the OnChange event handler for the
drive combo box and the OnChange event handler for the
directory list box.  When the user changes the drive using
the combo box, the directory list box and file list box will
update to reflect the new drive and the current directory on
that drive.  When the user double clicks on a directory in
the file list box will update to reflect the new directory.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  System.ChDir(Edit1.Text);
  if IOResult <> 0 then
    MessageDlg('Cannot find directory', mtWarning, [mbOk], 0);
  Edit2.Text := GetCurrentDir;
  FileListBox1.Directory := GetCurrentDir;
  DirectoryListBox1.Directory := GetCurrentDir;
end;

procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
  Edit2.Text := GetCurrentDir;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
  Edit2.Text := GetCurrentDir;
end;
```

See Also

GetDir (see page 22467)

CreateDir (see page 22817)

ExtractFileDir (see page 22843)

RemoveDir (see page 22933)

SetCurrentDir (see page 22940)

159.3.121 SysUtils.GetEnvironmentVariable Function

Retrieves an environment variable value.

Pascal

```
function GetEnvironmentVariable(const Name: string): string; overload;
```

C++

```
AnsiString GetEnvironmentVariable(const AnsiString Name);
```

File

SysUtils

Description

Call GetEnvironmentVariable to retrieve the value of an environment variable, passed as `Name`, for the current process.

Note:

The case of the specified environment variable `Name` is ignored.

The following table defines the standard environment variables for Microsoft Windows.

Environment variable name	Environment value
ALLUSERSPROFILE	Generic user profile.
APPDATA	Path of the application data folder.
CLIENTNAME	Name of Client machine.
COMMONPROGRAMFILES	Path of common program files folder.
COMPUTERNAME	Name of Computer code is running on.
COMSPEC	Path of the cmd.exe program.
HOMEDRIVE	Current home drive designation, such as 'C:'
HOMEPATH	Path to current location for document storage.
LOGONSERVER	
NUMBER_OF_PROCESSORS	Number of processors on current machine.
OS	Base name of the Operating System (see page 22079). Note that Windows XP is given as Windows_NT.
PATH	The current program path.
PATHEXT	
PROCESSOR_ARCHITECTURE	Type of CPU architecture. For example, X86 for Intel Pentium processors.
PROCESSOR_IDENTIFIER	
PROCESSOR_LEVEL	More detailed description of the CPU architecture.
PROCESSOR_REVISION	Processor revision level.
PROGRAMFILES	Path of the program files folder.

SESSIONNAME	Name of the current OS session.
SYSTEMDRIVE	Drive the OS operates from.
SYSTEMROOT	
TEMP	Path of the temporary files folder.
TMP	
USERDOMAIN	
USERNAME	Name of the current user.
USERPROFILE	Path of the folder holding the current user's information.
WINDIR	Path of the Windows folder.

159.3.122 SysUtils.GetFileVersion Function

Returns file version.

Pascal

```
function GetFileVersion(const AFileName: string): Cardinal;
```

C++

```
unsigned GetFileVersion(const AnsiString AFileName);
```

File

SysUtils

Description

GetFileVersion returns the most significant 32 bits of a file binary version number. AFileName specifies the name of the file, which is searched using the same path as the LoadLibrary function.

GetFileVersion is only available on Windows and must not be used in cross-platform code.

159.3.123 SysUtils.GetFormatSettings Function

Resets the date and number format parameters to initial values.

Pascal

```
procedure GetFormatSettings;
```

C++

```
GetFormatSettings();
```

File

SysUtils

Description

GetFormatSettings resets all date and number format variables to their initial values.

Note: On Windows, GetFormatSettings is called automatically in response to WM_WININICHANGE messages.

159.3.124 SysUtils.GetLocaleChar Function

Retrieves the first character of a localisation value.

Pascal

```
function GetLocaleChar(Locale: Integer; LocaleType: Integer; Default: Char): Char; platform;
```

C++

```
Char GetLocaleChar(int Locale, int LocaleType, Char Default);
```

File

SysUtils

Description

GetLocaleChar retrieves the first character of the passed LocaleType value, or the passed Default value if not found. The actual value is determined by the location specified by Locale. For example, on Windows, $0809 is the Locale for United Kingdom English, and $0409 is the Locale for US English. For the LOCALE_SCURRENCY LocaleType, these locales would yield '£' and '$' respectively.

Note: This is an OS specific function.

See Also

GetLocaleStr (see page 22913)

159.3.125 SysUtils.GetLocaleFormatSettings Function

Populates a TFormatSettings (see page 23001) data structure.

Pascal

```
procedure GetLocaleFormatSettings(LCID: Integer; var FormatSettings: TFormatSettings);
```

C++

```
GetLocaleFormatSettings(int LCID, TFormatSettings FormatSettings);
```

File

SysUtils

Description

GetLocaleFormatSettings populates FormatSettings based on the Windows locale ID specified in LCID. A TFormatSettings (see page 23001) data structure must be initialized by a call to GetLocaleFormatSettings before it can be used with any string formatting routines.

159.3.126 SysUtils.GetLocaleStr Function

Retrieves a localisation value.

Pascal

```
function GetLocaleStr(Locale: Integer; LocaleType: Integer; const Default: string): string;
platform;
```

C++

```
AnsiString GetLocaleStr(int Locale, int LocaleType, const AnsiString Default);
```

File

SysUtils

Description

GetLocaleStr retrieves the passed LocaleType value, or the passed Default value if not found. The actual value is determined by the location specified by Locale. For example, on Windows, $040c is the Locale for France French, and $0409 is the Locale for US English. For the LOCALE_SMONTHNAME1 LocaleType, these locales would yield 'janvier' and 'January' respectively.

Note: This is an OS specific function.

See Also

GetLocaleChar (see page 22913)

159.3.127 SysUtils.GetModuleName Function

Returns the fully qualified name for a module, given its handle.

Pascal

```
function GetModuleName(Module: HMODULE): string;
```

C++

```
AnsiString GetModuleName(HMODULE Module);
```

File

SysUtils

Description

Call GetModuleName to obtain the fully qualified name of a module, given its handle.

Module is the handle of the module whose file name is required.

Warning: :On Linux, GetModuleName is not always successful. Depending on the way the application is called, the presence of the /proc file system, and other factors, there are times when GetModuleName may return a name that is not fully qualified or in some cases, no file name at all. Be sure to check the results of this function before using the returned value.

C++ Examples:

```
/*
This example requires a button and a label.  When the user clicks
the button, the GetModuleName function retieves the path and name
of the project executable using the file handle HInstance.
*/
AnsiString __fastcall GetUsersFileName(void)
{
  return GetModuleName(unsigned(HInstance));
}
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = GetUsersFileName();
}
```

Delphi Examples:

```
{
This example requires a button and a label.  When the user clicks
the button, the GetModuleName function retieves the path and name
of the project executable using the file handle HInstance.
}
function GetUsersFileName: string;
begin
  Result := SysUtils.GetModuleName(HInstance);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := GetUsersFileName;
end;
```

159.3.128 SysUtils.GetPackageDescription Function

Returns the description resource from a package.

Pascal

```
function GetPackageDescription(ModuleName: PChar): string;
```

C++

```
AnsiString GetPackageDescription(const char * ModuleName);
```

File

SysUtils

Description

Call GetPackageDescription to obtain the description stored with the named package. If the package does not have a description resource, GetPackageDescription returns an empty string.

See Also

GetPackageInfo (see page 22915)

159.3.129 SysUtils.GetPackageInfo Function

Enumerates all the units and required packages for a package.

Pascal

```
procedure GetPackageInfo(Module: HMODULE; Param: Pointer; var Flags: Integer; InfoProc:
TPackageInfoProc);
```

C++

```
GetPackageInfo(HMODULE Module, void * Param, int Flags, TPackageInfoProc InfoProc);
```

File

SysUtils

Description

Call GetPackageInfo to process the information in a package's information table. The InfoProc callback is called for every unit included in the package and for every package required by the package. Module is the module handle for the package. GetPackageInfo passes any application-supplied information passed in the Param parameter to the InfoProc callback. The package's information flags are passed to the InfoProc and returned in the Flags parameter.

See Also

LoadPackage (see page 22927)

GetPackageDescription (see page 22915)

159.3.130 SysUtils.GetTime Function

Returns the current time.

Pascal

```
function GetTime: TDateTime;
```

C++

```
TDateTime GetTime();
```

File

SysUtils

Description

GetTime and Time (see page 22976) return the current time as a TDateTime value. The two functions are completely equivalent.

Note: Time (see page 22976) clashes with an Xlib function of the same name. Use GetTime in Linux and cross-platform code.

See Also

Time (see page 22976)

Date (see page 22820)

DecodeTime (see page 22830)

Now (see page 22930)

TimeToStr (see page 22977)

159.3.131 SysUtils.GUIDToString Function

Converts a class identifier GUID to a string.

Pascal

```
function GUIDToString(const GUID: TGUID): string;
```

C++

```
AnsiString GUIDToString(const TGUID GUID);
```

File

SysUtils

Description

GUIDToString converts a GUID into a string of printable characters. Each GUID always converts to a unique string.

If GUIDToString succeeds it returns a string for the requested GUID. If GUIDToString fails, it raises EConvertError (see page 22724).

Delphi Examples:

```
{
OleCheck, GuidToString example
The following example manufactures a GUID and displays it in a dialog box after translating it
to a string.
}
uses ActiveX, ComObj;
procedure TForm1.Button1Click(Sender: TObject);
var
  G: TGUID;
begin
  OleCheck(CoCreateGuid(G));
  ShowMessage(SysUtils.GuidToString(G));
end;
```

See Also

StringToGUID (see page 22952)

EConvertError (see page 22724)

159.3.132 SysUtils.IncAMonth Function

Increments date data by one month.

Pascal

```
procedure IncAMonth(var Year: Word; var Month: Word; var Day: Word; NumberOfMonths: Integer =
1);
```

C++

```
IncAMonth(Word Year, Word Month, Word Day, int NumberOfMonths = 1);
```

File

SysUtils

Description

IncAMonth modifies Year, Month, and Day parameters so the date they describe is incremented by NumberOfMonths months. NumberOfMonths can be negative, to return a date N months previous.

If the input day of month is greater than the last day of the resulting month, the day is set to the last day of the resulting month.

See Also

IncMonth (see page 22919)

DecodeDate (see page 22828)

EncodeDate (see page 22836)

ReplaceDate (see page 22936)

IncDay (see page 6643)

IncHour (see page 6644)

IncMilliSecond (see page 6645)

IncMinute (see page 6645)

IncSecond (see page 6646)

IncWeek (see page 6647)

IncYear (see page 6647)

159.3.133 SysUtils.IncludeTrailingBackslash Function

Ensures path name ends with delimiter

Pascal

```
function IncludeTrailingBackslash(const S: string): string; platform;
```

C++

```
AnsiString IncludeTrailingBackslash(const AnsiString S);
```

File

SysUtils

Description

IncludeTrailingBackslash ensures that a path name ends with a trailing path delimiter ('\" on Windows, '/' on Linux). This function is included for backward-compatibility only. New programs should use IncludeTrailingPathDelimiter (see page 22918).

Note: IncludeTrailingBackslash works with multibyte character sets.

See Also

ExcludeTrailingBackslash (see page 22839)

IsPathDelimiter (see page 22925)

159.3.134 SysUtils.IncludeTrailingPathDelimiter Function

Ensures path name ends with delimiter.

Pascal

```
function IncludeTrailingPathDelimiter(const S: string): string;
```

C++

```
AnsiString IncludeTrailingPathDelimiter(const AnsiString S);
```

File

SysUtils

Description

IncludeTrailingPathDelimiter ensures that a path name ends with a trailing path delimiter ('\" on Windows, '/' on Linux). If S already ends with a trailing delimiter character, it is returned unchanged; otherwise S with appended delimiter character is returned.

Note: IncludeTrailingPathDelimiter works with multibyte character sets.

See Also

ExcludeTrailingPathDelimiter (see page 22840)

IsPathDelimiter (see page 22925)

159.3.135 SysUtils.IncMonth Function

Returns a date shifted by a specified number of months.

Pascal

```
function IncMonth(const DateTime: TDateTime; NumberOfMonths: Integer = 1): TDateTime;
```

C++

```
TDateTime IncMonth(const TDateTime DateTime, int NumberOfMonths = 1);
```

File

SysUtils

Description

IncMonth returns the value of the Date (see page 22820) parameter, incremented by NumberOfMonths months. NumberOfMonths can be negative, to return a date N months previous.

If the input day of month is greater than the last day of the resulting month, the day is set to the last day of the resulting month. The time of day specified by the Date (see page 22820) parameter is copied to the result.

See Also

IncAMonth (see page 22917)

DecodeDate (see page 22828)

EncodeDate (see page 22836)

ReplaceDate (see page 22936)

IncDay (see page 6643)

IncHour (see page 6644)

IncMilliSecond (see page 6645)

IncMinute (see page 6645)

IncSecond (see page 6646)

IncWeek (see page 6647)

IncYear (see page 6647)

159.3.136 SysUtils.InitializePackage Function

Validates and initializes a given package.

Pascal

```
procedure InitializePackage(Module: HMODULE); overload;
procedure InitializePackage(Module: HMODULE; AValidatePackage: TValidatePackageProc); overload;
```

C++

```
InitializePackage(HMODULE Module);
InitializePackage(HMODULE Module, TValidatePackageProc AValidatePackage);
```

File

SysUtils

Description

InitializePackage checks for duplicate units in a package and calls the initialization blocks of the contained units.

Note: InitializePackage is called internally by the LoadPackage (see page 22927) function.

See Also

LoadPackage (see page 22927)

159.3.137 SysUtils.IntToHex Function

Returns the hex representation of an integer.

Pascal

```
function IntToHex(Value: Integer; Digits: Integer): string; overload;
function IntToHex(Value: Int64; Digits: Integer): string; overload;
```

C++

```
AnsiString IntToHex(int Value, int Digits);
AnsiString IntToHex(Int64 Value, int Digits);
```

File

SysUtils

Description

IntToHex converts a number into a string containing the number's hexadecimal (base 16) representation. Value is the number to convert. Digits indicates the minimum number of hexadecimal digits to return.

C++ Examples:

```
/*
The following example uses an edit control, a button, and a
label on a form. When the button is clicked, the hexadecimal
value of each character in the edit control is displayed in
the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = "";
```

```
    for (int i = 1; i <= Edit1->Text.Length(); i++)
    {
      try
      {
        Label1->Caption =
          Label1->Caption +
          IntToHex(Byte(Edit1->Text[i]),2) + ' ';
      }
      catch (...)
      {
        Beep();
      }
    }
  }
```

Delphi Examples:

```
{
The following example uses an edit control, a button, and a
label on a form. When the button is clicked, the hexadecimal
value of each character in the edit control is displayed in
the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
begin
  Label1.Caption := '';
  for i := 1 to Length(Edit1.Text) do
  begin
    try
      Label1.Caption :=
        Label1.Caption +
        SysUtils.IntToHex(Byte(Edit1.Text[i]),2) + ' ';
    except
      Beep;
    end;
  end;
end;
```

See Also

IntToStr (see page 22921)

159.3.138 SysUtils.IntToStr Function

Converts an integer to a string.

Pascal

```
function IntToStr(Value: Integer): string; overload;
function IntToStr(Value: Int64): string; overload;
```

C++

```
AnsiString IntToStr(int Value);
AnsiString IntToStr(Int64 Value);
```

File

SysUtils

Description

IntToStr converts an integer into a string containing the decimal representation of that number.

C++ Examples:

```
/*
The following example uses two edit controls, a button, and
a label on a form. When the button is clicked, the integers
in the edit controls are multiplied together and the result
is displayed in the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  try
  {
    Label1->Caption = IntToStr(StrToInt(Edit1->Text) * StrToInt(Edit2->Text));
  }
  catch(...)
  {
    ShowMessage("You must specify integer values. Please try again.");
  }
}
```

Delphi Examples:

```
{
The following example uses two edit controls, a button, and
a label on a form. When the button is clicked, the integers
in the edit controls are multiplied together and the result
is displayed in the label.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  try
    Label1.Caption :=
      SysUtils.IntToStr(StrToInt(Edit1.Text) * StrToInt(Edit2.Text));
  except
    ShowMessage('You must specify integer values. Please try again.');
  end;
end;
```

See Also

IntToHex (see page 22920)

StrToInt (see page 22970)

StrToIntDef (see page 22971)

FloatToStr (see page 22890)

DateToStr (see page 22825)

TimeToStr (see page 22977)

159.3.139 SysUtils.IsAssembly Function

Determines whether a dll is an Assembly.

Pascal

```
function IsAssembly(const FileName: string): Boolean;
```

C++

```
Boolean IsAssembly(const AnsiString FileName);
```

File

SysUtils

Description

Call IsAssembly to determine whether the passed FileName is that of a .Net Assembly or a traditional WIN32 DLL. The result is true if FileName is an Assembly.

159.3.140 SysUtils.IsDelimiter Function

Indicates whether a specified character in a string matches one of a set of delimiters.

Pascal

```
function IsDelimiter(const Delimiters: string; const S: string; Index: Integer): Boolean;
```

C++

```
Boolean IsDelimiter(const AnsiString Delimiters, const AnsiString S, int Index);
```

File

SysUtils

Description

Call IsDelimiter to determine whether the character at byte offset Index in the string S is one of the delimiters in the string Delimiters. Index is the index of the byte in question, where 1 is the first byte of the string, 2 is the second byte, and so on.

When working with a multi-byte character system (MBCS), IsDelimiter checks to make sure the indicated byte is not part of a double byte character. The delimiters in the Delimiters parameter must all be single byte characters.

See Also

AnsiStrRScan (see page 22799)

AnsiStrScan (see page 22799)

IsPathDelimiter (see page 22925)

LastDelimiter (see page 22926)

159.3.141 SysUtils.IsEqualGUID Function

Indicates whether two TGUID values are the same.

Pascal

```
function IsEqualGUID(const guid1: TGUID; const guid2: TGUID): Boolean; stdcall;
```

C++

```
__stdcall Boolean IsEqualGUID(const TGUID guid1, const TGUID guid2);
```

File

SysUtils

Description

IsEqualGUID returns true if the TGUID values are equal.

159.3.142 SysUtils.IsLeapYear Function

Indicates whether a specified year is a leap year.

Pascal

```
function IsLeapYear(Year: Word): Boolean;
```

C++

```
Boolean IsLeapYear(Word Year);
```

File

SysUtils

Description

Call IsLeapYear to determine whether the year specified by the Year parameter is a leap year. Year specifies the calendar year.

Use YearOf to obtain the value of Year for IsLeapYear from a TDateTime value.

C++ Examples:

```
/*
This example requires a button and a text edit. When the
user presses the button a message indicates if the year
specified in the text edit is a leap year.
*/
bool __fastcall IsThisLeapYear(int year)
{
  return IsLeapYear(year);
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int year = StrToInt(Edit1->Text);
  if (IsThisLeapYear(year))
    ShowMessage(Edit1->Text + " is a leap year.");
  else
    ShowMessage(Edit1->Text + " is not a leap year.");
}
```

Delphi Examples:

```
{
This example requires a button and a text edit. When the
user presses the button a message indicates if the year
specified in the text edit is a leap year.
}
function IsThisLeapYear(year : Word): Boolean;
begin
  Result := IsLeapYear(year);
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  year : integer;
begin
  year := StrToInt(Edit1.Text);
```

```
  if (IsThisLeapYear(year)) then
    ShowMessage(Edit1.Text + ' is a leap year.')
  else
    ShowMessage(Edit1.Text + ' is not a leap year.');
end;
```

See Also

Date (see page 22820)

YearOf (see page 6714)

IsInLeapYear (see page 6650)

159.3.143 SysUtils.IsPathDelimiter Function

Indicates whether the byte at a specified position in a string is the path delimiter.

Pascal

```
function IsPathDelimiter(const S: string; Index: Integer): Boolean;
```

C++

```
Boolean IsPathDelimiter(const AnsiString S, int Index);
```

File

SysUtils

Description

Call IsPathDelimiter to determine whether a position in the string S contains the path delimiter character ('/' on Linux, '\' on Windows). Positions are numbered from 1.

When working with a multi-byte character system (MBCS), IsPathDelimiter distinguishes between a true delimiter character, and the byte of the same value that can appear as the second byte of a double byte character.

To test for more characters than just the delimiter, use IsDelimiter (see page 22923).

See Also

IsDelimiter (see page 22923)

LastDelimiter (see page 22926)

159.3.144 SysUtils.IsValidIdent Function

Determines whether a string conforms to Delphi identifier rules.

Pascal

```
function IsValidIdent(const Ident: string; AllowDots: Boolean = False): Boolean;
```

C++

```
Boolean IsValidIdent(const AnsiString Ident, Boolean AllowDots = False);
```

File

SysUtils

Description

Call IsValidIdent to determine whether the passed Ident is valid for use as an identifier, such as the name of a component.

An identifier is a sequence of letters, digits or underscores. Only the first 63 characters are used uniquely. Letters are limited to A-Z and a-z, although the case is ignored. The first character cannot be a digit.

159.3.145 SysUtils.Languages Function

Lists the locales for which support is available.

Pascal

```
function Languages: TLanguages;
```

C++

```
TLanguages Languages();
```

File

SysUtils

Description

Use the value returned by Languages to obtain information about the locales that the system supports. This function is only available on Windows.

See Also

TSysLocale (see page 23008)

159.3.146 SysUtils.LastDelimiter Function

Returns the byte index of the last character that matches any character in a specified set of delimiters.

Pascal

```
function LastDelimiter(const Delimiters: string; const S: string): Integer;
```

C++

```
int LastDelimiter(const AnsiString Delimiters, const AnsiString S);
```

File

SysUtils

Description

Call LastDelimiter to locate the last delimiter in a specified string.

Delimiters is a string where each character is a valid delimiter.

S is the string to search for delimiters.

For example, the line

MyIndex := LastDelimiter('\.:','c:\filename.ext'); // Delphi

MyIndex = LastDelimiter("\\.:","c:\\filename.ext"); // C++

sets MyIndex to 12.

When working with multi-byte character sets (MBCS), S may contain double byte characters, but the delimiters listed in the Delimiters parameter must all be single byte non-null characters.

See Also

IsDelimiter (see page 22923)

AnsiStrRScan (see page 22799)

159.3.147 SysUtils.LoadPackage Function

Loads a specified package.

Pascal

```
function LoadPackage(const Name: string): HMODULE; overload;
function LoadPackage(const Name: string; AValidatePackage: TValidatePackageProc): HMODULE;
overload;
```

C++

```
HMODULE LoadPackage(const AnsiString Name);
HMODULE LoadPackage(const AnsiString Name, TValidatePackageProc AValidatePackage);
```

File

SysUtils

Description

LoadPackage loads the package specified by the Name parameter, checks for duplicate units, and calls the initialization blocks of all units contained in the package.

See Also

UnloadPackage (see page 22987)

159.3.148 SysUtils.LoadStr Function

Loads a string resource from the application's executable file.

Pascal

```
function LoadStr(Ident: Integer): string;
```

C++

```
AnsiString LoadStr(int Ident);
```

File

SysUtils

Description

LoadStr loads the string resource given by Ident from the application's executable file or shared library. If the string resource does not exist, LoadStr returns an empty string.

Moving display strings into string resources makes an application easier to localize without rewriting the program.

Note: :LoadStr is provided primarily for backward compatibility. To isolate string resources in Delphi, use the resourcestring keyword in new applications.

159.3.149 SysUtils.LowerCase Function

Converts an ASCII string to lowercase.

Pascal

```
function LowerCase(const S: string): string; overload;
function LowerCase(const S: string; LocaleOptions: TLocaleOptions): string; overload;
```

C++

```
AnsiString LowerCase(const AnsiString S);
AnsiString LowerCase(const AnsiString S, TLocaleOptions LocaleOptions);
```

File

SysUtils

Description

LowerCase returns a string with the same text as the string passed in S, but with all letters converted to lowercase. The conversion affects only 7-bit ASCII characters between 'A' and 'Z'. To convert 8-bit international characters, use AnsiLowerCase (see page 22792).

C++ Examples:

```
/*
This example uses an edit control, a label, and a button on a
form. When the user clicks the button, the label shows the text of
the edit control in lower case.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = LowerCase(Edit1->Text);
}
```

Delphi Examples:

```
{
This example uses an edit control, a label, and a button on a
form. When the user clicks the button, the label shows the text of
the edit control in lower case.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := SysUtils.LowerCase(Edit1.Text);
end;
{
The following example beeps once for each "beep" passed in
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//    'System ParamCount = ' + IntToStr(System.ParamCount),
```

```
//    mtConfirmation, [mbOK], 0);
  for i := 1 to System.ParamCount do
  begin
    if LowerCase(ParamStr(i)) = 'beep' then
      Beep
    else if LowerCase(ParamStr(i)) = 'exit' then
      Application.Terminate;
    Sleep(250);
  end;
end;
```

See Also

AnsiLowerCase (see page 22792)

UpperCase (see page 22988)

159.3.150 SysUtils.MSecsToTimeStamp Function

Converts a specified number of milliseconds into a TTimeStamp (see page 23009) value.

Pascal

```
function MSecsToTimeStamp(MSecs: Comp): TTimeStamp;
```

C++

```
TTimeStamp MSecsToTimeStamp(Comp MSecs);
```

File

SysUtils

Description

Call MSecsToTimeStamp to convert MSecs, a number of milliseconds, into the number of days represented, plus the number of milliseconds remaining. The days and milliseconds are expressed as a TTimeStamp (see page 23009) value, where the Date (see page 22820) field is the number of days and the Time (see page 22976) field is the remainder in milliseconds.

See Also

TimeStampToMSecs (see page 22977)

TTimeStamp (see page 23009)

159.3.151 SysUtils.NewStr Function

Allocates a string on the heap.

Pascal

```
function NewStr(const S: string): PString; deprecated;
```

C++

```
PString NewStr(const AnsiString S);
```

File

SysUtils

Description

NewStr allocates a new string on the heap. Do not use NewStr in new applications. This function is provided for backward compatibility only.

159.3.152 SysUtils.NextCharIndex Function

Returns the byte index of the next character.

Pascal

```
function NextCharIndex(const S: String; Index: Integer): Integer;
```

C++

```
int NextCharIndex(const AnsiString S, int Index);
```

File

SysUtils

Description

NextCharIndex returns the byte index of the first byte of the character following the character starting at S[Index].

159.3.153 SysUtils.Now Function

Returns the current date and time.

Pascal

```
function Now: TDateTime;
```

C++

```
TDateTime Now();
```

File

SysUtils

Description

Returns the current date and time, corresponding to the sum of the value returned by the global Date (see page 22820) and Time (see page 22976) functions.

Note: Although TDateTime values can represent milliseconds, Now is accurate only to the nearest second.

C++ Examples:

```
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date and time is
converted to a string and displayed as the caption of the
label:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = DateTimeToStr(Now());
```

```
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, the current date and time is
converted to a string and displayed as the caption of the
label:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := SysUtils.DateTimeToStr(Now);
end;
```

See Also

Date (see page 22820)

DateTimeToStr (see page 22822)

Time (see page 22976)

159.3.154 SysUtils.OutOfMemoryError Function

Raises an EOutOfMemory (see page 22742) exception.

Pascal

```
procedure OutOfMemoryError;
```

C++

```
OutOfMemoryError();
```

File

SysUtils

Description

Call OutOfMemoryError to raise the EOutOfMemory (see page 22742) exception. OutOfMemoryError runs without allocating additional memory.

See Also

EOutOfMemory (see page 22742)

159.3.155 SysUtils.QuotedStr Function

Returns the quoted version of a string.

Pascal

```
function QuotedStr(const S: string): string;
```

C++

```
AnsiString QuotedStr(const AnsiString S);
```

File

SysUtils

Description

Use QuotedStr to convert the string S to a quoted string. A single quote character (') is inserted at the beginning and end of S, and each single quote character in the string is repeated.

Note: When working with multi-byte character systems (MBCS), use AnsiQuotedStr (see page 22793) instead.

159.3.156 SysUtils.RaiseLastOSError Function

Raises an exception for the last occurring OS or system library error.

Pascal

```
procedure RaiseLastOSError; overload;
procedure RaiseLastOSError(LastError: Integer); overload;
```

C++

```
RaiseLastOSError();
RaiseLastOSError(int LastError);
```

File

SysUtils

Description

Call RaiseLastOSError to raise an EOSError (see page 22740) exception for the last OS API call that failed. RaiseLastOSError retrieves the code for the last occurring API call error, if any. If this belongs to a previously occurring error, RaiseLastOSError raises an EOSError (see page 22740) exception with the error code and message associated with that error.

See Also

EOSError (see page 22740)

159.3.157 SysUtils.RaiseLastWin32Error Function

Raises an exception for the last occurring Win32 error.

Pascal

```
procedure RaiseLastWin32Error; deprecated;
```

C++

```
RaiseLastWin32Error();
```

File

SysUtils

Description

Call RaiseLastWin32Error to raise an EWin32Error (see page 22749) exception for the last Windows API call that failed. RaiseLastWin32Error retrieves the code for the last occurring Win32 error, if any. If Windows returns a previously occurring error code, RaiseLastWin32Error raises an EWin32Error (see page 22749) exception with the error code and message associated with that error.

Note: RaiseLastWin32Error is a deprecated Windows-only procedure. New applications should use RaiseLastOSError (see page 22932) instead.

See Also

EWin32Error (see page 22749)

Win32Check (see page 22994)

RaiseLastOSError (see page 22932)

159.3.158 SysUtils.RCS Function

Check for Delphi Client/Server Licence.

Pascal

```
procedure RCS;
```

C++

```
RCS();
```

File

SysUtils

Description

RCS is used by VCL components to check whether there is a Client/Server licence, raising an exception of not.

See Also

GDAL (see page 22909)

RPR (see page 22938)

159.3.159 SysUtils.RemoveDir Function

Deletes an existing empty directory.

Pascal

```
function RemoveDir(const Dir: string): Boolean;
```

C++

```
Boolean RemoveDir(const AnsiString Dir);
```

File

SysUtils

Description

Call RemoveDir to remove the directory specified by the Dir parameter. The return value is true if a new directory was successfully deleted, false if an error occurred. The directory must be empty before it can be successfully deleted.

See Also

ExtractFileDir (see page 22843)

GetCurrentDir (see page 22910)

SetCurrentDir (see page 22940)

159.3.160 SysUtils.RenameFile Function

Changes a file name.

Pascal

```
function RenameFile(const OldName: string; const NewName: string): Boolean;
```

C++

```
Boolean RenameFile(const AnsiString OldName, const AnsiString NewName);
```

File

SysUtils

Description

RenameFile attempts to change the name of the file specified by OldFile to NewFile. If the operation succeeds, RenameFile returns true. If RenameFile cannot rename the file (for example, if the application does not have permission to modify the file), it returns false.

Note: If NewFile represents the name of an existing file, RenameFile behaves differently under Windows and Linux. Under Windows, the rename operation fails. Under Linux, RenameFile silently removes the other file.

C++ Examples:

```
/*
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
*/

#include <dir.h>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
    if (FileExists(SaveDialog1->FileName))
    {
      fnsplit(SaveDialog1->FileName.c_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, (char*)&(StringGrid1->ColCount), sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, (char*)&(StringGrid1->RowCount), sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
    {
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
```

```
        FileWrite(iFileHandle, (char*)&iLength, sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].c_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
    FileClose(iFileHandle);
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "Column 1";
  StringGrid1->Cells[2][0] = "Column 2";
  StringGrid1->Cells[3][0] = "Column 3";
  StringGrid1->Cells[4][0] = "Column 4";
  StringGrid1->Cells[0][1] = "Row 1";
  StringGrid1->Cells[1][1] = "Object";
  StringGrid1->Cells[2][1] = "Pascal";
  StringGrid1->Cells[3][1] = "is";
  StringGrid1->Cells[4][1] = "excellent";
  StringGrid1->Cells[0][2] = "Row 2";
  StringGrid1->Cells[1][2] = "Delphi";
  StringGrid1->Cells[2][2] = "is";
  StringGrid1->Cells[4][2] = "RAD";
};
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      Pchar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
```

```
          try
            { Write out the length of each string, followed by the string itself. }
            StringLen := Length(StringGrid1.Cells[X,Y]);
            FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
            GetMem(Buffer, StringLen); { allocate the buffer }
            cellString := StringGrid1.Cells[X,Y];
            for I := 1 to StringLen do
               Buffer[I - 1] := cellString[I];
            FileWrite(FileHandle, Buffer^, StringLen);
          finally
            FreeMem(Buffer, StringLen);
          end;
        end;
      end;
      FileClose(FileHandle);
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with StringGrid1 do
    begin
      Cells[1,0] := 'Column 1';
      Cells[2,0] := 'Column 2';
      Cells[3,0] := 'Column 3';
      Cells[4,0] := 'Column 4';
      Cells[0,1] := 'Row 1';
      Cells[1,1] := 'Object';
      Cells[2,1] := 'Pascal';
      Cells[3,1] := 'is';
      Cells[4,1] := 'excellent';
      Cells[0,2] := 'Row 2';
      Cells[1,2] := 'Delphi';
      Cells[2,2] := 'is';
      Cells[4,2] := 'RAD';
    end;
  end;
```

See Also

DeleteFile (see page 22831)

Rename (see page 22510)

159.3.161 SysUtils.ReplaceDate Function

Replaces the date portion of a TDateTime value with a specified date.

Pascal

```
procedure ReplaceDate(var DateTime: TDateTime; const NewDate: TDateTime);
```

C++

```
ReplaceDate(TDateTime DateTime, const TDateTime NewDate);
```

File

SysUtils

Description

ReplaceDate changes the value of the DateTime parameter so that it has the same date as the NewDate parameter. The time portion of DateTime remains the same. ReplaceDate makes any necessary adjustments for negative dates (before 1900).

See Also

DecodeDate (see page 22828)

EncodeDate (see page 22836)

ReplaceTime (see page 22937)

IncMonth (see page 22919)

RecodeDateDay

RecodeDateHour

RecodeDateMillisecond

RecodeDateMinute

RecodeDateMonth

RecodeDateSecond

RecodeDateYear

RecodeDate (see page 6673)

159.3.162 SysUtils.ReplaceTime Function

Replaces the time portion of a TDateTime value with a specified time.

Pascal

```
procedure ReplaceTime(var DateTime: TDateTime; const NewTime: TDateTime);
```

C++

```
ReplaceTime(TDateTime DateTime, const TDateTime NewTime);
```

File

SysUtils

Description

ReplaceTime changes the value of the DateTime parameter so that it has the same time as the NewTime parameter. The date portion of DateTime remains the same. ReplaceTime makes any necessary adjustments for negative dates. (before 1900)

See Also

DecodeTime (see page 22830)

EncodeTime (see page 22837)

ReplaceDate (see page 22936)

RecodeDateDay

RecodeDateHour

RecodeDateMillisecond

RecodeDateMinute

RecodeDateMonth

RecodeDateSecond

RecodeDateYear

StartOfADay (see page 6689)

EndOfADay (see page 6633)

RecodeTime (see page 6679)

159.3.163 SysUtils.RPR Function

Check for Delphi Professional Licence.

Pascal

```
procedure RPR;
```

C++

```
RPR();
```

File

SysUtils

Description

RPR is used by VCL components to check whether there is a Professional licence, raising an exception of not.

See Also

GDAL (see page 22909)

RCS (see page 22933)

159.3.164 SysUtils.SafeLoadLibrary Function

Safe and simple library loading.

Pascal

```
function SafeLoadLibrary(const FileName: string; ErrorMode: UINT = SEM_NOOPENFILEERRORBOX):
HMODULE;
```

C++

```
HMODULE SafeLoadLibrary(const AnsiString FileName, UINT ErrorMode = SEM_NOOPENFILEERRORBOX);
```

File

SysUtils

Description

SafeLoadLibrary loads a Windows DLL or Linux shared object file, as specified by Filename. SafeLoadLibrary preserves the current FPU control word, preventing library initialization code from permanently overwriting precision and exception masks.

Note: On Windows, SafeLoadLibrary temporarily sets the system error mode to ErrorMode. The default, SEM_NOOPENFILEERRORBOX, suppresses error dialogs. The previous error mode is restored before SafeLoadLibrary exits. For a list of error modes, refer to SetErrorMode in the Microsoft documentation.

Note: On Linux, the Dummy argument is ignored.

See Also

LoadPackage (see page 22927)

159.3.165 SysUtils.SameFileName Function

Compares file names based on the current locale.

Pascal

```
function SameFileName(const S1: string; const S2: string): Boolean;
```

C++

```
Boolean SameFileName(const AnsiString S1, const AnsiString S2);
```

File

SysUtils

Description

SameFileName compares two file names in the same way as AnsiCompareFileName (see page 22788). SameFileName returns true if the file names match, false otherwise. Like AnsiCompareFileName (see page 22788), SameFileName supports locales and MBCS strings; see the AnsiCompareFileName (see page 22788) topic for more details.

See Also

AnsiCompareText (see page 22790)

159.3.166 SysUtils.SameStr Function

Compares two strings by ordinal value with case sensitivity.

Pascal

```
function SameStr(const S1: string; const S2: string): Boolean; overload;
function SameStr(const S1: string; const S2: string; LocaleOptions: TLocaleOptions): Boolean;
overload;
```

C++

```
Boolean SameStr(const AnsiString S1, const AnsiString S2);
Boolean SameStr(const AnsiString S1, const AnsiString S2, TLocaleOptions LocaleOptions);
```

File

SysUtils

Description

SameStr compares S1 and S2 and returns true if they are identical. SameStr is case sensitive and is not affected by the current locale.

See Also

SameText (see page 22940)

CompareStr (see page 22813)

CompareText (see page 22814)

AnsicompareStr

AnsicompareText

CompareMem (see page 22812)

159.3.167 SysUtils.SameText Function

Compares two strings by ordinal value without case sensitivity.

Pascal

```
function SameText(const S1: string; const S2: string): Boolean; overload;
function SameText(const S1: string; const S2: string; LocaleOptions: TLocaleOptions): Boolean;
overload;
```

C++

```
Boolean SameText(const AnsiString S1, const AnsiString S2);
Boolean SameText(const AnsiString S1, const AnsiString S2, TLocaleOptions LocaleOptions);
```

File

SysUtils

Description

SameText compares S1 and S2 and returns true if they are equal. SameText is not case sensitive and is not affected by the current locale.

See Also

CompareStr (see page 22813)

AnsiCompareText (see page 22790)

CompareMem (see page 22812)

CompareText (see page 22814)

159.3.168 SysUtils.SetCurrentDir Function

Sets the current directory.

Pascal

```
function SetCurrentDir(const Dir: string): Boolean;
```

C++

```
Boolean SetCurrentDir(const AnsiString Dir);
```

File

SysUtils

Description

The SetCurrentDir function sets the current directory. The return value is true if the current directory was successfully changed, or false if an error occurred.

See Also

ChDir (see page 22429)

CreateDir (see page 22817)

ExtractFileDir (see page 22843)

GetCurrentDir (see page 22910)

RemoveDir (see page 22933)

159.3.169 SysUtils.ShowException Function

Displays an exception message with its physical address

Pascal

```
procedure ShowException(ExceptObject: TObject; ExceptAddr: Pointer);
```

C++

```
ShowException(TObject * ExceptObject, void * ExceptAddr);
```

File

SysUtils

Description

ShowException displays the message associated with an exception, together with the exception's physical address. The exception's address can be used with Search | Find Error menu command to find the statement that raised the exception.

See Also

ExceptionErrorMessage (see page 22839)

ExceptObject (see page 22452)

ExceptAddr (see page 22452)

159.3.170 SysUtils.Sleep Function

Delay program execution for a specified number of milliseconds.

Pascal

```
procedure Sleep(milliseconds: Cardinal); stdcall;
```

C++

```
__stdcall Sleep(unsigned milliseconds);
```

File

SysUtils

Description

Sleep pauses program executions as specified by the milliseconds parameter. Under Windows, Sleep is just a link to the Sleep function in the system API.

159.3.171 SysUtils.StrAlloc Function

Allocates a buffer for a null-terminated string and returns a pointer to its first character.

Pascal

```
function StrAlloc(Size: Cardinal): PChar;
```

C++

```
const char * StrAlloc(unsigned Size);
```

File

SysUtils

Description

StrAlloc allocates a buffer for a null-terminated string with a maximum length of Size - 1 (1 byte must be reserved for the termination character). The result points to the location where the first character of the string is to be stored. A 32-bit number giving the total amount of memory allocated is stored in the four bytes preceding the first character; it is equal to Size + 4. If space for a string is allocated with StrAlloc, it should be deallocated via StrDispose (see page 22947).

Because AnsiStrings (long strings) are implicitly null terminated and dynamically allocated, the use of StrAlloc is deprecated.

See Also

StrDispose (see page 22947)

StrNew (see page 22957)

StrBufSize (see page 22942)

159.3.172 SysUtils.StrBufSize Function

Returns the maximum number of characters that can be stored in a buffer allocated by StrAlloc (see page 22942).

Pascal

```
function StrBufSize(const Str: PChar): Cardinal;
```

C++

```
unsigned StrBufSize(const const char * Str);
```

File

SysUtils

Description

StrBufSize returns the number of bytes in Str, including the byte for the termination character.

Note: NoteIf Str does not point to a string buffer allocated by StrAlloc (see page 22942) or StrNew (see page 22957), no error message is returned, and the result is unpredictable.

See Also

StrAlloc (see page 22942)

StrNew (see page 22957)

159.3.173 SysUtils.StrByteType Function

Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.

Pascal

```
function StrByteType(Str: PChar; Index: Cardinal): TMbcsByteType;
```

C++

```
TMbcsByteType StrByteType(const char * Str, unsigned Index);
```

File

SysUtils

Description

Call StrByteType to determine the type of the byte specified by the Index parameter, where 0 specifies the first byte in the string Str, 1 specifies the second byte, and so on.

If the system is not using a multi-byte character system (MBCS), StrByteType always returns mbSingleByte. Otherwise, StrByteType returns mbSingleByte if the indicated byte represents a complete character in Str, mbLeadByte if it represents the first byte of a double byte character, and mbTrailByte if it represents the second byte of a double byte character.

Note: No checking is done to ensure that Index is less than the length of Str. It is the caller's responsibility to ensure that Index is not out of bounds.

See Also

ByteType (see page 22807)

LeadBytes (see page 23026)

159.3.174 SysUtils.StrCat Function

Appends a copy of Source to the end of Dest and returns the concatenated string.

Pascal

```
function StrCat(Dest: PChar; const Source: PChar): PChar;
```

C++

```
const char * StrCat(const char * Dest, const const char * Source);
```

File

SysUtils

Description

Use StrCat to concatenate Source to the end of Dest. StrCat does not perform any length checking. The destination buffer must

have room for at least StrLen (see page 22954)(Dest)+StrLen (see page 22954)(Source)+1 characters.

To check length, use the StrLCat (see page 22952) function.

C++ Examples:

```
/*
The following example uses an edit control, a label, and a button
on a form. When the button is clicked, the label's caption and the
edit control's text are combined in a buffer. Then the buffer is
displayed in the label's caption.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char* szBuffer = new char[Label1->Caption.Length() + Edit1->Text.Length() + 1];
  // The following 2 lines are used in VCL applications
  StrCopy(szBuffer, Label1->Caption.c_str());
  StrCat(szBuffer, Edit1->Text.c_str());
  Label1->Caption = szBuffer;
  Edit1->Clear();
  delete [] szBuffer;
}
```

Delphi Examples:

```
{
The following example uses an edit control, a label, and a button
on a form. When the button is clicked, the label's caption and the
edit control's text are combined in a buffer. Then the buffer is
displayed in the label's caption.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
begin
  GetMem(Buffer,Length(Label1.Caption) + Length(Edit1.Text) + 1);
  StrCopy(Buffer, PChar(Label1.Caption));
  SysUtils.StrCat(Buffer, PChar(Edit1.Text));
  Label1.Caption := Buffer;
  Edit1.Clear;
  FreeMem(Buffer);
end;
```

See Also

StrLCat (see page 22952)

StrLen (see page 22954)

159.3.175 SysUtils.StrCharLength Function

Returns the length, in bytes, of a character.

Pascal

```
function StrCharLength(const Str: PChar): Integer;
```

C++

```
int StrCharLength(const const char * Str);
```

File

SysUtils

Description

StrCharLength returns the size, in bytes, of the first character in Str. This value can be 1 or 2 under Windows, 1 to 6 under Linux.

See Also

ByteType (see page 22807)

LeadBytes (see page 23026)

159.3.176 SysUtils.StrComp Function

Compares two strings with case sensitivity.

Pascal

```
function StrComp(const Str1: PChar; const Str2: PChar): Integer;
```

C++

```
int StrComp(const const char * Str1, const const char * Str2);
```

File

SysUtils

Description

Call StrComp to compare two null-terminated strings, with case sensitivity. The return value is indicated in the following table:

Return value	Condition
<0	Str1 sorts before Str2
=0	Str1is the same as Str2
>0	Str1 sorts after Str2

Note: When working with International characters, use AnsiStrComp (see page 22795) instead.

C++ Examples:

```
/*
The following example uses two edit controls and a button on
a form. When the button is clicked, the text in the edit
controls is compared.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char* szResult = new char[Edit1->Text.Length() + Edit2->Text.Length() + 20];
  int iResult = StrComp(Edit1->Text.c_str(), Edit2->Text.c_str());
  StrCopy(szResult, Edit1->Text.c_str());
  if (iResult < 0)
    StrCat(szResult, " is less than ");
  else if (iResult > 0)
    StrCat(szResult, " is greater than ");
  else
    StrCat(szResult, " is equal to ");

  StrCat(szResult, Edit2->Text.c_str());
  ShowMessage(AnsiString(szResult));
  delete [] szResult;
```

```
}
```

Delphi Examples:

```
{
The following example uses two edit controls and a button on
a form. When the button is clicked, the text in the edit
controls is compared.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Msg: string;
  CompResult: Integer;
begin
  Msg := Edit1.Text;
  CompResult := SysUtils.StrComp(PChar(Edit1.Text), PChar(Edit2.Text));
  if CompResult < 0 then
    Msg := Msg + ' is less than '
  else if CompResult > 0 then
    Msg := Msg + ' is greater than '
  else
    Msg := Msg + ' is equal to ';
  Msg := Msg + Edit2.Text;
  ShowMessage(Msg);
end;
```

See Also

AnsiStrComp (see page 22795)

StrIComp (see page 22951)

StrLComp (see page 22953)

StrLIComp (see page 22955)

159.3.177 SysUtils.StrCopy Function

Copies a null-terminated string.

Pascal

```
function StrCopy(Dest: PChar; const Source: PChar): PChar;
```

C++

```
const char * StrCopy(const char * Dest, const const char * Source);
```

File

SysUtils

Description

Use StrCopy to copy Source to Dest. StrCopy returns Dest.

StrCpy does not perform any length checking. The destination buffer must have room for at least StrLen (see page 22954)(Source)+1 characters.

For length checking, use the StrLCopy (see page 22953) function.

C++ Examples:

```
/*
The following example uses an edit control, a label, and a button
```

```
on a form. When the button is clicked, the label's caption and the
edit control's text are combined in a buffer. Then the buffer is
displayed in the label's caption.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char* szBuffer = new char[Label1->Caption.Length() + Edit1->Text.Length() + 1];
  // The following 2 lines are used in VCL applications
  StrCopy(szBuffer, Label1->Caption.c_str());
  StrCat(szBuffer, Edit1->Text.c_str());
  Label1->Caption = szBuffer;
  Edit1->Clear();
  delete [] szBuffer;
}
```

Delphi Examples:

```
{
The following example uses an edit control, a label, and a button
on a form. When the button is clicked, the label's caption and the
edit control's text are combined in a buffer. Then the buffer is
displayed in the label's caption.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
begin
  GetMem(Buffer,Length(Label1.Caption) + Length(Edit1.Text) + 1);
  StrCopy(Buffer, PChar(Label1.Caption));
  SysUtils.StrCat(Buffer, PChar(Edit1.Text));
  Label1.Caption := Buffer;
  Edit1.Clear;
  FreeMem(Buffer);
end;
```

See Also

StrECopy (see page 22948)

StrLCopy (see page 22953)

159.3.178 SysUtils.StrDispose Function

Disposes of a string.

Pascal

```
procedure StrDispose(Str: PChar);
```

C++

```
StrDispose(const char * Str);
```

File

SysUtils

Description

StrDispose is provided for backward compatibility. StrDispose disposes of a string on a heap that was previously allocated with StrAlloc (see page 22942) or StrNew (see page 22957).

If Str is nil (Delphi) or NULL (C++), StrDispose does nothing.

C++ Examples:

```
/*
This example uses an edit control and a button on a form.
When the button is clicked, memory is allocated for a copy
of the text in the edit control, the text is displayed in a
message box, and then the memory is deallocated.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // Allocate memory.
  char* psz = StrNew(Edit1->Text.c_str());
  Application->MessageBox(psz, "StrNew, StrDispose example", MB_OK);
  // Deallocate memory.
  StrDispose(psz);
}
```

Delphi Examples:

```
{
This example uses an edit control and a button on a form.
When the button is clicked, memory is allocated for a copy
of the text in the edit control, the text is displayed in a
message box, and then the memory is deallocated.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Temp: PChar;
begin
  // Allocate memory.
  Temp := StrNew(PChar(Edit1.Text));
  Application.MessageBox(Temp, 'StrNew, StrDispose example', MB_OK);
  // Deallocate memory.
  SysUtils.StrDispose(Temp);
end;
```

See Also

StrAlloc (see page 22942)

StrNew (see page 22957)

159.3.179 SysUtils.StrECopy Function

Copies null-terminated string.

Pascal

```
function StrECopy(Dest: PChar; const Source: PChar): PChar;
```

C++

```
const char * StrECopy(const char * Dest, const const char * Source);
```

File

SysUtils

Description

StrECopy copies Source to Dest and returns StrEnd (see page 22949)(Dest).

StrECopy does not perform any length checking. The destination buffer must have room for at least StrLen (see page 22954)(Source)+1 characters.

Nested calls to StrECopy to concatenate a sequence of strings run more efficiently than multiple calls to StrCat (see page 22943).

C++ Examples:

```
/*
The following example uses an edit control, a label, and a
button on a form. When the button is clicked, the edit
control’s text is appended to a static buffer. Then the
buffer is displayed in the label’s caption.
*/
const static int MAX_LENGTH = 15;
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  static char szBuffer[MAX_LENGTH];
  static char* pszBuffer = szBuffer;

  if (StrLen(pszBuffer) + StrLen(Edit1->Text.c_str()) < MAX_LENGTH)
    pszBuffer = StrECopy(pszBuffer, Edit1->Text.c_str());
  else
    ShowMessage("Can't fit new string in allocated memory space.");
  Label1->Caption = szBuffer;
}
```

Delphi Examples:

```
{
This example uses a button on a form.  Since StrECopy returns
the destination char array, nested calls are possible and
actually more efficient than repeated calls.  StrECopy does
no bounds checking. When the button is clicked, the inline
text is appended to a static buffer. Then the buffer is
displayed in the form's canvas.
}

procedure TForm1.Button1Click(Sender: TObject);
const
  Turbo: PChar = 'Object';
  Pascal: PChar = 'Pascal';
 var
  S: array[0..15] of Char;
begin
  StrECopy(StrECopy(StrECopy(S, Turbo), ' '), Pascal);
  Canvas.TextOut(10, 10, string(S));
end;
```

See Also

StrCat (see page 22943)

StrCopy (see page 22946)

StrEnd (see page 22949)

StrLen (see page 22954)

159.3.180 SysUtils.StrEnd Function

Returns a pointer to the end of a null terminated string.

Pascal

```
function StrEnd(const Str: PChar): PChar;
```

C++

```
const char * StrEnd(const const char * Str);
```

File

SysUtils

Description

StrEnd returns a pointer to the null character at the end of Str.

See Also

AnsiStrLastChar (see page 22796)

StrLen (see page 22954)

159.3.181 SysUtils.StrFmt Function

Formats entries in an array.

Pascal

```
function StrFmt(Buffer: PChar; Format: PChar; const Args: array of const): PChar; overload;
function StrFmt(Buffer: PChar; Format: PChar; const Args: array of const; const
FormatSettings: TFormatSettings): PChar; overload;
```

C++

```
const char * StrFmt(const char * Buffer, const char * Format, const array of const Args);
const char * StrFmt(const char * Buffer, const char * Format, const array of const Args, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

StrFmt formats the series of arguments from an open array into a buffer.

Buffer is a buffer that receives the results.

Format (see page 22897) is a null-terminated format string that indicates how to generate the results. For more information about how the result is formatted, see Format (see page 22897) Strings.

Args is an array of arguments that replace the format specifiers in Format (see page 22897).

Note: Args_Size gives the index of the last argument (one less than the number of elements in Args).

StrFmt returns a pointer to the destination buffer.

The first form of StrFmt is not thread-safe, because it uses localization information contained in global variables. The second form of StrFmt, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrFmt, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

See Also

Format (see page 22897)

GetLocaleFormatSettings (see page 22913)

159.3.182 SysUtils.StrIComp Function

Compares two strings without case sensitivity.

Pascal

```
function StrIComp(const Str1: PChar; const Str2: PChar): Integer;
```

C++

```
int StrIComp(const const char * Str1, const const char * Str2);
```

File

SysUtils

Description

Call StrIComp to compare two strings without case sensitivity. StrIComp returns a value greater than 0 if Str1 > Str2, less than 0 if Str1 < Str2, and returns 0 if the strings are equal except for differences in case.

When working with international characters, use AnsiStrIComp (see page 22795) instead.

See Also

AnsiStrIComp (see page 22795)

StrComp (see page 22945)

StrLComp (see page 22953)

StrLIComp (see page 22955)

159.3.183 SysUtils.StringReplace Function

Returns a string with occurrences of one substring replaced by another substring.

Pascal

```
function StringReplace(const S: string; const OldPattern: string; const NewPattern: string;
Flags: TReplaceFlags): string;
```

C++

```
AnsiString StringReplace(const AnsiString S, const AnsiString OldPattern, const AnsiString
NewPattern, TReplaceFlags Flags);
```

File

SysUtils

Description

StringReplace replaces occurrences of the substring specified by OldPattern with the substring specified by NewPattern. StringReplace assumes that the source string may contain Multibyte characters.

S is the source string, whose substrings are changed.

OldPattern is the substring to locate and replace with NewPattern.

NewPattern is the substring to substitute for occurrences of OldPattern.

Flags is a set of flags that govern how StringReplace locates and replaces occurrences of OldPattern. If Flags does not include rfReplaceAll, StringReplace only replaces the first occurrence of OldPattern in S. Otherwise, StringReplace replaces all instances of OldPattern with NewPattern. If the Flags parameter includes rfIgnoreCase, the comparison operation is case insensitive.

159.3.184 SysUtils.StringToGUID Function

Converts a string to a GUID.

Pascal

```
function StringToGUID(const S: string): TGUID;
```

C++

```
TGUID StringToGUID(const AnsiString S);
```

File

SysUtils

Description

StringToGUID converts the string representation of a GUID, S, into an equivalent TGUID data structure.

If StringToGUID succeeds it returns the requested GUID. If StringToGUID fails an EConvertError (see page 22724) exception is raised.

See Also

GUIDToString (see page 22916)

159.3.185 SysUtils.StrLCat Function

Appends up to a specified maximum number of characters to string.

Pascal

```
function StrLCat(Dest: PChar; const Source: PChar; MaxLen: Cardinal): PChar;
```

C++

```
const char * StrLCat(const char * Dest, const const char * Source, unsigned MaxLen);
```

File

SysUtils

Description

StrLCat appends at most MaxLen - StrLen (see page 22954)(Dest) characters from Source to the end of Dest and returns Dest. That is, MaxLen indicates the maximum length that is allowed in the result string.

See Also

SizeOf (see page 22528)

StrCat (see page 22943)

StrLen (see page 22954)

159.3.186 SysUtils.StrLComp Function

Compares up to a specified maximum number of characters in two strings.

Pascal

```
function StrLComp(const Str1: PChar; const Str2: PChar; MaxLen: Cardinal): Integer;
```

C++

```
int StrLComp(const const char * Str1, const const char * Str2, unsigned MaxLen);
```

File

SysUtils

Description

StrLComp compares Str1 to Str2, up to a maximum length of MaxLen characters. The return value is indicated in the following table:

Return value	Condition
<0	Str1sorts before Str2
=0	Str1is the same as Str2
>0	Str1 sorts after Str2

Note: When working with international characters, use AnsiStrLComp (see page 22796) instead.

See Also

AnsiStrLComp (see page 22796)

StrComp (see page 22945)

StrIComp (see page 22951)

StrLIComp (see page 22955)

159.3.187 SysUtils.StrLCopy Function

Copies up to a specified maximum number of characters from Source to Dest.

Pascal

```
function StrLCopy(Dest: PChar; const Source: PChar; MaxLen: Cardinal): PChar;
```

C++

```
const char * StrLCopy(const char * Dest, const const char * Source, unsigned MaxLen);
```

File

SysUtils

Description

StrLCopy copies at most MaxLen characters from Source to Dest, then adds a null terminator to Dest and returns Dest. The SizeOf standard function (Delphi) or the sizeof operator (C++) can be used to determine the MaxLen parameter. Usually MaxLen

equals SizeOf(Dest)-1.

Note: StrLCopy copies a null terminator as well as MaxLen characters. Be sure that Dest is large enough to accommodate the null terminator at the end.

See Also

SizeOf (see page 22528)

StrCopy (see page 22946)

StrLen (see page 22954)

159.3.188 SysUtils.StrLen Function

Returns number of characters in a string excluding the null terminator.

Pascal

```
function StrLen(const Str: PChar): Cardinal;
```

C++

```
unsigned StrLen(const const char * Str);
```

File

SysUtils

Description

StrLen returns the number of characters in Str, not counting the null terminator.

See Also

StrEnd (see page 22949)

159.3.189 SysUtils.StrLFmt Function

Formats a series of arguments from a specified open array into a buffer.

Pascal

```
function StrLFmt(Buffer: PChar; MaxBufLen: Cardinal; Format: PChar; const Args: array of
const): PChar; overload;
function StrLFmt(Buffer: PChar; MaxBufLen: Cardinal; Format: PChar; const Args: array of
const; const FormatSettings: TFormatSettings): PChar; overload;
```

C++

```
const char * StrLFmt(const char * Buffer, unsigned MaxBufLen, const char * Format, const array
of const Args);
const char * StrLFmt(const char * Buffer, unsigned MaxBufLen, const char * Format, const array
of const Args, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

StrLFmt formats the series of arguments in and open array and writes the results to a buffer.

Buffer is the buffer that receives the results.

MaxLen is the maximum number of bytes that can be written to Buffer. This value does not include the terminating null character.

Format (see page 22897) is the format string that governs the operation.

Args is an array of arguments that replace the format specifiers in Format (see page 22897).

Note: Args_Size is the index of the last argument in Args (one less than the number of arguments).

StrLFmt returns a pointer to the destination buffer.

The first form of StrLFmt is not thread-safe, because it uses localization information contained in global variables. The second form of StrLFmt, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrLFmt, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

See Also

FormatBuf (see page 22900)

StrFmt (see page 22950)

FmtStr (see page 22894)

GetLocaleFormatSettings (see page 22913)

159.3.190 SysUtils.StrLIComp Function

Compares strings up to a specified maximum number of characters, without case sensitivity.

Pascal

```
function StrLIComp(const Str1: PChar; const Str2: PChar; MaxLen: Cardinal): Integer;
```

C++

```
int StrLIComp(const const char * Str1, const const char * Str2, unsigned MaxLen);
```

File

SysUtils

Description

StrLIComp compares Str1 to Str2, up to a maximum length of MaxLen characters, without case sensitivity. The return value is indicated in the following table:

Return value	Condition
<0	Str1 sorts before Str2
=0	Str1is the same as Str2
>0	Str1 sorts after Str2

Note: When working with international characters, use AnsiStrLIComp (see page 22797) instead.

See Also

AnsiStrLIComp (see page 22797)

StrComp (see page 22945)

StrIComp (see page 22951)

StrLComp (see page 22953)

159.3.191 SysUtils.StrLower Function

Converts a string to lowercase.

Pascal

```
function StrLower(Str: PChar): PChar;
```

C++

```
const char * StrLower(const char * Str);
```

File

SysUtils

Description

StrLower converts Str to lowercase and returns Str.

Note: If the source string contains international characters, use AnsiStrLower (see page 22798) instead. If the source string is an AnsiString (long string), use LowerCase (see page 22928) (or AnsiLowerCase (see page 22792)) instead.

See Also

AnsiLowerCase (see page 22792)

AnsiStrLower (see page 22798)

LowerCase (see page 22928)

StrUpper (see page 22973)

159.3.192 SysUtils.StrMove Function

Copies specified number of characters to string.

Pascal

```
function StrMove(Dest: PChar; const Source: PChar; Count: Cardinal): PChar;
```

C++

```
const char * StrMove(const char * Dest, const const char * Source, unsigned Count);
```

File

SysUtils

Description

StrMove copies exactly Count characters from Source to Dest and returns Dest. Source and Dest can overlap.

See Also

Move (see page 22490)

159.3.193 SysUtils.StrNew Function

Allocates space on and copies a string to the heap; returning a pointer to the string.

Pascal

```
function StrNew(const Str: PChar): PChar;
```

C++

```
const char * StrNew(const const char * Str);
```

File

SysUtils

Description

StrNew allocates a copy of Str on the heap.

If Str is nil (Delphi) or NULL (C++) or points to an empty string, StrNew returns a pointer to a new empty string.

Otherwise, StrNew makes a duplicate of Str, obtaining space with a call to StrAlloc (see page 22942), and returns a pointer to the duplicated string.

The allocated space is the length of Str + 5 bytes.

C++ Examples:

```
/*
This example uses an edit control and a button on a form.
When the button is clicked, memory is allocated for a copy
of the text in the edit control, the text is displayed in a
message box, and then the memory is deallocated.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // Allocate memory.
  char* psz = StrNew(Edit1->Text.c_str());
  Application->MessageBox(psz, "StrNew, StrDispose example", MB_OK);
  // Deallocate memory.
  StrDispose(psz);
}
```

Delphi Examples:

```
{
This example uses an edit control and a button on a form.
When the button is clicked, memory is allocated for a copy
of the text in the edit control, the text is displayed in a
message box, and then the memory is deallocated.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Temp: PChar;
begin
  // Allocate memory.
  Temp := StrNew(PChar(Edit1.Text));
  Application.MessageBox(Temp, 'StrNew, StrDispose example', MB_OK);
  // Deallocate memory.
  SysUtils.StrDispose(Temp);
end;
```

See Also

GetMem (see page 22469)

StrDispose (see page 22947)

StrAlloc (see page 22942)

159.3.194 SysUtils.StrNextChar Function

Returns pointer to following character.

Pascal

```
function StrNextChar(const Str: PChar): PChar;
```

C++

```
const char * StrNextChar(const const char * Str);
```

File

SysUtils

Description

Note: StrNextChar returns a pointer to the second character in Str, allowing for multi-byte characters.

159.3.195 SysUtils.StrPas Function

Converts null-terminated string to an AnsiString (long string).

Pascal

```
function StrPas(const Str: PChar): string;
```

C++

```
AnsiString StrPas(const const char * Str);
```

File

SysUtils

Description

This function is provided for backwards compatibility only. To convert a null terminated string to an AnsiString or native Delphi language string, use a typecast or an assignment.

See Also

StrPCopy (see page 22958)

159.3.196 SysUtils.StrPCopy Function

Copies an AnsiString (long string) to a null-terminated string.

Pascal

```
function StrPCopy(Dest: PChar; const Source: string): PChar;
```

C++

```
const char * StrPCopy(const char * Dest, const AnsiString Source);
```

File

SysUtils

Description

StrPCopy copies Source into a null-terminated string Dest. It returns a pointer to Dest.

StrPCopy does not perform any length checking.

The destination buffer must have room for at least Length(Source)+1 characters.

See Also

StrCopy (see page 22946)

159.3.197 SysUtils.StrPLCopy Function

Copies characters from an AnsiString (long string) into a null-terminated string.

Pascal

```
function StrPLCopy(Dest: PChar; const Source: string; MaxLen: Cardinal): PChar;
```

C++

```
const char * StrPLCopy(const char * Dest, const AnsiString Source, unsigned MaxLen);
```

File

SysUtils

Description

StrPLCopy copies up to a maximum of MaxLen characters from Source into the null-terminated string Dest. StrPLCopy returns a pointer to Dest.

Note: Dest should point to a location with room for MaxLen + 1 characters, including the null terminator.

159.3.198 SysUtils.StrPos Function

Returns a pointer to the first occurrence of STR2 in STR1.

Pascal

```
function StrPos(const Str1: PChar; const Str2: PChar): PChar;
```

C++

```
const char * StrPos(const const char * Str1, const const char * Str2);
```

File

SysUtils

Description

StrPos returns a pointer to the first occurrence of Str2 in Str1. If Str2 does not occur in Str1, StrPos returns nil (Delphi) or NULL (C++).

Note: If the source string contains international characters, use AnsiStrPos (see page 22798) instead.

See Also

AnsiStrPos (see page 22798)

159.3.199 SysUtils.StrRScan Function

Returns a pointer to the last occurrence of a specified character in a string.

Pascal

```
function StrRScan(const Str: PChar; Chr: Char): PChar;
```

C++

```
const char * StrRScan(const const char * Str, Char Chr);
```

File

SysUtils

Description

StrRScan returns a pointer to the last occurrence of Chr in Str.

If Chr does not occur in Str, StrRScan returns nil (Delphi) or NULL (C++). The null terminator is considered to be part of the string.

Note: If the source string contains international characters, use AnsiStrRScan (see page 22799) instead.

See Also

AnsiStrRScan (see page 22799)

StrScan (see page 22960)

159.3.200 SysUtils.StrScan Function

Returns a pointer to first occurrence of a specified character in a string.

Pascal

```
function StrScan(const Str: PChar; Chr: Char): PChar;
```

C++

```
const char * StrScan(const const char * Str, Char Chr);
```

File

SysUtils

Description

StrScan returns a pointer to the first occurrence of Chr in Str. If Chr does not occur in Str, StrScan returns nil (Delphi) or NULL

(C++). The null terminator is considered to be part of the string.

Note: If the source string contains international characters, use AnsiStrScan (see page 22799) instead.

See Also

AnsiStrScan (see page 22799)

StrRScan (see page 22960)

159.3.201 SysUtils.StrToBool Function

Converts a string to a Boolean value.

Pascal

```
function StrToBool(const S: string): Boolean;
```

C++

```
Boolean StrToBool(const AnsiString S);
```

File

SysUtils

Description

StrToBool converts the string specified by S to a Boolean. If S can be converted to a number, StrToBool returns false if that number is 0, true otherwise. If S can't be converted to a number, StrToBool returns true if S is any of the strings listed in TrueBoolStrs (see page 23034) (or differs from one only by case) and false if it is any of the strings listed in FalseBoolStrs (see page 23025) (or differs from one only by case). If S is not a number and not one of the strings listed in TrueBoolStrs (see page 23034) or FalseBoolStrs (see page 23025), StrToBool raises an EConvertError (see page 22724) exception.

Note: If you do not assign a value to TrueBoolStrs (see page 23034), it is treated as if it contains the single string 'True'. Similarly, if you do not assign a value to FalseBoolStrs (see page 23025), it is treated as if it contains the single string 'False';

Note: The TrueBoolStrs (see page 23034) and FalseBoolStrs (see page 23025) variables are undefined when an application starts. Any use of StrToBool, BoolToStr (see page 22804) or TryStrToBool (see page 22982) will cause these variables to be initialised with the default values above.

See Also

BoolToStr (see page 22804)

TrueBoolStrs (see page 23034)

FalseBoolStrs (see page 23025)

159.3.202 SysUtils.StrToBoolDef Function

Converts a string to a Boolean value, with error default.

Pascal

```
function StrToBoolDef(const S: string; const Default: Boolean): Boolean;
```

C++

```
Boolean StrToBoolDef(const AnsiString S, const Boolean Default);
```

File

SysUtils

Description

StrToBoolDef converts the string specified by S to a Boolean. If S can be converted to a number, StrToBoolDef returns false if that number is 0, true otherwise. If S can't be converted to a number, StrToBoolDef returns true if S is any of the strings listed in TrueBoolStrs (see page 23034) (or differs from one only by case) and false if it is any of the strings listed in FalseBoolStrs (see page 23025) (or differs from one only by case). If S is not a number and not one of the strings listed in TrueBoolStrs (see page 23034) or FalseBoolStrs (see page 23025), StrToBoolDef returns Default.

Note: If you do not assign a value to TrueBoolStrs (see page 23034), it is treated as if it contains the single string 'True'. Similarly, if you do not assign a value to FalseBoolStrs (see page 23025), it is treated as if it contains the single string 'False';

Note: The TrueBoolStrs (see page 23034) and FalseBoolStrs (see page 23025) variables are undefined when an application starts. Any use of StrToBool (see page 22961), BoolToStr (see page 22804) or TryStrToBool (see page 22982) will cause these variables to be initialised with the default values above.

See Also

BoolToStr (see page 22804)

TrueBoolStrs (see page 23034)

FalseBoolStrs (see page 23025)

StrToBool (see page 22961)

159.3.203 SysUtils.StrToCurr Function

Converts a string to a Currency value.

Pascal

```
function StrToCurr(const S: string): Currency; overload;
function StrToCurr(const S: string; const FormatSettings: TFormatSettings): Currency; overload;
```

C++

```
Currency StrToCurr(const AnsiString S);
Currency StrToCurr(const AnsiString S, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Call StrToCurr to convert a string that represents a floating-point value to the corresponding Currency value. The source string, S, must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional 'E' or 'e' followed by a signed integer. Leading and trailing blanks are ignored.

Thousand separators and currency symbols are not allowed in the string. If the string doesn't contain a valid value, StrToCurr raises an EConvertError (see page 22724) exception.

The first form of StrToCurr is not thread-safe, because it uses localization information contained in global variables. The second form of StrToCurr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToCurr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

159.3.204 SysUtils.StrToCurrDef Function

Converts a string to a Currency value, with error default.

Pascal

```
function StrToCurrDef(const S: string; const Default: Currency): Currency; overload;
function StrToCurrDef(const S: string; const Default: Currency; const FormatSettings:
TFormatSettings): Currency; overload;
```

C++

```
Currency StrToCurrDef(const AnsiString S, const Currency Default);
Currency StrToCurrDef(const AnsiString S, const Currency Default, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call StrToCurrDef to convert a string that represents a floating-point value to the corresponding Currency value. The source string, S, must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional 'E' or 'e' followed by a signed integer. Leading and trailing blanks are ignored.

Thousand separators and currency symbols are not allowed in the string. If the string doesn't contain a valid value, StrToCurrDef returns Default.

The first form of StrToCurrDef is not thread-safe, because it uses localization information contained in global variables. The second form of StrToCurrDef, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToCurrDef, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

159.3.205 SysUtils.StrToDate Function

Converts a string to a TDateTime value.

Pascal

```
function StrToDate(const S: string): TDateTime; overload;
function StrToDate(const S: string; const FormatSettings: TFormatSettings): TDateTime;
overload;
```

C++

```
TDateTime StrToDate(const AnsiString S);
TDateTime StrToDate(const AnsiString S, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Call StrToDate to parse a string that specifies a date. If S does not contain a valid date, StrToDate raises an EConvertError (see page 22724) exception.

S must consist of two or three numbers, separated by the character defined by the DateSeparator (see page 23023) global variable or its TFormatSettings (see page 23001) equivalent. The order for month, day, and year is determined by the ShortDateFormat (see page 23030) global variable or its TFormatSettings (see page 23001) equivalent--possible combinations are m/d/y, d/m/y, and y/m/d.

If S contains only two numbers, it is interpreted as a date (m/d or d/m) in the current year.

Year values between 0 and 99 are converted using the TwoDigitYearCenturyWindow (see page 23034). This value is stored either in a global variable (first form) or as a field in the FormatSettings parameter (second form) See "Currency and Date (see page 22820)-Time (see page 22976) Formatting Variables" for more information.

The first form of StrToDate is not thread-safe, because it uses localization information contained in global variables. The second form of StrToDate, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToDate, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user enters a date in the edit box in the format
associated with the current locale (for example MM/DD/YY
format in the US), the string entered is converted to a
TDateTime value. This value is used to indicate the day of
the week the date represents.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char days[7][10] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday" };
  TDateTime dtDate = StrToDate(Edit1->Text);
  ShowMessage(Edit1->Text + AnsiString(" is a ") + days[dtDate.DayOfWeek() - 1]);
}
```

Delphi Examples:

```
{
This example uses an edit box and a button on a form. When
the user enters a date in the edit box in the format
associated with the current locale (for example MM/DD/YY
format in the US), the string entered is converted to a
TDateTime value. This value is used to indicate the day of
the week the date represents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ADate: TDateTime;
  days: array[1..7] of string;
begin
  days[1] := 'Sunday';
  days[2] := 'Monday';
  days[3] := 'Tuesday';
  days[4] := 'Wednesday';
  days[5] := 'Thursday';
  days[6] := 'Friday';
  days[7] := 'Saturday';
  ADate := StrToDate(Edit1.Text);
  ShowMessage(Edit1.Text + ' is a ' + days[SysUtils.DayOfWeek(ADate)]);
end;
```

159.3.206 SysUtils.StrToDateDef Function

Converts a string to a TDateTime value, with error default.

Pascal

```
function StrToDateDef(const S: string; const Default: TDateTime): TDateTime; overload;
function StrToDateDef(const S: string; const Default: TDateTime; const FormatSettings:
TFormatSettings): TDateTime; overload;
```

C++

```
TDateTime StrToDateDef(const AnsiString S, const TDateTime Default);
TDateTime StrToDateDef(const AnsiString S, const TDateTime Default, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call StrToDateDef to parse a string that specifies a date. If S does not contain a valid date, StrToDateDef returns Default.

S must consist of two or three numbers, separated by the character defined by the DateSeparator (see page 23023) global variable or its TFormatSettings (see page 23001) equivalent. The order for month, day, and year is determined by the ShortDateFormat (see page 23030) global variable or its TFormatSettings (see page 23001) equivalent--possible combinations are m/d/y, d/m/y, and y/m/d.

If S contains only two numbers, it is interpreted as a date (m/d or d/m) in the current year.

Year values between 0 and 99 are converted using the TwoDigitYearCenturyWindow (see page 23034). This value is stored either in a global variable (first form) or as a field in the FormatSettings parameter (second form) See "Currency and Date (see page 22820)-Time (see page 22976) Formatting Variables" for more information.

The first form of StrToDateDef is not thread-safe, because it uses localization information contained in global variables. The second form of StrToDateDef, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToDateDef, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

159.3.207 SysUtils.StrToDateTime Function

Converts a string to a TDateTime value.

Pascal

```
function StrToDateTime(const S: string): TDateTime; overload;
function StrToDateTime(const S: string; const FormatSettings: TFormatSettings): TDateTime;
overload;
```

C++

```
TDateTime StrToDateTime(const AnsiString S);
TDateTime StrToDateTime(const AnsiString S, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Call StrToDateTime to parse a string that specifies a date and time value. If S does not contain a valid date, StrToDateTime raises an EConvertError (see page 22724) exception.

The S parameter must use the current locale's date/time format. In the US, this is commonly MM/DD/YY HH:MM:SS format. Specifying AM or PM as part of the time is optional, as are the seconds. Use 24-hour time (7:45 PM is entered as 19:45, for example) if AM or PM is not specified.

Year values between 0 and 99 are converted using the TwoDigitYearCenturyWindow (see page 23034). This value is stored either in a global variable (first form) or as a field in the FormatSettings parameter (second form) See "Currency and Date (see page 22820)-Time (see page 22976) Formatting Variables" for more information.

The first form of StrToDateTime is not thread-safe, because it uses localization information contained in global variables. The second form of StrToDateTime, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToDateTime, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

159.3.208 SysUtils.StrToDateTimeDef Function

Converts a string to a TDateTime value with error default.

Pascal

```
function StrToDateTimeDef(const S: string; const Default: TDateTime): TDateTime; overload;
function StrToDateTimeDef(const S: string; const Default: TDateTime; const FormatSettings:
TFormatSettings): TDateTime; overload;
```

C++

```
TDateTime StrToDateTimeDef(const AnsiString S, const TDateTime Default);
TDateTime StrToDateTimeDef(const AnsiString S, const TDateTime Default, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call StrToDateTimeDef to parse a string that specifies a date and time value. If S does not contain a valid date, StrToDateTimeDef returns Default.

The S parameter must use the current locale's date/time format. In the US, this is commonly MM/DD/YY HH:MM:SS format. Specifying AM or PM as part of the time is optional, as are the seconds. Use 24-hour time (7:45 PM is entered as 19:45, for example) if AM or PM is not specified.

Year values between 0 and 99 are converted using the TwoDigitYearCenturyWindow (see page 23034). This value is stored either in a global variable (first form) or as a field in the FormatSettings parameter (second form) See "Currency and Date (see page 22820)-Time (see page 22976) Formatting Variables" for more information.

The first form of StrToDateTimeDef is not thread-safe, because it uses localization information contained in global variables. The second form of StrToDateTimeDef, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToDateTimeDef, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

See Also

DateTimeToStr (see page 22822)

StrToDate (see page 22963)

StrToTime (see page 22972)

CurrencyAndDatetimeFormatting

StrToDateTime (see page 22965)

159.3.209 SysUtils.StrToFloat Function

Converts a given string to a floating-point value.

Pascal

```
function StrToFloat(const S: string): Extended; overload;
function StrToFloat(const S: string; const FormatSettings: TFormatSettings): Extended;
overload;
```

C++

```
Extended StrToFloat(const AnsiString S);
Extended StrToFloat(const AnsiString S, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Use StrToFloat to convert a string, S, to a floating-point value. S must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional mantissa. The mantissa consists of 'E' or 'e' followed by an optional sign (+ or -) and a whole number. Leading and trailing blanks are ignored.

The DecimalSeparator (see page 23023) global variable or its TFormatSettings (see page 23001) equivalent defines the character that is used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If S doesn't contain a valid value, StrToFloat raises an EConvertError (see page 22724) exception.

The first form of StrToFloat is not thread-safe, because it uses localization information contained in global variables. The second form of StrToFloat, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToFloat, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

C++ Examples:

```
/*
The following example uses two buttons, a string grid, and
text edit on a form. When the conversion button is clicked,
the values across the top of the string grid are converted
using the formats along the left side.  Select a cell and
use the Alter Cell button and text edit to change the
values converted and the conversion formats.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int x = 1; x < StringGrid1->ColCount; x++)
  {
    for (int y = 1; y < StringGrid1->RowCount; y++)
    {
      StringGrid1->Cells[x][y] = FormatFloat(
      StringGrid1->Cells[0][y], StrToFloat(StringGrid1->Cells[x][0]));
    }
  }
```

```
}

int currCellCol, currCellRow;

void __fastcall TForm1::StringGrid1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  StringGrid1->MouseToCell(X, Y, currCellCol, currCellRow);
}

// Alter Cell
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  StringGrid1->Cells[currCellCol][currCellRow] = Edit1->Text;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "1234";
  StringGrid1->Cells[2][0] = "-1234";
  StringGrid1->Cells[3][0] = "0.5";
  StringGrid1->Cells[4][0] = "0";
  StringGrid1->Cells[0][1] = "";
  StringGrid1->Cells[0][2] = "0";
  StringGrid1->Cells[0][3] = "0.00";
  StringGrid1->Cells[0][4] = "#.##";
  StringGrid1->Cells[0][5] = "#,##0.00";
  StringGrid1->Cells[0][6] = "#,##0.00;(#,##0.00)";
  StringGrid1->Cells[0][7] = "#,##0.00;;Zero";
  StringGrid1->Cells[0][8] = "0.000E+00";
  StringGrid1->Cells[0][9] = "#.###E-0";
}
```

Delphi Examples:

```
{
The following example uses two buttons, a string grid, and
text edit on a form. When the conversion button is clicked,
the values across the top of the string grid are converted
using the formats along the left side.  Select a cell and
use the Alter Cell button and text edit to change the
values converted and the conversion formats.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  X, Y, I: Integer;
begin
  for X := 1 to StringGrid1.ColCount - 1 do
  begin
    for Y := 0 to StringGrid1.RowCount - 1 do
    begin
      StringGrid1.Cells[X,Y] := SysUtils.FormatFloat(
        StringGrid1.Cells[0,Y], StrToFloat(StringGrid1.Cells[X,0]));
    end;
  end;

end;

var currCellCol, currCellRow: Integer;

// Alter Cell
procedure TForm1.Button2Click(Sender: TObject);
begin
  StringGrid1.Cells[currCellCol,currCellRow] := Edit1.Text;
end;

procedure TForm1.StringGrid1MouseUp(Sender: TObject; Button: TMouseButton;
```

```
  Shift: TShiftState; X, Y: Integer);
begin
  StringGrid1.MouseToCell(X, Y, &currCellCol, &currCellRow);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  StringGrid1.Cells[1,0] := '1234';
  StringGrid1.Cells[2,0] := '-1234';
  StringGrid1.Cells[3,0] := '0.5';
  StringGrid1.Cells[4,0] := '0';
  StringGrid1.Cells[0,1] := '';
  StringGrid1.Cells[0,2] := '0';
  StringGrid1.Cells[0,3] := '0.00';
  StringGrid1.Cells[0,4] := '#.##';
  StringGrid1.Cells[0,5] := '#,##0.00';
  StringGrid1.Cells[0,6] := '#,##0.00;(#,##0.00)';
  StringGrid1.Cells[0,7] := '#,##0.00;;Zero';
  StringGrid1.Cells[0,8] := '0.000E+00';
  StringGrid1.Cells[0,9] := '#.###E-0';
end;
```

159.3.210 SysUtils.StrToFloatDef Function

Converts a given string to a floating-point value with error default.

Pascal

```
function StrToFloatDef(const S: string; const Default: Extended): Extended; overload;
function StrToFloatDef(const S: string; const Default: Extended; const FormatSettings:
TFormatSettings): Extended; overload;
```

C++

```
Extended StrToFloatDef(const AnsiString S, const Extended Default);
Extended StrToFloatDef(const AnsiString S, const Extended Default, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Use StrToFloatDef to convert a string, S, to a floating-point value. S must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional mantissa. The mantissa consists of 'E' or 'e' followed by an optional sign (+ or -) and a whole number. Leading and trailing blanks are ignored.

The DecimalSeparator (see page 23023) global variable or its TFormatSettings (see page 23001) equivalent defines the character that must be used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If S doesn't contain a valid value, StrToFloatDef returns Default.

The first form of StrToFloatDef is not thread-safe, because it uses localization information contained in global variables. The second form of StrToFloatDef, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToFloatDef, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

159.3.211 SysUtils.StrToInt Function

Converts a string that represents an integer (decimal or hex notation) to a number.

Pascal

```
function StrToInt(const S: string): Integer;
```

C++

```
int StrToInt(const AnsiString S);
```

File

SysUtils

Description

StrToInt converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number. If S does not represent a valid number, StrToInt raises an EConvertError (see page 22724) exception.

See Also

IntToHex (see page 22920)

IntToStr (see page 22921)

StrToIntDef (see page 22971)

TryStrToInt (see page 22986)

159.3.212 SysUtils.StrToInt64 Function

Converts a string that represents an integer (decimal or hex notation) to a number.

Pascal

```
function StrToInt64(const S: string): Int64;
```

C++

```
Int64 StrToInt64(const AnsiString S);
```

File

SysUtils

Description

StrToInt64 converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number. If S does not represent a valid number, StrToInt64 raises an EConvertError (see page 22724) exception.

See Also

IntToHex (see page 22920)

IntToStr (see page 22921)

StrToIntDef (see page 22971)

StrToInt64Def (see page 22971)

StrToInt (see page 22970)

TryStrToInt (see page 22986)

159.3.213 SysUtils.StrToInt64Def Function

Converts a string that represents an integer (decimal or hex notation) to a number.

Pascal

```
function StrToInt64Def(const S: string; const Default: Int64): Int64;
```

C++

```
Int64 StrToInt64Def(const AnsiString S, const Int64 Default);
```

File

SysUtils

Description

StrToInt64Def converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number. If S does not represent a valid number, StrToInt64Def returns the number passed in Default.

See Also

IntToStr (see page 22921)

StrToInt (see page 22970)

StrToInt64 (see page 22970)

StrToInt (see page 22970)

TryStrToInt64 (see page 22986)

159.3.214 SysUtils.StrToIntDef Function

Converts a string that represents an integer (decimal or hex notation) to a number with error default.

Pascal

```
function StrToIntDef(const S: string; Default: Integer): Integer;
```

C++

```
int StrToIntDef(const AnsiString S, int Default);
```

File

SysUtils

Description

StrToIntDef converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number. If S does not represent a valid number, StrToIntDef returns Default.

See Also

IntToHex (see page 22920)

IntToStr (see page 22921)

StrToInt (see page 22970)

TryStrToInt (see page 22986)

159.3.215 SysUtils.StrToTime Function

Converts a string to a TDateTime value.

Pascal

```
function StrToTime(const S: string): TDateTime; overload;
function StrToTime(const S: string; const FormatSettings: TFormatSettings): TDateTime;
overload;
```

C++

```
TDateTime StrToTime(const AnsiString S);
TDateTime StrToTime(const AnsiString S, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Call StrToTime to parse a string that specifies a time value. If S does not contain a valid time, StrToTime raises an EConvertError (see page 22724) exception.

The S parameter must consist of two or three numbers, separated by the character defined by the TimeSeparator (see page 23033) global variable or its TFormatSettings (see page 23001) equivalent, optionally followed by an AM or PM indicator. The numbers represent hour, minute, and (optionally) second, in that order. If the time is followed by AM or PM, it is assumed to be in 12-hour clock format. If no AM or PM indicator is included, the time is assumed to be in 24-hour clock format.

The first form of StrToTime is not thread-safe, because it uses localization information contained in global variables. The second form of StrToTime, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToTime, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

159.3.216 SysUtils.StrToTimeDef Function

Converts a string to a TDateTime value with error default.

Pascal

```
function StrToTimeDef(const S: string; const Default: TDateTime): TDateTime; overload;
function StrToTimeDef(const S: string; const Default: TDateTime; const FormatSettings:
TFormatSettings): TDateTime; overload;
```

C++

```
TDateTime StrToTimeDef(const AnsiString S, const TDateTime Default);
TDateTime StrToTimeDef(const AnsiString S, const TDateTime Default, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call StrToTimeDef to parse a string that specifies a time value. If S does not contain a valid time, StrToTimeDef returns Default.

The S parameter must consist of two or three numbers, separated by the character defined by the TimeSeparator (see page 23033) global variable or its TFormatSettings (see page 23001) equivalent, optionally followed by an AM or PM indicator. The numbers represent hour, minute, and (optionally) second, in that order. If the time is followed by AM or PM, it is assumed to be in 12-hour clock format. If no AM or PM indicator is included, the time is assumed to be in 24-hour clock format.

The first form of StrToTimeDef is not thread-safe, because it uses localization information contained in global variables. The second form of StrToTimeDef, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToTimeDef, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

159.3.217 SysUtils.StrUpper Function

Returns a string in upper case.

Pascal

```
function StrUpper(Str: PChar): PChar;
```

C++

```
const char * StrUpper(const char * Str);
```

File

SysUtils

Description

StrUpper converts Str to uppercase and returns Str.

Note: If the source string contains international characters, use AnsiStrUpper (see page 22800) instead. If the source string is an AnsiString (long string), use UpperCase (see page 22988) (or AnsiUpperCase (see page 22800)) instead.

See Also

AnsiUpperCase (see page 22800)

UpperCase (see page 22988)

StrLower (see page 22956)

159.3.218 SysUtils.Supports Function

Indicates whether a given object or interface supports a specified interface.

Pascal

```
function Supports(const Instance: IInterface; const IID: TGUID; out Intf): Boolean; overload;
function Supports(const Instance: TObject; const IID: TGUID; out Intf): Boolean; overload;
function Supports(const Instance: IInterface; const IID: TGUID): Boolean; overload;
function Supports(const Instance: TObject; const IID: TGUID): Boolean; overload;
function Supports(const AClass: TClass; const IID: TGUID): Boolean; overload;
```

C++

```
Boolean Supports(const IInterface Instance, const TGUID IID,  Intf);
Boolean Supports(const TObject * Instance, const TGUID IID,  Intf);
Boolean Supports(const IInterface Instance, const TGUID IID);
Boolean Supports(const TObject * Instance, const TGUID IID);
Boolean Supports(const TClass AClass, const TGUID IID);
```

File

SysUtils

Description

Call Supports to determine whether the object or interface specified by Instance, or the class specified by AClass, supports the interface identified by the IID parameter. If Instance supports the interface, Supports returns the interface as the Intf parameter and returns true. If AClass supports the interface, Supports does not return an interface, but still returns true. If the interface specified by IID is not supported, Supports returns false.

On the Win32 platform, to determine whether the interface specified by IID is supported, Supports calls the QueryInterface method of the supplied interface (or the interface of the supplied object). However, unlike calling QueryInterface directly, Supports lets you pass a nil (Delphi) or NULL (C++) value as the instance.

On the .NET platform, Supports uses the typecasting facilities of the .NET framework.

See Also

TComponent (see page 2762)

159.3.219 SysUtils.SysErrorMessage Function

Converts OS error codes into strings.

Pascal

```
function SysErrorMessage(ErrorCode: Integer): string;
```

C++

```
AnsiString SysErrorMessage(int ErrorCode);
```

File

SysUtils

Description

SysErrorMessage returns an error message string that corresponds to the specified OS error code.

See Also

RaiseLastOSError (see page 22932)

EWin32Error (see page 22749)

159.3.220 SysUtils.SystemTimeToDateTime Function

Converts a system time value into a TDateTime value.

Pascal

```
function SystemTimeToDateTime(const SystemTime: TSystemTime): TDateTime;
```

C++

```
TDateTime SystemTimeToDateTime(const TSystemTime SystemTime);
```

File

SysUtils

Description

Call SystemTimeToDateTime to convert a value from Windows 32's TSystemTime (Delphi) or _SYSTEMTIME (C++) API format to a TDateTime value. SystemTimeToDateTime is available on Windows only.

See Also

DateTimeToSystemTime (see page 22825)

159.3.221 SysUtils.TextToFloat Function

Converts a null-terminated string to a floating-point value.

Pascal

```
function TextToFloat(Buffer: PChar; var Value; ValueType: TFloatValue): Boolean; overload;
function TextToFloat(Buffer: PChar; var Value; ValueType: TFloatValue; const FormatSettings:
TFormatSettings): Boolean; overload;
```

C++

```
Boolean TextToFloat(const char * Buffer,  Value, TFloatValue ValueType);
Boolean TextToFloat(const char * Buffer,  Value, TFloatValue ValueType, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Use TextToFloat to convert a null-terminated string, Buffer, to a floating-point value. The string must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional mantissa. The mantissa consists of 'E' or 'e' followed by an optional sign (+ or -) and a whole number. Leading and trailing blanks are ignored.

The converted value is assigned to the Value parameter, which must be of type Extended or Currency, as indicated by the ValueType parameter.

TextToFloat returns true if the conversion is successful, or false if the Buffer does not contain a valid floating-point value.

The first form of TextToFloat is not thread-safe, because it uses localization information contained in global variables. The second form of TextToFloat, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of TextToFloat, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

159.3.222 SysUtils.Time Function

Returns the current time.

Pascal

```
function Time: TDateTime;
```

C++

```
TDateTime Time();
```

File

SysUtils

Description

Time and GetTime (see page 22916) return the current time as a TDateTime value. The two functions are completely equivalent.

Note: Time clashes with an Xlib function of the same name. Use GetTime (see page 22916) in Linux and cross-platform code.

See Also

Date (see page 22820)

DecodeTime (see page 22830)

Now (see page 22930)

TimeToStr (see page 22977)

159.3.223 SysUtils.TimeStampToDateTime Function

Converts a TTimeStamp (see page 23009) value into the corresponding TDateTime value.

Pascal

```
function TimeStampToDateTime(const TimeStamp: TTimeStamp): TDateTime;
```

C++

```
TDateTime TimeStampToDateTime(const TTimeStamp TimeStamp);
```

File

SysUtils

Description

Call TimeStampToDateTime to convert a TTimeStamp (see page 23009) value into a TDateTime value. TTimeStamp (see page 23009) values represent time as separate date and time values, where the date is the number of calendar days since the start of the current calendar (that is, January 1, 0001 would have a value of 1), and time is the number of milliseconds since midnight. TDateTime values represent time as the number of days (including fractional days) that have elapsed since 12:00 am on December 30, 1899.

See Also

DateTimeToTimeStamp (see page 22825)

159.3.224 SysUtils.TimeStampToMSecs Function

Converts a TTimeStamp (see page 23009) value into an absolute number of milliseconds.

Pascal

```
function TimeStampToMSecs(const TimeStamp: TTimeStamp): Comp;
```

C++

```
Comp TimeStampToMSecs(const TTimeStamp TimeStamp);
```

File

SysUtils

Description

Call TimeStampToMSecs to convert a TTimeStamp (see page 23009) value, into the total number of milliseconds that comprise the Date (see page 22820) field plus the number of milliseconds given in the Time (see page 22976) field.

See Also

MSecsToTimeStamp (see page 22929)

TimeStampToDateTime (see page 22976)

159.3.225 SysUtils.TimeToStr Function

Returns a string that represents a TDateTime value.

Pascal

```
function TimeToStr(const DateTime: TDateTime): string; overload;
function TimeToStr(const DateTime: TDateTime; const FormatSettings: TFormatSettings): string;
overload;
```

C++

```
AnsiString TimeToStr(const TDateTime DateTime);
AnsiString TimeToStr(const TDateTime DateTime, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

TimeToStr converts the Time (see page 22976) parameter, a TDateTime value, to a string. The conversion uses the format specified by the LongTimeFormat (see page 23028) global variable or its TFormatSettings (see page 23001) equivalent. Change the format of the string by changing the values of some of the date and time variables.

The first form of TimeToStr is not thread-safe, because it uses localization information contained in global variables. The second form of TimeToStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of TimeToStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

C++ Examples:

```
/*
```

```
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TDateTime dt;
  try
  {
    dt = EncodeTime((Word)StrToInt(Edit1->Text), (Word)StrToInt(Edit2->Text),
(Word)StrToInt(Edit3->Text), (Word)StrToInt(Edit4->Text));
    Label1->Caption = TimeToStr(dt);
  }
  catch(...)
  {
    ShowMessage("Invalid hour, minute, second, or millisecond.");
  }
}
```

Delphi Examples:

```
{
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  MyDate: TDateTime;
begin
  MyDate := SysUtils.EncodeTime(StrToInt(Edit1.Text), StrToInt(Edit2.Text),
StrToInt(Edit3.Text), StrToInt(Edit4.Text));
  Label1.Caption := TimeToStr(MyDate);
end;
```

See Also

DateTimeToStr (see page 22822)

DateToStr (see page 22825)

StrToDateTime (see page 22965)

Time (see page 22976)

GetLocaleFormatSettings (see page 22913)

159.3.226 SysUtils.Trim Function

Trims leading and trailing spaces and control characters from a string.

Pascal

```
function Trim(const S: string): string; overload;
function Trim(const S: WideString): WideString; overload;
```

C++

```
AnsiString Trim(const AnsiString S);
BSTR Trim(const BSTR S);
```

File

SysUtils

Description

Trim removes leading and trailing spaces and control characters from the given string S.

See Also

TrimLeft (see page 22979)

TrimRight (see page 22979)

159.3.227 SysUtils.TrimLeft Function

Trims leading spaces and control characters from a string.

Pascal

```
function TrimLeft(const S: string): string; overload;
function TrimLeft(const S: WideString): WideString; overload;
```

C++

```
AnsiString TrimLeft(const AnsiString S);
BSTR TrimLeft(const BSTR S);
```

File

SysUtils

Description

TrimLeft returns a copy of the string S with leading spaces and control characters removed.

See Also

Trim (see page 22978)

TrimRight (see page 22979)

159.3.228 SysUtils.TrimRight Function

Trims trailing spaces and control characters from a string.

Pascal

```
function TrimRight(const S: string): string; overload;
function TrimRight(const S: WideString): WideString; overload;
```

C++

```
AnsiString TrimRight(const AnsiString S);
BSTR TrimRight(const BSTR S);
```

File

SysUtils

Description

TrimRight returns a copy of the string S with trailing spaces and control characters removed.

See Also

TrimLeft (see page 22979)

Trim (see page 22978)

159.3.229 SysUtils.TryEncodeDate Function

Creates a TDateTime value that represents a specified Year, Month, and Day.

Pascal

```
function TryEncodeDate(Year: Word; Month: Word; Day: Word; out Date: TDateTime): Boolean;
```

C++

```
Boolean TryEncodeDate(Word Year, Word Month, Word Day, TDateTime Date);
```

File

SysUtils

Description

TryEncodeDate generates a TDateTime value from the values specified as the Year, Month, and Day parameters.

The year must be between 1 and 9999.

Valid Month values are 1 through 12.

Valid Day values are 1 through 28, 29, 30, or 31, depending on the Month value. For example, the possible Day values for month 2 (February) are 1 through 28 or 1 through 29, depending on whether or not the Year value specifies a leap year.

The Date (see page 22820) parameter returns the generated value if TryEncodeDate returns true.

If the specified values are not within range, TryEncodeDate returns false.

See Also

DateToStr (see page 22825)

DecodeDate (see page 22828)

TryEncodeTime (see page 22980)

TryEncodeDateTime (see page 6697)

EncodeDate (see page 22836)

159.3.230 SysUtils.TryEncodeTime Function

Creates a TDateTime value for a specified Hour, Min, Sec, and MSec.

Pascal

```
function TryEncodeTime(Hour: Word; Min: Word; Sec: Word; MSec: Word; out Time: TDateTime):
Boolean;
```

C++

```
Boolean TryEncodeTime(Word Hour, Word Min, Word Sec, Word MSec, TDateTime Time);
```

File

SysUtils

Description

EncodeTime (see page 22837) encodes the given hour, minute, second, and millisecond into a TDateTime value.

Valid Hour values are 0 through 24. If Hour is 24, Min, Sec, and MSec must all be 0, and the resulting TDateTime value represents midnight (12:00:00:000 AM) of the following day.

Valid Min and Sec values are 0 through 59.

Valid MSec values are 0 through 999.

The Time (see page 22976) paremter returns the generated TDateTime value of the specifies values are in range. This value is a number between 0 and 1 (inclusive) that indicates the fractional part of a day given by the specified time or (if 1.0) midnight on the following day. The value 0 corresponds to midnight, 0.5 corresponds to noon, 0.75 corresponds to 6:00 pm, and so on.

See Also

DecodeTime (see page 22830)

TryEncodeDate (see page 22980)

TryEncodeDateTime (see page 6697)

EncodeTime (see page 22837)

159.3.231 SysUtils.TryFloatToCurr Function

Converts a floating-point value to a Currency value.

Pascal

```
function TryFloatToCurr(const Value: Extended; out AResult: Currency): Boolean;
```

C++

```
Boolean TryFloatToCurr(const Extended Value, Currency AResult);
```

File

SysUtils

Description

TryFloatToCurr converts the specified floating-point value into a Currency value.

Value is the floating-point value to convert.

AResult returns the equivalent Currency value, if possible.

If TryFloatToCurr is successful, it returns true. If Value is not within the valid range for a Currency value, TryFloatToCurr returns false.

See Also

FloatToCurr (see page 22889)

159.3.232 SysUtils.TryFloatToDateTime Function

Converts a floating-point value to a TDateTime value.

Pascal

```
function TryFloatToDateTime(const Value: Extended; out AResult: TDateTime): Boolean;
```

C++

```
Boolean TryFloatToDateTime(const Extended Value, TDateTime AResult);
```

File

SysUtils

Description

TryFloatToDateTime converts a floating-point value into the corresponding TDateTime value.

Value is the floating-point value to convert.

AResult returns the corresponding TDateTime value if possible.

TryFloatToDateTime returns true if the conversion was successful, false otherwise.

See Also

MinDateTime (see page 23052)

FloatToDateTime (see page 22889)

159.3.233 SysUtils.TryStrToBool Function

Converts a string to a Boolean value, with a Boolean success code.

Pascal

```
function TryStrToBool(const S: string; out Value: Boolean): Boolean;
```

C++

```
Boolean TryStrToBool(const AnsiString S, Boolean Value);
```

File

SysUtils

Description

TryStrToBool converts the string specified by S to a Boolean and sets Value to the result. If S can be converted to a number, TryStrToBool sets Value to false if that number is 0, true otherwise. If S can't be converted to a number, TryStrToBool returns true if S is any of the strings listed in TrueBoolStrs (see page 23034) (or differs from one only by case) and false if it is any of the strings listed in FalseBoolStrs (see page 23025) (or differs from one only by case). If S is not a number and not one of the strings listed in TrueBoolStrs (see page 23034) or FalseBoolStrs (see page 23025), TryStrToBool returns false to indicate that the conversion failed. If the conversion is successful, TryStrToBool returns true.

Note: If you do not assign a value to TrueBoolStrs (see page 23034), it is treated as if it contains the single string 'True'. Similarly, if you do not assign a value to FalseBoolStrs (see page 23025), it is treated as if it contains the single string 'False';

Note: The TrueBoolStrs (see page 23034) and FalseBoolStrs (see page 23025) variables are undefined when an application starts. Any use of StrToBool (see page 22961), BoolToStr (see page 22804) or TryStrToBool will cause these variables to be initialised with the default values above.

See Also

BoolToStr (see page 22804)

TrueBoolStrs (see page 23034)

FalseBoolStrs (see page 23025)

StrToBool (see page 22961)

159.3.234 SysUtils.TryStrToCurr Function

Converts a string to a Currency value, with a Boolean success code.

Pascal

```
function TryStrToCurr(const S: string; out Value: Currency): Boolean; overload;
function TryStrToCurr(const S: string; out Value: Currency; const FormatSettings:
TFormatSettings): Boolean; overload;
```

C++

```
Boolean TryStrToCurr(const AnsiString S, Currency Value);
Boolean TryStrToCurr(const AnsiString S, Currency Value, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Call TryStrToCurr to convert a string that represents a floating-point value to the corresponding Currency value. The source string, S, must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional 'E' or 'e' followed by a signed integer. Leading and trailing blanks are ignored.

The DecimalSeparator (see page 23023) global variable or its TFormatSettings (see page 23001) equivalent defines the character that is used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If the string doesn't contain a valid value, TryStrToCurr returns Default.

The first form of TryStrToCurr is not thread-safe, because it uses localization information contained in global variables. The second form of TryStrToCurr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of TryStrToCurr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

159.3.235 SysUtils.TryStrToDate Function

Converts a string to a TDateTime value, with a Boolean success code.

Pascal

```
function TryStrToDate(const S: string; out Value: TDateTime): Boolean; overload;
function TryStrToDate(const S: string; out Value: TDateTime; const FormatSettings:
TFormatSettings): Boolean; overload;
```

C++

```
Boolean TryStrToDate(const AnsiString S, TDateTime Value);
Boolean TryStrToDate(const AnsiString S, TDateTime Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call TryStrToDate to parse a string that specifies a date. If S does not contain a valid date, TryStrToDate returns false.

S must consist of two or three numbers, separated by the character defined by the DateSeparator (see page 23023) global variable or its TFormatSettings (see page 23001) equivalent. The order for month, day, and year is determined by the ShortDateFormat (see page 23030) global variable or its TFormatSettings (see page 23001) equivalent--possible combinations are m/d/y, d/m/y, and y/m/d.

If S contains only two numbers, it is interpreted as a date (m/d or d/m) in the current year.

Year values between 0 and 99 are converted using the TwoDigitYearCenturyWindow (see page 23034). This value is stored either in a global variable (first form) or as a field in the FormatSettings parameter (second form) See "Currency and Date (see page 22820)-Time (see page 22976) Formatting Variables" for more information.

The first form of TryStrToDate is not thread-safe, because it uses localization information contained in global variables. The second form of TryStrToDate, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of TryStrToDate, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

159.3.236 SysUtils.TryStrToDateTime Function

Converts a string to a TDateTime value with a Boolean success code.

Pascal

```
function TryStrToDateTime(const S: string; out Value: TDateTime): Boolean; overload;
function TryStrToDateTime(const S: string; out Value: TDateTime; const FormatSettings:
TFormatSettings): Boolean; overload;
```

C++

```
Boolean TryStrToDateTime(const AnsiString S, TDateTime Value);
Boolean TryStrToDateTime(const AnsiString S, TDateTime Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call TryStrToDateTime to parse a string that specifies a date and time value. If S does not contain a valid date, TryStrToDateTime returns false.

The S parameter must use the current locale's date/time format. In the US, this is commonly MM/DD/YY HH:MM:SS format. Specifying AM or PM as part of the time is optional, as are the seconds. Use 24-hour time (7:45 PM is entered as 19:45, for example) if AM or PM is not specified.

Year values between 0 and 99 are converted using the TwoDigitYearCenturyWindow (see page 23034). This value is stored either in a global variable (first form) or as a field in the FormatSettings parameter (second form) See "Currency and Date (see

page 22820)-Time (see page 22976) Formatting Variables" for more information.

The first form of TryStrToDateTime is not thread-safe, because it uses localization information contained in global variables. The second form of TryStrToDateTime, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of TryStrToDateTime, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

159.3.237 SysUtils.TryStrToFloat Function

Converts a given string to a floating-point value with a boolean success code.

Pascal

```
function TryStrToFloat(const S: string; out Value: Extended): Boolean; overload;
function TryStrToFloat(const S: string; out Value: Extended; const FormatSettings:
TFormatSettings): Boolean; overload;
function TryStrToFloat(const S: string; out Value: Double): Boolean; overload;
function TryStrToFloat(const S: string; out Value: Double; const FormatSettings:
TFormatSettings): Boolean; overload;
function TryStrToFloat(const S: string; out Value: Single): Boolean; overload;
function TryStrToFloat(const S: string; out Value: Single; const FormatSettings:
TFormatSettings): Boolean; overload;
```

C++

```
Boolean TryStrToFloat(const AnsiString S, Extended Value);
Boolean TryStrToFloat(const AnsiString S, Extended Value, const TFormatSettings
FormatSettings);
Boolean TryStrToFloat(const AnsiString S, Double Value);
Boolean TryStrToFloat(const AnsiString S, Double Value, const TFormatSettings FormatSettings);
Boolean TryStrToFloat(const AnsiString S, Single Value);
Boolean TryStrToFloat(const AnsiString S, Single Value, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Use TryStrToFloat to convert a astring, S, to a floating-point value. S must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional mantissa. The mantissa consists of 'E' or 'e' followed by an optional sign (+ or -) and a whole number. Leading and trailing blanks are ignored.

The DecimalSeparator (see page 23023) global variable defines the character that must be used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If S doesn't contain a valid value, TryStrToFloat returns Default.

TryStrToFloat places the result in Value, and returns true if the conversion was successful, false otherwise.

The first three forms of TryStrToFloat are not thread-safe, because they use localization information contained in global variables. The second three forms of TryStrToFloat, which are thread-safe, refer to localization information contained in the FormatSettings parameter. Before calling the thread-safe forms of TryStrToFloat, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

159.3.238 SysUtils.TryStrToInt Function

Converts a string that represents an integer (decimal or hex notation) to a number with Boolean success code.

Pascal

```
function TryStrToInt(const S: string; out Value: Integer): Boolean;
```

C++

```
Boolean TryStrToInt(const AnsiString S, int Value);
```

File

SysUtils

Description

TryStrToInt converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number, which is assigned to Value. If S does not represent a valid number, TryStrToInt returns false; otherwise TryStrToInt returns true.

To accept decimal but not hexadecimal values in the input string, you may use code like this:

```
function TryDecimalStrToInt( const S: string; out Value: Integer): Boolean;
begin
   result := ( pos( '$', S ) = 0 ) and TryStrToInt( S, Value );
end;
```

See Also

IntToHex (see page 22920)

IntToStr (see page 22921)

StrToInt (see page 22970)

StrToIntDef (see page 22971)

159.3.239 SysUtils.TryStrToInt64 Function

Converts a string that represents an integer (decimal or hex notation) to a number.

Pascal

```
function TryStrToInt64(const S: string; out Value: Int64): Boolean;
```

C++

```
Boolean TryStrToInt64(const AnsiString S, Int64 Value);
```

File

SysUtils

Description

TryStrToInt64 converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number, which is assigned to Value. If S does not represent a valid number, TryStrToInt64 returns false; otherwise TryStrToInt64 returns true.

To accept decimal but not hexadecimal values in the input string, you may use code like this:

```
function TryDecimalStrToInt( const S: string; out Value: Integer): Boolean;
begin
   result := ( pos( '$', S ) = 0 ) and TryStrToInt( S, Value );
end;
```

See Also

IntToStr (see page 22921)

StrToInt (see page 22970)

StrToInt64 (see page 22970)

StrToInt (see page 22970)

159.3.240 SysUtils.TryStrToTime Function

Converts a string to a TDateTime value with an error default.

Pascal

```
function TryStrToTime(const S: string; out Value: TDateTime): Boolean; overload;
function TryStrToTime(const S: string; out Value: TDateTime; const FormatSettings:
TFormatSettings): Boolean; overload;
```

C++

```
Boolean TryStrToTime(const AnsiString S, TDateTime Value);
Boolean TryStrToTime(const AnsiString S, TDateTime Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call TryStrToTime to parse a string that specifies a time value. If S does not contain a valid time, TryStrToTime returns false.

The S parameter must consist of two or three numbers, separated by the character defined by the TimeSeparator (see page 23033) global variable or its TFormatSettings (see page 23001) equivalent, optionally followed by an AM or PM indicator. The numbers represent hour, minute, and (optionally) second, in that order. If the time is followed by AM or PM, it is assumed to be in 12-hour clock format. If no AM or PM indicator is included, the time is assumed to be in 24-hour clock format.

The first form of TryStrToTime is not thread-safe, because it uses localization information contained in global variables. The second form of TryStrToTime, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of TryStrToTime, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

159.3.241 SysUtils.UnloadPackage Function

Unloads a package.

Pascal

```
procedure UnloadPackage(Module: HMODULE);
```

C++

```
UnloadPackage(HMODULE Module);
```

File

SysUtils

Description

Call UnloadPackage to unload a package that was loaded using the LoadPackage (see page 22927) function. UnloadPackage performs any finalization of units contained in the package and then unloads the package BPL.

See Also

LoadPackage (see page 22927)

159.3.242 SysUtils.UpperCase Function

Returns a copy of a string in uppercase.

Pascal

```
function UpperCase(const S: string): string; overload;
function UpperCase(const S: string; LocaleOptions: TLocaleOptions): string; overload;
```

C++

```
AnsiString UpperCase(const AnsiString S);
AnsiString UpperCase(const AnsiString S, TLocaleOptions LocaleOptions);
```

File

SysUtils

Description

UpperCase returns a copy of the string S, with the same text but with all 7-bit ASCII characters between 'a' and 'z' converted to uppercase. To convert 8-bit international characters, use AnsiUpperCase (see page 22800) instead.

C++ Examples:

```
/*
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
list of strings in the list box. When the application runs
and the button is clicked, the strings in the list box
become uppercase.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int I = 0; I < ListBox1->Items->Count; I++)
    ListBox1->Items->Strings[I] = UpperCase(ListBox1->Items->Strings[I]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");
}
```

Delphi Examples:

```
{
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
```

```
  list of strings in the list box. When the application runs
  and the button is clicked, the strings in the list box
  become uppercase.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    I: Integer;
  begin
    for I := 0 to ListBox1.Items.Count -1 do
      ListBox1.Items[I] := SysUtils.UpperCase(ListBox1.Items[I]);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    Listbox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;
```

See Also

AnsiUpperCase (see page 22800)

LowerCase (see page 22928)

159.3.243 SysUtils.WideCompareStr Function

Compares Unicode strings based on the current locale with case sensitivity.

Pascal

```
function WideCompareStr(const S1: WideString; const S2: WideString): Integer;
```

C++

```
int WideCompareStr(const BSTR S1, const BSTR S2);
```

File

SysUtils

Description

WideCompareStr compares S1 to S2, with case sensitivity. The compare operation is controlled by the current locale. The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

Note: Most locales consider lowercase characters to be less than the corresponding uppercase characters. This is in contrast to ASCII order, in which lowercase characters are greater than uppercase characters. with S1 set to 'a' and S2 set to 'A', WideCompareStr returns a value lsess than zero, while CompareStr (see page 22813) returns a value greater than zero.

See Also

WideCompareText (see page 22990)

WideSameStr (see page 22992)

159.3.244 SysUtils.WideCompareText Function

Compares Unicode strings based on the current locale without case sensitivity.

Pascal

```
function WideCompareText(const S1: WideString; const S2: WideString): Integer;
```

C++

```
int WideCompareText(const BSTR S1, const BSTR S2);
```

File

SysUtils

Description

WideCompareText compares S1 to S2, without case sensitivity. The compare operation is controlled by the current locale. WideCompareText returns a value less than 0 if S1 < S2, a value greater than 0 if S1 > S2, and returns 0 if S1 = S2.

See Also

WideCompareStr (see page 22989)

WideSameText (see page 22993)

159.3.245 SysUtils.WideFmtStr Function

Assembles a formatted Unicode string using a format string and an array of arguments.

Pascal

```
procedure WideFmtStr(var Result: WideString; const Format: WideString; const Args: array of
const); overload;
procedure WideFmtStr(var Result: WideString; const Format: WideString; const Args: array of
const; const FormatSettings: TFormatSettings); overload;
```

C++

```
WideFmtStr(BSTR Result, const BSTR Format, const array of const Args);
WideFmtStr(BSTR Result, const BSTR Format, const array of const Args, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

This function formats the series of arguments in the open array Args into a Unicode string. Formatting is controlled by the format string Format (see page 22897); the results are returned in the parameter Result.

For information on the format strings, see Format (see page 22897) Strings.

The first form of WideFmtStr is not thread-safe, because it uses localization information contained in global variables. The second form of WideFmtStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of WideFmtStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

See Also

GetLocaleFormatSettings (see page 22913)

159.3.246 SysUtils.WideFormat Function

Returns a formatted Unicode string assembled from a format string and an array of arguments.

Pascal

```
function WideFormat(const Format: WideString; const Args: array of const): WideString;
overload;
function WideFormat(const Format: WideString; const Args: array of const; const
FormatSettings: TFormatSettings): WideString; overload;
```

C++

```
BSTR WideFormat(const BSTR Format, const array of const Args);
BSTR WideFormat(const BSTR Format, const array of const Args, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

This function formats the series of arguments in the open array Args. Formatting is controlled by the format string Format (see page 22897); the results are returned in the function result as a Unicode string.

For information on the format strings, see Format (see page 22897) Strings.

The first form of WideFormat is not thread-safe, because it uses localization information contained in global variables. The second form of WideFormat, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of WideFormat, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

See Also

GetLocaleFormatSettings (see page 22913)

159.3.247 SysUtils.WideFormatBuf Function

Formats the arguments from an array, placing the result in a buffer.

Pascal

```
function WideFormatBuf(var Buffer; BufLen: Cardinal; const Format; FmtLen: Cardinal; const
Args: array of const): Cardinal; overload;
function WideFormatBuf(var Buffer; BufLen: Cardinal; const Format; FmtLen: Cardinal; const
Args: array of const; const FormatSettings: TFormatSettings): Cardinal; overload;
```

C++

```
unsigned WideFormatBuf( Buffer, unsigned BufLen, const  Format, unsigned FmtLen, const array
of const Args);
unsigned WideFormatBuf( Buffer, unsigned BufLen, const  Format, unsigned FmtLen, const array
of const Args, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

This function formats the series of arguments in the open array Args. Formatting is controlled by the Unicode format string Format (see page 22897) (whose length is given by FmtLen); the results are returned in Buffer (whose length is given by BufLen). The function returns the number of Unicode characters in the formatted string.

Note: Both buffers contain wide (Unicode) characters. BufLen and FmtLen must specify the number of Unicode characters in their respective buffers, not the number of bytes.

For information on the format strings, see Format (see page 22897) Strings.

The first form of WideFormatBuf is not thread-safe, because it uses localization information contained in global variables. The second form of WideFormatBuf, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of WideFormatBuf, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 22913).

159.3.248 SysUtils.WideLowerCase Function

Returns Unicode string converted to lower case.

Pascal

```
function WideLowerCase(const S: WideString): WideString;
```

C++

```
BSTR WideLowerCase(const BSTR S);
```

File

SysUtils

Description

WideLowerCase returns S converted to lower case. The conversion uses the current locale.

See Also

WideUpperCase (see page 22993)

LowerCase (see page 22928)

159.3.249 SysUtils.WideSameStr Function

Compares Unicode strings based on the current locale with case sensitivity.

Pascal

```
function WideSameStr(const S1: WideString; const S2: WideString): Boolean;
```

C++

```
Boolean WideSameStr(const BSTR S1, const BSTR S2);
```

File

SysUtils

Description

WideSameStr compares S1 to S2, with case sensitivity. The compare operation is controlled by the current locale. The return value is true if the two strings have the same value, Ffalse otherwise.

See Also

WideSameText (see page 22993)

WideCompareStr (see page 22989)

159.3.250 SysUtils.WideSameText Function

Compares Unicode strings based on the current locale without case sensitivity.

Pascal

```
function WideSameText(const S1: WideString; const S2: WideString): Boolean;
```

C++

```
Boolean WideSameText(const BSTR S1, const BSTR S2);
```

File

SysUtils

Description

WideSameText compares S1 to S2, without case sensitivity. The compare operation is controlled by the current locale. The return value is true if the two strings have the same value, false otherwise.

See Also

WideSameStr (see page 22992)

WideCompareText (see page 22990)

159.3.251 SysUtils.WideUpperCase Function

Returns Unicode string converted to upper case.

Pascal

```
function WideUpperCase(const S: WideString): WideString;
```

C++

```
BSTR WideUpperCase(const BSTR S);
```

File

SysUtils

Description

WideUpperCase returns S converted to upper case. The conversion uses the current locale.

See Also

WideLowerCase (see page 22992)

UpperCase (see page 22988)

159.3.252 SysUtils.Win32Check Function

Checks the return value of a Windows API call and raises an appropriate exception when it indicates failure.

Pascal

```
function Win32Check(RetVal: BOOL): BOOL; platform;
```

C++

```
BOOL Win32Check(BOOL RetVal);
```

File

SysUtils

Description

Call Win32Check with the return value of a Windows API call that returns a Boolean to indicate success or failure. If the Windows API returns false (indicating failure), Win32Check calls RaiseLastOSError (see page 22932) to raise an exception. If the Windows function returns true, Win32Check returns true.

Win32Check is available on Windows only.

See Also

RaiseLastOSError (see page 22932)

159.3.253 SysUtils.WrapText Function

Splits a string into multiple lines as its length approaches a specified size.

Pascal

```
function WrapText(const Line: string; const BreakStr: string; const BreakChars: TSysCharSet;
MaxCol: Integer): string; overload;
function WrapText(const Line: string; MaxCol: Integer = 45): string; overload;
```

C++

```
AnsiString WrapText(const AnsiString Line, const AnsiString BreakStr, const TSysCharSet
BreakChars, int MaxCol);
AnsiString WrapText(const AnsiString Line, int MaxCol = 45);
```

File

SysUtils

Description

WrapText scans a string for occurrences of any of the characters specified by nBreakChars and inserts a line-break, specified by BreakStr, at the last occurrence of a character in nBreakChars before MaxCol. Line is the text WrapText scans. MaxCol is the maximum line length.

If the BreakStr and nBreakChars parameters are omitted, WrapText searches for space, hyphen, or tab characters on which to

break the line and inserts a carriage return/line feed pair (Windows) or a linefeed (Linux) at the break points.

WrapText does not insert a break into an embedded quoted string (both single quotes and double quotes are supported).

For example, the following call wraps the text into two lines at the last space character:

```
WrapText('The rain in Spain falls mainly on the plain.', #13#10, ['.',' ',#9,'-'], 42);
```

TSysCharSet (see page 23019) bchars

```
bchars << '.' << '-' << ' ';
WrapText("The rain in Spain falls mainly on the plain.", "\r\n", bchars, 42);
```

The result:

```
The rain in Spain falls mainly on the
plain.
```

159.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TFilenameCaseMatch (see page 22998)	TFilenameCaseMatch indicates how close a match was found to a specified file name.
	TFloatFormat (see page 22999)	TFloatFormat is an enumerated list of formatting codes for float functions.
	TFloatValue (see page 23001)	This is record SysUtils.TFloatValue.
	TLocaleOptions (see page 23003)	TLocaleOptions defines a choice of dependent and independent locale options.
	TMbcsByteType (see page 23004)	TMbcsByteType represents the use of a single byte in a string that uses a multi-byte character set (MBCS).
	TNameType (see page 23005)	This is record SysUtils.TNameType.

Records

	Name	Description
	Int64Rec (see page 22996)	Int64Rec declares a utility record to provide access to the bytes of an Int64 value.
	LongRec (see page 22997)	LongRec declares a utility record to provide access to the bytes of a LongWord value.
	TExceptionRecord (see page 22997)	TExceptionRecord holds information about an exception in an application.
	TFloatRec (see page 23000)	TFloatRec declares a utility record that provides access to the attributes of a floating point number.
	TFormatSettings (see page 23001)	TFormatSettings defines a thread-safe string formatting context.
	TLangRec (see page 23002)	TLangRec declares a record used to describe a Windows locale.
	TSearchRec (see page 23005)	TSearchRec defines file information searched for by FindFirst (see page 22884) or FindNext (see page 22887).
	TSysLocale (see page 23008)	TSysLocale represents locale information.
	TThreadInfo (see page 23009)	This is record SysUtils.TThreadInfo.
	TTimeStamp (see page 23009)	TTimeStamp represents time and date values.
	WordRec (see page 23010)	WordRec declares a utility record to store high and low order bytes of a variable.

159.4.1 SysUtils.Int64Rec Record

Int64Rec declares a utility record to provide access to the bytes of an Int64 value.

Pascal

```
Int64Rec = packed record
  case Integer of
    0: (Hi: Cardinal;
        Lo: Cardinal;);
    1: (Cardinals: array [0..1] of Cardinal);
        Words: array [0..3] of Word);
        Bytes: array [0..7] of Byte););
end;
```

C++

```
struct Int64Rec {
  unsigned Hi;
  unsigned Lo;
  array [0..1] of Cardinal) Cardinals;
  array [0..3] of Word) Words;
```

```
  array [0..7] of Byte) Bytes;
};
```

File

SysUtils

Description

The Int64Rec type declares a utility record that stores 8 contiguous (packed) bytes of data. Int64Rec is used primarily for typecasting, where Int64 data needs to be dissected.

The 8 bytes may be accessed individually via the `Bytes` array, or as an array of 4 `Words`, or an array of 2 `Cardinals`. And finally, the value may be accessed via the `Hi` and `Lo` order Cardinals.

159.4.2 SysUtils.LongRec Record

LongRec declares a utility record to provide access to the bytes of a LongWord value.

Pascal

```
LongRec = packed record
  case Integer of
    0: (Hi: Word;
        Lo: Word;);
    1: (Words: array [0..1] of Word);
        Bytes: array [0..3] of Byte););
end;
```

C++

```
struct LongRec {
  Word Hi;
  Word Lo;
  array [0..1] of Word) Words;
  array [0..3] of Byte) Bytes;
};
```

File

SysUtils

Description

The LongRec type declares a utility record that stores 4 contiguous (packed) bytes of data. LongRec is used primarily for typecasting, where LongWord data needs to be dissected.

The 4 bytes may be accessed individually via the `Bytes` array, or as an array of 2 `Words`, or the value may be accessed via the `Hi` and `Lo` order Words.

159.4.3 SysUtils.TExceptionRecord Record

TExceptionRecord holds information about an exception in an application.

Pascal

```
TExceptionRecord = record
  ExceptionCode: Cardinal;
  ExceptionFlags: Cardinal;
  ExceptionRecord: PExceptionRecord;
```

```
    ExceptionAddress: Pointer;
    NumberParameters: Cardinal;
    ExceptionInformation: array[0..14] of Cardinal;
  end;
```

C++

```
  struct TExceptionRecord {
    unsigned ExceptionCode;
    unsigned ExceptionFlags;
    PExceptionRecord ExceptionRecord;
    void * ExceptionAddress;
    unsigned NumberParameters;
    array[0..14] of Cardinal ExceptionInformation;
  };
```

File

SysUtils

Description

TExceptionRecord holds information about an exception in an application. It is furnished by the Delphi exception handling process.

The record comprises the following elements :

Element	Meaning
`ExceptionCode`	The exception code, such as EXCEPTION_INT_OVERFLOW.
`ExceptionFlags`	Comprises one flag : EXCEPTION_NONCONTINUABLE_EXCEPTION. If 0, then the exception is allowed to continue.
`ExceptionRecord`	Pointer to subsequent exception records, if any.
`ExceptionAddress`	Address where the exception occured.
`NumberParameters`	Number of elements in the `ExceptionInformation` array.
`ExceptionInformation`	An array of numbers that describe the exception.

159.4.4 SysUtils.TFilenameCaseMatch Enumeration

TFilenameCaseMatch indicates how close a match was found to a specified file name.

Pascal

```
  TFilenameCaseMatch = (
    mkNone,
    mkExactMatch,
    mkSingleMatch,
    mkAmbiguous
  );
```

C++

```
  enum TFilenameCaseMatch {
    mkNone,
    mkExactMatch,
    mkSingleMatch,
    mkAmbiguous
  };
```

File

SysUtils

Description

These are the possible values of the TFilenameCaseMatch type:

Value	Meaning
mkNone	No actual file matches the file name.
mkExactMatch	A single file matches the file name.
mkSingleMatch	No file matches the file name, but a single match can be found by ignoring differences in case.
mkAmbiguous	No file matches the file name, but two or more matches can be found by ignoring differences in case.

159.4.5 SysUtils.TFloatFormat Enumeration

TFloatFormat is an enumerated list of formatting codes for float functions.

Pascal

```
TFloatFormat = (
  ffGeneral,
  ffExponent,
  ffFixed,
  ffNumber,
  ffCurrency
);
```

C++

```
enum TFloatFormat {
  ffGeneral,
  ffExponent,
  ffFixed,
  ffNumber,
  ffCurrency
};
```

File

SysUtils

Description

TFloatFormat defines an enumerated list of formatting codes for the float functions.

Format (see page 22897)	Defines
ffGeneral	General number format. The value is converted to the shortest possible decimal string using scientific format. Trailing zeros are removed from the resulting string.
ffExponent	Scientific format. The value is converted to a string of the form "-d.ddd...E+dddd". The resulting string starts with a minus sign if the number is negative, and one digit always precedes the decimal point. The total number of digits before the exponent in the resulting string (including the one before the decimal point) is given by the Precision parameter. The "E" exponent character in the resulting string is always followed by a plus or minus sign and up to four digits. The Digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).

ffFixed	Fixed point format. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative, and at least one digit always precedes the decimal point. The number of digits after the decimal point is given by the Digits parameter--it must be between 0 and 18. If the number of digits to the left of the decimal point is greater than the specified precision, the resulting value will use scientific format. The resulting values is padded with zeros when the Digits parameter is higher than the number of digits dictated by precision. For example, given ffFixed with Precision set to 5 and digits set to 3, fomatting 345.6789 gives the string '345.680', padding with zeros after the precision is met.
ffNumber	Number format. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The ffNumber format corresponds to the ffFixed format, except that the resulting string contains thousandth separators. The resulting values is padded with zeros when the Digits parameter is higher than the number of digits dictated by precision. For example, given ffNumber with Precision set to 5 and digits set to 3, fomatting 345.6789 gives the string '345.680', padding with zeros after the precision is met.
ffCurrency	Currency format. The value is converted to a string that represents a currency amount. The conversion is controlled by the CurrencyString (see page 23023), CurrencyFormat (see page 23022), NegCurrFormat (see page 23029), ThousandSeparator (see page 23032), and DecimalSeparator (see page 23023) global variables, all of which are initialized from the Currency Format (see page 22897) in the International section of the Windows Control Panel. The number of digits after the decimal point is given by the Digits parameter--it must be between 0 and 18. The resulting values is padded with zeros when the Digits parameter is higher than the number of digits dictated by precision. For example, given ffCurrency with Precision set to 5 and digits set to 3, fomatting 345.6789 gives the string '345.680', padding with zeros after the precision is met.

See Also

CurrencyAndDatetimeFormatting

FloatToDecimal

FloatToStr

FloatToStrF

FloatToText

FloatToTextFmt

159.4.6 SysUtils.TFloatRec Record

TFloatRec declares a utility record that provides access to the attributes of a floating point number.

Pascal

```
TFloatRec = packed record
  Exponent: Smallint;
  Negative: Boolean;
  Digits: array[0..20] of Char;
end;
```

C++

```
struct TFloatRec {
  Smallint Exponent;
  Boolean Negative;
  array[0..20] of Char Digits;
};
```

File

SysUtils

Description

The TFloatRec type declares a record that holds the attributes of a floating point number. The record makes the attributes accessible for easy formatting or specialised numerical processing.

`Exponent` holds the floating point number exponent value.

`Negative` is set true if the floating point number has a negative value.

`Digits` is a 21 element array of numeric characters representing the floating point number mantissa value : its significant digits.

TFloatRec is used by the FloatToDecimal (see page 22890) function to hold the results of the conversion of a floating point value to its decimal representation.

See Also

FloatToDecimal (see page 22890)

159.4.7 SysUtils.TFloatValue Enumeration

Pascal

```
TFloatValue = (
  fvExtended,
  fvCurrency
);
```

C++

```
enum TFloatValue {
  fvExtended,
  fvCurrency
};
```

File

SysUtils

Description

This is record SysUtils.TFloatValue.

159.4.8 SysUtils.TFormatSettings Record

TFormatSettings defines a thread-safe string formatting context.

Pascal

```
TFormatSettings = record
  CurrencyFormat: Byte;
  NegCurrFormat: Byte;
  ThousandSeparator: Char;
  DecimalSeparator: Char;
  CurrencyDecimals: Byte;
  DateSeparator: Char;
  TimeSeparator: Char;
  ListSeparator: Char;
  CurrencyString: string;
  ShortDateFormat: string;
  LongDateFormat: string;
```

```
    TimeAMString: string;
    TimePMString: string;
    ShortTimeFormat: string;
    LongTimeFormat: string;
    ShortMonthNames: array[1..12] of string;
    LongMonthNames: array[1..12] of string;
    ShortDayNames: array[1..7] of string;
    LongDayNames: array[1..7] of string;
    TwoDigitYearCenturyWindow: Word;
  end;
```

C++

```
  struct TFormatSettings {
    Byte CurrencyFormat;
    Byte NegCurrFormat;
    Char ThousandSeparator;
    Char DecimalSeparator;
    Byte CurrencyDecimals;
    Char DateSeparator;
    Char TimeSeparator;
    Char ListSeparator;
    AnsiString CurrencyString;
    AnsiString ShortDateFormat;
    AnsiString LongDateFormat;
    AnsiString TimeAMString;
    AnsiString TimePMString;
    AnsiString ShortTimeFormat;
    AnsiString LongTimeFormat;
    array[1..12] of string ShortMonthNames;
    array[1..12] of string LongMonthNames;
    array[1..7] of string ShortDayNames;
    array[1..7] of string LongDayNames;
    Word TwoDigitYearCenturyWindow;
  };
```

File

SysUtils

Description

TFormatSettings defines a data structure containing locale information used by string formatting routine. Each member of TFormatSettings is equivalent to the global variable with the same name. A variable of type TFormatSettings defines a thread-safe context that formatting functions can use in place of the default global context, which is not thread-safe.

To create and use the thread-safe environment defined by TFormatSettings, follow these steps:

Define a variable of type TFormatSettings

Call GetLocaleFormatSettings (see page 22913) to populate the TFormatSettings variable with locale information.

Pass the TFormatSettings variable as the last parameter of the string formatting routine.

Each routine that accepts a TFormatSettings parameter is thread-safe, and is an overload of an equivalent function that refers to the global formatting variables.

159.4.9 SysUtils.TLangRec Record

TLangRec declares a record used to describe a Windows locale.

Pascal

```
TLangRec = packed record
  FName: string;
  FLCID: LCID;
  FExt: string;
end;
```

C++

```
struct TLangRec {
  AnsiString FName;
  LCID FLCID;
  AnsiString FExt;
};
```

File

SysUtils

Description

The TLangRec type declares a record that holds the attributes of a Windows language. The record is principally used internally by TLanguages (see page 22762).

`FName` holds the Windows locale name.

`FLCID` holds the local id for the Windows locale.

`FExt` holds the recommended filename extension for the Windows locale.

See Also

TLanguages (see page 22762)

159.4.10 SysUtils.TLocaleOptions Enumeration

TLocaleOptions defines a choice of dependent and independent locale options.

Pascal

```
TLocaleOptions = (
  loInvariantLocale,
  loUserLocale
);
```

C++

```
enum TLocaleOptions {
  loInvariantLocale,
  loUserLocale
};
```

File

SysUtils

Description

The TLocaleOptions type is an enumeration of two choices for dealing with the operating locale of an application. This Microsoft Windows specific type is used in a number of string handling routines, where the locale language may have a bearing on the routine.

Format (see page 22897)	Defines
loInvariantLocale	Use this option where a consistent inter-application operation is desired, inedependent of the actual locale.
loUserLocale	Use this option where the user locale is taken into account.

See Also

CompareStr (see page 22813)

CompareText (see page 22814)

LowerCase (see page 22928)

SameStr (see page 22939)

SameText (see page 22940)

UpperCase (see page 22988)

159.4.11 SysUtils.TMbcsByteType Enumeration

TMbcsByteType represents the use of a single byte in a string that uses a multi-byte character set (MBCS).

Pascal

```
TMbcsByteType = (
  mbSingleByte,
  mbLeadByte,
  mbTrailByte
);
```

C++

```
enum TMbcsByteType {
  mbSingleByte,
  mbLeadByte,
  mbTrailByte
};
```

File

SysUtils

Description

TMbcsByteType represents the possible return values of the ByteType (see page 22807) and StrByteType (see page 22943) functions. Possible values are as follows:

Values	Meaning
mbSingleByte	The char is used to represent an entire character in the string. The value of Char can not be included in the LeadBytes (see page 23026) set.
mbLeadByte	The char is used to represent the first byte of a multi-byte character. The value of Char must be included in the LeadBytes (see page 23026) set.
mbTrailByte	The char is used to represent one of the trailing bytes in a multi-byte character. There are no a priori restrictions on the value of char.

See Also

ByteType

LeadBytes

StrByteType

159.4.12 SysUtils.TNameType Enumeration

Pascal

```
TNameType = (
  ntContainsUnit,
  ntRequiresPackage,
  ntDcpBpiName
);
```

C++

```
enum TNameType {
  ntContainsUnit,
  ntRequiresPackage,
  ntDcpBpiName
};
```

File

SysUtils

Description

This is record SysUtils.TNameType.

159.4.13 SysUtils.TSearchRec Record

TSearchRec defines file information searched for by FindFirst (see page 22884) or FindNext (see page 22887).

Pascal

```
TSearchRec = record
  Time: Integer;
  Size: Int64;
  Attr: Integer;
  Name: TFileName;
  ExcludeAttr: Integer;
  FindHandle: THandle platform;
  FindData: TWin32FindData platform;
end;
```

C++

```
struct TSearchRec {
  int Time;
  Int64 Size;
  int Attr;
  TFileName Name;
  int ExcludeAttr;
  THandle platform FindHandle;
  TWin32FindData platform FindData;
```

```
};
```

File

SysUtils

Description

The TSearchRec type defines file information searched for by a FindFirst (see page 22884) or FindNext (see page 22887) function call. If a file is found, the fields of the TSearchRec type parameter are modified to specify the found file.

Attr represents the file attributes of the file. Test Attr against the following attribute constants or values to determine if a file attribute matches the file's properties.

Note: On Windows, attribute constants map directly to DOS file attributes.

Note: On Linux, attribute constants map to sets of file characteristics; use the Mode field to access the specific permission mode.

Constant	Value	Description	Linux Meaning
faReadOnly (see page 23044)	1	Read-only files	Current user does not have write access.
faHidden (see page 23043)	2	Hidden files	File name begins with ".".
faSysFile (see page 23045)	4	System (see page 22079) files	File is socket, symbolic link, device file, or FIFO.
faVolumeID (see page 23045)	8	Volume ID files	Not used.
faDirectory (see page 23042)	16	Directory files	Directory.
faArchive (see page 23042)	32	Archive files	Not used.
faSymLink (see page 23044)	64	Symbolic link	File is a symbolic link.
faAnyFile (see page 23041)	71	Any file	Any file.

The faReadOnly (see page 23044) constant has the same name as the enumerated value that is defined by the TFieldAttribute type. If both the SysUtils (see page 22709) and the Db units are used in your source files, you must disambiguate by specifying the unit to qualify the use of faReadOnly (see page 23044). That is, write SysUtils.faReadOnly (see page 23044) (Delphi) or SysUtils::faReadOnly (see page 23044) (C++).

To test for an attribute, combine the value of the Attr field with the attribute constant with the and operator. If the file has that attribute, the result will be greater than 0. For example, if the found file is a hidden file, the following expression will evaluate to true:

(SearchRec.Attr and faHidden (see page 23043)) <> 0.

(SearchRec.Attr & faHidden (see page 23043)) != 0.

Time contains the time stamp of the file. It can be converted to a TDateTime value using FileDateToDateTime (see page 22858).

Size contains the size of the file in bytes.

Name contains the base file name, including extension.

FindHandle is an internal handle used to track find state.

FindData (Windows only) contains additional information such as the file creation time, last access time, and both the long and short file names.

Mode (Linux only) is the file's permission mask.

C++ Examples:

```
/*
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory).
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TSearchRec sr;
  int iAttributes = 0;
  StringGrid1->RowCount = 1;
  iAttributes |= faReadOnly * CheckBox1->Checked;
  iAttributes |= faHidden * CheckBox2->Checked;
  iAttributes |= faSysFile * CheckBox3->Checked;
  iAttributes |= faVolumeID * CheckBox4->Checked;
  iAttributes |= faDirectory * CheckBox5->Checked;
  iAttributes |= faArchive * CheckBox6->Checked;
  iAttributes |= faAnyFile * CheckBox7->Checked;
  StringGrid1->RowCount = 0;
  if (FindFirst(Edit1->Text, iAttributes, sr) == 0)
  {
    do
    {
      if ((sr.Attr & iAttributes) == sr.Attr)
      {
        StringGrid1->RowCount = StringGrid1->RowCount + 1;
        StringGrid1->Cells[1][StringGrid1->RowCount-1] = sr.Name;
        StringGrid1->Cells[2][StringGrid1->RowCount-1] = IntToStr(sr.Size);
      }
    } while (FindNext(sr) == 0);
    FindClose(sr);
  }
}
```

Delphi Examples:

```
{
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory).
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sr: TSearchRec;
  FileAttrs: Integer;
```

```
begin
  StringGrid1.RowCount := 1;
  if CheckBox1.Checked then
    FileAttrs := faReadOnly
  else
    FileAttrs := 0;
  if CheckBox2.Checked then
    FileAttrs := FileAttrs + faHidden;
  if CheckBox3.Checked then
    FileAttrs := FileAttrs + faSysFile;
  if CheckBox4.Checked then
    FileAttrs := FileAttrs + faVolumeID;
  if CheckBox5.Checked then
    FileAttrs := FileAttrs + faDirectory;
  if CheckBox6.Checked then
    FileAttrs := FileAttrs + faArchive;
  if CheckBox7.Checked then
    FileAttrs := FileAttrs + faAnyFile;
  with StringGrid1 do
  begin
    RowCount := 0;
    if SysUtils.FindFirst(Edit1.Text, FileAttrs, sr) = 0 then
    begin
      repeat
        if (sr.Attr and FileAttrs) = sr.Attr then
        begin
        RowCount := RowCount + 1;
        Cells[1,RowCount-1] := sr.Name;
        Cells[2,RowCount-1] := IntToStr(sr.Size);
        end;
      until FindNext(sr) <> 0;
      FindClose(sr);
    end;
  end;
end;
```

159.4.14 SysUtils.TSysLocale Record

TSysLocale represents locale information.

Pascal

```
TSysLocale = packed record
  DefaultLCID: Integer;
  PriLangID: Integer;
  SubLangID: Integer;
  FarEast: Boolean;
  MiddleEast: Boolean;
end;
```

C++

```
struct TSysLocale {
  int DefaultLCID;
  int PriLangID;
  int SubLangID;
  Boolean FarEast;
  Boolean MiddleEast;
};
```

File

SysUtils

Description

The TSysLocale type represents a locale. The locale determines how dates and times are formatted, how items are alphabetically sorted, and how strings are compared.

The **DefaultLCID** field represents the locale identifier. This is a 32-bit value that specifies a default sort order and a default language identifier.

The **PirLangID** field specifies the primary type of the language identifier. This is one of the constants that identify a language group, such as **LANG_ENGLISH** or **LANG_FRENCH**.

The **SubLangID** field specifies the subtype of the language identifier. This is one of the constants that identify a language subgroup, such as **SUBLANG_ENGLISH_US** or **SUBLANG_FRENCH_CANADIAN**.

FarEast is true or nonzero if User32.dll supports DBCS; false or zero otherwise. In other words, **FarEast** is true or nonzero if the double-byte character-set (DBCS) version of User.exe is installed; false or zero otherwise.

MiddleEast is true if the system is enabled for Hebrew and Arabic languages; false otherwise.

See Also

TLocale

Languages

TLocaleCode

159.4.15 SysUtils.TThreadInfo Record

Pascal

```
TThreadInfo = record
  Next: PThreadInfo;
  ThreadID: Cardinal;
  Active: Integer;
  RecursionCount: Cardinal;
end;
```

C++

```
struct TThreadInfo {
  PThreadInfo Next;
  unsigned ThreadID;
  int Active;
  unsigned RecursionCount;
};
```

File

SysUtils

Description

This is record SysUtils.TThreadInfo.

159.4.16 SysUtils.TTimeStamp Record

TTimeStamp represents time and date values.

Pascal

```
TTimeStamp = record
  Time: Integer;
  Date: Integer;
end;
```

C++

```
struct TTimeStamp {
  int Time;
  int Date;
};
```

File

SysUtils

Description

Use TTimeStamp to represent date and time values when a great deal of accuracy is required for the time portion. When the time values do not need to be precise to the millisecond, use the more compact TDateTime representation. If additional precision is needed, use TSQLTimeStamp (but do NOT assign it to a Variant).

The Time field indicates the number of milliseconds that have elapsed since midnight.

The Date field indicates the number of calendar days since the start of the calendar (the number of days since 1/1/0001 plus one).

See Also

DateTimeToTimeStamp

TDateTime

TimeStampToDateTime

TSQLTimeStamp

159.4.17 SysUtils.WordRec Record

WordRec declares a utility record to store high and low order bytes of a variable.

Pascal

```
WordRec = packed record
  case Integer of
    0: (Hi: Byte;
        Lo: Byte;);
    1: (Bytes: array [0..1] of Byte););
end;
```

C++

```
struct WordRec {
  Byte Hi;
  Byte Lo;
  array [0..1] of Byte) Bytes;
};
```

File

SysUtils

Description

The WordRec type declares a utility record that stores the high and low order bytes of the specified variable as type Byte. WordRec is used primarily for typecasting.

159.5 Types

The following table lists types in this documentation.

Types

Name	Description
ExceptClass (see page 23012)	ExceptClass declares an Exception (see page 22750) class.
PByteArray (see page 23013)	PByteArray declares a pointer to an array of Bytes.
PDayTable (see page 23013)	PDayTable is a pointer to a TDayTable (see page 23015) type.
PExceptionRecord (see page 23014)	PExceptionRecord is a pointer to a TExceptionRecord (see page 22997) type.
PThreadInfo (see page 23014)	This is type SysUtils.PThreadInfo.
PWordArray (see page 23014)	PWordArray is a pointer to a TWordArray (see page 23020) type.
TByteArray (see page 23015)	TByteArray declares an array of Bytes.
TBytes (see page 23015)	TBytes declares an array of Bytes.
TDayTable (see page 23015)	TDayTable is used to hold the number of days of each month of a year.
TFileName (see page 23016)	TFileName is a string that represents a file name.
TIntegerSet (see page 23016)	TIntegerSet is used to treat an integer as a set of bits.
TMREWSync (see page 23017)	TMREWSync is a short name equivalent to TMultiReadExclusiveWriteSynchronizer (see page 22766).
TPackageInfoProc (see page 23017)	TPackageInfoProc is a call-back procedure invoked by GetPackageInfo (see page 22915).
TProcedure (see page 23018)	TProcedure declares a procedure.
TReplaceFlags (see page 23018)	TReplaceFlags indicates how to perform a search-and-replace operation
TSysCharSet (see page 23019)	TSysCharSet is a set containing special characters that can be used to parse a string. The elements of the set vary, depending on the type of parsing performed.
TTerminateProc (see page 23019)	TValidatePackageProc (see page 23019) is a call-back function invoked by LoadPackage (see page 22927).
TValidatePackageProc (see page 23019)	This is type SysUtils.TValidatePackageProc.
TWordArray (see page 23020)	TWordArray declares an array of Words.

159.5.1 SysUtils.ExceptClass Type

ExceptClass declares an Exception (see page 22750) class.

Pascal

```
ExceptClass = class of Exception;
```

C++

```
Exception ExceptClass;
```

File

SysUtils

Description

The ExceptClass type declares an Exception (see page 22750) class. This class encapsulates the fundamental properties and methods for all exceptions. Use Exception (see page 22750) as the base class for creating custom exceptions.

See Also

Exception (see page 22750)

159.5.2 SysUtils.PByteArray Type

PByteArray declares a pointer to an array of Bytes.

Pascal

```
PByteArray = ^TByteArray;
```

C++

```
^TByteArray PByteArray;
```

File

SysUtils

Description

The PByteArray type declares a pointer to a TByteArray (see page 23015) : a static array of Bytes. The array size is 32,768 (2**15) elements.

Note: Static arrays are declared with an immutable size.

See Also

TByteArray (see page 23015)

159.5.3 SysUtils.PDayTable Type

PDayTable is a pointer to a TDayTable (see page 23015) type.

Pascal

```
PDayTable = ^TDayTable;
```

C++

```
^TDayTable PDayTable;
```

File

SysUtils

Description

The TDayTable (see page 23015) type points to a TDayTable (see page 23015) variable. TDayTable (see page 23015) declares a static array of 12 Words. It is principally used by the MonthDays (see page 23054) constant to hold the number of days in a month for normal and leap years.

Note: Static arrays are declared with an immutable size.

See Also

TDayTable (see page 23015)

159.5.4 SysUtils.PExceptionRecord Type

PExceptionRecord is a pointer to a TExceptionRecord (see page 22997) type.

Pascal

```
PExceptionRecord = ^TExceptionRecord;
```

C++

```
^TExceptionRecord PExceptionRecord;
```

File

SysUtils

Description

The PExceptionRecord type points to a TExceptionRecord (see page 22997) variable. TExceptionRecord (see page 22997) holds information about an exception in an application.

See Also

TExceptionRecord (see page 22997)

159.5.5 SysUtils.PThreadInfo Type

Pascal

```
PThreadInfo = ^TThreadInfo;
```

C++

```
^TThreadInfo PThreadInfo;
```

File

SysUtils

Description

This is type SysUtils.PThreadInfo.

159.5.6 SysUtils.PWordArray Type

PWordArray is a pointer to a TWordArray (see page 23020) type.

Pascal

```
PWordArray = ^TWordArray;
```

C++

```
^TWordArray PWordArray;
```

File

SysUtils

Description

The PWordArray type points to a TWordArray (see page 23020) variable. The TWordArray (see page 23020) type declares a static array of Words. The array size is 16,384 (2**14) elements.

See Also

TWordArray (see page 23020)

159.5.7 SysUtils.TByteArray Type

TByteArray declares an array of Bytes.

Pascal

```
TByteArray = array[0..32767] of Byte;
```

C++

```
array[0..32767] of Byte TByteArray;
```

File

SysUtils

Description

The TByteArray type declares a static array of Bytes. The array size is 32,768 (2**15) elements.

Note: Static arrays are declared with an immutable size.

159.5.8 SysUtils.TBytes Type

TBytes declares an array of Bytes.

Pascal

```
TBytes = array of Byte;
```

C++

```
array of Byte TBytes;
```

File

SysUtils

Description

The TBytes type declares a dynamic array of Bytes.

Note: Dynamic arrays have no intrinsic size. SetLength is used to allocate storage for the defined array size desired. Dynamic arrays are always integer-indexed, starting at 0.

159.5.9 SysUtils.TDayTable Type

TDayTable is used to hold the number of days of each month of a year.

Pascal

```
TDayTable = array[1..12] of Word;
```

C++

```
array[1..12] of Word TDayTable;
```

File

SysUtils

Description

The TDayTable type declares a static array of 12 Words. It is principally used by the MonthDays (see page 23054) constant to hold the number of days in a month for normal and leap years.

Note: Static arrays are declared with an immutable size.

See Also

MonthDays (see page 23054)

159.5.10 SysUtils.TFileName Type

TFileName is a string that represents a file name.

Pascal

```
TFileName = type string;
```

C++

```
type string TFileName;
```

File

SysUtils

Description

Use TFileName to represent strings that are only used for file names.

159.5.11 SysUtils.TIntegerSet Type

TIntegerSet is used to treat an integer as a set of bits.

Pascal

```
TIntegerSet = set of 0..SizeOf(Integer) * 8 - 1;
```

C++

```
0..SizeOf(Integer) * 8 - 1 TIntegerSet;
```

File

SysUtils

Description

Type TIntegerSet = set of 0 .. Integer-bit-size-1;

Assuming that an integer is 4 bytes (=32 bits), the set will 0 .. 31. TIntegerSet is typically used to treat an integer as a set of bits. To do so, an integer variable value is assigned to a TIntegerSet variable using TIntegerSet casting.

159.5.12 SysUtils.TMREWSync Type

TMREWSync is a short name equivalent to TMultiReadExclusiveWriteSynchronizer (see page 22766).

Pascal

```
TMREWSync = TMultiReadExclusiveWriteSynchronizer;
```

C++

```
TMultiReadExclusiveWriteSynchronizer TMREWSync;
```

File

SysUtils

Description

The TMREWSync type is used instead of TMultiReadExclusiveWriteSynchronizer (see page 22766).

Note: TMultiReadExclusiveWriteSynchronizer (see page 22766) is a class that minimizes thread serialization to gain read access to a resource shared among threads while still providing complete exclusivity to callers needing write access to the shared resource.

159.5.13 SysUtils.TPackageInfoProc Type

TPackageInfoProc is a call-back procedure invoked by GetPackageInfo (see page 22915).

Pascal

```
TPackageInfoProc = procedure (const Name: string; NameType: TNameType; Flags: Byte; Param:
Pointer);
```

C++

```
(const Name: string; NameType: TNameType; Flags: Byte; Param: Pointer) ( TPackageInfoProc)();
```

File

SysUtils

Description

The TPackageInfoProc type declares the procedure that is invoked by GetPackageInfo (see page 22915) to retrieve package information. It is called for every unit in the package, and every package required by that package.

The following table lists the information provided in these call back invocations of your TPackageInfoProc variable.

Parameter name	Parameter value
Name	Unit or required package name
NameType	Indicates if this call is for a contained unit, required package or DcpBpiName.
Flags	
Param	

See Also

GetPackageInfo (see page 22915)

NameType

159.5.14 SysUtils.TProcedure Type

TProcedure declares a procedure.

Pascal

```
TProcedure = procedure;
```

C++

```
( TProcedure)();
```

File

SysUtils

Description

Use the TProcedure type as a generic Procedure pointer.

159.5.15 SysUtils.TReplaceFlags Type

TReplaceFlags indicates how to perform a search-and-replace operation

Pascal

```
TReplaceFlags = set of (rfReplaceAll, rfIgnoreCase);
```

C++

```
(rfReplaceAll, rfIgnoreCase) TReplaceFlags;
```

File

SysUtils

Description

TReplaceFlags is a set of flags that govern how to locate and replace a substring within a string. This set can contain the following values:

Value	Meaning
rfReplaceAll	Replace all occurrences. If this flag is not present, only the first occurrence of the target substring is replaced.
rfIgnoreCase	Match occurrences of the substring case-insensitively. If this flag is not present, only case-sensitive matches are considered.

See Also

StringReplace

159.5.16 SysUtils.TSysCharSet Type

TSysCharSet is a set containing special characters that can be used to parse a string. The elements of the set vary, depending on the type of parsing performed.

Pascal

```
TSysCharSet = set of Char;
```

C++

```
Char TSysCharSet;
```

File

SysUtils

Description

type TSysCharSet = set of Char;

typedef Set<char, 0, 255> TSysCharSet;

TSysCharSet is a set containing special characters that can be used to parse a string. The elements of the set vary, depending on the type of parsing performed.

159.5.17 SysUtils.TTerminateProc Type

TValidatePackageProc (see page 23019) is a call-back function invoked by LoadPackage (see page 22927).

Pascal

```
TTerminateProc = function : Boolean;
```

C++

```
: Boolean ( TTerminateProc)();
```

File

SysUtils

Description

TValidatePackageProc (see page 23019) is optionally invoked by LoadPackage (see page 22927) when a package is loaded. The package is not loaded if TValidatePackageProc (see page 23019) determines that it is not valid and hence returns false.

See Also

LoadPackage (see page 22927)

159.5.18 SysUtils.TValidatePackageProc Type

Pascal

```
TValidatePackageProc = function (Module: HMODULE): Boolean;
```

C++

```
(Module: HMODULE): Boolean ( TValidatePackageProc)();
```

File

SysUtils

Description

This is type SysUtils.TValidatePackageProc.

159.5.19 SysUtils.TWordArray Type

TWordArray declares an array of Words.

Pascal

```
TWordArray = array[0..16383] of Word;
```

C++

```
array[0..16383] of Word TWordArray;
```

File

SysUtils

Description

The TWordArray type declares a static array of Words. The array size is 16,384 (2**14) elements.

Note: Static arrays are declared with an immutable size.

159.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
CurrencyDecimals (see page 23022)	CurrencyDecimals is the number of digits to the right of the decimal point in a currency amount.
CurrencyFormat (see page 23022)	CurrencyFormat defines the currency symbol placement and separation used in floating-point–to–decimal conversions. Possible values are: 0 = $1 \| 1 = 1$ | 2 = $ 1 | 3 = 1 $
CurrencyString (see page 23023)	CurrencyString defines the currency symbol (or characters) used in floating-point to decimal conversions.
DateSeparator (see page 23023)	DateSeparator is the character used to separate the year, month, and day parts of a date value.
DecimalSeparator (see page 23023)	DecimalSeparator is the character used to separate the integer part from the fractional part of a number. This variable must have a value other than #0.
EmptyStr (see page 23024)	An empty string constant.
EmptyWideStr (see page 23024)	An empty string version of the WideString constant.
EraNames (see page 23024)	Defines the names of calendar eras for East Asian locales.
EraYearOffsets (see page 23025)	Controls (see page 5711) the DateTimeToStr (see page 22822) function (for East Asian locales).
FalseBoolStrs (see page 23025)	Lists strings that can represent the boolean value false.
GetDiskFreeSpaceEx (see page 23026)	This is variable SysUtils.GetDiskFreeSpaceEx.
HexDisplayPrefix (see page 23026)	HexDisplayPrefix contains the prefix to display on hexadecimal values - `$` for Pascal (Delphi) syntax, `0x` for C++ syntax.
LeadBytes (see page 23026)	Lists all values that can be the first byte of a double byte character on the local system.
ListSeparator (see page 23027)	The character used to separate items in a list.
LongDateFormat (see page 23027)	LongDateFormat is the format string used to convert a date value to a long string suitable for display but not for editing.
LongDayNames (see page 23027)	LongDayNames is the array of strings containing long day names. The dddd format specifier in a format string passed to FormatDateTime (see page 22902) causes a long day name to be substituted.
LongMonthNames (see page 23028)	LongMonthNames is the array of strings containing long month names. The mmmm format specifier in a format string passed to FormatDateTime (see page 22902) causes a long month name to be substituted.
LongTimeFormat (see page 23028)	LongTimeFormat is the format string used to convert a time value to a string with hours, minutes, and seconds.
NegCurrFormat (see page 23029)	NegCurrFormat defines the currency format used in floating-point to decimal conversions of negative numbers.
NullStr (see page 23029)	A pointer to an empty string.
NullWideStr (see page 23030)	A pointer to an empty wide string.
ShortDateFormat (see page 23030)	ShortDateFormat is the format string used to convert a date value to a short string suitable for editing.
ShortDayNames (see page 23031)	ShortDayNames is the array of strings containing short day names. The ddd format specifier in a format string passed to FormatDateTime (see page 22902) causes a short day name to be substituted.
ShortMonthNames (see page 23031)	ShortMonthNames is the array of strings containing short month names. The mmm format specifier in a format string passed to FormatDateTime (see page 22902) causes a short month name to be substituted.
ShortTimeFormat (see page 23031)	ShortTimeFormat is the format string used to convert a time value to a short string with only hours and minutes.
SysLocale (see page 23032)	SysLocale holds locale information.
ThousandSeparator (see page 23032)	ThousandSeparator is the character used to separate thousands in numbers with more than three digits to the left of the decimal separator.
TimeAMString (see page 23033)	TimeAMString is the suffix string used for time values between 00:00 and 11:59 in 12-hour clock format.

TimePMString (see page 23033)	TimePMString is the suffix string used for time values between 12:00 and 23:59 in 12-hour clock format.
TimeSeparator (see page 23033)	TimeSeparator is the character used to separate the hour, minute, and second parts of a time value.
TrueBoolStrs (see page 23034)	Lists strings that can represent the boolean value false.
TwoDigitYearCenturyWindow (see page 23034)	Determines what century is added to two-digit years when string dates are converted to numeric dates.
Win32BuildNumber (see page 23035)	Specifies which the build level of the Windows Operating System (see page 22079) operating.
Win32CSDVersion (see page 23035)	Specifies the Win32 latest installed service pack.
Win32MajorVersion (see page 23036)	Specifies which major Windows Operating System (see page 22079) version is operating.
Win32MinorVersion (see page 23036)	Specifies which minor Windows Operating System (see page 22079) version is operating.
Win32Platform (see page 23037)	Specifies the Windows Operating System (see page 22079) platform identifier.

159.6.1 SysUtils.CurrencyDecimals Variable

CurrencyDecimals is the number of digits to the right of the decimal point in a currency amount.

Pascal

```
CurrencyDecimals: Byte;
```

C++

```
Byte CurrencyDecimals;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_ICURRDIGITS.

159.6.2 SysUtils.CurrencyFormat Variable

CurrencyFormat defines the currency symbol placement and separation used in floating-point–to–decimal conversions. Possible values are: 0 = $1 | 1 = 1$ | 2 = $ 1 | 3 = 1 $

Pascal

```
CurrencyFormat: Byte;
```

C++

```
Byte CurrencyFormat;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_ICURRENCY.

159.6.3 SysUtils.CurrencyString Variable

CurrencyString defines the currency symbol (or characters) used in floating-point to decimal conversions.

Pascal

```
CurrencyString: string;
```

C++

```
AnsiString CurrencyString;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_SCURRENCY.

159.6.4 SysUtils.DateSeparator Variable

DateSeparator is the character used to separate the year, month, and day parts of a date value.

Pascal

```
DateSeparator: Char;
```

C++

```
Char DateSeparator;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_SDATE.

159.6.5 SysUtils.DecimalSeparator Variable

DecimalSeparator is the character used to separate the integer part from the fractional part of a number. This variable must have a value other than #0.

Pascal

```
DecimalSeparator: Char;
```

C++

```
Char DecimalSeparator;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_SDECIMAL.

159.6.6 SysUtils.EmptyStr Variable

An empty string constant.

Pascal

```
EmptyStr: string = '';
```

C++

```
AnsiString EmptyStr = '';
```

File

SysUtils

Description

Included for backward compatibility only.

159.6.7 SysUtils.EmptyWideStr Variable

An empty string version of the WideString constant.

Pascal

```
EmptyWideStr: WideString = '';
```

C++

```
BSTR EmptyWideStr = '';
```

File

SysUtils

Description

Included for backward compatibility only.

159.6.8 SysUtils.EraNames Variable

Defines the names of calendar eras for East Asian locales.

Pascal

```
EraNames: array [1..MaxEraCount] of string;
```

C++

```
array [1..MaxEraCount] of string EraNames;
```

File

SysUtils

Description

EraNames defines an array of 7 era names for Japanese and Taiwanese locales. It is principally used for date formatting.

See Also

FormatDateTime (see page 22902)

DateTimeToString (see page 22823)

159.6.9 SysUtils.EraYearOffsets Variable

Controls (see page 5711) the DateTimeToStr (see page 22822) function (for East Asian locales).

Pascal

```
EraYearOffsets: array [1..MaxEraCount] of Integer;
```

C++

```
array [1..MaxEraCount] of Integer EraYearOffsets;
```

File

SysUtils

159.6.10 SysUtils.FalseBoolStrs Variable

Lists strings that can represent the boolean value false.

Pascal

```
FalseBoolStrs: array of String;
```

C++

```
array of String FalseBoolStrs;
```

File

SysUtils

Description

FalseBoolStrs is an array of strings that are used in conversions between Boolean values and strings.

When using the BoolToStr (see page 22804) function, false values are converted to the first string in the list.

When using the StrToBool (see page 22961) function, any string found in the list, regardless of case, is converted to false. If no string match is found, TrueBoolStrs (see page 23034) is checked, true being returned if a match is found. No matches result in the EConvertError (see page 22724) exception being thrown.

See Also

StrToBool (see page 22961)

TryStrToBool (see page 22982)

TrueBoolStrs (see page 23034)

BoolToStr (see page 22804)

159.6.11 SysUtils.GetDiskFreeSpaceEx Variable

Pascal

```
GetDiskFreeSpaceEx: function (Directory: PChar; var FreeAvailable, TotalSpace: TLargeInteger;
TotalFree: PLargeInteger): Bool stdcall = nil;
```

C++

```
function (Directory: PChar; var FreeAvailable, TotalSpace: TLargeInteger; TotalFree:
PLargeInteger): Bool stdcall GetDiskFreeSpaceEx = nil;
```

File

SysUtils

Description

This is variable SysUtils.GetDiskFreeSpaceEx.

159.6.12 SysUtils.HexDisplayPrefix Variable

HexDisplayPrefix contains the prefix to display on hexadecimal values - `$` for Pascal (Delphi) syntax, `0x` for C++ syntax.

Pascal

```
HexDisplayPrefix: string = '$';
```

C++

```
AnsiString HexDisplayPrefix = '$';
```

File

SysUtils

Description

This is for display only - this does not affect the string-to-integer conversion routines.

159.6.13 SysUtils.LeadBytes Variable

Lists all values that can be the first byte of a double byte character on the local system.

Pascal

```
LeadBytes: set of Char = [];
```

C++

```
set of Char LeadBytes = [];
```

File

SysUtils

Description

Use LeadBytes to test whether a value can represent the lead byte in a double byte character. If an individual value is in the

LeadBytes set, it is not necessarily used as the lead byte of a double byte character. It may also appear as the second byte of a double byte character. To determine whether the character is used as a lead byte, use the global ByteType (see page 22807) or StrByteType (see page 22943) function.

Note: LeadBytes is always empty when running on a system with a locale that does not use a multi byte character system (MBCS).

See Also

ByteType (see page 22807)

StrByteType (see page 22943)

159.6.14 SysUtils.ListSeparator Variable

The character used to separate items in a list.

Pascal

```
ListSeparator: Char;
```

C++

```
Char ListSeparator;
```

File

SysUtils

Description

On Windows, the default value is fetched from LOCALE_SLIST.

159.6.15 SysUtils.LongDateFormat Variable

LongDateFormat is the format string used to convert a date value to a long string suitable for display but not for editing.

Pascal

```
LongDateFormat: string;
```

C++

```
AnsiString LongDateFormat;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_SLONGDATE.

159.6.16 SysUtils.LongDayNames Variable

LongDayNames is the array of strings containing long day names. The dddd format specifier in a format string passed to FormatDateTime (see page 22902) causes a long day name to be substituted.

Pascal

```
LongDayNames: array[1..7] of string;
```

C++

```
array[1..7] of string LongDayNames;
```

File

SysUtils

Description

On Windows, the default values are fetched from the LOCALE_SDAYNAME system locale entries.

159.6.17 SysUtils.LongMonthNames Variable

LongMonthNames is the array of strings containing long month names. The mmmm format specifier in a format string passed to FormatDateTime (see page 22902) causes a long month name to be substituted.

Pascal

```
LongMonthNames: array[1..12] of string;
```

C++

```
array[1..12] of string LongMonthNames;
```

File

SysUtils

Description

On Windows, the default values are fetched from the LOCALE_SMONTHNAME system locale entries.

159.6.18 SysUtils.LongTimeFormat Variable

LongTimeFormat is the format string used to convert a time value to a string with hours, minutes, and seconds.

Pascal

```
LongTimeFormat: string;
```

C++

```
AnsiString LongTimeFormat;
```

File

SysUtils

Description

On Windows, the default value is computed from LOCALE_ITIME and LOCALE_ITLZERO.

159.6.19 SysUtils.NegCurrFormat Variable

NegCurrFormat defines the currency format used in floating-point to decimal conversions of negative numbers.

Pascal

```
NegCurrFormat: Byte;
```

C++

```
Byte NegCurrFormat;
```

File

SysUtils

Description

Possible values are:

Value	Format (see page 22897)
0	($1)
1	-$1
2	$-1
3	$1-
4	(1$)
5	-1$
6	1-$
7	1$-
8	-1 $
9	-$ 1
10	1 $-
11	$ 1-
12	$ -1
13	1- $
14	($ 1)
15	(1 $)

On Windows, the initial value is fetched from LOCALE_INEGCURR.

159.6.20 SysUtils.NullStr Variable

A pointer to an empty string.

Pascal

```
NullStr: PString = @EmptyStr;
```

C++

```
PString NullStr = @EmptyStr;
```

File

SysUtils

Description

NullStr defines a pointer to the empty string EmptyStr (see page 23024).

See Also

EmptyStr (see page 23024)

NullWideStr (see page 23030)

PString (see page 22603)

159.6.21 SysUtils.NullWideStr Variable

A pointer to an empty wide string.

Pascal

```
NullWideStr: PWideString = @EmptyWideStr;
```

C++

```
PWideString NullWideStr = @EmptyWideStr;
```

File

SysUtils

Description

NullWideStr defines a pointer to the empty wide string EmptyWideStr (see page 23024).

See Also

EmptyWideStr (see page 23024)

NullStr (see page 23029)

PWideString (see page 22608)

159.6.22 SysUtils.ShortDateFormat Variable

ShortDateFormat is the format string used to convert a date value to a short string suitable for editing.

Pascal

```
ShortDateFormat: string;
```

C++

```
AnsiString ShortDateFormat;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_SSHORTDATE.

159.6.23 SysUtils.ShortDayNames Variable

ShortDayNames is the array of strings containing short day names. The ddd format specifier in a format string passed to FormatDateTime (see page 22902) causes a short day name to be substituted.

Pascal

```
ShortDayNames: array[1..7] of string;
```

C++

```
array[1..7] of string ShortDayNames;
```

File

SysUtils

Description

On Windows, the default values are fetched from the LOCALE_SABBREVDAYNAME system locale entries.

159.6.24 SysUtils.ShortMonthNames Variable

ShortMonthNames is the array of strings containing short month names. The mmm format specifier in a format string passed to FormatDateTime (see page 22902) causes a short month name to be substituted.

Pascal

```
ShortMonthNames: array[1..12] of string;
```

C++

```
array[1..12] of string ShortMonthNames;
```

File

SysUtils

Description

On Windows, the default values are fetched from the LOCALE_SABBREVMONTHNAME system locale entries.

159.6.25 SysUtils.ShortTimeFormat Variable

ShortTimeFormat is the format string used to convert a time value to a short string with only hours and minutes.

Pascal

```
ShortTimeFormat: string;
```

C++

```
AnsiString ShortTimeFormat;
```

File

SysUtils

Description

On Windows, the default value is computed from LOCALE_ITIME and LOCALE_ITLZERO.

159.6.26 SysUtils.SysLocale Variable

SysLocale holds locale information.

Pascal

```
SysLocale: TSysLocale;
```

C++

```
TSysLocale SysLocale;
```

File

SysUtils

Description

The SysLocale holds the attributes of the current locale. The locale determines how dates and times are formatted, how items are alphabetically sorted, and how strings are compared.

The **DefaultLCID** field represents the locale identifier. This is a 32-bit value that specifies a default sort order and a default language identifier.

The **PirLangID** field specifies the primary type of the language identifier. This is one of the constants that identify a language group, such as **LANG_ENGLISH** or **LANG_FRENCH**.

The **SubLangID** field specifies the subtype of the language identifier. This is one of the constants that identify a language subgroup, such as **SUBLANG_ENGLISH_US** or **SUBLANG_FRENCH_CANADIAN**.

FarEast is true or nonzero if User32.dll supports DBCS; false or zero otherwise. In other words, **FarEast** is true or nonzero if the double-byte character-set (DBCS) version of User.exe is installed; false or zero otherwise.

MiddleEast is true if the system is enabled for Hebrew and Arabic languages; false otherwise.

See Also

GetTSysLocale

Languages (see page 22926)

159.6.27 SysUtils.ThousandSeparator Variable

ThousandSeparator is the character used to separate thousands in numbers with more than three digits to the left of the decimal separator.

Pascal

```
ThousandSeparator: Char;
```

C++

```
Char ThousandSeparator;
```

File

SysUtils

Description

A value of #0 indicates that no thousand separator character should be output, even if the format string specifies it.

On Windows, the initial value is fetched from LOCALE_STHOUSAND.

159.6.28 SysUtils.TimeAMString Variable

TimeAMString is the suffix string used for time values between 00:00 and 11:59 in 12-hour clock format.

Pascal

```
TimeAMString: string;
```

C++

```
AnsiString TimeAMString;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_S1159.

159.6.29 SysUtils.TimePMString Variable

TimePMString is the suffix string used for time values between 12:00 and 23:59 in 12-hour clock format.

Pascal

```
TimePMString: string;
```

C++

```
AnsiString TimePMString;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_S2359.

159.6.30 SysUtils.TimeSeparator Variable

TimeSeparator is the character used to separate the hour, minute, and second parts of a time value.

Pascal

```
TimeSeparator: Char;
```

C++

```
Char TimeSeparator;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_STIME.

159.6.31 SysUtils.TrueBoolStrs Variable

Lists strings that can represent the boolean value false.

Pascal

```
TrueBoolStrs: array of String;
```

C++

```
array of String TrueBoolStrs;
```

File

SysUtils

Description

TrueBoolStrs is an array of strings that are used in conversions between Boolean values and strings.

When using the BoolToStr (see page 22804) function, false values are converted to the first string in the list.

When using the StrToBool (see page 22961) function, any string found in the list, regardless of case, is converted to true. If no string match is found, FalseBoolStrs (see page 23025) is checked, false being returned if a match is found. No matches result in the EConvertError (see page 22724) exception being thrown.

See Also

StrToBool (see page 22961)

TryStrToBool (see page 22982)

FalseBoolStrs (see page 23025)

BoolToStr (see page 22804)

159.6.32 SysUtils.TwoDigitYearCenturyWindow Variable

Determines what century is added to two-digit years when string dates are converted to numeric dates.

Pascal

```
TwoDigitYearCenturyWindow: Word = 50;
```

C++

```
Word TwoDigitYearCenturyWindow = 50;
```

File

SysUtils

Description

The value of TwoDigitYearCenturyWindow, if it is nonzero, is subtracted from the current year to calculate the floating "pivot" prior to which two-digit years are interpreted as falling in the next century. The default value is 50. The table below shows some examples.

159.6.33 SysUtils.Win32BuildNumber Variable

Specifies which the build level of the Windows Operating System (see page 22079) operating.

Pascal

```
Win32BuildNumber: Integer = 0;
```

C++

```
int Win32BuildNumber = 0;
```

File

SysUtils

Description

Win32BuildNumber specifies the build number of the Operating System (see page 22079). The unique identifier for a Windows Operating System (see page 22079) is Win32MajorVersion.Win32MinorVersion.Win32BuildNumber

Note: Win32BuildNumber is available on Windows only.

See Also

Win32CSDVersion (see page 23035)

Win32MinorVersion (see page 23036)

Win32MajorVersion (see page 23036)

Win32Platform (see page 23037)

159.6.34 SysUtils.Win32CSDVersion Variable

Specifies the Win32 latest installed service pack.

Pascal

```
Win32CSDVersion: string = '';
```

C++

```
AnsiString Win32CSDVersion = '';
```

File

SysUtils

Description

Win32CSDVersion specifies the name of the most recent service pack.

Note: Win32CSDVersion is available on Windows only.

See Also

Win32BuildNumber (see page 23035)

Win32MajorVersion (see page 23036)

Win32MinorVersion (see page 23036)

Win32Platform (see page 23037)

159.6.35 SysUtils.Win32MajorVersion Variable

Specifies which major Windows Operating System (see page 22079) version is operating.

Pascal

```
Win32MajorVersion: Integer = 0;
```

C++

```
int Win32MajorVersion = 0;
```

File

SysUtils

Description

Win32MajorVersion specifies the Major version of the Operating System (see page 22079). The unique identifier for a Windows Operating System (see page 22079) is Win32MajorVersion.Win32MinorVersion.Win32BuildNumber (see page 23035)

Note: Win32MajorVersion is available on Windows only.

See Also

Win32BuildNumber (see page 23035)

Win32CSDVersion (see page 23035)

Win32MinorVersion (see page 23036)

Win32Platform (see page 23037)

159.6.36 SysUtils.Win32MinorVersion Variable

Specifies which minor Windows Operating System (see page 22079) version is operating.

Pascal

```
Win32MinorVersion: Integer = 0;
```

C++

```
int Win32MinorVersion = 0;
```

File

SysUtils

Description

Win32MinorVersion specifies the Minor version of the Operating System (see page 22079). The unique identifier for a Windows Operating System (see page 22079) is Win32MajorVersion.Win32MinorVersion.Win32BuildNumber (see page 23035)

Note: Win32MinorVersion is available on Windows only.

See Also

Win32BuildNumber (see page 23035)

Win32CSDVersion (see page 23035)

Win32MajorVersion (see page 23036)

Win32Platform (see page 23037)

159.6.37 SysUtils.Win32Platform Variable

Specifies the Windows Operating System (see page 22079) platform identifier.

Pascal

```
Win32Platform: Integer = 0;
```

C++

```
int Win32Platform = 0;
```

File

SysUtils

Description

Win32Platform is used to identify the type of Win32 platform the system is running. This will be one of the following values:

Value	Meaning
VER_PLATFORM_WIN32s = 0	System (see page 22079) is Win32s.
VER_PLATFORM_WIN32_WINDOWS = 1	System (see page 22079) is Windows 95.
VER_PLATFORM_WIN32_NT = 2	System (see page 22079) is Windows NT. This includes Windows XP and Vista.

Note: Win32Platform is available on Windows only.

See Also

Win32BuildNumber (see page 23035)

Win32CSDVersion (see page 23035)

Win32MajorVersion (see page 23036)

Win32MinorVersion (see page 23036)

159.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
DateDelta (see page 23039)	Specifies the difference between TDateTime and TTimeStamp (see page 23009) starting values.
DefaultFalseBoolStr (see page 23039)	Specifies the default string representation of the boolean false value.
DefaultTrueBoolStr (see page 23040)	Specifies the default string representation of the boolean true value.
DriveDelim (see page 23041)	Specifies the character used to separate the Windows drive from the path in a file name.
faAnyFile (see page 23041)	Specifies any file type.
faArchive (see page 23042)	Specifies only Windows archived file types.
faDirectory (see page 23042)	Specifies only directory 'file' types.
faHidden (see page 23043)	Specifies only hidden file types.
faReadOnly (see page 23044)	Specifies only read only file types.
faSymLink (see page 23044)	Specifies only symbolic link file types.
faSysFile (see page 23045)	Specifies only system file types.
faVolumeID (see page 23045)	Specifies Volume ID file types.
fmOpenRead (see page 23046)	Specifies a file should be opened for reading only.
fmOpenReadWrite (see page 23047)	Specifies a file should be opened for reading and writing.
fmOpenWrite (see page 23047)	Specifies a file should be opened for writing only.
fmShareCompat (see page 23048)	This is constant SysUtils.fmShareCompat.
fmShareDenyNone (see page 23048)	This is constant SysUtils.fmShareDenyNone.
fmShareDenyRead (see page 23049)	This is constant SysUtils.fmShareDenyRead.
fmShareDenyWrite (see page 23049)	This is constant SysUtils.fmShareDenyWrite.
fmShareExclusive (see page 23049)	This is constant SysUtils.fmShareExclusive.
HoursPerDay (see page 23050)	Specifies the number of hours in a day.
MaxCurrency (see page 23050)	Defines the largest value supported for Currency data types.
MaxDateTime (see page 23051)	Specifies the highest supported TDateTime value.
MaxEraCount (see page 23051)	This is constant SysUtils.MaxEraCount.
MinCurrency (see page 23051)	Defines the largest negative value supported for Currency data types.
MinDateTime (see page 23052)	Specifies the earliest supported TDateTime value.
MinsPerDay (see page 23052)	Specifies the number of minutes in a day.
MinsPerHour (see page 23053)	Specifies the number of minutes in an hour.
MonthDays (see page 23054)	An array specifying how many days are in a month.
MSecsPerDay (see page 23054)	Specifies the number of milliseconds in a day.
MSecsPerSec (see page 23055)	Specifies the number of milliseconds in a second.
PathDelim (see page 23055)	Specifies a path delimiter character.
PathSep (see page 23055)	Specifies the character used to separate a sequence of file or directory names.
pfBCB4Produced (see page 23056)	Indicates that a package was produced by via C++.
pfConsumerBCB (see page 23056)	Indicates that C++ can be a consumer of a package.
pfConsumerCompat (see page 23057)	This is constant SysUtils.pfConsumerCompat.
pfConsumerDelphi (see page 23057)	Indicates that Delphi can be a consumer of a package.
pfConsumerMask (see page 23058)	Indicates the comsumer of a package.
pfDelphi4Produced (see page 23058)	Indicates that a package was produced by Delphi 4 or later.
pfDesignOnly (see page 23059)	Indicates that a package was compiled with {$DESIGNONLY ON}.
pfExeModule (see page 23059)	Indicates that a module is an executable.
pfIgnoreDupUnits (see page 23060)	Indicates that a package was compiled with {$DUPLICATEENABLE ON}.
pfLibraryModule (see page 23060)	Indicates that a module is a library.

pfModuleTypeMask (see page 23060)	Indicates that a package is a module.
pfNeverBuild (see page 23061)	Indicates that a package was compiled with the -z flag.
pfPackageModule (see page 23061)	Indicates that a module is a package.
pfProducerMask (see page 23062)	Indicates the producer of a package.
pfProducerUndefined (see page 23062)	Indicates that a package was produced by an unknown developmemt environment.
pfRunOnly (see page 23063)	Indicates that a package was compiled with {$RUNONLY ON}.
pfV3Produced (see page 23063)	Indicates that a package was produced by Delphi 3 or later.
SecsPerDay (see page 23064)	Specifies the number of seconds in a day.
SecsPerMin (see page 23064)	Specifies the number of seconds in a minute.
SwitchChars (see page 23065)	Specifies the prefix character(s) for program command line arguments.
ufImplicitUnit (see page 23065)	Indicates that a unit was implicitly imported into a package.
ufMainUnit (see page 23066)	Indicates that a unit is the main one in a project.
ufOrgWeakUnit (see page 23066)	This is constant SysUtils.ufOrgWeakUnit.
ufPackageUnit (see page 23067)	Indicates that a unit is a package.
ufWeakPackageUnit (see page 23067)	This is constant SysUtils.ufWeakPackageUnit.
ufWeakUnit (see page 23068)	Indicates that a unit is a weak unit.
UnixDateDelta (see page 23068)	Specifies the difference between TDateTime and Lunix date/time origins.

159.7.1 SysUtils.DateDelta Constant

Specifies the difference between TDateTime and TTimeStamp (see page 23009) starting values.

Pascal

```
DateDelta = 693594;
```

C++

```
DateDelta = 693594;
```

File

SysUtils

Description

DateDelta is the number of days (693594) between January 1, 0001 (starting TTimeStamp (see page 23009) value), and December 31, 1899 (starting TDateTime value). This value is useful for converting date-time values between TTimeStamp (see page 23009) and TDateTime.

See Also

TDateTime (see page 22612)

TTimeStamp (see page 23009)

UnixDateDelta (see page 23068)

159.7.2 SysUtils.DefaultFalseBoolStr Constant

Specifies the default string representation of the boolean false value.

Pascal

```
DefaultFalseBoolStr = 'False';
```

C++

```
DefaultFalseBoolStr = 'False';
```

File

SysUtils

Description

DefaultFalseBoolStr is 'False'. It is used internally by the StrToBool (see page 22961) and BoolToStr (see page 22804) routines. They assign DefaultFalseBoolStr to the 1st element of the FalseBoolStrs (see page 23025) array, used by these routines as a default string representation of the boolean false value.

See Also

DefaultTrueBoolStr (see page 23040)

StrToBool (see page 22961)

FalseBoolStrs (see page 23025)

TrueBoolStrs (see page 23034)

BoolToStr (see page 22804)

159.7.3 SysUtils.DefaultTrueBoolStr Constant

Specifies the default string representation of the boolean true value.

Pascal

```
DefaultTrueBoolStr = 'True';
```

C++

```
DefaultTrueBoolStr = 'True';
```

File

SysUtils

Description

DefaultTrueBoolStr is 'True'. It is used internally by the StrToBool (see page 22961) and BoolToStr (see page 22804) routines. They assign DefaultTrueBoolStr to the 1st element of the TrueBoolStrs (see page 23034) array, used by these routines as a default string representation of the boolean true value.

See Also

DefaultFalseBoolStr (see page 23039)

StrToBool (see page 22961)

FalseBoolStrs (see page 23025)

TrueBoolStrs (see page 23034)

BoolToStr (see page 22804)

159.7.4 SysUtils.DriveDelim Constant

Specifies the character used to separate the Windows drive from the path in a file name.

Pascal

```
DriveDelim = ':';
```

C++

```
DriveDelim = ':';
```

File

SysUtils

Description

DriveDelim is the ':' character that follows the drive character ('A', 'B' and so on) in a full filename on a Windows Operating System (see page 22079). The value is a null character for Linux Operating Systems since they do not support drive designations.

See Also

PathDelim (see page 23055)

PathSep (see page 23055)

159.7.5 SysUtils.faAnyFile Constant

Specifies any file type.

Pascal

```
faAnyFile = $0000003F;
```

C++

```
faAnyFile = 0x0000003F;
```

File

SysUtils

Description

faAnyFile is principally used by FindFirst (see page 22884) to indicate that files of any type encountered sould be returned. Namely, all files.

See Also

faArchive (see page 23042)

faDirectory (see page 23042)

faHidden (see page 23043)

faReadOnly (see page 23044)

faSymLink (see page 23044)

faSysFile (see page 23045)

FindFirst (see page 22884)

TSearchRec (see page 23005)

159.7.6 SysUtils.faArchive Constant

Specifies only Windows archived file types.

Pascal

```
faArchive = $00000020 platform;
```

C++

```
faArchive = 0x00000020 platform;
```

File

SysUtils

Description

faArchive is principally used by FindFirst (see page 22884) to indicate that only archived files sould be returned. It is Windows specific : Linux does not support archiving.

See Also

faAnyFile (see page 23041)

faDirectory (see page 23042)

faHidden (see page 23043)

faReadOnly (see page 23044)

faSymLink (see page 23044)

faSysFile (see page 23045)

FindFirst (see page 22884)

TSearchRec (see page 23005)

159.7.7 SysUtils.faDirectory Constant

Specifies only directory 'file' types.

Pascal

```
faDirectory = $00000010;
```

C++

```
faDirectory = 0x00000010;
```

File

SysUtils

Description

faDirectory is principally used by FindFirst (see page 22884) to indicate that only directories sould be returned. Whilst FindFirst

(see page 22884) is generally used to search for files, it can also return directories within the search directory.

See Also

faAnyFile (see page 23041)

faArchive (see page 23042)

faHidden (see page 23043)

faReadOnly (see page 23044)

faSymLink (see page 23044)

faSysFile (see page 23045)

FindFirst (see page 22884)

TSearchRec (see page 23005)

159.7.8 SysUtils.faHidden Constant

Specifies only hidden file types.

Pascal

```
faHidden = $00000002 platform;
```

C++

```
faHidden = 0x00000002 platform;
```

File

SysUtils

Description

faHidden is principally used by FindFirst (see page 22884) to indicate that only hidden files sould be returned. On Linux systems, hidden files are those that start with a "." character.

See Also

faAnyFile (see page 23041)

faArchibe

faDirectory (see page 23042)

faReadOnly (see page 23044)

faSymLink (see page 23044)

faSysFile (see page 23045)

FindFirst (see page 22884)

TSearchRec (see page 23005)

159.7.9 SysUtils.faReadOnly Constant

Specifies only read only file types.

Pascal

```
faReadOnly = $00000001 platform;
```

C++

```
faReadOnly = 0x00000001 platform;
```

File

SysUtils

Description

faReadOnly is principally used by FindFirst (see page 22884) to indicate that only read only files sould be returned. On Linux systems, read only is not necessarily a system wide attribute - access rights are user and group dependent. Read only here means that the current application does read but not write to a file.

See Also

faAnyFile (see page 23041)

faArchibe

faDirectory (see page 23042)

faHidden (see page 23043)

faSymLink (see page 23044)

faSysFile (see page 23045)

FindFirst (see page 22884)

TSearchRec (see page 23005)

159.7.10 SysUtils.faSymLink Constant

Specifies only symbolic link file types.

Pascal

```
faSymLink = $00000040 platform;
```

C++

```
faSymLink = 0x00000040 platform;
```

File

SysUtils

Description

faSymLink is principally used by FindFirst (see page 22884) to indicate that only symbolic link files sould be returned. Symbolic links are a feature of Linux systems, with no Windows counterpart.

See Also

faAnyFile (see page 23041)

faArchibe

faDirectory (see page 23042)

faHidden (see page 23043)

faReadOnly (see page 23044)

faSysFile (see page 23045)

FindFirst (see page 22884)

TSearchRec (see page 23005)

159.7.11 SysUtils.faSysFile Constant

Specifies only system file types.

Pascal

```
faSysFile = $00000004 platform;
```

C++

```
faSysFile = 0x00000004 platform;
```

File

SysUtils

Description

faSysFile is principally used by FindFirst (see page 22884) to indicate that only system files sould be returned. On Linux systems, a system file is one that is treated as a socket, symbolic link, device file, or FIFO.

See Also

faAnyFile (see page 23041)

faArchibe

faDirectory (see page 23042)

faHidden (see page 23043)

faReadOnly (see page 23044)

faSymLink (see page 23044)

FindFirst (see page 22884)

TSearchRec (see page 23005)

159.7.12 SysUtils.faVolumeID Constant

Specifies Volume ID file types.

Pascal

```
faVolumeID = $00000008 platform deprecated;
```

C++

```
faVolumeID = 0x00000008 platform deprecated;
```

File

SysUtils

Description

faVolumeID is deprecated.

159.7.13 SysUtils.fmOpenRead Constant

Specifies a file should be opened for reading only.

Pascal

```
fmOpenRead = $0000;
```

C++

```
fmOpenRead = 0x0000;
```

File

SysUtils

Description

fmOpenRead is one of the values you can assign to FileMode. It is also one of the attributes assignable in the Create. The TFileStream is used to read and write files. When you assign the value fmOpenRead, subsequent calls to Reset open files for reading only.

Note: You must use this value when you plan to read a read only file - using fmOpenReadWrite (see page 23047) will cause the open to fail because it will attempt and fail to get write access.

See Also

fmOpenReadWrite (see page 23047)

fmOpenWrite (see page 23047)

faShareCompat

faShareDenyNone

faShareDenyRead

faShareDenyWrite

faShareExclusive

Reset (see page 22511)

FileMode (see page 22644)

Create (see page 2843)

159.7.14 SysUtils.fmOpenReadWrite Constant

Specifies a file should be opened for reading and writing.

Pascal

```
fmOpenReadWrite = $0002;
```

C++

```
fmOpenReadWrite = 0x0002;
```

File

SysUtils

Description

fmOpenReadWrite is one of the values you can assign to FileMode. It is also one of the attributes assignable in the Create. The TFileStream is used to read and write files. When you assign the value fmOpenReadWrite, subsequent calls to Reset open files for reading and writing.

Note: Do not assign this value when you plan to read a read only file - the open will fail because write access will be attempted but be denied.

See Also

fmOpenRead (see page 23046)

fmOpenWrite (see page 23047)

faShareCompat

faShareDenyNone

faShareDenyRead

faShareDenyWrite

faShareExclusive

Reset (see page 22511)

FileMode (see page 22644)

Create (see page 2843)

159.7.15 SysUtils.fmOpenWrite Constant

Specifies a file should be opened for writing only.

Pascal

```
fmOpenWrite = $0001;
```

C++

```
fmOpenWrite = 0x0001;
```

File

SysUtils

Description

fmOpenWrite is one of the values you can assign to FileMode. It is also one of the attributes assignable in the Create. The TFileStream is used to read and write files. When you assign the value fmOpenWrite, subsequent calls to Reset open files for writing only.

See Also

fmOpenRead (see page 23046)

fmOpenReadWrite (see page 23047)

faShareCompat

faShareDenyNone

faShareDenyRead

faShareDenyWrite

faShareExclusive

Reset (see page 22511)

FileMode (see page 22644)

Create (see page 2843)

159.7.16 SysUtils.fmShareCompat Constant

Pascal

```
fmShareCompat = $0000 platform;
```

C++

```
fmShareCompat = 0x0000 platform;
```

File

SysUtils

Description

This is constant SysUtils.fmShareCompat.

159.7.17 SysUtils.fmShareDenyNone Constant

Pascal

```
fmShareDenyNone = $0040;
```

C++

```
fmShareDenyNone = 0x0040;
```

File

SysUtils

Description

This is constant SysUtils.fmShareDenyNone.

159.7.18 SysUtils.fmShareDenyRead Constant

Pascal

```
fmShareDenyRead = $0030 platform;
```

C++

```
fmShareDenyRead = 0x0030 platform;
```

File

SysUtils

Description

This is constant SysUtils.fmShareDenyRead.

159.7.19 SysUtils.fmShareDenyWrite Constant

Pascal

```
fmShareDenyWrite = $0020;
```

C++

```
fmShareDenyWrite = 0x0020;
```

File

SysUtils

Description

This is constant SysUtils.fmShareDenyWrite.

159.7.20 SysUtils.fmShareExclusive Constant

Pascal

```
fmShareExclusive = $0010;
```

C++

```
fmShareExclusive = 0x0010;
```

File

SysUtils

Description

This is constant SysUtils.fmShareExclusive.

159.7.21 SysUtils.HoursPerDay Constant

Specifies the number of hours in a day.

Pascal

```
HoursPerDay = 24;
```

C++

```
HoursPerDay = 24;
```

File

SysUtils

Description

HoursPerDay is the number of hours (24) in a day. Using this constant rather than hard coding 24 makes for more readable code.

See Also

DateDelta (see page 23039)

MinsPerDay (see page 23052)

MinsPerHour (see page 23053)

MSecsPerDay (see page 23054)

MSecsPerSec (see page 23055)

SecsPerDay (see page 23064)

SecsPerMin (see page 23064)

159.7.22 SysUtils.MaxCurrency Constant

Defines the largest value supported for Currency data types.

Pascal

```
MaxCurrency: Currency = 922337203685477.5807;
```

C++

```
Currency MaxCurrency = 922337203685477.5807;
```

File

SysUtils

Description

MaxCurrency is the largest value that can be held by a Currency data variable on the Windows Opersting System (see page 22079).

Note: Linux supports a value 1.0 less than this maximum (922337203685476.5807).

See Also

MinCurrency (see page 23051)

159.7.23 SysUtils.MaxDateTime Constant

Specifies the highest supported TDateTime value.

Pascal

```
MaxDateTime: TDateTime = 2958465.99999;
```

C++

```
TDateTime MaxDateTime = 2958465.99999;
```

File

SysUtils

Description

MaxDateTime is the most future date : 12/31/9999 23:59:59.999 that the TDateTime type supports. The actual shown value of 2958465.99999 reflects the fact that TDateTime is actually a floating point number, rather than a structured type. The mantissa holds the number of days relative to 12/30/1899 00:00:00.000, the exponent holds a fraction of a day.

See Also

TDateTime (see page 22612)

MinDateTime (see page 23052)

159.7.24 SysUtils.MaxEraCount Constant

Pascal

```
MaxEraCount = 7;
```

C++

```
MaxEraCount = 7;
```

File

SysUtils

Description

This is constant SysUtils.MaxEraCount.

159.7.25 SysUtils.MinCurrency Constant

Defines the largest negative value supported for Currency data types.

Pascal

```
MinCurrency: Currency = -922337203685477.5807;
```

C++

```
Currency MinCurrency = -922337203685477.5807;
```

File

SysUtils

Description

MinCurrency is the largest negative value that can be held by a Currency data variable on the Windows Opersting System (see page 22079).

Note: Linux supports a value 1.0 less negative than this minimum (-922337203685476.5807).

See Also

MaxCurrency (see page 23050)

159.7.26 SysUtils.MinDateTime Constant

Specifies the earliest supported TDateTime value.

Pascal

```
MinDateTime: TDateTime = -657434.0;
```

C++

```
TDateTime MinDateTime = -657434.0;
```

File

SysUtils

Description

MinDateTime is the most historic date : 01/01/0100 00:00:00.000 that the TDateTime type supports. The actual shown value of -657434.0 reflects the fact that TDateTime is actually a floating point number, rather than a structured type. The mantissa holds the number of days relative to 12/30/1899 00:00:00.000, the exponent holds a fraction of a day.

Note: The fractional part of a TDateTime is always treated in a positive sense. It always means the elapsed part of a day - if the TDateTime value is negative, this refers only to a negative offset of the integral day value from 12/30/1899.

See Also

TDateTime (see page 22612)

MaxDateTime (see page 23051)

159.7.27 SysUtils.MinsPerDay Constant

Specifies the number of minutes in a day.

Pascal

```
MinsPerDay = HoursPerDay * MinsPerHour;
```

C++

```
MinsPerDay = HoursPerDay * MinsPerHour;
```

File

SysUtils

Description

MinsPerDay is the number of minutes (1440) in a day. It is the product of HoursPerDay (see page 23050) and MinsPerHour (see page 23053).

Using this constant rather than hard coding 1440 makes for more readable code.

See Also

DateDelta (see page 23039)

HoursPerDay (see page 23050)

MinsPerHour (see page 23053)

MSecsPerDay (see page 23054)

MSecsPerSec (see page 23055)

SecsPerDay (see page 23064)

SecsPerMinute

159.7.28 SysUtils.MinsPerHour Constant

Specifies the number of minutes in an hour.

Pascal

```
MinsPerHour = 60;
```

C++

```
MinsPerHour = 60;
```

File

SysUtils

Description

MinsPerHour is the number of minutes (60) in an hour. Using this constant rather than hard coding 60 makes for more readable code.

See Also

DateDelta (see page 23039)

HoursPerDay (see page 23050)

MinsPerHour

MSecsPerDay (see page 23054)

MSecsPerSec (see page 23055)

SecsPerDay (see page 23064)

SecsPerMin (see page 23064)

159.7.29 SysUtils.MonthDays Constant

An array specifying how many days are in a month.

Pascal

```
MonthDays: array [Boolean] of TDayTable = ((31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31),
(31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31));
```

C++

```
array [Boolean] of TDayTable MonthDays = ((31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31),
(31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31));
```

File

SysUtils

Description

The MonthDays array can be used to quickly find the number of days in a month:

```
MonthDays[IsLeapYear(Y), M]
```

159.7.30 SysUtils.MSecsPerDay Constant

Specifies the number of milliseconds in a day.

Pascal

```
MSecsPerDay = SecsPerDay * MSecsPerSec;
```

C++

```
MSecsPerDay = SecsPerDay * MSecsPerSec;
```

File

SysUtils

Description

MSecsPerDay is the number of milliseconds (86,400,000) in a day. It is the product of SecsPerDay (see page 23064) and MSecsPerSec (see page 23055).

Using this constant rather than hard coding 86400000 makes for more readable code.

See Also

DateDelta (see page 23039)

HoursPerDay (see page 23050)

MinsPerDay (see page 23052)

MinsPerHour (see page 23053)

MSecsPerSec (see page 23055)

SecsPerDay (see page 23064)

SecsPerMin (see page 23064)

159.7.31 SysUtils.MSecsPerSec Constant

Specifies the number of milliseconds in a second.

Pascal

```
MSecsPerSec = 1000;
```

C++

```
MSecsPerSec = 1000;
```

File

SysUtils

Description

MSecsPerSec is the number of milliseconds (1000) in a second.

Using this constant rather than hard coding 1000 makes for more readable code.

See Also

DateDelta (see page 23039)

HoursPerDay (see page 23050)

MinsPerDay (see page 23052)

MinsPerHour (see page 23053)

MSecsPerDay (see page 23054)

SecsPerMin (see page 23064)

SecsPerDay (see page 23064)

159.7.32 SysUtils.PathDelim Constant

Specifies a path delimiter character.

Pascal

```
PathDelim = '\';
```

C++

```
PathDelim = '\';
```

File

SysUtils

159.7.33 SysUtils.PathSep Constant

Specifies the character used to separate a sequence of file or directory names.

Pascal

```
PathSep = ';';
```

C++

```
PathSep = ';';
```

File

SysUtils

Description

PathSep is the ';' character for Windows, or ':' for Linux that is used to separate filenames in a sequence of file or directory names. FileSearch (see page 22872), for example, expects PathSep to be used to separate the values in the string of directory names.

See Also

DriveDelim (see page 23041)

PathSep

FileSearch (see page 22872)

159.7.34 SysUtils.pfBCB4Produced Constant

Indicates that a package was produced by via C++.

Pascal

```
pfBCB4Produced = $08000000;
```

C++

```
pfBCB4Produced = 0x08000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about a package. It returns a set of flags.

pfBCB4Produced is one of these flag values, indicating that a package was produced by C++. BCB stands for Borland (see page 1761) C++ Builder.

See Also

GetPackageInfo (see page 22915)

159.7.35 SysUtils.pfConsumerBCB Constant

Indicates that C++ can be a consumer of a package.

Pascal

```
pfConsumerBCB = $00200000;
```

C++

```
pfConsumerBCB = 0x00200000;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about a package. It returns a set of flags.

pfConsumerBCB is one of these flag values, indicating that C++ is a legitimate consumer, or user, of a package. The C++ consumer is the Borland (see page 1761) C++ Builder compiler, linking in the package.

A package will be marked with the pfConsumerBCB when the -JL flag has been set during compilation. This ensures that the extra support files (.lib, .bpi, .hpp)required by a C++ consumer are generated and included in the package.

See Also

GetPackageInfo (see page 22915)

159.7.36 SysUtils.pfConsumerCompat Constant

Pascal

```
pfConsumerCompat = $00000000;
```

C++

```
pfConsumerCompat = 0x00000000;
```

File

SysUtils

Description

This is constant SysUtils.pfConsumerCompat.

159.7.37 SysUtils.pfConsumerDelphi Constant

Indicates that Delphi can be a consumer of a package.

Pascal

```
pfConsumerDelphi = $00100000;
```

C++

```
pfConsumerDelphi = 0x00100000;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about a package. It returns a set of flags.

pfConsumerDelphi is one of these flag values, indicating that Delphi is a legitimate consumer, or user, of a package. The Delphi consumer is the Delphi compiler, linking in the package.

See Also

GetPackageInfo (see page 22915)

159.7.38 SysUtils.pfConsumerMask Constant

Indicates the comsumer of a package.

Pascal

```
pfConsumerMask = $00F00000;
```

C++

```
pfConsumerMask = 0x00F00000;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about a package. It returns a set of flags.

pfConsumerMask is a mask that you can use to detect the consumers can use a package. This mask covers these flags : pfConsumerBCB (see page 23056), pfConsumerCompat (see page 23057) and pfConsumerDelphi (see page 23057) flags.

Note: pfConsumerMask is not itself a returned flag - it is simply a mask.

See Also

GetPackageInfo (see page 22915)

pfConsumerBCB (see page 23056)

pfConsumerCompat (see page 23057)

pfConsumerDelphi (see page 23057)

159.7.39 SysUtils.pfDelphi4Produced Constant

Indicates that a package was produced by Delphi 4 or later.

Pascal

```
pfDelphi4Produced = $0C000000;
```

C++

```
pfDelphi4Produced = 0x0C000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about a package. It returns a set of flags.

pfDelphi4Produced is one of these flag values, indicating that the a package was produced by Delphi 4 or later. Packages were

introduced in Delphi 3.

See Also

GetPackageInfo (see page 22915)

159.7.40 SysUtils.pfDesignOnly Constant

Indicates that a package was compiled with {$DESIGNONLY ON}.

Pascal

```
pfDesignOnly = $00000002;
```

C++

```
pfDesignOnly = 0x00000002;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about a package. It returns a set of flags.

pfDesignOnly is one of these flag values, indicating that the package was compiled with the $DESIGNONLY compiler directive set ON. This means that the for package is intended for installation in the IDE.

See Also

GetPackageInfo (see page 22915)

159.7.41 SysUtils.pfExeModule Constant

Indicates that a module is an executable.

Pascal

```
pfExeModule = $00000000;
```

C++

```
pfExeModule = 0x00000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about a package. It returns a set of flags.

pfExeModule is one of these flag values, indicating that the module is an executable.

See Also

GetPackageInfo (see page 22915)

159.7.42 SysUtils.pfIgnoreDupUnits Constant

Indicates that a package was compiled with {$DUPLICATEENABLE ON}.

Pascal

```
pfIgnoreDupUnits = $00000008;
```

C++

```
pfIgnoreDupUnits = 0x00000008;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about a package. It returns a set of flags.

pfIgnoreDupUnits is one of these flag values, indicating that the package was compiled with the $DUPLICATEENABLE compiler directive set ON.

See Also

GetPackageInfo (see page 22915)

159.7.43 SysUtils.pfLibraryModule Constant

Indicates that a module is a library.

Pascal

```
pfLibraryModule = $80000000;
```

C++

```
pfLibraryModule = 0x80000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about a package. It returns a set of flags.

pfLibraryModule is one of these flag values, indicating that the module is a library (.dll file extension).

See Also

GetPackageInfo (see page 22915)

159.7.44 SysUtils.pfModuleTypeMask Constant

Indicates that a package is a module.

Pascal

```
pfModuleTypeMask = $C0000000;
```

C++

```
pfModuleTypeMask = 0xC0000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about a package. It returns a set of flags.

pfModuleTypeMask is a mask that you can use to detect whether the package is a module. This mask is the binary AND of the pfExeModule (see page 23059), pfPackageModule (see page 23061) and pfLibraryModule (see page 23060) flags.

Note: pfModuleTypeMask is not itself a returned flag - it is simply a mask.

See Also

GetPackageInfo (see page 22915)

159.7.45 SysUtils.pfNeverBuild Constant

Indicates that a package was compiled with the -z flag.

Pascal

```
pfNeverBuild = $00000001;
```

C++

```
pfNeverBuild = 0x00000001;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about a package. It returns a set of flags.

pfNeverBuild is one of these flag values, indicating that the package was compiled with the -z flag.

See Also

GetPackageInfo (see page 22915)

159.7.46 SysUtils.pfPackageModule Constant

Indicates that a module is a package.

Pascal

```
pfPackageModule = $40000000;
```

C++

```
pfPackageModule = 0x40000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about a package. It returns a set of flags.

pfPackageModule is one of these flag values, indicating that the module is a package (.bpl file extension).

See Also

GetPackageInfo (see page 22915)

159.7.47 SysUtils.pfProducerMask Constant

Indicates the producer of a package.

Pascal

```
pfProducerMask = $0C000000;
```

C++

```
pfProducerMask = 0x0C000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about a package. It returns a set of flags.

pfProducerMask is a mask that you can use to detect the producer of a package. This mask is the binary AND of the pfBCB4Produced (see page 23056), pfDelphi4Produced (see page 23058), pfProducerUndefined (see page 23062) and pfV3Produced (see page 23063) flags.

Note: pfProducerMask is not itself a returned flag - it is simply a mask.

See Also

GetPackageInfo (see page 22915)

pfBCB4Produced (see page 23056)

pfDelphi4Produced (see page 23058)

pfProducerUndefined (see page 23062)

pfV3Produced (see page 23063)

159.7.48 SysUtils.pfProducerUndefined Constant

Indicates that a package was produced by an unknown developmemt environment.

Pascal

```
pfProducerUndefined = $04000000;
```

C++

```
pfProducerUndefined = 0x04000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about a package. It returns a set of flags.

pfProducerUndefined is one of these flag values, indicating that the producer of a package was undefined.

See Also

GetPackageInfo (see page 22915)

159.7.49 SysUtils.pfRunOnly Constant

Indicates that a package was compiled with {$RUNONLY ON}.

Pascal

```
pfRunOnly = $00000004;
```

C++

```
pfRunOnly = 0x00000004;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about a package. It returns a set of flags.

pfRunOnly is one of these flag values, indicating that the package was compiled with the $RUNONLY compiler directive set ON. The {$RUNONLY ON} directive causes the package where it occurs to be compiled as runtime only. Packages compiled with {$RUNONLY ON} cannot be installed as design-time packages in the IDE.

See Also

GetPackageInfo (see page 22915)

159.7.50 SysUtils.pfV3Produced Constant

Indicates that a package was produced by Delphi 3 or later.

Pascal

```
pfV3Produced = $00000000;
```

C++

```
pfV3Produced = 0x00000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about a package. It returns a set of flags.

pfV3Produced is one of these flag values, indicating that the a package was produced by Delphi 3 or later. Packages were introduced in Delphi 3.

See Also

GetPackageInfo (see page 22915)

159.7.51 SysUtils.SecsPerDay Constant

Specifies the number of seconds in a day.

Pascal

```
SecsPerDay = MinsPerDay * SecsPerMin;
```

C++

```
SecsPerDay = MinsPerDay * SecsPerMin;
```

File

SysUtils

Description

SecsPerDay is the number of seconds (86,400) in a day. It is the product of MinsPerDay (see page 23052) and SecsPerMin (see page 23064).

Using this constant rather than hard coding 86400 makes for more readable code.

See Also

DateDelta (see page 23039)

HoursPerDay (see page 23050)

MinsPerDay (see page 23052)

MinsPerHour (see page 23053)

MSecsPerDay (see page 23054)

MSecsPerSec (see page 23055)

SecsPerMin (see page 23064)

159.7.52 SysUtils.SecsPerMin Constant

Specifies the number of seconds in a minute.

Pascal

```
SecsPerMin = 60;
```

C++

```
SecsPerMin = 60;
```

File

SysUtils

Description

SecsPerMin is the number of seconds (60) in a minute.

Using this constant rather than hard coding 60 makes for more readable code.

See Also

DateDelta (see page 23039)

HoursPerDay (see page 23050)

MinsPerDay (see page 23052)

MinsPerHour (see page 23053)

MSecsPerDay (see page 23054)

MSecsPerSec (see page 23055)

SecsPerDay (see page 23064)

159.7.53 SysUtils.SwitchChars Constant

Specifies the prefix character(s) for program command line arguments.

Pascal

```
SwitchChars = ['/','-'];
```

C++

```
SwitchChars = ['/','-'];
```

File

SysUtils

Description

When a program is started, an optional 'command line' string may be supplied. Such a string may specify program operational arguments, or 'switches', prefixed by a separator character. SwitchChars defines these separator characters as either '-' or '/' on a Windows system, and as '-' on Linux systems.

See Also

FindCmdLineSwitch (see page 22884)

159.7.54 SysUtils.ufImplicitUnit Constant

Indicates that a unit was implicitly imported into a package.

Pascal

```
ufImplicitUnit = $10;
```

C++

```
ufImplicitUnit = 0x10;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about the units within a package. TPackageInfoProc (see page 23017) is one of the parameters to that function : it is a callback procedure that is invoked for each unit in the package. It passes a set of flags associated with each unit.

If the ufImplicitUnit is set, then the unit was implicitly imported into a package. This means that the unit is not used in another package required by this package, but is used by one of the units this package contains.

See Also

GetPackageInfo (see page 22915)

TPackageInfoProc (see page 23017)

159.7.55 SysUtils.ufMainUnit Constant

Indicates that a unit is the main one in a project.

Pascal

```
ufMainUnit = $01;
```

C++

```
ufMainUnit = 0x01;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about the units within a package. TPackageInfoProc (see page 23017) is one of the parameters to that function : it is a callback procedure that is invoked for each unit in the package. It passes a set of flags associated with each unit.

If the ufMainUnit is set, then the unit is a Main unit.

See Also

GetPackageInfo (see page 22915)

TPackageInfoProc (see page 23017)

159.7.56 SysUtils.ufOrgWeakUnit Constant

Pascal

```
ufOrgWeakUnit = $08;
```

C++

```
ufOrgWeakUnit = 0x08;
```

File

SysUtils

Description

This is constant SysUtils.ufOrgWeakUnit.

159.7.57 SysUtils.ufPackageUnit Constant

Indicates that a unit is a package.

Pascal

```
ufPackageUnit = $02;
```

C++

```
ufPackageUnit = 0x02;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about the units within a package. TPackageInfoProc (see page 23017) is one of the parameters to that function : it is a callback procedure that is invoked for each unit in the package. It passes a set of flags associated with each unit.

If the ufPackageUnit is set, then the unit is a package unit.

See Also

GetPackageInfo (see page 22915)

TPackageInfoProc (see page 23017)

159.7.58 SysUtils.ufWeakPackageUnit Constant

Pascal

```
ufWeakPackageUnit = ufPackageUnit or ufWeakUnit;
```

C++

```
ufWeakPackageUnit = ufPackageUnit or ufWeakUnit;
```

File

SysUtils

Description

This is constant SysUtils.ufWeakPackageUnit.

159.7.59 SysUtils.ufWeakUnit Constant

Indicates that a unit is a weak unit.

Pascal

```
ufWeakUnit = $04;
```

C++

```
ufWeakUnit = 0x04;
```

File

SysUtils

Description

The GetPackageInfo (see page 22915) function returns information about the units within a package. TPackageInfoProc (see page 23017) is one of the parameters to that function : it is a callback procedure that is invoked for each unit in the package. It passes a set of flags associated with each unit.

If the ufWeakUnit is set, then the unit was compiled with the $WEAKPACKAGEUNIT compiler directive set.

See Also

GetPackageInfo (see page 22915)

TPackageInfoProc (see page 23017)

159.7.60 SysUtils.UnixDateDelta Constant

Specifies the difference between TDateTime and Lunix date/time origins.

Pascal

```
UnixDateDelta = 25569;
```

C++

```
UnixDateDelta = 25569;
```

File

SysUtils

Description

UnixDateDelta is the number of days (25569) between December 30, 1899 (starting TDateTime value), and Jamuary 1, 1970 (starting value for Linux date time values). This value is useful for converting date-time values between TDateTime and Linux file date time values.

See Also

TDateTime (see page 22612)

TTimeStamp (see page 23009)

DateDelta (see page 23039)

160 TabNotBk Namespace

This is namespace TabNotBk.

Topics

Name	Description
Classes (see page 23070)	The following table lists classes in this documentation.
Types (see page 23131)	The following table lists types in this documentation.
Constants (see page 23132)	The following table lists constants in this documentation.

Classes

	Name	Description
	TTabbedNotebook (see page 23070)	TTabbedNotebook contains multiple pages, each with its own set of controls.
	TTabPage (see page 23112)	TTabPage implements an individual tab page of a tabbed notebook control.

Constants

Name	Description
CM_TABFONTCHANGED (see page 23132)	This is constant TabNotBk.CM_TABFONTCHANGED.

Types

Name	Description
TPageChangeEvent (see page 23131)	TPageChangeEvent is the type for event handlers that respond when the current page changes in a TTabbedNotebook (see page 23070) control.

160.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TTabbedNotebook (see page 23070)	TTabbedNotebook contains multiple pages, each with its own set of controls.
	TTabPage (see page 23112)	TTabPage implements an individual tab page of a tabbed notebook control.

160.1.1 TTabbedNotebook Class

TTabbedNotebook contains multiple pages, each with its own set of controls.

Class Hierarchy

Pascal

```
TTabbedNotebook = class(TCustomTabControl);
```

C++

```
class TTabbedNotebook : public TCustomTabControl;
```

File

TabNotBk

Description

TTabbedNotebook is provided for backwards compatibility. New applications should use TPageControl instead.

160.1.1.1 TTabbedNotebook Members

The following tables list the members exposed by TTabbedNotebook.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnChanging (see page 3955)	Occurs immediately before a new tab is selected.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.

	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDrawTab (see page 3956)	Occurs when a tab is about to be drawn.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnGetImageIndex (see page 3956)	Occurs when a tab is about to display its associated image.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 3933)	Calculates the control's display area given a rectangle.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.

	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 23082)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanChange (see page 3933)	Change (see page 3934) permission event dispatcher.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	CanShowTab (see page 3934)	Selection permission event dispatcher.
	Change (see page 23083)	Pending tab change event dispatcher.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 23083)	Respond to user click.
	CMTabFontChanged (see page 23084)	This is CMTabFontChanged, a member of class TTabbedNotebook.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 23084)	Creates underlying screen object.
	CreateParams (see page 23084)	Sets up the window-creation parameters for the tab control.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 3935)	Creates the window for a tab control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.

	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
V	DrawTab (see page 3935)	Tab drawing event dispatcher.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
V	GetChildOwner (see page 23085)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
V	GetChildren (see page 23085)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetImageIndex (see page 3936)	Generates an OnGetImageIndex (see page 3956) event
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.

	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 23086)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 23086)	Form load event dispatcher.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 3937)	Event dispatcher for creation and destruction of child objects.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 23087)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.

	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetPageIndex (see page 23087)	This is SetPageIndex, a member of class TTabbedNotebook.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetTabIndex (see page 3938)	This is SetTabIndex, a member of class TCustomTabControl.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 23088)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateTabImages (see page 3939)	Updates the tab images.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
R	DisplayRect (see page 3941)	Specifies the bounding rectangle of a tab control client area.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	HotTrack (see page 3941)	Determines whether labels on the tab under the mouse are automatically highlighted.
	Images (see page 3941)	Specifies the images drawn in tabs.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	MultiLine (see page 3942)	Determines whether the tabs can appear on more than one row.
	MultiSelect (see page 3943)	Specifies whether multiple tabs can be selected.
	OwnerDraw (see page 3944)	Specifies whether the tab control handles its own painting.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	RaggedRight (see page 3944)	Specifies whether rows of tabs stretch to fill the width of the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	ScrollOpposite (see page 3944)	Determines how the rows of tabs are scrolled in a multi-line tab control.
	Style (see page 3945)	Specifies the style of the tab control.
	TabHeight (see page 3945)	Specifies the height, in pixels, of the tabs in the tab control.
	TabIndex (see page 3946)	Identifies the selected tab on a tab control.
	TabPosition (see page 3948)	Determines whether tabs appear at the top or bottom.
	Tabs (see page 3948)	Contains the list of text strings that label the tabs of the tab control.
	TabWidth (see page 3950)	Specifies the horizontal size, in pixels, of the tabs in the tab control.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 23081)	Constructs a tabbed notebook component.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 23082)	Destroys a tabbed notebook component.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHitTestInfoAt (see page 3936)	Returns information about the location of a point relative to the client area of the tab control.
	GetIndexForPage (see page 23085)	Returns the PageIndex (see page 23095) value of the specified page.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	IndexOfTabAt (see page 3936)	Indicates the index of the tab at a specified point.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	RowCount (see page 3938)	Sets TabIndex (see page 3946) property.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollTabs (see page 3938)	Scrolls the tabs that are visible when the tab control is not multi-line.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	TabFontChanged (see page 23088)	Invalidates the tabs when the TabFont (see page 23103) property changes.
	TabRect (see page 3939)	Returns the bounding rectangle for a specified tab.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 3940)	Gives access to the tab control's canvas.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
R	TopFont (see page 23104)	Returns the current value of font used on the tabs of the tabbed notebook.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnChange (see page 23106)	Occurs when the page of a tabbed notebook changes.
	OnClick (see page 23107)	Occurs when a tab of a tabbed notebook is selected.
	OnContextPopup (see page 23107)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnEnter (see page 23108)	Occurs when a control receives the input focus.
	OnExit (see page 23110)	Occurs when the input focus shifts away from one control to another.

Published Properties

	Name	Description
	ActivePage (see page 23089)	Determines which page displays in the tabbed notebook control.
	Align (see page 23089)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.

	Anchors (see page 23093)	Specifies how the control is anchored to its parent.
	Constraints (see page 23093)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Enabled (see page 23094)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 23094)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	PageIndex (see page 23095)	Determines which page displays in the tabbed notebook component.
	Pages (see page 23096)	Contains the strings that identify the individual pages of the tabbed notebook control.
	ParentShowHint (see page 23096)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 23097)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 23101)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabFont (see page 23103)	Determines the font used on the tabs of the tabbed notebook control.
	TabOrder (see page 23103)	Indicates the position of the control in its parent's tab order.
	TabsPerRow (see page 23104)	This is TabsPerRow, a member of class TTabbedNotebook.
	TabStop (see page 23104)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 23105)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

160.1.1.2 TTabbedNotebook.Create Constructor

Constructs a tabbed notebook component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TTabbedNotebook(TComponent * AOwner);
```

Description

Call Create TTabbedNotebook (see page 23070) indirectly, using the new keyword, to instantiate a tabbed notebook at runtime. Tabbed notebooks added to forms at design time are created automatically.

The AOwner parameter assigns a value to the notebook's Owner property.

See Also

Owner (see page 2807)

160.1.1.3 TTabbedNotebook.Destroy Destructor

Destroys a tabbed notebook component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTabbedNotebook();
```

Description

Do not call Destroy directly in an application, Instead, use Free method, which checks that the notebook is not nil before calling Destroy.

See Also

Free (see page 22300)

160.1.1.4 TTabbedNotebook Methods

The methods of the TTabbedNotebook class are listed here.

Protected Methods

	Name	Description
	AlignControls (see page 23082)	Aligns any controls for which the control is the parent within a specified area of the control.
	Change (see page 23083)	Pending tab change event dispatcher.
	Click (see page 23083)	Respond to user click.
	CMTabFontChanged (see page 23084)	This is CMTabFontChanged, a member of class TTabbedNotebook.
	CreateHandle (see page 23084)	Creates underlying screen object.
	CreateParams (see page 23084)	Sets up the window-creation parameters for the tab control.
	GetChildOwner (see page 23085)	Returns the owner of a child component being read from a stream.
	GetChildren (see page 23085)	Calls a specified method for each child of the control.
	KeyDown (see page 23086)	Respond to key press events.
	Loaded (see page 23086)	Form load event dispatcher.
	ReadState (see page 23087)	Prepares the control for having its properties assigned values from a stream.
	SetPageIndex (see page 23087)	This is SetPageIndex, a member of class TTabbedNotebook.
	ShowControl (see page 23088)	Ensures that a specified child control is visible.

Public Methods

	Name	Description
	GetIndexForPage (see page 23085)	Returns the PageIndex (see page 23095) value of the specified page.
	TabFontChanged (see page 23088)	Invalidates the tabs when the TabFont (see page 23103) property changes.

160.1.1.4.1 TTabbedNotebook.AlignControls Method

Aligns any controls for which the control is the parent within a specified area of the control.

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

Use AlignControls to align all controls within an area of a form, panel, group box, scroll box, or any container control. Specify the area in which to align the controls as the value of the Rect parameter.

AlignControls uses the Align property value for each child control to determine how to align it. The AControl parameter can be nil (Delphi) or NULL (C++). If you specify a control in AControl, that control takes precedence in alignment over other, similarly-aligned controls.

See Also

Align (see page 5798)

160.1.1.4.2 TTabbedNotebook.Change Method

Pending tab change event dispatcher.

Pascal

```
procedure Change; override;
```

C++

```
virtual __fastcall Change();
```

Description

Change is called automatically when the selected tab changes. Override Change to specify additional processing before the OnChange event handler executes, or to block the OnChange event.

See Also

OnChange (see page 3952)

160.1.1.4.3 TTabbedNotebook.Click Method

Respond to user click.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Click is called automatically when the user left-clicks the control. Component or application code can call Click to simulate a user mouse click. This is often done in menu actions and hotkey handlers.

As implemented in TControl, Click queries whether the OnClick event handler exists and is different from the OnExecute handler for the control's Action. If this is true, the OnClick event handler is called. If this is false, and the Action OnExecute event handler exists, the Action's Execute method is called. Override Click to provide additional behavior.

See Also

Action (see page 5797)

OnExecute (see page 2716)

Execute (see page 2712)

DblClick (see page 5747)

OnClick (see page 5851)

160.1.1.4.4 TTabbedNotebook.CMTabFontChanged Method

Pascal

```
procedure CMTabFontChanged(var Message: TMessage); message CM_TABFONTCHANGED;
```

C++

```
__fastcall CMTabFontChanged(TMessage Message);
```

Description

This is CMTabFontChanged, a member of class TTabbedNotebook.

160.1.1.4.5 TTabbedNotebook.CreateHandle Method

Creates underlying screen object.

Pascal

```
procedure CreateHandle; override;
```

C++

```
virtual __fastcall CreateHandle();
```

Description

CreateHandle creates the Microsoft Windows screen object that underlies the control. if it does not already exist. The new window handle becomes the value of the Handle property.

CreateHandle creates or locates the parent control's window before it creates a window for the control itself.

Note: If the Parent property is nil (Delphi) or NULL (C++), CreateHandle raises an exception.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

Handle (see page 6207)

Parent (see page 5832)

160.1.1.4.6 TTabbedNotebook.CreateParams Method

Sets up the window-creation parameters for the tab control.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

Tab controls call CreateParams internally when the window for the tab control must be generated. CreateParams overrides the inherited method to specify a tab control window with the current property settings.

See Also

CreateWnd (see page 3935)

160.1.1.4.7 TTabbedNotebook.GetChildOwner Method

Returns the owner of a child component being read from a stream.

Pascal

```
function GetChildOwner: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetChildOwner();
```

Description

GetChildOwner is used internally by the component streaming system. It is rarely necessary to call it directly.

In TComponent, GetChildOwner always returns nil (Delphi) or NULL (C++), indicating that the owner is the root component currently being read (usually a form or data module). The Owner of a component is responsible for destroying it.

See Also

Create (see page 2766)

Owner (see page 2807)

160.1.1.4.8 TTabbedNotebook.GetChildren Method

Calls a specified method for each child of the control.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the streaming system that loads and saves components. Applications seldom need to call this routine.

GetChildren executes the callback specified by the Proc parameter for every child control listed by the Controls property that lists the Root parameter as its Owner.

Override GetChildren to limit or augment which child controls are saved with the control. When overriding GetChildren, call the procedure passed as the Proc parameter for every child control that should be saved. The Root parameter indicates the component (usually a form) that owns the control.

See Also

Controls (see page 6197)

Owner (see page 2807)

160.1.1.4.9 TTabbedNotebook.GetIndexForPage Method

Returns the PageIndex (see page 23095) value of the specified page.

Pascal

```
function GetIndexForPage(const PageName: string): Integer;
```

C++

```
__fastcall int GetIndexForPage(const AnsiString PageName);
```

Description

The PageIndex (see page 23095) property value is determined by the page's position in the Pages (see page 23096) property array. Specify the name of the page as the value of the PageName parameter. The name you specify must be one of the strings in the Pages (see page 23096) property.

See Also

PageIndex (see page 23095)

Pages (see page 23096)

160.1.1.4.10 TTabbedNotebook.KeyDown Method

Respond to key press events.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-down message (WM_KEYDOWN) from Windows, its message handler calls the DoKeyDown method. If DoKeyDown determines that the control should, in fact, process the character, it calls KeyDown, passing the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

See Also

KeyUp (see page 6164)

OnKeyDown (see page 6225)

DoKeyDown (see page 6145)

DoKeyPress (see page 6146)

DoKeyUp (see page 6146)

160.1.1.4.11 TTabbedNotebook.Loaded Method

Form load event dispatcher.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

The protected Loaded method is called automatically when the form that contains the tab control is loaded into memory. It provides an opportunity for the tab control to perform any initializations after the control and all other components in the form are loaded into memory, but before the first paint operation.

In TCustomTabControl, Loaded ensures that the layout of the tabs accommodate the images specified by the Images property.

See Also

Images (see page 3941)

160.1.1.4.12 TTabbedNotebook.ReadState Method

Prepares the control for having its properties assigned values from a stream.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

Override ReadState to change the preparations the control makes as it readies itself to have its property values assigned from a stream. For example, a control might destroy temporary internal data structures or objects before new instances are loaded from the stream. Be sure to include a call to the inherited method when overriding ReadState.

As implemented in TWinControl, ReadState disables control alignment until all the child controls have been read from the stream, calls the inherited ReadState method, and re-enables the alignment of all the controls. The method then builds the tab order list of controls and synchronizes its visibility with that of its parent control.

See Also

Parent (see page 5832)

ReadState (see page 5778)

160.1.1.4.13 TTabbedNotebook.SetPageIndex Method

Pascal

```
procedure SetPageIndex(Value: Integer);
```

C++

```
__fastcall SetPageIndex(int Value);
```

Description

This is SetPageIndex, a member of class TTabbedNotebook.

160.1.1.4.14 TTabbedNotebook.ShowControl Method

Ensures that a specified child control is visible.

Pascal

```
procedure ShowControl(AControl: TControl); override;
```

C++

```
virtual __fastcall ShowControl(TControl * AControl);
```

Description

Call ShowControl to display a child control. Specify the control you want to ensure will be visible as the value of the AControl parameter.

As implemented in TWinControl, ShowControl simply calls the ShowControl method of the control's parent. Derived objects override ShowControl to perform whatever actions are necessary to allow the indicated child object to be visible. Possible actions include scrolling the particular child object into view, or changing to the appropriate notebook page.

See Also

Show (see page 5788)

Parent (see page 5832)

160.1.1.4.15 TTabbedNotebook.TabFontChanged Method

Invalidates the tabs when the TabFont (see page 23103) property changes.

Pascal

```
procedure TabFontChanged(Sender: TObject);
```

C++

```
__fastcall TabFontChanged(TObject * Sender);
```

Description

TabFontChanged is called automatically when the TabFont (see page 23103) property changes. It invalidates the notebook so that the tabs are repainted in the new font.

See Also

TabFont (see page 23103)

Invalidate (see page 6161)

160.1.1.5 TTabbedNotebook Properties

The properties of the TTabbedNotebook class are listed here.

Public Properties

	Name	Description
R	TopFont (see page 23104)	Returns the current value of font used on the tabs of the tabbed notebook.

Published Properties

	Name	Description
	ActivePage (see page 23089)	Determines which page displays in the tabbed notebook control.

	Align (see page 23089)	Determines how the control aligns within its container (parent control).
	Anchors (see page 23093)	Specifies how the control is anchored to its parent.
	Constraints (see page 23093)	Specifies the size constraints for the control.
	Enabled (see page 23094)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 23094)	Controls the attributes of text written on or in the control.
	PageIndex (see page 23095)	Determines which page displays in the tabbed notebook component.
	Pages (see page 23096)	Contains the strings that identify the individual pages of the tabbed notebook control.
	ParentShowHint (see page 23096)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 23097)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 23101)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabFont (see page 23103)	Determines the font used on the tabs of the tabbed notebook control.
	TabOrder (see page 23103)	Indicates the position of the control in its parent's tab order.
	TabsPerRow (see page 23104)	This is TabsPerRow, a member of class TTabbedNotebook.
	TabStop (see page 23104)	Determines if the user can tab to a control.
	Visible (see page 23105)	Determines whether the component appears onscreen.

160.1.1.5.1 TTabbedNotebook.ActivePage Property

Determines which page displays in the tabbed notebook control.

Pascal

```
property ActivePage: string;
```

C++

```
__property AnsiString ActivePage;
```

Description

The value of ActivePage must be one of the strings contained in the Pages (see page 23096) property.

Note: To indicate the page that appears in the tabbed notebook by index rather than text, use the PageIndex (see page 23095) property instead.

See Also

PageIndex (see page 23095)

Pages (see page 23096)

160.1.1.5.2 TTabbedNotebook.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always

equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
```

```
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
```

```
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
```

```
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

160.1.1.5.3 TTabbedNotebook.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

160.1.1.5.4 TTabbedNotebook.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

160.1.1.5.5 TTabbedNotebook.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

160.1.1.5.6 TTabbedNotebook.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name,

Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

160.1.1.5.7 TTabbedNotebook.PageIndex Property

Determines which page displays in the tabbed notebook component.

Pascal

```
property PageIndex: Integer;
```

C++

```
__property int PageIndex;
```

Description

Changing the PageIndex value changes the page in the control.

Each string in the Pages (see page 23096) property is automatically assigned a PageIndex value when the page is created. The first page receives a value of 0, the second has a value of 1, and so on. If you delete a string from the Pages (see page 23096) property, the PageIndex values are reassigned so that the values always begin with 0 and continue to increase without any gaps between values.

See Also

ActivePage (see page 23089)

Pages (see page 23096)

160.1.1.5.8 TTabbedNotebook.Pages Property

Contains the strings that identify the individual pages of the tabbed notebook control.

Pascal

```
property Pages: TStrings;
```

C++

```
__property TStrings Pages;
```

Description

Tabbed notebook controls create a separate page for each string in the Pages property. For example, if Pages contains three strings, First, Second, and Third, the control has three separate pages. Each string in Pages corresponds to a TTabPage (see page 23112) object whose Caption property is the same as the string.

You can change the active page in a tabbed notebook control with either the ActivePage (see page 23089) or PageIndex (see page 23095) property.

See Also

ActivePage (see page 23089)

PageIndex (see page 23095)

160.1.1.5.9 TTabbedNotebook.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

160.1.1.5.10 TTabbedNotebook.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
```

```
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Cut1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      const int num_items = 4;
      for (int index = 0; index < num_items; ++index)
      {
          TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
          PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
          NewItem->Caption = "Menu Item " + IntToStr(index);
          NewItem->Tag = index;
          NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
          TNotifyEvent();
      }
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
          {
              case 0:
              {
                  ShowMessage("first item clicked");
```

```
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
```

```
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
```

```
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

160.1.1.5.11 TTabbedNotebook.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
```

```
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

160.1.1.5.12 TTabbedNotebook.TabFont Property

Determines the font used on the tabs of the tabbed notebook control.

Pascal

```
property TabFont: TFont;
```

C++

```
__property TFont TabFont;
```

Description

Use TabFont to specify the font characteristics used to display the label on the tabs of the notebook.

Note: The text on the selected tab is boldfaced if the selected font for the TabFont property is not also boldfaced.

See Also

TopFont (see page 23104)

160.1.1.5.13 TTabbedNotebook.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the

control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

160.1.1.5.14 TTabbedNotebook.TabsPerRow Property

Pascal

```
property TabsPerRow: Integer;
```

C++

```
__property int TabsPerRow;
```

Description

This is TabsPerRow, a member of class TTabbedNotebook.

160.1.1.5.15 TTabbedNotebook.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop: Boolean;
```

C++

```
__property Boolean TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder (see page 6211)

160.1.1.5.16 TTabbedNotebook.TopFont Property

Returns the current value of font used on the tabs of the tabbed notebook.

Pascal

```
property TopFont: TFont;
```

C++

```
__property TFont TopFont;
```

Description

TopFont is a read-only property that reports the TabFont (see page 23103) value.

See Also

TabFont (see page 23103)

160.1.1.5.17 TTabbedNotebook.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

160.1.1.6 TTabbedNotebook Events

The events of the TTabbedNotebook class are listed here.

Published Events

	Name	Description
	OnChange (see page 23106)	Occurs when the page of a tabbed notebook changes.
	OnClick (see page 23107)	Occurs when a tab of a tabbed notebook is selected.
	OnContextPopup (see page 23107)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnEnter (see page 23108)	Occurs when a control receives the input focus.
	OnExit (see page 23110)	Occurs when the input focus shifts away from one control to another.

160.1.1.6.1 TTabbedNotebook.OnChange Event

Occurs when the page of a tabbed notebook changes.

Pascal

```
property OnChange: TPageChangeEvent;
```

C++

```
__property TPageChangeEvent OnChange;
```

Description

Write an OnPage event handler to respond to a page change in a tabbed notebook.

See Also

OnClick (see page 23107)

160.1.1.6.2 TTabbedNotebook.OnClick Event

Occurs when a tab of a tabbed notebook is selected.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Write an OnClick event handler to respond when the user selects a tab in the tabbed notebook. Immediately after the OnClick event occurs, the OnChange (see page 23106) event occurs.

See Also

OnChange (see page 23106)

160.1.1.6.3 TTabbedNotebook.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the

default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

160.1.1.6.4 TTabbedNotebook.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
```

```
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
```

```
      with myActiveControl as TEdit do
         Font.Assign(TFontDialog(Sender).Font)
    else if myActiveControl is TRichEdit then
      with myActiveControl as TRichEdit do
        SelAttributes.Assign(TFontDialog(Sender).Font)
    else
      Beep;
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

160.1.1.6.5 TTabbedNotebook.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
```

```
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

160.1.2 TTabPage Class

TTabPage implements an individual tab page of a tabbed notebook control.

Class Hierarchy

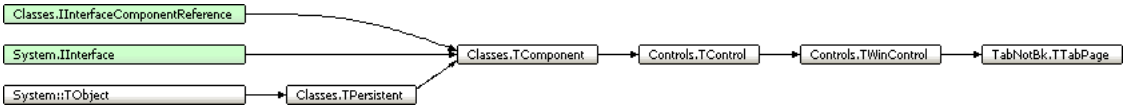

Pascal

```
TTabPage = class(TWinControl);
```

C++

```
class TTabPage : public TWinControl;
```

File

TabNotBk

Description

Each instance of this class holds the controls to be displayed when it is the active page of a tabbed notebook. TTabPage objects are associated with the strings in the corresponding tabbed notebook's Pages property by setting their Caption (see page 23124) property.

See Also

TTabbedNotebook (see page 23070)

TTabbedNotebook (see page 23070)

160.1.2.1 TTabPage Members

The following tables list the members exposed by TTabPage.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.

	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.

	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.

	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.

	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 23123)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.

	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.

	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 23122)	Creates an instance of TWinControl.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.

	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.

V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.

	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Caption (see page 23124)	Specifies a text string that identifies the control to the user.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Enabled (see page 23126)	Controls whether the control responds to mouse, keyboard, and timer events.
	Height (see page 23127)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	TabOrder (see page 23128)	Indicates the position of the control in its parent's tab order.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 23129)	Determines whether the component appears onscreen.
	Width (see page 23130)	Specifies the horizontal size of the control or form in pixels.

160.1.2.2 TTabPage.Create Constructor

Creates an instance of TWinControl.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TTabPage(TComponent * AOwner);
```

Description

Call Create to construct and initialize a new control and insert the newly-constructed control into its owner, as specified by the AOwner parameter. Create inserts the control in the owner by calling the owner's InsertComponent method.

Most controls override Create to initialize their unique properties. Objects that override the Create method must always call the inherited Create method first, and then proceed with the component-specific initialization. Specify the override directive when overriding the Create method.

If a component's Create method allocates resources or memory, override the Destroy method to free those resources.

See Also

Brush (see page 6195)

TabOrder (see page 6211)

CreateParented (see page 6123)

InsertComponent (see page 2780)

Create (see page 5731)

Owner (see page 2807)

160.1.2.3 TTabPage Methods

The methods of the TTabPage class are listed here.

Protected Methods

	Name	Description
	ReadState (see page 23123)	Prepares the control for having its properties assigned values from a stream.

160.1.2.3.1 TTabPage.ReadState Method

Prepares the control for having its properties assigned values from a stream.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

Override ReadState to change the preparations the control makes as it readies itself to have its property values assigned from a stream. For example, a control might destroy temporary internal data structures or objects before new instances are loaded from the stream. Be sure to include a call to the inherited method when overriding ReadState.

As implemented in TWinControl, ReadState disables control alignment until all the child controls have been read from the stream, calls the inherited ReadState method, and re-enables the alignment of all the controls. The method then builds the tab order list of controls and synchronizes its visibility with that of its parent control.

See Also

Parent (see page 5832)

ReadState (see page 5778)

160.1.2.4 TTabPage Properties

The properties of the TTabPage class are listed here.

Published Properties

	Name	Description
	Caption (see page 23124)	Specifies a text string that identifies the control to the user.
	Enabled (see page 23126)	Controls whether the control responds to mouse, keyboard, and timer events.
	Height (see page 23127)	Specifies the vertical size of the control in pixels.
	TabOrder (see page 23128)	Indicates the position of the control in its parent's tab order.
	Visible (see page 23129)	Determines whether the component appears onscreen.
	Width (see page 23130)	Specifies the horizontal size of the control or form in pixels.

160.1.2.4.1 TTabPage.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption: TCaption;
```

C++

```
__property TCaption Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long as the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
```

```
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this);
  myToolBar->Parent = this;
  TStringList *captions = new TStringList();
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions);
  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int I = myToolBar->ButtonCount - 1; I >= 0; I--)
    delete myToolBar->Buttons[I];
  delete myToolBar;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long as the caption of this label';
end;
```

```
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

160.1.2.4.2 TTabPage.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

160.1.2.4.3 TTabPage.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
HideSelection, Height example
This example needs only a blank form. Three Rich Edit
```

```
  controls are placed vertically on the form, each contain a
  line of text, and the second form has HideSelection set to
  false.  Try tabbing between the three Rich Edit controls and
  observe the effect on the selected text.
  }
  uses ComCtrls;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    I: Integer;
  begin
    for I := 0 to 2 do
      with TRichEdit.Create(Self) do
      begin
        Parent := Self;
        Top := Height * I;
        Text := 'This text will take the focus.';
        SelStart := 0;
        SelLength := Length(Text);
        if (I = 1) then
          HideSelection := False;
      end;
  end;
```

See Also

ClientHeight (see page 5808)

Width (see page 5849)

Using TDBListBox and TDBComboBox

160.1.2.4.4 TTabPage.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

160.1.2.4.5 TTabPage.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
```

```
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

160.1.2.4.6 TTabPage.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

See Also

ClientWidth (see page 5811)

Height (see page 5825)

Left (see page 5830)

160.2 Types

The following table lists types in this documentation.

Types

Name	Description
TPageChangeEvent (see page 23131)	TPageChangeEvent is the type for event handlers that respond when the current page changes in a TTabbedNotebook (see page 23070) control.

160.2.1 TabNotBk.TPageChangeEvent Type

TPageChangeEvent is the type for event handlers that respond when the current page changes in a TTabbedNotebook (see page 23070) control.

Pascal

```
TPageChangeEvent = procedure (Sender: TObject; NewTab: Integer; var AllowChange: Boolean) of object;
```

C++

```
(Sender: TObject; NewTab: Integer; var AllowChange: Boolean) ( TPageChangeEvent)();
```

File

TabNotBk

Description

Sender is the TTabbedNotebook (see page 23070) control.

NewTab is the tab that is about to become the selected or current tab.

AllowChange specifies whether the change to a new page is allowed. If the event handler sets AllowChange to false, a new page change doesn't occur.

160.3 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CM_TABFONTCHANGED (see page 23132)	This is constant TabNotBk.CM_TABFONTCHANGED.

160.3.1 TabNotBk.CM_TABFONTCHANGED Constant

Pascal

```
CM_TABFONTCHANGED = CM_BASE + 100;
```

C++

```
CM_TABFONTCHANGED = CM_BASE + 100;
```

File

TabNotBk

Description

This is constant TabNotBk.CM_TABFONTCHANGED.

161 Tabs Namespace

This is namespace Tabs.

Topics

Name	Description
Classes (see page 23134)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 23229)	The following table lists structs, records, enums in this documentation.
Types (see page 23232)	The following table lists types in this documentation.

Classes

	Name	Description
	TScroller (see page 23134)	TScroller is a horizontal scrolling control that provides basic horizontal scrolling capabilities.
	TTabList (see page 23149)	This is class Tabs.TTabList.
	TTabSet (see page 23159)	The TTabSet component presents horizontal tabs users can click to initiate actions.

Structs, Records, Enums

	Name	Description
	TEdgePart (see page 23229)	This is record Tabs.TEdgePart.
	TEdgeType (see page 23229)	This is record Tabs.TEdgeType.
	TScrollBtn (see page 23230)	This is record Tabs.TScrollBtn.
	TScrollOrientation (see page 23230)	This is record Tabs.TScrollOrientation.
	TTabSetTabStyle (see page 23231)	This is record Tabs.TTabSetTabStyle.

Types

Name	Description
TDrawTabEvent (see page 23232)	TDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 23159) control.
TMeasureTabEvent (see page 23232)	TMeasureTabEvent is the type for event handlers that indicate the size of the tabs in an owner-drawn TTabSet (see page 23159) control.
TTabChangeEvent (see page 23233)	TTabChangeEvent is the type for event handlers that respond when the current tab of a TTabSet (see page 23159) control changes.
TTabSetDrawTabEvent (see page 23233)	TTabSetDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 23159) control.
TTabStyle (see page 23234)	TTabStyle indicates whether a TTabSet (see page 23159) control is owner-drawn.

161.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TScroller (see page 23134)	TScroller is a horizontal scrolling control that provides basic horizontal scrolling capabilities.
	TTabList (see page 23149)	This is class Tabs.TTabList.
	TTabSet (see page 23159)	The TTabSet component presents horizontal tabs users can click to initiate actions.

161.1.1 TScroller Class

TScroller is a horizontal scrolling control that provides basic horizontal scrolling capabilities.

Class Hierarchy

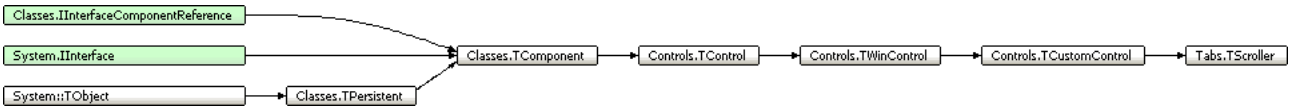

Pascal

```
TScroller = class(TCustomControl);
```

C++

```
class TScroller : public TCustomControl;
```

File

Tabs

Description

The TTabSet (see page 23159) object uses TScroller to scroll its visible tabs. TScroller is provided for backward compatibility only. TScroller is similar to an up-down control (TUpDown) with a horizontal orientation, except that it does not work with a companion control.

See Also

TTabSet (see page 23159)

TUpDown (see page 5380)

161.1.1.1 TScroller Members

The following tables list the members exposed by TScroller.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.

	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 6137)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
V	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
V	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.

	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).

	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.

	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
V	SetDragMode (see page 5785)	Sets the drag mode for the control.
V	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetName (see page 5785)	Sets the name of the control.
V	SetParent (see page 5786)	Sets the parent of the control.
V	SetParent (see page 6182)	Sets the parent of the control.
V	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
V	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.

	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 23145)	Creates an instance of TScroller (see page 23134).
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 23145)	Frees the memory associated with the TScroller (see page 23134) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Paint (see page 23146)	Paints the image of the TScroller (see page 23134) object into its canvas.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.

	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnClick (see page 23148)	OnClick occurs after the Position (see page 23148) property has been updated in response to the user clicking one of the arrow buttons.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Change (see page 23146)	Specifies the number of units to scroll when the user clicks the arrow buttons.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Max (see page 23147)	Specifies the maximum value the Position (see page 23148) property can take.
	Min (see page 23147)	Specifies the minimum value the Position (see page 23148) property can take.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Position (see page 23148)	Position is the current position in the scroll range.

	ScrollOrientation (see page 23148)	This is ScrollOrientation, a member of class TScroller.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

161.1.1.2 TScroller.Create Constructor

Creates an instance of TScroller (see page 23134).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TScroller(TComponent * AOwner);
```

Description

Call TScroller (see page 23134) to create an instance of TScroller (see page 23134) at runtime. TScroller (see page 23134) allocates the memory for the TScroller (see page 23134) object and initializes

Width to 24 and Height to 13

Min (see page 23147) and Max (see page 23147) to 0

Position (see page 23148) to 0

Change (see page 23146) to 1

The AOwner parameter specifies the component (usually the form) that is responsible for freeing the scroller.

See Also

Change (see page 23146)

Height (see page 5825)

Max (see page 23147)

Min (see page 23147)

Position (see page 23148)

Width (see page 5849)

161.1.1.3 TScroller.Destroy Destructor

Frees the memory associated with the TScroller (see page 23134) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScroller();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks whether the TScroller (see page 23134) reference is nil before calling Destroy.

See Also

Create (see page 23145)

Free (see page 22300)

161.1.1.4 TScroller Methods

The methods of the TScroller class are listed here.

Public Methods

	Name	Description
	Paint (see page 23146)	Paints the image of the TScroller (see page 23134) object into its canvas.

161.1.1.4.1 TScroller.Paint Method

Paints the image of the TScroller (see page 23134) object into its canvas.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Paint is called automatically in response to WM_PAINT messages. Paint paints the arrow buttons, graying them out when Position (see page 23148) cannot be moved in the associated direction.

See Also

Position (see page 23148)

Canvas (see page 5921)

161.1.1.5 TScroller Properties

The properties of the TScroller class are listed here.

Published Properties

	Name	Description
	Change (see page 23146)	Specifies the number of units to scroll when the user clicks the arrow buttons.
	Max (see page 23147)	Specifies the maximum value the Position (see page 23148) property can take.
	Min (see page 23147)	Specifies the minimum value the Position (see page 23148) property can take.
	Position (see page 23148)	Position is the current position in the scroll range.
	ScrollOrientation (see page 23148)	This is ScrollOrientation, a member of class TScroller.

161.1.1.5.1 TScroller.Change Property

Specifies the number of units to scroll when the user clicks the arrow buttons.

Pascal

```
property Change: Integer;
```

C++

```
__property int Change;
```

Description

Set Change to indicate the number of units between Min (see page 23147) and Max (see page 23147) the Position (see page 23148) moves whenever the user clicks on one of the arrow buttons. Change represents the mapping between the units used to specify Min (see page 23147), Max (see page 23147), and Position (see page 23148), and the logical units represented by the scroller.

For example, if the values of Min (see page 23147), Max (see page 23147), and Position (see page 23148) are given in screen pixels, Change would be the number of pixels to scroll whenever the user clicks the scroll arrow.

See Also

Max (see page 23147)

Min (see page 23147)

Position (see page 23148)

161.1.1.5.2 TScroller.Max Property

Specifies the maximum value the Position (see page 23148) property can take.

Pascal

```
property Max: Longint;
```

C++

```
__property Longint Max;
```

Description

Use Max to set the upper limit of the scrolling range. The Max and Min (see page 23147) properties define the range of values over which the scroller can scroll.

See Also

Change (see page 23146)

Min (see page 23147)

Position (see page 23148)

161.1.1.5.3 TScroller.Min Property

Specifies the minimum value the Position (see page 23148) property can take.

Pascal

```
property Min: Longint;
```

C++

```
__property Longint Min;
```

Description

Use Min to set the lower limit of the scrolling range. The Min and Max (see page 23147) properties define the range of values over which the scroller can scroll.

See Also

Change (see page 23146)

Max (see page 23147)

Position (see page 23148)

161.1.1.5.4 TScroller.Position Property

Position is the current position in the scroll range.

Pascal

```
property Position: Longint;
```

C++

```
__property Longint Position;
```

Description

Read Position to determine the current position the scroller represents. Position is a value between Min (see page 23147) and Max (see page 23147). Use an OnClick (see page 23148) event handler to notify the object being scrolled when the Position changes.

See Also

Max (see page 23147)

Min (see page 23147)

OnClick (see page 23148)

161.1.1.5.5 TScroller.ScrollOrientation Property

Pascal

```
property ScrollOrientation: TScrollOrientation;
```

C++

```
__property TScrollOrientation ScrollOrientation;
```

Description

This is ScrollOrientation, a member of class TScroller.

161.1.1.6 TScroller Events

The events of the TScroller class are listed here.

Published Events

	Name	Description
	OnClick (see page 23148)	OnClick occurs after the Position (see page 23148) property has been updated in response to the user clicking one of the arrow buttons.

161.1.1.6.1 TScroller.OnClick Event

OnClick occurs after the Position (see page 23148) property has been updated in response to the user clicking one of the arrow buttons.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Write an OnClick event handler to implement the scrolling action of the object that uses the TScroller (see page 23134) to control scrolling. When the user clicks one of the arrow buttons, the Position (see page 23148) gets updated by Change (see page 23146) (unless the Max (see page 23147) or Min (see page 23147) value is hit), and then an OnClick event occurs.

See Also

Change (see page 23146)

Max (see page 23147)

Min (see page 23147)

Position (see page 23148)

161.1.2 TTabList Class

Class Hierarchy

Pascal

```
TTabList = class(TStringList);
```

C++

```
class TTabList : public TStringList;
```

File

Tabs

Description

This is class Tabs.TTabList.

161.1.2.1 TTabList Members

The following tables list the members exposed by TTabList.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 3013)	This is Changed, a member of class TStringList.
	Changing (see page 3013)	This is Changing, a member of class TStringList.
	CompareStrings (see page 3014)	Compares two strings.
	CompareStrings (see page 3048)	Compares two strings.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	DefineProperties (see page 3049)	Reads and writes the Strings (see page 3080) property as if it were published.
	Error (see page 3052)	This is the overview for the Error method overload.
	ExtractName (see page 3053)	Returns the name portion of a string that is a name value pair.
	Get (see page 3017)	Returns a string given its index.
	Get (see page 3053)	Returns a string given its index.
	GetCapacity (see page 3017)	Returns the currently allocated size of the list of strings.
	GetCapacity (see page 3054)	Returns the currently allocated size of the list of strings.
	GetCount (see page 3018)	Returns the number of strings in the list
	GetCount (see page 3054)	Returns the number of strings in the list
	GetObject (see page 3018)	Returns the object associated with the string at a specified index.
	GetObject (see page 3055)	Returns the object associated with the string at a specified index.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTextStr (see page 3056)	Returns the value of the Text (see page 3084) property.
	InsertItem (see page 3020)	This is InsertItem, a member of class TStringList.
	PutObject (see page 3022)	Changes the object associated with the string at a specified index.
	PutObject (see page 3065)	Changes the object associated with the string at a specified index.
	SetCapacity (see page 3022)	Changes the amount of memory allocated to hold strings in the list.
	SetCapacity (see page 3068)	Changes the amount of memory allocated to hold strings in the list.
	SetTextStr (see page 3069)	Sets the Text (see page 3084) property.
	SetUpdateState (see page 3023)	Performs internal adjustments before or after a series of updates.
	SetUpdateState (see page 3069)	Performs internal adjustments before or after a series of updates.

Protected Properties

	Name	Description
	UpdateCount (see page 3085)	Indicates the number of calls to BeginUpdate (see page 3045) that have not been matched by a call to EndUpdate (see page 3050).

Public Events

	Name	Description
	OnChange (see page 3028)	Occurs immediately after the list of strings changes.
	OnChanging (see page 3028)	Occurs immediately before the list of strings changes.

Public Methods

	Name	Description
	Add (see page 23153)	Adds a new string to the list.
	AddObject (see page 3012)	Adds a string to the list, and associates an object with the string.
	AddObject (see page 3036)	Adds a string to the list, and associates an object with the string.
	AddStrings (see page 23155)	Adds a group of strings to the list.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 3041)	Adds a string to the list.
	Assign (see page 2930)	Copies the contents of another, similar object.
	Assign (see page 3043)	Sets the strings in the list, and possibly associated objects, from a source object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 3045)	Enables the TStrings (see page 3029) object to track when the list of strings is changing.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 23157)	Deletes all the strings from the list.
	Create (see page 23152)	This is Create, a member of class TTabList.
	CustomSort (see page 3014)	Sorts the strings in the list in a customized order.

	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 23157)	Removes the string specified by the Index parameter.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 3009)	Destroys an instance of TStringList (see page 3006).
	Destroy (see page 3032)	Destroys the TPersistent instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 3050)	Enables the TStrings (see page 3029) object to keep track of when the list of strings has finished changing.
	Equals (see page 3051)	Compares the list of strings to the list from another TStrings (see page 3029) object and returns true if the two lists match.
	Exchange (see page 3015)	Swaps the position of two strings in the list.
	Exchange (see page 3053)	Swaps the position of two strings in the list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Find (see page 3016)	Locates the index for a string in a sorted list and indicates whether a string with that value already exists in the list.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 3054)	Returns a TStrings (see page 3029) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetText (see page 3055)	Allocates a text buffer and fills it with the value of the Text (see page 3084) property.
	IndexOf (see page 3019)	Returns the position of a string in the list.
	IndexOf (see page 3056)	Returns the position of a string in the list.
	IndexOfName (see page 3057)	Returns the position of the first name-value pair with the specified name.
	IndexOfObject (see page 3059)	Returns the index of the first string in the list associated with a given object.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 23158)	Inserts a string to the list at the position specified by Index.
	InsertObject (see page 3020)	Inserts a string into the list at the specified position, and associates it with an object.
	InsertObject (see page 3060)	Inserts a string into the list at the specified position, and associates it with an object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LoadFromFile (see page 3061)	Fills the list with the lines of text in a specified file.
	LoadFromStream (see page 3062)	Fills the list with lines of text read from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	Move (see page 3063)	Changes the position of a string in the list.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Put (see page 23158)	Changes the value of the string with a specified index.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 3065)	Saves the strings in the list to the specified file.
	SaveToStream (see page 3067)	Writes the value of the Text (see page 3084) property to a stream object.
	SetText (see page 3068)	Sets the Text (see page 3084) property.
	Sort (see page 3023)	Sorts the strings in the list in ascending order.

Public Properties

	Name	Description
	Capacity (see page 3070)	Indicates the number of strings the TStrings (see page 3029) object can hold.

	CaseSensitive (see page 3025)	Controls (see page 5711) whether strings are located, sorted, and identified as duplicates in a case-sensitive or case-insensitive manner.
	CommaText (see page 3070)	Lists the strings in the TStrings (see page 3029) object in system data format (SDF).
R	Count (see page 3071)	Introduces an abstract property to represent the number of strings in the list.
	DelimitedText (see page 3074)	Represents all the strings in the TStrings (see page 3029) object as a single delimited string.
	Delimiter (see page 3074)	Specifies the delimiter used by the DelimitedText (see page 3074) property.
	Duplicates (see page 3025)	Specifies whether duplicate strings can be added to sorted lists.
	LineBreak (see page 3075)	Defines the line break character(s).
R	Names (see page 3075)	Indicates the name part of strings that are name-value pairs.
	NameValueSeparator (see page 3077)	Indicates the character used to separate names from values.
	Objects (see page 3077)	Represents a set of objects that are associated one with each of the strings in the Strings (see page 3080) property.
	QuoteChar (see page 3080)	Specifies the quote character used by the DelimitedText (see page 3074) property.
	Sorted (see page 3026)	Specifies whether the strings in the list should be automatically sorted.
	StrictDelimiter (see page 3080)	Determines how the Delimiter (see page 3074) property is used.
	Strings (see page 3080)	References the strings in the list by their positions.
	StringsAdapter (see page 3084)	Implements an IStringsAdapter (see page 3208) interface for the TStrings (see page 3029) object.
	Text (see page 3084)	Lists the strings in the TStrings (see page 3029) object as a single string with the individual strings delimited by carriage returns and line feeds.
	ValueFromIndex (see page 3085)	Represents the value part of a string with a given index, on strings that are name-value pairs.
	Values (see page 3086)	Represents the value part of a string associated with a given name, on strings that are name-value pairs.

161.1.2.2 TTabList.Create Constructor

Pascal

```
constructor Create(const ATabSet: TTabSet);
```

C++

```
__fastcall TTabList(const TTabSet ATabSet);
```

Description

This is Create, a member of class TTabList.

161.1.2.3 TTabList Methods

The methods of the TTabList class are listed here.

Public Methods

	Name	Description
V	Add (see page 23153)	Adds a new string to the list.
V	AddStrings (see page 23155)	Adds a group of strings to the list.
V	Clear (see page 23157)	Deletes all the strings from the list.
V	Delete (see page 23157)	Removes the string specified by the Index parameter.
V	Insert (see page 23158)	Inserts a string to the list at the position specified by Index.
V	Put (see page 23158)	Changes the value of the string with a specified index.

161.1.2.3.1 TTabList.Add Method

Adds a new string to the list.

Pascal

```
function Add(const S: string): Integer; override;
```

C++

```
virtual __fastcall int Add(const AnsiString S);
```

Description

Call Add to add the string S to the list. If the list is sorted, S is added to the appropriate position in the sort order. If the list is not sorted, S is added to the end of the list. Add returns the position of the item in the list, where the first item in the list has a value of 0.

Note: For sorted lists, Add will raise an EListError exception if the string S already appears in the list and Duplicates is set to dupError. If Duplicates is set to dupIgnore, trying to add a duplicate string causes Add to return the index of the existing entry.

C++ Examples:

```
/*
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  int Index;
  TStringList* MyList = new TStringList();
  try
  {
    MyList->Add("Animals");
    MyList->Add("Flowers");
    MyList->Add("Ocean");
    MyList->Sort(); // Find will only work on sorted lists
    if (MyList->Find("Flowers", Index))
    {
      ListBox1->Items->AddStrings(MyList);
      Label1->Caption = "Flowers has an index value of " + AnsiString(Index);
    }
  }
  __finally
  {
    delete MyList;
  }
}
/*
This example creates a stringlist, adds elements to it and
assigns the strings to items of a combo box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TStrings *StringList = new TStringList();
  try
  {
    StringList->Add("This example uses A string List. ");
```

```
    StringList->Add("It is the easiest way to add strings");
    StringList->Add("to a combobox's list of strings. ");
    StringList->Add("Always remember TStrings is abstract,");
    StringList->Add("So create a TStringList instead.");
    ComboBox1->Width = 210;
    ComboBox1->Items->Assign(StringList);
    ComboBox1->ItemIndex = 0;
  }
  __finally
  {
    delete StringList;
  }
}
```

Delphi Examples:

```
{
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  MyList: TStringList;
  Index: Integer;
begin
  MyList := TStringList.Create;
  try
    MyList.Add('Animals');
    MyList.Add('Flowers');
    MyList.Add('Cars');
    MyList.Sort;   { Find will only work on sorted lists! }
    if MyList.Find('Flowers', Index) then
    begin
      ListBox1.Items.AddStrings(MyList);
      Label1.Caption := 'Flowers has an index value of ' + IntToStr(Index);
    end;
  finally
    MyList.Free;
  end;
end;
{
This example creates a stringlist, adds elements to it and
assigns the strings to items of a combo box.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  StringList: TStringList;
begin
  StringList := TStringList.Create;
  try
    with StringList do begin
      Add('This example uses A string List.');
      Add('It is the easiest way to add strings');
      Add('to a combobox''s list of strings.');
      Add('Always remember TStrings.Create method');
      Add('is abstract; So use TStringList.Create');
      Add('method instead.');
    end;
    with ComboBox1 do begin
      Width := 210;
      Items.Assign(StringList);
      ItemIndex := 0;
```

```
    end;
  finally
    StringList.free;
  end;
end;
```

See Also

AddStrings (see page 3039)

Append (see page 3041)

Delete (see page 3015)

Duplicates (see page 3025)

Insert (see page 3019)

Sorted (see page 3026)

161.1.2.3.2 TTabList.AddStrings Method

Adds a group of strings to the list.

Pascal

```
procedure AddStrings(Strings: TStrings); override;
```

C++

```
virtual __fastcall AddStrings(TStrings Strings);
```

Description

Call AddStrings to add the strings from another TStrings object to the list. If both the source and destination TStrings objects support objects associated with their strings, references to the associated objects will be added as well.

C++ Examples:

```
/*
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  int Index;
  TStringList* MyList = new TStringList();
  try
  {
    MyList->Add("Animals");
    MyList->Add("Flowers");
    MyList->Add("Ocean");
    MyList->Sort(); // Find will only work on sorted lists
    if (MyList->Find("Flowers", Index))
    {
      ListBox1->Items->AddStrings(MyList);
      Label1->Caption = "Flowers has an index value of " + AnsiString(Index);
    }
  }
  __finally
  {
    delete MyList;
```

```
  }
}
/*
This example uses a list box on a form. When the application
runs, a string list object is created and three strings are
added to it. The strings are sorted and added to the list
box, where they appear in their sorted order:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TStringList *list = new TStringList();
  list->Add("Plants");
  list->Add("Animals");
  list->Add("Minerals");
  list->Sorted = true;
  ListBox1->Items->AddStrings(list);
  delete list;
}
```

Delphi Examples:

```
{
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  MyList: TStringList;
  Index: Integer;
begin
  MyList := TStringList.Create;
  try
    MyList.Add('Animals');
    MyList.Add('Flowers');
    MyList.Add('Cars');
    MyList.Sort;   { Find will only work on sorted lists! }
    if MyList.Find('Flowers', Index) then
    begin
      ListBox1.Items.AddStrings(MyList);
      Label1.Caption := 'Flowers has an index value of ' + IntToStr(Index);
    end;
  finally
    MyList.Free;
  end;
end;
{
This example uses a list box on a form. When the application
runs, a string list object is created and three strings are
added to it. The strings are sorted and added to the list
box, where they appear in their sorted order:
}
procedure TForm1.FormCreate(Sender: TObject);
var
  MyList: TStringList;
begin
  MyList := TStringList.Create;
  MyList.Add('Plants');
  MyList.Add('Animals');
  MyList.Add('Minerals');
  MyList.Sorted := True;
  ListBox1.Items.AddStrings(MyList);
  MyList.Free;
```

```
end;
```

See Also

Add (see page 3033)

AddObject (see page 3036)

Assign (see page 3043)

Count (see page 3071)

Objects (see page 3077)

161.1.2.3.3 TTabList.Clear Method

Deletes all the strings from the list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call clear to empty the list of strings. All references to associated objects are also removed. However, the objects themselves are not freed.

See Also

Delete (see page 3015)

Objects

Strings

161.1.2.3.4 TTabList.Delete Method

Removes the string specified by the Index parameter.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Call Delete to remove a single string from the list. If an object is associated with the string, the reference to the object is removed as well. Index gives the position of the string, where 0 is the first string, 1 is the second string, and so on.

See Also

Add (see page 3010)

AddObject (see page 3012)

Clear (see page 3013)

IndexOf (see page 3019)

InsertObject (see page 3060)

Objects

Strings

161.1.2.3.5 TTabList.Insert Method

Inserts a string to the list at the position specified by Index.

Pascal

```
procedure Insert(Index: Integer; const S: string); override;
```

C++

```
virtual __fastcall Insert(int Index, const AnsiString S);
```

Description

Call Insert to add the string S to the list at the position specified by Index. If Index is 0, the string is inserted at the beginning of the list. If Index is 1, the string is put in the second position of the list, and so on.

If the string has an associated object, use the InsertObject method instead.

Note: If the list is sorted, calling Insert or InsertObject will raise an EListError exception. Use Add or AddObject with sorted lists.

See Also

Add (see page 3010)

AddObject (see page 3012)

EListError (see page 2701)

InsertObject (see page 3020)

Sorted (see page 3026)

Strings

161.1.2.3.6 TTabList.Put Method

Changes the value of the string with a specified index.

Pascal

```
procedure Put(Index: Integer; const S: string); override;
```

C++

```
virtual __fastcall Put(int Index, const AnsiString S);
```

Description

Put is the protected write implementation of the Strings property.

Put changes the value of the string with the index specified by Index to S. Put does not change the object at the specified position. That is, any object associated with the previous string becomes associated with the new string.

See Also

Strings (see page 3080)

Objects (see page 3077)

161.1.3 TTabSet Class

The TTabSet component presents horizontal tabs users can click to initiate actions.

Class Hierarchy

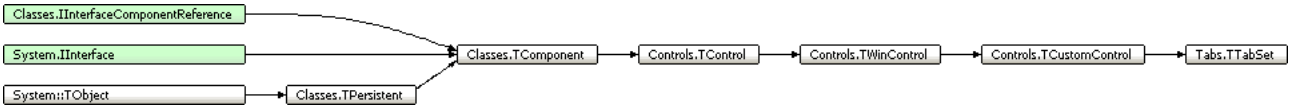

Pascal

```
TTabSet = class(TCustomControl);
```

C++

```
class TTabSet : public TCustomControl;
```

File

Tabs

Description

Tab set controls are commonly used to display tabbed pages within a dialog box. TTabSet is provided for backward compatibility. Use TTabControl component in 32-bit Windows applications.

You create a set of tabs for the tab set control when you specify a list of strings as the value of the Tabs (see page 23201) property. One tab is created for each string.

To determine which tab is currently selected or to use code to select a tab, use the TabIndex (see page 23200) property. To find out which tab is the first visible tab in the tab set control or to make a tab the first visible tab, use the FirstIndex (see page 23189) property.

161.1.3.1 TTabSet Members

The following tables list the members exposed by TTabSet.

Protected Events

	Name	Description
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.

	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanChange (see page 23171)	This is CanChange, a member of class TTabSet.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	CMHintShow (see page 23171)	This is CMHintShow, a member of class TTabSet.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.

	CreateParams (see page 23172)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	DrawTab (see page 23172)	This is DrawTab, a member of class TTabSet.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.

	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
V	GetChildren (see page 23172)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetImageIndex (see page 23173)	This is GetImageIndex, a member of class TTabSet.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
V	MeasureTab (see page 23174)	This is MeasureTab, a member of class TTabSet.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 23175)	OnMouseDown event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	Notification (see page 23176)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 23176)	Renders the image of a custom control.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.

	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetTabIndex (see page 23177)	This is SetTabIndex, a member of class TTabSet.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
R	Scroller (see page 23196)	This is Scroller, a member of class TTabSet.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 23170)	Constructs a tab set control.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 23170)	Frees the memory associated with the TTabSet (see page 23159) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ItemAtPos (see page 23173)	Returns the index of the tab indicated by the coordinates of a point on the control.
	ItemRect (see page 23173)	Returns the rectangle that surrounds the item specified in the Item parameter.
	ItemWidth (see page 23174)	Returns the width, in pixels, of a specified tab in the set.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MinClientRect (see page 23174)	This is the overview for the MinClientRect method overload.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SelectNext (see page 23177)	Selects the next tab in a tab set control, scrolling if necessary to bring the selected tab into view.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	Canvas (see page 23183)	Specifies the TCanvas object that presents a drawing surface for the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.

R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
	FirstIndex (see page 23189)	Specifies the tab that appears in the leftmost visible position in the tab set control.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 23204)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 23204)	Occurs when an object with custom alignment is aligned.
	OnChange (see page 23205)	Occurs just before a new tab is selected.
	OnClick (see page 23205)	Occurs when the user clicks the control.
	OnDragDrop (see page 23207)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 23210)	Occurs when the user drags an object over a control.
	OnDrawTab (see page 23212)	Occurs when a tab in an owner-drawn tab set control needs to redisplay.
	OnEndDock (see page 23213)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 23213)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 23214)	Occurs when a control receives the input focus.
	OnExit (see page 23216)	Occurs when the input focus shifts away from one control to another.

	OnGetImageIndex (see page 23218)	This is OnGetImageIndex, a member of class TTabSet.
	OnMeasureTab (see page 23218)	Occurs when an application needs to redisplay a tab in an owner-draw tab set.
	OnMouseActivate (see page 23219)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 23220)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 23222)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 23222)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 23222)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 23225)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 23227)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 23228)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 23178)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 23182)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 23182)	Determines if scroll buttons automatically appear in a tab set control if there isn't room in the control to display all the tabs.
	BackgroundColor (see page 23183)	Determines the background color of the tab set control.
	Constraints (see page 23185)	Specifies the size constraints for the control.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DitherBackground (see page 23186)	Determines whether the background color set with the BackgroundColor (see page 23183) property is dithered.
	DragCursor (see page 23186)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 23186)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 23187)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 23188)	Controls whether the control responds to mouse, keyboard, and timer events.
	EndMargin (see page 23189)	Determines how far, in pixels, the rightmost tab appears from the right edge of the tab set control.
	Font (see page 23190)	Controls the attributes of text written on or in the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Images (see page 23191)	This is Images, a member of class TTabSet.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ParentBackground (see page 23191)	Determines whether control uses parent's theme background.
	ParentShowHint (see page 23191)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 23192)	Identifies the pop-up menu associated with the control.
	SelectedColor (see page 23197)	Determines the color of the selected tab in the tab set control.
	ShowHint (see page 23197)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	ShrinkToFit (see page 23198)	This is ShrinkToFit, a member of class TTabSet.
	SoftTop (see page 23199)	Specifies whether a line appears at the edge of the tab set, setting off the non-current tabs.

	StartMargin (see page 23199)	Determines how far, in pixels, the first tab appears from the left edge of the tab set control.
	Style (see page 23199)	Determines whether the tab set is owner-drawn.
	TabHeight (see page 23200)	Determines the height of the tabs in an owner-drawn tab set control.
	TabIndex (see page 23200)	Determines which tab of a tab set control is currently selected.
	TabPosition (see page 23201)	This is TabPosition, a member of class TTabSet.
	Tabs (see page 23201)	Contains the list of text strings that appear on the tabs.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	UnselectedColor (see page 23201)	Determines the color of the tabs that aren't currently selected in the tab set control.
	Visible (see page 23201)	Determines whether the component appears onscreen.
R	VisibleTabs (see page 23203)	Contains the number of tabs currently visible in the tab set control.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

161.1.3.2 TTabSet.Create Constructor

Constructs a tab set control.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TTabSet(TComponent * AOwner);
```

Description

Create sets the initial values of the control, including creating a TTabList (see page 23149) object that holds the list of tabs for the control. The AOwner parameter specifies the component (usually a form) that is responsible for freeing the tab set.

See Also

Destroy (see page 23170)

161.1.3.3 TTabSet.Destroy Destructor

Frees the memory associated with the TTabSet (see page 23159) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTabSet();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks that the tab set is not nil before calling Destroy. Destroy frees the TTabList (see page 23149) object that holds the list of tabs.

See Also

Create (see page 23170)

Free (see page 22300)

161.1.3.4 TTabSet Methods

The methods of the TTabSet class are listed here.

Protected Methods

	Name	Description
	CanChange (see page 23171)	This is CanChange, a member of class TTabSet.
	CMHintShow (see page 23171)	This is CMHintShow, a member of class TTabSet.
V	CreateParams (see page 23172)	Initializes a window-creation parameter data structure.
V	DrawTab (see page 23172)	This is DrawTab, a member of class TTabSet.
V	GetChildren (see page 23172)	Calls a specified method for each child of the control.
V	GetImageIndex (see page 23173)	This is GetImageIndex, a member of class TTabSet.
V	MeasureTab (see page 23174)	This is MeasureTab, a member of class TTabSet.
V	MouseDown (see page 23175)	OnMouseDown event dispatcher.
V	Notification (see page 23176)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 23176)	Renders the image of a custom control.
V	SetTabIndex (see page 23177)	This is SetTabIndex, a member of class TTabSet.

Public Methods

	Name	Description
	ItemAtPos (see page 23173)	Returns the index of the tab indicated by the coordinates of a point on the control.
	ItemRect (see page 23173)	Returns the rectangle that surrounds the item specified in the Item parameter.
	ItemWidth (see page 23174)	Returns the width, in pixels, of a specified tab in the set.
	MinClientRect (see page 23174)	This is the overview for the MinClientRect method overload.
	SelectNext (see page 23177)	Selects the next tab in a tab set control, scrolling if necessary to bring the selected tab into view.

161.1.3.4.1 TTabSet.CanChange Method

Pascal

```
function CanChange(NewIndex: Integer): Boolean;
```

C++

```
__fastcall Boolean CanChange(int NewIndex);
```

Description

This is CanChange, a member of class TTabSet.

161.1.3.4.2 TTabSet.CMHintShow Method

Pascal

```
procedure CMHintShow(var Message: TCMHintShow); message CM_HINTSHOW;
```

C++

```
__fastcall CMHintShow(TCMHintShow Message);
```

Description

This is CMHintShow, a member of class TTabSet.

161.1.3.4.3 TTabSet.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle (see page 6137)

CreateWnd (see page 6137)

AddBiDiModeExStyle (see page 6128)

161.1.3.4.4 TTabSet.DrawTab Method

Pascal

```
procedure DrawTab(TabCanvas: TCanvas; R: TRect; Index: Integer; Selected: Boolean); virtual;
```

C++

```
virtual __fastcall DrawTab(TCanvas TabCanvas, TRect R, int Index, Boolean Selected);
```

Description

This is DrawTab, a member of class TTabSet.

161.1.3.4.5 TTabSet.GetChildren Method

Calls a specified method for each child of the control.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the streaming system that loads and saves components. Applications seldom need to call this routine.

GetChildren executes the callback specified by the Proc parameter for every child control listed by the Controls property that lists the Root parameter as its Owner.

Override GetChildren to limit or augment which child controls are saved with the control. When overriding GetChildren, call the procedure passed as the Proc parameter for every child control that should be saved. The Root parameter indicates the component (usually a form) that owns the control.

See Also

Controls (see page 6197)

Owner (see page 2807)

161.1.3.4.6 TTabSet.GetImageIndex Method

Pascal

```
function GetImageIndex(TabIndex: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int GetImageIndex(int TabIndex);
```

Description

This is GetImageIndex, a member of class TTabSet.

161.1.3.4.7 TTabSet.ItemAtPos Method

Returns the index of the tab indicated by the coordinates of a point on the control.

Pascal

```
function ItemAtPos(Pos: TPoint; IgnoreTabHeight: Boolean = False): Integer;
```

C++

```
__fastcall int ItemAtPos(TPoint Pos, Boolean IgnoreTabHeight = False);
```

Description

Call ItemAtPos to identify the tab under the point specified by the Pos parameter. Pos is the point in the control in client coordinates. If the returned index is 0, the tab is the first tab in the tab set, if the index is 1, the tab is the second tab, and so on. ItemAtPos returns –1 if there is no tab at the specified point.

See Also

ItemRect (see page 23173)

161.1.3.4.8 TTabSet.ItemRect Method

Returns the rectangle that surrounds the item specified in the Item parameter.

Pascal

```
function ItemRect(Index: Integer): TRect;
```

C++

```
__fastcall TRect ItemRect(int Index);
```

Description

Use ItemRect to obtain the rectangle that surrounds the tab when you are drawing on an owner-draw tab set.

See Also

ItemAtPos (see page 23173)

Style (see page 23199)

OnDrawTab (see page 23212)

OnMeasureTab (see page 23218)

161.1.3.4.9 TTabSet.ItemWidth Method

Returns the width, in pixels, of a specified tab in the set.

Pascal

```
function ItemWidth(Index: Integer): Integer;
```

C++

```
__fastcall int ItemWidth(int Index);
```

Description

Use ItemWidth to obtain the width, in pixels of a specified tab.

Index is the index of the tab, where 0 indicates the first tab, 1 indicates the second tab, and so on.

See Also

ItemAtPos (see page 23173)

ItemRect (see page 23173)

161.1.3.4.10 TTabSet.MeasureTab Method

Pascal

```
procedure MeasureTab(Index: Integer; var TabWidth: Integer); virtual;
```

C++

```
virtual __fastcall MeasureTab(int Index, int TabWidth);
```

Description

This is MeasureTab, a member of class TTabSet.

161.1.3.4.11 MinClientRect Method

This is the overview for the MinClientRect method overload.

Overload List

	Name	Description
	TTabSet.MinClientRect () (see page 23174)	Returns the boundaries that are required to display the tabs in the tab set.
	TTabSet.MinClientRect (Boolean) (see page 23174)	Returns the boundaries that are required to display the tabs in the tab set.
	TTabSet.MinClientRect (Integer, Boolean) (see page 23174)	Returns the boundaries that are required to display the tabs in the tab set.

161.1.3.4.11.1 TTabSet.MinClientRect Method ()

Returns the boundaries that are required to display the tabs in the tab set.

Pascal

```
function MinClientRect: TRect; overload;
function MinClientRect(IncludeScroller: Boolean): TRect; overload;
function MinClientRect(TabCount: Integer; IncludeScroller: Boolean = False): TRect; overload;
```

C++

```
__fastcall TRect MinClientRect();
__fastcall TRect MinClientRect(Boolean IncludeScroller);
__fastcall TRect MinClientRect(int TabCount, Boolean IncludeScroller = False);
```

Description

Use MinClientRect to obtain the minimum size that the tab set's client area must be to display its tabs and (if specified) a scroller for scrolling hidden tabs. MinClientRect calculates the width based on the widths of all the tabs up to a specified count, adding in the values of StartMargin (see page 23199) and EndMargin (see page 23189) and the space that appears between tabs. If requested, it also adds the width of a scroller for the tabs. It calculates the height based on the value of the TabHeight (see page 23200) property.

TabCount specifies the number of tabs to display. MinClientRect uses the widths of these tabs to calculate the width of the client rectangle. If you use a syntax that does not include a TabCount parameter, MinClientRect assumes that all tabs must be visible.

IncludeScroller indicates whether the client rectangle must include space for the scroller control that allows users to scroll the tabs, displaying any tabs that are not included in TabCount. If IncludeScroller is false, or if this parameter is omitted, MinClientRect does not include the width for the scroller in the resulting rectange.

MinClientRect returns the dimensions, in pixels, needed to display the specified number of tabs and, if specified, the scroller. The returned rectangle has an origin of 0,0.

See Also

TabHeight (see page 23200)

ItemWidth (see page 23174)

StartMargin (see page 23199)

EndMargin (see page 23189)

161.1.3.4.12 TTabSet.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which

it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove (see page 5773)

MouseUp (see page 5774)

OnMouseDown (see page 5862)

161.1.3.4.13 TTabSet.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu (see page 5835)

Notification (see page 2783)

161.1.3.4.14 TTabSet.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas (see page 5921)

PaintWindow (see page 5920)

GetDeviceContext (see page 6155)

ClipRect (see page 14486)

161.1.3.4.15 TTabSet.SelectNext Method

Selects the next tab in a tab set control, scrolling if necessary to bring the selected tab into view.

Pascal

```
procedure SelectNext(Direction: Boolean);
```

C++

```
__fastcall SelectNext(Boolean Direction);
```

Description

The value of the Direction parameters determines if the tab to the left or right is selected. If Direction is true, the tab to the right is selected. If Direction is false, the tab to the left is selected. When the last tab in either direction is selected, calling SelectNext using the same direction wraps around to the beginning of the tab order. For example, if your application has three tabs, First, Second, and Third, and Third is the current tab, calling SelectNext(true) selects First. Likewise, if First is the current tab, SelectNext(false) selects Third.

When SelectNext is called, the OnClick (see page 23205) event of the tab set occurs, followed by the OnChange (see page 23205) event, just as if the user had clicked on a new tab.

See Also

Tabs (see page 23201)

OnChange (see page 23205)

161.1.3.4.16 TTabSet.SetTabIndex Method

Pascal

```
procedure SetTabIndex(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetTabIndex(int Value);
```

Description

This is SetTabIndex, a member of class TTabSet.

161.1.3.5 TTabSet Properties

The properties of the TTabSet class are listed here.

Protected Properties

	Name	Description
R	Scroller (see page 23196)	This is Scroller, a member of class TTabSet.

Public Properties

	Name	Description
	Canvas (see page 23183)	Specifies the TCanvas object that presents a drawing surface for the control.
	FirstIndex (see page 23189)	Specifies the tab that appears in the leftmost visible position in the tab set control.

Published Properties

	Name	Description
	Align (see page 23178)	Determines how the control aligns within its container (parent control).
	Anchors (see page 23182)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 23182)	Determines if scroll buttons automatically appear in a tab set control if there isn't room in the control to display all the tabs.
	BackgroundColor (see page 23183)	Determines the background color of the tab set control.
	Constraints (see page 23185)	Specifies the size constraints for the control.
	DitherBackground (see page 23186)	Determines whether the background color set with the BackgroundColor (see page 23183) property is dithered.
	DragCursor (see page 23186)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 23186)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 23187)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 23188)	Controls whether the control responds to mouse, keyboard, and timer events.
	EndMargin (see page 23189)	Determines how far, in pixels, the rightmost tab appears from the right edge of the tab set control.
	Font (see page 23190)	Controls the attributes of text written on or in the control.
	Images (see page 23191)	This is Images, a member of class TTabSet.
	ParentBackground (see page 23191)	Determines whether control uses parent's theme background.
	ParentShowHint (see page 23191)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 23192)	Identifies the pop-up menu associated with the control.
	SelectedColor (see page 23197)	Determines the color of the selected tab in the tab set control.
	ShowHint (see page 23197)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	ShrinkToFit (see page 23198)	This is ShrinkToFit, a member of class TTabSet.
	SoftTop (see page 23199)	Specifies whether a line appears at the edge of the tab set, setting off the non-current tabs.
	StartMargin (see page 23199)	Determines how far, in pixels, the first tab appears from the left edge of the tab set control.
	Style (see page 23199)	Determines whether the tab set is owner-drawn.
	TabHeight (see page 23200)	Determines the height of the tabs in an owner-drawn tab set control.
	TabIndex (see page 23200)	Determines which tab of a tab set control is currently selected.
	TabPosition (see page 23201)	This is TabPosition, a member of class TTabSet.
	Tabs (see page 23201)	Contains the list of text strings that appear on the tabs.
	UnselectedColor (see page 23201)	Determines the color of the tabs that aren't currently selected in the tab set control.
	Visible (see page 23201)	Determines whether the component appears onscreen.
R	VisibleTabs (see page 23203)	Contains the number of tabs currently visible in the tab set control.

161.1.3.5.1 TTabSet.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
```

```
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
```

```
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
```

```
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
    end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

161.1.3.5.2 TTabSet.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

161.1.3.5.3 TTabSet.AutoScroll Property

Determines if scroll buttons automatically appear in a tab set control if there isn't room in the control to display all the tabs.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

If AutoScroll is false, the application can still access tabs that aren't visible by using the FirstIndex (see page 23189) or TabIndex (see page 23200) properties at design time or runtime, but the user can't click on the tabs with the mouse at runtime.

See Also

FirstIndex (see page 23189)

TabIndex (see page 23200)

161.1.3.5.4 TTabSet.BackgroundColor Property

Determines the background color of the tab set control.

Pascal

```
property BackgroundColor: TColor;
```

C++

```
__property TColor BackgroundColor;
```

Description

Use BackgroundColor to indicate the color that the tab set control uses to paint the area between the tabs and the border of the control.

See Also

DitherBackground (see page 23186)

SelectedColor (see page 23197)

Color (see page 5812)

161.1.3.5.5 TTabSet.Canvas Property

Specifies the TCanvas object that presents a drawing surface for the control.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control. Canvas encapsulates a Windows device context, providing all the tools and methods needed for drawing and painting.

Canvas is a protected property that is usually redeclared as public in descendants of TCustomControl.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
```

```
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas,Rect.Left,Rect.Top,index);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(index) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TStringGrid *myStringGrid;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TStringGrid")
  {
    myStringGrid = (TStringGrid *)ListBox1->Items->Objects[Index];
    // make sure we are using the TControls canvas and not the from's!
    myStringGrid->Canvas->MoveTo(myStringGrid->ClientRect.Left, myStringGrid->ClientRect.Top);
    myStringGrid->Canvas->LineTo(myStringGrid->ClientRect.Right,
myStringGrid->ClientRect.Bottom);
    myStringGrid = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
```

```
    index := ARow * DrawGrid1.ColCount + ACol;
    DrawGrid1.Canvas.Brush.Color := clWhite;
    DrawGrid1.Canvas.FillRect(Rect);
    if (gdFocused in State) then
    begin
      DrawGrid1.Canvas.DrawFocusRect(Rect);
      Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
    end;
    ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  end;

  procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  begin
    CanSelect:= True;
  end;
  {
  The following code uses ClientRect to find and draw a line
  from the top left to the bottom right of the current
  control.  Select a TControl in the ListBox list.  This
  example assumes that the current object is a descendent of
  TControl and has a public Canvas property.
  }
    if (ListBox1.Items.Objects[Index] is TStringGrid) then
    begin
      myStringGrid:= (ListBox1.Items.Objects[Index] as TStringGrid);
      with myStringGrid.ClientRect do
      begin // make sure we are using the TControl's canvas and not the form's!
        myStringGrid.Canvas.MoveTo(Left, Top);
        myStringGrid.Canvas.LineTo(Right, Bottom);
      end;
    myStringGrid:= nil;
    end;
```

See Also

Paint (see page 5920)

161.1.3.5.6 TTabSet.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

161.1.3.5.7 TTabSet.DitherBackground Property

Determines whether the background color set with the BackgroundColor (see page 23183) property is dithered.

Pascal

```
property DitherBackground: Boolean;
```

C++

```
__property Boolean DitherBackground;
```

Description

Dithering means the background is lightened by 50%, which is intended to make the tabs easier to see. If DitherBackground is true, the tab set control background is dithered. If it is false, there is no dithering.

See Also

BackgroundColor (see page 23183)

SelectedColor (see page 23197)

161.1.3.5.8 TTabSet.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

161.1.3.5.9 TTabSet.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

161.1.3.5.10 TTabSet.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
```

```
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

161.1.3.5.11 TTabSet.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

161.1.3.5.12 TTabSet.EndMargin Property

Determines how far, in pixels, the rightmost tab appears from the right edge of the tab set control.

Pascal

```
property EndMargin: Integer;
```

C++

```
__property int EndMargin;
```

Description

EndMargin determines how far in pixels the rightmost tab appears from the right edge of the tab set control. Together with the StartMargin (see page 23199) property, EndMargin can play a role in determining how many tabs can fit within the tab set control.

If AutoScroll (see page 23182) is true and scroll buttons appear in the tab set control, EndMargin determines how far in pixels the rightmost tab appears from the left edge of the scroll buttons, rather than the edge of the tab set control.

See Also

AutoScroll (see page 23182)

EndMargin

StartMargin (see page 23199)

161.1.3.5.13 TTabSet.FirstIndex Property

Specifies the tab that appears in the leftmost visible position in the tab set control.

Pascal

```
property FirstIndex: Integer;
```

C++

```
__property int FirstIndex;
```

Description

Any tabs with a lower value in the FirstIndex property scroll to the left in the tab set control and don't appear until the user scrolls the tabs.

The default value of FirstIndex is 0 indicating that the tab with an index of 0 is in the leftmost position. For example, if you have three tabs labeled First, Second, and Third with TabIndex (see page 23200) values of 0, 1, and 2, respectively, First appears first, by default, because it has an index value of 0. If you want to shift the tabs so the Second or Third tab appears leftmost in the tab set control, change the FirstIndex value to 1 or 2.

See Also

Tabs (see page 23201)

161.1.3.5.14 TTabSet.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;
```

```
procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

161.1.3.5.15 TTabSet.Images Property

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

This is Images, a member of class TTabSet.

161.1.3.5.16 TTabSet.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground: Boolean;
```

C++

```
__property Boolean ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

161.1.3.5.17 TTabSet.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

161.1.3.5.18 TTabSet.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
```

```
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
```

```
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
```

```
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

161.1.3.5.19 TTabSet.Scroller Property

Pascal

```
property Scroller: TScroller;
```

C++

```
__property TScroller Scroller;
```

Description

This is Scroller, a member of class TTabSet.

161.1.3.5.20 TTabSet.SelectedColor Property

Determines the color of the selected tab in the tab set control.

Pascal

```
property SelectedColor: TColor;
```

C++

```
__property TColor SelectedColor;
```

Description

SelectedColor determines the color of the selected tab in the tab set control.

See Also

BackgroundColor (see page 23183)

DitherBackground (see page 23186)

Color (see page 5812)

161.1.3.5.21 TTabSet.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
```

```
    for(int i = 1; i <= 10; i++)
    {
      strcpy(string, "Item");
      itoa(i, index, 10);
      strcat(string, index);
      ListBox1->Items->Add(string);
    }
    ListBox1->Hint = "Select an item";
    ListBox1->ShowHint = true;
    Application->ShowHint = true;
  }
  /*
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  */
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

161.1.3.5.22 TTabSet.ShrinkToFit Property

Pascal

```
property ShrinkToFit: Boolean;
```

C++

```
__property Boolean ShrinkToFit;
```

Description

This is ShrinkToFit, a member of class TTabSet.

161.1.3.5.23 TTabSet.SoftTop Property

Specifies whether a line appears at the edge of the tab set, setting off the non-current tabs.

Pascal

```
property SoftTop: Boolean;
```

C++

```
__property Boolean SoftTop;
```

Description

When SoftTop is false (the default), a line appears at the edge of the non-current tabs. When SoftTop is true, this line is not drawn, softening the border of the tab set.

161.1.3.5.24 TTabSet.StartMargin Property

Determines how far, in pixels, the first tab appears from the left edge of the tab set control.

Pascal

```
property StartMargin: Integer;
```

C++

```
__property int StartMargin;
```

Description

Together with the EndMargin (see page 23189) property, StartMargin determines how many tabs fit in the tab set control.

See Also

EndMargin (see page 23189)

161.1.3.5.25 TTabSet.Style Property

Determines whether the tab set is owner-drawn.

Pascal

```
property Style: TTabStyle;
```

C++

```
__property TTabStyle Style;
```

Description

Owner-draw tabs can display objects other than strings, such as graphical images. Owner-draw tabs require more programming, however, because the application needs information on how to render the image for each tab in the tab set control.

Each time an item is displayed in an owner-drawn tab, two events occur. The first is the OnMeasureTab (see page 23218) event. In the OnMeasureTab (see page 23218) event handler, you write the code that calculates the width of the tab needed to hold the text or graphical image. After the OnMeasureTab (see page 23218) event, the OnDrawTab (see page 23212) event occurs. The code you write for the OnDrawTab (see page 23212) event draws the tab and its contents using the width found with the OnMeasureTab (see page 23218) event and the height specified as the value of the TabHeight (see page 23200) property.

Note: The TTabStyle (see page 23234) type defined in the tabs unit for the Style property differs from the TTabStyle (see page 23234) type defined in the comctrls unit. Do not to use or include the comctrls unit in source files where you are working with the tab control. Doing so can lead to compiler errors because of type mismatches.

See Also

TabHeight (see page 23200)

OnDrawTab (see page 23212)

OnMeasureTab (see page 23218)

161.1.3.5.26 TTabSet.TabHeight Property

Determines the height of the tabs in an owner-drawn tab set control.

Pascal

```
property TabHeight: Integer;
```

C++

```
__property int TabHeight;
```

Description

If the Style (see page 23199) property value is tsStandard, the TabHeight property value is ignored. If the Style (see page 23199) property value is tsOwnerDraw, TabHeight is the height of the tabs.

See Also

Style (see page 23199)

161.1.3.5.27 TTabSet.TabIndex Property

Determines which tab of a tab set control is currently selected.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

The value of TabIndex can be from –1 to one less than the number of strings in the Tabs (see page 23201) property array (Count – 1). TabIndex is an index to the Tabs (see page 23201) array property: A value of 0 indicates the first tab in the tab set control, 1 is the second tab, and so on. If no tabs are selected, TabIndex has a value of –1.

When a value is assigned to TabIndex, the OnClick (see page 23205) event for the tab set occurs, followed by the OnChange (see page 23205) event, just as if the user had clicked on a new tab.

See Also

Tabs (see page 23201)

OnChange (see page 23205)

OnClick (see page 5851)

161.1.3.5.28 TTabSet.TabPosition Property

Pascal

```
property TabPosition: TTabPosition;
```

C++

```
__property TTabPosition TabPosition;
```

Description

This is TabPosition, a member of class TTabSet.

161.1.3.5.29 TTabSet.Tabs Property

Contains the list of text strings that appear on the tabs.

Pascal

```
property Tabs: TStrings;
```

C++

```
__property TStrings Tabs;
```

Description

For each string in the Tabs property, a new tab is created to display the text string. At design time, you enter the text strings you want to appear on the tabs using the String List editor. Double-click the value column of the Tabs property to display the String List editor. You can also change strings in the Tabs property at runtime.

See Also

VisibleTabs (see page 23203)

161.1.3.5.30 TTabSet.UnselectedColor Property

Determines the color of the tabs that aren't currently selected in the tab set control.

Pascal

```
property UnselectedColor: TColor;
```

C++

```
__property TColor UnselectedColor;
```

Description

UnselectedColor determines the color of the tabs that aren't currently selected in the tab set control.

See Also

BackgroundColor (see page 23183)

SelectedColor (see page 23197)

161.1.3.5.31 TTabSet.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
```

```
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

161.1.3.5.32 TTabSet.VisibleTabs Property

Contains the number of tabs currently visible in the tab set control.

Pascal

```
property VisibleTabs: Integer;
```

C++

```
__property int VisibleTabs;
```

Description

Use VisibleTabs to determine how many tabs are visible in the tab set.

See Also

Tabs (see page 23201)

161.1.3.6 TTabSet Events

The events of the TTabSet class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 23204)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 23204)	Occurs when an object with custom alignment is aligned.
	OnChange (see page 23205)	Occurs just before a new tab is selected.
	OnClick (see page 23205)	Occurs when the user clicks the control.
	OnDragDrop (see page 23207)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 23210)	Occurs when the user drags an object over a control.
	OnDrawTab (see page 23212)	Occurs when a tab in an owner-drawn tab set control needs to redisplay.
	OnEndDock (see page 23213)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

	OnEndDrag (see page 23213)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 23214)	Occurs when a control receives the input focus.
	OnExit (see page 23216)	Occurs when the input focus shifts away from one control to another.
	OnGetImageIndex (see page 23218)	This is OnGetImageIndex, a member of class TTabSet.
	OnMeasureTab (see page 23218)	Occurs when an application needs to redisplay a tab in an owner-draw tab set.
	OnMouseActivate (see page 23219)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 23220)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 23222)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 23222)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 23222)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 23225)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 23227)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 23228)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

161.1.3.6.1 TTabSet.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore: TAlignInsertBeforeEvent;
```

C++

```
__property TAlignInsertBeforeEvent OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore (see page 6137)

Align (see page 5798)

TAlignInsertBeforeEvent (see page 6293)

161.1.3.6.2 TTabSet.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition: TAlignPositionEvent;
```

C++

```
__property TAlignPositionEvent OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition (see page 6138)

Align (see page 5798)

Anchors (see page 5802)

TAlignInfo (see page 6269)

TAlignPositionEvent (see page 6293)

161.1.3.6.3 TTabSet.OnChange Event

Occurs just before a new tab is selected.

Pascal

```
property OnChange: TTabChangeEvent;
```

C++

```
__property TTabChangeEvent OnChange;
```

Description

OnChange occurs just as the TabIndex (see page 23200) value is about to change. To prevent the TabIndex (see page 23200) value from changing, set AllowChange to false in the OnChange event handler.

161.1.3.6.4 TTabSet.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
```

```
  edit control placed somewhere on the client area of the
  sheet.  Unchecking the checkbox will cause the client area
  (not the tab area) of the Tab Control to become invisible.
  }
  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    PageControl1.ActivePage.Visible := CheckBox1.Checked;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    CheckBox1.Checked := PageControl1.ActivePage.Visible;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

161.1.3.6.5 TTabSet.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
```

```
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code comes from an application that contains a list box
  and three labels, each with a different font and color.  The
  DragMode property for each of the labels is dmAutomatic.
  The user can select a label and drag it to a list box and
  drop it. When the label is dropped, the items in the list
  box assume the color and font of the dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

161.1.3.6.6 TTabSet.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
```

```
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

161.1.3.6.7 TTabSet.OnDrawTab Event

Occurs when a tab in an owner-drawn tab set control needs to redisplay.

Pascal

```
property OnDrawTab: TDrawTabEvent;
```

C++

```
__property TDrawTabEvent OnDrawTab;
```

Description

OnDrawTab occurs when a tab redisplays if the Style (see page 23199) property value is tsOwnerDraw. For example, it happens when the user selects a tab or scrolls the tabs.

You must write the code in the OnDrawTab event handler to draw the tab.

OnDrawTab occurs just after the OnMeasureTab (see page 23218) event, which contains the code to calculate the width of the tab needed. The height of the tab is determined by the value of the TabHeight (see page 23200) property of the tab set control. The code you write in the OnDrawTab event handler, therefore, must use the width determined with the OnMeasureTab (see page 23218) event to draw the tab.

See Also

TabHeight (see page 23200)

OnMeasureTab (see page 23218)

Style (see page 23199)

161.1.3.6.8 TTabSet.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

161.1.3.6.9 TTabSet.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
```

```
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

161.1.3.6.10 TTabSet.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
```

```
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

161.1.3.6.11 TTabSet.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

161.1.3.6.12 TTabSet.OnGetImageIndex Event

Pascal

```
property OnGetImageIndex: TTabGetImageEvent;
```

C++

```
__property TTabGetImageEvent OnGetImageIndex;
```

Description

This is OnGetImageIndex, a member of class TTabSet.

161.1.3.6.13 TTabSet.OnMeasureTab Event

Occurs when an application needs to redisplay a tab in an owner-draw tab set.

Pascal

```
property OnMeasureTab: TMeasureTabEvent;
```

C++

```
__property TMeasureTabEvent OnMeasureTab;
```

Description

OnMeasureTab occurs only if the Style (see page 23199) property is tsOwnerDraw. In the OnMeasureTab event handler, write the code to calculate the width needed to draw the tab. Your code is responsible for calculating and returning the tab width, depending on what you have drawn in the tab (if the tab is of Style (see page 23199) tsOwnerDraw). After the OnMeasureTab event occurs, the OnDrawTab (see page 23212) event occurs, when the OnDrawTab (see page 23212) event handler draws the tab using the width calculated in OnMeasureTab.

See Also

TabHeight (see page 23200)

OnDrawTab (see page 23212)

161.1.3.6.14 TTabSet.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

161.1.3.6.15 TTabSet.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
```

```
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
```

```
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

161.1.3.6.16 TTabSet.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

161.1.3.6.17 TTabSet.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

161.1.3.6.18 TTabSet.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
```

```
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

161.1.3.6.19 TTabSet.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
```

```
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
```

```
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

161.1.3.6.20 TTabSet.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

161.1.3.6.21 TTabSet.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

161.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TEdgePart (see page 23229)	This is record Tabs.TEdgePart.
	TEdgeType (see page 23229)	This is record Tabs.TEdgeType.
	TScrollBtn (see page 23230)	This is record Tabs.TScrollBtn.
	TScrollOrientation (see page 23230)	This is record Tabs.TScrollOrientation.
	TTabSetTabStyle (see page 23231)	This is record Tabs.TTabSetTabStyle.

161.2.1 Tabs.TEdgePart Enumeration

Pascal

```
TEdgePart = (
  epSelectedLeft,
  epUnselectedLeft,
  epSelectedRight,
  epUnselectedRight
);
```

C++

```
enum TEdgePart {
  epSelectedLeft,
  epUnselectedLeft,
  epSelectedRight,
  epUnselectedRight
};
```

File

Tabs

Description

This is record Tabs.TEdgePart.

161.2.2 Tabs.TEdgeType Enumeration

Pascal

```
TEdgeType = (
  etNone,
  etFirstIsSel,
  etFirstNotSel,
  etLastIsSel,
  etLastNotSel,
  etNotSelToSel,
  etSelToNotSel,
  etNotSelToNotSel
```

```
);
```

C++

```
enum TEdgeType {
  etNone,
  etFirstIsSel,
  etFirstNotSel,
  etLastIsSel,
  etLastNotSel,
  etNotSelToSel,
  etSelToNotSel,
  etNotSelToNotSel
};
```

File

Tabs

Description

This is record Tabs.TEdgeType.

161.2.3 Tabs.TScrollBtn Enumeration

Pascal

```
TScrollBtn = (
  sbLeft,
  sbRight
);
```

C++

```
enum TScrollBtn {
  sbLeft,
  sbRight
};
```

File

Tabs

Description

This is record Tabs.TScrollBtn.

161.2.4 Tabs.TScrollOrientation Enumeration

Pascal

```
TScrollOrientation = (
  soLeftRight,
  soUpDown
);
```

C++

```
enum TScrollOrientation {
  soLeftRight,
  soUpDown
};
```

File

Tabs

Description

This is record Tabs.TScrollOrientation.

161.2.5 Tabs.TTabSetTabStyle Enumeration

Pascal

```
TTabSetTabStyle = (
  tsStandard,
  tsOwnerDraw,
  tsSoftTabs,
  tsModernTabs,
  tsModernPopout
);
```

C++

```
enum TTabSetTabStyle {
  tsStandard,
  tsOwnerDraw,
  tsSoftTabs,
  tsModernTabs,
  tsModernPopout
};
```

File

Tabs

Description

This is record Tabs.TTabSetTabStyle.

161.3 Types

The following table lists types in this documentation.

Types

Name	Description
TDrawTabEvent (see page 23232)	TDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 23159) control.
TMeasureTabEvent (see page 23232)	TMeasureTabEvent is the type for event handlers that indicate the size of the tabs in an owner-drawn TTabSet (see page 23159) control.
TTabChangeEvent (see page 23233)	TTabChangeEvent is the type for event handlers that respond when the current tab of a TTabSet (see page 23159) control changes.
TTabSetDrawTabEvent (see page 23233)	TTabSetDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 23159) control.
TTabStyle (see page 23234)	TTabStyle indicates whether a TTabSet (see page 23159) control is owner-drawn.

161.3.1 Tabs.TDrawTabEvent Type

TDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 23159) control.

Pascal

```
TDrawTabEvent = TTabSetDrawTabEvent;
```

C++

```
TTabSetDrawTabEvent TDrawTabEvent;
```

File

Tabs

Description

TDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 23159) control.

Note: The TDrawTabEvent type defined in the tabs unit for the OnDrawTab event handler differs from the TDrawTabEvent type defined in the comctrls unit. Do not use the comctrls unit in source files where you are working with the tab control.

161.3.2 Tabs.TMeasureTabEvent Type

TMeasureTabEvent is the type for event handlers that indicate the size of the tabs in an owner-drawn TTabSet (see page 23159) control.

Pascal

```
TMeasureTabEvent = procedure (Sender: TObject; Index: Integer; var TabWidth: Integer) of
object;
```

C++

```
(Sender: TObject; Index: Integer; var TabWidth: Integer) ( TMeasureTabEvent)();
```

File

Tabs

Description

Sender is the TTabSet (see page 23159) control whose tab is about to be drawn.

Index identifies the position of the tab in the tab set control.

TabWidth is the width of the tab.

161.3.3 Tabs.TTabChangeEvent Type

TTabChangeEvent is the type for event handlers that respond when the current tab of a TTabSet (see page 23159) control changes.

Pascal

```
TTabChangeEvent = procedure (Sender: TObject; NewTab: Integer; var AllowChange: Boolean) of
object;
```

C++

```
(Sender: TObject; NewTab: Integer; var AllowChange: Boolean) ( TTabChangeEvent)();
```

File

Tabs

Description

Sender is the TTabSet (see page 23159) control whose tab is about to change.

NewTab is the tab that is about to become the selected tab.

AllowChange determines whether the change is permitted. If the event handler sets AllowChange to false, the user won't be able to select the new tab, in effect disabling it.

161.3.4 Tabs.TTabSetDrawTabEvent Type

TTabSetDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 23159) control.

Pascal

```
TTabSetDrawTabEvent = procedure (Sender: TObject; TabCanvas: TCanvas; R: TRect; Index:
Integer; Selected: Boolean) of object;
```

C++

```
(Sender: TObject; TabCanvas: TCanvas; R: TRect; Index: Integer; Selected: Boolean) (
TTabSetDrawTabEvent)();
```

File

Tabs

Description

TabCanvas is the canvas on which the item is drawn.

Index is the position of the tab in the tab set control.

R is the area in the tab where the item is to be drawn.

Selected indicates whether the tab is currently selected.

161.3.5 Tabs.TTabStyle Type

TTabStyle indicates whether a TTabSet (see page 23159) control is owner-drawn.

Pascal

```
TTabStyle = TTabSetTabStyle;
```

C++

```
TTabSetTabStyle TTabStyle;
```

File

Tabs

Description

These are the possible values:

Value	Meaning
tsStandard	Each tab has the standard size and look.

tsOwnerDrawEach tab has the height specified with the TabHeight property and width needed to hold the text or glyph.

162 TConnect Namespace

This is namespace TConnect.

Topics

Name	Description
Classes (see page 23236)	The following table lists classes in this documentation.

Classes

	Name	Description
	TLocalConnection (see page 23236)	TLocalConnection represents the connection between a client dataset or XML broker and a provider that resides in the same application.

162.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TLocalConnection (see page 23236)	TLocalConnection represents the connection between a client dataset or XML broker and a provider that resides in the same application.

162.1.1 TLocalConnection Class

TLocalConnection represents the connection between a client dataset or XML broker and a provider that resides in the same application.

Class Hierarchy

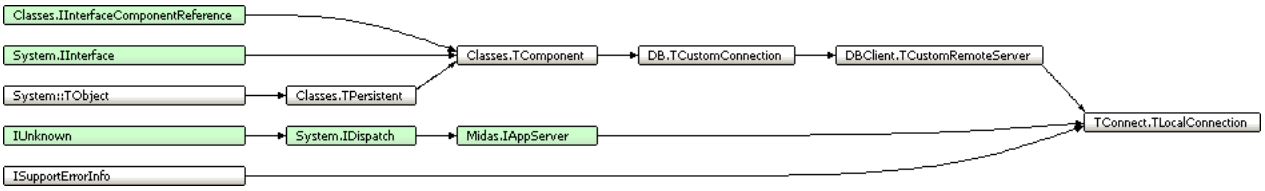

Pascal

```
TLocalConnection = class(TCustomRemoteServer, IAppServer, ISupportErrorInfo);
```

C++

```
class TLocalConnection : public TCustomRemoteServer, public IAppServer, public
ISupportErrorInfo;
```

File

TConnect

Description

Use TLocalConnection to manage the connection between one or more client datasets or XML brokers and the providers that reside in the same application. When working with local providers, it is not necessary to set the RemoteServer property of a client dataset or XML broker to a TLocalConnection component. However, using TLocalConnection lets your application communicate directly with local providers through the AppServer (see page 23250) property. This provides access to methods that would otherwise be unavailable, and makes it easier to scale up to a multi-tiered application at a later time.

TLocalConnection implements the IAppServer interface.

Warning: TLocalConnection only links clients to providers that are instances of TDataSetProvider.

See Also

TCustomClientDataSet (see page 8373)

TSocketConnection (see page 18928)

TWebConnection (see page 18968)

TDCOMConnection (see page 16008)

TCustomClientDataSet (see page 8373)

IAppServer (see page 16450)

TXMLBroker (see page 26049)

TDataSetProvider (see page 18508)

TSoapConnection (see page 19864)

Specifying a Provider

Communicating with the Client Dataset

162.1.1.1 TLocalConnection Members

The following tables list the members exposed by TLocalConnection.

Protected Events

	Name	Description
	OnGetUsername (see page 8533)	This is OnGetUsername, a member of class TCustomRemoteServer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AS_ApplyUpdates (see page 23242)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 23242)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 23243)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 23244)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 23245)	Returns a list of all the available providers on a remote data module.
	AS_GetRecords (see page 23245)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 23246)	Returns information from a specified record of the provider's dataset.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoConnect (see page 6864)	Provides the interface for a method that opens a connection.
V	DoDisconnect (see page 6864)	Provides the interface for a method that terminates the connection.
V	GetAppServer (see page 8530)	Returns the value of the AppServer (see page 8532) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetConnected (see page 23247)	Returns the value of the Connected property.
V	GetDataSet (see page 6865)	Returns a dataset, given its index in the DataSets (see page 6870) property array.

	GetDataSetCount (see page 6865)	Returns the value of the DataSetCount (see page 6869) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetProvider (see page 23247)	This is GetProvider, a member of class TLocalConnection.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InterfaceSupportsErrorInfo (see page 23248)	This is InterfaceSupportsErrorInfo, a member of class TLocalConnection.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 6866)	Initializes the connection component after all objects in the form or data module have been loaded.
	Notification (see page 23248)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RegisterClient (see page 6866)	Registers an object that uses the connection component.
	RegisterProvider (see page 23249)	This is RegisterProvider, a member of class TLocalConnection.
	SendConnectEvent (see page 6867)	Informs all registered clients that use the connection component when a connection is opened or closed.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppServer (see page 8531)	Sets the AppServer (see page 8532) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnected (see page 6867)	Changes the value of the Connected (see page 6868) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	UnRegisterClient (see page 6868)	Removes a registered client so that it is no longer associated with the connection component.
	UnRegisterProvider (see page 23250)	This is UnRegisterProvider, a member of class TLocalConnection.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	StreamedConnected (see page 6874)	Indicates whether the Connected (see page 6868) property was true when the connection component was loaded from a stream.

Public Events

	Name	Description
	AfterConnect (see page 6874)	Occurs after a connection is established.
	AfterDisconnect (see page 6875)	Occurs after the connection closes.

	BeforeConnect (see page 6875)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 6875)	Occurs immediately before the connection closes.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Close (see page 6863)	Closes the connection.
V	Create (see page 23240)	Creates and initializes a local connection component.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 23241)	Frees an instance of TLocalConnection (see page 23236).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIDsOfNames (see page 22395)	This is GetIDsOfNames, a member of class IDispatch.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetProviderNames (see page 23248)	Executes a callback for all provider objects on the application server.
V	GetServer (see page 23248)	Returns the IAppServer interface for communicating with local providers.
V	GetServerList (see page 8531)	Returns a list of application servers (remote data modules) to which the component can connect.
	GetTypeInfo (see page 22395)	This is GetTypeInfo, a member of class IDispatch.
	GetTypeInfoCount (see page 22395)	This is GetTypeInfoCount, a member of class IDispatch.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invoke (see page 22395)	This is Invoke, a member of class IDispatch.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Open (see page 6866)	Opens the connection.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 23249)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AppServer (see page 23250)	Provides access to the IAppServer interface that handles communication with providers.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Connected (see page 6868)	Determines whether a connection has been established to the remote source of data.
R	DataSetCount (see page 6869)	Indicates the number of datasets associated with the connection component.
R	DataSets (see page 6870)	Lists the datasets currently using this connection component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	LoginPrompt (see page 6872)	Specifies whether a login dialog appears immediately before opening a new connection.
	OnLogin (see page 6873)	Occurs when a communication channel opens to the server.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	ProviderCount (see page 23250)	Indicates the number of providers in the Providers (see page 23251) property list.
R	Providers (see page 23251)	Lists the local providers with which this local connection component can communicate.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

162.1.1.2 TLocalConnection.Create Constructor

Creates and initializes a local connection component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TLocalConnection(TComponent * AOwner);
```

Description

Call Create to instantiate TLocalConnection (see page 23236) at runtime. TLocalConnection (see page 23236) components placed in forms or data modules at design time are created automatically.

The AOwner parameter specifies the component (usually the form or data module) that is responsible for freeing the TLocalConnection (see page 23236) instance.

See Also

Owner (see page 2807)

162.1.1.3 TLocalConnection.Destroy Destructor

Frees an instance of TLocalConnection (see page 23236).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TLocalConnection();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which checks that the TLocalConnection (see page 23236) reference is not nil before calling the destructor.

See Also

Free (see page 22300)

162.1.1.4 TLocalConnection Methods

The methods of the TLocalConnection class are listed here.

Protected Methods

	Name	Description
	AS_ApplyUpdates (see page 23242)	Applies updates received from a client dataset using a specified provider.
	AS_DataRequest (see page 23242)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 23243)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 23244)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 23245)	Returns a list of all the available providers on a remote data module.
	AS_GetRecords (see page 23245)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 23246)	Returns information from a specified record of the provider's dataset.
V	GetConnected (see page 23247)	Returns the value of the Connected property.
V	GetProvider (see page 23247)	This is GetProvider, a member of class TLocalConnection.
	InterfaceSupportsErrorInfo (see page 23248)	This is InterfaceSupportsErrorInfo, a member of class TLocalConnection.
V	Notification (see page 23248)	Forwards notification messages to all owned components.
	RegisterProvider (see page 23249)	This is RegisterProvider, a member of class TLocalConnection.
	UnRegisterProvider (see page 23250)	This is UnRegisterProvider, a member of class TLocalConnection.

Public Methods

	Name	Description
	GetProviderNames (see page 23248)	Executes a callback for all provider objects on the application server.
	GetServer (see page 23248)	Returns the IAppServer interface for communicating with local providers.
	SafeCallException (see page 23249)	Handles exceptions in methods declared using the safecall calling convention.

162.1.1.4.1 TLocalConnection.AS_ApplyUpdates Method

Applies updates received from a client dataset using a specified provider.

Pascal

```
function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_ApplyUpdates(const BSTR ProviderName, OleVariant Delta, int
MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

AS_AS_ApplyUpdates writes client updates to a dataset or database server.

ProviderName specifies the name of the provider component that manages the update operation.

Delta is a Variant containing the updated, inserted, and deleted records to write to the database.

MaxErrors indicates the maximum number of errors to permit before stopping the update operation.

ErrorCount returns the actual number of errors encountered during the update operation.

OwnerData is information supplied by the client application in a BeforeAS_ApplyUpdates event handler and returned to the client application in an AfterAS_ApplyUpdates event handler.

AS_AS_ApplyUpdates (Delphi) or AS_ApplyUpdates_result parameter (C++) returns a Variant that is a data packet containing all records that could not be applied to the database.

See Also

ApplyUpdates (see page 8394)

applyupdates (see page 18459)

BeforeApplyUpdates (see page 18479)

AfterApplyUpdates (see page 18475)

BeforeApplyUpdates (see page 8511)

AfterApplyUpdates (see page 8497)

AS_ApplyUpdates (see page 6590)

as_applyupdates (see page 19892)

Communicating with the Client Dataset

162.1.1.4.2 TLocalConnection.AS_DataRequest Method

Generates an OnAS_DataRequest event on the specified provider component.

Pascal

```
function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

Use AS_AS_DataRequest to invoke an OnAS_DataRequest event handler of the provider named by ProviderName. This method is provided to allow application developers to customize the communication between a client application and a provider component. There is no predefined meaning for the Data parameter or the return value (Delphi) or the AS_DataRequest_result parameter (C++), which is returned to the caller.

See Also

OnDataRequest (see page 18482)

DataRequest (see page 8404)

Communicating with the Client Dataset

162.1.1.4.3 TLocalConnection.AS_Execute Method

AS_Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); safecall;
```

C++

```
__fastcall AS_Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant Params,
OleVariant OwnerData);
```

Description

Use AS_AS_Execute to pass an execute command to a remote provider, using a specified set of parameters. AS_AS_Execute calls the AS_Execute method of the specified provider after assigning any parameters. Output parameters that result from executing a query or stored procedure are returned in the Params parameter.

The provider's AS_Execute method generates a BeforeAS_Execute event, passes the command on to its associated dataset, and generates an AfterAS_Execute event. The dataset must represent a query or stored procedure that does not return a cursor.

The ProviderName parameter specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the query, or the name of a stored procedure that replaces the dataset's stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

The Params parameter encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams routine. To decode Params into a TParams object, use the Unpackparams routine.

The OwnerData parameter contains custom information that appears as an argument to the provider's BeforeAS_Execute and AfterAS_Execute event handlers. This information originates in a client dataset's BeforeAS_Execute event handler. The value returned in OwnerData is passed to the client dataset's AfterAS_Execute event handler.

See Also

DataSet (see page 18529)

beforeexecute (see page 18479)

AfterExecute (see page 18476)

Params (see page 8489)

Execute (see page 8413)

Execute (see page 18466)

BeforeExecute (see page 8514)

AfterExecute (see page 8504)

Options (see page 18440)

PackageParams (see page 8534)

UnpackParams (see page 8534)

Communicating with the Client Dataset

162.1.1.4.4 TLocalConnection.AS_GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

Use AS_AS_GetParams to retrieve the current parameter values from the dataset associated with a provider.

The ProviderName parameter specifies the name of the provider component whose dataset has the parameters.

The OwnerData represents custom information that originates in a client dataset's BeforeAS_GetParams event handler and returns information that is passed to the client dataset's AfterAS_GetParams event handler.

AS_GetParams_result (C++) or AS_AS_GetParams (Delphi) returns the parameters, encoded as a Variant array, where each element is a 2-element Variant array containing a parameter name and a parameter value. These parameters can be converted to a TParams object using the UnpackParams procedure. If the dataset has no parameters, or if the provider does not support parameter fetching, AS_GetParams_result (C++) or AS_AS_GetParams (Delphi) returns a Null Variant.

Note: AS_AS_GetParams should not be used to retrieve output parameters from stateless application servers, because parameter values may be changed by other applications. When writing a stateless application server, retrieve parameter values using the AS_Execute method instead.

See Also

DataSet (see page 18529)

Params (see page 8489)

FetchParams (see page 8415)

GetParams (see page 18467)

BeforeGetParams (see page 18480)

aftergetparams (see page 18477)

AS_Execute (see page 16453)

UnpackParams (see page 8534)

Communicating with the Client Dataset

162.1.1.4.5 TLocalConnection.AS_GetProviderNames Method

Returns a list of all the available providers on a remote data module.

Pascal

```
function AS_GetProviderNames: OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetProviderNames();
```

Description

Call AS_AS_GetProviderNames for a list of all providers available from a remote data module. These values indicate the possible values for the ProviderName property of a client dataset that uses the IAppServer interface of an application server. When calling any other IAppServer method, pass one of these names to indicate the provider that is the target of the method call.

AS_AS_GetProviderNames returns the names of the providers in a Variant array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

See Also

providers (see page 6599)

ProviderName (see page 8492)

Providers (see page 19908)

Communicating with the Client Dataset

162.1.1.4.6 TLocalConnection.AS_GetRecords Method

Returns a data packet that contains the specified data.

Pascal

```
function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetRecords(const BSTR ProviderName, int Count, int RecsOut, int
Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

Call AS_AS_GetRecords to retrieve a specified number of records. AS_GetRecords returns the requested records, starting with the current record of the provider's dataset. When working with a stateless remote data module, you may need to reposition the cursor or re-execute a query or stored procedure in the provider's BeforeAS_GetRecords event handler.

ProviderName gives the name of the provider component that provides the records.

Count indicates the number or type of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only Count records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants. For example, the following value requests metadata along with a data packet encoded in XML:

```
MetaDataOption + XMLOption
MetaDataOption || XMLOption
```

CommandText specifies an optional SQL statement that replaces the SQL of a query, or the name of a table or stored procedure that replaces the provider's table or stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

Params is a Variant containing any parameters that should be passed to the provider's dataset before it executes to generate the requested data. It returns any output parameters. You can convert between the Variant format for parameters used by this argument and a TParams object using the global PackageParams and UnpackParams routines.

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_GetRecords event handler. This information is passed to the provider's BeforeAS_GetRecords event handler. OwnerData returns information supplied by the provider's AfterAS_GetRecords event handler.

Records are returned as a data packet in a Variantparameter, AS_GetRecords_result (C++).

See Also

Data (see page 8467)

Params (see page 8489)

GetNextPacket (see page 8423)

GetRecords (see page 18468)

BeforeGetRecords (see page 8515)

BeforeGetRecords (see page 18480)

AfterGetRecords (see page 18477)

Options (see page 18440)

PackageParams (see page 8534)

UnpackParams (see page 8534)

Communicating with the Client Dataset

162.1.1.4.7 TLocalConnection.AS_RowRequest Method

Returns information from a specified record of the provider's dataset.

Pascal

```
function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_RowRequest(const BSTR ProviderName, OleVariant Row, int RequestType,
OleVariant OwnerData);
```

Description

Client datasets call AS_AS_RowRequest automatically to implement their FetchBlobs, FetchDetails , or RefreshRecord method.

The ProviderName parameter indicates the provider associated with the dataset from which information should be fetched.

The Row parameter is an OleVariant that describes the current record on the client dataset.

The RequestType parameter indicates the type of information required. It is an integer version of the TFetchOptions type. (An integer because the value may be marshaled to a remote application server). To create a value for RequestType, take the corresponding TFetchOptions value, cast it to a Byte, and cast the result to an Integer:

```
Integer(Byte([foBlobs, foDetails]));
(int) (Byte) (TFetchOptions() << foBlobs << foDetails);
```

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_RowRequest event handler. This information is passed to the provider's BeforeAS_RowRequest event handler. OwnerData returns custom information supplied by the provider's AfterAS_RowRequest event handler.

The requested data is returned as a delta packet in AS_RowRequest_result (C++).

See Also

RowRequest (see page 18472)

BeforeRowRequest (see page 18482)

AfterRowRequest (see page 18478)

FetchBlobs (see page 8414)

FetchDetails (see page 8414)

RefreshRecord (see page 8442)

Communicating with the Client Dataset

162.1.1.4.8 TLocalConnection.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected read implementation of the Connected property. Descendant classes override this method to return true if a connection is open and false otherwise. As defined in TCustomConnection, GetConnected always returns false.

See Also

Connected (see page 6868)

SetConnected (see page 6867)

162.1.1.4.9 TLocalConnection.GetProvider Method

Pascal

```
function GetProvider(const ProviderName: string): TCustomProvider; virtual;
```

C++

```
virtual __fastcall TCustomProvider GetProvider(const AnsiString ProviderName);
```

Description

This is GetProvider, a member of class TLocalConnection.

162.1.1.4.10 TLocalConnection.GetProviderNames Method

Executes a callback for all provider objects on the application server.

Pascal

```
procedure GetProviderNames(Proc: TGetStrProc); override;
```

C++

```
virtual __fastcall GetProviderNames(TGetStrProc Proc);
```

Description

Call GetProviderNames to execute the procedure specified by the Proc parameter for every provider exposed by the application server. For every provider listed by the remote data module's AS_GetProviderNames method, Proc is called with the name of the provider.

See Also

AS_GetProviderNames (see page 16454)

AppServer (see page 8532)

ProviderName (see page 8492)

162.1.1.4.11 TLocalConnection.GetServer Method

Returns the IAppServer interface for communicating with local providers.

Pascal

```
function GetServer: IAppServer; override;
```

C++

```
virtual __fastcall IAppServer GetServer();
```

Description

GetServer returns the value of the AppServer (see page 23250) property.

See Also

AppServer (see page 23250)

162.1.1.4.12 TLocalConnection.InterfaceSupportsErrorInfo Method

Pascal

```
function InterfaceSupportsErrorInfo(const iid: TIID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT InterfaceSupportsErrorInfo(const TIID iid);
```

Description

This is InterfaceSupportsErrorInfo, a member of class TLocalConnection.

162.1.1.4.13 TLocalConnection.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

162.1.1.4.14 TLocalConnection.RegisterProvider Method

Pascal

```
procedure RegisterProvider(Prov: TCustomProvider);
```

C++

```
__fastcall RegisterProvider(TCustomProvider Prov);
```

Description

This is RegisterProvider, a member of class TLocalConnection.

162.1.1.4.15 TLocalConnection.SafeCallException Method

Handles exceptions in methods declared using the safecall calling convention.

Pascal

```
function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): HResult; override;
```

C++

```
virtual __fastcall HRESULT SafeCallException(TObject * ExceptObject, void * ExceptAddr);
```

Description

SafeCallException handles exceptions in methods that use the safecall calling convention. Some classes that implement interfaces override this method to handle errors that might occur. TComponent calls the implementation of this method for the interface supported by the component, if it exists. If the component does not support interfaces, this method calls the SafeCallException method inherited from TObject, which returns E_UNEXPECTED. This is a default return value that is appropriate for classes that do not support any interfaces.

See Also

SafeCallException

162.1.1.4.16 TLocalConnection.UnRegisterProvider Method

Pascal

```
procedure UnRegisterProvider(Prov: TCustomProvider);
```

C++

```
__fastcall UnRegisterProvider(TCustomProvider Prov);
```

Description

This is UnRegisterProvider, a member of class TLocalConnection.

162.1.1.5 TLocalConnection Properties

The properties of the TLocalConnection class are listed here.

Public Properties

	Name	Description
R	AppServer (see page 23250)	Provides access to the IAppServer interface that handles communication with providers.
R	ProviderCount (see page 23250)	Indicates the number of providers in the Providers (see page 23251) property list.
R	Providers (see page 23251)	Lists the local providers with which this local connection component can communicate.

162.1.1.5.1 TLocalConnection.AppServer Property

Provides access to the IAppServer interface that handles communication with providers.

Pascal

```
property AppServer: IAppServer;
```

C++

```
__property IAppServer AppServer;
```

Description

Use AppServer to make explicit IAppServer calls to communicate with the local providers. The IAppServer interface lets you call provider methods that are otherwise unavailable because they are protected.

See Also

Providers (see page 23251)

AppServer (see page 8462)

162.1.1.5.2 TLocalConnection.ProviderCount Property

Indicates the number of providers in the Providers (see page 23251) property list.

Pascal

```
property ProviderCount: integer;
```

C++

```
__property int ProviderCount;
```

Description

Read ProviderCount to determine the number of local providers. This is the number of entries in the Providers (see page 23251) property list.

See Also

Providers (see page 23251)

162.1.1.5.3 TLocalConnection.Providers Property

Lists the local providers with which this local connection component can communicate.

Pascal

```
property Providers [const ProviderName: string]: TCustomProvider;
```

C++

```
__property TCustomProvider Providers[AnsiString const ProviderName];
```

Description

Providers provides access to the local providers that are available through the IAppServer interface. Providers only lists the providers whose Exported property is set to true.

ProviderName is the name of a provider. If you specify a name of a provider that is not listed, TLocalConnection (see page 23236) raises an exception.

See Also

ProviderCount (see page 23250)

AppServer (see page 23250)

Exported (see page 18474)

163 ToolWin Namespace

This is namespace ToolWin.

Topics

Name	Description
Classes (see page 23254)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 23289)	The following table lists structs, records, enums in this documentation.
Types (see page 23292)	The following table lists types in this documentation.

Classes

	Name	Description
	TToolDockForm (see page 23254)	TToolDockForm is a default floating dock site class for tool buttons.
	TToolDockObject (see page 23271)	Handles drag-and-dock operations for toolbar controls.
	TToolWindow (see page 23274)	TToolWindow is the ancestor for windowed controls that require a custom client area and edge borders.

Structs, Records, Enums

	Name	Description
	TEdgeBorder (see page 23289)	TEdgeBorders (see page 23292) is the type of the EdgeBorders property.
	TEdgeStyle (see page 23290)	TEdgeStyle describes the appearance of an edge border.
	TSizingOrientation (see page 23290)	TSizingOrientation specifies the dimension along which a control's size is adjusted when docked.

Types

Name	Description
TEdgeBorders (see page 23292)	This is type ToolWin.TEdgeBorders.

163.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TToolDockForm (see page 23254)	TToolDockForm is a default floating dock site class for tool buttons.
	TToolDockObject (see page 23271)	Handles drag-and-dock operations for toolbar controls.
	TToolWindow (see page 23274)	TToolWindow is the ancestor for windowed controls that require a custom client area and edge borders.

163.1.1 TToolDockForm Class

TToolDockForm is a default floating dock site class for tool buttons.

Class Hierarchy

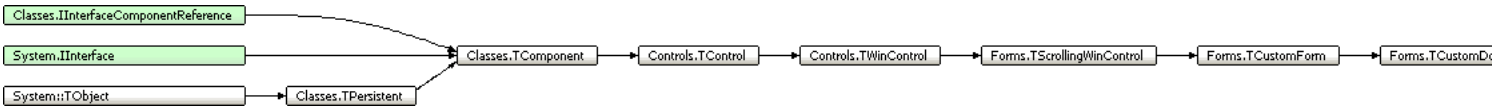

Pascal

```
TToolDockForm = class(TCustomDockForm);
```

C++

```
class TToolDockForm : public TCustomDockForm;
```

File

ToolWin

Description

Assign TToolDockForm to the FloatingDockSiteClass property of tool buttons to get appropriate docking behavior.

163.1.1.1 TToolDockForm Members

The following tables list the members exposed by TToolDockForm.

Protected Events

	Name	Description
	OnActivate (see page 13943)	Occurs when the form becomes active.
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnCanResize (see page 13947)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnClick (see page 13947)	Occurs when the user clicks the control.
	OnClose (see page 13949)	Occurs when the form closes.

	OnCloseQuery (see page 13950)	Occurs when close is attempted.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnCreate (see page 13951)	Occurs when the form is created.
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDblClick (see page 13953)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDeactivate (see page 13953)	Occurs when the form loses focus.
	OnDestroy (see page 13953)	Occurs when the form is destroyed.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragDrop (see page 13954)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnDragOver (see page 13957)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnHelp (see page 13959)	Occurs when the form receives a request for help.
	OnHide (see page 13960)	Occurs when the form is hidden (that is, when its Visible (see page 13940) property is set to false).
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyDown (see page 13962)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyPress (see page 13964)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnKeyUp (see page 13966)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseActivate (see page 13967)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseDown (see page 13968)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseMove (see page 13971)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseUp (see page 13973)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnPaint (see page 13975)	Occurs when the form is redrawn.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnResize (see page 13976)	Occurs immediately after the control is resized.
	OnShortCut (see page 13977)	Occurs when the user presses a key (before the OnKeyDown (see page 13962) event).

	OnShow (see page 13977)	Occurs when the form is shown (that is, when its Visible (see page 13940) property is set to true).
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FFormState (see page 13863)	This is FFormState, a member of class TCustomForm.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	Activate (see page 13865)	Activation event dispatcher.
	ActiveChanged (see page 13865)	Active (see page 13896) state event dispatcher.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
V	AdjustClientRect (see page 14324)	Adjusts the ClientRect property to exclude the scroll bars.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
V	AlignControls (see page 13866)	Aligns all child controls of the form that fall within a specified rectangle.
V	AlignControls (see page 14325)	Aligns the controls in the scrolling windowed control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
V	AutoScrollEnabled (see page 14325)	Indicates whether automatic scrolling is enabled.
V	AutoScrollInView (see page 14325)	Scrolls a control into the visible area of the scrolling windowed control.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
V	BeginAutoDrag (see page 13867)	Responds when the user starts dragging the form if DragMode is dmAutomatic.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.

	CanResize (see page 23269)	Specifies whether the control can resize itself.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	ChangeScale (see page 13867)	Repositions and resizes the form by the ratio M/D, adjusting its client area and font size as needed.
	ChangeScale (see page 14326)	Resizes the scrolling windowed control by a specific ratio.
	Click (see page 5743)	Respond to user click.
	CloseModal (see page 13868)	This is CloseModal, a member of class TCustomForm.
	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
	CreateHandle (see page 6135)	Creates underlying screen object.
	CreateParams (see page 23269)	Initializes the window-creation parameter record when the form window is created.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWindowHandle (see page 13869)	Creates a form window given a set of window-creation parameters.
	CreateWnd (see page 6137)	Creates the underlying window.
	CreateWnd (see page 13869)	Creates the form window.
	CreateWnd (see page 14327)	Creates a Windows control corresponding to the TScrollingWinControl (see page 14312) object.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	Deactivate (see page 13870)	Generates an OnDeactivate (see page 13953) event.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DefineProperties (see page 13871)	Designates methods for storing a form's unpublished data on a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyHandle (see page 13871)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWindowHandle (see page 13872)	Destroys the window created in the CreateWindowHandle (see page 13869) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 23269)	Performs actions when a control is docked to the form.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoClose (see page 13873)	Form closing event dispatcher.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoCreate (see page 13873)	Form creation event dispatcher.
	DoDestroy (see page 13873)	Form destruction event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDock (see page 13874)	Docking event dispatcher.

	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
V	DoFlipChildren (see page 14327)	Flips the control's children.
	DoHide (see page 13874)	Form hide event dispatcher.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
V	DoRemoveDockClient (see page 13844)	Performs actions when a control is undocked from the dock form.
	DoShow (see page 13875)	Form show event dispatcher.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
V	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
V	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetBorderIconStyles (see page 13875)	This is GetBorderIconStyles, a member of class TCustomForm.
	GetBorderStyles (see page 13876)	This is GetBorderStyles, a member of class TCustomForm.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetChildren (see page 13876)	Returns the "child" components of the form; that is, those components that return the form from their GetParentComponent method.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetClientRect (see page 13876)	Returns the value of the ClientRect property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloating (see page 13877)	Returns the value of the Floating property.

	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetOwnerWindow (see page 13878)	This is GetOwnerWindow, a member of class TCustomForm.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetSiteInfo (see page 13844)	Returns docking information for the dock form.
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleCreateException (see page 13878)	Handles exceptions that are raised by the OnCreate (see page 13951) event handler.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	Loaded (see page 13845)	Specifies actions to perform when the dock form is loaded.
	Loaded (see page 13880)	Performs fix-ups when the form is first loaded into memory.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	Notification (see page 13882)	Responds to notifications that objects are about to be inserted or removed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	Paint (see page 13882)	Generates an OnPaint (see page 13975) event.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaintWindow (see page 13882)	Renders the image of the form.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteChanged (see page 13883)	Responds to changes in the system's palette by realizing the control's palette as returned from GetPalette.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.

	QueryInterface (see page 13884)	Returns a reference to a specified interface if that interface is supported by the form.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	ReadState (see page 13884)	Reads the form's published properties, stored data, and owned components from a reader object.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	RequestAlign (see page 13885)	Instructs the form's parent (if any) to reposition the form, enforcing its Align property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	Resizing (see page 13885)	Makes any required adjustments when the form changes size.
	Resizing (see page 14328)	This is Resizing, a member of class TScrollingWinControl.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetChildOrder (see page 13886)	Changes the order in which Child appears when GetChildren (see page 13876) is called.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParent (see page 13887)	Sets the Parent property.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentBiDiMode (see page 13887)	Sets the ParentBiDiMode property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.

V	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
V	ShowControl (see page 6183)	Ensures that a specified child control is visible.
V	UpdateActions (see page 13891)	Updates all actions associated with the form.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
V	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	UpdateWindowState (see page 13892)	Updates the WindowState (see page 13942) property to reflect the current state of the form.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	ValidateRename (see page 13892)	Determines if a form can rename one of its owned components.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
V	VisibleChanging (see page 13893)	Responds just before the form changes the value of its Visible (see page 13940) property.
	WMNCCreate (see page 23270)	This is WMNCCreate, a member of class TToolDockForm.
	WMNCHitTest (see page 23270)	This is WMNCHitTest, a member of class TToolDockForm.
	WMNCLButtonDown (see page 23270)	This is WMNCLButtonDown, a member of class TToolDockForm.
	WMSize (see page 23270)	This is WMSize, a member of class TToolDockForm.
	WMSysCommand (see page 23271)	This is WMSysCommand, a member of class TToolDockForm.
V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WndProc (see page 13893)	Provides specific message responses for the form.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
R	ActiveMDIChild (see page 13897)	Specifies the MDI child that has focus.
	AlphaBlend (see page 13898)	Specifies whether the form is translucent.
	AlphaBlendValue (see page 13898)	Specifies the degree of translucency on a translucent form.
	AutoScroll (see page 13899)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoScroll (see page 14329)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderIcons (see page 13899)	Specifies which icons appear on the title bar of the form.
	BorderWidth (see page 6195)	Specifies the width of the control's border.

	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
R	ClientHandle (see page 13904)	Provides access to the handle of the form's internal MDI client window.
W	ClientHeight (see page 13905)	Specifies the height (in pixels) of the form's client area.
W	ClientWidth (see page 13905)	Specifies the width (in pixels) of the form's client area.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Ctl3D (see page 13909)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultMonitor (see page 13909)	Specifies the monitor on which the form appears.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	FormStyle (see page 13912)	Determines the form's style.
	Height (see page 13915)	Specifies the vertical size of the control in pixels.
	HorzScrollBar (see page 13916)	Represents the horizontal scroll bar for the scrolling windowed control.
	Icon (see page 13917)	Specifies the icon that appears when the form is minimized.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
R	MDIChildCount (see page 13921)	Specifies the number of open MDI child forms.
R	MDIChildren (see page 13922)	Provides indexed access to all MDI child forms.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ObjectMenuItem (see page 13926)	Represents an OLE object menu item that reacts to selections of OLE objects.
	OldCreateOrder (see page 13926)	Specifies when OnCreate (see page 13951) and OnDestroy (see page 13953) events occur.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentFont (see page 13927)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.
	PixelsPerInch (see page 13928)	Represents the proportion of the font on the system on which the form was designed.
R	PopupChildren (see page 13928)	This is PopupChildren, a member of class TCustomForm.
	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	PopupMenu (see page 13928)	Identifies the pop-up menu associated with the control.
	Position (see page 13933)	Represents the size and placement of the form.
	PrintScale (see page 13934)	Represents the proportions of a printed form.
	Scaled (see page 13935)	Specifies whether the form is sized according to the value of the PixelsPerInch (see page 13928) property.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	TileMode (see page 13936)	Represents how MDI child forms are arranged when the Tile method is called.
	TransparentColor (see page 13938)	Specifies whether a color on the form appears transparent.
	TransparentColorValue (see page 13939)	Indicates the color on the form that appears transparent when TransparentColor (see page 13938) is true.
	VertScrollBar (see page 13939)	Represents the vertical scroll bar for the scrolling windowed control.
W	Visible (see page 13940)	Indicates whether the form is visible.

	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	Width (see page 13941)	Specifies the horizontal size of the control or form in pixels.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowMenu (see page 13941)	Specifies the Window menu for an MDI parent form.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 13866)	Construction event dispatcher.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 13866)	Destruction event dispatcher.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	Close (see page 13867)	Closes the form.
V	CloseQuery (see page 13868)	Close (see page 13867) attempt event dispatcher.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 23268)	Creates an instance of TToolDockForm (see page 23254).
V	CreateNew (see page 13862)	Creates and initializes a new form.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 13870)	Provides message handling for all messages that the form does not fully process.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DefocusControl (see page 13871)	Removes focus from a control on the form.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 13862)	Removes the form from memory.
V	Destroy (see page 14323)	Removes the TScrollingWinControl (see page 14312) object from memory.
V	Destroy (see page 22287)	Disposes of an object instance.

	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	DisableAutoRange (see page 14327)	Disables automatic scrolling.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
V	Dock (see page 13872)	Handles automatic docking of the form into other windows.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EnableAutoRange (see page 14328)	Reenables automatic scrolling.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	FocusControl (see page 13875)	Sets focus to a control on the form.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFormImage (see page 13877)	Returns a bitmap of the form.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	Hide (see page 13879)	Hides the form.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	IsShortCut (see page 13880)	Processes shortcut keys when the form has focus.
	MakeFullyVisible (see page 13881)	Ensures that the form is fully visible on a specified monitor.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	MouseWheelHandler (see page 13881)	Directs mouse wheel messages to the appropriate control.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	Print (see page 13883)	Prints the form.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	RecreateAsPopup (see page 13884)	This is RecreateAsPopup, a member of class TCustomForm.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	Release (see page 13885)	Destroys the form and frees its associated memory.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	ScrollInView (see page 14328)	Scrolls a control into the visible area of the scrolling windowed control.
	SendCancelMode (see page 13886)	Cancels modes on the form.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
	SetDesignVisible (see page 6178)	Makes a control visible at design time.
	SetFocus (see page 6179)	Gives the input focus to the control.

V	SetFocus (see page 13886)	Sets focus to the form.
V	SetFocusedControl (see page 13887)	Sets focus to a control on the form.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
	Show (see page 13888)	Shows the form.
V	ShowModal (see page 13890)	Shows a form as a modal dialog.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UpdateGlassFrame (see page 13892)	This is UpdateGlassFrame, a member of class TCustomForm.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.
V	WantChildKey (see page 13893)	Indicates whether the form processes keyboard input for a control it owns.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Action (see page 13895)	Designates the action associated with the control.
R	Active (see page 13896)	Specifies whether the form has focus.
	ActiveControl (see page 13896)	Specifies the control that has focus on the form.
	ActiveOleControl (see page 13897)	Specifies the OLE control on the form that reacts to changes in focus.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 13845)	Indicates whether scroll bars appear automatically on the scrolling windowed control if it is not large enough to display all of its controls.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 13846)	Specifies the appearance and behavior of the form border.
	BorderStyle (see page 13900)	Specifies the appearance and behavior of the form border.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
R	Canvas (see page 13900)	Provides access to the drawing area of the form.
	Caption (see page 13902)	Specifies a text string that identifies the control to the user.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Color (see page 13908)	Specifies the background color of the control.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.

	Designer (see page 13910)	Specifies the designer interface for the form.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DropTarget (see page 13910)	Specifies whether the form is the target of a drag-and-drop operation.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	Font (see page 13910)	Controls the attributes of text written on or in the control.
R	FormState (see page 13911)	Indicates transitional state information about the form.
	FormStyle (see page 13846)	Determines the form's style.
	GlassFrame (see page 13914)	This is GlassFrame, a member of class TCustomForm.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HelpFile (see page 13916)	Specifies the name of the file the form uses to display Help.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	KeyPreview (see page 13918)	Specifies whether the form should receive keyboard events before the active control.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
	Menu (see page 13923)	Specifies the form's main menu.
	ModalResult (see page 13924)	Represents the return value of a form that is used as a modal dialog.
R	Monitor (see page 13926)	Provides access to the monitor on which the form appears.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
	OleFormObject (see page 13927)	Specifies the IOleForm (see page 14340) interface for an in-place OLE object contained in the form.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	PopupMode (see page 13933)	This is PopupMode, a member of class TCustomForm.
	PopupParent (see page 13933)	Sets an order for stacked forms that users cannot change.
	ScreenSnap (see page 13935)	Specifies whether form snaps to edge of screen.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	SnapBuffer (see page 13936)	Specifies distance for screen snap.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.
	WindowState (see page 13942)	Represents how the form appears on the screen.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 14332)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 14332)	Occurs when an object with custom alignment is aligned.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	HorzScrollBar (see page 14329)	Represents the horizontal scroll bar for the scrolling windowed control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Left (see page 13920)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	PixelsPerInch (see page 13848)	Represents the proportion of the font on the system on which the form was designed.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Top (see page 13937)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	VertScrollBar (see page 14330)	Represents the vertical scroll bar for the scrolling windowed control.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

163.1.1.2 TToolDockForm.Create Constructor

Creates an instance of TToolDockForm (see page 23254).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TToolDockForm(TComponent * AOwner);
```

Description

AOwner is the dock form's owner, which is responsible for freeing the tool dock form.

The constructor initializes the BorderWidth property to 1.

See Also

BorderWidth (see page 6195)

Owner (see page 2807)

163.1.1.3 TToolDockForm Methods

The methods of the TToolDockForm class are listed here.

Protected Methods

	Name	Description
	CanResize (see page 23269)	Specifies whether the control can resize itself.
	CreateParams (see page 23269)	Initializes the window-creation parameter record when the form window is created.
	DoAddDockClient (see page 23269)	Performs actions when a control is docked to the form.
	WMNCCreate (see page 23270)	This is WMNCCreate, a member of class TToolDockForm.
	WMNCHitTest (see page 23270)	This is WMNCHitTest, a member of class TToolDockForm.
	WMNCLButtonDown (see page 23270)	This is WMNCLButtonDown, a member of class TToolDockForm.
	WMSize (see page 23270)	This is WMSize, a member of class TToolDockForm.
	WMSysCommand (see page 23271)	This is WMSysCommand, a member of class TToolDockForm.

163.1.1.3.1 TToolDockForm.CanResize Method

Specifies whether the control can resize itself.

Pascal

```
function CanResize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanResize(int NewWidth, int NewHeight);
```

Description

CanResize is called automatically when an attempt is made to resize the control, after any autosizing has occurred. The NewWidth and NewHeight parameters specify the attempted new values for the control's height and width. CanResize optionally adjusts the values of NewWidth and NewHeight, and returns true if the control can be resized to the returned values. CanResize returns false if the control can't be resized.

163.1.1.3.2 TToolDockForm.CreateParams Method

Initializes the window-creation parameter record when the form window is created.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

CreateParams calls the inherited CreateParams method, and then adds WS_POPUP to the Style property.

163.1.1.3.3 TToolDockForm.DoAddDockClient Method

Performs actions when a control is docked to the form.

Pascal

```
procedure DoAddDockClient(Client: TControl; const ARect: TRect); override;
```

C++

```
virtual __fastcall DoAddDockClient(TControl * Client, const TRect ARect);
```

Description

DoAddDockClient sets the client's Align property to alNone and its BoundsRect property to ClientRect. Then DoAddDockClient calls the inherited DoAddDockClient method and sets the form's AutoSize property to true.

163.1.1.3.4 TToolDockForm.WMNCCreate Method

Pascal

```
procedure WMNCCreate(var Message: TWMNCCreate); message WM_NCCREATE;
```

C++

```
__fastcall WMNCCreate(TWMNCCreate Message);
```

Description

This is WMNCCreate, a member of class TToolDockForm.

163.1.1.3.5 TToolDockForm.WMNCHitTest Method

Pascal

```
procedure WMNCHitTest(var Message: TWMNCHitTest); message WM_NCHITTEST;
```

C++

```
__fastcall WMNCHitTest(TWMNCHitTest Message);
```

Description

This is WMNCHitTest, a member of class TToolDockForm.

163.1.1.3.6 TToolDockForm.WMNCLButtonDown Method

Pascal

```
procedure WMNCLButtonDown(var Message: TWMNCLButtonDown); message WM_NCLBUTTONDOWN;
```

C++

```
__fastcall WMNCLButtonDown(TWMNCLButtonDown Message);
```

Description

This is WMNCLButtonDown, a member of class TToolDockForm.

163.1.1.3.7 TToolDockForm.WMSize Method

Pascal

```
procedure WMSize(var Message: TWMSize); message WM_SIZE;
```

C++

```
__fastcall WMSize(TWMSize Message);
```

Description

This is WMSize, a member of class TToolDockForm.

163.1.1.3.8 TToolDockForm.WMSysCommand Method

Pascal

```
procedure WMSysCommand(var Message: TWMSysCommand); message WM_SYSCOMMAND;
```

C++

```
__fastcall WMSysCommand(TWMSysCommand Message);
```

Description

This is WMSysCommand, a member of class TToolDockForm.

163.1.2 TToolDockObject Class

Handles drag-and-dock operations for toolbar controls.

Class Hierarchy

Pascal

```
TToolDockObject = class(TDragDockObject);
```

C++

```
class TToolDockObject : public TDragDockObject;
```

File

ToolWin

Description

Use TToolDockObject to handle drag-and-dock operations for toolbar controls that are docked to a TControlBar object.

Note: TToolDockObject is designed for tool bars that are docked to TControlBar. It may not work as expected when the application is docking a tool bar to a form or panel.

See Also

TToolBar (see page 4994)

TControlBar (see page 12316)

163.1.2.1 TToolDockObject Members

The following tables list the members exposed by TToolDockObject.

Protected Methods

	Name	Description
	AdjustDockRect (see page 23273)	This is AdjustDockRect, a member of class TToolDockObject.
	DrawDragDockImage (see page 23274)	Called when the dock image for a toolbar control is drawn.
	EndDrag (see page 5723)	Generates an OnEndDrag event for the control that is being dragged.
	EndDrag (see page 6013)	Called when a control is released from dragging.
	EraseDragDockImage (see page 23274)	Called when the dock image for a toolbar control is erased.
	Finished (see page 5723)	Finished ends a drag and drop operation.

	Finished (see page 6035)	Responds when the mouse is released indicating the drag operation is complete.
	GetDragCursor (see page 6014)	Returns the cursor used during dragging.
	GetDragCursor (see page 6035)	Indicates which cursor should be displayed in response to mouse moves.
	GetDragImages (see page 6035)	Returns an image list containing the image to be displayed while dragging.
	GetEraseWhenMoving (see page 6014)	This is GetEraseWhenMoving, a member of class TDragDockObject.
	GetFrameWidth (see page 6014)	Returns the value of the FrameWidth (see page 6018) property.
	MainWndProc (see page 6037)	This is MainWndProc, a member of class TDragObject.
	WndProc (see page 6038)	This is WndProc, a member of class TDragObject.

Public Methods

	Name	Description
	AfterConstruction (see page 6033)	Responds after the last constructor has executed.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 5722)	Copies the properties of another drag control object.
	Assign (see page 6012)	Enables assignments of TDragDockObject (see page 6009) objects.
	Assign (see page 6034)	Copies the properties of another drag control object.
	BeforeDestruction (see page 6034)	Responds before the first destructor executes.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 23273)	Creates and initializes an instance of TDragDockObject.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 6012)	Destroys an instance of TDragDockObject (see page 6009).
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetName (see page 6036)	Returns a string that identifies the drag operation.
	HideDragImage (see page 6036)	Provides the interface for a method that hides the current drag image.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Instance (see page 6037)	Returns the instance (module) handle of the EXE or DLL that created the drag object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ShowDragImage (see page 6037)	Provides the interface for a method that hides the current drag image.

Public Properties

	Name	Description
	AlwaysShowDragImages (see page 6038)	This is AlwaysShowDragImages, a member of class TDragObject.
	Brush (see page 6015)	Specifies the brush used to draw the dock image.
	Cancelling (see page 6038)	Specifies whether the drag operation is being canceled.
	Control (see page 5724)	Indicates the control that is being dragged.
	DockRect (see page 6015)	Specifies the rectangle on which the control is docked if released.
	DragHandle (see page 6039)	Specifies the window handle of the dragged object.
	DragPos (see page 6039)	Specifies the position of the dragged object.
	DragTarget (see page 6039)	Specifies the dragged object's target.
	DragTargetPos (see page 6040)	Specifies the position of the drag object relative to the drag target.
R	DropAlign (see page 6017)	Specifies how the control will be aligned when docked.
R	DropOnControl (see page 6017)	Specifies the control on which the dragged control will be docked if released.
R	Dropped (see page 6040)	Drag ended with drop.
	EraseDockRect (see page 6017)	This is EraseDockRect, a member of class TDragDockObject.
R	EraseWhenMoving (see page 6018)	This is EraseWhenMoving, a member of class TDragDockObject.
	Floating (see page 6018)	Specifies whether the control is floating.
R	FrameWidth (see page 6018)	Specifies the border width, in pixels, of the frame drawn around the drag image.
R	MouseDeltaX (see page 6040)	Specifies the x-coordinate of the mouse position relative to the position of the dragged object's target.
R	MouseDeltaY (see page 6041)	Specifies the y-coordinate of the mouse position relative to the position of the dragged object's target.
	RightClickCancels (see page 6041)	This is RightClickCancels, a member of class TDragObject.

163.1.2.2 TToolDockObject.Create Constructor

Creates and initializes an instance of TDragDockObject.

Pascal

```
constructor Create(AControl: TControl); override;
```

C++

```
virtual __fastcall TToolDockObject(TControl * AControl);
```

Description

Create calls the inherited constructor and allocates memory for the Brush property.

163.1.2.3 TToolDockObject Methods

The methods of the TToolDockObject class are listed here.

Protected Methods

	Name	Description
V	AdjustDockRect (see page 23273)	This is AdjustDockRect, a member of class TToolDockObject.
V	DrawDragDockImage (see page 23274)	Called when the dock image for a toolbar control is drawn.
V	EraseDragDockImage (see page 23274)	Called when the dock image for a toolbar control is erased.

163.1.2.3.1 TToolDockObject.AdjustDockRect Method

Pascal

```
procedure AdjustDockRect(ARect: TRect); override;
```

C++

```
virtual __fastcall AdjustDockRect(TRect ARect);
```

Description

This is AdjustDockRect, a member of class TToolDockObject.

163.1.2.3.2 TToolDockObject.DrawDragDockImage Method

Called when the dock image for a toolbar control is drawn.

Pascal

```
procedure DrawDragDockImage; override;
```

C++

```
virtual __fastcall DrawDragDockImage();
```

Description

DrawDragDockImage draws a control that is moved in a toolbar.

163.1.2.3.3 TToolDockObject.EraseDragDockImage Method

Called when the dock image for a toolbar control is erased.

Pascal

```
procedure EraseDragDockImage; override;
```

C++

```
virtual __fastcall EraseDragDockImage();
```

Description

EraseDragDockImage erases a control that is moved in a toolbar.

163.1.3 TToolWindow Class

TToolWindow is the ancestor for windowed controls that require a custom client area and edge borders.

Class Hierarchy

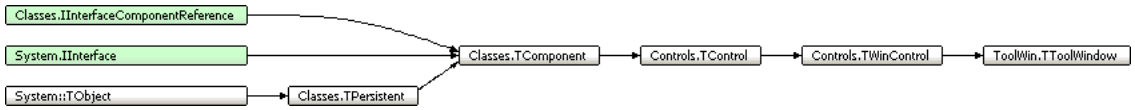

Pascal

```
TToolWindow = class(TWinControl);
```

C++

```
class TToolWindow : public TWinControl;
```

File

ToolWin

Description

TToolWindow is the ancestor of the TCoolBar and TToolBar components, which facilitate the arrangement of buttons and other visual controls.

See Also

TCoolBar (see page 3603)

TToolBar (see page 4994)

163.1.3.1 TToolWindow Members

The following tables list the members exposed by TToolWindow.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnClick (see page 5851)	Occurs when the user clicks the control.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnContextPopup (see page 5854)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 5854)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 5855)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 5857)	Occurs when the user drags an object over a control.
	OnEndDock (see page 5859)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 5860)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 6219)	Occurs when a control receives the input focus.
	OnExit (see page 6221)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnKeyDown (see page 6225)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 6227)	Occurs when a key is pressed.
	OnKeyUp (see page 6228)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 5861)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 5862)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 5864)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 5864)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 5865)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 5867)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnMouseWheelDown (see page 5869)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 5870)	Occurs when the mouse wheel is rotated upward.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnStartDock (see page 5871)	Occurs when the user begins to drag a control with a DragKind (see page 5819) of dkDock.
	OnStartDrag (see page 5871)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
V	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
V	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
V	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
V	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
V	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	Click (see page 5743)	Respond to user click.
	CMBorderChanged (see page 23285)	This is CMBorderChanged, a member of class TToolWindow.
	CMCtl3DChanged (see page 23286)	This is CMCtl3DChanged, a member of class TToolWindow.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateHandle (see page 6135)	Creates underlying screen object.

	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
	CreateWnd (see page 6137)	Creates the underlying window.
	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.
	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 6144)	Respond to losing input focus
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.

	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 6153)	Calls a specified method for each child of the control.
V	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
V	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
V	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
V	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.
V	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
V	GetDeviceContext (see page 5760)	Returns a device context for the control.
V	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
V	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
V	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
V	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
V	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 6163)	Respond to key press events.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyUp (see page 6164)	Respond to released key.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.
	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
	MouseDown (see page 5773)	OnMouseDown (see page 5862) event dispatcher.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	NCPaint (see page 23286)	This is NCPaint, a member of class TToolWindow.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.

	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
	Resize (see page 5780)	Respond to control resize.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.

	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.
	WMNCCalcSize (see page 23286)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCPaint (see page 23286)	This is WMNCPaint, a member of class TToolWindow.
	WndProc (see page 5791)	Provides specific message responses for the control.
	WndProc (see page 6187)	Provides specific message responses for the control.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Color (see page 5812)	Specifies the background color of the control.
	Ctl3D (see page 6198)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
	DragCursor (see page 5818)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 5819)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 5819)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Font (see page 5823)	Controls (see page 5711) the attributes of text written on or in the control.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 5833)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	ParentColor (see page 5834)	Determines where a control looks for its color information.
	ParentCtl3D (see page 6209)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 5834)	Determines where a control looks for its font information.
	ParentShowHint (see page 5835)	Determines where a control looks to find out if its Help Hint (see page 5827) should be shown.

	PopupMenu (see page 5835)	Identifies the pop-up menu associated with the control.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Text (see page 5842)	Contains a text string associated with the control.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
V	Create (see page 23285)	Creates and initializes a TToolWindow (see page 23274) instance.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
V	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
V	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 5732)	Destroys an instance of TControl (see page 5724).
V	Destroy (see page 6124)	Destroys an instance of TWinControl (see page 6112).
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.

	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	Invalidate (see page 5770)	Completely repaint control.
V	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.

V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 5779)	Repaints the control on the screen.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
	Align (see page 5798)	Determines how the control aligns within its container (parent control).
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	Anchors (see page 5802)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 5803)	Specifies the bi-directional mode for the control.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.

R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Constraints (see page 5812)	Specifies the size constraints for the control.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EdgeBorders (see page 23287)	Determines which sides of the control have visible borders.
	EdgeInner (see page 23287)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 23288)	Determines the border style for the control's outer edge.
	Enabled (see page 5821)	Controls (see page 5711) whether the control responds to mouse, keyboard, and timer events.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	ShowHint (see page 5840)	Determines whether the control displays a Help Hint (see page 5827) when the mouse pointer rests momentarily on the control.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabOrder (see page 6211)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

	Visible (see page 5847)	Determines whether the component appears onscreen.
R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.

Published Properties

	Name	Description
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

163.1.3.2 TToolWindow.Create Constructor

Creates and initializes a TToolWindow (see page 23274) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TToolWindow(TComponent * AOwner);
```

Description

Create generates a TToolWindow (see page 23274) and initializes its properties.

163.1.3.3 TToolWindow Methods

The methods of the TToolWindow class are listed here.

Protected Methods

	Name	Description
	CMBorderChanged (see page 23285)	This is CMBorderChanged, a member of class TToolWindow.
	CMCtl3DChanged (see page 23286)	This is CMCtl3DChanged, a member of class TToolWindow.
V	NCPaint (see page 23286)	This is NCPaint, a member of class TToolWindow.
	WMNCCalcSize (see page 23286)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCPaint (see page 23286)	This is WMNCPaint, a member of class TToolWindow.

163.1.3.3.1 TToolWindow.CMBorderChanged Method

Pascal

```
procedure CMBorderChanged(var Message: TMessage); message CM_BORDERCHANGED;
```

C++

```
__fastcall CMBorderChanged(TMessage Message);
```

Description

This is CMBorderChanged, a member of class TToolWindow.

163.1.3.3.2 TToolWindow.CMCtl3DChanged Method

Pascal

```
procedure CMCtl3DChanged(var Message: TMessage); message CM_CTL3DCHANGED;
```

C++

```
__fastcall CMCtl3DChanged(TMessage Message);
```

Description

This is CMCtl3DChanged, a member of class TToolWindow.

163.1.3.3.3 TToolWindow.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); virtual;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TToolWindow.

163.1.3.3.4 TToolWindow.WMNCCalcSize Method

Pascal

```
procedure WMNCCalcSize(var Message: TWMNCCalcSize); message WM_NCCALCSIZE;
```

C++

```
__fastcall WMNCCalcSize(TWMNCCalcSize Message);
```

Description

This is WMNCCalcSize, a member of class TToolWindow.

163.1.3.3.5 TToolWindow.WMNCPaint Method

Pascal

```
procedure WMNCPaint(var Message: TMessage); message WM_NCPAINT;
```

C++

```
__fastcall WMNCPaint(TMessage Message);
```

Description

This is WMNCPaint, a member of class TToolWindow.

163.1.3.4 TToolWindow Properties

The properties of the TToolWindow class are listed here.

Public Properties

	Name	Description
	EdgeBorders (see page 23287)	Determines which sides of the control have visible borders.
	EdgeInner (see page 23287)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 23288)	Determines the border style for the control's outer edge.

163.1.3.4.1 TToolWindow.EdgeBorders Property

Determines which sides of the control have visible borders.

Pascal

```
property EdgeBorders: TEdgeBorders;
```

C++

```
__property TEdgeBorders EdgeBorders;
```

Description

Use the EdgeBorders property to add or remove borders. The appearance of the borders can be set with the EdgeInner (see page 23287) and EdgeOuter (see page 23288) properties.

See Also

EdgeInner (see page 23287)

EdgeOuter (see page 23288)

163.1.3.4.2 TToolWindow.EdgeInner Property

Determines the border style for the control's inner edge.

Pascal

```
property EdgeInner: TEdgeStyle;
```

C++

```
__property TEdgeStyle EdgeInner;
```

Description

Use EdgeInner to provide the tool window with a three-dimensional beveled look. A tool window has two borders on each side, an outer border drawn next to the outer edge, and an inner border drawn inside the outer border. EdgeInner specifies the appearance of the inner border. It can be raised (esRaised), lowered (esLowered), or left off altogether (esNone).

Note: The EdgeInner property has an effect only on the sides specified by the EdgeBorders (see page 23287) property.

Use EdgeInner with EdgeOuter (see page 23288) to give the component a three-dimensional look.

See Also

EdgeBorders (see page 23287)

EdgeOuter (see page 23288)

163.1.3.4.3 TToolWindow.EdgeOuter Property

Determines the border style for the control's outer edge.

Pascal

```
property EdgeOuter: TEdgeStyle;
```

C++

```
__property TEdgeStyle EdgeOuter;
```

Description

Use EdgeOuter to provide the tool window with a three-dimensional beveled look. A tool window has two borders on each side, an outer border drawn next to the outer edge, and an inner border drawn inside the outer border. EdgeOuter specifies the appearance of the outer border. It can be raised (esRaised), lowered (esLowered), or left off altogether (esNone).

Note: The EdgeInner (see page 23287) property has an effect only on the sides specified by the EdgeBorders (see page 23287) property.

Use EdgeOuter with EdgeInner (see page 23287) to give the component a three-dimensional look.

See Also

EdgeBorders (see page 23287)

EdgeInner (see page 23287)

163.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TEdgeBorder (see page 23289)	TEdgeBorders (see page 23292) is the type of the EdgeBorders property.
	TEdgeStyle (see page 23290)	TEdgeStyle describes the appearance of an edge border.
	TSizingOrientation (see page 23290)	TSizingOrientation specifies the dimension along which a control's size is adjusted when docked.

163.2.1 ToolWin.TEdgeBorder Enumeration

TEdgeBorders (see page 23292) is the type of the EdgeBorders property.

Pascal

```
TEdgeBorder = (
  ebLeft,
  ebTop,
  ebRight,
  ebBottom
);
```

C++

```
enum TEdgeBorder {
  ebLeft,
  ebTop,
  ebRight,
  ebBottom
};
```

File

ToolWin

Description

TEdgeBorders (see page 23292) is a set of TEdgeBorder values. Each TEdgeBorder value indicates whether a beveled edge appears on a particular side of a control. The following table lists the TEdgeBorder values:

Value	Meaning
ebLeft	The control has a beveled left edge.
ebTop	The control has a beveled top edge.
ebRight	The control has a beveled right edge.
ebBottom	The control has a beveled bottom edge.

In addition to the values in the preceding table, the QGraphics unit defines the ebRect constant to represent all the edges of a control:

const ebRect = [ebLeft, ebTop, ebRight, ebBottom];

#define ebRect (System::Set<TEdgeBorder, ebLeft, ebBottom> () << TEdgeBorder(0) << TEdgeBorder(1) << TEdgeBorder(2)

<< TEdgeBorder(3))

163.2.2 ToolWin.TEdgeStyle Enumeration

TEdgeStyle describes the appearance of an edge border.

Pascal

```
TEdgeStyle = (
  esNone,
  esRaised,
  esLowered
);
```

C++

```
enum TEdgeStyle {
  esNone,
  esRaised,
  esLowered
};
```

File

ToolWin

Description

TEdgeStyle indicates the appearance of a particular border. The following table lists the possible values:

Value	Meaning
esNone	The border is not drawn.
esRaised	The border appears raised.
esLowered	The border appears lowered.

163.2.3 ToolWin.TSizingOrientation Enumeration

TSizingOrientation specifies the dimension along which a control's size is adjusted when docked.

Pascal

```
TSizingOrientation = (
  soNone,
  soHorizontal,
  soVertical
);
```

C++

```
enum TSizingOrientation {
  soNone,
  soHorizontal,
  soVertical
};
```

File

ToolWin

Description

TSizingOrientation can have one of the following values:

Value	Meaning
soNone	The control's size is not readjusted when it is docked.
soHorizontal	The control's width is adjusted when it is docked.
soVertical	The control's height is adjusted when it is docked.

163.3 Types

The following table lists types in this documentation.

Types

Name	Description
TEdgeBorders (see page 23292)	This is type ToolWin.TEdgeBorders.

163.3.1 ToolWin.TEdgeBorders Type

Pascal

```
TEdgeBorders = set of TEdgeBorder;
```

C++

```
TEdgeBorder TEdgeBorders;
```

File

ToolWin

Description

This is type ToolWin.TEdgeBorders.

164 Types Namespace

This is namespace Types.

Topics

Name	Description
Functions (see page 23295)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 23301)	The following table lists structs, records, enums in this documentation.
Types (see page 23304)	The following table lists types in this documentation.
Constants (see page 23316)	The following table lists constants in this documentation.

Constants

Name	Description
EqualsValue (see page 23316)	EqualsValue indicates that both values are equal in a comparison operation.
GreaterThanValue (see page 23316)	GreaterThanValue indicates that one value is greater than an other in a comparison operation.
LessThanValue (see page 23317)	LessThanValue indicates that one value is less than an other in a comparison operation.
RT_RCDATA (see page 23317)	This is constant Types.RT_RCDATA.

Functions

	Name	Description
	Bounds (see page 23295)	Returns the TRect (see page 23302) for a rectangle of given dimensions.
	CenterPoint (see page 23295)	Returns the point in the center of a rectangle.
	EqualRect (see page 23296)	Indicates whether two TRect (see page 23302) values are the same.
	IntersectRect (see page 23296)	Returns the intersection of two rectangles
	IsRectEmpty (see page 23297)	Indicates wither a specified rectangle has a positive width and height.
	OffsetRect (see page 23297)	Changes the origin of a rectangle by a specified amount.
	Point (see page 23298)	Creates a TPoint (see page 23301) structure from a pair of coordinates.
	PtInRect (see page 23298)	Indicates whether a specified point lies inside a specified rectangle.
	Rect (see page 23299)	Creates a TRect (see page 23302) structure from a set of coordinates.
	SmallPoint (see page 23299)	Creates a TSmallPoint (see page 23303) structure from a pair of coordinates.
	SmallPoint (see page 23299)	Creates a TSmallPoint (see page 23303) structure from a pair of coordinates.
	UnionRect (see page 23300)	Returns the union of two rectangles

Structs, Records, Enums

	Name	Description
	tagSIZE (see page 23301)	tagSIZE is a record holding width and height values.
	TPoint (see page 23301)	TPoint type defines a pixel location onscreen.
	TRect (see page 23302)	TRect defines a rectangle.
	TSmallPoint (see page 23303)	TSmallPoint type defines a point with two 16-bit coordinates.

Types

Name	Description
DWORD (see page 23304)	DWORD is equivalent to LongWord.
PByte (see page 23305)	PByte is a pointer to a Byte variable.
PDouble (see page 23305)	PDouble is a pointer to a Double variable.
PInteger (see page 23306)	PInteger is a pointer to a Integer variable.
PLongint (see page 23306)	This is type Types.PLongint.
PPoint (see page 23306)	PPoint is a pointer to a TPoint (see page 23301) variable.
PRect (see page 23307)	PRect is a pointer to a TRect (see page 23302) variable.
PSize (see page 23307)	PSize is a pointer to a TSize (see page 23313) variable.
PSmallInt (see page 23308)	This is type Types.PSmallInt.
PSmallPoint (see page 23308)	This is type Types.PSmallPoint.
SIZE (see page 23308)	This is type Types.SIZE.
tagPOINT (see page 23309)	This is type Types.tagPOINT.
TBooleanDynArray (see page 23309)	TBooleanDynArray defines an array of Boolean elements.
TByteDynArray (see page 23309)	This is type Types.TByteDynArray.
TCardinalDynArray (see page 23310)	TCardinalDynArray defines an array of Cardinal elements.
TDoubleDynArray (see page 23310)	TDoubleDynArray defines an array of Double elements.
TInt64DynArray (see page 23311)	TInt64DynArray defines an array of Int64 elements.
TIntegerDynArray (see page 23311)	This is type Types.TIntegerDynArray.
TLongWordDynArray (see page 23311)	TLongWordDynArray defines an array of LongWord elements.
TShortIntDynArray (see page 23312)	TShortIntDynArray defines an array of ShortInt elements.
TSingleDynArray (see page 23312)	TSingleDynArray defines an array of Single elements.
TSize (see page 23313)	TSize represents a width and height.
TSmallIntDynArray (see page 23313)	TSmallIntDynArray defines an array of SmallInt elements.
TStringDynArray (see page 23313)	TStringDynArray defines an array of String elements.
TValueRelationship (see page 23314)	TValueRelationship defines value comparison possibilities.
TWideStringDynArray (see page 23314)	TWideStringDynArray defines an array of WideString elements.
TWordDynArray (see page 23315)	TWordDynArray defines an array of Word elements.

164.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	Bounds (see page 23295)	Returns the TRect (see page 23302) for a rectangle of given dimensions.
	CenterPoint (see page 23295)	Returns the point in the center of a rectangle.
	EqualRect (see page 23296)	Indicates whether two TRect (see page 23302) values are the same.
	IntersectRect (see page 23296)	Returns the intersection of two rectangles
	IsRectEmpty (see page 23297)	Indicates wither a specified rectangle has a positive width and height.
	OffsetRect (see page 23297)	Changes the origin of a rectangle by a specified amount.
	Point (see page 23298)	Creates a TPoint (see page 23301) structure from a pair of coordinates.
	PtInRect (see page 23298)	Indicates whether a specified point lies inside a specified rectangle.
	Rect (see page 23299)	Creates a TRect (see page 23302) structure from a set of coordinates.
	SmallPoint (see page 23299)	Creates a TSmallPoint (see page 23303) structure from a pair of coordinates.
	SmallPoint (see page 23299)	Creates a TSmallPoint (see page 23303) structure from a pair of coordinates.
	UnionRect (see page 23300)	Returns the union of two rectangles

164.1.1 Types.Bounds Function

Returns the TRect (see page 23302) for a rectangle of given dimensions.

Pascal

```
function Bounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer): TRect;
```

C++

```
TRect Bounds(int ALeft, int ATop, int AWidth, int AHeight);
```

File

Types

Description

Call Bounds to obtain a TRect (see page 23302) for a rectangle with upper left corner at the point (ALeft, ATop) with width AWidth and height AHeight.

See Also

Rect (see page 23299)

164.1.2 Types.CenterPoint Function

Returns the point in the center of a rectangle.

Pascal

```
function CenterPoint(const Rect: TRect): TPoint;
```

C++

```
TPoint CenterPoint(const TRect Rect);
```

File

Types

Description

Call CenterPoint to obtain the point in the center of a rectangle.

Rect (see page 23299) is the rectangle whose center you want to locate.

See Also

PtInRect (see page 23298)

IsRectEmpty (see page 23297)

UnionRect (see page 23300)

IntersectRect (see page 23296)

164.1.3 Types.EqualRect Function

Indicates whether two TRect (see page 23302) values are the same.

Pascal

```
function EqualRect(const R1: TRect; const R2: TRect): Boolean;
```

C++

```
Boolean EqualRect(const TRect R1, const TRect R2);
```

File

Types

Description

Call EqualRect to compare two TRect (see page 23302) values. EqualRect returns true if R1 and R2 are identical (same position and size), false if they are different.

See Also

Rect (see page 23299)

Bounds (see page 23295)

IntersectRect (see page 23296)

164.1.4 Types.IntersectRect Function

Returns the intersection of two rectangles

Pascal

```
function IntersectRect(out Rect: TRect; const R1: TRect; const R2: TRect): Boolean;
```

C++

```
Boolean IntersectRect(TRect Rect, const TRect R1, const TRect R2);
```

File

Types

Description

Call IntersectRect to obtain the rectangle that is the intersection of R1 and R2. The intersection is returned as the Rect (see page 23299) parameter.

IntersectRect returns true if the two rectangles have a nonempty intersection. If R1 and R2 do not overlap, IntersectRect returns false, and the Rect (see page 23299) parameter is set to a rectangle at position (0,0) with 0 Width and 0 Height.

See Also

PtInRect (see page 23298)

IsRectEmpty (see page 23297)

UnionRect (see page 23300)

164.1.5 Types.IsRectEmpty Function

Indicates wither a specified rectangle has a positive width and height.

Pascal

```
function IsRectEmpty(const Rect: TRect): Boolean;
```

C++

```
Boolean IsRectEmpty(const TRect Rect);
```

File

Types

Description

Call IsRectEmpty to determine whether the rectangle specified by Rect (see page 23299) encloses a positive amount of space.

IsRectEmpty returns true if both the width and height of Rect (see page 23299) are greater than 0. If either the width or height is less than or equal to 0, IsRectEmpty returns false. Note that this means non-normalized rectangles (rectangles with negative height or width) are considered empty.

See Also

PtInRect (see page 23298)

IntersectRect (see page 23296)

164.1.6 Types.OffsetRect Function

Changes the origin of a rectangle by a specified amount.

Pascal

```
function OffsetRect(var Rect: TRect; DX: Integer; DY: Integer): Boolean;
```

C++

```
Boolean OffsetRect(TRect Rect, int DX, int DY);
```

File

Types

Description

Call OffsetRect to move the upper left corner of the rectangle specified by Rect (see page 23299) while maintaining the same width and height. The left of the rectangle moves by DX while the top moves by DY.

OffsetRect returns true if the Rect (see page 23299) parameter is not nil (Delphi) or NULL (C++), false if it can't return a new rectangle because Rect (see page 23299) is nil(Delphi) or NULL (C++).

See Also

Rect (see page 23299)

Bounds (see page 23295)

164.1.7 Types.Point Function

Creates a TPoint (see page 23301) structure from a pair of coordinates.

Pascal

```
function Point(X: Integer; Y: Integer): TPoint;
```

C++

```
TPoint Point(int X, int Y);
```

File

Types

Description

Call Point to create a TPoint (see page 23301) that represents the specified coordinates. Use Point to construct parameters for functions that require a point, rather than setting up local variables for each parameter.

See Also

SmallPoint (see page 23299)

Rect (see page 23299)

164.1.8 Types.PtInRect Function

Indicates whether a specified point lies inside a specified rectangle.

Pascal

```
function PtInRect(const Rect: TRect; const P: TPoint): Boolean;
```

C++

```
Boolean PtInRect(const TRect Rect, const TPoint P);
```

File

Types

Description

PtInRect returns true if the point, P, lies inside the rectangle, Rect (see page 23299), and false if P is not in Rect (see page 23299).

PtInRect does not consider a point inside a rectangle if the rectangle has a negative width or height. A point is considered inside a rectangle if it lies on the left or top edge, but not if it lies on the right or bottom edge.

See Also

IntersectRect (see page 23296)

IsRectEmpty (see page 23297)

CenterPoint (see page 23295)

164.1.9 Types.Rect Function

Creates a TRect (see page 23302) structure from a set of coordinates.

Pascal

```
function Rect(Left: Integer; Top: Integer; Right: Integer; Bottom: Integer): TRect;
```

C++

```
TRect Rect(int Left, int Top, int Right, int Bottom);
```

File

Types

Description

Call Rect to create a TRect (see page 23302) that represents the rectangle with the specified coordinates. Use Rect to construct parameters for functions that require TRect (see page 23302), rather than setting up local variables for each parameter.

See Also

Bounds (see page 23295)

Point (see page 23298)

164.1.10 Types.SmallPoint Function

Creates a TSmallPoint (see page 23303) structure from a pair of coordinates.

Pascal

```
function SmallPoint(X: Integer; Y: Integer): TSmallPoint; overload;
function SmallPoint(XY: LongWord): TSmallPoint; overload;
```

C++

```
TSmallPoint SmallPoint(int X, int Y);
TSmallPoint SmallPoint(LongWord XY);
```

File

Types

Description

Call SmallPoint to create a TSmallPoint (see page 23303) that represents the specified coordinates.

See Also

Point (see page 3244)

164.1.11 Types.UnionRect Function

Returns the union of two rectangles

Pascal

```
function UnionRect(out Rect: TRect; const R1: TRect; const R2: TRect): Boolean;
```

C++

```
Boolean UnionRect(TRect Rect, const TRect R1, const TRect R2);
```

File

Types

Description

Call UnionRect to obtain the smallest rectangle that includes both R1 and R2. The union is returned as the Rect (see page 23299) parameter.

UnionRect returns true if the resulting rectangle is nonempty. If the resulting rectangle as a width or height that is less than or equal to 0, UnionRect returns false.

See Also

IntersectRect (see page 23296)

OffsetRect (see page 23297)

IsRectEmpty (see page 23297)

164.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	tagSIZE (see page 23301)	tagSIZE is a record holding width and height values.
	TPoint (see page 23301)	TPoint type defines a pixel location onscreen.
	TRect (see page 23302)	TRect defines a rectangle.
	TSmallPoint (see page 23303)	TSmallPoint type defines a point with two 16-bit coordinates.

164.2.1 Types.tagSIZE Record

tagSIZE is a record holding width and height values.

Pascal

```
tagSIZE = packed record
  cx: Longint;
  cy: Longint;
end;
```

C++

```
struct tagSIZE {
  Longint cx;
  Longint cy;
};
```

File

Types

Description

The SIZE (see page 23308) equates to a TSize (see page 23313) type, which in turn equates to a tagSIZE record.

tagSIZE holds the width (cx) and height (cy) of something, typically in pixels. For example, TextExtent returns the width and height, in pixels, of a string rendered in the current font.

See Also

SIZE (see page 23308)

TextExtent (see page 14474)

TSize (see page 23313)

164.2.2 Types.TPoint Record

TPoint type defines a pixel location onscreen.

Pascal

```
TPoint = packed record
  X: Longint;
  Y: Longint;
end;
```

C++

```
struct TPoint {
  Longint X;
  Longint Y;
};
```

File

Types

Description

The TPoint type defines a pixel location onscreen, with the origin in the top left corner. X specifies the horizontal coordinate of the point, Y specifies the vertical coordinate.

C++ Examples:

```
/*
This example draws a polygon in the specified shape, and
fills it with the color teal.
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
  TPoint points[4];
  points[0] = Point(10,10);
  points[1] = Point(30,10);
  points[2] = Point(130,30);
  points[3] = Point(240,120);
  ((TPaintBox *)Sender)->Canvas->Brush->Color = clTeal;
  ((TPaintBox *)Sender)->Canvas->Polygon(points, 3);
}
```

See Also

Point

164.2.3 Types.TRect Record

TRect defines a rectangle.

Pascal

```
TRect = packed record
  case Integer of
    0: (Bottom: Longint;
        Left: Longint;
        Right: Longint;
        Top: Longint;);
    1: (BottomRight: TPoint;
        TopLeft: TPoint;);
end;
```

C++

```
struct TRect {
  Longint Bottom;
  Longint Left;
```

```
    Longint Right;
    Longint Top;
    TPoint BottomRight;
    TPoint TopLeft;
  };
```

File

Types

Description

TRect represents the dimensions of a rectangle. The coordinates are specified as either four separate integers representing the left, top, right, and bottom sides, or as two points representing the locations of the top left and bottom right corners.

Typically, TRect values represent pixel locations, where the origin of the pixel coordinate system is in the top left corner of the screen (screen coordinates) or the top left corner of a control's client area (client coordinates). When a TRect value represents a rectangle on the screen, by convention the top and left edges are considered inside the rectangle and the bottom and right edges are considered outside the rectangle. This convention allows the width of the rectangle to be Right - Left and the height to be Bottom - Top.

164.2.4 Types.TSmallPoint Record

TSmallPoint type defines a point with two 16-bit coordinates.

Pascal

```
  TSmallPoint = packed record
    x: SmallInt;
    y: SmallInt;
  end;
```

C++

```
  struct TSmallPoint {
    SmallInt x;
    SmallInt y;
  };
```

File

Types

Description

The TSmallPoint type defines a point where each coordinate is a 16-bit integer. x specifies the horizontal coordinate of the point, y specifies the vertical coordinate.

164.3 Types

The following table lists types in this documentation.

Types

Name	Description
DWORD (see page 23304)	DWORD is equivalent to LongWord.
PByte (see page 23305)	PByte is a pointer to a Byte variable.
PDouble (see page 23305)	PDouble is a pointer to a Double variable.
PInteger (see page 23306)	PInteger is a pointer to a Integer variable.
PLongint (see page 23306)	This is type Types.PLongint.
PPoint (see page 23306)	PPoint is a pointer to a TPoint (see page 23301) variable.
PRect (see page 23307)	PRect is a pointer to a TRect (see page 23302) variable.
PSize (see page 23307)	PSize is a pointer to a TSize (see page 23313) variable.
PSmallInt (see page 23308)	This is type Types.PSmallInt.
PSmallPoint (see page 23308)	This is type Types.PSmallPoint.
SIZE (see page 23308)	This is type Types.SIZE.
tagPOINT (see page 23309)	This is type Types.tagPOINT.
TBooleanDynArray (see page 23309)	TBooleanDynArray defines an array of Boolean elements.
TByteDynArray (see page 23309)	This is type Types.TByteDynArray.
TCardinalDynArray (see page 23310)	TCardinalDynArray defines an array of Cardinal elements.
TDoubleDynArray (see page 23310)	TDoubleDynArray defines an array of Double elements.
TInt64DynArray (see page 23311)	TInt64DynArray defines an array of Int64 elements.
TIntegerDynArray (see page 23311)	This is type Types.TIntegerDynArray.
TLongWordDynArray (see page 23311)	TLongWordDynArray defines an array of LongWord elements.
TShortIntDynArray (see page 23312)	TShortIntDynArray defines an array of ShortInt elements.
TSingleDynArray (see page 23312)	TSingleDynArray defines an array of Single elements.
TSize (see page 23313)	TSize represents a width and height.
TSmallIntDynArray (see page 23313)	TSmallIntDynArray defines an array of SmallInt elements.
TStringDynArray (see page 23313)	TStringDynArray defines an array of String elements.
TValueRelationship (see page 23314)	TValueRelationship defines value comparison possibilities.
TWideStringDynArray (see page 23314)	TWideStringDynArray defines an array of WideString elements.
TWordDynArray (see page 23315)	TWordDynArray defines an array of Word elements.

164.3.1 Types.DWORD Type

DWORD is equivalent to LongWord.

Pascal

```
DWORD = LongWord;
```

C++

```
LongWord DWORD;
```

File

Types

Description

The DWORD holds positive integers up to a maximum value of 4,294,967,295.

See Also

LongWord

164.3.2 Types.PByte Type

PByte is a pointer to a Byte variable.

Pascal

```
PByte = System.PByte;
```

C++

```
System.PByte PByte;
```

File

Types

Description

Types.PByte is the same as System.PByte (see page 22588).

See Also

Byte

PByte (see page 22588)

164.3.3 Types.PDouble Type

PDouble is a pointer to a Double variable.

Pascal

```
PDouble = System.PDouble;
```

C++

```
System.PDouble PDouble;
```

File

Types

Description

Types.PDouble is the same as System.PDouble (see page 22592).

See Also

Double

PDouble (see page 22592)

164.3.4 Types.PInteger Type

PInteger is a pointer to a Integer variable.

Pascal

```
PInteger = System.PInteger;
```

C++

```
System.PInteger PInteger;
```

File

Types

Description

Types.PInteger is the same as System.PInteger (see page 22594).

See Also

Integer

PInteger (see page 22594)

164.3.5 Types.PLongint Type

Pascal

```
PLongint = System.PLongint;
```

C++

```
System.PLongint PLongint;
```

File

Types

Description

This is type Types.PLongint.

164.3.6 Types.PPoint Type

PPoint is a pointer to a TPoint (see page 23301) variable.

Pascal

```
PPoint = ^TPoint;
```

C++

```
^TPoint PPoint;
```

File

Types

Description

The TPoint (see page 23301) type defines a pixel location onscreen, with the origin in the top left corner.

See Also

TPoint (see page 23301)

164.3.7 Types.PRect Type

PRect is a pointer to a TRect (see page 23302) variable.

Pascal

```
PRect = ^TRect;
```

C++

```
^TRect PRect;
```

File

Types

Description

The TRect (see page 23302) type represents the dimensions of a rectangle.

See Also

TRect (see page 23302)

164.3.8 Types.PSize Type

PSize is a pointer to a TSize (see page 23313) variable.

Pascal

```
PSize = ^TSize;
```

C++

```
^TSize PSize;
```

File

Types

Description

The TSize (see page 23313) type holds the width and height of something, typically in pixels.

See Also

TSize (see page 23313)

164.3.9 Types.PSmallInt Type

Pascal

```
PSmallInt = System.PSmallInt;
```

C++

```
System.PSmallInt PSmallInt;
```

File

Types

Description

This is type Types.PSmallInt.

164.3.10 Types.PSmallPoint Type

Pascal

```
PSmallPoint = ^TSmallPoint;
```

C++

```
^TSmallPoint PSmallPoint;
```

File

Types

Description

This is type Types.PSmallPoint.

164.3.11 Types.SIZE Type

Pascal

```
SIZE = tagSIZE;
```

C++

```
tagSIZE SIZE;
```

File

Types

Description

This is type Types.SIZE.

164.3.12 Types.tagPOINT Type

Pascal

```
tagPOINT = TPoint;
```

C++

```
TPoint tagPOINT;
```

File

Types

Description

This is type Types.tagPOINT.

164.3.13 Types.TBooleanDynArray Type

TBooleanDynArray defines an array of Boolean elements.

Pascal

```
TBooleanDynArray = array of Boolean;
```

C++

```
array of Boolean TBooleanDynArray;
```

File

Types

Description

The TBooleanDynArray defines a dynamic array of Boolean elements.

Use SetLength to allocate storage for such an array.

See Also

TSize (see page 23313)

164.3.14 Types.TByteDynArray Type

Pascal

```
TByteDynArray = array of Byte;
```

C++

```
array of Byte TByteDynArray;
```

File

Types

Description

This is type Types.TByteDynArray.

164.3.15 Types.TCardinalDynArray Type

TCardinalDynArray defines an array of Cardinal elements.

Pascal

```
TCardinalDynArray = array of Cardinal;
```

C++

```
array of Cardinal TCardinalDynArray;
```

File

Types

Description

The TCardinalDynArray defines a dynamic array of Cardinal elements.

Use SetLength to allocate storage for such an array.

See Also

TSize (see page 23313)

164.3.16 Types.TDoubleDynArray Type

TDoubleDynArray defines an array of Double elements.

Pascal

```
TDoubleDynArray = array of Double;
```

C++

```
array of Double TDoubleDynArray;
```

File

Types

Description

The TDoubleDynArray defines a dynamic array of Double elements.

Use SetLength to allocate storage for such an array.

See Also

TSize (see page 23313)

164.3.17 Types.TInt64DynArray Type

TInt64DynArray defines an array of Int64 elements.

Pascal

```
TInt64DynArray = array of Int64;
```

C++

```
array of Int64 TInt64DynArray;
```

File

Types

Description

The TInt64DynArray defines a dynamic array of Int64 elements.

Use SetLength to allocate storage for such an array.

See Also

TSize (see page 23313)

164.3.18 Types.TIntegerDynArray Type

Pascal

```
TIntegerDynArray = array of Integer;
```

C++

```
array of Integer TIntegerDynArray;
```

File

Types

Description

This is type Types.TIntegerDynArray.

164.3.19 Types.TLongWordDynArray Type

TLongWordDynArray defines an array of LongWord elements.

Pascal

```
TLongWordDynArray = array of LongWord;
```

C++

```
array of LongWord TLongWordDynArray;
```

File

Types

Description

The TLongWordDynArray defines a dynamic array of LongWord elements.

Use SetLength to allocate storage for such an array.

See Also

TSize (see page 23313)

164.3.20 Types.TShortIntDynArray Type

TShortIntDynArray defines an array of ShortInt elements.

Pascal

```
TShortIntDynArray = array of ShortInt;
```

C++

```
array of ShortInt TShortIntDynArray;
```

File

Types

Description

The TShortIntDynArray defines a dynamic array of ShortInt elements.

Use SetLength to allocate storage for such an array.

See Also

TSize (see page 23313)

164.3.21 Types.TSingleDynArray Type

TSingleDynArray defines an array of Single elements.

Pascal

```
TSingleDynArray = array of Single;
```

C++

```
array of Single TSingleDynArray;
```

File

Types

Description

The TSingleDynArray defines a dynamic array of Single elements.

Use SetLength to allocate storage for such an array.

See Also

TSize (see page 23313)

164.3.22 Types.TSize Type

TSize represents a width and height.

Pascal

```
TSize = tagSIZE;
```

C++

```
tagSIZE TSize;
```

File

Types

Description

TSize represents the size of something. Typically, the implied units of measurement are pixels.

cx is the width.

cy is the height.

164.3.23 Types.TSmallIntDynArray Type

TSmallIntDynArray defines an array of SmallInt elements.

Pascal

```
TSmallIntDynArray = array of SmallInt;
```

C++

```
array of SmallInt TSmallIntDynArray;
```

File

Types

Description

The TSmallIntDynArray defines a dynamic array of SmallInt elements.

Use SetLength to allocate storage for such an array.

See Also

TSize (see page 23313)

164.3.24 Types.TStringDynArray Type

TStringDynArray defines an array of String elements.

Pascal

```
TStringDynArray = array of string;
```

C++

```
array of string TStringDynArray;
```

File

Types

Description

The TStringDynArray defines a dynamic array of String elements.

Use SetLength to allocate storage for such an array.

See Also

TSize (see page 23313)

164.3.25 Types.TValueRelationship Type

TValueRelationship defines value comparison possibilities.

Pascal

```
TValueRelationship = -1..1;
```

C++

```
-1..1 TValueRelationship;
```

File

Types

Description

Routines such as the CompareValue routine compare two values for equality. They return the result in a TValueRelationship value as follows :

Value	Associated constant	Meaning
-1	LessThanValue (see page 23317)	The first value is less than the second value.
0	EqualsValue (see page 23316)	The two values are equal.
+1	GreaterThanValue (see page 23316)	The first value is greater than the second value.

See Also

LessThanValue (see page 23317)

EqualsValue (see page 23316)

GreaterThanValue (see page 23316)

TValueRelationship

164.3.26 Types.TWideStringDynArray Type

TWideStringDynArray defines an array of WideString elements.

Pascal

```
TWideStringDynArray = array of WideString;
```

C++

```
array of WideString TWideStringDynArray;
```

File

Types

Description

The TWideStringDynArray defines a dynamic array of WideString elements.

Use SetLength to allocate storage for such an array.

See Also

TSize (see page 23313)

164.3.27 Types.TWordDynArray Type

TWordDynArray defines an array of Word elements.

Pascal

```
TWordDynArray = array of Word;
```

C++

```
array of Word TWordDynArray;
```

File

Types

Description

The TWordDynArray defines a dynamic array of Word elements.

Use SetLength to allocate storage for such an array.

See Also

TSize (see page 23313)

164.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
EqualsValue (see page 23316)	EqualsValue indicates that both values are equal in a comparison operation.
GreaterThanValue (see page 23316)	GreaterThanValue indicates that one value is greater than an other in a comparison operation.
LessThanValue (see page 23317)	LessThanValue indicates that one value is less than an other in a comparison operation.
RT_RCDATA (see page 23317)	This is constant Types.RT_RCDATA.

164.4.1 Types.EqualsValue Constant

EqualsValue indicates that both values are equal in a comparison operation.

Pascal

```
EqualsValue = 0;
```

C++

```
EqualsValue = 0;
```

File

Types

Description

Routines such as the CompareValue routine compare two values for equality. They return the result in a TValueRelationship (see page 23314) value. If the first value is equal to the second, the value is EqualsValue

See Also

CompareValue (see page 15938)

GreaterThanValue

LessThanValue

TValueRelationship

164.4.2 Types.GreaterThanValue Constant

GreaterThanValue indicates that one value is greater than an other in a comparison operation.

Pascal

```
GreaterThanValue = High(TValueRelationship);
```

C++

```
GreaterThanValue = High(TValueRelationship);
```

File

Types

Description

Routines such as the CompareValue routine compare two values for equality. They return the result in a TValueRelationship (see page 23314) value. If the first value is greater than the second, the value is GreaterThanValue

See Also

EqualsValue

CompareValue (see page 15938)

LessThanValue

TValueRelationship

164.4.3 Types.LessThanValue Constant

LessThanValue indicates that one value is less than an other in a comparison operation.

Pascal

```
LessThanValue = Low(TValueRelationship);
```

C++

```
LessThanValue = Low(TValueRelationship);
```

File

Types

Description

Routines such as the CompareValue routine compare two values for equality. They return the result in a TValueRelationship (see page 23314) value. If the first value is less than the second, the value is LessThanValue

See Also

EqualsValue

CompareValue (see page 15938)

GreaterThanValue

TValueRelationship

164.4.4 Types.RT_RCDATA Constant

Pascal

```
RT_RCDATA = PChar(10);
```

C++

```
RT_RCDATA = PChar(10);
```

File

Types

Description

This is constant Types.RT_RCDATA.

165 TypeTrans Namespace

This is namespace TypeTrans.

Topics

Name	Description
Classes (see page 23320)	The following table lists classes in this documentation.
Functions (see page 23324)	The following table lists functions in this documentation.
Variables (see page 23326)	The following table lists variables in this documentation.

Classes

	Name	Description
	ETypeTransException (see page 23320)	ETypeTransException is the exception raised when a conversion between a SOAP type description and an Object Pascal type fails.
	TTypeTranslator (see page 23320)	This is class TypeTrans.TTypeTranslator.

Functions

	Name	Description
	FloatToStrEx (see page 23324)	This is function TypeTrans.FloatToStrEx.
	GetEnumValueEx (see page 23324)	This is function TypeTrans.GetEnumValueEx.
	SetEnumPropEx (see page 23325)	This is function TypeTrans.SetEnumPropEx.
	StrToFloatEx (see page 23325)	This is function TypeTrans.StrToFloatEx.

Variables

Name	Description
TypeTranslator (see page 23326)	This is variable TypeTrans.TypeTranslator.

165.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ETypeTransException (see page 23320)	ETypeTransException is the exception raised when a conversion between a SOAP type description and an Object Pascal type fails.
	TTypeTranslator (see page 23320)	This is class TypeTrans.TTypeTranslator.

165.1.1 ETypeTransException Class

ETypeTransException is the exception raised when a conversion between a SOAP type description and an Object Pascal type fails.

Class Hierarchy

Pascal

```
ETypeTransException = class(Exception);
```

C++

```
class ETypeTransException : public Exception;
```

File

TypeTrans

Description

ETypeTransException is thrown when an application attempts to convert a type in a SOAP packet to an Object Pascal type and fails for some reason.

See Also

TOPToSoapDomConvert (see page 18137)

165.1.2 TTypeTranslator Class

Class Hierarchy

Pascal

```
TTypeTranslator = class;
```

C++

```
class TTypeTranslator;
```

File

TypeTrans

Description

This is class TypeTrans.TTypeTranslator.

165.1.2.1 TTypeTranslator Members

The following tables list the members exposed by TTypeTranslator.

Public Methods

	Name	Description
	Base64ToVar (see page 23322)	This is the overview for the Base64ToVar method overload.
	CastNativeToSoap (see page 23322)	This is CastNativeToSoap, a member of class TTypeTranslator.
	CastSoapToNative (see page 23323)	This is CastSoapToNative, a member of class TTypeTranslator.
	CastSoapToVariant (see page 23323)	This is the overview for the CastSoapToVariant method overload.
	Create (see page 23321)	This is Create, a member of class TTypeTranslator.
V	Destroy (see page 23321)	This is Destroy, a member of class TTypeTranslator.

165.1.2.2 TTypeTranslator.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TTypeTranslator();
```

Description

This is Create, a member of class TTypeTranslator.

165.1.2.3 TTypeTranslator.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTypeTranslator();
```

Description

This is Destroy, a member of class TTypeTranslator.

165.1.2.4 TTypeTranslator Methods

The methods of the TTypeTranslator class are listed here.

Public Methods

	Name	Description
	Base64ToVar (see page 23322)	This is the overview for the Base64ToVar method overload.

	CastNativeToSoap (see page 23322)	This is CastNativeToSoap, a member of class TTypeTranslator.
	CastSoapToNative (see page 23323)	This is CastSoapToNative, a member of class TTypeTranslator.
	CastSoapToVariant (see page 23323)	This is the overview for the CastSoapToVariant method overload.

165.1.2.4.1 Base64ToVar Method

This is the overview for the Base64ToVar method overload.

Overload List

	Name	Description
	TTypeTranslator.Base64ToVar (Pointer, WideString) (see page 23322)	This is Base64ToVar, a member of class TTypeTranslator.
	TTypeTranslator.Base64ToVar (Variant, WideString) (see page 23322)	This is Base64ToVar, a member of class TTypeTranslator.

165.1.2.4.1.1 TTypeTranslator.Base64ToVar Method (Pointer, WideString)

Pascal

```
procedure Base64ToVar(NatData: Pointer; const SoapData: WideString); overload;
```

C++

```
__fastcall Base64ToVar(void * NatData, const BSTR SoapData);
```

Description

This is Base64ToVar, a member of class TTypeTranslator.

165.1.2.4.1.2 TTypeTranslator.Base64ToVar Method (Variant, WideString)

Pascal

```
procedure Base64ToVar(var V: Variant; const SoapData: WideString); overload;
```

C++

```
__fastcall Base64ToVar(Variant V, const BSTR SoapData);
```

Description

This is Base64ToVar, a member of class TTypeTranslator.

165.1.2.4.2 TTypeTranslator.CastNativeToSoap Method

Pascal

```
procedure CastNativeToSoap(Info: PTypeInfo; var SoapData: WideString; NatData: Pointer; var
IsNull: Boolean);
```

C++

```
__fastcall CastNativeToSoap(PTypeInfo Info, BSTR SoapData, void * NatData, Boolean IsNull);
```

Description

This is CastNativeToSoap, a member of class TTypeTranslator.

165.1.2.4.3 TTypeTranslator.CastSoapToNative Method

Pascal

```
function CastSoapToNative(Info: PTypeInfo; const SoapData: WideString; NatData: Pointer;
IsNull: Boolean): Boolean;
```

C++

```
__fastcall Boolean CastSoapToNative(PTypeInfo Info, const BSTR SoapData, void * NatData,
Boolean IsNull);
```

Description

This is CastSoapToNative, a member of class TTypeTranslator.

165.1.2.4.4 CastSoapToVariant Method

This is the overview for the CastSoapToVariant method overload.

Overload List

	Name	Description
	TTypeTranslator.CastSoapToVariant (PTypeInfo, WideString) (see page 23323)	This is CastSoapToVariant, a member of class TTypeTranslator.
	TTypeTranslator.CastSoapToVariant (PTypeInfo, WideString, Pointer) (see page 23323)	This is CastSoapToVariant, a member of class TTypeTranslator.

165.1.2.4.4.1 TTypeTranslator.CastSoapToVariant Method (PTypeInfo, WideString)

Pascal

```
function CastSoapToVariant(SoapInfo: PTypeInfo; const SoapData: WideString): Variant; overload;
```

C++

```
__fastcall Variant CastSoapToVariant(PTypeInfo SoapInfo, const BSTR SoapData);
```

Description

This is CastSoapToVariant, a member of class TTypeTranslator.

165.1.2.4.4.2 TTypeTranslator.CastSoapToVariant Method (PTypeInfo, WideString, Pointer)

Pascal

```
procedure CastSoapToVariant(SoapInfo: PTypeInfo; const SoapData: WideString; NatData:
Pointer); overload;
```

C++

```
__fastcall CastSoapToVariant(PTypeInfo SoapInfo, const BSTR SoapData, void * NatData);
```

Description

This is CastSoapToVariant, a member of class TTypeTranslator.

165.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FloatToStrEx (see page 23324)	This is function TypeTrans.FloatToStrEx.
	GetEnumValueEx (see page 23324)	This is function TypeTrans.GetEnumValueEx.
	SetEnumPropEx (see page 23325)	This is function TypeTrans.SetEnumPropEx.
	StrToFloatEx (see page 23325)	This is function TypeTrans.StrToFloatEx.

165.2.1 TypeTrans.FloatToStrEx Function

Pascal

```
function FloatToStrEx(Value: Extended): string;
```

C++

```
AnsiString FloatToStrEx(Extended Value);
```

File

TypeTrans

Description

This is function TypeTrans.FloatToStrEx.

165.2.2 TypeTrans.GetEnumValueEx Function

Pascal

```
function GetEnumValueEx(TypInfo: PTypeInfo; const Name: string): Integer;
```

C++

```
int GetEnumValueEx(PTypeInfo TypInfo, const AnsiString Name);
```

File

TypeTrans

Description

This is function TypeTrans.GetEnumValueEx.

165.2.3 TypeTrans.SetEnumPropEx Function

Pascal

```
procedure SetEnumPropEx(Instance: TObject; PropInfo: PPropInfo; const Value: string);
```

C++

```
SetEnumPropEx(TObject * Instance, PPropInfo PropInfo, const AnsiString Value);
```

File

TypeTrans

Description

This is function TypeTrans.SetEnumPropEx.

165.2.4 TypeTrans.StrToFloatEx Function

Pascal

```
function StrToFloatEx(const S: string): Extended;
```

C++

```
Extended StrToFloatEx(const AnsiString S);
```

File

TypeTrans

Description

This is function TypeTrans.StrToFloatEx.

165.3 Variables

The following table lists variables in this documentation.

Variables

Name	Description
TypeTranslator (see page 23326)	This is variable TypeTrans.TypeTranslator.

165.3.1 TypeTrans.TypeTranslator Variable

Pascal

```
TypeTranslator: TTypeTranslator;
```

C++

```
TTypeTranslator TypeTranslator;
```

File

TypeTrans

Description

This is variable TypeTrans.TypeTranslator.

166 TypInfo Namespace

This is namespace TypInfo.

Topics

Name	Description
Classes (see page 23331)	The following table lists classes in this documentation.
Functions (see page 23338)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 23372)	The following table lists structs, records, enums in this documentation.
Types (see page 23381)	The following table lists types in this documentation.
Variables (see page 23388)	The following table lists variables in this documentation.
Constants (see page 23389)	The following table lists constants in this documentation.

Classes

	Name	Description
	EPropertyConvertError (see page 23331)	EPropertyConvertError is the exception class for type errors when attempting to get or set the value of a property.
	EPropertyError (see page 23331)	EPropertyError is the exception class for unsuccessful attempts to set the value of a property.
	TPublishableVariantType (see page 23332)	TPublishableVariantType is the base class for custom Variant types that surface the published properties of an object that represents the Variant's data.

Constants

Name	Description
tkAny (see page 23389)	This is constant TypInfo.tkAny.
tkMethods (see page 23389)	This is constant TypInfo.tkMethods.
tkProperties (see page 23389)	This is constant TypInfo.tkProperties.

Functions

	Name	Description
	FreeAndNilProperties (see page 23339)	Frees any property values that are objects and sets their value to nil (Delphi) or NULL (C++).
	GetDynArrayProp (see page 23340)	Returns the value of a component property that is a dynamic array type.
	GetDynArrayProp (see page 23340)	Returns the value of a component property that is a dynamic array type.
	GetEnumName (see page 23341)	Returns the name of a Delphi enumerated type constant given its value.
	GetEnumProp (see page 23341)	Returns the value of a component property that is an enumerated type.
	GetEnumProp (see page 23341)	Returns the value of a component property that is an enumerated type.
	GetEnumValue (see page 23342)	Returns the value of an enumerated type constant given its string representation.

	GetFloatProp (see page 23343)	Returns the value of a component property that is a float type.
	GetFloatProp (see page 23343)	Returns the value of a component property that is a float type.
	GetInt64Prop (see page 23343)	Returns the value of a component property that is an Int64 type.
	GetInt64Prop (see page 23343)	Returns the value of a component property that is an Int64 type.
	GetInterfaceProp (see page 23344)	Returns the value of a component property that is an interface type.
	GetInterfaceProp (see page 23344)	Returns the value of a component property that is an interface type.
	GetMethodProp (see page 23345)	Returns the value of a component property that is a method type.
	GetMethodProp (see page 23345)	Returns the value of a component property that is a method type.
	GetObjectProp (see page 23346)	Returns the value of a component property that is an object type.
	GetObjectProp (see page 23346)	Returns the value of a component property that is an object type.
	GetObjectPropClass (see page 23347)	Returns the class type of a component property that is an object type.
	GetObjectPropClass (see page 23347)	Returns the class type of a component property that is an object type.
	GetObjectPropClass (see page 23347)	Returns the class type of a component property that is an object type.
	GetOrdProp (see page 23347)	Returns the value of a component property that is an ordinal type.
	GetOrdProp (see page 23347)	Returns the value of a component property that is an ordinal type.
	GetPropInfo (see page 23348)	Retrieves information about a component's property.
	GetPropInfo (see page 23348)	Retrieves information about a component's property.
	GetPropInfo (see page 23348)	Retrieves information about a component's property.
	GetPropInfo (see page 23348)	Retrieves information about a component's property.
	GetPropInfos (see page 23349)	Retrieves a list of component properties.
	GetPropList (see page 23350)	Retrieves a list of component properties.
	GetPropList (see page 23350)	Retrieves a list of component properties.
	GetPropList (see page 23350)	Retrieves a list of component properties.
	GetPropValue (see page 23351)	Retrieves a component's property value as a variant.
	GetPropValue (see page 23351)	Retrieves a component's property value as a variant.
	GetSetElementName (see page 23351)	This is function TypInfo.GetSetElementName.
	GetSetElementValue (see page 23352)	This is function TypInfo.GetSetElementValue.
	GetSetProp (see page 23352)	Returns the value of a component property that is a set type.
	GetSetProp (see page 23352)	Returns the value of a component property that is a set type.
	GetStrProp (see page 23353)	Returns the value of a component property that is a string type.
	GetStrProp (see page 23353)	Returns the value of a component property that is a string type.
	GetTypeData (see page 23354)	Returns a pointer to information about a type.
	GetVariantProp (see page 23354)	Returns the value of a component property that is a variant type.
	GetVariantProp (see page 23354)	Returns the value of a component property that is a variant type.
	GetWideStrProp (see page 23355)	Returns the value of a component property that is a WideString type.
	GetWideStrProp (see page 23355)	Returns the value of a component property that is a WideString type.
	IsPublishedProp (see page 23356)	Determines if a specified component property is a published property.
	IsPublishedProp (see page 23356)	Determines if a specified component property is a published property.
	IsStoredProp (see page 23356)	Determines if a specified component property has the stored attribute.
	IsStoredProp (see page 23356)	Determines if a specified component property has the stored attribute.
	PropIsType (see page 23357)	Determines if a component property is of a particular data type.
	PropIsType (see page 23357)	Determines if a component property is of a particular data type.
	PropType (see page 23358)	Retrieves a component's property type.
	PropType (see page 23358)	Retrieves a component's property type.
	SetDynArrayProp (see page 23358)	Sets the value of a component property that is a dynamic array type.
	SetDynArrayProp (see page 23358)	Sets the value of a component property that is a dynamic array type.
	SetEnumProp (see page 23359)	Sets the value of a component property that is an enumerated type.
	SetEnumProp (see page 23359)	Sets the value of a component property that is an enumerated type.
	SetFloatProp (see page 23360)	Sets the value of a component property that is a float type.
	SetFloatProp (see page 23360)	Sets the value of a component property that is a float type.
	SetInt64Prop (see page 23361)	Sets the value of a component property that is an Int64 type.
	SetInt64Prop (see page 23361)	Sets the value of a component property that is an Int64 type.
	SetInterfaceProp (see page 23362)	Sets the value of a component property that is an Interface type.
	SetInterfaceProp (see page 23362)	Sets the value of a component property that is an Interface type.

	SetMethodProp (see page 23362)	Sets the value of a component property that is a method type.
	SetMethodProp (see page 23362)	Sets the value of a component property that is a method type.
	SetObjectProp (see page 23363)	Sets the value of a component property that is an object type.
	SetObjectProp (see page 23363)	Sets the value of a component property that is an object type.
	SetOrdProp (see page 23364)	Sets the value of a component property that is an ordinal type.
	SetOrdProp (see page 23364)	Sets the value of a component property that is an ordinal type.
	SetPropValue (see page 23365)	Sets the value of a component property using a variant value.
	SetPropValue (see page 23365)	Sets the value of a component property using a variant value.
	SetSetProp (see page 23366)	Sets the value of a component property that is a set type.
	SetSetProp (see page 23366)	Sets the value of a component property that is a set type.
	SetStrProp (see page 23367)	Sets the value of a component property that is a string type.
	SetStrProp (see page 23367)	Sets the value of a component property that is a string type.
	SetToString (see page 23367)	Returns one or more of the set values of a component's property.
	SetVariantProp (see page 23368)	Sets the value of a component property that is a variant type.
	SetVariantProp (see page 23368)	Sets the value of a component property that is a variant type.
	SetWideStrProp (see page 23369)	Sets the value of a component property that is a WideString type.
	SetWideStrProp (see page 23369)	Sets the value of a component property that is a WideString type.
	SortPropList (see page 23370)	Sorts a list of component properties into alphanumeric sequence.
	StringToSet (see page 23370)	Returns a bit mask of set values for a set type component property.

Structs, Records, Enums

	Name	Description
	TFloatType (see page 23372)	This is record TypInfo.TFloatType.
	TIntfFlag (see page 23373)	This is record TypInfo.TIntfFlag.
	TMethodKind (see page 23373)	Enumerates floating point number types in the TTypeData (see page 23377) record.
	TOrdType (see page 23374)	Enumerates floating point number types in the TTypeData (see page 23377) record.
	TParamFlag (see page 23375)	TParamFlag and TParamFlags (see page 23386) indicates the attributes of a method parameter.
	TPropData (see page 23376)	This is record TypInfo.TPropData.
	TPropInfo (see page 23376)	TPropInfo is a record of component property type information.
	TTypeData (see page 23377)	TTypeData is used internally to represent type information.
	TTypeInfo (see page 23379)	Holds information for a data type as returned by TypeInfo.
	TTypeKind (see page 23379)	These types describe a type that is manipulated in the forms designer.

Types

Name	Description
PPropInfo (see page 23381)	PPropInfo points to a TPropInfo (see page 23376) record that holds component property information.
PPropList (see page 23382)	Defines a pointer to a TPropList (see page 23387) array.
PPTypeInfo (see page 23382)	Holds a pointer to a pointer to information for a data type as returned by TypeInfo.
PTypeData (see page 23383)	PTypeData points to a TTypeData (see page 23377) record that holds component property type information.
PTypeInfo (see page 23383)	Holds a pointer to information for a data type as returned by TypeInfo.
ShortStringBase (see page 23384)	Defines a fixed length string for general use in the TTypeData (see page 23377) record.
TIntfFlags (see page 23384)	Enumerates interface flag types in the TTypeData (see page 23377) record.
TIntfFlagsBase (see page 23385)	Set of interface flag types in the TTypeData (see page 23377) record.
TParamFlags (see page 23386)	This is type TypInfo.TParamFlags.
TParamFlagsBase (see page 23386)	This is type TypInfo.TParamFlagsBase.
TPropInfoProc (see page 23386)	This is type TypInfo.TPropInfoProc.
TPropList (see page 23387)	Defines an array of PPropInfo (see page 23381) values.
TTypeKinds (see page 23387)	Defines a set of TTypeKind (see page 23379) values.

Variables

Name	Description
BooleanIdents (see page 23388)	An array of string representations of true and false.
DotSep (see page 23388)	This is variable TypInfo.DotSep.

166.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EPropertyConvertError (see page 23331)	EPropertyConvertError is the exception class for type errors when attempting to get or set the value of a property.
	EPropertyError (see page 23331)	EPropertyError is the exception class for unsuccessful attempts to set the value of a property.
	TPublishableVariantType (see page 23332)	TPublishableVariantType is the base class for custom Variant types that surface the published properties of an object that represents the Variant's data.

166.1.1 EPropertyConvertError Class

EPropertyConvertError is the exception class for type errors when attempting to get or set the value of a property.

Class Hierarchy

Pascal

```
EPropertyConvertError = class(Exception);
```

C++

```
class EPropertyConvertError : public Exception;
```

File

TypInfo

Description

EPropertyConvertError is raised when an application tries to set a property to a value of the wrong type or to read a property that has invalid type information.

See Also

EPropertyError (see page 23331)

TPropertyEditor (see page 11019)

166.1.2 EPropertyError Class

EPropertyError is the exception class for unsuccessful attempts to set the value of a property.

Class Hierarchy

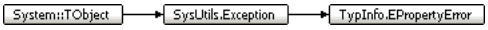

Pascal

```
EPropertyError = class(Exception);
```

C++

```
class EPropertyError : public Exception;
```

File

TypInfo

Description

EPropertyError is raised by the SetValue (or Edit) methods of property-editor objects when they cannot set a requested property value. This usually occurs because the requested value is out of range or of an invalid data type.

See Also

TPropertyEditor (see page 11019)

TPropertyEditor (see page 11019)

166.1.3 TPublishableVariantType Class

TPublishableVariantType is the base class for custom Variant types that surface the published properties of an object that represents the Variant's data.

Class Hierarchy

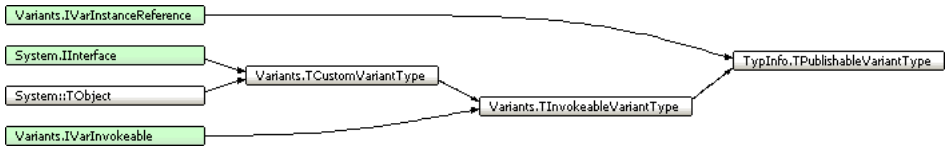

Pascal

```
TPublishableVariantType = class(TInvokeableVariantType, IVarInstanceReference);
```

C++

```
class TPublishableVariantType : public TInvokeableVariantType, public IVarInstanceReference;
```

File

TypInfo

Description

TPublishableVariantType is the base class for new data types for Variants (see page 23567) that have properties and methods. TPublishableVariantType descendants do not represent the Variants (see page 23567) themselves. Rather, they act as utility classes that the custom Variants (see page 23567) call on to perform their operations. TPublishableVariantType adds to its ancestor's ability to implement properties and methods on Variants (see page 23567) of the custom type by automatically implementing all published properties of the object it uses to store its data.

Note: To create a Variant type that descends from TPublishableVariantType,

Note: 1. Map the storage of the Variant's data on to the TVarData record.

Note: 2. Write a method that creates instances of your custom Variant. This fills in the Variant's data as defined in step 1.

Note: 3. Declare a class that descends from TPublishableVariantType.

Note: 4. Override the GetInstance (see page 23336) method to return the object that stores the custom Variant's data.

Note: 5. Implement all other required behavior (including type conversion rules) in the new class. This involves overriding the appropriate virtual methods (typically DoFunction, DoProcedure, BinaryOp, UnaryOp, CompareOp or Compare, Cast, CastTo, Clear, Copy, IsClear, LeftPromotion, and RightPromotion).

Note: 6. In the initialization section of the unit that defines your TPublishableVariantType descendant, create an instance of your class. When you instantiate your object, it automatically registers itself with the Variant-handling system so that the new Variant type is enabled.

Note: 7. In the finalization section of the unit that defines your TPublishableVariantType descendant, free the instance of your class. This automatically unregisters the Variant type.

Note: For an example, see the VarCmplx (see page 23513) (Windows) or FMTBcd (see page 13627) (Linux) unit, which implements a custom Variant type for complex (Windows) or binary-coded decimal (Linux) numbers.

Note: Do not create your own custom Variant types in C++. They are only supported in Delphi.

See Also

IDH

TInvokeableVariantType (see page 23606)

Supporting Properties and Methods in Custom Variants

166.1.3.1 TPublishableVariantType Members

The following tables list the members exposed by TPublishableVariantType.

Protected Methods

	Name	Description
	_AddRef (see page 23586)	Increments the reference count for this interface.
	_Release (see page 23587)	Decrements the reference count for this interface.
V	DispInvoke (see page 23592)	Invokes a method call when the Variant supports such operations.
V	DispInvoke (see page 23610)	Invokes a method call by dispatching it to the appropriate TInvokeableVariantType (see page 23606) method.
V	FixupIdent (see page 23612)	Fixes up dispatch identifier.
A	GetInstance (see page 23336)	Returns the object instance that represents the data from a Variant of the custom type.
V	LeftPromotion (see page 23593)	Indicates whether the left argument to a binary operation should be coerced to a different type.

	OlePromotion (see page 23594)	Indicates the type to which a Variant value must be cast to make it Automation-compatible.
	QueryInterface (see page 23594)	Returns a reference to a specified interface if the object supports that interface.
	RaiseCastError (see page 23595)	Generates a system error that indicates an invalid cast.
	RaiseDispError (see page 23595)	Generates a system error indicating an invalid dispatch attempt.
	RaiseInvalidOp (see page 23596)	Generates a system error indicating an invalid operation.
	RightPromotion (see page 23596)	Indicates whether the right argument to a binary operation should be coerced to a different type.
	SimplisticClear (see page 23597)	Clears a TVarData record by re-initializing it.
	SimplisticCopy (see page 23598)	Performs a simple block copy of one TVarData record to another.
	VarDataCast (see page 23599)	Performs a cast from one Variant type to another.
	VarDataCastTo (see page 23599)	This is the overview for the VarDataCastTo method overload.
	VarDataCastToOleStr (see page 23600)	Performs a cast to a Variant type of varOleStr.
	VarDataClear (see page 23600)	Clears the data from a built-in Variant type.
	VarDataCopy (see page 23601)	Copies the data from one built-in Variant type to another.
	VarDataCopyNoInd (see page 23601)	Copies the data from one built-in Variant type to another.
	VarDataFromOleStr (see page 23602)	Creates a TVarData record to represent a Unicode string.
	VarDataFromStr (see page 23602)	Creates a TVarData record to represent a string.
	VarDataInit (see page 23602)	Initializes a TVarData record.
	VarDataIsArray (see page 23603)	Indicates whether a TVarData record represents the data from a Variant array.
	VarDataIsByRef (see page 23603)	Indicates whether a TVarData record stores its data indirectly.
	VarDataIsEmptyParam (see page 23603)	Indicates whether a TVarData record represents an unassigned optional parameter.
	VarDataIsFloat (see page 23604)	Indicates whether a TVarData record represents a floating-point value.
	VarDataIsNumeric (see page 23604)	Indicates whether a TVarData record represents a numeric value.
	VarDataIsOrdinal (see page 23605)	Indicates whether a TVarData record represents a ordinal value.
	VarDataIsStr (see page 23605)	Indicates whether a TVarData record represents a string (normal or Unicode).
	VarDataToStr (see page 23605)	Returns a string that represents the value from a TVarData record.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BinaryOp (see page 23587)	Implements any binary operations of which the custom Variant type is capable.
	Cast (see page 23588)	Converts the data from another Variant type so that it represents this custom type's data format.
	CastTo (see page 23589)	This is CastTo, a member of class TCustomVariantType.
	CastToOle (see page 23589)	This is CastToOle, a member of class TCustomVariantType.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 23589)	Sets the data for this custom Variant type to a blank or unassigned value.
	Compare (see page 23590)	Implements comparisons that involve the custom Variant type.
	CompareOp (see page 23590)	Implements any comparison operations of which the custom Variant type is capable.
	Copy (see page 23591)	Copies the data from a Variant of this custom type.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 23584)	This is the overview for the Create constructor overload.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.

	Destroy (see page 23585)	Frees an instance of TCustomVariantType (see page 23581).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoFunction (see page 23616)	This is DoFunction, a member of class IVarInvokeable.
	DoFunction (see page 23610)	Implements all methods that return values.
	DoProcedure (see page 23616)	This is DoProcedure, a member of class IVarInvokeable.
	DoProcedure (see page 23611)	Implements all methods that do not return values.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetProperty (see page 23336)	Returns the value of a published property on the object that GetInstance (see page 23336) returns, so that it is available in custom Variants (see page 23567).
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsClear (see page 23592)	Indicates whether the data from a custom Variant represents a blank or unassigned value.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetProperty (see page 23337)	Sets the value of a published property on the object that GetInstance (see page 23336) returns, so that it is available in custom Variants (see page 23567).
	UnaryOp (see page 23598)	Implements any unary operations of which the custom Variant type is capable.

Public Properties

	Name	Description
	VarType (see page 23606)	Indicates the Variant type code for the custom Variant type.

166.1.3.2 TPublishableVariantType Methods

The methods of the TPublishableVariantType class are listed here.

Protected Methods

	Name	Description
	GetInstance (see page 23336)	Returns the object instance that represents the data from a Variant of the custom type.

Public Methods

	Name	Description
	GetProperty (see page 23336)	Returns the value of a published property on the object that GetInstance (see page 23336) returns, so that it is available in custom Variants (see page 23567).
	SetProperty (see page 23337)	Sets the value of a published property on the object that GetInstance (see page 23336) returns, so that it is available in custom Variants (see page 23567).

166.1.3.2.1 TPublishableVariantType.GetInstance Method

Returns the object instance that represents the data from a Variant of the custom type.

Pascal

```
function GetInstance(const V: TVarData): TObject; virtual; abstract;
```

C++

```
virtual __fastcall TObject * GetInstance(const TVarData V) = 0;
```

Description

Override GetInstance to return the object that represents the data from a Variant of the custom Type. TPublishableVariantType (see page 23332) implements all the published properties of that object so that they are published properties of the Variant.

V is the data from a Variant of the custom type. This data must include a reference to an object that represents the Variant's data, and which GetInstance returns.

GetInstance is abstract or, in C++ terminology, pure virtual in TPublishableVariantType (see page 23332), which means it has no implementation. Descendant classes override this method to enable Variants (see page 23567) to publish the properties from the object that implements their data.

See Also

GetProperty (see page 23336)

SetProperty (see page 23337)

Supporting Properties and Methods in Custom Variants

166.1.3.2.2 TPublishableVariantType.GetProperty Method

Returns the value of a published property on the object that GetInstance (see page 23336) returns, so that it is available in custom Variants (see page 23567).

Pascal

```
function GetProperty(var Dest: TVarData; const V: TVarData; const Name: string): Boolean;
override;
```

C++

```
virtual __fastcall Boolean GetProperty(TVarData Dest, const TVarData V, const AnsiString Name);
```

Description

GetProperty enables an application to read the properties of instances of the publishable Variant type. It implements all the properties that are published by the object that GetInstance (see page 23336) returns.

When the application contains a line such as

```
PropValue := MyVariant.SpecialProperty;
```

(Delphi) or calls the Exec method of the Variant (C++), a call is automatically generated to GetProperty. If SpecialProperty is a published property on the object that implements the Variant's data, GetProperty fills in the Dest parameter with the property value.

Dest is a TVarData record (Delphi) or structure (C++) that receives the property value.

V is the TVarData record or structure from the Variant instance whose property is being read.

Name is the name of the property.

GetProperty returns true if it returns a property value, and raises an exception if it does not implement the specified property.

See Also

DoFunction (see page 23610)

DoProcedure (see page 23611)

GetInstance (see page 23336)

Exec

166.1.3.2.3 TPublishableVariantType.SetProperty Method

Sets the value of a published property on the object that GetInstance (see page 23336) returns, so that it is available in custom Variants (see page 23567).

Pascal

```
function SetProperty(const V: TVarData; const Name: string; const Value: TVarData): Boolean;
override;
```

C++

```
virtual __fastcall Boolean SetProperty(const TVarData V, const AnsiString Name, const TVarData
Value);
```

Description

SetProperty enables an application to set the properties of instances of the publishable Variant type. It implements all the settable properties that are published by the object that GetInstance (see page 23336) returns.

When the application contains a line such as

```
MyVariant.SpecialProperty := NewValue;
```

(Delphi) or calls the Exec method of the Variant (C++), a call is automatically generated to SetProperty. If SpecialProperty is a published settable property on the object that implements the Variant's data, SetProperty sets the property on the object included in V so that the Variant's value is changed accordingly.

V is the TVarData record (Delphi) or structure (C++) from the Variant instance whose property is being set. It includes the object that the GetInstance (see page 23336) method returns.

Name is the name of the property.

Value is a TVarData record or structure that contains the new property value.

GetProperty (see page 23336) returns true if it sets a property value, and raises an exception of Name does not specify a published settable property on the object that GetInstance (see page 23336) returns.

See Also

DoFunction (see page 23610)

DoProcedure (see page 23611)

GetInstance (see page 23336)

Exec

166.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FreeAndNilProperties (see page 23339)	Frees any property values that are objects and sets their value to nil (Delphi) or NULL (C++).
	GetDynArrayProp (see page 23340)	Returns the value of a component property that is a dynamic array type.
	GetDynArrayProp (see page 23340)	Returns the value of a component property that is a dynamic array type.
	GetEnumName (see page 23341)	Returns the name of a Delphi enumerated type constant given its value.
	GetEnumProp (see page 23341)	Returns the value of a component property that is an enumerated type.
	GetEnumProp (see page 23341)	Returns the value of a component property that is an enumerated type.
	GetEnumValue (see page 23342)	Returns the value of an enumerated type constant given its string representation.
	GetFloatProp (see page 23343)	Returns the value of a component property that is a float type.
	GetFloatProp (see page 23343)	Returns the value of a component property that is a float type.
	GetInt64Prop (see page 23343)	Returns the value of a component property that is an Int64 type.
	GetInt64Prop (see page 23343)	Returns the value of a component property that is an Int64 type.
	GetInterfaceProp (see page 23344)	Returns the value of a component property that is an interface type.
	GetInterfaceProp (see page 23344)	Returns the value of a component property that is an interface type.
	GetMethodProp (see page 23345)	Returns the value of a component property that is a method type.
	GetMethodProp (see page 23345)	Returns the value of a component property that is a method type.
	GetObjectProp (see page 23346)	Returns the value of a component property that is an object type.
	GetObjectProp (see page 23346)	Returns the value of a component property that is an object type.
	GetObjectPropClass (see page 23347)	Returns the class type of a component property that is an object type.
	GetObjectPropClass (see page 23347)	Returns the class type of a component property that is an object type.
	GetObjectPropClass (see page 23347)	Returns the class type of a component property that is an object type.
	GetOrdProp (see page 23347)	Returns the value of a component property that is an ordinal type.
	GetOrdProp (see page 23347)	Returns the value of a component property that is an ordinal type.
	GetPropInfo (see page 23348)	Retrieves information about a component's property.
	GetPropInfo (see page 23348)	Retrieves information about a component's property.
	GetPropInfo (see page 23348)	Retrieves information about a component's property.
	GetPropInfo (see page 23348)	Retrieves information about a component's property.
	GetPropInfos (see page 23349)	Retrieves a list of component properties.
	GetPropList (see page 23350)	Retrieves a list of component properties.
	GetPropList (see page 23350)	Retrieves a list of component properties.
	GetPropList (see page 23350)	Retrieves a list of component properties.
	GetPropValue (see page 23351)	Retrieves a component's property value as a variant.
	GetPropValue (see page 23351)	Retrieves a component's property value as a variant.
	GetSetElementName (see page 23351)	This is function TypInfo.GetSetElementName.
	GetSetElementValue (see page 23352)	This is function TypInfo.GetSetElementValue.
	GetSetProp (see page 23352)	Returns the value of a component property that is a set type.
	GetSetProp (see page 23352)	Returns the value of a component property that is a set type.
	GetStrProp (see page 23353)	Returns the value of a component property that is a string type.
	GetStrProp (see page 23353)	Returns the value of a component property that is a string type.
	GetTypeData (see page 23354)	Returns a pointer to information about a type.
	GetVariantProp (see page 23354)	Returns the value of a component property that is a variant type.
	GetVariantProp (see page 23354)	Returns the value of a component property that is a variant type.
	GetWideStrProp (see page 23355)	Returns the value of a component property that is a WideString type.
	GetWideStrProp (see page 23355)	Returns the value of a component property that is a WideString type.
	IsPublishedProp (see page 23356)	Determines if a specified component property is a published property.

	IsPublishedProp (see page 23356)	Determines if a specified component property is a published property.
	IsStoredProp (see page 23356)	Determines if a specified component property has the stored attribute.
	IsStoredProp (see page 23356)	Determines if a specified component property has the stored attribute.
	PropIsType (see page 23357)	Determines if a component property is of a particular data type.
	PropIsType (see page 23357)	Determines if a component property is of a particular data type.
	PropType (see page 23358)	Retrieves a component's property type.
	PropType (see page 23358)	Retrieves a component's property type.
	SetDynArrayProp (see page 23358)	Sets the value of a component property that is a dynamic array type.
	SetDynArrayProp (see page 23358)	Sets the value of a component property that is a dynamic array type.
	SetEnumProp (see page 23359)	Sets the value of a component property that is an enumerated type.
	SetEnumProp (see page 23359)	Sets the value of a component property that is an enumerated type.
	SetFloatProp (see page 23360)	Sets the value of a component property that is a float type.
	SetFloatProp (see page 23360)	Sets the value of a component property that is a float type.
	SetInt64Prop (see page 23361)	Sets the value of a component property that is an Int64 type.
	SetInt64Prop (see page 23361)	Sets the value of a component property that is an Int64 type.
	SetInterfaceProp (see page 23362)	Sets the value of a component property that is an Interface type.
	SetInterfaceProp (see page 23362)	Sets the value of a component property that is an Interface type.
	SetMethodProp (see page 23362)	Sets the value of a component property that is a method type.
	SetMethodProp (see page 23362)	Sets the value of a component property that is a method type.
	SetObjectProp (see page 23363)	Sets the value of a component property that is an object type.
	SetObjectProp (see page 23363)	Sets the value of a component property that is an object type.
	SetOrdProp (see page 23364)	Sets the value of a component property that is an ordinal type.
	SetOrdProp (see page 23364)	Sets the value of a component property that is an ordinal type.
	SetPropValue (see page 23365)	Sets the value of a component property using a variant value.
	SetPropValue (see page 23365)	Sets the value of a component property using a variant value.
	SetSetProp (see page 23366)	Sets the value of a component property that is a set type.
	SetSetProp (see page 23366)	Sets the value of a component property that is a set type.
	SetStrProp (see page 23367)	Sets the value of a component property that is a string type.
	SetStrProp (see page 23367)	Sets the value of a component property that is a string type.
	SetToString (see page 23367)	Returns one or more of the set values of a component's property.
	SetVariantProp (see page 23368)	Sets the value of a component property that is a variant type.
	SetVariantProp (see page 23368)	Sets the value of a component property that is a variant type.
	SetWideStrProp (see page 23369)	Sets the value of a component property that is a WideString type.
	SetWideStrProp (see page 23369)	Sets the value of a component property that is a WideString type.
	SortPropList (see page 23370)	Sorts a list of component properties into alphanumeric sequence.
	StringToSet (see page 23370)	Returns a bit mask of set values for a set type component property.

166.2.1 TypInfo.FreeAndNilProperties Function

Frees any property values that are objects and sets their value to nil (Delphi) or NULL (C++).

Pascal

```
procedure FreeAndNilProperties(AObject: TObject);
```

C++

```
FreeAndNilProperties(TObject * AObject);
```

File

TypInfo

Description

Call FreeAndNilProperties from a destructor to free all internally created objects and set the corresponding property values to nil

(Delphi) or NULL (C++). FreeAndNilProperties searches the properties of AObject, identifying any that are objects. It frees those objects, and sets the corresponding property to nil (Delphi) or NULL (C++).

. By default, all persistent objects (TPersistent and descendants) are compiled with RTTI. Other TObject descendants can be compiled with RTTI by using the $M+ compiler switch in Delphi or the __declspec(delphirtti) directive in C++.

Warning: Calling FreeAndNilProperties frees all object properties, which may lead to unintended results. FreeAndNilProperties can't distinguish between objects that were created by AObject (and typically stored in a class member) from objects that are created independently and which should not be freed with AObject. To avoid unintentionally freeing independently-created objects, set properties that refer to external objects to nil (Delphi) or NULL (C++) before calling FreeAndNilProperties.

See Also

FreeAndNil

166.2.2 TypInfo.GetDynArrayProp Function

Returns the value of a component property that is a dynamic array type.

Pascal

```
function GetDynArrayProp(Instance: TObject; const PropName: string): Pointer; overload;
function GetDynArrayProp(Instance: TObject; PropInfo: PPropInfo): Pointer; overload;
```

C++

```
void * GetDynArrayProp(TObject * Instance, const AnsiString PropName);
void * GetDynArrayProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetDynArrayProp utilises Delphi's RTTI (Run Time Type Information) to return the current value of a component's property where that property is a dynamic array type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a pointer to the array.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 23351) to retrieve a property value where the type is unknown - it is returned in a Variant type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetPropInfo (see page 23348)

GetPropList (see page 23350)

GetPropValue (see page 23351)

PPropInfo (see page 23381)

SetDynArrayProp (see page 23358)

166.2.3 TypInfo.GetEnumName Function

Returns the name of a Delphi enumerated type constant given its value.

Pascal

```
function GetEnumName(TypeInfo: PTypeInfo; Value: Integer): string;
```

C++

```
AnsiString GetEnumName(PTypeInfo TypeInfo, int Value);
```

File

TypInfo

Description

Use GetEnumName to convert a Delphi enumerated value into the symbolic name that represents it in code.

TypeInfo is the type information record that describes the enumerated type. You can obtain the type information for a type by calling the TypeInfo function in Delphi or the __delphirtti operator in C++.

Value is the enumerated value, represented as an integer.

GetEnumName returns the string that represents the specified value.

See Also

IntToIdent

TypeInfo

GetEnumValue (see page 23342)

166.2.4 TypInfo.GetEnumProp Function

Returns the value of a component property that is an enumerated type.

Pascal

```
function GetEnumProp(Instance: TObject; const PropName: string): string; overload;
function GetEnumProp(Instance: TObject; PropInfo: PPropInfo): string; overload;
```

C++

```
AnsiString GetEnumProp(TObject * Instance, const AnsiString PropName);
AnsiString GetEnumProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetEnumProp utilises Delphi's RTTI (Run Time Type Information) to return the current value of a component's property where that property is an enumerated type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a string.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 23351) to retrieve a property value where the type is unknown - it is returned in a Variant type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetPropInfo (see page 23348)

GetPropList (see page 23350)

GetPropValue (see page 23351)

PPropInfo (see page 23381)

SetEnumProp (see page 23359)

166.2.5 TypInfo.GetEnumValue Function

Returns the value of an enumerated type constant given its string representation.

Pascal

```
function GetEnumValue(TypeInfo: PTypeInfo; const Name: string): Integer;
```

C++

```
int GetEnumValue(PTypeInfo TypeInfo, const AnsiString Name);
```

File

TypInfo

Description

Use GetEnumValue to convert the string that represents an enumerated value into the corresponding integer that is its implemented value.

TypeInfo is the type information record that describes the enumerated type. You can obtain the type information for a type by calling the TypeInfo function in Delphi or using the __delphirtti operator in C++.

Name is the string that is used in code to represent the enumerated value.

GetEnumValue returns the corresponding integer value.

See Also

IdentToInt

TypeInfo

GetEnumName (see page 23341)

166.2.6 TypInfo.GetFloatProp Function

Returns the value of a component property that is a float type.

Pascal

```
function GetFloatProp(Instance: TObject; const PropName: string): Extended; overload;
function GetFloatProp(Instance: TObject; PropInfo: PPropInfo): Extended; overload;
```

C++

```
Extended GetFloatProp(TObject * Instance, const AnsiString PropName);
Extended GetFloatProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetFloatProp utilises Delphi's RTTI (Run Time Type Information) to return the current value of a component's property where that property is a floating point number type. 0.0 is returned if the property is not of this type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a float value.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 23351) to retrieve a property value where the type is unknown - it is returned in a Variant type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetPropInfo (see page 23348)

GetPropList (see page 23350)

GetPropValue (see page 23351)

PPropInfo (see page 23381)

SetFloatProp (see page 23360)

166.2.7 TypInfo.GetInt64Prop Function

Returns the value of a component property that is an Int64 type.

Pascal

```
function GetInt64Prop(Instance: TObject; const PropName: string): Int64; overload;
function GetInt64Prop(Instance: TObject; PropInfo: PPropInfo): Int64; overload;
```

C++

```
Int64 GetInt64Prop(TObject * Instance, const AnsiString PropName);
Int64 GetInt64Prop(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetInt64Prop utilises Delphi's RTTI (Run Time Type Information) to return the current value of a component's property where that property is an Int64 type. 0 is returned if the property is not of this type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as an Int64 value.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 23351) to retrieve a property value where the type is unknown - it is returned in a Variant type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetPropInfo (see page 23348)

GetPropList (see page 23350)

GetPropValue (see page 23351)

PPropInfo (see page 23381)

SetInt64Prop (see page 23361)

166.2.8 TypInfo.GetInterfaceProp Function

Returns the value of a component property that is an interface type.

Pascal

```
function GetInterfaceProp(Instance: TObject; const PropName: string): IInterface; overload;
function GetInterfaceProp(Instance: TObject; PropInfo: PPropInfo): IInterface; overload;
```

C++

```
IInterface GetInterfaceProp(TObject * Instance, const AnsiString PropName);
IInterface GetInterfaceProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetInterfaceProp utilises Delphi's RTTI (Run Time Type Information) to return the current value of a component's property where that property is an interface type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as an interface value.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 23351) to retrieve a property value where the type is unknown - it is returned in a Variant type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetPropInfo (see page 23348)

GetPropList (see page 23350)

GetPropValue (see page 23351)

PPropInfo (see page 23381)

SetInterfaceProp (see page 23362)

166.2.9 TypInfo.GetMethodProp Function

Returns the value of a component property that is a method type.

Pascal

```
function GetMethodProp(Instance: TObject; const PropName: string): TMethod; overload;
function GetMethodProp(Instance: TObject; PropInfo: PPropInfo): TMethod; overload;
```

C++

```
TMethod GetMethodProp(TObject * Instance, const AnsiString PropName);
TMethod GetMethodProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetMethodProp utilises Delphi's RTTI (Run Time Type Information) to return the current value of a component's property where that property is a method type : an event routine.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a method value.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 23351) to retrieve a property value where the type is unknown - it is returned in a Variant type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is

thrown.

See Also

GetPropInfo (see page 23348)

GetPropList (see page 23350)

GetPropValue (see page 23351)

PPropInfo (see page 23381)

SetMethodProp (see page 23362)

166.2.10 TypInfo.GetObjectProp Function

Returns the value of a component property that is an object type.

Pascal

```
function GetObjectProp(Instance: TObject; const PropName: string; MinClass: TClass = nil):
TObject; overload;
function GetObjectProp(Instance: TObject; PropInfo: PPropInfo; MinClass: TClass = nil):
TObject; overload;
```

C++

```
TObject * GetObjectProp(TObject * Instance, const AnsiString PropName, TClass MinClass = nil);
TObject * GetObjectProp(TObject * Instance, PPropInfo PropInfo, TClass MinClass = nil);
```

File

TypInfo

Description

GetObjectProp utilises Delphi's RTTI (Run Time Type Information) to return the current value of a component's property where that property is an object type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as an object, or nil if the property is not an object type.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 23351) to retrieve a property value where the type is unknown - it is returned in a Variant type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetPropInfo (see page 23348)

GetPropList (see page 23350)

GetPropValue (see page 23351)

PPropInfo (see page 23381)

SetObjectProp (see page 23363)

166.2.11 TypInfo.GetObjectPropClass Function

Returns the class type of a component property that is an object type.

Pascal

```
function GetObjectPropClass(Instance: TObject; const PropName: string): TClass; overload;
function GetObjectPropClass(Instance: TObject; PropInfo: PPropInfo): TClass; overload;
function GetObjectPropClass(PropInfo: PPropInfo): TClass; overload;
```

C++

```
TClass GetObjectPropClass(TObject * Instance, const AnsiString PropName);
TClass GetObjectPropClass(TObject * Instance, PPropInfo PropInfo);
TClass GetObjectPropClass(PPropInfo PropInfo);
```

File

TypInfo

Description

GetObjectPropClass utilises Delphi's RTTI (Run Time Type Information) to return the class type of a component's property where that property is an object.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the class type of the property object value is returned.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 23351) to retrieve a property value where the type is unknown - it is returned in a Variant type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetPropInfo (see page 23348)

GetPropList (see page 23350)

GetPropValue (see page 23351)

PPropInfo (see page 23381)

166.2.12 TypInfo.GetOrdProp Function

Returns the value of a component property that is an ordinal type.

Pascal

```
function GetOrdProp(Instance: TObject; const PropName: string): Longint; overload;
function GetOrdProp(Instance: TObject; PropInfo: PPropInfo): Longint; overload;
```

C++

```
Longint GetOrdProp(TObject * Instance, const AnsiString PropName);
Longint GetOrdProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetOrdProp utilises Delphi's RTTI (Run Time Type Information) to return the current value of a component's property where that property is an Integer, Char, WideChar, or Class type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as an LongInt value.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 23351) to retrieve a property value where the type is unknown - it is returned in a Variant type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetPropInfo (see page 23348)

GetPropList (see page 23350)

GetPropValue (see page 23351)

PPropInfo (see page 23381)

SetOrdProp (see page 23364)

166.2.13 TypInfo.GetPropInfo Function

Retrieves information about a component's property.

Pascal

```
function GetPropInfo(Instance: TObject; const PropName: string; AKinds: TTypeKinds = []):
PPropInfo; overload;
function GetPropInfo(AClass: TClass; const PropName: string; AKinds: TTypeKinds = []):
PPropInfo; overload;
function GetPropInfo(TypeInfo: PTypeInfo; const PropName: string): PPropInfo; overload;
function GetPropInfo(TypeInfo: PTypeInfo; const PropName: string; AKinds: TTypeKinds):
PPropInfo; overload;
```

C++

```
PPropInfo GetPropInfo(TObject * Instance, const AnsiString PropName, TTypeKinds AKinds = []);
PPropInfo GetPropInfo(TClass AClass, const AnsiString PropName, TTypeKinds AKinds = []);
PPropInfo GetPropInfo(PTypeInfo TypeInfo, const AnsiString PropName);
PPropInfo GetPropInfo(PTypeInfo TypeInfo, const AnsiString PropName, TTypeKinds AKinds);
```

File

TypInfo

Description

GetPropInfo utilises Delphi's RTTI (Run Time Type Information) to retrieve a pointer to a property information record for a component.

In the first form of this function, **TypeInfo** specifies the component type whose property you want to retrieve. **PropName** specifies the property name as a string. You may optionally specify a **AKinds** set of TTypeKind (see page 23379) property value types that the property must belong to.

The second form of the function is similar to the first, with **AClass** replacing **TypeInfo** as the means of specifying the component type. **AKinds** is now a required value.

The final form of the function is like the second form, with **Instance** specifying a component instance.

In all forms of the function, the returned value is a pointer to a TPropInfo (see page 23376) value. This provides rich data about the property being investigated at run time. This enables more elegant component processing than can be achieved with hard coded component names/properties.

Note: If the specified property does not exist, a nil pointer is returned.

See Also

PTypeInfo (see page 23383)

PPropInfo (see page 23381)

TTypeKinds (see page 23387)

166.2.14 TypInfo.GetPropInfos Function

Retrieves a list of component properties.

Pascal

```
procedure GetPropInfos(TypeInfo: PTypeInfo; PropList: PPropList);
```

C++

```
GetPropInfos(PTypeInfo TypeInfo, PPropList PropList);
```

File

TypInfo

Description

GetPropInfos utilises Delphi's RTTI (Run Time Type Information) to retrieve a list of property information records for a component or component type.

The **TypeInfo** defines the component type and **PropList** defines a variable to receive the list of properties : a pointer to a TPropList (see page 23387) variable

The Delphi inbuilt TypeInfo function may be used to provide the **TypeInfo** value.

Note: If you want the count of properties returned in PropList

, use GetPropList (see page 23350) instead.

See Also

GetPropList (see page 23350)

PTypeInfo (see page 23383)

PPropList (see page 23382)

TPropList (see page 23387)

TTypeKinds (see page 23387)

166.2.15 TypInfo.GetPropList Function

Retrieves a list of component properties.

Pascal

```
function GetPropList(TypeInfo: PTypeInfo; TypeKinds: TTypeKinds; PropList: PPropList;
SortList: Boolean = True): Integer; overload;
function GetPropList(TypeInfo: PTypeInfo; out PropList: PPropList): Integer; overload;
function GetPropList(AObject: TObject; out PropList: PPropList): Integer; overload;
```

C++

```
int GetPropList(PTypeInfo TypeInfo, TTypeKinds TypeKinds, PPropList PropList, Boolean SortList
= True);
int GetPropList(PTypeInfo TypeInfo, PPropList PropList);
int GetPropList(TObject * AObject, PPropList PropList);
```

File

TypInfo

Description

GetPropList utilises Delphi's RTTI (Run Time Type Information) to retrieve a list of property information records for a component or component type.

In one form of this function, **TypeInfo** defines the component type, **PropList** defines a variable to receive the list of properties : a pointer to a TPropList (see page 23387) variable, and **TypeKinds** defines the type of properties that will be returned. Use tkAny (see page 23389) to return all types. The list may optionally be sorted by setting **SortList** true.

The Delphi inbuilt TypeInfo function may be used to provide the **TypeInfo** value.

In the other form of this function, you specify a component instance rather than component type. The **TypeKinds** and **SortList** parameters can no longer be specified.

In both forms of the function, the returned value is the number of properties in **PropList**.

Note: If you do not need the count of properties returned, use GetPropInfos (see page 23349).

See Also

GetPropInfos (see page 23349)

PTypeInfo (see page 23383)

PPropList (see page 23382)

TPropList (see page 23387)

TTypeKinds (see page 23387)

166.2.16 TypInfo.GetPropValue Function

Retrieves a component's property value as a variant.

Pascal

```
function GetPropValue(Instance: TObject; const PropName: string; PreferStrings: Boolean =
True): Variant; overload;
function GetPropValue(Instance: TObject; PropInfo: PPropInfo; PreferStrings: Boolean = True):
Variant; overload;
```

C++

```
Variant GetPropValue(TObject * Instance, const AnsiString PropName, Boolean PreferStrings =
True);
Variant GetPropValue(TObject * Instance, PPropInfo PropInfo, Boolean PreferStrings = True);
```

File

TypInfo

Description

GetPropValue utilises Delphi's RTTI (Run Time Type Information) to retrieve a a component property as a variant.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the returned value the property value as a variant.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

PTypeInfo (see page 23383)

PPropInfo (see page 23381)

TTypeKinds (see page 23387)

SetPropValue (see page 23365)

166.2.17 TypInfo.GetSetElementName Function

Pascal

```
function GetSetElementName(TypeInfo: PTypeInfo; Value: Integer): string;
```

C++

```
AnsiString GetSetElementName(PTypeInfo TypeInfo, int Value);
```

File

TypInfo

Description

This is function TypInfo.GetSetElementName.

166.2.18 TypInfo.GetSetElementValue Function

Pascal

```
function GetSetElementValue(TypeInfo: PTypeInfo; const Name: string): Integer;
```

C++

```
int GetSetElementValue(PTypeInfo TypeInfo, const AnsiString Name);
```

File

TypInfo

Description

This is function TypInfo.GetSetElementValue.

166.2.19 TypInfo.GetSetProp Function

Returns the value of a component property that is a set type.

Pascal

```
function GetSetProp(Instance: TObject; const PropName: string; Brackets: Boolean = False):
string; overload;
function GetSetProp(Instance: TObject; PropInfo: PPropInfo; Brackets: Boolean = False):
string; overload;
```

C++

```
AnsiString GetSetProp(TObject * Instance, const AnsiString PropName, Boolean Brackets = False);
AnsiString GetSetProp(TObject * Instance, PPropInfo PropInfo, Boolean Brackets = False);
```

File

TypInfo

Description

GetSetProp utilises Delphi's RTTI (Run Time Type Information) to return the current value of a component's property where that property is a set of values type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the list if set values is returned as a comma separated string. You may optionally set **Brackets** true so as to enclose this within square brackets, as shown in the IDE for the property.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 23351) to retrieve a property value where the type is unknown - it is returned in a Variant type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetPropInfo (see page 23348)

GetPropList (see page 23350)

GetPropValue (see page 23351)

PPropInfo (see page 23381)

SetSetProp (see page 23366)

166.2.20 TypInfo.GetStrProp Function

Returns the value of a component property that is a string type.

Pascal

```
function GetStrProp(Instance: TObject; const PropName: string): string; overload;
function GetStrProp(Instance: TObject; PropInfo: PPropInfo): string; overload;
```

C++

```
AnsiString GetStrProp(TObject * Instance, const AnsiString PropName);
AnsiString GetStrProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetStrProp utilises Delphi's RTTI (Run Time Type Information) to return the current value of a component's property where that property is a String, LongString or WideString type. An empty string is returned if the property is not of this type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a string.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 23351) to retrieve a property value where the type is unknown - it is returned in a Variant type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetPropInfo (see page 23348)

GetPropList (see page 23350)

GetPropValue (see page 23351)

PPropInfo (see page 23381)

SetStrProp (see page 23367)

166.2.21 TypInfo.GetTypeData Function

Returns a pointer to information about a type.

Pascal

```
function GetTypeData(TypeInfo: PTypeInfo): PTypeData;
```

C++

```
PTypeData GetTypeData(PTypeInfo TypeInfo);
```

File

TypInfo

Description

GetTypeData utilises Delphi's RTTI (Run Time Type Information) to return a pointer to TTypeData (see page 23377) record.

The TTypeData (see page 23377) record is a variant type, but does not contain the variant as a tag. To identify which variant applies to a particular type, you should retrieve the TTypeKind (see page 23379) value using the TypeInfo routine.

The TTypeData (see page 23377) record holds meta data about a property. For example, the name of the unit a class type is stored in.

See Also

PTypeData (see page 23383)

TTypeData (see page 23377)

TTypeKind (see page 23379)

166.2.22 TypInfo.GetVariantProp Function

Returns the value of a component property that is a variant type.

Pascal

```
function GetVariantProp(Instance: TObject; const PropName: string): Variant; overload;
function GetVariantProp(Instance: TObject; PropInfo: PPropInfo): Variant; overload;
```

C++

```
Variant GetVariantProp(TObject * Instance, const AnsiString PropName);
Variant GetVariantProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetVariantProp utilises Delphi's RTTI (Run Time Type Information) to return the current value of a component's property that is a variant type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a variant.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown. If the property is not a variant type, an EVariantBadVarTypeError exception is thrown.

See Also

GetPropInfo (see page 23348)

GetPropList (see page 23350)

GetPropValue (see page 23351)

PPropInfo (see page 23381)

SetVariantProp (see page 23368)

166.2.23 TypInfo.GetWideStrProp Function

Returns the value of a component property that is a WideString type.

Pascal

```
function GetWideStrProp(Instance: TObject; const PropName: string): WideString; overload;
function GetWideStrProp(Instance: TObject; PropInfo: PPropInfo): WideString; overload;
```

C++

```
BSTR GetWideStrProp(TObject * Instance, const AnsiString PropName);
BSTR GetWideStrProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetWideStrProp utilises Delphi's RTTI (Run Time Type Information) to return the current value of a component's property where that property is a WideString type. A string or LongString type is returned via a call to the GetWideStrProp routine. Otherwise, for other types, an empty string is returned if the property is not of this type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a WideString.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 23351) to retrieve a property value where the type is unknown - it is returned in a Variant type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetPropInfo (see page 23348)

GetPropList (see page 23350)

GetPropValue (see page 23351)

PPropInfo (see page 23381)

SetWideStrProp (see page 23369)

166.2.24 TypInfo.IsPublishedProp Function

Determines if a specified component property is a published property.

Pascal

```
function IsPublishedProp(Instance: TObject; const PropName: string): Boolean; overload;
function IsPublishedProp(AClass: TClass; const PropName: string): Boolean; overload;
```

C++

```
Boolean IsPublishedProp(TObject * Instance, const AnsiString PropName);
Boolean IsPublishedProp(TClass AClass, const AnsiString PropName);
```

File

TypInfo

Description

IsPublishedProp returns true if the specified property is published. Delphi's RTTI (Run Time Type Information) facility requires that properties are published.

In one form of this function, the component is defined by the **TClass** parameter.

In the other form of this function, an **Instance** of the component is used.

In both forms of the function, **PropName** specifies the property as a string.

See Also

GetPropInfo (see page 23348)

GetPropValue (see page 23351)

166.2.25 TypInfo.IsStoredProp Function

Determines if a specified component property has the stored attribute.

Pascal

```
function IsStoredProp(Instance: TObject; const PropName: string): Boolean; overload;
function IsStoredProp(Instance: TObject; PropInfo: PPropInfo): Boolean; overload;
```

C++

```
Boolean IsStoredProp(TObject * Instance, const AnsiString PropName);
Boolean IsStoredProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

IsStoredProp returns true if the specified property is saved in the form file. The stored attribute is set true by default : only those that have the keyword stored defined as false do not.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the value of the stored attribute is returned.

See Also

GetPropInfo (see page 23348)

GetPropValue (see page 23351)

PPropInfo (see page 23381)

166.2.26 TypInfo.PropIsType Function

Determines if a component property is of a particular data type.

Pascal

```
function PropIsType(Instance: TObject; const PropName: string; TypeKind: TTypeKind): Boolean;
overload;
function PropIsType(AClass: TClass; const PropName: string; TypeKind: TTypeKind): Boolean;
overload;
```

C++

```
Boolean PropIsType(TObject * Instance, const AnsiString PropName, TTypeKind TypeKind);
Boolean PropIsType(TClass AClass, const AnsiString PropName, TTypeKind TypeKind);
```

File

TypInfo

Description

PropIsType returns true if the property is of the specified data type.

In the first form of this function, **AClass** specifies the component class whose property you want to look at.

The second form of the function a class **Instance** replaces **AClass**.

In both forms of the function, **PropName** specifies the property name as a string. If the property has the **TypeKind** data type, then true is returned. Otherwise false is returned.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetPropInfo (see page 23348)

GetPropValue (see page 23351)

PPropInfo (see page 23381)

TTypeKinds (see page 23387)

166.2.27 TypInfo.PropType Function

Retrieves a component's property type.

Pascal

```
function PropType(Instance: TObject; const PropName: string): TTypeKind; overload;
function PropType(AClass: TClass; const PropName: string): TTypeKind; overload;
```

C++

```
TTypeKind PropType(TObject * Instance, const AnsiString PropName);
TTypeKind PropType(TClass AClass, const AnsiString PropName);
```

File

TypInfo

Description

PropType utilises Delphi's RTTI (Run Time Type Information) to return the type of a component property in a TTypeKind (see page 23379) value.

In the first form of this function, **AClass** specifies the component class whose property you want to retrieve. **PropName** specifies the property name as a string.

The second form of the function is similar to the first, with **Instance** replacing **AClass** as the means of specifying the component.

In both forms of the function, the returned value is a TTypeKind (see page 23379) value.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

PTypeInfo (see page 23383)

PPropInfo (see page 23381)

TTypeKind (see page 23379)

166.2.28 TypInfo.SetDynArrayProp Function

Sets the value of a component property that is a dynamic array type.

Pascal

```
procedure SetDynArrayProp(Instance: TObject; const PropName: string; const Value: Pointer);
overload;
procedure SetDynArrayProp(Instance: TObject; PropInfo: PPropInfo; const Value: Pointer);
overload;
```

C++

```
SetDynArrayProp(TObject * Instance, const AnsiString PropName, const void * Value);
SetDynArrayProp(TObject * Instance, PPropInfo PropInfo, const void * Value);
```

File

TypInfo

Description

SetDynArrayProp utilises Delphi's RTTI (Run Time Type Information) to set the value of a component's property where that property is a dynamic array type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 23365) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 23351) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetEnumProp (see page 23341)

GetPropInfo (see page 23348)

GetPropList (see page 23350)

SetPropValue (see page 23365)

PPropInfo (see page 23381)

166.2.29 TypInfo.SetEnumProp Function

Sets the value of a component property that is an enumerated type.

Pascal

```
procedure SetEnumProp(Instance: TObject; const PropName: string; const Value: string);
overload;
procedure SetEnumProp(Instance: TObject; PropInfo: PPropInfo; const Value: string); overload;
```

C++

```
SetEnumProp(TObject * Instance, const AnsiString PropName, const AnsiString Value);
SetEnumProp(TObject * Instance, PPropInfo PropInfo, const AnsiString Value);
```

File

TypInfo

Description

SetEnumProp utilises Delphi's RTTI (Run Time Type Information) to set the value of a component's property where that property is an enumerated type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 23365) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 23351) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetEnumProp (see page 23341)

GetPropInfo (see page 23348)

GetPropList (see page 23350)

SetPropValue (see page 23365)

PPropInfo (see page 23381)

166.2.30 TypInfo.SetFloatProp Function

Sets the value of a component property that is a float type.

Pascal

```
procedure SetFloatProp(Instance: TObject; const PropName: string; const Value: Extended);
overload;
procedure SetFloatProp(Instance: TObject; PropInfo: PPropInfo; const Value: Extended);
overload;
```

C++

```
SetFloatProp(TObject * Instance, const AnsiString PropName, const Extended Value);
SetFloatProp(TObject * Instance, PPropInfo PropInfo, const Extended Value);
```

File

TypInfo

Description

SetFloatProp utilises Delphi's RTTI (Run Time Type Information) to set the value of a component's property where that property is a floating point number.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 23365) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 23351) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetFloatProp (see page 23343)

GetPropInfo (see page 23348)

GetPropList (see page 23350)

SetPropValue (see page 23365)

PPropInfo (see page 23381)

166.2.31 TypInfo.SetInt64Prop Function

Sets the value of a component property that is an Int64 type.

Pascal

```
procedure SetInt64Prop(Instance: TObject; const PropName: string; const Value: Int64);
overload;
procedure SetInt64Prop(Instance: TObject; PropInfo: PPropInfo; const Value: Int64); overload;
```

C++

```
SetInt64Prop(TObject * Instance, const AnsiString PropName, const Int64 Value);
SetInt64Prop(TObject * Instance, PPropInfo PropInfo, const Int64 Value);
```

File

TypInfo

Description

SetInt64Prop utilises Delphi's RTTI (Run Time Type Information) to set the value of a component's property where that property is an Int64 number.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 23365) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 23351) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetInt64Prop (see page 23343)

GetPropInfo (see page 23348)

GetPropList (see page 23350)

SetPropValue (see page 23365)

PPropInfo (see page 23381)

166.2.32 TypInfo.SetInterfaceProp Function

Sets the value of a component property that is an Interface type.

Pascal

```
procedure SetInterfaceProp(Instance: TObject; const PropName: string; const Value:
IInterface); overload;
procedure SetInterfaceProp(Instance: TObject; PropInfo: PPropInfo; const Value: IInterface);
overload;
```

C++

```
SetInterfaceProp(TObject * Instance, const AnsiString PropName, const IInterface Value);
SetInterfaceProp(TObject * Instance, PPropInfo PropInfo, const IInterface Value);
```

File

TypInfo

Description

SetInterfaceProp utilises Delphi's RTTI (Run Time Type Information) to set the value of a component's property where that property is an Interface.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 23365) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 23351) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetInterfaceProp (see page 23344)

GetPropInfo (see page 23348)

GetPropList (see page 23350)

SetPropValue (see page 23365)

PPropInfo (see page 23381)

166.2.33 TypInfo.SetMethodProp Function

Sets the value of a component property that is a method type.

Pascal

```
procedure SetMethodProp(Instance: TObject; const PropName: string; const Value: TMethod);
overload;
procedure SetMethodProp(Instance: TObject; PropInfo: PPropInfo; const Value: TMethod);
overload;
```

C++

```
SetMethodProp(TObject * Instance, const AnsiString PropName, const TMethod Value);
SetMethodProp(TObject * Instance, PPropInfo PropInfo, const TMethod Value);
```

File

TypInfo

Description

SetMethodProp utilises Delphi's RTTI (Run Time Type Information) to set the value of a component's property where that property is an Event (Method property).

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 23365) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 23351) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetMethodProp (see page 23345)

GetPropInfo (see page 23348)

GetPropList (see page 23350)

SetPropValue (see page 23365)

PPropInfo (see page 23381)

166.2.34 TypInfo.SetObjectProp Function

Sets the value of a component property that is an object type.

Pascal

```
procedure SetObjectProp(Instance: TObject; const PropName: string; Value: TObject); overload;
procedure SetObjectProp(Instance: TObject; PropInfo: PPropInfo; Value: TObject; ValidateClass:
Boolean = True); overload;
```

C++

```
SetObjectProp(TObject * Instance, const AnsiString PropName, TObject * Value);
SetObjectProp(TObject * Instance, PPropInfo PropInfo, TObject * Value, Boolean ValidateClass =
True);
```

File

TypInfo

Description

SetObjectProp utilises Delphi's RTTI (Run Time Type Information) to set the value of a component's property where that property is an object.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 23365) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 23351) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetObjectProp (see page 23346)

GetPropInfo (see page 23348)

GetPropList (see page 23350)

SetPropValue (see page 23365)

PPropInfo (see page 23381)

166.2.35 TypInfo.SetOrdProp Function

Sets the value of a component property that is an ordinal type.

Pascal

```
procedure SetOrdProp(Instance: TObject; const PropName: string; Value: Longint); overload;
procedure SetOrdProp(Instance: TObject; PropInfo: PPropInfo; Value: Longint); overload;
```

C++

```
SetOrdProp(TObject * Instance, const AnsiString PropName, Longint Value);
SetOrdProp(TObject * Instance, PPropInfo PropInfo, Longint Value);
```

File

TypInfo

Description

SetOrdProp utilises Delphi's RTTI (Run Time Type Information) to set the value of a component's property where that property is an Integer, Char, WideChar, or Class type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 23365) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 23351) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetOrdProp (see page 23347)

GetPropInfo (see page 23348)

GetPropList (see page 23350)

SetPropValue (see page 23365)

PPropInfo (see page 23381)

166.2.36 TypInfo.SetPropValue Function

Sets the value of a component property using a variant value.

Pascal

```
procedure SetPropValue(Instance: TObject; const PropName: string; const Value: Variant);
overload;
procedure SetPropValue(Instance: TObject; PropInfo: PPropInfo; const Value: Variant); overload;
```

C++

```
SetPropValue(TObject * Instance, const AnsiString PropName, const Variant Value);
SetPropValue(TObject * Instance, PPropInfo PropInfo, const Variant Value);
```

File

TypInfo

Description

SetPropValue utilises Delphi's RTTI (Run Time Type Information) to set the value of a component's property using a variant value.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**. Note that only certain property assignments are possible using variants. It is normally be used when using GetPropValue (see page 23351) from another property.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetPropValue (see page 23351)

GetPropInfo (see page 23348)

GetPropList (see page 23350)

SetPropValue

PPropInfo (see page 23381)

166.2.37 TypInfo.SetSetProp Function

Sets the value of a component property that is a set type.

Pascal

```
procedure SetSetProp(Instance: TObject; const PropName: string; const Value: string); overload;
procedure SetSetProp(Instance: TObject; PropInfo: PPropInfo; const Value: string); overload;
```

C++

```
SetSetProp(TObject * Instance, const AnsiString PropName, const AnsiString Value);
SetSetProp(TObject * Instance, PPropInfo PropInfo, const AnsiString Value);
```

File

TypInfo

Description

SetSetProp utilises Delphi's RTTI (Run Time Type Information) to set the value of a component's property where that property is a set type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**. You specify the set of values as a comma separated string. You may optionally enclose these in square brackets. This allows the returned value from one GetSetProp (see page 23352) call to be used as the assignment value in a SetSetProp call.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 23365) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 23351) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetSetProp (see page 23352)

GetPropInfo (see page 23348)

GetPropList (see page 23350)

SetPropValue (see page 23365)

PPropInfo (see page 23381)

166.2.38 TypInfo.SetStrProp Function

Sets the value of a component property that is a string type.

Pascal

```
procedure SetStrProp(Instance: TObject; const PropName: string; const Value: string); overload;
procedure SetStrProp(Instance: TObject; PropInfo: PPropInfo; const Value: string); overload;
```

C++

```
SetStrProp(TObject * Instance, const AnsiString PropName, const AnsiString Value);
SetStrProp(TObject * Instance, PPropInfo PropInfo, const AnsiString Value);
```

File

TypInfo

Description

SetStrProp utilises Delphi's RTTI (Run Time Type Information) to set the value of a component's property where that property is a String, LongString or WideString type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 23365) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 23351) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetStrProp (see page 23353)

GetPropInfo (see page 23348)

GetPropList (see page 23350)

SetPropValue (see page 23365)

PPropInfo (see page 23381)

166.2.39 TypInfo.SetToString Function

Returns one or more of the set values of a component's property.

Pascal

```
function SetToString(PropInfo: PPropInfo; Value: Integer; Brackets: Boolean = False): string;
```

C++

```
AnsiString SetToString(PPropInfo PropInfo, int Value, Boolean Brackets = False);
```

File

TypInfo

Description

SetToString utilises Delphi's RTTI (Run Time Type Information) to retrieve one or more of the values of a component property that is of the set type.

The property to interrogate is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

Value defines a bit mask that selects the possible values of this property. Each bit corresponds to a set value. The low order bit corresponds to the first set value. A string is returned with the selected set values as a comma separated value string. Note that the current property set values are not taken into account - the set values selected by **Value** are those in the complete set.

You may optionally set **Brackets** true so as to enclose this returned string within square brackets, as shown in the IDE for the property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetSetProp (see page 23352)

GetPropInfo (see page 23348)

GetPropList (see page 23350)

PPropInfo (see page 23381)

SetPropValue (see page 23365)

SetSetProp (see page 23366)

166.2.40 TypInfo.SetVariantProp Function

Sets the value of a component property that is a variant type.

Pascal

```
procedure SetVariantProp(Instance: TObject; const PropName: string; const Value: Variant);
overload;
procedure SetVariantProp(Instance: TObject; PropInfo: PPropInfo; const Value: Variant);
overload;
```

C++

```
SetVariantProp(TObject * Instance, const AnsiString PropName, const Variant Value);
SetVariantProp(TObject * Instance, PPropInfo PropInfo, const Variant Value);
```

File

TypInfo

Description

SetVariantProp utilises Delphi's RTTI (Run Time Type Information) to set the value of a component's property where that

property is a variant type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 23365) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 23351) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetVariantProp (see page 23354)

GetPropInfo (see page 23348)

GetPropList (see page 23350)

SetPropValue (see page 23365)

PPropInfo (see page 23381)

166.2.41 TypInfo.SetWideStrProp Function

Sets the value of a component property that is a WideString type.

Pascal

```
procedure SetWideStrProp(Instance: TObject; const PropName: string; const Value: WideString);
overload;
procedure SetWideStrProp(Instance: TObject; PropInfo: PPropInfo; const Value: WideString);
overload;
```

C++

```
SetWideStrProp(TObject * Instance, const AnsiString PropName, const BSTR Value);
SetWideStrProp(TObject * Instance, PPropInfo PropInfo, const BSTR Value);
```

File

TypInfo

Description

SetWideStrProp utilises Delphi's RTTI (Run Time Type Information) to set the value of a component's property where that property is a WideString type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name.

This allows mass visual component processing at run time.

Use SetPropValue (see page 23365) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 23351) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 23331) exception is thrown.

See Also

GetWideStrProp (see page 23355)

GetPropInfo (see page 23348)

GetPropList (see page 23350)

SetPropValue (see page 23365)

PPropInfo (see page 23381)

166.2.42 TypInfo.SortPropList Function

Sorts a list of component properties into alphanumeric sequence.

Pascal

```
procedure SortPropList(PropList: PPropList; PropCount: Integer);
```

C++

```
SortPropList(PPropList PropList, int PropCount);
```

File

TypInfo

Description

SortPropList is used to sort a PPropList (see page 23382) that was originally created in natural rather than sorted sequence (by GetPropList (see page 23350)).

PropList defines a pointer to the array of TPropInfo (see page 23376) pointers (normally obtained using GetPropList (see page 23350)). **Count** specifies the number of TPropList (see page 23387)

See Also

GetPropList (see page 23350)

PTypeInfo (see page 23383)

PPropList (see page 23382)

PPropInfo (see page 23381)

TPropInfo (see page 23376)

166.2.43 TypInfo.StringToSet Function

Returns a bit mask of set values for a set type component property.

Pascal

```
function StringToSet(PropInfo: PPropInfo; const Value: string): Integer;
```

C++

```
int StringToSet(PPropInfo PropInfo, const AnsiString Value);
```

File

TypInfo

Description

StringToSet utilises Delphi's RTTI (Run Time Type Information) to return a bit mask of set indices for component's property where that property is a set type.

The property is defined by the **PropInfo** pointer. (GetPropInfo (see page 23348) and GetPropList (see page 23350) can be used to obtain such a record).

Value defines a comma separated string of set values the property may have.

The resulting integer value is a bit mask. The 0 bit relates to the 1st set value for the property. If that property value is included in the **Value** string, it is set to 1, otherwise 0. This continues for bits 1, 2 and so on referring to set positions 2, 3 and so on.

See Also

GetPropList (see page 23350)

SetToString (see page 23367)

PPropInfo (see page 23381)

TPropInfo (see page 23376)

166.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TFloatType (see page 23372)	This is record TypInfo.TFloatType.
	TIntfFlag (see page 23373)	This is record TypInfo.TIntfFlag.
	TMethodKind (see page 23373)	Enumerates floating point number types in the TTypeData (see page 23377) record.
	TOrdType (see page 23374)	Enumerates floating point number types in the TTypeData (see page 23377) record.
	TParamFlag (see page 23375)	TParamFlag and TParamFlags (see page 23386) indicates the attributes of a method parameter.
	TTypeKind (see page 23379)	These types describe a type that is manipulated in the forms designer.

Records

	Name	Description
	TPropData (see page 23376)	This is record TypInfo.TPropData.
	TPropInfo (see page 23376)	TPropInfo is a record of component property type information.
	TTypeData (see page 23377)	TTypeData is used internally to represent type information.
	TTypeInfo (see page 23379)	Holds information for a data type as returned by TypeInfo.

166.3.1 TypInfo.TFloatType Enumeration

Pascal

```
TFloatType = (
  ftSingle,
  ftDouble,
  ftExtended,
  ftComp,
  ftCurr
);
```

C++

```
enum TFloatType {
  ftSingle,
  ftDouble,
  ftExtended,
  ftComp,
  ftCurr
};
```

File

TypInfo

Description

This is record TypInfo.TFloatType.

166.3.2 TypInfo.TIntfFlag Enumeration

Pascal

```
TIntfFlag = (
  ifHasGuid,
  ifDispInterface,
  ifDispatch
);
```

C++

```
enum TIntfFlag {
  ifHasGuid,
  ifDispInterface,
  ifDispatch
};
```

File

TypInfo

Description

This is record TypInfo.TIntfFlag.

166.3.3 TypInfo.TMethodKind Enumeration

Enumerates floating point number types in the TTypeData (see page 23377) record.

Pascal

```
TMethodKind = (
  mkProcedure,
  mkFunction,
  mkConstructor,
  mkDestructor,
  mkClassProcedure,
  mkClassFunction,
  mkClassConstructor,
  mkOperatorOverload,
  mkSafeProcedure,
  mkSafeFunction
);
```

C++

```
enum TMethodKind {
  mkProcedure,
  mkFunction,
  mkConstructor,
  mkDestructor,
  mkClassProcedure,
  mkClassFunction,
  mkClassConstructor,
  mkOperatorOverload,
  mkSafeProcedure,
  mkSafeFunction
};
```

File

TypInfo

Description

The GetTypeData (see page 23354) routine utilises Delphi's RTTI (Run Time Type Information) to return a pointer to TTypeData (see page 23377) record that describes a component property data type.

TTypeData (see page 23377) is a variant record : the layout depends on the property TTypeKind (see page 23379) value. When it is tkMethod, the data is one of the TMethodKind enumerated values.

The TMethodKind value meanings are as follows :

Value	Meaning
mkProcedure	Stand alone procedure.
mkFunction	Stand alone function.
mkDestructor	Destructor.
mkConstructor	Constructor.
mkClassProcedure	Class procedure.
mkClassFunction	Class function.
mkClassConstructor	Class constructor.
mkOperatorOverload	Operator overload.
mkSafeProcedure	Safe procedure.
mkSafeFunction	Safe function.

See Also

GetTypeData (see page 23354)

TTypeData (see page 23377)

TTypeKind (see page 23379)

166.3.4 TypInfo.TOrdType Enumeration

Enumerates floating point number types in the TTypeData (see page 23377) record.

Pascal

```
TOrdType = (
  otSByte,
  otUByte,
  otSWord,
  otUWord,
  otSLong,
  otULong
);
```

C++

```
enum TOrdType {
  otSByte,
  otUByte,
  otSWord,
```

```
    otUWord,
    otSLong,
    otULong
  };
```

File

TypInfo

Description

The GetTypeData (see page 23354) routine utilises Delphi's RTTI (Run Time Type Information) to return a pointer to TTypeData (see page 23377) record that describes a component property data type.

TTypeData (see page 23377) is a variant record : the layout depends on the property TTypeKind (see page 23379) value. When it is tkWChar, the data is one of the TOrdType enumerated values.

The TOrdType value meanings are as follows :

Value	Meaning
otSByte	Signed byte.
otUByte	Unsigned byte
otSWord	Signed Word
otUWord	Unsigned Word
otSLong	Signed Long
otULong	Unsigned Long

See Also

GetTypeData (see page 23354)

TTypeData (see page 23377)

TTypeKind (see page 23379)

166.3.5 TypInfo.TParamFlag Enumeration

TParamFlag and TParamFlags (see page 23386) indicates the attributes of a method parameter.

Pascal

```
  TParamFlag = (
    pfVar,
    pfConst,
    pfArray,
    pfAddress,
    pfReference,
    pfOut
  );
```

C++

```
  enum TParamFlag {
    pfVar,
    pfConst,
    pfArray,
    pfAddress,
    pfReference,
    pfOut
```

```
};
```

File

TypInfo

Description

TParamFlags (see page 23386) indicates the attributes of a parameter on a Delphi language function or procedure. It is a set of TParamFlag values:

Value	Meaning
pfVar	The parameter is input/output. Caller must supply a variable to receive output.
pfConst	The parameter can't be changed by the function or procedure.
pfArray	The parameter represents an open array of values.
pfAddress	The parameter is a memory address (pointer).
pfReference	The parameter is passed by reference.
pfOut	The parameter is output. Caller supplies a variable to receive the output.

166.3.6 TypInfo.TPropData Record

Pascal

```
TPropData = packed record
  PropCount: Word;
  PropList: record;
end;
```

C++

```
struct TPropData {
  Word PropCount;
  record PropList;
};
```

File

TypInfo

Description

This is record TypInfo.TPropData.

166.3.7 TypInfo.TPropInfo Record

TPropInfo is a record of component property type information.

Pascal

```
TPropInfo = packed record
  PropType: PPTypeInfo;
  GetProc: Pointer;
  SetProc: Pointer;
  StoredProc: Pointer;
  Index: Integer;
  Default: Longint;
```

```
    NameIndex: SmallInt;
    Name: ShortString;
  end;
```

C++

```
  struct TPropInfo {
    PPTypeInfo PropType;
    void * GetProc;
    void * SetProc;
    void * StoredProc;
    int Index;
    Longint Default;
    SmallInt NameIndex;
    ShortString Name;
  };
```

File

TypInfo

Description

TPropInfo is principally used to hold the results of a call to GetPropInfo (see page 23348). It provides meta details about a component property as part of Delphi's RTTI (Run Time Type Information) mechanism, giving run time component property processing. This allows for more generalised component handling that avoids excessive hard coded values.

The TPropInfo fields have the foillowing meanings :

Field	Meaning
PropType	Points to a data structure that gives information about the property type.
GetProc	Method pointer for the method that gets the component property value.
SetProc	Method pointer for the method that sets the component property value.
Index	The index of the property. It is used as a parameter to the GetProc and SetProc methods on indexed properties.
Default	The default value of the property.
NameIndex	The offset into the Name field where the property's name starts.
Name	The qualified property name.

See Also

GetPropInfo (see page 23348)

PPropInfo (see page 23381)

166.3.8 TypInfo.TTypeData Record

TTypeData is used internally to represent type information.

Pascal

```
  TTypeData = packed record
    case TTypeKind of
      tkClass: (ClassType: TClass;
                ParentInfo: PPTypeInfo;
                PropCount: SmallInt;
                UnitName: ShortStringBase;);
      tkDynArray: (elSize: Longint;
```

```
                    elType: PPTypeInfo;
                    varType: Integer;
                    elType2: PPTypeInfo;
                    DynUnitName: ShortStringBase;));
      tkEnumeration: (BaseType: PPTypeInfo;
                       NameList: ShortStringBase;
                       EnumUnitName: ShortStringBase;));
      tkFloat: (FloatType: TFloatType;));
      tkInt64: (MaxInt64Value: Int64;
                MinInt64Value: Int64;));
      tkMethod: (MethodKind: TMethodKind;
                 ParamCount: Byte;
                 ParamList: array[0..1023] of Char );
                 tkInterface: ( IntfParent : PPTypeInfo; IntfFlags : TIntfFlagsBase; Guid :
  TGUID; IntfUnit : ShortStringBase;;));
      tkSet: (CompType: PPTypeInfo;));
      tkString: (MaxLength: Byte;));
      tkWChar: (OrdType: TOrdType;
                MinValue: Longint;
                MaxValue: Longint;));
  end;
```

C++

```
  struct TTypeData {
    TOrdType OrdType;
    Longint MinValue;
    Longint MaxValue;
    PPTypeInfo BaseType;
    ShortStringBase NameList;
    ShortStringBase EnumUnitName;
    PPTypeInfo CompType;
    TFloatType FloatType;
    Byte MaxLength;
    TClass ClassType;
    PPTypeInfo ParentInfo;
    SmallInt PropCount;
    ShortStringBase UnitName;
    TMethodKind MethodKind;
    Byte ParamCount;
    array[0..1023] of Char ) ParamList;
    ( IntfParent : PPTypeInfo; IntfFlags : TIntfFlagsBase; Guid : TGUID; IntfUnit :
  ShortStringBase; tkInterface;
    Int64 MaxInt64Value;
    Int64 MinInt64Value;
    Longint elSize;
    PPTypeInfo elType;
    int varType;
    PPTypeInfo elType2;
    ShortStringBase DynUnitName;
  };
```

File

TypInfo

Description

A TTypeData structure represents a category of properties, events, or methods in parameters passed to several routines in the TypInfo (see page 23327) unit.

166.3.9 TypInfo.TTypeInfo Record

Holds information for a data type as returned by TypeInfo.

Pascal

```
TTypeInfo = record
  Kind: TTypeKind;
  Name: ShortString;
end;
```

C++

```
struct TTypeInfo {
  TTypeKind Kind;
  ShortString Name;
};
```

File

TypInfo

Description

The TypeInfo function is part of Delphi's Run Time Type Information (RTTI) mechanism. It provides run time information about a specific data type. TypeInfo returns a PTypeInfo (see page 23383) pointer to a TTypeInfo value.

The TTypeInfo holds the following fields :

Field	Meaning
Kind	The kind of type in RTTI terms. This value is required to select the appropriate variant section in the TTypeData (see page 23377) returned by the GetTypeData (see page 23354).
Name	The name of the data type.

See Also

GetTypeData (see page 23354)

PTypeInfo (see page 23383)

PPTypeInfo (see page 23382)

TTypeKind (see page 23379)

TTypeData (see page 23377)

166.3.10 TypInfo.TTypeKind Enumeration

These types describe a type that is manipulated in the forms designer.

Pascal

```
TTypeKind = (
  tkUnknown,
  tkInteger,
  tkChar,
  tkEnumeration,
  tkFloat,
```

```
    tkString,
    tkSet,
    tkClass,
    tkMethod,
    tkWChar,
    tkLString,
    tkWString,
    tkVariant,
    tkArray,
    tkRecord,
    tkInterface,
    tkInt64,
    tkDynArray
  );
```

C++

```
  enum TTypeKind {
    tkUnknown,
    tkInteger,
    tkChar,
    tkEnumeration,
    tkFloat,
    tkString,
    tkSet,
    tkClass,
    tkMethod,
    tkWChar,
    tkLString,
    tkWString,
    tkVariant,
    tkArray,
    tkRecord,
    tkInterface,
    tkInt64,
    tkDynArray
  };
```

File

TypInfo

Description

TTypeKind is an enumerated type that classifies a type.

TTypeInfo (see page 23379) describes a type by giving its classification and its name.

PTypeInfo (see page 23383) is a pointer to a TTypeInfo (see page 23379) value.

PPTypeInfo (see page 23382) is a pointer to a PTypeInfo (see page 23383) pointer.

166.4 Types

The following table lists types in this documentation.

Types

Name	Description
PPropInfo (see page 23381)	PPropInfo points to a TPropInfo (see page 23376) record that holds component property information.
PPropList (see page 23382)	Defines a pointer to a TPropList (see page 23387) array.
PPTypeInfo (see page 23382)	Holds a pointer to a pointer to information for a data type as returned by TypeInfo.
PTypeData (see page 23383)	PTypeData points to a TTypeData (see page 23377) record that holds component property type information.
PTypeInfo (see page 23383)	Holds a pointer to information for a data type as returned by TypeInfo.
ShortStringBase (see page 23384)	Defines a fixed length string for general use in the TTypeData (see page 23377) record.
TIntfFlags (see page 23384)	Enumerates interface flag types in the TTypeData (see page 23377) record.
TIntfFlagsBase (see page 23385)	Set of interface flag types in the TTypeData (see page 23377) record.
TParamFlags (see page 23386)	This is type TypInfo.TParamFlags.
TParamFlagsBase (see page 23386)	This is type TypInfo.TParamFlagsBase.
TPropInfoProc (see page 23386)	This is type TypInfo.TPropInfoProc.
TPropList (see page 23387)	Defines an array of PPropInfo (see page 23381) values.
TTypeKinds (see page 23387)	Defines a set of TTypeKind (see page 23379) values.

166.4.1 TypInfo.PPropInfo Type

PPropInfo points to a TPropInfo (see page 23376) record that holds component property information.

Pascal

```
PPropInfo = ^TPropInfo;
```

C++

```
^TPropInfo PPropInfo;
```

File

TypInfo

Description

TPropInfo (see page 23376) is principally used to hold the results of a call to GetPropInfo (see page 23348). It provides meta details about a component property as part of Delphi's RTTI (Run Time Type Information) mechanism, giving run time component property processing. This allows for more generalised component handling that avoids excessive hard coded values. PPropInfo is a pointer to a TPropInfo (see page 23376) record.

See Also

GetPropInfo (see page 23348)

TPropInfo (see page 23376)

166.4.2 TypInfo.PPropList Type

Defines a pointer to a TPropList (see page 23387) array.

Pascal

```
PPropList = ^TPropList;
```

C++

```
^TPropList PPropList;
```

File

TypInfo

Description

PPropList defines a pointer to an array of PPropInfo (see page 23381) values. This TPropList (see page 23387) array is principally used to hold the results of a call to GetPropList (see page 23350). This list (array) holds pointers to TPropInfo (see page 23376) records that describe the nature of the properties of the component provided to GetPropList (see page 23350).

See Also

GetPropList (see page 23350)

PPropInfo (see page 23381)

TPropInfo (see page 23376)

TPropList (see page 23387)

166.4.3 TypInfo.PPTypeInfo Type

Holds a pointer to a pointer to information for a data type as returned by TypeInfo.

Pascal

```
PPTypeInfo = ^PTypeInfo;
```

C++

```
^PTypeInfo PPTypeInfo;
```

File

TypInfo

Description

The TypeInfo function is part of Delphi's Run Time Type Information (RTTI) mechanism. It provides run time information about a specific data type. TypeInfo returns a PTypeInfo (see page 23383) pointer to a TTypeInfo (see page 23379) value. PPTypeInfo is a PTypeInfo (see page 23383) pointer.

See Also

GetTypeData (see page 23354)

PTypeInfo (see page 23383)

TTypeInfo (see page 23379)

TTypeKind (see page 23379)

TTypeData (see page 23377)

166.4.4 TypInfo.PTypeData Type

PTypeData points to a TTypeData (see page 23377) record that holds component property type information.

Pascal

```
PTypeData = ^TTypeData;
```

C++

```
^TTypeData PTypeData;
```

File

TypInfo

Description

TTypeData (see page 23377) is principally used to hold the results of a call to GetTypeData (see page 23354). It provides meta details about a component property type as part of Delphi's RTTI (Run Time Type Information) mechanism, giving run time component property processing. This allows for more generalised component handling that avoids excessive hard coded values. PTypeData is a pointer to a TTypeData (see page 23377) record.

See Also

GetTypeData (see page 23354)

TTypeData (see page 23377)

166.4.5 TypInfo.PTypeInfo Type

Holds a pointer to information for a data type as returned by TypeInfo.

Pascal

```
PTypeInfo = ^TTypeInfo;
```

C++

```
^TTypeInfo PTypeInfo;
```

File

TypInfo

Description

The TypeInfo function is part of Delphi's Run Time Type Information (RTTI) mechanism. It provides run time information about a specific data type. TypeInfo returns a PTypeInfo pointer to a TTypeInfo (see page 23379) value.

The TTypeInfo (see page 23379) holds the following fields :

Field	Meaning
Kind	The kind of type in RTTI terms. This value is required to select the appropriate variant section in the TTypeData (see page 23377) returned by the GetTypeData (see page 23354).
Name	The name of the data type.

See Also

GetTypeData (see page 23354)

PPTypeInfo (see page 23382)

TTypeInfo (see page 23379)

TTypeKind (see page 23379)

TTypeData (see page 23377)

166.4.6 TypInfo.ShortStringBase Type

Defines a fixed length string for general use in the TTypeData (see page 23377) record.

Pascal

```
ShortStringBase = string[255];
```

C++

```
string[255] ShortStringBase;
```

File

TypInfo

Description

The GetTypeData (see page 23354) routine utilises Delphi's RTTI (Run Time Type Information) to return a pointer to TTypeData (see page 23377) record that describes a component property data type.

A number of the variant data sections in the TTypeData (see page 23377) record use the ShortStringBase to define a 255 character string to hold Name values such as UnitName for a class data type.

See Also

GetTypeData (see page 23354)

TTypeData (see page 23377)

166.4.7 TypInfo.TIntfFlags Type

Enumerates interface flag types in the TTypeData (see page 23377) record.

Pascal

```
TIntfFlags = set of TIntfFlag;
```

C++

```
TIntfFlag TIntfFlags;
```

File

TypInfo

Description

The GetTypeData (see page 23354) routine utilises Delphi's RTTI (Run Time Type Information) to return a pointer to TTypeData (see page 23377) record that describes a component property data type.

TTypeData (see page 23377) is a variant record : the layout depends on the property TTypeKind (see page 23379) value. When it is tkInterface, the data contains a TIntfFlagsBase (see page 23385) set of TIntfFlags values.

The TIntfFlags value meanings are as follows :

Value	Meaning
ifHasGuid	Interface has a GUID (Globally Unique IDentifier).
ifDispInterface	Is a dispatch interface.
ifDispatch	Can be dispatched.

See Also

GetTypeData (see page 23354)

TIntfFlagsBase (see page 23385)

TTypeData (see page 23377)

TTypeKind (see page 23379)

166.4.8 TypInfo.TIntfFlagsBase Type

Set of interface flag types in the TTypeData (see page 23377) record.

Pascal

```
TIntfFlagsBase = set of TIntfFlag;
```

C++

```
TIntfFlag TIntfFlagsBase;
```

File

TypInfo

Description

The GetTypeData (see page 23354) routine utilises Delphi's RTTI (Run Time Type Information) to return a pointer to TTypeData (see page 23377) record that describes a component property data type.

TTypeData (see page 23377) is a variant record : the layout depends on the property TTypeKind (see page 23379) value. When it is tkInterface, the data contains a TIntfFlagsBase set of TIntfFlag (see page 23373) values.

The TIntfFlag (see page 23373) value meanings are as follows :

Value	Meaning
ifHasGuid	Interface has a GUID (Globally Unique IDentifier).
ifDispInterface	Is a dispatch interface.

ifDispatch	Can be dispatched.

See Also

GetTypeData (see page 23354)

TIntfFlag (see page 23373)

TTypeData (see page 23377)

TTypeKind (see page 23379)

166.4.9 TypInfo.TParamFlags Type

Pascal

```
TParamFlags = set of TParamFlag;
```

C++

```
TParamFlag TParamFlags;
```

File

TypInfo

Description

This is type TypInfo.TParamFlags.

166.4.10 TypInfo.TParamFlagsBase Type

Pascal

```
TParamFlagsBase = set of TParamFlag;
```

C++

```
TParamFlag TParamFlagsBase;
```

File

TypInfo

Description

This is type TypInfo.TParamFlagsBase.

166.4.11 TypInfo.TPropInfoProc Type

Pascal

```
TPropInfoProc = procedure (PropInfo: PPropInfo) of object;
```

C++

```
(PropInfo: PPropInfo) ( TPropInfoProc)();
```

File

TypInfo

Description

This is type TypInfo.TPropInfoProc.

166.4.12 TypInfo.TPropList Type

Defines an array of PPropInfo (see page 23381) values.

Pascal

```
TPropList = array[0..16379] of PPropInfo;
```

C++

```
array[0..16379] of PPropInfo TPropList;
```

File

TypInfo

Description

TPropList is principally used to hold the results of a call to GetPropList (see page 23350). This list (array) holds pointers to TPropInfo (see page 23376) records that describe the nature of the properties of the component provided to GetPropList (see page 23350).

See Also

GetPropList (see page 23350)

TPropInfo (see page 23376)

166.4.13 TypInfo.TTypeKinds Type

Defines a set of TTypeKind (see page 23379) values.

Pascal

```
TTypeKinds = set of TTypeKind;
```

C++

```
TTypeKind TTypeKinds;
```

File

TypInfo

Description

TTypeKinds defines a set of TTypeKind (see page 23379) values. It is used to select which component property types to use in various routines, such as GetPropInfo (see page 23348).

See Also

GetPropInfo (see page 23348)

TTypeKind (see page 23379)

166.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
BooleanIdents (see page 23388)	An array of string representations of true and false.
DotSep (see page 23388)	This is variable TypInfo.DotSep.

166.5.1 TypInfo.BooleanIdents Variable

An array of string representations of true and false.

Pascal

```
BooleanIdents: array [Boolean] of string = ('False', 'True');
```

C++

```
array [Boolean] of string BooleanIdents = ('False', 'True');
```

File

TypInfo

Description

The GetEnumName (see page 23341) routine is part of Delphi's RTTI (Run Time Type Information). It returns a string representation of an enumerated type value. If that type is true or false, BooleanIdents is used internally.

See Also

GetEnumName (see page 23341)

166.5.2 TypInfo.DotSep Variable

Pascal

```
DotSep: string = '.';
```

C++

```
AnsiString DotSep = '.';
```

File

TypInfo

Description

This is variable TypInfo.DotSep.

166.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
tkAny (see page 23389)	This is constant TypInfo.tkAny.
tkMethods (see page 23389)	This is constant TypInfo.tkMethods.
tkProperties (see page 23389)	This is constant TypInfo.tkProperties.

166.6.1 TypInfo.tkAny Constant

Pascal

```
tkAny = [Low(TTypeKind)..High(TTypeKind)];
```

C++

```
tkAny = [Low(TTypeKind)..High(TTypeKind)];
```

File

TypInfo

Description

This is constant TypInfo.tkAny.

166.6.2 TypInfo.tkMethods Constant

Pascal

```
tkMethods = [tkMethod];
```

C++

```
tkMethods = [tkMethod];
```

File

TypInfo

Description

This is constant TypInfo.tkMethods.

166.6.3 TypInfo.tkProperties Constant

Pascal

```
tkProperties = tkAny - tkMethods - [tkUnknown];
```

C++

```
tkProperties = tkAny - tkMethods - [tkUnknown];
```

File

TypInfo

Description

This is constant TypInfo.tkProperties.

167 UDDIHelper Namespace

This is namespace UDDIHelper.

Topics

Name	Description
Functions (see page 23392)	The following table lists functions in this documentation.

Functions

	Name	Description
	GetBindingkeyAccessPoint (see page 23392)	Returns the URL for a Web Service as specified in a UDDI browser.

167.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetBindingkeyAccessPoint (see page 23392)	Returns the URL for a Web Service as specified in a UDDI browser.

167.1.1 UDDIHelper.GetBindingkeyAccessPoint Function

Returns the URL for a Web Service as specified in a UDDI browser.

Pascal

```
function GetBindingkeyAccessPoint(const Operator: String; const key: String): String;
```

C++

```
AnsiString GetBindingkeyAccessPoint(const AnsiString Operator, const AnsiString key);
```

File

UDDIHelper

Description

Use GetBindingkeyAccessPoint to fetch the URL for calling a Web Service from a specified UDDI registry.

Operator is the URL of the UDDI registry from which to fetch the entry. You can obtain this value for a registered invokable interface by calling the invocation registry's GetUDDIInfo method.

BindingKey is the unique identifier for the TModel that describes the Web Service for which you want the URL.

GetBindingkeyAccessPoint returns the URL for the specified Web Service. This can be used to set the URL property of a THTTPRIO component.

See Also

THTTPRIO

TInvokableClassRegistry

168 ValEdit Namespace

This is namespace ValEdit.

Topics

Name	Description
Classes (see page 23394)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 23509)	The following table lists structs, records, enums in this documentation.
Types (see page 23511)	The following table lists types in this documentation.

Classes

	Name	Description
	TItemProp (see page 23394)	TItemProp configures how users can edit the item values in a value list editor.
	TValueListEditor (see page 23401)	TValueListEditor provides a simple way to edit string lists that contain name/value pairs.
	TValueListStrings (see page 23498)	This is class ValEdit.TValueListStrings.

Structs, Records, Enums

	Name	Description
	TDisplayOption (see page 23509)	TDisplayOption and TDisplayOptions (see page 23511) control the appearance of a value list editor.
	TKeyOption (see page 23510)	TKeyOption and TKeyOptions (see page 23512) indicate the response of a value list editor to certain keystrokes.

Types

Name	Description
TDisplayOptions (see page 23511)	This is type ValEdit.TDisplayOptions.
TGetPickListEvent (see page 23511)	This is type ValEdit.TGetPickListEvent.
TItemProps (see page 23512)	This is type ValEdit.TItemProps.
TKeyOptions (see page 23512)	This is type ValEdit.TKeyOptions.
TOnValidateEvent (see page 23512)	This is type ValEdit.TOnValidateEvent.

168.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TItemProp (see page 23394)	TItemProp configures how users can edit the item values in a value list editor.
	TValueListEditor (see page 23401)	TValueListEditor provides a simple way to edit string lists that contain name/value pairs.
	TValueListStrings (see page 23498)	This is class ValEdit.TValueListStrings.

168.1.1 TItemProp Class

TItemProp configures how users can edit the item values in a value list editor.

Class Hierarchy

Pascal

```
TItemProp = class(TPersistent);
```

C++

```
class TItemProp : public TPersistent;
```

File

ValEdit

Description

For each item that it allows users to edit, the TValueListEditor (see page 23401) component maintains a TItemProp object. This object's properties govern how users can edit the value in the value list editor.

TItemProp objects are created and owned by the value list editor that uses them to represent properties of item values (such as an edit mask or maximum length) or to supply a fixed pick list of values that users can assign to the item.

See Also

TValueListEditor (see page 23401)

TValueListEditor (see page 23401)

168.1.1.1 TItemProp Members

The following tables list the members exposed by TItemProp.

Protected Methods

	Name	Description
	AssignTo (see page 23396)	Copies the properties of an object to a destination object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	GetOwner (see page 2932)	Returns the owner of an object.
	UpdateEdit (see page 23397)	This is UpdateEdit, a member of class TItemProp.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 23395)	Creates an instance of TItemProp (see page 23394).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 23396)	Destroys the TPersistent instance and frees its memory.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	HasPickList (see page 23397)	Indicates whether the PickList (see page 23400) property provides a static set of values for this item.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Published Properties

	Name	Description
	EditMask (see page 23398)	Restricts the value that can be entered for the associated item.
	EditStyle (see page 23398)	Determines whether the cell that contains the associated value has a drop-down arrow or ellipsis button.
	KeyDesc (see page 23399)	Specifies the name of the item, which appears in the names column of the Value list editor.
	MaxLength (see page 23399)	Specifies the maximum number of characters that users can enter for the item's value.
	PickList (see page 23400)	Lists the items that appear in the drop-down pick list for the item.
	ReadOnly (see page 23400)	Indicates whether users can edit the item's value by typing in a new value.

168.1.1.2 TItemProp.Create Constructor

Creates an instance of TItemProp (see page 23394).

Pascal

```
constructor Create(AEditor: TValueListEditor);
```

C++

```
__fastcall TItemProp(TValueListEditor AEditor);
```

Description

Do not call Create directly in an application. TValueListEditor (see page 23401) creates TItemProp (see page 23394) instances as needed when you make assignments to its ItemProps property.

The AEditor parameter is the value list editor that creates the TItemProp (see page 23394) instance to represent the properties of one of its items.

See Also

ItemProps (see page 23445)

168.1.1.3 TItemProp.Destroy Destructor

Destroys the TPersistent instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TItemProp();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

Free (see page 22300)

Destroy (see page 22287)

168.1.1.4 TItemProp Methods

The methods of the TItemProp class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 23396)	Copies the properties of an object to a destination object.
	UpdateEdit (see page 23397)	This is UpdateEdit, a member of class TItemProp.

Public Methods

	Name	Description
	HasPickList (see page 23397)	Indicates whether the PickList (see page 23400) property provides a static set of values for this item.

168.1.1.4.1 TItemProp.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign (see page 2930)

EConvertError (see page 22724)

168.1.1.4.2 TItemProp.HasPickList Method

Indicates whether the PickList (see page 23400) property provides a static set of values for this item.

Pascal

```
function HasPickList: Boolean;
```

C++

```
__fastcall Boolean HasPickList();
```

Description

Call PickList (see page 23400) to determine whether this TItemProp (see page 23394) object supplies values using the PickList (see page 23400) property. HasPickList returns true if the PickList (see page 23400) property contains a nonempty list of values.

See Also

PickList (see page 23400)

168.1.1.4.3 TItemProp.UpdateEdit Method

Pascal

```
procedure UpdateEdit;
```

C++

```
__fastcall UpdateEdit();
```

Description

This is UpdateEdit, a member of class TItemProp.

168.1.1.5 TItemProp Properties

The properties of the TItemProp class are listed here.

Published Properties

	Name	Description
	EditMask (see page 23398)	Restricts the value that can be entered for the associated item.
	EditStyle (see page 23398)	Determines whether the cell that contains the associated value has a drop-down arrow or ellipsis button.
	KeyDesc (see page 23399)	Specifies the name of the item, which appears in the names column of the Value list editor.
	MaxLength (see page 23399)	Specifies the maximum number of characters that users can enter for the item's value.
	PickList (see page 23400)	Lists the items that appear in the drop-down pick list for the item.
	ReadOnly (see page 23400)	Indicates whether users can edit the item's value by typing in a new value.

168.1.1.5.1 TItemProp.EditMask Property

Restricts the value that can be entered for the associated item.

Pascal

```
property EditMask: string;
```

C++

```
__property AnsiString EditMask;
```

Description

Use EditMask to restrict the characters a user can enter. If the user attempts to enter a character that is not valid, the character is not accepted. Validation using the EditMask property is performed on a character-by-character basis.

A mask consists of three fields with semicolons separating the fields. The first part of the mask is the mask itself. The second part is the character that determines whether the literal characters of a mask are included as part of the data. The third part of the mask is the character used to represent unentered characters in the mask.

Setting EditMask to an empty string removes the mask.

For more information on using edit masks, see the TEditMask type.

See Also

Values (see page 23457)

OnSetEditText (see page 14936)

168.1.1.5.2 TItemProp.EditStyle Property

Determines whether the cell that contains the associated value has a drop-down arrow or ellipsis button.

Pascal

```
property EditStyle: TEditStyle;
```

C++

```
__property TEditStyle EditStyle;
```

Description

Set EditStyle to specify whether the cell in the value list editor that contains the associated value should allow users to drop down a pick list or to launch a dialog to edit the value.

When EditStyle is esSimple, the value cell is a simple edit box that lets users edit the associated value directly (unless the ReadOnly (see page 23400) property is true).

When EditStyle is esEllipsis, the value cell contains an ellipsis button that users can click to generate an OnEditButtonClick event. Typically, applications use this event to launch a dialog that edits the value.

When EditStyle is esPickList, the value cell contains a drop-down arrow that users can click to display a list of values from which to select. This value must be supplied by the PickList (see page 23400) property or in an OnGetPickList event handler (or supplied by the PickList (see page 23400) property and modified in an OnGetPickList event handler).

See Also

PickList (see page 23400)

OnGetPickList (see page 23477)

OnEditButtonClick (see page 23470)

ReadOnly (see page 23400)

KeyOptions (see page 23445)

DisplayOptions (see page 23438)

168.1.1.5.3 TItemProp.KeyDesc Property

Specifies the name of the item, which appears in the names column of the Value list editor.

Pascal

```
property KeyDesc: string;
```

C++

```
__property AnsiString KeyDesc;
```

Description

Set KeyDesc to specify a name for this item. This name appears in the names column of the Value list editor.

If ItemProp is not set, then the name that appears in the key column of the Value list editor is the corresponding value of the Keys property.

See Also

Keys (see page 23445)

168.1.1.5.4 TItemProp.MaxLength Property

Specifies the maximum number of characters that users can enter for the item's value.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be typed when assigning a value to the item. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing value, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered.

See Also

Values (see page 23457)

EditMask (see page 23398)

168.1.1.5.5 TItemProp.PickList Property

Lists the items that appear in the drop-down pick list for the item.

Pascal

```
property PickList: TStrings;
```

C++

```
__property TStrings PickList;
```

Description

Use PickList to supply the values that appear in the drop-down pick list for the associated item. PickList lets you supply a static set of items that appear in the drop-down list for the associated item when EditStyle (see page 23398) is esPickList. These items can be subsequently modified by an OnGetPickList event handler.

Note: PickList is only used when the EditStyle (see page 23398) property is esPickList. If EditStyle (see page 23398) is esPickList, and PickList does list a set of values, you must supply an OnGetPickList event handler to supply the items.

See Also

EditStyle (see page 23398)

OnGetPickList (see page 23477)

DropDownRows (see page 23440)

168.1.1.5.6 TItemProp.ReadOnly Property

Indicates whether users can edit the item's value by typing in a new value.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set ReadOnly to true to prevent users from typing in a new value for the associated item. For example, to force users to select item values only from a pick list, you can set the EditStyle (see page 23398) to esPickList and set ReadOnly to true. Similarly, to force users to enter values only using a property editor, you can set EditStyle (see page 23398) to esEllipsis and set ReadOnly to true.

See Also

EditStyle (see page 23398)

168.1.2 TValueListEditor Class

TValueListEditor provides a simple way to edit string lists that contain name/value pairs.

Class Hierarchy

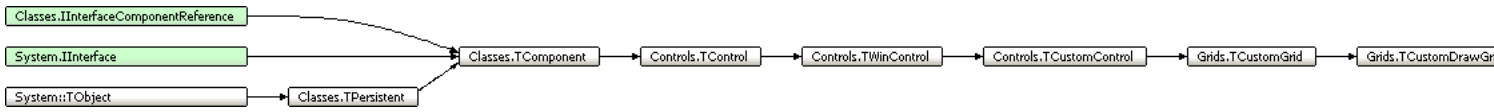

Pascal

```
TValueListEditor = class(TCustomDrawGrid);
```

C++

```
class TValueListEditor : public TCustomDrawGrid;
```

File

ValEdit

Description

Use TValueListEditor to let users edit string lists that consist of strings with the form Name=Value. By default, the names appear in a column headed "Key", and the values in a column headed "Value", although you can customize these labels. You can control whether users can select, add, or edit the strings in the Key column. For each entry in the Value column, you can let users edit the string, supply a drop-down list from which they can select values, or provide a dialog in which they can edit values.

See Also

TStringList (see page 3006)

TStrings (see page 3029)

Value List Editors (VCL Only)

168.1.2.1 TValueListEditor Members

The following tables list the members exposed by TValueListEditor.

Protected Events

	Name	Description
	OnAlignInsertBefore (see page 6213)	Occurs when an object with custom alignment is aligned.

	OnAlignPosition (see page 6214)	Occurs when an object with custom alignment is aligned.
	OnCanResize (see page 5851)	Occurs when an attempt is made to resize the control.
	OnColumnMoved (see page 14794)	Occurs immediately after the position of a column changes.
	OnConstrainedResize (see page 5853)	Adjust resize constraints.
	OnDockDrop (see page 6214)	Occurs when another control is docked to the control.
	OnDockOver (see page 6216)	Occurs when another control is dragged over the control.
	OnGetSiteInfo (see page 6223)	Returns the control's docking information.
	OnMouseWheel (see page 5869)	Occurs when the mouse wheel is rotated.
	OnResize (see page 5870)	Occurs immediately after the control is resized.
	OnUnDock (see page 6230)	Occurs when the application tries to undock a control that is docked to the windowed control.

Protected Fields

	Name	Description
	DesignOptionsBoost (see page 14814)	This is DesignOptionsBoost, a member of class TCustomGrid.
	FAnchorMove (see page 5733)	This is FAnchorMove, a member of class TControl.
	FAnchorOrigin (see page 5734)	This is FAnchorOrigin, a member of class TControl.
	FAnchorRules (see page 5734)	This is FAnchorRules, a member of class TControl.
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FDesignSize (see page 6124)	This is FDesignSize, a member of class TWinControl.
	FDoubleBuffered (see page 6124)	This is FDoubleBuffered, a member of class TWinControl.
	FExplicitHeight (see page 5734)	This is FExplicitHeight, a member of class TControl.
	FExplicitLeft (see page 5734)	This is FExplicitLeft, a member of class TControl.
	FExplicitTop (see page 5734)	This is FExplicitTop, a member of class TControl.
	FExplicitWidth (see page 5735)	This is FExplicitWidth, a member of class TControl.
	FGridState (see page 14814)	This is FGridState, a member of class TCustomGrid.
	FInImeComposition (see page 6125)	This is FInImeComposition, a member of class TWinControl.
	FOriginalParentSize (see page 5735)	This is FOriginalParentSize, a member of class TControl.
	FSaveCellExtents (see page 14814)	This is FSaveCellExtents, a member of class TCustomGrid.
	VirtualView (see page 14814)	This is VirtualView, a member of class TCustomGrid.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActionChange (see page 5737)	Responds when the action associated with the control changes.
	ActionChange (see page 6127)	Updates the control to reflect changes in its associated action.
	AddBiDiModeExStyle (see page 6128)	Sets control's window style according to its bi-directional support.
	AdjustClientRect (see page 6128)	Provides the interface that adjusts the ClientRect (see page 5810) property for idiosyncrasies of the window.
	AdjustColWidths (see page 23416)	This is AdjustColWidths, a member of class TValueListEditor.
	AdjustRowCount (see page 23416)	This is AdjustRowCount, a member of class TValueListEditor.
	AdjustSize (see page 5738)	Sets the size of the control as determined by the AutoSize (see page 5803) property.
	AdjustSize (see page 6129)	Adjusts the control's size according to its contents and constraints.
	AdjustSize (see page 14816)	Adds or deletes rows or columns from the grid.
	AlignControls (see page 6129)	Aligns any controls for which the control is the parent within a specified area of the control.
	ArrangeControl (see page 6130)	This is ArrangeControl, a member of class TWinControl.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	AssignTo (see page 5739)	Overrides the inherited AssignTo function to enable assignments to TAction objects.
	AssignTo (see page 6130)	Copies the properties of the windowed control to another object.
	BeginAutoDrag (see page 5739)	Begin Automatic Drag.
	BeginColumnDrag (see page 14817)	Starts the dragging of a column in the grid.
	BeginRowDrag (see page 14817)	Starts the dragging of a row in the grid.

	BoxRect (see page 14818)	Returns the screen coordinates of a specified set of cells.
	CalcCursorPos (see page 5741)	This is CalcCursorPos, a member of class TControl.
	CalcDrawInfo (see page 14818)	Calculates the pixel locations of the grid boundaries.
	CalcFixedInfo (see page 14818)	Calculates the pixel locations of the fixed cells.
V	CalcSizingState (see page 14819)	Determines the appearance of the mouse cursor during resizing.
V	CanAutoSize (see page 5741)	Indicates whether the control can resize itself automatically to accommodate its contents.
V	CanAutoSize (see page 6131)	Indicates how the control can be resized.
	CanEditAcceptKey (see page 14819)	Indicates whether the Key parameter can be entered into the current cell.
V	CanEditModify (see page 23417)	Indicates whether the current cell in the grid can be edited.
V	CanEditShow (see page 14820)	Indicates whether the inplace edit control can be created to allow editing.
	CanGridAcceptKey (see page 14820)	Indicates whether the grid should respond to a given key combination.
V	CanResize (see page 5742)	OnCanResize (see page 5851) event dispatcher.
V	CanResize (see page 6132)	Indicates how the control can respond to a resize attempt.
	CellRect (see page 14821)	Returns the screen coordinates of a cell in the grid.
	Changed (see page 5742)	Sends a CM_CHANGED (see page 6330) message to the control.
	ChangeGridOrientation (see page 14821)	Prepares the grid's canvas for painting to reflect a right-to-left or left-to-right orientation.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	ChangeScale (see page 5743)	Rescaling event dispatcher.
V	ChangeScale (see page 6132)	Resize (see page 5780) and reposition children and self.
	CheckColumnDrag (see page 14821)	CheckColumnDrag validates whether the column currently selected for dragging can be dropped at the current location.
	CheckRowDrag (see page 14822)	CheckRowDrag validates whether the row currently selected for dragging can be dropped at the current location.
	Click (see page 5743)	Respond to user click.
V	ColumnMoved (see page 14781)	Generates an OnColumnMoved (see page 14794) event.
	ColumnMoved (see page 14822)	Provides the interface for a method that responds when the position of a column changes.
V	ColWidthsChanged (see page 23417)	Responds when the column widths change.
V	ConstrainedResize (see page 5746)	Notify control of resize with constraints.
V	ConstrainedResize (see page 6133)	Generates an OnConstrainedResize (see page 5853) event.
	ControlsAligned (see page 6134)	This is ControlsAligned, a member of class TWinControl.
	CreateDockManager (see page 6134)	Creates the dock manager for the control.
V	CreateEditor (see page 23417)	Creates the inplace editor control.
V	CreateHandle (see page 6135)	Creates underlying screen object.
V	CreateParams (see page 6135)	Initializes a window-creation parameter data structure.
V	CreateParams (see page 14823)	Initializes the window-creation parameter record.
	CreateSubClass (see page 6136)	Creates a windowed control derived from an existing Windows window class.
V	CreateWindowHandle (see page 6137)	Creates a Windows control to represent the control.
V	CreateWnd (see page 23418)	Creates the underlying window.
V	CustomAlignInsertBefore (see page 6137)	Returns placement order of custom-aligned child control.
V	CustomAlignPosition (see page 6138)	Returns aligned position of custom-aligned child control.
	DblClick (see page 5747)	OnDblClick (see page 5854) event dispatcher.
	DefaultDockImage (see page 5747)	Handles the painting and erasing of a dock image.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 5748)	Provides methods to read and write the IsControl (see page 5829) property to a stream such as a form file.
V	DefineProperties (see page 6139)	Provides methods to read and write the IsControl property to a stream such as a form file.
V	DefineProperties (see page 14824)	Adds the ColWidths (see page 14845) and RowHeights (see page 14855) properties to the values saved with the grid.
V	DeleteColumn (see page 14824)	Deletes a specified column from the grid.
	DesignWndProc (see page 5749)	This is DesignWndProc, a member of class TControl.

	DestroyHandle (see page 6140)	Destroys the control's window without destroying the control.
	DestroyWindowHandle (see page 6140)	Destroys the window created in the CreateWindowHandle (see page 6137) method.
	DestroyWnd (see page 6141)	Destroys the control's window.
	DoAddDockClient (see page 6142)	Performs actions when a control is docked to the windowed control.
	DockOver (see page 6143)	Responds when a docking client is dragged over the windowed control.
	DockReplaceDockClient (see page 6143)	This is DockReplaceDockClient, a member of class TWinControl.
	DockTrackNoTarget (see page 5749)	Handles dragging of the control over an invalid docking target.
	DoContextPopup (see page 5750)	OnContextPopup (see page 5854) event dispatcher.
	DoDock (see page 5750)	Docking event dispatcher.
	DoDockOver (see page 6143)	Generates an OnDockOver (see page 6216) event.
	DoEndDock (see page 5750)	OnEndDock (see page 5859) event dispatcher.
	DoEndDrag (see page 5751)	OnEndDrag (see page 5860) event dispatcher.
	DoEnter (see page 6144)	Respond to receiving input focus.
	DoExit (see page 23419)	Hides the inplace editor when selection moves to another control in a form.
	DoFlipChildren (see page 6145)	Flips the control's children.
	DoKeyDown (see page 6145)	Performs some preprocessing before generating an OnKeyDown (see page 6225) event.
	DoKeyPress (see page 6146)	Performs some preprocessing before generating an OnKeyPress (see page 6227) event.
	DoKeyUp (see page 6146)	Performs some preprocessing before generating an OnKeyUp (see page 6228) event.
	DoMouseWheel (see page 5751)	Processes mouse wheel motion.
	DoMouseWheelDown (see page 5752)	OnMouseWheelDown (see page 5869) event dispatcher.
	DoMouseWheelDown (see page 14825)	Performs tasks when the mouse wheel is rotated downward.
	DoMouseWheelUp (see page 5752)	OnMouseWheelUp (see page 5870) event dispatcher.
	DoMouseWheelUp (see page 14826)	Performs tasks when the mouse wheel is rotated upward.
	DoOnValidate (see page 23419)	This is DoOnValidate, a member of class TValueListEditor.
	DoRemoveDockClient (see page 6147)	Responds when a control is undocked from the windowed control.
	DoStartDock (see page 5753)	OnStartDock (see page 5871) event dispatcher.
	DoStartDrag (see page 5753)	OnStartDrag (see page 5871) event dispatcher.
	DoUnDock (see page 6147)	Undocks a control that is currently docked to the windowed control.
	DragCanceled (see page 5754)	Provides the interface for a method that responds to the cancellation of a drag.
	DragOver (see page 5755)	OnDragOver (see page 5857) event dispatcher.
	DrawCell (see page 23419)	Generates an OnDrawCell event.
	DrawDragDockImage (see page 5755)	Draws the dock image for the control.
	EditListGetItems (see page 23420)	This is EditListGetItems, a member of class TValueListEditor.
	EndColumnDrag (see page 14827)	Ends the dragging of a column in the grid.
	EndRowDrag (see page 14827)	Ends the dragging of a row in the grid.
	EraseDragDockImage (see page 5757)	Erases the dock image for the control.
	FindNextControl (see page 6149)	Returns the control's next child in the tab order after the specified control.
	FixupTabList (see page 6149)	Sorts the child controls by their tab order.
	FocusCell (see page 14827)	Changes the current cell to a specified cell.
	GetAction (see page 5758)	This is GetAction, a member of class TControl.
	GetActionLinkClass (see page 5758)	Returns the appropriate action link class that is associated with the control class.
	GetActionLinkClass (see page 6152)	Returns the associated action link class.
	GetCell (see page 23420)	This is GetCell, a member of class TValueListEditor.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 6153)	Calls a specified method for each child of the control.
	GetClientOrigin (see page 5758)	Returns the screen coordinates of the top-left corner of the control's client area.
	GetClientOrigin (see page 6153)	Returns the value of the ClientOrigin (see page 5809) property.
	GetClientRect (see page 5759)	Returns a rectangle defining the client area of the control.
	GetClientRect (see page 6154)	Returns the value of the ClientRect (see page 5810) property.

	GetColCount (see page 23421)	This is GetColCount, a member of class TValueListEditor.
	GetControlExtents (see page 6154)	Returns the smallest rectangle in which all the control's children fit.
	GetDeviceContext (see page 5760)	Returns a device context for the control.
	GetDeviceContext (see page 6155)	Provides access to a device context for the control.
	GetDockEdge (see page 5763)	Returns the edge on which another control should be docked when it is dragged over the control.
	GetDragImages (see page 5763)	Returns the control's image list containing the image to be displayed while dragging.
	GetEditLimit (see page 23421)	Provides the interface for a method that limits the number of characters the user can type into cells.
	GetEditMask (see page 23421)	Generates an OnGetEditMask event.
	GetEditStyle (see page 23421)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
	GetEditText (see page 23422)	This is GetEditText, a member of class TValueListEditor.
	GetEnabled (see page 5763)	Returns the value of the Enabled (see page 5821) property.
	GetFloating (see page 5764)	Returns the value of the Floating (see page 5823) property.
	GetFloatingDockSiteClass (see page 5764)	Returns the class used for the control when it is floating.
	GetGridHeight (see page 14829)	Returns the value of the GridHeight (see page 14850) property.
	GetGridWidth (see page 14830)	Returns the value of the GridWidth (see page 14851) property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPalette (see page 5765)	Returns the handle to a palette for use by the control.
	GetParentHandle (see page 6157)	Returns the window handle for the window of the control's parent.
	GetPickList (see page 23422)	This is GetPickList, a member of class TValueListEditor.
	GetPopupMenu (see page 5765)	Returns the pop-up menu associated with the control.
	GetRowCount (see page 23422)	This is GetRowCount, a member of class TValueListEditor.
	GetSiteInfo (see page 6158)	Generates an OnGetSiteInfo (see page 6223) event
	GetTopParentHandle (see page 6159)	Returns the window handle of the first window not associated with a VCL control that contains the windowed control.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HideEdit (see page 14830)	Hides the inplace editor.
	HideEditor (see page 14830)	Hides the inplace editor when it is not in use.
	InvalidateCell (see page 14831)	Invalidates the region occupied by a cell so that it will be repainted with the next Windows paint message.
	InvalidateCol (see page 14831)	Invalidates the region occupied by a column so that it will be repainted with the next Windows paint message.
	InvalidateDockHostSite (see page 6162)	This is InvalidateDockHostSite, a member of class TWinControl.
	InvalidateEditor (see page 14832)	Causes the inplace editor to repaint itself, moving to a new location if necessary.
	InvalidateGrid (see page 14832)	Tells Windows to repaint the grid after other important Windows messages are handled.
	InvalidateRow (see page 14832)	Invalidates the region occupied by a row so that it will be repainted with the next Windows paint message.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsControlActivateMsg (see page 6162)	This is IsControlActivateMsg, a member of class TWinControl.
	IsControlMouseMsg (see page 6162)	Indicates whether a specified mouse message is directed to one of the windowed control's child controls.
	IsEmptyRow (see page 23423)	This is IsEmptyRow, a member of class TValueListEditor.
	IsQualifyingSite (see page 6163)	This is IsQualifyingSite, a member of class TWinControl.
	KeyDown (see page 23423)	Provides special processing when the user presses a key.
	KeyPress (see page 6163)	Respond to keyboard input.
	KeyPress (see page 14833)	Provides special processing when the user presses a key.
	KeyUp (see page 6164)	Respond to released key.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Loaded (see page 5771)	Initializes the control after it is loaded from a stream.

	MainWndProc (see page 6165)	Receives Windows messages for the control.
	MouseActivate (see page 5773)	This is MouseActivate, a member of class TControl.
V	MouseDown (see page 23424)	Provides special processing when the user presses a mouse button.
	MouseMove (see page 5773)	Respond to mouse moving over control area..
V	MouseMove (see page 14834)	Provides special processing when the user moves the mouse.
	MouseUp (see page 5774)	OnMouseUp (see page 5867) event dispatcher.
V	MouseUp (see page 14835)	Provides special processing when the user releases the mouse button.
	MoveColRow (see page 14835)	Selects the cell indicated by the ACol and ARow parameters.
	MoveColumn (see page 14836)	Moves a column to a new position.
	MoveRow (see page 14836)	Moves a row to a new position.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 5775)	Responds to notifications that components are being created or destroyed.
V	Notification (see page 6165)	Responds to notifications that components are being created or destroyed.
	NotifyControls (see page 6165)	Sends a message to all the child controls.
V	Paint (see page 5920)	Renders the image of a custom control.
V	Paint (see page 14836)	Draws the image of the grid control on the screen.
	PaintControls (see page 6166)	Paints each of the child controls in a windowed control using the specified device context.
	PaintHandler (see page 6166)	Responds to WM_PAINT messages.
V	PaintWindow (see page 5920)	Renders the image of the control on a specified device context.
V	PaintWindow (see page 6169)	Renders the image of a windowed control.
	PaletteChanged (see page 5775)	Responds to changes in the system's palette by realizing the control's palette.
V	PaletteChanged (see page 6170)	Responds to changes in the system's palette by realizing the control's palette and the palette for each child control.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PositionDockRect (see page 5777)	Positions the docking rectangle according to the current drag-and-dock state.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 5778)	Prepares the control for having its properties assigned values read from a stream.
V	ReadState (see page 6171)	Prepares the control for having its properties assigned values from a stream.
	RecreateWnd (see page 6171)	Recreate underlying Windows screen object.
	ReloadDockedControl (see page 6172)	Reloads a docked control.
	RemoveWindowProps (see page 6174)	This is RemoveWindowProps, a member of class TWinControl.
V	RequestAlign (see page 5780)	Instructs the parent of a control to reposition the control, enforcing its Align (see page 5798) property.
	ResetIme (see page 6174)	Restores the input method editor (IME) that was active when the application started.
	ResetImeComposition (see page 6175)	Directs the composition window of the input method editor (IME) to perform a specific action.
V	Resize (see page 23425)	Respond to control resize.
	RowHeightsChanged (see page 14837)	Responds when the row heights change.
V	RowMoved (see page 23425)	Generates an OnRowMoved event.
	ScaleConstraints (see page 5781)	This is ScaleConstraints, a member of class TControl.
	ScaleControls (see page 6176)	Rescale child controls only.
	ScrollData (see page 14838)	Scrolls the visual image of the cells in the grid.
V	SelectCell (see page 23426)	Determines whether a particular cell in the grid can be selected.
	SelectFirst (see page 6177)	Locates first selectable child control.
	SelectNext (see page 6177)	Moves the input focus from the current child control to the next one in the tab order.
	SendCancelMode (see page 5781)	Cancels the modal state of the control.
	SendDockNotification (see page 5782)	Sends a Windows message to the control's host dock site.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetAutoSize (see page 5783)	This is SetAutoSize, a member of class TControl.
V	SetBiDiMode (see page 5783)	Sets the control's bi-directional mode.
V	SetCell (see page 23426)	This is SetCell, a member of class TValueListEditor.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

	SetChildOrder (see page 6178)	Changes the order in which the child component appears in the list of child components returned by the GetChildren (see page 6153) method.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDragMode (see page 5785)	Sets the drag mode for the control.
	SetEditText (see page 23426)	This is SetEditText, a member of class TValueListEditor.
	SetEnabled (see page 5785)	Sets the value of the Enabled (see page 5821) property.
	SetIme (see page 6181)	Activates an input method editor (IME) using the values specified by the ImeMode (see page 6207) and ImeName (see page 6208) properties.
	SetImeCompositionWindow (see page 6181)	Sets the position and font of the composition window.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetName (see page 5785)	Sets the name of the control.
	SetParent (see page 5786)	Sets the parent of the control.
	SetParent (see page 6182)	Sets the parent of the control.
	SetParentBackground (see page 6182)	This is SetParentBackground, a member of class TWinControl.
	SetParentBiDiMode (see page 5786)	Specifies whether the control uses its parent's BiDiMode (see page 5803).
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 5787)	Sets the parent for the control.
	SetStrings (see page 23426)	This is SetStrings, a member of class TValueListEditor.
	SetZOrder (see page 5788)	Repositions the control in its parent's control list, thereby changing position onscreen.
	SetZOrder (see page 6182)	Moves the control to the top or bottom of the parent control's (or screen's) list of TWinControl (see page 6112) controls.
	ShowControl (see page 6183)	Ensures that a specified child control is visible.
	ShowEditor (see page 14839)	Displays the inplace editor when the grid enters edit mode.
	ShowEditorChar (see page 14839)	Displays the inplace editor, with its text set to the specified character.
	SizeChanged (see page 14840)	Responds to changes in the size of the grid.
	Sizing (see page 14840)	Indicates whether the indicated coordinates are on one of the resize hot zones of the graph.
	StringsChanging (see page 23427)	This is StringsChanging, a member of class TValueListEditor.
	TimedScroll (see page 14840)	Extends a move, resize, or selection out of the scrollable region when the user drags the mouse in a direction where the grid is scrolled.
	TitleCaptionsStored (see page 23427)	This is TitleCaptionsStored, a member of class TValueListEditor.
	TopLeftChanged (see page 14784)	Generates an OnTopLeftChanged (see page 14800) event.
	TopLeftChanged (see page 14841)	Updates the position of the inplace editor when the grid scrolls.
	UpdateBounds (see page 6185)	This is UpdateBounds, a member of class TWinControl.
	UpdateBoundsRect (see page 5789)	Changes the Top (see page 5844), Left (see page 5830), Width (see page 5849), and Height (see page 5825) properties of the control, but doesn't update its screen image and bypasses all resize events.
	UpdateControlOriginalParentSize (see page 6186)	This is UpdateControlOriginalParentSize, a member of class TWinControl.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateDesigner (see page 14841)	Notifies the designer when rows and columns of the grid are moved or resized.
	UpdateExplicitBounds (see page 5789)	This is UpdateExplicitBounds, a member of class TControl.
	UpdateRecreatingFlag (see page 6186)	This is UpdateRecreatingFlag, a member of class TWinControl.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateUIState (see page 6186)	Updates form User Interface State as needed
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	VisibleChanging (see page 5791)	Provides the interface for a method that responds to changes in the control's visibility.

V	WndProc (see page 5791)	Provides specific message responses for the control.
V	WndProc (see page 6187)	Provides specific message responses for the control.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	ActionLink (see page 5798)	Contains the action link object associated with the control.
	AutoSize (see page 5803)	Specifies whether the control sizes itself automatically to accommodate its contents.
	BevelEdges (see page 6192)	Specifies which edges of the control are beveled.
	BevelInner (see page 6193)	Specifies the cut of the inner bevel.
	BevelKind (see page 6193)	Specifies the control's bevel style.
	BevelOuter (see page 6194)	Specifies the cut of the outer bevel.
	BevelWidth (see page 6194)	Specifies the width of the inner and outer bevels.
	BorderWidth (see page 6195)	Specifies the width of the control's border.
R	Canvas (see page 5921)	Specifies the TCanvas object that presents a drawing surface for the control.
	Caption (see page 5806)	Specifies a text string that identifies the control to the user.
	Col (see page 14843)	Specifies the index of the column that contains the selected cell.
	ColWidths (see page 14845)	Indicates the width (in pixels) of all the columns in the grid.
	DefWndProc (see page 6199)	Specifies the default window procedure for the windowed control.
	DesktopFont (see page 5818)	Specifies whether the control uses the Windows icon font when writing text.
R	EditList (see page 23441)	This is EditList, a member of class TValueListEditor.
	EditorMode (see page 14848)	Determines whether the current cell can be edited.
	FixedRows (see page 14849)	Specifies the number of rows on the top of the grid that cannot be scrolled.
R	GridHeight (see page 14850)	Specifies the height of the grid in pixels.
R	GridWidth (see page 14851)	Specifies the width of the grid in pixels.
R	HitTest (see page 14851)	Indicates the current position of the mouse cursor.
	ImeMode (see page 6207)	Determines the behavior of the input method editor (IME).
	ImeName (see page 6208)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
R	InplaceEditor (see page 14852)	Designates the edit control the grid uses to allow users to edit the contents of the selected cell.
	IsControl (see page 5829)	Determines whether a form stores its form-specific properties to a stream.
	LeftCol (see page 14852)	Specifies the index of the first visible scrollable column in the grid.
	MouseCapture (see page 5831)	Indicates whether the control has "captured" mouse events.
	ParentBackground (see page 6209)	Determines whether control uses parent's theme background.
	Row (see page 14854)	Specifies the index of the row that contains the selected cell.
	RowHeights (see page 14855)	Gives the height (in pixels) of all rows in the grid.
	ScalingFlags (see page 5840)	Reflects which attributes of the control have been scaled.
	Selection (see page 14856)	Indicates the boundaries of the current selection.
	TabStops (see page 14858)	Indicates whether the user can tab to specified columns in the grid.
	Text (see page 5842)	Contains a text string associated with the control.
	TopRow (see page 14858)	Specifies the index of the first visible scrollable row in the grid.
	WheelAccumulator (see page 5849)	Accumulates mouse wheel rotations.
	WindowHandle (see page 6212)	Provides access to a window handle for the control.
	WindowText (see page 5850)	Contains the text string of the control.

Public Events

	Name	Description
	WindowProc (see page 5872)	Points to the window procedure that responds to messages sent to the control.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.

	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginDrag (see page 5740)	Starts the dragging of a control.
	BringToFront (see page 5740)	Puts the control in front of all other controls in its parent control.
	Broadcast (see page 6130)	Sends a message to each of the child controls.
	CanFocus (see page 6131)	Indicates whether a control can receive focus.
	CellRect (see page 14781)	Returns the screen coordinates of a cell in the grid.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClientToParent (see page 5744)	Translates client coordinates to parent coordinates.
	ClientToScreen (see page 5744)	Translates a given point from client area coordinates to global screen coordinates.
	ContainsControl (see page 6133)	Indicates whether a specified control exists within the control.
	ControlAtPos (see page 6133)	Returns the child control located at a specified position within the control.
	Create (see page 23415)	Creates and initializes new TValueListEditor (see page 23401) object.
	CreateParented (see page 6123)	Creates and initializes a control as the child of a specified non-VCL container.
	CreateParentedControl (see page 6136)	Creates and initializes a control as the child of a specified non-VCL window.
	DefaultHandler (see page 5748)	Provides message handling for all messages that do not have specific handlers.
	DefaultHandler (see page 6139)	Provides message handling for all messages that the control does not fully process itself.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DeleteRow (see page 23418)	Deletes a row in the value list editor and a corresponding string from the associated TStrings object.
	Destroy (see page 23415)	Frees the memory associated with the TValueListEditor (see page 23401) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	DisableAlign (see page 6141)	Disables the realignment of child controls.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Dock (see page 5749)	Used internally to dock the control.
	DockDrop (see page 6142)	Generates an OnDockDrop (see page 6214) event.
	DragDrop (see page 5754)	OnDragDrop (see page 5855) event dispatcher.
	Dragging (see page 5754)	Indicates whether a control is being dragged.
	DrawTextBiDiModeFlags (see page 5756)	Returns the text flags that reflect the current setting of the BiDiMode (see page 5803) property.
	DrawTextBiDiModeFlagsReadingOnly (see page 5756)	Returns the text flag to add that indicates whether the control's text should read from right to left.
	EnableAlign (see page 6148)	Decrements the reference count incremented by the DisableAlign (see page 6141) method, eventually realigning the child controls.
	EndDrag (see page 5757)	Stops a control from being dragged any further.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindChildControl (see page 6148)	Returns a child control given its name.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindRow (see page 23420)	Locates the row that corresponds to a specified key name.
	FlipChildren (see page 6150)	Reverses the positions of child controls.
	Focused (see page 6150)	Determines whether the control has input focus.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetControlsAlignment (see page 5759)	Indicates how text is aligned within the control.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 5765)	Returns the parent of the control.
	GetTabOrderList (see page 6158)	Builds a list of controls in tab order.
	GetTextBuf (see page 5766)	Retrieves the control's text, copies it into a buffer, and returns the number of characters copied.
	GetTextLen (see page 5767)	Returns the length of the control's text.
	HandleAllocated (see page 6159)	Reports whether a screen object handle exists for the control.
	HandleNeeded (see page 6160)	Creates a screen object for the control if it doesn't already exist.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 5768)	Indicates whether the control has a parent.
	Hide (see page 5769)	Makes the control invisible.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitiateAction (see page 5769)	Calls the action link's Update (see page 5789) method if the control is associated with an action link.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InsertControl (see page 6160)	Inserts a control into the Controls (see page 6197) array property.
	InsertRow (see page 23422)	Inserts a row in the value list editor and a corresponding string into the associated TStrings object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Invalidate (see page 5770)	Completely repaint control.
	Invalidate (see page 6161)	Schedules a control repaint.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsRightToLeft (see page 5770)	Indicates whether the control should be reversed right to left.
	ManualDock (see page 5771)	Docks the control.
	ManualFloat (see page 5772)	Undocks the control.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	MouseCoord (see page 14834)	Returns the row and column indexes of the cell that contains a point specified in screen coordinates.
	MouseToCell (see page 14783)	Returns the column and row of the cell at the position with screen coordinates (X,Y).
	MouseWheelHandler (see page 5774)	Dispatches messages received from a mouse wheel.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintTo (see page 6167)	This is the overview for the PaintTo method overload.
	ParentToClient (see page 5776)	Translate parent coordinates to client coordinates.
	Perform (see page 5776)	Responds as if the control received a specified Windows message.
	PreProcessMessage (see page 6170)	This is PreProcessMessage, a member of class TWinControl.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Realign (see page 6171)	Forces the control to realign children.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.

	Refresh (see page 23424)	Updates the display to reflect the current value of the Strings (see page 23456) property.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveControl (see page 6172)	Removes a specified control from the Controls (see page 6197) array.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Repaint (see page 5779)	Forces the control to repaint its image on the screen.
V	Repaint (see page 6174)	Repaints the entire control.
	ReplaceDockedControl (see page 5779)	Docks the control in the place where another control is already docked.
	RestoreCurrentRow (see page 23425)	Undoes all edits to the current row.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ScaleBy (see page 6175)	Rescale control and its children.
	ScreenToClient (see page 5781)	Converts the screen coordinates of a specified point on the screen to client coordinates.
	ScrollBy (see page 6176)	Scroll control contents.
	SendToBack (see page 5782)	Puts a windowed control behind all other windowed controls, or puts a non-windowed control behind all other non-windowed controls.
V	SetBounds (see page 5783)	Sets the Left (see page 5830), Top (see page 5844), Width (see page 5849), and Height (see page 5825) properties all at once.
V	SetBounds (see page 6178)	Sets the windowed control's boundary properties all at once.
	SetDesignVisible (see page 5784)	Makes a control visible at design time.
V	SetDesignVisible (see page 6178)	Makes a control visible at design time.
V	SetFocus (see page 6179)	Gives the input focus to the control.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetTextBuf (see page 5787)	Sets the text of the control.
	Show (see page 5788)	Makes a control visible.
V	Update (see page 5789)	Processes any pending paint messages immediately.
V	Update (see page 6183)	Forces the control to update.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateControlState (see page 6186)	Respond to state change.
	UseRightToLeftAlignment (see page 5790)	Specifies whether the control's alignment should be reversed right-to-left.
	UseRightToLeftReading (see page 5790)	Specifies whether the control is using a right-to-left reading order.
	UseRightToLeftScrollBar (see page 5791)	Specifies whether the vertical scroll bar appears on the left side of the control.

Public Properties

	Name	Description
	Action (see page 5797)	Designates the action associated with the control.
R	AlignDisabled (see page 6192)	Indicates child control realignment disabled.
	BoundsRect (see page 5803)	Specifies the bounding rectangle of the control, expressed in the coordinate system of the parent control.
R	Brush (see page 6195)	Determines the color and pattern used for painting the background of the control.
	Canvas (see page 14785)	Specifies the TCanvas object that presents a drawing surface for the control.
	Cells (see page 23433)	Lists the strings for each cell in the editor.
	ClientHeight (see page 5808)	Specifies the height of the control's client area in pixels.
R	ClientOrigin (see page 5809)	Specifies the screen coordinates (in pixels) of the top left corner of a control's client area.
R	ClientRect (see page 5810)	Specifies the size (in pixels) of a control's client area.
	ClientWidth (see page 5811)	Specifies the horizontal size of the control's client area in pixels.
	Col (see page 14787)	Specifies the index of the column that contains the selected cell.
R	ColCount (see page 23434)	Indicates the number of columns in the value list editor.
	ColWidths (see page 14788)	Indicates the width (in pixels) of all the columns in the grid.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.

R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	ControlCount (see page 6195)	Returns the number of child controls.
R	Controls (see page 6197)	Lists all child controls.
	ControlState (see page 5813)	Indicates the current state of a control at runtime.
	ControlStyle (see page 5814)	Determines style characteristics of the control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DockClientCount (see page 6199)	Specifies the number of controls that are docked on the windowed control.
R	DockClients (see page 6201)	Lists the controls that are docked to the windowed control.
	DockManager (see page 6202)	Specifies the control's docking manager interface.
	DockOrientation (see page 5818)	Specifies how the control is docked relative to other controls docked in the same parent.
	DockSite (see page 6204)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 6205)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	EditorMode (see page 14789)	Determines whether the current cell can be edited.
R	ExplicitHeight (see page 5821)	Specifies the explicit vertical size of the control in pixels.
R	ExplicitLeft (see page 5822)	Specifies the explicit horizontal pixel coordinate of the left edge of a component relative to its parent.
R	ExplicitTop (see page 5822)	Specifies the explicit vertical pixel coordinate of the top edge of a component relative to its parent.
R	ExplicitWidth (see page 5822)	Specifies the explicit horizontal size of the control in pixels.
R	Floating (see page 5823)	Indicates whether the control is floating.
	FloatingDockSiteClass (see page 5823)	Specifies the class of the temporary control that hosts the control when it is floating.
	GridHeight (see page 14789)	Specifies the height of the grid in pixels.
	GridWidth (see page 14789)	Specifies the width of the grid in pixels.
R	Handle (see page 6207)	Provides access to the underlying Windows screen object for the control.
	HostDockSite (see page 5829)	Specifies the control in which the control is docked.
	ItemProps (see page 23445)	Specifies various constraints on the way users can enter a value for each item in the list.
	Keys (see page 23445)	Lists the strings for each cell in the name (key) column.
	LeftCol (see page 14790)	Specifies the index of the first visible scrollable column in the grid.
	LRDockWidth (see page 5831)	Specifies the width of the control when it is docked horizontally.
R	MouseInClient (see page 6209)	This is MouseInClient, a member of class TWinControl.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Padding (see page 6209)	Specifies the padding of a control.
	Parent (see page 5832)	Indicates the parent of the control.
	ParentWindow (see page 6210)	Reference to parent's underlying control.
	Row (see page 14790)	Specifies the index of the row that contains the selected cell.
R	RowCount (see page 23453)	Indicates the number of rows in the value list editor.
	RowHeights (see page 14791)	Gives the height (in pixels) of all rows in the grid.
	Selection (see page 14791)	Indicates the boundaries of the current selection.
R	Showing (see page 6210)	Indicates whether the control is showing on the screen.
	TabStop (see page 6211)	Determines if the user can tab to a control.
	TabStops (see page 14793)	Indicates whether the user can tab to specified columns in the grid.
	TBDockHeight (see page 5842)	Specifies the height of the control when it is docked vertically.
	TopRow (see page 14793)	Specifies the index of the first visible scrollable row in the grid.
	UndockHeight (see page 5845)	Specifies the height of the control when it is floating.
	UndockWidth (see page 5846)	Specifies the width of the control when it is floating.
	UseDockManager (see page 6212)	Specifies whether the docking manager is used in drag-and-dock operations.
	Values (see page 23457)	Lists the strings for each value from the value column.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	VisibleColCount (see page 23459)	Indicates the number of scrollable columns visible in the grid.

R	VisibleDockClientCount (see page 6212)	Specifies the number of visible controls that are docked on the windowed control.
	VisibleRowCount (see page 23459)	Indicates the number of scrollable rows visible in the grid.

Published Events

	Name	Description
	OnClick (see page 23461)	Occurs when the user clicks the control.
	OnContextPopup (see page 23463)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 23463)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 23464)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 23466)	Occurs when the user drags an object over a control.
	OnDrawCell (see page 23468)	Occurs when a cell in the grid needs to be drawn.
	OnEditButtonClick (see page 23470)	Occurs when the user click the ellipsis button on a cell in the values column or types Ctrl+Enter while focus is on a value cell with an ellipsis button.
	OnEndDock (see page 23470)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 23471)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 23472)	Occurs when a control receives the input focus.
	OnExit (see page 23474)	Occurs when the input focus shifts away from one control to another.
	OnGetEditMask (see page 23476)	Occurs when the in-place editor requests an edit mask.
	OnGetEditText (see page 23476)	Occurs when the in-place editor requests the value of a cell.
	OnGetPickList (see page 23477)	Supplies a pick list for values associated with a particular key.
	OnKeyDown (see page 23478)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 23480)	Occurs when a key is pressed.
	OnKeyUp (see page 23481)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 23483)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 23484)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 23486)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 23487)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 23487)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 23489)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheelDown (see page 23492)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 23492)	Occurs when the mouse wheel is rotated upward.
	OnRowMoved (see page 23492)	Occurs immediately after the position of a row changes.
	OnSelectCell (see page 23493)	Occurs before a cell in the grid is selected.
	OnSetEditText (see page 23494)	Occurs when the user edits the value of a cell.
	OnStartDock (see page 23495)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 23495)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnStringsChange (see page 23496)	Occurs immediately after the names or values in the string list change.
	OnStringsChanging (see page 23496)	Occurs immediately before the names or values in the string list change.
	OnTopLeftChanged (see page 23497)	Occurs immediately after the TopRow property or the LeftCol property changes.
	OnValidate (see page 23497)	Occurs when focus shifts away from a cell in the value list editor.

Published Properties

	Name	Description
	Align (see page 23428)	Determines how the control aligns within its container (parent control).
	AlignWithMargins (see page 5802)	Indicates whether a control should be constrained by margins.
	Anchors (see page 23432)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 23432)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 23433)	Determines whether a single line border is drawn around the grid.

	Color (see page 23434)	Specifies the background color of the control.
	Constraints (see page 23435)	Specifies the size constraints for the control.
	Ctl3D (see page 23435)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	Cursor (see page 5815)	Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.
	DefaultColWidth (see page 23436)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 23437)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 23437)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DisplayOptions (see page 23438)	Controls (see page 5711) aspects of the value list editor's appearance.
	DragCursor (see page 23438)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 23438)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 23439)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownRows (see page 23440)	Limits the length of the drop-down list that appears for values that use a pick list.
	Enabled (see page 23441)	Controls whether the control responds to mouse, keyboard, and timer events.
	FixedColor (see page 23442)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 23442)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	Font (see page 23442)	Controls the attributes of text written on or in the control.
	GridLineWidth (see page 23444)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
	Height (see page 5825)	Specifies the vertical size of the control in pixels.
	HelpContext (see page 5826)	Numeric ID for control's context-sensitive help topic
	HelpKeyword (see page 5826)	Keyword for control's context-sensitive help topic
	HelpType (see page 5827)	Indicates whether the control's context sensitive help topic is identified by context ID or by keyword.
	Hint (see page 5827)	Contains the text string that can appear when the user moves the mouse over the control.
	KeyOptions (see page 23445)	Controls (see page 5711) the way the value list editor responds to user keystrokes.
	Left (see page 5830)	Specifies the horizontal coordinate of the left edge of a component relative to its parent.
	Margins (see page 5831)	Specifies the margins for the control.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Options (see page 23446)	Specifies various display and behavioral properties of the grid.
	ParentBiDiMode (see page 23446)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 23447)	Determines where a control looks for its color information.
	ParentCtl3D (see page 23447)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 23448)	Determines where a control looks for its font information.
	ParentShowHint (see page 23448)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 23449)	Identifies the pop-up menu associated with the control.
	ScrollBars (see page 23454)	Specifies whether the grid includes horizontal and vertical scroll bars.
	ShowHint (see page 23454)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Strings (see page 23456)	Provides access to the string list that is being edited.
	TabOrder (see page 23456)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 14857)	Determines if the user can tab to a control.
	Tag (see page 2808)	Stores an integer value as part of a component.
	TitleCaptions (see page 23457)	Specifies the captions that label the columns of the value list editor.
	Top (see page 5844)	Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.
	Visible (see page 23458)	Determines whether the component appears onscreen.
	Width (see page 5849)	Specifies the horizontal size of the control or form in pixels.

168.1.2.2 TValueListEditor.Create Constructor

Creates and initializes new TValueListEditor (see page 23401) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TValueListEditor(TComponent * AOwner);
```

Description

Call Create to create a value list editor component at runtime. TValueListEditor (see page 23401) components added to forms at design time are created automatically.

The AOwner parameter specifies another component (typically the form) that is responsible for freeing the TValueListEditor (see page 23401) instance.

See Also

Owner (see page 2807)

168.1.2.3 TValueListEditor.Destroy Destructor

Frees the memory associated with the TValueListEditor (see page 23401) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TValueListEditor();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the value list editor reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

168.1.2.4 TValueListEditor Methods

The methods of the TValueListEditor class are listed here.

Protected Methods

	Name	Description
	AdjustColWidths (see page 23416)	This is AdjustColWidths, a member of class TValueListEditor.
	AdjustRowCount (see page 23416)	This is AdjustRowCount, a member of class TValueListEditor.
	CanEditModify (see page 23417)	Indicates whether the current cell in the grid can be edited.
	ColWidthsChanged (see page 23417)	Responds when the column widths change.
	CreateEditor (see page 23417)	Creates the inplace editor control.
	CreateWnd (see page 23418)	Creates the underlying window.
	DoExit (see page 23419)	Hides the inplace editor when selection moves to another control in a form.
	DoOnValidate (see page 23419)	This is DoOnValidate, a member of class TValueListEditor.

	DrawCell (see page 23419)	Generates an OnDrawCell event.
	EditListGetItems (see page 23420)	This is EditListGetItems, a member of class TValueListEditor.
	GetCell (see page 23420)	This is GetCell, a member of class TValueListEditor.
	GetColCount (see page 23421)	This is GetColCount, a member of class TValueListEditor.
	GetEditLimit (see page 23421)	Provides the interface for a method that limits the number of characters the user can type into cells.
	GetEditMask (see page 23421)	Generates an OnGetEditMask event.
	GetEditStyle (see page 23421)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
	GetEditText (see page 23422)	This is GetEditText, a member of class TValueListEditor.
	GetPickList (see page 23422)	This is GetPickList, a member of class TValueListEditor.
	GetRowCount (see page 23422)	This is GetRowCount, a member of class TValueListEditor.
	IsEmptyRow (see page 23423)	This is IsEmptyRow, a member of class TValueListEditor.
	KeyDown (see page 23423)	Provides special processing when the user presses a key.
	MouseDown (see page 23424)	Provides special processing when the user presses a mouse button.
	Resize (see page 23425)	Respond to control resize.
	RowMoved (see page 23425)	Generates an OnRowMoved event.
	SelectCell (see page 23426)	Determines whether a particular cell in the grid can be selected.
	SetCell (see page 23426)	This is SetCell, a member of class TValueListEditor.
	SetEditText (see page 23426)	This is SetEditText, a member of class TValueListEditor.
	SetStrings (see page 23426)	This is SetStrings, a member of class TValueListEditor.
	StringsChanging (see page 23427)	This is StringsChanging, a member of class TValueListEditor.
	TitleCaptionsStored (see page 23427)	This is TitleCaptionsStored, a member of class TValueListEditor.

Public Methods

	Name	Description
	DeleteRow (see page 23418)	Deletes a row in the value list editor and a corresponding string from the associated TStrings object.
	FindRow (see page 23420)	Locates the row that corresponds to a specified key name.
	InsertRow (see page 23422)	Inserts a row in the value list editor and a corresponding string into the associated TStrings object.
	Refresh (see page 23424)	Updates the display to reflect the current value of the Strings (see page 23456) property.
	RestoreCurrentRow (see page 23425)	Undoes all edits to the current row.

168.1.2.4.1 TValueListEditor.AdjustColWidths Method

Pascal

```
procedure AdjustColWidths; virtual;
```

C++

```
virtual __fastcall AdjustColWidths();
```

Description

This is AdjustColWidths, a member of class TValueListEditor.

168.1.2.4.2 TValueListEditor.AdjustRowCount Method

Pascal

```
procedure AdjustRowCount; virtual;
```

C++

```
virtual __fastcall AdjustRowCount();
```

Description

This is AdjustRowCount, a member of class TValueListEditor.

168.1.2.4.3 TValueListEditor.CanEditModify Method

Indicates whether the current cell in the grid can be edited.

Pascal

```
function CanEditModify: Boolean; override;
```

C++

```
virtual __fastcall Boolean CanEditModify();
```

Description

Grid components use CanEditModify internally to determine whether to allow the user to edit the current cell. CanEditModify returns true unless the inplace editor has disallowed editing for the cell.

See Also

CanEditShow (see page 14820)

InplaceEditor (see page 14852)

168.1.2.4.4 TValueListEditor.ColWidthsChanged Method

Responds when the column widths change.

Pascal

```
procedure ColWidthsChanged; override;
```

C++

```
virtual __fastcall ColWidthsChanged();
```

Description

ColWidthsChanged is called immediately after the column widths change. The change can result from setting the ColWidths property, from setting the DefaultColWidth property, from moving one of the columns, or from resizing the columns with the mouse.

ColWidthsChanged adjusts the scroll bar and the position of the inplace editor to take into account the new column size and positions of the columns.

See Also

ColWidths (see page 14845)

DefaultColWidth (see page 14846)

InplaceEditor (see page 14852)

ScrollBars (see page 14856)

168.1.2.4.5 TValueListEditor.CreateEditor Method

Creates the inplace editor control.

Pascal

```
function CreateEditor: TInplaceEdit; override;
```

C++

```
virtual __fastcall TInplaceEdit CreateEditor();
```

Description

Grid controls call CreateEditor internally the first time the grid goes into edit mode. CreateEditor is exposed as a protected method so that descendants of TCustomGrid can substitute a specialized descendant of TInplaceEdit to handle cell editing.

See Also

EditorMode (see page 14848)

InplaceEditor (see page 14852)

ShowEditor (see page 14839)

168.1.2.4.6 TValueListEditor.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams (see page 6135)

CreateWindowHandle (see page 6137)

Perform (see page 5776)

DestroyWnd (see page 6141)

168.1.2.4.7 TValueListEditor.DeleteRow Method

Deletes a row in the value list editor and a corresponding string from the associated TStrings object.

Pascal

```
procedure DeleteRow(ARow: Integer); override;
```

C++

```
virtual __fastcall DeleteRow(int ARow);
```

Description

Call DeleteRow to remove the row specified by the ARow parameter. Delete removes the corresponding string from the TStrings object being edited and updates the display.

ARow must be the index of a row that displays a name/value pair. That is, it can't be less than 0 or greater than the number of rows. If DisplayOptions (see page 23438) includes doColumnTitles, ARow can't be 0, because that is the title row, not a name/value pair in the associated TStrings object.

Note: DeleteRow deletes the specified row regardless of the value of KeyOptions (see page 23445). The keyDelete member of KeyOptions (see page 23445) controls only whether the user can delete a row by pressing the Delete key.

See Also

Cells (see page 23433)

Strings (see page 23456)

KeyOptions (see page 23445)

DisplayOptions (see page 23438)

168.1.2.4.8 TValueListEditor.DoExit Method

Hides the inplace editor when selection moves to another control in a form.

Pascal

```
procedure DoExit; override;
```

C++

```
virtual __fastcall DoExit();
```

Description

After calling the inherited method, which generates an OnExit event, DoExit hides the inplace editor unless the Options property includes goAlwaysShowEditor.

See Also

HideEditor (see page 14830)

Options (see page 14853)

OnExit (see page 6221)

168.1.2.4.9 TValueListEditor.DoOnValidate Method

Pascal

```
procedure DoOnValidate; virtual;
```

C++

```
virtual __fastcall DoOnValidate();
```

Description

This is DoOnValidate, a member of class TValueListEditor.

168.1.2.4.10 TValueListEditor.DrawCell Method

Generates an OnDrawCell event.

Pascal

```
procedure DrawCell(ACol: Longint; ARow: Longint; ARect: TRect; AState: TGridDrawState);
override;
```

C++

```
virtual __fastcall DrawCell(Longint ACol, Longint ARow, TRect ARect, TGridDrawState AState);
```

Description

DrawCell is called automatically whenever a cell in the grid is drawn. If the DefaultDrawing property is true, the background of the cell is painted before DrawCell is called, and the 3D effects of the fixed cells or the focus rectangle around the focused cell

are drawn after DrawCell. Descendants of TCustomDrawGrid can override this method to draw the contents of the cell or to block the OnDrawCell event.

See Also

DefaultDrawing (see page 14847)

FixedCols (see page 14848)

FixedRows (see page 14849)

OnDrawCell (see page 14795)

168.1.2.4.11 TValueListEditor.EditListGetItems Method

Pascal

```
procedure EditListGetItems(ACol: Integer; ARow: Integer; Items: TStrings);
```

C++

```
__fastcall EditListGetItems(int ACol, int ARow, TStrings Items);
```

Description

This is EditListGetItems, a member of class TValueListEditor.

168.1.2.4.12 TValueListEditor.FindRow Method

Locates the row that corresponds to a specified key name.

Pascal

```
function FindRow(const KeyName: string; var Row: Integer): Boolean;
```

C++

```
__fastcall Boolean FindRow(const AnsiString KeyName, int Row);
```

Description

Call FindRow to locate the row in the value list editor that has the value specified by KeyName in the first column. This corresponds to a name/value pair in the string list where the name part is KeyName.

If a row in the value list editor has the specified key name, FindRow returns true and the Row parameter returns the index of the first row found with the specified name. If the specified name does not correspond to any of the rows in the value list editor, FindRow returns false.

See Also

Keys (see page 23445)

168.1.2.4.13 TValueListEditor.GetCell Method

Pascal

```
function GetCell(ACol: Integer; ARow: Integer): string; virtual;
```

C++

```
virtual __fastcall AnsiString GetCell(int ACol, int ARow);
```

Description

This is GetCell, a member of class TValueListEditor.

168.1.2.4.14 TValueListEditor.GetColCount Method

Pascal

```
function GetColCount: Integer;
```

C++

```
__fastcall int GetColCount();
```

Description

This is GetColCount, a member of class TValueListEditor.

168.1.2.4.15 TValueListEditor.GetEditLimit Method

Provides the interface for a method that limits the number of characters the user can type into cells.

Pascal

```
function GetEditLimit: Integer; override;
```

C++

```
virtual __fastcall int GetEditLimit();
```

Description

The inplace editor calls GetEditLimit to determine if a limit should be placed on the number of characters a user can type into a cell. GetEditLimit sets no limits. Descendants of TCustomGrid override this method to limit the text for certain cells.

See Also

InplaceEditor (see page 14852)

168.1.2.4.16 TValueListEditor.GetEditMask Method

Generates an OnGetEditMask event.

Pascal

```
function GetEditMask(ACol: Longint; ARow: Longint): string; override;
```

C++

```
virtual __fastcall AnsiString GetEditMask(Longint ACol, Longint ARow);
```

Description

The inplace editor calls GetEditMask to obtain any input mask associated with the cell specified by the ACol and ARow parameters. GetEditMask returns the Value parameter set by the OnGetEditMask event handler. If there is no OnGetEditMask event handler, GetEditMask returns an empty string, indicating that there is no edit mask.

See Also

OnGetEditMask (see page 14796)

168.1.2.4.17 TValueListEditor.GetEditStyle Method

Indicates whether a specified cell has an ellipsis button or drop-down arrow.

Pascal

```
function GetEditStyle(ACol: Longint; ARow: Longint): TEditStyle; override;
```

C++

```
virtual __fastcall TEditStyle GetEditStyle(Longint ACol, Longint ARow);
```

Description

GetEditStyle indicates whether the specified cell can only be edited directly (esSimple), whether it has an ellipsis button that users can click to launch an editor for modifying field values (esEllipsis), or whether it has a drop-down arrow that users can click to display a pick list of possible values (esPickList).

ACol is the column that contains the cell. ARow is the row that contains the cell.

In TCustomGrid, GetEditStyle always returns esSimple, indicating that there is no ellipsis button or drop-down arrow. Descendant classes can override this methods if they provide a way to add pick lists or ellipsis buttons to cells.

168.1.2.4.18 TValueListEditor.GetEditText Method

Pascal

```
function GetEditText(ACol: Integer; ARow: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetEditText(int ACol, int ARow);
```

Description

This is GetEditText, a member of class TValueListEditor.

168.1.2.4.19 TValueListEditor.GetPickList Method

Pascal

```
function GetPickList(Values: TStrings; ClearFirst: Boolean = True): Boolean;
```

C++

```
__fastcall Boolean GetPickList(TStrings Values, Boolean ClearFirst = True);
```

Description

This is GetPickList, a member of class TValueListEditor.

168.1.2.4.20 TValueListEditor.GetRowCount Method

Pascal

```
function GetRowCount: Integer;
```

C++

```
__fastcall int GetRowCount();
```

Description

This is GetRowCount, a member of class TValueListEditor.

168.1.2.4.21 TValueListEditor.InsertRow Method

Inserts a row in the value list editor and a corresponding string into the associated TStrings object.

Pascal

```
function InsertRow(const KeyName: string; const Value: string; Append: Boolean): Integer;
```

C++

```
__fastcall int InsertRow(const AnsiString KeyName, const AnsiString Value, Boolean Append);
```

Description

Call InsertRow to add a name/value pair to the value list editor and the TStrings object it is editing.

InsertRow adds a string of the form

`KeyName=Value`

to the Strings (see page 23456) property and updates the display to reflect the change.

If the current row of the value list editor is empty (has no name or value portion), InsertRow assigns the new KeyName and Value to the current row. Otherwise, if the Append parameter is true, the new string appears after the currently selected row, while if Append is false, the new string appears before the currently selected row.

Note: InsertRow inserts the specified row regardless of the value of KeyOptions (see page 23445). The keyAdd member of KeyOptions (see page 23445) controls only whether the user can add new rows by pressing the Insert key or navigating past the last row.

See Also

Cells (see page 23433)

Strings (see page 23456)

KeyOptions (see page 23445)

DisplayOptions (see page 23438)

168.1.2.4.22 TValueListEditor.IsEmptyRow Method

Pascal

```
function IsEmptyRow: Boolean;
```

C++

```
__fastcall Boolean IsEmptyRow();
```

Description

This is IsEmptyRow, a member of class TValueListEditor.

168.1.2.4.23 TValueListEditor.KeyDown Method

Provides special processing when the user presses a key.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

After generating an OnKeyDown event, KeyDown calls CanGridAcceptKey to screen out problematic key combinations. KeyDown moves to the appropriate cell in response to navigational key combinations, or sets the EditorMode property to true when the user presses F2.

See Also

CanGridAcceptKey (see page 14820)

EditorMode (see page 14848)

KeyPress (see page 14833)

OnKeyDown (see page 6225)

168.1.2.4.24 TValueListEditor.MouseDown Method

Provides special processing when the user presses a mouse button.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

MouseDown hides the editor window so that it can move selection to the cell that receives the mouse click. If the mouse lands in the hot region for beginning a move or resize of a row or column, MouseDown starts the move or resize, and provides visual feedback by changing the cursor. If the editor window was hidden, it is brought back up in its new location. Finally, MouseDown generates an OnMouseDown event.

See Also

HideEditor (see page 14830)

InplaceEditor (see page 14852)

MouseMove (see page 14834)

MouseUp (see page 14835)

ShowEditor (see page 14839)

OnMouseDown (see page 5862)

168.1.2.4.25 TValueListEditor.Refresh Method

Updates the display to reflect the current value of the Strings (see page 23456) property.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh to ensure that the value list editor accurately reflects the value of the Strings (see page 23456) property. For example, call Refresh after programmatically editing the value of the Strings (see page 23456) property.

See Also

RestoreCurrentRow (see page 23425)

Repaint (see page 6174)

Strings (see page 23456)

168.1.2.4.26 TValueListEditor.Resize Method

Respond to control resize.

Pascal

```
procedure Resize; override;
```

C++

```
virtual __fastcall Resize();
```

Description

Resize is called automatically immediately after the control's dimensions change. As implemented in TControl, Resize calls the control's OnResize event handler if it is assigned.

See Also

OnResize (see page 5870)

168.1.2.4.27 TValueListEditor.RestoreCurrentRow Method

Undoes all edits to the current row.

Pascal

```
function RestoreCurrentRow: Boolean;
```

C++

```
__fastcall Boolean RestoreCurrentRow();
```

Description

Call RestoreCurrentRow to back out edits to the current cell in the value list editor. RestoreCurrentRow backs out edits to the current cell if it is still in edit mode, or deletes a newly inserted row.

The user can accomplish the same effect by pressing the Escape key.

See Also

Cells (see page 23433)

InsertRow (see page 23422)

168.1.2.4.28 TValueListEditor.RowMoved Method

Generates an OnRowMoved event.

Pascal

```
procedure RowMoved(FromIndex: Longint; ToIndex: Longint); override;
```

C++

```
virtual __fastcall RowMoved(Longint FromIndex, Longint ToIndex);
```

Description

RowMoved is called immediately after a row in the grid changes position. It overrides the inherited method to generate an OnRowMoved event. Descendants of TCustomDrawGrid can override this method to make internal adjustments or to block the OnRowMoved event.

See Also

OnRowMoved (see page 14798)

168.1.2.4.29 TValueListEditor.SelectCell Method

Determines whether a particular cell in the grid can be selected.

Pascal

```
function SelectCell(ACol: Longint; ARow: Longint): Boolean; override;
```

C++

```
virtual __fastcall Boolean SelectCell(Longint ACol, Longint ARow);
```

Description

Grid controls call SelectCell internally before attempting to move selection to a the cell identified by the ACol and ARow parameters. ACol and ARow are the column and row indexes for the cell, where the first column has index 0, and the top row has index 0.

SelectCell generates an OnSelectCell event. This method then returns true, unless the OnSelectCell event handler indicates that the cell should not be selected. Descendants of TCustomDrawGrid can override this method to check the properties of the information displayed in the cell or block the OnSelectCell event.

See Also

OnSelectCell (see page 14798)

168.1.2.4.30 TValueListEditor.SetCell Method

Pascal

```
procedure SetCell(ACol: Integer; ARow: Integer; const Value: string); virtual;
```

C++

```
virtual __fastcall SetCell(int ACol, int ARow, const AnsiString Value);
```

Description

This is SetCell, a member of class TValueListEditor.

168.1.2.4.31 TValueListEditor.SetEditText Method

Pascal

```
procedure SetEditText(ACol: Integer; ARow: Integer; const Value: string); override;
```

C++

```
virtual __fastcall SetEditText(int ACol, int ARow, const AnsiString Value);
```

Description

This is SetEditText, a member of class TValueListEditor.

168.1.2.4.32 TValueListEditor.SetStrings Method

Pascal

```
procedure SetStrings(const Value: TStrings); virtual;
```

C++

```
virtual __fastcall SetStrings(const TStrings Value);
```

Description

This is SetStrings, a member of class TValueListEditor.

168.1.2.4.33 TValueListEditor.StringsChanging Method

Pascal

```
procedure StringsChanging; dynamic;
```

C++

```
__fastcall StringsChanging();
```

Description

This is StringsChanging, a member of class TValueListEditor.

168.1.2.4.34 TValueListEditor.TitleCaptionsStored Method

Pascal

```
function TitleCaptionsStored: Boolean;
```

C++

```
__fastcall Boolean TitleCaptionsStored();
```

Description

This is TitleCaptionsStored, a member of class TValueListEditor.

168.1.2.5 TValueListEditor Properties

The properties of the TValueListEditor class are listed here.

Protected Properties

	Name	Description
R	EditList (see page 23441)	This is EditList, a member of class TValueListEditor.

Public Properties

	Name	Description
	Cells (see page 23433)	Lists the strings for each cell in the editor.
R	ColCount (see page 23434)	Indicates the number of columns in the value list editor.
	ItemProps (see page 23445)	Specifies various constraints on the way users can enter a value for each item in the list.
	Keys (see page 23445)	Lists the strings for each cell in the name (key) column.
R	RowCount (see page 23453)	Indicates the number of rows in the value list editor.
	Values (see page 23457)	Lists the strings for each value from the value column.
	VisibleColCount (see page 23459)	Indicates the number of scrollable columns visible in the grid.
	VisibleRowCount (see page 23459)	Indicates the number of scrollable rows visible in the grid.

Published Properties

	Name	Description
	Align (see page 23428)	Determines how the control aligns within its container (parent control).
	Anchors (see page 23432)	Specifies how the control is anchored to its parent.

	BiDiMode (see page 23432)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 23433)	Determines whether a single line border is drawn around the grid.
	Color (see page 23434)	Specifies the background color of the control.
	Constraints (see page 23435)	Specifies the size constraints for the control.
	Ctl3D (see page 23435)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColWidth (see page 23436)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 23437)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 23437)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DisplayOptions (see page 23438)	Controls (see page 5711) aspects of the value list editor's appearance.
	DragCursor (see page 23438)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 23438)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 23439)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownRows (see page 23440)	Limits the length of the drop-down list that appears for values that use a pick list.
	Enabled (see page 23441)	Controls whether the control responds to mouse, keyboard, and timer events.
	FixedColor (see page 23442)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 23442)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	Font (see page 23442)	Controls the attributes of text written on or in the control.
	GridLineWidth (see page 23444)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
	KeyOptions (see page 23445)	Controls (see page 5711) the way the value list editor responds to user keystrokes.
	Options (see page 23446)	Specifies various display and behavioral properties of the grid.
	ParentBiDiMode (see page 23446)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 23447)	Determines where a control looks for its color information.
	ParentCtl3D (see page 23447)	Determines where a component looks to determine if it should appear three dimensional.
	ParentFont (see page 23448)	Determines where a control looks for its font information.
	ParentShowHint (see page 23448)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 23449)	Identifies the pop-up menu associated with the control.
	ScrollBars (see page 23454)	Specifies whether the grid includes horizontal and vertical scroll bars.
	ShowHint (see page 23454)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Strings (see page 23456)	Provides access to the string list that is being edited.
	TabOrder (see page 23456)	Indicates the position of the control in its parent's tab order.
	TitleCaptions (see page 23457)	Specifies the captions that label the columns of the value list editor.
	Visible (see page 23458)	Determines whether the component appears onscreen.

168.1.2.5.1 TValueListEditor.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align: TAlign;
```

C++

```
__property TAlign Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always

equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
#include <Extctrls.hpp>
#include <Filectrl.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1);
  TFileListBox* pflb = new TFileListBox(Form1);
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);
```

```
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this);
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this);
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}//---------------------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  for (int i=0;i<iTabTitles;i++)
        delete pts[i];
    delete ppc;
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
```

```
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
```

```
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent (see page 5832)

Anchors (see page 5802)

168.1.2.5.2 TValueListEditor.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors: TAnchors;
```

C++

```
__property TAnchors Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent (see page 5832)

Align (see page 5798)

168.1.2.5.3 TValueListEditor.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode: TBiDiMode;
```

C++

```
__property TBiDiMode BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of

the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment (see page 5790)

UseRightToLeftReading (see page 5790)

UseRightToLeftScrollBar (see page 5791)

Including Bi-directional Functionality in Applications

168.1.2.5.4 TValueListEditor.BorderStyle Property

Determines whether a single line border is drawn around the grid.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Set BorderStyle to bsSingle to add a single line border around the grid's image. Set BorderStyle to bsNone to omit the border.

168.1.2.5.5 TValueListEditor.Cells Property

Lists the strings for each cell in the editor.

Pascal

```
property Cells [ACol, ARow: Integer]: string;
```

C++

```
__property AnsiString Cells[int ACol, ARow];
```

Description

Use Cells to access the string in a particular cell. ACol is the column coordinate of the cell, and ARow is the row coordinate of the cell. ACol can be 0 (for the name column) or 1 (for the value column). ARow can range from 0 (for the title row) to the number of strings in the associated TStrings descendant.

To access the strings in the name column, use the Keys (see page 23445) property instead. To access the strings in the value column, use the Values (see page 23457) property instead. To access the underlying string list, use the Strings (see page 23456) property.

See Also

Keys (see page 23445)

Values (see page 23457)

Strings (see page 23456)

168.1.2.5.6 TValueListEditor.ColCount Property

Indicates the number of columns in the value list editor.

Pascal

```
property ColCount: Longint;
```

C++

```
__property Longint ColCount;
```

Description

TValueListEditor (see page 23401) redeclares the ColCount property, which it inherits from TCustomGrid, to change it to a read-only property. The value of ColCount is always 2 plus the number of fixed columns, because the value list editor is designed to display only the name and value of the strings in the associated string list.

See Also

Cells (see page 23433)

RowCount (see page 23453)

FixedCols (see page 14848)

168.1.2.5.7 TValueListEditor.Color Property

Specifies the background color of the control.

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
```

```
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor (see page 5834)

168.1.2.5.8 TValueListEditor.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints: TSizeConstraints;
```

C++

```
__property TSizeConstraints Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize (see page 5803)

Align (see page 5798)

Anchors (see page 5802)

OnConstrainedResize (see page 5853)

OnCanResize (see page 5851)

168.1.2.5.9 TValueListEditor.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D: Boolean;
```

C++

```
__property Boolean Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D (see page 6209)

BevelKind (see page 6193)

BevelInner (see page 6193)

BevelOuter (see page 6194)

BorderWidth (see page 6195)

168.1.2.5.10 TValueListEditor.DefaultColWidth Property

Determines the width (in pixels) of all columns that have not been explicitly resized.

Pascal

```
property DefaultColWidth: Integer;
```

C++

```
__property int DefaultColWidth;
```

Description

Set DefaultColWidth to change the size of all columns in the grid. When DefaultColWidth is set, columns that have been resized using the mouse or by setting the ColWidths property are given the DefaultColWidth as well.

When new columns are added to the grid using the ColCount property, they are created with a width of DefaultColWidth.

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if DrawGrid1.DefaultColWidth > 90 then
    DrawGrid1.GridLineWidth := 2
  else
    DrawGrid1.GridLineWidth := 1;
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clBackground;
```

```
    DrawGrid1.Canvas.FillRect(Rect);
    ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
    if (gdFocused in State) then
    begin
      DrawGrid1.Canvas.DrawFocusRect(Rect);
    end;
  end;

  procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  begin
    CanSelect:= True;
  end;
```

See Also

ColCount (see page 14844)

ColWidths (see page 14845)

DefaultRowHeight (see page 14847)

168.1.2.5.11 TValueListEditor.DefaultDrawing Property

Determines whether the Cells are painted when the grid is drawn.

Pascal

```
property DefaultDrawing: Boolean;
```

C++

```
__property Boolean DefaultDrawing;
```

Description

Set DefaultDrawing to true to have the grid automatically paint the cells in the grid, including adding the background color, the 3-D effects on the fixed cells, and the focus rectangle around the selected cell. Set DefaultDrawing to false to skip the painting of the cells when the grid is drawn.

When DefaultDrawing is false, the cells should be painted using the DrawCell method. For TCustomGrid, DrawCell is an abstract method. Descendants of TCustomGrid use DrawCell to provide their own customized drawing code, or to generate an OnDrawCell event.

See Also

DrawCell (see page 14826)

OnDrawCell (see page 14914)

168.1.2.5.12 TValueListEditor.DefaultRowHeight Property

Specifies the height (in pixels) of all rows that have not been explicitly resized.

Pascal

```
property DefaultRowHeight: Integer;
```

C++

```
__property int DefaultRowHeight;
```

Description

Set DefaultRowHeight to change the size of all rows in the grid. When DefaultRowHeight is set, rows that have been resized using the mouse or by setting the RowHeights property are given the DefaultRowHeight as well.

When new rows are added to the grid using the RowCount property, they are created with a height of DefaultRowHeight.

See Also

DefaultColWidth (see page 14846)

RowCount (see page 14854)

RowHeights (see page 14855)

168.1.2.5.13 TValueListEditor.DisplayOptions Property

Controls (see page 5711) aspects of the value list editor's appearance.

Pascal

```
property DisplayOptions: TDisplayOptions;
```

C++

```
__property TDisplayOptions DisplayOptions;
```

Description

Use DisplayOptions to control whether titles appear at the top of the columns in the value list editor and how the control responds when it is resized.

See Also

KeyOptions (see page 23445)

Options (see page 14853)

TitleCaptions (see page 23457)

168.1.2.5.14 TValueListEditor.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor: TCursor;
```

C++

```
__property TCursor DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor (see page 5815)

DragKind (see page 5819)

DragMode (see page 5819)

168.1.2.5.15 TValueListEditor.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind: TDragKind;
```

C++

```
__property TDragKind DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor (see page 5818)

DragMode (see page 5819)

168.1.2.5.16 TValueListEditor.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode: TDragMode;
```

C++

```
__property TDragMode DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != (TObject *)TreeView1 || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == (TObject *)TreeView1);
```

```
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag (see page 5740)

DragCursor (see page 5818)

EndDrag (see page 5757)

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

DragKind (see page 5819)

Arranging the Order of Persistent Columns

168.1.2.5.17 TValueListEditor.DropDownRows Property

Limits the length of the drop-down list that appears for values that use a pick list.

Pascal

```
property DropDownRows: Integer;
```

C++

```
__property int DropDownRows;
```

Description

Use DropDownRows to indicate the maximum height of the drop-down list that appears when the user clicks the drop-down button on values that have a pick list. DropDownRows is an upper limit: if the pick list has fewer entries that DropDownRows, the drop-down list includes only a line for every item in the list.

DropDownRows is ignored for values that do not have a pick list. To provide a value in the value list editor with a drop-down button and pick list, you do either of the following:

For the appropriate entry in the ItemProps (see page 23445) property, set the EditStyle property to esPickList and set the PickList property to the list of items.

Write an OnGetPickList (see page 23477) event handler that supplies the items in the list.

See Also

OnGetPickList (see page 23477)

Values (see page 23457)

ItemProps (see page 23445)

168.1.2.5.18 TValueListEditor.EditList Property

Pascal

```
property EditList: TInplaceEditList;
```

C++

```
__property TInplaceEditList EditList;
```

Description

This is EditList, a member of class TValueListEditor.

168.1.2.5.19 TValueListEditor.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled: Boolean;
```

C++

```
__property Boolean Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See Also

Visible (see page 5847)

DisableControls (see page 6925)

EnableControls (see page 6934)

Enabling Mouse

168.1.2.5.20 TValueListEditor.FixedColor Property

Specifies the background color of the fixed rows and columns in the grid.

Pascal

```
property FixedColor: TColor;
```

C++

```
__property TColor FixedColor;
```

Description

Set FixedColor to specify the color for the nonscrolling cells in the grid. The scrolling region of the grid is drawn using the Color property.

See Also

Color (see page 5812)

FixedCols (see page 14848)

FixedRows (see page 14849)

168.1.2.5.21 TValueListEditor.FixedCols Property

Specifies the number of columns on the left of the grid that cannot be scrolled.

Pascal

```
property FixedCols: Integer;
```

C++

```
__property int FixedCols;
```

Description

Set FixedCols to create or get rid of nonscrolling columns. Nonscrolling columns appear at the left of the grid, and are always visible, even when the user scrolls the other columns in the grid. Use nonscrolling columns for displaying row titles or row numbers, or to implement a scroll lock that the user can set.

Nonscrolling columns are visually distinct from the scrollable columns in the grid. They are painted with a separate background color, and their cells have a raised, 3-D look.

Note: Grids must include at least one scrolling column. Do not set FixedCols to a value greater than ColCount - 1.

See Also

ColCount (see page 14844)

DefaultDrawing (see page 14847)

FixedColor (see page 14848)

FixedRows (see page 14849)

168.1.2.5.22 TValueListEditor.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont (see page 5818)

ParentFont (see page 5834)

Displaying and Editing Text in a Memo Control

168.1.2.5.23 TValueListEditor.GridLineWidth Property

Specifies the width (in pixels) of the lines that separate the cells of the grid.

Pascal

```
property GridLineWidth: Integer;
```

C++

```
__property int GridLineWidth;
```

Description

Set GridLineWidth to make the lines that separate the cells in the grid heavier or lighter. When GridLineWidth is zero, no separators are drawn between the cells of the grid.

GridLineWidth will have no effect if the Options property does not include goFixedHorzLine, goFixedVertLine, goHorzLine, or goVertLine.

Note: Values greater than 3 pixels are not recommended for applications that run on Windows 9x or Windows ME because of distortions that can appear.

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if DrawGrid1.DefaultColWidth > 90 then
    DrawGrid1.GridLineWidth := 2
  else
    DrawGrid1.GridLineWidth := 1;
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
  end;
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

FixedCols (see page 14848)

FixedRows (see page 14849)

Options (see page 14853)

168.1.2.5.24 TValueListEditor.ItemProps Property

Specifies various constraints on the way users can enter a value for each item in the list.

Pascal

```
property ItemProps [const KeyOrIndex: Variant]: TItemProp;
```

C++

```
__property TItemProp ItemProps[Variant const KeyOrIndex];
```

Description

Use ItemProps to indicate, for each item:

- Whether the item is read only.
- The maximum length of strings the user can enter.
- An edit mask that limits the strings the user can enter as a value.
- Whether the item lets users drop down a pick list or launch a dialog to enter values.
- A fixed set of values from which users select (a pick list).

See Also

OnGetPickList (see page 23477)

Values (see page 23457)

168.1.2.5.25 TValueListEditor.KeyOptions Property

Controls (see page 5711) the way the value list editor responds to user keystrokes.

Pascal

```
property KeyOptions: TKeyOptions;
```

C++

```
__property TKeyOptions KeyOptions;
```

Description

Use KeyOptions to control how users can edit name/value pairs using the value list editor.

See Also

Keys (see page 23445)

Strings (see page 23456)

168.1.2.5.26 TValueListEditor.Keys Property

Lists the strings for each cell in the name (key) column.

Pascal

```
property Keys [Index: Integer]: string;
```

C++

```
__property AnsiString Keys[int Index];
```

Description

Use Keys to access the name (or key) value for a particular row. That is, Keys assumes the strings in the associated string list have the form Name=Value, and returns the Name portion.

Index identifies the Row of the value list editor. It takes a value from 0 to the number of rows. When DisplayOptions (see page 23438) includes doColumnTitles, the value of Keys is the title of the first column when Index is 0, and is the Name portion of the string when Index is greater than 0. When DisplayOptions (see page 23438) does not include doColumnTitles, the value of Keys is the Name portion of the string in the specified row.

To access the strings in the value column, use the Values (see page 23457) property instead. To access the underlying string list, use the Strings (see page 23456) property.

See Also

Cells (see page 23433)

Values (see page 23457)

Strings (see page 23456)

DisplayOptions (see page 23438)

Names (see page 3075)

168.1.2.5.27 TValueListEditor.Options Property

Specifies various display and behavioral properties of the grid.

Pascal

```
property Options: TGridOptions;
```

C++

```
__property TGridOptions Options;
```

Description

Set Options to include the desired properties for the grid. See the type Grids.TGridOptions for definitions of these properties.

See Also

ColWidths (see page 14788)

EditorMode (see page 14848)

GridLineWidth (see page 14850)

RowHeights (see page 14855)

Selection (see page 14856)

TabStops (see page 14858)

168.1.2.5.28 TValueListEditor.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode: Boolean;
```

C++

```
__property Boolean ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent (see page 5832)

BiDiMode (see page 5803)

168.1.2.5.29 TValueListEditor.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor: Boolean;
```

C++

```
__property Boolean ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color (see page 5812)

Parent (see page 5832)

ParentFont (see page 5834)

ParentShowHint (see page 5835)

168.1.2.5.30 TValueListEditor.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D: Boolean;
```

C++

```
__property Boolean ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D (see page 6198)

168.1.2.5.31 TValueListEditor.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont: Boolean;
```

C++

```
__property Boolean ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font (see page 5823)

Parent (see page 5832)

ParentColor (see page 5834)

ParentShowHint (see page 5835)

168.1.2.5.32 TValueListEditor.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint: Boolean;
```

C++

```
__property Boolean ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint (see page 5827)

Parent (see page 5832)

ParentColor (see page 5834)

ParentFont (see page 5834)

ShowHint (see page 5840)

168.1.2.5.33 TValueListEditor.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu: TPopupMenu;
```

C++

```
__property TPopupMenu PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      ((TEdit *)pComponent)->CopyToClipboard();
    else
      MessageBeep(0);
  }
```

```
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Cut1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        ((TEdit *)pComponent)->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      const int num_items = 4;
      for (int index = 0; index < num_items; ++index)
      {
          TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
          PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
          NewItem->Caption = "Menu Item " + IntToStr(index);
          NewItem->Tag = index;
          NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
          TNotifyEvent();
      }
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
          {
              case 0:
              {
                  ShowMessage("first item clicked");
```

```
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
```

```
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormDestroy(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
```

```
    for index := 0 to 3 do
    begin
      NewItem :=
        TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;
```

See Also

AutoPopup (see page 16178)

Popup (see page 16174)

OnContextPopup (see page 5854)

168.1.2.5.34 TValueListEditor.RowCount Property

Indicates the number of rows in the value list editor.

Pascal

```
property RowCount: Integer;
```

C++

```
__property int RowCount;
```

Description

TValueListEditor (see page 23401) redeclares the RowCount property, which it inherits from TCustomGrid, to change it to a read-only property. The value of RowCount is the number of strings in the Strings (see page 23456) property plus the number of fixed rows (typically one, for the column titles).

See Also

Strings (see page 23456)

ColCount (see page 23434)

FixedRows (see page 14849)

168.1.2.5.35 TValueListEditor.ScrollBars Property

Specifies whether the grid includes horizontal and vertical scroll bars.

Pascal

```
property ScrollBars: TScrollStyle;
```

C++

```
__property TScrollStyle ScrollBars;
```

Description

Use ScrollBars to give the grid horizontal or vertical scroll bars.

Note: If all the cells in the grid fit in the ClientWidth, no horizontal scroll bar appears, even if ScrollBars is ssHorizontal or ssBoth. If all the cells fit in the ClientHeight, no vertical scroll bar appears, even if ScrollBars is ssVertical or ssBoth.

See Also

ClientHeight (see page 5808)

ClientWidth (see page 5811)

168.1.2.5.36 TValueListEditor.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint: Boolean;
```

C++

```
__property Boolean ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint (see page 5827)

ParentShowHint (see page 5835)

ShowHint (see page 13721)

Displaying Fly-over Help

168.1.2.5.37 TValueListEditor.Strings Property

Provides access to the string list that is being edited.

Pascal

```
property Strings: TStrings;
```

C++

```
__property TStrings Strings;
```

Description

Use Strings to access the string list that the value list editor displays. Strings should be a TStrings descendant that contains strings of the form Name=Value.

At design time, add values to the Strings property using the string list editor. At runtime, you can use the methods of TStrings to add strings to the list one by one, or assign an existing TStrings descendant to the Strings property. When you set Strings at runtime, TValueListEditor (see page 23401) copies the strings from the TStrings object you assign: it does not take ownership of the object itself.

When Strings is assigned, the value list editor displays the name portion (TStrings . ::Names) of the strings in its key column and the value portion (TStrings . ::Values (see page 23457)) in the value column.

See Also

Names (see page 3075)

Values (see page 3086)

KeyOptions (see page 23445)

168.1.2.5.38 TValueListEditor.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder: TTabOrder;
```

C++

```
__property TTabOrder TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl (see page 6149)

TabStop (see page 6211)

168.1.2.5.39 TValueListEditor.TitleCaptions Property

Specifies the captions that label the columns of the value list editor.

Pascal

```
property TitleCaptions: TStrings;
```

C++

```
__property TStrings TitleCaptions;
```

Description

Use TitleCaptions to provide titles for the columns of the value list editor. TitleCaptions is a TStrings descendant that contains two strings: the first string labels the name (key) column, and the second string labels the value column. If TitleCaptions contains less than two strings, missing captions are treated as empty strings. If TitleCaptions contains more than two strings, all strings except the first two are ignored.

Note: TitleCaptions is only used when the DisplayOptions (see page 23438) property includes doColumnTitles.

See Also

DisplayOptions (see page 23438)

168.1.2.5.40 TValueListEditor.Values Property

Lists the strings for each value from the value column.

Pascal

```
property Values [const Key: string]: string;
```

C++

```
__property AnsiString Values[AnsiString const Key];
```

Description

Use Values to access the value associated with a specified name from the first (key) column. Key specifies a string that appears in the first column of the value list editor. Values is the corresponding string from the second (value) column.

To access the strings in the key column, use the Keys (see page 23445) property instead. To access the underlying string list, use the Strings (see page 23456) property.

Note: Setting Values with Key set to a name that is not in the list adds a new string to Strings (see page 23456) and a new row to the value list editor.

See Also

Cells (see page 23433)

Keys (see page 23445)

Strings (see page 23456)

Values (see page 3086)

168.1.2.5.41 TValueListEditor.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
```

```
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled (see page 5821)

Hide (see page 5769)

Show (see page 5788)

VisibleChanging (see page 5791)

168.1.2.5.42 TValueListEditor.VisibleColCount Property

Indicates the number of scrollable columns visible in the grid.

Pascal

```
property VisibleColCount: Integer;
```

C++

```
__property int VisibleColCount;
```

Description

Use VisibleColCount to determine the number of scrollable columns fully visible in the grid. VisibleColCount does not include the fixed columns counted by the FixedCols property. It does not include any partially visible columns on the right edge of the grid.

See Also

ColCount (see page 14844)

FixedCols (see page 14848)

VisibleRowCount (see page 14859)

168.1.2.5.43 TValueListEditor.VisibleRowCount Property

Indicates the number of scrollable rows visible in the grid.

Pascal

```
property VisibleRowCount: Integer;
```

C++

```
__property int VisibleRowCount;
```

Description

Use VisibleRowCount to determine the number of scrollable rows fully visible in the grid. VisibleRowCount does not include the fixed rows counted by the FixedRows property. It does not include any partially visible rows on the bottom of the grid.

See Also

FixedRows (see page 14849)

RowCount (see page 14854)

VisibleColCount (see page 14859)

168.1.2.6 TValueListEditor Events

The events of the TValueListEditor class are listed here.

Published Events

	Name	Description
	OnClick (see page 23461)	Occurs when the user clicks the control.
	OnContextPopup (see page 23463)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 23463)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 23464)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 23466)	Occurs when the user drags an object over a control.
	OnDrawCell (see page 23468)	Occurs when a cell in the grid needs to be drawn.
	OnEditButtonClick (see page 23470)	Occurs when the user click the ellipsis button on a cell in the values column or types Ctrl+Enter while focus is on a value cell with an ellipsis button.
	OnEndDock (see page 23470)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 23471)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 23472)	Occurs when a control receives the input focus.
	OnExit (see page 23474)	Occurs when the input focus shifts away from one control to another.
	OnGetEditMask (see page 23476)	Occurs when the in-place editor requests an edit mask.
	OnGetEditText (see page 23476)	Occurs when the in-place editor requests the value of a cell.
	OnGetPickList (see page 23477)	Supplies a pick list for values associated with a particular key.
	OnKeyDown (see page 23478)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 23480)	Occurs when a key is pressed.
	OnKeyUp (see page 23481)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 23483)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 23484)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 23486)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 23487)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 23487)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 23489)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheelDown (see page 23492)	Occurs when the mouse wheel is rotated downward.

	OnMouseWheelUp (see page 23492)	Occurs when the mouse wheel is rotated upward.
	OnRowMoved (see page 23492)	Occurs immediately after the position of a row changes.
	OnSelectCell (see page 23493)	Occurs before a cell in the grid is selected.
	OnSetEditText (see page 23494)	Occurs when the user edits the value of a cell.
	OnStartDock (see page 23495)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 23495)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnStringsChange (see page 23496)	Occurs immediately after the names or values in the string list change.
	OnStringsChanging (see page 23496)	Occurs immediately before the names or values in the string list change.
	OnTopLeftChanged (see page 23497)	Occurs immediately after the TopRow property or the LeftCol property changes.
	OnValidate (see page 23497)	Occurs when focus shifts away from a cell in the value list editor.

168.1.2.6.1 TValueListEditor.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}
```

```
void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this);
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this);
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick (see page 5854)

Cancel (see page 20929)

Default (see page 20932)

Click (see page 16116)

OnExecute (see page 2716)

Action (see page 5797)

Adding Silent Video Clips to an Application

168.1.2.6.2 TValueListEditor.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup: TContextPopupEvent;
```

C++

```
__property TContextPopupEvent OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu (see page 16169)

PopupMenu (see page 5835)

AutoPopup (see page 16178)

168.1.2.6.3 TValueListEditor.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick (see page 5851)

Responding to User Actions at Runtime

168.1.2.6.4 TValueListEditor.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop: TDragDropEvent;
```

C++

```
__property TDragDropEvent OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
```

```
      end;
    end;
  end;
```

See Also

BeginDrag (see page 5740)

DragDrop (see page 5754)

Dragging (see page 5754)

EndDrag (see page 5757)

OnDragOver (see page 5857)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

168.1.2.6.5 TValueListEditor.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver: TDragOverEvent;
```

C++

```
__property TDragOverEvent OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
```

```
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = (TListBox *)Sender;
    DestList->Font = ((TLabel *)Source)->Font;
    DestList->Color = ((TLabel *)Source)->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor (see page 5818)

OnDragDrop (see page 5855)

OnEndDrag (see page 5860)

OnStartDrag (see page 5871)

TDragObject (see page 6031)

DragKind (see page 5819)

OnEndDock (see page 5859)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

168.1.2.6.6 TValueListEditor.OnDrawCell Event

Occurs when a cell in the grid needs to be drawn.

Pascal

```
property OnDrawCell: TDrawCellEvent;
```

C++

```
__property TDrawCellEvent OnDrawCell;
```

Description

Write an OnDrawCell event handler to draw the contents of all the cells in the grid. Draw on the cell using the methods of the Canvas property. The Rect parameter indicates the location of the cell on the canvas. The Col and Row parameters indicate the

column and row indexes of the cell that should be drawn. The State parameter indicates whether the cell has input focus, whether the cell is selected, and whether the cell is a fixed (nonscrolling) cell.

If the OnDrawCell event handler is not assigned, all cells in the draw grid will appear empty. If the DefaultDrawing property is true, the draw grid paints the background color of the cell before the OnDrawCell event, and draws a focus rectangle around the selected cell after the OnDrawCell event handler finishes drawing the contents of the cell. If the DefaultDrawing property is false, the OnDrawCell event handler should paint the background of the cell and provide all visual indication of selection and focus.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas,Rect.Left,Rect.Top,index);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(index) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
```

```
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

Canvas (see page 5921)

DefaultDrawing (see page 14847)

168.1.2.6.7 TValueListEditor.OnEditButtonClick Event

Occurs when the user click the ellipsis button on a cell in the values column or types Ctrl+Enter while focus is on a value cell with an ellipsis button.

Pascal

```
property OnEditButtonClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEditButtonClick;
```

Description

Write an OnEditButtonClick to display a dialog that lets users edit a value in the Values (see page 23457) column. This event only occurs for items for which the ItemProps (see page 23445) property has an EditStyle of esEllipsis.

In the event handler, you can use the Row property to identify the row that the user is editing. The event handler (or the dialog that the event handler displays) can use the Keys (see page 23445) property to identify the current key value, and use that to get and set the corresponding entry in the Values (see page 23457) property that is being edited.

See Also

Values (see page 23457)

ItemProps (see page 23445)

EditStyle (see page 23398)

Row (see page 14854)

Keys (see page 23445)

168.1.2.6.8 TValueListEditor.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

OnStartDock (see page 5871)

168.1.2.6.9 TValueListEditor.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag: TEndDragEvent;
```

C++

```
__property TEndDragEvent OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = ((TLabel *)Sender)->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop (see page 5855)

OnDragOver (see page 5857)

OnStartDrag (see page 5871)

DragKind (see page 5819)

168.1.2.6.10 TValueListEditor.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   ((TEdit *)myActiveControl)->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    ((TRichEdit *)myActiveControl)->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
```

```
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
```

```
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit (see page 6221)

ActiveControl (see page 13896)

OnActivate (see page 13943)

Using a Single Navigator for Multiple Datasets

168.1.2.6.11 TValueListEditor.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK

button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter (see page 6219)

ActiveControl (see page 13896)

168.1.2.6.12 TValueListEditor.OnGetEditMask Event

Occurs when the in-place editor requests an edit mask.

Pascal

```
property OnGetEditMask: TGetEditEvent;
```

C++

```
__property TGetEditEvent OnGetEditMask;
```

Description

Write an OnGetEditMask event handler to provide the in-place editor with an edit mask that describes the valid input strings a user can type into a cell. Set the Value parameter to the string that describes the edit mask for the cell specified by the ACol and ARow parameters. If the cell does not have an edit mask, set the Value parameter to an empty string.

OnGetEditMask does not occur unless the Options property includes goEditing.

Delphi Examples:

```
{
This example specifies an edit mask in a string grid to
limit the cells that contain telephone numbers, as indicated
by a label cell in the first row.  Include goEditing in the
TStringGrid Options property.  Set the title of one of the
columns to 'Phone Number'.  Set the DefaultColWidth large
enough to see a phone number.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  StringGrid1.Cells[2, 0]:= 'Phone Number';
end;

procedure TForm1.StringGrid1GetEditMask(Sender: TObject; ACol, ARow: Longint; var Value:
string);
begin
  if StringGrid1.Cells[ACol, 0] = 'Phone Number' then
    Value :=  '!\(999\)000-0000;1';
end;
```

See Also

Options (see page 14853)

EditMask (see page 15821)

168.1.2.6.13 TValueListEditor.OnGetEditText Event

Occurs when the in-place editor requests the value of a cell.

Pascal

```
property OnGetEditText: TGetEditEvent;
```

C++

```
__property TGetEditEvent OnGetEditText;
```

Description

Write an OnGetEditText event handler to provide the in-place editor with a string representing the contents of the cell. Set the Value parameter to the string for the cell specified by the ACol and ARow parameters. When the cell enters edit mode, the contents of the cell are drawn as the Value string returned by the OnGetEditText event handler. This image need not match the appearance of the cell when it is not in edit mode, which is drawn using the OnDrawCell event handler.

OnGetEditText does not occur unless the Options property includes goEditing.

Delphi Examples:

```
{
The following code lowercases the text for a string grid
cell when the user begins to edit it.
}
procedure TForm1.StringGrid1GetEditText(Sender: TObject; ACol, ARow: Longint; var Value:
string);
begin
  Value := AnsiLowerCase(Value);
end;
```

See Also

OnDrawCell (see page 14795)

OnSetEditText (see page 14800)

168.1.2.6.14 TValueListEditor.OnGetPickList Event

Supplies a pick list for values associated with a particular key.

Pascal

```
property OnGetPickList: TGetPickListEvent;
```

C++

```
__property TGetPickListEvent OnGetPickList;
```

Description

Write an OnGetPickList event handler to provide a drop-down list of values for specific names from the first (key) column. OnGetPickList is called when the value column gets focus and the name (key) column contains the string specified by KeyName.

If the ItemProps (see page 23445) property has an EditStyle of esPickList for this item, The Values (see page 23457) parameter of the event handler is initialized to the value of its PickList property for this item. Otherwise, Values (see page 23457) is initialized to an empty list. In either case, the event handler can change the list of values that appear in the drop-down pick list.

If the OnGetPickList event handler exits with a non-empty string list as the value of the Values (see page 23457) parameter, a drop-down arrow appears in the value column. The user can click this arrow button to display the pick list you supplied as the value of Values (see page 23457). This makes the Value column behave like a drop-down combo box: the user can select items from the list or type in an entirely different value.

See Also

Keys (see page 23445)

Values (see page 23457)

DropDownRows (see page 23440)

ItemProps (see page 23445)

PickList (see page 23400)

EditStyle (see page 23398)

168.1.2.6.15 TValueListEditor.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

```
void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview (see page 13918)

OnKeyPress (see page 6227)

OnKeyUp (see page 6228)

168.1.2.6.16 TValueListEditor.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress: TKeyPressEvent;
```

C++

```
__property TKeyPressEvent OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown (see page 6225)

OnKeyUp (see page 6228)

KeyPreview (see page 13918)

168.1.2.6.17 TValueListEditor.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp: TKeyEvent;
```

C++

```
__property TKeyEvent OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
```

```
  event to FormKeyUp.  Set the KeyPreview on the form to True.
  }
  var
    FormColor: TColor;
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    KeyPreview := True;
    FormColor := Form1.Color;
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
```

See Also

OnKeyDown (see page 6225)

OnKeyPress (see page 6227)

168.1.2.6.18 TValueListEditor.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate: TMouseActivateEvent;
```

C++

```
__property TMouseActivateEvent OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

168.1.2.6.19 TValueListEditor.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
```

```
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
```

```
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick (see page 5851)

OnMouseMove (see page 5865)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

168.1.2.6.20 TValueListEditor.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

168.1.2.6.21 TValueListEditor.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

168.1.2.6.22 TValueListEditor.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove: TMouseMoveEvent;
```

C++

```
__property TMouseMoveEvent OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
```

```
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
```

```
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseUp (see page 5867)

MouseCapture (see page 5831)

168.1.2.6.23 TValueListEditor.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp: TMouseEvent;
```

C++

```
__property TMouseEvent OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form’s canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form’s unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
```

```
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown (see page 5862)

OnMouseMove (see page 5865)

Responding to a Mouse-up Action

168.1.2.6.24 TValueListEditor.OnMouseWheelDown Event

Occurs when the mouse wheel is rotated downward.

Pascal

```
property OnMouseWheelDown: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelDown;
```

Description

Write code in the OnMouseWheelDown event handler to perform actions when the mouse wheel is rotated downward. OnMouseWheelDown occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheel (see page 5869)

OnMouseWheelUp (see page 5870)

168.1.2.6.25 TValueListEditor.OnMouseWheelUp Event

Occurs when the mouse wheel is rotated upward.

Pascal

```
property OnMouseWheelUp: TMouseWheelUpDownEvent;
```

C++

```
__property TMouseWheelUpDownEvent OnMouseWheelUp;
```

Description

Write code in the OnMouseWheelUp event handler to perform actions when the mouse wheel is rotated upward. OnMouseWheelUp occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheelDown (see page 5869)

OnMouseWheel (see page 5869)

168.1.2.6.26 TValueListEditor.OnRowMoved Event

Occurs immediately after the position of a row changes.

Pascal

```
property OnRowMoved: TMovedEvent;
```

C++

```
__property TMovedEvent OnRowMoved;
```

Description

Write an OnRowMoved event handler to provide special processing whenever a row in the grid changes position. Rows can be moved programmatically or by user manipulation. OnRowMoved does not occur unless the Options property includes goRowMoving.

Delphi Examples:

```
{
The following code displays the number of rows a row was
moved in a label.  To move rows in a TStringGrid, goRowMoving
must be included in the Options property.
}
procedure TForm1.StringGrid1RowMoved(Sender: TObject; FromIndex, ToIndex: Longint);
begin
  Label1.Caption := IntToStr(Abs(FromIndex-ToIndex));
end;
```

See Also

OnColumnMoved (see page 14794)

Options (see page 14853)

168.1.2.6.27 TValueListEditor.OnSelectCell Event

Occurs before a cell in the grid is selected.

Pascal

```
property OnSelectCell: TSelectCellEvent;
```

C++

```
__property TSelectCellEvent OnSelectCell;
```

Description

Write an OnSelectCell event handler to specify whether any particular cell in the grid can be selected. The Col and Row parameters indicate the column and row indexes of the cell that is about to be selected. Set the CanSelect parameter to false to prevent the cell being selected.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas,Rect.Left,Rect.Top,index);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(index) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
{
The following code ensures that the user cannot select a cell unless it is empty.
}
procedure TForm1.StringGrid1SelectCell(Sender: TObject; ACol, ARow: Longint; var CanSelect:
Boolean);
begin
  CanSelect := (StringGrid1.Cells[ACol, ARow]='')
end;
```

See Also

Col (see page 14843)

Row (see page 14854)

TabStops (see page 14858)

168.1.2.6.28 TValueListEditor.OnSetEditText Event

Occurs when the user edits the value of a cell.

Pascal

```
property OnSetEditText: TSetEditEvent;
```

C++

```
__property TSetEditEvent OnSetEditText;
```

Description

Write an OnSetEditText event handler to perform any special processing of the text edited by the user in an in-place editor. For example, use the OnSetEditText event to retrieve and store the value of a cell so that it can be displayed in an OnDrawCell event handler. OnSetEditText occurs every time the user changes the text.

The Value parameter is the new value for the cell specified by the ACol and ARow parameters.

OnSetEditText does not occur unless the Options property includes goEditing.

See Also

OnDrawCell (see page 14795)

OnGetEditText (see page 14797)

Options (see page 14853)

168.1.2.6.29 TValueListEditor.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock: TStartDockEvent;
```

C++

```
__property TStartDockEvent OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock (see page 5859)

TDragDockObjectEx (see page 6018)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDockDrop (see page 6214)

168.1.2.6.30 TValueListEditor.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag: TStartDragEvent;
```

C++

```
__property TStartDragEvent OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag (see page 5860)

TDragControlObjectEx (see page 6007)

DragKind (see page 5819)

OnDragOver (see page 5857)

OnDragDrop (see page 5855)

168.1.2.6.31 TValueListEditor.OnStringsChange Event

Occurs immediately after the names or values in the string list change.

Pascal

```
property OnStringsChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnStringsChange;
```

Description

Write an OnStringsChange event handler to respond when the strings in the list change. For example, you can use the OnStringsChange event to update your display to reflect the new values of the strings in the value list editor.

OnStringsChange occurs after the change is complete. The properties of the value list editor reflect the new values of the strings.

See Also

OnStringsChanging (see page 23496)

Strings (see page 23456)

OnSetEditText (see page 14800)

168.1.2.6.32 TValueListEditor.OnStringsChanging Event

Occurs immediately before the names or values in the string list change.

Pascal

```
property OnStringsChanging: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnStringsChanging;
```

Description

Write an OnStringsChanging event handler to prepare for changes in the list of strings. For example, you can use the OnStringsChanging event to disable repaints until the OnStringsChange (see page 23496) event when the list has finished changing..

OnStringsChanging occurs before the change occurs. The properties of the value list editor reflect the old values of the strings.

See Also

OnStringsChange (see page 23496)

Strings (see page 23456)

OnSetEditText (see page 14800)

168.1.2.6.33 TValueListEditor.OnTopLeftChanged Event

Occurs immediately after the TopRow property or the LeftCol property changes.

Pascal

```
property OnTopLeftChanged: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnTopLeftChanged;
```

Description

Use OnTopLeftChanged to perform special processing when the non-fixed cells in the grid are scrolled.

See Also

LeftCol (see page 14852)

TopRow (see page 14858)

168.1.2.6.34 TValueListEditor.OnValidate Event

Occurs when focus shifts away from a cell in the value list editor.

Pascal

```
property OnValidate: TOnValidateEvent;
```

C++

```
__property TOnValidateEvent OnValidate;
```

Description

Write an OnValidate event handler to validate any edits the user enters in a cell before focus leaves it. OnValidate gives applications an opportunity to provide more validation than the EditMask property of the corresponding TItemProp (see page 23394) object can supply.

OnValidate only occurs if the user edited the value of the cell that is about to lose focus. The OnValidate event handler can verify the value the user supplied, and if it is not acceptable, raise an exception.

See Also

Keys (see page 23445)

Values (see page 23457)

ItemProps (see page 23445)

EditMask (see page 23398)

168.1.3 TValueListStrings Class

Class Hierarchy

System::TObject → Classes.TPersistent → Classes.TStrings → Classes.TStringList → ValEdit.TValueListStrings

Pascal

```
TValueListStrings = class(TStringList);
```

C++

```
class TValueListStrings : public TStringList;
```

File

ValEdit

Description

This is class ValEdit.TValueListStrings.

168.1.3.1 TValueListStrings Members

The following tables list the members exposed by TValueListStrings.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 23504)	This is Changed, a member of class TValueListStrings.
	Changing (see page 23505)	This is Changing, a member of class TValueListStrings.
	CompareStrings (see page 3014)	Compares two strings.
	CompareStrings (see page 3048)	Compares two strings.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DefineProperties (see page 3049)	Reads and writes the Strings (see page 3080) property as if it were published.
	Error (see page 3052)	This is the overview for the Error method overload.
	ExtractName (see page 3053)	Returns the name portion of a string that is a name value pair.
	FindItemProp (see page 23507)	This is FindItemProp, a member of class TValueListStrings.
	Get (see page 3017)	Returns a string given its index.
	Get (see page 3053)	Returns a string given its index.
	GetCapacity (see page 3017)	Returns the currently allocated size of the list of strings.
	GetCapacity (see page 3054)	Returns the currently allocated size of the list of strings.
	GetCount (see page 3018)	Returns the number of strings in the list
	GetCount (see page 3054)	Returns the number of strings in the list
	GetObject (see page 3018)	Returns the object associated with the string at a specified index.
	GetObject (see page 3055)	Returns the object associated with the string at a specified index.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTextStr (see page 3056)	Returns the value of the Text (see page 3084) property.
	InsertItem (see page 23507)	This is InsertItem, a member of class TValueListStrings.

	Put (see page 23507)	This is Put, a member of class TValueListStrings.
	PutObject (see page 3022)	Changes the object associated with the string at a specified index.
	PutObject (see page 3065)	Changes the object associated with the string at a specified index.
	SetCapacity (see page 3022)	Changes the amount of memory allocated to hold strings in the list.
	SetCapacity (see page 3068)	Changes the amount of memory allocated to hold strings in the list.
	SetTextStr (see page 3069)	Sets the Text (see page 3084) property.
	SetUpdateState (see page 3023)	Performs internal adjustments before or after a series of updates.
	SetUpdateState (see page 3069)	Performs internal adjustments before or after a series of updates.

Protected Properties

	Name	Description
	UpdateCount (see page 3085)	Indicates the number of calls to BeginUpdate (see page 3045) that have not been matched by a call to EndUpdate (see page 3050).

Public Events

	Name	Description
	OnChange (see page 3028)	Occurs immediately after the list of strings changes.
	OnChanging (see page 3028)	Occurs immediately before the list of strings changes.

Public Methods

	Name	Description
	Add (see page 3010)	Adds a new string to the list.
	Add (see page 3033)	Adds a string at the end of the list.
	AddObject (see page 3012)	Adds a string to the list, and associates an object with the string.
	AddObject (see page 3036)	Adds a string to the list, and associates an object with the string.
	AddStrings (see page 3039)	Adds a group of strings to the list.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 3041)	Adds a string to the list.
	Assign (see page 23502)	Sets the strings in the list, and possibly associated objects, from a source object.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 3045)	Enables the TStrings (see page 3029) object to track when the list of strings is changing.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 23505)	Deletes all the strings from the list.
	Create (see page 23501)	This is Create, a member of class TValueListStrings.
	CustomSort (see page 23505)	Sorts the strings in the list in a customized order.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 23506)	Removes the string specified by the Index parameter.
	Destroy (see page 23501)	Destroys an instance of TStringList.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 3050)	Enables the TStrings (see page 3029) object to keep track of when the list of strings has finished changing.
	Equals (see page 3051)	Compares the list of strings to the list from another TStrings (see page 3029) object and returns true if the two lists match.
	Exchange (see page 23506)	Swaps the position of two strings in the list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Find (see page 3016)	Locates the index for a string in a sorted list and indicates whether a string with that value already exists in the list.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 3054)	Returns a TStrings (see page 3029) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetText (see page 3055)	Allocates a text buffer and fills it with the value of the Text (see page 3084) property.
V	IndexOf (see page 3019)	Returns the position of a string in the list.
V	IndexOf (see page 3056)	Returns the position of a string in the list.
V	IndexOfName (see page 3057)	Returns the position of the first name-value pair with the specified name.
V	IndexOfObject (see page 3059)	Returns the index of the first string in the list associated with a given object.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Insert (see page 3019)	Inserts a string to the list at the position specified by Index.
A	Insert (see page 3059)	Introduces abstract (Delphi) or pure virtual (C++) method to insert a string at a specified position.
V	InsertObject (see page 3020)	Inserts a string into the list at the specified position, and associates it with an object.
V	InsertObject (see page 3060)	Inserts a string into the list at the specified position, and associates it with an object.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	KeyIsValid (see page 23507)	This is KeyIsValid, a member of class TValueListStrings.
V	LoadFromFile (see page 3061)	Fills the list with the lines of text in a specified file.
V	LoadFromStream (see page 3062)	Fills the list with lines of text read from a stream.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	Move (see page 3063)	Changes the position of a string in the list.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SaveToFile (see page 3065)	Saves the strings in the list to the specified file.
V	SaveToStream (see page 3067)	Writes the value of the Text (see page 3084) property to a stream object.
V	SetText (see page 3068)	Sets the Text (see page 3084) property.
V	Sort (see page 3023)	Sorts the strings in the list in ascending order.

Public Properties

	Name	Description
	Capacity (see page 3070)	Indicates the number of strings the TStrings (see page 3029) object can hold.
	CaseSensitive (see page 3025)	Controls (see page 5711) whether strings are located, sorted, and identified as duplicates in a case-sensitive or case-insensitive manner.
	CommaText (see page 3070)	Lists the strings in the TStrings (see page 3029) object in system data format (SDF).
R	Count (see page 3071)	Introduces an abstract property to represent the number of strings in the list.
	DelimitedText (see page 3074)	Represents all the strings in the TStrings (see page 3029) object as a single delimited string.
	Delimiter (see page 3074)	Specifies the delimiter used by the DelimitedText (see page 3074) property.
	Duplicates (see page 3025)	Specifies whether duplicate strings can be added to sorted lists.
	ItemProps (see page 23508)	This is ItemProps, a member of class TValueListStrings.
	LineBreak (see page 3075)	Defines the line break character(s).
R	Names (see page 3075)	Indicates the name part of strings that are name-value pairs.
	NameValueSeparator (see page 3077)	Indicates the character used to separate names from values.
	Objects (see page 3077)	Represents a set of objects that are associated one with each of the strings in the Strings (see page 3080) property.

	QuoteChar (see page 3080)	Specifies the quote character used by the DelimitedText (see page 3074) property.
	Sorted (see page 3026)	Specifies whether the strings in the list should be automatically sorted.
	StrictDelimiter (see page 3080)	Determines how the Delimiter (see page 3074) property is used.
	Strings (see page 3080)	References the strings in the list by their positions.
	StringsAdapter (see page 3084)	Implements an IStringsAdapter (see page 3208) interface for the TStrings (see page 3029) object.
	Text (see page 3084)	Lists the strings in the TStrings (see page 3029) object as a single string with the individual strings delimited by carriage returns and line feeds.
	ValueFromIndex (see page 3085)	Represents the value part of a string with a given index, on strings that are name-value pairs.
	Values (see page 3086)	Represents the value part of a string associated with a given name, on strings that are name-value pairs.

168.1.3.2 TValueListStrings.Create Constructor

Pascal

```
constructor Create(AEditor: TValueListEditor); reintroduce;
```

C++

```
__fastcall TValueListStrings(TValueListEditor AEditor);
```

Description

This is Create, a member of class TValueListStrings.

168.1.3.3 TValueListStrings.Destroy Destructor

Destroys an instance of TStringList.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TValueListStrings();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TStringList reference is not nil and only then calls Destroy.

Destroy frees the memory allocated to hold the list of strings and object references before calling the inherited destructor.

See Also

Free (see page 22300)

168.1.3.4 TValueListStrings Methods

The methods of the TValueListStrings class are listed here.

Protected Methods

	Name	Description
	Changed (see page 23504)	This is Changed, a member of class TValueListStrings.
	Changing (see page 23505)	This is Changing, a member of class TValueListStrings.
	FindItemProp (see page 23507)	This is FindItemProp, a member of class TValueListStrings.

	InsertItem (see page 23507)	This is InsertItem, a member of class TValueListStrings.
	Put (see page 23507)	This is Put, a member of class TValueListStrings.

Public Methods

	Name	Description
	Assign (see page 23502)	Sets the strings in the list, and possibly associated objects, from a source object.
	Clear (see page 23505)	Deletes all the strings from the list.
	CustomSort (see page 23505)	Sorts the strings in the list in a customized order.
	Delete (see page 23506)	Removes the string specified by the Index parameter.
	Exchange (see page 23506)	Swaps the position of two strings in the list.
	KeyIsValid (see page 23507)	This is KeyIsValid, a member of class TValueListStrings.

168.1.3.4.1 TValueListStrings.Assign Method

Sets the strings in the list, and possibly associated objects, from a source object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use Assign to set the value of the TStrings object from another object. If Source is of type TStrings, the list is set to the list of the source TStrings object, and if associated objects are supported, any associated objects are copied from the Source as well.

If Source is not of type TStrings, the inherited Assign will set the value of the list from any object that supports TStrings in its AssignTo method.

C++ Examples:

```
/*
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
*/
void __fastcall TForm1::Add_a_fileClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    int index = TabControl1->Tabs->Add(OpenDialog1->FileName);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[index]);
  }
}

void __fastcall TForm1::Assign_filesClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TabControl1->Tabs->Assign(OpenDialog1->Files);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[TabControl1->TabIndex]);
  }
```

```
}
/*
Place the following code in the tab control's OnChange event
handler:
*/
void __fastcall TForm1::TabControl1Change(TObject *Sender)
{
    Memo1->Lines->LoadFromFile(
      TabControl1->Tabs->Strings[TabControl1->TabIndex]);
}
/*
This example creates a stringlist, adds elements to it and
assigns the strings to items of a combo box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TStrings *StringList = new TStringList();
  try
  {
    StringList->Add("This example uses A string List. ");
    StringList->Add("It is the easiest way to add strings");
    StringList->Add("to a combobox's list of strings. ");
    StringList->Add("Always remember TStrings is abstract,");
    StringList->Add("So create a TStringList instead.");
    ComboBox1->Width = 210;
    ComboBox1->Items->Assign(StringList);
    ComboBox1->ItemIndex = 0;
  }
  __finally
  {
    delete StringList;
  }
}
```

Delphi Examples:

```
{
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
}
procedure TForm1.Add_a_fileClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    index:= TabControl1.Tabs.Add(OpenDialog1.FileName);
    Memo1.Lines.LoadFromFile(TabControl1.Tabs[index]);
  end;
end;

procedure TForm1.Assign_filesClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
```

```
      TabControl1.Tabs.Assign(OpenDialog1.Files);
      Memo1.Lines.LoadFromFile(
        TabControl1.Tabs[TabControl1.TabIndex]);
    end;
  end;

  {
  Place the following code in the tab control's OnChange event
  handler:
  }
  procedure TForm1.TabControl1Change(Sender: TObject);
  begin
    with TabControl1 do
      Memo1.Lines.LoadFromFile(Tabs[TabIndex]);
  end;
  {
  This example creates a stringlist, adds elements to it and
  assigns the strings to items of a combo box.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    StringList: TStringList;
  begin
    StringList := TStringList.Create;
    try
      with StringList do begin
        Add('This example uses A string List.');
        Add('It is the easiest way to add strings');
        Add('to a combobox''s list of strings.');
        Add('Always remember TStrings.Create method');
        Add('is abstract; So use TStringList.Create');
        Add('method instead.');
      end;
      with ComboBox1 do begin
        Width := 210;
        Items.Assign(StringList);
        ItemIndex := 0;
      end;
    finally
      StringList.free;
    end;
  end;
```

See Also

AddStrings (see page 3039)

Objects (see page 3077)

Strings (see page 3080)

AssignTo (see page 2931)

168.1.3.4.2 TValueListStrings.Changed Method

Pascal

```
procedure Changed; override;
```

C++

```
virtual __fastcall Changed();
```

Description

This is Changed, a member of class TValueListStrings.

168.1.3.4.3 TValueListStrings.Changing Method

Pascal

```
procedure Changing; override;
```

C++

```
virtual __fastcall Changing();
```

Description

This is Changing, a member of class TValueListStrings.

168.1.3.4.4 TValueListStrings.Clear Method

Deletes all the strings from the list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call clear to empty the list of strings. All references to associated objects are also removed. However, the objects themselves are not freed.

See Also

Delete (see page 3015)

Objects

Strings

168.1.3.4.5 TValueListStrings.CustomSort Method

Sorts the strings in the list in a customized order.

Pascal

```
procedure CustomSort(Compare: TStringListSortCompare); override;
```

C++

```
virtual __fastcall CustomSort(TStringListSortCompare Compare);
```

Description

Use CustomSort to sort the strings in the list, where the sort order is defined by the Compare parameter.

Supply a value for the Compare function that compares two strings in the string list. The List parameter provides access to the string list, while the Index1 and Index2 parameters identify the strings to be compared.

Do not pass nil (Delphi) or NULL (C++) as the value of the Compare parameter.

Note: You must explicitly call the CustomSort method. Setting the Sorted property only sorts strings using ANSI (Windows) or UTF-8 (Linux) order, as implemented in the Sort method.

See Also

Sort (see page 3023)

Strings

Sorted (see page 3026)

168.1.3.4.6 TValueListStrings.Delete Method

Removes the string specified by the Index parameter.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Call Delete to remove a single string from the list. If an object is associated with the string, the reference to the object is removed as well. Index gives the position of the string, where 0 is the first string, 1 is the second string, and so on.

See Also

Add (see page 3010)

AddObject (see page 3012)

Clear (see page 3013)

IndexOf (see page 3019)

InsertObject (see page 3060)

Objects

Strings

168.1.3.4.7 TValueListStrings.Exchange Method

Swaps the position of two strings in the list.

Pascal

```
procedure Exchange(Index1: Integer; Index2: Integer); override;
```

C++

```
virtual __fastcall Exchange(int Index1, int Index2);
```

Description

Call Exchange to rearrange the strings in the list. The strings are specified by their index values in the Index1 and Index2 parameters. Indexes are zero-based, so the first string in the list has an index value of 0, the second has an index value of 1, and so on.

If either string has an associated object, Exchange changes the index of the object as well.

Warning: Do not call Exchange on a sorted list except to swap two identical strings with different associated objects. Exchange does not check whether the list is sorted, and can destroy the sort order of a sorted list.

See Also

Move (see page 3063)

Objects

Sorted (see page 3026)

Strings

168.1.3.4.8 TValueListStrings.FindItemProp Method

Pascal

```
function FindItemProp(const KeyOrIndex: Variant; Create: Boolean = False): TItemProp;
```

C++

```
__fastcall TItemProp FindItemProp(const Variant KeyOrIndex, Boolean Create = False);
```

Description

This is FindItemProp, a member of class TValueListStrings.

168.1.3.4.9 TValueListStrings.InsertItem Method

Pascal

```
procedure InsertItem(Index: Integer; const S: string; AObject: TObject); override;
```

C++

```
virtual __fastcall InsertItem(int Index, const AnsiString S, TObject * AObject);
```

Description

This is InsertItem, a member of class TValueListStrings.

168.1.3.4.10 TValueListStrings.KeyIsValid Method

Pascal

```
function KeyIsValid(const Key: string; RaiseError: Boolean = True): Boolean;
```

C++

```
__fastcall Boolean KeyIsValid(const AnsiString Key, Boolean RaiseError = True);
```

Description

This is KeyIsValid, a member of class TValueListStrings.

168.1.3.4.11 TValueListStrings.Put Method

Pascal

```
procedure Put(Index: Integer; const S: String); override;
```

C++

```
virtual __fastcall Put(int Index, const AnsiString S);
```

Description

This is Put, a member of class TValueListStrings.

168.1.3.5 TValueListStrings Properties

The properties of the TValueListStrings class are listed here.

Public Properties

	Name	Description
	ItemProps (see page 23508)	This is ItemProps, a member of class TValueListStrings.

168.1.3.5.1 TValueListStrings.ItemProps Property

Pascal

```
property ItemProps [const KeyOrIndex: Variant]: TItemProp;
```

C++

```
__property TItemProp ItemProps[Variant const KeyOrIndex];
```

Description

This is ItemProps, a member of class TValueListStrings.

168.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TDisplayOption (see page 23509)	TDisplayOption and TDisplayOptions (see page 23511) control the appearance of a value list editor.
	TKeyOption (see page 23510)	TKeyOption and TKeyOptions (see page 23512) indicate the response of a value list editor to certain keystrokes.

168.2.1 ValEdit.TDisplayOption Enumeration

TDisplayOption and TDisplayOptions (see page 23511) control the appearance of a value list editor.

Pascal

```
TDisplayOption = (
  doColumnTitles,
  doAutoColResize,
  doKeyColFixed
);
```

C++

```
enum TDisplayOption {
  doColumnTitles,
  doAutoColResize,
  doKeyColFixed
};
```

File

ValEdit

Description

TDisplayOption is a set of TDisplayOptions (see page 23511) values. The possible values of TDisplayOptions (see page 23511) are

Option	Meaning
doColumnTitles	The first row of the value list editor is a fixed (nonscrolling) row that displays the captions specified by the TitleCaptions property.
doAutoColResize	When the editor is resized, the column widths automatically adjust so that they completely fill the new width of the editor. If DisplayOptions does not include doKeyColFixed, each column has a width equal to half the width of the editor. Otherwise, the value column takes up any space left over after the fixed (key) column.
doKeyColFixed	The first (key) column of the value list editor is fixed in width. When the value list editor is resized, only the second (value) column resizes. This option is only used when DisplayOptions includes doAutoColResize.

168.2.2 ValEdit.TKeyOption Enumeration

TKeyOption and TKeyOptions (see page 23512) indicate the response of a value list editor to certain keystrokes.

Pascal

```
TKeyOption = (
  keyEdit,
  keyAdd,
  keyDelete,
  keyUnique
);
```

C++

```
enum TKeyOption {
  keyEdit,
  keyAdd,
  keyDelete,
  keyUnique
};
```

File

ValEdit

Description

TKeyOption is a set of TKeyOptions (see page 23512) values. The possible values of TKeyOptions (see page 23512) are

Option	Meaning
keyEdit	The user can select and edit values in the first (key) column. That is, the user can specify the name in a name/value pair.
keyAdd	The user can add new name/value pairs. When included, the user can create new entries by pressing the Insert key or using the down arrow key when positioned on the last name/value pair. keyAdd requires that keyEdit is also included.
keyDelete	The user can delete the selected name/value pair by pressing the Delete key.
keyUnique	The user can't create name/value pairs where the name portion is the same as the name in another name/value pair. Attempting to add a duplicate name causes the value list editor to raise an exception. keyUnique requires that keyEdit is also included.

168.3 Types

The following table lists types in this documentation.

Types

Name	Description
TDisplayOptions (see page 23511)	This is type ValEdit.TDisplayOptions.
TGetPickListEvent (see page 23511)	This is type ValEdit.TGetPickListEvent.
TItemProps (see page 23512)	This is type ValEdit.TItemProps.
TKeyOptions (see page 23512)	This is type ValEdit.TKeyOptions.
TOnValidateEvent (see page 23512)	This is type ValEdit.TOnValidateEvent.

168.3.1 ValEdit.TDisplayOptions Type

Pascal

```
TDisplayOptions = set of TDisplayOption;
```

C++

```
TDisplayOption TDisplayOptions;
```

File

ValEdit

Description

This is type ValEdit.TDisplayOptions.

168.3.2 ValEdit.TGetPickListEvent Type

Pascal

```
TGetPickListEvent = procedure (Sender: TObject; const KeyName: string; Values: TStrings) of
object;
```

C++

```
(Sender: TObject; const KeyName: string; Values: TStrings) ( TGetPickListEvent)();
```

File

ValEdit

Description

This is type ValEdit.TGetPickListEvent.

168.3.3 ValEdit.TItemProps Type

Pascal

```
TItemProps = array of TItemProp;
```

C++

```
array of TItemProp TItemProps;
```

File

ValEdit

Description

This is type ValEdit.TItemProps.

168.3.4 ValEdit.TKeyOptions Type

Pascal

```
TKeyOptions = set of TKeyOption;
```

C++

```
TKeyOption TKeyOptions;
```

File

ValEdit

Description

This is type ValEdit.TKeyOptions.

168.3.5 ValEdit.TOnValidateEvent Type

Pascal

```
TOnValidateEvent = procedure (Sender: TObject; ACol, ARow: Longint; const KeyName, KeyValue:
string) of object;
```

C++

```
(Sender: TObject; ACol, ARow: Longint; const KeyName, KeyValue: string) ( TOnValidateEvent)();
```

File

ValEdit

Description

This is type ValEdit.TOnValidateEvent.

169 VarCmplx Namespace

This is namespace VarCmplx.

Topics

Name	Description
Functions (see page 23515)	The following table lists functions in this documentation.
Variables (see page 23559)	The following table lists variables in this documentation.

Functions

	Name	Description
	VarAsComplex (see page 23516)	Casts an arbitrary Variant so that it is a custom Variant representing a complex number.
	VarComplex (see page 23516)	Returns the Variant type code for custom Variants (see page 23567) that represent complex numbers.
	VarComplexAbs (see page 23517)	Returns the absolute value of a complex number.
	VarComplexAbsSqr (see page 23518)	Returns the square of the absolute value of a complex number.
	VarComplexAngle (see page 23518)	Returns the angle made by a complex number.
	VarComplexArcCos (see page 23519)	Returns the inverse cosine of a complex number.
	VarComplexArcCosH (see page 23520)	Returns the hyperbolic inverse cosine of a complex number.
	VarComplexArcCot (see page 23521)	Returns the inverse cotangent of a complex number.
	VarComplexArcCotH (see page 23522)	Returns the hyperbolic inverse cotangent of a complex number.
	VarComplexArcCsc (see page 23523)	Returns the inverse cosecant of a complex number.
	VarComplexArcCscH (see page 23524)	Returns the hyperbolic inverse cosecant of a complex number.
	VarComplexArcSec (see page 23525)	Returns the inverse secant of a complex number.
	VarComplexArcSecH (see page 23527)	Returns the hyperbolic inverse secant of a complex number.
	VarComplexArcSin (see page 23528)	Returns the inverse sine of a complex number.
	VarComplexArcSinH (see page 23529)	Returns the hyperbolic inverse sine of a complex number.
	VarComplexArcTan (see page 23530)	Returns the inverse tangent of a complex number.
	VarComplexArcTanH (see page 23531)	Returns the hyperbolic inverse tangent of a complex number.
	VarComplexConjugate (see page 23532)	Returns the conjugate of a complex number.
	VarComplexCos (see page 23533)	Returns the cosine of a complex number.
	VarComplexCosH (see page 23534)	Returns the hyperbolic cosine of a complex number.
	VarComplexCot (see page 23535)	Returns the cotangent of a complex number.
	VarComplexCotH (see page 23536)	Returns the hyperbolic cotangent of a complex number.
	VarComplexCreate (see page 23537)	Returns a custom Variant that represents a complex number.
	VarComplexCreate (see page 23537)	Returns a custom Variant that represents a complex number.

	VarComplexCreate (see page 23537)	Returns a custom Variant that represents a complex number.
	VarComplexCreate (see page 23537)	Returns a custom Variant that represents a complex number.
	VarComplexCsc (see page 23538)	Returns the cosecant of a complex number.
	VarComplexCscH (see page 23539)	Returns the hyperbolic cosecant of a complex number.
	VarComplexExp (see page 23540)	Returns the exponential of a complex number.
	VarComplexFromPolar (see page 23541)	Converts a set of polar coordinates to a complex number.
	VarComplexInverse (see page 23541)	Returns the inverse of a complex number.
	VarComplexLn (see page 23542)	Returns the natural log of a complex number.
	VarComplexLog10 (see page 23543)	Returns the log base 10 of a complex number.
	VarComplexLog2 (see page 23543)	Returns the log base 2 of a complex number.
	VarComplexLogN (see page 23544)	Returns the log base N of a complex number.
	VarComplexPower (see page 23545)	Returns the value of a complex number raised to a specified power.
	VarComplexSec (see page 23545)	Returns the secant of a complex number.
	VarComplexSecH (see page 23546)	Returns the hyperbolic secant of a complex number.
	VarComplexSign (see page 23548)	This is function VarCmplx.VarComplexSign.
	VarComplexSimplify (see page 23548)	Reduces a Variant that represents a complex number to one that represents a real value, if possible.
	VarComplexSin (see page 23548)	Returns the sine of a complex number.
	VarComplexSinH (see page 23550)	Returns the hyperbolic sine of a complex number.
	VarComplexSqr (see page 23551)	Returns the square of a complex number.
	VarComplexSqrt (see page 23551)	Returns the square root of a complex number.
	VarComplexTan (see page 23552)	Returns the tangent of a complex number.
	VarComplexTanH (see page 23553)	Returns the hyperbolic tangent of a complex number.
	VarComplexTimesImaginary (see page 23554)	Returns the value of a complex number multiplied by an imaginary number.
	VarComplexTimesNegI (see page 23555)	Returns the value of a complex number multiplied by –i.
	VarComplexTimesPosI (see page 23555)	Returns the value of a complex number multiplied by i.
	VarComplexTimesReal (see page 23556)	Returns the value of a complex number multiplied by a real number.
	VarComplexToPolar (see page 23556)	Computes the polar coordinates that correspond to a custom Variant that represents a complex number.
	VarIsComplex (see page 23557)	Indicates whether a Variant stores its data internally as a complex number.

Variables

Name	Description
ComplexNumberDefuzzAtZero (see page 23559)	This is variable VarCmplx.ComplexNumberDefuzzAtZero.
ComplexNumberSymbol (see page 23559)	This is variable VarCmplx.ComplexNumberSymbol.
ComplexNumberSymbolBeforeImaginary (see page 23560)	This is variable VarCmplx.ComplexNumberSymbolBeforeImaginary.

169.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	VarAsComplex (see page 23516)	Casts an arbitrary Variant so that it is a custom Variant representing a complex number.
	VarComplex (see page 23516)	Returns the Variant type code for custom Variants (see page 23567) that represent complex numbers.
	VarComplexAbs (see page 23517)	Returns the absolute value of a complex number.
	VarComplexAbsSqr (see page 23518)	Returns the square of the absolute value of a complex number.
	VarComplexAngle (see page 23518)	Returns the angle made by a complex number.
	VarComplexArcCos (see page 23519)	Returns the inverse cosine of a complex number.
	VarComplexArcCosH (see page 23520)	Returns the hyperbolic inverse cosine of a complex number.
	VarComplexArcCot (see page 23521)	Returns the inverse cotangent of a complex number.
	VarComplexArcCotH (see page 23522)	Returns the hyperbolic inverse cotangent of a complex number.
	VarComplexArcCsc (see page 23523)	Returns the inverse cosecant of a complex number.
	VarComplexArcCscH (see page 23524)	Returns the hyperbolic inverse cosecant of a complex number.
	VarComplexArcSec (see page 23525)	Returns the inverse secant of a complex number.
	VarComplexArcSecH (see page 23527)	Returns the hyperbolic inverse secant of a complex number.
	VarComplexArcSin (see page 23528)	Returns the inverse sine of a complex number.
	VarComplexArcSinH (see page 23529)	Returns the hyperbolic inverse sine of a complex number.
	VarComplexArcTan (see page 23530)	Returns the inverse tangent of a complex number.
	VarComplexArcTanH (see page 23531)	Returns the hyperbolic inverse tangent of a complex number.
	VarComplexConjugate (see page 23532)	Returns the conjugate of a complex number.
	VarComplexCos (see page 23533)	Returns the cosine of a complex number.
	VarComplexCosH (see page 23534)	Returns the hyperbolic cosine of a complex number.
	VarComplexCot (see page 23535)	Returns the cotangent of a complex number.
	VarComplexCotH (see page 23536)	Returns the hyperbolic cotangent of a complex number.
	VarComplexCreate (see page 23537)	Returns a custom Variant that represents a complex number.
	VarComplexCreate (see page 23537)	Returns a custom Variant that represents a complex number.
	VarComplexCreate (see page 23537)	Returns a custom Variant that represents a complex number.
	VarComplexCreate (see page 23537)	Returns a custom Variant that represents a complex number.
	VarComplexCsc (see page 23538)	Returns the cosecant of a complex number.
	VarComplexCscH (see page 23539)	Returns the hyperbolic cosecant of a complex number.
	VarComplexExp (see page 23540)	Returns the exponential of a complex number.
	VarComplexFromPolar (see page 23541)	Converts a set of polar coordinates to a complex number.
	VarComplexInverse (see page 23541)	Returns the inverse of a complex number.
	VarComplexLn (see page 23542)	Returns the natural log of a complex number.
	VarComplexLog10 (see page 23543)	Returns the log base 10 of a complex number.
	VarComplexLog2 (see page 23543)	Returns the log base 2 of a complex number.
	VarComplexLogN (see page 23544)	Returns the log base N of a complex number.
	VarComplexPower (see page 23545)	Returns the value of a complex number raised to a specified power.
	VarComplexSec (see page 23545)	Returns the secant of a complex number.
	VarComplexSecH (see page 23546)	Returns the hyperbolic secant of a complex number.
	VarComplexSign (see page 23548)	This is function VarCmplx.VarComplexSign.
	VarComplexSimplify (see page 23548)	Reduces a Variant that represents a complex number to one that represents a real value, if possible.
	VarComplexSin (see page 23548)	Returns the sine of a complex number.
	VarComplexSinH (see page 23550)	Returns the hyperbolic sine of a complex number.

	VarComplexSqr (see page 23551)	Returns the square of a complex number.
	VarComplexSqrt (see page 23551)	Returns the square root of a complex number.
	VarComplexTan (see page 23552)	Returns the tangent of a complex number.
	VarComplexTanH (see page 23553)	Returns the hyperbolic tangent of a complex number.
	VarComplexTimesImaginary (see page 23554)	Returns the value of a complex number multiplied by an imaginary number.
	VarComplexTimesNegI (see page 23555)	Returns the value of a complex number multiplied by –i.
	VarComplexTimesPosI (see page 23555)	Returns the value of a complex number multiplied by i.
	VarComplexTimesReal (see page 23556)	Returns the value of a complex number multiplied by a real number.
	VarComplexToPolar (see page 23556)	Computes the polar coordinates that correspond to a custom Variant that represents a complex number.
	VarIsComplex (see page 23557)	Indicates whether a Variant stores its data internally as a complex number.

169.1.1 VarCmplx.VarAsComplex Function

Casts an arbitrary Variant so that it is a custom Variant representing a complex number.

Pascal

```
function VarAsComplex(const AValue: Variant): Variant;
```

C++

```
Variant VarAsComplex(const Variant AValue);
```

File

VarCmplx

Description

VarAsComplex converts the Variant specified by AValue to a custom Variant representing the corresponding complex number. It returns the custom Variant.

If VarAsComplex can't perform the conversion, it raises an EInvalidCast exception.

Complex Variants (see page 23567) are custom Variants (see page 23567) that represent complex numbers. The Variant type supports direct manipulation using the addition, subtraction, multiplication, division (but not integer division), and negation operators. They have 5 published properties: Real, Imaginary, Radius, Theta, and FixedTheta. They can be cast to and from integer types, floating point types, string types, TDateTime values, and boolean values. In addition, the VarCmplx (see page 23513) unit implements a number of global functions for operating on complex Variants (see page 23567).

See Also

EInvalidCast

VarIsComplex (see page 23557)

VarComplexCreate (see page 23537)

VarComplexSimplify (see page 23548)

VarAsType

169.1.2 VarCmplx.VarComplex Function

Returns the Variant type code for custom Variants (see page 23567) that represent complex numbers.

Pascal

```
function VarComplex: TVarType;
```

C++

```
TVarType VarComplex();
```

File

VarCmplx

Description

Use VarComplex to obtain the variant type code for custom Variants (see page 23567) that represent complex numbers. The Variant type codes for custom Variants (see page 23567) are dynamically allocated when the application loads the unit that implements them. VarComplex allows the application to identify the Variant type code that is currently used for custom Variants (see page 23567) that represent complex numbers. This type code, however, may not match the type code used by another application that uses a different set of custom Variant types.

Complex Variants (see page 23567) are custom Variants (see page 23567) that represent complex numbers. The Variant type supports direct manipulation using the addition, subtraction, multiplication, division (but not integer division), and negation operators. They have 5 published properties: Real, Imaginary, Radius, Theta, and FixedTheta. They can be cast to and from integer types, floating point types, string types, TDateTime values, and boolean values. In addition, the VarCmplx (see page 23513) unit implements a number of global functions for operating on complex Variants (see page 23567).

See Also

VarType

169.1.3 VarCmplx.VarComplexAbs Function

Returns the absolute value of a complex number.

Pascal

```
function VarComplexAbs(const AValue: Variant): Double;
```

C++

```
Double VarComplexAbs(const Variant AValue);
```

File

VarCmplx

Description

Use VarComplexAbs to take the absolute value (length) of a complex number.

AValue is a Variant whose value represents the complex number.

VarComplexAbs returns the absolute value of AValue.

See Also

VarComplexAbsSqr (see page 23518)

VarComplexAngle (see page 23518)

VarComplexConjugate (see page 23532)

VarComplexInverse (see page 23541)

VarComplexTimesPosI (see page 23555)

VarComplexTimesNegI (see page 23555)

169.1.4 VarCmplx.VarComplexAbsSqr Function

Returns the square of the absolute value of a complex number.

Pascal

```
function VarComplexAbsSqr(const AValue: Variant): Double;
```

C++

```
Double VarComplexAbsSqr(const Variant AValue);
```

File

VarCmplx

Description

Use VarComplexAbsSqr to obtain the square of a complex number's absolute value.

AValue is a Variant whose value represents the complex number.

VarComplexAbs (see page 23517) returns the absolute value of AValue, squared.

See Also

VarComplexAbs (see page 23517)

VarComplexAngle (see page 23518)

VarComplexConjugate (see page 23532)

VarComplexInverse (see page 23541)

VarComplexTimesPosI (see page 23555)

VarComplexTimesNegI (see page 23555)

VarComplexLn (see page 23542)

VarComplexExp (see page 23540)

VarComplexSqr (see page 23551)

VarComplexSqrt (see page 23551)

VarComplexPower (see page 23545)

169.1.5 VarCmplx.VarComplexAngle Function

Returns the angle made by a complex number.

Pascal

```
function VarComplexAngle(const AValue: Variant): Double;
```

C++

```
Double VarComplexAngle(const Variant AValue);
```

File

VarCmplx

Description

Use VarComplexAngle to obtain the angle of a complex number.

AValue is a Variant whose value represents the complex number.

VarComplexAbs (see page 23517) returns the angle that corresponds to AValue.

See Also

VarComplexToPolar (see page 23556)

VarComplexAbs (see page 23517)

169.1.6 VarCmplx.VarComplexArcCos Function

Returns the inverse cosine of a complex number.

Pascal

```
function VarComplexArcCos(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcCos(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcCos computes the inverse cosine of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexArcCos returns a Variant whose value is the inverse cosine of AValue.

See Also

VarComplexArcCosH (see page 23520)

VarComplexArcCot (see page 23521)

VarComplexArcCotH (see page 23522)

VarComplexArcCsc (see page 23523)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexCsc (see page 23538)

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.7 VarCmplx.VarComplexArcCosH Function

Returns the hyperbolic inverse cosine of a complex number.

Pascal

```
function VarComplexArcCosH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcCosH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcCosH computes the hyperbolic inverse cosine of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexArcCosH returns a Variant whose value is the hyperbolic inverse cosine of AValue.

See Also

VarComplexArcCos (see page 23519)

VarComplexArcCot (see page 23521)

VarComplexArcCotH (see page 23522)

VarComplexArcCsc (see page 23523)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexCsc (see page 23538)

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.8 VarCmplx.VarComplexArcCot Function

Returns the inverse cotangent of a complex number.

Pascal

```
function VarComplexArcCot(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcCot(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcCot computes the inverse cotangent of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexArcCot returns a Variant whose value is the inverse cotangent of AValue.

See Also

VarComplexArcCosH (see page 23520)

VarComplexArcCos (see page 23519)

VarComplexArcCotH (see page 23522)

VarComplexArcCsc (see page 23523)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexCsc (see page 23538)

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.9 VarCmplx.VarComplexArcCotH Function

Returns the hyperbolic inverse cotangent of a complex number.

Pascal

```
function VarComplexArcCotH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcCotH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcCot (see page 23521) computes the hyperbolic inverse cotangent of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexArcCot (see page 23521) returns a Variant whose value is the hyperbolic inverse cotangent of AValue.

See Also

VarComplexArcCosH (see page 23520)

VarComplexArcCot (see page 23521)

VarComplexArcCos (see page 23519)

VarComplexArcCsc (see page 23523)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexCsc (see page 23538)

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.10 VarCmplx.VarComplexArcCsc Function

Returns the inverse cosecant of a complex number.

Pascal

```
function VarComplexArcCsc(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcCsc(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcCsc computes the inverse cosecant of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexArcCsc returns a Variant whose value is the inverse cosecant of AValue.

See Also

VarComplexArcCosH (see page 23520)

VarComplexArcCos (see page 23519)

VarComplexArcCotH (see page 23522)

VarComplexArcCot (see page 23521)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexCsc (see page 23538)

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.11 VarCmplx.VarComplexArcCscH Function

Returns the hyperbolic inverse cosecant of a complex number.

Pascal

```
function VarComplexArcCscH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcCscH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcCsc (see page 23523) computes the hyperbolic inverse cosecant of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexArcCsc (see page 23523) returns a Variant whose value is the hyperbolic inverse cosecant of AValue.

See Also

VarComplexArcCosH (see page 23520)

VarComplexArcCsc (see page 23523)

VarComplexArcCos (see page 23519)

VarComplexArcCot (see page 23521)

VarComplexArcCotH (see page 23522)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexCsc (see page 23538)

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.12 VarCmplx.VarComplexArcSec Function

Returns the inverse secant of a complex number.

Pascal

```
function VarComplexArcSec(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcSec(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcSec computes the inverse secant of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexArcSec returns a Variant whose value is the inverse secant of AValue.

See Also

VarComplexArcCosH (see page 23520)

VarComplexArcCos (see page 23519)

VarComplexArcCotH (see page 23522)

VarComplexArcCot (see page 23521)

VarComplexArcCscH (see page 23524)

VarComplexArcCsc (see page 23523)

VarComplexArcSecH (see page 23527)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.13 VarCmplx.VarComplexArcSecH Function

Returns the hyperbolic inverse secant of a complex number.

Pascal

```
function VarComplexArcSecH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcSecH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcSec (see page 23525) computes the hyperbolic inverse secant of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexArcSec (see page 23525) returns a Variant whose value is the hyperbolic inverse secant of AValue.

See Also

VarComplexArcCosH (see page 23520)

VarComplexArcCos (see page 23519)

VarComplexArcCotH (see page 23522)

VarComplexArcCot (see page 23521)

VarComplexArcCsc (see page 23523)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.14 VarCmplx.VarComplexArcSin Function

Returns the inverse sine of a complex number.

Pascal

```
function VarComplexArcSin(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcSin(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcSin computes the inverse sine of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexArcSin returns a Variant whose value is the inverse sine of AValue.

See Also

VarComplexArcCosH (see page 23520)

VarComplexArcCot (see page 23521)

VarComplexArcCotH (see page 23522)

VarComplexArcCsc (see page 23523)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcCos (see page 23519)

VarComplexArcSinH (see page 23529)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexCsc (see page 23538)

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.15 VarCmplx.VarComplexArcSinH Function

Returns the hyperbolic inverse sine of a complex number.

Pascal

```
function VarComplexArcSinH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcSinH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcSinH computes the hyperbolic inverse sine of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexArcSinH returns a Variant whose value is the hyperbolic inverse sine of AValue.

See Also

VarComplexArcSinH

VarComplexArcCot (see page 23521)

VarComplexArcCotH (see page 23522)

VarComplexArcCsc (see page 23523)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcCos (see page 23519)

VarComplexArcCosH (see page 23520)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexCsc (see page 23538)

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.16 VarCmplx.VarComplexArcTan Function

Returns the inverse tangent of a complex number.

Pascal

```
function VarComplexArcTan(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcTan(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcTan computes the inverse tangent of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexArcTan returns a Variant whose value is the inverse tangent of AValue.

See Also

VarComplexArcCosH (see page 23520)

VarComplexArcCot (see page 23521)

VarComplexArcCotH (see page 23522)

VarComplexArcCsc (see page 23523)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcCos (see page 23519)

VarComplexArcTanH (see page 23531)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexCsc (see page 23538)

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.17 VarCmplx.VarComplexArcTanH Function

Returns the hyperbolic inverse tangent of a complex number.

Pascal

```
function VarComplexArcTanH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcTanH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcTanH computes the hyperbolic inverse tangent of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexArcTanH returns a Variant whose value is the hyperbolic inverse tangent of AValue.

See Also

VarComplexArcTan (see page 23530)

VarComplexArcCot (see page 23521)

VarComplexArcCotH (see page 23522)

VarComplexArcCsc (see page 23523)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcCos (see page 23519)

VarComplexArcCosH (see page 23520)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexCsc (see page 23538)

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.18 VarCmplx.VarComplexConjugate Function

Returns the conjugate of a complex number.

Pascal

```
function VarComplexConjugate(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexConjugate(const Variant AValue);
```

File

VarCmplx

Description

VarComplexConjugate returns the conjugate of a complex number. The conjugate is the value that has the same real part and the inverse of the imaginary part.

AValue is a Variant whose value represents the complex number.

VarComplexConjugate returns a Variant whose value is the conjugate of AValue.

See Also

VarComplexAbs (see page 23517)

VarComplexAngle (see page 23518)

VarComplexAbsSqr (see page 23518)

VarComplexInverse (see page 23541)

VarComplexTimesPosI (see page 23555)

VarComplexTimesNegI (see page 23555)

VarComplexLn (see page 23542)

VarComplexExp (see page 23540)

VarComplexSqr (see page 23551)

VarComplexSqrt (see page 23551)

VarComplexPower (see page 23545)

169.1.19 VarCmplx.VarComplexCos Function

Returns the cosine of a complex number.

Pascal

```
function VarComplexCos(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexCos(const Variant AValue);
```

File

VarCmplx

Description

VarComplexCos computes the cosine of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexCos returns a Variant whose value is the cosine of AValue.

See Also

VarComplexArcCosH (see page 23520)

VarComplexArcCot (see page 23521)

VarComplexArcCotH (see page 23522)

VarComplexArcCsc (see page 23523)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexArcCos (see page 23519)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexCsc (see page 23538)

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.20 VarCmplx.VarComplexCosH Function

Returns the hyperbolic cosine of a complex number.

Pascal

```
function VarComplexCosH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexCosH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexCosH computes the hyperbolic cosine of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexCosH returns a Variant whose value is the hyperbolic cosine of AValue.

See Also

VarComplexArcCos (see page 23519)

VarComplexArcCot (see page 23521)

VarComplexArcCotH (see page 23522)

VarComplexArcCsc (see page 23523)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexCsc (see page 23538)

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.21 VarCmplx.VarComplexCot Function

Returns the cotangent of a complex number.

Pascal

```
function VarComplexCot(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexCot(const Variant AValue);
```

File

VarCmplx

Description

VarComplexCot computes the cotangent of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexCot returns a Variant whose value is the cotangent of AValue.

See Also

VarComplexArcCosH (see page 23520)

VarComplexArcCos (see page 23519)

VarComplexArcCotH (see page 23522)

VarComplexArcCsc (see page 23523)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot

VarComplexCotH (see page 23536)

VarComplexCsc (see page 23538)

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.22 VarCmplx.VarComplexCotH Function

Returns the hyperbolic cotangent of a complex number.

Pascal

```
function VarComplexCotH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexCotH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexCot (see page 23535) computes the hyperbolic cotangent of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexCot (see page 23535) returns a Variant whose value is the hyperbolic cotangent of AValue.

See Also

VarComplexArcCosH (see page 23520)

VarComplexArcCot (see page 23521)

VarComplexArcCos (see page 23519)

VarComplexArcCsc (see page 23523)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH

VarComplexCsc (see page 23538)

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.23 VarCmplx.VarComplexCreate Function

Returns a custom Variant that represents a complex number.

Pascal

```
function VarComplexCreate: Variant; overload;
function VarComplexCreate(const AReal: Double): Variant; overload;
function VarComplexCreate(const AReal: Double; const AImaginary: Double): Variant; overload;
function VarComplexCreate(const AText: string): Variant; overload;
```

C++

```
Variant VarComplexCreate();
Variant VarComplexCreate(const Double AReal);
Variant VarComplexCreate(const Double AReal, const Double AImaginary);
Variant VarComplexCreate(const AnsiString AText);
```

File

VarCmplx

Description

Call VarComplexCreate to create a Variant whose value is a complex number.

AReal is the real part of the complex number.

AImaginary is the imaginary part of the complex number.

Using the first three versions of the constructor, omitting AReal or AImaginary results in a real or imaginary part of 0.

Using the last syntax, AText is parsed using the ComplexNumberSymbol (see page 23559) and ComplexNumberSymbolBeforeImaginary (see page 23560) global variables and, if the string represents a complex value, that becomes the value of the new Variant. If the string can't be parsed into a complex number, VarComplexCreate raises an EConvertError exception.

Complex Variants (see page 23567) are custom Variants (see page 23567) that represent complex numbers. The Variant type supports direct manipulation using the addition, subtraction, multiplication, division (but not integer division), and negation operators. They have 5 published properties: Real, Imaginary, Radius, Theta, and FixedTheta. They can be cast to and from integer types, floating point types, string types, TDateTime values, and boolean values. In addition, the VarCmplx (see page 23513) unit implements a number of global functions for operating on complex Variants (see page 23567).

See Also

VarIsComplex (see page 23557)

VarAsComplex (see page 23516)

VarComplexSimplify (see page 23548)

VarComplexFromPolar (see page 23541)

ComplexNumberSymbol (see page 23559)

ComplexNumberSymbolBeforeImaginary (see page 23560)

EConvertError

169.1.24 VarCmplx.VarComplexCsc Function

Returns the cosecant of a complex number.

Pascal

```
function VarComplexCsc(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexCsc(const Variant AValue);
```

File

VarCmplx

Description

VarComplexCsc computes the cosecant of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexCsc returns a Variant whose value is the cosecant of AValue.

See Also

VarComplexArcCosH (see page 23520)

VarComplexArcCos (see page 23519)

VarComplexArcCotH (see page 23522)

VarComplexArcCot (see page 23521)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexCsc

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.25 VarCmplx.VarComplexCscH Function

Returns the hyperbolic cosecant of a complex number.

Pascal

```
function VarComplexCscH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexCscH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexCsc (see page 23538) computes the hyperbolic cosecant of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexCsc (see page 23538) returns a Variant whose value is the hyperbolic cosecant of AValue.

See Also

VarComplexArcCosH (see page 23520)

VarComplexArcCsc (see page 23523)

VarComplexArcCos (see page 23519)

VarComplexArcCot (see page 23521)

VarComplexArcCotH (see page 23522)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexCsc (see page 23538)

VarComplexCscH

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.26 VarCmplx.VarComplexExp Function

Returns the exponential of a complex number.

Pascal

```
function VarComplexExp(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexExp(const Variant AValue);
```

File

VarCmplx

Description

VarComplexExp returns value of e raised to the power of a specified complex number, where e is the base of the natural logarithms.

AValue is a Variant whose value represents the complex number.

VarComplexExp returns a Variant whose value is the exponential of AValue.

See Also

VarComplexAbs (see page 23517)

VarComplexAngle (see page 23518)

VarComplexAbsSqr (see page 23518)

VarComplexInverse (see page 23541)

VarComplexTimesPosI (see page 23555)

VarComplexTimesNegI (see page 23555)

VarComplexLn (see page 23542)

VarComplexConjugate (see page 23532)

VarComplexSqr (see page 23551)

VarComplexSqrt (see page 23551)

VarComplexPower (see page 23545)

169.1.27 VarCmplx.VarComplexFromPolar Function

Converts a set of polar coordinates to a complex number.

Pascal

```
function VarComplexFromPolar(const ARadius: Double; const ATheta: Double): Variant;
```

C++

```
Variant VarComplexFromPolar(const Double ARadius, const Double ATheta);
```

File

VarCmplx

Description

VarComplexFromPolar returns the complex number that corresponds to the radius (length) and angle specified by the ARadius and ATheta parameters, respectively. The result is returned as a Variant whose value is the corresponding complex number.

See Also

VarComplexCreate (see page 23537)

VarComplexToPolar (see page 23556)

169.1.28 VarCmplx.VarComplexInverse Function

Returns the inverse of a complex number.

Pascal

```
function VarComplexInverse(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexInverse(const Variant AValue);
```

File

VarCmplx

Description

VarComplexInverse returns the inverse of a complex number. The inverse is the value that results when you divide 1 by the complex number.

AValue is a Variant whose value represents the complex number.

VarComplexInverse returns a Variant whose value is the inverse of AValue.

See Also

VarComplexAbs (see page 23517)

VarComplexAngle (see page 23518)

VarComplexAbsSqr (see page 23518)

VarComplexConjugate (see page 23532)

VarComplexTimesPosI (see page 23555)

VarComplexTimesNegI (see page 23555)

VarComplexLn (see page 23542)

VarComplexExp (see page 23540)

VarComplexSqr (see page 23551)

VarComplexSqrt (see page 23551)

VarComplexPower (see page 23545)

169.1.29 VarCmplx.VarComplexLn Function

Returns the natural log of a complex number.

Pascal

```
function VarComplexLn(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexLn(const Variant AValue);
```

File

VarCmplx

Description

VarComplexLn returns the natural logarithm of a complex number. The natural logarithm is the power to which you must raise e to get the complex number.

AValue is a Variant whose value represents the complex number.

VarComplexLn returns a Variant whose value is the natural log of AValue.

See Also

VarComplexAbs (see page 23517)

VarComplexAngle (see page 23518)

VarComplexAbsSqr (see page 23518)

VarComplexConjugate (see page 23532)

VarComplexTimesPosI (see page 23555)

VarComplexTimesNegI (see page 23555)

VarComplexInverse (see page 23541)

VarComplexExp (see page 23540)

VarComplexSqr (see page 23551)

VarComplexSqrt (see page 23551)

VarComplexPower (see page 23545)

169.1.30 VarCmplx.VarComplexLog10 Function

Returns the log base 10 of a complex number.

Pascal

```
function VarComplexLog10(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexLog10(const Variant AValue);
```

File

VarCmplx

Description

VarComplexLog10 returns the base 10 logarithm of a complex number. The base 10 logarithm is the power to which you must raise 10 to get the complex number.

AValue is a Variant whose value represents the complex number.

VarComplexLog10 returns a Variant whose value is the log base 10 of AValue.

See Also

VarComplexLn (see page 23542)

VarComplexLog2 (see page 23543)

VarComplexLogN (see page 23544)

169.1.31 VarCmplx.VarComplexLog2 Function

Returns the log base 2 of a complex number.

Pascal

```
function VarComplexLog2(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexLog2(const Variant AValue);
```

File

VarCmplx

Description

VarComplexLog2 returns the base 2 logarithm of a complex number. The base 2 logarithm is the power to which you must raise 2 to get the complex number.

AValue is a Variant whose value represents the complex number.

VarComplexLog2 returns a Variant whose value is the log base 2 of AValue.

See Also

VarComplexLn (see page 23542)

VarComplexLog10 (see page 23543)

VarComplexLogN (see page 23544)

169.1.32 VarCmplx.VarComplexLogN Function

Returns the log base N of a complex number.

Pascal

```
function VarComplexLogN(const AValue: Variant; const X: Double): Variant;
```

C++

```
Variant VarComplexLogN(const Variant AValue, const Double X);
```

File

VarCmplx

Description

VarComplexLogN returns the base N logarithm of a complex number. The base N logarithm is the power to which you must raise an exponent N to get the complex number.

AValue is a Variant whose value represents the complex number.

X is the logarithm exponent.

VarComplexLogN returns a Variant whose value is the log base N of AValue.

See Also

VarComplexLn (see page 23542)

VarComplexLog2 (see page 23543)

VarComplexLog10 (see page 23543)

169.1.33 VarCmplx.VarComplexPower Function

Returns the value of a complex number raised to a specified power.

Pascal

```
function VarComplexPower(const AValue: Variant; const APower: Variant): Variant;
```

C++

```
Variant VarComplexPower(const Variant AValue, const Variant APower);
```

File

VarCmplx

Description

VarComplexPower returns the result of raising AValue to the power APower.

AValue is a Variant whose value represents the complex number.

APower is a Variant that specifies the power to which you want to raise AValue. It can also be a complex number.

VarComplexPower returns a Variant whose value is AValue raised to the APower power.

See Also

VarComplexAbs (see page 23517)

VarComplexAngle (see page 23518)

VarComplexAbsSqr (see page 23518)

VarComplexConjugate (see page 23532)

VarComplexTimesPosI (see page 23555)

VarComplexTimesNegI (see page 23555)

VarComplexInverse (see page 23541)

VarComplexExp (see page 23540)

VarComplexSqr (see page 23551)

VarComplexSqrt (see page 23551)

VarComplexLn (see page 23542)

169.1.34 VarCmplx.VarComplexSec Function

Returns the secant of a complex number.

Pascal

```
function VarComplexSec(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSec(const Variant AValue);
```

File

VarCmplx

Description

VarComplexSec computes the secant of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexSec returns a Variant whose value is the secant of AValue.

See Also

VarComplexArcCosH (see page 23520)

VarComplexArcCos (see page 23519)

VarComplexArcCotH (see page 23522)

VarComplexArcCot (see page 23521)

VarComplexArcCscH (see page 23524)

VarComplexArcCsc (see page 23523)

VarComplexArcSecH (see page 23527)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexSec

VarComplexSecH (see page 23546)

VarComplexSec

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.35 VarCmplx.VarComplexSecH Function

Returns the hyperbolic secant of a complex number.

Pascal

```
function VarComplexSecH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSecH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexSec (see page 23545) computes the hyperbolic secant of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexSec (see page 23545) returns a Variant whose value is the hyperbolic secant of AValue.

See Also

VarComplexArcCosH (see page 23520)

VarComplexArcCos (see page 23519)

VarComplexArcCotH (see page 23522)

VarComplexArcCot (see page 23521)

VarComplexArcCsc (see page 23523)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexSec (see page 23545)

VarComplexSecH

VarComplexSec (see page 23545)

VarComplexSecH

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.36 VarCmplx.VarComplexSign Function

Pascal

```
function VarComplexSign(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSign(const Variant AValue);
```

File

VarCmplx

Description

This is function VarCmplx.VarComplexSign.

169.1.37 VarCmplx.VarComplexSimplify Function

Reduces a Variant that represents a complex number to one that represents a real value, if possible.

Pascal

```
function VarComplexSimplify(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSimplify(const Variant AValue);
```

File

VarCmplx

Description

Use VarComplexSimplify to convert a complex number to a real value when possible. VarComplexSimplify checks whether the Variant specified by AValue represents a complex number that has no imaginary part. If so, it returns a Variant whose value is simply a real number. If AValue does not represent a complex number, or if AValue represents a complex number with a nonzero imaginary part, VarComplexSimplify returns AValue.

See Also

VarIsComplex (see page 23557)

VarComplexCreate (see page 23537)

VarAsComplex (see page 23516)

VarAsType

169.1.38 VarCmplx.VarComplexSin Function

Returns the sine of a complex number.

Pascal

```
function VarComplexSin(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSin(const Variant AValue);
```

File

VarCmplx

Description

VarComplexSin computes the sine of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexSin returns a Variant whose value is the sine of AValue.

See Also

VarComplexArcCosH (see page 23520)

VarComplexArcCot (see page 23521)

VarComplexArcCotH (see page 23522)

VarComplexArcCsc (see page 23523)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcCos (see page 23519)

VarComplexArcSinH (see page 23529)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexCsc (see page 23538)

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.39 VarCmplx.VarComplexSinH Function

Returns the hyperbolic sine of a complex number.

Pascal

```
function VarComplexSinH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSinH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexSinH computes the hyperbolic sine of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexSinH returns a Variant whose value is the hyperbolic sine of AValue.

See Also

VarComplexArcSinH (see page 23529)

VarComplexArcCot (see page 23521)

VarComplexArcCotH (see page 23522)

VarComplexArcCsc (see page 23523)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcCos (see page 23519)

VarComplexArcCosH (see page 23520)

VarComplexArcTan (see page 23530)

VarComplexArcTanH (see page 23531)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexCsc (see page 23538)

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH

VarComplexTan (see page 23552)

VarComplexTanH (see page 23553)

169.1.40 VarCmplx.VarComplexSqr Function

Returns the square of a complex number.

Pascal

```
function VarComplexSqr(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSqr(const Variant AValue);
```

File

VarCmplx

Description

VarComplexSqr returns the square of AValue.

AValue is a Variant whose value represents the complex number.

See Also

VarComplexAbs (see page 23517)

VarComplexAngle (see page 23518)

VarComplexAbsSqr (see page 23518)

VarComplexConjugate (see page 23532)

VarComplexTimesPosI (see page 23555)

VarComplexTimesNegI (see page 23555)

VarComplexInverse (see page 23541)

VarComplexExp (see page 23540)

VarComplexLn (see page 23542)

VarComplexSqrt (see page 23551)

VarComplexPower (see page 23545)

169.1.41 VarCmplx.VarComplexSqrt Function

Returns the square root of a complex number.

Pascal

```
function VarComplexSqrt(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSqrt(const Variant AValue);
```

File

VarCmplx

Description

VarComplexSqrt returns the square root of AValue.

AValue is a Variant whose value represents the complex number.

See Also

VarComplexAbs (see page 23517)

VarComplexAngle (see page 23518)

VarComplexAbsSqrt

VarComplexConjugate (see page 23532)

VarComplexTimesPosI (see page 23555)

VarComplexTimesNegI (see page 23555)

VarComplexInverse (see page 23541)

VarComplexExp (see page 23540)

VarComplexLn (see page 23542)

VarComplexSqr (see page 23551)

VarComplexPower (see page 23545)

169.1.42 VarCmplx.VarComplexTan Function

Returns the tangent of a complex number.

Pascal

```
function VarComplexTan(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexTan(const Variant AValue);
```

File

VarCmplx

Description

VarComplexTan computes the tangent of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexTan returns a Variant whose value is the tangent of AValue.

See Also

VarComplexArcCosH (see page 23520)

VarComplexArcCot (see page 23521)

VarComplexArcCotH (see page 23522)

VarComplexArcCsc (see page 23523)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcCos (see page 23519)

VarComplexArcTanH (see page 23531)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexCsc (see page 23538)

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan

VarComplexTanH (see page 23553)

169.1.43 VarCmplx.VarComplexTanH Function

Returns the hyperbolic tangent of a complex number.

Pascal

```
function VarComplexTanH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexTanH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexTanH computes the hyperbolic tangent of a complex number and returns the result.

AValue is a Variant whose value represents the complex number.

VarComplexTanH returns a Variant whose value is the hyperbolic tangent of AValue.

See Also

VarComplexArcTan (see page 23530)

VarComplexArcCot (see page 23521)

VarComplexArcCotH (see page 23522)

VarComplexArcCsc (see page 23523)

VarComplexArcCscH (see page 23524)

VarComplexArcSec (see page 23525)

VarComplexArcSecH (see page 23527)

VarComplexArcCos (see page 23519)

VarComplexArcCosH (see page 23520)

VarComplexArcSin (see page 23528)

VarComplexArcSinH (see page 23529)

VarComplexCos (see page 23533)

VarComplexCosH (see page 23534)

VarComplexCot (see page 23535)

VarComplexCotH (see page 23536)

VarComplexCsc (see page 23538)

VarComplexCscH (see page 23539)

VarComplexSec (see page 23545)

VarComplexSecH (see page 23546)

VarComplexSin (see page 23548)

VarComplexSinH (see page 23550)

VarComplexTan (see page 23552)

VarComplexTanH

169.1.44 VarCmplx.VarComplexTimesImaginary Function

Returns the value of a complex number multiplied by an imaginary number.

Pascal

```
function VarComplexTimesImaginary(const AValue: Variant; const AFactor: Double): Variant;
```

C++

```
Variant VarComplexTimesImaginary(const Variant AValue, const Double AFactor);
```

File

VarCmplx

Description

VarComplexTimesImaginary returns a complex value times an imaginary number.

AValue is a Variant whose value represents the complex number.

AFactor is a real number that represents the imaginary number. That is, the imaginary number is AFactor times i.

VarComplexTimesImaginary returns a variant whose value is AValue times AFactor times i.

See Also

VarComplexTimesNegI (see page 23555)

VarComplexTimesPosI (see page 23555)

VarComplexTimesReal (see page 23556)

169.1.45 VarCmplx.VarComplexTimesNegI Function

Returns the value of a complex number multiplied by –i.

Pascal

```
function VarComplexTimesNegI(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexTimesNegI(const Variant AValue);
```

File

VarCmplx

Description

VarComplexTimesNegI returns AValue times –i.

AValue is a Variant whose value represents the complex number.

See Also

VarComplexTimesImaginary (see page 23554)

VarComplexTimesPosI (see page 23555)

VarComplexTimesReal (see page 23556)

169.1.46 VarCmplx.VarComplexTimesPosI Function

Returns the value of a complex number multiplied by i.

Pascal

```
function VarComplexTimesPosI(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexTimesPosI(const Variant AValue);
```

File

VarCmplx

Description

VarComplexTimesPosI returns AValue times i.

AValue is a Variant whose value represents the complex number.

See Also

VarComplexTimesNegI (see page 23555)

VarComplexTimesImaginary (see page 23554)

VarComplexTimesReal (see page 23556)

169.1.47 VarCmplx.VarComplexTimesReal Function

Returns the value of a complex number multiplied by a real number.

Pascal

```
function VarComplexTimesReal(const AValue: Variant; const AFactor: Double): Variant;
```

C++

```
Variant VarComplexTimesReal(const Variant AValue, const Double AFactor);
```

File

VarCmplx

Description

VarComplexTimesReal returns a complex value times a real number.

AValue is a Variant whose value represents the complex number.

AFactor is a real number.

VarComplexTimesReal returns a variant whose value is AValue times AFactor.

See Also

VarComplexTimesNegI (see page 23555)

VarComplexTimesPosI (see page 23555)

VarComplexTimesReal

169.1.48 VarCmplx.VarComplexToPolar Function

Computes the polar coordinates that correspond to a custom Variant that represents a complex number.

Pascal

```
procedure VarComplexToPolar(const AValue: Variant; var ARadius: Double; var ATheta: Double;
AFixTheta: Boolean = True);
```

C++

```
VarComplexToPolar(const Variant AValue, Double ARadius, Double ATheta, Boolean AFixTheta =
True);
```

File

VarCmplx

Description

Call VarComplexToPolar to obtain the polar coordinates for a complex number.

AValue is a Variant whose value represents the complex number.

ARadius returns the radius of the corresponding polar coordinates.

ATheta returns the angle (Theta) of the corresponding polar coordinates.

AFixTheta indicates whether VarComplexToPolar should normalize the angle (ATheta). When AFixTheta is true, VarComplexToPolar always returns an angle between – pi and pi.

See Also

VarComplexAngle (see page 23518)

VarComplexFromPolar (see page 23541)

VarComplexAbs (see page 23517)

169.1.49 VarCmplx.VarIsComplex Function

Indicates whether a Variant stores its data internally as a complex number.

Pascal

```
function VarIsComplex(const AValue: Variant): Boolean;
```

C++

```
Boolean VarIsComplex(const Variant AValue);
```

File

VarCmplx

Description

VarIsComplex returns true if AValue represents a complex number (that is, if it has a VarType of VarComplex (see page 23516)).

Complex Variants (see page 23567) are custom Variants (see page 23567) that represent complex numbers. The Variant type supports direct manipulation using the addition, subtraction, multiplication, division (but not integer division), and negation operators. They have 5 published properties: Real, Imaginary, Radius, Theta, and FixedTheta. They can be cast to and from integer types, floating point types, string types, TDateTime values, and boolean values. In addition, the VarCmplx (see page 23513) unit implements a number of global functions for operating on complex Variants (see page 23567).

See Also

VarAsComplex (see page 23516)

VarComplexCreate (see page 23537)

VarComplexSimplify (see page 23548)

VarComplex (see page 23516)

VarType

169.2 Variables

The following table lists variables in this documentation.

Variables

Name	Description
ComplexNumberDefuzzAtZero (see page 23559)	This is variable VarCmplx.ComplexNumberDefuzzAtZero.
ComplexNumberSymbol (see page 23559)	This is variable VarCmplx.ComplexNumberSymbol.
ComplexNumberSymbolBeforeImaginary (see page 23560)	This is variable VarCmplx.ComplexNumberSymbolBeforeImaginary.

169.2.1 VarCmplx.ComplexNumberDefuzzAtZero Variable

Pascal

```
ComplexNumberDefuzzAtZero: Boolean = True;
```

C++

```
Boolean ComplexNumberDefuzzAtZero = True;
```

File

VarCmplx

Description

This is variable VarCmplx.ComplexNumberDefuzzAtZero.

169.2.2 VarCmplx.ComplexNumberSymbol Variable

Pascal

```
ComplexNumberSymbol: string = 'i';
```

C++

```
AnsiString ComplexNumberSymbol = 'i';
```

File

VarCmplx

Description

This is variable VarCmplx.ComplexNumberSymbol.

169.2.3 VarCmplx.ComplexNumberSymbolBeforeImaginary Variable

Pascal

```
ComplexNumberSymbolBeforeImaginary: Boolean = False;
```

C++

```
Boolean ComplexNumberSymbolBeforeImaginary = False;
```

File

VarCmplx

Description

This is variable VarCmplx.ComplexNumberSymbolBeforeImaginary.

170 VarConv Namespace

This is namespace VarConv.

Topics

Name	Description
Functions (see page 23562)	The following table lists functions in this documentation.

Functions

	Name	Description
	VarAsConvert (see page 23562)	Casts an arbitrary Variant so that it is a custom Variant representing a measurement.
	VarAsConvert (see page 23562)	Casts an arbitrary Variant so that it is a custom Variant representing a measurement.
	VarConvert (see page 23563)	Returns the Variant type code for custom Variants (see page 23567) that represent measurements.
	VarConvertCreate (see page 23564)	Returns a custom Variant that represents a measurement.
	VarConvertCreate (see page 23564)	Returns a custom Variant that represents a measurement.
	VarIsConvert (see page 23565)	Indicates whether a Variant is a custom Variant that represents a measurement.

170.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	VarAsConvert (see page 23562)	Casts an arbitrary Variant so that it is a custom Variant representing a measurement.
	VarAsConvert (see page 23562)	Casts an arbitrary Variant so that it is a custom Variant representing a measurement.
	VarConvert (see page 23563)	Returns the Variant type code for custom Variants (see page 23567) that represent measurements.
	VarConvertCreate (see page 23564)	Returns a custom Variant that represents a measurement.
	VarConvertCreate (see page 23564)	Returns a custom Variant that represents a measurement.
	VarIsConvert (see page 23565)	Indicates whether a Variant is a custom Variant that represents a measurement.

170.1.1 VarConv.VarAsConvert Function

Casts an arbitrary Variant so that it is a custom Variant representing a measurement.

Pascal

```
function VarAsConvert(const AValue: Variant): Variant; overload;
function VarAsConvert(const AValue: Variant; const AType: TConvType): Variant; overload;
```

C++

```
Variant VarAsConvert(const Variant AValue);
Variant VarAsConvert(const Variant AValue, const TConvType AType);
```

File

VarConv

Description

VarAsConvert converts the Variant specified by AValue to a VarConvert (see page 23563) custom Variant.

Using the first syntax, AValue is interpreted as the numeric portion (the number of units), and AType identifies the units.

Using the second syntax, AValue should be a Variant that can be parsed into a VarConv (see page 23561) custom Variant (for example, a variant whose value is a string such as '1.5 miles').

If VarAsConvert can't perform the conversion, it raises an EInvalidCast exception.

VarConvert (see page 23563) custom Variants (see page 23567) represent measurements (that is, a type of measurement units and the number of those units). The units must be registered measurement units such as those used by the ConvUtils (see page 6379) unit. You can use the negation operator on VarConv (see page 23561) variants, and use the addition, subtraction, multiplication, division, and comparison operators on a VarConv (see page 23561) Variant with a real number. You can use the addition, subtraction, division, and comparison operators on two VarConv (see page 23561) Variants (see page 23567). VarConv (see page 23561) custom Variants (see page 23567) also support the following properties:

Value is the numeric portion (the number of units). (read/write)

Type is units. (read-only)

TypeName is the name of the units. (read-only).

Family is the conversion family of the units (the thing measured). (read-only).

FamilyName is the name of the conversion family. (read-only)

Depending on the Family, each Variant has a set of conversion properties with names of the form As<Unit>, where <Unit> is the name of another type of unit in the same conversion family.

See Also

EInvalidCast

VarIsConvert (see page 23565)

VarConvertCreate (see page 23564)

VarAsType

TVarData

170.1.2 VarConv.VarConvert Function

Returns the Variant type code for custom Variants (see page 23567) that represent measurements.

Pascal

```
function VarConvert: TVarType;
```

C++

```
TVarType VarConvert();
```

File

VarConv

Description

Use VarConvert to obtain the variant type code for custom Variants (see page 23567) that represent measurements. The Variant type codes for custom Variants (see page 23567) are dynamically allocated when the application loads the unit that implements them. VarConvert allows the application to identify the Variant type code that is currently used for custom Variants (see page 23567) that represent VarConvert custom Variants (see page 23567). This type code, however, may not match the type code used by another application that uses a different set of custom Variant types.

VarConvert custom Variants (see page 23567) represent measurements (that is, a type of measurement units and the number of those units). The units must be registered measurement units such as those used by the ConvUtils (see page 6379) unit. You can use the negation operator on VarConv (see page 23561) variants, and use the addition, subtraction, multiplication, division, and comparison operators on a VarConv (see page 23561) Variant with a real number. You can use the addition, subtraction, division, and comparison operators on two VarConv (see page 23561) Variants (see page 23567). VarConv (see page 23561) custom Variants (see page 23567) also support the following properties:

Value is the numeric portion (the number of units). (read/write)

Type is units. (read-only)

TypeName is the name of the units. (read-only).

Family is the conversion family of the units (the thing measured). (read-only).

FamilyName is the name of the conversion family. (read-only)

Depending on the Family, each Variant has a set of conversion properties with names of the form As<Unit>, where <Unit> is the name of another type of unit in the same conversion family.

See Also

VarType

170.1.3 VarConv.VarConvertCreate Function

Returns a custom Variant that represents a measurement.

Pascal

```
function VarConvertCreate(const AValue: Double; const AType: TConvType): Variant; overload;
function VarConvertCreate(const AValue: string): Variant; overload;
```

C++

```
Variant VarConvertCreate(const Double AValue, const TConvType AType);
Variant VarConvertCreate(const AnsiString AValue);
```

File

VarConv

Description

Call VarConvertCreate to create a Variant whose value is a measurement.

Using the first syntax, AType indicates the measurement units, and AValue is the number of those units. AType must represent a registered conversion unit.

Using the second syntax, AValue is a string that can be parsed by the TryStrToConvUnit function. (For example '1.5 liters'.)

If the string can't be parsed into a measurement, VarConvertCreate raises an EInvalidCast exception.

VarConvert (see page 23563) custom Variants (see page 23567) represent measurements (that is, a type of measurement units and the number of those units). The units must be registered measurement units such as those used by the ConvUtils (see page 6379) unit. You can use the negation operator on VarConv (see page 23561) variants, and use the addition, subtraction, multiplication, division, and comparison operators on a VarConv (see page 23561) Variant with a real number. You can use the addition, subtraction, division, and comparison operators on two VarConv (see page 23561) Variants (see page 23567). VarConv (see page 23561) custom Variants (see page 23567) also support the following properties:

Value is the numeric portion (the number of units). (read/write)

Type is units. (read-only)

TypeName is the name of the units. (read-only).

Family is the conversion family of the units (the thing measured). (read-only).

FamilyName is the name of the conversion family. (read-only)

Depending on the Family, each Variant has a set of conversion properties with names of the form As<Unit>, where <Unit> is the name of another type of unit in the same conversion family.

See Also

VarIsConvert (see page 23565)

VarAsConvert (see page 23562)

EInvalidCast

170.1.4 VarConv.VarIsConvert Function

Indicates whether a Variant is a custom Variant that represents a measurement.

Pascal

```
function VarIsConvert(const AValue: Variant): Boolean;
```

C++

```
Boolean VarIsConvert(const Variant AValue);
```

File

VarConv

Description

VarIsConvert returns true if AValue represents a measurement (that is, if it has a VarType of VarConvert (see page 23563)).

VarConvert (see page 23563) custom Variants (see page 23567) represent measurements (that is, a type of measurement units and the number of those units). The units must be registered measurement units such as those used by the ConvUtils (see page 6379) unit. You can use the negation operator on VarConv (see page 23561) variants, and use the addition, subtraction, multiplication, division, and comparison operators on a VarConv (see page 23561) Variant with a real number. You can use the addition, subtraction, division, and comparison operators on two VarConv (see page 23561) Variants (see page 23567). VarConv (see page 23561) custom Variants (see page 23567) also support the following properties:

Value is the numeric portion (the number of units). (read/write)

Type is units. (read-only)

TypeName is the name of the units. (read-only).

Family is the conversion family of the units (the thing measured). (read-only).

FamilyName is the name of the conversion family. (read-only)

Depending on the Family, each Variant has a set of conversion properties with names of the form As<Unit>, where <Unit> is the name of another type of unit in the same conversion family.

See Also

VarAsConvert (see page 23562)

VarConvertCreate (see page 23564)

VarConvert (see page 23563)

VarType

171 Variants Namespace

This is namespace Variants.

Topics

Name	Description
Classes (see page 23572)	The following table lists classes in this documentation.
Interfaces (see page 23614)	The following table lists interfaces in this documentation.
Functions (see page 23617)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 23671)	The following table lists structs, records, enums in this documentation.
Types (see page 23674)	The following table lists types in this documentation.
Variables (see page 23676)	The following table lists variables in this documentation.

Classes

	Name	Description
	EVariantArrayCreateError (see page 23572)	EVariantArrayCreateError is the exception class for failure to create or resize a variant array.
	EVariantArrayLockedError (see page 23573)	EVariantArrayLockedError is the exception class for errors that occur because a variant array is locked or fixed.
	EVariantBadIndexError (see page 23574)	EVariantBadIndexError is the exception class for errors that occur because a variant array index is out of bounds.
	EVariantBadVarTypeError (see page 23574)	EVariantBadVarTypeError is the exception class for variant type errors.
	EVariantDispatchError (see page 23575)	EVariantDispatchError is the exception class for failed dispatch attempts.
	EVariantInvalidArgError (see page 23575)	EVariantInvalidArgError is the exception class for operations that fail due to invalid variant data.
	EVariantInvalidNullOpError (see page 23576)	EVariantInvalidNullOpError is the exception class for invalid Null (see page 23641) variant operations.
	EVariantInvalidOpError (see page 23577)	EVariantInvalidOpError is the exception class for invalid variant operations.
	EVariantNotAnArrayError (see page 23577)	EVariantNotAnArrayError is the exception class inappropriate variant array operations.
	EVariantNotImplError (see page 23578)	EVariantNotImplError is the exception class for unimplemented variant operations.
	EVariantOutOfMemoryError (see page 23578)	EVariantOutOfMemoryError is the exception class for variant operations that cannot allocate necessary memory.
	EVariantOverflowError (see page 23579)	EVariantOverflowError is the exception class for overflow in a variant operation.
	EVariantRangeCheckError (see page 23580)	EVariantRangeCheckError is the exception class for out-of-range errors in a variant operation.
	EVariantTypeCastError (see page 23580)	EVariantTypeCastError is the exception class for variant type cast errors.
	EVariantUnexpectedError (see page 23581)	EVariantUnexpectedError is the exception class for unexpected variant errors.

	TCustomVariantType (see page 23581)	TCustomVariantType is the base class for Delphi custom variant types.
	TInvokeableVariantType (see page 23606)	TInvokeableVariantType is the base class for custom Variant types that implement custom Variants with properties and methods.

Functions

	Name	Description
	_DispInvoke (see page 23619)	This is function Variants._DispInvoke.
	_OleVarFromInt (see page 23620)	This is function Variants._OleVarFromInt.
	_OleVarFromInt64 (see page 23620)	This is function Variants._OleVarFromInt64.
	_OleVarFromLStr (see page 23620)	This is function Variants._OleVarFromLStr.
	_OleVarFromPStr (see page 23621)	This is function Variants._OleVarFromPStr.
	_OleVarFromVar (see page 23621)	This is function Variants._OleVarFromVar.
	_VarAdd (see page 23621)	This is function Variants._VarAdd.
	_VarAddRef (see page 23622)	This is function Variants._VarAddRef.
	_VarAnd (see page 23622)	This is function Variants._VarAnd.
	_VarArrayGet (see page 23622)	This is function Variants._VarArrayGet.
	_VarArrayPut (see page 23623)	This is function Variants._VarArrayPut.
	_VarArrayRedim (see page 23623)	This is function Variants._VarArrayRedim.
	_VarCast (see page 23623)	This is function Variants._VarCast.
	_VarCastOle (see page 23624)	This is function Variants._VarCastOle.
	_VarClear (see page 23624)	This is function Variants._VarClear.
	_VarClr (see page 23624)	This is function Variants._VarClr.
	_VarCmpEQ (see page 23625)	This is function Variants._VarCmpEQ.
	_VarCmpGE (see page 23625)	This is function Variants._VarCmpGE.
	_VarCmpGT (see page 23625)	This is function Variants._VarCmpGT.
	_VarCmpLE (see page 23626)	This is function Variants._VarCmpLE.
	_VarCmpLT (see page 23626)	This is function Variants._VarCmpLT.
	_VarCmpNE (see page 23626)	This is function Variants._VarCmpNE.
	_VarCopy (see page 23627)	This is function Variants._VarCopy.
	_VarCopyNoInd (see page 23627)	This is function Variants._VarCopyNoInd.
	_VarDiv (see page 23627)	This is function Variants._VarDiv.
	_VarFromBool (see page 23628)	This is function Variants._VarFromBool.
	_VarFromCurr (see page 23628)	This is function Variants._VarFromCurr.
	_VarFromDisp (see page 23628)	This is function Variants._VarFromDisp.
	_VarFromDynArray (see page 23629)	This is function Variants._VarFromDynArray.
	_VarFromInt (see page 23629)	This is function Variants._VarFromInt.
	_VarFromInt64 (see page 23629)	This is function Variants._VarFromInt64.
	_VarFromIntf (see page 23630)	This is function Variants._VarFromIntf.
	_VarFromLStr (see page 23630)	This is function Variants._VarFromLStr.
	_VarFromPStr (see page 23630)	This is function Variants._VarFromPStr.
	_VarFromReal (see page 23631)	This is function Variants._VarFromReal.
	_VarFromTDateTime (see page 23631)	This is function Variants._VarFromTDateTime.
	_VarFromWStr (see page 23631)	This is function Variants._VarFromWStr.
	_VarMod (see page 23632)	This is function Variants._VarMod.
	_VarMul (see page 23632)	This is function Variants._VarMul.
	_VarNeg (see page 23632)	This is function Variants._VarNeg.
	_VarNot (see page 23633)	This is function Variants._VarNot.
	_VarOr (see page 23633)	This is function Variants._VarOr.
	_VarRDiv (see page 23633)	This is function Variants._VarRDiv.
	_VarShl (see page 23634)	This is function Variants._VarShl.
	_VarShr (see page 23634)	This is function Variants._VarShr.
	_VarSub (see page 23634)	This is function Variants._VarSub.
	_VarToBool (see page 23635)	This is function Variants._VarToBool.
	_VarToCurrency (see page 23635)	This is function Variants._VarToCurrency.
	_VarToDisp (see page 23635)	This is function Variants._VarToDisp.

	_VarToDynArray (see page 23636)	This is function Variants._VarToDynArray.
	_VarToInt64 (see page 23636)	This is function Variants._VarToInt64.
	_VarToInteger (see page 23636)	This is function Variants._VarToInteger.
	_VarToIntf (see page 23637)	This is function Variants._VarToIntf.
	_VarToLStr (see page 23637)	This is function Variants._VarToLStr.
	_VarToPStr (see page 23637)	This is function Variants._VarToPStr.
	_VarToReal (see page 23638)	This is function Variants._VarToReal.
	_VarToWStr (see page 23638)	This is function Variants._VarToWStr.
	_VarXor (see page 23638)	This is function Variants._VarXor.
	DynArrayFromVariant (see page 23639)	Creates a dynamic array from a Variant.
	DynArrayToVariant (see page 23639)	Creates a Variant array from a dynamic array.
	FindCustomVariantType (see page 23640)	Retrieves the object that implements a custom Variant type.
	FindCustomVariantType (see page 23640)	Retrieves the object that implements a custom Variant type.
	FindVarData (see page 23640)	Returns pointer to variant's data.
	HandleConversionException (see page 23641)	Handles exceptions during variant type conversions.
	Null (see page 23641)	Returns a Null variant.
	Unassigned (see page 23643)	Returns an "empty" variant.
	VarArrayAsPSafeArray (see page 23643)	This is function Variants.VarArrayAsPSafeArray.
	VarArrayCreate (see page 23644)	Creates a variant array.
	VarArrayCreateError (see page 23645)	Indicates an array creation error.
	VarArrayDimCount (see page 23645)	Returns number of dimensions of a variant array.
	VarArrayGet (see page 23645)	Returns a Variant that represents a single value from a multi-dimensional Variant array.
	VarArrayHighBound (see page 23646)	Returns high bound for a dimension in a variant array.
	VarArrayLock (see page 23646)	Locks a variant array and returns a pointer to the data.
	VarArrayLowBound (see page 23647)	Returns the low bound of a dimension in a variant array.
	VarArrayOf (see page 23648)	Creates and fills a one-dimensional variant array.
	VarArrayPut (see page 23648)	Sets the value of a single cell in a multi-dimensional Variant array.
	VarArrayRef (see page 23649)	Returns a reference to the specified variant array.
	VarArrayUnlock (see page 23649)	Unlocks a variant array.
	VarAsError (see page 23650)	Converts a HRESULT into an Error variant.
	VarAsType (see page 23650)	Converts a variant to specified type.
	VarCastError (see page 23651)	Raises an EVariantTypeCastError (see page 23580) exception.
	VarCastError (see page 23651)	Raises an EVariantTypeCastError (see page 23580) exception.
	VarCheckEmpty (see page 23651)	Raises an exception if a specified variant's value is Unassigned (see page 23643).
	VarCompareValue (see page 23652)	Returns the relationship between Variants.
	VarCopyNoInd (see page 23652)	Copies a Variant.
	VarEnsureRange (see page 23653)	Returns a Variant with a value in a specified range.
	VarFromDateTime (see page 23653)	Returns a variant containing a specified data and time.
	VarInRange (see page 23654)	Indicates whether a Variant's value is in a specified range.
	VarInvalidNullOp (see page 23654)	Indicate variant operation on null.
	VarInvalidOp (see page 23655)	Indicate invalid variant operation.
	VarIsArray (see page 23655)	Indicates whether the specified variant is an array.
	VarIsArray (see page 23655)	Indicates whether the specified variant is an array.
	VarIsByRef (see page 23656)	Indicates whether the specified variant's value is by reference.
	VarIsClear (see page 23656)	Indicates whether the specified variant has an undefined value.
	VarIsCustom (see page 23657)	Indicates whether the specified variant is a custom variant.
	VarIsEmpty (see page 23657)	Indicates whether the specified variant is unassigned.
	VarIsEmptyParam (see page 23658)	Indicates whether the specified variant represents an unassigned optional parameter.
	VarIsError (see page 23659)	Indicates whether the specified variant is an error variant; optionally converts variant back to HRESULT.
	VarIsError (see page 23659)	Indicates whether the specified variant is an error variant; optionally converts variant back to HRESULT.
	VarIsFloat (see page 23659)	Indicates whether the specified variant represents a floating-point value.

	VarIsNull (see page 23660)	Indicates whether the specified variant is Null (see page 23641).
	VarIsNumeric (see page 23660)	Indicates whether the specified variant represents a numeric value.
	VarIsOrdinal (see page 23661)	Indicates whether the specified variant represents an ordinal value.
	VarIsStr (see page 23661)	Indicates whether the specified variant represents a string value.
	VarIsType (see page 23662)	Indicates whether the specified variant represents a specified type.
	VarIsType (see page 23662)	Indicates whether the specified variant represents a specified type.
	VarOverflowError (see page 23662)	Raises an EVariantOverflowError (see page 23579) exception.
	VarRangeCheckError (see page 23663)	Raises an EVariantRangeCheckError (see page 23580) exception.
	VarResultCheck (see page 23663)	Raises variant exception based on HRESULT value.
	VarResultCheck (see page 23663)	Raises variant exception based on HRESULT value.
	VarSameValue (see page 23664)	Indicates whether two Variants have equivalent values.
	VarSupports (see page 23664)	Indicates whether the Variant's value is an interface that supports a specified interface.
	VarSupports (see page 23664)	Indicates whether the Variant's value is an interface that supports a specified interface.
	VarToDateTime (see page 23665)	Converts a specified variant to a TDateTime value.
	VarToStr (see page 23665)	Converts a variant's value to a string.
	VarToStrDef (see page 23666)	Converts a variant's value to a string.
	VarToWideStr (see page 23666)	Converts a variant's value to a WideString.
	VarToWideStrDef (see page 23667)	Converts a variant's value to a WideString.
	VarType (see page 23667)	Returns the type code of a specified variant.
	VarTypeAsText (see page 23668)	Returns variant type descriptive string.
	VarTypeIsValidArrayType (see page 23669)	Indicates whether a Variant type code represents a type that can be used for the elements in a Variant array.
	VarTypeIsValidElementType (see page 23669)	Indicates whether a Variant type code represents a type that can be used for the elements in a Variant array.

Interfaces

	Name	Description
	IVarInstanceReference (see page 23614)	This is class Variants.IVarInstanceReference.
	IVarInvokeable (see page 23615)	This is class Variants.IVarInvokeable.

Structs, Records, Enums

	Name	Description
	TBooleanToStringRule (see page 23671)	TBooleanToStringRule enumerates possible rules for BooleanToStringRule (see page 23676).
	TNullCompareRule (see page 23671)	TNullCompareRule enumerates possible rules for comparing variant nulls with other values.
	TVarCompareResult (see page 23672)	This is record Variants.TVarCompareResult.
	TVariantRelationship (see page 23672)	TVariantRelationship describes the results of comparing two Variants.

Types

Name	Description
TAnyProc (see page 23674)	This is type Variants.TAnyProc.
TCustomVariantTypeClass (see page 23674)	This is type Variants.TCustomVariantTypeClass.
TVarDataArray (see page 23675)	This is type Variants.TVarDataArray.
TVarDispProc (see page 23675)	This is type Variants.TVarDispProc.

Variables

Name	Description
BooleanToStringRule (see page 23676)	Specifies rule fixing up Boolean values converted to strings.
ChangeAnyProc (see page 23677)	This is variable Variants.ChangeAnyProc.
ClearAnyProc (see page 23677)	This is variable Variants.ClearAnyProc.
EmptyParam (see page 23677)	Contains an OleVariant that represents an unused optional parameter on a dual interface.
NullAsStringValue (see page 23678)	Specifies string value of Null (see page 23641) variants.
NullEqualityRule (see page 23678)	Specifies rule for Null (see page 23641) variant equality comparison.

NullMagnitudeRule (see page 23679)	Specifies rule for Null (see page 23641) variant magnitude comparison.
NullStrictConvert (see page 23679)	Specifies whether Null (see page 23641) conversion is an error.
OleVariantInt64AsDouble (see page 23680)	This is variable Variants.OleVariantInt64AsDouble.
PackVarCreation (see page 23680)	Specifies whether space-efficient Variant Types (see page 23293) are preferred.
RefAnyProc (see page 23681)	This is variable Variants.RefAnyProc.
VarDispProc (see page 23681)	Provides the support for IDispatch-based Automation.

171.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EVariantArrayCreateError (see page 23572)	EVariantArrayCreateError is the exception class for failure to create or resize a variant array.
	EVariantArrayLockedError (see page 23573)	EVariantArrayLockedError is the exception class for errors that occur because a variant array is locked or fixed.
	EVariantBadIndexError (see page 23574)	EVariantBadIndexError is the exception class for errors that occur because a variant array index is out of bounds.
	EVariantBadVarTypeError (see page 23574)	EVariantBadVarTypeError is the exception class for variant type errors.
	EVariantDispatchError (see page 23575)	EVariantDispatchError is the exception class for failed dispatch attempts.
	EVariantInvalidArgError (see page 23575)	EVariantInvalidArgError is the exception class for operations that fail due to invalid variant data.
	EVariantInvalidNullOpError (see page 23576)	EVariantInvalidNullOpError is the exception class for invalid Null (see page 23641) variant operations.
	EVariantInvalidOpError (see page 23577)	EVariantInvalidOpError is the exception class for invalid variant operations.
	EVariantNotAnArrayError (see page 23577)	EVariantNotAnArrayError is the exception class inappropriate variant array operations.
	EVariantNotImplError (see page 23578)	EVariantNotImplError is the exception class for unimplemented variant operations.
	EVariantOutOfMemoryError (see page 23578)	EVariantOutOfMemoryError is the exception class for variant operations that cannot allocate necessary memory.
	EVariantOverflowError (see page 23579)	EVariantOverflowError is the exception class for overflow in a variant operation.
	EVariantRangeCheckError (see page 23580)	EVariantRangeCheckError is the exception class for out-of-range errors in a variant operation.
	EVariantTypeCastError (see page 23580)	EVariantTypeCastError is the exception class for variant type cast errors.
	EVariantUnexpectedError (see page 23581)	EVariantUnexpectedError is the exception class for unexpected variant errors.
	TCustomVariantType (see page 23581)	TCustomVariantType is the base class for Delphi custom variant types.
	TInvokeableVariantType (see page 23606)	TInvokeableVariantType is the base class for custom Variant types that implement custom Variants (see page 23567) with properties and methods.

171.1.1 EVariantArrayCreateError Class

EVariantArrayCreateError is the exception class for failure to create or resize a variant array.

Class Hierarchy

Pascal

```
EVariantArrayCreateError = class(EVariantError);
```

C++

```
class EVariantArrayCreateError : public EVariantError;
```

File

Variants

Description

EVariantArrayCreateError is raised when an attempt to create or resize a variant array fails. To raise this exception programmatically, call VarArrayCreateError (see page 23645)

See Also

VarArrayCreateError (see page 23645)

TVarData (see page 22571)

VarArrayCreate (see page 23644)

VarArrayOf (see page 23648)

VarArrayRedim

TCustomVariantType (see page 23581)

VarArrayCreateError (see page 23645)

171.1.2 EVariantArrayLockedError Class

EVariantArrayLockedError is the exception class for errors that occur because a variant array is locked or fixed.

Class Hierarchy

Pascal

```
EVariantArrayLockedError = class(EVariantError);
```

C++

```
class EVariantArrayLockedError : public EVariantError;
```

File

Variants

Description

EVariantArrayLockedError is raised when an operation on a variant array fails because the array is locked or fixed.

See Also

TVarData (see page 22571)

TVariantManager

VarAsType (see page 23650)

VarArrayLock (see page 23646)

VarArrayUnlock (see page 23649)

VarArrayOf (see page 23648)

VarArrayRedim

171.1.3 EVariantBadIndexError Class

EVariantBadIndexError is the exception class for errors that occur because a variant array index is out of bounds.

Class Hierarchy

Pascal

```
EVariantBadIndexError = class(EVariantError);
```

C++

```
class EVariantBadIndexError : public EVariantError;
```

File

Variants

Description

EVariantBadIndexError is raised when an operation on a variant array fails because an index value is out of bounds.

See Also

TVarData (see page 22571)

TVariantManager

VarAsType (see page 23650)

VarArrayCreate (see page 23644)

VarArrayHighBound (see page 23646)

VarArrayLowBound (see page 23647)

VarArrayOf (see page 23648)

VarArrayRedim

TCustomVariantType (see page 23581)

171.1.4 EVariantBadVarTypeError Class

EVariantBadVarTypeError is the exception class for variant type errors.

Class Hierarchy

Pascal

```
EVariantBadVarTypeError = class(EVariantError);
```

C++

```
class EVariantBadVarTypeError : public EVariantError;
```

File

Variants

Description

EVariantBadVarTypeError is raised when a variant operation fails because of a value that isn't a valid variant type.

See Also

TVarData (see page 22571)

TVariantManager

VarAsType (see page 23650)

TCustomVariantType (see page 23581)

171.1.5 EVariantDispatchError Class

EVariantDispatchError is the exception class for failed dispatch attempts.

Class Hierarchy

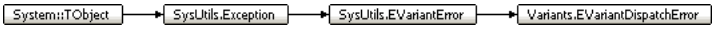

Pascal

```
EVariantDispatchError = class(EVariantError);
```

C++

```
class EVariantDispatchError : public EVariantError;
```

File

Variants

Description

EVariantDispatchError is raised when a variant type is unable to dispatch a method call. Do not directly instantiate EVariantDispatchError. Instead call the RaiseDispError method of TCustomVariant.

See Also

TVarData (see page 22571)

TVariantManager

TCustomVariantType (see page 23581)

TCustomVariantType (see page 23581)

171.1.6 EVariantInvalidArgError Class

EVariantInvalidArgError is the exception class for operations that fail due to invalid variant data.

Class Hierarchy

Pascal

```
EVariantInvalidArgError = class(EVariantError);
```

C++

```
class EVariantInvalidArgError : public EVariantError;
```

File

Variants

Description

EVariantInvalidArgError is raised when a variant operation fails because it was passed an inappropriate value.

See Also

TVarData (see page 22571)

TVariantManager

VarAsType (see page 23650)

TCustomVariantType (see page 23581)

171.1.7 EVariantInvalidNullOpError Class

EVariantInvalidNullOpError is the exception class for invalid Null (see page 23641) variant operations.

Class Hierarchy

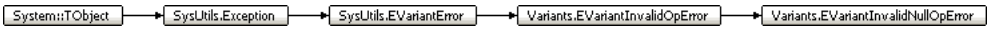

Pascal

```
EVariantInvalidNullOpError = class(EVariantInvalidOpError);
```

C++

```
class EVariantInvalidNullOpError : public EVariantInvalidOpError;
```

File

Variants

Description

EVariantInvalidNullOpError is raised when a variant operation fails because the operation is not valid for Null (see page 23641) variants. To raise this exception programmatically, call VarInvalidNullOp (see page 23654).

Note: Do not raise EVariantInvalidNullOpError unless NullEqualityRule (see page 23678) or NullMagnitudeRule (see page 23679) is ncrError, and a corresponding operation has been attempted.

See Also

TVarData (see page 22571)

TVariantManager

TCustomVariantType (see page 23581)

VarInvalidNullOp (see page 23654)

Null (see page 23641)

NullEqualityRule (see page 23678)

NullMagnitudeRule (see page 23679)

171.1.8 EVariantInvalidOpError Class

EVariantInvalidOpError is the exception class for invalid variant operations.

Class Hierarchy

Pascal

```
EVariantInvalidOpError = class(EVariantError);
```

C++

```
class EVariantInvalidOpError : public EVariantError;
```

File

Variants

Description

EVariantInvalidOpError is raised when a variant operation fails because the operands are incompatible with the specific operation. To raise this exception programmatically, call VarInvalidOp (see page 23655).

See Also

TVarData (see page 22571)

TVariantManager

TCustomVariantType (see page 23581)

VarInvalidOp (see page 23655)

171.1.9 EVariantNotAnArrayError Class

EVariantNotAnArrayError is the exception class inappropriate variant array operations.

Class Hierarchy

Pascal

```
EVariantNotAnArrayError = class(EVariantError);
```

C++

```
class EVariantNotAnArrayError : public EVariantError;
```

File

Variants

Description

EVariantNotAnArrayError is raised when an array operation is attempted on a variant that does not contain an array.

See Also

VarArrayCreateError (see page 23645)

TVarData (see page 22571)

VarArrayCreate (see page 23644)

VarArrayOf (see page 23648)

VarArrayRedim

TCustomVariantType (see page 23581)

VarArrayCreateError (see page 23645)

171.1.10 EVariantNotImplError Class

EVariantNotImplError is the exception class for unimplemented variant operations.

Class Hierarchy

Pascal

```
EVariantNotImplError = class(EVariantError);
```

C++

```
class EVariantNotImplError : public EVariantError;
```

File

Variants

Description

EVariantNotImplError is raised when an operation is attempted on a variant type that does not implement that operation.

See Also

TVarData (see page 22571)

TVariantManager

TCustomVariantType (see page 23581)

171.1.11 EVariantOutOfMemoryError Class

EVariantOutOfMemoryError is the exception class for variant operations that cannot allocate necessary memory.

Class Hierarchy

Pascal

```
EVariantOutOfMemoryError = class(EVariantError);
```

C++

```
class EVariantOutOfMemoryError : public EVariantError;
```

File

Variants

Description

EVariantOutOfMemoryError is raised when a variant operation fails because of a failure to allocate necessary memory.

See Also

TVarData (see page 22571)

TVariantManager

VarArrayCreate (see page 23644)

VarArrayOf (see page 23648)

VarArrayRedim

VarToDateTime (see page 23665)

TCustomVariantType (see page 23581)

EOutOfMemory (see page 22742)

171.1.12 EVariantOverflowError Class

EVariantOverflowError is the exception class for overflow in a variant operation.

Class Hierarchy

Pascal

```
EVariantOverflowError = class(EVariantError);
```

C++

```
class EVariantOverflowError : public EVariantError;
```

File

Variants

Description

EVariantOverflowError is raised when a variant operation fails because the result was too large. To raise this exception programmatically, call VarOverflowError (see page 23662).

See Also

TVarData (see page 22571)

TVariantManager

VarAsType (see page 23650)

VarToDateTime (see page 23665)

TCustomVariantType (see page 23581)

VarOverflowError (see page 23662)

171.1.13 EVariantRangeCheckError Class

EVariantRangeCheckError is the exception class for out-of-range errors in a variant operation.

Class Hierarchy

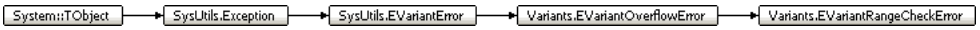

Pascal

```
EVariantRangeCheckError = class(EVariantOverflowError);
```

C++

```
class EVariantRangeCheckError : public EVariantOverflowError;
```

File

Variants

Description

EVariantRangeCheckError is raised when a variant operation fails because the result is out-of-range for the destination variant type. To raise this exception programmatically, call VarRangeCheckError (see page 23663).

See Also

TVarData (see page 22571)

TVariantManager

VarAsType (see page 23650)

VarToDateTime (see page 23665)

TCustomVariantType (see page 23581)

VarRangeCheckError (see page 23663)

171.1.14 EVariantTypeCastError Class

EVariantTypeCastError is the exception class for variant type cast errors.

Class Hierarchy

Pascal

```
EVariantTypeCastError = class(EVariantError);
```

C++

```
class EVariantTypeCastError : public EVariantError;
```

File

Variants

Description

EVariantTypeCastError is raised when a variant operation fails because a value could not be cast to the destination variant type. To raise this exception programmatically, call VarCastError (see page 23651).

See Also

TVarData (see page 22571)

TVariantManager

VarAsType (see page 23650)

VarToDateTime (see page 23665)

TCustomVariantType (see page 23581)

VarOverflowError (see page 23662)

171.1.15 EVariantUnexpectedError Class

EVariantUnexpectedError is the exception class for unexpected variant errors.

Class Hierarchy

Pascal

```
EVariantUnexpectedError = class(EVariantError);
```

C++

```
class EVariantUnexpectedError : public EVariantError;
```

File

Variants

Description

EVariantUnexpectedError is raised when a variant operation generates an unexpected error.

See Also

TVarData (see page 22571)

TVariantManager

TCustomVariantType (see page 23581)

171.1.16 TCustomVariantType Class

TCustomVariantType is the base class for Delphi custom variant types.

Class Hierarchy

Pascal

```
TCustomVariantType = class(TObject, IInterface);
```

C++

```
class TCustomVariantType : public TObject, public IInterface;
```

File

Variants

Description

Use TCustomVariantType to define new data types for Delphi Variants (see page 23567). TCustomVariantType descendants do not represent the Variants (see page 23567) themselves. Rather, they act as utility classes that the custom Variants (see page 23567) call on to perform their operations.

To create a Variant type,

1.Map the storage of the Variant's data on to the TVarData record.

2.Write a method that creates instances of your custom Variant. This fills in the Variant's data as defined in step 1.

3.Declare a class that descends from TCustomVariantType.

4.Implement all required behavior (including type conversion rules) in the new class. This involves overriding the appropriate virtual methods (typically BinaryOp (see page 23587), UnaryOp (see page 23598), CompareOp (see page 23590) or Compare (see page 23590), Cast (see page 23588), CastTo (see page 23589), Clear (see page 23589), Copy (see page 23591), IsClear (see page 23592), LeftPromotion (see page 23593), and RightPromotion (see page 23596)).

5.In the initialization section of the unit that defines your TCustomVariantType descendant, create an instance of your class. When you instantiate your object, it automatically registers itself with the Variant-handling system so that the new Variant type is enabled.

6.In the finalization section of the unit that defines your TCustomVariantType descendant, free the instance of your class. This automatically unregisters the Variant type.

Note: C++ can use Custom Variant types, but TCustomVariantType descendants must be implemented in Delphi.

For an example, see the FMTBcd (see page 13627) unit, which implements a custom Variant type for binary-coded decimal.

Tip: To create a custom Variant that allows users to call methods or procedures and to get or set property values, use TInvokeableVariantType (see page 23606) as a base class instead.

See Also

IDH

TInvokeableVariantType (see page 23606)

Creating a Class to Enable the Custom Variant Type

171.1.16.1 TCustomVariantType Members

The following tables list the members exposed by TCustomVariantType.

Protected Methods

	Name	Description
	_AddRef (see page 23586)	Increments the reference count for this interface.
	_Release (see page 23587)	Decrements the reference count for this interface.
	DispInvoke (see page 23592)	Invokes a method call when the Variant supports such operations.

	LeftPromotion (see page 23593)	Indicates whether the left argument to a binary operation should be coerced to a different type.
	OlePromotion (see page 23594)	Indicates the type to which a Variant value must be cast to make it Automation-compatible.
	QueryInterface (see page 23594)	Returns a reference to a specified interface if the object supports that interface.
	RaiseCastError (see page 23595)	Generates a system error that indicates an invalid cast.
	RaiseDispError (see page 23595)	Generates a system error indicating an invalid dispatch attempt.
	RaiseInvalidOp (see page 23596)	Generates a system error indicating an invalid operation.
	RightPromotion (see page 23596)	Indicates whether the right argument to a binary operation should be coerced to a different type.
	SimplisticClear (see page 23597)	Clears a TVarData record by re-initializing it.
	SimplisticCopy (see page 23598)	Performs a simple block copy of one TVarData record to another.
	VarDataCast (see page 23599)	Performs a cast from one Variant type to another.
	VarDataCastTo (see page 23599)	This is the overview for the VarDataCastTo method overload.
	VarDataCastToOleStr (see page 23600)	Performs a cast to a Variant type of varOleStr.
	VarDataClear (see page 23600)	Clears the data from a built-in Variant type.
	VarDataCopy (see page 23601)	Copies the data from one built-in Variant type to another.
	VarDataCopyNoInd (see page 23601)	Copies the data from one built-in Variant type to another.
	VarDataFromOleStr (see page 23602)	Creates a TVarData record to represent a Unicode string.
	VarDataFromStr (see page 23602)	Creates a TVarData record to represent a string.
	VarDataInit (see page 23602)	Initializes a TVarData record.
	VarDataIsArray (see page 23603)	Indicates whether a TVarData record represents the data from a Variant array.
	VarDataIsByRef (see page 23603)	Indicates whether a TVarData record stores its data indirectly.
	VarDataIsEmptyParam (see page 23603)	Indicates whether a TVarData record represents an unassigned optional parameter.
	VarDataIsFloat (see page 23604)	Indicates whether a TVarData record represents a floating-point value.
	VarDataIsNumeric (see page 23604)	Indicates whether a TVarData record represents a numeric value.
	VarDataIsOrdinal (see page 23605)	Indicates whether a TVarData record represents a ordinal value.
	VarDataIsStr (see page 23605)	Indicates whether a TVarData record represents a string (normal or Unicode).
	VarDataToStr (see page 23605)	Returns a string that represents the value from a TVarData record.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BinaryOp (see page 23587)	Implements any binary operations of which the custom Variant type is capable.
	Cast (see page 23588)	Converts the data from another Variant type so that it represents this custom type's data format.
	CastTo (see page 23589)	This is CastTo, a member of class TCustomVariantType.
	CastToOle (see page 23589)	This is CastToOle, a member of class TCustomVariantType.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 23589)	Sets the data for this custom Variant type to a blank or unassigned value.
	Compare (see page 23590)	Implements comparisons that involve the custom Variant type.
	CompareOp (see page 23590)	Implements any comparison operations of which the custom Variant type is capable.
	Copy (see page 23591)	Copies the data from a Variant of this custom type.
	Create (see page 23584)	This is the overview for the Create constructor overload.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 23585)	Frees an instance of TCustomVariantType (see page 23581).

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsClear (see page 23592)	Indicates whether the data from a custom Variant represents a blank or unassigned value.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	UnaryOp (see page 23598)	Implements any unary operations of which the custom Variant type is capable.

Public Properties

	Name	Description
	VarType (see page 23606)	Indicates the Variant type code for the custom Variant type.

171.1.16.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TCustomVariantType.Create () (see page 23584)	Creates an instance of TCustomVariantType (see page 23581).
	TCustomVariantType.Create (TVarType) (see page 23584)	Creates an instance of TCustomVariantType (see page 23581).

171.1.16.2.1 TCustomVariantType.Create Constructor ()

Creates an instance of TCustomVariantType (see page 23581).

Pascal

```
constructor Create; overload;
constructor Create(RequestedVarType: TVarType); overload;
```

C++

```
__fastcall TCustomVariantType();
__fastcall TCustomVariantType(TVarType RequestedVarType);
```

Description

Call Create in the initialization section of the unit that defines your TCustomVariantType (see page 23581) descendant to instantiate a single instance of the custom Variant type. This single instance automatically registers itself with the Variant system so that the custom Variant type is enabled in the application.

The RequestedVarType parameter specifies a type code for the new custom Variant type. It must be in the range from $010F to $0FFF.

When called with no parameters, the constructor generates a new type code for the custom Variant type.

The TCustomVariantType (see page 23581) constructor raises an exception if there is a problem with the new Variant type code. Such problems include

The RequestedVarType parameter specifies a type code that is already used by another Variant type.

The RequestedVarType parameter specifies a type code that is not in the valid range for custom Variants (see page 23567).

There are too many custom Variants (see page 23567) already defined to allow another one.

See Also

VarType (see page 23606)

171.1.16.3 TCustomVariantType.Destroy Destructor

Frees an instance of TCustomVariantType (see page 23581).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomVariantType();
```

Description

To free a TCustomVariantType (see page 23581) instance, use the Free method instead. Free checks that the instance is not already nil before calling Destroy.

Free the single instance of a TCustomVariantType (see page 23581) descendant from the finalization section of the unit that implements the class.

See Also

Create (see page 23584)

Free (see page 22300)

171.1.16.4 TCustomVariantType Methods

The methods of the TCustomVariantType class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 23586)	Increments the reference count for this interface.
	_Release (see page 23587)	Decrements the reference count for this interface.
V	DispInvoke (see page 23592)	Invokes a method call when the Variant supports such operations.
V	LeftPromotion (see page 23593)	Indicates whether the left argument to a binary operation should be coerced to a different type.
V	OlePromotion (see page 23594)	Indicates the type to which a Variant value must be cast to make it Automation-compatible.
V	QueryInterface (see page 23594)	Returns a reference to a specified interface if the object supports that interface.
	RaiseCastError (see page 23595)	Generates a system error that indicates an invalid cast.
	RaiseDispError (see page 23595)	Generates a system error indicating an invalid dispatch attempt.

	RaiseInvalidOp (see page 23596)	Generates a system error indicating an invalid operation.
V	RightPromotion (see page 23596)	Indicates whether the right argument to a binary operation should be coerced to a different type.
	SimplisticClear (see page 23597)	Clears a TVarData record by re-initializing it.
	SimplisticCopy (see page 23598)	Performs a simple block copy of one TVarData record to another.
	VarDataCast (see page 23599)	Performs a cast from one Variant type to another.
	VarDataCastTo (see page 23599)	This is the overview for the VarDataCastTo method overload.
	VarDataCastToOleStr (see page 23600)	Performs a cast to a Variant type of varOleStr.
	VarDataClear (see page 23600)	Clears the data from a built-in Variant type.
	VarDataCopy (see page 23601)	Copies the data from one built-in Variant type to another.
	VarDataCopyNoInd (see page 23601)	Copies the data from one built-in Variant type to another.
	VarDataFromOleStr (see page 23602)	Creates a TVarData record to represent a Unicode string.
	VarDataFromStr (see page 23602)	Creates a TVarData record to represent a string.
	VarDataInit (see page 23602)	Initializes a TVarData record.
	VarDataIsArray (see page 23603)	Indicates whether a TVarData record represents the data from a Variant array.
	VarDataIsByRef (see page 23603)	Indicates whether a TVarData record stores its data indirectly.
	VarDataIsEmptyParam (see page 23603)	Indicates whether a TVarData record represents an unassigned optional parameter.
	VarDataIsFloat (see page 23604)	Indicates whether a TVarData record represents a floating-point value.
	VarDataIsNumeric (see page 23604)	Indicates whether a TVarData record represents a numeric value.
	VarDataIsOrdinal (see page 23605)	Indicates whether a TVarData record represents a ordinal value.
	VarDataIsStr (see page 23605)	Indicates whether a TVarData record represents a string (normal or Unicode).
	VarDataToStr (see page 23605)	Returns a string that represents the value from a TVarData record.

Public Methods

	Name	Description
V	BinaryOp (see page 23587)	Implements any binary operations of which the custom Variant type is capable.
V	Cast (see page 23588)	Converts the data from another Variant type so that it represents this custom type's data format.
V	CastTo (see page 23589)	This is CastTo, a member of class TCustomVariantType.
V	CastToOle (see page 23589)	This is CastToOle, a member of class TCustomVariantType.
A	Clear (see page 23589)	Sets the data for this custom Variant type to a blank or unassigned value.
V	Compare (see page 23590)	Implements comparisons that involve the custom Variant type.
V	CompareOp (see page 23590)	Implements any comparison operations of which the custom Variant type is capable.
A	Copy (see page 23591)	Copies the data from a Variant of this custom type.
V	IsClear (see page 23592)	Indicates whether the data from a custom Variant represents a blank or unassigned value.
V	UnaryOp (see page 23598)	Implements any unary operations of which the custom Variant type is capable.

171.1.16.4.1 TCustomVariantType._AddRef Method

Increments the reference count for this interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

The Delphi compiler automatically generates a call to __AddRef for every reference to this interface. For most implementations, __AddRef increments the reference count for the interface and returns the new reference count. When the caller is through with the interface, the compiler generates a call to the _Release method, which decrements the reference count. When the reference count drops to zero, the object is automatically freed.

Note: Implementation objects can implement __AddRef and _Release so that they do not perform any reference counting. For such objects, the reference count never drops to zero and the object is not automatically freed. In these cases, it is the application's responsibility to free the object.

See Also

_Release (see page 22397)

171.1.16.4.2 TCustomVariantType._Release Method

Decrements the reference count for this interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

The Delphi compiler automatically generates a call to __Release whenever a reference to this interface goes out of scope. For most implementations, __Release decrements the reference count for the interface and returns the new reference count. When the reference count drops to zero, the object is automatically freed.

Note: Implementation objects can implement __Release so that it does not perform any reference counting. For such objects, the reference count never drops to zero and the object is not automatically freed. In these cases, it is the application's responsibility to free the object.

See Also

_AddRef (see page 22397)

171.1.16.4.3 TCustomVariantType.BinaryOp Method

Implements any binary operations of which the custom Variant type is capable.

Pascal

```
procedure BinaryOp(var Left: TVarData; const Right: TVarData; const Operator: TVarOp); virtual;
```

C++

```
virtual __fastcall BinaryOp(TVarData Left, const TVarData Right, const TVarOp Operator);
```

Description

Override BinaryOp to implement any binary operators the custom Variant type supports.

Left is the value that appears to the left of the operator. BinaryOp changes this value to indicate the result of the operation.

Right is the value that appears to the right of the operator.

Operator identifies the operator that appears between Left and Right. It can be any of the following values:

Value	Operation
opAdd	addition
opSubtract	subtraction
opMultiply	multiplication
opDivide	floating-point division

opIntDivide	integer division
opModulus	remainder
opShiftLeft	bitwise left shift
opShiftRight	bitwise right shift
opAnd	bitwise and
opOr	bitwise or
opXor	bitwise exclusive or

As implemented in TCustomVariantType (see page 23581), BinaryOp raises a system error indicating an invalid operation.

See Also

UnaryOp (see page 23598)

CompareOp (see page 23590)

LeftPromotion (see page 23593)

RightPromotion (see page 23596)

Implementing Binary Operations

171.1.16.4.4 TCustomVariantType.Cast Method

Converts the data from another Variant type so that it represents this custom type's data format.

Pascal

```
procedure Cast(var Dest: TVarData; const Source: TVarData); virtual;
```

C++

```
virtual __fastcall Cast(TVarData Dest, const TVarData Source);
```

Description

Override Cast to enable other types of Variants (see page 23567) to be cast to this custom Variant type.

Dest is a TVarData record that returns the Variant's data stored in the format used by this custom Variant type. The Cast method fills out this record from the information in Source.

Source is a TVarData record that contains the data from another Variant type. Check the VType field of Source to identify how it currently stores its data. Then use the other fields to fetch the data and convert it to the format used by this Variant type.

As implemented in TCustomVariantType (see page 23581), Cast checks whether Source is another custom Variant type. If so, it calls its CastTo (see page 23589) method, to convert it to this Variant type. If not, it raises a system error indicating an invalid cast.

See Also

castto (see page 23589)

VarDataCast (see page 23599)

Enabling Casting

171.1.16.4.5 TCustomVariantType.CastTo Method

Pascal

```
procedure CastTo(var Dest: TVarData; const Source: TVarData; const AVarType: TVarType);
virtual;
```

C++

```
virtual __fastcall CastTo(TVarData Dest, const TVarData Source, const TVarType AVarType);
```

Description

This is CastTo, a member of class TCustomVariantType.

171.1.16.4.6 TCustomVariantType.CastToOle Method

Pascal

```
procedure CastToOle(var Dest: TVarData; const Source: TVarData); virtual;
```

C++

```
virtual __fastcall CastToOle(TVarData Dest, const TVarData Source);
```

Description

This is CastToOle, a member of class TCustomVariantType.

171.1.16.4.7 TCustomVariantType.Clear Method

Sets the data for this custom Variant type to a blank or unassigned value.

Pascal

```
procedure Clear(var V: TVarData); virtual; abstract;
```

C++

```
virtual __fastcall Clear(TVarData V) = 0;
```

Description

Override Clear to implement the way this custom Variant type represents a blank or unassigned value.

V is a TVarData record that represents the data from a Variant of this custom type that needs to be cleared.

In TCustomVariantType (see page 23581), Clear is an abstract method. Descendant classes must override this method to provide an implementation. If the custom Variant type stores its data in a manner that includes memory that needs to be freed, Clear should free that memory. Once any memory is freed, Clear can call the SimplisticClear (see page 23597) method, which re-initializes the TVarData record, or set the VType field to varEmpty.

See Also

Unassigned (see page 23643)

IsClear (see page 23592)

SimplisticClear (see page 23597)

VarDataClear (see page 23600)

Copying and Clearing Custom Variants

171.1.16.4.8 TCustomVariantType.Compare Method

Implements comparisons that involve the custom Variant type.

Pascal

```
procedure Compare(const Left: TVarData; const Right: TVarData; var Relationship:
TVarCompareResult); virtual;
```

C++

```
virtual __fastcall Compare(const TVarData Left, const TVarData Right, TVarCompareResult
Relationship);
```

Description

Override Compare to implement any comparison operators the custom Variant type supports. Descendants can implement comparison operators by overriding either the Compare or the CompareOp (see page 23590) method.

Left is the value that appears to the left of the operator.

Right is the value that appears to the right of the operator.

Relationship returns a value indicating the relationship between Left and Right, as indicated in the following table:

Value	Operation
crLessThan	Left is less than Right.
crEqual	Left is equal to Right.
crGreaterThan	Left is greater than Right.

As implemented in TCustomVariantType (see page 23581), CompareOp (see page 23590) raises a system error indicating an invalid operation.

Tip: For custom Variants (see page 23567) that support a test of inequality, but not ordering, it makes more sense to override the CompareOp (see page 23590) method instead.

See Also

CompareOp (see page 23590)

CompareValue (see page 15938)

Implementing Comparison Operations

171.1.16.4.9 TCustomVariantType.CompareOp Method

Implements any comparison operations of which the custom Variant type is capable.

Pascal

```
function CompareOp(const Left: TVarData; const Right: TVarData; const Operator: TVarOp):
Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CompareOp(const TVarData Left, const TVarData Right, const TVarOp
Operator);
```

Description

Override CompareOp to implement any comparison operators the custom Variant type supports. Descendants can implement comparison operators by overriding either the Compare (see page 23590) or the CompareOp method.

Left is the value that appears to the left of the operator.

Right is the value that appears to the right of the operator.

Operator identifies the operator that appears between Left and Right. It can be any of the following values:

Value	Operation
opCmpEQ	equality comparison
opCmpNE	inequality comparison
opCmpLT	less than
opCmpLE	less than or equal
opCmpGT	greater than
opCmpGE	greater than or equal

CompareOp returns true if the comparison is true as specified. It returns false if the specified comparison does not hold true.

As implemented in TCustomVariantType (see page 23581), CompareOp raises a system error indicating an invalid operation.

See Also

UnaryOp (see page 23598)

BinaryOp (see page 23587)

LeftPromotion (see page 23593)

RightPromotion (see page 23596)

Compare (see page 23590)

Implementing Comparison Operations

171.1.16.4.10 TCustomVariantType.Copy Method

Copies the data from a Variant of this custom type.

Pascal

```
procedure Copy(var Dest: TVarData; const Source: TVarData; const Indirect: Boolean); virtual;
abstract;
```

C++

```
virtual __fastcall Copy(TVarData Dest, const TVarData Source, const Boolean Indirect) = 0;
```

Description

Override Copy to enable the custom Variant type to be copied (assigned) to another Variant.

Dest is the TVarData record from the Variant whose value is to be a copy of the source Variant.

Source is the TVarData record from a Variant of this custom type that is to be copied to Dest.

Indirect indicates whether the copy is direct or indirect. When Indirect is true, the source TVarData record contains an indirect reference to its data (the VarDataIsByRef (see page 23603) method should return true for Source). When making an indirect copy, it is not necessary to reallocate records or objects, because the Variant does not "own" the data, it merely points to it. When making a direct copy, Copy must allocate new memory for any allocated records or objects, so that Dest can "own" its own copy of the data.

In TCustomVariantType (see page 23581), Copy is an abstract method. Descendant classes must override this method to provide an implementation.

See Also

VarDataCopy (see page 23601)

VarDataCopyNoInd (see page 23601)

SimplisticCopy (see page 23598)

VarDataIsByRef (see page 23603)

Copying and Clearing Custom Variants

171.1.16.4.11 TCustomVariantType.DispInvoke Method

Invokes a method call when the Variant supports such operations.

Pascal

```
procedure DispInvoke(Dest: PVarData; const Source: TVarData; CallDesc: PCallDesc; Params:
Pointer); virtual;
```

C++

```
virtual __fastcall DispInvoke(PVarData Dest, const TVarData Source, PCallDesc CallDesc, void *
Params);
```

Description

When the application tries to call a method, set a property, or read a property using a Variant of the custom Variant type, the system automatically generates a call to DispInvoke. TCustomVariantType (see page 23581) responds by generating a system error indicating an invalid dispatch attempt.

To create a custom Variant type that allows applications to call methods or use properties, use TInvokeableVariantType (see page 23606) as a base class instead of TCustomVariantType (see page 23581). TInvokeableVariantType (see page 23606) introduces methods for executing method calls and implementing properties, and overrides the DispInvoke method so that it dispatches method calls to the appropriate methods.

Dest is a TVarData record that receives the return value of any function calls.

Source is the TVarData record that holds the data from the custom Variant whose property or method was invoked.

CallDesc identifies the method that was called, including the type and number of its arguments.

Params points to an array of parameter values that were passed in the method call.

See Also

TInvokeableVariantType (see page 23606)

171.1.16.4.12 TCustomVariantType.IsClear Method

Indicates whether the data from a custom Variant represents a blank or unassigned value.

Pascal

```
function IsClear(const V: TVarData): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsClear(const TVarData V);
```

Description

Override IsClear to identify when data from a Variant of the custom Variant type represents a blank or unassigned value. The global VarIsClear (see page 23656) method calls IsClear when given a custom Variant of this type.

As Implemented in TCustomVariantType (see page 23581), IsClear always returns false. Override this method to return true for invalid values or special values set by the Clear (see page 23589) method.

See Also

Clear (see page 23589)

VarIsClear (see page 23656)

Copying and Clearing Custom Variants

171.1.16.4.13 TCustomVariantType.LeftPromotion Method

Indicates whether the left argument to a binary operation should be coerced to a different type.

Pascal

```
function LeftPromotion(const V: TVarData; const Operator: TVarOp; out RequiredVarType:
TVarType): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean LeftPromotion(const TVarData V, const TVarOp Operator, TVarType
RequiredVarType);
```

Description

Override LeftPromotion to indicate when the left argument for a binary or comparison operation should be implicitly cast to another type before performing the operation. LeftPromotion is called when the right-hand Variant in a binary or comparison operation is of this Variant type and

The left-hand argument is a built-in Variant type.

The left-hand argument is a custom Variant type that does not indicate any problem with this Variant type for the right-hand argument.

V is the TVarData record for the left-hand argument of the operation.

Operator indicates the type of operation. It can be any of the operators in the following table:

Value	Operation
opAdd	addition
opSubtract	subtraction
opMultiply	multiplication
opDivide	floating-point division
opIntDivide	integer division
opModulus	remainder
opShiftLeft	left shift
opShiftRight	right shift
opAnd	bitwise and
opOr	bitwise or
opXor	bitwise exclusive or
opCompare	any comparison operation

RequiredVarType returns the Variant type code for the type to which the left-hand argument should be cast before performing the operation. If the operation can proceed with the left-hand argument left as-is, RequiredVarType returns the VType field of V.

LeftPromotion returns true if the TCustomVariantType (see page 23581) descendant can perform the operation (using the BinaryOp (see page 23587), CompareOp (see page 23590), or Compare (see page 23590) method) assuming the left-hand argument can be cast to RequiredVarType. It returns false if the TCustomVariantType (see page 23581) descendant can't perform the operation.

As implemented in TCustomVariantType (see page 23581), LeftPromotion returns the VarType (see page 23606) property as RequiredVarType, and returns true.

See Also

BinaryOp (see page 23587)

CompareOp (see page 23590)

Compare (see page 23590)

VarType (see page 23606)

RightPromotion (see page 23596)

Implementing Binary Operations

171.1.16.4.14 TCustomVariantType.OlePromotion Method

Indicates the type to which a Variant value must be cast to make it Automation-compatible.

Pascal

```
function OlePromotion(const V: TVarData; out RequiredVarType: TVarType): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean OlePromotion(const TVarData V, TVarType RequiredVarType);
```

Description

Override OlePromotion to indicate when a Variant of this custom Variant type must be cast to another type before it can be auto-marshaled using an Automation interface.

V is the TVarData record for the custom Variant that the application wants to marshal.

RequiredVarType returns the type to which V must be cast before it is Automation-compatible.

OlePromotion returns true if V can be cast to an Automation-compatible type, false if there is no type to which V can be cast.

As implemented in TCustomVariantType (see page 23581), OlePromotion sets RequiredVarType to varOleStr and returns true if V does not already represent a Unicode string.

See Also

casttoole (see page 23589)

171.1.16.4.15 TCustomVariantType.QueryInterface Method

Returns a reference to a specified interface if the object supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the TCustomVariantType (see page 23581) object supports the interface specified by IID. If so, QueryInterface

increments the reference count.

sets the Obj parameter so that it points to an instance of the specified interface.

returns 0 to indicate success.

If the object does not support the interface, QueryInterface returns a nonzero error code such as E_NoInterface.

See Also

_addref (see page 23586)

_release (see page 23587)

171.1.16.4.16 TCustomVariantType.RaiseCastError Method

Generates a system error that indicates an invalid cast.

Pascal

```
procedure RaiseCastError;
```

C++

```
__fastcall RaiseCastError();
```

Description

Call RaiseCastError from the implementation of a TCustomVariantType (see page 23581) descendant when an operation fails because a Variant value could not be cast to a necessary type.

See Also

Cast (see page 23588)

castto (see page 23589)

171.1.16.4.17 TCustomVariantType.RaiseDispError Method

Generates a system error indicating an invalid dispatch attempt.

Pascal

```
procedure RaiseDispError;
```

C++

```
__fastcall RaiseDispError();
```

Description

Call RaiseDispError from the implementation of a TCustomVariantType (see page 23581) descendant when an operation fails because the variant can't execute a method call. This raises EVariantDispatchError (see page 23575).

See Also

DispInvoke (see page 23592)

EVariantDispatchError (see page 23575)

171.1.16.4.18 TCustomVariantType.RaiseInvalidOp Method

Generates a system error indicating an invalid operation.

Pascal

```
procedure RaiseInvalidOp;
```

C++

```
__fastcall RaiseInvalidOp();
```

Description

Call RaiseInvalidOp from the implementation of a TCustomVariantType (see page 23581) descendant when it can't perform an operation.

See Also

BinaryOp (see page 23587)

UnaryOp (see page 23598)

CompareOp (see page 23590)

Compare (see page 23590)

171.1.16.4.19 TCustomVariantType.RightPromotion Method

Indicates whether the right argument to a binary operation should be coerced to a different type.

Pascal

```
function RightPromotion(const V: TVarData; const Operator: TVarOp; out RequiredVarType:
TVarType): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean RightPromotion(const TVarData V, const TVarOp Operator, TVarType
RequiredVarType);
```

Description

Override RightPromotion to indicate when the right-hand argument for a binary or comparison operation should be implicitly cast to another type before performing the operation. RightPromotion is called when the right-hand Variant in a binary or comparison operation is of this Variant type.

Note: In operations that involve two custom Variants (see page 23567), the left-hand argument's RightPromotion method is called first. The right-hand argument's LeftPromotion (see page 23593) method is then called only if the left-hand argument's RightPromotion method returns false.

V is the TVarData record for the left-hand argument of the operation.

Operator indicates the type of operation. It can be any of the operators in the following table:

Value	Operation
opAdd	addition
opSubtract	subtraction
opMultiply	multiplication
opDivide	floating-point division
opIntDivide	integer division

opModulus	remainder
opShiftRight	right shift
opShiftRight	right shift
opAnd	bitwise and
opOr	bitwise or
opXor	bitwise exclusive or
opCompare	any comparison operation

RequiredVarType returns the Variant type code for the type to which the right-hand argument should be cast before performing the operation. If the operation can proceed with the right-hand argument left as-is, RequiredVarType returns the VType field of V.

RightPromotion returns true if the TCustomVariantType (see page 23581) descendant can perform the operation (using the BinaryOp (see page 23587), CompareOp (see page 23590), or Compare (see page 23590) method) assuming the right-hand argument can be cast to RequiredVarType. It returns false if the TCustomVariantType (see page 23581) descendant can't perform the operation.

As implemented in TCustomVariantType (see page 23581), RightPromotion returns the VarType (see page 23606) property as RequiredVarType, and returns true.

See Also

BinaryOp (see page 23587)

CompareOp (see page 23590)

Compare (see page 23590)

VarType (see page 23606)

LeftPromotion (see page 23593)

Implementing Binary Operations

171.1.16.4.20 TCustomVariantType.SimplisticClear Method

Clears a TVarData record by re-initializing it.

Pascal

```
procedure SimplisticClear(var V: TVarData);
```

C++

```
__fastcall SimplisticClear(TVarData V);
```

Description

Call SimplisticClear from the implementation of a TCustomVariantType (see page 23581) descendant to clear the TVarData record of a Variant of the custom type. This method should only be called if the TVarData record does not contain any members that must be freed.

SimplisticClear simply calls the VarDataInit (see page 23602) method, which reinitializes the TVarData record passed as V.

See Also

Clear (see page 23589)

VarDataClear (see page 23600)

VarDataInit (see page 23602)

171.1.16.4.21 TCustomVariantType.SimplisticCopy Method

Performs a simple block copy of one TVarData record to another.

Pascal

```
procedure SimplisticCopy(var Dest: TVarData; const Source: TVarData; const Indirect: Boolean =
False);
```

C++

```
__fastcall SimplisticCopy(TVarData Dest, const TVarData Source, const Boolean Indirect =
False);
```

Description

Call SimplisticCopy from the implementation of a TCustomVariantType (see page 23581) descendant to block copy one TVarData record to another.

Dest is the TVarData record that receives the copy.

Source is the TVarData record to copy.

Indirect indicates whether Source represents the data (false) or merely a reference to the data (true).

See Also

Copy (see page 23591)

VarDataCopy (see page 23601)

VarDataCopyNoInd (see page 23601)

Copying and Clearing Custom Variants

171.1.16.4.22 TCustomVariantType.UnaryOp Method

Implements any unary operations of which the custom Variant type is capable.

Pascal

```
procedure UnaryOp(var Right: TVarData; const Operator: TVarOp); virtual;
```

C++

```
virtual __fastcall UnaryOp(TVarData Right, const TVarOp Operator);
```

Description

Override UnaryOp to implement any unary operators the custom Variant type supports.

Right is the value that appears to the right of the operator. UnaryOp changes this value to indicate the result of the operation.

Operator identifies the operator that appears before Right. It can be any of the following values:

Value	Operation
opNegate	unary minus
opNot	logical negation

As implemented in TCustomVariantType (see page 23581), UnaryOp raises a system error indicating an invalid operation.

See Also

BinaryOp (see page 23587)

CompareOp (see page 23590)

Implementing Unary Operations

171.1.16.4.23 TCustomVariantType.VarDataCast Method

Performs a cast from one Variant type to another.

Pascal

```
procedure VarDataCast(var Dest: TVarData; const Source: TVarData);
```

C++

```
__fastcall VarDataCast(TVarData Dest, const TVarData Source);
```

Description

Call VarDataCast from the implementation of a TCustomVariantType (see page 23581) descendant to perform casts on the data from another Variant type to the type specified by the VarType (see page 23606) property. Do not use VarDataCast as the entire implementation of the Cast (see page 23588) method, as this will lead to a circular reference.

Dest returns the result of the cast.

Source is the TVarData record of another Variant type that is to be converted.

See Also

Cast (see page 23588)

castto (see page 23589)

VarDataCastTo (see page 23599)

VarType (see page 23606)

171.1.16.4.24 VarDataCastTo Method

This is the overview for the VarDataCastTo method overload.

Overload List

	Name	Description
	TCustomVariantType.VarDataCastTo (TVarData, TVarData, TVarType) (see page 23599)	Performs a cast from one Variant type to another.
	TCustomVariantType.VarDataCastTo (TVarData, TVarType) (see page 23599)	Performs a cast from one Variant type to another.

171.1.16.4.24.1 TCustomVariantType.VarDataCastTo Method (TVarData, TVarData, TVarType)

Performs a cast from one Variant type to another.

Pascal

```
procedure VarDataCastTo(var Dest: TVarData; const Source: TVarData; const AVarType: TVarType);
overload;
procedure VarDataCastTo(var Dest: TVarData; const AVarType: TVarType); overload;
```

C++

```
__fastcall VarDataCastTo(TVarData Dest, const TVarData Source, const TVarType AVarType);
```

```
__fastcall VarDataCastTo(TVarData Dest, const TVarType AVarType);
```

Description

Call VarDataCastTo from the implementation of a TCustomVariantType (see page 23581) descendant to perform casts on the data from one Variant type to another. Do not use VarDataCastTo as the entire implementation of the CastTo (see page 23589) method, as this will lead to a circular reference.

Dest returns the result of the cast. If VarDataCastTo is called with no Source parameter, it also provides the source data to be cast.

Source is the TVarData record to be cast.

VarType (see page 23606) is the type code of the Variant type to which the data should be cast.

See Also

castto (see page 23589)

171.1.16.4.25 TCustomVariantType.VarDataCastToOleStr Method

Performs a cast to a Variant type of varOleStr.

Pascal

```
procedure VarDataCastToOleStr(var Dest: TVarData);
```

C++

```
__fastcall VarDataCastToOleStr(TVarData Dest);
```

Description

Call VarDataCastToOleStr from the implementation of a TCustomVariantType (see page 23581) descendant to perform casts to a TVarData record with Variant type code of varOleStr (Unicode string).

Dest is the TVarData record of a Variant, and returns the result of the cast.

See Also

VarDataFromOleStr (see page 23602)

VarDataToStr (see page 23605)

171.1.16.4.26 TCustomVariantType.VarDataClear Method

Clears the data from a built-in Variant type.

Pascal

```
procedure VarDataClear(var Dest: TVarData);
```

C++

```
__fastcall VarDataClear(TVarData Dest);
```

Description

Call VarDataClear from the implementation of a TCustomVariantType (see page 23581) descendant to clear the data in the TVarData record of a Variant. Do not use VarDataClear as the entire implementation of the Clear (see page 23589) method, as this will lead to a circular reference.

Dest is the TVarData record of a Variant type, and returns the result after clearing the data.

See Also

SimplisticClear (see page 23597)

Clear (see page 23589)

171.1.16.4.27 TCustomVariantType.VarDataCopy Method

Copies the data from one built-in Variant type to another.

Pascal

```
procedure VarDataCopy(var Dest: TVarData; const Source: TVarData);
```

C++

```
__fastcall VarDataCopy(TVarData Dest, const TVarData Source);
```

Description

Call VarDataCopy from the implementation of a TCustomVariantType (see page 23581) descendant to clear the data from a destination Variant and then copy the TVarData record from another Variant. VarDataCopy can handle data that is stored indirectly (when the VType field includes varByRef). Do not use VarDataCopy as the entire implementation of the Copy (see page 23591) method, as this will lead to a circular reference.

Dest is the TVarData record that receives the copy.

Source is the TVarData record to be copied.

See Also

SimplisticCopy (see page 23598)

VarDataCopyNoInd (see page 23601)

Copy (see page 23591)

171.1.16.4.28 TCustomVariantType.VarDataCopyNoInd Method

Copies the data from one built-in Variant type to another.

Pascal

```
procedure VarDataCopyNoInd(var Dest: TVarData; const Source: TVarData);
```

C++

```
__fastcall VarDataCopyNoInd(TVarData Dest, const TVarData Source);
```

Description

Call VarDataCopyNoInd from the implementation of a TCustomVariantType (see page 23581) descendant to clear the data from a destination Variant and then copy the TVarData record from another Variant. Unlike the VarDataCopy (see page 23601) method, it can't handle data that is stored indirectly (when the VType field includes varByRef). However, it is often more efficient than VarDataCopy (see page 23601).

Dest is the TVarData record that receives the copy.

Source is the TVarData record to be copied.

See Also

VarDataCopy (see page 23601)

Copy (see page 23591)

171.1.16.4.29 TCustomVariantType.VarDataFromOleStr Method

Creates a TVarData record to represent a Unicode string.

Pascal

```
procedure VarDataFromOleStr(var V: TVarData; const Value: WideString);
```

C++

```
__fastcall VarDataFromOleStr(TVarData V, const BSTR Value);
```

Description

Call VarDataCopy (see page 23601) from the implementation of a TCustomVariantType (see page 23581) descendant to fill out a TVarData record so that it represents a specified Unicode string.

V is the TVarData record to be filled out.

Value is a Unicode string that becomes the value of V.

See Also

VarDataCastToOleStr (see page 23600)

VarDataFromStr (see page 23602)

171.1.16.4.30 TCustomVariantType.VarDataFromStr Method

Creates a TVarData record to represent a string.

Pascal

```
procedure VarDataFromStr(var V: TVarData; const Value: string);
```

C++

```
__fastcall VarDataFromStr(TVarData V, const AnsiString Value);
```

Description

Call VarDataCopy (see page 23601) from the implementation of a TCustomVariantType (see page 23581) descendant to fill out a TVarData record so that it represents a specified string.

V is the TVarData record to be filled out.

Value is a string that becomes the value of V.

See Also

VarDataFromOleStr (see page 23602)

VarDataIsStr (see page 23605)

VarDataToStr (see page 23605)

171.1.16.4.31 TCustomVariantType.VarDataInit Method

Initializes a TVarData record.

Pascal

```
procedure VarDataInit(var Dest: TVarData);
```

C++

```
__fastcall VarDataInit(TVarData Dest);
```

Description

Call VarDataInit from the implementation of a TCustomVariantType (see page 23581) descendant to initialize a newly-allocated TVarData record.

See Also

VarDataClear (see page 23600)

171.1.16.4.32 TCustomVariantType.VarDataIsArray Method

Indicates whether a TVarData record represents the data from a Variant array.

Pascal

```
function VarDataIsArray(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsArray(const TVarData V);
```

Description

VarDataIsArray returns true if the VType field of V contains the varArray bit.

See Also

VarDataIsByRef (see page 23603)

VarType (see page 23667)

171.1.16.4.33 TCustomVariantType.VarDataIsByRef Method

Indicates whether a TVarData record stores its data indirectly.

Pascal

```
function VarDataIsByRef(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsByRef(const TVarData V);
```

Description

VarDataIsArray (see page 23603) returns true if the VType field of V contains the varByRef bit.

See Also

VarDataIsArray (see page 23603)

VarType (see page 23667)

171.1.16.4.34 TCustomVariantType.VarDataIsEmptyParam Method

Indicates whether a TVarData record represents an unassigned optional parameter.

Pascal

```
function VarDataIsEmptyParam(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsEmptyParam(const TVarData V);
```

Description

VarDataIsEmptyParam returns true if the given TVarData record represents an unassigned optional parameter.

See Also

VarDataIsArray (see page 23603)

VarType (see page 23667)

171.1.16.4.35 TCustomVariantType.VarDataIsFloat Method

Indicates whether a TVarData record represents a floating-point value.

Pascal

```
function VarDataIsFloat(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsFloat(const TVarData V);
```

Description

VarDataIsFloat returns true if the given TVarData record comes from a built-in Variant type that represents a floating-point value (Single, Double, or Currency). VarDataIsFloat does not return true for data from Custom Variant types.

See Also

VarDataIsNumeric (see page 23604)

VarType (see page 23667)

VarDataIsOrdinal (see page 23605)

VarDataIsStr (see page 23605)

171.1.16.4.36 TCustomVariantType.VarDataIsNumeric Method

Indicates whether a TVarData record represents a numeric value.

Pascal

```
function VarDataIsNumeric(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsNumeric(const TVarData V);
```

Description

VarDataIsNumeric returns true if the given TVarData record represents a numeric value of some sort. This can be either a floating-point value or an ordinal value. VarDataIsNumeric does not return true for data from Custom Variant types.

See Also

VarDataIsFloat (see page 23604)

VarType (see page 23667)

VarDataIsOrdinal (see page 23605)

VarDataIsStr (see page 23605)

171.1.16.4.37 TCustomVariantType.VarDataIsOrdinal Method

Indicates whether a TVarData record represents a ordinal value.

Pascal

```
function VarDataIsOrdinal(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsOrdinal(const TVarData V);
```

Description

VarDataIsOrdinal returns true if the given TVarData record comes from a built-in Variant type that represents a ordinal value (such as an integer or Boolean type). VarDataIsOrdinal does not return true for data from Custom Variant types.

See Also

VarDataIsNumeric (see page 23604)

VarType (see page 23667)

VarDataIsFloat (see page 23604)

VarDataIsStr (see page 23605)

171.1.16.4.38 TCustomVariantType.VarDataIsStr Method

Indicates whether a TVarData record represents a string (normal or Unicode).

Pascal

```
function VarDataIsStr(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsStr(const TVarData V);
```

Description

VarDataIsStr returns true if the given TVarData record has a VType field of varString or varOleStr.

See Also

VarDataIsNumeric (see page 23604)

VarType (see page 23667)

VarDataIsFloat (see page 23604)

VarDataIsStr

171.1.16.4.39 TCustomVariantType.VarDataToStr Method

Returns a string that represents the value from a TVarData record.

Pascal

```
function VarDataToStr(const V: TVarData): string;
```

C++

```
__fastcall AnsiString VarDataToStr(const TVarData V);
```

Description

Call VarDataToStr from the implementation of a TCustomVariantType (see page 23581) descendant to extract a string that represents the value of a Variant.

V is the TVarData record that represents the value of the Variant.

See Also

VarDataCastToOleStr (see page 23600)

VarDataFromStr (see page 23602)

171.1.16.5 TCustomVariantType Properties

The properties of the TCustomVariantType class are listed here.

Public Properties

	Name	Description
R	VarType (see page 23606)	Indicates the Variant type code for the custom Variant type.

171.1.16.5.1 TCustomVariantType.VarType Property

Indicates the Variant type code for the custom Variant type.

Pascal

```
property VarType: TVarType;
```

C++

```
__property TVarType VarType;
```

Description

The VarType property provides read-only access to the type code that identifies a custom Variant data type.

The value of VarType is set in the constructor. This can be an automatically generated value, or it can be a value explicitly passed to the constructor. Note that if the VarType is automatically-generated, it may not remain the same for every application that uses the custom Variant type: its value will depend on the other custom Variants (see page 23567) also used in the application.

See Also

TVarData (see page 22571)

Create (see page 23584)

VarType (see page 23667)

FindCustomVariantType (see page 23640)

Enabling Casting

171.1.17 TInvokeableVariantType Class

TInvokeableVariantType is the base class for custom Variant types that implement custom Variants (see page 23567) with properties and methods.

Class Hierarchy

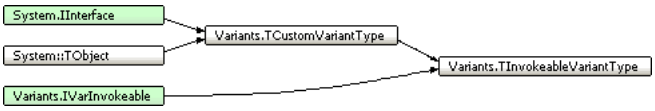

Pascal

```
TInvokeableVariantType = class(TCustomVariantType, IVarInvokeable);
```

C++

```
class TInvokeableVariantType : public TCustomVariantType, public IVarInvokeable;
```

File

Variants

Description

Use TInvokeableVariantType to define new data types for Delphi Variants (see page 23567) with properties and methods. TInvokeableVariantType descendants do not represent the Variants (see page 23567) themselves. Rather, they act as utility classes that the custom Variants (see page 23567) call on to perform their operations.

Note: Do not define TInvokeableVariantType descendants in C++. The logic for custom Variant types must be defined in Delphi code.

To create a Variant type that supports properties and methods,

1. Map the storage of the Variant's data on to the TVarData record.
2. Write a method that creates instances of your custom Variant. This fills in the Variant's data as defined in step 1.
3. Declare a class that descends from TInvokeableVariantType.
4. Implement all required behavior (including type conversion rules) in the new class. This involves overriding the appropriate virtual methods (typically DoFunction (see page 23610), DoProcedure (see page 23611), GetProperty (see page 23612), SetProperty (see page 23613), BinaryOp (see page 23587), UnaryOp (see page 23598), CompareOp (see page 23590) or Compare (see page 23590), Cast (see page 23588), CastTo (see page 23589), Clear (see page 23589), Copy (see page 23591), IsClear (see page 23592), LeftPromotion (see page 23593), and RightPromotion (see page 23596)).
5. In the initialization section of the unit that defines your TInvokeableVariantType descendant, create an instance of your class. When you instantiate your object, it automatically registers itself with the Variant-handling system so that the new Variant type is enabled.
6. In the finalization section of the unit that defines your TInvokeableVariantType descendant, free the instance of your class. This automatically unregisters the Variant type.

For an example, see the VarConv (see page 23561) unit, which implements a custom Variant type for measurements.

Tip: If the custom Variant type uses an object instance to store its data, you can automatically implement the published properties of that object by deriving your custom Variant type from TPublishableVariantType instead.

See Also

IDH

TPublishableVariantType (see page 23332)

TVarData (see page 22571)

Supporting Properties and Methods in Custom Variants

171.1.17.1 TInvokeableVariantType Members

The following tables list the members exposed by TInvokeableVariantType.

Protected Methods

	Name	Description
	_AddRef (see page 23586)	Increments the reference count for this interface.
	_Release (see page 23587)	Decrements the reference count for this interface.
V	DispInvoke (see page 23610)	Invokes a method call by dispatching it to the appropriate TInvokeableVariantType (see page 23606) method.
V	FixupIdent (see page 23612)	Fixes up dispatch identifier.
V	LeftPromotion (see page 23593)	Indicates whether the left argument to a binary operation should be coerced to a different type.
V	OlePromotion (see page 23594)	Indicates the type to which a Variant value must be cast to make it Automation-compatible.
V	QueryInterface (see page 23594)	Returns a reference to a specified interface if the object supports that interface.
	RaiseCastError (see page 23595)	Generates a system error that indicates an invalid cast.
	RaiseDispError (see page 23595)	Generates a system error indicating an invalid dispatch attempt.
	RaiseInvalidOp (see page 23596)	Generates a system error indicating an invalid operation.
V	RightPromotion (see page 23596)	Indicates whether the right argument to a binary operation should be coerced to a different type.
	SimplisticClear (see page 23597)	Clears a TVarData record by re-initializing it.
	SimplisticCopy (see page 23598)	Performs a simple block copy of one TVarData record to another.
	VarDataCast (see page 23599)	Performs a cast from one Variant type to another.
	VarDataCastTo (see page 23599)	This is the overview for the VarDataCastTo method overload.
	VarDataCastToOleStr (see page 23600)	Performs a cast to a Variant type of varOleStr.
	VarDataClear (see page 23600)	Clears the data from a built-in Variant type.
	VarDataCopy (see page 23601)	Copies the data from one built-in Variant type to another.
	VarDataCopyNoInd (see page 23601)	Copies the data from one built-in Variant type to another.
	VarDataFromOleStr (see page 23602)	Creates a TVarData record to represent a Unicode string.
	VarDataFromStr (see page 23602)	Creates a TVarData record to represent a string.
	VarDataInit (see page 23602)	Initializes a TVarData record.
	VarDataIsArray (see page 23603)	Indicates whether a TVarData record represents the data from a Variant array.
	VarDataIsByRef (see page 23603)	Indicates whether a TVarData record stores its data indirectly.
	VarDataIsEmptyParam (see page 23603)	Indicates whether a TVarData record represents an unassigned optional parameter.
	VarDataIsFloat (see page 23604)	Indicates whether a TVarData record represents a floating-point value.
	VarDataIsNumeric (see page 23604)	Indicates whether a TVarData record represents a numeric value.
	VarDataIsOrdinal (see page 23605)	Indicates whether a TVarData record represents a ordinal value.
	VarDataIsStr (see page 23605)	Indicates whether a TVarData record represents a string (normal or Unicode).
	VarDataToStr (see page 23605)	Returns a string that represents the value from a TVarData record.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BinaryOp (see page 23587)	Implements any binary operations of which the custom Variant type is capable.
V	Cast (see page 23588)	Converts the data from another Variant type so that it represents this custom type's data format.
V	CastTo (see page 23589)	This is CastTo, a member of class TCustomVariantType.
V	CastToOle (see page 23589)	This is CastToOle, a member of class TCustomVariantType.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 23589)	Sets the data for this custom Variant type to a blank or unassigned value.
V	Compare (see page 23590)	Implements comparisons that involve the custom Variant type.
V	CompareOp (see page 23590)	Implements any comparison operations of which the custom Variant type is capable.
A	Copy (see page 23591)	Copies the data from a Variant of this custom type.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 23584)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 23585)	Frees an instance of TCustomVariantType (see page 23581).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	DoFunction (see page 23610)	Implements all methods that return values.
V	DoProcedure (see page 23611)	Implements all methods that do not return values.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetProperty (see page 23612)	Returns the value for any property this Variant type supports.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsClear (see page 23592)	Indicates whether the data from a custom Variant represents a blank or unassigned value.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetProperty (see page 23613)	Sets the value for any property this Variant type supports.
V	UnaryOp (see page 23598)	Implements any unary operations of which the custom Variant type is capable.

Public Properties

	Name	Description
R	VarType (see page 23606)	Indicates the Variant type code for the custom Variant type.

171.1.17.2 TInvokeableVariantType Methods

The methods of the TInvokeableVariantType class are listed here.

Protected Methods

	Name	Description
	DispInvoke (see page 23610)	Invokes a method call by dispatching it to the appropriate TInvokeableVariantType (see page 23606) method.
	FixupIdent (see page 23612)	Fixes up dispatch identifier.

Public Methods

	Name	Description
	DoFunction (see page 23610)	Implements all methods that return values.
	DoProcedure (see page 23611)	Implements all methods that do not return values.
	GetProperty (see page 23612)	Returns the value for any property this Variant type supports.
	SetProperty (see page 23613)	Sets the value for any property this Variant type supports.

171.1.17.2.1 TInvokeableVariantType.DispInvoke Method

Invokes a method call by dispatching it to the appropriate TInvokeableVariantType (see page 23606) method.

Pascal

```
procedure DispInvoke(Dest: PVarData; const Source: TVarData; CallDesc: PCallDesc; Params:
Pointer); override;
```

C++

```
virtual __fastcall DispInvoke(PVarData Dest, const TVarData Source, PCallDesc CallDesc, void *
Params);
```

Description

When the application tries to call a method, set a property, or read a property using a Variant of the custom Variant type, the system automatically generates a call to DispInvoke. TInvokeableVariantType (see page 23606) responds by generating a call to the GetProperty (see page 23612), SetProperty (see page 23613), DoProcedure (see page 23611), or DoFunction (see page 23610) method, as appropriate.

Dest is a TVarData record that receives the return value of any function calls.

Source is the TVarData record that holds the data from the custom Variant whose property or method was invoked.

CallDesc identifies the method that was called, including the type and number of its arguments.

Params points to an array of parameter values that were passed in the method call.

See Also

GetProperty (see page 23612)

SetProperty (see page 23613)

DoProcedure (see page 23611)

DoFunction (see page 23610)

171.1.17.2.2 TInvokeableVariantType.DoFunction Method

Implements all methods that return values.

Pascal

```
function DoFunction(var Dest: TVarData; const V: TVarData; const Name: string; const
Arguments: TVarDataArray): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoFunction(TVarData Dest, const TVarData V, const AnsiString Name,
const TVarDataArray Arguments);
```

Description

Override DoFunction to provide the implementation for method calls that return values. When the application invokes a Variant member function (Delphi) or calls the Variant's Exec method (C++) a call is automatically generated to DoFunction.

Dest is a TVarData record that receives the value that the function returns.

V is the TVarData record from the Variant instance that was used to call the function.

Name is the name of the function.

Arguments is a dynamic array of TVarData records, where each one represents the value of an argument passed to the function.

DoFunction returns true if it executes the method call, false if it does not implement the specified function.

As implemented in TInvokeableVariant, DoFunction always returns false, because it does not implement any function calls.

See Also

GetProperty (see page 23612)

DoProcedure (see page 23611)

SetProperty (see page 23613)

Exec

Supporting Properties and Methods in Custom Variants

171.1.17.2.3 TInvokeableVariantType.DoProcedure Method

Implements all methods that do not return values.

Pascal

```
function DoProcedure(const V: TVarData; const Name: string; const Arguments: TVarDataArray):
Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoProcedure(const TVarData V, const AnsiString Name, const
TVarDataArray Arguments);
```

Description

Override DoProcedure to provide the implementation for methods calls that do not return values. When the application invokes a Variant member procedure (Delphi) or calls the Variant's Exec method (C++) a call is automatically generated to DoProcedure that allows the TInvokeableVariant descendant to implement the call.

V is the TVarData record from the Variant instance that was used to call the procedure.

Name is the name of the procedure.

Arguments is a dynamic array of TVarData records, where each one represents the value of an argument passed to the procedure.

DoProcedure returns true if it executes the procedure call, false if it does not implement the specified procedure.

As implemented in TInvokeableVariant, DoProcedure always returns false, because it does not implement any procedure calls.

See Also

GetProperty (see page 23612)

DoFunction (see page 23610)

SetProperty (see page 23613)

Exec

Supporting Properties and Methods in Custom Variants

171.1.17.2.4 TInvokeableVariantType.FixupIdent Method

Fixes up dispatch identifier.

Pascal

```
function FixupIdent(const AText: string): string; virtual;
```

C++

```
virtual __fastcall AnsiString FixupIdent(const AnsiString AText);
```

Description

FixupIdent is called internally to determine the final fixup for the dispatch identifier. As defined in TInvokeableVariantType (see page 23606), this fixup consists of converting the identifier to all upper case.

171.1.17.2.5 TInvokeableVariantType.GetProperty Method

Returns the value for any property this Variant type supports.

Pascal

```
function GetProperty(var Dest: TVarData; const V: TVarData; const Name: string): Boolean;
virtual;
```

C++

```
virtual __fastcall Boolean GetProperty(TVarData Dest, const TVarData V, const AnsiString Name);
```

Description

Override GetProperty to provide the read implementations for properties of instances of the Variant. When the application reads a Variant property (Delphi) or calls the Variant's Exec method (C++) a call is automatically generated to GetProperty that allows the TInvokeableVariant descendant to implement the call.

Dest is a TVarData record that receives the property value.

V is the TVarData record from the Variant instance whose property is being read.

Name is the name of the property.

GetProperty returns true if it returns a property value, false if it does not implement the specified property.

As implemented in TInvokeableVariant, GetProperty always returns false, because it does not implement any properties.

See Also

DoFunction (see page 23610)

DoProcedure (see page 23611)

SetProperty (see page 23613)

Exec

Supporting Properties and Methods in Custom Variants

171.1.17.2.6 TInvokeableVariantType.SetProperty Method

Sets the value for any property this Variant type supports.

Pascal

```
function SetProperty(const V: TVarData; const Name: string; const Value: TVarData): Boolean;
virtual;
```

C++

```
virtual __fastcall Boolean SetProperty(const TVarData V, const AnsiString Name, const TVarData
Value);
```

Description

Override SetProperty to provide the write implementation for the properties of instances of the invokeable Variant type. When the application writes to a Variant property (Delphi) or calls the Variant's Exec method (C++) a call is automatically generated to SetProperty that allows the TInvokeableVariant descendant to implement the call.

Name is the name of the property.

Value is a TVarData record that contains the new property value.

GetProperty (see page 23612) returns true if it sets a property value, false if it does not implement the specified property or if the specified property is read-only.

As implemented in TInvokeableVariant, GetProperty (see page 23612) always returns false, because it does not implement any properties.

See Also

DoFunction (see page 23610)

DoProcedure (see page 23611)

GetProperty (see page 23612)

Exec

Supporting Properties and Methods in Custom Variants

171.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IVarInstanceReference (see page 23614)	This is class Variants.IVarInstanceReference.
	IVarInvokeable (see page 23615)	This is class Variants.IVarInvokeable.

171.2.1 IVarInstanceReference Interface

Class Hierarchy

Variants.IVarInstanceReference

Pascal

```
IVarInstanceReference = interface;
```

C++

```
__interface IVarInstanceReference;
```

File

Variants

Description

This is class Variants.IVarInstanceReference.

171.2.1.1 IVarInstanceReference Members

The following tables list the members exposed by IVarInstanceReference.

Public Methods

	Name	Description
	GetInstance (see page 23615)	This is GetInstance, a member of class IVarInstanceReference.

171.2.1.2 IVarInstanceReference Methods

The methods of the IVarInstanceReference class are listed here.

Public Methods

	Name	Description
	GetInstance (see page 23615)	This is GetInstance, a member of class IVarInstanceReference.

171.2.1.2.1 IVarInstanceReference.GetInstance Method

Pascal

```
function GetInstance(const V: TVarData): TObject;
```

C++

```
__fastcall TObject * GetInstance(const TVarData V);
```

Description

This is GetInstance, a member of class IVarInstanceReference.

171.2.2 IVarInvokeable Interface

Class Hierarchy

Variants.IVarInvokeable

Pascal

```
IVarInvokeable = interface;
```

C++

```
__interface IVarInvokeable;
```

File

Variants

Description

This is class Variants.IVarInvokeable.

171.2.2.1 IVarInvokeable Members

The following tables list the members exposed by IVarInvokeable.

Public Methods

	Name	Description
	DoFunction (see page 23616)	This is DoFunction, a member of class IVarInvokeable.
	DoProcedure (see page 23616)	This is DoProcedure, a member of class IVarInvokeable.
	GetProperty (see page 23616)	This is GetProperty, a member of class IVarInvokeable.
	SetProperty (see page 23616)	This is SetProperty, a member of class IVarInvokeable.

171.2.2.2 IVarInvokeable Methods

The methods of the IVarInvokeable class are listed here.

Public Methods

	Name	Description
	DoFunction (see page 23616)	This is DoFunction, a member of class IVarInvokeable.
	DoProcedure (see page 23616)	This is DoProcedure, a member of class IVarInvokeable.
	GetProperty (see page 23616)	This is GetProperty, a member of class IVarInvokeable.

	SetProperty (see page 23616)	This is SetProperty, a member of class IVarInvokeable.

171.2.2.2.1 IVarInvokeable.DoFunction Method

Pascal

```
function DoFunction(var Dest: TVarData; const V: TVarData; const Name: string; const
Arguments: TVarDataArray): Boolean;
```

C++

```
__fastcall Boolean DoFunction(TVarData Dest, const TVarData V, const AnsiString Name, const
TVarDataArray Arguments);
```

Description

This is DoFunction, a member of class IVarInvokeable.

171.2.2.2.2 IVarInvokeable.DoProcedure Method

Pascal

```
function DoProcedure(const V: TVarData; const Name: string; const Arguments: TVarDataArray):
Boolean;
```

C++

```
__fastcall Boolean DoProcedure(const TVarData V, const AnsiString Name, const TVarDataArray
Arguments);
```

Description

This is DoProcedure, a member of class IVarInvokeable.

171.2.2.2.3 IVarInvokeable.GetProperty Method

Pascal

```
function GetProperty(var Dest: TVarData; const V: TVarData; const Name: string): Boolean;
```

C++

```
__fastcall Boolean GetProperty(TVarData Dest, const TVarData V, const AnsiString Name);
```

Description

This is GetProperty, a member of class IVarInvokeable.

171.2.2.2.4 IVarInvokeable.SetProperty Method

Pascal

```
function SetProperty(const V: TVarData; const Name: string; const Value: TVarData): Boolean;
```

C++

```
__fastcall Boolean SetProperty(const TVarData V, const AnsiString Name, const TVarData Value);
```

Description

This is SetProperty, a member of class IVarInvokeable.

171.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	_DispInvoke (see page 23619)	This is function Variants._DispInvoke.
	_OleVarFromInt (see page 23620)	This is function Variants._OleVarFromInt.
	_OleVarFromInt64 (see page 23620)	This is function Variants._OleVarFromInt64.
	_OleVarFromLStr (see page 23620)	This is function Variants._OleVarFromLStr.
	_OleVarFromPStr (see page 23621)	This is function Variants._OleVarFromPStr.
	_OleVarFromVar (see page 23621)	This is function Variants._OleVarFromVar.
	_VarAdd (see page 23621)	This is function Variants._VarAdd.
	_VarAddRef (see page 23622)	This is function Variants._VarAddRef.
	_VarAnd (see page 23622)	This is function Variants._VarAnd.
	_VarArrayGet (see page 23622)	This is function Variants._VarArrayGet.
	_VarArrayPut (see page 23623)	This is function Variants._VarArrayPut.
	_VarArrayRedim (see page 23623)	This is function Variants._VarArrayRedim.
	_VarCast (see page 23623)	This is function Variants._VarCast.
	_VarCastOle (see page 23624)	This is function Variants._VarCastOle.
	_VarClear (see page 23624)	This is function Variants._VarClear.
	_VarClr (see page 23624)	This is function Variants._VarClr.
	_VarCmpEQ (see page 23625)	This is function Variants._VarCmpEQ.
	_VarCmpGE (see page 23625)	This is function Variants._VarCmpGE.
	_VarCmpGT (see page 23625)	This is function Variants._VarCmpGT.
	_VarCmpLE (see page 23626)	This is function Variants._VarCmpLE.
	_VarCmpLT (see page 23626)	This is function Variants._VarCmpLT.
	_VarCmpNE (see page 23626)	This is function Variants._VarCmpNE.
	_VarCopy (see page 23627)	This is function Variants._VarCopy.
	_VarCopyNoInd (see page 23627)	This is function Variants._VarCopyNoInd.
	_VarDiv (see page 23627)	This is function Variants._VarDiv.
	_VarFromBool (see page 23628)	This is function Variants._VarFromBool.
	_VarFromCurr (see page 23628)	This is function Variants._VarFromCurr.
	_VarFromDisp (see page 23628)	This is function Variants._VarFromDisp.
	_VarFromDynArray (see page 23629)	This is function Variants._VarFromDynArray.
	_VarFromInt (see page 23629)	This is function Variants._VarFromInt.
	_VarFromInt64 (see page 23629)	This is function Variants._VarFromInt64.
	_VarFromIntf (see page 23630)	This is function Variants._VarFromIntf.
	_VarFromLStr (see page 23630)	This is function Variants._VarFromLStr.
	_VarFromPStr (see page 23630)	This is function Variants._VarFromPStr.
	_VarFromReal (see page 23631)	This is function Variants._VarFromReal.
	_VarFromTDateTime (see page 23631)	This is function Variants._VarFromTDateTime.
	_VarFromWStr (see page 23631)	This is function Variants._VarFromWStr.
	_VarMod (see page 23632)	This is function Variants._VarMod.
	_VarMul (see page 23632)	This is function Variants._VarMul.
	_VarNeg (see page 23632)	This is function Variants._VarNeg.
	_VarNot (see page 23633)	This is function Variants._VarNot.
	_VarOr (see page 23633)	This is function Variants._VarOr.
	_VarRDiv (see page 23633)	This is function Variants._VarRDiv.
	_VarShl (see page 23634)	This is function Variants._VarShl.

	_VarShr (see page 23634)	This is function Variants._VarShr.
	_VarSub (see page 23634)	This is function Variants._VarSub.
	_VarToBool (see page 23635)	This is function Variants._VarToBool.
	_VarToCurrency (see page 23635)	This is function Variants._VarToCurrency.
	_VarToDisp (see page 23635)	This is function Variants._VarToDisp.
	_VarToDynArray (see page 23636)	This is function Variants._VarToDynArray.
	_VarToInt64 (see page 23636)	This is function Variants._VarToInt64.
	_VarToInteger (see page 23636)	This is function Variants._VarToInteger.
	_VarToIntf (see page 23637)	This is function Variants._VarToIntf.
	_VarToLStr (see page 23637)	This is function Variants._VarToLStr.
	_VarToPStr (see page 23637)	This is function Variants._VarToPStr.
	_VarToReal (see page 23638)	This is function Variants._VarToReal.
	_VarToWStr (see page 23638)	This is function Variants._VarToWStr.
	_VarXor (see page 23638)	This is function Variants._VarXor.
	DynArrayFromVariant (see page 23639)	Creates a dynamic array from a Variant.
	DynArrayToVariant (see page 23639)	Creates a Variant array from a dynamic array.
	FindCustomVariantType (see page 23640)	Retrieves the object that implements a custom Variant type.
	FindCustomVariantType (see page 23640)	Retrieves the object that implements a custom Variant type.
	FindVarData (see page 23640)	Returns pointer to variant's data.
	HandleConversionException (see page 23641)	Handles exceptions during variant type conversions.
	Null (see page 23641)	Returns a Null variant.
	Unassigned (see page 23643)	Returns an "empty" variant.
	VarArrayAsPSafeArray (see page 23643)	This is function Variants.VarArrayAsPSafeArray.
	VarArrayCreate (see page 23644)	Creates a variant array.
	VarArrayCreateError (see page 23645)	Indicates an array creation error.
	VarArrayDimCount (see page 23645)	Returns number of dimensions of a variant array.
	VarArrayGet (see page 23645)	Returns a Variant that represents a single value from a multi-dimensional Variant array.
	VarArrayHighBound (see page 23646)	Returns high bound for a dimension in a variant array.
	VarArrayLock (see page 23646)	Locks a variant array and returns a pointer to the data.
	VarArrayLowBound (see page 23647)	Returns the low bound of a dimension in a variant array.
	VarArrayOf (see page 23648)	Creates and fills a one-dimensional variant array.
	VarArrayPut (see page 23648)	Sets the value of a single cell in a multi-dimensional Variant array.
	VarArrayRef (see page 23649)	Returns a reference to the specified variant array.
	VarArrayUnlock (see page 23649)	Unlocks a variant array.
	VarAsError (see page 23650)	Converts a HRESULT into an Error variant.
	VarAsType (see page 23650)	Converts a variant to specified type.
	VarCastError (see page 23651)	Raises an EVariantTypeCastError (see page 23580) exception.
	VarCastError (see page 23651)	Raises an EVariantTypeCastError (see page 23580) exception.
	VarCheckEmpty (see page 23651)	Raises an exception if a specified variant's value is Unassigned (see page 23643).
	VarCompareValue (see page 23652)	Returns the relationship between Variants (see page 23567).
	VarCopyNoInd (see page 23652)	Copies a Variant.
	VarEnsureRange (see page 23653)	Returns a Variant with a value in a specified range.
	VarFromDateTime (see page 23653)	Returns a variant containing a specified data and time.
	VarInRange (see page 23654)	Indicates whether a Variant's value is in a specified range.
	VarInvalidNullOp (see page 23654)	Indicate variant operation on null.
	VarInvalidOp (see page 23655)	Indicate invalid variant operation.
	VarIsArray (see page 23655)	Indicates whether the specified variant is an array.
	VarIsArray (see page 23655)	Indicates whether the specified variant is an array.
	VarIsByRef (see page 23656)	Indicates whether the specified variant's value is by reference.
	VarIsClear (see page 23656)	Indicates whether the specified variant has an undefined value.
	VarIsCustom (see page 23657)	Indicates whether the specified variant is a custom variant.
	VarIsEmpty (see page 23657)	Indicates whether the specified variant is unassigned.

	VarIsEmptyParam (see page 23658)	Indicates whether the specified variant represents an unassigned optional parameter.
	VarIsError (see page 23659)	Indicates whether the specified variant is an error variant; optionally converts variant back to HRESULT.
	VarIsError (see page 23659)	Indicates whether the specified variant is an error variant; optionally converts variant back to HRESULT.
	VarIsFloat (see page 23659)	Indicates whether the specified variant represents a floating-point value.
	VarIsNull (see page 23660)	Indicates whether the specified variant is Null (see page 23641).
	VarIsNumeric (see page 23660)	Indicates whether the specified variant represents a numeric value.
	VarIsOrdinal (see page 23661)	Indicates whether the specified variant represents an ordinal value.
	VarIsStr (see page 23661)	Indicates whether the specified variant represents a string value.
	VarIsType (see page 23662)	Indicates whether the specified variant represents a specified type.
	VarIsType (see page 23662)	Indicates whether the specified variant represents a specified type.
	VarOverflowError (see page 23662)	Raises an EVariantOverflowError (see page 23579) exception.
	VarRangeCheckError (see page 23663)	Raises an EVariantRangeCheckError (see page 23580) exception.
	VarResultCheck (see page 23663)	Raises variant exception based on HRESULT value.
	VarResultCheck (see page 23663)	Raises variant exception based on HRESULT value.
	VarSameValue (see page 23664)	Indicates whether two Variants (see page 23567) have equivalent values.
	VarSupports (see page 23664)	Indicates whether the Variant's value is an interface that supports a specified interface.
	VarSupports (see page 23664)	Indicates whether the Variant's value is an interface that supports a specified interface.
	VarToDateTime (see page 23665)	Converts a specified variant to a TDateTime value.
	VarToStr (see page 23665)	Converts a variant's value to a string.
	VarToStrDef (see page 23666)	Converts a variant's value to a string.
	VarToWideStr (see page 23666)	Converts a variant's value to a WideString.
	VarToWideStrDef (see page 23667)	Converts a variant's value to a WideString.
	VarType (see page 23667)	Returns the type code of a specified variant.
	VarTypeAsText (see page 23668)	Returns variant type descriptive string.
	VarTypeIsValidArrayType (see page 23669)	Indicates whether a Variant type code represents a type that can be used for the elements in a Variant array.
	VarTypeIsValidElementType (see page 23669)	Indicates whether a Variant type code represents a type that can be used for the elements in a Variant array.

171.3.1 Variants._DispInvoke Function

Pascal

```
procedure _DispInvoke(Dest: PVarData; const Source: TVarData; CallDesc: PCallDesc; Params:
Pointer); cdecl;
```

C++

```
__cdecl _DispInvoke(PVarData Dest, const TVarData Source, PCallDesc CallDesc, void * Params);
```

File

Variants

Description

This is function Variants._DispInvoke.

171.3.2 Variants._OleVarFromInt Function

Pascal

```
procedure _OleVarFromInt(var V: TVarData; const Value: Integer; const Range: ShortInt);
```

C++

```
_OleVarFromInt(TVarData V, const int Value, const ShortInt Range);
```

File

Variants

Description

This is function Variants._OleVarFromInt.

171.3.3 Variants._OleVarFromInt64 Function

Pascal

```
procedure _OleVarFromInt64(var V: TVarData; const Value: Int64);
```

C++

```
_OleVarFromInt64(TVarData V, const Int64 Value);
```

File

Variants

Description

This is function Variants._OleVarFromInt64.

171.3.4 Variants._OleVarFromLStr Function

Pascal

```
procedure _OleVarFromLStr(var V: TVarData; const Value: string);
```

C++

```
_OleVarFromLStr(TVarData V, const AnsiString Value);
```

File

Variants

Description

This is function Variants._OleVarFromLStr.

171.3.5 Variants._OleVarFromPStr Function

Pascal

```
procedure _OleVarFromPStr(var V: TVarData; const Value: ShortString);
```

C++

```
_OleVarFromPStr(TVarData V, const ShortString Value);
```

File

Variants

Description

This is function Variants._OleVarFromPStr.

171.3.6 Variants._OleVarFromVar Function

Pascal

```
procedure _OleVarFromVar(var Dest: TVarData; const Source: TVarData);
```

C++

```
_OleVarFromVar(TVarData Dest, const TVarData Source);
```

File

Variants

Description

This is function Variants._OleVarFromVar.

171.3.7 Variants._VarAdd Function

Pascal

```
procedure _VarAdd(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarAdd(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarAdd.

171.3.8 Variants._VarAddRef Function

Pascal

```
procedure _VarAddRef(var V: TVarData);
```

C++

```
_VarAddRef(TVarData V);
```

File

Variants

Description

This is function Variants._VarAddRef.

171.3.9 Variants._VarAnd Function

Pascal

```
procedure _VarAnd(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarAnd(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarAnd.

171.3.10 Variants._VarArrayGet Function

Pascal

```
function _VarArrayGet(var A: TVarData; IndexCount: Integer; const Indices:
TVarArrayCoorArray): TVarData; cdecl;
```

C++

```
__cdecl TVarData _VarArrayGet(TVarData A, int IndexCount, const TVarArrayCoorArray Indices);
```

File

Variants

Description

This is function Variants._VarArrayGet.

171.3.11 Variants._VarArrayPut Function

Pascal

```
procedure _VarArrayPut(var A: TVarData; const Value: TVarData; IndexCount: Integer; const
Indices: TVarArrayCoorArray); cdecl;
```

C++

```
__cdecl _VarArrayPut(TVarData A, const TVarData Value, int IndexCount, const
TVarArrayCoorArray Indices);
```

File

Variants

Description

This is function Variants._VarArrayPut.

171.3.12 Variants._VarArrayRedim Function

Pascal

```
procedure _VarArrayRedim(var A: TVarData; HighBound: Integer);
```

C++

```
_VarArrayRedim(TVarData A, int HighBound);
```

File

Variants

Description

This is function Variants._VarArrayRedim.

171.3.13 Variants._VarCast Function

Pascal

```
procedure _VarCast(var Dest: TVarData; const Source: TVarData; AVarType: Integer);
```

C++

```
_VarCast(TVarData Dest, const TVarData Source, int AVarType);
```

File

Variants

Description

This is function Variants._VarCast.

171.3.14 Variants._VarCastOle Function

Pascal

```
procedure _VarCastOle(var Dest: TVarData; const Source: TVarData; AVarType: Integer);
```

C++

```
_VarCastOle(TVarData Dest, const TVarData Source, int AVarType);
```

File

Variants

Description

This is function Variants._VarCastOle.

171.3.15 Variants._VarClear Function

Pascal

```
procedure _VarClear(var V: TVarData);
```

C++

```
_VarClear(TVarData V);
```

File

Variants

Description

This is function Variants._VarClear.

171.3.16 Variants._VarClr Function

Pascal

```
procedure _VarClr(var V: TVarData);
```

C++

```
_VarClr(TVarData V);
```

File

Variants

Description

This is function Variants._VarClr.

171.3.17 Variants._VarCmpEQ Function

Pascal

```
procedure _VarCmpEQ(const Left: TVarData; const Right: TVarData);
```

C++

```
_VarCmpEQ(const TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarCmpEQ.

171.3.18 Variants._VarCmpGE Function

Pascal

```
procedure _VarCmpGE(const Left: TVarData; const Right: TVarData);
```

C++

```
_VarCmpGE(const TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarCmpGE.

171.3.19 Variants._VarCmpGT Function

Pascal

```
procedure _VarCmpGT(const Left: TVarData; const Right: TVarData);
```

C++

```
_VarCmpGT(const TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarCmpGT.

171.3.20 Variants._VarCmpLE Function

Pascal

```
procedure _VarCmpLE(const Left: TVarData; const Right: TVarData);
```

C++

```
_VarCmpLE(const TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarCmpLE.

171.3.21 Variants._VarCmpLT Function

Pascal

```
procedure _VarCmpLT(const Left: TVarData; const Right: TVarData);
```

C++

```
_VarCmpLT(const TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarCmpLT.

171.3.22 Variants._VarCmpNE Function

Pascal

```
procedure _VarCmpNE(const Left: TVarData; const Right: TVarData);
```

C++

```
_VarCmpNE(const TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarCmpNE.

171.3.23 Variants._VarCopy Function

Pascal

```
procedure _VarCopy(var Dest: TVarData; const Source: TVarData);
```

C++

```
_VarCopy(TVarData Dest, const TVarData Source);
```

File

Variants

Description

This is function Variants._VarCopy.

171.3.24 Variants._VarCopyNoInd Function

Pascal

```
procedure _VarCopyNoInd(var Dest: TVarData; const Source: TVarData);
```

C++

```
_VarCopyNoInd(TVarData Dest, const TVarData Source);
```

File

Variants

Description

This is function Variants._VarCopyNoInd.

171.3.25 Variants._VarDiv Function

Pascal

```
procedure _VarDiv(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarDiv(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarDiv.

171.3.26 Variants._VarFromBool Function

Pascal

```
procedure _VarFromBool(var V: TVarData; const Value: Boolean);
```

C++

```
_VarFromBool(TVarData V, const Boolean Value);
```

File

Variants

Description

This is function Variants._VarFromBool.

171.3.27 Variants._VarFromCurr Function

Pascal

```
procedure _VarFromCurr;
```

C++

```
_VarFromCurr();
```

File

Variants

Description

This is function Variants._VarFromCurr.

171.3.28 Variants._VarFromDisp Function

Pascal

```
procedure _VarFromDisp(var V: TVarData; const Value: IDispatch);
```

C++

```
_VarFromDisp(TVarData V, const IDispatch Value);
```

File

Variants

Description

This is function Variants._VarFromDisp.

171.3.29 Variants._VarFromDynArray Function

Pascal

```
procedure _VarFromDynArray(var V: TVarData; const DynArray: Pointer; TypeInfo: Pointer);
```

C++

```
_VarFromDynArray(TVarData V, const void * DynArray, void * TypeInfo);
```

File

Variants

Description

This is function Variants._VarFromDynArray.

171.3.30 Variants._VarFromInt Function

Pascal

```
procedure _VarFromInt(var V: TVarData; const Value: Integer; const Range: ShortInt);
```

C++

```
_VarFromInt(TVarData V, const int Value, const ShortInt Range);
```

File

Variants

Description

This is function Variants._VarFromInt.

171.3.31 Variants._VarFromInt64 Function

Pascal

```
procedure _VarFromInt64(var V: TVarData; const Value: Int64);
```

C++

```
_VarFromInt64(TVarData V, const Int64 Value);
```

File

Variants

Description

This is function Variants._VarFromInt64.

171.3.32 Variants._VarFromIntf Function

Pascal

```
procedure _VarFromIntf(var V: TVarData; const Value: IInterface);
```

C++

```
_VarFromIntf(TVarData V, const IInterface Value);
```

File

Variants

Description

This is function Variants._VarFromIntf.

171.3.33 Variants._VarFromLStr Function

Pascal

```
procedure _VarFromLStr(var V: TVarData; const Value: string);
```

C++

```
_VarFromLStr(TVarData V, const AnsiString Value);
```

File

Variants

Description

This is function Variants._VarFromLStr.

171.3.34 Variants._VarFromPStr Function

Pascal

```
procedure _VarFromPStr(var V: TVarData; const Value: ShortString);
```

C++

```
_VarFromPStr(TVarData V, const ShortString Value);
```

File

Variants

Description

This is function Variants._VarFromPStr.

171.3.35 Variants._VarFromReal Function

Pascal

```
procedure _VarFromReal;
```

C++

```
_VarFromReal();
```

File

Variants

Description

This is function Variants._VarFromReal.

171.3.36 Variants._VarFromTDateTime Function

Pascal

```
procedure _VarFromTDateTime;
```

C++

```
_VarFromTDateTime();
```

File

Variants

Description

This is function Variants._VarFromTDateTime.

171.3.37 Variants._VarFromWStr Function

Pascal

```
procedure _VarFromWStr(var V: TVarData; const Value: WideString);
```

C++

```
_VarFromWStr(TVarData V, const BSTR Value);
```

File

Variants

Description

This is function Variants._VarFromWStr.

171.3.38 Variants._VarMod Function

Pascal

```
procedure _VarMod(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarMod(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarMod.

171.3.39 Variants._VarMul Function

Pascal

```
procedure _VarMul(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarMul(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarMul.

171.3.40 Variants._VarNeg Function

Pascal

```
procedure _VarNeg(var Dest: TVarData);
```

C++

```
_VarNeg(TVarData Dest);
```

File

Variants

Description

This is function Variants._VarNeg.

171.3.41 Variants._VarNot Function

Pascal

```
procedure _VarNot(var Dest: TVarData);
```

C++

```
_VarNot(TVarData Dest);
```

File

Variants

Description

This is function Variants._VarNot.

171.3.42 Variants._VarOr Function

Pascal

```
procedure _VarOr(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarOr(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarOr.

171.3.43 Variants._VarRDiv Function

Pascal

```
procedure _VarRDiv(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarRDiv(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarRDiv.

171.3.44 Variants._VarShl Function

Pascal

```
procedure _VarShl(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarShl(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarShl.

171.3.45 Variants._VarShr Function

Pascal

```
procedure _VarShr(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarShr(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarShr.

171.3.46 Variants._VarSub Function

Pascal

```
procedure _VarSub(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarSub(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarSub.

171.3.47 Variants._VarToBool Function

Pascal

```
function _VarToBool(const V: TVarData): LongBool;
```

C++

```
LongBool _VarToBool(const TVarData V);
```

File

Variants

Description

This is function Variants._VarToBool.

171.3.48 Variants._VarToCurrency Function

Pascal

```
function _VarToCurrency(const V: TVarData): Currency;
```

C++

```
Currency _VarToCurrency(const TVarData V);
```

File

Variants

Description

This is function Variants._VarToCurrency.

171.3.49 Variants._VarToDisp Function

Pascal

```
procedure _VarToDisp(var Dispatch: IDispatch; const V: TVarData);
```

C++

```
_VarToDisp(IDispatch Dispatch, const TVarData V);
```

File

Variants

Description

This is function Variants._VarToDisp.

171.3.50 Variants._VarToDynArray Function

Pascal

```
procedure _VarToDynArray(var DynArray: Pointer; const V: TVarData; TypeInfo: Pointer);
```

C++

```
_VarToDynArray(void * DynArray, const TVarData V, void * TypeInfo);
```

File

Variants

Description

This is function Variants._VarToDynArray.

171.3.51 Variants._VarToInt64 Function

Pascal

```
function _VarToInt64(const V: TVarData): Int64;
```

C++

```
Int64 _VarToInt64(const TVarData V);
```

File

Variants

Description

This is function Variants._VarToInt64.

171.3.52 Variants._VarToInteger Function

Pascal

```
function _VarToInteger(const V: TVarData): Integer;
```

C++

```
int _VarToInteger(const TVarData V);
```

File

Variants

Description

This is function Variants._VarToInteger.

171.3.53 Variants._VarToIntf Function

Pascal

```
procedure _VarToIntf(var Intf: IInterface; const V: TVarData);
```

C++

```
_VarToIntf(IInterface Intf, const TVarData V);
```

File

Variants

Description

This is function Variants._VarToIntf.

171.3.54 Variants._VarToLStr Function

Pascal

```
procedure _VarToLStr(var S: string; const V: TVarData);
```

C++

```
_VarToLStr(AnsiString S, const TVarData V);
```

File

Variants

Description

This is function Variants._VarToLStr.

171.3.55 Variants._VarToPStr Function

Pascal

```
procedure _VarToPStr(var S; const V: TVarData);
```

C++

```
_VarToPStr( S, const TVarData V);
```

File

Variants

Description

This is function Variants._VarToPStr.

171.3.56 Variants._VarToReal Function

Pascal

```
function _VarToReal(const V: TVarData): Extended;
```

C++

```
Extended _VarToReal(const TVarData V);
```

File

Variants

Description

This is function Variants._VarToReal.

171.3.57 Variants._VarToWStr Function

Pascal

```
procedure _VarToWStr(var S: WideString; const V: TVarData);
```

C++

```
_VarToWStr(BSTR S, const TVarData V);
```

File

Variants

Description

This is function Variants._VarToWStr.

171.3.58 Variants._VarXor Function

Pascal

```
procedure _VarXor(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarXor(TVarData Left, const TVarData Right);
```

File

Variants

Description

This is function Variants._VarXor.

171.3.59 Variants.DynArrayFromVariant Function

Creates a dynamic array from a Variant.

Pascal

```
procedure DynArrayFromVariant(var DynArray: Pointer; const V: Variant; TypeInfo: Pointer);
```

C++

```
DynArrayFromVariant(void * DynArray, const Variant V, void * TypeInfo);
```

File

Variants

Description

Call DynArrayFromVariant to copy the contents of a variant (V) to an existing dynamic array variable (DynArray).

DynArray is a dynamic array variable whose value is assigned from the Variant.

V is the Variant to copy. Its value must be an array.

TypeInfo points to the runtime type information that identifies the type of array elements. You can obtain this pointer using the TypeInfo function (Delphi) or the __delphirtti function (C++).

See Also

DynArrayToVariant (see page 23639)

TypeInfo

171.3.60 Variants.DynArrayToVariant Function

Creates a Variant array from a dynamic array.

Pascal

```
procedure DynArrayToVariant(var V: Variant; const DynArray: Pointer; TypeInfo: Pointer);
```

C++

```
DynArrayToVariant(Variant V, const void * DynArray, void * TypeInfo);
```

File

Variants

Description

Call DynArrayToVariant to assign the contents of a dynamic array as the value of a Variant variable.

V is the Variant whose value is assigned by DynArrayToVariant.

DynArray is a dynamic array to copy.

TypeInfo points to the runtime type information that identifies the type of array elements. You can obtain this pointer using the TypeInfo function (Delphi) or the __delphirtti function (C++).

See Also

DynArrayFromVariant (see page 23639)

TypeInfo

171.3.61 Variants.FindCustomVariantType Function

Retrieves the object that implements a custom Variant type.

Pascal

```
function FindCustomVariantType(const AVarType: TVarType; out CustomVariantType:
TCustomVariantType): Boolean; overload;
function FindCustomVariantType(const TypeName: string; out CustomVariantType:
TCustomVariantType): Boolean; overload;
```

C++

```
Boolean FindCustomVariantType(const TVarType AVarType, TCustomVariantType CustomVariantType);
Boolean FindCustomVariantType(const AnsiString TypeName, TCustomVariantType CustomVariantType);
```

File

Variants

Description

Use FindCustomVariantType to obtain a reference to the object that implements a custom Variant type.

TypeName is the name of the TCustomVariantType (see page 23581) descendant class you want to retrieve.

AVarType is the type code of Variant whose implementation class you want to retrieve. If you have an instance of a Variant whose value is a custom Variant Type, you can obtain a value for this parameter using the VarType (see page 23667) function.

CustomVariantType returns an instance of the TCustomVariantType (see page 23581) descendant that implements the indicated custom Variant type.

See Also

VarType (see page 23667)

VarIsCustom (see page 23657)

171.3.62 Variants.FindVarData Function

Returns pointer to variant's data.

Pascal

```
function FindVarData(const V: Variant): PVarData;
```

C++

```
PVarData FindVarData(const Variant V);
```

File

Variants

Description

FindVarData returns a pointer to the Variant data contained referenced by a Variant. If the Variant contains another Variant, FindVarData follows the chain of references until it finds the actual data.

See Also

VarType (see page 23667)

171.3.63 Variants.HandleConversionException Function

Handles exceptions during variant type conversions.

Pascal

```
procedure HandleConversionException(const ASourceType: TVarType; const ADestType: TVarType);
```

C++

```
HandleConversionException(const TVarType ASourceType, const TVarType ADestType);
```

File

Variants

Description

HandleConversionException handles exceptions resulting from attempts to convert between variant types. Code that implements such conversions should call HandleConversionException when a conversion throws an exception.

ASourceType and ADestType indicate the variant types of the source and destination. HandleConversionException uses these to format an appropriate error string. If the current exception corresponds to a variant exception, HandleConversionException raises that exception. Otherwise, HandleConversionException simply re-raises the original exception.

See Also

VarResultCheck (see page 23663)

EVariantError

171.3.64 Variants.Null Function

Returns a Null variant.

Pascal

```
function Null: Variant;
```

C++

```
Variant Null();
```

File

Variants

Description

Use Null to obtain a Null variant that can indicate unknown or missing data. Null Variants (see page 23567) can be assigned to variant variables in an application that must contain a null value. Assigning Null to a variant variable does not cause an error, and

Null can be returned from any function with a variant return value.

Assigning Null to a variable of any type other than Variant causes either a compile-time error or raises an EVariantTypeMismatch exception. For example, in the following code the assignment of v, the Null variant, to variant q is successful; whereas the conversion of variant v, which is now Null, to the integer return type of the Test function, raises an exception.

function Test(v: Variant): Integer;

```
var
  q: Variant;
  msg: string;
begin
  q := v; { this is ok, since q is a variant }
  if VarIsNull(q) then
    msg := 'q is a null variant'
  else
    msg := 'q is not a null variant';
  ShowMessage(msg);

  Result := v; { this raises an exception!!!}
```

end;

procedure TForm1.Button1Click(Sender: TObject);

```
begin
  Test(Null);
```

end;

int Test(Variant v)

```
{
  Variant q = v; // this is ok, since q is a variant
  AnsiString msg;
  if (VarIsNull(q))
    msg = "q is a null variant";
  else
    msg = "q is not a null variant";
  ShowMessage(msg);

   return v; // this throws an exception!!!
```

}

void __fastcall TForm1::Button1Click(TObject *Sender)

{

```
  Test(Null());
```

}

Except for comparisons, which always result in Boolean values, expressions involving Null variants always result in Null variants.

Note: The Null variant returned by the Null function is different from the nil constant in Delphi code and the NULL macro in C++ code. Neither nil nor NULL is a variant. The nil constant is an object reference that does not refer to any actual object. The NULL macro is a pointer that does not refer to any actual object or value.

See Also

Unassigned (see page 23643)

VarIsNull (see page 23660)

VarType (see page 23667)

NullEqualityRule (see page 23678)

NullMagnitudeRule (see page 23679)

171.3.65 Variants.Unassigned Function

Returns an "empty" variant.

Pascal

```
function Unassigned: Variant;
```

C++

```
Variant Unassigned();
```

File

Variants

Description

A variant variable can be "empty", meaning it has not yet been assigned to. The Unassigned function returns an empty variant, which can be assigned to a variant variable to restore the variable to its initial state.

Use the VarIsEmpty (see page 23657) function to test whether a variant is empty. When used on an empty variant, the VarType (see page 23667) standard function returns varEmpty.

If an empty variant is cast to another type (for example, by assigning to a non-variant variable or calling VarAsType (see page 23650)) the following conversions occur:

Destination	Conversion result
Numeric type	Zero
String type	Empty string
Boolean	False.

Note: Unassigned is useful with variants referencing OLE Automation Objects that you want to keep "alive" until another value is assigned to the variant.

There's corresponding functionality under Linux. Should document when time permits.

See Also

Null (see page 23641)

VarIsEmpty (see page 23657)

VarType (see page 23667)

VarAsType (see page 23650)

171.3.66 Variants.VarArrayAsPSafeArray Function

Pascal

```
function VarArrayAsPSafeArray(const A: Variant): PVarArray;
```

C++

```
PVarArray VarArrayAsPSafeArray(const Variant A);
```

File

Variants

Description

This is function Variants.VarArrayAsPSafeArray.

171.3.67 Variants.VarArrayCreate Function

Creates a variant array.

Pascal

```
function VarArrayCreate(const Bounds: array of Integer; AVarType: TVarType): Variant;
```

C++

```
Variant VarArrayCreate(const array of Integer Bounds, TVarType AVarType);
```

File

Variants

Description

VarArrayCreate creates a variant array with the bounds given by Bounds and the element type given by VarType (see page 23667).

The Bounds parameter must contain an even number of values, where each pair of values specifies the upper and lower bounds of one dimension of the array.

Note: The Bounds_size parameter is the index of the last value in the Bounds array. This is one less than the total number of values. Thus, Bounds_size must always be an odd number.

The element type of the array, given by the VarType (see page 23667) parameter, is a variant type code. This must be one of the constants defined in the System (see page 22079) unit. It cannot include the varArray or varByRef bits. The element type cannot be varString or a custom Variant type. To create a variant array of strings use the varOleStr type code. If the element type is varVariant, the elements of the array are themselves variants and can in turn contain variant arrays.

If VarArrayCreate is unable to create the array, an EVariantArrayCreateError (see page 23572) is raised.

Note: :Variant arrays with an element type of varByte are the preferred method of passing binary data between OLE Automation controllers and servers. Such arrays are subject to no translation of their data, and can be efficiently accessed using the VarArrayLock (see page 23646) and VarArrayUnlock (see page 23649) routines.

There's corresponding functionality under Linux. Should document when time permits.

See Also

VarArrayOf (see page 23648)

VarArrayLock (see page 23646)

VarArrayUnlock (see page 23649)

EVariantArrayCreateError (see page 23572)

171.3.68 Variants.VarArrayCreateError Function

Indicates an array creation error.

Pascal

```
procedure VarArrayCreateError;
```

C++

```
VarArrayCreateError();
```

File

Variants

See Also

EVariantArrayCreateError (see page 23572)

171.3.69 Variants.VarArrayDimCount Function

Returns number of dimensions of a variant array.

Pascal

```
function VarArrayDimCount(const A: Variant): Integer;
```

C++

```
int VarArrayDimCount(const Variant A);
```

File

Variants

Description

VarArrayDimCount returns the number of dimensions of the given variant. The resulting value is zero if the given variant is not a variant array.

See Also

VarArrayHighBound (see page 23646)

VarArrayLowBound (see page 23647)

171.3.70 Variants.VarArrayGet Function

Returns a Variant that represents a single value from a multi-dimensional Variant array.

Pascal

```
function VarArrayGet(const A: Variant; const Indices: array of Integer): Variant;
```

C++

```
Variant VarArrayGet(const Variant A, const array of Integer Indices);
```

File

Variants

Description

Use VarArrayGet to obtain the value of a single cell from a multi-dimensional Variant array.

A is the Variant array that contains the desired value.

Indices is an array of index values, one for each dimension of the array.

Note: Indices_Size is the index of the last value in Indices (one less than the number of dimensions in the array).

If the values specified by Indices are out of bounds, VarArrayGet raises an EVariantBadIndexError (see page 23574).

See Also

VarArrayPut (see page 23648)

EVariantBadIndexError (see page 23574)

171.3.71 Variants.VarArrayHighBound Function

Returns high bound for a dimension in a variant array.

Pascal

```
function VarArrayHighBound(const A: Variant; Dim: Integer): Integer;
```

C++

```
int VarArrayHighBound(const Variant A, int Dim);
```

File

Variants

Description

VarArrayHighBound returns the high bound of the given dimension in the given variant array. The Dim parameter should be 1 for the leftmost dimension, 2 for the second leftmost dimension, and so on. An exception is raised if the variant given by A is not an array (EVariantInvalidArgError (see page 23575)), or if the dimension specified by Dim does not exist (EVariantBadIndexError (see page 23574)).

See Also

VarArrayDimCount (see page 23645)

VarArrayLowBound (see page 23647)

EVariantInvalidArgError (see page 23575)

EVariantBadIndexError (see page 23574)

171.3.72 Variants.VarArrayLock Function

Locks a variant array and returns a pointer to the data.

Pascal

```
function VarArrayLock(const A: Variant): Pointer;
```

C++

```
void * VarArrayLock(const Variant A);
```

File

Variants

Description

VarArrayLock locks the given variant array and returns a pointer to the data stored in the array. While locked, a variant array can not be resized and any calls to VarArrayRedim will fail. Once locked, a variant array must later be unlocked using VarArrayUnlock (see page 23649). An EVariantInvalidArgError (see page 23575) exception is raise if the variant given by A is not an array.

The pointer returned by VarArrayLock points to an array of elements in which the leftmost dimension increases first. In other words, the dimensions of the returned array pointer are reversed from the dimensions of the variant array.

Once it is verified that a variant array has the correct bounds and dimensions, VarArrayLock can be used to gain direct access to the array data, which results in improved performance.

See Also

VarArrayUnlock (see page 23649)

VarArrayRedim

171.3.73 Variants.VarArrayLowBound Function

Returns the low bound of a dimension in a variant array.

Pascal

```
function VarArrayLowBound(const A: Variant; Dim: Integer): Integer;
```

C++

```
int VarArrayLowBound(const Variant A, int Dim);
```

File

Variants

Description

VarArrayLowBound returns the low bound of the given dimension in the given variant array. The Dim parameter should be 1 for the leftmost dimension, 2 for the second leftmost dimension, and so on. An exception is raised if the variant given by A is not an array (EVariantInvalidArgError (see page 23575)), or if the dimension specified by Dim does not exist (EVariantBadIndexError (see page 23574))..

See Also

VarArrayDimCount (see page 23645)

VarArrayHighBound (see page 23646)

EVariantInvalidArgError (see page 23575)

EVariantBadIndexError (see page 23574)

171.3.74 Variants.VarArrayOf Function

Creates and fills a one-dimensional variant array.

Pascal

```
function VarArrayOf(const Values: array of Variant): Variant;
```

C++

```
Variant VarArrayOf(const array of Variant Values);
```

File

Variants

Description

VarArrayOf returns a one-dimensional variant array with the elements given by the Values parameter

Values is an array of Variant values to convert into a Variant array.

Note: Values_Size indicates the index of the last value in the Values array (one less than the number of values).

The low bound of the returned array is zero, the high bound is the number of values given by the Values parameter less one (Values_Size), and the element type is Variant.

See Also

VarArrayCreate (see page 23644)

171.3.75 Variants.VarArrayPut Function

Sets the value of a single cell in a multi-dimensional Variant array.

Pascal

```
procedure VarArrayPut(var A: Variant; const Value: Variant; const Indices: array of Integer);
```

C++

```
VarArrayPut(Variant A, const Variant Value, const array of Integer Indices);
```

File

Variants

Description

Use VarArrayPut to set the value of a single cell in a multi-dimensional Variant array.

A is the Variant array that contains the cell to set.

Value is the value to assign to the specified cell.

Indices is an array of index values, one for each dimension of the array.

Note: Indices_Size is the index of the last value in Indices (one less than the number of dimensions in the array).

If the values specified by Indices are out of bounds, VarArrayPut raises an EVariantBadIndexError (see page 23574).

See Also

VarArrayGet (see page 23645)

EVariantBadIndexError (see page 23574)

171.3.76 Variants.VarArrayRef Function

Returns a reference to the specified variant array.

Pascal

```
function VarArrayRef(const A: Variant): Variant;
```

C++

```
Variant VarArrayRef(const Variant A);
```

File

Variants

Description

Use VarArrayRef to reference a variant array when a referenced variant array is expected in an API call. VarArrayRef creates a new variant array of the same type as A, but with an extra level of indirection if A does not include the varByRef type bit. The returned variant includes the varByRef type bit.

If A is not an array, VarArrayRef raises an EVariantInvalidArgError (see page 23575) exception.

See Also

VarArrayCreate (see page 23644)

TVarType

VarCopyNoInd (see page 23652)

EVariantInvalidArgError (see page 23575)

171.3.77 Variants.VarArrayUnlock Function

Unlocks a variant array.

Pascal

```
procedure VarArrayUnlock(const A: Variant);
```

C++

```
VarArrayUnlock(const Variant A);
```

File

Variants

Description

VarArrayUnlock unlocks a variant array that was previously locked by VarArrayLock (see page 23646). If the variant given by A is not an array, VarArrayUnlock raises an EVariantInvalidArtgError exception.

See Also

VarArrayLock (see page 23646)

171.3.78 Variants.VarAsError Function

Converts a HRESULT into an Error variant.

Pascal

```
function VarAsError(AResult: HRESULT): Variant;
```

C++

```
Variant VarAsError(HRESULT AResult);
```

File

Variants

Description

VarAsError converts an HRESULT result code into an equivalent variant of type varError.

See Also

VarType (see page 23667)

171.3.79 Variants.VarAsType Function

Converts a variant to specified type.

Pascal

```
function VarAsType(const V: Variant; AVarType: TVarType): Variant;
```

C++

```
Variant VarAsType(const Variant V, TVarType AVarType);
```

File

Variants

Description

VarAsType converts a variant to a specified type and returns a new Variant that has the specified type.

V is the Variant to convert.

VarType (see page 23667) is a variant type code that indicates the type to which V should be converted. This can be one of the constants defined in the System (see page 22079) unit, or it can be the type code for a custom variant type. It cannot include the varArray or varByRef bits.

VarAsType raises an exception if the variant cannot be converted to the given type. This will be EVariantError or one of its descendants, such as EVariantOverflowError (see page 23579) or EVariantTypeMismatchError.

See Also

VarType (see page 23667)

171.3.80 Variants.VarCastError Function

Raises an EVariantTypeCastError (see page 23580) exception.

Pascal

```
procedure VarCastError; overload;
procedure VarCastError(const ASourceType: TVarType; const ADestType: TVarType); overload;
```

C++

```
VarCastError();
VarCastError(const TVarType ASourceType, const TVarType ADestType);
```

File

Variants

Description

Call VarCastError to indicate that a variant operation has failed due to cast error. VarCastError raises EVariantTypeCastError (see page 23580). If no arguments are specified, the error string is a general error message. If ASourceType and ADestType are specified, the error string incorporates the names of the specified Variant Types (see page 23293).

See Also

EVariantTypeCastError (see page 23580)

TVarData

VarType (see page 23667)

171.3.81 Variants.VarCheckEmpty Function

Raises an exception if a specified variant's value is Unassigned (see page 23643).

Pascal

```
procedure VarCheckEmpty(const V: Variant);
```

C++

```
VarCheckEmpty(const Variant V);
```

File

Variants

Description

Call VarCheckEmpty when your code requires that a Variant have a value, and you want to raise an exception if it doesn't. VarCheckEmpty checks whether the Variant specified by V has a value of Unassigned (see page 23643). If so, it raises an EVariantTypeMismatchError exception.

To check whether the Variant is Unassigned (see page 23643) without raising an exception, use the VarIsEmpty (see page 23657) function instead. To check whether the Variant's value is undefined, use the VarIsClear (see page 23656) function instead.

Note: Do not confuse an Unassigned (see page 23643) variant with a Variant whose value is Null (see page 23641). A Null (see page 23641) variant is still assigned, but has the value Null (see page 23641). Unlike unassigned variants, Null (see

page 23641) variants can be used in expressions and can be converted to other types of variants.

See Also

VarIsEmpty (see page 23657)

Unassigned (see page 23643)

VarType (see page 23667)

VarIsNull (see page 23660)

VarIsClear (see page 23656)

EVariantTypeMismatchError

171.3.82 Variants.VarCompareValue Function

Returns the relationship between Variants (see page 23567).

Pascal

```
function VarCompareValue(const A: Variant; const B: Variant): TVariantRelationship;
```

C++

```
TVariantRelationship VarCompareValue(const Variant A, const Variant B);
```

File

Variants

Description

Call VarCompareValue to determine the relationship between two Variants (see page 23567).

A and B are the values to compare.

VarCompareValue returns a value that indicates whether A and B are equal (vrEqual), whether one is greater than the other (vrLessThan or vrGreaterThan), or whether the two values can't be ordered relative to each other (vrNotEqual).

See Also

VarSameValue (see page 23664)

CompareValue

171.3.83 Variants.VarCopyNoInd Function

Copies a Variant.

Pascal

```
procedure VarCopyNoInd(var Dest: Variant; const Source: Variant);
```

C++

```
VarCopyNoInd(Variant Dest, const Variant Source);
```

File

Variants

Description

VarCopyNoInd copies the variant given by Source into the variant given by Dest. Calling VarCopyNoInd in C++ or Delphi code is equivalent to a simple variant assignment in Delphi code.

Dest can be a Variant or an OleVariant, and it must be possible to assign a value to it.

Source is defined as a Variant, but an OleVariant is also allowed because it is a compatible type. However, Source can't have a VarType (see page 23667) that includes the varByRef bit.

The following table lists the expected behavior when using Variant and OleVariant types as parameters:

Dest	Source	Behavior
Variant	OleVariant	Works as expected.
Variant	Variant	Works as expected.
OleVariant	OleVariant	Works as expected.
OleVariant	Variant	Source should be implicitly cast to an OleVariant.

171.3.84 Variants.VarEnsureRange Function

Returns a Variant with a value in a specified range.

Pascal

```
function VarEnsureRange(const AValue: Variant; const AMin: Variant; const AMax: Variant):
Variant;
```

C++

```
Variant VarEnsureRange(const Variant AValue, const Variant AMin, const Variant AMax);
```

File

Variants

Description

VarEnsureRange returns a Variant whose value is guaranteed to be within a specified range.

AValue is a Variant with the value you want as long as it is in range. If AValue is in range, VarEnsureRange returns AValue.

AMin is a Variant with the minimum acceptable value. If AValue is less than AMin, VarEnsureRange returns AMin.

AMax is a Variant with the maximum acceptable value. If AValue is greater than AMax, VarEnsureRange returns AMax.

See Also

VarInRange (see page 23654)

171.3.85 Variants.VarFromDateTime Function

Returns a variant containing a specified data and time.

Pascal

```
function VarFromDateTime(const DateTime: TDateTime): Variant;
```

C++

```
Variant VarFromDateTime(const TDateTime DateTime);
```

File

Variants

Description

VarFromDateTime returns a Variant that represents the date and time of the given TDateTime value, DateTime. The type code of the resulting variant is varDate.

See Also

VarToDateTime (see page 23665)

VarType (see page 23667)

171.3.86 Variants.VarInRange Function

Indicates whether a Variant's value is in a specified range.

Pascal

```
function VarInRange(const AValue: Variant; const AMin: Variant; const AMax: Variant): Boolean;
```

C++

```
Boolean VarInRange(const Variant AValue, const Variant AMin, const Variant AMax);
```

File

Variants

Description

VarInRange checks whether a Variant has a value in a specified range.

AValue is a Variant to check.

AMin is a Variant with the minimum acceptable value.

AMax is a Variant with the maximum acceptable value.

VarInRange returns true if AValue is greater than or equal to AMin and less than or equal to AMax. Otherwise, it returns false.

See Also

VarEnsureRange (see page 23653)

171.3.87 Variants.VarInvalidNullOp Function

Indicate variant operation on null.

Pascal

```
procedure VarInvalidNullOp;
```

C++

```
VarInvalidNullOp();
```

File

Variants

See Also

EVariantInvalidOpError (see page 23577)

171.3.88 Variants.VarInvalidOp Function

Indicate invalid variant operation.

Pascal

```
procedure VarInvalidOp;
```

C++

```
VarInvalidOp();
```

File

Variants

See Also

EVariantInvalidOpError (see page 23577)

171.3.89 Variants.VarIsArray Function

Indicates whether the specified variant is an array.

Pascal

```
function VarIsArray(const A: Variant): Boolean; overload;
function VarIsArray(const A: Variant; AResolveByRef: Boolean): Boolean; overload;
```

C++

```
Boolean VarIsArray(const Variant A);
Boolean VarIsArray(const Variant A, Boolean AResolveByRef);
```

File

Variants

Description

VarIsArray returns true if the given variant is an array. Otherwise, the function result is false.

A is the Variant to check.

AResolveByRef indicates how VarIsArray should handle the case when A is a reference to a Variant (when the VarType (see page 23667) of A is varByRef or varVariant).When AResolveByRef is true, VarIsArray checks whether A is an array or A points to an array. When AResolveByRef is false, VarIsArray returns false when A has a VarType (see page 23667) of varByRef or varVariant.

See Also

VarArrayCreate (see page 23644)

VarType (see page 23667)

VarIsType (see page 23662)

171.3.90 Variants.VarIsByRef Function

Indicates whether the specified variant's value is by reference.

Pascal

```
function VarIsByRef(const V: Variant): Boolean;
```

C++

```
Boolean VarIsByRef(const Variant V);
```

File

Variants

Description

VarIsByRef returns true if the given variant's type code includes the includes the varByRef bit. Otherwise, the function result is false.

See Also

VarType (see page 23667)

VarIsType (see page 23662)

171.3.91 Variants.VarIsClear Function

Indicates whether the specified variant has an undefined value.

Pascal

```
function VarIsClear(const V: Variant): Boolean;
```

C++

```
Boolean VarIsClear(const Variant V);
```

File

Variants

Description

VarIsClear returns true if the given variant's value is undefined. The value can be undefined for any of several reasons:

The Variant may have had its value set to Unassigned (see page 23643).

The Variant's value may be an interface type that has been set to nil (Delphi) or NULL (C++).

The Variant may be a custom variant that returns true from its IsClear method.

In all other cases, the function result is false.

Note: Do not confuse an unassigned variant with a Null (see page 23641) variant. A Null (see page 23641) variant is still assigned, but has the value Null (see page 23641). Unlike unassigned variants, Null (see page 23641) variants can be used in expressions and can be converted to other types of variants.

See Also

Unassigned (see page 23643)

VarType (see page 23667)

VarCheckEmpty (see page 23651)

VarIsNull (see page 23660)

VarIsEmpty (see page 23657)

VarIsType (see page 23662)

TCustomVariantType (see page 23581)

171.3.92 Variants.VarIsCustom Function

Indicates whether the specified variant is a custom variant.

Pascal

```
function VarIsCustom(const V: Variant): Boolean;
```

C++

```
Boolean VarIsCustom(const Variant V);
```

File

Variants

Description

VarIsCustom returns true if the given variant represents a custom variant type. Otherwise, the function result is false.

See Also

VarType (see page 23667)

FindCustomVariantType (see page 23640)

VarIsType (see page 23662)

171.3.93 Variants.VarIsEmpty Function

Indicates whether the specified variant is unassigned.

Pascal

```
function VarIsEmpty(const V: Variant): Boolean;
```

C++

```
Boolean VarIsEmpty(const Variant V);
```

File

Variants

Description

VarIsEmpty returns true if the given variant contains the value Unassigned (see page 23643). If the variant contains any other value, the function result is false.

Note: VarIsEmpty does not indicate when a Variant that represents an interface has had the interface pointer cleared or when a custom Variant is clear. If either of these situations may exist, use the VarIsClear (see page 23656) method instead.

Note: Do not confuse an unassigned variant with a Null (see page 23641) variant. A Null (see page 23641) variant is still assigned, but has the value Null (see page 23641). Unlike unassigned variants, Null (see page 23641) variants can be used in expressions and can be converted to other types of variants.

See Also

Unassigned (see page 23643)

VarType (see page 23667)

VarCheckEmpty (see page 23651)

VarIsNull (see page 23660)

VarIsClear (see page 23656)

VarIsEmptyParam (see page 23658)

VarIsType (see page 23662)

171.3.94 Variants.VarIsEmptyParam Function

Indicates whether the specified variant represents an unassigned optional parameter.

Pascal

```
function VarIsEmptyParam(const V: Variant): Boolean;
```

C++

```
Boolean VarIsEmptyParam(const Variant V);
```

File

Variants

Description

VarIsEmptyParam returns true if the given variant represents an unassigned optional parameter. If the variant contains any other value, the function result is false.

See Also

VarIsClear (see page 23656)

VarIsEmpty (see page 23657)

VarIsNull (see page 23660)

EmptyParam (see page 23677)

VarIsType (see page 23662)

171.3.95 Variants.VarIsError Function

Indicates whether the specified variant is an error variant; optionally converts variant back to HRESULT.

Pascal

```
function VarIsError(const V: Variant; out AResult: HRESULT): Boolean; overload;
function VarIsError(const V: Variant): Boolean; overload;
```

C++

```
Boolean VarIsError(const Variant V, HRESULT AResult);
Boolean VarIsError(const Variant V);
```

File

Variants

Description

VarIsError returns true if the given variant is an Error variant (type varError). If AResult is specified, it is set to the corresponding result code.

See Also

VarType (see page 23667)

VarIsType (see page 23662)

171.3.96 Variants.VarIsFloat Function

Indicates whether the specified variant represents a floating-point value.

Pascal

```
function VarIsFloat(const V: Variant): Boolean;
```

C++

```
Boolean VarIsFloat(const Variant V);
```

File

Variants

Description

VarIsFloat returns true if the given variant represents a floating-point value (In Delphi: Single, Double, or Currency; In C++: float, double, or Currency). If the variant contains any other type of value, the function result is false.

See Also

VarIsOrdinal (see page 23661)

VarIsNumeric (see page 23660)

VarIsStr (see page 23661)

VarType (see page 23667)

VarIsType (see page 23662)

171.3.97 Variants.VarIsNull Function

Indicates whether the specified variant is Null (see page 23641).

Pascal

```
function VarIsNull(const V: Variant): Boolean;
```

C++

```
Boolean VarIsNull(const Variant V);
```

File

Variants

Description

VarIsNull returns true if the given variant contains the value Null (see page 23641). If the variant contains any other value, the function result is false.

Note: Do not confuse a Null (see page 23641) variant with an unassigned variant. A Null (see page 23641) variant is still assigned, but has the value Null (see page 23641). Unlike unassigned variants, Null (see page 23641) variants can be used in expressions and can be converted to other types of variants.

See Also

Null (see page 23641)

VarType (see page 23667)

VarIsEmpty (see page 23657)

VarIsClear (see page 23656)

VarIsEmptyParam (see page 23658)

VarIsType (see page 23662)

171.3.98 Variants.VarIsNumeric Function

Indicates whether the specified variant represents a numeric value.

Pascal

```
function VarIsNumeric(const V: Variant): Boolean;
```

C++

```
Boolean VarIsNumeric(const Variant V);
```

File

Variants

Description

VarIsNumeric returns true if the given variant's type code indicates a numeric value of some sort. This can be either a floating-point value or an ordinal value. If the variant contains any other type of value, the function result is false.

See Also

VarIsOrdinal (see page 23661)

VarIsFloat (see page 23659)

VarIsStr (see page 23661)

VarType (see page 23667)

VarIsType (see page 23662)

171.3.99 Variants.VarIsOrdinal Function

Indicates whether the specified variant represents an ordinal value.

Pascal

```
function VarIsOrdinal(const V: Variant): Boolean;
```

C++

```
Boolean VarIsOrdinal(const Variant V);
```

File

Variants

Description

VarIsOrdinal returns true if the given variant represents an ordinal value (such as an integer or boolean type). If the variant contains any other type of value, the function result is false.

See Also

VarIsNumeric (see page 23660)

VarIsFloat (see page 23659)

VarIsStr (see page 23661)

VarType (see page 23667)

VarIsType (see page 23662)

171.3.100 Variants.VarIsStr Function

Indicates whether the specified variant represents a string value.

Pascal

```
function VarIsStr(const V: Variant): Boolean;
```

C++

```
Boolean VarIsStr(const Variant V);
```

File

Variants

Description

VarIsStr returns true if the given variant's type code is varString or varOleStr. If the variant contains any other type of value, the function result is false.

See Also

VarIsNumeric (see page 23660)

VarIsFloat (see page 23659)

VarIsOrdinal (see page 23661)

VarType (see page 23667)

VarIsType (see page 23662)

171.3.101 Variants.VarIsType Function

Indicates whether the specified variant represents a specified type.

Pascal

```
function VarIsType(const V: Variant; AVarType: TVarType): Boolean; overload;
function VarIsType(const V: Variant; const AVarTypes: array of TVarType): Boolean; overload;
```

C++

```
Boolean VarIsType(const Variant V, TVarType AVarType);
Boolean VarIsType(const Variant V, const array of TVarType AVarTypes);
```

File

Variants

Description

VarIsType checks whether the type code for the Variant specified by V matches the specified type code or one of the type codes in the specified array.

V is the Variant to check.

AVarType is the type code against which to check V.

AVarTypes is an array of valid type codes against which to check V.

AVarTypes_size is the index of the last entry in AVarTypes (one less than the number of entries).

VarIsType returns true if the given variant's type code matches the supplied type code or codes, false otherwise.

See Also

VarType (see page 23667)

171.3.102 Variants.VarOverflowError Function

Raises an EVariantOverflowError (see page 23579) exception.

Pascal

```
procedure VarOverflowError(const ASourceType: TVarType; const ADestType: TVarType);
```

C++

```
VarOverflowError(const TVarType ASourceType, const TVarType ADestType);
```

File

Variants

Description

Call VarOverflowError to indicate that a variant operation has failed due to numeric overflow. ASourceType and ADestType should contain the variant type codes for the source and destination variant types. VarOverflowError uses this information to compose an error string and raise EVariantOverflowError (see page 23579).

See Also

EVariantOverflowError (see page 23579)

TVarType

171.3.103 Variants.VarRangeCheckError Function

Raises an EVariantRangeCheckError (see page 23580) exception.

Pascal

```
procedure VarRangeCheckError(const ASourceType: TVarType; const ADestType: TVarType);
```

C++

```
VarRangeCheckError(const TVarType ASourceType, const TVarType ADestType);
```

File

Variants

Description

Call VarRangeCheckError to indicate that a variant operation has failed because the result is not in the allowed range for the destination variant type. ASourceType and ADestType are the variant types of the source and destination.

See Also

EVariantRangeCheckError (see page 23580)

171.3.104 Variants.VarResultCheck Function

Raises variant exception based on HRESULT value.

Pascal

```
procedure VarResultCheck(AResult: HRESULT); overload;
procedure VarResultCheck(AResult: HRESULT; ASourceType: TVarType; ADestType: TVarType);
overload;
```

C++

```
VarResultCheck(HRESULT AResult);
VarResultCheck(HRESULT AResult, TVarType ASourceType, TVarType ADestType);
```

File

Variants

Description

VarResultCheck examines the HRESULT value returned by many variant routines. If AResult is S_OK, VarResultCheck does nothing. Otherwise, VarResultCheck attempts to map AResult to a variant exception and raise that exception. If VarResultCheck is unable to map AResult to a specific variant Exception, it raises EVariantError.

Specify ASourceType and ADestType when the operation involves conversion between two variant types.

See Also

EVariantError

171.3.105 Variants.VarSameValue Function

Indicates whether two Variants (see page 23567) have equivalent values.

Pascal

```
function VarSameValue(const A: Variant; const B: Variant): Boolean;
```

C++

```
Boolean VarSameValue(const Variant A, const Variant B);
```

File

Variants

Description

VarSameValue returns true when A has the same value as B, false if the two values are not equivalent.

If A and B can't be compared, VarSameValue raises an exception.

See Also

VarCompareValue (see page 23652)

SameValue

171.3.106 Variants.VarSupports Function

Indicates whether the Variant's value is an interface that supports a specified interface.

Pascal

```
function VarSupports(const V: Variant; const IID: TGUID; out Intf): Boolean; overload;
function VarSupports(const V: Variant; const IID: TGUID): Boolean; overload;
```

C++

```
Boolean VarSupports(const Variant V, const TGUID IID,  Intf);
Boolean VarSupports(const Variant V, const TGUID IID);
```

File

Variants

Description

VarSupports returns the lower twelve bits of a variant type code. These indicate the type of the variant. For a list of possible variant type codes, see VarType (see page 23667).

Call VarSupports to determine whether the Variant represents an interface that supports the interface identified by the IID parameter. If V supports the interface, VarSupports returns the interface as the Intf parameter (if using the first syntax) and returns true (both versions). If the interface specified by IID is not supported, VarSupports returns false.

Unlike calling the QueryInterface method of the interface, VarSupports handles the case when the Variant's value is nil (Delphi) or NULL (C++).

See Also

Supports

171.3.107 Variants.VarToDateTime Function

Converts a specified variant to a TDateTime value.

Pascal

```
function VarToDateTime(const V: Variant): TDateTime;
```

C++

```
TDateTime VarToDateTime(const Variant V);
```

File

Variants

Description

VarToDateTime converts the given variant to a TDateTime value. VarToDateTime raises an exception if the conversion fails.

See Also

VarFromDateTime (see page 23653)

171.3.108 Variants.VarToStr Function

Converts a variant's value to a string.

Pascal

```
function VarToStr(const V: Variant): string;
```

C++

```
AnsiString VarToStr(const Variant V);
```

File

Variants

Description

VarToStr converts the data in the variant V to a string and returns the result. If the variant has a null value, VarToStr returns an empty string.

See Also

VarToStrDef (see page 23666)

VarToWideStr (see page 23666)

VarToWideStrDef (see page 23667)

171.3.109 Variants.VarToStrDef Function

Converts a variant's value to a string.

Pascal

```
function VarToStrDef(const V: Variant; const ADefault: string): string;
```

C++

```
AnsiString VarToStrDef(const Variant V, const AnsiString ADefault);
```

File

Variants

Description

VarToStrDef converts the data in the variant V to a string and returns the result. If the variant has a null value, VarToStrDef returns ADefault.

See Also

VarToStr (see page 23665)

VarToWideStr (see page 23666)

VarToWideStrDef (see page 23667)

171.3.110 Variants.VarToWideStr Function

Converts a variant's value to a WideString.

Pascal

```
function VarToWideStr(const V: Variant): WideString;
```

C++

```
BSTR VarToWideStr(const Variant V);
```

File

Variants

Description

VarToWideStr converts the data in the variant V to a WideString and returns the result. If the variant has a null value, VarToWideStr returns an empty string.

See Also

VarToStrDef (see page 23666)

VarToStr (see page 23665)

VarToWideStrDef (see page 23667)

171.3.111 Variants.VarToWideStrDef Function

Converts a variant's value to a WideString.

Pascal

```
function VarToWideStrDef(const V: Variant; const ADefault: WideString): WideString;
```

C++

```
BSTR VarToWideStrDef(const Variant V, const BSTR ADefault);
```

File

Variants

Description

VarToWideStrDef converts the data in the variant V to a WideString and returns the result. If the variant has a null value, VarToWideStrDef returns ADefault.

See Also

VarToStr (see page 23665)

VarToWideStr (see page 23666)

VarToStrDef (see page 23666)

171.3.112 Variants.VarType Function

Returns the type code of a specified variant.

Pascal

```
function VarType(const V: Variant): TVarType;
```

C++

```
TVarType VarType(const Variant V);
```

File

Variants

Description

VarType returns the type code of the given variant. The resulting value is either the type code for a custom Variant type, or it is constructed from the constants declared in the System (see page 22079) unit.

The lower twelve bits of a variant type code (the bits defined by the varTypeMask bit mask) define the type of the variant. The varArray bit is set if the variant is an array of the given type. The varByRef bit is set if the variant is a reference to a value of the given type as opposed to an actual value.

The following table describes the meaning of each of the variant type codes defined in the System (see page 22079) unit:

VarType	Contents of variant
varEmpty	The variant is Unassigned (see page 23643).
varNull	The variant is Null (see page 23641).
varSmallint	16-bit signed integer (type Smallint in Delphi, short in C++).
varInteger	32-bit signed integer (type Integer in Delphi, int in C++).
varSingle	Single-precision floating-point value (type Single in Delphi, float in C++).
varDouble	Double-precision floating-point value (type double).
varCurrency	Currency floating-point value (type Currency).
varDate	Date and time value (type TDateTime).
varOleStr	Reference to a dynamically allocated UNICODE string.
varDispatch	Reference to an Automation object (an IDispatch interface pointer).
varError	Operating system error code.
varBoolean	16-bit boolean (type WordBool).
varVariant	A variant.
varUnknown	Reference to an unknown object (an IInterface or IUnknown interface pointer).
varShortInt	8-bit signed integer (type ShortInt in Delphi or signed char in C++)
varByte	A Byte
varWord	unsigned 16-bit value (Word)
varLongWord	unsigned 32-bit value (type LongWord in Delphi or unsigned long in C++)
varInt64	64-bit signed integer (Int64 in Delphi or __int64 in C++)
varStrArg	COM-compatible string.
varString	Reference to a dynamically allocated string (not COM compatible).

The value returned by VarType corresponds to the VType field of a TVarData record.

The type of a variant can be changed using VarAsType (see page 23650).

See Also

TVarData

VarAsType (see page 23650)

VarTypeToDataType

VarTypeAsText (see page 23668)

171.3.113 Variants.VarTypeAsText Function

Returns variant type descriptive string.

Pascal

```
function VarTypeAsText(const AType: TVarType): string;
```

C++

```
AnsiString VarTypeAsText(const TVarType AType);
```

File

Variants

Description

VarTypeAsText returns a short string that names the variant type indicated by AType.

See Also

TVarData

VarAsType (see page 23650)

VarTypeToDataType

VarType (see page 23667)

171.3.114 Variants.VarTypeIsValidArrayType Function

Indicates whether a Variant type code represents a type that can be used for the elements in a Variant array.

Pascal

```
function VarTypeIsValidArrayType(const AVarType: TVarType): Boolean;
```

C++

```
Boolean VarTypeIsValidArrayType(const TVarType AVarType);
```

File

Variants

Description

VarTypeIsValidArrayType indicates whether AVarType is the type code for a Variant type that can represent the elements of a Variant array. This function is only intended for use with built-in Variant types. It returns false for custom Variant types, although they can be used for the elements of Variant arrays.

See Also

VarArrayCreate (see page 23644)

VarTypeIsValidElementType (see page 23669)

171.3.115 Variants.VarTypeIsValidElementType Function

Indicates whether a Variant type code represents a type that can be used for the elements in a Variant array.

Pascal

```
function VarTypeIsValidElementType(const AVarType: TVarType): Boolean;
```

C++

```
Boolean VarTypeIsValidElementType(const TVarType AVarType);
```

File

Variants

Description

VarTypeIsValidElementType indicates whether AVarType is the type code for a Variant type that can represent the elements of a Variant array. Unlike the VarTypeIsValidArrayType (see page 23669) function, this function returns true for custom Variant types.

See Also

VarArrayCreate (see page 23644)

VarTypeIsValidArrayType (see page 23669)

171.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TBooleanToStringRule (see page 23671)	TBooleanToStringRule enumerates possible rules for BooleanToStringRule (see page 23676).
	TNullCompareRule (see page 23671)	TNullCompareRule enumerates possible rules for comparing variant nulls with other values.
	TVarCompareResult (see page 23672)	This is record Variants.TVarCompareResult.
	TVariantRelationship (see page 23672)	TVariantRelationship describes the results of comparing two Variants (see page 23567).

171.4.1 Variants.TBooleanToStringRule Enumeration

TBooleanToStringRule enumerates possible rules for BooleanToStringRule (see page 23676).

Pascal

```
TBooleanToStringRule = (
  bsrAsIs,
  bsrLower,
  bsrUpper
);
```

C++

```
enum TBooleanToStringRule {
  bsrAsIs,
  bsrLower,
  bsrUpper
};
```

File

Variants

Description

TBooleanToStringRule enumerates the values that can be assigned to the global variables NullEqualityRule (see page 23678) and NullMagnitudeRule (see page 23679).

171.4.2 Variants.TNullCompareRule Enumeration

TNullCompareRule enumerates possible rules for comparing variant nulls with other values.

Pascal

```
TNullCompareRule = (
  ncrError,
  ncrStrict,
  ncrLoose
);
```

C++

```
enum TNullCompareRule {
  ncrError,
  ncrStrict,
  ncrLoose
};
```

File

Variants

Description

TNullCompareRule enumerates the values that can be assigned to the global variables NullEqualityRule (see page 23678) and NullMagnitudeRule (see page 23679).

171.4.3 Variants.TVarCompareResult Enumeration

Pascal

```
TVarCompareResult = (
  crLessThan,
  crEqual,
  crGreaterThan
);
```

C++

```
enum TVarCompareResult {
  crLessThan,
  crEqual,
  crGreaterThan
};
```

File

Variants

Description

This is record Variants.TVarCompareResult.

171.4.4 Variants.TVariantRelationship Enumeration

TVariantRelationship describes the results of comparing two Variants (see page 23567).

Pascal

```
TVariantRelationship = (
  vrEqual,
  vrLessThan,
  vrGreaterThan,
  vrNotEqual
);
```

C++

```
enum TVariantRelationship {
  vrEqual,
  vrLessThan,
```

```
    vrGreaterThan,
    vrNotEqual
  };
```

File

Variants

Description

TVariantRelationship describes the relationship between two Variant values. The following table lists the possible values:

Value	Meaning
vrEqual	A is equivalent to B. Note that two NULL Variants (see page 23567) are considered equal, and two empty Variants (see page 23567) are considered equal, but a NULL Variant does not equal an empty Variant.
vrLessThan	A is less than B.
vrGreaterThan	A is larger than B.
vrNotEqual	A and B are not equal, but the values are such that they can't be ordered relative to each other. For example, this describes the relationship when either A or B is empty or NULL, and the other one is not.

See Also

VarCompareValue

171.5 Types

The following table lists types in this documentation.

Types

Name	Description
TAnyProc (see page 23674)	This is type Variants.TAnyProc.
TCustomVariantTypeClass (see page 23674)	This is type Variants.TCustomVariantTypeClass.
TVarDataArray (see page 23675)	This is type Variants.TVarDataArray.
TVarDispProc (see page 23675)	This is type Variants.TVarDispProc.

171.5.1 Variants.TAnyProc Type

Pascal

```
TAnyProc = procedure (var V: TVarData);
```

C++

```
(var V: TVarData) ( TAnyProc)();
```

File

Variants

Description

This is type Variants.TAnyProc.

171.5.2 Variants.TCustomVariantTypeClass Type

Pascal

```
TCustomVariantTypeClass = class of TCustomVariantType;
```

C++

```
TCustomVariantType TCustomVariantTypeClass;
```

File

Variants

Description

This is type Variants.TCustomVariantTypeClass.

171.5.3 Variants.TVarDataArray Type

Pascal

```
TVarDataArray = array of TVarData;
```

C++

```
array of TVarData TVarDataArray;
```

File

Variants

Description

This is type Variants.TVarDataArray.

171.5.4 Variants.TVarDispProc Type

Pascal

```
TVarDispProc = procedure (Dest: PVariant; const Source: Variant; CallDesc: PCallDesc; Params:
Pointer);
```

C++

```
(Dest: PVariant; const Source: Variant; CallDesc: PCallDesc; Params: Pointer) (
TVarDispProc)();
```

File

Variants

Description

This is type Variants.TVarDispProc.

171.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
BooleanToStringRule (see page 23676)	Specifies rule fixing up Boolean values converted to strings.
ChangeAnyProc (see page 23677)	This is variable Variants.ChangeAnyProc.
ClearAnyProc (see page 23677)	This is variable Variants.ClearAnyProc.
EmptyParam (see page 23677)	Contains an OleVariant that represents an unused optional parameter on a dual interface.
NullAsStringValue (see page 23678)	Specifies string value of Null (see page 23641) variants.
NullEqualityRule (see page 23678)	Specifies rule for Null (see page 23641) variant equality comparison.
NullMagnitudeRule (see page 23679)	Specifies rule for Null (see page 23641) variant magnitude comparison.
NullStrictConvert (see page 23679)	Specifies whether Null (see page 23641) conversion is an error.
OleVariantInt64AsDouble (see page 23680)	This is variable Variants.OleVariantInt64AsDouble.
PackVarCreation (see page 23680)	Specifies whether space-efficient Variant Types (see page 23293) are preferred.
RefAnyProc (see page 23681)	This is variable Variants.RefAnyProc.
VarDispProc (see page 23681)	Provides the support for IDispatch-based Automation.

171.6.1 Variants.BooleanToStringRule Variable

Specifies rule fixing up Boolean values converted to strings.

Pascal

```
BooleanToStringRule: TBooleanToStringRule = bsrAsIs;
```

C++

```
TBooleanToStringRule BooleanToStringRule = bsrAsIs;
```

File

Variants

Description

BooleanToStringRule determines the rule for fixing up the string generated by a conversion from Boolean to string.

Constant	Meaning
bsrAsIs	(Default.) The value obtained from the locale is left as is.
bsrUpper	The string is converted to all upper case.
bsrLower	The string is converted to all lower case.

171.6.2 Variants.ChangeAnyProc Variable

Pascal

```
ChangeAnyProc: TAnyProc;
```

C++

```
TAnyProc ChangeAnyProc;
```

File

Variants

Description

This is variable Variants.ChangeAnyProc.

171.6.3 Variants.ClearAnyProc Variable

Pascal

```
ClearAnyProc: TAnyProc;
```

C++

```
TAnyProc ClearAnyProc;
```

File

Variants

Description

This is variable Variants.ClearAnyProc.

171.6.4 Variants.EmptyParam Variable

Contains an OleVariant that represents an unused optional parameter on a dual interface.

Pascal

```
EmptyParam: OleVariant;
```

C++

```
OleVariant EmptyParam;
```

File

Variants

Description

EmptyParam can be used for Variants (see page 23567) that represent optional parameters whose value is unassigned. This is useful when the code that marshals method calls requires a fixed number of parameters, even if some of them are optional.

When marshaling interface calls that include optional parameters, COM requires a value for those parameters even when they are not used. EmptyParam returns an OleVariant you can pass as that value to indicate that the parameter is not used.

Marshaling interfaces also exist under Linux, but we can't spare the resources to document them right now.

See Also

VarIsEmptyParam (see page 23658)

171.6.5 Variants.NullAsStringValue Variable

Specifies string value of Null (see page 23641) variants.

Pascal

```
NullAsStringValue: string = '';
```

C++

```
AnsiString NullAsStringValue = '';
```

File

Variants

Description

NullAsStringValue specifies the string that is the result of a conversion from a Null (see page 23641) variant to a string type. By default, NullAsStringValue is the empty string.

NullAsStringValue is only used when the NullStrictConvert (see page 23679) global variable is set to false.

See Also

NullStrictConvert (see page 23679)

Null (see page 23641)

171.6.6 Variants.NullEqualityRule Variable

Specifies rule for Null (see page 23641) variant equality comparison.

Pascal

```
NullEqualityRule: TNullCompareRule = ncrLoose;
```

C++

```
TNullCompareRule NullEqualityRule = ncrLoose;
```

File

Variants

Description

NullEqualityRule determines the rule for handling equality comparisons (equals or not-equals) when one of the values compared is a variant containing a null value. NullEqualityRule can have the following values:

Constant	Meaning
ncrLoose	(Default.) A variant containing a null is treated as a unique value that is only equal to another variant containing a null.
ncrStrict	An equality comparison of a variant containing a null is always false, regardless of the other value.

ncrError	An equality comparison raises an EVariantInvalidNullOpError (see page 23576) if either value is a variant containing a null.

See Also

Null (see page 23641)

NullMagnitudeRule (see page 23679)

NullStrictConvert (see page 23679)

EVariantInvalidNullOpError (see page 23576)

171.6.7 Variants.NullMagnitudeRule Variable

Specifies rule for Null (see page 23641) variant magnitude comparison.

Pascal

```
NullMagnitudeRule: TNullCompareRule = ncrLoose;
```

C++

```
TNullCompareRule NullMagnitudeRule = ncrLoose;
```

File

Variants

Description

NullMagnitudeRule determines the rule for handling magnitude comparisons (one value is greater than, greater than or equals, less than, or less than or equals to another) when one of the values compared is a variant containing a null value. NullMagnitudeRule can have the following values:

Constant	Meaning
ncrLoose	(Default.) A variant containing a null is treated as a unique value that is less than any other value, but equal to another variant containing a null.
ncrStrict	A magnitude comparison of a variant containing a null is always false, regardless of the other value.
ncrError	A magnitude comparison raises an EVariantInvalidNullOpError (see page 23576) if either value is a variant containing a null.

See Also

Null (see page 23641)

NullMagnitudeRule

NullStrictConvert (see page 23679)

EVariantInvalidNullOpError (see page 23576)

171.6.8 Variants.NullStrictConvert Variable

Specifies whether Null (see page 23641) conversion is an error.

Pascal

```
NullStrictConvert: Boolean = True;
```

C++

```
Boolean NullStrictConvert = True;
```

File

Variants

Description

NullStrictConvert determines the outcome of attempts to convert Null (see page 23641) variants to other types. If NullStrictConvert is true (default), attempting to convert a Null (see page 23641) variant raises a EVariantTypeCastError (see page 23580), unless the conversion is to a custom variant that defines a conversion from Null (see page 23641). If NullStrictConvert is false, then conversion from Null (see page 23641) follows the following rules:

Destination	Conversion result
Numeric type	Zero
String type	Value of NullAsStringValue (see page 23678) global variable. (Default: empty string.)
Boolean	False.

See Also

TVarData

VarType (see page 23667)

NullAsStringValue (see page 23678)

EVariantTypeCastError (see page 23580)

171.6.9 Variants.OleVariantInt64AsDouble Variable

Pascal

```
OleVariantInt64AsDouble: Boolean = False;
```

C++

```
Boolean OleVariantInt64AsDouble = False;
```

File

Variants

Description

This is variable Variants.OleVariantInt64AsDouble.

171.6.10 Variants.PackVarCreation Variable

Specifies whether space-efficient Variant Types (see page 23293) are preferred.

Pascal

```
PackVarCreation: Boolean = True;
```

C++

```
Boolean PackVarCreation = True;
```

File

Variants

Description

PackVarCreation controls the way Variant Types (see page 23293) are chosen for newly-created Variants (see page 23567). If PackVarCreation is true (default), a new Variant has the smallest possible Variant Type. If PackVarCreation is false, a new Variant has the Variant Type that minimizes execution time.

See Also

TVarData

VarType (see page 23667)

171.6.11 Variants.RefAnyProc Variable

Pascal

```
RefAnyProc: TAnyProc;
```

C++

```
TAnyProc RefAnyProc;
```

File

Variants

Description

This is variable Variants.RefAnyProc.

171.6.12 Variants.VarDispProc Variable

Provides the support for IDispatch-based Automation.

Pascal

```
VarDispProc: TVarDispProc;
```

C++

```
TVarDispProc VarDispProc;
```

File

Variants

Description

In Delphi, VarDispProc points to a routine that is called to execute methods (or access properties) of an interface that is assigned to a Variant. The Variants (see page 23567) unit assigns a procedure that generates error 222 or raises an

EVariantDispatchError (see page 23575) exception. The ComObj (see page 5525) unit reassigns this variable to a procedure that generates the appropriate COM IDispatch calls.

To control how Variants (see page 23567) execute methods when they have a value that is an interface, assign your own procedure to VarDispProc.

The Linux implementation of IDispatch is currently undocumented, due to resource constraints. When that changes, decolorize and revise this topic.

See Also

EVariantDispatchError (see page 23575)

172 VCLCom Namespace

This is namespace VCLCom.

Topics

Name	Description
Classes (see page 23684)	The following table lists classes in this documentation.

Classes

	Name	Description
	TComponentFactory (see page 23684)	TComponentFactory is the class factory for creating instances of components that support interfaces.

172.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TComponentFactory (see page 23684)	TComponentFactory is the class factory for creating instances of components that support interfaces.

172.1.1 TComponentFactory Class

TComponentFactory is the class factory for creating instances of components that support interfaces.

Class Hierarchy

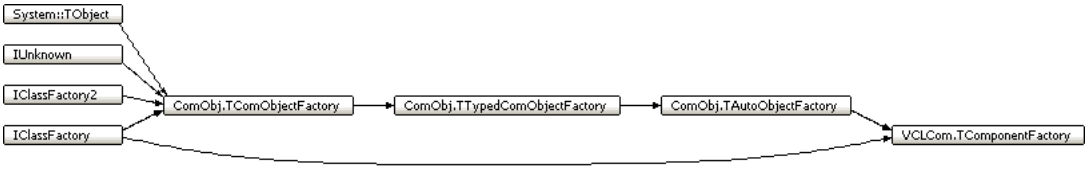

Pascal

```
TComponentFactory = class(TAutoObjectFactory, IClassFactory);
```

C++

```
class TComponentFactory : public TAutoObjectFactory, public IClassFactory;
```

File

VCLCom

Description

TComponentFactory creates instances of component classes that implement interfaces. TComponentFactory supports the IClassFactory interface and inherits support for the IClassFactory2 interface.

Note: These components are actually created as contained objects of an automation object (aggregate).

172.1.1.1 TComponentFactory Members

The following tables list the members exposed by TComponentFactory.

Protected Methods

	Name	Description
	_AddRef (see page 5567)	This is _AddRef, a member of class TComObjectFactory.
	_Release (see page 5567)	This is _Release, a member of class TComObjectFactory.
	CreateInstance (see page 23687)	This is CreateInstance, a member of class TComponentFactory.
	CreateInstanceLic (see page 5568)	This is CreateInstanceLic, a member of class TComObjectFactory.
V	GetLicenseString (see page 5568)	This is GetLicenseString, a member of class TComObjectFactory.

	GetLicInfo (see page 5568)	This is GetLicInfo, a member of class TComObjectFactory.
V	GetProgID (see page 5569)	This is GetProgID, a member of class TComObjectFactory.
V	HasMachineLicense (see page 5569)	This is HasMachineLicense, a member of class TComObjectFactory.
	LockServer (see page 5569)	This is LockServer, a member of class TComObjectFactory.
	QueryInterface (see page 5569)	This is QueryInterface, a member of class TComObjectFactory.
	RequestLicKey (see page 5570)	This is RequestLicKey, a member of class TComObjectFactory.
V	ValidateUserLicense (see page 5571)	This is ValidateUserLicense, a member of class TComObjectFactory.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 23686)	Instantiates an instance of a TComponentFactory (see page 23684) object.
V	CreateComObject (see page 23686)	Instantiates a TComObject object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 5566)	Disposes of an instance of a class factory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetInterfaceTypeInfo (see page 5586)	Retrieves the type information for the typed COM object created by the factory.
V	GetIntfEntry (see page 5548)	Returns the entry for a specific interface implemented in a class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RegisterClassObject (see page 5569)	Registers the COM class associated with the factory object.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	UpdateRegistry (see page 23687)	Registers the typed COM object class.

Public Properties

	Name	Description
R	ClassID (see page 5571)	This is ClassID, a member of class TComObjectFactory.
R	ClassInfo (see page 5587)	Specifies an interface reference that provides type information without having to load a type library.
R	ClassName (see page 5572)	Specifies the name of the class that the class factory instantiates.
R	ComClass (see page 5572)	This is ComClass, a member of class TComObjectFactory.
R	ComServer (see page 5572)	This is ComServer, a member of class TComObjectFactory.

R	Description (see page 5572)	Describes the class object.
R	DispIntfEntry (see page 5549)	Contains the entry for the dispatch interface implemented by the OLE Automation object created by the factory.
R	DispTypeInfo (see page 5549)	Returns an ITypeInfo interface reference to the type information for the dispinterface implemented by the OLE Automation object.
	ErrorIID (see page 5573)	Indicates the interface of the CoClass in which an error occurred.
R	EventIID (see page 5549)	This is EventIID, a member of class TAutoObjectFactory.
R	EventTypeInfo (see page 5550)	This is EventTypeInfo, a member of class TAutoObjectFactory.
R	Instancing (see page 5573)	Indicates the process space of COM object instances.
	LicString (see page 5573)	Contains licensing information about the object.
R	ProgID (see page 5574)	Specifies the name of the class factory.
	ShowErrors (see page 5574)	Indicates whether the COM object factory displays a message box when exceptions are raised.
	SupportsLicensing (see page 5574)	Indicates whether the object being instantiated supports licensing.
R	ThreadingModel (see page 5575)	Indicates the type of thread support provided by objects that the factory creates.

172.1.1.2 TComponentFactory.Create Constructor

Instantiates an instance of a TComponentFactory (see page 23684) object.

Pascal

```
constructor Create(ComServer: TComServerObject; ComponentClass: TComponentClass; const
ClassID: TGUID; Instancing: TClassInstancing; ThreadingModel: TThreadingModel = tmSingle);
```

C++

```
__fastcall TComponentFactory(TComServerObject ComServer, TComponentClass ComponentClass, const
TGUID ClassID, TClassInstancing Instancing, TThreadingModel ThreadingModel = tmSingle);
```

Description

Create allocates memory for a TComponentFactory (see page 23684) object, and calls the inherited Create, passing its parameters in the same order. The ComponentClass parameter is passed cast as a TAutoClass and passed to the AutoClass parameter.

See Also

Create

172.1.1.3 TComponentFactory Methods

The methods of the TComponentFactory class are listed here.

Protected Methods

	Name	Description
	CreateInstance (see page 23687)	This is CreateInstance, a member of class TComponentFactory.

Public Methods

	Name	Description
V	CreateComObject (see page 23686)	Instantiates a TComObject object.
V	UpdateRegistry (see page 23687)	Registers the typed COM object class.

172.1.1.3.1 TComponentFactory.CreateComObject Method

Instantiates a TComObject object.

Pascal

```
function CreateComObject(const Controller: IUnknown): TComObject; override;
```

C++

```
virtual __fastcall TComObject CreateComObject(const IUnknown Controller);
```

Description

CreateComObject is the method that instantiates the class type for which TComObjectFactory is the class factory. CreateComObject calls the CreateFromFactory constructor of the TComObject class to create a COM object. CreateComObject passes itself as the Factory parameter to CreateFromFactory.

CreateComClass creates a COM object based upon the value of the ComClass property. ComClass is the class identifier (CLSID) used by other applications to identify the COM class for which they request an object instance.

See Also

comclass (see page 5572)

CreateFromFactory (see page 5556)

TComObject (see page 5553)

172.1.1.3.2 TComponentFactory.CreateInstance Method

Pascal

```
function CreateInstance(const UnkOuter: IUnknown; const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT CreateInstance(const IUnknown UnkOuter, const TGUID IID,  Obj);
```

Description

This is CreateInstance, a member of class TComponentFactory.

172.1.1.3.3 TComponentFactory.UpdateRegistry Method

Registers the typed COM object class.

Pascal

```
procedure UpdateRegistry(Register: Boolean); override;
```

C++

```
virtual __fastcall UpdateRegistry(Boolean Register);
```

Description

UpdateRegistry is called for EXEs when the EXE is run, and for DLLs or EXEs when the routines are called to specifically register or unregister the server.

See Also

DLLRegisterServer

DLLUnregisterServer

RegisterClassObject (see page 5569)

173 WebAdapt Namespace

This is namespace WebAdapt.

Topics

Name	Description
Classes (see page 23694)	The following table lists classes in this documentation.
Interfaces (see page 24566)	The following table lists interfaces in this documentation.
Functions (see page 24589)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 24593)	The following table lists structs, records, enums in this documentation.
Types (see page 24595)	The following table lists types in this documentation.
Variables (see page 24612)	The following table lists variables in this documentation.
Constants (see page 24613)	The following table lists constants in this documentation.

Classes

	Name	Description
	EAdapterException (see page 23696)	EAdapterException is raised by WebSnap adapter objects.
	EAdapterFieldException (see page 23698)	EAdapterFieldException is an exception class for errors by adapters using fields.
	TAbstractFieldValuesAdapter (see page 23701)	tabstractfieldvaluesadapter is the base class for adapters that have a single default field.
	TAdapter (see page 23713)	TAdapter defines a scriptable interface to event handlers that return data items or execute commands in your application.
	TAdapterAction (see page 23726)	TAdapterAction is a generic adapter action that relies on event handlers.
	TAdapterActions (see page 23738)	TAdapterActions manages the list of actions that belong to an adapter.
	TAdapterApplicationField (see page 23745)	TAdapterApplicationField is a base class for adapter field components that are children of a TApplicationAdapter (see page 24032) component.
	TAdapterApplicationTitleField (see page 23752)	TAdapterApplicationTitleField is an adapter field used to access the application title.
	TAdapterBooleanField (see page 23758)	TAdapterBooleanField is an event handler based adapter field that works with a Boolean value.
	TAdapterDefaultActions (see page 23770)	TAdapterDefaultActions manages the list of actions that belong to an adapter that supports a default set of actions.
	TAdapterDefaultFields (see page 23778)	TAdapterDefaultFields manages the list of fields that belong to an adapter that supports a default set of fields.
	TAdapterEndUserDisplayNameField (see page 23787)	TAdapterEndUserDisplayNameField is an adapter field used to access the end-user's name.
	TAdapterEndUserField (see page 23793)	TAdapterEndUserField is a base class for adapter field components that are children of an end-user adapter component.
	TAdapterEndUserLoggedInField (see page 23800)	TAdapterEndUserLoggedInField is an adapter field used to access the end-user logged in flag.

	TAdapterError (see page 23807)	TAdapterError is the implementation of IAdapterError.
	TAdapterErrorsList (see page 23810)	TAdapterErrorsList keeps track of errors that an adapter encounters.
	TAdapterField (see page 23818)	TAdapterField is an event handler based adapter field that provides a single value.
	TAdapterFields (see page 23830)	TAdapterFields manages the list of fields that belong to an adapter.
	TAdapterFileField (see page 23837)	TAdapterFileField is an event handler based adapter field that can accept uploaded files.
	TAdapterGotoPageAction (see page 23849)	TAdapterGotoPageAction is an adapter action that supports commands to go to a page of a TDefaultFieldsPagedAdapter (see page 24403).
	TAdapterHiddenField (see page 23860)	This is class WebAdapt.TAdapterHiddenField.
	TAdapterHiddenFieldsList (see page 23862)	TAdapterHiddenFieldsList maintains a collection of name/value pairs that an adapter includes as hidden fields in an HTML form.
	TAdapterImageField (see page 23869)	TAdapterImageField is an event handler based adapter field that can provide the URL of an image file and also accept an uploaded image file.
	TAdapterLoginField (see page 23882)	TAdaperLoginField is a base class for adapter field components that are children of a TLoginFormAdapter (see page 24500) component.
	TAdapterMemoField (see page 23889)	TAdapterMemoField is an event handler based adapter field that supports a multi-line text value.
	TAdapterMultiValueField (see page 23900)	TAdapterMultiValueField is an event handler based adapter field that provides zero or more values.
	TAdapterNamedDisplayField (see page 23913)	The AdapterNamedDisplayField is a base class for adapter fields.
	TAdapterNamedField (see page 23925)	TAdapterNamedField is a based class for adapter field components.
	TAdapterNextPageAction (see page 23945)	TAdapterNextPageAction implements the next page command for TDefaultFieldsPagedAdapter (see page 24403).
	TAdapterNextPageField (see page 23957)	The TAdapterNextPageField navigates to the next page of data.
	TAdapterObjectErrorsList (see page 23968)	This is class WebAdapt.TAdapterObjectErrorsList.
	TAdapterPasswordField (see page 23971)	TAdapterPasswordField is an adapter field used to access the password entered by the end-user.
	TAdapterPrevPageAction (see page 23979)	TAdapterPrevPageAction implements the previous page command for TDefaultFieldsPagedAdapter (see page 24403).
	TAdapterSubComponent (see page 23991)	TAdapterSubComponent is the base class for objects that represent a collection of actions or fields that belong to an adapter.
	TAdapterUpdateField (see page 24007)	TAdapterUpdateField is a base class for adapter field components that support data entry into an HTML form.
	TAdapterUserNameField (see page 24024)	TAdapterUserNameField is an adapter field used to access the user name entered by the end-user.
	TApplicationAdapter (see page 24032)	TApplicationAdapter is a container for field and action components that are available through the Application script variable.
	TBaseAdapterAction (see page 24043)	TBaseAdapterAction is a based class for adapter action components.
	TBaseAdapterErrorsList (see page 24077)	TBaseAdapterErrorsList is the base class for adapter error object managers.
	TBaseAdapterMultiValueField (see page 24086)	TBaseAdapterMultiValueField is a base class for adapter fields that have zero or more values.
	TBaseEndUserAdapter (see page 24097)	TBaseEndUserAdapter contains user information.
	TBaseValuesListAdapter (see page 24120)	TBaseValuesListAdapter is a base class for adapters that provide a list of name/value pairs.
	TBaseValuesListField (see page 24132)	TBaseValuesListField is a base class for adapter field components that are children of a values list adapter component.
	TCustomAdapter (see page 24139)	TCustomAdapter is a base class for adapter components.
	TCustomAdapterAction (see page 24188)	TCustomAdapterAction is a base class for adapter actions.
	TCustomAdapterBooleanField (see page 24201)	tcustomadapterbooleanfield is a base class for adapter fields that provide a single Boolean value.
	TCustomAdapterField (see page 24211)	TCustomAdapterField is a base class for adapter fields that provide a single value.
	TCustomAdapterFileField (see page 24220)	TCustomAdapterFileField is a base class for adapter fields that can have associated files.
	TCustomAdapterGotoPageAction (see page 24233)	TCustomAdapterGotoPageAction is an adapter action that supports commands to go to a page of a TDefaultFieldsPagedAdapter (see page 24403).
	TCustomAdapterImageField (see page 24244)	TCustomAdapterImageField is a base class for adapter fields that have an associated image file.
	TCustomAdapterMemoField (see page 24262)	TCustomAdapterMemoField is a base class for adapter fields that have a multi-line text value.

	TCustomAdapterMultiValueField (see page 24273)	TCustomAdapterMultiValueField is a base class for adapter fields that have one or more values.
	TCustomAdapterPageAction (see page 24286)	TCustomPagedAdapterAction (see page 24371) is a base class for adapter actions that execute the next or previous page commands.
	TCustomApplicationAdapter (see page 24296)	TCustomApplicationAdapter is a base class for components that implement the server-side script Application object.
	TCustomApplicationAdapterAction (see page 24306)	TCustomApplicationAdapterAction is a base class for adapter actions that work with descendants of TCustomApplicationAdapter (see page 24296).
	TCustomEndUserAdapter (see page 24313)	TCustomEndUserAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TCustomEndUserAdapterAction (see page 24326)	TCustomEndUserAdapterAction is a base class for adapter actions that work with descendants of TCustomEndUserAdapter (see page 24313).
	TCustomEndUserSessionAdapter (see page 24333)	TCustomEndUserSessionAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TCustomLoginFormAdapter (see page 24344)	TCustomLoginFormAdapter is the base class for adapters that manage the fields and actions of a login form.
	TCustomLoginFormAdapterAction (see page 24363)	TCustomEndUserAdapterAction (see page 24326) is a base class for adapter actions that work with descendants of TCustomLoginFormAdapter (see page 24344).
	TCustomPagedAdapter (see page 24370)	TCustomPagedAdapter is a base class for adapters that support paging.
	TCustomPagedAdapterAction (see page 24371)	TCustomPagedAdapterAction is a base class for adapter actions that work with descendants of TDefaultFieldsPagedAdapter (see page 24403).
	TCustomStringsValuesList (see page 24378)	TCustomStringsValuesList is a base class for a values list that defines it's name/value pairs using a string list.
	TDefaultFieldsAdapter (see page 24392)	TDefaultFieldsAdapter is a base class for adapters that automatically adds default fields and/or actions.
	TDefaultFieldsPagedAdapter (see page 24403)	TDefaultFieldsPagedAdapter is a base class for adapters that support paging.
	TEndUserAdapter (see page 24421)	TEndUserAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TEndUserAdapterLoginFormAction (see page 24436)	TEndUserAdapterLoginFormAction is an adapter action that displays a login form when executed.
	TEndUserAdapterLogoutAction (see page 24447)	TEndUserAdapterLogoutAction is an adapter action that logs the end-user out.
	TEndUserSessionAdapter (see page 24458)	TEndUserSessionAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TFieldValuesAdapter (see page 24471)	TFieldValuesAdapter is an adapter that includes a single field value.
	TFieldValuesValueField (see page 24483)	TFieldValuesValueField is a adapter field child of TFieldValuesAdapter (see page 24471).
	TImplementedAdapterAction (see page 24491)	TImplementedAdapterAction is a base class for adapter actions.
	TLogicalPageList (see page 24492)	This is class WebAdapt.TLogicalPageList.
	TLoginFormAdapter (see page 24500)	TLoginFormAdapter contains the adapter fields and adapter actions used to generate a login form.
	TLoginFormAdapterLoginAction (see page 24514)	TLoginFormAdapterLoginAction executes the login command of a login form generated with TLoginFormAdapter (see page 24500).
	TPagedAdapter (see page 24525)	TPagedAdapter divides a large amount of data into several pages, each page displaying a portion of the data records.
	TStringsValuesList (see page 24540)	TStringsValuesList is an adapter component that provides a list of name/value pairs defined by items in a TStrings property.
	TUpdateFileList (see page 24550)	TUpdateFileList is a container for uploaded files.
	TValuesListNameField (see page 24552)	TValuesListNameField is an adapter field used to access the Name of the current list item of a values list adapter.
	TValuesListValueField (see page 24558)	TValuesListValueField is an adapter field used to access the Value (see page 23945) of the current list item of a values list adapter.

Constants

Name	Description
sActionRequestLong (see page 24613)	This is constant WebAdapt.sActionRequestLong.
sCurrentPage (see page 24613)	This is constant WebAdapt.sCurrentPage.
sGotoPageAction (see page 24614)	This is constant WebAdapt.sGotoPageAction.
sImageRequest (see page 24614)	This is constant WebAdapt.sImageRequest.
sLoginPassword (see page 24614)	This is constant WebAdapt.sLoginPassword.
sLoginUserName (see page 24615)	This is constant WebAdapt.sLoginUserName.

sNextPageAction (see page 24615)	This is constant WebAdapt.sNextPageAction.
sPrevPageAction (see page 24615)	This is constant WebAdapt.sPrevPageAction.

Functions

	Name	Description
	AddDefaultActions (see page 24589)	This is function WebAdapt.AddDefaultActions.
	AddDefaultFields (see page 24589)	This is function WebAdapt.AddDefaultFields.
	FindAdapterAction (see page 24590)	This is function WebAdapt.FindAdapterAction.
	FindAdapterField (see page 24590)	This is function WebAdapt.FindAdapterField.
	FindAdapterInParent (see page 24590)	This is function WebAdapt.FindAdapterInParent.
	MakeAdapterRequestIdentifier (see page 24591)	This is function WebAdapt.MakeAdapterRequestIdentifier.
	MakeValidAdapterFieldName (see page 24591)	This is function WebAdapt.MakeValidAdapterFieldName.
	RaiseFileUploadNotSupported (see page 24591)	This is function WebAdapt.RaiseFileUploadNotSupported.
	RaiseMultipleFilesException (see page 24592)	This is function WebAdapt.RaiseMultipleFilesException.
	RaiseMultipleValuesException (see page 24592)	This is function WebAdapt.RaiseMultipleValuesException.

Interfaces

	Name	Description
	IAdapterEditor (see page 24566)	This is class WebAdapt.IAdapterEditor.
	IAdapterNotifyAdapterChange (see page 24567)	This is class WebAdapt.IAdapterNotifyAdapterChange.
	ICheckOrigValue (see page 24568)	This is class WebAdapt.ICheckOrigValue.
	ICheckValueChange (see page 24569)	This is class WebAdapt.ICheckValueChange.
	IClearAdapterValues (see page 24570)	This is class WebAdapt.IClearAdapterValues.
	ICreateActionRequestContext (see page 24571)	This is class WebAdapt.ICreateActionRequestContext.
	IEchoActionFieldValue (see page 24572)	This is class WebAdapt.IEchoActionFieldValue.
	IEchoAdapterFieldValues (see page 24574)	This is class WebAdapt.IEchoAdapterFieldValues.
	IExecuteValue (see page 24575)	IExecuteValue is the interface for executing an adapter's fields with values passed in an HTTP request.
	IFullyQualifyInputNames (see page 24577)	This is class WebAdapt.IFullyQualifyInputNames.
	IGetAdapterErrorsList (see page 24578)	This is class WebAdapt.IGetAdapterErrorsList.
	IGetAdapterModeNames (see page 24579)	This is class WebAdapt.IGetAdapterModeNames.
	IGetRecordKeys (see page 24580)	This is class WebAdapt.IGetRecordKeys.
	IIdentifyAdapter (see page 24581)	This is class WebAdapt.IIdentifyAdapter.
	ISetSubComponents (see page 24581)	This is class WebAdapt.ISetSubComponents.
	ISetWebFieldValue (see page 24582)	This is class WebAdapt.ISetWebFieldValue.
	IUpdateValue (see page 24583)	This is class WebAdapt.IUpdateValue.
	IValidateValue (see page 24584)	This is class WebAdapt.IValidateValue.
	IWebActionsSubcomponent (see page 24585)	This is class WebAdapt.IWebActionsSubcomponent.
	IWebFieldsSubcomponent (see page 24585)	This is class WebAdapt.IWebFieldsSubcomponent.
	IWebGetFieldPrevValue (see page 24586)	This is class WebAdapt.IWebGetFieldPrevValue.

Structs, Records, Enums

	Name	Description
	TAdapterHiddenFieldOption (see page 24593)	This is record WebAdapt.TAdapterHiddenFieldOption.
	TIteratorMethod (see page 24593)	This is record WebAdapt.TIteratorMethod.

Types

Name	Description
IActionFieldValue (see page 24596)	This is type WebAdapt.IActionFieldValue.
IActionRequest (see page 24596)	This is type WebAdapt.IActionRequest.
TAbstractWebRequestFile (see page 24596)	This is type WebAdapt.TAbstractWebRequestFile.
TActionExecuteEvent (see page 24597)	This is type WebAdapt.TActionExecuteEvent.
TActionExecuteHandledEvent (see page 24597)	This is type WebAdapt.TActionExecuteHandledEvent.
TActionGetEnabledEvent (see page 24597)	This is type WebAdapt.TActionGetEnabledEvent.
TActionGetParamsEvent (see page 24598)	This is type WebAdapt.TActionGetParamsEvent.

TActionRequestFieldsEvent (see page 24598)	This is type WebAdapt.TActionRequestFieldsEvent.
TActionRequestFieldsHandledEvent (see page 24598)	This is type WebAdapt.TActionRequestFieldsHandledEvent.
TAdapterActionExecuteEvent (see page 24599)	This is type WebAdapt.TAdapterActionExecuteEvent.
TAdapterActionExecuteHandledEvent (see page 24599)	This is type WebAdapt.TAdapterActionExecuteHandledEvent.
TAdapterActionGetParamsEvent (see page 24599)	This is type WebAdapt.TAdapterActionGetParamsEvent.
TAdapterActionsClass (see page 24600)	This is type WebAdapt.TAdapterActionsClass.
TAdapterFieldBooleanValueEvent (see page 24600)	This is type WebAdapt.TAdapterFieldBooleanValueEvent.
TAdapterFieldFileEvent (see page 24600)	This is type WebAdapt.TAdapterFieldFileEvent.
TAdapterFieldFileListEvent (see page 24601)	This is type WebAdapt.TAdapterFieldFileListEvent.
TAdapterFieldGetBooleanEvent (see page 24601)	This is type WebAdapt.TAdapterFieldGetBooleanEvent.
TAdapterFieldGetMultiValueEvent (see page 24601)	This is type WebAdapt.TAdapterFieldGetMultiValueEvent.
TAdapterFieldGetMultiValueStringEvent (see page 24602)	This is type WebAdapt.TAdapterFieldGetMultiValueStringEvent.
TAdapterFieldGetStringEvent (see page 24602)	This is type WebAdapt.TAdapterFieldGetStringEvent.
TAdapterFieldGetValueCountEvent (see page 24602)	This is type WebAdapt.TAdapterFieldGetValueCountEvent.
TAdapterFieldGetValueEvent (see page 24603)	This is type WebAdapt.TAdapterFieldGetValueEvent.
TAdapterFieldHasValueEvent (see page 24603)	This is type WebAdapt.TAdapterFieldHasValueEvent.
TAdapterFieldsClass (see page 24603)	This is type WebAdapt.TAdapterFieldsClass.
TAdapterFieldStringEvent (see page 24604)	This is type WebAdapt.TAdapterFieldStringEvent.
TAdapterFieldStringsEvent (see page 24604)	This is type WebAdapt.TAdapterFieldStringsEvent.
TAdapterFieldValidateValueEvent (see page 24604)	This is type WebAdapt.TAdapterFieldValidateValueEvent.
TAdapterFieldValueEvent (see page 24605)	This is type WebAdapt.TAdapterFieldValueEvent.
TAdapterHiddenFieldOptions (see page 24605)	This is type WebAdapt.TAdapterHiddenFieldOptions.
TEndUserGetNameEvent (see page 24605)	This is type WebAdapt.TEndUserGetNameEvent.
TEndUserGetUserIDEvent (see page 24606)	This is type WebAdapt.TEndUserGetUserIDEvent.
TEndUserHandledEvent (see page 24606)	This is type WebAdapt.TEndUserHandledEvent.
TEndUserIsLoggedInEvent (see page 24606)	This is type WebAdapt.TEndUserIsLoggedInEvent.
TEndUserRightsEvent (see page 24607)	This is type WebAdapt.TEndUserRightsEvent.
TEndUserUserIDEvent (see page 24607)	This is type WebAdapt.TEndUserUserIDEvent.
TGetEOFEvent (see page 24607)	This is type WebAdapt.TGetEOFEvent.
TGetRecordCountEvent (see page 24608)	This is type WebAdapt.TGetRecordCountEvent.
TGetRecordEvent (see page 24608)	This is type WebAdapt.TGetRecordEvent.
TGetRecordIndexEvent (see page 24608)	This is type WebAdapt.TGetRecordIndexEvent.
THandledEvent (see page 24609)	This is type WebAdapt.THandledEvent.
TImageFieldGetHREFEvent (see page 24609)	This is type WebAdapt.TImageFieldGetHREFEvent.
TImageFieldGetImageEvent (see page 24609)	This is type WebAdapt.TImageFieldGetImageEvent.
TImageFieldGetParamsEvent (see page 24610)	This is type WebAdapt.TImageFieldGetParamsEvent.
TIterateRecordsEvent (see page 24610)	This is type WebAdapt.TIterateRecordsEvent.
TLoginUserIDEvent (see page 24610)	This is type WebAdapt.TLoginUserIDEvent.
TObjectList (see page 24611)	This is type WebAdapt.TObjectList.
TValidateAdapterFieldEvent (see page 24611)	This is type WebAdapt.TValidateAdapterFieldEvent.
TWebFormGetPageNameEvent (see page 24611)	This is type WebAdapt.TWebFormGetPageNameEvent.

Variables

Name	Description
NotifyPossibleHTMLChange (see page 24612)	This is variable WebAdapt.NotifyPossibleHTMLChange.

173.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EAdapterException (see page 23696)	EAdapterException is raised by WebSnap adapter objects.
	EAdapterFieldException (see page 23698)	EAdapterFieldException is an exception class for errors by adapters using fields.
	TAbstractFieldValuesAdapter (see page 23701)	tabstractfieldvaluesadapter is the base class for adapters that have a single default field.
	TAdapter (see page 23713)	TAdapter defines a scriptable interface to event handlers that return data items or execute commands in your application.
	TAdapterAction (see page 23726)	TAdapterAction is a generic adapter action that relies on event handlers.
	TAdapterActions (see page 23738)	TAdapterActions manages the list of actions that belong to an adapter.
	TAdapterApplicationField (see page 23745)	TAdapterApplicationField is a base class for adapter field components that are children of a TApplicationAdapter (see page 24032) component.
	TAdapterApplicationTitleField (see page 23752)	TAdapterApplicationTitleField is an adapter field used to access the application title.
	TAdapterBooleanField (see page 23758)	TAdapterBooleanField is an event handler based adapter field that works with a Boolean value.
	TAdapterDefaultActions (see page 23770)	TAdapterDefaultActions manages the list of actions that belong to an adapter that supports a default set of actions.
	TAdapterDefaultFields (see page 23778)	TAdapterDefaultFields manages the list of fields that belong to an adapter that supports a default set of fields.
	TAdapterEndUserDisplayNameField (see page 23787)	TAdapterEndUserDisplayNameField is an adapter field used to access the end-user's name.
	TAdapterEndUserField (see page 23793)	TAdapterEndUserField is a base class for adapter field components that are children of an end-user adapter component.
	TAdapterEndUserLoggedInField (see page 23800)	TAdapterEndUserLoggedInField is an adapter field used to access the end-user logged in flag.
	TAdapterError (see page 23807)	TAdapterError is the implementation of IAdapterError.
	TAdapterErrorsList (see page 23810)	TAdapterErrorsList keeps track of errors that an adapter encounters.
	TAdapterField (see page 23818)	TAdapterField is an event handler based adapter field that provides a single value.
	TAdapterFields (see page 23830)	TAdapterFields manages the list of fields that belong to an adapter.
	TAdapterFileField (see page 23837)	TAdapterFileField is an event handler based adapter field that can accept uploaded files.
	TAdapterGotoPageAction (see page 23849)	TAdapterGotoPageAction is an adapter action that supports commands to go to a page of a TDefaultFieldsPagedAdapter (see page 24403).
	TAdapterHiddenField (see page 23860)	This is class WebAdapt.TAdapterHiddenField.
	TAdapterHiddenFieldsList (see page 23862)	TAdapterHiddenFieldsList maintains a collection of name/value pairs that an adapter includes as hidden fields in an HTML form.
	TAdapterImageField (see page 23869)	TAdapterImageField is an event handler based adapter field that can provide the URL of an image file and also accept an uploaded image file.
	TAdapterLoginField (see page 23882)	TAdaperLoginField is a base class for adapter field components that are children of a TLoginFormAdapter (see page 24500) component.
	TAdapterMemoField (see page 23889)	TAdapterMemoField is an event handler based adapter field that supports a multi-line text value.
	TAdapterMultiValueField (see page 23900)	TAdapterMultiValueField is an event handler based adapter field that provides zero or more values.
	TAdapterNamedDisplayField (see page 23913)	The AdapterNamedDisplayField is a base class for adapter fields.
	TAdapterNamedField (see page 23925)	TAdapterNamedField is a based class for adapter field components.
	TAdapterNextPageAction (see page 23945)	TAdapterNextPageAction implements the next page command for TDefaultFieldsPagedAdapter (see page 24403).
	TAdapterNextPageField (see page 23957)	The TAdapterNextPageField navigates to the next page of data.
	TAdapterObjectErrorsList (see page 23968)	This is class WebAdapt.TAdapterObjectErrorsList.

	TAdapterPasswordField (see page 23971)	TAdapterPasswordField is an adapter field used to access the password entered by the end-user.
	TAdapterPrevPageAction (see page 23979)	TAdapterPrevPageAction implements the previous page command for TDefaultFieldsPagedAdapter (see page 24403).
	TAdapterSubComponent (see page 23991)	TAdapterSubComponent is the base class for objects that represent a collection of actions or fields that belong to an adapter.
	TAdapterUpdateField (see page 24007)	TAdapterUpdateField is a base class for adapter field components that support data entry into an HTML form.
	TAdapterUserNameField (see page 24024)	TAdapterUserNameField is an adapter field used to access the user name entered by the end-user.
	TApplicationAdapter (see page 24032)	TApplicationAdapter is a container for field and action components that are available through the Application script variable.
	TBaseAdapterAction (see page 24043)	TBaseAdapterAction is a based class for adapter action components.
	TBaseAdapterErrorsList (see page 24077)	TBaseAdapterErrorsList is the base class for adapter error object managers.
	TBaseAdapterMultiValueField (see page 24086)	TBaseAdapterMultiValueField is a base class for adapter fields that have zero or more values.
	TBaseEndUserAdapter (see page 24097)	TBaseEndUserAdapter contains user information.
	TBaseValuesListAdapter (see page 24120)	TBaseValuesListAdapter is a base class for adapters that provide a list of name/value pairs.
	TBaseValuesListField (see page 24132)	TBaseValuesListField is a base class for adapter field components that are children of a values list adapter component.
	TCustomAdapter (see page 24139)	TCustomAdapter is a base class for adapter components.
	TCustomAdapterAction (see page 24188)	TCustomAdapterAction is a base class for adapter actions.
	TCustomAdapterBooleanField (see page 24201)	tcustomadapterbooleanfield is a base class for adapter fields that provide a single Boolean value.
	TCustomAdapterField (see page 24211)	TCustomAdapterField is a base class for adapter fields that provide a single value.
	TCustomAdapterFileField (see page 24220)	TCustomAdapterFileField is a base class for adapter fields that can have associated files.
	TCustomAdapterGotoPageAction (see page 24233)	TCustomAdapterGotoPageAction is an adapter action that supports commands to go to a page of a TDefaultFieldsPagedAdapter (see page 24403).
	TCustomAdapterImageField (see page 24244)	TCustomAdapterImageField is a base class for adapter fields that have an associated image file.
	TCustomAdapterMemoField (see page 24262)	TCustomAdapterMemoField is a base class for adapter fields that have a multi-line text value.
	TCustomAdapterMultiValueField (see page 24273)	TCustomAdapterMultiValueField is a base class for adapter fields that have one or more values.
	TCustomAdapterPageAction (see page 24286)	TCustomPagedAdapterAction (see page 24371) is a base class for adapter actions that execute the next or previous page commands.
	TCustomApplicationAdapter (see page 24296)	TCustomApplicationAdapter is a base class for components that implement the server-side script Application object.
	TCustomApplicationAdapterAction (see page 24306)	TCustomApplicationAdapterAction is a base class for adapter actions that work with descendants of TCustomApplicationAdapter (see page 24296).
	TCustomEndUserAdapter (see page 24313)	TCustomEndUserAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TCustomEndUserAdapterAction (see page 24326)	TCustomEndUserAdapterAction is a base class for adapter actions that work with descendants of TCustomEndUserAdapter (see page 24313).
	TCustomEndUserSessionAdapter (see page 24333)	TCustomEndUserSessionAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TCustomLoginFormAdapter (see page 24344)	TCustomLoginFormAdapter is the base class for adapters that manage the fields and actions of a login form.
	TCustomLoginFormAdapterAction (see page 24363)	TCustomEndUserAdapterAction (see page 24326) is a base class for adapter actions that work with descendants of TCustomLoginFormAdapter (see page 24344).
	TCustomPagedAdapter (see page 24370)	TCustomPagedAdapter is a base class for adapters that support paging.
	TCustomPagedAdapterAction (see page 24371)	TCustomPagedAdapterAction is a base class for adapter actions that work with descendants of TDefaultFieldsPagedAdapter (see page 24403).
	TCustomStringsValuesList (see page 24378)	TCustomStringsValuesList is a base class for a values list that defines it's name/value pairs using a string list.
	TDefaultFieldsAdapter (see page 24392)	TDefaultFieldsAdapter is a base class for adapters that automatically adds default fields and/or actions.
	TDefaultFieldsPagedAdapter (see page 24403)	TDefaultFieldsPagedAdapter is a base class for adapters that support paging.

	TEndUserAdapter (see page 24421)	TEndUserAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TEndUserAdapterLoginFormAction (see page 24436)	TEndUserAdapterLoginFormAction is an adapter action that displays a login form when executed.
	TEndUserAdapterLogoutAction (see page 24447)	TEndUserAdapterLogoutAction is an adapter action that logs the end-user out.
	TEndUserSessionAdapter (see page 24458)	TEndUserSessionAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TFieldValuesAdapter (see page 24471)	TFieldValuesAdapter is an adapter that includes a single field value.
	TFieldValuesValueField (see page 24483)	TFieldValuesValueField is a adapter field child of TFieldValuesAdapter (see page 24471).
	TImplementedAdapterAction (see page 24491)	TImplementedAdapterAction is a base class for adapter actions.
	TLogicalPageList (see page 24492)	This is class WebAdapt.TLogicalPageList.
	TLoginFormAdapter (see page 24500)	TLoginFormAdapter contains the adapter fields and adapter actions used to generate a login form.
	TLoginFormAdapterLoginAction (see page 24514)	TLoginFormAdapterLoginAction executes the login command of a login form generated with TLoginFormAdapter (see page 24500).
	TPagedAdapter (see page 24525)	TPagedAdapter divides a large amount of data into several pages, each page displaying a portion of the data records.
	TStringsValuesList (see page 24540)	TStringsValuesList is an adapter component that provides a list of name/value pairs defined by items in a TStrings property.
	TUpdateFileList (see page 24550)	TUpdateFileList is a container for uploaded files.
	TValuesListNameField (see page 24552)	TValuesListNameField is an adapter field used to access the Name of the current list item of a values list adapter.
	TValuesListValueField (see page 24558)	TValuesListValueField is an adapter field used to access the Value (see page 23945) of the current list item of a values list adapter.

173.1.1 EAdapterException Class

EAdapterException is raised by WebSnap adapter objects.

Class Hierarchy

System::TObject → SysUtils.Exception → HTTPApp.EWebBrokerException → WebAdapt.EAdapterException

Pascal

```
EAdapterException = class(EWebBrokerException);
```

C++

```
class EAdapterException : public EWebBrokerException;
```

File

WebAdapt

Description

EAdapterException is raised when a WebSnap adapter object encounters an error. EAdapterException provides an ID (see page 23698) property to record application-defined errors.

173.1.1.1 EAdapterException Members

The following tables list the members exposed by EAdapterException.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 22752)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateID (see page 23697)	Create exception object with error ID (see page 23698).
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	ID (see page 23698)	Application error ID.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.

173.1.1.2 EAdapterException.CreateID Constructor

Create exception object with error ID (see page 23698).

Pascal

```
constructor CreateID(const AMsg: string; AID: Integer; Reserved: Integer = 0);
```

C++

```
__fastcall EAdapterException(const AnsiString AMsg, int AID, int Reserved = 0);
```

Description

CreateID creates an EAdapterException (see page 23696) object in the usual way, and also assigns AID to the ID (see page 23698) property.

Reserved is not implemented.

173.1.1.3 EAdapterException Properties

The properties of the EAdapterException class are listed here.

Public Properties

	Name	Description
	ID (see page 23698)	Application error ID.

173.1.1.3.1 EAdapterException.ID Property

Application error ID.

Pascal

```
property ID: Integer;
```

C++

```
__property int ID;
```

Description

The ID property returns an application-defined value relating to the error. To specify an ID value, create an EAdapterException (see page 23696) object using the CreateID (see page 23697) constructor (Delphi) or the constructor syntax that includes an ID as a parameter (C++). For exceptions raised by built-in WebSnap adapters, ID is always 0.

See Also

CreateID (see page 23697)

173.1.2 EAdapterFieldException Class

EAdapterFieldException is an exception class for errors by adapters using fields.

Class Hierarchy

Pascal

```
EAdapterFieldException = class(EAdapterException);
```

C++

```
class EAdapterFieldException : public EAdapterException;
```

File

WebAdapt

Description

EAdapterFieldException is raised when an adapter encounters an error using a field.

173.1.2.1 EAdapterFieldException Members

The following tables list the members exposed by EAdapterFieldException.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 23700)	Create an EAdapterFieldException (see page 23698) instance.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateID (see page 23697)	Create exception object with error ID (see page 23698).
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	FieldName (see page 23700)	FieldName contains the name of the field associated with the error.

	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	ID (see page 23698)	Application error ID.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.

173.1.2.2 EAdapterFieldException.Create Constructor

Create an EAdapterFieldException (see page 23698) instance.

Pascal

```
constructor Create(const AMsg: string; const AFieldName: string; AID: Integer = 0);
```

C++

```
__fastcall EAdapterFieldException(const AnsiString AMsg, const AnsiString AFieldName, int AID
= 0);
```

Description

Create instantiates an instance of EAdapterFieldException (see page 23698).

AMsg is the string containing the runtime error message to display in the exception dialog box. AMsg can be a hard-coded string, or can be a function call that returns a string.

AFieldName and AID are assigned to the FieldName (see page 23700) and ID (see page 23698) properties.

See Also

FieldName (see page 23700)

ID (see page 23698)

173.1.2.3 EAdapterFieldException Properties

The properties of the EAdapterFieldException class are listed here.

Public Properties

	Name	Description
	FieldName (see page 23700)	FieldName contains the name of the field associated with the error.

173.1.2.3.1 EAdapterFieldException.FieldName Property

FieldName contains the name of the field associated with the error.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;
```

Description

```
__property AnsiString FieldName = {read=FFieldName, write=FFieldName};
property FieldName: string;
```

FieldName contains the name of the field associated with the error.

173.1.3 TAbstractFieldValuesAdapter Class

tabstractfieldvaluesadapter is the base class for adapters that have a single default field.

Class Hierarchy

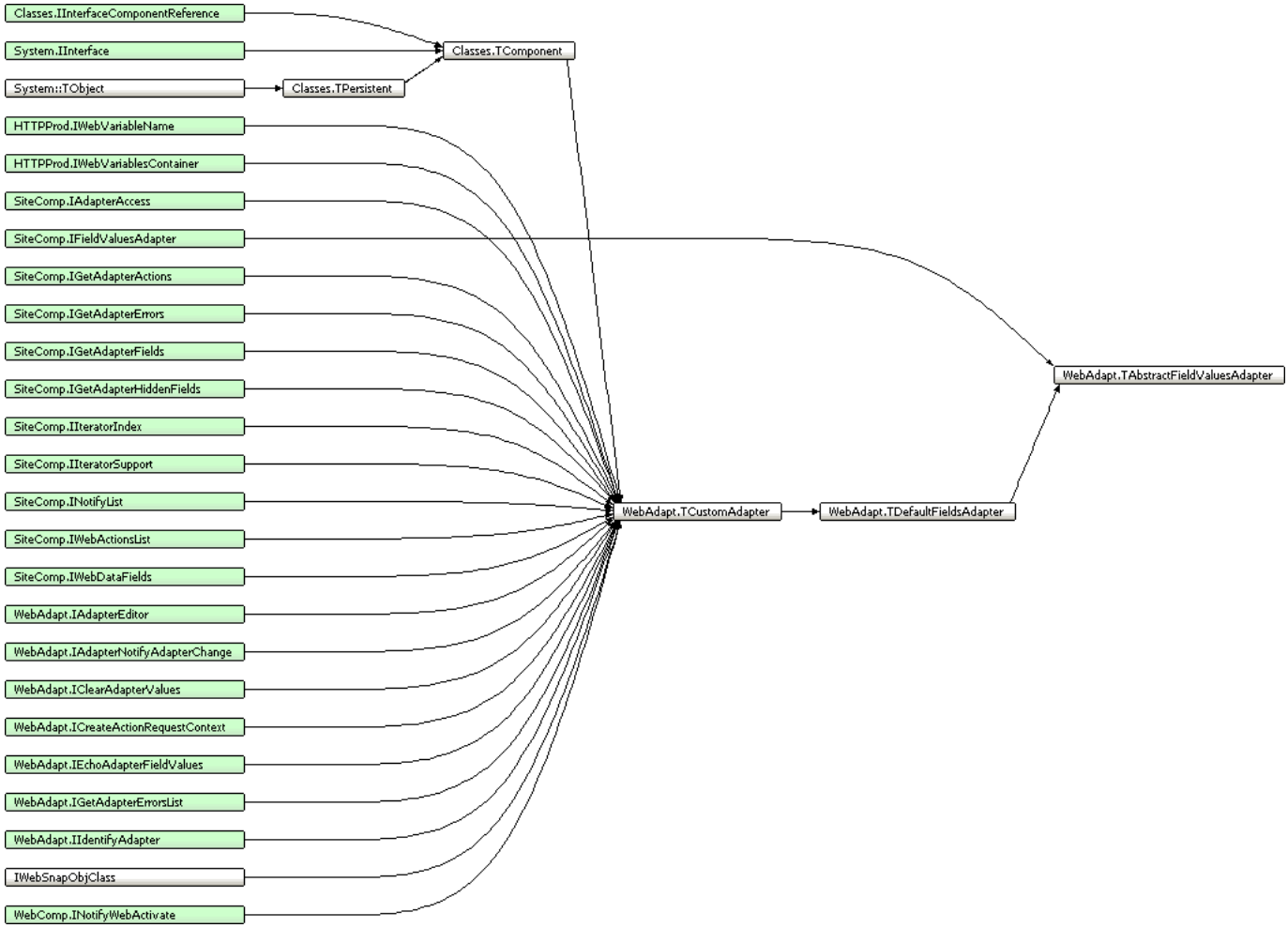

Pascal

```
TAbstractFieldValuesAdapter = class(TDefaultFieldsAdapter, IFieldValuesAdapter);
```

C++

```
class TAbstractFieldValuesAdapter : public TDefaultFieldsAdapter, public IFieldValuesAdapter;
```

File

WebAdapt

Description

Derive from tabstractfieldvaluesadapter to define adapter components that automatically include a single adapter field called ‘ValueField'.

Do not use TAbstractFieldValuesAdpter directly. It includes many abstract or, in C++ parlance, pure virtual methods, which must

be implemented by a descendant class such as TFieldValuesAdapter (see page 24471).

See Also

TFieldValuesAdapter (see page 24471)

173.1.3.1 TAbstractFieldValuesAdapter Members

The following tables list the members exposed by TAbstractFieldValuesAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.

	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetField (see page 23708)	Returns the object that represents the single field of this adapter.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
A	GetValueFieldClass (see page 23709)	Returns the class of the single adapter field this adapter includes.
	GetValueIndex (see page 23709)	Returns an index that identifies the current adapter field value.
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
V	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 23710)	Provides the underlying implementation for the CanAddFieldClass (see page 24150) method.
V	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
V	ImplEndIterator (see page 23710)	Provides the underlying implementation for the EndIterator (see page 24152) method.
V	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
V	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
A	ImplGetField (see page 23711)	Provides the underlying implementation for the GetField (see page 23708) method.
V	ImplGetFieldsList (see page 23711)	Provides the underlying implementation for the GetFieldsList (see page 24159) method.
V	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
A	ImplGetValueIndex (see page 23711)	Provides the underlying implementation for the GetValueIndex (see page 23709) method.

	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 23712)	Provides the underlying implementation for the NextIteration (see page 24173) method.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 23712)	Provides the underlying implementation for the StartIterator (see page 24176) method.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 24184)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24185)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24186)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24186)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 24187)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.

	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 24181)	Actions is the container for adapter actions components.
R	Actions (see page 24402)	Container for adapter actions components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Data (see page 24181)	Data is the container for adapter field components.
R	Data (see page 24402)	Container for adapter field components.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.

	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.3.2 TAbstractFieldValuesAdapter Methods

The methods of the TAbstractFieldValuesAdapter class are listed here.

Protected Methods

	Name	Description
	GetField (see page 23708)	Returns the object that represents the single field of this adapter.
A	GetValueFieldClass (see page 23709)	Returns the class of the single adapter field this adapter includes.
	GetValueIndex (see page 23709)	Returns an index that identifies the current adapter field value.
V	ImplCanAddFieldClass (see page 23710)	Provides the underlying implementation for the CanAddFieldClass (see page 24150) method.
V	ImplEndIterator (see page 23710)	Provides the underlying implementation for the EndIterator (see page 24152) method.
A	ImplGetField (see page 23711)	Provides the underlying implementation for the GetField (see page 23708) method.
V	ImplGetFieldsList (see page 23711)	Provides the underlying implementation for the GetFieldsList (see page 24159) method.
A	ImplGetValueIndex (see page 23711)	Provides the underlying implementation for the GetValueIndex (see page 23709) method.
V	ImplNextIteration (see page 23712)	Provides the underlying implementation for the NextIteration (see page 24173) method.
V	ImplStartIterator (see page 23712)	Provides the underlying implementation for the StartIterator (see page 24176) method.

173.1.3.2.1 TAbstractFieldValuesAdapter.GetField Method

Returns the object that represents the single field of this adapter.

Pascal

```
function GetField: TObject;
```

C++

```
__fastcall TObject * GetField();
```

Description

Call GetField to access the single field for this adapter. GetField returns an adapter field that is an instance of the class that the GetValueFieldClass (see page 23709) method returns.

GetField calls the abstract or, in C++ terminology, pure virtual ImplGetField (see page 23711) method to obtain the adapter field instance it returns. Descendant classes override ImplGetField (see page 23711) to provide the underlying implementation

for GetField.

See Also

ImplGetField (see page 23711)

GetValueFieldClass (see page 23709)

173.1.3.2.2 TAbstractFieldValuesAdapter.GetValueFieldClass Method

Returns the class of the single adapter field this adapter includes.

Pascal

```
function GetValueFieldClass: TComponentClass; virtual; abstract;
```

C++

```
virtual __fastcall TComponentClass GetValueFieldClass() = 0;
```

Description

GetValueFieldClass returns the class reference for the type of the single adapter field that the GetField (see page 23708) method returns. The return value must be a descendant of TComponent and is typically a descendant of TAdapterNamedField (see page 23925).

In TAbstractFieldValuesAdapter (see page 23701), GetValueFieldClass is abstract or, in C++ terminology, pure virtual. Descendant classes must override this method to indicate the type of field this adapter includes.

See Also

GetField (see page 23708)

TAdapterNamedField (see page 23925)

173.1.3.2.3 TAbstractFieldValuesAdapter.GetValueIndex Method

Returns an index that identifies the current adapter field value.

Pascal

```
function GetValueIndex: Integer;
```

C++

```
__fastcall int GetValueIndex();
```

Description

Although this adapter only includes a single adapter field, that field may have several values (for example, if it represents a column in a table). When the field has several values, GetValueIndex is the index of the current value. This index may be the same as the value that the GetIteratorIndex (see page 24160) method returns if the application is using the adapter's iteration support. However, TAbstractFieldValuesAdapter (see page 23701) introduces GetValueIndex as a separate method for descendants that support multiple values that change using some other mechanism.

GetValueIndex calls the abstract or, in C++ terminology, pure virtual ImplGetValueIndex (see page 23711) method to obtain the index of the current value. Descendant classes override ImplGetValueIndex (see page 23711) to provide the underlying implementation for GetValueIndex.

See Also

ImplGetValueIndex (see page 23711)

GetIteratorIndex (see page 24160)

173.1.3.2.4 TAbstractFieldValuesAdapter.ImplCanAddFieldClass Method

Provides the underlying implementation for the CanAddFieldClass (see page 24150) method.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddFieldClass (see page 24150) method calls ImplCanAddFieldClass to determine whether this adapter can include a field that is an instance of a specified class. This allows the adapter editor to indicate the available fields at design time.

AParent is an instance of this adapter class to which the field is to be added.

AClass is the class of the object that represents the adapter field.

ImplCanAddFieldClass returns true if AClass is the same as (or a descendant of) the class that the GetValueFieldClass (see page 23709) method returns, indicating that this is the only child field this adapter supports. It returns false for any other class.

See Also

CanAddFieldClass (see page 24150)

GetValueFieldClass (see page 23709)

173.1.3.2.5 TAbstractFieldValuesAdapter.ImplEndIterator Method

Provides the underlying implementation for the EndIterator (see page 24152) method.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer); override;
```

C++

```
virtual __fastcall ImplEndIterator(void * OwnerData);
```

Description

The EndIterator (see page 24152) method calls ImplEndIterator to stop the process of iterating through the adapter's field's possible values.

OwnerData is a pointer to adapter-specific data that can affect the way it iterates over field values. Each adapter can interpret this value in its own way.

ImplEndIterator raises an exception, because this method should never be called at the TAbstractFieldValuesAdapter (see page 23701) level: The TAbstractFieldValuesAdapter (see page 23701) ImplStartIterator (see page 23712) method returns false, indicating that this adapter does not support iteration.

Descendant classes that implement support for iterating over field values override this method to end that iteration process.

See Also

EndIterator (see page 24152)

ImplStartIterator (see page 23712)

ImplNextIteration (see page 23712)

173.1.3.2.6 TAbstractFieldValuesAdapter.ImplGetField Method

Provides the underlying implementation for the GetField (see page 23708) method.

Pascal

```
function ImplGetField: TObject; virtual; abstract;
```

C++

```
virtual __fastcall TObject * ImplGetField() = 0;
```

Description

The GetField (see page 23708) method calls ImplGetField to obtain the object that represents this adapter's single field. Unlike the GetField (see page 23708) method, ImplGetField is virtual, so that descendant classes can override it to provide a class-specific mechanism for obtaining the adapter field object.

In TAbstractFieldValuesAdapter (see page 23701), ImplGetField is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the GetField (see page 23708) method.

See Also

GetField (see page 23708)

173.1.3.2.7 TAbstractFieldValuesAdapter.ImplGetFieldsList Method

Provides the underlying implementation for the GetFieldsList (see page 24159) method.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

The GetFieldsList (see page 24159) method calls ImplGetFieldsList to fill a string list with a list of all fields that the adapter includes. Each entry in the list is the name of one of the adapter's field, and has an associated object that is the class of the object that represents that adapter field. GetFieldsList (see page 24159) is part of the design-time support for editing adapters.

AList is an existing string list to fill with entries for all of the fields this adapter can include.

In TAbstractFieldValuesAdapter (see page 23701), ImplGetFieldsList clears AList and inserts a single entry for a field named 'ValueField'. It associates the class that the GetValueFieldClass (see page 23709) method returns with this field name.

See Also

GetValueFieldClass (see page 23709)

GetFieldsList (see page 24159)

173.1.3.2.8 TAbstractFieldValuesAdapter.ImplGetValueIndex Method

Provides the underlying implementation for the GetValueIndex (see page 23709) method.

Pascal

```
function ImplGetValueIndex: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int ImplGetValueIndex() = 0;
```

Description

The GetValueIndex (see page 23709) method calls ImplGetValueIndex to obtain the index of current value of this adapter's single field. Unlike the GetValueIndex (see page 23709) method, ImplGetValueIndex is virtual, so that descendant classes can override it to provide a class-specific mechanism for obtaining this index.

In TAbstractFieldValuesAdapter (see page 23701), ImplGetValueIndex is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the GetValueIndex (see page 23709) method.

See Also

GetValueIndex (see page 23709)

173.1.3.2.9 TAbstractFieldValuesAdapter.ImplNextIteration Method

Provides the underlying implementation for the NextIteration (see page 24173) method.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

The NextIteration (see page 24173) method calls ImplNextIteration when iterating over field values to determine whether there is another field value after the current one and if so, to change the current value to the next one.

OwnerData is a pointer to adapter-specific data that can affect the way it iterates over field values. On entry, this value is typically the same as the value returned by the last call to NextIteration (see page 24173) or to StartIterator (see page 24176).

In TAbstractFieldValuesAdapter (see page 23701), ImplNextIteration always returns false, because TAbstractFieldValuesAdapter (see page 23701) does not implement support for iteration. Descendant classes that support iterating over field values override this method to check whether their is a next value and if so, move to it.

See Also

NextIteration (see page 24173)

ImplStartIterator (see page 23712)

ImplEndIterator (see page 23710)

173.1.3.2.10 TAbstractFieldValuesAdapter.ImplStartIterator Method

Provides the underlying implementation for the StartIterator (see page 24176) method.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

The StartIterator (see page 24176) method calls ImplStartIterator to start the process of iterating through the adapter's field's possible values. The return value indicates whether the adapter's field has any values over which to iterate.

OwnerData is a pointer to adapter-specific data that can affect the way it iterates over field values. Each adapter can interpret this value in its own way. Depending on the adapter, it may be supplied by the caller, or created by the ImplStartIterator method.

In TAbstractFieldValuesAdapter (see page 23701), ImplNextIteration (see page 23712) always returns false, because TAbstractFieldValuesAdapter (see page 23701) does not implement support for obtaining any field values. Descendant classes that override this method to check whether they can move to the first value in the field's set of values. If so, they move to the first value and return true. Otherwise they return false.

See Also

StartIterator (see page 24176)

ImplEndIterator (see page 23710)

ImplNextIteration (see page 23712)

173.1.4 TAdapter Class

TAdapter defines a scriptable interface to event handlers that return data items or execute commands in your application.

Class Hierarchy

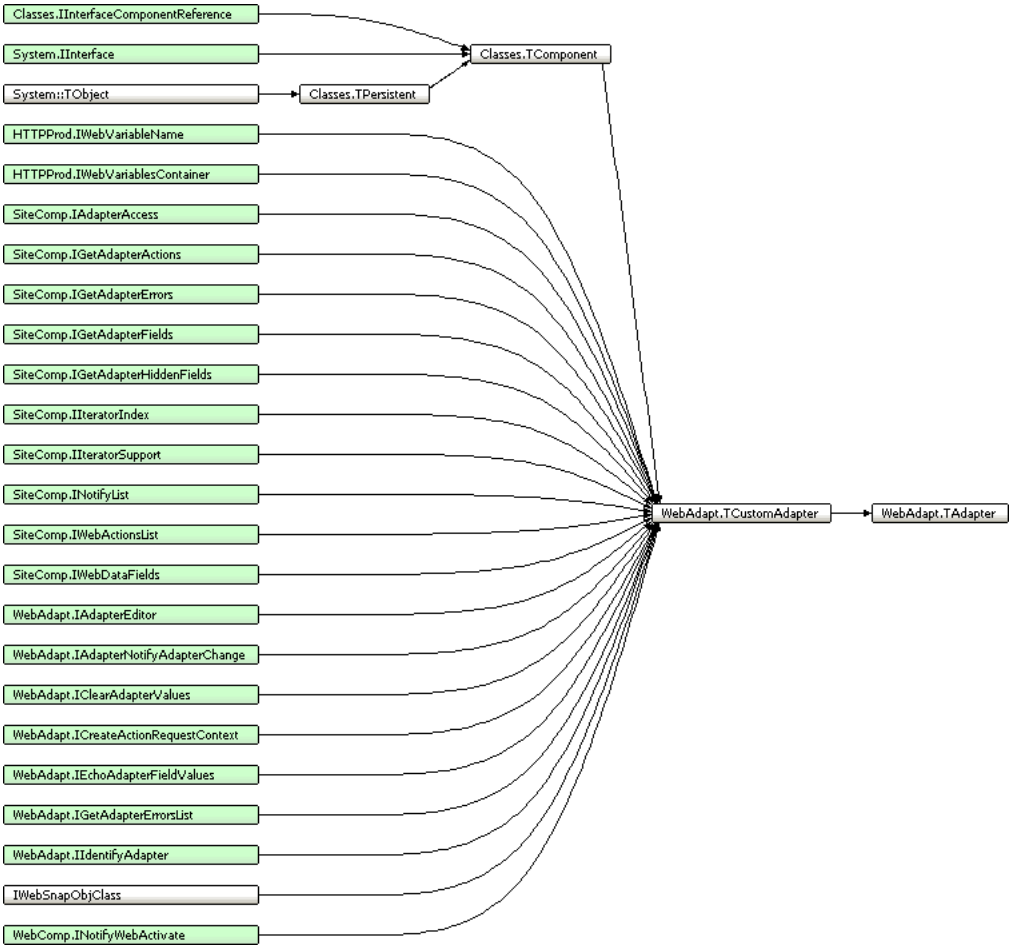

Pascal

```
TAdapter = class(TCustomAdapter);
```

C++

```
class TAdapter : public TCustomAdapter;
```

File

WebAdapt

Description

Use TAdapter to define a custom interface to application logic. Adapters contain field components and action components that represent data items and commands, respectively. Server-side script statements access the value of Adapter fields and the parameters of Adapter actions in order to build HTML forms and tables. Server-side script can be written by hand or automatically generated by the TAdapterPageProducer component. When an HTML form containing Adapter commands is submitted, the TAdapterDispatcher may automatically call an Adapter action component to execute the command. When an HTTP request for an adapter image field is received the TAdapterDispatcher may automatically call an Adapter field to retrieve the image.

For each field component added to TAdapter, you must implement an event handler to retrieve the data. For each action component added to TAdapter, you must implement an event handler to execute the command. Other adapter components,

such as TDataSetAdapter and TLoginFormAdapter (see page 24500), have built in fields and actions that do not require events.

173.1.4.1 TAdapter Members

The following tables list the members exposed by TAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.

	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.

	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.

	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.

	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 23722)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 23723)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 23723)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 23723)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 23724)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 23724)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 23725)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 23725)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 23725)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 23726)	Occurs when the adapter needs to iterate over its data records.

Published Properties

	Name	Description
	Actions (see page 23721)	Actions is the container for adapter actions components.
	Data (see page 23721)	Data is the container for adapter field components.
	ModifyAccess (see page 23721)	This is ModifyAccess, a member of class TAdapter.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	ViewAccess (see page 23722)	Identifies the rights required to view this adapter's fields.

173.1.4.2 TAdapter Properties

The properties of the TAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 23721)	Actions is the container for adapter actions components.
	Data (see page 23721)	Data is the container for adapter field components.
	ModifyAccess (see page 23721)	This is ModifyAccess, a member of class TAdapter.
	ViewAccess (see page 23722)	Identifies the rights required to view this adapter's fields.

173.1.4.2.1 TAdapter.Actions Property

Actions is the container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterActions (see page 23738)

173.1.4.2.2 TAdapter.Data Property

Data is the container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterFields (see page 23830)

173.1.4.2.3 TAdapter.ModifyAccess Property

Pascal

```
property ModifyAccess: string;
```

C++

```
__property AnsiString ModifyAccess;
```

Description

This is ModifyAccess, a member of class TAdapter.

173.1.4.2.4 TAdapter.ViewAccess Property

Identifies the rights required to view this adapter's fields.

Pascal

```
property ViewAccess: string;
```

C++

```
__property AnsiString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view fields of this adapter. When server-side script accesses the CanView property, the ViewAccess property value is checked against the end user's rights.

See Also

TWebUserList (see page 25625)

HasViewAccess (see page 24162)

implhasviewaccess (see page 24169)

173.1.4.3 TAdapter Events

The events of the TAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 23722)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 23723)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 23723)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 23723)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 23724)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 23724)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 23725)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 23725)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 23725)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 23726)	Occurs when the adapter needs to iterate over its data records.

173.1.4.3.1 TAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter

action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

173.1.4.3.2 TAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.4.3.3 TAdapter.OnAfterUpdateFields Event

Occurs after values in the HTTP request are used to update adapter fields.

Pascal

```
property OnAfterUpdateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterUpdateFields;
```

Description

Implement this event handler to perform custom updates. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record update errors.

See Also

UpdateFieldsUpdateValues (see page 24178)

OnBeforeUpdateFields (see page 24187)

Errors (see page 24182)

173.1.4.3.4 TAdapter.OnAfterValidateFields Event

Occurs after values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnAfterValidateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record validation errors.

See Also

UpdateFieldsValidateValues (see page 24178)

OnBeforeValidateFields (see page 24187)

Errors (see page 24182)

173.1.4.3.5 TAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 24076)

OnAfterExecuteAction (see page 24184)

173.1.4.3.6 TAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 24076)

OnAfterGetActionResponse (see page 24185)

173.1.4.3.7 TAdapter.OnBeforeUpdateFields Event

Occurs before values in the HTTP request are used to update adapter fields.

Pascal

```
property OnBeforeUpdateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeUpdateFields;
```

Description

Implement this event handler to perform custom updates of adapter fields. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further updates. Add update errors to the adapter's errors object.

See Also

UpdateFieldsUpdateValues (see page 24178)

OnAfterUpdateFields (see page 24185)

Errors (see page 24182)

173.1.4.3.8 TAdapter.OnBeforeValidateFields Event

Occurs before values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnBeforeValidateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further validation. Add validation errors to the adapter's errors object.

See Also

UpdateFieldsValidateValues (see page 24178)

OnAfterValidateFields (see page 24186)

Errors (see page 24182)

173.1.4.3.9 TAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 24077)

173.1.4.3.10 TAdapter.OnIterateRecords Event

Occurs when the adapter needs to iterate over its data records.

Pascal

```
property OnIterateRecords: TIterateRecordsEvent;
```

C++

```
__property TIterateRecordsEvent OnIterateRecords;
```

Description

Implement this event when the adapter supports multiple data records. This event handler is called when server-side script iterates over the records in adapter, such as when generating an HTML table displaying all records. The event handler must update an iteration pointer or cursor that adapter fields are also aware of. Adapter fields must use this iteration pointer when called to retrieve their value.

See Also

IIteratorSupport (see page 19722)

173.1.5 TAdapterAction Class

TAdapterAction is a generic adapter action that relies on event handlers.

Class Hierarchy

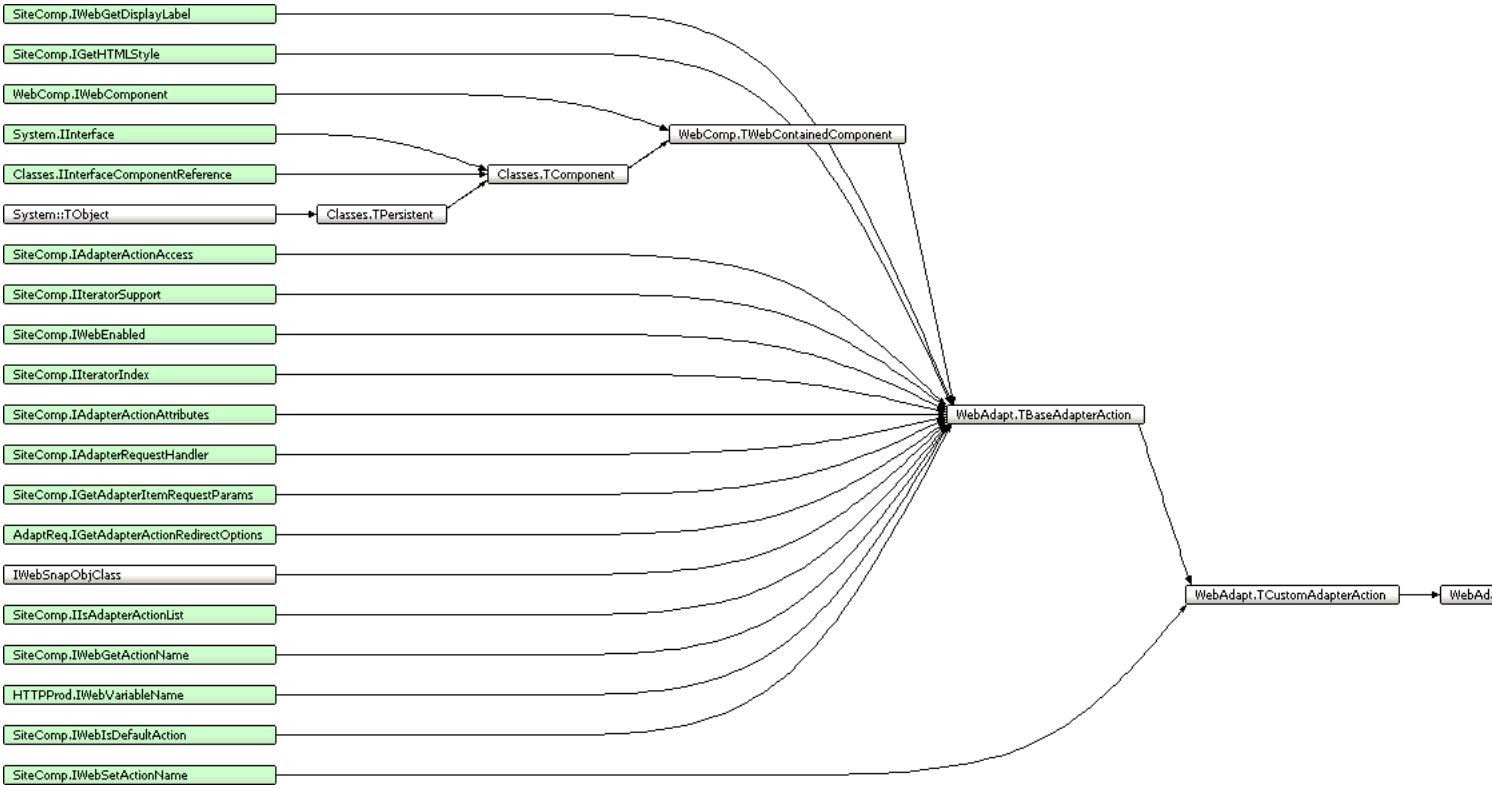

Pascal

```
TAdapterAction = class(TCustomAdapterAction);
```

C++

```
class TAdapterAction : public TCustomAdapterAction;
```

File

WebAdapt

Description

TAdapterAction publishes the properties in TCustomAdapterAction (see page 24188). TCustomAdapterAction (see page 24188) calls event handlers to retrieve the adapter action's parameters, get the state of the action, and execute the adapter action. Add TAdapterAction to TAdapter (see page 23713) or other adapter components types when you need an adapter action that will be controlled by event handlers. Other adapter action components declare events but also have built-in logic to work with a particular type of adapter. For example, TAdapterNextPageAction (see page 23945) has built-in logic to take the end user to the next page provided by TPagedAdapter (see page 24525). TAdapterAction, on the other hand, does not have this sort of built-in logic.

See Also

TAdapterActions (see page 23738)

TAdapter (see page 23713)

TAdapterDispatcher (see page 24840)

173.1.5.1 TAdapterAction Members

The following tables list the members exposed by TAdapterAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 24055)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.

	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 24063)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 24195)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 24069)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 24197)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24070)	Handles movement of the component to a different parent.

V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.

V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.

	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 23735)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 23735)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 23736)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 23736)	Called before the adapter action response is generated.
	OnExecute (see page 23737)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 23737)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 23737)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	ActionName (see page 23733)	Identifies the name of this action when using server-side script.
	DisplayLabel (see page 23734)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 23734)	Defines the rights required to execute this action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RedirectOptions (see page 23734)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.5.2 TAdapterAction Properties

The properties of the TAdapterAction class are listed here.

Published Properties

	Name	Description
	ActionName (see page 23733)	Identifies the name of this action when using server-side script.
	DisplayLabel (see page 23734)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 23734)	Defines the rights required to execute this action.
	RedirectOptions (see page 23734)	Indicates whether to perform an HTTP redirect after successfully executing this action.

173.1.5.2.1 TAdapterAction.ActionName Property

Identifies the name of this action when using server-side script.

Pascal

```
property ActionName: string;
```

C++

```
__property AnsiString ActionName;__property AnsiString ActionName;__property AnsiString
ActionName;
```

Description

The ActionName property defines the name used to access the adapter action with server-side script.

See Also

ImplGetActionName (see page 24195)

ImplSetActionName (see page 24197)

173.1.5.2.2 TAdapterAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

173.1.5.2.3 TAdapterAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property AnsiString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 25625)

hasexecuteaccess (see page 24162)

173.1.5.2.4 TAdapterAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect (see page 15201)

173.1.5.3 TAdapterAction Events

The events of the TAdapterAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 23735)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 23735)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 23736)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 23736)	Called before the adapter action response is generated.
	OnExecute (see page 23737)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 23737)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 23737)	Specifies name/value pairs needed to execute this adapter action.

173.1.5.3.1 TAdapterAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

173.1.5.3.2 TAdapterAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by

DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.5.3.3 TAdapterAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

173.1.5.3.4 TAdapterAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.5.3.5 TAdapterAction.OnExecute Event

Called to execute this action using parameters passed in the HTTP request.

Pascal

```
property OnExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnExecute;
```

Description

Implement this event handler to execute this adapter action. Params contains the name/value pairs that were passed in the HTTP request. OnExecute is called by ImplExecuteActionRequest.

See Also

ImplExecuteActionRequest (see page 24195)

OnGetParams (see page 24077)

173.1.5.3.6 TAdapterAction.OnGetEnabled Event

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
property OnGetEnabled: TActionGetEnabledEvent;
```

C++

```
__property TActionGetEnabledEvent OnGetEnabled;
```

Description

ImplWebEnabled calls the OnGetEnabled event handler. Set the Enabled parameter to false to indicate that this adapter action should not be enabled.

See Also

ImplWebEnabled (see page 24197)

173.1.5.3.7 TAdapterAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

173.1.6 TAdapterActions Class

TAdapterActions manages the list of actions that belong to an adapter.

Class Hierarchy

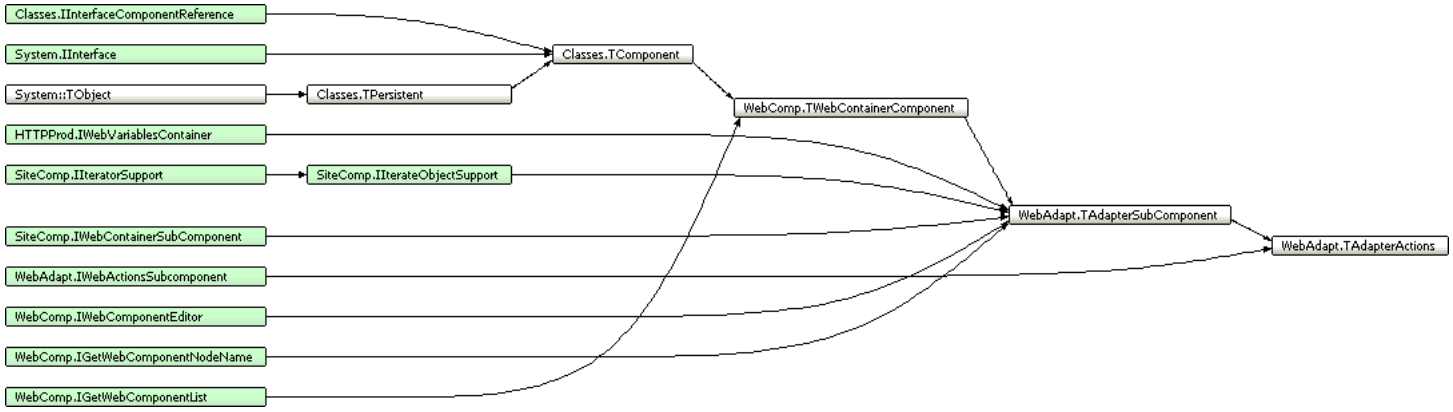

Pascal

```
TAdapterActions = class(TAdapterSubComponent, IWebActionsSubcomponent);
```

C++

```
class TAdapterActions : public TAdapterSubComponent, public IWebActionsSubcomponent;
```

File

WebAdapt

Description

TAdapterActions is the type of the Actions property on an adapter component. It manages a list that contains all the actions that belong to the adapter. The action objects (TBaseAdapterAction (see page 24043) descendants) are available using the WebActions (see page 23745) property. TAdapterActions also includes support for associating the actions with variable names so that server-side script can execute the actions.

TAdapterActions represents the list of explicitly assigned actions. Adapters that include a set of default actions use a TAdapterDefaultActions (see page 23770) object instead.

See Also

TCustomAdapter (see page 24139)

TAdapterFields (see page 23830)

TAdapterDefaultActions (see page 23770)

TAdapterDefaultFields (see page 23778)

TBaseAdapterAction (see page 24043)

173.1.6.1 TAdapterActions Members

The following tables list the members exposed by TAdapterActions.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AdapterCanAddClass (see page 23742)	Indicates whether the associated adapter supports a specified class of action component.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 23996)	Indicates whether an object of a specified type can be added to this collection.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EndIterator (see page 23997)	Stops the TAdapterSubComponent (see page 23991) object from iterating over the Web items it lists.
V	FindNamedVariable (see page 23997)	Returns the item from this list with a specified variable name.
	FindVariable (see page 23998)	Returns the item from this list with a specified variable name.
	FindVariableContainer (see page 23998)	Returns the Web module that contains a specified item from this list.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24755)	Executes a callback for each child Web item.
	GetCurrentObject (see page 23999)	Returns the current field or action from the list when an enumerator is iterating over child items.
V	GetDefaultWebComponents (see page 23999)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
V	GetDefaultWebComponents (see page 24756)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariable (see page 24000)	Returns a child item from this list, given its position in the list of child items with variable names.
	GetVariableCount (see page 24000)	Indicates the number of items in this list that can be accessed from server-side script.
V	GetVariablesInUse (see page 24001)	Returns the value of the VariablesInUse (see page 24007) property.
	GetWebComponentNodeName (see page 24001)	Returns a string that indicates the type of items in this collection.
	GetWebComponents (see page 24756)	Returns the list of Web items that this item manages.
V	ImplCanAddClass (see page 23743)	Provides the underlying implementation for the CanAddClass (see page 23996) method.
V	ImplGetDefaultComponentList (see page 24756)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetWebComponentNodeName (see page 23744)	Provides the underlying implementation for the GetWebComponentNodeName (see page 24001) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24003)	Changes the current object to the next item that has a variable name.
V	Notification (see page 2783)	Forwards notification messages to all owned components.

	NotifyAdapterChange (see page 24003)	Notifies the associated adapter when the list of items changes.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24757)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 24004)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	SetDefaultWebComponents (see page 24757)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetWebParent (see page 23744)	Introduces a method to associate this component with an adapter.
	StartIterator (see page 24006)	Makes the first child item with a variable name the current object.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24006)	Indicates the adapter that uses this object to manage its fields or actions.
R	VariablesInUse (see page 24007)	Lists those fields or actions in the collection for which variables are defined.
R	WebContainerComponentHelper (see page 24758)	Provides access to the helper object that allows this object to manage its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.

	Create (see page 24754)	Creates and initializes an instance of TWebContainerComponent (see page 24751).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24755)	Frees an instance of TWebContainerComponent (see page 24751).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetCurrentObject (see page 19719)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24000)	Returns the associated adapter.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetWebComponentNodeName (see page 24773)	This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24002)	Indicates whether this component has an associated adapter.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 24004)	Reads the component's published properties, stored data, and owned components from a reader object.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetParentComponent (see page 24005)	Sets the Adapter (see page 24006) property to be the same as this component's parent.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebActions (see page 23745)	Lists the action components that this object manages.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.6.2 TAdapterActions Methods

The methods of the TAdapterActions class are listed here.

Protected Methods

	Name	Description
	AdapterCanAddClass (see page 23742)	Indicates whether the associated adapter supports a specified class of action component.
V	ImplCanAddClass (see page 23743)	Provides the underlying implementation for the CanAddClass (see page 23996) method.
V	ImplGetWebComponentNodeName (see page 23744)	Provides the underlying implementation for the GetWebComponentNodeName (see page 24001) method.
V	SetWebParent (see page 23744)	Introduces a method to associate this component with an adapter.

173.1.6.2.1 TAdapterActions.AdapterCanAddClass Method

Indicates whether the associated adapter supports a specified class of action component.

Pascal

```
function AdapterCanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean AdapterCanAddClass(TComponent * AParent, TClass AClass);
```

Description

TAdapterActions (see page 23738) uses this method internally as part of its implementation for the CanAddClass (see page 23996) method. This allows the TAdapterActions (see page 23738) object to indicate, at design time, which types of action components are valid as components in the WebActions (see page 23745) property list.

AParent is the instance of TAdapterActions (see page 23738) to which the child actions are to be added.

AClass describes the class of the child component to be added to AParent.

AdapterCanAddClass returns true if the adapter recognizes and supports AClass as a child action, false otherwise.

AdapterCanAddClass calls the CanAddActionClass method of its associated adapter to determine whether it accepts the specified action class.

See Also

WebActions (see page 23745)

Adapter (see page 24006)

ImplCanAddClass (see page 23743)

CanAddActionClass (see page 24150)

173.1.6.2.2 TAdapterActions.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass (see page 23996) method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass (see page 23996) method calls ImplCanAddClass to determine whether an object of a specified type can be added to this collection.

AParent is the TAdapterActions (see page 23738) object to which the Web item will be added.

AClass is the class of the proposed child item.

ImplCanAddClass returns true if the TAdapterActions (see page 23738) object can include a child of type AClass, false otherwise.

ImplCanAddClass returns true if one of the following two conditions is met:

AClass is an adapter action class that the associated adapter recognizes and accepts.

AClass is a custom adapter action (a descendant of TCustomAdapterAction (see page 24188) that is not a descendant of TImplementedAdapterAction (see page 24491)).

See Also

CanAddClass (see page 23996)

AdapterCanAddClass (see page 23742)

TCustomAdapterAction (see page 24188)

TImplementedAdapterAction (see page 24491)

173.1.6.2.3 TAdapterActions.ImplGetWebComponentNodeName Method

Provides the underlying implementation for the GetWebComponentNodeName (see page 24001) method.

Pascal

```
function ImplGetWebComponentNodeName: string; override;
```

C++

```
virtual __fastcall AnsiString ImplGetWebComponentNodeName();
```

Description

The GetWebComponentNodeName (see page 24001) method calls ImplGetWebComponentNodeName to obtain a label for the type of items in the collection.

ImplGetWebComponentNodeName returns the string 'Actions'.

See Also

GetWebComponentNodeName (see page 24001)

173.1.6.2.4 TAdapterActions.SetWebParent Method

Introduces a method to associate this component with an adapter.

Pascal

```
procedure SetWebParent(const Value: TComponent); override;
```

C++

```
virtual __fastcall SetWebParent(const TComponent * Value);
```

Description

TAdapterSubComponent introduces SetWebParent as a virtual method for setting the read-only Adapter property.

Value is the adapter component that acts as a parent to this collection.

In TAdapterSubComponent, SetWebParent does nothing. Descendant classes override this method to set the Adapter property and populate the list with actions or fields appropriate to that adapter.

See Also

Adapter (see page 24006)

173.1.6.3 TAdapterActions Properties

The properties of the TAdapterActions class are listed here.

Public Properties

	Name	Description
R	WebActions (see page 23745)	Lists the action components that this object manages.

173.1.6.3.1 TAdapterActions.WebActions Property

Lists the action components that this object manages.

Pascal

```
property WebActions: TWebComponentList;
```

C++

```
__property TWebComponentList WebActions;
```

Description

Read WebActions to access the list of action components. Each component in this list is a descendant of TBaseAdapterAction (see page 24043).

See Also

TBaseAdapterAction (see page 24043)

173.1.7 TAdapterApplicationField Class

TAdapterApplicationField is a base class for adapter field components that are children of a TApplicationAdapter (see page 24032) component.

Class Hierarchy

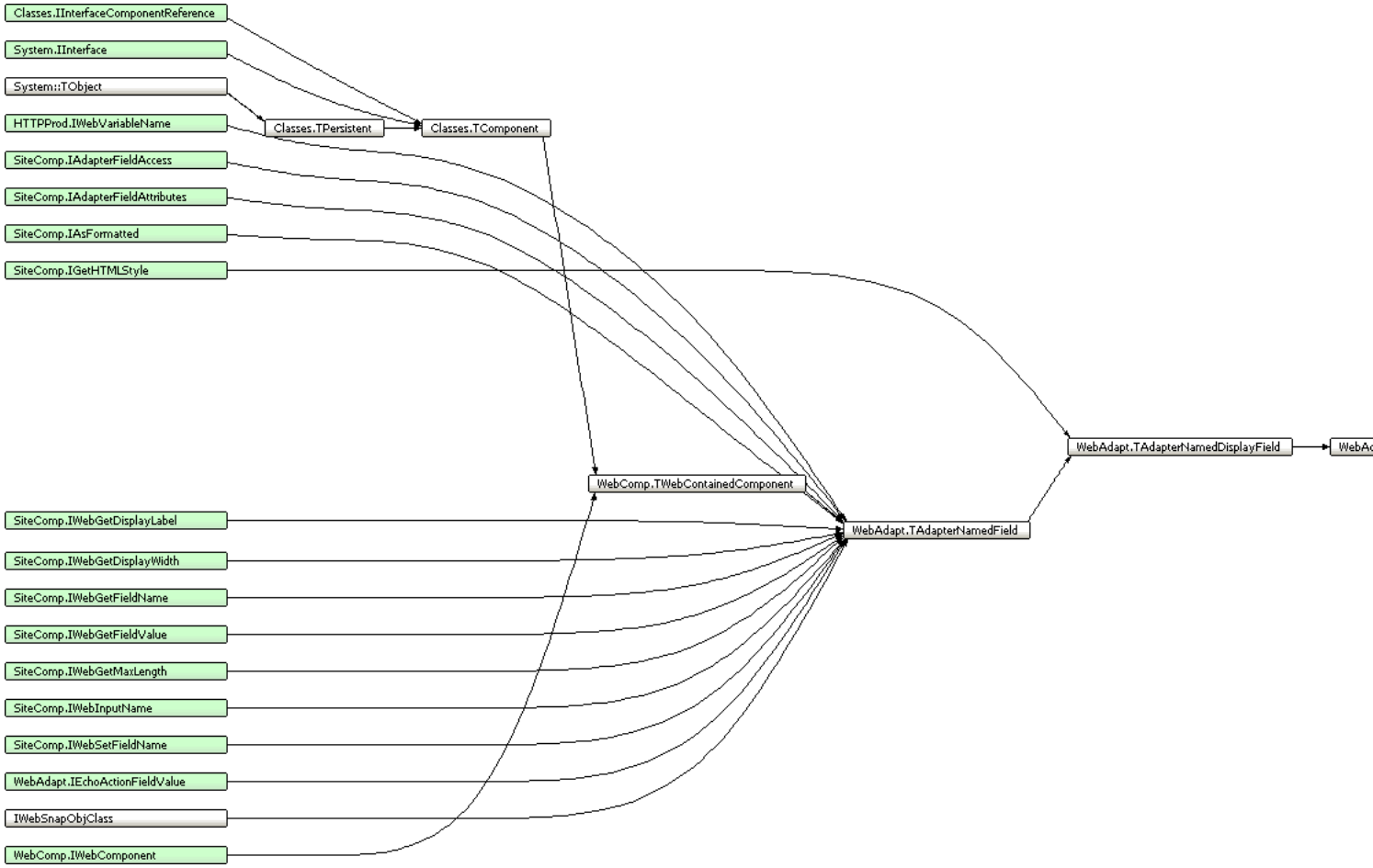

Pascal

```
TAdapterApplicationField = class(TAdapterNamedDisplayField);
```

C++

```
class TAdapterApplicationField : public TAdapterNamedDisplayField;
```

File

WebAdapt

Description

TAdapterApplicationField implements functionality useful to an adapter field that can be a child of TApplicationAdapter (see page 24032).

See Also

TApplicationAdapter (see page 24032)

TAdapterNamedDisplayField (see page 23913)

TAdapterApplicationTitleField (see page 23752)

173.1.7.1 TAdapterApplicationField Members

The following tables list the members exposed by TAdapterApplicationField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 23921)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.

	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23751)	Returns the parent adapter of this adapter field.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.

	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 23923)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23924)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 23924)	Indicates whether this adapter field is read-only.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.7.2 TAdapterApplicationField Properties

The properties of the TAdapterApplicationField class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 23751)	Returns the parent adapter of this adapter field.

173.1.7.2.1 TAdapterApplicationField.Adapter Property

Returns the parent adapter of this adapter field.

Pascal

```
property Adapter: TCustomApplicationAdapter;
```

C++

```
__property TCustomApplicationAdapter Adapter;
```

Description

Descendants use this property to access the parent adapter.

See Also

TCustomApplicationAdapter (see page 24296)

173.1.8 TAdapterApplicationTitleField Class

TAdapterApplicationTitleField is an adapter field used to access the application title.

Class Hierarchy

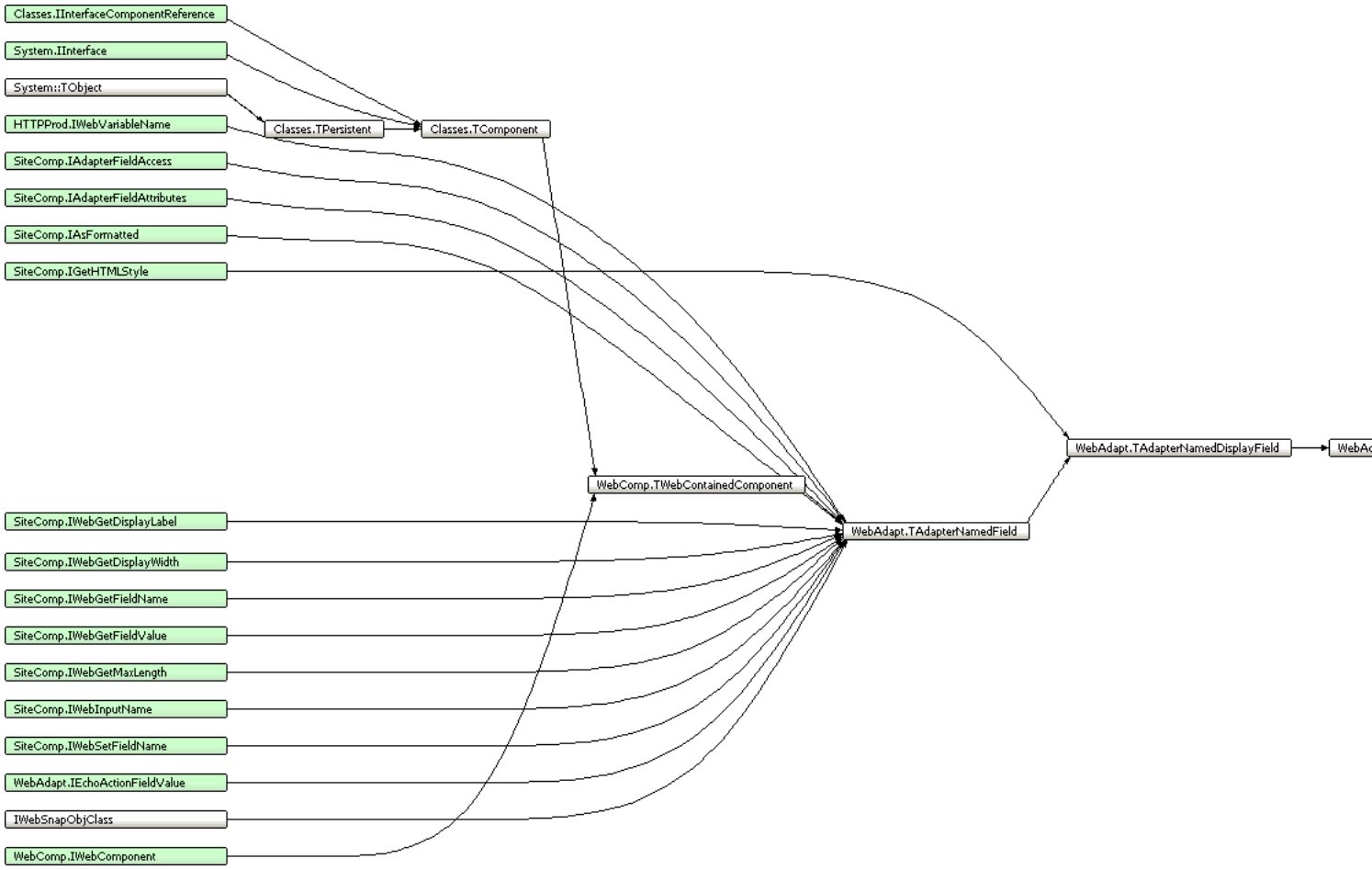

Pascal

```
TAdapterApplicationTitleField = class(TAdapterApplicationField);
```

C++

```
class TAdapterApplicationTitleField : public TAdapterApplicationField;
```

File

WebAdapt

Description

TAdapterApplicationTitleField is added to TApplicationAdapter (see page 24032).

See Also

TAdapterApplicationField (see page 23745)

TApplicationAdapter (see page 24032)

TCustomApplicationAdapter (see page 24296)

173.1.8.1 TAdapterApplicationTitleField Members

The following tables list the members exposed by TAdapterApplicationTitleField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23758)	Returns the name used to access this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 23921)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 23758)	Returns the value associated with this adapter field.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23751)	Returns the parent adapter of this adapter field.
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.

	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 23923)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23924)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 23924)	Indicates whether this adapter field is read-only.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.

	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.8.2 TAdapterApplicationTitleField Methods

The methods of the TAdapterApplicationTitleField class are listed here.

Protected Methods

	Name	Description
	GetDefaultFieldName (see page 23758)	Returns the name used to access this adapter field with server-side script.
	ImplGetValue (see page 23758)	Returns the value associated with this adapter field.

173.1.8.2.1 TAdapterApplicationTitleField.GetDefaultFieldName Method

Returns the name used to access this adapter field with server-side script.

Pascal

```
function GetDefaultFieldName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultFieldName();
```

Description

GetDefaultFieldName returns 'Title'. This allows server-side script to access this adapter field using the parent adapter's Title property.

173.1.8.2.2 TAdapterApplicationTitleField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns the value of the parent adapter's Title property. Server-side script accesses the result of this function using the DisplayText property of this adapter field. Server-side script may use this value to display the application title on a Web page.

See Also

ApplicationTitle (see page 24305)

173.1.9 TAdapterBooleanField Class

TAdapterBooleanField is an event handler based adapter field that works with a Boolean value.

Class Hierarchy

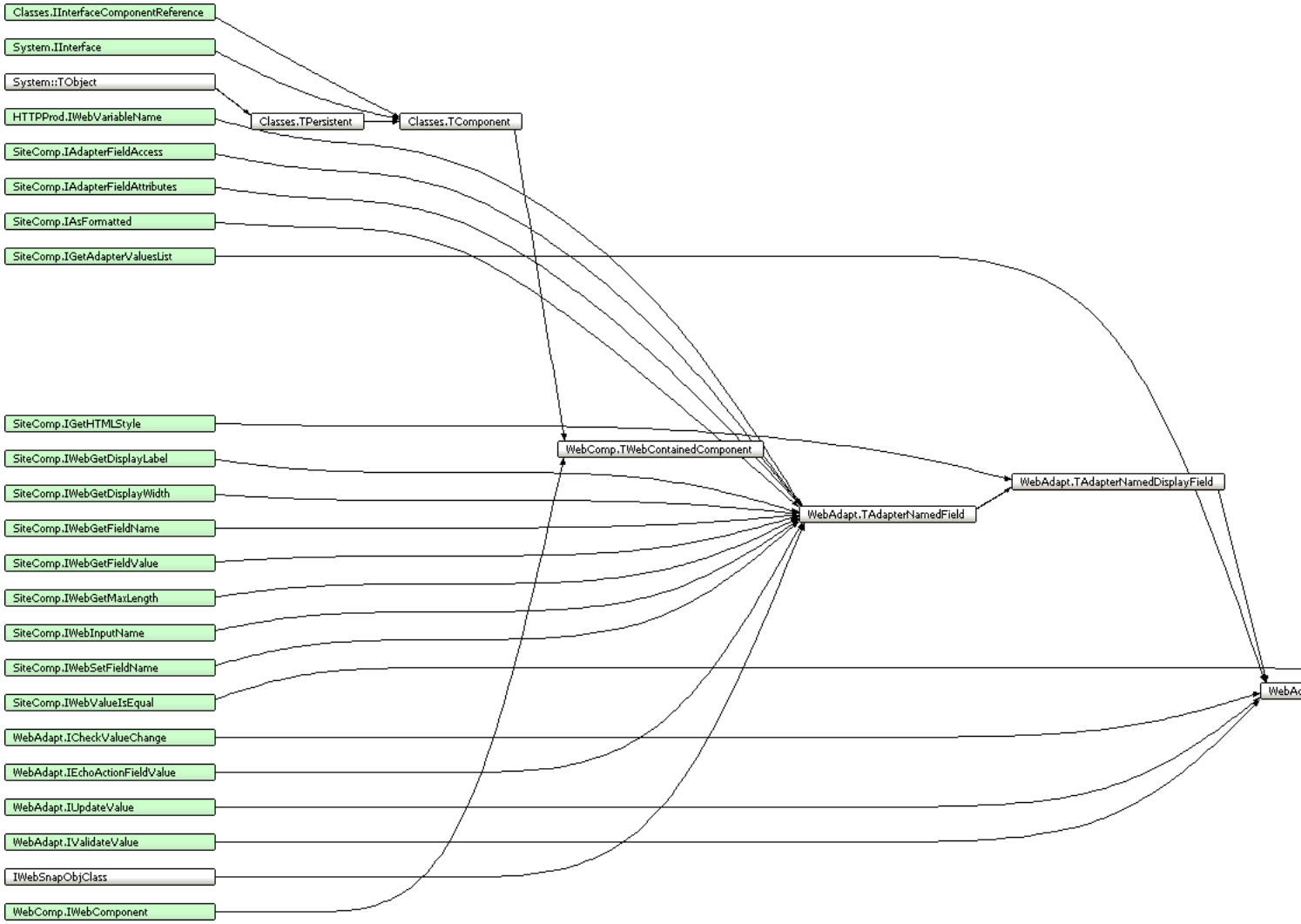

Pascal

```
TAdapterBooleanField = class(TCustomAdapterBooleanField);
```

C++

```
class TAdapterBooleanField : public TCustomAdapterBooleanField;
```

File

WebAdapt

Description

TAdapterBooleanField publishes the properties in TCustomAdapterBooleanField (see page 24201). TCustomAdapterBooleanField (see page 24201) calls event handlers to retrieve the adapter field's value, validate a value entered by the end-user, and update the adapter field's value.

Add TAdapterBooleanField to TAdapter (see page 23713) or other adapter components types when you need an adapter field that works with a Boolean value. There are other adapter field components that use event handlers. They differ in the type and number of values that are supported. See TAdapterField (see page 23818), TAdapterMultiValueField (see page 23900),

TAdapterMemoField (see page 23889), TAdapterFileField (see page 23837) and TAdapterImageField (see page 23869).

See Also

TAdapter (see page 23713)

TAdapterField (see page 23818)

TAdapterMemoField (see page 23889)

TAdapterMultiValueField (see page 23900)

TAdapterFileField (see page 23837)

TAdapterImageField (see page 23869)

173.1.9.1 TAdapterBooleanField Members

The following tables list the members exposed by TAdapterBooleanField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 24015)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 24015)	Retrieves a value entered by the end-user.
V	ConvertValue (see page 24208)	Retrieves a Boolean value entered by the end-user.
V	ConvertValueCount (see page 24015)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 24016)	Retrieves a value entered by the end-user.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoOnValidateValue (see page 24016)	Validates field values entered by the end-user.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 24016)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.

	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
	GetDisplayStyleType (see page 24208)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 23921)	Suggests an HTML input type for this adapter field.
	GetInputStyleType (see page 24017)	Indicates the preferred HTML element to use when editing this adapter field's value.
	GetInputStyleType (see page 24208)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetValueCount (see page 24017)	Provides the number of values associated with this adapter field.
	GetValues (see page 24017)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 24018)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 24018)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 24018)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 24019)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 24019)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
	ImplGetValue (see page 24209)	Returns the value of this adapter field as a variant.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 24019)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.

	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplIsEqual (see page 24209)	Indicates whether the value of this field is equal to a particular Variant value.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 24020)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 24020)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 24020)	Updates the adapter field with value(s) entered by the end-user.
	ImplUpdateValue (see page 24209)	Called to update the adapter field's value with a value entered by the end-user.
	ImplValidateValue (see page 24020)	Validates values(s) entered by the end-user.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsEqual (see page 24210)	Indicates whether the value of this field is equal to a particular Variant value.
	IsMultiValueField (see page 24021)	Indicates whether this adapter field can support multiple values.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24021)	Clears references to a deleted component.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 24021)	Updates the adapter field with value(s) entered by the end-user.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateValue (see page 24022)	Validates values(s) entered by the end-user.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 24023)	Event handler called to retrieve a string representation of the adapter field's value.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
V	Create (see page 24014)	Creates an instance of TAdapterUpdateField (see page 24007).
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterValuesList (see page 19695)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.

	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEqual (see page 19787)	This is IsEqual, a member of class IWebValueIsEqual.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	Value (see page 24022)	Retrieves the value associated with this adapter field.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetValue (see page 23769)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 23769)	Called to update the adapter field's value with a value entered by the end-user.
	OnValidateValue (see page 23769)	Event handler called to validate value(s) entered by the end-user.

Published Properties

	Name	Description
	DisplayLabel (see page 23766)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23766)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 23767)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 23767)	Identifies the rights required to update this field.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ReadOnly (see page 23767)	Indicates whether this adapter field is read-only.
	Tag (see page 2808)	Stores an integer value as part of a component.
	ValuesList (see page 23768)	Associates a list of values with the adapter field.
	ViewAccess (see page 23768)	Identifies the rights required to view this field.

173.1.9.2 TAdapterBooleanField Properties

The properties of the TAdapterBooleanField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 23766)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23766)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 23767)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 23767)	Identifies the rights required to update this field.
	ReadOnly (see page 23767)	Indicates whether this adapter field is read-only.
	ValuesList (see page 23768)	Associates a list of values with the adapter field.
	ViewAccess (see page 23768)	Identifies the rights required to view this field.

173.1.9.2.1 TAdapterBooleanField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 23933)

173.1.9.2.2 TAdapterBooleanField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 23933)

173.1.9.2.3 TAdapterBooleanField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

173.1.9.2.4 TAdapterBooleanField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property AnsiString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

hasmodifyaccess (see page 24162)

173.1.9.2.5 TAdapterBooleanField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 23921)

173.1.9.2.6 TAdapterBooleanField.ValuesList Property

Associates a list of values with the adapter field.

Pascal

```
property ValuesList: IValuesListAdapter;
```

C++

```
__property IValuesListAdapter ValuesList;
```

Description

Assign a component to the ValuesList property to define possible values for the adapter field. Server-side script can be used to enumerate the name/value pairs of the ValuesList component to generate HTML select lists, check boxes, or radio buttons.

See Also

IValuesListAdapter (see page 19740)

TStringsValuesList (see page 24540)

TDataSetValuesList (see page 8079)

173.1.9.2.7 TAdapterBooleanField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property AnsiString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

HasViewAccess (see page 24162)

173.1.9.3 TAdapterBooleanField Events

The events of the TAdapterBooleanField class are listed here.

Published Events

	Name	Description
	OnGetValue (see page 23769)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 23769)	Called to update the adapter field's value with a value entered by the end-user.
	OnValidateValue (see page 23769)	Event handler called to validate value(s) entered by the end-user.

173.1.9.3.1 TAdapterBooleanField.OnGetValue Event

Called to retrieve the adapter field's value.

Pascal

```
property OnGetValue: TAdapterFieldGetBooleanEvent;
```

C++

```
__property TAdapterFieldGetBooleanEvent OnGetValue;
```

Description

OnGetValue is called by ImplGetValue to retrieve the value associated with this adapter field.

See Also

ImplGetValue (see page 24209)

173.1.9.3.2 TAdapterBooleanField.OnUpdateValue Event

Called to update the adapter field's value with a value entered by the end-user.

Pascal

```
property OnUpdateValue: TAdapterFieldBooleanValueEvent;
```

C++

```
__property TAdapterFieldBooleanValueEvent OnUpdateValue;
```

Description

OnUpdateValue is called when an action request contains a value for this adapter field. If this event handler is not implemented, the value entered by the end-user will be ignored.

See Also

ImplUpdateValue (see page 24209)

173.1.9.3.3 TAdapterBooleanField.OnValidateValue Event

Event handler called to validate value(s) entered by the end-user.

Pascal

```
property OnValidateValue: TAdapterFieldValidateValueEvent;
```

C++

```
__property TAdapterFieldValidateValueEvent OnValidateValue;
```

Description

This event handler is called by DoOnValidateValue to validate the value(s) entered by the end-user. The event handler must raise an exception to indicate that the value(s) are not valid.

See Also

DoOnValidateValue (see page 24016)

IActionFieldValue (see page 920)

173.1.10 TAdapterDefaultActions Class

TAdapterDefaultActions manages the list of actions that belong to an adapter that supports a default set of actions.

Class Hierarchy

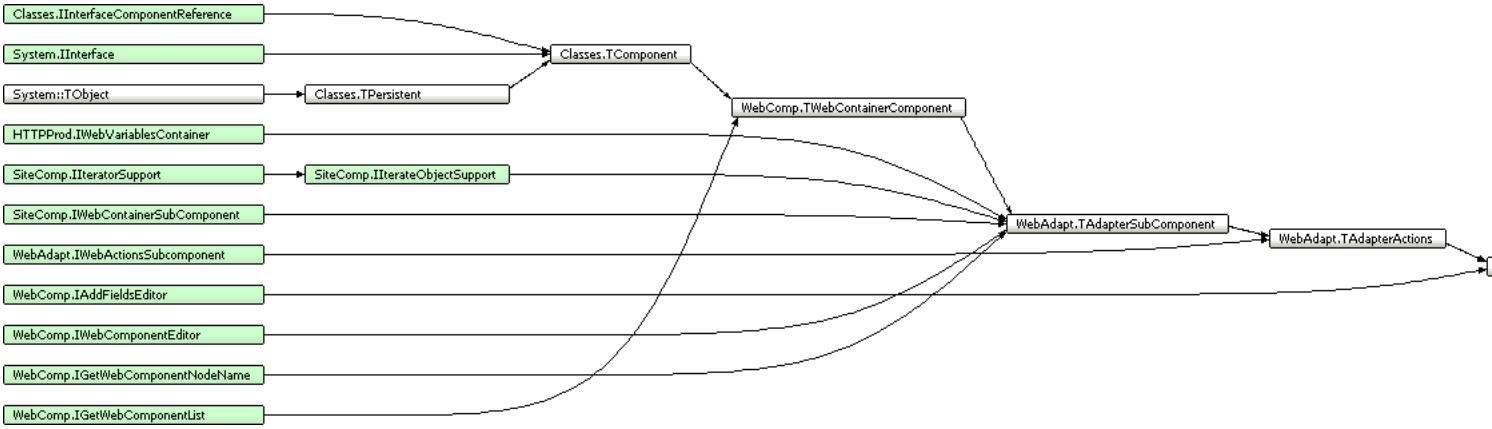

Pascal

```
TAdapterDefaultActions = class(TAdapterActions, IAddFieldsEditor);
```

C++

```
class TAdapterDefaultActions : public TAdapterActions, public IAddFieldsEditor;
```

File

WebAdapt

Description

TAdapterDefaultActions is the type of the Actions property on an adapter component. It manages two lists of actions (TBaseAdapterAction (see page 24043) descendants) that belong to the adapter: the list of actions that have been explicitly added to the adapter, and the list of actions that the adapter includes by default. The explicitly added action objects are available using the WebActions (see page 23745) property. The default actions are available using the DefaultWebComponents (see page 23778) property.

See Also

TDefaultFieldsAdapter (see page 24392)

TAdapterFields (see page 23830)

TAdapterActions (see page 23738)

TAdapterDefaultFields (see page 23778)

TBaseAdapterAction (see page 24043)

173.1.10.1 TAdapterDefaultActions Members

The following tables list the members exposed by TAdapterDefaultActions.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AdapterCanAddClass (see page 23742)	Indicates whether the associated adapter supports a specified class of action component.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 23996)	Indicates whether an object of a specified type can be added to this collection.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EndIterator (see page 23997)	Stops the TAdapterSubComponent (see page 23991) object from iterating over the Web items it lists.
V	FindNamedVariable (see page 23997)	Returns the item from this list with a specified variable name.
	FindVariable (see page 23998)	Returns the item from this list with a specified variable name.
	FindVariableContainer (see page 23998)	Returns the Web module that contains a specified item from this list.
	GetAddAllFieldsItem (see page 23775)	This is GetAddAllFieldsItem, a member of class TAdapterDefaultActions.
	GetAddFieldsDlgCaption (see page 23776)	This is GetAddFieldsDlgCaption, a member of class TAdapterDefaultActions.
	GetAddFieldsItem (see page 23776)	This is GetAddFieldsItem, a member of class TAdapterDefaultActions.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24755)	Executes a callback for each child Web item.
	GetCurrentObject (see page 23999)	Returns the current field or action from the list when an enumerator is iterating over child items.
V	GetDefaultWebComponents (see page 23776)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetNewFieldPrefix (see page 23776)	This is GetNewFieldPrefix, a member of class TAdapterDefaultActions.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariable (see page 24000)	Returns a child item from this list, given its position in the list of child items with variable names.
	GetVariableCount (see page 24000)	Indicates the number of items in this list that can be accessed from server-side script.
V	GetVariablesInUse (see page 23777)	Returns the value of the VariablesInUse property.
	GetWebComponentNodeName (see page 24001)	Returns a string that indicates the type of items in this collection.
	GetWebComponents (see page 24756)	Returns the list of Web items that this item manages.
V	ImplCanAddClass (see page 23743)	Provides the underlying implementation for the CanAddClass (see page 23996) method.
V	ImplCanAddClass (see page 24002)	Provides the underlying implementation for the CanAddClass (see page 23996) method.
V	ImplGetDefaultComponentList (see page 24756)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetWebComponentNodeName (see page 23744)	Provides the underlying implementation for the GetWebComponentNodeName (see page 24001) method.

V	ImplGetWebComponentNodeName (see page 24003)	Provides the underlying implementation for the GetWebComponentNodeName (see page 24001) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24003)	Changes the current object to the next item that has a variable name.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	NotifyAdapterChange (see page 24003)	Notifies the associated adapter when the list of items changes.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24757)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 23777)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetWebParent (see page 23744)	Introduces a method to associate this component with an adapter.
V	SetWebParent (see page 24005)	Introduces a method to associate this component with an adapter.
	StartIterator (see page 24006)	Makes the first child item with a variable name the current object.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24006)	Indicates the adapter that uses this object to manage its fields or actions.
R	VariablesInUse (see page 24007)	Lists those fields or actions in the collection for which variables are defined.
R	WebContainerComponentHelper (see page 24758)	Provides access to the helper object that allows this object to manage its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.

	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	ClearDefaultComponents (see page 23775)	Empties the list of default actions that is the value of the DefaultWebComponents (see page 23778) property.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24754)	Creates and initializes an instance of TWebContainerComponent (see page 24751).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 23774)	Frees an instance of TAdapterDefaultActions (see page 23770).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetCurrentObject (see page 19719)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24000)	Returns the associated adapter.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetWebComponentNodeName (see page 24773)	This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24002)	Indicates whether this component has an associated adapter.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 24004)	Reads the component's published properties, stored data, and owned components from a reader object.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetParentComponent (see page 24005)	Sets the Adapter (see page 24006) property to be the same as this component's parent.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	DefaultWebComponents (see page 23778)	Lists the action components that the associated adapter includes by default.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebActions (see page 23745)	Lists the action components that this object manages.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.10.2 TAdapterDefaultActions.Destroy Destructor

Frees an instance of TAdapterDefaultActions (see page 23770).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterDefaultActions();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks for a nil reference before calling Destroy.

See Also

Free (see page 22300)

173.1.10.3 TAdapterDefaultActions Methods

The methods of the TAdapterDefaultActions class are listed here.

Protected Methods

	Name	Description
	GetAddAllFieldsItem (see page 23775)	This is GetAddAllFieldsItem, a member of class TAdapterDefaultActions.
	GetAddFieldsDlgCaption (see page 23776)	This is GetAddFieldsDlgCaption, a member of class TAdapterDefaultActions.
	GetAddFieldsItem (see page 23776)	This is GetAddFieldsItem, a member of class TAdapterDefaultActions.
V	GetDefaultWebComponents (see page 23776)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
	GetNewFieldPrefix (see page 23776)	This is GetNewFieldPrefix, a member of class TAdapterDefaultActions.
V	GetVariablesInUse (see page 23777)	Returns the value of the VariablesInUse property.
V	SetDefaultWebComponents (see page 23777)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Public Methods

	Name	Description
V	ClearDefaultComponents (see page 23775)	Empties the list of default actions that is the value of the DefaultWebComponents (see page 23778) property.

173.1.10.3.1 TAdapterDefaultActions.ClearDefaultComponents Method

Empties the list of default actions that is the value of the DefaultWebComponents (see page 23778) property.

Pascal

```
procedure ClearDefaultComponents; virtual;
```

C++

```
virtual __fastcall ClearDefaultComponents();
```

Description

The associated adapter calls ClearDefaultComponents when it changes its set of default actions. After a call to ClearDefaultComponents, the adapter generates a new set of default components the next time the application reads the default actions.

See Also

DefaultWebComponents (see page 23778)

173.1.10.3.2 TAdapterDefaultActions.GetAddAllFieldsItem Method

Pascal

```
function GetAddAllFieldsItem: string;
```

C++

```
__fastcall AnsiString GetAddAllFieldsItem();
```

Description

This is GetAddAllFieldsItem, a member of class TAdapterDefaultActions.

173.1.10.3.3 TAdapterDefaultActions.GetAddFieldsDlgCaption Method

Pascal

```
function GetAddFieldsDlgCaption: string;
```

C++

```
__fastcall AnsiString GetAddFieldsDlgCaption();
```

Description

This is GetAddFieldsDlgCaption, a member of class TAdapterDefaultActions.

173.1.10.3.4 TAdapterDefaultActions.GetAddFieldsItem Method

Pascal

```
function GetAddFieldsItem: string;
```

C++

```
__fastcall AnsiString GetAddFieldsItem();
```

Description

This is GetAddFieldsItem, a member of class TAdapterDefaultActions.

173.1.10.3.5 TAdapterDefaultActions.GetDefaultWebComponents Method

Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function GetDefaultWebComponents: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents supplies the default set of child items for this component. As implemented in TAdapterSubComponent, GetDefaultWebComponents always returns nil (Delphi) or NULL (C++) because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions and TAdapterDefaultFields) override this method to return the list of default fields or actions.

See Also

GetWebComponents (see page 24756)

TAdapterDefaultActions (see page 23770)

TAdapterDefaultFields (see page 23778)

173.1.10.3.6 TAdapterDefaultActions.GetNewFieldPrefix Method

Pascal

```
function GetNewFieldPrefix: string;
```

C++

```
__fastcall AnsiString GetNewFieldPrefix();
```

Description

This is GetNewFieldPrefix, a member of class TAdapterDefaultActions.

173.1.10.3.7 TAdapterDefaultActions.GetVariablesInUse Method

Returns the value of the VariablesInUse property.

Pascal

```
function GetVariablesInUse: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetVariablesInUse();
```

Description

GetVariablesInUse is the protected read implementation for the VariablesInUse property.

In TAdapterSubComponent, GetVariablesInUse calls the GetWebComponents method to return a list of all the items in the list. Descendant classes can override this method to return only those items that meet a certain criterion.

See Also

VariablesInUse (see page 24007)

GetWebComponents (see page 24756)

173.1.10.3.8 TAdapterDefaultActions.SetDefaultWebComponents Method

Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList); override;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

GetDefaultWebComponents supplies the default set of child items for this component. As implemented in TAdapterSubComponent, GetDefaultWebComponents always returns nil (Delphi) or NULL (C++) because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions and TAdapterDefaultFields) override this method to return the list of default fields or actions.

SetDefaultWebComponents assigns the default set of child items for this component. As implemented in TWebContainerComponent, SetDefaultWebComponents does nothing because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions and TAdapterDefaultFields) override this method to store the list of default fields or actions.

See Also

GetDefaultWebComponents (see page 23999)

IsDefaultField (see page 24728)

173.1.10.4 TAdapterDefaultActions Properties

The properties of the TAdapterDefaultActions class are listed here.

Public Properties

	Name	Description
R	DefaultWebComponents (see page 23778)	Lists the action components that the associated adapter includes by default.

173.1.10.4.1 TAdapterDefaultActions.DefaultWebComponents Property

Lists the action components that the associated adapter includes by default.

Pascal

```
property DefaultWebComponents: TWebComponentList;
```

C++

```
__property TWebComponentList DefaultWebComponents;
```

Description

Read DefaultWebComponents to access the list of action components that the adapter uses when you do not explicitly add any actions to this object. Each default action is a descendant of TAdapterNamedAction.

If you explicitly add action components to this object, the adapter uses the actions in the WebActions (see page 23745) property instead.

See Also

TBaseAdapterAction (see page 24043)

WebActions (see page 23745)

Adapter (see page 24006)

173.1.11 TAdapterDefaultFields Class

TAdapterDefaultFields manages the list of fields that belong to an adapter that supports a default set of fields.

Class Hierarchy

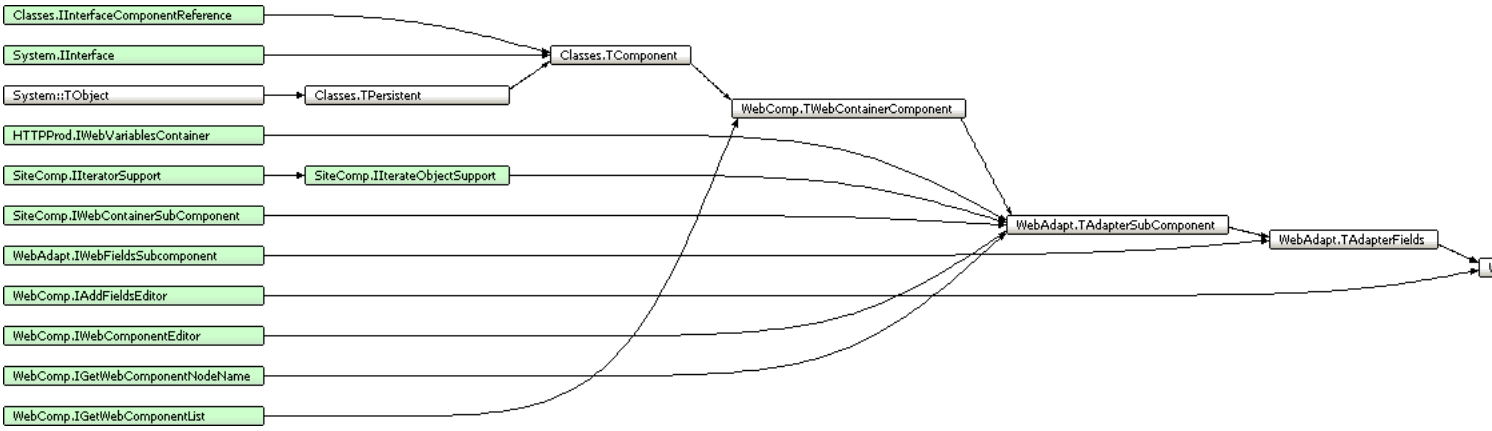

Pascal

```
TAdapterDefaultFields = class(TAdapterFields, IAddFieldsEditor);
```

C++

```
class TAdapterDefaultFields : public TAdapterFields, public IAddFieldsEditor;
```

File

WebAdapt

Description

TAdapterDefaultFields is the type of the Data property on an adapter component. It manages two lists of fields (TAdapterNamedField (see page 23925) descendants) that belong to the adapter: the list of fields that have been explicitly added to the adapter, and the list of fields that the adapter includes by default. The explicitly added field objects are available using the WebFields (see page 23837) property. The default fields are available using the DefaultWebComponents (see page 23786) property.

See Also

TDefaultFieldsAdapter (see page 24392)

TAdapterActions (see page 23738)

TAdapterDefaultActions (see page 23770)

TAdapterFields (see page 23830)

TAdapterNamedField (see page 23925)

173.1.11.1 TAdapterDefaultFields Members

The following tables list the members exposed by TAdapterDefaultFields.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AdapterCanAddClass (see page 23835)	Indicates whether the associated adapter supports a specified class of field component.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 23996)	Indicates whether an object of a specified type can be added to this collection.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EndIterator (see page 23997)	Stops the TAdapterSubComponent (see page 23991) object from iterating over the Web items it lists.
V	FindNamedVariable (see page 23997)	Returns the item from this list with a specified variable name.
	FindVariable (see page 23998)	Returns the item from this list with a specified variable name.
	FindVariableContainer (see page 23998)	Returns the Web module that contains a specified item from this list.
	GetAddAllFieldsItem (see page 23784)	This is GetAddAllFieldsItem, a member of class TAdapterDefaultFields.
	GetAddFieldsDlgCaption (see page 23784)	This is GetAddFieldsDlgCaption, a member of class TAdapterDefaultFields.
	GetAddFieldsItem (see page 23784)	This is GetAddFieldsItem, a member of class TAdapterDefaultFields.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.

	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24755)	Executes a callback for each child Web item.
	GetCurrentObject (see page 23999)	Returns the current field or action from the list when an enumerator is iterating over child items.
V	GetDefaultWebComponents (see page 23785)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetNewFieldPrefix (see page 23785)	This is GetNewFieldPrefix, a member of class TAdapterDefaultFields.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariable (see page 24000)	Returns a child item from this list, given its position in the list of child items with variable names.
	GetVariableCount (see page 24000)	Indicates the number of items in this list that can be accessed from server-side script.
V	GetVariablesInUse (see page 23785)	Returns the value of the VariablesInUse property.
	GetWebComponentNodeName (see page 24001)	Returns a string that indicates the type of items in this collection.
	GetWebComponents (see page 24756)	Returns the list of Web items that this item manages.
V	ImplCanAddClass (see page 23836)	Provides the underlying implementation for the CanAddClass (see page 23996) method.
V	ImplCanAddClass (see page 24002)	Provides the underlying implementation for the CanAddClass (see page 23996) method.
V	ImplGetDefaultComponentList (see page 24756)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetWebComponentNodeName (see page 23836)	Provides the underlying implementation for the GetWebComponentNodeName (see page 24001) method.
V	ImplGetWebComponentNodeName (see page 24003)	Provides the underlying implementation for the GetWebComponentNodeName (see page 24001) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24003)	Changes the current object to the next item that has a variable name.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	NotifyAdapterChange (see page 24003)	Notifies the associated adapter when the list of items changes.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24757)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 23786)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetWebParent (see page 23836)	Associates this component with an adapter.
V	SetWebParent (see page 24005)	Introduces a method to associate this component with an adapter.
	StartIterator (see page 24006)	Makes the first child item with a variable name the current object.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Adapter (see page 24006)	Indicates the adapter that uses this object to manage its fields or actions.
	VariablesInUse (see page 24007)	Lists those fields or actions in the collection for which variables are defined.
	WebContainerComponentHelper (see page 24758)	Provides access to the helper object that allows this object to manage its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDefaultComponents (see page 23784)	Empties the list of default fields that is the value of the DefaultWebComponents (see page 23786) property.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 24754)	Creates and initializes an instance of TWebContainerComponent (see page 24751).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 23783)	Frees an instance of TAdapterDefaultFields (see page 23778).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetCurrentObject (see page 19719)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24000)	Returns the associated adapter.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetWebComponentNodeName (see page 24773)	This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24002)	Indicates whether this component has an associated adapter.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 24004)	Reads the component's published properties, stored data, and owned components from a reader object.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetParentComponent (see page 24005)	Sets the Adapter (see page 24006) property to be the same as this component's parent.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.

R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	DefaultWebComponents (see page 23786)	Lists the field components that the associated adapter includes by default.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebFields (see page 23837)	Lists the field components that this object manages.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.11.2 TAdapterDefaultFields.Destroy Destructor

Frees an instance of TAdapterDefaultFields (see page 23778).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterDefaultFields();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks for a nil reference before calling Destroy.

See Also

Free (see page 22300)

173.1.11.3 TAdapterDefaultFields Methods

The methods of the TAdapterDefaultFields class are listed here.

Protected Methods

	Name	Description
	GetAddAllFieldsItem (see page 23784)	This is GetAddAllFieldsItem, a member of class TAdapterDefaultFields.
	GetAddFieldsDlgCaption (see page 23784)	This is GetAddFieldsDlgCaption, a member of class TAdapterDefaultFields.
	GetAddFieldsItem (see page 23784)	This is GetAddFieldsItem, a member of class TAdapterDefaultFields.
V	GetDefaultWebComponents (see page 23785)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
	GetNewFieldPrefix (see page 23785)	This is GetNewFieldPrefix, a member of class TAdapterDefaultFields.
V	GetVariablesInUse (see page 23785)	Returns the value of the VariablesInUse property.
V	SetDefaultWebComponents (see page 23786)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Public Methods

	Name	Description
V	ClearDefaultComponents (see page 23784)	Empties the list of default fields that is the value of the DefaultWebComponents (see page 23786) property.

173.1.11.3.1 TAdapterDefaultFields.ClearDefaultComponents Method

Empties the list of default fields that is the value of the DefaultWebComponents (see page 23786) property.

Pascal

```
procedure ClearDefaultComponents; virtual;
```

C++

```
virtual __fastcall ClearDefaultComponents();
```

Description

The associated adapter calls ClearDefaultComponents when it changes its set of default fields. After a call to ClearDefaultComponents, the adapter generates a new set of default components the next time the application reads the default fields.

See Also

DefaultWebComponents (see page 23786)

173.1.11.3.2 TAdapterDefaultFields.GetAddAllFieldsItem Method

Pascal

```
function GetAddAllFieldsItem: string;
```

C++

```
__fastcall AnsiString GetAddAllFieldsItem();
```

Description

This is GetAddAllFieldsItem, a member of class TAdapterDefaultFields.

173.1.11.3.3 TAdapterDefaultFields.GetAddFieldsDlgCaption Method

Pascal

```
function GetAddFieldsDlgCaption: string;
```

C++

```
__fastcall AnsiString GetAddFieldsDlgCaption();
```

Description

This is GetAddFieldsDlgCaption, a member of class TAdapterDefaultFields.

173.1.11.3.4 TAdapterDefaultFields.GetAddFieldsItem Method

Pascal

```
function GetAddFieldsItem: string;
```

C++

```
__fastcall AnsiString GetAddFieldsItem();
```

Description

This is GetAddFieldsItem, a member of class TAdapterDefaultFields.

173.1.11.3.5 TAdapterDefaultFields.GetDefaultWebComponents Method

Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function GetDefaultWebComponents: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents supplies the default set of child items for this component. As implemented in TAdapterSubComponent, GetDefaultWebComponents always returns nil (Delphi) or NULL (C++) because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions and TAdapterDefaultFields) override this method to return the list of default fields or actions.

See Also

GetWebComponents (see page 24756)

TAdapterDefaultActions (see page 23770)

TAdapterDefaultFields (see page 23778)

173.1.11.3.6 TAdapterDefaultFields.GetNewFieldPrefix Method

Pascal

```
function GetNewFieldPrefix: string;
```

C++

```
__fastcall AnsiString GetNewFieldPrefix();
```

Description

This is GetNewFieldPrefix, a member of class TAdapterDefaultFields.

173.1.11.3.7 TAdapterDefaultFields.GetVariablesInUse Method

Returns the value of the VariablesInUse property.

Pascal

```
function GetVariablesInUse: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetVariablesInUse();
```

Description

GetVariablesInUse is the protected read implementation for the VariablesInUse property.

In TAdapterSubComponent, GetVariablesInUse calls the GetWebComponents method to return a list of all the items in the list. Descendant classes can override this method to return only those items that meet a certain criterion.

See Also

VariablesInUse (see page 24007)

GetWebComponents (see page 24756)

173.1.11.3.8 TAdapterDefaultFields.SetDefaultWebComponents Method

Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList); override;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

GetDefaultWebComponents supplies the default set of child items for this component. As implemented in TAdapterSubComponent, GetDefaultWebComponents always returns nil (Delphi) or NULL (C++) because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions and TAdapterDefaultFields) override this method to return the list of default fields or actions.

SetDefaultWebComponents assigns the default set of child items for this component. As implemented in TWebContainerComponent, SetDefaultWebComponents does nothing because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions and TAdapterDefaultFields) override this method to store the list of default fields or actions.

See Also

GetDefaultWebComponents (see page 23999)

IsDefaultField (see page 24728)

173.1.11.4 TAdapterDefaultFields Properties

The properties of the TAdapterDefaultFields class are listed here.

Public Properties

	Name	Description
R	DefaultWebComponents (see page 23786)	Lists the field components that the associated adapter includes by default.

173.1.11.4.1 TAdapterDefaultFields.DefaultWebComponents Property

Lists the field components that the associated adapter includes by default.

Pascal

```
property DefaultWebComponents: TWebComponentList;
```

C++

```
__property TWebComponentList DefaultWebComponents;
```

Description

Read DefaultWebComponents to access the list of field components that the adapter uses when you do not explicitly add any field components to this object. Each default field component is a descendant of TAdapterNamedField (see page 23925).

If you explicitly add field components to this object, the adapter uses the fields in the WebFields (see page 23837) property instead.

See Also

TAdapterNamedField (see page 23925)

WebFields (see page 23837)

Adapter (see page 24006)

173.1.12 TAdapterEndUserDisplayNameField Class

TAdapterEndUserDisplayNameField is an adapter field used to access the end-user's name.

Class Hierarchy

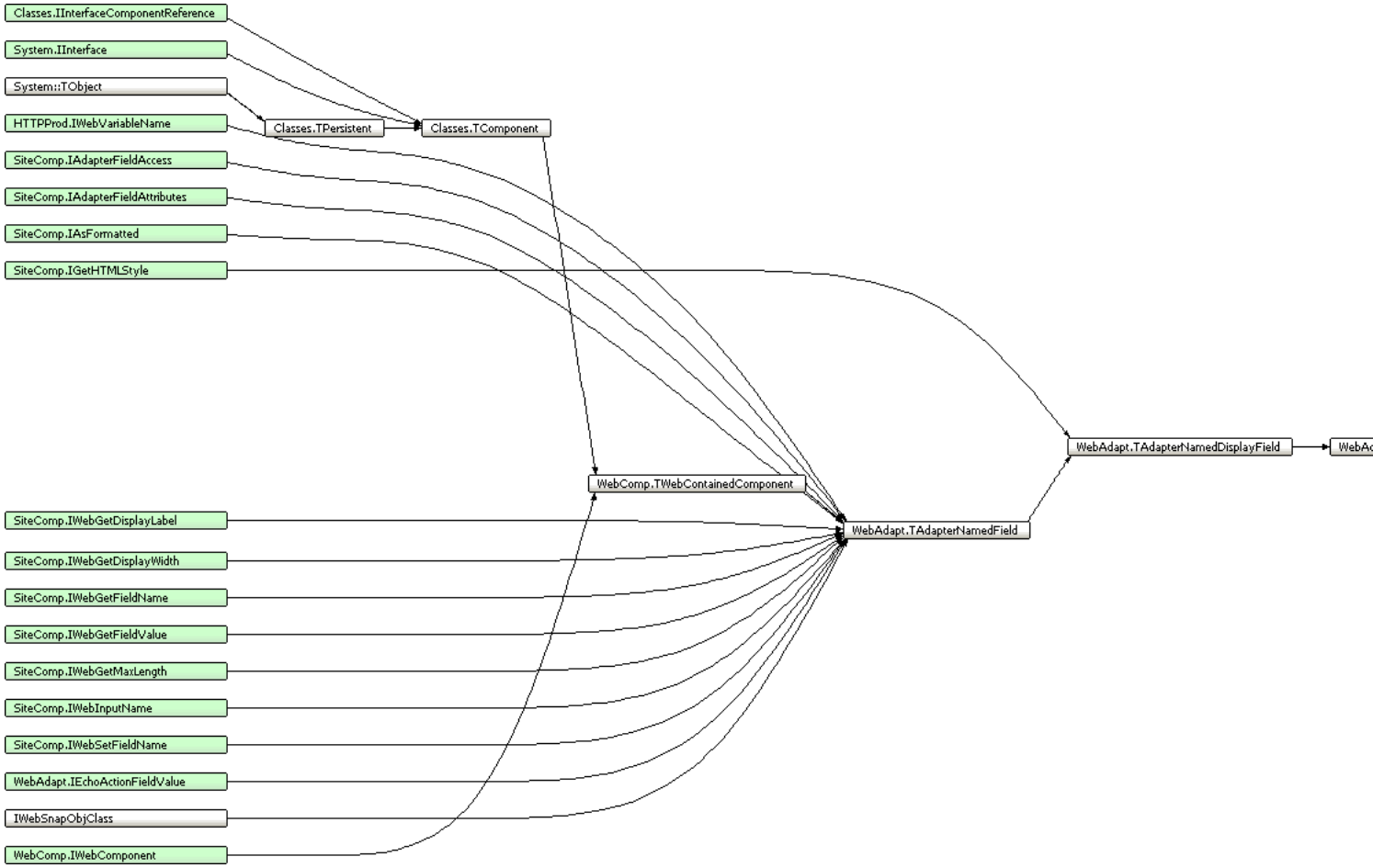

Pascal

```
TAdapterEndUserDisplayNameField = class(TAdapterEndUserField);
```

C++

```
class TAdapterEndUserDisplayNameField : public TAdapterEndUserField;
```

File

WebAdapt

Description

TAdapterEndUserDisplayNameField is added to TApplicationAdapter (see page 24032).

See Also

TAdapterEndUserField (see page 23793)

TEndUserAdapter (see page 24421)

TEndUserSessionAdapter (see page 24458)

173.1.12.1 TAdapterEndUserDisplayNameField Members

The following tables list the members exposed by TAdapterEndUserDisplayNameField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23793)	Returns the name used to access this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 23921)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
V	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
V	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
V	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
V	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 23793)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23800)	Returns the parent adapter of this adapter field.
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.

	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 23923)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23924)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 23924)	Indicates whether this adapter field is read-only.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.12.2 TAdapterEndUserDisplayNameField Methods

The methods of the TAdapterEndUserDisplayNameField class are listed here.

Protected Methods

	Name	Description
V	GetDefaultFieldName (see page 23793)	Returns the name used to access this adapter field with server-side script.

	ImplGetValue (see page 23793)	Returns the value associated with this adapter field.

173.1.12.2.1 TAdapterEndUserDisplayNameField.GetDefaultFieldName Method

Returns the name used to access this adapter field with server-side script.

Pascal

```
function GetDefaultFieldName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultFieldName();
```

Description

GetDefaultFieldName returns 'DisplayName'. This allows server-side script to access this adapter field using the parent adapter's DisplayName property.

173.1.12.2.2 TAdapterEndUserDisplayNameField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns the result of GetDisplayNameValue. Server-side script accesses the result of this function using the DisplayText property of this adapter field. Server-side script may use this value to display the end-user's name on a Web page.

See Also

GetDisplayNameValue (see page 24109)

173.1.13 TAdapterEndUserField Class

TAdapterEndUserField is a base class for adapter field components that are children of an end-user adapter component.

Class Hierarchy

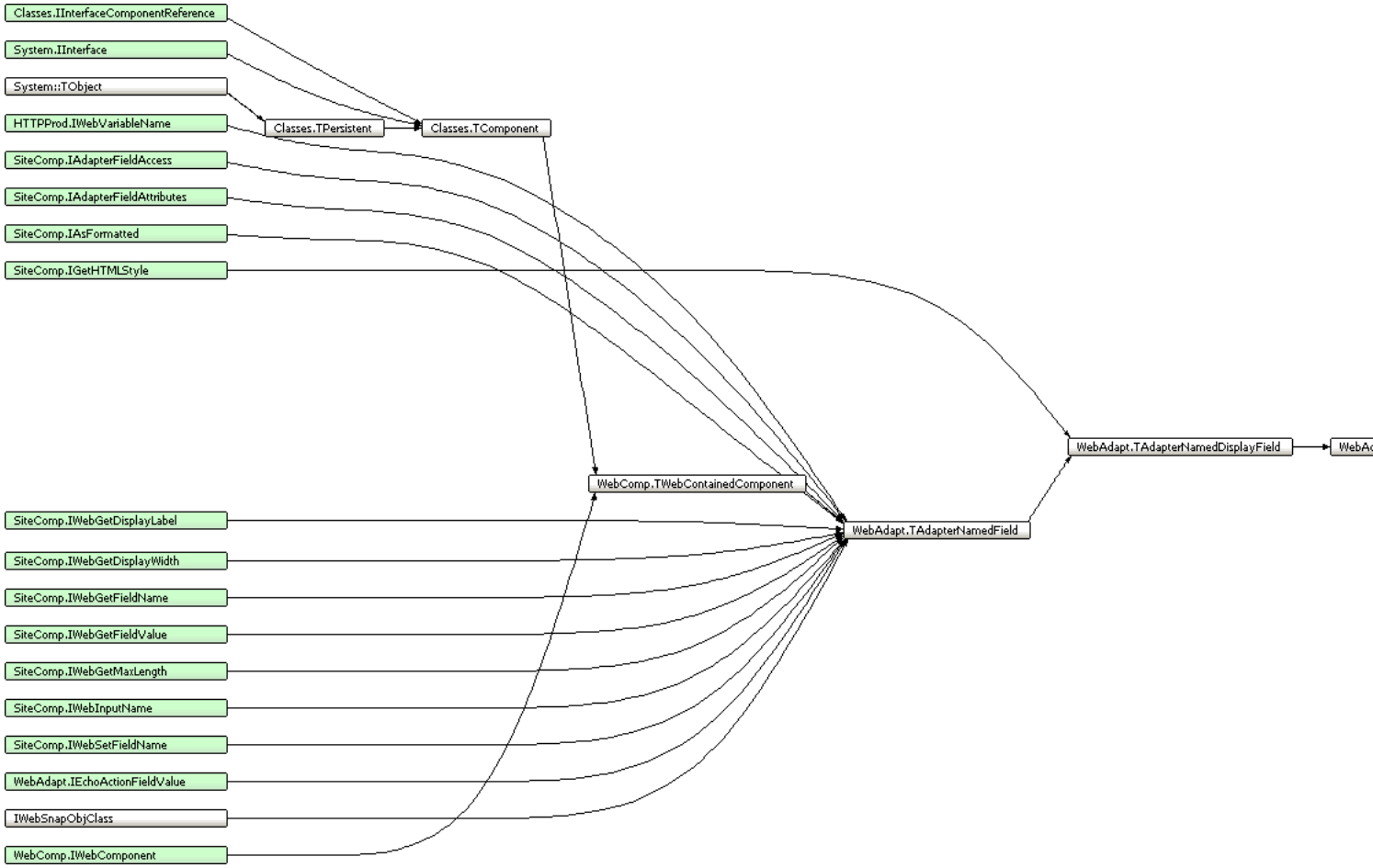

Pascal

```
TAdapterEndUserField = class(TAdapterNamedDisplayField);
```

C++

```
class TAdapterEndUserField : public TAdapterNamedDisplayField;
```

File

WebAdapt

Description

TAdapterEndUserField implements functionality useful to an adapter field that can be a child of TEndUserAdapter (see page 24421) or TEndUserSessionAdapter (see page 24458).

See Also

TAdapterNamedDisplayField (see page 23913)

TBaseEndUserAdapter (see page 24097)

TEndUserAdapter (see page 24421)

TEndUserSessionAdapter (see page 24458)

TAdapterEndUserLoggedInField (see page 23800)

TAdapterEndUserDisplayNameField (see page 23787)

173.1.13.1 TAdapterEndUserField Members

The following tables list the members exposed by TAdapterEndUserField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 23921)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.

	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
V	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
V	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
V	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
V	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
V	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23800)	Returns the parent adapter of this adapter field.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 23923)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23924)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 23924)	Indicates whether this adapter field is read-only.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.13.2 TAdapterEndUserField Properties

The properties of the TAdapterEndUserField class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 23800)	Returns the parent adapter of this adapter field.

173.1.13.2.1 TAdapterEndUserField.Adapter Property

Returns the parent adapter of this adapter field.

Pascal

```
property Adapter: TBaseEndUserAdapter;
```

C++

```
__property TBaseEndUserAdapter Adapter;
```

Description

Descendants use this property to access the parent adapter.

See Also

TBaseEndUserAdapter (see page 24097)

173.1.14 TAdapterEndUserLoggedInField Class

TAdapterEndUserLoggedInField is an adapter field used to access the end-user logged in flag.

Class Hierarchy

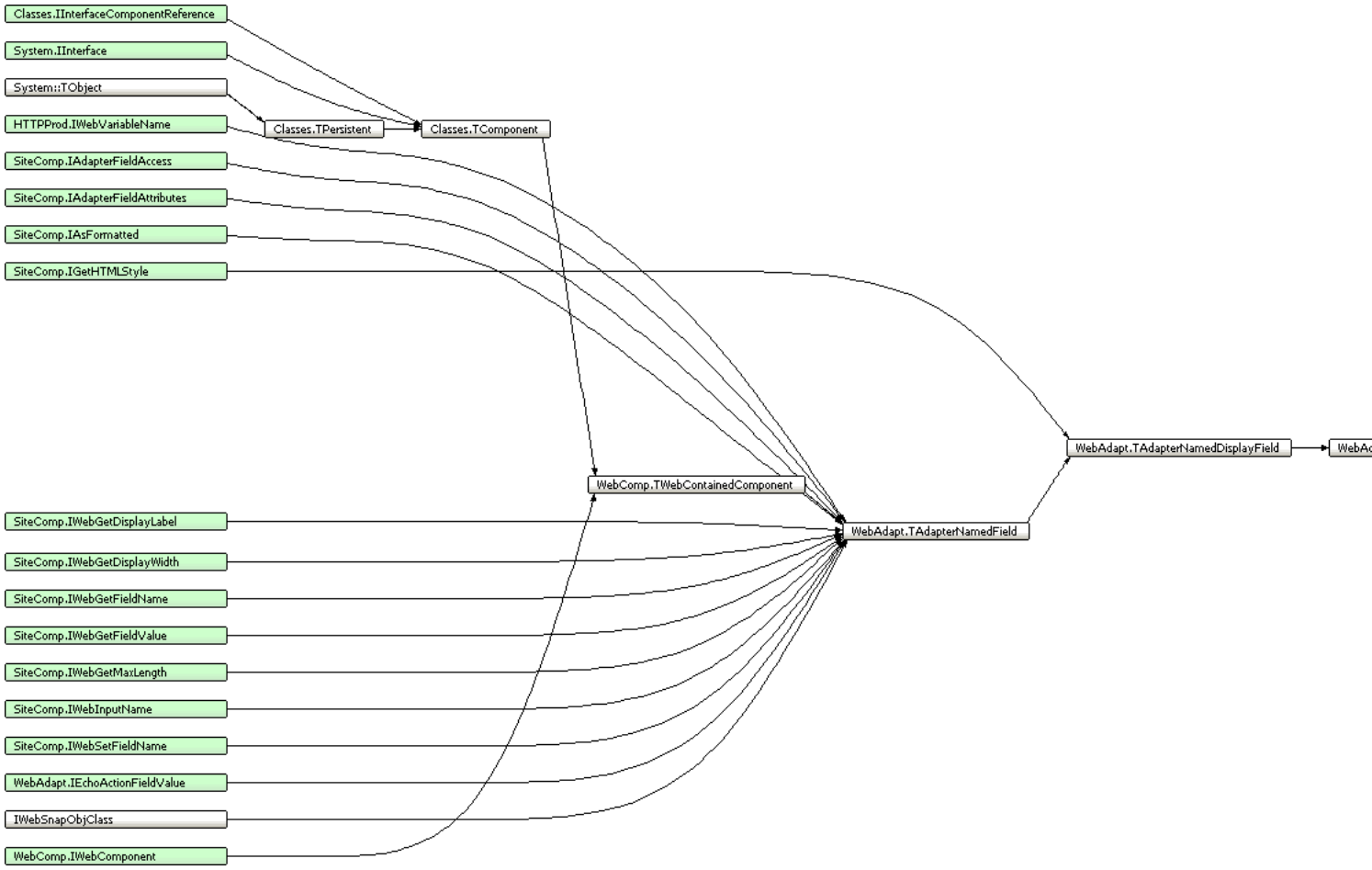

Pascal

```
TAdapterEndUserLoggedInField = class(TAdapterEndUserField);
```

C++

```
class TAdapterEndUserLoggedInField : public TAdapterEndUserField;
```

File

WebAdapt

Description

TAdapterEndUserLoggedInField is added to TApplicationAdapter (see page 24032). The logged-in flag indicates whether the user is logged-in. This value may be used by server-side script to conditionally generate HTML. For example, when the end-user is logged in, a logout hyperlink is displayed. When the end-user is not logged in, a logout hyperlink is not displayed.

See Also

TApplicationAdapter (see page 24032)

TEndUserAdapter (see page 24421)

TEndUserSessionAdapter (see page 24458)

173.1.14.1 TAdapterEndUserLoggedInField Members

The following tables list the members exposed by TAdapterEndUserLoggedInField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23807)	Returns the name used to access this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 23921)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.

	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
V	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
V	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
V	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
V	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 23807)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23800)	Returns the parent adapter of this adapter field.
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.

	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 23923)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23924)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 23924)	Indicates whether this adapter field is read-only.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.14.2 TAdapterEndUserLoggedInField Methods

The methods of the TAdapterEndUserLoggedInField class are listed here.

Protected Methods

	Name	Description
V	GetDefaultFieldName (see page 23807)	Returns the name used to access this adapter field with server-side script.
V	ImplGetValue (see page 23807)	Returns the value associated with this adapter field.

173.1.14.2.1 TAdapterEndUserLoggedInField.GetDefaultFieldName Method

Returns the name used to access this adapter field with server-side script.

Pascal

```
function GetDefaultFieldName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultFieldName();
```

Description

GetDefaultFieldName returns 'LoggedIn'. This allows server-side script to access this adapter field using the parent adapter's LoggedIn property.

173.1.14.2.2 TAdapterEndUserLoggedInField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns the result of GetLoggedInValue. Server-side script accesses the result of this function using the Value (see page 23945) property of this adapter field. This value may be used by server-side script to conditionally generate HTML. For example, when the end-user is logged in, a logout hyperlink is displayed. When the end-user is not logged in, a logout hyperlink is not displayed.

See Also

GetLoggedInValue (see page 24110)

173.1.15 TAdapterError Class

TAdapterError is the implementation of IAdapterError.

Class Hierarchy

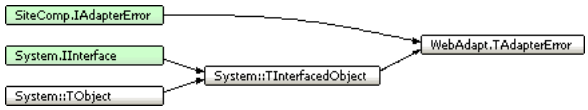

Pascal

```
TAdapterError = class(TInterfacedObject, IAdapterError);
```

C++

```
class TAdapterError : public TInterfacedObject, public IAdapterError;
```

File

WebAdapt

Description

TAdapterError is the implementation class for IAdapterError.

See Also

IAdapterError (see page 19660)

173.1.15.1 TAdapterError Members

The following tables list the members exposed by TAdapterError.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetErrorText (see page 23810)	Returns the value of the ErrorText property.
	GetField (see page 23810)	Returns the value of the Field property.
	GetID (see page 23810)	Returns the value of the ID property.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 23809)	Create instantiates an instance of TAdapterError (see page 23807). Do not call this constructor in application code. The adapter errors list object instantiates adapter error objects as needed.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ErrorText (see page 19662)	Returns text describing the error.
R	Field (see page 19662)	Returns text describing the error.
R	ID (see page 19662)	Returns error ID.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

173.1.15.2 TAdapterError.Create Constructor

Create instantiates an instance of TAdapterError (see page 23807). Do not call this constructor in application code. The adapter errors list object instantiates adapter error objects as needed.

Pascal

```
constructor Create(AErrors: TAdapterErrorsList; AIndex: Integer);
```

C++

```
__fastcall TAdapterError(TAdapterErrorsList AErrors, int AIndex);
```

Description

```
constructor Create(AErrors: TAdapterErrorsListTAdapterErrorsList; AIndex: Integer); public;
```

Create instantiates an instance of TAdapterError (see page 23807). Do not call this constructor in application code. The adapter errors list object instantiates adapter error objects as needed.

AErrors is the adapter errors list object that creates this TAdapterError (see page 23807) object.

AIndex is the index of this adapter error in the list that AErrors maintains.

173.1.15.3 TAdapterError Methods

The methods of the TAdapterError class are listed here.

Protected Methods

	Name	Description
	GetErrorText (see page 23810)	Returns the value of the ErrorText property.
	GetField (see page 23810)	Returns the value of the Field property.
	GetID (see page 23810)	Returns the value of the ID property.

173.1.15.3.1 TAdapterError.GetErrorText Method

Returns the value of the ErrorText property.

Pascal

```
function GetErrorText: string;
```

C++

```
__fastcall AnsiString GetErrorText();
```

Description

GetErrorText is the read implementation of the ErrorText property.

See Also

ErrorText (see page 19662)

173.1.15.3.2 TAdapterError.GetField Method

Returns the value of the Field property.

Pascal

```
function GetField: TComponent;
```

C++

```
__fastcall TComponent * GetField();
```

Description

GetField is the read implementation of the Field property.

See Also

Field (see page 19662)

173.1.15.3.3 TAdapterError.GetID Method

Returns the value of the ID property.

Pascal

```
function GetID: integer;
```

C++

```
__fastcall int GetID();
```

Description

GetID is the read implementation of the ID property.

See Also

ID (see page 19662)

173.1.16 TAdapterErrorsList Class

TAdapterErrorsList keeps track of errors that an adapter encounters.

Class Hierarchy

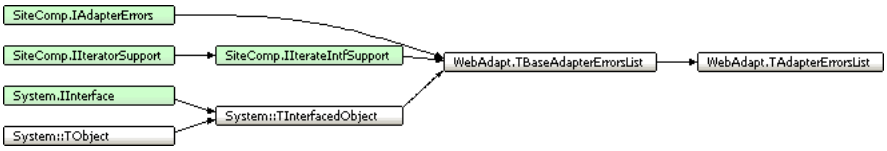

Pascal

```
TAdapterErrorsList = class(TBaseAdapterErrorsList);
```

C++

```
class TAdapterErrorsList : public TBaseAdapterErrorsList;
```

File

WebAdapt

Description

TAdapterErrorsList acts as a repository for all the errors an adapter (or any of its subcomponents) encounters while processing an HTTP request message. Adapter components create an instance of this class to implement their Errors (see page 24085) property. Server-side scripts can use the Errors (see page 24085) property to access the errors in this list. The usual purpose of such a script is to display error messages to the end user.

To add new error objects to the list, call the AddError (see page 23815) method. To access the errors in the list, read its Errors (see page 24085) property. Errors (see page 24085) lists the IAdapterError interfaces of adapter error objects (TAdapterError (see page 23807)) that represent each error in the list. You can also access the individual properties of those error objects using the FieldNames (see page 23817), IDs (see page 23817), and Messages (see page 23818) properties of the TAdapterErrorsList instance.

TAdapterErrorsList implements the IAdapterErrorsList interface.

Note: Do not confuse TAdapterErrorsList with TAdapterErrorList. TAdapterErrorsList stores the errors that the WebSnap application encounters when using an adapter. TAdapterErrorList is a display component that displays the errors from a TAdapterErrorsList object.

See Also

TAdapterErrorList (see page 25145)

TCustomAdapter (see page 24139)

TAdapterError (see page 23807)

IAdapterErrors (see page 19663)

IAdapterError (see page 19660)

173.1.16.1 TAdapterErrorsList Members

The following tables list the members exposed by TAdapterErrorsList.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	EndIterator (see page 24081)	This is EndIterator, a member of class TBaseAdapterErrorsList.
	GetCurrentIntf (see page 24081)	This is GetCurrentIntf, a member of class TBaseAdapterErrorsList.
	GetError (see page 24082)	Returns a particular error.
	GetErrorCount (see page 24082)	Return the number of errors in the errors list.
	GetObjectErrors (see page 24082)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 24082)	Indicates whether the errors list contains objects associated with a particular adapter field.
V	ImplClearErrors (see page 23815)	This is ImplClearErrors, a member of class TAdapterErrorsList.
V	ImplDefineLabel (see page 23816)	This is ImplDefineLabel, a member of class TAdapterErrorsList.
V	ImplGetError (see page 23816)	This is ImplGetError, a member of class TAdapterErrorsList.
V	ImplGetErrorCount (see page 23816)	This is ImplGetErrorCount, a member of class TAdapterErrorsList.
V	ImplGetObjectErrors (see page 23816)	This is ImplGetObjectErrors, a member of class TAdapterErrorsList.
V	ImplHasObjectErrors (see page 23816)	This is ImplHasObjectErrors, a member of class TAdapterErrorsList.
	NextIteration (see page 24084)	Advances to the objects next item.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	StartIterator (see page 24085)	This is StartIterator, a member of class TBaseAdapterErrorsList.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AddError (see page 23814)	This is the overview for the AddError method overload.
V	AddError (see page 24080)	This is the overview for the AddError method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearErrors (see page 19664)	Clears the errors list.
	ClearErrors (see page 24080)	Clear error list.
	Create (see page 23813)	Creates and initializes an instance of TAdapterErrorsList (see page 23810).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DefineLabel (see page 19664)	Associates a label with all errors associated with a particular adapter field.
	DefineLabel (see page 24081)	Define error label for field.
V	Destroy (see page 23814)	Frees an instance of TAdapterErrorsList (see page 23810).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetCurrentIntf (see page 19718)	This is GetCurrentIntf, a member of class IIterateIntfSupport.
	GetError (see page 19664)	Returns a particular error.
	GetErrorCount (see page 19665)	Return the number of errors in the errors list.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetObjectErrors (see page 19665)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 19665)	Indicates whether the errors list contains objects associated with a particular adapter field.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.

Public Properties

	Name	Description
	ErrorCount (see page 19666)	Returns the number of errors in the errors list.
	ErrorCount (see page 24085)	Number of error objects contained.
	Errors (see page 19666)	Returns an error from the list.
	Errors (see page 24085)	Access to error objects.
	FieldNames (see page 23817)	Lists the names of the objects that encountered the errors in the list.
	IDs (see page 23817)	Lists the error codes of the errors in the list.
	Messages (see page 23818)	Lists the error messages for the errors in the list.
	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

173.1.16.2 TAdapterErrorsList.Create Constructor

Creates and initializes an instance of TAdapterErrorsList (see page 23810).

Pascal

```
constructor Create(AAdapter: TObject);
```

C++

```
__fastcall TAdapterErrorsList(TObject * AAdapter);
```

Description

Typically, WebSnap applications do not create their own instances of TAdapterErrorsList (see page 23810). Adapter objects instantiate this class to implement their Errors (see page 24085) property.

AAdapter is the adapter component that uses this object to implement its Errors (see page 24085) property.

See Also

Errors (see page 24182)

173.1.16.3 TAdapterErrorsList.Destroy Destructor

Frees an instance of TAdapterErrorsList (see page 23810).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterErrorsList();
```

Description

Do not call Destroy directly. Instead, call the Free method, which checks whether the reference to the errors list is not nil before calling Destroy.

See Also

Free (see page 22300)

173.1.16.4 TAdapterErrorsList Methods

The methods of the TAdapterErrorsList class are listed here.

Protected Methods

	Name	Description
	ImplClearErrors (see page 23815)	This is ImplClearErrors, a member of class TAdapterErrorsList.
	ImplDefineLabel (see page 23816)	This is ImplDefineLabel, a member of class TAdapterErrorsList.
	ImplGetError (see page 23816)	This is ImplGetError, a member of class TAdapterErrorsList.
	ImplGetErrorCount (see page 23816)	This is ImplGetErrorCount, a member of class TAdapterErrorsList.
	ImplGetObjectErrors (see page 23816)	This is ImplGetObjectErrors, a member of class TAdapterErrorsList.
	ImplHasObjectErrors (see page 23816)	This is ImplHasObjectErrors, a member of class TAdapterErrorsList.

Public Methods

	Name	Description
	AddError (see page 23814)	This is the overview for the AddError method overload.

173.1.16.4.1 AddError Method

This is the overview for the AddError method overload.

Overload List

	Name	Description
	TAdapterErrorsList.AddError (Exception, string) (see page 23815)	Adds an error to the list.
	TAdapterErrorsList.AddError (Exception, TObject) (see page 23815)	Adds an error to the list.
	TAdapterErrorsList.AddError (string, string, Integer) (see page 23815)	Adds an error to the list.
	TAdapterErrorsList.AddError (string, TObject, Integer) (see page 23815)	Adds an error to the list.

173.1.16.4.1.1 TAdapterErrorsList.AddError Method (Exception, TObject)

Adds an error to the list.

Pascal

```
procedure AddError(AException: Exception; const AObjectName: string = ''); override;
procedure AddError(const AMessage: string; const AObjectName: string = ''; AID: Integer = 0);
override;
procedure AddError(const AMessage: string; AObject: TObject; AID: Integer = 0); override;
procedure AddError(AException: Exception; AObject: TObject); override;
```

C++

```
virtual __fastcall AddError(Exception AException, const AnsiString AObjectName = '');
virtual __fastcall AddError(const AnsiString AMessage, const AnsiString AObjectName = '', int
AID = 0);
virtual __fastcall AddError(const AnsiString AMessage, TObject * AObject, int AID = 0);
virtual __fastcall AddError(Exception AException, TObject * AObject);
```

Description

AddError adds an error to the object's internal list. An error is described by three pieces of information:

An error message string. You can specify this string directly with the AMessage parameter, or using an Exception object (with the AException parameter). If you specify an Exception object, the object's Message property becomes the message string.

The name of an object (Variable) related to the error. You can specify this value as a string (with the AObjectName parameter) or as an object reference (with the AObject parameter). If you omit both these parameters, but supply an EAdapterFieldException (see page 23698) object as the AException parameter, TAdapterErrorsList (see page 23810) uses the FieldName property of that exception object. If you do not supply an AObject parameter, an AObjectName parameter, or an EAdapterFIeldException object as the AException parameter, TAdapterErrorsList (see page 23810) uses an empty string.

An error ID value. You can specify this value using the AID parameter. If you do not supply a value for AID but specify an EAdapterException (see page 23696) object as the AException parameter, TAdapterErrorsList (see page 23810) uses the ID property of that exception object. If you do not supply an AID parameter or an EAdapterException (see page 23696) object, TAdapterErrorsList (see page 23810) uses 0.

See Also

FieldName (see page 23700)

ID (see page 23698)

Messages (see page 23818)

FieldNames (see page 23817)

IDs (see page 23817)

173.1.16.4.2 TAdapterErrorsList.ImplClearErrors Method

Pascal

```
procedure ImplClearErrors; override;
```

C++

```
virtual __fastcall ImplClearErrors();
```

Description

This is ImplClearErrors, a member of class TAdapterErrorsList.

173.1.16.4.3 TAdapterErrorsList.ImplDefineLabel Method

Pascal

```
procedure ImplDefineLabel(const AName: string; const ALabel: string); override;
```

C++

```
virtual __fastcall ImplDefineLabel(const AnsiString AName, const AnsiString ALabel);
```

Description

This is ImplDefineLabel, a member of class TAdapterErrorsList.

173.1.16.4.4 TAdapterErrorsList.ImplGetError Method

Pascal

```
function ImplGetError(I: Integer): IAdapterError; override;
```

C++

```
virtual __fastcall IAdapterError ImplGetError(int I);
```

Description

This is ImplGetError, a member of class TAdapterErrorsList.

173.1.16.4.5 TAdapterErrorsList.ImplGetErrorCount Method

Pascal

```
function ImplGetErrorCount: Integer; override;
```

C++

```
virtual __fastcall int ImplGetErrorCount();
```

Description

This is ImplGetErrorCount, a member of class TAdapterErrorsList.

173.1.16.4.6 TAdapterErrorsList.ImplGetObjectErrors Method

Pascal

```
function ImplGetObjectErrors(const AName: string): TObject; override;
```

C++

```
virtual __fastcall TObject * ImplGetObjectErrors(const AnsiString AName);
```

Description

This is ImplGetObjectErrors, a member of class TAdapterErrorsList.

173.1.16.4.7 TAdapterErrorsList.ImplHasObjectErrors Method

Pascal

```
function ImplHasObjectErrors(const AName: string): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplHasObjectErrors(const AnsiString AName);
```

Description

This is ImplHasObjectErrors, a member of class TAdapterErrorsList.

173.1.16.5 TAdapterErrorsList Properties

The properties of the TAdapterErrorsList class are listed here.

Public Properties

	Name	Description
R	FieldNames (see page 23817)	Lists the names of the objects that encountered the errors in the list.
R	IDs (see page 23817)	Lists the error codes of the errors in the list.
R	Messages (see page 23818)	Lists the error messages for the errors in the list.

173.1.16.5.1 TAdapterErrorsList.FieldNames Property

Lists the names of the objects that encountered the errors in the list.

Pascal

```
property FieldNames [I: Integer]: string;
```

C++

```
__property AnsiString FieldNames[int I];
```

Description

Use FieldNames to access the name of the object associated with a specified error. FieldNames values typically reflect the name of an adapter field that was the cause of the error. However, they can be any object reference for which there is a defined Variable that can be accessed from server-side script. If there is no object associated with an error, the value of FieldNames is an empty string.

I is the index of the error in the list, where 0 specifies the first error, 1 specifies the second error, and so on.

The value of FieldNames for an error is set by the AddError (see page 23815) method when the error is added to the list.

See Also

AddError (see page 23815)

Field (see page 19662)

FieldName (see page 23700)

173.1.16.5.2 TAdapterErrorsList.IDs Property

Lists the error codes of the errors in the list.

Pascal

```
property IDs [I: Integer]: integer;
```

C++

```
__property int IDs[int I];
```

Description

Use IDs to access the ID (error code) of a specified error in the list. If the error has no error code, the value of IDs is 0.

I is the index of the error in the list, where 0 specifies the first error, 1 specifies the second error, and so on.

The value of IDs for an error is set by the AddError (see page 23815) method when the error is added to the list.

See Also

AddError (see page 23815)

ID (see page 19662)

ID (see page 23698)

173.1.16.5.3 TAdapterErrorsList.Messages Property

Lists the error messages for the errors in the list.

Pascal

```
property Messages [I: Integer]: string;
```

C++

```
__property AnsiString Messages[int I];
```

Description

Use Messages to access the error message for a specified error in the list.

I is the index of the error in the list, where 0 specifies the first error, 1 specifies the second error, and so on.

The value of Messages for an error is set by the AddError (see page 23815) method when the error is added to the list.

See Also

AddError (see page 23815)

ErrorText (see page 19662)

173.1.17 TAdapterField Class

TAdapterField is an event handler based adapter field that provides a single value.

Class Hierarchy

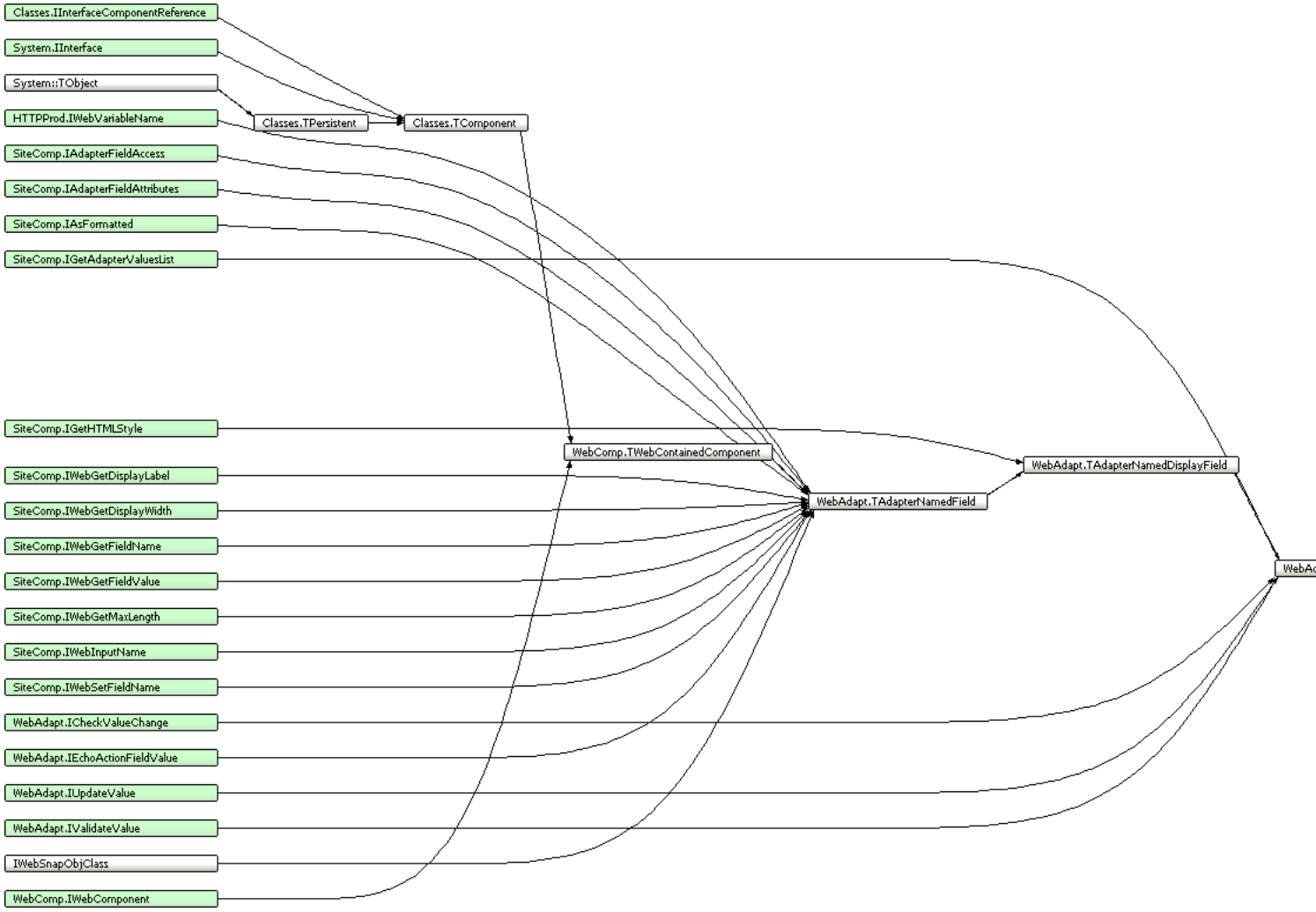

Pascal

```
TAdapterField = class(TCustomAdapterField);
```

C++

```
class TAdapterField : public TCustomAdapterField;
```

File

WebAdapt

Description

TAdapterField publishes the properties in TCustomAdapterField (see page 24211). TCustomAdapterField (see page 24211) calls event handlers to retrieve the adapter field's value, validate a value entered by the end-user, and update the adapter field's value.

Add TAdapterField to TAdapter (see page 23713) or other adapter components types when you need an adapter field that supports a single value using event handlers. Event handlers are used to provide the adapter field value(s), validate a value entered by the end-user, and update the adapter field with a value entered by the end-user.

There are other adapter field components that use event handlers. They differ in the type and number of values that are supported. For example, TAdapterMultiValue field is used when the end-user can enter multiple values using a group of check boxes or a multiple selection select list. TAdapterFileField (see page 23837), TAdapterMemoField (see page 23889), TAdapterBooleanField (see page 23758) and TAdapterImageField (see page 23869) are other event driven adapter fields.

See Also

TAdapterUpdateField (see page 24007)

TCustomAdapterField (see page 24211)

TAdapter (see page 23713)

TAdapterMultiValueField (see page 23900)

TAdapterMemoField (see page 23889)

TAdapterFileField (see page 23837)

TAdapterBooleanField (see page 23758)

TAdapterImageField (see page 23869)

173.1.17.1 TAdapterField Members

The following tables list the members exposed by TAdapterField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 24015)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 24015)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 24015)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 24016)	Retrieves a value entered by the end-user.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoOnValidateValue (see page 24016)	Validates field values entered by the end-user.
V	DoOnValidateValue (see page 24218)	Validates a single value entered by the end-user.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 24016)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.

	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 23921)	Suggests an HTML input type for this adapter field.
V	GetInputStyleType (see page 24017)	Indicates the preferred HTML element to use when editing this adapter field's value.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
V	GetValueCount (see page 24017)	Provides the number of values associated with this adapter field.
V	GetValues (see page 24017)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
V	ImplAsFormatted (see page 24018)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
V	ImplCheckValueChange (see page 24018)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
V	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
V	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayLabel (see page 24018)	Gets the preferred display label for this adapter field.
V	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
V	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
V	ImplGetDisplayWidth (see page 24019)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
V	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
V	ImplGetEchoActionFieldValue (see page 24019)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
V	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
V	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.

	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
	ImplGetValue (see page 24219)	Returns the value of this adapter field as a variant.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 24019)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 24020)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 24020)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 24020)	Updates the adapter field with value(s) entered by the end-user.
	ImplUpdateValue (see page 24219)	Called to update the adapter field's value with a value entered by the end-user.
	ImplValidateValue (see page 24020)	Validates values(s) entered by the end-user.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 24021)	Indicates whether this adapter field can support multiple values.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24021)	Clears references to a deleted component.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 24021)	Updates the adapter field with value(s) entered by the end-user.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateValue (see page 24022)	Validates values(s) entered by the end-user.

	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Adapter (see page 23943)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
	Create (see page 24014)	Creates an instance of TAdapterUpdateField (see page 24007).
	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterValuesList (see page 19695)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.

	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	Value (see page 24022)	Retrieves the value associated with this adapter field.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetDisplayText (see page 23829)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetValue (see page 23829)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 23829)	Called to update the adapter field's value with a value entered by the end-user.
	OnValidateValue (see page 23830)	Event handler called to validate value(s) entered by the end-user.

Published Properties

	Name	Description
	DisplayLabel (see page 23826)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23826)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 23826)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 23827)	Identifies the rights required to update this field.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ReadOnly (see page 23827)	Indicates whether this adapter field is read-only.
	Required (see page 23827)	Indicates whether the end-user must enter a value for this adapter field.
	Tag (see page 2808)	Stores an integer value as part of a component.
	ValuesList (see page 23828)	Associates a list of values with the adapter field.
	ViewAccess (see page 23828)	Identifies the rights required to view this field.

173.1.17.2 TAdapterField Properties

The properties of the TAdapterField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 23826)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23826)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 23826)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 23827)	Identifies the rights required to update this field.
	ReadOnly (see page 23827)	Indicates whether this adapter field is read-only.
	Required (see page 23827)	Indicates whether the end-user must enter a value for this adapter field.
	ValuesList (see page 23828)	Associates a list of values with the adapter field.
	ViewAccess (see page 23828)	Identifies the rights required to view this field.

173.1.17.2.1 TAdapterField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 23933)

173.1.17.2.2 TAdapterField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 23933)

173.1.17.2.3 TAdapterField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

173.1.17.2.4 TAdapterField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property AnsiString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

hasmodifyaccess (see page 24162)

173.1.17.2.5 TAdapterField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 23921)

173.1.17.2.6 TAdapterField.Required Property

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;__property Boolean Required;
```

Description

Use this property to indicate whether a value must be entered by the end user before submitting a form. Typically, an exception is raised when a required adapter field is not found when processing an adapter action.

See Also

GetRequired (see page 23934)

173.1.17.2.7 TAdapterField.ValuesList Property

Associates a list of values with the adapter field.

Pascal

```
property ValuesList: IValuesListAdapter;
```

C++

```
__property IValuesListAdapter ValuesList;
```

Description

Assign a component to the ValuesList property to define possible values for the adapter field. Server-side script can be used to enumerate the name/value pairs of the ValuesList component to generate HTML select lists, check boxes, or radio buttons.

See Also

IValuesListAdapter (see page 19740)

TStringsValuesList (see page 24540)

TDataSetValuesList (see page 8079)

173.1.17.2.8 TAdapterField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property AnsiString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

HasViewAccess (see page 24162)

173.1.17.3 TAdapterField Events

The events of the TAdapterField class are listed here.

Published Events

	Name	Description
	OnGetDisplayText (see page 23829)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetValue (see page 23829)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 23829)	Called to update the adapter field's value with a value entered by the end-user.
	OnValidateValue (see page 23830)	Event handler called to validate value(s) entered by the end-user.

173.1.17.3.1 TAdapterField.OnGetDisplayText Event

Event handler called to retrieve a string representation of the adapter field's value.

Pascal

```
property OnGetDisplayText: TAdapterFieldGetStringEvent;
```

C++

```
__property TAdapterFieldGetStringEvent OnGetDisplayText;
```

Description

The OnGetDisplayText event handler is called to retrieve a string representation an adapter field's value appropriate for a read-only display. If OnGetDisplayText is nil (Delphi) or NULL (C++), then the value of the adapter field will be used (after conversion to a string).

See Also

ImplAsFormatted (see page 24018)

173.1.17.3.2 TAdapterField.OnGetValue Event

Called to retrieve the adapter field's value.

Pascal

```
property OnGetValue: TAdapterFieldGetValueEvent;
```

C++

```
__property TAdapterFieldGetValueEvent OnGetValue;
```

Description

OnGetValue is called by ImplGetValue to retrieve the value associated with this adapter field.

See Also

ImplGetValue (see page 24219)

173.1.17.3.3 TAdapterField.OnUpdateValue Event

Called to update the adapter field's value with a value entered by the end-user.

Pascal

```
property OnUpdateValue: TAdapterFieldValueEvent;
```

C++

```
__property TAdapterFieldValueEvent OnUpdateValue;
```

Description

OnUpdateValue is called when an action request contains a value for this adapter field. If this event handler is not implemented, the value entered by the end-user will be ignored.

173.1.17.3.4 TAdapterField.OnValidateValue Event

Event handler called to validate value(s) entered by the end-user.

Pascal

```
property OnValidateValue: TAdapterFieldValidateValueEvent;
```

C++

```
__property TAdapterFieldValidateValueEvent OnValidateValue;
```

Description

This event handler is called by DoOnValidateValue to validate the value(s) entered by the end-user. The event handler must raise an exception to indicate that the value(s) are not valid.

See Also

DoOnValidateValue (see page 24016)

IActionFieldValue (see page 920)

173.1.18 TAdapterFields Class

TAdapterFields manages the list of fields that belong to an adapter.

Class Hierarchy

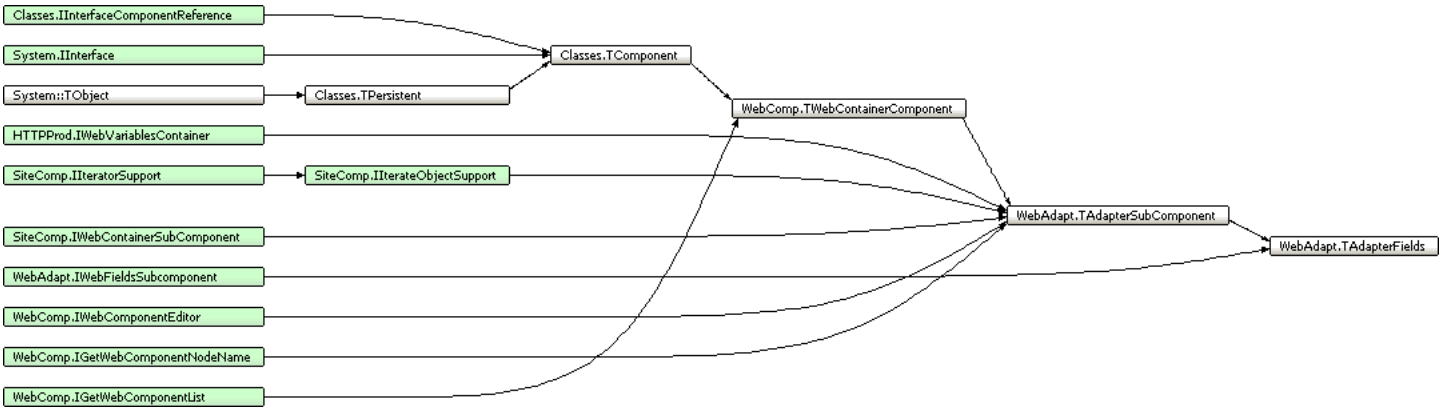

Pascal

```
TAdapterFields = class(TAdapterSubComponent, IWebFieldsSubcomponent);
```

C++

```
class TAdapterFields : public TAdapterSubComponent, public IWebFieldsSubcomponent;
```

File

WebAdapt

Description

TAdapterFields is the type of the Data property on an adapter component. It manages a list that contains all the fields that belong to the adapter. The field objects (TAdapterNamedField (see page 23925) descendants) are available using the WebFields (see page 23837) property. TAdapterFields also includes support for associating the fields with variable names so that server-side script can use the field values.

TAdapterFields represents the list of explicitly assigned fields. Adapters that include a set of default fields use a TAdapterDefaultFields (see page 23778) object instead.

See Also

TCustomAdapter (see page 24139)

TAdapterActions (see page 23738)

TAdapterDefaultActions (see page 23770)

TAdapterDefaultFields (see page 23778)

TAdapterNamedField (see page 23925)

173.1.18.1 TAdapterFields Members

The following tables list the members exposed by TAdapterFields.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AdapterCanAddClass (see page 23835)	Indicates whether the associated adapter supports a specified class of field component.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 23996)	Indicates whether an object of a specified type can be added to this collection.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EndIterator (see page 23997)	Stops the TAdapterSubComponent (see page 23991) object from iterating over the Web items it lists.
	FindNamedVariable (see page 23997)	Returns the item from this list with a specified variable name.
	FindVariable (see page 23998)	Returns the item from this list with a specified variable name.
	FindVariableContainer (see page 23998)	Returns the Web module that contains a specified item from this list.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24755)	Executes a callback for each child Web item.

	GetCurrentObject (see page 23999)	Returns the current field or action from the list when an enumerator is iterating over child items.
V	GetDefaultWebComponents (see page 23999)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
V	GetDefaultWebComponents (see page 24756)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariable (see page 24000)	Returns a child item from this list, given its position in the list of child items with variable names.
	GetVariableCount (see page 24000)	Indicates the number of items in this list that can be accessed from server-side script.
V	GetVariablesInUse (see page 24001)	Returns the value of the VariablesInUse (see page 24007) property.
	GetWebComponentNodeName (see page 24001)	Returns a string that indicates the type of items in this collection.
	GetWebComponents (see page 24756)	Returns the list of Web items that this item manages.
V	ImplCanAddClass (see page 23836)	Provides the underlying implementation for the CanAddClass (see page 23996) method.
V	ImplGetDefaultComponentList (see page 24756)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetWebComponentNodeName (see page 23836)	Provides the underlying implementation for the GetWebComponentNodeName (see page 24001) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24003)	Changes the current object to the next item that has a variable name.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	NotifyAdapterChange (see page 24003)	Notifies the associated adapter when the list of items changes.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24757)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 24004)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	SetDefaultWebComponents (see page 24757)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetWebParent (see page 23836)	Associates this component with an adapter.
	StartIterator (see page 24006)	Makes the first child item with a variable name the current object.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.

V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24006)	Indicates the adapter that uses this object to manage its fields or actions.
R	VariablesInUse (see page 24007)	Lists those fields or actions in the collection for which variables are defined.
R	WebContainerComponentHelper (see page 24758)	Provides access to the helper object that allows this object to manage its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24754)	Creates and initializes an instance of TWebContainerComponent (see page 24751).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24755)	Frees an instance of TWebContainerComponent (see page 24751).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetCurrentObject (see page 19719)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.

	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24000)	Returns the associated adapter.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetWebComponentNodeName (see page 24773)	This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24002)	Indicates whether this component has an associated adapter.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 24004)	Reads the component's published properties, stored data, and owned components from a reader object.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetParentComponent (see page 24005)	Sets the Adapter (see page 24006) property to be the same as this component's parent.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.

	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebFields (see page 23837)	Lists the field components that this object manages.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.18.2 TAdapterFields Methods

The methods of the TAdapterFields class are listed here.

Protected Methods

	Name	Description
	AdapterCanAddClass (see page 23835)	Indicates whether the associated adapter supports a specified class of field component.
V	ImplCanAddClass (see page 23836)	Provides the underlying implementation for the CanAddClass (see page 23996) method.
V	ImplGetWebComponentNodeName (see page 23836)	Provides the underlying implementation for the GetWebComponentNodeName (see page 24001) method.
V	SetWebParent (see page 23836)	Associates this component with an adapter.

173.1.18.2.1 TAdapterFields.AdapterCanAddClass Method

Indicates whether the associated adapter supports a specified class of field component.

Pascal

```
function AdapterCanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean AdapterCanAddClass(TComponent * AParent, TClass AClass);
```

Description

TAdapterActions (see page 23738) uses this method internally as part of its implementation for the CanAddClass (see page 23996) method. This allows the TAdapterFields (see page 23830) object to indicate, at design time, which types of components are valid as members of the WebFields (see page 23837) property list.

AParent is the instance of TAdapterFields (see page 23830) to which the child fields are to be added.

AClass describes the class of the child component to be added to AParent.

AdapterCanAddClass returns true if the adapter recognizes and supports AClass as a child field, false otherwise.

AdapterCanAddClass calls the CanAddFieldClass method of its associated adapter to determine whether it accepts the specified field class.

See Also

WebFields (see page 23837)

Adapter (see page 24006)

ImplCanAddClass (see page 23836)

CanAddFieldClass (see page 24150)

173.1.18.2.2 TAdapterFields.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass (see page 23996) method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass (see page 23996) method calls ImplCanAddClass to determine whether an object of a specified type can be added to this collection.

AParent is the TAdapterFields (see page 23830) object to which the Web item will be added.

AClass is the class of the proposed child item.

ImplCanAddClass returns true if the TAdapterFields (see page 23830) object can include a child of type AClass, false otherwise.

ImplCanAddClass returns true if one of the following two conditions is met:

AClass is an adapter field class that the associated adapter recognizes and accepts.

AClass is a data entry field (a descendant of TAdapterUpdateField (see page 24007)).

See Also

CanAddClass (see page 23996)

AdapterCanAddClass (see page 23835)

TAdapterUpdateField (see page 24007)

173.1.18.2.3 TAdapterFields.ImplGetWebComponentNodeName Method

Provides the underlying implementation for the GetWebComponentNodeName (see page 24001) method.

Pascal

```
function ImplGetWebComponentNodeName: string; override;
```

C++

```
virtual __fastcall AnsiString ImplGetWebComponentNodeName();
```

Description

The GetWebComponentNodeName (see page 24001) method calls ImplGetWebComponentNodeName to obtain a label for the type of items in the collection.

ImplGetWebComponentNodeName returns the string 'Fields'.

See Also

GetWebComponentNodeName (see page 24001)

173.1.18.2.4 TAdapterFields.SetWebParent Method

Associates this component with an adapter.

Pascal

```
procedure SetWebParent(const Value: TComponent); override;
```

C++

```
virtual __fastcall SetWebParent(const TComponent * Value);
```

Description

SetWebParent sets up the association between this component and its parent adapter.

Value is the adapter component that acts as a parent to this collection.

SetWebParent assigns Value to the protected Adapter (see page 24006) property and replaces that adapter's Data property with this TAdapterFields (see page 23830) instance.

See Also

Data (see page 24181)

Adapter (see page 24006)

173.1.18.3 TAdapterFields Properties

The properties of the TAdapterFields class are listed here.

Public Properties

	Name	Description
R	WebFields (see page 23837)	Lists the field components that this object manages.

173.1.18.3.1 TAdapterFields.WebFields Property

Lists the field components that this object manages.

Pascal

```
property WebFields: TWebComponentList;
```

C++

```
__property TWebComponentList WebFields;
```

Description

Read WebFields to access the list of field components. Each component in this list is a descendant of TAdapterNamedField (see page 23925).

See Also

TAdapterNamedField (see page 23925)

173.1.19 TAdapterFileField Class

TAdapterFileField is an event handler based adapter field that can accept uploaded files.

Class Hierarchy

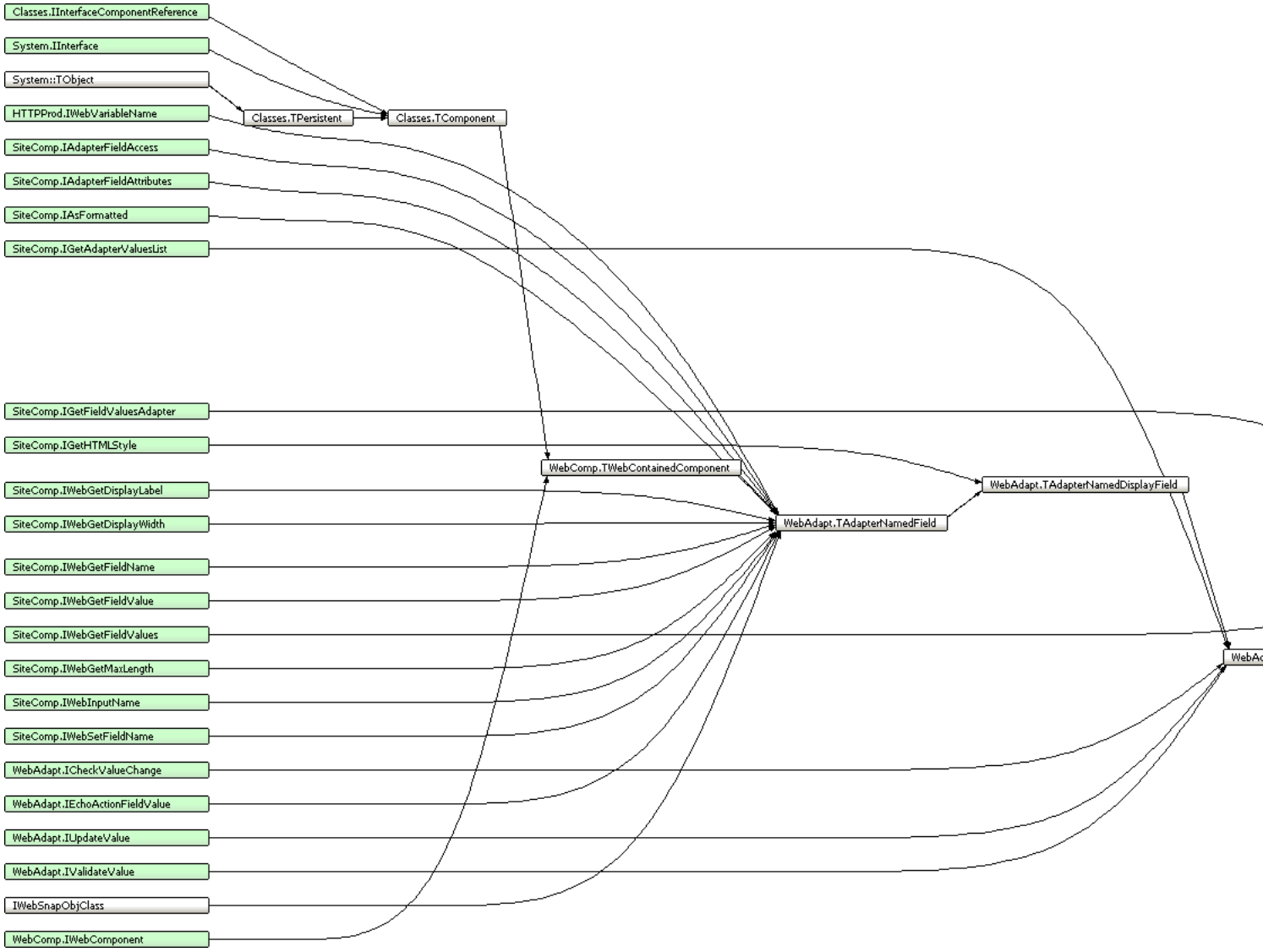

Pascal

```
TAdapterFileField = class(TCustomAdapterFileField);
```

C++

```
class TAdapterFileField : public TCustomAdapterFileField;
```

File

WebAdapt

Description

TAdapterFileField publishes the properties in TCustomAdapterFileField (see page 24220). TCustomAdapterFileField (see page 24220) calls event handlers to retrieve the adapter field's filenames, validate files uploaded by the end-user, and update the adapter field's files.

Add TAdapterFileField to TAdapter (see page 23713) or other adapter components types when you need an adapter field that

supports files using event handlers. There are other adapter field components that use event handlers. They differ in the type and number of values that are supported. See TAdapterField (see page 23818), TAdapterMultiValueField (see page 23900), TAdapterMemoField (see page 23889), TAdapterBooleanField (see page 23758) and TAdapterImageField (see page 23869).

See Also

TCustomAdapterFileField (see page 24220)

TAdapter (see page 23713)

TAdapterField (see page 23818)

TAdapterMemoField (see page 23889)

TAdapterMultiValueField (see page 23900)

TAdapterBooleanField (see page 23758)

TAdapterImageField (see page 23869)

173.1.19.1 TAdapterFileField Members

The following tables list the members exposed by TAdapterFileField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 24015)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 24015)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 24015)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 24016)	Retrieves a value entered by the end-user.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoOnValidateValue (see page 24016)	Validates field values entered by the end-user.
V	DoOnValidateValue (see page 24228)	Validates files uploaded by the end-user.
	EndValueIterator (see page 24093)	Completes iteration.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 24016)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.

	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetCurrentValue (see page 24094)	Implements IWebGetFieldValues' GetCurrentValue.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
V	GetDisplayStyleType (see page 24228)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetFieldValuesAdapter (see page 24094)	Returns the adapter object that implements the Values property.
	GetFileNames (see page 24229)	Returns the count of files associated with this adapter field.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 23921)	Suggests an HTML input type for this adapter field.
V	GetInputStyleType (see page 24017)	Indicates the preferred HTML element to use when editing this adapter field's value.
V	GetInputStyleType (see page 24229)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
V	GetValueCount (see page 24017)	Provides the number of values associated with this adapter field.
V	GetValueCount (see page 24094)	Returns the number of values associated with this adapter field.
V	GetValueCount (see page 24229)	Returns the number of values associated with this adapter field.
V	GetValues (see page 24017)	Returns one of the values provided by this adapter field.
V	GetValues (see page 24095)	Returns one of the values associated with the adapter field.
V	GetValues (see page 24230)	Returns one of the values associated with the adapter field.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasValue (see page 24095)	Implements IWebGetFieldValues' HasValue method.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
V	ImplAsFormatted (see page 24018)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
V	ImplCheckValueChange (see page 24018)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
V	ImplCheckValueChange (see page 24230)	Indicates whether the end-user may have uploaded new files for this adapter field.
V	ImplGetCurrentValue (see page 24095)	Implements IWebGetFieldValues' GetCurrentValue (see page 24094) method.
V	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
V	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayLabel (see page 24018)	Gets the preferred display label for this adapter field.
V	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.

	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 24019)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 24019)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetFieldValuesAdapter (see page 24096)	Returns the adapter object that implements the Values property.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 24019)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasValue (see page 24096)	Implements IWebGetFieldValues' HasValue (see page 24095) method.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 24020)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 24020)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 24020)	Updates the adapter field with value(s) entered by the end-user.
	ImplUpdateValue (see page 24230)	Updates the adapter field with value(s) entered by the end-user.
	ImplValidateValue (see page 24020)	Validates values(s) entered by the end-user.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 24021)	Indicates whether this adapter field can support multiple values.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextValueIteration (see page 24096)	Implements IWebGetFieldValues' NextValueIteration method.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24021)	Clears references to a deleted component.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.

	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartValueIterator (see page 24097)	Implements IWebGetFieldValues' StartValueIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 24021)	Updates the adapter field with value(s) entered by the end-user.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateValue (see page 24022)	Validates values(s) entered by the end-user.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 24023)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 24024)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
V	Create (see page 24014)	Creates an instance of TAdapterUpdateField (see page 24007).
V	Create (see page 24092)	Constructor for TBaseAdapterMultiValueField (see page 24086).
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.

	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24093)	Destructor for TBaseAdapterMultiValueField (see page 24086).
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndValueIterator (see page 19761)	Indicate the iteration is finished.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterValuesList (see page 19695)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetCurrentValue (see page 19761)	Retrieves the current value as a variant.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetFieldValuesAdapter (see page 19698)	This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasValue (see page 19761)	Indicates whether the adapter field has a matching value.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextValueIteration (see page 19762)	Position the iteration pointer on the next value.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartValueIterator (see page 19762)	Reset the iteration pointer to the first item.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FileCount (see page 24231)	Returns one of the filenames associated with this adapter field.
R	FileNames (see page 24231)	Returns a file associated with this adapter field.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	Value (see page 24022)	Retrieves the value associated with this adapter field.
	ValuesList (see page 24023)	Associates a list of values with the adapter field.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.

	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetFileCount (see page 23848)	Returns the number of values associated with this adapter field.
	OnGetFileNames (see page 23848)	Returns one of the filenames associated with the adapter field.
	OnUploadFiles (see page 23848)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateFiles (see page 23849)	Validates values entered by the end-user.

Published Properties

	Name	Description
	DisplayLabel (see page 23845)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23846)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 23846)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 23846)	Identifies the rights required to update this field.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ReadOnly (see page 23847)	Indicates whether this adapter field is read-only.
	Required (see page 23847)	Indicates whether the end-user must enter a value for this adapter field.
	Tag (see page 2808)	Stores an integer value as part of a component.
	ViewAccess (see page 23847)	Identifies the rights required to view this field.

173.1.19.2 TAdapterFileField Properties

The properties of the TAdapterFileField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 23845)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23846)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 23846)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 23846)	Identifies the rights required to update this field.
	ReadOnly (see page 23847)	Indicates whether this adapter field is read-only.
	Required (see page 23847)	Indicates whether the end-user must enter a value for this adapter field.
	ViewAccess (see page 23847)	Identifies the rights required to view this field.

173.1.19.2.1 TAdapterFileField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 23933)

173.1.19.2.2 TAdapterFileField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 23933)

173.1.19.2.3 TAdapterFileField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

173.1.19.2.4 TAdapterFileField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property AnsiString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

hasmodifyaccess (see page 24162)

173.1.19.2.5 TAdapterFileField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 23921)

173.1.19.2.6 TAdapterFileField.Required Property

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;__property Boolean Required;
```

Description

Use this property to indicate whether a value must be entered by the end user before submitting a form. Typically, an exception is raised when a required adapter field is not found when processing an adapter action.

See Also

GetRequired (see page 23934)

173.1.19.2.7 TAdapterFileField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property AnsiString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

HasViewAccess (see page 24162)

173.1.19.3 TAdapterFileField Events

The events of the TAdapterFileField class are listed here.

Published Events

	Name	Description
	OnGetFileCount (see page 23848)	Returns the number of values associated with this adapter field.
	OnGetFileNames (see page 23848)	Returns one of the filenames associated with the adapter field.
	OnUploadFiles (see page 23848)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateFiles (see page 23849)	Validates values entered by the end-user.

173.1.19.3.1 TAdapterFileField.OnGetFileCount Event

Returns the number of values associated with this adapter field.

Pascal

```
property OnGetFileCount: TAdapterFieldGetValueCountEvent;
```

C++

```
__property TAdapterFieldGetValueCountEvent OnGetFileCount;
```

Description

Write an OnGetFileCount event handler to indicate the number of filenames associated with this adapter field.

See Also

GetValueCount (see page 24229)

173.1.19.3.2 TAdapterFileField.OnGetFileNames Event

Returns one of the filenames associated with the adapter field.

Pascal

```
property OnGetFileNames: TAdapterFieldGetMultiValueStringEvent;
```

C++

```
__property TAdapterFieldGetMultiValueStringEvent OnGetFileNames;
```

Description

Write an OnGetFileNames event handler to provide the names of files associated with this adapter field.

See Also

GetFileNames (see page 24229)

173.1.19.3.3 TAdapterFileField.OnUploadFiles Event

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
property OnUploadFiles: TAdapterFieldFileListEvent;
```

C++

```
__property TAdapterFieldFileListEvent OnUploadFiles;
```

Description

OnUploadFiles is called by ImplUpdateValue. Write an OnUploadFiles event handler to accept files uploaded by the end-user. Raise an exception to indicate an error.

See Also

ImplUpdateValue (see page 24230)

TUpdateFileList (see page 24550)

173.1.19.3.4 TAdapterFileField.OnValidateFiles Event

Validates values entered by the end-user.

Pascal

```
property OnValidateFiles: TAdapterFieldFileListEvent;
```

C++

```
__property TAdapterFieldFileListEvent OnValidateFiles;
```

Description

OnValidateFiles is called by DoOnValidateValue. Implement this event handler to validate files uploaded by the end-user. Raise an exception to indicate a validation error.

See Also

DoOnValidateValue (see page 24228)

TUpdateFileList (see page 24550)

173.1.20 TAdapterGotoPageAction Class

TAdapterGotoPageAction is an adapter action that supports commands to go to a page of a TDefaultFieldsPagedAdapter (see page 24403).

Class Hierarchy

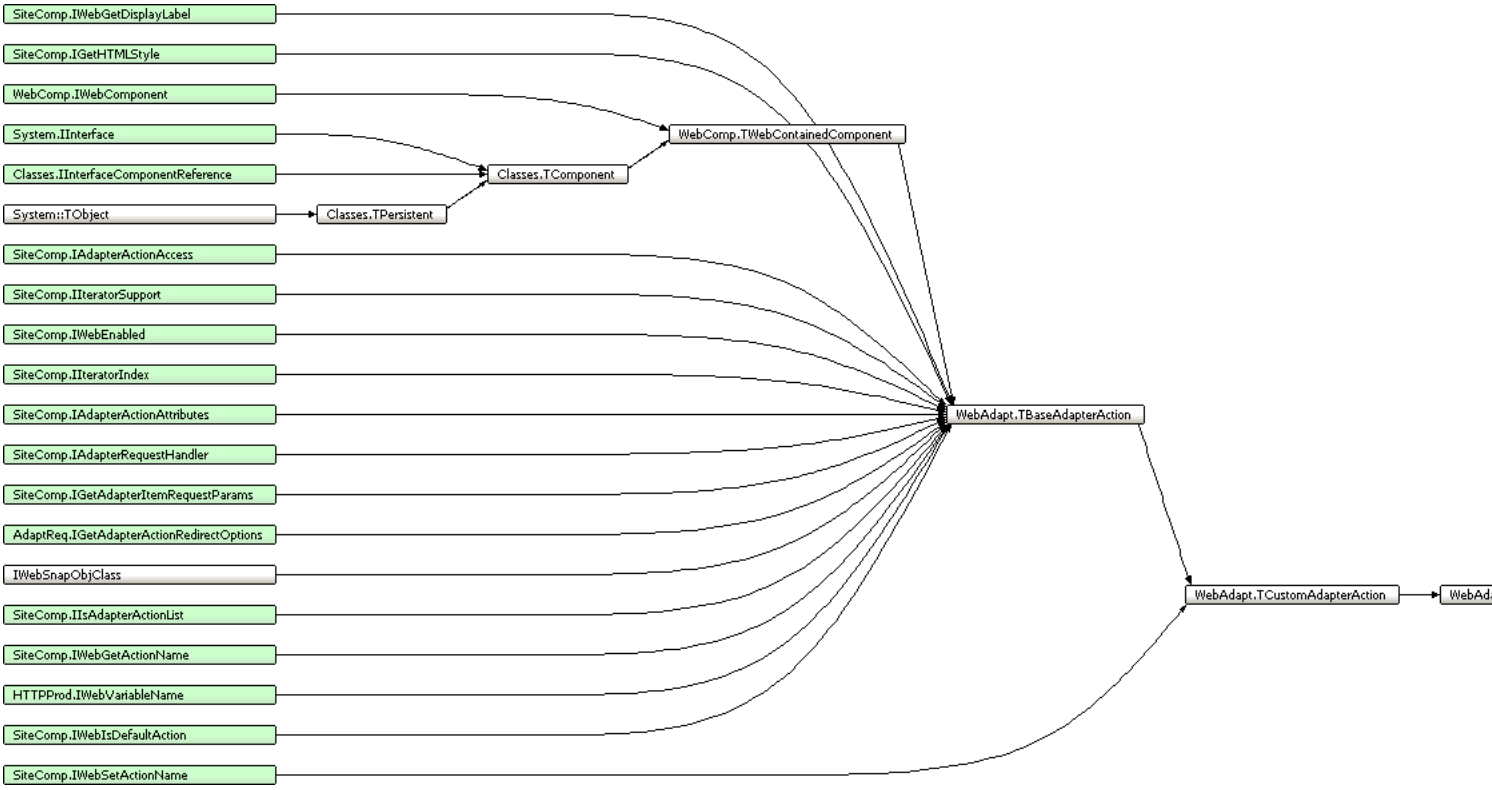

Pascal

```
TAdapterGotoPageAction = class(TCustomAdapterGotoPageAction);
```

C++

```
class TAdapterGotoPageAction : public TCustomAdapterGotoPageAction;
```

File

WebAdapt

Description

TAdapterGotoPageAction is an adapter action that has a list of commands. Each command has a parameter that identifies a page in the parent adapter. When executed, TAdapterGotoPageAction changes the current page in the parent adapter. TAdapterGotoPageAction publishes the properties of TCustomAdapterGotoPageAction (see page 24233). Add TAdapterGotoPageAction to a descendant class of TDefaultFieldsPagedAdapter (see page 24403) such as TPagedAdapter (see page 24525).

See Also

TCustomAdapterGotoPageAction (see page 24233)

TDefaultFieldsPagedAdapter (see page 24403)

TPagedAdapter (see page 24525)

TAdapterNextPageAction (see page 23945)

TAdapterPrevPageAction (see page 23979)

173.1.20.1 TAdapterGotoPageAction Members

The following tables list the members exposed by TAdapterGotoPageAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 24377)	Returns the parent adapter of this adapter action.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 24055)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultActionName (see page 24240)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
V	GetDisplayStyleType (see page 24241)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.

	GetItemCount (see page 24241)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 24063)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 24195)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 24241)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24242)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetDisplayLabel (see page 24242)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIsAdapterActionList (see page 24243)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplIsDefaultAction (see page 24243)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 24069)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 24197)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 24243)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Adapter (see page 24378)	Returns the parent adapter of this adapter action.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.

	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 24073)	Suggests a display label for this adapter action.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 23857)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 23858)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 23858)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 23859)	Called before the adapter action response is generated.
	OnGetPageName (see page 23859)	Provides the display labels for each goto page command.
	OnGetParams (see page 23859)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	ExecuteAccess (see page 23856)	Defines the rights required to execute this action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RedirectOptions (see page 23857)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.20.2 TAdapterGotoPageAction Properties

The properties of the TAdapterGotoPageAction class are listed here.

Published Properties

	Name	Description
	ExecuteAccess (see page 23856)	Defines the rights required to execute this action.
	RedirectOptions (see page 23857)	Indicates whether to perform an HTTP redirect after successfully executing this action.

173.1.20.2.1 TAdapterGotoPageAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property AnsiString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 25625)

hasexecuteaccess (see page 24162)

173.1.20.2.2 TAdapterGotoPageAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect (see page 15201)

173.1.20.3 TAdapterGotoPageAction Events

The events of the TAdapterGotoPageAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 23857)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 23858)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 23858)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 23859)	Called before the adapter action response is generated.
	OnGetPageName (see page 23859)	Provides the display labels for each goto page command.
	OnGetParams (see page 23859)	Specifies name/value pairs needed to execute this adapter action.

173.1.20.3.1 TAdapterGotoPageAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

173.1.20.3.2 TAdapterGotoPageAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.20.3.3 TAdapterGotoPageAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

173.1.20.3.4 TAdapterGotoPageAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.20.3.5 TAdapterGotoPageAction.OnGetPageName Event

Provides the display labels for each goto page command.

Pascal

```
property OnGetPageName: TWebFormGetPageNameEvent;
```

C++

```
__property TWebFormGetPageNameEvent OnGetPageName;
```

Description

Implement this event handler to specify display labels for the commands that goto pages in the parent adapter. APageIndex is the zero-based index of the page to goto.

See Also

ImplGetDisplayLabel (see page 24242)

173.1.20.3.6 TAdapterGotoPageAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

173.1.21 TAdapterHiddenField Class

Class Hierarchy

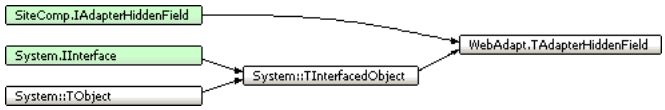

Pascal

```
TAdapterHiddenField = class(TInterfacedObject, IAdapterHiddenField);
```

C++

```
class TAdapterHiddenField : public TInterfacedObject, public IAdapterHiddenField;
```

File

WebAdapt

Description

This is class WebAdapt.TAdapterHiddenField.

173.1.21.1 TAdapterHiddenField Members

The following tables list the members exposed by TAdapterHiddenField.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetName (see page 23862)	This is GetName, a member of class TAdapterHiddenField.
	GetValue (see page 23862)	This is GetValue, a member of class TAdapterHiddenField.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 23861)	This is Create, a member of class TAdapterHiddenField.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Name (see page 19671)	This is Name, a member of class IAdapterHiddenField.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	Value (see page 19671)	This is Value, a member of class IAdapterHiddenField.

173.1.21.2 TAdapterHiddenField.Create Constructor

Pascal

```
constructor Create(AHiddenFields: TAdapterHiddenFieldsList; AIndex: Integer);
```

C++

```
__fastcall TAdapterHiddenField(TAdapterHiddenFieldsList AHiddenFields, int AIndex);
```

Description

This is Create, a member of class TAdapterHiddenField.

173.1.21.3 TAdapterHiddenField Methods

The methods of the TAdapterHiddenField class are listed here.

Protected Methods

	Name	Description
	GetName (see page 23862)	This is GetName, a member of class TAdapterHiddenField.
	GetValue (see page 23862)	This is GetValue, a member of class TAdapterHiddenField.

173.1.21.3.1 TAdapterHiddenField.GetName Method

Pascal

```
function GetName: string;
```

C++

```
__fastcall AnsiString GetName();
```

Description

This is GetName, a member of class TAdapterHiddenField.

173.1.21.3.2 TAdapterHiddenField.GetValue Method

Pascal

```
function GetValue: string;
```

C++

```
__fastcall AnsiString GetValue();
```

Description

This is GetValue, a member of class TAdapterHiddenField.

173.1.22 TAdapterHiddenFieldsList Class

TAdapterHiddenFieldsList maintains a collection of name/value pairs that an adapter includes as hidden fields in an HTML form.

Class Hierarchy

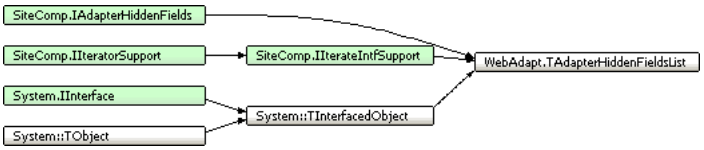

Pascal

```
TAdapterHiddenFieldsList = class(TInterfacedObject, IAdapterHiddenFields, IIterateIntfSupport);
```

C++

```
class TAdapterHiddenFieldsList : public TInterfacedObject, public IAdapterHiddenFields, public
IIterateIntfSupport;
```

File

WebAdapt

Description

Adapter components create instances of TAdapterHiddenFieldsList to store the name/value pairs they include as hidden fields in HTML forms. Each entry in the list has a field name and a value for that field. The adapter uses the methods of TAdapterHiddenFieldsList to add name/value pairs to the list, based on its hidden field options. When an adapter page producer generates a form using the adapter, it includes hidden fields for every name/value pair in the list. Those hidden fields are then included in any request that originates from the resulting HTML form, so that the adapter and its subcomponents can access those values to respond to the request.

For example, dataset adapters that have a Mode property that indicates whether the form is used for read-only browsing, inserting a new record, editing an existing record, or generating a query. The adapter includes the current mode as a hidden field. When the WebSnap application receives a message from the HTML form with that hidden field, the adapter can reestablish the correct mode for processing the request.

See Also

TCustomAdapter (see page 24139)

TCustomAdapter (see page 24139)

TCustomAdapter (see page 24139)

TCustomAdapter (see page 24139)

TAdapterPageProducer (see page 19802)

TCustomDataSetAdapter (see page 7766)

173.1.22.1 TAdapterHiddenFieldsList Members

The following tables list the members exposed by TAdapterHiddenFieldsList.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	EndIterator (see page 23866)	This is EndIterator, a member of class TAdapterHiddenFieldsList.
	GetCurrentIntf (see page 23867)	This is GetCurrentIntf, a member of class TAdapterHiddenFieldsList.
	GetField (see page 23867)	This is GetField, a member of class TAdapterHiddenFieldsList.
	GetFieldCount (see page 23867)	This is GetFieldCount, a member of class TAdapterHiddenFieldsList.
	NextIteration (see page 23867)	Advances to the objects next item.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	StartIterator (see page 23868)	This is StartIterator, a member of class TAdapterHiddenFieldsList.

Protected Properties

	Name	Description
R	FieldCount (see page 23868)	This is FieldCount, a member of class TAdapterHiddenFieldsList.
R	Fields (see page 23868)	This is Fields, a member of class TAdapterHiddenFieldsList.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Add (see page 23866)	Adds a name/value pair to the list.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 23866)	Clears the list of all the name/value pairs it currently holds.
	Create (see page 23865)	Creates and initializes a TAdapterHiddenFieldsList (see page 23862) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 23865)	Frees an instance of TAdapterHiddenFieldsList (see page 23862).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

173.1.22.2 TAdapterHiddenFieldsList.Create Constructor

Creates and initializes a TAdapterHiddenFieldsList (see page 23862) object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TAdapterHiddenFieldsList();
```

Description

Typically, applications have no need to create TAdapterHiddenFieldsList (see page 23862) instances. The adapter component creates and maintains its own internal TAdapterHiddenFieldsList (see page 23862) objects, which it uses to store the hidden fields it needs to add to HTML forms.

See Also

listhiddenfields (see page 24173)

ListHiddenRecordFields (see page 24173)

173.1.22.3 TAdapterHiddenFieldsList.Destroy Destructor

Frees an instance of TAdapterHiddenFieldsList (see page 23862).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterHiddenFieldsList();
```

Description

Do not use the Destroy method directly to free a TAdapterHiddenFieldsList (see page 23862) instance. Instead, use the Free method, which checks that the TAdapterHiddenFieldsList (see page 23862) reference is not nil before calling Destroy.

See Also

Free (see page 22300)

173.1.22.4 TAdapterHiddenFieldsList Methods

The methods of the TAdapterHiddenFieldsList class are listed here.

Protected Methods

	Name	Description
	EndIterator (see page 23866)	This is EndIterator, a member of class TAdapterHiddenFieldsList.
	GetCurrentIntf (see page 23867)	This is GetCurrentIntf, a member of class TAdapterHiddenFieldsList.
	GetField (see page 23867)	This is GetField, a member of class TAdapterHiddenFieldsList.
	GetFieldCount (see page 23867)	This is GetFieldCount, a member of class TAdapterHiddenFieldsList.
	NextIteration (see page 23867)	Advances to the objects next item.
	StartIterator (see page 23868)	This is StartIterator, a member of class TAdapterHiddenFieldsList.

Public Methods

	Name	Description
	Add (see page 23866)	Adds a name/value pair to the list.
	Clear (see page 23866)	Clears the list of all the name/value pairs it currently holds.

173.1.22.4.1 TAdapterHiddenFieldsList.Add Method

Adds a name/value pair to the list.

Pascal

```
procedure Add(const AName: string; const AValue: string);
```

C++

```
__fastcall Add(const AnsiString AName, const AnsiString AValue);
```

Description

Call Add to add a new name/value pair to the list of hidden fields that the adapter maintains.

AName is the name of the field.

AValue is the value of the field.

See Also

Clear (see page 23866)

173.1.22.4.2 TAdapterHiddenFieldsList.Clear Method

Clears the list of all the name/value pairs it currently holds.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the TAdapterHiddenFieldsList (see page 23862) object. Clear discards any name/value pairs that have been added previously by a call to the Add (see page 23866) method.

See Also

Add (see page 23866)

173.1.22.4.3 TAdapterHiddenFieldsList.EndIterator Method

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

This is EndIterator, a member of class TAdapterHiddenFieldsList.

173.1.22.4.4 TAdapterHiddenFieldsList.GetCurrentIntf Method

Pascal

```
function GetCurrentIntf(OwnerData: Pointer): IUnknown;
```

C++

```
__fastcall IUnknown GetCurrentIntf(void * OwnerData);
```

Description

This is GetCurrentIntf, a member of class TAdapterHiddenFieldsList.

173.1.22.4.5 TAdapterHiddenFieldsList.GetField Method

Pascal

```
function GetField(I: Integer): IAdapterHiddenField;
```

C++

```
__fastcall IAdapterHiddenField GetField(int I);
```

Description

This is GetField, a member of class TAdapterHiddenFieldsList.

173.1.22.4.6 TAdapterHiddenFieldsList.GetFieldCount Method

Pascal

```
function GetFieldCount: Integer;
```

C++

```
__fastcall int GetFieldCount();
```

Description

This is GetFieldCount, a member of class TAdapterHiddenFieldsList.

173.1.22.4.7 TAdapterHiddenFieldsList.NextIteration Method

Advances to the objects next item.

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

Server-side script calls NextIteration to move to the next item.

OwnerData is a pointer to information that the object can use to influence the iteration process. Each object stores and uses this information in its own way. This information is first supplied by the initial call to the StartIterator method. NextIteration can modify the information, after which it is supplied to the next call to the NextIteration or EndIterator method.

NextIteration returns true if it successfully moves to the next item, false if there were no more items.

See Also

startiterator (see page 19724)

enditerator (see page 19723)

173.1.22.4.8 TAdapterHiddenFieldsList.StartIterator Method

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

This is StartIterator, a member of class TAdapterHiddenFieldsList.

173.1.22.5 TAdapterHiddenFieldsList Properties

The properties of the TAdapterHiddenFieldsList class are listed here.

Protected Properties

	Name	Description
R	FieldCount (see page 23868)	This is FieldCount, a member of class TAdapterHiddenFieldsList.
R	Fields (see page 23868)	This is Fields, a member of class TAdapterHiddenFieldsList.

173.1.22.5.1 TAdapterHiddenFieldsList.FieldCount Property

Pascal

```
property FieldCount: Integer;
```

C++

```
__property int FieldCount;
```

Description

This is FieldCount, a member of class TAdapterHiddenFieldsList.

173.1.22.5.2 TAdapterHiddenFieldsList.Fields Property

Pascal

```
property Fields [I: Integer]: IAdapterHiddenField;
```

C++

```
__property IAdapterHiddenField Fields[int I];
```

Description

This is Fields, a member of class TAdapterHiddenFieldsList.

173.1.23 TAdapterImageField Class

TAdapterImageField is an event handler based adapter field that can provide the URL of an image file and also accept an uploaded image file.

Class Hierarchy

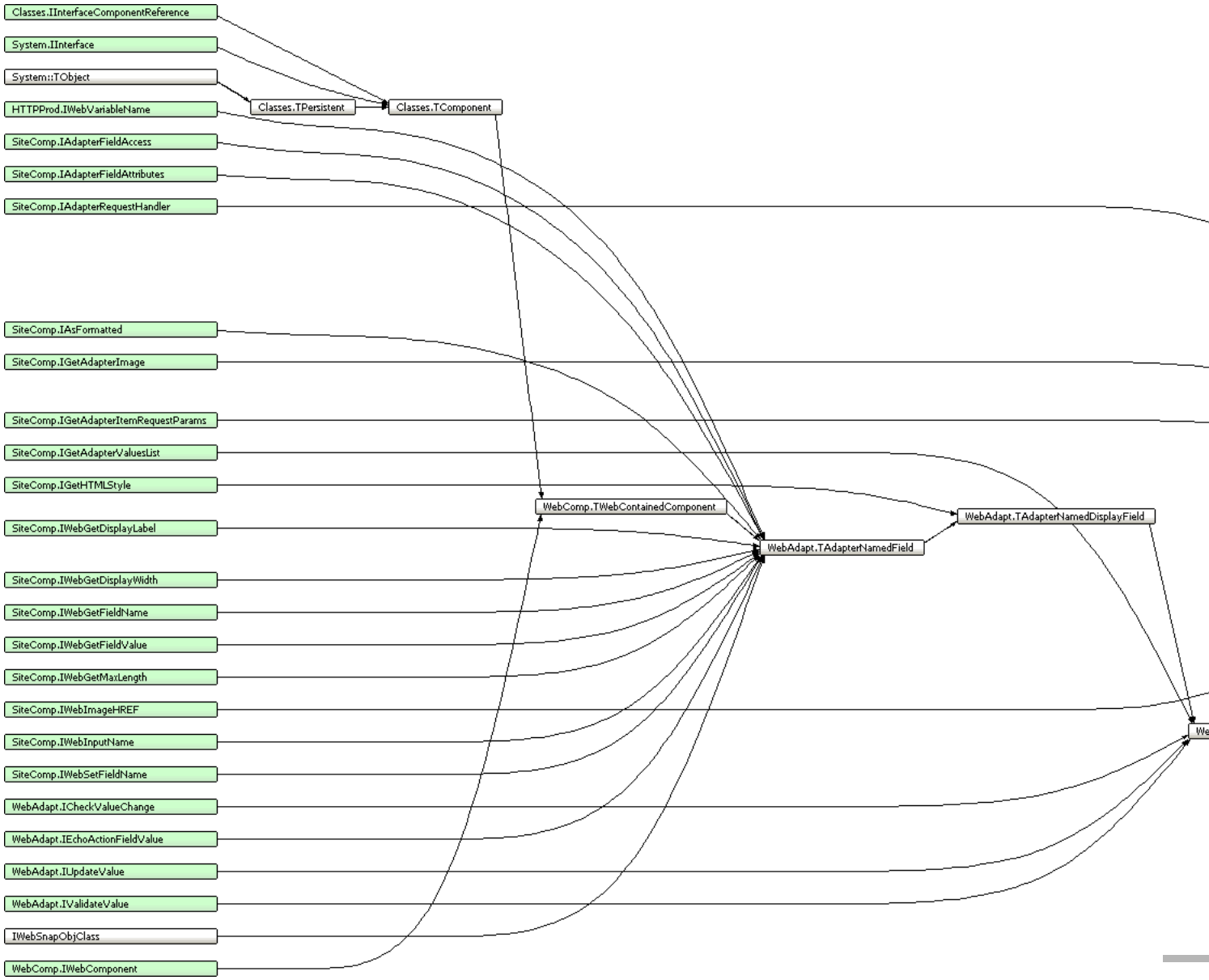

Pascal

```
TAdapterImageField = class(TCustomAdapterImageField);
```

C++

```
class TAdapterImageField : public TCustomAdapterImageField;
```

File

WebAdapt

Description

TAdapterImageField publishes the properties in TCustomAdapterImageField (see page 24244). TCustomAdapterImageField (see page 24244) calls event handlers to retrieve the adapter field's image file name, image file, and image file URL. It also defines event handler that are called to validate files uploaded by the end-user, and update the adapter field's image file.

Add TAdapterFileField (see page 23837) to TAdapter (see page 23713) or other adapter components types when you need an adapter field that supports an image using event handlers. There are other adapter field components that use event handlers. They differ in the type and number of values that are supported. See TAdapterField (see page 23818), TAdapterMultiValueField (see page 23900), TAdapterMemoField (see page 23889), TAdapterBooleanField (see page 23758) and TAdapterFileField (see page 23837).

See Also

TCustomAdapterImageField (see page 24244)

TAdapter (see page 23713)

TAdapterField (see page 23818)

TAdapterMemoField (see page 23889)

TAdapterMultiValueField (see page 23900)

TAdapterBooleanField (see page 23758)

TAdapterFileField (see page 23837)

173.1.23.1 TAdapterImageField Members

The following tables list the members exposed by TAdapterImageField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 24015)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 24015)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 24015)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 24016)	Retrieves a value entered by the end-user.
	CreateRequestContext (see page 24252)	Creates AdapterRequest and AdapterResponse objects.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoOnValidateValue (see page 24016)	Validates field values entered by the end-user.
	DoOnValidateValue (see page 24253)	Validates files uploaded by the end-user.
	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterImage (see page 24253)	Provides an object that can be used to access the image properties of this adapter field.
	GetAdapterItemRequestParams (see page 24253)	Provides a unique identifier and a set of name/value pairs that will be used to create a URL and also be used later when a request is received to generate the adapter field's image file.
	GetAdapterValuesList (see page 24016)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
	GetDisplayStyleType (see page 24254)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 23921)	Suggests an HTML input type for this adapter field.
	GetInputStyleType (see page 24017)	Indicates the preferred HTML element to use when editing this adapter field's value.
	GetInputStyleType (see page 24254)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetValueCount (see page 24017)	Provides the number of values associated with this adapter field.
	GetValues (see page 24017)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HandleRequest (see page 24255)	Handles a request for the image file associated with this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 24018)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.

	ImplCheckValueChange (see page 24018)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplCheckValueChange (see page 24255)	Indicates whether the end-user may have uploaded a new image file for this adapter field.
	ImplCreateRequestContext (see page 24255)	Creates AdapterRequest and AdapterResponse objects.
	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 24018)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 24019)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 24019)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
	ImplGetValue (see page 24256)	Returns the name of the image file associated with this adapter field.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 24019)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHandleRequest (see page 24256)	Handles a request for the image file associated with this adapter field.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 24020)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 24020)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 24020)	Updates the adapter field with value(s) entered by the end-user.
	ImplUpdateValue (see page 24257)	Updates the adapter field with value entered by the end-user.
	ImplValidateValue (see page 24020)	Validates values(s) entered by the end-user.
	ImplWebImageHREF (see page 24257)	Returns a URL that can be used to access the image file associated with this adapter field.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 24021)	Indicates whether this adapter field can support multiple values.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24021)	Clears references to a deleted component.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RenderAdapterImage (see page 24258)	Responds to a request for an image file by providing a stream containing the image.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 24021)	Updates the adapter field with value(s) entered by the end-user.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateValue (see page 24022)	Validates values(s) entered by the end-user.
	WebImageHREF (see page 24258)	Returns a URL that can be used to access the image file associated with this adapter field.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 24023)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 24024)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
V	Create (see page 24014)	Creates an instance of TAdapterUpdateField (see page 24007).
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterImage (see page 19691)	This is GetAdapterImage, a member of class IGetAdapterImage.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetAdapterValuesList (see page 19695)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.

	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.
	WebImageHREF (see page 19766)	Retrieves a URL for an external image file.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.

R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	Value (see page 24022)	Retrieves the value associated with this adapter field.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetHREF (see page 23880)	Provides the URL of an external image file associated with this adapter field.
	OnGetImage (see page 23880)	Retrieves a stream containing the image file associated with this adapter field.
	OnGetImageName (see page 23881)	Returns the name of the image file associated with this adapter field.
	OnGetParams (see page 23881)	Provides name/value pairs that identify this adapter field's image.
	OnUploadFile (see page 23881)	Updates this adapter field with an image file uploaded by the end-user.
	OnValidateFile (see page 23882)	Used to validate files uploaded by the end user.

Published Properties

	Name	Description
	DisplayLabel (see page 23876)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23877)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 23877)	Identifies the name of this field when using server-side script.
	HREF (see page 23877)	Defines a URL that references the image associated with the adapter field.
	ModifyAccess (see page 23878)	Identifies the rights required to update this field.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ReadOnly (see page 23878)	Indicates whether this adapter field is read-only.
	Required (see page 23879)	Indicates whether the end-user must enter a value for this adapter field.
	Tag (see page 2808)	Stores an integer value as part of a component.
	ValuesList (see page 23879)	Associates a list of values with the adapter field.
	ViewAccess (see page 23879)	Identifies the rights required to view this field.

173.1.23.2 TAdapterImageField Properties

The properties of the TAdapterImageField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 23876)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23877)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 23877)	Identifies the name of this field when using server-side script.
	HREF (see page 23877)	Defines a URL that references the image associated with the adapter field.
	ModifyAccess (see page 23878)	Identifies the rights required to update this field.
	ReadOnly (see page 23878)	Indicates whether this adapter field is read-only.
	Required (see page 23879)	Indicates whether the end-user must enter a value for this adapter field.
	ValuesList (see page 23879)	Associates a list of values with the adapter field.
	ViewAccess (see page 23879)	Identifies the rights required to view this field.

173.1.23.2.1 TAdapterImageField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 23933)

173.1.23.2.2 TAdapterImageField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 23933)

173.1.23.2.3 TAdapterImageField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

173.1.23.2.4 TAdapterImageField.HREF Property

Defines a URL that references the image associated with the adapter field.

Pascal

```
property HREF: string;
```

C++

```
__property AnsiString HREF;
```

Description

Use the HREF property when this adapter field is associated with an external image file URL. Use the OnGetHREF event to dynamically define an external image file URL. If the image file will be dynamically generated, use the OnGetImage and OnGetImageParams event handlers instead of the HREF property or the OnGetHREF event handler.

See Also

ImplWebImageHREF (see page 24257)

OnGetHREF (see page 24260)

OnGetImage (see page 24260)

OnGetImageName (see page 24260)

173.1.23.2.5 TAdapterImageField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property AnsiString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

hasmodifyaccess (see page 24162)

173.1.23.2.6 TAdapterImageField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 23921)

173.1.23.2.7 TAdapterImageField.Required Property

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;__property Boolean Required;
```

Description

Use this property to indicate whether a value must be entered by the end user before submitting a form. Typically, an exception is raised when a required adapter field is not found when processing an adapter action.

See Also

GetRequired (see page 23934)

173.1.23.2.8 TAdapterImageField.ValuesList Property

Associates a list of values with the adapter field.

Pascal

```
property ValuesList: IValuesListAdapter;
```

C++

```
__property IValuesListAdapter ValuesList;
```

Description

Assign a component to the ValuesList property to define possible values for the adapter field. Server-side script can be used to enumerate the name/value pairs of the ValuesList component to generate HTML select lists, check boxes, or radio buttons.

See Also

IValuesListAdapter (see page 19740)

TStringsValuesList (see page 24540)

TDataSetValuesList (see page 8079)

173.1.23.2.9 TAdapterImageField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property AnsiString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

HasViewAccess (see page 24162)

173.1.23.3 TAdapterImageField Events

The events of the TAdapterImageField class are listed here.

Published Events

	Name	Description
	OnGetHREF (see page 23880)	Provides the URL of an external image file associated with this adapter field.
	OnGetImage (see page 23880)	Retrieves a stream containing the image file associated with this adapter field.
	OnGetImageName (see page 23881)	Returns the name of the image file associated with this adapter field.
	OnGetParams (see page 23881)	Provides name/value pairs that identify this adapter field's image.
	OnUploadFile (see page 23881)	Updates this adapter field with an image file uploaded by the end-user.
	OnValidateFile (see page 23882)	Used to validate files uploaded by the end user.

173.1.23.3.1 TAdapterImageField.OnGetHREF Event

Provides the URL of an external image file associated with this adapter field.

Pascal

```
property OnGetHREF: TImageFieldGetHREFEvent;
```

C++

```
__property TImageFieldGetHREFEvent OnGetHREF;
```

Description

Implement OnGetHREF to dynamically identify the URL of an external image file associated with this adapter field. When OnGetHREF is called, the HREF parameter will contain the value of the HREF property. To dynamically generate an image file for this adapter file, implement the OnGetImage event handler instead.

See Also

ImplWebImageHREF (see page 24257)

HREF (see page 24259)

OnGetImage (see page 24260)

173.1.23.3.2 TAdapterImageField.OnGetImage Event

Retrieves a stream containing the image file associated with this adapter field.

Pascal

```
property OnGetImage: TImageFieldGetImageEvent;
```

C++

```
__property TImageFieldGetImageEvent OnGetImage;
```

Description

Implement OnGetImage to dynamically generate an image file associated with this adapter field. Don't implement OnGetImage if the image file is external. Instead implement OnGetHREF event handler or use the HREF property to specify the URL of an

external image.

See Also

RenderAdapterImage (see page 24258)

OnGetParams (see page 24261)

OnGetHREF (see page 24260)

HREF (see page 24259)

173.1.23.3.3 TAdapterImageField.OnGetImageName Event

Returns the name of the image file associated with this adapter field.

Pascal

```
property OnGetImageName: TAdapterFieldGetStringEvent;
```

C++

```
__property TAdapterFieldGetStringEvent OnGetImageName;
```

Description

Implement this event handler to identify the name of the image file associated with this adapter field.

See Also

ImplGetValue (see page 24256)

173.1.23.3.4 TAdapterImageField.OnGetParams Event

Provides name/value pairs that identify this adapter field's image.

Pascal

```
property OnGetParams: TImageFieldGetParamsEvent;
```

C++

```
__property TImageFieldGetParamsEvent OnGetParams;
```

Description

Implement this event handler to provide name/value pairs that are needed to identify this adapter field's image file. OnGetParams is called while generating an HTML page with a URL reference to this adapter field's image. The parameters returned by OnGetParams will be passed to OnGetImage when the image file is requested.

See Also

GetAdapterItemRequestParams (see page 24253)

OnGetImage (see page 24260)

173.1.23.3.5 TAdapterImageField.OnUploadFile Event

Updates this adapter field with an image file uploaded by the end-user.

Pascal

```
property OnUploadFile: TAdapterFieldFileEvent;
```

C++

```
__property TAdapterFieldFileEvent OnUploadFile;
```

Description

This event handler is called by ImplUpdateValue. This event handler should store the uploaded image file. Raise an exception to indicate an error.

See Also

ImplUpdateValue (see page 24257)

TAbstractWebRequestFile (see page 15073)

173.1.23.3.6 TAdapterImageField.OnValidateFile Event

Used to validate files uploaded by the end user.

Pascal

```
property OnValidateFile: TAdapterFieldFileEvent;
```

C++

```
__property TAdapterFieldFileEvent OnValidateFile;
```

Description

Validates the file uploaded by the end-user. Raise an exception to indicate an error. For example, it raises an exception if the Mime type of the file is not supported.

See Also

DoOnValidateValue (see page 24253)

TAbstractWebRequestFile (see page 15073)

173.1.24 TAdapterLoginField Class

TAdaperLoginField is a base class for adapter field components that are children of a TLoginFormAdapter (see page 24500) component.

Class Hierarchy

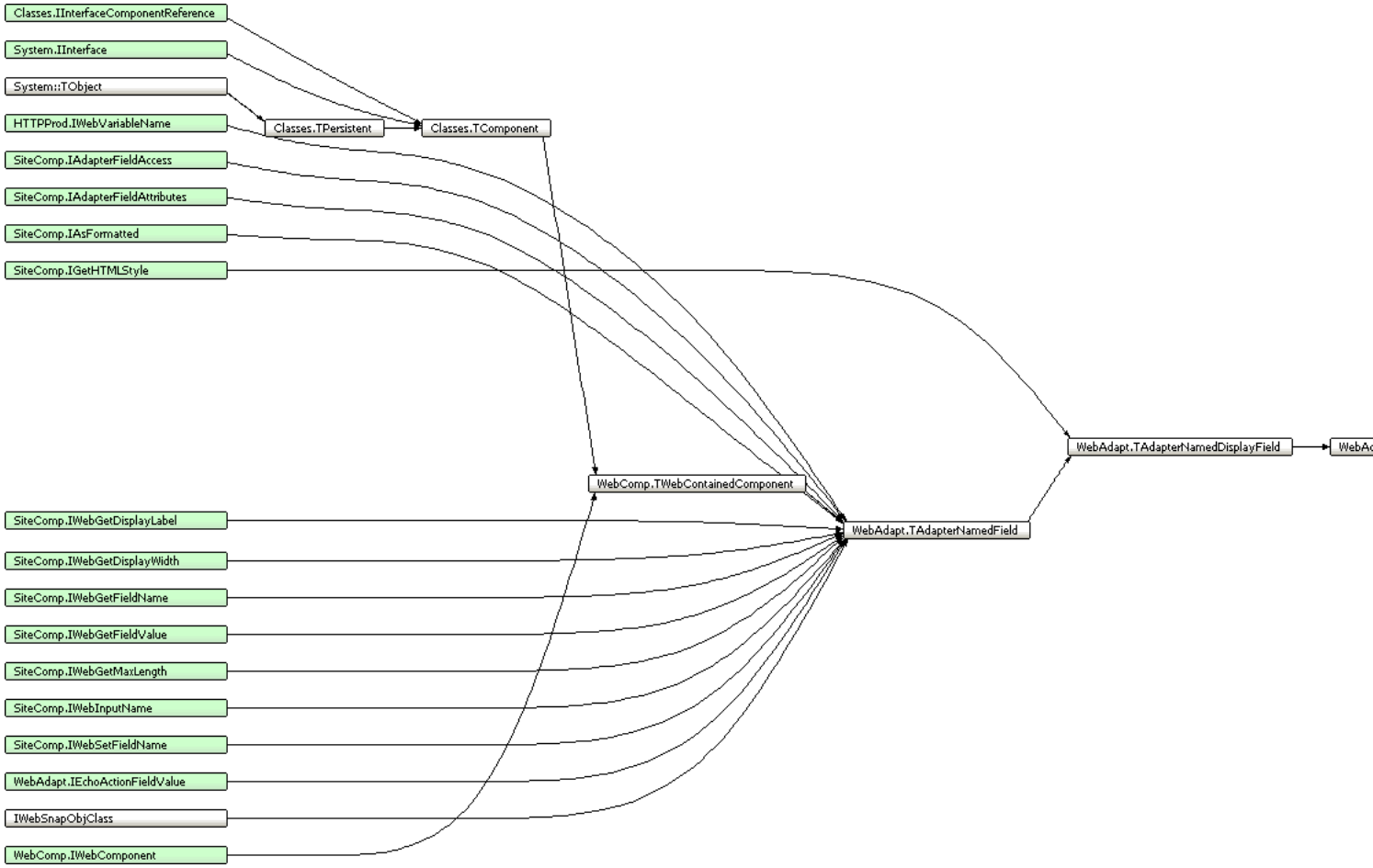

Pascal

```
TAdapterLoginField = class(TAdapterNamedDisplayField);
```

C++

```
class TAdapterLoginField : public TAdapterNamedDisplayField;
```

File

WebAdapt

Description

TAdapterLoginField implements functionality that is used by an adapter field that is a child of TLoginFormAdapter (see page 24500).

See Also

TLoginFormAdapter (see page 24500)

TAdapterNamedDisplayField (see page 23913)

TAdapterUserNameField (see page 24024)

TAdapterPasswordField (see page 23971)

173.1.24.1 TAdapterLoginField Members

The following tables list the members exposed by TAdapterLoginField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 23921)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.

	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
V	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
V	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
V	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
V	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
V	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.

	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.

R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	Adapter (see page 23888)	Returns the parent adapter of this adapter field.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 23923)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23924)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 23924)	Indicates whether this adapter field is read-only.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.24.2 TAdapterLoginField Properties

The properties of the TAdapterLoginField class are listed here.

Public Properties

	Name	Description
R	Adapter (see page 23888)	Returns the parent adapter of this adapter field.

173.1.24.2.1 TAdapterLoginField.Adapter Property

Returns the parent adapter of this adapter field.

Pascal

```
property Adapter: TCustomLoginFormAdapter;
```

C++

```
__property TCustomLoginFormAdapter Adapter;
```

Description

Descendants use this property to access the parent adapter. GetAdapter (see page 23932) implements this property.

See Also

TCustomLoginFormAdapter (see page 24344)

GetAdapter

173.1.25 TAdapterMemoField Class

TAdapterMemoField is an event handler based adapter field that supports a multi-line text value.

Class Hierarchy

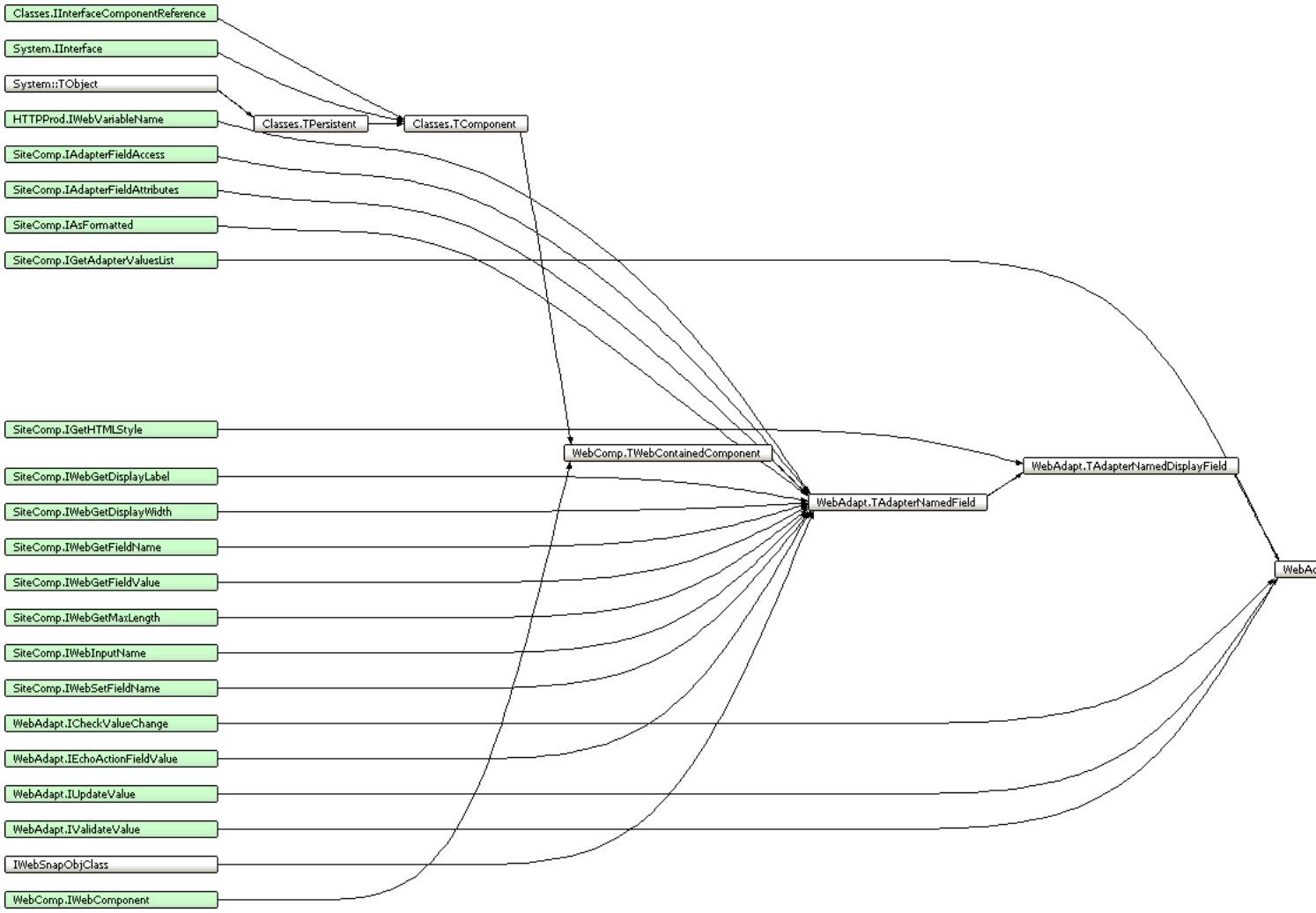

Pascal

```
TAdapterMemoField = class(TCustomAdapterMemoField);
```

C++

```
class TAdapterMemoField : public TCustomAdapterMemoField;
```

File

WebAdapt

Description

TAdapterMemoField publishes the properties in TCustomAdapterMemoField (see page 24262). TCustomAdapterMemoField (see page 24262) calls event handlers to retrieve the adapter field's text value, validate a value entered end-user, and update the adapter field's value.

Add TAdapterMemoField to TAdapter (see page 23713) or other adapter components types when you need an adapter field that supports a multi-line text value. There are other adapter field components that use event handlers. They differ in the type and number of values that are supported. See TAdapterField (see page 23818), TAdapterMultiValueField (see page 23900), TAdapterFileField (see page 23837), TAdapterBooleanField (see page 23758) and TAdapterImageField (see page

23869).

See Also

TCustomAdapterMemoField (see page 24262)

TAdapter (see page 23713)

TAdapterField (see page 23818)

TAdapterFileField (see page 23837)

TAdapterMultiValueField (see page 23900)

TAdapterBooleanField (see page 23758)

TAdapterImageField (see page 23869)

173.1.25.1 TAdapterMemoField Members

The following tables list the members exposed by TAdapterMemoField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 24015)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 24015)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 24015)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 24016)	Retrieves a value entered by the end-user.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoOnValidateValue (see page 24016)	Validates field values entered by the end-user.
V	DoOnValidateValue (see page 24269)	Validates a value entered by the end user.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 24016)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.

	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
	GetDisplayStyleType (see page 24270)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 23921)	Suggests an HTML input type for this adapter field.
	GetInputStyleType (see page 24017)	Indicates the preferred HTML element to use when editing this adapter field's value.
	GetInputStyleType (see page 24270)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetValueCount (see page 24017)	Provides the number of values associated with this adapter field.
	GetValues (see page 24017)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 24018)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 24018)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplCheckValueChange (see page 24270)	Indicates whether the end-user may have entered a new value for this adapter field.
	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 24018)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 24019)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 24019)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
	ImplGetValue (see page 24271)	Returns the value of this adapter field as a variant.

	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 24019)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 24020)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 24020)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 24020)	Updates the adapter field with value(s) entered by the end-user.
	ImplUpdateValue (see page 24271)	Updates the adapter field with value entered by the end-user.
	ImplValidateValue (see page 24020)	Validates values(s) entered by the end-user.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 24021)	Indicates whether this adapter field can support multiple values.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24021)	Clears references to a deleted component.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 24021)	Updates the adapter field with value(s) entered by the end-user.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateValue (see page 24022)	Validates values(s) entered by the end-user.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 24023)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 24024)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
V	Create (see page 24014)	Creates an instance of TAdapterUpdateField (see page 24007).
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterValuesList (see page 19695)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	Value (see page 24022)	Retrieves the value associated with this adapter field.
	ValuesList (see page 24023)	Associates a list of values with the adapter field.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetValue (see page 23899)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 23900)	Called to update the adapter field's value with a text value entered by the end-user.
	OnUploadFile (see page 23900)	Called to update the adapter field's value with a file uploaded by the end-user.

Published Properties

	Name	Description
	DisplayLabel (see page 23897)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23897)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 23898)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 23898)	Identifies the rights required to update this field.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ReadOnly (see page 23898)	Indicates whether this adapter field is read-only.
	Required (see page 23899)	Indicates whether the end-user must enter a value for this adapter field.
	Tag (see page 2808)	Stores an integer value as part of a component.
	ViewAccess (see page 23899)	Identifies the rights required to view this field.

173.1.25.2 TAdapterMemoField Properties

The properties of the TAdapterMemoField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 23897)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23897)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 23898)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 23898)	Identifies the rights required to update this field.
	ReadOnly (see page 23898)	Indicates whether this adapter field is read-only.
	Required (see page 23899)	Indicates whether the end-user must enter a value for this adapter field.
	ViewAccess (see page 23899)	Identifies the rights required to view this field.

173.1.25.2.1 TAdapterMemoField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 23933)

173.1.25.2.2 TAdapterMemoField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 23933)

173.1.25.2.3 TAdapterMemoField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

173.1.25.2.4 TAdapterMemoField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property AnsiString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

hasmodifyaccess (see page 24162)

173.1.25.2.5 TAdapterMemoField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 23921)

173.1.25.2.6 TAdapterMemoField.Required Property

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;__property Boolean Required;
```

Description

Use this property to indicate whether a value must be entered by the end user before submitting a form. Typically, an exception is raised when a required adapter field is not found when processing an adapter action.

See Also

GetRequired (see page 23934)

173.1.25.2.7 TAdapterMemoField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property AnsiString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

HasViewAccess (see page 24162)

173.1.25.3 TAdapterMemoField Events

The events of the TAdapterMemoField class are listed here.

Published Events

	Name	Description
	OnGetValue (see page 23899)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 23900)	Called to update the adapter field's value with a text value entered by the end-user.
	OnUploadFile (see page 23900)	Called to update the adapter field's value with a file uploaded by the end-user.

173.1.25.3.1 TAdapterMemoField.OnGetValue Event

Called to retrieve the adapter field's value.

Pascal

```
property OnGetValue: TAdapterFieldGetStringEvent;
```

C++

```
__property TAdapterFieldGetStringEvent OnGetValue;
```

Description

OnGetValue is called by ImplGetValue to retrieve the value associated with this adapter field.

See Also

ImplGetValue (see page 24271)

173.1.25.3.2 TAdapterMemoField.OnUpdateValue Event

Called to update the adapter field's value with a text value entered by the end-user.

Pascal

```
property OnUpdateValue: TAdapterFieldStringEvent;
```

C++

```
__property TAdapterFieldStringEvent OnUpdateValue;
```

Description

OnUpdateValue is called when an action request contains a value for this adapter field. If this event handler is not implemented, the value entered by the end-user will be ignored.

See Also

ImplUpdateValue (see page 24271)

173.1.25.3.3 TAdapterMemoField.OnUploadFile Event

Called to update the adapter field's value with a file uploaded by the end-user.

Pascal

```
property OnUploadFile: TAdapterFieldFileEvent;
```

C++

```
__property TAdapterFieldFileEvent OnUploadFile;
```

Description

OnUploadFile is called when an action request contains an uploaded file for this adapter field. If this event handler is not implemented, the file uploaded by the end-user will be ignored.

See Also

ImplUpdateValue (see page 24271)

173.1.26 TAdapterMultiValueField Class

TAdapterMultiValueField is an event handler based adapter field that provides zero or more values.

Class Hierarchy

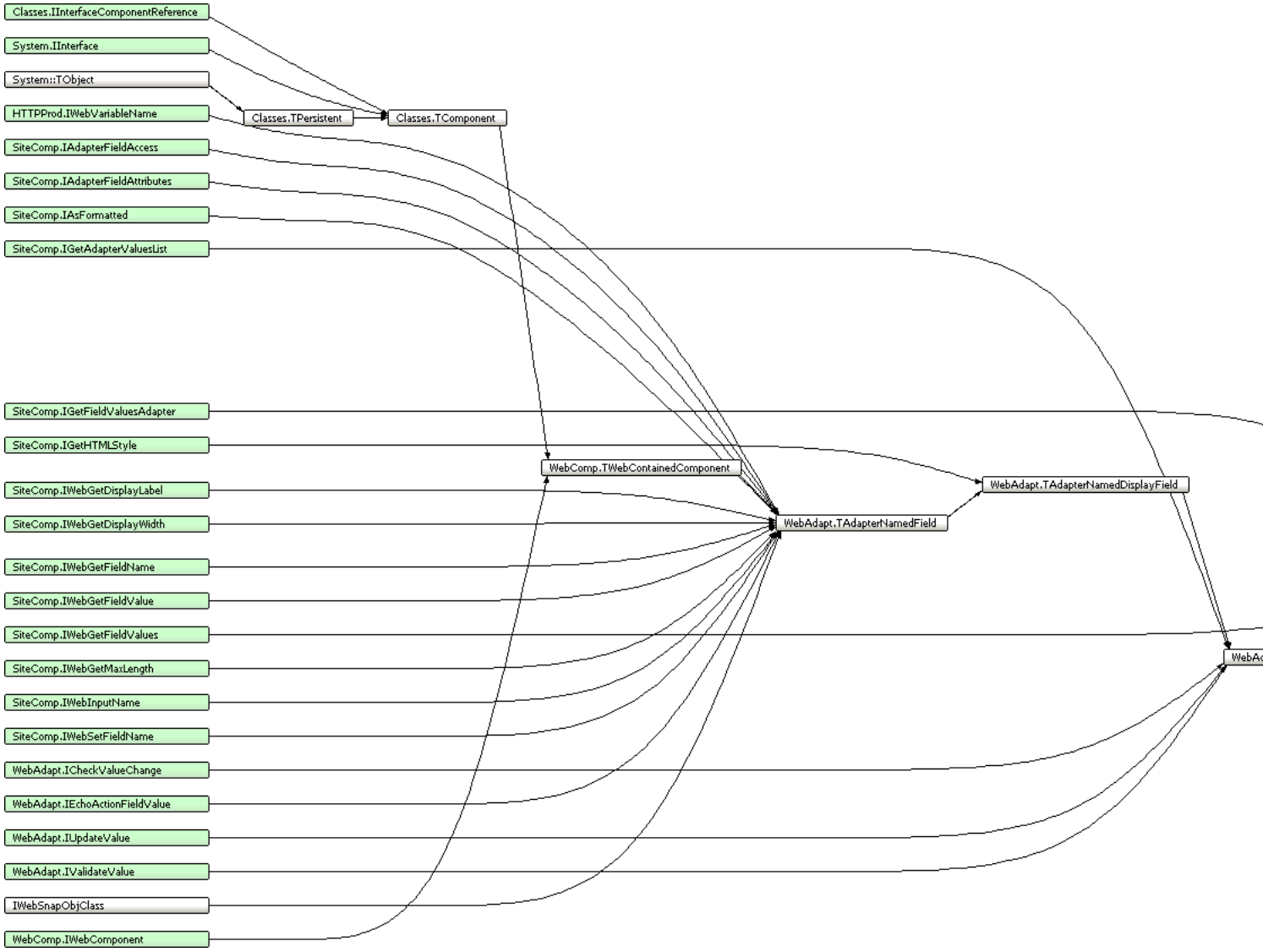

Pascal

```
TAdapterMultiValueField = class(TCustomAdapterMultiValueField);
```

C++

```
class TAdapterMultiValueField : public TCustomAdapterMultiValueField;
```

File

WebAdapt

Description

TAdapterMultiValueField publishes the properties in TCustomAdapterMultiValueField (see page 24273). TCustomAdapterMultiValueField (see page 24273) calls event handlers to retrieve the adapter field's values, validate values entered by the end-user, and update the adapter field's values.

Add TAdapterMultiValueField to TAdapter (see page 23713) or other adapter components types when you need an adapter

field that supports a multiple values using event handlers. There are other adapter field components that use event handlers. They differ in the type and number of values that are supported. See TAdapterField (see page 23818), TAdapterFileField (see page 23837), TAdapterMemoField (see page 23889), TAdapterBooleanField (see page 23758) and TAdapterImageField (see page 23869).

See Also

TCustomAdapterMultiValueField (see page 24273)

TAdapter (see page 23713)

TAdapterField (see page 23818)

TAdapterMemoField (see page 23889)

TAdapterFileField (see page 23837)

TAdapterBooleanField (see page 23758)

TAdapterImageField (see page 23869)

173.1.26.1 TAdapterMultiValueField Members

The following tables list the members exposed by TAdapterMultiValueField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 24015)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 24015)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 24015)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 24016)	Retrieves a value entered by the end-user.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoOnValidateValue (see page 24016)	Validates field values entered by the end-user.
V	DoOnValidateValue (see page 24281)	Validates values entered by the end-user.
	EndValueIterator (see page 24093)	Completes iteration.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 24016)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.

	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetCurrentValue (see page 24094)	Implements IWebGetFieldValues' GetCurrentValue.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
V	GetDisplayStyleType (see page 24281)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetFieldValuesAdapter (see page 24094)	Returns the adapter object that implements the Values property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 23921)	Suggests an HTML input type for this adapter field.
V	GetInputStyleType (see page 24017)	Indicates the preferred HTML element to use when editing this adapter field's value.
V	GetInputStyleType (see page 24282)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
V	GetValueCount (see page 24017)	Provides the number of values associated with this adapter field.
V	GetValueCount (see page 24094)	Returns the number of values associated with this adapter field.
V	GetValueCount (see page 24282)	Returns the number of values associated with this adapter field.
V	GetValues (see page 24017)	Returns one of the values provided by this adapter field.
V	GetValues (see page 24095)	Returns one of the values associated with the adapter field.
V	GetValues (see page 24282)	Returns one of the values associated with the adapter field.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasValue (see page 24095)	Implements IWebGetFieldValues' HasValue method.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
V	ImplAsFormatted (see page 24018)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
V	ImplCheckValueChange (see page 24018)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
V	ImplGetCurrentValue (see page 24095)	Implements IWebGetFieldValues' GetCurrentValue (see page 24094) method.
V	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
V	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayLabel (see page 24018)	Gets the preferred display label for this adapter field.
V	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
V	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.

	ImplGetDisplayWidth (see page 24019)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 24019)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetFieldValuesAdapter (see page 24096)	Returns the adapter object that implements the Values property.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 24019)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasValue (see page 24096)	Implements IWebGetFieldValues' HasValue (see page 24095) method.
	ImplHasValue (see page 24283)	Implements IWebGetFieldValues' HasValue (see page 24095) method.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 24020)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 24020)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 24020)	Updates the adapter field with value(s) entered by the end-user.
	ImplUpdateValue (see page 24283)	Updates the adapter field with value(s) entered by the end-user.
	ImplValidateValue (see page 24020)	Validates values(s) entered by the end-user.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 24021)	Indicates whether this adapter field can support multiple values.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextValueIteration (see page 24096)	Implements IWebGetFieldValues' NextValueIteration method.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24021)	Clears references to a deleted component.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.

	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartValueIterator (see page 24097)	Implements IWebGetFieldValues' StartValueIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 24021)	Updates the adapter field with value(s) entered by the end-user.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateValue (see page 24022)	Validates values(s) entered by the end-user.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 24023)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 24024)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
V	Create (see page 24014)	Creates an instance of TAdapterUpdateField (see page 24007).
V	Create (see page 24092)	Constructor for TBaseAdapterMultiValueField (see page 24086).
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.

	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24093)	Destructor for TBaseAdapterMultiValueField (see page 24086).
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndValueIterator (see page 19761)	Indicate the iteration is finished.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterValuesList (see page 19695)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetCurrentValue (see page 19761)	Retrieves the current value as a variant.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetFieldValuesAdapter (see page 19698)	This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasValue (see page 19761)	Indicates whether the adapter field has a matching value.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextValueIteration (see page 19762)	Position the iteration pointer on the next value.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartValueIterator (see page 19762)	Reset the iteration pointer to the first item.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	Value (see page 24022)	Retrieves the value associated with this adapter field.
R	ValueCount (see page 24284)	Returns the number of values associated with this adapter field.
R	Values (see page 24284)	Returns one of the values associated with the adapter field.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetValueCount (see page 23911)	Responsible for returning the count of values associated with the adapter field.
	OnGetValues (see page 23912)	Responsible for returning one of the values associated with the adapter field.
	OnHasValue (see page 23912)	Indicates whether a particular value is associated with this adapter field.
	OnUpdateValues (see page 23912)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateValues (see page 23913)	Validates field values entered by the end-user.

Published Properties

	Name	Description
	DisplayLabel (see page 23908)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23909)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 23909)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 23909)	Identifies the rights required to update this field.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ReadOnly (see page 23910)	Indicates whether this adapter field is read-only.
	Required (see page 23910)	Indicates whether the end-user must enter a value for this adapter field.
	Tag (see page 2808)	Stores an integer value as part of a component.
	ValuesList (see page 23910)	Associates a list of values with the adapter field.
	ViewAccess (see page 23911)	Identifies the rights required to view this field.

173.1.26.2 TAdapterMultiValueField Properties

The properties of the TAdapterMultiValueField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 23908)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23909)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 23909)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 23909)	Identifies the rights required to update this field.
	ReadOnly (see page 23910)	Indicates whether this adapter field is read-only.
	Required (see page 23910)	Indicates whether the end-user must enter a value for this adapter field.
	ValuesList (see page 23910)	Associates a list of values with the adapter field.
	ViewAccess (see page 23911)	Identifies the rights required to view this field.

173.1.26.2.1 TAdapterMultiValueField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 23933)

173.1.26.2.2 TAdapterMultiValueField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 23933)

173.1.26.2.3 TAdapterMultiValueField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

173.1.26.2.4 TAdapterMultiValueField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property AnsiString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

hasmodifyaccess (see page 24162)

173.1.26.2.5 TAdapterMultiValueField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 23921)

173.1.26.2.6 TAdapterMultiValueField.Required Property

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;__property Boolean Required;
```

Description

Use this property to indicate whether a value must be entered by the end user before submitting a form. Typically, an exception is raised when a required adapter field is not found when processing an adapter action.

See Also

GetRequired (see page 23934)

173.1.26.2.7 TAdapterMultiValueField.ValuesList Property

Associates a list of values with the adapter field.

Pascal

```
property ValuesList: IValuesListAdapter;
```

C++

```
__property IValuesListAdapter ValuesList;
```

Description

Assign a component to the ValuesList property to define possible values for the adapter field. Server-side script can be used to enumerate the name/value pairs of the ValuesList component to generate HTML select lists, check boxes, or radio buttons.

See Also

IValuesListAdapter (see page 19740)

TStringsValuesList (see page 24540)

TDataSetValuesList (see page 8079)

173.1.26.2.8 TAdapterMultiValueField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property AnsiString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

HasViewAccess (see page 24162)

173.1.26.3 TAdapterMultiValueField Events

The events of the TAdapterMultiValueField class are listed here.

Published Events

	Name	Description
	OnGetValueCount (see page 23911)	Responsible for returning the count of values associated with the adapter field.
	OnGetValues (see page 23912)	Responsible for returning one of the values associated with the adapter field.
	OnHasValue (see page 23912)	Indicates whether a particular value is associated with this adapter field.
	OnUpdateValues (see page 23912)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateValues (see page 23913)	Validates field values entered by the end-user.

173.1.26.3.1 TAdapterMultiValueField.OnGetValueCount Event

Responsible for returning the count of values associated with the adapter field.

Pascal

```
property OnGetValueCount: TAdapterFieldGetValueCountEvent;
```

C++

```
__property TAdapterFieldGetValueCountEvent OnGetValueCount;
```

Description

Implement this event handler to provide the count of values associated with an adapter field. Also implement OnGetValues to provide the individual values.

See Also

GetValueCount (see page 24282)

OnGetValues (see page 24285)

173.1.26.3.2 TAdapterMultiValueField.OnGetValues Event

Responsible for returning one of the values associated with the adapter field.

Pascal

```
property OnGetValues: TAdapterFieldGetMultiValueEvent;
```

C++

```
__property TAdapterFieldGetMultiValueEvent OnGetValues;
```

Description

Implement this event handler to provide the values associated with an adapter field. The Index parameter is zero-based. Also implement OnGetValueCount to provide the count of values associated with this adapter field.

See Also

GetValues (see page 24282)

OnGetValueCount (see page 24285)

173.1.26.3.3 TAdapterMultiValueField.OnHasValue Event

Indicates whether a particular value is associated with this adapter field.

Pascal

```
property OnHasValue: TAdapterFieldHasValueEvent;
```

C++

```
__property TAdapterFieldHasValueEvent OnHasValue;
```

Description

ImplHasValue calls this event handler. Implement this event handler to lookup a matching value. Implementation of this event handler is optional. If OnHasValue is nil, ImplHasValue will search for a matching value the Values and ValueCount properties.

See Also

ImplHasValue (see page 24283)

ValueCount (see page 24284)

Values (see page 24284)

173.1.26.3.4 TAdapterMultiValueField.OnUpdateValues Event

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
property OnUpdateValues: TAdapterFieldStringsEvent;
```

C++

```
__property TAdapterFieldStringsEvent OnUpdateValues;
```

Description

Implement this event handler to process values entered by the end-user. Raise an exception if an error occurs.

See Also

OnValidateValues (see page 24286)

ImplUpdateValue (see page 24283)

173.1.26.3.5 TAdapterMultiValueField.OnValidateValues Event

Validates field values entered by the end-user.

Pascal

```
property OnValidateValues: TAdapterFieldStringsEvent;
```

C++

```
__property TAdapterFieldStringsEvent OnValidateValues;
```

Description

Implement this event handler to validate values entered by the end-user. Raise an exception if an error occurs.

See Also

OnValidateValues (see page 24286)

173.1.27 TAdapterNamedDisplayField Class

The AdapterNamedDisplayField is a base class for adapter fields.

Class Hierarchy

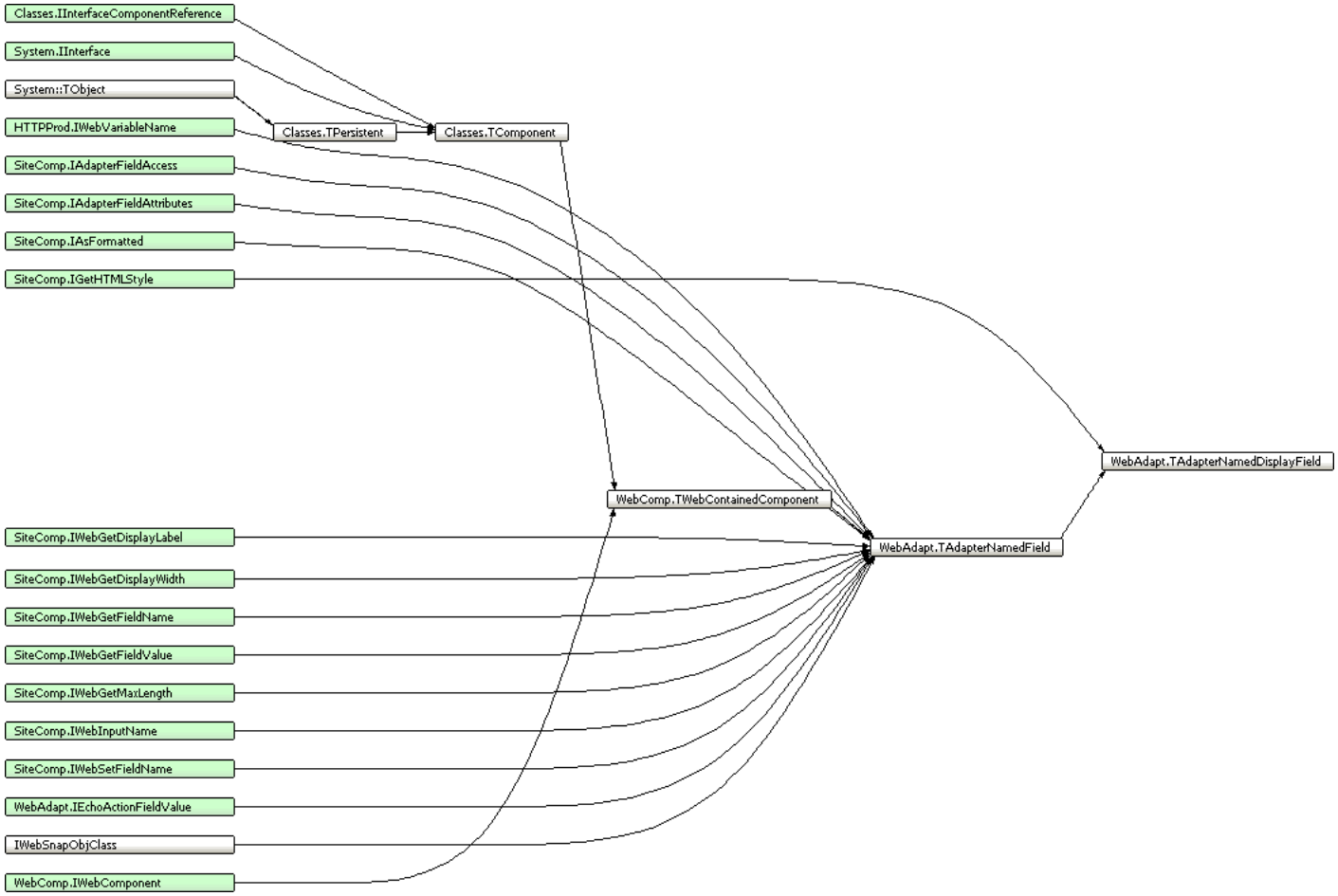

Pascal

```
TAdapterNamedDisplayField = class(TAdapterNamedField, IGetHTMLStyle);
```

C++

```
class TAdapterNamedDisplayField : public TAdapterNamedField, public IGetHTMLStyle;
```

File

WebAdapt

Description

TAdapterNameDisplayField implements the IGetHTMLStyle interface used to inquire how this adapter field should be displayed in an HTML form. For example, IGetHTMLStyle's GetInputStyle (see page 23920) method may indicate that this field should be displayed using a select list. TAdapterNamedField (see page 23925) also declares properties that may be modified at design-time to control how the adapter field should be displayed. For example, the DisplayLabel (see page 23923) property is used to suggest display label for this adapter field.

See Also

TAdapterNamedField (see page 23925)

TCustomAdapter (see page 24139)

TAdapter (see page 23713)

IGetHTMLStyle (see page 19699)

173.1.27.1 TAdapterNamedDisplayField Members

The following tables list the members exposed by TAdapterNamedDisplayField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 23921)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.

	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.

	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.

R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 23923)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23924)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 23924)	Indicates whether this adapter field is read-only.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.27.2 TAdapterNamedDisplayField.Create Constructor

Creates an instance of TAdapterNamedDisplayField (see page 23913).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TAdapterNamedDisplayField(TComponent * AOwner);
```

Description

Do not call Create. It will be called automatically when the TAdapterNamedDisplayField (see page 23913) object is created

173.1.27.3 TAdapterNamedDisplayField Methods

The methods of the TAdapterNamedDisplayField class are listed here.

Protected Methods

	Name	Description
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.

	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 23921)	Suggests an HTML input type for this adapter field.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
V	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
V	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
V	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.

173.1.27.3.1 TAdapterNamedDisplayField.GetDisplayStyle Method

Returns a string value to suggest an HTML display style for this adapter field.

Pascal

```
function GetDisplayStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall AnsiString GetDisplayStyle(const AnsiString AAdapterMode);
```

Description

GetDisplayStyle implements IGetHTMLStyle's GetDisplayStyle by translating the result value of GetDisplayStyleType (see page 23920) into a string.

See Also

GetDisplayStyle (see page 19700)

GetDisplayStyleType (see page 23920)

173.1.27.3.2 TAdapterNamedDisplayField.GetDisplayStyleType Method

Suggests an HTML display type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
virtual;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const AnsiString
AAdapterMode);
```

Description

GetDisplayStyleType returns htmldText, indicating that this adapter field should be displayed by simple text. Descendant classes may override this method to suggest other HTML display types.

See Also

GetDisplayStyle (see page 23920)

173.1.27.3.3 TAdapterNamedDisplayField.GetInputStyle Method

Returns a string value to suggest an HTML input style for this adapter field.

Pascal

```
function GetInputStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall AnsiString GetInputStyle(const AnsiString AAdapterMode);
```

Description

GetInputStyle implements IGetHTMLStyle's GetInputType method by translating the result value of GetInputStyleType (see page 23921) into a string.

See Also

GetInputStyle (see page 19700)

GetInputStyleType (see page 23921)

173.1.27.3.4 TAdapterNamedDisplayField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; virtual;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const AnsiString
AAdapterMode);
```

Description

GetInputStyleType returns htmliTextInput, indicating that this adapter field should be edited by a single line input field. Descendant classes may override this method to suggest a different HTML input type.

See Also

GetInputStyle (see page 23920)

173.1.27.3.5 TAdapterNamedDisplayField.GetViewMode Method

Returns a string value to suggest a view mode for this adapter field.

Pascal

```
function GetViewMode(const AAdapterMode: string): string;
```

C++

```
__fastcall AnsiString GetViewMode(const AnsiString AAdapterMode);
```

Description

GetViewMode implements IGetHTMLStyle's GetViewMode method by translating the result value of GetViewModeType (see page 23921) into a string.

See Also

GetViewMode (see page 19701)

GetViewModeType (see page 23921)

173.1.27.3.6 TAdapterNamedDisplayField.GetViewModeType Method

Suggests a view mode for this adapter field.

Pascal

```
function GetViewModeType(const AAdapterMode: string): TAdapterDisplayViewModeType; virtual;
```

C++

```
virtual __fastcall TAdapterDisplayViewModeType GetViewModeType(const AnsiString AAdapterMode);
```

Description

GetViewModeType returns htmlvmDisplay if the ReadOnly (see page 23924) property is true. Otherwise, htmlvmInput is returned.

See Also

ReadOnly (see page 23924)

GetViewMode (see page 23921)

173.1.27.3.7 TAdapterNamedDisplayField.ImplGetDisplayLabel Method

Implements interface access to the DisplayLabel (see page 23923) property.

Pascal

```
function ImplGetDisplayLabel: string; override;
```

C++

```
virtual __fastcall AnsiString ImplGetDisplayLabel();
```

Description

ImplGetDisplayLabel returns the value of the DisplayLabel (see page 23923) property.

See Also

DisplayLabel (see page 23923)

173.1.27.3.8 TAdapterNamedDisplayField.ImplGetDisplayWidth Method

Implements interface access to the DisplayWidth (see page 23924) property.

Pascal

```
function ImplGetDisplayWidth: Integer; override;
```

C++

```
virtual __fastcall int ImplGetDisplayWidth();
```

Description

ImplGetDisplayWidth returns the value of the DisplayWidth (see page 23924) property.

See Also

DisplayWidth (see page 23924)

173.1.27.3.9 TAdapterNamedDisplayField.ImplGetRequired Method

Implements interface access to the Required (see page 23924) property.

Pascal

```
function ImplGetRequired: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplGetRequired();
```

Description

ImplGetRequired returns the value of the Required (see page 23924) property.

See Also

Required (see page 23924)

173.1.27.3.10 TAdapterNamedDisplayField.SetDisplayLabel Method

SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.

Pascal

```
procedure SetDisplayLabel(const AValue: string);
```

C++

```
__fastcall SetDisplayLabel(const AnsiString AValue);
```

Description

Use SetDisplayLabel to set the value of the DisplayLabel (see page 23923) property to AValue.

See Also

DisplayLabel (see page 23923)

173.1.27.4 TAdapterNamedDisplayField Properties

The properties of the TAdapterNamedDisplayField class are listed here.

Public Properties

	Name	Description
	DisplayLabel (see page 23923)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23924)	Suggests a display width, in characters, for this adapter field.
	ReadOnly (see page 23924)	Indicates whether this adapter field is read-only.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.

173.1.27.4.1 TAdapterNamedDisplayField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 23933)

173.1.27.4.2 TAdapterNamedDisplayField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 23933)

173.1.27.4.3 TAdapterNamedDisplayField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode (see page 23921) which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 23921)

173.1.27.4.4 TAdapterNamedDisplayField.Required Property

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;
```

Description

Use this property to indicate whether a value must be entered by the end user before submitting a form. Typically, an exception is raised when a required adapter field is not found when processing an adapter action.

See Also

GetRequired (see page 23934)

173.1.28 TAdapterNamedField Class

TAdapterNamedField is a based class for adapter field components.

Class Hierarchy

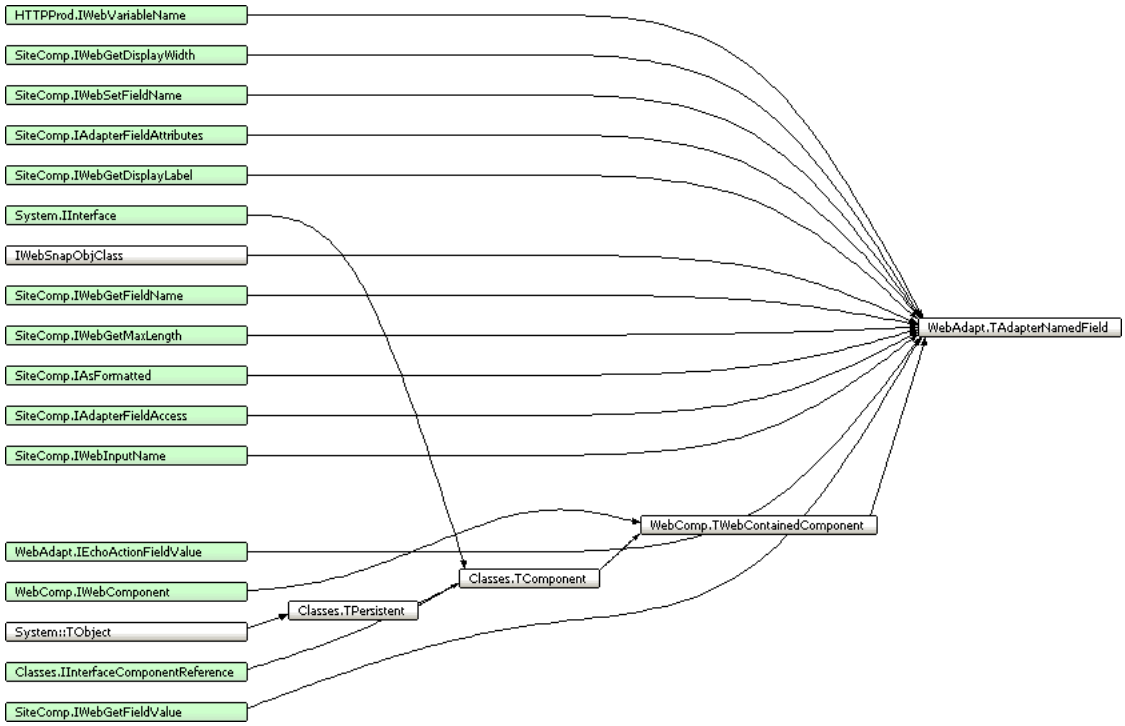

Pascal

```
TAdapterNamedField = class(TWebContainedComponent, IWebVariableName, IWebInputName,
IWebGetFieldValue, IEchoActionFieldValue, IAsFormatted, IWebGetFieldName, IWebSetFieldName,
IWebGetDisplayWidth, IWebGetDisplayLabel, IWebGetMaxLength, IWebSnapObjClass,
IAdapterFieldAccess, IAdapterFieldAttributes);
```

C++

```
class TAdapterNamedField : public TWebContainedComponent, public IWebVariableName, public
IWebInputName, public IWebGetFieldValue, public IEchoActionFieldValue, public IAsFormatted,
public IWebGetFieldName, public IWebSetFieldName, public IWebGetDisplayWidth, public
IWebGetDisplayLabel, public IWebGetMaxLength, public IWebSnapObjClass, public
IAdapterFieldAccess, public IAdapterFieldAttributes;
```

File

WebAdapt

Description

TAdapterNamedField implements interfaces used to access an adapter field's properties using server-side script. Most of the interface implementation methods call virtual functions. When defining new derived classes, override these virtual functions to

customize the behavior of the adapter field class.

See Also

TAdapterNamedDisplayField (see page 23913)

TAdapter (see page 23713)

173.1.28.1 TAdapterNamedField Members

The following tables list the members exposed by TAdapterNamedField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.

	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.

	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.

	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.28.2 TAdapterNamedField Methods

The methods of the TAdapterNamedField class are listed here.

Protected Methods

	Name	Description
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
V	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
V	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
V	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.

	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.

173.1.28.2.1 TAdapterNamedField.AsFormatted Method

Returns text used to generate a representation of the adapter field value.

Pascal

```
function AsFormatted: string;
```

C++

```
__fastcall AnsiString AsFormatted();
```

Description

This function returns a text that is typically used when generating a read-only representation of the adapter field value. The value returned by this function can be accessed with server-side script using the DisplayText property.

See Also

ImplAsFormatted (see page 23937)

173.1.28.2.2 TAdapterNamedField.CheckModifyAccess Method

Checks end-user rights to modify the adapter field value.

Pascal

```
procedure CheckModifyAccess;
```

C++

```
__fastcall CheckModifyAccess();
```

Description

This procedure is called to verify that the current user has rights to modify this field. An exception will be raised if access is denied.

See Also

ModifyAccess (see page 23944)

HasModifyAccess (see page 23936)

173.1.28.2.3 TAdapterNamedField.CheckViewAccess Method

Checks end-user rights to view the adapter field value.

Pascal

```
procedure CheckViewAccess;
```

C++

```
__fastcall CheckViewAccess();
```

Description

This procedure is called to verify that the current user has rights to view this field. An exception will be raised if access is denied.

See Also

ViewAccess (see page 23945)

HasModifyAccess (see page 23936)

173.1.28.2.4 TAdapterNamedField.FullyQualifyInputName Method

Indicates whether the HTML input names must be qualified with the name of the adapter.

Pascal

```
function FullyQualifyInputName: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean FullyQualifyInputName();
```

Description

This functions returns true if, when generating an HTML input referencing this adapter field, the name of the HTML input must be qualified with the name of the adapter component. Input names must be fully qualified when a single HTML form is used to input values for multiple adapters.

See Also

MasterAdapter (see page 7809)

173.1.28.2.5 TAdapterNamedField.GetAdapter Method

GetAdapter returns the parent adapter component.

Pascal

```
function GetAdapter: TCustomAdapter;
```

C++

```
__fastcall TCustomAdapter GetAdapter();
```

Description

To improve the efficiency of this function, a reference to the parent adapter component is cached in an instance variable.

173.1.28.2.6 TAdapterNamedField.GetDefaultDisplayLabel Method

Provides the default display label for this adapter field.

Pascal

```
function GetDefaultDisplayLabel: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetDefaultDisplayLabel();
```

Description

When the display label for the field matches this value then the display label property does not need to be stored in the form file.

173.1.28.2.7 TAdapterNamedField.GetDefaultFieldName Method

Gets the default name used to reference this adapter field with server-side script.

Pascal

```
function GetDefaultFieldName: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetDefaultFieldName();
```

Description

This function returns the default field name for the field. This name will be used when a field name has not be set using ImplSetFieldName (see page 23941). GetDefaultFieldName returns the name of the adapter field component. Descendants may override this function.

See Also

ImplGetVariableName (see page 23939)

ImplSetFieldName (see page 23941)

173.1.28.2.8 TAdapterNamedField.GetDisplayLabel Method

Returns the suggested display label for this adapter field.

Pascal

```
function GetDisplayLabel: string;
```

C++

```
__fastcall AnsiString GetDisplayLabel();
```

Description

GetDisplayLabel provides label that may be used when generating HTML to display the value of this adapter field. The value returned by this function can be accessed with server-side script using the DisplayLabel property. GetDisplayLabel calls ImplGetDisplayLabel (see page 23937).

See Also

ImplGetDisplayLabel (see page 23937)

173.1.28.2.9 TAdapterNamedField.GetDisplayWidth Method

Returns the suggested display width for this adapter field.

Pascal

```
function GetDisplayWidth: Integer;
```

C++

```
__fastcall int GetDisplayWidth();
```

Description

This function returns the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. The value returned by this function can be accessed with server-side script using the DisplayWidth property. GetDisplayWidth calls ImplGetDisplayWidth (see page 23937).

See Also

ImplGetDisplayWidth (see page 23937)

173.1.28.2.10 TAdapterNamedField.GetEchoActionFieldValue Method

Returns the value of the echo flag.

Pascal

```
function GetEchoActionFieldValue: Boolean;
```

C++

```
__fastcall Boolean GetEchoActionFieldValue();
```

Description

EchoActionFieldValue (see page 23944) calls ImplEchoActionFieldValue.

See Also

EchoActionFieldValue (see page 23944)

173.1.28.2.11 TAdapterNamedField.GetInputName Method

Identifies the name to use when generating an HTML input for this adapter field.

Pascal

```
function GetInputName: string;
```

C++

```
__fastcall AnsiString GetInputName();
```

Description

This functions returns the name used for an HTML input associated with this adapter field. This value is accessed with server-side script using the InputName property. GetInputName calls ImplGetInputName (see page 23938).

See Also

ImplGetInputName (see page 23938)

173.1.28.2.12 TAdapterNamedField.GetMaxLength Method

Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.

Pascal

```
function GetMaxLength: Integer;
```

C++

```
__fastcall int GetMaxLength();
```

Description

The value returned by this function can be accessed with server-side script using the MaxLength property. Server-side script may use this value to define HTML elements that limit the number of characters that can be entered by the end-user. GetMaxLength calls ImplGetMaxLength (see page 23938).

See Also

ImplGetMaxLength (see page 23938)

173.1.28.2.13 TAdapterNamedField.GetRequired Method

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
function GetRequired: Boolean;
```

C++

```
__fastcall Boolean GetRequired();
```

Description

This function returns true or false to indicate whether a value must be entered by the end user before submitting a form. The value returned by this function can be accessed with server-side script using the Required property. Typically, this function is called while executing an adapter action that validates field values. The adapter action component will raise an exception if the value of the submitted field is empty. GetRequired calls ImplGetRequired (see page 23939).

See Also

ImplGetRequired (see page 23939)

173.1.28.2.14 TAdapterNamedField.GetValue Method

Gets the value of this adapter field as a Variant.

Pascal

```
function GetValue: Variant;
```

C++

```
__fastcall Variant GetValue();
```

Description

GetValue returns the result of ImplGetValue (see page 23939).

See Also

Value (see page 23945)

ImplGetValue (see page 23939)

173.1.28.2.15 TAdapterNamedField.GetVariableName Method

Gets the name used to reference this adapter field with server-side script.

Pascal

```
function GetVariableName: string;
```

C++

```
__fastcall AnsiString GetVariableName();
```

Description

GetVariableName is called to retrieve the name used to access this adapter field with server-side script. GetVariableName calls ImplGetVariableName (see page 23939).

See Also

ImplGetVariableName (see page 23939)

173.1.28.2.16 TAdapterNamedField.GetVisible Method

Indicates whether the value of this adapter field should be displayed to the end-user.

Pascal

```
function GetVisible: Boolean;
```

C++

```
__fastcall Boolean GetVisible();
```

Description

The value returned by this function can be accessed with server-side script using the Visible property. GetVisible calls ImplGetVisible (see page 23940).

See Also

ImplGetVisible (see page 23940)

173.1.28.2.17 TAdapterNamedField.GetWebSnapObjClass Method

Returns the class type for this adapter field.

Pascal

```
function GetWebSnapObjClass: TObjectScripterClass; virtual;
```

C++

```
virtual __fastcall TObjectScripterClass GetWebSnapObjClass();
```

Description

This function returns the class type for this adapter field, which is used by the script engine to create a script object. GetWebSnapObjClass tells the script engine what type of object to create.

173.1.28.2.18 TAdapterNamedField.HasModifyAccess Method

Checks the end-user's right to modify this adapter field.

Pascal

```
function HasModifyAccess: Boolean;
```

C++

```
__fastcall Boolean HasModifyAccess();
```

Description

The value returned by this function can be accessed with server-side script using the CanModify property. HasModifyAccess calls ImplHasModifyAccess (see page 23940).

See Also

ImplHasModifyAccess (see page 23940)

173.1.28.2.19 TAdapterNamedField.HasViewAccess Method

Checks the end-user's right to view this adapter field.

Pascal

```
function HasViewAccess: Boolean;
```

C++

```
__fastcall Boolean HasViewAccess();
```

Description

The value returned by this function can be accessed with server-side script using the CanView property. HasViewAccess calls ImplHasViewAccess (see page 23940).

See Also

ImplHasViewAccess (see page 23940)

173.1.28.2.20 TAdapterNamedField.ImplAsFormatted Method

Returns text used to generate a representation of the adapter field value.

Pascal

```
function ImplAsFormatted: string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplAsFormatted();
```

Description

ImplAsFormatted is called by AsFormatted (see page 23931).

See Also

AsFormatted (see page 23931)

173.1.28.2.21 TAdapterNamedField.ImplGetDisplayLabel Method

Returns the suggested display label for this adapter field.

Pascal

```
function ImplGetDisplayLabel: string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplGetDisplayLabel();
```

Description

ImplGetDisplayLabel is called by GetDisplayLabel (see page 23933). ImplGetDisplayLabel returns the result of GetDefaultDisplayLabel (see page 23932). Descendants may override this function.

See Also

GetDisplayLabel (see page 23933)

GetDefaultDisplayLabel (see page 23932)

173.1.28.2.22 TAdapterNamedField.ImplGetDisplayWidth Method

Returns the suggested display width for this adapter field.

Pascal

```
function ImplGetDisplayWidth: Integer; virtual;
```

C++

```
virtual __fastcall int ImplGetDisplayWidth();
```

Description

This function is called by GetDisplayWidth (see page 23933) to retrieve the suggested display width for this adapter field. ImplGetDisplayWidth returns –1 to indicate an undefined display width. Descendants may override this function.

See Also

GetDisplayWidth (see page 23933)

173.1.28.2.23 TAdapterNamedField.ImplGetEchoActionFieldValue Method

Returns the value of the echo flag.

Pascal

```
function ImplGetEchoActionFieldValue: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetEchoActionFieldValue();
```

Description

This function is called by GetEchoActionFieldValue (see page 23933).

See Also

EchoActionFieldValue (see page 23944)

GetEchoActionFieldValue (see page 23933)

173.1.28.2.24 TAdapterNamedField.ImplGetInputName Method

Identifies the name to use when generating the HTML input for this adapter field.

Pascal

```
function ImplGetInputName: string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplGetInputName();
```

Description

When an action request is received, the name/value pairs in the HTTP are searched for adapter field values. For each adapter field value found, an IActionFieldValue (see page 24596) implementation is created. In order for the HTTP request to be interpreted properly, the HTML input fields must have names that correspond to the adapter fields they represent.

See Also

GetInputName (see page 23934)

IActionFieldValue (see page 920)

173.1.28.2.25 TAdapterNamedField.ImplGetMaxLength Method

Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.

Pascal

```
function ImplGetMaxLength: Integer; virtual;
```

C++

```
virtual __fastcall int ImplGetMaxLength();
```

Description

ImplGetMaxLength is called by GetMaxLength (see page 23934). ImplGetMaxLength returns –1 to indicate an undefined maximum length. Descendants may override this function.

See Also

GetMaxLength (see page 23934)

173.1.28.2.26 TAdapterNamedField.ImplGetRequired Method

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
function ImplGetRequired: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetRequired();
```

Description

ImplGetRequired returns False. Descendants may override this function.

See Also

GetRequired (see page 23934)

173.1.28.2.27 TAdapterNamedField.ImplGetValue Method

Gets the value of this adapter field as a Variant.

Pascal

```
function ImplGetValue: Variant; virtual;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

This function is called by GetValue (see page 23935). ImplGetValue returns Unassigned. Descendants override this function.

See Also

Value (see page 23945)

GetValue (see page 23935)

173.1.28.2.28 TAdapterNamedField.ImplGetVariableName Method

Gets the name used to reference this adapter field with server-side script.

Pascal

```
function ImplGetVariableName: string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplGetVariableName();
```

Description

This function is called by GetVariableName (see page 23935). GetVariableName (see page 23935) returns the field name set by ImplSetFieldName (see page 23941) unless this name is blank. If the field name is blank, the result of GetDefaultFieldName (see page 23932) is returned.

See Also

GetVariableName (see page 23935)

ImplSetFieldName (see page 23941)

GetDefaultFieldName (see page 23932)

173.1.28.2.29 TAdapterNamedField.ImplGetVisible Method

Indicates whether the value of this adapter field should be displayed to the end-user.

Pascal

```
function ImplGetVisible: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetVisible();
```

Description

This function returns true to indicate that the field should be displayed to the end user. Descendants may override this method.

See Also

GetVisible (see page 23935)

173.1.28.2.30 TAdapterNamedField.ImplHasModifyAccess Method

Checks the end-user's right to modify this adapter field.

Pascal

```
function ImplHasModifyAccess: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasModifyAccess();
```

Description

This function is called by HasModifyAccess (see page 23936) to check the end-user's right to modify the adapter field. If the ModifyAccess (see page 23944) property is not blank, this function calls the EndUserAdapter HasModifyRights function, passing the ModifyAccess (see page 23944) property value.

See Also

hasmodifyaccess (see page 24162)

ModifyAccess (see page 23944)

HasModifyAccess (see page 23936)

173.1.28.2.31 TAdapterNamedField.ImplHasViewAccess Method

Checks the end-user's right to view this adapter field.

Pascal

```
function ImplHasViewAccess: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasViewAccess();
```

Description

This function is called by HasViewAccess (see page 23936) to check the end-user's right to view the adapter field. If the ViewAccess (see page 23945) property is not blank, this function calls the EndUserAdapter's HasViewRights function, passing the ViewAccess (see page 23945) property value.

See Also

HasViewAccess (see page 24162)

ViewAccess (see page 23945)

HasViewAccess (see page 23936)

173.1.28.2.32 TAdapterNamedField.ImplSetEchoActionFieldValue Method

Sets the echo flag for this adapter field.

Pascal

```
procedure ImplSetEchoActionFieldValue(AValue: Boolean); virtual;
```

C++

```
virtual __fastcall ImplSetEchoActionFieldValue(Boolean AValue);
```

Description

ImplEchoActionFieldValue is called by SetEchoActionFieldValue (see page 23942). ImplEchoActionFieldValue is a placeholder. Descendants may override this method.

See Also

EchoActionFieldValue (see page 23944)

SetEchoActionFieldValue (see page 23942)

173.1.28.2.33 TAdapterNamedField.ImplSetFieldName Method

Sets the name that is used to reference this adapter field with server-side script.

Pascal

```
procedure ImplSetFieldName(const Value: string); virtual;
```

C++

```
virtual __fastcall ImplSetFieldName(const AnsiString Value);
```

Description

ImplSetFieldName is called by SetFieldName (see page 23942). ImplSetFieldName stores the name in an instance variable. This method may be overridden by descendants.

See Also

SetFieldName (see page 23942)

173.1.28.2.34 TAdapterNamedField.ParentChanged Method

Handles movement of the component to a different parent.

Pascal

```
procedure ParentChanged; override;
```

C++

```
virtual __fastcall ParentChanged();
```

Description

ParentChanged is called when the parent component is changed. TAdapterNamedField (see page 23925) clears its cached reference to the parent adapter.

173.1.28.2.35 TAdapterNamedField.SetEchoActionFieldValue Method

Sets the echo flag for this adapter field.

Pascal

```
procedure SetEchoActionFieldValue(AValue: Boolean);
```

C++

```
__fastcall SetEchoActionFieldValue(Boolean AValue);
```

Description

SetEchoActionFieldValue calls ImplSetActionFieldValue.

See Also

EchoActionFieldValue (see page 23944)

ImplSetEchoActionFieldValue (see page 23941)

173.1.28.2.36 TAdapterNamedField.SetFieldName Method

Sets the name that is used to reference this adapter field with server-side script.

Pascal

```
procedure SetFieldName(const AName: string);
```

C++

```
__fastcall SetFieldName(const AnsiString AName);
```

Description

SetFieldName is typically called at design time to set the field name of a new adapter field. SetFieldName calls ImplSetFieldName (see page 23941).

See Also

ImplSetFieldName (see page 23941)

173.1.28.3 TAdapterNamedField Properties

The properties of the TAdapterNamedField class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.

173.1.28.3.1 TAdapterNamedField.ActionOrigValue Property

Retrieve the original value of this adapter field.

Pascal

```
property ActionOrigValue: IActionFieldValue;
```

C++

```
__property IActionFieldValue ActionOrigValue;
```

Description

Some adapter fields will include their original value as a hidden field in an HTML form. The original value can be compared with the current value to verify that the current value hasn't been changed by another user between the time that the HTML form was generated and the HTML form was submitted. Use this property when executing an adapter action to retrieve the original value for this adapter field at the time that the HTML form was generated.

See Also

FieldFlags (see page 7752)

173.1.28.3.2 TAdapterNamedField.ActionValue Property

Retrieve the value for this adapter field entered by the end-user.

Pascal

```
property ActionValue: IActionFieldValue;
```

C++

```
__property IActionFieldValue ActionValue;
```

Description

The ActionValue property retrieves the value(s) entered by the end-user and passed in the HTTP request when an HTML form is submitted. When executing an adapter action, use this property to retrieve the value(s) entered by the end-user.

173.1.28.3.3 TAdapterNamedField.Adapter Property

Returns the parent adapter component.

Pascal

```
property Adapter: TCustomAdapter;
```

C++

```
__property TCustomAdapter Adapter;
```

Description

Use this property to access the parent adapter component. The type of this class is TComponent because adapter components are not required to descend from a common adapter base class. Typically the adapter component is accessed through interfaces.

173.1.28.3.4 TAdapterNamedField.EchoActionFieldValue Property

Used to echo the value(s) entered by the end-user when generating an HTML form.

Pascal

```
property EchoActionFieldValue: Boolean;
```

C++

```
__property Boolean EchoActionFieldValue;
```

Description

The EchoActionFieldValue property is used to alter the source of the adapter fields value. When this flag is set, the adapter field retrieves it's value from the HTTP request. When this flag is clear, the adapter field retrieves it's current value. Typically, this flag is set after an error occurs while processing an adapter action. Setting this flag results in generated HTML that displays values entered by the end-user. This allows the end-user to make corrections and resubmit the form.

See Also

SetEchoActionFieldValue (see page 23942)

173.1.28.3.5 TAdapterNamedField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

173.1.28.3.6 TAdapterNamedField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property AnsiString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

hasmodifyaccess (see page 24162)

173.1.28.3.7 TAdapterNamedField.Value Property

Provides the value of an adapter field as a variant.

Pascal

```
property Value: Variant;
```

C++

```
__property Variant Value;
```

Description

The Value property returns the value of an adapter field as a Variant. Server-side script can access this value using the Value property. Note that descendant adapter fields support multiple values per field, files and images. For these fields, the Value property will not correctly represent the value of the adapter field.

See Also

GetValueCount (see page 24017)

GetValues (see page 24017)

173.1.28.3.8 TAdapterNamedField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property AnsiString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 25625)

HasViewAccess (see page 24162)

173.1.29 TAdapterNextPageAction Class

TAdapterNextPageAction implements the next page command for TDefaultFieldsPagedAdapter (see page 24403).

Class Hierarchy

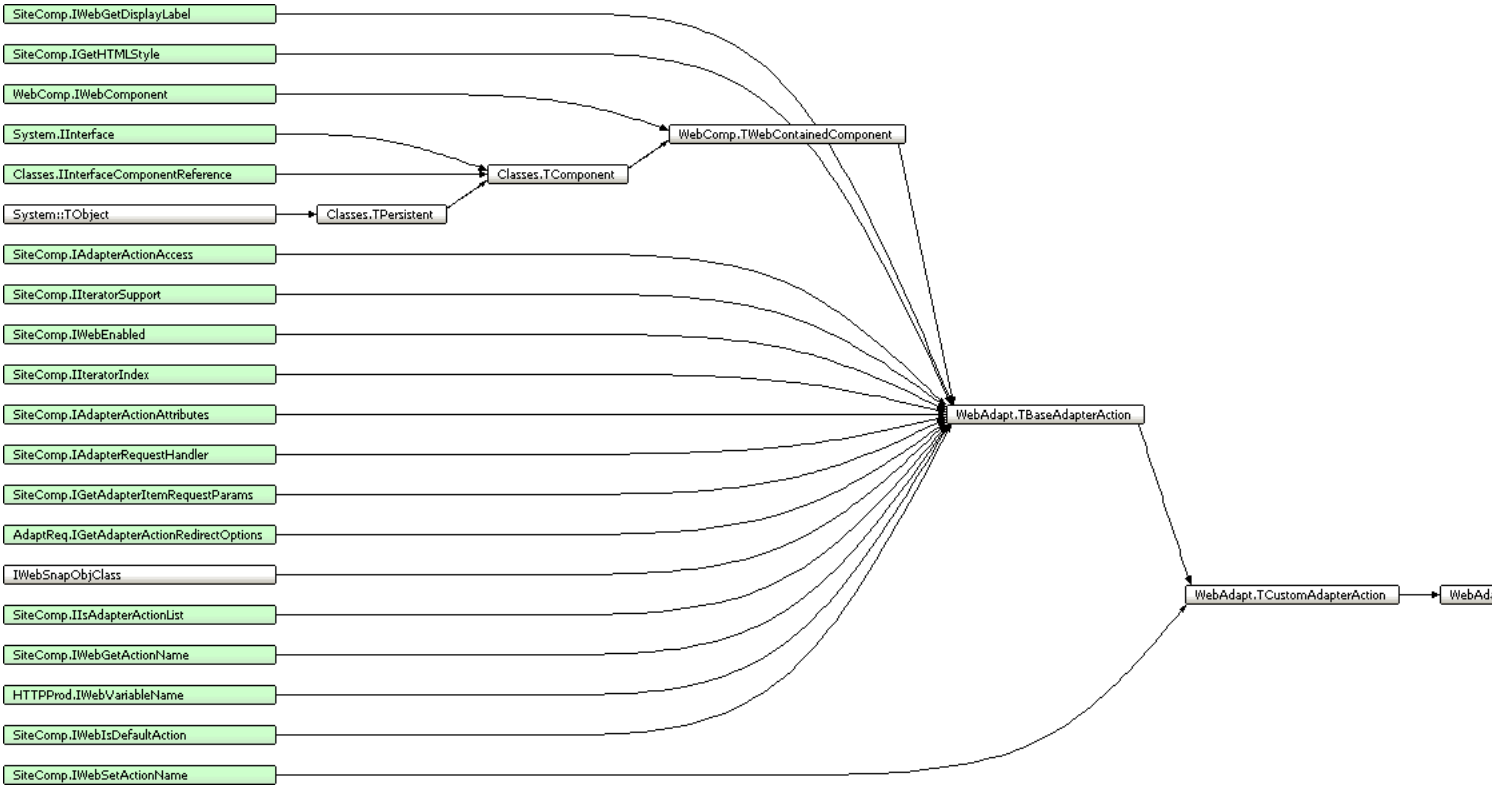

Pascal

```
TAdapterNextPageAction = class(TCustomAdapterPageAction);
```

C++

```
class TAdapterNextPageAction : public TCustomAdapterPageAction;
```

File

WebAdapt

Description

Add TAdapterNextPageAction to a descendant class of TDefaultFieldsPagedAdapter (see page 24403) such as TPagedAdapter (see page 24525).

See Also

TCustomAdapterPageAction (see page 24286)

TDefaultFieldsPagedAdapter (see page 24403)

TAdapterPrevPageAction (see page 23979)

TAdapterGotoPageAction (see page 23849)

TPagedAdapter (see page 24525)

TDefaultFieldsPagedAdapter (see page 24403)

173.1.29.1 TAdapterNextPageAction Members

The following tables list the members exposed by TAdapterNextPageAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 24377)	Returns the parent adapter of this adapter action.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 23952)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 23953)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
V	GetDisplayStyleType (see page 24293)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
V	GetNewPage (see page 23953)	Indicates the page to display when this action is executed.

	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 24063)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 24195)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 24294)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24294)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplIsDefaultAction (see page 24295)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 24069)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 24197)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 24295)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.

V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Adapter (see page 24378)	Returns the parent adapter of this adapter action.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.

R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 23955)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 23955)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 23956)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 23956)	Called before the adapter action response is generated.
	OnGetParams (see page 23957)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 23954)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 23954)	Defines the rights required to execute this action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RedirectOptions (see page 23954)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.29.2 TAdapterNextPageAction Methods

The methods of the TAdapterNextPageAction class are listed here.

Protected Methods

	Name	Description
V	GetDefaultActionName (see page 23952)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 23953)	Provides the default display label for this adapter action.
V	GetNewPage (see page 23953)	Indicates the page to display when this action is executed.

173.1.29.2.1 TAdapterNextPageAction.GetDefaultActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultActionName();
```

Description

GetDefaultActionName returns "NextPage".

See Also

GetActionName (see page 24054)

GetDefaultActionName (see page 24055)

173.1.29.2.2 TAdapterNextPageAction.GetDefaultDisplayLabel Method

Provides the default display label for this adapter action.

Pascal

```
function GetDefaultDisplayLabel: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultDisplayLabel();
```

Description

GetDefaultDisplayLabel returns ">>". This displays as '>>' in the browser.

See Also

DisplayLabel (see page 24073)

173.1.29.2.3 TAdapterNextPageAction.GetNewPage Method

Indicates the page to display when this action is executed.

Pascal

```
function GetNewPage(var AValue: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean GetNewPage(int AValue);
```

Description

GetNewPage determines whether the current page can be incremented. If so, the AValue parameter is modified and true is returned. Otherwise, false is returned.

See Also

GetNewPage (see page 24294)

RecordCount (see page 24419)

PageSize (see page 24419)

CurrentPage (see page 24418)

173.1.29.3 TAdapterNextPageAction Properties

The properties of the TAdapterNextPageAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 23954)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 23954)	Defines the rights required to execute this action.
	RedirectOptions (see page 23954)	Indicates whether to perform an HTTP redirect after successfully executing this action.

173.1.29.3.1 TAdapterNextPageAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

173.1.29.3.2 TAdapterNextPageAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property AnsiString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 25625)

hasexecuteaccess (see page 24162)

173.1.29.3.3 TAdapterNextPageAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect (see page 15201)

173.1.29.4 TAdapterNextPageAction Events

The events of the TAdapterNextPageAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 23955)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 23955)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 23956)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 23956)	Called before the adapter action response is generated.
	OnGetParams (see page 23957)	Specifies name/value pairs needed to execute this adapter action.

173.1.29.4.1 TAdapterNextPageAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

173.1.29.4.2 TAdapterNextPageAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.29.4.3 TAdapterNextPageAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

173.1.29.4.4 TAdapterNextPageAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.29.4.5 TAdapterNextPageAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

173.1.30 TAdapterNextPageField Class

The TAdapterNextPageField navigates to the next page of data.

Class Hierarchy

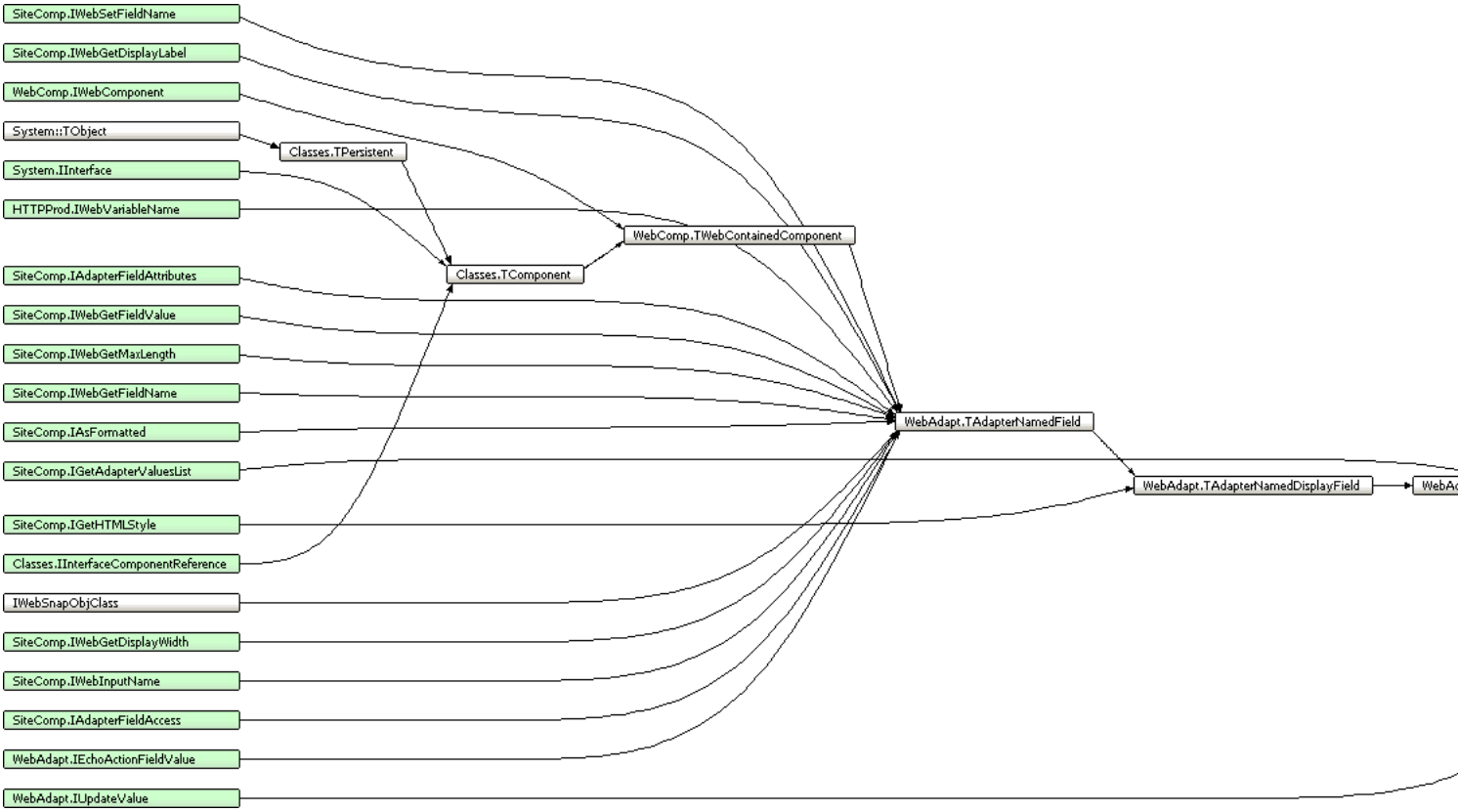

Pascal

```
TAdapterNextPageField = class(TAdapterLoginField, IGetAdapterValuesList, IUpdateValue);
```

C++

```
class TAdapterNextPageField : public TAdapterLoginField, public IGetAdapterValuesList, public
IUpdateValue;
```

File

WebAdapt

Description

TAdapterNextPageField is one of the components that is used by TPagedAdapter (see page 24525) along with the TAdapterNextPageAction (see page 23945) to navigate to the next page of data.

The TPagedAdapter (see page 24525) PageSize property indicates the number of rows displayed on a page. When the PageSize is not 0, the TPagedAdapter (see page 24525) component automatically adds built-in actions for moving between pages. These actions are TAdapterGotoPageAction (see page 23849), TAdapterPrevPageAction (see page 23979), and TAdapterNextPageAction (see page 23945). The TAdapterPageProducer can be used to generate HTML hyperlinks or buttons for these actions.

See Also

TPagedAdapter (see page 24525)

TAdapterPrevPageAction (see page 23979)

TAdapterGotoPageAction (see page 23849)

TCustomAdapterNextPageAction

TCustomAdapterPageAction (see page 24286)

173.1.30.1 TAdapterNextPageField Members

The following tables list the members exposed by TAdapterNextPageField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 23964)	Returns a values list adapter containing the names of the published pages.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 23964)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetEchoActionFieldValue (see page 23965)	Returns the echo flag for this adapter field.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 23965)	Returns the page name to be displayed after login.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23965)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
V	UpdateValue (see page 23966)	Updates the parent adapter's next page property with the value entered by the end-user.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 23963)	Frees an instance of TAdapterNextPageField (see page 23957).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	Adapter (see page 23888)	Returns the parent adapter of this adapter field.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 23924)	Indicates whether this adapter field is read-only.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	DisplayLabel (see page 23966)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23967)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 23967)	Identifies the name of this field when using server-side script.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.30.2 TAdapterNextPageField.Destroy Destructor

Frees an instance of TAdapterNextPageField (see page 23957).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterNextPageField();
```

Description

Destroy removes this component from its parent's list of components that generate HTML before freeing the memory used by TAdapterNextPageField (see page 23957).

173.1.30.3 TAdapterNextPageField Methods

The methods of the TAdapterNextPageField class are listed here.

Protected Methods

	Name	Description
	GetAdapterValuesList (see page 23964)	Returns a values list adapter containing the names of the published pages.
V	GetInputStyleType (see page 23964)	Suggests an HTML input type for this adapter field.
V	ImplGetEchoActionFieldValue (see page 23965)	Returns the echo flag for this adapter field.
V	ImplGetValue (see page 23965)	Returns the page name to be displayed after login.
V	ImplSetEchoActionFieldValue (see page 23965)	Sets the echo flag for this adapter field.
V	UpdateValue (see page 23966)	Updates the parent adapter's next page property with the value entered by the end-user.

173.1.30.3.1 TAdapterNextPageField.GetAdapterValuesList Method

Returns a values list adapter containing the names of the published pages.

Pascal

```
function GetAdapterValuesList: IValuesListAdapter;
```

C++

```
__fastcall IValuesListAdapter GetAdapterValuesList();
```

Description

GetAdapterValuesList returns a reference to TLogicalPageList (see page 24492). This function will be called when server-side script references the ValuesList property of this adapter field. Typically, server-side script uses the ValuesList property to populate an HTML select list with the names of published pages. The end-user may select one of these pages to go to after login.

See Also

IValuesListAdapter (see page 19740)

173.1.30.3.2 TAdapterNextPageField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const AnsiString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType. GetInputStyleType returns htmliSelect indicating that a single selection list HTML element should be used to input a value for this adapter field.

See Also

GetInputStyleType (see page 23921)

TAdapterInputHTMLElementType

173.1.30.3.3 TAdapterNextPageField.ImplGetEchoActionFieldValue Method

Returns the echo flag for this adapter field.

Pascal

```
function ImplGetEchoActionFieldValue: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplGetEchoActionFieldValue();
```

Description

ImplGetEchoActionFieldValue returns the value of an instance variable.

See Also

ImplGetEchoActionFieldValue (see page 23938)

EchoActionFieldValue (see page 23944)

ImplGetValue (see page 23965)

ImplSetEchoActionFieldValue (see page 23965)

173.1.30.3.4 TAdapterNextPageField.ImplGetValue Method

Returns the page name to be displayed after login.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

If the EchoActionFieldValue (see page 23944) property is true, then the page name selected or entered by end-user is returned. If the EchoActionFieldValue (see page 23944) property is false then the parent adapter's NextPage property is returned. The value returned by ImplGetValue is used when server-side script references the EditText property of this adapter field while generating the login form.

See Also

EchoActionFieldValue (see page 23944)

ActionValue (see page 23943)

NextPage (see page 24359)

173.1.30.3.5 TAdapterNextPageField.ImplSetEchoActionFieldValue Method

Sets the echo flag for this adapter field.

Pascal

```
procedure ImplSetEchoActionFieldValue(AValue: Boolean); override;
```

C++

```
virtual __fastcall ImplSetEchoActionFieldValue(Boolean AValue);
```

Description

ImplSetEchoActionFieldValue sets an internal variable to true. The echo flag is typically set by an adapter action when an error occurs while processing an HTTP request. The purpose of the echo flag is to preserve the value entered by the end-user. When the echo flag is true, ImplGetValue (see page 23965) returns the value entered by the end-user rather than the value of the adapter field.

See Also

ImplSetEchoActionFieldValue (see page 23941)

EchoActionFieldValue (see page 23944)

ImplGetValue (see page 23965)

ImplGetEchoActionFieldValue (see page 23965)

173.1.30.3.6 TAdapterNextPageField.UpdateValue Method

Updates the parent adapter's next page property with the value entered by the end-user.

Pascal

```
procedure UpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); virtual;
```

C++

```
virtual __fastcall UpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

Typically a single selection list HTML element is displayed on the login form. The list contains all of the published pages. The HTTP request will contain the page name selected by the end-user.

See Also

NextPage (see page 24359)

IActionRequest (see page 935)

173.1.30.4 TAdapterNextPageField Properties

The properties of the TAdapterNextPageField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 23966)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23967)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 23967)	Identifies the name of this field when using server-side script.

173.1.30.4.1 TAdapterNextPageField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 23933)

173.1.30.4.2 TAdapterNextPageField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 23933)

173.1.30.4.3 TAdapterNextPageField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

173.1.31 TAdapterObjectErrorsList Class

Class Hierarchy

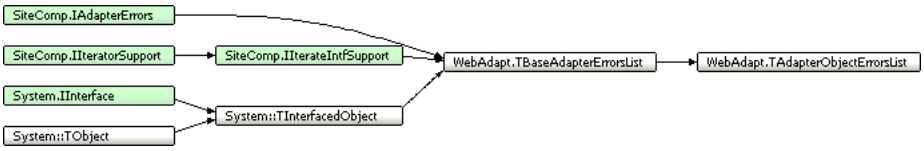

Pascal

```
TAdapterObjectErrorsList = class(TBaseAdapterErrorsList);
```

C++

```
class TAdapterObjectErrorsList : public TBaseAdapterErrorsList;
```

File

WebAdapt

Description

This is class WebAdapt.TAdapterObjectErrorsList.

173.1.31.1 TAdapterObjectErrorsList Members

The following tables list the members exposed by TAdapterObjectErrorsList.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	EndIterator (see page 24081)	This is EndIterator, a member of class TBaseAdapterErrorsList.
	GetCurrentIntf (see page 24081)	This is GetCurrentIntf, a member of class TBaseAdapterErrorsList.
	GetError (see page 24082)	Returns a particular error.
	GetErrorCount (see page 24082)	Return the number of errors in the errors list.
	GetObjectErrors (see page 24082)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 24082)	Indicates whether the errors list contains objects associated with a particular adapter field.
V	ImplClearErrors (see page 24083)	This is ImplClearErrors, a member of class TBaseAdapterErrorsList.
V	ImplDefineLabel (see page 23970)	This is ImplDefineLabel, a member of class TAdapterObjectErrorsList.
V	ImplGetError (see page 23970)	This is ImplGetError, a member of class TAdapterObjectErrorsList.
V	ImplGetErrorCount (see page 23971)	This is ImplGetErrorCount, a member of class TAdapterObjectErrorsList.
V	ImplGetObjectErrors (see page 24084)	This is ImplGetObjectErrors, a member of class TBaseAdapterErrorsList.
V	ImplHasObjectErrors (see page 24084)	This is ImplHasObjectErrors, a member of class TBaseAdapterErrorsList.
	NextIteration (see page 24084)	Advances to the objects next item.

	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	StartIterator (see page 24085)	This is StartIterator, a member of class TBaseAdapterErrorsList.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AddError (see page 24080)	This is the overview for the AddError method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearErrors (see page 19664)	Clears the errors list.
	ClearErrors (see page 24080)	Clear error list.
	Create (see page 23970)	This is Create, a member of class TAdapterObjectErrorsList.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DefineLabel (see page 19664)	Associates a label with all errors associated with a particular adapter field.
	DefineLabel (see page 24081)	Define error label for field.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetCurrentIntf (see page 19718)	This is GetCurrentIntf, a member of class IIterateIntfSupport.
	GetError (see page 19664)	Returns a particular error.
	GetErrorCount (see page 19665)	Return the number of errors in the errors list.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetObjectErrors (see page 19665)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 19665)	Indicates whether the errors list contains objects associated with a particular adapter field.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.

Public Properties

	Name	Description
	ErrorCount (see page 19666)	Returns the number of errors in the errors list.
	ErrorCount (see page 24085)	Number of error objects contained.
	Errors (see page 19666)	Returns an error from the list.
	Errors (see page 24085)	Access to error objects.
	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

173.1.31.2 TAdapterObjectErrorsList.Create Constructor

Pascal

```
constructor Create(AErrors: TAdapterErrorsList; const AName: string);
```

C++

```
__fastcall TAdapterObjectErrorsList(TAdapterErrorsList AErrors, const AnsiString AName);
```

Description

This is Create, a member of class TAdapterObjectErrorsList.

173.1.31.3 TAdapterObjectErrorsList Methods

The methods of the TAdapterObjectErrorsList class are listed here.

Protected Methods

	Name	Description
	ImplDefineLabel (see page 23970)	This is ImplDefineLabel, a member of class TAdapterObjectErrorsList.
	ImplGetError (see page 23970)	This is ImplGetError, a member of class TAdapterObjectErrorsList.
	ImplGetErrorCount (see page 23971)	This is ImplGetErrorCount, a member of class TAdapterObjectErrorsList.

173.1.31.3.1 TAdapterObjectErrorsList.ImplDefineLabel Method

Pascal

```
procedure ImplDefineLabel(const AName: string; const ALabel: string); override;
```

C++

```
virtual __fastcall ImplDefineLabel(const AnsiString AName, const AnsiString ALabel);
```

Description

This is ImplDefineLabel, a member of class TAdapterObjectErrorsList.

173.1.31.3.2 TAdapterObjectErrorsList.ImplGetError Method

Pascal

```
function ImplGetError(I: Integer): IAdapterError; override;
```

C++

```
virtual __fastcall IAdapterError ImplGetError(int I);
```

Description

This is ImplGetError, a member of class TAdapterObjectErrorsList.

173.1.31.3.3 TAdapterObjectErrorsList.ImplGetErrorCount Method

Pascal

```
function ImplGetErrorCount: Integer; override;
```

C++

```
virtual __fastcall int ImplGetErrorCount();
```

Description

This is ImplGetErrorCount, a member of class TAdapterObjectErrorsList.

173.1.32 TAdapterPasswordField Class

TAdapterPasswordField is an adapter field used to access the password entered by the end-user.

Class Hierarchy

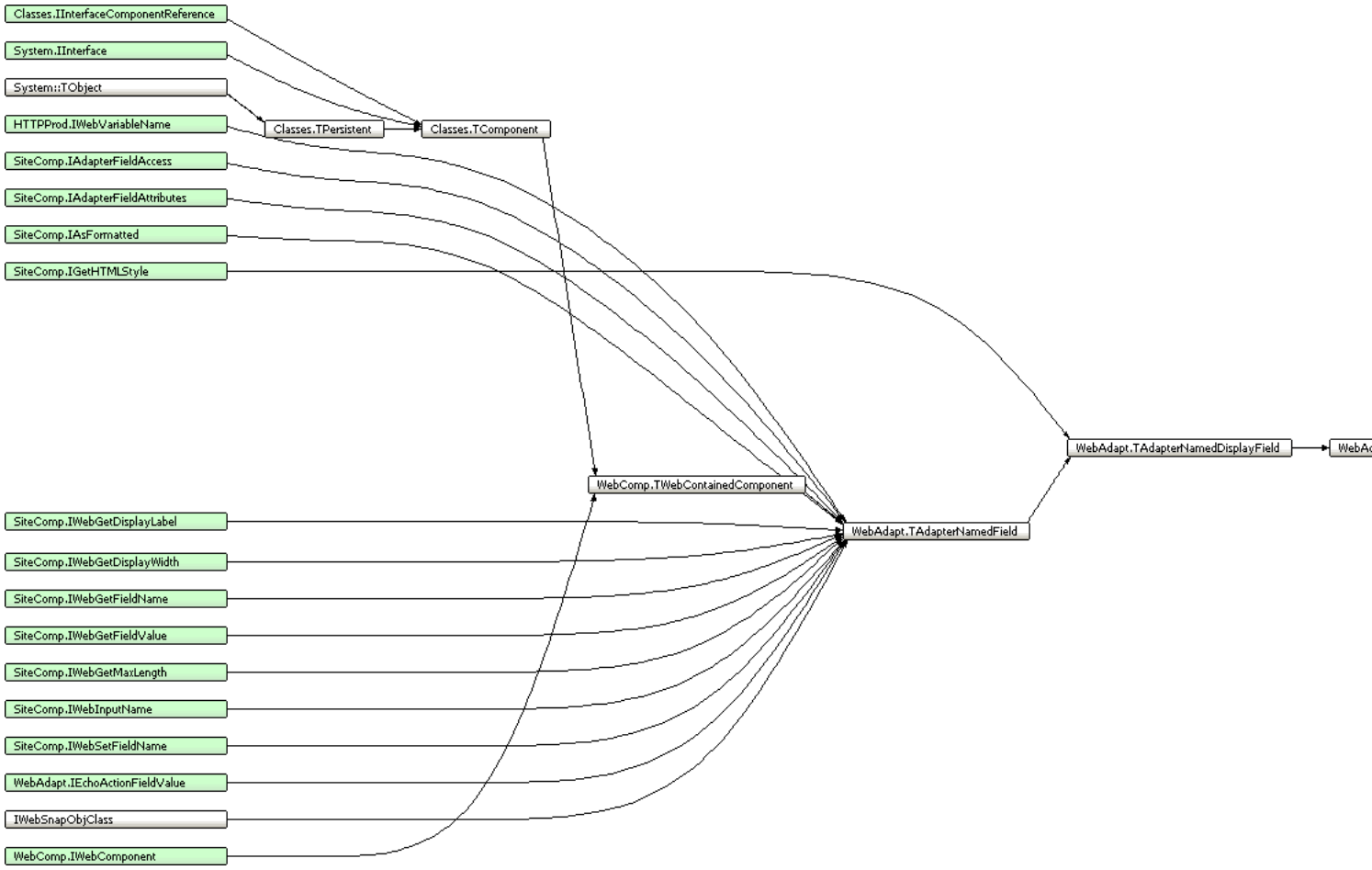

Pascal

```
TAdapterPasswordField = class(TAdapterLoginField);
```

C++

```
class TAdapterPasswordField : public TAdapterLoginField;
```

File

WebAdapt

Description

TAdapterPasswordField is added to TLoginFormAdapter (see page 24500). TAdapterUserNameField (see page 24024), TAdapterPasswordField, TLoginFormAdapterLoginAction (see page 24514) and TAdapterNextPageField (see page 23957) are used with TLoginFormAdapter (see page 24500) to generate an HTML login form. When the HTML form is submitted, TAdapterPasswordField is used to extract the end user's password from the HTTP request.

See Also

TAdapterLoginField (see page 23882)

TLoginFormAdapter (see page 24500)

TAdapterUserNameField (see page 24024)

TAdapterNextPageField (see page 23957)

TLoginFormAdapterLoginAction (see page 24514)

173.1.32.1 TAdapterPasswordField Members

The following tables list the members exposed by TAdapterPasswordField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 23978)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.

	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 23978)	Returns the value associated with this adapter field.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Adapter (see page 23943)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	Adapter (see page 23888)	Returns the parent adapter of this adapter field.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 23924)	Indicates whether this adapter field is read-only.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	DisplayLabel (see page 23978)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23979)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 23979)	Identifies the name of this field when using server-side script.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.32.2 TAdapterPasswordField Methods

The methods of the TAdapterPasswordField class are listed here.

Protected Methods

	Name	Description
V	GetInputStyleType (see page 23978)	Suggests an HTML input type for this adapter field.
V	ImplGetValue (see page 23978)	Returns the value associated with this adapter field.

173.1.32.2.1 TAdapterPasswordField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const AnsiString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType method. GetInputStyleType returns htmliPassword indicating that a password input HTML element should be used to input a value for this adapter field.

See Also

GetInputStyleType (see page 23921)

TAdapterInputHTMLElementType

173.1.32.2.2 TAdapterPasswordField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns an empty string. Server-side script accesses the result of this function using the EditText or Value (see page 23945) property of the adapter field. These server-side script properties may be used to generate the HTML login form. ImplGetValue returns an empty string so that the generated login form's password field will be blank. When the login form is submitted, the ActionValue (see page 23943) property can be used to access the password entered by the end-user.

See Also

ActionValue (see page 23943)

173.1.32.3 TAdapterPasswordField Properties

The properties of the TAdapterPasswordField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 23978)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23979)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 23979)	Identifies the name of this field when using server-side script.

173.1.32.3.1 TAdapterPasswordField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 23933)

173.1.32.3.2 TAdapterPasswordField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 23933)

173.1.32.3.3 TAdapterPasswordField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

173.1.33 TAdapterPrevPageAction Class

TAdapterPrevPageAction implements the previous page command for TDefaultFieldsPagedAdapter (see page 24403).

Class Hierarchy

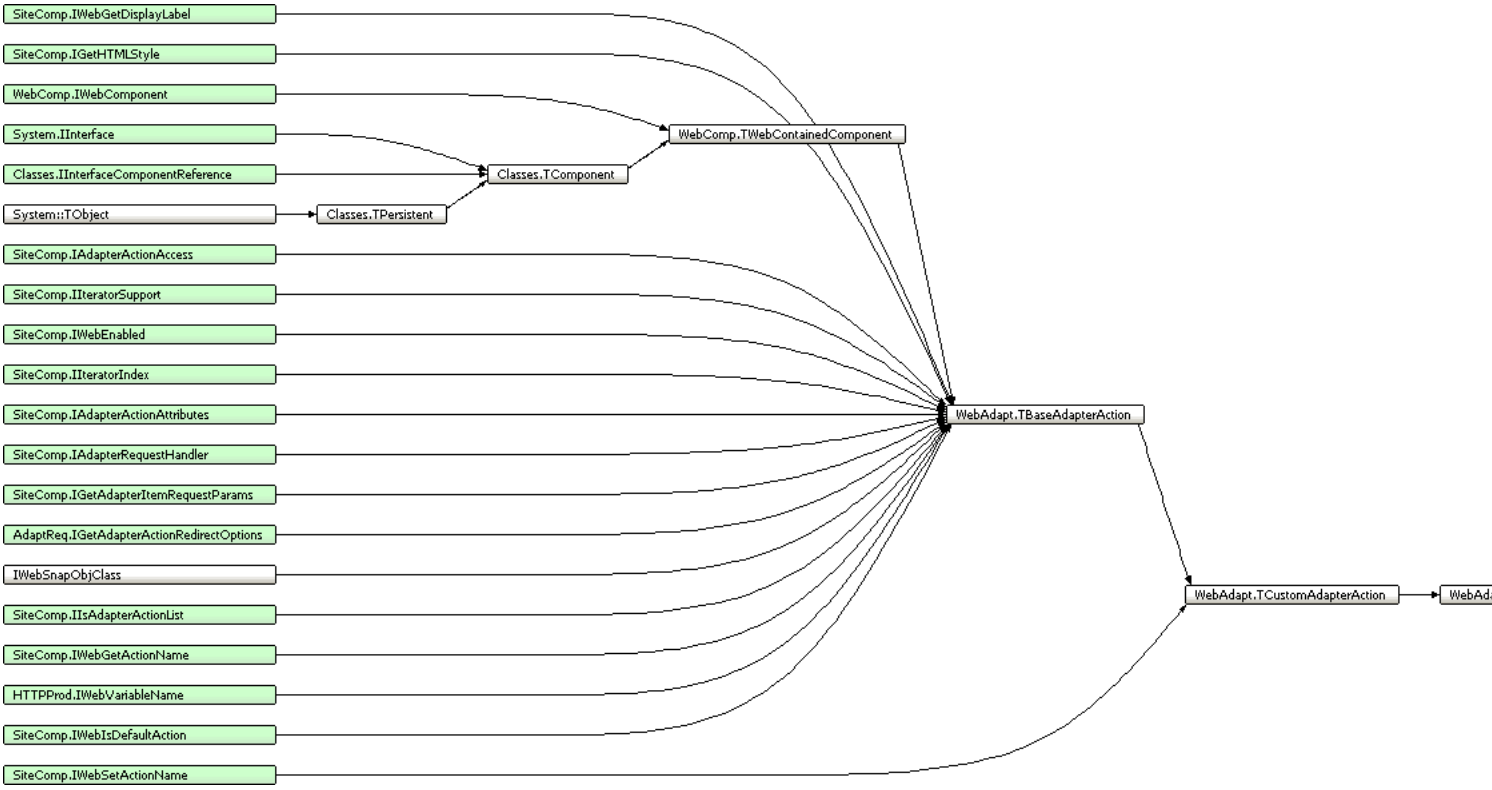

Pascal

```
TAdapterPrevPageAction = class(TCustomAdapterPageAction);
```

C++

```
class TAdapterPrevPageAction : public TCustomAdapterPageAction;
```

File

WebAdapt

Description

Add TAdapterPrevPageAction to a descendant class of TDefaultFieldsPagedAdapter (see page 24403) such as TPagedAdapter (see page 24525).

See Also

TCustomAdapterPageAction (see page 24286)

TDefaultFieldsPagedAdapter (see page 24403)

TAdapterNextPageAction (see page 23945)

TAdapterGotoPageAction (see page 23849)

TPagedAdapter (see page 24525)

TDefaultFieldsPagedAdapter (see page 24403)

173.1.33.1 TAdapterPrevPageAction Members

The following tables list the members exposed by TAdapterPrevPageAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 24377)	Returns the parent adapter of this adapter action.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 23986)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 23987)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
V	GetDisplayStyleType (see page 24293)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
V	GetNewPage (see page 23987)	Indicates the page to display when this action is executed.

	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 24063)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 24195)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 24294)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24294)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplIsDefaultAction (see page 24295)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 24069)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 24197)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 24295)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.

V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Adapter (see page 24378)	Returns the parent adapter of this adapter action.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.

R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 23989)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 23989)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 23990)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 23990)	Called before the adapter action response is generated.
	OnGetParams (see page 23991)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 23988)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 23988)	Defines the rights required to execute this action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RedirectOptions (see page 23988)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.33.2 TAdapterPrevPageAction Methods

The methods of the TAdapterPrevPageAction class are listed here.

Protected Methods

	Name	Description
V	GetDefaultActionName (see page 23986)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 23987)	Provides the default display label for this adapter action.
V	GetNewPage (see page 23987)	Indicates the page to display when this action is executed.

173.1.33.2.1 TAdapterPrevPageAction.GetDefaultActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultActionName();
```

Description

GetDefaultActionName returns "PrevPage".

See Also

GetActionName (see page 24054)

GetDefaultActionName (see page 24055)

173.1.33.2.2 TAdapterPrevPageAction.GetDefaultDisplayLabel Method

Provides the default display label for this adapter action.

Pascal

```
function GetDefaultDisplayLabel: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultDisplayLabel();
```

Description

GetDefaultDisplayLabel returns "<<". This displays as '<<' in the browser.

See Also

DisplayLabel (see page 24073)

173.1.33.2.3 TAdapterPrevPageAction.GetNewPage Method

Indicates the page to display when this action is executed.

Pascal

```
function GetNewPage(var AValue: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean GetNewPage(int AValue);
```

Description

GetNewPage determines whether the current page can be incremented. If so, the AValue parameter is modified and true is returned. Otherwise, false is returned.

See Also

GetNewPage (see page 24294)

RecordCount (see page 24419)

PageSize (see page 24419)

CurrentPage (see page 24418)

173.1.33.3 TAdapterPrevPageAction Properties

The properties of the TAdapterPrevPageAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 23988)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 23988)	Defines the rights required to execute this action.
	RedirectOptions (see page 23988)	Indicates whether to perform an HTTP redirect after successfully executing this action.

173.1.33.3.1 TAdapterPrevPageAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

173.1.33.3.2 TAdapterPrevPageAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property AnsiString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 25625)

hasexecuteaccess (see page 24162)

173.1.33.3.3 TAdapterPrevPageAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect (see page 15201)

173.1.33.4 TAdapterPrevPageAction Events

The events of the TAdapterPrevPageAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 23989)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 23989)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 23990)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 23990)	Called before the adapter action response is generated.
	OnGetParams (see page 23991)	Specifies name/value pairs needed to execute this adapter action.

173.1.33.4.1 TAdapterPrevPageAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

173.1.33.4.2 TAdapterPrevPageAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.33.4.3 TAdapterPrevPageAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

173.1.33.4.4 TAdapterPrevPageAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.33.4.5 TAdapterPrevPageAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

173.1.34 TAdapterSubComponent Class

TAdapterSubComponent is the base class for objects that represent a collection of actions or fields that belong to an adapter.

Class Hierarchy

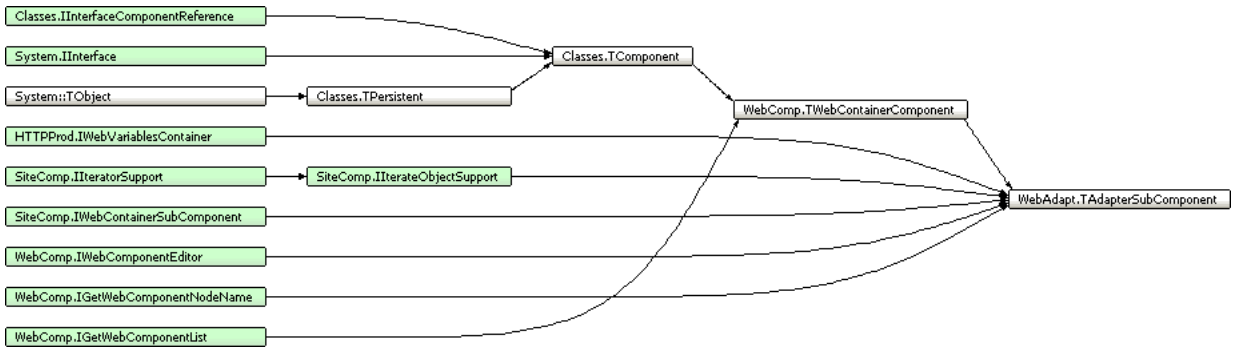

Pascal

```
TAdapterSubComponent = class(TWebContainerComponent, IWebContainerSubComponent,
IWebComponentEditor, IWebVariablesContainer, IGetWebComponentNodeName, IIterateObjectSupport);
```

C++

```
class TAdapterSubComponent : public TWebContainerComponent, public IWebContainerSubComponent,
public IWebComponentEditor, public IWebVariablesContainer, public IGetWebComponentNodeName,
public IIterateObjectSupport;
```

File

WebAdapt

Description

Adapter (see page 24006) components use a TAdapterSubComponent descendant to represent the list of actions they can

execute or the list of fields whose values they support. Each TAdapterSubComponent is specialized for a specific type of object (actions or fields), and represents either the set of explicitly assigned actions or fields, or the set that the adapter contains by default.

TAdapterSubComponent introduces support for:

Associating the collection with an adapter component.

Assigning variable names to the child items in the collection. These are the names of variables that server-side script can use to refer to the actions or fields.

Letting server-side script use an enumerator to iterate over all the child items in the collection.

Working with the WebSnap designers at design time.

See Also

TAdapterActions (see page 23738)

TAdapterFields (see page 23830)

TAdapterDefaultActions (see page 23770)

TAdapterDefaultFields (see page 23778)

173.1.34.1 TAdapterSubComponent Members

The following tables list the members exposed by TAdapterSubComponent.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 23996)	Indicates whether an object of a specified type can be added to this collection.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EndIterator (see page 23997)	Stops the TAdapterSubComponent (see page 23991) object from iterating over the Web items it lists.
V	FindNamedVariable (see page 23997)	Returns the item from this list with a specified variable name.
	FindVariable (see page 23998)	Returns the item from this list with a specified variable name.
	FindVariableContainer (see page 23998)	Returns the Web module that contains a specified item from this list.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24755)	Executes a callback for each child Web item.
	GetCurrentObject (see page 23999)	Returns the current field or action from the list when an enumerator is iterating over child items.
V	GetDefaultWebComponents (see page 23999)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariable (see page 24000)	Returns a child item from this list, given its position in the list of child items with variable names.
	GetVariableCount (see page 24000)	Indicates the number of items in this list that can be accessed from server-side script.
V	GetVariablesInUse (see page 24001)	Returns the value of the VariablesInUse (see page 24007) property.
	GetWebComponentNodeName (see page 24001)	Returns a string that indicates the type of items in this collection.
	GetWebComponents (see page 24756)	Returns the list of Web items that this item manages.
V	ImplCanAddClass (see page 24002)	Provides the underlying implementation for the CanAddClass (see page 23996) method.
V	ImplGetDefaultComponentList (see page 24756)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetWebComponentNodeName (see page 24003)	Provides the underlying implementation for the GetWebComponentNodeName (see page 24001) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24003)	Changes the current object to the next item that has a variable name.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	NotifyAdapterChange (see page 24003)	Notifies the associated adapter when the list of items changes.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24757)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 24004)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetWebParent (see page 24005)	Introduces a method to associate this component with an adapter.
	StartIterator (see page 24006)	Makes the first child item with a variable name the current object.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24006)	Indicates the adapter that uses this object to manage its fields or actions.
R	VariablesInUse (see page 24007)	Lists those fields or actions in the collection for which variables are defined.
R	WebContainerComponentHelper (see page 24758)	Provides access to the helper object that allows this object to manage its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24754)	Creates and initializes an instance of TWebContainerComponent (see page 24751).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24755)	Frees an instance of TWebContainerComponent (see page 24751).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 24000)	Returns the associated adapter.
V	HasParent (see page 24002)	Indicates whether this component has an associated adapter.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 24004)	Reads the component's published properties, stored data, and owned components from a reader object.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	SetParentComponent (see page 24005)	Sets the Adapter (see page 24006) property to be the same as this component's parent.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.34.2 TAdapterSubComponent Methods

The methods of the TAdapterSubComponent class are listed here.

Protected Methods

	Name	Description
	CanAddClass (see page 23996)	Indicates whether an object of a specified type can be added to this collection.
	EndIterator (see page 23997)	Stops the TAdapterSubComponent (see page 23991) object from iterating over the Web items it lists.
V	FindNamedVariable (see page 23997)	Returns the item from this list with a specified variable name.
	FindVariable (see page 23998)	Returns the item from this list with a specified variable name.
	FindVariableContainer (see page 23998)	Returns the Web module that contains a specified item from this list.
	GetCurrentObject (see page 23999)	Returns the current field or action from the list when an enumerator is iterating over child items.

	GetDefaultWebComponents (see page 23999)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
	GetVariable (see page 24000)	Returns a child item from this list, given its position in the list of child items with variable names.
	GetVariableCount (see page 24000)	Indicates the number of items in this list that can be accessed from server-side script.
	GetVariablesInUse (see page 24001)	Returns the value of the VariablesInUse (see page 24007) property.
	GetWebComponentNodeName (see page 24001)	Returns a string that indicates the type of items in this collection.
	ImplCanAddClass (see page 24002)	Provides the underlying implementation for the CanAddClass (see page 23996) method.
	ImplGetWebComponentNodeName (see page 24003)	Provides the underlying implementation for the GetWebComponentNodeName (see page 24001) method.
	NextIteration (see page 24003)	Changes the current object to the next item that has a variable name.
	NotifyAdapterChange (see page 24003)	Notifies the associated adapter when the list of items changes.
	SetDefaultWebComponents (see page 24004)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetWebParent (see page 24005)	Introduces a method to associate this component with an adapter.
	StartIterator (see page 24006)	Makes the first child item with a variable name the current object.

Public Methods

	Name	Description
	GetParentComponent (see page 24000)	Returns the associated adapter.
	HasParent (see page 24002)	Indicates whether this component has an associated adapter.
	ReadState (see page 24004)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetParentComponent (see page 24005)	Sets the Adapter (see page 24006) property to be the same as this component's parent.

173.1.34.2.1 TAdapterSubComponent.CanAddClass Method

Indicates whether an object of a specified type can be added to this collection.

Pascal

```
function CanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClass(TComponent * AParent, TClass AClass);
```

Description

The IDE calls CanAddClass to determine what Web items can be added to this list. This method allows the IDE to limit the actions or fields in the collection to those that are appropriate for the associated adapter.

AParent is the TAdapterSubComponent (see page 23991) descendant to which the Web item will be added.

AClass is the class of the proposed child item.

CanAddClass returns true if the TAdapterSubComponent (see page 23991) descendant can include a child of type AClass, false otherwise.

CanAddClass calls the virtual ImplCanAddClass (see page 24002) method to determine whether a class of Web item is valid.

See Also

ImplCanAddClass (see page 24002)

Adapter (see page 24006)

173.1.34.2.2 TAdapterSubComponent.EndIterator Method

Stops the TAdapterSubComponent (see page 23991) object from iterating over the Web items it lists.

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

EndIterator is part of the support that allows server-side script to use an enumerator to iterate over all the child items in the collection. This method is called when the enumeration stops. In TAdapterSubComponent (see page 23991), EndIterator does nothing. This method exists only as a backstop for the standard call made from server-side script.

OwnerData is a pointer to object-specific information returned by the last call to the StartIterator (see page 24006) or NextIteration (see page 24003) method. TAdapterSubComponent (see page 23991) does not use the OwnerData parameter.

See Also

StartIterator (see page 24006)

NextIteration (see page 24003)

GetCurrentObject (see page 23999)

173.1.34.2.3 TAdapterSubComponent.FindNamedVariable Method

Returns the item from this list with a specified variable name.

Pascal

```
function FindNamedVariable(const AName: string): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * FindNamedVariable(const AnsiString AName);
```

Description

FindNamedVariable searches the list of child items that have variable names (the VariablesInUse (see page 24007) property) to find one with the specified name. The associated adapter component calls this method from its FindNamedField or FindNamedAction method to look up a field or action with a specified name.

AName is a variable name. Typically, this comes from server-side script that is looking for a child of an adapter with this name.

FindNamedVariable returns the item from the list with the specified variable name. If none of the items in the list have the name AName, FindNamedVariable returns nil (Delphi) or NULL (C++).

See Also

VariablesInUse (see page 24007)

FindVariable (see page 23998)

GetVariable (see page 24000)

GetVariableCount (see page 24000)

FindVariableContainer (see page 23998)

FindNamedField (see page 24154)

FindNamedAction (see page 24154)

173.1.34.2.4 TAdapterSubComponent.FindVariable Method

Returns the item from this list with a specified variable name.

Pascal

```
function FindVariable(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariable(const AnsiString AName);
```

Description

FindVariable method is called from server-side script to locate a field or action with a specified name. This method is identical to the FindNamedVariable (see page 23997) method, except that it is not virtual. Instead, FindVariable calls the virtual FindNamedVariable (see page 23997) method.

AName is a variable name.

FindNamedVariable (see page 23997) returns the item from the list with the specified variable name. If none of the items in the list have the name AName, FindNamedVariable (see page 23997) returns nil (Delphi) or NULL (C++).

See Also

VariablesInUse (see page 24007)

FindNamedVariable (see page 23997)

GetVariable (see page 24000)

GetVariableCount (see page 24000)

FindVariableContainer (see page 23998)

173.1.34.2.5 TAdapterSubComponent.FindVariableContainer Method

Returns the Web module that contains a specified item from this list.

Pascal

```
function FindVariableContainer(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariableContainer(const AnsiString AName);
```

Description

FindVariableContainer searches the list of child items that have variable names (the VariablesInUse (see page 24007) property) to find one with the specified name. It then returns the Web module that contains the component it located.

AName is the name of the action or field to locate.

FindVariableContainer returns the Web module that containse the specified item. If there is no item in the list with the name AName, FindVariableContainer returns nil (Delphi) or NULL (C++). If FindVariableContainer includes an item with the specified name, but is unable to determine what Web module contains that item, FindVariableContainer raises an EAdapterException (see page 23696) exception.

See Also

VariablesInUse (see page 24007)

FindNamedVariable (see page 23997)

GetVariable (see page 24000)

GetVariableCount (see page 24000)

EAdapterException (see page 23696)

173.1.34.2.6 TAdapterSubComponent.GetCurrentObject Method

Returns the current field or action from the list when an enumerator is iterating over child items.

Pascal

```
function GetCurrentObject(OwnerData: Pointer): TObject;
```

C++

```
__fastcall TObject * GetCurrentObject(void * OwnerData);
```

Description

GetCurrentObject is part of the support that allows server-side script to use an enumerator to iterate over all the child items in the collection. This method returns the current child item.

OwnerData is a pointer to object-specific information returned by the last call to the StartIterator (see page 24006) or NextIteration (see page 24003) method. TAdapterSubComponent (see page 23991) does not use the OwnerData parameter.

See Also

StartIterator (see page 24006)

NextIteration (see page 24003)

EndIterator (see page 23997)

173.1.34.2.7 TAdapterSubComponent.GetDefaultWebComponents Method

Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function GetDefaultWebComponents: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents supplies the default set of child items for this component. As implemented in TAdapterSubComponent (see page 23991), GetDefaultWebComponents always returns nil (Delphi) or NULL (C++) because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions (see page 23770) and TAdapterDefaultFields (see page 23778)) override this method to return the list of default fields or actions.

See Also

GetWebComponents (see page 24756)

TAdapterDefaultActions (see page 23770)

TAdapterDefaultFields (see page 23778)

173.1.34.2.8 TAdapterSubComponent.GetParentComponent Method

Returns the associated adapter.

Pascal

```
function GetParentComponent: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetParentComponent();
```

Description

TAdapterSubComponent (see page 23991) overrides the GetParentComponent method that is introduced in TComponent for the streaming system that loads and saves components. It returns the value of the Adapter (see page 24006) property (the adapter that uses this component to represent the set of its fields or actions).

See Also

GetParentComponent

HasParent (see page 24002)

Adapter (see page 24006)

173.1.34.2.9 TAdapterSubComponent.GetVariable Method

Returns a child item from this list, given its position in the list of child items with variable names.

Pascal

```
function GetVariable(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetVariable(int I);
```

Description

FindNamedVariable (see page 23997) indexes into the list of child items that have variable names (the VariablesInUse (see page 24007) property) and returns the item in the specified position.

I is the index of the child item, where 0 is the first item that can be reference from server-side script, 1 is the second item, and so on. The GetVariableCount (see page 24000) method returns an upper limit for this index.

See Also

VariablesInUse (see page 24007)

FindNamedVariable (see page 23997)

FindVariable (see page 23998)

GetVariableCount (see page 24000)

173.1.34.2.10 TAdapterSubComponent.GetVariableCount Method

Indicates the number of items in this list that can be accessed from server-side script.

Pascal

```
function GetVariableCount: Integer;
```

C++

```
__fastcall int GetVariableCount();
```

Description

GetVariableCount returns the number of child items that have variable names. This is the number of items in the VariablesInUse (see page 24007) proprety. Use GetVariableCount as an upper bound to the I parameter of the GetVariable (see page 24000) method.

See Also

VariablesInUse (see page 24007)

GetVariable (see page 24000)

173.1.34.2.11 TAdapterSubComponent.GetVariablesInUse Method

Returns the value of the VariablesInUse (see page 24007) property.

Pascal

```
function GetVariablesInUse: TWebComponentList; virtual;
```

C++

```
virtual __fastcall TWebComponentList GetVariablesInUse();
```

Description

GetVariablesInUse is the protected read implementation for the VariablesInUse (see page 24007) property.

In TAdapterSubComponent (see page 23991), GetVariablesInUse calls the GetWebComponents method to return a list of all the items in the list. Descendant classes can override this method to return only those items that meet a certain criterion.

See Also

VariablesInUse (see page 24007)

GetWebComponents (see page 24756)

173.1.34.2.12 TAdapterSubComponent.GetWebComponentNodeName Method

Returns a string that indicates the type of items in this collection.

Pascal

```
function GetWebComponentNodeName: string;
```

C++

```
__fastcall AnsiString GetWebComponentNodeName();
```

Description

GetWebComponentNodeName returns the label for the type of items in the collection. For example, if the TAdapterSubComponent (see page 23991) descendant holds the fields belonging to an adapter, GetWebComponentNodeName returns the string 'Fields'. If the TAdapterSubComponent (see page 23991) descendant holds the actions belonging to an adapter, GetWebComponentNodeName returns the string 'Actions'.

GetWebComponentNodeName delegates the task of supplying a name to the virtual ImplGetWebComponentNodeName (see page 24003) method.

See Also

ImplGetWebComponentNodeName (see page 24003)

173.1.34.2.13 TAdapterSubComponent.HasParent Method

Indicates whether this component has an associated adapter.

Pascal

```
function HasParent: Boolean; override;
```

C++

```
virtual __fastcall Boolean HasParent();
```

Description

TAdapterSubComponent (see page 23991) overrides the HasParent method that is introduced in TComponent for the streaming system that loads and saves components. It returns true if the Adapter (see page 24006) property is not nil (Delphi) or NULL (C++).

If the TAdapterSubComponent (see page 23991) object has a parent, that parent component is responsible for writing this component to a stream.

See Also

HasParent

GetParentComponent (see page 24000)

Adapter (see page 24006)

173.1.34.2.14 TAdapterSubComponent.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass (see page 23996) method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass (see page 23996) method calls ImplCanAddClass to determine whether an object of a specified type can be added to this collection.

AParent is the TAdapterSubComponent (see page 23991) descendant to which the Web item will be added.

AClass is the class of the proposed child item.

ImplCanAddClass returns true if the TAdapterSubComponent (see page 23991) descendant can include a child of type AClass, false otherwise.

In TAdapterSubComponent (see page 23991), ImplCanAddClass always returns false, disallowing any child items. Descendant classes override this method to conditionally allow those fields or actions that are appropriate to the associated adapter.

See Also

CanAddClass (see page 23996)

Adapter (see page 24006)

173.1.34.2.15 TAdapterSubComponent.ImplGetWebComponentNodeName Method

Provides the underlying implementation for the GetWebComponentNodeName (see page 24001) method.

Pascal

```
function ImplGetWebComponentNodeName: string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplGetWebComponentNodeName();
```

Description

The GetWebComponentNodeName (see page 24001) method calls ImplGetWebComponentNodeName to obtain a label for the type of items in the collection.

ImplGetWebComponentNodeName returns the name of the type of items in this list.

In TAdapterSubComponent (see page 23991), ImplGetWebComponentNodeName returns an empty string. Descendant classes created for specific types of elements override this method to return an appropriate string such as 'Fields' or 'Actions'.

See Also

GetWebComponentNodeName (see page 24001)

173.1.34.2.16 TAdapterSubComponent.NextIteration Method

Changes the current object to the next item that has a variable name.

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

NextIteration is part of the support that allows server-side script to use an enumerator to iterate over all the child items in the collection. This method is called to move to the next item in the list. After a call to NextIteration, the GetCurrentObject (see page 23999) method returns the next item in the VariablesInUse (see page 24007) list.

OwnerData is a pointer to object-specific information returned by the last call to the StartIterator (see page 24006) or NextIteration method. TAdapterSubComponent (see page 23991) does not use the OwnerData parameter.

NextIteration returns true if it successfully moves to a new item, false if there was no addition item to which it could move.

See Also

StartIterator (see page 24006)

EndIterator (see page 23997)

GetCurrentObject (see page 23999)

VariablesInUse (see page 24007)

173.1.34.2.17 TAdapterSubComponent.NotifyAdapterChange Method

Notifies the associated adapter when the list of items changes.

Pascal

```
procedure NotifyAdapterChange;
```

C++

```
__fastcall NotifyAdapterChange();
```

Description

TAdapterSubComponent (see page 23991) descendants call NotifyAdapterChange internally when the items in this list change. NotifyAdapterChange passes this notification on to the associated adapter so that it can make any necessary adjustments.

See Also

Adapter (see page 24006)

NotifyAdapterChange (see page 24174)

173.1.34.2.18 TAdapterSubComponent.ReadState Method

Reads the component's published properties, stored data, and owned components from a reader object.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

ReadState is part of a sequence of calls used by the streaming system that loads and saves components. It reads the values of all the component's published properties, stored data, and owned components from the reader object passed in Reader. Do not call ReadState directly.

TAdapterSubComponent (see page 23991) overrides ReadState to obtain the parent component from the reader, which it assigns as the value of the Adapter (see page 24006) property.

Note: Any descendant classes overriding ReadState should end with a call to the inherited ReadState method.

See Also

GetParentComponent (see page 24000)

Adapter (see page 24006)

173.1.34.2.19 TAdapterSubComponent.SetDefaultWebComponents Method

Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList); override;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

GetDefaultWebComponents (see page 23999) supplies the default set of child items for this component. As implemented in TAdapterSubComponent (see page 23991), GetDefaultWebComponents (see page 23999) always returns nil (Delphi) or NULL (C++) because this class represents only a single list of items. Descendants that represent the default fields or actions of

the associated adapter (TAdapterDefaultActions (see page 23770) and TAdapterDefaultFields (see page 23778)) override this method to return the list of default fields or actions.

SetDefaultWebComponents assigns the default set of child items for this component. As implemented in TWebContainerComponent, SetDefaultWebComponents does nothing because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions (see page 23770) and TAdapterDefaultFields (see page 23778)) override this method to store the list of default fields or actions.

See Also

GetDefaultWebComponents (see page 23999)

IsDefaultField (see page 24728)

173.1.34.2.20 TAdapterSubComponent.SetParentComponent Method

Sets the Adapter (see page 24006) property to be the same as this component's parent.

Pascal

```
procedure SetParentComponent(AParent: TComponent); override;
```

C++

```
virtual __fastcall SetParentComponent(TComponent * AParent);
```

Description

TAdapterSubComponent (see page 23991) overrides the SetParentComponent method that is introduced in TComponent for the streaming system that loads and saves components. It sets the Adapter (see page 24006) property to AParent, so that it is the same as the parent that loads and saves this component.

See Also

SetParentComponent

HasParent (see page 24002)

Adapter (see page 24006)

173.1.34.2.21 TAdapterSubComponent.SetWebParent Method

Introduces a method to associate this component with an adapter.

Pascal

```
procedure SetWebParent(const Value: TComponent); virtual;
```

C++

```
virtual __fastcall SetWebParent(const TComponent * Value);
```

Description

TAdapterSubComponent (see page 23991) introduces SetWebParent as a virtual method for setting the read-only Adapter (see page 24006) property.

Value is the adapter component that acts as a parent to this collection.

In TAdapterSubComponent (see page 23991), SetWebParent does nothing. Descendant classes override this method to set the Adapter (see page 24006) property and populate the list with actions or fields appropriate to that adapter.

See Also

Adapter (see page 24006)

173.1.34.2.22 TAdapterSubComponent.StartIterator Method

Makes the first child item with a variable name the current object.

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

StartIterator is part of the support that allows server-side script to use an enumerator to iterate over all the child items in the collection. This method is called to begin iterating over the items in the list. After a call to StartIterator, the GetCurrentObject (see page 23999) method returns the first item in the VariablesInUse (see page 24007) list.

OwnerData is a pointer to object-specific information returned that should be passed to the NextIteration (see page 24003) or EndIterator (see page 23997) method. TAdapterSubComponent (see page 23991) does not use the OwnerData parameter.

StartIterator returns true if it successfully moves to the first item, false if the VariablesInUse (see page 24007) property does not list any items.

See Also

NextIteration (see page 24003)

EndIterator (see page 23997)

GetCurrentObject (see page 23999)

VariablesInUse (see page 24007)

173.1.34.3 TAdapterSubComponent Properties

The properties of the TAdapterSubComponent class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 24006)	Indicates the adapter that uses this object to manage its fields or actions.
R	VariablesInUse (see page 24007)	Lists those fields or actions in the collection for which variables are defined.

173.1.34.3.1 TAdapterSubComponent.Adapter Property

Indicates the adapter that uses this object to manage its fields or actions.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

Read Adapter to access the adapter component that uses this collection. This TAdapterSubComponent (see page 23991) object is the value of the adapter's Data or Actions property.

See Also

Actions (see page 24181)

Data (see page 24181)

Actions (see page 24402)

Data (see page 24402)

173.1.34.3.2 TAdapterSubComponent.VariablesInUse Property

Lists those fields or actions in the collection for which variables are defined.

Pascal

```
property VariablesInUse: TWebComponentList;
```

C++

```
__property TWebComponentList VariablesInUse;
```

Description

Read VariablesInUse to access the child items (fields or actions) that can be addressed from server-side script.

TAdapterSubComponent (see page 23991) implements VariablesInUse to include all of the fields or actions that have been added to the list. Descendant classes can override this property to provide a different set of items, such as the default fields or actions of the associated adapter.

See Also

FindNamedVariable (see page 23997)

FindVariable (see page 23998)

GetVariable (see page 24000)

GetVariableCount (see page 24000)

FindVariableContainer (see page 23998)

173.1.35 TAdapterUpdateField Class

TAdapterUpdateField is a base class for adapter field components that support data entry into an HTML form.

Class Hierarchy

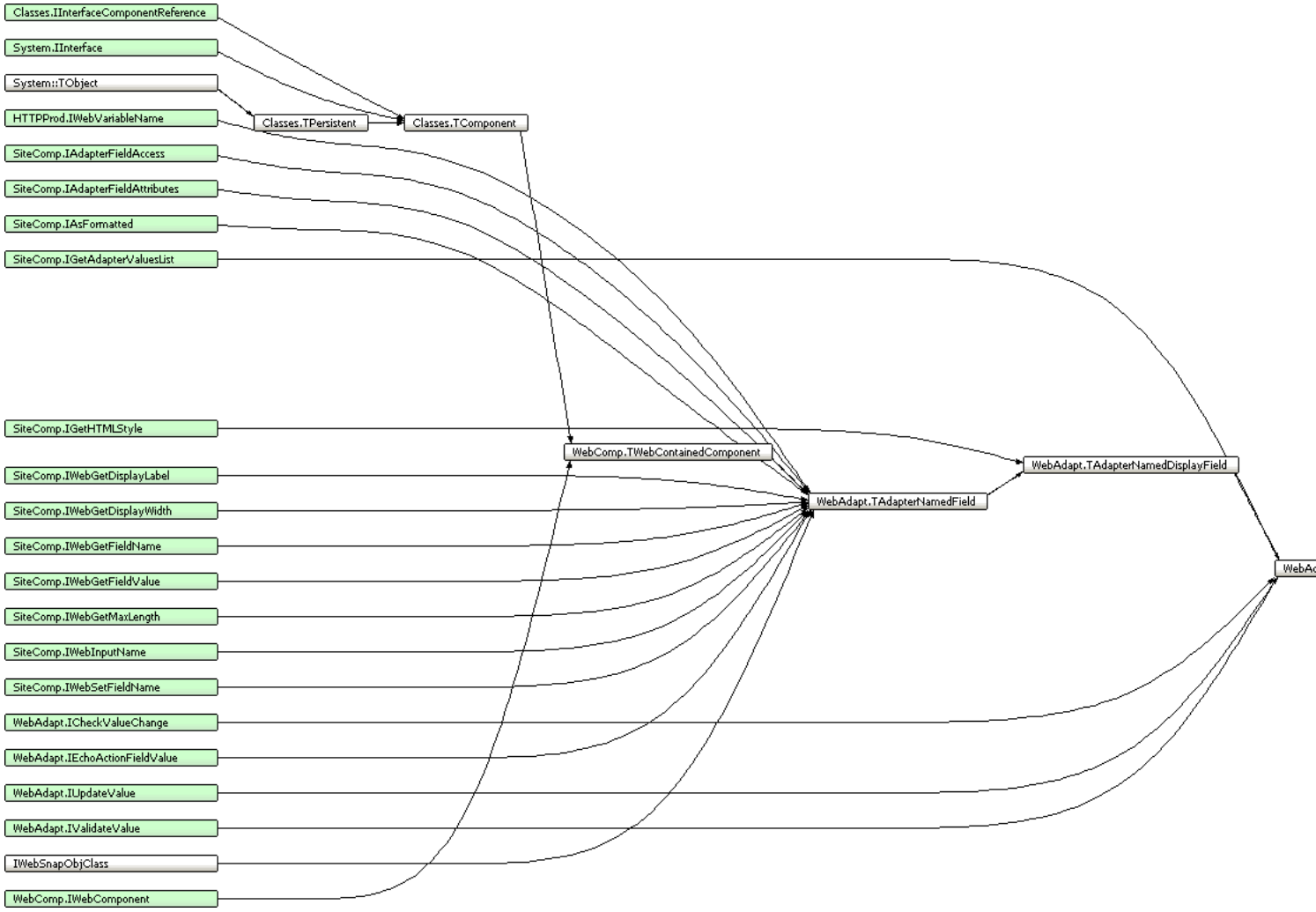

Pascal

```
TAdapterUpdateField = class(TAdapterNamedDisplayField, IGetAdapterValuesList, IUpdateValue,
IValidateValue, ICheckValueChange);
```

C++

```
class TAdapterUpdateField : public TAdapterNamedDisplayField, public IGetAdapterValuesList,
public IUpdateValue, public IValidateValue, public ICheckValueChange;
```

File

WebAdapt

Description

TAdapterUpdateField implements the interfaces necessary to support field validation and updating. Descendant classes support different types of data that can be entered by the user (for example, image upload, file upload, single value, multiple value, memo and Boolean).

See Also

TAdapterNamedDisplayField (see page 23913)

TCustomAdapterField (see page 24211)

TAdapter (see page 23713)

173.1.35.1 TAdapterUpdateField Members

The following tables list the members exposed by TAdapterUpdateField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 24015)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 24015)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 24015)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 24016)	Retrieves a value entered by the end-user.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoOnValidateValue (see page 24016)	Validates field values entered by the end-user.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 24016)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 24017)	Indicates the preferred HTML element to use when editing this adapter field's value.

	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetValueCount (see page 24017)	Provides the number of values associated with this adapter field.
	GetValues (see page 24017)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 24018)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 24018)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplGetDisplayLabel (see page 24018)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 24019)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 24019)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
	ImplGetVariableName (see page 24019)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 24020)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 24020)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 24020)	Updates the adapter field with value(s) entered by the end-user.
	ImplValidateValue (see page 24020)	Validates values(s) entered by the end-user.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 24021)	Indicates whether this adapter field can support multiple values.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 24021)	Clears references to a deleted component.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.

	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 24021)	Updates the adapter field with value(s) entered by the end-user.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateValue (see page 24022)	Validates values(s) entered by the end-user.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 24023)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 24024)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 24014)	Creates an instance of TAdapterUpdateField (see page 24007).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 23923)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23924)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 23924)	Indicates whether this adapter field is read-only.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 24022)	Retrieves the value associated with this adapter field.
	ValuesList (see page 24023)	Associates a list of values with the adapter field.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.35.2 TAdapterUpdateField.Create Constructor

Creates an instance of TAdapterUpdateField (see page 24007).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TAdapterUpdateField(TComponent * AOwner);
```

Description

Do not call TAdapterUpdateField (see page 24007). It will be called automatically when the TAdapterUpdateField (see page 24007) object is created.

See Also

Create_d

173.1.35.3 TAdapterUpdateField Methods

The methods of the TAdapterUpdateField class are listed here.

Protected Methods

	Name	Description
	CheckValueChange (see page 24015)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
V	ConvertValue (see page 24015)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 24015)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 24016)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 24016)	Validates field values entered by the end-user.
	GetAdapterValuesList (see page 24016)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
V	GetInputStyleType (see page 24017)	Indicates the preferred HTML element to use when editing this adapter field's value.
V	GetValueCount (see page 24017)	Provides the number of values associated with this adapter field.
V	GetValues (see page 24017)	Returns one of the values provided by this adapter field.
V	ImplAsFormatted (see page 24018)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
V	ImplCheckValueChange (see page 24018)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
V	ImplGetDisplayLabel (see page 24018)	Gets the preferred display label for this adapter field.
V	ImplGetDisplayWidth (see page 24019)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
V	ImplGetEchoActionFieldValue (see page 24019)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
V	ImplGetVariableName (see page 24019)	Gets the name used to access this adapter field with server-side script.
V	ImplSetEchoActionFieldValue (see page 24020)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.

	ImplSetFieldName (see page 24020)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 24020)	Updates the adapter field with value(s) entered by the end-user.
	ImplValidateValue (see page 24020)	Validates values(s) entered by the end-user.
	IsMultiValueField (see page 24021)	Indicates whether this adapter field can support multiple values.
	Notification (see page 24021)	Clears references to a deleted component.
	UpdateValue (see page 24021)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 24022)	Validates values(s) entered by the end-user.

173.1.35.3.1 TAdapterUpdateField.CheckValueChange Method

Checks the value(s) entered by the end-user against the value associated with this adapter field.

Pascal

```
function CheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
```

C++

```
__fastcall Boolean CheckValueChange(IActionRequest AActionRequest, int AFieldIndex);
```

Description

CheckValueChange is called to determine whether the value(s) entered by the user is different from the value(s) provided by the adapter field. CheckValueChange calls the virtual function ImplCheckValueChange (see page 24018).

See Also

ImplCheckValueChange (see page 24018)

173.1.35.3.2 TAdapterUpdateField.ConvertValue Method

Retrieves a value entered by the end-user.

Pascal

```
function ConvertValue(AValue: IActionFieldValue): Variant; virtual;
```

C++

```
virtual __fastcall Variant ConvertValue(IActionFieldValue AValue);
```

Description

ConvertValue is called to retrieve, as a variant, the value entered by the user for this adapter field. ConvertValue using index 0 to retrieve a value from the IActionFieldValue (see page 24596) parameter. Typically, IActionFieldValue (see page 24596) will contain a string value. Descendants may override this function to caste the value into another type.

See Also

ConvertValue (see page 24208)

173.1.35.3.3 TAdapterUpdateField.ConvertValueCount Method

Retrieves the number of values entered by the end-user.

Pascal

```
function ConvertValueCount(AValue: IActionFieldValue): Integer; virtual;
```

C++

```
virtual __fastcall int ConvertValueCount(IActionFieldValue AValue);
```

Description

ConvertValueCount is called to retrieve the number of values entered by the user. ConvertValueCount returns 1. Descendants may override this method.

173.1.35.3.4 TAdapterUpdateField.ConvertValues Method

Retrieves a value entered by the end-user.

Pascal

```
function ConvertValues(AValue: IActionFieldValue; Index: Integer): Variant; virtual;
```

C++

```
virtual __fastcall Variant ConvertValues(IActionFieldValue AValue, int Index);
```

Description

ConvertValues is called to retrieve, as a variant, a particular value entered by the end-user for this adapter field.

173.1.35.3.5 TAdapterUpdateField.DoOnValidateValue Method

Validates field values entered by the end-user.

Pascal

```
procedure DoOnValidateValue(AValue: IActionFieldValue; var AHandled: Boolean); virtual;
```

C++

```
virtual __fastcall DoOnValidateValue(IActionFieldValue AValue, Boolean AHandled);
```

Description

DoOnValidateValue is called when the field value entered by the end-user is to be validated. DoOnValidateValue calls the OnValidateValue (see page 24024) event handler. DoOnValidateValue may be overridden by descendants.

See Also

OnValidateValue (see page 24024)

ImplValidateValue (see page 24020)

173.1.35.3.6 TAdapterUpdateField.GetAdapterValuesList Method

Provides an interface to a component that can be used to generate a list of possible values for this adapter field.

Pascal

```
function GetAdapterValuesList: IValuesListAdapter;
```

C++

```
__fastcall IValuesListAdapter GetAdapterValuesList();
```

Description

GetAdapterValuesList provides an interface to a component that can be used to generate a list of possible values for this adapter field. It does this by returning the value of the ValuesList (see page 24023) property.

See Also

IValuesListAdapter (see page 19740)

173.1.35.3.7 TAdapterUpdateField.GetInputStyleType Method

Indicates the preferred HTML element to use when editing this adapter field's value.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const AnsiString
AAdapterMode);
```

Description

GetInputStyleType returns htmliSelect if the ValuesList (see page 24023) property is not nil (Delphi) or NULL (C++). Otherwise, htmliTextInput is returned.

See Also

TAdapterInputHTMLElementType

ValuesList (see page 24023)

173.1.35.3.8 TAdapterUpdateField.GetValueCount Method

Provides the number of values associated with this adapter field.

Pascal

```
function GetValueCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetValueCount();
```

Description

GetValueCount returns the number of values provided by this adapter field. GetValueCount returns 1. Descendant classes override this method.

See Also

TAdapterMultiValueField (see page 23900)

GetValueCount

173.1.35.3.9 TAdapterUpdateField.GetValues Method

Returns one of the values provided by this adapter field.

Pascal

```
function GetValues(I: Integer): Variant; virtual;
```

C++

```
virtual __fastcall Variant GetValues(int I);
```

Description

GetValues returns one of the values provided by this adapter field. GetValues calls GetValue (see page 23935). Descendant classes override this method.

See Also

GetValues

TAdapterMultiValueField (see page 23900)

173.1.35.3.10 TAdapterUpdateField.ImplAsFormatted Method

Returns a string representation of the adapter field's value that is appropriate for a read-only display.

Pascal

```
function ImplAsFormatted: string; override;
```

C++

```
virtual __fastcall AnsiString ImplAsFormatted();
```

Description

ImplAsFormatted calls the OnGetDisplayText (see page 24023) event handler, if one is defined. Otherwise, ImplAsFormatted returns the adapter field's value converted to a string. Server-side script accesses the value returned by this function using the AsDisplayText property.

See Also

OnGetDisplayText (see page 24023)

GetValue (see page 23935)

173.1.35.3.11 TAdapterUpdateField.ImplCheckValueChange Method

Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.

Pascal

```
function ImplCheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
virtual;
```

C++

```
virtual __fastcall Boolean ImplCheckValueChange(IActionRequest AActionRequest, int
AFieldIndex);
```

Description

ImplCheckValueChange compares the number and content of values associated with the adapter field with the number and content of values entered by the end-user.

See Also

CheckValueChange (see page 24015)

173.1.35.3.12 TAdapterUpdateField.ImplGetDisplayLabel Method

Gets the preferred display label for this adapter field.

Pascal

```
function ImplGetDisplayLabel: string; override;
```

C++

```
virtual __fastcall AnsiString ImplGetDisplayLabel();
```

Description

ImplGetDisplayLabel returns the value of the DisplayLabel (see page 23923) property.

See Also

DisplayLabel (see page 23923)

173.1.35.3.13 TAdapterUpdateField.ImplGetDisplayWidth Method

Returns the preferred display width for the HTML element used to edit or display this adapter field's value.

Pascal

```
function ImplGetDisplayWidth: Integer; override;
```

C++

```
virtual __fastcall int ImplGetDisplayWidth();
```

Description

ImplGetDisplayWidth returns the value of the DisplayWidth (see page 23924) property.

See Also

DisplayWidth (see page 23924)

173.1.35.3.14 TAdapterUpdateField.ImplGetEchoActionFieldValue Method

Gets a flag that causes the adapter field to return the value(s) entered by the end-user.

Pascal

```
function ImplGetEchoActionFieldValue: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplGetEchoActionFieldValue();
```

Description

Gets a flag that causes the adapter field to return the value(s) entered by the end-user.

See Also

ImplSetEchoActionFieldValue (see page 24020)

173.1.35.3.15 TAdapterUpdateField.ImplGetVariableName Method

Gets the name used to access this adapter field with server-side script.

Pascal

```
function ImplGetVariableName: string; override;
```

C++

```
virtual __fastcall AnsiString ImplGetVariableName();
```

Description

ImplGetVariableName returns the value of the FieldName (see page 23944) property.

See Also

FieldName (see page 23944)

173.1.35.3.16 TAdapterUpdateField.ImplSetEchoActionFieldValue Method

Sets a flag that causes the adapter field to return the value(s) entered by the end-user.

Pascal

```
procedure ImplSetEchoActionFieldValue(AValue: Boolean); override;
```

C++

```
virtual __fastcall ImplSetEchoActionFieldValue(Boolean AValue);
```

Description

ImplSetEchoActionFieldValue is typically called when an error occurs during the execution of an adapter action. It allows the page generated after the error to display the values entered by the user rather than the action adapter field values.

173.1.35.3.17 TAdapterUpdateField.ImplSetFieldName Method

Sets the name used to access this adapter field with server-side script.

Pascal

```
procedure ImplSetFieldName(const Value: string); override;
```

C++

```
virtual __fastcall ImplSetFieldName(const AnsiString Value);
```

Description

ImplSetVariableName sets the value of the FieldName (see page 23944) property.

See Also

FieldName (see page 23944)

173.1.35.3.18 TAdapterUpdateField.ImplUpdateValue Method

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); virtual;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

ImplUpdateValue is a placeholder. Descendants override this method.

See Also

UpdateValue (see page 24021)

173.1.35.3.19 TAdapterUpdateField.ImplValidateValue Method

Validates values(s) entered by the end-user.

Pascal

```
procedure ImplValidateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); virtual;
```

C++

```
virtual __fastcall ImplValidateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

ImplValidateValue calls the virtual function DoOnValidateValue (see page 24016) after extracting an IActionFieldValue (see page 24596) that contains the value(s) entered by the user. The AFieldIndex parameter identifies that location of the IActionFieldValue (see page 24596) in the AActionRequest.

See Also

ValidateValue (see page 24022)

DoOnValidateValue (see page 24016)

IActionRequest (see page 935)

173.1.35.3.20 TAdapterUpdateField.IsMultiValueField Method

Indicates whether this adapter field can support multiple values.

Pascal

```
function IsMultiValueField: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsMultiValueField();
```

Description

IsMultiValueField return true to indicate that this adapter field can provide multiple values. IsMultiValueField returns false. Descendants may override this function.

See Also

GetValueCount (see page 24017)

GetValues (see page 24017)

TAdapterMultiValueField (see page 23900)

173.1.35.3.21 TAdapterUpdateField.Notification Method

Clears references to a deleted component.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called when a component is inserted or deleted. TAdapterUpdateField (see page 24007) clears the ValuesList (see page 24023) property when the associated component is deleted.

173.1.35.3.22 TAdapterUpdateField.UpdateValue Method

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
procedure UpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

```
__fastcall UpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

UpdateValue calls the virtual method ImplUpdateValue (see page 24020).

See Also

ImplUpdateValue (see page 24020)

173.1.35.3.23 TAdapterUpdateField.ValidateValue Method

Validates values(s) entered by the end-user.

Pascal

```
procedure ValidateValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

```
__fastcall ValidateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

ValidateValue calls the virtual function ImplValidateValue (see page 24020).

See Also

ImplValidateValue (see page 24020)

IActionRequest (see page 935)

173.1.35.4 TAdapterUpdateField Properties

The properties of the TAdapterUpdateField class are listed here.

Public Properties

	Name	Description
R	Value (see page 24022)	Retrieves the value associated with this adapter field.
	ValuesList (see page 24023)	Associates a list of values with the adapter field.

173.1.35.4.1 TAdapterUpdateField.Value Property

Retrieves the value associated with this adapter field.

Pascal

```
property Value: Variant;
```

C++

```
__property Variant Value;
```

Description

GetValue (see page 23935) calls the GetValue (see page 23935) method to retrieve the value of the adapter field as a variant.

See Also

GetValue (see page 23935)

173.1.35.4.2 TAdapterUpdateField.ValuesList Property

Associates a list of values with the adapter field.

Pascal

```
property ValuesList: IValuesListAdapter;
```

C++

```
__property IValuesListAdapter ValuesList;
```

Description

Assign a component to the ValuesList property to define possible values for the adapter field. Server-side script can be used to enumerate the name/value pairs of the ValuesList component to generate HTML select lists, check boxes, or radio buttons.

See Also

IValuesListAdapter (see page 19740)

TStringsValuesList (see page 24540)

TDataSetValuesList (see page 8079)

173.1.35.5 TAdapterUpdateField Events

The events of the TAdapterUpdateField class are listed here.

Public Events

	Name	Description
	OnGetDisplayText (see page 24023)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 24024)	Event handler called to validate value(s) entered by the end-user.

173.1.35.5.1 TAdapterUpdateField.OnGetDisplayText Event

Event handler called to retrieve a string representation of the adapter field's value.

Pascal

```
property OnGetDisplayText: TAdapterFieldGetStringEvent;
```

C++

```
__property TAdapterFieldGetStringEvent OnGetDisplayText;
```

Description

The OnGetDisplayText event handler is called to retrieve a string representation an adapter field's value appropriate for a read-only display. If OnGetDisplayText is nil (Delphi) or NULL (C++), then the value of the adapter field will be used (after conversion to a string).

See Also

ImplAsFormatted (see page 24018)

173.1.35.5.2 TAdapterUpdateField.OnValidateValue Event

Event handler called to validate value(s) entered by the end-user.

Pascal

```
property OnValidateValue: TAdapterFieldValidateValueEvent;
```

C++

```
__property TAdapterFieldValidateValueEvent OnValidateValue;
```

Description

This event handler is called by DoOnValidateValue (see page 24016) to validate the value(s) entered by the end-user. The event handler must raise an exception to indicate that the value(s) are not valid.

See Also

DoOnValidateValue (see page 24016)

IActionFieldValue (see page 920)

173.1.36 TAdapterUserNameField Class

TAdapterUserNameField is an adapter field used to access the user name entered by the end-user.

Class Hierarchy

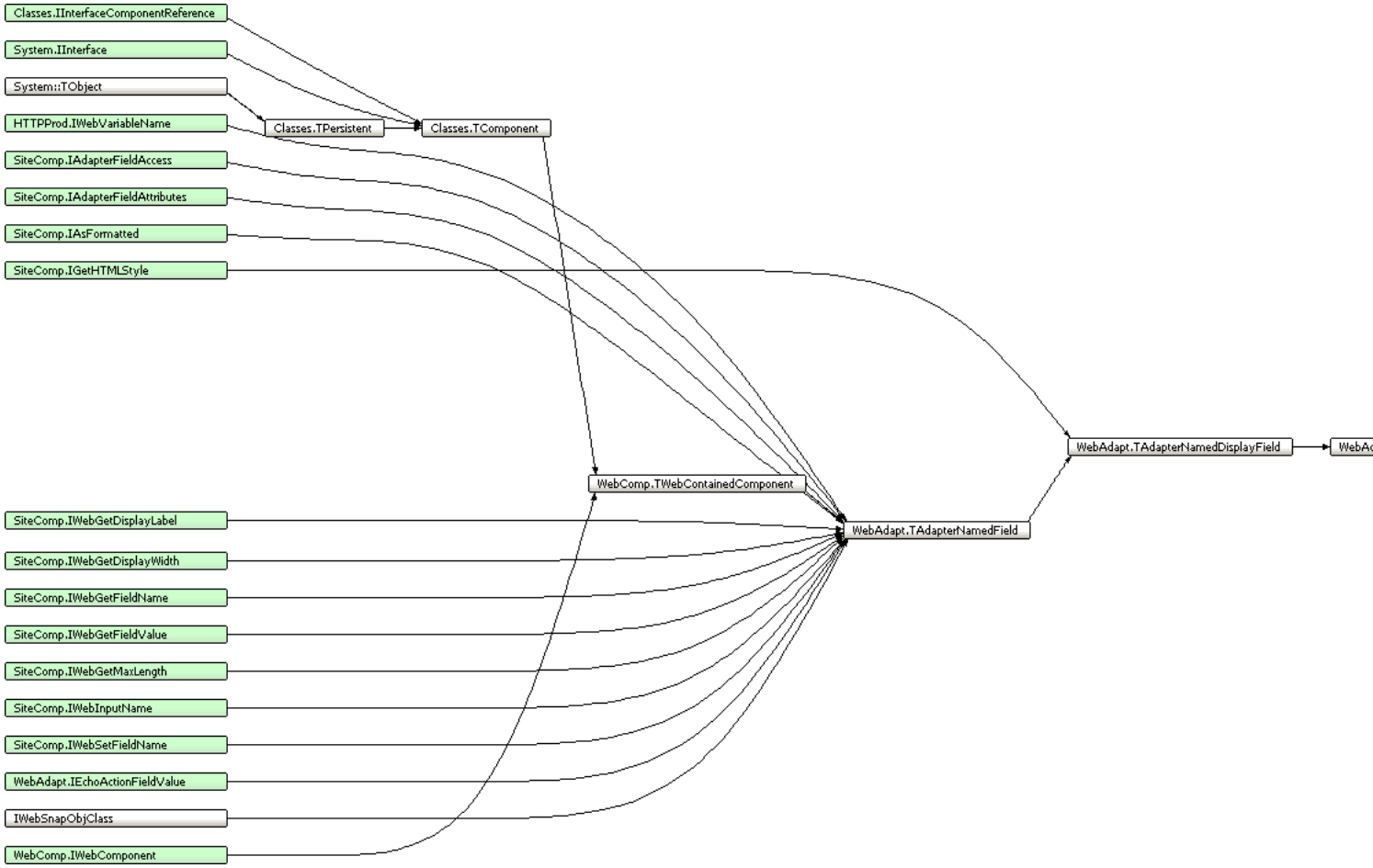

Pascal

```
TAdapterUserNameField = class(TAdapterLoginField);
```

C++

```
class TAdapterUserNameField : public TAdapterLoginField;
```

File

WebAdapt

Description

TAdapterUserNameField is added to TLoginFormAdapter (see page 24500). TAdapterUserNameField, TAdapterPasswordField (see page 23971), TLoginFormAdapterLoginAction (see page 24514) and TAdapterNextPageField (see page 23957) are used with TLoginFormAdapter (see page 24500) to generate an HTML login form. When the HTML form is submitted, TAdapterUserNameField is used to extract the end user's name from the HTTP request.

See Also

TAdapterLoginField (see page 23882)

TLoginFormAdapter (see page 24500)

TAdapterPasswordField (see page 23971)

TAdapterNextPageField (see page 23957)

TLoginFormAdapterLoginAction (see page 24514)

173.1.36.1 TAdapterUserNameField Members

The following tables list the members exposed by TAdapterUserNameField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 23921)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.

	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 24031)	Returns the value associated with this adapter field.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	Adapter (see page 23943)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	Adapter (see page 23888)	Returns the parent adapter of this adapter field.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 23924)	Indicates whether this adapter field is read-only.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	DisplayLabel (see page 24031)	Suggests a display label for this adapter field.
	DisplayWidth (see page 24031)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 24032)	Identifies the name of this field when using server-side script.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.36.2 TAdapterUserNameField Methods

The methods of the TAdapterUserNameField class are listed here.

Protected Methods

	Name	Description
V	ImplGetValue (see page 24031)	Returns the value associated with this adapter field.

173.1.36.2.1 TAdapterUserNameField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns an empty string. Server-side script accesses the result of this function using the EditText or Value (see page 23945) property of the adapter field. These server-side script properties may be used to generate the HTML login form. ImplGetValue returns an empty string so that the generated login form's user name field will be blank. When the login form is submitted, the ActionValue (see page 23943) property can be used to access the user name entered by the end-user.

See Also

ActionValue (see page 23943)

173.1.36.3 TAdapterUserNameField Properties

The properties of the TAdapterUserNameField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 24031)	Suggests a display label for this adapter field.
	DisplayWidth (see page 24031)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 24032)	Identifies the name of this field when using server-side script.

173.1.36.3.1 TAdapterUserNameField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 23933)

173.1.36.3.2 TAdapterUserNameField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 23933)

173.1.36.3.3 TAdapterUserNameField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

173.1.37 TApplicationAdapter Class

TApplicationAdapter is a container for field and action components that are available through the Application script variable.

Class Hierarchy

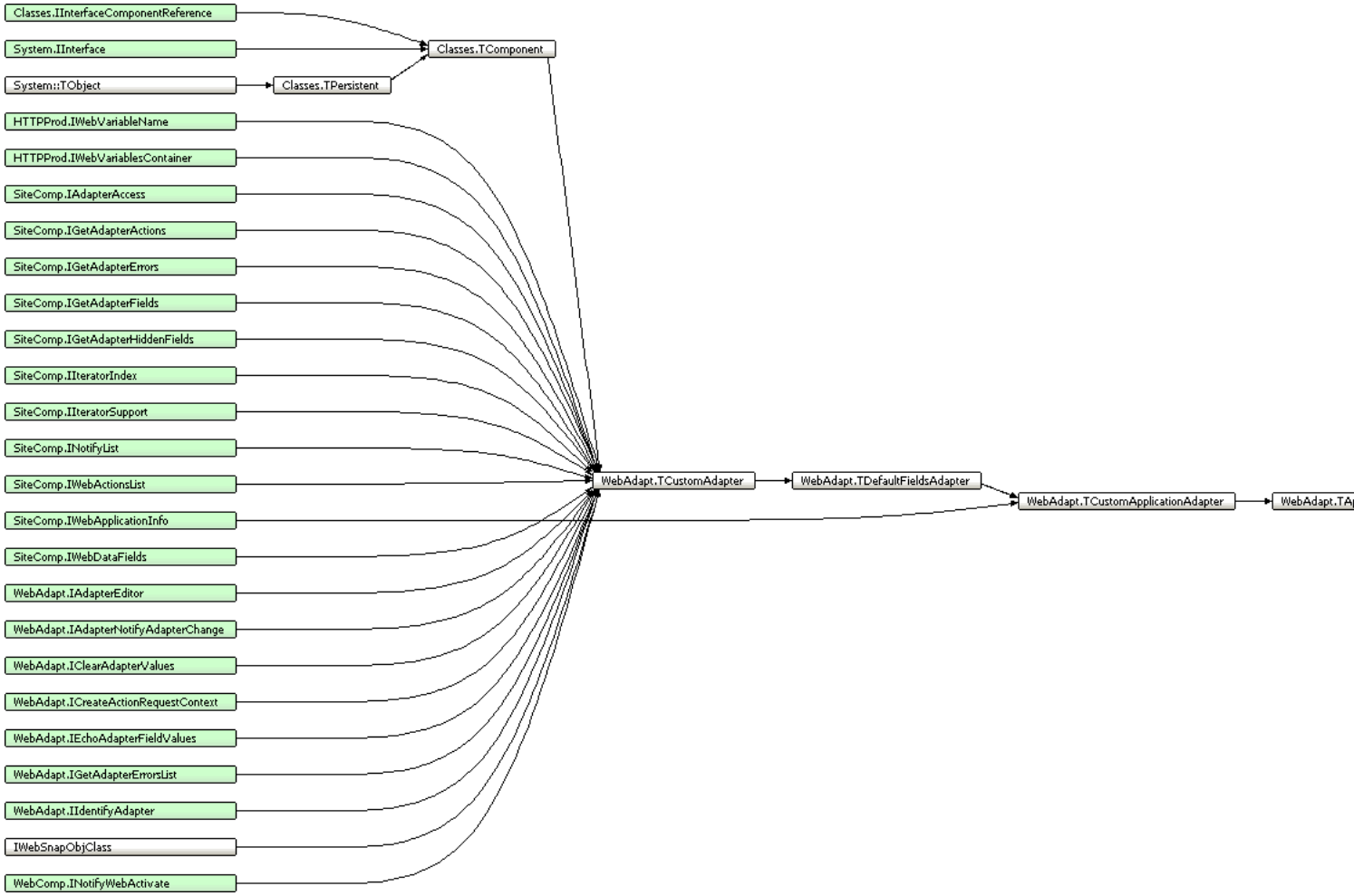

Pascal

```
TApplicationAdapter = class(TCustomApplicationAdapter);
```

C++

```
class TApplicationAdapter : public TCustomApplicationAdapter;
```

File

WebAdapt

Description

By default, TApplicationAdapter contains a single field called Title that gets its value from the ApplicationTitle (see page 24040) property. The TApplicationAdapter can be customized by adding additional field and action components. When writing server-side script, use the Application variable to access fields and actions of the ApplicationAdapter. For example, <%= Application.Title %> writes the title of the application.

See Also

TAdapter (see page 23713)

TApplicationAdapter

173.1.37.1 TApplicationAdapter Members

The following tables list the members exposed by TApplicationAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.

	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetApplicationAdapter (see page 24303)	Implements IWebApplicationInfo's GetApplicationAdapter method.
	GetApplicationTitle (see page 24303)	Implements IWebApplicationInfo's GetApplicationTitle method.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
V	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
V	GetWebSnapObjClass (see page 24304)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddActionClass (see page 24304)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplCanAddFieldClass (see page 24304)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
V	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
V	ImplGetActionsList (see page 24305)	Provides a list of action types that can be added to this adapter.
V	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
V	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.
V	ImplGetFieldsList (see page 24305)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
V	ImplGetVariable (see page 24166)	Returns an action or field.
V	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
V	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
V	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
V	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.

	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.

	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetApplicationAdapter (see page 19747)	Returns the value of the ApplicationAdapter (see page 19747) property.
	GetApplicationTitle (see page 19747)	Returns the value of the ApplicationTitle (see page 19748) property.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	ApplicationAdapter (see page 19747)	Provides access to the adapter that implements the fields and actions of the Application variable in server-side scripts.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.

R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 24041)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24042)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 24042)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24042)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 24043)	Supplies name/value pairs needed to execute an adapter action.

Published Properties

	Name	Description
	Actions (see page 24040)	Container for adapter actions components.
	ApplicationTitle (see page 24040)	Defines the title of the application.
	Data (see page 24041)	Container for adapter field components.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.37.2 TApplicationAdapter Properties

The properties of the TApplicationAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 24040)	Container for adapter actions components.
	ApplicationTitle (see page 24040)	Defines the title of the application.
	Data (see page 24041)	Container for adapter field components.

173.1.37.2.1 TApplicationAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 23770)

173.1.37.2.2 TApplicationAdapter.ApplicationTitle Property

Defines the title of the application.

Pascal

```
property ApplicationTitle: string;
```

C++

```
__property AnsiString ApplicationTitle;__property AnsiString ApplicationTitle;
```

Description

Set this property to identify an application title that can be accessed with server-side script as the Application.Title property.

173.1.37.2.3 TApplicationAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 23778)

173.1.37.3 TApplicationAdapter Events

The events of the TApplicationAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 24041)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24042)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 24042)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24042)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 24043)	Supplies name/value pairs needed to execute an adapter action.

173.1.37.3.1 TApplicationAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

173.1.37.3.2 TApplicationAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.37.3.3 TApplicationAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 24076)

OnAfterExecuteAction (see page 24184)

173.1.37.3.4 TApplicationAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 24076)

OnAfterGetActionResponse (see page 24185)

173.1.37.3.5 TApplicationAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 24077)

173.1.38 TBaseAdapterAction Class

TBaseAdapterAction is a based class for adapter action components.

Class Hierarchy

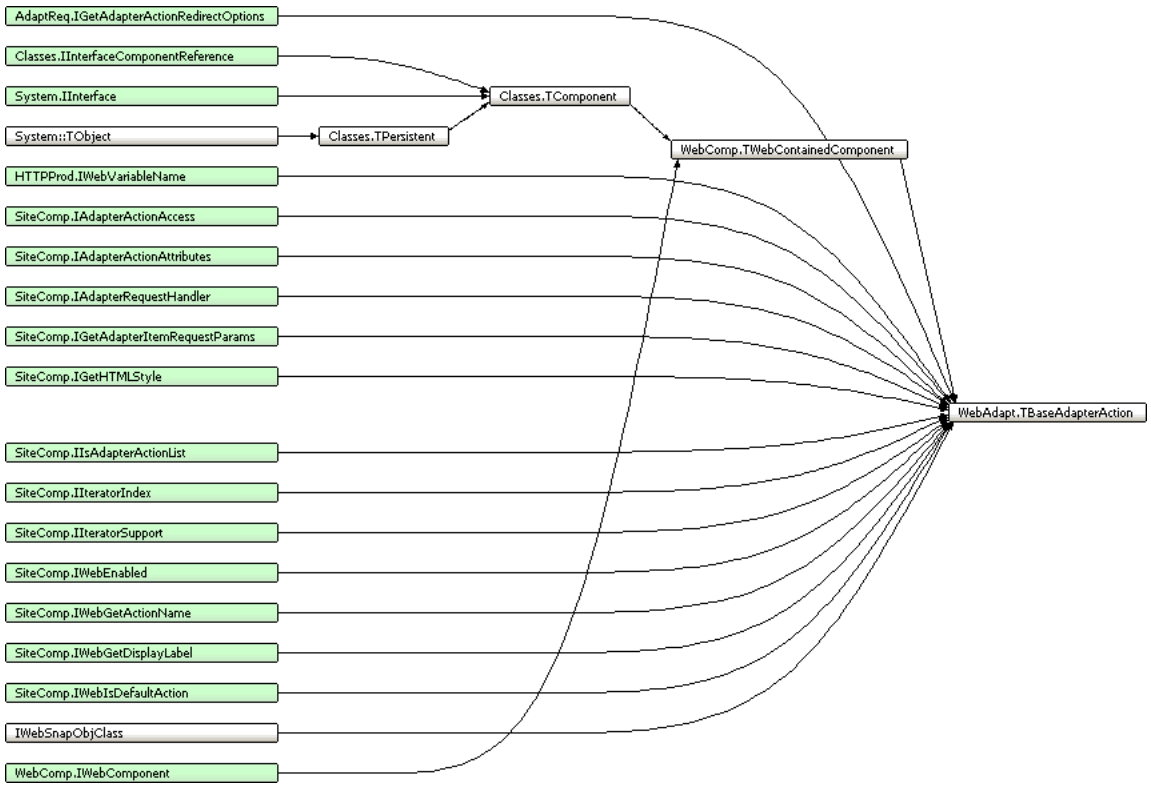

Pascal

```
TBaseAdapterAction = class(TWebContainedComponent, IWebVariableName, IWebEnabled,
IGetAdapterItemRequestParams, IWebGetActionName, IWebGetDisplayLabel, IWebSnapObjClass,
IAdapterActionAccess, IAdapterRequestHandler, IGetAdapterActionRedirectOptions, IGetHTMLStyle,
IAdapterActionAttributes, IIteratorSupport, IIteratorIndex, IIsAdapterActionList,
IWebIsDefaultAction);
```

C++

```
class TBaseAdapterAction : public TWebContainedComponent, public IWebVariableName, public
IWebEnabled, public IGetAdapterItemRequestParams, public IWebGetActionName, public
IWebGetDisplayLabel, public IWebSnapObjClass, public IAdapterActionAccess, public
IAdapterRequestHandler, public IGetAdapterActionRedirectOptions, public IGetHTMLStyle, public
IAdapterActionAttributes, public IIteratorSupport, public IIteratorIndex, public
IIsAdapterActionList, public IWebIsDefaultAction;
```

File

WebAdapt

Description

TBaseAdapterAction implements interfaces used to access an adapter action's properties using server-side script. Most of the interface implementation methods call virtual functions. When defining new derived classes, override these virtual functions to customize the behavior of the adapter action class.

See Also

TCustomAdapterAction (see page 24188)

TAdapter (see page 23713)

173.1.38.1 TBaseAdapterAction Members

The following tables list the members exposed by TBaseAdapterAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.

	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 24055)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
V	ImplExecuteActionRequest (see page 24063)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
V	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
V	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.

V	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
V	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
V	ImplWebEnabled (see page 24069)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 24075)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 24075)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24076)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24076)	Called before the adapter action response is generated.
	OnGetParams (see page 24077)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.

	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 24073)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 24073)	Defines the rights required to execute this action.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RedirectOptions (see page 24074)	Indicates whether to perform an HTTP redirect after successfully executing this action.

R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.38.2 TBaseAdapterAction Methods

The methods of the TBaseAdapterAction class are listed here.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
V	GetDefaultActionName (see page 24055)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
V	ImplExecuteActionRequest (see page 24063)	Executes the adapter action using parameters passed in the HTTP request.

	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 24069)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

173.1.38.2.1 TBaseAdapterAction.CheckExecuteAccess Method

Checks end-user rights to execute this adapter action.

Pascal

```
procedure CheckExecuteAccess;
```

C++

```
__fastcall CheckExecuteAccess();
```

Description

This procedure is called to verify that the current user has rights to execute this action. An exception will be raised if access is denied.

See Also

ExecuteAccess (see page 24073)

HasExecuteAccess (see page 24062)

ExecuteActionRequest (see page 24053)

173.1.38.2.2 TBaseAdapterAction.CreateRequestContext Method

Creates an object that describes the adapter action parameters and adapter field values.

Pascal

```
procedure CreateRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall CreateRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

The adapter dispatcher calls this function before calling HandleRequest (see page 24061). The result of this function is stored in the WebContext variable. During the course of executing this action, the object returned by CreateRequestContext will be used to access the action parameters and field values that were passed in the HTTP request to execute this action.

See Also

ImplCreateRequestContext (see page 24062)

AdapterRequest (see page 24655)

TAdapterDispatcher (see page 24840)

HandleRequest (see page 24061)

173.1.38.2.3 TBaseAdapterAction.DoAfterExecuteActionRequest Method

Calls event handlers after the adapter action is executed.

Pascal

```
procedure DoAfterExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); virtual;
```

C++

```
virtual __fastcall DoAfterExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

DoAfterExecuteActionRequest is called by ExecuteActionRequest (see page 24053). DoAfterExecuteActionRrequest packages the action request parameters into TStrings and calls two event handlers. The parent adapter's OnAfterExecuteAction event handler is called first. If it does not handle the action request then the OnAfterExecute (see page 24075) event handler is called.

See Also

OnAfterExecute (see page 24075)

OnAfterExecuteAction (see page 24184)

DoAfterExecuteActionRequest

ExecuteActionRequest (see page 24053)

173.1.38.2.4 TBaseAdapterAction.DoAfterGetExecuteActionResponse Method

Calls event handlers after the adapter action response is formulated.

Pascal

```
procedure DoAfterGetExecuteActionResponse(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); virtual;
```

C++

```
virtual __fastcall DoAfterGetExecuteActionResponse(IActionRequest AActionRequest,
IActionResponse AActionResponse);
```

Description

DoAfterGetExecuteActionResponse is called by GetExecuteActionResponse (see page 24057). The parent adapter's OnAfterGetActionResponse event handler is called first. If it does not handle the action response then the OnAfterGetResponse (see page 24075) event handler is called.

See Also

OnAfterGetResponse (see page 24075)

OnAfterGetActionResponse (see page 24185)

DoBeforeGetExecuteActionResponse (see page 24052)

GetExecuteActionResponse (see page 24057)

173.1.38.2.5 TBaseAdapterAction.DoBeforeExecuteActionRequest Method

Calls event handlers before the adapter action is executed.

Pascal

```
procedure DoBeforeExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); virtual;
```

C++

```
virtual __fastcall DoBeforeExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

DoBeforeExecuteActionRequest is called by ExecuteActionRequest (see page 24053). DoBeforeExecuteActionRequest packages the action request parameters into TStrings and calls two event handlers. The parent adapter's OnBeforeExecuteAction event handler is called first. If it does not handle the action request then the OnBeforeExecute (see page 24076) event handler is called. If either event handler handles the request then the IActionRequest (see page 24596)'s Handled property is set to true.

See Also

OnBeforeExecute (see page 24076)

OnBeforeExecuteAction (see page 24186)

DoAfterExecuteActionRequest (see page 24051)

ExecuteActionRequest (see page 24053)

173.1.38.2.6 TBaseAdapterAction.DoBeforeGetExecuteActionResponse Method

Calls event handlers before the adapter action response is formulated.

Pascal

```
procedure DoBeforeGetExecuteActionResponse(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); virtual;
```

C++

```
virtual __fastcall DoBeforeGetExecuteActionResponse(IActionRequest AActionRequest,
IActionResponse AActionResponse);
```

Description

DoBeforeGetExecuteActionResponse is called by GetExecuteActionResponse (see page 24057). The parent adapter's OnBeforeGetActionResponse event handler is called first. If it does not handle the action response then the OnBeforeGetResponse (see page 24076) event handler is called. If either event handler handles the action response then the IActionResponse's Handled property is set to true.

See Also

OnBeforeGetResponse (see page 24076)

OnBeforeGetActionResponse (see page 24186)

DoAfterGetExecuteActionResponse (see page 24051)

GetExecuteActionResponse (see page 24057)

173.1.38.2.7 TBaseAdapterAction.EndIterator Method

Implements IIteratorSupport's EndIterator method.

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

IIteratorSupport is used to enumerate the list of commands supported by this adapter action. Most adapter actions have a single command. TAdapterGotoPageAction (see page 23849) is an example of an adapter action that supports multiple commands. EndIterator calls ImplNextIteration (see page 24068).

See Also

IIteratorSupport (see page 19722)

IIteratorIndex (see page 19720)

enditerator (see page 19723)

ImplEndIterator (see page 24062)

173.1.38.2.8 TBaseAdapterAction.ExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse);
```

C++

```
__fastcall ExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ExecuteActionRequest is called by ImplHandleRequest (see page 24067) when a request to execute this adapter action is found in the HTTP request. ExecuteActionRequest first calls CheckExecuteAccess (see page 24050). CheckExecuteAccess (see page 24050) raises an exception if the end-user does not have execute rights. ExecuteActionRequest then calls, in turn, DoBeforeExecuteActionRequest (see page 24052), ImplExecuteActionRequest (see page 24063), and DoAfterExecuteActionRequest (see page 24051).

See Also

DoBeforeExecuteActionRequest (see page 24052)

DoAfterExecuteActionRequest (see page 24051)

CheckExecuteAccess (see page 24050)

ImplExecuteActionRequest (see page 24063)

ImplHandleRequest (see page 24067)

173.1.38.2.9 TBaseAdapterAction.GetActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetActionName: string;
```

C++

```
__fastcall AnsiString GetActionName();
```

Description

GetActionName is called to retrieve the name used to access this adapter field with server-side script. GetActionName calls ImplGetActionName (see page 24063).

See Also

ActionName (see page 24072)

ImplGetActionName (see page 24063)

173.1.38.2.10 TBaseAdapterAction.GetAdapter Method

GetAdapter returns the parent adapter component.

Pascal

```
function GetAdapter: TCustomAdapter;
```

C++

```
__fastcall TCustomAdapter GetAdapter();
```

Description

To improve the efficiency of this function, a reference to the parent adapter component is cached in an instance variable.

See Also

Adapter (see page 24072)

173.1.38.2.11 TBaseAdapterAction.GetAdapterItemRequestParams Method

Provides the name/value pairs needed to execute this adapter action.

Pascal

```
procedure GetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams);
```

C++

```
__fastcall GetAdapterItemRequestParams(AnsiString AIdentifier, IAdapterItemRequestParams
Params);
```

Description

The name/value pairs provided by this method will written to the HTTP response when the HTML page is generated and passed back in the HTTP request when this adapter action is to be executed. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted.

See Also

ImplGetAdapterItemRequestParams (see page 24063)

173.1.38.2.12 TBaseAdapterAction.GetDefaultActionName Method

Gets the default name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetDefaultActionName();
```

Description

GetDefaultActionName returns the name of the adapter action component. Descendants may override this function.

See Also

ImplGetActionName (see page 24063)

173.1.38.2.13 TBaseAdapterAction.GetDefaultDisplayLabel Method

Provides the default display label for this adapter action.

Pascal

```
function GetDefaultDisplayLabel: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetDefaultDisplayLabel();
```

Description

When the display label for the action matches this value then the display label property does not need to be stored in the form file.

173.1.38.2.14 TBaseAdapterAction.GetDisplayLabel Method

Returns the suggested display label for this adapter action.

Pascal

```
function GetDisplayLabel: string;
```

C++

```
__fastcall AnsiString GetDisplayLabel();
```

Description

GetDisplayLabel provides label that may be used when generating HTML to display a button or hyperlink for this adapter action. The value returned by this function can be accessed with server-side script using the DisplayLabel (see page 24073) property. GetDisplayLabel calls ImplGetDisplayLabel (see page 24064).

See Also

DisplayLabel (see page 24073)

ImplGetDisplayLabel (see page 24064)

173.1.38.2.15 TBaseAdapterAction.GetDisplayStyle Method

Returns a string value to suggest an HTML display style for this adapter action.

Pascal

```
function GetDisplayStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall AnsiString GetDisplayStyle(const AnsiString AAdapterMode);
```

Description

GetDisplayStyle implements IGetHTMLStyle's GetDisplayStyle method by translating the result value of GetDisplayStyleType (see page 24056) into a string.

See Also

GetDisplayStyle (see page 19700)

GetDisplayStyleType (see page 24056)

173.1.38.2.16 TBaseAdapterAction.GetDisplayStyleType Method

Suggests an HTML display type for this adapter action.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterActionHTMLElementType;
virtual;
```

C++

```
virtual __fastcall TAdapterActionHTMLElementType GetDisplayStyleType(const AnsiString
AAdapterMode);
```

Description

GetDisplayStyleType returns htmlaButton, indicating that this adapter action should be displayed using a submit button. Descendant classes may override this method to suggest other HTML display types.

See Also

GetDisplayStyle (see page 24056)

173.1.38.2.17 TBaseAdapterAction.GetErrors Method

Pascal

```
function GetErrors: TBaseAdapterErrorsList;
```

C++

```
__fastcall TBaseAdapterErrorsList GetErrors();
```

Description

This is GetErrors, a member of class TBaseAdapterAction.

173.1.38.2.18 TBaseAdapterAction.GetExecuteActionResponse Method

Generates a response after the adapter action has been executed.

Pascal

```
procedure GetExecuteActionResponse(AActionRequest: IActionRequest; AActionResponse:
IActionResponse);
```

C++

```
__fastcall GetExecuteActionResponse(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

GetExecuteActionResponse is called by ImplHandleRequest (see page 24067) after executing an adapter action specified by the HTTP request. GetExecuteActionResponse calls, in turn, DoBeforeGetExecuteActionResponse (see page 24052), ImplGetExecuteActionResponse (see page 24064), and DoAfterGetExecuteActionResponse (see page 24051).

See Also

IActionRequest (see page 935)

IActionResponse (see page 941)

DoBeforeGetExecuteActionResponse (see page 24052)

DoAfterGetExecuteActionResponse (see page 24051)

ImplGetExecuteActionResponse (see page 24064)

ImplHandleRequest (see page 24067)

173.1.38.2.19 TBaseAdapterAction.GetFailureOptions Method

Indicates whether to perform an HTTP redirect after execution of this adapter action fails.

Pascal

```
function GetFailureOptions: TAdapterActionRedirectOptions;
```

C++

```
__fastcall TAdapterActionRedirectOptions GetFailureOptions();
```

Description

Set RedirectOptions (see page 24074) when an HTTP redirect should be performed after execution of this adapter action fails. GetFailureOptions returns the result of ImplGetFailureOptions (see page 24065).

See Also

ImplGetFailureOptions (see page 24065)

GetSuccessOptions (see page 24059)

RedirectOptions (see page 24074)

173.1.38.2.20 TBaseAdapterAction.GetInputStyle Method

Returns a string value to suggest an HTML display style for this adapter action.

Pascal

```
function GetInputStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall AnsiString GetInputStyle(const AnsiString AAdapterMode);
```

Description

GetInputStyle implements IGetHTMLStyle's GetInputType method by returning the result of GetDisplayStyle (see page 24056). This method must be implemented as part of IGetHTMLStyle. GetInputStyle is treated as a synonym of GetDisplayStyle (see page 24056).

See Also

GetInputStyle (see page 19700)

GetDisplayStyle (see page 24056)

173.1.38.2.21 TBaseAdapterAction.GetIsAdapterActionList Method

Indicates whether this adapter action may have multiple commands.

Pascal

```
function GetIsAdapterActionList: Boolean;
```

C++

```
__fastcall Boolean GetIsAdapterActionList();
```

Description

An adapter action may have a list of commands. The TAdapterGotoPageAction (see page 23849) is an example of an adapter action with a list of commands. Each command goes to a different page in the parent TPagedAdapter (see page 24525). GetIsAdapterActionList calls ImplGetIsAdapterActionList (see page 24065).

See Also

ImplGetIsAdapterActionList (see page 24065)

TAdapterGotoPageAction (see page 23849)

173.1.38.2.22 TBaseAdapterAction.GetItemCount Method

Returns the number of commands defined by this adapter action.

Pascal

```
function GetItemCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetItemCount();
```

Description

An adapter action may have a list of commands. The TAdapterGotoPageAction (see page 23849) is an example of an adapter action with a list of commands. Each command goes to a different page in the parent TPagedAdapter (see page 24525). GetItemCount returns 1. GetItemCount may be overridden by a descendant class.

See Also

ItemCount (see page 24074)

ItemIndex (see page 24074)

TAdapterGotoPageAction (see page 23849)

GetIsAdapterActionList (see page 24058)

173.1.38.2.23 TBaseAdapterAction.GetIteratorIndex Method

Implements IIteratorIndex's GetIteratorIndex method.

Pascal

```
function GetIteratorIndex: Integer;
```

C++

```
__fastcall int GetIteratorIndex();
```

Description

IIteratorIndex and IIteratorSupport are used to iterate through the commands supported by an adapter action. Most adapter actions support a single command. TAdapterActionGotoPage is adapter action that supports multiple commands. GetIteratorIndex returns the index of the current command. GetIteratorIndex calls ImplGetIteratorIndex (see page 24065).

See Also

IIteratorIndex (see page 19720)

IIteratorSupport (see page 19722)

GetIteratorIndex (see page 19721)

ImplGetIteratorIndex (see page 24065)

173.1.38.2.24 TBaseAdapterAction.GetSuccessOptions Method

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
function GetSuccessOptions: TAdapterActionRedirectOptions;
```

C++

```
__fastcall TAdapterActionRedirectOptions GetSuccessOptions();
```

Description

Set RedirectOptions (see page 24074) when an HTTP redirect should be performed after executing an action. GetSuccessOptions returns the result of ImplGetSuccessOptions (see page 24066).

See Also

ImplGetSuccessOptions (see page 24066)

GetFailureOptions (see page 24057)

RedirectOptions (see page 24074)

173.1.38.2.25 TBaseAdapterAction.GetVariableName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetVariableName: string;
```

C++

```
__fastcall AnsiString GetVariableName();
```

Description

GetVariableName is called to retrieve the name used to access this adapter action with server-side script. GetVariableName calls ImplGetVariableName (see page 24066).

See Also

ImplGetVariableName (see page 24066)

173.1.38.2.26 TBaseAdapterAction.GetViewMode Method

Returns a string value to suggest a view mode for this adapter action.

Pascal

```
function GetViewMode(const AAdapterMode: string): string;
```

C++

```
__fastcall AnsiString GetViewMode(const AnsiString AAdapterMode);
```

Description

GetViewMode implements IGetHTMLStyle's GetViewMode method by translating the result value of GetViewModeType (see page 24060) into a string.

See Also

GetViewMode (see page 19701)

GetViewModeType (see page 24060)

173.1.38.2.27 TBaseAdapterAction.GetViewModeType Method

Suggests a view mode for this adapter action.

Pascal

```
function GetViewModeType(const AAdapterMode: string): TAdapterDisplayViewModeType; virtual;
```

C++

```
virtual __fastcall TAdapterDisplayViewModeType GetViewModeType(const AnsiString AAdapterMode);
```

Description

GetViewModeType returns htmlvmInput.

See Also

ReadOnly (see page 23924)

GetViewMode (see page 23921)

173.1.38.2.28 TBaseAdapterAction.GetVisible Method

Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.

Pascal

```
function GetVisible: Boolean;
```

C++

```
__fastcall Boolean GetVisible();
```

Description

The state of the parent adapter may dictate whether a button or hyperlink for a particular adapter action should not be shown. For example, TDataSetAdapterApplyRowAction can be configured to return false when the parent TDataSetAdapter is in browse mode. GetVisible calls ImplGetVisible (see page 24066). Server-side script can access the value returned by GetVisible using the Visible property of an adapter action object.

See Also

ImplGetVisible (see page 24066)

173.1.38.2.29 TBaseAdapterAction.GetWebSnapObjClass Method

Returns the class type for this adapter field.

Pascal

```
function GetWebSnapObjClass: TObjectScripterClass; virtual;
```

C++

```
virtual __fastcall TObjectScripterClass GetWebSnapObjClass();
```

Description

This function returns the class type for this adapter field, which is used by the script engine to create a script object. GetWebSnapObjClass tells the script engine what type of object to create.

173.1.38.2.30 TBaseAdapterAction.HandleRequest Method

Executes the adapter action and generates a response.

Pascal

```
function HandleRequest(DispatchParams: IAdapterDispatchParams): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(IAdapterDispatchParams DispatchParams);
```

Description

The adapter dispatcher calls HandleRequest to execute the adapter action and generate a response. HandleRequest calls

ImplHandleRequest (see page 24067).

See Also

CreateRequestContext (see page 24050)

173.1.38.2.31 TBaseAdapterAction.HasExecuteAccess Method

Checks the end-user's right to execute this adapter action.

Pascal

```
function HasExecuteAccess: Boolean;
```

C++

```
__fastcall Boolean HasExecuteAccess();
```

Description

The value returned by this function can be accessed with server-side script using the CanExecute property. HasExecuteAccess calls ImplHasExecuteAccess (see page 24067).

See Also

CheckExecuteAccess (see page 24050)

ImplHasExecuteAccess (see page 24067)

173.1.38.2.32 TBaseAdapterAction.ImplCreateRequestContext Method

Creates an object that describes the action parameters and field values.

Pascal

```
procedure ImplCreateRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall ImplCreateRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

ImplCreateRequestContext is called by CreateRequestContext (see page 24050). ImplCreateRequestContext returns the result of TCustomAdapter (see page 24139)'s CreateActionRequestContext method.

See Also

CreateRequestContext (see page 24050)

CreateActionRequestContext (see page 24151)

173.1.38.2.33 TBaseAdapterAction.ImplEndIterator Method

Implements IIteratorSupport's EndIterator (see page 24053) method.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer);
```

C++

```
__fastcall ImplEndIterator(void * OwnerData);
```

Description

ImplEndIterator is called by EndIterator (see page 24053). ImplEndIterator resets internal variables.

See Also

IIteratorIndex (see page 19720)

IIteratorSupport (see page 19722)

EndIterator (see page 24053)

173.1.38.2.34 TBaseAdapterAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); virtual;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest is called by ExecuteActionRequest (see page 24053). ImplExecuteActionRequest must be overridden by a descendant class.

See Also

IActionRequest (see page 935)

IActionResponse (see page 941)

ExecuteActionRequest (see page 24053)

173.1.38.2.35 TBaseAdapterAction.ImplGetActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function ImplGetActionName: string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplGetActionName();
```

Description

This function is called by GetActionName (see page 24054). ImplGetActionName returns the result of GetDefaultActionName (see page 24055). Descendants may override this function.

See Also

GetActionName (see page 24054)

GetDefaultActionName (see page 24055)

173.1.38.2.36 TBaseAdapterAction.ImplGetAdapterItemRequestParams Method

Provides the name/value pairs needed to execute this adapter action.

Pascal

```
procedure ImplGetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams); virtual;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(AnsiString AIdentifier,
IAdapterItemRequestParams Params);
```

Description

ImplGetAdapterItemRequestParams is called by GetAdapterItemRequestParams (see page 24055). ImplGetAdapterItemRequestParams calls the OnGetActionParams event handler of the parent adapter and then calls the OnGetParams (see page 24077) event handler.

See Also

GetAdapterItemRequestParams (see page 24055)

OnGetActionParams (see page 24187)

OnGetParams (see page 24077)

173.1.38.2.37 TBaseAdapterAction.ImplGetDisplayLabel Method

Returns the suggested display label for this adapter action.

Pascal

```
function ImplGetDisplayLabel: string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplGetDisplayLabel();
```

Description

ImplGetDisplayLabel returns the value of the DisplayLabel (see page 24073) property.

See Also

DisplayLabel (see page 24073)

173.1.38.2.38 TBaseAdapterAction.ImplGetExecuteActionResponse Method

Generates a response after the adapter action has been executed.

Pascal

```
procedure ImplGetExecuteActionResponse(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); virtual;
```

C++

```
virtual __fastcall ImplGetExecuteActionResponse(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplGetExecuteActionResponse is called by GetExecuteActionResponse (see page 24057). ImplGetExecuteActionResponse sets properties of IActionResponse to identify the response to this action request.

See Also

GetExecuteActionResponse (see page 24057)

173.1.38.2.39 TBaseAdapterAction.ImplGetFailureOptions Method

Indicates whether to perform an HTTP redirect after execution of this adapter action fails.

Pascal

```
function ImplGetFailureOptions: TAdapterActionRedirectOptions; virtual;
```

C++

```
virtual __fastcall TAdapterActionRedirectOptions ImplGetFailureOptions();
```

Description

ImplGetFailureOptions returns an empty set.

See Also

GetFailureOptions (see page 24057)

173.1.38.2.40 TBaseAdapterAction.ImplGetIsAdapterActionList Method

Indicates whether this adapter action may have multiple commands.

Pascal

```
function ImplGetIsAdapterActionList: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetIsAdapterActionList();
```

Description

ImplGetIsAdapterActionList is called by GetIsAdapterActionList (see page 24058). ImplGetIsAdapterActionList returns false. ImplGetIsAdapterActionList may be overridden by descendant classes.

See Also

GetIsAdapterActionList (see page 24058)

ImplGetIsAdapterActionList (see page 24243)

173.1.38.2.41 TBaseAdapterAction.ImplGetIteratorIndex Method

Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.

Pascal

```
function ImplGetIteratorIndex: Integer; virtual;
```

C++

```
virtual __fastcall int ImplGetIteratorIndex();
```

Description

ImplGetIteratorIndex is called by GetIteratorIndex (see page 24059). ImplGetIteratorIndex returns the value of an internal iteration index.

See Also

GetIteratorIndex (see page 19721)

GetIteratorIndex (see page 24059)

173.1.38.2.42 TBaseAdapterAction.ImplGetSuccessOptions Method

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
function ImplGetSuccessOptions: TAdapterActionRedirectOptions; virtual;
```

C++

```
virtual __fastcall TAdapterActionRedirectOptions ImplGetSuccessOptions();
```

Description

ImplGetSuccessOptions returns the value of the RedirectOptions (see page 24074) property.

See Also

GetSuccessOptions (see page 24059)

RedirectOptions (see page 24074)

173.1.38.2.43 TBaseAdapterAction.ImplGetVariableName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function ImplGetVariableName: string;
```

C++

```
__fastcall AnsiString ImplGetVariableName();
```

Description

This function is called by GetVariableName (see page 24060). GetVariableName (see page 24060) returns the result of GetActionName (see page 24054).

See Also

GetVariableName (see page 24060)

GetActionName (see page 24054)

173.1.38.2.44 TBaseAdapterAction.ImplGetVisible Method

Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.

Pascal

```
function ImplGetVisible: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetVisible();
```

Description

ImplGetVisible is called by GetVisible (see page 24061). ImplGetVisible returns true. ImplGetVisible may be overridden by descendant components.

See Also

GetVisible (see page 24061)

173.1.38.2.45 TBaseAdapterAction.ImplHandleRequest Method

Executes the adapter action and generates a response.

Pascal

```
function ImplHandleRequest(DispatchParams: IAdapterDispatchParams): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest(IAdapterDispatchParams DispatchParams);
```

Description

ImplHandleRequest is called by HandleRequest (see page 24061). ImplHandleRequest calls ExecuteActionRequest (see page 24053) followed by GetExecuteActionResponse (see page 24057).

See Also

ExecuteActionRequest (see page 24053)

GetExecuteActionResponse (see page 24057)

173.1.38.2.46 TBaseAdapterAction.ImplHasExecuteAccess Method

Checks the end-user's right to execute this adapter action.

Pascal

```
function ImplHasExecuteAccess: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasExecuteAccess();
```

Description

This function is called by HasExecuteAccess (see page 24062) to check the end-user's right to execute the adapter action. First, the parent adapter's HasExecuteAccess (see page 24062) method is called. If the adapter allows execute access and the ExecuteAccess (see page 24073) property is not blank, this function calls the EndUserAdapter HasModifyRights function, passing the ExecuteAccess (see page 24073) property value.

See Also

HasExecuteRights (see page 24111)

ExecuteAccess (see page 24073)

HasExecuteAccess (see page 24062)

hasexecuteaccess (see page 24162)

173.1.38.2.47 TBaseAdapterAction.ImplInIterator Method

Implements IIteratorIndex's InIterator (see page 24069) method.

Pascal

```
function ImplInIterator: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplInIterator();
```

Description

ImplInIterator is called by InIterator (see page 24069). ImplInIterator returns the value of an internal flag.

See Also

IIteratorIndex (see page 19720)

IIteratorSupport (see page 19722)

InIterator (see page 19721)

ImplInIterator

173.1.38.2.48 TBaseAdapterAction.ImplIsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function ImplIsDefaultAction(ADisplay: TObject): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplIsDefaultAction(TObject * ADisplay);
```

Description

ImplIsDefaultAction is called by IsDefaultAction (see page 24070). ImplIsDefaultAction returns true. ImplIsDefaultAction may be overridden by a descendant class.

See Also

IsDefaultAction (see page 24070)

ImplIsDefaultAction (see page 24243)

173.1.38.2.49 TBaseAdapterAction.ImplNextIteration Method

Implements IIteratorSupport's NextIteration (see page 24070) method.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

ImplNextIteration is called by NextIteration (see page 24070). ImplNextIteration increments an internal iteration index. true is returned if the iteration index is within the bounds defined by ItemCount (see page 24074).

See Also

IIteratorIndex (see page 19720)

IIteratorSupport (see page 19722)

NextIteration (see page 24070)

173.1.38.2.50 TBaseAdapterAction.ImplStartIterator Method

Implements IIteratorSupport's StartIterator (see page 24071) method.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

ImplStartIterator is called by StartIterator (see page 24071). ImplStartIterator initializes the internal iteration index. ImplStartIterator returns true because it is assumed that there is at least one command associated with this adapter action.

See Also

IIteratorIndex (see page 19720)

IIteratorSupport (see page 19722)

startiterator (see page 19724)

StartIterator (see page 24071)

173.1.38.2.51 TBaseAdapterAction.ImplWebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function ImplWebEnabled: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

ImplWebEnabled is called by WebEnabled (see page 24071). ImplWebEnabled returns true. Descendant classes may override this method.

See Also

WebEnabled (see page 24071)

173.1.38.2.52 TBaseAdapterAction.InIterator Method

Implements IIteratorIndex's InIterator method.

Pascal

```
function InIterator: Boolean;
```

C++

```
__fastcall Boolean InIterator();
```

Description

IIteratorIndex and IIteratorSupport are used to iterate through the commands supported by an adapter action. Most adapter actions support a single command. TAdapterActionGotoPage is adapter action that supports multiple commands. InIterator calls ImplInIterator (see page 24067).

See Also

IIteratorIndex (see page 19720)

IIteratorSupport (see page 19722)

InIterator (see page 19721)

ImplInIterator (see page 24067)

173.1.38.2.53 TBaseAdapterAction.IsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function IsDefaultAction(ADisplay: TObject): Boolean;
```

C++

```
__fastcall Boolean IsDefaultAction(TObject * ADisplay);
```

Description

Some adapter components have built in support for specific types of adapter actions. These actions are often added to the adapter component by default unless the developer explicitly adds one or more adapter actions. IsDefaultAction is called by the adapter while adding default actions to determine whether this adapter action should be added. ADisplay identifies characteristics of the adapter component. IsDefaultAction calls ImplIsDefaultAction (see page 24068).

See Also

ImplIsDefaultAction (see page 24068)

173.1.38.2.54 TBaseAdapterAction.NextIteration Method

Implements IIteratorSupport's NextIteration method.

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

IIteratorSupport is used to enumerate the list of commands supported by this adapter action. Most adapter actions have a single command. TAdapterGotoPageAction (see page 23849) is an example of an adapter action that supports multiple commands. NextIterations returns the result of ImplNextIteration (see page 24068).

See Also

IIteratorSupport (see page 19722)

IIteratorIndex (see page 19720)

ImplNextIteration (see page 24068)

NextIteration (see page 19723)

173.1.38.2.55 TBaseAdapterAction.ParentChanged Method

Handles movement of the component to a different parent.

Pascal

```
procedure ParentChanged; override;
```

C++

```
virtual __fastcall ParentChanged();
```

Description

ParentChanged is called when the parent component is changed. TBaseAdapterAction (see page 24043) clears its cached reference to the parent adapter.

173.1.38.2.56 TBaseAdapterAction.SetDisplayLabel Method

Set the value of the DisplayLabel (see page 24073) property.

Pascal

```
procedure SetDisplayLabel(const Value: string);
```

C++

```
__fastcall SetDisplayLabel(const AnsiString Value);
```

Description

SetDisplayLabel sets the value of the DisplayLabel (see page 24073) property.

See Also

DisplayLabel (see page 24073)

173.1.38.2.57 TBaseAdapterAction.StartIterator Method

Implements IIteratorSupport's StartIterator method.

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

IIteratorIndex and IIteratorSupport are used to iterate through the commands supported by an adapter action. Most adapter actions support a single command. TAdapterActionGotoPage is adapter action that supports multiple commands. StartIterator calls ImplImplInIterator.

See Also

IIteratorIndex (see page 19720)

IIteratorSupport (see page 19722)

startiterator (see page 19724)

ImplStartIterator (see page 24068)

173.1.38.2.58 TBaseAdapterAction.WebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function WebEnabled: Boolean;
```

C++

```
__fastcall Boolean WebEnabled();
```

Description

The state of the parent adapter may dictate when a button or hyperlink for a particular adapter action should not be enabled. For example, TDataSetAdapterPrevRowAction returns False when the dataset is positioned at BOF. WebEnabled calls ImplWebEnabled (see page 24069). Server-side script can access the value returned by WebEnabled using the Enabled property of an adapter action object.

See Also

ImplWebEnabled (see page 24069)

173.1.38.3 TBaseAdapterAction Properties

The properties of the TBaseAdapterAction class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.

Public Properties

	Name	Description
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	DisplayLabel (see page 24073)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 24073)	Defines the rights required to execute this action.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
	RedirectOptions (see page 24074)	Indicates whether to perform an HTTP redirect after successfully executing this action.

173.1.38.3.1 TBaseAdapterAction.ActionName Property

Identifies the name of this action when using server-side script.

Pascal

```
property ActionName: string;
```

C++

```
__property AnsiString ActionName;
```

Description

The ActionName property defines the name used to access the adapter action with server-side script.

See Also

GetActionName (see page 24054)

173.1.38.3.2 TBaseAdapterAction.Adapter Property

Provides access to the parent adapter component.

Pascal

```
property Adapter: TCustomAdapter;
```

C++

```
__property TCustomAdapter Adapter;
```

Description

Descendant classes use this property to access the parent adapter component.

173.1.38.3.3 TBaseAdapterAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

173.1.38.3.4 TBaseAdapterAction.Errors Property

Pascal

```
property Errors: TBaseAdapterErrorsList;
```

C++

```
__property TBaseAdapterErrorsList Errors;
```

Description

This is Errors, a member of class TBaseAdapterAction.

173.1.38.3.5 TBaseAdapterAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property AnsiString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user

attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 25625)

hasexecuteaccess (see page 24162)

173.1.38.3.6 TBaseAdapterAction.ItemCount Property

Returns the number of items defined by an adapter action.

Pascal

```
property ItemCount: Integer;
```

C++

```
__property int ItemCount;
```

Description

An adapter action may have a list of commands. The ItemCount property is implemented by GetItemCount (see page 24058). The TAdapterGotoPageAction (see page 23849) is an example of an adapter action with a list of commands. Each command of TAdapterGotoPageAction (see page 23849) displays a different page of the parent TPagedAdapter (see page 24525).

See Also

GetItemCount (see page 24058)

ItemIndex (see page 24074)

TAdapterGotoPageAction (see page 23849)

GetIsAdapterActionList (see page 24058)

173.1.38.3.7 TBaseAdapterAction.ItemIndex Property

Returns the index of the current item of an adapter action.

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

Description

An adapter action may have a list of commands. The ItemIndex indicates the current item in the list. The TAdapterGotoPageAction (see page 23849) is an example of an adapter action with a list of commands. Each command of TAdapterGotoPageAction (see page 23849) displays a different page of the parent TPagedAdapter (see page 24525).

See Also

ItemCount (see page 24074)

TAdapterGotoPageAction (see page 23849)

GetIsAdapterActionList (see page 24058)

173.1.38.3.8 TBaseAdapterAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect (see page 15201)

173.1.38.4 TBaseAdapterAction Events

The events of the TBaseAdapterAction class are listed here.

Public Events

	Name	Description
	OnAfterExecute (see page 24075)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 24075)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24076)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24076)	Called before the adapter action response is generated.
	OnGetParams (see page 24077)	Specifies name/value pairs needed to execute this adapter action.

173.1.38.4.1 TBaseAdapterAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest (see page 24051).

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

173.1.38.4.2 TBaseAdapterAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse

OnBeforeGetActionResponse (see page 24186)

173.1.38.4.3 TBaseAdapterAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest (see page 24052).

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

173.1.38.4.4 TBaseAdapterAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.38.4.5 TBaseAdapterAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

173.1.39 TBaseAdapterErrorsList Class

TBaseAdapterErrorsList is the base class for adapter error object managers.

Class Hierarchy

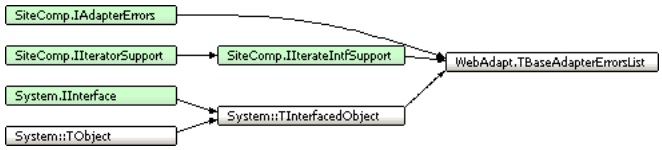

Pascal

```
TBaseAdapterErrorsList = class(TInterfacedObject, IAdapterErrors, IIterateIntfSupport);
```

C++

```
class TBaseAdapterErrorsList : public TInterfacedObject, public IAdapterErrors, public
IIterateIntfSupport;
```

File

WebAdapt

Description

TBaseAdapterErrorsList is the base class for objects that manage lists of adapter error objects. Do not instantiate TBaseAdapterErrorsList directly. Use TAdapterErrorsList (see page 23810).

See Also

TAdapterErrorsList (see page 23810)

IAdapterError (see page 19660)

173.1.39.1 TBaseAdapterErrorsList Members

The following tables list the members exposed by TBaseAdapterErrorsList.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	EndIterator (see page 24081)	This is EndIterator, a member of class TBaseAdapterErrorsList.
	GetCurrentIntf (see page 24081)	This is GetCurrentIntf, a member of class TBaseAdapterErrorsList.
	GetError (see page 24082)	Returns a particular error.
	GetErrorCount (see page 24082)	Return the number of errors in the errors list.
	GetObjectErrors (see page 24082)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 24082)	Indicates whether the errors list contains objects associated with a particular adapter field.
V	ImplClearErrors (see page 24083)	This is ImplClearErrors, a member of class TBaseAdapterErrorsList.
V	ImplDefineLabel (see page 24083)	This is ImplDefineLabel, a member of class TBaseAdapterErrorsList.
V	ImplGetError (see page 24083)	This is ImplGetError, a member of class TBaseAdapterErrorsList.
V	ImplGetErrorCount (see page 24083)	This is ImplGetErrorCount, a member of class TBaseAdapterErrorsList.
V	ImplGetObjectErrors (see page 24084)	This is ImplGetObjectErrors, a member of class TBaseAdapterErrorsList.
V	ImplHasObjectErrors (see page 24084)	This is ImplHasObjectErrors, a member of class TBaseAdapterErrorsList.
	NextIteration (see page 24084)	Advances to the objects next item.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	StartIterator (see page 24085)	This is StartIterator, a member of class TBaseAdapterErrorsList.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AddError (see page 24080)	This is the overview for the AddError method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearErrors (see page 24080)	Clear error list.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

	DefineLabel (see page 24081)	Define error label for field.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ErrorCount (see page 24085)	Number of error objects contained.
R	Errors (see page 24085)	Access to error objects.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

173.1.39.2 TBaseAdapterErrorsList Methods

The methods of the TBaseAdapterErrorsList class are listed here.

Protected Methods

	Name	Description
	EndIterator (see page 24081)	This is EndIterator, a member of class TBaseAdapterErrorsList.
	GetCurrentIntf (see page 24081)	This is GetCurrentIntf, a member of class TBaseAdapterErrorsList.
	GetError (see page 24082)	Returns a particular error.
	GetErrorCount (see page 24082)	Return the number of errors in the errors list.
	GetObjectErrors (see page 24082)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 24082)	Indicates whether the errors list contains objects associated with a particular adapter field.
V	ImplClearErrors (see page 24083)	This is ImplClearErrors, a member of class TBaseAdapterErrorsList.
V	ImplDefineLabel (see page 24083)	This is ImplDefineLabel, a member of class TBaseAdapterErrorsList.
V	ImplGetError (see page 24083)	This is ImplGetError, a member of class TBaseAdapterErrorsList.
V	ImplGetErrorCount (see page 24083)	This is ImplGetErrorCount, a member of class TBaseAdapterErrorsList.
V	ImplGetObjectErrors (see page 24084)	This is ImplGetObjectErrors, a member of class TBaseAdapterErrorsList.
V	ImplHasObjectErrors (see page 24084)	This is ImplHasObjectErrors, a member of class TBaseAdapterErrorsList.
	NextIteration (see page 24084)	Advances to the objects next item.
	StartIterator (see page 24085)	This is StartIterator, a member of class TBaseAdapterErrorsList.

Public Methods

	Name	Description
	AddError (see page 24080)	This is the overview for the AddError method overload.
	ClearErrors (see page 24080)	Clear error list.
	DefineLabel (see page 24081)	Define error label for field.

173.1.39.2.1 AddError Method

This is the overview for the AddError method overload.

Overload List

	Name	Description
	TBaseAdapterErrorsList.AddError (Exception, string) (see page 24080)	Add error to list.
	TBaseAdapterErrorsList.AddError (Exception, TObject) (see page 24080)	Add error to list.
	TBaseAdapterErrorsList.AddError (string, string, Integer) (see page 24080)	Add error to list.
	TBaseAdapterErrorsList.AddError (string, TObject, Integer) (see page 24080)	Add error to list.

173.1.39.2.1.1 TBaseAdapterErrorsList.AddError Method (Exception, TObject)

Add error to list.

Pascal

```
procedure AddError(AException: Exception; const AObjectName: string = ''); virtual; overload;
procedure AddError(const AMessage: string; const AObjectName: string = ''; AID: Integer = 0);
virtual; overload;
procedure AddError(const AMessage: string; AObject: TObject; AID: Integer = 0); virtual;
overload;
procedure AddError(AException: Exception; AObject: TObject); virtual; overload;
```

C++

```
virtual __fastcall AddError(Exception AException, const AnsiString AObjectName = '');
virtual __fastcall AddError(const AnsiString AMessage, const AnsiString AObjectName = '', int
AID = 0);
virtual __fastcall AddError(const AnsiString AMessage, TObject * AObject, int AID = 0);
virtual __fastcall AddError(Exception AException, TObject * AObject);
```

Description

AddError adds an error to the list maintained by the TBaseAdapterErrorsList (see page 24077) instance. As implemented in TBaseAdapterErrorsList (see page 24077), AddError does nothing. This method must be implemented by descendant classes.

See Also

AddError (see page 23815)

173.1.39.2.2 TBaseAdapterErrorsList.ClearErrors Method

Clear error list.

Pascal

```
procedure ClearErrors;
```

C++

```
__fastcall ClearErrors();
```

Description

Clear errors deletes all error information maintained by the TBaseAdapterErrorsList (see page 24077) instance.

See Also

AddError (see page 23815)

173.1.39.2.3 TBaseAdapterErrorsList.DefineLabel Method

Define error label for field.

Pascal

```
procedure DefineLabel(const AName: string; const ALabel: string);
```

C++

```
__fastcall DefineLabel(const AnsiString AName, const AnsiString ALabel);
```

Description

DefineLabel defines a string label that represents a dataset field in error message. AName is the the field name. ALabel is the label.

See Also

AddError (see page 23815)

173.1.39.2.4 TBaseAdapterErrorsList.EndIterator Method

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

This is EndIterator, a member of class TBaseAdapterErrorsList.

173.1.39.2.5 TBaseAdapterErrorsList.GetCurrentIntf Method

Pascal

```
function GetCurrentIntf(OwnerData: Pointer): IUnknown;
```

C++

```
__fastcall IUnknown GetCurrentIntf(void * OwnerData);
```

Description

This is GetCurrentIntf, a member of class TBaseAdapterErrorsList.

173.1.39.2.6 TBaseAdapterErrorsList.GetError Method

Returns a particular error.

Pascal

```
function GetError(I: Integer): IAdapterError;
```

C++

```
__fastcall IAdapterError GetError(int I);
```

Description

I is a zero-based index.

See Also

IAdapterError (see page 19660)

173.1.39.2.7 TBaseAdapterErrorsList.GetErrorCount Method

Return the number of errors in the errors list.

Pascal

```
function GetErrorCount: Integer;
```

C++

```
__fastcall int GetErrorCount();
```

Description

Return the number of errors in the errors list.

173.1.39.2.8 TBaseAdapterErrorsList.GetObjectErrors Method

Returns an object that is used to enumerate all errors associated with an adapter field.

Pascal

```
function GetObjectErrors(const AName: string): TObject;
```

C++

```
__fastcall TObject * GetObjectErrors(const AnsiString AName);
```

Description

AName identifies an adapter field. nil (Delphi) or NULL (C++) is returned if there are no errors associated with the adapter field.

See Also

HasObjectErrors (see page 19665)

173.1.39.2.9 TBaseAdapterErrorsList.HasObjectErrors Method

Indicates whether the errors list contains objects associated with a particular adapter field.

Pascal

```
function HasObjectErrors(const AName: string): Boolean;
```

C++

```
__fastcall Boolean HasObjectErrors(const AnsiString AName);
```

Description

HasObjectErrors returns true if any errors are associated with the given adapter field. AName identifies an adapter field.

See Also

GetObjectErrors (see page 19665)

173.1.39.2.10 TBaseAdapterErrorsList.ImplClearErrors Method

Pascal

```
procedure ImplClearErrors; virtual;
```

C++

```
virtual __fastcall ImplClearErrors();
```

Description

This is ImplClearErrors, a member of class TBaseAdapterErrorsList.

173.1.39.2.11 TBaseAdapterErrorsList.ImplDefineLabel Method

Pascal

```
procedure ImplDefineLabel(const AName: string; const ALabel: string); virtual;
```

C++

```
virtual __fastcall ImplDefineLabel(const AnsiString AName, const AnsiString ALabel);
```

Description

This is ImplDefineLabel, a member of class TBaseAdapterErrorsList.

173.1.39.2.12 TBaseAdapterErrorsList.ImplGetError Method

Pascal

```
function ImplGetError(I: Integer): IAdapterError; virtual;
```

C++

```
virtual __fastcall IAdapterError ImplGetError(int I);
```

Description

This is ImplGetError, a member of class TBaseAdapterErrorsList.

173.1.39.2.13 TBaseAdapterErrorsList.ImplGetErrorCount Method

Pascal

```
function ImplGetErrorCount: Integer; virtual;
```

C++

```
virtual __fastcall int ImplGetErrorCount();
```

Description

This is ImplGetErrorCount, a member of class TBaseAdapterErrorsList.

173.1.39.2.14 TBaseAdapterErrorsList.ImplGetObjectErrors Method

Pascal

```
function ImplGetObjectErrors(const AName: string): TObject; virtual;
```

C++

```
virtual __fastcall TObject * ImplGetObjectErrors(const AnsiString AName);
```

Description

This is ImplGetObjectErrors, a member of class TBaseAdapterErrorsList.

173.1.39.2.15 TBaseAdapterErrorsList.ImplHasObjectErrors Method

Pascal

```
function ImplHasObjectErrors(const AName: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasObjectErrors(const AnsiString AName);
```

Description

This is ImplHasObjectErrors, a member of class TBaseAdapterErrorsList.

173.1.39.2.16 TBaseAdapterErrorsList.NextIteration Method

Advances to the objects next item.

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

Server-side script calls NextIteration to move to the next item.

OwnerData is a pointer to information that the object can use to influence the iteration process. Each object stores and uses this information in its own way. This information is first supplied by the initial call to the StartIterator method. NextIteration can modify the information, after which it is supplied to the next call to the NextIteration or EndIterator method.

NextIteration returns true if it successfully moves to the next item, false if there were no more items.

See Also

startiterator (see page 19724)

enditerator (see page 19723)

173.1.39.2.17 TBaseAdapterErrorsList.StartIterator Method

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

This is StartIterator, a member of class TBaseAdapterErrorsList.

173.1.39.3 TBaseAdapterErrorsList Properties

The properties of the TBaseAdapterErrorsList class are listed here.

Public Properties

	Name	Description
R	ErrorCount (see page 24085)	Number of error objects contained.
R	Errors (see page 24085)	Access to error objects.

173.1.39.3.1 TBaseAdapterErrorsList.ErrorCount Property

Number of error objects contained.

Pascal

```
property ErrorCount: Integer;
```

C++

```
__property int ErrorCount;
```

Description

ErrorCount returns the number of error objects available.

See Also

Errors (see page 24085)

173.1.39.3.2 TBaseAdapterErrorsList.Errors Property

Access to error objects.

Pascal

```
property Errors [I: Integer]: IAdapterError;
```

C++

```
__property IAdapterError Errors[int I];
```

Description

Errors returns a specific error object.

See Also

ErrorCount (see page 24085)

173.1.40 TBaseAdapterMultiValueField Class

TBaseAdapterMultiValueField is a base class for adapter fields that have zero or more values.

Class Hierarchy

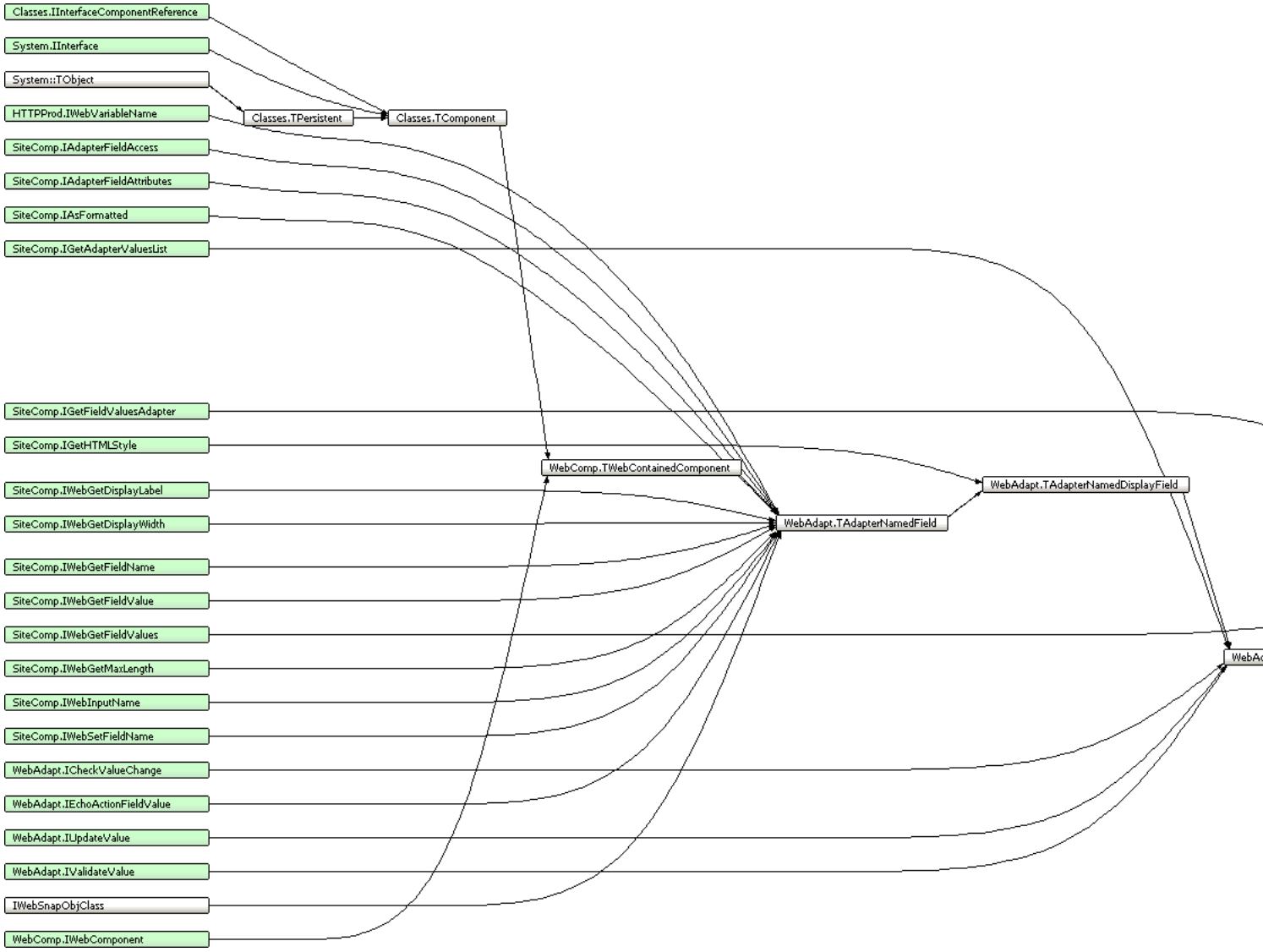

Pascal

```
TBaseAdapterMultiValueField = class(TAdapterUpdateField, IGetFieldValuesAdapter,
IWebGetFieldValues);
```

C++

```
class TBaseAdapterMultiValueField : public TAdapterUpdateField, public IGetFieldValuesAdapter,
public IWebGetFieldValues;
```

File

WebAdapt

Description

TBaseAdapterMultiValueField is used to represent a data value that can have multiple values. Server-side script is used to generate a form displaying this field. Typically, the HTML form will contain group check boxes or a select list supporting multiple selection. When an Adapter (see page 23943) action is executed, TBaseAdapterMultiValueField is used to retrieve the submitted field value(s) from the HTTP request.

See Also

TCustomAdapterMultiValueField (see page 24273)

TCustomAdapterFileField (see page 24220)

173.1.40.1 TBaseAdapterMultiValueField Members

The following tables list the members exposed by TBaseAdapterMultiValueField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 24015)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 24015)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 24015)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 24016)	Retrieves a value entered by the end-user.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoOnValidateValue (see page 24016)	Validates field values entered by the end-user.
	EndValueIterator (see page 24093)	Completes iteration.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 24016)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetCurrentValue (see page 24094)	Implements IWebGetFieldValues' GetCurrentValue.

	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetFieldValuesAdapter (see page 24094)	Returns the adapter object that implements the Values property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 23921)	Suggests an HTML input type for this adapter field.
	GetInputStyleType (see page 24017)	Indicates the preferred HTML element to use when editing this adapter field's value.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetValueCount (see page 24094)	Returns the number of values associated with this adapter field.
	GetValues (see page 24095)	Returns one of the values associated with the adapter field.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasValue (see page 24095)	Implements IWebGetFieldValues' HasValue method.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 24018)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 24018)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplGetCurrentValue (see page 24095)	Implements IWebGetFieldValues' GetCurrentValue (see page 24094) method.
	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 24018)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 24019)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 24019)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetFieldValuesAdapter (see page 24096)	Returns the adapter object that implements the Values property.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.

	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 24019)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasValue (see page 24096)	Implements IWebGetFieldValues' HasValue (see page 24095) method.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 24020)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 24020)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 24020)	Updates the adapter field with value(s) entered by the end-user.
	ImplValidateValue (see page 24020)	Validates values(s) entered by the end-user.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 24021)	Indicates whether this adapter field can support multiple values.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextValueIteration (see page 24096)	Implements IWebGetFieldValues' NextValueIteration method.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24021)	Clears references to a deleted component.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartValueIterator (see page 24097)	Implements IWebGetFieldValues' StartValueIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 24021)	Updates the adapter field with value(s) entered by the end-user.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.

	ValidateValue (see page 24022)	Validates values(s) entered by the end-user.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 24023)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 24024)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 24092)	Constructor for TBaseAdapterMultiValueField (see page 24086).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 24093)	Destructor for TBaseAdapterMultiValueField (see page 24086).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterValuesList (see page 19695)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.

	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.

R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 23923)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23924)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 23924)	Indicates whether this adapter field is read-only.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	Value (see page 24022)	Retrieves the value associated with this adapter field.
	ValuesList (see page 24023)	Associates a list of values with the adapter field.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.40.2 TBaseAdapterMultiValueField.Create Constructor

Constructor for TBaseAdapterMultiValueField (see page 24086).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBaseAdapterMultiValueField(TComponent * AOwner);
```

Description

Creates an instance of TFieldValuesAdapter (see page 24471) and calls the inherited creator method. TFieldValuesAdapter (see page 24471) is used to implement the Values property used by server-side script.

See Also

TFieldValuesAdapter (see page 24471)

ImplGetFieldValuesAdapter (see page 24096)

Destroy (see page 24093)

173.1.40.3 TBaseAdapterMultiValueField.Destroy Destructor

Destructor for TBaseAdapterMultiValueField (see page 24086).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBaseAdapterMultiValueField();
```

Description

Destroys the instance of TFieldValuesAdapter (see page 24471) and calls the inherited destructor. TFieldValuesAdapter (see page 24471) is used to implement the Values property used by server-side script.

See Also

TFieldValuesAdapter (see page 24471)

ImplGetFieldValuesAdapter (see page 24096)

Create (see page 24092)

173.1.40.4 TBaseAdapterMultiValueField Methods

The methods of the TBaseAdapterMultiValueField class are listed here.

Protected Methods

	Name	Description
	EndValueIterator (see page 24093)	Completes iteration.
	GetCurrentValue (see page 24094)	Implements IWebGetFieldValues' GetCurrentValue.
	GetFieldValuesAdapter (see page 24094)	Returns the adapter object that implements the Values property.
V	GetValueCount (see page 24094)	Returns the number of values associated with this adapter field.
V	GetValues (see page 24095)	Returns one of the values associated with the adapter field.
	HasValue (see page 24095)	Implements IWebGetFieldValues' HasValue method.
V	ImplGetCurrentValue (see page 24095)	Implements IWebGetFieldValues' GetCurrentValue (see page 24094) method.
V	ImplGetFieldValuesAdapter (see page 24096)	Returns the adapter object that implements the Values property.
V	ImplHasValue (see page 24096)	Implements IWebGetFieldValues' HasValue (see page 24095) method.
	NextValueIteration (see page 24096)	Implements IWebGetFieldValues' NextValueIteration method.
	StartValueIterator (see page 24097)	Implements IWebGetFieldValues' StartValueIterator method.

173.1.40.4.1 TBaseAdapterMultiValueField.EndValueIterator Method

Completes iteration.

Pascal

```
procedure EndValueIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndValueIterator(void * OwnerData);
```

Description

The EndValueIterator procedure is called when iteration has completed.

See Also

NextValueIteration (see page 24096)

StartValueIterator (see page 24097)

173.1.40.4.2 TBaseAdapterMultiValueField.GetCurrentValue Method

Implements IWebGetFieldValues' GetCurrentValue.

Pascal

```
function GetCurrentValue: Variant;
```

C++

```
__fastcall Variant GetCurrentValue();
```

Description

GetCurrentValue returns the result of ImplGetCurrentValue (see page 24095).

See Also

GetCurrentValue (see page 19761)

ImplGetCurrentValue (see page 24095)

173.1.40.4.3 TBaseAdapterMultiValueField.GetFieldValuesAdapter Method

Returns the adapter object that implements the Values property.

Pascal

```
function GetFieldValuesAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetFieldValuesAdapter();
```

Description

GetFieldValuesAdapter is called when server-side script references the Values property of this adapter field. GetFieldValuesAdapter calls ImplGetFieldValuesAdapter (see page 24096).

See Also

ImplGetFieldValuesAdapter (see page 24096)

173.1.40.4.4 TBaseAdapterMultiValueField.GetValueCount Method

Returns the number of values associated with this adapter field.

Pascal

```
function GetValueCount: Integer; override;
```

C++

```
virtual __fastcall int GetValueCount();
```

Description

GetValueCount returns 0. Descendants of TBaseAdapterMultiValueField (see page 24086) override this method. This method will be called when server-side script is used to iterate over field values using the Values property.

See Also

GetValues (see page 24095)

173.1.40.4.5 TBaseAdapterMultiValueField.GetValues Method

Returns one of the values associated with the adapter field.

Pascal

```
function GetValues(I: Integer): Variant; override;
```

C++

```
virtual __fastcall Variant GetValues(int I);
```

Description

GetValues returns Unassigned. Descendants override this method. This method will be called when server-side script is used to iterate over field values using the Values property.

See Also

GetValueCount (see page 24094)

173.1.40.4.6 TBaseAdapterMultiValueField.HasValue Method

Implements IWebGetFieldValues' HasValue method.

Pascal

```
function HasValue(const AValue: Variant): Boolean;
```

C++

```
__fastcall Boolean HasValue(const Variant AValue);
```

Description

HasValue returns the result of ImplHasValue (see page 24096).

See Also

HasValue (see page 19761)

ImplHasValue (see page 24096)

173.1.40.4.7 TBaseAdapterMultiValueField.ImplGetCurrentValue Method

Implements IWebGetFieldValues' GetCurrentValue (see page 24094) method.

Pascal

```
function ImplGetCurrentValue: Variant; virtual;
```

C++

```
virtual __fastcall Variant ImplGetCurrentValue();
```

Description

ImplGetCurrentValue is called by GetCurrentValue (see page 24094). ImplGetCurrentValue calls GetValues (see page 24095) with the value of the internal iterator index.

See Also

GetCurrentValue (see page 24094)

GetValues (see page 24095)

173.1.40.4.8 TBaseAdapterMultiValueField.ImplGetFieldValuesAdapter Method

Returns the adapter object that implements the Values property.

Pascal

```
function ImplGetFieldValuesAdapter: TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplGetFieldValuesAdapter();
```

Description

ImplGetFieldValuesAdapter returns an instance of TFieldValuesAdapter (see page 24471).

See Also

TFieldValuesAdapter (see page 24471)

GetFieldValuesAdapter (see page 24094)

Create (see page 24092)

Destroy (see page 24093)

173.1.40.4.9 TBaseAdapterMultiValueField.ImplHasValue Method

Implements IWebGetFieldValues' HasValue (see page 24095) method.

Pascal

```
function ImplHasValue(const Value: Variant): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasValue(const Variant Value);
```

Description

ImplHasValue is called by HasValue (see page 24095). ImplHasValue returns False (indicating that there is no matching value). Descendants override this method.

See Also

HasValue (see page 24095)

173.1.40.4.10 TBaseAdapterMultiValueField.NextValueIteration Method

Implements IWebGetFieldValues' NextValueIteration method.

Pascal

```
function NextValueIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextValueIteration(void * OwnerData);
```

Description

NextValueIteration increments an internal iterator index. false is returned if the iterator index is not less than the GetValueCount (see page 24094) .

See Also

NextValueIteration (see page 19762)

GetValueCount (see page 24094)

173.1.40.4.11 TBaseAdapterMultiValueField.StartValueIterator Method

Implements IWebGetFieldValues' StartValueIterator method.

Pascal

```
function StartValueIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartValueIterator(void * OwnerData);
```

Description

StartValueIterator initializes an internal iterator index to zero. false is returned if GetValueCount (see page 24094) is zero.

See Also

StartValueIterator (see page 19762)

GetValueCount (see page 24094)

173.1.41 TBaseEndUserAdapter Class

TBaseEndUserAdapter contains user information.

Class Hierarchy

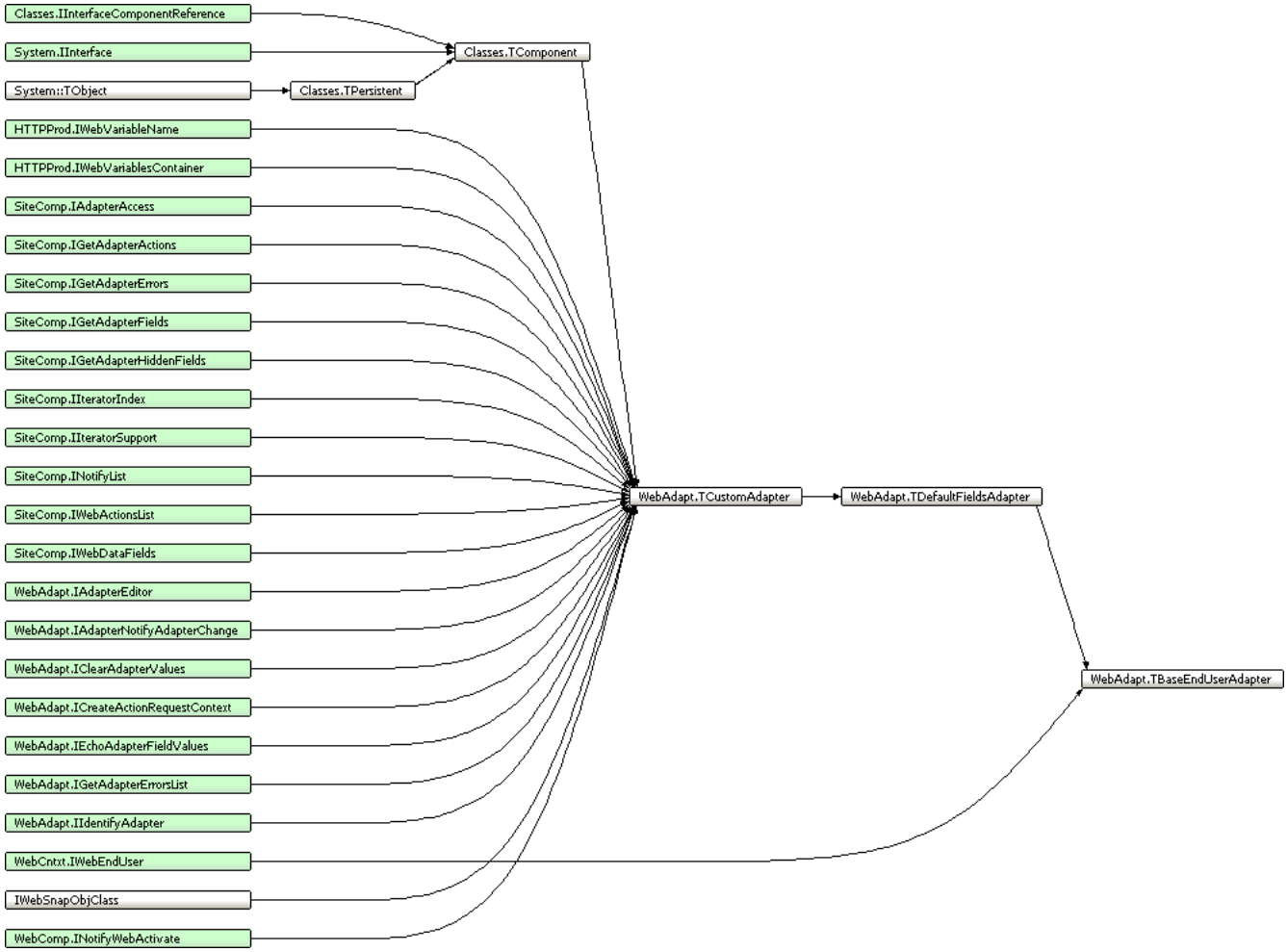

Pascal

```
TBaseEndUserAdapter = class(TDefaultFieldsAdapter, IWebEndUser);
```

C++

```
class TBaseEndUserAdapter : public TDefaultFieldsAdapter, public IWebEndUser;
```

File

WebAdapt

Description

TBaseEndUserAdapter contains information about a user such as their name, access rights, and whether they are logged in. TBaseEndUserAdapter has default fields to represent the end user's display name and login state. It has default actions for logging a user in and logging a user out. End user adapters are responsible validating end user access. For example, when the end user attempts to access a page, the end-user adapter is called to determine whether the end user has rights to access the page.

See Also

TEndUserAdapter (see page 24421)

TEndUserSessionAdapter (see page 24458)

173.1.41.1 TBaseEndUserAdapter Members

The following tables list the members exposed by TBaseEndUserAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
V	CanExecuteLoginForm (see page 24106)	Determines if there is a static login page.
V	CanExecuteLogout (see page 24107)	Determines if the user is logged in.
V	CanProduceLoginPage (see page 24107)	Determines whether there is a static login page or an automatically produced login page.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoOnHasRights (see page 24107)	Determines whether the user has view rights.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
V	ExecuteLogin (see page 24108)	Starts the server-side login process.
V	ExecuteLoginForm (see page 24108)	Creates the HTML login page.
V	ExecuteLogout (see page 24108)	Performs server-side logout.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.

	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetDisplayName (see page 24109)	Retrieves the display name string.
	GetDisplayNameValue (see page 24109)	Gets the display name of the end user.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetEndUserAdapter (see page 24109)	Gets the end-user adapter.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetLoggedIn (see page 24110)	Determines if the user is logged in
	GetLoggedInValue (see page 24110)	Determines if the user is logged in.
	GetLoginFormAction (see page 24110)	Gets the action component for the login form.
	GetLogoutAction (see page 24111)	Gets the action component for the logout form.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetUserID (see page 24111)	Gets the UserID from this TBaseEndUserAdapter (see page 24097).
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasExecuteRights (see page 24111)	Determines if the user has execute rights for the TBaseEndUserAdapter (see page 24097).
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasModifyRights (see page 24112)	Determines if the user has modify rights for the TBaseEndUserAdapter (see page 24097).
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
	HasViewRights (see page 24112)	Determine if the user has view rights to the TBaseEndUserAdapter (see page 24097).
	ImplCanAddActionClass (see page 24113)	Determines if the given action class can be added.
	ImplCanAddFieldClass (see page 24113)	Determines if the given action class can be added.

	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 24113)	Lists supported end-user adapter actions.
	ImplGetDisplayName (see page 24114)	Gets the display name for this TBaseEndUserAdapter (see page 24097).
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetEndUserAdapter (see page 24114)	Returns this TBaseEndUserAdapter (see page 24097).
	ImplGetFieldsList (see page 24114)	Obtains the list of all of the fields for the end-user adapter.
	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetLoggedIn (see page 24114)	Checks the value of the LoggedIn field.
	ImplGetLoginFormAction (see page 24115)	Gets the LoginForm action component for this TBaseEndUserAdapter (see page 24097).
	ImplGetLogoutAction (see page 24115)	Gets the Logout action component for this TBaseEndUserAdapter (see page 24097).
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasExecuteRights (see page 24115)	Checks user rights and determines if the user has execute rights.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasModifyRights (see page 24116)	Checks user rights and determines if the user has modify rights.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplHasViewRights (see page 24116)	Checks user rights and determines if the user has view rights.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplLogIn (see page 24116)	Processes the login for the given user.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplProduceLoginPage (see page 24117)	Generate the HTML login page.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	LogIn (see page 24117)	Logs in a specified user.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.

V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceLoginPage (see page 24117)	Generates the login page.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UserList (see page 24118)	Gets the user list.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 24184)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24185)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.

	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24186)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24186)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 24187)	Supplies name/value pairs needed to execute an adapter action.
	OnHasRights (see page 24119)	Determines if the user rights match the rights being queried.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.
	OnProduceLoginPage (see page 24119)	Allows implementation of a customized login page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.

V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 24181)	Actions is the container for adapter actions components.
R	Actions (see page 24402)	Container for adapter actions components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Data (see page 24181)	Data is the container for adapter field components.
R	Data (see page 24402)	Container for adapter field components.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 24678)	Indicates the name of the end user as it should be used for display in Web pages.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	EndUserAdapter (see page 24678)	Provides access to the end user adapter that implements this interface.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	LoggedIn (see page 24678)	Indicates whether the current user is logged in.
R	LoginFormAction (see page 24679)	Indicates the action component for the login form.
	LoginPage (see page 24118)	Specifies the name of the page that acts as the login page.
R	LogoutAction (see page 24680)	Indicates the action component for logging a user out.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.41.2 TBaseEndUserAdapter Methods

The methods of the TBaseEndUserAdapter class are listed here.

Protected Methods

	Name	Description
	CanExecuteLoginForm (see page 24106)	Determines if there is a static login page.
	CanExecuteLogout (see page 24107)	Determines if the user is logged in.
	CanProduceLoginPage (see page 24107)	Determines whether there is a static login page or an automatically produced login page.
	DoOnHasRights (see page 24107)	Determines whether the user has view rights.
	ExecuteLogin (see page 24108)	Starts the server-side login process.
	ExecuteLoginForm (see page 24108)	Creates the HTML login page.
	ExecuteLogout (see page 24108)	Performs server-side logout.
	GetDisplayName (see page 24109)	Retrieves the display name string.
	GetDisplayNameValue (see page 24109)	Gets the display name of the end user.
	GetEndUserAdapter (see page 24109)	Gets the end-user adapter.
	GetLoggedIn (see page 24110)	Determines if the user is logged in
	GetLoggedInValue (see page 24110)	Determines if the user is logged in.
	GetLoginFormAction (see page 24110)	Gets the action component for the login form.
	GetLogoutAction (see page 24111)	Gets the action component for the logout form.
	GetUserID (see page 24111)	Gets the UserID from this TBaseEndUserAdapter (see page 24097).
	HasExecuteRights (see page 24111)	Determines if the user has execute rights for the TBaseEndUserAdapter (see page 24097).
	HasModifyRights (see page 24112)	Determines if the user has modify rights for the TBaseEndUserAdapter (see page 24097).
	HasViewRights (see page 24112)	Determine if the user has view rights to the TBaseEndUserAdapter (see page 24097).
	ImplCanAddActionClass (see page 24113)	Determines if the given action class can be added.
	ImplCanAddFieldClass (see page 24113)	Determines if the given action class can be added.
	ImplGetActionsList (see page 24113)	Lists supported end-user adapter actions.
	ImplGetDisplayName (see page 24114)	Gets the display name for this TBaseEndUserAdapter (see page 24097).
	ImplGetEndUserAdapter (see page 24114)	Returns this TBaseEndUserAdapter (see page 24097).
	ImplGetFieldsList (see page 24114)	Obtains the list of all of the fields for the end-user adapter.
	ImplGetLoggedIn (see page 24114)	Checks the value of the LoggedIn field.
	ImplGetLoginFormAction (see page 24115)	Gets the LoginForm action component for this TBaseEndUserAdapter (see page 24097).
	ImplGetLogoutAction (see page 24115)	Gets the Logout action component for this TBaseEndUserAdapter (see page 24097).
	ImplHasExecuteRights (see page 24115)	Checks user rights and determines if the user has execute rights.
	ImplHasModifyRights (see page 24116)	Checks user rights and determines if the user has modify rights.
	ImplHasViewRights (see page 24116)	Checks user rights and determines if the user has view rights.
	ImplLogIn (see page 24116)	Processes the login for the given user.
	ImplProduceLoginPage (see page 24117)	Generate the HTML login page.
	LogIn (see page 24117)	Logs in a specified user.
	ProduceLoginPage (see page 24117)	Generates the login page.
	UserList (see page 24118)	Gets the user list.

173.1.41.2.1 TBaseEndUserAdapter.CanExecuteLoginForm Method

Determines if there is a static login page.

Pascal

```
function CanExecuteLoginForm: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanExecuteLoginForm();
```

Description

The CanExecuteLoginForm calls CanProduceLoginPage (see page 24107) to determine if there is a static login page, or if one

can be generated. Then if the user is not already logged in, this function returns true. If the login page cannot be produced or the user is already logged in, then this function returns false.

See Also

CanProduceLoginPage (see page 24107)

173.1.41.2.2 TBaseEndUserAdapter.CanExecuteLogout Method

Determines if the user is logged in.

Pascal

```
function CanExecuteLogout: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanExecuteLogout();
```

Description

The CanExecuteLogout function calls ImplGetLoggedIn (see page 24114) to determine if the user is logged in. If the user is logged in, this function returns true.

See Also

ImplGetLoggedIn (see page 24114)

173.1.41.2.3 TBaseEndUserAdapter.CanProduceLoginPage Method

Determines whether there is a static login page or an automatically produced login page.

Pascal

```
function CanProduceLoginPage: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanProduceLoginPage();
```

Description

The CanProduceLoginPage returns true if there is a static login page, or if there is an event handler defined to automatically produce the login page.

See Also

OnProduceLoginPage (see page 24119)

173.1.41.2.4 TBaseEndUserAdapter.DoOnHasRights Method

Determines whether the user has view rights.

Pascal

```
function DoOnHasRights(AEvent: TEndUserRightsEvent; const ARights: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoOnHasRights(TEndUserRightsEvent AEvent, const AnsiString ARights);
```

Description

The DoOnHasRights function is called from ImplHasViewRights (see page 24116). It matches the given access rights to the rights in the user list and returns true if the user has view rights.

AEvent is the event handler that DoOnHasRights should call first to check access rights.

ARights is a string that represents the rights the user needs.

See Also

ImplHasViewRights (see page 24116)

HasViewRights (see page 24112)

173.1.41.2.5 TBaseEndUserAdapter.ExecuteLogin Method

Starts the server-side login process.

Pascal

```
procedure ExecuteLogin(AUserID: Variant); virtual;
```

C++

```
virtual __fastcall ExecuteLogin(Variant AUserID);
```

Description

The ExecuteLogin function is called from sever-side script to start the login process. This base class version of the function is empty. A customized overridden version of this function can be written to do specific login processing for the application.

See Also

ImplProduceLoginPage (see page 24117)

173.1.41.2.6 TBaseEndUserAdapter.ExecuteLoginForm Method

Creates the HTML login page.

Pascal

```
procedure ExecuteLoginForm; virtual;
```

C++

```
virtual __fastcall ExecuteLoginForm();
```

Description

The ExecuteLoginForm function is called from server-side script to create the HTML login page. This function calls ImplProduceLoginPage (see page 24117) to make the page producer generate the login page.

See Also

ImplProduceLoginPage (see page 24117)

173.1.41.2.7 TBaseEndUserAdapter.ExecuteLogout Method

Performs server-side logout.

Pascal

```
procedure ExecuteLogout; virtual;
```

C++

```
virtual __fastcall ExecuteLogout();
```

Description

The ExecuteLogout function is called from server-side script to perform the logout process. This base class version of the

function is empty. A customized overridden version of this function can be written to do specific logout processing for the application.

See Also

CanExecuteLogout (see page 24107)

173.1.41.2.8 TBaseEndUserAdapter.GetDisplayName Method

Retrieves the display name string.

Pascal

```
function GetDisplayName: string;
```

C++

```
__fastcall AnsiString GetDisplayName();
```

Description

The GetDisplayName function is called from server-side script to get the display name string. It calls ImplGetDisplayName (see page 24114) to get the display name for this TBaseEndUserAdapter (see page 24097).

See Also

ImplGetDisplayName (see page 24114)

GetDisplayNameValue (see page 24109)

173.1.41.2.9 TBaseEndUserAdapter.GetDisplayNameValue Method

Gets the display name of the end user.

Pascal

```
function GetDisplayNameValue: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetDisplayNameValue();
```

Description

The GetDisplayNameValue is called from server-side script to get the display name of the end user. This base class version of the function returns an empty string. A customized overridden version of this function could be written to return the display name for the end user in your application.

See Also

ImplGetDisplayName (see page 24114)

GetDisplayName (see page 24109)

173.1.41.2.10 TBaseEndUserAdapter.GetEndUserAdapter Method

Gets the end-user adapter.

Pascal

```
function GetEndUserAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetEndUserAdapter();
```

Description

The GetEndUserAdapter function is called from server-side script to get the end-user adapter. This function returns itself, that is, this TBaseEndUserAdapter (see page 24097).

See Also

ImplGetEndUserAdapter (see page 24114)

173.1.41.2.11 TBaseEndUserAdapter.GetLoggedIn Method

Determines if the user is logged in

Pascal

```
function GetLoggedIn: Boolean;
```

C++

```
__fastcall Boolean GetLoggedIn();
```

Description

The GetLoggedIn function is called from server-side script to determine if the user is logged in. This function calls ImplGetLoggedIn (see page 24114) to get the result.

See Also

ImplGetLoggedIn (see page 24114)

GetLoggedInValue (see page 24110)

173.1.41.2.12 TBaseEndUserAdapter.GetLoggedInValue Method

Determines if the user is logged in.

Pascal

```
function GetLoggedInValue: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetLoggedInValue();
```

Description

The GetLoggedInValue function is called from server-side script to determine if the user is logged in. This base class version of the function always returns false. A customized overridden version could be written specifically for your application.

See Also

ImplGetLoggedIn (see page 24114)

GetLoggedIn (see page 24110)

173.1.41.2.13 TBaseEndUserAdapter.GetLoginFormAction Method

Gets the action component for the login form.

Pascal

```
function GetLoginFormAction: TComponent;
```

C++

```
__fastcall TComponent * GetLoginFormAction();
```

Description

The GetLoginFormAction is called from server-side script to get the action component for the login form. This function calls ImplGetLoginFormAction (see page 24115) to get the action component.

See Also

ImplGetLoginFormAction (see page 24115)

173.1.41.2.14 TBaseEndUserAdapter.GetLogoutAction Method

Gets the action component for the logout form.

Pascal

```
function GetLogoutAction: TComponent;
```

C++

```
__fastcall TComponent * GetLogoutAction();
```

Description

The GetLogoutAction is called from server-side script to get the action component for the logout form. This function calls ImplGetLogoutFormAction to get the action component.

See Also

ImplGetLogoutAction (see page 24115)

173.1.41.2.15 TBaseEndUserAdapter.GetUserID Method

Gets the UserID from this TBaseEndUserAdapter (see page 24097).

Pascal

```
function GetUserID: Variant; virtual;
```

C++

```
virtual __fastcall Variant GetUserID();
```

Description

The GetUserID function is called from server-side script to get the UserID from this TBaseEndUserAdapter (see page 24097). This base class version of the function always returns Unassigned. A customized overridden version of this function could be written specifically for your application.

173.1.41.2.16 TBaseEndUserAdapter.HasExecuteRights Method

Determines if the user has execute rights for the TBaseEndUserAdapter (see page 24097).

Pascal

```
function HasExecuteRights(ARights: string): Boolean;
```

C++

```
__fastcall Boolean HasExecuteRights(AnsiString ARights);
```

Description

The HasExecuteRights function is called from server-side script to determine if the user has execute rights for the TBaseEndUserAdapter (see page 24097). This function calls ImplHasExecuteRights (see page 24115) to check the user rights.

See Also

ImplHasExecuteRights (see page 24115)

HasModifyRights (see page 24112)

HasViewRights (see page 24112)

173.1.41.2.17 TBaseEndUserAdapter.HasModifyRights Method

Determines if the user has modify rights for the TBaseEndUserAdapter (see page 24097).

Pascal

```
function HasModifyRights(ARights: string): Boolean;
```

C++

```
__fastcall Boolean HasModifyRights(AnsiString ARights);
```

Description

The HasModifyRights function is called from server-side script to determine if the user has modify rights for the TBaseEndUserAdapter (see page 24097). This function calls ImplHasModifyRights (see page 24116) to check the user rights.

See Also

ImplHasModifyRights (see page 24116)

HasModifyRights

HasViewRights (see page 24112)

173.1.41.2.18 TBaseEndUserAdapter.HasViewRights Method

Determine if the user has view rights to the TBaseEndUserAdapter (see page 24097).

Pascal

```
function HasViewRights(ARights: string): Boolean;
```

C++

```
__fastcall Boolean HasViewRights(AnsiString ARights);
```

Description

The HasViewRights function is called from server-side script to determine if the user has view rights for the TBaseEndUserAdapter (see page 24097). This function calls ImplHasViewRights (see page 24116) to check the user rights.

See Also

ImplHasViewRights (see page 24116)

HasModifyRights (see page 24112)

HasViewRights

173.1.41.2.19 TBaseEndUserAdapter.ImplCanAddActionClass Method

Determines if the given action class can be added.

Pascal

```
function ImplCanAddActionClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

The ImplCanAddActionClass function calls the base class to determine if the given action class can be added. In addition, it allows actions components to be added if they are derived from TCustomEndUserAdapterAction (see page 24326) at some level. This function returns true if the action class is allowed to be added, or false otherwise.

See Also

ImplCanAddActionClass (see page 24163)

TCustomEndUserAdapterAction (see page 24326)

173.1.41.2.20 TBaseEndUserAdapter.ImplCanAddFieldClass Method

Determines if the given action class can be added.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

The ImplCanAddFieldClass function calls the base class to determine if the given action class can be added. In addition to that, it allows fields components to be added if they are derived from TAdapterEndUserField (see page 23793).

See Also

ImplCanAddFieldClass (see page 24163)

TAdapterEndUserField (see page 23793)

173.1.41.2.21 TBaseEndUserAdapter.ImplGetActionsList Method

Lists supported end-user adapter actions.

Pascal

```
procedure ImplGetActionsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetActionsList(TStrings AList);
```

Description

The ImplGetActionsList procedure obtains the list of all supported actions for the end-user adapter.

173.1.41.2.22 TBaseEndUserAdapter.ImplGetDisplayName Method

Gets the display name for this TBaseEndUserAdapter (see page 24097).

Pascal

```
function ImplGetDisplayName: string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplGetDisplayName();
```

Description

The ImplGetDisplayName function is called from GetDisplayName (see page 24109) to get the display name for this TBaseEndUserAdapter (see page 24097). It gets the display name from the DisplayName field component.

See Also

GetDisplayName (see page 24109)

173.1.41.2.23 TBaseEndUserAdapter.ImplGetEndUserAdapter Method

Returns this TBaseEndUserAdapter (see page 24097).

Pascal

```
function ImplGetEndUserAdapter: TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplGetEndUserAdapter();
```

Description

The ImplGetEndUserAdapter simply returns this TBaseEndUserAdapter (see page 24097).

See Also

GetEndUserAdapter (see page 24109)

173.1.41.2.24 TBaseEndUserAdapter.ImplGetFieldsList Method

Obtains the list of all of the fields for the end-user adapter.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

The ImplGetFieldsList procedure obtains the list of all of the fields for the end-user adapter.

173.1.41.2.25 TBaseEndUserAdapter.ImplGetLoggedIn Method

Checks the value of the LoggedIn field.

Pascal

```
function ImplGetLoggedIn: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetLoggedIn();
```

Description

The ImplGetLoggedIn function is called from GetLoggedIn (see page 24110). It checks the value of the LoggedIn field component and returns the result.

See Also

GetLoggedIn (see page 24110)

173.1.41.2.26 TBaseEndUserAdapter.ImplGetLoginFormAction Method

Gets the LoginForm action component for this TBaseEndUserAdapter (see page 24097).

Pascal

```
function ImplGetLoginFormAction: TComponent;
```

C++

```
__fastcall TComponent * ImplGetLoginFormAction();
```

Description

The ImplGetLoginFormAction function is called from GetLoginFormAction (see page 24110) to get the LoginForm action component for this TBaseEndUserAdapter (see page 24097).

See Also

GetLoginFormAction (see page 24110)

173.1.41.2.27 TBaseEndUserAdapter.ImplGetLogoutAction Method

Gets the Logout action component for this TBaseEndUserAdapter (see page 24097).

Pascal

```
function ImplGetLogoutAction: TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplGetLogoutAction();
```

Description

The ImplGetLogoutAction function is called from GetLogoutAction (see page 24111) to get the Logout action component for this TBaseEndUserAdapter (see page 24097).

See Also

GetLogoutAction (see page 24111)

173.1.41.2.28 TBaseEndUserAdapter.ImplHasExecuteRights Method

Checks user rights and determines if the user has execute rights.

Pascal

```
function ImplHasExecuteRights(ARights: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasExecuteRights(AnsiString ARights);
```

Description

The ImplHasExecuteRights function is called from the HasExecuteRights (see page 24111) function. It calls the DoOnHasRights (see page 24107) function to check the user rights and determines if the user has execute rights.

See Also

OnHasRights (see page 24119)

173.1.41.2.29 TBaseEndUserAdapter.ImplHasModifyRights Method

Checks user rights and determines if the user has modify rights.

Pascal

```
function ImplHasModifyRights(ARights: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasModifyRights(AnsiString ARights);
```

Description

The ImplHasModifyRights function is called from the HasModifyRights (see page 24112) function. It calls the DoOnHasRights (see page 24107) function to check the user rights and determine if the user has modify rights.

See Also

OnHasRights (see page 24119)

173.1.41.2.30 TBaseEndUserAdapter.ImplHasViewRights Method

Checks user rights and determines if the user has view rights.

Pascal

```
function ImplHasViewRights(ARights: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasViewRights(AnsiString ARights);
```

Description

The ImplHasViewRights is called from the HasViewRights (see page 24112) function. It calls the DoOnHasRights (see page 24107) function to check the user rights and determine if the user has view rights.

See Also

DoOnHasRights (see page 24107)

173.1.41.2.31 TBaseEndUserAdapter.ImplLogIn Method

Processes the login for the given user.

Pascal

```
procedure ImplLogIn(AUserID: Variant); virtual;
```

C++

```
virtual __fastcall ImplLogIn(Variant AUserID);
```

Description

The ImplLogIn procedure is called from LogIn (see page 24117). This function calls ExecuteLogin (see page 24108) to

process the login for the given user.

See Also

Login (see page 24117)

ExecuteLogin (see page 24108)

173.1.41.2.32 TBaseEndUserAdapter.ImplProduceLoginPage Method

Generate the HTML login page.

Pascal

```
function ImplProduceLoginPage: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplProduceLoginPage();
```

Description

The ImplProduceLoginPage is called to generate the HTML login page It first calls the OnProduceLoginPage (see page 24119) event handler to allow for a customized version. If no event handler is defined, it then has default code to generate the page using the page producer.

See Also

OnProduceLoginPage (see page 24119)

173.1.41.2.33 TBaseEndUserAdapter.LogIn Method

Logs in a specified user.

Pascal

```
procedure LogIn(AUserID: Variant);
```

C++

```
__fastcall LogIn(Variant AUserID);
```

Description

Call LogIn to record a user as logged in. Typically, LogIn is called in response to a message from the login page that ProduceLoginPage generates.

UserID is the identifier of the user that logged in. It is the value of the UserID property of the entry for the user in a WebUserList component, or, in applications that implement their own login scheme, any other Variant that identifies an end user.

See Also

LoggedIn (see page 24678)

ProduceLoginPage (see page 24677)

TWebUserList (see page 25625)

UserID (see page 25619)

TLoginFormAdapter (see page 24500)

173.1.41.2.34 TBaseEndUserAdapter.ProduceLoginPage Method

Generates the login page.

Pascal

```
function ProduceLoginPage: Boolean;
```

C++

```
__fastcall Boolean ProduceLoginPage();
```

Description

The ProduceLoginPage function is called from server-side script to generate the login page. This function calls ImplProduceLoginPage (see page 24117) to generate the page.

See Also

ImplProduceLoginPage (see page 24117)

173.1.41.2.35 TBaseEndUserAdapter.UserList Method

Gets the user list.

Pascal

```
function UserList: IWebUserList;
```

C++

```
__fastcall IWebUserList UserList();
```

Description

The UserList function is called from server-side script to get the user list. If the user list is not stored in this TBaseEndUserAdapter (see page 24097), it then attempts to retrieve the user list through the application module.

173.1.41.3 TBaseEndUserAdapter Properties

The properties of the TBaseEndUserAdapter class are listed here.

Public Properties

	Name	Description
	LoginPage (see page 24118)	Specifies the name of the page that acts as the login page.

173.1.41.3.1 TBaseEndUserAdapter.LoginPage Property

Specifies the name of the page that acts as the login page.

Pascal

```
property LoginPage: string;
```

C++

```
__property AnsiString LoginPage;
```

Description

Use LoginPage to read or set the name of the login page for this TBaseEndUserAdapter (see page 24097). The name of the login page is the name of the Web page module that generates the login page.

173.1.41.4 TBaseEndUserAdapter Events

The events of the TBaseEndUserAdapter class are listed here.

Public Events

	Name	Description
	OnHasRights (see page 24119)	Determines if the user rights match the rights being queried.
	OnProduceLoginPage (see page 24119)	Allows implementation of a customized login page.

173.1.41.4.1 TBaseEndUserAdapter.OnHasRights Event

Determines if the user rights match the rights being queried.

Pascal

```
property OnHasRights: TEndUserRightsEvent;
```

C++

```
__property TEndUserRightsEvent OnHasRights;
```

Description

The OnHasRights event handler is called from DoOnHasRights (see page 24107) to determine if the user rights match the rights being queried. A customized event handler should be written to determine the rights for your specific application.

See Also

DoOnHasRights (see page 24107)

ImplHasExecuteRights (see page 24115)

ImplHasModifyRights (see page 24116)

ImplHasViewRights (see page 24116)

173.1.41.4.2 TBaseEndUserAdapter.OnProduceLoginPage Event

Allows implementation of a customized login page.

Pascal

```
property OnProduceLoginPage: TEndUserHandledEvent;
```

C++

```
__property TEndUserHandledEvent OnProduceLoginPage;
```

Description

The OnProduceLoginPage event is triggered in the ImplProduceLoginPage (see page 24117). If a customized login page is desired for your application, a customized event handler can be written for this event.

See Also

ImplProduceLoginPage (see page 24117)

ProduceLoginPage (see page 24117)

CanProduceLoginPage (see page 24107)

173.1.42 TBaseValuesListAdapter Class

TBaseValuesListAdapter is a base class for adapters that provide a list of name/value pairs.

Class Hierarchy

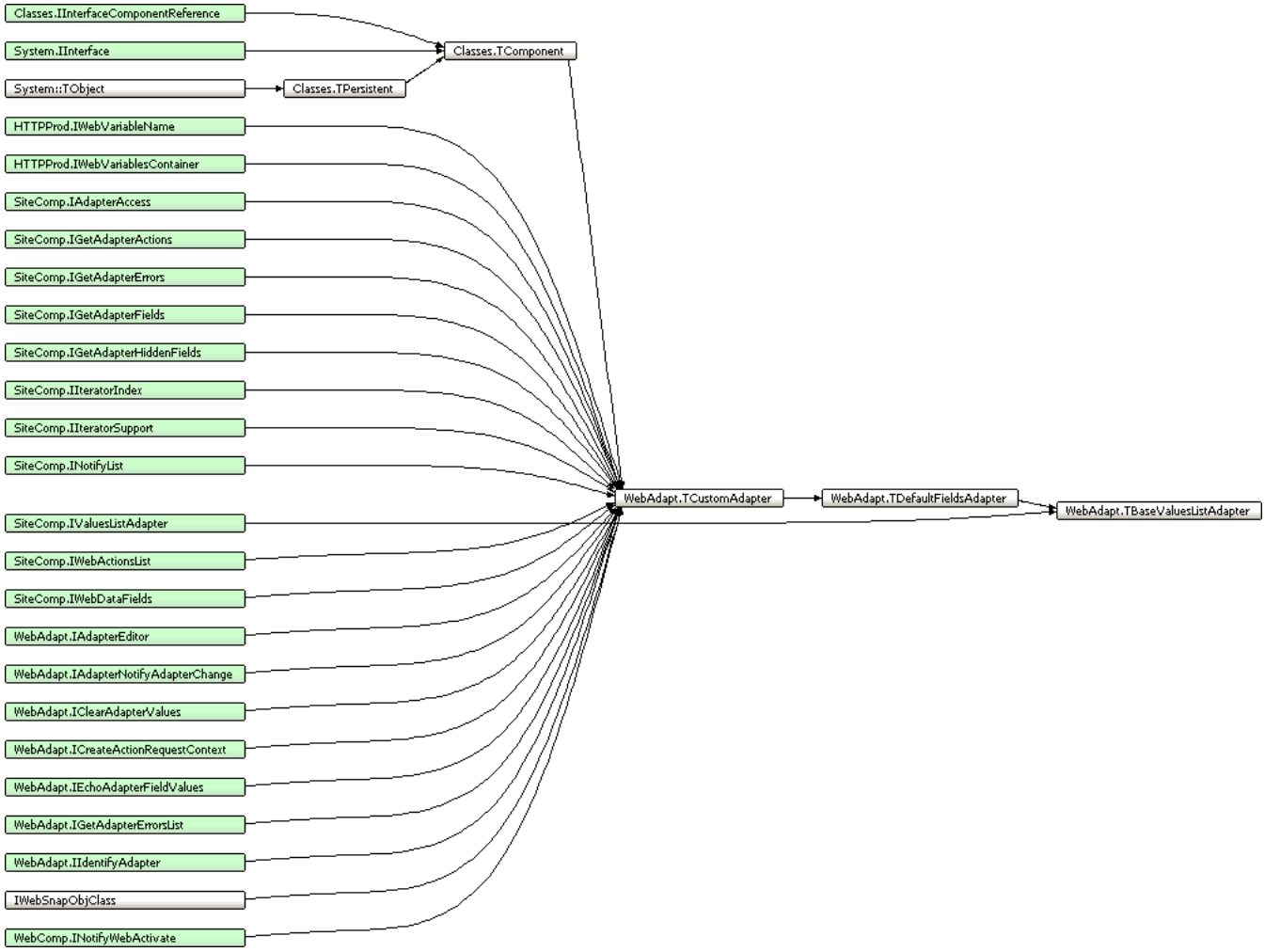

Pascal

```
TBaseValuesListAdapter = class(TDefaultFieldsAdapter, IValuesListAdapter);
```

C++

```
class TBaseValuesListAdapter : public TDefaultFieldsAdapter, public IValuesListAdapter;
```

File

WebAdapt

Description

Descendants of TBaseValuesListAdapter (such as TStringsValuesList (see page 24540) and TDataSetValuesList) are used to provide name/value pairs for an adapter field. Adapter fields reference a values list adapter using the adapter field's ValuesList

property. Typically, the list of name/value pairs is used by server-side scripts to populate <select>, <input type=checkbox>, or <input type=radio>.

TBaseValuesListAdapter has two default fields. One field is responsible for providing a value and the other for providing a name.

See Also

TStringsValuesList (see page 24540)

TDataSetValuesList (see page 8079)

TAdapterUpdateField (see page 24007)

173.1.42.1 TBaseValuesListAdapter Members

The following tables list the members exposed by TBaseValuesListAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.

	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetListImage (see page 24128)	Returns an image associated with the current name/value pair.
	GetListImageOfValue (see page 24128)	Looks up an image for a value.
	GetListName (see page 24128)	Responsible returning the name of the name/value pair.
	GetListNameOfValue (see page 24128)	Looks up a name for a value.
	GetListValue (see page 24129)	Responsible for returning the value of the name/value pair.
	GetNameFieldClass (see page 24129)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValueFieldClass (see page 24129)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 24130)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 24130)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.

	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 24130)	Adds default field components to this adapter.
	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetListImage (see page 24131)	Responsible returning an image associated with the current name/value pair.
	ImplGetListImageOfValue (see page 24131)	Looks up an image for a value.
	ImplGetListName (see page 24131)	Responsible returning the name of the name/value pair.
	ImplGetListNameOfValue (see page 24132)	Looks up a name for a value.
	ImplGetListValue (see page 24132)	Responsible for returning the value of the name/value pair.
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

V	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 24184)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24185)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24186)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24186)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 24187)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.

	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 24181)	Actions is the container for adapter actions components.
R	Actions (see page 24402)	Container for adapter actions components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.

R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Data (see page 24181)	Data is the container for adapter field components.
R	Data (see page 24402)	Container for adapter field components.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.42.2 TBaseValuesListAdapter Methods

The methods of the TBaseValuesListAdapter class are listed here.

Protected Methods

	Name	Description
	GetListImage (see page 24128)	Returns an image associated with the current name/value pair.
	GetListImageOfValue (see page 24128)	Looks up an image for a value.
	GetListName (see page 24128)	Responsible returning the name of the name/value pair.
	GetListNameOfValue (see page 24128)	Looks up a name for a value.
	GetListValue (see page 24129)	Responsible for returning the value of the name/value pair.
V	GetNameFieldClass (see page 24129)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
V	GetValueFieldClass (see page 24129)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
V	GetWebSnapObjClass (see page 24130)	Returns the class type for this adapter field.
V	ImplCanAddFieldClass (see page 24130)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplGetFieldsList (see page 24130)	Adds default field components to this adapter.
V	ImplGetListImage (see page 24131)	Responsible returning an image associated with the current name/value pair.
V	ImplGetListImageOfValue (see page 24131)	Looks up an image for a value.
V	ImplGetListName (see page 24131)	Responsible returning the name of the name/value pair.
V	ImplGetListNameOfValue (see page 24132)	Looks up a name for a value.
V	ImplGetListValue (see page 24132)	Responsible for returning the value of the name/value pair.

173.1.42.2.1 TBaseValuesListAdapter.GetListImage Method

Returns an image associated with the current name/value pair.

Pascal

```
function GetListImage: TComponent;
```

C++

```
__fastcall TComponent * GetListImage();
```

Description

GetListImage returns the result of ImplGetListImage (see page 24131).

See Also

ImplGetListImage (see page 24131)

173.1.42.2.2 TBaseValuesListAdapter.GetListImageOfValue Method

Looks up an image for a value.

Pascal

```
function GetListImageOfValue(Value: Variant): TComponent;
```

C++

```
__fastcall TComponent * GetListImageOfValue(Variant Value);
```

Description

GetListImageOfValue returns the result of ImplGetListImageOfValue (see page 24131).

See Also

ImplGetListImageOfValue (see page 24131)

173.1.42.2.3 TBaseValuesListAdapter.GetListName Method

Responsible returning the name of the name/value pair.

Pascal

```
function GetListName: string;
```

C++

```
__fastcall AnsiString GetListName();
```

Description

GetListName returns the result of ImplGetListName (see page 24131).

See Also

GetListName

ImplGetListValue (see page 24132)

173.1.42.2.4 TBaseValuesListAdapter.GetListNameOfValue Method

Looks up a name for a value.

Pascal

```
function GetListNameOfValue(Value: Variant): string;
```

C++

```
__fastcall AnsiString GetListNameOfValue(Variant Value);
```

Description

GetListNameOfValue returns the result of ImplGetListNameOfValue (see page 24132).

See Also

ImplGetListNameOfValue (see page 24132)

173.1.42.2.5 TBaseValuesListAdapter.GetListValue Method

Responsible for returning the value of the name/value pair.

Pascal

```
function GetListValue: Variant;
```

C++

```
__fastcall Variant GetListValue();
```

Description

GetListValue returns the result of ImplGetListValue (see page 24132).

See Also

GetListName (see page 24128)

ImplGetListValue (see page 24132)

173.1.42.2.6 TBaseValuesListAdapter.GetNameFieldClass Method

Returns the class of adapter field that is responsible for providing the value of the name/value pair.

Pascal

```
function GetNameFieldClass: TComponentClass; virtual;
```

C++

```
virtual __fastcall TComponentClass GetNameFieldClass();
```

Description

GetNameFieldClass returns nil (Delphi) or NULL (C++). Descendant classes must override this method.

See Also

GetValueFieldClass (see page 24129)

173.1.42.2.7 TBaseValuesListAdapter.GetValueFieldClass Method

Returns the class of adapter field that is responsible for providing the name of the name/value pair.

Pascal

```
function GetValueFieldClass: TComponentClass; virtual;
```

C++

```
virtual __fastcall TComponentClass GetValueFieldClass();
```

Description

GetValueFieldClass returns nil (Delphi) or NULL (C++). Descendant classes must override this method.

See Also

GetNameFieldClass (see page 24129)

173.1.42.2.8 TBaseValuesListAdapter.GetWebSnapObjClass Method

Returns the class type for this adapter field.

Pascal

```
function GetWebSnapObjClass: TObjectScripterClass; override;
```

C++

```
virtual __fastcall TObjectScripterClass GetWebSnapObjClass();
```

Description

This function returns the class type for this adapter field, which is used by the script engine to create a script object. GetWebSnapObjClass tells the script engine what type of object to create.

173.1.42.2.9 TBaseValuesListAdapter.ImplCanAddFieldClass Method

Indicates whether a type of adapter field can be added to this adapter.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddFieldClass check the field class by calling GetNameFieldClass (see page 24129), GetValueFieldClass (see page 24129), and TCustomAdapter (see page 24139)'s ImplCanAddFieldClass method.

See Also

GetNameFieldClass (see page 24129)

GetValueFieldClass (see page 24129)

ImplCanAddFieldClass (see page 24163)

173.1.42.2.10 TBaseValuesListAdapter.ImplGetFieldsList Method

Adds default field components to this adapter.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

ImplGetFieldsList adds a value field and a name field using the classes returned by GetNameFieldClass (see page 24129) and GetValueFieldClass (see page 24129).

See Also

GetNameFieldClass (see page 24129)

GetValueFieldClass (see page 24129)

173.1.42.2.11 TBaseValuesListAdapter.ImplGetListImage Method

Responsible returning an image associated with the current name/value pair.

Pascal

```
function ImplGetListImage: TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplGetListImage();
```

Description

ImplGetListImage is a placeholder. Descendant adapters that support images must override this method.

See Also

GetListImage (see page 24128)

173.1.42.2.12 TBaseValuesListAdapter.ImplGetListImageOfValue Method

Looks up an image for a value.

Pascal

```
function ImplGetListImageOfValue(Value: Variant): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplGetListImageOfValue(Variant Value);
```

Description

ImplGetListImageOfValue is a placeholder. Descendant classes that support image lookups must override this function.

See Also

GetListImageOfValue (see page 24128)

173.1.42.2.13 TBaseValuesListAdapter.ImplGetListName Method

Responsible returning the name of the name/value pair.

Pascal

```
function ImplGetListName: string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplGetListName();
```

Description

ImplGetListName returns an empty string. ImplGetListName must be overridden by a descendant class.

See Also

GetListName (see page 24128)

173.1.42.2.14 TBaseValuesListAdapter.ImplGetListNameOfValue Method

Looks up a name for a value.

Pascal

```
function ImplGetListNameOfValue(Value: Variant): string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplGetListNameOfValue(Variant Value);
```

Description

ImplGetListNameOfValue is a placeholder. Descendant classes that support name lookups must override this function.

See Also

GetListNameOfValue (see page 24128)

173.1.42.2.15 TBaseValuesListAdapter.ImplGetListValue Method

Responsible for returning the value of the name/value pair.

Pascal

```
function ImplGetListValue: Variant; virtual;
```

C++

```
virtual __fastcall Variant ImplGetListValue();
```

Description

ImplGetListValue is a placeholder. ImplGetListValue must be overriden by a descendant class.

See Also

GetListValue (see page 24129)

173.1.43 TBaseValuesListField Class

TBaseValuesListField is a base class for adapter field components that are children of a values list adapter component.

Class Hierarchy

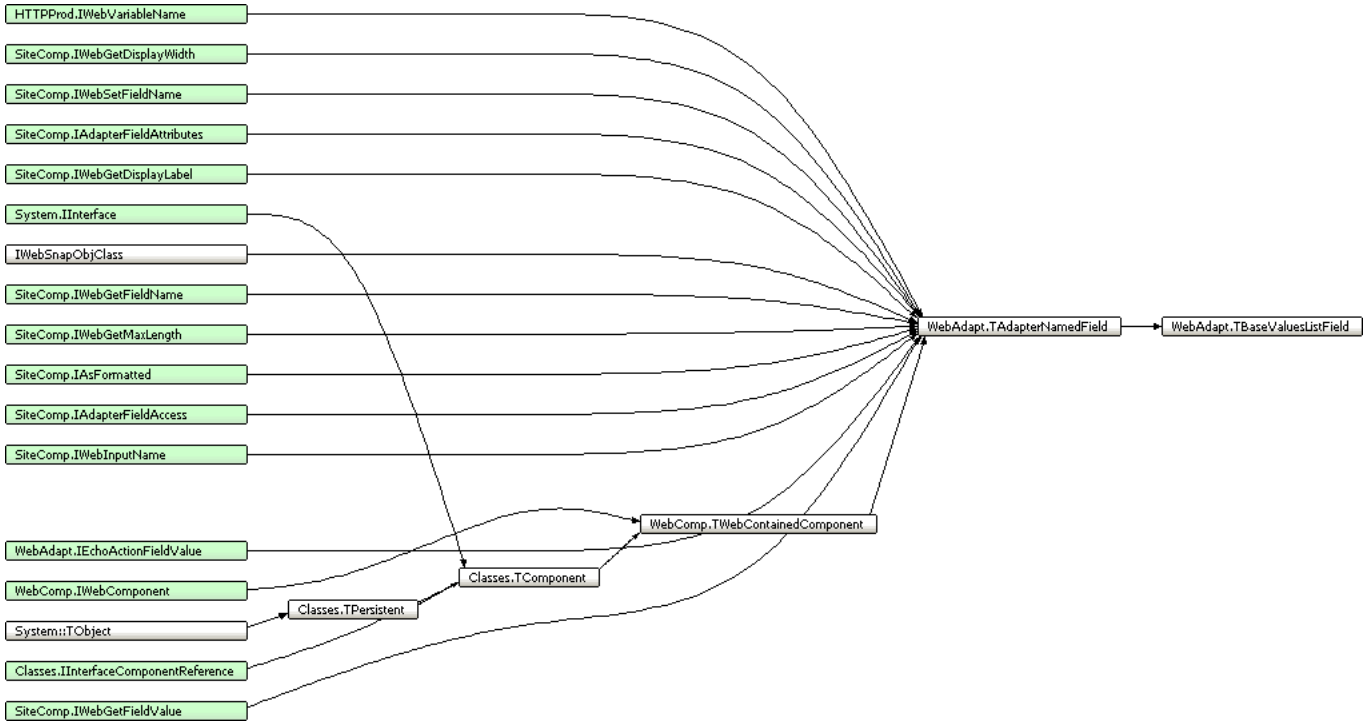

Pascal

```
TBaseValuesListField = class(TAdapterNamedField);
```

C++

```
class TBaseValuesListField : public TAdapterNamedField;
```

File

WebAdapt

Description

Derive from TBaseValuesListField to declare an adapter field that can be a child of TBaseValuesListAdapter (see page 24120).

See Also

TStringsValuesList (see page 24540)

TBaseValuesListAdapter (see page 24120)

TAdapterNamedField (see page 23925)

TValuesListNameField (see page 24552)

TValuesListValueField (see page 24558)

173.1.43.1 TBaseValuesListField Members

The following tables list the members exposed by TBaseValuesListField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 24138)	Retrieves the parent adapter component.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
V	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.

V	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
V	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24139)	Retrieves the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.

	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.

	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.43.2 TBaseValuesListField Methods

The methods of the TBaseValuesListField class are listed here.

Protected Methods

	Name	Description
	GetAdapter (see page 24138)	Retrieves the parent adapter component.

173.1.43.2.1 TBaseValuesListField.GetAdapter Method

Retrieves the parent adapter component.

Pascal

```
function GetAdapter: TBaseValuesListAdapter;
```

C++

```
__fastcall TBaseValuesListAdapter GetAdapter();
```

Description

GetAdapter implements the Adapter (see page 24139) property.

See Also

Adapter (see page 24139)

TBaseValuesListAdapter (see page 24120)

173.1.43.3 TBaseValuesListField Properties

The properties of the TBaseValuesListField class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 24139)	Retrieves the parent adapter component.

173.1.43.3.1 TBaseValuesListField.Adapter Property

Retrieves the parent adapter component.

Pascal

```
property Adapter: TBaseValuesListAdapter;
```

C++

```
__property TBaseValuesListAdapter Adapter;
```

Description

The Adapter property is used by descendants to access the parent adapter component.

See Also

Adapter

TBaseValuesListAdapter (see page 24120)

173.1.44 TCustomAdapter Class

TCustomAdapter is a base class for adapter components.

Class Hierarchy

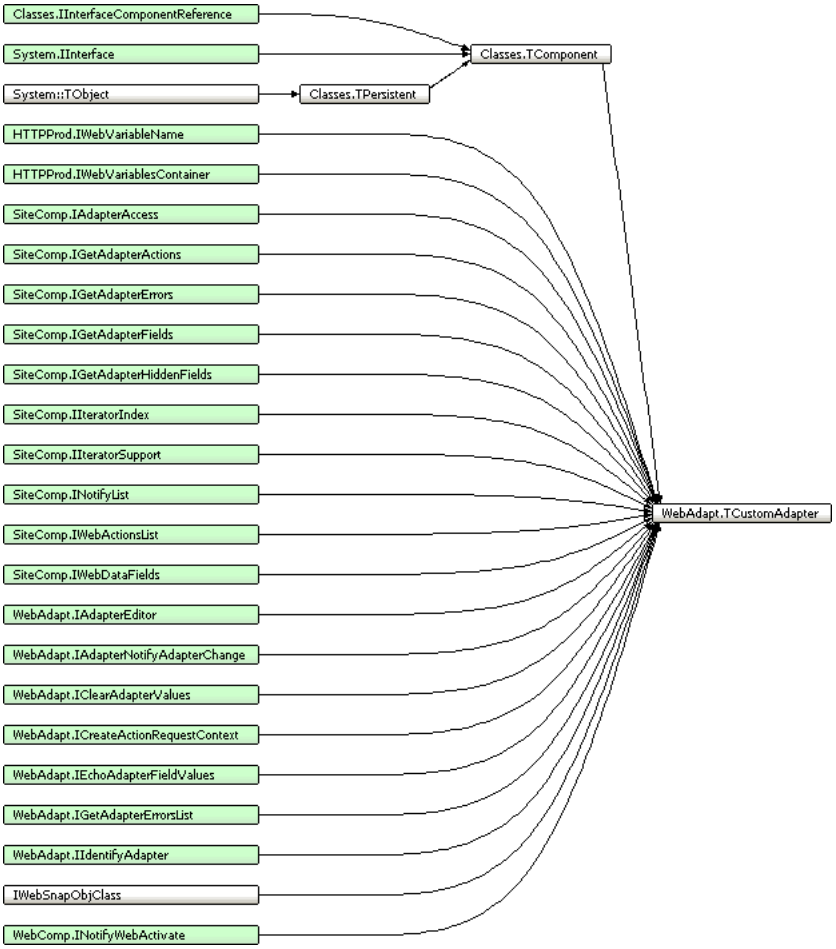

Pascal

```
TCustomAdapter = class(TComponent, IIdentifyAdapter, IWebVariableName, IWebVariablesContainer,
INotifyList, IWebSnapObjClass, IGetAdapterErrors, IGetAdapterErrorsList, INotifyWebActivate,
IAdapterEditor, IGetAdapterFields, IGetAdapterActions, IIteratorSupport, IClearAdapterValues,
IEchoAdapterFieldValues, IAdapterAccess, IGetAdapterHiddenFields, ICreateActionRequestContext,
IWebDataFields, IWebActionsList, IAdapterNotifyAdapterChange, IIteratorIndex);
```

C++

```
class TCustomAdapter : public TComponent, public IIdentifyAdapter, public IWebVariableName,
public IWebVariablesContainer, public INotifyList, public IWebSnapObjClass, public
IGetAdapterErrors, public IGetAdapterErrorsList, public INotifyWebActivate, public
IAdapterEditor, public IGetAdapterFields, public IGetAdapterActions, public IIteratorSupport,
public IClearAdapterValues, public IEchoAdapterFieldValues, public IAdapterAccess, public
IGetAdapterHiddenFields, public ICreateActionRequestContext, public IWebDataFields, public
IWebActionsList, public IAdapterNotifyAdapterChange, public IIteratorIndex;
```

File

WebAdapt

Description

TCustomAdapter implements interfaces used to access an adapter properties using server-side script. Most of the interface implementation methods call virtual functions. When defining new derived classes, override these virtual functions to customize

the behavior of the adapter class.

See Also

TAdapter (see page 23713)

TEndUserAdapter (see page 24421)

TEndUserSessionAdapter (see page 24458)

TFieldValuesAdapter (see page 24471)

TDataSetValuesList (see page 8079)

TStringsValuesList (see page 24540)

TApplicationAdapter (see page 24032)

TLoginFormAdapter (see page 24500)

TDataSetAdapter (see page 7872)

TPagedAdapter (see page 24525)

173.1.44.1 TCustomAdapter Members

The following tables list the members exposed by TCustomAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.

	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.

	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.

	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 24184)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24185)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24186)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24186)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 24187)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).

	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 24181)	Actions is the container for adapter actions components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.

R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Data (see page 24181)	Data is the container for adapter field components.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.44.2 TCustomAdapter.Create Constructor

Constructor for TCustomAdapter (see page 24139).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapter(TComponent * AOwner);
```

Description

Creates instance variables that hold interfaces, object references used by TCustomAdapter (see page 24139).

See Also

Create_d

173.1.44.3 TCustomAdapter.Destroy Destructor

Destructor for TCustomAdapter (see page 24139).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapter();
```

Description

Frees objects used by TCustomAdapter (see page 24139).

See Also

Free (see page 22300)

173.1.44.4 TCustomAdapter Methods

The methods of the TCustomAdapter class are listed here.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
V	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.

	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
V	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
V	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
V	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
V	ImplGetVariable (see page 24166)	Returns an action or field.
V	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
V	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
V	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
V	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
V	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
V	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
V	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
V	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
V	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
V	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
V	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
V	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
V	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
V	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
V	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
V	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
V	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
V	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.

	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Public Methods

	Name	Description
	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

173.1.44.4.1 TCustomAdapter.AddAdapterModeHiddenField Method

Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.

Pascal

```
procedure AddAdapterModeHiddenField(AHiddenFields: TAdapterHiddenFieldsList);
```

C++

```
__fastcall AddAdapterModeHiddenField(TAdapterHiddenFieldsList AHiddenFields);
```

Description

Descendant adapters may support modes. For example, TDataSetAdapter supports Edit, Insert, and Browse modes. The hidden field defined by AddAdapterModeHiddenField is added to an HTML form using the server-side script method HiddenFields.WriteFields.

See Also

listhiddenfields (see page 24173)

173.1.44.4.2 TCustomAdapter.AddOrigValueHiddenFields Method

Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.

Pascal

```
procedure AddOrigValueHiddenFields(AHiddenFields: TAdapterHiddenFieldsList);
```

C++

```
__fastcall AddOrigValueHiddenFields(TAdapterHiddenFieldsList AHiddenFields);
```

Description

In order to ensure that the data associated with the adapter hasn't been changed by another user, descendant adapters may check that the current values of adapter fields are the same as the original values. For example, TDataSetAdapter raises an exception if the original values are not the same as the dataset's values. The hidden fields defined by AddOrigValueHiddenFields are added to an HTML form using the server-side script method HiddenRecordFields.WriteFields.

See Also

ListHiddenRecordFields (see page 24173)

173.1.44.4.3 TCustomAdapter.AddRecordKeysHiddenFields Method

Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.

Pascal

```
procedure AddRecordKeysHiddenFields(AHiddenFields: TAdapterHiddenFieldsList);
```

C++

```
__fastcall AddRecordKeysHiddenFields(TAdapterHiddenFieldsList AHiddenFields);
```

Description

Descendant adapters may use key values to define the relationship between values in the HTTP request and adapter data values. For example, TDataSetAdapter supports updates to multiple detail records by using hidden fields to identify the relationship between values in the HTTP request with dataset rows. The hidden fields defined by AddRecordKeysHiddenFields are added to an HTML form using the server-side script method HiddenRecordFields.WriteFields.

See Also

ListHiddenRecordFields (see page 24173)

173.1.44.4.4 TCustomAdapter.CanAddActionClass Method

Indicates whether a type of adapter action can be added to this adapter.

Pascal

```
function CanAddActionClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

CanAddActionClass is called at design-time when editing an adapter. CanAddActionClass returns the result of ImplCanAddActionClass (see page 24163).

See Also

ImplCanAddActionClass (see page 24163)

173.1.44.4.5 TCustomAdapter.CanAddFieldClass Method

Indicates whether a type of adapter field can be added to this adapter.

Pascal

```
function CanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

CanAddFieldClass is called at design-time when editing an adapter. CanAddFieldClass returns the result of ImplCanAddFieldClass (see page 24163).

See Also

ImplCanAddFieldClass (see page 24163)

173.1.44.4.6 TCustomAdapter.CheckModifyAccess Method

Pascal

```
procedure CheckModifyAccess;
```

C++

```
__fastcall CheckModifyAccess();
```

Description

This is CheckModifyAccess, a member of class TCustomAdapter.

173.1.44.4.7 TCustomAdapter.ClearAdapterValues Method

Clears adapter state information.

Pascal

```
procedure ClearAdapterValues;
```

C++

```
__fastcall ClearAdapterValues();
```

Description

ClearAdapterValues calls ImplClearAdapterValues (see page 24163).

See Also

ImplNotifyDeactivate (see page 24170)

ImplClearAdapterValues (see page 24163)

173.1.44.4.8 TCustomAdapter.ClearErrors Method

Clears the content of the adapter's Errors (see page 24182) property.

Pascal

```
procedure ClearErrors;
```

C++

```
__fastcall ClearErrors();
```

Description

The errors are cleared when the web module is deactivated. This ensures that the errors encountered during the processing of an HTTP request will not affect another HTTP request.

See Also

Errors (see page 24182)

ClearErrors (see page 24080)

ImplNotifyDeactivate (see page 24170)

173.1.44.4.9 TCustomAdapter.CreateActionRequestContext Method

Creates an object that describes the adapter action parameters and adapter field values.

Pascal

```
procedure CreateActionRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall CreateActionRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

Child adapter actions call this method.

See Also

IAdapterDispatchParams (see page 19655)

CreateRequestContext (see page 24050)

WebContext (see page 24687)

173.1.44.4.10 TCustomAdapter.CreateErrorsList Method

Creates the object used to manage errors associated with this adapter.

Pascal

```
function CreateErrorsList: TBaseAdapterErrorsList; virtual;
```

C++

```
virtual __fastcall TBaseAdapterErrorsList CreateErrorsList();
```

Description

CreateErrorsList creates TAdapterErrorsList (see page 23810). Descendants may override this function to create a custom errors list.

See Also

Errors (see page 24182)

TAdapterErrorsList (see page 23810)

173.1.44.4.11 TCustomAdapter.EncodeActionParams Method

Writes name/value pairs that represent adapter state.

Pascal

```
procedure EncodeActionParams(AParams: IAdapterItemRequestParams); virtual;
```

C++

```
virtual __fastcall EncodeActionParams(IAdapterItemRequestParams AParams);
```

Description

EncodeActionParams is called by a child adapter actions when generating the name/value pairs that must be sent in the HTTP request when executing an action. For example, TDefaultFieldsPagedAdapter (see page 24403) writes the current page number as a name/value pair. ExtractRequestParams (see page 24153) and EncodeActionParams work together. ExtractRequestParams (see page 24153) reads name/value pairs that are written by EncodeActionParams.

See Also

EncodeActionParams (see page 24411)

ExtractRequestParams (see page 24153)

173.1.44.4.12 TCustomAdapter.EndIterator Method

Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. EndIterator calls ImplEndIterator (see page 24164).

See Also

enditerator (see page 19723)

ImplEndIterator (see page 24164)

173.1.44.4.13 TCustomAdapter.ExtractRequestParams Method

Reads name/value pairs from the HTTP request.

Pascal

```
procedure ExtractRequestParams(ARequest: IUnknown); virtual;
```

C++

```
virtual __fastcall ExtractRequestParams(IUnknown ARequest);
```

Description

ExtractRequestParams is called by a child adapter action before executing the action. This gives the adapter an opportunity to extract values from the HTTP request that are important to the behavior of the adapter. For example, TDefaultFieldsPagedAdapter (see page 24403) extracts the current page number from the request. ExtractRequestParams and EncodeActionParams (see page 24152) work together. ExtractRequestParams reads name/value pairs that are written by EncodeActionParams (see page 24152).

See Also

ExtractRequestParams (see page 24412)

EncodeActionParams (see page 24152)

173.1.44.4.14 TCustomAdapter.FindAction Method

Returns an action component with a matching action name.

Pascal

```
function FindAction(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindAction(const AnsiString AName);
```

Description

The action name is used when a component property references an adapter action by name. Actions (see page 24181) also have a variable name that is used to reference the action with server-side script. The variable name and action name are usually the same. The only time that they need to be different is when the action name is not a valid server-side script identifier.

See Also

FindField (see page 24154)

173.1.44.4.15 TCustomAdapter.FindField Method

Returns an adapter field component with a matching field name.

Pascal

```
function FindField(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindField(const AnsiString AName);
```

Description

The field name is used when a component property references an adapter field by name. Adapter fields also have a variable name that is used to reference the field with server-side script. The variable name and field name are usually the same. The only time that they need to be different is when the field name is not a valid server-side script identifier.

See Also

FindField

173.1.44.4.16 TCustomAdapter.FindNamedAction Method

Finds an action with a matching variable name.

Pascal

```
function FindNamedAction(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindNamedAction(const AnsiString AName);
```

Description

The AName parameter contains the name used when referencing an action with server-side script. This name may not be the same as the component name. For example, server-side script accesses actions in a TDataSetAdapter with using the command name (e.g.; DeleteRow) rather than the adapter action component name.

See Also

FindNamedVariable (see page 24155)

173.1.44.4.17 TCustomAdapter.FindNamedField Method

Finds an adapter field with a matching variable name.

Pascal

```
function FindNamedField(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindNamedField(const AnsiString AName);
```

Description

The AName parameter contains the name used when referencing a field with server-side script. This name may not be the same as the component name. For example, server-side script accesses fields in a TDataSetAdapter with using the dataset field name rather than the adapter field component name.

See Also

FindNamedVariable (see page 24155)

173.1.44.4.18 TCustomAdapter.FindNamedVariable Method

Finds a field or action with a matching variable name.

Pascal

```
function FindNamedVariable(const AName: string): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * FindNamedVariable(const AnsiString AName);
```

Description

FindNamedVariable calls FindNamedField (see page 24154) and FindNamedAction (see page 24154). The AName parameter contains the name used when referencing an action or field with server-side script. This name may not be the same as the component name. For example, server-side script references fields in a TDataSetAdapter using the dataset field name rather than the adapter field component name.

See Also

FindNamedAction (see page 24154)

FindNamedField (see page 24154)

173.1.44.4.19 TCustomAdapter.FindVariable Method

Finds an adapter action or adapter field with the given variable name.

Pascal

```
function FindVariable(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariable(const AnsiString AName);
```

Description

FindVariable is called when server side script references a property of an adapter object. FindVariable calls ImplFindVariable (see page 24164).

See Also

ImplFindVariable (see page 24164)

173.1.44.4.20 TCustomAdapter.FindVariableContainer Method

Finds an adapter action or adapter field that is a container for other scriptable objects.

Pascal

```
function FindVariableContainer(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariableContainer(const AnsiString AName);
```

Description

FindVariableContainer is called when server side script references a property of a property. FindVariableContainer returns the result of ImplFindVariableContainer (see page 24164).

See Also

FindVariable (see page 24155)

ImplFindVariableContainer (see page 24164)

173.1.44.4.21 TCustomAdapter.GetActionsClass Method

Specifies the class to create as a container for adapter actions components.

Pascal

```
function GetActionsClass: TAdapterActionsClass; virtual;
```

C++

```
virtual __fastcall TAdapterActionsClass GetActionsClass();
```

Description

Descendants override this method to specify a different adapter action container.

See Also

Actions (see page 24181)

173.1.44.4.22 TCustomAdapter.GetActionsList Method

Provides a list of action types that can be added to this adapter.

Pascal

```
procedure GetActionsList(AList: TStrings);
```

C++

```
__fastcall GetActionsList(TStrings AList);
```

Description

GetActionsList is called at design time to get a list containing names and objects. Each name identifies an action name to insert and each object identifies a class of adapter action component. When editing the adapter, these action names are displayed in the dialog box that is used to add new actions. For example, when editing a TDataSetAdapter a list of all of the dataset commands are displayed such as NextRow, PrevRow, DeleteRow, etc.

See Also

ImplGetActionsList (see page 24165)

173.1.44.4.23 TCustomAdapter.GetAdapterActions Method

Returns the adapter actions container object.

Pascal

```
function GetAdapterActions: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterActions();
```

Description

Returns the adapter actions container object.

See Also

Actions (see page 24181)

173.1.44.4.24 TCustomAdapter.GetAdapterErrors Method

Pascal

```
function GetAdapterErrors: TObject;
```

C++

```
__fastcall TObject * GetAdapterErrors();
```

Description

This is GetAdapterErrors, a member of class TCustomAdapter.

173.1.44.4.25 TCustomAdapter.GetAdapterErrorsList Method

<insert text here>

Pascal

```
function GetAdapterErrorsList: TBaseAdapterErrorsList;
```

C++

```
__fastcall TBaseAdapterErrorsList GetAdapterErrorsList();
```

Description

Returns an object used to track errors.

See Also

TBaseAdapterErrorsList (see page 24077)

Errors (see page 24182)

173.1.44.4.26 TCustomAdapter.GetAdapterFields Method

Returns the adapter fields container object.

Pascal

```
function GetAdapterFields: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterFields();
```

Description

Returns the adapter fields container object.

See Also

Data (see page 24181)

173.1.44.4.27 TCustomAdapter.GetAdapterHiddenFields Method

Returns a list of hidden fields that hold adapter state information.

Pascal

```
function GetAdapterHiddenFields: TObject;
```

C++

```
__fastcall TObject * GetAdapterHiddenFields();
```

Description

GetAdapterHiddenFields populates a list by calling ListHiddenFields (see page 24173).

See Also

listhiddenfields (see page 24173)

173.1.44.4.28 TCustomAdapter.GetAdapterHiddenRecordFields Method

Returns a list of hidden fields that associate HTML form fields and adapter field.

Pascal

```
function GetAdapterHiddenRecordFields: TObject;
```

C++

```
__fastcall TObject * GetAdapterHiddenRecordFields();
```

Description

GetAdapterHiddenRecordsFields populates a list by calling ListHiddenRecordFields (see page 24173). If the list is empty, nil (Delphi) or NULL (C++) is returned. Otherwise, the list is returned.

See Also

ListHiddenRecordFields (see page 24173)

173.1.44.4.29 TCustomAdapter.GetChildren Method

Provides the interface for a method that adapters use to return their child components.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

TCustomAdapter (see page 24139) overrides the GetChildren method that is introduced in TComponent for the streaming system that loads and saves components. It calls the Proc procedure passing the Actions (see page 24181) and Data (see page 24181) properties. This allows the adapter actions and adapter fields to be loaded and saved with the TCustomAdapter (see page 24139) object.

See Also

Actions (see page 24181)

Data (see page 24181)

173.1.44.4.30 TCustomAdapter.GetDataClass Method

Specifies the class to create as a container for adapter fields components.

Pascal

```
function GetDataClass: TAdapterFieldsClass; virtual;
```

C++

```
virtual __fastcall TAdapterFieldsClass GetDataClass();
```

Description

Descendants override this method to specify a different adapter fields container.

See Also

TAdapterField (see page 23818)

Data (see page 24181)

173.1.44.4.31 TCustomAdapter.GetEchoActionFieldValues Method

Returns the value of the echo flag.

Pascal

```
function GetEchoActionFieldValues: Boolean;
```

C++

```
__fastcall Boolean GetEchoActionFieldValues();
```

Description

EchoActionFieldValues (see page 24182) calls ImplEchoActionFieldValues.

See Also

EchoActionFieldValues (see page 24182)

ImplGetEchoActionFieldValues (see page 24165)

173.1.44.4.32 TCustomAdapter.GetFieldsList Method

Provides a list of field types that can be added to this adapter.

Pascal

```
procedure GetFieldsList(AList: TStrings);
```

C++

```
__fastcall GetFieldsList(TStrings AList);
```

Description

GetFieldsList is called at design time to get a list containing names and objects. Each name identifies a field name to insert and each object identifies a class of adapter field. When editing the adapter, these field names are displayed in the dialog box that is used to add new adapter fields. For example, when editing a TDataSetAdapter a list of all of the dataset's fields are displayed. GetFieldsList calls ImplGetFieldsList (see page 24165).

See Also

ImplGetFieldsList (see page 24165)

173.1.44.4.33 TCustomAdapter.GetHiddenFieldOptions Method

Indicates the types of hidden fields that this adapter requires in a submitted HTML form.

Pascal

```
function GetHiddenFieldOptions: TAdapterHiddenFieldOptions; virtual;
```

C++

```
virtual __fastcall TAdapterHiddenFieldOptions GetHiddenFieldOptions();
```

Description

An adapter may use hidden fields to pass state information down to the client with the HTTP response so that the state information is available when processing a resulting HTTP response. Descendant classes may override this method.

See Also

listhiddenfields (see page 24173)

ListHiddenRecordFields (see page 24173)

173.1.44.4.34 TCustomAdapter.GetIteratorIndex Method

Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.

Pascal

```
function GetIteratorIndex: Integer;
```

C++

```
__fastcall int GetIteratorIndex();
```

Description

Adapters may support multiple data records. For example, TDataSetAdapter provides access to the records in a dataset. GetIteratorIndex calls ImplGetIteratorIndex (see page 24166).

See Also

GetIteratorIndex (see page 19721)

ImplGetIteratorIndex (see page 24166)

173.1.44.4.35 TCustomAdapter.GetVariable Method

Returns an action or field.

Pascal

```
function GetVariable(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetVariable(int I);
```

Description

GetVariableCount (see page 24161) and GetVariable is used to iterate over actions and fields in an adapter. The I parameter is a zero-based index that must be less than the result of GetVariableCount (see page 24161). GetVariable calls ImplGetVariable (see page 24166).

See Also

GetVariable

ImplGetVariable (see page 24166)

GetVariableCount (see page 24161)

173.1.44.4.36 TCustomAdapter.GetVariableCount Method

Returns the total count of actions and fields.

Pascal

```
function GetVariableCount: Integer;
```

C++

```
__fastcall int GetVariableCount();
```

Description

GetVariableCount and GetVariable (see page 24160) is used to iterate over the actions and fields in an adapter. GetVariableCount calls ImplGetVariableCount (see page 24167).

See Also

GetVariable (see page 24160)

ImplGetVariableCount (see page 24167)

173.1.44.4.37 TCustomAdapter.GetVariableName Method

Gets the name used to reference this adapter with server-side script.

Pascal

```
function GetVariableName: string;
```

C++

```
__fastcall AnsiString GetVariableName();
```

Description

GetVariableName is called to retrieve the name used to access this adapter with server-side script. GetVariableName calls ImplGetVariableName (see page 24167).

See Also

ImplGetVariableName (see page 24167)

173.1.44.4.38 TCustomAdapter.GetVisibleActions Method

Pascal

```
function GetVisibleActions: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleActions();
```

Description

This is GetVisibleActions, a member of class TCustomAdapter.

173.1.44.4.39 TCustomAdapter.GetVisibleFields Method

Pascal

```
function GetVisibleFields: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleFields();
```

Description

This is GetVisibleFields, a member of class TCustomAdapter.

173.1.44.4.40 TCustomAdapter.GetWebSnapObjClass Method

Returns the class type for this adapter field.

Pascal

```
function GetWebSnapObjClass: TObjectScripterClass; virtual;
```

C++

```
virtual __fastcall TObjectScripterClass GetWebSnapObjClass();
```

Description

This function returns the class type for this adapter field, which is used by the script engine to create a script object. GetWebSnapObjClass tells the script engine what type of object to create.

173.1.44.4.41 TCustomAdapter.HasExecuteAccess Method

Pascal

```
function HasExecuteAccess: Boolean;
```

C++

```
__fastcall Boolean HasExecuteAccess();
```

Description

This is HasExecuteAccess, a member of class TCustomAdapter.

173.1.44.4.42 TCustomAdapter.HasModifyAccess Method

Pascal

```
function HasModifyAccess: Boolean;
```

C++

```
__fastcall Boolean HasModifyAccess();
```

Description

This is HasModifyAccess, a member of class TCustomAdapter.

173.1.44.4.43 TCustomAdapter.HasViewAccess Method

Checks the end-user's right to view this adapter's fields.

Pascal

```
function HasViewAccess: Boolean;
```

C++

```
__fastcall Boolean HasViewAccess();
```

Description

The value returned by this function can be accessed with server-side script using the CanView property. HasViewAccess calls ImplHasViewAccess (see page 24169).

See Also

ViewAccess (see page 24183)

173.1.44.4.44 TCustomAdapter.ImplCanAddActionClass Method

Indicates whether a type of adapter action can be added to this adapter.

Pascal

```
function ImplCanAddActionClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddActionClass returns false. This means that only TAdapterAction (see page 23726) can be added. Descendants override this method to allow additional field types.

See Also

CanAddActionClass (see page 24150)

ImplCanAddClass (see page 23743)

173.1.44.4.45 TCustomAdapter.ImplCanAddFieldClass Method

Indicates whether a type of adapter field can be added to this adapter.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddFieldClass returns false. This means that only TAdapterField (see page 23818) can be added. Descendants override this method to allow additional field types.

See Also

CanAddFieldClass (see page 24150)

ImplCanAddClass (see page 23836)

173.1.44.4.46 TCustomAdapter.ImplClearAdapterValues Method

Clears adapter state information.

Pascal

```
procedure ImplClearAdapterValues; virtual;
```

C++

```
virtual __fastcall ImplClearAdapterValues();
```

Description

ClearAdapterValues (see page 24151) clears the echo flag.

See Also

ImplSetEchoActionFieldValues (see page 24171)

ClearAdapterValues (see page 24151)

173.1.44.4.47 TCustomAdapter.ImplEndIterator Method

Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer); virtual;
```

C++

```
virtual __fastcall ImplEndIterator(void * OwnerData);
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. ImplEndIterator sets the ItemIndex to 0 and the iterator flag to false.

See Also

enditerator (see page 19723)

EndIterator (see page 24152)

173.1.44.4.48 TCustomAdapter.ImplFindVariable Method

Finds an adapter action or adapter field child with a matching variable name.

Pascal

```
function ImplFindVariable(const AName: string): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplFindVariable(const AnsiString AName);
```

Description

ImpFindVariable returns the result of FindNamedVariable (see page 24155).

See Also

FindVariable (see page 24155)

173.1.44.4.49 TCustomAdapter.ImplFindVariableContainer Method

Finds an adapter action or adapter field that is a container for other scriptable objects.

Pascal

```
function ImplFindVariableContainer(const AName: string): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplFindVariableContainer(const AnsiString AName);
```

Description

FindVariableContainer (see page 24155) is called when server side script references a property of a property. ImplFindVariableContainer tries to find an adapter field or adapter action that is a container for other objects.

See Also

FindVariableContainer (see page 24155)

173.1.44.4.50 TCustomAdapter.ImplGetActionsList Method

Provides a list of action types that can be added to this adapter.

Pascal

```
procedure ImplGetActionsList(AList: TStrings); virtual;
```

C++

```
virtual __fastcall ImplGetActionsList(TStrings AList);
```

Description

ImplGetActionsList is called by GetActionsList (see page 24156). ImplGetActionsList provides an empty list. ImplGetActionsList may be overridden by descendant classes.

See Also

GetActionsList (see page 24156)

173.1.44.4.51 TCustomAdapter.ImplGetEchoActionFieldValues Method

Returns the value of the echo flag.

Pascal

```
function ImplGetEchoActionFieldValues: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetEchoActionFieldValues();
```

Description

ImplGetEchoActionFieldValues returns the value passed to ImplSetEchoActionFieldValues (see page 24171). If ImplSetEchoActionFieldValues (see page 24171) has not been processed during the current HTTP request then False is returned.

See Also

EchoActionFieldValues (see page 24182)

GetEchoActionFieldValues (see page 24159)

173.1.44.4.52 TCustomAdapter.ImplGetFieldsList Method

Provides a list of field types that can be added to this adapter.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); virtual;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

ImplGetFieldsList is called by GetFieldsList (see page 24159). ImplGetFieldsList provides an empty list. ImplGetFieldsList may be overridden by descendant classes.

See Also

GetFieldsList (see page 24159)

173.1.44.4.53 TCustomAdapter.ImplGetIteratorIndex Method

Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.

Pascal

```
function ImplGetIteratorIndex: Integer; virtual;
```

C++

```
virtual __fastcall int ImplGetIteratorIndex();
```

Description

ImplGetIteratorIndex returns the value ItemIndex. ItemIndex is modified by ImplStartIterator (see page 24171), ImplNextIteration (see page 24169), and ImplEndIterator (see page 24164).

See Also

GetIteratorIndex (see page 19721)

GetIteratorIndex (see page 24160)

ImplStartIterator (see page 24171)

ImplNextIteration (see page 24169)

ImplEndIterator (see page 24164)

173.1.44.4.54 TCustomAdapter.ImplGetVariable Method

Returns an action or field.

Pascal

```
function ImplGetVariable(I: Integer): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplGetVariable(int I);
```

Description

ImplGetVariable is called by GetVariable (see page 24160). ImplGetVariable uses the I parameter to locate an adapter action or adapter field.

See Also

GetVariable (see page 24160)

173.1.44.4.55 TCustomAdapter.ImplGetVariableCount Method

Returns the total count of actions and fields.

Pascal

```
function ImplGetVariableCount: Integer; virtual;
```

C++

```
virtual __fastcall int ImplGetVariableCount();
```

Description

ImplGetVariableCount is called by GetVariableCount (see page 24161).

See Also

GetVariableCount (see page 24161)

173.1.44.4.56 TCustomAdapter.ImplGetVariableName Method

Gets the name used to reference this adapter with server-side script.

Pascal

```
function ImplGetVariableName: string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplGetVariableName();
```

Description

ImplGetVariableName returns the component name.

See Also

GetVariableName (see page 24161)

173.1.44.4.57 TCustomAdapter.ImplGetVisibleActions Method

Returns a list of this adapter's action components.

Pascal

```
function ImplGetVisibleActions: TWebComponentList; virtual;
```

C++

```
virtual __fastcall TWebComponentList ImplGetVisibleActions();
```

Description

ImplGetVisibleActions is called by GetVisibleActions (see page 24161). ImplGetVisibleActions may be overriden by descendant classes.

See Also

getvisibleactions (see page 24161)

173.1.44.4.58 TCustomAdapter.ImplGetVisibleFields Method

Returns a list of this adapter's field components.

Pascal

```
function ImplGetVisibleFields: TWebComponentList; virtual;
```

C++

```
virtual __fastcall TWebComponentList ImplGetVisibleFields();
```

Description

ImplGetVisibleFields is called by GetVisibleFields (see page 24161). ImplGetVisibleFields may be overridden by descendant classes.

See Also

getvisiblefields (see page 24161)

173.1.44.4.59 TCustomAdapter.ImplHasExecuteAccess Method

Checks the end-user's right to execute this adapter's actions.

Pascal

```
function ImplHasExecuteAccess: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasExecuteAccess();
```

Description

This function is called by HasExecuteAccess (see page 24162) to check the end-user's right to execute the adapter's actions. If the ExecuteAccess (see page 24183) property is not blank, this function calls the EndUserAdapter's HasExecuteRights function, passing the ExecuteAccess (see page 24183) property value.

See Also

hasexecuteaccess (see page 24162)

executeaccess (see page 24183)

HasExecuteAccess (see page 24062)

173.1.44.4.60 TCustomAdapter.ImplHasModifyAccess Method

Checks the end-user's right to modify this adapter's fields.

Pascal

```
function ImplHasModifyAccess: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasModifyAccess();
```

Description

This function is called by HasModifyAccess (see page 24162) to check the end-user's right to modify the adapter field. If the ModifyAccess (see page 24183) property is not blank, this function calls the EndUserAdapter HasModifyRights function, passing the ModifyAccess (see page 24183) property value.

See Also

hasmodifyaccess (see page 24162)

modifyaccess (see page 24183)

hasmodifyaccess (see page 24162)

173.1.44.4.61 TCustomAdapter.ImplHasViewAccess Method

Pascal

```
function ImplHasViewAccess: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasViewAccess();
```

Description

This is ImplHasViewAccess, a member of class TCustomAdapter.

173.1.44.4.62 TCustomAdapter.ImplInIterator Method

Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.

Pascal

```
function ImplInIterator: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplInIterator();
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. ImplGetInIterator returns the value of the iteration flag. The iteration flag is set to true by TCustomAdapter (see page 24139)'s ImplStartIterator (see page 24171) method.

See Also

InIterator (see page 19721)

InIterator (see page 24172)

ImplStartIterator (see page 24171)

173.1.44.4.63 TCustomAdapter.ImplNextIteration Method

Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. ImplNextIteration increments ItemIndex and returns true if the iteration index is less than ItemCount.

See Also

NextIteration (see page 19723)

NextIteration (see page 24173)

ImplStartIterator (see page 24171)

ImplEndIterator (see page 24164)

173.1.44.4.64 TCustomAdapter.ImplNotifyActivate Method

Called when the web module is activated.

Pascal

```
procedure ImplNotifyActivate; virtual;
```

C++

```
virtual __fastcall ImplNotifyActivate();
```

Description

A web module is activated when an HTTP request is received. ImplNotifyActivate is a placeholder. Descendant classes override this method to initialize the adapter before an HTTP request processed.

See Also

NotifyActivate (see page 24174)

173.1.44.4.65 TCustomAdapter.ImplNotifyDeactivate Method

Called when the web module is deactivated.

Pascal

```
procedure ImplNotifyDeactivate; virtual;
```

C++

```
virtual __fastcall ImplNotifyDeactivate();
```

Description

A web module is deactivated after the HTTP request has been handled.. ImplNotifyDeactivate discards adapter state information including adapter errors, iteration flag, and iteration index.

See Also

NotifyDeactivate (see page 24175)

173.1.44.4.66 TCustomAdapter.ImplSetActions Method

Initializes the Actions (see page 24181) property.

Pascal

```
procedure ImplSetActions(const Value: TAdapterSubComponent); virtual;
```

C++

```
virtual __fastcall ImplSetActions(const TAdapterSubComponent Value);
```

Description

The Actions (see page 24181) property is the container for adapter action components. Descendants may override this method.

See Also

Actions (see page 24181)

SetActions (see page 24175)

173.1.44.4.67 TCustomAdapter.ImplSetData Method

Initializes the Data (see page 24181) property.

Pascal

```
procedure ImplSetData(const Value: TAdapterSubComponent); virtual;
```

C++

```
virtual __fastcall ImplSetData(const TAdapterSubComponent Value);
```

Description

The Data (see page 24181) property is the container for adapter field components. Descendants may override this method.

See Also

Data (see page 24181)

SetData (see page 24175)

173.1.44.4.68 TCustomAdapter.ImplSetEchoActionFieldValues Method

Sets the echo flag for this adapter's fields.

Pascal

```
procedure ImplSetEchoActionFieldValues(AValue: Boolean); virtual;
```

C++

```
virtual __fastcall ImplSetEchoActionFieldValues(Boolean AValue);
```

Description

ImplEchoActionFieldValues modified the EchoActionFieldValue property of the adapter's fields.

See Also

GetEchoActionFieldValues (see page 24159)

SetEchoActionFieldValues (see page 24176)

EchoActionFieldValue (see page 23944)

173.1.44.4.69 TCustomAdapter.ImplStartIterator Method

Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. ImplStartIterator sets ItemIndex to 0 and sets the iteration flag to true. true is returned.

See Also

startiterator (see page 19724)

StartIterator (see page 24176)

ImplNextIteration (see page 24169)

ImplEndIterator (see page 24164)

173.1.44.4.70 TCustomAdapter.InIterator Method

Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.

Pascal

```
function InIterator: Boolean;
```

C++

```
__fastcall Boolean InIterator();
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. InIterator calls ImplInIterator (see page 24169).

See Also

InIterator (see page 19721)

ImplInIterator (see page 24169)

173.1.44.4.71 TCustomAdapter.IsActionInUse Method

Indicates whether a particular action has been added to the adapter.

Pascal

```
function IsActionInUse(const AName: string): Boolean;
```

C++

```
__fastcall Boolean IsActionInUse(const AnsiString AName);
```

Description

IsActionInUse is called at design time to determine whether an action with a particular name has been added to the adapter. When editing the adapter, the names of actions in use are not displayed in the dialog box used to add new actions.

See Also

IsFieldInUse (see page 24172)

GetFieldsList (see page 24159)

173.1.44.4.72 TCustomAdapter.IsFieldInUse Method

Indicates whether a particular field has been added to the adapter.

Pascal

```
function IsFieldInUse(const AName: string): Boolean;
```

C++

```
__fastcall Boolean IsFieldInUse(const AnsiString AName);
```

Description

IsFieldInUse is called at design time to determine whether a field with a particular name has been added to the adapter. When editing the adapter, the names of these fields if use are not displayed in the dialog box used to add new fields.

See Also

IsActionInUse (see page 24172)

GetFieldsList (see page 24159)

173.1.44.4.73 TCustomAdapter.ListHiddenFields Method

Pascal

```
procedure ListHiddenFields(AHiddenFields: TAdapterHiddenFieldsList);
```

C++

```
__fastcall ListHiddenFields(TAdapterHiddenFieldsList AHiddenFields);
```

Description

This is ListHiddenFields, a member of class TCustomAdapter.

173.1.44.4.74 TCustomAdapter.ListHiddenRecordFields Method

Specifies the HTML form hidden fields needed by the adapter.

Pascal

```
procedure ListHiddenRecordFields(AHiddenFields: TAdapterHiddenFieldsList);
```

C++

```
__fastcall ListHiddenRecordFields(TAdapterHiddenFieldsList AHiddenFields);
```

Description

Some adapters may use hidden fields to define the relationship between input fields in the HTML form and data of the adapter. For example, an HTML form that contains an HTML table for updating 5 detail records of a TDataSetAdapter will have 5 sets of hidden fields. Each set provides state information needed by the adapter to determine the relationship between the input fields in the HTTP request and the dataset.

ListHiddenRecordFields may call AddOrigValueHiddenFields (see page 24149) and/or AddRecordKeysHiddenFields (see page 24149) depending on the flags returned by GetHiddenFieldOptions (see page 24159).

See Also

GetHiddenFieldOptions (see page 24159)

TAdapterHiddenFieldsList (see page 23862)

ListHiddenRecordFields

AddOrigValueHiddenFields (see page 24149)

AddRecordKeysHiddenFields (see page 24149)

GetAdapterHiddenRecordFields (see page 24158)

173.1.44.4.75 TCustomAdapter.NextIteration Method

Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. NextIteration calls ImplNextIteration (see page 24169).

See Also

NextIteration (see page 19723)

ImplNextIteration (see page 24169)

173.1.44.4.76 TCustomAdapter.Notification Method

Clears references to deleted components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called when a component is inserted or deleted.

173.1.44.4.77 TCustomAdapter.NotifyActivate Method

Called when the web module is activated.

Pascal

```
procedure NotifyActivate;
```

C++

```
__fastcall NotifyActivate();
```

Description

A web module is activated when an HTTP request is received. NotifyActivate calls ImplNotifyActivate (see page 24170).

See Also

ImplNotifyActivate (see page 24170)

NotifyDeactivate (see page 24175)

173.1.44.4.78 TCustomAdapter.NotifyAdapterChange Method

Notifies components that the adapter has changed in some way.

Pascal

```
procedure NotifyAdapterChange;
```

C++

```
__fastcall NotifyAdapterChange();
```

Description

NotifyAdapterChange calls components that have registered a notification callback to let them know that the adapter has changed. For example, TAdapterFieldGroup needs to know when an adapter field is added to the adapter component that the TAdapterFieldGroup references.

See Also

NotifyList (see page 24183)

173.1.44.4.79 TCustomAdapter.NotifyDeactivate Method

Called when the web module is deactivated.

Pascal

```
procedure NotifyDeactivate;
```

C++

```
__fastcall NotifyDeactivate();
```

Description

A web module is deactivated after the HTTP request has been handled.. NotifyDeactivate calls ImplNotifyDeactivate (see page 24170).

See Also

ImplNotifyDeactivate (see page 24170)

NotifyActivate (see page 24174)

173.1.44.4.80 TCustomAdapter.SetActions Method

Initializes the Actions (see page 24181) property.

Pascal

```
procedure SetActions(const Value: TAdapterSubComponent);
```

C++

```
__fastcall SetActions(const TAdapterSubComponent Value);
```

Description

The Actions (see page 24181) property is the container for adapter action components. SetActions calls ImplSetActions (see page 24170).

See Also

Actions (see page 24181)

ImplSetActions (see page 24170)

173.1.44.4.81 TCustomAdapter.SetData Method

Initializes the Data (see page 24181) property.

Pascal

```
procedure SetData(const Value: TAdapterSubComponent); virtual;
```

C++

```
virtual __fastcall SetData(const TAdapterSubComponent Value);
```

Description

The Data (see page 24181) property is the container for adapter field components. SetData calls ImplSetData (see page 24171).

See Also

Data (see page 24181)

ImplSetData (see page 24171)

173.1.44.4.82 TCustomAdapter.SetEchoActionFieldValues Method

Sets the echo flag for this adapter's fields.

Pascal

```
procedure SetEchoActionFieldValues(AValue: Boolean);
```

C++

```
__fastcall SetEchoActionFieldValues(Boolean AValue);
```

Description

SetEchoActionFieldValues calls ImplSetActionFieldValues.

See Also

EchoActionFieldValues (see page 24182)

ImplSetEchoActionFieldValues (see page 24171)

173.1.44.4.83 TCustomAdapter.StartIterator Method

Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. StartIterator calls ImplStartIterator (see page 24171).

See Also

startiterator (see page 19724)

ImplStartIterator (see page 24171)

173.1.44.4.84 TCustomAdapter.UpdateFieldsExecuteValues Method

Executes the adapter's fields with values passed in the HTTP request.

Pascal

```
procedure UpdateFieldsExecuteValues(AActionRequest: IActionRequest; AAdapterFields:
TObjectList); virtual;
```

C++

```
virtual __fastcall UpdateFieldsExecuteValues(IActionRequest AActionRequest, TObjectList
AAdapterFields);
```

Description

UpdateFieldsExecuteValues loops through the adapter fields passed in the AAdapterFields parameter, calling the ExecuteValue function. Adapter fields raise an exception to indicate an error. If an exception occurs, the exception message is added to the Errors (see page 24182) object and execution continues.

When a field is executable, the field value passed in the HTTP request is used to execute a command rather than update a value. For example, an HTML form may contain a table with a delete row checkbox on each row. When the form is submitted, the values in the HTTP request indicate whether to delete one or more rows. Executable adapter fields must implement the IExecuteValue (see page 24575) interface.

See Also

Errors (see page 24182)

IActionRequest (see page 935)

IExecuteValue (see page 24575)

173.1.44.4.85 TCustomAdapter.UpdateFieldsGetAnyChanges Method

Determines whether the values in the HTTP request change any adapter fields.

Pascal

```
function UpdateFieldsGetAnyChanges(AActionRequest: IActionRequest; AAdapterFields:
TObjectList): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean UpdateFieldsGetAnyChanges(IActionRequest AActionRequest,
TObjectList AAdapterFields);
```

Description

TAdapterUpdateField (see page 24007)'s CheckValueChange method loops through the adapter fields passed in the AAdapterFields parameter, calling the CheckValueChange function. UpdateFieldsGetAnyChanges will return true if any adapter field returns true when its CheckValueChange method is called.

See Also

Errors (see page 24182)

ValidateValue (see page 24022)

IActionRequest (see page 935)

CheckValueChange (see page 24015)

173.1.44.4.86 TCustomAdapter.UpdateFieldsUpdateValues Method

Updates the adapter's fields with values passed in the HTTP request.

Pascal

```
procedure UpdateFieldsUpdateValues(AActionRequest: IActionRequest; AAdapterFields:
TObjectList); virtual;
```

C++

```
virtual __fastcall UpdateFieldsUpdateValues(IActionRequest AActionRequest, TObjectList
AAdapterFields);
```

Description

UpdateFieldsUpdateValues loops through the adapter fields passed in the AAdapterFields parameter, calling the UpdateValue function. Adapter fields raise an exception to indicate a update error. If an exception occurs, the exception message is added to the Errors (see page 24182) object and updating continues. OnBeforeUpdateFields (see page 24187) and OnAfterUpdateFields (see page 24185) are at the beginning and end of this method.

See Also

OnBeforeUpdateFields (see page 24187)

OnAfterUpdateFields (see page 24185)

Errors (see page 24182)

UpdateValue (see page 24021)

IActionRequest (see page 935)

173.1.44.4.87 TCustomAdapter.UpdateFieldsValidateValues Method

Validate the adapter's field values passed in the HTTP request.

Pascal

```
procedure UpdateFieldsValidateValues(AActionRequest: IActionRequest; AAdapterFields:
TObjectList); virtual;
```

C++

```
virtual __fastcall UpdateFieldsValidateValues(IActionRequest AActionRequest, TObjectList
AAdapterFields);
```

Description

UpdateFieldsValidateValues loops through the adapter fields passed in the AAdapterFields parameter, calling the ValidateValue function. Adapter fields raise an exception to indicate a validation error. When an exception occurs, th exception messages are added to the Errors (see page 24182) object and validation continues. OnBeforeValidateFields (see page 24187) and OnAfterValidateFields (see page 24186) are at the beginning and end of this method.

See Also

OnBeforeValidateFields (see page 24187)

OnAfterValidateFields (see page 24186)

Errors (see page 24182)

ValidateValue (see page 24022)

IActionRequest (see page 935)

173.1.44.4.88 TCustomAdapter.UpdateRecords Method

Updates the adapter's fields with the field values passed in an HTTP request.

Pascal

```
function UpdateRecords(AActionRequest: IActionRequest = nil): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean UpdateRecords(IActionRequest AActionRequest = nil);
```

Description

Call UpdateRecords to update the adapter's fields with the field values passed in an HTTP request. Typically, UpdateRecords will be called by an adapter action that submits an HTML form. UpdatesRecords first verifies that the end-user has rights to modify the adapter by calling CheckModifyAccess (see page 24150). CheckModifyAccess (see page 24150) will raise an exception if access is denied. Next, UpdateRecordsGetAnyChanges (see page 24179) is called to determine whether the values in the HTTP request represent changes to the adapter fields. If there are changes then UpdateRecordsValidateValues (see page 24180) is called followed by UpdateRecordsUpdateValues (see page 24180). Exceptions raised during validation or update are added to the adapter's errors object. UpdateRecords returns true if the errors object contains no errors. The AActionRequest parameter is optional.

See Also

IActionRequest (see page 935)

checkmodifyaccess (see page 24150)

UpdateRecordsGetAnyChanges (see page 24179)

UpdateRecordsValidateValues (see page 24180)

UpdateRecordsUpdateValues (see page 24180)

Errors (see page 24182)

173.1.44.4.89 TCustomAdapter.UpdateRecordsGetAnyChanges Method

Determines whether values passed in the HTTP request will change any adapter fields.

Pascal

```
function UpdateRecordsGetAnyChanges(AActionRequest: IActionRequest; AAdapterFields:
TObjectList): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean UpdateRecordsGetAnyChanges(IActionRequest AActionRequest,
TObjectList AAdapterFields);
```

Description

The HTTP request may contain values that are associated with multiple adapter records. For example, an HTML form that contains a grid with input fields on each row will generate an HTTP request to update multiple records. UpdateRecordsGetAnyChanges iterates all the records in the HTTP request, calling UpdateFieldsGetAnyChanges (see page 24177) for each record.

See Also

UpdateFieldsGetAnyChanges (see page 24177)

173.1.44.4.90 TCustomAdapter.UpdateRecordsUpdateValues Method

Updates adapter fields with values passed in the HTTP request.

Pascal

```
procedure UpdateRecordsUpdateValues(AActionRequest: IActionRequest; AAdapterFields:
TObjectList); virtual;
```

C++

```
virtual __fastcall UpdateRecordsUpdateValues(IActionRequest AActionRequest, TObjectList
AAdapterFields);
```

Description

The HTTP request may contain values that are associated with multiple adapter records. For example, an HTML form that contains a grid with input fields on each row will generate an HTTP request to update multiple records. UpdateRecordsUpdateValues iterates all the records in the HTTP request, calling UpdateFieldUpdateValues for each record.

See Also

UpdateFieldsUpdateValues (see page 24178)

173.1.44.4.91 TCustomAdapter.UpdateRecordsValidateValues Method

Validate the adapter's field values passed in the HTTP request.

Pascal

```
procedure UpdateRecordsValidateValues(AActionRequest: IActionRequest; AAdapterFields:
TObjectList); virtual;
```

C++

```
virtual __fastcall UpdateRecordsValidateValues(IActionRequest AActionRequest, TObjectList
AAdapterFields);
```

Description

UpdateFieldsValidateValues (see page 24178) loops through the adapter fields passed in the AAdapterFields parameter, calling the ValidateValue function. Adapter fields raise an exception to indicate a validation error. When an exception occurs, the exception message is added to the Errors (see page 24182) object and validation continues. OnBeforeValidateFields (see page 24187) and OnAfterValidateFields (see page 24186) are at the beginning and end of this method.

See Also

OnBeforeValidateFields (see page 24187)

OnAfterValidateFields (see page 24186)

Errors (see page 24182)

ValidateValue (see page 24022)

IActionRequest (see page 935)

173.1.44.4.92 TCustomAdapter.WebComponentsChange Method

WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Pascal

```
procedure WebComponentsChange(Sender: TObject);
```

C++

```
__fastcall WebComponentsChange(TObject * Sender);
```

Description

WebComponentsChange will be called, for example, when an adapter field is added to the adapter or when an adapter field is removed from the adapter. WebComponentsChange calls NotifyAdapterChange (see page 24174).

See Also

NotifyAdapterChange (see page 24174)

173.1.44.5 TCustomAdapter Properties

The properties of the TCustomAdapter class are listed here.

Protected Properties

	Name	Description
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Properties

	Name	Description
R	Actions (see page 24181)	Actions is the container for adapter actions components.
R	Data (see page 24181)	Data is the container for adapter field components.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.

173.1.44.5.1 TCustomAdapter.Actions Property

Actions is the container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterActions (see page 23738)

173.1.44.5.2 TCustomAdapter.Data Property

Data is the container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterFields (see page 23830)

173.1.44.5.3 TCustomAdapter.EchoActionFieldValues Property

Modifies the adapter's echo flag.

Pascal

```
property EchoActionFieldValues: Boolean;
```

C++

```
__property Boolean EchoActionFieldValues;
```

Description

The echo flag is used to alter the value returned by the adapter's fields. When this flag is set, the adapter's fields retrieves their value from the HTTP request. When this flag is clear, the adapter's fields retrieve their current value. Typically, this flag is set after an error occurs while executing an adapter action. Setting this flag results in generated HTML that displays values entered by the end-user. This allows the end-user to make corrections and resubmit the form.

See Also

SetEchoActionFieldValues (see page 24176)

GetEchoActionFieldValues (see page 24159)

173.1.44.5.4 TCustomAdapter.Errors Property

Returns an object used to track errors on behalf of this adapter.

Pascal

```
property Errors: TBaseAdapterErrorsList;
```

C++

```
__property TBaseAdapterErrorsList Errors;
```

Description

Adapter actions typically catch exceptions raised during execution and add the exception messages to the parent adapter's errors object. Server-side script may enumerate the errors recorded by the Errors property in order to write error messages to an HTML page.

See Also

GetAdapterErrorsList (see page 24157)

TBaseAdapterErrorsList (see page 24077)

TAdapterErrorsList (see page 23810)

173.1.44.5.5 TCustomAdapter.ExecuteAccess Property

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property AnsiString ExecuteAccess;
```

Description

This is ExecuteAccess, a member of class TCustomAdapter.

173.1.44.5.6 TCustomAdapter.ModifyAccess Property

Pascal

```
property ModifyAccess: string;
```

C++

```
__property AnsiString ModifyAccess;
```

Description

This is ModifyAccess, a member of class TCustomAdapter.

173.1.44.5.7 TCustomAdapter.NotifyList Property

Keeps track of components that are to be notified when the adapter is changed.

Pascal

```
property NotifyList: TNotifyList;
```

C++

```
__property TNotifyList NotifyList;
```

Description

Keeps track of components that are to be notified when the adapter is changed.

See Also

NotifyAdapterChange (see page 24174)

TNotifyList (see page 19636)

173.1.44.5.8 TCustomAdapter.ViewAccess Property

Identifies the rights required to view this adapter's fields.

Pascal

```
property ViewAccess: string;
```

C++

```
__property AnsiString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view fields of this adapter. When server-side script accesses the CanView property, the ViewAccess property value is checked against the end user's rights.

See Also

TWebUserList (see page 25625)

HasViewAccess (see page 24162)

implhasviewaccess (see page 24169)

173.1.44.5.9 TCustomAdapter.VisibleActions Property

Pascal

```
property VisibleActions: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleActions;
```

Description

This is VisibleActions, a member of class TCustomAdapter.

173.1.44.5.10 TCustomAdapter.VisibleFields Property

Pascal

```
property VisibleFields: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleFields;
```

Description

This is VisibleFields, a member of class TCustomAdapter.

173.1.44.6 TCustomAdapter Events

The events of the TCustomAdapter class are listed here.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 24184)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24185)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24186)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24186)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 24187)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.

173.1.44.6.1 TCustomAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

173.1.44.6.2 TCustomAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.44.6.3 TCustomAdapter.OnAfterUpdateFields Event

Occurs after values in the HTTP request are used to update adapter fields.

Pascal

```
property OnAfterUpdateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterUpdateFields;
```

Description

Implement this event handler to perform custom updates. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record update errors.

See Also

UpdateFieldsUpdateValues (see page 24178)

OnBeforeUpdateFields (see page 24187)

Errors (see page 24182)

173.1.44.6.4 TCustomAdapter.OnAfterValidateFields Event

Occurs after values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnAfterValidateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record validation errors.

See Also

UpdateFieldsValidateValues (see page 24178)

OnBeforeValidateFields (see page 24187)

Errors (see page 24182)

173.1.44.6.5 TCustomAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 24076)

OnAfterExecuteAction (see page 24184)

173.1.44.6.6 TCustomAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to

indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 24076)

OnAfterGetActionResponse (see page 24185)

173.1.44.6.7 TCustomAdapter.OnBeforeUpdateFields Event

Occurs before values in the HTTP request are used to update adapter fields.

Pascal

```
property OnBeforeUpdateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeUpdateFields;
```

Description

Implement this event handler to perform custom updates of adapter fields. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further updates. Add update errors to the adapter's errors object.

See Also

UpdateFieldsUpdateValues (see page 24178)

OnAfterUpdateFields (see page 24185)

Errors (see page 24182)

173.1.44.6.8 TCustomAdapter.OnBeforeValidateFields Event

Occurs before values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnBeforeValidateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further validation. Add validation errors to the adapter's errors object.

See Also

UpdateFieldsValidateValues (see page 24178)

OnAfterValidateFields (see page 24186)

Errors (see page 24182)

173.1.44.6.9 TCustomAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 24077)

173.1.44.6.10 TCustomAdapter.OnIterateRecords Event

Occurs when the adapter needs to iterate over its data records.

Pascal

```
property OnIterateRecords: TIterateRecordsEvent;
```

C++

```
__property TIterateRecordsEvent OnIterateRecords;
```

Description

Implement this event when the adapter supports multiple data records. This event handler is called when server-side script iterates over the records in adapter, such as when generating an HTML table displaying all records. The event handler must update an iteration pointer or cursor that adapter fields are also aware of. Adapter fields must use this iteration pointer when called to retrieve their value.

See Also

IIteratorSupport (see page 19722)

173.1.45 TCustomAdapterAction Class

TCustomAdapterAction is a base class for adapter actions.

Class Hierarchy

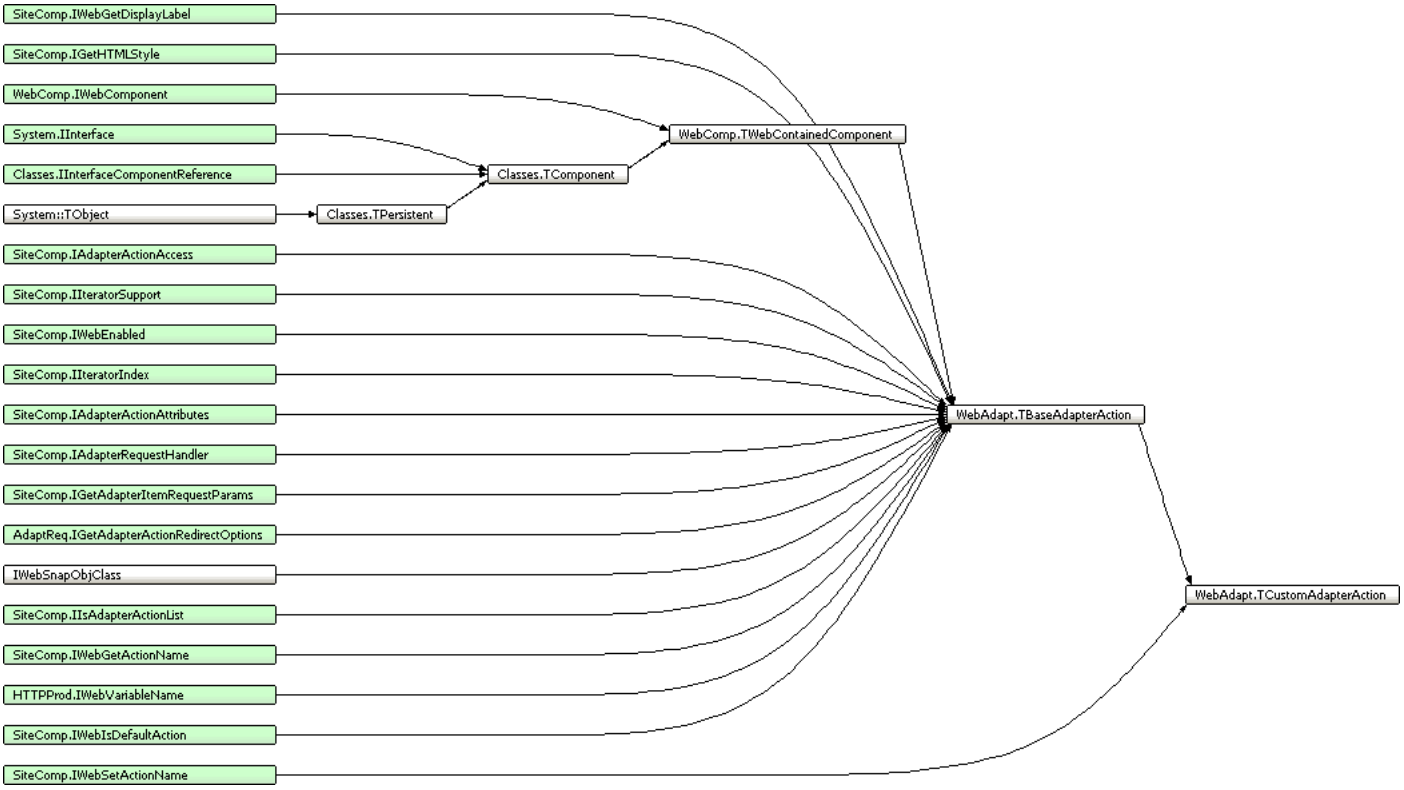

Pascal

```
TCustomAdapterAction = class(TBaseAdapterAction, IWebSetActionName);
```

C++

```
class TCustomAdapterAction : public TBaseAdapterAction, public IWebSetActionName;
```

File

WebAdapt

Description

TCustomAdapterAction calls event handlers to retrieve the adapter action parameters and execute the adapter action. Derive from TCustomAdapterAction to define an adapter action component that can be added to any adapter. Derive from TImplementAdapterAction to define an adapter action component that works with specific adapter component types.

See Also

TBaseAdapterAction (see page 24043)

TAdapter (see page 23713)

TAdapterAction (see page 23726)

TImplementedAdapterAction (see page 24491)

173.1.45.1 TCustomAdapterAction Members

The following tables list the members exposed by TCustomAdapterAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 24055)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.

	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 24195)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 24197)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.

	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 24198)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 24199)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24199)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24200)	Called before the adapter action response is generated.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 24077)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.

	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 24073)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 24073)	Defines the rights required to execute this action.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.

R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RedirectOptions (see page 24074)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.45.2 TCustomAdapterAction Methods

The methods of the TCustomAdapterAction class are listed here.

Protected Methods

	Name	Description
V	ImplExecuteActionRequest (see page 24195)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
V	ImplWebEnabled (see page 24197)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.

173.1.45.2.1 TCustomAdapterAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest create a TStrings containing the name/value pairs in AActionRequest and calls the OnExecute (see page 24200) event handler.

See Also

ImplExecuteActionRequest (see page 24063)

OnExecute (see page 24200)

173.1.45.2.2 TCustomAdapterAction.ImplGetActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function ImplGetActionName: string; override;
```

C++

```
virtual __fastcall AnsiString ImplGetActionName();
```

Description

Gets the name used to reference this adapter action with server-side script.

See Also

ActionName (see page 24198)

ImplGetActionName (see page 24063)

173.1.45.2.3 TCustomAdapterAction.ImplGetAdapterItemRequestParams Method

Provides the name/value pairs needed to execute this adapter action.

Pascal

```
procedure ImplGetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams); override;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(AnsiString AIdentifier,
IAdapterItemRequestParams Params);
```

Description

When this adapter action is executed, it provides name/value pairs which will be written when the HTML page is generated and passed back in the HTTP request. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted. When the HTML page is generated a hyperlink or button is written that identifies the name and parameters of the action. When the hyperlink or button is clicked the name of the action and parameters are passed back to the web application in the HTTP request.

See Also

ImplGetAdapterItemRequestParams (see page 24063)

173.1.45.2.4 TCustomAdapterAction.ImplGetSuccessOptions Method

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
function ImplGetSuccessOptions: TAdapterActionRedirectOptions; override;
```

C++

```
virtual __fastcall TAdapterActionRedirectOptions ImplGetSuccessOptions();
```

Description

ImplGetSuccessOptions returns the value of the RedirectOptions (see page 24074) property.

See Also

RedirectOptions (see page 24074)

173.1.45.2.5 TCustomAdapterAction.ImplSetActionName Method

Sets the name that is used to reference this adapter action with server-side script.

Pascal

```
procedure ImplSetActionName(const Value: string);
```

C++

```
__fastcall ImplSetActionName(const AnsiString Value);
```

Description

ImplSetActionName is called by SetActionName (see page 24197). ImplSetActionName stores the name in an instance variable.

See Also

SetActionName (see page 24197)

173.1.45.2.6 TCustomAdapterAction.ImplWebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

ImplWebEnabled calls the OnGetEnabled (see page 24200) event handler.

See Also

ImplWebEnabled (see page 24069)

OnGetEnabled (see page 24200)

173.1.45.2.7 TCustomAdapterAction.SetActionName Method

Sets the name that is used to reference this adapter action with server-side script.

Pascal

```
procedure SetActionName(const Value: string);
```

C++

```
__fastcall SetActionName(const AnsiString Value);
```

Description

SetActionName is typically called at design time to set the action name of a new adapter action. SetActionName calls ImplSetActionName (see page 24197).

See Also

ImplSetActionName (see page 24197)

173.1.45.3 TCustomAdapterAction Properties

The properties of the TCustomAdapterAction class are listed here.

Public Properties

	Name	Description
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.

173.1.45.3.1 TCustomAdapterAction.ActionName Property

Identifies the name of this action when using server-side script.

Pascal

```
property ActionName: string;
```

C++

```
__property AnsiString ActionName;
```

Description

The ActionName property defines the name used to access the adapter action with server-side script.

See Also

ImplGetActionName (see page 24195)

ImplSetActionName (see page 24197)

173.1.45.4 TCustomAdapterAction Events

The events of the TCustomAdapterAction class are listed here.

Public Events

	Name	Description
	OnAfterExecute (see page 24198)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 24199)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24199)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24200)	Called before the adapter action response is generated.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

173.1.45.4.1 TCustomAdapterAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action

parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

173.1.45.4.2 TCustomAdapterAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.45.4.3 TCustomAdapterAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

173.1.45.4.4 TCustomAdapterAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.45.4.5 TCustomAdapterAction.OnExecute Event

Called to execute this action using parameters passed in the HTTP request.

Pascal

```
property OnExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnExecute;
```

Description

Implement this event handler to execute this adapter action. Params contains the name/value pairs that were passed in the HTTP request. OnExecute is called by ImplExecuteActionRequest (see page 24195).

See Also

ImplExecuteActionRequest (see page 24195)

OnGetParams (see page 24077)

173.1.45.4.6 TCustomAdapterAction.OnGetEnabled Event

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
property OnGetEnabled: TActionGetEnabledEvent;
```

C++

```
__property TActionGetEnabledEvent OnGetEnabled;
```

Description

ImplWebEnabled (see page 24197) calls the OnGetEnabled event handler. Set the Enabled parameter to false to indicate that

this adapter action should not be enabled.

See Also

ImplWebEnabled (see page 24197)

173.1.46 TCustomAdapterBooleanField Class

tcustomadapterbooleanfield is a base class for adapter fields that provide a single Boolean value.

Class Hierarchy

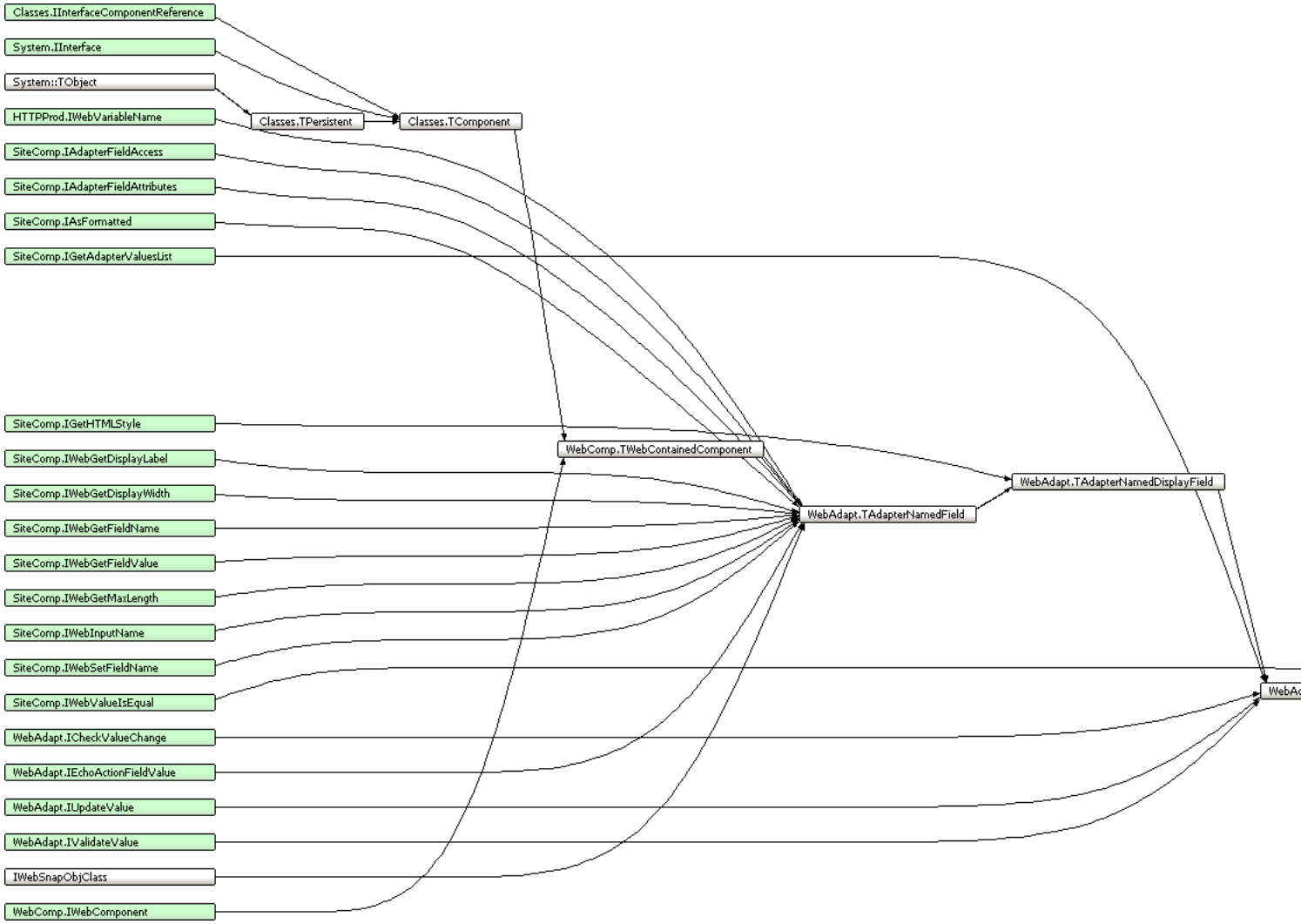

Pascal

```
TCustomAdapterBooleanField = class(TAdapterUpdateField, IWebValueIsEqual);
```

C++

```
class TCustomAdapterBooleanField : public TAdapterUpdateField, public IWebValueIsEqual;
```

File

WebAdapt

Description

tcustomadapterbooleanfield calls event handlers to retrieve the adapter field's value, validate a value entered by the end-user, and update the adapter field's value.

See Also

TAdapterUpdateField (see page 24007)

TAdapterBooleanField (see page 23758)

TAdapter (see page 23713)

173.1.46.1 TCustomAdapterBooleanField Members

The following tables list the members exposed by TCustomAdapterBooleanField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 24015)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 24208)	Retrieves a Boolean value entered by the end-user.
V	ConvertValueCount (see page 24015)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 24016)	Retrieves a value entered by the end-user.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoOnValidateValue (see page 24016)	Validates field values entered by the end-user.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 24016)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.

	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
	GetDisplayStyleType (see page 24208)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 24208)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetValueCount (see page 24017)	Provides the number of values associated with this adapter field.
	GetValues (see page 24017)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 24018)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 24018)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 24018)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 24019)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 24019)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 24209)	Returns the value of this adapter field as a variant.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 24019)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplIsEqual (see page 24209)	Indicates whether the value of this field is equal to a particular Variant value.

V	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
V	ImplSetEchoActionFieldValue (see page 24020)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
V	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
V	ImplSetFieldName (see page 24020)	Sets the name used to access this adapter field with server-side script.
V	ImplUpdateValue (see page 24209)	Called to update the adapter field's value with a value entered by the end-user.
V	ImplValidateValue (see page 24020)	Validates values(s) entered by the end-user.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsEqual (see page 24210)	Indicates whether the value of this field is equal to a particular Variant value.
V	IsMultiValueField (see page 24021)	Indicates whether this adapter field can support multiple values.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 24021)	Clears references to a deleted component.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 24021)	Updates the adapter field with value(s) entered by the end-user.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateValue (see page 24022)	Validates values(s) entered by the end-user.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.

R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 24023)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetValue (see page 24210)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 24211)	Called to update the adapter field's value with a value entered by the end-user.
	OnValidateValue (see page 24024)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
V	Create (see page 24014)	Creates an instance of TAdapterUpdateField (see page 24007).
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterValuesList (see page 19695)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.

	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 23923)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23924)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 23924)	Indicates whether this adapter field is read-only.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	Value (see page 24022)	Retrieves the value associated with this adapter field.
	ValuesList (see page 24023)	Associates a list of values with the adapter field.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.46.2 TCustomAdapterBooleanField Methods

The methods of the TCustomAdapterBooleanField class are listed here.

Protected Methods

	Name	Description
V	ConvertValue (see page 24208)	Retrieves a Boolean value entered by the end-user.
V	GetDisplayStyleType (see page 24208)	Suggests an HTML display type for this adapter field.
V	GetInputStyleType (see page 24208)	Suggests an HTML input type for this adapter field.

	ImplGetValue (see page 24209)	Returns the value of this adapter field as a variant.
	ImplIsEqual (see page 24209)	Indicates whether the value of this field is equal to a particular Variant value.
	ImplUpdateValue (see page 24209)	Called to update the adapter field's value with a value entered by the end-user.
	IsEqual (see page 24210)	Indicates whether the value of this field is equal to a particular Variant value.

173.1.46.2.1 TCustomAdapterBooleanField.ConvertValue Method

Retrieves a Boolean value entered by the end-user.

Pascal

```
function ConvertValue(AValue: IActionFieldValue): Variant; override;
```

C++

```
virtual __fastcall Variant ConvertValue(IActionFieldValue AValue);
```

Description

ConvertValue overrides TAdapterUpdateField (see page 24007)'s ConvertValue method. Convert value attempts to cast the value entered by the end-user to a Boolean. If the cast fails, false is returned.

See Also

ConvertValue (see page 24015)

173.1.46.2.2 TCustomAdapterBooleanField.GetDisplayStyleType Method

Suggests an HTML display type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const AnsiString
AAdapterMode);
```

Description

GetDisplayStyleType overrides TAdapterNamedDisplayField (see page 23913)'s GetDisplayStyleType method. GetDispayStyleType returns htmldText, indicating that this adapter field should be displayed by as an HTML text block (paragraph).

See Also

GetDisplayStyleType (see page 23920)

TAdapterDisplayHTMLElementType

173.1.46.2.3 TCustomAdapterBooleanField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const AnsiString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType method. GetInputStyleType returns htmliCheckBox indicating that a check box should be used to input this adapter field's value.

See Also

GetInputStyleType (see page 23921)

TAdapterInputHTMLElementType

173.1.46.2.4 TCustomAdapterBooleanField.ImplGetValue Method

Returns the value of this adapter field as a variant.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns the value of this adapter field as a variant. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to view this field, an exception is raised. If the EchoActionFieldValue (see page 23944) property is true, then the value entered by end-user is returned. If the EchoActionFieldValue (see page 23944) property is false then the OnGetValue (see page 24210) event handler is called to retrieve the value of the adapter field.

See Also

OnGetValue (see page 24210)

173.1.46.2.5 TCustomAdapterBooleanField.ImplIsEqual Method

Indicates whether the value of this field is equal to a particular Variant value.

Pascal

```
function ImplIsEqual(const AValue: Variant): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplIsEqual(const Variant AValue);
```

Description

ImplIsEqual is called by IsEqual (see page 24210). ImplIsEqual compares the parameter value with the value associated with this adapter field.

See Also

IsEqual (see page 24210)

173.1.46.2.6 TCustomAdapterBooleanField.ImplUpdateValue Method

Called to update the adapter field's value with a value entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The ImplUpdateValue procedure updates the value of this adapter field using the given action request. The AFieldIndex parameter is used to extract the value entered by the user from the action request. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to modify this field, an exception is raised. If the OnUpdateValue (see page 24211) event handler is not nil (Delphi) or NULL (C++), it is called to update the adapter field's value.

See Also

OnUpdateValue (see page 24211)

173.1.46.2.7 TCustomAdapterBooleanField.IsEqual Method

Indicates whether the value of this field is equal to a particular Variant value.

Pascal

```
function IsEqual(const AValue: Variant): Boolean;
```

C++

```
__fastcall Boolean IsEqual(const Variant AValue);
```

Description

IsEqual is called when server-side script calls the IsEqual method of this adapter field. Server-side script may use the IsEqual function to determine whether to check an HTML checkbox that displays this adapter field's value. IsEqual returns the result of ImplIsEqual (see page 24209).

See Also

ImplIsEqual (see page 24209)

173.1.46.3 TCustomAdapterBooleanField Events

The events of the TCustomAdapterBooleanField class are listed here.

Public Events

	Name	Description
	OnGetValue (see page 24210)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 24211)	Called to update the adapter field's value with a value entered by the end-user.

173.1.46.3.1 TCustomAdapterBooleanField.OnGetValue Event

Called to retrieve the adapter field's value.

Pascal

```
property OnGetValue: TAdapterFieldGetBooleanEvent;
```

C++

```
__property TAdapterFieldGetBooleanEvent OnGetValue;
```

Description

OnGetValue is called by ImplGetValue (see page 24209) to retrieve the value associated with this adapter field.

See Also

ImplGetValue (see page 24209)

173.1.46.3.2 TCustomAdapterBooleanField.OnUpdateValue Event

Called to update the adapter field's value with a value entered by the end-user.

Pascal

```
property OnUpdateValue: TAdapterFieldBooleanValueEvent;
```

C++

```
__property TAdapterFieldBooleanValueEvent OnUpdateValue;
```

Description

OnUpdateValue is called when an action request contains a value for this adapter field. If this event handler is not implemented, the value entered by the end-user will be ignored.

See Also

ImplUpdateValue (see page 24209)

173.1.47 TCustomAdapterField Class

TCustomAdapterField is a base class for adapter fields that provide a single value.

Class Hierarchy

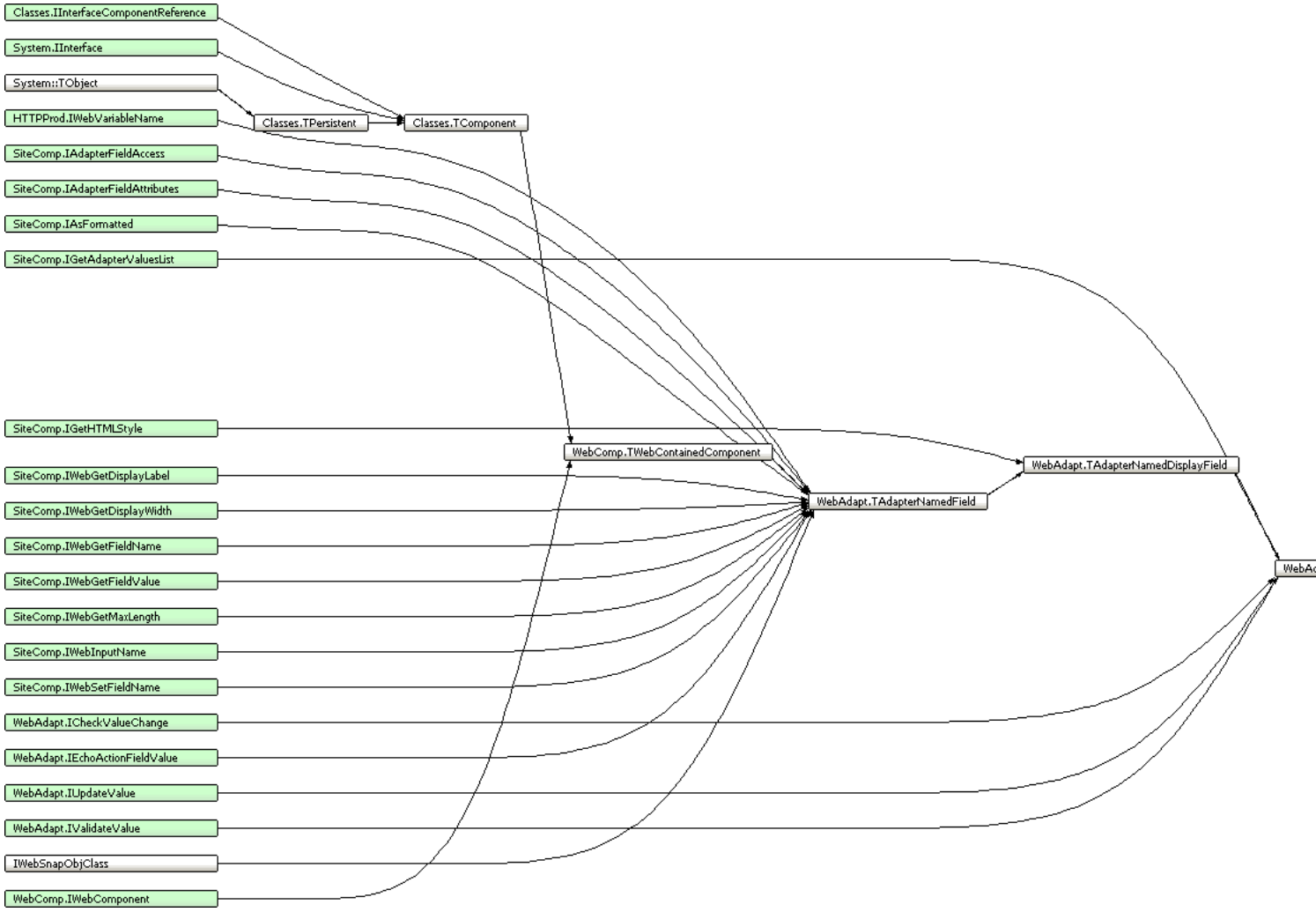

Pascal

```
TCustomAdapterField = class(TAdapterUpdateField);
```

C++

```
class TCustomAdapterField : public TAdapterUpdateField;
```

File

WebAdapt

Description

TCustomAdapterField calls event handlers to retrieve the adapter field's value, validate a value entered by the end-user, and update the adapter field's value.

See Also

TAdapterUpdateField (see page 24007)

TAdapterField (see page 23818)

TAdapter (see page 23713)

173.1.47.1 TCustomAdapterField Members

The following tables list the members exposed by TCustomAdapterField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 24015)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 24015)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 24015)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 24016)	Retrieves a value entered by the end-user.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoOnValidateValue (see page 24218)	Validates a single value entered by the end-user.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 24016)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 23920)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 23921)	Suggests an HTML input type for this adapter field.
V	GetInputStyleType (see page 24017)	Indicates the preferred HTML element to use when editing this adapter field's value.

	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetValueCount (see page 24017)	Provides the number of values associated with this adapter field.
	GetValues (see page 24017)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 24018)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 24018)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 24018)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 24019)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 24019)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 24219)	Returns the value of this adapter field as a variant.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 24019)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 24020)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 24020)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 24219)	Called to update the adapter field's value with a value entered by the end-user.
	ImplValidateValue (see page 24020)	Validates values(s) entered by the end-user.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 24021)	Indicates whether this adapter field can support multiple values.

V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 24021)	Clears references to a deleted component.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 24021)	Updates the adapter field with value(s) entered by the end-user.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateValue (see page 24022)	Validates values(s) entered by the end-user.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 24023)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetValue (see page 24220)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 24220)	Called to update the adapter field's value with a value entered by the end-user.
	OnValidateValue (see page 24024)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.

	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
	Create (see page 24014)	Creates an instance of TAdapterUpdateField (see page 24007).
	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterValuesList (see page 19695)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.

	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.

	DisplayLabel (see page 23923)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23924)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 23924)	Indicates whether this adapter field is read-only.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	Value (see page 24022)	Retrieves the value associated with this adapter field.
	ValuesList (see page 24023)	Associates a list of values with the adapter field.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.47.2 TCustomAdapterField Methods

The methods of the TCustomAdapterField class are listed here.

Protected Methods

	Name	Description
V	DoOnValidateValue (see page 24218)	Validates a single value entered by the end-user.
V	ImplGetValue (see page 24219)	Returns the value of this adapter field as a variant.
V	ImplUpdateValue (see page 24219)	Called to update the adapter field's value with a value entered by the end-user.

173.1.47.2.1 TCustomAdapterField.DoOnValidateValue Method

Validates a single value entered by the end-user.

Pascal

```
procedure DoOnValidateValue(AValue: IActionFieldValue; var AHandled: Boolean); override;
```

C++

```
virtual __fastcall DoOnValidateValue(IActionFieldValue AValue, Boolean AHandled);
```

Description

DoOnValidateValue calls the inherited DoOnValidateValue. If this succeeds by not raising an exception then additional checks are made. One check tests that the request does not contain more than one value for this adapter field. Another check is performed if the Required (see page 23924) property is True. In this case, the request must contain a value for this adapter

field. An exception is raised if any of these checks fail.

See Also

DoOnValidateValue (see page 24016)

Required (see page 23924)

173.1.47.2.2 TCustomAdapterField.ImplGetValue Method

Returns the value of this adapter field as a variant.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue function returns the value of this adapter field as a variant. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to view this field, an exception is raised. If the EchoActionFieldValue (see page 23944) property is True, then the value entered by end-user should is returned. If the EchoActionFieldValue (see page 23944) property is False then the OnGetValue (see page 24220) event handler is called to retrieve the value of the adapter field.

See Also

OnGetValue (see page 24220)

CheckViewAccess (see page 23931)

EchoActionFieldValue (see page 23944)

173.1.47.2.3 TCustomAdapterField.ImplUpdateValue Method

Called to update the adapter field's value with a value entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The ImplUpdateValue procedure updates the value using the given action request. The AfieldIndex parameter is used to extract the value entered by the user from the action request. First, the access rights defined for this adapter field are validated agains the rights of the end-user. If the end-user does not have rights to modify this field, an exception is raised. If the OnUpdateValue (see page 24220) event handler is not nil, it is called to update the adapter field's value.

See Also

OnUpdateValue (see page 24220)

173.1.47.3 TCustomAdapterField Events

The events of the TCustomAdapterField class are listed here.

Public Events

	Name	Description
	OnGetValue (see page 24220)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 24220)	Called to update the adapter field's value with a value entered by the end-user.

173.1.47.3.1 TCustomAdapterField.OnGetValue Event

Called to retrieve the adapter field's value.

Pascal

```
property OnGetValue: TAdapterFieldGetValueEvent;
```

C++

```
__property TAdapterFieldGetValueEvent OnGetValue;
```

Description

OnGetValue is called by ImplGetValue (see page 24219) to retrieve the value associated with this adapter field.

See Also

ImplGetValue (see page 24219)

173.1.47.3.2 TCustomAdapterField.OnUpdateValue Event

Called to update the adapter field's value with a value entered by the end-user.

Pascal

```
property OnUpdateValue: TAdapterFieldValueEvent;
```

C++

```
__property TAdapterFieldValueEvent OnUpdateValue;
```

Description

OnUpdateValue is called when an action request contains a value for this adapter field. If this event handler is not implemented, the value entered by the end-user will be ignored.

173.1.48 TCustomAdapterFileField Class

TCustomAdapterFileField is a base class for adapter fields that can have associated files.

Class Hierarchy

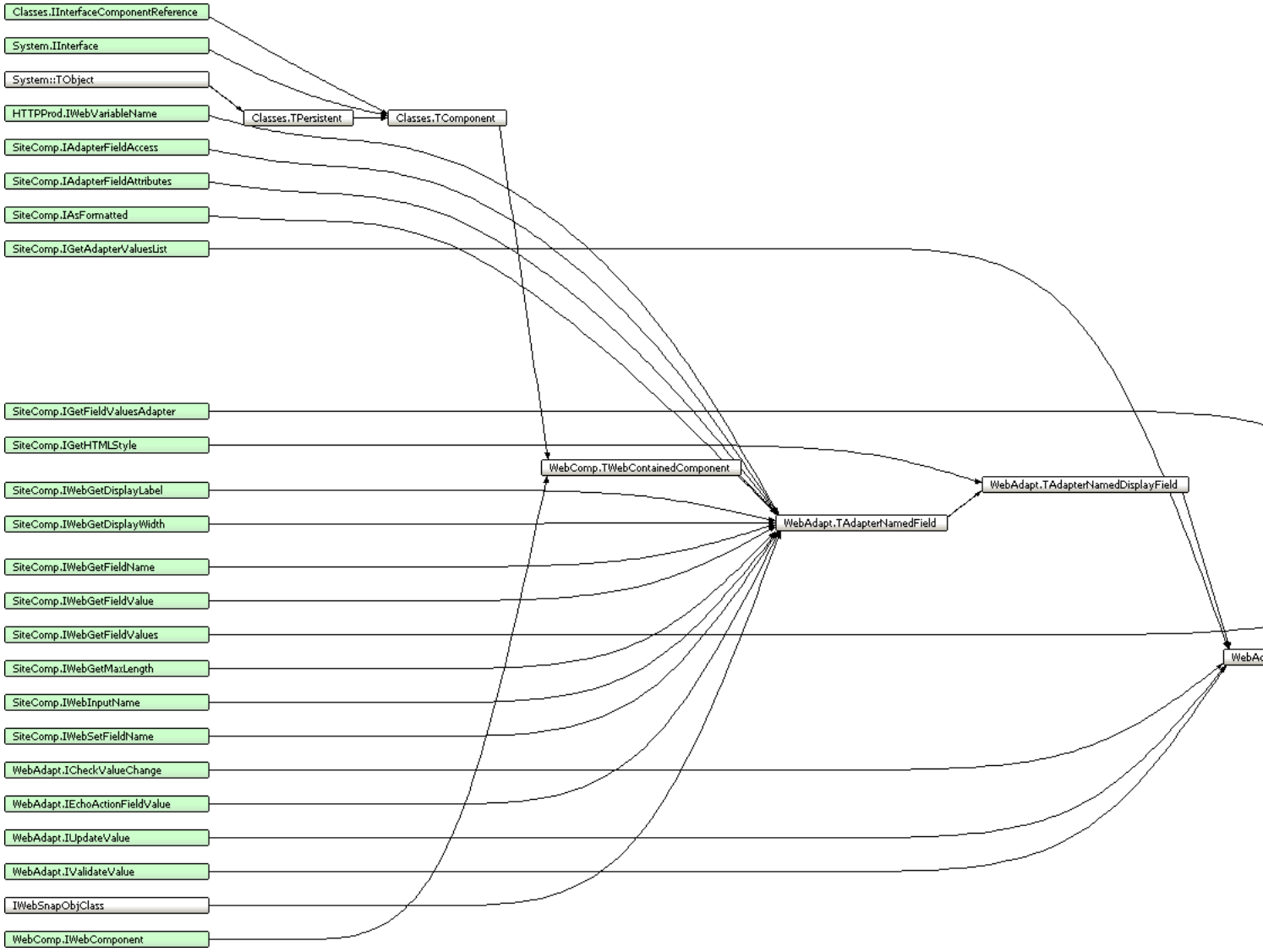

Pascal

```
TCustomAdapterFileField = class(TBaseAdapterMultiValueField);
```

C++

```
class TCustomAdapterFileField : public TBaseAdapterMultiValueField;
```

File

WebAdapt

Description

Derive from TCustomAdapterFileField when the adapter field can have zero or more associated files. Typically, file input HTML elements are used to upload files. Multiple file input HTML elements having the same name can be used to upload multiple files in a single request.

See Also

TBaseAdapterMultiValueField (see page 24086)

TAdapterFileField (see page 23837)

TAdapterImageField (see page 23869)

173.1.48.1 TCustomAdapterFileField Members

The following tables list the members exposed by TCustomAdapterFileField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 24015)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 24015)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 24015)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 24016)	Retrieves a value entered by the end-user.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoOnValidateValue (see page 24228)	Validates files uploaded by the end-user.
	EndValueIterator (see page 24093)	Completes iteration.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 24016)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetCurrentValue (see page 24094)	Implements IWebGetFieldValues' GetCurrentValue.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 24228)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetFieldValuesAdapter (see page 24094)	Returns the adapter object that implements the Values property.

	GetFileNames (see page 24229)	Returns the count of files associated with this adapter field.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 24229)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetValueCount (see page 24229)	Returns the number of values associated with this adapter field.
	GetValues (see page 24230)	Returns one of the values associated with the adapter field.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasValue (see page 24095)	Implements IWebGetFieldValues' HasValue method.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 24018)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 24230)	Indicates whether the end-user may have uploaded new files for this adapter field.
	ImplGetCurrentValue (see page 24095)	Implements IWebGetFieldValues' GetCurrentValue (see page 24094) method.
	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 24018)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 24019)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 24019)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetFieldValuesAdapter (see page 24096)	Returns the adapter object that implements the Values property.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 24019)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasValue (see page 24096)	Implements IWebGetFieldValues' HasValue (see page 24095) method.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.

V	ImplSetEchoActionFieldValue (see page 24020)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
V	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
V	ImplSetFieldName (see page 24020)	Sets the name used to access this adapter field with server-side script.
V	ImplUpdateValue (see page 24230)	Updates the adapter field with value(s) entered by the end-user.
V	ImplValidateValue (see page 24020)	Validates values(s) entered by the end-user.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	IsMultiValueField (see page 24021)	Indicates whether this adapter field can support multiple values.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextValueIteration (see page 24096)	Implements IWebGetFieldValues' NextValueIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 24021)	Clears references to a deleted component.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartValueIterator (see page 24097)	Implements IWebGetFieldValues' StartValueIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 24021)	Updates the adapter field with value(s) entered by the end-user.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateValue (see page 24022)	Validates values(s) entered by the end-user.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.

R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 24023)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetFileCount (see page 24232)	Returns the number of values associated with this adapter field.
	OnGetFileNames (see page 24232)	Returns one of the filenames associated with the adapter field.
	OnUploadFiles (see page 24233)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateFiles (see page 24233)	Validates values entered by the end-user.
	OnValidateValue (see page 24024)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
V	Create (see page 24014)	Creates an instance of TAdapterUpdateField (see page 24007).
V	Create (see page 24092)	Constructor for TBaseAdapterMultiValueField (see page 24086).
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24093)	Destructor for TBaseAdapterMultiValueField (see page 24086).
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndValueIterator (see page 19761)	Indicate the iteration is finished.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterValuesList (see page 19695)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetCurrentValue (see page 19761)	Retrieves the current value as a variant.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetFieldValuesAdapter (see page 19698)	This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasValue (see page 19761)	Indicates whether the adapter field has a matching value.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextValueIteration (see page 19762)	Position the iteration pointer on the next value.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.

	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartValueIterator (see page 19762)	Reset the iteration pointer to the first item.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 23923)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23924)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
R	FileCount (see page 24231)	Returns one of the filenames associated with this adapter field.
R	FileNames (see page 24231)	Returns a file associated with this adapter field.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 23924)	Indicates whether this adapter field is read-only.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	Value (see page 24022)	Retrieves the value associated with this adapter field.
	ValuesList (see page 24023)	Associates a list of values with the adapter field.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.48.2 TCustomAdapterFileField Methods

The methods of the TCustomAdapterFileField class are listed here.

Protected Methods

	Name	Description
	DoOnValidateValue (see page 24228)	Validates files uploaded by the end-user.
	GetDisplayStyleType (see page 24228)	Suggests an HTML display type for this adapter field.
	GetFileNames (see page 24229)	Returns the count of files associated with this adapter field.
	GetInputStyleType (see page 24229)	Suggests an HTML input type for this adapter field.
	GetValueCount (see page 24229)	Returns the number of values associated with this adapter field.
	GetValues (see page 24230)	Returns one of the values associated with the adapter field.
	ImplCheckValueChange (see page 24230)	Indicates whether the end-user may have uploaded new files for this adapter field.
	ImplUpdateValue (see page 24230)	Updates the adapter field with value(s) entered by the end-user.

173.1.48.2.1 TCustomAdapterFileField.DoOnValidateValue Method

Validates files uploaded by the end-user.

Pascal

```
procedure DoOnValidateValue(AValue: IActionFieldValue; var AHandled: Boolean); override;
```

C++

```
virtual __fastcall DoOnValidateValue(IActionFieldValue AValue, Boolean AHandled);
```

Description

DoOnValidateValue overrides TAdapterUpdateField (see page 24007)'s DoOnValidateValue method. If the OnValidateFiles (see page 24233) event handler is not nil (Delphi) or NULL (C++), DoOnValidateValue calls it to validate uploaded files. If the Required (see page 23924) property is true, DoOnValidateValues also verifies that 1 of more values have been entered by the end-user. An exception will be raised to indicate a validation error.

See Also

DoOnValidateValue (see page 24016)

OnValidateValue (see page 24024)

Required (see page 23924)

OnValidateFiles (see page 24233)

173.1.48.2.2 TCustomAdapterFileField.GetDisplayStyleType Method

Suggests an HTML display type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const AnsiString
AAdapterMode);
```

Description

GetDisplayStyleType overrides TAdapterNamedDisplayField (see page 23913)'s GetDisplayStyleType.

GetDispayStyleTypereturns htmldList, indicating that this adapter field should be displayed by as a list of read-only values.

See Also

GetDisplayStyle (see page 23920)

TAdapterDisplayHTMLElementType

173.1.48.2.3 TCustomAdapterFileField.GetFileNames Method

Returns the count of files associated with this adapter field.

Pascal

```
function GetFileNames(I: Integer): string;
```

C++

```
__fastcall AnsiString GetFileNames(int I);
```

Description

GetFileNames implements the FileNames (see page 24231) property. If the EchoActionFieldValue (see page 23944) flag is true then the filename is extracted from the HTTP request. Otherwise, the OnGetFileNames (see page 24232) event is called. GetFileNames returns and empty string if a value can't be found because OnGetFileNames (see page 24232) is nil (Delphi) or NULL (C++) or the HTTP request does not contain a filename for this adapter field.

See Also

FileNames (see page 24231)

OnGetFileNames (see page 24232)

173.1.48.2.4 TCustomAdapterFileField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const AnsiString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType method. GetInputStyleType returns htmliFile indicating that file upload HTML elements should be used to input values for this adapter field.

See Also

GetInputStyleType (see page 23921)

TAdapterInputHTMLElementType

173.1.48.2.5 TCustomAdapterFileField.GetValueCount Method

Returns the number of values associated with this adapter field.

Pascal

```
function GetValueCount: Integer; override;
```

C++

```
virtual __fastcall int GetValueCount();
```

Description

GetValueCount implements the FileCount (see page 24231) property. If the EchoActionFieldValue (see page 23944) flag is true then the result is the number of values in the HTTP request for this adapter field. Otherwise, the OnGetFileCount (see page 24232) event is called. GetValueCount returns zero if a value can't be found because OnGetFileCount (see page 24232) is nil (Delphi) or NULL (C++) or the HTTP request does not contain a value for this adapter field.

See Also

FileCount (see page 24231)

OnGetFileCount (see page 24232)

173.1.48.2.6 TCustomAdapterFileField.GetValues Method

Returns one of the values associated with the adapter field.

Pascal

```
function GetValues(I: Integer): Variant; override;
```

C++

```
virtual __fastcall Variant GetValues(int I);
```

Description

GetValues returns the result of GetFileNames (see page 24229).

See Also

GetFileNames (see page 24229)

173.1.48.2.7 TCustomAdapterFileField.ImplCheckValueChange Method

Indicates whether the end-user may have uploaded new files for this adapter field.

Pascal

```
function ImplCheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
override;
```

C++

```
virtual __fastcall Boolean ImplCheckValueChange(IActionRequest AActionRequest, int
AFieldIndex);
```

Description

ImplCheckValueChange overrides TAdapterUpdateField (see page 24007)'s ImplCheckValueChange method. ImplCheckValueChange returns true if the end-user has uploaded any files.

See Also

IActionRequest (see page 935)

ImplCheckValueChange (see page 24018)

173.1.48.2.8 TCustomAdapterFileField.ImplUpdateValue Method

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

ImplUpdateValue overrides TAdapterUpdateField (see page 24007)'s ImplUpdateValue method. After checking that the end-user has rights to modify this adapter field, the OnUploadFiles (see page 24233) event is called.

See Also

IActionRequest (see page 935)

OnUploadFiles (see page 24233)

ImplUpdateValue (see page 24020)

173.1.48.3 TCustomAdapterFileField Properties

The properties of the TCustomAdapterFileField class are listed here.

Public Properties

	Name	Description
R	FileCount (see page 24231)	Returns one of the filenames associated with this adapter field.
R	FileNames (see page 24231)	Returns a file associated with this adapter field.

173.1.48.3.1 TCustomAdapterFileField.FileCount Property

Returns one of the filenames associated with this adapter field.

Pascal

```
property FileCount: Integer;
```

C++

```
__property int FileCount;
```

Description

```
__property int FileCount = {read=GetValueCount, nodefault};
```

FileCount can be used with the FileNames (see page 24231) property to access all the filenames associated with this adapter field. FileCount is implemented by GetValueCount (see page 24229).

See Also

FileNames (see page 24231)

GetValueCount (see page 24229)

173.1.48.3.2 TCustomAdapterFileField.FileNames Property

Returns a file associated with this adapter field.

Pascal

```
property FileNames [I: Integer]: string;
```

C++

```
__property AnsiString FileNames[int I];
```

Description

FileNames can be used with the FileCount (see page 24231) property to access all the filenames associated with this adapter field. FileNames is implemented by GetFileNames (see page 24229).

See Also

FileCount (see page 24231)

GetFileNames (see page 24229)

173.1.48.4 TCustomAdapterFileField Events

The events of the TCustomAdapterFileField class are listed here.

Public Events

	Name	Description
	OnGetFileCount (see page 24232)	Returns the number of values associated with this adapter field.
	OnGetFileNames (see page 24232)	Returns one of the filenames associated with the adapter field.
	OnUploadFiles (see page 24233)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateFiles (see page 24233)	Validates values entered by the end-user.

173.1.48.4.1 TCustomAdapterFileField.OnGetFileCount Event

Returns the number of values associated with this adapter field.

Pascal

```
property OnGetFileCount: TAdapterFieldGetValueCountEvent;
```

C++

```
__property TAdapterFieldGetValueCountEvent OnGetFileCount;
```

Description

Write an OnGetFileCount event handler to indicate the number of filenames associated with this adapter field.

See Also

GetValueCount (see page 24229)

173.1.48.4.2 TCustomAdapterFileField.OnGetFileNames Event

Returns one of the filenames associated with the adapter field.

Pascal

```
property OnGetFileNames: TAdapterFieldGetMultiValueStringEvent;
```

C++

```
__property TAdapterFieldGetMultiValueStringEvent OnGetFileNames;
```

Description

Write an OnGetFileNames event handler to provide the names of files associated with this adapter field.

See Also

GetFileNames (see page 24229)

173.1.48.4.3 TCustomAdapterFileField.OnUploadFiles Event

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
property OnUploadFiles: TAdapterFieldFileListEvent;
```

C++

```
__property TAdapterFieldFileListEvent OnUploadFiles;
```

Description

OnUploadFiles is called by ImplUpdateValue (see page 24230). Write an OnUploadFiles event handler to accept files uploaded by the end-user. Raise an exception to indicate an error.

See Also

ImplUpdateValue (see page 24230)

TUpdateFileList (see page 24550)

173.1.48.4.4 TCustomAdapterFileField.OnValidateFiles Event

Validates values entered by the end-user.

Pascal

```
property OnValidateFiles: TAdapterFieldFileListEvent;
```

C++

```
__property TAdapterFieldFileListEvent OnValidateFiles;
```

Description

OnValidateFiles is called by DoOnValidateValue (see page 24228). Implement this event handler to validate files uploaded by the end-user. Raise an exception to indicate a validation error.

See Also

DoOnValidateValue (see page 24228)

TUpdateFileList (see page 24550)

173.1.49 TCustomAdapterGotoPageAction Class

TCustomAdapterGotoPageAction is an adapter action that supports commands to go to a page of a TDefaultFieldsPagedAdapter (see page 24403).

Class Hierarchy

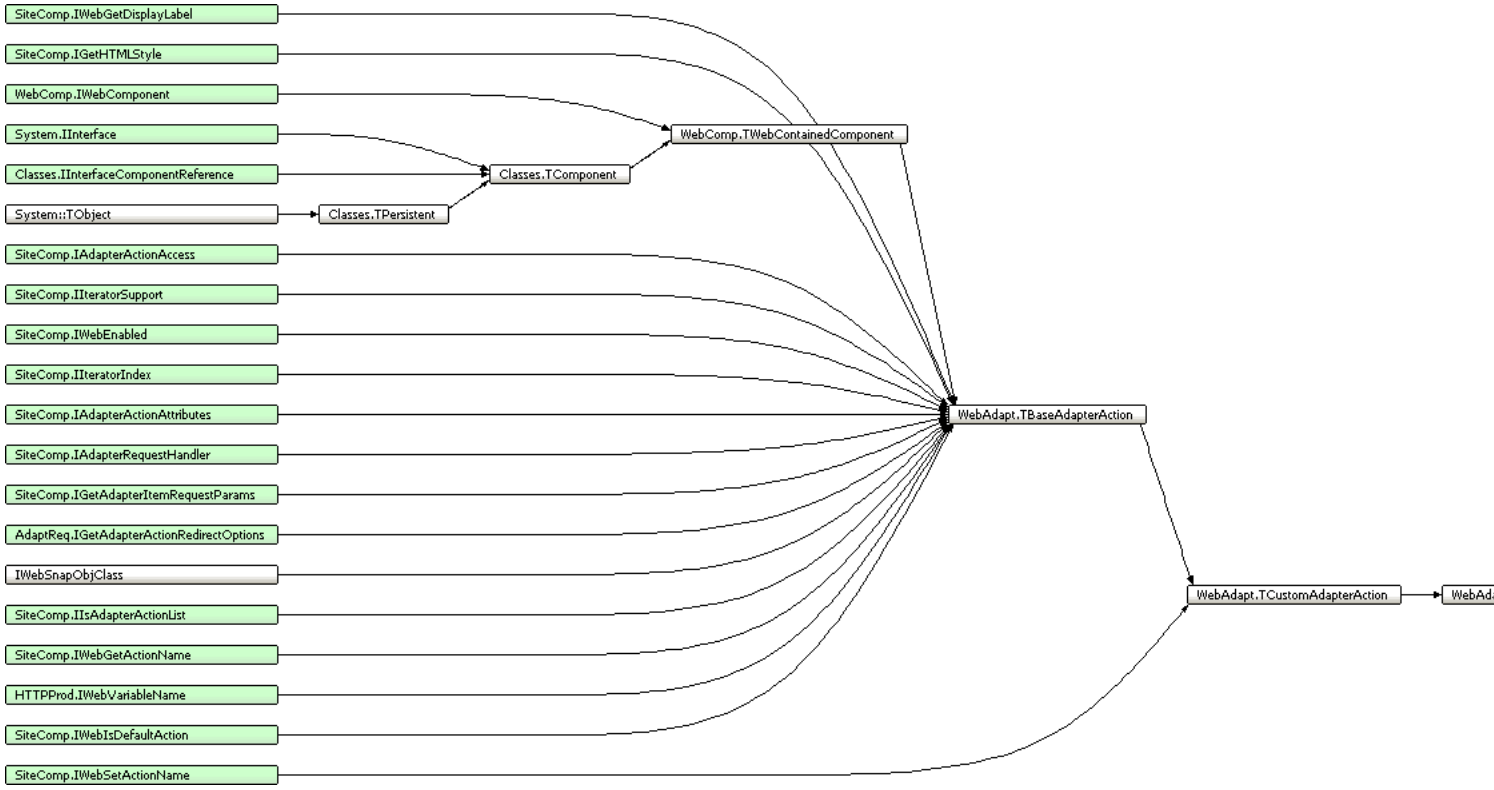

Pascal

```
TCustomAdapterGotoPageAction = class(TCustomPagedAdapterAction);
```

C++

```
class TCustomAdapterGotoPageAction : public TCustomPagedAdapterAction;
```

File

WebAdapt

Description

TCustomAdapterGotoPageAction is an adapter action that has a list of commands. Each command has a parameter that identifies a page in the parent adapter. When executed, TCustomAdapterGotoPageAction examines the parameter and changes the current page in the parent adapter.

See Also

TCustomPagedAdapterAction (see page 24371)

TAdapterGotoPageAction (see page 23849)

173.1.49.1 TCustomAdapterGotoPageAction Members

The following tables list the members exposed by TCustomAdapterGotoPageAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 24377)	Returns the parent adapter of this adapter action.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 24240)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24241)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 24241)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.

	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 24241)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 24242)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24242)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24243)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 24243)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 24243)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Adapter (see page 24378)	Returns the parent adapter of this adapter action.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 24075)	Called after the adapter action has executed.
	OnAfterExecute (see page 24198)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 24075)	Called after the adapter action response is generated.
	OnAfterGetResponse (see page 24199)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24076)	Called before the adapter action is executed.
	OnBeforeExecute (see page 24199)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24076)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 24200)	Called before the adapter action response is generated.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetPageName (see page 24244)	Provides the display labels for each goto page command.
	OnGetParams (see page 24077)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.

	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 24073)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 24073)	Defines the rights required to execute this action.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.

	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RedirectOptions (see page 24074)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.49.2 TCustomAdapterGotoPageAction Methods

The methods of the TCustomAdapterGotoPageAction class are listed here.

Protected Methods

	Name	Description
V	GetDefaultActionName (see page 24240)	Gets the name used to reference this adapter action with server-side script.
V	GetDisplayStyleType (see page 24241)	Suggests an HTML display type for this adapter action.
V	GetItemCount (see page 24241)	Returns the number of commands defined by this adapter action.
V	ImplExecuteActionRequest (see page 24241)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetAdapterItemRequestParams (see page 24242)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 24242)	Returns the suggested display label for this adapter action.
V	ImplGetIsAdapterActionList (see page 24243)	Indicates whether this adapter action may have multiple commands.
V	ImplIsDefaultAction (see page 24243)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	ImplWebEnabled (see page 24243)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

173.1.49.2.1 TCustomAdapterGotoPageAction.GetDefaultActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultActionName();
```

Description

GetDefaultActionName returns 'GotoPage'.

See Also

GetActionName (see page 24054)

GetDefaultActionName (see page 24055)

173.1.49.2.2 TCustomAdapterGotoPageAction.GetDisplayStyleType Method

Suggests an HTML display type for this adapter action.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterActionHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterActionHTMLElementType GetDisplayStyleType(const AnsiString
AAdapterMode);
```

Description

GetDisplayStyleType returns htmlaAnchor, indicating that goto page commands should be displayed using a hyperlink.

See Also

GetDisplayStyle (see page 24056)

173.1.49.2.3 TCustomAdapterGotoPageAction.GetItemCount Method

Returns the number of commands defined by this adapter action.

Pascal

```
function GetItemCount: Integer; override;
```

C++

```
virtual __fastcall int GetItemCount();
```

Description

TCustomAdapterGotoPageAction (see page 24233) returns the value of the parent adapter's PageCount property.

See Also

ItemCount (see page 24074)

ItemIndex (see page 24074)

GetIsAdapterActionList (see page 24058)

PageCount (see page 24418)

173.1.49.2.4 TCustomAdapterGotoPageAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest sets the parent adapter's current page using the page index passed in the AActionRequest

parameter.

See Also

ImplExecuteActionRequest (see page 24063)

173.1.49.2.5 TCustomAdapterGotoPageAction.ImplGetAdapterItemRequestParams Method

Provides the name/value pairs needed to execute this adapter action.

Pascal

```
procedure ImplGetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams); override;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(AnsiString AIdentifier,
IAdapterItemRequestParams Params);
```

Description

ImplGetAdapterItemRequestParams is called by GetAdapterItemRequestParams (see page 24055). ImplGetAdapterItemRequestParams adds the value of the iteration index as a parameter. Then the inherited ImplGetAdapterItemRequestParams method is called so that any event handlers that may be defined will be called.

See Also

GetAdapterItemRequestParams (see page 24055)

OnGetActionParams (see page 24187)

OnGetParams (see page 24077)

ItemIndex (see page 24074)

173.1.49.2.6 TCustomAdapterGotoPageAction.ImplGetDisplayLabel Method

Returns the suggested display label for this adapter action.

Pascal

```
function ImplGetDisplayLabel: string; override;
```

C++

```
virtual __fastcall AnsiString ImplGetDisplayLabel();
```

Description

ImplGetDisplayLabel returns the display label suggested for the goto page command that is identified by the iteration index. If the OnGetPageName (see page 24244) event handler is not nil (Delphi) or NULL (C++), it is called. Otherwise, the ItemIndex (see page 24074) property is converted to a string and returned.

See Also

DisplayLabel (see page 24073)

ItemIndex (see page 24074)

OnGetPageName (see page 24244)

173.1.49.2.7 TCustomAdapterGotoPageAction.ImplGetIsAdapterActionList Method

Indicates whether this adapter action may have multiple commands.

Pascal

```
function ImplGetIsAdapterActionList: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplGetIsAdapterActionList();
```

Description

ImplGetIsAdapterActionList is called by GetIsAdapterActionList (see page 24058). ImplGetIsAdapterActionList returns true if the parent adapters PageSize property is greater than zero.

See Also

GetIsAdapterActionList (see page 24058)

PageSize (see page 24419)

173.1.49.2.8 TCustomAdapterGotoPageAction.ImplIsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function ImplIsDefaultAction(ADisplay: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplIsDefaultAction(TObject * ADisplay);
```

Description

ImplIsDefaultAction is called by IsDefaultAction (see page 24070). ImplIsDefaultAction returns true if the display characteristics identified by ADisplay indicate that the goto page commands are appropriate.

See Also

IsDefaultAction (see page 24070)

173.1.49.2.9 TCustomAdapterGotoPageAction.ImplWebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

ImplWebEnabled returns the result of false if the iteration index is equal to the parent adapter's current page. In other words, the command that goes to the parent adapter's current page will be disabled. Commands to goto all other pages will be enabled.

See Also

WebEnabled (see page 24071)

CurrentPage (see page 24418)

ItemIndex (see page 24074)

173.1.49.3 TCustomAdapterGotoPageAction Events

The events of the TCustomAdapterGotoPageAction class are listed here.

Public Events

	Name	Description
	OnGetPageName (see page 24244)	Provides the display labels for each goto page command.

173.1.49.3.1 TCustomAdapterGotoPageAction.OnGetPageName Event

Provides the display labels for each goto page command.

Pascal

```
property OnGetPageName: TWebFormGetPageNameEvent;
```

C++

```
__property TWebFormGetPageNameEvent OnGetPageName;
```

Description

Implement this event handler to specify display labels for the commands that goto pages in the parent adapter. APageIndex is the zero-based index of the page to goto.

See Also

ImplGetDisplayLabel (see page 24242)

173.1.50 TCustomAdapterImageField Class

TCustomAdapterImageField is a base class for adapter fields that have an associated image file.

Class Hierarchy

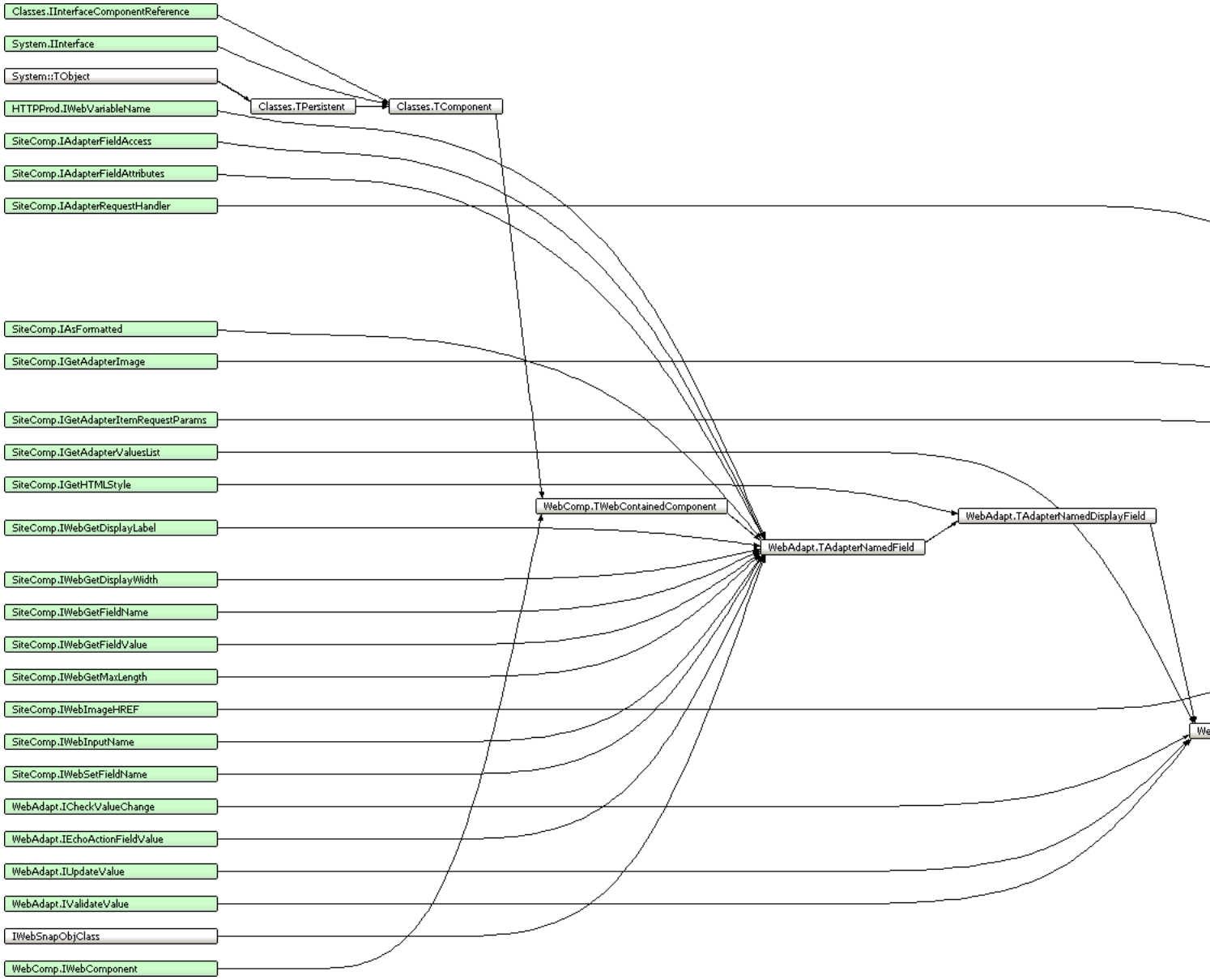

Pascal

```
TCustomAdapterImageField = class(TAdapterUpdateField, IGetAdapterImage,
IGetAdapterItemRequestParams, IAdapterRequestHandler, IWebImageHREF);
```

C++

```
class TCustomAdapterImageField : public TAdapterUpdateField, public IGetAdapterImage, public
IGetAdapterItemRequestParams, public IAdapterRequestHandler, public IWebImageHREF;
```

File

WebAdapt

Description

Derived from TCustomAdapterImageField when the adapter field can an associated image file. Typically, a file input HTML element is used to upload an image file. HTML IMG elements are used to display the image.

See Also

TAdapterImageField (see page 23869)

TAdapterFileField (see page 23837)

173.1.50.1 TCustomAdapterImageField Members

The following tables list the members exposed by TCustomAdapterImageField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 24015)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 24015)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 24015)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 24016)	Retrieves a value entered by the end-user.
	CreateRequestContext (see page 24252)	Creates AdapterRequest and AdapterResponse objects.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoOnValidateValue (see page 24253)	Validates files uploaded by the end-user.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterImage (see page 24253)	Provides an object that can be used to access the image properties of this adapter field.
	GetAdapterItemRequestParams (see page 24253)	Provides a unique identifier and a set of name/value pairs that will be used to create a URL and also be used later when a request is received to generate the adapter field's image file.
	GetAdapterValuesList (see page 24016)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.

	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 24254)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 24254)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
V	GetValueCount (see page 24017)	Provides the number of values associated with this adapter field.
V	GetValues (see page 24017)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HandleRequest (see page 24255)	Handles a request for the image file associated with this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
V	ImplAsFormatted (see page 24018)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
V	ImplCheckValueChange (see page 24255)	Indicates whether the end-user may have uploaded a new image file for this adapter field.
	ImplCreateRequestContext (see page 24255)	Creates AdapterRequest and AdapterResponse objects.
V	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
V	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayLabel (see page 24018)	Gets the preferred display label for this adapter field.
V	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
V	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
V	ImplGetDisplayWidth (see page 24019)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
V	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
V	ImplGetEchoActionFieldValue (see page 24019)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
V	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
V	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 24256)	Returns the name of the image file associated with this adapter field.
V	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVariableName (see page 24019)	Gets the name used to access this adapter field with server-side script.
V	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHandleRequest (see page 24256)	Handles a request for the image file associated with this adapter field.

	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 24020)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 24020)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 24257)	Updates the adapter field with value entered by the end-user.
	ImplValidateValue (see page 24020)	Validates values(s) entered by the end-user.
	ImplWebImageHREF (see page 24257)	Returns a URL that can be used to access the image file associated with this adapter field.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 24021)	Indicates whether this adapter field can support multiple values.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24021)	Clears references to a deleted component.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RenderAdapterImage (see page 24258)	Responds to a request for an image file by providing a stream containing the image.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 24021)	Updates the adapter field with value(s) entered by the end-user.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateValue (see page 24022)	Validates values(s) entered by the end-user.
	WebImageHREF (see page 24258)	Returns a URL that can be used to access the image file associated with this adapter field.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 24023)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetHREF (see page 24260)	Provides the URL of an external image file associated with this adapter field.
	OnGetImage (see page 24260)	Retrieves a stream containing the image file associated with this adapter field.
	OnGetImageName (see page 24260)	Returns the name of the image file associated with this adapter field.
	OnGetParams (see page 24261)	Provides name/value pairs that identify this adapter field's image.
	OnUploadFile (see page 24261)	Updates this adapter field with an image file uploaded by the end-user.
	OnValidateFile (see page 24261)	Used to validate files uploaded by the end user.
	OnValidateValue (see page 24024)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
V	Create (see page 24014)	Creates an instance of TAdapterUpdateField (see page 24007).
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterValuesList (see page 19695)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.

	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 23923)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23924)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	HREF (see page 24259)	Defines a URL that references the image associated with the adapter field.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 24259)	Indicates whether this adapter field is read-only.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	Value (see page 24022)	Retrieves the value associated with this adapter field.
	ValuesList (see page 24023)	Associates a list of values with the adapter field.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.50.2 TCustomAdapterImageField Methods

The methods of the TCustomAdapterImageField class are listed here.

Protected Methods

	Name	Description
	CreateRequestContext (see page 24252)	Creates AdapterRequest and AdapterResponse objects.
V	DoOnValidateValue (see page 24253)	Validates files uploaded by the end-user.
	GetAdapterImage (see page 24253)	Provides an object that can be used to access the image properties of this adapter field.
	GetAdapterItemRequestParams (see page 24253)	Provides a unique identifier and a set of name/value pairs that will be used to create a URL and also be used later when a request is received to generate the adapter field's image file.
V	GetDisplayStyleType (see page 24254)	Suggests an HTML display type for this adapter field.
V	GetInputStyleType (see page 24254)	Suggests an HTML input type for this adapter field.
	HandleRequest (see page 24255)	Handles a request for the image file associated with this adapter field.
V	ImplCheckValueChange (see page 24255)	Indicates whether the end-user may have uploaded a new image file for this adapter field.
	ImplCreateRequestContext (see page 24255)	Creates AdapterRequest and AdapterResponse objects.
V	ImplGetValue (see page 24256)	Returns the name of the image file associated with this adapter field.
V	ImplHandleRequest (see page 24256)	Handles a request for the image file associated with this adapter field.
V	ImplUpdateValue (see page 24257)	Updates the adapter field with value entered by the end-user.
V	ImplWebImageHREF (see page 24257)	Returns a URL that can be used to access the image file associated with this adapter field.
V	RenderAdapterImage (see page 24258)	Responds to a request for an image file by providing a stream containing the image.
	WebImageHREF (see page 24258)	Returns a URL that can be used to access the image file associated with this adapter field.

173.1.50.2.1 TCustomAdapterImageField.CreateRequestContext Method

Creates AdapterRequest and AdapterResponse objects.

Pascal

```
procedure CreateRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall CreateRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

CreateRequestContext implements IAdapterRequestHandler's CreateRequestContext method. CreateRequestContext calls ImplCreateRequestContext (see page 24255).

See Also

IAdapterDispatchParams (see page 19655)

IAdapterRequestHandler (see page 19680)

AdapterRequest (see page 24655)

AdapterResponse (see page 24656)

ImplCreateRequestContext (see page 24255)

173.1.50.2.2 TCustomAdapterImageField.DoOnValidateValue Method

Validates files uploaded by the end-user.

Pascal

```
procedure DoOnValidateValue(AValue: IActionFieldValue; var AHandled: Boolean); override;
```

C++

```
virtual __fastcall DoOnValidateValue(IActionFieldValue AValue, Boolean AHandled);
```

Description

DoOnValidateValue overrides TAdapterUpdateField (see page 24007)'s DoOnValidateValue method. If the OnValidateFile (see page 24261) event handler is not nil (Delphi) or NULL (C++), DoOnValidateValue calls it to validate an uploaded file. DoOnValidateValues also validates that not more than one image file was uploaded and, if the required property is true, that a single file was uploaded. An exception is raised to indicate a validation error.

See Also

DoOnValidateValue (see page 24016)

OnValidateValue (see page 24024)

Required (see page 23924)

OnValidateFile (see page 24261)

173.1.50.2.3 TCustomAdapterImageField.GetAdapterImage Method

Provides an object that can be used to access the image properties of this adapter field.

Pascal

```
function GetAdapterImage: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterImage();
```

Description

GetAdapterImage returns Self. GetAdapterImage will be called when server-side script references the Image property of this adapter field.

173.1.50.2.4 TCustomAdapterImageField.GetAdapterItemRequestParams Method

Provides a unique identifier and a set of name/value pairs that will be used to create a URL and also be used later when a request is received to generate the adapter field's image file.

Pascal

```
procedure GetAdapterItemRequestParams(var AIdentifier: string; AParams:
IAdapterItemRequestParams);
```

C++

```
__fastcall GetAdapterItemRequestParams(AnsiString AIdentifier, IAdapterItemRequestParams
AParams);
```

Description

WebImageHREF (see page 24258) and GetAdapterItemRequestParams are used to identify the URL that is used to retrieve the image file associated with this adapter field. These functions are called when server-side script uses the Image.AsHREF property. If WebImageHREF (see page 24258) returns False then GetAdapterItemRequestParams is called. WebImageHREF (see page 24258) is responsible for returning a reference to an external object. GetAdapterItemRequestParams is used to generate a URL that will result in a request to the adapter field to dynamically generate the image file. GetAdapterItemRequestParams calls OnGetParams (see page 24261) to get the name/value pairs.

See Also

IAdapterItemRequestParams (see page 19673)

RenderAdapterImage (see page 24258)

OnGetParams (see page 24261)

WebImageHREF (see page 24258)

TAdapterDispatcher (see page 24840)

173.1.50.2.5 TCustomAdapterImageField.GetDisplayStyleType Method

Suggests an HTML display type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const AnsiString
AAdapterMode);
```

Description

GetDisplayStyleType overrides TAdapterNamedDisplayField (see page 23913)'s GetDisplayStyleType method. GetDispayStyleType returns htmldImage, indicating that this adapter field should be displayed by an HTML image element.

See Also

GetDisplayStyleType (see page 23920)

173.1.50.2.6 TCustomAdapterImageField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const AnsiString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType method. GetInputStyleType returns htmliFile indicating that a file upload HTML element should be used to input a value for this adapter field.

See Also

GetInputStyleType (see page 23921)

173.1.50.2.7 TCustomAdapterImageField.HandleRequest Method

Handles a request for the image file associated with this adapter field.

Pascal

```
function HandleRequest(DispatchParams: IAdapterDispatchParams): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(IAdapterDispatchParams DispatchParams);
```

Description

HandleRequest implements IAdapterRequestHandler's HandleRequest method. HandleRequest calls ImplHandleRequest (see page 24256).

See Also

IAdapterDispatchParams (see page 19655)

ImplHandleRequest (see page 24256)

IAdapterRequestHandler (see page 19680)

173.1.50.2.8 TCustomAdapterImageField.ImplCheckValueChange Method

Indicates whether the end-user may have uploaded a new image file for this adapter field.

Pascal

```
function ImplCheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
override;
```

C++

```
virtual __fastcall Boolean ImplCheckValueChange(IActionRequest AActionRequest, int
AFieldIndex);
```

Description

ImplCheckValueChange overrides TAdapterUpdateField (see page 24007)'s ImplCheckValueChange method. ImplCheckValueChange returns true if the end-user has uploaded an image file.

See Also

IActionRequest (see page 935)

ImplCheckValueChange (see page 24018)

173.1.50.2.9 TCustomAdapterImageField.ImplCreateRequestContext Method

Creates AdapterRequest and AdapterResponse objects.

Pascal

```
procedure ImplCreateRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall ImplCreateRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

ImplCreateRequestContext creates a new request and response objects based on the specified parameters and stores it with the global Web context. The adapter dispatcher calls this function before calling HandleRequest (see page 24255). The

WebSnap can access the resulting request and response context by reading the AdapterRequest and AdapterResponse properties of the Web context, which is available through the global WebContext function.

DispatchParams is an interface that contains the parameters that were included in the incoming request. These parameters can then be used to identify an image for this field's value.

See Also

CreateRequestContext (see page 24252)

AdapterRequest (see page 24655)

AdapterResponse (see page 24656)

IAdapterDispatchParams (see page 19655)

WebContext (see page 24687)

173.1.50.2.10 TCustomAdapterImageField.ImplGetValue Method

Returns the name of the image file associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns the name of the image file associated with this adapter field

First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to view this field, an exception is raised. If the EchoActionFieldValue (see page 23944) property is true, then the file name entered by end-user is returned. If the EchoActionFieldValue (see page 23944) property is false then the OnGetImageName (see page 24260) event handler is called to retrieve the name of the image file. The value returned by ImplGetValue is used when server-side script references the Value (see page 24022) or DisplayText properties.

See Also

OnGetImageName (see page 24260)

EchoActionFieldValue (see page 23944)

173.1.50.2.11 TCustomAdapterImageField.ImplHandleRequest Method

Handles a request for the image file associated with this adapter field.

Pascal

```
function ImplHandleRequest(DispatchParams: IAdapterDispatchParams): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest(IAdapterDispatchParams DispatchParams);
```

Description

ImplHandleRequest is called by HandleRequest (see page 24255). ImplHandleRequest extracts IImageRequest and IImageResponse from the AdapterRequest and AdapterResponse. ImplHandleRequest returns the result of RenderAdapterImage (see page 24258).

See Also

IAdapterDispatchParams (see page 19655)

ImplHandleRequest

AdapterRequest (see page 24655)

AdapterResponse (see page 24656)

RenderAdapterImage (see page 24258)

IImageRequest (see page 965)

IImageRequest (see page 965)

173.1.50.2.12 TCustomAdapterImageField.ImplUpdateValue Method

Updates the adapter field with value entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The ImplUpdateValue procedure updates the value of this adapter field using the given action request. The AFieldIndex parameter is used to extract the value entered by the user from the action request. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to modify this field, an exception is raised. If action request single file for this adapter field, the OnUploadFile (see page 24261) event handler is called.

See Also

OnUploadFile (see page 24261)

ImplUpdateValue (see page 24020)

173.1.50.2.13 TCustomAdapterImageField.ImplWebImageHREF Method

Returns a URL that can be used to access the image file associated with this adapter field.

Pascal

```
function ImplWebImageHREF(var AHREF: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplWebImageHREF(AnsiString AHREF);
```

Description

ImplWebImageHREF is called by WebImageHREF (see page 24258). If the OnGetHREF (see page 24260) event handler is not nil (Delphi) or NULL (C++), ImplWebImageHREF will call it passing the value of the HREF (see page 24259) property.

See Also

WebImageHREF (see page 24258)

HREF (see page 24259)

OnGetHREF (see page 24260)

173.1.50.2.14 TCustomAdapterImageField.RenderAdapterImage Method

Responds to a request for an image file by providing a stream containing the image.

Pascal

```
procedure RenderAdapterImage(ARequest: IImageRequest; AResponse: IImageResponse); virtual;
```

C++

```
virtual __fastcall RenderAdapterImage(IImageRequest ARequest, IImageResponse AResponse);
```

Description

RenderAdapterImage is called when the TAdapterDispatcher detects a request for an image from this adapter field. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to view this field, an exception is raised. If the OnGetImage (see page 24260) event handler is not nil (Delphi) or NULL (C++), it is called to retrieve the image file as a stream.

See Also

OnGetImage (see page 24260)

TAdapterDispatcher (see page 24840)

ImplHandleRequest (see page 24256)

173.1.50.2.15 TCustomAdapterImageField.WebImageHREF Method

Returns a URL that can be used to access the image file associated with this adapter field.

Pascal

```
function WebImageHREF(var AHREF: string): Boolean;
```

C++

```
__fastcall Boolean WebImageHREF(AnsiString AHREF);
```

Description

WebImageHREF and GetAdapterItemRequestParams (see page 24253) are used to identify the URL that is used to retrieve an image file associated with this adapter field. These functions are called when server-side script uses the Image.AsHREF property. WebImageHREF returns the result of ImplWebImageHREF (see page 24257).

See Also

ImplWebImageHREF (see page 24257)

GetAdapterItemRequestParams (see page 24253)

173.1.50.3 TCustomAdapterImageField Properties

The properties of the TCustomAdapterImageField class are listed here.

Public Properties

	Name	Description
	HREF (see page 24259)	Defines a URL that references the image associated with the adapter field.
	ReadOnly (see page 24259)	Indicates whether this adapter field is read-only.

173.1.50.3.1 TCustomAdapterImageField.HREF Property

Defines a URL that references the image associated with the adapter field.

Pascal

```
property HREF: string;
```

C++

```
__property AnsiString HREF;
```

Description

Use the HREF property when this adapter field is associated with an external image file URL. Use the OnGetHREF (see page 24260) event to dynamically define an external image file URL. If the image file will be dynamically generated, use the OnGetImage (see page 24260) and OnGetImageParams event handlers instead of the HREF property or the OnGetHREF (see page 24260) event handler.

See Also

ImplWebImageHREF (see page 24257)

OnGetHREF (see page 24260)

OnGetImage (see page 24260)

OnGetImageName (see page 24260)

173.1.50.3.2 TCustomAdapterImageField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 23921)

173.1.50.4 TCustomAdapterImageField Events

The events of the TCustomAdapterImageField class are listed here.

Public Events

	Name	Description
	OnGetHREF (see page 24260)	Provides the URL of an external image file associated with this adapter field.
	OnGetImage (see page 24260)	Retrieves a stream containing the image file associated with this adapter field.
	OnGetImageName (see page 24260)	Returns the name of the image file associated with this adapter field.
	OnGetParams (see page 24261)	Provides name/value pairs that identify this adapter field's image.

	OnUploadFile (see page 24261)	Updates this adapter field with an image file uploaded by the end-user.
	OnValidateFile (see page 24261)	Used to validate files uploaded by the end user.

173.1.50.4.1 TCustomAdapterImageField.OnGetHREF Event

Provides the URL of an external image file associated with this adapter field.

Pascal

```
property OnGetHREF: TImageFieldGetHREFEvent;
```

C++

```
__property TImageFieldGetHREFEvent OnGetHREF;
```

Description

Implement OnGetHREF to dynamically identify the URL of an external image file associated with this adapter field. When OnGetHREF is called, the HREF (see page 24259) parameter will contain the value of the HREF (see page 24259) property. To dynamically generate an image file for this adapter file, implement the OnGetImage (see page 24260) event handler instead.

See Also

ImplWebImageHREF (see page 24257)

HREF (see page 24259)

OnGetImage (see page 24260)

173.1.50.4.2 TCustomAdapterImageField.OnGetImage Event

Retrieves a stream containing the image file associated with this adapter field.

Pascal

```
property OnGetImage: TImageFieldGetImageEvent;
```

C++

```
__property TImageFieldGetImageEvent OnGetImage;
```

Description

Implement OnGetImage to dynamically generate an image file associated with this adapter field. Don't implement OnGetImage if the image file is external. Instead implement OnGetHREF (see page 24260) event handler or use the HREF (see page 24259) property to specify the URL of an external image.

See Also

RenderAdapterImage (see page 24258)

OnGetParams (see page 24261)

OnGetHREF (see page 24260)

HREF (see page 24259)

173.1.50.4.3 TCustomAdapterImageField.OnGetImageName Event

Returns the name of the image file associated with this adapter field.

Pascal

```
property OnGetImageName: TAdapterFieldGetStringEvent;
```

C++

```
__property TAdapterFieldGetStringEvent OnGetImageName;
```

Description

Implement this event handler to identify the name of the image file associated with this adapter field.

See Also

ImplGetValue (see page 24256)

173.1.50.4.4 TCustomAdapterImageField.OnGetParams Event

Provides name/value pairs that identify this adapter field's image.

Pascal

```
property OnGetParams: TImageFieldGetParamsEvent;
```

C++

```
__property TImageFieldGetParamsEvent OnGetParams;
```

Description

Implement this event handler to provide name/value pairs that are needed to identify this adapter field's image file. OnGetParams is called while generating an HTML page with a URL reference to this adapter field's image. The parameters returned by OnGetParams will be passed to OnGetImage (see page 24260) when the image file is requested.

See Also

GetAdapterItemRequestParams (see page 24253)

OnGetImage (see page 24260)

173.1.50.4.5 TCustomAdapterImageField.OnUploadFile Event

Updates this adapter field with an image file uploaded by the end-user.

Pascal

```
property OnUploadFile: TAdapterFieldFileEvent;
```

C++

```
__property TAdapterFieldFileEvent OnUploadFile;
```

Description

This event handler is called by ImplUpdateValue (see page 24257). This event handler should store the uploaded image file. Raise an exception to indicate an error.

See Also

ImplUpdateValue (see page 24257)

TAbstractWebRequestFile (see page 15073)

173.1.50.4.6 TCustomAdapterImageField.OnValidateFile Event

Used to validate files uploaded by the end user.

Pascal

```
property OnValidateFile: TAdapterFieldFileEvent;
```

C++

```
__property TAdapterFieldFileEvent OnValidateFile;
```

Description

Validates the file uploaded by the end-user. Raise an exception to indicate an error. For example, it raises an exception if the Mime type of the file is not supported.

See Also

DoOnValidateValue (see page 24253)

TAbstractWebRequestFile (see page 15073)

173.1.51 TCustomAdapterMemoField Class

TCustomAdapterMemoField is a base class for adapter fields that have a multi-line text value.

Class Hierarchy

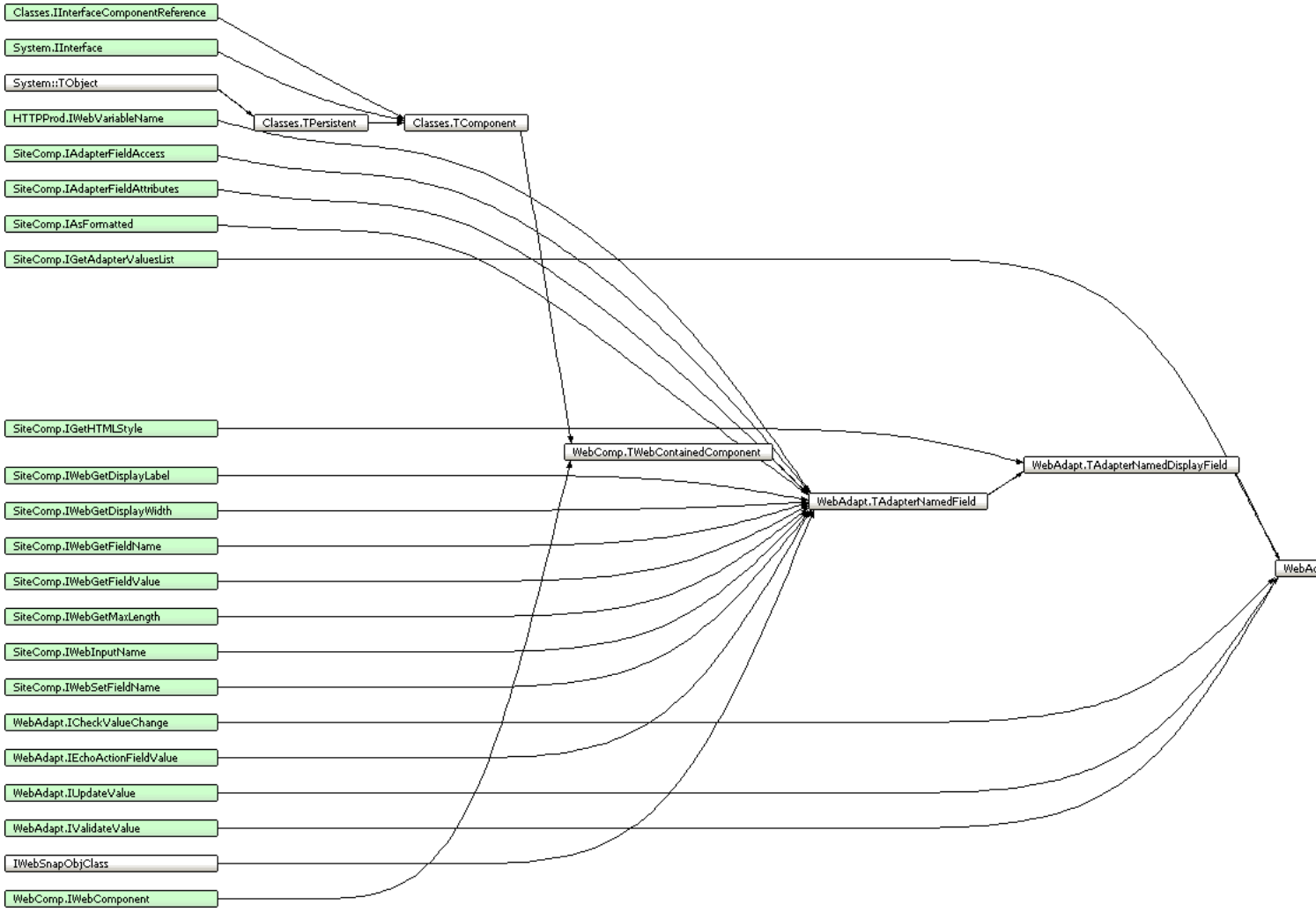

Pascal

```
TCustomAdapterMemoField = class(TAdapterUpdateField);
```

C++

```
class TCustomAdapterMemoField : public TAdapterUpdateField;
```

File

WebAdapt

Description

Derive from TCustomAdapterMemoField when the adapter field has a multi-line text value. Typically, a text area HTML element is used to edit the value of this adapter field.

See Also

TAdapterUpdateField (see page 24007)

TAdapterMemoField (see page 23889)

173.1.51.1 TCustomAdapterMemoField Members

The following tables list the members exposed by TCustomAdapterMemoField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 24015)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 24015)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 24015)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 24016)	Retrieves a value entered by the end-user.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoOnValidateValue (see page 24269)	Validates a value entered by the end user.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 24016)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 24270)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 24270)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.

	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetValueCount (see page 24017)	Provides the number of values associated with this adapter field.
	GetValues (see page 24017)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 24018)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 24270)	Indicates whether the end-user may have entered a new value for this adapter field.
	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 24018)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 24019)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 24019)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 24271)	Returns the value of this adapter field as a variant.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 24019)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 24020)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 24020)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 24271)	Updates the adapter field with value entered by the end-user.
	ImplValidateValue (see page 24020)	Validates values(s) entered by the end-user.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 24021)	Indicates whether this adapter field can support multiple values.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24021)	Clears references to a deleted component.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 23941)	Handles movement of the component to a different parent.

V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 24021)	Updates the adapter field with value(s) entered by the end-user.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateValue (see page 24022)	Validates values(s) entered by the end-user.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 24023)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetValue (see page 24272)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 24272)	Called to update the adapter field's value with a text value entered by the end-user.
	OnUploadFile (see page 24273)	Called to update the adapter field's value with a file uploaded by the end-user.
	OnValidateValue (see page 24024)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.

	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
	Create (see page 24014)	Creates an instance of TAdapterUpdateField (see page 24007).
	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterValuesList (see page 19695)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.

	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 23923)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23924)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.

	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 23924)	Indicates whether this adapter field is read-only.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	Value (see page 24022)	Retrieves the value associated with this adapter field.
	ValuesList (see page 24023)	Associates a list of values with the adapter field.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.51.2 TCustomAdapterMemoField Methods

The methods of the TCustomAdapterMemoField class are listed here.

Protected Methods

	Name	Description
V	DoOnValidateValue (see page 24269)	Validates a value entered by the end user.
V	GetDisplayStyleType (see page 24270)	Suggests an HTML display type for this adapter field.
V	GetInputStyleType (see page 24270)	Suggests an HTML input type for this adapter field.
V	ImplCheckValueChange (see page 24270)	Indicates whether the end-user may have entered a new value for this adapter field.
V	ImplGetValue (see page 24271)	Returns the value of this adapter field as a variant.
V	ImplUpdateValue (see page 24271)	Updates the adapter field with value entered by the end-user.

173.1.51.2.1 TCustomAdapterMemoField.DoOnValidateValue Method

Validates a value entered by the end user.

Pascal

```
procedure DoOnValidateValue(AValue: IActionFieldValue; var AHandled: Boolean); override;
```

C++

```
virtual __fastcall DoOnValidateValue(IActionFieldValue AValue, Boolean AHandled);
```

Description

DoOnValidateValue overrides TAdapterUpdateField (see page 24007)'s DoOnValidateValue method. After calling TAdapterUpdateField (see page 24007)'s DoOnValidateValue, DoOnValidate checks that only a single text file or file has been uploaded for this field. An exception will be raised to indicate a validation error.

See Also

DoOnValidateValue (see page 24016)

Required (see page 23924)

IActionFieldValue (see page 920)

OnValidateFiles (see page 24233)

173.1.51.2.2 TCustomAdapterMemoField.GetDisplayStyleType Method

Suggests an HTML display type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const AnsiString
AAdapterMode);
```

Description

GetDisplayStyleType overrides TAdapterNamedDisplayField (see page 23913)'s GetDisplayStyleType method. GetDispayStyleType returns htmldText, indicating that this adapter field should be displayed by as an HTML text block.

See Also

GetDisplayStyleType (see page 23920)

TAdapterDisplayHTMLElementType

173.1.51.2.3 TCustomAdapterMemoField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const AnsiString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType method. GetDispayStyleTypereturns htmliTextArea, indicating that this adapter field's value should be input with an HTML multiline input field.

See Also

GetInputStyleType (see page 23921)

TAdapterInputHTMLElementType

173.1.51.2.4 TCustomAdapterMemoField.ImplCheckValueChange Method

Indicates whether the end-user may have entered a new value for this adapter field.

Pascal

```
function ImplCheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
override;
```

C++

```
virtual __fastcall Boolean ImplCheckValueChange(IActionRequest AActionRequest, int
AFieldIndex);
```

Description

ImplCheckValueChange overrides TAdapterUpdateField (see page 24007)'s ImplCheckValueChange method. ImplCheckValueChange returns true if the HTTP request has a new value for this adapter field.

See Also

ImplCheckValueChange (see page 24018)

173.1.51.2.5 TCustomAdapterMemoField.ImplGetValue Method

Returns the value of this adapter field as a variant.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue function returns the value of this adapter field as a variant. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to view this field, an exception is raised. If the EchoActionFieldValue (see page 23944) property is true, then the value entered by end-user should is returned. If the EchoActionFieldValue (see page 23944) property is false then the OnGetValue (see page 24272) event handler is called to retrieve the value of the adapter field.

See Also

OnGetValue (see page 24272)

EchoActionFieldValue (see page 23944)

173.1.51.2.6 TCustomAdapterMemoField.ImplUpdateValue Method

Updates the adapter field with value entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The ImplUpdateValue procedure updates the value of this adapter field using the given action request. The AFieldIndex parameter is used to extract the value entered by the user from the action request. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to modify this field, an exception is raised. If action request contains a string value for this adapter field, the OnUpdateValue (see page 24272) event handler is called. If the action request contains an uploaded file for this adapter field, the OnUploadFile (see page 24273) event handler is called.

See Also

OnUpdateValue (see page 24272)

OnUploadFile (see page 24273)

ImplUpdateValue (see page 24020)

173.1.51.3 TCustomAdapterMemoField Events

The events of the TCustomAdapterMemoField class are listed here.

Public Events

	Name	Description
	OnGetValue (see page 24272)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 24272)	Called to update the adapter field's value with a text value entered by the end-user.
	OnUploadFile (see page 24273)	Called to update the adapter field's value with a file uploaded by the end-user.

173.1.51.3.1 TCustomAdapterMemoField.OnGetValue Event

Called to retrieve the adapter field's value.

Pascal

```
property OnGetValue: TAdapterFieldGetStringEvent;
```

C++

```
__property TAdapterFieldGetStringEvent OnGetValue;
```

Description

OnGetValue is called by ImplGetValue (see page 24271) to retrieve the value associated with this adapter field.

See Also

ImplGetValue (see page 24271)

173.1.51.3.2 TCustomAdapterMemoField.OnUpdateValue Event

Called to update the adapter field's value with a text value entered by the end-user.

Pascal

```
property OnUpdateValue: TAdapterFieldStringEvent;
```

C++

```
__property TAdapterFieldStringEvent OnUpdateValue;
```

Description

OnUpdateValue is called when an action request contains a value for this adapter field. If this event handler is not implemented, the value entered by the end-user will be ignored.

See Also

ImplUpdateValue (see page 24271)

173.1.51.3.3 TCustomAdapterMemoField.OnUploadFile Event

Called to update the adapter field's value with a file uploaded by the end-user.

Pascal

```
property OnUploadFile: TAdapterFieldFileEvent;
```

C++

```
__property TAdapterFieldFileEvent OnUploadFile;
```

Description

OnUploadFile is called when an action request contains an uploaded file for this adapter field. If this event handler is not implemented, the file uploaded by the end-user will be ignored.

See Also

ImplUpdateValue (see page 24271)

173.1.52 TCustomAdapterMultiValueField Class

TCustomAdapterMultiValueField is a base class for adapter fields that have one or more values.

Class Hierarchy

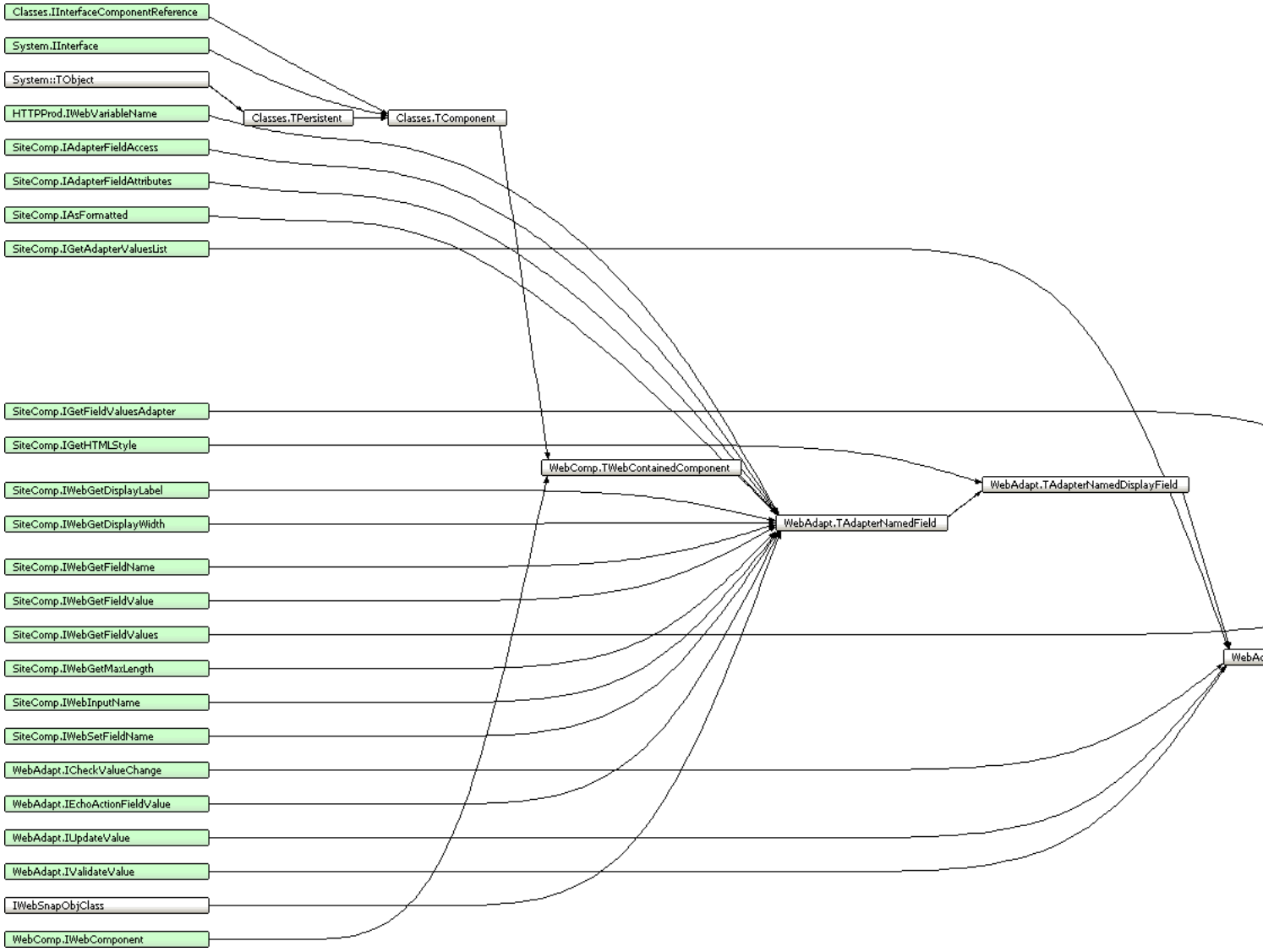

Pascal

```
TCustomAdapterMultiValueField = class(TBaseAdapterMultiValueField);
```

C++

```
class TCustomAdapterMultiValueField : public TBaseAdapterMultiValueField;
```

File

WebAdapt

Description

Derive from TCustomAdapterMultiValueField when the adapter field can have multiple values. Typically, multi-value fields are represented in HTML with multiple selection select lists or check boxes.

See Also

TBaseAdapterMultiValueField (see page 24086)

TAdapterMultiValueField (see page 23900)

173.1.52.1 TCustomAdapterMultiValueField Members

The following tables list the members exposed by TCustomAdapterMultiValueField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 24015)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 24015)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 24015)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 24016)	Retrieves a value entered by the end-user.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoOnValidateValue (see page 24281)	Validates values entered by the end-user.
	EndValueIterator (see page 24093)	Completes iteration.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 24016)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetCurrentValue (see page 24094)	Implements IWebGetFieldValues' GetCurrentValue.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 23932)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 23920)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 24281)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetFieldValuesAdapter (see page 24094)	Returns the adapter object that implements the Values property.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 23920)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 24282)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetValueCount (see page 24282)	Returns the number of values associated with this adapter field.
	GetValues (see page 24282)	Returns one of the values associated with the adapter field.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 23921)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 23921)	Suggests a view mode for this adapter field.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasValue (see page 24095)	Implements IWebGetFieldValues' HasValue method.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 24018)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 24018)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplGetCurrentValue (see page 24095)	Implements IWebGetFieldValues' GetCurrentValue (see page 24094) method.
	ImplGetDisplayLabel (see page 23922)	Implements interface access to the DisplayLabel (see page 23923) property.
	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 24018)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 23922)	Implements interface access to the DisplayWidth (see page 23924) property.
	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 24019)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 24019)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetFieldValuesAdapter (see page 24096)	Returns the adapter object that implements the Values property.
	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 23922)	Implements interface access to the Required (see page 23924) property.
	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 23939)	Gets the value of this adapter field as a Variant.
	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 24019)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
	ImplHasValue (see page 24283)	Implements IWebGetFieldValues' HasValue (see page 24095) method.
	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 24020)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.

V	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
V	ImplSetFieldName (see page 24020)	Sets the name used to access this adapter field with server-side script.
V	ImplUpdateValue (see page 24283)	Updates the adapter field with value(s) entered by the end-user.
V	ImplValidateValue (see page 24020)	Validates values(s) entered by the end-user.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	IsMultiValueField (see page 24021)	Indicates whether this adapter field can support multiple values.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextValueIteration (see page 24096)	Implements IWebGetFieldValues' NextValueIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 24021)	Clears references to a deleted component.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 23923)	SetDisplayLabel sets the value of the DisplayLabel (see page 23923) property.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartValueIterator (see page 24097)	Implements IWebGetFieldValues' StartValueIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	UpdateValue (see page 24021)	Updates the adapter field with value(s) entered by the end-user.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateValue (see page 24022)	Validates values(s) entered by the end-user.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 24023)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetValueCount (see page 24285)	Responsible for returning the count of values associated with the adapter field.
	OnGetValues (see page 24285)	Responsible for returning one of the values associated with the adapter field.
	OnHasValue (see page 24285)	Indicates whether a particular value is associated with this adapter field.
	OnUpdateValues (see page 24286)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateValue (see page 24024)	Event handler called to validate value(s) entered by the end-user.
	OnValidateValues (see page 24286)	Validates field values entered by the end-user.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 23919)	Creates an instance of TAdapterNamedDisplayField (see page 23913).
V	Create (see page 24014)	Creates an instance of TAdapterUpdateField (see page 24007).
V	Create (see page 24092)	Constructor for TBaseAdapterMultiValueField (see page 24086).
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24093)	Destructor for TBaseAdapterMultiValueField (see page 24086).
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndValueIterator (see page 19761)	Indicate the iteration is finished.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetAdapterValuesList (see page 19695)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetCurrentValue (see page 19761)	Retrieves the current value as a variant.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetFieldValuesAdapter (see page 19698)	This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasValue (see page 19761)	Indicates whether the adapter field has a matching value.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextValueIteration (see page 19762)	Position the iteration pointer on the next value.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.

	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartValueIterator (see page 19762)	Reset the iteration pointer to the first item.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 23923)	Suggests a display label for this adapter field.
	DisplayWidth (see page 23924)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ReadOnly (see page 23924)	Indicates whether this adapter field is read-only.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 23924)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	Value (see page 24022)	Retrieves the value associated with this adapter field.
R	ValueCount (see page 24284)	Returns the number of values associated with this adapter field.
R	Values (see page 24284)	Returns one of the values associated with the adapter field.
	ValuesList (see page 24023)	Associates a list of values with the adapter field.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.52.2 TCustomAdapterMultiValueField Methods

The methods of the TCustomAdapterMultiValueField class are listed here.

Protected Methods

	Name	Description
	DoOnValidateValue (see page 24281)	Validates values entered by the end-user.
	GetDisplayStyleType (see page 24281)	Suggests an HTML display type for this adapter field.
	GetInputStyleType (see page 24282)	Suggests an HTML input type for this adapter field.
	GetValueCount (see page 24282)	Returns the number of values associated with this adapter field.
	GetValues (see page 24282)	Returns one of the values associated with the adapter field.
	ImplHasValue (see page 24283)	Implements IWebGetFieldValues' HasValue (see page 24095) method.
	ImplUpdateValue (see page 24283)	Updates the adapter field with value(s) entered by the end-user.

173.1.52.2.1 TCustomAdapterMultiValueField.DoOnValidateValue Method

Validates values entered by the end-user.

Pascal

```
procedure DoOnValidateValue(AValue: IActionFieldValue; var AHandled: Boolean); override;
```

C++

```
virtual __fastcall DoOnValidateValue(IActionFieldValue AValue, Boolean AHandled);
```

Description

DoOnValidateValue overrides TAdapterUpdateField (see page 24007)'s DoOnValidateValue method. If the OnValidateValues (see page 24286) event handler is not nil (Delphi) or NULL (C++), it is called it to validate values. If the Required (see page 23924) property is true, DoOnValidateValues also validates that 1 of more values have been entered by the end user. An exception will be raised to indicate a validation error.

See Also

DoOnValidateValue (see page 24016)

Required (see page 23924)

OnValidateValues (see page 24286)

173.1.52.2.2 TCustomAdapterMultiValueField.GetDisplayStyleType Method

Suggests an HTML display type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const AnsiString
AAdapterMode);
```

Description

GetDisplayStyleType returns the display type for an adapter field. Here, it returns htmldList to indicate a list.

See Also

GetDisplayStyleType (see page 23920)

173.1.52.2.3 TCustomAdapterMultiValueField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const AnsiString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType method. If the ValuesList (see page 24023) property is not nil (Delphi) or NULL (C++), GetInputStyleType returns htmliCheckBox indicating that this adapter field should be edited using a group of check boxes. If the ValuesList (see page 24023) property is nil (Delphi) or NULL (C++), then htmliTextInput is returned. htmliTextInput indicates that the adapter field value should be edited with a single line input.

See Also

GetInputStyleType (see page 23921)

ValuesList (see page 24023)

173.1.52.2.4 TCustomAdapterMultiValueField.GetValueCount Method

Returns the number of values associated with this adapter field.

Pascal

```
function GetValueCount: Integer; override;
```

C++

```
virtual __fastcall int GetValueCount();
```

Description

GetValueCount implements the ValueCount (see page 24284) property. If the EchoActionFieldValue (see page 23944) flag is true then the result is extracted from the HTTP request. Otherwise, the OnGetValueCount (see page 24285) event is called. GetValueCount returns zero if a value can't be found because OnGetValueCount (see page 24285) is nil (Delphi) or NULL (C++) or the HTTP request does not contain a value for this adapter field.

See Also

ValueCount (see page 24284)

Values (see page 24284)

EchoActionFieldValue (see page 23944)

173.1.52.2.5 TCustomAdapterMultiValueField.GetValues Method

Returns one of the values associated with the adapter field.

Pascal

```
function GetValues(I: Integer): Variant; override;
```

C++

```
virtual __fastcall Variant GetValues(int I);
```

Description

GetValues implements the Values (see page 24284) property. If the EchoActionFieldValue (see page 23944) flag is true then the result is extracted from the HTTP request. Otherwise, the OnGetValues (see page 24285) event is called. GetValues returns Unassigned if a value can't be found because OnGetValues (see page 24285) is nil (Delphi) or NULL (C++) or the HTTP request does not contain a value for this adapter field.

See Also

Values (see page 24284)

OnGetValues (see page 24285)

EchoActionFieldValue (see page 23944)

EchoActionFieldValue (see page 23944)

173.1.52.2.6 TCustomAdapterMultiValueField.ImplHasValue Method

Implements IWebGetFieldValues' HasValue (see page 24095) method.

Pascal

```
function ImplHasValue(const Value: Variant): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplHasValue(const Variant Value);
```

Description

ImplHasValue overrides TBaseAdapterMultiValueField (see page 24086)'s ImplHasValue method. If the OnHasValue (see page 24285) event is not nil (Delphi) or NULL (C++), it will be called for the result of ImplHasValue. Otherwise, ImplHasValue will loop through the values of the adapter field, looking for a matching value.

See Also

ImplHasValue (see page 24096)

HasValue (see page 19761)

173.1.52.2.7 TCustomAdapterMultiValueField.ImplUpdateValue Method

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

ImplUpdateValue overrides TAdapterUpdateField (see page 24007)'s ImplUpdateValue method. After checking that the end-user has rights to modify this adapter field, the OnUpdateValues (see page 24286) event is called.

See Also

OnUpdateValues (see page 24286)

ImplUpdateValue (see page 24020)

173.1.52.3 TCustomAdapterMultiValueField Properties

The properties of the TCustomAdapterMultiValueField class are listed here.

Public Properties

	Name	Description
R	ValueCount (see page 24284)	Returns the number of values associated with this adapter field.
R	Values (see page 24284)	Returns one of the values associated with the adapter field.

173.1.52.3.1 TCustomAdapterMultiValueField.ValueCount Property

Returns the number of values associated with this adapter field.

Pascal

```
property ValueCount: Integer;
```

C++

```
__property int ValueCount;
```

Description

ValueCount can be used with the Values (see page 24284) property to access all the values associated with this adapter field. ValueCount is implemented by GetValueCount (see page 24282).

See Also

GetValueCount (see page 24282)

Values (see page 24284)

173.1.52.3.2 TCustomAdapterMultiValueField.Values Property

Returns one of the values associated with the adapter field.

Pascal

```
property Values [I: Integer]: Variant;
```

C++

```
__property Variant Values[int I];
```

Description

Values can be used with the ValueCount (see page 24284) property to access all the values associated with this adapter field. Values is implemented by GetValues (see page 24282).

See Also

GetValues (see page 24282)

ValueCount (see page 24284)

173.1.52.4 TCustomAdapterMultiValueField Events

The events of the TCustomAdapterMultiValueField class are listed here.

Public Events

	Name	Description
	OnGetValueCount (see page 24285)	Responsible for returning the count of values associated with the adapter field.
	OnGetValues (see page 24285)	Responsible for returning one of the values associated with the adapter field.
	OnHasValue (see page 24285)	Indicates whether a particular value is associated with this adapter field.
	OnUpdateValues (see page 24286)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateValues (see page 24286)	Validates field values entered by the end-user.

173.1.52.4.1 TCustomAdapterMultiValueField.OnGetValueCount Event

Responsible for returning the count of values associated with the adapter field.

Pascal

```
property OnGetValueCount: TAdapterFieldGetValueCountEvent;
```

C++

```
__property TAdapterFieldGetValueCountEvent OnGetValueCount;
```

Description

Implement this event handler to provide the count of values associated with an adapter field. Also implement OnGetValues (see page 24285) to provide the individual values.

See Also

GetValueCount (see page 24282)

OnGetValues (see page 24285)

173.1.52.4.2 TCustomAdapterMultiValueField.OnGetValues Event

Responsible for returning one of the values associated with the adapter field.

Pascal

```
property OnGetValues: TAdapterFieldGetMultiValueEvent;
```

C++

```
__property TAdapterFieldGetMultiValueEvent OnGetValues;
```

Description

Implement this event handler to provide the values associated with an adapter field. The Index parameter is zero-based. Also implement OnGetValueCount (see page 24285) to provide the count of values associated with this adapter field.

See Also

GetValues (see page 24282)

OnGetValueCount (see page 24285)

173.1.52.4.3 TCustomAdapterMultiValueField.OnHasValue Event

Indicates whether a particular value is associated with this adapter field.

Pascal

```
property OnHasValue: TAdapterFieldHasValueEvent;
```

C++

```
__property TAdapterFieldHasValueEvent OnHasValue;
```

Description

ImplHasValue (see page 24283) calls this event handler. Implement this event handler to lookup a matching value. Implementation of this event handler is optional. If OnHasValue is nil, ImplHasValue (see page 24283) will search for a matching value the Values (see page 24284) and ValueCount (see page 24284) properties.

See Also

ImplHasValue (see page 24283)

ValueCount (see page 24284)

Values (see page 24284)

173.1.52.4.4 TCustomAdapterMultiValueField.OnUpdateValues Event

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
property OnUpdateValues: TAdapterFieldStringsEvent;
```

C++

```
__property TAdapterFieldStringsEvent OnUpdateValues;
```

Description

Implement this event handler to process values entered by the end-user. Raise an exception if an error occurs.

See Also

OnValidateValues (see page 24286)

ImplUpdateValue (see page 24283)

173.1.52.4.5 TCustomAdapterMultiValueField.OnValidateValues Event

Validates field values entered by the end-user.

Pascal

```
property OnValidateValues: TAdapterFieldStringsEvent;
```

C++

```
__property TAdapterFieldStringsEvent OnValidateValues;
```

Description

Implement this event handler to validate values entered by the end-user. Raise an exception if an error occurs.

See Also

OnValidateValues

173.1.53 TCustomAdapterPageAction Class

TCustomPagedAdapterAction (see page 24371) is a base class for adapter actions that execute the next or previous page

commands.

Class Hierarchy

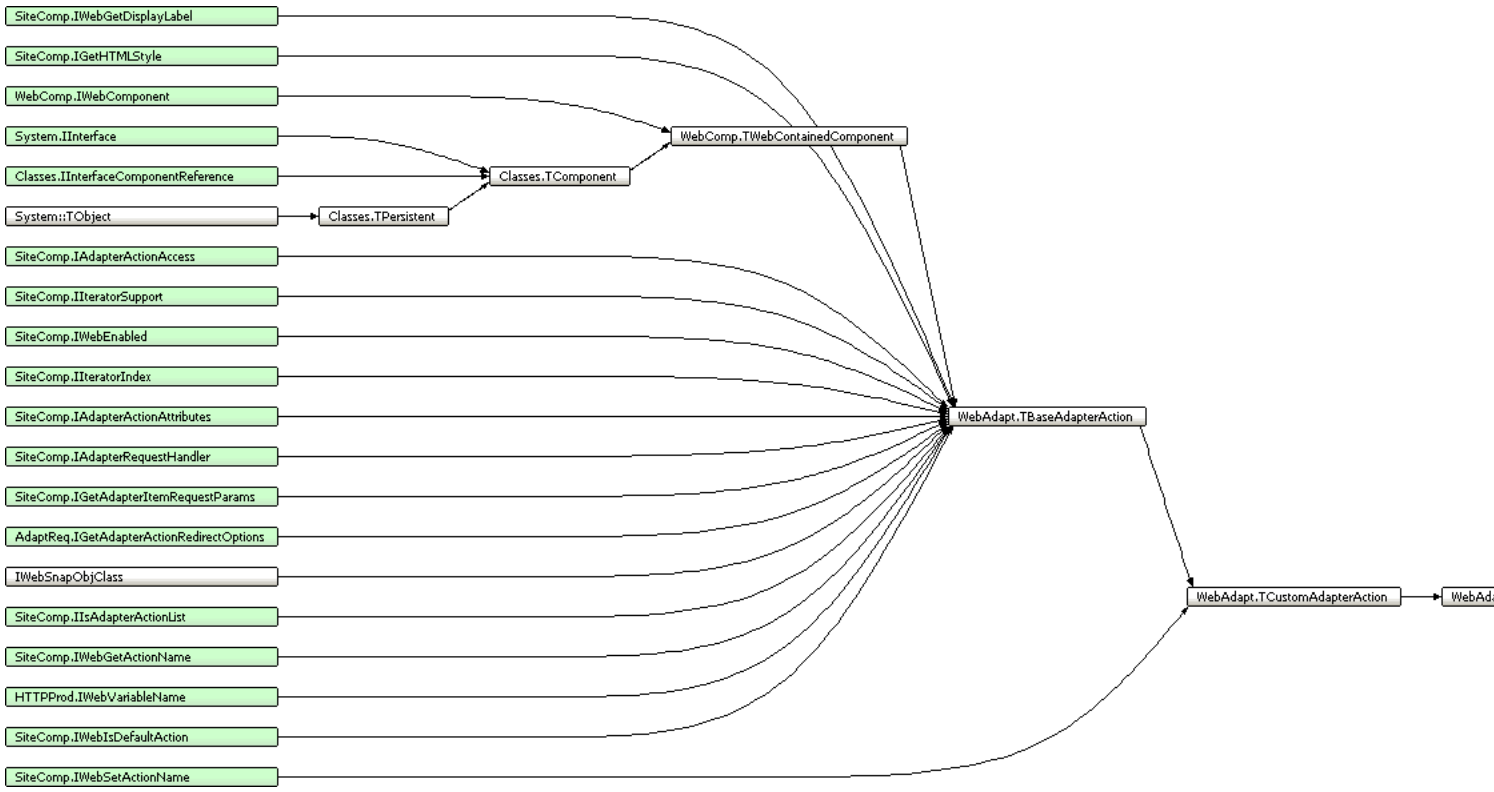

Pascal

```
TCustomAdapterPageAction = class(TCustomPagedAdapterAction);
```

C++

```
class TCustomAdapterPageAction : public TCustomPagedAdapterAction;
```

File

WebAdapt

Description

TCustomPagedAdapterAction (see page 24371) provide methods that are used by TAdapterNextPageAction (see page 23945) and TAdapterPrevPageAction (see page 23979).

See Also

TDefaultFieldsPagedAdapter (see page 24403)

TCustomPagedAdapterAction (see page 24371)

TAdapterNextPageAction (see page 23945)

TAdapterPrevPageAction (see page 23979)

173.1.53.1 TCustomAdapterPageAction Members

The following tables list the members exposed by TCustomAdapterPageAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 24377)	Returns the parent adapter of this adapter action.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 24055)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24293)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
V	GetNewPage (see page 24294)	Indicates the page to display when this action is executed.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 24294)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 24294)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 24295)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 24295)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.

	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Adapter (see page 24378)	Returns the parent adapter of this adapter action.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 24075)	Called after the adapter action has executed.
	OnAfterExecute (see page 24198)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 24075)	Called after the adapter action response is generated.
	OnAfterGetResponse (see page 24199)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24076)	Called before the adapter action is executed.
	OnBeforeExecute (see page 24199)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24076)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 24200)	Called before the adapter action response is generated.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 24077)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.

	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.

	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.

	DisplayLabel (see page 24073)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 24073)	Defines the rights required to execute this action.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RedirectOptions (see page 24074)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.53.2 TCustomAdapterPageAction Methods

The methods of the TCustomAdapterPageAction class are listed here.

Protected Methods

	Name	Description
V	GetDisplayStyleType (see page 24293)	Suggests an HTML display type for this adapter action.
V	GetNewPage (see page 24294)	Indicates the page to display when this action is executed.
V	ImplExecuteActionRequest (see page 24294)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetAdapterItemRequestParams (see page 24294)	Provides the name/value pairs needed to execute this adapter action.
V	ImplIsDefaultAction (see page 24295)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	ImplWebEnabled (see page 24295)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

173.1.53.2.1 TCustomAdapterPageAction.GetDisplayStyleType Method

Suggests an HTML display type for this adapter action.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterActionHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterActionHTMLElementType GetDisplayStyleType(const AnsiString
AAdapterMode);
```

Description

GetDisplayStyleType returns htmlaAnchor, indicating that next page and previous page commands should be displayed using a hyperlink.

See Also

GetDisplayStyle (see page 24056)

173.1.53.2.2 TCustomAdapterPageAction.GetNewPage Method

Indicates the page to display when this action is executed.

Pascal

```
function GetNewPage(var AValue: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetNewPage(int AValue);
```

Description

This is a placeholder. Descendant classes must override this method. true is returned if the action can move to a new page. The AValue parameter identifies the new page. GetNewPage is called by ImplExecuteActionRequest (see page 24294).

See Also

GetNewPage (see page 23953)

GetNewPage (see page 23987)

ImplExecuteActionRequest (see page 24294)

173.1.53.2.3 TCustomAdapterPageAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest is called by ExecuteActionRequest (see page 24053). ImplExecuteActionRequest calls ExtractRequestParams followed by GetNewPage (see page 24294). If GetNewPage (see page 24294) returns true then the parent adapter's current page is changed.

See Also

ExecuteActionRequest (see page 24053)

ExtractRequestParams (see page 24412)

GetNewPage (see page 24294)

CurrentPage (see page 24418)

173.1.53.2.4 TCustomAdapterPageAction.ImplGetAdapterItemRequestParams Method

Provides the name/value pairs needed to execute this adapter action.

Pascal

```
procedure ImplGetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams); override;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(AnsiString AIdentifier,
IAdapterItemRequestParams Params);
```

Description

ImplGetAdapterItemRequestParams is called by GetAdapterItemRequestParams (see page 24055). ImplGetAdapterItemRequestParams calls the parent adapter's EncodeActionParams method to add the current page as a parameter. Then the inherited ImplGetAdapterItemRequestParams method is called so that any event handlers that may be defined will be called.

See Also

ImplGetAdapterItemRequestParams (see page 24063)

GetAdapterItemRequestParams (see page 24055)

EncodeActionParams (see page 24411)

OnGetActionParams (see page 24187)

OnGetParams (see page 24077)

173.1.53.2.5 TCustomAdapterPageAction.ImplIsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function ImplIsDefaultAction(ADisplay: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplIsDefaultAction(TObject * ADisplay);
```

Description

ImplIsDefaultAction is called by IsDefaultAction (see page 24070). ImplIsDefaultAction returns true if the display characteristics identified by ADisplay indicate the a previous or next page command is appropriate.

See Also

IsDefaultAction (see page 24070)

ImplIsDefaultAction (see page 24243)

173.1.53.2.6 TCustomAdapterPageAction.ImplWebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

ImplWebEnabled is called by WebEnabled (see page 24071). ImplWebEnabled returns the result of GetNewPage (see page

24294).

See Also

WebEnabled (see page 24071)

GetNewPage (see page 24294)

173.1.54 TCustomApplicationAdapter Class

TCustomApplicationAdapter is a base class for components that implement the server-side script Application object.

Class Hierarchy

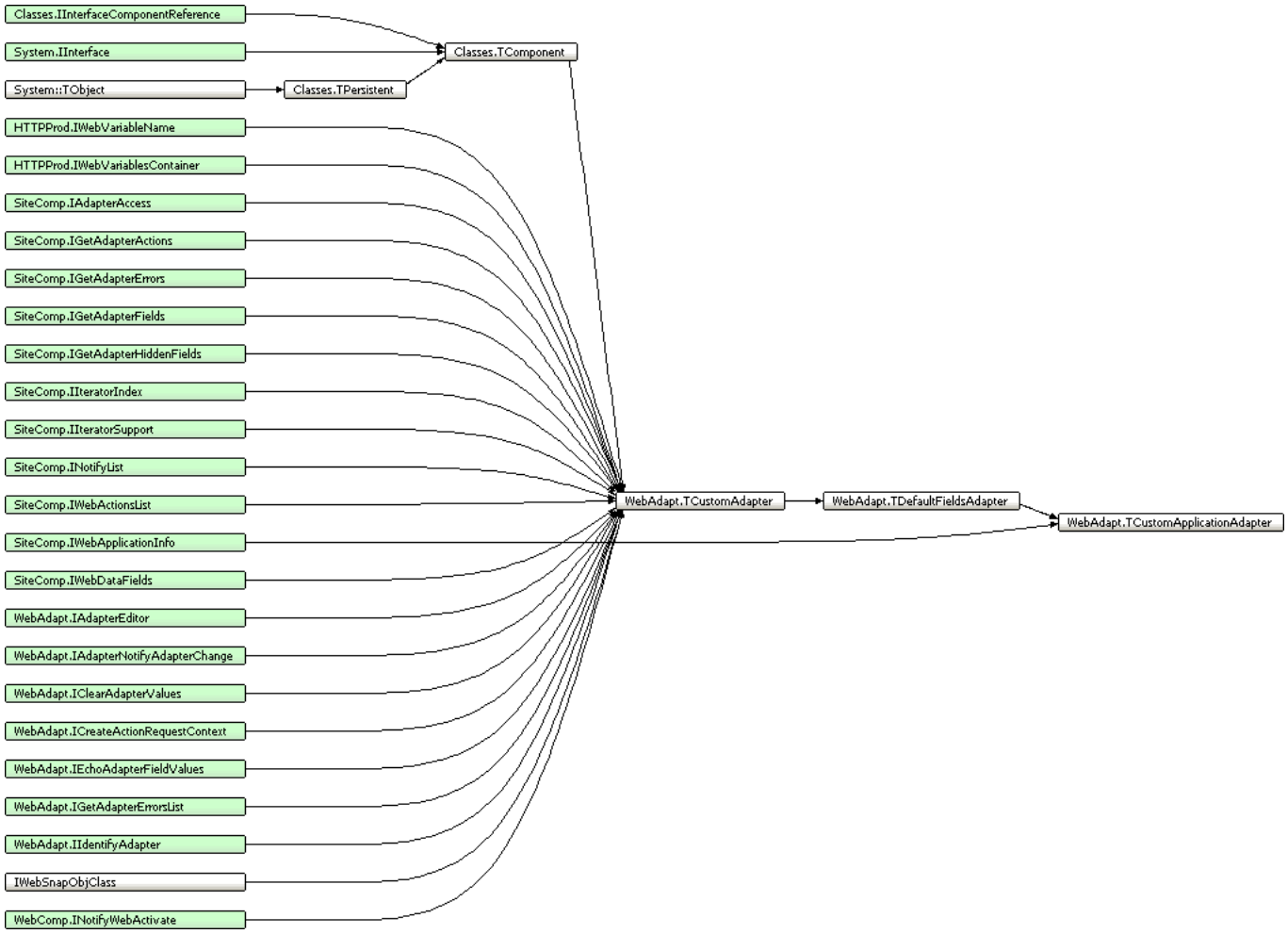

Pascal

```
TCustomApplicationAdapter = class(TDefaultFieldsAdapter, IWebApplicationInfo);
```

C++

```
class TCustomApplicationAdapter : public TDefaultFieldsAdapter, public IWebApplicationInfo;
```

File

WebAdapt

Description

TCustomApplicationAdapter implements IWebApplicationInfo. This interface is used to support server-side scripting. TCustomApplicationAdapter automatically adds an adapter field that provides the application title. Additional fields and actions may be added to this adapter. They are accessed as named properties of the server-side script Application object.

173.1.54.1 TCustomApplicationAdapter Members

The following tables list the members exposed by TCustomApplicationAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.

	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetApplicationAdapter (see page 24303)	Implements IWebApplicationInfo's GetApplicationAdapter method.
	GetApplicationTitle (see page 24303)	Implements IWebApplicationInfo's GetApplicationTitle method.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 24304)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 24304)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 24304)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 24305)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 24305)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 24166)	Returns an action or field.

	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.

	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 24184)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24185)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24186)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24186)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 24187)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.

	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 24181)	Actions is the container for adapter actions components.
R	Actions (see page 24402)	Container for adapter actions components.
R	ApplicationAdapter (see page 19747)	Provides access to the adapter that implements the fields and actions of the Application variable in server-side scripts.
	ApplicationTitle (see page 24305)	Defines the title of the application.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Data (see page 24181)	Data is the container for adapter field components.
R	Data (see page 24402)	Container for adapter field components.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.

R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.54.2 TCustomApplicationAdapter Methods

The methods of the TCustomApplicationAdapter class are listed here.

Protected Methods

	Name	Description
	GetApplicationAdapter (see page 24303)	Implements IWebApplicationInfo's GetApplicationAdapter method.
	GetApplicationTitle (see page 24303)	Implements IWebApplicationInfo's GetApplicationTitle method.
V	GetWebSnapObjClass (see page 24304)	Returns the class type for this adapter field.
V	ImplCanAddActionClass (see page 24304)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 24304)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplGetActionsList (see page 24305)	Provides a list of action types that can be added to this adapter.
V	ImplGetFieldsList (see page 24305)	Provides a list of field types that can be added to this adapter.

173.1.54.2.1 TCustomApplicationAdapter.GetApplicationAdapter Method

Implements IWebApplicationInfo's GetApplicationAdapter method.

Pascal

```
function GetApplicationAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetApplicationAdapter();
```

Description

GetApplicationAdapter returns Self.

See Also

GetApplicationAdapter (see page 19747)

173.1.54.2.2 TCustomApplicationAdapter.GetApplicationTitle Method

Implements IWebApplicationInfo's GetApplicationTitle method.

Pascal

```
function GetApplicationTitle: string;
```

C++

```
__fastcall AnsiString GetApplicationTitle();
```

Description

GetApplicationTitle returns the application title that can be accessed with server-side script as the Application.Title property.

See Also

GetApplicationTitle (see page 19747)

173.1.54.2.3 TCustomApplicationAdapter.GetWebSnapObjClass Method

Returns the class type for this adapter field.

Pascal

```
function GetWebSnapObjClass: TObjectScripterClass; override;
```

C++

```
virtual __fastcall TObjectScripterClass GetWebSnapObjClass();
```

Description

This function returns the class type for this adapter field, which is used by the script engine to create a script object. GetWebSnapObjClass tells the script engine what type of object to create.

173.1.54.2.4 TCustomApplicationAdapter.ImplCanAddActionClass Method

Indicates whether a type of adapter action can be added to this adapter.

Pascal

```
function ImplCanAddActionClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddActionClass allows actions classes that derive from TCustomApplicationAdapterAction (see page 24306) in addition to those allows by TCustomAdapter (see page 24139)'s ImplCanAddActionsClass method.

See Also

ImplCanAddActionClass (see page 24163)

TCustomApplicationAdapterAction (see page 24306)

173.1.54.2.5 TCustomApplicationAdapter.ImplCanAddFieldClass Method

Indicates whether a type of adapter field can be added to this adapter.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddFieldClass allows field classes that derive from TApplicationAdapterField in addition to those allowed by TCustomAdapter (see page 24139)'s ImplCanAddFieldClass method.

See Also

ImplCanAddFieldClass (see page 24163)

173.1.54.2.6 TCustomApplicationAdapter.ImplGetActionsList Method

Provides a list of action types that can be added to this adapter.

Pascal

```
procedure ImplGetActionsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetActionsList(TStrings AList);
```

Description

ImplGetActionsList clears the list. There are no default actions.

173.1.54.2.7 TCustomApplicationAdapter.ImplGetFieldsList Method

Provides a list of field types that can be added to this adapter.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

ImplGetActionsList (see page 24305) lists the title field.

See Also

TAdapterApplicationTitleField (see page 23752)

173.1.54.3 TCustomApplicationAdapter Properties

The properties of the TCustomApplicationAdapter class are listed here.

Public Properties

	Name	Description
	ApplicationTitle (see page 24305)	Defines the title of the application.

173.1.54.3.1 TCustomApplicationAdapter.ApplicationTitle Property

Defines the title of the application.

Pascal

```
property ApplicationTitle: string;
```

C++

```
__property AnsiString ApplicationTitle;
```

Description

Set this property to identify an application title that can be accessed with server-side script as the Application.Title property.

173.1.55 TCustomApplicationAdapterAction Class

TCustomApplicationAdapterAction is a base class for adapter actions that work with descendants of TCustomApplicationAdapter (see page 24296).

Class Hierarchy

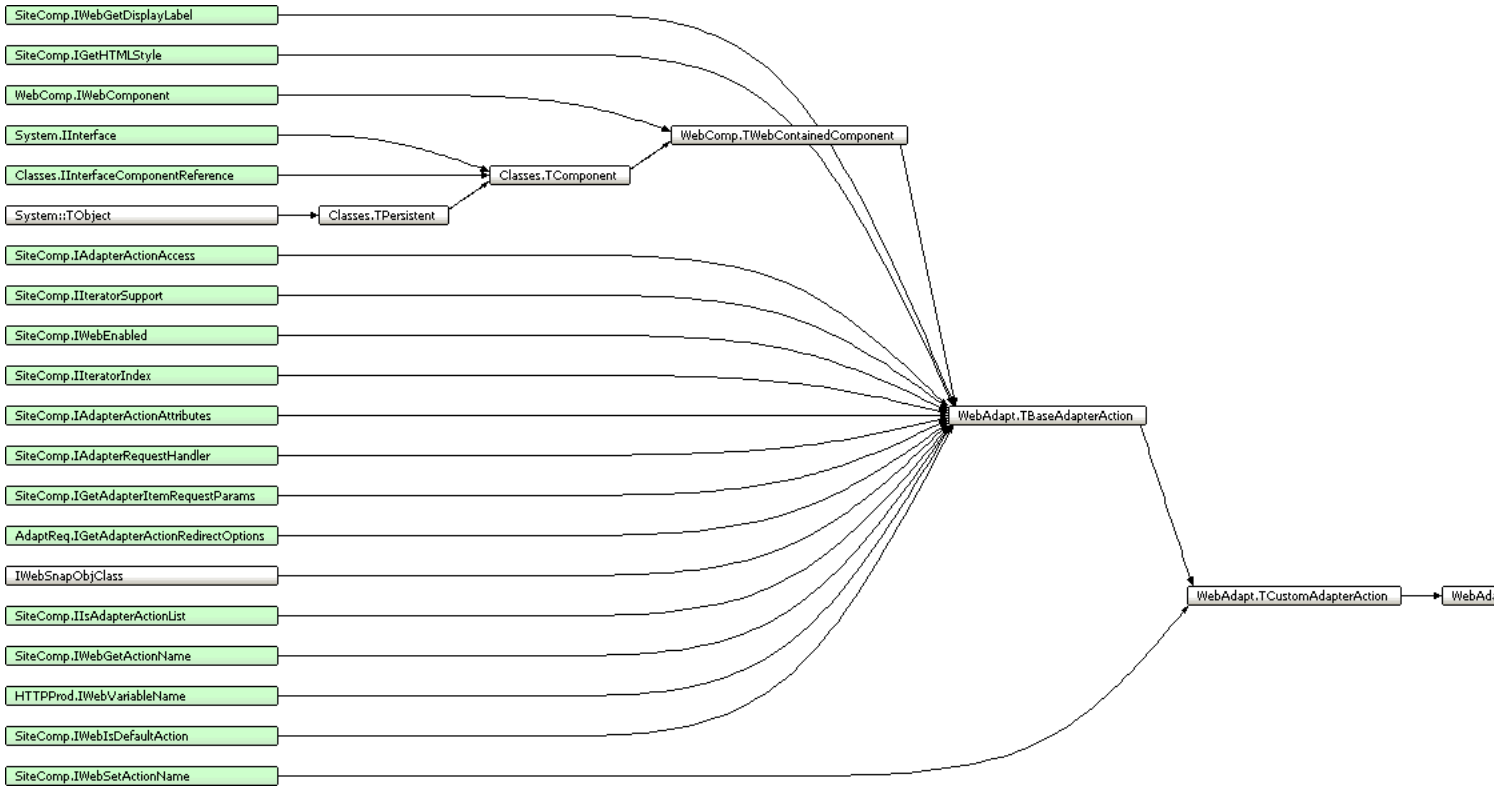

Pascal

```
TCustomApplicationAdapterAction = class(TImplementedAdapterAction);
```

C++

```
class TCustomApplicationAdapterAction : public TImplementedAdapterAction;
```

File

WebAdapt

Description

TCustomApplicationAdapterAction has two purposes. First, it provides a property that is used by descendants to access the parent adapter. Second, it defines a base class that TCustomApplicationAdapter (see page 24296) checks for when

determining whether an adapter action can be added as a child.

See Also

TImplementedAdapterAction (see page 24491)

TCustomAdapterAction (see page 24188)

TAdapter (see page 23713)

TApplicationAdapter (see page 24032)

173.1.55.1 TCustomApplicationAdapterAction Members

The following tables list the members exposed by TCustomApplicationAdapterAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 24055)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.

	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 24063)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 24195)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 24069)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 24197)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24312)	Returns the parent adapter of this adapter action.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 24075)	Called after the adapter action has executed.
	OnAfterExecute (see page 24198)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 24075)	Called after the adapter action response is generated.

	OnAfterGetResponse (see page 24199)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24076)	Called before the adapter action is executed.
	OnBeforeExecute (see page 24199)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24076)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 24200)	Called before the adapter action response is generated.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 24077)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.

	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 24073)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 24073)	Defines the rights required to execute this action.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RedirectOptions (see page 24074)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.55.2 TCustomApplicationAdapterAction Properties

The properties of the TCustomApplicationAdapterAction class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 24312)	Returns the parent adapter of this adapter action.

173.1.55.2.1 TCustomApplicationAdapterAction.Adapter Property

Returns the parent adapter of this adapter action.

Pascal

```
property Adapter: TCustomApplicationAdapter;
```

C++

```
__property TCustomApplicationAdapter Adapter;
```

Description

Descendant classes use this property to access the parent adapter.

173.1.56 TCustomEndUserAdapter Class

TCustomEndUserAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.

Class Hierarchy

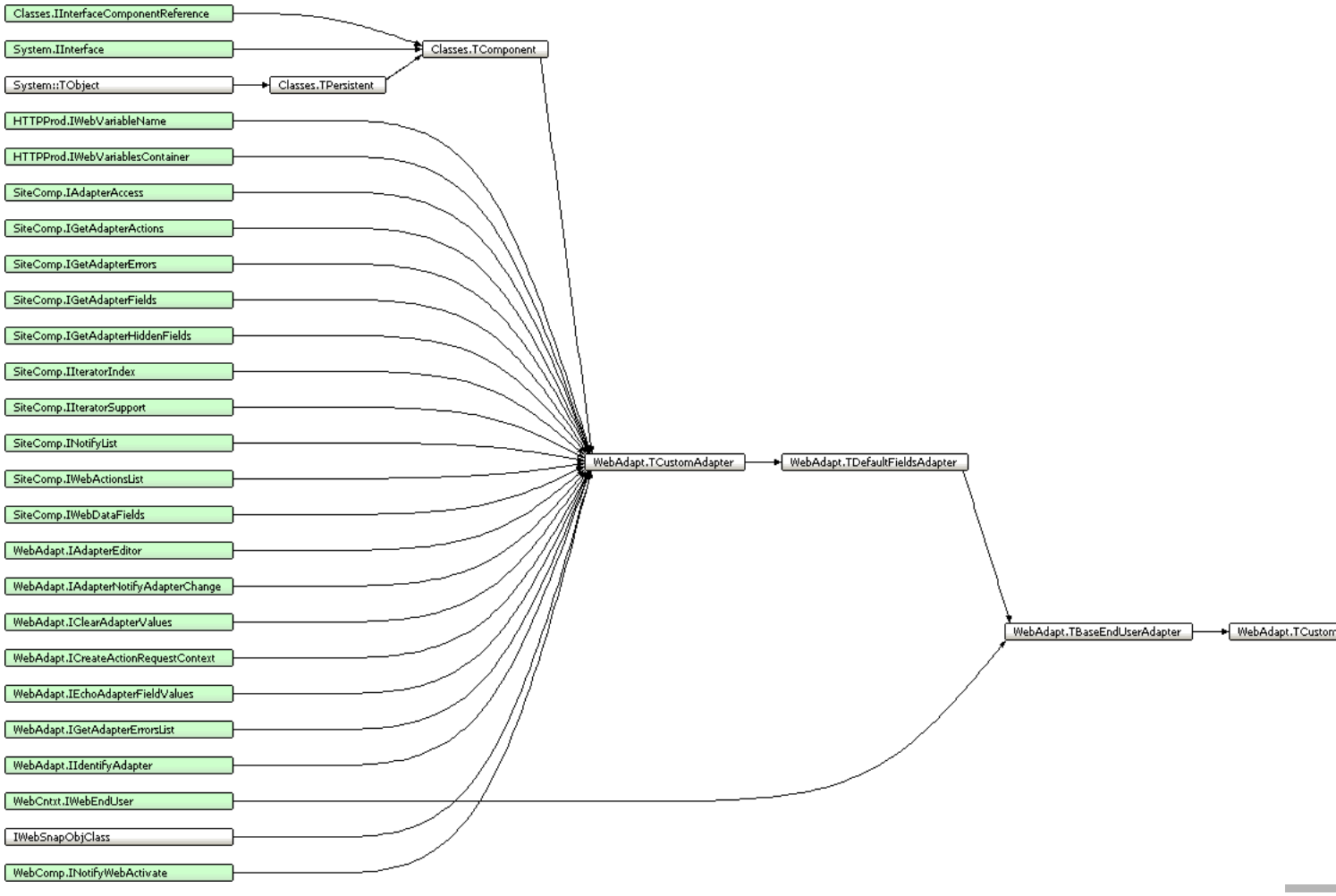

Pascal

```
TCustomEndUserAdapter = class(TBaseEndUserAdapter);
```

C++

```
class TCustomEndUserAdapter : public TBaseEndUserAdapter;
```

File

WebAdapt

Description

TCustomEndUserAdapter is the base class for end user adapters that use event handlers to implement the default adapter fields and actions. It introduces events to provide values for the default end user fields (DisplayName and LoggedIn) and to implement the default end user actions (logging in and logging out).

As with any end user adapter, TCustomEndUserAdapter has a LoginPage (see page 24118) property that lets you specify the page name of the Web page that handles the login process. As an end user adapter, it also handles validation of end user access. When the end user attempts to access a page, the end-user adapter is called to determine whether the end user has rights to access that page.

Do not create instances of TCustomEndUserAdapter. Instead, use a descendant such as the TEndUserAdapter (see page 24421) component, which publishes the events whose handlers you must supply to make this adapter work.

See Also

TEndUserAdapter (see page 24421)

173.1.56.1 TCustomEndUserAdapter Members

The following tables list the members exposed by TCustomEndUserAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
V	CanExecuteLoginForm (see page 24106)	Determines if there is a static login page.
V	CanExecuteLogout (see page 24107)	Determines if the user is logged in.
V	CanProduceLoginPage (see page 24107)	Determines whether there is a static login page or an automatically produced login page.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.

	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoOnHasRights (see page 24107)	Determines whether the user has view rights.
	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
	ExecuteLogin (see page 24321)	Generates an OnLogin (see page 24325) event.
	ExecuteLoginForm (see page 24108)	Creates the HTML login page.
	ExecuteLogout (see page 24322)	Generates an OnLogout (see page 24326) event.
	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetDisplayName (see page 24109)	Retrieves the display name string.
	GetDisplayNameValue (see page 24322)	Generates an OnGetDisplayName (see page 24324) event.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetEndUserAdapter (see page 24109)	Gets the end-user adapter.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetLoggedIn (see page 24110)	Determines if the user is logged in
	GetLoggedInValue (see page 24323)	Generates an OnIsLoggedIn (see page 24324) event.
	GetLoginFormAction (see page 24110)	Gets the action component for the login form.
	GetLogoutAction (see page 24111)	Gets the action component for the logout form.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.

	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetUserID (see page 24323)	Generates an OnGetUserID (see page 24324) event.
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasExecuteRights (see page 24111)	Determines if the user has execute rights for the TBaseEndUserAdapter (see page 24097).
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasModifyRights (see page 24112)	Determines if the user has modify rights for the TBaseEndUserAdapter (see page 24097).
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
	HasViewRights (see page 24112)	Determine if the user has view rights to the TBaseEndUserAdapter (see page 24097).
	ImplCanAddActionClass (see page 24113)	Determines if the given action class can be added.
	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 24113)	Determines if the given action class can be added.
	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 24113)	Lists supported end-user adapter actions.
	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
	ImplGetDisplayName (see page 24114)	Gets the display name for this TBaseEndUserAdapter (see page 24097).
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetEndUserAdapter (see page 24114)	Returns this TBaseEndUserAdapter (see page 24097).
	ImplGetFieldsList (see page 24114)	Obtains the list of all of the fields for the end-user adapter.
	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetLoggedIn (see page 24114)	Checks the value of the LoggedIn field.
	ImplGetLoginFormAction (see page 24115)	Gets the LoginForm action component for this TBaseEndUserAdapter (see page 24097).
	ImplGetLogoutAction (see page 24115)	Gets the Logout action component for this TBaseEndUserAdapter (see page 24097).
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasExecuteRights (see page 24115)	Checks user rights and determines if the user has execute rights.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasModifyRights (see page 24116)	Checks user rights and determines if the user has modify rights.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplHasViewRights (see page 24116)	Checks user rights and determines if the user has view rights.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.

	ImplLogIn (see page 24116)	Processes the login for the given user.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplProduceLoginPage (see page 24117)	Generate the HTML login page.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	LogIn (see page 24117)	Logs in a specified user.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceLoginPage (see page 24117)	Generates the login page.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UserList (see page 24118)	Gets the user list.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 24184)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24185)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24186)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24186)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 24187)	Supplies name/value pairs needed to execute an adapter action.
	OnGetDisplayName (see page 24324)	Occurs when the adapter needs to provide the name of the current end user.
	OnGetUserID (see page 24324)	Occurs when the adapter needs to provide the ID of the current end user.
	OnHasRights (see page 24119)	Determines if the user rights match the rights being queried.
	OnIsLoggedIn (see page 24324)	Occurs when the adapter needs to indicate whether the current end user has logged in.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.
	OnLogin (see page 24325)	Logs in the current end user.
	OnLogout (see page 24326)	Occurs when the current user logs out.
	OnProduceLoginPage (see page 24119)	Allows implementation of a customized login page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetDisplayName (see page 24673)	Returns the value of the DisplayName (see page 24678) property.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEndUserAdapter (see page 24674)	Returns the value of the EndUserAdapter (see page 24678) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
	GetLoggedIn (see page 24674)	Returns the value of the LoggedIn (see page 24678) property.
	GetLoginFormAction (see page 24674)	Returns the value of the LoginFormAction (see page 24679) property.
	GetLogoutAction (see page 24675)	Returns the value of the LogoutAction (see page 24680) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.

	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasExecuteRights (see page 24675)	Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasModifyRights (see page 24676)	Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	HasViewRights (see page 24676)	Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LogIn (see page 24677)	Logs in a specified user.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	ProduceLoginPage (see page 24677)	Generates the Web page for logging in users.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 24181)	Actions is the container for adapter actions components.
R	Actions (see page 24402)	Container for adapter actions components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Data (see page 24181)	Data is the container for adapter field components.
R	Data (see page 24402)	Container for adapter field components.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 24678)	Indicates the name of the end user as it should be used for display in Web pages.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	EndUserAdapter (see page 24678)	Provides access to the end user adapter that implements this interface.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	LoggedIn (see page 24678)	Indicates whether the current user is logged in.
R	LoginFormAction (see page 24679)	Indicates the action component for the login form.
	LoginPage (see page 24118)	Specifies the name of the page that acts as the login page.
R	LogoutAction (see page 24680)	Indicates the action component for logging a user out.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.56.2 TCustomEndUserAdapter Methods

The methods of the TCustomEndUserAdapter class are listed here.

Protected Methods

	Name	Description
V	ExecuteLogin (see page 24321)	Generates an OnLogin (see page 24325) event.
V	ExecuteLogout (see page 24322)	Generates an OnLogout (see page 24326) event.
V	GetDisplayNameValue (see page 24322)	Generates an OnGetDisplayName (see page 24324) event.
V	GetLoggedInValue (see page 24323)	Generates an OnIsLoggedIn (see page 24324) event.
V	GetUserID (see page 24323)	Generates an OnGetUserID (see page 24324) event.

173.1.56.2.1 TCustomEndUserAdapter.ExecuteLogin Method

Generates an OnLogin (see page 24325) event.

Pascal

```
procedure ExecuteLogin(AUserID: Variant); override;
```

C++

```
virtual __fastcall ExecuteLogin(Variant AUserID);
```

Description

ExecuteLogin is the implementation of the Login action. When a server-side script executes the login action, it generates a call to ExecuteLogin to start the login process. TCustomEndUserAdapter (see page 24313) responds by generating an OnLogin (see page 24325) event, where you can take whatever action is necessary to log in the specified user.

AUserID is the user ID of the user to log in.

See Also

OnLogin (see page 24325)

173.1.56.2.2 TCustomEndUserAdapter.ExecuteLogout Method

Generates an OnLogout (see page 24326) event.

Pascal

```
procedure ExecuteLogout; override;
```

C++

```
virtual __fastcall ExecuteLogout();
```

Description

ExecuteLogout is the implementation of the Logout action. When a server-side script executes the logout action, it generates a call to ExecuteLogout. TCustomEndUserAdapter (see page 24313) responds by generating an OnLogout (see page 24326) event, where you can take whatever action is necessary to log the specified user out.

See Also

OnLogout (see page 24326)

173.1.56.2.3 TCustomEndUserAdapter.GetDisplayNameValue Method

Generates an OnGetDisplayName (see page 24324) event.

Pascal

```
function GetDisplayNameValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayNameValue();
```

Description

GetDisplayNameValue is the implementation of the DisplayName adapter field. When a server-side script references the display name of the current user, it generates a call to GetDisplayNameValue. TCustomEndUserAdapter (see page 24313) responds by generating an OnGetDisplayName (see page 24324) event, where you can take supply the name of the current user.

GetDisplayNameValue returns the display name that the OnGetDisplayName (see page 24324) event handler provides. If you do not supply an OnGetDisplayName (see page 24324) event handler, GetDisplayNameValue returns an empty string.

See Also

OnGetDisplayName (see page 24324)

173.1.56.2.4 TCustomEndUserAdapter.GetLoggedInValue Method

Generates an OnIsLoggedIn (see page 24324) event.

Pascal

```
function GetLoggedInValue: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetLoggedInValue();
```

Description

GetLoggedInValue is the implementation of the LoggedIn adapter field. When a server-side script references the LoggedIn field for the current user, it generates a call to GetLoggedInValue. TCustomEndUserAdapter (see page 24313) responds by generating an OnIsLoggedIn (see page 24324) event, where you can indicate whether the current user has logged in.

GetLoggedInValue returns the value that the OnGetLoggedIn event handler provides. If you do not supply an OnGetLoggedIn event handler, GetLoggedInValue returns false.

See Also

OnIsLoggedIn (see page 24324)

173.1.56.2.5 TCustomEndUserAdapter.GetUserID Method

Generates an OnGetUserID (see page 24324) event.

Pascal

```
function GetUserID: Variant; override;
```

C++

```
virtual __fastcall Variant GetUserID();
```

Description

GetUserIDValue is the implementation of the UserID adapter field. When a server-side script references the user ID of the current user, it generates a call to GetUserIDValue. TCustomEndUserAdapter (see page 24313) responds by generating an OnGetUserID (see page 24324) event, where you can take supply the ID of the current user.

GetUserIDValue returns the user ID that the OnGetUserID (see page 24324) event handler provides. If you do not supply an OnGetUserID (see page 24324) event handler, GetUserIDValue returns an unassigned Variant.

See Also

OnGetUserID (see page 24324)

173.1.56.3 TCustomEndUserAdapter Events

The events of the TCustomEndUserAdapter class are listed here.

Public Events

	Name	Description
	OnGetDisplayName (see page 24324)	Occurs when the adapter needs to provide the name of the current end user.
	OnGetUserID (see page 24324)	Occurs when the adapter needs to provide the ID of the current end user.
	OnIsLoggedIn (see page 24324)	Occurs when the adapter needs to indicate whether the current end user has logged in.
	OnLogin (see page 24325)	Logs in the current end user.

	OnLogout (see page 24326)	Occurs when the current user logs out.

173.1.56.3.1 TCustomEndUserAdapter.OnGetDisplayName Event

Occurs when the adapter needs to provide the name of the current end user.

Pascal

```
property OnGetDisplayName: TEndUserGetNameEvent;
```

C++

```
__property TEndUserGetNameEvent OnGetDisplayName;
```

Description

Write an OnGetDisplayName event handler to provide the value of the DisplayName field. When a server-side script (or a caller of the IWebEndUser interface) requests the display name for the current end user, TCustomEndUserAdapter (see page 24313) generates an OnGetDisplayName event to retrieve that value.

Note: If you do not supply an OnGetDisplayName event handler, TCustomEndUserAdapter (see page 24313) always returns an empty string as the display name for the current end user.

See Also

DisplayName (see page 24678)

EndUser (see page 24657)

WebContext (see page 24687)

173.1.56.3.2 TCustomEndUserAdapter.OnGetUserID Event

Occurs when the adapter needs to provide the ID of the current end user.

Pascal

```
property OnGetUserID: TEndUserGetUserIDEvent;
```

C++

```
__property TEndUserGetUserIDEvent OnGetUserID;
```

Description

Write an OnGetDisplayName (see page 24324) event handler to provide the user ID for the current user. When a server-side script requests ID for the current end user, TCustomEndUserAdapter (see page 24313) generates an OnGetUserID event to retrieve that value.

Note: If you do not supply an OnGetUserID event handler, TCustomEndUserAdapter (see page 24313) always returns an unassigned Variant as the user ID.

See Also

GetDisplayNameValue (see page 24322)

TWebUserList (see page 25625)

OnLogin (see page 24325)

173.1.56.3.3 TCustomEndUserAdapter.OnIsLoggedIn Event

Occurs when the adapter needs to indicate whether the current end user has logged in.

Pascal

```
property OnIsLoggedIn: TEndUserIsLoggedInEvent;
```

C++

```
__property TEndUserIsLoggedInEvent OnIsLoggedIn;
```

Description

Write an OnIsLoggedIn event handler to provide the value of the LoggedIn field. When a server-side script (or a caller of the IWebEndUser interface) inquires whether the current user has logged in, TCustomEndUserAdapter (see page 24313) generates an OnIsLoggedIn event to retrieve that information.

Note: If you do not supply an OnIsLoggedIn event handler, TCustomEndUserAdapter (see page 24313) always returns false, indicating that the current end user has not logged in.

See Also

LoggedIn (see page 24678)

EndUser (see page 24657)

WebContext (see page 24687)

OnLogin (see page 24325)

OnLogout (see page 24326)

173.1.56.3.4 TCustomEndUserAdapter.OnLogin Event

Logs in the current end user.

Pascal

```
property OnLogin: TEndUserUserIDEvent;
```

C++

```
__property TEndUserUserIDEvent OnLogin;
```

Description

Write an OnLogin event handler to allow the TCustomEndUserAdapter (see page 24313) object to log users in. When a server-side script (or a caller of the IWebEndUser interface) attempts to log in a user, TCustomEndUserAdapter (see page 24313) generates an OnLogin event to perform whatever steps are involved in the login process.

Note: If you do not supply an OnLogin event handler, TCustomEndUserAdapter (see page 24313) does nothing to log users in.

See Also

LogIn (see page 24677)

EndUser (see page 24657)

WebContext (see page 24687)

OnIsLoggedIn (see page 24324)

OnLogout (see page 24326)

TWebUserList (see page 25625)

OnGetUserID (see page 24324)

173.1.56.3.5 TCustomEndUserAdapter.OnLogout Event

Occurs when the current user logs out.

Pascal

```
property OnLogout: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnLogout;
```

Description

Write an OnLogout event handler to allow the TCustomEndUserAdapter (see page 24313) object to log users out. When a server-side script attempts to log a user out, TCustomEndUserAdapter (see page 24313) generates an OnLogout event to perform whatever steps are involved in the logout process.

Note: If you do not supply an OnLogin (see page 24325) event handler, TCustomEndUserAdapter (see page 24313) does nothing to log users out.

See Also

OnIsLoggedIn (see page 24324)

OnLogin (see page 24325)

173.1.57 TCustomEndUserAdapterAction Class

TCustomEndUserAdapterAction is a base class for adapter actions that work with descendants of TCustomEndUserAdapter (see page 24313).

Class Hierarchy

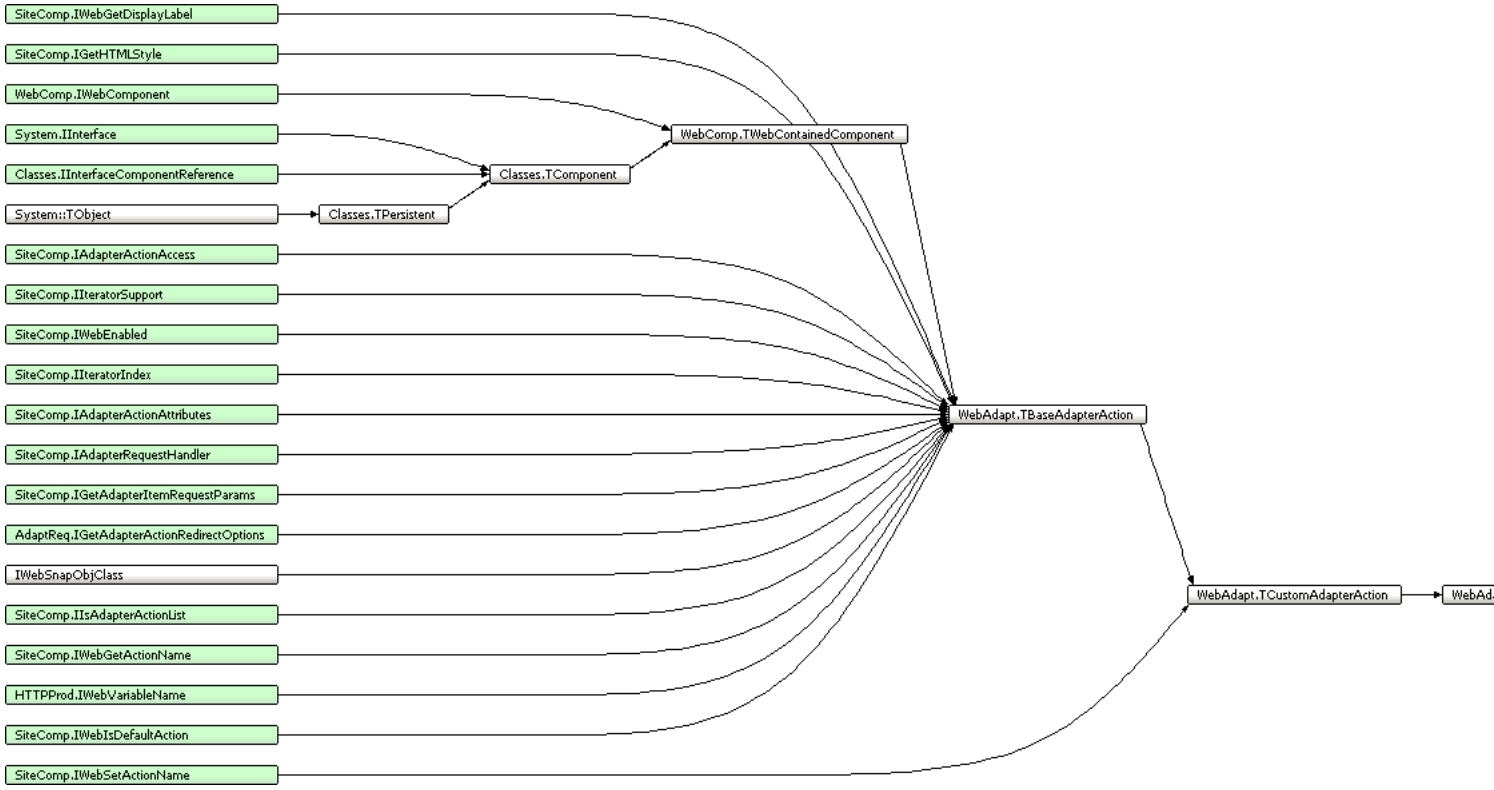

Pascal

```
TCustomEndUserAdapterAction = class(TImplementedAdapterAction);
```

C++

```
class TCustomEndUserAdapterAction : public TImplementedAdapterAction;
```

File

WebAdapt

Description

TCustomEndUserAdapterAction has two purposes. First, it provides a property that descendants use to access the parent adapter. Second, it defines a base class that TCustomEndUserAdapter (see page 24313) checks for when determining whether an adapter action can be added as a child.

173.1.57.1 TCustomEndUserAdapterAction Members

The following tables list the members exposed by TCustomEndUserAdapterAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDefaultActionName (see page 24055)	Gets the default name used to reference this adapter action with server-side script.
	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.

	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 24063)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 24195)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 24069)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 24197)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24333)	Returns the parent adapter of this adapter action.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 24075)	Called after the adapter action has executed.
	OnAfterExecute (see page 24198)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 24075)	Called after the adapter action response is generated.
	OnAfterGetResponse (see page 24199)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24076)	Called before the adapter action is executed.
	OnBeforeExecute (see page 24199)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24076)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 24200)	Called before the adapter action response is generated.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 24077)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.

	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 24073)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 24073)	Defines the rights required to execute this action.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.

	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RedirectOptions (see page 24074)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.57.2 TCustomEndUserAdapterAction Properties

The properties of the TCustomEndUserAdapterAction class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 24333)	Returns the parent adapter of this adapter action.

173.1.57.2.1 TCustomEndUserAdapterAction.Adapter Property

Returns the parent adapter of this adapter action.

Pascal

```
property Adapter: TBaseEndUserAdapter;
```

C++

```
__property TBaseEndUserAdapter Adapter;
```

Description

Descendant classes use this property to access the parent adapter.

See Also

TCustomEndUserAdapter (see page 24313)

173.1.58 TCustomEndUserSessionAdapter Class

TCustomEndUserSessionAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.

Class Hierarchy

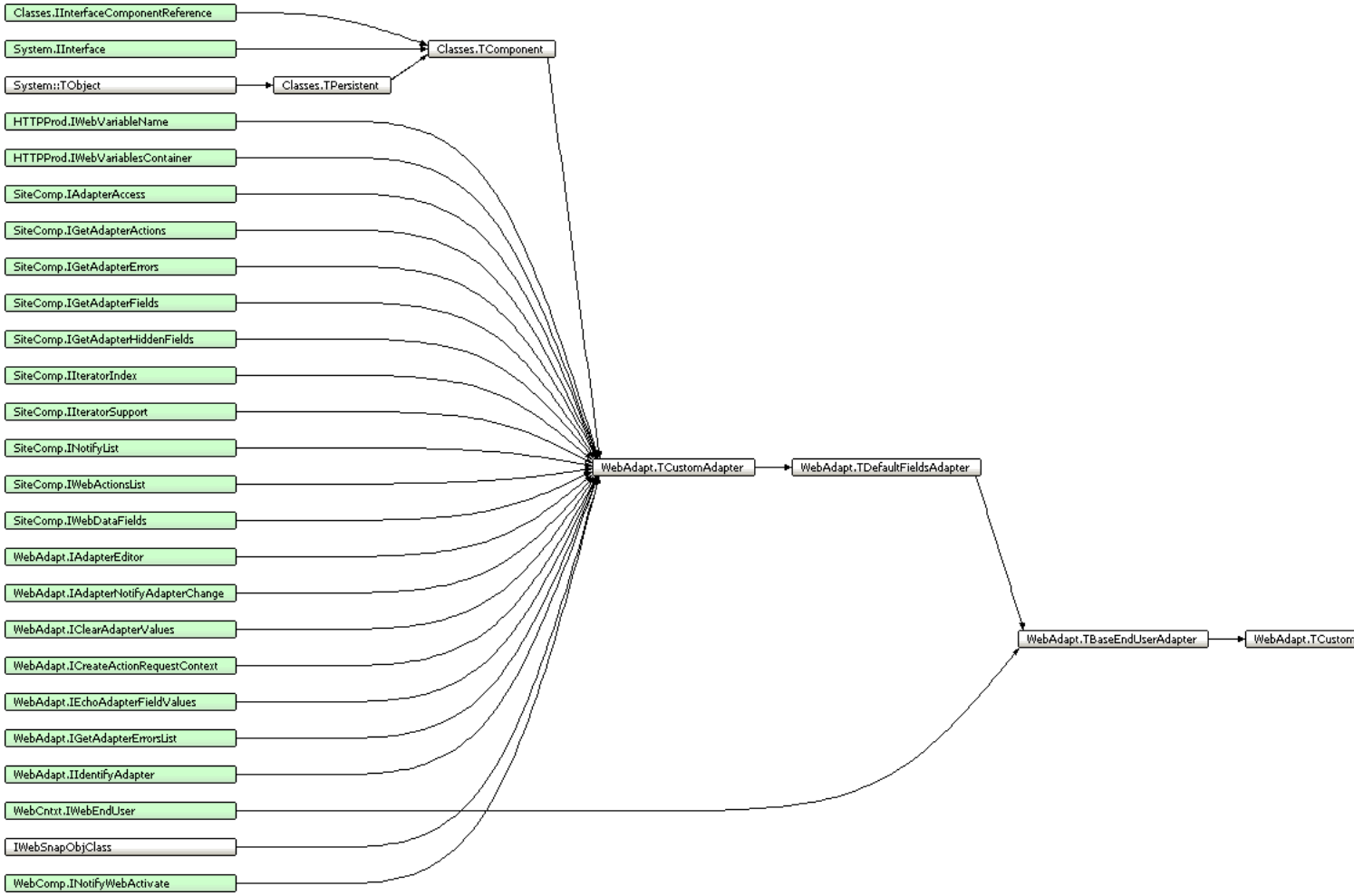

Pascal

```
TCustomEndUserSessionAdapter = class(TBaseEndUserAdapter);
```

C++

```
class TCustomEndUserSessionAdapter : public TBaseEndUserAdapter;
```

File

WebAdapt

Description

TCustomEndUserSessionAdapter is the base class for end user adapters that use a TSessionsService object to implement the default adapter fields and actions. It handles login and logout actions on behalf of the end user, and uses the TSessionsService component to store information about the current end user. The WebSnap application must contain a TSessionsService component (or equivalent) in order for the TCustomEndUserSessionAdapter to work properly.

As with any end user adapter, TCustomEndUserSessionAdapter has a LoginPage (see page 24118) property that lets you specify the page name of the Web page that handles the login process. As an end user adapter, it also handles validation of end user access. When the end user attempts to access a page, the end-user adapter is called to determine whether the end user has rights to access that page.

See Also

TEndUserSessionAdapter (see page 24458)

TSessionsService (see page 25582)

173.1.58.1 TCustomEndUserSessionAdapter Members

The following tables list the members exposed by TCustomEndUserSessionAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
V	CanExecuteLoginForm (see page 24106)	Determines if there is a static login page.
V	CanExecuteLogout (see page 24107)	Determines if the user is logged in.
V	CanProduceLoginPage (see page 24107)	Determines whether there is a static login page or an automatically produced login page.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoOnHasRights (see page 24107)	Determines whether the user has view rights.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
V	ExecuteLogin (see page 24342)	Logs in an end user.
V	ExecuteLoginForm (see page 24108)	Creates the HTML login page.
V	ExecuteLogout (see page 24342)	Logs out the current user.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.

	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetDisplayName (see page 24109)	Retrieves the display name string.
V	GetDisplayNameValue (see page 24343)	Returns the name of the current end user.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetEndUserAdapter (see page 24109)	Gets the end-user adapter.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetLoggedIn (see page 24110)	Determines if the user is logged in
V	GetLoggedInValue (see page 24343)	Indicates whether the current end user has logged in.
	GetLoginFormAction (see page 24110)	Gets the action component for the login form.
	GetLogoutAction (see page 24111)	Gets the action component for the logout form.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	GetUserID (see page 24343)	Returns the user ID of the current user.
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
V	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasExecuteRights (see page 24111)	Determines if the user has execute rights for the TBaseEndUserAdapter (see page 24097).
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasModifyRights (see page 24112)	Determines if the user has modify rights for the TBaseEndUserAdapter (see page 24097).
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.

	HasViewRights (see page 24112)	Determine if the user has view rights to the TBaseEndUserAdapter (see page 24097).
	ImplCanAddActionClass (see page 24113)	Determines if the given action class can be added.
	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 24113)	Determines if the given action class can be added.
	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 24113)	Lists supported end-user adapter actions.
	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
	ImplGetDisplayName (see page 24114)	Gets the display name for this TBaseEndUserAdapter (see page 24097).
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetEndUserAdapter (see page 24114)	Returns this TBaseEndUserAdapter (see page 24097).
	ImplGetFieldsList (see page 24114)	Obtains the list of all of the fields for the end-user adapter.
	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetLoggedIn (see page 24114)	Checks the value of the LoggedIn field.
	ImplGetLoginFormAction (see page 24115)	Gets the LoginForm action component for this TBaseEndUserAdapter (see page 24097).
	ImplGetLogoutAction (see page 24115)	Gets the Logout action component for this TBaseEndUserAdapter (see page 24097).
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasExecuteRights (see page 24115)	Checks user rights and determines if the user has execute rights.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasModifyRights (see page 24116)	Checks user rights and determines if the user has modify rights.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplHasViewRights (see page 24116)	Checks user rights and determines if the user has view rights.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplLogIn (see page 24116)	Processes the login for the given user.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplProduceLoginPage (see page 24117)	Generate the HTML login page.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.

	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	LogIn (see page 24117)	Logs in a specified user.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceLoginPage (see page 24117)	Generates the login page.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UserList (see page 24118)	Gets the user list.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.

R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 24184)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24185)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24186)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24186)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 24187)	Supplies name/value pairs needed to execute an adapter action.
	OnHasRights (see page 24119)	Determines if the user rights match the rights being queried.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.
	OnProduceLoginPage (see page 24119)	Allows implementation of a customized login page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetDisplayName (see page 24673)	Returns the value of the DisplayName (see page 24678) property.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEndUserAdapter (see page 24674)	Returns the value of the EndUserAdapter (see page 24678) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
	GetLoggedIn (see page 24674)	Returns the value of the LoggedIn (see page 24678) property.
	GetLoginFormAction (see page 24674)	Returns the value of the LoginFormAction (see page 24679) property.
	GetLogoutAction (see page 24675)	Returns the value of the LogoutAction (see page 24680) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasExecuteRights (see page 24675)	Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasModifyRights (see page 24676)	Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	HasViewRights (see page 24676)	Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LogIn (see page 24677)	Logs in a specified user.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	ProduceLoginPage (see page 24677)	Generates the Web page for logging in users.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 24181)	Actions is the container for adapter actions components.
R	Actions (see page 24402)	Container for adapter actions components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Data (see page 24181)	Data is the container for adapter field components.
R	Data (see page 24402)	Container for adapter field components.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 24678)	Indicates the name of the end user as it should be used for display in Web pages.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	EndUserAdapter (see page 24678)	Provides access to the end user adapter that implements this interface.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	LoggedIn (see page 24678)	Indicates whether the current user is logged in.
R	LoginFormAction (see page 24679)	Indicates the action component for the login form.
	LoginPage (see page 24118)	Specifies the name of the page that acts as the login page.
R	LogoutAction (see page 24680)	Indicates the action component for logging a user out.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.

R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	UserID (see page 24344)	Specifies the user ID of the current end user.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.58.2 TCustomEndUserSessionAdapter Methods

The methods of the TCustomEndUserSessionAdapter class are listed here.

Protected Methods

	Name	Description
V	ExecuteLogin (see page 24342)	Logs in an end user.
V	ExecuteLogout (see page 24342)	Logs out the current user.
V	GetDisplayNameValue (see page 24343)	Returns the name of the current end user.
V	GetLoggedInValue (see page 24343)	Indicates whether the current end user has logged in.
V	GetUserID (see page 24343)	Returns the user ID of the current user.

173.1.58.2.1 TCustomEndUserSessionAdapter.ExecuteLogin Method

Logs in an end user.

Pascal

```
procedure ExecuteLogin(AUserID: Variant); override;
```

C++

```
virtual __fastcall ExecuteLogin(Variant AUserID);
```

Description

ExecuteLogin is the implementation of the Login action. When a server-side script executes the login action, it generates a call to ExecuteLogin to start the login process.

In TCustomEndUserAdapter (see page 24313), ExecuteLogin assigns the supplied user ID to the UserID (see page 24344) property, which saves the current user ID with the current session.

AUserID is the user ID of the user to log in.

173.1.58.2.2 TCustomEndUserSessionAdapter.ExecuteLogout Method

Logs out the current user.

Pascal

```
procedure ExecuteLogout; override;
```

C++

```
virtual __fastcall ExecuteLogout();
```

Description

ExecuteLogout is the implementation of the Logout action. When a server-side script executes the logout action, it generates a call to ExecuteLogout.

In TCustomEndUserAdapter (see page 24313), ExecuteLogout terminates the current Web session.

173.1.58.2.3 TCustomEndUserSessionAdapter.GetDisplayNameValue Method

Returns the name of the current end user.

Pascal

```
function GetDisplayNameValue: string; override;
```

C++

```
virtual __fastcall AnsiString GetDisplayNameValue();
```

Description

GetDisplayNameValue is the implementation of the DisplayName adapter field. When a server-side script references the display name of the current user, it generates a call to GetDisplayNameValue.

In TCustomEndUserSessionAdapter (see page 24333), GetDisplayNameValue returns the value of the UserID (see page 24344) property, formatted as a string.

173.1.58.2.4 TCustomEndUserSessionAdapter.GetLoggedInValue Method

Indicates whether the current end user has logged in.

Pascal

```
function GetLoggedInValue: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetLoggedInValue();
```

Description

GetLoggedInValue is the implementation of the LoggedIn adapter field. When a server-side script references the LoggedIn field for the current user, it generates a call to GetLoggedInValue.

In TCustomEndUserAdapter (see page 24313), GetLoggedInValue checks whether the UserID (see page 24344) property has been assigned. If UserID (see page 24344) is an assigned Variant other than an empty string, GetLoggedInValue returns true, indicating that the user has logged in. Otherwise, GetLoggedInValue returns false.

173.1.58.2.5 TCustomEndUserSessionAdapter.GetUserID Method

Returns the user ID of the current user.

Pascal

```
function GetUserID: Variant; override;
```

C++

```
virtual __fastcall Variant GetUserID();
```

Description

GetUserIDValue is the implementation of the UserID (see page 24344) adapter field. When a server-side script references the user ID of the current user, it generates a call to GetUserIDValue.

In TCustomEndUserSessionAdapter (see page 24333), GetUserID fetches the value of the UserID (see page 24344) property from the current Web session. This value is set when the user first logs in.

See Also

GetUserID (see page 24323)

ExecuteLogin (see page 24321)

TWebSession (see page 24968)

173.1.58.3 TCustomEndUserSessionAdapter Properties

The properties of the TCustomEndUserSessionAdapter class are listed here.

Public Properties

	Name	Description
	UserID (see page 24344)	Specifies the user ID of the current end user.

173.1.58.3.1 TCustomEndUserSessionAdapter.UserID Property

Specifies the user ID of the current end user.

Pascal

```
property UserID: Variant;
```

C++

```
__property Variant UserID;
```

Description

Use UserID to get or set the user ID of the current user. When a user logs in (that is, when TCustomEndUserSessionAdapter (see page 24333) executes the Login action), TCustomEndUserSessionAdapter (see page 24333) assigns the user's ID as the value of UserID. TCustomEndUserSessionAdapter (see page 24333) stores this value with the current session.

173.1.59 TCustomLoginFormAdapter Class

TCustomLoginFormAdapter is the base class for adapters that manage the fields and actions of a login form.

Class Hierarchy

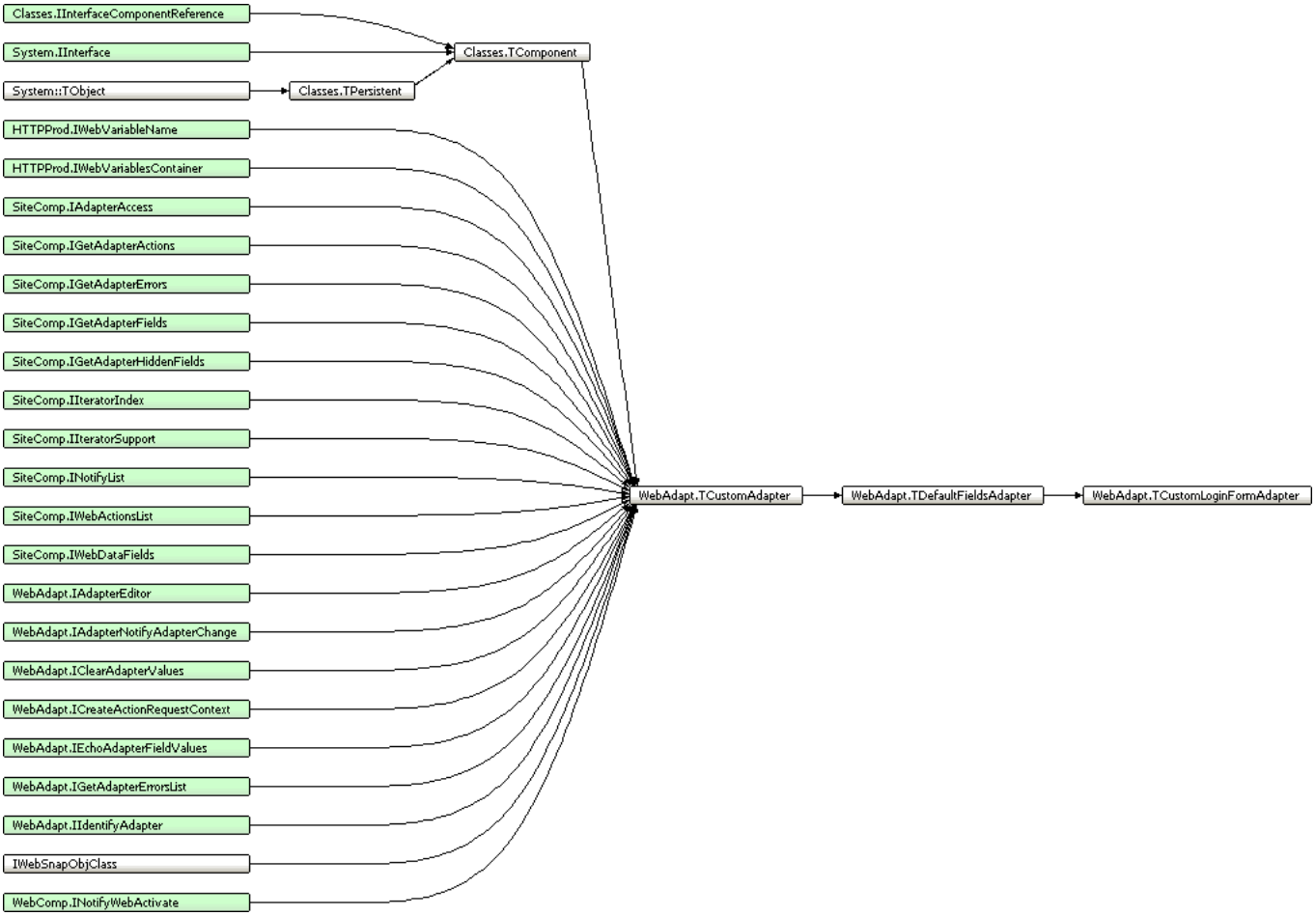

Pascal

```
TCustomLoginFormAdapter = class(TDefaultFieldsAdapter);
```

C++

```
class TCustomLoginFormAdapter : public TDefaultFieldsAdapter;
```

File

WebAdapt

Description

TCustomLoginFormAdapter introduces default fields and actions that support building a login form. The adapter fields include user name (UserName (see page 24360)), password (Password (see page 24360)), and next page (NextPage (see page 24359)). The next page field lists the Web pages in the application. The properties of TCustomLoginFormAdapter reflect the values that are assigned to those fields. TCustomLoginFormAdapter implements a single default action for logging in. If the WebSnap application also contains an end user adapter such as TEndUserAdapter (see page 24421) or TEndUserSessionAdapter (see page 24458), TLoginFormAdapter (see page 24500) calls the end user adapter's Login method to log the user in. As an alternate approach, you can provide an OnLogin (see page 24361) event handler.

Do not create instances of TCustomLoginFormAdapter. It is intended only as a base class. Instead, use a TCustomLoginFormAdapter descendant such as TLoginFormAdapter (see page 24500).

See Also

TEndUserAdapter (see page 24421)

TEndUserSessionAdapter (see page 24458)

TAdapterPageProducer (see page 19802)

TLoginFormAdapter (see page 24500)

173.1.59.1 TCustomLoginFormAdapter Members

The following tables list the members exposed by TCustomLoginFormAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
	EndUser (see page 24353)	Returns the interface of the WebSnap application's end user adapter.
	ExecuteLogin (see page 24353)	Logs in the user after the user name and password have been validated.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.

	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
V	GetNextPage (see page 24354)	Returns the value of the NextPage (see page 24359) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
V	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 24354)	Provides the underlying implementation for the CanAddActionClass (see page 24150) method.
V	ImplCanAddFieldClass (see page 24355)	Provides the underlying implementation for the CanAddFieldClass (see page 24150) method.
V	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
V	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 24355)	Provides the underlying implementation for the GetActionsList (see page 24156) method.
V	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
V	ImplGetFieldsList (see page 24356)	Provides the underlying implementation for the GetFieldsList (see page 24159) method.

	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24356)	Provides the underlying implementation for the NotifyDeactivate (see page 24175) method.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	RaiseBlankPassword (see page 24357)	Raises an EAdapterFieldException (see page 23698) with a message that the supplied password is blank.
	RaiseBlankUserName (see page 24357)	Raises an EAdapterFieldException (see page 23698) with a message that the supplied user name is blank.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property

V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UserList (see page 24357)	Returns the interface of the WebSnap application's Web user list.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidatePassword (see page 24358)	Validates the password that the user supplied to the login form.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateUserName (see page 24358)	Validates the name that the user supplied to the login form.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 24184)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24185)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24186)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24186)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 24187)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.
	OnLogin (see page 24361)	Occurs after the password and user ID are validated.
	OnValidatePassword (see page 24362)	Occurs when the login form adapter must validate the password that the user supplied.
	OnValidateUserName (see page 24362)	Occurs when the login form adapter must validate the name that the user supplied.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.

	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 24181)	Actions is the container for adapter actions components.
R	Actions (see page 24402)	Container for adapter actions components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Data (see page 24181)	Data is the container for adapter field components.
R	Data (see page 24402)	Container for adapter field components.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
	NextPage (see page 24359)	Names the Web page the user should see after successfully logging in.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Password (see page 24360)	Indicates the password that the user supplied to the login form.
	PasswordRequired (see page 24360)	Indicates whether the user must supply a password to log in.
R	UserName (see page 24360)	Indicates the name that the user supplied to the login form.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.59.2 TCustomLoginFormAdapter Methods

The methods of the TCustomLoginFormAdapter class are listed here.

Protected Methods

	Name	Description
	EndUser (see page 24353)	Returns the interface of the WebSnap application's end user adapter.
	ExecuteLogin (see page 24353)	Logs in the user after the user name and password have been validated.
V	GetNextPage (see page 24354)	Returns the value of the NextPage (see page 24359) property.
V	ImplCanAddActionClass (see page 24354)	Provides the underlying implementation for the CanAddActionClass (see page 24150) method.
V	ImplCanAddFieldClass (see page 24355)	Provides the underlying implementation for the CanAddFieldClass (see page 24150) method.
V	ImplGetActionsList (see page 24355)	Provides the underlying implementation for the GetActionsList (see page 24156) method.
V	ImplGetFieldsList (see page 24356)	Provides the underlying implementation for the GetFieldsList (see page 24159) method.
V	ImplNotifyDeactivate (see page 24356)	Provides the underlying implementation for the NotifyDeactivate (see page 24175) method.
	RaiseBlankPassword (see page 24357)	Raises an EAdapterFieldException (see page 23698) with a message that the supplied password is blank.

	RaiseBlankUserName (see page 24357)	Raises an EAdapterFieldException (see page 23698) with a message that the supplied user name is blank.
	UserList (see page 24357)	Returns the interface of the WebSnap application's Web user list.
	ValidatePassword (see page 24358)	Validates the password that the user supplied to the login form.
	ValidateUserName (see page 24358)	Validates the name that the user supplied to the login form.

173.1.59.2.1 TCustomLoginFormAdapter.EndUser Method

Returns the interface of the WebSnap application's end user adapter.

Pascal

```
function EndUser: IWebEndUser;
```

C++

```
__fastcall IWebEndUser EndUser();
```

Description

TCustomLoginFormAdapter (see page 24344) uses EndUser internally to interact with the WebSnap application's end user adapter. For example, the login form adapter calls the Login method on this interface to log in an end user after validating the user name and password.

EndUser returns the interface of an end user adapter such as TEndUserAdapter (see page 24421) or TEndUserSessionAdapter (see page 24458). If the WebSnap application does not include an end user adapter, EndUser returns nil (Delphi) or NULL (C++).

See Also

TEndUserAdapter (see page 24421)

TEndUserSessionAdapter (see page 24458)

LogIn (see page 24677)

173.1.59.2.2 TCustomLoginFormAdapter.ExecuteLogin Method

Logs in the user after the user name and password have been validated.

Pascal

```
procedure ExecuteLogin(AUserID: Variant);
```

C++

```
__fastcall ExecuteLogin(Variant AUserID);
```

Description

The Login action calls ExecuteLogin to log the user in after validating the user name and password. This method first generates an OnLogin (see page 24361) event, so that the OnLogin (see page 24361) event handler can perform any application-specific tasks to log the user in. Then, if the EndUser (see page 24353) property is not nil (Delphi) or NULL (C++), ExecuteLogin calls the end user adapter's Login method, so that the end user adapter can log the user in.

AUserID is the user ID of the end user. If the WebSnap application includes a TWebUserList component, this is the user ID that the Web user list associates with the current user name and password. Otherwise, AUserID is the value of the UserName (see page 24360) property.

See Also

EndUser (see page 24353)

OnLogin (see page 24361)

LogIn (see page 24677)

UserName (see page 24360)

TWebUserList (see page 25625)

173.1.59.2.3 TCustomLoginFormAdapter.GetNextPage Method

Returns the value of the NextPage (see page 24359) property.

Pascal

```
function GetNextPage: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetNextPage();
```

Description

GetNextPage is the getter method for the NextPage (see page 24359) property. See the NextPage (see page 24359) property for more information.

See Also

NextPage (see page 24359)

173.1.59.2.4 TCustomLoginFormAdapter.ImplCanAddActionClass Method

Provides the underlying implementation for the CanAddActionClass (see page 24150) method.

Pascal

```
function ImplCanAddActionClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddActionClass provides the underlying implementation of the CanAddActionClass (see page 24150) method that is introduced in TCustomAdapter (see page 24139). At design time, the Actions (see page 24402) editor calls this method to determine what actions it can add to the login form adapter.

AParent is the adapter that will act as a parent to the added action.

AClass is the class of the proposed action.

ImplCanAddActionClass returns true if the login form adapter can act as a parent to an action of the specified class.

In TCustomLoginFormAdapter (see page 24344), ImplCanAddActionClass returns true if AClass is (or descends from) TLoginFormAdapterLoginAction (see page 24514) (the Login action).

See Also

ImplCanAddFieldClass (see page 24355)

CanAddActionClass (see page 24150)

TLoginFormAdapterLoginAction (see page 24514)

ImplGetActionsList (see page 24355)

173.1.59.2.5 TCustomLoginFormAdapter.ImplCanAddFieldClass Method

Provides the underlying implementation for the CanAddFieldClass (see page 24150) method.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddFieldClass provides the underlying implementation of the CanAddFieldClass (see page 24150) method that is introduced in TCustomAdapter (see page 24139). At design time, the Fields editor calls this method to determine what fields it can add to the login form adapter.

AParent is the adapter that will act as a parent to the added field.

AClass is the class of the proposed field.

ImplCanAddFieldClass returns true if the login form adapter can act as a parent to a field of the specified class.

In TCustomLoginFormAdapter (see page 24344), ImplCanAddFieldClass returns true if AClass is (or descends from) TAdapterPasswordField (see page 23971), TAdapterUserNameField (see page 24024) or TAdapterNextPageField (see page 23957) (the Password (see page 24360), UserName (see page 24360), or NextPage (see page 24359) field).

See Also

ImplCanAddActionClass (see page 24354)

CanAddFieldClass (see page 24150)

TAdapterNextPageField (see page 23957)

TAdapterPasswordField (see page 23971)

TAdapterUserNameField (see page 24024)

ImplGetFieldsList (see page 24356)

173.1.59.2.6 TCustomLoginFormAdapter.ImplGetActionsList Method

Provides the underlying implementation for the GetActionsList (see page 24156) method.

Pascal

```
procedure ImplGetActionsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetActionsList(TStrings AList);
```

Description

ImplGetActionsList provides the underlying implementation of the GetActionsList (see page 24156) method that is introduced in TCustomAdapter (see page 24139). This method fills a string list with the names of all the actions an adapter can support, associating each action name with a class reference to the corresponding action class.

AList is an existing string list instance to fill with the names of the actions that this adapter supports.

In TCustomLoginFormAdapter (see page 24344), ImplGetActionsList clears the list of any strings it may contain on entry, and adds a single string, 'Login', which is the name of the login action. It associates this string with TLoginFormAdapterLoginAction (see page 24514).

See Also

ImplGetFieldsList (see page 24356)

GetActionsList (see page 24156)

TLoginFormAdapterLoginAction (see page 24514)

173.1.59.2.7 TCustomLoginFormAdapter.ImplGetFieldsList Method

Provides the underlying implementation for the GetFieldsList (see page 24159) method.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

ImplGetFieldsList provides the underlying implementation of the GetFieldsList (see page 24159) method that is introduced in TCustomAdapter (see page 24139). This method fills a string list with the names of all the fields an adapter can support, associating each field name with a class reference to the corresponding field class.

AList is an existing string list instance to fill with the names of the fields that this adapter supports.

In TCustomLoginFormAdapter (see page 24344), ImplGetFieldsList clears the list of any strings it may contain on entry, and adds the strings, 'UserName (see page 24360)', 'Password (see page 24360)', and 'NextPage (see page 24359)', which it associates with TAdapterUserNameField (see page 24024), TAdapterPasswordField (see page 23971), and TAdapterNextPageField (see page 23957), respectively.

See Also

ImplGetActionsList (see page 24355)

GetFieldsList (see page 24159)

TAdapterNextPageField (see page 23957)

TAdapterPasswordField (see page 23971)

TAdapterUserNameField (see page 24024)

173.1.59.2.8 TCustomLoginFormAdapter.ImplNotifyDeactivate Method

Provides the underlying implementation for the NotifyDeactivate (see page 24175) method.

Pascal

```
procedure ImplNotifyDeactivate; override;
```

C++

```
virtual __fastcall ImplNotifyDeactivate();
```

Description

ImplNotifyDeactivate provides the underlying implementation of the NotifyDeactivate (see page 24175) method that is introduced in TCustomAdapter (see page 24139). This method is called when the WebSnap application deactivates the Web module that contains the adapter after processing an HTTP request message.

In TCustomLoginFormAdapter (see page 24344), ImplNotifyDeactivate clears the NextPage (see page 24359) property so

that the next time a request message arrives from the login form, it can be redirected to an appropriate next page.

See Also

NotifyDeactivate (see page 24175)

NextPage (see page 24359)

173.1.59.2.9 TCustomLoginFormAdapter.RaiseBlankPassword Method

Raises an EAdapterFieldException (see page 23698) with a message that the supplied password is blank.

Pascal

```
procedure RaiseBlankPassword;
```

C++

```
__fastcall RaiseBlankPassword();
```

Description

TCustomLoginFormAdapter (see page 24344) calls RaiseBlankPassword internally when it is trying to validate a password that is an empty string and the PasswordRequired (see page 24360) property is true.

See Also

PasswordRequired (see page 24360)

EAdapterFieldException (see page 23698)

ValidatePassword (see page 24358)

173.1.59.2.10 TCustomLoginFormAdapter.RaiseBlankUserName Method

Raises an EAdapterFieldException (see page 23698) with a message that the supplied user name is blank.

Pascal

```
procedure RaiseBlankUserName;
```

C++

```
__fastcall RaiseBlankUserName();
```

Description

TCustomLoginFormAdapter (see page 24344) calls RaiseBlankUserName internally when it is trying to validate a user name that is an empty string.

See Also

EAdapterFieldException (see page 23698)

ValidateUserName (see page 24358)

173.1.59.2.11 TCustomLoginFormAdapter.UserList Method

Returns the interface of the WebSnap application's Web user list.

Pascal

```
function UserList: IWebUserList;
```

C++

```
__fastcall IWebUserList UserList();
```

Description

TCustomLoginFormAdapter (see page 24344) uses UserList internally to interact with the WebSnap application's Web user list. For example, the login form adapter calls the ValidateUser method on this interface when validating the user name and password.

UserList returns the interface of a Web user list such as TWebUserList. If the WebSnap application does not include a Web user list, UserList returns nil (Delphi) or NULL (C++).

See Also

TWebUserList (see page 25625)

ValidateUser (see page 19785)

173.1.59.2.12 TCustomLoginFormAdapter.ValidatePassword Method

Validates the password that the user supplied to the login form.

Pascal

```
procedure ValidatePassword;
```

C++

```
__fastcall ValidatePassword();
```

Description

The Login action associated with this adapter calls ValidatePassword to validate the password that the user supplies. ValidatePassword first generates an OnValidatePassword (see page 24362) event. If the event handler does not exist, or does not handle the validation, ValidatePassword checks whether the password is an empty string. If so, and if the PasswordRequired (see page 24360) property is true, ValidatePassword raises an EAdapterFieldException (see page 23698) exception.

See Also

OnValidatePassword (see page 24362)

PasswordRequired (see page 24360)

EAdapterFieldException (see page 23698)

Password (see page 24360)

173.1.59.2.13 TCustomLoginFormAdapter.ValidateUserName Method

Validates the name that the user supplied to the login form.

Pascal

```
procedure ValidateUserName;
```

C++

```
__fastcall ValidateUserName();
```

Description

The Login action associated with this adapter calls ValidateUserName to validate the name that the user supplies. ValidateUserName first generates an OnValidateUserName (see page 24362) event. If the event handler does not exist, or does not handle the validation, ValidateUserName checks whether the user name is an empty string. If so, it raises an

EAdapterFieldException (see page 23698) exception.

See Also

OnValidateUserName (see page 24362)

EAdapterFieldException (see page 23698)

UserName (see page 24360)

173.1.59.3 TCustomLoginFormAdapter Properties

The properties of the TCustomLoginFormAdapter class are listed here.

Public Properties

	Name	Description
	NextPage (see page 24359)	Names the Web page the user should see after successfully logging in.
R	Password (see page 24360)	Indicates the password that the user supplied to the login form.
	PasswordRequired (see page 24360)	Indicates whether the user must supply a password to log in.
R	UserName (see page 24360)	Indicates the name that the user supplied to the login form.

173.1.59.3.1 TCustomLoginFormAdapter.NextPage Property

Names the Web page the user should see after successfully logging in.

Pascal

```
property NextPage: string;
```

C++

```
__property AnsiString NextPage;
```

Description

NextPage specifies the page that the WebSnap application presents to end users after they successfully log in.

The NextPage adapter field supplies a value for the NextPage property. This is a page that the user selects using the login form. If NextPage is unassigned (that is, the if the login form does not contain a NextPage field or the user does not select a next page), TCustomLoginFormAdapter (see page 24344) checks whether the request parameters include a "success page", and if so, uses that. If there is no "success page" parameter, TCustomLoginFormAdapter (see page 24344) checks the Web context to see if there is a currently requested page the user was trying to reach before being diverted to the login page. If there is no "success page" parameter and no currently requested page, TCustomLoginFormAdapter (see page 24344) uses the application's default page.

See Also

DefaultPage (see page 24900)

RequestedPageName (see page 24658)

TAdapterPageProducer (see page 19802)

TLoginFormAdapter (see page 24500)

SuccessPage (see page 19677)

GetNextPage (see page 24354)

173.1.59.3.2 TCustomLoginFormAdapter.Password Property

Indicates the password that the user supplied to the login form.

Pascal

```
property Password: Variant;
```

C++

```
__property Variant Password;
```

Description

Read Password to obtain the password that the user supplied to the login form. The login form adapter reads this value from the current request message.

See Also

PasswordRequired (see page 24360)

OnValidatePassword (see page 24362)

173.1.59.3.3 TCustomLoginFormAdapter.PasswordRequired Property

Indicates whether the user must supply a password to log in.

Pascal

```
property PasswordRequired: Boolean;
```

C++

```
__property Boolean PasswordRequired;
```

Description

Set PasswordRequired to indicate whether users must supply a password to log in. If PasswordRequired is true and the OnValidatePassword (see page 24362) event handler does not handle the validation of the password, then the login form adapter checks that the value of Password (see page 24360) is not an empty string. If Password (see page 24360) is an empty string, the login form adapter raises an EAdapterFieldException (see page 23698) exception when it tries to validate the password. If the OnValidatePassword (see page 24362) event handler takes care of validating the password, PasswordRequired is ignored.

Note: The login form adapter validates the password (and hence checks PasswordRequired) when the WebSnap application executes this adapter's Login action.

See Also

Password (see page 24360)

OnValidatePassword (see page 24362)

TLoginFormAdapterLoginAction (see page 24514)

EAdapterFieldException (see page 23698)

173.1.59.3.4 TCustomLoginFormAdapter.UserName Property

Indicates the name that the user supplied to the login form.

Pascal

```
property UserName: Variant;
```

C++

```
__property Variant UserName;
```

Description

Read UserName to obtain the name that the user supplied when logging in. The login form adapter reads this value from the current request message.

See Also

Password (see page 24360)

OnValidateUserName (see page 24362)

173.1.59.4 TCustomLoginFormAdapter Events

The events of the TCustomLoginFormAdapter class are listed here.

Public Events

	Name	Description
	OnLogin (see page 24361)	Occurs after the password and user ID are validated.
	OnValidatePassword (see page 24362)	Occurs when the login form adapter must validate the password that the user supplied.
	OnValidateUserName (see page 24362)	Occurs when the login form adapter must validate the name that the user supplied.

173.1.59.4.1 TCustomLoginFormAdapter.OnLogin Event

Occurs after the password and user ID are validated.

Pascal

```
property OnLogin: TLoginUserIDEvent;
```

C++

```
__property TLoginUserIDEvent OnLogin;
```

Description

Write an OnLogin event handler to respond when a user has successfully logged in. This event allows your application to record the user login in an application-specific manner.

When the login form adapter's Login action executes, the adapter first receives an OnValidatePassword (see page 24362) and an OnValidateUserName (see page 24362) event. After validating the password and user name, the login form adapter receives an OnLogin event so that it can log the user in. After the OnLogin event, if the WebSnap application includes an end user adapter, the login form adapter calls the end user adapter's Login method. Thus, it may not be necessary to write an OnLogin event handler if the application includes an end user adapter.

See Also

OnValidatePassword (see page 24362)

OnValidateUserName (see page 24362)

LogIn (see page 24677)

TWebUserList (see page 25625)

UserName (see page 24360)

173.1.59.4.2 TCustomLoginFormAdapter.OnValidatePassword Event

Occurs when the login form adapter must validate the password that the user supplied.

Pascal

```
property OnValidatePassword: TValidateAdapterFieldEvent;
```

C++

```
__property TValidateAdapterFieldEvent OnValidatePassword;
```

Description

Write an OnValidatePassword event handler to validate the password that the user supplies to the login form. This event is especially important if the WebSnap application does not include a TWebUserList component for checking passwords against user names. The supplied Password (see page 24360) is the value of the Value parameter on the event handler. You can check this against the user's name by reading the UserName (see page 24360) property.

When the login form adapter's Login action executes, the adapter first receives an OnValidatePassword event to check the password. If the event handler sets the Handled parameter to true, indicating that the password is valid, the adapter goes on to validate the user name using an OnValidateUserName (see page 24362) event, and then to log the user in using an OnLogin (see page 24361) event (and, if the application includes an end user adapter, by calling the end user adapter's Login method). If the event handler finds the password to be invalid, it should raise an exception. If the event handler does not validate the supplied password (that is, if it sets the Handled parameter to false), the login form adapter validates the password by checking whether it is non-blank if the PasswordRequired (see page 24360) property is true.

See Also

OnLogin (see page 24361)

OnValidateUserName (see page 24362)

Password (see page 24360)

UserName (see page 24360)

PasswordRequired (see page 24360)

173.1.59.4.3 TCustomLoginFormAdapter.OnValidateUserName Event

Occurs when the login form adapter must validate the name that the user supplied.

Pascal

```
property OnValidateUserName: TValidateAdapterFieldEvent;
```

C++

```
__property TValidateAdapterFieldEvent OnValidateUserName;
```

Description

Write an OnValidateUserName event handler to validate the name that the user supplies when logging in. This event occurs after the OnValidatePassword (see page 24362) event and before the OnLogin (see page 24361) event.

If the event handler sets the Handled parameter to true, the login form adapter assumes that the user name (which is the value of the Value parameter and of the UserName (see page 24360) property) is valid. In this case, the adapter checks whether the WebSnap application includes a TWebUserList component, and if so, checks the user name against the password. If the event handler sets the Handled parameter to false, the adapter validates the user name by checking whether it is nonempty. In this case, the adapter raises an EAdapterFieldException (see page 23698) exception if the value of UserName (see page 24360) is an empty string. If the OnValidateUserName event handler finds the user name to be invalid, it should raise an

exception.

See Also

OnLogin (see page 24361)

OnValidatePassword (see page 24362)

UserName (see page 24360)

EAdapterFieldException (see page 23698)

173.1.60 TCustomLoginFormAdapterAction Class

TCustomEndUserAdapterAction (see page 24326) is a base class for adapter actions that work with descendants of TCustomLoginFormAdapter (see page 24344).

Class Hierarchy

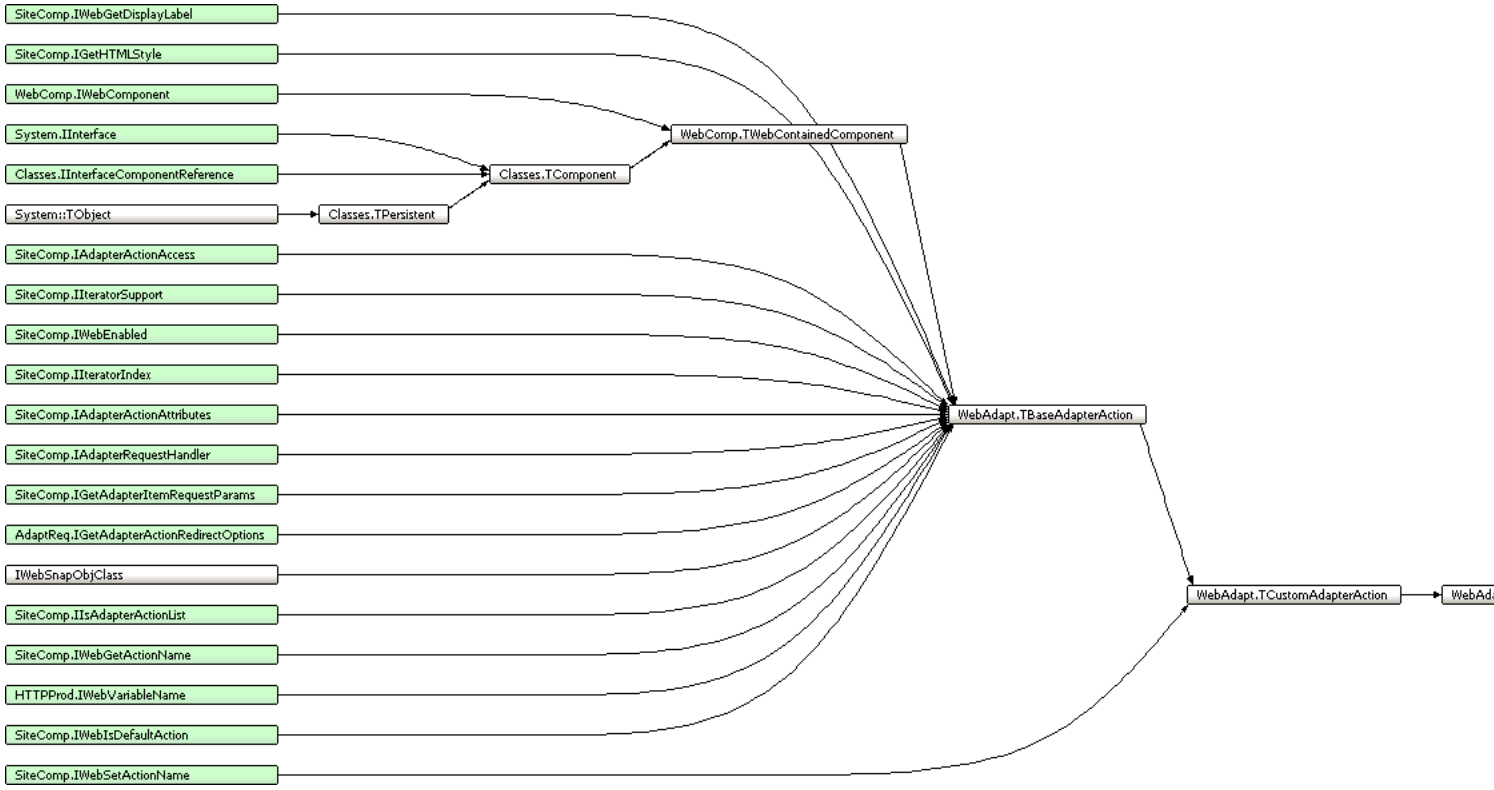

Pascal

```
TCustomLoginFormAdapterAction = class(TImplementedAdapterAction);
```

C++

```
class TCustomLoginFormAdapterAction : public TImplementedAdapterAction;
```

File

WebAdapt

Description

TCustomLoginFormAdapterAction has two purposes. First, it implements a property that descendants use to access the parent adapter. Second, it defines a base class that TCustomLoginFormAdapter (see page 24344) checks for when determining whether an adapter action can be added as a child.

See Also

TImplementedAdapterAction (see page 24491)

TCustomLoginFormAdapter (see page 24344)

173.1.60.1 TCustomLoginFormAdapterAction Members

The following tables list the members exposed by TCustomLoginFormAdapterAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 24055)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.

	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 24063)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 24195)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 24069)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 24197)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24369)	Returns the adapter that acts as a parent to this action.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 24075)	Called after the adapter action has executed.
	OnAfterExecute (see page 24198)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 24075)	Called after the adapter action response is generated.

	OnAfterGetResponse (see page 24199)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24076)	Called before the adapter action is executed.
	OnBeforeExecute (see page 24199)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24076)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 24200)	Called before the adapter action response is generated.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 24077)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.

	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 24073)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 24073)	Defines the rights required to execute this action.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RedirectOptions (see page 24074)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.60.2 TCustomLoginFormAdapterAction Properties

The properties of the TCustomLoginFormAdapterAction class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 24369)	Returns the adapter that acts as a parent to this action.

173.1.60.2.1 TCustomLoginFormAdapterAction.Adapter Property

Returns the adapter that acts as a parent to this action.

Pascal

```
property Adapter: TCustomLoginFormAdapter;
```

C++

```
__property TCustomLoginFormAdapter Adapter;
```

Description

Descendant classes use this property to access the parent adapter.

173.1.61 TCustomPagedAdapter Class

TCustomPagedAdapter is a base class for adapters that support paging.

Class Hierarchy

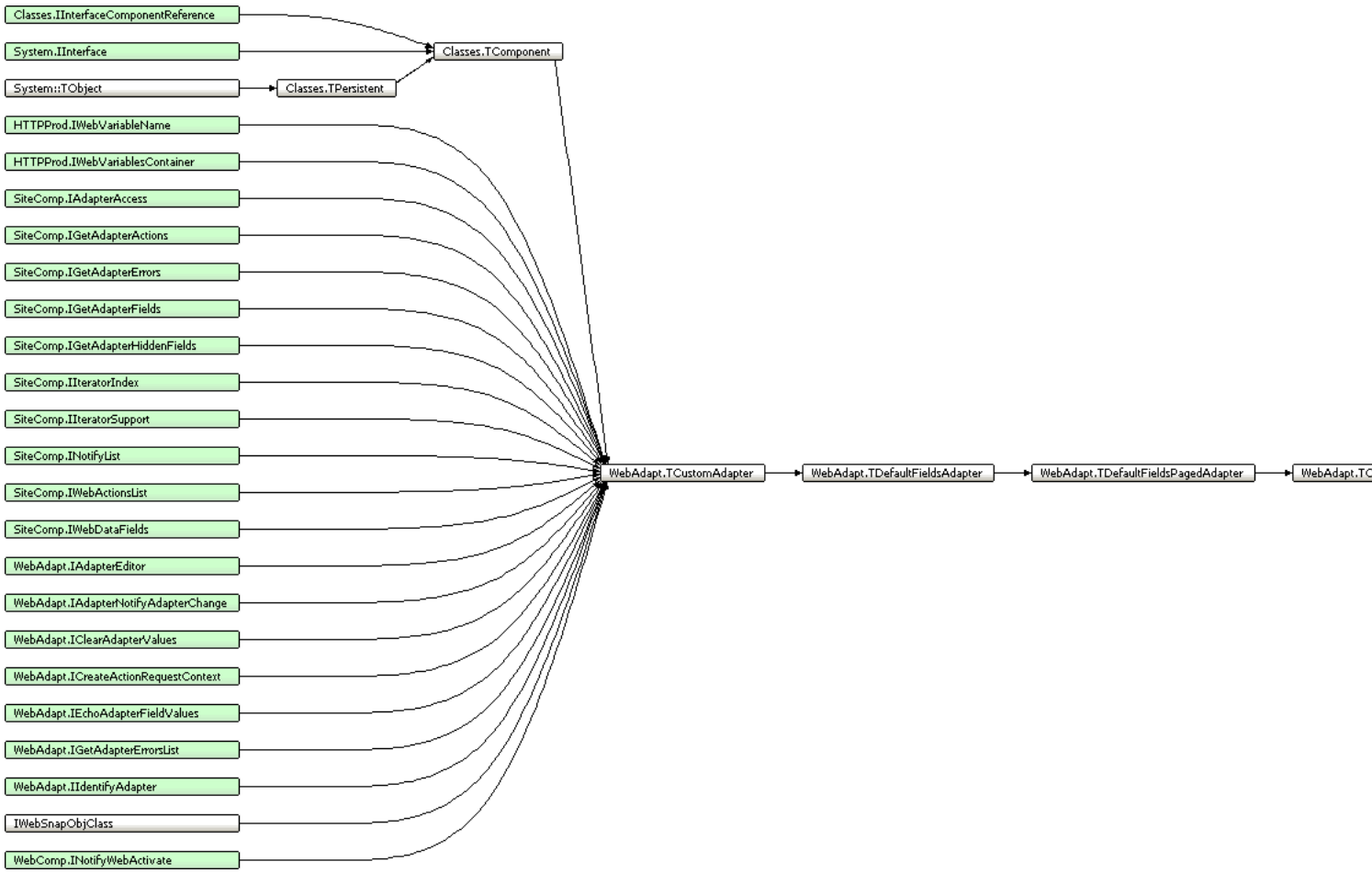

Pascal

```
TCustomPagedAdapter = class(TDefaultFieldsPagedAdapter);
```

C++

```
class TCustomPagedAdapter : public TDefaultFieldsPagedAdapter;
```

File

WebAdapt

Description

Derive from TCustomPagedAdapter to define a new adapter class that support paging. Paging is enabled when the PageSize (see page 24419) property is set.

See Also

TPagedAdapter (see page 24525)

173.1.62 TCustomPagedAdapterAction Class

TCustomPagedAdapterAction is a base class for adapter actions that work with descendants of TDefaultFieldsPagedAdapter (see page 24403).

Class Hierarchy

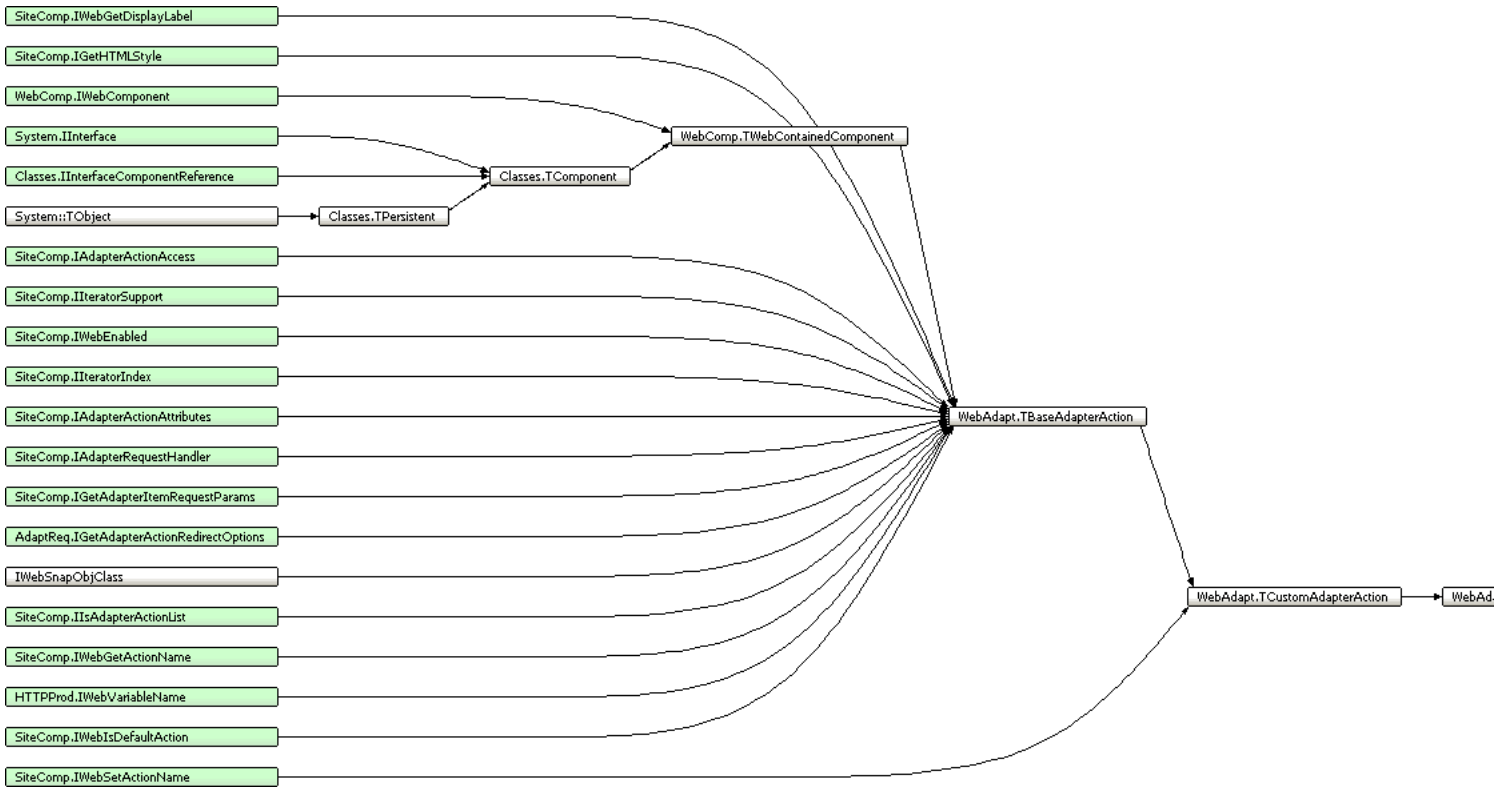

Pascal

```
TCustomPagedAdapterAction = class(TImplementedAdapterAction);
```

C++

```
class TCustomPagedAdapterAction : public TImplementedAdapterAction;
```

File

WebAdapt

Description

TCustomPagedAdapterAction has two purposes. First, it provides a property that is used by descendants to access the parent

adapter. Second, it defines a base class that TDefaultFieldsPagedAdapter (see page 24403) checks for when determining whether an adapter action can be added as a child.

See Also

TImplementedAdapterAction (see page 24491)

TDefaultFieldsPagedAdapter (see page 24403)

TDefaultFieldsPagedAdapter (see page 24403)

173.1.62.1 TCustomPagedAdapterAction Members

The following tables list the members exposed by TCustomPagedAdapterAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 24377)	Returns the parent adapter of this adapter action.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 24055)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.

	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 24063)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 24195)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 24069)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 24197)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24378)	Returns the parent adapter of this adapter action.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 24075)	Called after the adapter action has executed.
	OnAfterExecute (see page 24198)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 24075)	Called after the adapter action response is generated.

	OnAfterGetResponse (see page 24199)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24076)	Called before the adapter action is executed.
	OnBeforeExecute (see page 24199)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24076)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 24200)	Called before the adapter action response is generated.
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 24077)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.

	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayLabel (see page 24073)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 24073)	Defines the rights required to execute this action.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RedirectOptions (see page 24074)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.62.2 TCustomPagedAdapterAction Methods

The methods of the TCustomPagedAdapterAction class are listed here.

Protected Methods

	Name	Description
	GetAdapter (see page 24377)	Returns the parent adapter of this adapter action.

173.1.62.2.1 TCustomPagedAdapterAction.GetAdapter Method

Returns the parent adapter of this adapter action.

Pascal

```
function GetAdapter: TDefaultFieldsPagedAdapter;
```

C++

```
__fastcall TDefaultFieldsPagedAdapter GetAdapter();
```

Description

GetAdapter implements the Adapter (see page 24378) property.

See Also

TDefaultFieldsPagedAdapter (see page 24403)

Adapter (see page 24378)

173.1.62.3 TCustomPagedAdapterAction Properties

The properties of the TCustomPagedAdapterAction class are listed here.

Protected Properties

	Name	Description
	Adapter (see page 24378)	Returns the parent adapter of this adapter action.

173.1.62.3.1 TCustomPagedAdapterAction.Adapter Property

Returns the parent adapter of this adapter action.

Pascal

```
property Adapter: TDefaultFieldsPagedAdapter;
```

C++

```
__property TDefaultFieldsPagedAdapter Adapter;
```

Description

Descendant classes use this property to access the parent adapter.

See Also

GetAdapter (see page 24377)

173.1.63 TCustomStringsValuesList Class

TCustomStringsValuesList is a base class for a values list that defines it's name/value pairs using a string list.

Class Hierarchy

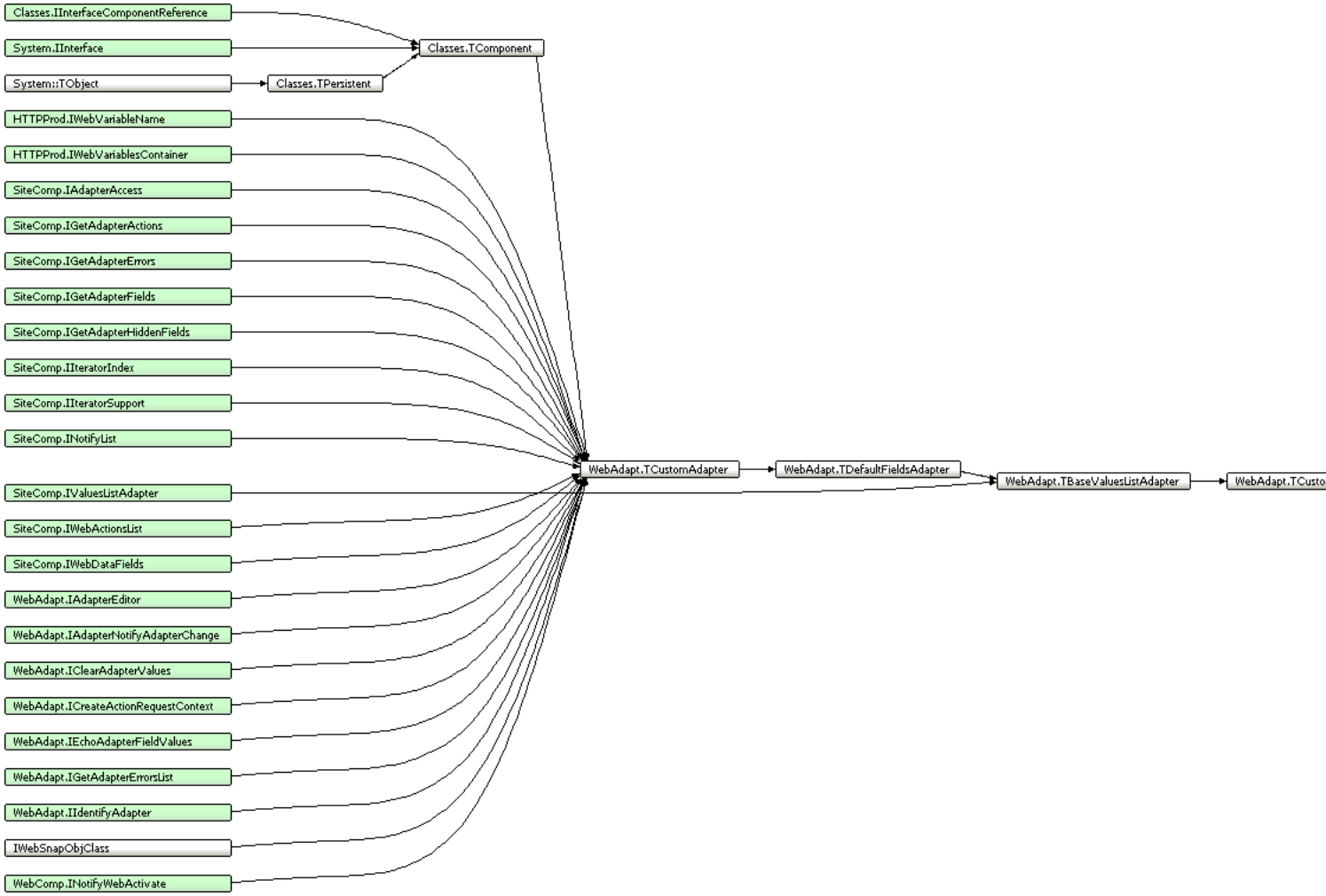

Pascal

```
TCustomStringsValuesList = class(TBaseValuesListAdapter);
```

C++

```
class TCustomStringsValuesList : public TBaseValuesListAdapter;
```

File

WebAdapt

Description

Use descendants of TCustomStringsValuesList with adapter fields to define a list of name/value pairs that are possible values for the adapter field. Adapter fields reference a values list adapter using the adapter field's ValuesList property.

See Also

TBaseValuesListAdapter (see page 24120)

TStringsValuesList (see page 24540)

TAdapterUpdateField (see page 24007)

173.1.63.1 TCustomStringsValuesList Members

The following tables list the members exposed by TCustomStringsValuesList.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.

	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetListImage (see page 24128)	Returns an image associated with the current name/value pair.
	GetListImageOfValue (see page 24128)	Looks up an image for a value.
	GetListName (see page 24128)	Responsible returning the name of the name/value pair.
	GetListNameOfValue (see page 24128)	Looks up a name for a value.
	GetListValue (see page 24129)	Responsible for returning the value of the name/value pair.
	GetNameFieldClass (see page 24387)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValueFieldClass (see page 24387)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 24130)	Returns the class type for this adapter field.
	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 24130)	Indicates whether a type of adapter field can be added to this adapter.
	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
	ImplEndIterator (see page 24388)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the records of the adapter.
	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 24130)	Adds default field components to this adapter.
	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetListImage (see page 24131)	Responsible returning an image associated with the current name/value pair.

	ImplGetListImageOfValue (see page 24131)	Looks up an image for a value.
	ImplGetListName (see page 24388)	Responsible returning the name of the name/value pair.
	ImplGetListNameOfValue (see page 24388)	Looks up a name for a value.
	ImplGetListValue (see page 24389)	Responsible for returning the value of the name/value pair.
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 24389)	Implements IteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24389)	Clears the iteration index.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 24390)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	PrepareStrings (see page 24390)	Customizes the content of the strings list.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property

V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
	StringsPrepared (see page 24391)	This flag indicates whether the strings list has been prepared or customized.
R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 24184)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24185)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24186)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24186)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 24187)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.
	OnPrepareStrings (see page 24391)	Implement OnPrepareStrings to customize the content of the strings list.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 24386)	Create is the constructor for TCustomStringsValuesList (see page 24378).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 24386)	Destroy is the destructor for TCustomStringsList.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
	GetListImage (see page 19741)	Returns the image object associated with the current name/value pair.
	GetListImageOfValue (see page 19742)	Returns the image object associated with a specified value.
	GetListName (see page 19742)	Returns the name part of the current name/value pair.
	GetListNameOfValue (see page 19742)	Returns the name associated with a specified value.
	GetListValue (see page 19743)	Returns the value of the current name/value pair.

V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 24181)	Actions is the container for adapter actions components.
R	Actions (see page 24402)	Container for adapter actions components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.

R	Data (see page 24181)	Data is the container for adapter field components.
R	Data (see page 24402)	Container for adapter field components.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Strings (see page 24390)	This string list defines the list of name/value pairs.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.63.2 TCustomStringsValuesList.Create Constructor

Create is the constructor for TCustomStringsValuesList (see page 24378).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomStringsValuesList(TComponent * AOwner);
```

Description

Create creates the strings list.

See Also

Strings (see page 24390)

173.1.63.3 TCustomStringsValuesList.Destroy Destructor

Destroy is the destructor for TCustomStringsList.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomStringsValuesList();
```

Description

Destroy frees the strings list.

See Also

Strings (see page 24390)

173.1.63.4 TCustomStringsValuesList Methods

The methods of the TCustomStringsValuesList class are listed here.

Protected Methods

	Name	Description
	GetNameFieldClass (see page 24387)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
	GetValueFieldClass (see page 24387)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	ImplEndIterator (see page 24388)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the records of the adapter.
	ImplGetListName (see page 24388)	Responsible returning the name of the name/value pair.
	ImplGetListNameOfValue (see page 24388)	Looks up a name for a value.
	ImplGetListValue (see page 24389)	Responsible for returning the value of the name/value pair.
	ImplNextIteration (see page 24389)	Implements IteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyDeactivate (see page 24389)	Clears the iteration index.
	ImplStartIterator (see page 24390)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	PrepareStrings (see page 24390)	Customizes the content of the strings list.

173.1.63.4.1 TCustomStringsValuesList.GetNameFieldClass Method

Returns the class of adapter field that is responsible for providing the value of the name/value pair.

Pascal

```
function GetNameFieldClass: TComponentClass; override;
```

C++

```
virtual __fastcall TComponentClass GetNameFieldClass();
```

Description

GetNameFieldClass returns a TValuesListNameField (see page 24552).

See Also

GetValueFieldClass (see page 24387)

TValuesListNameField (see page 24552)

173.1.63.4.2 TCustomStringsValuesList.GetValueFieldClass Method

Returns the class of adapter field that is responsible for providing the name of the name/value pair.

Pascal

```
function GetValueFieldClass: TComponentClass; override;
```

C++

```
virtual __fastcall TComponentClass GetValueFieldClass();
```

Description

GetValueFieldClass returns TValuesListValueField (see page 24558).

See Also

GetNameFieldClass (see page 24387)

TValuesListValueField (see page 24558)

173.1.63.4.3 TCustomStringsValuesList.ImplEndIterator Method

Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the records of the adapter.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer); override;
```

C++

```
virtual __fastcall ImplEndIterator(void * OwnerData);
```

Description

Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the records of the adapter.

See Also

enditerator (see page 19723)

173.1.63.4.4 TCustomStringsValuesList.ImplGetListName Method

Responsible returning the name of the name/value pair.

Pascal

```
function ImplGetListName: string; override;
```

C++

```
virtual __fastcall AnsiString ImplGetListName();
```

Description

ImplGetListName uses the iteration index to retrieve a name from the strings list. The name may be blank.

See Also

Strings (see page 24390)

173.1.63.4.5 TCustomStringsValuesList.ImplGetListNameOfValue Method

Looks up a name for a value.

Pascal

```
function ImplGetListNameOfValue(Value: Variant): string; override;
```

C++

```
virtual __fastcall AnsiString ImplGetListNameOfValue(Variant Value);
```

Description

ImplGetListNameOfValue searches the string list for a matching value and returns the name associated with this value. An empty string is returned if the value is not found or if the name associated with the value is blank.

See Also

Strings (see page 24390)

173.1.63.4.6 TCustomStringsValuesList.ImplGetListValue Method

Responsible for returning the value of the name/value pair.

Pascal

```
function ImplGetListValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetListValue();
```

Description

ImplGetListValue uses the iteration index to retrieve a value from the strings list.

See Also

Strings (see page 24390)

173.1.63.4.7 TCustomStringsValuesList.ImplNextIteration Method

Implements IteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

ImplNextIteration increments the iteration index. false is returned if the iteration index is out of the range defined by the Strings (see page 24390) property's Count property. The iteration index is used when retrieving a name or value from the strings list.

See Also

NextIteration (see page 19723)

173.1.63.4.8 TCustomStringsValuesList.ImplNotifyDeactivate Method

Clears the iteration index.

Pascal

```
procedure ImplNotifyDeactivate; override;
```

C++

```
virtual __fastcall ImplNotifyDeactivate();
```

Description

The StringsPrepared (see page 24391) property is cleared. This ensures that the strings list will be prepared during the processing of the next HTTP request.

173.1.63.4.9 TCustomStringsValuesList.ImplStartIterator Method

Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

ImplStartIterator initializes the iteration index to 0. false is returned if the iteration index is out of the range defined by the Strings (see page 24390) property's Count property. The iteration index is used when retrieving a name or value from the strings list.

See Also

startiterator (see page 19724)

173.1.63.4.10 TCustomStringsValuesList.PrepareStrings Method

Customizes the content of the strings list.

Pascal

```
procedure PrepareStrings; virtual;
```

C++

```
virtual __fastcall PrepareStrings();
```

Description

PrepareStrings calls the OnPrepareStrings (see page 24391) event handler.

See Also

StringsPrepared (see page 24391)

OnPrepareStrings (see page 24391)

173.1.63.5 TCustomStringsValuesList Properties

The properties of the TCustomStringsValuesList class are listed here.

Protected Properties

	Name	Description
	StringsPrepared (see page 24391)	This flag indicates whether the strings list has been prepared or customized.

Public Properties

	Name	Description
	Strings (see page 24390)	This string list defines the list of name/value pairs.

173.1.63.5.1 TCustomStringsValuesList.Strings Property

This string list defines the list of name/value pairs.

Pascal

```
property Strings: TStrings;
```

C++

```
__property TStrings Strings;
```

Description

Each line in the string list represents a name/value pair. The name is optional. To specify a name/value enter a line with the name following by "=" followed by the value. For example, "Joe Smith=EMP111" is a named value. "EMP111" is an unnamed value.

173.1.63.5.2 TCustomStringsValuesList.StringsPrepared Property

This flag indicates whether the strings list has been prepared or customized.

Pascal

```
property StringsPrepared: Boolean;
```

C++

```
__property Boolean StringsPrepared;
```

Description

This flag is checked before calling PrepareStrings (see page 24390).

See Also

PrepareStrings (see page 24390)

173.1.63.6 TCustomStringsValuesList Events

The events of the TCustomStringsValuesList class are listed here.

Public Events

	Name	Description
	OnPrepareStrings (see page 24391)	Implement OnPrepareStrings to customize the content of the strings list.

173.1.63.6.1 TCustomStringsValuesList.OnPrepareStrings Event

Implement OnPrepareStrings to customize the content of the strings list.

Pascal

```
property OnPrepareStrings: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPrepareStrings;
```

Description

OnPrepareStrings is called before the name/value pairs are retrieved by server-side script.

See Also

TNotifyEvent (see page 3298)

PrepareStrings (see page 24390)

173.1.64 TDefaultFieldsAdapter Class

TDefaultFieldsAdapter is a base class for adapters that automatically adds default fields and/or actions.

Class Hierarchy

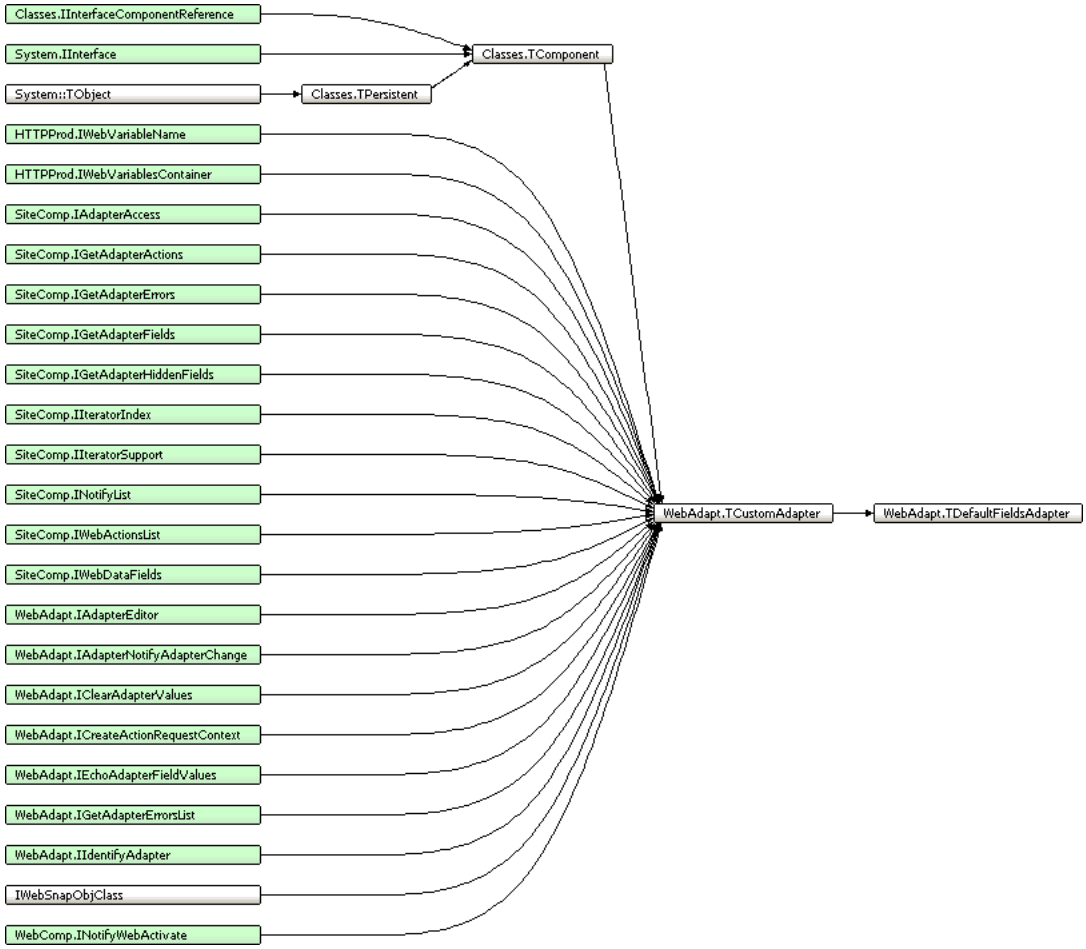

Pascal

```
TDefaultFieldsAdapter = class(TCustomAdapter);
```

C++

```
class TDefaultFieldsAdapter : public TCustomAdapter;
```

File

WebAdapt

Description

Derive from TDefaultFieldsAdapter to define adapter components that automatically add adapter field and/or adapter action components. Default fields are added to the adapter unless one or more field has been explicitly added.

Default actions are added to the adapter unless one or more action has been explicitly added.

See Also

TCustomAdapter (see page 24139)

173.1.64.1 TDefaultFieldsAdapter Members

The following tables list the members exposed by TDefaultFieldsAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.

	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.

	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.

	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 24184)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24185)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24186)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24186)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 24187)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 24402)	Container for adapter actions components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Data (see page 24402)	Container for adapter field components.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.64.2 TDefaultFieldsAdapter Methods

The methods of the TDefaultFieldsAdapter class are listed here.

Protected Methods

	Name	Description
	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.

173.1.64.2.1 TDefaultFieldsAdapter.AddDefaultActions Method

Adds default action components to this adapter.

Pascal

```
procedure AddDefaultActions(AComponents: TWebComponentList); virtual;
```

C++

```
virtual __fastcall AddDefaultActions(TWebComponentList AComponents);
```

Description

AddDefaultActions calls ImplGetActionsList (see page 24165). Descendant classes should override TCustomAdapter (see page 24139)'s ImplGetActionsList (see page 24165) method to provide the list of default actions that will be automatically added to this adapter.

See Also

TWebComponentList (see page 24708)

ImplGetActionsList (see page 24165)

173.1.64.2.2 TDefaultFieldsAdapter.AddDefaultFields Method

Adds default field components to this adapter.

Pascal

```
procedure AddDefaultFields(AComponents: TWebComponentList); virtual;
```

C++

```
virtual __fastcall AddDefaultFields(TWebComponentList AComponents);
```

Description

AddDefaultFields calls ImplGetFieldList. Descendant classes should override TCustomAdapter (see page 24139)'s ImplGetFieldList method to provide the list of default fields that will be automatically added to this adapter.

See Also

TWebComponentList (see page 24708)

ImplGetFieldsList (see page 24165)

173.1.64.2.3 TDefaultFieldsAdapter.DefaultWebComponentsChange Method

Notifies other components of changes to the list of default actions or fields.

Pascal

```
procedure DefaultWebComponentsChange(Sender: TObject);
```

C++

```
__fastcall DefaultWebComponentsChange(TObject * Sender);
```

Description

The list of default fields or actions may be based on properties of the adapter or properties of other components. For example, the default fields of TDataSetAdapter are based on the fields in the dataset. If the TDataSetAdapter references a different dataset then the default fields list will change. If the dataset's field list changes then TDataSetAdapter's default fields will change. DefaultWebCompoentsChange is called in these circumstances so that components that are using this adapter can be notified of changes to the list of fields or actions. DefaultWebComponentsChange calls NotifyAdapterChange (see page 24174).

See Also

NotifyAdapterChange (see page 24174)

173.1.64.2.4 TDefaultFieldsAdapter.GetActionsClass Method

Specifies the class to create as a container for adapter actions components.

Pascal

```
function GetActionsClass: TAdapterActionsClass; override;
```

C++

```
virtual __fastcall TAdapterActionsClass GetActionsClass();
```

Description

GetActionsClass returns TAdapterDefaultActions (see page 23770).

See Also

TAdapterDefaultActions (see page 23770)

173.1.64.2.5 TDefaultFieldsAdapter.GetDataClass Method

Specifies the class to create as a container for adapter field components.

Pascal

```
function GetDataClass: TAdapterFieldsClass; override;
```

C++

```
virtual __fastcall TAdapterFieldsClass GetDataClass();
```

Description

GetDataClass returns TAdapterDefaultFields (see page 23778).

See Also

TAdapterDefaultFields (see page 23778)

173.1.64.2.6 TDefaultFieldsAdapter.ImplGetVisibleActions Method

Returns a list of this adapter's action components.

Pascal

```
function ImplGetVisibleActions: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList ImplGetVisibleActions();
```

Description

ImplGetVisibleActions returns a list of actions that have been added to the adapter. If no adapter fields have been explicitly added, ImplGetVisibleActions returns a list containing all default actions.

See Also

TWebComponentList (see page 24708)

ImplGetVisibleActions (see page 24167)

173.1.64.2.7 TDefaultFieldsAdapter.ImplGetVisibleFields Method

Returns a list of this adapter's field components.

Pascal

```
function ImplGetVisibleFields: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList ImplGetVisibleFields();
```

Description

ImplGetVisibleFields returns a list of fields that have been added to the adapter. If no adapter fields have been explicitly added, ImplGetVisibleFields returns a list containing all default fields.

See Also

TWebComponentList (see page 24708)

ImplGetVisibleFields (see page 24167)

173.1.64.2.8 TDefaultFieldsAdapter.ImplSetActions Method

Initializes the Actions (see page 24402) property.

Pascal

```
procedure ImplSetActions(const Value: TAdapterSubComponent); override;
```

C++

```
virtual __fastcall ImplSetActions(const TAdapterSubComponent Value);
```

Description

The Actions (see page 24402) property is the container for adapter action components.

See Also

ImplSetActions (see page 24170)

173.1.64.2.9 TDefaultFieldsAdapter.ImplSetData Method

Initializes the Data (see page 24402) property.

Pascal

```
procedure ImplSetData(const Value: TAdapterSubComponent); override;
```

C++

```
virtual __fastcall ImplSetData(const TAdapterSubComponent Value);
```

Description

The Data (see page 24402) property is the container for adapter field components.

See Also

ImplSetData (see page 24171)

173.1.64.3 TDefaultFieldsAdapter Properties

The properties of the TDefaultFieldsAdapter class are listed here.

Public Properties

	Name	Description
R	Actions (see page 24402)	Container for adapter actions components.
R	Data (see page 24402)	Container for adapter field components.

173.1.64.3.1 TDefaultFieldsAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterDefaultActions;
```

C++

```
__property TAdapterDefaultActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 23770)

173.1.64.3.2 TDefaultFieldsAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterDefaultFields;
```

C++

```
__property TAdapterDefaultFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 23778)

173.1.65 TDefaultFieldsPagedAdapter Class

TDefaultFieldsPagedAdapter is a base class for adapters that support paging.

Class Hierarchy

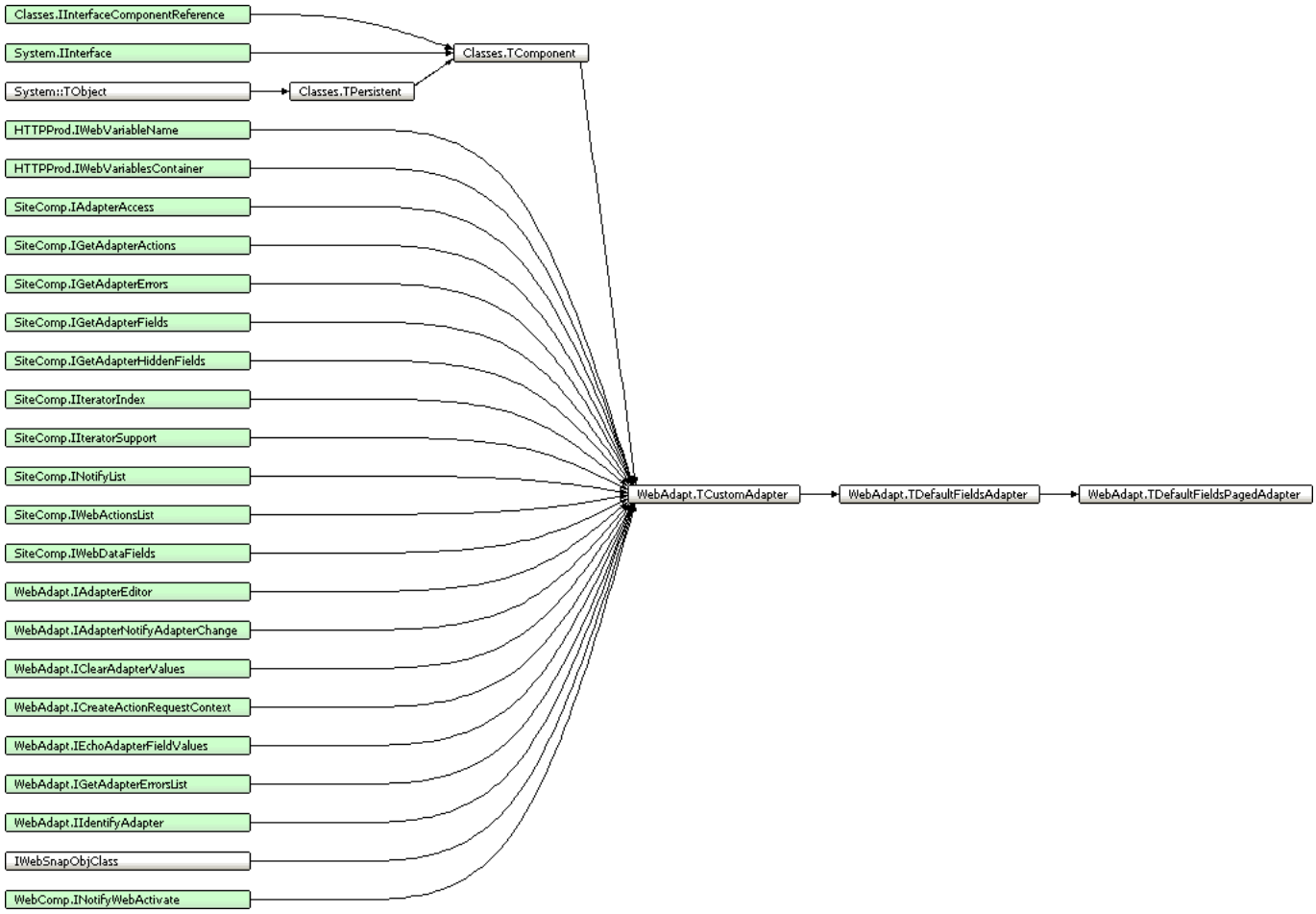

Pascal

```
TDefaultFieldsPagedAdapter = class(TDefaultFieldsAdapter);
```

C++

```
class TDefaultFieldsPagedAdapter : public TDefaultFieldsAdapter;
```

File

WebAdapt

Description

Enable paging by setting the PageSize (see page 24419) property. When paging is enabled, the way that the adapter enumerates records is affected. The adapter calculates the number of pages by dividing the number of records by the page size. It keeps track of a current page number. When the enumerating records, the records on the current page are enumerated. Server-side script simply enumerates the records. Server-side script does not need to be aware of the page size or current page number. When paging is enabled, default actions for moving between pages are automatically added to the adapter.

See Also

TDefaultFieldsAdapter (see page 24392)

TAdapterNextPageAction (see page 23945)

TAdapterPrevPageAction (see page 23979)

TAdapterGotoPageAction (see page 23849)

173.1.65.1 TDefaultFieldsPagedAdapter Members

The following tables list the members exposed by TDefaultFieldsPagedAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24411)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	EncodeActionParams (see page 24411)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
	EndOfPage (see page 24412)	Indicates whether the record index has gone past the last record in the current page.
	ExtractRequestParams (see page 24412)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetEOF (see page 24412)	Indicates whether the record position has passed the last record.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
	GetFirstRecord (see page 24413)	Set the record position to the first record.
	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetNextRecord (see page 24413)	Increments the record position.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPageCount (see page 24413)	Calculates the number of pages needed to display all adapter records.
	GetRecordCount (see page 24414)	Returns the total number of records.
	GetRecordIndex (see page 24414)	Returns the index of the current record.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.

	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 24415)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
V	ImplEndIterator (see page 24415)	Implements IteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 24415)	Provides a list of action types that can be added to this adapter.
V	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
V	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
V	ImplGetVariable (see page 24166)	Returns an action or field.
V	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
V	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
V	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
V	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
V	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
V	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
V	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
V	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
V	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
V	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
V	ImplNextIteration (see page 24416)	Implements IteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
V	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
V	ImplNotifyDeactivate (see page 24416)	Clears the current page variable.
V	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
V	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
V	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
V	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
V	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
V	ImplStartIterator (see page 24416)	Implements IteratorSupport.StartIterator (see page 24176) to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.

	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetPageSize (see page 24417)	Updates the pagesize variable.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SkipToPage (see page 24417)	Skips records until the current record is within the current page.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 24184)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24185)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24186)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24186)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 24187)	Supplies name/value pairs needed to execute an adapter action.
	OnGetEOF (see page 24420)	Implement this event to indicate when the current record index has exceeded the total record count.

	OnGetFirstRecord (see page 24420)	Implement this method to initialize the iteration index to zero.
	OnGetNextRecord (see page 24421)	Implement this method to increment the iteration index.
	OnGetRecordCount (see page 24421)	Implement this event to identify the total number of records that can be iterated.
	OnGetRecordIndex (see page 24421)	Implement this event to identify the zero-based current record.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.

	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 24181)	Actions is the container for adapter actions components.
R	Actions (see page 24402)	Container for adapter actions components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	CurrentPage (see page 24418)	Sets or gets the current page index.
R	Data (see page 24181)	Data is the container for adapter field components.
R	Data (see page 24402)	Container for adapter field components.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	PageCount (see page 24418)	Calculates the number of pages needed to display all adapter records.
	PageSize (see page 24419)	Sets or gets the page size variable.
R	RecordCount (see page 24419)	Returns the total number of records.
R	RecordIndex (see page 24419)	Returns the index of the current record.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.65.2 TDefaultFieldsPagedAdapter Methods

The methods of the TDefaultFieldsPagedAdapter class are listed here.

Protected Methods

	Name	Description
V	AddDefaultActions (see page 24411)	Adds default action components to this adapter.
V	EncodeActionParams (see page 24411)	Writes name/value pairs that represent adapter state.
	EndOfPage (see page 24412)	Indicates whether the record index has gone past the last record in the current page.

	ExtractRequestParams (see page 24412)	Reads name/value pairs from the HTTP request.
	GetEOF (see page 24412)	Indicates whether the record position has passed the last record.
	GetFirstRecord (see page 24413)	Set the record position to the first record.
	GetNextRecord (see page 24413)	Increments the record position.
	GetPageCount (see page 24413)	Calculates the number of pages needed to display all adapter records.
	GetRecordCount (see page 24414)	Returns the total number of records.
	GetRecordIndex (see page 24414)	Returns the index of the current record.
	ImplCanAddActionClass (see page 24415)	Indicates whether a type of adapter action can be added to this adapter.
	ImplEndIterator (see page 24415)	Implements IteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
	ImplGetActionsList (see page 24415)	Provides a list of action types that can be added to this adapter.
	ImplNextIteration (see page 24416)	Implements IteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyDeactivate (see page 24416)	Clears the current page variable.
	ImplStartIterator (see page 24416)	Implements IteratorSupport.StartIterator (see page 24176) to support iterating over the records of the adapter.
	SetPageSize (see page 24417)	Updates the pagesize variable.
	SkipToPage (see page 24417)	Skips records until the current record is within the current page.

173.1.65.2.1 TDefaultFieldsPagedAdapter.AddDefaultActions Method

Adds default action components to this adapter.

Pascal

```
procedure AddDefaultActions(AComponents: TWebComponentList); override;
```

C++

```
virtual __fastcall AddDefaultActions(TWebComponentList AComponents);
```

Description

AddDefaultActions adds TAdapterNextPageAction (see page 23945), TAdapterGotoPageAction (see page 23849), and TAdapterPrevPageAction (see page 23979).

See Also

TWebComponentList (see page 24708)

TAdapterNextPageAction (see page 23945)

TAdapterGotoPageAction (see page 23849)

TAdapterPrevPageAction (see page 23979)

173.1.65.2.2 TDefaultFieldsPagedAdapter.EncodeActionParams Method

Writes name/value pairs that represent adapter state.

Pascal

```
procedure EncodeActionParams(AParams: IAdapterItemRequestParams); override;
```

C++

```
virtual __fastcall EncodeActionParams(IAdapterItemRequestParams AParams);
```

Description

ExtractRequestParams (see page 24412) writes the current page number as a name/value pair.

See Also

ExtractRequestParams (see page 24153)

ExtractRequestParams (see page 24412)

173.1.65.2.3 TDefaultFieldsPagedAdapter.EndOfPage Method

Indicates whether the record index has gone past the last record in the current page.

Pascal

```
function EndOfPage: Boolean;
```

C++

```
__fastcall Boolean EndOfPage();
```

Description

EndOfPage tests for the end of page using PageSize (see page 24419), RecordIndex (see page 24419), and CurrentPage (see page 24418).

See Also

PageSize (see page 24419)

RecordIndex (see page 24419)

PageSize (see page 24419)

CurrentPage (see page 24418)

173.1.65.2.4 TDefaultFieldsPagedAdapter.ExtractRequestParams Method

Reads name/value pairs from the HTTP request.

Pascal

```
procedure ExtractRequestParams(ARequest: IUnknown); override;
```

C++

```
virtual __fastcall ExtractRequestParams(IUnknown ARequest);
```

Description

ExtractRequestParams is called by a child adapter action before executing the action. This gives the adapter an opportunity to extract values from the HTTP request that are important to the behavior of the adapter. ExtractRequestParams extracts the current page number from the request.

See Also

EncodeActionParams (see page 24411)

ExtractRequestParams (see page 24153)

173.1.65.2.5 TDefaultFieldsPagedAdapter.GetEOF Method

Indicates whether the record position has passed the last record.

Pascal

```
function GetEOF: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetEOF();
```

Description

GetEOF calls the OnGetFirstRecord (see page 24420) event handler. If OnGetFirstRecord (see page 24420) is not implemented then the RecordIndex (see page 24419) is compared to the RecordCount (see page 24419). Descendant classes may override this method. For example, TDataSetAdapter tests the dataset's EOF.

See Also

OnGetEOF (see page 24420)

RecordIndex (see page 24419)

RecordCount (see page 24419)

173.1.65.2.6 TDefaultFieldsPagedAdapter.GetFirstRecord Method

Set the record position to the first record.

Pascal

```
function GetFirstRecord: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetFirstRecord();
```

Description

GetFirstRecord calls the OnGetFirstRecord (see page 24420) event handler. Descendant classes may override this method. For example, TDataSetAdapter sets the record position of a dataset.

See Also

OnGetFirstRecord (see page 24420)

173.1.65.2.7 TDefaultFieldsPagedAdapter.GetNextRecord Method

Increments the record position.

Pascal

```
function GetNextRecord: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetNextRecord();
```

Description

GetNextRecord calls the OnGetNextRecord (see page 24421) event handler. Descendant classes may override this method. For example, TDataSetAdapter increments the record position of a dataset.

See Also

OnGetNextRecord (see page 24421)

173.1.65.2.8 TDefaultFieldsPagedAdapter.GetPageCount Method

Calculates the number of pages needed to display all adapter records.

Pascal

```
function GetPageCount: Integer;
```

C++

```
__fastcall int GetPageCount();
```

Description

GetPageCount calculates the page count using PageCount (see page 24418), PageSize (see page 24419), and RecordCount (see page 24419).

See Also

RecordCount (see page 24419)

PageCount (see page 24418)

PageSize (see page 24419)

173.1.65.2.9 TDefaultFieldsPagedAdapter.GetRecordCount Method

Returns the total number of records.

Pascal

```
function GetRecordCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetRecordCount();
```

Description

GetRecordCount calls the OnGetRecordCount (see page 24421) event handler. Descendant classes may override this method. For example, TDataSetAdapter returns the number of records in the dataset.

See Also

OnGetRecordCount (see page 24421)

RecordCount (see page 24419)

173.1.65.2.10 TDefaultFieldsPagedAdapter.GetRecordIndex Method

Returns the index of the current record.

Pascal

```
function GetRecordIndex: Integer; virtual;
```

C++

```
virtual __fastcall int GetRecordIndex();
```

Description

GetRecordIndex calls the OnGetRecordIndex (see page 24421) event handler.

See Also

OnGetRecordIndex (see page 24421)

173.1.65.2.11 TDefaultFieldsPagedAdapter.ImplCanAddActionClass Method

Indicates whether a type of adapter action can be added to this adapter.

Pascal

```
function ImplCanAddActionClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddActionClass allows action classes that derive from TCustomPagedAdapterAction (see page 24371) in addition to those allowed by TCustomAdapter (see page 24139)'s ImplCanAddActionsClass method.

See Also

ImplCanAddActionClass (see page 24163)

ImplCanAddClass (see page 23743)

TCustomPagedAdapterAction (see page 24371)

173.1.65.2.12 TDefaultFieldsPagedAdapter.ImplEndIterator Method

Implements IteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer); override;
```

C++

```
virtual __fastcall ImplEndIterator(void * OwnerData);
```

Description

EndIterator (see page 24152) is called when record iteration is complete.

See Also

enditerator (see page 19723)

173.1.65.2.13 TDefaultFieldsPagedAdapter.ImplGetActionsList Method

Provides a list of action types that can be added to this adapter.

Pascal

```
procedure ImplGetActionsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetActionsList(TStrings AList);
```

Description

ImplGetActionsList lists PrevPage, GotoPage, and NextPage actions.

See Also

TAdapterNextPageAction (see page 23945)

TAdapterGotoPageAction (see page 23849)

TAdapterPrevPageAction (see page 23979)

ImplGetActionsList (see page 24165)

173.1.65.2.14 TDefaultFieldsPagedAdapter.ImplNextIteration Method

Implements IteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

ImplNextIteration calls GetNextRecord (see page 24413) if the RecordIndex (see page 24419) can be incremented without running past the end of the current page. false is returned if RecordIndex (see page 24419) can't be incremented.

See Also

NextIteration (see page 19723)

RecordIndex (see page 24419)

RecordCount (see page 24419)

GetNextRecord (see page 24413)

GetEOF (see page 24412)

EndOfPage (see page 24412)

173.1.65.2.15 TDefaultFieldsPagedAdapter.ImplNotifyDeactivate Method

Clears the current page variable.

Pascal

```
procedure ImplNotifyDeactivate; override;
```

C++

```
virtual __fastcall ImplNotifyDeactivate();
```

Description

The current page is reset to zero when the module is deactivated. This ensures that the current page set during the processing of an HTTP request will not affect another HTTP request. To keep track of the current page for a particular end-user, the current page is typically passed between end-user requests using action parameters.

See Also

ImplNotifyDeactivate (see page 24170)

EncodeActionParams (see page 24411)

ExtractRequestParams (see page 24412)

173.1.65.2.16 TDefaultFieldsPagedAdapter.ImplStartIterator Method

Implements IteratorSupport.StartIterator (see page 24176) to support iterating over the records of the adapter.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

ImplStartIterator locates the first record in the current page using the GetFirstRecord (see page 24413) and SkipToPage (see page 24417) methods.

See Also

startiterator (see page 19724)

ImplNextIteration (see page 24416)

GetFirstRecord (see page 24413)

SkipToPage (see page 24417)

CurrentPage (see page 24418)

173.1.65.2.17 TDefaultFieldsPagedAdapter.SetPageSize Method

Updates the pagesize variable.

Pascal

```
procedure SetPageSize(const Value: Integer);
```

C++

```
__fastcall SetPageSize(const int Value);
```

Description

If the pagesize variable is not zero and no actions have been explicitly added to the adapter then the NextPage, PrevPage, and GotoPage are automatically added. If the pagesize variable is changed back to zero then the these actions are removed.

See Also

PageSize (see page 24419)

TAdapterNextPageAction (see page 23945)

TAdapterPrevPageAction (see page 23979)

TAdapterGotoPageAction (see page 23849)

173.1.65.2.18 TDefaultFieldsPagedAdapter.SkipToPage Method

Skips records until the current record is within the current page.

Pascal

```
procedure SkipToPage;
```

C++

```
__fastcall SkipToPage();
```

Description

SkipToPage calls GetNextRecord (see page 24413) to reach the first record in the current page.

See Also

CurrentPage (see page 24418)

GetNextRecord (see page 24413)

173.1.65.3 TDefaultFieldsPagedAdapter Properties

The properties of the TDefaultFieldsPagedAdapter class are listed here.

Public Properties

	Name	Description
	CurrentPage (see page 24418)	Sets or gets the current page index.
R	PageCount (see page 24418)	Calculates the number of pages needed to display all adapter records.
	PageSize (see page 24419)	Sets or gets the page size variable.
R	RecordCount (see page 24419)	Returns the total number of records.
R	RecordIndex (see page 24419)	Returns the index of the current record.

173.1.65.3.1 TDefaultFieldsPagedAdapter.CurrentPage Property

Sets or gets the current page index.

Pascal

```
property CurrentPage: Integer;
```

C++

```
__property int CurrentPage;
```

Description

Current page is a zero-based index that must be less than PageCount (see page 24418). Current page is used by the NextPage, GotoPage, and PrevPage actions to move between page.

See Also

PageCount (see page 24418)

TAdapterNextPageAction (see page 23945)

TAdapterPrevPageAction (see page 23979)

TAdapterGotoPageAction (see page 23849)

EncodeActionParams (see page 24411)

ExtractRequestParams (see page 24412)

173.1.65.3.2 TDefaultFieldsPagedAdapter.PageCount Property

Calculates the number of pages needed to display all adapter records.

Pascal

```
property PageCount: Integer;
```

C++

```
__property int PageCount;
```

Description

PageCount returns the result of GetPageCount (see page 24413).

See Also

GetPageCount (see page 24413)

RecordCount (see page 24419)

PageCount

PageSize (see page 24419)

173.1.65.3.3 TDefaultFieldsPagedAdapter.PageSize Property

Sets or gets the page size variable.

Pascal

```
property PageSize: Integer;
```

C++

```
__property int PageSize;
```

Description

If the page size variable is set to a value greater than zero and no actions have been explicitly added to the adapter then the NextPage, PrevPage, and GotoPage are automatically added. If the page size variable is changed back to zero then the these actions are removed.

See Also

TAdapterNextPageAction (see page 23945)

TAdapterPrevPageAction (see page 23979)

TAdapterGotoPageAction (see page 23849)

173.1.65.3.4 TDefaultFieldsPagedAdapter.RecordCount Property

Returns the total number of records.

Pascal

```
property RecordCount: Integer;
```

C++

```
__property int RecordCount;
```

Description

The RecordCount property returns the result of GetRecordCount (see page 24414).

See Also

GetRecordCount (see page 24414)

173.1.65.3.5 TDefaultFieldsPagedAdapter.RecordIndex Property

Returns the index of the current record.

Pascal

```
property RecordIndex: Integer;
```

C++

```
__property int RecordIndex;
```

Description

RecordIndex returns the result of GetRecordIndex (see page 24414).

See Also

OnGetRecordIndex (see page 24421)

GetRecordIndex (see page 24414)

173.1.65.4 TDefaultFieldsPagedAdapter Events

The events of the TDefaultFieldsPagedAdapter class are listed here.

Public Events

	Name	Description
	OnGetEOF (see page 24420)	Implement this event to indicate when the current record index has exceeded the total record count.
	OnGetFirstRecord (see page 24420)	Implement this method to initialize the iteration index to zero.
	OnGetNextRecord (see page 24421)	Implement this method to increment the iteration index.
	OnGetRecordCount (see page 24421)	Implement this event to identify the total number of records that can be iterated.
	OnGetRecordIndex (see page 24421)	Implement this event to identify the zero-based current record.

173.1.65.4.1 TDefaultFieldsPagedAdapter.OnGetEOF Event

Implement this event to indicate when the current record index has exceeded the total record count.

Pascal

```
property OnGetEOF: TGetEOFEvent;
```

C++

```
__property TGetEOFEvent OnGetEOF;
```

Description

OnGetEOF is called to determine when to stop the iteration. If OnGetEOF is not implemented, the result of the OnGetRecordCount (see page 24421) event handler will be compared to the result of the OnGetRecordIndex (see page 24421) event handler.

173.1.65.4.2 TDefaultFieldsPagedAdapter.OnGetFirstRecord Event

Implement this method to initialize the iteration index to zero.

Pascal

```
property OnGetFirstRecord: TGetRecordEvent;
```

C++

```
__property TGetRecordEvent OnGetFirstRecord;
```

Description

This method is called when the iteration is started. If there are no records then set the EOF flag.

173.1.65.4.3 TDefaultFieldsPagedAdapter.OnGetNextRecord Event

Implement this method to increment the iteration index.

Pascal

```
property OnGetNextRecord: TGetRecordEvent;
```

C++

```
__property TGetRecordEvent OnGetNextRecord;
```

Description

This method is called when the next record is requested. This method is not a next then set the EOF flag.

173.1.65.4.4 TDefaultFieldsPagedAdapter.OnGetRecordCount Event

Implement this event to identify the total number of records that can be iterated.

Pascal

```
property OnGetRecordCount: TGetRecordCountEvent;
```

C++

```
__property TGetRecordCountEvent OnGetRecordCount;
```

Description

OnGetRecordCount is called to calculate the number of pages.

173.1.65.4.5 TDefaultFieldsPagedAdapter.OnGetRecordIndex Event

Implement this event to identify the zero-based current record.

Pascal

```
property OnGetRecordIndex: TGetRecordIndexEvent;
```

C++

```
__property TGetRecordIndexEvent OnGetRecordIndex;
```

Description

OnGetRecordIndex is called to calculate the current page.

173.1.66 TEndUserAdapter Class

TEndUserAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.

Class Hierarchy

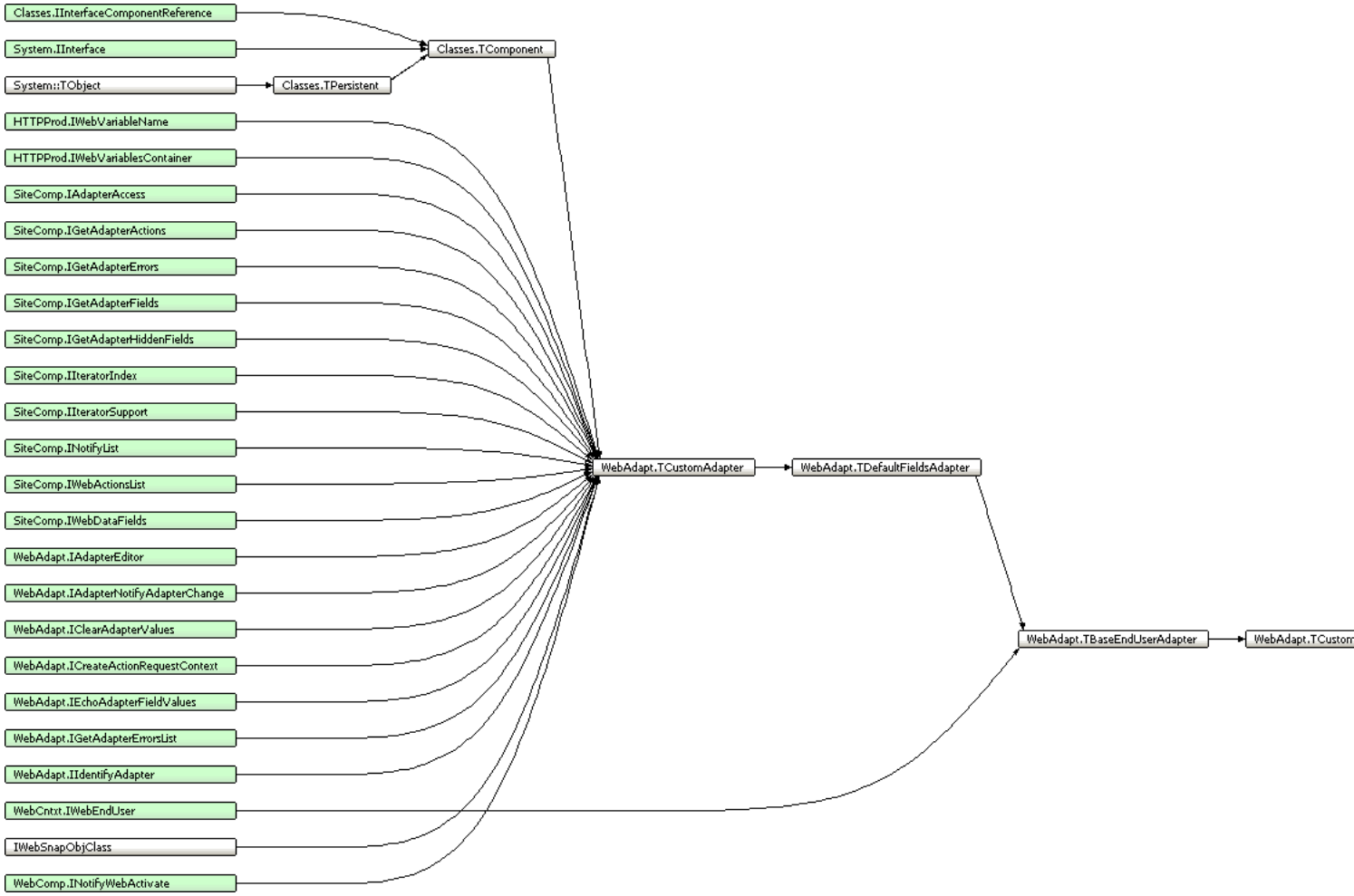

Pascal

```
TEndUserAdapter = class(TCustomEndUserAdapter);
```

C++

```
class TEndUserAdapter : public TCustomEndUserAdapter;
```

File

WebAdapt

Description

Add TEndUserSessionAdapter (see page 24458) to a WebSnap application where users must log in before accessing Web pages. TEndUserSessionAdapter (see page 24458) handles login and logout actions on behalf of the end user by generating events where you can retrieve user information and execute user actions

TEndUserAdapter supports two default Adapter fields: DisplayName and LoggedIn. These fields provide access to the user's name and login state. Implement the OnGetDisplayName (see page 24434) and OnIsLoggedIn (see page 24434) events to provide values for these fields.

TEndUserAdapter contains two default actions: LoginForm and LogoutOut. These actions are used to display a login form and to

log the user out. Use the LoginPage (see page 24431) property to specify the name of the login page. Implement the OnLogin (see page 24435) and OnLogout (see page 24436) events to change the login state of an end user.

The EndUser script variable is used to by server-side script to display end user information and execute end user actions. The following JavaScript displays the name of the logged in end user, if any.

```
<% if (EndUser.LoggedIn) { %>
<h1>Welcome <%= EndUser.DisplayName %> </h1>
<% } %>
```

The following JavaScript displays a login or logout hyperlink.

```
<%   if (EndUser.Logout.Enabled) { %>
  <a href="<%=EndUser.Logout.AsHREF%>">Logout</a>
<%   } %>
<%   if (EndUser.LoginForm.Enabled) { %>
  <a href=<%=EndUser.LoginForm.AsHREF%>>Login</a>
<%   } %>
```

TEndUserAdapter also determines access rights for a user. If the WebSnap application contains a TWebUserList or equivalent, TEndUserAdapter calls the TWebUserList component to check rights for a particular user ID. Implement the OnHasRights (see page 24434) event to check rights in an event handler.

173.1.66.1 TEndUserAdapter Members

The following tables list the members exposed by TEndUserAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
V	CanExecuteLoginForm (see page 24106)	Determines if there is a static login page.
V	CanExecuteLogout (see page 24107)	Determines if the user is logged in.
V	CanProduceLoginPage (see page 24107)	Determines whether there is a static login page or an automatically produced login page.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.

	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoOnHasRights (see page 24107)	Determines whether the user has view rights.
	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
	ExecuteLogin (see page 24108)	Starts the server-side login process.
	ExecuteLogin (see page 24321)	Generates an OnLogin (see page 24325) event.
	ExecuteLoginForm (see page 24108)	Creates the HTML login page.
	ExecuteLogout (see page 24108)	Performs server-side logout.
	ExecuteLogout (see page 24322)	Generates an OnLogout (see page 24326) event.
	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetDisplayName (see page 24109)	Retrieves the display name string.
	GetDisplayNameValue (see page 24109)	Gets the display name of the end user.
	GetDisplayNameValue (see page 24322)	Generates an OnGetDisplayName (see page 24324) event.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetEndUserAdapter (see page 24109)	Gets the end-user adapter.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetLoggedIn (see page 24110)	Determines if the user is logged in
	GetLoggedInValue (see page 24110)	Determines if the user is logged in.
	GetLoggedInValue (see page 24323)	Generates an OnIsLoggedIn (see page 24324) event.
	GetLoginFormAction (see page 24110)	Gets the action component for the login form.
	GetLogoutAction (see page 24111)	Gets the action component for the logout form.
	GetOwner (see page 2778)	Returns the owner of a component.

	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetUserID (see page 24111)	Gets the UserID from this TBaseEndUserAdapter (see page 24097).
	GetUserID (see page 24323)	Generates an OnGetUserID (see page 24324) event.
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasExecuteRights (see page 24111)	Determines if the user has execute rights for the TBaseEndUserAdapter (see page 24097).
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasModifyRights (see page 24112)	Determines if the user has modify rights for the TBaseEndUserAdapter (see page 24097).
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
	HasViewRights (see page 24112)	Determine if the user has view rights to the TBaseEndUserAdapter (see page 24097).
	ImplCanAddActionClass (see page 24113)	Determines if the given action class can be added.
	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 24113)	Determines if the given action class can be added.
	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 24113)	Lists supported end-user adapter actions.
	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
	ImplGetDisplayName (see page 24114)	Gets the display name for this TBaseEndUserAdapter (see page 24097).
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetEndUserAdapter (see page 24114)	Returns this TBaseEndUserAdapter (see page 24097).
	ImplGetFieldsList (see page 24114)	Obtains the list of all of the fields for the end-user adapter.
	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetLoggedIn (see page 24114)	Checks the value of the LoggedIn field.
	ImplGetLoginFormAction (see page 24115)	Gets the LoginForm action component for this TBaseEndUserAdapter (see page 24097).
	ImplGetLogoutAction (see page 24115)	Gets the Logout action component for this TBaseEndUserAdapter (see page 24097).
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasExecuteRights (see page 24115)	Checks user rights and determines if the user has execute rights.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasModifyRights (see page 24116)	Checks user rights and determines if the user has modify rights.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.

	ImplHasViewRights (see page 24116)	Checks user rights and determines if the user has view rights.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplLogIn (see page 24116)	Processes the login for the given user.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplProduceLoginPage (see page 24117)	Generate the HTML login page.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	LogIn (see page 24117)	Logs in a specified user.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceLoginPage (see page 24117)	Generates the login page.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.

	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UserList (see page 24118)	Gets the user list.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetUserID (see page 24324)	Occurs when the adapter needs to provide the ID of the current end user.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).

	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetDisplayName (see page 24673)	Returns the value of the DisplayName (see page 24678) property.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEndUserAdapter (see page 24674)	Returns the value of the EndUserAdapter (see page 24678) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
	GetLoggedIn (see page 24674)	Returns the value of the LoggedIn (see page 24678) property.
	GetLoginFormAction (see page 24674)	Returns the value of the LoginFormAction (see page 24679) property.
	GetLogoutAction (see page 24675)	Returns the value of the LogoutAction (see page 24680) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.

	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasExecuteRights (see page 24675)	Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasModifyRights (see page 24676)	Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	HasViewRights (see page 24676)	Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LogIn (see page 24677)	Logs in a specified user.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	ProduceLoginPage (see page 24677)	Generates the Web page for logging in users.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 24678)	Indicates the name of the end user as it should be used for display in Web pages.

	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	EndUserAdapter (see page 24678)	Provides access to the end user adapter that implements this interface.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	LoggedIn (see page 24678)	Indicates whether the current user is logged in.
R	LoginFormAction (see page 24679)	Indicates the action component for the login form.
R	LogoutAction (see page 24680)	Indicates the action component for logging a user out.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 24432)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24432)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 24433)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24433)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 24433)	Supplies name/value pairs needed to execute an adapter action.
	OnGetDisplayName (see page 24434)	Occurs when the adapter needs to provide the name of the current end user.
	OnHasRights (see page 24434)	Determines if the user rights match the rights being queried.
	OnIsLoggedIn (see page 24434)	Occurs when the adapter needs to indicate whether the current end user has logged in.
	OnLogin (see page 24435)	Logs in the current end user.
	OnLogout (see page 24436)	Occurs when the current user logs out.
	OnProduceLoginPage (see page 24436)	Allows implementation of a customized login page.

Published Properties

	Name	Description
	Actions (see page 24431)	Container for adapter actions components.
	Data (see page 24431)	Container for adapter field components.
	LoginPage (see page 24431)	Specifies the name of the page that acts as the login page.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.66.2 TEndUserAdapter Properties

The properties of the TEndUserAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 24431)	Container for adapter actions components.
	Data (see page 24431)	Container for adapter field components.
	LoginPage (see page 24431)	Specifies the name of the page that acts as the login page.

173.1.66.2.1 TEndUserAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 23770)

173.1.66.2.2 TEndUserAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 23778)

173.1.66.2.3 TEndUserAdapter.LoginPage Property

Specifies the name of the page that acts as the login page.

Pascal

```
property LoginPage: string;
```

C++

```
__property AnsiString LoginPage;
```

Description

Use LoginPage to read or set the name of the login page for this TBaseEndUserAdapter. The name of the login page is the name of the Web page module that generates the login page.

173.1.66.3 TEndUserAdapter Events

The events of the TEndUserAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 24432)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24432)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 24433)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24433)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 24433)	Supplies name/value pairs needed to execute an adapter action.
	OnGetDisplayName (see page 24434)	Occurs when the adapter needs to provide the name of the current end user.
	OnHasRights (see page 24434)	Determines if the user rights match the rights being queried.
	OnIsLoggedIn (see page 24434)	Occurs when the adapter needs to indicate whether the current end user has logged in.
	OnLogin (see page 24435)	Logs in the current end user.
	OnLogout (see page 24436)	Occurs when the current user logs out.
	OnProduceLoginPage (see page 24436)	Allows implementation of a customized login page.

173.1.66.3.1 TEndUserAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

173.1.66.3.2 TEndUserAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.66.3.3 TEndUserAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 24076)

OnAfterExecuteAction (see page 24184)

173.1.66.3.4 TEndUserAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 24076)

OnAfterGetActionResponse (see page 24185)

173.1.66.3.5 TEndUserAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 24077)

173.1.66.3.6 TEndUserAdapter.OnGetDisplayName Event

Occurs when the adapter needs to provide the name of the current end user.

Pascal

```
property OnGetDisplayName: TEndUserGetNameEvent;
```

C++

```
__property TEndUserGetNameEvent OnGetDisplayName;
```

Description

Write an OnGetDisplayName event handler to provide the value of the DisplayName field. When a server-side script (or a caller of the IWebEndUser interface) requests the display name for the current end user, TCustomEndUserAdapter generates an OnGetDisplayName event to retrieve that value.

Note: If you do not supply an OnGetDisplayName event handler, TCustomEndUserAdapter always returns an empty string as the display name for the current end user.

See Also

DisplayName (see page 24678)

EndUser (see page 24657)

WebContext (see page 24687)

173.1.66.3.7 TEndUserAdapter.OnHasRights Event

Determines if the user rights match the rights being queried.

Pascal

```
property OnHasRights: TEndUserRightsEvent;
```

C++

```
__property TEndUserRightsEvent OnHasRights;
```

Description

The OnHasRights event handler is called from DoOnHasRights to determine if the user rights match the rights being queried. A customized event handler should be written to determine the rights for your specific application.

See Also

DoOnHasRights (see page 24107)

ImplHasExecuteRights (see page 24115)

ImplHasModifyRights (see page 24116)

ImplHasViewRights (see page 24116)

173.1.66.3.8 TEndUserAdapter.OnIsLoggedIn Event

Occurs when the adapter needs to indicate whether the current end user has logged in.

Pascal

```
property OnIsLoggedIn: TEndUserIsLoggedInEvent;
```

C++

```
__property TEndUserIsLoggedInEvent OnIsLoggedIn;
```

Description

Write an OnIsLoggedIn event handler to provide the value of the LoggedIn field. When a server-side script (or a caller of the IWebEndUser interface) inquires whether the current user has logged in, TCustomEndUserAdapter generates an OnIsLoggedIn event to retrieve that information.

Note: If you do not supply an OnIsLoggedIn event handler, TCustomEndUserAdapter always returns false, indicating that the current end user has not logged in.

See Also

LoggedIn (see page 24678)

EndUser (see page 24657)

WebContext (see page 24687)

OnLogin (see page 24325)

OnLogout (see page 24326)

173.1.66.3.9 TEndUserAdapter.OnLogin Event

Logs in the current end user.

Pascal

```
property OnLogin: TEndUserUserIDEvent;
```

C++

```
__property TEndUserUserIDEvent OnLogin;
```

Description

Write an OnLogin event handler to allow the TCustomEndUserAdapter object to log users in. When a server-side script (or a caller of the IWebEndUser interface) attempts to log in a user, TCustomEndUserAdapter generates an OnLogin event to perform whatever steps are involved in the login process.

Note: If you do not supply an OnLogin event handler, TCustomEndUserAdapter does nothing to log users in.

See Also

LogIn (see page 24677)

EndUser (see page 24657)

WebContext (see page 24687)

OnIsLoggedIn (see page 24324)

OnLogout (see page 24326)

TWebUserList (see page 25625)

OnGetUserID (see page 24324)

173.1.66.3.10 TEndUserAdapter.OnLogout Event

Occurs when the current user logs out.

Pascal

```
property OnLogout: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnLogout;
```

Description

Write an OnLogout event handler to allow the TCustomEndUserAdapter object to log users out. When a server-side script attempts to log a user out, TCustomEndUserAdapter generates an OnLogout event to perform whatever steps are involved in the logout process.

Note: If you do not supply an OnLogin event handler, TCustomEndUserAdapter does nothing to log users out.

See Also

OnIsLoggedIn (see page 24324)

OnLogin (see page 24325)

173.1.66.3.11 TEndUserAdapter.OnProduceLoginPage Event

Allows implementation of a customized login page.

Pascal

```
property OnProduceLoginPage: TEndUserHandledEvent;
```

C++

```
__property TEndUserHandledEvent OnProduceLoginPage;
```

Description

The OnProduceLoginPage event is triggered in the ImplProduceLoginPage. If a customized login page is desired for your application, a customized event handler can be written for this event.

See Also

ImplProduceLoginPage (see page 24117)

ProduceLoginPage (see page 24117)

CanProduceLoginPage (see page 24107)

173.1.67 TEndUserAdapterLoginFormAction Class

TEndUserAdapterLoginFormAction is an adapter action that displays a login form when executed.

Class Hierarchy

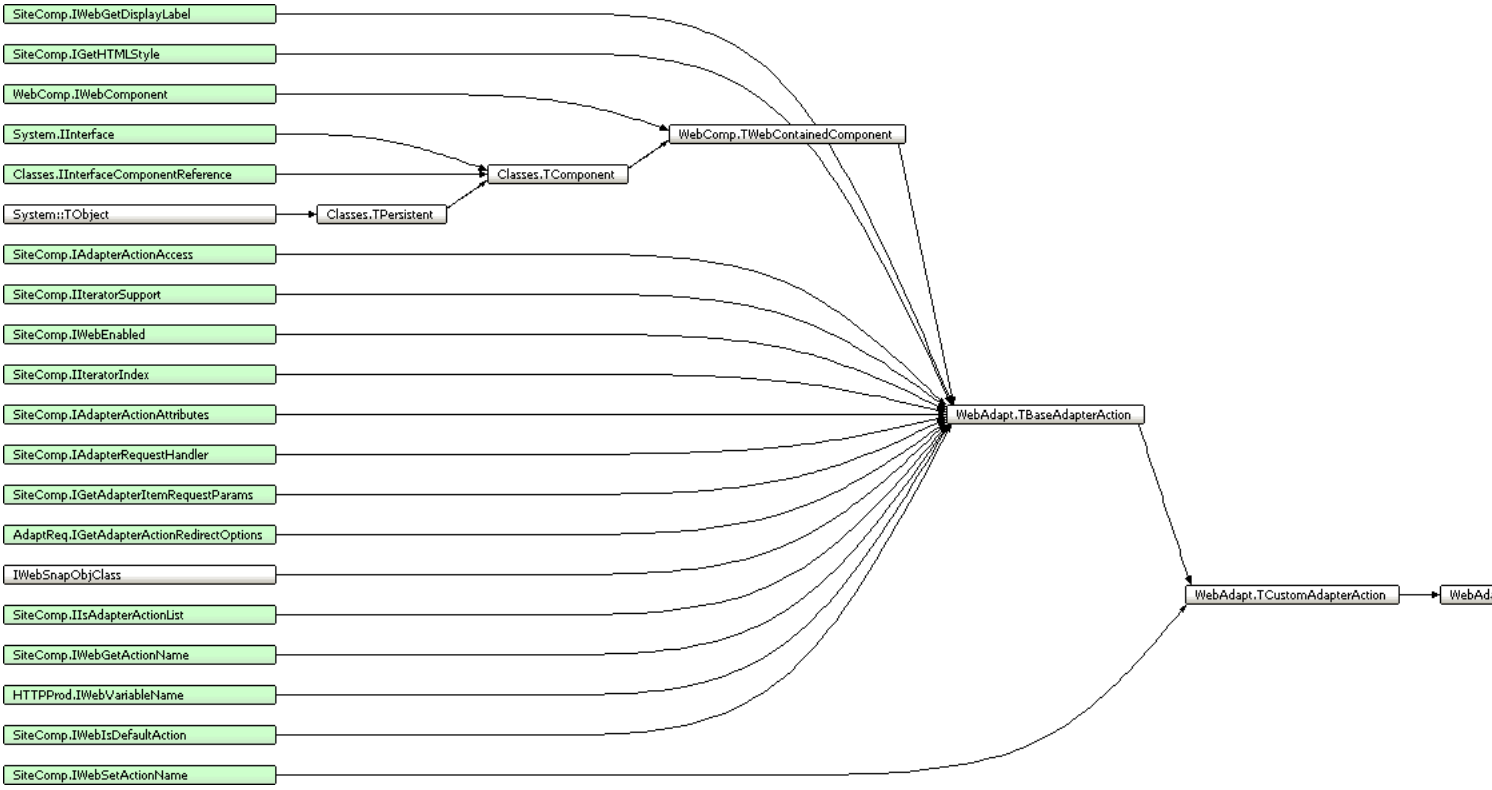

Pascal

```
TEndUserAdapterLoginFormAction = class(TCustomEndUserAdapterAction);
```

C++

```
class TEndUserAdapterLoginFormAction : public TCustomEndUserAdapterAction;
```

File

WebAdapt

Description

This adapter action is typically executed when the end-user clicks on a "Login" hyperlink displayed at the top of a web page. The parent adapter's LoginPage property is used to specify the page displayed when this adapter action is executed.

See Also

TCustomEndUserAdapterAction (see page 24326)

TBaseEndUserAdapter (see page 24097)

TLoginFormAdapter (see page 24500)

TBaseEndUserAdapter (see page 24097)

173.1.67.1 TEndUserAdapterLoginFormAction Members

The following tables list the members exposed by TEndUserAdapterLoginFormAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 24443)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.

	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 24443)	Executes the adapter action.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 24444)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Adapter (see page 24333)	Returns the parent adapter of this adapter action.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.

	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	ExecuteAccess (see page 24073)	Defines the rights required to execute this action.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RedirectOptions (see page 24074)	Indicates whether to perform an HTTP redirect after successfully executing this action.

R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 24445)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 24445)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24446)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24446)	Called before the adapter action response is generated.
	OnGetParams (see page 24447)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 24444)	Suggests a display label for this adapter action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.67.2 TEndUserAdapterLoginFormAction Methods

The methods of the TEndUserAdapterLoginFormAction class are listed here.

Protected Methods

	Name	Description
V	GetDefaultActionName (see page 24443)	Gets the name used to reference this adapter action with server-side script.
V	ImplExecuteActionRequest (see page 24443)	Executes the adapter action.
V	ImplWebEnabled (see page 24444)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

173.1.67.2.1 TEndUserAdapterLoginFormAction.GetDefaultActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultActionName();
```

Description

GetDefaultActionName returns "LoginForm".

See Also

GetActionName (see page 24054)

GetDefaultActionName (see page 24055)

173.1.67.2.2 TEndUserAdapterLoginFormAction.ImplExecuteActionRequest Method

Executes the adapter action.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest calls the parent adapter's ExecuteLogin method. If an exception is raised, the exception message is added to the parent adapter's errors object.

See Also

IActionRequest (see page 935)

IActionResponse (see page 941)

ImplExecuteActionRequest (see page 24063)

ExecuteLogin (see page 24108)

Errors (see page 24182)

173.1.67.2.3 TEndUserAdapterLoginFormAction.ImplWebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

ImplWebEnabled returns the result of the parent adapter's CanExecuteLoginForm function.

See Also

CanExecuteLoginForm (see page 24106)

WebEnabled (see page 24071)

173.1.67.3 TEndUserAdapterLoginFormAction Properties

The properties of the TEndUserAdapterLoginFormAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 24444)	Suggests a display label for this adapter action.

173.1.67.3.1 TEndUserAdapterLoginFormAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

173.1.67.4 TEndUserAdapterLoginFormAction Events

The events of the TEndUserAdapterLoginFormAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 24445)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 24445)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24446)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24446)	Called before the adapter action response is generated.
	OnGetParams (see page 24447)	Specifies name/value pairs needed to execute this adapter action.

173.1.67.4.1 TEndUserAdapterLoginFormAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

173.1.67.4.2 TEndUserAdapterLoginFormAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.67.4.3 TEndUserAdapterLoginFormAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

173.1.67.4.4 TEndUserAdapterLoginFormAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.67.4.5 TEndUserAdapterLoginFormAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

173.1.68 TEndUserAdapterLogoutAction Class

TEndUserAdapterLogoutAction is an adapter action that logs the end-user out.

Class Hierarchy

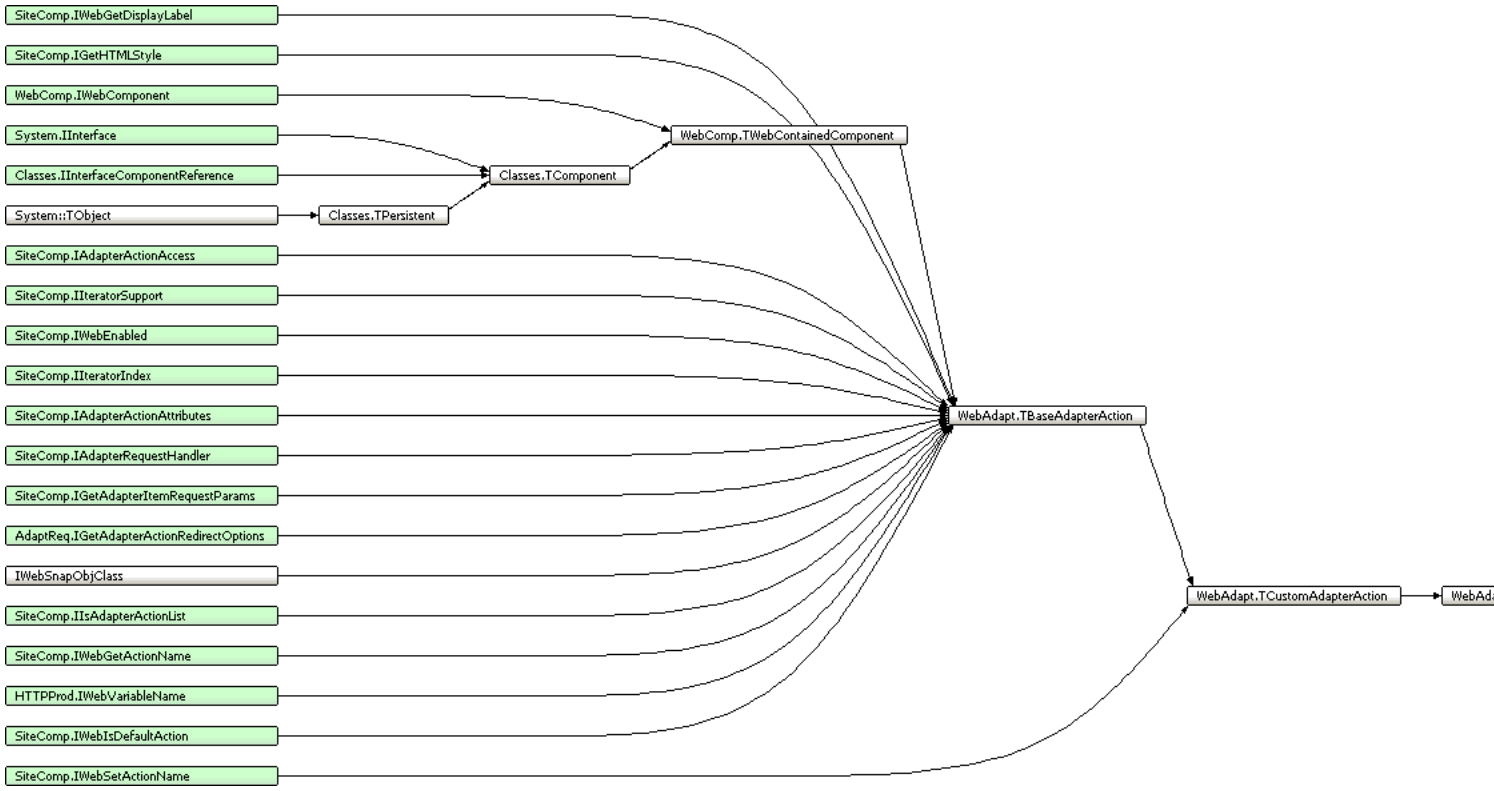

Pascal

```
TEndUserAdapterLogoutAction = class(TCustomEndUserAdapterAction);
```

C++

```
class TEndUserAdapterLogoutAction : public TCustomEndUserAdapterAction;
```

File

WebAdapt

Description

This action is typically executed when the end-user clicks on a "Logout" hyperlink displayed at the top of a web page.

173.1.68.1 TEndUserAdapterLogoutAction Members

The following tables list the members exposed by TEndUserAdapterLogoutAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.

	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 24454)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
V	ImplExecuteActionRequest (see page 24454)	Executes the adapter action.
V	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.

	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 24063)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 24196)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 24455)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Adapter (see page 24333)	Returns the parent adapter of this adapter action.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).

	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	ExecuteAccess (see page 24073)	Defines the rights required to execute this action.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RedirectOptions (see page 24074)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 24456)	Called after the adapter action has executed.

	OnAfterGetResponse (see page 24456)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24457)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24457)	Called before the adapter action response is generated.
	OnGetParams (see page 24458)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 24455)	Suggests a display label for this adapter action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.68.2 TEndUserAdapterLogoutAction Methods

The methods of the TEndUserAdapterLogoutAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 24454)	Gets the name used to reference this adapter action with server-side script.
	ImplExecuteActionRequest (see page 24454)	Executes the adapter action.
	ImplWebEnabled (see page 24455)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

173.1.68.2.1 TEndUserAdapterLogoutAction.GetDefaultActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultActionName();
```

Description

GetDefaultActionName returns "Logout".

See Also

GetActionName (see page 24054)

GetDefaultActionName (see page 24055)

173.1.68.2.2 TEndUserAdapterLogoutAction.ImplExecuteActionRequest Method

Executes the adapter action.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest calls the parent adapter's ExecuteLogout method. If an exception is raised, the exception message is added to the parent adapter's errors object.

See Also

IActionRequest (see page 935)

IActionResponse (see page 941)

ImplExecuteActionRequest (see page 24063)

ExecuteLogout (see page 24108)

Errors (see page 24182)

173.1.68.2.3 TEndUserAdapterLogoutAction.ImplWebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

ImplWebEnabled returns the result of the parent adapter's CanExecuteLogout function.

See Also

CanExecuteLogout (see page 24107)

WebEnabled (see page 24071)

173.1.68.3 TEndUserAdapterLogoutAction Properties

The properties of the TEndUserAdapterLogoutAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 24455)	Suggests a display label for this adapter action.

173.1.68.3.1 TEndUserAdapterLogoutAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side

script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

173.1.68.4 TEndUserAdapterLogoutAction Events

The events of the TEndUserAdapterLogoutAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 24456)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 24456)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24457)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24457)	Called before the adapter action response is generated.
	OnGetParams (see page 24458)	Specifies name/value pairs needed to execute this adapter action.

173.1.68.4.1 TEndUserAdapterLogoutAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

173.1.68.4.2 TEndUserAdapterLogoutAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by

DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.68.4.3 TEndUserAdapterLogoutAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

173.1.68.4.4 TEndUserAdapterLogoutAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.68.4.5 TEndUserAdapterLogoutAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

173.1.69 TEndUserSessionAdapter Class

TEndUserSessionAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.

Class Hierarchy

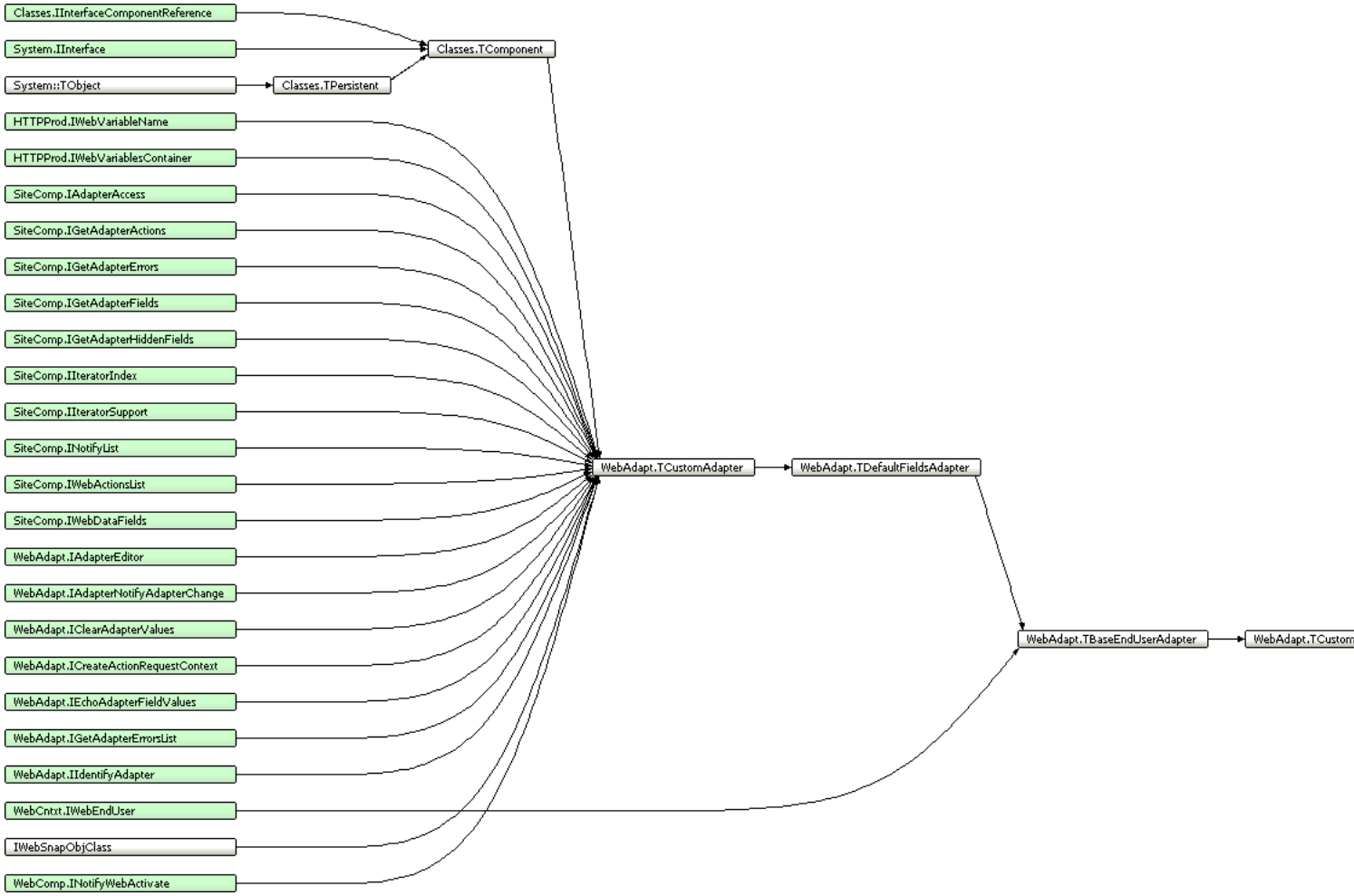

Pascal

```
TEndUserSessionAdapter = class(TCustomEndUserSessionAdapter);
```

C++

```
class TEndUserSessionAdapter : public TCustomEndUserSessionAdapter;
```

File

WebAdapt

Description

Add TEndUserSessionAdapter to a WebSnap application where users must log in before accessing Web pages. TEndUserSessionAdapter handles login and logout actions on behalf of the end user. TEndUserSessionAdapter uses the TSessionsService component to store information about the current end user. The WebSnap application must contain a TSessionsService component (or equivalent) in order for the TEndUserSessionAdapter to work properly.

TEndUserSessionAdapter supports two default Adapter fields: DisplayName and LoggedIn. These fields provide access to the user's name and login state. TEndUserSessionAdapter retrieves the values for these fields from a Web session, which is maintained by the TSessionsService component.

TEndUserSessionAdapter supports two default actions: LoginForm and LogoutOut. These actions are used to display a login form and to log the user out. Use the LoginPage (see page 24468) property to specify the name of the login page. The default implementation of the logout action terminates the session.

The EndUser script variable is used by server-side script to display end user information and execute end user actions. The following JavaScript displays the name of the logged in end user, if any.

```
<% if (EndUser.LoggedIn) { %>
<h1>Welcome <%= EndUser.DisplayName %> </h1>
<% } %>
```

The following JavaScript displays a login or logout hyperlink.

```
<%   if (EndUser.Logout.Enabled) { %>
  <a href="<%=EndUser.Logout.AsHREF%>">Logout</a>
<%   } %>
<%   if (EndUser.LoginForm.Enabled) { %>
  <a href=<%=EndUser.LoginForm.AsHREF%>>Login</a>
<%   } %>
```

TEndUserSessionAdapter also determines access rights for a user. If the WebSnap application contains a TWebUserList or equivalent, TEndUserAdapter (see page 24421) calls the TWebUserList component to check rights for a particular user ID. Implement the OnHasRights (see page 24471) event to check rights in an event handler.

See Also

TEndUserAdapter (see page 24421)

TSessionsService (see page 25582)

TLoginFormAdapter (see page 24500)

TWebUserList (see page 25625)

IWebEndUser (see page 24672)

Adding Login Support

173.1.69.1 TEndUserSessionAdapter Members

The following tables list the members exposed by TEndUserSessionAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.

	CanExecuteLoginForm (see page 24106)	Determines if there is a static login page.
	CanExecuteLogout (see page 24107)	Determines if the user is logged in.
	CanProduceLoginPage (see page 24107)	Determines whether there is a static login page or an automatically produced login page.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoOnHasRights (see page 24107)	Determines whether the user has view rights.
	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
	ExecuteLogin (see page 24108)	Starts the server-side login process.
	ExecuteLogin (see page 24342)	Logs in an end user.
	ExecuteLoginForm (see page 24108)	Creates the HTML login page.
	ExecuteLogout (see page 24108)	Performs server-side logout.
	ExecuteLogout (see page 24342)	Logs out the current user.
	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetDisplayName (see page 24109)	Retrieves the display name string.
	GetDisplayNameValue (see page 24109)	Gets the display name of the end user.
	GetDisplayNameValue (see page 24343)	Returns the name of the current end user.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetEndUserAdapter (see page 24109)	Gets the end-user adapter.

	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetLoggedIn (see page 24110)	Determines if the user is logged in
	GetLoggedInValue (see page 24110)	Determines if the user is logged in.
	GetLoggedInValue (see page 24343)	Indicates whether the current end user has logged in.
	GetLoginFormAction (see page 24110)	Gets the action component for the login form.
	GetLogoutAction (see page 24111)	Gets the action component for the logout form.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetUserID (see page 24111)	Gets the UserID from this TBaseEndUserAdapter (see page 24097).
	GetUserID (see page 24343)	Returns the user ID of the current user.
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasExecuteRights (see page 24111)	Determines if the user has execute rights for the TBaseEndUserAdapter (see page 24097).
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasModifyRights (see page 24112)	Determines if the user has modify rights for the TBaseEndUserAdapter (see page 24097).
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
	HasViewRights (see page 24112)	Determine if the user has view rights to the TBaseEndUserAdapter (see page 24097).
	ImplCanAddActionClass (see page 24113)	Determines if the given action class can be added.
	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 24113)	Determines if the given action class can be added.
	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 24113)	Lists supported end-user adapter actions.
	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
	ImplGetDisplayName (see page 24114)	Gets the display name for this TBaseEndUserAdapter (see page 24097).
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetEndUserAdapter (see page 24114)	Returns this TBaseEndUserAdapter (see page 24097).
	ImplGetFieldsList (see page 24114)	Obtains the list of all of the fields for the end-user adapter.
	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetLoggedIn (see page 24114)	Checks the value of the LoggedIn field.
	ImplGetLoginFormAction (see page 24115)	Gets the LoginForm action component for this TBaseEndUserAdapter (see page 24097).
	ImplGetLogoutAction (see page 24115)	Gets the Logout action component for this TBaseEndUserAdapter (see page 24097).

	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasExecuteRights (see page 24115)	Checks user rights and determines if the user has execute rights.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasModifyRights (see page 24116)	Checks user rights and determines if the user has modify rights.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplHasViewRights (see page 24116)	Checks user rights and determines if the user has view rights.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplLogIn (see page 24116)	Processes the login for the given user.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplProduceLoginPage (see page 24117)	Generate the HTML login page.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	LogIn (see page 24117)	Logs in a specified user.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceLoginPage (see page 24117)	Generates the login page.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UserList (see page 24118)	Gets the user list.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetDisplayName (see page 24673)	Returns the value of the DisplayName (see page 24678) property.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEndUserAdapter (see page 24674)	Returns the value of the EndUserAdapter (see page 24678) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
	GetLoggedIn (see page 24674)	Returns the value of the LoggedIn (see page 24678) property.
	GetLoginFormAction (see page 24674)	Returns the value of the LoginFormAction (see page 24679) property.
	GetLogoutAction (see page 24675)	Returns the value of the LogoutAction (see page 24680) property.

V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasExecuteRights (see page 24675)	Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasModifyRights (see page 24676)	Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	HasViewRights (see page 24676)	Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LogIn (see page 24677)	Logs in a specified user.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	ProduceLoginPage (see page 24677)	Generates the Web page for logging in users.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	DisplayName (see page 24678)	Indicates the name of the end user as it should be used for display in Web pages.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	EndUserAdapter (see page 24678)	Provides access to the end user adapter that implements this interface.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	LoggedIn (see page 24678)	Indicates whether the current user is logged in.
R	LoginFormAction (see page 24679)	Indicates the action component for the login form.
R	LogoutAction (see page 24680)	Indicates the action component for logging a user out.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	UserID (see page 24344)	Specifies the user ID of the current end user.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 24469)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24469)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 24470)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24470)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 24470)	Supplies name/value pairs needed to execute an adapter action.
	OnHasRights (see page 24471)	Determines if the user rights match the rights being queried.
	OnProduceLoginPage (see page 24471)	Allows implementation of a customized login page.

Published Properties

	Name	Description
	Actions (see page 24468)	Container for adapter actions components.
	Data (see page 24468)	Container for adapter field components.
	LoginPage (see page 24468)	Specifies the name of the page that acts as the login page.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.69.2 TEndUserSessionAdapter Properties

The properties of the TEndUserSessionAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 24468)	Container for adapter actions components.

	Data (see page 24468)	Container for adapter field components.
	LoginPage (see page 24468)	Specifies the name of the page that acts as the login page.

173.1.69.2.1 TEndUserSessionAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 23770)

173.1.69.2.2 TEndUserSessionAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 23778)

173.1.69.2.3 TEndUserSessionAdapter.LoginPage Property

Specifies the name of the page that acts as the login page.

Pascal

```
property LoginPage: string;
```

C++

```
__property AnsiString LoginPage;
```

Description

Use LoginPage to read or set the name of the login page for this TBaseEndUserAdapter. The name of the login page is the name of the Web page module that generates the login page.

173.1.69.3 TEndUserSessionAdapter Events

The events of the TEndUserSessionAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 24469)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24469)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 24470)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24470)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 24470)	Supplies name/value pairs needed to execute an adapter action.
	OnHasRights (see page 24471)	Determines if the user rights match the rights being queried.
	OnProduceLoginPage (see page 24471)	Allows implementation of a customized login page.

173.1.69.3.1 TEndUserSessionAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

173.1.69.3.2 TEndUserSessionAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.69.3.3 TEndUserSessionAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 24076)

OnAfterExecuteAction (see page 24184)

173.1.69.3.4 TEndUserSessionAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 24076)

OnAfterGetActionResponse (see page 24185)

173.1.69.3.5 TEndUserSessionAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 24077)

173.1.69.3.6 TEndUserSessionAdapter.OnHasRights Event

Determines if the user rights match the rights being queried.

Pascal

```
property OnHasRights: TEndUserRightsEvent;
```

C++

```
__property TEndUserRightsEvent OnHasRights;
```

Description

The OnHasRights event handler is called from DoOnHasRights to determine if the user rights match the rights being queried. A customized event handler should be written to determine the rights for your specific application.

See Also

DoOnHasRights (see page 24107)

ImplHasExecuteRights (see page 24115)

ImplHasModifyRights (see page 24116)

ImplHasViewRights (see page 24116)

173.1.69.3.7 TEndUserSessionAdapter.OnProduceLoginPage Event

Allows implementation of a customized login page.

Pascal

```
property OnProduceLoginPage: TEndUserHandledEvent;
```

C++

```
__property TEndUserHandledEvent OnProduceLoginPage;
```

Description

The OnProduceLoginPage event is triggered in the ImplProduceLoginPage. If a customized login page is desired for your application, a customized event handler can be written for this event.

See Also

ImplProduceLoginPage (see page 24117)

ProduceLoginPage (see page 24117)

CanProduceLoginPage (see page 24107)

173.1.70 TFieldValuesAdapter Class

TFieldValuesAdapter is an adapter that includes a single field value.

Class Hierarchy

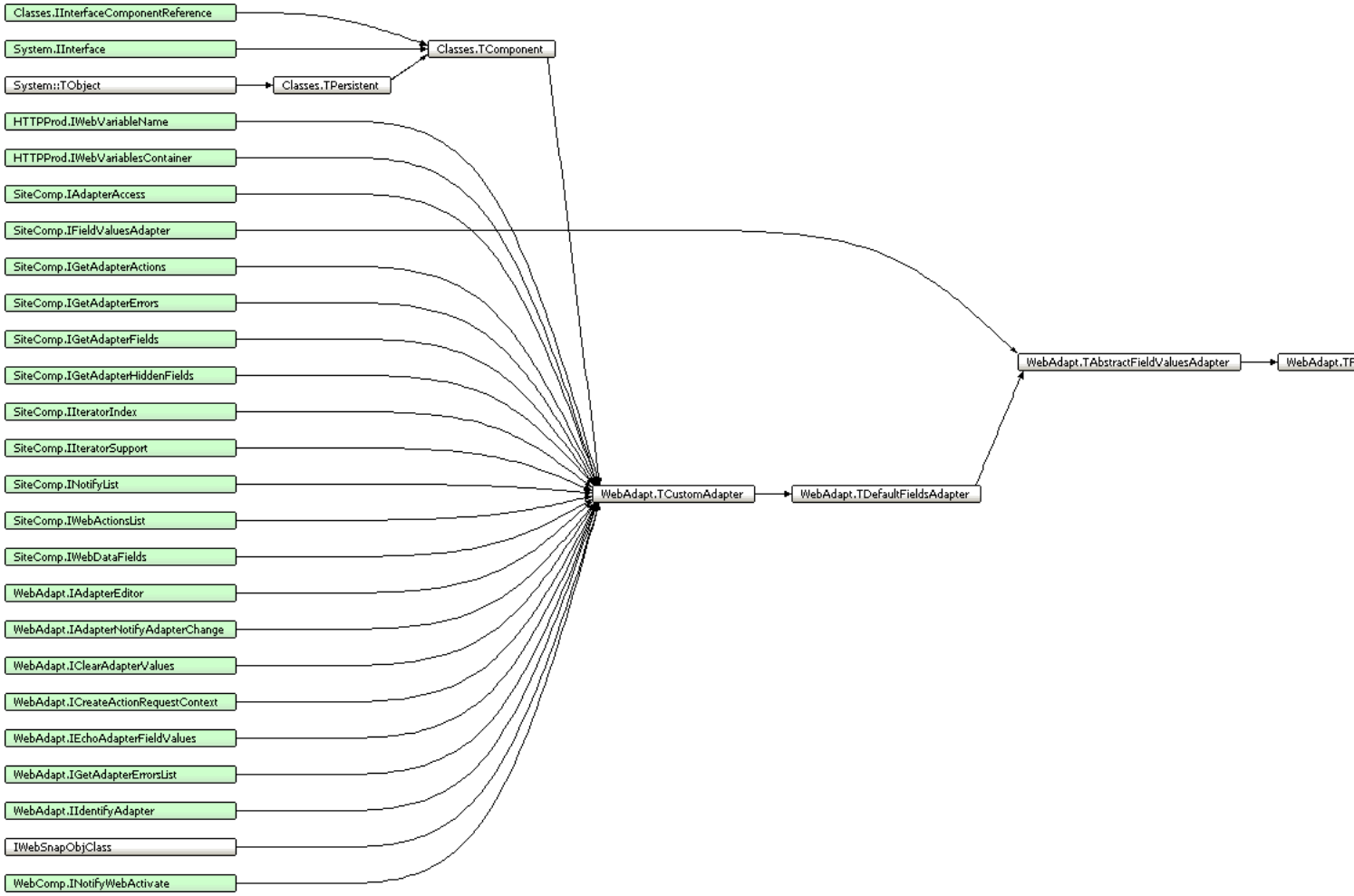

Pascal

```
TFieldValuesAdapter = class(TAbstractFieldValuesAdapter);
```

C++

```
class TFieldValuesAdapter : public TAbstractFieldValuesAdapter;
```

File

WebAdapt

Description

Use TFieldValuesAdapter to represent a single field value, which is available as a field with the name 'ValueField'. The single field is represented by an adapter field object that is of type TFieldValuesValueField (see page 24483), or one of its descendants. This adapter field object is supplied to the TFieldValuesAdapter constructor.

The 'ValueField' field can have multiple values, such as when it represents a value in the column of a table. For example, the following server-side script iterates over the values of the 'ValueField' field and adds them to a table:

```
<table border=1>
<%
   var e = new Enumerator(FieldValuesAdapter1)
```

```
    for (; !e.atEnd(); e.moveNext())
    {
%>
  <tr><td><%=e.item().ValueField%></td></tr>
<%
    }
%>
</table>
```

See Also

TStringsValuesList (see page 24540)

TPagedAdapter (see page 24525)

TDataSetValuesList (see page 8079)

TFieldValuesValueField (see page 24483)

173.1.70.1 TFieldValuesAdapter Members

The following tables list the members exposed by TFieldValuesAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.

	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetField (see page 23708)	Returns the object that represents the single field of this adapter.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValueFieldClass (see page 24480)	Returns the class of the object that represents this adapter's ValueField field.
	GetValueIndex (see page 23709)	Returns an index that identifies the current adapter field value.
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 23710)	Provides the underlying implementation for the CanAddFieldClass (see page 24150) method.
	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
	ImplEndIterator (see page 24480)	Provides the underlying implementation for the EndIterator (see page 24152) method.

	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetField (see page 24481)	Provides the underlying implementation for the GetField (see page 23708) method.
	ImplGetFieldsList (see page 23711)	Provides the underlying implementation for the GetFieldsList (see page 24159) method.
	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetValueIndex (see page 24481)	Provides the underlying implementation for the GetValueIndex (see page 23709) method.
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 24481)	Provides the underlying implementation for the NextIteration (see page 24173) method.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 24482)	Provides the underlying implementation for the StartIterator (see page 24176) method.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.

	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Field (see page 24483)	Indicates the object that represents this adapter's single field.
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 24184)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24185)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24186)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24186)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 24187)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.

	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 24479)	This is Create, a member of class TFieldValuesAdapter.
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetField (see page 19684)	This is GetField, a member of class IFieldValuesAdapter.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetValueIndex (see page 19684)	This is GetValueIndex, a member of class IFieldValuesAdapter.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 24181)	Actions is the container for adapter actions components.
R	Actions (see page 24402)	Container for adapter actions components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.

R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Data (see page 24181)	Data is the container for adapter field components.
R	Data (see page 24402)	Container for adapter field components.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.70.2 TFieldValuesAdapter.Create Constructor

Pascal

```
constructor Create(AOwner: TComponent; AField: TObject); virtual; reintroduce;
```

C++

```
virtual __fastcall TFieldValuesAdapter(TComponent * AOwner, TObject * AField);
```

Description

This is Create, a member of class TFieldValuesAdapter.

173.1.70.3 TFieldValuesAdapter Methods

The methods of the TFieldValuesAdapter class are listed here.

Protected Methods

	Name	Description
V	GetValueFieldClass (see page 24480)	Returns the class of the object that represents this adapter's ValueField field.
V	ImplEndIterator (see page 24480)	Provides the underlying implementation for the EndIterator (see page 24152) method.
V	ImplGetField (see page 24481)	Provides the underlying implementation for the GetField (see page 23708) method.

	ImplGetValueIndex (see page 24481)	Provides the underlying implementation for the GetValueIndex (see page 23709) method.
	ImplNextIteration (see page 24481)	Provides the underlying implementation for the NextIteration (see page 24173) method.
	ImplStartIterator (see page 24482)	Provides the underlying implementation for the StartIterator (see page 24176) method.

173.1.70.3.1 TFieldValuesAdapter.GetValueFieldClass Method

Returns the class of the object that represents this adapter's ValueField field.

Pascal

```
function GetValueFieldClass: TComponentClass; override;
```

C++

```
virtual __fastcall TComponentClass GetValueFieldClass();
```

Description

GetValueFieldClass returns the class reference for the type of the single adapter field (ValueField) that is the value of the Field (see page 24483) property. This is TFieldValuesValueField (see page 24483).

See Also

Field (see page 24483)

TFieldValuesValueField (see page 24483)

173.1.70.3.2 TFieldValuesAdapter.ImplEndIterator Method

Provides the underlying implementation for the EndIterator (see page 24152) method.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer); override;
```

C++

```
virtual __fastcall ImplEndIterator(void * OwnerData);
```

Description

The EndIterator (see page 24152) method calls ImplEndIterator to stop the process of iterating through the adapter's field's possible values. This provides the underlying support that allows server-side script to iterate through adapter field values.

OwnerData is a pointer to adapter-specific data that can affect the way it iterates over field values. Each adapter can interpret this value in its own way.

ImplEndIterator obtains an IWebGetFieldValues interface for the adapter field that is the value of the Field (see page 24483) property and calls its EndValueIterator method, passing on the value of the OwnerData parameter.

See Also

EndIterator (see page 24152)

ImplStartIterator (see page 24482)

ImplNextIteration (see page 24481)

EndValueIterator (see page 19761)

Field (see page 24483)

173.1.70.3.3 TFieldValuesAdapter.ImplGetField Method

Provides the underlying implementation for the GetField (see page 23708) method.

Pascal

```
function ImplGetField: TObject; override;
```

C++

```
virtual __fastcall TObject * ImplGetField();
```

Description

The GetField (see page 23708) method calls ImplGetField to obtain the object that represents this adapter's single field. This method returns the value of the Field (see page 24483) property.

See Also

GetField (see page 23708)

Field (see page 24483)

173.1.70.3.4 TFieldValuesAdapter.ImplGetValueIndex Method

Provides the underlying implementation for the GetValueIndex (see page 23709) method.

Pascal

```
function ImplGetValueIndex: Integer; override;
```

C++

```
virtual __fastcall int ImplGetValueIndex();
```

Description

The GetValueIndex (see page 23709) method calls ImplGetValueIndex to obtain the index of current value of the ValueField field. The first value has an index of 0, the second an index of 1, and so on.

This method returns the index of the current value, as indicated by the index that the GetIteratorIndex (see page 24160) method returns.

See Also

GetValueIndex (see page 23709)

GetIteratorIndex (see page 24160)

173.1.70.3.5 TFieldValuesAdapter.ImplNextIteration Method

Provides the underlying implementation for the NextIteration (see page 24173) method.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

The NextIteration (see page 24173) method calls ImplNextIteration when iterating over field values to determine whether there is another field value after the current one and if so, to change the current value to the next one.

OwnerData is a pointer to adapter-specific data that can affect the way it iterates over field values. On entry, this value is typically the same as the value returned by the last call to NextIteration (see page 24173) or to StartIterator (see page 24176).

ImplNextIteration returns true if it moves to the next field value, false if it can't change to a new field value (usually because the field is on its last value).

In TFieldValuesAdapter (see page 24471), ImplNextIteration obtains an IWebGetFieldValues interface for the adapter field that is the value of the Field (see page 24483) property and calls its NextValueIteration method, passing on the value of the OwnerData parameter.

See Also

NextIteration (see page 24173)

ImplStartIterator (see page 24482)

ImplEndIterator (see page 24480)

NextValueIteration (see page 19762)

173.1.70.3.6 TFieldValuesAdapter.ImplStartIterator Method

Provides the underlying implementation for the StartIterator (see page 24176) method.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

The StartIterator (see page 24176) method calls ImplStartIterator to start the process of iterating through the adapter's field's possible values. The return value indicates whether the adapter's field has any values over which to iterate.

OwnerData is a pointer to adapter-specific data that can affect the way it iterates over field values. Depending on the adapter field object, it may be supplied by the caller, or created by the adapter field object inside this method.

ImplStartIterator returns true if it positions the ValueField field on its first value, false if the adapter field can't supply a first value.

In TFieldValuesAdapter (see page 24471), ImplStartIterator obtains an IWebGetFieldValues interface for the adapter field that is the value of the Field (see page 24483) property and calls its StartValueIterator method, passing on the value of the OwnerData parameter.

See Also

StartIterator (see page 24176)

ImplEndIterator (see page 24480)

ImplNextIteration (see page 24481)

StartValueIterator (see page 19762)

Field (see page 24483)

173.1.70.4 TFieldValuesAdapter Properties

The properties of the TFieldValuesAdapter class are listed here.

Protected Properties

	Name	Description
R	Field (see page 24483)	Indicates the object that represents this adapter's single field.

173.1.70.4.1 TFieldValuesAdapter.Field Property

Indicates the object that represents this adapter's single field.

Pascal

```
property Field: TObject;
```

C++

```
__property TObject * Field;
```

Description

Read Field to access the object that represents this adapter's ValueField field. Field is an instance of TFieldValuesValueField (see page 24483) or one of its descendants.

The TFieldValuesAdapter (see page 24471) constructor supplies the value for this property.

See Also

TFieldValuesValueField (see page 24483)

create (see page 24479)

173.1.71 TFieldValuesValueField Class

TFieldValuesValueField is a adapter field child of TFieldValuesAdapter (see page 24471).

Class Hierarchy

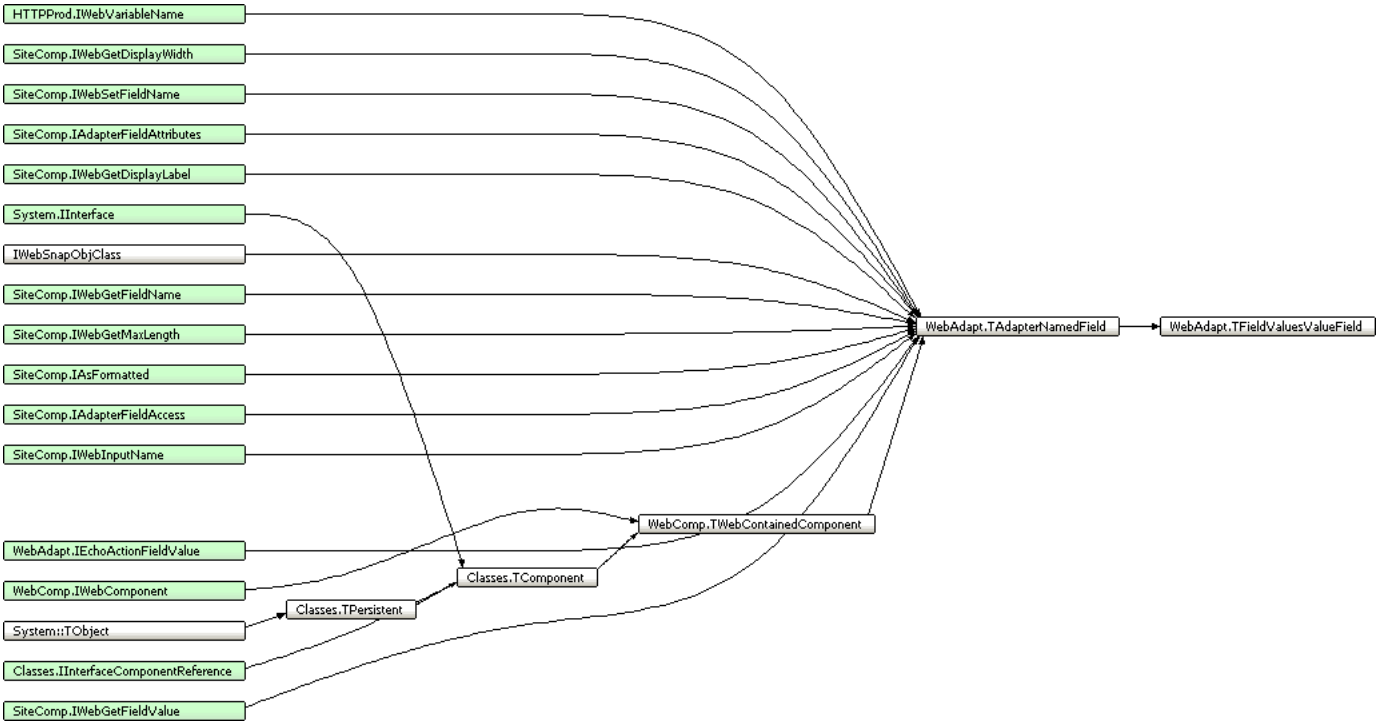

Pascal

```
TFieldValuesValueField = class(TAdapterNamedField);
```

C++

```
class TFieldValuesValueField : public TAdapterNamedField;
```

File

WebAdapt

Description

The parent adapter, TFieldValuesAdapter (see page 24471), is used to iterate over the values associated with an adapter field. TFieldValuesValueField provides the value and display text of the current item.

See Also

TFieldValuesAdapter (see page 24471)

TBaseAdapterMultiValueField (see page 24086)

173.1.71.1 TFieldValuesValueField Members

The following tables list the members exposed by TFieldValuesValueField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 24489)	Returns the name used to access this adapter field using server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 24489)	Returns text used to display a read-only representation of the value associated with this adapter field.
V	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
V	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 24490)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.

V	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	Field (see page 24490)	Returns the adapter field that is using TFieldValueAdapter.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.

	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.

	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.

R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.71.2 TFieldValuesValueField Methods

The methods of the TFieldValuesValueField class are listed here.

Protected Methods

	Name	Description
V	GetDefaultFieldName (see page 24489)	Returns the name used to access this adapter field using server-side script.
V	ImplAsFormatted (see page 24489)	Returns text used to display a read-only representation of the value associated with this adapter field.
V	ImplGetValue (see page 24490)	Returns the value associated with this adapter field.

173.1.71.2.1 TFieldValuesValueField.GetDefaultFieldName Method

Returns the name used to access this adapter field using server-side script.

Pascal

```
function GetDefaultFieldName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultFieldName();
```

Description

GetDefaultFieldName returns 'ValueField'. This allows server-side script to access this adapter field using the parent adapter's ValueField property.

See Also

ImplGetVariableName (see page 23939)

ImplSetFieldName (see page 23941)

GetDefaultFieldName (see page 23932)

173.1.71.2.2 TFieldValuesValueField.ImplAsFormatted Method

Returns text used to display a read-only representation of the value associated with this adapter field.

Pascal

```
function ImplAsFormatted: string; override;
```

C++

```
virtual __fastcall AnsiString ImplAsFormatted();
```

Description

ImplAsFormatted determines whether the adapter field using TFieldValuesAdapter (see page 24471) has a values list. If a values list is available, the values list is used to look up a value name to return. If there is no values list or no value name, the result of ImplGetValue (see page 24490) is returned. Server-side script accesses the result of this function using the DisplayText property of this adapter field. ImplAsFormatted overrides TAdapterNamedField (see page 23925)'s ImplAsFormatted method.

See Also

Value (see page 23945)

ImplAsFormatted (see page 23937)

GetAdapterValuesList (see page 24016)

ImplGetValue (see page 24490)

173.1.71.2.3 TFieldValuesValueField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns the result of GetCurrentValue. GetCurrentValue is implemented by the adapter field using TFieldValuesAdapter (see page 24471). Server-side script accesses the result of this function using the Value (see page 23945) property of this adapter field.

See Also

GetCurrentValue (see page 24094)

Value (see page 23945)

173.1.71.3 TFieldValuesValueField Properties

The properties of the TFieldValuesValueField class are listed here.

Protected Properties

	Name	Description
R	Field (see page 24490)	Returns the adapter field that is using TFieldValueAdapter.

173.1.71.3.1 TFieldValuesValueField.Field Property

Returns the adapter field that is using TFieldValueAdapter.

Pascal

```
property Field: TObject;
```

C++

```
__property TObject * Field;
```

Description

Adapter (see page 23943) fields use the TFieldValuesAdapter (see page 24471) component to iterate over the values associated with the adapter field. The Field property returns the adapter field that is using the TFieldValuesAdapter (see page 24471).

173.1.72 TImplementedAdapterAction Class

TImplementedAdapterAction is a base class for adapter actions.

Class Hierarchy

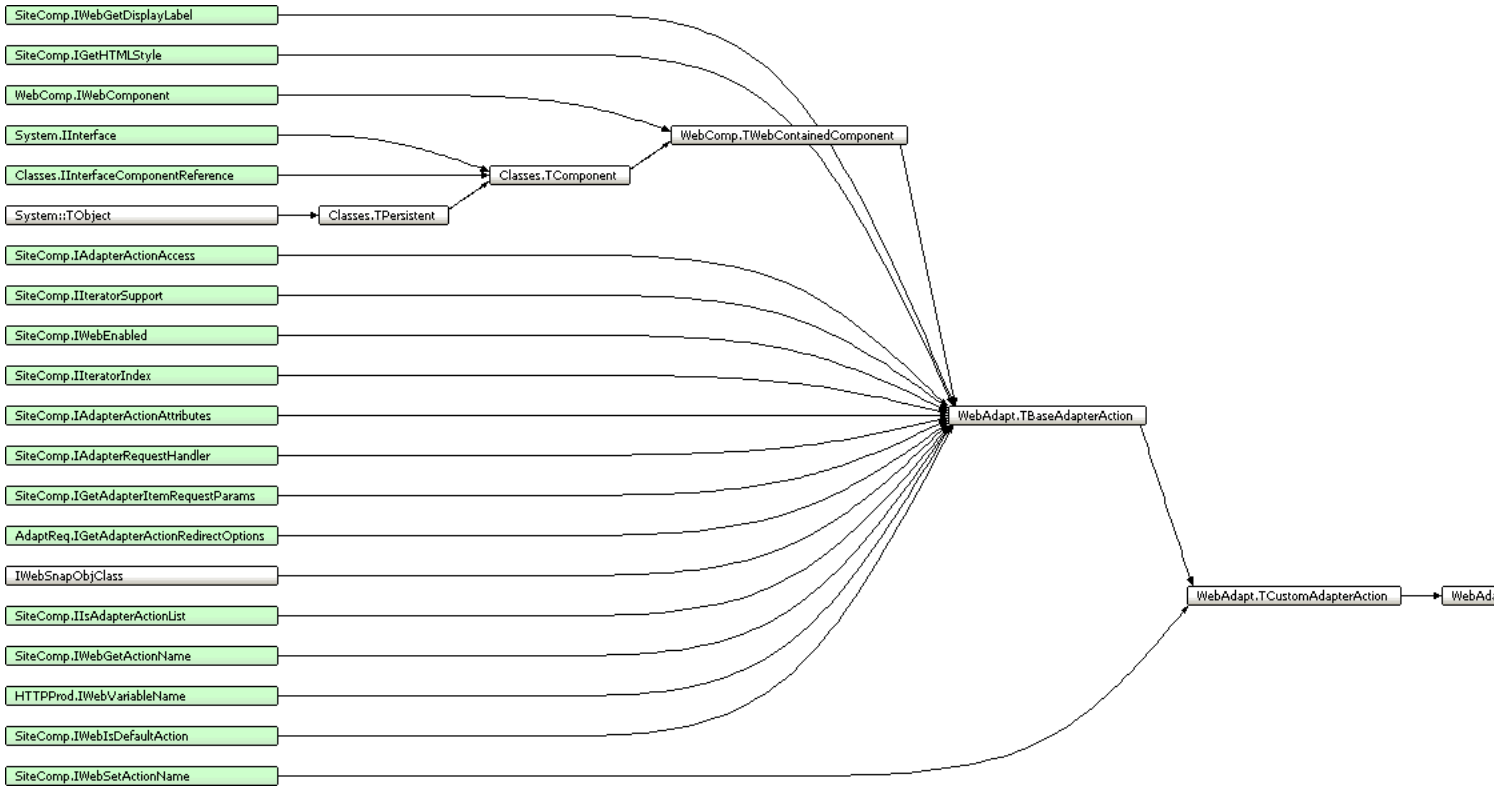

Pascal

```
TImplementedAdapterAction = class(TCustomAdapterAction);
```

C++

```
class TImplementedAdapterAction : public TCustomAdapterAction;
```

File

WebAdapt

Description

Derive from TImplementedAdapterAction to define an adapter action component that works with specific adapter component

types. Classes (see page 2685) derived from TImplementedAdapterAction can only be added to adapter components that allow the particular class. On the other hand, classes derived from TCustomAdapterAction (see page 24188) (such as TAdapter (see page 23713)) can be added to any adapter component class.

See Also

TCustomAdapterAction (see page 24188)

TAdapter (see page 23713)

TAdapterActions (see page 23738)

173.1.73 TLogicalPageList Class

Class Hierarchy

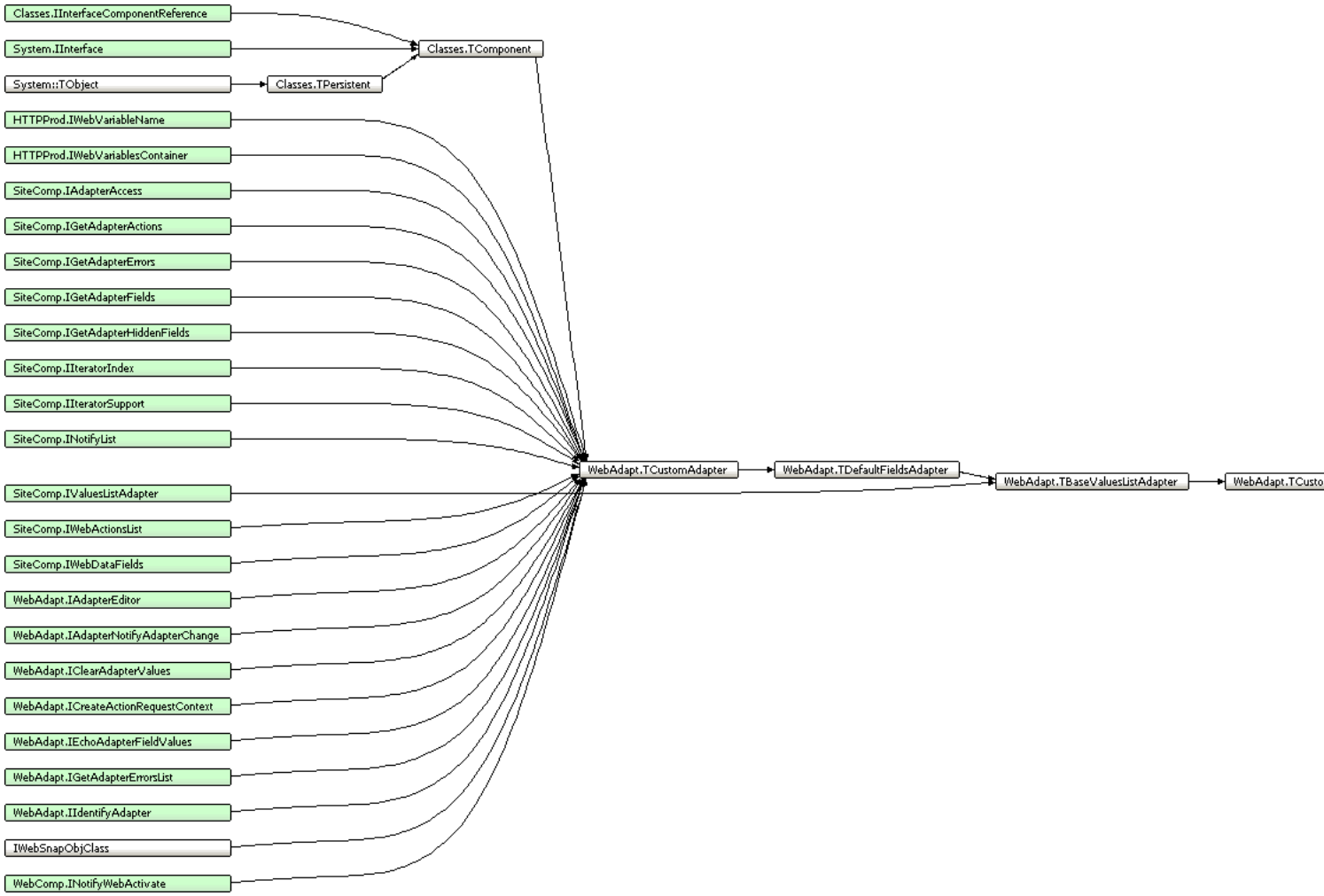

Pascal

```
TLogicalPageList = class(TCustomStringsValuesList);
```

C++

```
class TLogicalPageList : public TCustomStringsValuesList;
```

File

WebAdapt

Description

This is class WebAdapt.TLogicalPageList.

173.1.73.1 TLogicalPageList Members

The following tables list the members exposed by TLogicalPageList.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.

	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetListImage (see page 24128)	Returns an image associated with the current name/value pair.
	GetListImageOfValue (see page 24128)	Looks up an image for a value.
	GetListName (see page 24128)	Responsible returning the name of the name/value pair.
	GetListNameOfValue (see page 24128)	Looks up a name for a value.
	GetListValue (see page 24129)	Responsible for returning the value of the name/value pair.
	GetNameFieldClass (see page 24129)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
	GetNameFieldClass (see page 24387)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValueFieldClass (see page 24129)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetValueFieldClass (see page 24387)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 24130)	Returns the class type for this adapter field.
	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 24130)	Indicates whether a type of adapter field can be added to this adapter.
	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.

	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
	ImplEndIterator (see page 24388)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the records of the adapter.
	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 24130)	Adds default field components to this adapter.
	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetListImage (see page 24131)	Responsible returning an image associated with the current name/value pair.
	ImplGetListImageOfValue (see page 24131)	Looks up an image for a value.
	ImplGetListName (see page 24131)	Responsible returning the name of the name/value pair.
	ImplGetListName (see page 24388)	Responsible returning the name of the name/value pair.
	ImplGetListNameOfValue (see page 24132)	Looks up a name for a value.
	ImplGetListNameOfValue (see page 24388)	Looks up a name for a value.
	ImplGetListValue (see page 24132)	Responsible for returning the value of the name/value pair.
	ImplGetListValue (see page 24389)	Responsible for returning the value of the name/value pair.
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNextIteration (see page 24389)	Implements IteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplNotifyDeactivate (see page 24389)	Clears the iteration index.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	ImplStartIterator (see page 24390)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.

V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	PrepareStrings (see page 24390)	Customizes the content of the strings list.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
	StringsPrepared (see page 24391)	This flag indicates whether the strings list has been prepared or customized.
R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 24184)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24185)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24186)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24186)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 24187)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.
	OnPrepareStrings (see page 24391)	Implement OnPrepareStrings to customize the content of the strings list.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 24500)	Create is the constructor for TCustomStringsValuesList.
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
V	Destroy (see page 24386)	Destroy is the destructor for TCustomStringsList.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.

	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
	GetListImage (see page 19741)	Returns the image object associated with the current name/value pair.
	GetListImageOfValue (see page 19742)	Returns the image object associated with a specified value.
	GetListName (see page 19742)	Returns the name part of the current name/value pair.
	GetListNameOfValue (see page 19742)	Returns the name associated with a specified value.
	GetListValue (see page 19743)	Returns the value of the current name/value pair.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.

	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 24181)	Actions is the container for adapter actions components.
R	Actions (see page 24402)	Container for adapter actions components.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Data (see page 24181)	Data is the container for adapter field components.
R	Data (see page 24402)	Container for adapter field components.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Strings (see page 24390)	This string list defines the list of name/value pairs.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.73.2 TLogicalPageList.Create Constructor

Create is the constructor for TCustomStringsValuesList.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TLogicalPageList(TComponent * AOwner);
```

Description

Create creates the strings list.

See Also

Strings (see page 24390)

173.1.74 TLoginFormAdapter Class

TLoginFormAdapter contains the adapter fields and adapter actions used to generate a login form.

Class Hierarchy

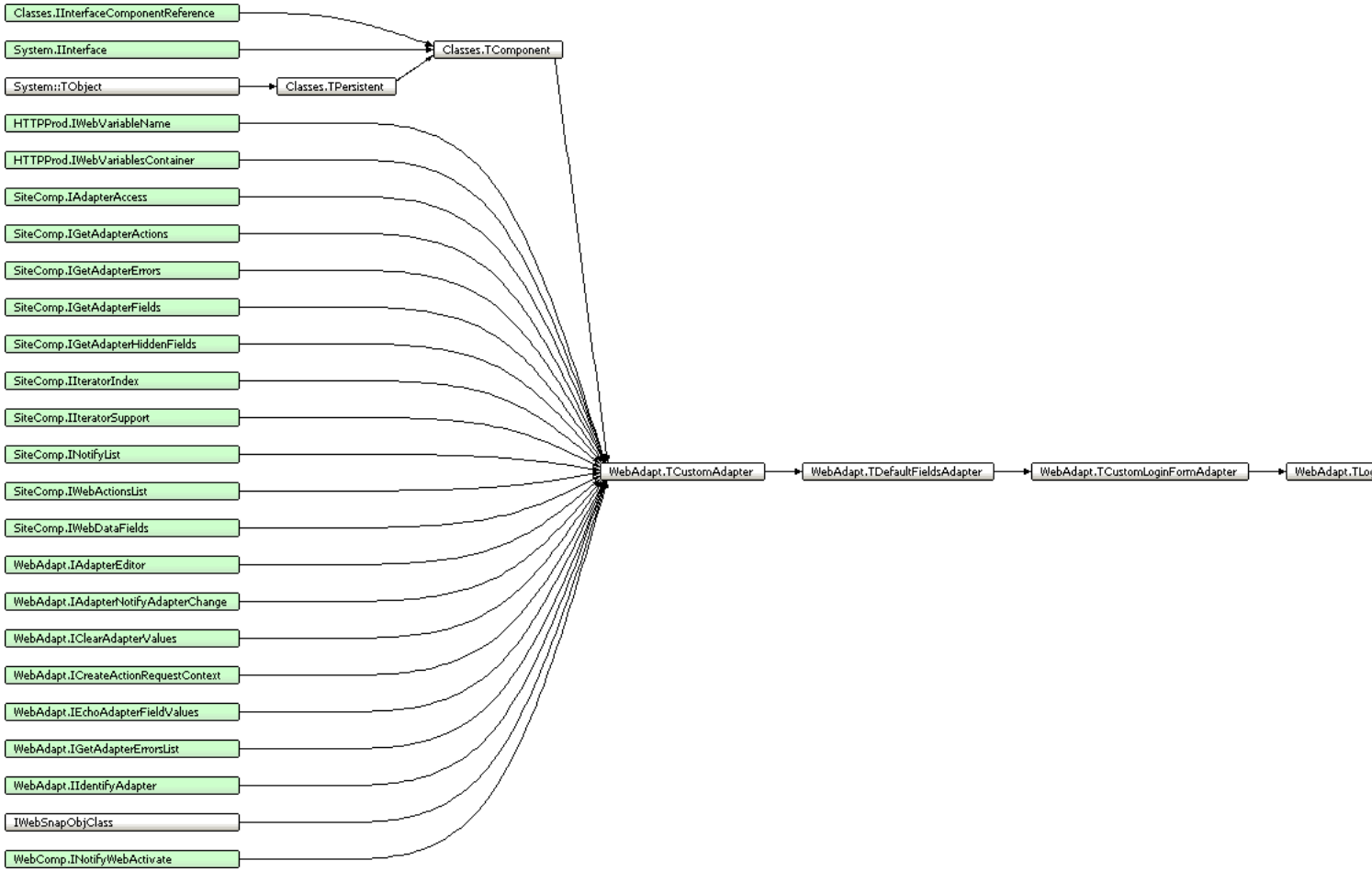

Pascal

```
TLoginFormAdapter = class(TCustomLoginFormAdapter);
```

C++

```
class TLoginFormAdapter : public TCustomLoginFormAdapter;
```

File

WebAdapt

Description

Use TLoginFormAdapter to generate a login form. TLoginFormAdapter has default fields for user name, password, and next page. The next page field lets the user select a Web page to see after logging in. TLoginFormAdapter has a single default action for logging in. If the WebSnap application also contains an end user adapter such as TEndUserAdapter (see page 24421) or TEndUserSessionAdapter (see page 24458), TLoginFormAdapter calls the end user adapter's Login method to log the user in. You can also implement an OnLogin (see page 24512) event handler to provide custom login support.

To build a simple login form

1. Create (see page 24146) a new Web page module that contains a TAdapterPageProducer component.

2. Add a TLoginFormAdapter component to that module.
3. Right click on the TAdapterPageProducer object and display the Web page editor.
4. In the Web page editor, add an adapter form, and to that form add an adapter field group and an adapter command group.
5. Select the field group, and set its Adapter property to the TLoginFormAdapter object.
6. Select the command group, and set its display component to the adapter field group.

You can embellish this simple form as you like. When you are done, connect the form to an end user adapter by setting the end user adapter's LoginPage property to the name of the page module.

To provide complete login support, add TWebUserList, TEndUserSessionAdapter (see page 24458), and TSessionsService components to the web application module to identify user names and passwords and to keep track of end user login state.

See Also

TEndUserAdapter (see page 24421)

TEndUserSessionAdapter (see page 24458)

TAdapterPageProducer (see page 19802)

TWebUserList (see page 25625)

TSessionsService (see page 25582)

TAdapterControlGroup (see page 25070)

TAdapterFieldGroup (see page 25150)

173.1.74.1 TLoginFormAdapter Members

The following tables list the members exposed by TLoginFormAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.

	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
	EndUser (see page 24353)	Returns the interface of the WebSnap application's end user adapter.
	ExecuteLogin (see page 24353)	Logs in the user after the user name and password have been validated.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
V	GetNextPage (see page 24354)	Returns the value of the NextPage (see page 24359) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.

	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddActionClass (see page 24354)	Provides the underlying implementation for the CanAddActionClass (see page 24150) method.
	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.
	ImplCanAddFieldClass (see page 24355)	Provides the underlying implementation for the CanAddFieldClass (see page 24150) method.
	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
	ImplGetActionsList (see page 24355)	Provides the underlying implementation for the GetActionsList (see page 24156) method.
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.
	ImplGetFieldsList (see page 24356)	Provides the underlying implementation for the GetFieldsList (see page 24159) method.
	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplNotifyDeactivate (see page 24356)	Provides the underlying implementation for the NotifyDeactivate (see page 24175) method.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.

V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	RaiseBlankPassword (see page 24357)	Raises an EAdapterFieldException (see page 23698) with a message that the supplied password is blank.
	RaiseBlankUserName (see page 24357)	Raises an EAdapterFieldException (see page 23698) with a message that the supplied user name is blank.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	UserList (see page 24357)	Returns the interface of the WebSnap application's Web user list.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidatePassword (see page 24358)	Validates the password that the user supplied to the login form.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateUserName (see page 24358)	Validates the name that the user supplied to the login form.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.

R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.

V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
	NextPage (see page 24359)	Names the Web page the user should see after successfully logging in.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Password (see page 24360)	Indicates the password that the user supplied to the login form.
R	UserName (see page 24360)	Indicates the name that the user supplied to the login form.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 24510)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24511)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 24511)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24511)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 24512)	Supplies name/value pairs needed to execute an adapter action.
	OnLogin (see page 24512)	Occurs after the password and user ID are validated.
	OnValidatePassword (see page 24513)	Occurs when the login form adapter must validate the password that the user supplied.
	OnValidateUserName (see page 24513)	Occurs when the login form adapter must validate the name that the user supplied.

Published Properties

	Name	Description
	Actions (see page 24509)	Container for adapter actions components.
	Data (see page 24509)	Container for adapter field components.
	Name (see page 2805)	Specifies the name of the component as referenced in code.

	PasswordRequired (see page 24509)	Indicates whether the user must supply a password to log in.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.74.2 TLoginFormAdapter Properties

The properties of the TLoginFormAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 24509)	Container for adapter actions components.
	Data (see page 24509)	Container for adapter field components.
	PasswordRequired (see page 24509)	Indicates whether the user must supply a password to log in.

173.1.74.2.1 TLoginFormAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 23770)

173.1.74.2.2 TLoginFormAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 23778)

173.1.74.2.3 TLoginFormAdapter.PasswordRequired Property

Indicates whether the user must supply a password to log in.

Pascal

```
property PasswordRequired: Boolean;
```

C++

```
__property Boolean PasswordRequired;
```

Description

Set PasswordRequired to indicate whether users must supply a password to log in. If PasswordRequired is true and the OnValidatePassword event handler does not handle the validation of the password, then the login form adapter checks that the value of Password is not an empty string. If Password is an empty string, the login form adapter raises an EAdapterFieldException exception when it tries to validate the password. If the OnValidatePassword event handler takes care of validating the password, PasswordRequired is ignored.

Note: The login form adapter validates the password (and hence checks PasswordRequired) when the WebSnap application executes this adapter's Login action.

See Also

Password (see page 24360)

OnValidatePassword (see page 24362)

TLoginFormAdapterLoginAction (see page 24514)

EAdapterFieldException (see page 23698)

173.1.74.3 TLoginFormAdapter Events

The events of the TLoginFormAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 24510)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24511)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 24511)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24511)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 24512)	Supplies name/value pairs needed to execute an adapter action.
	OnLogin (see page 24512)	Occurs after the password and user ID are validated.
	OnValidatePassword (see page 24513)	Occurs when the login form adapter must validate the password that the user supplied.
	OnValidateUserName (see page 24513)	Occurs when the login form adapter must validate the name that the user supplied.

173.1.74.3.1 TLoginFormAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

173.1.74.3.2 TLoginFormAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.74.3.3 TLoginFormAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 24076)

OnAfterExecuteAction (see page 24184)

173.1.74.3.4 TLoginFormAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 24076)

OnAfterGetActionResponse (see page 24185)

173.1.74.3.5 TLoginFormAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 24077)

173.1.74.3.6 TLoginFormAdapter.OnLogin Event

Occurs after the password and user ID are validated.

Pascal

```
property OnLogin: TLoginUserIDEvent;
```

C++

```
__property TLoginUserIDEvent OnLogin;
```

Description

Write an OnLogin event handler to respond when a user has successfully logged in. This event allows your application to record the user login in an application-specific manner.

When the login form adapter's Login action executes, the adapter first receives an OnValidatePassword and an OnValidateUserName event. After validating the password and user name, the login form adapter receives an OnLogin event so that it can log the user in. After the OnLogin event, if the WebSnap application includes an end user adapter, the login form adapter calls the end user adapter's Login method. Thus, it may not be necessary to write an OnLogin event handler if the application includes an end user adapter.

See Also

OnValidatePassword (see page 24362)

OnValidateUserName (see page 24362)

LogIn (see page 24677)

TWebUserList (see page 25625)

UserName (see page 24360)

173.1.74.3.7 TLoginFormAdapter.OnValidatePassword Event

Occurs when the login form adapter must validate the password that the user supplied.

Pascal

```
property OnValidatePassword: TValidateAdapterFieldEvent;
```

C++

```
__property TValidateAdapterFieldEvent OnValidatePassword;
```

Description

Write an OnValidatePassword event handler to validate the password that the user supplies to the login form. This event is especially important if the WebSnap application does not include a TWebUserList component for checking passwords against user names. The supplied Password is the value of the Value parameter on the event handler. You can check this against the user's name by reading the UserName property.

When the login form adapter's Login action executes, the adapter first receives an OnValidatePassword event to check the password. If the event handler sets the Handled parameter to true, indicating that the password is valid, the adapter goes on to validate the user name using an OnValidateUserName event, and then to log the user in using an OnLogin event (and, if the application includes an end user adapter, by calling the end user adapter's Login method). If the event handler finds the password to be invalid, it should raise an exception. If the event handler does not validate the supplied password (that is, if it sets the Handled parameter to false), the login form adapter validates the password by checking whether it is non-blank if the PasswordRequired property is true.

See Also

OnLogin (see page 24361)

OnValidateUserName (see page 24362)

Password (see page 24360)

UserName (see page 24360)

PasswordRequired (see page 24360)

173.1.74.3.8 TLoginFormAdapter.OnValidateUserName Event

Occurs when the login form adapter must validate the name that the user supplied.

Pascal

```
property OnValidateUserName: TValidateAdapterFieldEvent;
```

C++

```
__property TValidateAdapterFieldEvent OnValidateUserName;
```

Description

Write an OnValidateUserName event handler to validate the name that the user supplies when logging in. This event occurs after the OnValidatePassword event and before the OnLogin event.

If the event handler sets the Handled parameter to true, the login form adapter assumes that the user name (which is the value of the Value parameter and of the UserName property) is valid. In this case, the adapter checks whether the WebSnap application includes a TWebUserList component, and if so, checks the user name against the password. If the event handler sets the Handled parameter to false, the adapter validates the user name by checking whether it is nonempty. In this case, the adapter raises an EAdapterFieldException exception if the value of UserName is an empty string. If the OnValidateUserName

event handler finds the user name to be invalid, it should raise an exception.

See Also

OnLogin (see page 24361)

OnValidatePassword (see page 24362)

UserName (see page 24360)

EAdapterFieldException (see page 23698)

173.1.75 TLoginFormAdapterLoginAction Class

TLoginFormAdapterLoginAction executes the login command of a login form generated with TLoginFormAdapter (see page 24500).

Class Hierarchy

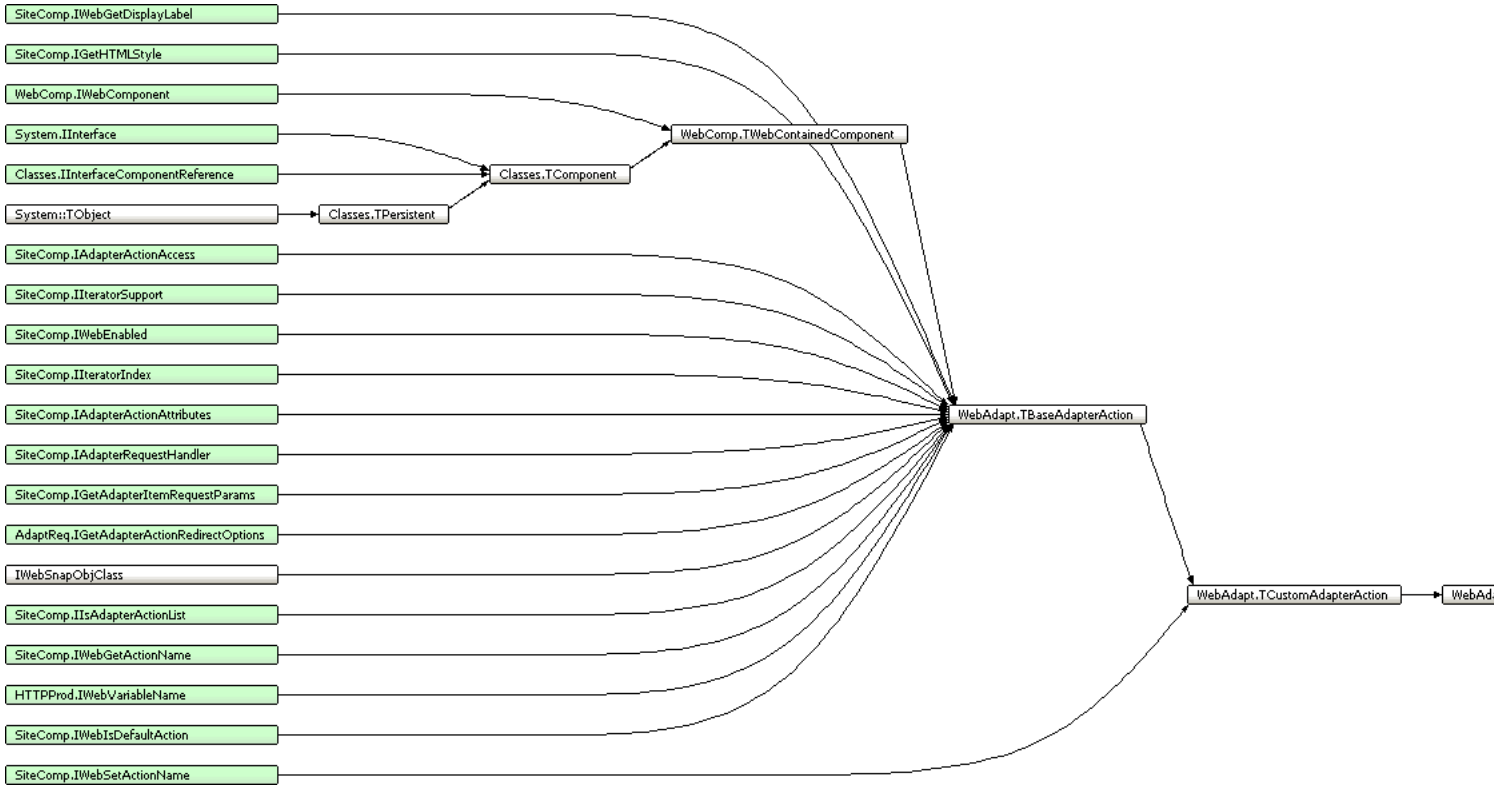

Pascal

```
TLoginFormAdapterLoginAction = class(TCustomLoginFormAdapterAction);
```

C++

```
class TLoginFormAdapterLoginAction : public TCustomLoginFormAdapterAction;
```

File

WebAdapt

Description

Add TLoginFormAdapterLoginAction to TLoginFormAdapter (see page 24500). The login form typically contains three fields: user name, password, and next page.

See Also

TCustomLoginFormAdapter (see page 24344)

TAdapterNextPageField (see page 23957)

TAdapterUserNameField (see page 24024)

TAdapterPasswordField (see page 23971)

173.1.75.1 TLoginFormAdapterLoginAction Members

The following tables list the members exposed by TLoginFormAdapterLoginAction.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckExecuteAccess (see page 24050)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 24050)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterExecuteActionRequest (see page 24051)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 24051)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 24052)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 24052)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 24053)	Implements IIteratorSupport's EndIterator method.
	ExecuteActionRequest (see page 24053)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 24054)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 24054)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 24055)	Provides the name/value pairs needed to execute this adapter action.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 24520)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 24055)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 24055)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 24056)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 24056)	Suggests an HTML display type for this adapter action.

	GetErrors (see page 24057)	This is GetErrors, a member of class TBaseAdapterAction.
	GetExecuteActionResponse (see page 24057)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 24057)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputStyle (see page 24058)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 24058)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 24058)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 24059)	Implements IIteratorIndex's GetIteratorIndex method.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSuccessOptions (see page 24059)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariableName (see page 24060)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 24060)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 24060)	Suggests a view mode for this adapter action.
	GetVisible (see page 24061)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 24061)	Returns the class type for this adapter field.
	HandleRequest (see page 24061)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 24062)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 24062)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 24062)	Implements IIteratorSupport's EndIterator (see page 24053) method.
	ImplExecuteActionRequest (see page 24521)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 24063)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 24195)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 24521)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 24064)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 24064)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 24065)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 24065)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 24065)	Implements IIteratorIndex's GetIteratorIndex (see page 24059) method.
	ImplGetSuccessOptions (see page 24066)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 24196)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 24066)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 24066)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 24067)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 24067)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 24067)	Implements IIteratorIndex's InIterator (see page 24069) method.
	ImplIsDefaultAction (see page 24068)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 24068)	Implements IIteratorSupport's NextIteration (see page 24070) method.
	ImplSetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 24068)	Implements IIteratorSupport's StartIterator (see page 24071) method.
	ImplWebEnabled (see page 24069)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 24197)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 24069)	Implements IIteratorIndex's InIterator method.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultAction (see page 24070)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24070)	Implements IIteratorSupport's NextIteration method.
V	NextPage (see page 24522)	Identifies the name of the next page to display after login.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24070)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 24197)	Sets the name that is used to reference this adapter action with server-side script.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDisplayLabel (see page 24071)	Set the value of the DisplayLabel (see page 24073) property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	StartIterator (see page 24071)	Implements IIteratorSupport's StartIterator method.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebEnabled (see page 24071)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 24072)	Provides access to the parent adapter component.
R	Adapter (see page 24369)	Returns the adapter that acts as a parent to this action.
R	Errors (see page 24073)	This is Errors, a member of class TBaseAdapterAction.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 24200)	Called to execute this action using parameters passed in the HTTP request.

	OnGetEnabled (see page 24200)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	CreateRequestContext (see page 19681)	Creates an object that describes the adapter action parameters and adapter field values.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 19692)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 19700)	Returns the preferred display style for an adapter field or adapter action.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFailureOptions (see page 964)	This is GetFailureOptions, a member of class IGetAdapterActionRedirectOptions.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputStyle (see page 19700)	Returns the preferred editing style for an adapter field.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsAdapterActionList (see page 19717)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetSuccessOptions (see page 964)	This is GetSuccessOptions, a member of class IGetAdapterActionRedirectOptions.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetViewMode (see page 19701)	Returns the preferred mode of an adapter field.
	GetVisible (see page 19652)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 19681)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 19651)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultAction (see page 19768)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	WebEnabled (see page 19752)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 19755)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 24072)	Identifies the name of this action when using server-side script.
	ActionName (see page 24198)	Identifies the name of this action when using server-side script.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	ExecuteAccess (see page 24073)	Defines the rights required to execute this action.
R	FailureOptions (see page 965)	This is FailureOptions, a member of class IGetAdapterActionRedirectOptions.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 19717)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 24074)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 24074)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RedirectOptions (see page 24074)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	SuccessOptions (see page 965)	This is SuccessOptions, a member of class IGetAdapterActionRedirectOptions.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	Visible (see page 19652)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 24523)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 24523)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24524)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24524)	Called before the adapter action response is generated.
	OnGetParams (see page 24525)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 24522)	Suggests a display label for this adapter action.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.75.2 TLoginFormAdapterLoginAction Methods

The methods of the TLoginFormAdapterLoginAction class are listed here.

Protected Methods

	Name	Description
V	GetDefaultActionName (see page 24520)	Gets the name used to reference this adapter action with server-side script.
V	ImplExecuteActionRequest (see page 24521)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetAdapterItemRequestParams (see page 24521)	Provides the name/value pairs needed to execute this adapter action.
V	NextPage (see page 24522)	Identifies the name of the next page to display after login.

173.1.75.2.1 TLoginFormAdapterLoginAction.GetDefaultActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultActionName();
```

Description

GetDefaultActionName returns "Login".

See Also

GetActionName (see page 24054)

GetDefaultActionName (see page 24055)

173.1.75.2.2 TLoginFormAdapterLoginAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest uses the user name and password passed in the HTTP request to log the user in. Exceptions are raised if the user name or password contain invalid values or if the user name/password is not known. If an exception is raised the exception message is added to the parent adapter's error list and the echo flag is set to true. If the login is successful then ImplExecuteActionRequest causes an HTTP redirect to the page specified in the next page field.

See Also

ImplExecuteActionRequest (see page 24063)

IActionRequest (see page 935)

IActionResponse (see page 941)

ValidatePassword (see page 24358)

ValidateUserName (see page 24358)

ValidateUser (see page 19785)

Errors (see page 24182)

EchoActionFieldValues (see page 24182)

173.1.75.2.3 TLoginFormAdapterLoginAction.ImplGetAdapterItemRequestParams Method

Provides the name/value pairs needed to execute this adapter action.

Pascal

```
procedure ImplGetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams); override;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(AnsiString AIdentifier,
IAdapterItemRequestParams Params);
```

Description

ImplGetAdapterItemRequestParams adds a name/value pair identifying the name of the next page that should be displayed after login. When the login action is executed, the name/value specified by ImplGetAdapterItemRequestParams will be used to generate the HTTP response unless the login form contains a next page field. In that case, the value entered by the end-user will be used to the generate the HTTP response.

See Also

NextPage (see page 24359)

RequestedPageName (see page 24658)

ImplGetAdapterItemRequestParams (see page 24063)

IAdapterItemRequestParams (see page 19673)

TAdapterNextPageField (see page 23957)

173.1.75.2.4 TLoginFormAdapterLoginAction.NextPage Method

Identifies the name of the next page to display after login.

Pascal

```
function NextPage: string; virtual;
```

C++

```
virtual __fastcall AnsiString NextPage();
```

Description

This value is used when generating the login form. NextPage returns the result of TCustomLoginFormAdapter.NextPage.

See Also

NextPage (see page 24359)

173.1.75.3 TLoginFormAdapterLoginAction Properties

The properties of the TLoginFormAdapterLoginAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 24522)	Suggests a display label for this adapter action.

173.1.75.3.1 TLoginFormAdapterLoginAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property AnsiString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 24064)

SetDisplayLabel (see page 24071)

173.1.75.4 TLoginFormAdapterLoginAction Events

The events of the TLoginFormAdapterLoginAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 24523)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 24523)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 24524)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 24524)	Called before the adapter action response is generated.
	OnGetParams (see page 24525)	Specifies name/value pairs needed to execute this adapter action.

173.1.75.4.1 TLoginFormAdapterLoginAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 24051)

OnAfterExecuteAction (see page 24184)

OnBeforeExecute (see page 24076)

173.1.75.4.2 TLoginFormAdapterLoginAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.75.4.3 TLoginFormAdapterLoginAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 24052)

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

173.1.75.4.4 TLoginFormAdapterLoginAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 24052)

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.75.4.5 TLoginFormAdapterLoginAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 24063)

173.1.76 TPagedAdapter Class

TPagedAdapter divides a large amount of data into several pages, each page displaying a portion of the data records.

Class Hierarchy

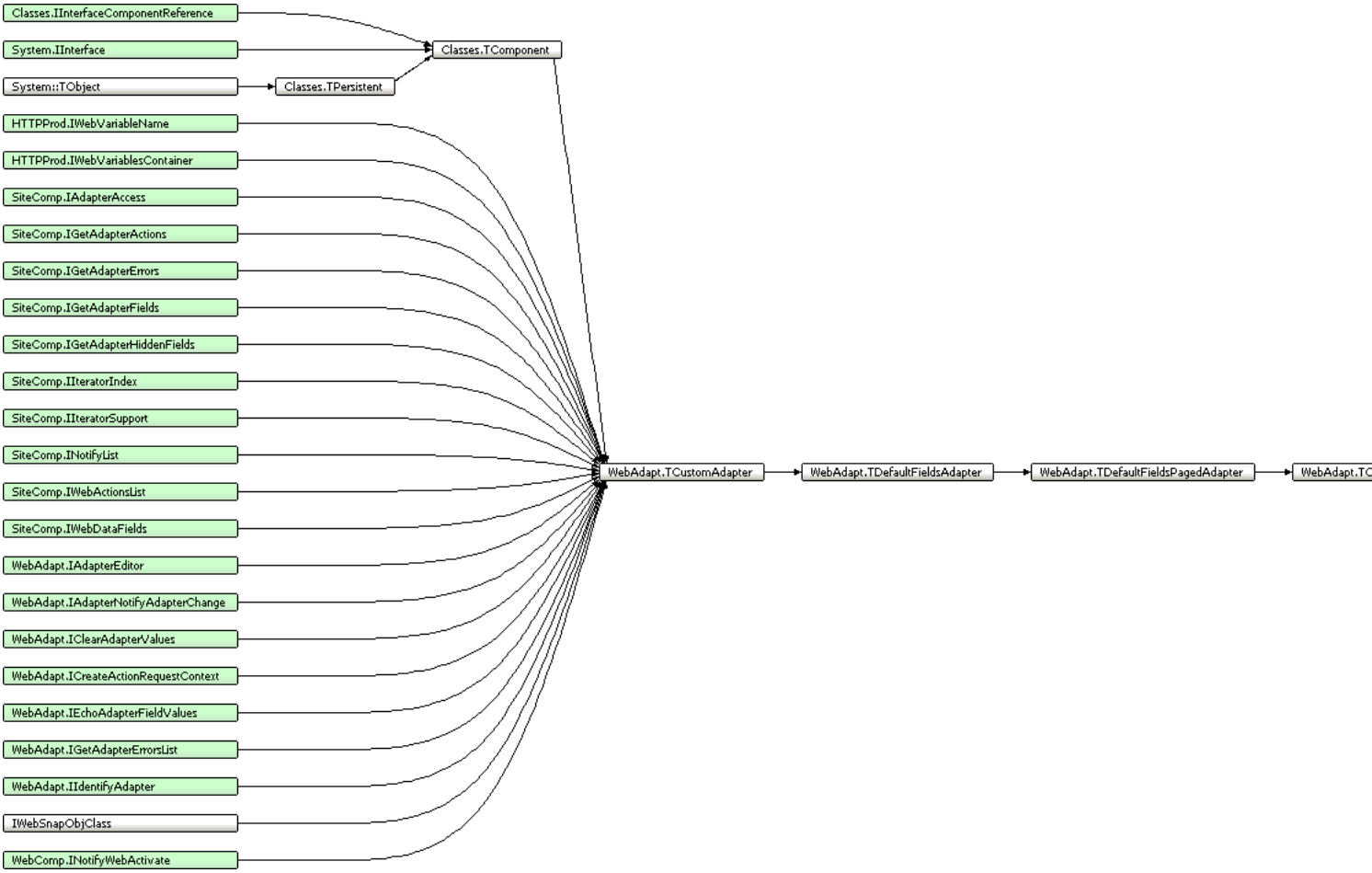

Pascal

```
TPagedAdapter = class(TCustomPagedAdapter);
```

C++

```
class TPagedAdapter : public TCustomPagedAdapter;
```

File

WebAdapt

Description

Use TPagedAdapter like TAdapter (see page 23713) to define a custom interface to application logic. Set the PageSize (see page 24534) property to indicate the number of rows that a page displays. When the PageSize (see page 24534) is not 0, the TPagedAdapter component automatically adds built-in actions called TAdapterNextPageAction (see page 23945), TAdapterPrevPageAction (see page 23979), and TAdapterGotoPageAction (see page 23849) for moving between pages. To generate HTML buttons for these actions, use TAdapterPageProducer.

The TDataSetAdapter component also supports paging.

173.1.76.1 TPagedAdapter Members

The following tables list the members exposed by TPagedAdapter.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultActions (see page 24411)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
V	EncodeActionParams (see page 24411)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
	EndOfPage (see page 24412)	Indicates whether the record index has gone past the last record in the current page.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
V	ExtractRequestParams (see page 24412)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.

	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetEOF (see page 24412)	Indicates whether the record position has passed the last record.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
	GetFirstRecord (see page 24413)	Set the record position to the first record.
	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetNextRecord (see page 24413)	Increments the record position.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPageCount (see page 24413)	Calculates the number of pages needed to display all adapter records.
	GetRecordCount (see page 24414)	Returns the total number of records.
	GetRecordIndex (see page 24414)	Returns the index of the current record.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddActionClass (see page 24415)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
	ImplEndIterator (see page 24415)	Implements IteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.
	ImplGetActionsList (see page 24415)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.

	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNextIteration (see page 24416)	Implements IteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplNotifyDeactivate (see page 24416)	Clears the current page variable.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	ImplStartIterator (see page 24416)	Implements IteratorSupport.StartIterator (see page 24176) to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetPageSize (see page 24417)	Updates the pagesize variable.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SkipToPage (see page 24417)	Skips records until the current record is within the current page.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.

	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.

	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	CurrentPage (see page 24418)	Sets or gets the current page index.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	PageCount (see page 24418)	Calculates the number of pages needed to display all adapter records.

R	RecordCount (see page 24419)	Returns the total number of records.
R	RecordIndex (see page 24419)	Returns the index of the current record.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 24536)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24536)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24536)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24537)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24537)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24537)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24538)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24538)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 24539)	Supplies name/value pairs needed to execute an adapter action.
	OnGetEOF (see page 24539)	Implement this event to indicate when the current record index has exceeded the total record count.
	OnGetFirstRecord (see page 24539)	Implement this method to initialize the iteration index to zero.
	OnGetNextRecord (see page 24539)	Implement this method to increment the iteration index.
	OnGetRecordCount (see page 24540)	Implement this event to identify the total number of records that can be iterated.
	OnGetRecordIndex (see page 24540)	Implement this event to identify the zero-based current record.

Published Properties

	Name	Description
	Actions (see page 24533)	Container for adapter actions components.
	Data (see page 24534)	Container for adapter field components.
	ModifyAccess (see page 24534)	This is ModifyAccess, a member of class TPagedAdapter.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	PageSize (see page 24534)	Sets or gets the page size variable.
	Tag (see page 2808)	Stores an integer value as part of a component.
	ViewAccess (see page 24535)	Identifies the rights required to view this adapter's fields.

173.1.76.2 TPagedAdapter Properties

The properties of the TPagedAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 24533)	Container for adapter actions components.
	Data (see page 24534)	Container for adapter field components.
	ModifyAccess (see page 24534)	This is ModifyAccess, a member of class TPagedAdapter.
	PageSize (see page 24534)	Sets or gets the page size variable.
	ViewAccess (see page 24535)	Identifies the rights required to view this adapter's fields.

173.1.76.2.1 TPagedAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 23770)

173.1.76.2.2 TPagedAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 23778)

173.1.76.2.3 TPagedAdapter.ModifyAccess Property

Pascal

```
property ModifyAccess: string;
```

C++

```
__property AnsiString ModifyAccess;
```

Description

This is ModifyAccess, a member of class TPagedAdapter.

173.1.76.2.4 TPagedAdapter.PageSize Property

Sets or gets the page size variable.

Pascal

```
property PageSize: Integer;
```

C++

```
__property int PageSize;
```

Description

If the page size variable is set to a value greater than zero and no actions have been explicitly added to the adapter then the

NextPage, PrevPage, and GotoPage are automatically added. If the page size variable is changed back to zero then the these actions are removed.

See Also

TAdapterNextPageAction (see page 23945)

TAdapterPrevPageAction (see page 23979)

TAdapterGotoPageAction (see page 23849)

173.1.76.2.5 TPagedAdapter.ViewAccess Property

Identifies the rights required to view this adapter's fields.

Pascal

```
property ViewAccess: string;
```

C++

```
__property AnsiString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view fields of this adapter. When server-side script accesses the CanView property, the ViewAccess property value is checked against the end user's rights.

See Also

TWebUserList (see page 25625)

HasViewAccess (see page 24162)

implhasviewaccess (see page 24169)

173.1.76.3 TPagedAdapter Events

The events of the TPagedAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 24536)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24536)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24536)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24537)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24537)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24537)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24538)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24538)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 24539)	Supplies name/value pairs needed to execute an adapter action.
	OnGetEOF (see page 24539)	Implement this event to indicate when the current record index has exceeded the total record count.
	OnGetFirstRecord (see page 24539)	Implement this method to initialize the iteration index to zero.
	OnGetNextRecord (see page 24539)	Implement this method to increment the iteration index.
	OnGetRecordCount (see page 24540)	Implement this event to identify the total number of records that can be iterated.
	OnGetRecordIndex (see page 24540)	Implement this event to identify the zero-based current record.

173.1.76.3.1 TPagedAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 24075)

OnBeforeExecuteAction (see page 24186)

173.1.76.3.2 TPagedAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 24075)

OnBeforeGetActionResponse (see page 24186)

173.1.76.3.3 TPagedAdapter.OnAfterUpdateFields Event

Occurs after values in the HTTP request are used to update adapter fields.

Pascal

```
property OnAfterUpdateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterUpdateFields;
```

Description

Implement this event handler to perform custom updates. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record update errors.

See Also

UpdateFieldsUpdateValues (see page 24178)

OnBeforeUpdateFields (see page 24187)

Errors (see page 24182)

173.1.76.3.4 TPagedAdapter.OnAfterValidateFields Event

Occurs after values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnAfterValidateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record validation errors.

See Also

UpdateFieldsValidateValues (see page 24178)

OnBeforeValidateFields (see page 24187)

Errors (see page 24182)

173.1.76.3.5 TPagedAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 24076)

OnAfterExecuteAction (see page 24184)

173.1.76.3.6 TPagedAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 24076)

OnAfterGetActionResponse (see page 24185)

173.1.76.3.7 TPagedAdapter.OnBeforeUpdateFields Event

Occurs before values in the HTTP request are used to update adapter fields.

Pascal

```
property OnBeforeUpdateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeUpdateFields;
```

Description

Implement this event handler to perform custom updates of adapter fields. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further updates. Add update errors to the adapter's errors object.

See Also

UpdateFieldsUpdateValues (see page 24178)

OnAfterUpdateFields (see page 24185)

Errors (see page 24182)

173.1.76.3.8 TPagedAdapter.OnBeforeValidateFields Event

Occurs before values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnBeforeValidateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further validation. Add validation errors to the adapter's errors object.

See Also

UpdateFieldsValidateValues (see page 24178)

OnAfterValidateFields (see page 24186)

Errors (see page 24182)

173.1.76.3.9 TPagedAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 24077)

173.1.76.3.10 TPagedAdapter.OnGetEOF Event

Implement this event to indicate when the current record index has exceeded the total record count.

Pascal

```
property OnGetEOF: TGetEOFEvent;
```

C++

```
__property TGetEOFEvent OnGetEOF;
```

Description

OnGetEOF is called to determine when to stop the iteration. If OnGetEOF is not implemented, the result of the OnGetRecordCount event handler will be compared to the result of the OnGetRecordIndex event handler.

173.1.76.3.11 TPagedAdapter.OnGetFirstRecord Event

Implement this method to initialize the iteration index to zero.

Pascal

```
property OnGetFirstRecord: TGetRecordEvent;
```

C++

```
__property TGetRecordEvent OnGetFirstRecord;
```

Description

This method is called when the iteration is started. If there are no records then set the EOF flag.

173.1.76.3.12 TPagedAdapter.OnGetNextRecord Event

Implement this method to increment the iteration index.

Pascal

```
property OnGetNextRecord: TGetRecordEvent;
```

C++

```
__property TGetRecordEvent OnGetNextRecord;
```

Description

This method is called when the next record is requested. This method is not a next then set the EOF flag.

173.1.76.3.13 TPagedAdapter.OnGetRecordCount Event

Implement this event to identify the total number of records that can be iterated.

Pascal

```
property OnGetRecordCount: TGetRecordCountEvent;
```

C++

```
__property TGetRecordCountEvent OnGetRecordCount;
```

Description

OnGetRecordCount is called to calculate the number of pages.

173.1.76.3.14 TPagedAdapter.OnGetRecordIndex Event

Implement this event to identify the zero-based current record.

Pascal

```
property OnGetRecordIndex: TGetRecordIndexEvent;
```

C++

```
__property TGetRecordIndexEvent OnGetRecordIndex;
```

Description

OnGetRecordIndex is called to calculate the current page.

173.1.77 TStringsValuesList Class

TStringsValuesList is an adapter component that provides a list of name/value pairs defined by items in a TStrings property.

Class Hierarchy

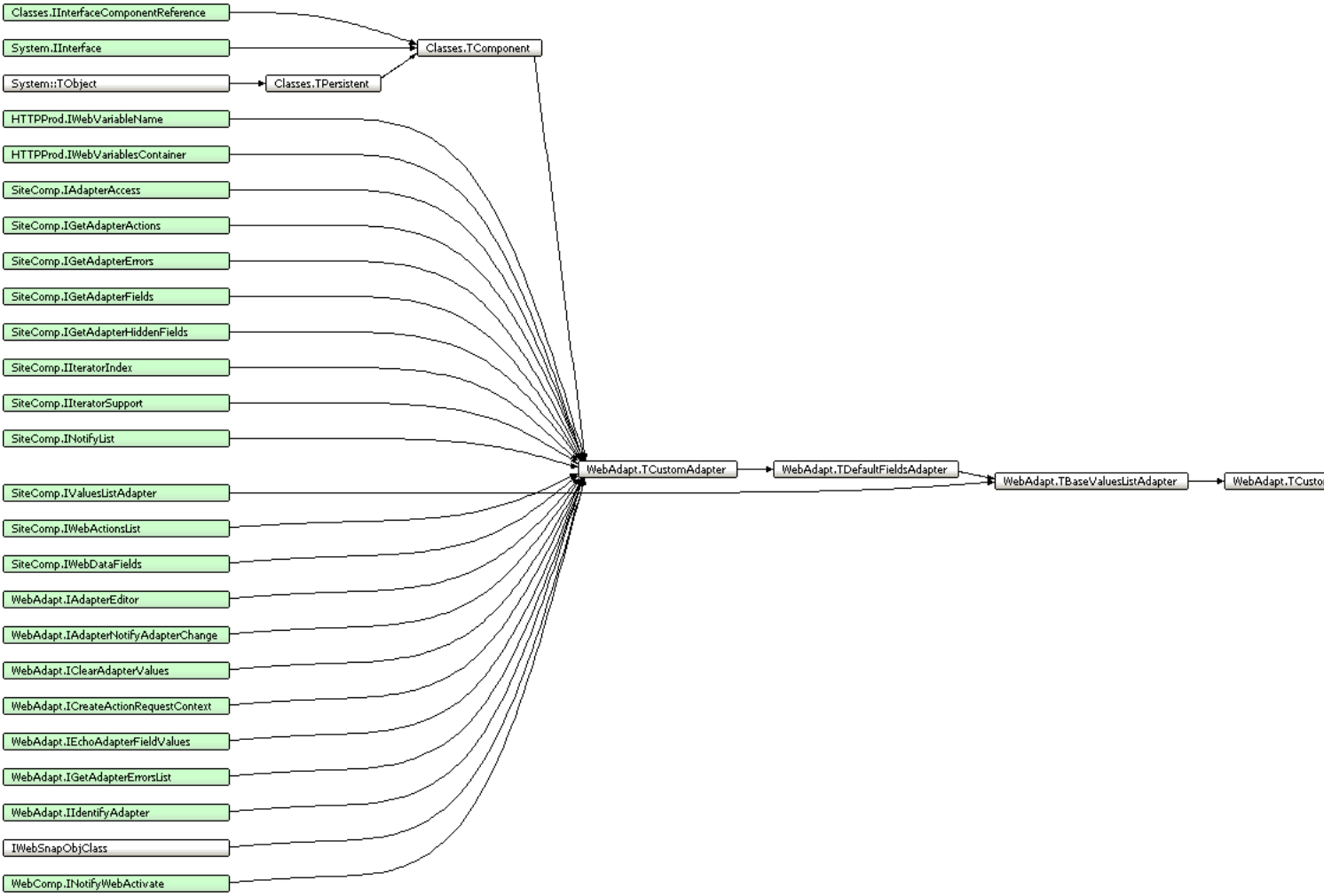

Pascal

```
TStringsValuesList = class(TCustomStringsValuesList);
```

C++

```
class TStringsValuesList : public TCustomStringsValuesList;
```

File

WebAdapt

Description

TStringsValuesList defines a list of possible values for an adapter field by assigning the TStringsValuesList component to the ValuesList property of the adapter field. Typically, the values are used to populate HTML elements, such as a select list or radio buttons, when generating HTML to edit an adapter field. The TAdapterPageProducer will automatically generate appropriate HTML/JavaScript for an editable field when the ValuesList is set.

173.1.77.1 TStringsValuesList Members

The following tables list the members exposed by TStringsValuesList.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAdapterModeHiddenField (see page 24149)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 24399)	Adds default action components to this adapter.
V	AddDefaultFields (see page 24399)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 24149)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 24149)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddActionClass (see page 24150)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 24150)	Indicates whether a type of adapter field can be added to this adapter.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 24150)	This is CheckModifyAccess, a member of class TCustomAdapter.
	ClearAdapterValues (see page 24151)	Clears adapter state information.
	ClearErrors (see page 24151)	Clears the content of the adapter's Errors (see page 24182) property.
	CreateActionRequestContext (see page 24151)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 24152)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 24400)	Notifies other components of changes to the list of default actions or fields.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EncodeActionParams (see page 24152)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 24152)	Implements IIteratorSupport's EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 24153)	Reads name/value pairs from the HTTP request.
	FindAction (see page 24153)	Returns an action component with a matching action name.
	FindField (see page 24154)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 24154)	Finds an action with a matching variable name.
	FindNamedField (see page 24154)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 24155)	Finds a field or action with a matching variable name.
	FindVariable (see page 24155)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 24155)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 24156)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 24400)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 24156)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 24156)	Returns the adapter actions container object.
	GetAdapterErrors (see page 24157)	This is GetAdapterErrors, a member of class TCustomAdapter.
	GetAdapterErrorsList (see page 24157)	<insert text here>
	GetAdapterFields (see page 24157)	Returns the adapter fields container object.

	GetAdapterHiddenFields (see page 24157)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 24158)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24158)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 24158)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 24400)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 24159)	Returns the value of the echo flag.
	GetFieldsList (see page 24159)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 24159)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetIteratorIndex (see page 24160)	Implements IIteratorIndex's GetIteratorIndex method to support iterating over the data records of the adapter.
	GetListImage (see page 24128)	Returns an image associated with the current name/value pair.
	GetListImageOfValue (see page 24128)	Looks up an image for a value.
	GetListName (see page 24128)	Responsible returning the name of the name/value pair.
	GetListNameOfValue (see page 24128)	Looks up a name for a value.
	GetListValue (see page 24129)	Responsible for returning the value of the name/value pair.
	GetNameFieldClass (see page 24129)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
	GetNameFieldClass (see page 24387)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValueFieldClass (see page 24129)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetValueFieldClass (see page 24387)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetVariable (see page 24160)	Returns an action or field.
	GetVariableCount (see page 24161)	Returns the total count of actions and fields.
	GetVariableName (see page 24161)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 24161)	This is GetVisibleActions, a member of class TCustomAdapter.
	GetVisibleFields (see page 24161)	This is GetVisibleFields, a member of class TCustomAdapter.
	GetWebSnapObjClass (see page 24130)	Returns the class type for this adapter field.
	GetWebSnapObjClass (see page 24162)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 24162)	This is HasExecuteAccess, a member of class TCustomAdapter.
	HasModifyAccess (see page 24162)	This is HasModifyAccess, a member of class TCustomAdapter.
	HasViewAccess (see page 24162)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 24163)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 24130)	Indicates whether a type of adapter field can be added to this adapter.
	ImplCanAddFieldClass (see page 24163)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 24163)	Clears adapter state information.
	ImplEndIterator (see page 24164)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the data records of the adapter.
	ImplEndIterator (see page 24388)	Implements IIteratorSupport's EndIterator (see page 24152) method to support iterating over the records of the adapter.
	ImplFindVariable (see page 24164)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 24164)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 24165)	Provides a list of action types that can be added to this adapter.

	ImplGetEchoActionFieldValues (see page 24165)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 24130)	Adds default field components to this adapter.
	ImplGetFieldsList (see page 24165)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 24166)	Implements IIteratorIndex's GetIteratorIndex (see page 24160) method to support iterating over the data records of the adapter.
	ImplGetListImage (see page 24131)	Responsible returning an image associated with the current name/value pair.
	ImplGetListImageOfValue (see page 24131)	Looks up an image for a value.
	ImplGetListName (see page 24131)	Responsible returning the name of the name/value pair.
	ImplGetListName (see page 24388)	Responsible returning the name of the name/value pair.
	ImplGetListNameOfValue (see page 24132)	Looks up a name for a value.
	ImplGetListNameOfValue (see page 24388)	Looks up a name for a value.
	ImplGetListValue (see page 24132)	Responsible for returning the value of the name/value pair.
	ImplGetListValue (see page 24389)	Responsible for returning the value of the name/value pair.
	ImplGetVariable (see page 24166)	Returns an action or field.
	ImplGetVariableCount (see page 24167)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 24167)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 24167)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 24401)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 24167)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 24401)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 24168)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 24168)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 24169)	This is ImplHasViewAccess, a member of class TCustomAdapter.
	ImplInIterator (see page 24169)	Implements IIteratorIndex's InIterator (see page 24172) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 24169)	Implements IIteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNextIteration (see page 24389)	Implements IteratorSupport's NextIteration (see page 24173) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 24170)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 24170)	Called when the web module is deactivated.
	ImplNotifyDeactivate (see page 24389)	Clears the iteration index.
	ImplSetActions (see page 24170)	Initializes the Actions (see page 24181) property.
	ImplSetActions (see page 24401)	Initializes the Actions (see page 24402) property.
	ImplSetData (see page 24171)	Initializes the Data (see page 24181) property.
	ImplSetData (see page 24402)	Initializes the Data (see page 24402) property.
	ImplSetEchoActionFieldValues (see page 24171)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 24171)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	ImplStartIterator (see page 24390)	Implements IIteratorSupport's StartIterator (see page 24176) method to support iterating over the records of the adapter.
	InIterator (see page 24172)	Implements IIteratorIndex's InIterator method to support iterating over the data records of the adapter.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsActionInUse (see page 24172)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 24172)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 24173)	This is ListHiddenFields, a member of class TCustomAdapter.
	ListHiddenRecordFields (see page 24173)	Specifies the HTML form hidden fields needed by the adapter.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	NextIteration (see page 24173)	Implements IIteratorSupport's NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 24174)	Clears references to deleted components.
	NotifyActivate (see page 24174)	Called when the web module is activated.
	NotifyAdapterChange (see page 24174)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 24175)	Called when the web module is deactivated.

	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	PrepareStrings (see page 24390)	Customizes the content of the strings list.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActions (see page 24175)	Initializes the Actions (see page 24181) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetData (see page 24175)	Initializes the Data (see page 24181) property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValues (see page 24176)	Sets the echo flag for this adapter's fields.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	StartIterator (see page 24176)	Implements IIteratorSupport's StartIterator method to support iterating over the records of the adapter.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateFieldsExecuteValues (see page 24177)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 24177)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 24178)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 24178)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 24179)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 24180)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 24180)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WebComponentsChange (see page 24180)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	NotifyList (see page 24183)	Keeps track of components that are to be notified when the adapter is changed.
	StringsPrepared (see page 24391)	This flag indicates whether the strings list has been prepared or customized.
R	VisibleActions (see page 24184)	This is VisibleActions, a member of class TCustomAdapter.
R	VisibleFields (see page 24184)	This is VisibleFields, a member of class TCustomAdapter.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 24184)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 24185)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 24185)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 24186)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 24186)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 24186)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 24187)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 24187)	Occurs before values in the HTTP request are validated by the adapter's fields.

	OnGetActionParams (see page 24187)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 24188)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24146)	Constructor for TCustomAdapter (see page 24139).
V	Create (see page 24386)	Create is the constructor for TCustomStringsValuesList (see page 24378).
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24146)	Destructor for TCustomAdapter (see page 24139).
V	Destroy (see page 24386)	Destroy is the destructor for TCustomStringsList.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 19686)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 19688)	This is GetAdapterErrors, a member of class IGetAdapterErrors.

	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 19689)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 19690)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 19690)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIteratorIndex (see page 19721)	Returns the value of the IteratorIndex (see page 19721) property.
	GetListImage (see page 19741)	Returns the image object associated with the current name/value pair.
	GetListImageOfValue (see page 19742)	Returns the image object associated with a specified value.
	GetListName (see page 19742)	Returns the name part of the current name/value pair.
	GetListNameOfValue (see page 19742)	Returns the name associated with a specified value.
	GetListValue (see page 19743)	Returns the value of the current name/value pair.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 19650)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 19650)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasViewAccess (see page 19650)	This is HasViewAccess, a member of class IAdapterAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InIterator (see page 19721)	Indicates whether the object is iterating over its items.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.

V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
V	UpdateRecords (see page 24179)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 24182)	Modifies the adapter's echo flag.
R	Errors (see page 24182)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 24183)	This is ExecuteAccess, a member of class TCustomAdapter.
R	IteratorIndex (see page 19721)	Indicates the index of the current item.
	ModifyAccess (see page 24183)	This is ModifyAccess, a member of class TCustomAdapter.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 24183)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnPrepareStrings (see page 24550)	Implement OnPrepareStrings to customize the content of the strings list.

Published Properties

	Name	Description
	Actions (see page 24549)	Container for adapter actions components.
	Data (see page 24549)	Container for adapter field components.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Strings (see page 24549)	This string list defines the list of name/value pairs.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.77.2 TStringsValuesList Properties

The properties of the TStringsValuesList class are listed here.

Published Properties

	Name	Description
	Actions (see page 24549)	Container for adapter actions components.

	Data (see page 24549)	Container for adapter field components.
	Strings (see page 24549)	This string list defines the list of name/value pairs.

173.1.77.2.1 TStringsValuesList.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 23770)

173.1.77.2.2 TStringsValuesList.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 23778)

173.1.77.2.3 TStringsValuesList.Strings Property

This string list defines the list of name/value pairs.

Pascal

```
property Strings: TStrings;
```

C++

```
__property TStrings Strings;
```

Description

Each line in the string list represents a name/value pair. The name is optional. To specify a name/value enter a line with the name following by "=" followed by the value. For example, "Joe Smith=EMP111" is a named value. "EMP111" is an unnamed value.

173.1.77.3 TStringsValuesList Events

The events of the TStringsValuesList class are listed here.

Published Events

	Name	Description
	OnPrepareStrings (see page 24550)	Implement OnPrepareStrings to customize the content of the strings list.

173.1.77.3.1 TStringsValuesList.OnPrepareStrings Event

Implement OnPrepareStrings to customize the content of the strings list.

Pascal

```
property OnPrepareStrings: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPrepareStrings;
```

Description

OnPrepareStrings is called before the name/value pairs are retrieved by server-side script.

See Also

TNotifyEvent (see page 3298)

PrepareStrings (see page 24390)

173.1.78 TUpdateFileList Class

TUpdateFileList is a container for uploaded files.

Class Hierarchy

WebAdapt.TUpdateFileList

Pascal

```
TUpdateFileList = class;
```

C++

```
class TUpdateFileList;
```

File

WebAdapt

Description

TUpdateFileList contains the files that are being uploaded to a particular adapter field. TUpdateFileList is passed as a parameter to event handlers that operate on a list of uploaded files.

See Also

TCustomAdapterFileField (see page 24220)

TCustomAdapterFileField (see page 24220)

173.1.78.1 TUpdateFileList Members

The following tables list the members exposed by TUpdateFileList.

Public Methods

	Name	Description
	Create (see page 24551)	Creates an instance of TUpdateFileList (see page 24550).

Public Properties

	Name	Description
R	Count (see page 24551)	Returns the count of uploaded files.
R	Files (see page 24552)	Returns a particular uploaded file.

173.1.78.2 TUpdateFileList.Create Constructor

Creates an instance of TUpdateFileList (see page 24550).

Pascal

```
constructor Create(AValue: IActionFieldValue);
```

C++

```
__fastcall TUpdateFileList(IActionFieldValue AValue);
```

Description

```
constructor Create(AValue: IActionFieldValueIActionFieldValue);
```

Create initializes an instance of TUpdateFileList (see page 24550) and calls the inherited Create method.

173.1.78.3 TUpdateFileList Properties

The properties of the TUpdateFileList class are listed here.

Public Properties

	Name	Description
R	Count (see page 24551)	Returns the count of uploaded files.
R	Files (see page 24552)	Returns a particular uploaded file.

173.1.78.3.1 TUpdateFileList.Count Property

Returns the count of uploaded files.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use Count with the Files (see page 24552) property to loop through the uploaded files.

See Also

Files (see page 24552)

173.1.78.3.2 TUpdateFileList.Files Property

Returns a particular uploaded file.

Pascal

```
property Files [I: Integer]: TAbstractWebRequestFile;
```

C++

```
__property TAbstractWebRequestFile Files[int I];
```

Description

Use Files with the Count (see page 24551) property to loop through the uploaded files.

See Also

Count (see page 24551)

173.1.79 TValuesListNameField Class

TValuesListNameField is an adapter field used to access the Name of the current list item of a values list adapter.

Class Hierarchy

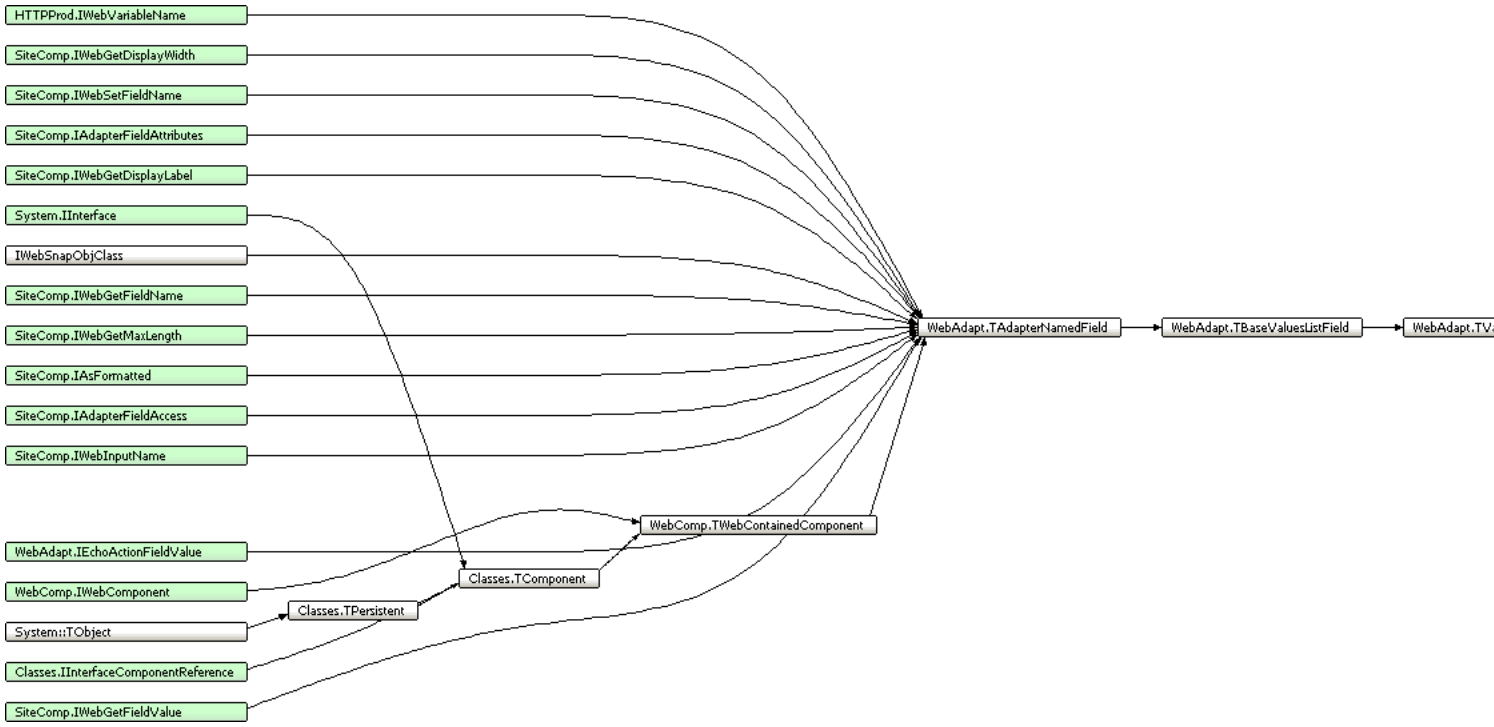

Pascal

```
TValuesListNameField = class(TBaseValuesListField);
```

C++

```
class TValuesListNameField : public TBaseValuesListField;
```

File

WebAdapt

Description

Values list adapters such as TStringValuesList contain at least two adapter fields. One adapter field is responsible for name of the current list item. The other adapter field is responsible for providing the value of the current list item. TValuesListNameField is an adapter field that returns the name of the current list item.

See Also

TBaseValuesListField (see page 24132)

TStringsValuesList (see page 24540)

TValuesListValueField (see page 24558)

173.1.79.1 TValuesListNameField Members

The following tables list the members exposed by TValuesListNameField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 24138)	Retrieves the parent adapter component.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 24557)	Returns the name used to access this adapter field using server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.

	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
V	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 24558)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	Adapter (see page 24139)	Retrieves the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.79.2 TValuesListNameField Methods

The methods of the TValuesListNameField class are listed here.

Protected Methods

	Name	Description
V	GetDefaultFieldName (see page 24557)	Returns the name used to access this adapter field using server-side script.
V	ImplGetValue (see page 24558)	Returns the value associated with this adapter field.

173.1.79.2.1 TValuesListNameField.GetDefaultFieldName Method

Returns the name used to access this adapter field using server-side script.

Pascal

```
function GetDefaultFieldName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultFieldName();
```

Description

GetDefaultFieldName returns 'NameField'. Server-side script can access this adapter field using the NameField property.

173.1.79.2.2 TValuesListNameField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue calls the parent adapter's GetListName function to retrieve the name of the current list item.

See Also

GetListName (see page 24128)

173.1.80 TValuesListValueField Class

TValuesListValueField is an adapter field used to access the Value (see page 23945) of the current list item of a values list adapter.

Class Hierarchy

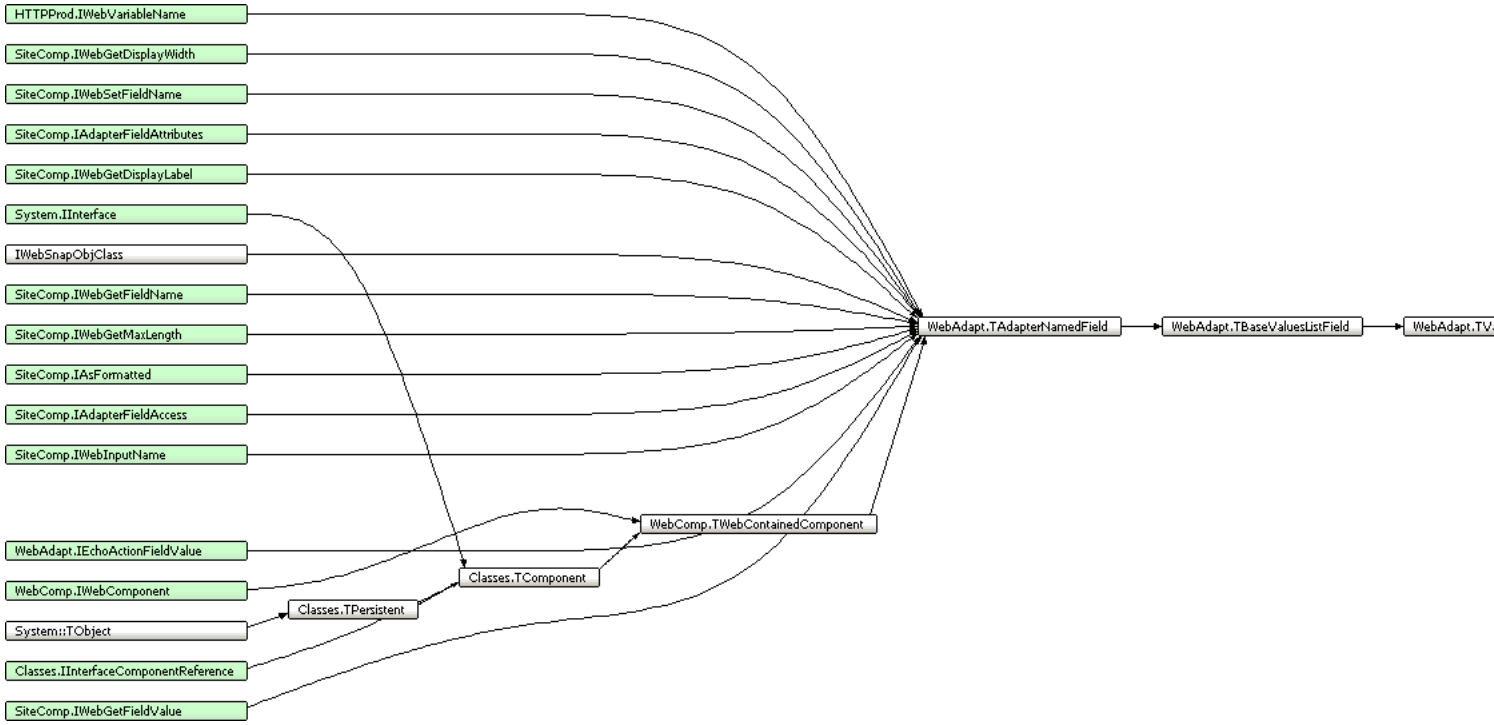

Pascal

```
TValuesListValueField = class(TBaseValuesListField);
```

C++

```
class TValuesListValueField : public TBaseValuesListField;
```

File

WebAdapt

Description

Values list adapters such as TStringValuesList contain at least two adapter fields. One adapter field is responsible for name of the current list item. The other adapter field is responsible for providing the value of the current list item. TValuesListValueField is an adapter field that returns the value of the current list item.

See Also

TBaseValuesListField (see page 24132)

TCustomStringsValuesList (see page 24378)

TValuesListValueField

173.1.80.1 TValuesListValueField Members

The following tables list the members exposed by TValuesListValueField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AsFormatted (see page 23931)	Returns text used to generate a representation of the adapter field value.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckModifyAccess (see page 23931)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 23931)	Checks end-user rights to view the adapter field value.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	FullyQualifyInputName (see page 23932)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 23932)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 24138)	Retrieves the parent adapter component.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 23932)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 24564)	Returns the name used to access this adapter field using server-side script.
	GetDisplayLabel (see page 23933)	Returns the suggested display label for this adapter field.
	GetDisplayWidth (see page 23933)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 23933)	Returns the value of the echo flag.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetInputName (see page 23934)	Identifies the name to use when generating an HTML input for this adapter field.
	GetMaxLength (see page 23934)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequired (see page 23934)	Indicates whether the end-user must enter a value for this adapter field.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetValue (see page 23935)	Gets the value of this adapter field as a Variant.
	GetVariableName (see page 23935)	Gets the name used to reference this adapter field with server-side script.
	GetVisible (see page 23935)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 23936)	Returns the class type for this adapter field.
	HasModifyAccess (see page 23936)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 23936)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 23937)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 23937)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 23937)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 23938)	Returns the value of the echo flag.
V	ImplGetInputName (see page 23938)	Identifies the name to use when generating the HTML input for this adapter field.

V	ImplGetMaxLength (see page 23938)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 23939)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 24564)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 23939)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 23940)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 23940)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 23940)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 23941)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 23941)	Sets the name that is used to reference this adapter field with server-side script.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 23941)	Handles movement of the component to a different parent.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetEchoActionFieldValue (see page 23942)	Sets the echo flag for this adapter field.
	SetFieldName (see page 23942)	Sets the name that is used to reference this adapter field with server-side script.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 23943)	Returns the parent adapter component.
R	Adapter (see page 24139)	Retrieves the parent adapter component.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AsFormatted (see page 19682)	This is AsFormatted, a member of class IAsFormatted.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDisplayLabel (see page 19756)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayWidth (see page 19757)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInputName (see page 19767)	This is GetInputName, a member of class IWebInputName.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetMaxLength (see page 19763)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.

	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 19668)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 19759)	This is GetValue, a member of class IWebGetFieldValue.
	GetVariableName (see page 15329)	This is GetVariableName, a member of class IWebVariableName.
	GetVisible (see page 19668)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 19667)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 19667)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActionOrigValue (see page 23943)	Retrieve the original value of this adapter field.
R	ActionValue (see page 23943)	Retrieve the value for this adapter field entered by the end-user.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 23944)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.

	FieldName (see page 23944)	Identifies the name of this field when using server-side script.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 19767)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 23944)	Identifies the rights required to update this field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Required (see page 19669)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 19760)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 23945)	Provides the value of an adapter field as a variant.
R	VariableName (see page 15329)	This is VariableName, a member of class IWebVariableName.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewAccess (see page 23945)	Identifies the rights required to view this field.
R	Visible (see page 19669)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

173.1.80.2 TValuesListValueField Methods

The methods of the TValuesListValueField class are listed here.

Protected Methods

	Name	Description
V	GetDefaultFieldName (see page 24564)	Returns the name used to access this adapter field using server-side script.
V	ImplGetValue (see page 24564)	Returns the value associated with this adapter field.

173.1.80.2.1 TValuesListValueField.GetDefaultFieldName Method

Returns the name used to access this adapter field using server-side script.

Pascal

```
function GetDefaultFieldName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDefaultFieldName();
```

Description

GetDefaultFieldName returns 'ValueField'. Server-side script can access this adapter field using the ValueField property.

173.1.80.2.2 TValuesListValueField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue calls the parent adapter's GetListValue function to retrieve the value of the current list item.

See Also

GetListValue (see page 24129)

173.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAdapterEditor (see page 24566)	This is class WebAdapt.IAdapterEditor.
	IAdapterNotifyAdapterChange (see page 24567)	This is class WebAdapt.IAdapterNotifyAdapterChange.
	ICheckOrigValue (see page 24568)	This is class WebAdapt.ICheckOrigValue.
	ICheckValueChange (see page 24569)	This is class WebAdapt.ICheckValueChange.
	IClearAdapterValues (see page 24570)	This is class WebAdapt.IClearAdapterValues.
	ICreateActionRequestContext (see page 24571)	This is class WebAdapt.ICreateActionRequestContext.
	IEchoActionFieldValue (see page 24572)	This is class WebAdapt.IEchoActionFieldValue.
	IEchoAdapterFieldValues (see page 24574)	This is class WebAdapt.IEchoAdapterFieldValues.
	IExecuteValue (see page 24575)	IExecuteValue is the interface for executing an adapter's fields with values passed in an HTTP request.
	IFullyQualifyInputNames (see page 24577)	This is class WebAdapt.IFullyQualifyInputNames.
	IGetAdapterErrorsList (see page 24578)	This is class WebAdapt.IGetAdapterErrorsList.
	IGetAdapterModeNames (see page 24579)	This is class WebAdapt.IGetAdapterModeNames.
	IGetRecordKeys (see page 24580)	This is class WebAdapt.IGetRecordKeys.
	IIdentifyAdapter (see page 24581)	This is class WebAdapt.IIdentifyAdapter.
	ISetSubComponents (see page 24581)	This is class WebAdapt.ISetSubComponents.
	ISetWebFieldValue (see page 24582)	This is class WebAdapt.ISetWebFieldValue.
	IUpdateValue (see page 24583)	This is class WebAdapt.IUpdateValue.
	IValidateValue (see page 24584)	This is class WebAdapt.IValidateValue.
	IWebActionsSubcomponent (see page 24585)	This is class WebAdapt.IWebActionsSubcomponent.
	IWebFieldsSubcomponent (see page 24585)	This is class WebAdapt.IWebFieldsSubcomponent.
	IWebGetFieldPrevValue (see page 24586)	This is class WebAdapt.IWebGetFieldPrevValue.

173.2.1 IAdapterEditor Interface

Class Hierarchy

WebAdapt.IAdapterEditor

Pascal

```
IAdapterEditor = interface;
```

C++

```
__interface IAdapterEditor;
```

File

WebAdapt

Description

This is class WebAdapt.IAdapterEditor.

173.2.1.1 IAdapterEditor Members

The following tables list the members exposed by IAdapterEditor.

Public Methods

	Name	Description
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.

173.2.1.2 IAdapterEditor Methods

The methods of the IAdapterEditor class are listed here.

Public Methods

	Name	Description
	CanAddActionClass (see page 24567)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 24567)	This is CanAddFieldClass, a member of class IAdapterEditor.

173.2.1.2.1 IAdapterEditor.CanAddActionClass Method

Pascal

```
function CanAddActionClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

This is CanAddActionClass, a member of class IAdapterEditor.

173.2.1.2.2 IAdapterEditor.CanAddFieldClass Method

Pascal

```
function CanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

This is CanAddFieldClass, a member of class IAdapterEditor.

173.2.2 IAdapterNotifyAdapterChange Interface

Class Hierarchy

WebAdapt.IAdapterNotifyAdapterChange

Pascal

```
IAdapterNotifyAdapterChange = interface;
```

C++

```
__interface IAdapterNotifyAdapterChange;
```

File

WebAdapt

Description

This is class WebAdapt.IAdapterNotifyAdapterChange.

173.2.2.1 IAdapterNotifyAdapterChange Members

The following tables list the members exposed by IAdapterNotifyAdapterChange.

Public Methods

	Name	Description
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.

173.2.2.2 IAdapterNotifyAdapterChange Methods

The methods of the IAdapterNotifyAdapterChange class are listed here.

Public Methods

	Name	Description
	NotifyAdapterChange (see page 24568)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.

173.2.2.2.1 IAdapterNotifyAdapterChange.NotifyAdapterChange Method

Pascal

```
procedure NotifyAdapterChange;
```

C++

```
__fastcall NotifyAdapterChange();
```

Description

This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.

173.2.3 ICheckOrigValue Interface

Class Hierarchy

WebAdapt.ICheckOrigValue

Pascal

```
ICheckOrigValue = interface;
```

C++

```
__interface ICheckOrigValue;
```

File

WebAdapt

Description

This is class WebAdapt.ICheckOrigValue.

173.2.3.1 ICheckOrigValue Members

The following tables list the members exposed by ICheckOrigValue.

Public Methods

	Name	Description
	CheckOrigValue (see page 24569)	This is CheckOrigValue, a member of class ICheckOrigValue.

173.2.3.2 ICheckOrigValue Methods

The methods of the ICheckOrigValue class are listed here.

Public Methods

	Name	Description
	CheckOrigValue (see page 24569)	This is CheckOrigValue, a member of class ICheckOrigValue.

173.2.3.2.1 ICheckOrigValue.CheckOrigValue Method

Pascal

```
procedure CheckOrigValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

```
__fastcall CheckOrigValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

This is CheckOrigValue, a member of class ICheckOrigValue.

173.2.4 ICheckValueChange Interface

Class Hierarchy

WebAdapt.ICheckValueChange

Pascal

```
ICheckValueChange = interface;
```

C++

```
__interface ICheckValueChange;
```

File

WebAdapt

Description

This is class WebAdapt.ICheckValueChange.

173.2.4.1 ICheckValueChange Members

The following tables list the members exposed by ICheckValueChange.

Public Methods

	Name	Description
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.

173.2.4.2 ICheckValueChange Methods

The methods of the ICheckValueChange class are listed here.

Public Methods

	Name	Description
	CheckValueChange (see page 24570)	This is CheckValueChange, a member of class ICheckValueChange.

173.2.4.2.1 ICheckValueChange.CheckValueChange Method

Pascal

```
function CheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
```

C++

```
__fastcall Boolean CheckValueChange(IActionRequest AActionRequest, int AFieldIndex);
```

Description

This is CheckValueChange, a member of class ICheckValueChange.

173.2.5 IClearAdapterValues Interface

Class Hierarchy

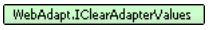

Pascal

```
IClearAdapterValues = interface;
```

C++

```
__interface IClearAdapterValues;
```

File

WebAdapt

Description

This is class WebAdapt.IClearAdapterValues.

173.2.5.1 IClearAdapterValues Members

The following tables list the members exposed by IClearAdapterValues.

Public Methods

	Name	Description
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.

173.2.5.2 IClearAdapterValues Methods

The methods of the IClearAdapterValues class are listed here.

Public Methods

	Name	Description
	ClearAdapterValues (see page 24571)	This is ClearAdapterValues, a member of class IClearAdapterValues.

173.2.5.2.1 IClearAdapterValues.ClearAdapterValues Method

Pascal

```
procedure ClearAdapterValues;
```

C++

```
__fastcall ClearAdapterValues();
```

Description

This is ClearAdapterValues, a member of class IClearAdapterValues.

173.2.6 ICreateActionRequestContext Interface

Class Hierarchy

WebAdapt.ICreateActionRequestContext

Pascal

```
ICreateActionRequestContext = interface;
```

C++

```
__interface ICreateActionRequestContext;
```

File

WebAdapt

Description

This is class WebAdapt.ICreateActionRequestContext.

173.2.6.1 ICreateActionRequestContext Members

The following tables list the members exposed by ICreateActionRequestContext.

Public Methods

	Name	Description
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.

173.2.6.2 ICreateActionRequestContext Methods

The methods of the ICreateActionRequestContext class are listed here.

Public Methods

	Name	Description
	CreateActionRequestContext (see page 24572)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.

173.2.6.2.1 ICreateActionRequestContext.CreateActionRequestContext Method

Pascal

```
procedure CreateActionRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall CreateActionRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

This is CreateActionRequestContext, a member of class ICreateActionRequestContext.

173.2.7 IEchoActionFieldValue Interface

Class Hierarchy

WebAdapt.IEchoActionFieldValue

Pascal

```
IEchoActionFieldValue = interface;
```

C++

```
__interface IEchoActionFieldValue;
```

File

WebAdapt

Description

This is class WebAdapt.IEchoActionFieldValue.

173.2.7.1 IEchoActionFieldValue Members

The following tables list the members exposed by IEchoActionFieldValue.

Public Methods

	Name	Description
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.

Public Properties

	Name	Description
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.

173.2.7.2 IEchoActionFieldValue Methods

The methods of the IEchoActionFieldValue class are listed here.

Public Methods

	Name	Description
	GetEchoActionFieldValue (see page 24573)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetEchoActionFieldValue (see page 24573)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.

173.2.7.2.1 IEchoActionFieldValue.GetEchoActionFieldValue Method

Pascal

```
function GetEchoActionFieldValue: Boolean;
```

C++

```
__fastcall Boolean GetEchoActionFieldValue();
```

Description

This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.

173.2.7.2.2 IEchoActionFieldValue.SetEchoActionFieldValue Method

Pascal

```
procedure SetEchoActionFieldValue(AValue: Boolean);
```

C++

```
__fastcall SetEchoActionFieldValue(Boolean AValue);
```

Description

This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.

173.2.7.3 IEchoActionFieldValue Properties

The properties of the IEchoActionFieldValue class are listed here.

Public Properties

	Name	Description
	EchoActionFieldValue (see page 24574)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.

173.2.7.3.1 IEchoActionFieldValue.EchoActionFieldValue Property

Pascal

```
property EchoActionFieldValue: Boolean;
```

C++

```
__property Boolean EchoActionFieldValue;
```

Description

This is EchoActionFieldValue, a member of class IEchoActionFieldValue.

173.2.8 IEchoAdapterFieldValues Interface

Class Hierarchy

WebAdapt.IEchoAdapterFieldValues

Pascal

```
IEchoAdapterFieldValues = interface;
```

C++

```
__interface IEchoAdapterFieldValues;
```

File

WebAdapt

Description

This is class WebAdapt.IEchoAdapterFieldValues.

173.2.8.1 IEchoAdapterFieldValues Members

The following tables list the members exposed by IEchoAdapterFieldValues.

Public Methods

	Name	Description
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.

Public Properties

	Name	Description
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.

173.2.8.2 IEchoAdapterFieldValues Methods

The methods of the IEchoAdapterFieldValues class are listed here.

Public Methods

	Name	Description
	GetEchoActionFieldValues (see page 24575)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.

	SetEchoActionFieldValues (see page 24575)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.

173.2.8.2.1 IEchoAdapterFieldValues.GetEchoActionFieldValues Method

Pascal

```
function GetEchoActionFieldValues: Boolean;
```

C++

```
__fastcall Boolean GetEchoActionFieldValues();
```

Description

This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.

173.2.8.2.2 IEchoAdapterFieldValues.SetEchoActionFieldValues Method

Pascal

```
procedure SetEchoActionFieldValues(AValue: Boolean);
```

C++

```
__fastcall SetEchoActionFieldValues(Boolean AValue);
```

Description

This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.

173.2.8.3 IEchoAdapterFieldValues Properties

The properties of the IEchoAdapterFieldValues class are listed here.

Public Properties

	Name	Description
	EchoActionFieldValues (see page 24575)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.

173.2.8.3.1 IEchoAdapterFieldValues.EchoActionFieldValues Property

Pascal

```
property EchoActionFieldValues: Boolean;
```

C++

```
__property Boolean EchoActionFieldValues;
```

Description

This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.

173.2.9 IExecuteValue Interface

IExecuteValue is the interface for executing an adapter's fields with values passed in an HTTP request.

Class Hierarchy

WebAdapt.IExecuteValue

Pascal

```
IExecuteValue = interface;
```

C++

```
__interface IExecuteValue;
```

File

WebAdapt

Description

Adapter fields implement the IExecuteValue interface when they are executable. When a field is executable, HTTP requests can supply field values that are used to execute a command rather than update a value. For example, an HTML form may contain a check box or radio button control that indicates an action the user wants to take. When the form is submitted, the values for the field that underlies the check box or radio buttons indicate the user's choice. When the WebSnap application receives the HTTP request, it uses the IExecuteValue interface to call the field's ExecuteValue (see page 24576) method.

Note: Method declarations that use IExecuteValue use the _di_IExecuteValue type instead. This type is a DelphiInterface wrapper around the IExecuteValue interface:

Note: typedef System::DelphiInterface (see page 22167)< IExecuteValue > _di_IExecuteValue;

See Also

TCustomAdapter (see page 24139)

173.2.9.1 IExecuteValue Members

The following tables list the members exposed by IExecuteValue.

Public Methods

	Name	Description
	ExecuteValue (see page 24576)	Executes the field based on a value supplied in the current HTTP request.

173.2.9.2 IExecuteValue Methods

The methods of the IExecuteValue class are listed here.

Public Methods

	Name	Description
	ExecuteValue (see page 24576)	Executes the field based on a value supplied in the current HTTP request.

173.2.9.2.1 IExecuteValue.ExecuteValue Method

Executes the field based on a value supplied in the current HTTP request.

Pascal

```
procedure ExecuteValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

```
__fastcall ExecuteValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

An adapter calls ExecuteValue when responding to an HTTP request that supplies values from executable fields.

AActionRequest is the current request object. ExecuteValue uses this interface to extract the value to use when executing the field.

AFieldIndex is the index of the target field in the list of field values that the request object supplies.

See Also

UpdateFieldsExecuteValues (see page 24177)

173.2.10 IFullyQualifyInputNames Interface

Class Hierarchy

WebAdapt.IFullyQualifyInputNames

Pascal

```
IFullyQualifyInputNames = interface;
```

C++

```
__interface IFullyQualifyInputNames;
```

File

WebAdapt

Description

This is class WebAdapt.IFullyQualifyInputNames.

173.2.10.1 IFullyQualifyInputNames Members

The following tables list the members exposed by IFullyQualifyInputNames.

Public Methods

	Name	Description
	FullyQualifyInputNames (see page 24577)	This is FullyQualifyInputNames, a member of class IFullyQualifyInputNames.

173.2.10.2 IFullyQualifyInputNames Methods

The methods of the IFullyQualifyInputNames class are listed here.

Public Methods

	Name	Description
	FullyQualifyInputNames (see page 24577)	This is FullyQualifyInputNames, a member of class IFullyQualifyInputNames.

173.2.10.2.1 IFullyQualifyInputNames.FullyQualifyInputNames Method

Pascal

```
function FullyQualifyInputNames: Boolean;
```

C++

```
__fastcall Boolean FullyQualifyInputNames();
```

Description

This is FullyQualifyInputNames, a member of class IFullyQualifyInputNames.

173.2.11 IGetAdapterErrorsList Interface

Class Hierarchy

WebAdapt.IGetAdapterErrorsList

Pascal

```
IGetAdapterErrorsList = interface;
```

C++

```
__interface IGetAdapterErrorsList;
```

File

WebAdapt

Description

This is class WebAdapt.IGetAdapterErrorsList.

173.2.11.1 IGetAdapterErrorsList Members

The following tables list the members exposed by IGetAdapterErrorsList.

Public Methods

	Name	Description
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.

173.2.11.2 IGetAdapterErrorsList Methods

The methods of the IGetAdapterErrorsList class are listed here.

Public Methods

	Name	Description
	GetAdapterErrorsList (see page 24578)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.

173.2.11.2.1 IGetAdapterErrorsList.GetAdapterErrorsList Method

Pascal

```
function GetAdapterErrorsList: TBaseAdapterErrorsList;
```

C++

```
__fastcall TBaseAdapterErrorsList GetAdapterErrorsList();
```

Description

This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.

173.2.12 IGetAdapterModeNames Interface

Class Hierarchy

WebAdapt.IGetAdapterModeNames

Pascal

```
IGetAdapterModeNames = interface;
```

C++

```
__interface IGetAdapterModeNames;
```

File

WebAdapt

Description

This is class WebAdapt.IGetAdapterModeNames.

173.2.12.1 IGetAdapterModeNames Members

The following tables list the members exposed by IGetAdapterModeNames.

Public Methods

	Name	Description
	GetAdapterModeNames (see page 24579)	This is GetAdapterModeNames, a member of class IGetAdapterModeNames.

173.2.12.2 IGetAdapterModeNames Methods

The methods of the IGetAdapterModeNames class are listed here.

Public Methods

	Name	Description
	GetAdapterModeNames (see page 24579)	This is GetAdapterModeNames, a member of class IGetAdapterModeNames.

173.2.12.2.1 IGetAdapterModeNames.GetAdapterModeNames Method

Pascal

```
procedure GetAdapterModeNames(AStrings: TStrings);
```

C++

```
__fastcall GetAdapterModeNames(TStrings AStrings);
```

Description

This is GetAdapterModeNames, a member of class IGetAdapterModeNames.

173.2.13 IGetRecordKeys Interface

Class Hierarchy

WebAdapt.IGetRecordKeys

Pascal

```
IGetRecordKeys = interface;
```

C++

```
__interface IGetRecordKeys;
```

File

WebAdapt

Description

This is class WebAdapt.IGetRecordKeys.

173.2.13.1 IGetRecordKeys Members

The following tables list the members exposed by IGetRecordKeys.

Public Methods

	Name	Description
	GetRecordKeys (see page 24580)	This is GetRecordKeys, a member of class IGetRecordKeys.

173.2.13.2 IGetRecordKeys Methods

The methods of the IGetRecordKeys class are listed here.

Public Methods

	Name	Description
	GetRecordKeys (see page 24580)	This is GetRecordKeys, a member of class IGetRecordKeys.

173.2.13.2.1 IGetRecordKeys.GetRecordKeys Method

Pascal

```
procedure GetRecordKeys(AParams: TStrings; var AFullyQualify: Boolean);
```

C++

```
__fastcall GetRecordKeys(TStrings AParams, Boolean AFullyQualify);
```

Description

This is GetRecordKeys, a member of class IGetRecordKeys.

173.2.14 IIdentifyAdapter Interface

Class Hierarchy

WebAdapt.IIdentifyAdapter

Pascal

```
IIdentifyAdapter = interface;
```

C++

```
__interface IIdentifyAdapter;
```

File

WebAdapt

Description

This is class WebAdapt.IIdentifyAdapter.

173.2.15 ISetSubComponents Interface

Class Hierarchy

WebAdapt.ISetSubComponents

Pascal

```
ISetSubComponents = interface;
```

C++

```
__interface ISetSubComponents;
```

File

WebAdapt

Description

This is class WebAdapt.ISetSubComponents.

173.2.15.1 ISetSubComponents Members

The following tables list the members exposed by ISetSubComponents.

Public Methods

	Name	Description
	SetActions (see page 24582)	This is SetActions, a member of class ISetSubComponents.
	SetData (see page 24582)	This is SetData, a member of class ISetSubComponents.

173.2.15.2 ISetSubComponents Methods

The methods of the ISetSubComponents class are listed here.

Public Methods

	Name	Description
	SetActions (see page 24582)	This is SetActions, a member of class ISetSubComponents.
	SetData (see page 24582)	This is SetData, a member of class ISetSubComponents.

173.2.15.2.1 ISetSubComponents.SetActions Method

Pascal

```
procedure SetActions(const Value: TAdapterSubComponent);
```

C++

```
__fastcall SetActions(const TAdapterSubComponent Value);
```

Description

This is SetActions, a member of class ISetSubComponents.

173.2.15.2.2 ISetSubComponents.SetData Method

Pascal

```
procedure SetData(const Value: TAdapterSubComponent);
```

C++

```
__fastcall SetData(const TAdapterSubComponent Value);
```

Description

This is SetData, a member of class ISetSubComponents.

173.2.16 ISetWebFieldValue Interface

Class Hierarchy

WebAdapt.ISetWebFieldValue

Pascal

```
ISetWebFieldValue = interface;
```

C++

```
__interface ISetWebFieldValue;
```

File

WebAdapt

Description

This is class WebAdapt.ISetWebFieldValue.

173.2.16.1 ISetWebFieldValue Members

The following tables list the members exposed by ISetWebFieldValue.

Public Methods

	Name	Description
	SetValue (see page 24583)	This is SetValue, a member of class ISetWebFieldValue.

173.2.16.2 ISetWebFieldValue Methods

The methods of the ISetWebFieldValue class are listed here.

Public Methods

	Name	Description
	SetValue (see page 24583)	This is SetValue, a member of class ISetWebFieldValue.

173.2.16.2.1 ISetWebFieldValue.SetValue Method

Pascal

```
procedure SetValue(const AValue: Variant);
```

C++

```
__fastcall SetValue(const Variant AValue);
```

Description

This is SetValue, a member of class ISetWebFieldValue.

173.2.17 IUpdateValue Interface

Class Hierarchy

WebAdapt.IUpdateValue

Pascal

```
IUpdateValue = interface;
```

C++

```
__interface IUpdateValue;
```

File

WebAdapt

Description

This is class WebAdapt.IUpdateValue.

173.2.17.1 IUpdateValue Members

The following tables list the members exposed by IUpdateValue.

Public Methods

	Name	Description
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.

173.2.17.2 IUpdateValue Methods

The methods of the IUpdateValue class are listed here.

Public Methods

	Name	Description
	UpdateValue (see page 24584)	This is UpdateValue, a member of class IUpdateValue.

173.2.17.2.1 IUpdateValue.UpdateValue Method

Pascal

```
procedure UpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

```
__fastcall UpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

This is UpdateValue, a member of class IUpdateValue.

173.2.18 IValidateValue Interface

Class Hierarchy

WebAdapt.IValidateValue

Pascal

```
IValidateValue = interface;
```

C++

```
__interface IValidateValue;
```

File

WebAdapt

Description

This is class WebAdapt.IValidateValue.

173.2.18.1 IValidateValue Members

The following tables list the members exposed by IValidateValue.

Public Methods

	Name	Description
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.

173.2.18.2 IValidateValue Methods

The methods of the IValidateValue class are listed here.

Public Methods

	Name	Description
	ValidateValue (see page 24585)	This is ValidateValue, a member of class IValidateValue.

173.2.18.2.1 IValidateValue.ValidateValue Method

Pascal

```
procedure ValidateValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

```
__fastcall ValidateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

This is ValidateValue, a member of class IValidateValue.

173.2.19 IWebActionsSubcomponent Interface

Class Hierarchy

WebAdapt.IWebActionsSubcomponent

Pascal

```
IWebActionsSubcomponent = interface;
```

C++

```
__interface IWebActionsSubcomponent;
```

File

WebAdapt

Description

This is class WebAdapt.IWebActionsSubcomponent.

173.2.20 IWebFieldsSubcomponent Interface

Class Hierarchy

WebAdapt.IWebFieldsSubcomponent

Pascal

```
IWebFieldsSubcomponent = interface;
```

C++

```
__interface IWebFieldsSubcomponent;
```

File

WebAdapt

Description

This is class WebAdapt.IWebFieldsSubcomponent.

173.2.21 IWebGetFieldPrevValue Interface

Class Hierarchy

WebAdapt.IWebGetFieldPrevValue

Pascal

```
IWebGetFieldPrevValue = interface;
```

C++

```
__interface IWebGetFieldPrevValue;
```

File

WebAdapt

Description

This is class WebAdapt.IWebGetFieldPrevValue.

173.2.21.1 IWebGetFieldPrevValue Members

The following tables list the members exposed by IWebGetFieldPrevValue.

Public Methods

	Name	Description
	GetNeedPrevValue (see page 24587)	This is GetNeedPrevValue, a member of class IWebGetFieldPrevValue.
	GetPrevValue (see page 24587)	This is GetPrevValue, a member of class IWebGetFieldPrevValue.

Public Properties

	Name	Description
R	NeedPrevValue (see page 24587)	This is NeedPrevValue, a member of class IWebGetFieldPrevValue.
R	PrevValue (see page 24587)	This is PrevValue, a member of class IWebGetFieldPrevValue.

173.2.21.2 IWebGetFieldPrevValue Methods

The methods of the IWebGetFieldPrevValue class are listed here.

Public Methods

	Name	Description
	GetNeedPrevValue (see page 24587)	This is GetNeedPrevValue, a member of class IWebGetFieldPrevValue.
	GetPrevValue (see page 24587)	This is GetPrevValue, a member of class IWebGetFieldPrevValue.

173.2.21.2.1 IWebGetFieldPrevValue.GetNeedPrevValue Method

Pascal

```
function GetNeedPrevValue: Boolean;
```

C++

```
__fastcall Boolean GetNeedPrevValue();
```

Description

This is GetNeedPrevValue, a member of class IWebGetFieldPrevValue.

173.2.21.2.2 IWebGetFieldPrevValue.GetPrevValue Method

Pascal

```
function GetPrevValue: Variant;
```

C++

```
__fastcall Variant GetPrevValue();
```

Description

This is GetPrevValue, a member of class IWebGetFieldPrevValue.

173.2.21.3 IWebGetFieldPrevValue Properties

The properties of the IWebGetFieldPrevValue class are listed here.

Public Properties

	Name	Description
R	NeedPrevValue (see page 24587)	This is NeedPrevValue, a member of class IWebGetFieldPrevValue.
R	PrevValue (see page 24587)	This is PrevValue, a member of class IWebGetFieldPrevValue.

173.2.21.3.1 IWebGetFieldPrevValue.NeedPrevValue Property

Pascal

```
property NeedPrevValue: Boolean;
```

C++

```
__property Boolean NeedPrevValue;
```

Description

This is NeedPrevValue, a member of class IWebGetFieldPrevValue.

173.2.21.3.2 IWebGetFieldPrevValue.PrevValue Property

Pascal

```
property PrevValue: Variant;
```

C++

```
__property Variant PrevValue;
```

Description

This is PrevValue, a member of class IWebGetFieldPrevValue.

173.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AddDefaultActions (see page 24589)	This is function WebAdapt.AddDefaultActions.
	AddDefaultFields (see page 24589)	This is function WebAdapt.AddDefaultFields.
	FindAdapterAction (see page 24590)	This is function WebAdapt.FindAdapterAction.
	FindAdapterField (see page 24590)	This is function WebAdapt.FindAdapterField.
	FindAdapterInParent (see page 24590)	This is function WebAdapt.FindAdapterInParent.
	MakeAdapterRequestIdentifier (see page 24591)	This is function WebAdapt.MakeAdapterRequestIdentifier.
	MakeValidAdapterFieldName (see page 24591)	This is function WebAdapt.MakeValidAdapterFieldName.
	RaiseFileUploadNotSupported (see page 24591)	This is function WebAdapt.RaiseFileUploadNotSupported.
	RaiseMultipleFilesException (see page 24592)	This is function WebAdapt.RaiseMultipleFilesException.
	RaiseMultipleValuesException (see page 24592)	This is function WebAdapt.RaiseMultipleValuesException.

173.3.1 WebAdapt.AddDefaultActions Function

Pascal

```
procedure AddDefaultActions(AList: TStrings; AComponents: TWebComponentList);
```

C++

```
AddDefaultActions(TStrings AList, TWebComponentList AComponents);
```

File

WebAdapt

Description

This is function WebAdapt.AddDefaultActions.

173.3.2 WebAdapt.AddDefaultFields Function

Pascal

```
procedure AddDefaultFields(AList: TStrings; AComponents: TWebComponentList);
```

C++

```
AddDefaultFields(TStrings AList, TWebComponentList AComponents);
```

File

WebAdapt

Description

This is function WebAdapt.AddDefaultFields.

173.3.3 WebAdapt.FindAdapterAction Function

Pascal

```
function FindAdapterAction(AWebComponents: TWebComponentList; const AName: string): TComponent;
```

C++

```
TComponent * FindAdapterAction(TWebComponentList AWebComponents, const AnsiString AName);
```

File

WebAdapt

Description

This is function WebAdapt.FindAdapterAction.

173.3.4 WebAdapt.FindAdapterField Function

Pascal

```
function FindAdapterField(AWebComponents: TWebComponentList; const AName: string): TComponent;
```

C++

```
TComponent * FindAdapterField(TWebComponentList AWebComponents, const AnsiString AName);
```

File

WebAdapt

Description

This is function WebAdapt.FindAdapterField.

173.3.5 WebAdapt.FindAdapterInParent Function

Pascal

```
function FindAdapterInParent(AComponent: TComponent): TComponent;
```

C++

```
TComponent * FindAdapterInParent(TComponent * AComponent);
```

File

WebAdapt

Description

This is function WebAdapt.FindAdapterInParent.

173.3.6 WebAdapt.MakeAdapterRequestIdentifier Function

Pascal

```
function MakeAdapterRequestIdentifier(AComponent: TComponent): string;
```

C++

```
AnsiString MakeAdapterRequestIdentifier(TComponent * AComponent);
```

File

WebAdapt

Description

This is function WebAdapt.MakeAdapterRequestIdentifier.

173.3.7 WebAdapt.MakeValidAdapterFieldName Function

Pascal

```
function MakeValidAdapterFieldName(const AFieldName: string): string;
```

C++

```
AnsiString MakeValidAdapterFieldName(const AnsiString AFieldName);
```

File

WebAdapt

Description

This is function WebAdapt.MakeValidAdapterFieldName.

173.3.8 WebAdapt.RaiseFileUploadNotSupported Function

Pascal

```
procedure RaiseFileUploadNotSupported(AFieldName: string);
```

C++

```
RaiseFileUploadNotSupported(AnsiString AFieldName);
```

File

WebAdapt

Description

This is function WebAdapt.RaiseFileUploadNotSupported.

173.3.9 WebAdapt.RaiseMultipleFilesException Function

Pascal

```
procedure RaiseMultipleFilesException(AFieldName: string);
```

C++

```
RaiseMultipleFilesException(AnsiString AFieldName);
```

File

WebAdapt

Description

This is function WebAdapt.RaiseMultipleFilesException.

173.3.10 WebAdapt.RaiseMultipleValuesException Function

Pascal

```
procedure RaiseMultipleValuesException(AFieldName: string);
```

C++

```
RaiseMultipleValuesException(AnsiString AFieldName);
```

File

WebAdapt

Description

This is function WebAdapt.RaiseMultipleValuesException.

173.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAdapterHiddenFieldOption (see page 24593)	This is record WebAdapt.TAdapterHiddenFieldOption.
	TIteratorMethod (see page 24593)	This is record WebAdapt.TIteratorMethod.

173.4.1 WebAdapt.TAdapterHiddenFieldOption Enumeration

Pascal

```
TAdapterHiddenFieldOption = (
  hoOrigValues,
  hoMode,
  hoRecordKeys
);
```

C++

```
enum TAdapterHiddenFieldOption {
  hoOrigValues,
  hoMode,
  hoRecordKeys
};
```

File

WebAdapt

Description

This is record WebAdapt.TAdapterHiddenFieldOption.

173.4.2 WebAdapt.TIteratorMethod Enumeration

Pascal

```
TIteratorMethod = (
  itStart,
  itNext,
  itEnd
);
```

C++

```
enum TIteratorMethod {
  itStart,
  itNext,
  itEnd
};
```

File

WebAdapt

Description

This is record WebAdapt.TIteratorMethod.

173.5 Types

The following table lists types in this documentation.

Types

Name	Description
IActionFieldValue (see page 24596)	This is type WebAdapt.IActionFieldValue.
IActionRequest (see page 24596)	This is type WebAdapt.IActionRequest.
TAbstractWebRequestFile (see page 24596)	This is type WebAdapt.TAbstractWebRequestFile.
TActionExecuteEvent (see page 24597)	This is type WebAdapt.TActionExecuteEvent.
TActionExecuteHandledEvent (see page 24597)	This is type WebAdapt.TActionExecuteHandledEvent.
TActionGetEnabledEvent (see page 24597)	This is type WebAdapt.TActionGetEnabledEvent.
TActionGetParamsEvent (see page 24598)	This is type WebAdapt.TActionGetParamsEvent.
TActionRequestFieldsEvent (see page 24598)	This is type WebAdapt.TActionRequestFieldsEvent.
TActionRequestFieldsHandledEvent (see page 24598)	This is type WebAdapt.TActionRequestFieldsHandledEvent.
TAdapterActionExecuteEvent (see page 24599)	This is type WebAdapt.TAdapterActionExecuteEvent.
TAdapterActionExecuteHandledEvent (see page 24599)	This is type WebAdapt.TAdapterActionExecuteHandledEvent.
TAdapterActionGetParamsEvent (see page 24599)	This is type WebAdapt.TAdapterActionGetParamsEvent.
TAdapterActionsClass (see page 24600)	This is type WebAdapt.TAdapterActionsClass.
TAdapterFieldBooleanValueEvent (see page 24600)	This is type WebAdapt.TAdapterFieldBooleanValueEvent.
TAdapterFieldFileEvent (see page 24600)	This is type WebAdapt.TAdapterFieldFileEvent.
TAdapterFieldFileListEvent (see page 24601)	This is type WebAdapt.TAdapterFieldFileListEvent.
TAdapterFieldGetBooleanEvent (see page 24601)	This is type WebAdapt.TAdapterFieldGetBooleanEvent.
TAdapterFieldGetMultiValueEvent (see page 24601)	This is type WebAdapt.TAdapterFieldGetMultiValueEvent.
TAdapterFieldGetMultiValueStringEvent (see page 24602)	This is type WebAdapt.TAdapterFieldGetMultiValueStringEvent.
TAdapterFieldGetStringEvent (see page 24602)	This is type WebAdapt.TAdapterFieldGetStringEvent.
TAdapterFieldGetValueCountEvent (see page 24602)	This is type WebAdapt.TAdapterFieldGetValueCountEvent.
TAdapterFieldGetValueEvent (see page 24603)	This is type WebAdapt.TAdapterFieldGetValueEvent.
TAdapterFieldHasValueEvent (see page 24603)	This is type WebAdapt.TAdapterFieldHasValueEvent.
TAdapterFieldsClass (see page 24603)	This is type WebAdapt.TAdapterFieldsClass.
TAdapterFieldStringEvent (see page 24604)	This is type WebAdapt.TAdapterFieldStringEvent.
TAdapterFieldStringsEvent (see page 24604)	This is type WebAdapt.TAdapterFieldStringsEvent.
TAdapterFieldValidateValueEvent (see page 24604)	This is type WebAdapt.TAdapterFieldValidateValueEvent.
TAdapterFieldValueEvent (see page 24605)	This is type WebAdapt.TAdapterFieldValueEvent.
TAdapterHiddenFieldOptions (see page 24605)	This is type WebAdapt.TAdapterHiddenFieldOptions.
TEndUserGetNameEvent (see page 24605)	This is type WebAdapt.TEndUserGetNameEvent.
TEndUserGetUserIDEvent (see page 24606)	This is type WebAdapt.TEndUserGetUserIDEvent.
TEndUserHandledEvent (see page 24606)	This is type WebAdapt.TEndUserHandledEvent.
TEndUserIsLoggedInEvent (see page 24606)	This is type WebAdapt.TEndUserIsLoggedInEvent.
TEndUserRightsEvent (see page 24607)	This is type WebAdapt.TEndUserRightsEvent.
TEndUserUserIDEvent (see page 24607)	This is type WebAdapt.TEndUserUserIDEvent.
TGetEOFEvent (see page 24607)	This is type WebAdapt.TGetEOFEvent.
TGetRecordCountEvent (see page 24608)	This is type WebAdapt.TGetRecordCountEvent.
TGetRecordEvent (see page 24608)	This is type WebAdapt.TGetRecordEvent.
TGetRecordIndexEvent (see page 24608)	This is type WebAdapt.TGetRecordIndexEvent.
THandledEvent (see page 24609)	This is type WebAdapt.THandledEvent.
TImageFieldGetHREFEvent (see page 24609)	This is type WebAdapt.TImageFieldGetHREFEvent.
TImageFieldGetImageEvent (see page 24609)	This is type WebAdapt.TImageFieldGetImageEvent.
TImageFieldGetParamsEvent (see page 24610)	This is type WebAdapt.TImageFieldGetParamsEvent.
TIterateRecordsEvent (see page 24610)	This is type WebAdapt.TIterateRecordsEvent.

TLoginUserIDEvent (see page 24610)	This is type WebAdapt.TLoginUserIDEvent.
TObjectList (see page 24611)	This is type WebAdapt.TObjectList.
TValidateAdapterFieldEvent (see page 24611)	This is type WebAdapt.TValidateAdapterFieldEvent.
TWebFormGetPageNameEvent (see page 24611)	This is type WebAdapt.TWebFormGetPageNameEvent.

173.5.1 WebAdapt.IActionFieldValue Type

Pascal

```
IActionFieldValue = AdaptReq.IActionFieldValue;
```

C++

```
AdaptReq.IActionFieldValue IActionFieldValue;
```

File

WebAdapt

Description

This is type WebAdapt.IActionFieldValue.

173.5.2 WebAdapt.IActionRequest Type

Pascal

```
IActionRequest = AdaptReq.IActionRequest;
```

C++

```
AdaptReq.IActionRequest IActionRequest;
```

File

WebAdapt

Description

This is type WebAdapt.IActionRequest.

173.5.3 WebAdapt.TAbstractWebRequestFile Type

Pascal

```
TAbstractWebRequestFile = HTTPApp.TAbstractWebRequestFile;
```

C++

```
HTTPApp.TAbstractWebRequestFile TAbstractWebRequestFile;
```

File

WebAdapt

Description

This is type WebAdapt.TAbstractWebRequestFile.

173.5.4 WebAdapt.TActionExecuteEvent Type

Pascal

```
TActionExecuteEvent = procedure (Sender: TObject; Params: TStrings) of object;
```

C++

```
(Sender: TObject; Params: TStrings) ( TActionExecuteEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TActionExecuteEvent.

173.5.5 WebAdapt.TActionExecuteHandledEvent Type

Pascal

```
TActionExecuteHandledEvent = procedure (Sender: TObject; Params: TStrings; var Handled:
Boolean) of object;
```

C++

```
(Sender: TObject; Params: TStrings; var Handled: Boolean) ( TActionExecuteHandledEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TActionExecuteHandledEvent.

173.5.6 WebAdapt.TActionGetEnabledEvent Type

Pascal

```
TActionGetEnabledEvent = procedure (Sender: TObject; var Enabled: Boolean) of object;
```

C++

```
(Sender: TObject; var Enabled: Boolean) ( TActionGetEnabledEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TActionGetEnabledEvent.

173.5.7 WebAdapt.TActionGetParamsEvent Type

Pascal

```
TActionGetParamsEvent = procedure (Sender: TObject; Params: TStrings) of object;
```

C++

```
(Sender: TObject; Params: TStrings) ( TActionGetParamsEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TActionGetParamsEvent.

173.5.8 WebAdapt.TActionRequestFieldsEvent Type

Pascal

```
TActionRequestFieldsEvent = procedure (Sender: TObject; ActionRequest: IActionRequest;
AdapterFields: TObjectList) of object;
```

C++

```
(Sender: TObject; ActionRequest: IActionRequest; AdapterFields: TObjectList) (
TActionRequestFieldsEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TActionRequestFieldsEvent.

173.5.9 WebAdapt.TActionRequestFieldsHandledEvent Type

Pascal

```
TActionRequestFieldsHandledEvent = procedure (Sender: TObject; ActionRequest: IActionRequest;
AdapterFields: TObjectList; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; ActionRequest: IActionRequest; AdapterFields: TObjectList; var Handled:
Boolean) ( TActionRequestFieldsHandledEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TActionRequestFieldsHandledEvent.

173.5.10 WebAdapt.TAdapterActionExecuteEvent Type

Pascal

```
TAdapterActionExecuteEvent = procedure (Sender: TObject; Action: TObject; Params: TStrings) of
object;
```

C++

```
(Sender: TObject; Action: TObject; Params: TStrings) ( TAdapterActionExecuteEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterActionExecuteEvent.

173.5.11 WebAdapt.TAdapterActionExecuteHandledEvent Type

Pascal

```
TAdapterActionExecuteHandledEvent = procedure (Sender: TObject; Action: TObject; Params:
TStrings; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; Action: TObject; Params: TStrings; var Handled: Boolean) (
TAdapterActionExecuteHandledEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterActionExecuteHandledEvent.

173.5.12 WebAdapt.TAdapterActionGetParamsEvent Type

Pascal

```
TAdapterActionGetParamsEvent = procedure (Sender: TObject; Action: TObject; Params: TStrings)
of object;
```

C++

```
(Sender: TObject; Action: TObject; Params: TStrings) ( TAdapterActionGetParamsEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterActionGetParamsEvent.

173.5.13 WebAdapt.TAdapterActionsClass Type

Pascal

```
TAdapterActionsClass = class of TAdapterActions;
```

C++

```
TAdapterActions TAdapterActionsClass;
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterActionsClass.

173.5.14 WebAdapt.TAdapterFieldBooleanValueEvent Type

Pascal

```
TAdapterFieldBooleanValueEvent = procedure (Sender: TObject; Value: Boolean) of object;
```

C++

```
(Sender: TObject; Value: Boolean) ( TAdapterFieldBooleanValueEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldBooleanValueEvent.

173.5.15 WebAdapt.TAdapterFieldFileEvent Type

Pascal

```
TAdapterFieldFileEvent = procedure (Sender: TObject; AFile: TAbstractWebRequestFile) of object;
```

C++

```
(Sender: TObject; AFile: TAbstractWebRequestFile) ( TAdapterFieldFileEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldFileEvent.

173.5.16 WebAdapt.TAdapterFieldFileListEvent Type

Pascal

```
TAdapterFieldFileListEvent = procedure (Sender: TObject; Files: TUpdateFileList) of object;
```

C++

```
(Sender: TObject; Files: TUpdateFileList) ( TAdapterFieldFileListEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldFileListEvent.

173.5.17 WebAdapt.TAdapterFieldGetBooleanEvent Type

Pascal

```
TAdapterFieldGetBooleanEvent = procedure (Sender: TObject; var Value: Boolean) of object;
```

C++

```
(Sender: TObject; var Value: Boolean) ( TAdapterFieldGetBooleanEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldGetBooleanEvent.

173.5.18 WebAdapt.TAdapterFieldGetMultiValueEvent Type

Pascal

```
TAdapterFieldGetMultiValueEvent = procedure (Sender: TObject; Index: Integer; var Value:
Variant) of object;
```

C++

```
(Sender: TObject; Index: Integer; var Value: Variant) ( TAdapterFieldGetMultiValueEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldGetMultiValueEvent.

173.5.19 WebAdapt.TAdapterFieldGetMultiValueStringEvent Type

Pascal

```
TAdapterFieldGetMultiValueStringEvent = procedure (Sender: TObject; Index: Integer; var Value:
string) of object;
```

C++

```
(Sender: TObject; Index: Integer; var Value: string) (
TAdapterFieldGetMultiValueStringEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldGetMultiValueStringEvent.

173.5.20 WebAdapt.TAdapterFieldGetStringEvent Type

Pascal

```
TAdapterFieldGetStringEvent = procedure (Sender: TObject; var Value: string) of object;
```

C++

```
(Sender: TObject; var Value: string) ( TAdapterFieldGetStringEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldGetStringEvent.

173.5.21 WebAdapt.TAdapterFieldGetValueCountEvent Type

Pascal

```
TAdapterFieldGetValueCountEvent = procedure (Sender: TObject; var Count: Integer) of object;
```

C++

```
(Sender: TObject; var Count: Integer) ( TAdapterFieldGetValueCountEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldGetValueCountEvent.

173.5.22 WebAdapt.TAdapterFieldGetValueEvent Type

Pascal

```
TAdapterFieldGetValueEvent = procedure (Sender: TObject; var Value: Variant) of object;
```

C++

```
(Sender: TObject; var Value: Variant) ( TAdapterFieldGetValueEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldGetValueEvent.

173.5.23 WebAdapt.TAdapterFieldHasValueEvent Type

Pascal

```
TAdapterFieldHasValueEvent = procedure (Sender: TObject; const Value: Variant; var HasValue:
Boolean) of object;
```

C++

```
(Sender: TObject; const Value: Variant; var HasValue: Boolean) ( TAdapterFieldHasValueEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldHasValueEvent.

173.5.24 WebAdapt.TAdapterFieldsClass Type

Pascal

```
TAdapterFieldsClass = class of TAdapterFields;
```

C++

```
TAdapterFields TAdapterFieldsClass;
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldsClass.

173.5.25 WebAdapt.TAdapterFieldStringEvent Type

Pascal

```
TAdapterFieldStringEvent = procedure (Sender: TObject; Value: string) of object;
```

C++

```
(Sender: TObject; Value: string) ( TAdapterFieldStringEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldStringEvent.

173.5.26 WebAdapt.TAdapterFieldStringsEvent Type

Pascal

```
TAdapterFieldStringsEvent = procedure (Sender: TObject; Values: TStrings) of object;
```

C++

```
(Sender: TObject; Values: TStrings) ( TAdapterFieldStringsEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldStringsEvent.

173.5.27 WebAdapt.TAdapterFieldValidateValueEvent Type

Pascal

```
TAdapterFieldValidateValueEvent = procedure (Sender: TObject; AValue: IActionFieldValue; var
Handled: Boolean) of object;
```

C++

```
(Sender: TObject; AValue: IActionFieldValue; var Handled: Boolean) (
TAdapterFieldValidateValueEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldValidateValueEvent.

173.5.28 WebAdapt.TAdapterFieldValueEvent Type

Pascal

```
TAdapterFieldValueEvent = procedure (Sender: TObject; Value: Variant) of object;
```

C++

```
(Sender: TObject; Value: Variant) ( TAdapterFieldValueEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldValueEvent.

173.5.29 WebAdapt.TAdapterHiddenFieldOptions Type

Pascal

```
TAdapterHiddenFieldOptions = set of TAdapterHiddenFieldOption;
```

C++

```
TAdapterHiddenFieldOption TAdapterHiddenFieldOptions;
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterHiddenFieldOptions.

173.5.30 WebAdapt.TEndUserGetNameEvent Type

Pascal

```
TEndUserGetNameEvent = procedure (Sender: TObject; var Name: string) of object;
```

C++

```
(Sender: TObject; var Name: string) ( TEndUserGetNameEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TEndUserGetNameEvent.

173.5.31 WebAdapt.TEndUserGetUserIDEvent Type

Pascal

```
TEndUserGetUserIDEvent = procedure (Sender: TObject; var UserID: Variant) of object;
```

C++

```
(Sender: TObject; var UserID: Variant) ( TEndUserGetUserIDEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TEndUserGetUserIDEvent.

173.5.32 WebAdapt.TEndUserHandledEvent Type

Pascal

```
TEndUserHandledEvent = procedure (Sender: TObject; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; var Handled: Boolean) ( TEndUserHandledEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TEndUserHandledEvent.

173.5.33 WebAdapt.TEndUserIsLoggedInEvent Type

Pascal

```
TEndUserIsLoggedInEvent = procedure (Sender: TObject; var IsLoggedIn: Boolean) of object;
```

C++

```
(Sender: TObject; var IsLoggedIn: Boolean) ( TEndUserIsLoggedInEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TEndUserIsLoggedInEvent.

173.5.34 WebAdapt.TEndUserRightsEvent Type

Pascal

```
TEndUserRightsEvent = procedure (Sender: TObject; Rights: TStrings; var HasRights, Handled:
Boolean) of object;
```

C++

```
(Sender: TObject; Rights: TStrings; var HasRights, Handled: Boolean) ( TEndUserRightsEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TEndUserRightsEvent.

173.5.35 WebAdapt.TEndUserUserIDEvent Type

Pascal

```
TEndUserUserIDEvent = procedure (Sender: TObject; UserID: Variant) of object;
```

C++

```
(Sender: TObject; UserID: Variant) ( TEndUserUserIDEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TEndUserUserIDEvent.

173.5.36 WebAdapt.TGetEOFEvent Type

Pascal

```
TGetEOFEvent = procedure (Sender: TObject; var Eof: Boolean) of object;
```

C++

```
(Sender: TObject; var Eof: Boolean) ( TGetEOFEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TGetEOFEvent.

173.5.37 WebAdapt.TGetRecordCountEvent Type

Pascal

```
TGetRecordCountEvent = procedure (Sender: TObject; var Count: Integer) of object;
```

C++

```
(Sender: TObject; var Count: Integer) ( TGetRecordCountEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TGetRecordCountEvent.

173.5.38 WebAdapt.TGetRecordEvent Type

Pascal

```
TGetRecordEvent = procedure (Sender: TObject; var Eof: Boolean) of object;
```

C++

```
(Sender: TObject; var Eof: Boolean) ( TGetRecordEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TGetRecordEvent.

173.5.39 WebAdapt.TGetRecordIndexEvent Type

Pascal

```
TGetRecordIndexEvent = procedure (Sender: TObject; var Index: Integer) of object;
```

C++

```
(Sender: TObject; var Index: Integer) ( TGetRecordIndexEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TGetRecordIndexEvent.

173.5.40 WebAdapt.THandledEvent Type

Pascal

```
THandledEvent = procedure (Sender: TObject; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; var Handled: Boolean) ( THandledEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.THandledEvent.

173.5.41 WebAdapt.TImageFieldGetHREFEvent Type

Pascal

```
TImageFieldGetHREFEvent = procedure (Sender: TObject; var HREF: string) of object;
```

C++

```
(Sender: TObject; var HREF: string) ( TImageFieldGetHREFEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TImageFieldGetHREFEvent.

173.5.42 WebAdapt.TImageFieldGetImageEvent Type

Pascal

```
TImageFieldGetImageEvent = procedure (Sender: TObject; Params: TStrings; var MimeType: string;
var Image: TStream; var Owned: Boolean) of object;
```

C++

```
(Sender: TObject; Params: TStrings; var MimeType: string; var Image: TStream; var Owned:
Boolean) ( TImageFieldGetImageEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TImageFieldGetImageEvent.

173.5.43 WebAdapt.TImageFieldGetParamsEvent Type

Pascal

```
TImageFieldGetParamsEvent = procedure (Sender: TObject; Params: TStrings) of object;
```

C++

```
(Sender: TObject; Params: TStrings) ( TImageFieldGetParamsEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TImageFieldGetParamsEvent.

173.5.44 WebAdapt.TIterateRecordsEvent Type

Pascal

```
TIterateRecordsEvent = procedure (Sender: TObject; Action: TIteratorMethod; var Eof: Boolean)
of object;
```

C++

```
(Sender: TObject; Action: TIteratorMethod; var Eof: Boolean) ( TIterateRecordsEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TIterateRecordsEvent.

173.5.45 WebAdapt.TLoginUserIDEvent Type

Pascal

```
TLoginUserIDEvent = procedure (Sender: TObject; UserID: Variant) of object;
```

C++

```
(Sender: TObject; UserID: Variant) ( TLoginUserIDEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TLoginUserIDEvent.

173.5.46 WebAdapt.TObjectList Type

Pascal

```
TObjectList = Contnrs.TObjectList;
```

C++

```
Contnrs.TObjectList TObjectList;
```

File

WebAdapt

Description

This is type WebAdapt.TObjectList.

173.5.47 WebAdapt.TValidateAdapterFieldEvent Type

Pascal

```
TValidateAdapterFieldEvent = procedure (Sender: TObject; Value: Variant; var Handled: Boolean)
of object;
```

C++

```
(Sender: TObject; Value: Variant; var Handled: Boolean) ( TValidateAdapterFieldEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TValidateAdapterFieldEvent.

173.5.48 WebAdapt.TWebFormGetPageNameEvent Type

Pascal

```
TWebFormGetPageNameEvent = procedure (Sender: TObject; APageIndex: Integer; var Value: string)
of object;
```

C++

```
(Sender: TObject; APageIndex: Integer; var Value: string) ( TWebFormGetPageNameEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.TWebFormGetPageNameEvent.

173.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
NotifyPossibleHTMLChange (see page 24612)	This is variable WebAdapt.NotifyPossibleHTMLChange.

173.6.1 WebAdapt.NotifyPossibleHTMLChange Variable

Pascal

```
NotifyPossibleHTMLChange: procedure;
```

C++

```
procedure NotifyPossibleHTMLChange;
```

File

WebAdapt

Description

This is variable WebAdapt.NotifyPossibleHTMLChange.

173.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
sActionRequestLong (see page 24613)	This is constant WebAdapt.sActionRequestLong.
sCurrentPage (see page 24613)	This is constant WebAdapt.sCurrentPage.
sGotoPageAction (see page 24614)	This is constant WebAdapt.sGotoPageAction.
sImageRequest (see page 24614)	This is constant WebAdapt.sImageRequest.
sLoginPassword (see page 24614)	This is constant WebAdapt.sLoginPassword.
sLoginUserName (see page 24615)	This is constant WebAdapt.sLoginUserName.
sNextPageAction (see page 24615)	This is constant WebAdapt.sNextPageAction.
sPrevPageAction (see page 24615)	This is constant WebAdapt.sPrevPageAction.

173.7.1 WebAdapt.sActionRequestLong Constant

Pascal

```
sActionRequestLong = '__action';
```

C++

```
sActionRequestLong = '__action';
```

File

WebAdapt

Description

This is constant WebAdapt.sActionRequestLong.

173.7.2 WebAdapt.sCurrentPage Constant

Pascal

```
sCurrentPage = '_cp';
```

C++

```
sCurrentPage = '_cp';
```

File

WebAdapt

Description

This is constant WebAdapt.sCurrentPage.

173.7.3 WebAdapt.sGotoPageAction Constant

Pascal

```
sGotoPageAction = 'GotoPage';
```

C++

```
sGotoPageAction = 'GotoPage';
```

File

WebAdapt

Description

This is constant WebAdapt.sGotoPageAction.

173.7.4 WebAdapt.sImageRequest Constant

Pascal

```
sImageRequest = '__img';
```

C++

```
sImageRequest = '__img';
```

File

WebAdapt

Description

This is constant WebAdapt.sImageRequest.

173.7.5 WebAdapt.sLoginPassword Constant

Pascal

```
sLoginPassword = 'Password';
```

C++

```
sLoginPassword = 'Password';
```

File

WebAdapt

Description

This is constant WebAdapt.sLoginPassword.

173.7.6 WebAdapt.sLoginUserName Constant

Pascal

```
sLoginUserName = 'UserName';
```

C++

```
sLoginUserName = 'UserName';
```

File

WebAdapt

Description

This is constant WebAdapt.sLoginUserName.

173.7.7 WebAdapt.sNextPageAction Constant

Pascal

```
sNextPageAction = 'NextPage';
```

C++

```
sNextPageAction = 'NextPage';
```

File

WebAdapt

Description

This is constant WebAdapt.sNextPageAction.

173.7.8 WebAdapt.sPrevPageAction Constant

Pascal

```
sPrevPageAction = 'PrevPage';
```

C++

```
sPrevPageAction = 'PrevPage';
```

File

WebAdapt

Description

This is constant WebAdapt.sPrevPageAction.

174 WebBroker Namespace

This is namespace WebBroker.

Topics

Name	Description
Classes (see page 24618)	The following table lists classes in this documentation.
Types (see page 24625)	The following table lists types in this documentation.
Variables (see page 24626)	The following table lists variables in this documentation.

Classes

	Name	Description
	TWebApplication (see page 24618)	TWebApplication encapsulates a Web server application.

Types

Name	Description
THandleShutdownException (see page 24625)	This is type WebBroker.THandleShutdownException.
TServerExceptionEvent (see page 24625)	This is type WebBroker.TServerExceptionEvent.

Variables

Name	Description
Application (see page 24626)	This is variable WebBroker.Application.
HandleShutdownException (see page 24626)	This is variable WebBroker.HandleShutdownException.

174.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TWebApplication (see page 24618)	TWebApplication encapsulates a Web server application.

174.1.1 TWebApplication Class

TWebApplication encapsulates a Web server application.

Class Hierarchy

Pascal

```
TWebApplication = class(TWebRequestHandler);
```

C++

```
class TWebApplication : public TWebRequestHandler;
```

File

WebBroker

Description

TWebApplication provides the fundamental behavior of a Web server application. TWebApplication provides methods to create TWebRequest and TWebResponse objects that represent HTTP request messages and their responses, pass these objects to the dispatcher so that the response can be filled in, and send the response back to the Web client.

Each Web server project automatically declares an Application (see page 24626) variable of type TWebApplication as the instance of the application. For CGI applications, this variable is initialized to a TCGIApplication object in the CGIApp (see page 2553) unit. For Apache applications, this variable is initialized to a TApacheApplication object in the ApacheApp unit. For ISAPI applications, this variable is initialized to a TISAPIApplication object in the ISAPIApp (see page 15647) unit.

The Application (see page 24626) variable is added to a project by selecting File | New from the menu of the IDE main window and selecting Web Server Application (see page 24626) in the New Items dialog. Do not add the Forms (see page 13657) unit to the uses list of the project (Delphi) or include Forms.hpp in the project source file (C++). These actions declare a conflicting Application (see page 24626) variable of type TApplication.In addition, do not add the ApacheApp or ISAPIApp (see page 15647) unit to a CGI application or the CGIApp (see page 2553) unit to an Apache, ISAPI or NSAPI application. The CGIApp (see page 2553) unit, the ISAPIApp (see page 15647) unit, and the ApacheApp unit each create the initial instance of the Application (see page 24626) variable. Including more than one of these units in the uses list causes the memory for the object created in the first unit to be lost.

See Also

Application (see page 24626)

TCGIApplication (see page 2554)

TISAPIApplication (see page 15648)

TApacheApplication

TWebDispatcher (see page 15147)

TWebRequest (see page 15161)

TWebResponse (see page 15191)

174.1.1.1 TWebApplication Members

The following tables list the members exposed by TWebApplication.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActivateWebModules (see page 25453)	Activates a Web module for the application and returns it.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DeactivateWebModules (see page 25453)	Deactivates the Web module of the application.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleRequest (see page 25454)	Responds to an HTTP request message received by the server.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property

	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddWebModuleFactory (see page 25453)	Adds a Web module factory to the request handler.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 24622)	Creates an instance of TWebApplication (see page 24618).
	CreateForm (see page 24623)	Registers the single Web module with the Web application.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 24622)	Destroys an instance of TWebApplication (see page 24618).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.

	GetParentComponent (see page 2778)	Returns the containing Component.
	HandleException (see page 25454)	Responds to exceptions related to TWebRequestHandler (see page 25448) objects.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
V	Initialize (see page 24623)	Introduces an interface for a method that initializes the application.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	Run (see page 24623)	Executes the Web application.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActiveCount (see page 25455)	Indicates the number of active connections currently being serviced by the application.
	CacheConnections (see page 25455)	Determines whether the Web modules used by the application are cached for re-use.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	InactiveCount (see page 25456)	Indicates the number of Web modules cached for later reuse.
	MaxConnections (see page 25457)	Sets or returns the MaxConnections used by the socket.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Title (see page 24624)	Specifies a title for the Web application.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	WebModuleClass (see page 25457)	Contains a reference to a Web module.
R	WebModuleFactory (see page 25457)	A list of Web module factory objects.
R	WebModuleFactoryCount (see page 25458)	The number of Web module factories in use by the request handler.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

174.1.1.2 TWebApplication.Create Constructor

Creates an instance of TWebApplication (see page 24618).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebApplication(TComponent * AOwner);
```

Description

Do not call Create. The single Web application object named Application (see page 24626) is instantiated automatically when the CGIApp (see page 2553) unit, the ISAPIApp (see page 15647) unit, or the ApacheApp unit is included in the uses list of the project.

Create allocates the helper objects used by TWebApplication (see page 24618).

See Also

Create (see page 2766)

174.1.1.3 TWebApplication.Destroy Destructor

Destroys an instance of TWebApplication (see page 24618).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebApplication();
```

Description

Never call the Destroy or Free method. The application is freed automatically when the application stops running.

Destroy frees the helper objects that were created at startup, as well as all Web modules created by the application.

See Also

Free (see page 22300)

Run (see page 24623)

174.1.1.4 TWebApplication Methods

The methods of the TWebApplication class are listed here.

Public Methods

	Name	Description
	CreateForm (see page 24623)	Registers the single Web module with the Web application.
	Initialize (see page 24623)	Introduces an interface for a method that initializes the application.
	Run (see page 24623)	Executes the Web application.

174.1.1.4.1 TWebApplication.CreateForm Method

Registers the single Web module with the Web application.

Pascal

```
procedure CreateForm(InstanceClass: TComponentClass; var Reference); virtual;
```

C++

```
virtual __fastcall CreateForm(TComponentClass InstanceClass,  Reference);
```

Description

Do not call CreateForm in an application. The module manager calls CreateForm internally to register the Web module for the Web application.

CreateForm registers the class of the Web module for the application.

See Also

TWebModule (see page 15154)

174.1.1.4.2 TWebApplication.Initialize Method

Introduces an interface for a method that initializes the application.

Pascal

```
procedure Initialize; virtual;
```

C++

```
virtual __fastcall Initialize();
```

Description

The Initialize method for TWebApplication (see page 24618) does nothing. Initialize is the first method called for every project. Descendants of TWebApplication (see page 24618) that must perform any initialization before the application starts running override this method for that purpose.

Note: Although Initialize is the first method called in the project source code, it is not the first code that is executed in an application.

See Also

Run (see page 24623)

174.1.1.4.3 TWebApplication.Run Method

Executes the Web application.

Pascal

```
procedure Run; virtual;
```

C++

```
virtual __fastcall Run();
```

Description

When the application is executed, the application's Run method is calledautomatically.

The Run method for TWebApplication (see page 24618) sets up the default error handler and shut down sequence for the

application. Descendants of TWebApplication (see page 24618) override the Run method to receive HTTP request messages, create the TWebRequest and TWebResponse objects for incoming requests, and call the HandleRequest method to process those requests.

See Also

TWebRequest (see page 15161)

TWebResponse (see page 15191)

174.1.1.5 TWebApplication Properties

The properties of the TWebApplication class are listed here.

Public Properties

	Name	Description
	Title (see page 24624)	Specifies a title for the Web application.

174.1.1.5.1 TWebApplication.Title Property

Specifies a title for the Web application.

Pascal

```
property Title: string;
```

C++

```
__property AnsiString Title;
```

Description

Use the Title property to assign a title to your Web application. This title can be used in HTTP messages that the application sends to identify the application. The default value is taken from the executable DLL/EXE .so filename at runtime.

Do not confuse Title with the Name property. Name is the identifier for the Web application when it is referred to in code.

See Also

Name (see page 2805)

174.2 Types

The following table lists types in this documentation.

Types

Name	Description
THandleShutdownException (see page 24625)	This is type WebBroker.THandleShutdownException.
TServerExceptionEvent (see page 24625)	This is type WebBroker.TServerExceptionEvent.

174.2.1 WebBroker.THandleShutdownException Type

Pascal

```
THandleShutdownException = procedure (E: Exception);
```

C++

```
(E: Exception) ( THandleShutdownException)();
```

File

WebBroker

Description

This is type WebBroker.THandleShutdownException.

174.2.2 WebBroker.TServerExceptionEvent Type

Pascal

```
TServerExceptionEvent = procedure (E: Exception; wr: TWebResponse) of object;
```

C++

```
(E: Exception; wr: TWebResponse) ( TServerExceptionEvent)();
```

File

WebBroker

Description

This is type WebBroker.TServerExceptionEvent.

174.3 Variables

The following table lists variables in this documentation.

Variables

Name	Description
Application (see page 24626)	This is variable WebBroker.Application.
HandleShutdownException (see page 24626)	This is variable WebBroker.HandleShutdownException.

174.3.1 WebBroker.Application Variable

Pascal

```
Application: TWebApplication = nil;
```

C++

```
TWebApplication Application = nil;
```

File

WebBroker

Description

This is variable WebBroker.Application.

174.3.2 WebBroker.HandleShutdownException Variable

Pascal

```
HandleShutdownException: THandleShutdownException;
```

C++

```
THandleShutdownException HandleShutdownException;
```

File

WebBroker

Description

This is variable WebBroker.HandleShutdownException.

175 WebBrokerSOAP Namespace

This is namespace WebBrokerSOAP.

Topics

Name	Description
Classes (see page 24628)	The following table lists classes in this documentation.
Functions (see page 24636)	The following table lists functions in this documentation.

Classes

	Name	Description
	THTTPSoapDispatcher (see page 24628)	THTTPSoapDispatcher responds to SOAP messages by forwarding them to an invoker for interpretation.

Functions

	Name	Description
	GetSOAPWebModule (see page 24636)	Returns the Web module that contains the SOAP dispatcher that is processing the current request.

175.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	THTTPSoapDispatcher (see page 24628)	THTTPSoapDispatcher responds to SOAP messages by forwarding them to an invoker for interpretation.

175.1.1 THTTPSoapDispatcher Class

THTTPSoapDispatcher responds to SOAP messages by forwarding them to an invoker for interpretation.

Class Hierarchy

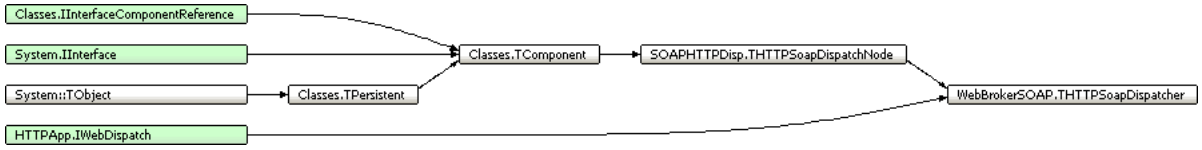

Pascal

```
THTTPSoapDispatcher = class(THTTPSoapDispatchNode, IWebDispatch);
```

C++

```
class THTTPSoapDispatcher : public THTTPSoapDispatchNode, public IWebDispatch;
```

File

WebBrokerSOAP

Description

Use THTTPSoapDispatcher in an application that publishes a Web Service using SOAP. THTTPSoapDispatcher responds to all SOAP-based calls made to the invokable interfaces that are registered in the application. These interfaces are registered by calling the RegisterInterface method of the invocation registry (InvRegistry).

THTTPSoapDispatcher only acts as a dispatcher, receiving incoming messages and forwarding them on to another component that handles the task of interpreting and executing them. The component that interprets and executes SOAP message is identified by the Dispatcher property. Dispatcher is the interface of an invoker component, which interprets the SOAP message, identifies the invokable interface that is the target of the call, executes the call, and assembles the content of the response message.

THTTPSoapDispatcher automatically registers itself with the Web module (or Web dispatcher) as an auto-dispatching object. This means that the Web module or Web dispatcher forwards all incoming HTTP messages directed to the SOAP dispatcher without requiring the use of Web action items.

See Also

THTTPSOAPPascalInvoker

THTTPSOAPCppInvoker

IInvokable (see page 22398)

InvRegistry

TInvokableClassRegistry (see page 15526)

Using the SOAP Application Wizard

Writing Servers that Support Web Services

175.1.1.1 THTTPSoapDispatcher Members

The following tables list the members exposed by THTTPSoapDispatcher.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.
	FSoapDispatcher (see page 19943)	This is FSoapDispatcher, a member of class THTTPSoapDispatchNode.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DispatchEnabled (see page 24632)	Indicates whether the HTTP SOAP dispatcher is enabled to respond to incoming SOAP messages.
	DispatchMask (see page 24633)	Returns a mask that matches the path portion of any URL to which this object responds.
	DispatchMethodType (see page 24633)	Returns the method type of all HTTP messages to which this component responds.
	DispatchRequest (see page 24634)	Responds to an incoming message and generates the appropriate response message.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 19944)	Responds when objects are added or removed from the Web module.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 24631)	Creates a new THTTPSoapDispatcher (see page 24628) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 24632)	Frees the memory associated with the THTTPSoapDispatcher (see page 24628) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DispatchSOAP (see page 19944)	Forwards the SOAP message on to the invoker for interpretation and execution.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Enabled (see page 15232)	
R	Mask (see page 15233)	
R	MethodType (see page 15233)	This is MethodType, a member of class IWebDispatch.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Dispatcher (see page 19945)	Specifies the interface of the component that interprets SOAP messages.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	WebDispatch (see page 24634)	Lets the Web dispatcher identify what HTTP messages should be forwarded to the HTTP SOAP dispatcher.

175.1.1.2 THTTPSoapDispatcher.Create Constructor

Creates a new THTTPSoapDispatcher (see page 24628) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall THTTPSoapDispatcher(TComponent * AOwner);
```

Description

Most applications do not need to call the THTTPSoapDispatcher (see page 24628) constructor. Typically the HTTP SOAP dispatcher is added to a Web module at design time, so that the constructor is called automatically.

The Owner parameter specifies the component (usually the Web module) that is responsible for freeing the THTTPSoapDispatcher (see page 24628) instance.

See Also

Owner (see page 2807)

175.1.1.3 THTTPSoapDispatcher.Destroy Destructor

Frees the memory associated with the THTTPSoapDispatcher (see page 24628) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THTTPSoapDispatcher();
```

Description

Do not call Destroy directly. Instead,call Free, which verifies that the THTTPSoapDispatcher (see page 24628) reference is not nil before calling Destroy.

See Also

Free (see page 22300)

175.1.1.4 THTTPSoapDispatcher Methods

The methods of the THTTPSoapDispatcher class are listed here.

Protected Methods

	Name	Description
	DispatchEnabled (see page 24632)	Indicates whether the HTTP SOAP dispatcher is enabled to respond to incoming SOAP messages.
	DispatchMask (see page 24633)	Returns a mask that matches the path portion of any URL to which this object responds.
	DispatchMethodType (see page 24633)	Returns the method type of all HTTP messages to which this component responds.
	DispatchRequest (see page 24634)	Responds to an incoming message and generates the appropriate response message.

175.1.1.4.1 THTTPSoapDispatcher.DispatchEnabled Method

Indicates whether the HTTP SOAP dispatcher is enabled to respond to incoming SOAP messages.

Pascal

```
function DispatchEnabled: Boolean;
```

C++

```
__fastcall Boolean DispatchEnabled();
```

Description

The DispatchEnabled method must be implemented by all auto-dispatching objects to allow Web Module (or Web Dispatcher) to determine whether it can receive incoming messages. In THTTPSoapDispatcher (see page 24628), DispatchEnabled always returns true, because the HTTP SOAP dispatcher can't be disabled.

See Also

WebDispatch (see page 24634)

175.1.1.4.2 THTTPSoapDispatcher.DispatchMask Method

Returns a mask that matches the path portion of any URL to which this object responds.

Pascal

```
function DispatchMask: TMask;
```

C++

```
__fastcall TMask DispatchMask();
```

Description

The DispatchMask method must be implemented by all auto-dispatching objects to allow Web Module (or Web Dispatcher) to identify the HTTP messages it should forward to the this component. It returns the Mask (see page 15797) property of the TWebDispatch object specified by the WebDispatch (see page 24634) property.

See Also

WebDispatch (see page 24634)

Mask (see page 1482)

175.1.1.4.3 THTTPSoapDispatcher.DispatchMethodType Method

Returns the method type of all HTTP messages to which this component responds.

Pascal

```
function DispatchMethodType: TMethodType;
```

C++

```
__fastcall TMethodType DispatchMethodType();
```

Description

The DispatchMethodType method must be implemented by all auto-dispatching objects to allow Web Module (or Web Dispatcher) to identify the HTTP messages it should forward to the this component. It returns the MethodType property of the TWebDispatch object specified by the WebDispatch (see page 24634) property.

See Also

MethodType

WebDispatch (see page 24634)

175.1.1.4.4 THTTPSoapDispatcher.DispatchRequest Method

Responds to an incoming message and generates the appropriate response message.

Pascal

```
function DispatchRequest(Sender: TObject; Request: TWebRequest; Response: TWebResponse):
Boolean;
```

C++

```
__fastcall Boolean DispatchRequest(TObject * Sender, TWebRequest Request, TWebResponse
Response);
```

Description

The Web Module (or Web Dispatcher) calls DispatchRequest when it identifies an HTTP message that matches the properties of the WebDispatch (see page 24634) property. DispatchRequest passes the request on to the component whose interface is the value of the Dispatcher property. That object interprets the request message, calls the appropriate invokable interface, and returns the content of the response message.

Sender is not used in the DispatchRequest method.

Request contains information about the incoming SOAP message.

Response is a TWebResponse object that is filled out to indicate the response that should be returned to the Web client.

DispatchRequest returns true if it successfully passes the request on to an invoker and that invoker handles the request and fills out the Response parameter. It returns false if it could not handle the request.

See Also

Dispatcher (see page 19945)

WebDispatch (see page 24634)

175.1.1.5 THTTPSoapDispatcher Properties

The properties of the THTTPSoapDispatcher class are listed here.

Published Properties

	Name	Description
	WebDispatch (see page 24634)	Lets the Web dispatcher identify what HTTP messages should be forwarded to the HTTP SOAP dispatcher.

175.1.1.5.1 THTTPSoapDispatcher.WebDispatch Property

Lets the Web dispatcher identify what HTTP messages should be forwarded to the HTTP SOAP dispatcher.

Pascal

```
property WebDispatch: TWebDispatch;
```

C++

```
__property TWebDispatch WebDispatch;
```

Description

Use WebDispatch to specify the SOAP messages to which this component responds.

The TWebDispatch object that implements WebDispatch includes properties that let you specify the path portion of URLs to

which the Web Service application responds, as well as the values allowed in the types of messages it accepts. By default, WebDispatch enables the HTTP SOAP dispatcher to respond to any URL whose path portion begins with the string "soap".

Note: You can't use WebDispatch to disable the THTTPSoapDispatcher (see page 24628) object. Setting its Enabled property to false does nothing.

See Also

TWebModule (see page 15154)

TWebDispatcher (see page 15147)

175.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetSOAPWebModule (see page 24636)	Returns the Web module that contains the SOAP dispatcher that is processing the current request.

175.2.1 WebBrokerSOAP.GetSOAPWebModule Function

Returns the Web module that contains the SOAP dispatcher that is processing the current request.

Pascal

```
function GetSOAPWebModule: TWebModule;
```

C++

```
TWebModule GetSOAPWebModule();
```

File

WebBrokerSOAP

Description

Call GetSOAPWebModule to access the application Web module of a Web Services application. This is the Web module that contains the SOAP dispatcher (THTTPSoapDispatcher (see page 24628)).

See Also

THTTPSOAPDispatcher

176 WebCntxt Namespace

This is namespace WebCntxt.

Topics

Name	Description
Classes (see page 24639)	The following table lists classes in this documentation.
Interfaces (see page 24672)	The following table lists interfaces in this documentation.
Functions (see page 24687)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 24688)	The following table lists structs, records, enums in this documentation.
Types (see page 24689)	The following table lists types in this documentation.
Variables (see page 24691)	The following table lists variables in this documentation.

Classes

	Name	Description
	TAbstractAdapterRequest (see page 24639)	TAbstractAdapterRequest is the base class for adapter request objects.
	TAbstractAdapterResponse (see page 24642)	TAbstractAdapterResponse is the base class for adapter response objects.
	TAbstractWebContext (see page 24644)	TAbstractWebContext is the base class for objects that represent the current state of a WebSnap application.
	TWebContext (see page 24660)	TWebContext represents the current state of a WebSnap application.

Functions

	Name	Description
	WebContext (see page 24687)	Returns the current Web context

Interfaces

	Name	Description
	IWebEndUser (see page 24672)	IWebEndUser provides access information about the current end user in a WebSnap application.
	IWebPageInfo (see page 24680)	IWebPageInfo describes a Web page in a WebSnap application.

Structs, Records, Enums

	Name	Description
	TFindPageOption (see page 24688)	This is record WebCntxt.TFindPageOption.

Types

Name	Description
TFindPageOptions (see page 24689)	This is type WebCntxt.TFindPageOptions.

TGetWebContextProc (see page 24689)	This is type WebCntxt.TGetWebContextProc.
TGetWebEndUserProc (see page 24690)	This is type WebCntxt.TGetWebEndUserProc.
TSetWebContextProc (see page 24690)	This is type WebCntxt.TSetWebContextProc.

Variables

Name	Description
GetModuleFileNameProc (see page 24691)	This is variable WebCntxt.GetModuleFileNameProc.
GetWebContextProc (see page 24691)	This is variable WebCntxt.GetWebContextProc.
GetWebEndUserProc (see page 24692)	This is variable WebCntxt.GetWebEndUserProc.
SetWebContextProc (see page 24692)	This is variable WebCntxt.SetWebContextProc.

176.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TAbstractAdapterRequest (see page 24639)	TAbstractAdapterRequest is the base class for adapter request objects.
	TAbstractAdapterResponse (see page 24642)	TAbstractAdapterResponse is the base class for adapter response objects.
	TAbstractWebContext (see page 24644)	TAbstractWebContext is the base class for objects that represent the current state of a WebSnap application.
	TWebContext (see page 24660)	TWebContext represents the current state of a WebSnap application.

176.1.1 TAbstractAdapterRequest Class

TAbstractAdapterRequest is the base class for adapter request objects.

Class Hierarchy

Pascal

```
TAbstractAdapterRequest = class(TInterfacedObject);
```

C++

```
class TAbstractAdapterRequest : public TInterfacedObject;
```

File

WebCntxt

Description

TAbstractAdapterRequest is the base class for the object that adapters use to obtain information about the current HTTP request message. An application interacts with an adapter request object through interfaces implemented by various descendants of TAbstractAdapterRequest. These descendant classes are considered internal to WebSnap and are not documented.

See Also

TAbstractWebContext (see page 24644)

IActionRequest (see page 935)

IAdapterModes (see page 944)

IactionFieldValues

IEnumActionFieldValueRecords (see page 957)

IAdapterRecordKeys (see page 949)

IActionFieldValuesOfAdapter (see page 933)

IAdapterDispatchParams (see page 19655)

Iadaptermodes

iactionfieldvalues

ienumactionfieldvaluerecords

iadapterrecordkeys

iactionfieldvaluesofadapter

176.1.1.1 TAbstractAdapterRequest Members

The following tables list the members exposed by TAbstractAdapterRequest.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Protected Properties

	Name	Description
R	Request (see page 24642)	This is Request, a member of class TAbstractAdapterRequest.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 24641)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

176.1.1.2 TAbstractAdapterRequest.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TAbstractAdapterRequest();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

176.1.1.3 TAbstractAdapterRequest Properties

The properties of the TAbstractAdapterRequest class are listed here.

Protected Properties

	Name	Description
R	Request (see page 24642)	This is Request, a member of class TAbstractAdapterRequest.

176.1.1.3.1 TAbstractAdapterRequest.Request Property

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

This is Request, a member of class TAbstractAdapterRequest.

176.1.2 TAbstractAdapterResponse Class

TAbstractAdapterResponse is the base class for adapter response objects.

Class Hierarchy

Pascal

```
TAbstractAdapterResponse = class(TInterfacedObject);
```

C++

```
class TAbstractAdapterResponse : public TInterfacedObject;
```

File

WebCntxt

Description

TAbstractAdapterResponse is the base class for the object that adapters use to obtain information about the current HTTP response message. An application interacts with an adapter response object through interfaces implemented by various descendants of TAbstractAdapterResponse. These descendant classes are considered internal to WebSnap and are not documented.

See Also

TAbstractWebContext (see page 24644)

IActionResponse (see page 941)

176.1.2.1 TAbstractAdapterResponse Members

The following tables list the members exposed by TAbstractAdapterResponse.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Protected Properties

	Name	Description
R	Response (see page 24644)	This is Response, a member of class TAbstractAdapterResponse.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 24644)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

176.1.2.2 TAbstractAdapterResponse.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TAbstractAdapterResponse();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

176.1.2.3 TAbstractAdapterResponse Properties

The properties of the TAbstractAdapterResponse class are listed here.

Protected Properties

	Name	Description
R	Response (see page 24644)	This is Response, a member of class TAbstractAdapterResponse.

176.1.2.3.1 TAbstractAdapterResponse.Response Property

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

This is Response, a member of class TAbstractAdapterResponse.

176.1.3 TAbstractWebContext Class

TAbstractWebContext is the base class for objects that represent the current state of a WebSnap application.

Class Hierarchy

System::TObject → WebCntxt.TAbstractWebContext

Pascal

```
TAbstractWebContext = class(TObject);
```

C++

```
class TAbstractWebContext : public TObject;
```

File

WebCntxt

Description

A Web context stores information about the components that are available in a WebSnap application and about the current request that is being processed. Applications do not create instances of the Web context. Rather, they access the current Web context object by calling the global WebContext (see page 24687) function.

See Also

WebContext (see page 24687)

TWebContext (see page 24660)

176.1.3.1 TAbstractWebContext Members

The following tables list the members exposed by TAbstractWebContext.

Protected Methods

	Name	Description
	GetAdapterRequest (see page 24651)	This is GetAdapterRequest, a member of class TAbstractWebContext.
	GetAdapterResponse (see page 24652)	This is GetAdapterResponse, a member of class TAbstractWebContext.
	GetDispatchedPageName (see page 24652)	This is GetDispatchedPageName, a member of class TAbstractWebContext.
	GetEndUser (see page 24652)	This is GetEndUser, a member of class TAbstractWebContext.
	GetPageCount (see page 24652)	
	GetPageInfo (see page 24653)	
	GetRequestedPageName (see page 24653)	
	GetSession (see page 24653)	
	GetWebModules (see page 24653)	
	GetWebRequest (see page 24654)	
	GetWebResponse (see page 24654)	
	SetAdapterRequest (see page 24654)	This is SetAdapterRequest, a member of class TAbstractWebContext.
	SetAdapterResponse (see page 24654)	This is SetAdapterResponse, a member of class TAbstractWebContext.
	SetDispatchedPageName (see page 24655)	This is SetDispatchedPageName, a member of class TAbstractWebContext.
	SetRequestedPageName (see page 24655)	This is SetRequestedPageName, a member of class TAbstractWebContext.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 24646)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 24647)	Frees an instance of TAbstractWebContext (see page 24644).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
A	FindApplicationModule (see page 24648)	Returns the application module instance.
A	FindDefaultPageInfo (see page 24648)	Retrieves the Web page information object for a specified class of Web module.
A	FindModuleClass (see page 24649)	Returns an instance of a Web module, given its class reference.
A	FindModuleName (see page 24649)	Returns an instance of a Web module, given its class name.
A	FindPage (see page 24650)	Retrieves a Web page module given its name.
A	FindPageInfo (see page 24650)	Retrieves a Web page information object given the name of the corresponding Web page module.
A	FindVariableContainer (see page 24651)	Returns an instance of a Web data module, given its class name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	AdapterRequest (see page 24655)	Provides access to the adapter request object that manages access to the current request.
R	AdapterResponse (see page 24656)	Provides access to the adapter response object that manages access to the current response.
	DispatchedPageName (see page 24656)	Specifies the name of the Web page module that is producing content to respond to the current request.
R	EndUser (see page 24657)	Provides access to information about the end user that sent the current request.
R	PageCount (see page 24657)	Indicates the number of registered Web pages available in the WebSnap application.
R	Pages (see page 24658)	Lists the page information interfaces of the registered Web pages available in the WebSnap application.
R	Request (see page 24658)	Represents the current HTTP request that the application is handling.
	RequestedPageName (see page 24658)	Indicates the name of the Web page module to which the current request is addressed.
R	Response (see page 24659)	Represents the current HTTP response that the application is assembling.
R	Session (see page 24659)	Provides access to the current Web session.
R	WebModules (see page 24660)	Lists all the Modules in the WebSnap application.

176.1.3.2 TAbstractWebContext.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TAbstractWebContext();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

176.1.3.3 TAbstractWebContext.Destroy Destructor

Frees an instance of TAbstractWebContext (see page 24644).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAbstractWebContext();
```

Description

Do not free the global Web context object. This object is automatically maintained by the WebSnap application.

The TAbstractWebContext (see page 24644) destructor unregisters itself with the global WebContext (see page 24687) function so that the WebContext (see page 24687) function never returns a Web context that no longer exists.

See Also

WebContext (see page 24687)

176.1.3.4 TAbstractWebContext Methods

The methods of the TAbstractWebContext class are listed here.

Protected Methods

	Name	Description
	GetAdapterRequest (see page 24651)	This is GetAdapterRequest, a member of class TAbstractWebContext.
	GetAdapterResponse (see page 24652)	This is GetAdapterResponse, a member of class TAbstractWebContext.
	GetDispatchedPageName (see page 24652)	This is GetDispatchedPageName, a member of class TAbstractWebContext.
	GetEndUser (see page 24652)	This is GetEndUser, a member of class TAbstractWebContext.
	GetPageCount (see page 24652)	
	GetPageInfo (see page 24653)	
	GetRequestedPageName (see page 24653)	
	GetSession (see page 24653)	
	GetWebModules (see page 24653)	

	GetWebRequest (see page 24654)	
	GetWebResponse (see page 24654)	
	SetAdapterRequest (see page 24654)	This is SetAdapterRequest, a member of class TAbstractWebContext.
	SetAdapterResponse (see page 24654)	This is SetAdapterResponse, a member of class TAbstractWebContext.
	SetDispatchedPageName (see page 24655)	This is SetDispatchedPageName, a member of class TAbstractWebContext.
	SetRequestedPageName (see page 24655)	This is SetRequestedPageName, a member of class TAbstractWebContext.

Public Methods

	Name	Description
	FindApplicationModule (see page 24648)	Returns the application module instance.
	FindDefaultPageInfo (see page 24648)	Retrieves the Web page information object for a specified class of Web module.
	FindModuleClass (see page 24649)	Returns an instance of a Web module, given its class reference.
	FindModuleName (see page 24649)	Returns an instance of a Web module, given its class name.
	FindPage (see page 24650)	Retrieves a Web page module given its name.
	FindPageInfo (see page 24650)	Retrieves a Web page information object given the name of the corresponding Web page module.
	FindVariableContainer (see page 24651)	Returns an instance of a Web data module, given its class name.

176.1.3.4.1 TAbstractWebContext.FindApplicationModule Method

Returns the application module instance.

Pascal

```
function FindApplicationModule(WebModuleContext: TWebModuleContext): TComponent; virtual;
abstract;
```

C++

```
virtual __fastcall TComponent * FindApplicationModule(TWebModuleContext WebModuleContext) = 0;
```

Description

Call FindApplicationModule to access the main Web module in the WebSnap application. FindApplicationModule returns a reference to the module that the WebSnap application wizard creates. This can be an instance of TWebAppPageModule or TWebAppDataModule.

The Application module is the module that holds the application-wide components such as the page dispatcher, end user adapter, application adapter, and so on.

WebModuleContext is a context object that holds design-time information about the application modules. It is only used at design time.

See Also

WebModules (see page 24660)

TWebAppPageModule (see page 25394)

TWebAppDataModule (see page 25388)

176.1.3.4.2 TAbstractWebContext.FindDefaultPageInfo Method

Retrieves the Web page information object for a specified class of Web module.

Pascal

```
function FindDefaultPageInfo(AComponentClass: TComponentClass; out APageInfo:
TAbstractWebPageInfo): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean FindDefaultPageInfo(TComponentClass AComponentClass,
TAbstractWebPageInfo APageInfo) = 0;
```

Description

Call FindDefaultPageInfo to obtain the Web page information object for a Web module. FindDefaultPageInfo retrieves the information object that is registered for the Web page module class specified by AComponentClass. The Web page information object describes the characteristics of the Web page, including its name, title, HREF, view access string, an indication of whether users must log in to access the page, and so on.

AComponentClass is a class reference to the class that implements the Web page module. This class is declared in code that the Web page module wizard generates.

APageInfo returns the Web page information object that describes the characteristics of the specified Web page module, and that provides access to the factory object that you can use to create instances of the Web page module.

FindDefaultPageInfo returns false if it can't return a value for the APageInfo parameter. FindDefaultPageInfo returns true if it locates a registered Web page module for the class specified by AComponentClass.

See Also

FindPageInfo (see page 24650)

176.1.3.4.3 TAbstractWebContext.FindModuleClass Method

Returns an instance of a Web module, given its class reference.

Pascal

```
function FindModuleClass(AClass: TComponentClass): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * FindModuleClass(TComponentClass AClass) = 0;
```

Description

Call FindModuleClass to obtain a reference to a Web module given its class reference. If an instance of the specified class already exists, FindModuleClass returns that instance. Otherwise, if the specified class represents a registered Web module class, FindModuleClass creates an instance of the specified class and returns the new instance.

AClass is the class reference to a registered Web module class. This class is declared in code that the Web page module wizard or Web data module wizard generates.

See Also

FindModuleName (see page 24649)

FindPage (see page 24650)

176.1.3.4.4 TAbstractWebContext.FindModuleName Method

Returns an instance of a Web module, given its class name.

Pascal

```
function FindModuleName(const AClass: string): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * FindModuleName(const AnsiString AClass) = 0;
```

Description

Call FindModuleName to obtain a reference to a Web module given the name of its implementation class. If an instance of the specified class already exists, FindModuleName returns that instance. Otherwise, if the specified class represents a registered Web module class, FindModuleClass (see page 24649) creates an instance of the specified class and returns the new instance.

AClass is the name of a registered Web module class. This class is declared in code that the Web page module wizard or Web data module wizard generates.

See Also

FindModuleClass (see page 24649)

FindPage (see page 24650)

FindVariableContainer (see page 24651)

176.1.3.4.5 TAbstractWebContext.FindPage Method

Retrieves a Web page module given its name.

Pascal

```
function FindPage(const APage: string; AOptions: TFindPageOptions; out AComponent: TObject):
Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean FindPage(const AnsiString APage, TFindPageOptions AOptions, TObject
* AComponent) = 0;
```

Description

Call FindPage to obtain an instance of a registered Web page module given its name. If an instance of the specified Web page module already exists, FindPage returns that instance. Otherwise, if the named Web page module is registered, FindPage creates an instance of the module and returns the new instance.

APage is the name of the desired Web page module. The name is specified using the Web page module wizard, and forms the basis of the Web page module's class name. Thus, a Web page module with a name MyPage has the class name TMyPage.

AOptions is a set of options that can limit the page modules that FindPage considers. For any option included in this set, FindPage only considers Web page modules that have the corresponding attribute.

AComponent returns the instance of the specified Web page module.

FindPage returns true if it can find or create an instance of the specified Web page module, false otherwise.

See Also

FindModuleName (see page 24649)

FindPageInfo (see page 24650)

FindModuleClass (see page 24649)

176.1.3.4.6 TAbstractWebContext.FindPageInfo Method

Retrieves a Web page information object given the name of the corresponding Web page module.

Pascal

```
function FindPageInfo(const APage: string; AOptions: TFindPageOptions; out APageInfo:
TAbstractWebPageInfo): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean FindPageInfo(const AnsiString APage, TFindPageOptions AOptions,
TAbstractWebPageInfo APageInfo) = 0;
```

Description

Call FindPageInfo to obtain the Web page information object for a Web page module. FindPageInfo retrieves the information object that is registered for the Web page module that has a specified name. The Web page information object describes the characteristics of the Web page, including its name, title, HREF, view access string, an indication of whether users must log in to access the page, and so on.

APage is the name of the desired Web page module. The name is specified using the Web page module wizard, and forms the basis of the Web page module's class name. Thus, a Web page module with a name MyPage has the class name TMyPage.

AOptions is a set of options that can limit the page modules that FindPageInfo considers. For any option included in this set, FindPageInfo only considers Web page modules that have the corresponding attribute.

APageInfo returns the Web page information object that describes the characteristics of the specified Web page module, and that provides access to the factory object that you can use to create instances of the Web page module.

FindPageInfo returns false if it can't return a value for the APageInfo parameter. FindPageInfo returns true if it locates a registered Web page module with the specified name and attributes.

See Also

FindPage (see page 24650)

FindDefaultPageInfo (see page 24648)

Pages (see page 24658)

176.1.3.4.7 TAbstractWebContext.FindVariableContainer Method

Returns an instance of a Web data module, given its class name.

Pascal

```
function FindVariableContainer(const AIdent: string): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * FindVariableContainer(const AnsiString AIdent) = 0;
```

Description

Call FindVariableContainer to obtain a reference to a Web data module given the name of its implementation class. If an instance of the specified class already exists, FindVariableContainer returns that instance. Otherwise, if the specified class represents a registered Web module class, FindVariableContainer creates an instance of the specified class and returns the new instance.

AIdent is the name of a registered Web data module class. This class is declared in code that the Web data module wizard generates.

See Also

FindModuleName (see page 24649)

176.1.3.4.8 TAbstractWebContext.GetAdapterRequest Method

Pascal

```
function GetAdapterRequest: IUnknown; virtual; abstract;
```

C++

```
virtual __fastcall IUnknown GetAdapterRequest() = 0;
```

Description

This is GetAdapterRequest, a member of class TAbstractWebContext.

176.1.3.4.9 TAbstractWebContext.GetAdapterResponse Method

Pascal

```
function GetAdapterResponse: IUnknown; virtual; abstract;
```

C++

```
virtual __fastcall IUnknown GetAdapterResponse() = 0;
```

Description

This is GetAdapterResponse, a member of class TAbstractWebContext.

176.1.3.4.10 TAbstractWebContext.GetDispatchedPageName Method

Pascal

```
function GetDispatchedPageName: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetDispatchedPageName() = 0;
```

Description

This is GetDispatchedPageName, a member of class TAbstractWebContext.

176.1.3.4.11 TAbstractWebContext.GetEndUser Method

Pascal

```
function GetEndUser: IWebEndUser; virtual; abstract;
```

C++

```
virtual __fastcall IWebEndUser GetEndUser() = 0;
```

Description

This is GetEndUser, a member of class TAbstractWebContext.

176.1.3.4.12 TAbstractWebContext.GetPageCount Method

Pascal

```
function GetPageCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetPageCount() = 0;
```

See Also

GetPageCount

GetPageInfo (see page 24653)

PageCount (see page 24657)

176.1.3.4.13 TAbstractWebContext.GetPageInfo Method

Pascal

```
function GetPageInfo(I: Integer): IWebPageInfo; virtual; abstract;
```

C++

```
virtual __fastcall IWebPageInfo GetPageInfo(int I) = 0;
```

See Also

GetPageCount (see page 24652)

Pages (see page 24658)

176.1.3.4.14 TAbstractWebContext.GetRequestedPageName Method

Pascal

```
function GetRequestedPageName: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString GetRequestedPageName() = 0;
```

See Also

GetPageCount (see page 24652)

Pages (see page 24658)

176.1.3.4.15 TAbstractWebContext.GetSession Method

Pascal

```
function GetSession: TAbstractWebSession; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractWebSession GetSession() = 0;
```

See Also

GetPageCount (see page 24652)

Pages (see page 24658)

176.1.3.4.16 TAbstractWebContext.GetWebModules Method

Pascal

```
function GetWebModules: TAbstractWebModuleList; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractWebModuleList GetWebModules() = 0;
```

See Also

WebModules (see page 24660)

176.1.3.4.17 TAbstractWebContext.GetWebRequest Method

Pascal

```
function GetWebRequest: TWebRequest; virtual; abstract;
```

C++

```
virtual __fastcall TWebRequest GetWebRequest() = 0;
```

See Also

GetWebResponse (see page 24654)

Request (see page 24658)

176.1.3.4.18 TAbstractWebContext.GetWebResponse Method

Pascal

```
function GetWebResponse: TWebResponse; virtual; abstract;
```

C++

```
virtual __fastcall TWebResponse GetWebResponse() = 0;
```

See Also

GetWebRequest (see page 24654)

Response (see page 24659)

176.1.3.4.19 TAbstractWebContext.SetAdapterRequest Method

Pascal

```
procedure SetAdapterRequest(Intf: IUnknown); virtual; abstract;
```

C++

```
virtual __fastcall SetAdapterRequest(IUnknown Intf) = 0;
```

Description

This is SetAdapterRequest, a member of class TAbstractWebContext.

176.1.3.4.20 TAbstractWebContext.SetAdapterResponse Method

Pascal

```
procedure SetAdapterResponse(Intf: IUnknown); virtual; abstract;
```

C++

```
virtual __fastcall SetAdapterResponse(IUnknown Intf) = 0;
```

Description

This is SetAdapterResponse, a member of class TAbstractWebContext.

176.1.3.4.21 TAbstractWebContext.SetDispatchedPageName Method

Pascal

```
procedure SetDispatchedPageName(const APageName: string); virtual; abstract;
```

C++

```
virtual __fastcall SetDispatchedPageName(const AnsiString APageName) = 0;
```

Description

This is SetDispatchedPageName, a member of class TAbstractWebContext.

176.1.3.4.22 TAbstractWebContext.SetRequestedPageName Method

Pascal

```
procedure SetRequestedPageName(const APageName: string); virtual; abstract;
```

C++

```
virtual __fastcall SetRequestedPageName(const AnsiString APageName) = 0;
```

Description

This is SetRequestedPageName, a member of class TAbstractWebContext.

176.1.3.5 TAbstractWebContext Properties

The properties of the TAbstractWebContext class are listed here.

Public Properties

	Name	Description
R	AdapterRequest (see page 24655)	Provides access to the adapter request object that manages access to the current request.
R	AdapterResponse (see page 24656)	Provides access to the adapter response object that manages access to the current response.
	DispatchedPageName (see page 24656)	Specifies the name of the Web page module that is producing content to respond to the current request.
R	EndUser (see page 24657)	Provides access to information about the end user that sent the current request.
R	PageCount (see page 24657)	Indicates the number of registered Web pages available in the WebSnap application.
R	Pages (see page 24658)	Lists the page information interfaces of the registered Web pages available in the WebSnap application.
R	Request (see page 24658)	Represents the current HTTP request that the application is handling.
	RequestedPageName (see page 24658)	Indicates the name of the Web page module to which the current request is addressed.
R	Response (see page 24659)	Represents the current HTTP response that the application is assembling.
R	Session (see page 24659)	Provides access to the current Web session.
R	WebModules (see page 24660)	Lists all the Modules in the WebSnap application.

176.1.3.5.1 TAbstractWebContext.AdapterRequest Property

Provides access to the adapter request object that manages access to the current request.

Pascal

```
property AdapterRequest: IUnknown;
```

C++

```
__property IUnknown AdapterRequest;
```

Description

AdapterRequest is the interface of the object that adapters use to obtain information about the current HTTP request message. The adapter request object obtains its information from the TWebRequest object that is the value of the Request (see page 24658) property.

Note: In Delphi, this interface can typically be cast to an IActionRequest interface.

Note: In C++, you can generally obtain an IActionRequest interface from this interface.

See Also

AdapterResponse (see page 24656)

TAbstractAdapterRequest (see page 24639)

Request (see page 24658)

IActionRequest (see page 935)

176.1.3.5.2 TAbstractWebContext.AdapterResponse Property

Provides access to the adapter response object that manages access to the current response.

Pascal

```
property AdapterResponse: IUnknown;
```

C++

```
__property IUnknown AdapterResponse;
```

Description

AdapterResponse is the interface of the object that adapters use to obtain information about the current HTTP response message. The adapter response object obtains its information from the TWebResponse object that is the value of the Response (see page 24659) property.

Note: In Delphi, this interface can typically be cast to an IActionResponse interface.

Note: In C++, you can generally obtain an IActionResponse interface from this interface.

See Also

AdapterRequest (see page 24655)

TAbstractAdapterResponse (see page 24642)

Response (see page 24659)

IActionResponse (see page 941)

176.1.3.5.3 TAbstractWebContext.DispatchedPageName Property

Specifies the name of the Web page module that is producing content to respond to the current request.

Pascal

```
property DispatchedPageName: string;
```

C++

```
__property AnsiString DispatchedPageName;
```

Description

Read DispatchedPageName to determine the Web page module that is handling the current HTTP request message. Although this is typically the same as the page that the RequestedPageName (see page 24658) property specifies, it can differ if the request was redirected to another Web page or if its use was delayed because the caller had not yet logged in.

See Also

RequestedPageName (see page 24658)

FindPageInfo (see page 24650)

pagename

176.1.3.5.4 TAbstractWebContext.EndUser Property

Provides access to information about the end user that sent the current request.

Pascal

```
property EndUser: IWebEndUser;
```

C++

```
__property IWebEndUser EndUser;
```

Description

Use EndUser to obtain information about the caller that sent the current request. You can use the IWebEndUser (see page 24672) interface to determine whether the end user has logged in, whether the user has access rights to perform specified operations on pages that have a specified access string, the display name for the end user, and so on.

EndUser is nil (Delphi) or NULL (C++) if the WebSnap application does not include an end user adapter such as TEndUserAdapter or TEndUserSessionAdapter.

See Also

TEndUserAdapter (see page 24421)

TEndUserSessionAdapter (see page 24458)

176.1.3.5.5 TAbstractWebContext.PageCount Property

Indicates the number of registered Web pages available in the WebSnap application.

Pascal

```
property PageCount: Integer;
```

C++

```
__property int PageCount;
```

Description

PageCount indicates the number of entries in the Pages (see page 24658) property array. Use this value as an upper bound for the index when iterating over all the pages in the WebSnap application.

See Also

Pages (see page 24658)

176.1.3.5.6 TAbstractWebContext.Pages Property

Lists the page information interfaces of the registered Web pages available in the WebSnap application.

Pascal

```
property Pages [I: Integer]: IWebPageInfo;
```

C++

```
__property IWebPageInfo Pages[int I];
```

Description

Use Pages to obtain an IWebPageInfo (see page 24680) interface for a registered Web page module. The IWebPageInfo (see page 24680) interface describes the characteristics of a Web page, including its name, title, HREF, description, an indication of whether users must log in to access the page, an indication of whether the current user has access rights to view the page, and so on.

Pages provides indexed access to page information interfaces. To access the information about a Web page given its name, use the FindPageInfo (see page 24650) method instead. To access the information given the class of the Web page module, use the FindDefaultPageInfo (see page 24648) method instead. To access the page module itself (rather than the page information interface), use the WebModules (see page 24660) property instead.

See Also

PageCount (see page 24657)

FindPageInfo (see page 24650)

WebModules (see page 24660)

FindDefaultPageInfo (see page 24648)

176.1.3.5.7 TAbstractWebContext.Request Property

Represents the current HTTP request that the application is handling.

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

Read Request to obtain information about the current HTTP request message. The request object provides access to the headers of the request message, as well as the content of the message.

See Also

Response (see page 24659)

176.1.3.5.8 TAbstractWebContext.RequestedPageName Property

Indicates the name of the Web page module to which the current request is addressed.

Pascal

```
property RequestedPageName: string;
```

C++

```
__property AnsiString RequestedPageName;
```

Description

When the page dispatcher receives an HTTP request message, it sets RequestedPageName to the name of the target Web page module. This is taken from the pathinfo portion of the request message's target URL.

Note: There is no guarantee that RequestedPageName refers to a registered Web page module, because this property is set before the dispatcher verifies that the requested page exists.

See Also

DispatchedPageName (see page 24656)

FindPageInfo (see page 24650)

Request (see page 24658)

TPageDispatcher (see page 24949)

176.1.3.5.9 TAbstractWebContext.Response Property

Represents the current HTTP response that the application is assembling.

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

Read Response to obtain information about or add to the response message that the application builds as a reply to the message that the Request (see page 24658) property specifies. The response object provides access to the headers of the response message, as well as the content of the message.

See Also

Request (see page 24658)

176.1.3.5.10 TAbstractWebContext.Session Property

Provides access to the current Web session.

Pascal

```
property Session: TAbstractWebSession;
```

C++

```
__property TAbstractWebSession Session;
```

Description

Use Session to get or set information about the current session. Sessions store information about an end user that is needed for a short period of time. This can be used for such tasks as login/logout support, recording user preferences, and so on.

Note: If you want to use a session to store end user information, the WebSnap application should include a TSessionsService

component.

See Also

TSessionsService (see page 25582)

176.1.3.5.11 TAbstractWebContext.WebModules Property

Lists all the Modules in the WebSnap application.

Pascal

```
property WebModules: TAbstractWebModuleList;
```

C++

```
__property TAbstractWebModuleList WebModules;
```

Description

Use WebModules to access one of the Web modules that make up the WebSnap application. WebModules lists all of the currently instantiated Web modules. It also provides access to the factory objects for all Web modules in the application, including those that are not yet instantiated.

See Also

Pages (see page 24658)

FindModuleClass (see page 24649)

FindModuleName (see page 24649)

FindApplicationModule (see page 24648)

FindModuleClass (see page 24649)

176.1.4 TWebContext Class

TWebContext represents the current state of a WebSnap application.

Class Hierarchy

Pascal

```
TWebContext = class(TAbstractWebContext);
```

C++

```
class TWebContext : public TAbstractWebContext;
```

File

WebCntxt

Description

A Web context stores information about the components that are available in a WebSnap application and about the current request that is being processed. Applications do not create instances of the Web context. Rather, they access the current Web context object by calling the global WebContext (see page 24687) function.

TWebContext provides the implementation of the abstract or (in C++ terminology) pure virtual methods introduced by its

immediate ancestor, TAbstractWebContext (see page 24644).

See Also

WebContext (see page 24687)

Dispatching Requests and Responses

176.1.4.1 TWebContext Members

The following tables list the members exposed by TWebContext.

Protected Methods

	Name	Description
	GetAdapterRequest (see page 24667)	This is GetAdapterRequest, a member of class TWebContext.
	GetAdapterResponse (see page 24668)	This is GetAdapterResponse, a member of class TWebContext.
	GetDispatchedPageName (see page 24668)	This is GetDispatchedPageName, a member of class TWebContext.
	GetEndUser (see page 24668)	This is GetEndUser, a member of class TWebContext.
	GetPageCount (see page 24668)	
	GetPageInfo (see page 24669)	
	GetRequestedPageName (see page 24669)	
	GetSession (see page 24669)	
	GetWebModules (see page 24669)	
	GetWebRequest (see page 24670)	
	GetWebResponse (see page 24670)	
	SetAdapterRequest (see page 24670)	This is SetAdapterRequest, a member of class TWebContext.
	SetAdapterResponse (see page 24670)	This is SetAdapterResponse, a member of class TWebContext.
	SetDispatchedPageName (see page 24671)	This is SetDispatchedPageName, a member of class TWebContext.
	SetRequestedPageName (see page 24671)	This is SetRequestedPageName, a member of class TWebContext.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 24662)	Creates an initializes an instance of TWebContext (see page 24660).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 24663)	Frees an instance of TWebContext (see page 24660).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindApplicationModule (see page 24664)	Returns the application module instance.
	FindDefaultPageInfo (see page 24664)	Retrieves the Web page information object for a specified class of Web module.
	FindModuleClass (see page 24665)	Returns an instance of a Web module, given its class reference.
	FindModuleName (see page 24665)	Returns an instance of a Web module, given its class name.
	FindPage (see page 24666)	Retrieves a Web page module given its name.
	FindPageInfo (see page 24666)	Retrieves a Web page information object given the name of the corresponding Web page module.
	FindVariableContainer (see page 24667)	Returns an instance of a Web data module, given its class name.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	AdapterRequest (see page 24655)	Provides access to the adapter request object that manages access to the current request.
R	AdapterResponse (see page 24656)	Provides access to the adapter response object that manages access to the current response.
	DispatchedPageName (see page 24656)	Specifies the name of the Web page module that is producing content to respond to the current request.
R	EndUser (see page 24657)	Provides access to information about the end user that sent the current request.
R	PageCount (see page 24657)	Indicates the number of registered Web pages available in the WebSnap application.
R	Pages (see page 24658)	Lists the page information interfaces of the registered Web pages available in the WebSnap application.
R	Request (see page 24658)	Represents the current HTTP request that the application is handling.
	RequestedPageName (see page 24658)	Indicates the name of the Web page module to which the current request is addressed.
R	Response (see page 24659)	Represents the current HTTP response that the application is assembling.
R	Session (see page 24659)	Provides access to the current Web session.
R	WebModules (see page 24660)	Lists all the Modules in the WebSnap application.

176.1.4.2 TWebContext.Create Constructor

Creates an initializes an instance of TWebContext (see page 24660).

Pascal

```
constructor Create(AWebModuleList: TAbstractWebModuleList; ARequest: TWebRequest; AResponse:
TWebResponse; ASession: TAbstractWebSession);
```

C++

```
__fastcall TWebContext(TAbstractWebModuleList AWebModuleList, TWebRequest ARequest,
TWebResponse AResponse, TAbstractWebSession ASession);
```

Description

Do not create instances of TWebContext (see page 24660). WebSnap applications create a global Web context automatically to represent the current state of the application. To access this global Web context object, use the global WebContext (see page 24687) function.

The TWebContext (see page 24660) constructor registers itself with the global WebContext (see page 24687) function so that the application can access the Web context using that function, and initializes the properties of the Web context to the values passed in as parameters.

AWebModuleList is the component that manages the Web modules in the application. It becomes the value of the WebModules (see page 24660) property.

ARequest is the current HTTP request message. It becomes the value of the Request (see page 24658) property.

AResponse is the current response object that the application is filling out. It becomes the value of the Response (see page 24659) property.

ASession is the Session (see page 24659) object associated with ARequest. It becomes the value of the Session (see page 24659) property.

See Also

WebContext (see page 24687)

WebModules (see page 24660)

Request (see page 24658)

Response (see page 24659)

Session (see page 24659)

176.1.4.3 TWebContext.Destroy Destructor

Frees an instance of TWebContext (see page 24660).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebContext();
```

Description

Do not free the global Web context object. This object is automatically maintained by the WebSnap application.

The TWebContext (see page 24660) destructor unregisters itself with the global WebContext (see page 24687) function so that the WebContext (see page 24687) function never returns a Web context that no longer exists.

See Also

WebContext (see page 24687)

176.1.4.4 TWebContext Methods

The methods of the TWebContext class are listed here.

Protected Methods

	Name	Description
	GetAdapterRequest (see page 24667)	This is GetAdapterRequest, a member of class TWebContext.
	GetAdapterResponse (see page 24668)	This is GetAdapterResponse, a member of class TWebContext.
	GetDispatchedPageName (see page 24668)	This is GetDispatchedPageName, a member of class TWebContext.
	GetEndUser (see page 24668)	This is GetEndUser, a member of class TWebContext.
	GetPageCount (see page 24668)	
	GetPageInfo (see page 24669)	
	GetRequestedPageName (see page 24669)	
	GetSession (see page 24669)	

	GetWebModules (see page 24669)	
	GetWebRequest (see page 24670)	
	GetWebResponse (see page 24670)	
	SetAdapterRequest (see page 24670)	This is SetAdapterRequest, a member of class TWebContext.
	SetAdapterResponse (see page 24670)	This is SetAdapterResponse, a member of class TWebContext.
	SetDispatchedPageName (see page 24671)	This is SetDispatchedPageName, a member of class TWebContext.
	SetRequestedPageName (see page 24671)	This is SetRequestedPageName, a member of class TWebContext.

Public Methods

	Name	Description
	FindApplicationModule (see page 24664)	Returns the application module instance.
	FindDefaultPageInfo (see page 24664)	Retrieves the Web page information object for a specified class of Web module.
	FindModuleClass (see page 24665)	Returns an instance of a Web module, given its class reference.
	FindModuleName (see page 24665)	Returns an instance of a Web module, given its class name.
	FindPage (see page 24666)	Retrieves a Web page module given its name.
	FindPageInfo (see page 24666)	Retrieves a Web page information object given the name of the corresponding Web page module.
	FindVariableContainer (see page 24667)	Returns an instance of a Web data module, given its class name.

176.1.4.4.1 TWebContext.FindApplicationModule Method

Returns the application module instance.

Pascal

```
function FindApplicationModule(WebModuleContext: TWebModuleContext): TComponent; override;
```

C++

```
virtual __fastcall TComponent * FindApplicationModule(TWebModuleContext WebModuleContext);
```

Description

Call FindApplicationModule to access the main Web module in the WebSnap application. FindApplicationModule returns a reference to the module that the WebSnap application wizard creates. This can be an instance of TWebAppPageModule or TWebAppDataModule.

The Application module is the module that holds the application-wide components such as the page dispatcher, end user adapter, application adapter, and so on.

WebModuleContext is a context object that holds design-time information about the application modules. It is only used at design time.

See Also

WebModules (see page 24660)

TWebAppPageModule (see page 25394)

TWebAppDataModule (see page 25388)

176.1.4.4.2 TWebContext.FindDefaultPageInfo Method

Retrieves the Web page information object for a specified class of Web module.

Pascal

```
function FindDefaultPageInfo(AComponentClass: TComponentClass; out APageInfo:
TAbstractWebPageInfo): Boolean; override;
```

C++

```
virtual __fastcall Boolean FindDefaultPageInfo(TComponentClass AComponentClass,
TAbstractWebPageInfo APageInfo);
```

Description

Call FindDefaultPageInfo to obtain the Web page information object for a Web module. FindDefaultPageInfo retrieves the information object that is registered for the Web page module class specified by AComponentClass. The Web page information object describes the characteristics of the Web page, including its name, title, HREF, view access string, an indication of whether users must log in to access the page, and so on.

AComponentClass is a class reference to the class that implements the Web page module. This class is declared in code that the Web page module wizard generates.

APageInfo returns the Web page information object that describes the characteristics of the specified Web page module, and that provides access to the factory object that you can use to create instances of the Web page module.

FindDefaultPageInfo returns false if it can't return a value for the APageInfo parameter. FindDefaultPageInfo returns true if it locates a registered Web page module for the class specified by AComponentClass.

See Also

FindPageInfo (see page 24666)

176.1.4.4.3 TWebContext.FindModuleClass Method

Returns an instance of a Web module, given its class reference.

Pascal

```
function FindModuleClass(AClass: TComponentClass): TComponent; override;
```

C++

```
virtual __fastcall TComponent * FindModuleClass(TComponentClass AClass);
```

Description

Call FindModuleClass to obtain a reference to a Web module given its class reference. If an instance of the specified class already exists, FindModuleClass returns that instance. Otherwise, if the specified class represents a registered Web module class, FindModuleClass creates an instance of the specified class and returns the new instance.

AClass is the class reference to a registered Web module class. This class is declared in code that the Web page module wizard or Web data module wizard generates.

See Also

FindModuleName (see page 24665)

FindPage (see page 24666)

176.1.4.4.4 TWebContext.FindModuleName Method

Returns an instance of a Web module, given its class name.

Pascal

```
function FindModuleName(const AName: string): TComponent; override;
```

C++

```
virtual __fastcall TComponent * FindModuleName(const AnsiString AName);
```

Description

Call FindModuleName to obtain a reference to a Web module given the name of its implementation class. If an instance of the specified class already exists, FindModuleName returns that instance. Otherwise, if the specified class represents a registered Web module class, FindModuleClass (see page 24665) creates an instance of the specified class and returns the new instance.

AClass is the name of a registered Web module class. This class is declared in code that the Web page module wizard or Web data module wizard generates.

See Also

FindModuleClass (see page 24665)

FindPage (see page 24666)

FindVariableContainer (see page 24667)

176.1.4.4.5 TWebContext.FindPage Method

Retrieves a Web page module given its name.

Pascal

```
function FindPage(const APage: string; AOptions: TFindPageOptions; out AComponent: TObject):
Boolean; override;
```

C++

```
virtual __fastcall Boolean FindPage(const AnsiString APage, TFindPageOptions AOptions, TObject
* AComponent);
```

Description

Call FindPage to obtain an instance of a registered Web page module given its name. If an instance of the specified Web page module already exists, FindPage returns that instance. Otherwise, if the named Web page module is registered, FindPage creates an instance of the module and returns the new instance.

APage is the name of the desired Web page module. The name is specified using the Web page module wizard, and forms the basis of the Web page module's class name. Thus, a Web page module with a name MyPage has the class name TMyPage.

AOptions is a set of options that can limit the page modules that FindPage considers. For any option included in this set, FindPage only considers Web page modules that have the corresponding attribute.

AComponent returns the instance of the specified Web page module.

FindPage returns true if it can find or create an instance of the specified Web page module, false otherwise.

See Also

FindModuleName (see page 24665)

FindPageInfo (see page 24666)

FindModuleClass (see page 24665)

176.1.4.4.6 TWebContext.FindPageInfo Method

Retrieves a Web page information object given the name of the corresponding Web page module.

Pascal

```
function FindPageInfo(const APage: string; AOptions: TFindPageOptions; out APageInfo:
TAbstractWebPageInfo): Boolean; override;
```

C++

```
virtual __fastcall Boolean FindPageInfo(const AnsiString APage, TFindPageOptions AOptions,
TAbstractWebPageInfo APageInfo);
```

Description

Call FindPageInfo to obtain the Web page information object for a Web page module. FindPageInfo retrieves the information object that is registered for the Web page module that has a specified name. The Web page information object describes the characteristics of the Web page, including its name, title, HREF, view access string, an indication of whether users must log in to access the page, and so on.

APage is the name of the desired Web page module. The name is specified using the Web page module wizard, and forms the basis of the Web page module's class name. Thus, a Web page module with a name MyPage has the class name TMyPage.

AOptions is a set of options that can limit the page modules that FindPageInfo considers. For any option included in this set, FindPageInfo only considers Web page modules that have the corresponding attribute.

APageInfo returns the Web page information object that describes the characteristics of the specified Web page module, and that provides access to the factory object that you can use to create instances of the Web page module.

FindPageInfo returns false if it can't return a value for the APageInfo parameter. FindPageInfo returns true if it locates a registered Web page module with the specified name and attributes.

See Also

FindPage (see page 24666)

FindDefaultPageInfo (see page 24664)

Pages (see page 24658)

176.1.4.4.7 TWebContext.FindVariableContainer Method

Returns an instance of a Web data module, given its class name.

Pascal

```
function FindVariableContainer(const AIdent: string): TComponent; override;
```

C++

```
virtual __fastcall TComponent * FindVariableContainer(const AnsiString AIdent);
```

Description

Call FindVariableContainer to obtain a reference to a Web data module given the name of its implementation class. If an instance of the specified class already exists, FindVariableContainer returns that instance. Otherwise, if the specified class represents a registered Web module class, FindVariableContainer creates an instance of the specified class and returns the new instance.

AIdent is the name of a registered Web data module class. This class is declared in code that the Web data module wizard generates.

See Also

FindModuleName (see page 24665)

176.1.4.4.8 TWebContext.GetAdapterRequest Method

Pascal

```
function GetAdapterRequest: IUnknown; override;
```

C++

```
virtual __fastcall IUnknown GetAdapterRequest();
```

Description

This is GetAdapterRequest, a member of class TWebContext.

176.1.4.4.9 TWebContext.GetAdapterResponse Method

Pascal

```
function GetAdapterResponse: IUnknown; override;
```

C++

```
virtual __fastcall IUnknown GetAdapterResponse();
```

Description

This is GetAdapterResponse, a member of class TWebContext.

176.1.4.4.10 TWebContext.GetDispatchedPageName Method

Pascal

```
function GetDispatchedPageName: string; override;
```

C++

```
virtual __fastcall AnsiString GetDispatchedPageName();
```

Description

This is GetDispatchedPageName, a member of class TWebContext.

176.1.4.4.11 TWebContext.GetEndUser Method

Pascal

```
function GetEndUser: IWebEndUser; override;
```

C++

```
virtual __fastcall IWebEndUser GetEndUser();
```

Description

This is GetEndUser, a member of class TWebContext.

176.1.4.4.12 TWebContext.GetPageCount Method

Pascal

```
function GetPageCount: Integer; override;
```

C++

```
virtual __fastcall int GetPageCount();
```

See Also

GetPageCount (see page 24652)

GetPageInfo (see page 24653)

PageCount (see page 24657)

176.1.4.4.13 TWebContext.GetPageInfo Method

Pascal

```
function GetPageInfo(I: Integer): IWebPageInfo; override;
```

C++

```
virtual __fastcall IWebPageInfo GetPageInfo(int I);
```

See Also

GetPageCount (see page 24652)

Pages (see page 24658)

176.1.4.4.14 TWebContext.GetRequestedPageName Method

Pascal

```
function GetRequestedPageName: string; override;
```

C++

```
virtual __fastcall AnsiString GetRequestedPageName();
```

See Also

GetPageCount (see page 24652)

Pages (see page 24658)

176.1.4.4.15 TWebContext.GetSession Method

Pascal

```
function GetSession: TAbstractWebSession; override;
```

C++

```
virtual __fastcall TAbstractWebSession GetSession();
```

See Also

GetPageCount (see page 24652)

Pages (see page 24658)

176.1.4.4.16 TWebContext.GetWebModules Method

Pascal

```
function GetWebModules: TAbstractWebModuleList; override;
```

C++

```
virtual __fastcall TAbstractWebModuleList GetWebModules();
```

See Also

WebModules (see page 24660)

176.1.4.4.17 TWebContext.GetWebRequest Method

Pascal

```
function GetWebRequest: TWebRequest; override;
```

C++

```
virtual __fastcall TWebRequest GetWebRequest();
```

See Also

GetWebResponse (see page 24654)

Request (see page 24658)

176.1.4.4.18 TWebContext.GetWebResponse Method

Pascal

```
function GetWebResponse: TWebResponse; override;
```

C++

```
virtual __fastcall TWebResponse GetWebResponse();
```

See Also

GetWebRequest (see page 24654)

Response (see page 24659)

176.1.4.4.19 TWebContext.SetAdapterRequest Method

Pascal

```
procedure SetAdapterRequest(Intf: IUnknown); override;
```

C++

```
virtual __fastcall SetAdapterRequest(IUnknown Intf);
```

Description

This is SetAdapterRequest, a member of class TWebContext.

176.1.4.4.20 TWebContext.SetAdapterResponse Method

Pascal

```
procedure SetAdapterResponse(Intf: IUnknown); override;
```

C++

```
virtual __fastcall SetAdapterResponse(IUnknown Intf);
```

Description

This is SetAdapterResponse, a member of class TWebContext.

176.1.4.4.21 TWebContext.SetDispatchedPageName Method

Pascal

```
procedure SetDispatchedPageName(const APageName: string); override;
```

C++

```
virtual __fastcall SetDispatchedPageName(const AnsiString APageName);
```

Description

This is SetDispatchedPageName, a member of class TWebContext.

176.1.4.4.22 TWebContext.SetRequestedPageName Method

Pascal

```
procedure SetRequestedPageName(const APageName: string); override;
```

C++

```
virtual __fastcall SetRequestedPageName(const AnsiString APageName);
```

Description

This is SetRequestedPageName, a member of class TWebContext.

176.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IWebEndUser (see page 24672)	IWebEndUser provides access information about the current end user in a WebSnap application.
	IWebPageInfo (see page 24680)	IWebPageInfo describes a Web page in a WebSnap application.

176.2.1 IWebEndUser Interface

IWebEndUser provides access information about the current end user in a WebSnap application.

Class Hierarchy

WebCntxt.IWebEndUser

Pascal

```
IWebEndUser = interface;
```

C++

```
__interface IWebEndUser;
```

File

WebCntxt

Description

Use IWebEndUser to obtain information about the current end user, such as the user name, user ID, and whether the user is logged in. In addition to properties that provide information about the current end user, IWebEndUser has methods to log in an end user.

End user adapter components such as TEndUserAdapter and TEndUserSessionAdapter implement the IWebEndUser interface. If a WebSnap application includes an end user adapter, you can access its IWebEndUser interface by reading the EndUser property of the Web context.

Note: In C++, method declarations that use IWebEndUser use the _di_IWebEndUser type instead. This type is a DelphiInterface wrapper around the IWebEndUser interface:

Note: typedef System::DelphiInterface (see page 22167)< IWebEndUser > _di_IWebEndUser;

See Also

TAbstractWebContext (see page 24644)

TEndUserAdapter (see page 24421)

TEndUserSessionAdapter (see page 24458)

176.2.1.1 IWebEndUser Members

The following tables list the members exposed by IWebEndUser.

Public Methods

	Name	Description
	GetDisplayName (see page 24673)	Returns the value of the DisplayName (see page 24678) property.
	GetEndUserAdapter (see page 24674)	Returns the value of the EndUserAdapter (see page 24678) property.
	GetLoggedIn (see page 24674)	Returns the value of the LoggedIn (see page 24678) property.
	GetLoginFormAction (see page 24674)	Returns the value of the LoginFormAction (see page 24679) property.
	GetLogoutAction (see page 24675)	Returns the value of the LogoutAction (see page 24680) property.
	HasExecuteRights (see page 24675)	Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.
	HasModifyRights (see page 24676)	Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.
	HasViewRights (see page 24676)	Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.
	LogIn (see page 24677)	Logs in a specified user.
	ProduceLoginPage (see page 24677)	Generates the Web page for logging in users.

Public Properties

	Name	Description
R	DisplayName (see page 24678)	Indicates the name of the end user as it should be used for display in Web pages.
R	EndUserAdapter (see page 24678)	Provides access to the end user adapter that implements this interface.
R	LoggedIn (see page 24678)	Indicates whether the current user is logged in.
R	LoginFormAction (see page 24679)	Indicates the action component for the login form.
R	LogoutAction (see page 24680)	Indicates the action component for logging a user out.

176.2.1.2 IWebEndUser Methods

The methods of the IWebEndUser class are listed here.

Public Methods

	Name	Description
	GetDisplayName (see page 24673)	Returns the value of the DisplayName (see page 24678) property.
	GetEndUserAdapter (see page 24674)	Returns the value of the EndUserAdapter (see page 24678) property.
	GetLoggedIn (see page 24674)	Returns the value of the LoggedIn (see page 24678) property.
	GetLoginFormAction (see page 24674)	Returns the value of the LoginFormAction (see page 24679) property.
	GetLogoutAction (see page 24675)	Returns the value of the LogoutAction (see page 24680) property.
	HasExecuteRights (see page 24675)	Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.
	HasModifyRights (see page 24676)	Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.
	HasViewRights (see page 24676)	Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.
	LogIn (see page 24677)	Logs in a specified user.
	ProduceLoginPage (see page 24677)	Generates the Web page for logging in users.

176.2.1.2.1 IWebEndUser.GetDisplayName Method

Returns the value of the DisplayName (see page 24678) property.

Pascal

```
function GetDisplayName: string;
```

C++

```
__fastcall AnsiString GetDisplayName();
```

Description

GetDisplayName is the read implementation of the DisplayName (see page 24678) property.

See Also

DisplayName (see page 24678)

176.2.1.2.2 IWebEndUser.GetEndUserAdapter Method

Returns the value of the EndUserAdapter (see page 24678) property.

Pascal

```
function GetEndUserAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetEndUserAdapter();
```

Description

GetEndUserAdapter is the read implementation of the EndUserAdapter (see page 24678) property.

See Also

EndUserAdapter (see page 24678)

176.2.1.2.3 IWebEndUser.GetLoggedIn Method

Returns the value of the LoggedIn (see page 24678) property.

Pascal

```
function GetLoggedIn: Boolean;
```

C++

```
__fastcall Boolean GetLoggedIn();
```

Description

GetLoggedIn is the read implementation of the LoggedIn (see page 24678) property.

See Also

LoggedIn (see page 24678)

176.2.1.2.4 IWebEndUser.GetLoginFormAction Method

Returns the value of the LoginFormAction (see page 24679) property.

Pascal

```
function GetLoginFormAction: TComponent;
```

C++

```
__fastcall TComponent * GetLoginFormAction();
```

Description

GetLoginFormAction is the read implementation of the LoginFormAction (see page 24679) property.

See Also

LoginFormAction (see page 24679)

176.2.1.2.5 IWebEndUser.GetLogoutAction Method

Returns the value of the LogoutAction (see page 24680) property.

Pascal

```
function GetLogoutAction: TComponent;
```

C++

```
__fastcall TComponent * GetLogoutAction();
```

Description

GetLogoutAction is the read implementation of the LogoutAction (see page 24680) property.

See Also

LogoutAction (see page 24680)

176.2.1.2.6 IWebEndUser.HasExecuteRights Method

Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.

Pascal

```
function HasExecuteRights(ARights: string): Boolean;
```

C++

```
__fastcall Boolean HasExecuteRights(AnsiString ARights);
```

Description

Call HasExecuteRights to determine whether the current end user is authorized to execute Web pages or adapters whose ViewRights property is the value specified by ARights.

Typically, the end user adapter that implements this interface implements HasExecuteRights by checking the AccessRightsStrings property of the entry for this end user in a TWebUserList component. Note that in this case, the end user must be logged in before HasExecuteRights can determine the user's access rights.

See Also

LoggedIn (see page 24678)

HasViewRights (see page 24676)

HasModifyRights (see page 24676)

ViewAccess (see page 15069)

ViewAccess (see page 24183)

TWebUserList (see page 25625)

AccessRightsStrings (see page 25619)

176.2.1.2.7 IWebEndUser.HasModifyRights Method

Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.

Pascal

```
function HasModifyRights(ARights: string): Boolean;
```

C++

```
__fastcall Boolean HasModifyRights(AnsiString ARights);
```

Description

Call HasModifyRights to determine whether the current end user is authorized to modify Web pages or adapters whose ViewRights property is the value specified by ARights.

Typically, the end user adapter that implements this interface implements HasModifyRights by checking the AccessRightsStrings property of the entry for this end user in a TWebUserList component. Note that in this case, the end user must be logged in before HasModifyRights can determine the user's access rights.

See Also

LoggedIn (see page 24678)

HasViewRights (see page 24676)

HasExecuteRights (see page 24675)

ViewAccess (see page 15069)

ViewAccess (see page 24183)

TWebUserList (see page 25625)

AccessRightsStrings (see page 25619)

176.2.1.2.8 IWebEndUser.HasViewRights Method

Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.

Pascal

```
function HasViewRights(ARights: string): Boolean;
```

C++

```
__fastcall Boolean HasViewRights(AnsiString ARights);
```

Description

Call HasViewRights to determine whether the current end user is authorized to view Web pages or adapters whose ViewRights property is the value specified by ARights.

Typically, the end user adapter that implements this interface implements HasViewRights by checking the AccessRightsStrings property of the entry for this end user in a TWebUserList component. Note that in this case, the end user must be logged in before HasViewRights can determine the user's access rights.

See Also

LoggedIn (see page 24678)

HasExecuteRights (see page 24675)

HasModifyRights (see page 24676)

ViewAccess (see page 15069)

ViewAccess (see page 24183)

TWebUserList (see page 25625)

AccessRightsStrings (see page 25619)

176.2.1.2.9 IWebEndUser.LogIn Method

Logs in a specified user.

Pascal

```
procedure LogIn(UserID: Variant);
```

C++

```
__fastcall LogIn(Variant UserID);
```

Description

Call LogIn to record a user as logged in. Typically, LogIn is called in response to a message from the login page that ProduceLoginPage (see page 24677) generates.

UserID is the identifier of the user that logged in. It is the value of the UserID property of the entry for the user in a WebUserList component, or, in applications that implement their own login scheme, any other Variant that identifies an end user.

See Also

LoggedIn (see page 24678)

ProduceLoginPage (see page 24677)

TWebUserList (see page 25625)

UserID (see page 25619)

TLoginFormAdapter (see page 24500)

176.2.1.2.10 IWebEndUser.ProduceLoginPage Method

Generates the Web page for logging in users.

Pascal

```
function ProduceLoginPage: Boolean;
```

C++

```
__fastcall Boolean ProduceLoginPage();
```

Description

Call ProduceLoginPage to generate a Web page for logging in users.

ProduceLoginPage returns true if it generates a login form page and updates the current response message so that it will be displayed to the user. ProduceLoginPage returns false if the end user adapter is unable to generate a login page.

See Also

LoggedIn (see page 24678)

LogIn (see page 24677)

LoginFormAction (see page 24679)

176.2.1.3 IWebEndUser Properties

The properties of the IWebEndUser class are listed here.

Public Properties

	Name	Description
R	DisplayName (see page 24678)	Indicates the name of the end user as it should be used for display in Web pages.
R	EndUserAdapter (see page 24678)	Provides access to the end user adapter that implements this interface.
R	LoggedIn (see page 24678)	Indicates whether the current user is logged in.
R	LoginFormAction (see page 24679)	Indicates the action component for the login form.
R	LogoutAction (see page 24680)	Indicates the action component for logging a user out.

176.2.1.3.1 IWebEndUser.DisplayName Property

Indicates the name of the end user as it should be used for display in Web pages.

Pascal

```
property DisplayName: string;
```

C++

```
__property AnsiString DisplayName;
```

Description

Read DisplayName to obtain a string that can be added to Web pages to refer to the current user.

See Also

UserName (see page 25620)

176.2.1.3.2 IWebEndUser.EndUserAdapter Property

Provides access to the end user adapter that implements this interface.

Pascal

```
property EndUserAdapter: TComponent;
```

C++

```
__property TComponent * EndUserAdapter;
```

Description

Read EndUserAdapter to obtain access to the end user adapter. This is the component that implements this interface. Typically, it is a TEndUserAdapter or TEndUserSessionAdapter instance.

See Also

TEndUserAdapter (see page 24421)

TEndUserSessionAdapter (see page 24458)

176.2.1.3.3 IWebEndUser.LoggedIn Property

Indicates whether the current user is logged in.

Pascal

```
property LoggedIn: Boolean;
```

C++

```
__property Boolean LoggedIn;
```

Description

Read LoggedIn to determine whether the current user has logged in to the WebSnap application. By logging in, a caller can be identified as one of the users that a TWebUserList component lists.

Once the user has logged in, the HasViewRights (see page 24676), HasExecuteRights (see page 24675), and HasModifyRights (see page 24676) methods can be used to determine whether the user is authorized to view, execute actions, and modify data on restricted Web pages.

To log in a user that is not logged in, call the ProduceLoginPage (see page 24677) method to display a login Web page. That page should allow the user to provide some sort of UserID, which can then be used to log the user in by a call to the LogIn (see page 24677) method.

See Also

LogIn (see page 24677)

ProduceLoginPage (see page 24677)

TWebUserList (see page 25625)

HasExecuteRights (see page 24675)

HasModifyRights (see page 24676)

HasViewRights (see page 24676)

176.2.1.3.4 IWebEndUser.LoginFormAction Property

Indicates the action component for the login form.

Pascal

```
property LoginFormAction: TComponent;
```

C++

```
__property TComponent * LoginFormAction;
```

Description

Read LoginFormAction to access the login form action component. The login form action component can be called from server-side scripts to display a login form Web page.

Note: To display the login form Web page using the IWebEndUser (see page 24672) interface (rather than from a server-side script), call the ProduceLoginPage (see page 24677) method.

See Also

ProduceLoginPage (see page 24677)

TEndUserAdapterLoginFormAction (see page 24436)

LogoutAction (see page 24680)

176.2.1.3.5 IWebEndUser.LogoutAction Property

Indicates the action component for logging a user out.

Pascal

```
property LogoutAction: TComponent;
```

C++

```
__property TComponent * LogoutAction;
```

Description

Read LogoutAction to access the logout action component. The logout action component can be called from server-side scripts to perform any necessary tasks when the user logs out.

See Also

LoginFormAction (see page 24679)

TEndUserAdapterLogoutAction (see page 24447)

176.2.2 IWebPageInfo Interface

IWebPageInfo describes a Web page in a WebSnap application.

Class Hierarchy

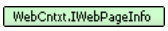

Pascal

```
IWebPageInfo = interface;
```

C++

```
__interface IWebPageInfo;
```

File

WebCntxt

Description

Use IWebPageInfo to obtain information about a Web page, including its page name, page title, descriptive text, publish status, and access rights. This information is taken from a TWebPageInfo object that is stored with the Web page module factory.

Note: VCL Method declarations that use IWebPageInfo use the _di_IWebPageInfo type instead. This type is a DelphiInterface wrapper around the IWebPageInfo interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IWebPageInfo >  _di_IWebPageInfo;
```

See Also

TWebPageInfo (see page 25016)

TAbstractWebContext (see page 24644)

176.2.2.1 IWebPageInfo Members

The following tables list the members exposed by IWebPageInfo.

Public Methods

	Name	Description
	GetHasViewAccess (see page 24681)	Returns the value of the HasViewAccess (see page 24684) property.
	GetLoginRequired (see page 24682)	Returns the value of the LoginRequired (see page 24684) property.
	GetPageDescription (see page 24682)	Returns the value of the PageDescription (see page 24684) property.
	GetPageHREF (see page 24682)	Returns the value of the PageHREF (see page 24685) property.
	GetPageIsPublished (see page 24683)	Returns the value of the PageIsPublished (see page 24685) property.
	GetPageName (see page 24683)	Returns the value of the PageName (see page 24686) property.
	GetPageTitle (see page 24683)	Returns the value of the PageTitle (see page 24686) property.

Public Properties

	Name	Description
R	HasViewAccess (see page 24684)	Indicates whether the current user has access rights to view the Web page.
R	LoginRequired (see page 24684)	Indicates whether callers must log in before accessing this Web page.
R	PageDescription (see page 24684)	Describes the Web page.
R	PageHREF (see page 24685)	Identifies the HREF to use for references to the associated Web page.
R	PageIsPublished (see page 24685)	Indicates whether the WebSnap application automatically displays the Web page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 24686).
R	PageName (see page 24686)	Denotes the logical name for the Web page.
R	PageTitle (see page 24686)	Indicates the title that appears on the Web page.

176.2.2.2 IWebPageInfo Methods

The methods of the IWebPageInfo class are listed here.

Public Methods

	Name	Description
	GetHasViewAccess (see page 24681)	Returns the value of the HasViewAccess (see page 24684) property.
	GetLoginRequired (see page 24682)	Returns the value of the LoginRequired (see page 24684) property.
	GetPageDescription (see page 24682)	Returns the value of the PageDescription (see page 24684) property.
	GetPageHREF (see page 24682)	Returns the value of the PageHREF (see page 24685) property.
	GetPageIsPublished (see page 24683)	Returns the value of the PageIsPublished (see page 24685) property.
	GetPageName (see page 24683)	Returns the value of the PageName (see page 24686) property.
	GetPageTitle (see page 24683)	Returns the value of the PageTitle (see page 24686) property.

176.2.2.2.1 IWebPageInfo.GetHasViewAccess Method

Returns the value of the HasViewAccess (see page 24684) property.

Pascal

```
function GetHasViewAccess: Boolean;
```

C++

```
__fastcall Boolean GetHasViewAccess();
```

Description

GetHasViewAccess is the read implementation of the HasViewAccess (see page 24684) property.

See Also

HasViewAccess (see page 24684)

176.2.2.2.2 IWebPageInfo.GetLoginRequired Method

Returns the value of the LoginRequired (see page 24684) property.

Pascal

```
function GetLoginRequired: Boolean;
```

C++

```
__fastcall Boolean GetLoginRequired();
```

Description

GetLoginRequired is the read implementation of the LoginRequired (see page 24684) property.

See Also

LoginRequired (see page 24684)

176.2.2.2.3 IWebPageInfo.GetPageDescription Method

Returns the value of the PageDescription (see page 24684) property.

Pascal

```
function GetPageDescription: string;
```

C++

```
__fastcall AnsiString GetPageDescription();
```

Description

GetPageDescription is the read implementation of the PageDescription (see page 24684) property.

See Also

PageDescription (see page 24684)

176.2.2.2.4 IWebPageInfo.GetPageHREF Method

Returns the value of the PageHREF (see page 24685) property.

Pascal

```
function GetPageHREF: string;
```

C++

```
__fastcall AnsiString GetPageHREF();
```

Description

GetPageHREF is the read implementation of the PageHREF (see page 24685) property.

See Also

PageHREF (see page 24685)

176.2.2.2.5 IWebPageInfo.GetPageIsPublished Method

Returns the value of the PageIsPublished (see page 24685) property.

Pascal

```
function GetPageIsPublished: Boolean;
```

C++

```
__fastcall Boolean GetPageIsPublished();
```

Description

GetPageIsPublished is the read implementation of the PageIsPublished (see page 24685) property.

See Also

PageIsPublished (see page 24685)

176.2.2.2.6 IWebPageInfo.GetPageName Method

Returns the value of the PageName (see page 24686) property.

Pascal

```
function GetPageName: string;
```

C++

```
__fastcall AnsiString GetPageName();
```

Description

GetPageName is the read implementation of the PageName (see page 24686) property.

See Also

PageName (see page 24686)

176.2.2.2.7 IWebPageInfo.GetPageTitle Method

Returns the value of the PageTitle (see page 24686) property.

Pascal

```
function GetPageTitle: string;
```

C++

```
__fastcall AnsiString GetPageTitle();
```

Description

GetPageTitle is the read implementation of the PageTitle (see page 24686) property.

See Also

PageTitle (see page 24686)

176.2.2.3 IWebPageInfo Properties

The properties of the IWebPageInfo class are listed here.

Public Properties

	Name	Description
R	HasViewAccess (see page 24684)	Indicates whether the current user has access rights to view the Web page.
R	LoginRequired (see page 24684)	Indicates whether callers must log in before accessing this Web page.
R	PageDescription (see page 24684)	Describes the Web page.
R	PageHREF (see page 24685)	Identifies the HREF to use for references to the associated Web page.
R	PageIsPublished (see page 24685)	Indicates whether the WebSnap application automatically displays the Web page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 24686).
R	PageName (see page 24686)	Denotes the logical name for the Web page.
R	PageTitle (see page 24686)	Indicates the title that appears on the Web page.

176.2.2.3.1 IWebPageInfo.HasViewAccess Property

Indicates whether the current user has access rights to view the Web page.

Pascal

```
property HasViewAccess: Boolean;
```

C++

```
__property Boolean HasViewAccess;
```

Description

HasViewAccess is true if the current end user has view rights for the associated Web page, false otherwise.

See Also

HasViewRights (see page 24676)

176.2.2.3.2 IWebPageInfo.LoginRequired Property

Indicates whether callers must log in before accessing this Web page.

Pascal

```
property LoginRequired: Boolean;
```

C++

```
__property Boolean LoginRequired;
```

Description

Read LoginRequired to determine whether users must log in before they can access this Web page. The application can overrule the LoginRequired property by supplying an OnIsLoginRequired event handler to the page dispatcher.

See Also

OnIsLoginRequired (see page 24904)

TPageDispatcher (see page 24949)

176.2.2.3.3 IWebPageInfo.PageDescription Property

Describes the Web page.

Pascal

```
property PageDescription: string;
```

C++

```
__property AnsiString PageDescription;
```

Description

Read PageDescription to obtain a brief description of the Web page.

See Also

PageName (see page 24686)

PageTitle (see page 24686)

176.2.2.3.4 IWebPageInfo.PageHREF Property

Identifies the HREF to use for references to the associated Web page.

Pascal

```
property PageHREF: string;
```

C++

```
__property AnsiString PageHREF;
```

Description

Read PageHREF to obtain the HTTP path string that should be inserted in HTML code to create a hypertext link to the associated Web page.

See Also

PageName (see page 24686)

176.2.2.3.5 IWebPageInfo.PageIsPublished Property

Indicates whether the WebSnap application automatically displays the Web page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 24686).

Pascal

```
property PageIsPublished: Boolean;
```

C++

```
__property Boolean PageIsPublished;
```

Description

Read PageIsPublished to determine whether the page dispatcher (TPageDispatcher) automatically calls on the associated Web page module when it receives HTTP request messages that address this Web page by name. HTTP request messages address a Web page when the pathinfo portion of their target URL matches the value of the PageName (see page 24686) property.

When IsPublished is false, the WebSnap application must explicitly retrieve the content for the Web page and add it to a response message.

See Also

PageName (see page 24686)

TPageDispatcher (see page 24949)

176.2.2.3.6 IWebPageInfo.PageName Property

Denotes the logical name for the Web page.

Pascal

```
property PageName: string;
```

C++

```
__property AnsiString PageName;
```

Description

Read PageName to determine the name of the Web page. This name is used to identify the page in server-side scripts, and as the pathinfo portion of the URL for the page when the PageIsPublished (see page 24685) property is true.

See Also

PageTitle (see page 24686)

PageHREF (see page 24685)

PageIsPublished (see page 24685)

176.2.2.3.7 IWebPageInfo.PageTitle Property

Indicates the title that appears on the Web page.

Pascal

```
property PageTitle: string;
```

C++

```
__property AnsiString PageTitle;
```

Description

Read PageTitle to determine the title that appears on the Web page when viewed from a user's browser.

See Also

PageName (see page 24686)

176.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	WebContext (see page 24687)	Returns the current Web context

176.3.1 WebCntxt.WebContext Function

Returns the current Web context

Pascal

```
function WebContext: TAbstractWebContext;
```

C++

```
TAbstractWebContext WebContext();
```

File

WebCntxt

Description

Use the WebContext function to obtain a reference to the global Web context. This object stores information about the components that are available in the Web application as well as about the current request that is being processed.

176.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TFindPageOption (see page 24688)	This is record WebCntxt.TFindPageOption.

176.4.1 WebCntxt.TFindPageOption Enumeration

Pascal

```
TFindPageOption = (
  fpPublished,
  fpLoginRequired
);
```

C++

```
enum TFindPageOption {
  fpPublished,
  fpLoginRequired
};
```

File

WebCntxt

Description

This is record WebCntxt.TFindPageOption.

176.5 Types

The following table lists types in this documentation.

Types

Name	Description
TFindPageOptions (see page 24689)	This is type WebCntxt.TFindPageOptions.
TGetWebContextProc (see page 24689)	This is type WebCntxt.TGetWebContextProc.
TGetWebEndUserProc (see page 24690)	This is type WebCntxt.TGetWebEndUserProc.
TSetWebContextProc (see page 24690)	This is type WebCntxt.TSetWebContextProc.

176.5.1 WebCntxt.TFindPageOptions Type

Pascal

```
TFindPageOptions = set of TFindPageOption;
```

C++

```
TFindPageOption TFindPageOptions;
```

File

WebCntxt

Description

This is type WebCntxt.TFindPageOptions.

176.5.2 WebCntxt.TGetWebContextProc Type

Pascal

```
TGetWebContextProc = function : TAbstractWebContext;
```

C++

```
: TAbstractWebContext ( TGetWebContextProc)();
```

File

WebCntxt

Description

This is type WebCntxt.TGetWebContextProc.

176.5.3 WebCntxt.TGetWebEndUserProc Type

Pascal

```
TGetWebEndUserProc = function : IWebEndUser;
```

C++

```
: IWebEndUser ( TGetWebEndUserProc)();
```

File

WebCntxt

Description

This is type WebCntxt.TGetWebEndUserProc.

176.5.4 WebCntxt.TSetWebContextProc Type

Pascal

```
TSetWebContextProc = procedure (AWebContext: TAbstractWebContext);
```

C++

```
(AWebContext: TAbstractWebContext) ( TSetWebContextProc)();
```

File

WebCntxt

Description

This is type WebCntxt.TSetWebContextProc.

176.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
GetModuleFileNameProc (see page 24691)	This is variable WebCntxt.GetModuleFileNameProc.
GetWebContextProc (see page 24691)	This is variable WebCntxt.GetWebContextProc.
GetWebEndUserProc (see page 24692)	This is variable WebCntxt.GetWebEndUserProc.
SetWebContextProc (see page 24692)	This is variable WebCntxt.SetWebContextProc.

176.6.1 WebCntxt.GetModuleFileNameProc Variable

Pascal

```
GetModuleFileNameProc: function: string = nil;
```

C++

```
function: string GetModuleFileNameProc = nil;
```

File

WebCntxt

Description

This is variable WebCntxt.GetModuleFileNameProc.

176.6.2 WebCntxt.GetWebContextProc Variable

Pascal

```
GetWebContextProc: TGetWebContextProc;
```

C++

```
TGetWebContextProc GetWebContextProc;
```

File

WebCntxt

Description

This is variable WebCntxt.GetWebContextProc.

176.6.3 WebCntxt.GetWebEndUserProc Variable

Pascal

```
GetWebEndUserProc: TGetWebEndUserProc;
```

C++

```
TGetWebEndUserProc GetWebEndUserProc;
```

File

WebCntxt

Description

This is variable WebCntxt.GetWebEndUserProc.

176.6.4 WebCntxt.SetWebContextProc Variable

Pascal

```
SetWebContextProc: TSetWebContextProc;
```

C++

```
TSetWebContextProc SetWebContextProc;
```

File

WebCntxt

Description

This is variable WebCntxt.SetWebContextProc.

177 WebComp Namespace

This is namespace WebComp.

Topics

Name	Description
Classes (see page 24695)	The following table lists classes in this documentation.
Interfaces (see page 24769)	The following table lists interfaces in this documentation.
Functions (see page 24796)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 24799)	The following table lists structs, records, enums in this documentation.
Types (see page 24801)	The following table lists types in this documentation.
Constants (see page 24804)	The following table lists constants in this documentation.

Classes

	Name	Description
	TLayoutAttributes (see page 24695)	TLayoutAttributes describes how an HTML control should be laid out.
	TLayoutWebContent (see page 24698)	TLayoutWebContent lets child Web items adhere to layout rules imposed by their parent.
	TWebComponentClassDescription (see page 24706)	This is class WebComp.TWebComponentClassDescription.
	TWebComponentList (see page 24708)	TWebComponentList maintains a list of Web item components.
	TWebComponentsEditorHelper (see page 24719)	This is class WebComp.TWebComponentsEditorHelper.
	TWebContainedComponent (see page 24722)	TWebContainedComponent is the base class for Web items that generate HTML on behalf of another component.
	TWebContainedComponentHelper (see page 24731)	TWebContainedComponentHelper implements the common behavior of components that generate HTML on behalf of another component.
	TWebContainedContainerComponent (see page 24738)	TWebContainedContainerComponent is the base class for Web items that generate HTML on behalf of another component and also use Web items to generate HTML.
	TWebContainerComponent (see page 24751)	TWebContainerComponent is the base class for components that use Web items to generate HTML.
	TWebContainerComponentHelper (see page 24758)	TWebContainerComponentHelper implements the common behavior of components that maintain a list of Web items to generate HTML.
	TWebContentOptions (see page 24766)	TWebContentOptions controls what a Web item can include in its generated HTML.

Constants

Name	Description
EnumRegisteredWebComponentsProc (see page 24804)	This is constant WebComp.EnumRegisteredWebComponentsProc.
FindWebComponentEditorHelperProc (see page 24804)	This is constant WebComp.FindWebComponentEditorHelperProc.

RegisterWebComponentsProc (see page 24805)	This is constant WebComp.RegisterWebComponentsProc.
UnRegisterWebComponentsProc (see page 24805)	This is constant WebComp.UnRegisterWebComponentsProc.

Functions

	Name	Description
	DesigningComponent (see page 24796)	This is function WebComp.DesigningComponent.
	EnumRegisteredWebComponents (see page 24796)	This is function WebComp.EnumRegisteredWebComponents.
	FindWebComponentEditorHelper (see page 24797)	This is function WebComp.FindWebComponentEditorHelper.
	QualifyFileName (see page 24797)	This is function WebComp.QualifyFileName.
	RegisterWebComponents (see page 24797)	This is function WebComp.RegisterWebComponents.
	UnRegisterWebComponents (see page 24798)	This is function WebComp.UnRegisterWebComponents.
	ValidateWebParent (see page 24798)	This is function WebComp.ValidateWebParent.

Interfaces

	Name	Description
	IAddFieldsEditor (see page 24769)	This is class WebComp.IAddFieldsEditor.
	IGetWebComponentList (see page 24771)	This is class WebComp.IGetWebComponentList.
	IGetWebComponentNodeName (see page 24772)	This is class WebComp.IGetWebComponentNodeName.
	IGetXMLStream (see page 24773)	This is class WebComp.IGetXMLStream.
	ILayoutWebContent (see page 24774)	ILayoutWebContent lets child Web items adhere to layout rules imposed by their parent.
	INotifyActivateModule (see page 24776)	This is class WebComp.INotifyActivateModule.
	INotifyWebActivate (see page 24777)	This is class WebComp.INotifyWebActivate.
	IProducerEditorViewSupport (see page 24779)	This is class WebComp.IProducerEditorViewSupport.
	ISetWebContentOptions (see page 24781)	This is class WebComp.ISetWebContentOptions.
	ITopLevelWebComponent (see page 24782)	This is class WebComp.ITopLevelWebComponent.
	IWebComponent (see page 24782)	IWebComponent is the interface supported by components that TWebComponentList (see page 24708) manages.
	IWebComponentCommands (see page 24786)	This is class WebComp.IWebComponentCommands.
	IWebComponentContainer (see page 24788)	This is class WebComp.IWebComponentContainer.
	IWebComponentEditor (see page 24790)	IWebComponentEditor enables a Web item to be assigned children in the IDE.
	IWebComponentEditorHelper (see page 24791)	This is class WebComp.IWebComponentEditorHelper.
	IWebContent (see page 24792)	IWebContent is an interface that allows components page producers and Web items to access the HTML generated by other components.
	IWebVariablesContainerName (see page 24794)	This is class WebComp.IWebVariablesContainerName.

Structs, Records, Enums

	Name	Description
	TLabelPosition (see page 24799)	This is record WebComp.TLabelPosition.
	TWebContentFlag (see page 24799)	This is record WebComp.TWebContentFlag.

Types

Name	Description
TEnumWebComponentsProc (see page 24801)	This is type WebComp.TEnumWebComponentsProc.
TGetComponentListEvent (see page 24801)	This is type WebComp.TGetComponentListEvent.
TLayout (see page 24802)	TLayout is the type for helper objects that generate layout code for a Web item.
TSetComponentListEvent (see page 24802)	This is type WebComp.TSetComponentListEvent.
TWebComponentContainer (see page 24802)	TWebComponentContainer is the type for components that maintain a list of Web items.
TWebContentFlags (see page 24803)	This is type WebComp.TWebContentFlags.

177.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TLayoutAttributes (see page 24695)	TLayoutAttributes describes how an HTML control should be laid out.
	TLayoutWebContent (see page 24698)	TLayoutWebContent lets child Web items adhere to layout rules imposed by their parent.
	TWebComponentClassDescription (see page 24706)	This is class WebComp.TWebComponentClassDescription.
	TWebComponentList (see page 24708)	TWebComponentList maintains a list of Web item components.
	TWebComponentsEditorHelper (see page 24719)	This is class WebComp.TWebComponentsEditorHelper.
	TWebContainedComponent (see page 24722)	TWebContainedComponent is the base class for Web items that generate HTML on behalf of another component.
	TWebContainedComponentHelper (see page 24731)	TWebContainedComponentHelper implements the common behavior of components that generate HTML on behalf of another component.
	TWebContainedContainerComponent (see page 24738)	TWebContainedContainerComponent is the base class for Web items that generate HTML on behalf of another component and also use Web items to generate HTML.
	TWebContainerComponent (see page 24751)	TWebContainerComponent is the base class for components that use Web items to generate HTML.
	TWebContainerComponentHelper (see page 24758)	TWebContainerComponentHelper implements the common behavior of components that maintain a list of Web items to generate HTML.
	TWebContentOptions (see page 24766)	TWebContentOptions controls what a Web item can include in its generated HTML.

177.1.1 TLayoutAttributes Class

TLayoutAttributes describes how an HTML control should be laid out.

Class Hierarchy

Pascal

```
TLayoutAttributes = class(TObject);
```

C++

```
class TLayoutAttributes : public TObject;
```

File

WebComp

Description

TLayoutAttributes embodies a set of flags that describe how an HTML control should be laid out. These flags form the value of several public fields of the object, including

LabelAttributes (see page 24697): a string that describes style attributes that apply to the control's label (if any).

LabelPosition (see page 24697): an enumerated value describing the the position of the label (if any) relative to the control. Possible values are lposLeft (to the left of the control), lposRight (to the right of the control), lposAbove (above the control) and lposBelow (below the control).

ControlAttributes (see page 24697): a string that describes style attributes that apply to the control.

See Also

IWebContent (see page 24792)

177.1.1.1 TLayoutAttributes Members

The following tables list the members exposed by TLayoutAttributes.

Public Fields

	Name	Description
	ControlAttributes (see page 24697)	This is ControlAttributes, a member of class TLayoutAttributes.
	LabelAttributes (see page 24697)	This is LabelAttributes, a member of class TLayoutAttributes.
	LabelPosition (see page 24697)	This is LabelPosition, a member of class TLayoutAttributes.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 24696)	Creates a new TLayoutAttributes (see page 24695) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

177.1.1.2 TLayoutAttributes.Create Constructor

Creates a new TLayoutAttributes (see page 24695) object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TLayoutAttributes();
```

Description

Typically, applications do not instantiate TLayoutAttributes (see page 24695). Instead, each Web item that generates HTML for an InternetExpress application instantiates an internal TLayoutAttributes (see page 24695) object and uses it to encode property values that affect the layout of its generated HTML.

177.1.1.3 TLayoutAttributes Fields

The fields of the TLayoutAttributes class are listed here.

Public Fields

	Name	Description
	ControlAttributes (see page 24697)	This is ControlAttributes, a member of class TLayoutAttributes.
	LabelAttributes (see page 24697)	This is LabelAttributes, a member of class TLayoutAttributes.
	LabelPosition (see page 24697)	This is LabelPosition, a member of class TLayoutAttributes.

177.1.1.3.1 TLayoutAttributes.ControlAttributes Field

Pascal

```
ControlAttributes: string;
```

C++

```
AnsiString ControlAttributes;
```

Description

This is ControlAttributes, a member of class TLayoutAttributes.

177.1.1.3.2 TLayoutAttributes.LabelAttributes Field

Pascal

```
LabelAttributes: string;
```

C++

```
AnsiString LabelAttributes;
```

Description

This is LabelAttributes, a member of class TLayoutAttributes.

177.1.1.3.3 TLayoutAttributes.LabelPosition Field

Pascal

```
LabelPosition: TLabelPosition;
```

C++

```
TLabelPosition LabelPosition;
```

Description

This is LabelPosition, a member of class TLayoutAttributes.

177.1.2 TLayoutWebContent Class

TLayoutWebContent lets child Web items adhere to layout rules imposed by their parent.

Class Hierarchy

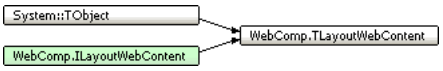

Pascal

```
TLayoutWebContent = class(TObject, ILayoutWebContent);
```

C++

```
class TLayoutWebContent : public TObject, public ILayoutWebContent;
```

File

WebComp

Description

TLayoutWebContent is the base class for objects that implement the ILayoutWebContent (see page 24774) interface. Parent components pass a TLayoutWebContent descendant to child Web items as the ParentLayout (see page 24706) parameter of their Content method. This object provides helper functions that the child items use to fit their generated content into the parent object's HTML structure (typically a <TABLE> tag). For example, the TLayoutWebContent methods may add <TD> tags around generated HTML that fills the cells of a parent's HTML table.

Do not create instances of TLayoutWebContent. This implementation of the ILayoutWebContent (see page 24774) methods consists of abstract or, in C++ terminology, pure virtual methods. Instead, use a TLayoutWebContent descendant such as TFormLayout or TAdapterFormLayout, which supplies an implementation for those methods.

See Also

TLayout (see page 24802)

IWebContent (see page 24792)

ILayoutWebContent (see page 24774)

TFormLayout (see page 16656)

TAdapterFormLayout

177.1.2.1 TLayoutWebContent Members

The following tables list the members exposed by TLayoutWebContent.

Protected Methods

	Name	Description
	_AddRef (see page 24700)	Responds when the caller acquires a reference to this object's interface.
	_Release (see page 24701)	Responds when the caller releases a reference to this object's interface.

	ImplLayoutButton (see page 24701)	Provides the underlying implementation for the LayoutButton (see page 24703) method.
	ImplLayoutField (see page 24702)	Provides the underlying implementation for the LayoutField (see page 24704) method.
	ImplLayoutLabelAndField (see page 24702)	Provides the underlying implementation for the LayoutLabelAndField (see page 24704) method.
	ImplLayoutTable (see page 24703)	Provides the underlying implementation for the LayoutTable (see page 24705) method.
	LayoutButton (see page 24703)	Converts the HTML declaring a button into a form the associated Web item can use.
	LayoutField (see page 24704)	Converts the HTML declaring a field into a form the associated Web item can use.
	LayoutLabelAndField (see page 24704)	Converts the HTML declaring a labeled field into a form the associated Web item can use.
	LayoutTable (see page 24705)	This is LayoutTable, a member of class TLayoutWebContent.
	QueryInterface (see page 24705)	Returns a reference to a specified interface if the object supports that interface.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 24700)	Creates a new TLayoutWebContent (see page 24698) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	ParentLayout (see page 24706)	Provides access to the layout object of the Web item that is the parent of the Web item to which this object belongs.

177.1.2.2 TLayoutWebContent.Create Constructor

Creates a new TLayoutWebContent (see page 24698) object.

Pascal

```
constructor Create(AParentLayout: TLayout);
```

C++

```
__fastcall TLayoutWebContent(TLayout AParentLayout);
```

Description

Call Create to instantiate a TLayoutWebContent (see page 24698) descendant to use when fetching the content produced by child Web items.

AParentLayout is the layout object of the parent Web item (the grandparent of the child Web items that call this object's methods). AParentLayout becomes the value of the ParentLayout (see page 24706) property.

See Also

ParentLayout (see page 24706)

177.1.2.3 TLayoutWebContent Methods

The methods of the TLayoutWebContent class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 24700)	Responds when the caller acquires a reference to this object's interface.
	_Release (see page 24701)	Responds when the caller releases a reference to this object's interface.
A	ImplLayoutButton (see page 24701)	Provides the underlying implementation for the LayoutButton (see page 24703) method.
A	ImplLayoutField (see page 24702)	Provides the underlying implementation for the LayoutField (see page 24704) method.
A	ImplLayoutLabelAndField (see page 24702)	Provides the underlying implementation for the LayoutLabelAndField (see page 24704) method.
A	ImplLayoutTable (see page 24703)	Provides the underlying implementation for the LayoutTable (see page 24705) method.
	LayoutButton (see page 24703)	Converts the HTML declaring a button into a form the associated Web item can use.
	LayoutField (see page 24704)	Converts the HTML declaring a field into a form the associated Web item can use.
	LayoutLabelAndField (see page 24704)	Converts the HTML declaring a labeled field into a form the associated Web item can use.
	LayoutTable (see page 24705)	This is LayoutTable, a member of class TLayoutWebContent.
V	QueryInterface (see page 24705)	Returns a reference to a specified interface if the object supports that interface.

177.1.2.3.1 TLayoutWebContent._AddRef Method

Responds when the caller acquires a reference to this object's interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

In Delphi, when an application uses an interface that is implemented by TLayoutWebContent (see page 24698) or one of its descendants, it automatically calls __AddRef. Typically, interfaced objects use the __AddRef method to increment a reference count, which is then decremented when the application releases the interface. In this typical scenario, when the reference count drops to zero, the object frees itself.

TLayoutWebContent (see page 24698) implements its own version of the interface methods __AddRef and _Release (see page 24701) that avoid this typical behavior. This prevents the object from being freed when its reference count drops to zero. In TLayoutWebContent (see page 24698), __AddRef always returns –1.

See Also

_Release (see page 24701)

_AddRef (see page 22397)

177.1.2.3.2 TLayoutWebContent._Release Method

Responds when the caller releases a reference to this object's interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

In Delphi, when an application releases an interface that is implemented by TLayoutWebContent (see page 24698) or one of its descendants, it automatically calls __Release. Typically, interfaced objects use the __Release method to decrement a reference count. When the reference count drops to zero, the object is automatically freed.

TLayoutWebContent (see page 24698) implements its own version of the interface methods _AddRef (see page 24700) and __Release that avoid this typical behavior. This prevents the object from being freed when its reference count drops to zero. In TLayoutWebContent (see page 24698), __Release always returns –1.

See Also

_AddRef (see page 24700)

_Release (see page 22397)

177.1.2.3.3 TLayoutWebContent.ImplLayoutButton Method

Provides the underlying implementation for the LayoutButton (see page 24703) method.

Pascal

```
function ImplLayoutButton(const HTMLButton: string; Attributes: TLayoutAttributes): string;
virtual; abstract;
```

C++

```
virtual __fastcall AnsiString ImplLayoutButton(const AnsiString HTMLButton, TLayoutAttributes
Attributes) = 0;
```

Description

The LayoutButton (see page 24703) method calls ImplLayoutButton to convert the HTML that declares a button (typically an <INPUT> tag) into the form that the associated Web item requires.

HTMLButton is the HTML that declares a button. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the button.

LayoutButton (see page 24703) returns the HTML for the button in a form that the associated Web item can use.

In TLayoutWebContent (see page 24698), ImplLayoutButton is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutButton (see page 24703) method.

See Also

LayoutButton (see page 24703)

177.1.2.3.4 TLayoutWebContent.ImplLayoutField Method

Provides the underlying implementation for the LayoutField (see page 24704) method.

Pascal

```
function ImplLayoutField(const HTMLField: string; Attributes: TLayoutAttributes): string;
virtual; abstract;
```

C++

```
virtual __fastcall AnsiString ImplLayoutField(const AnsiString HTMLField, TLayoutAttributes
Attributes) = 0;
```

Description

The LayoutField (see page 24704) method calls ImplLayoutField to convert the HTML that declares a field (an input control that represents a data field or a named value) into the form that the associated Web item requires.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field.

LayoutField (see page 24704) returns the HTML for the field in a form that the associated Web item can use.

In TLayoutWebContent (see page 24698), ImplLayoutField is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutField (see page 24704) method.

See Also

LayoutField (see page 24704)

177.1.2.3.5 TLayoutWebContent.ImplLayoutLabelAndField Method

Provides the underlying implementation for the LayoutLabelAndField (see page 24704) method.

Pascal

```
function ImplLayoutLabelAndField(const HTMLLabel: string; const HTMLField: string; Attributes:
TLayoutAttributes): string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString ImplLayoutLabelAndField(const AnsiString HTMLLabel, const
AnsiString HTMLField, TLayoutAttributes Attributes) = 0;
```

Description

The LayoutLabelAndField (see page 24704) method calls ImplLayoutLabelAndField to convert the HTML that declares a labeled field (an input control that represents a data field or a named value along with its label) into the form that the associated Web item requires.

HTMLLabel is the HTML that declares the label. This is produced by the child Web item.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field and its label.

LayoutLabelAndField (see page 24704) returns the HTML for the field and label in a form that the associated Web item can use.

In TLayoutWebContent (see page 24698), ImplLayoutLabelAndField is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutLabelAndField (see page 24704) method.

See Also

LayoutLabelAndField (see page 24704)

177.1.2.3.6 TLayoutWebContent.ImplLayoutTable Method

Provides the underlying implementation for the LayoutTable (see page 24705) method.

Pascal

```
function ImplLayoutTable(const HTMLTable: string; Attributes: TLayoutAttributes): string;
virtual; abstract;
```

C++

```
virtual __fastcall AnsiString ImplLayoutTable(const AnsiString HTMLTable, TLayoutAttributes
Attributes) = 0;
```

Description

The LayoutTable (see page 24705) method calls ImplLayoutTable to convert the HTML that declares a table (typically a structure used to layout the content produced by the child Web item's children) into the form that the associated Web item requires.

HTMLTable is the HTML that declares the table. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the table.

LayoutTable (see page 24705) returns the HTML for the table in a form that the associated Web item can use.

In TLayoutWebContent (see page 24698), ImplLayoutTable is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutTable (see page 24705) method.

See Also

layouttable (see page 24705)

177.1.2.3.7 TLayoutWebContent.LayoutButton Method

Converts the HTML declaring a button into a form the associated Web item can use.

Pascal

```
function LayoutButton(const HTMLButton: string; Attributes: TLayoutAttributes): string;
```

C++

```
__fastcall AnsiString LayoutButton(const AnsiString HTMLButton, TLayoutAttributes Attributes);
```

Description

Call LayoutButton to convert the HTML that declares a button (typically an <INPUT> tag) into the form that the associated Web item requires.

HTMLButton is the HTML that declares a button. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the button.

LayoutButton returns the HTML for the button in a form that the associated Web item can use.

LayoutButton delegates the task of converting the HTML for the button to the ImplLayoutButton (see page 24701) method.

See Also

ImplLayoutButton (see page 24701)

177.1.2.3.8 TLayoutWebContent.LayoutField Method

Converts the HTML declaring a field into a form the associated Web item can use.

Pascal

```
function LayoutField(const HTMLField: string; Attributes: TLayoutAttributes): string;
```

C++

```
__fastcall AnsiString LayoutField(const AnsiString HTMLField, TLayoutAttributes Attributes);
```

Description

Call LayoutField to convert the HTML that declares a field (an input control that represents a data field or a named value) into the form that the associated Web item requires.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field.

LayoutField returns the HTML for the field in a form that the associated Web item can use.

LayoutField delegates the task of converting the HTML for the field to the ImplLayoutField (see page 24702) method.

See Also

ImplLayoutField (see page 24702)

177.1.2.3.9 TLayoutWebContent.LayoutLabelAndField Method

Converts the HTML declaring a labeled field into a form the associated Web item can use.

Pascal

```
function LayoutLabelAndField(const HTMLLabel: string; const HTMLField: string; Attributes:
TLayoutAttributes): string;
```

C++

```
__fastcall AnsiString LayoutLabelAndField(const AnsiString HTMLLabel, const AnsiString
HTMLField, TLayoutAttributes Attributes);
```

Description

Call LayoutLabelAndField to convert the HTML that declares a labeled field (an input control that represents a data field or a named value along with its label) into the form that the associated Web item requires.

HTMLLabel is the HTML that declares the label. This is produced by the child Web item.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field and its label.

LayoutLabelAndField returns the HTML for the field and label in a form that the associated Web item can use.

LayoutLabelAndField delegates the task of converting the HTML for the label and field to the ImplLayoutLabelAndField (see page 24702) method.

See Also

ImplLayoutLabelAndField (see page 24702)

177.1.2.3.10 TLayoutWebContent.LayoutTable Method

Pascal

```
function LayoutTable(const HTMLTable: string; Attributes: TLayoutAttributes): string;
```

C++

```
__fastcall AnsiString LayoutTable(const AnsiString HTMLTable, TLayoutAttributes Attributes);
```

Description

This is LayoutTable, a member of class TLayoutWebContent.

177.1.2.3.11 TLayoutWebContent.QueryInterface Method

Returns a reference to a specified interface if the object supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the TLayoutWebContent (see page 24698) object supports the interface specified by IID. If so, QueryInterface sets the Obj parameter so that it points to an instance of the specified interface and returns 0 to indicate success.

If the layout object does not support the interface, QueryInterface returns a nonzero error code such as E_NoInterface.

IID is the desired interface.

Obj returns a reference to the interface if the TLayoutWebContent (see page 24698) object supports it.

See Also

_AddRef (see page 24700)

_Release (see page 24701)

177.1.2.4 TLayoutWebContent Properties

The properties of the TLayoutWebContent class are listed here.

Public Properties

	Name	Description
R	ParentLayout (see page 24706)	Provides access to the layout object of the Web item that is the parent of the Web item to which this object belongs.

177.1.2.4.1 TLayoutWebContent.ParentLayout Property

Provides access to the layout object of the Web item that is the parent of the Web item to which this object belongs.

Pascal

```
property ParentLayout: TLayout;
```

C++

```
__property TLayout ParentLayout;
```

Description

Web items form a hierarchy, where each child Web item generates its content on behalf of a parent object. This TLayoutWebContent (see page 24698) object is associated with a Web item, and supplies methods to help lay out the content of its child Web items. The associated Web item, in turn, may be the child Web item of another Web item. ParentLayout is the layout object of the associated Web item's parent item.

The TLayoutWebContent (see page 24698) object uses ParentLayout in the implementation of its methods so that they reflect the needs of the parent item as well as the needs of the associated Web item.

ParentLayout supports the ILayoutWebContent (see page 24774) interface. Typically, it is another TLayoutWebContent (see page 24698) descendant.

See Also

Create (see page 24700)

ILayoutWebContent (see page 24774)

177.1.3 TWebComponentClassDescription Class

Class Hierarchy

Pascal

```
TWebComponentClassDescription = class(TObject);
```

C++

```
class TWebComponentClassDescription : public TObject;
```

File

WebComp

Description

This is class WebComp.TWebComponentClassDescription.

177.1.3.1 TWebComponentClassDescription Members

The following tables list the members exposed by TWebComponentClassDescription.

Public Fields

	Name	Description
	ComponentClass (see page 24708)	This is ComponentClass, a member of class TWebComponentClassDescription.
	ComponentClassName (see page 24708)	This is ComponentClassName, a member of class TWebComponentClassDescription.
	Editor (see page 24708)	This is Editor, a member of class TWebComponentClassDescription.
	FrameClass (see page 24708)	This is FrameClass, a member of class TWebComponentClassDescription.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

177.1.3.2 TWebComponentClassDescription Fields

The fields of the TWebComponentClassDescription class are listed here.

Public Fields

	Name	Description
	ComponentClass (see page 24708)	This is ComponentClass, a member of class TWebComponentClassDescription.
	ComponentClassName (see page 24708)	This is ComponentClassName, a member of class TWebComponentClassDescription.
	Editor (see page 24708)	This is Editor, a member of class TWebComponentClassDescription.
	FrameClass (see page 24708)	This is FrameClass, a member of class TWebComponentClassDescription.

177.1.3.2.1 TWebComponentClassDescription.ComponentClass Field

Pascal

```
ComponentClass: TComponentClass;
```

C++

```
TComponentClass ComponentClass;
```

Description

This is ComponentClass, a member of class TWebComponentClassDescription.

177.1.3.2.2 TWebComponentClassDescription.ComponentClassName Field

Pascal

```
ComponentClassName: string;
```

C++

```
AnsiString ComponentClassName;
```

Description

This is ComponentClassName, a member of class TWebComponentClassDescription.

177.1.3.2.3 TWebComponentClassDescription.Editor Field

Pascal

```
Editor: TObject;
```

C++

```
TObject * Editor;
```

Description

This is Editor, a member of class TWebComponentClassDescription.

177.1.3.2.4 TWebComponentClassDescription.FrameClass Field

Pascal

```
FrameClass: Boolean;
```

C++

```
Boolean FrameClass;
```

Description

This is FrameClass, a member of class TWebComponentClassDescription.

177.1.4 TWebComponentList Class

TWebComponentList maintains a list of Web item components.

Class Hierarchy

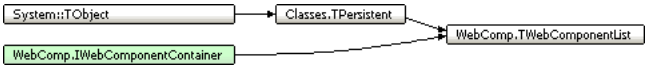

Pascal

```
TWebComponentList = class(TPersistent, IWebComponentContainer);
```

C++

```
class TWebComponentList : public TPersistent, public IWebComponentContainer;
```

File

WebComp

Description

TWebComponentList provides the structure for the hierarchy of Web items that generate HTML documents in a WebSnap or InternetExpress application. At the top of the hierarchy is a page producer (TINetXPageProducer or TAdapterPageProducer) or an adapter. The page producer assembles an HTML document, or the adapter provides information and performs tasks on behalf of an adapter page producer or server-side script. To perform their tasks, the root objects use a set of helper components. Each of these components, in turn, uses one or more helper components, and so on. Each component that uses helper components maintains a list of its helper components. That list is a TWebComponentList object.

Each component in the list must support the IWebComponent (see page 24782) interface. TWebComponentList uses this interface to coordinate the objects it manages. In addition, children of a page producer should support the IWebContent (see page 24792) interface. IWebContent (see page 24792) allows the parent component to extract HTML generated by its children.

Use the properties and methods of TWebComponentList to

Add (see page 24713) or delete the components in the list.

Locate and access components in the list.

Rearrange components in the list.

Access the parent component that uses the list.

See Also

IWebContent (see page 24792)

TBasePageItemsProducer (see page 18376)

TWebControlGroup (see page 16872)

TWebForm (see page 16922)

TInetXPageProducer (see page 17145)

IWebComponent (see page 24782)

TAdapterPageProducer (see page 19802)

177.1.4.1 TWebComponentList Members

The following tables list the members exposed by TWebComponentList.

Protected Methods

	Name	Description
	_AddRef (see page 24712)	This is _AddRef, a member of class TWebComponentList.
	_Release (see page 24712)	This is _Release, a member of class TWebComponentList.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 24713)	This is Changed, a member of class TWebComponentList.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetComponent (see page 24714)	This is GetComponent, a member of class TWebComponentList.
	GetCount (see page 24714)	This is GetCount, a member of class TWebComponentList.
V	GetOwner (see page 24714)	Returns the owner of an object.
V	NewComponent (see page 24715)	This is NewComponent, a member of class TWebComponentList.
V	QueryInterface (see page 24716)	This is QueryInterface, a member of class TWebComponentList.
	WebComponentContainer_GetComponent (see page 24717)	This is WebComponentContainer_GetComponent, a member of class TWebComponentList.
	WebComponentContainer_GetComponentCount (see page 24717)	This is WebComponentContainer_GetComponentCount, a member of class TWebComponentList.

Public Events

	Name	Description
	OnChange (see page 24719)	Occurs when a change occurs in the web component list.

Public Methods

	Name	Description
	Add (see page 24713)	Inserts a new component at the end of the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 24713)	Deletes all components from the list.
	Create (see page 24711)	Creates and initializes new TWebComponentList (see page 24708) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 24711)	Frees an instance of TWebComponentList (see page 24708).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetComponentCount (see page 24789)	This is GetComponentCount, a member of class IWebComponentContainer.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	IndexOf (see page 24715)	Returns the index of a specified component in the WebComponents (see page 24718) array.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LockNotify (see page 24715)	This is LockNotify, a member of class TWebComponentList.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NotifyLocked (see page 24715)	This is NotifyLocked, a member of class TWebComponentList.
	Remove (see page 24716)	Removes a component from the WebComponents (see page 24718) array.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentIndex (see page 24716)	Moves a component to a new position in the WebComponents (see page 24718) array.
	UnlockNotify (see page 24717)	This is UnlockNotify, a member of class TWebComponentList.

Public Properties

	Name	Description
R	ComponentCount (see page 24789)	This is ComponentCount, a member of class IWebComponentContainer.
R	Components (see page 24790)	This is Components, a member of class IWebComponentContainer.
R	Count (see page 24718)	Indicates the number of components in the list.
R	ParentComponent (see page 24718)	Identifies the component that uses this Web component list to maintain its Web Items.
R	WebComponents (see page 24718)	Lists the Web items in the list.

177.1.4.2 TWebComponentList.Create Constructor

Creates and initializes new TWebComponentList (see page 24708) object.

Pascal

```
constructor Create(AParentComponent: TComponent);
```

C++

```
__fastcall TWebComponentList(TComponent * AParentComponent);
```

Description

Call Create to instantiate TWebComponentList (see page 24708) at runtime. Most applications do not need to instantiate TWebComponentList (see page 24708). Instead, the Web items that use TWebComponentList (see page 24708) to manage their own Web items instantiate a TWebComponentList (see page 24708) from their constructors.

177.1.4.3 TWebComponentList.Destroy Destructor

Frees an instance of TWebComponentList (see page 24708).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebComponentList();
```

Description

Do not call Destroy directly. Instead, call Free. Free verifies that the TWebComponentList (see page 24708) reference is not nil, and only then calls Destroy.

Destroy frees the memory used to store the list of WebComponents (see page 24718).

Note: Destroy does not free the components in the list.

See Also

Free (see page 22300)

177.1.4.4 TWebComponentList Methods

The methods of the TWebComponentList class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 24712)	This is _AddRef, a member of class TWebComponentList.
	_Release (see page 24712)	This is _Release, a member of class TWebComponentList.
	Changed (see page 24713)	This is Changed, a member of class TWebComponentList.
	GetComponent (see page 24714)	This is GetComponent, a member of class TWebComponentList.
	GetCount (see page 24714)	This is GetCount, a member of class TWebComponentList.
V	GetOwner (see page 24714)	Returns the owner of an object.
V	NewComponent (see page 24715)	This is NewComponent, a member of class TWebComponentList.
V	QueryInterface (see page 24716)	This is QueryInterface, a member of class TWebComponentList.
	WebComponentContainer_GetComponent (see page 24717)	This is WebComponentContainer_GetComponent, a member of class TWebComponentList.
	WebComponentContainer_GetComponentCount (see page 24717)	This is WebComponentContainer_GetComponentCount, a member of class TWebComponentList.

Public Methods

	Name	Description
	Add (see page 24713)	Inserts a new component at the end of the list.
	Clear (see page 24713)	Deletes all components from the list.
	IndexOf (see page 24715)	Returns the index of a specified component in the WebComponents (see page 24718) array.
	LockNotify (see page 24715)	This is LockNotify, a member of class TWebComponentList.
	NotifyLocked (see page 24715)	This is NotifyLocked, a member of class TWebComponentList.
	Remove (see page 24716)	Removes a component from the WebComponents (see page 24718) array.
	SetComponentIndex (see page 24716)	Moves a component to a new position in the WebComponents (see page 24718) array.
	UnlockNotify (see page 24717)	This is UnlockNotify, a member of class TWebComponentList.

177.1.4.4.1 TWebComponentList._AddRef Method

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

This is _AddRef, a member of class TWebComponentList.

177.1.4.4.2 TWebComponentList._Release Method

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

This is _Release, a member of class TWebComponentList.

177.1.4.4.3 TWebComponentList.Add Method

Inserts a new component at the end of the list.

Pascal

```
procedure Add(Component: TComponent);
```

C++

```
__fastcall Add(TComponent * Component);
```

Description

Call Add to insert a component at the end of the WebComponents (see page 24718) array.

Add increments the Count (see page 24718) property. After calling Add, the value of Count (see page 24718) is one greater than the index of the inserted component in the WebComponents (see page 24718) array. To move this component to another position, use the SetComponentIndex (see page 24716) method.

See Also

WebComponents (see page 24718)

Count (see page 24718)

SetComponentIndex (see page 24716)

Remove (see page 24716)

177.1.4.4.4 TWebComponentList.Changed Method

Pascal

```
procedure Changed;
```

C++

```
__fastcall Changed();
```

Description

This is Changed, a member of class TWebComponentList.

177.1.4.4.5 TWebComponentList.Clear Method

Deletes all components from the list.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the WebComponents (see page 24718) array and set Count (see page 24718) to 0.

See Also

Count (see page 24718)

Remove (see page 24716)

WebComponents (see page 24718)

177.1.4.4.6 TWebComponentList.GetComponent Method

Pascal

```
function GetComponent(Index: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetComponent(int Index);
```

Description

This is GetComponent, a member of class TWebComponentList.

177.1.4.4.7 TWebComponentList.GetCount Method

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

This is GetCount, a member of class TWebComponentList.

177.1.4.4.8 TWebComponentList.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

177.1.4.4.9 TWebComponentList.IndexOf Method

Returns the index of a specified component in the WebComponents (see page 24718) array.

Pascal

```
function IndexOf(Component: TComponent): Integer;
```

C++

```
__fastcall int IndexOf(TComponent * Component);
```

Description

Call IndexOf to get the index for a component in the WebComponents (see page 24718) array. Specify the desired component as the Component parameter.

The first component in the array has index 0, the second component has index 1, and so on. If a component is not in the list, IndexOf returns -1. If a component appears more than once in the array, IndexOf returns the index of the first appearance.

See Also

WebComponents (see page 24718)

177.1.4.4.10 TWebComponentList.LockNotify Method

Pascal

```
procedure LockNotify;
```

C++

```
__fastcall LockNotify();
```

Description

This is LockNotify, a member of class TWebComponentList.

177.1.4.4.11 TWebComponentList.NewComponent Method

Pascal

```
function NewComponent(AOwner: TComponent): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * NewComponent(TComponent * AOwner);
```

Description

This is NewComponent, a member of class TWebComponentList.

177.1.4.4.12 TWebComponentList.NotifyLocked Method

Pascal

```
function NotifyLocked: Boolean;
```

C++

```
__fastcall Boolean NotifyLocked();
```

Description

This is NotifyLocked, a member of class TWebComponentList.

177.1.4.4.13 TWebComponentList.QueryInterface Method

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

This is QueryInterface, a member of class TWebComponentList.

177.1.4.4.14 TWebComponentList.Remove Method

Removes a component from the WebComponents (see page 24718) array.

Pascal

```
procedure Remove(Component: TComponent);
```

C++

```
__fastcall Remove(TComponent * Component);
```

Description

Call Remove to remove a specific component from the WebComponents (see page 24718) array. After a component is removed, all the WebComponents (see page 24718) that follow it are moved up in index position and Count (see page 24718) is reduced by one.

See Also

Add (see page 24713)

Count (see page 24718)

Clear (see page 24713)

WebComponents (see page 24718)

177.1.4.4.15 TWebComponentList.SetComponentIndex Method

Moves a component to a new position in the WebComponents (see page 24718) array.

Pascal

```
procedure SetComponentIndex(Component: TComponent; Value: Integer);
```

C++

```
__fastcall SetComponentIndex(TComponent * Component, int Value);
```

Description

Call SetComponentIndex to move a component to a new position in the list. Component specifies the component to move. Value specifies the index it should have after it is moved. If Value is less than 0, the component is moved to the first position. If Value is greater than or equal to count, the component is moved to the last position.

See Also

WebComponents (see page 24718)

177.1.4.4.16 TWebComponentList.UnlockNotify Method

Pascal

```
procedure UnlockNotify;
```

C++

```
__fastcall UnlockNotify();
```

Description

This is UnlockNotify, a member of class TWebComponentList.

177.1.4.4.17 TWebComponentList.WebComponentContainer_GetComponent Method

Pascal

```
function WebComponentContainer_GetComponent(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * WebComponentContainer_GetComponent(int I);
```

Description

This is WebComponentContainer_GetComponent, a member of class TWebComponentList.

177.1.4.4.18 TWebComponentList.WebComponentContainer_GetComponentCount Method

Pascal

```
function WebComponentContainer_GetComponentCount: Integer;
```

C++

```
__fastcall int WebComponentContainer_GetComponentCount();
```

Description

This is WebComponentContainer_GetComponentCount, a member of class TWebComponentList.

177.1.4.5 TWebComponentList Properties

The properties of the TWebComponentList class are listed here.

Public Properties

	Name	Description
R	Count (see page 24718)	Indicates the number of components in the list.
R	ParentComponent (see page 24718)	Identifies the component that uses this Web component list to maintain its Web Items.
R	WebComponents (see page 24718)	Lists the Web items in the list.

177.1.4.5.1 TWebComponentList.Count Property

Indicates the number of components in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read Count to determine the number of components in the WebComponents (see page 24718) array. Count is a read-only property. To change the size of the WebComponents (see page 24718) array, add or remove components using the Add (see page 24713) and Remove (see page 24716) methods.

See Also

WebComponents (see page 24718)

Add (see page 24713)

Remove (see page 24716)

177.1.4.5.2 TWebComponentList.ParentComponent Property

Identifies the component that uses this Web component list to maintain its Web Items.

Pascal

```
property ParentComponent: TComponent;
```

C++

```
__property TComponent * ParentComponent;
```

Description

Read ParentComponent to access the parent of the Web Items in the list. The value of ParentComponent is assigned in the constructor when the parent first instantiates the Web component list.

See Also

Create (see page 24711)

177.1.4.5.3 TWebComponentList.WebComponents Property

Lists the Web items in the list.

Pascal

```
property WebComponents [Index: Integer]: TComponent;
```

C++

```
__property TComponent * WebComponents[int Index];
```

Description

Use WebComponents to access the Web components listed by TWebComponentList (see page 24708). Use WebComponents with the Count (see page 24718) property to iterate through all of the components in the list. Use the Add (see page 24713) or Remove (see page 24716) method to add or remove components from the list.

WebComponents is the default property for TWebComponentList (see page 24708). This means that the property name can be omitted when accessing a component in the list. Thus, the line

```
WebItem := InetXPageProducer1.WebPageItems.WebComponents[0];
WebItem = InetXPageProducer1->WebPageItems->WebComponents[0];
```

can be written

```
WebItem := InetXPageProducer1.WebPageItems[0];
WebItem = (*(InetXPageProducer1->WebPageItems))[0];
```

See Also

Count (see page 24718)

Add (see page 24713)

Remove (see page 24716)

operator_sb

177.1.4.6 TWebComponentList Events

The events of the TWebComponentList class are listed here.

Public Events

	Name	Description
	OnChange (see page 24719)	Occurs when a change occurs in the web component list.

177.1.4.6.1 TWebComponentList.OnChange Event

Occurs when a change occurs in the web component list.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Use OnChange to write an event handler to process an action when a change occurs in the web component list, such as when a new component is added.

177.1.5 TWebComponentsEditorHelper Class

Class Hierarchy

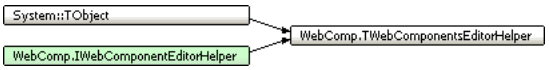

Pascal

```
TWebComponentsEditorHelper = class(TObject, IWebComponentEditorHelper);
```

C++

```
class TWebComponentsEditorHelper : public TObject, public IWebComponentEditorHelper;
```

File

WebComp

Description

This is class WebComp.TWebComponentsEditorHelper.

177.1.5.1 TWebComponentsEditorHelper Members

The following tables list the members exposed by TWebComponentsEditorHelper.

Protected Methods

	Name	Description
	_AddRef (see page 24721)	This is _AddRef, a member of class TWebComponentsEditorHelper.
	_Release (see page 24721)	This is _Release, a member of class TWebComponentsEditorHelper.
	CanAddClassHelper (see page 24721)	This is CanAddClassHelper, a member of class TWebComponentsEditorHelper.
V	ImplCanAddClassHelper (see page 24721)	This is ImplCanAddClassHelper, a member of class TWebComponentsEditorHelper.
V	QueryInterface (see page 24722)	This is QueryInterface, a member of class TWebComponentsEditorHelper.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

177.1.5.2 TWebComponentsEditorHelper Methods

The methods of the TWebComponentsEditorHelper class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 24721)	This is _AddRef, a member of class TWebComponentsEditorHelper.
	_Release (see page 24721)	This is _Release, a member of class TWebComponentsEditorHelper.
	CanAddClassHelper (see page 24721)	This is CanAddClassHelper, a member of class TWebComponentsEditorHelper.
V	ImplCanAddClassHelper (see page 24721)	This is ImplCanAddClassHelper, a member of class TWebComponentsEditorHelper.
V	QueryInterface (see page 24722)	This is QueryInterface, a member of class TWebComponentsEditorHelper.

177.1.5.2.1 TWebComponentsEditorHelper._AddRef Method

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

This is _AddRef, a member of class TWebComponentsEditorHelper.

177.1.5.2.2 TWebComponentsEditorHelper._Release Method

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

This is _Release, a member of class TWebComponentsEditorHelper.

177.1.5.2.3 TWebComponentsEditorHelper.CanAddClassHelper Method

Pascal

```
function CanAddClassHelper(AEditor: TComponent; AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClassHelper(TComponent * AEditor, TComponent * AParent, TClass
AClass);
```

Description

This is CanAddClassHelper, a member of class TWebComponentsEditorHelper.

177.1.5.2.4 TWebComponentsEditorHelper.ImplCanAddClassHelper Method

Pascal

```
function ImplCanAddClassHelper(AEditor: TComponent; AParent: TComponent; AClass: TClass):
```

```
Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddClassHelper(TComponent * AEditor, TComponent * AParent,
TClass AClass);
```

Description

This is ImplCanAddClassHelper, a member of class TWebComponentsEditorHelper.

177.1.5.2.5 TWebComponentsEditorHelper.QueryInterface Method

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

This is QueryInterface, a member of class TWebComponentsEditorHelper.

177.1.6 TWebContainedComponent Class

TWebContainedComponent is the base class for Web items that generate HTML on behalf of another component.

Class Hierarchy

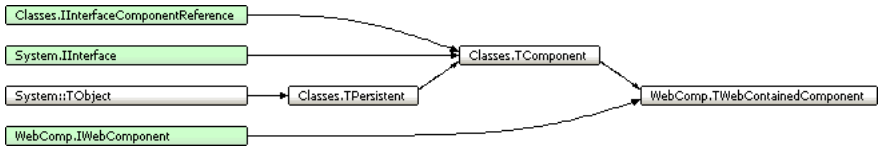

Pascal

```
TWebContainedComponent = class(TComponent, IWebComponent);
```

C++

```
class TWebContainedComponent : public TComponent, public IWebComponent;
```

File

WebComp

Description

Use TWebContainedComponent as a base class for Web items. Web items are items that generate HTML statements on behalf of another component. The component that uses the generated HTML has a property whose type is a TWebComponentList (see page 24708) object that lists a set of its Web items.

TWebContainedComponent implements the logic that fits a component into the hierarchy of Web items. TWebContainedComponent descendants act as leaf nodes in this hierarchy. That is, they do not have their own Web items that they use to generate HTML.

In order to allow TWebComponentList (see page 24708) to manage the Web item, TWebContainedComponent implements the IWebComponent (see page 24782) interface.

Do not create instances of TWebContainedComponent. TWebContainedComponent does not produce any HTML, but merely

handles the relationship to a managing TWebComponentList (see page 24708) object.

See Also

TWebComponentList (see page 24708)

IWebComponent (see page 24782)

177.1.6.1 TWebContainedComponent Members

The following tables list the members exposed by TWebContainedComponent.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.

	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.

	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.

	Tag (see page 2808)	Stores an integer value as part of a component.

177.1.6.2 TWebContainedComponent.Create Constructor

Creates and initializes an instance of TWebContainedComponent (see page 24722).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebContainedComponent(TComponent * AOwner);
```

Description

Call Create to instantiate a contained component at runtime. Web items added at design time are instantiated automatically.

AOwner is the component, typically a Web module, that becomes the value of the Owner property. The Owner is responsible for initiating the loading and saving of this component, and for freeing the TWebContainedComponent (see page 24722) instance.

See Also

Owner (see page 2807)

177.1.6.3 TWebContainedComponent.Destroy Destructor

Frees an instance of TWebContainedComponent (see page 24722).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebContainedComponent();
```

Description

Destroy removes this component from its parent's list of components that generate HTML before freeing the memory used by TWebContainedComponent (see page 24722).

177.1.6.4 TWebContainedComponent Methods

The methods of the TWebContainedComponent class are listed here.

Protected Methods

	Name	Description
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.

Public Methods

	Name	Description
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.

177.1.6.4.1 TWebContainedComponent.GetContainerList Method

Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.

Pascal

```
function GetContainerList: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetContainerList();
```

Description

Applications can't access the protected GetContainerList method. It is used internally to access the object manages this item along with the other Web items of the Web parent.

See Also

IsDefaultField (see page 24728)

WebParent (see page 24731)

177.1.6.4.2 TWebContainedComponent.GetParentComponent Method

Returns the object that uses this component to generate HTML.

Pascal

```
function GetParentComponent: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetParentComponent();
```

Description

TWebContainedComponent (see page 24722) overrides the GetParentComponent method that is introduced in TComponent for the streaming system that loads and saves components. It returns the value of the WebParent (see page 24731) property (the object that uses the HTML generated by this component when assembling its own portion of an HTML document.)

See Also

GetParentComponent

HasParent (see page 24727)

WebParent (see page 24731)

177.1.6.4.3 TWebContainedComponent.HasParent Method

Indicates whether this component appears in another component's list of Web items.

Pascal

```
function HasParent: Boolean; override;
```

C++

```
virtual __fastcall Boolean HasParent();
```

Description

TWebContainedComponent (see page 24722) overrides the HasParent method that is introduced in TComponent for the streaming system that loads and saves components. It returns true if the WebParent (see page 24731) property is not nil (Delphi) or NULL (C++).

If the contained component has a parent, that parent component is responsible for writing this component to a stream.

See Also

HasParent

GetParentComponent (see page 24727)

WebParent (see page 24731)

177.1.6.4.4 TWebContainedComponent.IsDefaultField Method

Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.

Pascal

```
function IsDefaultField: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsDefaultField();
```

Description

TWebContainedComponent (see page 24722) uses IsDefaultField internally to locate the property on the Web parent that lists this item along with other Web items.

IsDefaultField returns true if the TWebComponentList (see page 24708) that contains this item represents a set of Web items that are used when the application does not explicitly assign Web items to the Web parent. IsDefaultField returns false if this component does not belong to such a default list.

See Also

GetContainerList (see page 24727)

WebParent (see page 24731)

TWebComponentList (see page 24708)

177.1.6.4.5 TWebContainedComponent.ParentChanged Method

Responds when the WebParent (see page 24731) property changes.

Pascal

```
procedure ParentChanged; virtual;
```

C++

```
virtual __fastcall ParentChanged();
```

Description

The protected ParentChanged method is called automatically when the WebParent (see page 24731) property changes. As

implemented in TWebContainedComponent (see page 24722), ParentChanged does nothing. Descendant classes can override this method to perform any necessary updates in response to the change in Web parent.

See Also

WebParent (see page 24731)

177.1.6.4.6 TWebContainedComponent.ReadState Method

Reads the component's published properties, stored data, and owned components from a reader object.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

ReadState is part of a sequence of calls used by the streaming system that loads and saves components. It reads the values of all the component's published properties, stored data, and owned components from the reader object passed in Reader. Do not call ReadState directly.

TWebContainedComponent (see page 24722) overrides ReadState to obtain the parent component from the reader, which it assigns as the value of the WebParent (see page 24731) property.

Note: Any descendant classes overriding ReadState should end with a call to the inherited ReadState method.

See Also

GetParentComponent (see page 24727)

WebParent (see page 24731)

177.1.6.4.7 TWebContainedComponent.SetParentComponent Method

Sets the WebParent (see page 24731) property to be the same as this component's parent.

Pascal

```
procedure SetParentComponent(AParent: TComponent); override;
```

C++

```
virtual __fastcall SetParentComponent(TComponent * AParent);
```

Description

TWebContainedComponent (see page 24722) overrides the SetParentComponent method that is introduced in TComponent for the streaming system that loads and saves components. It sets the WebParent (see page 24731) property to AParent, so that it is the same as the parent that loads and saves this component.

See Also

SetParentComponent

HasParent (see page 24727)

WebParent (see page 24731)

177.1.6.4.8 TWebContainedComponent.SetWebParent Method

Sets the value of the WebParent (see page 24731) property.

Pascal

```
procedure SetWebParent(Value: TComponent);
```

C++

```
__fastcall SetWebParent(TComponent * Value);
```

Description

SetWebParent is used internally to set the read-only WebParent (see page 24731) property. To assign a Web parent in an application, add this component to the TWebComponentList (see page 24708) object that implements a property on the new Web parent.

See Also

WebParent (see page 24731)

Add (see page 24713)

177.1.6.5 TWebContainedComponent Properties

The properties of the TWebContainedComponent class are listed here.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

177.1.6.5.1 TWebContainedComponent.WebContainedComponentHelper Property

Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.

Pascal

```
property WebContainedComponentHelper: TWebContainedComponentHelper;
```

C++

```
__property TWebContainedComponentHelper WebContainedComponentHelper;
```

Description

TWebContainedComponent (see page 24722) uses WebContainedComponentHelper internally to access the separate helper object that implements the IWebComponent (see page 24782) interface. In addition to providing the implementation for the IWebComponent (see page 24782) interface, WebContainedComponentHelper provides the underlying implementation for most of the other properties and methods that are common to components that appear in a list of Web items.

See Also

IWebComponent (see page 24782)

177.1.6.5.2 TWebContainedComponent.WebParent Property

Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Pascal

```
property WebParent: TComponent;
```

C++

```
__property TComponent * WebParent;
```

Description

TWebContainedComponent (see page 24722) uses WebParent to access the component on whose behalf it generates HTML. WebParent uses a TWebComponentList (see page 24708) object that includes this TWebContainedComponent (see page 24722) object to represent its Web items.

See Also

TWebComponentList (see page 24708)

177.1.7 TWebContainedComponentHelper Class

TWebContainedComponentHelper implements the common behavior of components that generate HTML on behalf of another component.

Class Hierarchy

Pascal

```
TWebContainedComponentHelper = class(TObject);
```

C++

```
class TWebContainedComponentHelper : public TObject;
```

File

WebComp

Description

Components that generate HTML for another component use TWebContainedComponentHelper to provide the underlying implementation of many of their properties and methods. TWebContainedComponentHelper implements the IWebComponent (see page 24782) interface, which allows a TWebComponentList (see page 24708) object manage a list of child components. In addition, it implements a number of properties and methods that allow a child Web item to interact with its parent item.

See Also

IWebComponent (see page 24782)

TWebContainedComponent (see page 24722)

TWebContainedContainerComponent (see page 24738)

177.1.7.1 TWebContainedComponentHelper Members

The following tables list the members exposed by TWebContainedComponentHelper.

Protected Events

	Name	Description
	OnParentChanged (see page 24738)	Occurs when the WebParent (see page 24737) property changes.

Protected Methods

	Name	Description
	GetContainer (see page 24734)	Returns the value of the WebParent (see page 24737) property.
	GetIndex (see page 24734)	Returns the position of the associated Web item in the parent item's array of Web items.
V	ParentChanged (see page 24735)	Generates an OnParentChanged (see page 24738) event.
	SetComponentList (see page 24735)	Sets the value of the ContainerList (see page 24737) property.
	SetContainer (see page 24736)	Sets the value of the WebParent (see page 24737) property.
	SetIndex (see page 24736)	Sets the position of the associated component in the list specified by ContainerList (see page 24737).
	SetWebParent (see page 24736)	Sets the value of the WebParent (see page 24737) property.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 24733)	Creates an instance of TWebContainedComponentHelper (see page 24731).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 24733)	Destroys an instance of TWebContainedComponentHelper (see page 24731).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsDefaultField (see page 24735)	Indicates whether the associated component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ContainerList (see page 24737)	Indicates the list that contains the associated component.
R	WebParent (see page 24737)	Indicates the component that uses the list of Web items that includes the associated component.

177.1.7.2 TWebContainedComponentHelper.Create Constructor

Creates an instance of TWebContainedComponentHelper (see page 24731).

Pascal

```
constructor Create(AComponent: TComponent; AOnParentChanged: TNotifyEvent);
```

C++

```
__fastcall TWebContainedComponentHelper(TComponent * AComponent, TNotifyEvent
AOnParentChanged);
```

Description

Applications do not create instances of TWebContainedComponentHelper (see page 24731). The component that uses TWebContainedComponentHelper (see page 24731) so that it can appear as a child item in a TWebComponentList (see page 24708) object instantiates TWebContainedComponentHelper (see page 24731).

AComponent is the component that uses TWebContainedComponentHelper (see page 24731) to implement those properties and methods that allow it to appear in a TWebComponentList (see page 24708) object.

AOnParentChanged is an event handler that AComponent supplies so that it can update itself when it is moved to the list of a different parent component. This becomes the event handler for the OnParentChanged (see page 24738) event.

See Also

OnParentChanged (see page 24738)

177.1.7.3 TWebContainedComponentHelper.Destroy Destructor

Destroys an instance of TWebContainedComponentHelper (see page 24731).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebContainedComponentHelper();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TWebContainedComponentHelper (see page 24731) object is not nil and only then calls Destroy.

See Also

Free (see page 22300)

177.1.7.4 TWebContainedComponentHelper Methods

The methods of the TWebContainedComponentHelper class are listed here.

Protected Methods

	Name	Description
	GetContainer (see page 24734)	Returns the value of the WebParent (see page 24737) property.
	GetIndex (see page 24734)	Returns the position of the associated Web item in the parent item's array of Web items.
V	ParentChanged (see page 24735)	Generates an OnParentChanged (see page 24738) event.
	SetComponentList (see page 24735)	Sets the value of the ContainerList (see page 24737) property.
	SetContainer (see page 24736)	Sets the value of the WebParent (see page 24737) property.
	SetIndex (see page 24736)	Sets the position of the associated component in the list specified by ContainerList (see page 24737).
	SetWebParent (see page 24736)	Sets the value of the WebParent (see page 24737) property.

Public Methods

	Name	Description
	IsDefaultField (see page 24735)	Indicates whether the associated component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.

177.1.7.4.1 TWebContainedComponentHelper.GetContainer Method

Returns the value of the WebParent (see page 24737) property.

Pascal

```
function GetContainer: TWebComponentContainer;
```

C++

```
__fastcall TWebComponentContainer GetContainer();
```

Description

Use GetContainer to access the component that uses this helper object's associated component to assemble its HTML output. The component that this method returns uses the list specified by ContainerList (see page 24737) to manage its list of Web items.

See Also

WebParent (see page 24737)

ContainerList (see page 24737)

177.1.7.4.2 TWebContainedComponentHelper.GetIndex Method

Returns the position of the associated Web item in the parent item's array of Web items.

Pascal

```
function GetIndex: Integer;
```

C++

```
__fastcall int GetIndex();
```

Description

GetIndex returns the index of the associated component in the list that is the value of the ContainerList (see page 24737) property. The index of the first item is 0, the index of the second item is 1, and so on.

See Also

ContainerList (see page 24737)

177.1.7.4.3 TWebContainedComponentHelper.IsDefaultField Method

Indicates whether the associated component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.

Pascal

```
function IsDefaultField: Boolean;
```

C++

```
__fastcall Boolean IsDefaultField();
```

Description

TWebContainedComponentHelper (see page 24731) uses IsDefaultField internally to locate the property on the Web parent that lists the associated item along with other Web items.

IsDefaultField returns true if the TWebComponentList (see page 24708) that contains the associated item represents a set of Web items that are used when the application does not explicitly assign Web items to the Web parent. IsDefaultField returns false if the associated component does not belong to such a default list.

See Also

ContainerList (see page 24737)

WebParent (see page 24737)

TWebComponentList (see page 24708)

177.1.7.4.4 TWebContainedComponentHelper.ParentChanged Method

Generates an OnParentChanged (see page 24738) event.

Pascal

```
procedure ParentChanged; virtual;
```

C++

```
virtual __fastcall ParentChanged();
```

Description

ParentChanged is called automatically when the WebParent (see page 24737) property changes. It generates an OnParentChanged (see page 24738) event, which is handled by the event handler that the associated component passes to the TWebContainedComponentHelper (see page 24731) constructor.

See Also

OnParentChanged (see page 24738)

Create (see page 24733)

177.1.7.4.5 TWebContainedComponentHelper.SetComponentList Method

Sets the value of the ContainerList (see page 24737) property.

Pascal

```
procedure SetComponentList(List: TObject);
```

C++

```
__fastcall SetComponentList(TObject * List);
```

Description

SetComponentList assigns the object specified by List as the value of the ContainerList (see page 24737) property. Note that List must be an instance of TWebComponentList (see page 24708) (or nil (Delphi) or NULL (C++)).

See Also

ContainerList (see page 24737)

177.1.7.4.6 TWebContainedComponentHelper.SetContainer Method

Sets the value of the WebParent (see page 24737) property.

Pascal

```
procedure SetContainer(Container: TWebComponentContainer);
```

C++

```
__fastcall SetContainer(TWebComponentContainer Container);
```

Description

SetContainer assigns a new value to the WebParent (see page 24737) property.

Container is the new Web parent.

See Also

WebParent (see page 24737)

177.1.7.4.7 TWebContainedComponentHelper.SetIndex Method

Sets the position of the associated component in the list specified by ContainerList (see page 24737).

Pascal

```
procedure SetIndex(Value: Integer);
```

C++

```
__fastcall SetIndex(int Value);
```

Description

SetIndex moves the associated component to a specified position in the list of Web items that contains it.

Value is the new index of the associated component, where 0 specifies the first position, 1 specifies the second position, and so on.

See Also

ContainerList (see page 24737)

177.1.7.4.8 TWebContainedComponentHelper.SetWebParent Method

Sets the value of the WebParent (see page 24737) property.

Pascal

```
procedure SetWebParent(Value: TComponent);
```

C++

```
__fastcall SetWebParent(TComponent * Value);
```

Description

SetWebParent moves the associated component to the component list of a new parent. To do so, it removes the associated component from the current parent's list of Web items, adds it to the new Web parent's list of Web items, and then generates an OnParentChanged (see page 24738) event.

Value is the new index of the associated component, where 0 specifies the first position, 1 specifies the second position, and so on.

Value is the new Web parent.

See Also

WebParent (see page 24737)

OnParentChanged (see page 24738)

ContainerList (see page 24737)

177.1.7.5 TWebContainedComponentHelper Properties

The properties of the TWebContainedComponentHelper class are listed here.

Public Properties

	Name	Description
R	ContainerList (see page 24737)	Indicates the list that contains the associated component.
R	WebParent (see page 24737)	Indicates the component that uses the list of Web items that includes the associated component.

177.1.7.5.1 TWebContainedComponentHelper.ContainerList Property

Indicates the list that contains the associated component.

Pascal

```
property ContainerList: TWebComponentList;
```

C++

```
__property TWebComponentList ContainerList;
```

Description

Use ContainerList to access the list in which the associated component appears. The parent item uses the components in the list to generate HTML, which it then assembles into its own HTML output.

See Also

WebParent (see page 24737)

177.1.7.5.2 TWebContainedComponentHelper.WebParent Property

Indicates the component that uses the list of Web items that includes the associated component.

Pascal

```
property WebParent: TComponent;
```

C++

```
__property TComponent * WebParent;
```

Description

WebParent specifies the component on whose behalf the associated component generates HTML. WebParent is the component that uses the list that is the value of the ContainerList (see page 24737) property.

See Also

ContainerList (see page 24737)

177.1.7.6 TWebContainedComponentHelper Events

The events of the TWebContainedComponentHelper class are listed here.

Protected Events

	Name	Description
	OnParentChanged (see page 24738)	Occurs when the WebParent (see page 24737) property changes.

177.1.7.6.1 TWebContainedComponentHelper.OnParentChanged Event

Occurs when the WebParent (see page 24737) property changes.

Pascal

```
property OnParentChanged: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnParentChanged;
```

Description

Components that use TWebContainedComponentHelper (see page 24731) so that they can act as a child Web item in a hierarchy of Web items assign an event handler to OnParentChanged.

The associated component uses TWebContainedComponentHelper (see page 24731) to implement the IWebComponent (see page 24782) interface. This interface can be used to change the value of WebParent (see page 24737). By assigning an OnParentChanged event handler, the associated component can update itself when it moves from one parent item to another.

See Also

WebParent (see page 24737)

IWebComponent (see page 24782)

177.1.8 TWebContainedContainerComponent Class

TWebContainedContainerComponent is the base class for Web items that generate HTML on behalf of another component and also use Web items to generate HTML.

Class Hierarchy

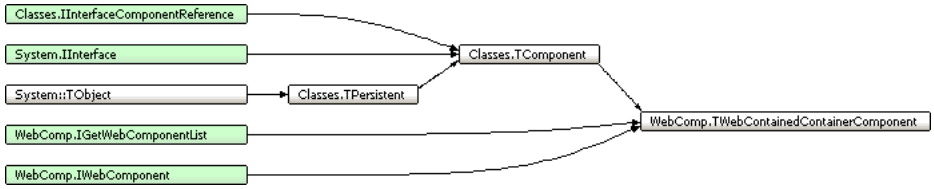

Pascal

```
TWebContainedContainerComponent = class(TComponent, IGetWebComponentList, IWebComponent);
```

C++

```
class TWebContainedContainerComponent : public TComponent, public IGetWebComponentList, public
IWebComponent;
```

File

WebComp

Description

Use TWebContainedContainerComponent as a base class for Web items that have their own subitems. Web items are items that generate HTML statements on behalf of another component. The component that uses the generated HTML has a property whose type is a TWebComponentList (see page 24708) object that lists a set of its Web items. In addition, the TWebContainedContainerComponent descendant uses a TWebComponentList (see page 24708) object to store a set of Web items that it uses.

TWebContainedContainerComponent implements the logic that fits a component into the hierarchy of Web items. TWebContainedContainerComponent descendants act as internal nodes in this hierarchy. That is, they have both a Web parent and a set of child Web items.

In order to allow the parent's TWebComponentList (see page 24708) object to manage this Web item, TWebContainedContainerComponent implements the IWebComponent (see page 24782) interface.

Do not create instances of TWebContainedContainerComponent. TWebContainedContainerComponent does not produce any HTML, but merely handles the relationships to parent and child TWebComponentList (see page 24708) objects.

See Also

TWebComponentList (see page 24708)

IWebComponent (see page 24782)

177.1.8.1 TWebContainedContainerComponent Members

The following tables list the members exposed by TWebContainedContainerComponent.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.

	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 24742)	Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

177.1.8.2 TWebContainedContainerComponent.Create Constructor

Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebContainedContainerComponent(TComponent * AOwner);
```

Description

Call Create to instantiate a contained component at runtime. Web items added at design time using the Web Page editor are instantiated automatically.

AOwner is the component, typically a Web module, that becomes the value of the Owner property. The Owner is responsible for initiating the loading and saving of this component, and for freeing the TWebContainedContainerComponent (see page 24738)

instance.

See Also

Owner (see page 2807)

177.1.8.3 TWebContainedContainerComponent.Destroy Destructor

Frees an instance of TWebContainedContainerComponent (see page 24738).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebContainedContainerComponent();
```

Description

Destroy removes this component from its parent's list of components that generate HTML before freeing the memory used by TWebContainedContainerComponent (see page 24738).

177.1.8.4 TWebContainedContainerComponent Methods

The methods of the TWebContainedContainerComponent class are listed here.

Protected Methods

	Name	Description
	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.

Public Methods

	Name	Description
	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.

177.1.8.4.1 TWebContainedContainerComponent.GetChildren Method

Executes a callback for each child Web item.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

TWebContainedContainerComponent (see page 24738) overrides the GetChildren method that is introduced in TComponent for the streaming system that loads and saves components. It executes the Proc procedure for each child that in the list that GetWebComponents (see page 24745) returns, as long as that component has Root for an Owner. This allows the sub Web items to be loaded and saved with the TWebContainedContainerComponent (see page 24738) object.

See Also

GetWebComponents (see page 24745)

GetParentComponent

SetChildOrder (see page 24748)

177.1.8.4.2 TWebContainedContainerComponent.GetContainerList Method

Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.

Pascal

```
function GetContainerList: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetContainerList();
```

Description

Applications can't access the protected GetContainerList method. It is used internally to access the object manages this item along with the other Web items of the Web parent.

See Also

IsDefaultField (see page 24746)

WebParent (see page 24750)

177.1.8.4.3 TWebContainedContainerComponent.GetDefaultWebComponents Method

Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function GetDefaultWebComponents: TWebComponentList; virtual;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents supplies the default set of child items for this Web item. As implemented in TWebContainedContainerComponent (see page 24738), GetDefaultWebComponents always returns nil (Delphi) or NULL (C++). Descendant classes override this method to supply a set of default items.

See Also

GetWebComponents (see page 24745)

IsDefaultField (see page 24746)

177.1.8.4.4 TWebContainedContainerComponent.GetParentComponent Method

Returns the object that uses this component to generate HTML.

Pascal

```
function GetParentComponent: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetParentComponent();
```

Description

TWebContainedContainerComponent (see page 24738) overrides the GetParentComponent method that is introduced in TComponent for the streaming system that loads and saves components. It returns the value of the WebParent (see page 24750) property (the object that uses the HTML generated by this component when assembling its own portion of an HTML document.)

See Also

GetParentComponent

HasParent (see page 24745)

WebParent (see page 24750)

177.1.8.4.5 TWebContainedContainerComponent.GetWebComponents Method

Returns the list of Web items that this item uses to generate its HTML.

Pascal

```
function GetWebComponents: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetWebComponents();
```

Description

GetWebComponents returns the TWebComponentList (see page 24708) object that manages all of the child items of this component.

See Also

GetDefaultWebComponents (see page 24744)

177.1.8.4.6 TWebContainedContainerComponent.HasParent Method

Indicates whether this component appears in another component's list of Web items.

Pascal

```
function HasParent: Boolean; override;
```

C++

```
virtual __fastcall Boolean HasParent();
```

Description

TWebContainedContainerComponent (see page 24738) overrides the HasParent method that is introduced in TComponent for the streaming system that loads and saves components. It returns true if the WebParent (see page 24750) property is not nil (Delphi) or NULL (C++).

If the contained component has a parent, that parent component is responsible for writing this component to a stream.

See Also

HasParent

GetParentComponent (see page 24745)

WebParent (see page 24750)

177.1.8.4.7 TWebContainedContainerComponent.ImplGetDefaultComponentList Method

Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function ImplGetDefaultComponentList: TObject; virtual;
```

C++

```
virtual __fastcall TObject * ImplGetDefaultComponentList();
```

Description

ImplGetDefaultComponentList supplies the default set of child items for this Web item. ImplGetDefaultComponentList calls the virtual GetDefaultWebComponents (see page 24744) method, which descendant classes override to return this list.

See Also

GetDefaultWebComponents (see page 24744)

IsDefaultField (see page 24746)

177.1.8.4.8 TWebContainedContainerComponent.IsDefaultField Method

Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.

Pascal

```
function IsDefaultField: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsDefaultField();
```

Description

TWebContainedComponent (see page 24722) uses IsDefaultField internally to locate the property on the Web parent that lists this item along with other Web items.

IsDefaultField returns true if the TWebComponentList (see page 24708) that contains this item represents a set of Web items that are used when the application does not explicitly assign Web items to the Web parent. IsDefaultField returns false if this component does not belong to such a default list.

See Also

GetContainerList (see page 24744)

WebParent (see page 24750)

TWebComponentList (see page 24708)

GetDefaultWebComponents (see page 24744)

GetWebComponents (see page 24745)

177.1.8.4.9 TWebContainedContainerComponent.ParentChanged Method

Responds when the WebParent (see page 24750) property changes.

Pascal

```
procedure ParentChanged; virtual;
```

C++

```
virtual __fastcall ParentChanged();
```

Description

The protected ParentChanged method is called automatically when the WebParent (see page 24750) property changes. As implemented in TWebContainedContainerComponent (see page 24738), ParentChanged does nothing. Descendant classes can override this method to perform any necessary updates in response to the change in Web parent.

See Also

WebParent (see page 24750)

177.1.8.4.10 TWebContainedContainerComponent.ReadState Method

Reads the component's published properties, stored data, and owned components from a reader object.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

ReadState is part of a sequence of calls used by the streaming system that loads and saves components. It reads the values of all the component's published properties, stored data, and owned components from the reader object passed in Reader. Do not call ReadState directly.

TWebContainedContainerComponent (see page 24738) overrides ReadState to obtain the parent component from the reader, which it assigns as the value of the WebParent (see page 24750) property.

Note: Any descendant classes overriding ReadState should end with a call to the inherited ReadState method.

See Also

GetParentComponent (see page 24745)

WebParent (see page 24750)

177.1.8.4.11 TWebContainedContainerComponent.SetChildOrder Method

Positions a child component in the list of child Web items.

Pascal

```
procedure SetChildOrder(Component: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Component, int Order);
```

Description

SetChildOrder is part of the streaming system that loads and saves components. It positions a newly loaded child component in the list that GetWebComponents (see page 24745) returns.

Child is the newly loaded child component.

Order indicates its position in the list of child Web items, where 0 is the first position, 1 is the second position, and so on.

See Also

GetWebComponents (see page 24745)

SetChildOrder

177.1.8.4.12 TWebContainedContainerComponent.SetDefaultWebComponents Method

Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList); virtual;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

SetDefaultWebComponents assigns the default set of child items for this Web item. This default list can be retrieved by calling the GetDefaultWebComponents (see page 24744) method.

As implemented in TWebContainedContainerComponent (see page 24738), SetDefaultWebComponents does nothing. Descendant classes override this method to store the list of default items in a class-specific manner.

See Also

GetDefaultWebComponents (see page 24744)

IsDefaultField (see page 24746)

177.1.8.4.13 TWebContainedContainerComponent.SetParentComponent Method

Sets the WebParent (see page 24750) property to be the same as this component's parent.

Pascal

```
procedure SetParentComponent(AParent: TComponent); override;
```

C++

```
virtual __fastcall SetParentComponent(TComponent * AParent);
```

Description

TWebContainedContainerComponent (see page 24738) overrides the SetParentComponent method that is introduced in TComponent for the streaming system that loads and saves components. It sets the WebParent (see page 24750) property to AParent, so that it is the same as the parent that loads and saves this component.

See Also

SetParentComponent

HasParent (see page 24745)

WebParent (see page 24750)

177.1.8.4.14 TWebContainedContainerComponent.SetWebParent Method

Sets the value of the WebParent (see page 24750) property.

Pascal

```
procedure SetWebParent(Value: TComponent);
```

C++

```
__fastcall SetWebParent(TComponent * Value);
```

Description

SetWebParent is used internally to set the read-only WebParent (see page 24750) property. To assign a Web parent in an application, add this component to the TWebComponentList (see page 24708) object that implements a property on the new Web parent.

See Also

WebParent (see page 24750)

Add (see page 24713)

177.1.8.5 TWebContainedContainerComponent Properties

The properties of the TWebContainedContainerComponent class are listed here.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

177.1.8.5.1 TWebContainedContainerComponent.WebContainedComponentHelper Property

Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained

component.

Pascal

```
property WebContainedComponentHelper: TWebContainedComponentHelper;
```

C++

```
__property TWebContainedComponentHelper WebContainedComponentHelper;
```

Description

TWebContainedContainerComponent (see page 24738) uses WebContainedComponentHelper internally to access the separate helper object that implements the IWebComponent (see page 24782) interface. In addition to providing the implementation for the IWebComponent (see page 24782) interface, WebContainedComponentHelper provides the underlying implementation for other properties and methods that are common to components that appear in a list of Web items.

See Also

IWebComponent (see page 24782)

177.1.8.5.2 TWebContainedContainerComponent.WebContainerComponentHelper Property

Provides access to the helper object that allows this object to manage sub Web items.

Pascal

```
property WebContainerComponentHelper: TWebContainerComponentHelper;
```

C++

```
__property TWebContainerComponentHelper WebContainerComponentHelper;
```

Description

TWebContainedContainerComponent (see page 24738) uses WebContainerComponentHelper internally to handle two lists of child Web items:

One list contains the default Web items that this component uses if the application does not explicitly assign sub Web items. The GetDefaultWebComponents (see page 24744) method returns this list.

The other list contains explicitly assigned Web items that this component uses. The GetWebComponents (see page 24745) method returns this list.

See Also

GetDefaultWebComponents (see page 24744)

GetWebComponents (see page 24745)

177.1.8.5.3 TWebContainedContainerComponent.WebParent Property

Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Pascal

```
property WebParent: TComponent;
```

C++

```
__property TComponent * WebParent;
```

Description

TWebContainedContainerComponent (see page 24738) uses WebParent to access the component on whose behalf it generates HTML. WebParent uses a TWebComponentList (see page 24708) object that includes this TWebContainedContainerComponent (see page 24738) object to represent its child Web items.

See Also

TWebComponentList (see page 24708)

177.1.9 TWebContainerComponent Class

TWebContainerComponent is the base class for components that use Web items to generate HTML.

Class Hierarchy

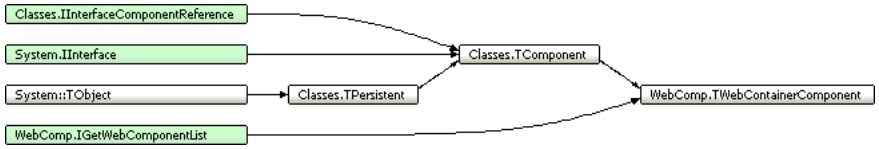

Pascal

```
TWebContainerComponent = class(TComponent, IGetWebComponentList);
```

C++

```
class TWebContainerComponent : public TComponent, public IGetWebComponentList;
```

File

WebComp

Description

Use TWebContainerComponent as a base class for objects that maintain a collection of Web items but that do not have a parent Web item. TWebContainerComponent uses a TWebComponentList (see page 24708) object to store the set of child items that it manages.

Do not create instances of TWebContainerComponent, it is intended only as a base class. It only provides support for storing and retrieving child Web items.

See Also

TWebComponentList (see page 24708)

IWebComponent (see page 24782)

TWebContainedContainerComponent (see page 24738)

177.1.9.1 TWebContainerComponent Members

The following tables list the members exposed by TWebContainerComponent.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
V	GetChildren (see page 24755)	Executes a callback for each child Web item.
V	GetDefaultWebComponents (see page 24756)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebComponents (see page 24756)	Returns the list of Web items that this item manages.
V	ImplGetDefaultComponentList (see page 24756)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetChildOrder (see page 24757)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 24757)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainerComponentHelper (see page 24758)	Provides access to the helper object that allows this object to manage its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 24754)	Creates and initializes an instance of TWebContainerComponent (see page 24751).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 24755)	Frees an instance of TWebContainerComponent (see page 24751).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

177.1.9.2 TWebContainerComponent.Create Constructor

Creates and initializes an instance of TWebContainerComponent (see page 24751).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebContainerComponent(TComponent * AOwner);
```

Description

Call Create to instantiate a contained component at runtime.

AOwner is the component, typically a Web module, that becomes the value of the Owner property. The Owner is responsible for initiating the loading and saving of this component, and for freeing the TWebContainerComponent (see page 24751) instance.

See Also

Owner (see page 2807)

177.1.9.3 TWebContainerComponent.Destroy Destructor

Frees an instance of TWebContainerComponent (see page 24751).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebContainerComponent();
```

Description

Destroy frees the memory used by TWebContainerComponent (see page 24751).

177.1.9.4 TWebContainerComponent Methods

The methods of the TWebContainerComponent class are listed here.

Protected Methods

	Name	Description
	GetChildren (see page 24755)	Executes a callback for each child Web item.
	GetDefaultWebComponents (see page 24756)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetWebComponents (see page 24756)	Returns the list of Web items that this item manages.
	ImplGetDefaultComponentList (see page 24756)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetChildOrder (see page 24757)	Positions a child component in the list of child Web items.
	SetDefaultWebComponents (see page 24757)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

177.1.9.4.1 TWebContainerComponent.GetChildren Method

Executes a callback for each child Web item.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

TWebContainerComponent (see page 24751) overrides the GetChildren method that is introduced in TComponent for the streaming system that loads and saves components. It executes the Proc procedure for each child that in the list that GetWebComponents (see page 24756) returns, as long as that component has Root for an Owner. This allows the child Web items to be loaded and saved with the TWebContainerComponent (see page 24751) object.

See Also

GetWebComponents (see page 24756)

GetParentComponent

SetChildOrder (see page 24757)

177.1.9.4.2 TWebContainerComponent.GetDefaultWebComponents Method

Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function GetDefaultWebComponents: TWebComponentList; virtual;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents supplies the default set of child items for this component. As implemented in TWebContainerComponent (see page 24751), GetDefaultWebComponents always returns nil (Delphi) or NULL (C++). Descendant classes override this method to supply a set of default items.

See Also

GetWebComponents (see page 24756)

IsDefaultField (see page 24728)

177.1.9.4.3 TWebContainerComponent.GetWebComponents Method

Returns the list of Web items that this item manages.

Pascal

```
function GetWebComponents: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetWebComponents();
```

Description

GetWebComponents returns the TWebComponentList (see page 24708) object that manages all of the child items of this component.

See Also

GetDefaultWebComponents (see page 24756)

177.1.9.4.4 TWebContainerComponent.ImplGetDefaultComponentList Method

Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function ImplGetDefaultComponentList: TObject; virtual;
```

C++

```
virtual __fastcall TObject * ImplGetDefaultComponentList();
```

Description

ImplGetDefaultComponentList supplies the default set of child items for this Web item. ImplGetDefaultComponentList calls the virtual GetDefaultWebComponents (see page 24756) method, which descendant classes override to return this list.

See Also

GetDefaultWebComponents (see page 24756)

IsDefaultField (see page 24728)

177.1.9.4.5 TWebContainerComponent.SetChildOrder Method

Positions a child component in the list of child Web items.

Pascal

```
procedure SetChildOrder(Component: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Component, int Order);
```

Description

SetChildOrder is part of the streaming system that loads and saves components. It positions a newly loaded child component in the list that GetWebComponents (see page 24756) returns.

Child is the newly loaded child component.

Order indicates its position in the list of child Web items, where 0 is the first position, 1 is the second position, and so on.

See Also

GetWebComponents (see page 24756)

SetChildOrder

177.1.9.4.6 TWebContainerComponent.SetDefaultWebComponents Method

Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList); virtual;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

SetDefaultWebComponents assigns the default set of child items for this Web item. This default list can be retrieved by calling the GetDefaultWebComponents (see page 24756) method.

As implemented in TWebContainerComponent (see page 24751), SetDefaultWebComponents does nothing. Descendant classes override this method to store the list of default items in a class-specific manner.

See Also

GetDefaultWebComponents (see page 24756)

IsDefaultField (see page 24728)

177.1.9.5 TWebContainerComponent Properties

The properties of the TWebContainerComponent class are listed here.

Protected Properties

	Name	Description
R	WebContainerComponentHelper (see page 24758)	Provides access to the helper object that allows this object to manage its Web items.

177.1.9.5.1 TWebContainerComponent.WebContainerComponentHelper Property

Provides access to the helper object that allows this object to manage its Web items.

Pascal

```
property WebContainerComponentHelper: TWebContainerComponentHelper;
```

C++

```
__property TWebContainerComponentHelper WebContainerComponentHelper;
```

Description

TWebContainerComponent (see page 24751) uses WebContainerComponentHelper internally to handle two lists of child Web items:

One list contains the default Web items that this component uses if the application does not explicitly assign child Web items. The GetDefaultWebComponents (see page 24756) method returns this list.

The other list contains explicitly assigned Web items that this component uses. The GetWebComponents (see page 24756) method returns this list.

See Also

GetDefaultWebComponents (see page 24756)

GetWebComponents (see page 24756)

177.1.10 TWebContainerComponentHelper Class

TWebContainerComponentHelper implements the common behavior of components that maintain a list of Web items to generate HTML.

Class Hierarchy

Pascal

```
TWebContainerComponentHelper = class(TObject);
```

C++

```
class TWebContainerComponentHelper : public TObject;
```

File

WebComp

Description

Components that maintain a list of Web items to help them generate HTML use TWebContainerComponentHelper to provide the common implementation of properties and methods for managing the list of sub components. This includes

The logic for handling two lists of Web items: one that represents the Web items that the application assigns, and one that represents a set of default items that can be used when the application does not explicitly assign Web items.

The logic for interacting with the component streaming system, so that child Web items are correctly loaded and saved with the component that uses TWebContainerComponentHelper.

See Also

TWebContainedContainerComponent (see page 24738)

TWebContainerComponent (see page 24751)

TWebComponentList (see page 24708)

177.1.10.1 TWebContainerComponentHelper Members

The following tables list the members exposed by TWebContainerComponentHelper.

Protected Events

	Name	Description
	OnGetDefaultComponentList (see page 24765)	Occurs when TWebContainerComponentHelper (see page 24758) needs to access the value of DefaultWebComponents (see page 24764).
	OnSetDefaultComponentList (see page 24765)	Occurs when TWebContainerComponentHelper (see page 24758) needs to assign the value of DefaultWebComponents (see page 24764).

Protected Methods

	Name	Description
	GetComponentList (see page 24762)	Returns the value of the WebComponents property.
	GetDefaultComponentList (see page 24762)	Returns the list of Web items that is used if an application does not explicitly assign any Web items to the associated component.
	GetDefaultWebComponents (see page 24762)	Generates an OnGetDefaultComponentList (see page 24765) event.
	SetDefaultWebComponents (see page 24763)	Generates an OnSetDefaultComponentList (see page 24765) event.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 24760)	Creates an instance of TWebContainerComponentHelper (see page 24758).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 24760)	Destroys an instance of TWebContainerComponentHelper (see page 24758).
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetChildren (see page 24761)	Executes a callback for each child Web item.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetChildOrder (see page 24763)	Positions a child component in the list of child Web items.

Public Properties

	Name	Description
	DefaultWebComponents (see page 24764)	Lists the Web items that the associated component uses by default if the application does not explicitly assign any Web items.
R	Webcomponents (see page 24764)	This is Webcomponents, a member of class TWebContainerComponentHelper.

177.1.10.2 TWebContainerComponentHelper.Create Constructor

Creates an instance of TWebContainerComponentHelper (see page 24758).

Pascal

```
constructor Create(AComponent: TComponent; AOnGetDefaultComponentList: TGetComponentListEvent;
AOnSetDefaultComponentList: TSetComponentListEvent);
```

C++

```
__fastcall TWebContainerComponentHelper(TComponent * AComponent, TGetComponentListEvent
AOnGetDefaultComponentList, TSetComponentListEvent AOnSetDefaultComponentList);
```

Description

Applications do not create instances of TWebContainerComponentHelper (see page 24758). The component that uses TWebContainerComponentHelper (see page 24758) so that it can manage a set of Web items instantiates TWebContainerComponentHelper (see page 24758).

AComponent is the component that uses TWebContainerComponentHelper (see page 24758) to manage its list of Web items and to interact with the component streaming system.

AOnGetDefaultComponentList is an event handler that AComponent supplies to provide the read implementation of the DefaultWebComponents (see page 24764) property. It is assigned as the event handler for OnGetDefaultComponentList (see page 24765).

AOnSetDefaultComponentList is an event handler that AComponent supplies to provide the write implementation of the DefaultWebComponents (see page 24764) property. It is assigned as the event handler for OnSetDefaultComponentList (see page 24765).

See Also

OnGetDefaultComponentList (see page 24765)

OnSetDefaultComponentList (see page 24765)

DefaultWebComponents (see page 24764)

177.1.10.3 TWebContainerComponentHelper.Destroy Destructor

Destroys an instance of TWebContainerComponentHelper (see page 24758).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebContainerComponentHelper();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TWebContainerComponentHelper (see page 24758) object is not nil and only then calls Destroy.

See Also

Free (see page 22300)

177.1.10.4 TWebContainerComponentHelper Methods

The methods of the TWebContainerComponentHelper class are listed here.

Protected Methods

	Name	Description
	GetComponentList (see page 24762)	Returns the value of the WebComponents property.
	GetDefaultComponentList (see page 24762)	Returns the list of Web items that is used if an application does not explicitly assign any Web items to the associated component.
	GetDefaultWebComponents (see page 24762)	Generates an OnGetDefaultComponentList (see page 24765) event.
	SetDefaultWebComponents (see page 24763)	Generates an OnSetDefaultComponentList (see page 24765) event.

Public Methods

	Name	Description
	GetChildren (see page 24761)	Executes a callback for each child Web item.
	SetChildOrder (see page 24763)	Positions a child component in the list of child Web items.

177.1.10.4.1 TWebContainerComponentHelper.GetChildren Method

Executes a callback for each child Web item.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent);
```

C++

```
__fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren provides the underlying implementation of the associated component's GetChildren method. GetChildren is introduced in TComponent for the streaming system that loads and saves components.

GetChildren executes the Proc procedure for each child that in the list specified by the WebComponents property, as long as that component has Root for an Owner. This allows the sub Web items to be loaded and saved with the component that uses TWebContainerComponentHelper (see page 24758).

See Also

WebComponents (see page 24764)

SetChildOrder (see page 24763)

GetChildren

177.1.10.4.2 TWebContainerComponentHelper.GetComponentList Method

Returns the value of the WebComponents property.

Pascal

```
function GetComponentList: TObject;
```

C++

```
__fastcall TObject * GetComponentList();
```

Description

GetComponentList returns the list of Web items that have been assigned to the associated component. The return value is an instance of TWebComponentList (see page 24708).

See Also

WebComponents (see page 24764)

GetDefaultComponentList (see page 24762)

TWebComponentList (see page 24708)

177.1.10.4.3 TWebContainerComponentHelper.GetDefaultComponentList Method

Returns the list of Web items that is used if an application does not explicitly assign any Web items to the associated component.

Pascal

```
function GetDefaultComponentList: TObject;
```

C++

```
__fastcall TObject * GetDefaultComponentList();
```

Description

GetComponentList (see page 24762) returns the value of the DefaultWebComponents (see page 24764) property. The return value is an instance of TWebComponentList (see page 24708).

See Also

DefaultWebComponents (see page 24764)

WebComponents (see page 24764)

TWebComponentList (see page 24708)

177.1.10.4.4 TWebContainerComponentHelper.GetDefaultWebComponents Method

Generates an OnGetDefaultComponentList (see page 24765) event.

Pascal

```
function GetDefaultWebComponents: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents is the read implementation of the DefaultWebComponents (see page 24764) property. It generates an OnGetDefaultComponentList (see page 24765) event, so that the associated component can supply this list of default items. If there is no OnGetDefaultComponentList (see page 24765) event handler, GetDefaultWebComponents returns nil (Delphi) or NULL (C++).

See Also

OnGetDefaultComponentList (see page 24765)

DefaultWebComponents (see page 24764)

177.1.10.4.5 TWebContainerComponentHelper.SetChildOrder Method

Positions a child component in the list of child Web items.

Pascal

```
procedure SetChildOrder(Component: TComponent; Order: Integer);
```

C++

```
__fastcall SetChildOrder(TComponent * Component, int Order);
```

Description

SetChildOrder provides the underlying implementation of the associated component's SetChildOrder method. SetChildOrder is introduced in TComponent for the streaming system that loads and saves components. It positions a newly loaded child component in the list specified by WebComponents.

Child is the newly loaded child component.

Order indicates its position in WebComponents, where 0 is the first position, 1 is the second position, and so on.

See Also

WebComponents (see page 24764)

SetChildOrder

GetChildren (see page 24761)

177.1.10.4.6 TWebContainerComponentHelper.SetDefaultWebComponents Method

Generates an OnSetDefaultComponentList (see page 24765) event.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList);
```

C++

```
__fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

GetDefaultWebComponents (see page 24762) is the write implementation of the DefaultWebComponents (see page 24764) property. It generates an OnSetDefaultComponentList (see page 24765) event, so that the associated component can assign the new list of default items.

See Also

Create (see page 24760)

OnSetDefaultComponentList (see page 24765)

DefaultWebComponents (see page 24764)

177.1.10.5 TWebContainerComponentHelper Properties

The properties of the TWebContainerComponentHelper class are listed here.

Public Properties

	Name	Description
	DefaultWebComponents (see page 24764)	Lists the Web items that the associated component uses by default if the application does not explicitly assign any Web items.
R	Webcomponents (see page 24764)	This is Webcomponents, a member of class TWebContainerComponentHelper.

177.1.10.5.1 TWebContainerComponentHelper.DefaultWebComponents Property

Lists the Web items that the associated component uses by default if the application does not explicitly assign any Web items.

Pascal

```
property DefaultWebComponents: TWebComponentList;
```

C++

```
__property TWebComponentList DefaultWebComponents;
```

Description

Use DefaultWebComponents to access the list of default Web items that can be used when no items have been explicitly assigned.

TWebContainerComponentHelper (see page 24758) does not actually store this list. It leaves that task to the component that uses TWebContainerComponentHelper (see page 24758). Instead, TWebContainerComponentHelper (see page 24758) uses two events, OnGetDefaultComponentList (see page 24765) and OnSetDefaultComponentList (see page 24765) for reading and writing this list. The associated component that uses TWebContainerComponentHelper (see page 24758) assigns event handlers to these two events, thereby providing the implementation for this property.

See Also

WebComponents (see page 24764)

OnGetDefaultComponentList (see page 24765)

OnGetDefaultComponentList (see page 24765)

177.1.10.5.2 TWebContainerComponentHelper.Webcomponents Property

Pascal

```
property Webcomponents: TWebComponentList;
```

C++

```
__property TWebComponentList Webcomponents;
```

Description

This is Webcomponents, a member of class TWebContainerComponentHelper.

177.1.10.6 TWebContainerComponentHelper Events

The events of the TWebContainerComponentHelper class are listed here.

Protected Events

	Name	Description
	OnGetDefaultComponentList (see page 24765)	Occurs when TWebContainerComponentHelper (see page 24758) needs to access the value of DefaultWebComponents (see page 24764).
	OnSetDefaultComponentList (see page 24765)	Occurs when TWebContainerComponentHelper (see page 24758) needs to assign the value of DefaultWebComponents (see page 24764).

177.1.10.6.1 TWebContainerComponentHelper.OnGetDefaultComponentList Event

Occurs when TWebContainerComponentHelper (see page 24758) needs to access the value of DefaultWebComponents (see page 24764).

Pascal

```
property OnGetDefaultComponentList: TGetComponentListEvent;
```

C++

```
__property TGetComponentListEvent OnGetDefaultComponentList;
```

Description

The component that uses TWebContainerComponentHelper (see page 24758) assigns an event handler for OnGetDefaultComponentList when it creates the TWebContainerComponentHelper (see page 24758) instance. This event handler provides the underlying read implementation of the DefaultWebComponents (see page 24764) property.

If the component that uses TWebContainerComponentHelper (see page 24758) does not provide an OnGetDefaultComponentList event handler, then the value of the DefaultWebComponents (see page 24764) property is nil (Delphi) or NULL (C++).

See Also

OnSetDefaultComponentList (see page 24765)

DefaultWebComponents (see page 24764)

Create (see page 24760)

177.1.10.6.2 TWebContainerComponentHelper.OnSetDefaultComponentList Event

Occurs when TWebContainerComponentHelper (see page 24758) needs to assign the value of DefaultWebComponents (see page 24764).

Pascal

```
property OnSetDefaultComponentList: TSetComponentListEvent;
```

C++

```
__property TSetComponentListEvent OnSetDefaultComponentList;
```

Description

The component that uses TWebContainerComponentHelper (see page 24758) assigns an event handler for OnSetDefaultComponentList when it creates the TWebContainerComponentHelper (see page 24758) instance. This event handler provides the underlying write implementation of the DefaultWebComponents (see page 24764) property.

If the component that uses TWebContainerComponentHelper (see page 24758) does not provide an OnSetDefaultComponentList event handler, then the value of the DefaultWebComponents (see page 24764) can't be changed.

See Also

OnGetDefaultComponentList (see page 24765)

DefaultWebComponents (see page 24764)

Create (see page 24760)

177.1.11 TWebContentOptions Class

TWebContentOptions controls what a Web item can include in its generated HTML.

Class Hierarchy

Pascal

```
TWebContentOptions = class(TObject);
```

C++

```
class TWebContentOptions : public TObject;
```

File

WebComp

Description

Use TWebContentOptions to influence the HTML generated by a component that supports the IWebContent (see page 24792) interface. Each TWebContentOptions represents a set of flags that are passed to the constructor.

See Also

IWebContent (see page 24792)

177.1.11.1 TWebContentOptions Members

The following tables list the members exposed by TWebContentOptions.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 24767)	Creates a new TWebContentOptions (see page 24766) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Flags (see page 24768)	Lists the flags that indicate what type of generated HTML is permitted.

177.1.11.2 TWebContentOptions.Create Constructor

Creates a new TWebContentOptions (see page 24766) object.

Pascal

```
constructor Create(AFlags: TWebContentFlags);
```

C++

```
__fastcall TWebContentOptions(TWebContentFlags AFlags);
```

Description

Call Create to create a TWebContentOptions (see page 24766) instance at runtime. Pass the flags that the object should represent as the value of the AFlags parameter. This value initializes the Flags (see page 24768) property.

See Also

Flags (see page 24768)

177.1.11.3 TWebContentOptions Properties

The properties of the TWebContentOptions class are listed here.

Public Properties

	Name	Description
R	Flags (see page 24768)	Lists the flags that indicate what type of generated HTML is permitted.

177.1.11.3.1 TWebContentOptions.Flags Property

Lists the flags that indicate what type of generated HTML is permitted.

Pascal

```
property Flags: TWebContentFlags;
```

C++

```
__property TWebContentFlags Flags;
```

Description

Read Flags to determine what types of HTML a Web item can generate.

See Also

Create (see page 24767)

177.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAddFieldsEditor (see page 24769)	This is class WebComp.IAddFieldsEditor.
	IGetWebComponentList (see page 24771)	This is class WebComp.IGetWebComponentList.
	IGetWebComponentNodeName (see page 24772)	This is class WebComp.IGetWebComponentNodeName.
	IGetXMLStream (see page 24773)	This is class WebComp.IGetXMLStream.
	ILayoutWebContent (see page 24774)	ILayoutWebContent lets child Web items adhere to layout rules imposed by their parent.
	INotifyActivateModule (see page 24776)	This is class WebComp.INotifyActivateModule.
	INotifyWebActivate (see page 24777)	This is class WebComp.INotifyWebActivate.
	IProducerEditorViewSupport (see page 24779)	This is class WebComp.IProducerEditorViewSupport.
	ISetWebContentOptions (see page 24781)	This is class WebComp.ISetWebContentOptions.
	ITopLevelWebComponent (see page 24782)	This is class WebComp.ITopLevelWebComponent.
	IWebComponent (see page 24782)	IWebComponent is the interface supported by components that TWebComponentList (see page 24708) manages.
	IWebComponentCommands (see page 24786)	This is class WebComp.IWebComponentCommands.
	IWebComponentContainer (see page 24788)	This is class WebComp.IWebComponentContainer.
	IWebComponentEditor (see page 24790)	IWebComponentEditor enables a Web item to be assigned children in the IDE.
	IWebComponentEditorHelper (see page 24791)	This is class WebComp.IWebComponentEditorHelper.
	IWebContent (see page 24792)	IWebContent is an interface that allows components page producers and Web items to access the HTML generated by other components.
	IWebVariablesContainerName (see page 24794)	This is class WebComp.IWebVariablesContainerName.

177.2.1 IAddFieldsEditor Interface

Class Hierarchy

WebComp.IAddFieldsEditor

Pascal

```
IAddFieldsEditor = interface;
```

C++

```
__interface IAddFieldsEditor;
```

File

WebComp

Description

This is class WebComp.IAddFieldsEditor.

177.2.1.1 IAddFieldsEditor Members

The following tables list the members exposed by IAddFieldsEditor.

Public Methods

	Name	Description
	GetAddAllFieldsItem (see page 24770)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 24770)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 24770)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetNewFieldPrefix (see page 24771)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.

177.2.1.2 IAddFieldsEditor Methods

The methods of the IAddFieldsEditor class are listed here.

Public Methods

	Name	Description
	GetAddAllFieldsItem (see page 24770)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 24770)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 24770)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetNewFieldPrefix (see page 24771)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.

177.2.1.2.1 IAddFieldsEditor.GetAddAllFieldsItem Method

Pascal

```
function GetAddAllFieldsItem: string;
```

C++

```
__fastcall AnsiString GetAddAllFieldsItem();
```

Description

This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.

177.2.1.2.2 IAddFieldsEditor.GetAddFieldsDlgCaption Method

Pascal

```
function GetAddFieldsDlgCaption: string;
```

C++

```
__fastcall AnsiString GetAddFieldsDlgCaption();
```

Description

This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.

177.2.1.2.3 IAddFieldsEditor.GetAddFieldsItem Method

Pascal

```
function GetAddFieldsItem: string;
```

C++

```
__fastcall AnsiString GetAddFieldsItem();
```

Description

This is GetAddFieldsItem, a member of class IAddFieldsEditor.

177.2.1.2.4 IAddFieldsEditor.GetNewFieldPrefix Method

Pascal

```
function GetNewFieldPrefix: string;
```

C++

```
__fastcall AnsiString GetNewFieldPrefix();
```

Description

This is GetNewFieldPrefix, a member of class IAddFieldsEditor.

177.2.2 IGetWebComponentList Interface

Class Hierarchy

WebComp.IGetWebComponentList

Pascal

```
IGetWebComponentList = interface;
```

C++

```
__interface IGetWebComponentList;
```

File

WebComp

Description

This is class WebComp.IGetWebComponentList.

177.2.2.1 IGetWebComponentList Members

The following tables list the members exposed by IGetWebComponentList.

Public Methods

	Name	Description
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.

177.2.2.2 IGetWebComponentList Methods

The methods of the IGetWebComponentList class are listed here.

Public Methods

	Name	Description
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.

177.2.2.2.1 IGetWebComponentList.GetComponentList Method

Pascal

```
function GetComponentList: TObject;
```

C++

```
__fastcall TObject * GetComponentList();
```

Description

This is GetComponentList, a member of class IGetWebComponentList.

177.2.2.2.2 IGetWebComponentList.GetDefaultComponentList Method

Pascal

```
function GetDefaultComponentList: TObject;
```

C++

```
__fastcall TObject * GetDefaultComponentList();
```

Description

This is GetDefaultComponentList, a member of class IGetWebComponentList.

177.2.3 IGetWebComponentNodeName Interface

Class Hierarchy

WebComp.IGetWebComponentNodeName

Pascal

```
IGetWebComponentNodeName = interface;
```

C++

```
__interface IGetWebComponentNodeName;
```

File

WebComp

Description

This is class WebComp.IGetWebComponentNodeName.

177.2.3.1 IGetWebComponentNodeName Members

The following tables list the members exposed by IGetWebComponentNodeName.

Public Methods

	Name	Description
	GetWebComponentNodeName (see page 24773)	This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.

177.2.3.2 IGetWebComponentNodeName Methods

The methods of the IGetWebComponentNodeName class are listed here.

Public Methods

	Name	Description
	GetWebComponentNodeName (see page 24773)	This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.

177.2.3.2.1 IGetWebComponentNodeName.GetWebComponentNodeName Method

Pascal

```
function GetWebComponentNodeName: string;
```

C++

```
__fastcall AnsiString GetWebComponentNodeName();
```

Description

This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.

177.2.4 IGetXMLStream Interface

Class Hierarchy

WebComp.IGetXMLStream

Pascal

```
IGetXMLStream = interface;
```

C++

```
__interface IGetXMLStream;
```

File

WebComp

Description

This is class WebComp.IGetXMLStream.

177.2.4.1 IGetXMLStream Members

The following tables list the members exposed by IGetXMLStream.

Public Methods

	Name	Description
	GetXMLStream (see page 24774)	This is GetXMLStream, a member of class IGetXMLStream.

177.2.4.2 IGetXMLStream Methods

The methods of the IGetXMLStream class are listed here.

Public Methods

	Name	Description
	GetXMLStream (see page 24774)	This is GetXMLStream, a member of class IGetXMLStream.

177.2.4.2.1 IGetXMLStream.GetXMLStream Method

Pascal

```
function GetXMLStream(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXMLStream(Boolean Owned);
```

Description

This is GetXMLStream, a member of class IGetXMLStream.

177.2.5 ILayoutWebContent Interface

ILayoutWebContent lets child Web items adhere to layout rules imposed by their parent.

Class Hierarchy

WebComp.ILayoutWebContent

Pascal

```
ILayoutWebContent = interface;
```

C++

```
__interface ILayoutWebContent;
```

File

WebComp

Description

The Content method, implemented by all Web items to return the generated HTML, takes as a parameter an object of type TLayout (see page 24802). TLayout (see page 24802) is any object that implements the ILayoutWebContent interface. Typically, this is a descendant of TLayoutWebContent (see page 24698).

Parent components pass the TLayout (see page 24802) object to child Web items to provide helper functions that the child items use to fit their generated content into the parent object's HTML structure (typically a <TABLE> tag). For example, the ILayoutWebContent methods may add <TD> tags around generated HTML that fills the cells of a parent's HTML table.

See Also

TLayout (see page 24802)

IWebContent (see page 24792)

TLayoutWebContent (see page 24698)

177.2.5.1 ILayoutWebContent Members

The following tables list the members exposed by ILayoutWebContent.

Public Methods

	Name	Description
	LayoutButton (see page 24775)	This is LayoutButton, a member of class ILayoutWebContent.
	LayoutField (see page 24775)	This is LayoutField, a member of class ILayoutWebContent.
	LayoutLabelAndField (see page 24775)	Converts the HTML declaring a labeled field into a form the Web item's parent can use.
	LayoutTable (see page 24776)	This is LayoutTable, a member of class ILayoutWebContent.

177.2.5.2 ILayoutWebContent Methods

The methods of the ILayoutWebContent class are listed here.

Public Methods

	Name	Description
	LayoutButton (see page 24775)	This is LayoutButton, a member of class ILayoutWebContent.
	LayoutField (see page 24775)	This is LayoutField, a member of class ILayoutWebContent.
	LayoutLabelAndField (see page 24775)	Converts the HTML declaring a labeled field into a form the Web item's parent can use.
	LayoutTable (see page 24776)	This is LayoutTable, a member of class ILayoutWebContent.

177.2.5.2.1 ILayoutWebContent.LayoutButton Method

Pascal

```
function LayoutButton(const HTMLButton: string; Attributes: TLayoutAttributes): string;
```

C++

```
__fastcall AnsiString LayoutButton(const AnsiString HTMLButton, TLayoutAttributes Attributes);
```

Description

This is LayoutButton, a member of class ILayoutWebContent.

177.2.5.2.2 ILayoutWebContent.LayoutField Method

Pascal

```
function LayoutField(const HTMLField: string; Attributes: TLayoutAttributes): string;
```

C++

```
__fastcall AnsiString LayoutField(const AnsiString HTMLField, TLayoutAttributes Attributes);
```

Description

This is LayoutField, a member of class ILayoutWebContent.

177.2.5.2.3 ILayoutWebContent.LayoutLabelAndField Method

Converts the HTML declaring a labeled field into a form the Web item's parent can use.

Pascal

```
function LayoutLabelAndField(const HTMLLabel: string; const HTMLField: string; Attributes:
TLayoutAttributes): string;
```

C++

```
__fastcall AnsiString LayoutLabelAndField(const AnsiString HTMLLabel, const AnsiString
HTMLField, TLayoutAttributes Attributes);
```

Description

Call LayoutLabelAndField to convert the HTML that declares a labeled field (an input control that represents a data field or a named value along with its label) into the form required by the Web item's parent.

HTMLLabel is the HTML that declares the label. This is produced by the Web item.

HTMLField is the HTML that declares the field. This is produced by the Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field and its label.

LayoutLabelAndField returns the HTML for the field and label in a form that can be used by the Web item's parent.

177.2.5.2.4 ILayoutWebContent.LayoutTable Method

Pascal

```
function LayoutTable(const HTMLTable: string; Attributes: TLayoutAttributes): string;
```

C++

```
__fastcall AnsiString LayoutTable(const AnsiString HTMLTable, TLayoutAttributes Attributes);
```

Description

This is LayoutTable, a member of class ILayoutWebContent.

177.2.6 INotifyActivateModule Interface

Class Hierarchy

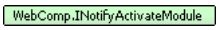

Pascal

```
INotifyActivateModule = interface;
```

C++

```
__interface INotifyActivateModule;
```

File

WebComp

Description

This is class WebComp.INotifyActivateModule.

177.2.6.1 INotifyActivateModule Members

The following tables list the members exposed by INotifyActivateModule.

Public Methods

	Name	Description
	NotifyActivateModule (see page 24777)	This is NotifyActivateModule, a member of class INotifyActivateModule.
	NotifyDeactivateModule (see page 24777)	This is NotifyDeactivateModule, a member of class INotifyActivateModule.

177.2.6.2 INotifyActivateModule Methods

The methods of the INotifyActivateModule class are listed here.

Public Methods

	Name	Description
	NotifyActivateModule (see page 24777)	This is NotifyActivateModule, a member of class INotifyActivateModule.
	NotifyDeactivateModule (see page 24777)	This is NotifyDeactivateModule, a member of class INotifyActivateModule.

177.2.6.2.1 INotifyActivateModule.NotifyActivateModule Method

Pascal

```
procedure NotifyActivateModule(AWebModule: TComponent);
```

C++

```
__fastcall NotifyActivateModule(TComponent * AWebModule);
```

Description

This is NotifyActivateModule, a member of class INotifyActivateModule.

177.2.6.2.2 INotifyActivateModule.NotifyDeactivateModule Method

Pascal

```
procedure NotifyDeactivateModule(AWebModule: TComponent);
```

C++

```
__fastcall NotifyDeactivateModule(TComponent * AWebModule);
```

Description

This is NotifyDeactivateModule, a member of class INotifyActivateModule.

177.2.7 INotifyWebActivate Interface

Class Hierarchy

WebComp.INotifyWebActivate

Pascal

```
INotifyWebActivate = interface;
```

C++

```
__interface INotifyWebActivate;
```

File

WebComp

Description

This is class WebComp.INotifyWebActivate.

177.2.7.1 INotifyWebActivate Members

The following tables list the members exposed by INotifyWebActivate.

Public Methods

	Name	Description
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.

177.2.7.2 INotifyWebActivate Methods

The methods of the INotifyWebActivate class are listed here.

Public Methods

	Name	Description
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.

177.2.7.2.1 INotifyWebActivate.NotifyActivate Method

Pascal

```
procedure NotifyActivate;
```

C++

```
__fastcall NotifyActivate();
```

Description

This is NotifyActivate, a member of class INotifyWebActivate.

177.2.7.2.2 INotifyWebActivate.NotifyDeactivate Method

Pascal

```
procedure NotifyDeactivate;
```

C++

```
__fastcall NotifyDeactivate();
```

Description

This is NotifyDeactivate, a member of class INotifyWebActivate.

177.2.8 IProducerEditorViewSupport Interface

Class Hierarchy

WebComp.IProducerEditorViewSupport

Pascal

```
IProducerEditorViewSupport = interface;
```

C++

```
__interface IProducerEditorViewSupport;
```

File

WebComp

Description

This is class WebComp.IProducerEditorViewSupport.

177.2.8.1 IProducerEditorViewSupport Members

The following tables list the members exposed by IProducerEditorViewSupport.

Public Methods

	Name	Description
	GetTemplateFileType (see page 24780)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 24780)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 24780)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 24780)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 24780)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 24781)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 24781)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.

177.2.8.2 IProducerEditorViewSupport Methods

The methods of the IProducerEditorViewSupport class are listed here.

Public Methods

	Name	Description
	GetTemplateFileType (see page 24780)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 24780)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 24780)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 24780)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 24780)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 24781)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 24781)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.

177.2.8.2.1 IProducerEditorViewSupport.GetTemplateFileType Method

Pascal

```
function GetTemplateFileType: string;
```

C++

```
__fastcall AnsiString GetTemplateFileType();
```

Description

This is GetTemplateFileType, a member of class IProducerEditorViewSupport.

177.2.8.2.2 IProducerEditorViewSupport.GetXMLData Method

Pascal

```
function GetXMLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXMLData(Boolean Owned);
```

Description

This is GetXMLData, a member of class IProducerEditorViewSupport.

177.2.8.2.3 IProducerEditorViewSupport.GetXSLData Method

Pascal

```
function GetXSLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXSLData(Boolean Owned);
```

Description

This is GetXSLData, a member of class IProducerEditorViewSupport.

177.2.8.2.4 IProducerEditorViewSupport.HasHTMLBrowserView Method

Pascal

```
function HasHTMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasHTMLBrowserView();
```

Description

This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.

177.2.8.2.5 IProducerEditorViewSupport.HasScriptView Method

Pascal

```
function HasScriptView: Boolean;
```

C++

```
__fastcall Boolean HasScriptView();
```

Description

This is HasScriptView, a member of class IProducerEditorViewSupport.

177.2.8.2.6 IProducerEditorViewSupport.HasXMLBrowserView Method

Pascal

```
function HasXMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXMLBrowserView();
```

Description

This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.

177.2.8.2.7 IProducerEditorViewSupport.HasXSLBrowserView Method

Pascal

```
function HasXSLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXSLBrowserView();
```

Description

This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.

177.2.9 ISetWebContentOptions Interface

Class Hierarchy

WebComp.ISetWebContentOptions

Pascal

```
ISetWebContentOptions = interface;
```

C++

```
__interface ISetWebContentOptions;
```

File

WebComp

Description

This is class WebComp.ISetWebContentOptions.

177.2.9.1 ISetWebContentOptions Members

The following tables list the members exposed by ISetWebContentOptions.

Public Methods

	Name	Description
	SetWebContentOptions (see page 24782)	This is SetWebContentOptions, a member of class ISetWebContentOptions.

177.2.9.2 ISetWebContentOptions Methods

The methods of the ISetWebContentOptions class are listed here.

Public Methods

	Name	Description
	SetWebContentOptions (see page 24782)	This is SetWebContentOptions, a member of class ISetWebContentOptions.

177.2.9.2.1 ISetWebContentOptions.SetWebContentOptions Method

Pascal

```
procedure SetWebContentOptions(AOptions: TWebContentOptions);
```

C++

```
__fastcall SetWebContentOptions(TWebContentOptions AOptions);
```

Description

This is SetWebContentOptions, a member of class ISetWebContentOptions.

177.2.10 ITopLevelWebComponent Interface

Class Hierarchy

WebComp.ITopLevelWebComponent

Pascal

```
ITopLevelWebComponent = interface;
```

C++

```
__interface ITopLevelWebComponent;
```

File

WebComp

Description

This is class WebComp.ITopLevelWebComponent.

177.2.11 IWebComponent Interface

IWebComponent is the interface supported by components that TWebComponentList (see page 24708) manages.

Class Hierarchy

WebComp.IWebComponent

Pascal

```
IWebComponent = interface;
```

C++

```
__interface IWebComponent;
```

File

WebComp

Description

TWebComponentList (see page 24708) uses IWebComponent to ensure that all components in its list have access to information about how they fit in the hierarchy of Web items.

Note: VCL methods that work with the IWebComponent interface wrap it using the DelphiInterface class:

Note: typedef System::DelphiInterface (see page 22167)< IWebComponent > _di_IWebComponent;

See Also

TWebComponentList (see page 24708)

177.2.11.1 IWebComponent Members

The following tables list the members exposed by IWebComponent.

Public Methods

	Name	Description
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.

Public Properties

	Name	Description
	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.

177.2.11.2 IWebComponent Methods

The methods of the IWebComponent class are listed here.

Public Methods

	Name	Description
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.

177.2.11.2.1 IWebComponent.GetContainer Method

Returns the value of the Container (see page 24786) property.

Pascal

```
function GetContainer: TWebComponentContainer;
```

C++

```
__fastcall TWebComponentContainer GetContainer();
```

Description

GetContainer is the read method for the Container (see page 24786) property.

See Also

Container (see page 24786)

177.2.11.2.2 IWebComponent.GetIndex Method

Returns the value of the Index (see page 24786) property.

Pascal

```
function GetIndex: Integer;
```

C++

```
__fastcall int GetIndex();
```

Description

GetIndex is the read method for the Index (see page 24786) property.

See Also

Index (see page 24786)

177.2.11.2.3 IWebComponent.SetComponentList Method

Sets the value of the ComponentList (see page 24785) property.

Pascal

```
procedure SetComponentList(List: TObject);
```

C++

```
__fastcall SetComponentList(TObject * List);
```

Description

SetComponentList is the write method for the ComponentList (see page 24785) property. The List parameter specifies the value to assign to ComponentList (see page 24785).

See Also

ComponentList (see page 24785)

177.2.11.2.4 IWebComponent.SetContainer Method

Sets the value of the Container (see page 24786) property.

Pascal

```
procedure SetContainer(Container: TWebComponentContainer);
```

C++

```
__fastcall SetContainer(TWebComponentContainer Container);
```

Description

SetContainer is the write method for the Container (see page 24786) property. The Container (see page 24786) parameter specifies the value to assign to Container (see page 24786).

See Also

Container (see page 24786)

177.2.11.2.5 IWebComponent.SetIndex Method

Sets the value of the Index (see page 24786) property.

Pascal

```
procedure SetIndex(Value: Integer);
```

C++

```
__fastcall SetIndex(int Value);
```

Description

SetIndex is the write method for the Index (see page 24786) property. The Value parameter specifies the value to assign to Index (see page 24786).

See Also

Index (see page 24786)

177.2.11.3 IWebComponent Properties

The properties of the IWebComponent class are listed here.

Public Properties

	Name	Description
	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.

177.2.11.3.1 IWebComponent.ComponentList Property

Indicates the component list that contains the Web item.

Pascal

```
property ComponentList: TObject;
```

C++

```
__property TObject * ComponentList;
```

Description

The Web component list assigns a value to ComponentList when the component is added to it. The component can use this

value to access the list and obtain information about the other components in the list.

Note that the IWebComponent (see page 24782) interface does not include any methods for reading the value of ComponentList.

See Also

Container (see page 24786)

Index (see page 24786)

TWebComponentList (see page 24708)

177.2.11.3.2 IWebComponent.Container Property

Specifies the component that uses content generated by this Web item.

Pascal

```
property Container: TWebComponentContainer;
```

C++

```
__property TWebComponentContainer Container;
```

Description

Web items are arranged in a hierarchy, where each item generates its content using content produced by its children. Container specifies the parent that uses the content of this Web item. Container maintains a list of all its children using a TWebComponentList (see page 24708) object.

See Also

ComponentList (see page 24785)

ParentComponent (see page 24718)

177.2.11.3.3 IWebComponent.Index Property

Indicates where the Web item appears in the Web component list's array of Web items.

Pascal

```
property Index: Integer;
```

C++

```
__property int Index;
```

Description

The Web component list sets Index to indicate where the component sits in its WebComponents property array.

See Also

WebComponents (see page 24718)

177.2.12 IWebComponentCommands Interface

Class Hierarchy

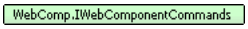

Pascal

```
IWebComponentCommands = interface;
```

C++

```
__interface IWebComponentCommands;
```

File

WebComp

Description

This is class WebComp.IWebComponentCommands.

177.2.12.1 IWebComponentCommands Members

The following tables list the members exposed by IWebComponentCommands.

Public Methods

	Name	Description
	ExecuteVerb (see page 24787)	This is ExecuteVerb, a member of class IWebComponentCommands.
	GetVerb (see page 24787)	This is GetVerb, a member of class IWebComponentCommands.
	GetVerbCount (see page 24788)	This is GetVerbCount, a member of class IWebComponentCommands.

177.2.12.2 IWebComponentCommands Methods

The methods of the IWebComponentCommands class are listed here.

Public Methods

	Name	Description
	ExecuteVerb (see page 24787)	This is ExecuteVerb, a member of class IWebComponentCommands.
	GetVerb (see page 24787)	This is GetVerb, a member of class IWebComponentCommands.
	GetVerbCount (see page 24788)	This is GetVerbCount, a member of class IWebComponentCommands.

177.2.12.2.1 IWebComponentCommands.ExecuteVerb Method

Pascal

```
procedure ExecuteVerb(AComponent: TComponent; Index: Integer);
```

C++

```
__fastcall ExecuteVerb(TComponent * AComponent, int Index);
```

Description

This is ExecuteVerb, a member of class IWebComponentCommands.

177.2.12.2.2 IWebComponentCommands.GetVerb Method

Pascal

```
function GetVerb(AComponent: TComponent; Index: Integer): string;
```

C++

```
__fastcall AnsiString GetVerb(TComponent * AComponent, int Index);
```

Description

This is GetVerb, a member of class IWebComponentCommands.

177.2.12.2.3 IWebComponentCommands.GetVerbCount Method

Pascal

```
function GetVerbCount(AComponent: TComponent): Integer;
```

C++

```
__fastcall int GetVerbCount(TComponent * AComponent);
```

Description

This is GetVerbCount, a member of class IWebComponentCommands.

177.2.13 IWebComponentContainer Interface

Class Hierarchy

WebComp.IWebComponentContainer

Pascal

```
IWebComponentContainer = interface;
```

C++

```
__interface IWebComponentContainer;
```

File

WebComp

Description

This is class WebComp.IWebComponentContainer.

177.2.13.1 IWebComponentContainer Members

The following tables list the members exposed by IWebComponentContainer.

Public Methods

	Name	Description
	GetComponent (see page 24789)	This is GetComponent, a member of class IWebComponentContainer.
	GetComponentCount (see page 24789)	This is GetComponentCount, a member of class IWebComponentContainer.

Public Properties

	Name	Description
R	ComponentCount (see page 24789)	This is ComponentCount, a member of class IWebComponentContainer.
R	Components (see page 24790)	This is Components, a member of class IWebComponentContainer.

177.2.13.2 IWebComponentContainer Methods

The methods of the IWebComponentContainer class are listed here.

Public Methods

	Name	Description
	GetComponent (see page 24789)	This is GetComponent, a member of class IWebComponentContainer.
	GetComponentCount (see page 24789)	This is GetComponentCount, a member of class IWebComponentContainer.

177.2.13.2.1 IWebComponentContainer.GetComponent Method

Pascal

```
function GetComponent(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetComponent(int I);
```

Description

This is GetComponent, a member of class IWebComponentContainer.

177.2.13.2.2 IWebComponentContainer.GetComponentCount Method

Pascal

```
function GetComponentCount: Integer;
```

C++

```
__fastcall int GetComponentCount();
```

Description

This is GetComponentCount, a member of class IWebComponentContainer.

177.2.13.3 IWebComponentContainer Properties

The properties of the IWebComponentContainer class are listed here.

Public Properties

	Name	Description
R	ComponentCount (see page 24789)	This is ComponentCount, a member of class IWebComponentContainer.
R	Components (see page 24790)	This is Components, a member of class IWebComponentContainer.

177.2.13.3.1 IWebComponentContainer.ComponentCount Property

Pascal

```
property ComponentCount: Integer;
```

C++

```
__property int ComponentCount;
```

Description

This is ComponentCount, a member of class IWebComponentContainer.

177.2.13.3.2 IWebComponentContainer.Components Property

Pascal

```
property Components [I: Integer]: TComponent;
```

C++

```
__property TComponent * Components[int I];
```

Description

This is Components, a member of class IWebComponentContainer.

177.2.14 IWebComponentEditor Interface

IWebComponentEditor enables a Web item to be assigned children in the IDE.

Class Hierarchy

Pascal

```
IWebComponentEditor = interface;
```

C++

```
__interface IWebComponentEditor;
```

File

WebComp

Description

When the Web Page editor builds the list of new items that can be added as the child of a Web item, it calls the IWebComponentEditor interface. Components that contain child items for generating portions of their HTML content must implement this interface so that the Web Page editor can display the appropriate list of components to add.

IWebComponentEditor defines a single method, CanAddClass (see page 24791), which indicates whether a specified class can be used for a child Web item by the Web item that implements the interface.

See Also

IRestoreDefaults

TAdapterPageProducer (see page 19802)

TInetXPageProducer (see page 17145)

TWebForm (see page 16922)

;TWebControlGroup

TAdapterSubComponent (see page 23991)

177.2.14.1 IWebComponentEditor Members

The following tables list the members exposed by IWebComponentEditor.

Public Methods

	Name	Description
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.

177.2.14.2 IWebComponentEditor Methods

The methods of the IWebComponentEditor class are listed here.

Public Methods

	Name	Description
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.

177.2.14.2.1 IWebComponentEditor.CanAddClass Method

Indicates whether a specified class can be used as a child Web item.

Pascal

```
function CanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClass(TComponent * AParent, TClass AClass);
```

Description

The Web page editor calls CanAddClass to determine what Web items can be added to supply content for the Web item that implements this interface.

AParent is the Web item that will act as a parent to the added Web item.

AClass is the class of the proposed child Web item.

CanAddClass returns true if the Web item can use content generated by a component of type AClass, false otherwise.

177.2.15 IWebComponentEditorHelper Interface

Class Hierarchy

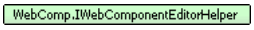

Pascal

```
IWebComponentEditorHelper = interface;
```

C++

```
__interface IWebComponentEditorHelper;
```

File

WebComp

Description

This is class WebComp.IWebComponentEditorHelper.

177.2.15.1 IWebComponentEditorHelper Members

The following tables list the members exposed by IWebComponentEditorHelper.

Public Methods

	Name	Description
	CanAddClassHelper (see page 24792)	This is CanAddClassHelper, a member of class IWebComponentEditorHelper.

177.2.15.2 IWebComponentEditorHelper Methods

The methods of the IWebComponentEditorHelper class are listed here.

Public Methods

	Name	Description
	CanAddClassHelper (see page 24792)	This is CanAddClassHelper, a member of class IWebComponentEditorHelper.

177.2.15.2.1 IWebComponentEditorHelper.CanAddClassHelper Method

Pascal

```
function CanAddClassHelper(AEditor: TComponent; AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClassHelper(TComponent * AEditor, TComponent * AParent, TClass
AClass);
```

Description

This is CanAddClassHelper, a member of class IWebComponentEditorHelper.

177.2.16 IWebContent Interface

IWebContent is an interface that allows components page producers and Web items to access the HTML generated by other components.

Class Hierarchy

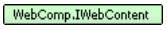

Pascal

```
IWebContent = interface;
```

C++

```
__interface IWebContent;
```

File

WebComp

Description

TInetXPageProducer, TAdapterPageProducer, and TComponentsPageProducer create HTML documents that include content generated by other components, called Web items. Similarly, the Web items that generate content can make use of other Web items when producing their content.

For a component to act as a Web item, it must support the IWebContent interface. This interface defines one new method, Content (see page 24793), which returns the HTML generated by the Web item.

Note: The only other required interface for Web items is IWebComponent (see page 24782).

See Also

TAdapterPageProducer (see page 19802)

TInetXPageProducer (see page 17145)

TBasePageItemsProducer (see page 18376)

TWebComponentList (see page 24708)

IWebComponent (see page 24782)

Customizing the InternetExpress Page Producer Template

177.2.16.1 IWebContent Members

The following tables list the members exposed by IWebContent.

Public Methods

	Name	Description
	Content (see page 24793)	Returns the HTML that the Web item generates.

177.2.16.2 IWebContent Methods

The methods of the IWebContent class are listed here.

Public Methods

	Name	Description
	Content (see page 24793)	Returns the HTML that the Web item generates.

177.2.16.2.1 IWebContent.Content Method

Returns the HTML that the Web item generates.

Pascal

```
function Content(Options: TWebContentOptions; ParentLayout: TLayout): string;
```

C++

```
__fastcall AnsiString Content(TWebContentOptions Options, TLayout ParentLayout);
```

Description

Call Content to access the HTML produced by the Web item.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web item should lay out its constituent parts. This object supports the ILayoutWebContent (see page 24774) interface, which the Web item uses to fit its own content into the parent's HTML structures.

Content returns a string of HTML produced by the Web item.

See Also

ILayoutWebContent (see page 24774)

177.2.17 IWebVariablesContainerName Interface

Class Hierarchy

WebComp.IWebVariablesContainerName

Pascal

```
IWebVariablesContainerName = interface;
```

C++

```
__interface IWebVariablesContainerName;
```

File

WebComp

Description

This is class WebComp.IWebVariablesContainerName.

177.2.17.1 IWebVariablesContainerName Members

The following tables list the members exposed by IWebVariablesContainerName.

Public Methods

	Name	Description
	GetContainerName (see page 24794)	This is GetContainerName, a member of class IWebVariablesContainerName.

Public Properties

	Name	Description
R	ContainerName (see page 24795)	This is ContainerName, a member of class IWebVariablesContainerName.

177.2.17.2 IWebVariablesContainerName Methods

The methods of the IWebVariablesContainerName class are listed here.

Public Methods

	Name	Description
	GetContainerName (see page 24794)	This is GetContainerName, a member of class IWebVariablesContainerName.

177.2.17.2.1 IWebVariablesContainerName.GetContainerName Method

Pascal

```
function GetContainerName: string;
```

C++

```
__fastcall AnsiString GetContainerName();
```

Description

This is GetContainerName, a member of class IWebVariablesContainerName.

177.2.17.3 IWebVariablesContainerName Properties

The properties of the IWebVariablesContainerName class are listed here.

Public Properties

	Name	Description
R	ContainerName (see page 24795)	This is ContainerName, a member of class IWebVariablesContainerName.

177.2.17.3.1 IWebVariablesContainerName.ContainerName Property

Pascal

```
property ContainerName: string;
```

C++

```
__property AnsiString ContainerName;
```

Description

This is ContainerName, a member of class IWebVariablesContainerName.

177.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DesigningComponent (see page 24796)	This is function WebComp.DesigningComponent.
	EnumRegisteredWebComponents (see page 24796)	This is function WebComp.EnumRegisteredWebComponents.
	FindWebComponentEditorHelper (see page 24797)	This is function WebComp.FindWebComponentEditorHelper.
	QualifyFileName (see page 24797)	This is function WebComp.QualifyFileName.
	RegisterWebComponents (see page 24797)	This is function WebComp.RegisterWebComponents.
	UnRegisterWebComponents (see page 24798)	This is function WebComp.UnRegisterWebComponents.
	ValidateWebParent (see page 24798)	This is function WebComp.ValidateWebParent.

177.3.1 WebComp.DesigningComponent Function

Pascal

```
function DesigningComponent(AComponent: TComponent): Boolean;
```

C++

```
Boolean DesigningComponent(TComponent * AComponent);
```

File

WebComp

Description

This is function WebComp.DesigningComponent.

177.3.2 WebComp.EnumRegisteredWebComponents Function

Pascal

```
procedure EnumRegisteredWebComponents(Proc: TEnumWebComponentsProc; Info: Pointer);
```

C++

```
EnumRegisteredWebComponents(TEnumWebComponentsProc Proc, void * Info);
```

File

WebComp

Description

This is function WebComp.EnumRegisteredWebComponents.

177.3.3 WebComp.FindWebComponentEditorHelper Function

Pascal

```
function FindWebComponentEditorHelper(AClass: TClass): TObject;
```

C++

```
TObject * FindWebComponentEditorHelper(TClass AClass);
```

File

WebComp

Description

This is function WebComp.FindWebComponentEditorHelper.

177.3.4 WebComp.QualifyFileName Function

Pascal

```
function QualifyFileName(const AFileName: string): string;
```

C++

```
AnsiString QualifyFileName(const AnsiString AFileName);
```

File

WebComp

Description

This is function WebComp.QualifyFileName.

177.3.5 WebComp.RegisterWebComponents Function

Pascal

```
procedure RegisterWebComponents(const AClasses: array of TComponentClass; Editor: TObject =
nil);
```

C++

```
RegisterWebComponents(const array of TComponentClass AClasses, TObject * Editor = nil);
```

File

WebComp

Description

This is function WebComp.RegisterWebComponents.

177.3.6 WebComp.UnRegisterWebComponents Function

Pascal

```
procedure UnRegisterWebComponents(const AClasses: array of TComponentClass);
```

C++

```
UnRegisterWebComponents(const array of TComponentClass AClasses);
```

File

WebComp

Description

This is function WebComp.UnRegisterWebComponents.

177.3.7 WebComp.ValidateWebParent Function

Pascal

```
procedure ValidateWebParent(AComponent: TComponent; AParent: TComponent);
```

C++

```
ValidateWebParent(TComponent * AComponent, TComponent * AParent);
```

File

WebComp

Description

This is function WebComp.ValidateWebParent.

177.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TLabelPosition (see page 24799)	This is record WebComp.TLabelPosition.
	TWebContentFlag (see page 24799)	This is record WebComp.TWebContentFlag.

177.4.1 WebComp.TLabelPosition Enumeration

Pascal

```
TLabelPosition = (
  lposLeft,
  lposRight,
  lposAbove,
  lposBelow
);
```

C++

```
enum TLabelPosition {
  lposLeft,
  lposRight,
  lposAbove,
  lposBelow
};
```

File

WebComp

Description

This is record WebComp.TLabelPosition.

177.4.2 WebComp.TWebContentFlag Enumeration

Pascal

```
TWebContentFlag = (
  coNoScript,
  coLocalPaths,
  coNoExecuteScript
);
```

C++

```
enum TWebContentFlag {
  coNoScript,
  coLocalPaths,
  coNoExecuteScript
};
```

File

WebComp

Description

This is record WebComp.TWebContentFlag.

177.5 Types

The following table lists types in this documentation.

Types

Name	Description
TEnumWebComponentsProc (see page 24801)	This is type WebComp.TEnumWebComponentsProc.
TGetComponentListEvent (see page 24801)	This is type WebComp.TGetComponentListEvent.
TLayout (see page 24802)	TLayout is the type for helper objects that generate layout code for a Web item.
TSetComponentListEvent (see page 24802)	This is type WebComp.TSetComponentListEvent.
TWebComponentContainer (see page 24802)	TWebComponentContainer is the type for components that maintain a list of Web items.
TWebContentFlags (see page 24803)	This is type WebComp.TWebContentFlags.

177.5.1 WebComp.TEnumWebComponentsProc Type

Pascal

```
TEnumWebComponentsProc = procedure (WebComponentClass: TWebComponentClassDescription; Info:
Pointer) of object;
```

C++

```
(WebComponentClass: TWebComponentClassDescription; Info: Pointer) ( TEnumWebComponentsProc)();
```

File

WebComp

Description

This is type WebComp.TEnumWebComponentsProc.

177.5.2 WebComp.TGetComponentListEvent Type

Pascal

```
TGetComponentListEvent = procedure (Sender: TObject; var AList: TWebComponentList) of object;
```

C++

```
(Sender: TObject; var AList: TWebComponentList) ( TGetComponentListEvent)();
```

File

WebComp

Description

This is type WebComp.TGetComponentListEvent.

177.5.3 WebComp.TLayout Type

TLayout is the type for helper objects that generate layout code for a Web item.

Pascal

```
TLayout = TObject;
```

C++

```
TObject * TLayout;
```

File

WebComp

Description

TLayout is the type for objects that Web items use to transmit common functions that control the layout of HTML controls. When a Web item generates content that includes HTML produced by child components, it passes a TLayout object to those children. The TLayout object implements an interface (ILayoutWebContent (see page 24774)) that generates HTML to lay out the controls which the child components generate.

Note: Typically, TLayout objects are descendants of TLayoutWebContent (see page 24698).

See Also

IWebContent (see page 24792)

ILayoutWebContent (see page 24774)

TLayoutWebContent (see page 24698)

177.5.4 WebComp.TSetComponentListEvent Type

Pascal

```
TSetComponentListEvent = procedure (Sender: TObject; AList: TWebComponentList) of object;
```

C++

```
(Sender: TObject; AList: TWebComponentList) ( TSetComponentListEvent)();
```

File

WebComp

Description

This is type WebComp.TSetComponentListEvent.

177.5.5 WebComp.TWebComponentContainer Type

TWebComponentContainer is the type for components that maintain a list of Web items.

Pascal

```
TWebComponentContainer = TObject;
```

C++

```
TObject * TWebComponentContainer;
```

File

WebComp

Description

TWebComponentContainer is the type for objects that make use of Web items (components that generate HTML and support the IWebContent (see page 24792) interface). Each TWebComponentContainer has a property of type TWebComponentList (see page 24708) that lists its Web items. The TWebComponentContainer uses these Web items to generate constituent parts of the HTML that it generates.

See Also

TWebComponentList (see page 24708)

IWebContent (see page 24792)

177.5.6 WebComp.TWebContentFlags Type

Pascal

```
TWebContentFlags = set of TWebContentFlag;
```

C++

```
TWebContentFlag TWebContentFlags;
```

File

WebComp

Description

This is type WebComp.TWebContentFlags.

177.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
EnumRegisteredWebComponentsProc (see page 24804)	This is constant WebComp.EnumRegisteredWebComponentsProc.
FindWebComponentEditorHelperProc (see page 24804)	This is constant WebComp.FindWebComponentEditorHelperProc.
RegisterWebComponentsProc (see page 24805)	This is constant WebComp.RegisterWebComponentsProc.
UnRegisterWebComponentsProc (see page 24805)	This is constant WebComp.UnRegisterWebComponentsProc.

177.6.1 WebComp.EnumRegisteredWebComponentsProc Constant

Pascal

```
EnumRegisteredWebComponentsProc: procedure (Proc: TEnumWebComponentsProc; Info: Pointer) = nil;
```

C++

```
procedure (Proc: TEnumWebComponentsProc; Info: Pointer) EnumRegisteredWebComponentsProc = nil;
```

File

WebComp

Description

This is constant WebComp.EnumRegisteredWebComponentsProc.

177.6.2 WebComp.FindWebComponentEditorHelperProc Constant

Pascal

```
FindWebComponentEditorHelperProc: function (AClass: TClass): TObject = nil;
```

C++

```
function (AClass: TClass): TObject FindWebComponentEditorHelperProc = nil;
```

File

WebComp

Description

This is constant WebComp.FindWebComponentEditorHelperProc.

177.6.3 WebComp.RegisterWebComponentsProc Constant

Pascal

```
RegisterWebComponentsProc: procedure ( const AClasses: array of TComponentClass; Editor:
TObject) = nil;
```

C++

```
procedure ( const AClasses: array of TComponentClass; Editor: TObject)
RegisterWebComponentsProc = nil;
```

File

WebComp

Description

This is constant WebComp.RegisterWebComponentsProc.

177.6.4 WebComp.UnRegisterWebComponentsProc Constant

Pascal

```
UnRegisterWebComponentsProc: procedure (const AClasses: array of TComponentClass) = nil;
```

C++

```
procedure (const AClasses: array of TComponentClass) UnRegisterWebComponentsProc = nil;
```

File

WebComp

Description

This is constant WebComp.UnRegisterWebComponentsProc.

178 WebContnrs Namespace

This is namespace WebContnrs.

Topics

Name	Description
Classes (see page 24808)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 24832)	The following table lists structs, records, enums in this documentation.
Types (see page 24833)	The following table lists types in this documentation.

Classes

	Name	Description
	ENamedVariantsError (see page 24808)	ENamedVariantsError is the exception for errors encountered by a TNamedVariantsList (see page 24824) object.
	TAbstractNamedVariants (see page 24808)	TAbstractNamedVariants is the base class for persistent lists of Variant values.
	TNamedVariantsList (see page 24824)	TNamedVariantsList is a persistent list of name/Variant pairs.

Structs, Records, Enums

	Name	Description
	TNamedVariantItem (see page 24832)	This is record WebContnrs.TNamedVariantItem.

Types

Name	Description
PNamedVariantItem (see page 24833)	This is type WebContnrs.PNamedVariantItem.
PNamedVariantList (see page 24833)	This is type WebContnrs.PNamedVariantList.
TNamedVariantList (see page 24833)	This is type WebContnrs.TNamedVariantList.

178.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ENamedVariantsError (see page 24808)	ENamedVariantsError is the exception for errors encountered by a TNamedVariantsList (see page 24824) object.
	TAbstractNamedVariants (see page 24808)	TAbstractNamedVariants is the base class for persistent lists of Variant values.
	TNamedVariantsList (see page 24824)	TNamedVariantsList is a persistent list of name/Variant pairs.

178.1.1 ENamedVariantsError Class

ENamedVariantsError is the exception for errors encountered by a TNamedVariantsList (see page 24824) object.

Class Hierarchy

System::TObject → SysUtils.Exception → WebContnrs.ENamedVariantsError

Pascal

```
ENamedVariantsError = class(Exception);
```

C++

```
class ENamedVariantsError : public Exception;
```

File

WebContnrs

Description

ENamedVariantsError is raised when an error occurs in TNamedVariantsList (see page 24824). For example, when an application supplies an invalid index to one of the TNamedVariantsList (see page 24824) methods, it raises an ENamedVariantsError exception.

See Also

TNamedVariantsList (see page 24824)

178.1.2 TAbstractNamedVariants Class

TAbstractNamedVariants is the base class for persistent lists of Variant values.

Class Hierarchy

System::TObject → Classes.TPersistent → WebContnrs.TAbstractNamedVariants

Pascal

```
TAbstractNamedVariants = class(TPersistent);
```

C++

```
class TAbstractNamedVariants : public TPersistent;
```

File

WebContnrs

Description

Use TAbstractNamedVariants as a base class for objects that represents a set of name/value pairs where the values are all Variants (see page 24824). TAbstractNamedVariants is a descendant of TPersistent, which means that if it is the type of a published property on a component, the named variants list is saved with the component.

Do not create instances of TAbstractNamedVariants: This class contains abstract or, in C++ terminology, pure virtual methods. Instead, use a descendant of TAbstractNamedVariants such as TNamedVariantsList (see page 24824). TNamedVariantsList (see page 24824) provides an implementation for the abstract or pure virtual methods introduced in TAbstractNamedVariants.

See Also

TStrings (see page 3029)

TCollection (see page 2731)

TList (see page 2868)

TNamedVariantsList (see page 24824)

178.1.2.1 TAbstractNamedVariants Members

The following tables list the members exposed by TAbstractNamedVariants.

Protected Methods

	Name	Description
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	CompareStrings (see page 24814)	Compares two strings.
V	DefineProperties (see page 24814)	Allows the list of names and their corresponding Variant values to be saved with the named Variants (see page 24824) list.
	Error (see page 24816)	Raises an ENamedVariantsError (see page 24808) exception.
A	Get (see page 24817)	Retrieves the name and Variant stored at a specified position in the list.
V	GetCapacity (see page 24817)	Returns the value of the Capacity (see page 24821) property.
A	GetCount (see page 24818)	Returns the value of the Count (see page 24822) property.
	GetOwner (see page 2932)	Returns the owner of an object.
A	Put (see page 24820)	Stores a name/Variant pair at a specified position in the list.
V	SetCapacity (see page 24820)	Sets the value of the Capacity (see page 24821) property.
V	SetUpdateState (see page 24821)	Responds when an update operation begins or ends.

Protected Properties

	Name	Description
R	UpdateCount (see page 24823)	Indicates the number of calls to BeginUpdate (see page 24813) that have not been matched by a corresponding call to EndUpdate (see page 24815).

Public Methods

	Name	Description
V	Add (see page 24812)	Adds a named Variant to the end of the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 24812)	Adds a named Variant to the end of the list.
V	Assign (see page 24813)	Copies the list of named Variants (see page 24824) from another object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	BeginUpdate (see page 24813)	Signals the start of an update operation.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 24813)	Empties the list of all its name/Variant pairs.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
A	Delete (see page 24815)	Deletes an entry from the list.
V	Destroy (see page 24810)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 24815)	Signals the end of an update operation.
	Equals (see page 24816)	Compares this named Variants (see page 24824) list with another one.
V	Exchange (see page 24817)	Swaps the positions of two items in the list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	IndexOfName (see page 24818)	Returns the index of the first item in the list with a specified name.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
A	Insert (see page 24819)	Inserts a name/Variant pair at a specified position.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	Move (see page 24819)	Moves a name/Variant pair to a new position in the list.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 24821)	Specifies the allocated size of the named variants list.
R	Count (see page 24822)	Indicates the number of name/value pairs in the list.
R	Names (see page 24822)	Lists the names of all the Variants (see page 24824) in the list.
	Values (see page 24823)	Lists the Variants (see page 24824) in the list by name.
	Variants (see page 24824)	Lists the Variants in the list by position.

178.1.2.2 TAbstractNamedVariants.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAbstractNamedVariants();
```

Description

Do not call ~Destroy directly. Instead, use the delete keyword when destroying VCL objects.

~Destroy deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

Warning: Never destroy a component from one of its own event handlers or from the event handler of a component it owns or contains. For example, don't destroy a button, or the form that owns the button, in the button's OnClick event handler.

To destroy a form, call its Release method, which destroys the form and releases the memory allocated for it after all its event handlers and those of the components it contains have executed.

See Also

delete

178.1.2.3 TAbstractNamedVariants Methods

The methods of the TAbstractNamedVariants class are listed here.

Protected Methods

	Name	Description
V	CompareStrings (see page 24814)	Compares two strings.
V	DefineProperties (see page 24814)	Allows the list of names and their corresponding Variant values to be saved with the named Variants (see page 24824) list.
	Error (see page 24816)	Raises an ENamedVariantsError (see page 24808) exception.
A	Get (see page 24817)	Retrieves the name and Variant stored at a specified position in the list.
V	GetCapacity (see page 24817)	Returns the value of the Capacity (see page 24821) property.
A	GetCount (see page 24818)	Returns the value of the Count (see page 24822) property.
A	Put (see page 24820)	Stores a name/Variant pair at a specified position in the list.
V	SetCapacity (see page 24820)	Sets the value of the Capacity (see page 24821) property.
V	SetUpdateState (see page 24821)	Responds when an update operation begins or ends.

Public Methods

	Name	Description
V	Add (see page 24812)	Adds a named Variant to the end of the list.
	Append (see page 24812)	Adds a named Variant to the end of the list.
V	Assign (see page 24813)	Copies the list of named Variants (see page 24824) from another object.
	BeginUpdate (see page 24813)	Signals the start of an update operation.
A	Clear (see page 24813)	Empties the list of all its name/Variant pairs.
A	Delete (see page 24815)	Deletes an entry from the list.
	EndUpdate (see page 24815)	Signals the end of an update operation.
	Equals (see page 24816)	Compares this named Variants (see page 24824) list with another one.
V	Exchange (see page 24817)	Swaps the positions of two items in the list.
V	IndexOfName (see page 24818)	Returns the index of the first item in the list with a specified name.
A	Insert (see page 24819)	Inserts a name/Variant pair at a specified position.

	Move (see page 24819)	Moves a name/Variant pair to a new position in the list.

178.1.2.3.1 TAbstractNamedVariants.Add Method

Adds a named Variant to the end of the list.

Pascal

```
function Add(const S: string; const AValue: Variant): Integer; virtual;
```

C++

```
virtual __fastcall int Add(const AnsiString S, const Variant AValue);
```

Description

Call Add to add a name/Variant pair to the end of the list. Add adds a new name/Variant pair to the list with a position index equal to the value of the Count (see page 24822) property.

S is the name of the Variant value. This can be used to look up the value using the Values (see page 24823) property.

AValue is the Variant value to associate with S.

Note: Unlike setting the Values (see page 24823) property, calling Add adds a duplicate name/Variant pair to the list if S matches the name of a Variant that is already in the list.

See Also

Values (see page 24823)

Append (see page 24812)

Insert (see page 24819)

Delete (see page 24815)

Count (see page 24822)

178.1.2.3.2 TAbstractNamedVariants.Append Method

Adds a named Variant to the end of the list.

Pascal

```
procedure Append(const S: string; const AValue: Variant);
```

C++

```
__fastcall Append(const AnsiString S, const Variant AValue);
```

Description

Call Append to add a name/Variant pair to the end of the list. Append simply calls the Add (see page 24812) method to add the specified name and its associated Variant value.

S is the name of the Variant value. This can be used to look up the value using the Values (see page 24823) property.

AValue is the Variant value to associate with S.

See Also

Values (see page 24823)

Add (see page 24812)

Insert (see page 24819)

Delete (see page 24815)

178.1.2.3.3 TAbstractNamedVariants.Assign Method

Copies the list of named Variants (see page 24824) from another object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy another list to this named Variants (see page 24824) list. If the source object is another named Variants (see page 24824) list, Assign discards any name/Variant pairs currently in the list and replaces them with the name/Variant pairs in the source list. If the source is any other type of object, Assign calls the inherited method, which copies the list from any object that implements the AssignTo method so that it copies its items to a named Variants (see page 24824) list.

Source is the object from which to copy the list.

See Also

Values (see page 24823)

AssignTo (see page 2931)

178.1.2.3.4 TAbstractNamedVariants.BeginUpdate Method

Signals the start of an update operation.

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

Call BeginUpdate before starting an operation that makes multiple changes to the named Variants (see page 24824) list. After completing all of the changes, call the EndUpdate (see page 24815) method to signal the end of the operation.

Every call to BeginUpdate must be matched by a corresponding call to the EndUpdate (see page 24815) method.

Note: TAbstractNamedVariants (see page 24808) calls BeginUpdate and EndUpdate (see page 24815) internally when performing operations that affect multiple entries in the list.

See Also

EndUpdate (see page 24815)

178.1.2.3.5 TAbstractNamedVariants.Clear Method

Empties the list of all its name/Variant pairs.

Pascal

```
procedure Clear; virtual; abstract;
```

C++

```
virtual __fastcall Clear() = 0;
```

Description

Call Clear to empty the named Variants (see page 24824) list. After a call to Clear, the value of the Count (see page 24822) property is zero.

TAbstractNamedVariants (see page 24808) declares Clear as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Count (see page 24822)

Delete (see page 24815)

178.1.2.3.6 TAbstractNamedVariants.CompareStrings Method

Compares two strings.

Pascal

```
function CompareStrings(const S1: string; const S2: string): Integer; virtual;
```

C++

```
virtual __fastcall int CompareStrings(const AnsiString S1, const AnsiString S2);
```

Description

TAbstractNamedVariants (see page 24808) calls CompareStrings internally to locate names in the list (for example, in the implementation of the Values (see page 24823) property). Descendant classes can use this method to sort the list entries by name.

S1 and S2 are the strings to compare.

CompareStrings returns a value less than 0 if S1 < S2, 0 if S1 = S2, and a value greater than 0 if S1 > S2.

In TAbstractNamedVariants (see page 24808), CompareStrings calls the AnsiCompareText function. This performs a locale-based case-insensitive comparison. Override this method in a descendant class to change the way the named Variants (see page 24824) list compares names. For example, by changing CompareStrings to a case sensitive function, the Values (see page 24823) property would distinguish two names that differed only in case.

See Also

AnsiCompareText (see page 22790)

Values (see page 24823)

Equals (see page 24816)

178.1.2.3.7 TAbstractNamedVariants.DefineProperties Method

Allows the list of names and their corresponding Variant values to be saved with the named Variants (see page 24824) list.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

The component streaming system calls DefineProperties when it loads or saves the named Variants (see page 24824) list. TAbstractNamedVariants (see page 24808) overrides the inherited method to specify how to load and save the list of names and Variant values.

Filer is the current TReader or TWriter object that is loading or saving the named Variants (see page 24824) list.

See Also

Values (see page 24823)

DefineProperties (see page 2931)

TReader (see page 2933)

TWriter (see page 3175)

178.1.2.3.8 TAbstractNamedVariants.Delete Method

Deletes an entry from the list.

Pascal

```
procedure Delete(Index: Integer); virtual; abstract;
```

C++

```
virtual __fastcall Delete(int Index) = 0;
```

Description

Call Delete to remove a specified name/Variant pair from the list. After a call to Delete, any name/Variant pairs that appeared after the deleted item in the list move up by one (getting a new index).

Index is the position of the item to delete.

TAbstractNamedVariants (see page 24808) declares Delete as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Names (see page 24822)

Variants (see page 24824)

Clear (see page 24813)

178.1.2.3.9 TAbstractNamedVariants.EndUpdate Method

Signals the end of an update operation.

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

Call EndUpdate after completing an operation that was preceded by a call to the BeginUpdate (see page 24813) method. Every call to BeginUpdate (see page 24813) must be matched by a corresponding call to the EndUpdate method. Calls to BeginUpdate (see page 24813) and EndUpdate can be nested.

Note: TAbstractNamedVariants (see page 24808) calls BeginUpdate (see page 24813) and EndUpdate internally when performing operations that affect multiple entries in the list.

See Also

BeginUpdate (see page 24813)

178.1.2.3.10 TAbstractNamedVariants.Equals Method

Compares this named Variants (see page 24824) list with another one.

Pascal

```
function Equals(ANamedVariants: TAbstractNamedVariants): Boolean;
```

C++

```
__fastcall Boolean Equals(TAbstractNamedVariants ANamedVariants);
```

Description

Call Equals to determine whether another named Variants (see page 24824) list contains the same name/Variant pairs as this one.

ANamedVariants is the named Variants (see page 24824) list to compare to this one.

Equals returns true if every entry in ANamedVariants has the same name and Variant value as the corresponding entry in this list. Equals returns false if ANamedVariants has a different number of entries, if its entries appear in a different order, or if it contains different entries.

Equals considers two name to be the same if they differ only by case.

178.1.2.3.11 TAbstractNamedVariants.Error Method

Raises an ENamedVariantsError (see page 24808) exception.

Pascal

```
procedure Error(const Msg: string; Data: Integer); overload;
```

C++

```
__fastcall Error(const AnsiString Msg, int Data);
```

Description

TAbstractNamedVariants (see page 24808) calls Error internally when it encounters an invalid index. For example, if an application tries to delete or change a name/Variant pair and supplies an index that is greater than Count (see page 24822), TAbstractNamedVariants (see page 24808) calls Error to generate the exception about the out-of-bounds index.

Msg is the error message for the exception. This message contains a format specifier for the integer that represents the index that caused the problem.

Data is the index that caused the problem.

See Also

ENamedVariantsError (see page 24808)

Count (see page 24822)

178.1.2.3.12 TAbstractNamedVariants.Exchange Method

Swaps the positions of two items in the list.

Pascal

```
procedure Exchange(Index1: Integer; Index2: Integer); virtual;
```

C++

```
virtual __fastcall Exchange(int Index1, int Index2);
```

Description

Call Exchange to interchange the positions of two name/Variant pairs.

Index1 and Index2 are the indexes of the two entries to swap. Exchange copies the name and Variant stored at Index1 to the position specified by Index2, and the name and Variant from Index2 to the position specified by Index1.

See Also

Move (see page 24819)

178.1.2.3.13 TAbstractNamedVariants.Get Method

Retrieves the name and Variant stored at a specified position in the list.

Pascal

```
function Get(Index: Integer; out AName: string; out AValue: Variant): Boolean; virtual;
abstract;
```

C++

```
virtual __fastcall Boolean Get(int Index, AnsiString AName, Variant AValue) = 0;
```

Description

Get provides the underlying implementation of all methods that extract information from the named Variants (see page 24824) list based on item position. For example, the property read methods for the Names (see page 24822) and Variants (see page 24824) properties both call Get to extract the relevant information from the list.

Index is the index of the name and Variant to read.

AName returns the name stored at the specified position.

AValue returns the Variant value stored at the specified position.

Get returns true if Index is the index of an item in the list, false if there is no name/Variant pair at the specified position.

TAbstractNamedVariants (see page 24808) declares Get as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Names (see page 24822)

Variants (see page 24824)

Put (see page 24820)

178.1.2.3.14 TAbstractNamedVariants.GetCapacity Method

Returns the value of the Capacity (see page 24821) property.

Pascal

```
function GetCapacity: Integer; virtual;
```

C++

```
virtual __fastcall int GetCapacity();
```

Description

GetCapacity is the protected read implementation of the Capacity (see page 24821) property.

In TAbstractNamedVariants (see page 24808), GetCapacity always returns the value of the Count (see page 24822) property, because TAbstractNamedVariants (see page 24808) does not implement support for preallocating memory to hold name/Variant pairs that have not been added to the list. Descendant classes that provide support for allocating memory before adding items to the list override GetCapacity to reflect the capacity of the list.

See Also

Capacity (see page 24821)

Count (see page 24822)

178.1.2.3.15 TAbstractNamedVariants.GetCount Method

Returns the value of the Count (see page 24822) property.

Pascal

```
function GetCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetCount() = 0;
```

Description

GetCapacity (see page 24817) is the protected read implementation of the Count (see page 24822) property.

TAbstractNamedVariants (see page 24808) declares GetCount as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation for the Count (see page 24822) property.

See Also

Count (see page 24822)

178.1.2.3.16 TAbstractNamedVariants.IndexOfName Method

Returns the index of the first item in the list with a specified name.

Pascal

```
function IndexOfName(const Name: string): Integer; virtual;
```

C++

```
virtual __fastcall int IndexOfName(const AnsiString Name);
```

Description

Call IndexOfName to determine where a name appears in the list, if it appears at all.

Name is the name to locate in the list.

If Name appears as the name part of a name/Variant pair in the list, IndexOfName returns the index of the first item with that

name. If there are no items with the specified name, IndexOfName returns –1.

Note: In TAbstractNamedVariants (see page 24808), IndexOfName uses a case-insensitive comparison to locate the name. In descendant classes that treat strings case sensitively, IndexOfName makes a case sensitive comparison.

See Also

Names (see page 24822)

178.1.2.3.17 TAbstractNamedVariants.Insert Method

Inserts a name/Variant pair at a specified position.

Pascal

```
procedure Insert(Index: Integer; const S: string; const AValue: Variant); virtual; abstract;
```

C++

```
virtual __fastcall Insert(int Index, const AnsiString S, const Variant AValue) = 0;
```

Description

Call Insert to add a name/Variant pair to the list at a specified position. After a call to Insert, any name/Variant pairs that appeared at or after the specified position move down by one (getting a new index).

Index is the position where the new name and Variant should be added.

S is the name of the new entry.

AValue is the Variant value of the new entry.

TAbstractNamedVariants (see page 24808) declares Insert as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Values (see page 24823)

Append (see page 24812)

Add (see page 24812)

Delete (see page 24815)

Count (see page 24822)

178.1.2.3.18 TAbstractNamedVariants.Move Method

Moves a name/Variant pair to a new position in the list.

Pascal

```
procedure Move(CurIndex: Integer; NewIndex: Integer); virtual;
```

C++

```
virtual __fastcall Move(int CurIndex, int NewIndex);
```

Description

Call move to reposition an item in the list. Move removes the item from its current position and inserts it in the new position, readjusting the positions (indexes) of other items as necessary.

CurIndex is the current position of the item to move.

NewIndex is the position to which the item is moved.

See Also

Exchange (see page 24817)

178.1.2.3.19 TAbstractNamedVariants.Put Method

Stores a name/Variant pair at a specified position in the list.

Pascal

```
procedure Put(Index: Integer; const AName: string; const AValue: Variant); virtual; abstract;
```

C++

```
virtual __fastcall Put(int Index, const AnsiString AName, const Variant AValue) = 0;
```

Description

Put provides the underlying implementation of all methods that change the information stored in the named Variants (see page 24824) list based on item position. For example, the property write methods for the Names (see page 24822) and Variants (see page 24824) properties both call Put to assign new values to items in the list.

Index is the index of the name and Variant to write.

AName is the name to store at the specified position.

AValue is the Variant value to store at the specified position.

TAbstractNamedVariants (see page 24808) declares Put as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Names (see page 24822)

Variants (see page 24824)

Get (see page 24817)

178.1.2.3.20 TAbstractNamedVariants.SetCapacity Method

Sets the value of the Capacity (see page 24821) property.

Pascal

```
procedure SetCapacity(NewCapacity: Integer); virtual;
```

C++

```
virtual __fastcall SetCapacity(int NewCapacity);
```

Description

SetCapacity is the protected write implementation of the Capacity (see page 24821) property.

In TAbstractNamedVariants (see page 24808), SetCapacity does nothing, because TAbstractNamedVariants (see page 24808) does not implement support for preallocating memory to hold name/Variant pairs that have not been added to the list. Descendant classes that provide support for allocating memory before adding items to the list override SetCapacity to reallocate the memory used to hold list items.

See Also

Capacity (see page 24821)

GetCapacity (see page 24817)

178.1.2.3.21 TAbstractNamedVariants.SetUpdateState Method

Responds when an update operation begins or ends.

Pascal

```
procedure SetUpdateState(Updating: Boolean); virtual;
```

C++

```
virtual __fastcall SetUpdateState(Boolean Updating);
```

Description

The BeginUpdate (see page 24813) and EndUpdate (see page 24815) methods call SetUpdate state when an update operation first starts or finally stops. If the calls to BeginUpdate (see page 24813) and EndUpdate (see page 24815) are nested, SetUpdate is only called by the first BeginUpdate (see page 24813) call and the last EndUpdate (see page 24815) call.

In TAbstractNamedVariants (see page 24808), SetUpdateState does nothing. Descendant classes can override this method to respond in some way (such as enabling or suppressing repaints of a control that displays the values in the list) when an update operation begins or ends.

See Also

BeginUpdate (see page 24813)

EndUpdate (see page 24815)

178.1.2.4 TAbstractNamedVariants Properties

The properties of the TAbstractNamedVariants class are listed here.

Protected Properties

	Name	Description
R	UpdateCount (see page 24823)	Indicates the number of calls to BeginUpdate (see page 24813) that have not been matched by a corresponding call to EndUpdate (see page 24815).

Public Properties

	Name	Description
	Capacity (see page 24821)	Specifies the allocated size of the named variants list.
R	Count (see page 24822)	Indicates the number of name/value pairs in the list.
R	Names (see page 24822)	Lists the names of all the Variants (see page 24824) in the list.
	Values (see page 24823)	Lists the Variants (see page 24824) in the list by name.
	Variants (see page 24824)	Lists the Variants in the list by position.

178.1.2.4.1 TAbstractNamedVariants.Capacity Property

Specifies the allocated size of the named variants list.

Pascal

```
property Capacity: Integer;
```

C++

```
__property int Capacity;
```

Description

Set Capacity to the number of named Variants (see page 24824) the list will need to contain. When a value is added to a list that is already filled to capacity, the Capacity property is automatically increased. Setting Capacity before adding values can reduce the number of memory reallocations and thereby improve performance. In addition, in descendants that allocate memory in chunks, setting Capacity can ensure that you do not allocate more memory for the list than your application needs.

Read Capacity to learn number of values the list can hold without reallocating memory. Do not confuse Capacity with the Count (see page 24822) property, which is the number of entries in the list that are in use. The value of Capacity is always greater than or equal to the value of Count (see page 24822). When Capacity is greater than Count (see page 24822), the unused memory can be reclaimed by setting Capacity to Count (see page 24822).

See Also

Count (see page 24822)

Insert (see page 24819)

178.1.2.4.2 TAbstractNamedVariants.Count Property

Indicates the number of name/value pairs in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read Count to determine the number of values the list contains. Count acts as an upper bound to the index that you can use for the Names (see page 24822) and Variants (see page 24824) properties.

See Also

Capacity (see page 24821)

Names (see page 24822)

Variants (see page 24824)

178.1.2.4.3 TAbstractNamedVariants.Names Property

Lists the names of all the Variants (see page 24824) in the list.

Pascal

```
property Names [Index: Integer]: string;
```

C++

```
__property AnsiString Names[int Index];
```

Description

Read Names to obtain the name of a Variant in the list.

Index is the position of the name (and its corresponding Variant) in the named Variants (see page 24824) list. The value of Index falls in the range from 0 to Count (see page 24822) – 1.

See Also

Count (see page 24822)

Values (see page 24823)

Variants (see page 24824)

178.1.2.4.4 TAbstractNamedVariants.UpdateCount Property

Indicates the number of calls to BeginUpdate (see page 24813) that have not been matched by a corresponding call to EndUpdate (see page 24815).

Pascal

```
property UpdateCount: Integer;
```

C++

```
__property int UpdateCount;
```

Description

UpdateCount keeps track of whether the named Variants (see page 24824) list is in the middle of an update operation (an operation that makes several changes to the list). Every time the application calls BeginUpdate (see page 24813), the value of UpdateCount is incremented. Every call to EndUpdate (see page 24815) decrements the value of UpdateCount.

When UpdateCount changes from 0 to 1, and when it changes from 1 to 0, TAbstractNamedVariants (see page 24808) calls the SetUpdateState (see page 24821) method. This allows TAbstractNamedVariants (see page 24808) descendants to respond in some way (such as enabling or suppressing repaints of a control that displays the values in the list) when an update operation begins or ends.

See Also

BeginUpdate (see page 24813)

EndUpdate (see page 24815)

SetUpdateState (see page 24821)

178.1.2.4.5 TAbstractNamedVariants.Values Property

Lists the Variants (see page 24824) in the list by name.

Pascal

```
property Values [const Name: string]: Variant;
```

C++

```
__property Variant Values[AnsiString const Name];
```

Description

Read Values to look up the Variant that is stored under a particular name. Set Values to change the Variant associated with a name, or to add a new name and Variant to the list.

Name is the name of the Variant. If there is no Variant in the list with the specified name, reading Values gives an unassigned Variant, and setting Values adds a new name and Variant to the list.

See Also

Count (see page 24822)

Values

Variants (see page 24824)

178.1.2.4.6 TAbstractNamedVariants.Variants Property

Lists the Variants in the list by position.

Pascal

```
property Variants [Index: Integer]: Variant;
```

C++

```
__property Variant Variants[int Index];
```

Description

Read Variants to look up the Variant that is stored at a particular position. Set Variants to change the Variant stored in a particular position.

Index is the position of the Variant in the named Variants list. The value of Index falls in the range from 0 to Count (see page 24822) – 1. If you specify a value of Index that is out of range, TAbstractNamedVariants (see page 24808) raises an ENamedVariantsError (see page 24808) exception.

See Also

Count (see page 24822)

Values (see page 24823)

Names (see page 24822)

ENamedVariantsError (see page 24808)

178.1.3 TNamedVariantsList Class

TNamedVariantsList is a persistent list of name/Variant pairs.

Class Hierarchy

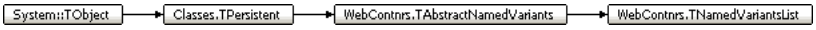

Pascal

```
TNamedVariantsList = class(TAbstractNamedVariants);
```

C++

```
class TNamedVariantsList : public TAbstractNamedVariants;
```

File

WebContnrs

Description

Use TNamedVariantsList to represent a set of name/value pairs where the values are all Variants (see page 23567). Because TNamedVariantsList is a descendant of TPersistent, it can be used as the type of a published property on a component.

TNamedVariantsList has properties and methods that let you:

Look up Variant values by name or by index.

Locate a name in the list.

Add (see page 24812) or delete name/Variant pairs.

Rearrange the items in the list.

Compare this list to another named variants list.

Note: TNamedVariantsList treats the names it stores in a case-insensitive manner. When locating a name in the list or assigning a Variant value to an existing name, TNamedVariantsList treats names that differ only by case as the same.

See Also

TStringList (see page 3006)

THashedStringList (see page 15450)

TCollection (see page 2731)

TList (see page 2868)

TInterfaceList (see page 2855)

178.1.3.1 TNamedVariantsList Members

The following tables list the members exposed by TNamedVariantsList.

Protected Methods

	Name	Description
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
V	CompareStrings (see page 24814)	Compares two strings.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 24814)	Allows the list of names and their corresponding Variant values to be saved with the named Variants (see page 24824) list.
	Error (see page 24816)	Raises an ENamedVariantsError (see page 24808) exception.
V	Get (see page 24828)	Retrieves the name and Variant stored at a specified position in the list.
V	GetCapacity (see page 24828)	Returns the value of the Capacity property.
V	GetCount (see page 24829)	Returns the value of the Count property.
	GetOwner (see page 2932)	Returns the owner of an object.
V	Put (see page 24830)	Stores a name/Variant pair at a specified position in the list.
V	SetCapacity (see page 24830)	Sets the value of the Capacity property.
V	SetUpdateState (see page 24821)	Responds when an update operation begins or ends.

Protected Properties

	Name	Description
R	UpdateCount (see page 24823)	Indicates the number of calls to BeginUpdate (see page 24813) that have not been matched by a corresponding call to EndUpdate (see page 24815).

Public Methods

	Name	Description
V	Add (see page 24812)	Adds a named Variant to the end of the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Append (see page 24812)	Adds a named Variant to the end of the list.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	Assign (see page 24813)	Copies the list of named Variants (see page 24824) from another object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeginUpdate (see page 24813)	Signals the start of an update operation.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.

	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 24827)	Deletes all the name/Variant pairs from the list.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Delete (see page 24827)	Removes a specified Variant specified by the Index parameter.
V	Destroy (see page 24826)	Frees an instance of TNamedVariantsList (see page 24824).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndUpdate (see page 24815)	Signals the end of an update operation.
	Equals (see page 24816)	Compares this named Variants (see page 24824) list with another one.
V	Exchange (see page 24817)	Swaps the positions of two items in the list.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	IndexOfName (see page 24818)	Returns the index of the first item in the list with a specified name.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
V	Insert (see page 24829)	Inserts a name/Variant pair at a specified position.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	Move (see page 24819)	Moves a name/Variant pair to a new position in the list.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 24821)	Specifies the allocated size of the named variants list.
R	Count (see page 24822)	Indicates the number of name/value pairs in the list.
R	Names (see page 24822)	Lists the names of all the Variants (see page 24824) in the list.
	Values (see page 24823)	Lists the Variants (see page 24824) in the list by name.
	Variants (see page 24824)	Lists the Variants in the list by position.

178.1.3.2 TNamedVariantsList.Destroy Destructor

Frees an instance of TNamedVariantsList (see page 24824).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TNamedVariantsList();
```

Description

Do not call the TNamedVariantsList (see page 24824) destructor directly. Instead, use the Free method, which first checks

whether the reference to the object is nil before calling Destroy.

See Also

Free (see page 22300)

178.1.3.3 TNamedVariantsList Methods

The methods of the TNamedVariantsList class are listed here.

Protected Methods

	Name	Description
	Get (see page 24828)	Retrieves the name and Variant stored at a specified position in the list.
	GetCapacity (see page 24828)	Returns the value of the Capacity property.
	GetCount (see page 24829)	Returns the value of the Count property.
	Put (see page 24830)	Stores a name/Variant pair at a specified position in the list.
	SetCapacity (see page 24830)	Sets the value of the Capacity property.

Public Methods

	Name	Description
	Clear (see page 24827)	Deletes all the name/Variant pairs from the list.
	Delete (see page 24827)	Removes a specified Variant specified by the Index parameter.
	Insert (see page 24829)	Inserts a name/Variant pair at a specified position.

178.1.3.3.1 TNamedVariantsList.Clear Method

Deletes all the name/Variant pairs from the list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call Clear to empty the list. Clear frees the memory that the Variants (see page 23567) in the list use to store their values as well as the memory used to store the list itself.

After a call to Clear, the value of the Count (see page 24822) property is zero.

See Also

Delete (see page 24827)

Variants (see page 24824)

Count

178.1.3.3.2 TNamedVariantsList.Delete Method

Removes a specified Variant specified by the Index parameter.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Call Delete to remove a specified name/Variant pair from the list. Delete frees the Variant when it removes the item from the list.

After a call to Delete, any name/Variant pairs that appeared after the deleted item in the list move up by one (getting a new index).

Index is the position of the item to delete.

See Also

Clear (see page 24827)

IndexOfName (see page 24818)

178.1.3.3.3 TNamedVariantsList.Get Method

Retrieves the name and Variant stored at a specified position in the list.

Pascal

```
function Get(Index: Integer; out AName: string; out AValue: Variant): Boolean; override;
```

C++

```
virtual __fastcall Boolean Get(int Index, AnsiString AName, Variant AValue);
```

Description

Get provides the underlying implementation of all methods that extract information from the named Variants list based on item position. For example, the property read methods for the Names and Variants properties both call Get to extract the relevant information from the list.

Index is the index of the name and Variant to read.

AName returns the name stored at the specified position.

AValue returns the Variant value stored at the specified position.

Get returns true if Index is the index of an item in the list, false if there is no name/Variant pair at the specified position.

TAbstractNamedVariants declares Get as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Names (see page 24822)

Variants (see page 24824)

Put (see page 24820)

178.1.3.3.4 TNamedVariantsList.GetCapacity Method

Returns the value of the Capacity property.

Pascal

```
function GetCapacity: Integer; override;
```

C++

```
virtual __fastcall int GetCapacity();
```

Description

GetCapacity is the protected read implementation of the Capacity property.

In TAbstractNamedVariants, GetCapacity always returns the value of the Count property, because TAbstractNamedVariants does not implement support for preallocating memory to hold name/Variant pairs that have not been added to the list. Descendant classes that provide support for allocating memory before adding items to the list override GetCapacity to reflect the capacity of the list.

See Also

Capacity (see page 24821)

Count (see page 24822)

178.1.3.3.5 TNamedVariantsList.GetCount Method

Returns the value of the Count property.

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCapacity is the protected read implementation of the Count property.

TAbstractNamedVariants declares GetCount as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation for the Count property.

See Also

Count (see page 24822)

178.1.3.3.6 TNamedVariantsList.Insert Method

Inserts a name/Variant pair at a specified position.

Pascal

```
procedure Insert(Index: Integer; const S: string; const AValue: Variant); override;
```

C++

```
virtual __fastcall Insert(int Index, const AnsiString S, const Variant AValue);
```

Description

Call Insert to add a name/Variant pair to the list at a specified position. After a call to Insert, any name/Variant pairs that appeared at or after the specified position move down by one (getting a new index).

Index is the position where the new name and Variant should be added.

S is the name of the new entry.

AValue is the Variant value of the new entry.

See Also

Values (see page 24823)

Append (see page 24812)

Add (see page 24812)

Delete (see page 24827)

Count

178.1.3.3.7 TNamedVariantsList.Put Method

Stores a name/Variant pair at a specified position in the list.

Pascal

```
procedure Put(Index: Integer; const AName: string; const AValue: Variant); override;
```

C++

```
virtual __fastcall Put(int Index, const AnsiString AName, const Variant AValue);
```

Description

Put provides the underlying implementation of all methods that change the information stored in the named Variants list based on item position. For example, the property write methods for the Names and Variants properties both call Put to assign new values to items in the list.

Index is the index of the name and Variant to write.

AName is the name to store at the specified position.

AValue is the Variant value to store at the specified position.

TAbstractNamedVariants declares Put as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Names (see page 24822)

Variants (see page 24824)

Get (see page 24817)

178.1.3.3.8 TNamedVariantsList.SetCapacity Method

Sets the value of the Capacity property.

Pascal

```
procedure SetCapacity(NewCapacity: Integer); override;
```

C++

```
virtual __fastcall SetCapacity(int NewCapacity);
```

Description

SetCapacity is the protected write implementation of the Capacity property.

In TAbstractNamedVariants, SetCapacity does nothing, because TAbstractNamedVariants does not implement support for preallocating memory to hold name/Variant pairs that have not been added to the list. Descendant classes that provide support for allocating memory before adding items to the list override SetCapacity to reallocate the memory used to hold list items.

See Also

Capacity (see page 24821)

GetCapacity (see page 24817)

178.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TNamedVariantItem (see page 24832)	This is record WebContnrs.TNamedVariantItem.

178.2.1 WebContnrs.TNamedVariantItem Record

Pascal

```
TNamedVariantItem = record
  FString: string;
  FVariant: Variant;
end;
```

C++

```
struct TNamedVariantItem {
  AnsiString FString;
  Variant FVariant;
};
```

File

WebContnrs

Description

This is record WebContnrs.TNamedVariantItem.

178.3 Types

The following table lists types in this documentation.

Types

Name	Description
PNamedVariantItem (see page 24833)	This is type WebContnrs.PNamedVariantItem.
PNamedVariantList (see page 24833)	This is type WebContnrs.PNamedVariantList.
TNamedVariantList (see page 24833)	This is type WebContnrs.TNamedVariantList.

178.3.1 WebContnrs.PNamedVariantItem Type

Pascal

```
PNamedVariantItem = ^TNamedVariantItem;
```

C++

```
^TNamedVariantItem PNamedVariantItem;
```

File

WebContnrs

Description

This is type WebContnrs.PNamedVariantItem.

178.3.2 WebContnrs.PNamedVariantList Type

Pascal

```
PNamedVariantList = ^TNamedVariantList;
```

C++

```
^TNamedVariantList PNamedVariantList;
```

File

WebContnrs

Description

This is type WebContnrs.PNamedVariantList.

178.3.3 WebContnrs.TNamedVariantList Type

Pascal

```
TNamedVariantList = array[0..2000000000 div sizeof(TNamedVariantItem)] of TNamedVariantItem;
```

C++

```
array[0..2000000000 div sizeof(TNamedVariantItem)] of TNamedVariantItem TNamedVariantList;
```

File

WebContnrs

Description

This is type WebContnrs.TNamedVariantList.

179 WebDisp Namespace

This is namespace WebDisp.

Topics

Name	Description
Classes (see page 24838)	The following table lists classes in this documentation.
Interfaces (see page 24975)	The following table lists interfaces in this documentation.
Functions (see page 24977)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 24983)	The following table lists structs, records, enums in this documentation.
Types (see page 24984)	The following table lists types in this documentation.
Constants (see page 24991)	The following table lists constants in this documentation.

Classes

	Name	Description
	EAdapterDispatcherException (see page 24838)	eadapterdispatcherexception is the exception class for WebSnap actions and image requests that can't be serviced.
	EPageDispatcherException (see page 24839)	EPageDispatcherException raises an exception when a page dispatcher encounters an error.
	ESessionException (see page 24840)	ESessionException is the exception that is raised when a WebSnap application tries to use an invalid session.
	TAdapterDispatcher (see page 24840)	TAdapterDispatcher automatically handles HTML form submissions and requests for dynamic images by calling adapter action and field components.
	TAdapterDispatchParams (see page 24848)	TAdapterDispatchParams parses a request to determine the adapter to which it is directed, and, if available, the Web pages that should appear when the request is successfully or unsuccessfully handled.
	TAdapterItemRequestParams (see page 24855)	TAdapterItemRequestParams holds the name/value pairs that must be included in request messages addressed to an adapter or its subcomponents.
	TBaseRequestHandler (see page 24860)	The TBaseRequestHandler component is a base class for WebSnap dispatcher components.
	TCustomAdapterDispatcher (see page 24865)	TCustomAdapterDispatcher automatically handles HTML form submissions and requests for dynamic images by calling adapter action and field components.
	TCustomLocateFileService (see page 24880)	TCustomLocateFileService uses event handlers to control access to templates and server-side script include files at runtime.
	TCustomPageDispatcher (see page 24889)	Dispatches incoming HTTP request messages in a WebSnap application.
	TCustomWebAppComponents (see page 24906)	TCustomWebAppComponents manages the components that provide a WebSnap application's functionality.
	TInvalidWebSession (see page 24923)	Serves as a placeholder when an HTTP request could not be given a session.
	TLocateFileService (see page 24928)	TLocateFileService controls the access of templates and server-side script include files at runtime.

	TMultiModuleWebAppServices (see page 24933)	TMultiModuleWebAppServices coordinates the response to HTTP request messages by locating the appropriate components to handle various tasks.
	TPageDispatcher (see page 24949)	TPageDispatcher dispatches incoming HTTP request messages in a WebSnap application.
	TWebAppComponents (see page 24958)	TWebAppComponents holds the application components that provide the WebSnap application's functionality.
	TWebSession (see page 24968)	TWebSession stores temporary information associated with the user in a WebSnap application.

Constants

Name	Description
sFailurePageParamName (see page 24991)	This is constant WebDisp.sFailurePageParamName.
sPageParamName (see page 24991)	This is constant WebDisp.sPageParamName.
sRequestIDParamName (see page 24992)	This is constant WebDisp.sRequestIDParamName.
sSessionCookieName (see page 24992)	This is constant WebDisp.sSessionCookieName.
sSuccessPageParamName (see page 24992)	This is constant WebDisp.sSuccessPageParamName.

Functions

	Name	Description
	AdapterRequestParamsAsStrings (see page 24977)	This is function WebDisp.AdapterRequestParamsAsStrings.
	DispatchPageName (see page 24977)	This is function WebDisp.DispatchPageName.
	EncodeParamNameValues (see page 24978)	This is function WebDisp.EncodeParamNameValues.
	EncodeParamsAsHTTPQuery (see page 24978)	This is function WebDisp.EncodeParamsAsHTTPQuery.
	ExpandScriptException (see page 24978)	This is function WebDisp.ExpandScriptException.
	ExtractParamNameValues (see page 24979)	This is function WebDisp.ExtractParamNameValues.
	FindDefaultAppPageName (see page 24979)	This is function WebDisp.FindDefaultAppPageName.
	PageNameOfRequest (see page 24979)	This is function WebDisp.PageNameOfRequest.
	RaiseDispatchBlankPageNameException (see page 24980)	This is function WebDisp.RaiseDispatchBlankPageNameException.
	RaisePageDoesNotProvideContent (see page 24980)	This is function WebDisp.RaisePageDoesNotProvideContent.
	RaisePageNotFound (see page 24980)	This is function WebDisp.RaisePageNotFound.
	RedirectToPageName (see page 24981)	This is function WebDisp.RedirectToPageName.
	RegisterAdapterRequestIdentifier (see page 24981)	This is function WebDisp.RegisterAdapterRequestIdentifier.
	SafeRedirect (see page 24981)	Redirects a browser to another specified page.

Interfaces

	Name	Description
	IWebSessionAccess (see page 24975)	This is class WebDisp.IWebSessionAccess.

Structs, Records, Enums

	Name	Description
	TPageAccessDenied (see page 24983)	This is record WebDisp.TPageAccessDenied.

Types

Name	Description
TActivateWebModuleEvent (see page 24984)	This is type WebDisp.TActivateWebModuleEvent.
TCanViewPageEvent (see page 24984)	This is type WebDisp.TCanViewPageEvent.
TDispatchActionEvent (see page 24985)	This is type WebDisp.TDispatchActionEvent.
TDispatchActionHandledEvent (see page 24985)	This is type WebDisp.TDispatchActionHandledEvent.
TDispatchImageEvent (see page 24985)	This is type WebDisp.TDispatchImageEvent.
TDispatchImageHandledEvent (see page 24986)	This is type WebDisp.TDispatchImageHandledEvent.
TDispatchPageEvent (see page 24986)	This is type WebDisp.TDispatchPageEvent.
TDispatchPageHandledEvent (see page 24986)	This is type WebDisp.TDispatchPageHandledEvent.
TDispatchPageParamsHandledEvent (see page 24987)	This is type WebDisp.TDispatchPageParamsHandledEvent.
TGetAdapterRequestParamsEvent (see page 24987)	This is type WebDisp.TGetAdapterRequestParamsEvent.
TLocateFileServiceFindFileEvent (see page 24987)	This is type WebDisp.TLocateFileServiceFindFileEvent.
TLocateFileServiceFindStreamEvent (see page 24988)	This is type WebDisp.TLocateFileServiceFindStreamEvent.

TLoginPagedHandledEvent (see page 24988)	This is type WebDisp.TLoginPagedHandledEvent.
TLoginRequiredEvent (see page 24988)	This is type WebDisp.TLoginRequiredEvent.
TPageAccessDeniedEvent (see page 24989)	This is type WebDisp.TPageAccessDeniedEvent.
TWebRequest (see page 24989)	This is type WebDisp.TWebRequest.
TWebResponse (see page 24989)	This is type WebDisp.TWebResponse.

179.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EAdapterDispatcherException (see page 24838)	eadapterdispatcherexception is the exception class for WebSnap actions and image requests that can't be serviced.
	EPageDispatcherException (see page 24839)	EPageDispatcherException raises an exception when a page dispatcher encounters an error.
	ESessionException (see page 24840)	ESessionException is the exception that is raised when a WebSnap application tries to use an invalid session.
	TAdapterDispatcher (see page 24840)	TAdapterDispatcher automatically handles HTML form submissions and requests for dynamic images by calling adapter action and field components.
	TAdapterDispatchParams (see page 24848)	TAdapterDispatchParams parses a request to determine the adapter to which it is directed, and, if available, the Web pages that should appear when the request is successfully or unsuccessfully handled.
	TAdapterItemRequestParams (see page 24855)	TAdapterItemRequestParams holds the name/value pairs that must be included in request messages addressed to an adapter or its subcomponents.
	TBaseRequestHandler (see page 24860)	The TBaseRequestHandler component is a base class for WebSnap dispatcher components.
	TCustomAdapterDispatcher (see page 24865)	TCustomAdapterDispatcher automatically handles HTML form submissions and requests for dynamic images by calling adapter action and field components.
	TCustomLocateFileService (see page 24880)	TCustomLocateFileService uses event handlers to control access to templates and server-side script include files at runtime.
	TCustomPageDispatcher (see page 24889)	Dispatches incoming HTTP request messages in a WebSnap application.
	TCustomWebAppComponents (see page 24906)	TCustomWebAppComponents manages the components that provide a WebSnap application's functionality.
	TInvalidWebSession (see page 24923)	Serves as a placeholder when an HTTP request could not be given a session.
	TLocateFileService (see page 24928)	TLocateFileService controls the access of templates and server-side script include files at runtime.
	TMultiModuleWebAppServices (see page 24933)	TMultiModuleWebAppServices coordinates the response to HTTP request messages by locating the appropriate components to handle various tasks.
	TPageDispatcher (see page 24949)	TPageDispatcher dispatches incoming HTTP request messages in a WebSnap application.
	TWebAppComponents (see page 24958)	TWebAppComponents holds the application components that provide the WebSnap application's functionality.
	TWebSession (see page 24968)	TWebSession stores temporary information associated with the user in a WebSnap application.

179.1.1 EAdapterDispatcherException Class

eadapterdispatcherexception is the exception class for WebSnap actions and image requests that can't be serviced.

Class Hierarchy

Pascal

```
EAdapterDispatcherException = class(EWebBrokerException);
```

C++

```
class EAdapterDispatcherException : public EWebBrokerException;
```

File

WebDisp

Description

When the TAdapterDispatcher (see page 24840) component can't successfully dispatch an action or obtain a dynamic image in response to a request, it generates an OnActionRequestNotHandled or an OnImageRequestNotHandled event. If there is no event handler for that event, or if the event handler does not handle the situation, TAdapterDispatcher (see page 24840) raises an eadapterdispatcherexception exception.

See Also

TCustomAdapterDispatcher (see page 24865)

TCustomAdapterDispatcher (see page 24865)

179.1.2 EPageDispatcherException Class

EPageDispatcherException raises an exception when a page dispatcher encounters an error.

Class Hierarchy

Pascal

```
EPageDispatcherException = class(EWebBrokerException);
```

C++

```
class EPageDispatcherException : public EWebBrokerException;
```

File

WebDisp

Description

EPageDispatcherException represents exceptions that occur when a page dispatcher tries to obtain the content of a Web page in response to an incoming HTTP request. These exceptions include

The incoming request does not request a specific Web page and the application does not provide a default Web page to handle this situation.

The current user does not have access rights for the requested page.

The page dispatcher could not locate a Web page module to provide the requested page.

The Web module with the same name as the requested page can't produce a Web page.

The application is attempting to send a response that redirects the user to a page that the dispatcher can't locate.

See Also

TPageDispatcher (see page 24949)

179.1.3 ESessionException Class

ESessionException is the exception that is raised when a WebSnap application tries to use an invalid session.

Class Hierarchy

Pascal

```
ESessionException = class(EWebBrokerException);
```

C++

```
class ESessionException : public EWebBrokerException;
```

File

WebDisp

Description

The TInvalidWebSession (see page 24923) class raises ESessionException when the application tries to use the current session and that session could not be created. This includes attempts to change the properties of the current session (such as setting the time-out period), attempts to get or set values, or attempts to terminate the current session.

See Also

TInvalidWebSession (see page 24923)

179.1.4 TAdapterDispatcher Class

TAdapterDispatcher automatically handles HTML form submissions and requests for dynamic images by calling adapter action and field components.

Class Hierarchy

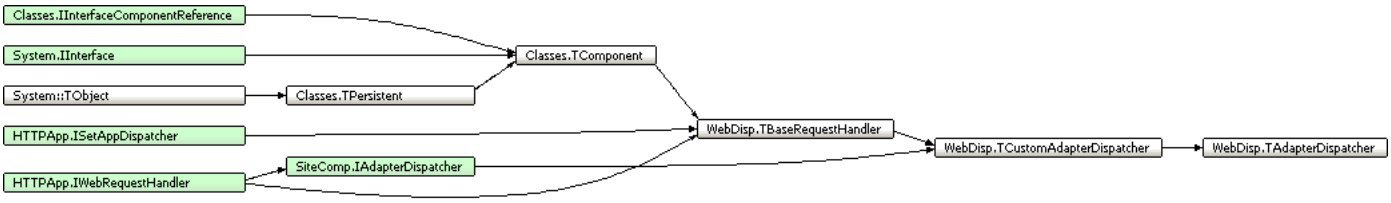

Pascal

```
TAdapterDispatcher = class(TCustomAdapterDispatcher);
```

C++

```
class TAdapterDispatcher : public TCustomAdapterDispatcher;
```

File

WebDisp

Description

TAdapterDispatcher is one of the default application components that is added when creating a WebSnap application. Events

are triggered before and after dispatching an action or image, when adapter parameters are requested, and in cases when the action or image request was not handled.

Note: Your web application will be unable to execute adapter actions or obtain images from adapter fields unless it includes a TAdapterDispatcher component.

See Also

TAdapter (see page 23713)

TCustomWebAppComponents (see page 24906)

179.1.4.1 TAdapterDispatcher Members

The following tables list the members exposed by TAdapterDispatcher.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddSpecialParams (see page 24870)	Adds parameters for the success and failure Web pages to the list of adapter parameters.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoAdapterRequestNotHandled (see page 24870)	Generates an OnActionRequestNotHandled (see page 24877) and an OnImageRequestNotHandled (see page 24879) event.
	DoAfterDispatchAdapterRequest (see page 24871)	Generates an OnAfterDispatchAction (see page 24877) and an OnAfterDispatchImage (see page 24878) event.
	DoBeforeDispatchAdapterRequest (see page 24871)	Generates an OnBeforeDispatchAction (see page 24878) and an OnBeforeDispatchImage (see page 24878) event.
	EncodeAdapterItemRequestAsFieldValue (see page 24872)	Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.
	EncodeAdapterItemRequestAsHREF (see page 24872)	Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.
	GetAdapterItemRequestParams (see page 24873)	Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDispatchParams (see page 24873)	Parses a request to determine the adapter to which it is directed.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleRequest (see page 24864)	Handles an HTTP request.

	ImplEncodeAdapterItemRequestAsFieldValue (see page 24874)	Provides the underlying implementation for the EncodeAdapterItemRequestAsFieldValue (see page 24872) method.
	ImplEncodeAdapterItemRequestAsHREF (see page 24874)	Provides the underlying implementation for the EncodeAdapterItemRequestAsHREF (see page 24872) method.
	ImplGetAdapterItemRequestParams (see page 24875)	Provides the underlying implementation for the GetAdapterItemRequestParams (see page 24873) method.
	ImplHandleRequest (see page 24864)	Handles application-specific processing of an HTTP request.
	ImplHandleRequest (see page 24876)	Provides the underlying implementation for the HandleRequest (see page 24864) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 24864)	Sets the AppDispatcher (see page 24865) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	AppDispatcher (see page 24865)	Provides read-only access to the application dispatcher component.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EncodeAdapterItemRequestAsFieldValue (see page 19654)	Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.
	EncodeAdapterItemRequestAsHREF (see page 19654)	Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterItemRequestParams (see page 19655)	Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandleRequest (see page 15238)	Processes an HTTP request message.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnActionRequestNotHandled (see page 24844)	Occurs when the WebSnap application could not execute a requested action.
	OnAfterDispatchAction (see page 24845)	Occurs after the target adapter has finished executing an action.
	OnAfterDispatchImage (see page 24845)	Occurs after the target adapter has finished handling an image request.
	OnBeforeDispatchAction (see page 24846)	Occurs before the dispatcher forwards an action to an adapter for execution.
	OnBeforeDispatchImage (see page 24846)	Occurs before the dispatcher forwards an image request to an adapter for handling.
	OnGetAdapterRequestParams (see page 24847)	Occurs when the adapter dispatcher needs to obtain the parameters for executing an action or retrieving an image.
	OnImageRequestNotHandled (see page 24847)	Occurs when the WebSnap application could not provide a requested image.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

179.1.4.2 TAdapterDispatcher Events

The events of the TAdapterDispatcher class are listed here.

Published Events

	Name	Description
	OnActionRequestNotHandled (see page 24844)	Occurs when the WebSnap application could not execute a requested action.
	OnAfterDispatchAction (see page 24845)	Occurs after the target adapter has finished executing an action.
	OnAfterDispatchImage (see page 24845)	Occurs after the target adapter has finished handling an image request.
	OnBeforeDispatchAction (see page 24846)	Occurs before the dispatcher forwards an action to an adapter for execution.
	OnBeforeDispatchImage (see page 24846)	Occurs before the dispatcher forwards an image request to an adapter for handling.
	OnGetAdapterRequestParams (see page 24847)	Occurs when the adapter dispatcher needs to obtain the parameters for executing an action or retrieving an image.
	OnImageRequestNotHandled (see page 24847)	Occurs when the WebSnap application could not provide a requested image.

179.1.4.2.1 TAdapterDispatcher.OnActionRequestNotHandled Event

Occurs when the WebSnap application could not execute a requested action.

Pascal

```
property OnActionRequestNotHandled: TDispatchActionHandledEvent;
```

C++

```
__property TDispatchActionHandledEvent OnActionRequestNotHandled;
```

Description

Write an OnActionRequestNotHandled event handler to provide a response to actions that the adapter components in the application can't execute. This may occur because the adapter dispatcher could not find an adapter to handle the action, or it may occur because the identified adapter was unable to execute the action.

If the OnActionRequestNotHandled event handler does not set its Handled parameter to true, the adapter dispatcher generates an OnImageRequestNotHandled. If the OnImageRequestNotHandled event handler also does not handle the event, the dispatcher raises an EAdapterDispatcherException exception.

Tip: Use the Response property of the current Web context to access the response message from the OnActionRequestNotHandled event handler. You can obtain a reference to the Web context by calling the global WebContext function.

See Also

OnAfterDispatchAction (see page 24877)

OnBeforeDispatchAction (see page 24878)

EAdapterDispatcherException (see page 24838)

WebContext (see page 24687)

Response (see page 24659)

OnImageRequestNotHandled (see page 24879)

179.1.4.2.2 TAdapterDispatcher.OnAfterDispatchAction Event

Occurs after the target adapter has finished executing an action.

Pascal

```
property OnAfterDispatchAction: TDispatchActionEvent;
```

C++

```
__property TDispatchActionEvent OnAfterDispatchAction;
```

Description

Write an OnAfterDispatchAction event handler to take specific action after an adapter has executed an action. Note that this event can occur even if the request does not request the execution of an action or if the adapter was not successful in its attempt to execute the action.

See Also

OnActionRequestNotHandled (see page 24877)

OnBeforeDispatchAction (see page 24878)

179.1.4.2.3 TAdapterDispatcher.OnAfterDispatchImage Event

Occurs after the target adapter has finished handling an image request.

Pascal

```
property OnAfterDispatchImage: TDispatchImageEvent;
```

C++

```
__property TDispatchImageEvent OnAfterDispatchImage;
```

Description

Write an OnAfterDispatchImage event handler to take specific action after an adapter has responded to an image request. Note that this event can occur even if the request does not request the value of an image field or if the adapter was not successful in its attempt to honor that request.

See Also

OnImageRequestNotHandled (see page 24879)

OnBeforeDispatchImage (see page 24878)

179.1.4.2.4 TAdapterDispatcher.OnBeforeDispatchAction Event

Occurs before the dispatcher forwards an action to an adapter for execution.

Pascal

```
property OnBeforeDispatchAction: TDispatchActionHandledEvent;
```

C++

```
__property TDispatchActionHandledEvent OnBeforeDispatchAction;
```

Description

Write an OnBeforeDispatchAction event handler to respond to action requests before they are forwarded to the associated adapter for execution. Note that if the request does not request the execution of an action, the Params parameter of the event handler is nil (Delphi) or NULL (C++).

If the OnBeforeDispatchAction event handler does not set its Handled parameter to true, the adapter dispatcher forwards the request to the identified adapter that is the value of the Action parameter. If the event handler sets Handled to false, the dispatcher does not forward the request to the adapter for execution, and does not generate an OnAfterDispatchAction event.

Tip: Use the Response property of the current Web context to access the response message from the OnBeforeDispatchAction event handler. You can obtain a reference to the Web context by calling the global WebContext function.

See Also

OnActionRequestNotHandled (see page 24877)

OnAfterDispatchAction (see page 24877)

WebContext (see page 24687)

Response (see page 24659)

179.1.4.2.5 TAdapterDispatcher.OnBeforeDispatchImage Event

Occurs before the dispatcher forwards an image request to an adapter for handling.

Pascal

```
property OnBeforeDispatchImage: TDispatchImageHandledEvent;
```

C++

```
__property TDispatchImageHandledEvent OnBeforeDispatchImage;
```

Description

Write an OnBeforeDispatchImage event handler to respond to image requests before they are forwarded to the associated adapter for handling. Note that if the request does not ask for the value of an image field, the Params parameter of the event handler is nil (Delphi) or NULL (C++).

If the OnBeforeDispatchImage event handler does not set its Handled parameter to true, the adapter dispatcher forwards the request to the identified adapter that is the value of the Image parameter. If the event handler sets Handled to false, the dispatcher does not forward the request to the adapter for handling, and does not generate an OnAfterDispatchImage event.

Tip: Use the Response property of the current Web context to access the response message from the OnBeforeDispatchImage event handler. You can obtain a reference to the Web context by calling the global WebContext function.

See Also

OnImageRequestNotHandled (see page 24879)

OnAfterDispatchImage (see page 24878)

WebContext (see page 24687)

Response (see page 24659)

179.1.4.2.6 TAdapterDispatcher.OnGetAdapterRequestParams Event

Occurs when the adapter dispatcher needs to obtain the parameters for executing an action or retrieving an image.

Pascal

```
property OnGetAdapterRequestParams: TGetAdapterRequestParamsEvent;
```

C++

```
__property TGetAdapterRequestParamsEvent OnGetAdapterRequestParams;
```

Description

Write an OnGetAdapterRequestParams event handler to modify the list of parameters for an action or image request before they are used. The OnGetAdapterRequestParams event handler can modify the current parameters or add its own parameters to the list. This event occurs both when the dispatcher formats parameters that are added to HTML forms and when it is dispatching a request message that contains an action or image request.

See Also

GetAdapterItemRequestParams (see page 24873)

EncodeAdapterItemRequestAsFieldValue (see page 24872)

EncodeAdapterItemRequestAsHREF (see page 24872)

AddSpecialParams (see page 24870)

179.1.4.2.7 TAdapterDispatcher.OnImageRequestNotHandled Event

Occurs when the WebSnap application could not provide a requested image.

Pascal

```
property OnImageRequestNotHandled: TDispatchImageHandledEvent;
```

C++

```
__property TDispatchImageHandledEvent OnImageRequestNotHandled;
```

Description

Write an OnImageRequestNotHandled event handler to provide a response to image requests that the adapter components in the application can't honor. This may occur because the adapter dispatcher could not find an adapter to handle the request, or it may occur because the identified adapter was unable to provide the requested image.

If the OnImageRequestNotHandled event handler exits with its Handled parameter set to false, the dispatcher raises an EAdapterDispatcherException exception.

Tip: Use the Response property of the current Web context to access the response message from the OnImageRequestNotHandled event handler. You can obtain a reference to the Web context by calling the global WebContext function.

See Also

OnAfterDispatchImage (see page 24878)

OnBeforeDispatchImage (see page 24878)

EAdapterDispatcherException (see page 24838)

WebContext (see page 24687)

Response (see page 24659)

OnActionRequestNotHandled (see page 24877)

179.1.5 TAdapterDispatchParams Class

TAdapterDispatchParams parses a request to determine the adapter to which it is directed, and, if available, the Web pages that should appear when the request is successfully or unsuccessfully handled.

Class Hierarchy

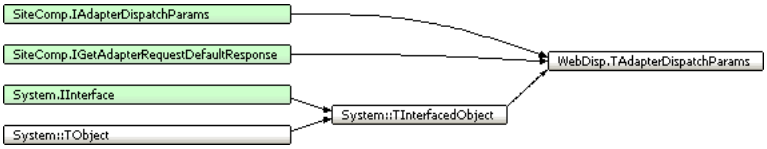

Pascal

```
TAdapterDispatchParams = class(TInterfacedObject, IAdapterDispatchParams,
IGetAdapterRequestDefaultResponse);
```

C++

```
class TAdapterDispatchParams : public TInterfacedObject, public IAdapterDispatchParams, public
IGetAdapterRequestDefaultResponse;
```

File

WebDisp

Description

TAdapterDispatcher (see page 24840) uses TAdapterDispatchParams as a helper object for locating the adapter that can handle a request message. The TAdapterDispatchParams constructor takes a request message as an argument, and parses the request to determine the adapter, if any, that should handle it and the parameters that the adapter needs to handle the request.

The TAdapterDispatchParms properties identify the adapter that can process the request, the name of the action or field that the

adapter uses to process the request, and the type of request (an action to execute or a field value to retrieve).

See Also

TAdapterDispatcher (see page 24840)

IAdapterDispatchParams (see page 19655)

179.1.5.1 TAdapterDispatchParams Members

The following tables list the members exposed by TAdapterDispatchParams.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetFailurePage (see page 24851)	This is GetFailurePage, a member of class TAdapterDispatchParams.
	GetHandler (see page 24852)	Returns the value of the Handler property.
	GetObjectIdentifier (see page 24852)	Returns the value of the ObjectIdentifier property.
	GetParams (see page 24852)	Returns the value of the Params property.
	GetRequestIdentifier (see page 24853)	Returns the value of the RequestIdentifier property.
	GetSuccessPage (see page 24853)	This is GetSuccessPage, a member of class TAdapterDispatchParams.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 24850)	Creates and initializes a TAdapterDispatchParams (see page 24848) object based on the information in an HTTP request.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 24851)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	FailurePage (see page 19694)	This is FailurePage, a member of class IGetAdapterRequestDefaultResponse.
R	Handler (see page 24853)	Indicates the adapter object that can handle the current request.
R	ObjectIdentifier (see page 24854)	Specifies the name of the action to execute or the field to retrieve.
R	Params (see page 19659)	Indicates the parameter values to use when executing the action or looking up the image.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	RequestIdentifier (see page 24854)	Identifies the type of the request identified by the ObjectIdentifier (see page 24854) property.
R	SuccessPage (see page 19694)	This is SuccessPage, a member of class IGetAdapterRequestDefaultResponse.

179.1.5.2 TAdapterDispatchParams.Create Constructor

Creates and initializes a TAdapterDispatchParams (see page 24848) object based on the information in an HTTP request.

Pascal

```
constructor Create(ARequest: TWebRequest);
```

C++

```
__fastcall TAdapterDispatchParams(TWebRequest ARequest);
```

Description

Call Create to create a new TAdapterDispatchParams (see page 24848) object and use it to obtain information encoded in an HTTP request message.

The TAdapterDispatchParams (see page 24848) constructor parses the request message that is supplied as ARequest, locating the values that identify the adapter that must be used to handle the request, and the name and type of the information that that adapter supplies. Create uses this information to initialize the Handler (see page 24853), ObjectIdentifier (see page 24854), and RequestIdentifier (see page 24854) properties.

See Also

Handler (see page 24853)

ObjectIdentifier (see page 24854)

RequestIdentifier (see page 24854)

179.1.5.3 TAdapterDispatchParams.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterDispatchParams();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

179.1.5.4 TAdapterDispatchParams Methods

The methods of the TAdapterDispatchParams class are listed here.

Protected Methods

	Name	Description
	GetFailurePage (see page 24851)	This is GetFailurePage, a member of class TAdapterDispatchParams.
	GetHandler (see page 24852)	Returns the value of the Handler property.
	GetObjectIdentifier (see page 24852)	Returns the value of the ObjectIdentifier property.
	GetParams (see page 24852)	Returns the value of the Params property.
	GetRequestIdentifier (see page 24853)	Returns the value of the RequestIdentifier property.
	GetSuccessPage (see page 24853)	This is GetSuccessPage, a member of class TAdapterDispatchParams.

179.1.5.4.1 TAdapterDispatchParams.GetFailurePage Method

Pascal

```
function GetFailurePage: string;
```

C++

```
__fastcall AnsiString GetFailurePage();
```

Description

This is GetFailurePage, a member of class TAdapterDispatchParams.

179.1.5.4.2 TAdapterDispatchParams.GetHandler Method

Returns the value of the Handler property.

Pascal

```
function GetHandler: TObject;
```

C++

```
__fastcall TObject * GetHandler();
```

Description

GetHandler is the read implementation of the Handler property.

See Also

Handler (see page 19658)

179.1.5.4.3 TAdapterDispatchParams.GetObjectIdentifier Method

Returns the value of the ObjectIdentifier property.

Pascal

```
function GetObjectIdentifier: string;
```

C++

```
__fastcall AnsiString GetObjectIdentifier();
```

Description

GetObjectIdentifier is the read implementation of the ObjectIdentifier property.

See Also

ObjectIdentifier (see page 19658)

179.1.5.4.4 TAdapterDispatchParams.GetParams Method

Returns the value of the Params property.

Pascal

```
function GetParams: TStrings;
```

C++

```
__fastcall TStrings GetParams();
```

Description

GetParams is the read implementation of the Params property.

See Also

Params (see page 19659)

179.1.5.4.5 TAdapterDispatchParams.GetRequestIdentifier Method

Returns the value of the RequestIdentifier property.

Pascal

```
function GetRequestIdentifier: string;
```

C++

```
__fastcall AnsiString GetRequestIdentifier();
```

Description

GetRequestIdentifier is the read implementation of the RequestIdentifier property.

See Also

RequestIdentifier (see page 19659)

179.1.5.4.6 TAdapterDispatchParams.GetSuccessPage Method

Pascal

```
function GetSuccessPage: string;
```

C++

```
__fastcall AnsiString GetSuccessPage();
```

Description

This is GetSuccessPage, a member of class TAdapterDispatchParams.

179.1.5.5 TAdapterDispatchParams Properties

The properties of the TAdapterDispatchParams class are listed here.

Public Properties

	Name	Description
R	Handler (see page 24853)	Indicates the adapter object that can handle the current request.
R	ObjectIdentifier (see page 24854)	Specifies the name of the action to execute or the field to retrieve.
R	RequestIdentifier (see page 24854)	Identifies the type of the request identified by the ObjectIdentifier (see page 24854) property.

179.1.5.5.1 TAdapterDispatchParams.Handler Property

Indicates the adapter object that can handle the current request.

Pascal

```
property Handler: TObject;
```

C++

```
__property TObject * Handler;
```

Description

Read Handler to access the adapter that can execute the action or supply the image that is requested. The TAdapterDispatchParams (see page 24848) constructor initializes this property based on the information it parses from the current request message.

See Also

Create (see page 24850)

ObjectIdentifier (see page 24854)

RequestIdentifier (see page 24854)

179.1.5.5.2 TAdapterDispatchParams.ObjectIdentifier Property

Specifies the name of the action to execute or the field to retrieve.

Pascal

```
property ObjectIdentifier: string;
```

C++

```
__property AnsiString ObjectIdentifier;
```

Description

Read ObjectIdentifier to determine the name of the requested action or field. The TAdapterDispatchParams (see page 24848) constructor initializes this property based on the information it parses from the current request message.

The interpretation of this value is the responsibility of the object specified by the Handler (see page 24853) parameter.

See Also

Create (see page 24850)

Handler (see page 24853)

RequestIdentifier (see page 24854)

179.1.5.5.3 TAdapterDispatchParams.RequestIdentifier Property

Identifies the type of the request identified by the ObjectIdentifier (see page 24854) property.

Pascal

```
property RequestIdentifier: string;
```

C++

```
__property AnsiString RequestIdentifier;
```

Description

Read RequestIdentifier to determine the type of processing the current request requires. For example, the RequestIdentifier may identify the request as a request for the value of an image field or a request to execute an action. The TAdapterDispatchParams (see page 24848) constructor initializes this property based on the information it parses from the current request message.

The interpretation of this value is the responsibility of the object specified by the Handler (see page 24853) parameter.

See Also

Create (see page 24850)

Handler (see page 24853)

ObjectIdentifier (see page 24854)

179.1.6 TAdapterItemRequestParams Class

TAdapterItemRequestParams holds the name/value pairs that must be included in request messages addressed to an adapter or its subcomponents.

Class Hierarchy

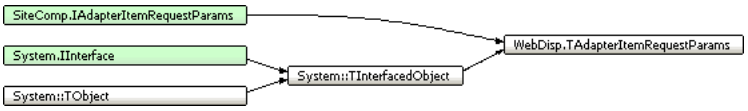

Pascal

```
TAdapterItemRequestParams = class(TInterfacedObject, IAdapterItemRequestParams);
```

C++

```
class TAdapterItemRequestParams : public TInterfacedObject, public IAdapterItemRequestParams;
```

File

WebDisp

Description

TAdapterItemRequestParams is the default implementation of the IAdapterItemRequestParams interface. It holds the name/value pairs that should be included as request parameters to an adapter or one of its subcomponents.

The parameters, which can be accessed through the ParamValues (see page 24860) property, are a set of 'Name=Value' strings. In addition to holding these parameters, TAdapterItemRequestParams has two properties, SuccessPage (see page 24860) and FailurePage (see page 24859), which identify the Web pages that should be returned to the caller when the request succeeds or fails.

See Also

TAdapterDispatcher (see page 24840)

IAdapterItemRequestParams (see page 19673)

GetAdapterItemRequestParams

179.1.6.1 TAdapterItemRequestParams Members

The following tables list the members exposed by TAdapterItemRequestParams.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetFailurePage (see page 24857)	Returns the value of the FailurePage property.
	GetParamValues (see page 24858)	Returns the value of the ParamValues property.

	GetSuccessPage (see page 24858)	Returns the value of the SuccessPage property.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetFailurePage (see page 24858)	Sets the value of the FailurePage property.
	SetSuccessPage (see page 24859)	Sets the value of the SuccessPage property.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 24857)	Creates and initializes a TAdapterItemRequestParams (see page 24855) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 24857)	Frees an instance of TAdapterItemRequestParams (see page 24855).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	FailurePage (see page 24859)	Indicates the Web page that the application should return if the current request fails.
R	ParamValues (see page 24860)	Lists the parameter values that are included in requests to the adapter action or field.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	SuccessPage (see page 24860)	Indicates the Web page that the application should return if the request succeeds.

179.1.6.2 TAdapterItemRequestParams.Create Constructor

Creates and initializes a TAdapterItemRequestParams (see page 24855) object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TAdapterItemRequestParams();
```

Description

Typically, applications have no need to create TAdapterItemRequestParams (see page 24855) instances. The adapter dispatcher creates these objects to hold the parameters for a request.

See Also

TAdapterDispatcher (see page 24840)

179.1.6.3 TAdapterItemRequestParams.Destroy Destructor

Frees an instance of TAdapterItemRequestParams (see page 24855).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterItemRequestParams();
```

Description

Most applications have no need to create or free TAdapterItemRequestParams (see page 24855) instances. These objects are managed by the adapter dispatcher.

See Also

TAdapterDispatcher (see page 24840)

179.1.6.4 TAdapterItemRequestParams Methods

The methods of the TAdapterItemRequestParams class are listed here.

Protected Methods

	Name	Description
	GetFailurePage (see page 24857)	Returns the value of the FailurePage property.
	GetParamValues (see page 24858)	Returns the value of the ParamValues property.
	GetSuccessPage (see page 24858)	Returns the value of the SuccessPage property.
	SetFailurePage (see page 24858)	Sets the value of the FailurePage property.
	SetSuccessPage (see page 24859)	Sets the value of the SuccessPage property.

179.1.6.4.1 TAdapterItemRequestParams.GetFailurePage Method

Returns the value of the FailurePage property.

Pascal

```
function GetFailurePage: string;
```

C++

```
__fastcall AnsiString GetFailurePage();
```

Description

GetFailurePage is the read implementation of the FailurePage property.

See Also

FailurePage (see page 19676)

179.1.6.4.2 TAdapterItemRequestParams.GetParamValues Method

Returns the value of the ParamValues property.

Pascal

```
function GetParamValues: TStrings;
```

C++

```
__fastcall TStrings GetParamValues();
```

Description

GetParamValues is the read implementation of the ParamValues property.

See Also

ParamValues (see page 19677)

179.1.6.4.3 TAdapterItemRequestParams.GetSuccessPage Method

Returns the value of the SuccessPage property.

Pascal

```
function GetSuccessPage: string;
```

C++

```
__fastcall AnsiString GetSuccessPage();
```

Description

GetSuccessPage is the read implementation of the SuccessPage property.

See Also

SuccessPage (see page 19677)

179.1.6.4.4 TAdapterItemRequestParams.SetFailurePage Method

Sets the value of the FailurePage property.

Pascal

```
procedure SetFailurePage(const AValue: string);
```

C++

```
__fastcall SetFailurePage(const AnsiString AValue);
```

Description

SetFailurePage is the write implementation of the FailurePage property.

AValue is the name of the Web page that should be returned if the adapter request fails.

See Also

FailurePage (see page 19676)

179.1.6.4.5 TAdapterItemRequestParams.SetSuccessPage Method

Sets the value of the SuccessPage property.

Pascal

```
procedure SetSuccessPage(const AValue: string);
```

C++

```
__fastcall SetSuccessPage(const AnsiString AValue);
```

Description

SetSuccessPage is the write implementation of the SuccessPage property.

AValue is the name of the Web page that should be returned when the adapter request succeeds.

See Also

SuccessPage (see page 19677)

179.1.6.5 TAdapterItemRequestParams Properties

The properties of the TAdapterItemRequestParams class are listed here.

Public Properties

	Name	Description
	FailurePage (see page 24859)	Indicates the Web page that the application should return if the current request fails.
R	ParamValues (see page 24860)	Lists the parameter values that are included in requests to the adapter action or field.
	SuccessPage (see page 24860)	Indicates the Web page that the application should return if the request succeeds.

179.1.6.5.1 TAdapterItemRequestParams.FailurePage Property

Indicates the Web page that the application should return if the current request fails.

Pascal

```
property FailurePage: string;
```

C++

```
__property AnsiString FailurePage;
```

Description

FailurePage is the name of the registered Web page module that should generate the content of the response message if the request to the adapter fails.

See Also

SuccessPage (see page 24860)

179.1.6.5.2 TAdapterItemRequestParams.ParamValues Property

Lists the parameter values that are included in requests to the adapter action or field.

Pascal

```
property ParamValues: TStrings;
```

C++

```
__property TStrings ParamValues;
```

Description

Use ParamValues to access the list of parameter values. Use the Names property of ParamValues to obtain the names of the parameters and the Values property to access the corresponding values.

See Also

Names (see page 3075)

Values (see page 3086)

179.1.6.5.3 TAdapterItemRequestParams.SuccessPage Property

Indicates the Web page that the application should return if the request succeeds.

Pascal

```
property SuccessPage: string;
```

C++

```
__property AnsiString SuccessPage;
```

Description

SuccessPage is the name of the registered Web page module that should generate the content of the response message when the request to the adapter succeeds.

See Also

SuccessPage

179.1.7 TBaseRequestHandler Class

The TBaseRequestHandler component is a base class for WebSnap dispatcher components.

Class Hierarchy

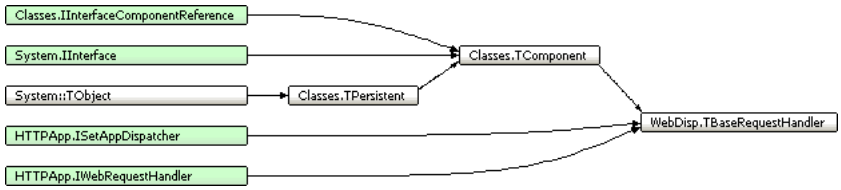

Pascal

```
TBaseRequestHandler = class(TComponent, IWebRequestHandler, ISetAppDispatcher);
```

C++

```
class TBaseRequestHandler : public TComponent, public IWebRequestHandler, public
ISetAppDispatcher;
```

File

WebDisp

Description

The TCustomAdapterDispatcher (see page 24865) and TCustomPageDispatcher (see page 24889) components are derived from TBaseRequestHandler.

179.1.7.1 TBaseRequestHandler Members

The following tables list the members exposed by TBaseRequestHandler.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleRequest (see page 24864)	Handles an HTTP request.
V	ImplHandleRequest (see page 24864)	Handles application-specific processing of an HTTP request.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 24864)	Sets the AppDispatcher (see page 24865) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	AppDispatcher (see page 24865)	Provides read-only access to the application dispatcher component.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.

	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

179.1.7.2 TBaseRequestHandler Methods

The methods of the TBaseRequestHandler class are listed here.

Protected Methods

	Name	Description
	HandleRequest (see page 24864)	Handles an HTTP request.
V	ImplHandleRequest (see page 24864)	Handles application-specific processing of an HTTP request.
	SetAppDispatcher (see page 24864)	Sets the AppDispatcher (see page 24865) property.

179.1.7.2.1 TBaseRequestHandler.HandleRequest Method

Handles an HTTP request.

Pascal

```
function HandleRequest(ARequest: TWebRequest; AResponse: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(TWebRequest ARequest, TWebResponse AResponse);
```

Description

HandleRequest returns true if it handles the request specified by ARequest. This function calls the ImplHandleRequest (see page 24864) to process the request. A customized overridden function should be written for ImplHandleRequest (see page 24864) to do the necessary processing specific to the application.

See Also

ImplHandleRequest (see page 24864)

179.1.7.2.2 TBaseRequestHandler.ImplHandleRequest Method

Handles application-specific processing of an HTTP request.

Pascal

```
function ImplHandleRequest(ARequest: TWebRequest; AResponse: TWebResponse): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest(TWebRequest ARequest, TWebResponse AResponse);
```

Description

The ImplHandleRequest function is called from HandleRequest (see page 24864) to handle the application-specific processing of the HTTP request.

This base version of the function generates an assertion failure as a reminder that a customized overridden version of this function needs to be defined in order to handle the request processing specific to the application.

See Also

HandleRequest (see page 24864)

179.1.7.2.3 TBaseRequestHandler.SetAppDispatcher Method

Sets the AppDispatcher (see page 24865) property.

Pascal

```
procedure SetAppDispatcher(const AWebAppDispatcher: TComponent);
```

C++

```
__fastcall SetAppDispatcher(const TComponent * AWebAppDispatcher);
```

Description

SetAppDispatcher provides a way to set the AppDispatcher (see page 24865) property.

See Also

AppDispatcher (see page 24865)

179.1.7.3 TBaseRequestHandler Properties

The properties of the TBaseRequestHandler class are listed here.

Protected Properties

	Name	Description
R	AppDispatcher (see page 24865)	Provides read-only access to the application dispatcher component.

179.1.7.3.1 TBaseRequestHandler.AppDispatcher Property

Provides read-only access to the application dispatcher component.

Pascal

```
property AppDispatcher: TComponent;
```

C++

```
__property TComponent * AppDispatcher;
```

Description

The application dispatcher component is typically TWebAppComponents (see page 24958). The application dispatcher component is used to access application objects such as TWebRequest (see page 24989) and application components such as TEndUserAdapter.

See Also

TCustomAdapterDispatcher (see page 24865)

TCustomPageDispatcher (see page 24889)

TWebAppComponents (see page 24958)

179.1.8 TCustomAdapterDispatcher Class

TCustomAdapterDispatcher automatically handles HTML form submissions and requests for dynamic images by calling adapter action and field components.

Class Hierarchy

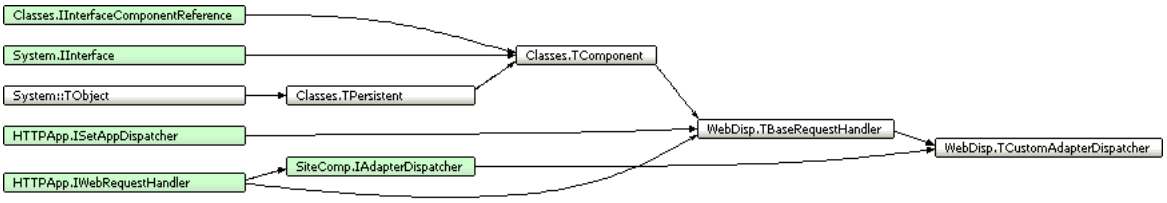

Pascal

```
TCustomAdapterDispatcher = class(TBaseRequestHandler, IAdapterDispatcher);
```

C++

```
class TCustomAdapterDispatcher : public TBaseRequestHandler, public IAdapterDispatcher;
```

File

WebDisp

Description

TCustomAdapterDispatcher is the base class for the adapter dispatcher. The adapter dispatcher is responsible for dispatching adapter actions and references to image fields. It also handles the encoding of parameters that must be included in HTML form submissions that invoke those adapter actions or field references. If a WebSnap application does not include an AdapterDispatcher component, then it can't execute adapter actions or obtain images from adapter fields.

See Also

TAdapter (see page 23713)

TAdapterDispatcher (see page 24840)

TCustomWebAppComponents (see page 24906)

179.1.8.1 TCustomAdapterDispatcher Members

The following tables list the members exposed by TCustomAdapterDispatcher.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AddSpecialParams (see page 24870)	Adds parameters for the success and failure Web pages to the list of adapter parameters.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoAdapterRequestNotHandled (see page 24870)	Generates an OnActionRequestNotHandled (see page 24877) and an OnImageRequestNotHandled (see page 24879) event.
	DoAfterDispatchAdapterRequest (see page 24871)	Generates an OnAfterDispatchAction (see page 24877) and an OnAfterDispatchImage (see page 24878) event.
	DoBeforeDispatchAdapterRequest (see page 24871)	Generates an OnBeforeDispatchAction (see page 24878) and an OnBeforeDispatchImage (see page 24878) event.
	EncodeAdapterItemRequestAsFieldValue (see page 24872)	Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.
	EncodeAdapterItemRequestAsHREF (see page 24872)	Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.
	GetAdapterItemRequestParams (see page 24873)	Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.

	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetDispatchParams (see page 24873)	Parses a request to determine the adapter to which it is directed.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleRequest (see page 24864)	Handles an HTTP request.
V	ImplEncodeAdapterItemRequestAsFieldValue (see page 24874)	Provides the underlying implementation for the EncodeAdapterItemRequestAsFieldValue (see page 24872) method.
V	ImplEncodeAdapterItemRequestAsHREF (see page 24874)	Provides the underlying implementation for the EncodeAdapterItemRequestAsHREF (see page 24872) method.
V	ImplGetAdapterItemRequestParams (see page 24875)	Provides the underlying implementation for the GetAdapterItemRequestParams (see page 24873) method.
V	ImplHandleRequest (see page 24876)	Provides the underlying implementation for the HandleRequest (see page 24864) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 24864)	Sets the AppDispatcher (see page 24865) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	AppDispatcher (see page 24865)	Provides read-only access to the application dispatcher component.

Public Events

	Name	Description
	OnActionRequestNotHandled (see page 24877)	Occurs when the WebSnap application could not execute a requested action.
	OnAfterDispatchAction (see page 24877)	Occurs after the target adapter has finished executing an action.
	OnAfterDispatchImage (see page 24878)	Occurs after the target adapter has finished handling an image request.
	OnBeforeDispatchAction (see page 24878)	Occurs before the dispatcher forwards an action to an adapter for execution.

	OnBeforeDispatchImage (see page 24878)	Occurs before the dispatcher forwards an image request to an adapter for handling.
	OnGetAdapterRequestParams (see page 24879)	Occurs when the adapter dispatcher needs to obtain the parameters for executing an action or retrieving an image.
	OnImageRequestNotHandled (see page 24879)	Occurs when the WebSnap application could not provide a requested image.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandleRequest (see page 15238)	Processes an HTTP request message.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

179.1.8.2 TCustomAdapterDispatcher Methods

The methods of the TCustomAdapterDispatcher class are listed here.

Protected Methods

	Name	Description
V	AddSpecialParams (see page 24870)	Adds parameters for the success and failure Web pages to the list of adapter parameters.
	DoAdapterRequestNotHandled (see page 24870)	Generates an OnActionRequestNotHandled (see page 24877) and an OnImageRequestNotHandled (see page 24879) event.
	DoAfterDispatchAdapterRequest (see page 24871)	Generates an OnAfterDispatchAction (see page 24877) and an OnAfterDispatchImage (see page 24878) event.
	DoBeforeDispatchAdapterRequest (see page 24871)	Generates an OnBeforeDispatchAction (see page 24878) and an OnBeforeDispatchImage (see page 24878) event.
	EncodeAdapterItemRequestAsFieldValue (see page 24872)	Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.
	EncodeAdapterItemRequestAsHREF (see page 24872)	Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.

	GetAdapterItemRequestParams (see page 24873)	Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.
V	GetDispatchParams (see page 24873)	Parses a request to determine the adapter to which it is directed.
V	ImplEncodeAdapterItemRequestAsFieldValue (see page 24874)	Provides the underlying implementation for the EncodeAdapterItemRequestAsFieldValue (see page 24872) method.
V	ImplEncodeAdapterItemRequestAsHREF (see page 24874)	Provides the underlying implementation for the EncodeAdapterItemRequestAsHREF (see page 24872) method.
V	ImplGetAdapterItemRequestParams (see page 24875)	Provides the underlying implementation for the GetAdapterItemRequestParams (see page 24873) method.
V	ImplHandleRequest (see page 24876)	Provides the underlying implementation for the HandleRequest (see page 24864) method.

179.1.8.2.1 TCustomAdapterDispatcher.AddSpecialParams Method

Adds parameters for the success and failure Web pages to the list of adapter parameters.

Pascal

```
procedure AddSpecialParams(AHandler: IUnknown; Params: IAdapterItemRequestParams); virtual;
```

C++

```
virtual __fastcall AddSpecialParams(IUnknown AHandler, IAdapterItemRequestParams Params);
```

Description

The adapter dispatcher calls AddSpecialParams internally after retrieving a set of parameters from an adapter (handler) and before generating an OnGetAdapterRequestParams (see page 24879) event. This method adds parameters to the list that are not required by the adapter for handling the request.

AddSpecialParams retrieves the Web pages that should be displayed when the adapter succeeds or fails in processing the request with which the handler is associated. If the handler has any special pages to display, AddSpecialParams adds the names of these pages to the parameters, using the parameter names of '__sp' and '__fp' (for success page and failure page), or '__p' (if the same page should be used for both success and failure).

AHandler is the interface for the adapter associated with the action or image.

Params is an interface that provides access to the current list of parameters.

See Also

GetAdapterItemRequestParams (see page 24873)

OnGetAdapterRequestParams (see page 24879)

FailurePage (see page 19676)

SuccessPage (see page 19677)

179.1.8.2.2 TCustomAdapterDispatcher.DoAdapterRequestNotHandled Method

Generates an OnActionRequestNotHandled (see page 24877) and an OnImageRequestNotHandled (see page 24879) event.

Pascal

```
procedure DoAdapterRequestNotHandled(AObject: TObject; var AHandled: Boolean);
```

C++

```
__fastcall DoAdapterRequestNotHandled(TObject * AObject, Boolean AHandled);
```

Description

TCustomAdapterDispatcher (see page 24865) calls DoAdapterRequestNotHandled internally when an adapter request

remains unhandled. The request may be unhandled because the adapter could not execute the requested action or obtain the requested image, or it may be unhandled because the dispatcher could not locate an adapter to handle the request.

DoAdapterRequestNotHandled assembles the parameters for an action request and generates an OnActionRequestNotHandled (see page 24877) event. It then assembles the parameters for an image request and generates an OnImageRequestNotHandled (see page 24879) event.

AObject is the adapter that failed to handle the request (or nil (Delphi) or NULL (C++) if the dispatcher could not locate an adapter for the request).

AHandled returns true if an event handler handled the request.

See Also

OnActionRequestNotHandled (see page 24877)

OnImageRequestNotHandled (see page 24879)

179.1.8.2.3 TCustomAdapterDispatcher.DoAfterDispatchAdapterRequest Method

Generates an OnAfterDispatchAction (see page 24877) and an OnAfterDispatchImage (see page 24878) event.

Pascal

```
procedure DoAfterDispatchAdapterRequest(AObject: TObject);
```

C++

```
__fastcall DoAfterDispatchAdapterRequest(TObject * AObject);
```

Description

TCustomAdapterDispatcher (see page 24865) calls DoAfterDispatchAdapterRequest internally after calling on an adapter to handle a request. It calls this method both when the adapter indicates success and when it indicates failure in handling the request.

DoAfterDispatchAdapterRequest assembles the parameters for an action request and generates an OnAfterDispatchAction (see page 24877) event. It then assembles the parameters for an image request and generates an OnAfterDispatchImage (see page 24878) event.

AObject is the adapter that just handled the current request.

See Also

OnAfterDispatchAction (see page 24877)

OnAfterDispatchImage (see page 24878)

179.1.8.2.4 TCustomAdapterDispatcher.DoBeforeDispatchAdapterRequest Method

Generates an OnBeforeDispatchAction (see page 24878) and an OnBeforeDispatchImage (see page 24878) event.

Pascal

```
procedure DoBeforeDispatchAdapterRequest(AObject: TObject; var AHandled: Boolean);
```

C++

```
__fastcall DoBeforeDispatchAdapterRequest(TObject * AObject, Boolean AHandled);
```

Description

TCustomAdapterDispatcher (see page 24865) calls DoBeforeDispatchAdapterRequest internally before calling on an adapter to handle a request. It calls this method only after it has identified the appropriate handler for responding to the current request.

DoBeforeDispatchAdapterRequest assembles the parameters for an action request and generates an OnBeforeDispatchAction (see page 24878) event. It then assembles the parameters for an image request and generates an OnBeforeDispatchImage (see page 24878) event.

AObject is the adapter for handling the current request.

AHandled returns true if an event handler handled the request.

See Also

OnBeforeDispatchAction (see page 24878)

OnBeforeDispatchImage (see page 24878)

179.1.8.2.5 TCustomAdapterDispatcher.EncodeAdapterItemRequestAsFieldValue Method

Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.

Pascal

```
function EncodeAdapterItemRequestAsFieldValue(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions): string;
```

C++

```
__fastcall AnsiString EncodeAdapterItemRequestAsFieldValue(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions);
```

Description

Adapter components call EncodeAdapterItemRequestAsFieldValue to encode the action or field parameters as they should appear as field values in form submissions. The resulting parameter string has a form such as '?param1=value1&param2=value2'.

AHandler is the interface of the adapter whose action or image field requires the parameters.

AOptions is a set of flags that indicate what types of parameters to include.

EncodeAdapterItmRequestAsFieldValue returns the parameter string.

See Also

GetAdapterItemRequestParams (see page 24873)

EncodeAdapterItemRequestAsHREF (see page 24872)

179.1.8.2.6 TCustomAdapterDispatcher.EncodeAdapterItemRequestAsHREF Method

Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.

Pascal

```
function EncodeAdapterItemRequestAsHREF(AHandler: IUnknown; AOptions:
```

```
TAdapterItemRequestParamOptions): string;
```

C++

```
__fastcall AnsiString EncodeAdapterItemRequestAsHREF(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions);
```

Description

Adapter components call EncodeAdapterItemRequestAsHREF to encode the action or image field parameters as they should appear as parameters to an anchor tag in an HTML form.

AHandler is the interface of the adapter whose action or image field requires the parameters.

AOptions is a set of flags that indicate what types of parameters to include.

EncodeAdapterItemRequestAsHREF returns the parameter string.

See Also

GetAdapterItemRequestParams (see page 24873)

EncodeAdapterItemRequestAsFieldValue (see page 24872)

179.1.8.2.7 TCustomAdapterDispatcher.GetAdapterItemRequestParams Method

Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.

Pascal

```
procedure GetAdapterItemRequestParams(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions; Params: IAdapterItemRequestParams);
```

C++

```
__fastcall GetAdapterItemRequestParams(IUnknown AHandler, TAdapterItemRequestParamOptions
AOptions, IAdapterItemRequestParams Params);
```

Description

GetAdapterItemRequestParams retrieves the parameters for a call to execute an adapter action or to fetch the value of an image field.

AHandler is the interface of the adapter that requires the parameters to execute the action or produce the image field value.

AOptions is a set of flags that indicate what types of parameters should be included.

AParams is an interface for reading and writing parameter values. GetAdapterItemRequestParams uses this interface to write the parameter values.

GetAdapterItemRequestParams simply calls the ImplGetAdapterItemRequestParams (see page 24875) method, which fetches the parameter values and writes them to AParams.

See Also

EncodeAdapterItemRequestAsHREF (see page 24872)

EncodeAdapterItemRequestAsFieldValue (see page 24872)

ImplGetAdapterItemRequestParams (see page 24875)

179.1.8.2.8 TCustomAdapterDispatcher.GetDispatchParams Method

Parses a request to determine the adapter to which it is directed.

Pascal

```
function GetDispatchParams(Request: TWebRequest): TAdapterDispatchParams; virtual;
```

C++

```
virtual __fastcall TAdapterDispatchParams GetDispatchParams(TWebRequest Request);
```

Description

TCustomAdapterDispatcher (see page 24865) calls GetDispatchParams internally to obtain information about the adapter to which a request is directed. GetDispatchParams creates a new TAdapterDispatchParams (see page 24848) object, which parses the current request to determine the adapter to which it is directed.

Request is the current http request object.

GetDispatchParams returns the new TAdapterDispatchParams (see page 24848) object, whose properties have been filled out to reflect the information in the current request. It is the caller's responsibility to free this object.

179.1.8.2.9
TCustomAdapterDispatcher.ImplEncodeAdapterItemRequestAsFieldValue Method

Provides the underlying implementation for the EncodeAdapterItemRequestAsFieldValue (see page 24872) method.

Pascal

```
function ImplEncodeAdapterItemRequestAsFieldValue(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions): string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplEncodeAdapterItemRequestAsFieldValue(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions);
```

Description

The EncodeAdapterItemRequestAsFieldValue (see page 24872) method calls ImplEncodeAdapterItemRequestAsFieldValue to retrieve a set of parameters formatted as a set of field values for an HTTP request. ImplEncodeAdapterItemRequestAsFieldValue calls GetAdapterItemRequestParams (see page 24873) to fetch the parameter values, and then formats them as a set of field values.

AHandler is the interface of the adapter whose action or image field requires the parameters.

AOptions is a set of flags that indicate what types of parameters to include.

EncodeAdapterItmRequestAsFieldValue returns the parameter string.

See Also

EncodeAdapterItemRequestAsFieldValue (see page 24872)

GetAdapterItemRequestParams (see page 24873)

179.1.8.2.10
TCustomAdapterDispatcher.ImplEncodeAdapterItemRequestAsHREF Method

Provides the underlying implementation for the EncodeAdapterItemRequestAsHREF (see page 24872) method.

Pascal

```
function ImplEncodeAdapterItemRequestAsHREF(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions): string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplEncodeAdapterItemRequestAsHREF(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions);
```

Description

The EncodeAdapterItemRequestAsHREF (see page 24872) method calls ImplEncodeAdapterItemRequestAsFieldValue (see page 24874) to retrieve a set of parameters formatted as the parameters to an anchor tag in an HTML form. ImplEncodeAdapterItemRequestAsHREF calls GetAdapterItemRequestParams (see page 24873) to fetch the parameter values, and then formats them.

AHandler is the interface of the adapter whose action or image field requires the parameters.

AOptions is a set of flags that indicate what types of parameters to include.

EncodeAdapterItemRequestAsHREF (see page 24872) returns the parameter string.

See Also

EncodeAdapterItemRequestAsHREF (see page 24872)

GetAdapterItemRequestParams (see page 24873)

179.1.8.2.11 TCustomAdapterDispatcher.ImplGetAdapterItemRequestParams Method

Provides the underlying implementation for the GetAdapterItemRequestParams (see page 24873) method.

Pascal

```
procedure ImplGetAdapterItemRequestParams(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions; Params: IAdapterItemRequestParams); virtual;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions, IAdapterItemRequestParams Params);
```

Description

GetAdapterItemRequestParams (see page 24873) calls ImplGetAdapterItemRequestParams to retrieve the parameters for a call to execute an adapter action or to fetch the value of an image field. This method fetches the requested parameters from the adapter interface, calls the AddSpecialParams (see page 24870) method to add additional parameters for the return Web page, and then generates an OnGetAdapterRequestParams (see page 24879) event so that you can make your own final adjustments to the parameter list.

AHandler is the interface of the adapter that requires the parameters to execute the action or produce the image field value.

AOptions is a set of flags that indicate what types of parameters should be included.

AParams is an interface for reading and writing parameter values. ImplGetAdapterItemRequestParams uses this interface to write the parameter values.

See Also

GetAdapterItemRequestParams (see page 24873)

AddSpecialParams (see page 24870)

OnGetAdapterRequestParams (see page 24879)

179.1.8.2.12 TCustomAdapterDispatcher.ImplHandleRequest Method

Provides the underlying implementation for the HandleRequest (see page 24864) method.

Pascal

```
function ImplHandleRequest(Request: TWebRequest; Response: TWebResponse): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest(TWebRequest Request, TWebResponse Response);
```

Description

The HandleRequest (see page 24864) method calls ImplHandleRequest to process an http request message and obtain the response that should be sent back to the caller. (The HandleRequest (see page 24864) method is called using the IWebRequestHandler interface.)

ImplHandleRequest

1. Calls GetDispatchParams (see page 24873) to parse the request and identify the adapter that should handle it.
2. If it finds an adapter, calls DoBeforeDispatchAdapterRequest (see page 24871) to generate the OnBeforeDispatchAction (see page 24878) and OnBeforeDispatchImage (see page 24878) events.
3. If the event handlers do not handle the event, forwards the request to the adapter for processing.
4. Calls DoAfterDispatchAdapterRequest (see page 24871) to generate the OnAfterDispatchAction (see page 24877) and OnAfterDispatchImage (see page 24878) events.
5. Calls DoAdapterRequestNotHandled (see page 24870) to generate the OnActionRequestNotHandled (see page 24877) and OnImageRequestNotHandled (see page 24879) events if the request has not yet been handled.

See Also

IWebRequestHandler (see page 15237)

HandleRequest (see page 24864)

GetDispatchParams (see page 24873)

DoBeforeDispatchAdapterRequest (see page 24871)

DoAfterDispatchAdapterRequest (see page 24871)

DoAdapterRequestNotHandled (see page 24870)

179.1.8.3 TCustomAdapterDispatcher Events

The events of the TCustomAdapterDispatcher class are listed here.

Public Events

	Name	Description
	OnActionRequestNotHandled (see page 24877)	Occurs when the WebSnap application could not execute a requested action.
	OnAfterDispatchAction (see page 24877)	Occurs after the target adapter has finished executing an action.
	OnAfterDispatchImage (see page 24878)	Occurs after the target adapter has finished handling an image request.
	OnBeforeDispatchAction (see page 24878)	Occurs before the dispatcher forwards an action to an adapter for execution.
	OnBeforeDispatchImage (see page 24878)	Occurs before the dispatcher forwards an image request to an adapter for handling.
	OnGetAdapterRequestParams (see page 24879)	Occurs when the adapter dispatcher needs to obtain the parameters for executing an action or retrieving an image.

	OnImageRequestNotHandled (see page 24879)	Occurs when the WebSnap application could not provide a requested image.

179.1.8.3.1 TCustomAdapterDispatcher.OnActionRequestNotHandled Event

Occurs when the WebSnap application could not execute a requested action.

Pascal

```
property OnActionRequestNotHandled: TDispatchActionHandledEvent;
```

C++

```
__property TDispatchActionHandledEvent OnActionRequestNotHandled;
```

Description

Write an OnActionRequestNotHandled event handler to provide a response to actions that the adapter components in the application can't execute. This may occur because the adapter dispatcher could not find an adapter to handle the action, or it may occur because the identified adapter was unable to execute the action.

If the OnActionRequestNotHandled event handler does not set its Handled parameter to true, the adapter dispatcher generates an OnImageRequestNotHandled (see page 24879). If the OnImageRequestNotHandled (see page 24879) event handler also does not handle the event, the dispatcher raises an EAdapterDispatcherException (see page 24838) exception.

Tip: Use the Response property of the current Web context to access the response message from the OnActionRequestNotHandled event handler. You can obtain a reference to the Web context by calling the global WebContext function.

See Also

OnAfterDispatchAction (see page 24877)

OnBeforeDispatchAction (see page 24878)

EAdapterDispatcherException (see page 24838)

WebContext (see page 24687)

Response (see page 24659)

OnImageRequestNotHandled (see page 24879)

179.1.8.3.2 TCustomAdapterDispatcher.OnAfterDispatchAction Event

Occurs after the target adapter has finished executing an action.

Pascal

```
property OnAfterDispatchAction: TDispatchActionEvent;
```

C++

```
__property TDispatchActionEvent OnAfterDispatchAction;
```

Description

Write an OnAfterDispatchAction event handler to take specific action after an adapter has executed an action. Note that this event can occur even if the request does not request the execution of an action or if the adapter was not successful in its attempt to execute the action.

See Also

OnActionRequestNotHandled (see page 24877)

OnBeforeDispatchAction (see page 24878)

179.1.8.3.3 TCustomAdapterDispatcher.OnAfterDispatchImage Event

Occurs after the target adapter has finished handling an image request.

Pascal

```
property OnAfterDispatchImage: TDispatchImageEvent;
```

C++

```
__property TDispatchImageEvent OnAfterDispatchImage;
```

Description

Write an OnAfterDispatchImage event handler to take specific action after an adapter has responded to an image request. Note that this event can occur even if the request does not request the value of an image field or if the adapter was not successful in its attempt to honor that request.

See Also

OnImageRequestNotHandled (see page 24879)

OnBeforeDispatchImage (see page 24878)

179.1.8.3.4 TCustomAdapterDispatcher.OnBeforeDispatchAction Event

Occurs before the dispatcher forwards an action to an adapter for execution.

Pascal

```
property OnBeforeDispatchAction: TDispatchActionHandledEvent;
```

C++

```
__property TDispatchActionHandledEvent OnBeforeDispatchAction;
```

Description

Write an OnBeforeDispatchAction event handler to respond to action requests before they are forwarded to the associated adapter for execution. Note that if the request does not request the execution of an action, the Params parameter of the event handler is nil (Delphi) or NULL (C++).

If the OnBeforeDispatchAction event handler does not set its Handled parameter to true, the adapter dispatcher forwards the request to the identified adapter that is the value of the Action parameter. If the event handler sets Handled to false, the dispatcher does not forward the request to the adapter for execution, and does not generate an OnAfterDispatchAction (see page 24877) event.

Tip: Use the Response property of the current Web context to access the response message from the OnBeforeDispatchAction event handler. You can obtain a reference to the Web context by calling the global WebContext function.

See Also

OnActionRequestNotHandled (see page 24877)

OnAfterDispatchAction (see page 24877)

WebContext (see page 24687)

Response (see page 24659)

179.1.8.3.5 TCustomAdapterDispatcher.OnBeforeDispatchImage Event

Occurs before the dispatcher forwards an image request to an adapter for handling.

Pascal

```
property OnBeforeDispatchImage: TDispatchImageHandledEvent;
```

C++

```
__property TDispatchImageHandledEvent OnBeforeDispatchImage;
```

Description

Write an OnBeforeDispatchImage event handler to respond to image requests before they are forwarded to the associated adapter for handling. Note that if the request does not ask for the value of an image field, the Params parameter of the event handler is nil (Delphi) or NULL (C++).

If the OnBeforeDispatchImage event handler does not set its Handled parameter to true, the adapter dispatcher forwards the request to the identified adapter that is the value of the Image parameter. If the event handler sets Handled to false, the dispatcher does not forward the request to the adapter for handling, and does not generate an OnAfterDispatchImage (see page 24878) event.

Tip: Use the Response property of the current Web context to access the response message from the OnBeforeDispatchImage event handler. You can obtain a reference to the Web context by calling the global WebContext function.

See Also

OnImageRequestNotHandled (see page 24879)

OnAfterDispatchImage (see page 24878)

WebContext (see page 24687)

Response (see page 24659)

179.1.8.3.6 TCustomAdapterDispatcher.OnGetAdapterRequestParams Event

Occurs when the adapter dispatcher needs to obtain the parameters for executing an action or retrieving an image.

Pascal

```
property OnGetAdapterRequestParams: TGetAdapterRequestParamsEvent;
```

C++

```
__property TGetAdapterRequestParamsEvent OnGetAdapterRequestParams;
```

Description

Write an OnGetAdapterRequestParams event handler to modify the list of parameters for an action or image request before they are used. The OnGetAdapterRequestParams event handler can modify the current parameters or add its own parameters to the list. This event occurs both when the dispatcher formats parameters that are added to HTML forms and when it is dispatching a request message that contains an action or image request.

See Also

GetAdapterItemRequestParams (see page 24873)

EncodeAdapterItemRequestAsFieldValue (see page 24872)

EncodeAdapterItemRequestAsHREF (see page 24872)

AddSpecialParams (see page 24870)

179.1.8.3.7 TCustomAdapterDispatcher.OnImageRequestNotHandled Event

Occurs when the WebSnap application could not provide a requested image.

Pascal

```
property OnImageRequestNotHandled: TDispatchImageHandledEvent;
```

C++

```
__property TDispatchImageHandledEvent OnImageRequestNotHandled;
```

Description

Write an OnImageRequestNotHandled event handler to provide a response to image requests that the adapter components in the application can't honor. This may occur because the adapter dispatcher could not find an adapter to handle the request, or it may occur because the identified adapter was unable to provide the requested image.

If the OnImageRequestNotHandled event handler exits with its Handled parameter set to false, the dispatcher raises an EAdapterDispatcherException (see page 24838) exception.

Tip: Use the Response property of the current Web context to access the response message from the OnImageRequestNotHandled event handler. You can obtain a reference to the Web context by calling the global WebContext function.

See Also

OnAfterDispatchImage (see page 24878)

OnBeforeDispatchImage (see page 24878)

EAdapterDispatcherException (see page 24838)

WebContext (see page 24687)

Response (see page 24659)

OnActionRequestNotHandled (see page 24877)

179.1.9 TCustomLocateFileService Class

TCustomLocateFileService uses event handlers to control access to templates and server-side script include files at runtime.

Class Hierarchy

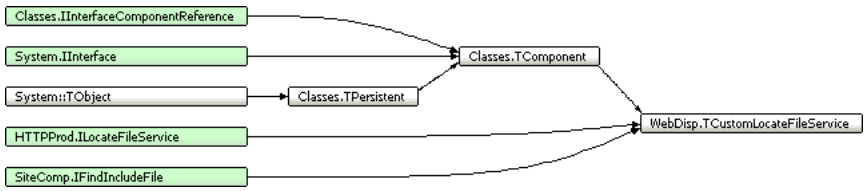

Pascal

```
TCustomLocateFileService = class(TComponent, ILocateFileService, IFindIncludeFile);
```

C++

```
class TCustomLocateFileService : public TComponent, public ILocateFileService, public
IFindIncludeFile;
```

File

WebDisp

Description

TCustomLocateFileService is the base class for components that control access to files in a WebSnap application. TCustomLocateFileService defines events that indicate where to locate a requested file or that supply a stream object from which to read the contents of a file. By using a TCustomLocateFileService descendant, a WebSnap application can adjust to differences in the way files are deployed or even to use different files, depending on the current Web context.

Do not create instances of TCustomLocateFileService. All of the events it introduces are protected, and so unavailable to your application. Instead, use a TCustomLocateFileService descendant, such as TLocateFileService (see page 24928), which publishes those events.

See Also

TLocateFileService (see page 24928)

TStream (see page 2973)

179.1.9.1 TCustomLocateFileService Members

The following tables list the members exposed by TCustomLocateFileService.

Protected Events

	Name	Description
	OnFindIncludeFile (see page 24888)	Occurs when the WebSnap application tries to locate an include file for a server-side script.
	OnFindStream (see page 24889)	Occurs when the WebSnap application tries to locate a file.
	OnFindTemplateFile (see page 24889)	Occurs when the WebSnap application tries to locate the template file for a content producer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FindIncludeFile (see page 24884)	Returns the fully-qualified path name for an include file for a server-side script.
V	FindStream (see page 24885)	Generates an OnFindStream (see page 24889) event.
	FindTemplateFile (see page 24885)	Generates an OnFindTemplateFile (see page 24889) event.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetFileStream (see page 24886)	Returns a stream object from which to read a specified file.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTemplateStream (see page 24886)	Returns a stream from which to read a template file.

	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	ImplFindIncludeFile (see page 24887)	Provides the underlying implementation for the FindIncludeFile (see page 24884) method.
V	ImplGetTemplateStream (see page 24887)	Provides the underlying implementation for the GetTemplateStream (see page 24886) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.

	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.

	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

179.1.9.2 TCustomLocateFileService Methods

The methods of the TCustomLocateFileService class are listed here.

Protected Methods

	Name	Description
	FindIncludeFile (see page 24884)	Returns the fully-qualified path name for an include file for a server-side script.
V	FindStream (see page 24885)	Generates an OnFindStream (see page 24889) event.
	FindTemplateFile (see page 24885)	Generates an OnFindTemplateFile (see page 24889) event.
V	GetFileStream (see page 24886)	Returns a stream object from which to read a specified file.
	GetTemplateStream (see page 24886)	Returns a stream from which to read a template file.
V	ImplFindIncludeFile (see page 24887)	Provides the underlying implementation for the FindIncludeFile (see page 24884) method.
V	ImplGetTemplateStream (see page 24887)	Provides the underlying implementation for the GetTemplateStream (see page 24886) method.

179.1.9.2.1 TCustomLocateFileService.FindIncludeFile Method

Returns the fully-qualified path name for an include file for a server-side script.

Pascal

```
function FindIncludeFile(AComponent: TComponent; const AFileName: string; var AResult:
string): Boolean;
```

C++

```
__fastcall Boolean FindIncludeFile(TComponent * AComponent, const AnsiString AFileName,
AnsiString AResult);
```

Description

WebSnap applications call FindIncludeFile when they encounter a reference to an include file in server-side script. This method returns the fully-qualified path name for the include file.

FindIncludeFile calls the virtual ImplFindIncludeFile (see page 24887) method to obtain the path and name of the include file. In TCustomLocateFileService (see page 24880), ImplFindIncludeFile (see page 24887) fetches the include file name by generating an OnFindIncludeFile (see page 24888) event. Component writers that which to locate include files in some other way should override the ImplFindIncludeFile (see page 24887) method.

AComponent is the Web module context object that stores the reference to the include file.

AFileName is the name of the include file to locate.

AResult returns the fully-qualified path name where the WebSnap application can find the requested file.

FindIncludeFile returns true if it locates the requested file and returns its fully-qualified name as AResult.

See Also

OnFindIncludeFile (see page 24888)

ImplFindIncludeFile (see page 24887)

WebModuleContext (see page 15279)

179.1.9.2.2 TCustomLocateFileService.FindStream Method

Generates an OnFindStream (see page 24889) event.

Pascal

```
function FindStream(AComponent: TComponent; const AFileName: string; out AResult: TStream; out
AOwned: Boolean): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean FindStream(TComponent * AComponent, const AnsiString AFileName,
TStream AResult, Boolean AOwned);
```

Description

TCustomLocateFileService (see page 24880) calls FindStream internally to generate an OnFindStream (see page 24889) event.

AComponent is the object that uses or is associated with the requested file. Typically, this is a content producer.

AFileName is the name of the file to locate. This can be a "logical" file that does not exist on disk.

AResult returns a stream from which the caller can read the contents of the requested file.

AOwned indicates whether the caller should free AResult after reading the file contents. When AOwned returns true, the caller should free the stream object. When AOwned returns false, the caller should not free the stream object.

FindStream returns true if it supplies a stream as the value of the AResult parameter. It returns false if there is no OnFindStream (see page 24889) event handler or if the event handler does not return a value for AResult.

See Also

OnFindStream (see page 24889)

179.1.9.2.3 TCustomLocateFileService.FindTemplateFile Method

Generates an OnFindTemplateFile (see page 24889) event.

Pascal

```
function FindTemplateFile(AComponent: TComponent; const AFileName: string; var AResult:
string): Boolean;
```

C++

```
__fastcall Boolean FindTemplateFile(TComponent * AComponent, const AnsiString AFileName,
AnsiString AResult);
```

Description

TCustomLocateFileService (see page 24880) calls FindTemplateFile internally to generate an OnFindTemplateFile (see page 24889) event.

AComponent is the object that uses or is associated with the requested template. Typically, this is a content producer.

AFileName is the name of the file to locate.

AResult returns the fully-qualified path name for the requested file.

FindTemplateFile returns true if it supplies a file name as the value of the AResult parameter. It returns false if there is no OnFindTemplateFile (see page 24889) event handler or if the event handler does not return a value for AResult.

See Also

OnFindTemplateFile (see page 24889)

179.1.9.2.4 TCustomLocateFileService.GetFileStream Method

Returns a stream object from which to read a specified file.

Pascal

```
function GetFileStream(const AFileName: string; out AOwned: Boolean): TStream; virtual;
```

C++

```
virtual __fastcall TStream GetFileStream(const AnsiString AFileName, Boolean AOwned);
```

Description

The GetTemplateStream (see page 24886) method calls GetFileStream internally to obtain the stream it returns when the requested file maps to a fully-qualified path name.

AFileName is the fully-qualified name of a file whose contents can be read from the returned stream.

AOwned indicates whether the caller should free the stream after reading the file contents. When AOwned returns true, the caller should free the stream object. When AOwned returns false, the caller should not free the stream object.

GetFileStream returns a stream object from which to read the contents of AFileName.

See Also

GetTemplateStream (see page 24886)

179.1.9.2.5 TCustomLocateFileService.GetTemplateStream Method

Returns a stream from which to read a template file.

Pascal

```
function GetTemplateStream(AComponent: TComponent; AFileName: string; out AOwned: Boolean):
TStream;
```

C++

```
__fastcall TStream GetTemplateStream(TComponent * AComponent, AnsiString AFileName, Boolean
AOwned);
```

Description

Producer components call GetTemplateStream to read the contents of the template files they use to generate their content. GetTemplateStream locates the specified template file (which can be a logical rather than a physical file name), and returns a stream from which the caller can read the file contents.

GetTemplateStream calls the ImplGetTemplateStream (see page 24887) method, which generates first an OnFindStream (see page 24889) event, and, if OnFindStream (see page 24889) is unsuccessful, an OnFindTemplateFile (see page 24889) event.

AComponent is the object that uses or is associated with the requested template. Typically, this is a content producer.

AFileName is the logical name of the file to locate.

AOwned indicates whether the caller should free the stream after reading the file contents. When AOwned returns true, the caller should free the stream object. When AOwned returns false, the caller should not free the stream object.

GetTemplateStream returns a stream from which to read the requested file, or, if it could not locate the file, nil (Delphi) or NULL

(C++).

See Also

ImplGetTemplateStream (see page 24887)

OnFindStream (see page 24889)

OnFindTemplateFile (see page 24889)

179.1.9.2.6 TCustomLocateFileService.ImplFindIncludeFile Method

Provides the underlying implementation for the FindIncludeFile (see page 24884) method.

Pascal

```
function ImplFindIncludeFile(AComponent: TComponent; const AFileName: string; var AResult:
string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplFindIncludeFile(TComponent * AComponent, const AnsiString
AFileName, AnsiString AResult);
```

Description

The FindIncludeFile (see page 24884) method calls ImplFindIncludeFile to locate the fully-qualified path name for an include file to a server-side script. ImplFindIncludeFile generates an OnFindIncludeFile (see page 24888) event to obtain the fully-qualified path name.

AComponent is the Web module context object that stores the reference to the include file.

AFileName is the name of the include file to locate.

AResult returns the fully-qualified path name where the WebSnap application can find the requested file.

FindIncludeFile (see page 24884) returns true if it locates the requested file and returns its fully-qualified name as AResult. It returns false if there is no OnFindIncludeFile (see page 24888) event handler or if the event handler does not return a value for AResult.

See Also

OnFindIncludeFile (see page 24888)

FindIncludeFile (see page 24884)

WebModuleContext (see page 15279)

179.1.9.2.7 TCustomLocateFileService.ImplGetTemplateStream Method

Provides the underlying implementation for the GetTemplateStream (see page 24886) method.

Pascal

```
function ImplGetTemplateStream(AComponent: TComponent; AFileName: string; out AOwned:
Boolean): TStream; virtual;
```

C++

```
virtual __fastcall TStream ImplGetTemplateStream(TComponent * AComponent, AnsiString
AFileName, Boolean AOwned);
```

Description

The GetTemplateStream (see page 24886) method calls ImplGetTemplateStream to obtain a stream from which to read a specified template. ImplGetTemplateStream generates an OnFindStream (see page 24889) event, and, if OnFindStream (

see page 24889) is unsuccessful, an OnFindTemplateFile (see page 24889) event. If it gets a file name from the OnFindTemplateFile (see page 24889) event handler, ImplGetTemplateStream calls the GetFileStream (see page 24886) method to obtain a stream object from which to read the file.

AComponent is the object that uses or is associated with the requested template. Typically, this is a content producer.

AFileName is the logical name of the file to locate.

AOwned indicates whether the caller should free the stream after reading the file contents. When AOwned returns true, the caller should free the stream object. When AOwned returns false, the caller should not free the stream object.

GetTemplateStream (see page 24886) returns a stream from which to read the requested file, or, if it could not locate the file, nil (Delphi) or NULL (C++).

See Also

GetTemplateStream (see page 24886)

OnFindStream (see page 24889)

OnFindTemplateFile (see page 24889)

GetFileStream (see page 24886)

179.1.9.3 TCustomLocateFileService Events

The events of the TCustomLocateFileService class are listed here.

Protected Events

	Name	Description
	OnFindIncludeFile (see page 24888)	Occurs when the WebSnap application tries to locate an include file for a server-side script.
	OnFindStream (see page 24889)	Occurs when the WebSnap application tries to locate a file.
	OnFindTemplateFile (see page 24889)	Occurs when the WebSnap application tries to locate the template file for a content producer.

179.1.9.3.1 TCustomLocateFileService.OnFindIncludeFile Event

Occurs when the WebSnap application tries to locate an include file for a server-side script.

Pascal

```
property OnFindIncludeFile: TLocateFileServiceFindFileEvent;
```

C++

```
__property TLocateFileServiceFindFileEvent OnFindIncludeFile;
```

Description

Write an OnFindIncludeFile event handler to indicate the location on disk of include files for server-side scripts. In WebSnap applications that include a locate file service component, you must supply either an OnFindIncludeFile or an OnFindStream (see page 24889) event handler (or both) if the server-side scripts have include files. Otherwise the application can't locate these include files at all.

See Also

OnFindStream (see page 24889)

OnFindTemplateFile (see page 24889)

179.1.9.3.2 TCustomLocateFileService.OnFindStream Event

Occurs when the WebSnap application tries to locate a file.

Pascal

```
property OnFindStream: TLocateFileServiceFindStreamEvent;
```

C++

```
__property TLocateFileServiceFindStreamEvent OnFindStream;
```

Description

Write an OnFindStream event handler to supply the WebSnap application with the contents of a file given its name. This event is particularly useful for applications that use "logical" files that do not exist physically on disk. When the WebSnap application tries to locate a file, the locate file service generates this event, along with an OnFindTemplateFile (see page 24889). One of these event handlers must be provided in order for the application to use the requested file.

See Also

OnFindIncludeFile (see page 24888)

OnFindTemplateFile (see page 24889)

179.1.9.3.3 TCustomLocateFileService.OnFindTemplateFile Event

Occurs when the WebSnap application tries to locate the template file for a content producer.

Pascal

```
property OnFindTemplateFile: TLocateFileServiceFindFileEvent;
```

C++

```
__property TLocateFileServiceFindFileEvent OnFindTemplateFile;
```

Description

Write an OnFindIncludeFile (see page 24888) event handler to indicate the location on disk of templates that a content producer component uses to generate its content. In WebSnap applications that include a locate file service component, you must supply either an OnFindTemplateFile or an OnFindStream (see page 24889) event handler (or both) if the application has content producers that use template files. Otherwise content producers can't locate their templates.

See Also

OnFindStream (see page 24889)

OnFindIncludeFile (see page 24888)

179.1.10 TCustomPageDispatcher Class

Dispatches incoming HTTP request messages in a WebSnap application.

Class Hierarchy

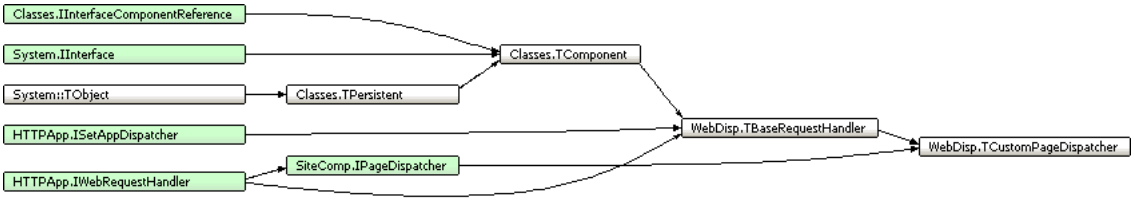

Pascal

```
TCustomPageDispatcher = class(TBaseRequestHandler, IPageDispatcher);
```

C++

```
class TCustomPageDispatcher : public TBaseRequestHandler, public IPageDispatcher;
```

File

WebDisp

Description

TCustomPageDispatcher is the base class for components that locate the appropriate page module for handling an incoming HTTP message and retrieve the content that the identified page module creates for the corresponding response message. Typically, the WebSnap application wizard adds a TCustomPageDispatcher descendant to the Web application module that it creates.

A WebSnap application only requires a single page dispatcher. The dispatcher examines all incoming HTTP request messages, checking the pathInfo portion of the URL to which a request is addressed for the name of a registered Web page module. It then forwards the request to the matching Web page module so that the page module can handle the request and produce the content of a response. If the request is not addressed to an URL with a pathinfo portion, the page dispatcher forwards the request to the Web page module identified by its DefaultPage (see page 24900) property.

See Also

TPageDispatcher (see page 24949)

179.1.10.1 TCustomPageDispatcher Members

The following tables list the members exposed by TCustomPageDispatcher.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DispatchPageName (see page 24894)	Checks whether a specified user has a given access rights string.

	EndUser (see page 24895)	This is EndUser, a member of class TCustomPageDispatcher.
	GetCanView (see page 24895)	Indicates whether the current user is permitted to view a specified Web page.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDefaultPageName (see page 24895)	Returns the value of the DefaultPageName property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetLoginRequired (see page 24896)	Indicates whether a user must log in before accessing a specified Web page.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequestedPage (see page 24896)	This is GetRequestedPage, a member of class TCustomPageDispatcher.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleRequest (see page 24864)	Handles an HTTP request.
	ImplDispatchPageName (see page 24896)	This is ImplDispatchPageName, a member of class TCustomPageDispatcher.
V	ImplGetCanView (see page 24896)	This is ImplGetCanView, a member of class TCustomPageDispatcher.
V	ImplGetDefaultPageName (see page 24897)	This is ImplGetDefaultPageName, a member of class TCustomPageDispatcher.
V	ImplGetLoginRequired (see page 24897)	This is ImplGetLoginRequired, a member of class TCustomPageDispatcher.
V	ImplHandleRequest (see page 24897)	Handles application-specific processing of an HTTP request.
V	ImplPageNameOfRequest (see page 24897)	This is ImplPageNameOfRequest, a member of class TCustomPageDispatcher.
	ImplRedirectToPageName (see page 24898)	This is ImplRedirectToPageName, a member of class TCustomPageDispatcher.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsLoggedIn (see page 24898)	This is IsLoggedIn, a member of class TCustomPageDispatcher.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PageNameOfRequest (see page 24898)	Returns the name of the Web page to which an HTTP request is addressed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceLoginPage (see page 24899)	This is ProduceLoginPage, a member of class TCustomPageDispatcher.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RedirectToPageName (see page 24899)	Fills out a response message so that it redirects the caller to another page in the application.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 24864)	Sets the AppDispatcher (see page 24865) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidatePage (see page 24899)	This is ValidatePage, a member of class TCustomPageDispatcher.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	AppDispatcher (see page 24865)	Provides read-only access to the application dispatcher component.

Public Events

	Name	Description
	OnAfterDispatchPage (see page 24901)	Occurs after the page dispatcher successfully dispatches an HTTP request message.
	OnAfterRedirectToPage (see page 24901)	Occurs after the page dispatcher successfully dispatches an HTTP request message in response to an internal redirection request.
	OnBeforeDispatchPage (see page 24902)	Occurs before the page dispatcher passes an HTTP request to a Web page module for processing.
	OnBeforeRedirectToPage (see page 24902)	Occurs before the page dispatcher dispatches an HTTP request message in response to an internal redirection request.
	OnCanViewPage (see page 24903)	Occurs just before the dispatcher passes a request message to a Web page module.
	OnIsLoginRequired (see page 24904)	Occurs when a request arrives from a caller that has not yet logged in.
	OnPageAccessDenied (see page 24904)	Occurs when the page dispatcher fails to validate the caller's right to access a requested Web page.
	OnPageNotFound (see page 24905)	Occurs when the dispatcher can't locate a registered Web page module with the name specified in an HTTP request message.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandleRequest (see page 15238)	Processes an HTTP request message.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DefaultPage (see page 24900)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.
R	DefaultPageName (see page 19731)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

179.1.10.2 TCustomPageDispatcher Methods

The methods of the TCustomPageDispatcher class are listed here.

Protected Methods

	Name	Description
	DispatchPageName (see page 24894)	Checks whether a specified user has a given access rights string.
	EndUser (see page 24895)	This is EndUser, a member of class TCustomPageDispatcher.
	GetCanView (see page 24895)	Indicates whether the current user is permitted to view a specified Web page.
	GetDefaultPageName (see page 24895)	Returns the value of the DefaultPageName property.
	GetLoginRequired (see page 24896)	Indicates whether a user must log in before accessing a specified Web page.
	GetRequestedPage (see page 24896)	This is GetRequestedPage, a member of class TCustomPageDispatcher.
	ImplDispatchPageName (see page 24896)	This is ImplDispatchPageName, a member of class TCustomPageDispatcher.
V	ImplGetCanView (see page 24896)	This is ImplGetCanView, a member of class TCustomPageDispatcher.
V	ImplGetDefaultPageName (see page 24897)	This is ImplGetDefaultPageName, a member of class TCustomPageDispatcher.
V	ImplGetLoginRequired (see page 24897)	This is ImplGetLoginRequired, a member of class TCustomPageDispatcher.
V	ImplHandleRequest (see page 24897)	Handles application-specific processing of an HTTP request.
V	ImplPageNameOfRequest (see page 24897)	This is ImplPageNameOfRequest, a member of class TCustomPageDispatcher.
	ImplRedirectToPageName (see page 24898)	This is ImplRedirectToPageName, a member of class TCustomPageDispatcher.
	IsLoggedIn (see page 24898)	This is IsLoggedIn, a member of class TCustomPageDispatcher.
	PageNameOfRequest (see page 24898)	Returns the name of the Web page to which an HTTP request is addressed.
	ProduceLoginPage (see page 24899)	This is ProduceLoginPage, a member of class TCustomPageDispatcher.
	RedirectToPageName (see page 24899)	Fills out a response message so that it redirects the caller to another page in the application.
	ValidatePage (see page 24899)	This is ValidatePage, a member of class TCustomPageDispatcher.

179.1.10.2.1 TCustomPageDispatcher.DispatchPageName Method

Checks whether a specified user has a given access rights string.

Pascal

```
function DispatchPageName(const APageName: string; AResponse: TWebResponse; AFlags:
TDispatchPageFlags): Boolean;
```

C++

```
__fastcall Boolean DispatchPageName(const AnsiString APageName, TWebResponse AResponse,
TDispatchPageFlags AFlags);
```

Description

Call DispatchPageName to fill out a response message with the Web page that a specified Web page module provides. DispatchPageName locates the Web page module that produces the requested Web page and uses it to fill out the content of a response object.

APageName is the name of the Web page to fetch. If APageName is an empty string, DispatchPageName uses the first Web page module that is registered with the current Web context.

AResponse is the response object whose Content property receives the requested Web page.

AFlags indicates the attributes that must be present on the Web page module. For example, if AFlags includes dpPublished, APageName must specify a published Web page module or the page dispatcher won't fetch the Web page.

DispatchPageName returns true if it successfully fetches the requested page, false if it fails to obtain the requested page.

See Also

RedirectToPageName (see page 19730)

HandleRequest (see page 15238)

Content (see page 15207)

WebContext (see page 24687)

179.1.10.2.2 TCustomPageDispatcher.EndUser Method

Pascal

```
function EndUser: IWebEndUser;
```

C++

```
__fastcall IWebEndUser EndUser();
```

Description

This is EndUser, a member of class TCustomPageDispatcher.

179.1.10.2.3 TCustomPageDispatcher.GetCanView Method

Indicates whether the current user is permitted to view a specified Web page.

Pascal

```
function GetCanView(const APageName: string): Boolean;
```

C++

```
__fastcall Boolean GetCanView(const AnsiString APageName);
```

Description

Call GetCanView to check whether the current user should be allowed to view a specified Web page. Typically, this method is called after the user logs in.

APageName is the name of the Web page the user wants to view.

See Also

GetLoginRequired (see page 19729)

HasViewRights (see page 24676)

OnCanViewPage (see page 24903)

179.1.10.2.4 TCustomPageDispatcher.GetDefaultPageName Method

Returns the value of the DefaultPageName property.

Pascal

```
function GetDefaultPageName: string;
```

C++

```
__fastcall AnsiString GetDefaultPageName();
```

Description

GetDefaultPageName is the read implementation of the DefaultPageName property.

See Also

DefaultPageName (see page 19731)

179.1.10.2.5 TCustomPageDispatcher.GetLoginRequired Method

Indicates whether a user must log in before accessing a specified Web page.

Pascal

```
function GetLoginRequired(const APageName: string): Boolean;
```

C++

```
__fastcall Boolean GetLoginRequired(const AnsiString APageName);
```

Description

Call GetLoginRequired to check whether users must log in before accessing a specified Web page. GetLoginRequired can check the Web page information object for the target Web page, but can also use other means (such as an event handler) to obtain this information.

APageName is the name of the Web page to check whether it requires users to first log in.

See Also

GetCanView (see page 19729)

IsLoginRequired

OnIsLoginRequired (see page 24904)

179.1.10.2.6 TCustomPageDispatcher.GetRequestedPage Method

Pascal

```
function GetRequestedPage(const APageName: string; AFlags: TDispatchPageFlags): TObject;
```

C++

```
__fastcall TObject * GetRequestedPage(const AnsiString APageName, TDispatchPageFlags AFlags);
```

Description

This is GetRequestedPage, a member of class TCustomPageDispatcher.

179.1.10.2.7 TCustomPageDispatcher.ImplDispatchPageName Method

Pascal

```
function ImplDispatchPageName(const APageName: string; AResponse: TWebResponse; AFlags:
TDispatchPageFlags): Boolean;
```

C++

```
__fastcall Boolean ImplDispatchPageName(const AnsiString APageName, TWebResponse AResponse,
TDispatchPageFlags AFlags);
```

Description

This is ImplDispatchPageName, a member of class TCustomPageDispatcher.

179.1.10.2.8 TCustomPageDispatcher.ImplGetCanView Method

Pascal

```
function ImplGetCanView(const APageName: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetCanView(const AnsiString APageName);
```

Description

This is ImplGetCanView, a member of class TCustomPageDispatcher.

179.1.10.2.9 TCustomPageDispatcher.ImplGetDefaultPageName Method

Pascal

```
function ImplGetDefaultPageName: string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplGetDefaultPageName();
```

Description

This is ImplGetDefaultPageName, a member of class TCustomPageDispatcher.

179.1.10.2.10 TCustomPageDispatcher.ImplGetLoginRequired Method

Pascal

```
function ImplGetLoginRequired(const APageName: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetLoginRequired(const AnsiString APageName);
```

Description

This is ImplGetLoginRequired, a member of class TCustomPageDispatcher.

179.1.10.2.11 TCustomPageDispatcher.ImplHandleRequest Method

Handles application-specific processing of an HTTP request.

Pascal

```
function ImplHandleRequest(Request: TWebRequest; Response: TWebResponse): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest(TWebRequest Request, TWebResponse Response);
```

Description

The ImplHandleRequest function is called from HandleRequest to handle the application-specific processing of the HTTP request.

This base version of the function generates an assertion failure as a reminder that a customized overridden version of this function needs to be defined in order to handle the request processing specific to the application.

See Also

HandleRequest (see page 24864)

179.1.10.2.12 TCustomPageDispatcher.ImplPageNameOfRequest Method

Pascal

```
function ImplPageNameOfRequest(Request: TWebRequest): string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplPageNameOfRequest(TWebRequest Request);
```

Description

This is ImplPageNameOfRequest, a member of class TCustomPageDispatcher.

179.1.10.2.13 TCustomPageDispatcher.ImplRedirectToPageName Method

Pascal

```
function ImplRedirectToPageName(const APageName: string; AParams: TStrings; AResponse:
TWebResponse; AFlags: TDispatchPageFlags): Boolean;
```

C++

```
__fastcall Boolean ImplRedirectToPageName(const AnsiString APageName, TStrings AParams,
TWebResponse AResponse, TDispatchPageFlags AFlags);
```

Description

This is ImplRedirectToPageName, a member of class TCustomPageDispatcher.

179.1.10.2.14 TCustomPageDispatcher.IsLoggedIn Method

Pascal

```
function IsLoggedIn: Boolean;
```

C++

```
__fastcall Boolean IsLoggedIn();
```

Description

This is IsLoggedIn, a member of class TCustomPageDispatcher.

179.1.10.2.15 TCustomPageDispatcher.PageNameOfRequest Method

Returns the name of the Web page to which an HTTP request is addressed.

Pascal

```
function PageNameOfRequest(Request: TWebRequest): string;
```

C++

```
__fastcall AnsiString PageNameOfRequest(TWebRequest Request);
```

Description

Call PageNameOfRequest to obtain the name of the Web page that a request message is trying to fetch.

ARequest is the request object that describes the HTTP request message.

PageNameOfRequest returns the last string in the path info portion of the request message's target URL.

See Also

DefaultPageName (see page 19731)

179.1.10.2.16 TCustomPageDispatcher.ProduceLoginPage Method

Pascal

```
function ProduceLoginPage: Boolean;
```

C++

```
__fastcall Boolean ProduceLoginPage();
```

Description

This is ProduceLoginPage, a member of class TCustomPageDispatcher.

179.1.10.2.17 TCustomPageDispatcher.RedirectToPageName Method

Fills out a response message so that it redirects the caller to another page in the application.

Pascal

```
function RedirectToPageName(const APageName: string; AParams: TStrings; AResponse:
TWebResponse; AFlags: TDispatchPageFlags): Boolean;
```

C++

```
__fastcall Boolean RedirectToPageName(const AnsiString APageName, TStrings AParams,
TWebResponse AResponse, TDispatchPageFlags AFlags);
```

Description

Call RedirectToPageName to generate a response that redirects the caller to another Web page in the WebSnap application.

APageName is the name of the Web page to which the request should be redirected. If APageName is an empty string, RedirectToPageName uses the first Web page module that is registered with the current Web context.

AParams is a string list containing application-defined name-value pairs. These name-value pairs supply the query fields of the redirected request message, and are added to the URL that is the target of the SendRedirect call.

AResponse is the response object that the WebSnap uses to make the SendRedirect call.

AFlags indicates the attributes that must be present on the Web page module to which the current request is redirected. For example, if AFlags includes dpPublished, APageName must specify a published Web page module.

RedirectToPageName returns true if it successfully redirects to the specified page.

See Also

DispatchPageName (see page 19728)

HandleRequest (see page 15238)

Content (see page 15207)

WebContext (see page 24687)

179.1.10.2.18 TCustomPageDispatcher.ValidatePage Method

Pascal

```
function ValidatePage(const APage: string; var AHandled: Boolean): Boolean;
```

C++

```
__fastcall Boolean ValidatePage(const AnsiString APage, Boolean AHandled);
```

Description

This is ValidatePage, a member of class TCustomPageDispatcher.

179.1.10.3 TCustomPageDispatcher Properties

The properties of the TCustomPageDispatcher class are listed here.

Public Properties

	Name	Description
	DefaultPage (see page 24900)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

179.1.10.3.1 TCustomPageDispatcher.DefaultPage Property

Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

Pascal

```
property DefaultPage: string;
```

C++

```
__property AnsiString DefaultPage;
```

Description

DefaultPage is the name of the registered Web page module that should handle incoming request messages that do not have a pathinfo portion on their target URL. This is the name that is supplied to the Web page module wizard when you created the default Web page module. It forms the basis of that Web page module's class name. For example, if the default Web page module has the name MyDefaultPage, the Web page module class is of type TMyDefaultPage.

If DefaultPage is an empty string, the page dispatcher uses the Web application module as a default Web page module, as long as it is a Web Page module.

179.1.10.4 TCustomPageDispatcher Events

The events of the TCustomPageDispatcher class are listed here.

Public Events

	Name	Description
	OnAfterDispatchPage (see page 24901)	Occurs after the page dispatcher successfully dispatches an HTTP request message.
	OnAfterRedirectToPage (see page 24901)	Occurs after the page dispatcher successfully dispatches an HTTP request message in response to an internal redirection request.
	OnBeforeDispatchPage (see page 24902)	Occurs before the page dispatcher passes an HTTP request to a Web page module for processing.
	OnBeforeRedirectToPage (see page 24902)	Occurs before the page dispatcher dispatches an HTTP request message in response to an internal redirection request.
	OnCanViewPage (see page 24903)	Occurs just before the dispatcher passes a request message to a Web page module.
	OnIsLoginRequired (see page 24904)	Occurs when a request arrives from a caller that has not yet logged in.
	OnPageAccessDenied (see page 24904)	Occurs when the page dispatcher fails to validate the caller's right to access a requested Web page.
	OnPageNotFound (see page 24905)	Occurs when the dispatcher can't locate a registered Web page module with the name specified in an HTTP request message.

179.1.10.4.1 TCustomPageDispatcher.OnAfterDispatchPage Event

Occurs after the page dispatcher successfully dispatches an HTTP request message.

Pascal

```
property OnAfterDispatchPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterDispatchPage;
```

Description

Write an OnAfterDispatchPage event handler to take specific action after an HTTP request has been successfully handled by a Web page module. For example, an OnAfterDispatchPage event handler can maintain a log of all Web traffic handled by the WebSnap application.

OnAfterDispatchPage occurs only if the target Web page module succeeds in handling the request. If the target Web page module could not be validated, an OnPageAccessDenied (see page 24904) event occurs instead. If the page could not be located, an OnPageNotFound (see page 24905) event occurs instead.

See Also

OnBeforeDispatchPage (see page 24902)

OnPageNotFound (see page 24905)

OnCanViewPage (see page 24903)

OnIsLoginRequired (see page 24904)

OnPageAccessDenied (see page 24904)

179.1.10.4.2 TCustomPageDispatcher.OnAfterRedirectToPage Event

Occurs after the page dispatcher successfully dispatches an HTTP request message in response to an internal redirection request.

Pascal

```
property OnAfterRedirectToPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterRedirectToPage;
```

Description

Write an OnAfterRedirectToPage event handler to take specific action after an HTTP request has been successfully redirected to a Web page module. This event only occurs if an adapter in the WebSnap application executes an adapter action whose options include a redirection (roRedirectHTTPGet or roRedirectHTTPPost), and the new target Web page module succeeds in handling the request.

See Also

OnBeforeRedirectToPage (see page 24902)

OnAfterDispatchPage (see page 24901)

RedirectOptions (see page 24074)

179.1.10.4.3 TCustomPageDispatcher.OnBeforeDispatchPage Event

Occurs before the page dispatcher passes an HTTP request to a Web page module for processing.

Pascal

```
property OnBeforeDispatchPage: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnBeforeDispatchPage;
```

Description

Write an OnBeforeDispatchPage event handler to process an HTTP request message before the dispatcher forwards it to the target Web page module for processing.

This event can provide an alternate mechanism for handling request messages, and then set the Handled parameter to true to prevent the page dispatcher from locating the target Web page module, verifying that it can handle the request, and forwarding the request to that Web page module.

If the event handler does not handle the request (for example, if it is merely maintaining a log of incoming requests), it can leave the Handled parameter as false and the dispatcher will continue with the default processing.

Tip: To access the Web request and response objects from within an OnBeforeDispatchPage event handler, call the global WebContext function to access the Web context and read its Request and Response properties.

See Also

OnAfterDispatchPage (see page 24901)

OnPageNotFound (see page 24905)

OnCanViewPage (see page 24903)

OnIsLoginRequired (see page 24904)

OnPageAccessDenied (see page 24904)

WebContext (see page 24687)

Request (see page 24658)

Response (see page 24659)

179.1.10.4.4 TCustomPageDispatcher.OnBeforeRedirectToPage Event

Occurs before the page dispatcher dispatches an HTTP request message in response to an internal redirection request.

Pascal

```
property OnBeforeRedirectToPage: TDispatchPageParamsHandledEvent;
```

C++

```
__property TDispatchPageParamsHandledEvent OnBeforeRedirectToPage;
```

Description

Write an OnBeforeRedirectToPage event handler to take specific action before an HTTP request is redirected to a Web page module. This event only occurs if an adapter in the WebSnap application executes an adapter action whose options include a redirection (roRedirectHTTPGet or roRedirectHTTPPost).

If the event handler sets Handled to true, the dispatcher assumes it has handled the request and performs no further processing. Otherwise, after the event handler exits, the page dispatcher calls on the Web Module to set up the response message and call

SendRedirect.

Tip: To access the Web request and response objects from within an OnBeforeRedirectToPage event handler, call the global WebContext function to access the Web context and read its Request and Response properties.

See Also

OnAfterRedirectToPage (see page 24901)

OnBeforeDispatchPage (see page 24902)

WebContext (see page 24687)

Request (see page 24658)

Response (see page 24659)

SendRedirect (see page 15201)

179.1.10.4.5 TCustomPageDispatcher.OnCanViewPage Event

Occurs just before the dispatcher passes a request message to a Web page module.

Pascal

```
property OnCanViewPage: TCanViewPageEvent;
```

C++

```
__property TCanViewPageEvent OnCanViewPage;
```

Description

Write an OnCanViewPage event handler to customize the way the WebSnap application determines whether a given request can be passed to a Web page module for processing. This event is part of the mechanism by which the dispatcher validates a Web page for viewing. The validation process, which occurs after the OnBeforeDispatchPage (see page 24902) event, occurs as follows:

1. If the caller has not already logged in, the dispatcher generates an OnIsLoginRequired (see page 24904) event to check whether the specified Web page requires the caller to log in.
2. If a login is required, the dispatcher does not validate the page (and so does not forward the request to the specified Web page module). Rather, it initiates the process of creating a response to allow the user to log in.

a. First, the dispatcher generates an OnPageAccessDenied (see page 24904) event, where the application can handle the login process.

b. If the OnPageAccessDenied (see page 24904) event handler does not handle the login process, the dispatcher forwards the request to the login page specified by the end user adapter's LoginPage property.

3. If a login is not required (either because the caller already logged in or because there is no need to log in), the dispatcher generates an OnCanViewPage event. This event can validate a caller that has logged in, or provide any other type of validation that governs access to the requested page.
4. If there is no OnCanViewPage event handler, or if the OnCanViewPage event handler does not handle the validation, the dispatcher calls on a TEndUserAdapter component to check the user's access rights.

   Tip: To access the Web request and response objects from within an OnCanViewPage event handler, call the global WebContext function to access the Web context and read its Request and Response properties. To access information about the access rights that the target page requires, call the Web context's FindPageInfo method to obtain a Web page info object for the target page and check its ViewAccess property.

See Also

OnIsLoginRequired (see page 24904)

OnBeforeDispatchPage (see page 24902)

OnPageAccessDenied (see page 24904)

WebContext (see page 24687)

Request (see page 24658)

Response (see page 24659)

ViewAccess (see page 15069)

FindPageInfo (see page 24650)

179.1.10.4.6 TCustomPageDispatcher.OnIsLoginRequired Event

Occurs when a request arrives from a caller that has not yet logged in.

Pascal

```
property OnIsLoginRequired: TLoginRequiredEvent;
```

C++

```
__property TLoginRequiredEvent OnIsLoginRequired;
```

Description

Write an OnIsLoginRequired event handler to customize the way the WebSnap application determines whether a caller must log in before obtaining access to a requested page. This event occurs after the OnBeforeDispatchPage (see page 24902) event if the caller has not already logged in to the WebSnap application.

If there is no OnPageAccessDenied (see page 24904) event handler, or if the OnPageAccessDenied (see page 24904) event handler does not handle the login process, the dispatcher obtains a response message from the login page specified by the end user adapter.

See Also

OnCanViewPage (see page 24903)

OnBeforeDispatchPage (see page 24902)

OnPageAccessDenied (see page 24904)

LoginPage (see page 24118)

179.1.10.4.7 TCustomPageDispatcher.OnPageAccessDenied Event

Occurs when the page dispatcher fails to validate the caller's right to access a requested Web page.

Pascal

```
property OnPageAccessDenied: TPageAccessDeniedEvent;
```

C++

```
__property TPageAccessDeniedEvent OnPageAccessDenied;
```

Description

Write an OnPageAccessDenied event handler to customize the response when a request for a specific Web page fails to be

validated.

Tip: To access the Web request and response objects from within an OnPageAccessDenied event handler, call the global WebContext function to access the Web context and read its Request and Response properties.

See Also

OnIsLoginRequired (see page 24904)

OnCanViewPage (see page 24903)

WebContext (see page 24687)

Request (see page 24658)

Response (see page 24659)

LoginPage (see page 24118)

179.1.10.4.8 TCustomPageDispatcher.OnPageNotFound Event

Occurs when the dispatcher can't locate a registered Web page module with the name specified in an HTTP request message.

Pascal

```
property OnPageNotFound: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnPageNotFound;
```

Description

Write an OnPageNotFound event handler to generate a response when the dispatcher can't locate the Web page module to which an HTTP request message is addressed. For example, an OnPageNotFound event handler can provide the response object with content that indicates that the requested page does not exist or that represents simple content for a page that does not have an associated Web page module.

If the OnPageNotFound event handler sets its Handled parameter to true, the page dispatcher assumes it has updated the response message and returns it. If the OnPageNotFound leaves the Handled parameter set to false, the dispatcher raises an EPageDispatcherException (see page 24839) exception after the event handler exits.

Tip: To access the Web request and response objects from within an OnPageNotFound event handler, call the global WebContext function to access the Web context and read its Request and Response properties.

See Also

OnBeforeDispatchPage (see page 24902)

OnAfterDispatchPage (see page 24901)

OnPageAccessDenied (see page 24904)

EPageDispatcherException (see page 24839)

WebContext (see page 24687)

Request (see page 24658)

Response (see page 24659)

179.1.11 TCustomWebAppComponents Class

TCustomWebAppComponents manages the components that provide a WebSnap application's functionality.

Class Hierarchy

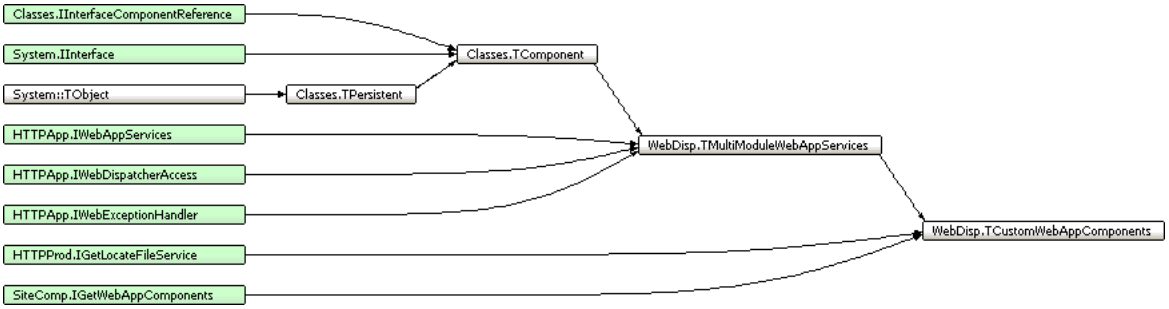

Pascal

```
TCustomWebAppComponents = class(TMultiModuleWebAppServices, IGetWebAppComponents,
IGetLocateFileService);
```

C++

```
class TCustomWebAppComponents : public TMultiModuleWebAppServices, public
IGetWebAppComponents, public IGetLocateFileService;
```

File

WebDisp

Description

Use TCustomWebAppComponents as a base class for the component that centralizes access to the main WebSnap components. It keeps track of the current request and response objects, maintains a list of the available Web modules, and locates the appropriate components to handle incoming request messages. TCustomWebAppComponents introduces properties that provide access to the components that perform many common tasks in a WebSnap application. These properties include:

PageDispatcher (see page 24921), which locates the Web page modules to process incoming request messages.

AdapterDispatcher (see page 24919), which encodes calls to adapter actions and fields, and which handles HTML form submissions and requests for dynamic images by calling adapter actions and fields.

Sessions (see page 24921), which manages the sessions for each caller.

ApplicationAdapter (see page 24919), which implements the fields and actions of the Application variable in server-side scripts.

EndUserAdapter (see page 24920), which implements the fields and actions of the EndUser variable in server-side scripts.

UserListService (see page 24922), which maintains a list of end users and their passwords and access privileges.

LocateFileService (see page 24920), which controls access to the templates and server-side script include files that are used while processing requests.

DispatchActions (see page 24919), which maintains a list of action items that can respond to request messages.

The components these properties reference are optional. The WebSnap Application wizard automatically creates components and fill in these properties for any components that you select.

See Also

TApplicationAdapter (see page 24032)

TEndUserAdapter (see page 24421)

TPageDispatcher (see page 24949)

TAdapterDispatcher (see page 24840)

TLocateFileService (see page 24928)

TSessionsService (see page 25582)

TWebUserList (see page 25625)

WebContext

179.1.11.1 TCustomWebAppComponents Members

The following tables list the members exposed by TCustomWebAppComponents.

Protected Events

	Name	Description
	OnAfterDispatch (see page 24948)	Occurs after the WebSnap application generates the content of a response message.
	OnBeforeDispatch (see page 24948)	Occurs before TMultiModuleWebAppServices (see page 24933) passes an HTTP request onto the other components in the application for processing.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	Access_Request (see page 24937)	Returns the value of the Access_Request property.
	Access_Response (see page 24938)	Returns the value of the Access_Response property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CreateContext (see page 24911)	Creates the Web context for the current request message.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoAfterDispatch (see page 24939)	Generates an OnAfterDispatch (see page 24948) event.
	DoBeforeDispatch (see page 24939)	Generates an OnBeforeDispatch (see page 24948) event.
V	DoneModule (see page 24939)	Performs any final tasks before deactivating a Web module.
	FinishContext (see page 24940)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetAdapterDispatcher (see page 24912)	Returns the value of the AdapterDispatcher (see page 24919) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetExceptionHandler (see page 24940)	Returns the object that handles exceptions that arise in the WebSnap application.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetLocateFileService (see page 24912)	Returns the value of the LocateFileService (see page 24920) property.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPageDispatcher (see page 24912)	Returns the value of the PageDispatcher (see page 24921) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebApplicationInfo (see page 24913)	Returns the value of the ApplicationAdapter (see page 24919) property.
	GetWebEndUser (see page 24913)	Returns the value of the EndUserAdapter (see page 24920) property.
	GetWebUserList (see page 24913)	Returns the value of the UserListService (see page 24922) property.
	HandleException (see page 24941)	Responds to any exceptions not caught by the WebSnap application.
	HandleRequest (see page 24941)	Directs the processing of the current request message.
	ImplFinishContext (see page 24914)	Provides the underlying implementation of the FinishContext (see page 24940) method.
	ImplHandleException (see page 24914)	Provides the underlying implementation of the HandleException (see page 24941) method.
	ImplHandleRequest (see page 24915)	Provides the underlying implementation for the HandleRequest (see page 24941) method.
	ImplInitContext (see page 24915)	Provides the underlying implementation for the InitContext (see page 24944) method.
	InitContext (see page 24944)	Creates and initializes the Web context for the current Web request.
	InitModule (see page 24916)	Initializes a Web module before the WebSnap application uses it to handle the current request.
	InitModules (see page 24917)	Initializes all the currently active Web modules.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	ListRequestHandlers (see page 24917)	Fills an interface list with the interfaces of all dispatcher components in the WebSnap application.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	ModuleAdded (see page 24946)	Initializes a Web module when it is activated while processing the current request.
	Notification (see page 24918)	Updates the TCustomWebAppComponents (see page 24906) properties when components are added to or removed from the application.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Request (see page 24946)	Provides access to the current request object.
R	Response (see page 24947)	Provides access to the current response object.
R	WebModules (see page 24947)	Lists the Web modules available in the WebSnap application.

Public Events

	Name	Description
	OnException (see page 24922)	Occurs when an exception escapes all try blocks in the WebSnap application.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FinishContext (see page 15228)	Performs any final tasks after the WebSnap application has finished processing a request.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetExceptionHandler (see page 15229)	Returns the object that handles exceptions that arise in the WebSnap application.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.

	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandleException (see page 15236)	This is HandleException, a member of class IWebExceptionHandler.
	HandleRequest (see page 15229)	Directs the processing of the current request message.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitContext (see page 15229)	Creates and initializes the Web context for the current Web request.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Request (see page 15235)	Returns the Web request object for the HTTP request message to which the application is responding.
	Response (see page 15235)	Returns the Web response object for the HTTP response message that the application is assembling.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	AdapterDispatcher (see page 24919)	Provides access to the current adapter dispatcher object.
	ApplicationAdapter (see page 24919)	Provides access to the application adapter.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DispatchActions (see page 24919)	Provides access to an action-based Web dispatcher.
	EndUserAdapter (see page 24920)	Provides access to an end-user adapter.
R	ExceptionHandler (see page 15230)	Indicates the object that handles exceptions that arise when processing a request message.
	LocateFileService (see page 24920)	Provides access to the file location service.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	PageDispatcher (see page 24921)	Provides access to the current page dispatcher object.
	Sessions (see page 24921)	Provides access to the sessions service.
	UserListService (see page 24922)	Provides access to the file location service.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebApplicationInfo (see page 19707)	This is WebApplicationInfo, a member of class IGetWebAppComponents.
R	WebEndUser (see page 19707)	This is WebEndUser, a member of class IGetWebAppComponents.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

179.1.11.2 TCustomWebAppComponents Methods

The methods of the TCustomWebAppComponents class are listed here.

Protected Methods

	Name	Description
	CreateContext (see page 24911)	Creates the Web context for the current request message.
	GetAdapterDispatcher (see page 24912)	Returns the value of the AdapterDispatcher (see page 24919) property.
	GetLocateFileService (see page 24912)	Returns the value of the LocateFileService (see page 24920) property.
	GetPageDispatcher (see page 24912)	Returns the value of the PageDispatcher (see page 24921) property.
	GetWebApplicationInfo (see page 24913)	Returns the value of the ApplicationAdapter (see page 24919) property.
	GetWebEndUser (see page 24913)	Returns the value of the EndUserAdapter (see page 24920) property.
	GetWebUserList (see page 24913)	Returns the value of the UserListService (see page 24922) property.
	ImplFinishContext (see page 24914)	Provides the underlying implementation of the FinishContext (see page 24940) method.
	ImplHandleException (see page 24914)	Provides the underlying implementation of the HandleException (see page 24941) method.
	ImplHandleRequest (see page 24915)	Provides the underlying implementation for the HandleRequest (see page 24941) method.
	ImplInitContext (see page 24915)	Provides the underlying implementation for the InitContext (see page 24944) method.
	InitModule (see page 24916)	Initializes a Web module before the WebSnap application uses it to handle the current request.
	InitModules (see page 24917)	Initializes all the currently active Web modules.
	ListRequestHandlers (see page 24917)	Fills an interface list with the interfaces of all dispatcher components in the WebSnap application.
	Notification (see page 24918)	Updates the TCustomWebAppComponents (see page 24906) properties when components are added to or removed from the application.

179.1.11.2.1 TCustomWebAppComponents.CreateContext Method

Creates the Web context for the current request message.

Pascal

```
function CreateContext: TAbstractWebContext; override;
```

C++

```
virtual __fastcall TAbstractWebContext CreateContext();
```

Description

The ImplInitContext (see page 24915) method calls CreateContext to instantiate the Web context object for the current request. This context can then be accessed by the WebSnap application using the global WebContext function.

In TCustomWebAppComponents (see page 24906), CreateContext creates a TWebContext object, creating a new session object for the current request and passing the session object along with the values of the protected Request (see page 24946), Response (see page 24947), and WebModules (see page 24947) properties to the TWebContext constructor.

See Also

WebContext (see page 24687)

ImplInitContext (see page 24915)

Create (see page 24662)

Request (see page 24946)

Response (see page 24947)

WebModules (see page 24947)

179.1.11.2.2 TCustomWebAppComponents.GetAdapterDispatcher Method

Returns the value of the AdapterDispatcher (see page 24919) property.

Pascal

```
function GetAdapterDispatcher: IAdapterDispatcher;
```

C++

```
__fastcall IAdapterDispatcher GetAdapterDispatcher();
```

Description

The protected GetAdapterDispatcher method is used internally to fetch the value of the AdapterDispatcher (see page 24919) property.

See Also

AdapterDispatcher (see page 24919)

179.1.11.2.3 TCustomWebAppComponents.GetLocateFileService Method

Returns the value of the LocateFileService (see page 24920) property.

Pascal

```
function GetLocateFileService: ILocateFileService;
```

C++

```
__fastcall ILocateFileService GetLocateFileService();
```

Description

The protected GetLocateFileService method is used internally to fetch the value of the LocateFileService (see page 24920) property.

See Also

LocateFileService (see page 24920)

179.1.11.2.4 TCustomWebAppComponents.GetPageDispatcher Method

Returns the value of the PageDispatcher (see page 24921) property.

Pascal

```
function GetPageDispatcher: IPageDispatcher;
```

C++

```
__fastcall IPageDispatcher GetPageDispatcher();
```

Description

The protected GetPageDispatcher method is used internally to fetch the value of the PageDispatcher (see page 24921) property.

See Also

PageDispatcher (see page 24921)

179.1.11.2.5 TCustomWebAppComponents.GetWebApplicationInfo Method

Returns the value of the ApplicationAdapter (see page 24919) property.

Pascal

```
function GetWebApplicationInfo: IWebApplicationInfo;
```

C++

```
__fastcall IWebApplicationInfo GetWebApplicationInfo();
```

Description

The protected GetWebApplicationInfo method is used internally to fetch the value of the ApplicationAdapter (see page 24919) property.

See Also

ApplicationAdapter (see page 24919)

179.1.11.2.6 TCustomWebAppComponents.GetWebEndUser Method

Returns the value of the EndUserAdapter (see page 24920) property.

Pascal

```
function GetWebEndUser: IWebEndUser;
```

C++

```
__fastcall IWebEndUser GetWebEndUser();
```

Description

The protected GetWebEndUser method is used internally to fetch the value of the EndUserAdapter (see page 24920) property.

See Also

EndUserAdapter (see page 24920)

179.1.11.2.7 TCustomWebAppComponents.GetWebUserList Method

Returns the value of the UserListService (see page 24922) property.

Pascal

```
function GetWebUserList: IWebUserList;
```

C++

```
__fastcall IWebUserList GetWebUserList();
```

Description

The protected GetWebUserList method is used internally to fetch the value of the UserListService (see page 24922) property.

See Also

UserListService (see page 24922)

179.1.11.2.8 TCustomWebAppComponents.ImplFinishContext Method

Provides the underlying implementation of the FinishContext (see page 24940) method.

Pascal

```
procedure ImplFinishContext; override;
```

C++

```
virtual __fastcall ImplFinishContext();
```

Description

ImplFinishContext performs any final tasks and cleanup after the WebSnap application has finished processing a request message.

ImplFinishContext

1. Instructs the Web dispatcher that implements the DispatchActions (see page 24919) property to perform any final tasks.
2. Calls the DoneModule (see page 24939) method for every active Web module so that it can be freed or deactivated.
3. Frees the Web context for the current request.
4. Adds the session ID of the current session to the response message so that subsequent requests from the same caller can use the same session.

See Also

FinishContext (see page 24940)

DoneModule (see page 24939)

WebContext (see page 24687)

DispatchActions (see page 24919)

Sessions (see page 24921)

179.1.11.2.9 TCustomWebAppComponents.ImplHandleException Method

Provides the underlying implementation of the HandleException (see page 24941) method.

Pascal

```
procedure ImplHandleException(E: Exception; var Handled: Boolean); override;
```

C++

```
virtual __fastcall ImplHandleException(Exception E, Boolean Handled);
```

Description

The HandleException (see page 24941) method calls ImplHandleException to handle any exception that passes through all the try blocks in the application code.

ImplHandleException generates an OnException (see page 24922) event, so that an event handler can take care of the exception. If the OnException (see page 24922) event handler does not handle the exception, ImplHandleException checks whether the response message has not yet been sent. If not, it adds the exception error message to the response.

E is the exception object for the exception that occurred.

Handled returns true if the OnException (see page 24922) event handler handled the exception or if ImplHandleException added the error message to the response object. If Handled returns false, the WebSnap application displays a dialog box that indicates an error occurred.

See Also

HandleException (see page 24941)

OnException (see page 24922)

179.1.11.2.10 TCustomWebAppComponents.ImplHandleRequest Method

Provides the underlying implementation for the HandleRequest (see page 24941) method.

Pascal

```
function ImplHandleRequest: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest();
```

Description

The HandleRequest (see page 24941) method calls ImplHandleRequest to handle the current HTTP request message.

ImpleHandleRequest

1.Generates an OnBeforeDispatch (see page 24948) event.

2.If the OnBeforeDispatch (see page 24948) event handler does not handle the request, checks the PageDispatcher (see page 24921), AdapterDispatcher (see page 24919), and DispatchActions (see page 24919) properties to find a dispatcher that can handle the request. If it finds one, the dispatcher forwards the request to the appropriate components for handling.

3.If the request was handled by the OnBeforeDispatch (see page 24948) event or by one of the dispatchers, and if the response was not already sent to the caller, generates an OnAfterDispatch (see page 24948) event.

ImplHandleRequest returns true if the request is handled and the response should be sent back to the caller (if it was not already sent). It returns false if the WebSnap application could not handle the request.

See Also

HandleRequest (see page 24941)

OnBeforeDispatch (see page 24948)

OnAfterDispatch (see page 24948)

PageDispatcher (see page 24921)

AdapterDispatcher (see page 24919)

DispatchActions (see page 24919)

179.1.11.2.11 TCustomWebAppComponents.ImplInitContext Method

Provides the underlying implementation for the InitContext (see page 24944) method.

Pascal

```
procedure ImplInitContext(AWebModuleList: TAbstractWebModuleList; ARequest: TWebRequest;
AResponse: TWebResponse); override;
```

C++

```
virtual __fastcall ImplInitContext(TAbstractWebModuleList AWebModuleList, TWebRequest
ARequest, TWebResponse AResponse);
```

Description

The InitContext (see page 24944) method, which is called when the WebSnap application first receives a request, calls ImplInitContext to perform any initializations and to create the Web context for the new request message.

ImplInitContext performs the following tasks:

1. If the DispatchActions (see page 24919) property is set, instructs the Web dispatcher to initialize itself.
2. Calls the CreateContext (see page 24911) method to create a Web context for the current request, supplying it with the values of the AWebModuleList, ARequest, and AResponse parameters. This Web context can then be accessed using the global WebContext function.
3. Calls the InitModules (see page 24917) method to initialize all the Web modules that the AWebModuleList parameter indicates are currently instantiated. (These are the Web modules that are created for every request message)
4. Hooks up the InitModule (see page 24916) method so that it can initialize any Web modules that are instantiated in the course of processing the current request.

See Also

InitContext (see page 24944)

CreateContext (see page 24911)

WebContext (see page 24687)

Request (see page 24658)

Response (see page 24659)

WebModules (see page 24660)

InitModules (see page 24917)

DispatchActions (see page 24919)

InitModule (see page 24916)

179.1.11.2.12 TCustomWebAppComponents.InitModule Method

Initializes a Web module before the WebSnap application uses it to handle the current request.

Pascal

```
procedure InitModule(AModule: TComponent); override;
```

C++

```
virtual __fastcall InitModule(TComponent * AModule);
```

Description

InitModule is called whenever the WebSnap application instantiates a Web module to handle the current request. InitModule handles any initializations, including assigning itself as the Web dispatcher to any content producers in the Web module, generating an OnActivate event on the Web module, and allowing the Web dispatcher that is the value of the DispatchActions (see page 24919) property to perform its own initializations.

AWebModule is the Web module component to initialize.

See Also

InitModules (see page 24917)

Dispatcher (see page 15109)

DispatchActions (see page 24919)

OnActivate (see page 25356)

179.1.11.2.13 TCustomWebAppComponents.InitModules Method

Initializes all the currently active Web modules.

Pascal

```
procedure InitModules; override;
```

C++

```
virtual __fastcall InitModules();
```

Description

InitModules is called to initialize the currently active Web modules when the WebSnap application first sets up the Web context for a new request. It iterates over all the active Web modules that the WebModules (see page 24947) property lists, calling the InitModule (see page 24916) method to initialize each one. It then checks whether the session for the request is new (rather than a continuation of the session from a previous request), and if so, generates an OnStartSession event on the sessions service.

See Also

InitModule (see page 24916)

WebModules (see page 24947)

InitContext (see page 24944)

OnStartSession (see page 25582)

179.1.11.2.14 TCustomWebAppComponents.ListRequestHandlers Method

Fills an interface list with the interfaces of all dispatcher components in the WebSnap application.

Pascal

```
procedure ListRequestHandlers(L: IInterfaceList); virtual;
```

C++

```
virtual __fastcall ListRequestHandlers(IInterfaceList L);
```

Description

The ImplHandleRequest (see page 24915) method calls ListRequestHandlers to obtain a list of all interfaces it can call to dispatch the request. ListRequestHandlers adds the interfaces that are the values of the PageDispatcher (see page 24921), AdapterDispatcher (see page 24919), and DispatchActions (see page 24919) properties to the list.

L is the interface list to which the dispatcher interfaces should be added.

See Also

ImplHandleRequest (see page 24915)

PageDispatcher (see page 24921)

AdapterDispatcher (see page 24919)

DispatchActions (see page 24919)

179.1.11.2.15 TCustomWebAppComponents.Notification Method

Updates the TCustomWebAppComponents (see page 24906) properties when components are added to or removed from the application.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. It checks whether AComponent supports one of the interfaces that is the type of a TCustomWebAppComponents (see page 24906) property, and if so, sets that property to nil (Delphi) or NULL (C++) if the object is about to be removed, or sets the property to the AComponent interface if the object was just added.

See Also

Notification (see page 2783)

AdapterDispatcher (see page 24919)

ApplicationAdapter (see page 24919)

DispatchActions (see page 24919)

EndUserAdapter (see page 24920)

LocateFileService (see page 24920)

Sessions (see page 24921)

PageDispatcher (see page 24921)

UserListService (see page 24922)

179.1.11.3 TCustomWebAppComponents Properties

The properties of the TCustomWebAppComponents class are listed here.

Public Properties

	Name	Description
	AdapterDispatcher (see page 24919)	Provides access to the current adapter dispatcher object.
	ApplicationAdapter (see page 24919)	Provides access to the application adapter.
	DispatchActions (see page 24919)	Provides access to an action-based Web dispatcher.
	EndUserAdapter (see page 24920)	Provides access to an end-user adapter.
	LocateFileService (see page 24920)	Provides access to the file location service.
	PageDispatcher (see page 24921)	Provides access to the current page dispatcher object.
	Sessions (see page 24921)	Provides access to the sessions service.
	UserListService (see page 24922)	Provides access to the file location service.

179.1.11.3.1 TCustomWebAppComponents.AdapterDispatcher Property

Provides access to the current adapter dispatcher object.

Pascal

```
property AdapterDispatcher: IAdapterDispatcher;
```

C++

```
__property IAdapterDispatcher AdapterDispatcher;
```

Description

Use AdapterDispatcher to get or set the interface of the adapter dispatcher. The adapter dispatcher is responsible for encoding the hidden fields and HREF values that represent calls to adapter fields and actions, and for interpreting the fields in incoming requests by calling adapters to execute their actions or to fetch image field values.

Typically, AdapterDispatcher is the interface of a TAdapterDispatcher (see page 24840) object. If the WebSnap application does not include a TAdapterDispatcher (see page 24840) component, or some other component that implements the IAdapterDispatcher interface, the value of this property is nil (Delphi) or NULL (C++). When AdapterDispatcher is nil (Delphi) or NULL (C++), the WebSnap application can't execute adapter actions or fetch image fields from adapters.

See Also

TAdapterDispatcher (see page 24840)

DispatchActions (see page 24919)

179.1.11.3.2 TCustomWebAppComponents.ApplicationAdapter Property

Provides access to the application adapter.

Pascal

```
property ApplicationAdapter: IWebApplicationInfo;
```

C++

```
__property IWebApplicationInfo ApplicationAdapter;
```

Description

Use ApplicationAdapter to get or set the interface of the application adapter. The application adapter implements the fields and actions of the Application variable in server-side scripts.

Typically, ApplicationAdapter is the interface of a TApplicationAdapter object. If the WebSnap application does not include a TApplicationAdapter component, or some other component that implements the IWebApplicationInfo interface, the value of this property is nil (Delphi) or NULL (C++). When ApplicationAdapter is nil (Delphi) or NULL (C++), the WebSnap application can't use the Application variable in server-side scripts.

See Also

TApplicationAdapter (see page 24032)

179.1.11.3.3 TCustomWebAppComponents.DispatchActions Property

Provides access to an action-based Web dispatcher.

Pascal

```
property DispatchActions: IWebDispatchActions;
```

C++

```
__property IWebDispatchActions DispatchActions;
```

Description

Read DispatchActions to determine whether the WebSnap application contains a Web dispatcher. A Web dispatcher maintains a list of action items that respond to incoming request messages. Web dispatchers can appear in both WebSnap and Web broker applications.

The IWebDispatchActions interface that is the value of DispatchActions does nothing special other than identify its component as a Web dispatcher. You can, however, use this interface to obtain other interfaces that belong to the Web dispatcher.

Typically, DispatchActions is the interface of a TWebDispatcher object. If the WebSnap application does not include a TWebDispatcher component, or some other component that implements the IWebDispatchActions interface, the value of this property is nil (Delphi) or NULL (C++).

See Also

TWebModule (see page 15154)

TWebDispatcher (see page 15147)

179.1.11.3.4 TCustomWebAppComponents.EndUserAdapter Property

Provides access to an end-user adapter.

Pascal

```
property EndUserAdapter: IWebEndUser;
```

C++

```
__property IWebEndUser EndUserAdapter;
```

Description

Use EndUserAdapter to get or set the interface of the end user adapter. An end user adapter maintains information about the current end user, and supports the actions for logging users in and out. End user adapters implement the fields and actions of the EndUser variable in server-side scripts.

Typically, EndUserAdapter is the interface of a TEndUserAdapter or TEndUserSessionAdapter object. If the WebSnap application does not include a component that implements the IWebEndUser interface, the value of this property is nil (Delphi) or NULL (C++). When EndUserAdapter is nil (Delphi) or NULL (C++), the WebSnap application can't use the EndUser variable in server-side scripts or take advantage of the login support that end user adapters provide.

See Also

TEndUserAdapter (see page 24421)

TEndUserSessionAdapter (see page 24458)

179.1.11.3.5 TCustomWebAppComponents.LocateFileService Property

Provides access to the file location service.

Pascal

```
property LocateFileService: ILocateFileService;
```

C++

```
__property ILocateFileService LocateFileService;
```

Description

Use LocateFileService to get or set the interface of the file location service. The file location service controls access to the templates and server-side script include files that are used while processing requests.

Typically, LocateFileService is the interface of a TLocateFileService (see page 24928) object. If the WebSnap application does not include a component that implements the ILocateFileService interface, the value of this property is nil (Delphi) or NULL (C++). When LocateFileService is nil (Delphi) or NULL (C++), the WebSnap application can still use template files and include files, but you have less control over how the application locates these files (that is, they must reside in the current directory or references to them must be qualified).

See Also

TLocateFileService (see page 24928)

179.1.11.3.6 TCustomWebAppComponents.PageDispatcher Property

Provides access to the current page dispatcher object.

Pascal

```
property PageDispatcher: IPageDispatcher;
```

C++

```
__property IPageDispatcher PageDispatcher;
```

Description

Use PageDispatcher to get or set the interface of the page dispatcher. The page dispatcher is responsible for interpreting requests for specific Web pages, checking the access rights for requested Web pages, and dispatching Web page requests to the appropriate Web page module.

Typically, PageDispatcher is the interface of a TPageDispatcher (see page 24949) object. If the WebSnap application does not include a TPageDispatcher (see page 24949) component, or some other component that implements the IPageDispatcher interface, the value of this property is nil (Delphi) or NULL (C++). When PageDispatcher is nil (Delphi) or NULL (C++), the WebSnap application does not automatically generate Web page content in response to requests that give the Web page name in their target URL.

See Also

TPageDispatcher (see page 24949)

AdapterDispatcher (see page 24919)

179.1.11.3.7 TCustomWebAppComponents.Sessions Property

Provides access to the sessions service.

Pascal

```
property Sessions: ISessionsService;
```

C++

```
__property ISessionsService Sessions;
```

Description

Use Sessions to get or set the interface of the sessions service. The sessions service maintains Web sessions so that the WebSnap application can store temporary information about an end user.

Typically, Sessions is the interface of a TSessionsService object. If the WebSnap application does not include a component that

implements the ISessionsService interface, the value of this property is nil (Delphi) or NULL (C++). When Sessions is nil (Delphi) or NULL (C++), the WebSnap application can't use Web sessions to store named information about the current end user.

See Also

TSessionsService (see page 25582)

179.1.11.3.8 TCustomWebAppComponents.UserListService Property

Provides access to the file location service.

Pascal

```
property UserListService: IWebUserList;
```

C++

```
__property IWebUserList UserListService;
```

Description

Use UserListService to get or set the interface of the user list service. The user list service maintains a list of user names, passwords, and access rights. Using this information, it can validate logins and check a user's access rights.

Typically, UserListService is the interface of a TWebUserList object. If the WebSnap application does not include a component that implements the IWebUserList interface, the value of this property is nil (Delphi) or NULL (C++). When UserListService is nil (Delphi) or NULL (C++), the WebSnap application can still allow users to log in and check access rights, but the support for this must be coded using event handlers on the page dispatcher.

See Also

TWebUserList (see page 25625)

TPageDispatcher (see page 24949)

179.1.11.4 TCustomWebAppComponents Events

The events of the TCustomWebAppComponents class are listed here.

Public Events

	Name	Description
	OnException (see page 24922)	Occurs when an exception escapes all try blocks in the WebSnap application.

179.1.11.4.1 TCustomWebAppComponents.OnException Event

Occurs when an exception escapes all try blocks in the WebSnap application.

Pascal

```
property OnException: TWebExceptionEvent;
```

C++

```
__property TWebExceptionEvent OnException;
```

Description

Write an OnException event handler respond to exceptions that arise while processing a request message. OnException occurs when an exception is not caught by the components that process the request message. If the OnException event handler does not handle the exception, TCustomWebAppComponents (see page 24906) updates the response message so that it describes the exception to the caller.

See Also

OnAfterDispatch (see page 24948)

179.1.12 TInvalidWebSession Class

Serves as a placeholder when an HTTP request could not be given a session.

Class Hierarchy

System::TObject → HTTPApp.TAbstractWebSession → WebDisp.TInvalidWebSession

Pascal

```
TInvalidWebSession = class(TAbstractWebSession);
```

C++

```
class TInvalidWebSession : public TAbstractWebSession;
```

File

WebDisp

Description

TInvalidWebSession implements the Session property on the Web context when the WebSnap application fails to create an active session (TWebSession (see page 24968)) for the current request. The failure to create an active session can arise from various reasons, including the situation where the WebSnap application does not have a TSessionsService component or the situation where an incoming request message tries to reactivate a session that was terminated.

TInvalidWebSession acts as a placeholder, so that the WebSnap application does not need to raise an exception when the attempt to create an active session fails. Any attempt to use TInvalidWebSession, however, leads to an ESessionException (see page 24840) exception.

See Also

TSessionsService (see page 25582)

TAbstractWebContext (see page 24644)

TWebSession (see page 24968)

ESessionException (see page 24840)

179.1.12.1 TInvalidWebSession Members

The following tables list the members exposed by TInvalidWebSession.

Protected Methods

	Name	Description
	GetSessionID (see page 24925)	Returns the value of the SessionID property.
	GetTimeoutMinutes (see page 24925)	Returns the value of the TimeoutMinutes property.
	GetValue (see page 24926)	Returns the value associated with a specified name.
	SetTimeoutMinutes (see page 24926)	Sets the value of the TimoutMinutes property.
	SetValue (see page 24927)	Assigns a value to a specified name.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 24924)	Creates and initializes an instance of TInvalidWebSession (see page 24923).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Terminate (see page 24927)	Raises an ESessionException (see page 24840) exception.
	UpdateResponse (see page 24927)	Does nothing.

Public Properties

	Name	Description
	SessionID (see page 15087)	Uniquely identifies the Session.
	TimeoutMinutes (see page 15087)	Indicates the time, in minutes, that a session can be idle before it terminates itself.
	Values (see page 15088)	Lists the named values that the session stores.

179.1.12.2 TInvalidWebSession.Create Constructor

Creates and initializes an instance of TInvalidWebSession (see page 24923).

Pascal

```
constructor Create(const AMsg: string);
```

C++

```
__fastcall TInvalidWebSession(const AnsiString AMsg);
```

Description

Do not create instances of TInvalidWebSession (see page 24923). WebSnap applications automatically create

TInvalidWebSession (see page 24923) objects when a call to the TWebSession (see page 24968) constructor fails.

AMsg is the error message from the exception that was raised within a call to the TWebSession (see page 24968) constructor. This error message is used for any ESessionException (see page 24840) exception that the TInvalidWebSession (see page 24923) object raises.

See Also

ESessionException (see page 24840)

Create (see page 24970)

179.1.12.3 TInvalidWebSession Methods

The methods of the TInvalidWebSession class are listed here.

Protected Methods

	Name	Description
	GetSessionID (see page 24925)	Returns the value of the SessionID property.
	GetTimeoutMinutes (see page 24925)	Returns the value of the TimeoutMinutes property.
	GetValue (see page 24926)	Returns the value associated with a specified name.
	SetTimeoutMinutes (see page 24926)	Sets the value of the TimoutMinutes property.
	SetValue (see page 24927)	Assigns a value to a specified name.

Public Methods

	Name	Description
	Terminate (see page 24927)	Raises an ESessionException (see page 24840) exception.
	UpdateResponse (see page 24927)	Does nothing.

179.1.12.3.1 TInvalidWebSession.GetSessionID Method

Returns the value of the SessionID property.

Pascal

```
function GetSessionID: string; override;
```

C++

```
virtual __fastcall AnsiString GetSessionID();
```

Description

GetSessionID is the protected read implementation of the SessionID property.

In TAbstractWebSession, GetSessionID is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the SessionID property.

See Also

SessionID (see page 15087)

179.1.12.3.2 TInvalidWebSession.GetTimeoutMinutes Method

Returns the value of the TimeoutMinutes property.

Pascal

```
function GetTimeoutMinutes: Integer; override;
```

C++

```
virtual __fastcall int GetTimeoutMinutes();
```

Description

GetTimeoutMinutes is the protected read implementation of the TimeoutMinutes property.

In TAbstractWebSession, GetTimeoutMinutes is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the TimeoutMinutes property.

See Also

TimeoutMinutes (see page 15087)

SetTimeoutMinutes (see page 15085)

179.1.12.3.3 TInvalidWebSession.GetValue Method

Returns the value associated with a specified name.

Pascal

```
function GetValue(const AName: string): Variant; override;
```

C++

```
virtual __fastcall Variant GetValue(const AnsiString AName);
```

Description

GetValue is the protected read implementation of the Values property.

AName is the name of the value to return.

In TAbstractWebSession, GetValue is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Values property.

See Also

Values (see page 15088)

SetValue (see page 15085)

179.1.12.3.4 TInvalidWebSession.SetTimeoutMinutes Method

Sets the value of the TimoutMinutes property.

Pascal

```
procedure SetTimeoutMinutes(AValue: Integer); override;
```

C++

```
virtual __fastcall SetTimeoutMinutes(int AValue);
```

Description

SetTimeoutMinutes is the protected write implementation of the TimeoutMinutes property.

AValue is the number of minutes that can elapse before the session terminates itself.

In TAbstractWebSession, SetTimeoutMinutes is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the TimeoutMinutes property.

See Also

GetTimeoutMinutes (see page 15084)

TimeoutMinutes (see page 15087)

179.1.12.3.5 TInvalidWebSession.SetValue Method

Assigns a value to a specified name.

Pascal

```
procedure SetValue(const AName: string; const AValue: Variant); override;
```

C++

```
virtual __fastcall SetValue(const AnsiString AName, const Variant AValue);
```

Description

SetValue is the protected write implementation of the Value property.

AName is the name of the value to store.

AValue is the value to store under the specified name.

In TAbstractWebSession, SetValue is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Values property.

See Also

GetValue (see page 15084)

Values (see page 15088)

179.1.12.3.6 TInvalidWebSession.Terminate Method

Raises an ESessionException (see page 24840) exception.

Pascal

```
procedure Terminate; override;
```

C++

```
virtual __fastcall Terminate();
```

Description

Do not call the Terminate method. TInvalidWebSession (see page 24923) does not represent an active Web session, so there is no Web session to terminate.

If the application calls the Terminate method, TInvalidWebSession (see page 24923) raises an ESessionException (see page 24840) object.

See Also

ESessionException (see page 24840)

179.1.12.3.7 TInvalidWebSession.UpdateResponse Method

Does nothing.

Pascal

```
procedure UpdateResponse(AResponse: TWebResponse); override;
```

C++

```
virtual __fastcall UpdateResponse(TWebResponse AResponse);
```

Description

UpdateResponse is called automatically to allow the WebSnap application to add the session ID to the response message before it is sent to the user. In TInvalidWebSession (see page 24923), UpdateResponse does nothing, because invalid Web sessions do not have session ID's.

See Also

SessionID

179.1.13 TLocateFileService Class

TLocateFileService controls the access of templates and server-side script include files at runtime.

Class Hierarchy

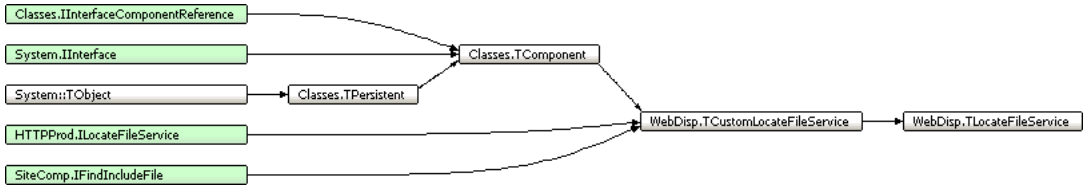

Pascal

```
TLocateFileService = class(TCustomLocateFileService);
```

C++

```
class TLocateFileService : public TCustomLocateFileService;
```

File

WebDisp

Description

Add a TLocateFileService component to your WebSnap application when you want greater control over how template and server-side script include files are accessed at runtime. Define event handlers that will be called when the application requires one of these files. Event handlers can specify a different directory location for a file or return a TStream containing the file.

See Also

TStream (see page 2973)

179.1.13.1 TLocateFileService Members

The following tables list the members exposed by TLocateFileService.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FindIncludeFile (see page 24884)	Returns the fully-qualified path name for an include file for a server-side script.
V	FindStream (see page 24885)	Generates an OnFindStream (see page 24889) event.
	FindTemplateFile (see page 24885)	Generates an OnFindTemplateFile (see page 24889) event.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetFileStream (see page 24886)	Returns a stream object from which to read a specified file.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTemplateStream (see page 24886)	Returns a stream from which to read a template file.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	ImplFindIncludeFile (see page 24887)	Provides the underlying implementation for the FindIncludeFile (see page 24884) method.
V	ImplGetTemplateStream (see page 24887)	Provides the underlying implementation for the GetTemplateStream (see page 24886) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.

	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindIncludeFile (see page 19685)	This is FindIncludeFile, a member of class IFindIncludeFile.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetTemplateStream (see page 15321)	This is GetTemplateStream, a member of class ILocateFileService.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnFindIncludeFile (see page 24932)	Occurs when the WebSnap application tries to locate an include file for a server-side script.
	OnFindStream (see page 24932)	Occurs when the WebSnap application tries to locate a file.
	OnFindTemplateFile (see page 24932)	Occurs when the WebSnap application tries to locate the template file for a content producer.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

179.1.13.2 TLocateFileService Events

The events of the TLocateFileService class are listed here.

Published Events

	Name	Description
	OnFindIncludeFile (see page 24932)	Occurs when the WebSnap application tries to locate an include file for a server-side script.
	OnFindStream (see page 24932)	Occurs when the WebSnap application tries to locate a file.
	OnFindTemplateFile (see page 24932)	Occurs when the WebSnap application tries to locate the template file for a content producer.

179.1.13.2.1 TLocateFileService.OnFindIncludeFile Event

Occurs when the WebSnap application tries to locate an include file for a server-side script.

Pascal

```
property OnFindIncludeFile: TLocateFileServiceFindFileEvent;
```

C++

```
__property TLocateFileServiceFindFileEvent OnFindIncludeFile;
```

Description

Write an OnFindIncludeFile event handler to indicate the location on disk of include files for server-side scripts. In WebSnap applications that include a locate file service component, you must supply either an OnFindIncludeFile or an OnFindStream event handler (or both) if the server-side scripts have include files. Otherwise the application can't locate these include files at all.

See Also

OnFindStream (see page 24889)

OnFindTemplateFile (see page 24889)

179.1.13.2.2 TLocateFileService.OnFindStream Event

Occurs when the WebSnap application tries to locate a file.

Pascal

```
property OnFindStream: TLocateFileServiceFindStreamEvent;
```

C++

```
__property TLocateFileServiceFindStreamEvent OnFindStream;
```

Description

Write an OnFindStream event handler to supply the WebSnap application with the contents of a file given its name. This event is particularly useful for applications that use "logical" files that do not exist physically on disk. When the WebSnap application tries to locate a file, the locate file service generates this event, along with an OnFindTemplateFile. One of these event handlers must be provided in order for the application to use the requested file.

See Also

OnFindIncludeFile (see page 24888)

OnFindTemplateFile (see page 24889)

179.1.13.2.3 TLocateFileService.OnFindTemplateFile Event

Occurs when the WebSnap application tries to locate the template file for a content producer.

Pascal

```
property OnFindTemplateFile: TLocateFileServiceFindFileEvent;
```

C++

```
__property TLocateFileServiceFindFileEvent OnFindTemplateFile;
```

Description

Write an OnFindIncludeFile event handler to indicate the location on disk of templates that a content producer component uses to generate its content. In WebSnap applications that include a locate file service component, you must supply either an

OnFindTemplateFile or an OnFindStream event handler (or both) if the application has content producers that use template files. Otherwise content producers can't locate their templates.

See Also

OnFindStream (see page 24889)

OnFindIncludeFile (see page 24888)

179.1.14 TMultiModuleWebAppServices Class

TMultiModuleWebAppServices coordinates the response to HTTP request messages by locating the appropriate components to handle various tasks.

Class Hierarchy

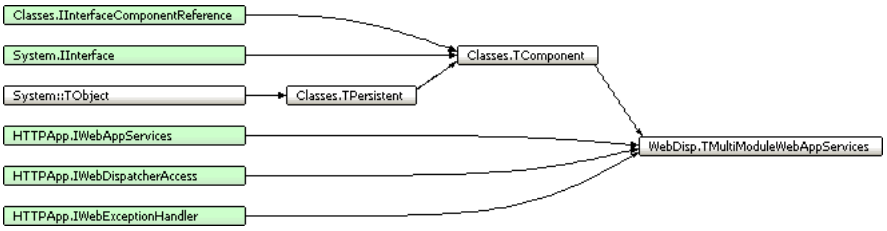

Pascal

```
TMultiModuleWebAppServices = class(TComponent, IWebAppServices, IWebDispatcherAccess,
IWebExceptionHandler);
```

C++

```
class TMultiModuleWebAppServices : public TComponent, public IWebAppServices, public
IWebDispatcherAccess, public IWebExceptionHandler;
```

File

WebDisp

Description

TMultiModuleWebAppServices is the base class for components that provide access to the services available in a WebSnap application. It keeps track of the current request and response objects, maintains a list of the available Web modules, and introduces methods for locating the appropriate components to handle incoming request messages.

Do not use TMultiModuleWebAppServices directly. The methods that process request messages and any exceptions that arise during that processing are simple stubs that should be overridden by a descendant class. Rather, use a TMultiModuleWebAppServices descendant such as TWebAppComponents (see page 24958), to provide access to the central components of the WebSnap application.

See Also

TWebRequest (see page 15161)

TWebResponse (see page 15191)

TWebRequestHandler (see page 25448)

TWebAppComponents (see page 24958)

IWebAppServices (see page 15227)

179.1.14.1 TMultiModuleWebAppServices Members

The following tables list the members exposed by TMultiModuleWebAppServices.

Protected Events

	Name	Description
	OnAfterDispatch (see page 24948)	Occurs after the WebSnap application generates the content of a response message.
	OnBeforeDispatch (see page 24948)	Occurs before TMultiModuleWebAppServices (see page 24933) passes an HTTP request onto the other components in the application for processing.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	Access_Request (see page 24937)	Returns the value of the Access_Request property.
	Access_Response (see page 24938)	Returns the value of the Access_Response property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CreateContext (see page 24938)	Creates the Web context for the current request message.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoAfterDispatch (see page 24939)	Generates an OnAfterDispatch (see page 24948) event.
	DoBeforeDispatch (see page 24939)	Generates an OnBeforeDispatch (see page 24948) event.
V	DoneModule (see page 24939)	Performs any final tasks before deactivating a Web module.
	FinishContext (see page 24940)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetExceptionHandler (see page 24940)	Returns the object that handles exceptions that arise in the WebSnap application.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleException (see page 24941)	Responds to any exceptions not caught by the WebSnap application.
	HandleRequest (see page 24941)	Directs the processing of the current request message.
V	ImplFinishContext (see page 24942)	Provides the underlying implementation of the FinishContext (see page 24940) method.
V	ImplHandleException (see page 24942)	Provides the underlying implementation of the HandleException (see page 24941) method.
V	ImplHandleRequest (see page 24943)	Provides the underlying implementation for the HandleRequest (see page 24941) method.
V	ImplInitContext (see page 24943)	Provides the underlying implementation for the InitContext (see page 24944) method.
	InitContext (see page 24944)	Creates and initializes the Web context for the current Web request.

V	InitModule (see page 24945)	Initializes a Web module before the WebSnap application uses it to handle the current request.
V	InitModules (see page 24945)	Initializes all the currently active Web modules.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	ModuleAdded (see page 24946)	Initializes a Web module when it is activated while processing the current request.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Request (see page 24946)	Provides access to the current request object.
R	Response (see page 24947)	Provides access to the current response object.
R	WebModules (see page 24947)	Lists the Web modules available in the WebSnap application.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.

R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	ExceptionHandler (see page 15230)	Indicates the object that handles exceptions that arise when processing a request message.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

179.1.14.2 TMultiModuleWebAppServices Methods

The methods of the TMultiModuleWebAppServices class are listed here.

Protected Methods

	Name	Description
	Access_Request (see page 24937)	Returns the value of the Access_Request property.
	Access_Response (see page 24938)	Returns the value of the Access_Response property.
V	CreateContext (see page 24938)	Creates the Web context for the current request message.
	DoAfterDispatch (see page 24939)	Generates an OnAfterDispatch (see page 24948) event.
	DoBeforeDispatch (see page 24939)	Generates an OnBeforeDispatch (see page 24948) event.
V	DoneModule (see page 24939)	Performs any final tasks before deactivating a Web module.
	FinishContext (see page 24940)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetExceptionHandler (see page 24940)	Returns the object that handles exceptions that arise in the WebSnap application.
	HandleException (see page 24941)	Responds to any exceptions not caught by the WebSnap application.
	HandleRequest (see page 24941)	Directs the processing of the current request message.
V	ImplFinishContext (see page 24942)	Provides the underlying implementation of the FinishContext (see page 24940) method.
V	ImplHandleException (see page 24942)	Provides the underlying implementation of the HandleException (see page 24941) method.
V	ImplHandleRequest (see page 24943)	Provides the underlying implementation for the HandleRequest (see page 24941) method.
V	ImplInitContext (see page 24943)	Provides the underlying implementation for the InitContext (see page 24944) method.
	InitContext (see page 24944)	Creates and initializes the Web context for the current Web request.
V	InitModule (see page 24945)	Initializes a Web module before the WebSnap application uses it to handle the current request.
V	InitModules (see page 24945)	Initializes all the currently active Web modules.
	ModuleAdded (see page 24946)	Initializes a Web module when it is activated while processing the current request.

179.1.14.2.1 TMultiModuleWebAppServices.Access_Request Method

Returns the value of the Access_Request property.

Pascal

```
function Access_Request: TWebRequest;
```

C++

```
__fastcall TWebRequest Access_Request();
```

Description

Access_Access_Request provides the TMultiModuleWebAppServices (see page 24933) object's implementation of the

Access_Request method of its IWebDispatcherAccess interface. This allows content producer components in the application to access the current request message.

See Also

Access_Request

Request (see page 15235)

Dispatcher (see page 15109)

179.1.14.2.2 TMultiModuleWebAppServices.Access_Response Method

Returns the value of the Access_Response property.

Pascal

```
function Access_Response: TWebResponse;
```

C++

```
__fastcall TWebResponse Access_Response();
```

Description

Access_Access_Response provides the TMultiModuleWebAppServices (see page 24933) object's implementation of the Access_Response method of its IWebDispatcherAccess interface. This allows content producer components in the application to access the current Access_Response message.

See Also

Access_Response

Response (see page 15235)

Dispatcher (see page 15109)

179.1.14.2.3 TMultiModuleWebAppServices.CreateContext Method

Creates the Web context for the current request message.

Pascal

```
function CreateContext: TAbstractWebContext; virtual;
```

C++

```
virtual __fastcall TAbstractWebContext CreateContext();
```

Description

The InitContext (see page 24944) method calls CreateContext to instantiate the Web context object for the current request. This context can then be accessed by the WebSnap application using the global WebContext function.

In TMultiModuleWebAppServices (see page 24933), CreateContext creates a TWebContext object, passing the values of the Request (see page 24946), Response (see page 24947), and WebModules (see page 24947) properties to the constructor. The Web context is created without a session.

See Also

WebContext (see page 24687)

InitContext (see page 24944)

Create (see page 24662)

Request (see page 24946)

Response (see page 24947)

WebModules (see page 24947)

179.1.14.2.4 TMultiModuleWebAppServices.DoAfterDispatch Method

Generates an OnAfterDispatch (see page 24948) event.

Pascal

```
function DoAfterDispatch(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean DoAfterDispatch(TWebRequest Request, TWebResponse Response);
```

Description

TMultiModuleWebAppServices (see page 24933) descendants call DoAfterDispatch to generate an OnAfterDispatch (see page 24948) event after dispatching a request message to the other components in the WebSnap application.

Request (see page 24946) is the current request message that was just handled.

Response (see page 24947) is the response message that was filled in by the rest of the WebSnap application.

DoAfterDispatch returns true if the WebSnap application should send the response message back to the caller.

See Also

OnAfterDispatch (see page 24948)

179.1.14.2.5 TMultiModuleWebAppServices.DoBeforeDispatch Method

Generates an OnBeforeDispatch (see page 24948) event.

Pascal

```
function DoBeforeDispatch(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean DoBeforeDispatch(TWebRequest Request, TWebResponse Response);
```

Description

TMultiModuleWebAppServices (see page 24933) descendants call DoBeforeDispatch to generate an OnBeforeDispatch (see page 24948) event before dispatching a request message to the other components in the WebSnap application.

Request (see page 24946) is the current request message that needs to be handled.

Response (see page 24947) is the response message that needs to be filled out.

DoBeforeDispatch returns true if the OnBeforeDispatch (see page 24948) event handler has handled the event. When DoBeforeDispatch returns true, the WebSnap application does not need to continue processing the request.

See Also

OnBeforeDispatch (see page 24948)

179.1.14.2.6 TMultiModuleWebAppServices.DoneModule Method

Performs any final tasks before deactivating a Web module.

Pascal

```
procedure DoneModule(AWebModule: TComponent); virtual;
```

C++

```
virtual __fastcall DoneModule(TComponent * AWebModule);
```

Description

The FinishContext (see page 24940) method calls DoneModule for every active Web module as part of the final cleanup before freeing the Web modules and Web context when the WebSnap application is finished handling an HTTP request. DoneModule notifies the Web module that it is about to be deactivated, so that it can perform any final tasks before it is freed or deactivated.

AWebModule is the Web module instance that is about to be deactivated.

See Also

WebModules (see page 24947)

InitModule (see page 24945)

FinishContext (see page 24940)

179.1.14.2.7 TMultiModuleWebAppServices.FinishContext Method

Performs any final tasks after the WebSnap application has finished processing a request.

Pascal

```
procedure FinishContext;
```

C++

```
__fastcall FinishContext();
```

Description

FinishContext is called as the last step before a WebSnap application sends a response message back to the caller. FinishContext performs any final tasks, and frees all the objects and components that were created to handle the request message.

FinishContext is not a virtual method. Descendant classes can't override this method to add additional final tasks. Instead, FinishContext calls the virtual ImplFinishContext (see page 24942) method, which descendant classes override to add their own final tasks.

See Also

InitContext (see page 24944)

ImplFinishContext (see page 24942)

179.1.14.2.8 TMultiModuleWebAppServices.GetExceptionHandler Method

Returns the object that handles exceptions that arise in the WebSnap application.

Pascal

```
function GetExceptionHandler: TObject;
```

C++

```
__fastcall TObject * GetExceptionHandler();
```

Description

GetExceptionHandler returns the object that the WebSnap application should call when an exception is raised while trying to process a request message. The request handler must implement the IWebExceptionHandler interface, which includes only a single method: HandleException (see page 24941).

In TMultiModuleWebAppServices (see page 24933), GetExceptionHandler returns self this, because TMultiModuleWebAppServices (see page 24933) implements the IWebExceptionHandler interface.

See Also

HandleException (see page 24941)

179.1.14.2.9 TMultiModuleWebAppServices.HandleException Method

Responds to any exceptions not caught by the WebSnap application.

Pascal

```
procedure HandleException(E: Exception; var Handled: Boolean);
```

C++

```
__fastcall HandleException(Exception E, Boolean Handled);
```

Description

HandleException provides the last opportunity for a WebSnap application to handle an exception that passes through all the try blocks in the application code. If the HandleException method does not handle the exception, the WebSnap application displays a dialog box that indicates an error occurred.

HandleException calls the ImplHandleException (see page 24942) method to handle the exception. In TMultiModuleWebAppServices (see page 24933), ImplHandleException (see page 24942) does nothing. However, descendant classes override ImplHandleException (see page 24942) to provide more meaningful responses, such as generating an OnException event or creating a response message that describes the exception to the caller.

E is the exception object for the exception that occurred.

Handled returns true if HandleException handled the exception. If Handled returns false, the WebSnap application displays a dialog box that indicates an error occurred.

See Also

ImplHandleException (see page 24942)

179.1.14.2.10 TMultiModuleWebAppServices.HandleRequest Method

Directs the processing of the current request message.

Pascal

```
function HandleRequest: Boolean;
```

C++

```
__fastcall Boolean HandleRequest();
```

Description

When the WebSnap application receives a request message, it makes the following calls:

1. The InitContext (see page 24944) method is called to generate a Web context for the request.

2. HandleRequest is called to generate a response to the request.
3. FinishContext (see page 24940) is called to perform any final cleanup.

HandleRequest simply calls the ImplHandleRequest (see page 24943) method, which directs the processing of the request. In TMultiModuleWebAppServices (see page 24933), ImplHandleRequest (see page 24943) simply declares the request as handled while doing no processing. Descendant classes override ImplHandleRequest (see page 24943) to locate the components that should handle the request and forwarding the request to those components for processing.

HandleRequest returns true if the request is handled and the response should be sent back to the caller (if it was not already sent). It returns false if the WebSnap application could not handle the request.

See Also

ImplHandleRequest (see page 24943)

Request (see page 24946)

InitContext (see page 24944)

FinishContext (see page 24940)

179.1.14.2.11 TMultiModuleWebAppServices.ImplFinishContext Method

Provides the underlying implementation of the FinishContext (see page 24940) method.

Pascal

```
procedure ImplFinishContext; virtual;
```

C++

```
virtual __fastcall ImplFinishContext();
```

Description

ImplFinishContext performs any final tasks and cleanup after the WebSnap application has finished processing a request message.

In TMultiModuleWebAppServices (see page 24933), ImplFinishContext calls the DoneModule (see page 24939) method for every active Web module so that it can be freed or deactivated, and then frees the Web context for the current request.

Descendant classes override ImplFinishContext to add their own final tasks.

See Also

FinishContext (see page 24940)

DoneModule (see page 24939)

WebContext (see page 24687)

179.1.14.2.12 TMultiModuleWebAppServices.ImplHandleException Method

Provides the underlying implementation of the HandleException (see page 24941) method.

Pascal

```
procedure ImplHandleException(E: Exception; var Handled: Boolean); virtual;
```

C++

```
virtual __fastcall ImplHandleException(Exception E, Boolean Handled);
```

Description

The HandleException (see page 24941) method calls ImplHandleException to handle any exception that passes through all the try blocks in the application code. If ImplHandleException does not handle the exception, the WebSnap application displays a dialog box that indicates an error occurred.

In TMultiModuleWebAppServices (see page 24933), ImplHandleException returns true, ignoring the exception that occurred. Descendant classes override this method to provide more meaningful responses, such as generating an OnException event or creating a response message that describes the exception to the caller.

E is the exception object for the exception that occurred.

Handled returns true if ImplHandleException handled the exception. If Handled returns false, the WebSnap application displays a dialog box that indicates an error occurred.

See Also

HandleException (see page 24941)

179.1.14.2.13 TMultiModuleWebAppServices.ImplHandleRequest Method

Provides the underlying implementation for the HandleRequest (see page 24941) method.

Pascal

```
function ImplHandleRequest: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest();
```

Description

The HandleRequest (see page 24941) method calls ImplHandleRequest to handle the current HTTP request message. In TMultiModuleWebAppServices (see page 24933), ImplHandleRequest returns true, indicating that the request is handled, but does nothing to fill out the response object. Descendant classes override this method to locate components in the WebSnap application that can process the request and forward the request to them for handling.

ImplHandleRequest returns true if the request is handled and the response should be sent back to the caller (if it was not already sent). It returns false if the WebSnap application could not handle the request.

See Also

HandleRequest (see page 24941)

Request (see page 24946)

Response (see page 24947)

179.1.14.2.14 TMultiModuleWebAppServices.ImplInitContext Method

Provides the underlying implementation for the InitContext (see page 24944) method.

Pascal

```
procedure ImplInitContext(AWebModuleList: TAbstractWebModuleList; ARequest: TWebRequest;
AResponse: TWebResponse); virtual;
```

C++

```
virtual __fastcall ImplInitContext(TAbstractWebModuleList AWebModuleList, TWebRequest
ARequest, TWebResponse AResponse);
```

Description

The InitContext (see page 24944) method, which is called when the WebSnap application first receives a request, calls ImplInitContext to perform any initializations and to create the Web context for the new request message.

ImplInitContext performs the following tasks:

1. Sets the WebModules (see page 24947), Request (see page 24946), and Response (see page 24947) properties to the values of the AWebModuleList, ARequest, and AResponse parameters, respectively.
2. Calls the CreateContext (see page 24938) method to create a Web context for the current request. This Web context can then be accessed using the global WebContext function.
3. Calls the InitModules (see page 24945) method to initialize all the Web modules that the AWebModuleList parameter indicates are currently instantiated. (These are the Web modules that are created for every request message.)
4. Hooks up the ModuleAdded (see page 24946) method so that it can initialize any Web modules that are instantiated in the course of processing the current request.

See Also

InitContext (see page 24944)

CreateContext (see page 24938)

WebContext (see page 24687)

Request (see page 24946)

Response (see page 24947)

WebModules (see page 24947)

InitModules (see page 24945)

179.1.14.2.15 TMultiModuleWebAppServices.InitContext Method

Creates and initializes the Web context for the current Web request.

Pascal

```
procedure InitContext(WebModuleList: TAbstractWebModuleList; Request: TWebRequest; Response:
TWebResponse);
```

C++

```
__fastcall InitContext(TAbstractWebModuleList WebModuleList, TWebRequest Request, TWebResponse
Response);
```

Description

InitContext is called when the WebSnap application first receives a request message to set up the Web context before the application attempts to handle the request. It creates and sets up the global Web context, which can be accessed by calling the global WebContext function. It also performs any other initializations that ready the application for handling a new request message.

WebModuleList lists the currently active Web modules in the WebSnap application and provides access to all Web module factories so that the application can create any new modules it needs for processing the request message. This becomes the value of the WebModules (see page 24947) property.

Request (see page 24946) describes the new request message. This becomes the value of the Request (see page 24946) property.

Response (see page 24947) is the response object that the WebSnap application should fill out when handling the current

request. This becomes the value of the Response (see page 24947) property.

InitContext is not a virtual method. Descendant classes can't override this method to add additional initializations. Instead, InitContext calls the virtual ImplInitContext (see page 24943) method, which descendant classes override to add their own initial tasks.

See Also

ImplInitContext (see page 24943)

WebContext (see page 24687)

Request (see page 24946)

Response (see page 24947)

WebModules (see page 24947)

179.1.14.2.16 TMultiModuleWebAppServices.InitModule Method

Initializes a Web module before the WebSnap application uses it to handle the current request.

Pascal

```
procedure InitModule(AWebModule: TComponent); virtual;
```

C++

```
virtual __fastcall InitModule(TComponent * AWebModule);
```

Description

Both the ModuleAdded (see page 24946) and the InitModules (see page 24945) methods call InitModule to initialize a Web module after it is first created. InitModule assigns itself as the Web dispatcher to any content producers in the Web module, and notifies the Web module that it has been activated.

AWebModule is the Web module component to initialize.

See Also

InitModules (see page 24945)

ModuleAdded (see page 24946)

WebModules (see page 24947)

Dispatcher (see page 15109)

179.1.14.2.17 TMultiModuleWebAppServices.InitModules Method

Initializes all the currently active Web modules.

Pascal

```
procedure InitModules; virtual;
```

C++

```
virtual __fastcall InitModules();
```

Description

InitModules is called to initialize the currently active Web modules when the WebSnap application first sets up the Web context for a new request. It iterates over all the active Web modules that the WebModules (see page 24947) property lists, calling the InitModule (see page 24945) method to initialize each one.

See Also

InitModule (see page 24945)

WebModules (see page 24947)

InitContext (see page 24944)

179.1.14.2.18 TMultiModuleWebAppServices.ModuleAdded Method

Initializes a Web module when it is activated while processing the current request.

Pascal

```
procedure ModuleAdded(AWebModule: TComponent);
```

C++

```
__fastcall ModuleAdded(TComponent * AWebModule);
```

Description

TMultiModuleWebAppServices (see page 24933) assigns ModuleAdded as the OnModuleAdded event handler on the TWebModuleList that is the value of the WebModules (see page 24947) property. As a result, this method is called every time the WebSnap application activates a new Web module while processing the current request.

ModuleAdded calls the InitModule (see page 24945) method to initialize the new Web module before it is used.

AWebModule is the newly activated Web module.

See Also

InitModule (see page 24945)

OnModuleAdded (see page 15059)

WebModules (see page 24947)

179.1.14.3 TMultiModuleWebAppServices Properties

The properties of the TMultiModuleWebAppServices class are listed here.

Protected Properties

	Name	Description
R	Request (see page 24946)	Provides access to the current request object.
R	Response (see page 24947)	Provides access to the current response object.
R	WebModules (see page 24947)	Lists the Web modules available in the WebSnap application.

179.1.14.3.1 TMultiModuleWebAppServices.Request Property

Provides access to the current request object.

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

Read Request to obtain information about the current HTTP request message. The request object provides access to the

headers of the request message, as well as the content of the message.

See Also

Response (see page 24947)

179.1.14.3.2 TMultiModuleWebAppServices.Response Property

Provides access to the current response object.

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

Read Response to obtain information about or add to the response message that the application builds as a reply to the message that the Request (see page 24946) property specifies. The response object provides access to the headers of the response message, as well as the content of the message.

See Also

Request (see page 24946)

179.1.14.3.3 TMultiModuleWebAppServices.WebModules Property

Lists the Web modules available in the WebSnap application.

Pascal

```
property WebModules: TAbstractWebModuleList;
```

C++

```
__property TAbstractWebModuleList WebModules;
```

Description

Use WebModules to access one of the Web modules that make up the WebSnap application. WebModules lists all of the currently instantiated Web modules. It also provides access to the factory objects for all Web modules in the application, including those that are not yet instantiated.

When the application first receives a Web request message, the Web request handler initializes WebModules to include all Web modules that have the crAlways flag set in their factory's CreateMode property.

See Also

CreateMode

179.1.14.4 TMultiModuleWebAppServices Events

The events of the TMultiModuleWebAppServices class are listed here.

Protected Events

	Name	Description
	OnAfterDispatch (see page 24948)	Occurs after the WebSnap application generates the content of a response message.

	OnBeforeDispatch (see page 24948)	Occurs before TMultiModuleWebAppServices (see page 24933) passes an HTTP request onto the other components in the application for processing.

179.1.14.4.1 TMultiModuleWebAppServices.OnAfterDispatch Event

Occurs after the WebSnap application generates the content of a response message.

Pascal

```
property OnAfterDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent OnAfterDispatch;
```

Description

Write an AfterDispatch event handler to perform any final actions before sending a response back to the Web browser that sent the WebSnap application a request. OnAfterDispatch occurs after the other components in the WebSnap application have attempted to generate a response for the current request. It does not occur if the response message has already been sent back to the caller.

If the AfterDispatch event handler sets the Handled parameter to true, the WebSnap application sends the response back to the caller after the event handler exits. If it sets the Handled parameter to false, the WebSnap application does not send the response message.

Note: Although TMultiModuleWebAppServices (see page 24933) introduces the OnAfterDispatch event, it never calls the event handler. Descendant classes add the code to generate this event after dispatching a request message.

See Also

OnBeforeDispatch (see page 24948)

179.1.14.4.2 TMultiModuleWebAppServices.OnBeforeDispatch Event

Occurs before TMultiModuleWebAppServices (see page 24933) passes an HTTP request onto the other components in the application for processing.

Pascal

```
property OnBeforeDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent OnBeforeDispatch;
```

Description

Write an OnBeforeDispatch event handler to initialize the response to an HTTP request or to handle any messages that the other components in the WebSnap application can't handle. If the event handler completely handles the request, filling out the response object so that it is ready to send, it should set the Handled parameter to true, thereby preventing this component from trying to locate a component in the WebSnap application that can handle the message.

Note: Although TMultiModuleWebAppServices (see page 24933) introduces the OnBeforeDispatch event, it never calls the event handler. Descendant classes add the code to generate this event before dispatching a request message.

See Also

OnAfterDispatch (see page 24948)

179.1.15 TPageDispatcher Class

TPageDispatcher dispatches incoming HTTP request messages in a WebSnap application.

Class Hierarchy

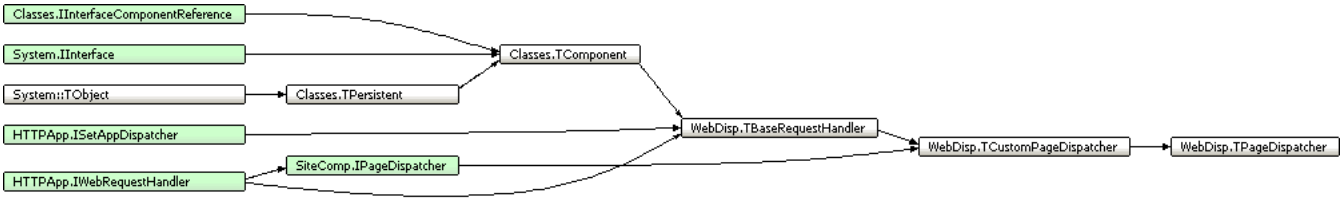

Pascal

```
TPageDispatcher = class(TCustomPageDispatcher);
```

C++

```
class TPageDispatcher : public TCustomPageDispatcher;
```

File

WebDisp

Description

Add a single TPageDispatcher object to the Web application module of a WebSnap application. Typically, the WebSnap application wizard adds a page dispatcher to the Web application module that it creates.

A WebSnap application only requires a single page dispatcher. The page dispatcher examines all incoming HTTP request messages, checking the pathInfo portion of the URL to which a request is addressed for the name of a registered Web page module. It then forwards the request to the matching Web page module so that the page module can handle the request and produce the content of a response. If the request is not addressed to a URL with a pathinfo portion, the page dispatcher forwards the request to the Web page module identified by its DefaultPage (see page 24953) property.

See Also

TWebDispatcher (see page 15147)

179.1.15.1 TPageDispatcher Members

The following tables list the members exposed by TPageDispatcher.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.

	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DispatchPageName (see page 24894)	Checks whether a specified user has a given access rights string.
	EndUser (see page 24895)	This is EndUser, a member of class TCustomPageDispatcher.
	GetCanView (see page 24895)	Indicates whether the current user is permitted to view a specified Web page.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDefaultPageName (see page 24895)	Returns the value of the DefaultPageName property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetLoginRequired (see page 24896)	Indicates whether a user must log in before accessing a specified Web page.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetRequestedPage (see page 24896)	This is GetRequestedPage, a member of class TCustomPageDispatcher.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleRequest (see page 24864)	Handles an HTTP request.
	ImplDispatchPageName (see page 24896)	This is ImplDispatchPageName, a member of class TCustomPageDispatcher.
	ImplGetCanView (see page 24896)	This is ImplGetCanView, a member of class TCustomPageDispatcher.
	ImplGetDefaultPageName (see page 24897)	This is ImplGetDefaultPageName, a member of class TCustomPageDispatcher.
	ImplGetLoginRequired (see page 24897)	This is ImplGetLoginRequired, a member of class TCustomPageDispatcher.
	ImplHandleRequest (see page 24864)	Handles application-specific processing of an HTTP request.
	ImplHandleRequest (see page 24897)	Handles application-specific processing of an HTTP request.
	ImplPageNameOfRequest (see page 24897)	This is ImplPageNameOfRequest, a member of class TCustomPageDispatcher.
	ImplRedirectToPageName (see page 24898)	This is ImplRedirectToPageName, a member of class TCustomPageDispatcher.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsLoggedIn (see page 24898)	This is IsLoggedIn, a member of class TCustomPageDispatcher.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PageNameOfRequest (see page 24898)	Returns the name of the Web page to which an HTTP request is addressed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceLoginPage (see page 24899)	This is ProduceLoginPage, a member of class TCustomPageDispatcher.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	RedirectToPageName (see page 24899)	Fills out a response message so that it redirects the caller to another page in the application.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 24864)	Sets the AppDispatcher (see page 24865) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidatePage (see page 24899)	This is ValidatePage, a member of class TCustomPageDispatcher.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	AppDispatcher (see page 24865)	Provides read-only access to the application dispatcher component.

Public Events

	Name	Description
	OnAfterRedirectToPage (see page 24901)	Occurs after the page dispatcher successfully dispatches an HTTP request message in response to an internal redirection request.
	OnBeforeRedirectToPage (see page 24902)	Occurs before the page dispatcher dispatches an HTTP request message in response to an internal redirection request.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DispatchPageName (see page 19728)	Checks whether a specified user has a given access rights string.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetCanView (see page 19729)	Indicates whether the current user is permitted to view a specified Web page.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.

	GetDefaultPageName (see page 19729)	Returns the value of the DefaultPageName (see page 19731) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetLoginRequired (see page 19729)	Indicates whether a user must log in before accessing a specified Web page.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandleRequest (see page 15238)	Processes an HTTP request message.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PageNameOfRequest (see page 19730)	Returns the name of the Web page to which an HTTP request is addressed.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RedirectToPageName (see page 19730)	Fills out a response message so that it redirects the caller to another page in the application.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	DefaultPageName (see page 19731)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnAfterDispatchPage (see page 24954)	Occurs after the page dispatcher successfully dispatches an HTTP request message.
	OnBeforeDispatchPage (see page 24954)	Occurs before the page dispatcher passes an HTTP request to a Web page module for processing.
	OnCanViewPage (see page 24955)	Occurs just before the dispatcher passes a request message to a Web page module.
	OnIsLoginRequired (see page 24956)	Occurs when a request arrives from a caller that has not yet logged in.
	OnPageAccessDenied (see page 24956)	Occurs when the page dispatcher fails to validate the caller's right to access a requested Web page.
	OnPageNotFound (see page 24957)	Occurs when the dispatcher can't locate a registered Web page module with the name specified in an HTTP request message.

Published Properties

	Name	Description
	DefaultPage (see page 24953)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

179.1.15.2 TPageDispatcher Properties

The properties of the TPageDispatcher class are listed here.

Published Properties

	Name	Description
	DefaultPage (see page 24953)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

179.1.15.2.1 TPageDispatcher.DefaultPage Property

Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

Pascal

```
property DefaultPage: string;
```

C++

```
__property AnsiString DefaultPage;
```

Description

DefaultPage is the name of the registered Web page module that should handle incoming request messages that do not have a pathinfo portion on their target URL. This is the name that is supplied to the Web page module wizard when you created the default Web page module. It forms the basis of that Web page module's class name. For example, if the default Web page module has the name MyDefaultPage, the Web page module class is of type TMyDefaultPage.

If DefaultPage is an empty string, the page dispatcher uses the Web application module as a default Web page module, as long as it is a Web Page module.

179.1.15.3 TPageDispatcher Events

The events of the TPageDispatcher class are listed here.

Published Events

	Name	Description
	OnAfterDispatchPage (see page 24954)	Occurs after the page dispatcher successfully dispatches an HTTP request message.
	OnBeforeDispatchPage (see page 24954)	Occurs before the page dispatcher passes an HTTP request to a Web page module for processing.
	OnCanViewPage (see page 24955)	Occurs just before the dispatcher passes a request message to a Web page module.
	OnIsLoginRequired (see page 24956)	Occurs when a request arrives from a caller that has not yet logged in.
	OnPageAccessDenied (see page 24956)	Occurs when the page dispatcher fails to validate the caller's right to access a requested Web page.
	OnPageNotFound (see page 24957)	Occurs when the dispatcher can't locate a registered Web page module with the name specified in an HTTP request message.

179.1.15.3.1 TPageDispatcher.OnAfterDispatchPage Event

Occurs after the page dispatcher successfully dispatches an HTTP request message.

Pascal

```
property OnAfterDispatchPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterDispatchPage;
```

Description

Write an OnAfterDispatchPage event handler to take specific action after an HTTP request has been successfully handled by a Web page module. For example, an OnAfterDispatchPage event handler can maintain a log of all Web traffic handled by the WebSnap application.

OnAfterDispatchPage occurs only if the target Web page module succeeds in handling the request. If the target Web page module could not be validated, an OnPageAccessDenied event occurs instead. If the page could not be located, an OnPageNotFound event occurs instead.

See Also

OnBeforeDispatchPage (see page 24902)

OnPageNotFound (see page 24905)

OnCanViewPage (see page 24903)

OnIsLoginRequired (see page 24904)

OnPageAccessDenied (see page 24904)

179.1.15.3.2 TPageDispatcher.OnBeforeDispatchPage Event

Occurs before the page dispatcher passes an HTTP request to a Web page module for processing.

Pascal

```
property OnBeforeDispatchPage: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnBeforeDispatchPage;
```

Description

Write an OnBeforeDispatchPage event handler to process an HTTP request message before the dispatcher forwards it to the target Web page module for processing.

This event can provide an alternate mechanism for handling request messages, and then set the Handled parameter to true to prevent the page dispatcher from locating the target Web page module, verifying that it can handle the request, and forwarding the request to that Web page module.

If the event handler does not handle the request (for example, if it is merely maintaining a log of incoming requests), it can leave the Handled parameter as false and the dispatcher will continue with the default processing.

Tip: To access the Web request and response objects from within an OnBeforeDispatchPage event handler, call the global WebContext function to access the Web context and read its Request and Response properties.

See Also

OnAfterDispatchPage (see page 24901)

OnPageNotFound (see page 24905)

OnCanViewPage (see page 24903)

OnIsLoginRequired (see page 24904)

OnPageAccessDenied (see page 24904)

WebContext (see page 24687)

Request (see page 24658)

Response (see page 24659)

179.1.15.3.3 TPageDispatcher.OnCanViewPage Event

Occurs just before the dispatcher passes a request message to a Web page module.

Pascal

```
property OnCanViewPage: TCanViewPageEvent;
```

C++

```
__property TCanViewPageEvent OnCanViewPage;
```

Description

Write an OnCanViewPage event handler to customize the way the WebSnap application determines whether a given request can be passed to a Web page module for processing. This event is part of the mechanism by which the dispatcher validates a Web page for viewing. The validation process, which occurs after the OnBeforeDispatchPage event, occurs as follows:

1. If the caller has not already logged in, the dispatcher generates an OnIsLoginRequired event to check whether the specified Web page requires the caller to log in.
2. If a login is required, the dispatcher does not validate the page (and so does not forward the request to the specified Web page module). Rather, it initiates the process of creating a response to allow the user to log in.

a. First, the dispatcher generates an OnPageAccessDenied event, where the application can handle the login process.

b. If the OnPageAccessDenied event handler does not handle the login process, the dispatcher forwards the request to the login page specified by the end user adapter's LoginPage property.

3. If a login is not required (either because the caller already logged in or because there is no need to log in), the dispatcher generates an OnCanViewPage event. This event can validate a caller that has logged in, or provide any other type of validation that governs access to the requested page.
4. If there is no OnCanViewPage event handler, or if the OnCanViewPage event handler does not handle the validation, the

dispatcher calls on a TEndUserAdapter component to check the user's access rights.

Tip: To access the Web request and response objects from within an OnCanViewPage event handler, call the global WebContext function to access the Web context and read its Request and Response properties. To access information about the access rights that the target page requires, call the Web context's FindPageInfo method to obtain a Web page info object for the target page and check its ViewAccess property.

See Also

OnIsLoginRequired (see page 24904)

OnBeforeDispatchPage (see page 24902)

OnPageAccessDenied (see page 24904)

WebContext (see page 24687)

Request (see page 24658)

Response (see page 24659)

ViewAccess (see page 15069)

FindPageInfo (see page 24650)

179.1.15.3.4 TPageDispatcher.OnIsLoginRequired Event

Occurs when a request arrives from a caller that has not yet logged in.

Pascal

```
property OnIsLoginRequired: TLoginRequiredEvent;
```

C++

```
__property TLoginRequiredEvent OnIsLoginRequired;
```

Description

Write an OnIsLoginRequired event handler to customize the way the WebSnap application determines whether a caller must log in before obtaining access to a requested page. This event occurs after the OnBeforeDispatchPage event if the caller has not already logged in to the WebSnap application.

If there is no OnPageAccessDenied event handler, or if the OnPageAccessDenied event handler does not handle the login process, the dispatcher obtains a response message from the login page specified by the end user adapter.

See Also

OnCanViewPage (see page 24903)

OnBeforeDispatchPage (see page 24902)

OnPageAccessDenied (see page 24904)

LoginPage (see page 24118)

179.1.15.3.5 TPageDispatcher.OnPageAccessDenied Event

Occurs when the page dispatcher fails to validate the caller's right to access a requested Web page.

Pascal

```
property OnPageAccessDenied: TPageAccessDeniedEvent;
```

C++

```
__property TPageAccessDeniedEvent OnPageAccessDenied;
```

Description

Write an OnPageAccessDenied event handler to customize the response when a request for a specific Web page fails to be validated.

Tip: To access the Web request and response objects from within an OnPageAccessDenied event handler, call the global WebContext function to access the Web context and read its Request and Response properties.

See Also

OnIsLoginRequired (see page 24904)

OnCanViewPage (see page 24903)

WebContext (see page 24687)

Request (see page 24658)

Response (see page 24659)

LoginPage (see page 24118)

179.1.15.3.6 TPageDispatcher.OnPageNotFound Event

Occurs when the dispatcher can't locate a registered Web page module with the name specified in an HTTP request message.

Pascal

```
property OnPageNotFound: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnPageNotFound;
```

Description

Write an OnPageNotFound event handler to generate a response when the dispatcher can't locate the Web page module to which an HTTP request message is addressed. For example, an OnPageNotFound event handler can provide the response object with content that indicates that the requested page does not exist or that represents simple content for a page that does not have an associated Web page module.

If the OnPageNotFound event handler sets its Handled parameter to true, the page dispatcher assumes it has updated the response message and returns it. If the OnPageNotFound leaves the Handled parameter set to false, the dispatcher raises an EPageDispatcherException exception after the event handler exits.

Tip: To access the Web request and response objects from within an OnPageNotFound event handler, call the global WebContext function to access the Web context and read its Request and Response properties.

See Also

OnBeforeDispatchPage (see page 24902)

OnAfterDispatchPage (see page 24901)

OnPageAccessDenied (see page 24904)

EPageDispatcherException (see page 24839)

WebContext (see page 24687)

Request (see page 24658)

Response (see page 24659)

179.1.16 TWebAppComponents Class

TWebAppComponents holds the application components that provide the WebSnap application's functionality.

Class Hierarchy

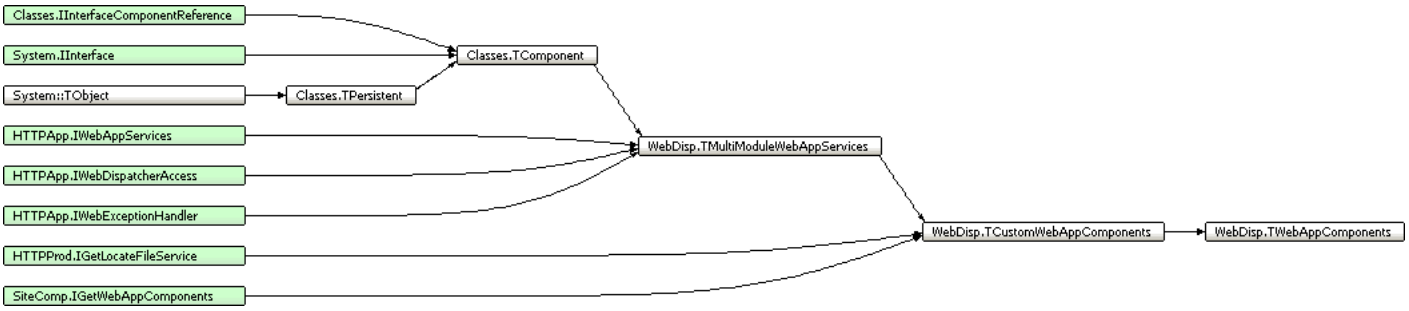

Pascal

```
TWebAppComponents = class(TCustomWebAppComponents);
```

C++

```
class TWebAppComponents : public TCustomWebAppComponents;
```

File

WebDisp

Description

Use TWebAppComponents to centralize access to the main WebSnap components. It initializes the Web context for new request messages and locates the appropriate components to handle the request message. TWebAppComponents publishes properties that provide access to the components that perform many common tasks in a WebSnap application. These properties include

PageDispatcher (see page 24965), which locates the Web page modules to process incoming request messages.

AdapterDispatcher (see page 24963), which encodes calls to adapter actions and fields, and which handles HTML form submissions and requests for dynamic images by calling adapter actions and fields.

Sessions (see page 24966), which manages the sessions for each caller.

ApplicationAdapter (see page 24963), which implements the fields and actions of the Application variable in server-side scripts.

EndUserAdapter (see page 24964), which implements the fields and actions of the EndUser variable in server-side scripts.

UserListService (see page 24966), which maintains a list of end users and their passwords and access privileges.

LocateFileService (see page 24965), which controls access to the templates and server-side script include files that are used while processing requests.

DispatchActions (see page 24964), which maintains a list of action items that can respond to request messages.

The components these properties reference are optional. The WebSnap Application wizard automatically creates components and fills in these properties for any components that you select.

Add TWebAppComponents to the application module of the WebSnap application to provide centralized access to the other components that govern the behavior of the WebSnap application.

179.1.16.1 TWebAppComponents Members

The following tables list the members exposed by TWebAppComponents.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	Access_Request (see page 24937)	Returns the value of the Access_Request property.
	Access_Response (see page 24938)	Returns the value of the Access_Response property.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	CreateContext (see page 24911)	Creates the Web context for the current request message.
V	CreateContext (see page 24938)	Creates the Web context for the current request message.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoAfterDispatch (see page 24939)	Generates an OnAfterDispatch (see page 24948) event.
	DoBeforeDispatch (see page 24939)	Generates an OnBeforeDispatch (see page 24948) event.
V	DoneModule (see page 24939)	Performs any final tasks before deactivating a Web module.
	FinishContext (see page 24940)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetAdapterDispatcher (see page 24912)	Returns the value of the AdapterDispatcher (see page 24919) property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetExceptionHandler (see page 24940)	Returns the object that handles exceptions that arise in the WebSnap application.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetLocateFileService (see page 24912)	Returns the value of the LocateFileService (see page 24920) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetPageDispatcher (see page 24912)	Returns the value of the PageDispatcher (see page 24921) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebApplicationInfo (see page 24913)	Returns the value of the ApplicationAdapter (see page 24919) property.
	GetWebEndUser (see page 24913)	Returns the value of the EndUserAdapter (see page 24920) property.
	GetWebUserList (see page 24913)	Returns the value of the UserListService (see page 24922) property.
	HandleException (see page 24941)	Responds to any exceptions not caught by the WebSnap application.
	HandleRequest (see page 24941)	Directs the processing of the current request message.
V	ImplFinishContext (see page 24914)	Provides the underlying implementation of the FinishContext (see page 24940) method.
V	ImplFinishContext (see page 24942)	Provides the underlying implementation of the FinishContext (see page 24940) method.
V	ImplHandleException (see page 24914)	Provides the underlying implementation of the HandleException (see page 24941) method.
V	ImplHandleException (see page 24942)	Provides the underlying implementation of the HandleException (see page 24941) method.

V	ImplHandleRequest (see page 24915)	Provides the underlying implementation for the HandleRequest (see page 24941) method.
V	ImplHandleRequest (see page 24943)	Provides the underlying implementation for the HandleRequest (see page 24941) method.
V	ImplInitContext (see page 24915)	Provides the underlying implementation for the InitContext (see page 24944) method.
V	ImplInitContext (see page 24943)	Provides the underlying implementation for the InitContext (see page 24944) method.
	InitContext (see page 24944)	Creates and initializes the Web context for the current Web request.
V	InitModule (see page 24916)	Initializes a Web module before the WebSnap application uses it to handle the current request.
V	InitModule (see page 24945)	Initializes a Web module before the WebSnap application uses it to handle the current request.
V	InitModules (see page 24917)	Initializes all the currently active Web modules.
V	InitModules (see page 24945)	Initializes all the currently active Web modules.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	ListRequestHandlers (see page 24917)	Fills an interface list with the interfaces of all dispatcher components in the WebSnap application.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	ModuleAdded (see page 24946)	Initializes a Web module when it is activated while processing the current request.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 24918)	Updates the TCustomWebAppComponents (see page 24906) properties when components are added to or removed from the application.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Request (see page 24946)	Provides access to the current request object.
R	Response (see page 24947)	Provides access to the current response object.
R	WebModules (see page 24947)	Lists the Web modules available in the WebSnap application.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FinishContext (see page 15228)	Performs any final tasks after the WebSnap application has finished processing a request.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterDispatcher (see page 19706)	This is GetAdapterDispatcher, a member of class IGetWebAppComponents.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetExceptionHandler (see page 15229)	Returns the object that handles exceptions that arise in the WebSnap application.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetLocateFileService (see page 15319)	This is GetLocateFileService, a member of class IGetLocateFileService.
	GetLocateFileService (see page 19706)	This is GetLocateFileService, a member of class IGetWebAppComponents.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetPageDispatcher (see page 19706)	This is GetPageDispatcher, a member of class IGetWebAppComponents.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetWebApplicationInfo (see page 19706)	This is GetWebApplicationInfo, a member of class IGetWebAppComponents.
	GetWebEndUser (see page 19706)	This is GetWebEndUser, a member of class IGetWebAppComponents.
	GetWebUserList (see page 19707)	This is GetWebUserList, a member of class IGetWebAppComponents.
	HandleException (see page 15236)	This is HandleException, a member of class IWebExceptionHandler.
	HandleRequest (see page 15229)	Directs the processing of the current request message.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitContext (see page 15229)	Creates and initializes the Web context for the current Web request.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Request (see page 15235)	Returns the Web request object for the HTTP request message to which the application is responding.
	Response (see page 15235)	Returns the Web response object for the HTTP response message that the application is assembling.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	ExceptionHandler (see page 15230)	Indicates the object that handles exceptions that arise when processing a request message.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebApplicationInfo (see page 19707)	This is WebApplicationInfo, a member of class IGetWebAppComponents.
R	WebEndUser (see page 19707)	This is WebEndUser, a member of class IGetWebAppComponents.

Published Events

	Name	Description
	OnAfterDispatch (see page 24967)	Occurs after the WebSnap application generates the content of a response message.
	OnBeforeDispatch (see page 24967)	Occurs before TMultiModuleWebAppServices passes an HTTP request onto the other components in the application for processing.
	OnException (see page 24968)	Occurs when an exception escapes all try blocks in the WebSnap application.

Published Properties

	Name	Description
	AdapterDispatcher (see page 24963)	Provides access to the current adapter dispatcher object.
	ApplicationAdapter (see page 24963)	Provides access to the application adapter.
	DispatchActions (see page 24964)	Provides access to an action-based Web dispatcher.
	EndUserAdapter (see page 24964)	Provides access to an end-user adapter.
	LocateFileService (see page 24965)	Provides access to the file location service.
	Name (see page 2805)	Specifies the name of the component as referenced in code.

	PageDispatcher (see page 24965)	Provides access to the current page dispatcher object.
	Sessions (see page 24966)	Provides access to the sessions service.
	Tag (see page 2808)	Stores an integer value as part of a component.
	UserListService (see page 24966)	Provides access to the file location service.

179.1.16.2 TWebAppComponents Properties

The properties of the TWebAppComponents class are listed here.

Published Properties

	Name	Description
	AdapterDispatcher (see page 24963)	Provides access to the current adapter dispatcher object.
	ApplicationAdapter (see page 24963)	Provides access to the application adapter.
	DispatchActions (see page 24964)	Provides access to an action-based Web dispatcher.
	EndUserAdapter (see page 24964)	Provides access to an end-user adapter.
	LocateFileService (see page 24965)	Provides access to the file location service.
	PageDispatcher (see page 24965)	Provides access to the current page dispatcher object.
	Sessions (see page 24966)	Provides access to the sessions service.
	UserListService (see page 24966)	Provides access to the file location service.

179.1.16.2.1 TWebAppComponents.AdapterDispatcher Property

Provides access to the current adapter dispatcher object.

Pascal

```
property AdapterDispatcher: IAdapterDispatcher;
```

C++

```
__property IAdapterDispatcher AdapterDispatcher;
```

Description

Use AdapterDispatcher to get or set the interface of the adapter dispatcher. The adapter dispatcher is responsible for encoding the hidden fields and HREF values that represent calls to adapter fields and actions, and for interpreting the fields in incoming requests by calling adapters to execute their actions or to fetch image field values.

Typically, AdapterDispatcher is the interface of a TAdapterDispatcher object. If the WebSnap application does not include a TAdapterDispatcher component, or some other component that implements the IAdapterDispatcher interface, the value of this property is nil (Delphi) or NULL (C++). When AdapterDispatcher is nil (Delphi) or NULL (C++), the WebSnap application can't execute adapter actions or fetch image fields from adapters.

See Also

TAdapterDispatcher (see page 24840)

DispatchActions (see page 24919)

179.1.16.2.2 TWebAppComponents.ApplicationAdapter Property

Provides access to the application adapter.

Pascal

```
property ApplicationAdapter: IWebApplicationInfo;
```

C++

```
__property IWebApplicationInfo ApplicationAdapter;
```

Description

Use ApplicationAdapter to get or set the interface of the application adapter. The application adapter implements the fields and actions of the Application variable in server-side scripts.

Typically, ApplicationAdapter is the interface of a TApplicationAdapter object. If the WebSnap application does not include a TApplicationAdapter component, or some other component that implements the IWebApplicationInfo interface, the value of this property is nil (Delphi) or NULL (C++). When ApplicationAdapter is nil (Delphi) or NULL (C++), the WebSnap application can't use the Application variable in server-side scripts.

See Also

TApplicationAdapter (see page 24032)

179.1.16.2.3 TWebAppComponents.DispatchActions Property

Provides access to an action-based Web dispatcher.

Pascal

```
property DispatchActions: IWebDispatchActions;
```

C++

```
__property IWebDispatchActions DispatchActions;
```

Description

Read DispatchActions to determine whether the WebSnap application contains a Web dispatcher. A Web dispatcher maintains a list of action items that respond to incoming request messages. Web dispatchers can appear in both WebSnap and Web broker applications.

The IWebDispatchActions interface that is the value of DispatchActions does nothing special other than identify its component as a Web dispatcher. You can, however, use this interface to obtain other interfaces that belong to the Web dispatcher.

Typically, DispatchActions is the interface of a TWebDispatcher object. If the WebSnap application does not include a TWebDispatcher component, or some other component that implements the IWebDispatchActions interface, the value of this property is nil (Delphi) or NULL (C++).

See Also

TWebModule (see page 15154)

TWebDispatcher (see page 15147)

179.1.16.2.4 TWebAppComponents.EndUserAdapter Property

Provides access to an end-user adapter.

Pascal

```
property EndUserAdapter: IWebEndUser;
```

C++

```
__property IWebEndUser EndUserAdapter;
```

Description

Use EndUserAdapter to get or set the interface of the end user adapter. An end user adapter maintains information about the

current end user, and supports the actions for logging users in and out. End user adapters implement the fields and actions of the EndUser variable in server-side scripts.

Typically, EndUserAdapter is the interface of a TEndUserAdapter or TEndUserSessionAdapter object. If the WebSnap application does not include a component that implements the IWebEndUser interface, the value of this property is nil (Delphi) or NULL (C++). When EndUserAdapter is nil (Delphi) or NULL (C++), the WebSnap application can't use the EndUser variable in server-side scripts or take advantage of the login support that end user adapters provide.

See Also

TEndUserAdapter (see page 24421)

TEndUserSessionAdapter (see page 24458)

179.1.16.2.5 TWebAppComponents.LocateFileService Property

Provides access to the file location service.

Pascal

```
property LocateFileService: ILocateFileService;
```

C++

```
__property ILocateFileService LocateFileService;
```

Description

Use LocateFileService to get or set the interface of the file location service. The file location service controls access to the templates and server-side script include files that are used while processing requests.

Typically, LocateFileService is the interface of a TLocateFileService object. If the WebSnap application does not include a component that implements the ILocateFileService interface, the value of this property is nil (Delphi) or NULL (C++). When LocateFileService is nil (Delphi) or NULL (C++), the WebSnap application can still use template files and include files, but you have less control over how the application locates these files (that is, they must reside in the current directory or references to them must be qualified).

See Also

TLocateFileService (see page 24928)

179.1.16.2.6 TWebAppComponents.PageDispatcher Property

Provides access to the current page dispatcher object.

Pascal

```
property PageDispatcher: IPageDispatcher;
```

C++

```
__property IPageDispatcher PageDispatcher;
```

Description

Use PageDispatcher to get or set the interface of the page dispatcher. The page dispatcher is responsible for interpreting requests for specific Web pages, checking the access rights for requested Web pages, and dispatching Web page requests to the appropriate Web page module.

Typically, PageDispatcher is the interface of a TPageDispatcher object. If the WebSnap application does not include a TPageDispatcher component, or some other component that implements the IPageDispatcher interface, the value of this property is nil (Delphi) or NULL (C++). When PageDispatcher is nil (Delphi) or NULL (C++), the WebSnap application does not automatically generate Web page content in response to requests that give the Web page name in their target URL.

See Also

TPageDispatcher (see page 24949)

AdapterDispatcher (see page 24919)

179.1.16.2.7 TWebAppComponents.Sessions Property

Provides access to the sessions service.

Pascal

```
property Sessions: ISessionsService;
```

C++

```
__property ISessionsService Sessions;
```

Description

Use Sessions to get or set the interface of the sessions service. The sessions service maintains Web sessions so that the WebSnap application can store temporary information about an end user.

Typically, Sessions is the interface of a TSessionsService object. If the WebSnap application does not include a component that implements the ISessionsService interface, the value of this property is nil (Delphi) or NULL (C++). When Sessions is nil (Delphi) or NULL (C++), the WebSnap application can't use Web sessions to store named information about the current end user.

See Also

TSessionsService (see page 25582)

179.1.16.2.8 TWebAppComponents.UserListService Property

Provides access to the file location service.

Pascal

```
property UserListService: IWebUserList;
```

C++

```
__property IWebUserList UserListService;
```

Description

Use UserListService to get or set the interface of the user list service. The user list service maintains a list of user names, passwords, and access rights. Using this information, it can validate logins and check a user's access rights.

Typically, UserListService is the interface of a TWebUserList object. If the WebSnap application does not include a component that implements the IWebUserList interface, the value of this property is nil (Delphi) or NULL (C++). When UserListService is nil (Delphi) or NULL (C++), the WebSnap application can still allow users to log in and check access rights, but the support for this must be coded using event handlers on the page dispatcher.

See Also

TWebUserList (see page 25625)

TPageDispatcher (see page 24949)

179.1.16.3 TWebAppComponents Events

The events of the TWebAppComponents class are listed here.

Published Events

	Name	Description
	OnAfterDispatch (see page 24967)	Occurs after the WebSnap application generates the content of a response message.
	OnBeforeDispatch (see page 24967)	Occurs before TMultiModuleWebAppServices passes an HTTP request onto the other components in the application for processing.
	OnException (see page 24968)	Occurs when an exception escapes all try blocks in the WebSnap application.

179.1.16.3.1 TWebAppComponents.OnAfterDispatch Event

Occurs after the WebSnap application generates the content of a response message.

Pascal

```
property OnAfterDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent OnAfterDispatch;
```

Description

Write an AfterDispatch event handler to perform any final actions before sending a response back to the Web browser that sent the WebSnap application a request. OnAfterDispatch occurs after the other components in the WebSnap application have attempted to generate a response for the current request. It does not occur if the response message has already been sent back to the caller.

If the AfterDispatch event handler sets the Handled parameter to true, the WebSnap application sends the response back to the caller after the event handler exits. If it sets the Handled parameter to false, the WebSnap application does not send the response message.

Note: Although TMultiModuleWebAppServices introduces the OnAfterDispatch event, it never calls the event handler. Descendant classes add the code to generate this event after dispatching a request message.

See Also

OnBeforeDispatch (see page 24948)

179.1.16.3.2 TWebAppComponents.OnBeforeDispatch Event

Occurs before TMultiModuleWebAppServices passes an HTTP request onto the other components in the application for processing.

Pascal

```
property OnBeforeDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent OnBeforeDispatch;
```

Description

Write an OnBeforeDispatch event handler to initialize the response to an HTTP request or to handle any messages that the other components in the WebSnap application can't handle. If the event handler completely handles the request, filling out the response object so that it is ready to send, it should set the Handled parameter to true, thereby preventing this component from trying to locate a component in the WebSnap application that can handle the message.

Note: Although TMultiModuleWebAppServices introduces the OnBeforeDispatch event, it never calls the event handler. Descendant classes add the code to generate this event before dispatching a request message.

See Also

OnAfterDispatch (see page 24948)

179.1.16.3.3 TWebAppComponents.OnException Event

Occurs when an exception escapes all try blocks in the WebSnap application.

Pascal

```
property OnException: TWebExceptionEvent;
```

C++

```
__property TWebExceptionEvent OnException;
```

Description

Write an OnException event handler respond to exceptions that arise while processing a request message. OnException occurs when an exception is not caught by the components that process the request message. If the OnException event handler does not handle the exception, TCustomWebAppComponents updates the response message so that it describes the exception to the caller.

See Also

OnAfterDispatch (see page 24948)

179.1.17 TWebSession Class

TWebSession stores temporary information associated with the user in a WebSnap application.

Class Hierarchy

Pascal

```
TWebSession = class(TAbstractWebSession, IWebSessionAccess);
```

C++

```
class TWebSession : public TAbstractWebSession, public IWebSessionAccess;
```

File

WebDisp

Description

TWebSession represents an active Web session. Use its Values property to get or set application-specific information about the current user. If your application uses sessions to store user information, you need to include a TSessionsService component in the Web application module.

A Web session stores information that persists over multiple calls from the same user. This information is stored as a set of Variant values that are indexed by name. You can use these named Variants (see page 23567) to store application-defined information that is associated with a single user.

There is no need to instantiate session objects. A Session is created automatically when the WebSnap application receives an HTTP request message. The new session is given a unique identifier, which is added (in a cookie) to response messages, so that the same session can be used for subsequent requests from the same caller. The session persists until either the user logs

out, a specified time elapses with no request messages from the caller, or the application explicitly calls the session's Terminate (see page 24973) method.

See Also

TSessionsService (see page 25582)

TAbstractWebContext (see page 24644)

TInvalidWebSession (see page 24923)

179.1.17.1 TWebSession Members

The following tables list the members exposed by TWebSession.

Protected Methods

	Name	Description
	GetSessionID (see page 24971)	Returns the value of the SessionID property.
	GetTimeoutMinutes (see page 24971)	Returns the value of the TimeoutMinutes property.
	GetValue (see page 24972)	Returns the value associated with a specified name.
	SetTimeoutMinutes (see page 24972)	Sets the value of the TimoutMinutes property.
	SetValue (see page 24972)	Assigns a value to a specified name.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 24970)	This is the overview for the Create constructor overload.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetSessionsService (see page 24976)	This is GetSessionsService, a member of class IWebSessionAccess.
	GetWebSession (see page 24976)	This is GetWebSession, a member of class IWebSessionAccess.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Terminate (see page 24973)	Terminates the session.
	UpdateResponse (see page 24973)	Adds the session ID to a cookie in a response message.

Public Properties

	Name	Description
	SessionID (see page 15087)	Uniquely identifies the Session.
	TimeoutMinutes (see page 15087)	Indicates the time, in minutes, that a session can be idle before it terminates itself.
	Values (see page 15088)	Lists the named values that the session stores.

179.1.17.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TWebSession.Create (ISessionsService, TSessionID) (see page 24970)	This is Create, a member of class TWebSession.
	TWebSession.Create (ISessionsService, TWebRequest, TWebResponse) (see page 24970)	This is Create, a member of class TWebSession.

179.1.17.2.1 TWebSession.Create Constructor (ISessionsService, TSessionID)

Pascal

```
constructor Create(ASessions: ISessionsService; AID: TSessionID); overload;
```

C++

```
__fastcall TWebSession(ISessionsService ASessions, TSessionID AID);
```

Description

This is Create, a member of class TWebSession.

179.1.17.2.2 TWebSession.Create Constructor (ISessionsService, TWebRequest, TWebResponse)

Pascal

```
constructor Create(ASessions: ISessionsService; ARequest: TWebRequest; AResponse:
TWebResponse); overload;
```

C++

```
__fastcall TWebSession(ISessionsService ASessions, TWebRequest ARequest, TWebResponse
AResponse);
```

Description

This is Create, a member of class TWebSession.

179.1.17.3 TWebSession Methods

The methods of the TWebSession class are listed here.

Protected Methods

	Name	Description
	GetSessionID (see page 24971)	Returns the value of the SessionID property.
	GetTimeoutMinutes (see page 24971)	Returns the value of the TimeoutMinutes property.
	GetValue (see page 24972)	Returns the value associated with a specified name.
	SetTimeoutMinutes (see page 24972)	Sets the value of the TimoutMinutes property.
	SetValue (see page 24972)	Assigns a value to a specified name.

Public Methods

	Name	Description
	Terminate (see page 24973)	Terminates the session.
	UpdateResponse (see page 24973)	Adds the session ID to a cookie in a response message.

179.1.17.3.1 TWebSession.GetSessionID Method

Returns the value of the SessionID property.

Pascal

```
function GetSessionID: string; override;
```

C++

```
virtual __fastcall AnsiString GetSessionID();
```

Description

GetSessionID is the protected read implementation of the SessionID property.

In TAbstractWebSession, GetSessionID is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the SessionID property.

See Also

SessionID (see page 15087)

179.1.17.3.2 TWebSession.GetTimeoutMinutes Method

Returns the value of the TimeoutMinutes property.

Pascal

```
function GetTimeoutMinutes: Integer; override;
```

C++

```
virtual __fastcall int GetTimeoutMinutes();
```

Description

GetTimeoutMinutes is the protected read implementation of the TimeoutMinutes property.

In TAbstractWebSession, GetTimeoutMinutes is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the TimeoutMinutes property.

See Also

TimeoutMinutes (see page 15087)

SetTimeoutMinutes (see page 15085)

179.1.17.3.3 TWebSession.GetValue Method

Returns the value associated with a specified name.

Pascal

```
function GetValue(const AName: string): Variant; override;
```

C++

```
virtual __fastcall Variant GetValue(const AnsiString AName);
```

Description

GetValue is the protected read implementation of the Values property.

AName is the name of the value to return.

In TAbstractWebSession, GetValue is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Values property.

See Also

Values (see page 15088)

SetValue (see page 15085)

179.1.17.3.4 TWebSession.SetTimeoutMinutes Method

Sets the value of the TimoutMinutes property.

Pascal

```
procedure SetTimeoutMinutes(AValue: Integer); override;
```

C++

```
virtual __fastcall SetTimeoutMinutes(int AValue);
```

Description

SetTimeoutMinutes is the protected write implementation of the TimeoutMinutes property.

AValue is the number of minutes that can elapse before the session terminates itself.

In TAbstractWebSession, SetTimeoutMinutes is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the TimeoutMinutes property.

See Also

GetTimeoutMinutes (see page 15084)

TimeoutMinutes (see page 15087)

179.1.17.3.5 TWebSession.SetValue Method

Assigns a value to a specified name.

Pascal

```
procedure SetValue(const AName: string; const AValue: Variant); override;
```

C++

```
virtual __fastcall SetValue(const AnsiString AName, const Variant AValue);
```

Description

SetValue is the protected write implementation of the Value property.

AName is the name of the value to store.

AValue is the value to store under the specified name.

In TAbstractWebSession, SetValue is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Values property.

See Also

GetValue (see page 15084)

Values (see page 15088)

179.1.17.3.6 TWebSession.Terminate Method

Terminates the session.

Pascal

```
procedure Terminate; override;
```

C++

```
virtual __fastcall Terminate();
```

Description

Call Terminate to explicitly terminate a session. Once a session has been terminated, the application can no longer access its Values.

Sessions are terminated automatically when the application does not receive any requests for the session for TimoutMinutes minutes.

See Also

TimeoutMinutes

Values

179.1.17.3.7 TWebSession.UpdateResponse Method

Adds the session ID to a cookie in a response message.

Pascal

```
procedure UpdateResponse(AResponse: TWebResponse); override;
```

C++

```
virtual __fastcall UpdateResponse(TWebResponse AResponse);
```

Description

Call UpdateResponse to add the value of the SessionID property to one of the response message's cookies. This allows the user's next request message to use the Values stored in this session.

AResponse is the Web response message to which a cookie with the session ID should be added.

See Also

SessionID

Values

179.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IWebSessionAccess (see page 24975)	This is class WebDisp.IWebSessionAccess.

179.2.1 IWebSessionAccess Interface

Class Hierarchy

WebDisp.IWebSessionAccess

Pascal

```
IWebSessionAccess = interface;
```

C++

```
__interface IWebSessionAccess;
```

File

WebDisp

Description

This is class WebDisp.IWebSessionAccess.

179.2.1.1 IWebSessionAccess Members

The following tables list the members exposed by IWebSessionAccess.

Public Methods

	Name	Description
	GetSessionsService (see page 24976)	This is GetSessionsService, a member of class IWebSessionAccess.
	GetWebSession (see page 24976)	This is GetWebSession, a member of class IWebSessionAccess.

179.2.1.2 IWebSessionAccess Methods

The methods of the IWebSessionAccess class are listed here.

Public Methods

	Name	Description
	GetSessionsService (see page 24976)	This is GetSessionsService, a member of class IWebSessionAccess.
	GetWebSession (see page 24976)	This is GetWebSession, a member of class IWebSessionAccess.

179.2.1.2.1 IWebSessionAccess.GetSessionsService Method

Pascal

```
function GetSessionsService: ISessionsService;
```

C++

```
__fastcall ISessionsService GetSessionsService();
```

Description

This is GetSessionsService, a member of class IWebSessionAccess.

179.2.1.2.2 IWebSessionAccess.GetWebSession Method

Pascal

```
function GetWebSession: IWebSession;
```

C++

```
__fastcall IWebSession GetWebSession();
```

Description

This is GetWebSession, a member of class IWebSessionAccess.

179.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AdapterRequestParamsAsStrings (see page 24977)	This is function WebDisp.AdapterRequestParamsAsStrings.
	DispatchPageName (see page 24977)	This is function WebDisp.DispatchPageName.
	EncodeParamNameValues (see page 24978)	This is function WebDisp.EncodeParamNameValues.
	EncodeParamsAsHTTPQuery (see page 24978)	This is function WebDisp.EncodeParamsAsHTTPQuery.
	ExpandScriptException (see page 24978)	This is function WebDisp.ExpandScriptException.
	ExtractParamNameValues (see page 24979)	This is function WebDisp.ExtractParamNameValues.
	FindDefaultAppPageName (see page 24979)	This is function WebDisp.FindDefaultAppPageName.
	PageNameOfRequest (see page 24979)	This is function WebDisp.PageNameOfRequest.
	RaiseDispatchBlankPageNameException (see page 24980)	This is function WebDisp.RaiseDispatchBlankPageNameException.
	RaisePageDoesNotProvideContent (see page 24980)	This is function WebDisp.RaisePageDoesNotProvideContent.
	RaisePageNotFound (see page 24980)	This is function WebDisp.RaisePageNotFound.
	RedirectToPageName (see page 24981)	This is function WebDisp.RedirectToPageName.
	RegisterAdapterRequestIdentifier (see page 24981)	This is function WebDisp.RegisterAdapterRequestIdentifier.
	SafeRedirect (see page 24981)	Redirects a browser to another specified page.

179.3.1 WebDisp.AdapterRequestParamsAsStrings Function

Pascal

```
function AdapterRequestParamsAsStrings(AAdapterRequest: IUnknown): TStrings;
```

C++

```
TStrings AdapterRequestParamsAsStrings(IUnknown AAdapterRequest);
```

File

WebDisp

Description

This is function WebDisp.AdapterRequestParamsAsStrings.

179.3.2 WebDisp.DispatchPageName Function

Pascal

```
function DispatchPageName(const APageName: string; AResponse: TWebResponse; AFlags:
TDispatchPageFlags): Boolean;
```

C++

```
Boolean DispatchPageName(const AnsiString APageName, TWebResponse AResponse,
TDispatchPageFlags AFlags);
```

File

WebDisp

Description

This is function WebDisp.DispatchPageName.

179.3.3 WebDisp.EncodeParamNameValues Function

Pascal

```
function EncodeParamNameValues(ParamValues: TStrings): string;
```

C++

```
AnsiString EncodeParamNameValues(TStrings ParamValues);
```

File

WebDisp

Description

This is function WebDisp.EncodeParamNameValues.

179.3.4 WebDisp.EncodeParamsAsHTTPQuery Function

Pascal

```
function EncodeParamsAsHTTPQuery(AParams: TStrings): string;
```

C++

```
AnsiString EncodeParamsAsHTTPQuery(TStrings AParams);
```

File

WebDisp

Description

This is function WebDisp.EncodeParamsAsHTTPQuery.

179.3.5 WebDisp.ExpandScriptException Function

Pascal

```
function ExpandScriptException(E: EScriptError): string;
```

C++

```
AnsiString ExpandScriptException(EScriptError E);
```

File

WebDisp

Description

This is function WebDisp.ExpandScriptException.

179.3.6 WebDisp.ExtractParamNameValues Function

Pascal

```
procedure ExtractParamNameValues(const AParam: string; AList: TStrings);
```

C++

```
ExtractParamNameValues(const AnsiString AParam, TStrings AList);
```

File

WebDisp

Description

This is function WebDisp.ExtractParamNameValues.

179.3.7 WebDisp.FindDefaultAppPageName Function

Pascal

```
function FindDefaultAppPageName: string;
```

C++

```
AnsiString FindDefaultAppPageName();
```

File

WebDisp

Description

This is function WebDisp.FindDefaultAppPageName.

179.3.8 WebDisp.PageNameOfRequest Function

Pascal

```
function PageNameOfRequest(ARequest: TWebRequest): string;
```

C++

```
AnsiString PageNameOfRequest(TWebRequest ARequest);
```

File

WebDisp

Description

This is function WebDisp.PageNameOfRequest.

179.3.9 WebDisp.RaiseDispatchBlankPageNameException Function

Pascal

```
procedure RaiseDispatchBlankPageNameException;
```

C++

```
RaiseDispatchBlankPageNameException();
```

File

WebDisp

Description

This is function WebDisp.RaiseDispatchBlankPageNameException.

179.3.10 WebDisp.RaisePageDoesNotProvideContent Function

Pascal

```
procedure RaisePageDoesNotProvideContent(const APageName: string);
```

C++

```
RaisePageDoesNotProvideContent(const AnsiString APageName);
```

File

WebDisp

Description

This is function WebDisp.RaisePageDoesNotProvideContent.

179.3.11 WebDisp.RaisePageNotFound Function

Pascal

```
procedure RaisePageNotFound(const APageName: string);
```

C++

```
RaisePageNotFound(const AnsiString APageName);
```

File

WebDisp

Description

This is function WebDisp.RaisePageNotFound.

179.3.12 WebDisp.RedirectToPageName Function

Pascal

```
function RedirectToPageName(const APageName: string; AParams: TStrings; AResponse:
TWebResponse; AFlags: TDispatchPageFlags): Boolean;
```

C++

```
Boolean RedirectToPageName(const AnsiString APageName, TStrings AParams, TWebResponse
AResponse, TDispatchPageFlags AFlags);
```

File

WebDisp

Description

This is function WebDisp.RedirectToPageName.

179.3.13 WebDisp.RegisterAdapterRequestIdentifier Function

Pascal

```
procedure RegisterAdapterRequestIdentifier(const AIdentifier: string);
```

C++

```
RegisterAdapterRequestIdentifier(const AnsiString AIdentifier);
```

File

WebDisp

Description

This is function WebDisp.RegisterAdapterRequestIdentifier.

179.3.14 WebDisp.SafeRedirect Function

Redirects a browser to another specified page.

Pascal

```
procedure SafeRedirect(AResponse: TWebResponse; const AURL: string);
```

C++

```
SafeRedirect(TWebResponse AResponse, const AnsiString AURL);
```

File

WebDisp

Description

Use the SafeRedirect function to send a response which redirects a Web browser to another page. AResponse is the response

object which will send the redirect. AURL is the URL for the page that the browser is being redirected to.

SafeRedirect differs from other redirect methods (such as an override of TWebResponse (see page 24989)'s SendRedirect method) in that it allows browsers to accept cookies. Other redirect methods don't allow cookies to be set during a redirect.

179.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TPageAccessDenied (see page 24983)	This is record WebDisp.TPageAccessDenied.

179.4.1 WebDisp.TPageAccessDenied Enumeration

Pascal

```
TPageAccessDenied = (
  adLoginRequired,
  adCantView
);
```

C++

```
enum TPageAccessDenied {
  adLoginRequired,
  adCantView
};
```

File

WebDisp

Description

This is record WebDisp.TPageAccessDenied.

179.5 Types

The following table lists types in this documentation.

Types

Name	Description
TActivateWebModuleEvent (see page 24984)	This is type WebDisp.TActivateWebModuleEvent.
TCanViewPageEvent (see page 24984)	This is type WebDisp.TCanViewPageEvent.
TDispatchActionEvent (see page 24985)	This is type WebDisp.TDispatchActionEvent.
TDispatchActionHandledEvent (see page 24985)	This is type WebDisp.TDispatchActionHandledEvent.
TDispatchImageEvent (see page 24985)	This is type WebDisp.TDispatchImageEvent.
TDispatchImageHandledEvent (see page 24986)	This is type WebDisp.TDispatchImageHandledEvent.
TDispatchPageEvent (see page 24986)	This is type WebDisp.TDispatchPageEvent.
TDispatchPageHandledEvent (see page 24986)	This is type WebDisp.TDispatchPageHandledEvent.
TDispatchPageParamsHandledEvent (see page 24987)	This is type WebDisp.TDispatchPageParamsHandledEvent.
TGetAdapterRequestParamsEvent (see page 24987)	This is type WebDisp.TGetAdapterRequestParamsEvent.
TLocateFileServiceFindFileEvent (see page 24987)	This is type WebDisp.TLocateFileServiceFindFileEvent.
TLocateFileServiceFindStreamEvent (see page 24988)	This is type WebDisp.TLocateFileServiceFindStreamEvent.
TLoginPagedHandledEvent (see page 24988)	This is type WebDisp.TLoginPagedHandledEvent.
TLoginRequiredEvent (see page 24988)	This is type WebDisp.TLoginRequiredEvent.
TPageAccessDeniedEvent (see page 24989)	This is type WebDisp.TPageAccessDeniedEvent.
TWebRequest (see page 24989)	This is type WebDisp.TWebRequest.
TWebResponse (see page 24989)	This is type WebDisp.TWebResponse.

179.5.1 WebDisp.TActivateWebModuleEvent Type

Pascal

```
TActivateWebModuleEvent = procedure (Sender: TObject; WebModule: TComponent) of object;
```

C++

```
(Sender: TObject; WebModule: TComponent) ( TActivateWebModuleEvent)();
```

File

WebDisp

Description

This is type WebDisp.TActivateWebModuleEvent.

179.5.2 WebDisp.TCanViewPageEvent Type

Pascal

```
TCanViewPageEvent = procedure (Sender: TObject; const PageName: string; var CanView: Boolean;
var AHandled: Boolean) of object;
```

C++

```
(Sender: TObject; const PageName: string; var CanView: Boolean; var AHandled: Boolean) (
TCanViewPageEvent)();
```

File

WebDisp

Description

This is type WebDisp.TCanViewPageEvent.

179.5.3 WebDisp.TDispatchActionEvent Type

Pascal

```
TDispatchActionEvent = procedure (Sender: TObject; Action: TObject; Params: TStrings) of
object;
```

C++

```
(Sender: TObject; Action: TObject; Params: TStrings) ( TDispatchActionEvent)();
```

File

WebDisp

Description

This is type WebDisp.TDispatchActionEvent.

179.5.4 WebDisp.TDispatchActionHandledEvent Type

Pascal

```
TDispatchActionHandledEvent = procedure (Sender: TObject; Action: TObject; Params: TStrings;
var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; Action: TObject; Params: TStrings; var Handled: Boolean) (
TDispatchActionHandledEvent)();
```

File

WebDisp

Description

This is type WebDisp.TDispatchActionHandledEvent.

179.5.5 WebDisp.TDispatchImageEvent Type

Pascal

```
TDispatchImageEvent = procedure (Sender: TObject; Image: TObject; Params: TStrings) of object;
```

C++

```
(Sender: TObject; Image: TObject; Params: TStrings) ( TDispatchImageEvent)();
```

File

WebDisp

Description

This is type WebDisp.TDispatchImageEvent.

179.5.6 WebDisp.TDispatchImageHandledEvent Type

Pascal

```
TDispatchImageHandledEvent = procedure (Sender: TObject; Image: TObject; Params: TStrings; var
Handled: Boolean) of object;
```

C++

```
(Sender: TObject; Image: TObject; Params: TStrings; var Handled: Boolean) (
TDispatchImageHandledEvent)();
```

File

WebDisp

Description

This is type WebDisp.TDispatchImageHandledEvent.

179.5.7 WebDisp.TDispatchPageEvent Type

Pascal

```
TDispatchPageEvent = procedure (Sender: TObject; const PageName: string) of object;
```

C++

```
(Sender: TObject; const PageName: string) ( TDispatchPageEvent)();
```

File

WebDisp

Description

This is type WebDisp.TDispatchPageEvent.

179.5.8 WebDisp.TDispatchPageHandledEvent Type

Pascal

```
TDispatchPageHandledEvent = procedure (Sender: TObject; const PageName: string; var Handled:
Boolean) of object;
```

C++

```
(Sender: TObject; const PageName: string; var Handled: Boolean) ( TDispatchPageHandledEvent)();
```

File

WebDisp

Description

This is type WebDisp.TDispatchPageHandledEvent.

179.5.9 WebDisp.TDispatchPageParamsHandledEvent Type

Pascal

```
TDispatchPageParamsHandledEvent = procedure (Sender: TObject; const PageName: string; Params:
TStrings; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; const PageName: string; Params: TStrings; var Handled: Boolean) (
TDispatchPageParamsHandledEvent)();
```

File

WebDisp

Description

This is type WebDisp.TDispatchPageParamsHandledEvent.

179.5.10 WebDisp.TGetAdapterRequestParamsEvent Type

Pascal

```
TGetAdapterRequestParamsEvent = procedure (Sender: TObject; Handler: IUnknown; var Identifer:
string; Params: TStrings) of object;
```

C++

```
(Sender: TObject; Handler: IUnknown; var Identifer: string; Params: TStrings) (
TGetAdapterRequestParamsEvent)();
```

File

WebDisp

Description

This is type WebDisp.TGetAdapterRequestParamsEvent.

179.5.11 WebDisp.TLocateFileServiceFindFileEvent Type

Pascal

```
TLocateFileServiceFindFileEvent = procedure (ASender: TObject; AComponent: TComponent; const
AFileName: string; var AFoundFile: string; var AHandled: Boolean) of object;
```

C++

```
(ASender: TObject; AComponent: TComponent; const AFileName: string; var AFoundFile: string;
var AHandled: Boolean) ( TLocateFileServiceFindFileEvent)();
```

File

WebDisp

Description

This is type WebDisp.TLocateFileServiceFindFileEvent.

179.5.12 WebDisp.TLocateFileServiceFindStreamEvent Type

Pascal

```
TLocateFileServiceFindStreamEvent = procedure (ASender: TObject; AComponent: TComponent; const
AFileName: string; var AFoundStream: TStream; var AOwned: Boolean; var AHandled: Boolean) of
object;
```

C++

```
(ASender: TObject; AComponent: TComponent; const AFileName: string; var AFoundStream: TStream;
var AOwned: Boolean; var AHandled: Boolean) ( TLocateFileServiceFindStreamEvent)();
```

File

WebDisp

Description

This is type WebDisp.TLocateFileServiceFindStreamEvent.

179.5.13 WebDisp.TLoginPagedHandledEvent Type

Pascal

```
TLoginPagedHandledEvent = procedure (Sender: TObject; const PageName: string; var Handled:
Boolean) of object;
```

C++

```
(Sender: TObject; const PageName: string; var Handled: Boolean) ( TLoginPagedHandledEvent)();
```

File

WebDisp

Description

This is type WebDisp.TLoginPagedHandledEvent.

179.5.14 WebDisp.TLoginRequiredEvent Type

Pascal

```
TLoginRequiredEvent = procedure (Sender: TObject; const PageName: string; var Required:
```

```
Boolean) of object;
```

C++

```
(Sender: TObject; const PageName: string; var Required: Boolean) ( TLoginRequiredEvent)();
```

File

WebDisp

Description

This is type WebDisp.TLoginRequiredEvent.

179.5.15 WebDisp.TPageAccessDeniedEvent Type

Pascal

```
TPageAccessDeniedEvent = procedure (Sender: TObject; const PageName: string; Reason:
TPageAccessDenied; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; const PageName: string; Reason: TPageAccessDenied; var Handled: Boolean) (
TPageAccessDeniedEvent)();
```

File

WebDisp

Description

This is type WebDisp.TPageAccessDeniedEvent.

179.5.16 WebDisp.TWebRequest Type

Pascal

```
TWebRequest = HTTPApp.TWebRequest;
```

C++

```
HTTPApp.TWebRequest TWebRequest;
```

File

WebDisp

Description

This is type WebDisp.TWebRequest.

179.5.17 WebDisp.TWebResponse Type

Pascal

```
TWebResponse = HTTPApp.TWebResponse;
```

C++

```
HTTPApp.TWebResponse TWebResponse;
```

File

WebDisp

Description

This is type WebDisp.TWebResponse.

179.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
sFailurePageParamName (see page 24991)	This is constant WebDisp.sFailurePageParamName.
sPageParamName (see page 24991)	This is constant WebDisp.sPageParamName.
sRequestIDParamName (see page 24992)	This is constant WebDisp.sRequestIDParamName.
sSessionCookieName (see page 24992)	This is constant WebDisp.sSessionCookieName.
sSuccessPageParamName (see page 24992)	This is constant WebDisp.sSuccessPageParamName.

179.6.1 WebDisp.sFailurePageParamName Constant

Pascal

```
sFailurePageParamName = '__fp';
```

C++

```
sFailurePageParamName = '__fp';
```

File

WebDisp

Description

This is constant WebDisp.sFailurePageParamName.

179.6.2 WebDisp.sPageParamName Constant

Pascal

```
sPageParamName = '__p';
```

C++

```
sPageParamName = '__p';
```

File

WebDisp

Description

This is constant WebDisp.sPageParamName.

179.6.3 WebDisp.sRequestIDParamName Constant

Pascal

```
sRequestIDParamName = '__id';
```

C++

```
sRequestIDParamName = '__id';
```

File

WebDisp

Description

This is constant WebDisp.sRequestIDParamName.

179.6.4 WebDisp.sSessionCookieName Constant

Pascal

```
sSessionCookieName = 'WebBrokerSessionID';
```

C++

```
sSessionCookieName = 'WebBrokerSessionID';
```

File

WebDisp

Description

This is constant WebDisp.sSessionCookieName.

179.6.5 WebDisp.sSuccessPageParamName Constant

Pascal

```
sSuccessPageParamName = '__sp';
```

C++

```
sSuccessPageParamName = '__sp';
```

File

WebDisp

Description

This is constant WebDisp.sSuccessPageParamName.

180 WebFact Namespace

This is namespace WebFact.

Topics

Name	Description
Classes (see page 24994)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 25022)	The following table lists structs, records, enums in this documentation.
Types (see page 25023)	The following table lists types in this documentation.

Classes

	Name	Description
	TBaseWebModuleFactory (see page 24994)	TBaseWebModuleFactory is the base class for Web module factory objects whose generated Web modules do not create Web pages.
	TBaseWebPageInfo (see page 24999)	TBaseWebPageInfo stores information about a page registered with a Web page module factory.
	TBaseWebPageModuleFactory (see page 25004)	TBaseWebPageModuleFactory is the base class for Web module factory objects whose generated Web modules create Web pages.
	TWebAppDataModuleFactory (see page 25009)	TWebAppDataModule is the factory object for WebSnap application data modules.
	TWebAppPageModuleFactory (see page 25012)	TWebAppPageModuleFactory is the factory object for application Web page modules in a WebSnap application.
	TWebDataModuleFactory (see page 25014)	TWebDataModuleFactory is the factory class for WebSnap data modules.
	TWebPageInfo (see page 25016)	TWebPageInfo describes a WebSnap page.
	TWebPageModuleFactory (see page 25019)	TWebPageModuleFactory is the factory object for Web page modules in a WebSnap application.

Structs, Records, Enums

	Name	Description
	TWebPageAccessFlags (see page 25022)	This is record WebFact.TWebPageAccessFlags.

Types

Name	Description
TWebPageAccess (see page 25023)	This is type WebFact.TWebPageAccess.

180.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TBaseWebModuleFactory (see page 24994)	TBaseWebModuleFactory is the base class for Web module factory objects whose generated Web modules do not create Web pages.
	TBaseWebPageInfo (see page 24999)	TBaseWebPageInfo stores information about a page registered with a Web page module factory.
	TBaseWebPageModuleFactory (see page 25004)	TBaseWebPageModuleFactory is the base class for Web module factory objects whose generated Web modules create Web pages.
	TWebAppDataModuleFactory (see page 25009)	TWebAppDataModule is the factory object for WebSnap application data modules.
	TWebAppPageModuleFactory (see page 25012)	TWebAppPageModuleFactory is the factory object for application Web page modules in a WebSnap application.
	TWebDataModuleFactory (see page 25014)	TWebDataModuleFactory is the factory class for WebSnap data modules.
	TWebPageInfo (see page 25016)	TWebPageInfo describes a WebSnap page.
	TWebPageModuleFactory (see page 25019)	TWebPageModuleFactory is the factory object for Web page modules in a WebSnap application.

180.1.1 TBaseWebModuleFactory Class

TBaseWebModuleFactory is the base class for Web module factory objects whose generated Web modules do not create Web pages.

Class Hierarchy

Pascal

```
TBaseWebModuleFactory = class(TAbstractWebModuleFactory);
```

C++

```
class TBaseWebModuleFactory : public TAbstractWebModuleFactory;
```

File

WebFact

Description

All WebSnap modules are registered with the WebSnap application using a factory object. The factory object is responsible for creating Web module instance. Applications can set flags for each factory object to indicate when it should create instances of its Web module and whether it can cache Web module instance for reuse with subsequent request messages.

TBaseWebModuleFactory descendants generate Web modules that do generate Web pages. These Web modules can contain components, such as adapters, that perform tasks for the WebSnap application.

Do not create instances of TBaseWebModuleFactory. Although TBaseWebModuleFactory provides an implementation for the abstract or, in C++ terminology, pure virtual methods introduced by its ancestors, it is intended only as a base class. Instead, create instances of a TBaseWebModuleFactory descendant such as TWebDataModuleFactory (see page 25014) or TWebAppDataModuleFactory (see page 25009).

See Also

TWebAppDataModuleFactory (see page 25009)

TWebDataModuleFactory (see page 25014)

TWebRequestHandler (see page 25448)

180.1.1.1 TBaseWebModuleFactory Members

The following tables list the members exposed by TBaseWebModuleFactory.

Protected Methods

	Name	Description
	Create (see page 24995)	Creates and initializes an instance of TBaseWebModuleFactory (see page 24994).
V	GetCacheMode (see page 24996)	Returns the value of the CacheMode property.
V	GetComponentClass (see page 24997)	Returns the value of the ComponentClass property.
V	GetCreateMode (see page 24997)	Returns the value of the CreateMode property.
V	GetIsAppModule (see page 24997)	Returns the value of the IsAppModule property.
V	GetModuleName (see page 24998)	Returns the value of the ModuleName property.
V	GetWebPageInfo (see page 15047)	Returns the value of the WebPageInfo (see page 15050) property.

Public Methods

	Name	Description
V	GetModule (see page 24998)	Creates an instance of the associated Web module.
V	PreventDestruction (see page 24998)	Prevents the WebSnap application from freeing the Web module instance.

Public Properties

	Name	Description
R	CacheMode (see page 15048)	Indicates whether the factory can cache Web modules for reuse.
R	ComponentClass (see page 15049)	Specifies the class type of the Web module that this factory creates.
R	CreateMode (see page 15049)	Indicates when the factory creates instances of its Web module.
R	IsAppModule (see page 15050)	Indicates whether the corresponding Web module is the main Web application module.
R	ModuleName (see page 15050)	Specifies the name of the Web module class that this factory instantiates.
R	WebPageInfo (see page 15050)	Describes the attributes of the Web page that the corresponding Web module generates, if any.

180.1.1.2 TBaseWebModuleFactory.Create Constructor

Creates and initializes an instance of TBaseWebModuleFactory (see page 24994).

Pascal

```
constructor Create(AComponentClass: TComponentClass; ACreateMode: TWebModuleCreateMode;
ACacheMode: TWebModuleCacheMode);
```

C++

```
__fastcall TBaseWebModuleFactory(TComponentClass AComponentClass, TWebModuleCreateMode
ACreateMode, TWebModuleCacheMode ACacheMode);
```

Description

Call Create to instantiate a Web module factory. Typically, Web module factory objects are created in the initialization section of the unit for the Web module. The call to the constructor is added by the Web module wizard or WebSnap application wizard when it generates the unit for the Web module.

AComponentClass is the class of the Web module that this factory object creates. It becomes the value of the ComponentClass property.

ACreateMode describes whether the factory should create instances of the Web module for every request message, or only when the application explicitly requests the Web module. This becomes the value of the CreateMode property.

ACacheMode indicates whether the application can cache Web module instances for reuse with subsequent request messages. This becomes the value of the CacheMode property.

See Also

ComponentClass

CreateMode

CacheMode

180.1.1.3 TBaseWebModuleFactory Methods

The methods of the TBaseWebModuleFactory class are listed here.

Protected Methods

	Name	Description
	GetCacheMode (see page 24996)	Returns the value of the CacheMode property.
	GetComponentClass (see page 24997)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 24997)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 24997)	Returns the value of the IsAppModule property.
	GetModuleName (see page 24998)	Returns the value of the ModuleName property.

Public Methods

	Name	Description
	GetModule (see page 24998)	Creates an instance of the associated Web module.
	PreventDestruction (see page 24998)	Prevents the WebSnap application from freeing the Web module instance.

180.1.1.3.1 TBaseWebModuleFactory.GetCacheMode Method

Returns the value of the CacheMode property.

Pascal

```
function GetCacheMode: TWebModuleCacheMode; override;
```

C++

```
virtual __fastcall TWebModuleCacheMode GetCacheMode();
```

Description

GetCacheMode is the protected read implementation of the CacheMode property. It returns the cache mode flag that was supplied to the constructor.

See Also

CacheMode

Create (see page 24995)

180.1.1.3.2 TBaseWebModuleFactory.GetComponentClass Method

Returns the value of the ComponentClass property.

Pascal

```
function GetComponentClass: TComponentClass; override;
```

C++

```
virtual __fastcall TComponentClass GetComponentClass();
```

Description

GetComponentClass is the protected read implementation of the ComponentClass property. It returns the class of the Web page module this factory creates, as supplied to the Web page module factory constructor.

See Also

ComponentClass

Create (see page 24995)

180.1.1.3.3 TBaseWebModuleFactory.GetCreateMode Method

Returns the value of the CreateMode property.

Pascal

```
function GetCreateMode: TWebModuleCreateMode; override;
```

C++

```
virtual __fastcall TWebModuleCreateMode GetCreateMode();
```

Description

GetCreateMode is the protected read implementation of the CreateMode property. It returns the create mode flag that was supplied to the constructor.

See Also

CreateMode

Create (see page 24995)

180.1.1.3.4 TBaseWebModuleFactory.GetIsAppModule Method

Returns the value of the IsAppModule property.

Pascal

```
function GetIsAppModule: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsAppModule();
```

Description

GetIsAppModule is the protected read implementation of the IsAppModule property.

In TBaseWebModuleFactory (see page 24994), GetIsAppModule returns false. Descendant classes that generate Web application modules (such as TWebAppDataModuleFactory (see page 25009)) override this method to return true.

See Also

IsAppModule

TWebAppPageModuleFactory (see page 25012)

180.1.1.3.5 TBaseWebModuleFactory.GetModule Method

Creates an instance of the associated Web module.

Pascal

```
function GetModule: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetModule();
```

Description

Call GetModule to obtain a new instance of the Web module that the factory creates. GetModule creates an instance of the class that the ComponentClass property specifies and returns it.

See Also

ComponentClass

CreateMode

CacheMode

180.1.1.3.6 TBaseWebModuleFactory.GetModuleName Method

Returns the value of the ModuleName property.

Pascal

```
function GetModuleName: string; override;
```

C++

```
virtual __fastcall AnsiString GetModuleName();
```

Description

GetModuleName is the protected read implementation of the ModuleName property. It returns the name of the class that the ComponentClass property specifies.

See Also

ModuleName

ComponentClass

180.1.1.3.7 TBaseWebModuleFactory.PreventDestruction Method

Prevents the WebSnap application from freeing the Web module instance.

Pascal

```
procedure PreventDestruction; override;
```

C++

```
virtual __fastcall PreventDestruction();
```

Description

Call PreventDestruction if the WebSnap application should not free the Web module instance when the current HTTP request message has been fully processed. For example, if another Web module contains a reference to the Web module that this factory creates, you may not want to allow the WebSnap application to free the Web module that this factory creates while the other module exists.

In TBaseWebModuleFactory (see page 24994), PreventDestruction changes the CacheMode property to caCache.

See Also

CacheMode

180.1.2 TBaseWebPageInfo Class

TBaseWebPageInfo stores information about a page registered with a Web page module factory.

Class Hierarchy

Pascal

```
TBaseWebPageInfo = class(TAbstractWebPageInfo);
```

C++

```
class TBaseWebPageInfo : public TAbstractWebPageInfo;
```

File

WebFact

Description

The TBaseWebPageInfo component stores basic information about an HTML page such as the filename, page name, page title, descriptive text, publish status, and access rights. This information is stored with the Web page module factory.

An instance of a TBaseWebPageInfo descendant, such as TWebPageInfo (see page 25016), is supplied to the constructor of the factory object that creates instances of a Web page module. Typically, the code to create this object and supply it to the Web page factory's constructor is generated by a Web page module wizard. You can change this generated code, which appears in the generated unit for the Web page module, to alter the properties of the Web page information object. For example, you may want to add a ViewAccess string to allow your application to limit access to the Web page.

In a WebSnap application, you can access the Web page information object of any Web page module by calling the FindPageInfo method of the application's global Web context.

See Also

TWebPageInfo (see page 25016)

TAbstractWebContext (see page 24644)

TWebAppPageModuleFactory (see page 25012)

TWebPageModuleFactory (see page 25019)

WebContext

180.1.2.1 TBaseWebPageInfo Members

The following tables list the members exposed by TBaseWebPageInfo.

Protected Methods

	Name	Description
	Create (see page 25000)	Creates and initializes a TBaseWebPageInfo (see page 24999) object.
	GetFactory (see page 15062)	Returns the value of the Factory (see page 15066) property.
V	GetIsLoginRequired (see page 25001)	Returns the value of the IsLoginRequired property.
V	GetIsPublished (see page 25002)	Returns the value of the IsPublished property.
V	GetPageDescription (see page 25002)	Returns the value of the PageDescription property.
V	GetPageFile (see page 25002)	Returns the value of the PageFile property.
V	GetPageHREF (see page 25003)	Returns the value of the PageHREF property.
V	GetPageName (see page 25003)	Returns the value of the PageName property.
V	GetPageTitle (see page 25003)	Returns the value of the PageTitle property.
V	GetViewAccess (see page 25004)	Returns the value of the ViewAccess property.
V	SetFactory (see page 25004)	Connects a specified Web page module factory with the Web page information object.

Public Properties

	Name	Description
	Factory (see page 15066)	Specifies the factory object that creates instances of the Web page module this object describes.
R	IsLoginRequired (see page 15066)	Indicates whether callers must log in before accessing the associated Web page.
R	IsPublished (see page 15067)	Indicates whether the WebSnap application automatically displays the associated page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 15068).
R	PageDescription (see page 15067)	Describes the Web page.
R	PageFile (see page 15068)	Identifies an associated file that the Web page module uses to generate its Web page.
R	PageHREF (see page 15068)	Identifies the HREF to use for references to the associated Web page that appear in hypertext links.
R	PageName (see page 15068)	Denotes the logical name for the Web page that the associated module generates
R	PageTitle (see page 15069)	Indicates the title that appears on the associated Web page.
R	ViewAccess (see page 15069)	Indicates the view access string that controls access to the associated Web page.

180.1.2.2 TBaseWebPageInfo.Create Constructor

Creates and initializes a TBaseWebPageInfo (see page 24999) object.

Pascal

```
constructor Create(AAccess: TWebPageAccess; const AFile: string; const AName: string; const
ATitle: string; const ADescription: string; const AViewAccess: string);
```

C++

```
__fastcall TBaseWebPageInfo(TWebPageAccess AAccess, const AnsiString AFile, const AnsiString
AName, const AnsiString ATitle, const AnsiString ADescription, const AnsiString AViewAccess);
```

Description

Typically, you do not need to create a call to the Web page information object constructor. The Web page module wizard generates code to make this call and pass the resulting object instance to the constructor of a Web page module factory. You may, however, want to edit the generated call to the constructor, changing some of the parameter values that the wizard supplies.

AAccess specifies access information for the associated Web page.

AFile specifies the name (or extension) of the file that the associated Web page module uses to generate its content. It becomes the value of the PageFile property.

AName is the logical name of the associated Web page. It becomes the value of the PageName property.

ATitle is the title that appears on the associated Web page. It becomes the value of the PageTitle property.

ADescription is a brief description of the associated Web page. It becomes the value of the PageDescription property.

AViewAccess is the view access string that represents a set of access rights needed to work with the associated Web page. It becomes the value of the ViewAccess property.

See Also

pagefile

pagename

pagetitle

PageDescription

viewaccess

IsPublished

IsLoginRequired

180.1.2.3 TBaseWebPageInfo Methods

The methods of the TBaseWebPageInfo class are listed here.

Protected Methods

	Name	Description
	GetIsLoginRequired (see page 25001)	Returns the value of the IsLoginRequired property.
	GetIsPublished (see page 25002)	Returns the value of the IsPublished property.
	GetPageDescription (see page 25002)	Returns the value of the PageDescription property.
	GetPageFile (see page 25002)	Returns the value of the PageFile property.
	GetPageHREF (see page 25003)	Returns the value of the PageHREF property.
	GetPageName (see page 25003)	Returns the value of the PageName property.
	GetPageTitle (see page 25003)	Returns the value of the PageTitle property.
	GetViewAccess (see page 25004)	Returns the value of the ViewAccess property.
	SetFactory (see page 25004)	Connects a specified Web page module factory with the Web page information object.

180.1.2.3.1 TBaseWebPageInfo.GetIsLoginRequired Method

Returns the value of the IsLoginRequired property.

Pascal

```
function GetIsLoginRequired: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsLoginRequired();
```

Description

GetIsLoginRequired is the protected read implementation of the IsLoginRequired property. It returns true if wpLoginRequired is included in the access flags passed to the constructor.

See Also

IsLoginRequired

Create (see page 25000)

180.1.2.3.2 TBaseWebPageInfo.GetIsPublished Method

Returns the value of the IsPublished property.

Pascal

```
function GetIsPublished: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsPublished();
```

Description

GetIsPublished is the protected read implementation of the IsPublished property. It returns true if wpPublished is included in the access flags passed to the constructor.

See Also

IsPublished

Create (see page 25000)

180.1.2.3.3 TBaseWebPageInfo.GetPageDescription Method

Returns the value of the PageDescription property.

Pascal

```
function GetPageDescription: string; override;
```

C++

```
virtual __fastcall AnsiString GetPageDescription();
```

Description

GetPageDescription is the protected read implementation of the PageDescription property. It returns the description string that is passed to the constructor.

See Also

PageDescription

Create (see page 25000)

180.1.2.3.4 TBaseWebPageInfo.GetPageFile Method

Returns the value of the PageFile property.

Pascal

```
function GetPageFile: string; override;
```

C++

```
virtual __fastcall AnsiString GetPageFile();
```

Description

GetPageFile is the protected read implementation of the PageFile property.

See Also

pagefile

180.1.2.3.5 TBaseWebPageInfo.GetPageHREF Method

Returns the value of the PageHREF property.

Pascal

```
function GetPageHREF: string; override;
```

C++

```
virtual __fastcall AnsiString GetPageHREF();
```

Description

GetPageHREF is the protected read implementation of the PageHREF property. It returns the HTTP path string that is passed to the constructor.

See Also

PageHREF

Create (see page 25000)

180.1.2.3.6 TBaseWebPageInfo.GetPageName Method

Returns the value of the PageName property.

Pascal

```
function GetPageName: string; override;
```

C++

```
virtual __fastcall AnsiString GetPageName();
```

Description

GetPageName is the protected read implementation of the PageName property. It returns the page name that is passed to the constructor.

See Also

pagename

Create (see page 25000)

180.1.2.3.7 TBaseWebPageInfo.GetPageTitle Method

Returns the value of the PageTitle property.

Pascal

```
function GetPageTitle: string; override;
```

C++

```
virtual __fastcall AnsiString GetPageTitle();
```

Description

GetPageTitle is the protected read implementation of the PageTitle property. It returns the title that is passed to the constructor.

See Also

pagetitle

Create (see page 25000)

180.1.2.3.8 TBaseWebPageInfo.GetViewAccess Method

Returns the value of the ViewAccess property.

Pascal

```
function GetViewAccess: string; override;
```

C++

```
virtual __fastcall AnsiString GetViewAccess();
```

Description

GetViewAccess is the protected read implementation of the ViewAccess property. It returns the view access string that is passed to the constructor.

See Also

viewaccess

Create (see page 25000)

180.1.2.3.9 TBaseWebPageInfo.SetFactory Method

Connects a specified Web page module factory with the Web page information object.

Pascal

```
procedure SetFactory(AFactory: TAbstractWebPageModuleFactory); override;
```

C++

```
virtual __fastcall SetFactory(TAbstractWebPageModuleFactory AFactory);
```

Description

SetFactory is the protected write implementation of the Factory property.

AFactory is the Web page module factory object that instantiates the Web page module that this object describes.

See Also

Factory

180.1.3 TBaseWebPageModuleFactory Class

TBaseWebPageModuleFactory is the base class for Web module factory objects whose generated Web modules create Web pages.

Class Hierarchy

Pascal

```
TBaseWebPageModuleFactory = class(TAbstractWebPageModuleFactory);
```

C++

```
class TBaseWebPageModuleFactory : public TAbstractWebPageModuleFactory;
```

File

WebFact

Description

All WebSnap modules are registered with the WebSnap application using a factory object. The factory object is responsible for creating Web module instances. Applications can set flags for each factory object to indicate when it should create instances of its Web module and whether it can cache Web module instances for reuse with subsequent request messages.

TBaseWebPageModuleFactory descendants generate Web page modules. In addition to holding components that perform tasks for the WebSnap application, Web page modules can generate a Web page. The Web page is described by a Web page information object, which the Web page module factory makes available as its WebPageInfo property.

Do not create instances of TBaseWebPageModuleFactory. Although TBaseWebPageModuleFactory provides an implementation for the abstract or, in C++ terminology, pure virtual methods introduced by its ancestors, it is intended only as a base class. Instead, create instances of a TBaseWebPageModuleFactory descendant such as TWebPageModuleFactory (see page 25019) or TWebAppPageModuleFactory (see page 25012).

See Also

TWebAppPageModuleFactory (see page 25012)

TWebPageModuleFactory (see page 25019)

TWebRequestHandler (see page 25448)

TWebPageInfo (see page 25016)

180.1.3.1 TBaseWebPageModuleFactory Members

The following tables list the members exposed by TBaseWebPageModuleFactory.

Protected Methods

	Name	Description
	Create (see page 25006)	Creates and initializes an instance of TBaseWebPageModuleFactory (see page 25004).
V	GetCacheMode (see page 25007)	Returns the value of the CacheMode property.
V	GetComponentClass (see page 25007)	Returns the value of the ComponentClass property.
V	GetCreateMode (see page 25007)	Returns the value of the CreateMode property.
V	GetIsAppModule (see page 25008)	Returns the value of the IsAppModule property.
V	GetModuleName (see page 25009)	Returns the value of the ModuleName property.
V	GetWebPageInfo (see page 15047)	Returns the value of the WebPageInfo (see page 15050) property.
V	GetWebPageInfo (see page 15072)	Returns the value of the WebPageInfo (see page 15050) property.

Public Methods

	Name	Description
V	Destroy (see page 15071)	Disposes of an object instance.
V	GetModule (see page 25008)	Creates an instance of the associated Web module.
V	PreventDestruction (see page 25009)	Prevents the WebSnap application from freeing the Web module instance.

Public Properties

	Name	Description
R	CacheMode (see page 15048)	Indicates whether the factory can cache Web modules for reuse.
R	ComponentClass (see page 15049)	Specifies the class type of the Web module that this factory creates.
R	CreateMode (see page 15049)	Indicates when the factory creates instances of its Web module.
R	IsAppModule (see page 15050)	Indicates whether the corresponding Web module is the main Web application module.
R	ModuleName (see page 15050)	Specifies the name of the Web module class that this factory instantiates.
R	WebPageInfo (see page 15050)	Describes the attributes of the Web page that the corresponding Web module generates, if any.

180.1.3.2 TBaseWebPageModuleFactory.Create Constructor

Creates and initializes an instance of TBaseWebPageModuleFactory (see page 25004).

Pascal

```
constructor Create(AComponentClass: TComponentClass; AWebPageInfo: TAbstractWebPageInfo;
ACreateMode: TWebModuleCreateMode; ACacheMode: TWebModuleCacheMode);
```

C++

```
__fastcall TBaseWebPageModuleFactory(TComponentClass AComponentClass, TAbstractWebPageInfo
AWebPageInfo, TWebModuleCreateMode ACreateMode, TWebModuleCacheMode ACacheMode);
```

Description

Call Create to instantiate a Web page module factory. Typically, Web page module factory objects are created in the initialization section of the unit for the Web page module. The call to the constructor is added by the Web module wizard or WebSnap application wizard when it generates the unit for the Web page module.

AComponentClass is the class of the Web page module that this factory object creates. It becomes the value of the ComponentClass property.

AWebPageInfo is the Web page information object that describes the Web page that this factory's Web page module generates. It becomes the value of the WebPageInfo property.

ACreateMode describes whether the factory should create instances of the Web page module for every request message, or only when the application explicitly requests the Web page module. This becomes the value of the CreateMode property.

ACacheMode indicates whether the application can cache Web page module instances for reuse with subsequent request messages. This becomes the value of the CacheMode property.

See Also

WebPageInfo

ComponentClass

CreateMode

CacheMode

180.1.3.3 TBaseWebPageModuleFactory Methods

The methods of the TBaseWebPageModuleFactory class are listed here.

Protected Methods

	Name	Description
	GetCacheMode (see page 25007)	Returns the value of the CacheMode property.
	GetComponentClass (see page 25007)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 25007)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 25008)	Returns the value of the IsAppModule property.
	GetModuleName (see page 25009)	Returns the value of the ModuleName property.

Public Methods

	Name	Description
	GetModule (see page 25008)	Creates an instance of the associated Web module.
	PreventDestruction (see page 25009)	Prevents the WebSnap application from freeing the Web module instance.

180.1.3.3.1 TBaseWebPageModuleFactory.GetCacheMode Method

Returns the value of the CacheMode property.

Pascal

```
function GetCacheMode: TWebModuleCacheMode; override;
```

C++

```
virtual __fastcall TWebModuleCacheMode GetCacheMode();
```

Description

GetCacheMode is the protected read implementation of the CacheMode property. It returns the cache mode flag that was supplied to the constructor.

See Also

CacheMode

Create (see page 25006)

180.1.3.3.2 TBaseWebPageModuleFactory.GetComponentClass Method

Returns the value of the ComponentClass property.

Pascal

```
function GetComponentClass: TComponentClass; override;
```

C++

```
virtual __fastcall TComponentClass GetComponentClass();
```

Description

GetComponentClass is the protected read implementation of the ComponentClass property. It returns the class of the Web page module this factory creates, as supplied to the Web page module factory constructor.

See Also

ComponentClass

Create (see page 25006)

180.1.3.3.3 TBaseWebPageModuleFactory.GetCreateMode Method

Returns the value of the CreateMode property.

Pascal

```
function GetCreateMode: TWebModuleCreateMode; override;
```

C++

```
virtual __fastcall TWebModuleCreateMode GetCreateMode();
```

Description

GetCreateMode is the protected read implementation of the CreateMode property. It returns the create mode flag that was supplied to the constructor.

See Also

CreateMode

Create (see page 25006)

180.1.3.3.4 TBaseWebPageModuleFactory.GetIsAppModule Method

Returns the value of the IsAppModule property.

Pascal

```
function GetIsAppModule: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsAppModule();
```

Description

GetIsAppModule is the protected read implementation of the IsAppModule property.

In TBaseWebPageModuleFactory (see page 25004), GetIsAppModule returns false. Descendant classes that generate Web application modules (such as TWebAppPageModuleFactory (see page 25012)) override this method to return true.

See Also

IsAppModule

TWebAppPageModuleFactory (see page 25012)

180.1.3.3.5 TBaseWebPageModuleFactory.GetModule Method

Creates an instance of the associated Web module.

Pascal

```
function GetModule: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetModule();
```

Description

Call GetModule to obtain a new instance of the Web module that the factory creates. GetModule creates an instance of the class that the ComponentClass property specifies and returns it.

See Also

ComponentClass

CreateMode

CacheMode

180.1.3.3.6 TBaseWebPageModuleFactory.GetModuleName Method

Returns the value of the ModuleName property.

Pascal

```
function GetModuleName: string; override;
```

C++

```
virtual __fastcall AnsiString GetModuleName();
```

Description

GetModuleName is the protected read implementation of the ModuleName property. It returns the name of the class that the ComponentClass property specifies.

See Also

ModuleName

ComponentClass

180.1.3.3.7 TBaseWebPageModuleFactory.PreventDestruction Method

Prevents the WebSnap application from freeing the Web module instance.

Pascal

```
procedure PreventDestruction; override;
```

C++

```
virtual __fastcall PreventDestruction();
```

Description

Call PreventDestruction if the WebSnap application should not free the Web module instance when the current HTTP request message has been fully processed. For example, if another Web module contains a reference to the Web module that this factory creates, you may not want to allow the WebSnap application to free the Web module that this factory creates while the other module exists.

In TBaseWebPageModuleFactory (see page 25004), PreventDestruction changes the CacheMode property to caCache.

See Also

CacheMode

180.1.4 TWebAppDataModuleFactory Class

TWebAppDataModule is the factory object for WebSnap application data modules.

Class Hierarchy

Pascal

```
TWebAppDataModuleFactory = class(TBaseWebModuleFactory);
```

C++

```
class TWebAppDataModuleFactory : public TBaseWebModuleFactory;
```

File

WebFact

Description

All WebSnap modules are registered with the WebSnap application using a factory object. The factory is responsible for creating the module and also contains flags that specify how the module should be managed by the WebSnap application. TWebAppDataModuleFactory contains creation flags. Creation flags determine whether the module should be cached for future requests or destroyed after each request and created for each new request.

TWebAppDataModuleFactory identifies the module as an application module. The application module contains the components responsible for dispatching the HTTP request. WebSnap applications register a single application module class.

See Also

TWebAppDataModule (see page 25388)

TWebAppPageModuleFactory (see page 25012)

TWebDataModuleFactory (see page 25014)

TWebRequestHandler (see page 25448)

TWebPageModuleFactory (see page 25019)

180.1.4.1 TWebAppDataModuleFactory Members

The following tables list the members exposed by TWebAppDataModuleFactory.

Protected Methods

	Name	Description
A	GetCacheMode (see page 15045)	Returns the value of the CacheMode (see page 15048) property.
V	GetCacheMode (see page 24996)	Returns the value of the CacheMode property.
A	GetComponentClass (see page 15045)	Returns the value of the ComponentClass (see page 15049) property.
V	GetComponentClass (see page 24997)	Returns the value of the ComponentClass property.
A	GetCreateMode (see page 15046)	Returns the value of the CreateMode (see page 15049) property.
V	GetCreateMode (see page 24997)	Returns the value of the CreateMode property.
V	GetIsAppModule (see page 25011)	Returns the value of the IsAppModule property.
A	GetModuleName (see page 15047)	Returns the value of the ModuleName (see page 15050) property.
V	GetModuleName (see page 24998)	Returns the value of the ModuleName property.
V	GetWebPageInfo (see page 15047)	Returns the value of the WebPageInfo (see page 15050) property.

Public Methods

	Name	Description
	Create (see page 25011)	Creates and initializes an instance of TWebAppDataModuleFactory (see page 25009).
A	GetModule (see page 15046)	Creates an instance of the associated Web module.
V	GetModule (see page 24998)	Creates an instance of the associated Web module.
A	PreventDestruction (see page 15048)	Prevents the WebSnap application from freeing the Web module instance.
V	PreventDestruction (see page 24998)	Prevents the WebSnap application from freeing the Web module instance.

Public Properties

	Name	Description
R	CacheMode (see page 15048)	Indicates whether the factory can cache Web modules for reuse.

R	ComponentClass (see page 15049)	Specifies the class type of the Web module that this factory creates.
R	CreateMode (see page 15049)	Indicates when the factory creates instances of its Web module.
R	IsAppModule (see page 15050)	Indicates whether the corresponding Web module is the main Web application module.
R	ModuleName (see page 15050)	Specifies the name of the Web module class that this factory instantiates.
R	WebPageInfo (see page 15050)	Describes the attributes of the Web page that the corresponding Web module generates, if any.

180.1.4.2 TWebAppDataModuleFactory.Create Constructor

Creates and initializes an instance of TWebAppDataModuleFactory (see page 25009).

Pascal

```
constructor Create(AComponentClass: TComponentClass; ACacheMode: TWebModuleCacheMode =
caCache);
```

C++

```
__fastcall TWebAppDataModuleFactory(TComponentClass AComponentClass, TWebModuleCacheMode
ACacheMode = caCache);
```

Description

Call Create to instantiate a Web application module factory. Typically, the WebSnap application wizard generates the call to the constructor in the initialization section of the Web application module's unit.

AComponentClass is the class of the Web data module that this factory object creates. It becomes the value of the ComponentClass property.

ACacheMode indicates whether the application can cache Web application module instances for reuse with subsequent request messages. This becomes the value of the CacheMode property.

See Also

CacheMode

ComponentClass

180.1.4.3 TWebAppDataModuleFactory Methods

The methods of the TWebAppDataModuleFactory class are listed here.

Protected Methods

	Name	Description
V	GetIsAppModule (see page 25011)	Returns the value of the IsAppModule property.

180.1.4.3.1 TWebAppDataModuleFactory.GetIsAppModule Method

Returns the value of the IsAppModule property.

Pascal

```
function GetIsAppModule: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsAppModule();
```

Description

GetIsAppModule is the protected read implementation of the IsAppModule property.

In TBaseWebModuleFactory, GetIsAppModule returns false. Descendant classes that generate Web application modules (such as TWebAppDataModuleFactory) override this method to return true.

See Also

IsAppModule

TWebAppPageModuleFactory (see page 25012)

180.1.5 TWebAppPageModuleFactory Class

TWebAppPageModuleFactory is the factory object for application Web page modules in a WebSnap application.

Class Hierarchy

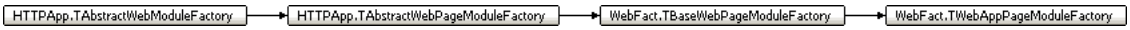

Pascal

```
TWebAppPageModuleFactory = class(TBaseWebPageModuleFactory);
```

C++

```
class TWebAppPageModuleFactory : public TBaseWebPageModuleFactory;
```

File

WebFact

Description

All WebSnap modules are registered with the WebSnap application using a factory object. The factory object is responsible for creating the module and also contains flags that determine whether the module should be cached for future request or destroyed after each request and created for each new request.

Page module factories also contain information about the page such as the page name, description, and access rights. This information is available through the factory so that it can be retrieved without creating the module.

TWebAppPageModuleFactory identifies the module as an application module. The application module contains the components responsible for dispatching the HTTP request. WebSnap applications register a single application module class.

See Also

TWebAppPageModule (see page 25394)

TWebAppDataModuleFactory (see page 25009)

TWebPageModuleFactory (see page 25019)

TWebRequestHandler (see page 25448)

TWebPageInfo (see page 25016)

TWebDataModuleFactory (see page 25014)

180.1.5.1 TWebAppPageModuleFactory Members

The following tables list the members exposed by TWebAppPageModuleFactory.

Protected Methods

	Name	Description
A	GetCacheMode (see page 15045)	Returns the value of the CacheMode (see page 15048) property.
V	GetCacheMode (see page 25007)	Returns the value of the CacheMode property.
A	GetComponentClass (see page 15045)	Returns the value of the ComponentClass (see page 15049) property.
V	GetComponentClass (see page 25007)	Returns the value of the ComponentClass property.
A	GetCreateMode (see page 15046)	Returns the value of the CreateMode (see page 15049) property.
V	GetCreateMode (see page 25007)	Returns the value of the CreateMode property.
V	GetIsAppModule (see page 25014)	Returns the value of the IsAppModule property.
A	GetModuleName (see page 15047)	Returns the value of the ModuleName (see page 15050) property.
V	GetModuleName (see page 25009)	Returns the value of the ModuleName property.
V	GetWebPageInfo (see page 15047)	Returns the value of the WebPageInfo (see page 15050) property.
V	GetWebPageInfo (see page 15072)	Returns the value of the WebPageInfo (see page 15050) property.

Public Methods

	Name	Description
	Create (see page 25013)	Creates and initializes an instance of TWebAppPageModuleFactory (see page 25012).
V	Destroy (see page 15071)	Disposes of an object instance.
A	GetModule (see page 15046)	Creates an instance of the associated Web module.
V	GetModule (see page 25008)	Creates an instance of the associated Web module.
A	PreventDestruction (see page 15048)	Prevents the WebSnap application from freeing the Web module instance.
V	PreventDestruction (see page 25009)	Prevents the WebSnap application from freeing the Web module instance.

Public Properties

	Name	Description
R	CacheMode (see page 15048)	Indicates whether the factory can cache Web modules for reuse.
R	ComponentClass (see page 15049)	Specifies the class type of the Web module that this factory creates.
R	CreateMode (see page 15049)	Indicates when the factory creates instances of its Web module.
R	IsAppModule (see page 15050)	Indicates whether the corresponding Web module is the main Web application module.
R	ModuleName (see page 15050)	Specifies the name of the Web module class that this factory instantiates.
R	WebPageInfo (see page 15050)	Describes the attributes of the Web page that the corresponding Web module generates, if any.

180.1.5.2 TWebAppPageModuleFactory.Create Constructor

Creates and initializes an instance of TWebAppPageModuleFactory (see page 25012).

Pascal

```
constructor Create(AComponentClass: TComponentClass; AWebPageInfo: TAbstractWebPageInfo;
ACacheMode: TWebModuleCacheMode = caCache);
```

C++

```
__fastcall TWebAppPageModuleFactory(TComponentClass AComponentClass, TAbstractWebPageInfo
AWebPageInfo, TWebModuleCacheMode ACacheMode = caCache);
```

Description

Call Create to instantiate a Web page module factory. Typically, the WebSnap application wizard generates the call to the constructor in the initialization section of the Web application page module's unit.

AComponentClass is the class of the Web application page module that this factory object creates.

AWebPageInfo describes the attributes of the Web application page module, such as the page name, page title, access strings, and so on. This becomes the value of the WebPageInfo property.

ACacheMode indicates whether the factory can cache Web application page module instances for reuse with subsequent request messages. This becomes the value of the CacheMode property.

See Also

CacheMode

ComponentClass

WebPageInfo

180.1.5.3 TWebAppPageModuleFactory Methods

The methods of the TWebAppPageModuleFactory class are listed here.

Protected Methods

	Name	Description
	GetIsAppModule (see page 25014)	Returns the value of the IsAppModule property.

180.1.5.3.1 TWebAppPageModuleFactory.GetIsAppModule Method

Returns the value of the IsAppModule property.

Pascal

```
function GetIsAppModule: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsAppModule();
```

Description

GetIsAppModule is the protected read implementation of the IsAppModule property.

In TBaseWebPageModuleFactory, GetIsAppModule returns false. Descendant classes that generate Web application modules (such as TWebAppPageModuleFactory) override this method to return true.

See Also

IsAppModule

TWebAppPageModuleFactory (see page 25012)

180.1.6 TWebDataModuleFactory Class

TWebDataModuleFactory is the factory class for WebSnap data modules.

Class Hierarchy

Pascal

```
TWebDataModuleFactory = class(TBaseWebModuleFactory);
```

C++

```
class TWebDataModuleFactory : public TBaseWebModuleFactory;
```

File

WebFact

Description

All WebSnap modules are registered with the WebSnap application using a factory object. (In Delphi, the factory object is typically created in the initialization section of the module source file; in C++, it is added in a #pragma startup.)The factory is responsible for creating the module and also contains flags that specify how the module should be managed by the WebSnap application. TWebDataModuleFactory includes caching flags and creation flags. Caching flags determine whether the module should be cached for future request, or destroyed after each request and created for each new request. Creation flags determine whether the module should be automatically created for every request or created on demand.

The WebSnap application may register multiple Web data module classes. Each class is registered with an instance of TWebDataModuleFactory.

See Also

TWebDataModule (see page 25405)

TWebAppDataModuleFactory (see page 25009)

TWebAppPageModuleFactory (see page 25012)

TWebRequestHandler (see page 25448)

TWebPageModuleFactory (see page 25019)

180.1.6.1 TWebDataModuleFactory Members

The following tables list the members exposed by TWebDataModuleFactory.

Protected Methods

	Name	Description
A	GetCacheMode (see page 15045)	Returns the value of the CacheMode (see page 15048) property.
V	GetCacheMode (see page 24996)	Returns the value of the CacheMode property.
A	GetComponentClass (see page 15045)	Returns the value of the ComponentClass (see page 15049) property.
V	GetComponentClass (see page 24997)	Returns the value of the ComponentClass property.
A	GetCreateMode (see page 15046)	Returns the value of the CreateMode (see page 15049) property.
V	GetCreateMode (see page 24997)	Returns the value of the CreateMode property.
A	GetIsAppModule (see page 15046)	Returns the value of the IsAppModule (see page 15050) property.
V	GetIsAppModule (see page 24997)	Returns the value of the IsAppModule property.
A	GetModuleName (see page 15047)	Returns the value of the ModuleName (see page 15050) property.
V	GetModuleName (see page 24998)	Returns the value of the ModuleName property.
V	GetWebPageInfo (see page 15047)	Returns the value of the WebPageInfo (see page 15050) property.

Public Methods

	Name	Description
	Create (see page 25016)	Creates and initializes an instance of TWebDataModuleFactory (see page 25014).
A	GetModule (see page 15046)	Creates an instance of the associated Web module.
V	GetModule (see page 24998)	Creates an instance of the associated Web module.

A	PreventDestruction (see page 15048)	Prevents the WebSnap application from freeing the Web module instance.
V	PreventDestruction (see page 24998)	Prevents the WebSnap application from freeing the Web module instance.

Public Properties

	Name	Description
R	CacheMode (see page 15048)	Indicates whether the factory can cache Web modules for reuse.
R	ComponentClass (see page 15049)	Specifies the class type of the Web module that this factory creates.
R	CreateMode (see page 15049)	Indicates when the factory creates instances of its Web module.
R	IsAppModule (see page 15050)	Indicates whether the corresponding Web module is the main Web application module.
R	ModuleName (see page 15050)	Specifies the name of the Web module class that this factory instantiates.
R	WebPageInfo (see page 15050)	Describes the attributes of the Web page that the corresponding Web module generates, if any.

180.1.6.2 TWebDataModuleFactory.Create Constructor

Creates and initializes an instance of TWebDataModuleFactory (see page 25014).

Pascal

```
constructor Create(AComponentClass: TComponentClass; ACreateMode: TWebModuleCreateMode =
crOnDemand; ACacheMode: TWebModuleCacheMode = caCache);
```

C++

```
__fastcall TWebDataModuleFactory(TComponentClass AComponentClass, TWebModuleCreateMode
ACreateMode = crOnDemand, TWebModuleCacheMode ACacheMode = caCache);
```

Description

Call Create to instantiate a Web data module factory. Typically, the Web module wizard generates the call to the constructor in the initialization section of the Web data module's unit.

AComponentClass is the class of the Web data module that this factory object creates.

ACreateMode describes whether the factory should create instances of the Web data module for every request message, or only when the application explicitly requests the Web data module. This becomes the value of the CreateMode property.

ACacheMode indicates whether the application can cache Web data module instances for reuse with subsequent request messages. This becomes the value of the CacheMode property.

See Also

CreateMode

CacheMode

ComponentClass

180.1.7 TWebPageInfo Class

TWebPageInfo describes a WebSnap page.

Class Hierarchy

Pascal

```
TWebPageInfo = class(TBaseWebPageInfo);
```

C++

```
class TWebPageInfo : public TBaseWebPageInfo;
```

File

WebFact

Description

The TWebPageInfo module stores information relating to a web page, such as the template filename, page name, title, description, and access rights. This information is part of the web page module factory. This information is stored as part of the factory rather than as part of the module so that it can be accessed without creating the web page module.

An instance of a TWebPageInfo is supplied to the constructor of the factory object that creates instances of the Web page module. Typically, the code to create this object and supply it to the Web page factory's constructor is generated by a Web page module wizard (in Delphi, the generated code appears in the initialization section of the generated unit for the Web page module; in C++, it appears in a #pragma startup). You can change the generated code to alter the properties of the Web page information object. For example, you may want to add a ViewAccess string to allow your application to limit access to the Web page.

In a WebSnap application, you can access the Web page information object of any Web page module by calling the FindPageInfo method of the application's global Web context.

See Also

TWebPageModuleFactory (see page 25019)

TWebAppPageModuleFactory (see page 25012)

TWebModuleFactoryList (see page 25436)

TAbstractWebContext (see page 24644)

WebContext

180.1.7.1 TWebPageInfo Members

The following tables list the members exposed by TWebPageInfo.

Protected Methods

	Name	Description
	GetFactory (see page 15062)	Returns the value of the Factory (see page 15066) property.
V	GetIsLoginRequired (see page 15062)	Returns the value of the IsLoginRequired (see page 15066) property.
V	GetIsLoginRequired (see page 25001)	Returns the value of the IsLoginRequired property.
V	GetIsPublished (see page 15062)	Returns the value of the IsPublished (see page 15067) property.
V	GetIsPublished (see page 25002)	Returns the value of the IsPublished property.
V	GetPageDescription (see page 15063)	Returns the value of the PageDescription (see page 15067) property.
V	GetPageDescription (see page 25002)	Returns the value of the PageDescription property.
A	GetPageFile (see page 15063)	Returns the value of the PageFile (see page 15068) property.
V	GetPageFile (see page 25002)	Returns the value of the PageFile property.
V	GetPageHREF (see page 15064)	Returns the value of the PageHREF (see page 15068) property.
V	GetPageHREF (see page 25003)	Returns the value of the PageHREF property.
A	GetPageName (see page 15064)	Returns the value of the PageName (see page 15068) property.
V	GetPageName (see page 25003)	Returns the value of the PageName property.
V	GetPageTitle (see page 15064)	Returns the value of the PageTitle (see page 15069) property.
V	GetPageTitle (see page 25003)	Returns the value of the PageTitle property.
V	GetViewAccess (see page 15065)	Returns the value of the ViewAccess (see page 15069) property.
V	GetViewAccess (see page 25004)	Returns the value of the ViewAccess property.

	SetFactory (see page 15065)	Sets the value of the Factory (see page 15066) property.
	SetFactory (see page 25004)	Connects a specified Web page module factory with the Web page information object.

Public Methods

	Name	Description
	Create (see page 25018)	Creates and initializes a TWebPageInfo (see page 25016) object.

Public Properties

	Name	Description
	Factory (see page 15066)	Specifies the factory object that creates instances of the Web page module this object describes.
R	IsLoginRequired (see page 15066)	Indicates whether callers must log in before accessing the associated Web page.
R	IsPublished (see page 15067)	Indicates whether the WebSnap application automatically displays the associated page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 15068).
R	PageDescription (see page 15067)	Describes the Web page.
R	PageFile (see page 15068)	Identifies an associated file that the Web page module uses to generate its Web page.
R	PageHREF (see page 15068)	Identifies the HREF to use for references to the associated Web page that appear in hypertext links.
R	PageName (see page 15068)	Denotes the logical name for the Web page that the associated module generates
R	PageTitle (see page 15069)	Indicates the title that appears on the associated Web page.
R	ViewAccess (see page 15069)	Indicates the view access string that controls access to the associated Web page.

180.1.7.2 TWebPageInfo.Create Constructor

Creates and initializes a TWebPageInfo (see page 25016) object.

Pascal

```
constructor Create(AAccess: TWebPageAccess = [wpPublished]; const APageFile: string = '.html';
const APageName: string = ''; const ACaption: string = ''; const ADescription: string = '';
const AViewAccess: string = '');
```

C++

```
__fastcall TWebPageInfo(TWebPageAccess AAccess = [wpPublished], const AnsiString APageFile =
'.html', const AnsiString APageName = '', const AnsiString ACaption = '', const AnsiString
ADescription = '', const AnsiString AViewAccess = '');
```

Description

Typically, you do not need create a call to the Web page information object constructor. The Web page module wizard generates code to make this call and pass the resulting object instance to the constructor of a Web page module factory. You may, however, want to edit the generated call to the constructor, changing some of the parameter values that the wizard supplies.

AAccess specifies access information for the associated Web page.

APageFile specifies the name (or extension) of the file that the associated Web page module uses to generate its content. It becomes the value of the PageFile property.

APageName is the logical name of the associated Web page. It becomes the value of the PageName property.

ACaption is the title that appears on the associated Web page. It becomes the value of the PageTitle property.

ADescription is a brief description of the associated Web page. It becomes the value of the PageDescription property.

AViewAccess is the view access string that represents a set of access rights needed to work with the associated Web page. It becomes the value of the ViewAccess property.

See Also

pagefile

pagename

pagetitle

PageDescription

viewaccess

IsPublished

IsLoginRequired

180.1.8 TWebPageModuleFactory Class

TWebPageModuleFactory is the factory object for Web page modules in a WebSnap application.

Class Hierarchy

Pascal

```
TWebPageModuleFactory = class(TBaseWebPageModuleFactory);
```

C++

```
class TWebPageModuleFactory : public TBaseWebPageModuleFactory;
```

File

WebFact

Description

All WebSnap modules are registered with the WebSnap application using a factory object. The factory object is responsible for creating the module and also contains flags that specify how the WebSnap application should manage the creation, destruction, and use of the Web module. TWebPageModuleFactory also contains flags that determine whether the module should be cached for future request or destroyed after each request and created for each new request.

Note: In the Delphi language, the factory object is created in the initialization section of the module source file.

Page module factories also contain information about the page such as the page name, description, and access rights. This information is available through the factory so that it can be retrieved without creating the module.

WebSnap applications can contain multiple Web page module factories (hence multiple Web page modules), each of which creates the Web page module for a different Web page that the application can create.

See Also

TWebPageModule (see page 25414)

TWebAppDataModuleFactory (see page 25009)

TWebAppPageModuleFactory (see page 25012)

TWebRequestHandler (see page 25448)

TWebPageInfo (see page 25016)

TWebDataModuleFactory (see page 25014)

180.1.8.1 TWebPageModuleFactory Members

The following tables list the members exposed by TWebPageModuleFactory.

Protected Methods

	Name	Description
	GetCacheMode (see page 15045)	Returns the value of the CacheMode (see page 15048) property.
	GetCacheMode (see page 25007)	Returns the value of the CacheMode property.
	GetComponentClass (see page 15045)	Returns the value of the ComponentClass (see page 15049) property.
	GetComponentClass (see page 25007)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 15046)	Returns the value of the CreateMode (see page 15049) property.
	GetCreateMode (see page 25007)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 15046)	Returns the value of the IsAppModule (see page 15050) property.
	GetIsAppModule (see page 25008)	Returns the value of the IsAppModule property.
	GetModuleName (see page 15047)	Returns the value of the ModuleName (see page 15050) property.
	GetModuleName (see page 25009)	Returns the value of the ModuleName property.
	GetWebPageInfo (see page 15047)	Returns the value of the WebPageInfo (see page 15050) property.
	GetWebPageInfo (see page 15072)	Returns the value of the WebPageInfo (see page 15050) property.

Public Methods

	Name	Description
	Create (see page 25020)	Creates and initializes an instance of TWebPageModuleFactory (see page 25019).
	Destroy (see page 15071)	Disposes of an object instance.
	GetModule (see page 15046)	Creates an instance of the associated Web module.
	GetModule (see page 25008)	Creates an instance of the associated Web module.
	PreventDestruction (see page 15048)	Prevents the WebSnap application from freeing the Web module instance.
	PreventDestruction (see page 25009)	Prevents the WebSnap application from freeing the Web module instance.

Public Properties

	Name	Description
	CacheMode (see page 15048)	Indicates whether the factory can cache Web modules for reuse.
	ComponentClass (see page 15049)	Specifies the class type of the Web module that this factory creates.
	CreateMode (see page 15049)	Indicates when the factory creates instances of its Web module.
	IsAppModule (see page 15050)	Indicates whether the corresponding Web module is the main Web application module.
	ModuleName (see page 15050)	Specifies the name of the Web module class that this factory instantiates.
	WebPageInfo (see page 15050)	Describes the attributes of the Web page that the corresponding Web module generates, if any.

180.1.8.2 TWebPageModuleFactory.Create Constructor

Creates and initializes an instance of TWebPageModuleFactory (see page 25019).

Pascal

```
constructor Create(AComponentClass: TComponentClass; AWebPageInfo: TAbstractWebPageInfo;
ACreateMode: TWebModuleCreateMode = crOnDemand; ACacheMode: TWebModuleCacheMode = caCache);
```

C++

```
__fastcall TWebPageModuleFactory(TComponentClass AComponentClass, TAbstractWebPageInfo
AWebPageInfo, TWebModuleCreateMode ACreateMode = crOnDemand, TWebModuleCacheMode ACacheMode =
caCache);
```

Description

Call Create to instantiate a Web page module factory. Typically, the Web page module wizard generates the call to the constructor in the Web page module's unit. In Delphi, the constructor call is placed in the initialization section of the unit. In C++, the call is placed in a #pragma startup.

AComponentClass is the class of the Web page module that this factory object creates.

AWebPageInfo describes the attributes of the Web page module, such as the page name, page title, access strings, and so on. This becomes the value of the WebPageInfo property.

ACreateMode describes whether the factory should create instances of the Web page module for every request message, or only when the application explicitly requests the Web page module. This becomes the value of the CreateMode property.

ACacheMode indicates whether the application can cache Web page module instances for reuse with subsequent request messages. This becomes the value of the CacheMode property.

See Also

CreateMode

CacheMode

ComponentClass

WebPageInfo

180.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TWebPageAccessFlags (see page 25022)	This is record WebFact.TWebPageAccessFlags.

180.2.1 WebFact.TWebPageAccessFlags Enumeration

Pascal

```
TWebPageAccessFlags = (
  wpPublished,
  wpLoginRequired
);
```

C++

```
enum TWebPageAccessFlags {
  wpPublished,
  wpLoginRequired
};
```

File

WebFact

Description

This is record WebFact.TWebPageAccessFlags.

180.3 Types

The following table lists types in this documentation.

Types

Name	Description
TWebPageAccess (see page 25023)	This is type WebFact.TWebPageAccess.

180.3.1 WebFact.TWebPageAccess Type

Pascal

```
TWebPageAccess = set of TWebPageAccessFlags;
```

C++

```
TWebPageAccessFlags TWebPageAccess;
```

File

WebFact

Description

This is type WebFact.TWebPageAccess.

181 WebForm Namespace

This is namespace WebForm.

Topics

Name	Description
Classes (see page 25028)	The following table lists classes in this documentation.
Interfaces (see page 25303)	The following table lists interfaces in this documentation.
Functions (see page 25312)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 25315)	The following table lists structs, records, enums in this documentation.
Types (see page 25321)	The following table lists types in this documentation.
Constants (see page 25323)	The following table lists constants in this documentation.

Classes

	Name	Description
	TAbstractControlRequirements (see page 25029)	TAbstractControlRequirements lists the variables and functions that the controls in an adapter form require to generate their HTML.
	TAbstractFormRequirements (see page 25032)	This is class WebForm.TAbstractFormRequirements.
	TAdapterActionButton (see page 25034)	TAdapterActionButton generates HTML controls for executing adapter actions.
	TAdapterActionsListHelper (see page 25046)	TAdapterActionsListHelper assists a display component that represents a collection of adapter action buttons to work with the design-time editors.
	TAdapterCommandColumn (see page 25052)	TAdapterCommandColumn generates a column of action controls in an adapter grid.
	TAdapterCommandGroup (see page 25062)	TAdapterCommandGroup generates a group of buttons or other HTML elements that executes adapter actions.
	TAdapterControlGroup (see page 25070)	TAdapterControlGroup is a container for adapter display field components such as TAdapterDisplayField (see page 25109) or TAdapterDisplayColumn (see page 25099).
	TAdapterDataDisplay (see page 25086)	TAdapterDataDisplay is the base class for components that display adapter fields as form fields.
	TAdapterDisplayColumn (see page 25099)	TAdapterDisplayColumn displays an Adapter field in a grid.
	TAdapterDisplayField (see page 25109)	TAdapterDisplayField displays an Adapter field as a form field.
	TAdapterDisplayReferenceGroup (see page 25122)	TAdapterDisplayReferenceGroup is the base class for components that generate HTML controls that refer to a control which displays data from an adapter.
	TAdapterEditColumn (see page 25132)	TAdapterEditColumn displays an adapter field as a form field in a grid.
	TAdapterErrorList (see page 25145)	TAdapterErrorList is the error list display component.
	TAdapterFieldGroup (see page 25150)	TAdapterFieldGroup is a container for Adapter (see page 25156) display field components such as TAdapterDisplayField (see page 25109). TAdapterFieldGroup labels and fields in an HTML form.
	TAdapterForm (see page 25159)	TAdapterForm generates an HTML form displaying Adapter fields.

	TAdapterFormLayout (see page 25166)	This is class WebForm.TAdapterFormLayout.
	TAdapterGrid (see page 25175)	TAdapterGrid is a container for Adapter (see page 25182) display field components such as TAdapterDisplayColumn (see page 25099). TAdapterGrid generates an HTML table.
	TControlRequirements (see page 25185)	TControlRequirements lists the variables and functions that the controls in an adapter form require to generate their HTML.
	TCustomAdapterActionButton (see page 25189)	TCustomAdapterActionButton is the base class for Web items that generate HTML controls for executing adapter actions.
	TCustomAdapterCommandColumn (see page 25205)	TCustomAdapterCommandColumn is the base class for components that generate a column of action controls in an adapter grid.
	TCustomAdapterCommandGroup (see page 25220)	TCustomAdapterCommandGroup is the base class for TAdapterCommandGroup (see page 25062).
	TCustomAdapterDisplayColumn (see page 25230)	TCustomAdapterDisplayColumn is the base class for the grid display of adapter field data.
	TCustomAdapterDisplayField (see page 25240)	TCustomAdapterDisplayField is the base class for the free-form display and editing of adapter field data.
	TCustomAdapterEditColumn (see page 25252)	TCustomAdapterEditColumn is the base class for components that display adapter fields as form fields in grids.
	TCustomAdapterErrorList (see page 25259)	TCustomAdapterErrorList is the base class adapter for components that display error lists.
	TCustomAdapterFieldGroup (see page 25265)	TCustomAdapterFieldGroup manages a collection of Web Items that generate a group of HTML controls.
	TCustomAdapterForm (see page 25276)	TCustomAdapterForm is the base class for components that generate an HTML form for an adapter page producer.
	TCustomAdapterGrid (see page 25286)	TCustomAdapterGrid is the base class for Web items that generate an HTML table to display a datasetin an XML data packet or web application.
	TDesigntimeWarnings (see page 25297)	This is class WebForm.TDesigntimeWarnings.
	TGeneratedFunctions (see page 25300)	This is class WebForm.TGeneratedFunctions.

Constants

Name	Description
sDisplayText (see page 25323)	This is constant WebForm.sDisplayText.
sFieldText (see page 25323)	This is constant WebForm.sFieldText.
sFormGet (see page 25324)	This is constant WebForm.sFormGet.
sFormPost (see page 25324)	This is constant WebForm.sFormPost.
sMultipartForm (see page 25324)	This is constant WebForm.sMultipartForm.

Functions

	Name	Description
	CalcDisplayFieldHTMLElementType (see page 25312)	This is function WebForm.CalcDisplayFieldHTMLElementType.
	CalcDisplayFieldViewMode (see page 25312)	This is function WebForm.CalcDisplayFieldViewMode.
	FindVariablesContainer (see page 25313)	This is function WebForm.FindVariablesContainer.
	FindVariablesContainerInParent (see page 25313)	This is function WebForm.FindVariablesContainerInParent.
	GetAdapterModeOfAdapterInParent (see page 25313)	This is function WebForm.GetAdapterModeOfAdapterInParent.
	MakeAdapterHTMLName (see page 25314)	This is function WebForm.MakeAdapterHTMLName.
	MakeAdapterVariableName (see page 25314)	This is function WebForm.MakeAdapterVariableName.

Interfaces

	Name	Description
	IAdapterDisplay (see page 25303)	This is class WebForm.IAdapterDisplay.
	IGetAdapterFormLayout (see page 25303)	This is class WebForm.IGetAdapterFormLayout.
	IGetAdapterModeOfDisplay (see page 25304)	This is class WebForm.IGetAdapterModeOfDisplay.
	IGetAdapterOfDisplay (see page 25305)	This is class WebForm.IGetAdapterOfDisplay.
	IGetAdapterVariableName (see page 25306)	This is class WebForm.IGetAdapterVariableName.
	IGetControlRequirements (see page 25308)	This is class WebForm.IGetControlRequirements.
	IGetFormRequirements (see page 25309)	This is class WebForm.IGetFormRequirements.
	IGetVariablesContainerOfDisplay (see page 25310)	This is class WebForm.IGetVariablesContainerOfDisplay.

Structs, Records, Enums

	Name	Description
	TActionButtonHideOption (see page 25315)	TActionButtonHideOptions (see page 25321) indicates when a control to represent an adapter action should not be generated.
	TColumnHTMLElementType (see page 25316)	This is record WebForm.TColumnHTMLElementType.
	TCommandHTMLElementType (see page 25316)	TCommandHTMLElementType identifies an HTML control type that can be used execute an adapter action.
	TDisplayFieldHideOption (see page 25317)	TDisplayFieldHideOptions (see page 25321) indicates when a control to represent a dataset field should not be generated.
	TDisplayFieldHTMLElementType (see page 25318)	TDisplayFieldHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.
	TDisplayFieldViewMode (see page 25318)	TDisplayFieldViewMode indicates the mode in which an adapter field's value is displayed.
	TInputFieldHTMLElementType (see page 25319)	TInputFieldHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.
	TLayoutState (see page 25320)	This is record WebForm.TLayoutState.

Types

Name	Description
TActionButtonHideOptions (see page 25321)	This is type WebForm.TActionButtonHideOptions.
TAdapterDataDisplayClass (see page 25321)	This is type WebForm.TAdapterDataDisplayClass.
TDisplayFieldHideOptions (see page 25321)	This is type WebForm.TDisplayFieldHideOptions.

181.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TAbstractControlRequirements (see page 25029)	TAbstractControlRequirements lists the variables and functions that the controls in an adapter form require to generate their HTML.
	TAbstractFormRequirements (see page 25032)	This is class WebForm.TAbstractFormRequirements.
	TAdapterActionButton (see page 25034)	TAdapterActionButton generates HTML controls for executing adapter actions.
	TAdapterActionsListHelper (see page 25046)	TAdapterActionsListHelper assists a display component that represents a collection of adapter action buttons to work with the design-time editors.
	TAdapterCommandColumn (see page 25052)	TAdapterCommandColumn generates a column of action controls in an adapter grid.
	TAdapterCommandGroup (see page 25062)	TAdapterCommandGroup generates a group of buttons or other HTML elements that executes adapter actions.
	TAdapterControlGroup (see page 25070)	TAdapterControlGroup is a container for adapter display field components such as TAdapterDisplayField (see page 25109) or TAdapterDisplayColumn (see page 25099).
	TAdapterDataDisplay (see page 25086)	TAdapterDataDisplay is the base class for components that display adapter fields as form fields.
	TAdapterDisplayColumn (see page 25099)	TAdapterDisplayColumn displays an Adapter field in a grid.
	TAdapterDisplayField (see page 25109)	TAdapterDisplayField displays an Adapter field as a form field.
	TAdapterDisplayReferenceGroup (see page 25122)	TAdapterDisplayReferenceGroup is the base class for components that generate HTML controls that refer to a control which displays data from an adapter.
	TAdapterEditColumn (see page 25132)	TAdapterEditColumn displays an adapter field as a form field in a grid.
	TAdapterErrorList (see page 25145)	TAdapterErrorList is the error list display component.
	TAdapterFieldGroup (see page 25150)	TAdapterFieldGroup is a container for Adapter (see page 25156) display field components such as TAdapterDisplayField (see page 25109). TAdapterFieldGroup labels and fields in an HTML form.
	TAdapterForm (see page 25159)	TAdapterForm generates an HTML form displaying Adapter fields.
	TAdapterFormLayout (see page 25166)	This is class WebForm.TAdapterFormLayout.
	TAdapterGrid (see page 25175)	TAdapterGrid is a container for Adapter (see page 25182) display field components such as TAdapterDisplayColumn (see page 25099). TAdapterGrid generates an HTML table.
	TControlRequirements (see page 25185)	TControlRequirements lists the variables and functions that the controls in an adapter form require to generate their HTML.
	TCustomAdapterActionButton (see page 25189)	TCustomAdapterActionButton is the base class for Web items that generate HTML controls for executing adapter actions.
	TCustomAdapterCommandColumn (see page 25205)	TCustomAdapterCommandColumn is the base class for components that generate a column of action controls in an adapter grid.
	TCustomAdapterCommandGroup (see page 25220)	TCustomAdapterCommandGroup is the base class for TAdapterCommandGroup (see page 25062).
	TCustomAdapterDisplayColumn (see page 25230)	TCustomAdapterDisplayColumn is the base class for the grid display of adapter field data.
	TCustomAdapterDisplayField (see page 25240)	TCustomAdapterDisplayField is the base class for the free-form display and editing of adapter field data.
	TCustomAdapterEditColumn (see page 25252)	TCustomAdapterEditColumn is the base class for components that display adapter fields as form fields in grids.
	TCustomAdapterErrorList (see page 25259)	TCustomAdapterErrorList is the base class adapter for components that display error lists.
	TCustomAdapterFieldGroup (see page 25265)	TCustomAdapterFieldGroup manages a collection of Web Items that generate a group of HTML controls.
	TCustomAdapterForm (see page 25276)	TCustomAdapterForm is the base class for components that generate an HTML form for an adapter page producer.
	TCustomAdapterGrid (see page 25286)	TCustomAdapterGrid is the base class for Web items that generate an HTML table to display a datasetin an XML data packet or web application.

	TDesigntimeWarnings (see page 25297)	This is class WebForm.TDesigntimeWarnings.
	TGeneratedFunctions (see page 25300)	This is class WebForm.TGeneratedFunctions.

181.1.1 TAbstractControlRequirements Class

TAbstractControlRequirements lists the variables and functions that the controls in an adapter form require to generate their HTML.

Class Hierarchy

WebForm.TAbstractControlRequirements

Pascal

```
TAbstractControlRequirements = class;
```

C++

```
class TAbstractControlRequirements;
```

File

WebForm

Description

Components that generate HTML controls to represent the fields of an adapter (TAdapterControlGroup (see page 25070) and TAdapterDataDisplay (see page 25086) descendants) use descendants of TAbstractControlRequirements. The descendants of TAbstractControlRequirements store the variable and function declarations the adapter form must include before the control group can assemble HTML controls for the fields of the adapter it represents.

TAbstractControlRequirements introduces methods to

Store a specified function or variable (along with its value).

Query whether the function or variable declaration has already been added to the list of requirements.

Store an indication of whether the control requires that it be contained in a multipart form.

TAbstractControlRequirements is abstract or, in C++ terminology, pure virtual. It does not implement any of the methods it declares. WebSnap applications use TAbstractControlRequirements descendants such as TControlRequirements (see page 25185), which provide implementations for these methods.

See Also

TAdapterControlGroup (see page 25070)

TAdapterDataDisplay (see page 25086)

TAdapterForm (see page 25159)

TControlRequirements (see page 25185)

181.1.1.1 TAbstractControlRequirements Members

The following tables list the members exposed by TAbstractControlRequirements.

Public Methods

	Name	Description
	DefineFunction (see page 25030)	Adds the declaration of a function to those that this control requirements object stores.
	DefineVariable (see page 25030)	Adds the declaration of a variable to those that this control requirements object stores.
	IsFunctionDefined (see page 25031)	Indicates whether a function with the specified name has already been added to the list of control requirements.
	IsVariableDefined (see page 25031)	Indicates whether a variable with the specified name has already been added to the list of control requirements.
	RequiresMultipartForm (see page 25032)	Stores the fact that the generated HTML form should be multipart.

181.1.1.2 TAbstractControlRequirements Methods

The methods of the TAbstractControlRequirements class are listed here.

Public Methods

	Name	Description
	DefineFunction (see page 25030)	Adds the declaration of a function to those that this control requirements object stores.
	DefineVariable (see page 25030)	Adds the declaration of a variable to those that this control requirements object stores.
	IsFunctionDefined (see page 25031)	Indicates whether a function with the specified name has already been added to the list of control requirements.
	IsVariableDefined (see page 25031)	Indicates whether a variable with the specified name has already been added to the list of control requirements.
	RequiresMultipartForm (see page 25032)	Stores the fact that the generated HTML form should be multipart.

181.1.1.2.1 TAbstractControlRequirements.DefineFunction Method

Adds the declaration of a function to those that this control requirements object stores.

Pascal

```
procedure DefineFunction(const AName: string; const ABody: string); virtual; abstract;
```

C++

```
virtual __fastcall DefineFunction(const AnsiString AName, const AnsiString ABody) = 0;
```

Description

DefineFunction stores the declaration of a function that has the name AName, where ABody is the text of the function body.

AName is the name of the function. This name is one of a standard set that the adapter form knows to call under specific circumstances. For example, the name can be the standard name of the function to generate an HTML button.

ABody is the body of the specified function. This value is typically customized to reflect the properties of a specific control in the form.

See Also

IsFunctionDefined (see page 25031)

DefineVariable (see page 25030)

181.1.1.2.2 TAbstractControlRequirements.DefineVariable Method

Adds the declaration of a variable to those that this control requirements object stores.

Pascal

```
procedure DefineVariable(const AName: string; const AValue: string; AAdapter: Boolean);
virtual; abstract;
```

C++

```
virtual __fastcall DefineVariable(const AnsiString AName, const AnsiString AValue, Boolean
AAdapter) = 0;
```

Description

DefineVariable stores the name and value of a variable declaration.

AName is the name of the variable.

AValue is the value to assign to the variable.

AAdapter indicates whether this variable is the one that declares the adapter object whose field a control represents or whose set of fields a control group represents.

See Also

DefineFunction (see page 25030)

IsVariableDefined (see page 25031)

Adapter (see page 25084)

181.1.1.2.3 TAbstractControlRequirements.IsFunctionDefined Method

Indicates whether a function with the specified name has already been added to the list of control requirements.

Pascal

```
function IsFunctionDefined(const AName: string): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean IsFunctionDefined(const AnsiString AName) = 0;
```

Description

IsFunctionDefined returns true if the control requirements object already lists a function with the specified name. That is, IsFunctionDefined returns true if the DefineFunction (see page 25030) method has been called to define the specified function, and false otherwise.

AName is the name of the function.

See Also

DefineFunction (see page 25030)

IsVariableDefined (see page 25031)

181.1.1.2.4 TAbstractControlRequirements.IsVariableDefined Method

Indicates whether a variable with the specified name has already been added to the list of control requirements.

Pascal

```
function IsVariableDefined(const AName: string): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean IsVariableDefined(const AnsiString AName) = 0;
```

Description

IsVariableDefined returns true if the control requirements object already lists a variable with the specified name. That is, IsVariableDefined returns true if the DefineVariable (see page 25030) method has been called to define a value for the specified variable, and false otherwise.

AName is the name of the variable.

See Also

DefineVariable (see page 25030)

IsFunctionDefined (see page 25031)

181.1.1.2.5 TAbstractControlRequirements.RequiresMultipartForm Method

Stores the fact that the generated HTML form should be multipart.

Pascal

```
procedure RequiresMultipartForm; virtual; abstract;
```

C++

```
virtual __fastcall RequiresMultipartForm() = 0;
```

Description

Web items call this method if they need to be contained in a multipart form (for example, when a field in the form is represented by a reference to another file). A multipart form's HTML declaration includes the attribute 'enctype=multipart/form-data'.

181.1.2 TAbstractFormRequirements Class

Class Hierarchy

Pascal

```
TAbstractFormRequirements = class;
```

C++

```
class TAbstractFormRequirements;
```

File

WebForm

Description

This is class WebForm.TAbstractFormRequirements.

181.1.2.1 TAbstractFormRequirements Members

The following tables list the members exposed by TAbstractFormRequirements.

Public Methods

	Name	Description
	DefineFunction (see page 25033)	This is DefineFunction, a member of class TAbstractFormRequirements.
	DefineVariable (see page 25033)	This is DefineVariable, a member of class TAbstractFormRequirements.
	IsFunctionDefined (see page 25033)	This is IsFunctionDefined, a member of class TAbstractFormRequirements.
	IsVariableDefined (see page 25034)	This is IsVariableDefined, a member of class TAbstractFormRequirements.

181.1.2.2 TAbstractFormRequirements Methods

The methods of the TAbstractFormRequirements class are listed here.

Public Methods

	Name	Description
	DefineFunction (see page 25033)	This is DefineFunction, a member of class TAbstractFormRequirements.
	DefineVariable (see page 25033)	This is DefineVariable, a member of class TAbstractFormRequirements.
	IsFunctionDefined (see page 25033)	This is IsFunctionDefined, a member of class TAbstractFormRequirements.
	IsVariableDefined (see page 25034)	This is IsVariableDefined, a member of class TAbstractFormRequirements.

181.1.2.2.1 TAbstractFormRequirements.DefineFunction Method

Pascal

```
procedure DefineFunction(const AName: string; const ABody: string); virtual; abstract;
```

C++

```
virtual __fastcall DefineFunction(const AnsiString AName, const AnsiString ABody) = 0;
```

Description

This is DefineFunction, a member of class TAbstractFormRequirements.

181.1.2.2.2 TAbstractFormRequirements.DefineVariable Method

Pascal

```
procedure DefineVariable(const AName: string; const AValue: string); virtual; abstract;
```

C++

```
virtual __fastcall DefineVariable(const AnsiString AName, const AnsiString AValue) = 0;
```

Description

This is DefineVariable, a member of class TAbstractFormRequirements.

181.1.2.2.3 TAbstractFormRequirements.IsFunctionDefined Method

Pascal

```
function IsFunctionDefined(const AName: string): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean IsFunctionDefined(const AnsiString AName) = 0;
```

Description

This is IsFunctionDefined, a member of class TAbstractFormRequirements.

181.1.2.2.4 TAbstractFormRequirements.IsVariableDefined Method

Pascal

```
function IsVariableDefined(const AName: string): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean IsVariableDefined(const AnsiString AName) = 0;
```

Description

This is IsVariableDefined, a member of class TAbstractFormRequirements.

181.1.3 TAdapterActionButton Class

TAdapterActionButton generates HTML controls for executing adapter actions.

Class Hierarchy

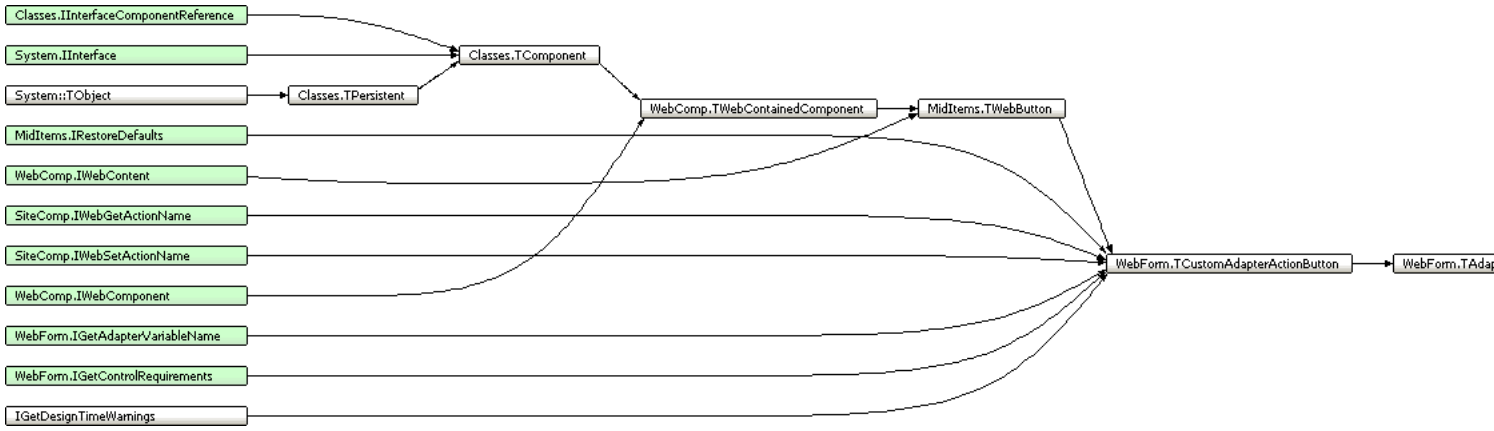

Pascal

```
TAdapterActionButton = class(TCustomAdapterActionButton);
```

C++

```
class TAdapterActionButton : public TCustomAdapterActionButton;
```

File

WebForm

Description

Add TAdapterActionButton to the hierarchy of Web items that an adapter page producer (TAdapterPageProducer) maintains to include a control in the HTML form that executes an adapter action. When the end user clicks the control, it sends a request message that executes the adapter action. Use the ActionName (see page 25039) property to specify the adapter action that the generated control executes.

TAdapterActionButton can generate a variety of controls, including labeled buttons, images that the user can click, and hypertext links. Use the DisplayType (see page 25041) property to specify the type of control you want. Other properties customize the appearance of the control, or indicate when it is included in HTML forms.

See Also

TAdapterPageProducer (see page 19802)

TCustomAdapterAction (see page 24188)

TAdapterForm (see page 25159)

181.1.3.1 TAdapterActionButton Members

The following tables list the members exposed by TAdapterActionButton.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16868)	Returns the HTML that defines the button control.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FindAssociatedAction (see page 25195)	This is FindAssociatedAction, a member of class TCustomAdapterActionButton.
	GetActionName (see page 25196)	This is GetActionName, a member of class TCustomAdapterActionButton.
	GetAdapterModeOfAdapter (see page 25196)	This is GetAdapterModeOfAdapter, a member of class TCustomAdapterActionButton.
	GetAdapterVariableName (see page 25196)	This is GetAdapterVariableName, a member of class TCustomAdapterActionButton.
V	GetCaption (see page 16869)	Returns the value of the Caption (see page 16870) property.
V	GetCaption (see page 25196)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetControlRequirements (see page 25197)	This is GetControlRequirements, a member of class TCustomAdapterActionButton.
	GetDesigntimeWarnings (see page 25197)	This is GetDesigntimeWarnings, a member of class TCustomAdapterActionButton.
	GetHTMLFormVarName (see page 25197)	This is GetHTMLFormVarName, a member of class TCustomAdapterActionButton.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16869)	Returns the layout object that specifies how the generated control should look.
V	GetLayoutAttributes (see page 25197)	Returns the layout object that specifies how the generated control should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).

A	ImplContent (see page 16869)	Provides the underlying implementation for the Content (see page 16868) method.
V	ImplContent (see page 25198)	Provides the underlying implementation for the Content method.
V	ImplGetControlRequirements (see page 25198)	This is ImplGetControlRequirements, a member of class TCustomAdapterActionButton.
V	ImplRestoreDefaults (see page 25198)	This is ImplRestoreDefaults, a member of class TCustomAdapterActionButton.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 25199)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 25199)	This is RestoreDefaults, a member of class TCustomAdapterActionButton.
	SetActionName (see page 25199)	This is SetActionName, a member of class TCustomAdapterActionButton.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16870)	Sets the value of the Caption (see page 16870) property.
V	SetCaption (see page 25200)	Sets the value of the Caption property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 16868)	Creates and initializes an instance of TWebButton (see page 16863).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	Create (see page 25194)	Creates and initializes an instance of TCustomAdapterActionButton (see page 25189).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	Destroy (see page 25195)	Frees an instance of TCustomAdapterActionButton (see page 25189).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionName (see page 19755)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterVariableName (see page 25307)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetControlRequirements (see page 25309)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetActionName (see page 19783)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AdapterVariableName (see page 25308)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	ActionName (see page 25039)	Specifies the name of the action taken by the Web button.
	Align (see page 25040)	Specifies the horizontal alignment of the generated HTML control.
	Caption (see page 25040)	Specifies the caption that appears on the button.
	Custom (see page 25040)	Represents a tag option and its value in the generated INPUT tag.
	DisplayColumns (see page 25041)	Indicates the number of columns to use when laying out the generated HTML control.
	DisplayType (see page 25041)	Specifies the DisplayType of the generated HTML control.
	ErrorPageName (see page 25042)	Specifies the name of the page to display when an error occurs executing the action.

	HideOptions (see page 25042)	Controls when the generated HTML element appears in HTML forms.
	ImageHeight (see page 25043)	Specifies the height of the generated element if it is a clickable image.
	ImageProducer (see page 25043)	Specifies the interface that this adapter action button uses to generate images.
	ImageWidth (see page 25043)	Specifies the width of the generated element if it is a clickable image.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	PageName (see page 25044)	Specifies the name of the page to display after successfully executing the action.
	Style (see page 25044)	Adds style attributes to the generated button control.
	StyleRule (see page 25045)	Applies a style in the page producer's style sheet to the button.
	Tag (see page 2808)	Stores an integer value as part of a component.
	VAlign (see page 25045)	Specifies the vertical alignment of the generated HTML control.

181.1.3.2 TAdapterActionButton Properties

The properties of the TAdapterActionButton class are listed here.

Published Properties

	Name	Description
	ActionName (see page 25039)	Specifies the name of the action taken by the Web button.
	Align (see page 25040)	Specifies the horizontal alignment of the generated HTML control.
	Caption (see page 25040)	Specifies the caption that appears on the button.
	Custom (see page 25040)	Represents a tag option and its value in the generated INPUT tag.
	DisplayColumns (see page 25041)	Indicates the number of columns to use when laying out the generated HTML control.
	DisplayType (see page 25041)	Specifies the DisplayType of the generated HTML control.
	ErrorPageName (see page 25042)	Specifies the name of the page to display when an error occurs executing the action.
	HideOptions (see page 25042)	Controls when the generated HTML element appears in HTML forms.
	ImageHeight (see page 25043)	Specifies the height of the generated element if it is a clickable image.
	ImageProducer (see page 25043)	Specifies the interface that this adapter action button uses to generate images.
	ImageWidth (see page 25043)	Specifies the width of the generated element if it is a clickable image.
	PageName (see page 25044)	Specifies the name of the page to display after successfully executing the action.
	Style (see page 25044)	Adds style attributes to the generated button control.
	StyleRule (see page 25045)	Applies a style in the page producer's style sheet to the button.
	VAlign (see page 25045)	Specifies the vertical alignment of the generated HTML control.

181.1.3.2.1 TAdapterActionButton.ActionName Property

Specifies the name of the action taken by the Web button.

Pascal

```
property ActionName: string;
```

C++

```
__property AnsiString ActionName;__property AnsiString ActionName;
```

Description

Use ActionName to get or set the name of the adapter action that executes in response to requests sent when the user clicks this object's generated control. ActionName is the same as the ActionName property of the adapter action object that this control executes.

See Also

Caption

ActionName (see page 24198)

181.1.3.2.2 TAdapterActionButton.Align Property

Specifies the horizontal alignment of the generated HTML control.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align controls where the generated HTML control is positioned horizontally within its parent HTML element. If the adapter action button is included in the column of an adapter grid, this property has no effect unless the associated action has multiple items. It affects the position of the HTML controls for each individual of the items within the grid generated to position them into rows.

Align indicates the desired value of the Align option of the HTML tag for the generated control.

The default value of Align is haDefault, which means the web browser that reads the HTML document horizontally positions the control wherever it wishes.

See Also

VAlign (see page 25204)

DisplayColumns (see page 25201)

ItemCount (see page 24074)

181.1.3.2.3 TAdapterActionButton.Caption Property

Specifies the caption that appears on the button.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the caption that appears on the generated HTML button. The value of Caption is used as the value of a VALUE parameter on the HTML INPUT tag generated to implement the button.

See Also

Custom (see page 16871)

GetCaption (see page 16869)

SetCaption (see page 16870)

181.1.3.2.4 TAdapterActionButton.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to represent one or more options that qualify the INPUT tag generated by the Web button. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

```
ALIGN=TOP TABINDEX=2
```

See Also

Caption (see page 16870)

Style (see page 16871)

181.1.3.2.5 TAdapterActionButton.DisplayColumns Property

Indicates the number of columns to use when laying out the generated HTML control.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

An adapter action may represent multiple commands. When the TCustomAdapterActionButton object generates controls for such an action, it must handle the layout of individual items for each of the action's commands. DisplayColumns controls how many items (controls) appear in a row before TCustomAdapterActionButton starts a new row of controls.

If the corresponding adapter action does not represent multiple commands (that is, if its ItemCount property is not greater than 1), the value of DisplayColumns has no effect.

If DisplayColumns is –1, the Web browser controls the layout of the adapter action's items.

See Also

ItemCount (see page 24074)

DisplayType (see page 25202)

181.1.3.2.6 TAdapterActionButton.DisplayType Property

Specifies the DisplayType of the generated HTML control.

Pascal

```
property DisplayType: TCommandHTMLElementType;
```

C++

```
__property TCommandHTMLElementType DisplayType;
```

Description

DisplayType controls the type of HTML element that this Web item generates.

When DisplayType is ctDefault (the default), TCustomAdapterActionButton uses the element type that the associated adapter action suggests. Change DisplayType to generate a different type of control or set of controls.

When DisplayType is ctImage or ctEventImages, you must set the ImageProducer property to a component that can provide the image or images.

See Also

ImageProducer (see page 25203)

Caption

GetDisplayStyle (see page 24056)

181.1.3.2.7 TAdapterActionButton.ErrorPageName Property

Specifies the name of the page to display when an error occurs executing the action.

Pascal

```
property ErrorPageName: string;
```

C++

```
__property AnsiString ErrorPageName;
```

Description

Use ErrorPageName to indicate the Web page that should appear if the WebSnap application encounters an error when executing the adapter action in response to the user clicking the generated control. This is the name used to identify the Web page in server-side scripts, and appears as the pathinfo portion of the URL for that page if it is published.

See Also

PageName (see page 25204)

pagename

181.1.3.2.8 TAdapterActionButton.HideOptions Property

Controls when the generated HTML element appears in HTML forms.

Pascal

```
property HideOptions: TActionButtonHideOptions;
```

C++

```
__property TActionButtonHideOptions HideOptions;
```

Description

HideOptions controls when this Web item generates an HTML control for its associated action. This property is a set of flags that control when the adapter page producer suppresses the controls that this object generates. If HideOptions is an empty set, TCustomAdapterActionButton always generates a control for its associated adapter action.

See Also

GetVisible (see page 24061)

WebEnabled (see page 24071)

HasExecuteAccess (see page 24062)

181.1.3.2.9 TAdapterActionButton.ImageHeight Property

Specifies the height of the generated element if it is a clickable image.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

ImageHeight controls the height of the image that the user can click to execute the associated action. If ImageHeight is –1 (the default), the browser controls the height it uses to display the generated image.

ImageHeight is only used when the value of the DisplayType property is ctImage or ctEventImages.

See Also

ImageProducer (see page 25203)

ImageWidth (see page 25204)

DisplayType (see page 25202)

181.1.3.2.10 TAdapterActionButton.ImageProducer Property

Specifies the interface that this adapter action button uses to generate images.

Pascal

```
property ImageProducer: IActionImageProducer;
```

C++

```
__property IActionImageProducer ImageProducer;
```

Description

Set ImageProducer when the value of the DisplayType is ctImage or ctEventImages. ImageProducer is the interface of the component that supplies the image or images that appear in HTML forms as the control that this Web item generates. When the user clicks on the generated image in the HTML form, the associated action executes.

See Also

DisplayType (see page 25202)

ImageHeight (see page 25203)

ImageWidth (see page 25204)

181.1.3.2.11 TAdapterActionButton.ImageWidth Property

Specifies the width of the generated element if it is a clickable image.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

ImageWidth controls the width of the image that the user can click to execute the associated action. If ImageWidth is –1 (the default), the browser controls the width it uses to display the generated image.

ImageWidth is only used when the value of the DisplayType property is ctImage or ctEventImages.

See Also

ImageProducer (see page 25203)

ImageHeight (see page 25203)

DisplayType (see page 25202)

181.1.3.2.12 TAdapterActionButton.PageName Property

Specifies the name of the page to display after successfully executing the action.

Pascal

```
property PageName: string;
```

C++

```
__property AnsiString PageName;
```

Description

Use PageName to indicate the Web page that should appear if the WebSnap application successfully executes the adapter action in response to the user clicking the generated control. This is the name used to identify the Web page in server-side scripts, and appears as the pathinfo portion of the URL for that page if it is published.

See Also

ErrorPageName (see page 25202)

pagename

181.1.3.2.13 TAdapterActionButton.Style Property

Adds style attributes to the generated button control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to add a STYLE option that qualifies the INPUT tag generated by the Web button. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated INPUT tag as the value of a STYLE option.

See Also

Custom (see page 16871)

StyleRule (see page 16872)

181.1.3.2.14 TAdapterActionButton.StyleRule Property

Applies a style in the page producer's style sheet to the button.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated button control. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the button's Style property.

See Also

Style (see page 16871)

Styles (see page 17127)

StylesFile (see page 17128)

Styles (see page 19830)

StylesFile (see page 19831)

181.1.3.2.15 TAdapterActionButton.VAlign Property

Specifies the vertical alignment of the generated HTML control.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign controls where the generated HTML control is positioned vertically within its parent HTML element. VAlign indicates the desired value of the VAlign option of the HTML tag for the generated control. . If the adapter action button is included in the column of an adapter grid, this property has no effect unless the associated action has multiple items. It affects the position of the HTML controls for each of the individual items within the grid generated to position them into rows.

The default value of VAlign is haVDefault, which means the web browser that reads the HTML document vertically positions the control wherever it wishes.

See Also

Align (see page 25201)

DisplayColumns (see page 25201)

ItemCount (see page 24074)

181.1.4 TAdapterActionsListHelper Class

TAdapterActionsListHelper assists a display component that represents a collection of adapter action buttons to work with the design-time editors.

Class Hierarchy

Pascal

```
TAdapterActionsListHelper = class(TObject);
```

C++

```
class TAdapterActionsListHelper : public TObject;
```

File

WebForm

Description

AdapterActionsListHelper is a helper object that works with components that generate a collection of controls for executing adapter actions. Such components include TAdapterCommandColumn (see page 25052) and TAdapterCommandGroup (see page 25062). TAdapterActionsListHelper handles common tasks that allow the display component to work with the IDE. These tasks include locating child Web items for specified actions, listing the child Web items (adapter action buttons) of the display component, and specifying the text of menu commands that operate on the set of child Web items.

See Also

TAdapterCommandColumn (see page 25052)

TAdapterCommandGroup (see page 25062)

TAdapterActionButton (see page 25034)

181.1.4.1 TAdapterActionsListHelper Members

The following tables list the members exposed by TAdapterActionsListHelper.

Protected Methods

	Name	Description
	FindAction (see page 25048)	Returns the child Web item that corresponds to the specified adapter action.
	GetActionsList (see page 25049)	Fills a list with the names of all the actions for which there are child Web items.
	GetAddAllFieldsItem (see page 25049)	Returns the string that the IDE uses for the command to add all default actions to the list of child Web items.
	GetAddFieldsDlgCaption (see page 25049)	Returns the string that the IDE uses to label the dialog in which you can select child Web items to add.
	GetAddFieldsItem (see page 25050)	Returns the string that the IDE uses for the command to display the Add Commands dialog.
	GetNewFieldPrefix (see page 25050)	Returns the string that is used to start the names of all automatically-generated Web items.
	GetVisibleActions (see page 25051)	Fills a list with the names of all the actions for which there are child Web items.
	IsActionInUse (see page 25051)	Indicates whether there is a child Web item that corresponds to a specified action.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25047)	Creates and initializes an instance of TAdapterActionsListHelper (see page 25046).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	AddDefaultActions (see page 25052)	Indicates whether the display control represents all of the adapter's default actions if you do not explicitly add items to its WebFieldControls property.

181.1.4.2 TAdapterActionsListHelper.Create Constructor

Creates and initializes an instance of TAdapterActionsListHelper (see page 25046).

Pascal

```
constructor Create(AComponent: TComponent; AHelper: TWebContainerComponentHelper);
```

C++

```
__fastcall TAdapterActionsListHelper(TComponent * AComponent, TWebContainerComponentHelper
AHelper);
```

Description

Do not instantiate TAdapterActionsListHelper (see page 25046). It is created by display components that use it to handle their interactions with the IDE.

AComponent is the adapter display component that uses TAdapterActionsListHelper (see page 25046) to interact with the IDE. This component is a Web item, such as TAdapterCommandColumn (see page 25052) or TAdapterCommandGroup (see page 25062), that assembles a set of HTML controls for executing adapter actions.

AHelper is the helper object that ACompnent uses to manage its list of child Web items.

See Also

TAdapterCommandColumn (see page 25052)

TAdapterCommandGroup (see page 25062)

181.1.4.3 TAdapterActionsListHelper Methods

The methods of the TAdapterActionsListHelper class are listed here.

Protected Methods

	Name	Description
	FindAction (see page 25048)	Returns the child Web item that corresponds to the specified adapter action.
	GetActionsList (see page 25049)	Fills a list with the names of all the actions for which there are child Web items.
	GetAddAllFieldsItem (see page 25049)	Returns the string that the IDE uses for the command to add all default actions to the list of child Web items.
	GetAddFieldsDlgCaption (see page 25049)	Returns the string that the IDE uses to label the dialog in which you can select child Web items to add.
	GetAddFieldsItem (see page 25050)	Returns the string that the IDE uses for the command to display the Add Commands dialog.
	GetNewFieldPrefix (see page 25050)	Returns the string that is used to start the names of all automatically-generated Web items.
	GetVisibleActions (see page 25051)	Fills a list with the names of all the actions for which there are child Web items.
	IsActionInUse (see page 25051)	Indicates whether there is a child Web item that corresponds to a specified action.

181.1.4.3.1 TAdapterActionsListHelper.FindAction Method

Returns the child Web item that corresponds to the specified adapter action.

Pascal

```
function FindAction(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindAction(const AnsiString AName);
```

Description

FindAction searches through the list of child Web items for a component that corresponds to the specified adapter action.

AName is the name of the adapter action to locate.

FindAction returns the child Web item (typically a TAdapterActionButton (see page 25034) instance) that generates a control for the specified action. If there is no child Web item for the specified action, FindAction returns nil (Delphi) or NULL (C++).

See Also

GetActionsList (see page 25049)

GetVisibleActions (see page 25051)

IsActionInUse (see page 25051)

ActionName (see page 25200)

181.1.4.3.2 TAdapterActionsListHelper.GetActionsList Method

Fills a list with the names of all the actions for which there are child Web items.

Pascal

```
procedure GetActionsList(List: TStrings);
```

C++

```
__fastcall GetActionsList(TStrings List);
```

Description

GetActionsList searches through the variables that have been defined for child Web items that represent adapter actions. It adds the names of any actions it finds to the list, along with the class of the Web item that represents each action.

List is the list to fill with the names of adapter actions for which there are child Web items.

See Also

FindAction (see page 25048)

GetVisibleActions (see page 25051)

IsActionInUse (see page 25051)

ActionName (see page 25200)

181.1.4.3.3 TAdapterActionsListHelper.GetAddAllFieldsItem Method

Returns the string that the IDE uses for the command to add all default actions to the list of child Web items.

Pascal

```
function GetAddAllFieldsItem: string;
```

C++

```
__fastcall AnsiString GetAddAllFieldsItem();
```

Description

The IDE calls GetAddAllFieldsItem to customize the label of the command that adds all the default items to the list of child Web items. Because TAdapterActionsListHelper (see page 25046) is specialized for controls that contain adapter action buttons, it returns the string 'Add All Commands'.

See Also

GetAddFieldsDlgCaption (see page 25049)

GetAddFieldsItem (see page 25050)

GetNewFieldPrefix (see page 25050)

181.1.4.3.4 TAdapterActionsListHelper.GetAddFieldsDlgCaption Method

Returns the string that the IDE uses to label the dialog in which you can select child Web items to add.

Pascal

```
function GetAddFieldsDlgCaption: string;
```

C++

```
__fastcall AnsiString GetAddFieldsDlgCaption();
```

Description

The IDE calls GetAddFieldsDlgCaption to label the dialog that lists available child Web items. Because TAdapterActionsListHelper (see page 25046) is specialized for controls that contain adapter action buttons, it returns the string 'Add Commands'.

See Also

GetAddAllFieldsItem (see page 25049)

GetAddFieldsItem (see page 25050)

GetNewFieldPrefix (see page 25050)

181.1.4.3.5 TAdapterActionsListHelper.GetAddFieldsItem Method

Returns the string that the IDE uses for the command to display the Add Commands dialog.

Pascal

```
function GetAddFieldsItem: string;
```

C++

```
__fastcall AnsiString GetAddFieldsItem();
```

Description

The IDE calls GetAddFieldsItem to customize the label of the command that brings up a dialog where you can select child Web items to add. Because TAdapterActionsListHelper (see page 25046) is specialized for controls that contain adapter action buttons, it returns the string 'Add Commands ...'.

See Also

GetAddFieldsDlgCaption (see page 25049)

GetAddAllFieldsItem (see page 25049)

GetNewFieldPrefix (see page 25050)

181.1.4.3.6 TAdapterActionsListHelper.GetNewFieldPrefix Method

Returns the string that is used to start the names of all automatically-generated Web items.

Pascal

```
function GetNewFieldPrefix: string;
```

C++

```
__fastcall AnsiString GetNewFieldPrefix();
```

Description

When the IDE adds new child Web items, it generates names of the components it adds. Because TAdapterActionsListHelper (see page 25046) is specialized for controls that contain adapter action buttons, the names of those TAdapterActionButton (see page 25034) components are generated by prepending the string that this method returns to the name of the action the button executes.

GetNewFieldPrefix returns the string 'Cmd'.

See Also

GetAddFieldsDlgCaption (see page 25049)

GetAddAllFieldsItem (see page 25049)

GetAddFieldsItem (see page 25050)

181.1.4.3.7 TAdapterActionsListHelper.GetVisibleActions Method

Fills a list with the names of all the actions for which there are child Web items.

Pascal

```
function GetVisibleActions: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleActions();
```

Description

GetVisibleActions returns the list of child Web items that generate HTML controls for the associated component. Each element in this list is an adapter action button (TAdapterActionButton (see page 25034) object).

If you have explicitly added child Web items, the returned list contains the child Web items you added. Otherwise, if the AddDefaultActions (see page 25052) property is true, the list contains adapter action buttons for the default actions of the adapter. If AddDefaultActions (see page 25052) is false, the list is empty.

See Also

FindAction (see page 25048)

GetActionsList (see page 25049)

IsActionInUse (see page 25051)

TAdapterActionButton (see page 25034)

AddDefaultActions (see page 25052)

181.1.4.3.8 TAdapterActionsListHelper.IsActionInUse Method

Indicates whether there is a child Web item that corresponds to a specified action.

Pascal

```
function IsActionInUse(const AName: string): Boolean;
```

C++

```
__fastcall Boolean IsActionInUse(const AnsiString AName);
```

Description

IsActionInUse indicates whether the associated component already has a child Web item that executes a specified adapter action.

AName is the name of the adapter action.

IsActionInUse returns true if there is a child Web item that executes the specified action, false if there is no child Web item for the action.

See Also

FindAction (see page 25048)

GetActionsList (see page 25049)

GetVisibleActions (see page 25051)

ActionName (see page 25200)

181.1.4.4 TAdapterActionsListHelper Properties

The properties of the TAdapterActionsListHelper class are listed here.

Public Properties

	Name	Description
	AddDefaultActions (see page 25052)	Indicates whether the display control represents all of the adapter's default actions if you do not explicitly add items to its WebFieldControls property.

181.1.4.4.1 TAdapterActionsListHelper.AddDefaultActions Property

Indicates whether the display control represents all of the adapter's default actions if you do not explicitly add items to its WebFieldControls property.

Pascal

```
property AddDefaultActions: Boolean;
```

C++

```
__property Boolean AddDefaultActions;
```

Description

AddDefaultCommands determines what controls appear in the generated set of HTML controls if you do not explicitly add any adapter action buttons to the WebFieldControls property of the component that uses this TAdapterActionsListHelper (see page 25046). When AddDefaultCommands is true (the default), the associated component automatically adds controls for every default action of its adapter when you do not explicitly set the WebFieldControls property. When AddDefaultCommands is false, the associated component only generates controls if you explicitly add them to the WebFieldControls property. AddDefaultCommands must be false to allow an empty column.

See Also

WebFieldControls (see page 16884)

DefaultWebComponents (see page 23778)

AddDefaultCommands (see page 25215)

AddDefaultCommands (see page 25229)

181.1.5 TAdapterCommandColumn Class

TAdapterCommandColumn generates a column of action controls in an adapter grid.

Class Hierarchy

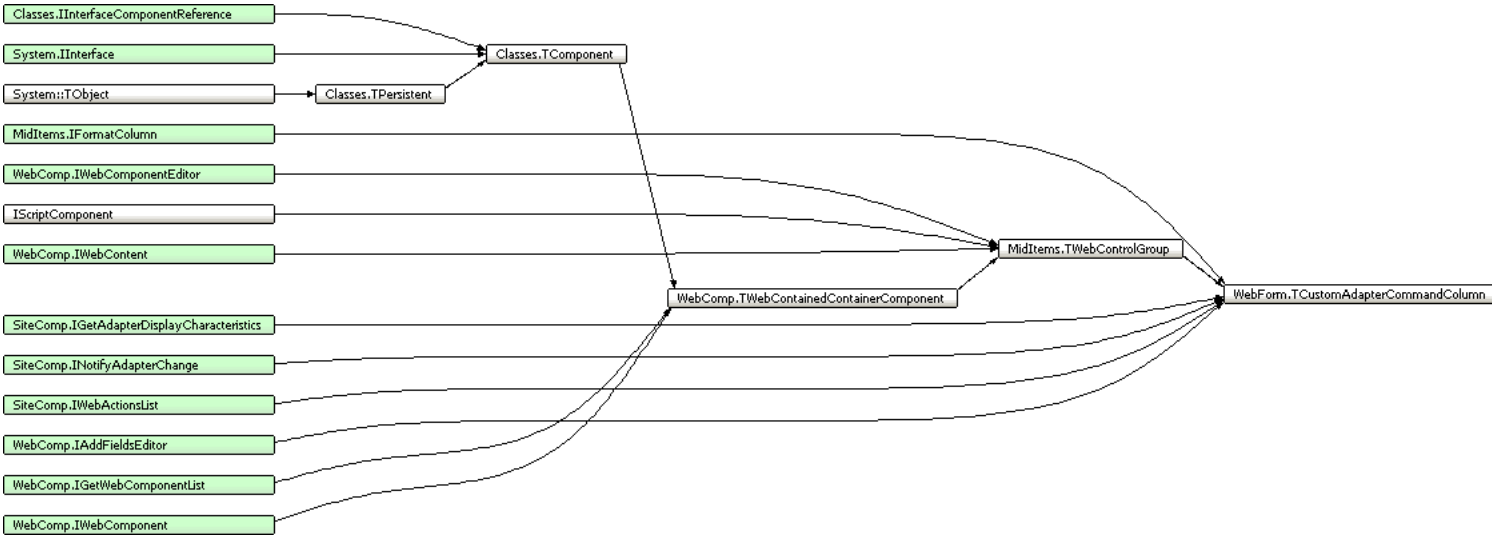

Pascal

```
TAdapterCommandColumn = class(TCustomAdapterCommandColumn);
```

C++

```
class TAdapterCommandColumn : public TCustomAdapterCommandColumn;
```

File

WebForm

Description

Add TAdapterCommandColumn as a child Web item to an adapter grid (TAdapterGrid (see page 25175)). TAdapterCommandColumn adds a column to the grid that contains controls that execute adapter actions. Specify the controls in the cells of this column by adding adapter action buttons (TAdapterActionButton (see page 25034) objects) as child Web items of the TAdapterCommandColumn object. Each cell in the column contains all the adapter action buttons you add as child Web items. For each row in the grid, the column contains duplicates of the same set of adapter action buttons.

Use the TAdapterCommandColumn properties to specify which adapter action buttons appear in the column and to customize their layout and appearance. These properties include:

The AddDefaultCommands (see page 25058), WebFieldControls, and VisibleCommands (see page 25219) properties, which specify which adapter action buttons appear in the column.

The Caption (see page 25059) and CaptionAttributes (see page 25060) properties, which determine the appearance of the label at the top of the column.

The Align (see page 25059), VAlign (see page 25062), and DisplayColumns (see page 25060) properties, which determine the layout of controls inside each cell.

The Style (see page 25061) and StyleRule (see page 25061) properties, which let you apply a style from a stylesheet to the column.

See Also

TAdapterGrid (see page 25175)

TAdapterEditColumn (see page 25132)

TAdapterDisplayColumn (see page 25099)

TAdapterActionButton (see page 25034)

181.1.5.1 TAdapterCommandColumn Members

The following tables list the members exposed by TAdapterCommandColumn.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FormatColumnData (see page 25211)	Generates the HTML for the cells in the column that contain controls to execute actions.
	FormatColumnHeading (see page 25212)	Generates the HTML for the table header cell for this column.
	GetAdapterDisplayCharacteristics (see page 25212)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterCommandColumn.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
V	GetLayoutAttributes (see page 25212)	Returns an object that specifies how the generated group of controls should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
V	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.

V	ImplCanAddClass (see page 16881)	Provides the underlying implementation for the CanAddClass (see page 16878) method.
V	ImplCanAddClass (see page 25213)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 16882)	Provides the underlying implementation for the Content (see page 16879) method.
V	ImplContent (see page 25213)	Provides the underlying implementation for the Content method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	NotifyAdapterChange (see page 25214)	Responds to changes that affect the associated adapter.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	AdapterActionsListHelper (see page 25215)	Identifies the helper object that enables this component to work with the editor at design time.
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.

R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16877)	Creates and initializes an instance of TWebControlGroup (see page 16872).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24742)	Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).
V	Create (see page 25210)	Creates and initializes an instance of TCustomAdapterCommandColumn (see page 25205).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16877)	Frees an instance of TWebControlGroup (see page 16872).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
V	Destroy (see page 25211)	Frees an instance of TCustomAdapterCommandColumn (see page 25205).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FormatColumnData (see page 17062)	This is FormatColumnData, a member of class IFormatColumn.
	FormatColumnHeading (see page 17062)	This is FormatColumnHeading, a member of class IFormatColumn.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterDisplayCharacteristics (see page 19687)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.
	GetAddAllFieldsItem (see page 24770)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.

	GetAddFieldsDlgCaption (see page 24770)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 24770)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNewFieldPrefix (see page 24771)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NotifyAdapterChange (see page 19725)	Responds when the adapter changes.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleCommands (see page 25219)	Lists the child Web items that generate controls that appear in cells of the column.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	AddDefaultCommands (see page 25058)	Indicates whether cells in the column contain controls for all of the adapter's default actions if you do not explicitly add items to the WebFieldControls property.
	Align (see page 25059)	Specifies the horizontal alignment of the action control buttons within the column.
	Caption (see page 25059)	Specifies the text label that appears at the top of the column.
	CaptionAttributes (see page 25060)	Specifies the attributes of the Caption property.
	Custom (see page 25060)	Represents a tag option and its value.
	DisplayColumns (see page 25060)	Specifies the number of adapter action buttons that can appear in a single row.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 25061)	Adds style attributes to the generated caption.
	StyleRule (see page 25061)	Applies a style in the page producer's style sheet to the generated column.
	Tag (see page 2808)	Stores an integer value as part of a component.
	VAlign (see page 25062)	Specifies the vertical alignment of the action control buttons within the column.

181.1.5.2 TAdapterCommandColumn Properties

The properties of the TAdapterCommandColumn class are listed here.

Published Properties

	Name	Description
	AddDefaultCommands (see page 25058)	Indicates whether cells in the column contain controls for all of the adapter's default actions if you do not explicitly add items to the WebFieldControls property.
	Align (see page 25059)	Specifies the horizontal alignment of the action control buttons within the column.
	Caption (see page 25059)	Specifies the text label that appears at the top of the column.
	CaptionAttributes (see page 25060)	Specifies the attributes of the Caption property.
	Custom (see page 25060)	Represents a tag option and its value.
	DisplayColumns (see page 25060)	Specifies the number of adapter action buttons that can appear in a single row.
	Style (see page 25061)	Adds style attributes to the generated caption.
	StyleRule (see page 25061)	Applies a style in the page producer's style sheet to the generated column.
	VAlign (see page 25062)	Specifies the vertical alignment of the action control buttons within the column.

181.1.5.2.1 TAdapterCommandColumn.AddDefaultCommands Property

Indicates whether cells in the column contain controls for all of the adapter's default actions if you do not explicitly add items to the WebFieldControls property.

Pascal

```
property AddDefaultCommands: Boolean;
```

C++

```
__property Boolean AddDefaultCommands;
```

Description

Set AddDefaultCommands property to specify what should appear in the generated column when you do not explicitly add any adapter action buttons to the WebFieldControls property. When AddDefaultCommands is true (the default), the adapter command column automatically adds controls for every default action of the associated adapter if you do not explicitly set the WebFieldControls property. When AddDefaultCommands is false, the adapter command column only generates controls if you explicitly add them to the WebFieldControls property. AddDefaultCommands must be false to allow an empty column.

See Also

WebFieldControls (see page 16884)

DefaultWebComponents (see page 23778)

VisibleCommands (see page 25219)

181.1.5.2.2 TAdapterCommandColumn.Align Property

Specifies the horizontal alignment of the action control buttons within the column.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align controls where the child HTML control are positioned horizontally within this column. Align indicates the desired value of the Align option of the HTML tag.

The default value of Align is haDefault, which means the web browser that reads the HTML document horizontally positions the control wherever it wishes.

See Also

VAlign (see page 25219)

DisplayColumns (see page 25217)

181.1.5.2.3 TAdapterCommandColumn.Caption Property

Specifies the text label that appears at the top of the column.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to get or set the caption for the column in the adapter grid. The caption is the text label that appears at the top of the column.

To change the appearance or layout of the caption, use the CaptionAttribute property.

See Also

CaptionAttributes (see page 25217)

181.1.5.2.4 TAdapterCommandColumn.CaptionAttributes Property

Specifies the attributes of the Caption property.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the column caption.

See Also

Caption (see page 25216)

181.1.5.2.5 TAdapterCommandColumn.Custom Property

Represents a tag option and its value.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to add one or more options to the tag that surrounds the column. Each option and its value has the form

`OPTIONNAME=Value`

The options specified by Custom should be separated by spaces.

TCustomAdapterCommandColumn inserts the Custom string into the HTML tag when it generates its content.

See Also

Style (see page 25218)

StyleRule (see page 25218)

181.1.5.2.6 TAdapterCommandColumn.DisplayColumns Property

Specifies the number of adapter action buttons that can appear in a single row.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Set DisplayColumns to organize the layout of adapter action buttons when the WebFieldControls property lists more than one

adapter action button. TCustomAdapterCommandColumn places the adapter action buttons by rows in the cells of the adapter grid column. DisplayColumns limits the number of controls that can appear in a single row within each column cell. If the WebFieldControls property lists more adapter action buttons than DisplayColumns, then the cell contains multiple rows of controls.

See Also

WebFieldControls (see page 16884)

181.1.5.2.7 TAdapterCommandColumn.Style Property

Adds style attributes to the generated caption.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated column. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the HTML tag that generates the column as the value of a STYLE option.

Note: Style affects the contents of the column cells. To specify display characteristics of the column's caption, use the CaptionAttributes property instead.

See Also

Custom (see page 25217)

StyleRule (see page 25218)

CaptionAttributes (see page 25217)

181.1.5.2.8 TAdapterCommandColumn.StyleRule Property

Applies a style in the page producer's style sheet to the generated column.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated column. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

To apply a style definition that is not included in the style sheet, use the Style property instead.

See Also

Custom (see page 25217)

Style (see page 25218)

Styles (see page 19830)

StylesFile (see page 19831)

181.1.5.2.9 TAdapterCommandColumn.VAlign Property

Specifies the vertical alignment of the action control buttons within the column.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign controls where the child HTML control are positioned vertically within the cells of this column. VAlign indicates the desired value of the VAlign option of the HTML tag.

The default value of VAlign is haVDefault, which means the web browser that reads the HTML document vertically positions the control wherever it wishes.

See Also

Align (see page 25216)

DisplayColumns (see page 25217)

181.1.6 TAdapterCommandGroup Class

TAdapterCommandGroup generates a group of buttons or other HTML elements that executes adapter actions.

Class Hierarchy

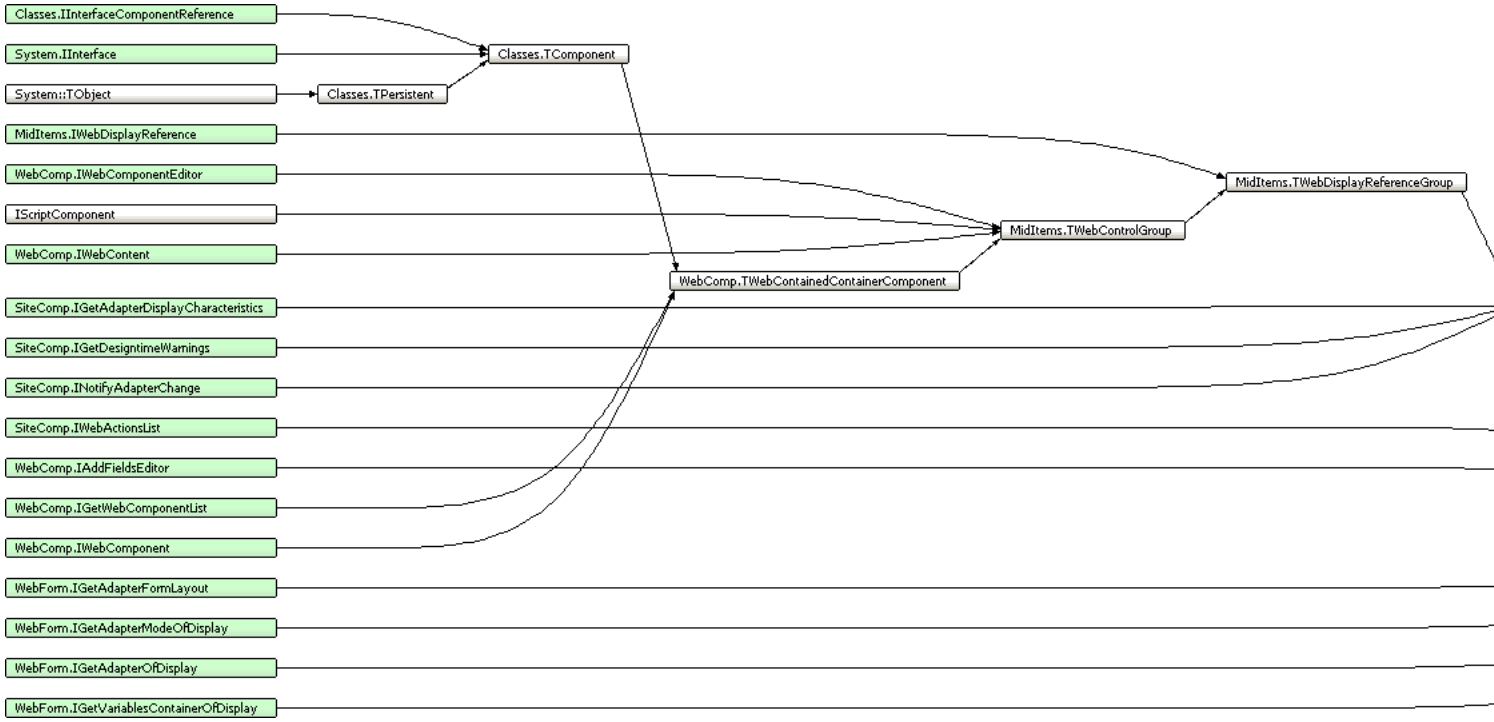

Pascal

```
TAdapterCommandGroup = class(TCustomAdapterCommandGroup);
```

C++

```
class TAdapterCommandGroup : public TCustomAdapterCommandGroup;
```

File

WebForm

Description

TAdapterCommandGroup is added to a TAdapterPageProducer to cause the page producer to generate a group of HTML elements that executes adapter actions. First add a TAdapterForm (see page 25159) to the TAdapterPageProducer and then add TAdapterCommandGroup as a child of TAdapterForm (see page 25159). TAdapterCommandGroup does not reference an adapter directly. Instead, it references a component that references an adapter. Typically, TAdapterCommandGroup references either TAdapterGrid (see page 25175) or TAdapterFieldGroup (see page 25150). TAdapterCommandGroup contains child components, such as TAdapterActionButton (see page 25034), that are responsible for rendering individual HTML elements, such as buttons. By default, TAdapterCommandGroup will automatically add one child component for each action in the adapter.

See Also

TAdapterDisplayReferenceGroup (see page 25122)

TCustomAdapterCommandGroup (see page 25220)

TAdapterForm (see page 25159)

TAdapterFieldGroup (see page 25150)

TAdapterGrid (see page 25175)

TAdapterActionButton (see page 25034)

181.1.6.1 TAdapterCommandGroup Members

The following tables list the members exposed by TAdapterCommandGroup.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AdapterChange (see page 25128)	Responds to changes in the adapter on which the controls in this group operate.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DisplayComponentChange (see page 25129)	Responds when the DisplayComponent (see page 25132) property changes.
	GetAdapter (see page 25226)	This is GetAdapter, a member of class TCustomAdapterCommandGroup.
	GetAdapterDisplayCharacteristics (see page 25129)	Indicates the display characteristics of the HTML this component generates.
	GetAdapterFormLayout (see page 25226)	This is GetAdapterFormLayout, a member of class TCustomAdapterCommandGroup.
	GetAdapterModeOfAdapter (see page 25227)	Returns the adapter mode defined by the adapter component.
	GetAdapterModeOfDisplay (see page 25227)	Returns the adapter mode defined by the TAdapterPageProducer components.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetDesigntimeWarnings (see page 25130)	Populates a design-time warnings object with any warnings that the reference group needs to emit.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.

	GetWebDisplayComponent (see page 16920)	This is GetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
V	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
V	ImplCanAddClass (see page 16881)	Provides the underlying implementation for the CanAddClass (see page 16878) method.
V	ImplCanAddClass (see page 25227)	Checks the inheritance of the given component.
V	ImplContent (see page 16882)	Provides the underlying implementation for the Content (see page 16879) method.
V	ImplContent (see page 25228)	Lays out the buttons for the command group using the TAdapterFormLayout (see page 25166) component.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16920)	Forwards notification messages to all owned components.
	NotifyAdapterChange (see page 25130)	Responds when other components notify this one of changes to the adapter.
	OnDisplayComponentChange (see page 25131)	Responds when the associated display component changes.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
V	SetWebDisplayComponent (see page 16921)	This is SetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
V	SetWebDisplayComponent (see page 25131)	Sets the value of the DisplayComponent (see page 25132) property.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 25228)	Provides read-only access to the Adapter that this command group uses.
R	AdapterActionsListHelper (see page 25229)	Provides read-only access to a helper object that implements design-time editing functions on behalf of this component.
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebDisplay (see page 16921)	This is WebDisplay, a member of class TWebDisplayReferenceGroup.
	WebDisplayComponent (see page 16921)	
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16877)	Creates and initializes an instance of TWebControlGroup (see page 16872).
V	Create (see page 16919)	Creates an instance of TWebDisplayReferenceGroup (see page 16915).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24742)	Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).
V	Create (see page 25127)	Creates an instance of TWebDisplayReferenceGroup.
V	Create (see page 25225)	Creator for TCustomAdapterCommandGroup (see page 25220).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16877)	Frees an instance of TWebControlGroup (see page 16872).
V	Destroy (see page 16919)	Frees an instance of TWebDisplayReferenceGroup (see page 16915).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
V	Destroy (see page 25128)	Frees an instance of TAdapterDisplayReferenceGroup (see page 25122).
V	Destroy (see page 25225)	Frees an instance of TAdapterDisplayReferenceGroup.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapter (see page 25306)	This is GetAdapter, a member of class IGetAdapterOfDisplay.
	GetAdapterDisplayCharacteristics (see page 19687)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.
	GetAdapterFormLayout (see page 25304)	This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.
	GetAdapterModeOfAdapter (see page 25305)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 25305)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.
	GetAddAllFieldsItem (see page 24770)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 24770)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 24770)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetDesigntimeWarnings (see page 19697)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNewFieldPrefix (see page 24771)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	GetVariablesContainer (see page 25311)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetWebDisplayComponent (see page 17081)	This is GetWebDisplayComponent, a member of class IWebDisplayReference.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NotifyAdapterChange (see page 19725)	Responds when the adapter changes.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleCommands (see page 25230)	Provides read-only access to the TWebComponentList that stores the list of visible components.
R	WebDisplayComponent (see page 17081)	
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	AddDefaultCommands (see page 25069)	Provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.
	Custom (see page 25069)	Provides read and write access to the custom HTML element attributes, such as Valign="top."
	DisplayComponent (see page 25069)	Associates the group with a component that generates the display controls.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 25070)	Provides read and write access to the internal string that stores the CSS style, such as making the background color red.
	StyleRule (see page 25070)	Provides read and write access to the internal string that stores the CSS style rule for the control group.
	Tag (see page 2808)	Stores an integer value as part of a component.

181.1.6.2 TAdapterCommandGroup Properties

The properties of the TAdapterCommandGroup class are listed here.

Published Properties

	Name	Description
	AddDefaultCommands (see page 25069)	Provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.
	Custom (see page 25069)	Provides read and write access to the custom HTML element attributes, such as Valign="top."
	DisplayComponent (see page 25069)	Associates the group with a component that generates the display controls.
	Style (see page 25070)	Provides read and write access to the internal string that stores the CSS style, such as making the background color red.
	StyleRule (see page 25070)	Provides read and write access to the internal string that stores the CSS style rule for the control group.

181.1.6.2.1 TAdapterCommandGroup.AddDefaultCommands Property

Provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.

Pascal

```
property AddDefaultCommands: Boolean;
```

C++

```
__property Boolean AddDefaultCommands;
```

Description

The AddDefaultCommands property provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.

181.1.6.2.2 TAdapterCommandGroup.Custom Property

Provides read and write access to the custom HTML element attributes, such as Valign="top."

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

The Custom property provides read and write access to the custom HTML element attributes, such as Valign="top." The content of this string will be written as it is when generating HTML.

181.1.6.2.3 TAdapterCommandGroup.DisplayComponent Property

Associates the group with a component that generates the display controls.

Pascal

```
property DisplayComponent: TComponent;
```

C++

```
__property TComponent * DisplayComponent;
```

Description

Use DisplayComponent to associate the group of controls with a Web item that generates the display elements those controls influence. Typical values of DisplayComponent are TAdapterGrid or TAdapterFieldGroup.

TAdapterDisplayReferenceGroup descendants use the value of DisplayComponent to access the display controls to influence, and to access the adapter, whose information is displayed or manipulated.

See Also

TAdapterGrid (see page 25175)

TAdapterFieldGroup (see page 25150)

181.1.6.2.4 TAdapterCommandGroup.Style Property

Provides read and write access to the internal string that stores the CSS style, such as making the background color red.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

The Style property provides read and write access to the internal string that stores the CSS style, such as making the background color red.

181.1.6.2.5 TAdapterCommandGroup.StyleRule Property

Provides read and write access to the internal string that stores the CSS style rule for the control group.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

The StyleRule property provides read and write access to the internal string that stores the CSS style rule for the control group. This property is used to reference a CSS style defined in TAdapterPageProducer. ::Styles.

181.1.7 TAdapterControlGroup Class

TAdapterControlGroup is a container for adapter display field components such as TAdapterDisplayField (see page 25109) or TAdapterDisplayColumn (see page 25099).

Class Hierarchy

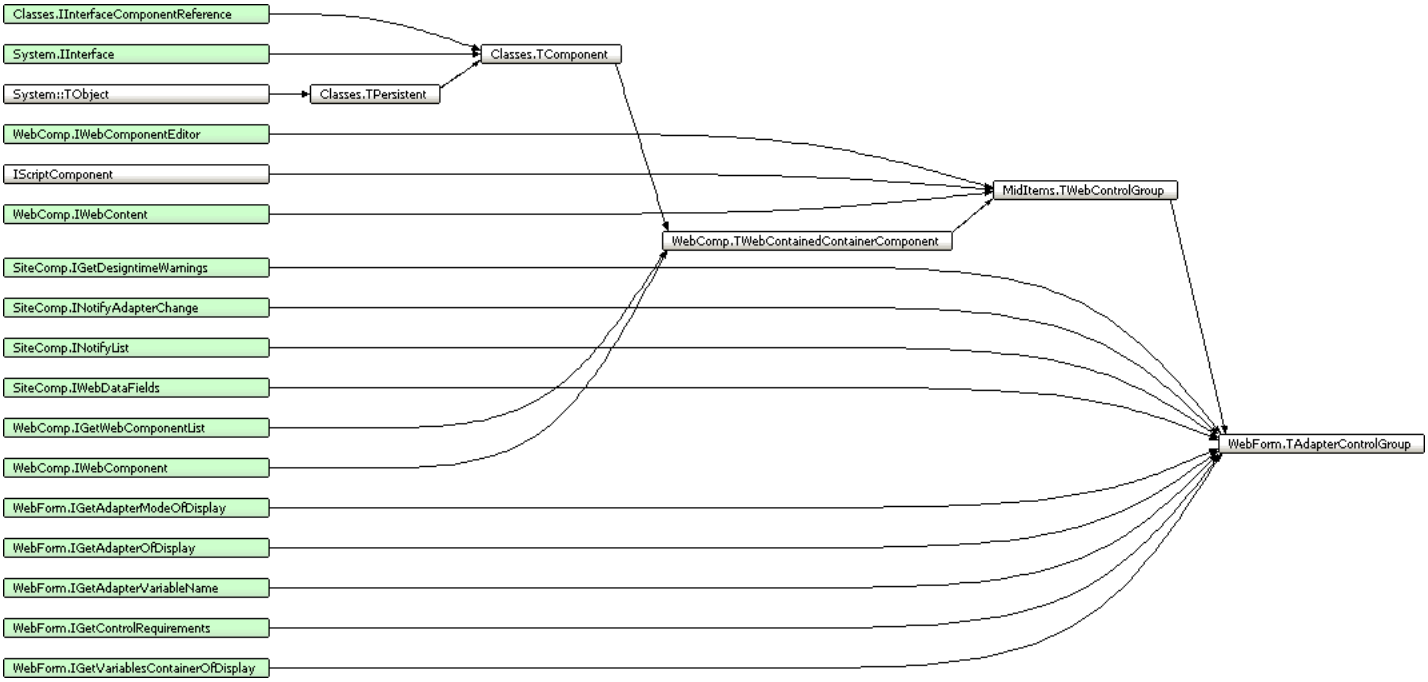

Pascal

```
TAdapterControlGroup = class(TWebControlGroup, IWebDataFields, INotifyAdapterChange,
IGetAdapterVariableName, IGetAdapterOfDisplay, INotifyList, IGetVariablesContainerOfDisplay,
IGetDesigntimeWarnings, IGetAdapterModeOfDisplay, IGetControlRequirements);
```

C++

```
class TAdapterControlGroup : public TWebControlGroup, public IWebDataFields, public
INotifyAdapterChange, public IGetAdapterVariableName, public IGetAdapterOfDisplay, public
INotifyList, public IGetVariablesContainerOfDisplay, public IGetDesigntimeWarnings, public
IGetAdapterModeOfDisplay, public IGetControlRequirements;
```

File

WebForm

Description

Do not create instances of TAdapterControlGroup. Instead, use a descendant of TAdapterControlGroup that manages particular types of display field components. These descendants include TAdapterFieldGroup (see page 25150) and TAdapterGrid (see page 25175).

Descendants of TAdapterControlGroup are children of TAdapterPageProducer form components. Use the Web page editor to add TAdapterControlGroup descendants to an adapter form when building an HTML form.

See Also

TAdapterFieldGroup (see page 25150)

TAdapterGrid (see page 25175)

TAdapterPageProducer (see page 19802)

TAdapterForm (see page 25159)

TAdapter (see page 23713)

181.1.7.1 TAdapterControlGroup Members

The following tables list the members exposed by TAdapterControlGroup.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AdapterChange (see page 25077)	Responds when the adapter associated with this control group changes.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
	AddNotify (see page 25077)	Registers interest so that the adapter control group notifies an object of changes to its adapter.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
	CreateDefaultFields (see page 25078)	Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FindField (see page 25078)	Returns the child Web item that represents the value of a specified adapter field.
	GetAdapter (see page 25079)	Returns the value of the Adapter (see page 25084) property.
	GetAdapterModeOfAdapter (see page 25079)	Returns the current mode of the control group's adapter.
	GetAdapterModeOfDisplay (see page 25080)	Returns the value of the AdapterMode (see page 25085) property.
	GetAdapterVariableName (see page 25080)	Returns the variable name that refers to the control group's adapter in server-side script.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetControlRequirements (see page 25080)	Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetDesigntimeWarnings (see page 25081)	Populates a design-time warnings object with any warnings that the adapter control group needs to emit.
	GetFieldsList (see page 25081)	Fills a string list with the names of all of the associated adapter's fields.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.

	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVisibleFields (see page 25082)	Returns the value of the VisibleFields (see page 25086) property.
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
V	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
V	ImplCanAddClass (see page 16881)	Provides the underlying implementation for the CanAddClass (see page 16878) method.
V	ImplContent (see page 16882)	Provides the underlying implementation for the Content (see page 16879) method.
V	ImplGetControlRequirements (see page 25082)	Provides the underlying implementation for the GetControlRequirements (see page 25080) method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsFieldInUse (see page 25082)	Indicates whether one of the adapter control group's child Web items represents a specified adapter field.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 25083)	Updates the TAdapterControlGroup (see page 25070) object when components are added to or removed from the application.
	NotifyAdapterChange (see page 25083)	Responds when the control group's adapter notifies it of a change.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	RemoveNotify (see page 25083)	Unregisters interest in changes to the control group's adapter.
	SetAdapter (see page 25084)	Sets the value of the Adapter (see page 25084) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.

V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 25076)	Creates and initializes an instance of TAdapterControlGroup (see page 25070).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25076)	Frees an instance of TAdapterControlGroup (see page 25070).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	GetVariablesContainer (see page 25311)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Adapter (see page 25084)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 25085)	Specifies the mode of the adapter that the controls in the group support.
R	AdapterVariableName (see page 25308)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	AddDefaultFields (see page 25085)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.

	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleFields (see page 25086)	Lists the child Web items that generate the visible HTML controls this adapter control group arranges.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

181.1.7.2 TAdapterControlGroup.Create Constructor

Creates and initializes an instance of TAdapterControlGroup (see page 25070).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TAdapterControlGroup(TComponent * AOwner);
```

Description

Call Create to instantiate TAdapterControlGroup (see page 25070) at runtime. TAdapterControlGroup (see page 25070) objects added at design time are instantiated automatically.

See Also

Owner (see page 2807)

181.1.7.3 TAdapterControlGroup.Destroy Destructor

Frees an instance of TAdapterControlGroup (see page 25070).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterControlGroup();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks for that the object is not nil before calling Destroy.

See Also

Free (see page 22300)

181.1.7.4 TAdapterControlGroup Methods

The methods of the TAdapterControlGroup class are listed here.

Protected Methods

	Name	Description
V	AdapterChange (see page 25077)	Responds when the adapter associated with this control group changes.
	AddNotify (see page 25077)	Registers interest so that the adapter control group notifies an object of changes to its adapter.

	CreateDefaultFields (see page 25078)	Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.
	FindField (see page 25078)	Returns the child Web item that represents the value of a specified adapter field.
	GetAdapter (see page 25079)	Returns the value of the Adapter (see page 25084) property.
	GetAdapterModeOfAdapter (see page 25079)	Returns the current mode of the control group's adapter.
	GetAdapterModeOfDisplay (see page 25080)	Returns the value of the AdapterMode (see page 25085) property.
	GetAdapterVariableName (see page 25080)	Returns the variable name that refers to the control group's adapter in server-side script.
	GetControlRequirements (see page 25080)	Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.
	GetDesigntimeWarnings (see page 25081)	Populates a design-time warnings object with any warnings that the adapter control group needs to emit.
	GetFieldsList (see page 25081)	Fills a string list with the names of all of the associated adapter's fields.
	GetVisibleFields (see page 25082)	Returns the value of the VisibleFields (see page 25086) property.
V	ImplGetControlRequirements (see page 25082)	Provides the underlying implementation for the GetControlRequirements (see page 25080) method.
	IsFieldInUse (see page 25082)	Indicates whether one of the adapter control group's child Web items represents a specified adapter field.
V	Notification (see page 25083)	Updates the TAdapterControlGroup (see page 25070) object when components are added to or removed from the application.
	NotifyAdapterChange (see page 25083)	Responds when the control group's adapter notifies it of a change.
	RemoveNotify (see page 25083)	Unregisters interest in changes to the control group's adapter.
	SetAdapter (see page 25084)	Sets the value of the Adapter (see page 25084) property.

181.1.7.4.1 TAdapterControlGroup.AdapterChange Method

Responds when the adapter associated with this control group changes.

Pascal

```
procedure AdapterChange; virtual;
```

C++

```
virtual __fastcall AdapterChange();
```

Description

AdapterChange is called automatically when the value of the Adapter (see page 25084) property changes, or when the application makes changes to the adapter that is the value of the Adapter (see page 25084) property.

AdapterChange performs the following tasks to respond to the change:

Updates the list of default adapter display field components that is the value of VisibleFields (see page 25086) when you have not explicitly assigned child Web items.

Notifies all of the child Web items listed in the WebFieldControls property that the adapter has changed.

Notifies any objects that registered interest by calling the AddNotify (see page 25077) method that the adapter has changed.

See Also

Adapter (see page 25084)

AddNotify (see page 25077)

WebFieldControls (see page 16884)

VisibleFields (see page 25086)

181.1.7.4.2 TAdapterControlGroup.AddNotify Method

Registers interest so that the adapter control group notifies an object of changes to its adapter.

Pascal

```
procedure AddNotify(ANotify: TObject);
```

C++

```
__fastcall AddNotify(TObject * ANotify);
```

Description

Other Web items in the adapter form call AddNotify to register an interest in when the control group's adapter changes. AddNotify adds the specified object to an internal list of interested objects. When the Adapter (see page 25084) property changes, it calls every object in this internal list to notify it of the change.

There is no need for the child Web items listed in the WebFieldControls property to call AddNotify. These objects are notified automatically.

ANotify is the object that registers an interest in change notifications.

See Also

AdapterChange (see page 25077)

RemoveNotify (see page 25083)

Adapter (see page 25084)

181.1.7.4.3 TAdapterControlGroup.CreateDefaultFields Method

Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.

Pascal

```
procedure CreateDefaultFields;
```

C++

```
__fastcall CreateDefaultFields();
```

Description

TAdapterControlGroup (see page 25070) calls CreateDefaultFields internally when it needs to generate child Web items for the fields of the adapter that the Adapter (see page 25084) property specifies. The resulting list becomes the value of the VisibleFields (see page 25086) property if you have not explicitly added any child Web items to the WebFieldControls property and AddDefaultFields (see page 25085) is true.

See Also

VisibleFields (see page 25086)

Adapter (see page 25084)

AddDefaultFields (see page 25085)

WebFieldControls (see page 16884)

181.1.7.4.4 TAdapterControlGroup.FindField Method

Returns the child Web item that represents the value of a specified adapter field.

Pascal

```
function FindField(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindField(const AnsiString AName);
```

Description

FindField searches the list of child Web items for one that represents the specified adapter field. If it finds a Web item that represents the specified field, it returns that adapter display field component. Otherwise, FindField returns nil (Delphi) or NULL (C++).

AName is the name of a field that the adapter control group's adapter contains.

See Also

VisibleFields (see page 25086)

GetFieldsList (see page 25081)

Adapter (see page 25084)

181.1.7.4.5 TAdapterControlGroup.GetAdapter Method

Returns the value of the Adapter (see page 25084) property.

Pascal

```
function GetAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetAdapter();
```

Description

GetAdapter is the protected read implementation of the Adapter (see page 25084) property.

See Also

Adapter (see page 25084)

181.1.7.4.6 TAdapterControlGroup.GetAdapterModeOfAdapter Method

Returns the current mode of the control group's adapter.

Pascal

```
function GetAdapterModeOfAdapter: string;
```

C++

```
__fastcall AnsiString GetAdapterModeOfAdapter();
```

Description

GetAdapterModeOfAdapter returns the current mode of the adapter that the Adapter (see page 25084) property specifies. If the adapter's mode is not currently set, GetAdapterModeOfAdapter returns the value of the AdapterMode (see page 25085) property.

See Also

AdapterMode (see page 25085)

Adapter (see page 25084)

GetAdapterModeOfDisplay (see page 25080)

181.1.7.4.7 TAdapterControlGroup.GetAdapterModeOfDisplay Method

Returns the value of the AdapterMode (see page 25085) property.

Pascal

```
function GetAdapterModeOfDisplay: string;
```

C++

```
__fastcall AnsiString GetAdapterModeOfDisplay();
```

Description

GetAdapterModeOfDisplay returns the value of the AdapterMode (see page 25085) property. This is the value that influences the type and behavior of the HTML controls that child Web items generate. It may differ from the current mode of the control group's adapter, which the GetAdapterModeOfAdapter (see page 25079) method returns.

See Also

AdapterMode (see page 25085)

GetAdapterModeOfAdapter (see page 25079)

181.1.7.4.8 TAdapterControlGroup.GetAdapterVariableName Method

Returns the variable name that refers to the control group's adapter in server-side script.

Pascal

```
function GetAdapterVariableName: string;
```

C++

```
__fastcall AnsiString GetAdapterVariableName();
```

Description

TAdapterControlGroup (see page 25070) uses GetAdapterVariableName internally when generating server-side script that refers to its adapter.

See Also

Adapter (see page 25084)

181.1.7.4.9 TAdapterControlGroup.GetControlRequirements Method

Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.

Pascal

```
procedure GetControlRequirements(ARequirements: TAbstractControlRequirements);
```

C++

```
__fastcall GetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

The adapter form object that includes this control group as a child Web item calls GetControlRequirements to obtain declarations for the variables and functions that the control group uses to assemble HTML controls for its adapter's fields.

ARequirements is the control requirements object that collects a list of required variables and functions.

GetControlRequirements simply calls the ImplGetControlRequirements (see page 25082) method, which adds a variable that represents the control group's adapter to ARequirements if it is not already there.

See Also

ImplGetControlRequirements (see page 25082)

TAdapterForm (see page 25159)

GetAdapterVariableName (see page 25080)

181.1.7.4.10 TAdapterControlGroup.GetDesigntimeWarnings Method

Populates a design-time warnings object with any warnings that the adapter control group needs to emit.

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

GetDesigntimeWarnings is called automatically at design time when the adapter page producer is assembling the current version of its HTML form. This gives the adapter control group an opportunity to add messages to the version of the HTML form that appears at design time, describing problems that you can rectify.

If the Adapter (see page 25084) property has not been set, GetDesigntimeWarnings adds a message that indicates this omission. Otherwise, GetDesigntimeWarnings adds a reference to Adapter (see page 25084) to the design-time warnings object so that it can obtain any warning messages that the adapter wants to add.

AWarnings is the design-time warnings object that assembles design-time warning messages that appear in the HTML form.

See Also

Adapter (see page 25084)

TAdapterPageProducer (see page 19802)

181.1.7.4.11 TAdapterControlGroup.GetFieldsList Method

Fills a string list with the names of all of the associated adapter's fields.

Pascal

```
procedure GetFieldsList(List: TStrings);
```

C++

```
__fastcall GetFieldsList(TStrings List);
```

Description

The CreateDefaultFields (see page 25078) method calls GetFieldsList to obtain a list of adapter fields for which it needs to create default child Web items. GetFieldsList obtains a list of the names of its fields from the adapter, and adds them to the specified string list. For any field that is a "default" field (as defined by the adapter), the string in the list is associated with the class of the child Web item that generates a control to represent that field.

List is a string list to fill with the names of (and Web item classes for) the adapter's default fields.

See Also

CreateDefaultFields (see page 25078)

Adapter (see page 25084)

181.1.7.4.12 TAdapterControlGroup.GetVisibleFields Method

Returns the value of the VisibleFields (see page 25086) property.

Pascal

```
function GetVisibleFields: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleFields();
```

Description

GetVisibleFields is the protected implementation of the VisibleFields (see page 25086) property.

See Also

VisibleFields (see page 25086)

181.1.7.4.13 TAdapterControlGroup.ImplGetControlRequirements Method

Provides the underlying implementation for the GetControlRequirements (see page 25080) method.

Pascal

```
procedure ImplGetControlRequirements(ARequirements: TAbstractControlRequirements); virtual;
```

C++

```
virtual __fastcall ImplGetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

The GetControlRequirements (see page 25080) method calls ImplGetControlRequirements to add the variables and functions the control group needs to have declared to the control requirements object. This method adds the requirement of a variable with the name that the GetAdapterVariableName (see page 25080) method returns to represent the adapter whose fields the control group displays.

ARequirements is the control requirements object that collects a list of required variables and functions.

See Also

GetControlRequirements (see page 25080)

GetAdapterVariableName (see page 25080)

181.1.7.4.14 TAdapterControlGroup.IsFieldInUse Method

Indicates whether one of the adapter control group's child Web items represents a specified adapter field.

Pascal

```
function IsFieldInUse(const AName: string): Boolean;
```

C++

```
__fastcall Boolean IsFieldInUse(const AnsiString AName);
```

Description

IsFieldInUse returns true if one of its child Web items represents the adapter field with the specified name, false otherwise.

AName is the name of a field that the adapter control group's adapter contains.

See Also

FindField (see page 25078)

VisibleFields (see page 25086)

Adapter (see page 25084)

181.1.7.4.15 TAdapterControlGroup.Notification Method

Updates the TAdapterControlGroup (see page 25070) object when components are added to or removed from the application.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

In TAdapterControlGroup (see page 25070), Notification checks whether AComponent is the value of the Adapter (see page 25084) property. If so, and Operation is opRemove, Notification sets the Adapter (see page 25084) property to nil (Delphi) or NULL (C++).

See Also

Adapter (see page 25084)

181.1.7.4.16 TAdapterControlGroup.NotifyAdapterChange Method

Responds when the control group's adapter notifies it of a change.

Pascal

```
procedure NotifyAdapterChange(Sender: TComponent);
```

C++

```
__fastcall NotifyAdapterChange(TComponent * Sender);
```

Description

NotifyAdapterChange is called automatically when the application makes changes to the adapter that is the value of the Adapter (see page 25084) property. This method calls the AdapterChange (see page 25077) method, which updates the list of default fields to be represented, and forwards the notification on to other Web items that may need to adjust themselves.

Sender is the component that sent the notification.

See Also

Adapter (see page 25084)

AdapterChange (see page 25077)

181.1.7.4.17 TAdapterControlGroup.RemoveNotify Method

Unregisters interest in changes to the control group's adapter.

Pascal

```
procedure RemoveNotify(ANotify: TObject);
```

C++

```
__fastcall RemoveNotify(TObject * ANotify);
```

Description

Other Web items in the adapter form call RemoveNotify to unregister an interest in when the control group's adapter changes. The object must have previously registered interest by a call to the AddNotify (see page 25077) method. RemoveNotify removes the specified object from the internal list of interested objects.

ANotify is the object that had previously registered an interest in change notifications.

See Also

AdapterChange (see page 25077)

AddNotify (see page 25077)

181.1.7.4.18 TAdapterControlGroup.SetAdapter Method

Sets the value of the Adapter (see page 25084) property.

Pascal

```
procedure SetAdapter(const Value: TComponent);
```

C++

```
__fastcall SetAdapter(const TComponent * Value);
```

Description

SetAdapter is the protected write implementaiton of the Adapter (see page 25084) property.

See Also

Adapter (see page 25084)

181.1.7.5 TAdapterControlGroup Properties

The properties of the TAdapterControlGroup class are listed here.

Public Properties

	Name	Description
	Adapter (see page 25084)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 25085)	Specifies the mode of the adapter that the controls in the group support.
	AddDefaultFields (see page 25085)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
R	VisibleFields (see page 25086)	Lists the child Web items that generate the visible HTML controls this adapter control group arranges.

181.1.7.5.1 TAdapterControlGroup.Adapter Property

Specifies the adapter that supplies values to or receives values from the controls in the group.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

Set Adapter to link this adapter control group and all its Web items to an adapter component. Each component that the WebFieldControls property lists generates an HTML control that represents the value of one of the adapter's fields.

See Also

GetFieldsList (see page 24159)

GetFieldsList (see page 25081)

AdapterMode (see page 25085)

WebFieldControls (see page 16884)

181.1.7.5.2 TAdapterControlGroup.AdapterMode Property

Specifies the mode of the adapter that the controls in the group support.

Pascal

```
property AdapterMode: string;
```

C++

```
__property AnsiString AdapterMode;
```

Description

Some adapter components support different modes, or states. For example, dataset adapters can be in Edit, Insert, Browse, or Query mode. The mode of the adapter has an effect on the types of controls that the control group's Web items generate, and how those controls behave.

If the adapter that is assigned to the Adapter (see page 25084) property supports different modes, you can set AdapterMode to force the controls in the group to represent the adapter fields in the specified mode. When AdapterMode is blank, the adapter control group uses the current mode of the adapter at the time when it generates its controls in the HTML form.

The possible values of AdapterMode depend on the modes that the adapter supports. If the adapter supports different modes, you can select a value for AdapterMode from a drop-down list in the Object Inspector that lists all the possible mode strings. If the adapter does not support different modes, the Object Inspector does not display a list, and the value of AdapterMode should be blank.

See Also

Mode (see page 7810)

Adapter (see page 25084)

WebFieldControls (see page 16884)

181.1.7.5.3 TAdapterControlGroup.AddDefaultFields Property

Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.

Pascal

```
property AddDefaultFields: Boolean;
```

C++

```
__property Boolean AddDefaultFields;
```

Description

Set AddDefaultFields to indicate how you want the adapter control group to respond when you do not explicitly add child Web items. You can add child Web items at design time using Adapter (see page 25084) page editor, or by right-clicking the control group and selecting Add Fields or Add Component. You can add child Web items at runtime using the Add method of the WebFieldControls property.

When AddDefaultFields is true, if you do not add child Web items to the adapter control group, it generates a default set of child Web items to display the fields of the adapter component that the Adapter (see page 25084) property specifies.

When AddDefaultFields is false, if you do not add child Web items to the adapter control group, it remains empty, displaying no field values.

See Also

VisibleFields (see page 25086)

WebFieldControls (see page 16884)

Adapter (see page 25084)

181.1.7.5.4 TAdapterControlGroup.VisibleFields Property

Lists the child Web items that generate the visible HTML controls this adapter control group arranges.

Pascal

```
property VisibleFields: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleFields;
```

Description

Read VisibleFields to determine the adapter display field components that generate HTML controls for this adapter control group. If you have explicitly added child Web items to the control group, or if the AddDefaultFields (see page 25085) property is false, VisibleFields is the same as the value of the WebFieldControls property. If you have not added any child Web items and AddDefaultFields (see page 25085) is true, VisibleFields is a list of adapter display field components that generate controls for the default fields of the control group's adapter.

See Also

WebFieldControls (see page 16884)

AddDefaultFields (see page 25085)

Adapter (see page 25084)

181.1.8 TAdapterDataDisplay Class

TAdapterDataDisplay is the base class for components that display adapter fields as form fields.

Class Hierarchy

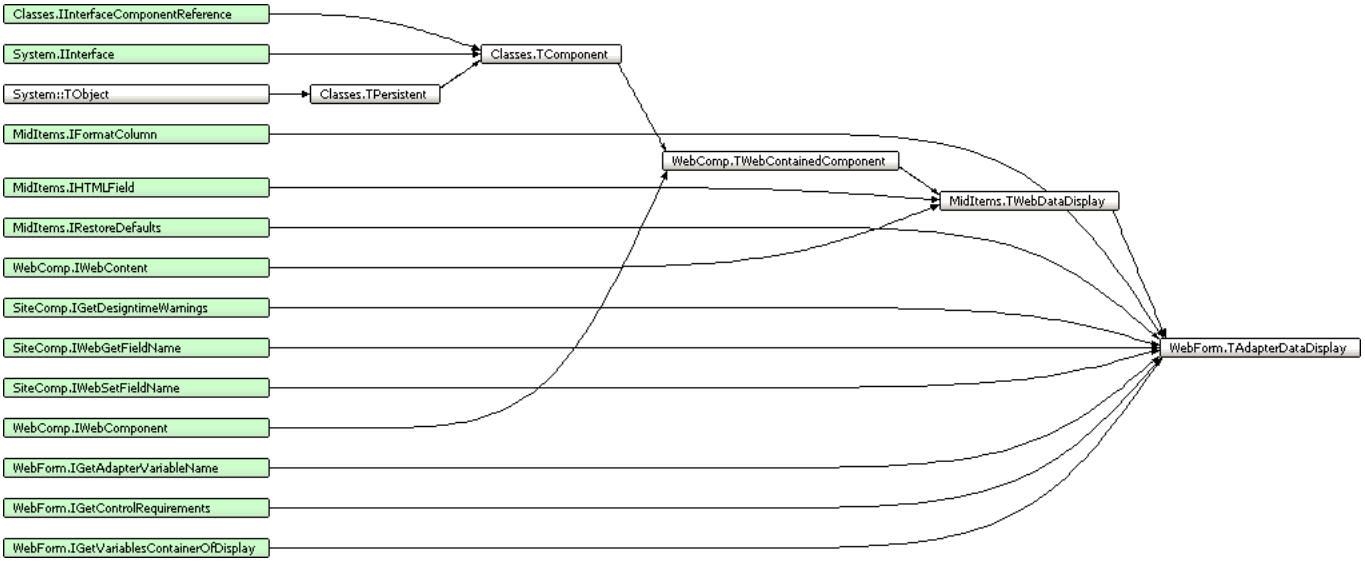

Pascal

```
TAdapterDataDisplay = class(TWebDataDisplay, IWebGetFieldName, IWebSetFieldName,
IGetAdapterVariableName, IRestoreDefaults, IGetControlRequirements,
IGetVariablesContainerOfDisplay, IGetDesigntimeWarnings, IFormatColumn);
```

C++

```
class TAdapterDataDisplay : public TWebDataDisplay, public IWebGetFieldName, public
IWebSetFieldName, public IGetAdapterVariableName, public IRestoreDefaults, public
IGetControlRequirements, public IGetVariablesContainerOfDisplay, public
IGetDesigntimeWarnings, public IFormatColumn;
```

File

WebForm

Description

TAdapterDataDisplay is the base class for components that display adapter fields as HTML form fields. Its descendants include the following classes:

TAdapterDisplayField (see page 25109), which supports free-form display and editing of adapter field data.

TAdapterDisplayColumn (see page 25099), which supports grid display of adapter field data.

TAdapterEditColumn (see page 25132), which supports grid editing of adapter field data.

See Also

TAdapterDisplayField (see page 25109)

TAdapterDisplayColumn (see page 25099)

TAdapterEditColumn (see page 25132)

181.1.8.1 TAdapterDataDisplay Members

The following tables list the members exposed by TAdapterDataDisplay.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	FieldConditions (see page 25092)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 25093)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
V	FormatCaption (see page 25093)	This is FormatCaption, a member of class TAdapterDataDisplay.
	FormatColumnData (see page 25093)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 25093)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	GetAdapterModeOfAdapter (see page 25093)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 25094)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
V	GetCaption (see page 25094)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetControlRequirements (see page 25094)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 25094)	This is GetDesigntimeWarnings, a member of class TAdapterDataDisplay.
	GetFieldName (see page 25095)	This is GetFieldName, a member of class TAdapterDataDisplay.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariablesContainer (see page 25095)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 25095)	Provides the underlying implementation for the Content method.
V	ImplFormatColumnData (see page 25096)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
V	ImplFormatColumnHeading (see page 25096)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
V	ImplGetControlRequirements (see page 25096)	This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.

V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
V	ImplRestoreDefaults (see page 25096)	This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 25096)	This is RestoreDefaults, a member of class TAdapterDataDisplay.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 25097)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 25097)	This is SetFieldName, a member of class TAdapterDataDisplay.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	CaptionPosition (see page 16897)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
	Create (see page 25092)	Creates and initializes an instance of TWebDataDisplay.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.

V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AdapterVariableName (see page 25308)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	Align (see page 25097)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 16897)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16897)	Customizes the appearance of the generated control's label.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16898)	Represents a tag option and its value in the tag for the display control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	FieldName (see page 25098)	Specifies the name of the associated dataset field.
	HideOptions (see page 25098)	When field should be hidden.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16899)	Adds style attributes to the generated display control.
	StyleRule (see page 16899)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 16899)	Determines the position of the generated control in the HTML form's tab order.
	VAlign (see page 25098)	Specifies the vertical alignment of the generated display control.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

181.1.8.2 TAdapterDataDisplay.Create Constructor

Creates and initializes an instance of TWebDataDisplay.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TAdapterDataDisplay(TComponent * AOwner);
```

Description

Call Create to instantiate TWebDataDisplay at runtime. TWebDataDisplay descendants added at design time are instantiated automatically.

181.1.8.3 TAdapterDataDisplay Methods

The methods of the TAdapterDataDisplay class are listed here.

Protected Methods

	Name	Description
	FieldConditions (see page 25092)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 25093)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
	FormatCaption (see page 25093)	This is FormatCaption, a member of class TAdapterDataDisplay.
	FormatColumnData (see page 25093)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 25093)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	GetAdapterModeOfAdapter (see page 25093)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 25094)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
	GetCaption (see page 25094)	Returns the value of the Caption property.
	GetControlRequirements (see page 25094)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 25094)	This is GetDesigntimeWarnings, a member of class TAdapterDataDisplay.
	GetFieldName (see page 25095)	This is GetFieldName, a member of class TAdapterDataDisplay.
	GetVariablesContainer (see page 25095)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
	ImplContent (see page 25095)	Provides the underlying implementation for the Content method.
	ImplFormatColumnData (see page 25096)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
	ImplFormatColumnHeading (see page 25096)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
	ImplGetControlRequirements (see page 25096)	This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.
	ImplRestoreDefaults (see page 25096)	This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.
	RestoreDefaults (see page 25096)	This is RestoreDefaults, a member of class TAdapterDataDisplay.
	SetCaption (see page 25097)	Sets the value of the Caption property.
	SetFieldName (see page 25097)	This is SetFieldName, a member of class TAdapterDataDisplay.

181.1.8.3.1 TAdapterDataDisplay.FieldConditions Method

Pascal

```
function FieldConditions(Options: TWebContentOptions): string; virtual;
```

C++

```
virtual __fastcall AnsiString FieldConditions(TWebContentOptions Options);
```

Description

This is FieldConditions, a member of class TAdapterDataDisplay.

181.1.8.3.2 TAdapterDataDisplay.FindAssociatedField Method

Pascal

```
function FindAssociatedField: TComponent;
```

C++

```
__fastcall TComponent * FindAssociatedField();
```

Description

This is FindAssociatedField, a member of class TAdapterDataDisplay.

181.1.8.3.3 TAdapterDataDisplay.FormatCaption Method

Pascal

```
function FormatCaption: string; override;
```

C++

```
virtual __fastcall AnsiString FormatCaption();
```

Description

This is FormatCaption, a member of class TAdapterDataDisplay.

181.1.8.3.4 TAdapterDataDisplay.FormatColumnData Method

Pascal

```
function FormatColumnData(const Content: string; Options: TWebContentOptions): string;
```

C++

```
__fastcall AnsiString FormatColumnData(const AnsiString Content, TWebContentOptions Options);
```

Description

This is FormatColumnData, a member of class TAdapterDataDisplay.

181.1.8.3.5 TAdapterDataDisplay.FormatColumnHeading Method

Pascal

```
function FormatColumnHeading(Options: TWebContentOptions): string;
```

C++

```
__fastcall AnsiString FormatColumnHeading(TWebContentOptions Options);
```

Description

This is FormatColumnHeading, a member of class TAdapterDataDisplay.

181.1.8.3.6 TAdapterDataDisplay.GetAdapterModeOfAdapter Method

Pascal

```
function GetAdapterModeOfAdapter: string;
```

C++

```
__fastcall AnsiString GetAdapterModeOfAdapter();
```

Description

This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.

181.1.8.3.7 TAdapterDataDisplay.GetAdapterVariableName Method

Pascal

```
function GetAdapterVariableName: string;
```

C++

```
__fastcall AnsiString GetAdapterVariableName();
```

Description

This is GetAdapterVariableName, a member of class TAdapterDataDisplay.

181.1.8.3.8 TAdapterDataDisplay.GetCaption Method

Returns the value of the Caption property.

Pascal

```
function GetCaption: string; override;
```

C++

```
virtual __fastcall AnsiString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption property.

See Also

Caption (see page 16897)

181.1.8.3.9 TAdapterDataDisplay.GetControlRequirements Method

Pascal

```
procedure GetControlRequirements(ARequirements: TAbstractControlRequirements);
```

C++

```
__fastcall GetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is GetControlRequirements, a member of class TAdapterDataDisplay.

181.1.8.3.10 TAdapterDataDisplay.GetDesigntimeWarnings Method

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

This is GetDesigntimeWarnings, a member of class TAdapterDataDisplay.

181.1.8.3.11 TAdapterDataDisplay.GetFieldName Method

Pascal

```
function GetFieldName: string;
```

C++

```
__fastcall AnsiString GetFieldName();
```

Description

This is GetFieldName, a member of class TAdapterDataDisplay.

181.1.8.3.12 TAdapterDataDisplay.GetVariablesContainer Method

Pascal

```
function GetVariablesContainer: TComponent;
```

C++

```
__fastcall TComponent * GetVariablesContainer();
```

Description

This is GetVariablesContainer, a member of class TAdapterDataDisplay.

181.1.8.3.13 TAdapterDataDisplay.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML that defines the display control and its label. ImplContent formats the caption and combines this HTML with the results of the ControlContent method. ControlContent is an abstract (pure virtual) method in TWebDataDisplay. TWebDataDisplay descendants override it to generate the definition of their particular display control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebDataDisplay should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the display control.

See Also

Content (see page 16890)

Caption (see page 16897)

ILayoutWebContent (see page 24774)

181.1.8.3.14 TAdapterDataDisplay.ImplFormatColumnData Method

Pascal

```
function ImplFormatColumnData(const Content: string; Options: TWebContentOptions): string;
virtual;
```

C++

```
virtual __fastcall AnsiString ImplFormatColumnData(const AnsiString Content,
TWebContentOptions Options);
```

Description

This is ImplFormatColumnData, a member of class TAdapterDataDisplay.

181.1.8.3.15 TAdapterDataDisplay.ImplFormatColumnHeading Method

Pascal

```
function ImplFormatColumnHeading(Options: TWebContentOptions): string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplFormatColumnHeading(TWebContentOptions Options);
```

Description

This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.

181.1.8.3.16 TAdapterDataDisplay.ImplGetControlRequirements Method

Pascal

```
procedure ImplGetControlRequirements(ARequirements: TAbstractControlRequirements); virtual;
```

C++

```
virtual __fastcall ImplGetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.

181.1.8.3.17 TAdapterDataDisplay.ImplRestoreDefaults Method

Pascal

```
procedure ImplRestoreDefaults; virtual;
```

C++

```
virtual __fastcall ImplRestoreDefaults();
```

Description

This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.

181.1.8.3.18 TAdapterDataDisplay.RestoreDefaults Method

Pascal

```
procedure RestoreDefaults;
```

C++

```
__fastcall RestoreDefaults();
```

Description

This is RestoreDefaults, a member of class TAdapterDataDisplay.

181.1.8.3.19 TAdapterDataDisplay.SetCaption Method

Sets the value of the Caption property.

Pascal

```
procedure SetCaption(Value: string); override;
```

C++

```
virtual __fastcall SetCaption(AnsiString Value);
```

Description

SetCaption is the protected write implementation for the Caption property.

See Also

Caption (see page 16897)

181.1.8.3.20 TAdapterDataDisplay.SetFieldName Method

Pascal

```
procedure SetFieldName(const Value: string);
```

C++

```
__fastcall SetFieldName(const AnsiString Value);
```

Description

This is SetFieldName, a member of class TAdapterDataDisplay.

181.1.8.4 TAdapterDataDisplay Properties

The properties of the TAdapterDataDisplay class are listed here.

Public Properties

	Name	Description
	Align (see page 25097)	Specifies the horizontal alignment of the component relative to its containing cell.
	FieldName (see page 25098)	Specifies the name of the associated dataset field.
	HideOptions (see page 25098)	When field should be hidden.
	VAlign (see page 25098)	Specifies the vertical alignment of the generated display control.

181.1.8.4.1 TAdapterDataDisplay.Align Property

Specifies the horizontal alignment of the component relative to its containing cell.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align specifies the component's horizontal alignment relative to the containing HTML table cell.

See Also

VAlign (see page 25098)

181.1.8.4.2 TAdapterDataDisplay.FieldName Property

Specifies the name of the associated dataset field.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;
```

Description

FieldName specifies the name of the dataset field that the adapter field represents.

See Also

DataSet (see page 7807)

181.1.8.4.3 TAdapterDataDisplay.HideOptions Property

When field should be hidden.

Pascal

```
property HideOptions: TDisplayFieldHideOptions;
```

C++

```
__property TDisplayFieldHideOptions HideOptions;
```

Description

Set HideOptions to indicate when the field should be hidden from view.

181.1.8.4.4 TAdapterDataDisplay.VAlign Property

Specifies the vertical alignment of the generated display control.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign specifies the control's vertical alignment relative to the containing HTML table cell.

See Also

Align (see page 25097)

181.1.9 TAdapterDisplayColumn Class

TAdapterDisplayColumn displays an Adapter field in a grid.

Class Hierarchy

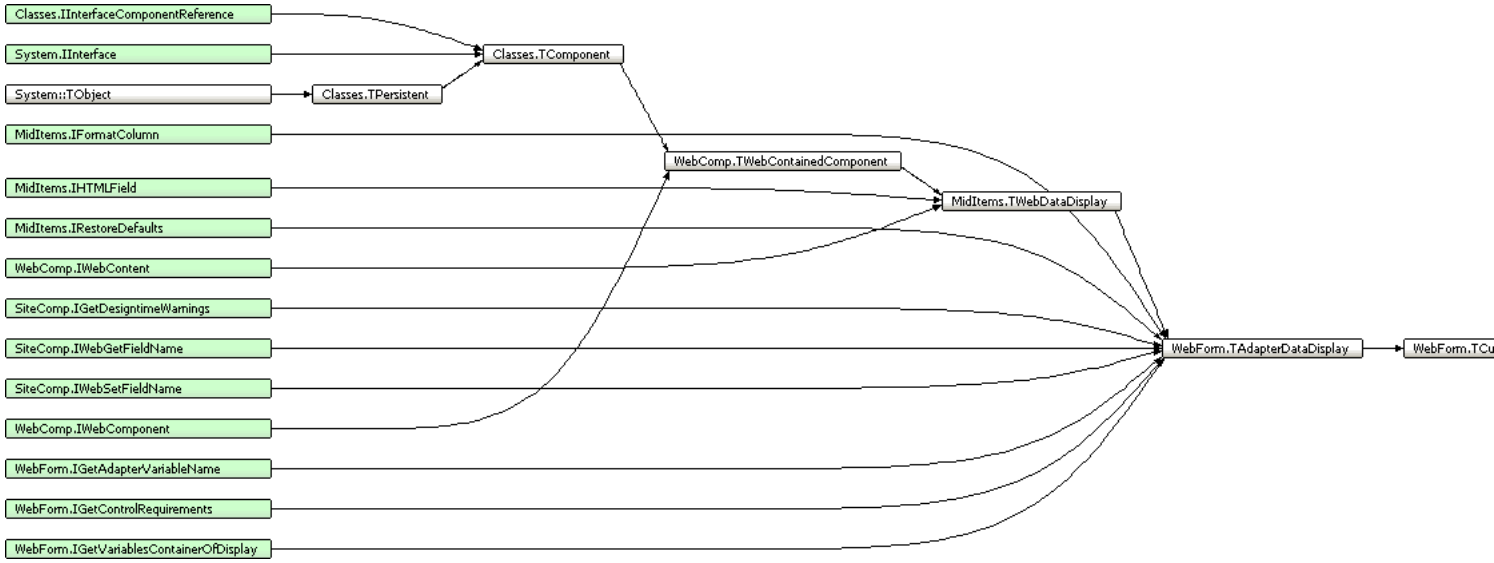

Pascal

```
TAdapterDisplayColumn = class(TCustomAdapterDisplayColumn);
```

C++

```
class TAdapterDisplayColumn : public TCustomAdapterDisplayColumn;
```

File

WebForm

Description

Add TAdapterDisplayColumn as a child of a TAdapterGrid (see page 25175) to display an Adapter field. The TAdapterGrid (see page 25175) specifies the Adapter used by TAdapterDisplayColumn.

Use the properties and methods of TAdapterDisplayColumn to:

Specify the Adapter field whose value the component displays.

Specify a caption for the column.

Specify when the column should be hidden.

Customize the appearance of the column.

TAdapterDisplayColumn supports multiple display presentation: text, list, and image. The display type can be set explicitly or TAdapterDisplayColumn will automatically choose a display type that is appropriate for the Adapter field.

TAdapterDisplayColumn is a minimal implementation of TCustomAdapterDisplayColumn (see page 25230). To extend this component, create your own concrete descendant of TCustomAdapterDisplayColumn (see page 25230).

See Also

TCustomAdapterDisplayColumn (see page 25230)

181.1.9.1 TAdapterDisplayColumn Members

The following tables list the members exposed by TAdapterDisplayColumn.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 25237)	This is CanAddClass, a member of class TCustomAdapterDisplayColumn.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 25237)	This is ControlContent, a member of class TCustomAdapterDisplayColumn.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	FieldConditions (see page 25092)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 25093)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	FormatCaption (see page 25093)	This is FormatCaption, a member of class TAdapterDataDisplay.
	FormatColumnData (see page 25093)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 25093)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	FormatImage (see page 25237)	This is FormatImage, a member of class TCustomAdapterDisplayColumn.
	FormatListDisplay (see page 25237)	This is FormatListDisplay, a member of class TCustomAdapterDisplayColumn.
	FormatTextDisplay (see page 25238)	This is FormatTextDisplay, a member of class TCustomAdapterDisplayColumn.
	GetAdapterModeOfAdapter (see page 25093)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 25094)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 25094)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetControlRequirements (see page 25094)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 25094)	This is GetDesigntimeWarnings, a member of class TAdapterDataDisplay.
	GetFieldName (see page 25095)	This is GetFieldName, a member of class TAdapterDataDisplay.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).

	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariablesContainer (see page 25095)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplCanAddClass (see page 25238)	This is ImplCanAddClass, a member of class TCustomAdapterDisplayColumn.
	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
	ImplContent (see page 25095)	Provides the underlying implementation for the Content method.
	ImplFormatColumnData (see page 25096)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
	ImplFormatColumnHeading (see page 25096)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
	ImplFormatColumnHeading (see page 25238)	This is ImplFormatColumnHeading, a member of class TCustomAdapterDisplayColumn.
	ImplGetControlRequirements (see page 25096)	This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.
	ImplGetControlRequirements (see page 25238)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayColumn.
	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
	ImplRestoreDefaults (see page 25096)	This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.
	ImplRestoreDefaults (see page 25239)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayColumn.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 25096)	This is RestoreDefaults, a member of class TAdapterDataDisplay.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
	SetCaption (see page 25097)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 25097)	This is SetFieldName, a member of class TAdapterDataDisplay.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	CaptionPosition (see page 16897)	Specifies where the label appears relative to the generated display control.
	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 16889)	Creates and initializes an instance of TWebDataDisplay (see page 16884).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
	Create (see page 25092)	Creates and initializes an instance of TWebDataDisplay.
	Create (see page 25236)	Creates and initializes an instance of TCustomAdapterDisplayColumn (see page 25230).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FormatColumnData (see page 17062)	This is FormatColumnData, a member of class IFormatColumn.
	FormatColumnHeading (see page 17062)	This is FormatColumnHeading, a member of class IFormatColumn.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterVariableName (see page 25307)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetControlRequirements (see page 25309)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 19697)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetVariablesContainer (see page 25311)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
V	IsColumn (see page 25239)	Indicates whether the component generates the controls that appear in the column of a data grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.

	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AdapterVariableName (see page 25308)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	TabIndex (see page 16899)	Determines the position of the generated control in the HTML form's tab order.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Align (see page 25105)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 25105)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 25105)	Customizes the appearance of the generated control's label.
	Custom (see page 25106)	Represents a tag option and its value in the tag for the display control.
	DisplayType (see page 25106)	Sets the display type for the column.
	FieldName (see page 25107)	Specifies the name of the associated dataset field.
	HideOptions (see page 25107)	When field should be hidden.
	ImageHeight (see page 25107)	Specifies the height of the image in the column.
	ImageWidth (see page 25107)	Specifies the width of the image in the column.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 25108)	Adds style attributes to the generated display control.
	StyleRule (see page 25108)	This is StyleRule, a member of class TAdapterDisplayColumn.
	Tag (see page 2808)	Stores an integer value as part of a component.
	VAlign (see page 25108)	Specifies the vertical alignment of the generated display control.

181.1.9.2 TAdapterDisplayColumn Properties

The properties of the TAdapterDisplayColumn class are listed here.

Published Properties

	Name	Description
	Align (see page 25105)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 25105)	Specifies the text of a label that indicates what information the generated control displays.

	CaptionAttributes (see page 25105)	Customizes the appearance of the generated control's label.
	Custom (see page 25106)	Represents a tag option and its value in the tag for the display control.
	DisplayType (see page 25106)	Sets the display type for the column.
	FieldName (see page 25107)	Specifies the name of the associated dataset field.
	HideOptions (see page 25107)	When field should be hidden.
	ImageHeight (see page 25107)	Specifies the height of the image in the column.
	ImageWidth (see page 25107)	Specifies the width of the image in the column.
	Style (see page 25108)	Adds style attributes to the generated display control.
	StyleRule (see page 25108)	This is StyleRule, a member of class TAdapterDisplayColumn.
	VAlign (see page 25108)	Specifies the vertical alignment of the generated display control.

181.1.9.2.1 TAdapterDisplayColumn.Align Property

Specifies the horizontal alignment of the component relative to its containing cell.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align specifies the component's horizontal alignment relative to the containing HTML table cell.

See Also

VAlign (see page 25098)

181.1.9.2.2 TAdapterDisplayColumn.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 16897)

CaptionPosition (see page 16897)

181.1.9.2.3 TAdapterDisplayColumn.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 16897)

CaptionPosition (see page 16897)

HeadingAttributes (see page 16509)

181.1.9.2.4 TAdapterDisplayColumn.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 16899)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

181.1.9.2.5 TAdapterDisplayColumn.DisplayType Property

Sets the display type for the column.

Pascal

```
property DisplayType: TColumnHTMLElementType;
```

C++

```
__property TColumnHTMLElementType DisplayType;
```

Description

Set DisplayType to set the display type for the column to text or image.

See Also

ImageHeight (see page 25240)

ImageWidth (see page 25240)

181.1.9.2.6 TAdapterDisplayColumn.FieldName Property

Specifies the name of the associated dataset field.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

FieldName specifies the name of the dataset field that the adapter field represents.

See Also

DataSet (see page 7807)

181.1.9.2.7 TAdapterDisplayColumn.HideOptions Property

When field should be hidden.

Pascal

```
property HideOptions: TDisplayFieldHideOptions;
```

C++

```
__property TDisplayFieldHideOptions HideOptions;
```

Description

Set HideOptions to indicate when the field should be hidden from view.

181.1.9.2.8 TAdapterDisplayColumn.ImageHeight Property

Specifies the height of the image in the column.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

Set ImageHeight to indicate the height of the image in pixels.

See Also

ImageWidth (see page 25240)

181.1.9.2.9 TAdapterDisplayColumn.ImageWidth Property

Specifies the width of the image in the column.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

Set ImageHeight to indicate the width of the image in pixels.

See Also

ImageHeight (see page 25240)

181.1.9.2.10 TAdapterDisplayColumn.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 16898)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

Setting Web Item Properties

181.1.9.2.11 TAdapterDisplayColumn.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TAdapterDisplayColumn.

181.1.9.2.12 TAdapterDisplayColumn.VAlign Property

Specifies the vertical alignment of the generated display control.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign specifies the control's vertical alignment relative to the containing HTML table cell.

See Also

Align (see page 25097)

181.1.10 TAdapterDisplayField Class

TAdapterDisplayField displays an Adapter field as a form field.

Class Hierarchy

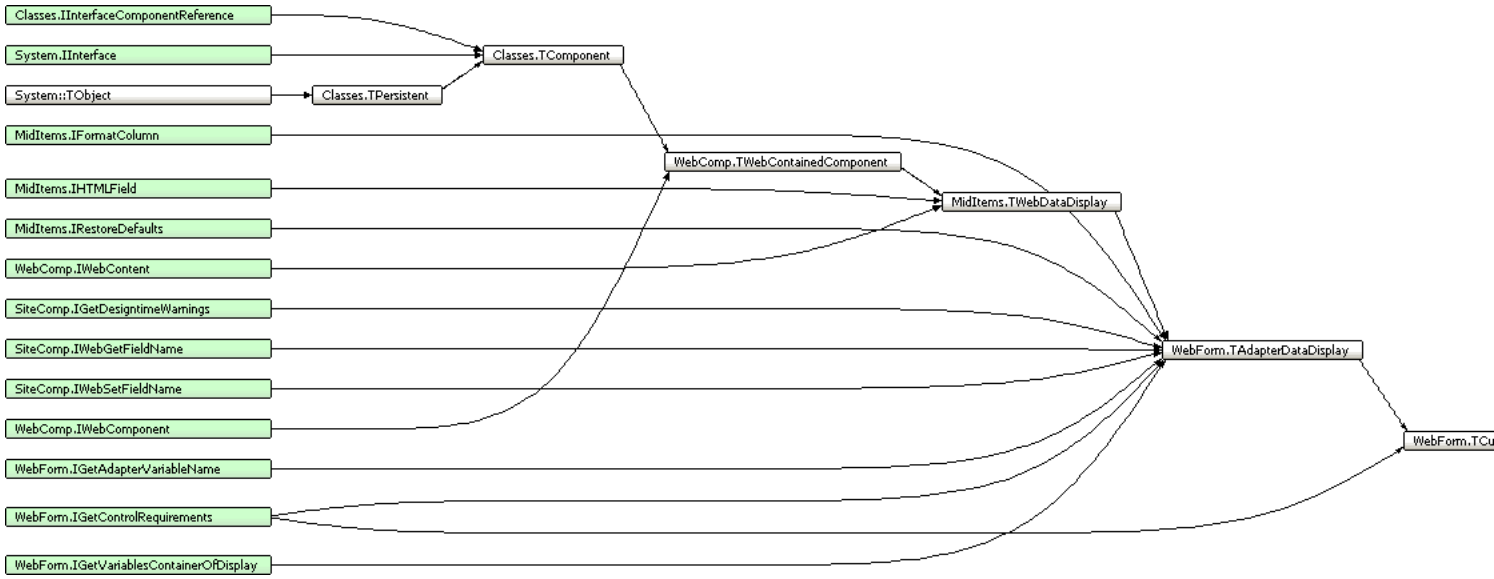

Pascal

```
TAdapterDisplayField = class(TCustomAdapterDisplayField);
```

C++

```
class TAdapterDisplayField : public TCustomAdapterDisplayField;
```

File

WebForm

Description

Add TAdapterDisplayField as a child of a TAdapterFieldGroup (see page 25150) to display an Adapter field. The TAdapterFieldGroup (see page 25150) specifies the Adapter used by TAdapterDisplayField.

Use the properties and methods of TAdapterDisplayField to:

Specify the Adapter field whose value the component displays.

Specify a caption for the field.

Specify when the field should be hidden.

Customize the appearance of the column.

Specify whether the field is used for input or display only.

TAdapterDisplayField supports multiple display type: text, list, and image. The display type can be set explicitly or TAdapterDisplayField will automatically choose a display type that is appropriate for the Adapter field.

TAdapterDisplayField supports multiple input types: Text, Password, Select, MultipleSelect, Radio, CheckBox, TextArea, and File. The input type can be set explicitly or TAdapterDisplayField will automatically choose an input type that is appropriate for the Adapter field.

TAdapterDisplayField instances are not used in grids. TAdapterEditColumn (see page 25132) provides equivalent functionality in a grid context.

TAdapterDisplayField is a minimal implementation of TCustomAdapterDisplayField (see page 25240). To extend the features of this component, define your own descendant of TCustomAdapterDisplayField (see page 25240).

See Also

TAdapterEditColumn (see page 25132)

181.1.10.1 TAdapterDisplayField Members

The following tables list the members exposed by TAdapterDisplayField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAttributes (see page 25247)	This is AddAttributes, a member of class TCustomAdapterDisplayField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 25247)	This is ControlContent, a member of class TCustomAdapterDisplayField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	FieldConditions (see page 25092)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 25093)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	FormatCaption (see page 25093)	This is FormatCaption, a member of class TAdapterDataDisplay.
	FormatColumnData (see page 25093)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 25093)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	GetAdapterModeOfAdapter (see page 25093)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 25094)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 25094)	Returns the value of the Caption property.

	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetControlRequirements (see page 25094)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 25094)	This is GetDesigntimeWarnings, a member of class TAdapterDataDisplay.
	GetFieldName (see page 25095)	This is GetFieldName, a member of class TAdapterDataDisplay.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetLayoutAttributes (see page 25247)	Returns the value of the LayoutAttributes property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariablesContainer (see page 25095)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
V	ImplContent (see page 25095)	Provides the underlying implementation for the Content method.
V	ImplContent (see page 25247)	Provides the underlying implementation for the Content method.
V	ImplFormatColumnData (see page 25096)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
V	ImplFormatColumnHeading (see page 25096)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
V	ImplGetControlRequirements (see page 25096)	This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.
V	ImplGetControlRequirements (see page 25248)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayField.
V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
V	ImplRestoreDefaults (see page 25096)	This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.
V	ImplRestoreDefaults (see page 25248)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 25096)	This is RestoreDefaults, a member of class TAdapterDataDisplay.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
V	SetCaption (see page 25097)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.

	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 25097)	This is SetFieldName, a member of class TAdapterDataDisplay.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16889)	Creates and initializes an instance of TWebDataDisplay (see page 16884).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	Create (see page 25092)	Creates and initializes an instance of TWebDataDisplay.
V	Create (see page 25246)	Creates and initializes an instance of TCustomAdapterDisplayField (see page 25240).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.

	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FormatColumnData (see page 17062)	This is FormatColumnData, a member of class IFormatColumn.
	FormatColumnHeading (see page 17062)	This is FormatColumnHeading, a member of class IFormatColumn.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterVariableName (see page 25307)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetControlRequirements (see page 25309)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 19697)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetVariablesContainer (see page 25311)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AdapterVariableName (see page 25308)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	TabIndex (see page 16899)	Determines the position of the generated control in the HTML form's tab order.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Align (see page 25115)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 25116)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 25116)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 25116)	Specifies where the label appears relative to the generated display control.
	Custom (see page 25117)	Represents a tag option and its value in the tag for the display control.
	DisplayColumns (see page 25117)	Specifies the number of columns in a display field.
	DisplayRows (see page 25117)	Specifies the number of rows in a display field.
	DisplayType (see page 25118)	Sets the display type for the column.
	DisplayWidth (see page 25118)	Specifies the width, in characters, of the generated HTML field.
	FieldName (see page 25118)	Specifies the name of the associated dataset field.
	HideOptions (see page 25119)	When field should be hidden.
	ImageHeight (see page 25119)	Specifies the height of the image in the column.
	ImageWidth (see page 25119)	Specifies the width of the image in the column.
	InputType (see page 25120)	Specifies the input type for the display field.
	MaxLength (see page 25120)	Specifies the maximum number of characters the user can enter into the display field.

	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SelectRows (see page 25120)	Indicates the number of rows that can be selected.
	Style (see page 25121)	Adds style attributes to the generated display control.
	StyleRule (see page 25121)	This is StyleRule, a member of class TAdapterDisplayField.
	Tag (see page 2808)	Stores an integer value as part of a component.
	TextAreaWrap (see page 25121)	Specifies how the display field treats lines that are longer than the display width.
	VAlign (see page 25122)	Specifies the vertical alignment of the generated display control.
	ViewMode (see page 25122)	Indicates the viewing state of the adapter field.

181.1.10.2 TAdapterDisplayField Properties

The properties of the TAdapterDisplayField class are listed here.

Published Properties

	Name	Description
	Align (see page 25115)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 25116)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 25116)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 25116)	Specifies where the label appears relative to the generated display control.
	Custom (see page 25117)	Represents a tag option and its value in the tag for the display control.
	DisplayColumns (see page 25117)	Specifies the number of columns in a display field.
	DisplayRows (see page 25117)	Specifies the number of rows in a display field.
	DisplayType (see page 25118)	Sets the display type for the column.
	DisplayWidth (see page 25118)	Specifies the width, in characters, of the generated HTML field.
	FieldName (see page 25118)	Specifies the name of the associated dataset field.
	HideOptions (see page 25119)	When field should be hidden.
	ImageHeight (see page 25119)	Specifies the height of the image in the column.
	ImageWidth (see page 25119)	Specifies the width of the image in the column.
	InputType (see page 25120)	Specifies the input type for the display field.
	MaxLength (see page 25120)	Specifies the maximum number of characters the user can enter into the display field.
	SelectRows (see page 25120)	Indicates the number of rows that can be selected.
	Style (see page 25121)	Adds style attributes to the generated display control.
	StyleRule (see page 25121)	This is StyleRule, a member of class TAdapterDisplayField.
	TextAreaWrap (see page 25121)	Specifies how the display field treats lines that are longer than the display width.
	VAlign (see page 25122)	Specifies the vertical alignment of the generated display control.
	ViewMode (see page 25122)	Indicates the viewing state of the adapter field.

181.1.10.2.1 TAdapterDisplayField.Align Property

Specifies the horizontal alignment of the component relative to its containing cell.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align specifies the component's horizontal alignment relative to the containing HTML table cell.

See Also

VAlign (see page 25098)

181.1.10.2.2 TAdapterDisplayField.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 16897)

CaptionPosition (see page 16897)

181.1.10.2.3 TAdapterDisplayField.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 16897)

CaptionPosition (see page 16897)

HeadingAttributes (see page 16509)

181.1.10.2.4 TAdapterDisplayField.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 16897)

Caption (see page 16897)

181.1.10.2.5 TAdapterDisplayField.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 16899)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

181.1.10.2.6 TAdapterDisplayField.DisplayColumns Property

Specifies the number of columns in a display field.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Use DisplayColumns to control the number of columns being displayed.

See Also

DisplayRows (see page 25249)

181.1.10.2.7 TAdapterDisplayField.DisplayRows Property

Specifies the number of rows in a display field.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayColumns to control the number of rows being displayed.

See Also

DisplayColumns (see page 25249)

181.1.10.2.8 TAdapterDisplayField.DisplayType Property

Sets the display type for the column.

Pascal

```
property DisplayType: TDisplayFieldHTMLElementType;
```

C++

```
__property TDisplayFieldHTMLElementType DisplayType;
```

Description

Set DisplayType to indicate the type of HTML control that should be used to display this field's value when it is read-only.

See Also

ImageHeight (see page 25250)

ImageWidth (see page 25250)

InputType (see page 25251)

181.1.10.2.9 TAdapterDisplayField.DisplayWidth Property

Specifies the width, in characters, of the generated HTML field.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to control the width of the HTML field that displays record status information. By default, DisplayWidth is 1 because the TCustomAdapterDisplayField control values are one character (M for modified or I for inserted).

181.1.10.2.10 TAdapterDisplayField.FieldName Property

Specifies the name of the associated dataset field.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

FieldName specifies the name of the dataset field that the adapter field represents.

See Also

DataSet (see page 7807)

181.1.10.2.11 TAdapterDisplayField.HideOptions Property

When field should be hidden.

Pascal

```
property HideOptions: TDisplayFieldHideOptions;
```

C++

```
__property TDisplayFieldHideOptions HideOptions;
```

Description

Set HideOptions to indicate when the field should be hidden from view.

181.1.10.2.12 TAdapterDisplayField.ImageHeight Property

Specifies the height of the image in the column.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

Set ImageHeight to indicate the height of the image in the column.

See Also

DisplayWidth (see page 25250)

ImageWidth (see page 25250)

181.1.10.2.13 TAdapterDisplayField.ImageWidth Property

Specifies the width of the image in the column.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

Set ImageHeight to indicate the width of the image in the column.

See Also

ImageHeight (see page 25250)

181.1.10.2.14 TAdapterDisplayField.InputType Property

Specifies the input type for the display field.

Pascal

```
property InputType: TInputFieldHTMLElementType;
```

C++

```
__property TInputFieldHTMLElementType InputType;
```

Description

Set InputType to indicate the type of control that should be generated to let users assign values to the adapter field.

See Also

ImageHeight (see page 25250)

ImageWidth (see page 25250)

DisplayType (see page 25249)

181.1.10.2.15 TAdapterDisplayField.MaxLength Property

Specifies the maximum number of characters the user can enter into the display field.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the display field. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

181.1.10.2.16 TAdapterDisplayField.SelectRows Property

Indicates the number of rows that can be selected.

Pascal

```
property SelectRows: Integer;
```

C++

```
__property int SelectRows;
```

Description

Use SelectRows to indicate the number of rows that can be selected.

181.1.10.2.17 TAdapterDisplayField.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 16898)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

Setting Web Item Properties

181.1.10.2.18 TAdapterDisplayField.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TAdapterDisplayField.

181.1.10.2.19 TAdapterDisplayField.TextAreaWrap Property

Specifies how the display field treats lines that are longer than the display width.

Pascal

```
property TextAreaWrap: TTextAreaWrap;
```

C++

```
__property TTextAreaWrap TextAreaWrap;
```

Description

Use TextAreaWrap to specify how the generated display field formats lines that are longer than the number of characters specified by DisplayWidth.

See Also

DisplayWidth (see page 25250)

181.1.10.2.20 TAdapterDisplayField.VAlign Property

Specifies the vertical alignment of the generated display control.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign specifies the control's vertical alignment relative to the containing HTML table cell.

See Also

Align (see page 25097)

181.1.10.2.21 TAdapterDisplayField.ViewMode Property

Indicates the viewing state of the adapter field.

Pascal

```
property ViewMode: TDisplayFieldViewMode;
```

C++

```
__property TDisplayFieldViewMode ViewMode;
```

Description

Use ViewMode to specify the viewing state of the field.

181.1.11 TAdapterDisplayReferenceGroup Class

TAdapterDisplayReferenceGroup is the base class for components that generate HTML controls that refer to a control which displays data from an adapter.

Class Hierarchy

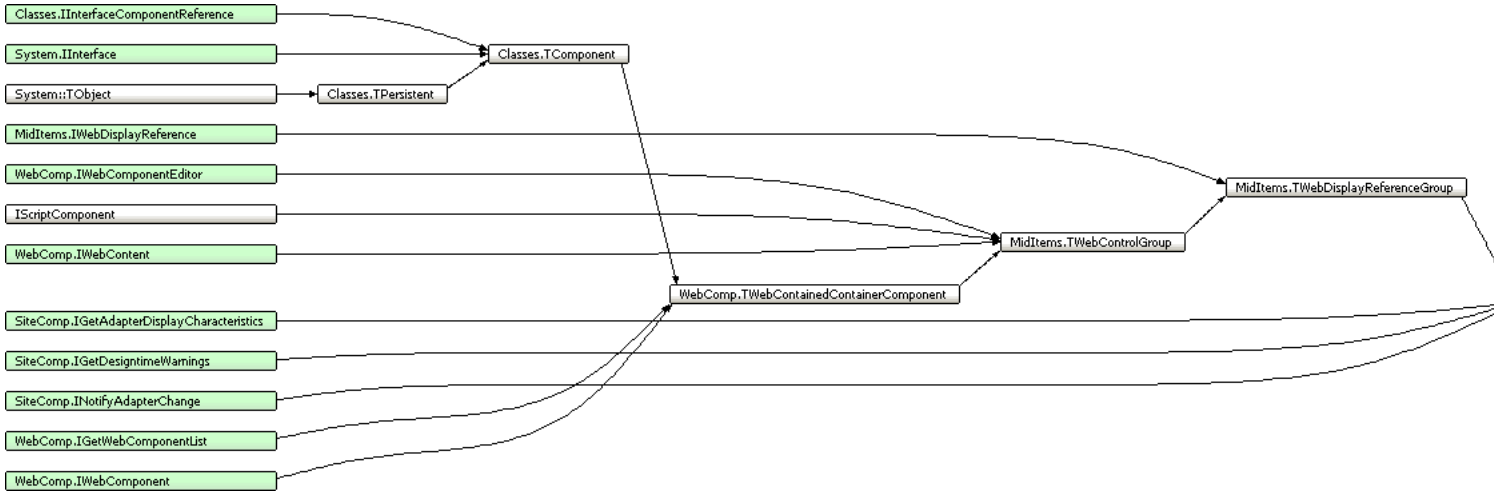

Pascal

```
TAdapterDisplayReferenceGroup = class(TWebDisplayReferenceGroup, IGetDesigntimeWarnings,
INotifyAdapterChange, IGetAdapterDisplayCharacteristics);
```

C++

```
class TAdapterDisplayReferenceGroup : public TWebDisplayReferenceGroup, public
IGetDesigntimeWarnings, public INotifyAdapterChange, public IGetAdapterDisplayCharacteristics;
```

File

WebForm

Description

Use TAdapterDisplayReferenceGroup as a base class for Web items that generate the HTML to arrange a collection of controls. The TAdapterDisplayReferenceGroup is associated with another component that generates HTML elements to display the data from an adapter. This component is the value of the DisplayComponent (see page 25132) property. The child Web items of TAdapterDisplayReferenceGroup generate the individual controls in the collection this class arranges. Each of those controls operates on the display element produced by the component that is the value of the DisplayComponent (see page 25132) property.

Do not create instances of TAdapterDisplayReferenceGroup. TAdapterDisplayReferenceGroup only manages the Web items that generate controls, the association to a display generator, and the logic for working with the WebSnap editors in the IDE. It does not include any special logic for working with the display generator's adapter. To add a component that organizes controls in an adapter form, use a descendant of TAdapterDisplayReferenceGroup such as TAdapterCommandGroup (see page 25062).

See Also

TAdapterCommandGroup (see page 25062)

181.1.11.1 TAdapterDisplayReferenceGroup Members

The following tables list the members exposed by TAdapterDisplayReferenceGroup.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AdapterChange (see page 25128)	Responds to changes in the adapter on which the controls in this group operate.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DisplayComponentChange (see page 25129)	Responds when the DisplayComponent (see page 25132) property changes.
	GetAdapterDisplayCharacteristics (see page 25129)	Indicates the display characteristics of the HTML this component generates.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetDesigntimeWarnings (see page 25130)	Populates a design-time warnings object with any warnings that the reference group needs to emit.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	GetWebDisplayComponent (see page 16920)	This is GetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
V	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
V	ImplCanAddClass (see page 16881)	Provides the underlying implementation for the CanAddClass (see page 16878) method.
V	ImplContent (see page 16882)	Provides the underlying implementation for the Content (see page 16879) method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16920)	Forwards notification messages to all owned components.
	NotifyAdapterChange (see page 25130)	Responds when other components notify this one of changes to the adapter.
	OnDisplayComponentChange (see page 25131)	Responds when the associated display component changes.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
V	SetWebDisplayComponent (see page 25131)	Sets the value of the DisplayComponent (see page 25132) property.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebDisplay (see page 16921)	This is WebDisplay, a member of class TWebDisplayReferenceGroup.
	WebDisplayComponent (see page 16921)	
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 25127)	Creates an instance of TWebDisplayReferenceGroup.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25128)	Frees an instance of TAdapterDisplayReferenceGroup (see page 25122).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	GetWebDisplayComponent (see page 17081)	This is GetWebDisplayComponent, a member of class IWebDisplayReference.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayComponent (see page 25132)	Associates the group with a component that generates the display controls.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebDisplayComponent (see page 17081)	
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

181.1.11.2 TAdapterDisplayReferenceGroup.Create Constructor

Creates an instance of TWebDisplayReferenceGroup.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TAdapterDisplayReferenceGroup(TComponent * AOwner);
```

Description

Call TWebDisplayReferenceGroup to instantiate TWebDisplayReferenceGroup at runtime. TWebDisplayReferenceGroup components added at design time are instantiated automatically.

181.1.11.3 TAdapterDisplayReferenceGroup.Destroy Destructor

Frees an instance of TAdapterDisplayReferenceGroup (see page 25122).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterDisplayReferenceGroup();
```

Description

Do not call Destroy directly. Instead, call the Free method, which checks whether the reference to the producer is not nil before calling Destroy.

See Also

Free (see page 22300)

181.1.11.4 TAdapterDisplayReferenceGroup Methods

The methods of the TAdapterDisplayReferenceGroup class are listed here.

Protected Methods

	Name	Description
	AdapterChange (see page 25128)	Responds to changes in the adapter on which the controls in this group operate.
	DisplayComponentChange (see page 25129)	Responds when the DisplayComponent (see page 25132) property changes.
	GetAdapterDisplayCharacteristics (see page 25129)	Indicates the display characteristics of the HTML this component generates.
	GetDesigntimeWarnings (see page 25130)	Populates a design-time warnings object with any warnings that the reference group needs to emit.
	NotifyAdapterChange (see page 25130)	Responds when other components notify this one of changes to the adapter.
	OnDisplayComponentChange (see page 25131)	Responds when the associated display component changes.
	SetWebDisplayComponent (see page 25131)	Sets the value of the DisplayComponent (see page 25132) property.

181.1.11.4.1 TAdapterDisplayReferenceGroup.AdapterChange Method

Responds to changes in the adapter on which the controls in this group operate.

Pascal

```
procedure AdapterChange; virtual;
```

C++

```
virtual __fastcall AdapterChange();
```

Description

TAdapterDisplayReferenceGroup (see page 25122) calls AdapterChange internally whenever the adapter associated with its display component changes. This happens in response to the following occurrences:

The component specified by DisplayComponent (see page 25132) notifies this one that it has changed.

Another component notifies this one of a change to the adapter by calling the NotifyAdapterChange (see page 25130) method.

AdapterChange responds to the change by regenerating its list of default child Web items and passing the notification on to every child Web item in its WebFieldControls property list.

See Also

NotifyAdapterChange (see page 25130)

DisplayComponent (see page 25132)

DisplayComponentChange (see page 25129)

WebFieldControls (see page 16884)

181.1.11.4.2 TAdapterDisplayReferenceGroup.DisplayComponentChange Method

Responds when the DisplayComponent (see page 25132) property changes.

Pascal

```
procedure DisplayComponentChange; virtual;
```

C++

```
virtual __fastcall DisplayComponentChange();
```

Description

The component specified by the DisplayComponent (see page 25132) property calls DisplayComponentChange when its properties change. This allows the TAdapterDisplayReferenceGroup (see page 25122) object to update itself to reflect any changes.

In TAdapterDisplayReferenceGroup (see page 25122), DisplayComponentChange calls the AdapterChange (see page 25128) method, to ensure that its Web items are up-to-date in case the change was to the associated adapter.

See Also

AdapterChange (see page 25128)

DisplayComponent (see page 25132)

181.1.11.4.3 TAdapterDisplayReferenceGroup.GetAdapterDisplayCharacteristics Method

Indicates the display characteristics of the HTML this component generates.

Pascal

```
function GetAdapterDisplayCharacteristics: TAdapterDisplayCharacteristics;
```

C++

```
__fastcall TAdapterDisplayCharacteristics GetAdapterDisplayCharacteristics();
```

Description

Adapter actions call GetAdapterDisplayCharacteristics from their IsDefaultAction methods to determine whether their function should be included in the adapter's default set of actions. If the adapter action is appropriate to operate on an HTML form with the indicated characteristics, the adapter action returns true from its IsDefaultAction method.

In TAdapterDisplayReferenceGroup (see page 25122), GetAdapterDisplayCharacteristics delegates this task by fetching the display characteristics from the associated display component.

See Also

IsDefaultAction (see page 24070)

DisplayComponent (see page 25132)

181.1.11.4.4 TAdapterDisplayReferenceGroup.GetDesigntimeWarnings Method

Populates a design-time warnings object with any warnings that the reference group needs to emit.

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

GetDesigntimeWarnings is called automatically at design time when the adapter page producer is assembling the current version of its HTML form. This gives the display reference group an opportunity to add messages to the version of the HTML form that appears at design time, describing problems that you can rectify.

AWarnings is the design-time warnings object that assembles design-time warning messages that appear in the HTML form.

If the DisplayComponent (see page 25132) property has not been set, GetDesigntimeWarnings adds a message that indicates this omission. Otherwise, GetDesigntimeWarnings adds the display component to the list of objects that can supply design-time warnings.

See Also

DisplayComponent (see page 25132)

TAdapterPageProducer (see page 19802)

181.1.11.4.5 TAdapterDisplayReferenceGroup.NotifyAdapterChange Method

Responds when other components notify this one of changes to the adapter.

Pascal

```
procedure NotifyAdapterChange(Sender: TComponent);
```

C++

```
__fastcall NotifyAdapterChange(TComponent * Sender);
```

Description

Other components in the WebSnap application call NotifyAdapterChange when they detect possible changes to the adapter on which this group's controls operate.

Sender is the component that detected the possible change and sent the notification.

NotifyAdapterChange calls the AdapterChange (see page 25128) method, which responds to the notification.

See Also

AdapterChange (see page 25128)

DisplayComponent (see page 25132)

181.1.11.4.6 TAdapterDisplayReferenceGroup.OnDisplayComponentChange Method

Responds when the associated display component changes.

Pascal

```
procedure OnDisplayComponentChange(ASender: TObject);
```

C++

```
__fastcall OnDisplayComponentChange(TObject * ASender);
```

Description

The component specified by the DisplayComponent (see page 25132) property calls OnDisplayComponentChange to notify this component of changes to its properties or child Web items.

ASender is the display component that calls this method.

OnDisplayComponentChange calls the DisplayComponentChange (see page 25129) method, which responds to the change.

See Also

DisplayComponentChange (see page 25129)

DisplayComponent (see page 25132)

181.1.11.4.7 TAdapterDisplayReferenceGroup.SetWebDisplayComponent Method

Sets the value of the DisplayComponent (see page 25132) property.

Pascal

```
procedure SetWebDisplayComponent(const AComponent: TComponent); override;
```

C++

```
virtual __fastcall SetWebDisplayComponent(const TComponent * AComponent);
```

Description

SetWebDisplayComponent is the protected write implementation of the DisplayComponent (see page 25132) property.

AComponent is the new value of the DisplayComponent (see page 25132) property.

See Also

DisplayComponent (see page 25132)

181.1.11.5 TAdapterDisplayReferenceGroup Properties

The properties of the TAdapterDisplayReferenceGroup class are listed here.

Public Properties

	Name	Description
	DisplayComponent (see page 25132)	Associates the group with a component that generates the display controls.

181.1.11.5.1 TAdapterDisplayReferenceGroup.DisplayComponent Property

Associates the group with a component that generates the display controls.

Pascal

```
property DisplayComponent: TComponent;
```

C++

```
__property TComponent * DisplayComponent;
```

Description

Use DisplayComponent to associate the group of controls with a Web item that generates the display elements those controls influence. Typical values of DisplayComponent are TAdapterGrid (see page 25175) or TAdapterFieldGroup (see page 25150).

TAdapterDisplayReferenceGroup (see page 25122) descendants use the value of DisplayComponent to access the display controls to influence, and to access the adapter, whose information is displayed or manipulated.

See Also

TAdapterGrid (see page 25175)

TAdapterFieldGroup (see page 25150)

181.1.12 TAdapterEditColumn Class

TAdapterEditColumn displays an adapter field as a form field in a grid.

Class Hierarchy

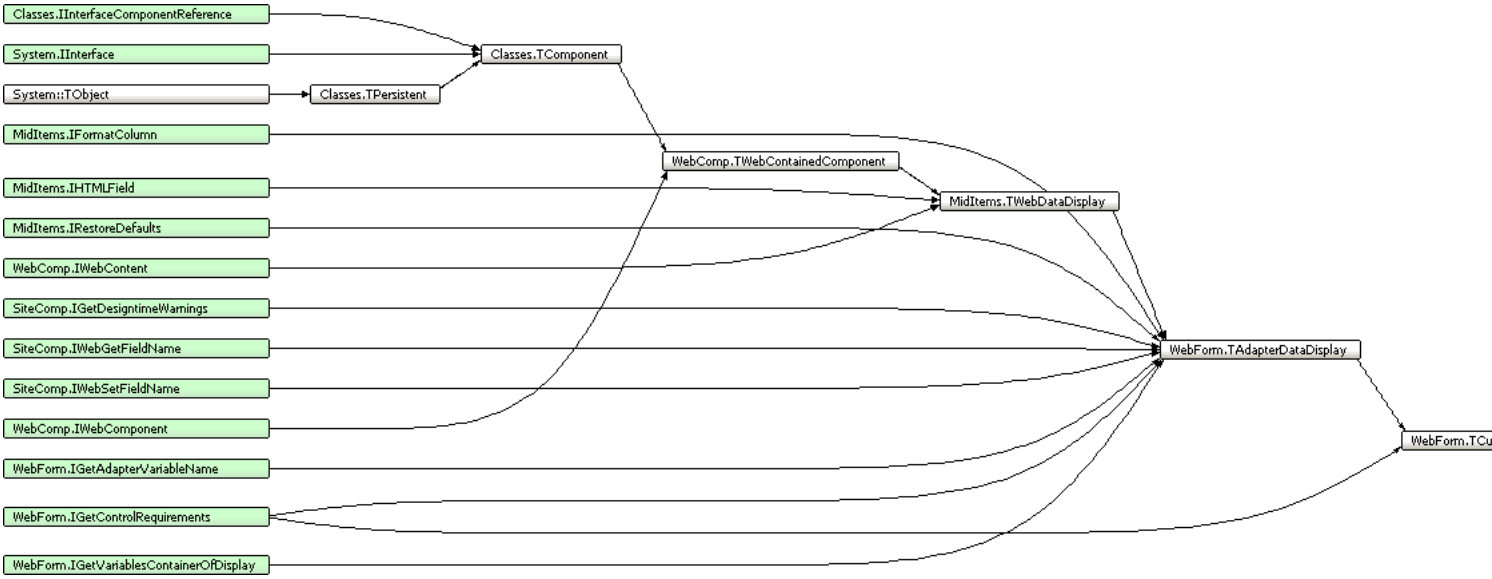

Pascal

```
TAdapterEditColumn = class(TCustomAdapterEditColumn);
```

C++

```
class TAdapterEditColumn : public TCustomAdapterEditColumn;
```

File

WebForm

Description

TAdapterEditColumn has the same functionality as TAdapterDisplayField (see page 25109), but is designed to work with grids. TAdapterEditColumn is a minimal implementation of TCustomAdapterEditColumn (see page 25252).

See Also

TCustomAdapterEditColumn (see page 25252)

TAdapterDisplayField (see page 25109)

181.1.12.1 TAdapterEditColumn Members

The following tables list the members exposed by TAdapterEditColumn.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAttributes (see page 25247)	This is AddAttributes, a member of class TCustomAdapterDisplayField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
A	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
V	ControlContent (see page 25247)	This is ControlContent, a member of class TCustomAdapterDisplayField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	FieldConditions (see page 25092)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 25093)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	FormatCaption (see page 25093)	This is FormatCaption, a member of class TAdapterDataDisplay.
	FormatColumnData (see page 25093)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 25093)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	GetAdapterModeOfAdapter (see page 25093)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 25094)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 25094)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.

	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetControlRequirements (see page 25094)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 25094)	This is GetDesigntimeWarnings, a member of class TAdapterDataDisplay.
	GetFieldName (see page 25095)	This is GetFieldName, a member of class TAdapterDataDisplay.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetLayoutAttributes (see page 25247)	Returns the value of the LayoutAttributes property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariablesContainer (see page 25095)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
V	ImplContent (see page 25095)	Provides the underlying implementation for the Content method.
V	ImplContent (see page 25247)	Provides the underlying implementation for the Content method.
V	ImplContent (see page 25258)	Provides the underlying implementation for the Content method.
V	ImplFormatColumnData (see page 25096)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
V	ImplFormatColumnHeading (see page 25096)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
V	ImplGetControlRequirements (see page 25096)	This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.
V	ImplGetControlRequirements (see page 25248)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayField.
V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
V	ImplRestoreDefaults (see page 25096)	This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.
V	ImplRestoreDefaults (see page 25248)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 25096)	This is RestoreDefaults, a member of class TAdapterDataDisplay.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
V	SetCaption (see page 25097)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetFieldName (see page 25097)	This is SetFieldName, a member of class TAdapterDataDisplay.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16889)	Creates and initializes an instance of TWebDataDisplay (see page 16884).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	Create (see page 25092)	Creates and initializes an instance of TWebDataDisplay.
V	Create (see page 25246)	Creates and initializes an instance of TCustomAdapterDisplayField (see page 25240).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).

	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FormatColumnData (see page 17062)	This is FormatColumnData, a member of class IFormatColumn.
	FormatColumnHeading (see page 17062)	This is FormatColumnHeading, a member of class IFormatColumn.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterVariableName (see page 25307)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetControlRequirements (see page 25309)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 19697)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetVariablesContainer (see page 25311)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.

	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AdapterVariableName (see page 25308)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	TabIndex (see page 16899)	Determines the position of the generated control in the HTML form's tab order.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Align (see page 25138)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 25139)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 25139)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 25139)	Specifies where the label appears relative to the generated display control.
	Custom (see page 25140)	Represents a tag option and its value in the tag for the display control.
	DisplayColumns (see page 25140)	Specifies the number of columns in a display field.
	DisplayRows (see page 25140)	Specifies the number of rows in a display field.
	DisplayType (see page 25141)	Sets the display type for the column.
	DisplayWidth (see page 25141)	Specifies the width, in characters, of the generated HTML field.
	FieldName (see page 25141)	Specifies the name of the associated dataset field.
	HideOptions (see page 25142)	When field should be hidden.
	ImageHeight (see page 25142)	Specifies the height of the image in the column.
	ImageWidth (see page 25142)	Specifies the width of the image in the column.
	InputType (see page 25143)	Specifies the input type for the display field.
	MaxLength (see page 25143)	Specifies the maximum number of characters the user can enter into the display field.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	SelectRows (see page 25143)	Indicates the number of rows that can be selected.

	Style (see page 25144)	Adds style attributes to the generated display control.
	StyleRule (see page 25144)	This is StyleRule, a member of class TAdapterEditColumn.
	Tag (see page 2808)	Stores an integer value as part of a component.
	TextAreaWrap (see page 25144)	Specifies how the display field treats lines that are longer than the display width.
	VAlign (see page 25145)	Specifies the vertical alignment of the generated display control.
	ViewMode (see page 25145)	Indicates the viewing state of the adapter field.

181.1.12.2 TAdapterEditColumn Properties

The properties of the TAdapterEditColumn class are listed here.

Published Properties

	Name	Description
	Align (see page 25138)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 25139)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 25139)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 25139)	Specifies where the label appears relative to the generated display control.
	Custom (see page 25140)	Represents a tag option and its value in the tag for the display control.
	DisplayColumns (see page 25140)	Specifies the number of columns in a display field.
	DisplayRows (see page 25140)	Specifies the number of rows in a display field.
	DisplayType (see page 25141)	Sets the display type for the column.
	DisplayWidth (see page 25141)	Specifies the width, in characters, of the generated HTML field.
	FieldName (see page 25141)	Specifies the name of the associated dataset field.
	HideOptions (see page 25142)	When field should be hidden.
	ImageHeight (see page 25142)	Specifies the height of the image in the column.
	ImageWidth (see page 25142)	Specifies the width of the image in the column.
	InputType (see page 25143)	Specifies the input type for the display field.
	MaxLength (see page 25143)	Specifies the maximum number of characters the user can enter into the display field.
	SelectRows (see page 25143)	Indicates the number of rows that can be selected.
	Style (see page 25144)	Adds style attributes to the generated display control.
	StyleRule (see page 25144)	This is StyleRule, a member of class TAdapterEditColumn.
	TextAreaWrap (see page 25144)	Specifies how the display field treats lines that are longer than the display width.
	VAlign (see page 25145)	Specifies the vertical alignment of the generated display control.
	ViewMode (see page 25145)	Indicates the viewing state of the adapter field.

181.1.12.2.1 TAdapterEditColumn.Align Property

Specifies the horizontal alignment of the component relative to its containing cell.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align specifies the component's horizontal alignment relative to the containing HTML table cell.

See Also

VAlign (see page 25098)

181.1.12.2.2 TAdapterEditColumn.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes (see page 16897)

CaptionPosition (see page 16897)

181.1.12.2.3 TAdapterEditColumn.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption (see page 16897)

CaptionPosition (see page 16897)

HeadingAttributes (see page 16509)

181.1.12.2.4 TAdapterEditColumn.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition: TCaptionPosition;
```

C++

```
__property TCaptionPosition CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes (see page 16897)

Caption (see page 16897)

181.1.12.2.5 TAdapterEditColumn.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style (see page 16899)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

181.1.12.2.6 TAdapterEditColumn.DisplayColumns Property

Specifies the number of columns in a display field.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Use DisplayColumns to control the number of columns being displayed.

See Also

DisplayRows (see page 25249)

181.1.12.2.7 TAdapterEditColumn.DisplayRows Property

Specifies the number of rows in a display field.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayColumns to control the number of rows being displayed.

See Also

DisplayColumns (see page 25249)

181.1.12.2.8 TAdapterEditColumn.DisplayType Property

Sets the display type for the column.

Pascal

```
property DisplayType: TDisplayFieldHTMLElementType;
```

C++

```
__property TDisplayFieldHTMLElementType DisplayType;
```

Description

Set DisplayType to indicate the type of HTML control that should be used to display this field's value when it is read-only.

See Also

ImageHeight (see page 25250)

ImageWidth (see page 25250)

InputType (see page 25251)

181.1.12.2.9 TAdapterEditColumn.DisplayWidth Property

Specifies the width, in characters, of the generated HTML field.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to control the width of the HTML field that displays record status information. By default, DisplayWidth is 1 because the TCustomAdapterDisplayField control values are one character (M for modified or I for inserted).

181.1.12.2.10 TAdapterEditColumn.FieldName Property

Specifies the name of the associated dataset field.

Pascal

```
property FieldName: string;
```

C++

```
__property AnsiString FieldName;__property AnsiString FieldName;
```

Description

FieldName specifies the name of the dataset field that the adapter field represents.

See Also

DataSet (see page 7807)

181.1.12.2.11 TAdapterEditColumn.HideOptions Property

When field should be hidden.

Pascal

```
property HideOptions: TDisplayFieldHideOptions;
```

C++

```
__property TDisplayFieldHideOptions HideOptions;
```

Description

Set HideOptions to indicate when the field should be hidden from view.

181.1.12.2.12 TAdapterEditColumn.ImageHeight Property

Specifies the height of the image in the column.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

Set ImageHeight to indicate the height of the image in the column.

See Also

DisplayWidth (see page 25250)

ImageWidth (see page 25250)

181.1.12.2.13 TAdapterEditColumn.ImageWidth Property

Specifies the width of the image in the column.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

Set ImageHeight to indicate the width of the image in the column.

See Also

ImageHeight (see page 25250)

181.1.12.2.14 TAdapterEditColumn.InputType Property

Specifies the input type for the display field.

Pascal

```
property InputType: TInputFieldHTMLElementType;
```

C++

```
__property TInputFieldHTMLElementType InputType;
```

Description

Set InputType to indicate the type of control that should be generated to let users assign values to the adapter field.

See Also

ImageHeight (see page 25250)

ImageWidth (see page 25250)

DisplayType (see page 25249)

181.1.12.2.15 TAdapterEditColumn.MaxLength Property

Specifies the maximum number of characters the user can enter into the display field.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the display field. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

181.1.12.2.16 TAdapterEditColumn.SelectRows Property

Indicates the number of rows that can be selected.

Pascal

```
property SelectRows: Integer;
```

C++

```
__property int SelectRows;
```

Description

Use SelectRows to indicate the number of rows that can be selected.

181.1.12.2.17 TAdapterEditColumn.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom (see page 16898)

stylerule (see page 16899)

CaptionAttributes (see page 16897)

Setting Web Item Properties

181.1.12.2.18 TAdapterEditColumn.StyleRule Property

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

This is StyleRule, a member of class TAdapterEditColumn.

181.1.12.2.19 TAdapterEditColumn.TextAreaWrap Property

Specifies how the display field treats lines that are longer than the display width.

Pascal

```
property TextAreaWrap: TTextAreaWrap;
```

C++

```
__property TTextAreaWrap TextAreaWrap;
```

Description

Use TextAreaWrap to specify how the generated display field formats lines that are longer than the number of characters specified by DisplayWidth.

See Also

DisplayWidth (see page 25250)

181.1.12.2.20 TAdapterEditColumn.VAlign Property

Specifies the vertical alignment of the generated display control.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign specifies the control's vertical alignment relative to the containing HTML table cell.

See Also

Align (see page 25097)

181.1.12.2.21 TAdapterEditColumn.ViewMode Property

Indicates the viewing state of the adapter field.

Pascal

```
property ViewMode: TDisplayFieldViewMode;
```

C++

```
__property TDisplayFieldViewMode ViewMode;
```

Description

Use ViewMode to specify the viewing state of the field.

181.1.13 TAdapterErrorList Class

TAdapterErrorList is the error list display component.

Class Hierarchy

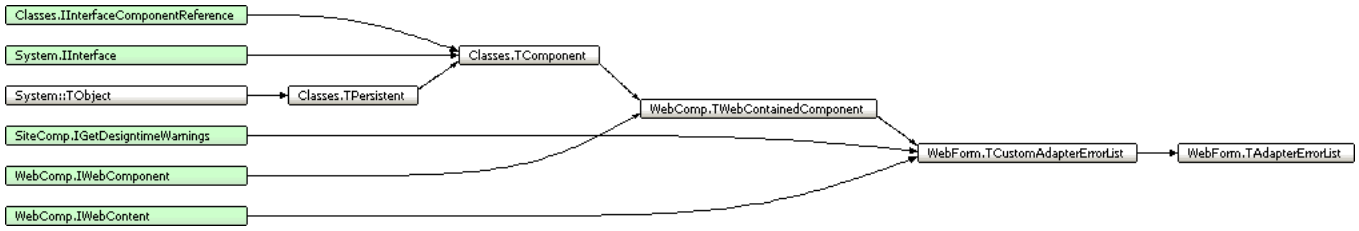

Pascal

```
TAdapterErrorList = class(TCustomAdapterErrorList);
```

C++

```
class TAdapterErrorList : public TCustomAdapterErrorList;
```

File

WebForm

Description

Use TAdapterErrorList to display errors relating to a specific adapter component.

TAdapterErrorList is a minimal implementation of TCustomAdapterErrorList (see page 25259). To extend its features, create your own concrete descendant of TCustomAdapterErrorList (see page 25259).

Note: Do not confuse TAdapterErrorList with TAdapterErrorsList.

See Also

TAdapterErrorsList (see page 23810)

181.1.13.1 TAdapterErrorList Members

The following tables list the members exposed by TAdapterErrorList.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 25263)	Returns the HTML that the Web item generates.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDesigntimeWarnings (see page 25264)	This is GetDesigntimeWarnings, a member of class TCustomAdapterErrorList.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	ImplContent (see page 25264)	This is ImplContent, a member of class TCustomAdapterErrorList.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 25264)	Forwards notification messages to all owned components.

	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).

	Create (see page 25262)	Creates an instance of TCustomAdapterErrorList (see page 25259).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	Destroy (see page 25263)	Frees the memory associated with an instance of TCustomAdapterErrorList (see page 25259).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDesigntimeWarnings (see page 19697)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.

	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Adapter (see page 25149)	Specifies a related adapter.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

181.1.13.2 TAdapterErrorList Properties

The properties of the TAdapterErrorList class are listed here.

Published Properties

	Name	Description
	Adapter (see page 25149)	Specifies a related adapter.

181.1.13.2.1 TAdapterErrorList.Adapter Property

Specifies a related adapter.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

Adapter specifies the adapter component for which this component reports errors.

181.1.14 TAdapterFieldGroup Class

TAdapterFieldGroup is a container for Adapter (see page 25156) display field components such as TAdapterDisplayField (see page 25109). TAdapterFieldGroup labels and fields in an HTML form.

Class Hierarchy

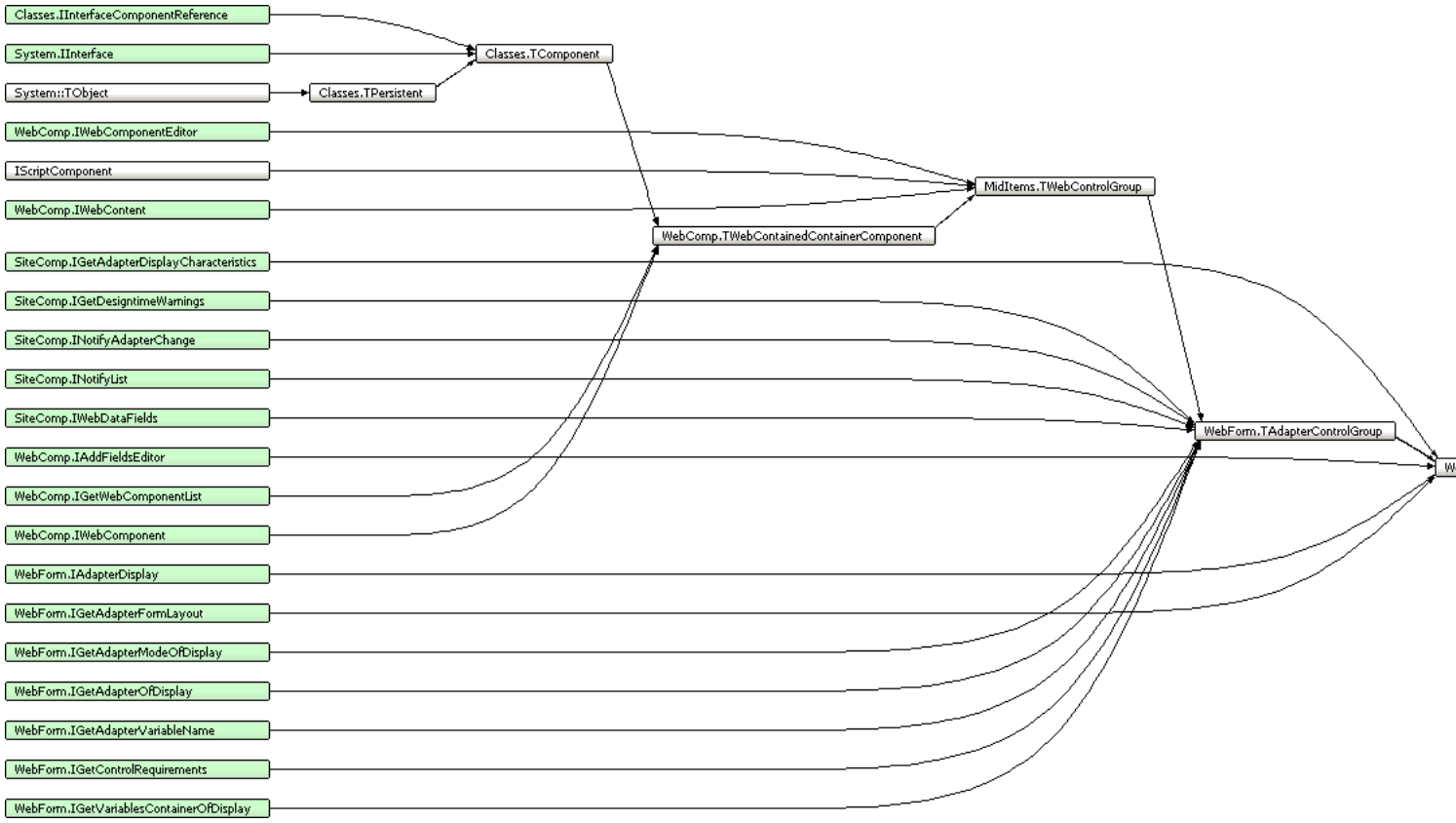

Pascal

```
TAdapterFieldGroup = class(TCustomAdapterFieldGroup);
```

C++

```
class TAdapterFieldGroup : public TCustomAdapterFieldGroup;
```

File

WebForm

Description

Add TAdapterFieldGroup as a child of TAdapterForm (see page 25159).

TAdapterFieldGroup is associated with an Adapter (see page 25156). All of its child display field components will use the same Adapter (see page 25156).

If no display fields have been explicitly added to the TAdapterFieldGroup, display fields associated with all of the Adapter (see page 25156) fields will be added by default. The properties of default display fields cannot be modified.

Modifying the AdapterMode (see page 25157) property of TAdapterFieldGroup may change the default display type or input type of Adapter (see page 25156) fields. This may cause the display field children of TAdapterFieldGroup to change appearance.

Use the properties and methods of TAdapterFieldGroup to:

Specify the associated Adapter (see page 25156).

Specify when the Adapter (see page 25156) Mode.

Specify whether default fields should be added.

See Also

TAdapterDisplayField (see page 25109)

TAdapterForm (see page 25159)

181.1.14.1 TAdapterFieldGroup Members

The following tables list the members exposed by TAdapterFieldGroup.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AdapterChange (see page 25077)	Responds when the adapter associated with this control group changes.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
	AddNotify (see page 25077)	Registers interest so that the adapter control group notifies an object of changes to its adapter.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
	CreateDefaultFields (see page 25078)	Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FindField (see page 25078)	Returns the child Web item that represents the value of a specified adapter field.
	GetAdapter (see page 25079)	Returns the value of the Adapter (see page 25084) property.
	GetAdapterDisplayCharacteristics (see page 25272)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterFieldGroup.
	GetAdapterFormLayout (see page 25272)	This is GetAdapterFormLayout, a member of class TCustomAdapterFieldGroup.
	GetAdapterModeOfAdapter (see page 25079)	Returns the current mode of the control group's adapter.
	GetAdapterModeOfDisplay (see page 25080)	Returns the value of the AdapterMode (see page 25085) property.
	GetAdapterVariableName (see page 25080)	Returns the variable name that refers to the control group's adapter in server-side script.
	GetAddAllFieldsItem (see page 25273)	This is GetAddAllFieldsItem, a member of class TCustomAdapterFieldGroup.
	GetAddFieldsDlgCaption (see page 25273)	This is GetAddFieldsDlgCaption, a member of class TCustomAdapterFieldGroup.

	GetAddFieldsItem (see page 25273)	This is GetAddFieldsItem, a member of class TCustomAdapterFieldGroup.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetControlRequirements (see page 25080)	Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetDesigntimeWarnings (see page 25081)	Populates a design-time warnings object with any warnings that the adapter control group needs to emit.
	GetFieldsList (see page 25081)	Fills a string list with the names of all of the associated adapter's fields.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
V	GetLayoutAttributes (see page 25273)	Returns an object that specifies how the generated group of controls should look.
	GetNewFieldPrefix (see page 25274)	This is GetNewFieldPrefix, a member of class TCustomAdapterFieldGroup.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVisibleFields (see page 25082)	Returns the value of the VisibleFields (see page 25086) property.
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
V	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
V	ImplCanAddClass (see page 16881)	Provides the underlying implementation for the CanAddClass (see page 16878) method.
V	ImplCanAddClass (see page 25274)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 16882)	Provides the underlying implementation for the Content (see page 16879) method.
V	ImplContent (see page 25274)	Provides the underlying implementation for the Content method.
V	ImplGetControlRequirements (see page 25082)	Provides the underlying implementation for the GetControlRequirements (see page 25080) method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsFieldInUse (see page 25082)	Indicates whether one of the adapter control group's child Web items represents a specified adapter field.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 25083)	Updates the TAdapterControlGroup (see page 25070) object when components are added to or removed from the application.
	NotifyAdapterChange (see page 25083)	Responds when the control group's adapter notifies it of a change.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.

	RemoveNotify (see page 25083)	Unregisters interest in changes to the control group's adapter.
	SetAdapter (see page 25084)	Sets the value of the Adapter (see page 25084) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16877)	Creates and initializes an instance of TWebControlGroup (see page 16872).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24742)	Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).
V	Create (see page 25076)	Creates and initializes an instance of TAdapterControlGroup (see page 25070).
V	Create (see page 25271)	Creates and initializes an instance of TCustomAdapterFieldGroup (see page 25265).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16877)	Frees an instance of TWebControlGroup (see page 16872).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
V	Destroy (see page 25076)	Frees an instance of TAdapterControlGroup (see page 25070).
V	Destroy (see page 25272)	Frees an instance of TCustomAdapterFieldGroup (see page 25265).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapter (see page 25306)	This is GetAdapter, a member of class IGetAdapterOfDisplay.
	GetAdapterDisplayCharacteristics (see page 19687)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.
	GetAdapterFormLayout (see page 25304)	This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.
	GetAdapterModeOfAdapter (see page 25305)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 25305)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.
	GetAdapterVariableName (see page 25307)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetAddAllFieldsItem (see page 24770)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 24770)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 24770)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetControlRequirements (see page 25309)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetDesigntimeWarnings (see page 19697)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNewFieldPrefix (see page 24771)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	GetVariablesContainer (see page 25311)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NotifyAdapterChange (see page 19725)	Responds when the adapter changes.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AdapterVariableName (see page 25308)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.
R	VisibleFields (see page 25086)	Lists the child Web items that generate the visible HTML controls this adapter control group arranges.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Adapter (see page 25156)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 25157)	Specifies the mode of the adapter that the controls in the group support.
	AddDefaultFields (see page 25157)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
	Custom (see page 25158)	Represents a tag option and its value in the generated INPUT tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 25158)	Adds style attributes to the generated button group.
	StyleRule (see page 25158)	Applies a style in the page producer's style sheet to the button group.
	Tag (see page 2808)	Stores an integer value as part of a component.

181.1.14.2 TAdapterFieldGroup Properties

The properties of the TAdapterFieldGroup class are listed here.

Published Properties

	Name	Description
	Adapter (see page 25156)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 25157)	Specifies the mode of the adapter that the controls in the group support.
	AddDefaultFields (see page 25157)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
	Custom (see page 25158)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 25158)	Adds style attributes to the generated button group.
	StyleRule (see page 25158)	Applies a style in the page producer's style sheet to the button group.

181.1.14.2.1 TAdapterFieldGroup.Adapter Property

Specifies the adapter that supplies values to or receives values from the controls in the group.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

Set Adapter to link this adapter control group and all its Web items to an adapter component. Each component that the WebFieldControls property lists generates an HTML control that represents the value of one of the adapter's fields.

See Also

GetFieldsList (see page 24159)

GetFieldsList (see page 25081)

AdapterMode (see page 25085)

WebFieldControls (see page 16884)

181.1.14.2.2 TAdapterFieldGroup.AdapterMode Property

Specifies the mode of the adapter that the controls in the group support.

Pascal

```
property AdapterMode: string;
```

C++

```
__property AnsiString AdapterMode;
```

Description

Some adapter components support different modes, or states. For example, dataset adapters can be in Edit, Insert, Browse, or Query mode. The mode of the adapter has an effect on the types of controls that the control group's Web items generate, and how those controls behave.

If the adapter that is assigned to the Adapter property supports different modes, you can set AdapterMode to force the controls in the group to represent the adapter fields in the specified mode. When AdapterMode is blank, the adapter control group uses the current mode of the adapter at the time when it generates its controls in the HTML form.

The possible values of AdapterMode depend on the modes that the adapter supports. If the adapter supports different modes, you can select a value for AdapterMode from a drop-down list in the Object Inspector that lists all the possible mode strings. If the adapter does not support different modes, the Object Inspector does not display a list, and the value of AdapterMode should be blank.

See Also

Mode (see page 7810)

Adapter (see page 25084)

WebFieldControls (see page 16884)

181.1.14.2.3 TAdapterFieldGroup.AddDefaultFields Property

Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.

Pascal

```
property AddDefaultFields: Boolean;
```

C++

```
__property Boolean AddDefaultFields;
```

Description

Set AddDefaultFields to indicate how you want the adapter control group to respond when you do not explicitly add child Web items. You can add child Web items at design time using Adapter page editor, or by right-clicking the control group and selecting Add Fields or Add Component. You can add child Web items at runtime using the Add method of the WebFieldControls property.

When AddDefaultFields is true, if you do not add child Web items to the adapter control group, it generates a default set of child Web items to display the fields of the adapter component that the Adapter property specifies.

When AddDefaultFields is false, if you do not add child Web items to the adapter control group, it remains empty, displaying no field values.

See Also

VisibleFields (see page 25086)

WebFieldControls (see page 16884)

Adapter (see page 25084)

181.1.14.2.4 TAdapterFieldGroup.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to represent one or more options that qualify the INPUT tag generated by the Web button group. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

```
ALIGN=TOP TABINDEX=2
```

See Also

Style (see page 25275)

181.1.14.2.5 TAdapterFieldGroup.Style Property

Adds style attributes to the generated button group.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to add a STYLE option that qualifies the INPUT tag generated by the Web button group. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated INPUT tag as the value of a STYLE option.

See Also

Custom (see page 25275)

StyleRule (see page 25276)

181.1.14.2.6 TAdapterFieldGroup.StyleRule Property

Applies a style in the page producer's style sheet to the button group.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated button group. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the button's Style property.

See Also

Style (see page 25275)

Styles (see page 19830)

StylesFile (see page 19831)

181.1.15 TAdapterForm Class

TAdapterForm generates an HTML form displaying Adapter fields.

Class Hierarchy

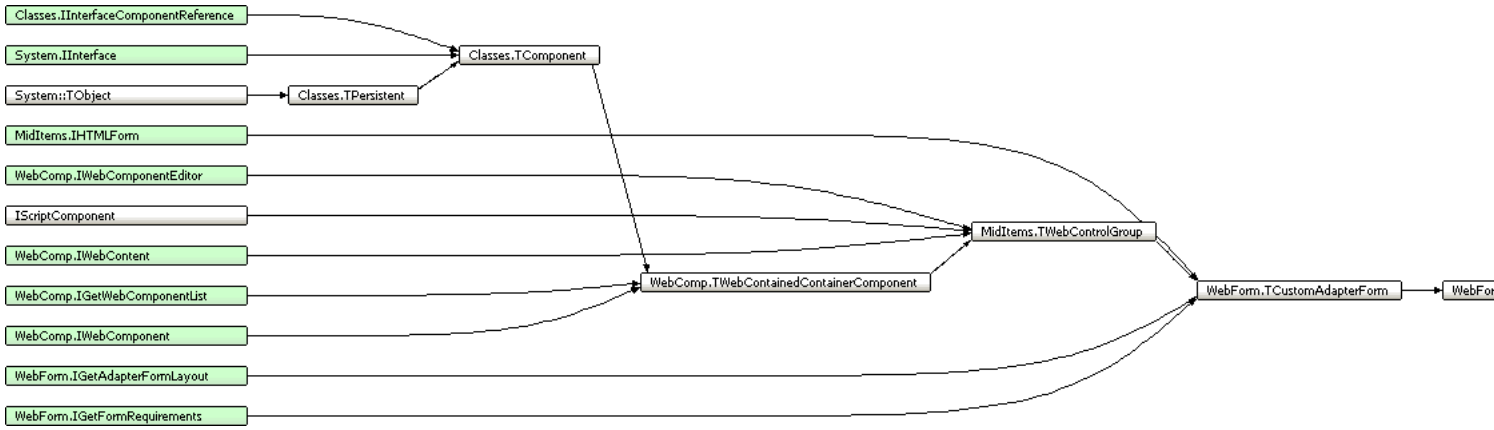

Pascal

```
TAdapterForm = class(TCustomAdapterForm);
```

C++

```
class TAdapterForm : public TCustomAdapterForm;
```

File

WebForm

Description

TAdapter form is a basic implementation of TCustomAdapterForm (see page 25276). Add TAdapterForm to a TAdapterPageProducer to display Adapter fields users can view and/or edit, and Adapter actions that users can execute.

Add child components to TAdapterForm to display Adapter fields and Adapter actions. Child components include TAdapterGrid (see page 25175), TAdapterFieldGroup (see page 25150), and TAdapterCommandGroup (see page 25062). Add TAdapterErrors to TAdapterForm to display errors that are generated by an Adapter.

Use the properties and methods of TAdapterForm to:

Customize the appearance of the HTML form.

Indicate the HTTP method used to submit the HTML form.

See Also

TAdapterPageProducer (see page 19802)

TAdapterGrid (see page 25175)

TAdapterFieldGroup (see page 25150)

TAdapterCommandGroup (see page 25062)

TAdapterErrorList (see page 25145)

181.1.15.1 TAdapterForm Members

The following tables list the members exposed by TAdapterForm.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetAdapterFormLayout (see page 25282)	This is GetAdapterFormLayout, a member of class TCustomAdapterForm.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetFormRequirements (see page 25282)	This is GetFormRequirements, a member of class TCustomAdapterForm.
	GetHTMLFormName (see page 25282)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 25283)	Returns the name of the javascript variable that refers to the generated form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).

	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
	GetLayoutAttributes (see page 25283)	Returns an object that specifies how the generated group of controls should look.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
	ImplCanAddClass (see page 16881)	Provides the underlying implementation for the CanAddClass (see page 16878) method.
	ImplCanAddClass (see page 25283)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 16882)	Provides the underlying implementation for the Content (see page 16879) method.
	ImplContent (see page 25284)	Provides the underlying implementation for the Content method.
	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.

V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16877)	Creates and initializes an instance of TWebControlGroup (see page 16872).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24742)	Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).
V	Create (see page 25281)	Creates and initializes an instance of TCustomAdapterForm (see page 25276).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16877)	Frees an instance of TWebControlGroup (see page 16872).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
V	Destroy (see page 25281)	Frees an instance of TCustomAdapterForm (see page 25276).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetAdapterFormLayout (see page 25304)	This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFormRequirements (see page 25310)	This is GetFormRequirements, a member of class IGetFormRequirements.
	GetHTMLFormName (see page 17065)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 17065)	Returns the name of the javascript variable that refers to the generated form.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.

	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	HTMLFormName (see page 17066)	Indicates the name of the generated HTML form.
R	HTMLFormVarName (see page 17066)	Indicates the name of the javascript variable that represents the HTML form.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Custom (see page 25164)	Represents a tag option and its value in the generated INPUT tag.
	FormMethod (see page 25165)	Specifies the method that the adapter form adds to its generated <form> tag.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Style (see page 25165)	Adds style attributes to the generated form.
	StyleRule (see page 25165)	Applies a style in the page producer's style sheet to the form.
	Tag (see page 2808)	Stores an integer value as part of a component.

181.1.15.2 TAdapterForm Properties

The properties of the TAdapterForm class are listed here.

Published Properties

	Name	Description
	Custom (see page 25164)	Represents a tag option and its value in the generated INPUT tag.
	FormMethod (see page 25165)	Specifies the method that the adapter form adds to its generated <form> tag.
	Style (see page 25165)	Adds style attributes to the generated form.
	StyleRule (see page 25165)	Applies a style in the page producer's style sheet to the form.

181.1.15.2.1 TAdapterForm.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to represent one or more options that qualify the <table> tag that the adapter form generates to manage the layout of its Web items. Each option and its value has the form

`OPTIONNAME=Value`

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

ALIGN=TOP TABINDEX=2

See Also

Style (see page 25285)

181.1.15.2.2 TAdapterForm.FormMethod Property

Specifies the method that the adapter form adds to its generated <form> tag.

Pascal

```
property FormMethod: TFormMethod;
```

C++

```
__property TFormMethod FormMethod;
```

Description

Set FormMethod to specify the METHOD header for HTTP messages that the form submits. This value gives rise to the MethodType property of the HTTP request message.

See Also

MethodType

181.1.15.2.3 TAdapterForm.Style Property

Adds style attributes to the generated form.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to add a STYLE option that qualifies the <table> tag that the adapter form generates to manage the layout of its Web items. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated <table> tag as the value of a STYLE option.

See Also

Custom (see page 25285)

StyleRule (see page 25286)

181.1.15.2.4 TAdapterForm.StyleRule Property

Applies a style in the page producer's style sheet to the form.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated form. The value of StyleRule specifies one of the user-defined style names defined in the adapter page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the form's Style property.

See Also

Style (see page 25285)

Styles (see page 19830)

StylesFile (see page 19831)

181.1.16 TAdapterFormLayout Class

Class Hierarchy

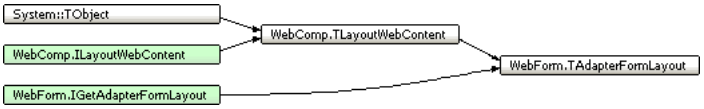

Pascal

```
TAdapterFormLayout = class(TLayoutWebContent, IGetAdapterFormLayout);
```

C++

```
class TAdapterFormLayout : public TLayoutWebContent, public IGetAdapterFormLayout;
```

File

WebForm

Description

This is class WebForm.TAdapterFormLayout.

181.1.16.1 TAdapterFormLayout Members

The following tables list the members exposed by TAdapterFormLayout.

Protected Methods

	Name	Description
	_AddRef (see page 24700)	Responds when the caller acquires a reference to this object's interface.
	_Release (see page 24701)	Responds when the caller releases a reference to this object's interface.
	EndButtons (see page 25169)	This is EndButtons, a member of class TAdapterFormLayout.
	EndField (see page 25169)	This is EndField, a member of class TAdapterFormLayout.
	EndFields (see page 25169)	This is EndFields, a member of class TAdapterFormLayout.
	EndRow (see page 25169)	This is EndRow, a member of class TAdapterFormLayout.
	EndTable (see page 25170)	This is EndTable, a member of class TAdapterFormLayout.
	GetAdapterFormLayout (see page 25170)	This is GetAdapterFormLayout, a member of class TAdapterFormLayout.
	GetChildTableIndent (see page 25170)	This is GetChildTableIndent, a member of class TAdapterFormLayout.
	GetFieldIndent (see page 25170)	This is GetFieldIndent, a member of class TAdapterFormLayout.
	GetTableIndent (see page 25171)	This is GetTableIndent, a member of class TAdapterFormLayout.
V	ImplLayoutButton (see page 25171)	Provides the underlying implementation for the LayoutButton method.
V	ImplLayoutField (see page 25171)	Provides the underlying implementation for the LayoutField method.
V	ImplLayoutLabelAndField (see page 25172)	Provides the underlying implementation for the LayoutLabelAndField method.
V	ImplLayoutTable (see page 25172)	Provides the underlying implementation for the LayoutTable method.
	LayoutButton (see page 24703)	Converts the HTML declaring a button into a form the associated Web item can use.
	LayoutField (see page 24704)	Converts the HTML declaring a field into a form the associated Web item can use.

	LayoutLabelAndField (see page 24704)	Converts the HTML declaring a labeled field into a form the associated Web item can use.
	LayoutTable (see page 24705)	This is LayoutTable, a member of class TLayoutWebContent.
	NextColumn (see page 25173)	This is NextColumn, a member of class TAdapterFormLayout.
V	QueryInterface (see page 24705)	Returns a reference to a specified interface if the object supports that interface.
	StartButtons (see page 25173)	This is StartButtons, a member of class TAdapterFormLayout.
	StartFields (see page 25173)	This is StartFields, a member of class TAdapterFormLayout.
	StartRow (see page 25173)	This is StartRow, a member of class TAdapterFormLayout.
	StartTable (see page 25174)	This is StartTable, a member of class TAdapterFormLayout.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25168)	Creates a new TLayoutWebContent object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndLayout (see page 25169)	This is EndLayout, a member of class TAdapterFormLayout.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LayoutButton (see page 24775)	This is LayoutButton, a member of class ILayoutWebContent.
	LayoutField (see page 24775)	This is LayoutField, a member of class ILayoutWebContent.
	LayoutLabelAndField (see page 24775)	Converts the HTML declaring a labeled field into a form the Web item's parent can use.
	LayoutTable (see page 24776)	This is LayoutTable, a member of class ILayoutWebContent.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	BreakButtons (see page 25174)	This is BreakButtons, a member of class TAdapterFormLayout.
R	ChildTableIndent (see page 25174)	This is ChildTableIndent, a member of class TAdapterFormLayout.
	ColumnCount (see page 25174)	This is ColumnCount, a member of class TAdapterFormLayout.
R	FieldIndent (see page 25175)	This is FieldIndent, a member of class TAdapterFormLayout.

R	ParentLayout (see page 24706)	Provides access to the layout object of the Web item that is the parent of the Web item to which this object belongs.
	TableHeader (see page 25175)	This is TableHeader, a member of class TAdapterFormLayout.
R	TableIndent (see page 25175)	This is TableIndent, a member of class TAdapterFormLayout.

181.1.16.2 TAdapterFormLayout.Create Constructor

Creates a new TLayoutWebContent object.

Pascal

```
constructor Create(AParentLayout: TLayout);
```

C++

```
__fastcall TAdapterFormLayout(TLayout AParentLayout);
```

Description

Call Create to instantiate a TLayoutWebContent descendant to use when fetching the content produced by child Web items.

AParentLayout is the layout object of the parent Web item (the grandparent of the child Web items that call this object's methods). AParentLayout becomes the value of the ParentLayout property.

See Also

ParentLayout (see page 24706)

181.1.16.3 TAdapterFormLayout Methods

The methods of the TAdapterFormLayout class are listed here.

Protected Methods

	Name	Description
	EndButtons (see page 25169)	This is EndButtons, a member of class TAdapterFormLayout.
	EndField (see page 25169)	This is EndField, a member of class TAdapterFormLayout.
	EndFields (see page 25169)	This is EndFields, a member of class TAdapterFormLayout.
	EndRow (see page 25169)	This is EndRow, a member of class TAdapterFormLayout.
	EndTable (see page 25170)	This is EndTable, a member of class TAdapterFormLayout.
	GetAdapterFormLayout (see page 25170)	This is GetAdapterFormLayout, a member of class TAdapterFormLayout.
	GetChildTableIndent (see page 25170)	This is GetChildTableIndent, a member of class TAdapterFormLayout.
	GetFieldIndent (see page 25170)	This is GetFieldIndent, a member of class TAdapterFormLayout.
	GetTableIndent (see page 25171)	This is GetTableIndent, a member of class TAdapterFormLayout.
V	ImplLayoutButton (see page 25171)	Provides the underlying implementation for the LayoutButton method.
V	ImplLayoutField (see page 25171)	Provides the underlying implementation for the LayoutField method.
V	ImplLayoutLabelAndField (see page 25172)	Provides the underlying implementation for the LayoutLabelAndField method.
V	ImplLayoutTable (see page 25172)	Provides the underlying implementation for the LayoutTable method.
	NextColumn (see page 25173)	This is NextColumn, a member of class TAdapterFormLayout.
	StartButtons (see page 25173)	This is StartButtons, a member of class TAdapterFormLayout.
	StartFields (see page 25173)	This is StartFields, a member of class TAdapterFormLayout.
	StartRow (see page 25173)	This is StartRow, a member of class TAdapterFormLayout.
	StartTable (see page 25174)	This is StartTable, a member of class TAdapterFormLayout.

Public Methods

	Name	Description
	EndLayout (see page 25169)	This is EndLayout, a member of class TAdapterFormLayout.

181.1.16.3.1 TAdapterFormLayout.EndButtons Method

Pascal

```
function EndButtons: string;
```

C++

```
__fastcall AnsiString EndButtons();
```

Description

This is EndButtons, a member of class TAdapterFormLayout.

181.1.16.3.2 TAdapterFormLayout.EndField Method

Pascal

```
function EndField: string;
```

C++

```
__fastcall AnsiString EndField();
```

Description

This is EndField, a member of class TAdapterFormLayout.

181.1.16.3.3 TAdapterFormLayout.EndFields Method

Pascal

```
function EndFields: string;
```

C++

```
__fastcall AnsiString EndFields();
```

Description

This is EndFields, a member of class TAdapterFormLayout.

181.1.16.3.4 TAdapterFormLayout.EndLayout Method

Pascal

```
function EndLayout: string;
```

C++

```
__fastcall AnsiString EndLayout();
```

Description

This is EndLayout, a member of class TAdapterFormLayout.

181.1.16.3.5 TAdapterFormLayout.EndRow Method

Pascal

```
function EndRow: string;
```

C++

```
__fastcall AnsiString EndRow();
```

Description

This is EndRow, a member of class TAdapterFormLayout.

181.1.16.3.6 TAdapterFormLayout.EndTable Method

Pascal

```
function EndTable: string;
```

C++

```
__fastcall AnsiString EndTable();
```

Description

This is EndTable, a member of class TAdapterFormLayout.

181.1.16.3.7 TAdapterFormLayout.GetAdapterFormLayout Method

Pascal

```
function GetAdapterFormLayout: TAdapterFormLayout;
```

C++

```
__fastcall TAdapterFormLayout GetAdapterFormLayout();
```

Description

This is GetAdapterFormLayout, a member of class TAdapterFormLayout.

181.1.16.3.8 TAdapterFormLayout.GetChildTableIndent Method

Pascal

```
function GetChildTableIndent: string;
```

C++

```
__fastcall AnsiString GetChildTableIndent();
```

Description

This is GetChildTableIndent, a member of class TAdapterFormLayout.

181.1.16.3.9 TAdapterFormLayout.GetFieldIndent Method

Pascal

```
function GetFieldIndent: string;
```

C++

```
__fastcall AnsiString GetFieldIndent();
```

Description

This is GetFieldIndent, a member of class TAdapterFormLayout.

181.1.16.3.10 TAdapterFormLayout.GetTableIndent Method

Pascal

```
function GetTableIndent: string;
```

C++

```
__fastcall AnsiString GetTableIndent();
```

Description

This is GetTableIndent, a member of class TAdapterFormLayout.

181.1.16.3.11 TAdapterFormLayout.ImplLayoutButton Method

Provides the underlying implementation for the LayoutButton method.

Pascal

```
function ImplLayoutButton(const HTMLButton: string; Attributes: TLayoutAttributes): string;
override;
```

C++

```
virtual __fastcall AnsiString ImplLayoutButton(const AnsiString HTMLButton, TLayoutAttributes
Attributes);
```

Description

The LayoutButton method calls ImplLayoutButton to convert the HTML that declares a button (typically an <INPUT> tag) into the form that the associated Web item requires.

HTMLButton is the HTML that declares a button. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the button.

LayoutButton returns the HTML for the button in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutButton is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutButton method.

See Also

LayoutButton (see page 24703)

181.1.16.3.12 TAdapterFormLayout.ImplLayoutField Method

Provides the underlying implementation for the LayoutField method.

Pascal

```
function ImplLayoutField(const HTMLField: string; Attributes: TLayoutAttributes): string;
override;
```

C++

```
virtual __fastcall AnsiString ImplLayoutField(const AnsiString HTMLField, TLayoutAttributes
Attributes);
```

Description

The LayoutField method calls ImplLayoutField to convert the HTML that declares a field (an input control that represents a data field or a named value) into the form that the associated Web item requires.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field.

LayoutField returns the HTML for the field in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutField is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutField method.

See Also

LayoutField (see page 24704)

181.1.16.3.13 TAdapterFormLayout.ImplLayoutLabelAndField Method

Provides the underlying implementation for the LayoutLabelAndField method.

Pascal

```
function ImplLayoutLabelAndField(const HTMLLabel: string; const HTMLField: string; Attributes:
TLayoutAttributes): string; override;
```

C++

```
virtual __fastcall AnsiString ImplLayoutLabelAndField(const AnsiString HTMLLabel, const
AnsiString HTMLField, TLayoutAttributes Attributes);
```

Description

The LayoutLabelAndField method calls ImplLayoutLabelAndField to convert the HTML that declares a labeled field (an input control that represents a data field or a named value along with its label) into the form that the associated Web item requires.

HTMLLabel is the HTML that declares the label. This is produced by the child Web item.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field and its label.

LayoutLabelAndField returns the HTML for the field and label in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutLabelAndField is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutLabelAndField method.

See Also

LayoutLabelAndField (see page 24704)

181.1.16.3.14 TAdapterFormLayout.ImplLayoutTable Method

Provides the underlying implementation for the LayoutTable method.

Pascal

```
function ImplLayoutTable(const HTMLTable: string; Attributes: TLayoutAttributes): string;
override;
```

C++

```
virtual __fastcall AnsiString ImplLayoutTable(const AnsiString HTMLTable, TLayoutAttributes
Attributes);
```

Description

The LayoutTable method calls ImplLayoutTable to convert the HTML that declares a table (typically a structure used to layout the content produced by the child Web item's children) into the form that the associated Web item requires.

HTMLTable is the HTML that declares the table. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the table.

LayoutTable returns the HTML for the table in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutTable is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutTable method.

See Also

layouttable (see page 24705)

181.1.16.3.15 TAdapterFormLayout.NextColumn Method

Pascal

```
function NextColumn(SubItemCount: Integer): string;
```

C++

```
__fastcall AnsiString NextColumn(int SubItemCount);
```

Description

This is NextColumn, a member of class TAdapterFormLayout.

181.1.16.3.16 TAdapterFormLayout.StartButtons Method

Pascal

```
function StartButtons: string;
```

C++

```
__fastcall AnsiString StartButtons();
```

Description

This is StartButtons, a member of class TAdapterFormLayout.

181.1.16.3.17 TAdapterFormLayout.StartFields Method

Pascal

```
function StartFields(ColCount: Integer): string;
```

C++

```
__fastcall AnsiString StartFields(int ColCount);
```

Description

This is StartFields, a member of class TAdapterFormLayout.

181.1.16.3.18 TAdapterFormLayout.StartRow Method

Pascal

```
function StartRow: string;
```

C++

```
__fastcall AnsiString StartRow();
```

Description

This is StartRow, a member of class TAdapterFormLayout.

181.1.16.3.19 TAdapterFormLayout.StartTable Method

Pascal

```
function StartTable: string;
```

C++

```
__fastcall AnsiString StartTable();
```

Description

This is StartTable, a member of class TAdapterFormLayout.

181.1.16.4 TAdapterFormLayout Properties

The properties of the TAdapterFormLayout class are listed here.

Public Properties

	Name	Description
	BreakButtons (see page 25174)	This is BreakButtons, a member of class TAdapterFormLayout.
R	ChildTableIndent (see page 25174)	This is ChildTableIndent, a member of class TAdapterFormLayout.
	ColumnCount (see page 25174)	This is ColumnCount, a member of class TAdapterFormLayout.
R	FieldIndent (see page 25175)	This is FieldIndent, a member of class TAdapterFormLayout.
	TableHeader (see page 25175)	This is TableHeader, a member of class TAdapterFormLayout.
R	TableIndent (see page 25175)	This is TableIndent, a member of class TAdapterFormLayout.

181.1.16.4.1 TAdapterFormLayout.BreakButtons Property

Pascal

```
property BreakButtons: Boolean;
```

C++

```
__property Boolean BreakButtons;
```

Description

This is BreakButtons, a member of class TAdapterFormLayout.

181.1.16.4.2 TAdapterFormLayout.ChildTableIndent Property

Pascal

```
property ChildTableIndent: string;
```

C++

```
__property AnsiString ChildTableIndent;
```

Description

This is ChildTableIndent, a member of class TAdapterFormLayout.

181.1.16.4.3 TAdapterFormLayout.ColumnCount Property

Pascal

```
property ColumnCount: Integer;
```

C++

```
__property int ColumnCount;
```

Description

This is ColumnCount, a member of class TAdapterFormLayout.

181.1.16.4.4 TAdapterFormLayout.FieldIndent Property

Pascal

```
property FieldIndent: string;
```

C++

```
__property AnsiString FieldIndent;
```

Description

This is FieldIndent, a member of class TAdapterFormLayout.

181.1.16.4.5 TAdapterFormLayout.TableHeader Property

Pascal

```
property TableHeader: string;
```

C++

```
__property AnsiString TableHeader;
```

Description

This is TableHeader, a member of class TAdapterFormLayout.

181.1.16.4.6 TAdapterFormLayout.TableIndent Property

Pascal

```
property TableIndent: string;
```

C++

```
__property AnsiString TableIndent;
```

Description

This is TableIndent, a member of class TAdapterFormLayout.

181.1.17 TAdapterGrid Class

TAdapterGrid is a container for Adapter (see page 25182) display field components such as TAdapterDisplayColumn (see page 25099). TAdapterGrid generates an HTML table.

Class Hierarchy

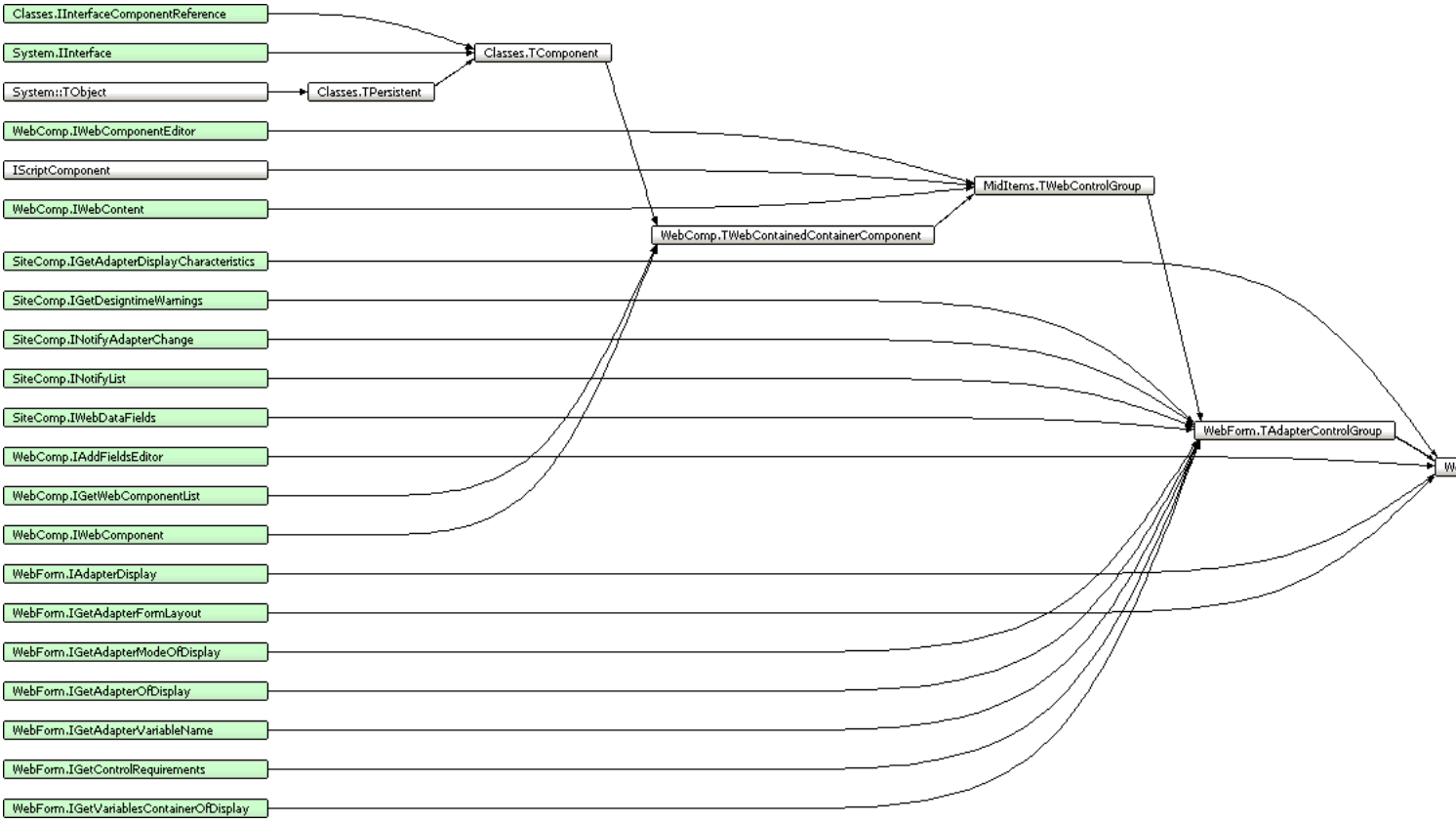

Pascal

```
TAdapterGrid = class(TCustomAdapterGrid);
```

C++

```
class TAdapterGrid : public TCustomAdapterGrid;
```

File

WebForm

Description

Add TAdapterGrid as a child of TAdapterForm (see page 25159). TAdapterGrid is associated with an Adapter (see page 25182). All of it's child display field components will use the same Adapter (see page 25182).

If no display fields have been explicitly added to the TAdapterGrid, display fields associated with all of the Adapter (see page 25182) fields will be added by default. The properties of default display fields cannot be modified.

Modifying the AdapterMode (see page 25182) property of TAdapterGrid may change the default display type or input type of Adapter (see page 25182) fields. This may cause the display field children of TAdapterGrid to change appearance.

Use the properties and methods of TAdapterGrid to:

Specify the associated Adapter (see page 25182).

Specify when the Adapter (see page 25182) Mode.

Specify whether default fields should be added.

Customize the appearance of the HTML table.

See Also

TAdapterDisplayColumn (see page 25099)

TAdapterForm (see page 25159)

181.1.17.1 TAdapterGrid Members

The following tables list the members exposed by TAdapterGrid.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AdapterChange (see page 25077)	Responds when the adapter associated with this control group changes.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
	AddNotify (see page 25077)	Registers interest so that the adapter control group notifies an object of changes to its adapter.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
	CreateDefaultFields (see page 25078)	Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FindField (see page 25078)	Returns the child Web item that represents the value of a specified adapter field.
	GetAdapter (see page 25079)	Returns the value of the Adapter (see page 25084) property.
	GetAdapterDisplayCharacteristics (see page 25293)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterGrid.
	GetAdapterFormLayout (see page 25294)	This is GetAdapterFormLayout, a member of class TCustomAdapterGrid.
	GetAdapterModeOfAdapter (see page 25079)	Returns the current mode of the control group's adapter.
	GetAdapterModeOfDisplay (see page 25080)	Returns the value of the AdapterMode (see page 25085) property.
	GetAdapterVariableName (see page 25080)	Returns the variable name that refers to the control group's adapter in server-side script.
	GetAddAllFieldsItem (see page 25294)	This is GetAddAllFieldsItem, a member of class TCustomAdapterGrid.
	GetAddFieldsDlgCaption (see page 25294)	This is GetAddFieldsDlgCaption, a member of class TCustomAdapterGrid.
	GetAddFieldsItem (see page 25294)	This is GetAddFieldsItem, a member of class TCustomAdapterGrid.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.

	GetControlRequirements (see page 25080)	Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetDesigntimeWarnings (see page 25081)	Populates a design-time warnings object with any warnings that the adapter control group needs to emit.
	GetFieldsList (see page 25081)	Fills a string list with the names of all of the associated adapter's fields.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
	GetNewFieldPrefix (see page 25294)	This is GetNewFieldPrefix, a member of class TCustomAdapterGrid.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVisibleFields (see page 25082)	Returns the value of the VisibleFields (see page 25086) property.
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
V	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
V	ImplCanAddClass (see page 16881)	Provides the underlying implementation for the CanAddClass (see page 16878) method.
V	ImplCanAddClass (see page 25295)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 16882)	Provides the underlying implementation for the Content (see page 16879) method.
V	ImplContent (see page 25295)	Provides the underlying implementation for the Content method.
V	ImplGetControlRequirements (see page 25082)	Provides the underlying implementation for the GetControlRequirements (see page 25080) method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsFieldInUse (see page 25082)	Indicates whether one of the adapter control group's child Web items represents a specified adapter field.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 25083)	Updates the TAdapterControlGroup (see page 25070) object when components are added to or removed from the application.
	NotifyAdapterChange (see page 25083)	Responds when the control group's adapter notifies it of a change.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	RemoveNotify (see page 25083)	Unregisters interest in changes to the control group's adapter.
	SetAdapter (see page 25084)	Sets the value of the Adapter (see page 25084) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16877)	Creates and initializes an instance of TWebControlGroup (see page 16872).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24742)	Creates and initializes an instance of TWebContainedContainerComponent (see page 24738).
V	Create (see page 25076)	Creates and initializes an instance of TAdapterControlGroup (see page 25070).
V	Create (see page 25292)	Creates and initializes an instance of TCustomAdapterGrid (see page 25286).

	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 16877)	Frees an instance of TWebControlGroup (see page 16872).
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24743)	Frees an instance of TWebContainedContainerComponent (see page 24738).
	Destroy (see page 25076)	Frees an instance of TAdapterControlGroup (see page 25070).
	Destroy (see page 25293)	Frees an instance of TAdapterControlGroup.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapter (see page 25306)	This is GetAdapter, a member of class IGetAdapterOfDisplay.
	GetAdapterDisplayCharacteristics (see page 19687)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.
	GetAdapterFormLayout (see page 25304)	This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.
	GetAdapterModeOfAdapter (see page 25305)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 25305)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.
	GetAdapterVariableName (see page 25307)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetAddAllFieldsItem (see page 24770)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 24770)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 24770)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetControlRequirements (see page 25309)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetDesigntimeWarnings (see page 19697)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNewFieldPrefix (see page 24771)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	GetVariablesContainer (see page 25311)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.

V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NotifyAdapterChange (see page 19725)	Responds when the adapter changes.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AdapterVariableName (see page 25308)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.
R	VisibleFields (see page 25086)	Lists the child Web items that generate the visible HTML controls this adapter control group arranges.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Adapter (see page 25182)	Specifies the adapter that supplies values to or receives values from the controls in the group.

	AdapterMode (see page 25182)	Specifies the mode of the adapter that the controls in the group support.
	AddDefaultFields (see page 25183)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
	HeadingAttributes (see page 25184)	Determines the display characteristics of the table's header.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	RowAttributes (see page 25184)	Determines the display characteristics of the table's body.
	TableAttributes (see page 25184)	Determines the basic display characteristics of the table.
	Tag (see page 2808)	Stores an integer value as part of a component.

181.1.17.2 TAdapterGrid Properties

The properties of the TAdapterGrid class are listed here.

Published Properties

	Name	Description
	Adapter (see page 25182)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 25182)	Specifies the mode of the adapter that the controls in the group support.
	AddDefaultFields (see page 25183)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
	HeadingAttributes (see page 25184)	Determines the display characteristics of the table's header.
	RowAttributes (see page 25184)	Determines the display characteristics of the table's body.
	TableAttributes (see page 25184)	Determines the basic display characteristics of the table.

181.1.17.2.1 TAdapterGrid.Adapter Property

Specifies the adapter that supplies values to or receives values from the controls in the group.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

Set Adapter to link this adapter control group and all its Web items to an adapter component. Each component that the WebFieldControls property lists generates an HTML control that represents the value of one of the adapter's fields.

See Also

GetFieldsList (see page 24159)

GetFieldsList (see page 25081)

AdapterMode (see page 25085)

WebFieldControls (see page 16884)

181.1.17.2.2 TAdapterGrid.AdapterMode Property

Specifies the mode of the adapter that the controls in the group support.

Pascal

```
property AdapterMode: string;
```

C++

```
__property AnsiString AdapterMode;
```

Description

Some adapter components support different modes, or states. For example, dataset adapters can be in Edit, Insert, Browse, or Query mode. The mode of the adapter has an effect on the types of controls that the control group's Web items generate, and how those controls behave.

If the adapter that is assigned to the Adapter property supports different modes, you can set AdapterMode to force the controls in the group to represent the adapter fields in the specified mode. When AdapterMode is blank, the adapter control group uses the current mode of the adapter at the time when it generates its controls in the HTML form.

The possible values of AdapterMode depend on the modes that the adapter supports. If the adapter supports different modes, you can select a value for AdapterMode from a drop-down list in the Object Inspector that lists all the possible mode strings. If the adapter does not support different modes, the Object Inspector does not display a list, and the value of AdapterMode should be blank.

See Also

Mode (see page 7810)

Adapter (see page 25084)

WebFieldControls (see page 16884)

181.1.17.2.3 TAdapterGrid.AddDefaultFields Property

Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.

Pascal

```
property AddDefaultFields: Boolean;
```

C++

```
__property Boolean AddDefaultFields;
```

Description

Set AddDefaultFields to indicate how you want the adapter control group to respond when you do not explicitly add child Web items. You can add child Web items at design time using Adapter page editor, or by right-clicking the control group and selecting Add Fields or Add Component. You can add child Web items at runtime using the Add method of the WebFieldControls property.

When AddDefaultFields is true, if you do not add child Web items to the adapter control group, it generates a default set of child Web items to display the fields of the adapter component that the Adapter property specifies.

When AddDefaultFields is false, if you do not add child Web items to the adapter control group, it remains empty, displaying no field values.

See Also

VisibleFields (see page 25086)

WebFieldControls (see page 16884)

Adapter (see page 25084)

181.1.17.2.4 TAdapterGrid.HeadingAttributes Property

Determines the display characteristics of the table's header.

Pascal

```
property HeadingAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes HeadingAttributes;
```

Description

Use HeadingAttributes to specify any display attributes that apply only to the heading of the generated HTML table. HeadingAttributes determines the options that are added to the <TR> tag that defines the header of the grid. These include the color of the header, the alignment of the caption for each column, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the rows of the grid, use the TableAttributes or RowAttributes property instead.

See Also

TableAttributes (see page 25297)

RowAttributes (see page 25296)

181.1.17.2.5 TAdapterGrid.RowAttributes Property

Determines the display characteristics of the table's body.

Pascal

```
property RowAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes RowAttributes;
```

Description

Use RowAttributes to specify any display attributes that apply only to the rows of the generated HTML table. RowAttributes determines the options that are added to the <TR> tag for each row in the body of the grid. These include the color of the rows, the alignment of data within each cell, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the header of the grid, use the TableAttributes or HeadingAttributes property instead.

See Also

TableAttributes (see page 25297)

HeadingAttributes (see page 25296)

181.1.17.2.6 TAdapterGrid.TableAttributes Property

Determines the basic display characteristics of the table.

Pascal

```
property TableAttributes: TGridAttributes;
```

C++

```
__property TGridAttributes TableAttributes;
```

Description

Use TableAttributes to specify values for the most common options on the HTML <TABLE> tag. TableAttributes lets you specify the alignment of the table in its HTML document, its color and border size, the offset of data within each table cell, as well as any custom options or style rules.

Note: To influence the display attributes of only the header of the grid, or only the rows in its body, use the TableAttributes or HeadingAttributes property instead.

See Also

RowAttributes (see page 25296)

HeadingAttributes (see page 25296)

181.1.18 TControlRequirements Class

TControlRequirements lists the variables and functions that the controls in an adapter form require to generate their HTML.

Class Hierarchy

Pascal

```
TControlRequirements = class(TAbstractControlRequirements);
```

C++

```
class TControlRequirements : public TAbstractControlRequirements;
```

File

WebForm

Description

Adapter forms create instances of TControlRequirements to store information about variables and functions that are needed for generating an HTML form to display the controls they contain. Components that generate HTML controls to represent the fields of an adapter (TAdapterControlGroup (see page 25070) and TAdapterDataDisplay (see page 25086) descendants) use the TControlRequirements instance that the form provides to store these variable and function declarations.

Typically, applications do not need to create or use TControlRequirements instances. Component writers that create Web items for an adapter form (that is, components that generate HTML controls to reflect the fields of an adapter) use the methods of TControlRequirements when implementing the GetControlRequirements method.

See Also

TAdapterControlGroup (see page 25070)

TAdapterDataDisplay (see page 25086)

TAdapterForm (see page 25159)

181.1.18.1 TControlRequirements Members

The following tables list the members exposed by TControlRequirements.

Public Methods

	Name	Description
	Create (see page 25186)	Creates and initializes an instance of TControlRequirements (see page 25185).
V	DefineFunction (see page 25187)	Adds the declaration of a function to those that this control requirements object stores.
V	DefineVariable (see page 25187)	Adds the declaration of a variable to those that this control requirements object stores.
V	Destroy (see page 25186)	Frees an instance of TControlRequirements (see page 25185).
V	IsFunctionDefined (see page 25188)	Indicates whether a function with the specified name has already been added to the list of control requirements.
V	IsVariableDefined (see page 25188)	Indicates whether a variable with the specified name has already been added to the list of control requirements.
V	RequiresMultipartForm (see page 25189)	Stores the fact that the generated HTML form should be multipart.

Public Properties

	Name	Description
R	NeedMultipartForm (see page 25189)	Indicates whether the form that the adapter form generates must be multipart.

181.1.18.2 TControlRequirements.Create Constructor

Creates and initializes an instance of TControlRequirements (see page 25185).

Pascal

```
constructor Create;
```

C++

```
__fastcall TControlRequirements();
```

Description

Applications should have no need to instantiate TControlRequirements (see page 25185). The adapter form creates instances of this object to assemble and store information from their Web items.

See Also

TAdapterForm (see page 25159)

181.1.18.3 TControlRequirements.Destroy Destructor

Frees an instance of TControlRequirements (see page 25185).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TControlRequirements();
```

Description

Applications should have no need to create or destroy TControlRequirements (see page 25185) instances. These objects are created and freed by the adapter form.

See Also

TAdapterForm (see page 25159)

181.1.18.4 TControlRequirements Methods

The methods of the TControlRequirements class are listed here.

Public Methods

	Name	Description
	DefineFunction (see page 25187)	Adds the declaration of a function to those that this control requirements object stores.
	DefineVariable (see page 25187)	Adds the declaration of a variable to those that this control requirements object stores.
	IsFunctionDefined (see page 25188)	Indicates whether a function with the specified name has already been added to the list of control requirements.
	IsVariableDefined (see page 25188)	Indicates whether a variable with the specified name has already been added to the list of control requirements.
	RequiresMultipartForm (see page 25189)	Stores the fact that the generated HTML form should be multipart.

181.1.18.4.1 TControlRequirements.DefineFunction Method

Adds the declaration of a function to those that this control requirements object stores.

Pascal

```
procedure DefineFunction(const AName: string; const ABody: string); override;
```

C++

```
virtual __fastcall DefineFunction(const AnsiString AName, const AnsiString ABody);
```

Description

DefineFunction stores the declaration of a function that has the name AName, where ABody is the text of the function body.

AName is the name of the function. This name is one of a standard set that the adapter form knows to call under specific circumstances. For example, the name can be the standard name of the function to generate an HTML button.

ABody is the body of the specified function. This value is typically customized to reflect the properties of a specific control in the form.

See Also

IsFunctionDefined (see page 25188)

DefineVariable (see page 25187)

181.1.18.4.2 TControlRequirements.DefineVariable Method

Adds the declaration of a variable to those that this control requirements object stores.

Pascal

```
procedure DefineVariable(const AName: string; const AValue: string; AAdapter: Boolean);
override;
```

C++

```
virtual __fastcall DefineVariable(const AnsiString AName, const AnsiString AValue, Boolean
AAdapter);
```

Description

DefineVariable stores the name and value of a variable declaration.

AName is the name of the variable.

AValue is the value to assign to the variable.

AAdapter indicates whether this variable is the one that declares the adapter object whose field a control represents or whose set of fields a control group represents.

See Also

IsVariableDefined (see page 25188)

DefineFunction (see page 25187)

181.1.18.4.3 TControlRequirements.IsFunctionDefined Method

Indicates whether a function with the specified name has already been added to the list of control requirements.

Pascal

```
function IsFunctionDefined(const AName: string): Boolean; override;
```

C++

```
virtual __fastcall Boolean IsFunctionDefined(const AnsiString AName);
```

Description

IsFunctionDefined returns true if the control requirements object already lists a function with the specified name. That is, IsFunctionDefined returns true if the DefineFunction (see page 25187) method has been called to define the specified function, and false otherwise.

AName is the name of the function.

See Also

DefineFunction (see page 25187)

IsVariableDefined (see page 25188)

181.1.18.4.4 TControlRequirements.IsVariableDefined Method

Indicates whether a variable with the specified name has already been added to the list of control requirements.

Pascal

```
function IsVariableDefined(const AName: string): Boolean; override;
```

C++

```
virtual __fastcall Boolean IsVariableDefined(const AnsiString AName);
```

Description

IsVariableDefined returns true if the control requirements object already lists a variable with the specified name. That is, IsVariableDefined returns true if the DefineVariable (see page 25187) method has been called to define a value for the specified variable, and false otherwise.

AName is the name of the variable.

See Also

DefineVariable (see page 25187)

IsFunctionDefined (see page 25188)

181.1.18.4.5 TControlRequirements.RequiresMultipartForm Method

Stores the fact that the generated HTML form should be multipart.

Pascal

```
procedure RequiresMultipartForm; override;
```

C++

```
virtual __fastcall RequiresMultipartForm();
```

Description

Web items call this method if they need to be contained in a multipart form (for example, when a field in the form is represented by a reference to another file). A multipart form's HTML declaration includes the attribute 'enctype=multipart/form-data'.

See Also

NeedMultipartForm (see page 25189)

181.1.18.5 TControlRequirements Properties

The properties of the TControlRequirements class are listed here.

Public Properties

	Name	Description
	NeedMultipartForm (see page 25189)	Indicates whether the form that the adapter form generates must be multipart.

181.1.18.5.1 TControlRequirements.NeedMultipartForm Property

Indicates whether the form that the adapter form generates must be multipart.

Pascal

```
property NeedMultipartForm: Boolean;
```

C++

```
__property Boolean NeedMultipartForm;
```

Description

NeedMultipartForm indicates whether any control has called the RequiresMultipartForm (see page 25189) method to indicate that the form that contains it must be multipart. A multipart form's HTML declaration includes the attribute 'enctype=multipart/form-data'.

See Also

RequiresMultipartForm (see page 25189)

181.1.19 TCustomAdapterActionButton Class

TCustomAdapterActionButton is the base class for Web items that generate HTML controls for executing adapter actions.

Class Hierarchy

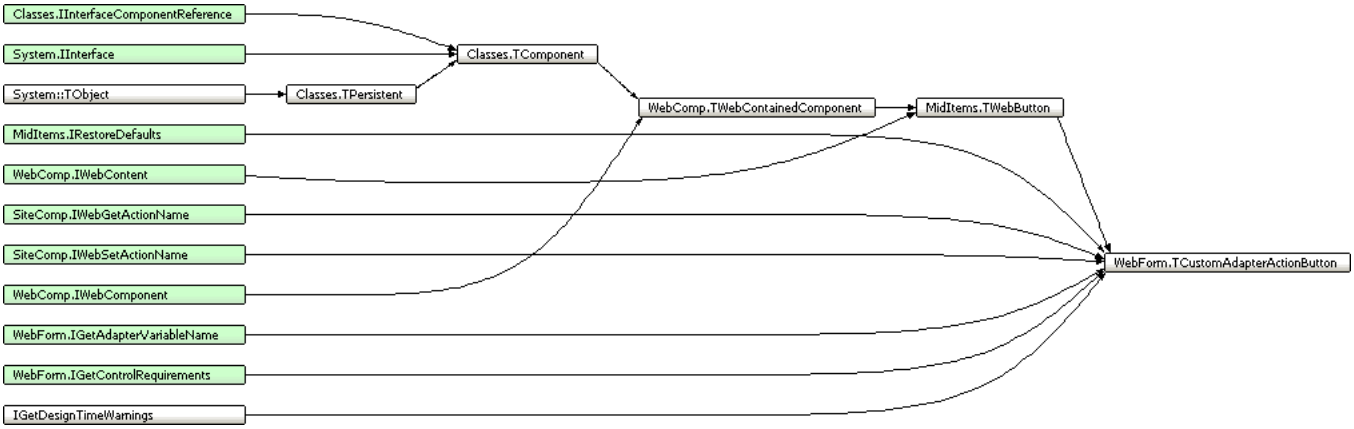

Pascal

```
TCustomAdapterActionButton = class(TWebButton, IWebGetActionName, IWebSetActionName,
IRestoreDefaults, IGetControlRequirements, IGetAdapterVariableName, IGetDesignTimeWarnings);
```

C++

```
class TCustomAdapterActionButton : public TWebButton, public IWebGetActionName, public
IWebSetActionName, public IRestoreDefaults, public IGetControlRequirements, public
IGetAdapterVariableName, public IGetDesignTimeWarnings;
```

File

WebForm

Description

Use TCustomAdapterActionButton as a base class when defining Web items that generate HTML controls for adapter actions. TCustomAdapterActionButton descendants are added to Web items that appear in the hierarchy of a TAdapterForm (see page 25159) object. The adapter action button generates an HTML control that appears in the HTML form. When the end user clicks that control, it sends a request message that executes the adapter action.

TCustomAdapterActionButton introduces properties to link this component to the adapter action it executes, and for defining the type and appearance of the generated control in the HTML form.

See Also

TAdapterActionButton (see page 25034)

TCustomAdapterAction (see page 24188)

TAdapterForm (see page 25159)

181.1.19.1 TCustomAdapterActionButton Members

The following tables list the members exposed by TCustomAdapterActionButton.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16868)	Returns the HTML that defines the button control.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FindAssociatedAction (see page 25195)	This is FindAssociatedAction, a member of class TCustomAdapterActionButton.
	GetActionName (see page 25196)	This is GetActionName, a member of class TCustomAdapterActionButton.
	GetAdapterModeOfAdapter (see page 25196)	This is GetAdapterModeOfAdapter, a member of class TCustomAdapterActionButton.
	GetAdapterVariableName (see page 25196)	This is GetAdapterVariableName, a member of class TCustomAdapterActionButton.
	GetCaption (see page 25196)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetControlRequirements (see page 25197)	This is GetControlRequirements, a member of class TCustomAdapterActionButton.
	GetDesigntimeWarnings (see page 25197)	This is GetDesigntimeWarnings, a member of class TCustomAdapterActionButton.
	GetHTMLFormVarName (see page 25197)	This is GetHTMLFormVarName, a member of class TCustomAdapterActionButton.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetLayoutAttributes (see page 25197)	Returns the layout object that specifies how the generated control should look.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	ImplContent (see page 25198)	Provides the underlying implementation for the Content method.
	ImplGetControlRequirements (see page 25198)	This is ImplGetControlRequirements, a member of class TCustomAdapterActionButton.
	ImplRestoreDefaults (see page 25198)	This is ImplRestoreDefaults, a member of class TCustomAdapterActionButton.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 25199)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 25199)	This is RestoreDefaults, a member of class TCustomAdapterActionButton.
	SetActionName (see page 25199)	This is SetActionName, a member of class TCustomAdapterActionButton.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.

V	SetCaption (see page 25200)	Sets the value of the Caption property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 25194)	Creates and initializes an instance of TCustomAdapterActionButton (see page 25189).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25195)	Frees an instance of TCustomAdapterActionButton (see page 25189).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	ActionName (see page 25200)	Specifies the name of the action taken by the Web button.
R	AdapterVariableName (see page 25308)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	Align (see page 25201)	Specifies the horizontal alignment of the generated HTML control.
	Caption (see page 16870)	Specifies the caption that appears on the button.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.

W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16871)	Represents a tag option and its value in the generated INPUT tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayColumns (see page 25201)	Indicates the number of columns to use when laying out the generated HTML control.
	DisplayType (see page 25202)	Specifies the DisplayType of the generated HTML control.
	ErrorPageName (see page 25202)	Specifies the name of the page to display when an error occurs executing the action.
	HideOptions (see page 25202)	Controls (see page 5711) when the generated HTML element appears in HTML forms.
	ImageHeight (see page 25203)	Specifies the height of the generated element if it is a clickable image.
	ImageProducer (see page 25203)	Specifies the interface that this adapter action button uses to generate images.
	ImageWidth (see page 25204)	Specifies the width of the generated element if it is a clickable image.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	PageName (see page 25204)	Specifies the name of the page to display after successfully executing the action.
	Style (see page 16871)	Adds style attributes to the generated button control.
	StyleRule (see page 16872)	Applies a style in the page producer's style sheet to the button.
	VAlign (see page 25204)	Specifies the vertical alignment of the generated HTML control.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

181.1.19.2 TCustomAdapterActionButton.Create Constructor

Creates and initializes an instance of TCustomAdapterActionButton (see page 25189).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterActionButton(TComponent * AOwner);
```

Description

Call Create to instantiate an adapter action button at runtime. Adapter action buttons added to an adapter form at design time are created automatically.

AOwner is the component, typically a Web module, that is responsible for freeing the TCustomAdapterActionButton (see page 25189) instance. This becomes the value of the Owner property.

See Also

Owner (see page 2807)

181.1.19.3 TCustomAdapterActionButton.Destroy Destructor

Frees an instance of TCustomAdapterActionButton (see page 25189).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterActionButton();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks that the TCustomAdapterActionButton (see page 25189) is not nil before calling Destroy.

See Also

Free (see page 22300)

181.1.19.4 TCustomAdapterActionButton Methods

The methods of the TCustomAdapterActionButton class are listed here.

Protected Methods

	Name	Description
	FindAssociatedAction (see page 25195)	This is FindAssociatedAction, a member of class TCustomAdapterActionButton.
	GetActionName (see page 25196)	This is GetActionName, a member of class TCustomAdapterActionButton.
	GetAdapterModeOfAdapter (see page 25196)	This is GetAdapterModeOfAdapter, a member of class TCustomAdapterActionButton.
	GetAdapterVariableName (see page 25196)	This is GetAdapterVariableName, a member of class TCustomAdapterActionButton.
V	GetCaption (see page 25196)	Returns the value of the Caption property.
	GetControlRequirements (see page 25197)	This is GetControlRequirements, a member of class TCustomAdapterActionButton.
	GetDesigntimeWarnings (see page 25197)	This is GetDesigntimeWarnings, a member of class TCustomAdapterActionButton.
	GetHTMLFormVarName (see page 25197)	This is GetHTMLFormVarName, a member of class TCustomAdapterActionButton.
V	GetLayoutAttributes (see page 25197)	Returns the layout object that specifies how the generated control should look.
V	ImplContent (see page 25198)	Provides the underlying implementation for the Content method.
V	ImplGetControlRequirements (see page 25198)	This is ImplGetControlRequirements, a member of class TCustomAdapterActionButton.
V	ImplRestoreDefaults (see page 25198)	This is ImplRestoreDefaults, a member of class TCustomAdapterActionButton.
V	Notification (see page 25199)	Forwards notification messages to all owned components.
	RestoreDefaults (see page 25199)	This is RestoreDefaults, a member of class TCustomAdapterActionButton.
	SetActionName (see page 25199)	This is SetActionName, a member of class TCustomAdapterActionButton.
V	SetCaption (see page 25200)	Sets the value of the Caption property.

181.1.19.4.1 TCustomAdapterActionButton.FindAssociatedAction Method

Pascal

```
function FindAssociatedAction: TComponent;
```

C++

```
__fastcall TComponent * FindAssociatedAction();
```

Description

This is FindAssociatedAction, a member of class TCustomAdapterActionButton.

181.1.19.4.2 TCustomAdapterActionButton.GetActionName Method

Pascal

```
function GetActionName: string;
```

C++

```
__fastcall AnsiString GetActionName();
```

Description

This is GetActionName, a member of class TCustomAdapterActionButton.

181.1.19.4.3 TCustomAdapterActionButton.GetAdapterModeOfAdapter Method

Pascal

```
function GetAdapterModeOfAdapter: string;
```

C++

```
__fastcall AnsiString GetAdapterModeOfAdapter();
```

Description

This is GetAdapterModeOfAdapter, a member of class TCustomAdapterActionButton.

181.1.19.4.4 TCustomAdapterActionButton.GetAdapterVariableName Method

Pascal

```
function GetAdapterVariableName: string;
```

C++

```
__fastcall AnsiString GetAdapterVariableName();
```

Description

This is GetAdapterVariableName, a member of class TCustomAdapterActionButton.

181.1.19.4.5 TCustomAdapterActionButton.GetCaption Method

Returns the value of the Caption property.

Pascal

```
function GetCaption: string; override;
```

C++

```
virtual __fastcall AnsiString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption property.

See Also

Caption (see page 16870)

SetCaption (see page 16870)

181.1.19.4.6 TCustomAdapterActionButton.GetControlRequirements Method

Pascal

```
procedure GetControlRequirements(ARequirements: TAbstractControlRequirements);
```

C++

```
__fastcall GetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is GetControlRequirements, a member of class TCustomAdapterActionButton.

181.1.19.4.7 TCustomAdapterActionButton.GetDesigntimeWarnings Method

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

This is GetDesigntimeWarnings, a member of class TCustomAdapterActionButton.

181.1.19.4.8 TCustomAdapterActionButton.GetHTMLFormVarName Method

Pascal

```
function GetHTMLFormVarName: string;
```

C++

```
__fastcall AnsiString GetHTMLFormVarName();
```

Description

This is GetHTMLFormVarName, a member of class TCustomAdapterActionButton.

181.1.19.4.9 TCustomAdapterActionButton.GetLayoutAttributes Method

Returns the layout object that specifies how the generated control should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content method) to take into account layout properties of the child component.

As implemented by TWebButton, GetLayoutAttributes returns nil (Delphi) or NULL (C++), because the layout of buttons does not make use of layout attributes.

See Also

Content (see page 16868)

181.1.19.4.10 TCustomAdapterActionButton.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML that defines the button control. TWebButton introduces ImplContent as an abstract or, in C++ terminology, pure virtual method. TWebButton descendants must override ImplContent to generate the definition of their particular button control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web button should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the button control.

See Also

Content (see page 16868)

ILayoutWebContent (see page 24774)

181.1.19.4.11 TCustomAdapterActionButton.ImplGetControlRequirements Method

Pascal

```
procedure ImplGetControlRequirements(ARequirements: TAbstractControlRequirements); virtual;
```

C++

```
virtual __fastcall ImplGetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is ImplGetControlRequirements, a member of class TCustomAdapterActionButton.

181.1.19.4.12 TCustomAdapterActionButton.ImplRestoreDefaults Method

Pascal

```
procedure ImplRestoreDefaults; virtual;
```

C++

```
virtual __fastcall ImplRestoreDefaults();
```

Description

This is ImplRestoreDefaults, a member of class TCustomAdapterActionButton.

181.1.19.4.13 TCustomAdapterActionButton.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

181.1.19.4.14 TCustomAdapterActionButton.RestoreDefaults Method

Pascal

```
procedure RestoreDefaults;
```

C++

```
__fastcall RestoreDefaults();
```

Description

This is RestoreDefaults, a member of class TCustomAdapterActionButton.

181.1.19.4.15 TCustomAdapterActionButton.SetActionName Method

Pascal

```
procedure SetActionName(const AValue: string);
```

C++

```
__fastcall SetActionName(const AnsiString AValue);
```

Description

This is SetActionName, a member of class TCustomAdapterActionButton.

181.1.19.4.16 TCustomAdapterActionButton.SetCaption Method

Sets the value of the Caption property.

Pascal

```
procedure SetCaption(const Value: string); override;
```

C++

```
virtual __fastcall SetCaption(const AnsiString Value);
```

Description

SetCaption is the protected write implementation for the Caption property.

See Also

Caption (see page 16870)

GetCaption (see page 16869)

181.1.19.5 TCustomAdapterActionButton Properties

The properties of the TCustomAdapterActionButton class are listed here.

Public Properties

	Name	Description
	ActionName (see page 25200)	Specifies the name of the action taken by the Web button.
	Align (see page 25201)	Specifies the horizontal alignment of the generated HTML control.
	DisplayColumns (see page 25201)	Indicates the number of columns to use when laying out the generated HTML control.
	DisplayType (see page 25202)	Specifies the DisplayType of the generated HTML control.
	ErrorPageName (see page 25202)	Specifies the name of the page to display when an error occurs executing the action.
	HideOptions (see page 25202)	Controls (see page 5711) when the generated HTML element appears in HTML forms.
	ImageHeight (see page 25203)	Specifies the height of the generated element if it is a clickable image.
	ImageProducer (see page 25203)	Specifies the interface that this adapter action button uses to generate images.
	ImageWidth (see page 25204)	Specifies the width of the generated element if it is a clickable image.
	PageName (see page 25204)	Specifies the name of the page to display after successfully executing the action.
	VAlign (see page 25204)	Specifies the vertical alignment of the generated HTML control.

181.1.19.5.1 TCustomAdapterActionButton.ActionName Property

Specifies the name of the action taken by the Web button.

Pascal

```
property ActionName: string;
```

C++

```
__property AnsiString ActionName;
```

Description

Use ActionName to get or set the name of the adapter action that executes in response to requests sent when the user clicks this object's generated control. ActionName is the same as the ActionName property of the adapter action object that this control executes.

See Also

Caption

ActionName (see page 24198)

181.1.19.5.2 TCustomAdapterActionButton.Align Property

Specifies the horizontal alignment of the generated HTML control.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align controls where the generated HTML control is positioned horizontally within its parent HTML element. If the adapter action button is included in the column of an adapter grid, this property has no effect unless the associated action has multiple items. It affects the position of the HTML controls for each individual of the items within the grid generated to position them into rows.

Align indicates the desired value of the Align option of the HTML tag for the generated control.

The default value of Align is haDefault, which means the web browser that reads the HTML document horizontally positions the control wherever it wishes.

See Also

VAlign (see page 25204)

DisplayColumns (see page 25201)

ItemCount (see page 24074)

181.1.19.5.3 TCustomAdapterActionButton.DisplayColumns Property

Indicates the number of columns to use when laying out the generated HTML control.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

An adapter action may represent multiple commands. When the TCustomAdapterActionButton (see page 25189) object generates controls for such an action, it must handle the layout of individual items for each of the action's commands. DisplayColumns controls how many items (controls) appear in a row before TCustomAdapterActionButton (see page 25189) starts a new row of controls.

If the corresponding adapter action does not represent multiple commands (that is, if its ItemCount property is not greater than 1), the value of DisplayColumns has no effect.

If DisplayColumns is –1, the Web browser controls the layout of the adapter action's items.

See Also

ItemCount (see page 24074)

DisplayType (see page 25202)

181.1.19.5.4 TCustomAdapterActionButton.DisplayType Property

Specifies the DisplayType of the generated HTML control.

Pascal

```
property DisplayType: TCommandHTMLElementType;
```

C++

```
__property TCommandHTMLElementType DisplayType;
```

Description

DisplayType controls the type of HTML element that this Web item generates.

When DisplayType is ctDefault (the default), TCustomAdapterActionButton (see page 25189) uses the element type that the associated adapter action suggests. Change DisplayType to generate a different type of control or set of controls.

When DisplayType is ctImage or ctEventImages, you must set the ImageProducer (see page 25203) property to a component that can provide the image or images.

See Also

ImageProducer (see page 25203)

Caption

GetDisplayStyle (see page 24056)

181.1.19.5.5 TCustomAdapterActionButton.ErrorPageName Property

Specifies the name of the page to display when an error occurs executing the action.

Pascal

```
property ErrorPageName: string;
```

C++

```
__property AnsiString ErrorPageName;
```

Description

Use ErrorPageName to indicate the Web page that should appear if the WebSnap application encounters an error when executing the adapter action in response to the user clicking the generated control. This is the name used to identify the Web page in server-side scripts, and appears as the pathinfo portion of the URL for that page if it is published.

See Also

PageName (see page 25204)

pagename

181.1.19.5.6 TCustomAdapterActionButton.HideOptions Property

Controls (see page 5711) when the generated HTML element appears in HTML forms.

Pascal

```
property HideOptions: TActionButtonHideOptions;
```

C++

```
__property TActionButtonHideOptions HideOptions;
```

Description

HideOptions controls when this Web item generates an HTML control for its associated action. This property is a set of flags that control when the adapter page producer suppresses the controls that this object generates. If HideOptions is an empty set, TCustomAdapterActionButton (see page 25189) always generates a control for its associated adapter action.

See Also

GetVisible (see page 24061)

WebEnabled (see page 24071)

HasExecuteAccess (see page 24062)

181.1.19.5.7 TCustomAdapterActionButton.ImageHeight Property

Specifies the height of the generated element if it is a clickable image.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

ImageHeight controls the height of the image that the user can click to execute the associated action. If ImageHeight is –1 (the default), the browser controls the height it uses to display the generated image.

ImageHeight is only used when the value of the DisplayType (see page 25202) property is ctImage or ctEventImages.

See Also

ImageProducer (see page 25203)

ImageWidth (see page 25204)

DisplayType (see page 25202)

181.1.19.5.8 TCustomAdapterActionButton.ImageProducer Property

Specifies the interface that this adapter action button uses to generate images.

Pascal

```
property ImageProducer: IActionImageProducer;
```

C++

```
__property IActionImageProducer ImageProducer;
```

Description

Set ImageProducer when the value of the DisplayType (see page 25202) is ctImage or ctEventImages. ImageProducer is the interface of the component that supplies the image or images that appear in HTML forms as the control that this Web item generates. When the user clicks on the generated image in the HTML form, the associated action executes.

See Also

DisplayType (see page 25202)

ImageHeight (see page 25203)

ImageWidth (see page 25204)

181.1.19.5.9 TCustomAdapterActionButton.ImageWidth Property

Specifies the width of the generated element if it is a clickable image.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

ImageWidth controls the width of the image that the user can click to execute the associated action. If ImageWidth is –1 (the default), the browser controls the width it uses to display the generated image.

ImageWidth is only used when the value of the DisplayType (see page 25202) property is ctImage or ctEventImages.

See Also

ImageProducer (see page 25203)

ImageHeight (see page 25203)

DisplayType (see page 25202)

181.1.19.5.10 TCustomAdapterActionButton.PageName Property

Specifies the name of the page to display after successfully executing the action.

Pascal

```
property PageName: string;
```

C++

```
__property AnsiString PageName;
```

Description

Use PageName to indicate the Web page that should appear if the WebSnap application successfully executes the adapter action in response to the user clicking the generated control. This is the name used to identify the Web page in server-side scripts, and appears as the pathinfo portion of the URL for that page if it is published.

See Also

ErrorPageName (see page 25202)

pagename

181.1.19.5.11 TCustomAdapterActionButton.VAlign Property

Specifies the vertical alignment of the generated HTML control.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign controls where the generated HTML control is positioned vertically within its parent HTML element. VAlign indicates the desired value of the VAlign option of the HTML tag for the generated control. . If the adapter action button is included in the column of an adapter grid, this property has no effect unless the associated action has multiple items. It affects the position of the HTML controls for each of the individual items within the grid generated to position them into rows.

The default value of VAlign is haVDefault, which means the web browser that reads the HTML document vertically positions the control wherever it wishes.

See Also

Align (see page 25201)

DisplayColumns (see page 25201)

ItemCount (see page 24074)

181.1.20 TCustomAdapterCommandColumn Class

TCustomAdapterCommandColumn is the base class for components that generate a column of action controls in an adapter grid.

Class Hierarchy

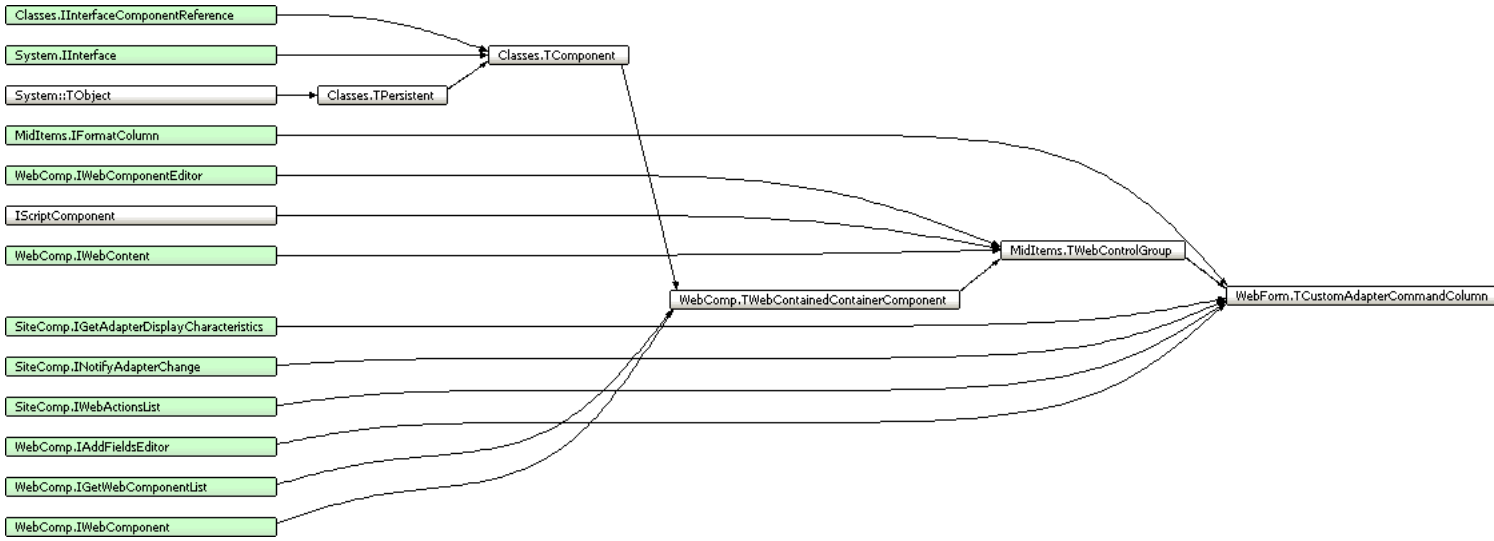

Pascal

```
TCustomAdapterCommandColumn = class(TWebControlGroup, IWebActionsList, IAddFieldsEditor,
IFormatColumn, IGetAdapterDisplayCharacteristics, INotifyAdapterChange);
```

C++

```
class TCustomAdapterCommandColumn : public TWebControlGroup, public IWebActionsList, public
IAddFieldsEditor, public IFormatColumn, public IGetAdapterDisplayCharacteristics, public
INotifyAdapterChange;
```

File

WebForm

Description

Use TCustomAdapterCommandColumn as a base class when creating a specialized display control that organizes the output of several adapter action buttons (TAdapterActionButton (see page 25034) objects) into a column of an adapter grid (TAdapterGrid (see page 25175)). TCustomAdapterCommandColumn generates the HTML to lay out a single column of an adapter grid. Each cell in the column can contain one or more adapter action buttons, each of which executes an adapter action. For each row in the grid, the column contains duplicates of the same set of adapter action buttons.

Use the TCustomAdapterCommandColumn properties to specify which adapter action buttons appear in the column and to customize their layout and appearance. These properties include:

The AddDefaultCommands (see page 25215), WebFieldControls, and VisibleCommands (see page 25219) properties, which specify which adapter action buttons appear in the column.

The Caption (see page 25216) and CaptionAttributes (see page 25217) properties, which determine the appearance of the label at the top of the column.

The Align (see page 25216), VAlign (see page 25219), and DisplayColumns (see page 25217) properties, which determine the layout of controls inside each cell.

The Style (see page 25218) and StyleRule (see page 25218) properties, which let you apply a style from a stylesheet to the column.

See Also

TAdapterCommandColumn (see page 25052)

TAdapterGrid (see page 25175)

TAdapterEditColumn (see page 25132)

TAdapterDisplayColumn (see page 25099)

TAdapterActionButton (see page 25034)

181.1.20.1 TCustomAdapterCommandColumn Members

The following tables list the members exposed by TCustomAdapterCommandColumn.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.

	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FormatColumnData (see page 25211)	Generates the HTML for the cells in the column that contain controls to execute actions.
	FormatColumnHeading (see page 25212)	Generates the HTML for the table header cell for this column.
	GetAdapterDisplayCharacteristics (see page 25212)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterCommandColumn.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetLayoutAttributes (see page 25212)	Returns an object that specifies how the generated group of controls should look.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
	ImplCanAddClass (see page 25213)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 25213)	Provides the underlying implementation for the Content method.
	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	NotifyAdapterChange (see page 25214)	Responds to changes that affect the associated adapter.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.

	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	AdapterActionsListHelper (see page 25215)	Identifies the helper object that enables this component to work with the editor at design time.
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 25210)	Creates and initializes an instance of TCustomAdapterCommandColumn (see page 25205).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25211)	Frees an instance of TCustomAdapterCommandColumn (see page 25205).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAddAllFieldsItem (see page 24770)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 24770)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 24770)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNewFieldPrefix (see page 24771)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	AddDefaultCommands (see page 25215)	Indicates whether cells in the column contain controls for all of the adapter's default actions if you do not explicitly add items to the WebFieldControls property.
	Align (see page 25216)	Specifies the horizontal alignment of the action control buttons within the column.
	Caption (see page 25216)	Specifies the text label that appears at the top of the column.
	CaptionAttributes (see page 25217)	Specifies the attributes of the Caption (see page 25216) property.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 25217)	Represents a tag option and its value.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayColumns (see page 25217)	Specifies the number of adapter action buttons that can appear in a single row.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 25218)	Adds style attributes to the generated caption.
	StyleRule (see page 25218)	Applies a style in the page producer's style sheet to the generated column.
	VAlign (see page 25219)	Specifies the vertical alignment of the action control buttons within the column.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleCommands (see page 25219)	Lists the child Web items that generate controls that appear in cells of the column.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

181.1.20.2 TCustomAdapterCommandColumn.Create Constructor

Creates and initializes an instance of TCustomAdapterCommandColumn (see page 25205).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterCommandColumn(TComponent * AOwner);
```

Description

Call Create to instantiate TCustomAdapterCommandColumn (see page 25205) at runtime. TCustomAdapterCommandColumn (see page 25205) descendants added at design time are instantiated automatically.

AOwner is the component that becomes the value of the Owner property. Typically, this is the Web module.

See Also

Owner (see page 2807)

181.1.20.3 TCustomAdapterCommandColumn.Destroy Destructor

Frees an instance of TCustomAdapterCommandColumn (see page 25205).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterCommandColumn();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks that the TCustomAdapterCommandColumn (see page 25205) reference is not nil before calling Destroy.

See Also

Free (see page 22300)

181.1.20.4 TCustomAdapterCommandColumn Methods

The methods of the TCustomAdapterCommandColumn class are listed here.

Protected Methods

	Name	Description
	FormatColumnData (see page 25211)	Generates the HTML for the cells in the column that contain controls to execute actions.
	FormatColumnHeading (see page 25212)	Generates the HTML for the table header cell for this column.
	GetAdapterDisplayCharacteristics (see page 25212)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterCommandColumn.
V	GetLayoutAttributes (see page 25212)	Returns an object that specifies how the generated group of controls should look.
V	ImplCanAddClass (see page 25213)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 25213)	Provides the underlying implementation for the Content method.
	NotifyAdapterChange (see page 25214)	Responds to changes that affect the associated adapter.

181.1.20.4.1 TCustomAdapterCommandColumn.FormatColumnData Method

Generates the HTML for the cells in the column that contain controls to execute actions.

Pascal

```
function FormatColumnData(const Content: string; Options: TWebContentOptions): string;
```

C++

```
__fastcall AnsiString FormatColumnData(const AnsiString Content, TWebContentOptions Options);
```

Description

The adapter grid calls FormatColumnData to update the HTML for the contents of a cell to include any column attributes that affect the layout of that content. This method adds attributes to the HTML that reflect the Align (see page 25216) and VAlign (see page 25219) properties.

Content is the HTML that represents the content of the cell. It is the value that the Content method returns.

Options contains a set of flags that limit what can be included in the generated HTML.

FormatColumnData returns the resulting HTML.

See Also

Content (see page 16879)

TAdapterGrid (see page 25175)

Align (see page 25216)

VAlign (see page 25219)

181.1.20.4.2 TCustomAdapterCommandColumn.FormatColumnHeading Method

Generates the HTML for the table header cell for this column.

Pascal

```
function FormatColumnHeading(Options: TWebContentOptions): string;
```

C++

```
__fastcall AnsiString FormatColumnHeading(TWebContentOptions Options);
```

Description

The adapter grid calls FormatColumnHeading to generate the HTML for the top cell of the column (the one that contains the caption). This method generates the HTML to reflect the Caption (see page 25216) and CaptionAttributes (see page 25217) properties.

Options contains a set of flags that limit what can be included in the generated HTML.

FormatColumnHeading returns the resulting HTML.

See Also

Caption (see page 25216)

TAdapterGrid (see page 25175)

CaptionAttributes (see page 25217)

181.1.20.4.3 TCustomAdapterCommandColumn.GetAdapterDisplayCharacteristics Method

Pascal

```
function GetAdapterDisplayCharacteristics: TAdapterDisplayCharacteristics;
```

C++

```
__fastcall TAdapterDisplayCharacteristics GetAdapterDisplayCharacteristics();
```

Description

This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterCommandColumn.

181.1.20.4.4 TCustomAdapterCommandColumn.GetLayoutAttributes Method

Returns an object that specifies how the generated group of controls should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content method) to take into account layout properties of the child component.

GetLayoutAttributes adds returns a TLayoutAttributes object that encodes the values of the Style (see page 25218), StyleRule (see page 25218), and Custom (see page 25217) properties.

See Also

Content (see page 16879)

Style (see page 25218)

StyleRule (see page 25218)

Custom (see page 25217)

181.1.20.4.5 TCustomAdapterCommandColumn.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TCustomAdapterCommandColumn (see page 25205) descendant can add a component of type AClass to its WebFieldControls property list.

AParent is the TCustomAdapterCommandColumn (see page 25205) instance to which the component will be added.

AClass is the class of the component to be added.

In TCustomAdapterCommandColumn (see page 25205), ImplCanAddClass returns true if AClass is a descendant of TCustomAdapterActionButton (see page 25189) or TCustomLayoutGroup.

See Also

CanAddClass (see page 16878)

WebFieldControls (see page 16884)

TCustomAdapterActionButton (see page 25189)

TCustomLayoutGroup (see page 16528)

181.1.20.4.6 TCustomAdapterCommandColumn.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML for the controls in one cell of the column. ImplContent generates the HTML to display the controls that the VisibleCommands (see page 25219) property lists, reflecting the values of the DisplayColumns (see page 25217), .Style (see page 25218), StyleRule (see page 25218), and Custom (see page 25217) properties.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TCustomAdapterCommandColumn (see page 25205) should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

Note: ImplContent does not include attributes for the alignment of controls inside the cell. The FormatColumnData (see page 25211) method adds this information to the HTML that ImplContent generates.

See Also

Content (see page 16879)

FormatColumnData (see page 25211)

ILayoutWebContent (see page 24774)

VisibleCommands (see page 25219)

DisplayColumns (see page 25217)

Style (see page 25218)

StyleRule (see page 25218)

Custom (see page 25217)

181.1.20.4.7 TCustomAdapterCommandColumn.NotifyAdapterChange Method

Responds to changes that affect the associated adapter.

Pascal

```
procedure NotifyAdapterChange(Sender: TComponent);
```

C++

```
__fastcall NotifyAdapterChange(TComponent * Sender);
```

Description

NotifyAdapterChange is called automatically when the associated adapter changes. This method updates the set of default actions for that adapter to reflect the current change, and passes on the change notification to the actions that the WebFieldControls property lists.

See Also

GetDefaultWebComponents (see page 16880)

WebFieldControls (see page 16884)

181.1.20.5 TCustomAdapterCommandColumn Properties

The properties of the TCustomAdapterCommandColumn class are listed here.

Protected Properties

	Name	Description
R	AdapterActionsListHelper (see page 25215)	Identifies the helper object that enables this component to work with the editor at design time.

Public Properties

	Name	Description
	AddDefaultCommands (see page 25215)	Indicates whether cells in the column contain controls for all of the adapter's default actions if you do not explicitly add items to the WebFieldControls property.
	Align (see page 25216)	Specifies the horizontal alignment of the action control buttons within the column.
	Caption (see page 25216)	Specifies the text label that appears at the top of the column.
	CaptionAttributes (see page 25217)	Specifies the attributes of the Caption (see page 25216) property.
	Custom (see page 25217)	Represents a tag option and its value.
	DisplayColumns (see page 25217)	Specifies the number of adapter action buttons that can appear in a single row.
	Style (see page 25218)	Adds style attributes to the generated caption.
	StyleRule (see page 25218)	Applies a style in the page producer's style sheet to the generated column.
	VAlign (see page 25219)	Specifies the vertical alignment of the action control buttons within the column.
R	VisibleCommands (see page 25219)	Lists the child Web items that generate controls that appear in cells of the column.

181.1.20.5.1 TCustomAdapterCommandColumn.AdapterActionsListHelper Property

Identifies the helper object that enables this component to work with the editor at design time.

Pascal

```
property AdapterActionsListHelper: TAdapterActionsListHelper;
```

C++

```
__property TAdapterActionsListHelper AdapterActionsListHelper;
```

Description

AdapterActionsListHelper provides read-only access to a helper object that implements design-time editing functions on behalf of this component. This helper object implements methods for locating child Web items for specified actions, for listing the child Web items of this component, and for specifying the text of menu commands that operate on this component's list of adapter action buttons.

See Also

WebFieldControls (see page 16884)

181.1.20.5.2 TCustomAdapterCommandColumn.AddDefaultCommands Property

Indicates whether cells in the column contain controls for all of the adapter's default actions if you do not explicitly add items to the WebFieldControls property.

Pascal

```
property AddDefaultCommands: Boolean;
```

C++

```
__property Boolean AddDefaultCommands;
```

Description

Set AddDefaultCommands property to specify what should appear in the generated column when you do not explicitly add any adapter action buttons to the WebFieldControls property. When AddDefaultCommands is true (the default), the adapter command column automatically adds controls for every default action of the associated adapter if you do not explicitly set the WebFieldControls property. When AddDefaultCommands is false, the adapter command column only generates controls if you explicitly add them to the WebFieldControls property. AddDefaultCommands must be false to allow an empty column.

See Also

WebFieldControls (see page 16884)

DefaultWebComponents (see page 23778)

VisibleCommands (see page 25219)

181.1.20.5.3 TCustomAdapterCommandColumn.Align Property

Specifies the horizontal alignment of the action control buttons within the column.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align controls where the child HTML control are positioned horizontally within this column. Align indicates the desired value of the Align option of the HTML tag.

The default value of Align is haDefault, which means the web browser that reads the HTML document horizontally positions the control wherever it wishes.

See Also

VAlign (see page 25219)

DisplayColumns (see page 25217)

181.1.20.5.4 TCustomAdapterCommandColumn.Caption Property

Specifies the text label that appears at the top of the column.

Pascal

```
property Caption: string;
```

C++

```
__property AnsiString Caption;
```

Description

Use Caption to get or set the caption for the column in the adapter grid. The caption is the text label that appears at the top of the column.

To change the appearance or layout of the caption, use the CaptionAttribute property.

See Also

CaptionAttributes (see page 25217)

181.1.20.5.5 TCustomAdapterCommandColumn.CaptionAttributes Property

Specifies the attributes of the Caption (see page 25216) property.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the column caption.

See Also

Caption (see page 25216)

181.1.20.5.6 TCustomAdapterCommandColumn.Custom Property

Represents a tag option and its value.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to add one or more options to the tag that surrounds the column. Each option and its value has the form

`OPTIONNAME=Value`

The options specified by Custom should be separated by spaces.

TCustomAdapterCommandColumn (see page 25205) inserts the Custom string into the HTML tag when it generates its content.

See Also

Style (see page 25218)

StyleRule (see page 25218)

181.1.20.5.7 TCustomAdapterCommandColumn.DisplayColumns Property

Specifies the number of adapter action buttons that can appear in a single row.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Set DisplayColumns to organize the layout of adapter action buttons when the WebFieldControls property lists more than one adapter action button. TCustomAdapterCommandColumn (see page 25205) places the adapter action buttons by rows in the cells of the adapter grid column. DisplayColumns limits the number of controls that can appear in a single row within each column cell. If the WebFieldControls property lists more adapter action buttons than DisplayColumns, then the cell contains multiple rows of controls.

See Also

WebFieldControls (see page 16884)

181.1.20.5.8 TCustomAdapterCommandColumn.Style Property

Adds style attributes to the generated caption.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to specify display characteristics that affect the generated column. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the HTML tag that generates the column as the value of a STYLE option.

Note: Style affects the contents of the column cells. To specify display characteristics of the column's caption, use the CaptionAttributes (see page 25217) property instead.

See Also

Custom (see page 25217)

StyleRule (see page 25218)

CaptionAttributes (see page 25217)

181.1.20.5.9 TCustomAdapterCommandColumn.StyleRule Property

Applies a style in the page producer's style sheet to the generated column.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated column. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

To apply a style definition that is not included in the style sheet, use the Style (see page 25218) property instead.

See Also

Custom (see page 25217)

Style (see page 25218)

Styles (see page 19830)

StylesFile (see page 19831)

181.1.20.5.10 TCustomAdapterCommandColumn.VAlign Property

Specifies the vertical alignment of the action control buttons within the column.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign controls where the child HTML control are positioned vertically within the cells of this column. VAlign indicates the desired value of the VAlign option of the HTML tag.

The default value of VAlign is haVDefault, which means the web browser that reads the HTML document vertically positions the control wherever it wishes.

See Also

Align (see page 25216)

DisplayColumns (see page 25217)

181.1.20.5.11 TCustomAdapterCommandColumn.VisibleCommands Property

Lists the child Web items that generate controls that appear in cells of the column.

Pascal

```
property VisibleCommands: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleCommands;
```

Description

VisibleCommands is the list of child Web items that generate HTML controls for the cells of the adapter command column. Each element in this list is an adapter action button (TAdapterActionButton (see page 25034) object).

If you have explicitly added child Web items to this component, VisibleCommands is the same as the value of the WebFieldControls property. Otherwise, if the AddDefaultCommands (see page 25215) property is true, VisibleCommands lists adapter action buttons for the default actions of the associated adapter. If AddDefaultCommands (see page 25215) is false, VisibleCommands is an empty list.

See Also

WebFieldControls (see page 16884)

AddDefaultCommands (see page 25215)

TAdapterActionButton (see page 25034)

181.1.21 TCustomAdapterCommandGroup Class

TCustomAdapterCommandGroup is the base class for TAdapterCommandGroup (see page 25062).

Class Hierarchy

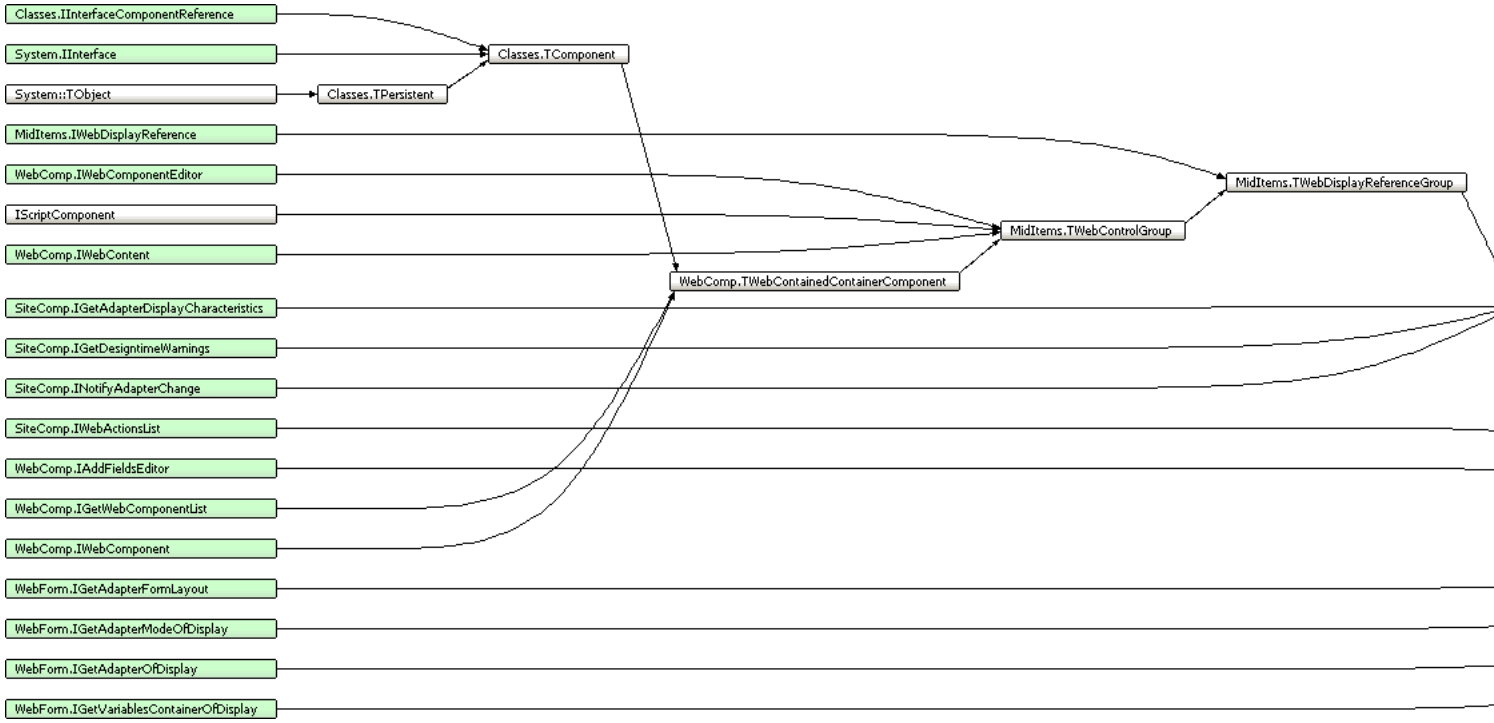

Pascal

```
TCustomAdapterCommandGroup = class(TAdapterDisplayReferenceGroup, IWebActionsList,
IGetAdapterOfDisplay, IGetVariablesContainerOfDisplay, IAddFieldsEditor,
IGetAdapterModeOfDisplay, IGetAdapterFormLayout);
```

C++

```
class TCustomAdapterCommandGroup : public TAdapterDisplayReferenceGroup, public
IWebActionsList, public IGetAdapterOfDisplay, public IGetVariablesContainerOfDisplay, public
IAddFieldsEditor, public IGetAdapterModeOfDisplay, public IGetAdapterFormLayout;
```

File

WebForm

Description

Derive from TCustomAdapterCommandGroup to define a component that generates a group of HTML buttons or other HTML elements that execute adapter actions.

See Also

TAdapterDisplayReferenceGroup (see page 25122)

TAdapterCommandGroup (see page 25062)

TAdapterForm (see page 25159)

TAdapterFieldGroup (see page 25150)

TAdapterGrid (see page 25175)

TAdapterActionButton (see page 25034)

181.1.21.1 TCustomAdapterCommandGroup Members

The following tables list the members exposed by TCustomAdapterCommandGroup.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AdapterChange (see page 25128)	Responds to changes in the adapter on which the controls in this group operate.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DisplayComponentChange (see page 25129)	Responds when the DisplayComponent (see page 25132) property changes.
	GetAdapter (see page 25226)	This is GetAdapter, a member of class TCustomAdapterCommandGroup.
	GetAdapterDisplayCharacteristics (see page 25129)	Indicates the display characteristics of the HTML this component generates.
	GetAdapterFormLayout (see page 25226)	This is GetAdapterFormLayout, a member of class TCustomAdapterCommandGroup.
	GetAdapterModeOfAdapter (see page 25227)	Returns the adapter mode defined by the adapter component.
	GetAdapterModeOfDisplay (see page 25227)	Returns the adapter mode defined by the TAdapterPageProducer components.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetDesigntimeWarnings (see page 25130)	Populates a design-time warnings object with any warnings that the reference group needs to emit.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.

	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	GetWebDisplayComponent (see page 16920)	This is GetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
V	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
V	ImplCanAddClass (see page 25227)	Checks the inheritance of the given component.
V	ImplContent (see page 25228)	Lays out the buttons for the command group using the TAdapterFormLayout (see page 25166) component.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 16920)	Forwards notification messages to all owned components.
	NotifyAdapterChange (see page 25130)	Responds when other components notify this one of changes to the adapter.
	OnDisplayComponentChange (see page 25131)	Responds when the associated display component changes.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
V	SetWebDisplayComponent (see page 16921)	This is SetWebDisplayComponent, a member of class TWebDisplayReferenceGroup.
V	SetWebDisplayComponent (see page 25131)	Sets the value of the DisplayComponent (see page 25132) property.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	Adapter (see page 25228)	Provides read-only access to the Adapter that this command group uses.
R	AdapterActionsListHelper (see page 25229)	Provides read-only access to a helper object that implements design-time editing functions on behalf of this component.
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebDisplay (see page 16921)	This is WebDisplay, a member of class TWebDisplayReferenceGroup.
	WebDisplayComponent (see page 16921)	
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 25225)	Creator for TCustomAdapterCommandGroup (see page 25220).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25225)	Frees an instance of TAdapterDisplayReferenceGroup.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindAction (see page 19744)	This is FindAction, a member of class IWebActionsList.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetActionsList (see page 19744)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterDisplayCharacteristics (see page 19687)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.
	GetAddAllFieldsItem (see page 24770)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 24770)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.

	GetAddFieldsItem (see page 24770)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetDesigntimeWarnings (see page 19697)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNewFieldPrefix (see page 24771)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	GetVariablesContainer (see page 25311)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	GetVisibleActions (see page 19745)	This is GetVisibleActions, a member of class IWebActionsList.
	GetWebDisplayComponent (see page 17081)	This is GetWebDisplayComponent, a member of class IWebDisplayReference.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsActionInUse (see page 19745)	This is IsActionInUse, a member of class IWebActionsList.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NotifyAdapterChange (see page 19725)	Responds when the adapter changes.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	AddDefaultCommands (see page 25229)	Provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.

	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 25229)	Provides read and write access to the custom HTML element attributes, such as Valign="top."
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayComponent (see page 25132)	Associates the group with a component that generates the display controls.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 25229)	Provides read and write access to the internal string that stores the CSS style, such as making the background color red.
	StyleRule (see page 25230)	Provides read and write access to the internal string that stores the CSS style rule for the control group.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleActions (see page 19745)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleCommands (see page 25230)	Provides read-only access to the TWebComponentList that stores the list of visible components.
R	WebDisplayComponent (see page 17081)	
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

181.1.21.2 TCustomAdapterCommandGroup.Create Constructor

Creator for TCustomAdapterCommandGroup (see page 25220).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterCommandGroup(TComponent * AOwner);
```

Description

This constructor calls the base class version, and then creates a TAdapterActionsListHelper (see page 25046) component and sets the AddDefaultActions property to true by default.

See Also

Create_d

AddDefaultActions (see page 25052)

181.1.21.3 TCustomAdapterCommandGroup.Destroy Destructor

Frees an instance of TAdapterDisplayReferenceGroup.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterCommandGroup();
```

Description

Do not call Destroy directly. Instead, call the Free method, which checks whether the reference to the producer is not nil before calling Destroy.

See Also

Free (see page 22300)

181.1.21.4 TCustomAdapterCommandGroup Methods

The methods of the TCustomAdapterCommandGroup class are listed here.

Protected Methods

	Name	Description
	GetAdapter (see page 25226)	This is GetAdapter, a member of class TCustomAdapterCommandGroup.
	GetAdapterFormLayout (see page 25226)	This is GetAdapterFormLayout, a member of class TCustomAdapterCommandGroup.
	GetAdapterModeOfAdapter (see page 25227)	Returns the adapter mode defined by the adapter component.
	GetAdapterModeOfDisplay (see page 25227)	Returns the adapter mode defined by the TAdapterPageProducer components.
V	ImplCanAddClass (see page 25227)	Checks the inheritance of the given component.
V	ImplContent (see page 25228)	Lays out the buttons for the command group using the TAdapterFormLayout (see page 25166) component.

181.1.21.4.1 TCustomAdapterCommandGroup.GetAdapter Method

Pascal

```
function GetAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetAdapter();
```

Description

This is GetAdapter, a member of class TCustomAdapterCommandGroup.

181.1.21.4.2 TCustomAdapterCommandGroup.GetAdapterFormLayout Method

Pascal

```
function GetAdapterFormLayout: TAdapterFormLayout;
```

C++

```
__fastcall TAdapterFormLayout GetAdapterFormLayout();
```

Description

This is GetAdapterFormLayout, a member of class TCustomAdapterCommandGroup.

181.1.21.4.3 TCustomAdapterCommandGroup.GetAdapterModeOfAdapter Method

Returns the adapter mode defined by the adapter component.

Pascal

```
function GetAdapterModeOfAdapter: string;
```

C++

```
__fastcall AnsiString GetAdapterModeOfAdapter();
```

Description

The GetAdapterModeOfAdapter function returns the adapter mode defined by the adapter component. Some adapters support a mode. For example, the TDataSetAdapter supports Edit, Insert, and Delete modes. The mode can be set by TAdapterPageProducer components or during adapter action execution. The mode defined by adapter action takes precedence over the mode defined by TAdapterPageProducer components.

181.1.21.4.4 TCustomAdapterCommandGroup.GetAdapterModeOfDisplay Method

Returns the adapter mode defined by the TAdapterPageProducer components.

Pascal

```
function GetAdapterModeOfDisplay: string;
```

C++

```
__fastcall AnsiString GetAdapterModeOfDisplay();
```

Description

This function returns the adapter mode defined by the TAdapterPageProducer components. Some adapters support a mode. For example, the TDataSetAdapter supports Edit, Insert, and Delete modes. The mode can be set by TAdapterPageProducer components or during adapter action execution. The mode defined by adapter action takes precedence over the mode defined by TAdapterPageProducer components.

181.1.21.4.5 TCustomAdapterCommandGroup.ImplCanAddClass Method

Checks the inheritance of the given component.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The ImplCanAddClass checks the inheritance of the given component. If it is derived from either TCustomAdapterActionButton (see page 25189) or TCustomLayoutGroup, it will return True to indicate that the class can be added to this TCustomAdapterCommandGroup (see page 25220).

181.1.21.4.6 TCustomAdapterCommandGroup.ImplContent Method

Lays out the buttons for the command group using the TAdapterFormLayout (see page 25166) component.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The ImplContent function lays out the buttons for the command group using the TAdapterFormLayout (see page 25166) component. It loops through all of the visible child components, calling them to render their HTML content.

181.1.21.5 TCustomAdapterCommandGroup Properties

The properties of the TCustomAdapterCommandGroup class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 25228)	Provides read-only access to the Adapter that this command group uses.
R	AdapterActionsListHelper (see page 25229)	Provides read-only access to a helper object that implements design-time editing functions on behalf of this component.

Public Properties

	Name	Description
	AddDefaultCommands (see page 25229)	Provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.
	Custom (see page 25229)	Provides read and write access to the custom HTML element attributes, such as Valign="top."
	Style (see page 25229)	Provides read and write access to the internal string that stores the CSS style, such as making the background color red.
	StyleRule (see page 25230)	Provides read and write access to the internal string that stores the CSS style rule for the control group.
R	VisibleCommands (see page 25230)	Provides read-only access to the TWebComponentList that stores the list of visible components.

181.1.21.5.1 TCustomAdapterCommandGroup.Adapter Property

Provides read-only access to the Adapter that this command group uses.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

The Adapter property provides read-only access to the Adapter that this command group uses.

181.1.21.5.2 TCustomAdapterCommandGroup.AdapterActionsListHelper Property

Provides read-only access to a helper object that implements design-time editing functions on behalf of this component.

Pascal

```
property AdapterActionsListHelper: TAdapterActionsListHelper;
```

C++

```
__property TAdapterActionsListHelper AdapterActionsListHelper;
```

Description

The AdapterActionsListHelper provides read-only access to a helper object that implements design-time editing functions on behalf of this component.

181.1.21.5.3 TCustomAdapterCommandGroup.AddDefaultCommands Property

Provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.

Pascal

```
property AddDefaultCommands: Boolean;
```

C++

```
__property Boolean AddDefaultCommands;
```

Description

The AddDefaultCommands property provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.

181.1.21.5.4 TCustomAdapterCommandGroup.Custom Property

Provides read and write access to the custom HTML element attributes, such as Valign="top."

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

The Custom property provides read and write access to the custom HTML element attributes, such as Valign="top." The content of this string will be written as it is when generating HTML.

181.1.21.5.5 TCustomAdapterCommandGroup.Style Property

Provides read and write access to the internal string that stores the CSS style, such as making the background color red.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

The Style property provides read and write access to the internal string that stores the CSS style, such as making the background color red.

181.1.21.5.6 TCustomAdapterCommandGroup.StyleRule Property

Provides read and write access to the internal string that stores the CSS style rule for the control group.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

The StyleRule property provides read and write access to the internal string that stores the CSS style rule for the control group. This property is used to reference a CSS style defined in TAdapterPageProducer. ::Styles.

181.1.21.5.7 TCustomAdapterCommandGroup.VisibleCommands Property

Provides read-only access to the TWebComponentList that stores the list of visible components.

Pascal

```
property VisibleCommands: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleCommands;
```

Description

The VisibleCommands property provides read-only access to the TWebComponentList that stores the list of visible components.

181.1.22 TCustomAdapterDisplayColumn Class

TCustomAdapterDisplayColumn is the base class for the grid display of adapter field data.

Class Hierarchy

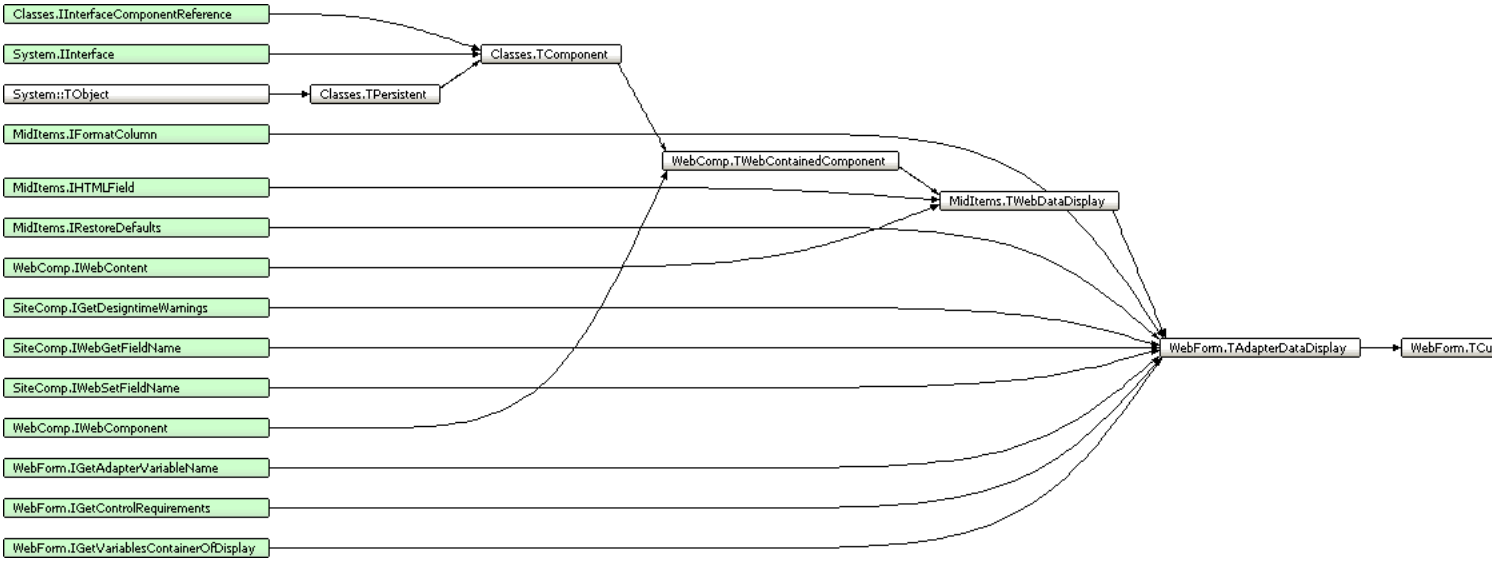

Pascal

```
TCustomAdapterDisplayColumn = class(TAdapterDataDisplay);
```

C++

```
class TCustomAdapterDisplayColumn : public TAdapterDataDisplay;
```

File

WebForm

Description

TCustomAdapterDisplayColumn is the base class for grid components that display adapter field data. Do not directly instantiate TCustomAdapterDisplayColumn. Use TAdapterDisplayColumn (see page 25099) or create your own concrete descendants of TCustomAdapterDisplayColumn.

See Also

IWebContent (see page 24792)

TAdapterGrid (see page 25175)

TAdapterFieldGroup (see page 25150)

TAdapterCommandColumn (see page 25052)

TAdapterCommandGroup (see page 25062)

TAdapterDisplayColumn (see page 25099)

181.1.22.1 TCustomAdapterDisplayColumn Members

The following tables list the members exposed by TCustomAdapterDisplayColumn.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 25237)	This is CanAddClass, a member of class TCustomAdapterDisplayColumn.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
V	ControlContent (see page 25237)	This is ControlContent, a member of class TCustomAdapterDisplayColumn.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	FieldConditions (see page 25092)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 25093)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	FormatCaption (see page 25093)	This is FormatCaption, a member of class TAdapterDataDisplay.
	FormatColumnData (see page 25093)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 25093)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	FormatImage (see page 25237)	This is FormatImage, a member of class TCustomAdapterDisplayColumn.
	FormatListDisplay (see page 25237)	This is FormatListDisplay, a member of class TCustomAdapterDisplayColumn.
	FormatTextDisplay (see page 25238)	This is FormatTextDisplay, a member of class TCustomAdapterDisplayColumn.
	GetAdapterModeOfAdapter (see page 25093)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 25094)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 25094)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetControlRequirements (see page 25094)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 25094)	This is GetDesigntimeWarnings, a member of class TAdapterDataDisplay.
	GetFieldName (see page 25095)	This is GetFieldName, a member of class TAdapterDataDisplay.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariablesContainer (see page 25095)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.

	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplCanAddClass (see page 25238)	This is ImplCanAddClass, a member of class TCustomAdapterDisplayColumn.
	ImplContent (see page 16894)	Provides the underlying implementation for the Content (see page 16890) method.
	ImplContent (see page 25095)	Provides the underlying implementation for the Content method.
	ImplFormatColumnData (see page 25096)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
	ImplFormatColumnHeading (see page 25238)	This is ImplFormatColumnHeading, a member of class TCustomAdapterDisplayColumn.
	ImplGetControlRequirements (see page 25238)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayColumn.
	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
	ImplRestoreDefaults (see page 25239)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayColumn.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 25096)	This is RestoreDefaults, a member of class TAdapterDataDisplay.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
	SetCaption (see page 25097)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 25097)	This is SetFieldName, a member of class TAdapterDataDisplay.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	CaptionPosition (see page 16897)	Specifies where the label appears relative to the generated display control.

R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 25236)	Creates and initializes an instance of TCustomAdapterDisplayColumn (see page 25230).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FormatColumnData (see page 17062)	This is FormatColumnData, a member of class IFormatColumn.
	FormatColumnHeading (see page 17062)	This is FormatColumnHeading, a member of class IFormatColumn.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterVariableName (see page 25307)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetControlRequirements (see page 25309)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 19697)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetVariablesContainer (see page 25311)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsColumn (see page 25239)	Indicates whether the component generates the controls that appear in the column of a data grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AdapterVariableName (see page 25308)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	Align (see page 25097)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 16897)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16897)	Customizes the appearance of the generated control's label.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.

R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16898)	Represents a tag option and its value in the tag for the display control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayType (see page 25239)	Sets the display type for the column.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 25098)	Specifies the name of the associated dataset field.
	HideOptions (see page 25098)	When field should be hidden.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	ImageHeight (see page 25240)	Specifies the height of the image in the column.
	ImageWidth (see page 25240)	Specifies the width of the image in the column.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 16899)	Adds style attributes to the generated display control.
	StyleRule (see page 16899)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 16899)	Determines the position of the generated control in the HTML form's tab order.
	VAlign (see page 25098)	Specifies the vertical alignment of the generated display control.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

181.1.22.2 TCustomAdapterDisplayColumn.Create Constructor

Creates and initializes an instance of TCustomAdapterDisplayColumn (see page 25230).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterDisplayColumn(TComponent * AOwner);
```

Description

Call Create to instantiate TCustomAdapterDisplayColumn (see page 25230) at runtime. TCustomAdapterDisplayColumn (see page 25230) objects added at design time using the Web Page editor are instantiated automatically.

181.1.22.3 TCustomAdapterDisplayColumn Methods

The methods of the TCustomAdapterDisplayColumn class are listed here.

Protected Methods

	Name	Description
	CanAddClass (see page 25237)	This is CanAddClass, a member of class TCustomAdapterDisplayColumn.
V	ControlContent (see page 25237)	This is ControlContent, a member of class TCustomAdapterDisplayColumn.
	FormatImage (see page 25237)	This is FormatImage, a member of class TCustomAdapterDisplayColumn.
	FormatListDisplay (see page 25237)	This is FormatListDisplay, a member of class TCustomAdapterDisplayColumn.
	FormatTextDisplay (see page 25238)	This is FormatTextDisplay, a member of class TCustomAdapterDisplayColumn.
V	ImplCanAddClass (see page 25238)	This is ImplCanAddClass, a member of class TCustomAdapterDisplayColumn.

	ImplFormatColumnHeading (see page 25238)	This is ImplFormatColumnHeading, a member of class TCustomAdapterDisplayColumn.
	ImplGetControlRequirements (see page 25238)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayColumn.
	ImplRestoreDefaults (see page 25239)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayColumn.

Public Methods

	Name	Description
	IsColumn (see page 25239)	Indicates whether the component generates the controls that appear in the column of a data grid.

181.1.22.3.1 TCustomAdapterDisplayColumn.CanAddClass Method

Pascal

```
function CanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClass(TComponent * AParent, TClass AClass);
```

Description

This is CanAddClass, a member of class TCustomAdapterDisplayColumn.

181.1.22.3.2 TCustomAdapterDisplayColumn.ControlContent Method

Pascal

```
function ControlContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall AnsiString ControlContent(TWebContentOptions Options);
```

Description

This is ControlContent, a member of class TCustomAdapterDisplayColumn.

181.1.22.3.3 TCustomAdapterDisplayColumn.FormatImage Method

Pascal

```
function FormatImage(const VarName: string; const HTMLName: string; Options:
TWebContentOptions): string;
```

C++

```
__fastcall AnsiString FormatImage(const AnsiString VarName, const AnsiString HTMLName,
TWebContentOptions Options);
```

Description

This is FormatImage, a member of class TCustomAdapterDisplayColumn.

181.1.22.3.4 TCustomAdapterDisplayColumn.FormatListDisplay Method

Pascal

```
function FormatListDisplay(const VarName: string; const HTMLName: string): string;
```

C++

```
__fastcall AnsiString FormatListDisplay(const AnsiString VarName, const AnsiString HTMLName);
```

Description

This is FormatListDisplay, a member of class TCustomAdapterDisplayColumn.

181.1.22.3.5 TCustomAdapterDisplayColumn.FormatTextDisplay Method

Pascal

```
function FormatTextDisplay(const VarName: string; const HTMLName: string): string;
```

C++

```
__fastcall AnsiString FormatTextDisplay(const AnsiString VarName, const AnsiString HTMLName);
```

Description

This is FormatTextDisplay, a member of class TCustomAdapterDisplayColumn.

181.1.22.3.6 TCustomAdapterDisplayColumn.ImplCanAddClass Method

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

This is ImplCanAddClass, a member of class TCustomAdapterDisplayColumn.

181.1.22.3.7 TCustomAdapterDisplayColumn.ImplFormatColumnHeading Method

Pascal

```
function ImplFormatColumnHeading(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall AnsiString ImplFormatColumnHeading(TWebContentOptions Options);
```

Description

This is ImplFormatColumnHeading, a member of class TCustomAdapterDisplayColumn.

181.1.22.3.8 TCustomAdapterDisplayColumn.ImplGetControlRequirements Method

Pascal

```
procedure ImplGetControlRequirements(ARequirements: TAbstractControlRequirements); override;
```

C++

```
virtual __fastcall ImplGetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayColumn.

181.1.22.3.9 TCustomAdapterDisplayColumn.ImplRestoreDefaults Method

Pascal

```
procedure ImplRestoreDefaults; override;
```

C++

```
virtual __fastcall ImplRestoreDefaults();
```

Description

This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayColumn.

181.1.22.3.10 TCustomAdapterDisplayColumn.IsColumn Method

Indicates whether the component generates the controls that appear in the column of a data grid.

Pascal

```
class function IsColumn: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsColumn();
```

Description

Call IsColumn to determine whether a TCustomAdapterDisplayColumn (see page 25230) object generates the controls that appear in a single column of a data grid. As implemented in TCustomAdapterDisplayColumn (see page 25230), IsColumn always returns true.

See Also

IsColumn (see page 16895)

181.1.22.4 TCustomAdapterDisplayColumn Properties

The properties of the TCustomAdapterDisplayColumn class are listed here.

Public Properties

	Name	Description
	DisplayType (see page 25239)	Sets the display type for the column.
	ImageHeight (see page 25240)	Specifies the height of the image in the column.
	ImageWidth (see page 25240)	Specifies the width of the image in the column.

181.1.22.4.1 TCustomAdapterDisplayColumn.DisplayType Property

Sets the display type for the column.

Pascal

```
property DisplayType: TColumnHTMLElementType;
```

C++

```
__property TColumnHTMLElementType DisplayType;
```

Description

Set DisplayType to set the display type for the column to text or image.

See Also

ImageHeight (see page 25240)

ImageWidth (see page 25240)

181.1.22.4.2 TCustomAdapterDisplayColumn.ImageHeight Property

Specifies the height of the image in the column.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

Set ImageHeight to indicate the height of the image in pixels.

See Also

ImageWidth (see page 25240)

181.1.22.4.3 TCustomAdapterDisplayColumn.ImageWidth Property

Specifies the width of the image in the column.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

Set ImageHeight (see page 25240) to indicate the width of the image in pixels.

See Also

ImageHeight (see page 25240)

181.1.23 TCustomAdapterDisplayField Class

TCustomAdapterDisplayField is the base class for the free-form display and editing of adapter field data.

Class Hierarchy

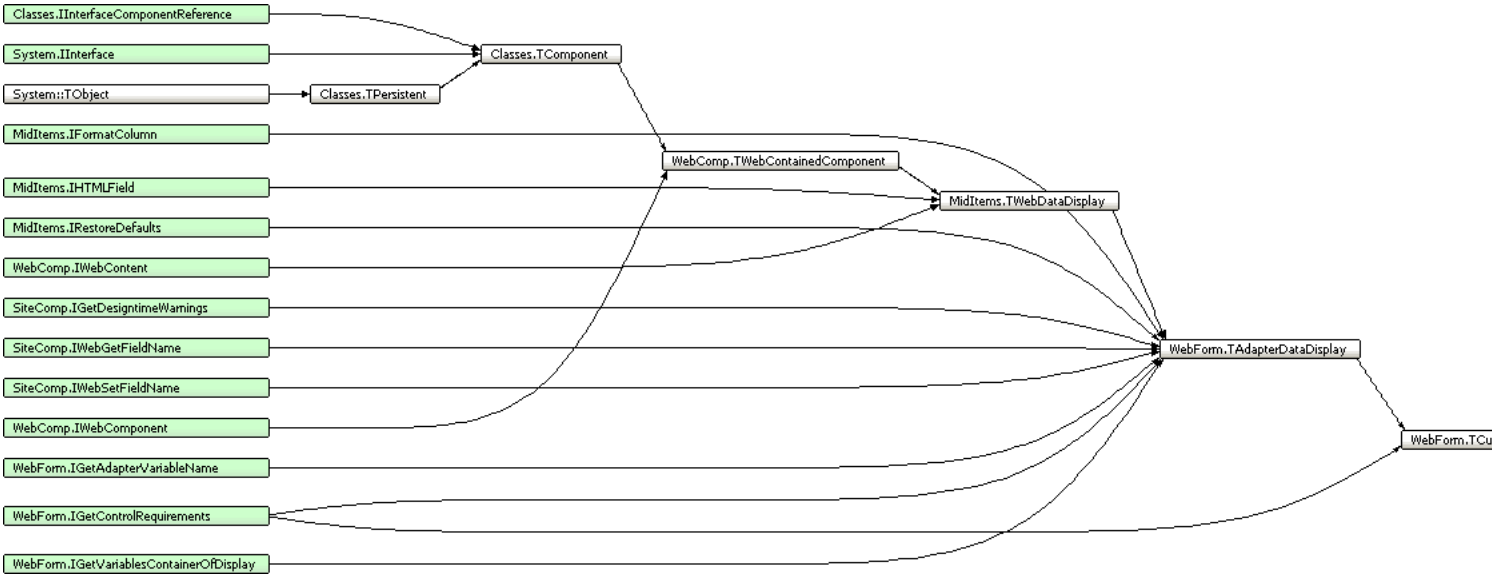

Pascal

```
TCustomAdapterDisplayField = class(TAdapterDataDisplay, IGetControlRequirements);
```

C++

```
class TCustomAdapterDisplayField : public TAdapterDataDisplay, public IGetControlRequirements;
```

File

WebForm

Description

TCustomAdapterDisplayField is the base class for free-form components that display and edit adapter field data. Do not directly instantiate TCustomAdapterDisplayField. Use TAdapterDisplayField (see page 25109) or create your own concrete descendants of TCustomAdapterDisplayField.

See Also

TCustomAdapterEditColumn (see page 25252)

TAdapterDisplayField (see page 25109)

IWebContent (see page 24792)

TAdapterGrid (see page 25175)

TAdapterFieldGroup (see page 25150)

181.1.23.1 TCustomAdapterDisplayField Members

The following tables list the members exposed by TCustomAdapterDisplayField.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAttributes (see page 25247)	This is AddAttributes, a member of class TCustomAdapterDisplayField.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
V	ControlContent (see page 25247)	This is ControlContent, a member of class TCustomAdapterDisplayField.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
V	FieldConditions (see page 25092)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 25093)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
V	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
V	FormatCaption (see page 25093)	This is FormatCaption, a member of class TAdapterDataDisplay.
	FormatColumnData (see page 25093)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 25093)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	GetAdapterModeOfAdapter (see page 25093)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 25094)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
V	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
V	GetCaption (see page 25094)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetControlRequirements (see page 25094)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 25094)	This is GetDesigntimeWarnings, a member of class TAdapterDataDisplay.
	GetFieldName (see page 25095)	This is GetFieldName, a member of class TAdapterDataDisplay.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 25247)	Returns the value of the LayoutAttributes property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariablesContainer (see page 25095)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
V	ImplContent (see page 25247)	Provides the underlying implementation for the Content method.
V	ImplFormatColumnData (see page 25096)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
V	ImplFormatColumnHeading (see page 25096)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
V	ImplGetControlRequirements (see page 25248)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayField.
V	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.

V	ImplRestoreDefaults (see page 25248)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 25096)	This is RestoreDefaults, a member of class TAdapterDataDisplay.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
V	SetCaption (see page 25097)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 25097)	This is SetFieldName, a member of class TAdapterDataDisplay.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	CaptionPosition (see page 16897)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.

	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
	Create (see page 25246)	Creates and initializes an instance of TCustomAdapterDisplayField (see page 25240).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FormatColumnData (see page 17062)	This is FormatColumnData, a member of class IFormatColumn.
	FormatColumnHeading (see page 17062)	This is FormatColumnHeading, a member of class IFormatColumn.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterVariableName (see page 25307)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetControlRequirements (see page 25309)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 19697)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetVariablesContainer (see page 25311)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AdapterVariableName (see page 25308)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	Align (see page 25097)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 16897)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16897)	Customizes the appearance of the generated control's label.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16898)	Represents a tag option and its value in the tag for the display control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayColumns (see page 25249)	Specifies the number of columns in a display field.
	DisplayRows (see page 25249)	Specifies the number of rows in a display field.
	DisplayType (see page 25249)	Sets the display type for the column.
	DisplayWidth (see page 25250)	Specifies the width, in characters, of the generated HTML field.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 25098)	Specifies the name of the associated dataset field.
	HideOptions (see page 25098)	When field should be hidden.

R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	ImageHeight (see page 25250)	Specifies the height of the image in the column.
	ImageWidth (see page 25250)	Specifies the width of the image in the column.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
	InputType (see page 25251)	Specifies the input type for the display field.
	MaxLength (see page 25251)	Specifies the maximum number of characters the user can enter into the display field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SelectRows (see page 25251)	Indicates the number of rows that can be selected.
	Style (see page 16899)	Adds style attributes to the generated display control.
	StyleRule (see page 16899)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 16899)	Determines the position of the generated control in the HTML form's tab order.
	TextAreaWrap (see page 25251)	Specifies how the display field treats lines that are longer than the display width.
	VAlign (see page 25098)	Specifies the vertical alignment of the generated display control.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewMode (see page 25252)	Indicates the viewing state of the adapter field.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

181.1.23.2 TCustomAdapterDisplayField.Create Constructor

Creates and initializes an instance of TCustomAdapterDisplayField (see page 25240).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterDisplayField(TComponent * AOwner);
```

Description

Call Create to instantiate TCustomAdapterDisplayField (see page 25240) at runtime. TCustomAdapterDisplayField (see page 25240) descendants added at design time using the Web Page editor are instantiated automatically.

181.1.23.3 TCustomAdapterDisplayField Methods

The methods of the TCustomAdapterDisplayField class are listed here.

Protected Methods

	Name	Description
	AddAttributes (see page 25247)	This is AddAttributes, a member of class TCustomAdapterDisplayField.
V	ControlContent (see page 25247)	This is ControlContent, a member of class TCustomAdapterDisplayField.
V	GetLayoutAttributes (see page 25247)	Returns the value of the LayoutAttributes property.
V	ImplContent (see page 25247)	Provides the underlying implementation for the Content method.
V	ImplGetControlRequirements (see page 25248)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayField.
V	ImplRestoreDefaults (see page 25248)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayField.

181.1.23.3.1 TCustomAdapterDisplayField.AddAttributes Method

Pascal

```
procedure AddAttributes(var Attrs: string);
```

C++

```
__fastcall AddAttributes(AnsiString Attrs);
```

Description

This is AddAttributes, a member of class TCustomAdapterDisplayField.

181.1.23.3.2 TCustomAdapterDisplayField.ControlContent Method

Pascal

```
function ControlContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall AnsiString ControlContent(TWebContentOptions Options);
```

Description

This is ControlContent, a member of class TCustomAdapterDisplayField.

181.1.23.3.3 TCustomAdapterDisplayField.GetLayoutAttributes Method

Returns the value of the LayoutAttributes property.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content method) to take into account layout properties of the child component.

GetLayoutAttributes returns the value of the LayoutAttributes property.

See Also

Content (see page 16890)

LayoutAttributes (see page 16898)

181.1.23.3.4 TCustomAdapterDisplayField.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML that defines the display control and its label. ImplContent formats the caption and combines this HTML with the results of the ControlContent method. ControlContent is an abstract (pure virtual) method in TWebDataDisplay. TWebDataDisplay descendants override it to generate the definition of their particular display control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebDataDisplay should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the display control.

See Also

Content (see page 16890)

Caption (see page 16897)

ILayoutWebContent (see page 24774)

181.1.23.3.5 TCustomAdapterDisplayField.ImplGetControlRequirements Method

Pascal

```
procedure ImplGetControlRequirements(ARequirements: TAbstractControlRequirements); override;
```

C++

```
virtual __fastcall ImplGetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayField.

181.1.23.3.6 TCustomAdapterDisplayField.ImplRestoreDefaults Method

Pascal

```
procedure ImplRestoreDefaults; override;
```

C++

```
virtual __fastcall ImplRestoreDefaults();
```

Description

This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayField.

181.1.23.4 TCustomAdapterDisplayField Properties

The properties of the TCustomAdapterDisplayField class are listed here.

Public Properties

	Name	Description
	DisplayColumns (see page 25249)	Specifies the number of columns in a display field.
	DisplayRows (see page 25249)	Specifies the number of rows in a display field.
	DisplayType (see page 25249)	Sets the display type for the column.

	DisplayWidth (see page 25250)	Specifies the width, in characters, of the generated HTML field.
	ImageHeight (see page 25250)	Specifies the height of the image in the column.
	ImageWidth (see page 25250)	Specifies the width of the image in the column.
	InputType (see page 25251)	Specifies the input type for the display field.
	MaxLength (see page 25251)	Specifies the maximum number of characters the user can enter into the display field.
	SelectRows (see page 25251)	Indicates the number of rows that can be selected.
	TextAreaWrap (see page 25251)	Specifies how the display field treats lines that are longer than the display width.
	ViewMode (see page 25252)	Indicates the viewing state of the adapter field.

181.1.23.4.1 TCustomAdapterDisplayField.DisplayColumns Property

Specifies the number of columns in a display field.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Use DisplayColumns to control the number of columns being displayed.

See Also

DisplayRows (see page 25249)

181.1.23.4.2 TCustomAdapterDisplayField.DisplayRows Property

Specifies the number of rows in a display field.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayColumns (see page 25249) to control the number of rows being displayed.

See Also

DisplayColumns (see page 25249)

181.1.23.4.3 TCustomAdapterDisplayField.DisplayType Property

Sets the display type for the column.

Pascal

```
property DisplayType: TDisplayFieldHTMLElementType;
```

C++

```
__property TDisplayFieldHTMLElementType DisplayType;
```

Description

Set DisplayType to indicate the type of HTML control that should be used to display this field's value when it is read-only.

See Also

ImageHeight (see page 25250)

ImageWidth (see page 25250)

InputType (see page 25251)

181.1.23.4.4 TCustomAdapterDisplayField.DisplayWidth Property

Specifies the width, in characters, of the generated HTML field.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to control the width of the HTML field that displays record status information. By default, DisplayWidth is 1 because the TCustomAdapterDisplayField (see page 25240) control values are one character (M for modified or I for inserted).

181.1.23.4.5 TCustomAdapterDisplayField.ImageHeight Property

Specifies the height of the image in the column.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

Set ImageHeight to indicate the height of the image in the column.

See Also

DisplayWidth (see page 25250)

ImageWidth (see page 25250)

181.1.23.4.6 TCustomAdapterDisplayField.ImageWidth Property

Specifies the width of the image in the column.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

Set ImageHeight (see page 25250) to indicate the width of the image in the column.

See Also

ImageHeight (see page 25250)

181.1.23.4.7 TCustomAdapterDisplayField.InputType Property

Specifies the input type for the display field.

Pascal

```
property InputType: TInputFieldHTMLElementType;
```

C++

```
__property TInputFieldHTMLElementType InputType;
```

Description

Set InputType to indicate the type of control that should be generated to let users assign values to the adapter field.

See Also

ImageHeight (see page 25250)

ImageWidth (see page 25250)

DisplayType (see page 25249)

181.1.23.4.8 TCustomAdapterDisplayField.MaxLength Property

Specifies the maximum number of characters the user can enter into the display field.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the display field. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

181.1.23.4.9 TCustomAdapterDisplayField.SelectRows Property

Indicates the number of rows that can be selected.

Pascal

```
property SelectRows: Integer;
```

C++

```
__property int SelectRows;
```

Description

Use SelectRows to indicate the number of rows that can be selected.

181.1.23.4.10 TCustomAdapterDisplayField.TextAreaWrap Property

Specifies how the display field treats lines that are longer than the display width.

Pascal

```
property TextAreaWrap: TTextAreaWrap;
```

C++

```
__property TTextAreaWrap TextAreaWrap;
```

Description

Use TextAreaWrap to specify how the generated display field formats lines that are longer than the number of characters specified by DisplayWidth (see page 25250).

See Also

DisplayWidth (see page 25250)

181.1.23.4.11 TCustomAdapterDisplayField.ViewMode Property

Indicates the viewing state of the adapter field.

Pascal

```
property ViewMode: TDisplayFieldViewMode;
```

C++

```
__property TDisplayFieldViewMode ViewMode;
```

Description

Use ViewMode to specify the viewing state of the field.

181.1.24 TCustomAdapterEditColumn Class

TCustomAdapterEditColumn is the base class for components that display adapter fields as form fields in grids.

Class Hierarchy

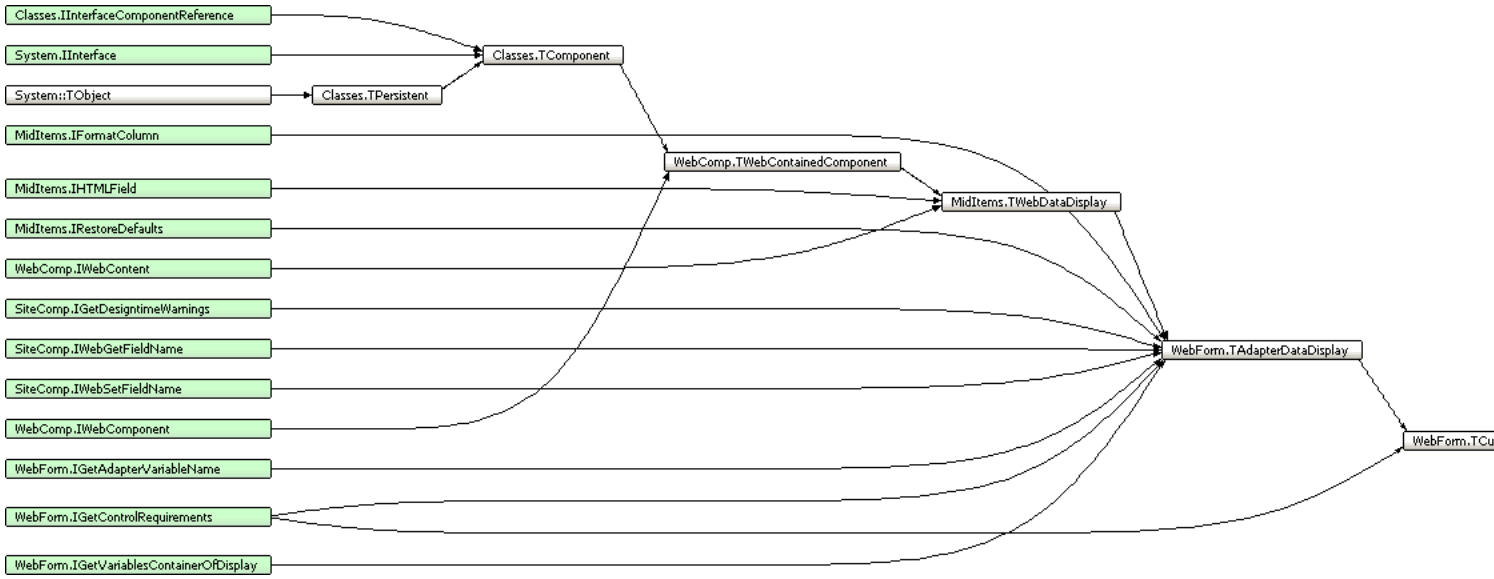

Pascal

```
TCustomAdapterEditColumn = class(TCustomAdapterDisplayField);
```

C++

```
class TCustomAdapterEditColumn : public TCustomAdapterDisplayField;
```

File

WebForm

Description

TCustomAdapterEditColumn has the same functionality as TCustomAdapterDisplayField (see page 25240), but is designed to work with grids. Do not directly instantiate TCustomAdapterEditColumn. Use TAdapterEditColumn (see page 25132) or create your own concrete descendants of TCustomAdapterEditColumn.

See Also

TAdapterEditColumn (see page 25132)

181.1.24.1 TCustomAdapterEditColumn Members

The following tables list the members exposed by TCustomAdapterEditColumn.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddAttributes (see page 25247)	This is AddAttributes, a member of class TCustomAdapterDisplayField.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.

	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16890)	Returns the HTML that defines the display control.
	ControlContent (see page 16891)	This is ControlContent, a member of class TWebDataDisplay.
	ControlContent (see page 25247)	This is ControlContent, a member of class TCustomAdapterDisplayField.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	EventContent (see page 16891)	This is EventContent, a member of class TWebDataDisplay.
	FieldConditions (see page 25092)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 25093)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
	FormatCaption (see page 16891)	This is FormatCaption, a member of class TWebDataDisplay.
	FormatCaption (see page 25093)	This is FormatCaption, a member of class TAdapterDataDisplay.
	FormatColumnData (see page 25093)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 25093)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	GetAdapterModeOfAdapter (see page 25093)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 25094)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
	GetCaption (see page 16891)	Returns the value of the Caption (see page 16897) property.
	GetCaption (see page 25094)	Returns the value of the Caption property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetControlRequirements (see page 25094)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 25094)	This is GetDesigntimeWarnings, a member of class TAdapterDataDisplay.
	GetFieldName (see page 25095)	This is GetFieldName, a member of class TAdapterDataDisplay.
	GetHTMLControlName (see page 16892)	Returns the name that other components can use to refer to the generated display control.
	GetHTMLForm (see page 16892)	Used internally to retrieve information about the form in which the display control is defined.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetLayoutAttributes (see page 16892)	Returns the value of the LayoutAttributes (see page 16898) property.
	GetLayoutAttributes (see page 25247)	Returns the value of the LayoutAttributes property.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVariablesContainer (see page 25095)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
	GetXmlDisplayName (see page 16893)	Returns the name of the javascript object that uses TWebDataDisplay (see page 16884) to generate one of its display controls. Not used on Linux.
	GetXmlRowSetName (see page 16893)	Returns the name of the rowset in the XML data packet whose data is represented by the HTML this component generates. Not used on Linux.
	ImplContent (see page 25258)	Provides the underlying implementation for the Content method.
	ImplFormatColumnData (see page 25096)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
	ImplFormatColumnHeading (see page 25096)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
	ImplGetControlRequirements (see page 25096)	This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.
	ImplGetControlRequirements (see page 25248)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayField.
	ImplGetHTMLControlName (see page 16894)	Returns the name that other components can use to refer to the generated display control.
	ImplRestoreDefaults (see page 25096)	This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.
	ImplRestoreDefaults (see page 25248)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayField.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.

V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	LabelContent (see page 16895)	This is LabelContent, a member of class TWebDataDisplay.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	RestoreDefaults (see page 25096)	This is RestoreDefaults, a member of class TAdapterDataDisplay.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
V	SetCaption (see page 16896)	Sets the value of the Caption (see page 16897) property.
V	SetCaption (see page 25097)	Sets the value of the Caption property.
	SetCaptionAttributes (see page 16896)	Sets the value of the CaptionAttributes (see page 16897) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetFieldName (see page 25097)	This is SetFieldName, a member of class TAdapterDataDisplay.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	CaptionPosition (see page 16897)	Specifies where the label appears relative to the generated display control.
R	LayoutAttributes (see page 16898)	Encodes attributes that describe how the generated control should look.
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 16889)	Creates and initializes an instance of TWebDataDisplay (see page 16884).
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 24726)	Creates and initializes an instance of TWebContainedComponent (see page 24722).
V	Create (see page 25092)	Creates and initializes an instance of TWebDataDisplay.
V	Create (see page 25246)	Creates and initializes an instance of TCustomAdapterDisplayField (see page 25240).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 16889)	Frees an instance of TWebDataDisplay (see page 16884).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 24726)	Frees an instance of TWebContainedComponent (see page 24722).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FormatColumnData (see page 17062)	This is FormatColumnData, a member of class IFormatColumn.
	FormatColumnHeading (see page 17062)	This is FormatColumnHeading, a member of class IFormatColumn.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapterVariableName (see page 25307)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetControlRequirements (see page 25309)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 19697)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldName (see page 19758)	This is GetFieldName, a member of class IWebGetFieldName.
	GetHTMLControlName (see page 17063)	Returns the name of the generated HTML <INPUT> control.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	GetVariablesContainer (see page 25311)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.

	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
V	IsColumn (see page 16895)	Indicates whether the component generates the controls that appear in the column of a grid.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
V	IsQueryField (see page 16895)	Indicates whether the component generates the controls that appear in a query form. Not used on Linux.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RestoreDefaults (see page 17075)	This is RestoreDefaults, a member of class IRestoreDefaults.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetFieldName (see page 19784)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AdapterVariableName (see page 25308)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	Align (see page 25097)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 16897)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 16897)	Customizes the appearance of the generated control's label.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 16898)	Represents a tag option and its value in the tag for the display control.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DisplayColumns (see page 25249)	Specifies the number of columns in a display field.
	DisplayRows (see page 25249)	Specifies the number of rows in a display field.
	DisplayType (see page 25249)	Sets the display type for the column.
	DisplayWidth (see page 25250)	Specifies the width, in characters, of the generated HTML field.
R	FieldName (see page 19758)	This is FieldName, a member of class IWebGetFieldName.

	FieldName (see page 25098)	Specifies the name of the associated dataset field.
	HideOptions (see page 25098)	When field should be hidden.
R	HTMLControlName (see page 17064)	Indicates the name of the generated HTML control.
	ImageHeight (see page 25250)	Specifies the height of the image in the column.
	ImageWidth (see page 25250)	Specifies the width of the image in the column.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
	InputType (see page 25251)	Specifies the input type for the display field.
	MaxLength (see page 25251)	Specifies the maximum number of characters the user can enter into the display field.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	SelectRows (see page 25251)	Indicates the number of rows that can be selected.
	Style (see page 16899)	Adds style attributes to the generated display control.
	StyleRule (see page 16899)	This is StyleRule, a member of class TWebDataDisplay.
	TabIndex (see page 16899)	Determines the position of the generated control in the HTML form's tab order.
	TextAreaWrap (see page 25251)	Specifies how the display field treats lines that are longer than the display width.
	VAlign (see page 25098)	Specifies the vertical alignment of the generated display control.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	ViewMode (see page 25252)	Indicates the viewing state of the adapter field.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

181.1.24.2 TCustomAdapterEditColumn Methods

The methods of the TCustomAdapterEditColumn class are listed here.

Protected Methods

	Name	Description
V	ImplContent (see page 25258)	Provides the underlying implementation for the Content method.

181.1.24.2.1 TCustomAdapterEditColumn.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML that defines the display control and its label. ImplContent formats the caption and combines this HTML with the results of the ControlContent method. ControlContent is an abstract (pure virtual) method in TWebDataDisplay. TWebDataDisplay descendants override it to generate the definition of their particular display control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebDataDisplay should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the display control.

See Also

Content (see page 16890)

Caption (see page 16897)

ILayoutWebContent (see page 24774)

181.1.25 TCustomAdapterErrorList Class

TCustomAdapterErrorList is the base class adapter for components that display error lists.

Class Hierarchy

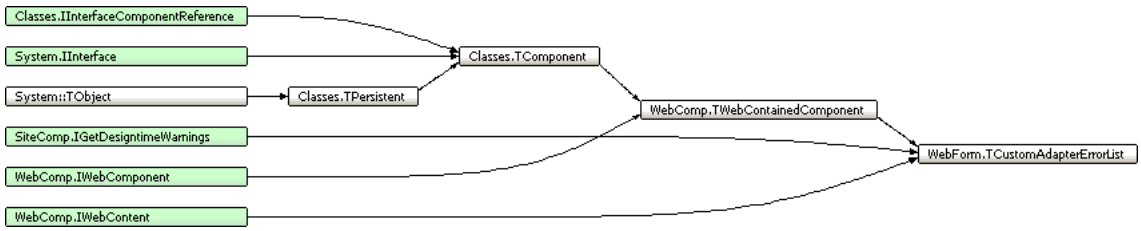

Pascal

```
TCustomAdapterErrorList = class(TWebContainedComponent, IWebContent, IGetDesigntimeWarnings);
```

C++

```
class TCustomAdapterErrorList : public TWebContainedComponent, public IWebContent, public
IGetDesigntimeWarnings;
```

File

WebForm

Description

TCustomAdapterErrorList is the base class for adapter components that display error lists. Do not instantiate TCustomAdapterErrorList directly. Use TAdapterErrorList (see page 25145), or create your own concrete descendant class.

181.1.25.1 TCustomAdapterErrorList Members

The following tables list the members exposed by TCustomAdapterErrorList.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.

	Content (see page 25263)	Returns the HTML that the Web item generates.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetContainerList (see page 24727)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDesigntimeWarnings (see page 25264)	This is GetDesigntimeWarnings, a member of class TCustomAdapterErrorList.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	ImplContent (see page 25264)	This is ImplContent, a member of class TCustomAdapterErrorList.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24728)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 25264)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24728)	Responds when the WebParent (see page 24731) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24729)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24729)	Sets the WebParent (see page 24731) property to be the same as this component's parent.
	SetWebParent (see page 24730)	Sets the value of the WebParent (see page 24731) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 24730)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.

R	WebParent (see page 24731)	Indicates the component that uses this TWebContainedComponent (see page 24722) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 25262)	Creates an instance of TCustomAdapterErrorList (see page 25259).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25263)	Frees the memory associated with an instance of TCustomAdapterErrorList (see page 25259).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24727)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24727)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Adapter (see page 25265)	Specifies a related adapter.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

181.1.25.2 TCustomAdapterErrorList.Create Constructor

Creates an instance of TCustomAdapterErrorList (see page 25259).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterErrorList(TComponent * AOwner);
```

Description

Call Create to create and initialize an instance of TCustomAdapterErrorList (see page 25259).

See Also

Create_d

181.1.25.3 TCustomAdapterErrorList.Destroy Destructor

Frees the memory associated with an instance of TCustomAdapterErrorList (see page 25259).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterErrorList();
```

Description

Call Destroy to destroy an instance of TCustomAdapterErrorList (see page 25259) and free the memory associated with it.

181.1.25.4 TCustomAdapterErrorList Methods

The methods of the TCustomAdapterErrorList class are listed here.

Protected Methods

	Name	Description
	Content (see page 25263)	Returns the HTML that the Web item generates.
	GetDesigntimeWarnings (see page 25264)	This is GetDesigntimeWarnings, a member of class TCustomAdapterErrorList.
V	ImplContent (see page 25264)	This is ImplContent, a member of class TCustomAdapterErrorList.
V	Notification (see page 25264)	Forwards notification messages to all owned components.

181.1.25.4.1 TCustomAdapterErrorList.Content Method

Returns the HTML that the Web item generates.

Pascal

```
function Content(Options: TWebContentOptions; Layout: TLayout): string;
```

C++

```
__fastcall AnsiString Content(TWebContentOptions Options, TLayout Layout);
```

Description

Call Content to access the HTML produced by the Web item.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web item should lay out its constituent parts. This object supports the ILayoutWebContent interface, which the Web item uses to fit its own content into the parent's HTML structures.

Content returns a string of HTML produced by the Web item.

See Also

ILayoutWebContent (see page 24774)

181.1.25.4.2 TCustomAdapterErrorList.GetDesigntimeWarnings Method

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

This is GetDesigntimeWarnings, a member of class TCustomAdapterErrorList.

181.1.25.4.3 TCustomAdapterErrorList.ImplContent Method

Pascal

```
function ImplContent(Options: TWebContentOptions; Layout: TLayout): string; virtual;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout Layout);
```

Description

This is ImplContent, a member of class TCustomAdapterErrorList.

181.1.25.4.4 TCustomAdapterErrorList.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

181.1.25.5 TCustomAdapterErrorList Properties

The properties of the TCustomAdapterErrorList class are listed here.

Public Properties

	Name	Description
	Adapter (see page 25265)	Specifies a related adapter.

181.1.25.5.1 TCustomAdapterErrorList.Adapter Property

Specifies a related adapter.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

Adapter specifies the adapter component for which this component reports errors.

181.1.26 TCustomAdapterFieldGroup Class

TCustomAdapterFieldGroup manages a collection of Web Items that generate a group of HTML controls.

Class Hierarchy

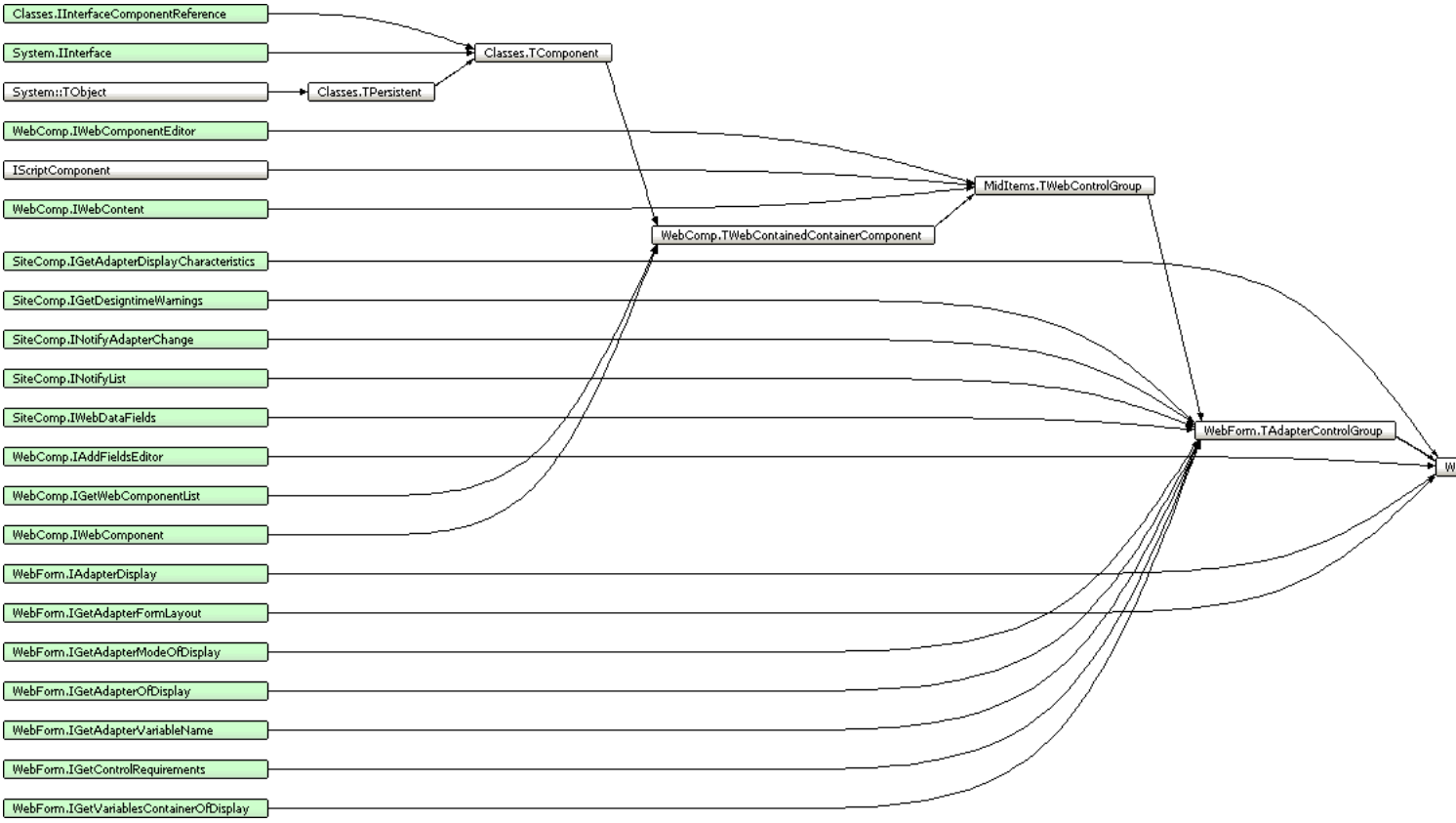

Pascal

```
TCustomAdapterFieldGroup = class(TAdapterControlGroup, IAdapterDisplay,
IGetAdapterDisplayCharacteristics, IAddFieldsEditor, IGetAdapterFormLayout);
```

C++

```
class TCustomAdapterFieldGroup : public TAdapterControlGroup, public IAdapterDisplay, public
IGetAdapterDisplayCharacteristics, public IAddFieldsEditor, public IGetAdapterFormLayout;
```

File

WebForm

Description

Use TCustomAdapterFieldGroup as a base class for components that generate organized portions of the HTML document containing one or more related HTML controls. TCustomAdapterFieldGroup introduces the properties and methods for managing a collection of Web items that generate the HTML controls that make up the group.

In order to let other Web items use the HTML that defines the organized collection of controls, TCustomAdapterFieldGroup implements the IWebContent interface.

Do not create instances of TCustomAdapterFieldGroup. TCustomAdapterFieldGroup manages the Web items that generate HTML defining controls, but does not combine the generated content into any content of its own. Instead, use a descendant of TCustomAdapterFieldGroup such as TAdapterFieldGroup (see page 25150).

See Also

TAdapterGrid (see page 25175)

TAdapterActionButton (see page 25034)

IWebContent (see page 24792)

181.1.26.1 TCustomAdapterFieldGroup Members

The following tables list the members exposed by TCustomAdapterFieldGroup.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AdapterChange (see page 25077)	Responds when the adapter associated with this control group changes.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
	AddNotify (see page 25077)	Registers interest so that the adapter control group notifies an object of changes to its adapter.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
	CreateDefaultFields (see page 25078)	Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FindField (see page 25078)	Returns the child Web item that represents the value of a specified adapter field.
	GetAdapter (see page 25079)	Returns the value of the Adapter (see page 25084) property.
	GetAdapterDisplayCharacteristics (see page 25272)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterFieldGroup.
	GetAdapterFormLayout (see page 25272)	This is GetAdapterFormLayout, a member of class TCustomAdapterFieldGroup.
	GetAdapterModeOfAdapter (see page 25079)	Returns the current mode of the control group's adapter.
	GetAdapterModeOfDisplay (see page 25080)	Returns the value of the AdapterMode (see page 25085) property.
	GetAdapterVariableName (see page 25080)	Returns the variable name that refers to the control group's adapter in server-side script.
	GetAddAllFieldsItem (see page 25273)	This is GetAddAllFieldsItem, a member of class TCustomAdapterFieldGroup.
	GetAddFieldsDlgCaption (see page 25273)	This is GetAddFieldsDlgCaption, a member of class TCustomAdapterFieldGroup.
	GetAddFieldsItem (see page 25273)	This is GetAddFieldsItem, a member of class TCustomAdapterFieldGroup.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetControlRequirements (see page 25080)	Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.

	GetDesigntimeWarnings (see page 25081)	Populates a design-time warnings object with any warnings that the adapter control group needs to emit.
	GetFieldsList (see page 25081)	Fills a string list with the names of all of the associated adapter's fields.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetLayoutAttributes (see page 25273)	Returns an object that specifies how the generated group of controls should look.
	GetNewFieldPrefix (see page 25274)	This is GetNewFieldPrefix, a member of class TCustomAdapterFieldGroup.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVisibleFields (see page 25082)	Returns the value of the VisibleFields (see page 25086) property.
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
	ImplCanAddClass (see page 25274)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 25274)	Provides the underlying implementation for the Content method.
	ImplGetControlRequirements (see page 25082)	Provides the underlying implementation for the GetControlRequirements (see page 25080) method.
	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsFieldInUse (see page 25082)	Indicates whether one of the adapter control group's child Web items represents a specified adapter field.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 25083)	Updates the TAdapterControlGroup (see page 25070) object when components are added to or removed from the application.
	NotifyAdapterChange (see page 25083)	Responds when the control group's adapter notifies it of a change.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	RemoveNotify (see page 25083)	Unregisters interest in changes to the control group's adapter.
	SetAdapter (see page 25084)	Sets the value of the Adapter (see page 25084) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.

	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 25271)	Creates and initializes an instance of TCustomAdapterFieldGroup (see page 25265).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25272)	Frees an instance of TCustomAdapterFieldGroup (see page 25265).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapter (see page 25306)	This is GetAdapter, a member of class IGetAdapterOfDisplay.
	GetAdapterModeOfAdapter (see page 25305)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 25305)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.
	GetAdapterVariableName (see page 25307)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetControlRequirements (see page 25309)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetDesigntimeWarnings (see page 19697)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	GetVariablesContainer (see page 25311)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NotifyAdapterChange (see page 19725)	Responds when the adapter changes.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Adapter (see page 25084)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 25085)	Specifies the mode of the adapter that the controls in the group support.
R	AdapterVariableName (see page 25308)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	AddDefaultFields (see page 25085)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 25275)	Represents a tag option and its value in the generated INPUT tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 25275)	Adds style attributes to the generated button group.
	StyleRule (see page 25276)	Applies a style in the page producer's style sheet to the button group.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.
R	VisibleFields (see page 25086)	Lists the child Web items that generate the visible HTML controls this adapter control group arranges.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

181.1.26.2 TCustomAdapterFieldGroup.Create Constructor

Creates and initializes an instance of TCustomAdapterFieldGroup (see page 25265).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterFieldGroup(TComponent * AOwner);
```

Description

Call Create to instantiate TCustomAdapterFieldGroup (see page 25265) at runtime. TCustomAdapterFieldGroup (see page 25265) objects added at design time using the Web Page editor are instantiated automatically.

181.1.26.3 TCustomAdapterFieldGroup.Destroy Destructor

Frees an instance of TCustomAdapterFieldGroup (see page 25265).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterFieldGroup();
```

Description

Destroy removes the Web control group from its parent's list of components that generate HTML before freeing its own resources.

181.1.26.4 TCustomAdapterFieldGroup Methods

The methods of the TCustomAdapterFieldGroup class are listed here.

Protected Methods

	Name	Description
	GetAdapterDisplayCharacteristics (see page 25272)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterFieldGroup.
	GetAdapterFormLayout (see page 25272)	This is GetAdapterFormLayout, a member of class TCustomAdapterFieldGroup.
	GetAddAllFieldsItem (see page 25273)	This is GetAddAllFieldsItem, a member of class TCustomAdapterFieldGroup.
	GetAddFieldsDlgCaption (see page 25273)	This is GetAddFieldsDlgCaption, a member of class TCustomAdapterFieldGroup.
	GetAddFieldsItem (see page 25273)	This is GetAddFieldsItem, a member of class TCustomAdapterFieldGroup.
V	GetLayoutAttributes (see page 25273)	Returns an object that specifies how the generated group of controls should look.
	GetNewFieldPrefix (see page 25274)	This is GetNewFieldPrefix, a member of class TCustomAdapterFieldGroup.
V	ImplCanAddClass (see page 25274)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 25274)	Provides the underlying implementation for the Content method.

181.1.26.4.1 TCustomAdapterFieldGroup.GetAdapterDisplayCharacteristics Method

Pascal

```
function GetAdapterDisplayCharacteristics: TAdapterDisplayCharacteristics;
```

C++

```
__fastcall TAdapterDisplayCharacteristics GetAdapterDisplayCharacteristics();
```

Description

This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterFieldGroup.

181.1.26.4.2 TCustomAdapterFieldGroup.GetAdapterFormLayout Method

Pascal

```
function GetAdapterFormLayout: TAdapterFormLayout;
```

C++

```
__fastcall TAdapterFormLayout GetAdapterFormLayout();
```

Description

This is GetAdapterFormLayout, a member of class TCustomAdapterFieldGroup.

181.1.26.4.3 TCustomAdapterFieldGroup.GetAddAllFieldsItem Method

Pascal

```
function GetAddAllFieldsItem: string;
```

C++

```
__fastcall AnsiString GetAddAllFieldsItem();
```

Description

This is GetAddAllFieldsItem, a member of class TCustomAdapterFieldGroup.

181.1.26.4.4 TCustomAdapterFieldGroup.GetAddFieldsDlgCaption Method

Pascal

```
function GetAddFieldsDlgCaption: string;
```

C++

```
__fastcall AnsiString GetAddFieldsDlgCaption();
```

Description

This is GetAddFieldsDlgCaption, a member of class TCustomAdapterFieldGroup.

181.1.26.4.5 TCustomAdapterFieldGroup.GetAddFieldsItem Method

Pascal

```
function GetAddFieldsItem: string;
```

C++

```
__fastcall AnsiString GetAddFieldsItem();
```

Description

This is GetAddFieldsItem, a member of class TCustomAdapterFieldGroup.

181.1.26.4.6 TCustomAdapterFieldGroup.GetLayoutAttributes Method

Returns an object that specifies how the generated group of controls should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content method) to take into account layout properties of the child component.

As implemented by TWebControlGroup, GetLayoutAttributes returns nil (Delphi) or NULL (C++), because TWebControlGroup descendants not make use of layout attributes.

See Also

Content (see page 16879)

181.1.26.4.7 TCustomAdapterFieldGroup.GetNewFieldPrefix Method

Pascal

```
function GetNewFieldPrefix: string;
```

C++

```
__fastcall AnsiString GetNewFieldPrefix();
```

Description

This is GetNewFieldPrefix, a member of class TCustomAdapterFieldGroup.

181.1.26.4.8 TCustomAdapterFieldGroup.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass (see page 16878)

WebFieldControls (see page 16884)

181.1.26.4.9 TCustomAdapterFieldGroup.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup, ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the

HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content (see page 16879)

AddElements (see page 16878)

ILayoutWebContent (see page 24774)

181.1.26.5 TCustomAdapterFieldGroup Properties

The properties of the TCustomAdapterFieldGroup class are listed here.

Public Properties

	Name	Description
	Custom (see page 25275)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 25275)	Adds style attributes to the generated button group.
	StyleRule (see page 25276)	Applies a style in the page producer's style sheet to the button group.

181.1.26.5.1 TCustomAdapterFieldGroup.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to represent one or more options that qualify the INPUT tag generated by the Web button group. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

```
ALIGN=TOP TABINDEX=2
```

See Also

Style (see page 25275)

181.1.26.5.2 TCustomAdapterFieldGroup.Style Property

Adds style attributes to the generated button group.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to add a STYLE option that qualifies the INPUT tag generated by the Web button group. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated INPUT tag as the value of a STYLE option.

See Also

Custom (see page 25275)

StyleRule (see page 25276)

181.1.26.5.3 TCustomAdapterFieldGroup.StyleRule Property

Applies a style in the page producer's style sheet to the button group.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated button group. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the button's Style (see page 25275) property.

See Also

Style (see page 25275)

Styles (see page 19830)

StylesFile (see page 19831)

181.1.27 TCustomAdapterForm Class

TCustomAdapterForm is the base class for components that generate an HTML form for an adapter page producer.

Class Hierarchy

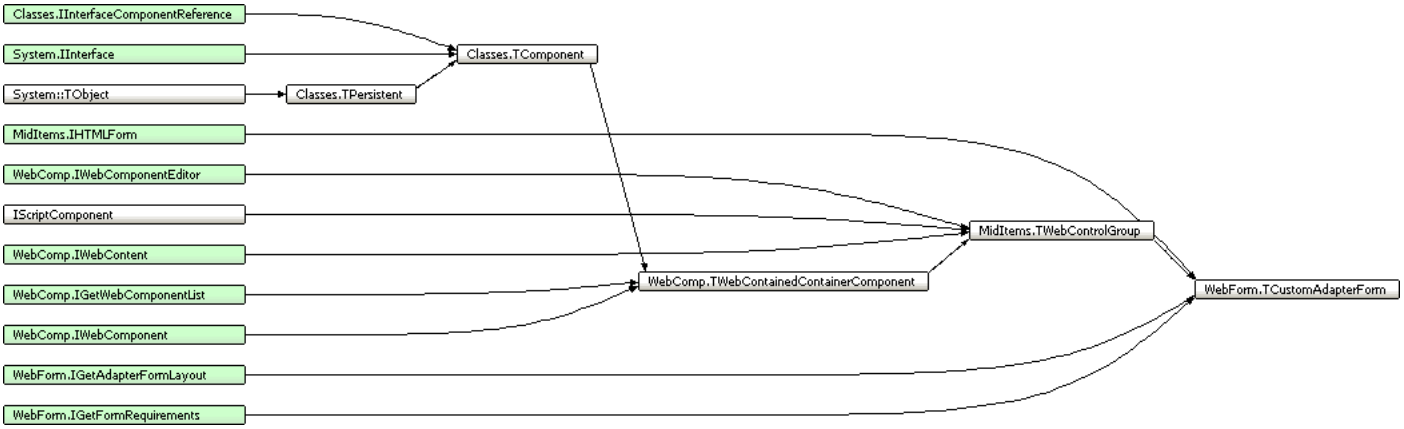

Pascal

```
TCustomAdapterForm = class(TWebControlGroup, IHTMLForm, IGetFormRequirements,
IGetAdapterFormLayout);
```

C++

```
class TCustomAdapterForm : public TWebControlGroup, public IHTMLForm, public
IGetFormRequirements, public IGetAdapterFormLayout;
```

File

WebForm

Description

Use TCustomAdapterForm as a base class when defining a form that can contain adapter fields and actions. TCustomAdapterForm is an abstract class, and should not be instantiated directly. Use TAdapterForm (see page 25159), which is a minimal implementation of TCustomAdapterForm, or create your own implementation.

See Also

TDataNavigator (see page 16569)

TQueryButtons (see page 16730)

TDataGrid (see page 16560)

TFieldGroup (see page 16591)

TQueryFieldGroup (see page 16736)

IWebContent (see page 24792)

181.1.27.1 TCustomAdapterForm Members

The following tables list the members exposed by TCustomAdapterForm.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetAdapterFormLayout (see page 25282)	This is GetAdapterFormLayout, a member of class TCustomAdapterForm.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetFormRequirements (see page 25282)	This is GetFormRequirements, a member of class TCustomAdapterForm.
	GetHTMLFormName (see page 25282)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 25283)	Returns the name of the javascript variable that refers to the generated form.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetLayoutAttributes (see page 25283)	Returns an object that specifies how the generated group of controls should look.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
	ImplCanAddClass (see page 25283)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 25284)	Provides the underlying implementation for the Content method.
	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.

V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.

	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 25281)	Creates and initializes an instance of TCustomAdapterForm (see page 25276).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25281)	Frees an instance of TCustomAdapterForm (see page 25276).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
V	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
V	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	Custom (see page 25285)	Represents a tag option and its value in the generated INPUT tag.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	FormMethod (see page 25285)	Specifies the method that the adapter form adds to its generated <form> tag.
R	HTMLFormName (see page 17066)	Indicates the name of the generated HTML form.
R	HTMLFormVarName (see page 17066)	Indicates the name of the javascript variable that represents the HTML form.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	Style (see page 25285)	Adds style attributes to the generated form.
	StyleRule (see page 25286)	Applies a style in the page producer's style sheet to the form.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

181.1.27.2 TCustomAdapterForm.Create Constructor

Creates and initializes an instance of TCustomAdapterForm (see page 25276).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterForm(TComponent * AOwner);
```

Description

Call Create to instantiate TCustomAdapterForm (see page 25276) at runtime. TCustomAdapterForm (see page 25276) objects added at design time using the Web Page editor are instantiated automatically.

181.1.27.3 TCustomAdapterForm.Destroy Destructor

Frees an instance of TCustomAdapterForm (see page 25276).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterForm();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks for that the object is not nil before calling Destroy.

See Also

Free (see page 22300)

181.1.27.4 TCustomAdapterForm Methods

The methods of the TCustomAdapterForm class are listed here.

Protected Methods

	Name	Description
	GetAdapterFormLayout (see page 25282)	This is GetAdapterFormLayout, a member of class TCustomAdapterForm.
	GetFormRequirements (see page 25282)	This is GetFormRequirements, a member of class TCustomAdapterForm.
	GetHTMLFormName (see page 25282)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 25283)	Returns the name of the javascript variable that refers to the generated form.
V	GetLayoutAttributes (see page 25283)	Returns an object that specifies how the generated group of controls should look.
V	ImplCanAddClass (see page 25283)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 25284)	Provides the underlying implementation for the Content method.

181.1.27.4.1 TCustomAdapterForm.GetAdapterFormLayout Method

Pascal

```
function GetAdapterFormLayout: TAdapterFormLayout;
```

C++

```
__fastcall TAdapterFormLayout GetAdapterFormLayout();
```

Description

This is GetAdapterFormLayout, a member of class TCustomAdapterForm.

181.1.27.4.2 TCustomAdapterForm.GetFormRequirements Method

Pascal

```
procedure GetFormRequirements(ARequirements: TAbstractFormRequirements);
```

C++

```
__fastcall GetFormRequirements(TAbstractFormRequirements ARequirements);
```

Description

This is GetFormRequirements, a member of class TCustomAdapterForm.

181.1.27.4.3 TCustomAdapterForm.GetHTMLFormName Method

Returns the name of the generated HTML form.

Pascal

```
function GetHTMLFormName: string;
```

C++

```
__fastcall AnsiString GetHTMLFormName();
```

Description

GetHTMLFormName is the read access method for the HTMLFormName property.

See Also

HTMLFormName (see page 17066)

181.1.27.4.4 TCustomAdapterForm.GetHTMLFormVarName Method

Returns the name of the javascript variable that refers to the generated form.

Pascal

```
function GetHTMLFormVarName: string;
```

C++

```
__fastcall AnsiString GetHTMLFormVarName();
```

Description

GetHTMLFormVarName is the read access method for the HTMLFormVarName property.

See Also

HTMLFormVarName (see page 17066)

181.1.27.4.5 TCustomAdapterForm.GetLayoutAttributes Method

Returns an object that specifies how the generated group of controls should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content method) to take into account layout properties of the child component.

As implemented by TWebControlGroup, GetLayoutAttributes returns nil (Delphi) or NULL (C++), because TWebControlGroup descendants not make use of layout attributes.

See Also

Content (see page 16879)

181.1.27.4.6 TCustomAdapterForm.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass (see page 16878)

WebFieldControls (see page 16884)

181.1.27.4.7 TCustomAdapterForm.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup, ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content (see page 16879)

AddElements (see page 16878)

ILayoutWebContent (see page 24774)

181.1.27.5 TCustomAdapterForm Properties

The properties of the TCustomAdapterForm class are listed here.

Public Properties

	Name	Description
	Custom (see page 25285)	Represents a tag option and its value in the generated INPUT tag.
	FormMethod (see page 25285)	Specifies the method that the adapter form adds to its generated <form> tag.
	Style (see page 25285)	Adds style attributes to the generated form.
	StyleRule (see page 25286)	Applies a style in the page producer's style sheet to the form.

181.1.27.5.1 TCustomAdapterForm.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property AnsiString Custom;
```

Description

Use Custom to represent one or more options that qualify the <table> tag that the adapter form generates to manage the layout of its Web items. Each option and its value has the form

`OPTIONNAME=Value`

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

ALIGN=TOP TABINDEX=2

See Also

Style (see page 25285)

181.1.27.5.2 TCustomAdapterForm.FormMethod Property

Specifies the method that the adapter form adds to its generated <form> tag.

Pascal

```
property FormMethod: TFormMethod;
```

C++

```
__property TFormMethod FormMethod;
```

Description

Set FormMethod to specify the METHOD header for HTTP messages that the form submits. This value gives rise to the MethodType property of the HTTP request message.

See Also

MethodType

181.1.27.5.3 TCustomAdapterForm.Style Property

Adds style attributes to the generated form.

Pascal

```
property Style: string;
```

C++

```
__property AnsiString Style;
```

Description

Use Style to add a STYLE option that qualifies the <table> tag that the adapter form generates to manage the layout of its Web items. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated <table> tag as the value of a STYLE option.

See Also

Custom (see page 25285)

StyleRule (see page 25286)

181.1.27.5.4 TCustomAdapterForm.StyleRule Property

Applies a style in the page producer's style sheet to the form.

Pascal

```
property StyleRule: string;
```

C++

```
__property AnsiString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated form. The value of StyleRule specifies one of the user-defined style names defined in the adapter page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the form's Style (see page 25285) property.

See Also

Style (see page 25285)

Styles (see page 19830)

StylesFile (see page 19831)

181.1.28 TCustomAdapterGrid Class

TCustomAdapterGrid is the base class for Web items that generate an HTML table to display a datasetin an XML data packet or web application.

Class Hierarchy

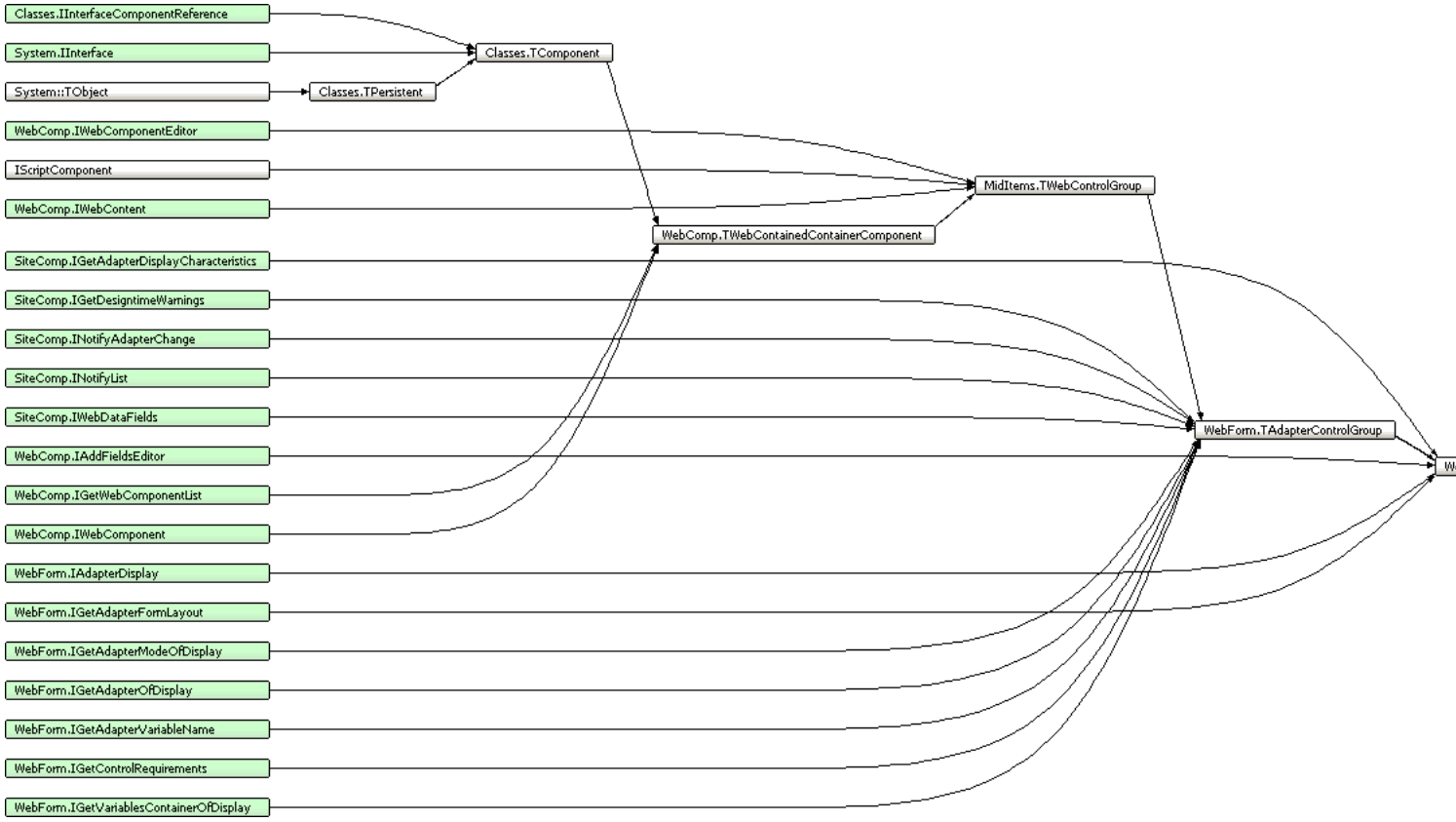

Pascal

```
TCustomAdapterGrid = class(TAdapterControlGroup, IAdapterDisplay,
IGetAdapterDisplayCharacteristics, IAddFieldsEditor, IGetAdapterFormLayout);
```

C++

```
class TCustomAdapterGrid : public TAdapterControlGroup, public IAdapterDisplay, public
IGetAdapterDisplayCharacteristics, public IAddFieldsEditor, public IGetAdapterFormLayout;
```

File

WebForm

Description

Use TCustomAdapterGrid as a base class for Web items that generate an HTML table to display multiple records of data supplied by an adapter component. All of the adapter grid's child Web items display field components that use the same adapter, which is specified by a property of the adapter grid.

If no display fields are explicitly added to the adapter grid, it displays all of the adapter's display fields by default. The properties of default display fields cannot be modified.

TCustomAdapterGrid introduces properties to

Specify the associated Adapter (see page 25084)

Specify whether default fields should be added

Customize the appearance of the HTML table.

Do not create instances of TCustomAdapterGrid. Instead, use a descendant of TCustomAdapterGrid such as TAdapterGrid (see page 25175).

See Also

TAdapterCommandColumn (see page 25052)

TAdapterGrid (see page 25175)

TAdapterFieldGroup (see page 25150)

IWebContent (see page 24792)

181.1.28.1 TCustomAdapterGrid Members

The following tables list the members exposed by TCustomAdapterGrid.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AdapterChange (see page 25077)	Responds when the adapter associated with this control group changes.
	AddElements (see page 16878)	Used internally to generate any script declarations needed by the group of controls.
	AddNotify (see page 25077)	Registers interest so that the adapter control group notifies an object of changes to its adapter.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	CanAddClass (see page 16878)	Specifies whether a Web item of a specified type can be added to the WebFieldControls (see page 16884) property list.
	ChangeFieldControls (see page 16879)	Responds when the group of HTML controls changes.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	Content (see page 16879)	Returns the HTML that defines the group of HTML controls.
	CreateDefaultFields (see page 25078)	Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	FindField (see page 25078)	Returns the child Web item that represents the value of a specified adapter field.
	GetAdapter (see page 25079)	Returns the value of the Adapter (see page 25084) property.
	GetAdapterDisplayCharacteristics (see page 25293)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterGrid.
	GetAdapterFormLayout (see page 25294)	This is GetAdapterFormLayout, a member of class TCustomAdapterGrid.
	GetAdapterModeOfAdapter (see page 25079)	Returns the current mode of the control group's adapter.
	GetAdapterModeOfDisplay (see page 25080)	Returns the value of the AdapterMode (see page 25085) property.
	GetAdapterVariableName (see page 25080)	Returns the variable name that refers to the control group's adapter in server-side script.
	GetAddAllFieldsItem (see page 25294)	This is GetAddAllFieldsItem, a member of class TCustomAdapterGrid.
	GetAddFieldsDlgCaption (see page 25294)	This is GetAddFieldsDlgCaption, a member of class TCustomAdapterGrid.
	GetAddFieldsItem (see page 25294)	This is GetAddFieldsItem, a member of class TCustomAdapterGrid.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.

	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 24743)	Executes a callback for each child Web item.
	GetContainerList (see page 24744)	Returns the TWebComponentList (see page 24708) object that the Web parent uses to manage this and its other Web items.
	GetControlRequirements (see page 25080)	Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.
V	GetDefaultWebComponents (see page 16880)	Returns the value of the DefaultWebComponents (see page 16883) property.
V	GetDefaultWebComponents (see page 24744)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetDesigntimeWarnings (see page 25081)	Populates a design-time warnings object with any warnings that the adapter control group needs to emit.
	GetFieldsList (see page 25081)	Fills a string list with the names of all of the associated adapter's fields.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetLayoutAttributes (see page 16880)	Returns an object that specifies how the generated group of controls should look.
	GetNewFieldPrefix (see page 25294)	This is GetNewFieldPrefix, a member of class TCustomAdapterGrid.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSubComponents (see page 16880)	Returns the value of the WebFieldControls (see page 16884) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVisibleFields (see page 25082)	Returns the value of the VisibleFields (see page 25086) property.
	GetWebComponents (see page 24745)	Returns the list of Web items that this item uses to generate its HTML.
V	ImplAddElements (see page 16881)	Provides the underlying implementation for the AddElements (see page 16878) method.
V	ImplCanAddClass (see page 25295)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 25295)	Provides the underlying implementation for the Content method.
V	ImplGetControlRequirements (see page 25082)	Provides the underlying implementation for the GetControlRequirements (see page 25080) method.
V	ImplGetDefaultComponentList (see page 24746)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetSubComponents (see page 16882)	Provides the underlying implementation for the GetSubComponents (see page 16880) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	IsDefaultField (see page 24746)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsFieldInUse (see page 25082)	Indicates whether one of the adapter control group's child Web items represents a specified adapter field.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 25083)	Updates the TAdapterControlGroup (see page 25070) object when components are added to or removed from the application.
	NotifyAdapterChange (see page 25083)	Responds when the control group's adapter notifies it of a change.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	ParentChanged (see page 24747)	Responds when the WebParent (see page 24750) property changes.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 24747)	Reads the component's published properties, stored data, and owned components from a reader object.
	RemoveNotify (see page 25083)	Unregisters interest in changes to the control group's adapter.
	SetAdapter (see page 25084)	Sets the value of the Adapter (see page 25084) property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
V	SetChildOrder (see page 24748)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 16883)	Sets the value of the DefaultWebComponents (see page 16883) property.
V	SetDefaultWebComponents (see page 24748)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
V	SetParentComponent (see page 24748)	Sets the WebParent (see page 24750) property to be the same as this component's parent.
	SetWebParent (see page 24749)	Sets the value of the WebParent (see page 24750) property.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	DefaultWebComponents (see page 16883)	Specifies the default value of the WebFieldControls (see page 16884) property when that property is not explicitly set.
R	WebContainedComponentHelper (see page 24749)	Provides access to the helper object that implements the IWebComponent (see page 24782) interface for this contained component.
R	WebContainerComponentHelper (see page 24750)	Provides access to the helper object that allows this object to manage sub Web items.
R	WebParent (see page 24750)	Indicates the component that uses this TWebContainedContainerComponent (see page 24738) as one of its Web items.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 19726)	Registers an interest in change notifications.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CanAddClass (see page 24791)	Indicates whether a specified class can be used as a child Web item.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Content (see page 24793)	Returns the HTML that the Web item generates.
V	Create (see page 25292)	Creates and initializes an instance of TCustomAdapterGrid (see page 25286).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25293)	Frees an instance of TAdapterControlGroup.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindField (see page 19749)	This is FindField, a member of class IWebDataFields.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetAdapter (see page 25306)	This is GetAdapter, a member of class IGetAdapterOfDisplay.
	GetAdapterModeOfAdapter (see page 25305)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 25305)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.
	GetAdapterVariableName (see page 25307)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetComponentList (see page 24772)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 24784)	Returns the value of the Container (see page 24786) property.
	GetControlRequirements (see page 25309)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDefaultComponentList (see page 24772)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetDesigntimeWarnings (see page 19697)	This is GetDesigntimeWarnings, a member of class IGetDesigntimeWarnings.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFieldsList (see page 19750)	This is GetFieldsList, a member of class IWebDataFields.
	GetIndex (see page 24784)	Returns the value of the Index (see page 24786) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParentComponent (see page 24745)	Returns the object that uses this component to generate HTML.
	GetVariablesContainer (see page 25311)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	GetVisibleFields (see page 19750)	This is GetVisibleFields, a member of class IWebDataFields.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasParent (see page 24745)	Indicates whether this component appears in another component's list of Web items.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsFieldInUse (see page 19750)	This is IsFieldInUse, a member of class IWebDataFields.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NotifyAdapterChange (see page 19725)	Responds when the adapter changes.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.

	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 19726)	Unregisters interest in change notifications.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetComponentList (see page 24784)	Sets the value of the ComponentList (see page 24785) property.
	SetContainer (see page 24784)	Sets the value of the Container (see page 24786) property.
	SetIndex (see page 24785)	Sets the value of the Index (see page 24786) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Adapter (see page 25084)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 25085)	Specifies the mode of the adapter that the controls in the group support.
R	AdapterVariableName (see page 25308)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	AddDefaultFields (see page 25085)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
W	ComponentList (see page 24785)	Indicates the component list that contains the Web item.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Container (see page 24786)	Specifies the component that uses content generated by this Web item.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	HeadingAttributes (see page 25296)	Determines the display characteristics of the table's header.
	Index (see page 24786)	Indicates where the Web item appears in the Web component list's array of Web items.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	RowAttributes (see page 25296)	Determines the display characteristics of the table's body.
	TableAttributes (see page 25297)	Determines the basic display characteristics of the table.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
R	VisibleFields (see page 19751)	This is VisibleFields, a member of class IWebDataFields.
R	VisibleFields (see page 25086)	Lists the child Web items that generate the visible HTML controls this adapter control group arranges.
R	WebFieldControls (see page 16884)	Lists the components that generate HTML controls for the Web control group.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

181.1.28.2 TCustomAdapterGrid.Create Constructor

Creates and initializes an instance of TCustomAdapterGrid (see page 25286).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterGrid(TComponent * AOwner);
```

Description

Call Create to instantiate TCustomAdapterGrid (see page 25286) at runtime. TCustomAdapterGrid (see page 25286) objects added at design time using the Web Page editor are instantiated automatically.

181.1.28.3 TCustomAdapterGrid.Destroy Destructor

Frees an instance of TAdapterControlGroup.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterGrid();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks for that the object is not nil before calling Destroy.

See Also

Free (see page 22300)

181.1.28.4 TCustomAdapterGrid Methods

The methods of the TCustomAdapterGrid class are listed here.

Protected Methods

	Name	Description
	GetAdapterDisplayCharacteristics (see page 25293)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterGrid.
	GetAdapterFormLayout (see page 25294)	This is GetAdapterFormLayout, a member of class TCustomAdapterGrid.
	GetAddAllFieldsItem (see page 25294)	This is GetAddAllFieldsItem, a member of class TCustomAdapterGrid.
	GetAddFieldsDlgCaption (see page 25294)	This is GetAddFieldsDlgCaption, a member of class TCustomAdapterGrid.
	GetAddFieldsItem (see page 25294)	This is GetAddFieldsItem, a member of class TCustomAdapterGrid.
	GetNewFieldPrefix (see page 25294)	This is GetNewFieldPrefix, a member of class TCustomAdapterGrid.
V	ImplCanAddClass (see page 25295)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 25295)	Provides the underlying implementation for the Content method.

181.1.28.4.1 TCustomAdapterGrid.GetAdapterDisplayCharacteristics Method

Pascal

```
function GetAdapterDisplayCharacteristics: TAdapterDisplayCharacteristics;
```

C++

```
__fastcall TAdapterDisplayCharacteristics GetAdapterDisplayCharacteristics();
```

Description

This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterGrid.

181.1.28.4.2 TCustomAdapterGrid.GetAdapterFormLayout Method

Pascal

```
function GetAdapterFormLayout: TAdapterFormLayout;
```

C++

```
__fastcall TAdapterFormLayout GetAdapterFormLayout();
```

Description

This is GetAdapterFormLayout, a member of class TCustomAdapterGrid.

181.1.28.4.3 TCustomAdapterGrid.GetAddAllFieldsItem Method

Pascal

```
function GetAddAllFieldsItem: string;
```

C++

```
__fastcall AnsiString GetAddAllFieldsItem();
```

Description

This is GetAddAllFieldsItem, a member of class TCustomAdapterGrid.

181.1.28.4.4 TCustomAdapterGrid.GetAddFieldsDlgCaption Method

Pascal

```
function GetAddFieldsDlgCaption: string;
```

C++

```
__fastcall AnsiString GetAddFieldsDlgCaption();
```

Description

This is GetAddFieldsDlgCaption, a member of class TCustomAdapterGrid.

181.1.28.4.5 TCustomAdapterGrid.GetAddFieldsItem Method

Pascal

```
function GetAddFieldsItem: string;
```

C++

```
__fastcall AnsiString GetAddFieldsItem();
```

Description

This is GetAddFieldsItem, a member of class TCustomAdapterGrid.

181.1.28.4.6 TCustomAdapterGrid.GetNewFieldPrefix Method

Pascal

```
function GetNewFieldPrefix: string;
```

C++

```
__fastcall AnsiString GetNewFieldPrefix();
```

Description

This is GetNewFieldPrefix, a member of class TCustomAdapterGrid.

181.1.28.4.7 TCustomAdapterGrid.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass (see page 16878)

WebFieldControls (see page 16884)

181.1.28.4.8 TCustomAdapterGrid.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall AnsiString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup, ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content (see page 16879)

AddElements (see page 16878)

ILayoutWebContent (see page 24774)

181.1.28.5 TCustomAdapterGrid Properties

The properties of the TCustomAdapterGrid class are listed here.

Public Properties

	Name	Description
	HeadingAttributes (see page 25296)	Determines the display characteristics of the table's header.
	RowAttributes (see page 25296)	Determines the display characteristics of the table's body.
	TableAttributes (see page 25297)	Determines the basic display characteristics of the table.

181.1.28.5.1 TCustomAdapterGrid.HeadingAttributes Property

Determines the display characteristics of the table's header.

Pascal

```
property HeadingAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes HeadingAttributes;
```

Description

Use HeadingAttributes to specify any display attributes that apply only to the heading of the generated HTML table. HeadingAttributes determines the options that are added to the <TR> tag that defines the header of the grid. These include the color of the header, the alignment of the caption for each column, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the rows of the grid, use the TableAttributes (see page 25297) or RowAttributes (see page 25296) property instead.

See Also

TableAttributes (see page 25297)

RowAttributes (see page 25296)

181.1.28.5.2 TCustomAdapterGrid.RowAttributes Property

Determines the display characteristics of the table's body.

Pascal

```
property RowAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes RowAttributes;
```

Description

Use RowAttributes to specify any display attributes that apply only to the rows of the generated HTML table. RowAttributes determines the options that are added to the <TR> tag for each row in the body of the grid. These include the color of the rows, the alignment of data within each cell, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the header of the grid, use the TableAttributes (see page 25297) or HeadingAttributes (see page 25296) property instead.

See Also

TableAttributes (see page 25297)

HeadingAttributes (see page 25296)

181.1.28.5.3 TCustomAdapterGrid.TableAttributes Property

Determines the basic display characteristics of the table.

Pascal

```
property TableAttributes: TGridAttributes;
```

C++

```
__property TGridAttributes TableAttributes;
```

Description

Use TableAttributes to specify values for the most common options on the HTML <TABLE> tag. TableAttributes lets you specify the alignment of the table in its HTML document, its color and border size, the offset of data within each table cell, as well as any custom options or style rules.

Note: To influence the display attributes of only the header of the grid, or only the rows in its body, use the TableAttributes or HeadingAttributes (see page 25296) property instead.

See Also

RowAttributes (see page 25296)

HeadingAttributes (see page 25296)

181.1.29 TDesigntimeWarnings Class

Class Hierarchy

Pascal

```
TDesigntimeWarnings = class(TAbstractDesigntimeWarnings);
```

C++

```
class TDesigntimeWarnings : public TAbstractDesigntimeWarnings;
```

File

WebForm

Description

This is class WebForm.TDesigntimeWarnings.

181.1.29.1 TDesigntimeWarnings Members

The following tables list the members exposed by TDesigntimeWarnings.

Public Methods

	Name	Description
	AddObject (see page 25298)	Requests any design time warnings from a specified object.
	AddString (see page 25299)	Adds a warning string to the set of warnings that appear at design time.

	Create (see page 25298)	This is Create, a member of class TDesigntimeWarnings.
V	Destroy (see page 25298)	This is Destroy, a member of class TDesigntimeWarnings.

Public Properties

	Name	Description
R	Strings (see page 25299)	This is Strings, a member of class TDesigntimeWarnings.

181.1.29.2 TDesigntimeWarnings.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TDesigntimeWarnings();
```

Description

This is Create, a member of class TDesigntimeWarnings.

181.1.29.3 TDesigntimeWarnings.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDesigntimeWarnings();
```

Description

This is Destroy, a member of class TDesigntimeWarnings.

181.1.29.4 TDesigntimeWarnings Methods

The methods of the TDesigntimeWarnings class are listed here.

Public Methods

	Name	Description
V	AddObject (see page 25298)	Requests any design time warnings from a specified object.
V	AddString (see page 25299)	Adds a warning string to the set of warnings that appear at design time.

181.1.29.4.1 TDesigntimeWarnings.AddObject Method

Requests any design time warnings from a specified object.

Pascal

```
procedure AddObject(AObject: TObject); override;
```

C++

```
virtual __fastcall AddObject(TObject * AObject);
```

Description

AddObject collects the design time warnings from the specified object by calling its GetDesigntimeWarnings method (if it has one). If the object has already been added by a previous call to AddObject, AddObject detects this and does nothing.

AObject is the Web item from which to request design-time warnings.

See Also

GetDesigntimeWarnings (see page 25130)

AddString (see page 19632)

181.1.29.4.2 TDesigntimeWarnings.AddString Method

Adds a warning string to the set of warnings that appear at design time.

Pascal

```
procedure AddString(AWarning: string); override;
```

C++

```
virtual __fastcall AddString(AnsiString AWarning);
```

Description

AddString adds a string to the design-time warnings object. The adapter page producer displays all such warning strings in the HTML form that appears in the HTML view of the editor.

AWarning is a warning string that describes a problem that the developer can rectify (such as an invalid property value or missing component).

See Also

TAdapterPageProducer (see page 19802)

AddObject (see page 19632)

181.1.29.5 TDesigntimeWarnings Properties

The properties of the TDesigntimeWarnings class are listed here.

Public Properties

	Name	Description
R	Strings (see page 25299)	This is Strings, a member of class TDesigntimeWarnings.

181.1.29.5.1 TDesigntimeWarnings.Strings Property

Pascal

```
property Strings: TStrings;
```

C++

```
__property TStrings Strings;
```

Description

This is Strings, a member of class TDesigntimeWarnings.

181.1.30 TGeneratedFunctions Class

Class Hierarchy

WebForm.TGeneratedFunctions

Pascal

```
TGeneratedFunctions = class;
```

C++

```
class TGeneratedFunctions;
```

File

WebForm

Description

This is class WebForm.TGeneratedFunctions.

181.1.30.1 TGeneratedFunctions Members

The following tables list the members exposed by TGeneratedFunctions.

Public Methods

	Name	Description
	Add (see page 25301)	This is Add, a member of class TGeneratedFunctions.
	Create (see page 25300)	This is Create, a member of class TGeneratedFunctions.
V	Destroy (see page 25301)	This is Destroy, a member of class TGeneratedFunctions.
	Exists (see page 25301)	This is Exists, a member of class TGeneratedFunctions.

Public Properties

	Name	Description
R	ItemCount (see page 25302)	This is ItemCount, a member of class TGeneratedFunctions.
R	Names (see page 25302)	This is Names, a member of class TGeneratedFunctions.
R	Script (see page 25302)	This is Script, a member of class TGeneratedFunctions.

181.1.30.2 TGeneratedFunctions.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TGeneratedFunctions();
```

Description

This is Create, a member of class TGeneratedFunctions.

181.1.30.3 TGeneratedFunctions.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TGeneratedFunctions();
```

Description

This is Destroy, a member of class TGeneratedFunctions.

181.1.30.4 TGeneratedFunctions Methods

The methods of the TGeneratedFunctions class are listed here.

Public Methods

	Name	Description
	Add (see page 25301)	This is Add, a member of class TGeneratedFunctions.
	Exists (see page 25301)	This is Exists, a member of class TGeneratedFunctions.

181.1.30.4.1 TGeneratedFunctions.Add Method

Pascal

```
procedure Add(const AName: string; const AScript: string);
```

C++

```
__fastcall Add(const AnsiString AName, const AnsiString AScript);
```

Description

This is Add, a member of class TGeneratedFunctions.

181.1.30.4.2 TGeneratedFunctions.Exists Method

Pascal

```
function Exists(const AName: string): Boolean;
```

C++

```
__fastcall Boolean Exists(const AnsiString AName);
```

Description

This is Exists, a member of class TGeneratedFunctions.

181.1.30.5 TGeneratedFunctions Properties

The properties of the TGeneratedFunctions class are listed here.

Public Properties

	Name	Description
R	ItemCount (see page 25302)	This is ItemCount, a member of class TGeneratedFunctions.

R	Names (see page 25302)	This is Names, a member of class TGeneratedFunctions.
R	Script (see page 25302)	This is Script, a member of class TGeneratedFunctions.

181.1.30.5.1 TGeneratedFunctions.ItemCount Property

Pascal

```
property ItemCount: Integer;
```

C++

```
__property int ItemCount;
```

Description

This is ItemCount, a member of class TGeneratedFunctions.

181.1.30.5.2 TGeneratedFunctions.Names Property

Pascal

```
property Names [I: Integer]: string;
```

C++

```
__property AnsiString Names[int I];
```

Description

This is Names, a member of class TGeneratedFunctions.

181.1.30.5.3 TGeneratedFunctions.Script Property

Pascal

```
property Script [I: Integer]: string;
```

C++

```
__property AnsiString Script[int I];
```

Description

This is Script, a member of class TGeneratedFunctions.

181.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAdapterDisplay (see page 25303)	This is class WebForm.IAdapterDisplay.
	IGetAdapterFormLayout (see page 25303)	This is class WebForm.IGetAdapterFormLayout.
	IGetAdapterModeOfDisplay (see page 25304)	This is class WebForm.IGetAdapterModeOfDisplay.
	IGetAdapterOfDisplay (see page 25305)	This is class WebForm.IGetAdapterOfDisplay.
	IGetAdapterVariableName (see page 25306)	This is class WebForm.IGetAdapterVariableName.
	IGetControlRequirements (see page 25308)	This is class WebForm.IGetControlRequirements.
	IGetFormRequirements (see page 25309)	This is class WebForm.IGetFormRequirements.
	IGetVariablesContainerOfDisplay (see page 25310)	This is class WebForm.IGetVariablesContainerOfDisplay.

181.2.1 IAdapterDisplay Interface

Class Hierarchy

WebForm.IAdapterDisplay

Pascal

```
IAdapterDisplay = interface;
```

C++

```
__interface IAdapterDisplay;
```

File

WebForm

Description

This is class WebForm.IAdapterDisplay.

181.2.2 IGetAdapterFormLayout Interface

Class Hierarchy

WebForm.IGetAdapterFormLayout

Pascal

```
IGetAdapterFormLayout = interface;
```

C++

```
__interface IGetAdapterFormLayout;
```

File

WebForm

Description

This is class WebForm.IGetAdapterFormLayout.

181.2.2.1 IGetAdapterFormLayout Members

The following tables list the members exposed by IGetAdapterFormLayout.

Public Methods

	Name	Description
	GetAdapterFormLayout (see page 25304)	This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.

181.2.2.2 IGetAdapterFormLayout Methods

The methods of the IGetAdapterFormLayout class are listed here.

Public Methods

	Name	Description
	GetAdapterFormLayout (see page 25304)	This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.

181.2.2.2.1 IGetAdapterFormLayout.GetAdapterFormLayout Method

Pascal

```
function GetAdapterFormLayout: TAdapterFormLayout;
```

C++

```
__fastcall TAdapterFormLayout GetAdapterFormLayout();
```

Description

This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.

181.2.3 IGetAdapterModeOfDisplay Interface

Class Hierarchy

WebForm.IGetAdapterModeOfDisplay

Pascal

```
IGetAdapterModeOfDisplay = interface;
```

C++

```
__interface IGetAdapterModeOfDisplay;
```

File

WebForm

Description

This is class WebForm.IGetAdapterModeOfDisplay.

181.2.3.1 IGetAdapterModeOfDisplay Members

The following tables list the members exposed by IGetAdapterModeOfDisplay.

Public Methods

	Name	Description
	GetAdapterModeOfAdapter (see page 25305)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 25305)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.

181.2.3.2 IGetAdapterModeOfDisplay Methods

The methods of the IGetAdapterModeOfDisplay class are listed here.

Public Methods

	Name	Description
	GetAdapterModeOfAdapter (see page 25305)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 25305)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.

181.2.3.2.1 IGetAdapterModeOfDisplay.GetAdapterModeOfAdapter Method

Pascal

```
function GetAdapterModeOfAdapter: string;
```

C++

```
__fastcall AnsiString GetAdapterModeOfAdapter();
```

Description

This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.

181.2.3.2.2 IGetAdapterModeOfDisplay.GetAdapterModeOfDisplay Method

Pascal

```
function GetAdapterModeOfDisplay: string;
```

C++

```
__fastcall AnsiString GetAdapterModeOfDisplay();
```

Description

This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.

181.2.4 IGetAdapterOfDisplay Interface

Class Hierarchy

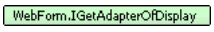

Pascal

```
IGetAdapterOfDisplay = interface;
```

C++

```
__interface IGetAdapterOfDisplay;
```

File

WebForm

Description

This is class WebForm.IGetAdapterOfDisplay.

181.2.4.1 IGetAdapterOfDisplay Members

The following tables list the members exposed by IGetAdapterOfDisplay.

Public Methods

	Name	Description
	GetAdapter (see page 25306)	This is GetAdapter, a member of class IGetAdapterOfDisplay.

181.2.4.2 IGetAdapterOfDisplay Methods

The methods of the IGetAdapterOfDisplay class are listed here.

Public Methods

	Name	Description
	GetAdapter (see page 25306)	This is GetAdapter, a member of class IGetAdapterOfDisplay.

181.2.4.2.1 IGetAdapterOfDisplay.GetAdapter Method

Pascal

```
function GetAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetAdapter();
```

Description

This is GetAdapter, a member of class IGetAdapterOfDisplay.

181.2.5 IGetAdapterVariableName Interface

Class Hierarchy

WebForm.IGetAdapterVariableName

Pascal

```
IGetAdapterVariableName = interface;
```

C++

```
__interface IGetAdapterVariableName;
```

File

WebForm

Description

This is class WebForm.IGetAdapterVariableName.

181.2.5.1 IGetAdapterVariableName Members

The following tables list the members exposed by IGetAdapterVariableName.

Public Methods

	Name	Description
	GetAdapterVariableName (see page 25307)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.

Public Properties

	Name	Description
R	AdapterVariableName (see page 25308)	This is AdapterVariableName, a member of class IGetAdapterVariableName.

181.2.5.2 IGetAdapterVariableName Methods

The methods of the IGetAdapterVariableName class are listed here.

Public Methods

	Name	Description
	GetAdapterVariableName (see page 25307)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.

181.2.5.2.1 IGetAdapterVariableName.GetAdapterVariableName Method

Pascal

```
function GetAdapterVariableName: string;
```

C++

```
__fastcall AnsiString GetAdapterVariableName();
```

Description

This is GetAdapterVariableName, a member of class IGetAdapterVariableName.

181.2.5.3 IGetAdapterVariableName Properties

The properties of the IGetAdapterVariableName class are listed here.

Public Properties

	Name	Description
R	AdapterVariableName (see page 25308)	This is AdapterVariableName, a member of class IGetAdapterVariableName.

181.2.5.3.1 IGetAdapterVariableName.AdapterVariableName Property

Pascal

```
property AdapterVariableName: string;
```

C++

```
__property AnsiString AdapterVariableName;
```

Description

This is AdapterVariableName, a member of class IGetAdapterVariableName.

181.2.6 IGetControlRequirements Interface

Class Hierarchy

WebForm.IGetControlRequirements

Pascal

```
IGetControlRequirements = interface;
```

C++

```
__interface IGetControlRequirements;
```

File

WebForm

Description

This is class WebForm.IGetControlRequirements.

181.2.6.1 IGetControlRequirements Members

The following tables list the members exposed by IGetControlRequirements.

Public Methods

	Name	Description
	GetControlRequirements (see page 25309)	This is GetControlRequirements, a member of class IGetControlRequirements.

181.2.6.2 IGetControlRequirements Methods

The methods of the IGetControlRequirements class are listed here.

Public Methods

	Name	Description
	GetControlRequirements (see page 25309)	This is GetControlRequirements, a member of class IGetControlRequirements.

181.2.6.2.1 IGetControlRequirements.GetControlRequirements Method

Pascal

```
procedure GetControlRequirements(ARequirements: TAbstractControlRequirements);
```

C++

```
__fastcall GetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is GetControlRequirements, a member of class IGetControlRequirements.

181.2.7 IGetFormRequirements Interface

Class Hierarchy

WebForm.IGetFormRequirements

Pascal

```
IGetFormRequirements = interface;
```

C++

```
__interface IGetFormRequirements;
```

File

WebForm

Description

This is class WebForm.IGetFormRequirements.

181.2.7.1 IGetFormRequirements Members

The following tables list the members exposed by IGetFormRequirements.

Public Methods

	Name	Description
	GetFormRequirements (see page 25310)	This is GetFormRequirements, a member of class IGetFormRequirements.

181.2.7.2 IGetFormRequirements Methods

The methods of the IGetFormRequirements class are listed here.

Public Methods

	Name	Description
	GetFormRequirements (see page 25310)	This is GetFormRequirements, a member of class IGetFormRequirements.

181.2.7.2.1 IGetFormRequirements.GetFormRequirements Method

Pascal

```
procedure GetFormRequirements(ARequirements: TAbstractFormRequirements);
```

C++

```
__fastcall GetFormRequirements(TAbstractFormRequirements ARequirements);
```

Description

This is GetFormRequirements, a member of class IGetFormRequirements.

181.2.8 IGetVariablesContainerOfDisplay Interface

Class Hierarchy

WebForm.IGetVariablesContainerOfDisplay

Pascal

```
IGetVariablesContainerOfDisplay = interface;
```

C++

```
__interface IGetVariablesContainerOfDisplay;
```

File

WebForm

Description

This is class WebForm.IGetVariablesContainerOfDisplay.

181.2.8.1 IGetVariablesContainerOfDisplay Members

The following tables list the members exposed by IGetVariablesContainerOfDisplay.

Public Methods

	Name	Description
	GetVariablesContainer (see page 25311)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.

181.2.8.2 IGetVariablesContainerOfDisplay Methods

The methods of the IGetVariablesContainerOfDisplay class are listed here.

Public Methods

	Name	Description
	GetVariablesContainer (see page 25311)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.

181.2.8.2.1 IGetVariablesContainerOfDisplay.GetVariablesContainer Method

Pascal

```
function GetVariablesContainer: TComponent;
```

C++

```
__fastcall TComponent * GetVariablesContainer();
```

Description

This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.

181.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CalcDisplayFieldHTMLElementType (see page 25312)	This is function WebForm.CalcDisplayFieldHTMLElementType.
	CalcDisplayFieldViewMode (see page 25312)	This is function WebForm.CalcDisplayFieldViewMode.
	FindVariablesContainer (see page 25313)	This is function WebForm.FindVariablesContainer.
	FindVariablesContainerInParent (see page 25313)	This is function WebForm.FindVariablesContainerInParent.
	GetAdapterModeOfAdapterInParent (see page 25313)	This is function WebForm.GetAdapterModeOfAdapterInParent.
	MakeAdapterHTMLName (see page 25314)	This is function WebForm.MakeAdapterHTMLName.
	MakeAdapterVariableName (see page 25314)	This is function WebForm.MakeAdapterVariableName.

181.3.1 WebForm.CalcDisplayFieldHTMLElementType Function

Pascal

```
function CalcDisplayFieldHTMLElementType(AField: TComponent; const AAdapterMode: string;
ADisplayType: TDisplayFieldHTMLElementType; var AType: TDisplayFieldHTMLElementType): Boolean;
```

C++

```
Boolean CalcDisplayFieldHTMLElementType(TComponent * AField, const AnsiString AAdapterMode,
TDisplayFieldHTMLElementType ADisplayType, TDisplayFieldHTMLElementType AType);
```

File

WebForm

Description

This is function WebForm.CalcDisplayFieldHTMLElementType.

181.3.2 WebForm.CalcDisplayFieldViewMode Function

Pascal

```
function CalcDisplayFieldViewMode(AField: TComponent; const AAdapterMode: string; AViewMode:
TDisplayFieldViewMode; var AMode: TDisplayFieldViewMode): Boolean;
```

C++

```
Boolean CalcDisplayFieldViewMode(TComponent * AField, const AnsiString AAdapterMode,
TDisplayFieldViewMode AViewMode, TDisplayFieldViewMode AMode);
```

File

WebForm

Description

This is function WebForm.CalcDisplayFieldViewMode.

181.3.3 WebForm.FindVariablesContainer Function

Pascal

```
function FindVariablesContainer(AComponent: TComponent): TComponent;
```

C++

```
TComponent * FindVariablesContainer(TComponent * AComponent);
```

File

WebForm

Description

This is function WebForm.FindVariablesContainer.

181.3.4 WebForm.FindVariablesContainerInParent Function

Pascal

```
function FindVariablesContainerInParent(AComponent: TComponent): TComponent;
```

C++

```
TComponent * FindVariablesContainerInParent(TComponent * AComponent);
```

File

WebForm

Description

This is function WebForm.FindVariablesContainerInParent.

181.3.5 WebForm.GetAdapterModeOfAdapterInParent Function

Pascal

```
function GetAdapterModeOfAdapterInParent(AComponent: TComponent): string;
```

C++

```
AnsiString GetAdapterModeOfAdapterInParent(TComponent * AComponent);
```

File

WebForm

Description

This is function WebForm.GetAdapterModeOfAdapterInParent.

181.3.6 WebForm.MakeAdapterHTMLName Function

Pascal

```
function MakeAdapterHTMLName(AVarName: string): string;
```

C++

```
AnsiString MakeAdapterHTMLName(AnsiString AVarName);
```

File

WebForm

Description

This is function WebForm.MakeAdapterHTMLName.

181.3.7 WebForm.MakeAdapterVariableName Function

Pascal

```
function MakeAdapterVariableName(AComponent: TComponent): string;
```

C++

```
AnsiString MakeAdapterVariableName(TComponent * AComponent);
```

File

WebForm

Description

This is function WebForm.MakeAdapterVariableName.

181.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TActionButtonHideOption (see page 25315)	TActionButtonHideOptions (see page 25321) indicates when a control to represent an adapter action should not be generated.
	TColumnHTMLElementType (see page 25316)	This is record WebForm.TColumnHTMLElementType.
	TCommandHTMLElementType (see page 25316)	TCommandHTMLElementType identifies an HTML control type that can be used execute an adapter action.
	TDisplayFieldHideOption (see page 25317)	TDisplayFieldHideOptions (see page 25321) indicates when a control to represent a dataset field should not be generated.
	TDisplayFieldHTMLElementType (see page 25318)	TDisplayFieldHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.
	TDisplayFieldViewMode (see page 25318)	TDisplayFieldViewMode indicates the mode in which an adapter field's value is displayed.
	TInputFieldHTMLElementType (see page 25319)	TInputFieldHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.
	TLayoutState (see page 25320)	This is record WebForm.TLayoutState.

181.4.1 WebForm.TActionButtonHideOption Enumeration

TActionButtonHideOptions (see page 25321) indicates when a control to represent an adapter action should not be generated.

Pascal

```
TActionButtonHideOption = (
  bhoHideOnNoExecuteAccess,
  bhoHideOnActionNotVisible,
  bhoHideOnDisabledAction,
  bhoHideAlways
);
```

C++

```
enum TActionButtonHideOption {
  bhoHideOnNoExecuteAccess,
  bhoHideOnActionNotVisible,
  bhoHideOnDisabledAction,
  bhoHideAlways
};
```

File

WebForm

Description

TActionButtonHideOptions (see page 25321) is a set of TActionButtonHideOption values. It can include zero or more of the following values:

Value	Meaning
bhoHideOnNoExecuteAccess	Suppress the HTML control when the user does not have rights to execute the associated action, as indicated by the action's HasExecuteAccess method. When this flag is set, server-side script can suppress the control by setting the action's CanExecute property to false.

bhoHideOnActionNotVisible	Suppress the HTML control when the action returns false from its GetVisible method. When this flag is set, server-side script can suppress the control by setting the action's Visible property to false. This flag is set by default.
bhoHideOnDisabledAction	Suppress the HTML control when the action returns false from its WebEnabled method. When this flag is set, server-side script can suppress the control by setting the action's Enabled property to false.
bhoHideAlways	Always suppress the HTML control.

181.4.2 WebForm.TColumnHTMLElementType Enumeration

Pascal

```
TColumnHTMLElementType = (
  coltDefault,
  coltText,
  coltImage,
  coltList
);
```

C++

```
enum TColumnHTMLElementType {
  coltDefault,
  coltText,
  coltImage,
  coltList
};
```

File

WebForm

Description

This is record WebForm.TColumnHTMLElementType.

181.4.3 WebForm.TCommandHTMLElementType Enumeration

TCommandHTMLElementType identifies an HTML control type that can be used execute an adapter action.

Pascal

```
TCommandHTMLElementType = (
  ctDefault,
  ctButton,
  ctImage,
  ctAnchor,
  ctEventImages
);
```

C++

```
enum TCommandHTMLElementType {
  ctDefault,
  ctButton,
```

```
  ctImage,
  ctAnchor,
  ctEventImages
};
```

File

WebForm

Description

Each element type indicates a different HTML element. TCommandHTMLElementType defines HTML elements that the user can click to execute a command.

The following table lists the possible values:

Value	Meaning
ctDefault	Generate the element type that the adapter action object suggests.
ctButton	Generate a push button with a text label.
ctImage	Generate an image that the user can click.
ctAnchor	Generate a hypertext link.
ctEventImages	Generate an element that displays a different image, depending on the users action. That is, the image changes dynamically depending on user events such as whether the mouse is over the control, whether the user is clicking the control, and so on.

See Also

DisplayType

181.4.4 WebForm.TDisplayFieldHideOption Enumeration

TDisplayFieldHideOptions (see page 25321) indicates when a control to represent a dataset field should not be generated.

Pascal

```
TDisplayFieldHideOption = (
  hoHideOnNoInputAccess,
  hoHideOnNoDisplayAccess,
  hoHideOnFieldNotVisible,
  hoHideAlways
);
```

C++

```
enum TDisplayFieldHideOption {
  hoHideOnNoInputAccess,
  hoHideOnNoDisplayAccess,
  hoHideOnFieldNotVisible,
  hoHideAlways
};
```

File

WebForm

Description

TDisplayFieldHideOptions (see page 25321) is a set of TDisplayFieldHideOption values. It can include zero or more of the following values:

Value	Meaning
hoHideOnNoInputAccess	Do not generate a control if the user does not have permission to modify this field.
hoHideOnNoDisplayAccess	Do not generate a control if the user does not have permission to view this field.
hoHideOnFieldNotVisable	Do not generate a control if the field is marked as hidden.
hoHideAlways	Never generate a control for the field.

181.4.5 WebForm.TDisplayFieldHTMLElementType Enumeration

TDisplayFieldHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.

Pascal

```
TDisplayFieldHTMLElementType = (
  dftDefault,
  dftText,
  dftImage,
  dftList
);
```

C++

```
enum TDisplayFieldHTMLElementType {
  dftDefault,
  dftText,
  dftImage,
  dftList
};
```

File

WebForm

Description

Each element type indicates a different HTML element. TDisplayFieldHTMLElementType defines read-only HTML elements.

The following table lists the possible values:

Value	Meaning
dftDefault	Use the default display type of the component that generates the HTML control
dftText	Display as a text string.
dftImage	Display as an image.
dftList	Display as a list of values.

181.4.6 WebForm.TDisplayFieldViewMode Enumeration

TDisplayFieldViewMode indicates the mode in which an adapter field's value is displayed.

Pascal

```
TDisplayFieldViewMode = (
  vmDefault,
  vmInput,
  vmDisplay,
  vmToggleOnAccess
);
```

C++

```
enum TDisplayFieldViewMode {
  vmDefault,
  vmInput,
  vmDisplay,
  vmToggleOnAccess
};
```

File

WebForm

Description

The following table lists the possible values for the TDisplayFieldViewMode type and the meaning corresponding to each:

Value	Meaning
vmDefault	Use the default mode associated with the adapter.
vmInput	Generate an input control for the field.
vmDisplay	Generate a read-only control for the field.
vmToggleOnAccess	Generate a control that toggles between read-only and input mode when the user tries to access it.

181.4.7 WebForm.TInputFieldHTMLElementType Enumeration

TInputFieldHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.

Pascal

```
TInputFieldHTMLElementType = (
  iftDefault,
  iftTextInput,
  iftPasswordInput,
  iftSelect,
  iftSelectMultiple,
  iftRadio,
  iftCheckBox,
  iftTextArea,
  iftFile
);
```

C++

```
enum TInputFieldHTMLElementType {
  iftDefault,
  iftTextInput,
  iftPasswordInput,
  iftSelect,
  iftSelectMultiple,
```

```
    iftRadio,
    iftCheckBox,
    iftTextArea,
    iftFile
  };
```

File

WebForm

Description

TInputFieldHTMLElementType defines the set of possible HTML controls that can be used to input an adapter field value. The following table lists the possible values for the TInputFieldHTMLElementType type and the meaning corresponding to each flag:

Flag	Input field type
iftCheckBox	The input field is a check box.
iftDefault	The input field is the default type.
iftFile	The input field is a file.
iftPasswordInput	The input field is a password prompt.
iftRadio	The input field is a radio button.
iftSelect	The input field is selectable. iftSelect is only used with iftCheckBox and iftRadio.
iftSelectMultiple	More than one input field may be selected at a time. iftSelectMultiple is only used with iftCheckBox and iftRadio.
iftTextArea	The field is a text area.
iftTextInput	The input field is a text input box.

181.4.8 WebForm.TLayoutState Enumeration

Pascal

```
  TLayoutState = (
    lsUnknown,
    lsFields,
    lsButtons
  );
```

C++

```
  enum TLayoutState {
    lsUnknown,
    lsFields,
    lsButtons
  };
```

File

WebForm

Description

This is record WebForm.TLayoutState.

181.5 Types

The following table lists types in this documentation.

Types

Name	Description
TActionButtonHideOptions (see page 25321)	This is type WebForm.TActionButtonHideOptions.
TAdapterDataDisplayClass (see page 25321)	This is type WebForm.TAdapterDataDisplayClass.
TDisplayFieldHideOptions (see page 25321)	This is type WebForm.TDisplayFieldHideOptions.

181.5.1 WebForm.TActionButtonHideOptions Type

Pascal

```
TActionButtonHideOptions = set of TActionButtonHideOption;
```

C++

```
TActionButtonHideOption TActionButtonHideOptions;
```

File

WebForm

Description

This is type WebForm.TActionButtonHideOptions.

181.5.2 WebForm.TAdapterDataDisplayClass Type

Pascal

```
TAdapterDataDisplayClass = class of TAdapterDataDisplay;
```

C++

```
TAdapterDataDisplay TAdapterDataDisplayClass;
```

File

WebForm

Description

This is type WebForm.TAdapterDataDisplayClass.

181.5.3 WebForm.TDisplayFieldHideOptions Type

Pascal

```
TDisplayFieldHideOptions = set of TDisplayFieldHideOption;
```

C++

```
TDisplayFieldHideOption TDisplayFieldHideOptions;
```

File

WebForm

Description

This is type WebForm.TDisplayFieldHideOptions.

181.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
sDisplayText (see page 25323)	This is constant WebForm.sDisplayText.
sFieldText (see page 25323)	This is constant WebForm.sFieldText.
sFormGet (see page 25324)	This is constant WebForm.sFormGet.
sFormPost (see page 25324)	This is constant WebForm.sFormPost.
sMultipartForm (see page 25324)	This is constant WebForm.sMultipartForm.

181.6.1 WebForm.sDisplayText Constant

Pascal

```
sDisplayText = '<%%= %0:s.DisplayText %%>';
```

C++

```
sDisplayText = '<%%= %0:s.DisplayText %%>';
```

File

WebForm

Description

This is constant WebForm.sDisplayText.

181.6.2 WebForm.sFieldText Constant

Pascal

```
sFieldText = '<%%= %0:s.EditText %%>';
```

C++

```
sFieldText = '<%%= %0:s.EditText %%>';
```

File

WebForm

Description

This is constant WebForm.sFieldText.

181.6.3 WebForm.sFormGet Constant

Pascal

```
sFormGet = 'get';
```

C++

```
sFormGet = 'get';
```

File

WebForm

Description

This is constant WebForm.sFormGet.

181.6.4 WebForm.sFormPost Constant

Pascal

```
sFormPost = 'post';
```

C++

```
sFormPost = 'post';
```

File

WebForm

Description

This is constant WebForm.sFormPost.

181.6.5 WebForm.sMultipartForm Constant

Pascal

```
sMultipartForm = 'multipart/form-data';
```

C++

```
sMultipartForm = 'multipart/form-data';
```

File

WebForm

Description

This is constant WebForm.sMultipartForm.

182 WebModu Namespace

This is namespace WebModu.

Topics

Name	Description
Classes (see page 25327)	The following table lists classes in this documentation.
Functions (see page 25426)	The following table lists functions in this documentation.
Types (see page 25428)	The following table lists types in this documentation.

Classes

	Name	Description
	EWebModuleException (see page 25327)	EWebModuleException is the exception that is raised when a WebSnap application tries to use a Web module to perform a task it does not support.
	TAdapterActionName (see page 25328)	TAdapterActionName describes an adapter action by name.
	TCustomWebAppDataModule (see page 25332)	TCustomWebAppDataModule is the base class from which to derive new WebSnap application data modules.
	TCustomWebAppPageModule (see page 25339)	TCustomWebAppPageModule is the base class for WebSnap application data modules that can produce Web pages.
	TCustomWebDataModule (see page 25348)	TCustomWebDataModule is the base class for all Web modules in a WebSnap application.
	TCustomWebPageModule (see page 25357)	TCustomWebPageModule is the base class for WebSnap modules that generate a Web page.
	TSiteModuleHelper (see page 25368)	TSiteModuleHelper manages common tasks for Web modules in a WebSnap application.
	TSitePageModuleHelper (see page 25377)	TSitePageModuleHelper manages common tasks for Web page modules in a WebSnap application.
	TWebAppDataModule (see page 25388)	TWebAppDataModule is used as the WebSnap application module when there is no need to generate a page.
	TWebAppPageModule (see page 25394)	TWebAppPageModule is used as the WebSnap application module when the application module generates a page.
	TWebAppPageModuleHelper (see page 25403)	This is class WebModu.TWebAppPageModuleHelper.
	TWebDataModule (see page 25405)	TWebDataModule is a container for components in a WebSnap application.
	TWebNotifyContainer (see page 25410)	This is class WebModu.TWebNotifyContainer.
	TWebPageModule (see page 25414)	TWebPageModule is a WebSnap module that generates a page.
	TWebPageModuleHelper (see page 25422)	This is class WebModu.TWebPageModuleHelper.

Functions

	Name	Description
	FindNamedModuleVariable (see page 25426)	This is function WebModu.FindNamedModuleVariable.
	GetModuleVariable (see page 25426)	This is function WebModu.GetModuleVariable.
	GetModuleVariableCount (see page 25426)	This is function WebModu.GetModuleVariableCount.

Types

Name	Description
TGetParamsEvent (see page 25428)	This is type WebModu.TGetParamsEvent.

182.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EWebModuleException (see page 25327)	EWebModuleException is the exception that is raised when a WebSnap application tries to use a Web module to perform a task it does not support.
	TAdapterActionName (see page 25328)	TAdapterActionName describes an adapter action by name.
	TCustomWebAppDataModule (see page 25332)	TCustomWebAppDataModule is the base class from which to derive new WebSnap application data modules.
	TCustomWebAppPageModule (see page 25339)	TCustomWebAppPageModule is the base class for WebSnap application data modules that can produce Web pages.
	TCustomWebDataModule (see page 25348)	TCustomWebDataModule is the base class for all Web modules in a WebSnap application.
	TCustomWebPageModule (see page 25357)	TCustomWebPageModule is the base class for WebSnap modules that generate a Web page.
	TSiteModuleHelper (see page 25368)	TSiteModuleHelper manages common tasks for Web modules in a WebSnap application.
	TSitePageModuleHelper (see page 25377)	TSitePageModuleHelper manages common tasks for Web page modules in a WebSnap application.
	TWebAppDataModule (see page 25388)	TWebAppDataModule is used as the WebSnap application module when there is no need to generate a page.
	TWebAppPageModule (see page 25394)	TWebAppPageModule is used as the WebSnap application module when the application module generates a page.
	TWebAppPageModuleHelper (see page 25403)	This is class WebModu.TWebAppPageModuleHelper.
	TWebDataModule (see page 25405)	TWebDataModule is a container for components in a WebSnap application.
	TWebNotifyContainer (see page 25410)	This is class WebModu.TWebNotifyContainer.
	TWebPageModule (see page 25414)	TWebPageModule is a WebSnap module that generates a page.
	TWebPageModuleHelper (see page 25422)	This is class WebModu.TWebPageModuleHelper.

182.1.1 EWebModuleException Class

EWebModuleException is the exception that is raised when a WebSnap application tries to use a Web module to perform a task it does not support.

Class Hierarchy

Pascal

```
EWebModuleException = class(EWebBrokerException);
```

C++

```
class EWebModuleException : public EWebBrokerException;
```

File

WebModu

Description

EWebModuleException is the exception that is raised when a server-side script in a WebSnap application tries to do direct a Web module to perform a task it does not know how to handle. For example, Web modules raise this exception when a script

tries to access a variable in a Web module that does not support the use of variables.

See Also

TWebDataModule (see page 25405)

TWebAppDataModule (see page 25388)

TWebPageModule (see page 25414)

TWebAppPageModule (see page 25394)

182.1.2 TAdapterActionName Class

TAdapterActionName describes an adapter action by name.

Class Hierarchy

Pascal

```
TAdapterActionName = class(TPersistent);
```

C++

```
class TAdapterActionName : public TPersistent;
```

File

WebModu

Description

TAdapterAction represents a TAdapterAction object. It is used when the action is referenced by name rather than directly as a component. TAdapterAction publishes properties that specify the name of the action and the parent adapter component to which it belongs. The read-only Action (see page 25331) property provides access to the action component with the specified name that belongs to the specified adapter.

See Also

TAdapter (see page 23713)

TAdapterAction (see page 23726)

182.1.2.1 TAdapterActionName Members

The following tables list the members exposed by TAdapterActionName.

Protected Methods

	Name	Description
	AssignTo (see page 25330)	Copies the properties of an object to a destination object.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetOwner (see page 2932)	Returns the owner of an object.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Action (see page 25331)	Indicates the action that this object specifies.

Published Methods

	Name	Description
	Create (see page 25329)	Creates and initializes an instance of TAdapterActionName (see page 25328).

Published Properties

	Name	Description
	ActionName (see page 25331)	Specifies the name of the action this object represents.
	Adapter (see page 25332)	Specifies the adapter to which the action belongs.

182.1.2.2 TAdapterActionName.Create Constructor

Creates and initializes an instance of TAdapterActionName (see page 25328).

Pascal

```
constructor Create(AModule: TComponent);
```

C++

```
__fastcall TAdapterActionName(TComponent * AModule);
```

Description

Call Create to instantiate TAdapterActionName (see page 25328).

AModule is the Web module instance that should be notified if the adapter specified by the Adapter (see page 25332) property is deleted. The TAdapterActionName (see page 25328) constructor calls this Adapter (see page 25332)'s FreeNotification method so that the adapter will notify the Web module if it (and its actions) are freed.

See Also

FreeNotification (see page 2775)

Adapter (see page 25332)

182.1.2.3 TAdapterActionName Methods

The methods of the TAdapterActionName class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 25330)	Copies the properties of an object to a destination object.

182.1.2.3.1 TAdapterActionName.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign (see page 2930)

EConvertError (see page 22724)

182.1.2.4 TAdapterActionName Properties

The properties of the TAdapterActionName class are listed here.

Public Properties

	Name	Description
R	Action (see page 25331)	Indicates the action that this object specifies.

Published Properties

	Name	Description
	ActionName (see page 25331)	Specifies the name of the action this object represents.
	Adapter (see page 25332)	Specifies the adapter to which the action belongs.

182.1.2.4.1 TAdapterActionName.Action Property

Indicates the action that this object specifies.

Pascal

```
property Action: TComponent;
```

C++

```
__property TComponent * Action;
```

Description

Read Action to access the adapter action component specified by the Adapter (see page 25332) and ActionName (see page 25331) properties. That is, Action is the action with the name specified by ActionName (see page 25331) that belongs to the adapter specified by Adapter (see page 25332).

See Also

ActionName (see page 25331)

Adapter (see page 25332)

TAdapterAction (see page 23726)

182.1.2.4.2 TAdapterActionName.ActionName Property

Specifies the name of the action this object represents.

Pascal

```
property ActionName: string;
```

C++

```
__property AnsiString ActionName;
```

Description

Set ActionName to the name of the desired action. This is the name of one of the actions that belongs to the adapter that the Adapter (see page 25332) property specifies.

See Also

Action (see page 25331)

Adapter (see page 25332)

Actions (see page 24181)

182.1.2.4.3 TAdapterActionName.Adapter Property

Specifies the adapter to which the action belongs.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

Set Adapter to the adapter component to which the named action belongs. The TAdapterActionName (see page 25328) object represents the action that this adapter returns when you call its FindAction method, passing in the value of the ActionName (see page 25331) property.

See Also

Action (see page 25331)

ActionName (see page 25331)

Actions (see page 24181)

FindAction (see page 24153)

182.1.3 TCustomWebAppDataModule Class

TCustomWebAppDataModule is the base class from which to derive new WebSnap application data modules.

Class Hierarchy

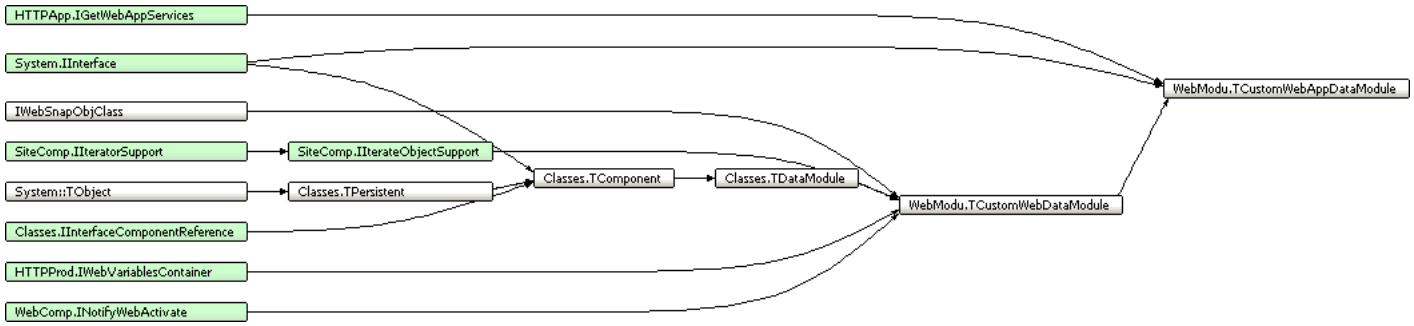

Pascal

```
TCustomWebAppDataModule = class(TCustomWebDataModule, IGetWebAppServices, IInterface);
```

C++

```
class TCustomWebAppDataModule : public TCustomWebDataModule, public IGetWebAppServices, public
IInterface;
```

File

WebModu

Description

Use TCustomWebAppDataModule as a base class for defining data modules that function as the main Web module in a WebSnap application. As the application module, TCustomWebAppDataModule provides access to the IWebAppServices interface that allows the WebSnap application to create a Web context for incoming requests and to forward requests to a dispatcher for handling.

If you want an application module that also has an associated HTML page, then derive from TCustomWebAppPageModule (see page 25339) instead.

See Also

TWebAppDataModule (see page 25388)

182.1.3.1 TCustomWebAppDataModule Members

The following tables list the members exposed by TCustomWebAppDataModule.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
V	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetSiteModuleHelper (see page 25353)	Returns the value of the SiteModuleHelper (see page 25356) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebAppServices (see page 25337)	Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.
V	GetWebSnapObjClass (see page 25354)	Returns the class type for this adapter field.
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.

	Notification (see page 25338)	Responds to notifications that components are about to be inserted or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 25338)	Returns a reference to a specified interface if the module supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 25356)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 25356) and OnDeactivate (see page 25357) events.

Public Events

	Name	Description
	OnActivate (see page 25356)	Occurs when the Web module is activated.
	OnDeactivate (see page 25357)	Occurs when the Web module is deactivated.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 2829)	Generates an OnCreate (see page 2833) event.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 2830)	Generates an OnDestroy (see page 2834) event.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25336)	Creates a new TCustomWebAppDataModule (see page 25332) instance.
	CreateNew (see page 25337)	Creates a data module without using the associated form file to initialize it.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.

	Destroy (see page 25337)	Frees a Web application data module instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentObject (see page 19719)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	AppServices (see page 25339)	Provides read and write access to the application services stored in this TCustomWebAppDataModule (see page 25332) object.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Request (see page 25355)	Provides access to the current request object.
R	Response (see page 25355)	Provides access to the current response object.
R	Session (see page 25355)	Provides access to the current Web session.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.
	Tag (see page 2808)	Stores an integer value as part of a component.

182.1.3.2 TCustomWebAppDataModule.Create Constructor

Creates a new TCustomWebAppDataModule (see page 25332) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomWebAppDataModule(TComponent * AOwner);
```

Description

Use this constructor to create a new TCustomWebAppDataModule (see page 25332). WebSnap modules are registered with a factory that is responsible for creating modules as needed. The module's source unit defines a function that returns an instance of the module.

AOwner is the component that becomes the value of the Owner property.

182.1.3.3 TCustomWebAppDataModule.CreateNew Constructor

Creates a data module without using the associated form file to initialize it.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); override;
```

C++

```
virtual __fastcall TCustomWebAppDataModule(TComponent * AOwner, int Dummy = 0);
```

Description

Use CreateNew instead of Create (see page 25336) to create a data module without using the associated form file to initialize it.

AOwner is the component that becomes the value of the Owner property.

Dummy is not used.

182.1.3.4 TCustomWebAppDataModule.Destroy Destructor

Frees a Web application data module instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWebAppDataModule();
```

Description

```
destructor Destroy; override;
```

Destroy is the destructor for the object instance. It clears the AppServices (see page 25339) property and calls the inherited destructor. The AppServices (see page 25339) interface is cleared to prevent design time problems that could otherwise occur because the object that implements AppServices (see page 25339) may also be owned by a Web page module.

182.1.3.5 TCustomWebAppDataModule Methods

The methods of the TCustomWebAppDataModule class are listed here.

Protected Methods

	Name	Description
	GetWebAppServices (see page 25337)	Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.
V	Notification (see page 25338)	Responds to notifications that components are about to be inserted or removed.
V	QueryInterface (see page 25338)	Returns a reference to a specified interface if the module supports that interface.

182.1.3.5.1 TCustomWebAppDataModule.GetWebAppServices Method

Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.

Pascal

```
function GetWebAppServices: IWebAppServices;
```

C++

```
__fastcall IWebAppServices GetWebAppServices();
```

Description

GetWebAppServices returns the IWebAppServices interface that is stored in this TCustomWebDataModule (see page 25348). This is the same as the value of the AppServices (see page 25339) property.

See Also

AppServices (see page 25339)

182.1.3.5.2 TCustomWebAppDataModule.Notification Method

Responds to notifications that components are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are added to or removed from the application. TCustomWebAppDataModule (see page 25332) uses this method to check whether the component that implements AppServices (see page 25339) is about to be removed, and if so, sets the value of the AppServices (see page 25339) property to nil (Delphi) or NULL (C++).

See Also

AppServices (see page 25339)

182.1.3.5.3 TCustomWebAppDataModule.QueryInterface Method

Returns a reference to a specified interface if the module supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override;
```

C++

```
virtual __fastcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the Web application module supports the interface specified by IID. If so, QueryInterface

increments the reference count.

sets the Obj parameter so that it points to an instance of the specified interface.

returns 0 to indicate success.

If the Web application module does not support the interface, QueryInterface returns a nonzero error code such as E_NoInterface.

See Also

_AddRef (see page 22397)

182.1.3.6 TCustomWebAppDataModule Properties

The properties of the TCustomWebAppDataModule class are listed here.

Public Properties

	Name	Description
	AppServices (see page 25339)	Provides read and write access to the application services stored in this TCustomWebAppDataModule (see page 25332) object.

182.1.3.6.1 TCustomWebAppDataModule.AppServices Property

Provides read and write access to the application services stored in this TCustomWebAppDataModule (see page 25332) object.

Pascal

```
property AppServices: IWebAppServices;
```

C++

```
__property IWebAppServices AppServices;
```

Description

AppServices provides access to the application services implemented by a component in this Web application module. The application services component sets up the Web context for each incoming request message, passes incoming request objects on to a dispatcher for handling, and provides the final backstop for exceptions that arise while processing request messages.

Typically, this property references the interface of a TWebAppComponents object.

See Also

TWebAppComponents (see page 24958)

WebContext (see page 24687)

182.1.4 TCustomWebAppPageModule Class

TCustomWebAppPageModule is the base class for WebSnap application data modules that can produce Web pages.

Class Hierarchy

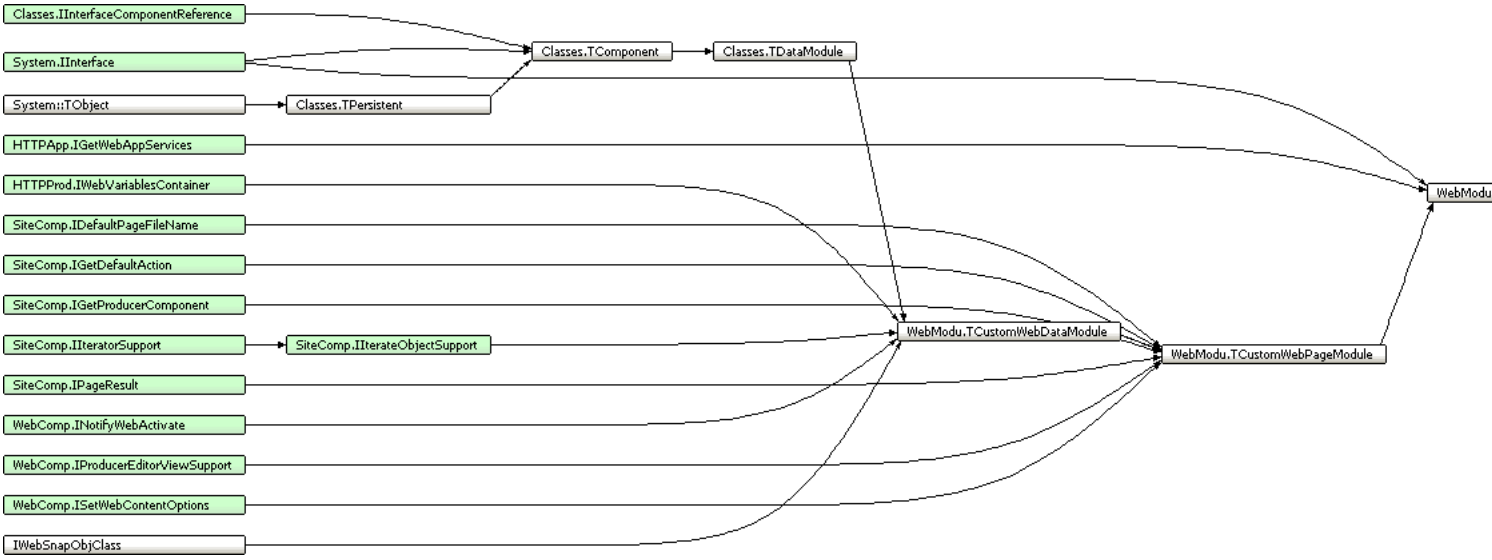

Pascal

```
TCustomWebAppPageModule = class(TCustomWebPageModule, IGetWebAppServices, IInterface);
```

C++

```
class TCustomWebAppPageModule : public TCustomWebPageModule, public IGetWebAppServices, public
IInterface;
```

File

WebModu

Description

Use TCustomWebAppPageModule as a base class for defining data modules that function as the main Web module in a WebSnap application and that have an associated Web page. As the application module, TCustomWebAppPageModule provides access to the IWebAppServices interface that allows the WebSnap application to create a Web context for incoming requests and to forward requests to a dispatcher for handling. As a Web page module, TCustomWebAppPageModule generates an associated Web page. This Web page is returned by default when incoming request messages do not specify what Web page they want.

See Also

IWebAppServices (see page 15227)

TWebAppPageModule (see page 25394)

WebContext

182.1.4.1 TCustomWebAppPageModule Members

The following tables list the members exposed by TCustomWebAppPageModule.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
V	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetSiteModuleHelper (see page 25353)	Returns the value of the SiteModuleHelper (see page 25356) property.
V	GetSiteModuleHelper (see page 25363)	Returns the value of the SiteModuleHelper (see page 25356) property.
V	GetSitePageModuleHelper (see page 25364)	Returns the value of the SitePageModuleHelper (see page 25366) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebAppServices (see page 25346)	Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.
V	GetWebSnapObjClass (see page 25354)	Returns the class type for this adapter field.
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 25346)	Responds to notifications that components are about to be inserted or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 25346)	Returns a reference to a specified interface if the module supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppServices (see page 25347)	Sets the value of the AppServices (see page 25347) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	SiteModuleHelper (see page 25356)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 25356) and OnDeactivate (see page 25357) events.
R	SitePageModuleHelper (see page 25366)	References the helper object that assists in assembling the module's Web page.

Public Events

	Name	Description
	OnActivate (see page 25356)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 25366)	Occurs after the page module generates its Web page.
	OnAfterRedirectToPage (see page 25367)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 25367)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnBeforeRedirectToPage (see page 25368)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.
	OnDeactivate (see page 25357)	Occurs when the Web module is deactivated.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 2829)	Generates an OnCreate (see page 2833) event.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 2830)	Generates an OnDestroy (see page 2834) event.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 25344)	Creates and initializes an instance of TCustomWebAppPageModule (see page 25339).
V	CreateNew (see page 25345)	Creates and initializes an instance of TCustomWebAppPageModule (see page 25339) without using an associated form file.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25345)	Frees an instance of TCustomWebAppPageModule (see page 25339).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DispatchPage (see page 19732)	This is DispatchPage, a member of class IPageResult.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentObject (see page 19719)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultAction (see page 19696)	This is GetDefaultAction, a member of class IGetDefaultAction.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProducerComponent (see page 19702)	This is GetProducerComponent, a member of class IGetProducerComponent.
	GetTemplateFileType (see page 24780)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetXMLData (see page 24780)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 24780)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 24780)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasScriptView (see page 24780)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 24781)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 24781)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	IncludePage (see page 19733)	This is IncludePage, a member of class IPageResult.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RedirectToPage (see page 19733)	This is RedirectToPage, a member of class IPageResult.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetDefaultPageFileName (see page 19683)	This is SetDefaultPageFileName, a member of class IDefaultPageFileName.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetWebContentOptions (see page 24782)	This is SetWebContentOptions, a member of class ISetWebContentOptions.

	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	AppServices (see page 25347)	Provides read and write access to the application services stored in this TCustomWebAppPageModule (see page 25339) object.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	DefaultAction (see page 19696)	This is DefaultAction, a member of class IGetDefaultAction.
	DefaultAction (see page 25365)	Represents the adapter action associated with the Web page module.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	PageProducer (see page 25365)	Indicates the interface of the page producer that generates the content of the Web page.
R	Request (see page 25355)	Provides access to the current request object.
R	Response (see page 25355)	Provides access to the current response object.
R	Session (see page 25355)	Provides access to the current Web session.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.
	Tag (see page 2808)	Stores an integer value as part of a component.

182.1.4.2 TCustomWebAppPageModule.Create Constructor

Creates and initializes an instance of TCustomWebAppPageModule (see page 25339).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomWebAppPageModule(TComponent * AOwner);
```

Description

Do not call the TCustomWebAppPageModule (see page 25339) constructor. Instead, use a TWebAppPageModuleFactory object to create Web module instances.

AOwner is the component that becomes the value of the Owner property.

See Also

TWebAppPageModuleFactory (see page 25012)

Owner (see page 2807)

182.1.4.3 TCustomWebAppPageModule.CreateNew Constructor

Creates and initializes an instance of TCustomWebAppPageModule (see page 25339) without using an associated form file.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); override;
```

C++

```
virtual __fastcall TCustomWebAppPageModule(TComponent * AOwner, int Dummy = 0);
```

Description

Use CreateNew instead of Create (see page 25344) to create a data module without using the associated form file to initialize it.

AOwner is the component that becomes the value of the Owner property.

Dummy is not used.

See Also

TWebAppPageModuleFactory (see page 25012)

Owner (see page 2807)

182.1.4.4 TCustomWebAppPageModule.Destroy Destructor

Frees an instance of TCustomWebAppPageModule (see page 25339).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWebAppPageModule();
```

Description

Destroy is the destructor for the object instance. It clears the AppServices (see page 25347) property and calls the inherited destructor. The AppServices (see page 25347) interface is cleared to prevent design time problems that could otherwise occur because the object that implements AppServices (see page 25347) may also be owned by a Web page module.

See Also

AppServices (see page 25347)

Free (see page 22300)

182.1.4.5 TCustomWebAppPageModule Methods

The methods of the TCustomWebAppPageModule class are listed here.

Protected Methods

	Name	Description
	GetWebAppServices (see page 25346)	Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.
V	Notification (see page 25346)	Responds to notifications that components are about to be inserted or removed.
V	QueryInterface (see page 25346)	Returns a reference to a specified interface if the module supports that interface.
	SetAppServices (see page 25347)	Sets the value of the AppServices (see page 25347) property.

182.1.4.5.1 TCustomWebAppPageModule.GetWebAppServices Method

Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.

Pascal

```
function GetWebAppServices: IWebAppServices;
```

C++

```
__fastcall IWebAppServices GetWebAppServices();
```

Description

GetWebAppServices returns the IWebAppServices interface that is stored in this Web application module. This is the same as the value of the AppServices (see page 25347) property.

See Also

AppServices (see page 25347)

182.1.4.5.2 TCustomWebAppPageModule.Notification Method

Responds to notifications that components are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are added to or removed from the application. TCustomWebAppPageModule (see page 25339) uses this method to check whether the component that implements AppServices (see page 25347) is about to be removed, and if so, sets the value of the AppServices (see page 25347) property to nil (Delphi) or NULL (C++).

See Also

AppServices (see page 25347)

182.1.4.5.3 TCustomWebAppPageModule.QueryInterface Method

Returns a reference to a specified interface if the module supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override;
```

C++

```
virtual __fastcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the Web application module supports the interface specified by IID. If so, QueryInterface

increments the reference count.

sets the Obj parameter so that it points to an instance of the specified interface.

returns 0 to indicate success.

If the Web application module does not support the interface, QueryInterface returns a nonzero error code such as E_NoInterface.

See Also

_AddRef (see page 22397)

182.1.4.5.4 TCustomWebAppPageModule.SetAppServices Method

Sets the value of the AppServices (see page 25347) property.

Pascal

```
procedure SetAppServices(const Value: IWebAppServices);
```

C++

```
__fastcall SetAppServices(const IWebAppServices Value);
```

Description

SetAppServices is the protected write implemenation of the AppServices (see page 25347) property.

See Also

AppServices (see page 25347)

182.1.4.6 TCustomWebAppPageModule Properties

The properties of the TCustomWebAppPageModule class are listed here.

Public Properties

	Name	Description
	AppServices (see page 25347)	Provides read and write access to the application services stored in this TCustomWebAppPageModule (see page 25339) object.

182.1.4.6.1 TCustomWebAppPageModule.AppServices Property

Provides read and write access to the application services stored in this TCustomWebAppPageModule (see page 25339) object.

Pascal

```
property AppServices: IWebAppServices;
```

C++

```
__property IWebAppServices AppServices;
```

Description

AppServices provides access to the application services implemented by a component in this Web application module. The application services component sets up the Web context for each incoming request message, passes incoming request objects

on to a dispatcher for handling, and provides the final backstop for exceptions that arise while processing request messages.

Typically, this property references the interface of a TWebAppComponents object.

See Also

TWebAppComponents (see page 24958)

WebContext (see page 24687)

182.1.5 TCustomWebDataModule Class

TCustomWebDataModule is the base class for all Web modules in a WebSnap application.

Class Hierarchy

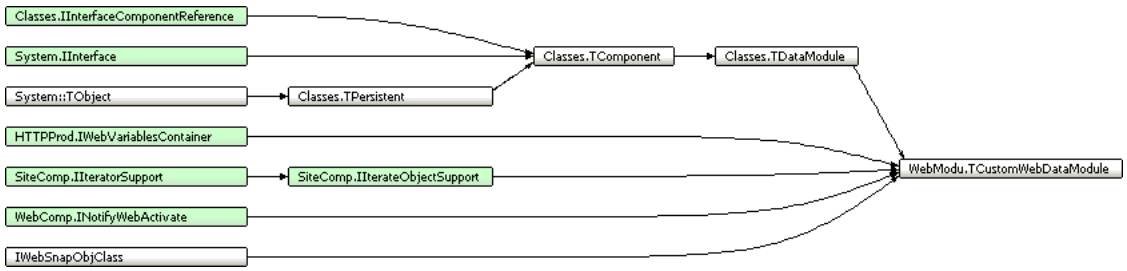

Pascal

```
TCustomWebDataModule = class(TDataModule, IWebVariablesContainer, INotifyWebActivate,
IIterateObjectSupport, IWebSnapObjClass);
```

C++

```
class TCustomWebDataModule : public TDataModule, public IWebVariablesContainer, public
INotifyWebActivate, public IIterateObjectSupport, public IWebSnapObjClass;
```

File

WebModu

Description

TCustomWebDataModule is the common base class for Web modules that can be used in a WebSnap application. This class introduces properties for the current HTTP request and response objects, a property to hold the current session object, and events that let you respond when the Web module is activated and deactivated. It also introduces support for interacting with server-side script and managing the WebSnap components that are placed into the Web module.

Do not create instances of TCustomWebDataModule. It is intended only as a base class. Instead, use the Web module wizard or the WebSnap application wizard to add a TCustomWebDataModule descendant to your WebSnap application. Descendant classes include TWebAppDataModule (see page 25388), TWebAppPageModule (see page 25394), TWebPageModule (see page 25414), and TWebDataModule (see page 25405).

See Also

TWebAppDataModule (see page 25388)

TWebAppPageModule (see page 25394)

TWebPageModule (see page 25414)

TWebDataModule (see page 25405)

182.1.5.1 TCustomWebDataModule Members

The following tables list the members exposed by TCustomWebDataModule.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
V	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetSiteModuleHelper (see page 25353)	Returns the value of the SiteModuleHelper (see page 25356) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	GetWebSnapObjClass (see page 25354)	Returns the class type for this adapter field.
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 25354)	Responds to notifications that components are about to be inserted or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 25356)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 25356) and OnDeactivate (see page 25357) events.

Public Events

	Name	Description
	OnActivate (see page 25356)	Occurs when the Web module is activated.
	OnDeactivate (see page 25357)	Occurs when the Web module is deactivated.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 2829)	Generates an OnCreate (see page 2833) event.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 2830)	Generates an OnDestroy (see page 2834) event.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25352)	Creates and initializes an instance of TCustomWebDataModule (see page 25348).
	CreateNew (see page 25352)	Creates and initializes an instance of TCustomWebDataModule (see page 25348) without using an associated form file.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 25353)	Frees an instance of TCustomWebDataModule (see page 25348).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentObject (see page 19719)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.

R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Request (see page 25355)	Provides access to the current request object.
R	Response (see page 25355)	Provides access to the current response object.
R	Session (see page 25355)	Provides access to the current Web session.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.
	Tag (see page 2808)	Stores an integer value as part of a component.

182.1.5.2 TCustomWebDataModule.Create Constructor

Creates and initializes an instance of TCustomWebDataModule (see page 25348).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomWebDataModule(TComponent * AOwner);
```

Description

Do not call the TCustomWebDataModule (see page 25348) constructor. Instead, use a Web module factory object to create Web module instances.

AOwner is the component that becomes the value of the Owner property.

See Also

TWebAppPageModuleFactory (see page 25012)

TWebAppDataModuleFactory (see page 25009)

TWebDataModuleFactory (see page 25014)

TWebPageModuleFactory (see page 25019)

Owner (see page 2807)

182.1.5.3 TCustomWebDataModule.CreateNew Constructor

Creates and initializes an instance of TCustomWebDataModule (see page 25348) without using an associated form file.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); override;
```

C++

```
virtual __fastcall TCustomWebDataModule(TComponent * AOwner, int Dummy = 0);
```

Description

Use CreateNew instead of Create (see page 25352) to create a Web module without using the associated form file to initialize it.

AOwner is the component that becomes the value of the Owner property.

Dummy is not used.

See Also

TWebAppPageModuleFactory (see page 25012)

Owner (see page 2807)

182.1.5.4 TCustomWebDataModule.Destroy Destructor

Frees an instance of TCustomWebDataModule (see page 25348).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWebDataModule();
```

Description

Typically, WebSnap applications do not explicitly free Web module instances. Instead, the FinishContext method of the Web application services object frees or deactivates Web modules automatically when the application has finished processing an HTTP request.

See Also

FinishContext (see page 15228)

Free (see page 22300)

182.1.5.5 TCustomWebDataModule Methods

The methods of the TCustomWebDataModule class are listed here.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 25353)	Returns the value of the SiteModuleHelper (see page 25356) property.
	GetWebSnapObjClass (see page 25354)	Returns the class type for this adapter field.
	Notification (see page 25354)	Responds to notifications that components are about to be inserted or removed.

182.1.5.5.1 TCustomWebDataModule.GetSiteModuleHelper Method

Returns the value of the SiteModuleHelper (see page 25356) property.

Pascal

```
function GetSiteModuleHelper: TSiteModuleHelper; virtual;
```

C++

```
virtual __fastcall TSiteModuleHelper GetSiteModuleHelper();
```

Description

GetSiteModuleHelper is the protected read implementation of the SiteModuleHelper (see page 25356) property.

See Also

SiteModuleHelper (see page 25356)

182.1.5.5.2 TCustomWebDataModule.GetWebSnapObjClass Method

Returns the class type for this adapter field.

Pascal

```
function GetWebSnapObjClass: TObjectScripterClass; virtual;
```

C++

```
virtual __fastcall TObjectScripterClass GetWebSnapObjClass();
```

Description

This function returns the class type for this adapter field, which is used by the script engine to create a script object. GetWebSnapObjClass tells the script engine what type of object to create.

182.1.5.5.3 TCustomWebDataModule.Notification Method

Responds to notifications that components are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are added to or removed from the application. TCustomWebDataModule (see page 25348) overrides this method to pass the notification on to the object that is the value of the SiteModuleHelper (see page 25356) property.

See Also

SiteModuleHelper (see page 25356)

182.1.5.6 TCustomWebDataModule Properties

The properties of the TCustomWebDataModule class are listed here.

Protected Properties

	Name	Description
R	SiteModuleHelper (see page 25356)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 25356) and OnDeactivate (see page 25357) events.

Public Properties

	Name	Description
R	Request (see page 25355)	Provides access to the current request object.
R	Response (see page 25355)	Provides access to the current response object.
R	Session (see page 25355)	Provides access to the current Web session.

182.1.5.6.1 TCustomWebDataModule.Request Property

Provides access to the current request object.

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

Read Request to obtain information about the current HTTP request message. The request object provides access to the headers of the request message, as well as the content of the message.

See Also

Response (see page 25355)

182.1.5.6.2 TCustomWebDataModule.Response Property

Provides access to the current response object.

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

Read Response to obtain information about or add to the response message that the application builds as a reply to the message that the Request (see page 25355) property specifies. The response object provides access to the headers of the response message, as well as the content of the message.

See Also

Request (see page 25355)

182.1.5.6.3 TCustomWebDataModule.Session Property

Provides access to the current Web session.

Pascal

```
property Session: TAbstractWebSession;
```

C++

```
__property TAbstractWebSession Session;
```

Description

Read Session to obtain information from or add information to the current Web session. All requests are assigned a Web

session, although you can only use the session to store information about the current user if the WebSnap application includes a TSessionsService component. Web sessions persist over multiple calls from the same end user until they are terminated (for example, when the user logs out or when a timeout period elapses).

See Also

TSessionsService (see page 25582)

182.1.5.6.4 TCustomWebDataModule.SiteModuleHelper Property

References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 25356) and OnDeactivate (see page 25357) events.

Pascal

```
property SiteModuleHelper: TSiteModuleHelper;
```

C++

```
__property TSiteModuleHelper SiteModuleHelper;
```

Description

Applications can't use the protected SiteModuleHelper property. TCustomWebDataModule (see page 25348) uses this property internally to delegate tasks involved in managing the WebSnap components it contains and to generate the OnActivate (see page 25356) and OnDeactivate (see page 25357) events when the Web module is activated and deactivated.

See Also

OnActivate (see page 25356)

OnDeactivate (see page 25357)

182.1.5.7 TCustomWebDataModule Events

The events of the TCustomWebDataModule class are listed here.

Public Events

	Name	Description
	OnActivate (see page 25356)	Occurs when the Web module is activated.
	OnDeactivate (see page 25357)	Occurs when the Web module is deactivated.

182.1.5.7.1 TCustomWebDataModule.OnActivate Event

Occurs when the Web module is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to perform any initializations when the Web module is first activated. The Web application module is the first module activated, and it is activated for every HTTP request. Other Web modules are activated based on the CreateMode of their corresponding factory objects, and whether they are needed to process the current request message.

See Also

OnDeactivate (see page 25357)

CreateMode

182.1.5.7.2 TCustomWebDataModule.OnDeactivate Event

Occurs when the Web module is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to perform any final cleanup when the Web module is deactivated. Web modules are deactivated after the HTTP request has been processed, just before the Web context is freed.

See Also

OnActivate (see page 25356)

182.1.6 TCustomWebPageModule Class

TCustomWebPageModule is the base class for WebSnap modules that generate a Web page.

Class Hierarchy

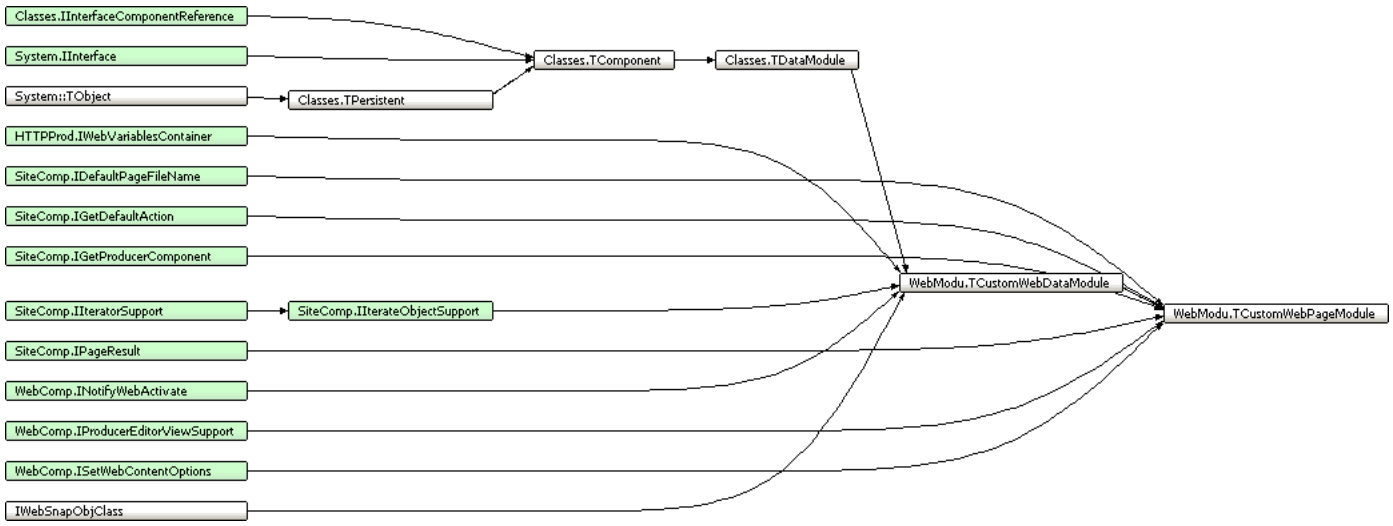

Pascal

```
TCustomWebPageModule = class(TCustomWebDataModule, IDefaultPageFileName, IPageResult,
ISetWebContentOptions, IGetDefaultAction, IGetProducerComponent, IProducerEditorViewSupport);
```

C++

```
class TCustomWebPageModule : public TCustomWebDataModule, public IDefaultPageFileName, public
```

```
IPageResult, public ISetWebContentOptions, public IGetDefaultAction, public
IGetProducerComponent, public IProducerEditorViewSupport;
```

File

WebModu

Description

TCustomWebPageModule is the common base class for Web modules that generate a Web page for a WebSnap application. This class introduces two new properties:

PageProducer (see page 25365), which is a page producer component that generates the Web page associated with this module.

DefaultAction (see page 25365), which is an adapter action associated with the module's Web page. Typically, the action sets up parameters that appear in HTTP requests for the Web page.

TCustomWebPageModule also introduces events that occur when the WebSnap application calls on the Web page module to generate its page or when it generates a redirect message to redirect a request to the associated page.

Do not create instances of TCustomWebPageModule. It is intended only as a base class. Instead, use the Web module wizard or the WebSnap application wizard to add a TCustomWebPageModule descendant to your WebSnap application. Descendant classes include TWebAppPageModule (see page 25394) and TWebPageModule (see page 25414).

See Also

TWebPageModule (see page 25414)

TWebAppPageModule (see page 25394)

182.1.6.1 TCustomWebPageModule Members

The following tables list the members exposed by TCustomWebPageModule.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
V	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetSiteModuleHelper (see page 25363)	Returns the value of the SiteModuleHelper (see page 25356) property.
V	GetSitePageModuleHelper (see page 25364)	Returns the value of the SitePageModuleHelper (see page 25366) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	GetWebSnapObjClass (see page 25354)	Returns the class type for this adapter field.
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 25364)	Responds to notifications that components are about to be inserted or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	SiteModuleHelper (see page 25356)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 25356) and OnDeactivate (see page 25357) events.
R	SitePageModuleHelper (see page 25366)	References the helper object that assists in assembling the module's Web page.

Public Events

	Name	Description
	OnActivate (see page 25356)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 25366)	Occurs after the page module generates its Web page.
	OnAfterRedirectToPage (see page 25367)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 25367)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnBeforeRedirectToPage (see page 25368)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.
	OnDeactivate (see page 25357)	Occurs when the Web module is deactivated.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 2829)	Generates an OnCreate (see page 2833) event.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 2830)	Generates an OnDestroy (see page 2834) event.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 25362)	Creates and initializes an instance of TCustomWebPageModule (see page 25357).
V	CreateNew (see page 25362)	Creates and initializes an instance of TCustomWebPageModule (see page 25357) without using an associated form file.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25363)	Frees an instance of TCustomWebPageModule (see page 25357).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DispatchPage (see page 19732)	This is DispatchPage, a member of class IPageResult.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentObject (see page 19719)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultAction (see page 19696)	This is GetDefaultAction, a member of class IGetDefaultAction.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProducerComponent (see page 19702)	This is GetProducerComponent, a member of class IGetProducerComponent.
	GetTemplateFileType (see page 24780)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.

	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetXMLData (see page 24780)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 24780)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 24780)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasScriptView (see page 24780)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 24781)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 24781)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	IncludePage (see page 19733)	This is IncludePage, a member of class IPageResult.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RedirectToPage (see page 19733)	This is RedirectToPage, a member of class IPageResult.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetDefaultPageFileName (see page 19683)	This is SetDefaultPageFileName, a member of class IDefaultPageFileName.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetWebContentOptions (see page 24782)	This is SetWebContentOptions, a member of class ISetWebContentOptions.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DefaultAction (see page 25365)	Represents the adapter action associated with the Web page module.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	PageProducer (see page 25365)	Indicates the interface of the page producer that generates the content of the Web page.
R	Request (see page 25355)	Provides access to the current request object.

R	Response (see page 25355)	Provides access to the current response object.
R	Session (see page 25355)	Provides access to the current Web session.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.
	Tag (see page 2808)	Stores an integer value as part of a component.

182.1.6.2 TCustomWebPageModule.Create Constructor

Creates and initializes an instance of TCustomWebPageModule (see page 25357).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomWebPageModule(TComponent * AOwner);
```

Description

Do not call the TCustomWebPageModule (see page 25357) constructor. Instead, use a Web page module factory object to create Web page module instances.

AOwner is the component that becomes the value of the Owner property.

See Also

TWebAppPageModuleFactory (see page 25012)

TWebPageModuleFactory (see page 25019)

Owner (see page 2807)

182.1.6.3 TCustomWebPageModule.CreateNew Constructor

Creates and initializes an instance of TCustomWebPageModule (see page 25357) without using an associated form file.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); override;
```

C++

```
virtual __fastcall TCustomWebPageModule(TComponent * AOwner, int Dummy = 0);
```

Description

Use CreateNew instead of Create (see page 25362) to create a Web page module without using the associated form file to initialize it.

AOwner is the component that becomes the value of the Owner property.

Dummy is not used.

See Also

TWebAppPageModuleFactory (see page 25012)

Owner (see page 2807)

182.1.6.4 TCustomWebPageModule.Destroy Destructor

Frees an instance of TCustomWebPageModule (see page 25357).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWebPageModule();
```

Description

Typically, WebSnap applications do not explicitly free Web page module instances. Instead, the FinishContext method of the Web application services object frees or deactivates Web modules automatically when the application has finished processing an HTTP request.

See Also

FinishContext (see page 15228)

Free (see page 22300)

182.1.6.5 TCustomWebPageModule Methods

The methods of the TCustomWebPageModule class are listed here.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 25363)	Returns the value of the SiteModuleHelper (see page 25356) property.
	GetSitePageModuleHelper (see page 25364)	Returns the value of the SitePageModuleHelper (see page 25366) property.
	Notification (see page 25364)	Responds to notifications that components are about to be inserted or removed.

182.1.6.5.1 TCustomWebPageModule.GetSiteModuleHelper Method

Returns the value of the SiteModuleHelper (see page 25356) property.

Pascal

```
function GetSiteModuleHelper: TSiteModuleHelper; override;
```

C++

```
virtual __fastcall TSiteModuleHelper GetSiteModuleHelper();
```

Description

GetSiteModuleHelper is the protected read implementation of the SiteModuleHelper (see page 25356) property. TCustomWebPageModule (see page 25357) overrides the inherited method to return the value of the SitePageModuleHelper (see page 25366) property.

See Also

SiteModuleHelper (see page 25356)

SitePageModuleHelper (see page 25366)

182.1.6.5.2 TCustomWebPageModule.GetSitePageModuleHelper Method

Returns the value of the SitePageModuleHelper (see page 25366) property.

Pascal

```
function GetSitePageModuleHelper: TSitePageModuleHelper; virtual;
```

C++

```
virtual __fastcall TSitePageModuleHelper GetSitePageModuleHelper();
```

Description

GetSitePageModuleHelper is the protected read implementation of the SitePageModuleHelper (see page 25366) property.

See Also

SitePageModuleHelper (see page 25366)

182.1.6.5.3 TCustomWebPageModule.Notification Method

Responds to notifications that components are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are added to or removed from the application. TCustomWebPageModule (see page 25357) overrides this method to update the PageProducer (see page 25365) and DefaultAction (see page 25365) properties as appropriate.

See Also

PageProducer (see page 25365)

DefaultAction (see page 25365)

182.1.6.6 TCustomWebPageModule Properties

The properties of the TCustomWebPageModule class are listed here.

Protected Properties

	Name	Description
	SitePageModuleHelper (see page 25366)	References the helper object that assists in assembling the module's Web page.

Public Properties

	Name	Description
	DefaultAction (see page 25365)	Represents the adapter action associated with the Web page module.

	PageProducer (see page 25365)	Indicates the interface of the page producer that generates the content of the Web page.

182.1.6.6.1 TCustomWebPageModule.DefaultAction Property

Represents the adapter action associated with the Web page module.

Pascal

```
property DefaultAction: TAdapterActionName;
```

C++

```
__property TAdapterActionName DefaultAction;
```

Description

Use DefaultAction to associate an adapter action with the Web page module. Server-side script can reference this action by calling on the default action of the Web page.

Typically, the default action sets up parameters that appear in HTTP requests for the Web page. By adding a default action, the parameters can then appear in requests for the Web page, which the page producer in the Web page module uses in generating its page. In addition, this action can be called to set up parameters in redirections to this Web page.

See Also

PageProducer (see page 25365)

182.1.6.6.2 TCustomWebPageModule.PageProducer Property

Indicates the interface of the page producer that generates the content of the Web page.

Pascal

```
property PageProducer: IProduceContent;
```

C++

```
__property IProduceContent PageProducer;
```

Description

PageProducer is the interface that supplies the content of the Web page this module creates. It is the interface of a page producer component that is included in the Web page module. If PageProducer is nil (Delphi) or NULL (C++), the Web page module can't produce a Web page.

See Also

TAdapterPageProducer (see page 19802)

TDataSetPagePRoducer

TXSLPagePRoducer

TPageProducer (see page 15309)

TDataSetTableProducer (see page 10277)

TQueryTableProducer (see page 8152)

TInetXPageProducer (see page 17145)

182.1.6.6.3 TCustomWebPageModule.SitePageModuleHelper Property

References the helper object that assists in assembling the module's Web page.

Pascal

```
property SitePageModuleHelper: TSitePageModuleHelper;
```

C++

```
__property TSitePageModuleHelper SitePageModuleHelper;
```

Description

Applications can't use the protected SitePageModuleHelper property. TCustomWebPageModule (see page 25357) uses this property internally to delegate tasks involved in assembling the content of its Web page and in interacting with editor views at design time.

182.1.6.7 TCustomWebPageModule Events

The events of the TCustomWebPageModule class are listed here.

Public Events

	Name	Description
	OnAfterDispatchPage (see page 25366)	Occurs after the page module generates its Web page.
	OnAfterRedirectToPage (see page 25367)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 25367)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnBeforeRedirectToPage (see page 25368)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.

182.1.6.7.1 TCustomWebPageModule.OnAfterDispatchPage Event

Occurs after the page module generates its Web page.

Pascal

```
property OnAfterDispatchPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterDispatchPage;
```

Description

Write an OnAfterDispatchPage event handler to take any final action after the page producer in the Web page module has generated its Web page. For example, the OnAfterDispatchPage event handler can make adjustments to the headers of the response message.

When OnAfterDispatchPage occurs, the Web page module has already added the content to the response message, but has not yet sent the response. This event does not occur if the BeforeDispatchPage event handler handles the request, bypassing the automatic use of the module's page producer.

Note: If the Web page module does not include a page producer, the OnAfterDispatchPage event still occurs, but the response message has no content.

See Also

OnBeforeDispatchPage (see page 25367)

PageProducer (see page 25365)

Response (see page 25355)

182.1.6.7.2 TCustomWebPageModule.OnAfterRedirectToPage Event

Occurs after the Web page module assembles a response message that represents a redirection to its Web page.

Pascal

```
property OnAfterRedirectToPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterRedirectToPage;
```

Description

Write an OnAfterRedirectToPage event handler to take any final action before returns a response message that redirects the user to this Web page. For example, the OnAfterRedirectToPage event handler can make adjustments to the headers of the response message.

This event only occurs if an adapter in the WebSnap application executes an adapter action whose options include a redirection (roRedirectHTTPGet or roRedirectHTTPPost) to this page.

See Also

OnBeforeRedirectToPage (see page 25368)

OnAfterDispatchPage (see page 25366)

RedirectOptions (see page 24074)

Response (see page 25355)

182.1.6.7.3 TCustomWebPageModule.OnBeforeDispatchPage Event

Occurs before the Web module calls on the page producer to generate the content of the Web page.

Pascal

```
property OnBeforeDispatchPage: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnBeforeDispatchPage;
```

Description

Write an OnBeforeDispatchPage event handler to process an HTTP request message before the Web page module performs the default processing.

This event can provide an alternate mechanism for handling request messages, and then set the Handled parameter to true to prevent the Web page module from fetching the template file for the page and calling on the page producer to generate the page content.

If the event handler does not handle the request (for example, if it is merely initializing values that the page producer uses), it can leave the Handled parameter as false and the Web page module will continue with the default processing.

Tip: To access the Web request and response objects from within an OnBeforeDispatchPage event handler, use the Request (see page 25355) and Response (see page 25355) properties.

See Also

OnAfterDispatchPage (see page 25366)

Request (see page 25355)

Response (see page 25355)

PageProducer (see page 25365)

182.1.6.7.4 TCustomWebPageModule.OnBeforeRedirectToPage Event

Occurs before the Web page module assembles a response that redirects the caller to its Web page.

Pascal

```
property OnBeforeRedirectToPage: TDispatchPageParamsHandledEvent;
```

C++

```
__property TDispatchPageParamsHandledEvent OnBeforeRedirectToPage;
```

Description

Write an OnBeforeRedirectToPage event handler to perform any tasks necessary for redirecting a request to the associated Web page. For example, the OnBeforeRedirectToPage event handler can add parameters to the Params parameter so that they appear in the request that results from the redirection. This event only occurs if an adapter in the WebSnap application executes an adapter action whose options include a redirection (roRedirectHTTPGet or roRedirectHTTPPost).

If the event handler sets Handled to true, the Web page module assumes it has handled the redirection and performs no further processing. Otherwise, after the event handler exits, the Web page module sets up the response message to encode the redirection parameters and calls SendRedirect.

See Also

OnAfterRedirectToPage (see page 25367)

OnBeforeDispatchPage (see page 25367)

SendRedirect (see page 15201)

Request (see page 25355)

Response (see page 25355)

182.1.7 TSiteModuleHelper Class

TSiteModuleHelper manages common tasks for Web modules in a WebSnap application.

Class Hierarchy

Pascal

```
TSiteModuleHelper = class(TObject);
```

C++

```
class TSiteModuleHelper : public TObject;
```

File

WebModu

Description

Web modules create and use an instance of TSiteModuleHelper to keep track of WebSnap objects contained in the module and to generate OnActivate (see page 25376) and OnDeactivate (see page 25376) events when the Web module is activated and deactivated.

You should not need to work directly with TSiteModuleHelper instances. The Web module creates and uses this object.

See Also

TCustomWebDataModule (see page 25348)

TCustomWebDataModule (see page 25348)

TCustomWebDataModule (see page 25348)

182.1.7.1 TSiteModuleHelper Members

The following tables list the members exposed by TSiteModuleHelper.

Protected Methods

	Name	Description
	EndIterator (see page 25371)	This is EndIterator, a member of class TSiteModuleHelper.
V	FindNamedVariable (see page 25371)	This is FindNamedVariable, a member of class TSiteModuleHelper.
	FindVariable (see page 25372)	This is FindVariable, a member of class TSiteModuleHelper.
	FindVariableContainer (see page 25372)	This is FindVariableContainer, a member of class TSiteModuleHelper.
	GetCurrentObject (see page 25372)	This is GetCurrentObject, a member of class TSiteModuleHelper.
	GetVariable (see page 25372)	This is GetVariable, a member of class TSiteModuleHelper.
	GetVariableCount (see page 25373)	This is GetVariableCount, a member of class TSiteModuleHelper.
	NextIteration (see page 25373)	This is NextIteration, a member of class TSiteModuleHelper.
	NotifyActivate (see page 25373)	This is NotifyActivate, a member of class TSiteModuleHelper.
	NotifyDeactivate (see page 25374)	This is NotifyDeactivate, a member of class TSiteModuleHelper.
	StartIterator (see page 25374)	This is StartIterator, a member of class TSiteModuleHelper.

Protected Properties

	Name	Description
R	Module (see page 25374)	This is Module, a member of class TSiteModuleHelper.

Public Events

	Name	Description
	OnActivate (see page 25376)	This is OnActivate, a member of class TSiteModuleHelper.
	OnDeactivate (see page 25376)	This is OnDeactivate, a member of class TSiteModuleHelper.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 25370)	Creates and initializes an instance of TSiteModuleHelper (see page 25368).

V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25370)	Frees an instance of TSiteModuleHelper (see page 25368).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Notification (see page 25373)	Responds to notifications that components are about to be inserted or removed.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Request (see page 25375)	Provides access to the current request object.
R	Response (see page 25375)	Provides access to the current response object.
R	Session (see page 25375)	Provides access to the current Web session.
R	WebModuleContext (see page 25376)	Keeps track of name/value pairs that store information for the Web module.

182.1.7.2 TSiteModuleHelper.Create Constructor

Creates and initializes an instance of TSiteModuleHelper (see page 25368).

Pascal

```
constructor Create(AModule: TComponent); virtual;
```

C++

```
virtual __fastcall TSiteModuleHelper(TComponent * AModule);
```

Description

Web modules call the TSiteModuleHelper (see page 25368) constructor from their own constructors.

AModule is the Web module that is using the TSiteModuleHelper (see page 25368) object to keep track of its components and activation events. The TSiteModuleHelper (see page 25368) object becomes the value of the module's SiteModuleHelper property.

See Also

SiteModuleHelper (see page 25356)

182.1.7.3 TSiteModuleHelper.Destroy Destructor

Frees an instance of TSiteModuleHelper (see page 25368).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSiteModuleHelper();
```

Description

Do not free TSiteModuleHelper (see page 25368) instances. This object is freed by the Web module that uses it.

See Also

Free (see page 22300)

182.1.7.4 TSiteModuleHelper Methods

The methods of the TSiteModuleHelper class are listed here.

Protected Methods

	Name	Description
	EndIterator (see page 25371)	This is EndIterator, a member of class TSiteModuleHelper.
	FindNamedVariable (see page 25371)	This is FindNamedVariable, a member of class TSiteModuleHelper.
	FindVariable (see page 25372)	This is FindVariable, a member of class TSiteModuleHelper.
	FindVariableContainer (see page 25372)	This is FindVariableContainer, a member of class TSiteModuleHelper.
	GetCurrentObject (see page 25372)	This is GetCurrentObject, a member of class TSiteModuleHelper.
	GetVariable (see page 25372)	This is GetVariable, a member of class TSiteModuleHelper.
	GetVariableCount (see page 25373)	This is GetVariableCount, a member of class TSiteModuleHelper.
	NextIteration (see page 25373)	This is NextIteration, a member of class TSiteModuleHelper.
	NotifyActivate (see page 25373)	This is NotifyActivate, a member of class TSiteModuleHelper.
	NotifyDeactivate (see page 25374)	This is NotifyDeactivate, a member of class TSiteModuleHelper.
	StartIterator (see page 25374)	This is StartIterator, a member of class TSiteModuleHelper.

Public Methods

	Name	Description
	Notification (see page 25373)	Responds to notifications that components are about to be inserted or removed.

182.1.7.4.1 TSiteModuleHelper.EndIterator Method

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

This is EndIterator, a member of class TSiteModuleHelper.

182.1.7.4.2 TSiteModuleHelper.FindNamedVariable Method

Pascal

```
function FindNamedVariable(const AName: string): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * FindNamedVariable(const AnsiString AName);
```

Description

This is FindNamedVariable, a member of class TSiteModuleHelper.

182.1.7.4.3 TSiteModuleHelper.FindVariable Method

Pascal

```
function FindVariable(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariable(const AnsiString AName);
```

Description

This is FindVariable, a member of class TSiteModuleHelper.

182.1.7.4.4 TSiteModuleHelper.FindVariableContainer Method

Pascal

```
function FindVariableContainer(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariableContainer(const AnsiString AName);
```

Description

This is FindVariableContainer, a member of class TSiteModuleHelper.

182.1.7.4.5 TSiteModuleHelper.GetCurrentObject Method

Pascal

```
function GetCurrentObject(OwnerData: Pointer): TObject;
```

C++

```
__fastcall TObject * GetCurrentObject(void * OwnerData);
```

Description

This is GetCurrentObject, a member of class TSiteModuleHelper.

182.1.7.4.6 TSiteModuleHelper.GetVariable Method

Pascal

```
function GetVariable(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetVariable(int I);
```

Description

This is GetVariable, a member of class TSiteModuleHelper.

182.1.7.4.7 TSiteModuleHelper.GetVariableCount Method

Pascal

```
function GetVariableCount: Integer;
```

C++

```
__fastcall int GetVariableCount();
```

Description

This is GetVariableCount, a member of class TSiteModuleHelper.

182.1.7.4.8 TSiteModuleHelper.NextIteration Method

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

This is NextIteration, a member of class TSiteModuleHelper.

182.1.7.4.9 TSiteModuleHelper.Notification Method

Responds to notifications that components are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation);
```

C++

```
__fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are added to or removed from the application. TSiteModuleHelper (see page 25368) overrides this method to ensure that any WebSnap objects added to the associated module receive notifications when the module is activated or deactivated.

182.1.7.4.10 TSiteModuleHelper.NotifyActivate Method

Pascal

```
procedure NotifyActivate;
```

C++

```
__fastcall NotifyActivate();
```

Description

This is NotifyActivate, a member of class TSiteModuleHelper.

182.1.7.4.11 TSiteModuleHelper.NotifyDeactivate Method

Pascal

```
procedure NotifyDeactivate;
```

C++

```
__fastcall NotifyDeactivate();
```

Description

This is NotifyDeactivate, a member of class TSiteModuleHelper.

182.1.7.4.12 TSiteModuleHelper.StartIterator Method

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

This is StartIterator, a member of class TSiteModuleHelper.

182.1.7.5 TSiteModuleHelper Properties

The properties of the TSiteModuleHelper class are listed here.

Protected Properties

	Name	Description
R	Module (see page 25374)	This is Module, a member of class TSiteModuleHelper.

Public Properties

	Name	Description
R	Request (see page 25375)	Provides access to the current request object.
R	Response (see page 25375)	Provides access to the current response object.
R	Session (see page 25375)	Provides access to the current Web session.
R	WebModuleContext (see page 25376)	Keeps track of name/value pairs that store information for the Web module.

182.1.7.5.1 TSiteModuleHelper.Module Property

Pascal

```
property Module: TComponent;
```

C++

```
__property TComponent * Module;
```

Description

This is Module, a member of class TSiteModuleHelper.

182.1.7.5.2 TSiteModuleHelper.Request Property

Provides access to the current request object.

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

Read Request to obtain information about the current HTTP request message. The request object provides access to the headers of the request message, as well as the content of the message.

See Also

Response (see page 25375)

182.1.7.5.3 TSiteModuleHelper.Response Property

Provides access to the current response object.

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

Read Response to obtain information about or add to the response message that the application builds as a reply to the message that the Request (see page 25375) property specifies. The response object provides access to the headers of the response message, as well as the content of the message.

See Also

Request (see page 25375)

182.1.7.5.4 TSiteModuleHelper.Session Property

Provides access to the current Web session.

Pascal

```
property Session: TAbstractWebSession;
```

C++

```
__property TAbstractWebSession Session;
```

Description

Read Session to obtain information from or add information to the current Web session. All requests are assigned a Web session, although you can only use the session to store information about the current user if the WebSnap application includes a TSessionsService component. Web sessions persist over multiple calls from the same end user until they are terminated (for example, when the user logs out or when a time-out period elapses).

See Also

TSessionsService (see page 25582)

182.1.7.5.5 TSiteModuleHelper.WebModuleContext Property

Keeps track of name/value pairs that store information for the Web module.

Pascal

```
property WebModuleContext: TWebModuleContext;
```

C++

```
__property TWebModuleContext WebModuleContext;
```

Description

Use WebModuleContext to access the TWebModuleContext object. The TWebModuleContext object maintains a list of named variables that the components in the Web module can use.

See Also

WebModuleContext (see page 15279)

182.1.7.6 TSiteModuleHelper Events

The events of the TSiteModuleHelper class are listed here.

Public Events

	Name	Description
	OnActivate (see page 25376)	This is OnActivate, a member of class TSiteModuleHelper.
	OnDeactivate (see page 25376)	This is OnDeactivate, a member of class TSiteModuleHelper.

182.1.7.6.1 TSiteModuleHelper.OnActivate Event

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

This is OnActivate, a member of class TSiteModuleHelper.

182.1.7.6.2 TSiteModuleHelper.OnDeactivate Event

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

This is OnDeactivate, a member of class TSiteModuleHelper.

182.1.8 TSitePageModuleHelper Class

TSitePageModuleHelper manages common tasks for Web page modules in a WebSnap application.

Class Hierarchy

Pascal

```
TSitePageModuleHelper = class(TSiteModuleHelper);
```

C++

```
class TSitePageModuleHelper : public TSiteModuleHelper;
```

File

WebModu

Description

Web page modules create and use an instance of TSitePageModuleHelper to assist in the following tasks:

Assembling the content of the associated Web page.

Interacting with editor views at design time.

Keeping track of WebSnap objects contained in the module.

Generating the events that occur when the Web module is activated or deactivated, and when it dispatches or redirects a request for a Web page.

You should not need to work directly with TSitePageModuleHelper instances. The Web page module creates and uses this object.

See Also

TCustomWebPageModule (see page 25357)

TCustomWebDataModule (see page 25348)

TCustomWebDataModule (see page 25348)

TCustomWebPageModule (see page 25357)

TCustomWebPageModule (see page 25357)

TCustomWebPageModule (see page 25357)

TCustomWebPageModule (see page 25357)

182.1.8.1 TSitePageModuleHelper Members

The following tables list the members exposed by TSitePageModuleHelper.

Protected Methods

	Name	Description
	DispatchPage (see page 25380)	This is DispatchPage, a member of class TSitePageModuleHelper.
	EndIterator (see page 25371)	This is EndIterator, a member of class TSiteModuleHelper.
	FindNamedVariable (see page 25371)	This is FindNamedVariable, a member of class TSiteModuleHelper.

	FindVariable (see page 25372)	This is FindVariable, a member of class TSiteModuleHelper.
	FindVariableContainer (see page 25372)	This is FindVariableContainer, a member of class TSiteModuleHelper.
	GetCurrentObject (see page 25372)	This is GetCurrentObject, a member of class TSiteModuleHelper.
	GetDefaultAction (see page 25381)	This is GetDefaultAction, a member of class TSitePageModuleHelper.
V	GetPageProducer (see page 25381)	This is GetPageProducer, a member of class TSitePageModuleHelper.
	GetProducerComponent (see page 25381)	This is GetProducerComponent, a member of class TSitePageModuleHelper.
	GetRedirectParams (see page 25381)	This is GetRedirectParams, a member of class TSitePageModuleHelper.
	GetTemplateFileType (see page 25382)	This is GetTemplateFileType, a member of class TSitePageModuleHelper.
	GetTemplateStream (see page 25382)	This is GetTemplateStream, a member of class TSitePageModuleHelper.
	GetVariable (see page 25372)	This is GetVariable, a member of class TSiteModuleHelper.
	GetVariableCount (see page 25373)	This is GetVariableCount, a member of class TSiteModuleHelper.
	GetXMLData (see page 25382)	This is GetXMLData, a member of class TSitePageModuleHelper.
	GetXSLData (see page 25382)	This is GetXSLData, a member of class TSitePageModuleHelper.
	HasHTMLBrowserView (see page 25382)	This is HasHTMLBrowserView, a member of class TSitePageModuleHelper.
	HasScriptView (see page 25383)	This is HasScriptView, a member of class TSitePageModuleHelper.
	HasXMLBrowserView (see page 25383)	This is HasXMLBrowserView, a member of class TSitePageModuleHelper.
	HasXSLBrowserView (see page 25383)	This is HasXSLBrowserView, a member of class TSitePageModuleHelper.
	HTMLFromTemplate (see page 25383)	This is HTMLFromTemplate, a member of class TSitePageModuleHelper.
V	ImplDispatchPage (see page 25384)	This is ImplDispatchPage, a member of class TSitePageModuleHelper.
V	ImplIncludePage (see page 25384)	This is ImplIncludePage, a member of class TSitePageModuleHelper.
V	ImplRedirectToPage (see page 25384)	This is ImplRedirectToPage, a member of class TSitePageModuleHelper.
V	ImplSetWebContentOptions (see page 25384)	This is ImplSetWebContentOptions, a member of class TSitePageModuleHelper.
	IncludePage (see page 25385)	This is IncludePage, a member of class TSitePageModuleHelper.
	NextIteration (see page 25373)	This is NextIteration, a member of class TSiteModuleHelper.
	NotifyActivate (see page 25373)	This is NotifyActivate, a member of class TSiteModuleHelper.
	NotifyDeactivate (see page 25374)	This is NotifyDeactivate, a member of class TSiteModuleHelper.
	RedirectToPage (see page 25385)	This is RedirectToPage, a member of class TSitePageModuleHelper.
	SetDefaultPageFileName (see page 25385)	This is SetDefaultPageFileName, a member of class TSitePageModuleHelper.
	SetWebContentOptions (see page 25385)	This is SetWebContentOptions, a member of class TSitePageModuleHelper.
	StartIterator (see page 25374)	This is StartIterator, a member of class TSiteModuleHelper.

Protected Properties

	Name	Description
R	Module (see page 25374)	This is Module, a member of class TSiteModuleHelper.

Public Events

	Name	Description
	OnActivate (see page 25376)	This is OnActivate, a member of class TSiteModuleHelper.
	OnAfterDispatchPage (see page 25386)	Occurs after the Web page module generates its Web page.
	OnAfterRedirectToPage (see page 25387)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 25387)	Occurs before the Web module generates the content of the Web page.
	OnBeforeRedirectToPage (see page 25387)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.
	OnDeactivate (see page 25376)	This is OnDeactivate, a member of class TSiteModuleHelper.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 25379)	Creates and initializes an instance of TSitePageModuleHelper (see page 25377).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25380)	Frees an instance of TSitePageModuleHelper (see page 25377).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Notification (see page 25373)	Responds to notifications that components are about to be inserted or removed.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	DefaultAction (see page 25386)	Represents the adapter action associated with the Web page module.
R	Request (see page 25375)	Provides access to the current request object.
R	Response (see page 25375)	Provides access to the current response object.
R	Session (see page 25375)	Provides access to the current Web session.
R	WebModuleContext (see page 25376)	Keeps track of name/value pairs that store information for the Web module.

182.1.8.2 TSitePageModuleHelper.Create Constructor

Creates and initializes an instance of TSitePageModuleHelper (see page 25377).

Pascal

```
constructor Create(AModule: TComponent); override;
```

C++

```
virtual __fastcall TSitePageModuleHelper(TComponent * AModule);
```

Description

Web page modules call the TSitePageModuleHelper (see page 25377) constructor from their own constructors.

AModule is the Web page module that is using the TSitePageModuleHelper (see page 25377) object to generate events and help assemble its Web content. The TSitePageModuleHelper (see page 25377) object becomes the value of the module's SitePageModuleHelper property.

See Also

SitePageModuleHelper (see page 25366)

182.1.8.3 TSitePageModuleHelper.Destroy Destructor

Frees an instance of TSitePageModuleHelper (see page 25377).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSitePageModuleHelper();
```

Description

Do not free TSitePageModuleHelper (see page 25377) instances. This object is freed by the Web page module that uses it.

See Also

Free (see page 22300)

182.1.8.4 TSitePageModuleHelper Methods

The methods of the TSitePageModuleHelper class are listed here.

Protected Methods

	Name	Description
	DispatchPage (see page 25380)	This is DispatchPage, a member of class TSitePageModuleHelper.
	GetDefaultAction (see page 25381)	This is GetDefaultAction, a member of class TSitePageModuleHelper.
V	GetPageProducer (see page 25381)	This is GetPageProducer, a member of class TSitePageModuleHelper.
	GetProducerComponent (see page 25381)	This is GetProducerComponent, a member of class TSitePageModuleHelper.
	GetRedirectParams (see page 25381)	This is GetRedirectParams, a member of class TSitePageModuleHelper.
	GetTemplateFileType (see page 25382)	This is GetTemplateFileType, a member of class TSitePageModuleHelper.
	GetTemplateStream (see page 25382)	This is GetTemplateStream, a member of class TSitePageModuleHelper.
	GetXMLData (see page 25382)	This is GetXMLData, a member of class TSitePageModuleHelper.
	GetXSLData (see page 25382)	This is GetXSLData, a member of class TSitePageModuleHelper.
	HasHTMLBrowserView (see page 25382)	This is HasHTMLBrowserView, a member of class TSitePageModuleHelper.
	HasScriptView (see page 25383)	This is HasScriptView, a member of class TSitePageModuleHelper.
	HasXMLBrowserView (see page 25383)	This is HasXMLBrowserView, a member of class TSitePageModuleHelper.
	HasXSLBrowserView (see page 25383)	This is HasXSLBrowserView, a member of class TSitePageModuleHelper.
	HTMLFromTemplate (see page 25383)	This is HTMLFromTemplate, a member of class TSitePageModuleHelper.
V	ImplDispatchPage (see page 25384)	This is ImplDispatchPage, a member of class TSitePageModuleHelper.
V	ImplIncludePage (see page 25384)	This is ImplIncludePage, a member of class TSitePageModuleHelper.
V	ImplRedirectToPage (see page 25384)	This is ImplRedirectToPage, a member of class TSitePageModuleHelper.
V	ImplSetWebContentOptions (see page 25384)	This is ImplSetWebContentOptions, a member of class TSitePageModuleHelper.
	IncludePage (see page 25385)	This is IncludePage, a member of class TSitePageModuleHelper.
	RedirectToPage (see page 25385)	This is RedirectToPage, a member of class TSitePageModuleHelper.
	SetDefaultPageFileName (see page 25385)	This is SetDefaultPageFileName, a member of class TSitePageModuleHelper.
	SetWebContentOptions (see page 25385)	This is SetWebContentOptions, a member of class TSitePageModuleHelper.

182.1.8.4.1 TSitePageModuleHelper.DispatchPage Method

Pascal

```
function DispatchPage(const APageName: string; AWebResponse: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean DispatchPage(const AnsiString APageName, TWebResponse AWebResponse);
```

Description

This is DispatchPage, a member of class TSitePageModuleHelper.

182.1.8.4.2 TSitePageModuleHelper.GetDefaultAction Method

Pascal

```
function GetDefaultAction: TComponent;
```

C++

```
__fastcall TComponent * GetDefaultAction();
```

Description

This is GetDefaultAction, a member of class TSitePageModuleHelper.

182.1.8.4.3 TSitePageModuleHelper.GetPageProducer Method

Pascal

```
function GetPageProducer: IProduceContent; virtual;
```

C++

```
virtual __fastcall IProduceContent GetPageProducer();
```

Description

This is GetPageProducer, a member of class TSitePageModuleHelper.

182.1.8.4.4 TSitePageModuleHelper.GetProducerComponent Method

Pascal

```
function GetProducerComponent: TComponent;
```

C++

```
__fastcall TComponent * GetProducerComponent();
```

Description

This is GetProducerComponent, a member of class TSitePageModuleHelper.

182.1.8.4.5 TSitePageModuleHelper.GetRedirectParams Method

Pascal

```
procedure GetRedirectParams(const APageName: string; AParams: TStrings);
```

C++

```
__fastcall GetRedirectParams(const AnsiString APageName, TStrings AParams);
```

Description

This is GetRedirectParams, a member of class TSitePageModuleHelper.

182.1.8.4.6 TSitePageModuleHelper.GetTemplateFileType Method

Pascal

```
function GetTemplateFileType: string;
```

C++

```
__fastcall AnsiString GetTemplateFileType();
```

Description

This is GetTemplateFileType, a member of class TSitePageModuleHelper.

182.1.8.4.7 TSitePageModuleHelper.GetTemplateStream Method

Pascal

```
function GetTemplateStream(const APage: string; out AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream GetTemplateStream(const AnsiString APage, Boolean AOwned);
```

Description

This is GetTemplateStream, a member of class TSitePageModuleHelper.

182.1.8.4.8 TSitePageModuleHelper.GetXMLData Method

Pascal

```
function GetXMLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXMLData(Boolean Owned);
```

Description

This is GetXMLData, a member of class TSitePageModuleHelper.

182.1.8.4.9 TSitePageModuleHelper.GetXSLData Method

Pascal

```
function GetXSLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXSLData(Boolean Owned);
```

Description

This is GetXSLData, a member of class TSitePageModuleHelper.

182.1.8.4.10 TSitePageModuleHelper.HasHTMLBrowserView Method

Pascal

```
function HasHTMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasHTMLBrowserView();
```

Description

This is HasHTMLBrowserView, a member of class TSitePageModuleHelper.

182.1.8.4.11 TSitePageModuleHelper.HasScriptView Method

Pascal

```
function HasScriptView: Boolean;
```

C++

```
__fastcall Boolean HasScriptView();
```

Description

This is HasScriptView, a member of class TSitePageModuleHelper.

182.1.8.4.12 TSitePageModuleHelper.HasXMLBrowserView Method

Pascal

```
function HasXMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXMLBrowserView();
```

Description

This is HasXMLBrowserView, a member of class TSitePageModuleHelper.

182.1.8.4.13 TSitePageModuleHelper.HasXSLBrowserView Method

Pascal

```
function HasXSLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXSLBrowserView();
```

Description

This is HasXSLBrowserView, a member of class TSitePageModuleHelper.

182.1.8.4.14 TSitePageModuleHelper.HTMLFromTemplate Method

Pascal

```
function HTMLFromTemplate(AStream: TStream): string;
```

C++

```
__fastcall AnsiString HTMLFromTemplate(TStream AStream);
```

Description

This is HTMLFromTemplate, a member of class TSitePageModuleHelper.

182.1.8.4.15 TSitePageModuleHelper.ImplDispatchPage Method

Pascal

```
function ImplDispatchPage(const APageName: string; AWebResponse: TWebResponse): Boolean;
virtual;
```

C++

```
virtual __fastcall Boolean ImplDispatchPage(const AnsiString APageName, TWebResponse
AWebResponse);
```

Description

This is ImplDispatchPage, a member of class TSitePageModuleHelper.

182.1.8.4.16 TSitePageModuleHelper.ImplIncludePage Method

Pascal

```
function ImplIncludePage(const APageName: string; var AOwned: Boolean): TStream; virtual;
```

C++

```
virtual __fastcall TStream ImplIncludePage(const AnsiString APageName, Boolean AOwned);
```

Description

This is ImplIncludePage, a member of class TSitePageModuleHelper.

182.1.8.4.17 TSitePageModuleHelper.ImplRedirectToPage Method

Pascal

```
function ImplRedirectToPage(const APageName: string; AParams: TStrings; AResponse:
TWebResponse): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplRedirectToPage(const AnsiString APageName, TStrings AParams,
TWebResponse AResponse);
```

Description

This is ImplRedirectToPage, a member of class TSitePageModuleHelper.

182.1.8.4.18 TSitePageModuleHelper.ImplSetWebContentOptions Method

Pascal

```
procedure ImplSetWebContentOptions(AOptions: TWebContentOptions); virtual;
```

C++

```
virtual __fastcall ImplSetWebContentOptions(TWebContentOptions AOptions);
```

Description

This is ImplSetWebContentOptions, a member of class TSitePageModuleHelper.

182.1.8.4.19 TSitePageModuleHelper.IncludePage Method

Pascal

```
function IncludePage(const APageName: string; var AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream IncludePage(const AnsiString APageName, Boolean AOwned);
```

Description

This is IncludePage, a member of class TSitePageModuleHelper.

182.1.8.4.20 TSitePageModuleHelper.RedirectToPage Method

Pascal

```
function RedirectToPage(const APageName: string; AParams: TStrings; AResponse: TWebResponse):
Boolean;
```

C++

```
__fastcall Boolean RedirectToPage(const AnsiString APageName, TStrings AParams, TWebResponse
AResponse);
```

Description

This is RedirectToPage, a member of class TSitePageModuleHelper.

182.1.8.4.21 TSitePageModuleHelper.SetDefaultPageFileName Method

Pascal

```
procedure SetDefaultPageFileName(const AValue: string);
```

C++

```
__fastcall SetDefaultPageFileName(const AnsiString AValue);
```

Description

This is SetDefaultPageFileName, a member of class TSitePageModuleHelper.

182.1.8.4.22 TSitePageModuleHelper.SetWebContentOptions Method

Pascal

```
procedure SetWebContentOptions(AOptions: TWebContentOptions);
```

C++

```
__fastcall SetWebContentOptions(TWebContentOptions AOptions);
```

Description

This is SetWebContentOptions, a member of class TSitePageModuleHelper.

182.1.8.5 TSitePageModuleHelper Properties

The properties of the TSitePageModuleHelper class are listed here.

Public Properties

	Name	Description
	DefaultAction (see page 25386)	Represents the adapter action associated with the Web page module.

182.1.8.5.1 TSitePageModuleHelper.DefaultAction Property

Represents the adapter action associated with the Web page module.

Pascal

```
property DefaultAction: TAdapterActionName;
```

C++

```
__property TAdapterActionName DefaultAction;
```

Description

Use DefaultAction to associate an adapter action with the Web page module. Server-side script can reference this action by calling on the default action of the Web page.

Typically, the default action sets up parameters that appear in HTTP requests for the Web page. By adding a default action, the parameters can then appear in requests for the Web page, which the page producer in the Web page module uses in generating its page. In addition, this action can be called to set up parameters in redirections to this Web page.

See Also

PageProducer (see page 25365)

182.1.8.6 TSitePageModuleHelper Events

The events of the TSitePageModuleHelper class are listed here.

Public Events

	Name	Description
	OnAfterDispatchPage (see page 25386)	Occurs after the Web page module generates its Web page.
	OnAfterRedirectToPage (see page 25387)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 25387)	Occurs before the Web module generates the content of the Web page.
	OnBeforeRedirectToPage (see page 25387)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.

182.1.8.6.1 TSitePageModuleHelper.OnAfterDispatchPage Event

Occurs after the Web page module generates its Web page.

Pascal

```
property OnAfterDispatchPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterDispatchPage;
```

Description

OnAfterDispatchPage uses the same event handler as that of the Web page module's OnAfterDispatchPage event. The TSitePageModuleHelper (see page 25377) object generates this event after the Web page module writes the content for its Web page to the response object.

See Also

ondeactivate (see page 25376)

OnAfterDispatchPage (see page 25366)

182.1.8.6.2 TSitePageModuleHelper.OnAfterRedirectToPage Event

Occurs after the Web page module assembles a response message that represents a redirection to its Web page.

Pascal

```
property OnAfterRedirectToPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterRedirectToPage;
```

Description

OnAfterRedirectToPage uses the same event handler as that of the Web page module's OnAfterRedirectToPage event. The TSitePageModuleHelper (see page 25377) object generates this event after the Web page module sends a response redirecting the caller to its Web page.

See Also

OnBeforeRedirectToPage (see page 25387)

OnAfterRedirectToPage (see page 25367)

182.1.8.6.3 TSitePageModuleHelper.OnBeforeDispatchPage Event

Occurs before the Web module generates the content of the Web page.

Pascal

```
property OnBeforeDispatchPage: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnBeforeDispatchPage;
```

Description

OnBeforeDispatchPage uses the same event handler as that of the Web page module's OnBeforeDispatchPage event. The TSitePageModuleHelper (see page 25377) object generates this event before the Web page module generates the content for its Web page.

See Also

OnAfterDispatchPage (see page 25386)

OnBeforeDispatchPage (see page 25367)

182.1.8.6.4 TSitePageModuleHelper.OnBeforeRedirectToPage Event

Occurs before the Web page module assembles a response that redirects the caller to its Web page.

Pascal

```
property OnBeforeRedirectToPage: TDispatchPageParamsHandledEvent;
```

C++

```
__property TDispatchPageParamsHandledEvent OnBeforeRedirectToPage;
```

Description

OnBeforeRedirectToPage uses the same event handler as that of the Web page module's OnBeforeRedirectToPage event. The TSitePageModuleHelper (see page 25377) object generates this event before the Web page module assembles and sends a response redirecting the caller to its Web page.

See Also

OnAfterRedirectToPage (see page 25387)

OnBeforeRedirectToPage (see page 25368)

182.1.9 TWebAppDataModule Class

TWebAppDataModule is used as the WebSnap application module when there is no need to generate a page.

Class Hierarchy

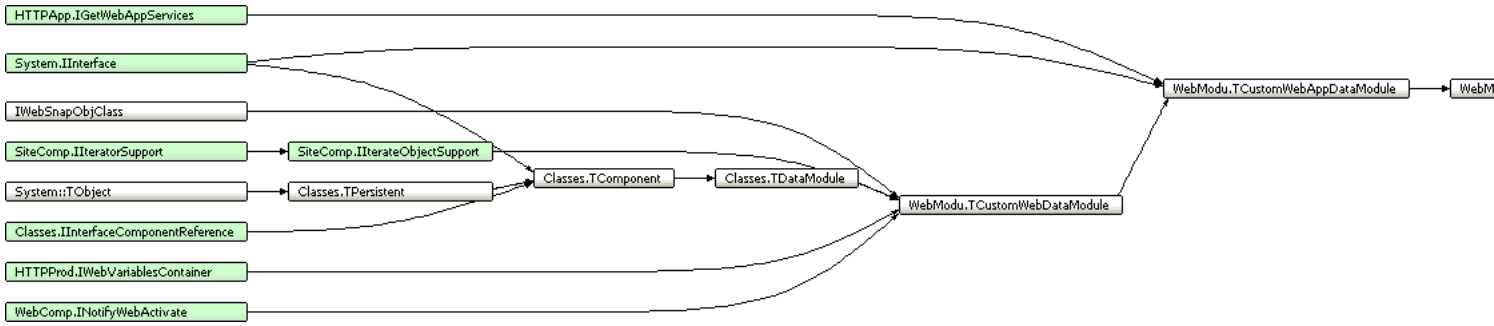

Pascal

```
TWebAppDataModule = class(TCustomWebAppDataModule);
```

C++

```
class TWebAppDataModule : public TCustomWebAppDataModule;
```

File

WebModu

Description

TWebAppDataModule is a container for WebSnap system components such as dispatchers, global adapters, user list, and session components. The WebSnap Application Wizard automatically adds system components to the TWebAppDataModule when it creates a TWebAppDataModule. System (see page 22079) components may be added or removed later. TWebAppComponents is the component in a TWebAppDataModule that references all of the system components in use by the WebSnap application.

See Also

Web Modules

182.1.9.1 TWebAppDataModule Members

The following tables list the members exposed by TWebAppDataModule.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
V	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetSiteModuleHelper (see page 25353)	Returns the value of the SiteModuleHelper (see page 25356) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebAppServices (see page 25337)	Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.
V	GetWebSnapObjClass (see page 25354)	Returns the class type for this adapter field.
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 25338)	Responds to notifications that components are about to be inserted or removed.
V	Notification (see page 25354)	Responds to notifications that components are about to be inserted or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	QueryInterface (see page 25338)	Returns a reference to a specified interface if the module supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.

	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 25356)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 25356) and OnDeactivate (see page 25357) events.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 2829)	Generates an OnCreate (see page 2833) event.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 2830)	Generates an OnDestroy (see page 2834) event.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 2827)	Creates an instance of a data module.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 25336)	Creates a new TCustomWebAppDataModule (see page 25332) instance.
	Create (see page 25352)	Creates and initializes an instance of TCustomWebDataModule (see page 25348).
	CreateNew (see page 2828)	Creates an instance of a data module and registers with the global screen object.
	CreateNew (see page 25337)	Creates a data module without using the associated form file to initialize it.
	CreateNew (see page 25352)	Creates and initializes an instance of TCustomWebDataModule (see page 25348) without using an associated form file.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2829)	Destroys the instance of the data module.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 25337)	Frees a Web application data module instance.
	Destroy (see page 25353)	Frees an instance of TCustomWebDataModule (see page 25348).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.

	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentObject (see page 19719)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetWebAppServices (see page 15221)	This is GetWebAppServices, a member of class IGetWebAppServices.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.

R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Request (see page 25355)	Provides access to the current request object.
R	Response (see page 25355)	Provides access to the current response object.
R	Session (see page 25355)	Provides access to the current Web session.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnActivate (see page 25393)	Occurs when the Web module is activated.
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDeactivate (see page 25393)	Occurs when the Web module is deactivated.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.

Published Properties

	Name	Description
	AppServices (see page 25392)	Provides read and write access to the application services stored in this TCustomWebAppDataModule object.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.
	Tag (see page 2808)	Stores an integer value as part of a component.

182.1.9.2 TWebAppDataModule Properties

The properties of the TWebAppDataModule class are listed here.

Published Properties

	Name	Description
	AppServices (see page 25392)	Provides read and write access to the application services stored in this TCustomWebAppDataModule object.

182.1.9.2.1 TWebAppDataModule.AppServices Property

Provides read and write access to the application services stored in this TCustomWebAppDataModule object.

Pascal

```
property AppServices: IWebAppServices;
```

C++

```
__property IWebAppServices AppServices;
```

Description

AppServices provides access to the application services implemented by a component in this Web application module. The application services component sets up the Web context for each incoming request message, passes incoming request objects on to a dispatcher for handling, and provides the final backstop for exceptions that arise while processing request messages.

Typically, this property references the interface of a TWebAppComponents object.

See Also

TWebAppComponents (see page 24958)

WebContext (see page 24687)

182.1.9.3 TWebAppDataModule Events

The events of the TWebAppDataModule class are listed here.

Published Events

	Name	Description
	OnActivate (see page 25393)	Occurs when the Web module is activated.
	OnDeactivate (see page 25393)	Occurs when the Web module is deactivated.

182.1.9.3.1 TWebAppDataModule.OnActivate Event

Occurs when the Web module is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to perform any initializations when the Web module is first activated. The Web application module is the first module activated, and it is activated for every HTTP request. Other Web modules are activated based on the CreateMode of their corresponding factory objects, and whether they are needed to process the current request message.

See Also

OnDeactivate (see page 25357)

CreateMode

182.1.9.3.2 TWebAppDataModule.OnDeactivate Event

Occurs when the Web module is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to perform any final cleanup when the Web module is deactivated. Web modules are deactivated after the HTTP request has been processed, just before the Web context is freed.

See Also

OnActivate (see page 25356)

182.1.10 TWebAppPageModule Class

TWebAppPageModule is used as the WebSnap application module when the application module generates a page.

Class Hierarchy

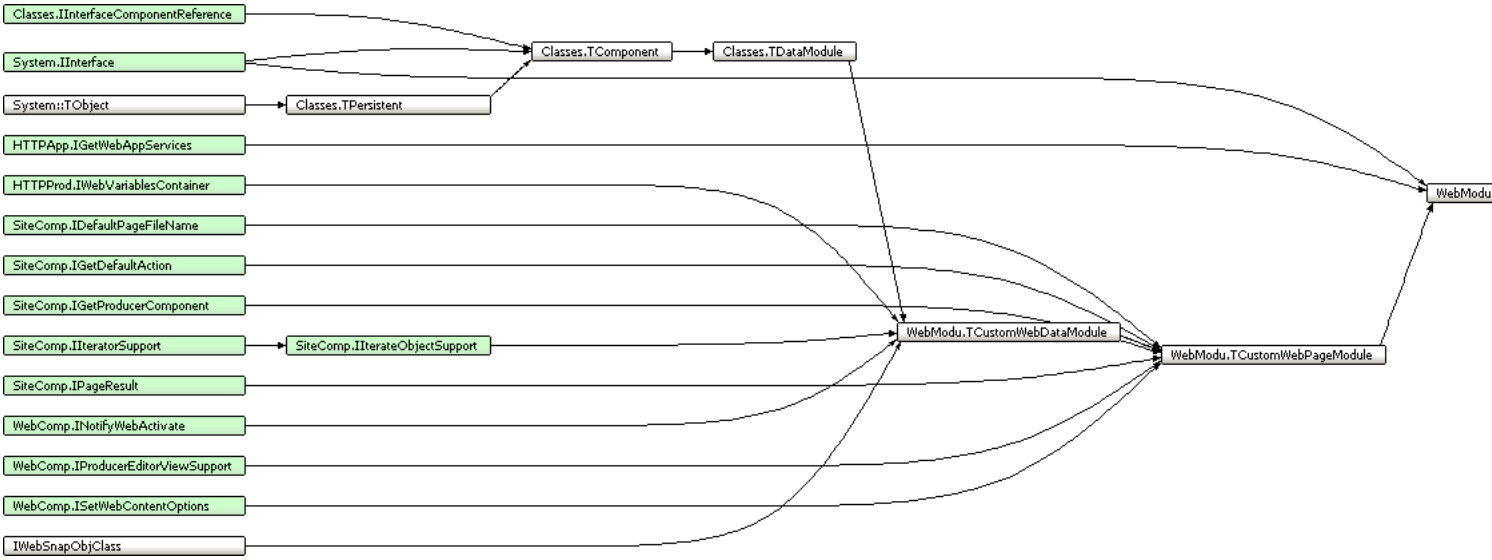

Pascal

```
TWebAppPageModule = class(TCustomWebAppPageModule);
```

C++

```
class TWebAppPageModule : public TCustomWebAppPageModule;
```

File

WebModu

Description

Like TWebAppDataModule (see page 25388), TWebAppPageModule is a container for WebSnap system components such as dispatchers, global adapters, user list, and session components. The WebSnap Application Wizard automatically adds system components to the TWebAppPageModule when it creates a TWebAppPageModule. System (see page 22079) components may be added or removed later. TWebAppComponents is the component in a TWebAppPageModule that references all of the system components in use by the WebSnap application.

The WebAppPageModule is different from TWebAppDataModule (see page 25388) because it generates a page. The PageProducer (see page 25400) property references the producer component responsible for generating content. This property must not be nil (Delphi) or NULL (C++) or no content will be generated. The WebSnap Application Wizard automatically adds a page producer component when it creates a new TWebAppPageModule.

See Also

Web Modules

182.1.10.1 TWebAppPageModule Members

The following tables list the members exposed by TWebAppPageModule.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
V	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetSiteModuleHelper (see page 25353)	Returns the value of the SiteModuleHelper (see page 25356) property.
V	GetSiteModuleHelper (see page 25363)	Returns the value of the SiteModuleHelper (see page 25356) property.
V	GetSitePageModuleHelper (see page 25364)	Returns the value of the SitePageModuleHelper (see page 25366) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetWebAppServices (see page 25346)	Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.
V	GetWebSnapObjClass (see page 25354)	Returns the class type for this adapter field.
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 25346)	Responds to notifications that components are about to be inserted or removed.
V	Notification (see page 25354)	Responds to notifications that components are about to be inserted or removed.
V	Notification (see page 25364)	Responds to notifications that components are about to be inserted or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	QueryInterface (see page 25346)	Returns a reference to a specified interface if the module supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.

	SetAppServices (see page 25347)	Sets the value of the AppServices (see page 25347) property.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	SiteModuleHelper (see page 25356)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 25356) and OnDeactivate (see page 25357) events.
R	SitePageModuleHelper (see page 25366)	References the helper object that assists in assembling the module's Web page.

Public Events

	Name	Description
	OnAfterRedirectToPage (see page 25367)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeRedirectToPage (see page 25368)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 2829)	Generates an OnCreate (see page 2833) event.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 2830)	Generates an OnDestroy (see page 2834) event.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 2827)	Creates an instance of a data module.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 25344)	Creates and initializes an instance of TCustomWebAppPageModule (see page 25339).

	Create (see page 25352)	Creates and initializes an instance of TCustomWebDataModule (see page 25348).
	Create (see page 25362)	Creates and initializes an instance of TCustomWebPageModule (see page 25357).
	CreateNew (see page 2828)	Creates an instance of a data module and registers with the global screen object.
	CreateNew (see page 25345)	Creates and initializes an instance of TCustomWebAppPageModule (see page 25339) without using an associated form file.
	CreateNew (see page 25352)	Creates and initializes an instance of TCustomWebDataModule (see page 25348) without using an associated form file.
	CreateNew (see page 25362)	Creates and initializes an instance of TCustomWebPageModule (see page 25357) without using an associated form file.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2829)	Destroys the instance of the data module.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 25345)	Frees an instance of TCustomWebAppPageModule (see page 25339).
	Destroy (see page 25353)	Frees an instance of TCustomWebDataModule (see page 25348).
	Destroy (see page 25363)	Frees an instance of TCustomWebPageModule (see page 25357).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DispatchPage (see page 19732)	This is DispatchPage, a member of class IPageResult.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentObject (see page 19719)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultAction (see page 19696)	This is GetDefaultAction, a member of class IGetDefaultAction.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProducerComponent (see page 19702)	This is GetProducerComponent, a member of class IGetProducerComponent.
	GetTemplateFileType (see page 24780)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetWebAppServices (see page 15221)	This is GetWebAppServices, a member of class IGetWebAppServices.
	GetXMLData (see page 24780)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 24780)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 24780)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.

	HasScriptView (see page 24780)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 24781)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 24781)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	IncludePage (see page 19733)	This is IncludePage, a member of class IPageResult.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RedirectToPage (see page 19733)	This is RedirectToPage, a member of class IPageResult.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetDefaultPageFileName (see page 19683)	This is SetDefaultPageFileName, a member of class IDefaultPageFileName.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetWebContentOptions (see page 24782)	This is SetWebContentOptions, a member of class ISetWebContentOptions.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Request (see page 25355)	Provides access to the current request object.
R	Response (see page 25355)	Provides access to the current response object.
R	Session (see page 25355)	Provides access to the current Web session.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnActivate (see page 25401)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 25401)	Occurs after the page module generates its Web page.
	OnBeforeDispatchPage (see page 25402)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDeactivate (see page 25402)	Occurs when the Web module is deactivated.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.

Published Properties

	Name	Description
	AppServices (see page 25399)	Provides read and write access to the application services stored in this TCustomWebAppPageModule object.
	DefaultAction (see page 25400)	Represents the adapter action associated with the Web page module.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.
	PageProducer (see page 25400)	Indicates the interface of the page producer that generates the content of the Web page.
	Tag (see page 2808)	Stores an integer value as part of a component.

182.1.10.2 TWebAppPageModule Properties

The properties of the TWebAppPageModule class are listed here.

Published Properties

	Name	Description
	AppServices (see page 25399)	Provides read and write access to the application services stored in this TCustomWebAppPageModule object.
	DefaultAction (see page 25400)	Represents the adapter action associated with the Web page module.
	PageProducer (see page 25400)	Indicates the interface of the page producer that generates the content of the Web page.

182.1.10.2.1 TWebAppPageModule.AppServices Property

Provides read and write access to the application services stored in this TCustomWebAppPageModule object.

Pascal

```
property AppServices: IWebAppServices;
```

C++

```
__property IWebAppServices AppServices;
```

Description

AppServices provides access to the application services implemented by a component in this Web application module. The application services component sets up the Web context for each incoming request message, passes incoming request objects on to a dispatcher for handling, and provides the final backstop for exceptions that arise while processing request messages.

Typically, this property references the interface of a TWebAppComponents object.

See Also

TWebAppComponents (see page 24958)

WebContext (see page 24687)

182.1.10.2.2 TWebAppPageModule.DefaultAction Property

Represents the adapter action associated with the Web page module.

Pascal

```
property DefaultAction: TComponent;
```

C++

```
__property TAdapterActionName DefaultAction;__property TComponent * DefaultAction;
```

Description

Use DefaultAction to associate an adapter action with the Web page module. Server-side script can reference this action by calling on the default action of the Web page.

Typically, the default action sets up parameters that appear in HTTP requests for the Web page. By adding a default action, the parameters can then appear in requests for the Web page, which the page producer in the Web page module uses in generating its page. In addition, this action can be called to set up parameters in redirections to this Web page.

See Also

PageProducer (see page 25365)

182.1.10.2.3 TWebAppPageModule.PageProducer Property

Indicates the interface of the page producer that generates the content of the Web page.

Pascal

```
property PageProducer: IProduceContent;
```

C++

```
__property IProduceContent PageProducer;
```

Description

PageProducer is the interface that supplies the content of the Web page this module creates. It is the interface of a page producer component that is included in the Web page module. If PageProducer is nil (Delphi) or NULL (C++), the Web page module can't produce a Web page.

See Also

TAdapterPageProducer (see page 19802)

TDataSetPagePRoducer

TXSLPagePRoducer

TPageProducer (see page 15309)

TDataSetTableProducer (see page 10277)

TQueryTableProducer (see page 8152)

TInetXPageProducer (see page 17145)

182.1.10.3 TWebAppPageModule Events

The events of the TWebAppPageModule class are listed here.

Published Events

	Name	Description
	OnActivate (see page 25401)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 25401)	Occurs after the page module generates its Web page.
	OnBeforeDispatchPage (see page 25402)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnDeactivate (see page 25402)	Occurs when the Web module is deactivated.

182.1.10.3.1 TWebAppPageModule.OnActivate Event

Occurs when the Web module is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to perform any initializations when the Web module is first activated. The Web application module is the first module activated, and it is activated for every HTTP request. Other Web modules are activated based on the CreateMode of their corresponding factory objects, and whether they are needed to process the current request message.

See Also

OnDeactivate (see page 25357)

CreateMode

182.1.10.3.2 TWebAppPageModule.OnAfterDispatchPage Event

Occurs after the page module generates its Web page.

Pascal

```
property OnAfterDispatchPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterDispatchPage;
```

Description

Write an OnAfterDispatchPage event handler to take any final action after the page producer in the Web page module has generated its Web page. For example, the OnAfterDispatchPage event handler can make adjustments to the headers of the response message.

When OnAfterDispatchPage occurs, the Web page module has already added the content to the response message, but has not yet sent the response. This event does not occur if the BeforeDispatchPage event handler handles the request, bypassing the automatic use of the module's page producer.

Note: If the Web page module does not include a page producer, the OnAfterDispatchPage event still occurs, but the response message has no content.

See Also

OnBeforeDispatchPage (see page 25367)

PageProducer (see page 25365)

Response (see page 25355)

182.1.10.3.3 TWebAppPageModule.OnBeforeDispatchPage Event

Occurs before the Web module calls on the page producer to generate the content of the Web page.

Pascal

```
property OnBeforeDispatchPage: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnBeforeDispatchPage;
```

Description

Write an OnBeforeDispatchPage event handler to process an HTTP request message before the Web page module performs the default processing.

This event can provide an alternate mechanism for handling request messages, and then set the Handled parameter to true to prevent the Web page module from fetching the template file for the page and calling on the page producer to generate the page content.

If the event handler does not handle the request (for example, if it is merely initializing values that the page producer uses), it can leave the Handled parameter as false and the Web page module will continue with the default processing.

Tip: To access the Web request and response objects from within an OnBeforeDispatchPage event handler, use the Request and Response properties.

See Also

OnAfterDispatchPage (see page 25366)

Request (see page 25355)

Response (see page 25355)

PageProducer (see page 25365)

182.1.10.3.4 TWebAppPageModule.OnDeactivate Event

Occurs when the Web module is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to perform any final cleanup when the Web module is deactivated. Web modules are deactivated after the HTTP request has been processed, just before the Web context is freed.

See Also

OnActivate (see page 25356)

182.1.11 TWebAppPageModuleHelper Class

Class Hierarchy

Pascal

```
TWebAppPageModuleHelper = class(TSitePageModuleHelper);
```

C++

```
class TWebAppPageModuleHelper : public TSitePageModuleHelper;
```

File

WebModu

Description

This is class WebModu.TWebAppPageModuleHelper.

182.1.11.1 TWebAppPageModuleHelper Members

The following tables list the members exposed by TWebAppPageModuleHelper.

Protected Methods

	Name	Description
	DispatchPage (see page 25380)	This is DispatchPage, a member of class TSitePageModuleHelper.
	EndIterator (see page 25371)	This is EndIterator, a member of class TSiteModuleHelper.
V	FindNamedVariable (see page 25371)	This is FindNamedVariable, a member of class TSiteModuleHelper.
	FindVariable (see page 25372)	This is FindVariable, a member of class TSiteModuleHelper.
	FindVariableContainer (see page 25372)	This is FindVariableContainer, a member of class TSiteModuleHelper.
	GetCurrentObject (see page 25372)	This is GetCurrentObject, a member of class TSiteModuleHelper.
	GetDefaultAction (see page 25381)	This is GetDefaultAction, a member of class TSitePageModuleHelper.
V	GetPageProducer (see page 25405)	This is GetPageProducer, a member of class TWebAppPageModuleHelper.
	GetProducerComponent (see page 25381)	This is GetProducerComponent, a member of class TSitePageModuleHelper.
	GetRedirectParams (see page 25381)	This is GetRedirectParams, a member of class TSitePageModuleHelper.
	GetTemplateFileType (see page 25382)	This is GetTemplateFileType, a member of class TSitePageModuleHelper.
	GetTemplateStream (see page 25382)	This is GetTemplateStream, a member of class TSitePageModuleHelper.
	GetVariable (see page 25372)	This is GetVariable, a member of class TSiteModuleHelper.
	GetVariableCount (see page 25373)	This is GetVariableCount, a member of class TSiteModuleHelper.
	GetXMLData (see page 25382)	This is GetXMLData, a member of class TSitePageModuleHelper.
	GetXSLData (see page 25382)	This is GetXSLData, a member of class TSitePageModuleHelper.
	HasHTMLBrowserView (see page 25382)	This is HasHTMLBrowserView, a member of class TSitePageModuleHelper.
	HasScriptView (see page 25383)	This is HasScriptView, a member of class TSitePageModuleHelper.
	HasXMLBrowserView (see page 25383)	This is HasXMLBrowserView, a member of class TSitePageModuleHelper.
	HasXSLBrowserView (see page 25383)	This is HasXSLBrowserView, a member of class TSitePageModuleHelper.
	HTMLFromTemplate (see page 25383)	This is HTMLFromTemplate, a member of class TSitePageModuleHelper.
V	ImplDispatchPage (see page 25384)	This is ImplDispatchPage, a member of class TSitePageModuleHelper.
V	ImplIncludePage (see page 25384)	This is ImplIncludePage, a member of class TSitePageModuleHelper.
V	ImplRedirectToPage (see page 25384)	This is ImplRedirectToPage, a member of class TSitePageModuleHelper.
V	ImplSetWebContentOptions (see page 25384)	This is ImplSetWebContentOptions, a member of class TSitePageModuleHelper.
	IncludePage (see page 25385)	This is IncludePage, a member of class TSitePageModuleHelper.
	NextIteration (see page 25373)	This is NextIteration, a member of class TSiteModuleHelper.

	NotifyActivate (see page 25373)	This is NotifyActivate, a member of class TSiteModuleHelper.
	NotifyDeactivate (see page 25374)	This is NotifyDeactivate, a member of class TSiteModuleHelper.
	RedirectToPage (see page 25385)	This is RedirectToPage, a member of class TSitePageModuleHelper.
	SetDefaultPageFileName (see page 25385)	This is SetDefaultPageFileName, a member of class TSitePageModuleHelper.
	SetWebContentOptions (see page 25385)	This is SetWebContentOptions, a member of class TSitePageModuleHelper.
	StartIterator (see page 25374)	This is StartIterator, a member of class TSiteModuleHelper.

Protected Properties

	Name	Description
R	Module (see page 25374)	This is Module, a member of class TSiteModuleHelper.

Public Events

	Name	Description
	OnActivate (see page 25376)	This is OnActivate, a member of class TSiteModuleHelper.
	OnAfterDispatchPage (see page 25386)	Occurs after the Web page module generates its Web page.
	OnAfterRedirectToPage (see page 25387)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 25387)	Occurs before the Web module generates the content of the Web page.
	OnBeforeRedirectToPage (see page 25387)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.
	OnDeactivate (see page 25376)	This is OnDeactivate, a member of class TSiteModuleHelper.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 25370)	Creates and initializes an instance of TSiteModuleHelper (see page 25368).
V	Create (see page 25379)	Creates and initializes an instance of TSitePageModuleHelper (see page 25377).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 25370)	Frees an instance of TSiteModuleHelper (see page 25368).
V	Destroy (see page 25380)	Frees an instance of TSitePageModuleHelper (see page 25377).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Notification (see page 25373)	Responds to notifications that components are about to be inserted or removed.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	DefaultAction (see page 25386)	Represents the adapter action associated with the Web page module.
R	Request (see page 25375)	Provides access to the current request object.
R	Response (see page 25375)	Provides access to the current response object.
R	Session (see page 25375)	Provides access to the current Web session.
R	WebModuleContext (see page 25376)	Keeps track of name/value pairs that store information for the Web module.

182.1.11.2 TWebAppPageModuleHelper Methods

The methods of the TWebAppPageModuleHelper class are listed here.

Protected Methods

	Name	Description
V	GetPageProducer (see page 25405)	This is GetPageProducer, a member of class TWebAppPageModuleHelper.

182.1.11.2.1 TWebAppPageModuleHelper.GetPageProducer Method

Pascal

```
function GetPageProducer: IProduceContent; override;
```

C++

```
virtual __fastcall IProduceContent GetPageProducer();
```

Description

This is GetPageProducer, a member of class TWebAppPageModuleHelper.

182.1.12 TWebDataModule Class

TWebDataModule is a container for components in a WebSnap application.

Class Hierarchy

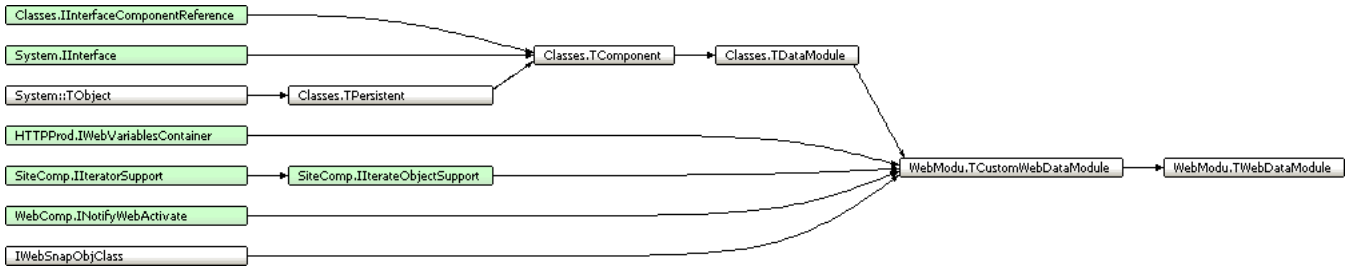

Pascal

```
TWebDataModule = class(TCustomWebDataModule);
```

C++

```
class TWebDataModule : public TCustomWebDataModule;
```

File

WebModu

Description

Use TWebDataModule as a container for components that may be used by other modules in your WebSnap application. TWebDataModule is compatible with WebSnap applications. The initialization section of TWebDataModule registers a factory that is used by the WebSnap application to create the data module.

See Also

Web Modules

182.1.12.1 TWebDataModule Members

The following tables list the members exposed by TWebDataModule.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
V	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetSiteModuleHelper (see page 25353)	Returns the value of the SiteModuleHelper (see page 25356) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	GetWebSnapObjClass (see page 25354)	Returns the class type for this adapter field.
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.

	Notification (see page 2783)	Forwards notification messages to all owned components.
	Notification (see page 25354)	Responds to notifications that components are about to be inserted or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 25356)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 25356) and OnDeactivate (see page 25357) events.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 2829)	Generates an OnCreate (see page 2833) event.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 2830)	Generates an OnDestroy (see page 2834) event.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 2827)	Creates an instance of a data module.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 25352)	Creates and initializes an instance of TCustomWebDataModule (see page 25348).
	CreateNew (see page 2828)	Creates an instance of a data module and registers with the global screen object.

	CreateNew (see page 25352)	Creates and initializes an instance of TCustomWebDataModule (see page 25348) without using an associated form file.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2829)	Destroys the instance of the data module.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 25353)	Frees an instance of TCustomWebDataModule (see page 25348).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentObject (see page 19719)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)

V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Request (see page 25355)	Provides access to the current request object.
R	Response (see page 25355)	Provides access to the current response object.
R	Session (see page 25355)	Provides access to the current Web session.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnActivate (see page 25409)	Occurs when the Web module is activated.
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDeactivate (see page 25410)	Occurs when the Web module is deactivated.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.
	Tag (see page 2808)	Stores an integer value as part of a component.

182.1.12.2 TWebDataModule Events

The events of the TWebDataModule class are listed here.

Published Events

	Name	Description
	OnActivate (see page 25409)	Occurs when the Web module is activated.
	OnDeactivate (see page 25410)	Occurs when the Web module is deactivated.

182.1.12.2.1 TWebDataModule.OnActivate Event

Occurs when the Web module is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to perform any initializations when the Web module is first activated. The Web application module is the first module activated, and it is activated for every HTTP request. Other Web modules are activated based on the CreateMode of their corresponding factory objects, and whether they are needed to process the current request message.

See Also

OnDeactivate (see page 25357)

CreateMode

182.1.12.2.2 TWebDataModule.OnDeactivate Event

Occurs when the Web module is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to perform any final cleanup when the Web module is deactivated. Web modules are deactivated after the HTTP request has been processed, just before the Web context is freed.

See Also

OnActivate (see page 25356)

182.1.13 TWebNotifyContainer Class

Class Hierarchy

Pascal

```
TWebNotifyContainer = class(TObject);
```

C++

```
class TWebNotifyContainer : public TObject;
```

File

WebModu

Description

This is class WebModu.TWebNotifyContainer.

182.1.13.1 TWebNotifyContainer Members

The following tables list the members exposed by TWebNotifyContainer.

Protected Fields

	Name	Description
	FList (see page 25413)	This is FList, a member of class TWebNotifyContainer.

Protected Methods

	Name	Description
	GetCount (see page 25413)	This is GetCount, a member of class TWebNotifyContainer.

Public Methods

	Name	Description
	Add (see page 25413)	This is Add, a member of class TWebNotifyContainer.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25411)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25412)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NotifyWebActivate (see page 25414)	This is NotifyWebActivate, a member of class TWebNotifyContainer.
	NotifyWebDeactivate (see page 25414)	This is NotifyWebDeactivate, a member of class TWebNotifyContainer.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

182.1.13.2 TWebNotifyContainer.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TWebNotifyContainer();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

182.1.13.3 TWebNotifyContainer.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebNotifyContainer();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

182.1.13.4 TWebNotifyContainer Fields

The fields of the TWebNotifyContainer class are listed here.

Protected Fields

	Name	Description
	FList (see page 25413)	This is FList, a member of class TWebNotifyContainer.

182.1.13.4.1 TWebNotifyContainer.FList Field

Pascal

```
FList: TComponentList;
```

C++

```
TComponentList FList;
```

Description

This is FList, a member of class TWebNotifyContainer.

182.1.13.5 TWebNotifyContainer Methods

The methods of the TWebNotifyContainer class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 25413)	This is GetCount, a member of class TWebNotifyContainer.

Public Methods

	Name	Description
	Add (see page 25413)	This is Add, a member of class TWebNotifyContainer.
	NotifyWebActivate (see page 25414)	This is NotifyWebActivate, a member of class TWebNotifyContainer.
	NotifyWebDeactivate (see page 25414)	This is NotifyWebDeactivate, a member of class TWebNotifyContainer.

182.1.13.5.1 TWebNotifyContainer.Add Method

Pascal

```
procedure Add(AComponent: TComponent);
```

C++

```
__fastcall Add(TComponent * AComponent);
```

Description

This is Add, a member of class TWebNotifyContainer.

182.1.13.5.2 TWebNotifyContainer.GetCount Method

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

This is GetCount, a member of class TWebNotifyContainer.

182.1.13.5.3 TWebNotifyContainer.NotifyWebActivate Method

Pascal

```
procedure NotifyWebActivate;
```

C++

```
__fastcall NotifyWebActivate();
```

Description

This is NotifyWebActivate, a member of class TWebNotifyContainer.

182.1.13.5.4 TWebNotifyContainer.NotifyWebDeactivate Method

Pascal

```
procedure NotifyWebDeactivate;
```

C++

```
__fastcall NotifyWebDeactivate();
```

Description

This is NotifyWebDeactivate, a member of class TWebNotifyContainer.

182.1.14 TWebPageModule Class

TWebPageModule is a WebSnap module that generates a page.

Class Hierarchy

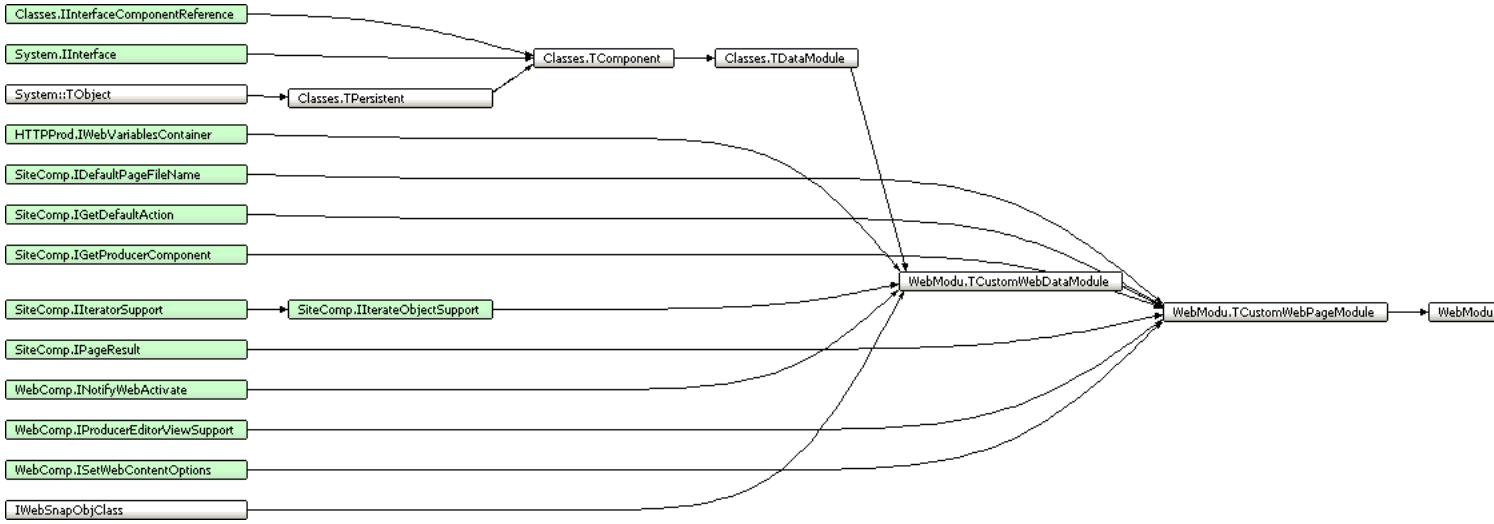

Pascal

```
TWebPageModule = class(TCustomWebPageModule);
```

C++

```
class TWebPageModule : public TCustomWebPageModule;
```

File

WebModu

Description

Use TWebPageModule in a WebSnap application to hold all the components needed for generating a specific Web page. When you use the WebSnap page module wizard to create a TWebPageModule descendant, the wizard adds a page producer to the module. The page producer generates the content of the Web page and adds code to the unit it creates (in Delphi, the code is added to the initialization section; in C++, it is added to a #pragma startup). The added code registers the TWebPageModule descendant with a factory object that the WebSnap application can use to create the Web page module.

The page producer that the wizard adds to the Web page module becomes the value of the PageProducer (see page 25420) property. If the PageProducer (see page 25420) property is nil (Delphi) or NULL (C++), the Web page module can't generate the content of a Web page. Add additional components to the Web page module that assist the page producer in creating its content. Other modules in the WebSnap application can access any components you add to the Web page module, but typically Web page module components are intended only for internal use.

See Also

TWebAppDataModule (see page 25388)

TWebAppPageModule (see page 25394)

TWebDataModule (see page 25405)

TWebPageModuleFactory (see page 25019)

Web Modules

182.1.14.1 TWebPageModule Members

The following tables list the members exposed by TWebPageModule.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2830)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoCreate (see page 2831)	This is DoCreate, a member of class TDataModule.
V	DoDestroy (see page 2831)	This is DoDestroy, a member of class TDataModule.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
V	GetChildren (see page 2831)	This is GetChildren, a member of class TDataModule.

	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetSiteModuleHelper (see page 25353)	Returns the value of the SiteModuleHelper (see page 25356) property.
V	GetSiteModuleHelper (see page 25363)	Returns the value of the SiteModuleHelper (see page 25356) property.
V	GetSitePageModuleHelper (see page 25364)	Returns the value of the SitePageModuleHelper (see page 25366) property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
V	GetWebSnapObjClass (see page 25354)	Returns the class type for this adapter field.
	HandleCreateException (see page 2831)	This is HandleCreateException, a member of class TDataModule.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 25354)	Responds to notifications that components are about to be inserted or removed.
V	Notification (see page 25364)	Responds to notifications that components are about to be inserted or removed.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
V	ReadState (see page 2832)	This is ReadState, a member of class TDataModule.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	SiteModuleHelper (see page 25356)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 25356) and OnDeactivate (see page 25357) events.
R	SitePageModuleHelper (see page 25366)	References the helper object that assists in assembling the module's Web page.

Public Events

	Name	Description
	OnAfterRedirectToPage (see page 25367)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeRedirectToPage (see page 25368)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 2829)	Generates an OnCreate (see page 2833) event.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 2830)	Generates an OnDestroy (see page 2834) event.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
V	Create (see page 2827)	Creates an instance of a data module.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 25352)	Creates and initializes an instance of TCustomWebDataModule (see page 25348).
V	Create (see page 25362)	Creates and initializes an instance of TCustomWebPageModule (see page 25357).
V	CreateNew (see page 2828)	Creates an instance of a data module and registers with the global screen object.
V	CreateNew (see page 25352)	Creates and initializes an instance of TCustomWebDataModule (see page 25348) without using an associated form file.
V	CreateNew (see page 25362)	Creates and initializes an instance of TCustomWebPageModule (see page 25357) without using an associated form file.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2829)	Destroys the instance of the data module.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 25353)	Frees an instance of TCustomWebDataModule (see page 25348).
V	Destroy (see page 25363)	Frees an instance of TCustomWebPageModule (see page 25357).
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DispatchPage (see page 19732)	This is DispatchPage, a member of class IPageResult.
	EndIterator (see page 19723)	This is EndIterator, a member of class IIteratorSupport.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	FindVariable (see page 15330)	This is FindVariable, a member of class IWebVariablesContainer.
	FindVariableContainer (see page 15331)	This is FindVariableContainer, a member of class IWebVariablesContainer.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetCurrentObject (see page 19719)	This is GetCurrentObject, a member of class IIterateObjectSupport.

	GetDefaultAction (see page 19696)	This is GetDefaultAction, a member of class IGetDefaultAction.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetProducerComponent (see page 19702)	This is GetProducerComponent, a member of class IGetProducerComponent.
	GetTemplateFileType (see page 24780)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetVariable (see page 15331)	This is GetVariable, a member of class IWebVariablesContainer.
	GetVariableCount (see page 15331)	This is GetVariableCount, a member of class IWebVariablesContainer.
	GetXMLData (see page 24780)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 24780)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 24780)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasScriptView (see page 24780)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 24781)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 24781)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	IncludePage (see page 19733)	This is IncludePage, a member of class IPageResult.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 19723)	Advances to the objects next item.
	NotifyActivate (see page 24778)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 24778)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RedirectToPage (see page 19733)	This is RedirectToPage, a member of class IPageResult.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetDefaultPageFileName (see page 19683)	This is SetDefaultPageFileName, a member of class IDefaultPageFileName.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetWebContentOptions (see page 24782)	This is SetWebContentOptions, a member of class ISetWebContentOptions.
	StartIterator (see page 19724)	This is StartIterator, a member of class IIteratorSupport.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.

R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DesignOffset (see page 2832)	Specifies the design offset for the data module at design time.
	DesignSize (see page 2832)	Specifies the design size for the data module at design time.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Request (see page 25355)	Provides access to the current request object.
R	Response (see page 25355)	Provides access to the current response object.
R	Session (see page 25355)	Provides access to the current Web session.
R	VariableCount (see page 15331)	This is VariableCount, a member of class IWebVariablesContainer.
R	Variables (see page 15332)	This is Variables, a member of class IWebVariablesContainer.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnActivate (see page 25421)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 25421)	Occurs after the page module generates its Web page.
	OnBeforeDispatchPage (see page 25421)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnCreate (see page 2833)	Occurs when an application instantiates a data module.
	OnDeactivate (see page 25422)	Occurs when the Web module is deactivated.
	OnDestroy (see page 2834)	Occurs when the data module is about to be destroyed.

Published Properties

	Name	Description
	DefaultAction (see page 25419)	Represents the adapter action associated with the Web page module.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	OldCreateOrder (see page 2833)	Specifies when OnCreate (see page 2833) and OnDestroy (see page 2834) events occur.
	PageProducer (see page 25420)	Indicates the interface of the page producer that generates the content of the Web page.
	Tag (see page 2808)	Stores an integer value as part of a component.

182.1.14.2 TWebPageModule Properties

The properties of the TWebPageModule class are listed here.

Published Properties

	Name	Description
	DefaultAction (see page 25419)	Represents the adapter action associated with the Web page module.
	PageProducer (see page 25420)	Indicates the interface of the page producer that generates the content of the Web page.

182.1.14.2.1 TWebPageModule.DefaultAction Property

Represents the adapter action associated with the Web page module.

Pascal

```
property DefaultAction: TComponent;
```

C++

```
__property TAdapterActionName DefaultAction;__property TComponent * DefaultAction;
```

Description

Use DefaultAction to associate an adapter action with the Web page module. Server-side script can reference this action by calling on the default action of the Web page.

Typically, the default action sets up parameters that appear in HTTP requests for the Web page. By adding a default action, the parameters can then appear in requests for the Web page, which the page producer in the Web page module uses in generating its page. In addition, this action can be called to set up parameters in redirections to this Web page.

See Also

PageProducer (see page 25365)

182.1.14.2.2 TWebPageModule.PageProducer Property

Indicates the interface of the page producer that generates the content of the Web page.

Pascal

```
property PageProducer: IProduceContent;
```

C++

```
__property IProduceContent PageProducer;
```

Description

PageProducer is the interface that supplies the content of the Web page this module creates. It is the interface of a page producer component that is included in the Web page module. If PageProducer is nil (Delphi) or NULL (C++), the Web page module can't produce a Web page.

See Also

TAdapterPageProducer (see page 19802)

TDataSetPagePRoducer

TXSLPagePRoducer

TPageProducer (see page 15309)

TDataSetTableProducer (see page 10277)

TQueryTableProducer (see page 8152)

TInetXPageProducer (see page 17145)

182.1.14.3 TWebPageModule Events

The events of the TWebPageModule class are listed here.

Published Events

	Name	Description
	OnActivate (see page 25421)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 25421)	Occurs after the page module generates its Web page.
	OnBeforeDispatchPage (see page 25421)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnDeactivate (see page 25422)	Occurs when the Web module is deactivated.

182.1.14.3.1 TWebPageModule.OnActivate Event

Occurs when the Web module is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to perform any initializations when the Web module is first activated. The Web application module is the first module activated, and it is activated for every HTTP request. Other Web modules are activated based on the CreateMode of their corresponding factory objects, and whether they are needed to process the current request message.

See Also

OnDeactivate (see page 25357)

CreateMode

182.1.14.3.2 TWebPageModule.OnAfterDispatchPage Event

Occurs after the page module generates its Web page.

Pascal

```
property OnAfterDispatchPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterDispatchPage;
```

Description

Write an OnAfterDispatchPage event handler to take any final action after the page producer in the Web page module has generated its Web page. For example, the OnAfterDispatchPage event handler can make adjustments to the headers of the response message.

When OnAfterDispatchPage occurs, the Web page module has already added the content to the response message, but has not yet sent the response. This event does not occur if the BeforeDispatchPage event handler handles the request, bypassing the automatic use of the module's page producer.

Note: If the Web page module does not include a page producer, the OnAfterDispatchPage event still occurs, but the response message has no content.

See Also

OnBeforeDispatchPage (see page 25367)

PageProducer (see page 25365)

Response (see page 25355)

182.1.14.3.3 TWebPageModule.OnBeforeDispatchPage Event

Occurs before the Web module calls on the page producer to generate the content of the Web page.

Pascal

```
property OnBeforeDispatchPage: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnBeforeDispatchPage;
```

Description

Write an OnBeforeDispatchPage event handler to process an HTTP request message before the Web page module performs the default processing.

This event can provide an alternate mechanism for handling request messages, and then set the Handled parameter to true to prevent the Web page module from fetching the template file for the page and calling on the page producer to generate the page content.

If the event handler does not handle the request (for example, if it is merely initializing values that the page producer uses), it can leave the Handled parameter as false and the Web page module will continue with the default processing.

Tip: To access the Web request and response objects from within an OnBeforeDispatchPage event handler, use the Request and Response properties.

See Also

OnAfterDispatchPage (see page 25366)

Request (see page 25355)

Response (see page 25355)

PageProducer (see page 25365)

182.1.14.3.4 TWebPageModule.OnDeactivate Event

Occurs when the Web module is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to perform any final cleanup when the Web module is deactivated. Web modules are deactivated after the HTTP request has been processed, just before the Web context is freed.

See Also

OnActivate (see page 25356)

182.1.15 TWebPageModuleHelper Class

Class Hierarchy

Pascal

```
TWebPageModuleHelper = class(TSitePageModuleHelper);
```

C++

```
class TWebPageModuleHelper : public TSitePageModuleHelper;
```

File

WebModu

Description

This is class WebModu.TWebPageModuleHelper.

182.1.15.1 TWebPageModuleHelper Members

The following tables list the members exposed by TWebPageModuleHelper.

Protected Methods

	Name	Description
	DispatchPage (see page 25380)	This is DispatchPage, a member of class TSitePageModuleHelper.
	EndIterator (see page 25371)	This is EndIterator, a member of class TSiteModuleHelper.
V	FindNamedVariable (see page 25371)	This is FindNamedVariable, a member of class TSiteModuleHelper.
	FindVariable (see page 25372)	This is FindVariable, a member of class TSiteModuleHelper.
	FindVariableContainer (see page 25372)	This is FindVariableContainer, a member of class TSiteModuleHelper.
	GetCurrentObject (see page 25372)	This is GetCurrentObject, a member of class TSiteModuleHelper.
	GetDefaultAction (see page 25381)	This is GetDefaultAction, a member of class TSitePageModuleHelper.
V	GetPageProducer (see page 25425)	This is GetPageProducer, a member of class TWebPageModuleHelper.
	GetProducerComponent (see page 25381)	This is GetProducerComponent, a member of class TSitePageModuleHelper.
	GetRedirectParams (see page 25381)	This is GetRedirectParams, a member of class TSitePageModuleHelper.
	GetTemplateFileType (see page 25382)	This is GetTemplateFileType, a member of class TSitePageModuleHelper.
	GetTemplateStream (see page 25382)	This is GetTemplateStream, a member of class TSitePageModuleHelper.
	GetVariable (see page 25372)	This is GetVariable, a member of class TSiteModuleHelper.
	GetVariableCount (see page 25373)	This is GetVariableCount, a member of class TSiteModuleHelper.
	GetXMLData (see page 25382)	This is GetXMLData, a member of class TSitePageModuleHelper.
	GetXSLData (see page 25382)	This is GetXSLData, a member of class TSitePageModuleHelper.
	HasHTMLBrowserView (see page 25382)	This is HasHTMLBrowserView, a member of class TSitePageModuleHelper.
	HasScriptView (see page 25383)	This is HasScriptView, a member of class TSitePageModuleHelper.
	HasXMLBrowserView (see page 25383)	This is HasXMLBrowserView, a member of class TSitePageModuleHelper.
	HasXSLBrowserView (see page 25383)	This is HasXSLBrowserView, a member of class TSitePageModuleHelper.
	HTMLFromTemplate (see page 25383)	This is HTMLFromTemplate, a member of class TSitePageModuleHelper.
V	ImplDispatchPage (see page 25384)	This is ImplDispatchPage, a member of class TSitePageModuleHelper.
V	ImplIncludePage (see page 25384)	This is ImplIncludePage, a member of class TSitePageModuleHelper.
V	ImplRedirectToPage (see page 25384)	This is ImplRedirectToPage, a member of class TSitePageModuleHelper.
V	ImplSetWebContentOptions (see page 25384)	This is ImplSetWebContentOptions, a member of class TSitePageModuleHelper.
	IncludePage (see page 25385)	This is IncludePage, a member of class TSitePageModuleHelper.
	NextIteration (see page 25373)	This is NextIteration, a member of class TSiteModuleHelper.
	NotifyActivate (see page 25373)	This is NotifyActivate, a member of class TSiteModuleHelper.
	NotifyDeactivate (see page 25374)	This is NotifyDeactivate, a member of class TSiteModuleHelper.
	RedirectToPage (see page 25385)	This is RedirectToPage, a member of class TSitePageModuleHelper.
	SetDefaultPageFileName (see page 25385)	This is SetDefaultPageFileName, a member of class TSitePageModuleHelper.
	SetWebContentOptions (see page 25385)	This is SetWebContentOptions, a member of class TSitePageModuleHelper.
	StartIterator (see page 25374)	This is StartIterator, a member of class TSiteModuleHelper.

Protected Properties

	Name	Description
R	Module (see page 25374)	This is Module, a member of class TSiteModuleHelper.

Public Events

	Name	Description
	OnActivate (see page 25376)	This is OnActivate, a member of class TSiteModuleHelper.
	OnAfterDispatchPage (see page 25386)	Occurs after the Web page module generates its Web page.
	OnAfterRedirectToPage (see page 25387)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 25387)	Occurs before the Web module generates the content of the Web page.
	OnBeforeRedirectToPage (see page 25387)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.
	OnDeactivate (see page 25376)	This is OnDeactivate, a member of class TSiteModuleHelper.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 25370)	Creates and initializes an instance of TSiteModuleHelper (see page 25368).
V	Create (see page 25379)	Creates and initializes an instance of TSitePageModuleHelper (see page 25377).
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 25370)	Frees an instance of TSiteModuleHelper (see page 25368).
V	Destroy (see page 25380)	Frees an instance of TSitePageModuleHelper (see page 25377).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Notification (see page 25373)	Responds to notifications that components are about to be inserted or removed.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	DefaultAction (see page 25386)	Represents the adapter action associated with the Web page module.
R	Request (see page 25375)	Provides access to the current request object.
R	Response (see page 25375)	Provides access to the current response object.

R	Session (see page 25375)	Provides access to the current Web session.
R	WebModuleContext (see page 25376)	Keeps track of name/value pairs that store information for the Web module.

182.1.15.2 TWebPageModuleHelper Methods

The methods of the TWebPageModuleHelper class are listed here.

Protected Methods

	Name	Description
V	GetPageProducer (see page 25425)	This is GetPageProducer, a member of class TWebPageModuleHelper.

182.1.15.2.1 TWebPageModuleHelper.GetPageProducer Method

Pascal

```
function GetPageProducer: IProduceContent; override;
```

C++

```
virtual __fastcall IProduceContent GetPageProducer();
```

Description

This is GetPageProducer, a member of class TWebPageModuleHelper.

182.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FindNamedModuleVariable (see page 25426)	This is function WebModu.FindNamedModuleVariable.
	GetModuleVariable (see page 25426)	This is function WebModu.GetModuleVariable.
	GetModuleVariableCount (see page 25426)	This is function WebModu.GetModuleVariableCount.

182.2.1 WebModu.FindNamedModuleVariable Function

Pascal

```
function FindNamedModuleVariable(AModule: TComponent; const AName: string): TComponent;
```

C++

```
TComponent * FindNamedModuleVariable(TComponent * AModule, const AnsiString AName);
```

File

WebModu

Description

This is function WebModu.FindNamedModuleVariable.

182.2.2 WebModu.GetModuleVariable Function

Pascal

```
function GetModuleVariable(AModule: TComponent; Index: Integer): TComponent;
```

C++

```
TComponent * GetModuleVariable(TComponent * AModule, int Index);
```

File

WebModu

Description

This is function WebModu.GetModuleVariable.

182.2.3 WebModu.GetModuleVariableCount Function

Pascal

```
function GetModuleVariableCount(AModule: TComponent): Integer;
```

C++

```
int GetModuleVariableCount(TComponent * AModule);
```

File

WebModu

Description

This is function WebModu.GetModuleVariableCount.

182.3 Types

The following table lists types in this documentation.

Types

Name	Description
TGetParamsEvent (see page 25428)	This is type WebModu.TGetParamsEvent.

182.3.1 WebModu.TGetParamsEvent Type

Pascal

```
TGetParamsEvent = procedure (Sender: TObject; Params: TStrings);
```

C++

```
(Sender: TObject; Params: TStrings) ( TGetParamsEvent)();
```

File

WebModu

Description

This is type WebModu.TGetParamsEvent.

183 WebNode Namespace

This is namespace WebNode.

Topics

Name	Description
Interfaces (see page 25430)	The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IWebNode (see page 25430)	IWebNode is the interface for executing invokable interface calls from the client of a Web Service.

183.1 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IWebNode (see page 25430)	IWebNode is the interface for executing invokable interface calls from the client of a Web Service.

183.1.1 IWebNode Interface

IWebNode is the interface for executing invokable interface calls from the client of a Web Service.

Class Hierarchy

WebNode.IWebNode

Pascal

```
IWebNode = interface;
```

C++

```
__interface IWebNode;
```

File

WebNode

Description

TRIO objects use IWebNode to execute method calls on an invokable interface. IWebNode takes an encoded method call, executes it, and returns the encoded results. Typically, objects that implement IWebNode use a specific transport protocol to forward the encoded method call to a remote server. For example, the THTTPReqResp class implements this interface for making interface calls to a remote Web Service application using HTTP.

When implementing the IWebNode interface, you can use another IWebNode implementation to delegate part of the method execution. For example, you can write a class that compresses or encrypts the encoded method call, passes it to another IWebNode implementation that expects the compressed or encrypted string, and then decompresses or decrypts the results returned by the other IWebNode implementation.

Note: C++ method declarations that use IWebNode use the _di_IWebNode type instead. This type is a DelphiInterface wrapper around the IWebNode interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< >  _di_IWebNode;
```

See Also

THTTPReqResp (see page 19961)

TRio

183.1.1.1 IWebNode Members

The following tables list the members exposed by IWebNode.

Public Methods

	Name	Description
	BeforeExecute (see page 25431)	Performs any transport-specific setup before executing an encoded method call.
	Execute (see page 25432)	This is the overview for the Execute method overload.
	GetMimeBoundary (see page 25433)	Returns the value of the MimeBoundary (see page 25433) property.
	SetMimeBoundary (see page 25433)	Sets the value of the MimeBoundary (see page 25433) property.

Public Properties

	Name	Description
	MimeBoundary (see page 25433)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

183.1.1.2 IWebNode Methods

The methods of the IWebNode class are listed here.

Public Methods

	Name	Description
	BeforeExecute (see page 25431)	Performs any transport-specific setup before executing an encoded method call.
	Execute (see page 25432)	This is the overview for the Execute method overload.
	GetMimeBoundary (see page 25433)	Returns the value of the MimeBoundary (see page 25433) property.
	SetMimeBoundary (see page 25433)	Sets the value of the MimeBoundary (see page 25433) property.

183.1.1.2.1 IWebNode.BeforeExecute Method

Performs any transport-specific setup before executing an encoded method call.

Pascal

```
procedure BeforeExecute(const IntfMD: TIntfMetaData; const MethMD: TIntfMethEntry;
MethodIndex: Integer; AttachHandler: IMimeAttachmentHandler);
```

C++

```
__fastcall BeforeExecute(const TIntfMetaData IntfMD, const TIntfMethEntry MethMD, int
MethodIndex, IMimeAttachmentHandler AttachHandler);
```

Description

BeforeExecute handles transport-specific tasks such as dynamically querying a WSDL document or setting up serialization options before a call is made to the Execute (see page 25432) method. Remote interfaced objects (TRio) call this method after their OnBeforeExecute event, but before delegating the execution of the method to the Execute (see page 25432) method.

IntfMD is the runtime type information (RTTI) of the invokable interface that includes the method to execute.

MethMD is describes the method that is about to be executed.

MethodIndex indicates which overload to use when methMD describes an overloaded method. 0 indicates the first overload, 1 indicates the second overload, and so on.

AttachHandler is an interface to use for handling any attachments that are used as parameters of the method.

See Also

Execute (see page 25432)

OnBeforeExecute (see page 18730)

183.1.1.2.2 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
	IWebNode.Execute (String, TStream) (see page 25432)	Executes an encoded method call and returns the encoded results in a stream.
	IWebNode.Execute (TStream) (see page 25432)	Executes an encoded method call and returns the encoded results in a stream.
	IWebNode.Execute (TStream, TStream) (see page 25432)	Executes an encoded method call and returns the encoded results in a stream.

183.1.1.2.2.1 IWebNode.Execute Method (String, TStream)

Executes an encoded method call and returns the encoded results in a stream.

Pascal

```
procedure Execute(const DataMsg: String; Response: TStream); overload; deprecated;
procedure Execute(const Request: TStream; Response: TStream); overload;
function Execute(const Request: TStream): TStream; overload;
```

C++

```
__fastcall Execute(const AnsiString DataMsg, TStream Response);
__fastcall Execute(const TStream Request, TStream Response);
__fastcall TStream Execute(const TStream Request);
```

Description

Execute interprets an encoded method call, executes it, and returns the encoded results using a stream. Typically, this method works in conjunction with an object that implements the IOPConvert interface, which marshals a method call into the DataMsg or Request string and unmarshals the return value.

DataMsg is an encoded method call. It represents the results of marshaling a method call on an invokable interface into a transportable string. The use of this syntax is not recommended.

Request is a stream from which to read the same string that is the value of the DataMsg parameter. This syntax is preferred because the stream need not store this value in memory (for example, it can be a file stream).

Response is a stream object that receives the encoded results that arise from executing the method call.

When using the third syntax, Execute returns a stream from which to read the encoded results from executing the method call. In this case, the caller is responsible for freeing the stream.

See Also

IOPConvert (see page 18122)

BeforeExecute (see page 25431)

183.1.1.2.3 IWebNode.GetMimeBoundary Method

Returns the value of the MimeBoundary (see page 25433) property.

Pascal

```
function GetMimeBoundary: string;
```

C++

```
__fastcall AnsiString GetMimeBoundary();
```

Description

GetMimeBoundary is the read access method for the MimeBoundary (see page 25433) property.

See Also

MimeBoundary (see page 25433)

183.1.1.2.4 IWebNode.SetMimeBoundary Method

Sets the value of the MimeBoundary (see page 25433) property.

Pascal

```
procedure SetMimeBoundary(Value: string);
```

C++

```
__fastcall SetMimeBoundary(AnsiString Value);
```

Description

SetMimeBoundary is the write implementation for the MimeBoundary (see page 25433) property.

Value is the new value for MimeBoundary (see page 25433).

See Also

MimeBoundary (see page 25433)

183.1.1.3 IWebNode Properties

The properties of the IWebNode class are listed here.

Public Properties

	Name	Description
	MimeBoundary (see page 25433)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

183.1.1.3.1 IWebNode.MimeBoundary Property

Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

Pascal

```
property MimeBoundary: string;
```

C++

```
__property AnsiString MimeBoundary;
```

Description

When the method call is included in a multipart form, MimeBoundary is the value of the MIME boundary header that uniquely identifies the part of the form that contains the method call. If the method call is not included in a multipart form, MimeBoundary is an empty string.

See Also

Execute (see page 25432)

184 WebReq Namespace

This is namespace WebReq.

Topics

Name	Description
Classes (see page 25436)	The following table lists classes in this documentation.
Functions (see page 25459)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 25460)	The following table lists structs, records, enums in this documentation.
Variables (see page 25461)	The following table lists variables in this documentation.

Classes

	Name	Description
	TWebModuleFactoryList (see page 25436)	
	TWebModuleList (see page 25440)	TWebModuleList lists the WebModules in a WebSnap application and provides access to their factory objects.
	TWebRequestHandler (see page 25448)	TWebRequestHandler maintains the Web modules in an application and creates request and response objects when the application receives HTTP request messages.

Functions

	Name	Description
	WebRequestHandler (see page 25459)	Returns a TWebRequestHandler (see page 25448) object.

Structs, Records, Enums

	Name	Description
	TRequestNotification (see page 25460)	This is record WebReq.TRequestNotification.

Variables

Name	Description
WebRequestHandlerProc (see page 25461)	This is variable WebReq.WebRequestHandlerProc.

184.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TWebModuleFactoryList (see page 25436)	
	TWebModuleList (see page 25440)	TWebModuleList lists the WebModules in a WebSnap application and provides access to their factory objects.
	TWebRequestHandler (see page 25448)	TWebRequestHandler maintains the Web modules in an application and creates request and response objects when the application receives HTTP request messages.

184.1.1 TWebModuleFactoryList Class

Class Hierarchy

Pascal

```
TWebModuleFactoryList = class(TObject);
```

C++

```
class TWebModuleFactoryList : public TObject;
```

File

WebReq

See Also

TWebModuleFactory

184.1.1.1 TWebModuleFactoryList Members

The following tables list the members exposed by TWebModuleFactoryList.

Protected Methods

	Name	Description
	GetItem (see page 25439)	This is GetItem, a member of class TWebModuleFactoryList.

Public Methods

	Name	Description
	AddFactory (see page 25438)	Adds a factory to the web module factory list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.

	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25437)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25438)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	AppModuleFactory (see page 25439)	
R	ItemCount (see page 25439)	Lists the number of items in the web module factory list.
R	Items (see page 25440)	

184.1.1.2 TWebModuleFactoryList.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TWebModuleFactoryList();
```

Description

Do not create instances of Create. Instead, construct descendants of Create using the new keyword, passing any arguments for the descendant object's constructor.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

Destroy (see page 22287)

new

184.1.1.3 TWebModuleFactoryList.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebModuleFactoryList();
```

Description

Do not call ~Destroy directly. Instead, use the delete keyword when destroying VCL objects.

~Destroy deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

Warning: Never destroy a component from one of its own event handlers or from the event handler of a component it owns or contains. For example, don't destroy a button, or the form that owns the button, in the button's OnClick event handler.

To destroy a form, call its Release method, which destroys the form and releases the memory allocated for it after all its event handlers and those of the components it contains have executed.

See Also

delete

184.1.1.4 TWebModuleFactoryList Methods

The methods of the TWebModuleFactoryList class are listed here.

Protected Methods

	Name	Description
	GetItem (see page 25439)	This is GetItem, a member of class TWebModuleFactoryList.

Public Methods

	Name	Description
	AddFactory (see page 25438)	Adds a factory to the web module factory list.

184.1.1.4.1 TWebModuleFactoryList.AddFactory Method

Adds a factory to the web module factory list.

Pascal

```
procedure AddFactory(AFactory: TAbstractWebModuleFactory);
```

C++

```
__fastcall AddFactory(TAbstractWebModuleFactory AFactory);
```

Description

Use the AddFactory method to add a factory to the web module factory list (more here, obviously).

See Also

AppModuleFactory (see page 25439)

184.1.1.4.2 TWebModuleFactoryList.GetItem Method

Pascal

```
function GetItem(I: Integer): TAbstractWebModuleFactory;
```

C++

```
__fastcall TAbstractWebModuleFactory GetItem(int I);
```

Description

This is GetItem, a member of class TWebModuleFactoryList.

184.1.1.5 TWebModuleFactoryList Properties

The properties of the TWebModuleFactoryList class are listed here.

Public Properties

	Name	Description
R	AppModuleFactory (see page 25439)	
R	ItemCount (see page 25439)	Lists the number of items in the web module factory list.
R	Items (see page 25440)	

184.1.1.5.1 TWebModuleFactoryList.AppModuleFactory Property

Pascal

```
property AppModuleFactory: TAbstractWebModuleFactory;
```

C++

```
__property TAbstractWebModuleFactory AppModuleFactory;
```

See Also

AddFactory (see page 25438)

184.1.1.5.2 TWebModuleFactoryList.ItemCount Property

Lists the number of items in the web module factory list.

Pascal

```
property ItemCount: Integer;
```

C++

```
__property int ItemCount;
```

Description

Use the ItemCount property to list the number of items in the web module factory list.

See Also

Items (see page 25440)

184.1.1.5.3 TWebModuleFactoryList.Items Property

Pascal

```
property Items [I: Integer]: TAbstractWebModuleFactory;
```

C++

```
__property TAbstractWebModuleFactory Items[int I];
```

See Also

AddFactory (see page 25438)

ItemCount (see page 25439)

184.1.2 TWebModuleList Class

TWebModuleList lists the WebModules in a WebSnap application and provides access to their factory objects.

Class Hierarchy

Pascal

```
TWebModuleList = class(TAbstractWebModuleList);
```

C++

```
class TWebModuleList : public TAbstractWebModuleList;
```

File

WebReq

Description

TWebModuleList contains every Web module that has been instantiated in order to serve the current request. In addition, it lists the factory objects for all Web modules in the application, including any that are not instantiated.

When a WebSnap application handles multiple requests concurrently, it includes multiple Web modules lists, each on a separate thread and each associated with a different request.

The Web context, which can be accessed using the global WebContext function, provides access to the TWebModuleList object that is associated with the current request. Use the Web context's WebModules property to obtain a reference to the Web module list.

See Also

TWebModuleFactoryList (see page 25436)

TAbstractWebModuleFactory (see page 15043)

WebContext

TAbstractWebContext (see page 24644)

184.1.2.1 TWebModuleList Members

The following tables list the members exposed by TWebModuleList.

Protected Methods

	Name	Description
	GetFactory (see page 25446)	This is GetFactory, a member of class TWebModuleList.
	GetFactoryCount (see page 25447)	
	GetItem (see page 25447)	This is GetItem, a member of class TWebModuleList.
	GetItemCount (see page 25447)	
	GetOnModuleAdded (see page 25447)	
	SetOnModuleAdded (see page 25448)	

Protected Properties

	Name	Description
	Factories (see page 25448)	This is Factories, a member of class TWebModuleList.

Public Events

	Name	Description
	OnModuleAdded (see page 15059)	Occurs when a new Web module is added to the Items (see page 15058) property array.

Public Methods

	Name	Description
	AddModule (see page 25443)	Adds a module to the Items property array.
	AddModuleClass (see page 25443)	Creates a new Web module, given its class, and adds it to the Items property array.
	AddModuleName (see page 25444)	Creates a new Web module, given its name, and adds it to the Items property array.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AutoCreateModules (see page 25445)	Instantiates the Web modules that are flagged to handle all incoming requests.
	AutoDestroyModules (see page 25445)	Frees all Web modules that are flagged as inappropriate for caching between requests.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25442)	Creates and initializes a TWebModuleList (see page 25440) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 25442)	Frees a TWebModuleList (see page 25440) instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindModuleClass (see page 25446)	Locates a Web module given its class reference.
	FindModuleName (see page 25446)	Locates a Web module given its name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Factory (see page 15057)	Lists the Web module factories for all the Web modules available in the application.
R	FactoryCount (see page 15058)	Indicates the number of Web module factories listed in the Factory (see page 15057) property array.
R	ItemCount (see page 15058)	Indicates the number of Web modules that have been instantiated in the current thread.
R	Items (see page 15058)	Lists the instantiated Web modules in the current thread.

184.1.2.2 TWebModuleList.Create Constructor

Creates and initializes a TWebModuleList (see page 25440) object.

Pascal

```
constructor Create(const AFactories: TWebModuleFactoryList);
```

C++

```
__fastcall TWebModuleList(const TWebModuleFactoryList AFactories);
```

Description

There is no need to call the TWebModuleList (see page 25440) constructor. Web module lists are created automatically for each incoming HTTP request.

AFactories is the list of Web module factories in the WebSnap application. TWebModuleList (see page 25440) uses this list to implement the Factory property.

See Also

Factory

184.1.2.3 TWebModuleList.Destroy Destructor

Frees a TWebModuleList (see page 25440) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebModuleList();
```

Description

Do not free TWebModuleList (see page 25440) objects. Web module lists are freed automatically when the WebSnap application finishes processing the HTTP request.

184.1.2.4 TWebModuleList Methods

The methods of the TWebModuleList class are listed here.

Protected Methods

	Name	Description
	GetFactory (see page 25446)	This is GetFactory, a member of class TWebModuleList.
	GetFactoryCount (see page 25447)	
	GetItem (see page 25447)	This is GetItem, a member of class TWebModuleList.
	GetItemCount (see page 25447)	
	GetOnModuleAdded (see page 25447)	
	SetOnModuleAdded (see page 25448)	

Public Methods

	Name	Description
	AddModule (see page 25443)	Adds a module to the Items property array.
	AddModuleClass (see page 25443)	Creates a new Web module, given its class, and adds it to the Items property array.
	AddModuleName (see page 25444)	Creates a new Web module, given its name, and adds it to the Items property array.
	AutoCreateModules (see page 25445)	Instantiates the Web modules that are flagged to handle all incoming requests.
	AutoDestroyModules (see page 25445)	Frees all Web modules that are flagged as inappropriate for caching between requests.
	FindModuleClass (see page 25446)	Locates a Web module given its class reference.
	FindModuleName (see page 25446)	Locates a Web module given its name.

184.1.2.4.1 TWebModuleList.AddModule Method

Adds a module to the Items property array.

Pascal

```
procedure AddModule(AComponent: TComponent);
```

C++

```
__fastcall AddModule(TComponent * AComponent);
```

Description

Call AddModule to add a newly instantiated Web module to the Items property array.

AComponent is a Web module instance that is not already listed in the Items property array. It must be one of the objects created by a Web module factory from the Factory property array.

See Also

AddModuleName (see page 25444)

AddModuleClass (see page 25443)

Items

Factory

184.1.2.4.2 TWebModuleList.AddModuleClass Method

Creates a new Web module, given its class, and adds it to the Items property array.

Pascal

```
function AddModuleClass(AClass: TComponentClass): TComponent; override;
```

C++

```
virtual __fastcall TComponent * AddModuleClass(TComponentClass AClass);
```

Description

AddModuleClass searches for a Web module factory that can create a Web module of the specified class. If it finds a match, it instantiates the Web module and adds it to the Items property array.

AClass is a class reference to the Web module you want to instantiate. It should not match the class type of any Web module already listed in the Items property array.

AddModuleClass returns the new Web module instance. If the Factory property array does not include a Web module factory that can create a Web module of the specified class, AddModuleClass returns nil (Delphi) or NULL (C++).

See Also

AddModuleName (see page 25444)

AddModule (see page 25443)

Items

Factory

FindModuleClass (see page 25446)

184.1.2.4.3 TWebModuleList.AddModuleName Method

Creates a new Web module, given its name, and adds it to the Items property array.

Pascal

```
function AddModuleName(const AName: string): TComponent; override;
```

C++

```
virtual __fastcall TComponent * AddModuleName(const AnsiString AName);
```

Description

AddModuleName searches for a Web module factory that can create a Web module of the specified name. If it finds a match, it instantiates the Web module and adds it to the Items property array.

AName is the name of the Web module you want to instantiate. It should not match the name of any Web module already listed in the Items property array.

AddModuleClass (see page 25443) returns the new Web module instance. If the Factory property array does not include a Web module factory that can create a Web module of the specified name, AddModuleClass (see page 25443) returns nil (Delphi) or NULL (C++).

See Also

AddModuleClass (see page 25443)

AddModule (see page 25443)

Items

Factory

FindModuleName (see page 25446)

184.1.2.4.4 TWebModuleList.AutoCreateModules Method

Instantiates the Web modules that are flagged to handle all incoming requests.

Pascal

```
procedure AutoCreateModules;
```

C++

```
__fastcall AutoCreateModules();
```

Description

TWebRequestHandler (see page 25448) calls AutoCreateModules to instantiate all of the Web modules that should be used to handle every incoming request object.

AutoCreateModules searches the module factory list for factory objects that have a CreateMode property with the crAlways flag. For each such factory object, AutoCreateModules creates the corresponding Web module if it does not already exist and adds it to the Items property array.

See Also

CreateMode (see page 15049)

AutoDestroyModules (see page 25445)

Items

Factory

TWebRequestHandler (see page 25448)

184.1.2.4.5 TWebModuleList.AutoDestroyModules Method

Frees all Web modules that are flagged as inappropriate for caching between requests.

Pascal

```
procedure AutoDestroyModules;
```

C++

```
__fastcall AutoDestroyModules();
```

Description

TWebRequestHandler (see page 25448) calls AutoDestroyModules when it finishes processing an HTTP request message. This allows the Web module list to free those Web modules that should not be cached for reuse with another request message.

AutoDestroyModules searches the module factory list for factory objects that have a CacheMode property with the caDestroy flag. For each such factory object, AutoDestroyModules frees the Web module if it exists and removes it from Items property array.

See Also

CacheMode (see page 15048)

AutoCreateModules (see page 25445)

Items

TWebRequestHandler (see page 25448)

Factory

184.1.2.4.6 TWebModuleList.FindModuleClass Method

Locates a Web module given its class reference.

Pascal

```
function FindModuleClass(AClass: TComponentClass): TComponent; override;
```

C++

```
virtual __fastcall TComponent * FindModuleClass(TComponentClass AClass);
```

Description

Call FindModuleClass to locate the Web module instance of a given class type. FindModuleClass searches the Items property array for a Web module instance of the specified class.

AClass is a class reference to the class type of the desired Web module.

FindModuleClass returns the requested Web module instance if it exists. If there is no Web module of the specified class, FindModuleClass returns nil (Delphi) or NULL (C++).

See Also

FindModuleName (see page 25446)

Items

184.1.2.4.7 TWebModuleList.FindModuleName Method

Locates a Web module given its name.

Pascal

```
function FindModuleName(const AName: string): TComponent; override;
```

C++

```
virtual __fastcall TComponent * FindModuleName(const AnsiString AName);
```

Description

Call FindModuleName to locate the Web module instance of a given name. FindModuleClass (see page 25446) searches the Items property array for a Web module instance of the specified name.

AName is the name of the desired Web module.

FindModuleName returns the requested Web module instance if it exists. If there is no Web module with the specified name, FindModuleClass (see page 25446) returns nil (Delphi) or NULL (C++).

See Also

FindModuleClass (see page 25446)

Items

184.1.2.4.8 TWebModuleList.GetFactory Method

Pascal

```
function GetFactory(I: Integer): TAbstractWebModuleFactory; override;
```

C++

```
virtual __fastcall TAbstractWebModuleFactory GetFactory(int I);
```

Description

This is GetFactory, a member of class TWebModuleList.

184.1.2.4.9 TWebModuleList.GetFactoryCount Method

Pascal

```
function GetFactoryCount: Integer; override;
```

C++

```
virtual __fastcall int GetFactoryCount();
```

See Also

GetFactory (see page 15055)

FactoryCount (see page 15058)

184.1.2.4.10 TWebModuleList.GetItem Method

Pascal

```
function GetItem(I: Integer): TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetItem(int I);
```

Description

This is GetItem, a member of class TWebModuleList.

184.1.2.4.11 TWebModuleList.GetItemCount Method

Pascal

```
function GetItemCount: Integer; override;
```

C++

```
virtual __fastcall int GetItemCount();
```

See Also

GetItem (see page 15056)

ItemCount (see page 15058)

184.1.2.4.12 TWebModuleList.GetOnModuleAdded Method

Pascal

```
function GetOnModuleAdded: TModuleAddedProc; override;
```

C++

```
virtual __fastcall TModuleAddedProc GetOnModuleAdded();
```

See Also

OnModuleAdded (see page 15059)

SetOnModuleAdded (see page 15057)

184.1.2.4.13 TWebModuleList.SetOnModuleAdded Method

Pascal

```
procedure SetOnModuleAdded(AProc: TModuleAddedProc); override;
```

C++

```
virtual __fastcall SetOnModuleAdded(TModuleAddedProc AProc);
```

See Also

GetOnModuleAdded (see page 15056)

OnModuleAdded (see page 15059)

184.1.2.5 TWebModuleList Properties

The properties of the TWebModuleList class are listed here.

Protected Properties

	Name	Description
	Factories (see page 25448)	This is Factories, a member of class TWebModuleList.

184.1.2.5.1 TWebModuleList.Factories Property

Pascal

```
property Factories: TWebModuleFactoryList;
```

C++

```
__property TWebModuleFactoryList Factories;
```

Description

This is Factories, a member of class TWebModuleList.

184.1.3 TWebRequestHandler Class

TWebRequestHandler maintains the Web modules in an application and creates request and response objects when the application receives HTTP request messages.

Class Hierarchy

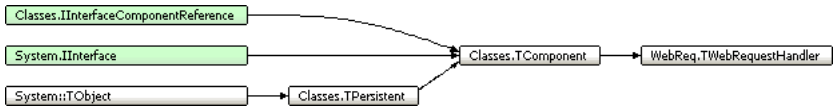

Pascal

```
TWebRequestHandler = class(TComponent);
```

C++

```
class TWebRequestHandler : public TComponent;
```

File

WebReq

Description

TWebRequestHandler keeps a pool of active Web modules. In response to a request from the application, TWebRequestHandler creates a request object and assigns it to one of the active Web modules.

Do not create instances of TWebRequestHandler. Use TWebRequestHandler as a base class when declaring nonvisual components that can appear on the component palette and be used in the form designer. Properties and methods of TWebRequestHandler provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TWebApplication (see page 24618)

TWebModule (see page 15154)

184.1.3.1 TWebRequestHandler Members

The following tables list the members exposed by TWebRequestHandler.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	ActivateWebModules (see page 25453)	Activates a Web module for the application and returns it.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	DeactivateWebModules (see page 25453)	Deactivates the Web module of the application.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	HandleRequest (see page 25454)	Responds to an HTTP request message received by the server.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.

	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddWebModuleFactory (see page 25453)	Adds a Web module factory to the request handler.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25452)	Creates an instance of TWebRequestHandler (see page 25448).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 25452)	Destroys the TWebRequestHandler (see page 25448) instance and frees its memory.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HandleException (see page 25454)	Responds to exceptions related to TWebRequestHandler (see page 25448) objects.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ActiveCount (see page 25455)	Indicates the number of active connections currently being serviced by the application.
	CacheConnections (see page 25455)	Determines whether the Web modules used by the application are cached for re-use.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	InactiveCount (see page 25456)	Indicates the number of Web modules cached for later reuse.
	MaxConnections (see page 25457)	Sets or returns the MaxConnections used by the socket.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	WebModuleClass (see page 25457)	Contains a reference to a Web module.
R	WebModuleFactory (see page 25457)	A list of Web module factory objects.
R	WebModuleFactoryCount (see page 25458)	The number of Web module factories in use by the request handler.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

184.1.3.2 TWebRequestHandler.Create Constructor

Creates an instance of TWebRequestHandler (see page 25448).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebRequestHandler(TComponent * AOwner);
```

Description

Call Create to instantiate a TWebRequestHandler (see page 25448) object.

See Also

Destroy (see page 25452)

184.1.3.3 TWebRequestHandler.Destroy Destructor

Destroys the TWebRequestHandler (see page 25448) instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebRequestHandler();
```

Description

Destroy cleans up references in the global fix-up list for form linking. It then calls the inherited destructor.

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

Free (see page 22300)

Destroy (see page 22287)

184.1.3.4 TWebRequestHandler Methods

The methods of the TWebRequestHandler class are listed here.

Protected Methods

	Name	Description
	ActivateWebModules (see page 25453)	Activates a Web module for the application and returns it.
	DeactivateWebModules (see page 25453)	Deactivates the Web module of the application.
	HandleRequest (see page 25454)	Responds to an HTTP request message received by the server.

Public Methods

	Name	Description
	AddWebModuleFactory (see page 25453)	Adds a Web module factory to the request handler.
	HandleException (see page 25454)	Responds to exceptions related to TWebRequestHandler (see page 25448) objects.

184.1.3.4.1 TWebRequestHandler.ActivateWebModules Method

Activates a Web module for the application and returns it.

Pascal

```
function ActivateWebModules: TWebModuleList;
```

C++

```
__fastcall TWebModuleList ActivateWebModules();
```

Description

ActivateWebModules is called internally when the application receives an HTTP request message. If the InactiveCount (see page 25456) property is greater than zero, a cached Web module is reactivated. Otherwise, a new instance of the Web module is created.

Before ActivateWebModules is called, the CreateForm method must be called to register the data module.

Note: In cross-platform applications, a Web application object contains only a single Web module. Usually, that Web module is a TWebModule object, but it may also be an ordinary data module that contains a TWebDispatcher object. If no Web module is associated with the Web application, ActivateWebModule raises an exception. If the ActiveCount (see page 25455) property is already at MaxConnections (see page 25457) when ActivateWebModules is called, ActivateWebModules raises an exception.

See Also

CreateForm (see page 24623)

ActiveCount (see page 25455)

InactiveCount (see page 25456)

MaxConnections (see page 25457)

184.1.3.4.2 TWebRequestHandler.AddWebModuleFactory Method

Adds a Web module factory to the request handler.

Pascal

```
procedure AddWebModuleFactory(AFactory: TAbstractWebModuleFactory);
```

C++

```
__fastcall AddWebModuleFactory(TAbstractWebModuleFactory AFactory);
```

Description

AddWebModuleFactory is used to add a Web module factory to the Web request handler. Web module factories are for internal use, and should not require any attention from developers.

184.1.3.4.3 TWebRequestHandler.DeactivateWebModules Method

Deactivates the Web module of the application.

Pascal

```
procedure DeactivateWebModules(WebModules: TWebModuleList);
```

C++

```
__fastcall DeactivateWebModules(TWebModuleList WebModules);
```

Description

Do not call DeactivateWebModules directly. The Web application calls DeactivateWebModules internally after handling an HTTP request message. When CacheConnections (see page 25455) is true, the Web module remains inactive until the next request message is received. When CacheConnections (see page 25455) is false, the runtime instance of the Web module is destroyed.

See Also

ActivateWebModules (see page 25453)

CacheConnections (see page 25455)

HandleRequest (see page 25454)

InactiveCount (see page 25456)

184.1.3.4.4 TWebRequestHandler.HandleException Method

Responds to exceptions related to TWebRequestHandler (see page 25448) objects.

Pascal

```
procedure HandleException(Sender: TObject);
```

C++

```
__fastcall HandleException(TObject * Sender);
```

Description

When exceptions occur in TWebRequestHandler (see page 25448) and descendant objects, HandleException will attempt to deal with it. If it cannot do so, it displays an error message to the user.

184.1.3.4.5 TWebRequestHandler.HandleRequest Method

Responds to an HTTP request message received by the server.

Pascal

```
function HandleRequest(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(TWebRequest Request, TWebResponse Response);
```

Description

When a descendant of TWebApplication receives an HTTP request message, it creates a TWebRequest object to represent the message, and a TWebResponse message to match the request. It then calls HandleRequest, passing those objects as the Request and Response parameters.

HandleRequest activates the Web module associated with the Web application. It then passes the request and response objects to the dispatcher, so that it can dispatch the request to the appropriate action items. When the dispatcher returns, HandleRequest checks whether the response was handled but not sent, and if not, sends it. Finally, HandleRequest deactivates the data module.

HandleRequest returns a Boolean value indicating whether the request message was handled and sent.

See Also

ActivateWebModules (see page 25453)

DeactivateWebModules (see page 25453)

184.1.3.5 TWebRequestHandler Properties

The properties of the TWebRequestHandler class are listed here.

Public Properties

	Name	Description
R	ActiveCount (see page 25455)	Indicates the number of active connections currently being serviced by the application.
	CacheConnections (see page 25455)	Determines whether the Web modules used by the application are cached for re-use.
R	InactiveCount (see page 25456)	Indicates the number of Web modules cached for later reuse.
	MaxConnections (see page 25457)	Sets or returns the MaxConnections used by the socket.
	WebModuleClass (see page 25457)	Contains a reference to a Web module.
R	WebModuleFactory (see page 25457)	A list of Web module factory objects.
R	WebModuleFactoryCount (see page 25458)	The number of Web module factories in use by the request handler.

184.1.3.5.1 TWebRequestHandler.ActiveCount Property

Indicates the number of active connections currently being serviced by the application.

Pascal

```
property ActiveCount: Integer;
```

C++

```
__property int ActiveCount;
```

Description

Use ActiveCount to monitor the number of HTTP request messages the application is servicing at any one time. Over time, this value provides an indication of the optimal value for the MaxConnections (see page 25457) property.

Each time an Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only) application receives an HTTP request message, it generates a new instance of the Web module. After the response is returned to the Web client, the Web module is either destroyed or deactivated and cached for later reuse. Each active Web module resides in a separate execution thread.

ActiveCount is the number of Web modules currently in use. InactiveCount (see page 25456) is the number of deactivated Web modules cached for later reuse. MaxConnections (see page 25457) is the maximum number of Web modules that can be active at any time. ActiveCount can never exceed MaxConnections (see page 25457).

Note: ActiveCount is only meaningful for Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only)

applications. Each CGI application handles only a single request message.

See Also

InactiveCount (see page 25456)

184.1.3.5.2 TWebRequestHandler.CacheConnections Property

Determines whether the Web modules used by the application are cached for re-use.

Pascal

```
property CacheConnections: Boolean;
```

C++

```
__property Boolean CacheConnections;
```

Description

Each time an Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only) application receives an HTTP request message, it spawns a thread to handle the request and creates a new Web module that contains instances of all the objects used in the Web application. To improve performance, these Web modules can be saved in a cache when the thread associated with the HTTP request message finishes executing.

Set CacheConnections to true to have the Web application save inactive Web modules in a cache. Caching the Web modules speeds performance as new instances of the Web module and its contents do not need to be created every time a request message is received. Set CacheConnections to false to have the Web application create a new Web module for every request thread. Recreating the Web module ensures that changes made to the module or its contents when servicing the last request do not cause unexpected behavior the next time the module is activated.

CacheConnections is only relevant to Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only) Web applications.

See Also

MaxConnections (see page 25457)

184.1.3.5.3 TWebRequestHandler.InactiveCount Property

Indicates the number of Web modules cached for later reuse.

Pascal

```
property InactiveCount: Integer;
```

C++

```
__property int InactiveCount;
```

Description

Use InactiveCount to monitor the number of unused Web modules. Over time, this value provides an indication of the optimal value for the MaxConnections (see page 25457) property.

When CacheConnections (see page 25455) is true, each time an Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only) application receives an HTTP request message, it reactivates one of the inactive Web modules. The application generates a new instance of the Web module if InactiveCount is 0. After the response is returned to the Web client, the Web module is deactivated again, and counted by the InactiveCount property.

When CacheConnections (see page 25455) is false, a new Web module is created for each HTTP request message and destroyed when the response is returned. InactiveCount is always 0 when CacheConnections (see page 25455) is false.

Note: InactiveCount is only meaningful for Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only)

applications. Each CGI application handles only a single request message, so there is no need to cache unused data modules.

See Also

ActiveCount (see page 25455)

184.1.3.5.4 TWebRequestHandler.MaxConnections Property

Sets or returns the MaxConnections used by the socket.

Pascal

```
property MaxConnections: Integer;
```

C++

```
__property int MaxConnections;
```

Description

Set MaxConnections to adjust the number of HTTP request messages the application can handle simultaneously. Use the ActiveCount (see page 25455) and InactiveCount (see page 25456) properties to get a sense of the frequency and duration of request messages. Then set MaxConnections so that it is high enough to avoid exceptions when the application runs out of Web modules, but low enough not to degrade performance too severely by running too many simultaneous threads.

If a request message is received when ActiveCount (see page 25455) is already at MaxConnections, the application raises an exception.

Note: InactiveCount (see page 25456) is only meaningful for Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only)

applications. Each CGI application handles only a single request message, so there is no need to cache unused data modules.

See Also

CacheConnections (see page 25455)

184.1.3.5.5 TWebRequestHandler.WebModuleClass Property

Contains a reference to a Web module.

Pascal

```
property WebModuleClass: TComponentClass;
```

C++

```
__property TComponentClass WebModuleClass;
```

Description

WebModuleClass is a reference to a Web module, and is for internal use only. When ActivateWebModules (see page 25453) is called, it is used in that method to help generate infrastructure for a Web module.

See Also

ActivateWebModules (see page 25453)

184.1.3.5.6 TWebRequestHandler.WebModuleFactory Property

A list of Web module factory objects.

Pascal

```
property WebModuleFactory [I: Integer]: TAbstractWebModuleFactory;
```

C++

```
__property TAbstractWebModuleFactory WebModuleFactory[int I];
```

Description

This is a list of Web module factory objects, which are intended for internal use.

See Also

WebModuleFactoryCount (see page 25458)

184.1.3.5.7 TWebRequestHandler.WebModuleFactoryCount Property

The number of Web module factories in use by the request handler.

Pascal

```
property WebModuleFactoryCount: Integer;
```

C++

```
__property int WebModuleFactoryCount;
```

Description

WebModuleFactoryCount is the number of Web module factories in use by the request handler. This property is intended for internal use.

See Also

WebModuleFactory (see page 25457)

184.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	WebRequestHandler (see page 25459)	Returns a TWebRequestHandler (see page 25448) object.

184.2.1 WebReq.WebRequestHandler Function

Returns a TWebRequestHandler (see page 25448) object.

Pascal

```
function WebRequestHandler: TWebRequestHandler;
```

C++

```
TWebRequestHandler WebRequestHandler();
```

File

WebReq

Description

Use the WebRequestHandler function to obtain a reference to the global TWebRequestHandler (see page 25448) object. This object manages all the Web modules in the application, and creates Web request and response objects when the application receives HTTP request messages.

See Also

TWebApplication

TWebModule

184.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TRequestNotification (see page 25460)	This is record WebReq.TRequestNotification.

184.3.1 WebReq.TRequestNotification Enumeration

Pascal

```
TRequestNotification = (
  rnActivateModule,
  rnDeactivateModule,
  rnCreateModule,
  rnFreeModule,
  rnStartRequest,
  rnFinishRequest
);
```

C++

```
enum TRequestNotification {
  rnActivateModule,
  rnDeactivateModule,
  rnCreateModule,
  rnFreeModule,
  rnStartRequest,
  rnFinishRequest
};
```

File

WebReq

Description

This is record WebReq.TRequestNotification.

184.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
WebRequestHandlerProc (see page 25461)	This is variable WebReq.WebRequestHandlerProc.

184.4.1 WebReq.WebRequestHandlerProc Variable

Pascal

```
WebRequestHandlerProc: function: TWebRequestHandler = nil;
```

C++

```
function: TWebRequestHandler WebRequestHandlerProc = nil;
```

File

WebReq

Description

This is variable WebReq.WebRequestHandlerProc.

185 WebScript Namespace

This is namespace WebScript.

Topics

Name	Description
Classes (see page 25465)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 25544)	The following table lists structs, records, enums in this documentation.
Types (see page 25546)	The following table lists types in this documentation.
Constants (see page 25547)	The following table lists constants in this documentation.

Classes

	Name	Description
	ERecursiveIncludeFile (see page 25465)	This is class WebScript.ERecursiveIncludeFile.
	EScriptEvalError (see page 25466)	This is class WebScript.EScriptEvalError.
	TAbstractScriptEngine (see page 25466)	This is class WebScript.TAbstractScriptEngine.
	TAbstractScriptObjectFactories (see page 25470)	This is class WebScript.TAbstractScriptObjectFactories.
	TAbstractScriptObjectFactory (see page 25473)	This is class WebScript.TAbstractScriptObjectFactory.
	TCurrentWebPageInfo (see page 25475)	This is class WebScript.TCurrentWebPageInfo.
	THTMLItem (see page 25480)	This is class WebScript.THTMLItem.
	THTMLItems (see page 25483)	This is class WebScript.THTMLItems.
	TLineNumberMap (see page 25488)	This is class WebScript.TLineNumberMap.
	TScriptCodeItem (see page 25490)	This is class WebScript.TScriptCodeItem.
	TScriptEnginesList (see page 25492)	This is class WebScript.TScriptEnginesList.
	TScriptError (see page 25497)	This is class WebScript.TScriptError.
	TScriptErrors (see page 25501)	This is class WebScript.TScriptErrors.
	TScriptFile (see page 25505)	This is class WebScript.TScriptFile.
	TScriptGlobalObjects (see page 25508)	This is class WebScript.TScriptGlobalObjects.
	TScriptIncludeFile (see page 25513)	This is class WebScript.TScriptIncludeFile.
	TScriptIncludeFiles (see page 25514)	This is class WebScript.TScriptIncludeFiles.
	TScriptIncludeItem (see page 25518)	This is class WebScript.TScriptIncludeItem.
	TScriptItem (see page 25520)	This is class WebScript.TScriptItem.
	TScriptItems (see page 25522)	This is class WebScript.TScriptItems.
	TScriptObjectFactories (see page 25526)	This is class WebScript.TScriptObjectFactories.
	TScriptProducer (see page 25530)	This is class WebScript.TScriptProducer.
	TVariableLookup (see page 25540)	This is class WebScript.TVariableLookup.

Constants

Name	Description
GlobalScriptVariableNames (see page 25547)	This is constant WebScript.GlobalScriptVariableNames.

Structs, Records, Enums

	Name	Description
	TGlobalScriptVariables (see page 25544)	This is record WebScript.TGlobalScriptVariables.
	TIncludeType (see page 25545)	This is record WebScript.TIncludeType.
	TLineNumbers (see page 25545)	This is record WebScript.TLineNumbers.

Types

Name	Description
CreateComponentStubFactory (see page 25546)	This is type WebScript.CreateComponentStubFactory.
PLineNumbers (see page 25546)	This is type WebScript.PLineNumbers.

185.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ERecursiveIncludeFile (see page 25465)	This is class WebScript.ERecursiveIncludeFile.
	EScriptEvalError (see page 25466)	This is class WebScript.EScriptEvalError.
	TAbstractScriptEngine (see page 25466)	This is class WebScript.TAbstractScriptEngine.
	TAbstractScriptObjectFactories (see page 25470)	This is class WebScript.TAbstractScriptObjectFactories.
	TAbstractScriptObjectFactory (see page 25473)	This is class WebScript.TAbstractScriptObjectFactory.
	TCurrentWebPageInfo (see page 25475)	This is class WebScript.TCurrentWebPageInfo.
	THTMLItem (see page 25480)	This is class WebScript.THTMLItem.
	THTMLItems (see page 25483)	This is class WebScript.THTMLItems.
	TLineNumberMap (see page 25488)	This is class WebScript.TLineNumberMap.
	TScriptCodeItem (see page 25490)	This is class WebScript.TScriptCodeItem.
	TScriptEnginesList (see page 25492)	This is class WebScript.TScriptEnginesList.
	TScriptError (see page 25497)	This is class WebScript.TScriptError.
	TScriptErrors (see page 25501)	This is class WebScript.TScriptErrors.
	TScriptFile (see page 25505)	This is class WebScript.TScriptFile.
	TScriptGlobalObjects (see page 25508)	This is class WebScript.TScriptGlobalObjects.
	TScriptIncludeFile (see page 25513)	This is class WebScript.TScriptIncludeFile.
	TScriptIncludeFiles (see page 25514)	This is class WebScript.TScriptIncludeFiles.
	TScriptIncludeItem (see page 25518)	This is class WebScript.TScriptIncludeItem.
	TScriptItem (see page 25520)	This is class WebScript.TScriptItem.
	TScriptItems (see page 25522)	This is class WebScript.TScriptItems.
	TScriptObjectFactories (see page 25526)	This is class WebScript.TScriptObjectFactories.
	TScriptProducer (see page 25530)	This is class WebScript.TScriptProducer.
	TVariableLookup (see page 25540)	This is class WebScript.TVariableLookup.

185.1.1 ERecursiveIncludeFile Class

Class Hierarchy

Pascal

```
ERecursiveIncludeFile = class(EScriptError);
```

C++

```
class ERecursiveIncludeFile : public EScriptError;
```

File

WebScript

Description

This is class WebScript.ERecursiveIncludeFile.

185.1.2 EScriptEvalError Class

Class Hierarchy

Pascal

```
EScriptEvalError = class(EWebBrokerException);
```

C++

```
class EScriptEvalError : public EWebBrokerException;
```

File

WebScript

Description

This is class WebScript.EScriptEvalError.

185.1.3 TAbstractScriptEngine Class

Class Hierarchy

Pascal

```
TAbstractScriptEngine = class(TObject);
```

C++

```
class TAbstractScriptEngine : public TObject;
```

File

WebScript

Description

This is class WebScript.TAbstractScriptEngine.

185.1.3.1 TAbstractScriptEngine Members

The following tables list the members exposed by TAbstractScriptEngine.

Public Methods

	Name	Description
	AddObjectFactory (see page 25468)	This is AddObjectFactory, a member of class TAbstractScriptEngine.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.

	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25467)	This is Create, a member of class TAbstractScriptEngine.
	CreateComponentStub (see page 25468)	This is CreateComponentStub, a member of class TAbstractScriptEngine.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25467)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
A	ListLanguageNames (see page 25469)	This is ListLanguageNames, a member of class TAbstractScriptEngine.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
A	SupportsLanguage (see page 25469)	This is SupportsLanguage, a member of class TAbstractScriptEngine.

Public Properties

	Name	Description
R	Name (see page 25469)	This is Name, a member of class TAbstractScriptEngine.
R	ScriptObjectFactories (see page 25470)	This is ScriptObjectFactories, a member of class TAbstractScriptEngine.
	ScriptProducerClass (see page 25470)	This is ScriptProducerClass, a member of class TAbstractScriptEngine.

185.1.3.2 TAbstractScriptEngine.Create Constructor

Pascal

```
constructor Create(const AName: string);
```

C++

```
__fastcall TAbstractScriptEngine(const AnsiString AName);
```

Description

This is Create, a member of class TAbstractScriptEngine.

185.1.3.3 TAbstractScriptEngine.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAbstractScriptEngine();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

185.1.3.4 TAbstractScriptEngine Methods

The methods of the TAbstractScriptEngine class are listed here.

Public Methods

	Name	Description
	AddObjectFactory (see page 25468)	This is AddObjectFactory, a member of class TAbstractScriptEngine.
	CreateComponentStub (see page 25468)	This is CreateComponentStub, a member of class TAbstractScriptEngine.
	ListLanguageNames (see page 25469)	This is ListLanguageNames, a member of class TAbstractScriptEngine.
	SupportsLanguage (see page 25469)	This is SupportsLanguage, a member of class TAbstractScriptEngine.

185.1.3.4.1 TAbstractScriptEngine.AddObjectFactory Method

Pascal

```
procedure AddObjectFactory(AFactory: TAbstractScriptObjectFactory);
```

C++

```
__fastcall AddObjectFactory(TAbstractScriptObjectFactory AFactory);
```

Description

This is AddObjectFactory, a member of class TAbstractScriptEngine.

185.1.3.4.2 TAbstractScriptEngine.CreateComponentStub Method

Pascal

```
function CreateComponentStub(AObject: TObject): IInterface;
```

C++

```
__fastcall IInterface CreateComponentStub(TObject * AObject);
```

Description

This is CreateComponentStub, a member of class TAbstractScriptEngine.

185.1.3.4.3 TAbstractScriptEngine.ListLanguageNames Method

Pascal

```
procedure ListLanguageNames(AList: TStrings); virtual; abstract;
```

C++

```
virtual __fastcall ListLanguageNames(TStrings AList) = 0;
```

Description

This is ListLanguageNames, a member of class TAbstractScriptEngine.

185.1.3.4.4 TAbstractScriptEngine.SupportsLanguage Method

Pascal

```
function SupportsLanguage(const AName: string): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean SupportsLanguage(const AnsiString AName) = 0;
```

Description

This is SupportsLanguage, a member of class TAbstractScriptEngine.

185.1.3.5 TAbstractScriptEngine Properties

The properties of the TAbstractScriptEngine class are listed here.

Public Properties

	Name	Description
R	Name (see page 25469)	This is Name, a member of class TAbstractScriptEngine.
R	ScriptObjectFactories (see page 25470)	This is ScriptObjectFactories, a member of class TAbstractScriptEngine.
	ScriptProducerClass (see page 25470)	This is ScriptProducerClass, a member of class TAbstractScriptEngine.

185.1.3.5.1 TAbstractScriptEngine.Name Property

Pascal

```
property Name: string;
```

C++

```
__property AnsiString Name;
```

Description

This is Name, a member of class TAbstractScriptEngine.

185.1.3.5.2 TAbstractScriptEngine.ScriptObjectFactories Property

Pascal

```
property ScriptObjectFactories: TAbstractScriptObjectFactories;
```

C++

```
__property TAbstractScriptObjectFactories ScriptObjectFactories;
```

Description

This is ScriptObjectFactories, a member of class TAbstractScriptEngine.

185.1.3.5.3 TAbstractScriptEngine.ScriptProducerClass Property

Pascal

```
property ScriptProducerClass: TScriptProducerClass;
```

C++

```
__property TScriptProducerClass ScriptProducerClass;
```

Description

This is ScriptProducerClass, a member of class TAbstractScriptEngine.

185.1.4 TAbstractScriptObjectFactories Class

Class Hierarchy

Pascal

```
TAbstractScriptObjectFactories = class(TObject);
```

C++

```
class TAbstractScriptObjectFactories : public TObject;
```

File

WebScript

Description

This is class WebScript.TAbstractScriptObjectFactories.

185.1.4.1 TAbstractScriptObjectFactories Members

The following tables list the members exposed by TAbstractScriptObjectFactories.

Protected Methods

	Name	Description
	GetCount (see page 25472)	This is GetCount, a member of class TAbstractScriptObjectFactories.
	GetFactory (see page 25472)	This is GetFactory, a member of class TAbstractScriptObjectFactories.
	GetObjectList (see page 25472)	This is GetObjectList, a member of class TAbstractScriptObjectFactories.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Count (see page 25472)	This is Count, a member of class TAbstractScriptObjectFactories.
R	Factories (see page 25473)	This is Factories, a member of class TAbstractScriptObjectFactories.
R	ObjectList (see page 25473)	This is ObjectList, a member of class TAbstractScriptObjectFactories.

185.1.4.2 TAbstractScriptObjectFactories Methods

The methods of the TAbstractScriptObjectFactories class are listed here.

Protected Methods

	Name	Description
A	GetCount (see page 25472)	This is GetCount, a member of class TAbstractScriptObjectFactories.
A	GetFactory (see page 25472)	This is GetFactory, a member of class TAbstractScriptObjectFactories.
A	GetObjectList (see page 25472)	This is GetObjectList, a member of class TAbstractScriptObjectFactories.

185.1.4.2.1 TAbstractScriptObjectFactories.GetCount Method

Pascal

```
function GetCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetCount() = 0;
```

Description

This is GetCount, a member of class TAbstractScriptObjectFactories.

185.1.4.2.2 TAbstractScriptObjectFactories.GetFactory Method

Pascal

```
function GetFactory(I: Integer): TAbstractScriptObjectFactory; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractScriptObjectFactory GetFactory(int I) = 0;
```

Description

This is GetFactory, a member of class TAbstractScriptObjectFactories.

185.1.4.2.3 TAbstractScriptObjectFactories.GetObjectList Method

Pascal

```
function GetObjectList: TObjectList; virtual; abstract;
```

C++

```
virtual __fastcall TObjectList GetObjectList() = 0;
```

Description

This is GetObjectList, a member of class TAbstractScriptObjectFactories.

185.1.4.3 TAbstractScriptObjectFactories Properties

The properties of the TAbstractScriptObjectFactories class are listed here.

Public Properties

	Name	Description
R	Count (see page 25472)	This is Count, a member of class TAbstractScriptObjectFactories.
R	Factories (see page 25473)	This is Factories, a member of class TAbstractScriptObjectFactories.
R	ObjectList (see page 25473)	This is ObjectList, a member of class TAbstractScriptObjectFactories.

185.1.4.3.1 TAbstractScriptObjectFactories.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TAbstractScriptObjectFactories.

185.1.4.3.2 TAbstractScriptObjectFactories.Factories Property

Pascal

```
property Factories [I: Integer]: TAbstractScriptObjectFactory;
```

C++

```
__property TAbstractScriptObjectFactory Factories[int I];
```

Description

This is Factories, a member of class TAbstractScriptObjectFactories.

185.1.4.3.3 TAbstractScriptObjectFactories.ObjectList Property

Pascal

```
property ObjectList: TObjectList;
```

C++

```
__property TObjectList ObjectList;
```

Description

This is ObjectList, a member of class TAbstractScriptObjectFactories.

185.1.5 TAbstractScriptObjectFactory Class

Class Hierarchy

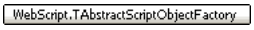

Pascal

```
TAbstractScriptObjectFactory = class;
```

C++

```
class TAbstractScriptObjectFactory;
```

File

WebScript

Description

This is class WebScript.TAbstractScriptObjectFactory.

185.1.5.1 TAbstractScriptObjectFactory Members

The following tables list the members exposed by TAbstractScriptObjectFactory.

Protected Methods

	Name	Description
	AddGlobalObjects (see page 25474)	This is AddGlobalObjects, a member of class TAbstractScriptObjectFactory.
	CreateComponentObject (see page 25474)	This is CreateComponentObject, a member of class TAbstractScriptObjectFactory.
	CreateProducerObject (see page 25475)	This is CreateProducerObject, a member of class TAbstractScriptObjectFactory.
	GetScriptEngine (see page 25475)	This is GetScriptEngine, a member of class TAbstractScriptObjectFactory.

Public Methods

	Name	Description
	CreateGlobalObject (see page 25475)	This is CreateGlobalObject, a member of class TAbstractScriptObjectFactory.

185.1.5.2 TAbstractScriptObjectFactory Methods

The methods of the TAbstractScriptObjectFactory class are listed here.

Protected Methods

	Name	Description
	AddGlobalObjects (see page 25474)	This is AddGlobalObjects, a member of class TAbstractScriptObjectFactory.
	CreateComponentObject (see page 25474)	This is CreateComponentObject, a member of class TAbstractScriptObjectFactory.
	CreateProducerObject (see page 25475)	This is CreateProducerObject, a member of class TAbstractScriptObjectFactory.
	GetScriptEngine (see page 25475)	This is GetScriptEngine, a member of class TAbstractScriptObjectFactory.

Public Methods

	Name	Description
	CreateGlobalObject (see page 25475)	This is CreateGlobalObject, a member of class TAbstractScriptObjectFactory.

185.1.5.2.1 TAbstractScriptObjectFactory.AddGlobalObjects Method

Pascal

```
procedure AddGlobalObjects(AScriptGlobalObjects: TScriptGlobalObjects); virtual;
```

C++

```
virtual __fastcall AddGlobalObjects(TScriptGlobalObjects AScriptGlobalObjects);
```

Description

This is AddGlobalObjects, a member of class TAbstractScriptObjectFactory.

185.1.5.2.2 TAbstractScriptObjectFactory.CreateComponentObject Method

Pascal

```
function CreateComponentObject(AComponent: TObject; AClassKey: TClass): IInterface; virtual;
abstract;
```

C++

```
virtual __fastcall IInterface CreateComponentObject(TObject * AComponent, TClass AClassKey) =
0;
```

Description

This is CreateComponentObject, a member of class TAbstractScriptObjectFactory.

185.1.5.2.3 TAbstractScriptObjectFactory.CreateGlobalObject Method

Pascal

```
function CreateGlobalObject(AVariable: TGlobalScriptVariables; AContext: IScriptContext):
IInterface; virtual; abstract;
```

C++

```
virtual __fastcall IInterface CreateGlobalObject(TGlobalScriptVariables AVariable,
IScriptContext AContext) = 0;
```

Description

This is CreateGlobalObject, a member of class TAbstractScriptObjectFactory.

185.1.5.2.4 TAbstractScriptObjectFactory.CreateProducerObject Method

Pascal

```
function CreateProducerObject(AVariable: TGlobalScriptVariables; AProducer: IScriptProducer):
IInterface; virtual; abstract;
```

C++

```
virtual __fastcall IInterface CreateProducerObject(TGlobalScriptVariables AVariable,
IScriptProducer AProducer) = 0;
```

Description

This is CreateProducerObject, a member of class TAbstractScriptObjectFactory.

185.1.5.2.5 TAbstractScriptObjectFactory.GetScriptEngine Method

Pascal

```
function GetScriptEngine: TAbstractScriptEngine; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractScriptEngine GetScriptEngine() = 0;
```

Description

This is GetScriptEngine, a member of class TAbstractScriptObjectFactory.

185.1.6 TCurrentWebPageInfo Class

Class Hierarchy

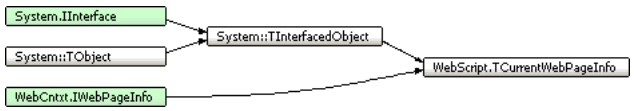

Pascal

```
TCurrentWebPageInfo = class(TInterfacedObject, IWebPageInfo);
```

C++

```
class TCurrentWebPageInfo : public TInterfacedObject, public IWebPageInfo;
```

File

WebScript

Description

This is class WebScript.TCurrentWebPageInfo.

185.1.6.1 TCurrentWebPageInfo Members

The following tables list the members exposed by TCurrentWebPageInfo.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetHasViewAccess (see page 25478)	Returns the value of the HasViewAccess property.
	GetLoginRequired (see page 25478)	Returns the value of the LoginRequired property.
	GetPageDescription (see page 25478)	Returns the value of the PageDescription property.
	GetPageHREF (see page 25479)	Returns the value of the PageHREF property.
	GetPageIsPublished (see page 25479)	Returns the value of the PageIsPublished property.
	GetPageName (see page 25479)	Returns the value of the PageName property.
	GetPageTitle (see page 25479)	Returns the value of the PageTitle property.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	WebPageInfo (see page 25480)	This is WebPageInfo, a member of class TCurrentWebPageInfo.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25477)	This is Create, a member of class TCurrentWebPageInfo.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.

	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	HasViewAccess (see page 24684)	Indicates whether the current user has access rights to view the Web page.
R	LoginRequired (see page 24684)	Indicates whether callers must log in before accessing this Web page.
R	PageDescription (see page 24684)	Describes the Web page.
R	PageHREF (see page 24685)	Identifies the HREF to use for references to the associated Web page.
R	PageIsPublished (see page 24685)	Indicates whether the WebSnap application automatically displays the Web page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 24686).
R	PageName (see page 24686)	Denotes the logical name for the Web page.
R	PageTitle (see page 24686)	Indicates the title that appears on the Web page.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

185.1.6.2 TCurrentWebPageInfo.Create Constructor

Pascal

```
constructor Create(AContext: IScriptContext);
```

C++

```
__fastcall TCurrentWebPageInfo(IScriptContext AContext);
```

Description

This is Create, a member of class TCurrentWebPageInfo.

185.1.6.3 TCurrentWebPageInfo Methods

The methods of the TCurrentWebPageInfo class are listed here.

Protected Methods

	Name	Description
	GetHasViewAccess (see page 25478)	Returns the value of the HasViewAccess property.
	GetLoginRequired (see page 25478)	Returns the value of the LoginRequired property.
	GetPageDescription (see page 25478)	Returns the value of the PageDescription property.
	GetPageHREF (see page 25479)	Returns the value of the PageHREF property.
	GetPageIsPublished (see page 25479)	Returns the value of the PageIsPublished property.

	GetPageName (see page 25479)	Returns the value of the PageName property.
	GetPageTitle (see page 25479)	Returns the value of the PageTitle property.
	WebPageInfo (see page 25480)	This is WebPageInfo, a member of class TCurrentWebPageInfo.

185.1.6.3.1 TCurrentWebPageInfo.GetHasViewAccess Method

Returns the value of the HasViewAccess property.

Pascal

```
function GetHasViewAccess: Boolean;
```

C++

```
__fastcall Boolean GetHasViewAccess();
```

Description

GetHasViewAccess is the read implementation of the HasViewAccess property.

See Also

HasViewAccess (see page 24684)

185.1.6.3.2 TCurrentWebPageInfo.GetLoginRequired Method

Returns the value of the LoginRequired property.

Pascal

```
function GetLoginRequired: Boolean;
```

C++

```
__fastcall Boolean GetLoginRequired();
```

Description

GetLoginRequired is the read implementation of the LoginRequired property.

See Also

LoginRequired (see page 24684)

185.1.6.3.3 TCurrentWebPageInfo.GetPageDescription Method

Returns the value of the PageDescription property.

Pascal

```
function GetPageDescription: string;
```

C++

```
__fastcall AnsiString GetPageDescription();
```

Description

GetPageDescription is the read implementation of the PageDescription property.

See Also

PageDescription (see page 24684)

185.1.6.3.4 TCurrentWebPageInfo.GetPageHREF Method

Returns the value of the PageHREF property.

Pascal

```
function GetPageHREF: string;
```

C++

```
__fastcall AnsiString GetPageHREF();
```

Description

GetPageHREF is the read implementation of the PageHREF property.

See Also

PageHREF (see page 24685)

185.1.6.3.5 TCurrentWebPageInfo.GetPageIsPublished Method

Returns the value of the PageIsPublished property.

Pascal

```
function GetPageIsPublished: Boolean;
```

C++

```
__fastcall Boolean GetPageIsPublished();
```

Description

GetPageIsPublished is the read implementation of the PageIsPublished property.

See Also

PageIsPublished (see page 24685)

185.1.6.3.6 TCurrentWebPageInfo.GetPageName Method

Returns the value of the PageName property.

Pascal

```
function GetPageName: string;
```

C++

```
__fastcall AnsiString GetPageName();
```

Description

GetPageName is the read implementation of the PageName property.

See Also

PageName (see page 24686)

185.1.6.3.7 TCurrentWebPageInfo.GetPageTitle Method

Returns the value of the PageTitle property.

Pascal

```
function GetPageTitle: string;
```

C++

```
__fastcall AnsiString GetPageTitle();
```

Description

GetPageTitle is the read implementation of the PageTitle property.

See Also

PageTitle (see page 24686)

185.1.6.3.8 TCurrentWebPageInfo.WebPageInfo Method

Pascal

```
function WebPageInfo: IWebPageInfo;
```

C++

```
__fastcall IWebPageInfo WebPageInfo();
```

Description

This is WebPageInfo, a member of class TCurrentWebPageInfo.

185.1.7 THTMLItem Class

Class Hierarchy

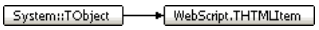

Pascal

```
THTMLItem = class(TObject);
```

C++

```
class THTMLItem : public TObject;
```

File

WebScript

Description

This is class WebScript.THTMLItem.

185.1.7.1 THTMLItem Members

The following tables list the members exposed by THTMLItem.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AssignString (see page 25482)	This is AssignString, a member of class THTMLItem.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25481)	This is Create, a member of class THTMLItem.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	HTML (see page 25482)	This is HTML, a member of class THTMLItem.
R	Ident (see page 25482)	This is Ident, a member of class THTMLItem.
R	Length (see page 25483)	This is Length, a member of class THTMLItem.
R	Offset (see page 25483)	This is Offset, a member of class THTMLItem.
	TagToken (see page 25483)	This is TagToken, a member of class THTMLItem.

185.1.7.2 THTMLItem.Create Constructor

Pascal

```
constructor Create(AItems: THTMLItems);
```

C++

```
__fastcall THTMLItem(THTMLItems AItems);
```

Description

This is Create, a member of class THTMLItem.

185.1.7.3 THTMLItem Methods

The methods of the THTMLItem class are listed here.

Public Methods

	Name	Description
	AssignString (see page 25482)	This is AssignString, a member of class THTMLItem.

185.1.7.3.1 THTMLItem.AssignString Method

Pascal

```
procedure AssignString(AOffset: Integer; AHTML: PChar; ALength: Integer);
```

C++

```
__fastcall AssignString(int AOffset, const char * AHTML, int ALength);
```

Description

This is AssignString, a member of class THTMLItem.

185.1.7.4 THTMLItem Properties

The properties of the THTMLItem class are listed here.

Public Properties

	Name	Description
R	HTML (see page 25482)	This is HTML, a member of class THTMLItem.
R	Ident (see page 25482)	This is Ident, a member of class THTMLItem.
R	Length (see page 25483)	This is Length, a member of class THTMLItem.
R	Offset (see page 25483)	This is Offset, a member of class THTMLItem.
	TagToken (see page 25483)	This is TagToken, a member of class THTMLItem.

185.1.7.4.1 THTMLItem.HTML Property

Pascal

```
property HTML: string;
```

C++

```
__property AnsiString HTML;
```

Description

This is HTML, a member of class THTMLItem.

185.1.7.4.2 THTMLItem.Ident Property

Pascal

```
property Ident: string;
```

C++

```
__property AnsiString Ident;
```

Description

This is Ident, a member of class THTMLItem.

185.1.7.4.3 THTMLItem.Length Property

Pascal

```
property Length: Integer;
```

C++

```
__property int Length;
```

Description

This is Length, a member of class THTMLItem.

185.1.7.4.4 THTMLItem.Offset Property

Pascal

```
property Offset: Integer;
```

C++

```
__property int Offset;
```

Description

This is Offset, a member of class THTMLItem.

185.1.7.4.5 THTMLItem.TagToken Property

Pascal

```
property TagToken: Boolean;
```

C++

```
__property Boolean TagToken;
```

Description

This is TagToken, a member of class THTMLItem.

185.1.8 THTMLItems Class

Class Hierarchy

Pascal

```
THTMLItems = class(TObjectList);
```

C++

```
class THTMLItems : public TObjectList;
```

File

WebScript

Description

This is class WebScript.THTMLItems.

185.1.8.1 THTMLItems Members

The following tables list the members exposed by THTMLItems.

Protected Methods

	Name	Description
	Get (see page 2878)	This is Get, a member of class TList.
	GetItem (see page 5682)	This is GetItem, a member of class TObjectList.
V	Grow (see page 2879)	This is Grow, a member of class TList.
V	Notify (see page 2884)	Responds when items are added, removed, or extracted from the list.
V	Notify (see page 5684)	Responds when items are added, removed, or extracted from the list.
	Put (see page 2887)	This is Put, a member of class TList.
	SetCapacity (see page 2891)	This is SetCapacity, a member of class TList.
	SetCount (see page 2892)	This is SetCount, a member of class TList.
	SetItem (see page 5685)	This is SetItem, a member of class TObjectList.

Public Methods

	Name	Description
	Add (see page 2871)	Inserts a new item at the end of the list.
	Add (see page 5680)	Inserts an object at the end of the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2874)	Copies elements of one list to another.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 2874)	Deletes all items from the list.
	Create (see page 25485)	This is Create, a member of class THTMLItems.
	Create (see page 5679)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2874)	Removes the item at the position given by the Index parameter.
V	Destroy (see page 25485)	Destroys an instance of TList.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Error (see page 2875)	This is the overview for the Error method overload.
	Exchange (see page 2876)	Swaps the position of two items in the Items (see page 2898) array.
	Expand (see page 2877)	Increases the Capacity (see page 2895) of the list.
	Extract (see page 2877)	Removes a specified item from the list.
	Extract (see page 5681)	Removes a specified object from the list without freeing it.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindInstanceOf (see page 5682)	Finds the first instance of a specified class in the list.
	First (see page 2878)	Returns Items (see page 2898)[0].
	First (see page 5682)	Returns Items (see page 5685)[0].
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2878)	Returns a TList (see page 2868) enumerator.
	GetHTMLItem (see page 25486)	This is GetHTMLItem, a member of class THTMLItems.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	IndexOf (see page 2879)	Returns the index of the first entry in the Items (see page 2898) array with a specified value.
	IndexOf (see page 5683)	Returns the index of the first object in the list with a specified value.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2882)	Adds an object to the Items (see page 2898) array at the position specified by Index.
	Insert (see page 5683)	Adds an object to the list at a specified position.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Last (see page 2882)	Returns Items (see page 2898)[Count (see page 2896) - 1].
	Last (see page 5684)	Returns the object at the end of the Items (see page 5685) array.
	MakeIdentifier (see page 25486)	This is MakeIdentifier, a member of class THTMLItems.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	Move (see page 2884)	Changes the position of an item in the Items (see page 2898) array.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Pack (see page 2885)	Deletes all nil (Delphi) or NULL (C++) items from the Items (see page 2898) array.
	Prepare (see page 25486)	This is Prepare, a member of class THTMLItems.
	Remove (see page 2887)	Deletes the first reference to the Item parameter from the Items (see page 2898) array.
	Remove (see page 5684)	Removes a specified object from the list and (if OwnsObjects (see page 5686) is true) frees the object.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Sort (see page 2892)	Performs a QuickSort on the list based on the comparison function Compare.

Public Properties

	Name	Description
	Capacity (see page 2895)	Specifies the allocated size of the array of pointers maintained by the TList (see page 2868) object.
	Count (see page 2896)	Indicates the number of entries in the list that are in use.
R	Items (see page 25487)	Lists the object references.
R	List (see page 2900)	Specifies the array of pointers that make up the Items (see page 2898) array.
	OwnsObjects (see page 5686)	Allows TObjectList (see page 5677) to free objects when they are deleted from the list or the list is destroyed.

185.1.8.2 THTMLItems.Create Constructor

Pascal

```
constructor Create(AProducer: TScriptProducer);
```

C++

```
__fastcall THTMLItems(TScriptProducer AProducer);
```

Description

This is Create, a member of class THTMLItems.

185.1.8.3 THTMLItems.Destroy Destructor

Destroys an instance of TList.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THTMLItems();
```

Description

Do not call Destroy directly. Instead, call Free. Free verifies that the TList reference is not nil, and only then calls Destroy.

Destroy frees the memory used to store the list of items.

Note: Destroy does not free the memory pointed to by the elements of the list.

See Also

Free (see page 22300)

185.1.8.4 THTMLItems Methods

The methods of the THTMLItems class are listed here.

Public Methods

	Name	Description
	GetHTMLItem (see page 25486)	This is GetHTMLItem, a member of class THTMLItems.
	MakeIdentifier (see page 25486)	This is MakeIdentifier, a member of class THTMLItems.
	Prepare (see page 25486)	This is Prepare, a member of class THTMLItems.

185.1.8.4.1 THTMLItems.GetHTMLItem Method

Pascal

```
function GetHTMLItem(I: Integer): THTMLItem;
```

C++

```
__fastcall THTMLItem GetHTMLItem(int I);
```

Description

This is GetHTMLItem, a member of class THTMLItems.

185.1.8.4.2 THTMLItems.MakeIdentifier Method

Pascal

```
function MakeIdentifier(AItem: THTMLItem): string;
```

C++

```
__fastcall AnsiString MakeIdentifier(THTMLItem AItem);
```

Description

This is MakeIdentifier, a member of class THTMLItems.

185.1.8.4.3 THTMLItems.Prepare Method

Pascal

```
procedure Prepare;
```

C++

```
__fastcall Prepare();
```

Description

This is Prepare, a member of class THTMLItems.

185.1.8.5 THTMLItems Properties

The properties of the THTMLItems class are listed here.

Public Properties

	Name	Description
R	Items (see page 25487)	Lists the object references.

185.1.8.5.1 THTMLItems.Items Property

Lists the object references.

Pascal

```
property Items [Index: Integer]: THTMLItem;
```

C++

```
__property THTMLItem Items[int Index];
```

Description

Use Items to access objects in the list. Items is a zero-based array: The first object is indexed as 0, the second object is indexed as 1, and so forth. You can read or change the value at a specific index, or use Items with the Count property to iterate through the list.

If OwnsObjects is true, reassigning an Items index frees the object that previously occupied that position in the list.

Items can contain nil (Delphi) or NULL (C++) references. To remove nil (Delphi) or NULL (C++) references and reduce the size of the array, call the Pack method.

Note: In Delphi, Items is the default property of TObjectList. This means you can omit the property name. Thus, the following two lines of code are both acceptable and do the same thing:

Note: ObjectList1.Items[i] := Form1.FindChildControl(ListBox1.Items[i]);

Note: ObjectList1[i] := Form1.FindChildControl(ListBox1.Items[i]);

See Also

OwnsObjects (see page 5686)

Add (see page 5680)

Insert (see page 5683)

Delete (see page 2874)

Remove (see page 5684)

List (see page 2900)

Extract (see page 5681)

Count (see page 2896)

Clear (see page 2874)

Pack (see page 2885)

operator_sb

185.1.9 TLineNumberMap Class

Class Hierarchy

Pascal

```
TLineNumberMap = class(TObject);
```

C++

```
class TLineNumberMap : public TObject;
```

File

WebScript

Description

This is class WebScript.TLineNumberMap.

185.1.9.1 TLineNumberMap Members

The following tables list the members exposed by TLineNumberMap.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	DestLineToSourceLine (see page 25489)	This is DestLineToSourceLine, a member of class TLineNumberMap.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	New (see page 25489)	This is New, a member of class TLineNumberMap.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Count (see page 25490)	This is Count, a member of class TLineNumberMap.
R	Items (see page 25490)	This is Items, a member of class TLineNumberMap.

185.1.9.2 TLineNumberMap Methods

The methods of the TLineNumberMap class are listed here.

Public Methods

	Name	Description
	DestLineToSourceLine (see page 25489)	This is DestLineToSourceLine, a member of class TLineNumberMap.
	New (see page 25489)	This is New, a member of class TLineNumberMap.

185.1.9.2.1 TLineNumberMap.DestLineToSourceLine Method

Pascal

```
function DestLineToSourceLine(ALine: Integer; var AFileName: string): Integer;
```

C++

```
__fastcall int DestLineToSourceLine(int ALine, AnsiString AFileName);
```

Description

This is DestLineToSourceLine, a member of class TLineNumberMap.

185.1.9.2.2 TLineNumberMap.New Method

Pascal

```
function New: PLineNumbers;
```

C++

```
__fastcall PLineNumbers New();
```

Description

This is New, a member of class TLineNumberMap.

185.1.9.3 TLineNumberMap Properties

The properties of the TLineNumberMap class are listed here.

Public Properties

	Name	Description
R	Count (see page 25490)	This is Count, a member of class TLineNumberMap.
R	Items (see page 25490)	This is Items, a member of class TLineNumberMap.

185.1.9.3.1 TLineNumberMap.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TLineNumberMap.

185.1.9.3.2 TLineNumberMap.Items Property

Pascal

```
property Items [I: Integer]: PLineNumbers;
```

C++

```
__property PLineNumbers Items[int I];
```

Description

This is Items, a member of class TLineNumberMap.

185.1.10 TScriptCodeItem Class

Class Hierarchy

Pascal

```
TScriptCodeItem = class(TScriptItem);
```

C++

```
class TScriptCodeItem : public TScriptItem;
```

File

WebScript

Description

This is class WebScript.TScriptCodeItem.

185.1.10.1 TScriptCodeItem Members

The following tables list the members exposed by TScriptCodeItem.

Protected Methods

	Name	Description
	GetScript (see page 25492)	This is GetScript, a member of class TScriptCodeItem.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	AssignString (see page 25492)	This is AssignString, a member of class TScriptCodeItem.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 25521)	This is Create, a member of class TScriptItem.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Script (see page 25522)	This is Script, a member of class TScriptItem.

185.1.10.2 TScriptCodeItem Methods

The methods of the TScriptCodeItem class are listed here.

Protected Methods

	Name	Description
	GetScript (see page 25492)	This is GetScript, a member of class TScriptCodeItem.

Public Methods

	Name	Description
	AssignString (see page 25492)	This is AssignString, a member of class TScriptCodeItem.

185.1.10.2.1 TScriptCodeItem.AssignString Method

Pascal

```
procedure AssignString(const AValue: string);
```

C++

```
__fastcall AssignString(const AnsiString AValue);
```

Description

This is AssignString, a member of class TScriptCodeItem.

185.1.10.2.2 TScriptCodeItem.GetScript Method

Pascal

```
function GetScript: string; override;
```

C++

```
virtual __fastcall AnsiString GetScript();
```

Description

This is GetScript, a member of class TScriptCodeItem.

185.1.11 TScriptEnginesList Class

Class Hierarchy

System::TObject → HTTPProd.TAbstractScriptEnginesList → WebScript.TScriptEnginesList

Pascal

```
TScriptEnginesList = class(TAbstractScriptEnginesList);
```

C++

```
class TScriptEnginesList : public TAbstractScriptEnginesList;
```

File

WebScript

Description

This is class WebScript.TScriptEnginesList.

185.1.11.1 TScriptEnginesList Members

The following tables list the members exposed by TScriptEnginesList.

Public Methods

	Name	Description
	AddEngine (see page 25495)	This is AddEngine, a member of class TScriptEnginesList.
	AddObjectFactory (see page 25495)	This is AddObjectFactory, a member of class TScriptEnginesList.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25493)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25494)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Find (see page 25495)	This is Find, a member of class TScriptEnginesList.
V	FindScriptProducerClass (see page 25495)	This is FindScriptProducerClass, a member of class TScriptEnginesList.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	ListLanguageNames (see page 25496)	This is ListLanguageNames, a member of class TScriptEnginesList.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RemoveEngine (see page 25496)	This is RemoveEngine, a member of class TScriptEnginesList.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Count (see page 25496)	This is Count, a member of class TScriptEnginesList.
R	Items (see page 25496)	This is Items, a member of class TScriptEnginesList.

185.1.11.2 TScriptEnginesList.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TScriptEnginesList();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

185.1.11.3 TScriptEnginesList.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScriptEnginesList();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

185.1.11.4 TScriptEnginesList Methods

The methods of the TScriptEnginesList class are listed here.

Public Methods

	Name	Description
	AddEngine (see page 25495)	This is AddEngine, a member of class TScriptEnginesList.
	AddObjectFactory (see page 25495)	This is AddObjectFactory, a member of class TScriptEnginesList.
	Find (see page 25495)	This is Find, a member of class TScriptEnginesList.
V	FindScriptProducerClass (see page 25495)	This is FindScriptProducerClass, a member of class TScriptEnginesList.
	ListLanguageNames (see page 25496)	This is ListLanguageNames, a member of class TScriptEnginesList.
	RemoveEngine (see page 25496)	This is RemoveEngine, a member of class TScriptEnginesList.

185.1.11.4.1 TScriptEnginesList.AddEngine Method

Pascal

```
procedure AddEngine(AEngine: TAbstractScriptEngine);
```

C++

```
__fastcall AddEngine(TAbstractScriptEngine AEngine);
```

Description

This is AddEngine, a member of class TScriptEnginesList.

185.1.11.4.2 TScriptEnginesList.AddObjectFactory Method

Pascal

```
procedure AddObjectFactory(AFactory: TAbstractScriptObjectFactory; const AEngineName: string);
```

C++

```
__fastcall AddObjectFactory(TAbstractScriptObjectFactory AFactory, const AnsiString
AEngineName);
```

Description

This is AddObjectFactory, a member of class TScriptEnginesList.

185.1.11.4.3 TScriptEnginesList.Find Method

Pascal

```
function Find(const AName: string): TAbstractScriptEngine;
```

C++

```
__fastcall TAbstractScriptEngine Find(const AnsiString AName);
```

Description

This is Find, a member of class TScriptEnginesList.

185.1.11.4.4 TScriptEnginesList.FindScriptProducerClass Method

Pascal

```
function FindScriptProducerClass(const ALanguageName: string): TScriptProducerClass; override;
```

C++

```
virtual __fastcall TScriptProducerClass FindScriptProducerClass(const AnsiString
ALanguageName);
```

Description

This is FindScriptProducerClass, a member of class TScriptEnginesList.

185.1.11.4.5 TScriptEnginesList.ListLanguageNames Method

Pascal

```
procedure ListLanguageNames(AList: TStrings);
```

C++

```
__fastcall ListLanguageNames(TStrings AList);
```

Description

This is ListLanguageNames, a member of class TScriptEnginesList.

185.1.11.4.6 TScriptEnginesList.RemoveEngine Method

Pascal

```
procedure RemoveEngine(AEngine: TAbstractScriptEngine);
```

C++

```
__fastcall RemoveEngine(TAbstractScriptEngine AEngine);
```

Description

This is RemoveEngine, a member of class TScriptEnginesList.

185.1.11.5 TScriptEnginesList Properties

The properties of the TScriptEnginesList class are listed here.

Public Properties

	Name	Description
R	Count (see page 25496)	This is Count, a member of class TScriptEnginesList.
R	Items (see page 25496)	This is Items, a member of class TScriptEnginesList.

185.1.11.5.1 TScriptEnginesList.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TScriptEnginesList.

185.1.11.5.2 TScriptEnginesList.Items Property

Pascal

```
property Items [I: Integer]: TAbstractScriptEngine;
```

C++

```
__property TAbstractScriptEngine Items[int I];
```

Description

This is Items, a member of class TScriptEnginesList.

185.1.12 TScriptError Class

Class Hierarchy

System::TObject → HTTPProd.TAbstractScriptError → WebScript.TScriptError

Pascal

```
TScriptError = class(TAbstractScriptError);
```

C++

```
class TScriptError : public TAbstractScriptError;
```

File

WebScript

Description

This is class WebScript.TScriptError.

185.1.12.1 TScriptError Members

The following tables list the members exposed by TScriptError.

Protected Fields

	Name	Description
	FCharPos (see page 25499)	This is FCharPos, a member of class TScriptError.
	FDescription (see page 25499)	This is FDescription, a member of class TScriptError.
	FFileName (see page 25499)	This is FFileName, a member of class TScriptError.
	FLine (see page 25499)	This is FLine, a member of class TScriptError.
	FSourceLine (see page 25500)	This is FSourceLine, a member of class TScriptError.

Protected Methods

	Name	Description
V	GetCharPos (see page 25500)	This is GetCharPos, a member of class TScriptError.
V	GetDescription (see page 25500)	This is GetDescription, a member of class TScriptError.
V	GetFileName (see page 25500)	This is GetFileName, a member of class TScriptError.
V	GetLine (see page 25501)	This is GetLine, a member of class TScriptError.
V	GetSourceLine (see page 25501)	This is GetSourceLine, a member of class TScriptError.
V	SetDescription (see page 25501)	This is SetDescription, a member of class TScriptError.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25498)	This is Create, a member of class TScriptError.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	CharPos (see page 15263)	This is CharPos, a member of class TAbstractScriptError.
	Description (see page 15263)	This is Description, a member of class TAbstractScriptError.
R	FileName (see page 15264)	This is FileName, a member of class TAbstractScriptError.
R	Line (see page 15264)	This is Line, a member of class TAbstractScriptError.
R	SourceLine (see page 15264)	This is SourceLine, a member of class TAbstractScriptError.

185.1.12.2 TScriptError.Create Constructor

Pascal

```
constructor Create(E: Exception); overload;
```

C++

```
__fastcall TScriptError(Exception E);
```

Description

This is Create, a member of class TScriptError.

185.1.12.3 TScriptError Fields

The fields of the TScriptError class are listed here.

Protected Fields

	Name	Description
	FCharPos (see page 25499)	This is FCharPos, a member of class TScriptError.
	FDescription (see page 25499)	This is FDescription, a member of class TScriptError.
	FFileName (see page 25499)	This is FFileName, a member of class TScriptError.
	FLine (see page 25499)	This is FLine, a member of class TScriptError.
	FSourceLine (see page 25500)	This is FSourceLine, a member of class TScriptError.

185.1.12.3.1 TScriptError.FCharPos Field

Pascal

```
FCharPos: Integer;
```

C++

```
int FCharPos;
```

Description

This is FCharPos, a member of class TScriptError.

185.1.12.3.2 TScriptError.FDescription Field

Pascal

```
FDescription: string;
```

C++

```
AnsiString FDescription;
```

Description

This is FDescription, a member of class TScriptError.

185.1.12.3.3 TScriptError.FFileName Field

Pascal

```
FFileName: string;
```

C++

```
AnsiString FFileName;
```

Description

This is FFileName, a member of class TScriptError.

185.1.12.3.4 TScriptError.FLine Field

Pascal

```
FLine: Integer;
```

C++

```
int FLine;
```

Description

This is FLine, a member of class TScriptError.

185.1.12.3.5 TScriptError.FSourceLine Field

Pascal

```
FSourceLine: string;
```

C++

```
AnsiString FSourceLine;
```

Description

This is FSourceLine, a member of class TScriptError.

185.1.12.4 TScriptError Methods

The methods of the TScriptError class are listed here.

Protected Methods

	Name	Description
	GetCharPos (see page 25500)	This is GetCharPos, a member of class TScriptError.
	GetDescription (see page 25500)	This is GetDescription, a member of class TScriptError.
	GetFileName (see page 25500)	This is GetFileName, a member of class TScriptError.
	GetLine (see page 25501)	This is GetLine, a member of class TScriptError.
	GetSourceLine (see page 25501)	This is GetSourceLine, a member of class TScriptError.
	SetDescription (see page 25501)	This is SetDescription, a member of class TScriptError.

185.1.12.4.1 TScriptError.GetCharPos Method

Pascal

```
function GetCharPos: Integer; override;
```

C++

```
virtual __fastcall int GetCharPos();
```

Description

This is GetCharPos, a member of class TScriptError.

185.1.12.4.2 TScriptError.GetDescription Method

Pascal

```
function GetDescription: string; override;
```

C++

```
virtual __fastcall AnsiString GetDescription();
```

Description

This is GetDescription, a member of class TScriptError.

185.1.12.4.3 TScriptError.GetFileName Method

Pascal

```
function GetFileName: string; override;
```

C++

```
virtual __fastcall AnsiString GetFileName();
```

Description

This is GetFileName, a member of class TScriptError.

185.1.12.4.4 TScriptError.GetLine Method

Pascal

```
function GetLine: Integer; override;
```

C++

```
virtual __fastcall int GetLine();
```

Description

This is GetLine, a member of class TScriptError.

185.1.12.4.5 TScriptError.GetSourceLine Method

Pascal

```
function GetSourceLine: string; override;
```

C++

```
virtual __fastcall AnsiString GetSourceLine();
```

Description

This is GetSourceLine, a member of class TScriptError.

185.1.12.4.6 TScriptError.SetDescription Method

Pascal

```
procedure SetDescription(const AValue: string); override;
```

C++

```
virtual __fastcall SetDescription(const AnsiString AValue);
```

Description

This is SetDescription, a member of class TScriptError.

185.1.13 TScriptErrors Class

Class Hierarchy

Pascal

```
TScriptErrors = class(TAbstractScriptErrors);
```

C++

```
class TScriptErrors : public TAbstractScriptErrors;
```

File

WebScript

Description

This is class WebScript.TScriptErrors.

185.1.13.1 TScriptErrors Members

The following tables list the members exposed by TScriptErrors.

Protected Methods

	Name	Description
	GetError (see page 25504)	This is GetError, a member of class TScriptErrors.
	GetErrorCount (see page 25504)	This is GetErrorCount, a member of class TScriptErrors.

Public Methods

	Name	Description
	Add (see page 25504)	This is Add, a member of class TScriptErrors.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25503)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 25503)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Count (see page 15267)	This is Count, a member of class TAbstractScriptErrors.

	Errors (see page 15267)	This is Errors, a member of class TAbstractScriptErrors.

185.1.13.2 TScriptErrors.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TScriptErrors();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

185.1.13.3 TScriptErrors.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScriptErrors();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

185.1.13.4 TScriptErrors Methods

The methods of the TScriptErrors class are listed here.

Protected Methods

	Name	Description
	GetError (see page 25504)	This is GetError, a member of class TScriptErrors.
	GetErrorCount (see page 25504)	This is GetErrorCount, a member of class TScriptErrors.

Public Methods

	Name	Description
	Add (see page 25504)	This is Add, a member of class TScriptErrors.

185.1.13.4.1 TScriptErrors.Add Method

Pascal

```
procedure Add(const AError: TAbstractScriptError); override;
```

C++

```
virtual __fastcall Add(const TAbstractScriptError AError);
```

Description

This is Add, a member of class TScriptErrors.

185.1.13.4.2 TScriptErrors.GetError Method

Pascal

```
function GetError(I: Integer): TAbstractScriptError; override;
```

C++

```
virtual __fastcall TAbstractScriptError GetError(int I);
```

Description

This is GetError, a member of class TScriptErrors.

185.1.13.4.3 TScriptErrors.GetErrorCount Method

Pascal

```
function GetErrorCount: Integer; override;
```

C++

```
virtual __fastcall int GetErrorCount();
```

Description

This is GetErrorCount, a member of class TScriptErrors.

185.1.14 TScriptFile Class

Class Hierarchy

WebScript.TScriptFile

Pascal

```
TScriptFile = class;
```

C++

```
class TScriptFile;
```

File

WebScript

Description

This is class WebScript.TScriptFile.

185.1.14.1 TScriptFile Members

The following tables list the members exposed by TScriptFile.

Public Fields

	Name	Description
	FLineNumberMap (see page 25506)	This is FLineNumberMap, a member of class TScriptFile.

Public Methods

	Name	Description
	Create (see page 25505)	This is Create, a member of class TScriptFile.
V	Destroy (see page 25506)	This is Destroy, a member of class TScriptFile.
	ParseStream (see page 25506)	This is ParseStream, a member of class TScriptFile.
	ParseString (see page 25507)	This is ParseString, a member of class TScriptFile.

Public Properties

	Name	Description
R	ApplicationName (see page 25507)	This is ApplicationName, a member of class TScriptFile.
R	LongName (see page 25507)	This is LongName, a member of class TScriptFile.
R	Script (see page 25508)	This is Script, a member of class TScriptFile.
R	ShortName (see page 25508)	This is ShortName, a member of class TScriptFile.
R	Source (see page 25508)	This is Source, a member of class TScriptFile.

185.1.14.2 TScriptFile.Create Constructor

Pascal

```
constructor Create(AProducer: TScriptProducer);
```

C++

```
__fastcall TScriptFile(TScriptProducer AProducer);
```

Description

This is Create, a member of class TScriptFile.

185.1.14.3 TScriptFile.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScriptFile();
```

Description

This is Destroy, a member of class TScriptFile.

185.1.14.4 TScriptFile Fields

The fields of the TScriptFile class are listed here.

Public Fields

	Name	Description
	FLineNumberMap (see page 25506)	This is FLineNumberMap, a member of class TScriptFile.

185.1.14.4.1 TScriptFile.FLineNumberMap Field

Pascal

```
FLineNumberMap: TLineNumberMap;
```

C++

```
TLineNumberMap FLineNumberMap;
```

Description

This is FLineNumberMap, a member of class TScriptFile.

185.1.14.5 TScriptFile Methods

The methods of the TScriptFile class are listed here.

Public Methods

	Name	Description
	ParseStream (see page 25506)	This is ParseStream, a member of class TScriptFile.
	ParseString (see page 25507)	This is ParseString, a member of class TScriptFile.

185.1.14.5.1 TScriptFile.ParseStream Method

Pascal

```
procedure ParseStream(ASource: TStream; AOwned: Boolean);
```

C++

```
__fastcall ParseStream(TStream ASource, Boolean AOwned);
```

Description

This is ParseStream, a member of class TScriptFile.

185.1.14.5.2 TScriptFile.ParseString Method

Pascal

```
procedure ParseString(const ASource: string);
```

C++

```
__fastcall ParseString(const AnsiString ASource);
```

Description

This is ParseString, a member of class TScriptFile.

185.1.14.6 TScriptFile Properties

The properties of the TScriptFile class are listed here.

Public Properties

	Name	Description
R	ApplicationName (see page 25507)	This is ApplicationName, a member of class TScriptFile.
R	LongName (see page 25507)	This is LongName, a member of class TScriptFile.
R	Script (see page 25508)	This is Script, a member of class TScriptFile.
R	ShortName (see page 25508)	This is ShortName, a member of class TScriptFile.
R	Source (see page 25508)	This is Source, a member of class TScriptFile.

185.1.14.6.1 TScriptFile.ApplicationName Property

Pascal

```
property ApplicationName: string;
```

C++

```
__property AnsiString ApplicationName;
```

Description

This is ApplicationName, a member of class TScriptFile.

185.1.14.6.2 TScriptFile.LongName Property

Pascal

```
property LongName: string;
```

C++

```
__property AnsiString LongName;
```

Description

This is LongName, a member of class TScriptFile.

185.1.14.6.3 TScriptFile.Script Property

Pascal

```
property Script: string;
```

C++

```
__property AnsiString Script;
```

Description

This is Script, a member of class TScriptFile.

185.1.14.6.4 TScriptFile.ShortName Property

Pascal

```
property ShortName: string;
```

C++

```
__property AnsiString ShortName;
```

Description

This is ShortName, a member of class TScriptFile.

185.1.14.6.5 TScriptFile.Source Property

Pascal

```
property Source: string;
```

C++

```
__property AnsiString Source;
```

Description

This is Source, a member of class TScriptFile.

185.1.15 TScriptGlobalObjects Class

Class Hierarchy

Pascal

```
TScriptGlobalObjects = class(TObject);
```

C++

```
class TScriptGlobalObjects : public TObject;
```

File

WebScript

Description

This is class WebScript.TScriptGlobalObjects.

185.1.15.1 TScriptGlobalObjects Members

The following tables list the members exposed by TScriptGlobalObjects.

Protected Methods

	Name	Description
	GetScriptEngine (see page 25512)	This is GetScriptEngine, a member of class TScriptGlobalObjects.

Public Methods

	Name	Description
	AddNamedIntf (see page 25511)	This is AddNamedIntf, a member of class TScriptGlobalObjects.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25510)	This is Create, a member of class TScriptGlobalObjects.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 25510)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindNamedItemIntf (see page 25511)	This is FindNamedItemIntf, a member of class TScriptGlobalObjects.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamedItemCount (see page 25511)	This is GetNamedItemCount, a member of class TScriptGlobalObjects.
	GetNamedItemName (see page 25511)	This is GetNamedItemName, a member of class TScriptGlobalObjects.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	NamedItemCount (see page 25512)	This is NamedItemCount, a member of class TScriptGlobalObjects.
	NamedItemName (see page 25512)	This is NamedItemName, a member of class TScriptGlobalObjects.
	Producer (see page 25512)	This is Producer, a member of class TScriptGlobalObjects.
	ScriptEngine (see page 25513)	This is ScriptEngine, a member of class TScriptGlobalObjects.

185.1.15.2 TScriptGlobalObjects.Create Constructor

Pascal

```
constructor Create(AProducer: IScriptProducer);
```

C++

```
__fastcall TScriptGlobalObjects(IScriptProducer AProducer);
```

Description

This is Create, a member of class TScriptGlobalObjects.

185.1.15.3 TScriptGlobalObjects.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScriptGlobalObjects();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

185.1.15.4 TScriptGlobalObjects Methods

The methods of the TScriptGlobalObjects class are listed here.

Protected Methods

	Name	Description
	GetScriptEngine (see page 25512)	This is GetScriptEngine, a member of class TScriptGlobalObjects.

Public Methods

	Name	Description
	AddNamedIntf (see page 25511)	This is AddNamedIntf, a member of class TScriptGlobalObjects.
	FindNamedItemIntf (see page 25511)	This is FindNamedItemIntf, a member of class TScriptGlobalObjects.
	GetNamedItemCount (see page 25511)	This is GetNamedItemCount, a member of class TScriptGlobalObjects.
	GetNamedItemName (see page 25511)	This is GetNamedItemName, a member of class TScriptGlobalObjects.

185.1.15.4.1 TScriptGlobalObjects.AddNamedIntf Method

Pascal

```
procedure AddNamedIntf(const AName: string; AIntf: IUnknown);
```

C++

```
__fastcall AddNamedIntf(const AnsiString AName, IUnknown AIntf);
```

Description

This is AddNamedIntf, a member of class TScriptGlobalObjects.

185.1.15.4.2 TScriptGlobalObjects.FindNamedItemIntf Method

Pascal

```
function FindNamedItemIntf(const AName: string): IUnknown;
```

C++

```
__fastcall IUnknown FindNamedItemIntf(const AnsiString AName);
```

Description

This is FindNamedItemIntf, a member of class TScriptGlobalObjects.

185.1.15.4.3 TScriptGlobalObjects.GetNamedItemCount Method

Pascal

```
function GetNamedItemCount: Integer;
```

C++

```
__fastcall int GetNamedItemCount();
```

Description

This is GetNamedItemCount, a member of class TScriptGlobalObjects.

185.1.15.4.4 TScriptGlobalObjects.GetNamedItemName Method

Pascal

```
function GetNamedItemName(I: Integer): string;
```

C++

```
__fastcall AnsiString GetNamedItemName(int I);
```

Description

This is GetNamedItemName, a member of class TScriptGlobalObjects.

185.1.15.4.5 TScriptGlobalObjects.GetScriptEngine Method

Pascal

```
function GetScriptEngine: TAbstractScriptEngine; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractScriptEngine GetScriptEngine() = 0;
```

Description

This is GetScriptEngine, a member of class TScriptGlobalObjects.

185.1.15.5 TScriptGlobalObjects Properties

The properties of the TScriptGlobalObjects class are listed here.

Public Properties

	Name	Description
R	NamedItemCount (see page 25512)	This is NamedItemCount, a member of class TScriptGlobalObjects.
R	NamedItemName (see page 25512)	This is NamedItemName, a member of class TScriptGlobalObjects.
R	Producer (see page 25512)	This is Producer, a member of class TScriptGlobalObjects.
R	ScriptEngine (see page 25513)	This is ScriptEngine, a member of class TScriptGlobalObjects.

185.1.15.5.1 TScriptGlobalObjects.NamedItemCount Property

Pascal

```
property NamedItemCount: Integer;
```

C++

```
__property int NamedItemCount;
```

Description

This is NamedItemCount, a member of class TScriptGlobalObjects.

185.1.15.5.2 TScriptGlobalObjects.NamedItemName Property

Pascal

```
property NamedItemName [I: Integer]: string;
```

C++

```
__property AnsiString NamedItemName[int I];
```

Description

This is NamedItemName, a member of class TScriptGlobalObjects.

185.1.15.5.3 TScriptGlobalObjects.Producer Property

Pascal

```
property Producer: IScriptProducer;
```

C++

```
__property IScriptProducer Producer;
```

Description

This is Producer, a member of class TScriptGlobalObjects.

185.1.15.5.4 TScriptGlobalObjects.ScriptEngine Property

Pascal

```
property ScriptEngine: TAbstractScriptEngine;
```

C++

```
__property TAbstractScriptEngine ScriptEngine;
```

Description

This is ScriptEngine, a member of class TScriptGlobalObjects.

185.1.16 TScriptIncludeFile Class

Class Hierarchy

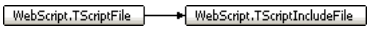

Pascal

```
TScriptIncludeFile = class(TScriptFile);
```

C++

```
class TScriptIncludeFile : public TScriptFile;
```

File

WebScript

Description

This is class WebScript.TScriptIncludeFile.

185.1.16.1 TScriptIncludeFile Members

The following tables list the members exposed by TScriptIncludeFile.

Public Fields

	Name	Description
	FLineNumberMap (see page 25506)	This is FLineNumberMap, a member of class TScriptFile.

Public Methods

	Name	Description
	Create (see page 25514)	This is Create, a member of class TScriptIncludeFile.
V	Destroy (see page 25506)	This is Destroy, a member of class TScriptFile.
	LoadFile (see page 25514)	This is LoadFile, a member of class TScriptIncludeFile.
	ParseStream (see page 25506)	This is ParseStream, a member of class TScriptFile.
	ParseString (see page 25507)	This is ParseString, a member of class TScriptFile.

Public Properties

	Name	Description
R	ApplicationName (see page 25507)	This is ApplicationName, a member of class TScriptFile.
R	LongName (see page 25507)	This is LongName, a member of class TScriptFile.
R	Script (see page 25508)	This is Script, a member of class TScriptFile.
R	ShortName (see page 25508)	This is ShortName, a member of class TScriptFile.
R	Source (see page 25508)	This is Source, a member of class TScriptFile.

185.1.16.2 TScriptIncludeFile.Create Constructor

Pascal

```
constructor Create(AProducer: TScriptProducer; AIncludeType: TIncludeType; const AFileName:
string);
```

C++

```
__fastcall TScriptIncludeFile(TScriptProducer AProducer, TIncludeType AIncludeType, const
AnsiString AFileName);
```

Description

This is Create, a member of class TScriptIncludeFile.

185.1.16.3 TScriptIncludeFile Methods

The methods of the TScriptIncludeFile class are listed here.

Public Methods

	Name	Description
	LoadFile (see page 25514)	This is LoadFile, a member of class TScriptIncludeFile.

185.1.16.3.1 TScriptIncludeFile.LoadFile Method

Pascal

```
procedure LoadFile;
```

C++

```
__fastcall LoadFile();
```

Description

This is LoadFile, a member of class TScriptIncludeFile.

185.1.17 TScriptIncludeFiles Class

Class Hierarchy

Pascal

```
TScriptIncludeFiles = class(TObjectList);
```

C++

```
class TScriptIncludeFiles : public TObjectList;
```

File

WebScript

Description

This is class WebScript.TScriptIncludeFiles.

185.1.17.1 TScriptIncludeFiles Members

The following tables list the members exposed by TScriptIncludeFiles.

Protected Methods

	Name	Description
	Get (see page 2878)	This is Get, a member of class TList.
	GetItem (see page 5682)	This is GetItem, a member of class TObjectList.
V	Grow (see page 2879)	This is Grow, a member of class TList.
V	Notify (see page 2884)	Responds when items are added, removed, or extracted from the list.
V	Notify (see page 5684)	Responds when items are added, removed, or extracted from the list.
	Put (see page 2887)	This is Put, a member of class TList.
	SetCapacity (see page 2891)	This is SetCapacity, a member of class TList.
	SetCount (see page 2892)	This is SetCount, a member of class TList.
	SetItem (see page 5685)	This is SetItem, a member of class TObjectList.

Public Methods

	Name	Description
	Add (see page 2871)	Inserts a new item at the end of the list.
	Add (see page 5680)	Inserts an object at the end of the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2874)	Copies elements of one list to another.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 2874)	Deletes all items from the list.
	Create (see page 25516)	This is Create, a member of class TScriptIncludeFiles.
	Create (see page 5679)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2874)	Removes the item at the position given by the Index parameter.
V	Destroy (see page 2870)	Destroys an instance of TList (see page 2868).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Error (see page 2875)	This is the overview for the Error method overload.
	Exchange (see page 2876)	Swaps the position of two items in the Items (see page 2898) array.
	Expand (see page 2877)	Increases the Capacity (see page 2895) of the list.
	Extract (see page 2877)	Removes a specified item from the list.

	Extract (see page 5681)	Removes a specified object from the list without freeing it.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindInstanceOf (see page 5682)	Finds the first instance of a specified class in the list.
	First (see page 2878)	Returns Items (see page 2898)[0].
	First (see page 5682)	Returns Items (see page 5685)[0].
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2878)	Returns a TList (see page 2868) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetScriptIncludeFile (see page 25517)	This is GetScriptIncludeFile, a member of class TScriptIncludeFiles.
	IndexOf (see page 2879)	Returns the index of the first entry in the Items (see page 2898) array with a specified value.
	IndexOf (see page 5683)	Returns the index of the first object in the list with a specified value.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2882)	Adds an object to the Items (see page 2898) array at the position specified by Index.
	Insert (see page 5683)	Adds an object to the list at a specified position.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Last (see page 2882)	Returns Items (see page 2898)[Count (see page 2896) - 1].
	Last (see page 5684)	Returns the object at the end of the Items (see page 5685) array.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	Move (see page 2884)	Changes the position of an item in the Items (see page 2898) array.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Pack (see page 2885)	Deletes all nil (Delphi) or NULL (C++) items from the Items (see page 2898) array.
	Remove (see page 2887)	Deletes the first reference to the Item parameter from the Items (see page 2898) array.
	Remove (see page 5684)	Removes a specified object from the list and (if OwnsObjects (see page 5686) is true) frees the object.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Sort (see page 2892)	Performs a QuickSort on the list based on the comparison function Compare.

Public Properties

	Name	Description
	Capacity (see page 2895)	Specifies the allocated size of the array of pointers maintained by the TList (see page 2868) object.
	Count (see page 2896)	Indicates the number of entries in the list that are in use.
R	Items (see page 25517)	Lists the object references.
R	List (see page 2900)	Specifies the array of pointers that make up the Items (see page 2898) array.
	OwnsObjects (see page 5686)	Allows TObjectList (see page 5677) to free objects when they are deleted from the list or the list is destroyed.

185.1.17.2 TScriptIncludeFiles.Create Constructor

Pascal

```
constructor Create(AProducer: TScriptProducer);
```

C++

```
__fastcall TScriptIncludeFiles(TScriptProducer AProducer);
```

Description

This is Create, a member of class TScriptIncludeFiles.

185.1.17.3 TScriptIncludeFiles Methods

The methods of the TScriptIncludeFiles class are listed here.

Public Methods

	Name	Description
	GetScriptIncludeFile (see page 25517)	This is GetScriptIncludeFile, a member of class TScriptIncludeFiles.

185.1.17.3.1 TScriptIncludeFiles.GetScriptIncludeFile Method

Pascal

```
function GetScriptIncludeFile(I: Integer): TScriptIncludeFile;
```

C++

```
__fastcall TScriptIncludeFile GetScriptIncludeFile(int I);
```

Description

This is GetScriptIncludeFile, a member of class TScriptIncludeFiles.

185.1.17.4 TScriptIncludeFiles Properties

The properties of the TScriptIncludeFiles class are listed here.

Public Properties

	Name	Description
R	Items (see page 25517)	Lists the object references.

185.1.17.4.1 TScriptIncludeFiles.Items Property

Lists the object references.

Pascal

```
property Items [Index: Integer]: TScriptIncludeFile;
```

C++

```
__property TScriptIncludeFile Items[int Index];
```

Description

Use Items to access objects in the list. Items is a zero-based array: The first object is indexed as 0, the second object is indexed as 1, and so forth. You can read or change the value at a specific index, or use Items with the Count property to iterate through the list.

If OwnsObjects is true, reassigning an Items index frees the object that previously occupied that position in the list.

Items can contain nil (Delphi) or NULL (C++) references. To remove nil (Delphi) or NULL (C++) references and reduce the size of the array, call the Pack method.

Note: In Delphi, Items is the default property of TObjectList. This means you can omit the property name. Thus, the following

two lines of code are both acceptable and do the same thing:

Note: ObjectList1.Items[i] := Form1.FindChildControl(ListBox1.Items[i]);

Note: ObjectList1[i] := Form1.FindChildControl(ListBox1.Items[i]);

See Also

OwnsObjects (see page 5686)

Add (see page 5680)

Insert (see page 5683)

Delete (see page 2874)

Remove (see page 5684)

List (see page 2900)

Extract (see page 5681)

Count (see page 2896)

Clear (see page 2874)

Pack (see page 2885)

operator_sb

185.1.18 TScriptIncludeItem Class

Class Hierarchy

Pascal

```
TScriptIncludeItem = class(TScriptItem);
```

C++

```
class TScriptIncludeItem : public TScriptItem;
```

File

WebScript

Description

This is class WebScript.TScriptIncludeItem.

185.1.18.1 TScriptIncludeItem Members

The following tables list the members exposed by TScriptIncludeItem.

Protected Methods

	Name	Description
	GetScript (see page 25520)	This is GetScript, a member of class TScriptIncludeItem.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25519)	This is Create, a member of class TScriptIncludeItem.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Script (see page 25522)	This is Script, a member of class TScriptItem.

185.1.18.2 TScriptIncludeItem.Create Constructor

Pascal

```
constructor Create(AItems: TScriptItems; AIncludeFile: TScriptIncludeFile);
```

C++

```
__fastcall TScriptIncludeItem(TScriptItems AItems, TScriptIncludeFile AIncludeFile);
```

Description

This is Create, a member of class TScriptIncludeItem.

185.1.18.3 TScriptIncludeItem Methods

The methods of the TScriptIncludeItem class are listed here.

Protected Methods

	Name	Description
	GetScript (see page 25520)	This is GetScript, a member of class TScriptIncludeItem.

185.1.18.3.1 TScriptIncludeItem.GetScript Method

Pascal

```
function GetScript: string; override;
```

C++

```
virtual __fastcall AnsiString GetScript();
```

Description

This is GetScript, a member of class TScriptIncludeItem.

185.1.19 TScriptItem Class

Class Hierarchy

Pascal

```
TScriptItem = class(TObject);
```

C++

```
class TScriptItem : public TObject;
```

File

WebScript

Description

This is class WebScript.TScriptItem.

185.1.19.1 TScriptItem Members

The following tables list the members exposed by TScriptItem.

Protected Methods

	Name	Description
	GetScript (see page 25522)	This is GetScript, a member of class TScriptItem.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25521)	This is Create, a member of class TScriptItem.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Script (see page 25522)	This is Script, a member of class TScriptItem.

185.1.19.2 TScriptItem.Create Constructor

Pascal

```
constructor Create(AItems: TScriptItems);
```

C++

```
__fastcall TScriptItem(TScriptItems AItems);
```

Description

This is Create, a member of class TScriptItem.

185.1.19.3 TScriptItem Methods

The methods of the TScriptItem class are listed here.

Protected Methods

	Name	Description
V	GetScript (see page 25522)	This is GetScript, a member of class TScriptItem.

185.1.19.3.1 TScriptItem.GetScript Method

Pascal

```
function GetScript: string; virtual;
```

C++

```
virtual __fastcall AnsiString GetScript();
```

Description

This is GetScript, a member of class TScriptItem.

185.1.19.4 TScriptItem Properties

The properties of the TScriptItem class are listed here.

Public Properties

	Name	Description
	Script (see page 25522)	This is Script, a member of class TScriptItem.

185.1.19.4.1 TScriptItem.Script Property

Pascal

```
property Script: string;
```

C++

```
__property AnsiString Script;
```

Description

This is Script, a member of class TScriptItem.

185.1.20 TScriptItems Class

Class Hierarchy

Pascal

```
TScriptItems = class(TObjectList);
```

C++

```
class TScriptItems : public TObjectList;
```

File

WebScript

Description

This is class WebScript.TScriptItems.

185.1.20.1 TScriptItems Members

The following tables list the members exposed by TScriptItems.

Protected Methods

	Name	Description
	Get (see page 2878)	This is Get, a member of class TList.
	GetItem (see page 5682)	This is GetItem, a member of class TObjectList.
V	Grow (see page 2879)	This is Grow, a member of class TList.
V	Notify (see page 2884)	Responds when items are added, removed, or extracted from the list.
V	Notify (see page 5684)	Responds when items are added, removed, or extracted from the list.
	Put (see page 2887)	This is Put, a member of class TList.
	SetCapacity (see page 2891)	This is SetCapacity, a member of class TList.
	SetCount (see page 2892)	This is SetCount, a member of class TList.
	SetItem (see page 5685)	This is SetItem, a member of class TObjectList.

Public Methods

	Name	Description
	Add (see page 2871)	Inserts a new item at the end of the list.
	Add (see page 5680)	Inserts an object at the end of the list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2874)	Copies elements of one list to another.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Clear (see page 2874)	Deletes all items from the list.
	Create (see page 25524)	This is Create, a member of class TScriptItems.
	Create (see page 5679)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2874)	Removes the item at the position given by the Index parameter.
V	Destroy (see page 2870)	Destroys an instance of TList (see page 2868).
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Error (see page 2875)	This is the overview for the Error method overload.
	Exchange (see page 2876)	Swaps the position of two items in the Items (see page 2898) array.
	Expand (see page 2877)	Increases the Capacity (see page 2895) of the list.
	Extract (see page 2877)	Removes a specified item from the list.
	Extract (see page 5681)	Removes a specified object from the list without freeing it.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindInstanceOf (see page 5682)	Finds the first instance of a specified class in the list.
	First (see page 2878)	Returns Items (see page 2898)[0].
	First (see page 5682)	Returns Items (see page 5685)[0].
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2878)	Returns a TList (see page 2868) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.

	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetScriptItem (see page 25525)	This is GetScriptItem, a member of class TScriptItems.
	IndexOf (see page 2879)	Returns the index of the first entry in the Items (see page 2898) array with a specified value.
	IndexOf (see page 5683)	Returns the index of the first object in the list with a specified value.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2882)	Adds an object to the Items (see page 2898) array at the position specified by Index.
	Insert (see page 5683)	Adds an object to the list at a specified position.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	Last (see page 2882)	Returns Items (see page 2898)[Count (see page 2896) - 1].
	Last (see page 5684)	Returns the object at the end of the Items (see page 5685) array.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	Move (see page 2884)	Changes the position of an item in the Items (see page 2898) array.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Pack (see page 2885)	Deletes all nil (Delphi) or NULL (C++) items from the Items (see page 2898) array.
	Remove (see page 2887)	Deletes the first reference to the Item parameter from the Items (see page 2898) array.
	Remove (see page 5684)	Removes a specified object from the list and (if OwnsObjects (see page 5686) is true) frees the object.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Sort (see page 2892)	Performs a QuickSort on the list based on the comparison function Compare.

Public Properties

	Name	Description
	Capacity (see page 2895)	Specifies the allocated size of the array of pointers maintained by the TList (see page 2868) object.
	Count (see page 2896)	Indicates the number of entries in the list that are in use.
R	Items (see page 25525)	Lists the object references.
R	List (see page 2900)	Specifies the array of pointers that make up the Items (see page 2898) array.
	OwnsObjects (see page 5686)	Allows TObjectList (see page 5677) to free objects when they are deleted from the list or the list is destroyed.

185.1.20.2 TScriptItems.Create Constructor

Pascal

```
constructor Create(AProducer: TScriptProducer);
```

C++

```
__fastcall TScriptItems(TScriptProducer AProducer);
```

Description

This is Create, a member of class TScriptItems.

185.1.20.3 TScriptItems Methods

The methods of the TScriptItems class are listed here.

Public Methods

	Name	Description
	GetScriptItem (see page 25525)	This is GetScriptItem, a member of class TScriptItems.

185.1.20.3.1 TScriptItems.GetScriptItem Method

Pascal

```
function GetScriptItem(I: Integer): TScriptItem;
```

C++

```
__fastcall TScriptItem GetScriptItem(int I);
```

Description

This is GetScriptItem, a member of class TScriptItems.

185.1.20.4 TScriptItems Properties

The properties of the TScriptItems class are listed here.

Public Properties

	Name	Description
R	Items (see page 25525)	Lists the object references.

185.1.20.4.1 TScriptItems.Items Property

Lists the object references.

Pascal

```
property Items [Index: Integer]: TScriptItem;
```

C++

```
__property TScriptItem Items[int Index];
```

Description

Use Items to access objects in the list. Items is a zero-based array: The first object is indexed as 0, the second object is indexed as 1, and so forth. You can read or change the value at a specific index, or use Items with the Count property to iterate through the list.

If OwnsObjects is true, reassigning an Items index frees the object that previously occupied that position in the list.

Items can contain nil (Delphi) or NULL (C++) references. To remove nil (Delphi) or NULL (C++) references and reduce the size of the array, call the Pack method.

Note: In Delphi, Items is the default property of TObjectList. This means you can omit the property name. Thus, the following two lines of code are both acceptable and do the same thing:

Note: ObjectList1.Items[i] := Form1.FindChildControl(ListBox1.Items[i]);

Note: ObjectList1[i] := Form1.FindChildControl(ListBox1.Items[i]);

See Also

OwnsObjects (see page 5686)

Add (see page 5680)

Insert (see page 5683)

Delete (see page 2874)

Remove (see page 5684)

List (see page 2900)

Extract (see page 5681)

Count (see page 2896)

Clear (see page 2874)

Pack (see page 2885)

operator_sb

185.1.21 TScriptObjectFactories Class

Class Hierarchy

System::TObject → WebScript.TAbstractScriptObjectFactories → WebScript.TScriptObjectFactories

Pascal

```
TScriptObjectFactories = class(TAbstractScriptObjectFactories);
```

C++

```
class TScriptObjectFactories : public TAbstractScriptObjectFactories;
```

File

WebScript

Description

This is class WebScript.TScriptObjectFactories.

185.1.21.1 TScriptObjectFactories Members

The following tables list the members exposed by TScriptObjectFactories.

Protected Methods

	Name	Description
	GetCount (see page 25529)	This is GetCount, a member of class TScriptObjectFactories.
	GetFactory (see page 25529)	This is GetFactory, a member of class TScriptObjectFactories.
	GetObjectList (see page 25529)	This is GetObjectList, a member of class TScriptObjectFactories.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25527)	Constructs an object and initializes its data before the object is first used.
	CreateGlobalObject (see page 25529)	This is CreateGlobalObject, a member of class TScriptObjectFactories.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25528)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Count (see page 25530)	This is Count, a member of class TScriptObjectFactories.
R	Factories (see page 25530)	This is Factories, a member of class TScriptObjectFactories.
R	ObjectList (see page 25473)	This is ObjectList, a member of class TAbstractScriptObjectFactories.

185.1.21.2 TScriptObjectFactories.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TScriptObjectFactories();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

185.1.21.3 TScriptObjectFactories.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScriptObjectFactories();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

185.1.21.4 TScriptObjectFactories Methods

The methods of the TScriptObjectFactories class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 25529)	This is GetCount, a member of class TScriptObjectFactories.
	GetFactory (see page 25529)	This is GetFactory, a member of class TScriptObjectFactories.
	GetObjectList (see page 25529)	This is GetObjectList, a member of class TScriptObjectFactories.

Public Methods

	Name	Description
	CreateGlobalObject (see page 25529)	This is CreateGlobalObject, a member of class TScriptObjectFactories.

185.1.21.4.1 TScriptObjectFactories.CreateGlobalObject Method

Pascal

```
function CreateGlobalObject(AVariable: TGlobalScriptVariables; AContext: IScriptContext):
IInterface;
```

C++

```
__fastcall IInterface CreateGlobalObject(TGlobalScriptVariables AVariable, IScriptContext
AContext);
```

Description

This is CreateGlobalObject, a member of class TScriptObjectFactories.

185.1.21.4.2 TScriptObjectFactories.GetCount Method

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

This is GetCount, a member of class TScriptObjectFactories.

185.1.21.4.3 TScriptObjectFactories.GetFactory Method

Pascal

```
function GetFactory(I: Integer): TAbstractScriptObjectFactory; override;
```

C++

```
virtual __fastcall TAbstractScriptObjectFactory GetFactory(int I);
```

Description

This is GetFactory, a member of class TScriptObjectFactories.

185.1.21.4.4 TScriptObjectFactories.GetObjectList Method

Pascal

```
function GetObjectList: TObjectList; override;
```

C++

```
virtual __fastcall TObjectList GetObjectList();
```

Description

This is GetObjectList, a member of class TScriptObjectFactories.

185.1.21.5 TScriptObjectFactories Properties

The properties of the TScriptObjectFactories class are listed here.

Public Properties

	Name	Description
R	Count (see page 25530)	This is Count, a member of class TScriptObjectFactories.
R	Factories (see page 25530)	This is Factories, a member of class TScriptObjectFactories.

185.1.21.5.1 TScriptObjectFactories.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TScriptObjectFactories.

185.1.21.5.2 TScriptObjectFactories.Factories Property

Pascal

```
property Factories [I: Integer]: TAbstractScriptObjectFactory;
```

C++

```
__property TAbstractScriptObjectFactory Factories[int I];
```

Description

This is Factories, a member of class TScriptObjectFactories.

185.1.22 TScriptProducer Class

Class Hierarchy

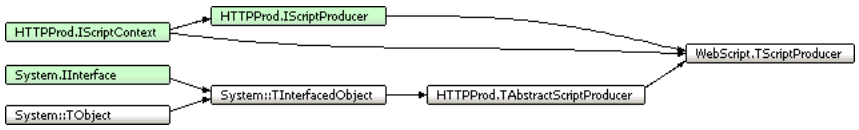

Pascal

```
TScriptProducer = class(TAbstractScriptProducer, IScriptProducer, IScriptContext);
```

C++

```
class TScriptProducer : public TAbstractScriptProducer, public IScriptProducer, public
IScriptContext;
```

File

WebScript

Description

This is class WebScript.TScriptProducer.

185.1.22.1 TScriptProducer Members

The following tables list the members exposed by TScriptProducer.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Fields

	Name	Description
	FErrors (see page 25534)	This is FErrors, a member of class TScriptProducer.
	FScriptEngine (see page 25534)	This is FScriptEngine, a member of class TScriptProducer.
	FScriptFile (see page 25534)	This is FScriptFile, a member of class TScriptProducer.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	Evaluate (see page 25535)	This is Evaluate, a member of class TScriptProducer.
A	EvaluateExpression (see page 25535)	This is EvaluateExpression, a member of class TScriptProducer.
	GetContent (see page 25535)	This is GetContent, a member of class TScriptProducer.
	GetErrors (see page 25535)	This is GetErrors, a member of class TScriptProducer.
	GetHTMLBlock (see page 25536)	This is GetHTMLBlock, a member of class TScriptProducer.
	GetHTMLBlockCount (see page 25536)	This is GetHTMLBlockCount, a member of class TScriptProducer.
	GetWebModuleContext (see page 25536)	This is GetWebModuleContext, a member of class TScriptProducer.
	HandleParserError (see page 25536)	This is HandleParserError, a member of class TScriptProducer.
	HandleScriptError (see page 25536)	This is HandleScriptError, a member of class TScriptProducer.
A	ImplHandleScriptError (see page 25537)	This is ImplHandleScriptError, a member of class TScriptProducer.
	ParseStream (see page 25537)	This is ParseStream, a member of class TScriptProducer.
	ParseString (see page 25537)	This is ParseString, a member of class TScriptProducer.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	ReplaceTags (see page 25537)	This is ReplaceTags, a member of class TScriptProducer.
	SetContent (see page 25538)	This is SetContent, a member of class TScriptProducer.
	Write (see page 25538)	This is the overview for the Write method overload.
	WriteItem (see page 25539)	This is WriteItem, a member of class TScriptProducer.

Protected Properties

	Name	Description
R	Component (see page 25539)	This is Component, a member of class TScriptProducer.
R	Content (see page 25539)	This is Content, a member of class TScriptProducer.
R	Designing (see page 25540)	This is Designing, a member of class TScriptProducer.
R	Errors (see page 25540)	This is Errors, a member of class TScriptProducer.
R	Script (see page 25540)	This is Script, a member of class TScriptProducer.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.

	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25532)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 25533)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	Write (see page 15326)	This is the overview for the Write method overload.

Public Properties

	Name	Description
	HTMLBlockCount (see page 15328)	This is HTMLBlockCount, a member of class IScriptProducer.
	HTMLBlocks (see page 15328)	This is HTMLBlocks, a member of class IScriptProducer.
	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	WebModuleContext (see page 15322)	This is WebModuleContext, a member of class IScriptContext.

185.1.22.2 TScriptProducer.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create(AWebModuleContext: TWebModuleContext; AStripParamQuotes: Boolean;
AHandleTag: THandleTagProc; const AScriptEngine: string; ALocateFileService:
ILocateFileService); override;
```

C++

```
virtual __fastcall TScriptProducer(TWebModuleContext AWebModuleContext, Boolean
AStripParamQuotes, THandleTagProc AHandleTag, const AnsiString AScriptEngine,
ILocateFileService ALocateFileService);
```

Description

Do not create instances of Create. Instead, construct descendants of Create using the new keyword, passing any arguments for the descendant object's constructor.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

Destroy (see page 22287)

new

185.1.22.3 TScriptProducer.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScriptProducer();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

185.1.22.4 TScriptProducer Fields

The fields of the TScriptProducer class are listed here.

Protected Fields

	Name	Description
	FErrors (see page 25534)	This is FErrors, a member of class TScriptProducer.

	FScriptEngine (see page 25534)	This is FScriptEngine, a member of class TScriptProducer.
	FScriptFile (see page 25534)	This is FScriptFile, a member of class TScriptProducer.

185.1.22.4.1 TScriptProducer.FErrors Field

Pascal

```
FErrors: TAbstractScriptErrors;
```

C++

```
TAbstractScriptErrors FErrors;
```

Description

This is FErrors, a member of class TScriptProducer.

185.1.22.4.2 TScriptProducer.FScriptEngine Field

Pascal

```
FScriptEngine: string;
```

C++

```
AnsiString FScriptEngine;
```

Description

This is FScriptEngine, a member of class TScriptProducer.

185.1.22.4.3 TScriptProducer.FScriptFile Field

Pascal

```
FScriptFile: TScriptFile;
```

C++

```
TScriptFile FScriptFile;
```

Description

This is FScriptFile, a member of class TScriptProducer.

185.1.22.5 TScriptProducer Methods

The methods of the TScriptProducer class are listed here.

Protected Methods

	Name	Description
	Evaluate (see page 25535)	This is Evaluate, a member of class TScriptProducer.
	EvaluateExpression (see page 25535)	This is EvaluateExpression, a member of class TScriptProducer.
	GetContent (see page 25535)	This is GetContent, a member of class TScriptProducer.
	GetErrors (see page 25535)	This is GetErrors, a member of class TScriptProducer.
	GetHTMLBlock (see page 25536)	This is GetHTMLBlock, a member of class TScriptProducer.
	GetHTMLBlockCount (see page 25536)	This is GetHTMLBlockCount, a member of class TScriptProducer.
	GetWebModuleContext (see page 25536)	This is GetWebModuleContext, a member of class TScriptProducer.
	HandleParserError (see page 25536)	This is HandleParserError, a member of class TScriptProducer.
	HandleScriptError (see page 25536)	This is HandleScriptError, a member of class TScriptProducer.

	ImplHandleScriptError (see page 25537)	This is ImplHandleScriptError, a member of class TScriptProducer.
	ParseStream (see page 25537)	This is ParseStream, a member of class TScriptProducer.
	ParseString (see page 25537)	This is ParseString, a member of class TScriptProducer.
	ReplaceTags (see page 25537)	This is ReplaceTags, a member of class TScriptProducer.
	SetContent (see page 25538)	This is SetContent, a member of class TScriptProducer.
	Write (see page 25538)	This is the overview for the Write method overload.
	WriteItem (see page 25539)	This is WriteItem, a member of class TScriptProducer.

185.1.22.5.1 TScriptProducer.Evaluate Method

Pascal

```
function Evaluate: string;
```

C++

```
__fastcall AnsiString Evaluate();
```

Description

This is Evaluate, a member of class TScriptProducer.

185.1.22.5.2 TScriptProducer.EvaluateExpression Method

Pascal

```
function EvaluateExpression(AScriptFile: TScriptFile): string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString EvaluateExpression(TScriptFile AScriptFile) = 0;
```

Description

This is EvaluateExpression, a member of class TScriptProducer.

185.1.22.5.3 TScriptProducer.GetContent Method

Pascal

```
function GetContent: string;
```

C++

```
__fastcall AnsiString GetContent();
```

Description

This is GetContent, a member of class TScriptProducer.

185.1.22.5.4 TScriptProducer.GetErrors Method

Pascal

```
function GetErrors: TAbstractScriptErrors;
```

C++

```
__fastcall TAbstractScriptErrors GetErrors();
```

Description

This is GetErrors, a member of class TScriptProducer.

185.1.22.5.5 TScriptProducer.GetHTMLBlock Method

Pascal

```
function GetHTMLBlock(I: Integer): string;
```

C++

```
__fastcall AnsiString GetHTMLBlock(int I);
```

Description

This is GetHTMLBlock, a member of class TScriptProducer.

185.1.22.5.6 TScriptProducer.GetHTMLBlockCount Method

Pascal

```
function GetHTMLBlockCount: Integer;
```

C++

```
__fastcall int GetHTMLBlockCount();
```

Description

This is GetHTMLBlockCount, a member of class TScriptProducer.

185.1.22.5.7 TScriptProducer.GetWebModuleContext Method

Pascal

```
function GetWebModuleContext: TWebModuleContext;
```

C++

```
__fastcall TWebModuleContext GetWebModuleContext();
```

Description

This is GetWebModuleContext, a member of class TScriptProducer.

185.1.22.5.8 TScriptProducer.HandleParserError Method

Pascal

```
procedure HandleParserError(E: Exception);
```

C++

```
__fastcall HandleParserError(Exception E);
```

Description

This is HandleParserError, a member of class TScriptProducer.

185.1.22.5.9 TScriptProducer.HandleScriptError Method

Pascal

```
function HandleScriptError(const ScriptError: IUnknown): HRESULT;
```

C++

```
__fastcall HRESULT HandleScriptError(const IUnknown ScriptError);
```

Description

This is HandleScriptError, a member of class TScriptProducer.

185.1.22.5.10 TScriptProducer.ImplHandleScriptError Method

Pascal

```
function ImplHandleScriptError(const ScriptError: IUnknown): HRESULT; virtual; abstract;
```

C++

```
virtual __fastcall HRESULT ImplHandleScriptError(const IUnknown ScriptError) = 0;
```

Description

This is ImplHandleScriptError, a member of class TScriptProducer.

185.1.22.5.11 TScriptProducer.ParseStream Method

Pascal

```
procedure ParseStream(Stream: TStream; Owned: Boolean);
```

C++

```
__fastcall ParseStream(TStream Stream, Boolean Owned);
```

Description

This is ParseStream, a member of class TScriptProducer.

185.1.22.5.12 TScriptProducer.ParseString Method

Pascal

```
procedure ParseString(const S: string);
```

C++

```
__fastcall ParseString(const AnsiString S);
```

Description

This is ParseString, a member of class TScriptProducer.

185.1.22.5.13 TScriptProducer.ReplaceTags Method

Pascal

```
function ReplaceTags(const AValue: string): string;
```

C++

```
__fastcall AnsiString ReplaceTags(const AnsiString AValue);
```

Description

This is ReplaceTags, a member of class TScriptProducer.

185.1.22.5.14 TScriptProducer.SetContent Method

Pascal

```
procedure SetContent(const Value: string);
```

C++

```
__fastcall SetContent(const AnsiString Value);
```

Description

This is SetContent, a member of class TScriptProducer.

185.1.22.5.15 Write Method

This is the overview for the Write method overload.

Overload List

	Name	Description
	TScriptProducer.Write (PChar) (see page 25538)	This is Write, a member of class TScriptProducer.
	TScriptProducer.Write (PWideChar, Integer) (see page 25538)	This is Write, a member of class TScriptProducer.
	TScriptProducer.Write (string) (see page 25538)	This is Write, a member of class TScriptProducer.

185.1.22.5.15.1 TScriptProducer.Write Method (PChar)

Pascal

```
procedure Write(Value: PChar); overload;
```

C++

```
__fastcall Write(const char * Value);
```

Description

This is Write, a member of class TScriptProducer.

185.1.22.5.15.2 TScriptProducer.Write Method (PWideChar, Integer)

Pascal

```
procedure Write(const AValue: PWideChar; ALength: Integer); overload;
```

C++

```
__fastcall Write(const PWideChar AValue, int ALength);
```

Description

This is Write, a member of class TScriptProducer.

185.1.22.5.15.3 TScriptProducer.Write Method (string)

Pascal

```
procedure Write(const Value: string); overload;
```

C++

```
__fastcall Write(const AnsiString Value);
```

Description

This is Write, a member of class TScriptProducer.

185.1.22.5.16 TScriptProducer.WriteItem Method

Pascal

```
procedure WriteItem(Index: Integer);
```

C++

```
__fastcall WriteItem(int Index);
```

Description

This is WriteItem, a member of class TScriptProducer.

185.1.22.6 TScriptProducer Properties

The properties of the TScriptProducer class are listed here.

Protected Properties

	Name	Description
R	Component (see page 25539)	This is Component, a member of class TScriptProducer.
R	Content (see page 25539)	This is Content, a member of class TScriptProducer.
R	Designing (see page 25540)	This is Designing, a member of class TScriptProducer.
R	Errors (see page 25540)	This is Errors, a member of class TScriptProducer.
R	Script (see page 25540)	This is Script, a member of class TScriptProducer.

185.1.22.6.1 TScriptProducer.Component Property

Pascal

```
property Component: TComponent;
```

C++

```
__property TComponent * Component;
```

Description

This is Component, a member of class TScriptProducer.

185.1.22.6.2 TScriptProducer.Content Property

Pascal

```
property Content: string;
```

C++

```
__property AnsiString Content;
```

Description

This is Content, a member of class TScriptProducer.

185.1.22.6.3 TScriptProducer.Designing Property

Pascal

```
property Designing: Boolean;
```

C++

```
__property Boolean Designing;
```

Description

This is Designing, a member of class TScriptProducer.

185.1.22.6.4 TScriptProducer.Errors Property

Pascal

```
property Errors: TAbstractScriptErrors;
```

C++

```
__property TAbstractScriptErrors Errors;
```

Description

This is Errors, a member of class TScriptProducer.

185.1.22.6.5 TScriptProducer.Script Property

Pascal

```
property Script: string;
```

C++

```
__property AnsiString Script;
```

Description

This is Script, a member of class TScriptProducer.

185.1.23 TVariableLookup Class

Class Hierarchy

Pascal

```
TVariableLookup = class(TObject);
```

C++

```
class TVariableLookup : public TObject;
```

File

WebScript

Description

This is class WebScript.TVariableLookup.

185.1.23.1 TVariableLookup Members

The following tables list the members exposed by TVariableLookup.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25541)	This is Create, a member of class TVariableLookup.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindGlobalVariable (see page 25542)	This is FindGlobalVariable, a member of class TVariableLookup.
	FindGlobalVariableContainer (see page 25542)	This is FindGlobalVariableContainer, a member of class TVariableLookup.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	LookupObject (see page 25542)	This is LookupObject, a member of class TVariableLookup.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Ident (see page 25543)	This is Ident, a member of class TVariableLookup.
R	Obj (see page 25543)	This is Obj, a member of class TVariableLookup.

185.1.23.2 TVariableLookup.Create Constructor

Pascal

```
constructor Create(const AIdent: string; AWebModuleContext: TWebModuleContext);
```

C++

```
__fastcall TVariableLookup(const AnsiString AIdent, TWebModuleContext AWebModuleContext);
```

Description

This is Create, a member of class TVariableLookup.

185.1.23.3 TVariableLookup Methods

The methods of the TVariableLookup class are listed here.

Public Methods

	Name	Description
	FindGlobalVariable (see page 25542)	This is FindGlobalVariable, a member of class TVariableLookup.
	FindGlobalVariableContainer (see page 25542)	This is FindGlobalVariableContainer, a member of class TVariableLookup.
	LookupObject (see page 25542)	This is LookupObject, a member of class TVariableLookup.

185.1.23.3.1 TVariableLookup.FindGlobalVariable Method

Pascal

```
function FindGlobalVariable(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindGlobalVariable(const AnsiString AName);
```

Description

This is FindGlobalVariable, a member of class TVariableLookup.

185.1.23.3.2 TVariableLookup.FindGlobalVariableContainer Method

Pascal

```
function FindGlobalVariableContainer(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindGlobalVariableContainer(const AnsiString AName);
```

Description

This is FindGlobalVariableContainer, a member of class TVariableLookup.

185.1.23.3.3 TVariableLookup.LookupObject Method

Pascal

```
function LookupObject: Boolean;
```

C++

```
__fastcall Boolean LookupObject();
```

Description

This is LookupObject, a member of class TVariableLookup.

185.1.23.4 TVariableLookup Properties

The properties of the TVariableLookup class are listed here.

Public Properties

	Name	Description
	Ident (see page 25543)	This is Ident, a member of class TVariableLookup.
R	Obj (see page 25543)	This is Obj, a member of class TVariableLookup.

185.1.23.4.1 TVariableLookup.Ident Property

Pascal

```
property Ident: string;
```

C++

```
__property AnsiString Ident;
```

Description

This is Ident, a member of class TVariableLookup.

185.1.23.4.2 TVariableLookup.Obj Property

Pascal

```
property Obj: TObject;
```

C++

```
__property TObject * Obj;
```

Description

This is Obj, a member of class TVariableLookup.

185.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TGlobalScriptVariables (see page 25544)	This is record WebScript.TGlobalScriptVariables.
	TIncludeType (see page 25545)	This is record WebScript.TIncludeType.

Records

	Name	Description
	TLineNumbers (see page 25545)	This is record WebScript.TLineNumbers.

185.2.1 WebScript.TGlobalScriptVariables Enumeration

Pascal

```
TGlobalScriptVariables = (
  svPage,
  svPages,
  svModules,
  svApplication,
  svSession,
  svEndUser,
  svResponse,
  svRequest,
  svHTMLItems,
  svProducer
);
```

C++

```
enum TGlobalScriptVariables {
  svPage,
  svPages,
  svModules,
  svApplication,
  svSession,
  svEndUser,
  svResponse,
  svRequest,
  svHTMLItems,
  svProducer
};
```

File

WebScript

Description

This is record WebScript.TGlobalScriptVariables.

185.2.2 WebScript.TIncludeType Enumeration

Pascal

```
TIncludeType = (
  incFile,
  incPage
);
```

C++

```
enum TIncludeType {
  incFile,
  incPage
};
```

File

WebScript

Description

This is record WebScript.TIncludeType.

185.2.3 WebScript.TLineNumbers Record

Pascal

```
TLineNumbers = record
  SourceStartLine: Integer;
  SourceLineCount: Integer;
  DestStartLine: Integer;
  DestLineCount: Integer;
  ScriptIncludeFile: TScriptIncludeFile;
end;
```

C++

```
struct TLineNumbers {
  int SourceStartLine;
  int SourceLineCount;
  int DestStartLine;
  int DestLineCount;
  TScriptIncludeFile ScriptIncludeFile;
};
```

File

WebScript

Description

This is record WebScript.TLineNumbers.

185.3 Types

The following table lists types in this documentation.

Types

Name	Description
CreateComponentStubFactory (see page 25546)	This is type WebScript.CreateComponentStubFactory.
PLineNumbers (see page 25546)	This is type WebScript.PLineNumbers.

185.3.1 WebScript.CreateComponentStubFactory Type

Pascal

```
CreateComponentStubFactory = function (AComponent: TObject; AComponentClass: TClass):
IInterface;
```

C++

```
(AComponent: TObject; AComponentClass: TClass): IInterface ( CreateComponentStubFactory)();
```

File

WebScript

Description

This is type WebScript.CreateComponentStubFactory.

185.3.2 WebScript.PLineNumbers Type

Pascal

```
PLineNumbers = ^TLineNumbers;
```

C++

```
^TLineNumbers PLineNumbers;
```

File

WebScript

Description

This is type WebScript.PLineNumbers.

185.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
GlobalScriptVariableNames (see page 25547)	This is constant WebScript.GlobalScriptVariableNames.

185.4.1 WebScript.GlobalScriptVariableNames Constant

Pascal

```
GlobalScriptVariableNames: array[TGlobalScriptVariables] of PChar = ('Page', 'Pages',
'Modules', 'Application', 'Session', 'EndUser', 'Response', 'Request', 'HTML_', 'Producer');
```

C++

```
array[TGlobalScriptVariables] of PChar GlobalScriptVariableNames = ('Page', 'Pages',
'Modules', 'Application', 'Session', 'EndUser', 'Response', 'Request', 'HTML_', 'Producer');
```

File

WebScript

Description

This is constant WebScript.GlobalScriptVariableNames.

186 WebServExp Namespace

This is namespace WebServExp.

Topics

Name	Description
Classes (see page 25551)	The following table lists classes in this documentation.
Interfaces (see page 25559)	The following table lists interfaces in this documentation.
Functions (see page 25562)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 25564)	The following table lists structs, records, enums in this documentation.
Types (see page 25566)	The following table lists types in this documentation.

Classes

	Name	Description
	TWebServExp (see page 25551)	This is class WebServExp.TWebServExp.

Functions

	Name	Description
	GetAliasBaseTypeInfo (see page 25562)	This is function WebServExp.GetAliasBaseTypeInfo.
	GetBindingType (see page 25562)	This is function WebServExp.GetBindingType.
	IsBaseClassTypeInfo (see page 25563)	This is function WebServExp.IsBaseClassTypeInfo.

Interfaces

	Name	Description
	IWebServExp (see page 25559)	IWebServExp is the interface for classes that generate schema definitions for a WSDL publisher.
	IWebServExpAccess (see page 25561)	This is class WebServExp.IWebServExpAccess.

Structs, Records, Enums

	Name	Description
	ArgumentType (see page 25564)	This is record WebServExp.ArgumentType.
	MessageType (see page 25564)	This is record WebServExp.MessageType.
	TSchemaType (see page 25565)	This is record WebServExp.TSchemaType.

Types

Name	Description
TAfterPublishingWSDLEvent (see page 25566)	This is type WebServExp.TAfterPublishingWSDLEvent.

TBeforePublishingTypesEvent (see page 25566)	This is type WebServExp.TBeforePublishingTypesEvent.
TPublishingTypeEvent (see page 25567)	This is type WebServExp.TPublishingTypeEvent.
TSchemaTypeArray (see page 25567)	This is type WebServExp.TSchemaTypeArray.

186.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TWebServExp (see page 25551)	This is class WebServExp.TWebServExp.

186.1.1 TWebServExp Class

Class Hierarchy

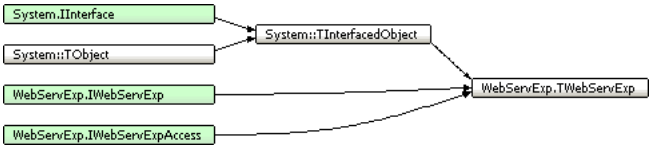

Pascal

```
TWebServExp = class(TInterfacedObject, IWebServExp, IWebServExpAccess);
```

C++

```
class TWebServExp : public TInterfacedObject, public IWebServExp, public IWebServExpAccess;
```

File

WebServExp

Description

This is class WebServExp.TWebServExp.

186.1.1.1 TWebServExp Members

The following tables list the members exposed by TWebServExp.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25553)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25553)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindOrAddSchema (see page 25560)	Ensures that a schema is eventually generated for a specified type.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GenerateXMLSchema (see page 25554)	Adds schema information to represent a specified type.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetWebServExp (see page 25554)	This is GetWebServExp, a member of class TWebServExp.
	GetWSDLForInterface (see page 25555)	This is GetWSDLForInterface, a member of class TWebServExp.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	ImportLocations (see page 25556)	This is ImportLocations, a member of class TWebServExp.
	ImportNames (see page 25556)	This is ImportNames, a member of class TWebServExp.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	TargetNameSpace (see page 25556)	This is TargetNameSpace, a member of class TWebServExp.

Published Events

	Name	Description
	OnAfterPublishingWSDL (see page 25557)	This is OnAfterPublishingWSDL, a member of class TWebServExp.

	OnBeforePublishingTypes (see page 25557)	This is OnBeforePublishingTypes, a member of class TWebServExp.
	OnPublishingType (see page 25557)	This is OnPublishingType, a member of class TWebServExp.

Published Properties

	Name	Description
	ArrayAsComplexContent (see page 25555)	This is ArrayAsComplexContent, a member of class TWebServExp.
	BindingType (see page 25555)	This is BindingType, a member of class TWebServExp.
	ServiceAddress (see page 25556)	This is ServiceAddress, a member of class TWebServExp.
	WSDLElements (see page 25557)	This is WSDLElements, a member of class TWebServExp.

186.1.1.2 TWebServExp.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TWebServExp();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

186.1.1.3 TWebServExp.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebServExp();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance

that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

186.1.1.4 TWebServExp Methods

The methods of the TWebServExp class are listed here.

Public Methods

	Name	Description
	GenerateXMLSchema (see page 25554)	Adds schema information to represent a specified type.
	GetWebServExp (see page 25554)	This is GetWebServExp, a member of class TWebServExp.
	GetWSDLForInterface (see page 25555)	This is GetWSDLForInterface, a member of class TWebServExp.

186.1.1.4.1 TWebServExp.GenerateXMLSchema Method

Adds schema information to represent a specified type.

Pascal

```
procedure GenerateXMLSchema(SchemaDef: IXMLSchemaDef; const ATypeInfo: PTypeinfo; const
ParentInfo: PTypeinfo; Namespace: WideString);
```

C++

```
__fastcall GenerateXMLSchema(IXMLSchemaDef SchemaDef, const PTypeinfo ATypeInfo, const
PTypeinfo ParentInfo, BSTR Namespace);
```

Description

GenerateXMLSchema adds information to an emerging schema to represent information about a specified type.

SchemaDef is the interface for the emerging schema. GenerateXMLSchema uses the properties and methods of this interface to add schema information for the specified type.

ATypeInfo is the runtime type information (RTT) for the type to add.

ParentInfo is the RTTI for the type from which the type specified by ATypeInfo is derived. If ATypeInfo does not describe a descendant type, ParentInfo is nil (Delphi) or NULL (C++).

Namespace is the namespace URI in which the schema definition is defined.

See Also

FindOrAddSchema (see page 25560)

186.1.1.4.2 TWebServExp.GetWebServExp Method

Pascal

```
function GetWebServExp: TWebServExp;
```

C++

```
__fastcall TWebServExp GetWebServExp();
```

Description

This is GetWebServExp, a member of class TWebServExp.

186.1.1.4.3 TWebServExp.GetWSDLForInterface Method

Pascal

```
procedure GetWSDLForInterface(const IntfTypeInfo: Pointer; WSDLDoc: IWSDLDocument; PortNames:
array of WideString; Locations: array of WideString);
```

C++

```
__fastcall GetWSDLForInterface(const void * IntfTypeInfo, IWSDLDocument WSDLDoc, array of
WideString PortNames, array of WideString Locations);
```

Description

This is GetWSDLForInterface, a member of class TWebServExp.

186.1.1.5 TWebServExp Properties

The properties of the TWebServExp class are listed here.

Public Properties

	Name	Description
	ImportLocations (see page 25556)	This is ImportLocations, a member of class TWebServExp.
	ImportNames (see page 25556)	This is ImportNames, a member of class TWebServExp.
	TargetNameSpace (see page 25556)	This is TargetNameSpace, a member of class TWebServExp.

Published Properties

	Name	Description
	ArrayAsComplexContent (see page 25555)	This is ArrayAsComplexContent, a member of class TWebServExp.
	BindingType (see page 25555)	This is BindingType, a member of class TWebServExp.
	ServiceAddress (see page 25556)	This is ServiceAddress, a member of class TWebServExp.
	WSDLElements (see page 25557)	This is WSDLElements, a member of class TWebServExp.

186.1.1.5.1 TWebServExp.ArrayAsComplexContent Property

Pascal

```
property ArrayAsComplexContent: Boolean;
```

C++

```
__property Boolean ArrayAsComplexContent;
```

Description

This is ArrayAsComplexContent, a member of class TWebServExp.

186.1.1.5.2 TWebServExp.BindingType Property

Pascal

```
property BindingType: TWebServiceBindingType;
```

C++

```
__property TWebServiceBindingType BindingType;
```

Description

This is BindingType, a member of class TWebServExp.

186.1.1.5.3 TWebServExp.ImportLocations Property

Pascal

```
property ImportLocations [const Index: Integer]: WideString;
```

C++

```
__property BSTR ImportLocations[int const Index];
```

Description

This is ImportLocations, a member of class TWebServExp.

186.1.1.5.4 TWebServExp.ImportNames Property

Pascal

```
property ImportNames [const Index: Integer]: WideString;
```

C++

```
__property BSTR ImportNames[int const Index];
```

Description

This is ImportNames, a member of class TWebServExp.

186.1.1.5.5 TWebServExp.ServiceAddress Property

Pascal

```
property ServiceAddress: WideString;
```

C++

```
__property BSTR ServiceAddress;
```

Description

This is ServiceAddress, a member of class TWebServExp.

186.1.1.5.6 TWebServExp.TargetNameSpace Property

Pascal

```
property TargetNameSpace: WideString;
```

C++

```
__property BSTR TargetNameSpace;
```

Description

This is TargetNameSpace, a member of class TWebServExp.

186.1.1.5.7 TWebServExp.WSDLElements Property

Pascal

```
property WSDLElements: TWSDLElements;
```

C++

```
__property TWSDLElements WSDLElements;
```

Description

This is WSDLElements, a member of class TWebServExp.

186.1.1.6 TWebServExp Events

The events of the TWebServExp class are listed here.

Published Events

	Name	Description
	OnAfterPublishingWSDL (see page 25557)	This is OnAfterPublishingWSDL, a member of class TWebServExp.
	OnBeforePublishingTypes (see page 25557)	This is OnBeforePublishingTypes, a member of class TWebServExp.
	OnPublishingType (see page 25557)	This is OnPublishingType, a member of class TWebServExp.

186.1.1.6.1 TWebServExp.OnAfterPublishingWSDL Event

Pascal

```
property OnAfterPublishingWSDL: TAfterPublishingWSDLEvent;
```

C++

```
__property TAfterPublishingWSDLEvent OnAfterPublishingWSDL;
```

Description

This is OnAfterPublishingWSDL, a member of class TWebServExp.

186.1.1.6.2 TWebServExp.OnBeforePublishingTypes Event

Pascal

```
property OnBeforePublishingTypes: TBeforePublishingTypesEvent;
```

C++

```
__property TBeforePublishingTypesEvent OnBeforePublishingTypes;
```

Description

This is OnBeforePublishingTypes, a member of class TWebServExp.

186.1.1.6.3 TWebServExp.OnPublishingType Event

Pascal

```
property OnPublishingType: TPublishingTypeEvent;
```

C++

```
__property TPublishingTypeEvent OnPublishingType;
```

Description

This is OnPublishingType, a member of class TWebServExp.

186.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IWebServExp (see page 25559)	IWebServExp is the interface for classes that generate schema definitions for a WSDL publisher.
	IWebServExpAccess (see page 25561)	This is class WebServExp.IWebServExpAccess.

186.2.1 IWebServExp Interface

IWebServExp is the interface for classes that generate schema definitions for a WSDL publisher.

Class Hierarchy

WebServExp.IWebServExp

Pascal

```
IWebServExp = interface;
```

C++

```
__interface IWebServExp;
```

File

WebServExp

Description

IWebServExp is an interface for generating the schema definitions that represent types in a WSDL document.

Note: C++ method declarations that use IWebServExp use the _di_IWebServExp type instead. This type is a DelphiInterface wrapper around the IWebServExp interface:

Note: typedef System::DelphiInterface (see page 22167)< IWebServExp > _di_IWebServExp;

See Also

TWSDLHTMLPublish (see page 26024)

186.2.1.1 IWebServExp Members

The following tables list the members exposed by IWebServExp.

Public Methods

	Name	Description
	FindOrAddSchema (see page 25560)	Ensures that a schema is eventually generated for a specified type.
	GenerateXMLSchema (see page 25560)	Adds schema information to represent a specified type.

186.2.1.2 IWebServExp Methods

The methods of the IWebServExp class are listed here.

Public Methods

	Name	Description
	FindOrAddSchema (see page 25560)	Ensures that a schema is eventually generated for a specified type.
	GenerateXMLSchema (see page 25560)	Adds schema information to represent a specified type.

186.2.1.2.1 IWebServExp.FindOrAddSchema Method

Ensures that a schema is eventually generated for a specified type.

Pascal

```
function FindOrAddSchema(const ATypeInfo: PTypeinfo; const TnsURI: string): Boolean;
```

C++

```
__fastcall Boolean FindOrAddSchema(const PTypeinfo ATypeInfo, const AnsiString TnsURI);
```

Description

Call FindOrAddSchema to indicate that a schema should be generated for a given type.

ATypeInfo is the runtime type information (RTTI) for the type.

TnsURI is the namespace URI in which the schema for the type should be defined.

FindOrAddSchema returns true if there is already a schema for the specified type. It returns false if the type has not previously been added.

See Also

GenerateXMLSchema (see page 25560)

186.2.1.2.2 IWebServExp.GenerateXMLSchema Method

Adds schema information to represent a specified type.

Pascal

```
procedure GenerateXMLSchema(SchemaDef: IXMLSchemaDef; const ATypeInfo: PTypeinfo; const
ParentInfo: PTypeinfo; Namespace: WideString);
```

C++

```
__fastcall GenerateXMLSchema(IXMLSchemaDef SchemaDef, const PTypeinfo ATypeInfo, const
PTypeinfo ParentInfo, BSTR Namespace);
```

Description

GenerateXMLSchema adds information to an emerging schema to represent information about a specified type.

SchemaDef is the interface for the emerging schema. GenerateXMLSchema uses the properties and methods of this interface to add schema information for the specified type.

ATypeInfo is the runtime type information (RTT) for the type to add.

ParentInfo is the RTTI for the type from which the type specified by ATypeInfo is derived. If ATypeInfo does not describe a descendant type, ParentInfo is nil (Delphi) or NULL (C++).

Namespace is the namespace URI in which the schema definition is defined.

See Also

FindOrAddSchema (see page 25560)

186.2.2 IWebServExpAccess Interface

Class Hierarchy

WebServExp.IWebServExpAccess

Pascal

```
IWebServExpAccess = interface;
```

C++

```
__interface IWebServExpAccess;
```

File

WebServExp

Description

This is class WebServExp.IWebServExpAccess.

186.2.2.1 IWebServExpAccess Members

The following tables list the members exposed by IWebServExpAccess.

Public Methods

	Name	Description
	GetWebServExp (see page 25561)	This is GetWebServExp, a member of class IWebServExpAccess.

186.2.2.2 IWebServExpAccess Methods

The methods of the IWebServExpAccess class are listed here.

Public Methods

	Name	Description
	GetWebServExp (see page 25561)	This is GetWebServExp, a member of class IWebServExpAccess.

186.2.2.2.1 IWebServExpAccess.GetWebServExp Method

Pascal

```
function GetWebServExp: TWebServExp;
```

C++

```
__fastcall TWebServExp GetWebServExp();
```

Description

This is GetWebServExp, a member of class IWebServExpAccess.

186.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetAliasBaseTypeInfo (see page 25562)	This is function WebServExp.GetAliasBaseTypeInfo.
	GetBindingType (see page 25562)	This is function WebServExp.GetBindingType.
	IsBaseClassTypeInfo (see page 25563)	This is function WebServExp.IsBaseClassTypeInfo.

186.3.1 WebServExp.GetAliasBaseTypeInfo Function

Pascal

```
function GetAliasBaseTypeInfo(const ParamType: TTypeKind): PTypeInfo;
```

C++

```
PTypeInfo GetAliasBaseTypeInfo(const TTypeKind ParamType);
```

File

WebServExp

Description

This is function WebServExp.GetAliasBaseTypeInfo.

186.3.2 WebServExp.GetBindingType Function

Pascal

```
function GetBindingType(const MethEntry: TIntfMethEntry; Input: Boolean):
TWebServiceBindingType; overload;
```

C++

```
TWebServiceBindingType GetBindingType(const TIntfMethEntry MethEntry, Boolean Input);
```

File

WebServExp

Description

This is function WebServExp.GetBindingType.

186.3.3 WebServExp.IsBaseClassTypeInfo Function

Pascal

```
function IsBaseClassTypeInfo(const ATypeInfo: PTypeInfo): Boolean;
```

C++

```
Boolean IsBaseClassTypeInfo(const PTypeInfo ATypeInfo);
```

File

WebServExp

Description

This is function WebServExp.IsBaseClassTypeInfo.

186.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	ArgumentType (see page 25564)	This is record WebServExp.ArgumentType.
	MessageType (see page 25564)	This is record WebServExp.MessageType.

Records

	Name	Description
	TSchemaType (see page 25565)	This is record WebServExp.TSchemaType.

186.4.1 WebServExp.ArgumentType Enumeration

Pascal

```
ArgumentType = (
  argIn,
  argOut,
  argInOut,
  argReturn
);
```

C++

```
enum ArgumentType {
  argIn,
  argOut,
  argInOut,
  argReturn
};
```

File

WebServExp

Description

This is record WebServExp.ArgumentType.

186.4.2 WebServExp.MessageType Enumeration

Pascal

```
MessageType = (
  mtInput,
  mtOutput,
  mtHeaderInput,
  mtHeaderOutput,
  mtFault
);
```

C++

```
enum MessageType {
  mtInput,
  mtOutput,
  mtHeaderInput,
  mtHeaderOutput,
  mtFault
};
```

File

WebServExp

Description

This is record WebServExp.MessageType.

186.4.3 WebServExp.TSchemaType Record

Pascal

```
TSchemaType = record
  TypeName: WideString;
  NameSpace: WideString;
  TypeInfo: PTypeinfo;
  NSPrefix: WideString;
  XSGenerated: Boolean;
end;
```

C++

```
struct TSchemaType {
  BSTR TypeName;
  BSTR NameSpace;
  PTypeinfo TypeInfo;
  BSTR NSPrefix;
  Boolean XSGenerated;
};
```

File

WebServExp

Description

This is record WebServExp.TSchemaType.

186.5 Types

The following table lists types in this documentation.

Types

Name	Description
TAfterPublishingWSDLEvent (see page 25566)	This is type WebServExp.TAfterPublishingWSDLEvent.
TBeforePublishingTypesEvent (see page 25566)	This is type WebServExp.TBeforePublishingTypesEvent.
TPublishingTypeEvent (see page 25567)	This is type WebServExp.TPublishingTypeEvent.
TSchemaTypeArray (see page 25567)	This is type WebServExp.TSchemaTypeArray.

186.5.1 WebServExp.TAfterPublishingWSDLEvent Type

Pascal

```
TAfterPublishingWSDLEvent = procedure (const WSDLDoc: IWSDLDocument) of object;
```

C++

```
(const WSDLDoc: IWSDLDocument) ( TAfterPublishingWSDLEvent)();
```

File

WebServExp

Description

This is type WebServExp.TAfterPublishingWSDLEvent.

186.5.2 WebServExp.TBeforePublishingTypesEvent Type

Pascal

```
TBeforePublishingTypesEvent = procedure (const WebServ: IWebServExp) of object;
```

C++

```
(const WebServ: IWebServExp) ( TBeforePublishingTypesEvent)();
```

File

WebServExp

Description

This is type WebServExp.TBeforePublishingTypesEvent.

186.5.3 WebServExp.TPublishingTypeEvent Type

Pascal

```
TPublishingTypeEvent = procedure (const WebServ: IWebServExp; const SchemaDef: IXMLSchemaDef;
const ATypeInfo: PTypeinfo; Namespace: WideString) of object;
```

C++

```
(const WebServ: IWebServExp; const SchemaDef: IXMLSchemaDef; const ATypeInfo: PTypeinfo;
Namespace: WideString) ( TPublishingTypeEvent)();
```

File

WebServExp

Description

This is type WebServExp.TPublishingTypeEvent.

186.5.4 WebServExp.TSchemaTypeArray Type

Pascal

```
TSchemaTypeArray = array of TSchemaType;
```

C++

```
array of TSchemaType TSchemaTypeArray;
```

File

WebServExp

Description

This is type WebServExp.TSchemaTypeArray.

187 WebSess Namespace

This is namespace WebSess.

Topics

Name	Description
Classes (see page 25570)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 25601)	The following table lists structs, records, enums in this documentation.
Types (see page 25602)	The following table lists types in this documentation.

Classes

	Name	Description
	TCustomSessionsService (see page 25570)	TCustomSessionsService stores end user data that is needed for a short period of time.
	TSessionsService (see page 25582)	TSessionsService stores end user data that is needed for a short period of time.
	TWebSessionIDsImpl (see page 25588)	This is class WebSess.TWebSessionIDsImpl.
	TWebSessionImpl (see page 25592)	This is class WebSess.TWebSessionImpl.

Structs, Records, Enums

	Name	Description
	TEndSessionReason (see page 25601)	This is record WebSess.TEndSessionReason.

Types

Name	Description
TEndSessionEvent (see page 25602)	This is type WebSess.TEndSessionEvent.
TStartSessionEvent (see page 25602)	This is type WebSess.TStartSessionEvent.

187.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TCustomSessionsService (see page 25570)	TCustomSessionsService stores end user data that is needed for a short period of time.
	TSessionsService (see page 25582)	TSessionsService stores end user data that is needed for a short period of time.
	TWebSessionIDsImpl (see page 25588)	This is class WebSess.TWebSessionIDsImpl.
	TWebSessionImpl (see page 25592)	This is class WebSess.TWebSessionImpl.

187.1.1 TCustomSessionsService Class

TCustomSessionsService stores end user data that is needed for a short period of time.

Class Hierarchy

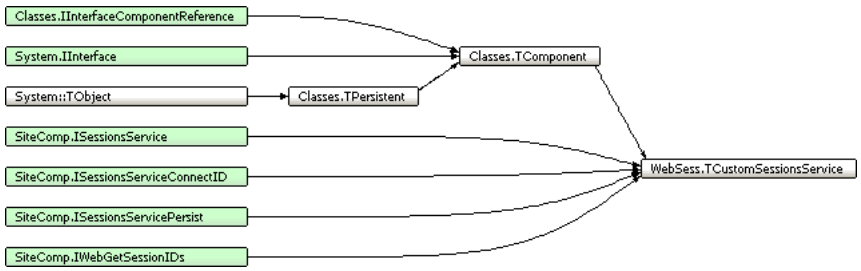

Pascal

```
TCustomSessionsService = class(TComponent, ISessionsService, ISessionsServiceConnectID,
IWebGetSessionIDs, ISessionsServicePersist);
```

C++

```
class TCustomSessionsService : public TComponent, public ISessionsService, public
ISessionsServiceConnectID, public IWebGetSessionIDs, public ISessionsServicePersist;
```

File

WebSess

Description

TCustomSessionsService is the base class for classes that manage the sessions in a WebSnap application. When a WebSnap application contains a TCustomSessionsService descendant, then each HTTP request is associated with a session. TCustomSessionsService can manage multiple sessions on behalf of different users, each on a separate thread of execution. Only one session is available when handling a request. This is called the current session. The current session is available as the Session property of the global Web context, which is accessed using the global WebContext function.

TCustomSessionsService stores session IDs and name/value pairs for all sessions in memory. This information must remain in memory between requests. Thus, Web application types that do not remain in memory between requests, such as CGI, do not support TCustomSessionsService.

TCustomSessionsService implements the ISessionsService interface.

See Also

TSessionsService (see page 25582)

TEndUserSessionAdapter (see page 24458)

TAbstractWebContext (see page 24644)

TWebSession (see page 24968)

WebContext

ISessionsService (see page 19733)

187.1.1.1 TCustomSessionsService Members

The following tables list the members exposed by TCustomSessionsService.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckStatusChange (see page 25575)	Checks whether any sessions have started or terminated.
	Clear (see page 25575)	Deletes all sessions from memory.
	Connect (see page 25575)	Obtains the interface for a Web session, given its Session ID.
	ConnectActive (see page 25576)	Obtains the interface for an active Web session, given its Session ID.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoEndSession (see page 25576)	Triggers the OnEndSession (see page 25581) event when appropriate.
V	DoStartSession (see page 25577)	Triggers the OnStartSession (see page 25582) event when appropriate.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSessionIDs (see page 25577)	Returns a list of the session IDs for all sessions currently in memory.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	LoadFromFile (see page 25577)	Loads the list of sessions from a file.
	LoadFromStream (see page 25578)	Loads the list of sessions from a stream.

	NewSession (see page 25578)	Starts a new session.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	NotifyStatusChange (see page 25579)	Generates an OnEndSession (see page 25581) or OnStartSession (see page 25582) event, as approrpriate.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SaveToFile (see page 25579)	Saves the list of sessions to a file.
	SaveToStream (see page 25580)	Saves the list of sessions to a stream.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnEndSession (see page 25581)	Occurs when the sessions service detects that a session has terminated.
	OnStartSession (see page 25582)	Occurs when the sessions service detects that a new session has started.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25574)	Creates and initializes an instance of TCustomSessionsService (see page 25570).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.

	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DefaultTimeout (see page 25580)	Specifies the number of minutes a session can persist without being accessed.
	DesignInfo (see page 2805)	Contains information used by the Form designer.

	MaxSessions (see page 25581)	Indicates the maximum number of sessions the TCustomSessionsService (see page 25570) can keep in memory.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

187.1.1.2 TCustomSessionsService.Create Constructor

Creates and initializes an instance of TCustomSessionsService (see page 25570).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomSessionsService(TComponent * AOwner);
```

Description

Call Create to instantiate a sessions service at runtime. Typically, this is unnecessary, because a WebSnap application needs at most a single sessions service component, and in most cases that is added at design time.

AOwner is the component that becomes the value of the TCustomSessionsService (see page 25570) object's Owner property. This should be the Web application module.

See Also

Owner (see page 2807)

187.1.1.3 TCustomSessionsService Methods

The methods of the TCustomSessionsService class are listed here.

Protected Methods

	Name	Description
	CheckStatusChange (see page 25575)	Checks whether any sessions have started or terminated.
	Clear (see page 25575)	Deletes all sessions from memory.
	Connect (see page 25575)	Obtains the interface for a Web session, given its Session ID.
	ConnectActive (see page 25576)	Obtains the interface for an active Web session, given its Session ID.
V	DoEndSession (see page 25576)	Triggers the OnEndSession (see page 25581) event when appropriate.
V	DoStartSession (see page 25577)	Triggers the OnStartSession (see page 25582) event when appropriate.
	GetSessionIDs (see page 25577)	Returns a list of the session IDs for all sessions currently in memory.
	LoadFromFile (see page 25577)	Loads the list of sessions from a file.
	LoadFromStream (see page 25578)	Loads the list of sessions from a stream.
	NewSession (see page 25578)	Starts a new session.
	NotifyStatusChange (see page 25579)	Generates an OnEndSession (see page 25581) or OnStartSession (see page 25582) event, as approrpriate.
	SaveToFile (see page 25579)	Saves the list of sessions to a file.
	SaveToStream (see page 25580)	Saves the list of sessions to a stream.

187.1.1.3.1 TCustomSessionsService.CheckStatusChange Method

Checks whether any sessions have started or terminated.

Pascal

```
function CheckStatusChange: Boolean;
```

C++

```
__fastcall Boolean CheckStatusChange();
```

Description

CheckStatusChange is called automatically when the application creates a new Web context or initializes a new Web module. That is, this method is called at least once every time the application receives a new HTTP request message.

If CheckStatusChange detects that a session has started or terminated since the last call, it returns true. Otherwise, it returns false.

See Also

NotifyStatusChange (see page 25579)

TWebContext (see page 24660)

187.1.1.3.2 TCustomSessionsService.Clear Method

Deletes all sessions from memory.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the sessions service of all the sessions it has stored in memory.

Clear does not generate an OnEndSession (see page 25581) event for the sessions it deletes when it frees their memory, it simply removes them from memory. To shut down all sessions in a more orderly fashion, iterate through all the active sessions by calling ConnectActive (see page 25576) for every session ID that the GetSessionIDs (see page 25577) method returns, and terminate each one. Then call NotifyStatusChange (see page 25579) to generate the OnEndSession (see page 25581) events for every terminated session. At that point, calling Clear does not prevent the application from receiving an OnEndSession (see page 25581) event for any session that the sessions service manages.

See Also

NewSession (see page 25578)

GetSessionIDs (see page 25577)

ConnectActive (see page 25576)

NotifyStatusChange (see page 25579)

187.1.1.3.3 TCustomSessionsService.Connect Method

Obtains the interface for a Web session, given its Session ID.

Pascal

```
function Connect(AID: TSessionID): IWebSession;
```

C++

```
__fastcall IWebSession Connect(TSessionID AID);
```

Description

Call Connect to access the session that has a specified Session ID. Unlike the ConnectActive (see page 25576) method, Connect may return a session that has expired or been terminated but that has not yet been freed from memory.

AID is the session ID of the session to locate.

Connect returns the IWebSession interface for the located session. This interface has a single method, Terminate, which lets you terminate the session. If the sessions service does not have a session with the specified session ID, Connect returns nil (Delphi) or NULL (C++).

Note: Calling Connect does not count as accessing a session. That is, if you call Connect for any session, it does not affect the time when that session times out.

See Also

ConnectActive (see page 25576)

GetSessionIDs (see page 25577)

187.1.1.3.4 TCustomSessionsService.ConnectActive Method

Obtains the interface for an active Web session, given its Session ID.

Pascal

```
function ConnectActive(AID: TSessionID): IWebSession;
```

C++

```
__fastcall IWebSession ConnectActive(TSessionID AID);
```

Description

Call ConnectActive to access the session that has a specified Session ID. Unlike the Connect (see page 25575) method, ConnectActive never returns a session that has expired or been terminated.

AID is the session ID of the session to locate.

ConnectActive returns the IWebSession interface for the located session. This interface has a single method, Terminate, which lets you terminate the session. If the sessions service does not have an active session with the specified session ID, ConnectActive returns nil (Delphi) or NULL (C++).

Note: Calling ConnectActive counts as accessing a session. That is, if you call ConnectActive for any session, the session will not time out until the entire time-out period elapses after the call to ConnectActive.

See Also

Connect (see page 25575)

GetSessionIDs (see page 25577)

187.1.1.3.5 TCustomSessionsService.DoEndSession Method

Triggers the OnEndSession (see page 25581) event when appropriate.

Pascal

```
procedure DoEndSession(ASession: TAbstractWebSession; AReason: TEndSessionReason); virtual;
```

C++

```
virtual __fastcall DoEndSession(TAbstractWebSession ASession, TEndSessionReason AReason);
```

Description

Use DoEndSession to trigger an OnEndSession (see page 25581) event, if it is appropriate to do so.

See Also

OnEndSession (see page 25581)

187.1.1.3.6 TCustomSessionsService.DoStartSession Method

Triggers the OnStartSession (see page 25582) event when appropriate.

Pascal

```
procedure DoStartSession(ASession: TAbstractWebSession); virtual;
```

C++

```
virtual __fastcall DoStartSession(TAbstractWebSession ASession);
```

Description

Use DoStartSession to trigger an OnStartSession (see page 25582) event, if it is appropriate to do so.

See Also

OnStartSession (see page 25582)

187.1.1.3.7 TCustomSessionsService.GetSessionIDs Method

Returns a list of the session IDs for all sessions currently in memory.

Pascal

```
function GetSessionIDs: IWebSessionIDs;
```

C++

```
__fastcall IWebSessionIDs GetSessionIDs();
```

Description

Call GetSessionIDs to obtain a list of the session IDs for all the sessions that the sessions service is storing. GetSessionIDs returns an IWebSessionIDs interface, which lists the session IDs.

See Also

Connect (see page 25575)

GetSessionIDs

187.1.1.3.8 TCustomSessionsService.LoadFromFile Method

Loads the list of sessions from a file.

Pascal

```
procedure LoadFromFile(const Filename: string);
```

C++

```
__fastcall LoadFromFile(const AnsiString Filename);
```

Description

Call LoadFromFile to replace the current list of sessions with a set of sessions stored in a file. The file must be previously created by a call to the SaveToFile (see page 25579) method.

Filename is the name of the file that contains the saved list of sessions.

See Also

SaveToFile (see page 25579)

LoadFromStream (see page 25578)

187.1.1.3.9 TCustomSessionsService.LoadFromStream Method

Loads the list of sessions from a stream.

Pascal

```
procedure LoadFromStream(S: TStream);
```

C++

```
__fastcall LoadFromStream(TStream S);
```

Description

Call LoadFromStream to replace the current list of sessions with a set of sessions read from a stream. The sessions must be previously saved by a call to the SaveToStream (see page 25580) method.

S is the stream from which to read the list of sessions.

See Also

SaveToStream (see page 25580)

LoadFromFile (see page 25577)

187.1.1.3.10 TCustomSessionsService.NewSession Method

Starts a new session.

Pascal

```
function NewSession: IWebSession;
```

C++

```
__fastcall IWebSession NewSession();
```

Description

Call NewSession to generate a new session. NewSession starts a new session and returns an interface that can be used to terminate that session.

If the sessions service already has MaxSessions (see page 25581) sessions, NewSession raises an ESessionsException exception.

Note: The OnStartSession (see page 25582) event does not occur immediately when you call NewSession to start a new session. It occurs on the next call to NotifyStatusChange (see page 25579).

Tip: To reconnect to an existing session, call the ConnectActive (see page 25576) method instead.

See Also

MaxSessions (see page 25581)

OnStartSession (see page 25582)

ConnectActive (see page 25576)

NotifyStatusChange (see page 25579)

187.1.1.3.11 TCustomSessionsService.NotifyStatusChange Method

Generates an OnEndSession (see page 25581) or OnStartSession (see page 25582) event, as approrpriate.

Pascal

```
procedure NotifyStatusChange;
```

C++

```
__fastcall NotifyStatusChange();
```

Description

Call NotifyStatusChange to generate the OnStartSession (see page 25582) event for any new sessions that have been created since the last call and to generate the OnEndSession (see page 25581) event for any sessions that terminated since the last call. For any terminated sessions, NotifyStatusChange frees the memory allocated to the session after the OnEndSession (see page 25581) event handler exits.

NotifyStatusChange is called automatically every time the WebSnap application initializes a new Web module.

See Also

OnEndSession (see page 25581)

OnStartSession (see page 25582)

CheckStatusChange (see page 25575)

187.1.1.3.12 TCustomSessionsService.SaveToFile Method

Saves the list of sessions to a file.

Pascal

```
procedure SaveToFile(const Filename: string);
```

C++

```
__fastcall SaveToFile(const AnsiString Filename);
```

Description

Call SaveToFile to save the current list of sessions to a file. SaveToFile saves all of the sessions, including their session IDs, name/value lists, time-out period, and status information about when they were last accessed.

The file can be read back into a sessions service by calling the LoadFromFile (see page 25577) method.

Filename is the name of the file to which the list of sessions is saved.

See Also

SaveToStream (see page 25580)

LoadFromFile (see page 25577)

187.1.1.3.13 TCustomSessionsService.SaveToStream Method

Saves the list of sessions to a stream.

Pascal

```
procedure SaveToStream(S: TStream);
```

C++

```
__fastcall SaveToStream(TStream S);
```

Description

Call SaveToStream to write the current list of sessions to a stream. SaveToStream saves all of the sessions, including their session IDs, name/value lists, time-out period, and status information about when they were last accessed.

The sessions can be read back into a sessions service by calling the LoadFromStream (see page 25578) method.

S is the stream to which the list of sessions is written.

See Also

LoadFromStream (see page 25578)

SaveToFile (see page 25579)

187.1.1.4 TCustomSessionsService Properties

The properties of the TCustomSessionsService class are listed here.

Public Properties

	Name	Description
	DefaultTimeout (see page 25580)	Specifies the number of minutes a session can persist without being accessed.
	MaxSessions (see page 25581)	Indicates the maximum number of sessions the TCustomSessionsService (see page 25570) can keep in memory.

187.1.1.4.1 TCustomSessionsService.DefaultTimeout Property

Specifies the number of minutes a session can persist without being accessed.

Pascal

```
property DefaultTimeout: Integer;
```

C++

```
__property int DefaultTimeout;
```

Description

Each session that the sessions service manages has a time-out period. When a session is not accessed within that time-out period, the sessions service automatically terminates the session, discarding its session ID and name/value pairs.

Set DefaultTimeout to specify the time-out period, in minutes, for the sessions in the WebSnap application. Individual sessions can override this value by setting their TimeoutMinutes property.

See Also

TimeoutMinutes

187.1.1.4.2 TCustomSessionsService.MaxSessions Property

Indicates the maximum number of sessions the TCustomSessionsService (see page 25570) can keep in memory.

Pascal

```
property MaxSessions: Integer;
```

C++

```
__property int MaxSessions;
```

Description

Set MaxSessions to limit the number of sessions that the sessions service stores in memory. If MaxSessions is –1, then there is no limit to the number of sessions, and the application can continue creating sessions until it runs out of memory. When MaxSessions is greater than or equal to zero, any attempt to create a new session when the sessions service has MaxSessions already stored in memory causes an ESessionsException exception. The Exception occurs before the OnStartSession (see page 25582) event.

Note: When the application terminates sessions, it frees up memory for new sessions. Thus, after reaching the limit imposed by MaxSessions, the application can add subsequent sessions after explicitly terminating a session or waiting for one to time out.

See Also

ESessionsException (see page 19012)

OnStartSession (see page 25582)

Terminate (see page 24973)

187.1.1.5 TCustomSessionsService Events

The events of the TCustomSessionsService class are listed here.

Public Events

	Name	Description
	OnEndSession (see page 25581)	Occurs when the sessions service detects that a session has terminated.
	OnStartSession (see page 25582)	Occurs when the sessions service detects that a new session has started.

187.1.1.5.1 TCustomSessionsService.OnEndSession Event

Occurs when the sessions service detects that a session has terminated.

Pascal

```
property OnEndSession: TEndSessionEvent;
```

C++

```
__property TEndSessionEvent OnEndSession;
```

Description

Write an OnEndSession event handler to take specific action when a session terminates. For example, an OnEndSession event handler could store information from sessions that time out in a more persistent form such as a database.

See Also

OnStartSession (see page 25582)

Terminate (see page 24973)

TimeoutMinutes

187.1.1.5.2 TCustomSessionsService.OnStartSession Event

Occurs when the sessions service detects that a new session has started.

Pascal

```
property OnStartSession: TStartSessionEvent;
```

C++

```
__property TStartSessionEvent OnStartSession;
```

Description

Write an OnStartSession event handler to take specific action when a session starts. For example, you can use this event to customize the time-out period for new sessions or initialize them by adding name/value pairs.

See Also

OnStartSession

187.1.2 TSessionsService Class

TSessionsService stores end user data that is needed for a short period of time.

Class Hierarchy

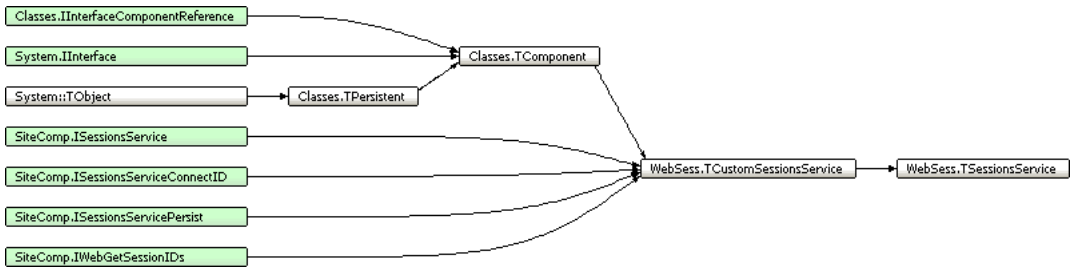

Pascal

```
TSessionsService = class(TCustomSessionsService);
```

C++

```
class TSessionsService : public TCustomSessionsService;
```

File

WebSess

Description

Add TSessionsService to a WebSnap application when you want to store temporary end user data in a WebSnap session. For example, TSessionsService can keep track of all users that are currently logged in and automatically log a user out after a period of inactivity.

When a WebSnap application contains a TSessionsService component, each HTTP request is associated with a session. The WebSnap application can manage multiple sessions on behalf of different users, each on a separate thread of execution. When handling a request, only one session is available, which is called the current session. The current session is available as the Session property of the global Web context, which is accessed using the global WebContext function.

TSessionsService stores session IDs and name/value pairs for all sessions in memory. This information must remain in memory between requests. Web application types that do not remain in memory between requests, such as CGI, do not support TSessionsService. When testing Web App Debugger executables, be sure to start the executables before browsing so that the executable remains running between requests.

Use TEndUserSessionsAdapter and TSessionsAdapter together to keep track of end user name and login state.

See Also

TEndUserSessionAdapter (see page 24458)

TAbstractWebContext (see page 24644)

TWebSession (see page 24968)

WebContext

Adding Login Support

Using the Sessions Service

187.1.2.1 TSessionsService Members

The following tables list the members exposed by TSessionsService.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckStatusChange (see page 25575)	Checks whether any sessions have started or terminated.
	Clear (see page 25575)	Deletes all sessions from memory.
	Connect (see page 25575)	Obtains the interface for a Web session, given its Session ID.
	ConnectActive (see page 25576)	Obtains the interface for an active Web session, given its Session ID.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoEndSession (see page 25576)	Triggers the OnEndSession (see page 25581) event when appropriate.
V	DoStartSession (see page 25577)	Triggers the OnStartSession (see page 25582) event when appropriate.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetSessionIDs (see page 25577)	Returns a list of the session IDs for all sessions currently in memory.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).

	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	LoadFromFile (see page 25577)	Loads the list of sessions from a file.
	LoadFromStream (see page 25578)	Loads the list of sessions from a stream.
	NewSession (see page 25578)	Starts a new session.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	NotifyStatusChange (see page 25579)	Generates an OnEndSession (see page 25581) or OnStartSession (see page 25582) event, as approrpriate.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SaveToFile (see page 25579)	Saves the list of sessions to a file.
	SaveToStream (see page 25580)	Saves the list of sessions to a stream.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckStatusChange (see page 19734)	Checks whether any sessions have started or terminated.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 19734)	Deletes all sessions from memory.
	Connect (see page 19736)	This is Connect, a member of class ISessionsServiceConnectID.
	ConnectActive (see page 19737)	This is ConnectActive, a member of class ISessionsServiceConnectID.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.

	Create (see page 25574)	Creates and initializes an instance of TCustomSessionsService (see page 25570).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetSessionIDs (see page 19764)	This is GetSessionIDs, a member of class IWebGetSessionIDs.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LoadFromFile (see page 19738)	This is LoadFromFile, a member of class ISessionsServicePersist.
	LoadFromStream (see page 19738)	This is LoadFromStream, a member of class ISessionsServicePersist.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NewSession (see page 19735)	Starts a new session.
	NotifyStatusChange (see page 19735)	Generates an OnEndSession or OnStartSession event, as approrpriate.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 19738)	This is SaveToFile, a member of class ISessionsServicePersist.
	SaveToStream (see page 19738)	This is SaveToStream, a member of class ISessionsServicePersist.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnEndSession (see page 25587)	Occurs when the sessions service detects that a session has terminated.
	OnStartSession (see page 25588)	Occurs when the sessions service detects that a new session has started.

Published Properties

	Name	Description
	DefaultTimeout (see page 25586)	Specifies the number of minutes a session can persist without being accessed.
	MaxSessions (see page 25587)	Indicates the maximum number of sessions the TCustomSessionsService can keep in memory.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

187.1.2.2 TSessionsService Properties

The properties of the TSessionsService class are listed here.

Published Properties

	Name	Description
	DefaultTimeout (see page 25586)	Specifies the number of minutes a session can persist without being accessed.
	MaxSessions (see page 25587)	Indicates the maximum number of sessions the TCustomSessionsService can keep in memory.

187.1.2.2.1 TSessionsService.DefaultTimeout Property

Specifies the number of minutes a session can persist without being accessed.

Pascal

```
property DefaultTimeout: Integer;
```

C++

```
__property int DefaultTimeout;
```

Description

Each session that the sessions service manages has a time-out period. When a session is not accessed within that time-out period, the sessions service automatically terminates the session, discarding its session ID and name/value pairs.

Set DefaultTimeout to specify the time-out period, in minutes, for the sessions in the WebSnap application. Individual sessions can override this value by setting their TimeoutMinutes property.

See Also

TimeoutMinutes

187.1.2.2.2 TSessionsService.MaxSessions Property

Indicates the maximum number of sessions the TCustomSessionsService can keep in memory.

Pascal

```
property MaxSessions: Integer;
```

C++

```
__property int MaxSessions;
```

Description

Set MaxSessions to limit the number of sessions that the sessions service stores in memory. If MaxSessions is –1, then there is no limit to the number of sessions, and the application can continue creating sessions until it runs out of memory. When MaxSessions is greater than or equal to zero, any attempt to create a new session when the sessions service has MaxSessions already stored in memory causes an ESessionsException exception. The Exception occurs before the OnStartSession event.

Note: When the application terminates sessions, it frees up memory for new sessions. Thus, after reaching the limit imposed by MaxSessions, the application can add subsequent sessions after explicitly terminating a session or waiting for one to time out.

See Also

ESessionsException (see page 19012)

OnStartSession (see page 25582)

Terminate (see page 24973)

187.1.2.3 TSessionsService Events

The events of the TSessionsService class are listed here.

Published Events

	Name	Description
	OnEndSession (see page 25587)	Occurs when the sessions service detects that a session has terminated.
	OnStartSession (see page 25588)	Occurs when the sessions service detects that a new session has started.

187.1.2.3.1 TSessionsService.OnEndSession Event

Occurs when the sessions service detects that a session has terminated.

Pascal

```
property OnEndSession: TEndSessionEvent;
```

C++

```
__property TEndSessionEvent OnEndSession;
```

Description

Write an OnEndSession event handler to take specific action when a session terminates. For example, an OnEndSession event handler could store information from sessions that time out in a more persistent form such as a database.

See Also

OnStartSession (see page 25582)

Terminate (see page 24973)

TimeoutMinutes

187.1.2.3.2 TSessionsService.OnStartSession Event

Occurs when the sessions service detects that a new session has started.

Pascal

```
property OnStartSession: TStartSessionEvent;
```

C++

```
__property TStartSessionEvent OnStartSession;
```

Description

Write an OnStartSession event handler to take specific action when a session starts. For example, you can use this event to customize the time-out period for new sessions or initialize them by adding name/value pairs.

See Also

OnStartSession (see page 25582)

187.1.3 TWebSessionIDsImpl Class

Class Hierarchy

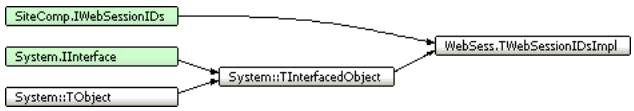

Pascal

```
TWebSessionIDsImpl = class(TInterfacedObject, IWebSessionIDs);
```

C++

```
class TWebSessionIDsImpl : public TInterfacedObject, public IWebSessionIDs;
```

File

WebSess

Description

This is class WebSess.TWebSessionIDsImpl.

187.1.3.1 TWebSessionIDsImpl Members

The following tables list the members exposed by TWebSessionIDsImpl.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetID (see page 25591)	Returns the value of the IDs property.
	GetIDCount (see page 25591)	Returns the value of the IDCount property.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25590)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25590)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	IDCount (see page 19779)	Indicates the number of session IDs (see page 19779) available.
R	IDs (see page 19779)	Lists the session IDs for all the sessions stored by the sessions service.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

187.1.3.2 TWebSessionIDsImpl.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TWebSessionIDsImpl();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 22287)

System::TObject::Free (see page 22300)

187.1.3.3 TWebSessionIDsImpl.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebSessionIDsImpl();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the

object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

187.1.3.4 TWebSessionIDsImpl Methods

The methods of the TWebSessionIDsImpl class are listed here.

Protected Methods

	Name	Description
	GetID (see page 25591)	Returns the value of the IDs property.
	GetIDCount (see page 25591)	Returns the value of the IDCount property.

187.1.3.4.1 TWebSessionIDsImpl.GetID Method

Returns the value of the IDs property.

Pascal

```
function GetID(I: Integer): TSessionID;
```

C++

```
__fastcall TSessionID GetID(int I);
```

Description

GetID is the read implementation of the IDs property.

I is the index of the session ID to return.

See Also

IDs (see page 19779)

187.1.3.4.2 TWebSessionIDsImpl.GetIDCount Method

Returns the value of the IDCount property.

Pascal

```
function GetIDCount: Integer;
```

C++

```
__fastcall int GetIDCount();
```

Description

GetIDCount is the read implementation of the IDCount property.

See Also

IDCount (see page 19779)

187.1.4 TWebSessionImpl Class

Class Hierarchy

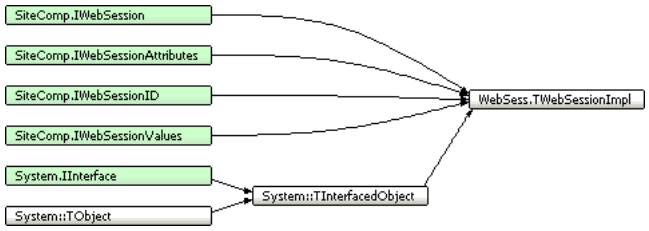

Pascal

```
TWebSessionImpl = class(TInterfacedObject, IWebSession, IWebSessionID, IWebSessionValues,
IWebSessionAttributes);
```

C++

```
class TWebSessionImpl : public TInterfacedObject, public IWebSession, public IWebSessionID,
public IWebSessionValues, public IWebSessionAttributes;
```

File

WebSess

Description

This is class WebSess.TWebSessionImpl.

187.1.4.1 TWebSessionImpl Members

The following tables list the members exposed by TWebSessionImpl.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	GetExpirationTime (see page 25595)	This is GetExpirationTime, a member of class TWebSessionImpl.
	GetExpired (see page 25595)	This is GetExpired, a member of class TWebSessionImpl.
	GetID (see page 25595)	This is GetID, a member of class TWebSessionImpl.
	GetIsActive (see page 25596)	This is GetIsActive, a member of class TWebSessionImpl.
	GetItemCount (see page 25596)	This is GetItemCount, a member of class TWebSessionImpl.
	GetItems (see page 25596)	This is GetItems, a member of class TWebSessionImpl.
	GetStartTime (see page 25596)	This is GetStartTime, a member of class TWebSessionImpl.
	GetTerminated (see page 25597)	This is GetTerminated, a member of class TWebSessionImpl.
	GetTimeoutMinutes (see page 25597)	This is GetTimeoutMinutes, a member of class TWebSessionImpl.
	GetTouchTime (see page 25597)	This is GetTouchTime, a member of class TWebSessionImpl.
	GetValue (see page 25597)	This is GetValue, a member of class TWebSessionImpl.

	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	SetTimeoutMinutes (see page 25597)	This is SetTimeoutMinutes, a member of class TWebSessionImpl.
	SetValue (see page 25598)	This is SetValue, a member of class TWebSessionImpl.
	Terminate (see page 25598)	Terminates the session.

Protected Properties

	Name	Description
R	ExpirationTime (see page 25599)	This is ExpirationTime, a member of class TWebSessionImpl.
R	Expired (see page 25599)	This is Expired, a member of class TWebSessionImpl.
R	ID (see page 25599)	This is ID, a member of class TWebSessionImpl.
R	StartTime (see page 25599)	This is StartTime, a member of class TWebSessionImpl.
R	Terminated (see page 25599)	This is Terminated, a member of class TWebSessionImpl.
R	TimeoutMinutes (see page 25600)	This is TimeoutMinutes, a member of class TWebSessionImpl.
R	TouchTime (see page 25600)	This is TouchTime, a member of class TWebSessionImpl.
	Values (see page 25600)	This is Values, a member of class TWebSessionImpl.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 25594)	This is Create, a member of class TWebSessionImpl.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25594)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	IsActive (see page 19775)	This is IsActive, a member of class IWebSessionAttributes.
	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

187.1.4.2 TWebSessionImpl.Create Constructor

Pascal

```
constructor Create(AID: TSessionID);
```

C++

```
__fastcall TWebSessionImpl(TSessionID AID);
```

Description

This is Create, a member of class TWebSessionImpl.

187.1.4.3 TWebSessionImpl.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebSessionImpl();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

187.1.4.4 TWebSessionImpl Methods

The methods of the TWebSessionImpl class are listed here.

Protected Methods

	Name	Description
	GetExpirationTime (see page 25595)	This is GetExpirationTime, a member of class TWebSessionImpl.
	GetExpired (see page 25595)	This is GetExpired, a member of class TWebSessionImpl.
	GetID (see page 25595)	This is GetID, a member of class TWebSessionImpl.
	GetIsActive (see page 25596)	This is GetIsActive, a member of class TWebSessionImpl.
	GetItemCount (see page 25596)	This is GetItemCount, a member of class TWebSessionImpl.
	GetItems (see page 25596)	This is GetItems, a member of class TWebSessionImpl.
	GetStartTime (see page 25596)	This is GetStartTime, a member of class TWebSessionImpl.
	GetTerminated (see page 25597)	This is GetTerminated, a member of class TWebSessionImpl.
	GetTimeoutMinutes (see page 25597)	This is GetTimeoutMinutes, a member of class TWebSessionImpl.
	GetTouchTime (see page 25597)	This is GetTouchTime, a member of class TWebSessionImpl.
	GetValue (see page 25597)	This is GetValue, a member of class TWebSessionImpl.
	SetTimeoutMinutes (see page 25597)	This is SetTimeoutMinutes, a member of class TWebSessionImpl.
	SetValue (see page 25598)	This is SetValue, a member of class TWebSessionImpl.
	Terminate (see page 25598)	Terminates the session.

187.1.4.4.1 TWebSessionImpl.GetExpirationTime Method

Pascal

```
function GetExpirationTime: TDateTime;
```

C++

```
__fastcall TDateTime GetExpirationTime();
```

Description

This is GetExpirationTime, a member of class TWebSessionImpl.

187.1.4.4.2 TWebSessionImpl.GetExpired Method

Pascal

```
function GetExpired: Boolean;
```

C++

```
__fastcall Boolean GetExpired();
```

Description

This is GetExpired, a member of class TWebSessionImpl.

187.1.4.4.3 TWebSessionImpl.GetID Method

Pascal

```
function GetID: TSessionID;
```

C++

```
__fastcall TSessionID GetID();
```

Description

This is GetID, a member of class TWebSessionImpl.

187.1.4.4.4 TWebSessionImpl.GetIsActive Method

Pascal

```
function GetIsActive: Boolean;
```

C++

```
__fastcall Boolean GetIsActive();
```

Description

This is GetIsActive, a member of class TWebSessionImpl.

187.1.4.4.5 TWebSessionImpl.GetItemCount Method

Pascal

```
function GetItemCount: Integer;
```

C++

```
__fastcall int GetItemCount();
```

Description

This is GetItemCount, a member of class TWebSessionImpl.

187.1.4.4.6 TWebSessionImpl.GetItems Method

Pascal

```
procedure GetItems(var AItems: TAbstractNamedVariants);
```

C++

```
__fastcall GetItems(TAbstractNamedVariants AItems);
```

Description

This is GetItems, a member of class TWebSessionImpl.

187.1.4.4.7 TWebSessionImpl.GetStartTime Method

Pascal

```
function GetStartTime: TDateTime;
```

C++

```
__fastcall TDateTime GetStartTime();
```

Description

This is GetStartTime, a member of class TWebSessionImpl.

187.1.4.4.8 TWebSessionImpl.GetTerminated Method

Pascal

```
function GetTerminated: Boolean;
```

C++

```
__fastcall Boolean GetTerminated();
```

Description

This is GetTerminated, a member of class TWebSessionImpl.

187.1.4.4.9 TWebSessionImpl.GetTimeoutMinutes Method

Pascal

```
function GetTimeoutMinutes: Integer;
```

C++

```
__fastcall int GetTimeoutMinutes();
```

Description

This is GetTimeoutMinutes, a member of class TWebSessionImpl.

187.1.4.4.10 TWebSessionImpl.GetTouchTime Method

Pascal

```
function GetTouchTime: TDateTime;
```

C++

```
__fastcall TDateTime GetTouchTime();
```

Description

This is GetTouchTime, a member of class TWebSessionImpl.

187.1.4.4.11 TWebSessionImpl.GetValue Method

Pascal

```
function GetValue(const AName: string): Variant;
```

C++

```
__fastcall Variant GetValue(const AnsiString AName);
```

Description

This is GetValue, a member of class TWebSessionImpl.

187.1.4.4.12 TWebSessionImpl.SetTimeoutMinutes Method

Pascal

```
procedure SetTimeoutMinutes(AValue: Integer);
```

C++

```
__fastcall SetTimeoutMinutes(int AValue);
```

Description

This is SetTimeoutMinutes, a member of class TWebSessionImpl.

187.1.4.4.13 TWebSessionImpl.SetValue Method

Pascal

```
procedure SetValue(const AName: string; const AValue: Variant);
```

C++

```
__fastcall SetValue(const AnsiString AName, const Variant AValue);
```

Description

This is SetValue, a member of class TWebSessionImpl.

187.1.4.4.14 TWebSessionImpl.Terminate Method

Terminates the session.

Pascal

```
procedure Terminate;
```

C++

```
__fastcall Terminate();
```

Description

Call Terminate to terminate a session. Once a session terminates, the application can no longer access its Values.

Sessions are terminated automatically when the application does not receive any requests for the session for TimeOutMinutes minutes.

See Also

TimeoutMinutes

Values

187.1.4.5 TWebSessionImpl Properties

The properties of the TWebSessionImpl class are listed here.

Protected Properties

	Name	Description
R	ExpirationTime (see page 25599)	This is ExpirationTime, a member of class TWebSessionImpl.
R	Expired (see page 25599)	This is Expired, a member of class TWebSessionImpl.
R	ID (see page 25599)	This is ID, a member of class TWebSessionImpl.
R	StartTime (see page 25599)	This is StartTime, a member of class TWebSessionImpl.
R	Terminated (see page 25599)	This is Terminated, a member of class TWebSessionImpl.
R	TimeoutMinutes (see page 25600)	This is TimeoutMinutes, a member of class TWebSessionImpl.
R	TouchTime (see page 25600)	This is TouchTime, a member of class TWebSessionImpl.
	Values (see page 25600)	This is Values, a member of class TWebSessionImpl.

187.1.4.5.1 TWebSessionImpl.ExpirationTime Property

Pascal

```
property ExpirationTime: TDateTime;
```

C++

```
__property TDateTime ExpirationTime;
```

Description

This is ExpirationTime, a member of class TWebSessionImpl.

187.1.4.5.2 TWebSessionImpl.Expired Property

Pascal

```
property Expired: Boolean;
```

C++

```
__property Boolean Expired;
```

Description

This is Expired, a member of class TWebSessionImpl.

187.1.4.5.3 TWebSessionImpl.ID Property

Pascal

```
property ID: TSessionID;
```

C++

```
__property TSessionID ID;
```

Description

This is ID, a member of class TWebSessionImpl.

187.1.4.5.4 TWebSessionImpl.StartTime Property

Pascal

```
property StartTime: TDateTime;
```

C++

```
__property TDateTime StartTime;
```

Description

This is StartTime, a member of class TWebSessionImpl.

187.1.4.5.5 TWebSessionImpl.Terminated Property

Pascal

```
property Terminated: Boolean;
```

C++

```
__property Boolean Terminated;
```

Description

This is Terminated, a member of class TWebSessionImpl.

187.1.4.5.6 TWebSessionImpl.TimeoutMinutes Property

Pascal

```
property TimeoutMinutes: Integer;
```

C++

```
__property int TimeoutMinutes;
```

Description

This is TimeoutMinutes, a member of class TWebSessionImpl.

187.1.4.5.7 TWebSessionImpl.TouchTime Property

Pascal

```
property TouchTime: TDateTime;
```

C++

```
__property TDateTime TouchTime;
```

Description

This is TouchTime, a member of class TWebSessionImpl.

187.1.4.5.8 TWebSessionImpl.Values Property

Pascal

```
property Values [const AName: string]: Variant;
```

C++

```
__property Variant Values[AnsiString const AName];
```

Description

This is Values, a member of class TWebSessionImpl.

187.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TEndSessionReason (see page 25601)	This is record WebSess.TEndSessionReason.

187.2.1 WebSess.TEndSessionReason Enumeration

Pascal

```
TEndSessionReason = (
  esTimeout,
  esTerminate
);
```

C++

```
enum TEndSessionReason {
  esTimeout,
  esTerminate
};
```

File

WebSess

Description

This is record WebSess.TEndSessionReason.

187.3 Types

The following table lists types in this documentation.

Types

Name	Description
TEndSessionEvent (see page 25602)	This is type WebSess.TEndSessionEvent.
TStartSessionEvent (see page 25602)	This is type WebSess.TStartSessionEvent.

187.3.1 WebSess.TEndSessionEvent Type

Pascal

```
TEndSessionEvent = procedure (ASender: TObject; ASession: TAbstractWebSession; AReason:
TEndSessionReason) of object;
```

C++

```
(ASender: TObject; ASession: TAbstractWebSession; AReason: TEndSessionReason) (
TEndSessionEvent)();
```

File

WebSess

Description

This is type WebSess.TEndSessionEvent.

187.3.2 WebSess.TStartSessionEvent Type

Pascal

```
TStartSessionEvent = procedure (ASender: TObject; ASession: TAbstractWebSession) of object;
```

C++

```
(ASender: TObject; ASession: TAbstractWebSession) ( TStartSessionEvent)();
```

File

WebSess

Description

This is type WebSess.TStartSessionEvent.

188 WebUsers Namespace

This is namespace WebUsers.

Topics

Name	Description
Classes (see page 25604)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 25633)	The following table lists structs, records, enums in this documentation.
Types (see page 25634)	The following table lists types in this documentation.

Classes

	Name	Description
	EUserIDNotFoundException (see page 25604)	EUserIDNotFoundException is raised when a Web user list attempts to locate information about a user that is not in the list.
	EValidateUserException (see page 25604)	EValidateUserException is raised when a Web user list fails to validate a user.
	EWebUsersListException (see page 25605)	EWebUsersListException is the base class for exceptions that TWebUserList (see page 25625) raises.
	TCustomWebUserList (see page 25606)	The TCustomWebUserList component uses a TWebUserItems (see page 25620) object to store the account information about a collection of users.
	TWebUserItem (see page 25615)	TWebUserItem stores the user identification, password, and access rights for a single user.
	TWebUserItems (see page 25620)	TWebUserItems stores multiple sets of user information.
	TWebUserList (see page 25625)	TWebUserList validates login and checks access rights for a particular user.

Structs, Records, Enums

	Name	Description
	TValidateUserError (see page 25633)	This is record WebUsers.TValidateUserError.

Types

Name	Description
TCheckAccessRightsEvent (see page 25634)	This is type WebUsers.TCheckAccessRightsEvent.
TCheckAccessRightsHandledEvent (see page 25634)	This is type WebUsers.TCheckAccessRightsHandledEvent.
TValidateUserErrorEvent (see page 25635)	This is type WebUsers.TValidateUserErrorEvent.
TValidateUserEvent (see page 25635)	This is type WebUsers.TValidateUserEvent.
TValidateUserHandledEvent (see page 25635)	This is type WebUsers.TValidateUserHandledEvent.

188.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EUserIDNotFoundException (see page 25604)	EUserIDNotFoundException is raised when a Web user list attempts to locate information about a user that is not in the list.
	EValidateUserException (see page 25604)	EValidateUserException is raised when a Web user list fails to validate a user.
	EWebUsersListException (see page 25605)	EWebUsersListException is the base class for exceptions that TWebUserList (see page 25625) raises.
	TCustomWebUserList (see page 25606)	The TCustomWebUserList component uses a TWebUserItems (see page 25620) object to store the account information about a collection of users.
	TWebUserItem (see page 25615)	TWebUserItem stores the user identification, password, and access rights for a single user.
	TWebUserItems (see page 25620)	TWebUserItems stores multiple sets of user information.
	TWebUserList (see page 25625)	TWebUserList validates login and checks access rights for a particular user.

188.1.1 EUserIDNotFoundException Class

EUserIDNotFoundException is raised when a Web user list attempts to locate information about a user that is not in the list.

Class Hierarchy

Pascal

```
EUserIDNotFoundException = class(EWebUsersListException);
```

C++

```
class EUserIDNotFoundException : public EWebUsersListException;
```

File

WebUsers

Description

When a Web user list (TWebUserList (see page 25625)) attempts to look up information about a user and fails to find an entry for that user, it generates an OnUserIDNotFound event. If that event handler does not handle the situation, then the Web user list raises an EUserIDNotFoundException exception.

See Also

TWebUserList (see page 25625)

TCustomWebUserList (see page 25606)

188.1.2 EValidateUserException Class

EValidateUserException is raised when a Web user list fails to validate a user.

Class Hierarchy

System::TObject → SysUtils.Exception → HTTPApp.EWebBrokerException → WebUsers.EWebUsersListException → WebUsers.EValidateUserException

Pascal

```
EValidateUserException = class(EWebUsersListException);
```

C++

```
class EValidateUserException : public EWebUsersListException;
```

File

WebUsers

Description

When a WebSnap application that contains a Web user list (TWebUserList (see page 25625)) tries to log a user in, the Web user list raises an EValidateUserException if it can't validate the supplied user ID and password and its OnValidateUserError event handler does not handle the situation.

The Web user list may fail to validate a user ID and password for any of the following reasons:

The WebSnap application failed to supply a password or user ID when trying to validate the user.

The supplied user ID does not refer to a user in the list.

The supplied password is not correct for the specified user.

See Also

TWebUserList (see page 25625)

TCustomWebUserList (see page 25606)

TCustomWebUserList (see page 25606)

188.1.3 EWebUsersListException Class

EWebUsersListException is the base class for exceptions that TWebUserList (see page 25625) raises.

Class Hierarchy

System::TObject → SysUtils.Exception → HTTPApp.EWebBrokerException → WebUsers.EWebUsersListException

Pascal

```
EWebUsersListException = class(EWebBrokerException);
```

C++

```
class EWebUsersListException : public EWebBrokerException;
```

File

WebUsers

Description

EWebUsersListException is the common base class for all exceptions that represent errors which TWebUserList (see page 25625) encounters. Its descendants include

EUserIDNotFoundException (see page 25604), which is raised when a WebSnap application tries to use the Web user list to look up information about a user that is not listed.

EValidateUserException (see page 25604), which is raised when the WebSnap application tries to log in a user and the Web user list fails to validate the supplied user id and password.

See Also

TWebUserList (see page 25625)

EUserIDNotFoundException (see page 25604)

EValidateUserException (see page 25604)

188.1.4 TCustomWebUserList Class

The TCustomWebUserList component uses a TWebUserItems (see page 25620) object to store the account information about a collection of users.

Class Hierarchy

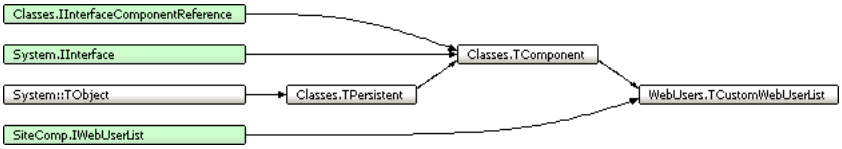

Pascal

```
TCustomWebUserList = class(TComponent, IWebUserList);
```

C++

```
class TCustomWebUserList : public TComponent, public IWebUserList;
```

File

WebUsers

Description

The TCustomWebUserList component uses a TWebUserItems (see page 25620) object to store the account information about a collection of users. It also provides a way to save and load the user list from a file. Several events are triggered at stages that occur when a user is logging in that allow checks to be performed during the user validation process, including the verification of user rights.

See Also

TWebUserItems (see page 25620)

TWebUserList (see page 25625)

188.1.4.1 TCustomWebUserList Members

The following tables list the members exposed by TCustomWebUserList.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckAccessRights (see page 25610)	Checks whether a specified user has a given access rights string.
	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateUser (see page 25611)	Verifies that the given user name and password matches one of the entries stored in the TCustomWebUserList (see page 25606) object.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Events

	Name	Description
	OnAfterCheckAccessRights (see page 25613)	Triggered at the start of the CheckAccessRights (see page 25610) protected function.
	OnAfterValidateUser (see page 25613)	Occurs after a user has been validated.

	OnBeforeCheckAccessRights (see page 25613)	Triggered at the end of the CheckAccessRights (see page 25610) protected function.
	OnBeforeValidateUser (see page 25614)	Triggered at the beginning of the ValidateUser (see page 25611) function.
	OnUserIDNotFound (see page 25614)	Triggered if the UserID is not found in the TCustomWebUserList (see page 25606) object.
	OnValidateUserError (see page 25614)	Triggered if an error occurs during the validation process.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 25609)	Creates a newTCustomWebUserList object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25610)	Frees the TCustomWebUserList (see page 25606) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LoadFromFile (see page 25611)	Loads the data from the file with the given filename.

	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 25611)	This procedure saves the collection of the user's account information to a file with the given filename.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	UserItems (see page 25612)	Stores the account information about a collection of users.
R	Users (see page 25612)	Provides read-only access to an individual user's account data in the form of an array of TWebUserItem (see page 25615).
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.

188.1.4.2 TCustomWebUserList.Create Constructor

Creates a newTCustomWebUserList object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomWebUserList(TComponent * AOwner);
```

Description

Creates a newTCustomWebUserList object and initializes the TWebUserItems (see page 25620) object that will store the user's account information.

188.1.4.3 TCustomWebUserList.Destroy Destructor

Frees the TCustomWebUserList (see page 25606) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWebUserList();
```

Description

Frees the TCustomWebUserList (see page 25606) object initialized by the constructor.

188.1.4.4 TCustomWebUserList Methods

The methods of the TCustomWebUserList class are listed here.

Protected Methods

	Name	Description
	CheckAccessRights (see page 25610)	Checks whether a specified user has a given access rights string.
	ValidateUser (see page 25611)	Verifies that the given user name and password matches one of the entries stored in the TCustomWebUserList (see page 25606) object.

Public Methods

	Name	Description
	LoadFromFile (see page 25611)	Loads the data from the file with the given filename.
	SaveToFile (see page 25611)	This procedure saves the collection of the user's account information to a file with the given filename.

188.1.4.4.1 TCustomWebUserList.CheckAccessRights Method

Checks whether a specified user has a given access rights string.

Pascal

```
function CheckAccessRights(AUserID: Variant; ARights: string): Boolean;
```

C++

```
__fastcall Boolean CheckAccessRights(Variant AUserID, AnsiString ARights);
```

Description

Call CheckAccessRights to determine whether the current end user is authorized to view Web pages or adapters whose access rights string is Rights. The end user must be logged in before CheckAccessRights can determine the user's access rights.

UserID is the user ID of the user to check.

Rights is the access rights string that CheckAccessRights must look for. Typically, this is the value of the ViewRights property on a Web page information object that describes the page the user wants to access or on an adapter that the user wants to call.

See Also

HasViewRights (see page 24112)

HasModifyRights (see page 24112)

HasExecuteRights (see page 24111)

ViewAccess (see page 15069)

ViewAccess (see page 24183)

188.1.4.4.2 TCustomWebUserList.LoadFromFile Method

Loads the data from the file with the given filename.

Pascal

```
procedure LoadFromFile(FileName: string);
```

C++

```
__fastcall LoadFromFile(AnsiString FileName);
```

Description

Loads the data from the file with the given filename into the TCustomWebUserList (see page 25606) object.

188.1.4.4.3 TCustomWebUserList.SaveToFile Method

This procedure saves the collection of the user's account information to a file with the given filename.

Pascal

```
procedure SaveToFile(FileName: string);
```

C++

```
__fastcall SaveToFile(AnsiString FileName);
```

Description

Saves the collection of the user's account information to a file with the given filename. This file can later be reloaded by calling LoadFromFile (see page 25611).

188.1.4.4.4 TCustomWebUserList.ValidateUser Method

Verifies that the given user name and password matches one of the entries stored in the TCustomWebUserList (see page 25606) object.

Pascal

```
function ValidateUser(Strings: TStrings): Variant;
```

C++

```
__fastcall Variant ValidateUser(TStrings Strings);
```

Description

This function verifies that the given user name and password matches one of the entries stored in the TCustomWebUserList (see page 25606) object. Before the validation begins, it triggers the OnBeforeValidateUser (see page 25614) event. If an error occurs during the validation check, the OnValidateUserError (see page 25614) event will be triggered and a description of the error is sent, which could be either a missing or invalid password, or a missing or unknown user name. After the validation is complete, this function triggers the OnAfterValidateUser (see page 25613) event.

188.1.4.5 TCustomWebUserList Properties

The properties of the TCustomWebUserList class are listed here.

Public Properties

	Name	Description
	UserItems (see page 25612)	Stores the account information about a collection of users.
R	Users (see page 25612)	Provides read-only access to an individual user's account data in the form of an array of TWebUserItem (see page 25615).

188.1.4.5.1 TCustomWebUserList.UserItems Property

Stores the account information about a collection of users.

Pascal

```
property UserItems: TWebUserItems;
```

C++

```
__property TWebUserItems UserItems;
```

Description

Use this object to store the account information about a collection of users. This property provides read and write access to the stored collection.

See Also

Users (see page 25612)

188.1.4.5.2 TCustomWebUserList.Users Property

Provides read-only access to an individual user's account data in the form of an array of TWebUserItem (see page 25615).

Pascal

```
property Users [I: Integer]: TWebUserItem;
```

C++

```
__property TWebUserItem Users[int I];
```

Description

Use this property to provide read-only access to an individual user's account data in the form of an array of TWebUserItem (see page 25615). An individual TWebUserItem (see page 25615) object can be indexed from the array using the standard bracketed array notation.

See Also

UserItems (see page 25612)

188.1.4.6 TCustomWebUserList Events

The events of the TCustomWebUserList class are listed here.

Public Events

	Name	Description
	OnAfterCheckAccessRights (see page 25613)	Triggered at the start of the CheckAccessRights (see page 25610) protected function.
	OnAfterValidateUser (see page 25613)	Occurs after a user has been validated.
	OnBeforeCheckAccessRights (see page 25613)	Triggered at the end of the CheckAccessRights (see page 25610) protected function.
	OnBeforeValidateUser (see page 25614)	Triggered at the beginning of the ValidateUser (see page 25611) function.

	OnUserIDNotFound (see page 25614)	Triggered if the UserID is not found in the TCustomWebUserList (see page 25606) object.
	OnValidateUserError (see page 25614)	Triggered if an error occurs during the validation process.

188.1.4.6.1 TCustomWebUserList.OnAfterCheckAccessRights Event

Triggered at the start of the CheckAccessRights (see page 25610) protected function.

Pascal

```
property OnAfterCheckAccessRights: TCheckAccessRightsEvent;
```

C++

```
__property TCheckAccessRightsEvent OnAfterCheckAccessRights;
```

Description

This event is triggered at the start of the CheckAccessRights (see page 25610) protected function. This event can be used if any additional checking or initializing is desired before the access rights are checked.

See Also

OnBeforeCheckAccessRights (see page 25613)

188.1.4.6.2 TCustomWebUserList.OnAfterValidateUser Event

Occurs after a user has been validated.

Pascal

```
property OnAfterValidateUser: TValidateUserEvent;
```

C++

```
__property TValidateUserEvent OnAfterValidateUser;
```

Description

This event is triggered at the end of the ValidateUser (see page 25611) function. It can be used if any additional processing needs to be done after a user's name and password are verified.

See Also

OnBeforeValidateUser (see page 25614)

ValidateUser (see page 25611)

188.1.4.6.3 TCustomWebUserList.OnBeforeCheckAccessRights Event

Triggered at the end of the CheckAccessRights (see page 25610) protected function.

Pascal

```
property OnBeforeCheckAccessRights: TCheckAccessRightsHandledEvent;
```

C++

```
__property TCheckAccessRightsHandledEvent OnBeforeCheckAccessRights;
```

Description

This event is triggered at the end of the CheckAccessRights (see page 25610) protected function. It can be used to perform your own custom checking instead of the default checking by the Web user list.

See Also

OnAfterCheckAccessRights (see page 25613)

188.1.4.6.4 TCustomWebUserList.OnBeforeValidateUser Event

Triggered at the beginning of the ValidateUser (see page 25611) function.

Pascal

```
property OnBeforeValidateUser: TValidateUserHandledEvent;
```

C++

```
__property TValidateUserHandledEvent OnBeforeValidateUser;
```

Description

This event is triggered at the beginning of the ValidateUser (see page 25611) function. It can be used to identify a user given a name and password and supply a user ID for the user, once identified.

See Also

OnAfterValidateUser (see page 25613)

188.1.4.6.5 TCustomWebUserList.OnUserIDNotFound Event

Triggered if the UserID is not found in the TCustomWebUserList (see page 25606) object.

Pascal

```
property OnUserIDNotFound: TCheckAccessRightsHandledEvent;
```

C++

```
__property TCheckAccessRightsHandledEvent OnUserIDNotFound;
```

Description

This event is triggered if the UserID is not found in the TCustomWebUserList (see page 25606) object. The CheckAccessRights (see page 25610) protected function triggers this function if the UserID is not found while checking the access rights.

188.1.4.6.6 TCustomWebUserList.OnValidateUserError Event

Triggered if an error occurs during the validation process.

Pascal

```
property OnValidateUserError: TValidateUserErrorEvent;
```

C++

```
__property TValidateUserErrorEvent OnValidateUserError;
```

Description

This event is triggered if an error occurs during the validation process. The ValidateUser (see page 25611) function calls this event with a description of the error. Examples of error descriptions include a missing or unknown user name or a missing or invalid password. Any additional processing that needs to be completed in the case of one of these error conditions can be added to this event handler.

See Also

ValidateUser (see page 25611)

188.1.5 TWebUserItem Class

TWebUserItem stores the user identification, password, and access rights for a single user.

Class Hierarchy

Pascal

```
TWebUserItem = class(TCollectionItem);
```

C++

```
class TWebUserItem : public TCollectionItem;
```

File

WebUsers

Description

TWebUserItem stores the user ID and password along with the associated access rights for a single user. Methods are provided to set the access rights as well as methods to retrieve the access rights in the form of a single delimited string, or as a collection in the form of a TString.

See Also

TWebUserList (see page 25625)

TWebUserItems (see page 25620)

188.1.5.1 TWebUserItem Members

The following tables list the members exposed by TWebUserItem.

Protected Methods

	Name	Description
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2758)	Updates the collection to reflect changes to the collection item.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetDisplayName (see page 2758)	Returns the name of the collection item as it appears in the collection editor.
	GetOwner (see page 2759)	Returns the collection to which this item belongs.
	GetOwner (see page 2932)	Returns the owner of an object.
	SetCollection (see page 2759)	Sets the value of the Collection (see page 2761) property.
	SetDisplayName (see page 2760)	Sets the name of the collection item as it appears in the collection editor.
	SetIndex (see page 2760)	Moves the collection item to a specified position in the parent collection.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 25617)	Copies the UserName (see page 25620), Password (see page 25619), and AccessRights (see page 25618) from the given TWebUserItem (see page 25615).
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckRights (see page 25618)	Checks the user's stored access rights against the access rights passed in.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 25616)	Creates an instance of Create.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 25617)	Destroys an instance of TWebUserItem (see page 25615).
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2759)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	AccessRightsStrings (see page 25619)	Stores the access rights for the user as a collection of individual strings in the form of a TStrings object.
	Collection (see page 2761)	Specifies the TCollection (see page 2731) instance to which the TCollectionItem (see page 2755) belongs.
	DisplayName (see page 2761)	The name displayed in the Collection (see page 2761) editor.
R	ID (see page 2761)	A unique, permanent index for the item.
	Index (see page 2762)	Returns the item's position in the Items array of TCollection (see page 2731).
R	UserID (see page 25619)	Provides read-only access to the user ID.

Published Properties

	Name	Description
	AccessRights (see page 25618)	Stores the user rights as a comma, semicolon, or space delimited string.
	Password (see page 25619)	Stores the user's password as a string.
	UserName (see page 25620)	Stores the user's name.

188.1.5.2 TWebUserItem.Create Constructor

Creates an instance of Create.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TWebUserItem(TCollection Collection);
```

Description

Do not call Create. It is called automatically when the Create object is created.

See Also

Create_c

188.1.5.3 TWebUserItem.Destroy Destructor

Destroys an instance of TWebUserItem (see page 25615).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebUserItem();
```

Description

The Destroy destructor frees all objects owned by the TWebUserItem (see page 25615) object. Never call the destructor explicitly; it is called automatically when the TWebUserItem (see page 25615) object is destroyed (for example, when the object falls out of the scope of the program).

See Also

delete

188.1.5.4 TWebUserItem Methods

The methods of the TWebUserItem class are listed here.

Public Methods

	Name	Description
	Assign (see page 25617)	Copies the UserName (see page 25620), Password (see page 25619), and AccessRights (see page 25618) from the given TWebUserItem (see page 25615).
	CheckRights (see page 25618)	Checks the user's stored access rights against the access rights passed in.

188.1.5.4.1 TWebUserItem.Assign Method

Copies the UserName (see page 25620), Password (see page 25619), and AccessRights (see page 25618) from the given TWebUserItem (see page 25615).

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

The Assign procedure copies the UserName (see page 25620), Password (see page 25619), and AccessRights (see page 25618) from the given TWebUserItem (see page 25615).

See Also

UserName (see page 25620)

Password (see page 25619)

AccessRights (see page 25618)

188.1.5.4.2 TWebUserItem.CheckRights Method

Checks the user's stored access rights against the access rights passed in.

Pascal

```
function CheckRights(const ARights: string): Boolean;
```

C++

```
__fastcall Boolean CheckRights(const AnsiString ARights);
```

Description

This function checks the user's stored access rights against the access rights passed in. If the user has a matching right, then the function returns true or it will return false if the rights do not match.

See Also

AccessRights (see page 25618)

188.1.5.5 TWebUserItem Properties

The properties of the TWebUserItem class are listed here.

Public Properties

	Name	Description
R	AccessRightsStrings (see page 25619)	Stores the access rights for the user as a collection of individual strings in the form of a TStrings object.
R	UserID (see page 25619)	Provides read-only access to the user ID.

Published Properties

	Name	Description
	AccessRights (see page 25618)	Stores the user rights as a comma, semicolon, or space delimited string.
	Password (see page 25619)	Stores the user's password as a string.
	UserName (see page 25620)	Stores the user's name.

188.1.5.5.1 TWebUserItem.AccessRights Property

Stores the user rights as a comma, semicolon, or space delimited string.

Pascal

```
property AccessRights: string;
```

C++

```
__property AnsiString AccessRights;
```

Description

The AccessRights property stores the user rights as a comma, semicolon, or space delimited string. This variable can be read or written directly or set as a single string using SetAccessRights. The individual rights can be obtained as a collection in the form

of a TString by calling the GetAccessRightsStrings function.

See Also

CheckRights (see page 25618)

188.1.5.5.2 TWebUserItem.AccessRightsStrings Property

Stores the access rights for the user as a collection of individual strings in the form of a TStrings object.

Pascal

```
property AccessRightsStrings: TStrings;
```

C++

```
__property TStrings AccessRightsStrings;
```

Description

The AccessRightsStrings property stores the access rights for the user as a collection of individual strings in the form of a TStrings object.

See Also

AccessRights (see page 25618)

TStrings (see page 3029)

188.1.5.5.3 TWebUserItem.Password Property

Stores the user's password as a string.

Pascal

```
property Password: string;
```

C++

```
__property AnsiString Password;
```

Description

The Password property stores the user's password as a string.

188.1.5.5.4 TWebUserItem.UserID Property

Provides read-only access to the user ID.

Pascal

```
property UserID: Variant;
```

C++

```
__property Variant UserID;
```

Description

The UserID property provides read-only access to the user ID.

See Also

Variant (see page 22307)

188.1.5.5.5 TWebUserItem.UserName Property

Stores the user's name.

Pascal

```
property UserName: string;
```

C++

```
__property AnsiString UserName;
```

Description

The UserName property stores the user's name. This property can be read or written directly.

188.1.6 TWebUserItems Class

TWebUserItems stores multiple sets of user information.

Class Hierarchy

Pascal

```
TWebUserItems = class(TCollection);
```

C++

```
class TWebUserItems : public TCollection;
```

File

WebUsers

Description

The TWebUserItems stores multiple TWebUserItem (see page 25615) modules as an array. It also provides methods to access a particular user's information by giving the user name or user ID.

188.1.6.1 TWebUserItems Members

The following tables list the members exposed by TWebUserItems.

Protected Methods

	Name	Description
	Added (see page 2736)	Responds when items are added to the collection.
	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 25623)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
	GetAttrCount (see page 25624)	Returns the number of custom attributes associated with items in the collection.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 25624)	Returns the value of a custom attribute assigned to one of the collection's items.

V	GetOwner (see page 25624)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.
V	Update (see page 2747)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 2736)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 25622)	Constructor used for the TWebUserItems (see page 25620) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	FindUserID (see page 25623)	Returns the user ID, password, and access rights information for the user.
	FindUserName (see page 25623)	Returns the user ID, password, and access rights information for the user.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 25625)	Allows public read and write access to individual TWebUserItem (see page 25615) records as if they were stored in an array.

188.1.6.2 TWebUserItems.Create Constructor

Constructor used for the TWebUserItems (see page 25620) object.

Pascal

```
constructor Create(AOwner: TComponent; ItemClass: TCollectionItemClass);
```

C++

```
__fastcall TWebUserItems(TComponent * AOwner, TCollectionItemClass ItemClass);
```

Description

The constructor used for the TWebUserItems (see page 25620) object. This method creates a new TWebUserItem (see page 25615) and sets the owner.

188.1.6.3 TWebUserItems Methods

The methods of the TWebUserItems class are listed here.

Protected Methods

	Name	Description
V	GetAttr (see page 25623)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
V	GetAttrCount (see page 25624)	Returns the number of custom attributes associated with items in the collection.
V	GetItemAttr (see page 25624)	Returns the value of a custom attribute assigned to one of the collection's items.
V	GetOwner (see page 25624)	Returns the owner of an object.

Public Methods

	Name	Description
	FindUserID (see page 25623)	Returns the user ID, password, and access rights information for the user.
	FindUserName (see page 25623)	Returns the user ID, password, and access rights information for the user.

188.1.6.3.1 TWebUserItems.FindUserID Method

Returns the user ID, password, and access rights information for the user.

Pascal

```
function FindUserID(AUserID: Variant): TWebUserItem;
```

C++

```
__fastcall TWebUserItem FindUserID(Variant AUserID);
```

Description

This method returns the user ID, password, and access rights information for the user with the given user name in the form of a TWebUserItem (see page 25615) object.

188.1.6.3.2 TWebUserItems.FindUserName Method

Returns the user ID, password, and access rights information for the user.

Pascal

```
function FindUserName(AUserName: string): TWebUserItem;
```

C++

```
__fastcall TWebUserItem FindUserName(AnsiString AUserName);
```

Description

This method returns the user ID, password, and access rights information for the user with the given user ID in the form of a TWebUserItem (see page 25615) object.

188.1.6.3.3 TWebUserItems.GetAttr Method

Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.

Pascal

```
function GetAttr(Index: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetAttr(int Index);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. The GetAttr method returns the name of an attribute.

Index identifies the attribute whose name is requested. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

As implemented by TCollection, GetAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttrCount (see page 2744)

GetItemAttr (see page 2745)

188.1.6.3.4 TWebUserItems.GetAttrCount Method

Returns the number of custom attributes associated with items in the collection.

Pascal

```
function GetAttrCount: Integer; override;
```

C++

```
virtual __fastcall int GetAttrCount();
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetAttrCount returns the number of distinct attributes assigned to each item in the collection.

As implemented in TCollection, GetAttrCount always returns 0, because TCollection defines no custom attributes.

See Also

GetAttr (see page 2743)

GetItemAttr (see page 2745)

188.1.6.3.5 TWebUserItems.GetItemAttr Method

Returns the value of a custom attribute assigned to one of the collection's items.

Pascal

```
function GetItemAttr(Index: Integer; ItemIndex: Integer): string; override;
```

C++

```
virtual __fastcall AnsiString GetItemAttr(int Index, int ItemIndex);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetItemAttr returns the value of one of these attributes for a specified item in the collection.

Index identifies which attribute's value is desired. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

ItemIndex identifies the item whose attribute value is desired. This is an index into the Items property array.

As implemented in TCollection, GetItemAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttr (see page 2743)

GetAttrCount (see page 2744)

188.1.6.3.6 TWebUserItems.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath (see page 2932)

TCollection (see page 2731)

TComponent (see page 2762)

Owner (see page 2807)

188.1.6.4 TWebUserItems Properties

The properties of the TWebUserItems class are listed here.

Public Properties

	Name	Description
	Items (see page 25625)	Allows public read and write access to individual TWebUserItem (see page 25615) records as if they were stored in an array.

188.1.6.4.1 TWebUserItems.Items Property

Allows public read and write access to individual TWebUserItem (see page 25615) records as if they were stored in an array.

Pascal

```
property Items [Index: Integer]: TWebUserItem;
```

C++

```
__property TWebUserItem Items[int Index];
```

Description

The Items property allows public read and write access to individual TWebUserItem (see page 25615) records as if they were stored in an array. It also provides methods to access a particular user's information by giving the user name or user ID.

See Also

operator_sb

188.1.7 TWebUserList Class

TWebUserList validates login and checks access rights for a particular user.

Class Hierarchy

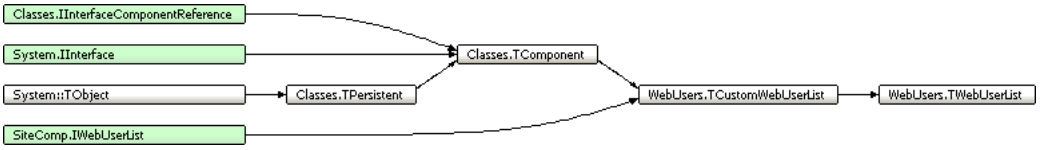

Pascal

```
TWebUserList = class(TCustomWebUserList);
```

C++

```
class TWebUserList : public TCustomWebUserList;
```

File

WebUsers

Description

Use TWebUserList, which contains a list of user names, passwords, and access rights, to validate login and to check a user's access rights.

The TWebUserList component provides events that are triggered during the initial stages when a user is logging in, before and after the user is validated, before and after the user's access rights are checked, in the case of an error, and when detecting an invalid user.

See Also

TWebUserItems (see page 25620)

TWebUserItem (see page 25615)

Adding Login Support

188.1.7.1 TWebUserList Members

The following tables list the members exposed by TWebUserList.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckAccessRights (see page 25610)	Checks whether a specified user has a given access rights string.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.

	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	ValidateUser (see page 25611)	Verifies that the given user name and password matches one of the entries stored in the TCustomWebUserList (see page 25606) object.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckAccessRights (see page 19785)	Checks whether a specified user has a given access rights string.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	Create (see page 25609)	Creates a newTCustomWebUserList object.

V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 25610)	Frees the TCustomWebUserList (see page 25606) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LoadFromFile (see page 25611)	Loads the data from the file with the given filename.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 25611)	This procedure saves the collection of the user's account information to a file with the given filename.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.
	ValidateUser (see page 19785)	Validates a user login.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.

R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	Users (see page 25612)	Provides read-only access to an individual user's account data in the form of an array of TWebUserItem (see page 25615).
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnAfterCheckAccessRights (see page 25630)	Triggered at the start of the CheckAccessRights protected function.
	OnAfterValidateUser (see page 25630)	Occurs after a user has been validated.
	OnBeforeCheckAccessRights (see page 25630)	Triggered at the end of the CheckAccessRights protected function.
	OnBeforeValidateUser (see page 25631)	Triggered at the beginning of the ValidateUser function.
	OnUserIDNotFound (see page 25631)	Triggered if the UserID is not found in the TCustomWebUserList object.
	OnValidateUserError (see page 25631)	Triggered if an error occurs during the validation process.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	UserItems (see page 25629)	Stores the account information about a collection of users.

188.1.7.2 TWebUserList Properties

The properties of the TWebUserList class are listed here.

Published Properties

	Name	Description
	UserItems (see page 25629)	Stores the account information about a collection of users.

188.1.7.2.1 TWebUserList.UserItems Property

Stores the account information about a collection of users.

Pascal

```
property UserItems: TWebUserItems;
```

C++

```
__property TWebUserItems UserItems;
```

Description

Use this object to store the account information about a collection of users. This property provides read and write access to the stored collection.

See Also

Users (see page 25612)

188.1.7.3 TWebUserList Events

The events of the TWebUserList class are listed here.

Published Events

	Name	Description
	OnAfterCheckAccessRights (see page 25630)	Triggered at the start of the CheckAccessRights protected function.
	OnAfterValidateUser (see page 25630)	Occurs after a user has been validated.
	OnBeforeCheckAccessRights (see page 25630)	Triggered at the end of the CheckAccessRights protected function.
	OnBeforeValidateUser (see page 25631)	Triggered at the beginning of the ValidateUser function.
	OnUserIDNotFound (see page 25631)	Triggered if the UserID is not found in the TCustomWebUserList object.
	OnValidateUserError (see page 25631)	Triggered if an error occurs during the validation process.

188.1.7.3.1 TWebUserList.OnAfterCheckAccessRights Event

Triggered at the start of the CheckAccessRights protected function.

Pascal

```
property OnAfterCheckAccessRights: TCheckAccessRightsEvent;
```

C++

```
__property TCheckAccessRightsEvent OnAfterCheckAccessRights;
```

Description

This event is triggered at the start of the CheckAccessRights protected function. This event can be used if any additional checking or initializing is desired before the access rights are checked.

See Also

OnBeforeCheckAccessRights (see page 25613)

188.1.7.3.2 TWebUserList.OnAfterValidateUser Event

Occurs after a user has been validated.

Pascal

```
property OnAfterValidateUser: TValidateUserEvent;
```

C++

```
__property TValidateUserEvent OnAfterValidateUser;
```

Description

This event is triggered at the end of the ValidateUser function. It can be used if any additional processing needs to be done after a user's name and password are verified.

See Also

OnBeforeValidateUser (see page 25614)

ValidateUser (see page 25611)

188.1.7.3.3 TWebUserList.OnBeforeCheckAccessRights Event

Triggered at the end of the CheckAccessRights protected function.

Pascal

```
property OnBeforeCheckAccessRights: TCheckAccessRightsHandledEvent;
```

C++

```
__property TCheckAccessRightsHandledEvent OnBeforeCheckAccessRights;
```

Description

This event is triggered at the end of the CheckAccessRights protected function. It can be used to perform your own custom checking instead of the default checking by the Web user list.

See Also

OnAfterCheckAccessRights (see page 25613)

188.1.7.3.4 TWebUserList.OnBeforeValidateUser Event

Triggered at the beginning of the ValidateUser function.

Pascal

```
property OnBeforeValidateUser: TValidateUserHandledEvent;
```

C++

```
__property TValidateUserHandledEvent OnBeforeValidateUser;
```

Description

This event is triggered at the beginning of the ValidateUser function. It can be used to identify a user given a name and password and supply a user ID for the user, once identified.

See Also

OnAfterValidateUser (see page 25613)

188.1.7.3.5 TWebUserList.OnUserIDNotFound Event

Triggered if the UserID is not found in the TCustomWebUserList object.

Pascal

```
property OnUserIDNotFound: TCheckAccessRightsHandledEvent;
```

C++

```
__property TCheckAccessRightsHandledEvent OnUserIDNotFound;
```

Description

This event is triggered if the UserID is not found in the TCustomWebUserList object. The CheckAccessRights protected function triggers this function if the UserID is not found while checking the access rights.

188.1.7.3.6 TWebUserList.OnValidateUserError Event

Triggered if an error occurs during the validation process.

Pascal

```
property OnValidateUserError: TValidateUserErrorEvent;
```

C++

```
__property TValidateUserErrorEvent OnValidateUserError;
```

Description

This event is triggered if an error occurs during the validation process. The ValidateUser function calls this event with a description of the error. Examples of error descriptions include a missing or unknown user name or a missing or invalid password. Any additional processing that needs to be completed in the case of one of these error conditions can be added to this event handler.

See Also

ValidateUser (see page 25611)

188.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TValidateUserError (see page 25633)	This is record WebUsers.TValidateUserError.

188.2.1 WebUsers.TValidateUserError Enumeration

Pascal

```
TValidateUserError = (
  vuBlankPassword,
  vuBlankUserName,
  vuUnknownUserName,
  vuUnknownPassword
);
```

C++

```
enum TValidateUserError {
  vuBlankPassword,
  vuBlankUserName,
  vuUnknownUserName,
  vuUnknownPassword
};
```

File

WebUsers

Description

This is record WebUsers.TValidateUserError.

188.3 Types

The following table lists types in this documentation.

Types

Name	Description
TCheckAccessRightsEvent (see page 25634)	This is type WebUsers.TCheckAccessRightsEvent.
TCheckAccessRightsHandledEvent (see page 25634)	This is type WebUsers.TCheckAccessRightsHandledEvent.
TValidateUserErrorEvent (see page 25635)	This is type WebUsers.TValidateUserErrorEvent.
TValidateUserEvent (see page 25635)	This is type WebUsers.TValidateUserEvent.
TValidateUserHandledEvent (see page 25635)	This is type WebUsers.TValidateUserHandledEvent.

188.3.1 WebUsers.TCheckAccessRightsEvent Type

Pascal

```
TCheckAccessRightsEvent = procedure (UserID: Variant; Rights: string; var HasRight: Boolean)
of object;
```

C++

```
(UserID: Variant; Rights: string; var HasRight: Boolean) ( TCheckAccessRightsEvent)();
```

File

WebUsers

Description

This is type WebUsers.TCheckAccessRightsEvent.

188.3.2 WebUsers.TCheckAccessRightsHandledEvent Type

Pascal

```
TCheckAccessRightsHandledEvent = procedure (UserID: Variant; Rights: string; var HasRight:
Boolean; var Handled: Boolean) of object;
```

C++

```
(UserID: Variant; Rights: string; var HasRight: Boolean; var Handled: Boolean) (
TCheckAccessRightsHandledEvent)();
```

File

WebUsers

Description

This is type WebUsers.TCheckAccessRightsHandledEvent.

188.3.3 WebUsers.TValidateUserErrorEvent Type

Pascal

```
TValidateUserErrorEvent = procedure (Error: TValidateUserError; Strings: TStrings; var UserID:
Variant; var Handled: Boolean) of object;
```

C++

```
(Error: TValidateUserError; Strings: TStrings; var UserID: Variant; var Handled: Boolean) (
TValidateUserErrorEvent)();
```

File

WebUsers

Description

This is type WebUsers.TValidateUserErrorEvent.

188.3.4 WebUsers.TValidateUserEvent Type

Pascal

```
TValidateUserEvent = procedure (Strings: TStrings; var UserID: Variant) of object;
```

C++

```
(Strings: TStrings; var UserID: Variant) ( TValidateUserEvent)();
```

File

WebUsers

Description

This is type WebUsers.TValidateUserEvent.

188.3.5 WebUsers.TValidateUserHandledEvent Type

Pascal

```
TValidateUserHandledEvent = procedure (Strings: TStrings; var UserID: Variant; var Handled:
Boolean) of object;
```

C++

```
(Strings: TStrings; var UserID: Variant; var Handled: Boolean) ( TValidateUserHandledEvent)();
```

File

WebUsers

Description

This is type WebUsers.TValidateUserHandledEvent.

189 WSDLBind Namespace

This is namespace WSDLBind.

Topics

Name	Description
Classes (see page 25640)	The following table lists classes in this documentation.
Interfaces (see page 25815)	The following table lists interfaces in this documentation.
Functions (see page 25908)	The following table lists functions in this documentation.
Constants (see page 25910)	The following table lists constants in this documentation.

Classes

	Name	Description
	TBinding (see page 25640)	This is class WSDLBind.TBinding.
	TBindingOperation (see page 25647)	This is class WSDLBind.TBindingOperation.
	TBindingOperations (see page 25654)	This is class WSDLBind.TBindingOperations.
	TBindings (see page 25661)	This is class WSDLBind.TBindings.
	TDefinition (see page 25667)	This is class WSDLBind.TDefinition.
	TDocumentation (see page 25676)	This is class WSDLBind.TDocumentation.
	TDocumentationCollection (see page 25676)	This is class WSDLBind.TDocumentationCollection.
	TDocumented (see page 25683)	This is class WSDLBind.TDocumented.
	TFault (see page 25689)	This is class WSDLBind.TFault.
	TFaults (see page 25689)	This is class WSDLBind.TFaults.
	TImport (see page 25696)	This is class WSDLBind.TImport.
	TImports (see page 25702)	This is class WSDLBind.TImports.
	TMessage (see page 25708)	This is class WSDLBind.TMessage.
	TMessages (see page 25715)	This is class WSDLBind.TMessages.
	TOperation (see page 25721)	This is class WSDLBind.TOperation.
	TOperations (see page 25728)	This is class WSDLBind.TOperations.
	TParam (see page 25735)	This is class WSDLBind.TParam.
	TPart (see page 25741)	This is class WSDLBind.TPart.
	TParts (see page 25748)	This is class WSDLBind.TParts.
	TPort (see page 25755)	This is class WSDLBind.TPort.
	TPorts (see page 25761)	This is class WSDLBind.TPorts.
	TPortType (see page 25767)	This is class WSDLBind.TPortType.
	TPortTypes (see page 25774)	This is class WSDLBind.TPortTypes.
	TService (see page 25780)	This is class WSDLBind.TService.

	TServices (see page 25787)	This is class WSDLBind.TServices.
	TTypes (see page 25793)	This is class WSDLBind.TTypes.
	TWSDLDocument (see page 25799)	TWSDLDocument represents a WSDL document.
	TXMLSchemaDefs (see page 25808)	This is class WSDLBind.TXMLSchemaDefs.

Constants

Name	Description
SBinding (see page 25910)	This is constant WSDLBind.SBinding.
SDefinitions (see page 25910)	This is constant WSDLBind.SDefinitions.
SFault (see page 25911)	This is constant WSDLBind.SFault.
SImport (see page 25911)	This is constant WSDLBind.SImport.
SInput (see page 25911)	This is constant WSDLBind.SInput.
SLocation (see page 25912)	This is constant WSDLBind.SLocation.
SMessage (see page 25912)	This is constant WSDLBind.SMessage.
SNamespace (see page 25912)	This is constant WSDLBind.SNamespace.
SOperation (see page 25913)	This is constant WSDLBind.SOperation.
SOutput (see page 25913)	This is constant WSDLBind.SOutput.
SPart (see page 25913)	This is constant WSDLBind.SPart.
SPort (see page 25914)	This is constant WSDLBind.SPort.
SPortType (see page 25914)	This is constant WSDLBind.SPortType.
SService (see page 25914)	This is constant WSDLBind.SService.
SType (see page 25915)	This is constant WSDLBind.SType.
STypes (see page 25915)	This is constant WSDLBind.STypes.

Functions

	Name	Description
	GetDefinition (see page 25908)	This is function WSDLBind.GetDefinition.
	LoadWSDLDoc (see page 25908)	This is function WSDLBind.LoadWSDLDoc.
	NewWSDLDoc (see page 25908)	This is function WSDLBind.NewWSDLDoc.

Interfaces

	Name	Description
	IBinding (see page 25815)	This is class WSDLBind.IBinding.
	IBindingInput (see page 25819)	This is class WSDLBind.IBindingInput.
	IBindingOperation (see page 25820)	This is class WSDLBind.IBindingOperation.
	IBindingOperations (see page 25824)	This is class WSDLBind.IBindingOperations.
	IBindingOutput (see page 25827)	This is class WSDLBind.IBindingOutput.
	IBindings (see page 25828)	This is class WSDLBind.IBindings.
	IDefinition (see page 25831)	This is class WSDLBind.IDefinition.
	IDocumentation (see page 25837)	This is class WSDLBind.IDocumentation.
	IDocumentationCollection (see page 25838)	This is class WSDLBind.IDocumentationCollection.
	IDocumented (see page 25841)	This is class WSDLBind.IDocumented.
	IFault (see page 25843)	This is class WSDLBind.IFault.
	IFaults (see page 25844)	This is class WSDLBind.IFaults.
	IImport (see page 25847)	This is class WSDLBind.IImport.
	IImports (see page 25850)	This is class WSDLBind.IImports.
	IMessage (see page 25853)	This is class WSDLBind.IMessage.
	IMessages (see page 25857)	This is class WSDLBind.IMessages.
	IOperation (see page 25860)	This is class WSDLBind.IOperation.
	IOperations (see page 25864)	This is class WSDLBind.IOperations.
	IParam (see page 25868)	This is class WSDLBind.IParam.
	IPart (see page 25871)	This is class WSDLBind.IPart.
	IParts (see page 25876)	This is class WSDLBind.IParts.
	IPort (see page 25879)	This is class WSDLBind.IPort.
	IPorts (see page 25882)	This is class WSDLBind.IPorts.

	IPortType (see page 25885)	This is class WSDLBind.IPortType.
	IPortTypes (see page 25889)	This is class WSDLBind.IPortTypes.
	IService (see page 25892)	This is class WSDLBind.IService.
	IServices (see page 25895)	This is class WSDLBind.IServices.
	ITypes (see page 25898)	This is class WSDLBind.ITypes.
	IWSDLDocument (see page 25901)	IWSDLDocument provides low-level access to the definitions in a WSDL document.
	IXMLSchemaDefs (see page 25905)	This is class WSDLBind.IXMLSchemaDefs.

189.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TBinding (see page 25640)	This is class WSDLBind.TBinding.
	TBindingOperation (see page 25647)	This is class WSDLBind.TBindingOperation.
	TBindingOperations (see page 25654)	This is class WSDLBind.TBindingOperations.
	TBindings (see page 25661)	This is class WSDLBind.TBindings.
	TDefinition (see page 25667)	This is class WSDLBind.TDefinition.
	TDocumentation (see page 25676)	This is class WSDLBind.TDocumentation.
	TDocumentationCollection (see page 25676)	This is class WSDLBind.TDocumentationCollection.
	TDocumented (see page 25683)	This is class WSDLBind.TDocumented.
	TFault (see page 25689)	This is class WSDLBind.TFault.
	TFaults (see page 25689)	This is class WSDLBind.TFaults.
	TImport (see page 25696)	This is class WSDLBind.TImport.
	TImports (see page 25702)	This is class WSDLBind.TImports.
	TMessage (see page 25708)	This is class WSDLBind.TMessage.
	TMessages (see page 25715)	This is class WSDLBind.TMessages.
	TOperation (see page 25721)	This is class WSDLBind.TOperation.
	TOperations (see page 25728)	This is class WSDLBind.TOperations.
	TParam (see page 25735)	This is class WSDLBind.TParam.
	TPart (see page 25741)	This is class WSDLBind.TPart.
	TParts (see page 25748)	This is class WSDLBind.TParts.
	TPort (see page 25755)	This is class WSDLBind.TPort.
	TPorts (see page 25761)	This is class WSDLBind.TPorts.
	TPortType (see page 25767)	This is class WSDLBind.TPortType.
	TPortTypes (see page 25774)	This is class WSDLBind.TPortTypes.
	TService (see page 25780)	This is class WSDLBind.TService.
	TServices (see page 25787)	This is class WSDLBind.TServices.
	TTypes (see page 25793)	This is class WSDLBind.TTypes.
	TWSDLDocument (see page 25799)	TWSDLDocument represents a WSDL document.
	TXMLSchemaDefs (see page 25808)	This is class WSDLBind.TXMLSchemaDefs.

189.1.1 TBinding Class

Class Hierarchy

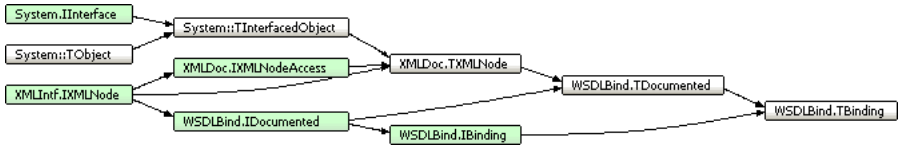

Pascal

```
TBinding = class(TDocumented, IBinding);
```

C++

```
class TBinding : public TDocumented, public IBinding;
```

File

WSDLBind

Description

This is class WSDLBind.TBinding.

189.1.1.1 TBinding Members

The following tables list the members exposed by TBinding.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.

	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetBindingOperations (see page 25646)	This is GetBindingOperations, a member of class TBinding.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetDocumentationCollection (see page 25688)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetName (see page 25646)	This is GetName, a member of class TBinding.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetType_ (see page 25647)	This is GetType_, a member of class TBinding.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.

	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetName (see page 25647)	This is SetName, a member of class TBinding.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	SetType_ (see page 25647)	This is SetType_, a member of class TBinding.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 25646)	Responds after the last constructor executes.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	BindingOperations (see page 25819)	This is BindingOperations, a member of class IBinding.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Name (see page 25819)	This is Name, a member of class IBinding.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
	Type_ (see page 25819)	This is Type_, a member of class IBinding.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.1.2 TBinding Methods

The methods of the TBinding class are listed here.

Protected Methods

	Name	Description
	GetBindingOperations (see page 25646)	This is GetBindingOperations, a member of class TBinding.
	GetName (see page 25646)	This is GetName, a member of class TBinding.
	GetType_ (see page 25647)	This is GetType_, a member of class TBinding.

	SetName (see page 25647)	This is SetName, a member of class TBinding.
	SetType_ (see page 25647)	This is SetType_, a member of class TBinding.

Public Methods

	Name	Description
	AfterConstruction (see page 25646)	Responds after the last constructor executes.

189.1.1.2.1 TBinding.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 22279)

System::TInterfacedObject::NewInstance (see page 22280)

System::TInterfacedObject::RefCount (see page 22281)

189.1.1.2.2 TBinding.GetBindingOperations Method

Pascal

```
function GetBindingOperations: IBindingOperations;
```

C++

```
__fastcall IBindingOperations GetBindingOperations();
```

Description

This is GetBindingOperations, a member of class TBinding.

189.1.1.2.3 TBinding.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TBinding.

189.1.1.2.4 TBinding.GetType_ Method

Pascal

```
function GetType_: WideString;
```

C++

```
__fastcall BSTR GetType_();
```

Description

This is GetType_, a member of class TBinding.

189.1.1.2.5 TBinding.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TBinding.

189.1.1.2.6 TBinding.SetType_ Method

Pascal

```
procedure SetType_(Value: WideString);
```

C++

```
__fastcall SetType_(BSTR Value);
```

Description

This is SetType_, a member of class TBinding.

189.1.2 TBindingOperation Class

Class Hierarchy

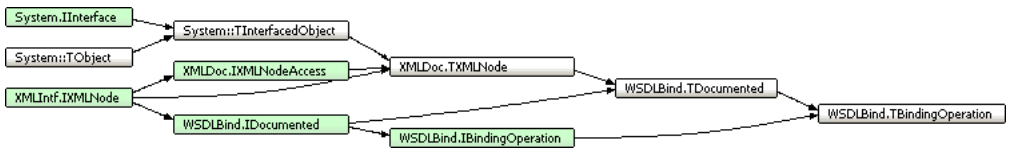

Pascal

```
TBindingOperation = class(TDocumented, IBindingOperation);
```

C++

```
class TBindingOperation : public TDocumented, public IBindingOperation;
```

File

WSDLBind

Description

This is class WSDLBind.TBindingOperation.

189.1.2.1 TBindingOperation Members

The following tables list the members exposed by TBindingOperation.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.

	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetDocumentationCollection (see page 25688)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetFaults (see page 25653)	This is GetFaults, a member of class TBindingOperation.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetInput (see page 25653)	This is GetInput, a member of class TBindingOperation.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetName (see page 25654)	This is GetName, a member of class TBindingOperation.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOutput (see page 25654)	This is GetOutput, a member of class TBindingOperation.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.

	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetName (see page 25654)	This is SetName, a member of class TBindingOperation.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 25653)	Responds after the last constructor executes.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.

	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.

	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	Faults (see page 25823)	This is Faults, a member of class IBindingOperation.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	Input (see page 25823)	This is Input, a member of class IBindingOperation.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Name (see page 25824)	This is Name, a member of class IBindingOperation.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	Output (see page 25824)	This is Output, a member of class IBindingOperation.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.2.2 TBindingOperation Methods

The methods of the TBindingOperation class are listed here.

Protected Methods

	Name	Description
	GetFaults (see page 25653)	This is GetFaults, a member of class TBindingOperation.
	GetInput (see page 25653)	This is GetInput, a member of class TBindingOperation.
	GetName (see page 25654)	This is GetName, a member of class TBindingOperation.
	GetOutput (see page 25654)	This is GetOutput, a member of class TBindingOperation.

	SetName (see page 25654)	This is SetName, a member of class TBindingOperation.

Public Methods

	Name	Description
	AfterConstruction (see page 25653)	Responds after the last constructor executes.

189.1.2.2.1 TBindingOperation.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 22279)

System::TInterfacedObject::NewInstance (see page 22280)

System::TInterfacedObject::RefCount (see page 22281)

189.1.2.2.2 TBindingOperation.GetFaults Method

Pascal

```
function GetFaults: IFaults;
```

C++

```
__fastcall IFaults GetFaults();
```

Description

This is GetFaults, a member of class TBindingOperation.

189.1.2.2.3 TBindingOperation.GetInput Method

Pascal

```
function GetInput: IBindingInput;
```

C++

```
__fastcall IBindingInput GetInput();
```

Description

This is GetInput, a member of class TBindingOperation.

189.1.2.2.4 TBindingOperation.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TBindingOperation.

189.1.2.2.5 TBindingOperation.GetOutput Method

Pascal

```
function GetOutput: IBindingOutput;
```

C++

```
__fastcall IBindingOutput GetOutput();
```

Description

This is GetOutput, a member of class TBindingOperation.

189.1.2.2.6 TBindingOperation.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TBindingOperation.

189.1.3 TBindingOperations Class

Class Hierarchy

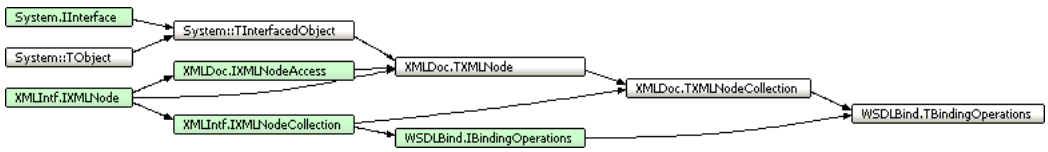

Pascal

```
TBindingOperations = class(TXMLNodeCollection, IBindingOperations);
```

C++

```
class TBindingOperations : public TXMLNodeCollection, public IBindingOperations;
```

File

WSDLBind

Description

This is class WSDLBind.TBindingOperations.

189.1.3.1 TBindingOperations Members

The following tables list the members exposed by TBindingOperations.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	Add (see page 25660)	This is Add, a member of class TBindingOperations.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
V	AddItem (see page 26202)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 26203)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 26204)	Removes all child nodes in the List (see page 26210) property from the ChildNodes (see page 26190) and List (see page 26210) properties.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	CreateItemList (see page 26204)	Creates the object that implements the List (see page 26210) property.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.

	Delete (see page 26205)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetBindingOperation (see page 25660)	This is GetBindingOperation, a member of class TBindingOperations.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetCount (see page 26205)	Returns the value of the Count (see page 26209) property.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
V	GetList (see page 26205)	Returns the value of the List (see page 26210) property.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 26206)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	Insert (see page 25660)	This is Insert, a member of class TBindingOperations.
	InsertInCollection (see page 26206)	Copies a child node from the ChildNodes (see page 26190) property to the List (see page 26210) property.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	IsCollectionItem (see page 26207)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.

	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	Remove (see page 26207)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
V	SetChildNodes (see page 26208)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 26208)	Updates the List (see page 26210) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	Count (see page 26209)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 26209)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 26210)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 26210)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 26210)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 26202)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.

	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	BindingOperation (see page 25827)	This is BindingOperation, a member of class IBindingOperations.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.

R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.3.2 TBindingOperations Methods

The methods of the TBindingOperations class are listed here.

Protected Methods

	Name	Description
	Add (see page 25660)	This is Add, a member of class TBindingOperations.
	GetBindingOperation (see page 25660)	This is GetBindingOperation, a member of class TBindingOperations.
	Insert (see page 25660)	This is Insert, a member of class TBindingOperations.

189.1.3.2.1 TBindingOperations.Add Method

Pascal

```
function Add(const Name: WideString): IBindingOperation;
```

C++

```
__fastcall IBindingOperation Add(const BSTR Name);
```

Description

This is Add, a member of class TBindingOperations.

189.1.3.2.2 TBindingOperations.GetBindingOperation Method

Pascal

```
function GetBindingOperation(Index: Integer): IBindingOperation;
```

C++

```
__fastcall IBindingOperation GetBindingOperation(int Index);
```

Description

This is GetBindingOperation, a member of class TBindingOperations.

189.1.3.2.3 TBindingOperations.Insert Method

Pascal

```
function Insert(Index: Integer): IBindingOperation;
```

C++

```
__fastcall IBindingOperation Insert(int Index);
```

Description

This is Insert, a member of class TBindingOperations.

189.1.4 TBindings Class

Class Hierarchy

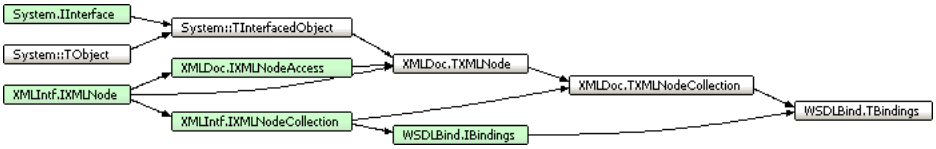

Pascal

```
TBindings = class(TXMLNodeCollection, IBindings);
```

C++

```
class TBindings : public TXMLNodeCollection, public IBindings;
```

File

WSDLBind

Description

This is class WSDLBind.TBindings.

189.1.4.1 TBindings Members

The following tables list the members exposed by TBindings.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	Add (see page 25666)	This is Add, a member of class TBindings.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AddItem (see page 26202)	Adds a new repetition of the repeating child element to this node.

	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 26203)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 26204)	Removes all child nodes in the List (see page 26210) property from the ChildNodes (see page 26190) and List (see page 26210) properties.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	CreateItemList (see page 26204)	Creates the object that implements the List (see page 26210) property.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26205)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetBinding (see page 25667)	This is GetBinding, a member of class TBindings.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetCount (see page 26205)	Returns the value of the Count (see page 26209) property.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
V	GetList (see page 26205)	Returns the value of the List (see page 26210) property.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 26206)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.

	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	Insert (see page 25667)	This is Insert, a member of class TBindings.
	InsertInCollection (see page 26206)	Copies a child node from the ChildNodes (see page 26190) property to the List (see page 26210) property.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	IsCollectionItem (see page 26207)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	Remove (see page 26207)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
V	SetChildNodes (see page 26208)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 26208)	Updates the List (see page 26210) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	Count (see page 26209)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 26209)	Indicates the interface type for the repeating child nodes.

	ItemNS (see page 26210)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 26210)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 26210)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 26202)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.

	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.

	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	Binding (see page 25831)	This is Binding, a member of class IBindings.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.4.2 TBindings Methods

The methods of the TBindings class are listed here.

Protected Methods

	Name	Description
	Add (see page 25666)	This is Add, a member of class TBindings.
	GetBinding (see page 25667)	This is GetBinding, a member of class TBindings.
	Insert (see page 25667)	This is Insert, a member of class TBindings.

189.1.4.2.1 TBindings.Add Method

Pascal

```
function Add(const Name: WideString; const AType: WideString): IBinding;
```

C++

```
__fastcall IBinding Add(const BSTR Name, const BSTR AType);
```

Description

This is Add, a member of class TBindings.

189.1.4.2.2 TBindings.GetBinding Method

Pascal

```
function GetBinding(Index: Integer): IBinding;
```

C++

```
__fastcall IBinding GetBinding(int Index);
```

Description

This is GetBinding, a member of class TBindings.

189.1.4.2.3 TBindings.Insert Method

Pascal

```
function Insert(Index: Integer): IBinding;
```

C++

```
__fastcall IBinding Insert(int Index);
```

Description

This is Insert, a member of class TBindings.

189.1.5 TDefinition Class

Class Hierarchy

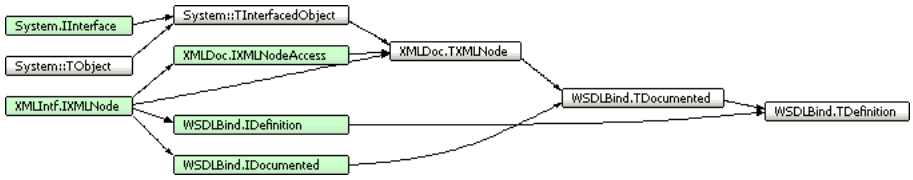

Pascal

```
TDefinition = class(TDocumented, IDefinition);
```

C++

```
class TDefinition : public TDocumented, public IDefinition;
```

File

WSDLBind

Description

This is class WSDLBind.TDefinition.

189.1.5.1 TDefinition Members

The following tables list the members exposed by TDefinition.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetBindings (see page 25673)	This is GetBindings, a member of class TDefinition.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetDocumentationCollection (see page 25688)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.

	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetImports (see page 25673)	This is GetImports, a member of class TDefinition.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetMessages (see page 25674)	This is GetMessages, a member of class TDefinition.
	GetName (see page 25674)	This is GetName, a member of class TDefinition.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPortTypes (see page 25674)	This is GetPortTypes, a member of class TDefinition.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetServices (see page 25674)	This is GetServices, a member of class TDefinition.
	GetTargetNamespace (see page 25675)	This is GetTargetNamespace, a member of class TDefinition.
	GetText (see page 26177)	Returns the text value of the node.
	GetTypes (see page 25675)	This is GetTypes, a member of class TDefinition.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetName (see page 25675)	This is SetName, a member of class TDefinition.
	SetNodeValue (see page 26186)	Sets the value of this node.
	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetTargetNamespace (see page 25675)	This is SetTargetNamespace, a member of class TDefinition.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 25673)	Responds after the last constructor executes.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.

	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	Bindings (see page 25836)	This is Bindings, a member of class IDefinition.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	Imports (see page 25836)	This is Imports, a member of class IDefinition.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	Messages (see page 25836)	This is Messages, a member of class IDefinition.
	Name (see page 25836)	This is Name, a member of class IDefinition.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	PortTypes (see page 25836)	This is PortTypes, a member of class IDefinition.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	Services (see page 25837)	This is Services, a member of class IDefinition.
	TargetNamespace (see page 25837)	This is TargetNamespace, a member of class IDefinition.
	Text (see page 26439)	Represents the Text of the node.
R	Types (see page 25837)	This is Types, a member of class IDefinition.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.5.2 TDefinition Methods

The methods of the TDefinition class are listed here.

Protected Methods

	Name	Description
	GetBindings (see page 25673)	This is GetBindings, a member of class TDefinition.
	GetImports (see page 25673)	This is GetImports, a member of class TDefinition.
	GetMessages (see page 25674)	This is GetMessages, a member of class TDefinition.
	GetName (see page 25674)	This is GetName, a member of class TDefinition.
	GetPortTypes (see page 25674)	This is GetPortTypes, a member of class TDefinition.
	GetServices (see page 25674)	This is GetServices, a member of class TDefinition.
	GetTargetNamespace (see page 25675)	This is GetTargetNamespace, a member of class TDefinition.

	GetTypes (see page 25675)	This is GetTypes, a member of class TDefinition.
	SetName (see page 25675)	This is SetName, a member of class TDefinition.
	SetTargetNamespace (see page 25675)	This is SetTargetNamespace, a member of class TDefinition.

Public Methods

	Name	Description
	AfterConstruction (see page 25673)	Responds after the last constructor executes.

189.1.5.2.1 TDefinition.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 22279)

System::TInterfacedObject::NewInstance (see page 22280)

System::TInterfacedObject::RefCount (see page 22281)

189.1.5.2.2 TDefinition.GetBindings Method

Pascal

```
function GetBindings: IBindings;
```

C++

```
__fastcall IBindings GetBindings();
```

Description

This is GetBindings, a member of class TDefinition.

189.1.5.2.3 TDefinition.GetImports Method

Pascal

```
function GetImports: IImports;
```

C++

```
__fastcall IImports GetImports();
```

Description

This is GetImports, a member of class TDefinition.

189.1.5.2.4 TDefinition.GetMessages Method

Pascal

```
function GetMessages: IMessages;
```

C++

```
__fastcall IMessages GetMessages();
```

Description

This is GetMessages, a member of class TDefinition.

189.1.5.2.5 TDefinition.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TDefinition.

189.1.5.2.6 TDefinition.GetPortTypes Method

Pascal

```
function GetPortTypes: IPortTypes;
```

C++

```
__fastcall IPortTypes GetPortTypes();
```

Description

This is GetPortTypes, a member of class TDefinition.

189.1.5.2.7 TDefinition.GetServices Method

Pascal

```
function GetServices: IServices;
```

C++

```
__fastcall IServices GetServices();
```

Description

This is GetServices, a member of class TDefinition.

189.1.5.2.8 TDefinition.GetTargetNamespace Method

Pascal

```
function GetTargetNamespace: WideString;
```

C++

```
__fastcall BSTR GetTargetNamespace();
```

Description

This is GetTargetNamespace, a member of class TDefinition.

189.1.5.2.9 TDefinition.GetTypes Method

Pascal

```
function GetTypes: ITypes;
```

C++

```
__fastcall ITypes GetTypes();
```

Description

This is GetTypes, a member of class TDefinition.

189.1.5.2.10 TDefinition.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TDefinition.

189.1.5.2.11 TDefinition.SetTargetNamespace Method

Pascal

```
procedure SetTargetNamespace(Value: WideString);
```

C++

```
__fastcall SetTargetNamespace(BSTR Value);
```

Description

This is SetTargetNamespace, a member of class TDefinition.

189.1.6 TDocumentation Class

Class Hierarchy

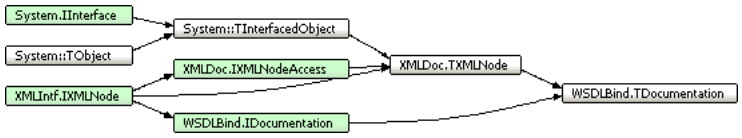

Pascal

```
TDocumentation = class(TXMLNode, IDocumentation);
```

C++

```
class TDocumentation : public TXMLNode, public IDocumentation;
```

File

WSDLBind

Description

This is class WSDLBind.TDocumentation.

189.1.7 TDocumentationCollection Class

Class Hierarchy

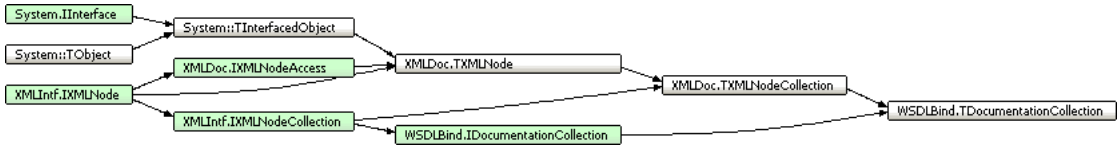

Pascal

```
TDocumentationCollection = class(TXMLNodeCollection, IDocumentationCollection);
```

C++

```
class TDocumentationCollection : public TXMLNodeCollection, public IDocumentationCollection;
```

File

WSDLBind

Description

This is class WSDLBind.TDocumentationCollection.

189.1.7.1 TDocumentationCollection Members

The following tables list the members exposed by TDocumentationCollection.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	Add (see page 25682)	This is Add, a member of class TDocumentationCollection.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
V	AddItem (see page 26202)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 26203)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 26204)	Removes all child nodes in the List (see page 26210) property from the ChildNodes (see page 26190) and List (see page 26210) properties.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	CreateItemList (see page 26204)	Creates the object that implements the List (see page 26210) property.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26205)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.

	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetCount (see page 26205)	Returns the value of the Count (see page 26209) property.
	GetDocumentation (see page 25682)	This is GetDocumentation, a member of class TDocumentationCollection.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
V	GetList (see page 26205)	Returns the value of the List (see page 26210) property.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 26206)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	Insert (see page 25682)	This is Insert, a member of class TDocumentationCollection.
	InsertInCollection (see page 26206)	Copies a child node from the ChildNodes (see page 26190) property to the List (see page 26210) property.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	IsCollectionItem (see page 26207)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	Remove (see page 26207)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
V	SetChildNodes (see page 26208)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.

	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 26208)	Updates the List (see page 26210) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	Count (see page 26209)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 26209)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 26210)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 26210)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 26210)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 26202)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.

	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	Documentation (see page 25841)	This is Documentation, a member of class IDocumentationCollection.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.7.2 TDocumentationCollection Methods

The methods of the TDocumentationCollection class are listed here.

Protected Methods

	Name	Description
	Add (see page 25682)	This is Add, a member of class TDocumentationCollection.
	GetDocumentation (see page 25682)	This is GetDocumentation, a member of class TDocumentationCollection.
	Insert (see page 25682)	This is Insert, a member of class TDocumentationCollection.

189.1.7.2.1 TDocumentationCollection.Add Method

Pascal

```
function Add(const Text: WideString): IDocumentation;
```

C++

```
__fastcall IDocumentation Add(const BSTR Text);
```

Description

This is Add, a member of class TDocumentationCollection.

189.1.7.2.2 TDocumentationCollection.GetDocumentation Method

Pascal

```
function GetDocumentation(Index: Integer): IDocumentation;
```

C++

```
__fastcall IDocumentation GetDocumentation(int Index);
```

Description

This is GetDocumentation, a member of class TDocumentationCollection.

189.1.7.2.3 TDocumentationCollection.Insert Method

Pascal

```
function Insert(Index: Integer): IDocumentation;
```

C++

```
__fastcall IDocumentation Insert(int Index);
```

Description

This is Insert, a member of class TDocumentationCollection.

189.1.8 TDocumented Class

Class Hierarchy

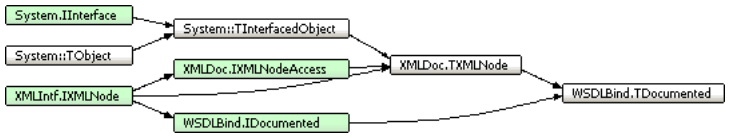

Pascal

```
TDocumented = class(TXMLNode, IDocumented);
```

C++

```
class TDocumented : public TXMLNode, public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.TDocumented.

189.1.8.1 TDocumented Members

The following tables list the members exposed by TDocumented.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.

	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetDocumentationCollection (see page 25688)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.

	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 25688)	Responds after the last constructor executes.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.

	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.

	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.8.2 TDocumented Methods

The methods of the TDocumented class are listed here.

Protected Methods

	Name	Description
	GetDocumentationCollection (see page 25688)	This is GetDocumentationCollection, a member of class TDocumented.

Public Methods

	Name	Description
	AfterConstruction (see page 25688)	Responds after the last constructor executes.

189.1.8.2.1 TDocumented.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 22279)

System::TInterfacedObject::NewInstance (see page 22280)

System::TInterfacedObject::RefCount (see page 22281)

189.1.8.2.2 TDocumented.GetDocumentationCollection Method

Pascal

```
function GetDocumentationCollection: IDocumentationCollection;
```

C++

```
__fastcall IDocumentationCollection GetDocumentationCollection();
```

Description

This is GetDocumentationCollection, a member of class TDocumented.

189.1.9 TFault Class

Class Hierarchy

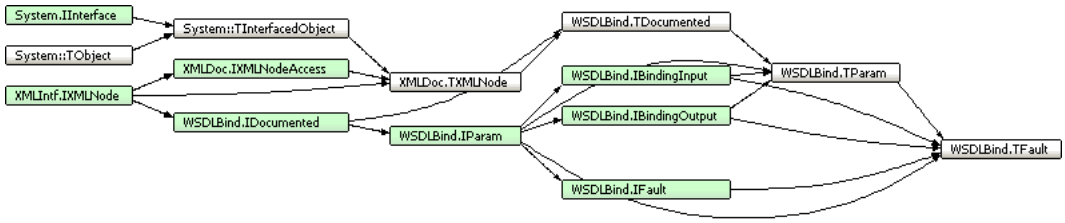

Pascal

```
TFault = class(TParam, IParam, IFault, IBindingInput, IBindingOutput);
```

C++

```
class TFault : public TParam, public IParam, public IFault, public IBindingInput, public
IBindingOutput;
```

File

WSDLBind

Description

This is class WSDLBind.TFault.

189.1.10 TFaults Class

Class Hierarchy

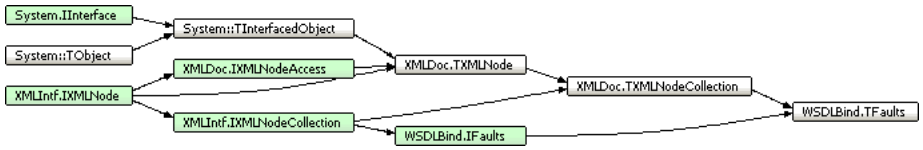

Pascal

```
TFaults = class(TXMLNodeCollection, IFaults);
```

C++

```
class TFaults : public TXMLNodeCollection, public IFaults;
```

File

WSDLBind

Description

This is class WSDLBind.TFaults.

189.1.10.1 TFaults Members

The following tables list the members exposed by TFaults.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	Add (see page 25695)	This is Add, a member of class TFaults.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
V	AddItem (see page 26202)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 26203)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 26204)	Removes all child nodes in the List (see page 26210) property from the ChildNodes (see page 26190) and List (see page 26210) properties.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	CreateItemList (see page 26204)	Creates the object that implements the List (see page 26210) property.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26205)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.

	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetCount (see page 26205)	Returns the value of the Count (see page 26209) property.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetFault (see page 25695)	This is GetFault, a member of class TFaults.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
V	GetList (see page 26205)	Returns the value of the List (see page 26210) property.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 26206)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	Insert (see page 25695)	This is Insert, a member of class TFaults.
	InsertInCollection (see page 26206)	Copies a child node from the ChildNodes (see page 26190) property to the List (see page 26210) property.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	IsCollectionItem (see page 26207)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	Remove (see page 26207)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.

	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
V	SetChildNodes (see page 26208)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 26208)	Updates the List (see page 26210) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	Count (see page 26209)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 26209)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 26210)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 26210)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 26210)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 26202)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.

	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	Fault (see page 25847)	This is Fault, a member of class IFaults.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.

R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.10.2 TFaults Methods

The methods of the TFaults class are listed here.

Protected Methods

	Name	Description
	Add (see page 25695)	This is Add, a member of class TFaults.
	GetFault (see page 25695)	This is GetFault, a member of class TFaults.
	Insert (see page 25695)	This is Insert, a member of class TFaults.

189.1.10.2.1 TFaults.Add Method

Pascal

```
function Add(const Name: WideString; const Message: WideString): IFault;
```

C++

```
__fastcall IFault Add(const BSTR Name, const BSTR Message);
```

Description

This is Add, a member of class TFaults.

189.1.10.2.2 TFaults.GetFault Method

Pascal

```
function GetFault(Index: Integer): IFault;
```

C++

```
__fastcall IFault GetFault(int Index);
```

Description

This is GetFault, a member of class TFaults.

189.1.10.2.3 TFaults.Insert Method

Pascal

```
function Insert(Index: Integer): IFault;
```

C++

```
__fastcall IFault Insert(int Index);
```

Description

This is Insert, a member of class TFaults.

189.1.11 TImport Class

Class Hierarchy

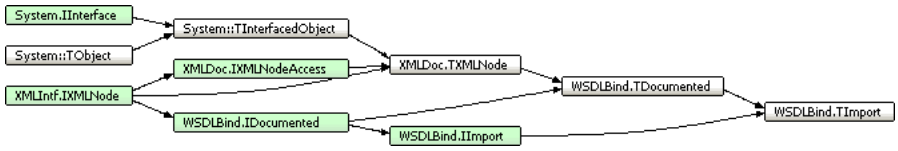

Pascal

```
TImport = class(TDocumented, IImport);
```

C++

```
class TImport : public TDocumented, public IImport;
```

File

WSDLBind

Description

This is class WSDLBind.TImport.

189.1.11.1 TImport Members

The following tables list the members exposed by TImport.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.

	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetDocumentationCollection (see page 25688)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetLocation (see page 25701)	This is GetLocation, a member of class TImport.
	GetNamespace (see page 25701)	This is GetNamespace, a member of class TImport.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.

	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetLocation (see page 25701)	This is SetLocation, a member of class TImport.
	SetNamespace (see page 25702)	This is SetNamespace, a member of class TImport.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 25688)	Responds after the last constructor executes.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Location (see page 25850)	This is Location, a member of class IImport.
	Namespace (see page 25850)	This is Namespace, a member of class IImport.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.

R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.11.2 TImport Methods

The methods of the TImport class are listed here.

Protected Methods

	Name	Description
	GetLocation (see page 25701)	This is GetLocation, a member of class TImport.
	GetNamespace (see page 25701)	This is GetNamespace, a member of class TImport.
	SetLocation (see page 25701)	This is SetLocation, a member of class TImport.
	SetNamespace (see page 25702)	This is SetNamespace, a member of class TImport.

189.1.11.2.1 TImport.GetLocation Method

Pascal

```
function GetLocation: WideString;
```

C++

```
__fastcall BSTR GetLocation();
```

Description

This is GetLocation, a member of class TImport.

189.1.11.2.2 TImport.GetNamespace Method

Pascal

```
function GetNamespace: WideString;
```

C++

```
__fastcall BSTR GetNamespace();
```

Description

This is GetNamespace, a member of class TImport.

189.1.11.2.3 TImport.SetLocation Method

Pascal

```
procedure SetLocation(Value: WideString);
```

C++

```
__fastcall SetLocation(BSTR Value);
```

Description

This is SetLocation, a member of class TImport.

189.1.11.2.4 TImport.SetNamespace Method

Pascal

```
procedure SetNamespace(Value: WideString);
```

C++

```
__fastcall SetNamespace(BSTR Value);
```

Description

This is SetNamespace, a member of class TImport.

189.1.12 TImports Class

Class Hierarchy

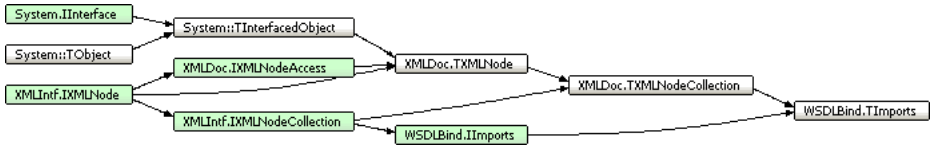

Pascal

```
TImports = class(TXMLNodeCollection, IImports);
```

C++

```
class TImports : public TXMLNodeCollection, public IImports;
```

File

WSDLBind

Description

This is class WSDLBind.TImports.

189.1.12.1 TImports Members

The following tables list the members exposed by TImports.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	Add (see page 25708)	This is Add, a member of class TImports.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
V	AddItem (see page 26202)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 26203)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 26204)	Removes all child nodes in the List (see page 26210) property from the ChildNodes (see page 26190) and List (see page 26210) properties.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	CreateItemList (see page 26204)	Creates the object that implements the List (see page 26210) property.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26205)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetCount (see page 26205)	Returns the value of the Count (see page 26209) property.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetImport (see page 25708)	This is GetImport, a member of class TImports.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.

	GetList (see page 26205)	Returns the value of the List (see page 26210) property.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 26206)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	Insert (see page 25708)	This is Insert, a member of class TImports.
	InsertInCollection (see page 26206)	Copies a child node from the ChildNodes (see page 26190) property to the List (see page 26210) property.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	IsCollectionItem (see page 26207)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	Remove (see page 26207)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildNodes (see page 26208)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetNodeValue (see page 26186)	Sets the value of this node.
	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 26208)	Updates the List (see page 26210) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	Count (see page 26209)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 26209)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 26210)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 26210)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 26210)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 26202)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	Import (see page 25853)	This is Import, a member of class IImports.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.12.2 TImports Methods

The methods of the TImports class are listed here.

Protected Methods

	Name	Description
	Add (see page 25708)	This is Add, a member of class TImports.
	GetImport (see page 25708)	This is GetImport, a member of class TImports.

	Insert (see page 25708)	This is Insert, a member of class TImports.

189.1.12.2.1 TImports.Add Method

Pascal

```
function Add(const Namespace: WideString; const Location: WideString): IImport;
```

C++

```
__fastcall IImport Add(const BSTR Namespace, const BSTR Location);
```

Description

This is Add, a member of class TImports.

189.1.12.2.2 TImports.GetImport Method

Pascal

```
function GetImport(Index: Integer): IImport;
```

C++

```
__fastcall IImport GetImport(int Index);
```

Description

This is GetImport, a member of class TImports.

189.1.12.2.3 TImports.Insert Method

Pascal

```
function Insert(Index: Integer): IImport;
```

C++

```
__fastcall IImport Insert(int Index);
```

Description

This is Insert, a member of class TImports.

189.1.13 TMessage Class

Class Hierarchy

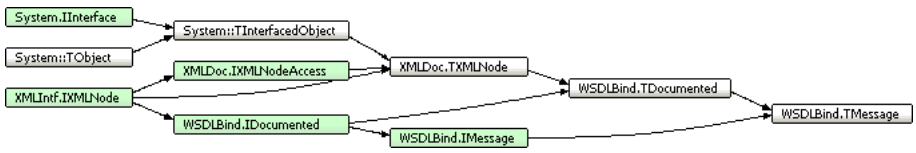

Pascal

```
TMessage = class(TDocumented, IMessage);
```

C++

```
class TMessage : public TDocumented, public IMessage;
```

File

WSDLBind

Description

This is class WSDLBind.TMessage.

189.1.13.1 TMessage Members

The following tables list the members exposed by TMessage.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.

	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetDocumentationCollection (see page 25688)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetName (see page 25714)	This is GetName, a member of class TMessage.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetParts (see page 25714)	This is GetParts, a member of class TMessage.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetName (see page 25714)	This is SetName, a member of class TMessage.

	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 25714)	Responds after the last constructor executes.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.

	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Name (see page 25856)	This is Name, a member of class IMessage.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Parts (see page 25856)	This is Parts, a member of class IMessage.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.13.2 TMessage Methods

The methods of the TMessage class are listed here.

Protected Methods

	Name	Description
	GetName (see page 25714)	This is GetName, a member of class TMessage.
	GetParts (see page 25714)	This is GetParts, a member of class TMessage.
	SetName (see page 25714)	This is SetName, a member of class TMessage.

Public Methods

	Name	Description
V	AfterConstruction (see page 25714)	Responds after the last constructor executes.

189.1.13.2.1 TMessage.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 22279)

System::TInterfacedObject::NewInstance (see page 22280)

System::TInterfacedObject::RefCount (see page 22281)

189.1.13.2.2 TMessage.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TMessage.

189.1.13.2.3 TMessage.GetParts Method

Pascal

```
function GetParts: IParts;
```

C++

```
__fastcall IParts GetParts();
```

Description

This is GetParts, a member of class TMessage.

189.1.13.2.4 TMessage.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TMessage.

189.1.14 TMessages Class

Class Hierarchy

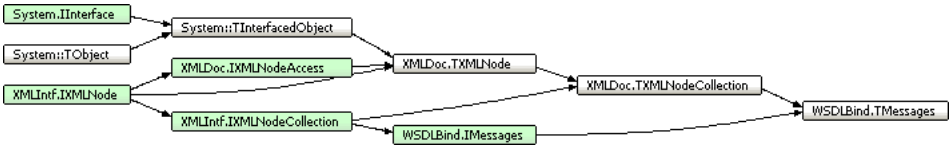

Pascal

```
TMessages = class(TXMLNodeCollection, IMessages);
```

C++

```
class TMessages : public TXMLNodeCollection, public IMessages;
```

File

WSDLBind

Description

This is class WSDLBind.TMessages.

189.1.14.1 TMessages Members

The following tables list the members exposed by TMessages.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	Add (see page 25720)	This is Add, a member of class TMessages.

	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AddItem (see page 26202)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
	ChildListNotify (see page 26203)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 26204)	Removes all child nodes in the List (see page 26210) property from the ChildNodes (see page 26190) and List (see page 26210) properties.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	CreateItemList (see page 26204)	Creates the object that implements the List (see page 26210) property.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26205)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetCount (see page 26205)	Returns the value of the Count (see page 26209) property.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetList (see page 26205)	Returns the value of the List (see page 26210) property.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetMessage (see page 25721)	This is GetMessage, a member of class TMessages.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 26206)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.

	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	Insert (see page 25721)	This is Insert, a member of class TMessages.
	InsertInCollection (see page 26206)	Copies a child node from the ChildNodes (see page 26190) property to the List (see page 26210) property.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	IsCollectionItem (see page 26207)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	Remove (see page 26207)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
V	SetChildNodes (see page 26208)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 26208)	Updates the List (see page 26210) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	Count (see page 26209)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.

	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 26209)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 26210)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 26210)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 26210)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 26202)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.

	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.

	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	Message (see page 25860)	This is Message, a member of class IMessages.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.14.2 TMessages Methods

The methods of the TMessages class are listed here.

Protected Methods

	Name	Description
	Add (see page 25720)	This is Add, a member of class TMessages.
	GetMessage (see page 25721)	This is GetMessage, a member of class TMessages.
	Insert (see page 25721)	This is Insert, a member of class TMessages.

189.1.14.2.1 TMessages.Add Method

Pascal

```
function Add(const Name: WideString): IMessage;
```

C++

```
__fastcall IMessage Add(const BSTR Name);
```

Description

This is Add, a member of class TMessages.

189.1.14.2.2 TMessages.GetMessage Method

Pascal

```
function GetMessage(Index: Integer): IMessage;
```

C++

```
__fastcall IMessage GetMessage(int Index);
```

Description

This is GetMessage, a member of class TMessages.

189.1.14.2.3 TMessages.Insert Method

Pascal

```
function Insert(Index: Integer): IMessage;
```

C++

```
__fastcall IMessage Insert(int Index);
```

Description

This is Insert, a member of class TMessages.

189.1.15 TOperation Class

Class Hierarchy

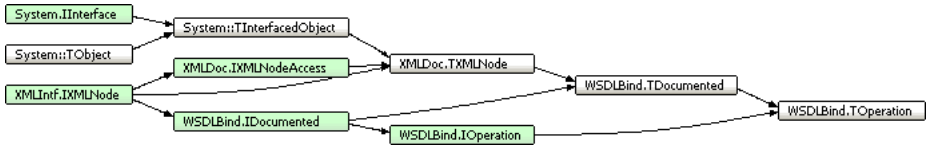

Pascal

```
TOperation = class(TDocumented, IOperation);
```

C++

```
class TOperation : public TDocumented, public IOperation;
```

File

WSDLBind

Description

This is class WSDLBind.TOperation.

189.1.15.1 TOperation Members

The following tables list the members exposed by TOperation.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.

	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetDocumentationCollection (see page 25688)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetFaults (see page 25727)	This is GetFaults, a member of class TOperation.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetInput (see page 25727)	This is GetInput, a member of class TOperation.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetName (see page 25727)	This is GetName, a member of class TOperation.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOutput (see page 25728)	This is GetOutput, a member of class TOperation.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetName (see page 25728)	This is SetName, a member of class TOperation.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 25727)	Responds after the last constructor executes.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.

	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	Faults (see page 25863)	This is Faults, a member of class IOperation.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	Input (see page 25863)	This is Input, a member of class IOperation.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Name (see page 25864)	This is Name, a member of class IOperation.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	Output (see page 25864)	This is Output, a member of class IOperation.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.15.2 TOperation Methods

The methods of the TOperation class are listed here.

Protected Methods

	Name	Description
	GetFaults (see page 25727)	This is GetFaults, a member of class TOperation.
	GetInput (see page 25727)	This is GetInput, a member of class TOperation.
	GetName (see page 25727)	This is GetName, a member of class TOperation.
	GetOutput (see page 25728)	This is GetOutput, a member of class TOperation.
	SetName (see page 25728)	This is SetName, a member of class TOperation.

Public Methods

	Name	Description
V	AfterConstruction (see page 25727)	Responds after the last constructor executes.

189.1.15.2.1 TOperation.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 22279)

System::TInterfacedObject::NewInstance (see page 22280)

System::TInterfacedObject::RefCount (see page 22281)

189.1.15.2.2 TOperation.GetFaults Method

Pascal

```
function GetFaults: IFaults;
```

C++

```
__fastcall IFaults GetFaults();
```

Description

This is GetFaults, a member of class TOperation.

189.1.15.2.3 TOperation.GetInput Method

Pascal

```
function GetInput: IParam;
```

C++

```
__fastcall IParam GetInput();
```

Description

This is GetInput, a member of class TOperation.

189.1.15.2.4 TOperation.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TOperation.

189.1.15.2.5 TOperation.GetOutput Method

Pascal

```
function GetOutput: IParam;
```

C++

```
__fastcall IParam GetOutput();
```

Description

This is GetOutput, a member of class TOperation.

189.1.15.2.6 TOperation.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TOperation.

189.1.16 TOperations Class

Class Hierarchy

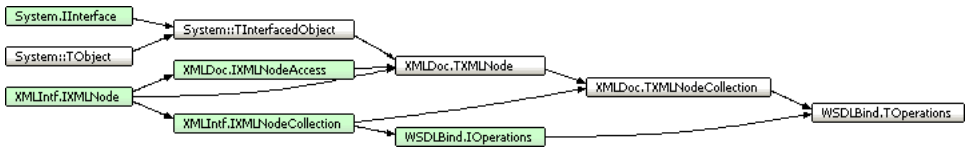

Pascal

```
TOperations = class(TXMLNodeCollection, IOperations);
```

C++

```
class TOperations : public TXMLNodeCollection, public IOperations;
```

File

WSDLBind

Description

This is class WSDLBind.TOperations.

189.1.16.1 TOperations Members

The following tables list the members exposed by TOperations.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	Add (see page 25734)	This is the overview for the Add method overload.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
V	AddItem (see page 26202)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 26203)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 26204)	Removes all child nodes in the List (see page 26210) property from the ChildNodes (see page 26190) and List (see page 26210) properties.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	CreateItemList (see page 26204)	Creates the object that implements the List (see page 26210) property.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26205)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.

	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetCount (see page 26205)	Returns the value of the Count (see page 26209) property.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
V	GetList (see page 26205)	Returns the value of the List (see page 26210) property.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 26206)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOperation (see page 25735)	This is GetOperation, a member of class TOperations.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	Insert (see page 25735)	This is Insert, a member of class TOperations.
	InsertInCollection (see page 26206)	Copies a child node from the ChildNodes (see page 26190) property to the List (see page 26210) property.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	IsCollectionItem (see page 26207)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	Remove (see page 26207)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.

	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
V	SetChildNodes (see page 26208)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 26208)	Updates the List (see page 26210) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	Count (see page 26209)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 26209)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 26210)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 26210)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 26210)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Add (see page 25866)	This is the overview for the Add method overload.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 26202)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.

	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	Operation (see page 25867)	This is Operation, a member of class IOperations.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.

R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.16.2 TOperations Methods

The methods of the TOperations class are listed here.

Protected Methods

	Name	Description
	Add (see page 25734)	This is the overview for the Add method overload.
	GetOperation (see page 25735)	This is GetOperation, a member of class TOperations.
	Insert (see page 25735)	This is Insert, a member of class TOperations.

189.1.16.2.1 Add Method

This is the overview for the Add method overload.

Overload List

	Name	Description
	TOperations.Add (WideString) (see page 25734)	This is Add, a member of class TOperations.
	TOperations.Add (WideString, WideString, WideString, WideString) (see page 25734)	This is Add, a member of class TOperations.

189.1.16.2.1.1 TOperations.Add Method (WideString)

Pascal

```
function Add(const Name: WideString): IOperation; overload;
```

C++

```
__fastcall IOperation Add(const BSTR Name);
```

Description

This is Add, a member of class TOperations.

189.1.16.2.1.2 TOperations.Add Method (WideString, WideString, WideString, WideString)

Pascal

```
function Add(const Name: WideString; const InputMessage: WideString; const InputName:
WideString; const OutputMessage: WideString): IOperation; overload;
```

C++

```
__fastcall IOperation Add(const BSTR Name, const BSTR InputMessage, const BSTR InputName,
const BSTR OutputMessage);
```

Description

This is Add, a member of class TOperations.

189.1.16.2.2 TOperations.GetOperation Method

Pascal

```
function GetOperation(Index: Integer): IOperation;
```

C++

```
__fastcall IOperation GetOperation(int Index);
```

Description

This is GetOperation, a member of class TOperations.

189.1.16.2.3 TOperations.Insert Method

Pascal

```
function Insert(Index: Integer): IOperation;
```

C++

```
__fastcall IOperation Insert(int Index);
```

Description

This is Insert, a member of class TOperations.

189.1.17 TParam Class

Class Hierarchy

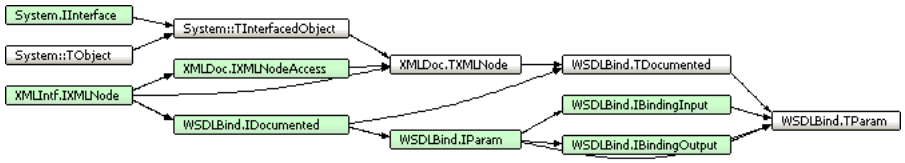

Pascal

```
TParam = class(TDocumented, IParam, IBindingInput, IBindingOutput);
```

C++

```
class TParam : public TDocumented, public IParam, public IBindingInput, public IBindingOutput;
```

File

WSDLBind

Description

This is class WSDLBind.TParam.

189.1.17.1 TParam Members

The following tables list the members exposed by TParam.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetDocumentationCollection (see page 25688)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.

	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetMessage (see page 25740)	This is GetMessage, a member of class TParam.
	GetName (see page 25741)	This is GetName, a member of class TParam.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetMessage (see page 25741)	This is SetMessage, a member of class TParam.
	SetName (see page 25741)	This is SetName, a member of class TParam.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.

R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 25688)	Responds after the last constructor executes.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.

	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.

	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Message (see page 25871)	This is Message, a member of class IParam.
	Name (see page 25871)	This is Name, a member of class IParam.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.17.2 TParam Methods

The methods of the TParam class are listed here.

Protected Methods

	Name	Description
	GetMessage (see page 25740)	This is GetMessage, a member of class TParam.
	GetName (see page 25741)	This is GetName, a member of class TParam.
	SetMessage (see page 25741)	This is SetMessage, a member of class TParam.
	SetName (see page 25741)	This is SetName, a member of class TParam.

189.1.17.2.1 TParam.GetMessage Method

Pascal

```
function GetMessage: WideString;
```

C++

```
__fastcall BSTR GetMessage();
```

Description

This is GetMessage, a member of class TParam.

189.1.17.2.2 TParam.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TParam.

189.1.17.2.3 TParam.SetMessage Method

Pascal

```
procedure SetMessage(Value: WideString);
```

C++

```
__fastcall SetMessage(BSTR Value);
```

Description

This is SetMessage, a member of class TParam.

189.1.17.2.4 TParam.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TParam.

189.1.18 TPart Class

Class Hierarchy

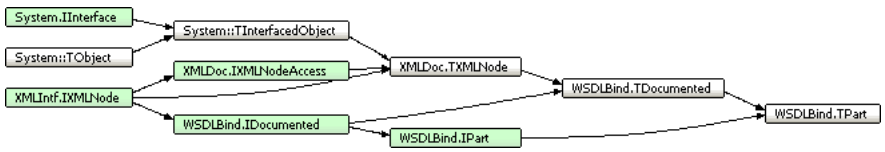

Pascal

```
TPart = class(TDocumented, IPart);
```

C++

```
class TPart : public TDocumented, public IPart;
```

File

WSDLBind

Description

This is class WSDLBind.TPart.

189.1.18.1 TPart Members

The following tables list the members exposed by TPart.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.

	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetDocumentationCollection (see page 25688)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetElement (see page 25747)	This is GetElement, a member of class TPart.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetName (see page 25747)	This is GetName, a member of class TPart.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetType_ (see page 25747)	This is GetType_, a member of class TPart.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.

	SetElement (see page 25747)	This is SetElement, a member of class TPart.
	SetName (see page 25748)	This is SetName, a member of class TPart.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	SetType_ (see page 25748)	This is SetType_, a member of class TPart.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 25688)	Responds after the last constructor executes.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.

	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
	Element (see page 25875)	This is Element, a member of class IPart.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Name (see page 25875)	This is Name, a member of class IPart.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
	Type_ (see page 25875)	This is Type_, a member of class IPart.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.18.2 TPart Methods

The methods of the TPart class are listed here.

Protected Methods

	Name	Description
	GetElement (see page 25747)	This is GetElement, a member of class TPart.
	GetName (see page 25747)	This is GetName, a member of class TPart.

	GetType_ (see page 25747)	This is GetType_, a member of class TPart.
	SetElement (see page 25747)	This is SetElement, a member of class TPart.
	SetName (see page 25748)	This is SetName, a member of class TPart.
	SetType_ (see page 25748)	This is SetType_, a member of class TPart.

189.1.18.2.1 TPart.GetElement Method

Pascal

```
function GetElement: WideString;
```

C++

```
__fastcall BSTR GetElement();
```

Description

This is GetElement, a member of class TPart.

189.1.18.2.2 TPart.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TPart.

189.1.18.2.3 TPart.GetType_ Method

Pascal

```
function GetType_: WideString;
```

C++

```
__fastcall BSTR GetType_();
```

Description

This is GetType_, a member of class TPart.

189.1.18.2.4 TPart.SetElement Method

Pascal

```
procedure SetElement(Value: WideString);
```

C++

```
__fastcall SetElement(BSTR Value);
```

Description

This is SetElement, a member of class TPart.

189.1.18.2.5 TPart.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TPart.

189.1.18.2.6 TPart.SetType_ Method

Pascal

```
procedure SetType_(Value: WideString);
```

C++

```
__fastcall SetType_(BSTR Value);
```

Description

This is SetType_, a member of class TPart.

189.1.19 TParts Class

Class Hierarchy

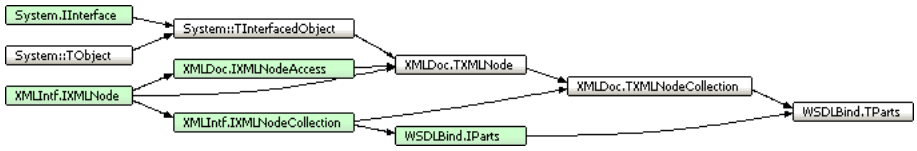

Pascal

```
TParts = class(TXMLNodeCollection, IParts);
```

C++

```
class TParts : public TXMLNodeCollection, public IParts;
```

File

WSDLBind

Description

This is class WSDLBind.TParts.

189.1.19.1 TParts Members

The following tables list the members exposed by TParts.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	Add (see page 25754)	This is Add, a member of class TParts.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
V	AddItem (see page 26202)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 26203)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 26204)	Removes all child nodes in the List (see page 26210) property from the ChildNodes (see page 26190) and List (see page 26210) properties.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	CreateItemList (see page 26204)	Creates the object that implements the List (see page 26210) property.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26205)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.

	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetCount (see page 26205)	Returns the value of the Count (see page 26209) property.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetList (see page 26205)	Returns the value of the List (see page 26210) property.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 26206)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPart (see page 25754)	This is GetPart, a member of class TParts.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	Insert (see page 25754)	This is Insert, a member of class TParts.
	InsertInCollection (see page 26206)	Copies a child node from the ChildNodes (see page 26190) property to the List (see page 26210) property.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	IsCollectionItem (see page 26207)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	Remove (see page 26207)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildNodes (see page 26208)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.

	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 26208)	Updates the List (see page 26210) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	Count (see page 26209)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 26209)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 26210)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 26210)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 26210)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 26202)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.

	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.

	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Part (see page 25879)	This is Part, a member of class IParts.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.19.2 TParts Methods

The methods of the TParts class are listed here.

Protected Methods

	Name	Description
	Add (see page 25754)	This is Add, a member of class TParts.
	GetPart (see page 25754)	This is GetPart, a member of class TParts.
	Insert (see page 25754)	This is Insert, a member of class TParts.

189.1.19.2.1 TParts.Add Method

Pascal

```
function Add(const Name: WideString; const Element: WideString; const AType: WideString = ''):
IPart;
```

C++

```
__fastcall IPart Add(const BSTR Name, const BSTR Element, const BSTR AType = '');
```

Description

This is Add, a member of class TParts.

189.1.19.2.2 TParts.GetPart Method

Pascal

```
function GetPart(Index: Integer): IPart;
```

C++

```
__fastcall IPart GetPart(int Index);
```

Description

This is GetPart, a member of class TParts.

189.1.19.2.3 TParts.Insert Method

Pascal

```
function Insert(Index: Integer): IPart;
```

C++

```
__fastcall IPart Insert(int Index);
```

Description

This is Insert, a member of class TParts.

189.1.20 TPort Class

Class Hierarchy

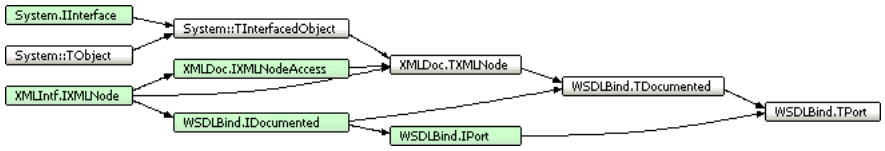

Pascal

```
TPort = class(TDocumented, IPort);
```

C++

```
class TPort : public TDocumented, public IPort;
```

File

WSDLBind

Description

This is class WSDLBind.TPort.

189.1.20.1 TPort Members

The following tables list the members exposed by TPort.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.

	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetBinding (see page 25760)	This is GetBinding, a member of class TPort.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetDocumentationCollection (see page 25688)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetName (see page 25760)	This is GetName, a member of class TPort.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.

	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetBinding (see page 25760)	This is SetBinding, a member of class TPort.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetName (see page 25761)	This is SetName, a member of class TPort.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 25688)	Responds after the last constructor executes.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.

	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
	Binding (see page 25882)	This is Binding, a member of class IPort.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Name (see page 25882)	This is Name, a member of class IPort.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.

R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.20.2 TPort Methods

The methods of the TPort class are listed here.

Protected Methods

	Name	Description
	GetBinding (see page 25760)	This is GetBinding, a member of class TPort.
	GetName (see page 25760)	This is GetName, a member of class TPort.
	SetBinding (see page 25760)	This is SetBinding, a member of class TPort.
	SetName (see page 25761)	This is SetName, a member of class TPort.

189.1.20.2.1 TPort.GetBinding Method

Pascal

```
function GetBinding: WideString;
```

C++

```
__fastcall BSTR GetBinding();
```

Description

This is GetBinding, a member of class TPort.

189.1.20.2.2 TPort.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TPort.

189.1.20.2.3 TPort.SetBinding Method

Pascal

```
procedure SetBinding(Value: WideString);
```

C++

```
__fastcall SetBinding(BSTR Value);
```

Description

This is SetBinding, a member of class TPort.

189.1.20.2.4 TPort.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TPort.

189.1.21 TPorts Class

Class Hierarchy

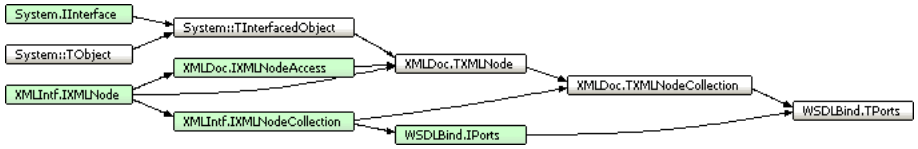

Pascal

```
TPorts = class(TXMLNodeCollection, IPorts);
```

C++

```
class TPorts : public TXMLNodeCollection, public IPorts;
```

File

WSDLBind

Description

This is class WSDLBind.TPorts.

189.1.21.1 TPorts Members

The following tables list the members exposed by TPorts.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	Add (see page 25767)	This is Add, a member of class TPorts.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
V	AddItem (see page 26202)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 26203)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 26204)	Removes all child nodes in the List (see page 26210) property from the ChildNodes (see page 26190) and List (see page 26210) properties.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	CreateItemList (see page 26204)	Creates the object that implements the List (see page 26210) property.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26205)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetCount (see page 26205)	Returns the value of the Count (see page 26209) property.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
V	GetList (see page 26205)	Returns the value of the List (see page 26210) property.

	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 26206)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPort (see page 25767)	This is GetPort, a member of class TPorts.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	Insert (see page 25767)	This is Insert, a member of class TPorts.
	InsertInCollection (see page 26206)	Copies a child node from the ChildNodes (see page 26190) property to the List (see page 26210) property.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	IsCollectionItem (see page 26207)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	Remove (see page 26207)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
V	SetChildNodes (see page 26208)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 26208)	Updates the List (see page 26210) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	Count (see page 26209)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 26209)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 26210)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 26210)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 26210)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 26202)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.

	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Port (see page 25885)	This is Port, a member of class IPorts.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.21.2 TPorts Methods

The methods of the TPorts class are listed here.

Protected Methods

	Name	Description
	Add (see page 25767)	This is Add, a member of class TPorts.
	GetPort (see page 25767)	This is GetPort, a member of class TPorts.

	Insert (see page 25767)	This is Insert, a member of class TPorts.

189.1.21.2.1 TPorts.Add Method

Pascal

```
function Add(const Name: WideString; const Binding: WideString): IPort;
```

C++

```
__fastcall IPort Add(const BSTR Name, const BSTR Binding);
```

Description

This is Add, a member of class TPorts.

189.1.21.2.2 TPorts.GetPort Method

Pascal

```
function GetPort(Index: Integer): IPort;
```

C++

```
__fastcall IPort GetPort(int Index);
```

Description

This is GetPort, a member of class TPorts.

189.1.21.2.3 TPorts.Insert Method

Pascal

```
function Insert(Index: Integer): IPort;
```

C++

```
__fastcall IPort Insert(int Index);
```

Description

This is Insert, a member of class TPorts.

189.1.22 TPortType Class

Class Hierarchy

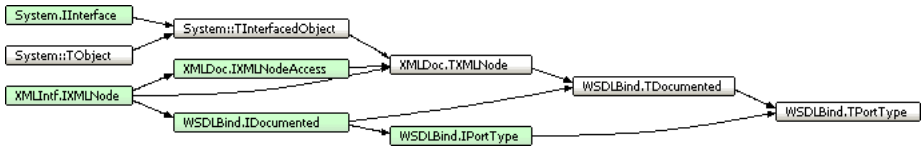

Pascal

```
TPortType = class(TDocumented, IPortType);
```

C++

```
class TPortType : public TDocumented, public IPortType;
```

File

WSDLBind

Description

This is class WSDLBind.TPortType.

189.1.22.1 TPortType Members

The following tables list the members exposed by TPortType.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.

	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetDocumentationCollection (see page 25688)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetName (see page 25773)	This is GetName, a member of class TPortType.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOperations (see page 25773)	This is GetOperations, a member of class TPortType.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetName (see page 25773)	This is SetName, a member of class TPortType.

	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 25773)	Responds after the last constructor executes.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.

	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Name (see page 25888)	This is Name, a member of class IPortType.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	Operations (see page 25888)	This is Operations, a member of class IPortType.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.22.2 TPortType Methods

The methods of the TPortType class are listed here.

Protected Methods

	Name	Description
	GetName (see page 25773)	This is GetName, a member of class TPortType.
	GetOperations (see page 25773)	This is GetOperations, a member of class TPortType.
	SetName (see page 25773)	This is SetName, a member of class TPortType.

Public Methods

	Name	Description
V	AfterConstruction (see page 25773)	Responds after the last constructor executes.

189.1.22.2.1 TPortType.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 22279)

System::TInterfacedObject::NewInstance (see page 22280)

System::TInterfacedObject::RefCount (see page 22281)

189.1.22.2.2 TPortType.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TPortType.

189.1.22.2.3 TPortType.GetOperations Method

Pascal

```
function GetOperations: IOperations;
```

C++

```
__fastcall IOperations GetOperations();
```

Description

This is GetOperations, a member of class TPortType.

189.1.22.2.4 TPortType.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TPortType.

189.1.23 TPortTypes Class

Class Hierarchy

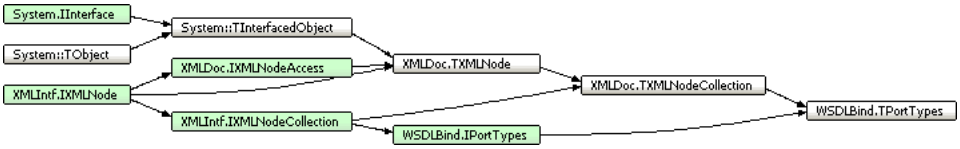

Pascal

```
TPortTypes = class(TXMLNodeCollection, IPortTypes);
```

C++

```
class TPortTypes : public TXMLNodeCollection, public IPortTypes;
```

File

WSDLBind

Description

This is class WSDLBind.TPortTypes.

189.1.23.1 TPortTypes Members

The following tables list the members exposed by TPortTypes.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	Add (see page 25779)	This is Add, a member of class TPortTypes.

	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AddItem (see page 26202)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
	ChildListNotify (see page 26203)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 26204)	Removes all child nodes in the List (see page 26210) property from the ChildNodes (see page 26190) and List (see page 26210) properties.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	CreateItemList (see page 26204)	Creates the object that implements the List (see page 26210) property.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26205)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetCount (see page 26205)	Returns the value of the Count (see page 26209) property.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetList (see page 26205)	Returns the value of the List (see page 26210) property.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 26206)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.

	GetPortType (see page 25780)	This is GetPortType, a member of class TPortTypes.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	Insert (see page 25780)	This is Insert, a member of class TPortTypes.
	InsertInCollection (see page 26206)	Copies a child node from the ChildNodes (see page 26190) property to the List (see page 26210) property.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	IsCollectionItem (see page 26207)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	Remove (see page 26207)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
V	SetChildNodes (see page 26208)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 26208)	Updates the List (see page 26210) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	Count (see page 26209)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.

	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 26209)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 26210)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 26210)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 26210)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 26202)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.

	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.

	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	PortType (see page 25892)	This is PortType, a member of class IPortTypes.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.23.2 TPortTypes Methods

The methods of the TPortTypes class are listed here.

Protected Methods

	Name	Description
	Add (see page 25779)	This is Add, a member of class TPortTypes.
	GetPortType (see page 25780)	This is GetPortType, a member of class TPortTypes.
	Insert (see page 25780)	This is Insert, a member of class TPortTypes.

189.1.23.2.1 TPortTypes.Add Method

Pascal

```
function Add(const Name: WideString): IPortType; overload;
```

C++

```
__fastcall IPortType Add(const BSTR Name);
```

Description

This is Add, a member of class TPortTypes.

189.1.23.2.2 TPortTypes.GetPortType Method

Pascal

```
function GetPortType(Index: Integer): IPortType;
```

C++

```
__fastcall IPortType GetPortType(int Index);
```

Description

This is GetPortType, a member of class TPortTypes.

189.1.23.2.3 TPortTypes.Insert Method

Pascal

```
function Insert(Index: Integer): IPortType;
```

C++

```
__fastcall IPortType Insert(int Index);
```

Description

This is Insert, a member of class TPortTypes.

189.1.24 TService Class

Class Hierarchy

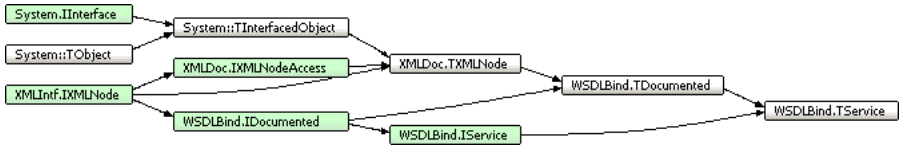

Pascal

```
TService = class(TDocumented, IService);
```

C++

```
class TService : public TDocumented, public IService;
```

File

WSDLBind

Description

This is class WSDLBind.TService.

189.1.24.1 TService Members

The following tables list the members exposed by TService.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.

	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetDocumentationCollection (see page 25688)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetName (see page 25786)	This is GetName, a member of class TService.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPorts (see page 25786)	This is GetPorts, a member of class TService.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetName (see page 25786)	This is SetName, a member of class TService.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 25785)	Responds after the last constructor executes.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.

	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.

	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Name (see page 25895)	This is Name, a member of class IService.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Ports (see page 25895)	This is Ports, a member of class IService.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.24.2 TService Methods

The methods of the TService class are listed here.

Protected Methods

	Name	Description
	GetName (see page 25786)	This is GetName, a member of class TService.
	GetPorts (see page 25786)	This is GetPorts, a member of class TService.
	SetName (see page 25786)	This is SetName, a member of class TService.

Public Methods

	Name	Description
V	AfterConstruction (see page 25785)	Responds after the last constructor executes.

189.1.24.2.1 TService.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 22279)

System::TInterfacedObject::NewInstance (see page 22280)

System::TInterfacedObject::RefCount (see page 22281)

189.1.24.2.2 TService.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TService.

189.1.24.2.3 TService.GetPorts Method

Pascal

```
function GetPorts: IPorts;
```

C++

```
__fastcall IPorts GetPorts();
```

Description

This is GetPorts, a member of class TService.

189.1.24.2.4 TService.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TService.

189.1.25 TServices Class

Class Hierarchy

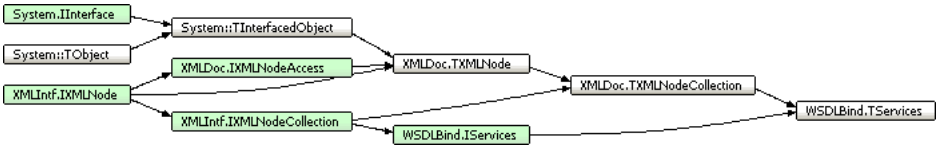

Pascal

```
TServices = class(TXMLNodeCollection, IServices);
```

C++

```
class TServices : public TXMLNodeCollection, public IServices;
```

File

WSDLBind

Description

This is class WSDLBind.TServices.

189.1.25.1 TServices Members

The following tables list the members exposed by TServices.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	Add (see page 25792)	This is Add, a member of class TServices.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AddItem (see page 26202)	Adds a new repetition of the repeating child element to this node.

	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 26203)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 26204)	Removes all child nodes in the List (see page 26210) property from the ChildNodes (see page 26190) and List (see page 26210) properties.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	CreateItemList (see page 26204)	Creates the object that implements the List (see page 26210) property.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26205)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetCount (see page 26205)	Returns the value of the Count (see page 26209) property.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
V	GetList (see page 26205)	Returns the value of the List (see page 26210) property.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 26206)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.

	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetService (see page 25793)	This is GetService, a member of class TServices.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	Insert (see page 25793)	This is Insert, a member of class TServices.
	InsertInCollection (see page 26206)	Copies a child node from the ChildNodes (see page 26190) property to the List (see page 26210) property.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	IsCollectionItem (see page 26207)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	Remove (see page 26207)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
V	SetChildNodes (see page 26208)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 26208)	Updates the List (see page 26210) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	Count (see page 26209)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 26209)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 26210)	This is ItemNS, a member of class TXMLNodeCollection.

	ItemTag (see page 26210)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 26210)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 26202)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.

	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.

	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	Service (see page 25898)	This is Service, a member of class IServices.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.25.2 TServices Methods

The methods of the TServices class are listed here.

Protected Methods

	Name	Description
	Add (see page 25792)	This is Add, a member of class TServices.
	GetService (see page 25793)	This is GetService, a member of class TServices.
	Insert (see page 25793)	This is Insert, a member of class TServices.

189.1.25.2.1 TServices.Add Method

Pascal

```
function Add(const Name: WideString): IService;
```

C++

```
__fastcall IService Add(const BSTR Name);
```

Description

This is Add, a member of class TServices.

189.1.25.2.2 TServices.GetService Method

Pascal

```
function GetService(Index: Integer): IService;
```

C++

```
__fastcall IService GetService(int Index);
```

Description

This is GetService, a member of class TServices.

189.1.25.2.3 TServices.Insert Method

Pascal

```
function Insert(Index: Integer): IService;
```

C++

```
__fastcall IService Insert(int Index);
```

Description

This is Insert, a member of class TServices.

189.1.26 TTypes Class

Class Hierarchy

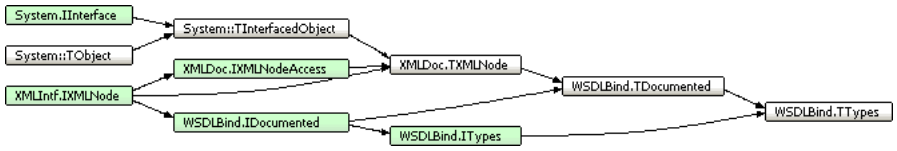

Pascal

```
TTypes = class(TDocumented, ITypes);
```

C++

```
class TTypes : public TDocumented, public ITypes;
```

File

WSDLBind

Description

This is class WSDLBind.TTypes.

189.1.26.1 TTypes Members

The following tables list the members exposed by TTypes.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetDocumentationCollection (see page 25688)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.

	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetSchemaDefs (see page 25799)	This is GetSchemaDefs, a member of class TTypes.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.

R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 25798)	Responds after the last constructor executes.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.

	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	SchemaDefs (see page 25900)	This is SchemaDefs, a member of class ITypes.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.26.2 TTypes Methods

The methods of the TTypes class are listed here.

Protected Methods

	Name	Description
	GetSchemaDefs (see page 25799)	This is GetSchemaDefs, a member of class TTypes.

Public Methods

	Name	Description
V	AfterConstruction (see page 25798)	Responds after the last constructor executes.

189.1.26.2.1 TTypes.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 22279)

System::TInterfacedObject::NewInstance (see page 22280)

System::TInterfacedObject::RefCount (see page 22281)

189.1.26.2.2 TTypes.GetSchemaDefs Method

Pascal

```
function GetSchemaDefs: IXMLSchemaDefs;
```

C++

```
__fastcall IXMLSchemaDefs GetSchemaDefs();
```

Description

This is GetSchemaDefs, a member of class TTypes.

189.1.27 TWSDLDocument Class

TWSDLDocument represents a WSDL document.

Class Hierarchy

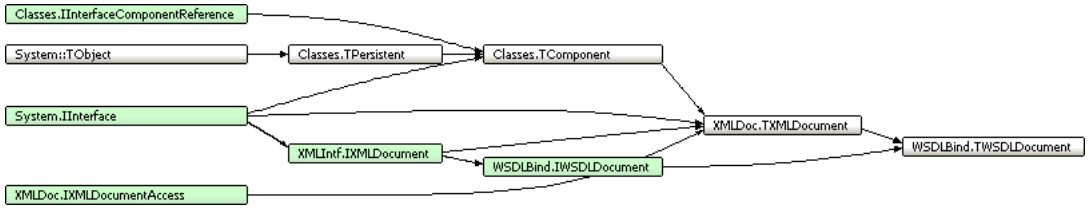

Pascal

```
TWSDLDocument = class(TXMLDocument, IWSDLDocument);
```

C++

```
class TWSDLDocument : public TXMLDocument, public IWSDLDocument;
```

File

WSDLBind

Description

Use TWSDLDocument to represent a WSDL (Web Services Definition (see page 25808) Language) document. A WSDL document is a special form of XML document that describes the contents of a Web Service application and how you call the services it implements.

TWSDLDocument can read an existing WSDL document from a file, it can be associated with an in-memory string that is the contents of a WSDL document, or it can create a new, empty WSDL document. TWSDLDocument uses an external DOM

(Document Object Model) parser to analyze the WSDL document. This parser is usually assigned by specifying the DOMVendor property. When you set the Active property to true, TWSDLDocument uses the DOM parser to parse the WSDL document so that you can examine or modify it.

When TWSDLDocument is created without an Owner, it behaves like an interfaced object. That is, when all references to its interface are released, the TWSDLDocument instance is automatically freed. When TWSDLDocument is created with an Owner, however, it behaves like any other component, and is freed by its Owner.

Tip: Although you can work directly with TWSDLDocument to access a WSDL document, it is often easier to use TWSDLItems, a TWSDLDocument descendant. TWSDLItems implements a number of methods for getting at the definitions in a WSDL document.

See Also

TWSDLItems (see page 25951)

TWSDLView (see page 26013)

189.1.27.1 TWSDLDocument Members

The following tables list the members exposed by TWSDLDocument.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_AddRef (see page 26106)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	_Release (see page 26106)	Called when an application releases a component interface.
	AssignParseOptions (see page 26108)	This is AssignParseOptions, a member of class TXMLDocument.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckActive (see page 26108)	This is CheckActive, a member of class TXMLDocument.
	CheckAutoSave (see page 26108)	This is CheckAutoSave, a member of class TXMLDocument.
	CheckDOM (see page 26109)	This is CheckDOM, a member of class TXMLDocument.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 26111)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoAfterClose (see page 26111)	This is DoAfterClose, a member of class TXMLDocument.
	DoAfterOpen (see page 26111)	This is DoAfterOpen, a member of class TXMLDocument.
	DoBeforeClose (see page 26112)	This is DoBeforeClose, a member of class TXMLDocument.
	DoBeforeOpen (see page 26112)	This is DoBeforeOpen, a member of class TXMLDocument.
	DoNodeChange (see page 26112)	This is DoNodeChange, a member of class TXMLDocument.
V	GetActive (see page 26113)	Returns the value of the Active property.
	GetAsyncLoadState (see page 26113)	Returns the value of the AsyncLoadState property.
V	GetChildNodeClass (see page 26113)	This is GetChildNodeClass, a member of class TXMLDocument.
	GetChildNodes (see page 26113)	Returns the value of the ChildNodes property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.

	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDefinition (see page 25807)	Returns the value of the Definition property.
	GetDocumentElement (see page 26114)	Returns the value of the DocumentElement property.
	GetDocumentNode (see page 26115)	Returns the value of the Node property.
	GetDocumentObject (see page 26115)	This is GetDocumentObject, a member of class TXMLDocument.
	GetDOMDocument (see page 26115)	Returns the value of the DOMDocument property.
	GetDOMPersist (see page 26115)	This is GetDOMPersist, a member of class TXMLDocument.
	GetEncoding (see page 26116)	Returns the value of the Encoding property.
	GetFileName (see page 26116)	Returns the value of the FileName property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetModified (see page 26116)	Returns the value of the Modified property.
	GetNodeIndentStr (see page 26117)	Returns the value of the NodeIndentStr property.
	GetOptions (see page 26117)	Returns the value of the Options property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetParseOptions (see page 26117)	Returns the value of the ParseOptions property.
	GetPrologNode (see page 26117)	This is GetPrologNode, a member of class TXMLDocument.
	GetPrologValue (see page 26118)	This is GetPrologValue, a member of class TXMLDocument.
	GetSchemaRef (see page 26118)	Returns the value of the SchemaRef property.
	GetStandAlone (see page 26118)	Returns the value of the StandAlone property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVersion (see page 26118)	Returns the value of the Version property.
	GetXML (see page 26119)	Returns the value of the XML property.
	InternalSetPrologValue (see page 26119)	This is InternalSetPrologValue, a member of class TXMLDocument.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	LoadData (see page 25807)	This is LoadData, a member of class TWSDLDocument.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 26120)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadDOMVendor (see page 26123)	This is ReadDOMVendor, a member of class TXMLDocument.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReleaseDoc (see page 26124)	This is ReleaseDoc, a member of class TXMLDocument.
	SaveToUTF8String (see page 26126)	This is SaveToUTF8String, a member of class TXMLDocument.
	SaveToXMLStrings (see page 26127)	This is SaveToXMLStrings, a member of class TXMLDocument.
V	SetActive (see page 26127)	Sets the value of the Active property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDefinition (see page 25807)	Sets the value of the Definition property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDocumentElement (see page 26127)	Sets the value of the DocumentElement property.
	SetDOMDocument (see page 26128)	Sets the value of the DOMDocument property.
	SetEncoding (see page 26128)	Sets the value of the Encoding property.
	SetFileName (see page 26128)	Sets the value of the FileName property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetModified (see page 26129)	This is SetModified, a member of class TXMLDocument.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.

	SetNodeIndentStr (see page 26129)	Sets the value of the NodeIndentStr property.
	SetOnAsyncLoad (see page 26129)	Supplies an event handler that responds when the AsyncLoadState property changes.
	SetOptions (see page 26130)	Sets the value of the Options property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParseOptions (see page 26130)	Sets the value of the ParseOptions property.
	SetPrologValue (see page 26130)	This is SetPrologValue, a member of class TXMLDocument.
	SetStandAlone (see page 26130)	Sets the value of the StandAlone property.
	SetVersion (see page 26131)	Sets the value of the Version property.
	SetXML (see page 26131)	Sets the value of the XML property.
	SetXMLStrings (see page 26131)	This is SetXMLStrings, a member of class TXMLDocument.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteDOMVendor (see page 26132)	This is WriteDOMVendor, a member of class TXMLDocument.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.
	XMLStringsChanging (see page 26132)	This is XMLStringsChanging, a member of class TXMLDocument.

Protected Properties

	Name	Description
R	DocBindingInfo (see page 26135)	This is DocBindingInfo, a member of class TXMLDocument.
	DocSource (see page 26135)	This is DocSource, a member of class TXMLDocument.
R	DOMParseOptions (see page 26137)	This is DOMParseOptions, a member of class TXMLDocument.
R	DOMPersist (see page 26137)	This is DOMPersist, a member of class TXMLDocument.
	PrefixID (see page 26142)	This is PrefixID, a member of class TXMLDocument.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26107)	This is the overview for the AddChild method overload.
	AddChild (see page 26385)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 25807)	Initializes the TWSDLDocument (see page 25799) object after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26103)	This is the overview for the Create constructor overload.
	CreateElement (see page 26109)	Creates a new element node that is associated with this document.

	CreateElement (see page 26386)	Creates a new element node that is associated with this document.
	CreateNode (see page 26109)	Creates a new node that is associated with this document.
	CreateNode (see page 26387)	Creates a new node that is associated with this document.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26104)	Disposes of a TXMLDocument (see page 26097) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GeneratePrefix (see page 26112)	Generates a unique namespace prefix that can be used for a node in the document.
	GeneratePrefix (see page 26388)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 26388)	Returns the value of the Active (see page 26403) property.
	GetAsyncLoadState (see page 26389)	Returns the value of the AsyncLoadState (see page 26404) property.
	GetChildNodes (see page 26389)	Returns the value of the ChildNodes (see page 26405) property.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetDocBinding (see page 26114)	Replaces the document element with an instance of a specified implementation class.
	GetDocBinding (see page 26389)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 26390)	Returns the value of the DocumentElement (see page 26405) property.
	GetDocumentNode (see page 26390)	Returns the value of the Node (see page 26407) property.
	GetDocumentObject (see page 26229)	This is GetDocumentObject, a member of class IXMLDocumentAccess.
	GetDOMDocument (see page 26390)	Returns the value of the DOMDocument (see page 26405) property.
	GetDOMPersist (see page 26229)	This is GetDOMPersist, a member of class IXMLDocumentAccess.
	GetEncoding (see page 26391)	Returns the value of the Encoding (see page 26406) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFileName (see page 26391)	Returns the value of the FileName (see page 26406) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetModified (see page 26391)	Returns the value of the Modified (see page 26407) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNodeIndentStr (see page 26392)	Returns the value of the NodeIndentStr (see page 26408) property.
	GetOptions (see page 26392)	Returns the value of the Options (see page 26408) property.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParseOptions (see page 26392)	Returns the value of the ParseOptions (see page 26409) property.
	GetSchemaRef (see page 26392)	Returns the value of the SchemaRef (see page 26409) property.
	GetStandAlone (see page 26393)	Returns the value of the StandAlone (see page 26410) property.
	GetVersion (see page 26393)	Returns the value of the Version (see page 26410) property.
	GetXML (see page 26393)	Returns the value of the XML (see page 26410) property.

	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmptyDoc (see page 26119)	Indicates whether the TXMLDocument (see page 26097) instance represents an empty document.
	IsEmptyDoc (see page 26394)	Indicates whether the IXMLDocument (see page 26382) instance represents an empty document.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LoadFromFile (see page 26121)	Loads an XML (see page 26144) document and activates it.
	LoadFromFile (see page 26394)	Loads an XML (see page 26410) document from disk.
	LoadFromStream (see page 26121)	Loads an XML (see page 26144) document from a stream and activates it.
	LoadFromStream (see page 26395)	Loads an XML (see page 26410) document from a stream and activates it.
	LoadFromXML (see page 26122)	This is the overview for the LoadFromXML method overload.
	LoadFromXML (see page 26395)	This is the overview for the LoadFromXML method overload.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	NewInstance (see page 26122)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 26123)	Updates the parsed XML (see page 26144) document to reflect external changes.
	Refresh (see page 26396)	Updates the parsed XML (see page 26410) document to reflect external changes.
	RegisterDocBinding (see page 26124)	Registers an implementation class for child nodes with a specified tag name.
	RegisterDocBinding (see page 26396)	Registers an implementation class for child nodes with a specified tag name.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Resync (see page 26125)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	Resync (see page 26397)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 26125)	Saves the XML (see page 26144) document to disk.
	SaveToFile (see page 26397)	Saves the XML (see page 26410) document to disk.
	SaveToStream (see page 26125)	Saves the XML (see page 26144) document to a stream.
	SaveToStream (see page 26398)	Saves the XML (see page 26410) document to a stream.
	SaveToXML (see page 26126)	This is the overview for the SaveToXML method overload.
	SaveToXML (see page 26398)	This is the overview for the SaveToXML method overload.
	SetActive (see page 26399)	Sets the value of the Active (see page 26403) property.
	SetDocumentElement (see page 26399)	Sets the value of the DocumentElement (see page 26405) property.
	SetDOMDocument (see page 26399)	Sets the value of the DOMDocument (see page 26405) property.
	SetEncoding (see page 26400)	Sets the value of the Encoding (see page 26406) property.
	SetFileName (see page 26400)	Sets the value of the FileName (see page 26406) property.
	SetNodeIndentStr (see page 26400)	Sets the value of the NodeIndentStr (see page 26408) property.
	SetOnAsyncLoad (see page 26401)	Supplies an event handler that responds when the AsyncLoadState (see page 26404) property changes.
	SetOptions (see page 26401)	Sets the value of the Options (see page 26408) property.
	SetParseOptions (see page 26402)	Sets the value of the ParseOptions (see page 26409) property.

	SetStandAlone (see page 26402)	Sets the value of the StandAlone (see page 26410) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetVersion (see page 26402)	Sets the value of the Version (see page 26410) property.
	SetXML (see page 26403)	Sets the value of the XML (see page 26410) property.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Active (see page 26403)	Specifies whether the XML (see page 26410) document has been parsed so that the DOM interfaces are available to examine or change its contents.
R	AsyncLoadState (see page 26134)	Indicates the state of the DOM parser when it parses the XML (see page 26144) document asynchronously.
R	AsyncLoadState (see page 26404)	Indicates the state of the DOM parser when it parses the XML (see page 26410) document asynchronously.
R	ChildNodes (see page 26135)	Lists all of the child nodes of the document.
R	ChildNodes (see page 26405)	Lists all of the child nodes of the document.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Definition (see page 25808)	Provides access to the definitions contained in the WSDL document.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DocumentElement (see page 26135)	Provides access to the root node of the XML (see page 26144) document.
	DocumentElement (see page 26405)	Provides access to the root node of the XML (see page 26410) document.
R	DocumentObject (see page 26229)	This is DocumentObject, a member of class IXMLDocumentAccess.
	DOMDocument (see page 26136)	Provides access to the DOM interface for the parsed XML (see page 26144) document.
	DOMDocument (see page 26405)	Provides access to the DOM interface for the parsed XML (see page 26410) document.
	DOMImplementation (see page 26137)	This is DOMImplementation, a member of class TXMLDocument.
R	DOMPersist (see page 26230)	This is DOMPersist, a member of class IXMLDocumentAccess.
	Encoding (see page 26138)	Indicates the character set that the XML (see page 26144) document uses to encode characters.
	Encoding (see page 26406)	Indicates the character set that the XML (see page 26410) document uses to encode characters.
	FileName (see page 26406)	Indicates the XML (see page 26410) document file, if any, that this interface represents.
R	Modified (see page 26139)	Indicates whether the XML (see page 26144) document has been modified.
R	Modified (see page 26407)	Indicates whether the XML (see page 26410) document has been modified.
R	Node (see page 26140)	Provides access to the document node for the XML (see page 26144) document.
R	Node (see page 26407)	Provides access to the document node for the XML (see page 26410) document.
	NodeIndentStr (see page 26408)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 26410) text.
	NSPrefixBase (see page 26141)	Specifies the base string for automatically-generated namespace prefixes.
	Options (see page 26408)	Influences various aspects of the document's behavior.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParseOptions (see page 26409)	Specifies implementation-specific options that are followed when parsing the XML (see page 26410) document.
R	SchemaRef (see page 26142)	Returns the name of the schema associated with this XML (see page 26144) document.
R	SchemaRef (see page 26409)	Returns the name of the schema associated with this XML (see page 26410) document.

	StandAlone (see page 26143)	Indicates whether the XML (see page 26144) document includes any external markup declarations.
	StandAlone (see page 26410)	Indicates whether the XML (see page 26410) document includes any external markup declarations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Version (see page 26143)	Indicates the XML (see page 26144) version that the document uses.
	Version (see page 26410)	Indicates the XML (see page 26410) version that the document uses.
	XML (see page 26410)	Specifies the content of the XML document associated with this IXMLDocument (see page 26382) interface.

Published Events

	Name	Description
	AfterClose (see page 26144)	Occurs after the XML (see page 26144) document is closed.
	AfterNodeChange (see page 26145)	Occurs after a change is made to one of the nodes in the document.
	AfterOpen (see page 26145)	Occurs immediately after the XML (see page 26144) document is opened.
	BeforeClose (see page 26146)	Occurs immediately before the XML (see page 26144) document is closed.
	BeforeNodeChange (see page 26146)	Occurs before a change is made to one of the nodes in the document.
	BeforeOpen (see page 26147)	Occurs immediately before the XML (see page 26144) document is opened.
	OnAsyncLoad (see page 26147)	Occurs at various stages during the asynchronous parsing of an XML (see page 26144) document.

Published Properties

	Name	Description
	Active (see page 26133)	Specifies whether the XML (see page 26144) document has been parsed so that the DOM interfaces are available to examine or change its contents.
	DOMVendor (see page 26137)	Specifies the DOM implementation to use for parsing and manipulating the XML (see page 26144) document.
	FileName (see page 26138)	Specifies the file name for the XML (see page 26144) document associated with this TXMLDocument (see page 26097) object.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NodeIndentStr (see page 26140)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 26144) text.
	Options (see page 26141)	Influences various aspects of the document's behavior.
	ParseOptions (see page 26142)	Specifies implementation-specific options that are followed when parsing the XML (see page 26144) document.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XML (see page 26144)	Specifies the content of the XML document associated with this TXMLDocument (see page 26097) object.

189.1.27.2 TWSDLDocument Methods

The methods of the TWSDLDocument class are listed here.

Protected Methods

	Name	Description
	GetDefinition (see page 25807)	Returns the value of the Definition property.
	LoadData (see page 25807)	This is LoadData, a member of class TWSDLDocument.
	SetDefinition (see page 25807)	Sets the value of the Definition property.

Public Methods

	Name	Description
	AfterConstruction (see page 25807)	Initializes the TWSDLDocument (see page 25799) object after the last constructor has executed.

189.1.27.2.1 TWSDLDocument.AfterConstruction Method

Initializes the TWSDLDocument (see page 25799) object after the last constructor has executed.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

The AfterConstruction method is called automatically after the last constructor has finished executing. TWSDLDocument (see page 25799) overrides this method to register an implementation class for the document element. That class implements the IDefinition (see page 25831) interface that is the value of the Definition (see page 25808) property.

See Also

RegisterDocBinding (see page 26124)

Definition (see page 25808)

189.1.27.2.2 TWSDLDocument.GetDefinition Method

Returns the value of the Definition property.

Pascal

```
function GetDefinition: IDefinition;
```

C++

```
__fastcall IDefinition GetDefinition();
```

Description

GetDefinition is the read access method for the Definition property.

See Also

Definition (see page 25904)

189.1.27.2.3 TWSDLDocument.LoadData Method

Pascal

```
procedure LoadData; override;
```

C++

```
virtual __fastcall LoadData();
```

Description

This is LoadData, a member of class TWSDLDocument.

189.1.27.2.4 TWSDLDocument.SetDefinition Method

Sets the value of the Definition property.

Pascal

```
procedure SetDefinition(const Value: IDefinition);
```

C++

```
__fastcall SetDefinition(const IDefinition Value);
```

Description

SetDefinition is the write method for the Definition property.

See Also

Definition (see page 25904)

189.1.27.3 TWSDLDocument Properties

The properties of the TWSDLDocument class are listed here.

Public Properties

	Name	Description
	Definition (see page 25808)	Provides access to the definitions contained in the WSDL document.

189.1.27.3.1 TWSDLDocument.Definition Property

Provides access to the definitions contained in the WSDL document.

Pascal

```
property Definition: IDefinition;
```

C++

```
__property IDefinition Definition;
```

Description

Definition is the IDefinition (see page 25831) interface of the root node in the WSDL document. This interface is implemented by the node specified by DocumentElement.

You can use the IDefinition (see page 25831) interface to access the Bindings, Messages (see page 16211), PortTypes, Services, and Types (see page 23293) defined in the WSDL document by reading properties of the same names. Alternately, you can use the methods of TWSDLItems (a TWSDLDocument (see page 25799) descendant) to access those definitions.

See Also

DocumentElement (see page 26135)

TWSDLItems (see page 25951)

189.1.28 TXMLSchemaDefs Class

Class Hierarchy

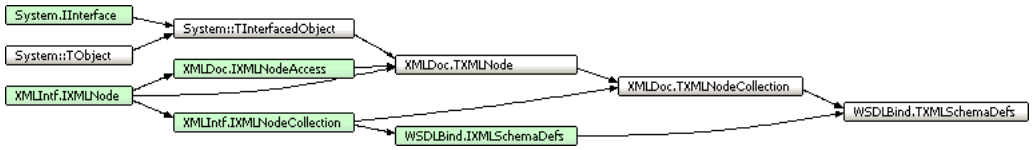

Pascal

```
TXMLSchemaDefs = class(TXMLNodeCollection, IXMLSchemaDefs);
```

C++

```
class TXMLSchemaDefs : public TXMLNodeCollection, public IXMLSchemaDefs;
```

File

WSDLBind

Description

This is class WSDLBind.TXMLSchemaDefs.

189.1.28.1 TXMLSchemaDefs Members

The following tables list the members exposed by TXMLSchemaDefs.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	Add (see page 25814)	This is Add, a member of class TXMLSchemaDefs.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
V	AddItem (see page 26202)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 26203)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 26204)	Removes all child nodes in the List (see page 26210) property from the ChildNodes (see page 26190) and List (see page 26210) properties.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.

	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	CreateItemList (see page 26204)	Creates the object that implements the List (see page 26210) property.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26205)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetCount (see page 26205)	Returns the value of the Count (see page 26209) property.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetList (see page 26205)	Returns the value of the List (see page 26210) property.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 26206)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	GetXMLSchemaDef (see page 25814)	This is GetXMLSchemaDef, a member of class TXMLSchemaDefs.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	InsertInCollection (see page 26206)	Copies a child node from the ChildNodes (see page 26190) property to the List (see page 26210) property.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	IsCollectionItem (see page 26207)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.

	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	Remove (see page 26207)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
V	SetChildNodes (see page 26208)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 26208)	Updates the List (see page 26210) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	Count (see page 26209)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 26209)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 26210)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 26210)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 26210)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 26202)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.

	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.

	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.

R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
R	SchemaDef (see page 25907)	This is SchemaDef, a member of class IXMLSchemaDefs.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.1.28.2 TXMLSchemaDefs Methods

The methods of the TXMLSchemaDefs class are listed here.

Protected Methods

	Name	Description
	Add (see page 25814)	This is Add, a member of class TXMLSchemaDefs.
	GetXMLSchemaDef (see page 25814)	This is GetXMLSchemaDef, a member of class TXMLSchemaDefs.

189.1.28.2.1 TXMLSchemaDefs.Add Method

Pascal

```
function Add(const NamespaceURI: WideString = ''; const TargetNamespace: WideString = ''):
IXMLSchemaDef;
```

C++

```
__fastcall IXMLSchemaDef Add(const BSTR NamespaceURI = '', const BSTR TargetNamespace = '');
```

Description

This is Add, a member of class TXMLSchemaDefs.

189.1.28.2.2 TXMLSchemaDefs.GetXMLSchemaDef Method

Pascal

```
function GetXMLSchemaDef(Index: Integer): IXMLSchemaDef;
```

C++

```
__fastcall IXMLSchemaDef GetXMLSchemaDef(int Index);
```

Description

This is GetXMLSchemaDef, a member of class TXMLSchemaDefs.

189.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IBinding (see page 25815)	This is class WSDLBind.IBinding.
	IBindingInput (see page 25819)	This is class WSDLBind.IBindingInput.
	IBindingOperation (see page 25820)	This is class WSDLBind.IBindingOperation.
	IBindingOperations (see page 25824)	This is class WSDLBind.IBindingOperations.
	IBindingOutput (see page 25827)	This is class WSDLBind.IBindingOutput.
	IBindings (see page 25828)	This is class WSDLBind.IBindings.
	IDefinition (see page 25831)	This is class WSDLBind.IDefinition.
	IDocumentation (see page 25837)	This is class WSDLBind.IDocumentation.
	IDocumentationCollection (see page 25838)	This is class WSDLBind.IDocumentationCollection.
	IDocumented (see page 25841)	This is class WSDLBind.IDocumented.
	IFault (see page 25843)	This is class WSDLBind.IFault.
	IFaults (see page 25844)	This is class WSDLBind.IFaults.
	IImport (see page 25847)	This is class WSDLBind.IImport.
	IImports (see page 25850)	This is class WSDLBind.IImports.
	IMessage (see page 25853)	This is class WSDLBind.IMessage.
	IMessages (see page 25857)	This is class WSDLBind.IMessages.
	IOperation (see page 25860)	This is class WSDLBind.IOperation.
	IOperations (see page 25864)	This is class WSDLBind.IOperations.
	IParam (see page 25868)	This is class WSDLBind.IParam.
	IPart (see page 25871)	This is class WSDLBind.IPart.
	IParts (see page 25876)	This is class WSDLBind.IParts.
	IPort (see page 25879)	This is class WSDLBind.IPort.
	IPorts (see page 25882)	This is class WSDLBind.IPorts.
	IPortType (see page 25885)	This is class WSDLBind.IPortType.
	IPortTypes (see page 25889)	This is class WSDLBind.IPortTypes.
	IService (see page 25892)	This is class WSDLBind.IService.
	IServices (see page 25895)	This is class WSDLBind.IServices.
	ITypes (see page 25898)	This is class WSDLBind.ITypes.
	IWSDLDocument (see page 25901)	IWSDLDocument provides low-level access to the definitions in a WSDL document.
	IXMLSchemaDefs (see page 25905)	This is class WSDLBind.IXMLSchemaDefs.

189.2.1 IBinding Interface

Class Hierarchy

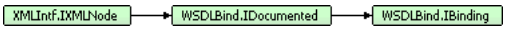

Pascal

```
IBinding = interface(IDocumented);
```

C++

```
__interface IBinding : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IBinding.

189.2.1.1 IBinding Members

The following tables list the members exposed by IBinding.

Public Methods

	Name	Description
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetBindingOperations (see page 25817)	This is GetBindingOperations, a member of class IBinding.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetName (see page 25818)	This is GetName, a member of class IBinding.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetType_ (see page 25818)	This is GetType_, a member of class IBinding.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetName (see page 25818)	This is SetName, a member of class IBinding.

	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	SetType_ (see page 25818)	This is SetType_, a member of class IBinding.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	BindingOperations (see page 25819)	This is BindingOperations, a member of class IBinding.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Name (see page 25819)	This is Name, a member of class IBinding.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
	Type_ (see page 25819)	This is Type_, a member of class IBinding.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.1.2 IBinding Methods

The methods of the IBinding class are listed here.

Public Methods

	Name	Description
	GetBindingOperations (see page 25817)	This is GetBindingOperations, a member of class IBinding.
	GetName (see page 25818)	This is GetName, a member of class IBinding.
	GetType_ (see page 25818)	This is GetType_, a member of class IBinding.
	SetName (see page 25818)	This is SetName, a member of class IBinding.
	SetType_ (see page 25818)	This is SetType_, a member of class IBinding.

189.2.1.2.1 IBinding.GetBindingOperations Method

Pascal

```
function GetBindingOperations: IBindingOperations;
```

C++

```
__fastcall IBindingOperations GetBindingOperations();
```

Description

This is GetBindingOperations, a member of class IBinding.

189.2.1.2.2 IBinding.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IBinding.

189.2.1.2.3 IBinding.GetType_ Method

Pascal

```
function GetType_: WideString;
```

C++

```
__fastcall BSTR GetType_();
```

Description

This is GetType_, a member of class IBinding.

189.2.1.2.4 IBinding.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IBinding.

189.2.1.2.5 IBinding.SetType_ Method

Pascal

```
procedure SetType_(Value: WideString);
```

C++

```
__fastcall SetType_(BSTR Value);
```

Description

This is SetType_, a member of class IBinding.

189.2.1.3 IBinding Properties

The properties of the IBinding class are listed here.

Public Properties

	Name	Description
	BindingOperations (see page 25819)	This is BindingOperations, a member of class IBinding.
	Name (see page 25819)	This is Name, a member of class IBinding.
	Type_ (see page 25819)	This is Type_, a member of class IBinding.

189.2.1.3.1 IBinding.BindingOperations Property

Pascal

```
property BindingOperations: IBindingOperations;
```

C++

```
__property IBindingOperations BindingOperations;
```

Description

This is BindingOperations, a member of class IBinding.

189.2.1.3.2 IBinding.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IBinding.

189.2.1.3.3 IBinding.Type_ Property

Pascal

```
property Type_: WideString;
```

C++

```
__property BSTR Type_;
```

Description

This is Type_, a member of class IBinding.

189.2.2 IBindingInput Interface

Class Hierarchy

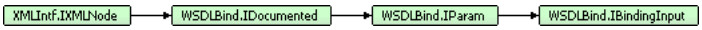

Pascal

```
IBindingInput = interface(IParam);
```

C++

```
__interface IBindingInput : public IParam;
```

File

WSDLBind

Description

This is class WSDLBind.IBindingInput.

189.2.3 IBindingOperation Interface

Class Hierarchy

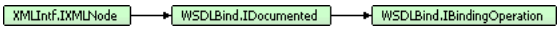

Pascal

```
IBindingOperation = interface(IDocumented);
```

C++

```
__interface IBindingOperation : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IBindingOperation.

189.2.3.1 IBindingOperation Members

The following tables list the members exposed by IBindingOperation.

Public Methods

	Name	Description
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetFaults (see page 25822)	This is GetFaults, a member of class IBindingOperation.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetInput (see page 25822)	This is GetInput, a member of class IBindingOperation.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetName (see page 25822)	This is GetName, a member of class IBindingOperation.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.

	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOutput (see page 25822)	This is GetOutput, a member of class IBindingOperation.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetName (see page 25823)	This is SetName, a member of class IBindingOperation.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	Faults (see page 25823)	This is Faults, a member of class IBindingOperation.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	Input (see page 25823)	This is Input, a member of class IBindingOperation.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Name (see page 25824)	This is Name, a member of class IBindingOperation.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	Output (see page 25824)	This is Output, a member of class IBindingOperation.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.3.2 IBindingOperation Methods

The methods of the IBindingOperation class are listed here.

Public Methods

	Name	Description
	GetFaults (see page 25822)	This is GetFaults, a member of class IBindingOperation.
	GetInput (see page 25822)	This is GetInput, a member of class IBindingOperation.
	GetName (see page 25822)	This is GetName, a member of class IBindingOperation.
	GetOutput (see page 25822)	This is GetOutput, a member of class IBindingOperation.
	SetName (see page 25823)	This is SetName, a member of class IBindingOperation.

189.2.3.2.1 IBindingOperation.GetFaults Method

Pascal

```
function GetFaults: IFaults;
```

C++

```
__fastcall IFaults GetFaults();
```

Description

This is GetFaults, a member of class IBindingOperation.

189.2.3.2.2 IBindingOperation.GetInput Method

Pascal

```
function GetInput: IBindingInput;
```

C++

```
__fastcall IBindingInput GetInput();
```

Description

This is GetInput, a member of class IBindingOperation.

189.2.3.2.3 IBindingOperation.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IBindingOperation.

189.2.3.2.4 IBindingOperation.GetOutput Method

Pascal

```
function GetOutput: IBindingOutput;
```

C++

```
__fastcall IBindingOutput GetOutput();
```

Description

This is GetOutput, a member of class IBindingOperation.

189.2.3.2.5 IBindingOperation.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IBindingOperation.

189.2.3.3 IBindingOperation Properties

The properties of the IBindingOperation class are listed here.

Public Properties

	Name	Description
R	Faults (see page 25823)	This is Faults, a member of class IBindingOperation.
R	Input (see page 25823)	This is Input, a member of class IBindingOperation.
	Name (see page 25824)	This is Name, a member of class IBindingOperation.
R	Output (see page 25824)	This is Output, a member of class IBindingOperation.

189.2.3.3.1 IBindingOperation.Faults Property

Pascal

```
property Faults: IFaults;
```

C++

```
__property IFaults Faults;
```

Description

This is Faults, a member of class IBindingOperation.

189.2.3.3.2 IBindingOperation.Input Property

Pascal

```
property Input: IBindingInput;
```

C++

```
__property IBindingInput Input;
```

Description

This is Input, a member of class IBindingOperation.

189.2.3.3.3 IBindingOperation.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IBindingOperation.

189.2.3.3.4 IBindingOperation.Output Property

Pascal

```
property Output: IBindingOutput;
```

C++

```
__property IBindingOutput Output;
```

Description

This is Output, a member of class IBindingOperation.

189.2.4 IBindingOperations Interface

Class Hierarchy

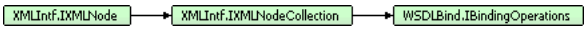

Pascal

```
IBindingOperations = interface(IXMLNodeCollection);
```

C++

```
__interface IBindingOperations : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IBindingOperations.

189.2.4.1 IBindingOperations Members

The following tables list the members exposed by IBindingOperations.

Public Methods

	Name	Description
	Add (see page 25826)	This is Add, a member of class IBindingOperations.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.

	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetBindingOperation (see page 25826)	This is GetBindingOperation, a member of class IBindingOperations.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	Insert (see page 25827)	This is Insert, a member of class IBindingOperations.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	BindingOperation (see page 25827)	This is BindingOperation, a member of class IBindingOperations.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.

R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.4.2 IBindingOperations Methods

The methods of the IBindingOperations class are listed here.

Public Methods

	Name	Description
	Add (see page 25826)	This is Add, a member of class IBindingOperations.
	GetBindingOperation (see page 25826)	This is GetBindingOperation, a member of class IBindingOperations.
	Insert (see page 25827)	This is Insert, a member of class IBindingOperations.

189.2.4.2.1 IBindingOperations.Add Method

Pascal

```
function Add(const Name: WideString): IBindingOperation;
```

C++

```
__fastcall IBindingOperation Add(const BSTR Name);
```

Description

This is Add, a member of class IBindingOperations.

189.2.4.2.2 IBindingOperations.GetBindingOperation Method

Pascal

```
function GetBindingOperation(Index: Integer): IBindingOperation;
```

C++

```
__fastcall IBindingOperation GetBindingOperation(int Index);
```

Description

This is GetBindingOperation, a member of class IBindingOperations.

189.2.4.2.3 IBindingOperations.Insert Method

Pascal

```
function Insert(Index: Integer): IBindingOperation;
```

C++

```
__fastcall IBindingOperation Insert(int Index);
```

Description

This is Insert, a member of class IBindingOperations.

189.2.4.3 IBindingOperations Properties

The properties of the IBindingOperations class are listed here.

Public Properties

	Name	Description
	BindingOperation (see page 25827)	This is BindingOperation, a member of class IBindingOperations.

189.2.4.3.1 IBindingOperations.BindingOperation Property

Pascal

```
property BindingOperation [Index: Integer]: IBindingOperation;
```

C++

```
__property IBindingOperation BindingOperation[int Index];
```

Description

This is BindingOperation, a member of class IBindingOperations.

189.2.5 IBindingOutput Interface

Class Hierarchy

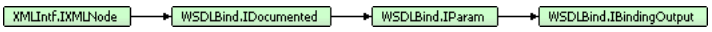

Pascal

```
IBindingOutput = interface(IParam);
```

C++

```
__interface IBindingOutput : public IParam;
```

File

WSDLBind

Description

This is class WSDLBind.IBindingOutput.

189.2.6 IBindings Interface

Class Hierarchy

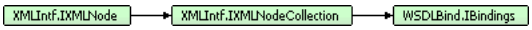

Pascal

```
IBindings = interface(IXMLNodeCollection);
```

C++

```
__interface IBindings : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IBindings.

189.2.6.1 IBindings Members

The following tables list the members exposed by IBindings.

Public Methods

	Name	Description
	Add (see page 25830)	This is Add, a member of class IBindings.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetBinding (see page 25830)	This is GetBinding, a member of class IBindings.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.

	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	Insert (see page 25830)	This is Insert, a member of class IBindings.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	Binding (see page 25831)	This is Binding, a member of class IBindings.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.6.2 IBindings Methods

The methods of the IBindings class are listed here.

Public Methods

	Name	Description
	Add (see page 25830)	This is Add, a member of class IBindings.
	GetBinding (see page 25830)	This is GetBinding, a member of class IBindings.
	Insert (see page 25830)	This is Insert, a member of class IBindings.

189.2.6.2.1 IBindings.Add Method

Pascal

```
function Add(const Name: WideString; const AType: WideString): IBinding;
```

C++

```
__fastcall IBinding Add(const BSTR Name, const BSTR AType);
```

Description

This is Add, a member of class IBindings.

189.2.6.2.2 IBindings.GetBinding Method

Pascal

```
function GetBinding(Index: Integer): IBinding;
```

C++

```
__fastcall IBinding GetBinding(int Index);
```

Description

This is GetBinding, a member of class IBindings.

189.2.6.2.3 IBindings.Insert Method

Pascal

```
function Insert(Index: Integer): IBinding;
```

C++

```
__fastcall IBinding Insert(int Index);
```

Description

This is Insert, a member of class IBindings.

189.2.6.3 IBindings Properties

The properties of the IBindings class are listed here.

Public Properties

	Name	Description
R	Binding (see page 25831)	This is Binding, a member of class IBindings.

189.2.6.3.1 IBindings.Binding Property

Pascal

```
property Binding [Index: Integer]: IBinding;
```

C++

```
__property IBinding Binding[int Index];
```

Description

This is Binding, a member of class IBindings.

189.2.7 IDefinition Interface

Class Hierarchy

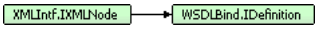

Pascal

```
IDefinition = interface(IXMLNode);
```

C++

```
__interface IDefinition : public IXMLNode;
```

File

WSDLBind

Description

This is class WSDLBind.IDefinition.

189.2.7.1 IDefinition Members

The following tables list the members exposed by IDefinition.

Public Methods

	Name	Description
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetBindings (see page 25833)	This is GetBindings, a member of class IDefinition.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetImports (see page 25833)	This is GetImports, a member of class IDefinition.

	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetMessages (see page 25834)	This is GetMessages, a member of class IDefinition.
	GetName (see page 25834)	This is GetName, a member of class IDefinition.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPortTypes (see page 25834)	This is GetPortTypes, a member of class IDefinition.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetServices (see page 25834)	This is GetServices, a member of class IDefinition.
	GetTargetNamespace (see page 25834)	This is GetTargetNamespace, a member of class IDefinition.
	GetText (see page 26423)	Returns the text value of the node.
	GetTypes (see page 25835)	This is GetTypes, a member of class IDefinition.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetName (see page 25835)	This is SetName, a member of class IDefinition.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetTargetNamespace (see page 25835)	This is SetTargetNamespace, a member of class IDefinition.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	Bindings (see page 25836)	This is Bindings, a member of class IDefinition.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	Imports (see page 25836)	This is Imports, a member of class IDefinition.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	Messages (see page 25836)	This is Messages, a member of class IDefinition.
	Name (see page 25836)	This is Name, a member of class IDefinition.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.

R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	PortTypes (see page 25836)	This is PortTypes, a member of class IDefinition.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	Services (see page 25837)	This is Services, a member of class IDefinition.
	TargetNamespace (see page 25837)	This is TargetNamespace, a member of class IDefinition.
	Text (see page 26439)	Represents the Text of the node.
R	Types (see page 25837)	This is Types, a member of class IDefinition.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.7.2 IDefinition Methods

The methods of the IDefinition class are listed here.

Public Methods

	Name	Description
	GetBindings (see page 25833)	This is GetBindings, a member of class IDefinition.
	GetImports (see page 25833)	This is GetImports, a member of class IDefinition.
	GetMessages (see page 25834)	This is GetMessages, a member of class IDefinition.
	GetName (see page 25834)	This is GetName, a member of class IDefinition.
	GetPortTypes (see page 25834)	This is GetPortTypes, a member of class IDefinition.
	GetServices (see page 25834)	This is GetServices, a member of class IDefinition.
	GetTargetNamespace (see page 25834)	This is GetTargetNamespace, a member of class IDefinition.
	GetTypes (see page 25835)	This is GetTypes, a member of class IDefinition.
	SetName (see page 25835)	This is SetName, a member of class IDefinition.
	SetTargetNamespace (see page 25835)	This is SetTargetNamespace, a member of class IDefinition.

189.2.7.2.1 IDefinition.GetBindings Method

Pascal

```
function GetBindings: IBindings;
```

C++

```
__fastcall IBindings GetBindings();
```

Description

This is GetBindings, a member of class IDefinition.

189.2.7.2.2 IDefinition.GetImports Method

Pascal

```
function GetImports: IImports;
```

C++

```
__fastcall IImports GetImports();
```

Description

This is GetImports, a member of class IDefinition.

189.2.7.2.3 IDefinition.GetMessages Method

Pascal

```
function GetMessages: IMessages;
```

C++

```
__fastcall IMessages GetMessages();
```

Description

This is GetMessages, a member of class IDefinition.

189.2.7.2.4 IDefinition.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IDefinition.

189.2.7.2.5 IDefinition.GetPortTypes Method

Pascal

```
function GetPortTypes: IPortTypes;
```

C++

```
__fastcall IPortTypes GetPortTypes();
```

Description

This is GetPortTypes, a member of class IDefinition.

189.2.7.2.6 IDefinition.GetServices Method

Pascal

```
function GetServices: IServices;
```

C++

```
__fastcall IServices GetServices();
```

Description

This is GetServices, a member of class IDefinition.

189.2.7.2.7 IDefinition.GetTargetNamespace Method

Pascal

```
function GetTargetNamespace: WideString;
```

C++

```
__fastcall BSTR GetTargetNamespace();
```

Description

This is GetTargetNamespace, a member of class IDefinition.

189.2.7.2.8 IDefinition.GetTypes Method

Pascal

```
function GetTypes: ITypes;
```

C++

```
__fastcall ITypes GetTypes();
```

Description

This is GetTypes, a member of class IDefinition.

189.2.7.2.9 IDefinition.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IDefinition.

189.2.7.2.10 IDefinition.SetTargetNamespace Method

Pascal

```
procedure SetTargetNamespace(Value: WideString);
```

C++

```
__fastcall SetTargetNamespace(BSTR Value);
```

Description

This is SetTargetNamespace, a member of class IDefinition.

189.2.7.3 IDefinition Properties

The properties of the IDefinition class are listed here.

Public Properties

	Name	Description
R	Bindings (see page 25836)	This is Bindings, a member of class IDefinition.
R	Imports (see page 25836)	This is Imports, a member of class IDefinition.
R	Messages (see page 25836)	This is Messages, a member of class IDefinition.
	Name (see page 25836)	This is Name, a member of class IDefinition.
R	PortTypes (see page 25836)	This is PortTypes, a member of class IDefinition.
R	Services (see page 25837)	This is Services, a member of class IDefinition.
	TargetNamespace (see page 25837)	This is TargetNamespace, a member of class IDefinition.
R	Types (see page 25837)	This is Types, a member of class IDefinition.

189.2.7.3.1 IDefinition.Bindings Property

Pascal

```
property Bindings: IBindings;
```

C++

```
__property IBindings Bindings;
```

Description

This is Bindings, a member of class IDefinition.

189.2.7.3.2 IDefinition.Imports Property

Pascal

```
property Imports: IImports;
```

C++

```
__property IImports Imports;
```

Description

This is Imports, a member of class IDefinition.

189.2.7.3.3 IDefinition.Messages Property

Pascal

```
property Messages: IMessages;
```

C++

```
__property IMessages Messages;
```

Description

This is Messages, a member of class IDefinition.

189.2.7.3.4 IDefinition.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IDefinition.

189.2.7.3.5 IDefinition.PortTypes Property

Pascal

```
property PortTypes: IPortTypes;
```

C++

```
__property IPortTypes PortTypes;
```

Description

This is PortTypes, a member of class IDefinition.

189.2.7.3.6 IDefinition.Services Property

Pascal

```
property Services: IServices;
```

C++

```
__property IServices Services;
```

Description

This is Services, a member of class IDefinition.

189.2.7.3.7 IDefinition.TargetNamespace Property

Pascal

```
property TargetNamespace: WideString;
```

C++

```
__property BSTR TargetNamespace;
```

Description

This is TargetNamespace, a member of class IDefinition.

189.2.7.3.8 IDefinition.Types Property

Pascal

```
property Types: ITypes;
```

C++

```
__property ITypes Types;
```

Description

This is Types, a member of class IDefinition.

189.2.8 IDocumentation Interface

Class Hierarchy

Pascal

```
IDocumentation = interface(IXMLNode);
```

C++

```
__interface IDocumentation : public IXMLNode;
```

File

WSDLBind

Description

This is class WSDLBind.IDocumentation.

189.2.9 IDocumentationCollection Interface

Class Hierarchy

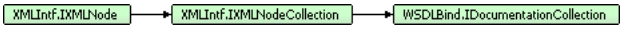

Pascal

```
IDocumentationCollection = interface(IXMLNodeCollection);
```

C++

```
__interface IDocumentationCollection : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IDocumentationCollection.

189.2.9.1 IDocumentationCollection Members

The following tables list the members exposed by IDocumentationCollection.

Public Methods

	Name	Description
	Add (see page 25840)	This is Add, a member of class IDocumentationCollection.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDocumentation (see page 25840)	This is GetDocumentation, a member of class IDocumentationCollection.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.

	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	Insert (see page 25840)	This is Insert, a member of class IDocumentationCollection.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	Documentation (see page 25841)	This is Documentation, a member of class IDocumentationCollection.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.9.2 IDocumentationCollection Methods

The methods of the IDocumentationCollection class are listed here.

Public Methods

	Name	Description
	Add (see page 25840)	This is Add, a member of class IDocumentationCollection.
	GetDocumentation (see page 25840)	This is GetDocumentation, a member of class IDocumentationCollection.
	Insert (see page 25840)	This is Insert, a member of class IDocumentationCollection.

189.2.9.2.1 IDocumentationCollection.Add Method

Pascal

```
function Add(const Text: WideString): IDocumentation;
```

C++

```
__fastcall IDocumentation Add(const BSTR Text);
```

Description

This is Add, a member of class IDocumentationCollection.

189.2.9.2.2 IDocumentationCollection.GetDocumentation Method

Pascal

```
function GetDocumentation(Index: Integer): IDocumentation;
```

C++

```
__fastcall IDocumentation GetDocumentation(int Index);
```

Description

This is GetDocumentation, a member of class IDocumentationCollection.

189.2.9.2.3 IDocumentationCollection.Insert Method

Pascal

```
function Insert(Index: Integer): IDocumentation;
```

C++

```
__fastcall IDocumentation Insert(int Index);
```

Description

This is Insert, a member of class IDocumentationCollection.

189.2.9.3 IDocumentationCollection Properties

The properties of the IDocumentationCollection class are listed here.

Public Properties

	Name	Description
R	Documentation (see page 25841)	This is Documentation, a member of class IDocumentationCollection.

189.2.9.3.1 IDocumentationCollection.Documentation Property

Pascal

```
property Documentation [Index: Integer]: IDocumentation;
```

C++

```
__property IDocumentation Documentation[int Index];
```

Description

This is Documentation, a member of class IDocumentationCollection.

189.2.10 IDocumented Interface

Class Hierarchy

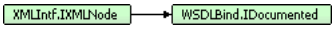

Pascal

```
IDocumented = interface(IXMLNode);
```

C++

```
__interface IDocumented : public IXMLNode;
```

File

WSDLBind

Description

This is class WSDLBind.IDocumented.

189.2.10.1 IDocumented Members

The following tables list the members exposed by IDocumented.

Public Methods

	Name	Description
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.

	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.10.2 IDocumented Methods

The methods of the IDocumented class are listed here.

Public Methods

	Name	Description
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.

189.2.10.2.1 IDocumented.GetDocumentationCollection Method

Pascal

```
function GetDocumentationCollection: IDocumentationCollection;
```

C++

```
__fastcall IDocumentationCollection GetDocumentationCollection();
```

Description

This is GetDocumentationCollection, a member of class IDocumented.

189.2.10.3 IDocumented Properties

The properties of the IDocumented class are listed here.

Public Properties

	Name	Description
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.

189.2.10.3.1 IDocumented.Documentation Property

Pascal

```
property Documentation: IDocumentationCollection;
```

C++

```
__property IDocumentationCollection Documentation;
```

Description

This is Documentation, a member of class IDocumented.

189.2.11 IFault Interface

Class Hierarchy

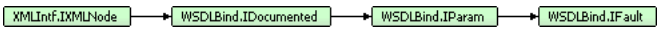

Pascal

```
IFault = interface(IParam);
```

C++

```
__interface IFault : public IParam;
```

File

WSDLBind

Description

This is class WSDLBind.IFault.

189.2.12 IFaults Interface

Class Hierarchy

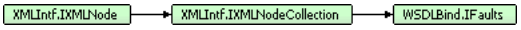

Pascal

```
IFaults = interface(IXMLNodeCollection);
```

C++

```
__interface IFaults : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IFaults.

189.2.12.1 IFaults Members

The following tables list the members exposed by IFaults.

Public Methods

	Name	Description
	Add (see page 25846)	This is Add, a member of class IFaults.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetFault (see page 25846)	This is GetFault, a member of class IFaults.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.

	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	Insert (see page 25846)	This is Insert, a member of class IFaults.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	Fault (see page 25847)	This is Fault, a member of class IFaults.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.12.2 IFaults Methods

The methods of the IFaults class are listed here.

Public Methods

	Name	Description
	Add (see page 25846)	This is Add, a member of class IFaults.
	GetFault (see page 25846)	This is GetFault, a member of class IFaults.
	Insert (see page 25846)	This is Insert, a member of class IFaults.

189.2.12.2.1 IFaults.Add Method

Pascal

```
function Add(const Name: WideString; const Message: WideString): IFault;
```

C++

```
__fastcall IFault Add(const BSTR Name, const BSTR Message);
```

Description

This is Add, a member of class IFaults.

189.2.12.2.2 IFaults.GetFault Method

Pascal

```
function GetFault(Index: Integer): IFault;
```

C++

```
__fastcall IFault GetFault(int Index);
```

Description

This is GetFault, a member of class IFaults.

189.2.12.2.3 IFaults.Insert Method

Pascal

```
function Insert(Index: Integer): IFault;
```

C++

```
__fastcall IFault Insert(int Index);
```

Description

This is Insert, a member of class IFaults.

189.2.12.3 IFaults Properties

The properties of the IFaults class are listed here.

Public Properties

	Name	Description
R	Fault (see page 25847)	This is Fault, a member of class IFaults.

189.2.12.3.1 IFaults.Fault Property

Pascal

```
property Fault [Index: Integer]: IFault;
```

C++

```
__property IFault Fault[int Index];
```

Description

This is Fault, a member of class IFaults.

189.2.13 IImport Interface

Class Hierarchy

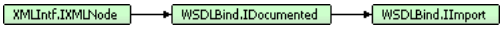

Pascal

```
IImport = interface(IDocumented);
```

C++

```
__interface IImport : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IImport.

189.2.13.1 IImport Members

The following tables list the members exposed by IImport.

Public Methods

	Name	Description
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.

	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetLocation (see page 25849)	This is GetLocation, a member of class IImport.
	GetNamespace (see page 25849)	This is GetNamespace, a member of class IImport.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetLocation (see page 25849)	This is SetLocation, a member of class IImport.
	SetNamespace (see page 25849)	This is SetNamespace, a member of class IImport.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Location (see page 25850)	This is Location, a member of class IImport.
	Namespace (see page 25850)	This is Namespace, a member of class IImport.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.13.2 IImport Methods

The methods of the IImport class are listed here.

Public Methods

	Name	Description
	GetLocation (see page 25849)	This is GetLocation, a member of class IImport.
	GetNamespace (see page 25849)	This is GetNamespace, a member of class IImport.
	SetLocation (see page 25849)	This is SetLocation, a member of class IImport.
	SetNamespace (see page 25849)	This is SetNamespace, a member of class IImport.

189.2.13.2.1 IImport.GetLocation Method

Pascal

```
function GetLocation: WideString;
```

C++

```
__fastcall BSTR GetLocation();
```

Description

This is GetLocation, a member of class IImport.

189.2.13.2.2 IImport.GetNamespace Method

Pascal

```
function GetNamespace: WideString;
```

C++

```
__fastcall BSTR GetNamespace();
```

Description

This is GetNamespace, a member of class IImport.

189.2.13.2.3 IImport.SetLocation Method

Pascal

```
procedure SetLocation(Value: WideString);
```

C++

```
__fastcall SetLocation(BSTR Value);
```

Description

This is SetLocation, a member of class IImport.

189.2.13.2.4 IImport.SetNamespace Method

Pascal

```
procedure SetNamespace(Value: WideString);
```

C++

```
__fastcall SetNamespace(BSTR Value);
```

Description

This is SetNamespace, a member of class IImport.

189.2.13.3 IImport Properties

The properties of the IImport class are listed here.

Public Properties

	Name	Description
	Location (see page 25850)	This is Location, a member of class IImport.
	Namespace (see page 25850)	This is Namespace, a member of class IImport.

189.2.13.3.1 IImport.Location Property

Pascal

```
property Location: WideString;
```

C++

```
__property BSTR Location;
```

Description

This is Location, a member of class IImport.

189.2.13.3.2 IImport.Namespace Property

Pascal

```
property Namespace: WideString;
```

C++

```
__property BSTR Namespace;
```

Description

This is Namespace, a member of class IImport.

189.2.14 IImports Interface

Class Hierarchy

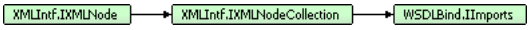

Pascal

```
IImports = interface(IXMLNodeCollection);
```

C++

```
__interface IImports : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IImports.

189.2.14.1 IImports Members

The following tables list the members exposed by IImports.

Public Methods

	Name	Description
	Add (see page 25852)	This is Add, a member of class IImports.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetImport (see page 25853)	This is GetImport, a member of class IImports.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	Insert (see page 25853)	This is Insert, a member of class IImports.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	Import (see page 25853)	This is Import, a member of class IImports.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.14.2 IImports Methods

The methods of the IImports class are listed here.

Public Methods

	Name	Description
	Add (see page 25852)	This is Add, a member of class IImports.
	GetImport (see page 25853)	This is GetImport, a member of class IImports.
	Insert (see page 25853)	This is Insert, a member of class IImports.

189.2.14.2.1 IImports.Add Method

Pascal

```
function Add(const Namespace: WideString; const Location: WideString): IImport;
```

C++

```
__fastcall IImport Add(const BSTR Namespace, const BSTR Location);
```

Description

This is Add, a member of class IImports.

189.2.14.2.2 IImports.GetImport Method

Pascal

```
function GetImport(Index: Integer): IImport;
```

C++

```
__fastcall IImport GetImport(int Index);
```

Description

This is GetImport, a member of class IImports.

189.2.14.2.3 IImports.Insert Method

Pascal

```
function Insert(Index: Integer): IImport;
```

C++

```
__fastcall IImport Insert(int Index);
```

Description

This is Insert, a member of class IImports.

189.2.14.3 IImports Properties

The properties of the IImports class are listed here.

Public Properties

	Name	Description
	Import (see page 25853)	This is Import, a member of class IImports.

189.2.14.3.1 IImports.Import Property

Pascal

```
property Import [Index: Integer]: IImport;
```

C++

```
__property IImport Import[int Index];
```

Description

This is Import, a member of class IImports.

189.2.15 IMessage Interface

Class Hierarchy

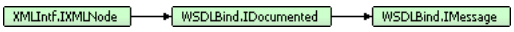

Pascal

```
IMessage = interface(IDocumented);
```

C++

```
__interface IMessage : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IMessage.

189.2.15.1 IMessage Members

The following tables list the members exposed by IMessage.

Public Methods

	Name	Description
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetName (see page 25855)	This is GetName, a member of class IMessage.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetParts (see page 25856)	This is GetParts, a member of class IMessage.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetName (see page 25856)	This is SetName, a member of class IMessage.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Name (see page 25856)	This is Name, a member of class IMessage.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Parts (see page 25856)	This is Parts, a member of class IMessage.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.15.2 IMessage Methods

The methods of the IMessage class are listed here.

Public Methods

	Name	Description
	GetName (see page 25855)	This is GetName, a member of class IMessage.
	GetParts (see page 25856)	This is GetParts, a member of class IMessage.
	SetName (see page 25856)	This is SetName, a member of class IMessage.

189.2.15.2.1 IMessage.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IMessage.

189.2.15.2.2 IMessage.GetParts Method

Pascal

```
function GetParts: IParts;
```

C++

```
__fastcall IParts GetParts();
```

Description

This is GetParts, a member of class IMessage.

189.2.15.2.3 IMessage.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IMessage.

189.2.15.3 IMessage Properties

The properties of the IMessage class are listed here.

Public Properties

	Name	Description
	Name (see page 25856)	This is Name, a member of class IMessage.
R	Parts (see page 25856)	This is Parts, a member of class IMessage.

189.2.15.3.1 IMessage.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IMessage.

189.2.15.3.2 IMessage.Parts Property

Pascal

```
property Parts: IParts;
```

C++

```
__property IParts Parts;
```

Description

This is Parts, a member of class IMessage.

189.2.16 IMessages Interface

Class Hierarchy

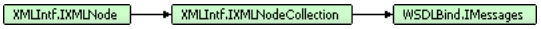

Pascal

```
IMessages = interface(IXMLNodeCollection);
```

C++

```
__interface IMessages : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IMessages.

189.2.16.1 IMessages Members

The following tables list the members exposed by IMessages.

Public Methods

	Name	Description
	Add (see page 25859)	This is Add, a member of class IMessages.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetMessage (see page 25859)	This is GetMessage, a member of class IMessages.

	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	Insert (see page 25859)	This is Insert, a member of class IMessages.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	Message (see page 25860)	This is Message, a member of class IMessages.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.16.2 IMessages Methods

The methods of the IMessages class are listed here.

Public Methods

	Name	Description
	Add (see page 25859)	This is Add, a member of class IMessages.
	GetMessage (see page 25859)	This is GetMessage, a member of class IMessages.
	Insert (see page 25859)	This is Insert, a member of class IMessages.

189.2.16.2.1 IMessages.Add Method

Pascal

```
function Add(const Name: WideString): IMessage;
```

C++

```
__fastcall IMessage Add(const BSTR Name);
```

Description

This is Add, a member of class IMessages.

189.2.16.2.2 IMessages.GetMessage Method

Pascal

```
function GetMessage(Index: Integer): IMessage;
```

C++

```
__fastcall IMessage GetMessage(int Index);
```

Description

This is GetMessage, a member of class IMessages.

189.2.16.2.3 IMessages.Insert Method

Pascal

```
function Insert(Index: Integer): IMessage;
```

C++

```
__fastcall IMessage Insert(int Index);
```

Description

This is Insert, a member of class IMessages.

189.2.16.3 IMessages Properties

The properties of the IMessages class are listed here.

Public Properties

	Name	Description
R	Message (see page 25860)	This is Message, a member of class IMessages.

189.2.16.3.1 IMessages.Message Property

Pascal

```
property Message [Index: Integer]: IMessage;
```

C++

```
__property IMessage Message[int Index];
```

Description

This is Message, a member of class IMessages.

189.2.17 IOperation Interface

Class Hierarchy

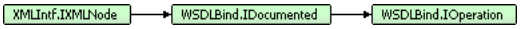

Pascal

```
IOperation = interface(IDocumented);
```

C++

```
__interface IOperation : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IOperation.

189.2.17.1 IOperation Members

The following tables list the members exposed by IOperation.

Public Methods

	Name	Description
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetFaults (see page 25862)	This is GetFaults, a member of class IOperation.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.

	GetInput (see page 25862)	This is GetInput, a member of class IOperation.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetName (see page 25862)	This is GetName, a member of class IOperation.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOutput (see page 25863)	This is GetOutput, a member of class IOperation.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetName (see page 25863)	This is SetName, a member of class IOperation.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	Faults (see page 25863)	This is Faults, a member of class IOperation.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	Input (see page 25863)	This is Input, a member of class IOperation.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Name (see page 25864)	This is Name, a member of class IOperation.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	Output (see page 25864)	This is Output, a member of class IOperation.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.

	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.17.2 IOperation Methods

The methods of the IOperation class are listed here.

Public Methods

	Name	Description
	GetFaults (see page 25862)	This is GetFaults, a member of class IOperation.
	GetInput (see page 25862)	This is GetInput, a member of class IOperation.
	GetName (see page 25862)	This is GetName, a member of class IOperation.
	GetOutput (see page 25863)	This is GetOutput, a member of class IOperation.
	SetName (see page 25863)	This is SetName, a member of class IOperation.

189.2.17.2.1 IOperation.GetFaults Method

Pascal

```
function GetFaults: IFaults;
```

C++

```
__fastcall IFaults GetFaults();
```

Description

This is GetFaults, a member of class IOperation.

189.2.17.2.2 IOperation.GetInput Method

Pascal

```
function GetInput: IParam;
```

C++

```
__fastcall IParam GetInput();
```

Description

This is GetInput, a member of class IOperation.

189.2.17.2.3 IOperation.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IOperation.

189.2.17.2.4 IOperation.GetOutput Method

Pascal

```
function GetOutput: IParam;
```

C++

```
__fastcall IParam GetOutput();
```

Description

This is GetOutput, a member of class IOperation.

189.2.17.2.5 IOperation.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IOperation.

189.2.17.3 IOperation Properties

The properties of the IOperation class are listed here.

Public Properties

	Name	Description
R	Faults (see page 25863)	This is Faults, a member of class IOperation.
R	Input (see page 25863)	This is Input, a member of class IOperation.
	Name (see page 25864)	This is Name, a member of class IOperation.
R	Output (see page 25864)	This is Output, a member of class IOperation.

189.2.17.3.1 IOperation.Faults Property

Pascal

```
property Faults: IFaults;
```

C++

```
__property IFaults Faults;
```

Description

This is Faults, a member of class IOperation.

189.2.17.3.2 IOperation.Input Property

Pascal

```
property Input: IParam;
```

C++

```
__property IParam Input;
```

Description

This is Input, a member of class IOperation.

189.2.17.3.3 IOperation.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IOperation.

189.2.17.3.4 IOperation.Output Property

Pascal

```
property Output: IParam;
```

C++

```
__property IParam Output;
```

Description

This is Output, a member of class IOperation.

189.2.18 IOperations Interface

Class Hierarchy

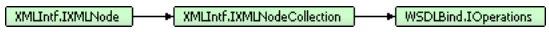

Pascal

```
IOperations = interface(IXMLNodeCollection);
```

C++

```
__interface IOperations : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IOperations.

189.2.18.1 IOperations Members

The following tables list the members exposed by IOperations.

Public Methods

	Name	Description
	Add (see page 25866)	This is the overview for the Add method overload.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOperation (see page 25867)	This is GetOperation, a member of class IOperations.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	Insert (see page 25867)	This is Insert, a member of class IOperations.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.

	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	Operation (see page 25867)	This is Operation, a member of class IOperations.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.18.2 IOperations Methods

The methods of the IOperations class are listed here.

Public Methods

	Name	Description
	Add (see page 25866)	This is the overview for the Add method overload.
	GetOperation (see page 25867)	This is GetOperation, a member of class IOperations.
	Insert (see page 25867)	This is Insert, a member of class IOperations.

189.2.18.2.1 Add Method

This is the overview for the Add method overload.

Overload List

	Name	Description
	IOperations.Add (WideString) (see page 25866)	This is Add, a member of class IOperations.
	IOperations.Add (WideString, WideString, WideString, WideString) (see page 25867)	This is Add, a member of class IOperations.

189.2.18.2.1.1 IOperations.Add Method (WideString)

Pascal

```
function Add(const Name: WideString): IOperation; overload;
```

C++

```
__fastcall IOperation Add(const BSTR Name);
```

Description

This is Add, a member of class IOperations.

189.2.18.2.1.2 IOperations.Add Method (WideString, WideString, WideString, WideString)

Pascal

```
function Add(const Name: WideString; const InputMessage: WideString; const InputName:
WideString; const OutputMessage: WideString): IOperation; overload;
```

C++

```
__fastcall IOperation Add(const BSTR Name, const BSTR InputMessage, const BSTR InputName,
const BSTR OutputMessage);
```

Description

This is Add, a member of class IOperations.

189.2.18.2.2 IOperations.GetOperation Method

Pascal

```
function GetOperation(Index: Integer): IOperation;
```

C++

```
__fastcall IOperation GetOperation(int Index);
```

Description

This is GetOperation, a member of class IOperations.

189.2.18.2.3 IOperations.Insert Method

Pascal

```
function Insert(Index: Integer): IOperation;
```

C++

```
__fastcall IOperation Insert(int Index);
```

Description

This is Insert, a member of class IOperations.

189.2.18.3 IOperations Properties

The properties of the IOperations class are listed here.

Public Properties

	Name	Description
R	Operation (see page 25867)	This is Operation, a member of class IOperations.

189.2.18.3.1 IOperations.Operation Property

Pascal

```
property Operation [Index: Integer]: IOperation;
```

C++

```
__property IOperation Operation[int Index];
```

Description

This is Operation, a member of class IOperations.

189.2.19 IParam Interface

Class Hierarchy

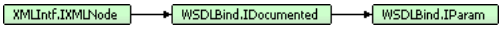

Pascal

```
IParam = interface(IDocumented);
```

C++

```
__interface IParam : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IParam.

189.2.19.1 IParam Members

The following tables list the members exposed by IParam.

Public Methods

	Name	Description
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetMessage (see page 25870)	This is GetMessage, a member of class IParam.
	GetName (see page 25870)	This is GetName, a member of class IParam.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.

	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetMessage (see page 25870)	This is SetMessage, a member of class IParam.
	SetName (see page 25870)	This is SetName, a member of class IParam.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Message (see page 25871)	This is Message, a member of class IParam.
	Name (see page 25871)	This is Name, a member of class IParam.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.19.2 IParam Methods

The methods of the IParam class are listed here.

Public Methods

	Name	Description
	GetMessage (see page 25870)	This is GetMessage, a member of class IParam.
	GetName (see page 25870)	This is GetName, a member of class IParam.
	SetMessage (see page 25870)	This is SetMessage, a member of class IParam.
	SetName (see page 25870)	This is SetName, a member of class IParam.

189.2.19.2.1 IParam.GetMessage Method

Pascal

```
function GetMessage: WideString;
```

C++

```
__fastcall BSTR GetMessage();
```

Description

This is GetMessage, a member of class IParam.

189.2.19.2.2 IParam.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IParam.

189.2.19.2.3 IParam.SetMessage Method

Pascal

```
procedure SetMessage(Value: WideString);
```

C++

```
__fastcall SetMessage(BSTR Value);
```

Description

This is SetMessage, a member of class IParam.

189.2.19.2.4 IParam.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IParam.

189.2.19.3 IParam Properties

The properties of the IParam class are listed here.

Public Properties

	Name	Description
	Message (see page 25871)	This is Message, a member of class IParam.
	Name (see page 25871)	This is Name, a member of class IParam.

189.2.19.3.1 IParam.Message Property

Pascal

```
property Message: WideString;
```

C++

```
__property BSTR Message;
```

Description

This is Message, a member of class IParam.

189.2.19.3.2 IParam.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IParam.

189.2.20 IPart Interface

Class Hierarchy

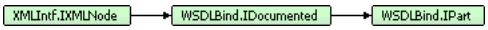

Pascal

```
IPart = interface(IDocumented);
```

C++

```
__interface IPart : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IPart.

189.2.20.1 IPart Members

The following tables list the members exposed by IPart.

Public Methods

	Name	Description
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetElement (see page 25873)	This is GetElement, a member of class IPart.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetName (see page 25874)	This is GetName, a member of class IPart.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetType_ (see page 25874)	This is GetType_, a member of class IPart.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetElement (see page 25874)	This is SetElement, a member of class IPart.
	SetName (see page 25874)	This is SetName, a member of class IPart.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	SetType_ (see page 25874)	This is SetType_, a member of class IPart.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
	Element (see page 25875)	This is Element, a member of class IPart.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Name (see page 25875)	This is Name, a member of class IPart.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
	Type_ (see page 25875)	This is Type_, a member of class IPart.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.20.2 IPart Methods

The methods of the IPart class are listed here.

Public Methods

	Name	Description
	GetElement (see page 25873)	This is GetElement, a member of class IPart.
	GetName (see page 25874)	This is GetName, a member of class IPart.
	GetType_ (see page 25874)	This is GetType_, a member of class IPart.
	SetElement (see page 25874)	This is SetElement, a member of class IPart.
	SetName (see page 25874)	This is SetName, a member of class IPart.
	SetType_ (see page 25874)	This is SetType_, a member of class IPart.

189.2.20.2.1 IPart.GetElement Method

Pascal

```
function GetElement: WideString;
```

C++

```
__fastcall BSTR GetElement();
```

Description

This is GetElement, a member of class IPart.

189.2.20.2.2 IPart.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IPart.

189.2.20.2.3 IPart.GetType_ Method

Pascal

```
function GetType_: WideString;
```

C++

```
__fastcall BSTR GetType_();
```

Description

This is GetType_, a member of class IPart.

189.2.20.2.4 IPart.SetElement Method

Pascal

```
procedure SetElement(Value: WideString);
```

C++

```
__fastcall SetElement(BSTR Value);
```

Description

This is SetElement, a member of class IPart.

189.2.20.2.5 IPart.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IPart.

189.2.20.2.6 IPart.SetType_ Method

Pascal

```
procedure SetType_(Value: WideString);
```

C++

```
__fastcall SetType_(BSTR Value);
```

Description

This is SetType_, a member of class IPart.

189.2.20.3 IPart Properties

The properties of the IPart class are listed here.

Public Properties

	Name	Description
	Element (see page 25875)	This is Element, a member of class IPart.
	Name (see page 25875)	This is Name, a member of class IPart.
	Type_ (see page 25875)	This is Type_, a member of class IPart.

189.2.20.3.1 IPart.Element Property

Pascal

```
property Element: WideString;
```

C++

```
__property BSTR Element;
```

Description

This is Element, a member of class IPart.

189.2.20.3.2 IPart.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IPart.

189.2.20.3.3 IPart.Type_ Property

Pascal

```
property Type_: WideString;
```

C++

```
__property BSTR Type_;
```

Description

This is Type_, a member of class IPart.

189.2.21 IParts Interface

Class Hierarchy

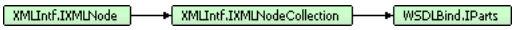

Pascal

```
IParts = interface(IXMLNodeCollection);
```

C++

```
__interface IParts : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IParts.

189.2.21.1 IParts Members

The following tables list the members exposed by IParts.

Public Methods

	Name	Description
	Add (see page 25878)	This is Add, a member of class IParts.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.

	GetPart (see page 25878)	This is GetPart, a member of class IParts.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	Insert (see page 25878)	This is Insert, a member of class IParts.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Part (see page 25879)	This is Part, a member of class IParts.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.21.2 IParts Methods

The methods of the IParts class are listed here.

Public Methods

	Name	Description
	Add (see page 25878)	This is Add, a member of class IParts.
	GetPart (see page 25878)	This is GetPart, a member of class IParts.
	Insert (see page 25878)	This is Insert, a member of class IParts.

189.2.21.2.1 IParts.Add Method

Pascal

```
function Add(const Name: WideString; const Element: WideString; const AType: WideString = ''):
IPart;
```

C++

```
__fastcall IPart Add(const BSTR Name, const BSTR Element, const BSTR AType = '');
```

Description

This is Add, a member of class IParts.

189.2.21.2.2 IParts.GetPart Method

Pascal

```
function GetPart(Index: Integer): IPart;
```

C++

```
__fastcall IPart GetPart(int Index);
```

Description

This is GetPart, a member of class IParts.

189.2.21.2.3 IParts.Insert Method

Pascal

```
function Insert(Index: Integer): IPart;
```

C++

```
__fastcall IPart Insert(int Index);
```

Description

This is Insert, a member of class IParts.

189.2.21.3 IParts Properties

The properties of the IParts class are listed here.

Public Properties

	Name	Description
R	Part (see page 25879)	This is Part, a member of class IParts.

189.2.21.3.1 IParts.Part Property

Pascal

```
property Part [Index: Integer]: IPart;
```

C++

```
__property IPart Part[int Index];
```

Description

This is Part, a member of class IParts.

189.2.22 IPort Interface

Class Hierarchy

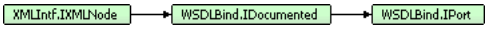

Pascal

```
IPort = interface(IDocumented);
```

C++

```
__interface IPort : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IPort.

189.2.22.1 IPort Members

The following tables list the members exposed by IPort.

Public Methods

	Name	Description
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetBinding (see page 25881)	This is GetBinding, a member of class IPort.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.

	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetName (see page 25881)	This is GetName, a member of class IPort.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetBinding (see page 25881)	This is SetBinding, a member of class IPort.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetName (see page 25881)	This is SetName, a member of class IPort.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
	Binding (see page 25882)	This is Binding, a member of class IPort.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Name (see page 25882)	This is Name, a member of class IPort.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.22.2 IPort Methods

The methods of the IPort class are listed here.

Public Methods

	Name	Description
	GetBinding (see page 25881)	This is GetBinding, a member of class IPort.
	GetName (see page 25881)	This is GetName, a member of class IPort.
	SetBinding (see page 25881)	This is SetBinding, a member of class IPort.
	SetName (see page 25881)	This is SetName, a member of class IPort.

189.2.22.2.1 IPort.GetBinding Method

Pascal

```
function GetBinding: WideString;
```

C++

```
__fastcall BSTR GetBinding();
```

Description

This is GetBinding, a member of class IPort.

189.2.22.2.2 IPort.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IPort.

189.2.22.2.3 IPort.SetBinding Method

Pascal

```
procedure SetBinding(Value: WideString);
```

C++

```
__fastcall SetBinding(BSTR Value);
```

Description

This is SetBinding, a member of class IPort.

189.2.22.2.4 IPort.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IPort.

189.2.22.3 IPort Properties

The properties of the IPort class are listed here.

Public Properties

	Name	Description
	Binding (see page 25882)	This is Binding, a member of class IPort.
	Name (see page 25882)	This is Name, a member of class IPort.

189.2.22.3.1 IPort.Binding Property

Pascal

```
property Binding: WideString;
```

C++

```
__property BSTR Binding;
```

Description

This is Binding, a member of class IPort.

189.2.22.3.2 IPort.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IPort.

189.2.23 IPorts Interface

Class Hierarchy

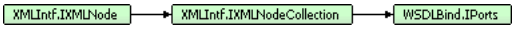

Pascal

```
IPorts = interface(IXMLNodeCollection);
```

C++

```
__interface IPorts : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IPorts.

189.2.23.1 IPorts Members

The following tables list the members exposed by IPorts.

Public Methods

	Name	Description
	Add (see page 25884)	This is Add, a member of class IPorts.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPort (see page 25885)	This is GetPort, a member of class IPorts.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	Insert (see page 25885)	This is Insert, a member of class IPorts.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Port (see page 25885)	This is Port, a member of class IPorts.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.23.2 IPorts Methods

The methods of the IPorts class are listed here.

Public Methods

	Name	Description
	Add (see page 25884)	This is Add, a member of class IPorts.
	GetPort (see page 25885)	This is GetPort, a member of class IPorts.
	Insert (see page 25885)	This is Insert, a member of class IPorts.

189.2.23.2.1 IPorts.Add Method

Pascal

```
function Add(const Name: WideString; const ABinding: WideString): IPort;
```

C++

```
__fastcall IPort Add(const BSTR Name, const BSTR ABinding);
```

Description

This is Add, a member of class IPorts.

189.2.23.2.2 IPorts.GetPort Method

Pascal

```
function GetPort(Index: Integer): IPort;
```

C++

```
__fastcall IPort GetPort(int Index);
```

Description

This is GetPort, a member of class IPorts.

189.2.23.2.3 IPorts.Insert Method

Pascal

```
function Insert(Index: Integer): IPort;
```

C++

```
__fastcall IPort Insert(int Index);
```

Description

This is Insert, a member of class IPorts.

189.2.23.3 IPorts Properties

The properties of the IPorts class are listed here.

Public Properties

	Name	Description
R	Port (see page 25885)	This is Port, a member of class IPorts.

189.2.23.3.1 IPorts.Port Property

Pascal

```
property Port [Index: Integer]: IPort;
```

C++

```
__property IPort Port[int Index];
```

Description

This is Port, a member of class IPorts.

189.2.24 IPortType Interface

Class Hierarchy

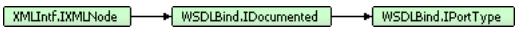

Pascal

```
IPortType = interface(IDocumented);
```

C++

```
__interface IPortType : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IPortType.

189.2.24.1 IPortType Members

The following tables list the members exposed by IPortType.

Public Methods

	Name	Description
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetName (see page 25887)	This is GetName, a member of class IPortType.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOperations (see page 25888)	This is GetOperations, a member of class IPortType.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetName (see page 25888)	This is SetName, a member of class IPortType.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Name (see page 25888)	This is Name, a member of class IPortType.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	Operations (see page 25888)	This is Operations, a member of class IPortType.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.24.2 IPortType Methods

The methods of the IPortType class are listed here.

Public Methods

	Name	Description
	GetName (see page 25887)	This is GetName, a member of class IPortType.
	GetOperations (see page 25888)	This is GetOperations, a member of class IPortType.
	SetName (see page 25888)	This is SetName, a member of class IPortType.

189.2.24.2.1 IPortType.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IPortType.

189.2.24.2.2 IPortType.GetOperations Method

Pascal

```
function GetOperations: IOperations;
```

C++

```
__fastcall IOperations GetOperations();
```

Description

This is GetOperations, a member of class IPortType.

189.2.24.2.3 IPortType.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IPortType.

189.2.24.3 IPortType Properties

The properties of the IPortType class are listed here.

Public Properties

	Name	Description
	Name (see page 25888)	This is Name, a member of class IPortType.
R	Operations (see page 25888)	This is Operations, a member of class IPortType.

189.2.24.3.1 IPortType.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IPortType.

189.2.24.3.2 IPortType.Operations Property

Pascal

```
property Operations: IOperations;
```

C++

```
__property IOperations Operations;
```

Description

This is Operations, a member of class IPortType.

189.2.25 IPortTypes Interface

Class Hierarchy

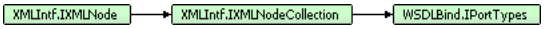

Pascal

```
IPortTypes = interface(IXMLNodeCollection);
```

C++

```
__interface IPortTypes : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IPortTypes.

189.2.25.1 IPortTypes Members

The following tables list the members exposed by IPortTypes.

Public Methods

	Name	Description
	Add (see page 25891)	This is Add, a member of class IPortTypes.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.

	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPortType (see page 25891)	This is GetPortType, a member of class IPortTypes.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	Insert (see page 25891)	This is Insert, a member of class IPortTypes.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	PortType (see page 25892)	This is PortType, a member of class IPortTypes.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.25.2 IPortTypes Methods

The methods of the IPortTypes class are listed here.

Public Methods

	Name	Description
	Add (see page 25891)	This is Add, a member of class IPortTypes.
	GetPortType (see page 25891)	This is GetPortType, a member of class IPortTypes.
	Insert (see page 25891)	This is Insert, a member of class IPortTypes.

189.2.25.2.1 IPortTypes.Add Method

Pascal

```
function Add(const Name: WideString): IPortType;
```

C++

```
__fastcall IPortType Add(const BSTR Name);
```

Description

This is Add, a member of class IPortTypes.

189.2.25.2.2 IPortTypes.GetPortType Method

Pascal

```
function GetPortType(Index: Integer): IPortType;
```

C++

```
__fastcall IPortType GetPortType(int Index);
```

Description

This is GetPortType, a member of class IPortTypes.

189.2.25.2.3 IPortTypes.Insert Method

Pascal

```
function Insert(Index: Integer): IPortType;
```

C++

```
__fastcall IPortType Insert(int Index);
```

Description

This is Insert, a member of class IPortTypes.

189.2.25.3 IPortTypes Properties

The properties of the IPortTypes class are listed here.

Public Properties

	Name	Description
R	PortType (see page 25892)	This is PortType, a member of class IPortTypes.

189.2.25.3.1 IPortTypes.PortType Property

Pascal

```
property PortType [Index: Integer]: IPortType;
```

C++

```
__property IPortType PortType[int Index];
```

Description

This is PortType, a member of class IPortTypes.

189.2.26 IService Interface

Class Hierarchy

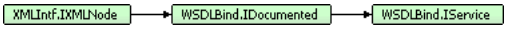

Pascal

```
IService = interface(IDocumented);
```

C++

```
__interface IService : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IService.

189.2.26.1 IService Members

The following tables list the members exposed by IService.

Public Methods

	Name	Description
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.

	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetName (see page 25894)	This is GetName, a member of class IService.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPorts (see page 25894)	This is GetPorts, a member of class IService.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetName (see page 25894)	This is SetName, a member of class IService.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
	Name (see page 25895)	This is Name, a member of class IService.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Ports (see page 25895)	This is Ports, a member of class IService.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.26.2 IService Methods

The methods of the IService class are listed here.

Public Methods

	Name	Description
	GetName (see page 25894)	This is GetName, a member of class IService.
	GetPorts (see page 25894)	This is GetPorts, a member of class IService.
	SetName (see page 25894)	This is SetName, a member of class IService.

189.2.26.2.1 IService.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IService.

189.2.26.2.2 IService.GetPorts Method

Pascal

```
function GetPorts: IPorts;
```

C++

```
__fastcall IPorts GetPorts();
```

Description

This is GetPorts, a member of class IService.

189.2.26.2.3 IService.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IService.

189.2.26.3 IService Properties

The properties of the IService class are listed here.

Public Properties

	Name	Description
	Name (see page 25895)	This is Name, a member of class IService.

	Ports (see page 25895)	This is Ports, a member of class IService.

189.2.26.3.1 IService.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IService.

189.2.26.3.2 IService.Ports Property

Pascal

```
property Ports: IPorts;
```

C++

```
__property IPorts Ports;
```

Description

This is Ports, a member of class IService.

189.2.27 IServices Interface

Class Hierarchy

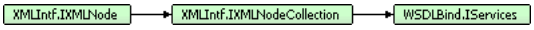

Pascal

```
IServices = interface(IXMLNodeCollection);
```

C++

```
__interface IServices : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IServices.

189.2.27.1 IServices Members

The following tables list the members exposed by IServices.

Public Methods

	Name	Description
	Add (see page 25897)	This is Add, a member of class IServices.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.

	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetService (see page 25897)	This is GetService, a member of class IServices.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	Insert (see page 25898)	This is Insert, a member of class IServices.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.

R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	Service (see page 25898)	This is Service, a member of class IServices.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.27.2 IServices Methods

The methods of the IServices class are listed here.

Public Methods

	Name	Description
	Add (see page 25897)	This is Add, a member of class IServices.
	GetService (see page 25897)	This is GetService, a member of class IServices.
	Insert (see page 25898)	This is Insert, a member of class IServices.

189.2.27.2.1 IServices.Add Method

Pascal

```
function Add(const Name: WideString): IService;
```

C++

```
__fastcall IService Add(const BSTR Name);
```

Description

This is Add, a member of class IServices.

189.2.27.2.2 IServices.GetService Method

Pascal

```
function GetService(Index: Integer): IService;
```

C++

```
__fastcall IService GetService(int Index);
```

Description

This is GetService, a member of class IServices.

189.2.27.2.3 IServices.Insert Method

Pascal

```
function Insert(Index: Integer): IService;
```

C++

```
__fastcall IService Insert(int Index);
```

Description

This is Insert, a member of class IServices.

189.2.27.3 IServices Properties

The properties of the IServices class are listed here.

Public Properties

	Name	Description
	Service (see page 25898)	This is Service, a member of class IServices.

189.2.27.3.1 IServices.Service Property

Pascal

```
property Service [Index: Integer]: IService;
```

C++

```
__property IService Service[int Index];
```

Description

This is Service, a member of class IServices.

189.2.28 ITypes Interface

Class Hierarchy

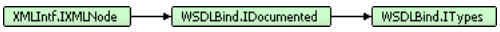

Pascal

```
ITypes = interface(IDocumented);
```

C++

```
__interface ITypes : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.ITypes.

189.2.28.1 ITypes Members

The following tables list the members exposed by ITypes.

Public Methods

	Name	Description
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDocumentationCollection (see page 25843)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetSchemaDefs (see page 25900)	This is GetSchemaDefs, a member of class ITypes.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.

	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Documentation (see page 25843)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	SchemaDefs (see page 25900)	This is SchemaDefs, a member of class ITypes.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.28.2 ITypes Methods

The methods of the ITypes class are listed here.

Public Methods

	Name	Description
	GetSchemaDefs (see page 25900)	This is GetSchemaDefs, a member of class ITypes.

189.2.28.2.1 ITypes.GetSchemaDefs Method

Pascal

```
function GetSchemaDefs: IXMLSchemaDefs;
```

C++

```
__fastcall IXMLSchemaDefs GetSchemaDefs();
```

Description

This is GetSchemaDefs, a member of class ITypes.

189.2.28.3 ITypes Properties

The properties of the ITypes class are listed here.

Public Properties

	Name	Description
R	SchemaDefs (see page 25900)	This is SchemaDefs, a member of class ITypes.

189.2.28.3.1 ITypes.SchemaDefs Property

Pascal

```
property SchemaDefs: IXMLSchemaDefs;
```

C++

```
__property IXMLSchemaDefs SchemaDefs;
```

Description

This is SchemaDefs, a member of class ITypes.

189.2.29 IWSDLDocument Interface

IWSDLDocument provides low-level access to the definitions in a WSDL document.

Class Hierarchy

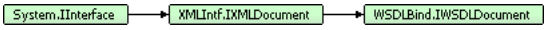

Pascal

```
IWSDLDocument = interface(IXMLDocument);
```

C++

```
__interface IWSDLDocument : public IXMLDocument;
```

File

WSDLBind

Description

IWSDLDocument is an interface to the TWSDLDocument (see page 25799) class. It is primarily for internal use, because it only provides access to the Definitions property, which is used for low-level access to the definitions in a WSDL document. It is generally easier to work with a TWSDLItems object. You can often obtain this from an IWSDLDocument interface by using the global Supports function (Delphi) or the Supports method (C++) to obtain an IWSDLItems interface from this one, and then obtaining a TWSDLItems object from that interface.

Note: C++ method declarations that use IWSDLDocument use the _di_IWSDLDocument type instead. This type is a DelphiInterface wrapper around the IWSDLDocument interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IWSDLDocument >  _di_IWSDLDocument;
```

See Also

TWSDLDocument (see page 25799)

IWSDLItems (see page 25998)

TWSDLItems (see page 25951)

189.2.29.1 IWSDLDocument Members

The following tables list the members exposed by IWSDLDocument.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26385)	This is the overview for the AddChild method overload.

	CreateElement (see page 26386)	Creates a new element node that is associated with this document.
	CreateNode (see page 26387)	Creates a new node that is associated with this document.
	GeneratePrefix (see page 26388)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 26388)	Returns the value of the Active (see page 26403) property.
	GetAsyncLoadState (see page 26389)	Returns the value of the AsyncLoadState (see page 26404) property.
	GetChildNodes (see page 26389)	Returns the value of the ChildNodes (see page 26405) property.
	GetDefinition (see page 25903)	Returns the value of the Definition (see page 25904) property.
	GetDocBinding (see page 26389)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 26390)	Returns the value of the DocumentElement (see page 26405) property.
	GetDocumentNode (see page 26390)	Returns the value of the Node (see page 26407) property.
	GetDOMDocument (see page 26390)	Returns the value of the DOMDocument (see page 26405) property.
	GetEncoding (see page 26391)	Returns the value of the Encoding (see page 26406) property.
	GetFileName (see page 26391)	Returns the value of the FileName (see page 26406) property.
	GetModified (see page 26391)	Returns the value of the Modified (see page 26407) property.
	GetNodeIndentStr (see page 26392)	Returns the value of the NodeIndentStr (see page 26408) property.
	GetOptions (see page 26392)	Returns the value of the Options (see page 26408) property.
	GetParseOptions (see page 26392)	Returns the value of the ParseOptions (see page 26409) property.
	GetSchemaRef (see page 26392)	Returns the value of the SchemaRef (see page 26409) property.
	GetStandAlone (see page 26393)	Returns the value of the StandAlone (see page 26410) property.
	GetVersion (see page 26393)	Returns the value of the Version (see page 26410) property.
	GetXML (see page 26393)	Returns the value of the XML (see page 26410) property.
	IsEmptyDoc (see page 26394)	Indicates whether the IXMLDocument (see page 26382) instance represents an empty document.
	LoadFromFile (see page 26394)	Loads an XML (see page 26410) document from disk.
	LoadFromStream (see page 26395)	Loads an XML (see page 26410) document from a stream and activates it.
	LoadFromXML (see page 26395)	This is the overview for the LoadFromXML method overload.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Refresh (see page 26396)	Updates the parsed XML (see page 26410) document to reflect external changes.
	RegisterDocBinding (see page 26396)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 26397)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 26397)	Saves the XML (see page 26410) document to disk.
	SaveToStream (see page 26398)	Saves the XML (see page 26410) document to a stream.
	SaveToXML (see page 26398)	This is the overview for the SaveToXML method overload.
	SetActive (see page 26399)	Sets the value of the Active (see page 26403) property.
	SetDefinition (see page 25903)	Sets the value of the Definition (see page 25904) property.
	SetDocumentElement (see page 26399)	Sets the value of the DocumentElement (see page 26405) property.
	SetDOMDocument (see page 26399)	Sets the value of the DOMDocument (see page 26405) property.
	SetEncoding (see page 26400)	Sets the value of the Encoding (see page 26406) property.
	SetFileName (see page 26400)	Sets the value of the FileName (see page 26406) property.
	SetNodeIndentStr (see page 26400)	Sets the value of the NodeIndentStr (see page 26408) property.
	SetOnAsyncLoad (see page 26401)	Supplies an event handler that responds when the AsyncLoadState (see page 26404) property changes.
	SetOptions (see page 26401)	Sets the value of the Options (see page 26408) property.
	SetParseOptions (see page 26402)	Sets the value of the ParseOptions (see page 26409) property.
	SetStandAlone (see page 26402)	Sets the value of the StandAlone (see page 26410) property.
	SetVersion (see page 26402)	Sets the value of the Version (see page 26410) property.
	SetXML (see page 26403)	Sets the value of the XML (see page 26410) property.

Public Properties

	Name	Description
	Active (see page 26403)	Specifies whether the XML (see page 26410) document has been parsed so that the DOM interfaces are available to examine or change its contents.

R	AsyncLoadState (see page 26404)	Indicates the state of the DOM parser when it parses the XML (see page 26410) document asynchronously.
R	ChildNodes (see page 26405)	Lists all of the child nodes of the document.
	Definition (see page 25904)	Provides access to the definitions contained in the WSDL document.
	DocumentElement (see page 26405)	Provides access to the root node of the XML (see page 26410) document.
	DOMDocument (see page 26405)	Provides access to the DOM interface for the parsed XML (see page 26410) document.
	Encoding (see page 26406)	Indicates the character set that the XML (see page 26410) document uses to encode characters.
	FileName (see page 26406)	Indicates the XML (see page 26410) document file, if any, that this interface represents.
R	Modified (see page 26407)	Indicates whether the XML (see page 26410) document has been modified.
R	Node (see page 26407)	Provides access to the document node for the XML (see page 26410) document.
	NodeIndentStr (see page 26408)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 26410) text.
	Options (see page 26408)	Influences various aspects of the document's behavior.
	ParseOptions (see page 26409)	Specifies implementation-specific options that are followed when parsing the XML (see page 26410) document.
R	SchemaRef (see page 26409)	Returns the name of the schema associated with this XML (see page 26410) document.
	StandAlone (see page 26410)	Indicates whether the XML (see page 26410) document includes any external markup declarations.
	Version (see page 26410)	Indicates the XML (see page 26410) version that the document uses.
	XML (see page 26410)	Specifies the content of the XML document associated with this IXMLDocument (see page 26382) interface.

189.2.29.2 IWSDLDocument Methods

The methods of the IWSDLDocument class are listed here.

Public Methods

	Name	Description
	GetDefinition (see page 25903)	Returns the value of the Definition (see page 25904) property.
	SetDefinition (see page 25903)	Sets the value of the Definition (see page 25904) property.

189.2.29.2.1 IWSDLDocument.GetDefinition Method

Returns the value of the Definition (see page 25904) property.

Pascal

```
function GetDefinition: IDefinition;
```

C++

```
__fastcall IDefinition GetDefinition();
```

Description

GetDefinition is the read access method for the Definition (see page 25904) property.

See Also

Definition (see page 25904)

189.2.29.2.2 IWSDLDocument.SetDefinition Method

Sets the value of the Definition (see page 25904) property.

Pascal

```
procedure SetDefinition(const Value: IDefinition);
```

C++

```
__fastcall SetDefinition(const IDefinition Value);
```

Description

SetDefinition is the write method for the Definition (see page 25904) property.

See Also

Definition (see page 25904)

189.2.29.3 IWSDLDocument Properties

The properties of the IWSDLDocument class are listed here.

Public Properties

	Name	Description
	Definition (see page 25904)	Provides access to the definitions contained in the WSDL document.

189.2.29.3.1 IWSDLDocument.Definition Property

Provides access to the definitions contained in the WSDL document.

Pascal

```
property Definition: IDefinition;
```

C++

```
__property IDefinition Definition;
```

Description

Definition is the IDefinition (see page 25831) interface of the root node in the WSDL document. This interface is implemented by the node specified by DocumentElement.

You can use the IDefinition (see page 25831) interface to access the Bindings, Messages (see page 16211), PortTypes, Services, and Types (see page 23293) defined in the WSDL document by reading properties of the same names. Alternately, you can use the methods of TWSDLItems. To obtain a TWSDLItems instance for this document,

1.Cast this interface to IWSDLItems using the global Supports function (C++) or the inherited Supports method (C++).

2.Use the GetWSDLItems method of that interface.

See Also

DocumentElement (see page 26405)

TWSDLItems (see page 25951)

Supports

Supports (see page 22973)

189.2.30 IXMLSchemaDefs Interface

Class Hierarchy

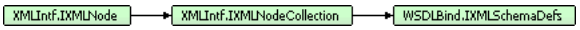

Pascal

```
IXMLSchemaDefs = interface(IXMLNodeCollection);
```

C++

```
__interface IXMLSchemaDefs : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IXMLSchemaDefs.

189.2.30.1 IXMLSchemaDefs Members

The following tables list the members exposed by IXMLSchemaDefs.

Public Methods

	Name	Description
	Add (see page 25907)	This is Add, a member of class IXMLSchemaDefs.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.

	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	GetXMLSchemaDef (see page 25907)	This is GetXMLSchemaDef, a member of class IXMLSchemaDefs.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	SchemaDef (see page 25907)	This is SchemaDef, a member of class IXMLSchemaDefs.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

189.2.30.2 IXMLSchemaDefs Methods

The methods of the IXMLSchemaDefs class are listed here.

Public Methods

	Name	Description
	Add (see page 25907)	This is Add, a member of class IXMLSchemaDefs.

	GetXMLSchemaDef (see page 25907)	This is GetXMLSchemaDef, a member of class IXMLSchemaDefs.

189.2.30.2.1 IXMLSchemaDefs.Add Method

Pascal

```
function Add(const NamespaceURI: WideString = ''; const TargetNamespace: WideString = ''):
IXMLSchemaDef;
```

C++

```
__fastcall IXMLSchemaDef Add(const BSTR NamespaceURI = '', const BSTR TargetNamespace = '');
```

Description

This is Add, a member of class IXMLSchemaDefs.

189.2.30.2.2 IXMLSchemaDefs.GetXMLSchemaDef Method

Pascal

```
function GetXMLSchemaDef(Index: Integer): IXMLSchemaDef;
```

C++

```
__fastcall IXMLSchemaDef GetXMLSchemaDef(int Index);
```

Description

This is GetXMLSchemaDef, a member of class IXMLSchemaDefs.

189.2.30.3 IXMLSchemaDefs Properties

The properties of the IXMLSchemaDefs class are listed here.

Public Properties

	Name	Description
R	SchemaDef (see page 25907)	This is SchemaDef, a member of class IXMLSchemaDefs.

189.2.30.3.1 IXMLSchemaDefs.SchemaDef Property

Pascal

```
property SchemaDef [Index: Integer]: IXMLSchemaDef;
```

C++

```
__property IXMLSchemaDef SchemaDef[int Index];
```

Description

This is SchemaDef, a member of class IXMLSchemaDefs.

189.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetDefinition (see page 25908)	This is function WSDLBind.GetDefinition.
	LoadWSDLDoc (see page 25908)	This is function WSDLBind.LoadWSDLDoc.
	NewWSDLDoc (see page 25908)	This is function WSDLBind.NewWSDLDoc.

189.3.1 WSDLBind.GetDefinition Function

Pascal

```
function GetDefinition(Doc: TXMLDocument): IDefinition;
```

C++

```
IDefinition GetDefinition(TXMLDocument Doc);
```

File

WSDLBind

Description

This is function WSDLBind.GetDefinition.

189.3.2 WSDLBind.LoadWSDLDoc Function

Pascal

```
function LoadWSDLDoc(const FileName: WideString): IWSDLDocument;
```

C++

```
IWSDLDocument LoadWSDLDoc(const BSTR FileName);
```

File

WSDLBind

Description

This is function WSDLBind.LoadWSDLDoc.

189.3.3 WSDLBind.NewWSDLDoc Function

Pascal

```
function NewWSDLDoc: IWSDLDocument;
```

C++

```
IWSDLDocument NewWSDLDoc();
```

File

WSDLBind

Description

This is function WSDLBind.NewWSDLDoc.

189.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
SBinding (see page 25910)	This is constant WSDLBind.SBinding.
SDefinitions (see page 25910)	This is constant WSDLBind.SDefinitions.
SFault (see page 25911)	This is constant WSDLBind.SFault.
SImport (see page 25911)	This is constant WSDLBind.SImport.
SInput (see page 25911)	This is constant WSDLBind.SInput.
SLocation (see page 25912)	This is constant WSDLBind.SLocation.
SMessage (see page 25912)	This is constant WSDLBind.SMessage.
SNamespace (see page 25912)	This is constant WSDLBind.SNamespace.
SOperation (see page 25913)	This is constant WSDLBind.SOperation.
SOutput (see page 25913)	This is constant WSDLBind.SOutput.
SPart (see page 25913)	This is constant WSDLBind.SPart.
SPort (see page 25914)	This is constant WSDLBind.SPort.
SPortType (see page 25914)	This is constant WSDLBind.SPortType.
SService (see page 25914)	This is constant WSDLBind.SService.
SType (see page 25915)	This is constant WSDLBind.SType.
STypes (see page 25915)	This is constant WSDLBind.STypes.

189.4.1 WSDLBind.SBinding Constant

Pascal

```
SBinding = 'binding';
```

C++

```
SBinding = 'binding';
```

File

WSDLBind

Description

This is constant WSDLBind.SBinding.

189.4.2 WSDLBind.SDefinitions Constant

Pascal

```
SDefinitions = 'definitions';
```

C++

```
SDefinitions = 'definitions';
```

File

WSDLBind

Description

This is constant WSDLBind.SDefinitions.

189.4.3 WSDLBind.SFault Constant

Pascal

```
SFault = 'fault';
```

C++

```
SFault = 'fault';
```

File

WSDLBind

Description

This is constant WSDLBind.SFault.

189.4.4 WSDLBind.SImport Constant

Pascal

```
SImport = 'import';
```

C++

```
SImport = 'import';
```

File

WSDLBind

Description

This is constant WSDLBind.SImport.

189.4.5 WSDLBind.SInput Constant

Pascal

```
SInput = 'input';
```

C++

```
SInput = 'input';
```

File

WSDLBind

Description

This is constant WSDLBind.SInput.

189.4.6 WSDLBind.SLocation Constant

Pascal

```
SLocation = 'location';
```

C++

```
SLocation = 'location';
```

File

WSDLBind

Description

This is constant WSDLBind.SLocation.

189.4.7 WSDLBind.SMessage Constant

Pascal

```
SMessage = 'message';
```

C++

```
SMessage = 'message';
```

File

WSDLBind

Description

This is constant WSDLBind.SMessage.

189.4.8 WSDLBind.SNamespace Constant

Pascal

```
SNamespace = 'namespace';
```

C++

```
SNamespace = 'namespace';
```

File

WSDLBind

Description

This is constant WSDLBind.SNamespace.

189.4.9 WSDLBind.SOperation Constant

Pascal

```
SOperation = 'operation';
```

C++

```
SOperation = 'operation';
```

File

WSDLBind

Description

This is constant WSDLBind.SOperation.

189.4.10 WSDLBind.SOutput Constant

Pascal

```
SOutput = 'output';
```

C++

```
SOutput = 'output';
```

File

WSDLBind

Description

This is constant WSDLBind.SOutput.

189.4.11 WSDLBind.SPart Constant

Pascal

```
SPart = 'part';
```

C++

```
SPart = 'part';
```

File

WSDLBind

Description

This is constant WSDLBind.SPart.

189.4.12 WSDLBind.SPort Constant

Pascal

```
SPort = 'port';
```

C++

```
SPort = 'port';
```

File

WSDLBind

Description

This is constant WSDLBind.SPort.

189.4.13 WSDLBind.SPortType Constant

Pascal

```
SPortType = 'portType';
```

C++

```
SPortType = 'portType';
```

File

WSDLBind

Description

This is constant WSDLBind.SPortType.

189.4.14 WSDLBind.SService Constant

Pascal

```
SService = 'service';
```

C++

```
SService = 'service';
```

File

WSDLBind

Description

This is constant WSDLBind.SService.

189.4.15 WSDLBind.SType Constant

Pascal

```
SType = 'type';
```

C++

```
SType = 'type';
```

File

WSDLBind

Description

This is constant WSDLBind.SType.

189.4.16 WSDLBind.STypes Constant

Pascal

```
STypes = 'types';
```

C++

```
STypes = 'types';
```

File

WSDLBind

Description

This is constant WSDLBind.STypes.

190 WSDLIntf Namespace

This is namespace WSDLIntf.

Topics

Name	Description
Classes (see page 25919)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 25925)	The following table lists structs, records, enums in this documentation.
Types (see page 25927)	The following table lists types in this documentation.
Constants (see page 25928)	The following table lists constants in this documentation.

Classes

	Name	Description
	TWideStrings (see page 25919)	TWideStrings represents a list of WideString values.

Constants

Name	Description
Directives (see page 25929)	This is constant WSDLIntf.Directives.
DynArrayTypes (see page 25929)	This is constant WSDLIntf.DynArrayTypes.
Httpns (see page 25930)	This is constant WSDLIntf.Httpns.
Operators (see page 25930)	This is constant WSDLIntf.Operators.
OtherReservedWords (see page 25930)	This is constant WSDLIntf.OtherReservedWords.
ReservedWords (see page 25931)	This is constant WSDLIntf.ReservedWords.
SAddress (see page 25931)	This is constant WSDLIntf.SAddress.
SAnySimpleType (see page 25932)	This is constant WSDLIntf.SAnySimpleType.
SAnyType (see page 25932)	This is constant WSDLIntf.SAnyType.
SArray (see page 25932)	This is constant WSDLIntf.SArray.
SArrayOf (see page 25933)	This is constant WSDLIntf.SArrayOf.
SArrayType (see page 25933)	This is constant WSDLIntf.SArrayType.
SBinding (see page 25933)	This is constant WSDLIntf.SBinding.
SBody (see page 25934)	This is constant WSDLIntf.SBody.
ScalarArrayTypes (see page 25934)	This is constant WSDLIntf.ScalarArrayTypes.
SComplexType (see page 25934)	This is constant WSDLIntf.SComplexType.
SDefinitions (see page 25935)	This is constant WSDLIntf.SDefinitions.
SDynArray (see page 25935)	This is constant WSDLIntf.SDynArray.
SElement (see page 25935)	This is constant WSDLIntf.SElement.
SEncodingStyle (see page 25936)	This is constant WSDLIntf.SEncodingStyle.

SFault (see page 25936)	This is constant WSDLIntf.SFault.
SHeader (see page 25936)	This is constant WSDLIntf.SHeader.
SImport (see page 25937)	This is constant WSDLIntf.SImport.
SInOut (see page 25937)	This is constant WSDLIntf.SInOut.
SInput (see page 25937)	This is constant WSDLIntf.SInput.
SLocation (see page 25938)	This is constant WSDLIntf.SLocation.
SMessage (see page 25938)	This is constant WSDLIntf.SMessage.
SName (see page 25938)	This is constant WSDLIntf.SName.
SNameSpace (see page 25939)	This is constant WSDLIntf.SNameSpace.
SNsPrefix (see page 25939)	This is constant WSDLIntf.SNsPrefix.
Soapns (see page 25939)	This is constant WSDLIntf.Soapns.
SOperation (see page 25940)	This is constant WSDLIntf.SOperation.
SOutput (see page 25940)	This is constant WSDLIntf.SOutput.
SPart (see page 25940)	This is constant WSDLIntf.SPart.
SParts (see page 25941)	This is constant WSDLIntf.SParts.
SPort (see page 25941)	This is constant WSDLIntf.SPort.
SPortType (see page 25941)	This is constant WSDLIntf.SPortType.
SRequest (see page 25942)	This is constant WSDLIntf.SRequest.
SResponse (see page 25942)	This is constant WSDLIntf.SResponse.
SReturn (see page 25942)	This is constant WSDLIntf.SReturn.
SSchema (see page 25943)	This is constant WSDLIntf.SSchema.
SService (see page 25943)	This is constant WSDLIntf.SService.
SSoapAction (see page 25943)	This is constant WSDLIntf.SSoapAction.
SStyle (see page 25944)	This is constant WSDLIntf.SStyle.
STns (see page 25944)	This is constant WSDLIntf.STns.
STransport (see page 25944)	This is constant WSDLIntf.STransport.
SType (see page 25945)	This is constant WSDLIntf.SType.
STypes (see page 25945)	This is constant WSDLIntf.STypes.
SUnknown (see page 25945)	This is constant WSDLIntf.SUnknown.
SUse (see page 25946)	This is constant WSDLIntf.SUse.
SVerb (see page 25946)	This is constant WSDLIntf.SVerb.
tns (see page 25946)	This is constant WSDLIntf.tns.
Wsdlns (see page 25947)	This is constant WSDLIntf.Wsdlns.
xsdns (see page 25947)	This is constant WSDLIntf.xsdns.

Structs, Records, Enums

	Name	Description
	TWebServiceBindingType (see page 25925)	This is record WSDLIntf.TWebServiceBindingType.
	TWSDLElement (see page 25925)	This is record WSDLIntf.TWSDLElement.
	TWString (see page 25926)	This is record WSDLIntf.TWString.

Types

Name	Description
TWSDLElements (see page 25927)	This is type WSDLIntf.TWSDLElements.

190.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TWideStrings (see page 25919)	TWideStrings represents a list of WideString values.

190.1.1 TWideStrings Class

TWideStrings represents a list of WideString values.

Class Hierarchy

WSDLIntf.TWideStrings

Pascal

```
TWideStrings = class;
```

C++

```
class TWideStrings;
```

File

WSDLIntf

Description

TWideStrings manages a list of WideString values. It is used for handling the strings in a WSDL document.

TWideStrings introduces properties and methods to

Add (see page 25921) a WideString to the list.

Read or set the WideString at a particular location.

Find the position of a specified WideString.

See Also

TStringList (see page 3006)

TList (see page 2868)

THashedStringList (see page 15450)

TWSDLDocument (see page 25799)

TWSDLItems (see page 25951)

WideString (see page 22377)

190.1.1.1 TWideStrings Members

The following tables list the members exposed by TWideStrings.

Public Methods

	Name	Description
	Add (see page 25921)	Adds a WideString at the end of the list.
	Clear (see page 25921)	Empties the list of all its strings.
	Count (see page 25922)	Returns the number of WideStrings in the list.
	Create (see page 25920)	Creates an instance of TWideStrings (see page 25919).
V	Destroy (see page 25920)	Destroys an instance of TWideStrings (see page 25919).
	IndexOf (see page 25922)	Returns the position of a string in the list.
	IndexOfIgnoreCase (see page 25922)	Returns the position of a string (or one that differs only in case) in the list.
	Insert (see page 25923)	Inserts a WideString at a specified position in the list.

Public Properties

	Name	Description
	Strings (see page 25923)	References the WideStrings in the list by position.

190.1.1.2 TWideStrings.Create Constructor

Creates an instance of TWideStrings (see page 25919).

Pascal

```
constructor Create;
```

C++

```
__fastcall TWideStrings();
```

Description

Call Create to create a TWideStrings (see page 25919) object at runtime.

190.1.1.3 TWideStrings.Destroy Destructor

Destroys an instance of TWideStrings (see page 25919).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWideStrings();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TWideStrings (see page 25919) reference is not nil, and only then calls Destroy.

See Also

Free (see page 22300)

190.1.1.4 TWideStrings Methods

The methods of the TWideStrings class are listed here.

Public Methods

	Name	Description
	Add (see page 25921)	Adds a WideString at the end of the list.
	Clear (see page 25921)	Empties the list of all its strings.
	Count (see page 25922)	Returns the number of WideStrings in the list.
	IndexOf (see page 25922)	Returns the position of a string in the list.
	IndexOfIgnoreCase (see page 25922)	Returns the position of a string (or one that differs only in case) in the list.
	Insert (see page 25923)	Inserts a WideString at a specified position in the list.

190.1.1.4.1 TWideStrings.Add Method

Adds a WideString at the end of the list.

Pascal

```
function Add(const S: WideString): Integer;
```

C++

```
__fastcall int Add(const BSTR S);
```

Description

Call Add to add a string to the end of the list. Add returns the index of the new string.

See Also

Clear (see page 25921)

Count (see page 25922)

Strings (see page 25923)

Insert (see page 25923)

190.1.1.4.2 TWideStrings.Clear Method

Empties the list of all its strings.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear method to delete all the strings in the list.

See Also

Add (see page 25921)

Strings (see page 25923)

190.1.1.4.3 TWideStrings.Count Method

Returns the number of WideStrings in the list.

Pascal

```
function Count: Integer;
```

C++

```
__fastcall int Count();
```

Description

Call Count method to determine the number of strings in the list. This value provides an upper bound to the Index parameter of the Strings (see page 25923) property.

See Also

Strings (see page 25923)

190.1.1.4.4 TWideStrings.IndexOf Method

Returns the position of a string in the list.

Pascal

```
function IndexOf(const S: WideString): Integer;
```

C++

```
__fastcall int IndexOf(const BSTR S);
```

Description

Call IndexOf to obtain the position of the first occurrence of the string S. IndexOf returns the 0-based index of the string. Thus, if S matches the first string in the list, IndexOf returns 0, if S is the second string, IndexOf returns 1, and so on. If the string is not in the string list, IndexOf returns -1.

IndexOf uses a case-sensitive search. To locate a string case-insensitively, use the IndexOfIgnoreCase (see page 25922) method instead.

Note: If the string appears in the list more than once, IndexOf returns the position of the first occurrence.

See Also

Strings (see page 25923)

IndexOfIgnoreCase (see page 25922)

190.1.1.4.5 TWideStrings.IndexOfIgnoreCase Method

Returns the position of a string (or one that differs only in case) in the list.

Pascal

```
function IndexOfIgnoreCase(const S: WideString): Integer;
```

C++

```
__fastcall int IndexOfIgnoreCase(const BSTR S);
```

Description

Call IndexOfIgnoreCase to obtain the position of the first occurrence of the string S, or of a string that differs from S only by case.

IndexOfIgnoreCase returns the 0-based index of the string. Thus, if S matches the first string in the list, IndexOfIgnoreCase returns 0, if S is the second string, IndexOfIgnoreCase returns 1, and so on. If the string is not in the string list, IndexOfIgnoreCase returns -1.

IndexOfIgnoreCase uses a case-insensitive search. To locate a string case-sensitively, use the IndexOf (see page 25922) method instead.

Note: If the string appears in the list more than once, IndexOfIgnoreCase returns the position of the first occurrence.

See Also

Strings (see page 25923)

IndexOf (see page 25922)

190.1.1.4.6 TWideStrings.Insert Method

Inserts a WideString at a specified position in the list.

Pascal

```
procedure Insert(Index: Integer; const S: WideString);
```

C++

```
__fastcall Insert(int Index, const BSTR S);
```

Description

Call Insert to add a WideString to a specified position in the list. Insert adds the string specified by S so that it has the index specified by Index. The string previously at that position, as well as all subsequent strings in the list are moved so that their indexes are one greater than before.

See Also

Clear (see page 25921)

Count (see page 25922)

Strings (see page 25923)

Add (see page 25921)

190.1.1.5 TWideStrings Properties

The properties of the TWideStrings class are listed here.

Public Properties

	Name	Description
	Strings (see page 25923)	References the WideStrings in the list by position.

190.1.1.5.1 TWideStrings.Strings Property

References the WideStrings in the list by position.

Pascal

```
property Strings [Index: Integer]: WideString;
```

C++

```
__property BSTR Strings[int Index];
```

Description

Use the Strings property to get or set the string at a particular position.

Note: In Delphi, Strings is the default property for TWideStrings (see page 25919). This means that the property name can be omitted in code. That is, the statement

```
WString := MyStrings.Strings[0];
```

Can be written

```
WString := MyStrings[0];
```

Note: In C++, Strings can be accessed using the [] operator, to achieve an affect similar to the default property in Delphi.

See Also

IndexOf (see page 25922)

operator_sb

190.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TWebServiceBindingType (see page 25925)	This is record WSDLIntf.TWebServiceBindingType.
	TWSDLElement (see page 25925)	This is record WSDLIntf.TWSDLElement.

Records

	Name	Description
	TWString (see page 25926)	This is record WSDLIntf.TWString.

190.2.1 WSDLIntf.TWebServiceBindingType Enumeration

Pascal

```
TWebServiceBindingType = (
  btUnknown,
  btSoap,
  btHttp,
  btMime,
  btDIME
);
```

C++

```
enum TWebServiceBindingType {
  btUnknown,
  btSoap,
  btHttp,
  btMime,
  btDIME
};
```

File

WSDLIntf

Description

This is record WSDLIntf.TWebServiceBindingType.

190.2.2 WSDLIntf.TWSDLElement Enumeration

Pascal

```
TWSDLElement = (
  weServiceIntf,
  weServiceImpl,
  weMessage,
  wePortType,
  weBinding,
```

```
    weService,
    weTypes,
    WeImport,
    weOperation,
    wePart
  );
```

C++

```
  enum TWSDLElement {
    weServiceIntf,
    weServiceImpl,
    weMessage,
    wePortType,
    weBinding,
    weService,
    weTypes,
    WeImport,
    weOperation,
    wePart
  };
```

File

WSDLIntf

Description

This is record WSDLIntf.TWSDLElement.

190.2.3 WSDLIntf.TWString Record

Pascal

```
  TWString = record
    WString: WideString;
  end;
```

C++

```
  struct TWString {
    BSTR WString;
  };
```

File

WSDLIntf

Description

This is record WSDLIntf.TWString.

190.3 Types

The following table lists types in this documentation.

Types

Name	Description
TWSDLElements (see page 25927)	This is type WSDLIntf.TWSDLElements.

190.3.1 WSDLIntf.TWSDLElements Type

Pascal

```
TWSDLElements = set of TWSDLElement;
```

C++

```
TWSDLElement TWSDLElements;
```

File

WSDLIntf

Description

This is type WSDLIntf.TWSDLElements.

190.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
Directives (see page 25929)	This is constant WSDLIntf.Directives.
DynArrayTypes (see page 25929)	This is constant WSDLIntf.DynArrayTypes.
Httpns (see page 25930)	This is constant WSDLIntf.Httpns.
Operators (see page 25930)	This is constant WSDLIntf.Operators.
OtherReservedWords (see page 25930)	This is constant WSDLIntf.OtherReservedWords.
ReservedWords (see page 25931)	This is constant WSDLIntf.ReservedWords.
SAddress (see page 25931)	This is constant WSDLIntf.SAddress.
SAnySimpleType (see page 25932)	This is constant WSDLIntf.SAnySimpleType.
SAnyType (see page 25932)	This is constant WSDLIntf.SAnyType.
SArray (see page 25932)	This is constant WSDLIntf.SArray.
SArrayOf (see page 25933)	This is constant WSDLIntf.SArrayOf.
SArrayType (see page 25933)	This is constant WSDLIntf.SArrayType.
SBinding (see page 25933)	This is constant WSDLIntf.SBinding.
SBody (see page 25934)	This is constant WSDLIntf.SBody.
ScalarArrayTypes (see page 25934)	This is constant WSDLIntf.ScalarArrayTypes.
SComplexType (see page 25934)	This is constant WSDLIntf.SComplexType.
SDefinitions (see page 25935)	This is constant WSDLIntf.SDefinitions.
SDynArray (see page 25935)	This is constant WSDLIntf.SDynArray.
SElement (see page 25935)	This is constant WSDLIntf.SElement.
SEncodingStyle (see page 25936)	This is constant WSDLIntf.SEncodingStyle.
SFault (see page 25936)	This is constant WSDLIntf.SFault.
SHeader (see page 25936)	This is constant WSDLIntf.SHeader.
SImport (see page 25937)	This is constant WSDLIntf.SImport.
SInOut (see page 25937)	This is constant WSDLIntf.SInOut.
SInput (see page 25937)	This is constant WSDLIntf.SInput.
SLocation (see page 25938)	This is constant WSDLIntf.SLocation.
SMessage (see page 25938)	This is constant WSDLIntf.SMessage.
SName (see page 25938)	This is constant WSDLIntf.SName.
SNameSpace (see page 25939)	This is constant WSDLIntf.SNameSpace.
SNsPrefix (see page 25939)	This is constant WSDLIntf.SNsPrefix.
Soapns (see page 25939)	This is constant WSDLIntf.Soapns.
SOperation (see page 25940)	This is constant WSDLIntf.SOperation.
SOutput (see page 25940)	This is constant WSDLIntf.SOutput.
SPart (see page 25940)	This is constant WSDLIntf.SPart.
SParts (see page 25941)	This is constant WSDLIntf.SParts.
SPort (see page 25941)	This is constant WSDLIntf.SPort.
SPortType (see page 25941)	This is constant WSDLIntf.SPortType.
SRequest (see page 25942)	This is constant WSDLIntf.SRequest.
SResponse (see page 25942)	This is constant WSDLIntf.SResponse.
SReturn (see page 25942)	This is constant WSDLIntf.SReturn.
SSchema (see page 25943)	This is constant WSDLIntf.SSchema.
SService (see page 25943)	This is constant WSDLIntf.SService.
SSoapAction (see page 25943)	This is constant WSDLIntf.SSoapAction.
SStyle (see page 25944)	This is constant WSDLIntf.SStyle.

STns (see page 25944)	This is constant WSDLIntf.STns.
STransport (see page 25944)	This is constant WSDLIntf.STransport.
SType (see page 25945)	This is constant WSDLIntf.SType.
STypes (see page 25945)	This is constant WSDLIntf.STypes.
SUnknown (see page 25945)	This is constant WSDLIntf.SUnknown.
SUse (see page 25946)	This is constant WSDLIntf.SUse.
SVerb (see page 25946)	This is constant WSDLIntf.SVerb.
tns (see page 25946)	This is constant WSDLIntf.tns.
Wsdlns (see page 25947)	This is constant WSDLIntf.Wsdlns.
xsdns (see page 25947)	This is constant WSDLIntf.xsdns.

190.4.1 WSDLIntf.Directives Constant

Pascal

```
Directives: array[0..43] of string =
('absolute','abstract','assembler','automated','cdecl','contains', 'default','dispid',
'dynamic', 'export','external','far','forward','implements','index','message',
'name','near','nodefault','overload','override','package','pascal','private','protected','publi
c',
'published','read','readonly','register','reintroduce','requires','resident','safecall','stdcal
l',
'stored','virtual','write','writeonly','on','platform','deprecated','local','varargs');
```

C++

```
array[0..43] of string Directives =
('absolute','abstract','assembler','automated','cdecl','contains', 'default','dispid',
'dynamic', 'export','external','far','forward','implements','index','message',
'name','near','nodefault','overload','override','package','pascal','private','protected','publi
c',
'published','read','readonly','register','reintroduce','requires','resident','safecall','stdcal
l',
'stored','virtual','write','writeonly','on','platform','deprecated','local','varargs');
```

File

WSDLIntf

Description

This is constant WSDLIntf.Directives.

190.4.2 WSDLIntf.DynArrayTypes Constant

Pascal

```
DynArrayTypes: array[0..12] of string =
('TIntegerDynArray','TCardinalDynArray','TWordDynArray',
'TSmallIntDynArray','TByteDynArray','TShortIntDynArray','TInt64DynArray','TLongWordDynArray','T
SingleDynArray',
'TDoubleDynArray','TBooleanDynArray','TStringDynArray','TWideStringDynArray');
```

C++

```
array[0..12] of string DynArrayTypes =
('TIntegerDynArray','TCardinalDynArray','TWordDynArray',
'TSmallIntDynArray','TByteDynArray','TShortIntDynArray','TInt64DynArray','TLongWordDynArray','T
SingleDynArray',
```

```
'TDoubleDynArray','TBooleanDynArray','TStringDynArray','TWideStringDynArray');
```

File

WSDLIntf

Description

This is constant WSDLIntf.DynArrayTypes.

190.4.3 WSDLIntf.Httpns Constant

Pascal

```
Httpns = Wsdlns +'http/';
```

C++

```
Httpns = Wsdlns +'http/';
```

File

WSDLIntf

Description

This is constant WSDLIntf.Httpns.

190.4.4 WSDLIntf.Operators Constant

Pascal

```
Operators: array[0..12] of string = ('+','-','*','/','@','^','=','>','<','<>','<=','>=',':=');
```

C++

```
array[0..12] of string Operators = ('+','-','*','/','@','^','=','>','<','<>','<=','>=',':=');
```

File

WSDLIntf

Description

This is constant WSDLIntf.Operators.

190.4.5 WSDLIntf.OtherReservedWords Constant

Pascal

```
OtherReservedWords: array[0..0] of string = ('Application');
```

C++

```
array[0..0] of string OtherReservedWords = ('Application');
```

File

WSDLIntf

Description

This is constant WSDLIntf.OtherReservedWords.

190.4.6 WSDLIntf.ReservedWords Constant

Pascal

```
ReservedWords: array[0..66] of string = ('and', 'array', 'as', 'asm', 'begin','case',
'class','const','constructor','destructor', 'dispinterface', 'div', 'do','downto', 'else',
'end', 'except', 'exports', 'file', 'finalization', 'finally', 'for','function', 'goto',
'high', 'if', 'implementation', 'in','inherited','initialization', 'inline','interface',
'is','label','library', 'low',
'mod','nil','not','object','of','or','out','packed','procedure',
'program','property','raise','record','repeat', 'resourcestring', 'set','shl','shr', 'string',
'then', 'threadvar', 'to','try','type','unit','until','uses','var','while','with','xor');
```

C++

```
array[0..66] of string ReservedWords = ('and', 'array', 'as', 'asm', 'begin','case',
'class','const','constructor','destructor', 'dispinterface', 'div', 'do','downto', 'else',
'end', 'except', 'exports', 'file', 'finalization', 'finally', 'for','function', 'goto',
'high', 'if', 'implementation', 'in','inherited','initialization', 'inline','interface',
'is','label','library', 'low',
'mod','nil','not','object','of','or','out','packed','procedure',
'program','property','raise','record','repeat', 'resourcestring', 'set','shl','shr', 'string',
'then', 'threadvar', 'to','try','type','unit','until','uses','var','while','with','xor');
```

File

WSDLIntf

Description

This is constant WSDLIntf.ReservedWords.

190.4.7 WSDLIntf.SAddress Constant

Pascal

```
SAddress = 'address';
```

C++

```
SAddress = 'address';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SAddress.

190.4.8 WSDLIntf.SAnySimpleType Constant

Pascal

```
SAnySimpleType = 'anySimpleType';
```

C++

```
SAnySimpleType = 'anySimpleType';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SAnySimpleType.

190.4.9 WSDLIntf.SAnyType Constant

Pascal

```
SAnyType = 'anyType';
```

C++

```
SAnyType = 'anyType';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SAnyType.

190.4.10 WSDLIntf.SArray Constant

Pascal

```
SArray = 'Array';
```

C++

```
SArray = 'Array';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SArray.

190.4.11 WSDLIntf.SArrayOf Constant

Pascal

```
SArrayOf = 'ArrayOf';
```

C++

```
SArrayOf = 'ArrayOf';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SArrayOf.

190.4.12 WSDLIntf.SArrayType Constant

Pascal

```
SArrayType = 'arrayType';
```

C++

```
SArrayType = 'arrayType';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SArrayType.

190.4.13 WSDLIntf.SBinding Constant

Pascal

```
SBinding = 'binding';
```

C++

```
SBinding = 'binding';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SBinding.

190.4.14 WSDLIntf.SBody Constant

Pascal

```
SBody = 'body';
```

C++

```
SBody = 'body';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SBody.

190.4.15 WSDLIntf.ScalarArrayTypes Constant

Pascal

```
ScalarArrayTypes: array[0..12] of string =
('Integer','Cardinal','Word','SmallInt','Byte','ShortInt','Int64',
'LongWord','Single','Double','Boolean','String','WideString');
```

C++

```
array[0..12] of string ScalarArrayTypes =
('Integer','Cardinal','Word','SmallInt','Byte','ShortInt','Int64',
'LongWord','Single','Double','Boolean','String','WideString');
```

File

WSDLIntf

Description

This is constant WSDLIntf.ScalarArrayTypes.

190.4.16 WSDLIntf.SComplexType Constant

Pascal

```
SComplexType = 'complexType';
```

C++

```
SComplexType = 'complexType';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SComplexType.

190.4.17 WSDLIntf.SDefinitions Constant

Pascal

```
SDefinitions = 'definitions';
```

C++

```
SDefinitions = 'definitions';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SDefinitions.

190.4.18 WSDLIntf.SDynArray Constant

Pascal

```
SDynArray = 'array of ';
```

C++

```
SDynArray = 'array of ';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SDynArray.

190.4.19 WSDLIntf.SElement Constant

Pascal

```
SElement = 'element';
```

C++

```
SElement = 'element';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SElement.

190.4.20 WSDLIntf.SEncodingStyle Constant

Pascal

```
SEncodingStyle = 'encodingStyle';
```

C++

```
SEncodingStyle = 'encodingStyle';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SEncodingStyle.

190.4.21 WSDLIntf.SFault Constant

Pascal

```
SFault = 'fault';
```

C++

```
SFault = 'fault';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SFault.

190.4.22 WSDLIntf.SHeader Constant

Pascal

```
SHeader = 'header';
```

C++

```
SHeader = 'header';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SHeader.

190.4.23 WSDLIntf.SImport Constant

Pascal

```
SImport = 'import';
```

C++

```
SImport = 'import';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SImport.

190.4.24 WSDLIntf.SInOut Constant

Pascal

```
SInOut = 'inout';
```

C++

```
SInOut = 'inout';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SInOut.

190.4.25 WSDLIntf.SInput Constant

Pascal

```
SInput = 'input';
```

C++

```
SInput = 'input';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SInput.

190.4.26 WSDLIntf.SLocation Constant

Pascal

```
SLocation = 'location';
```

C++

```
SLocation = 'location';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SLocation.

190.4.27 WSDLIntf.SMessage Constant

Pascal

```
SMessage = 'message';
```

C++

```
SMessage = 'message';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SMessage.

190.4.28 WSDLIntf.SName Constant

Pascal

```
SName = 'name';
```

C++

```
SName = 'name';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SName.

190.4.29 WSDLIntf.SNameSpace Constant

Pascal

```
SNameSpace = 'namespace';
```

C++

```
SNameSpace = 'namespace';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SNameSpace.

190.4.30 WSDLIntf.SNsPrefix Constant

Pascal

```
SNsPrefix = 'ns';
```

C++

```
SNsPrefix = 'ns';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SNsPrefix.

190.4.31 WSDLIntf.Soapns Constant

Pascal

```
Soapns = Wsdlns +'soap/';
```

C++

```
Soapns = Wsdlns +'soap/';
```

File

WSDLIntf

Description

This is constant WSDLIntf.Soapns.

190.4.32 WSDLIntf.SOperation Constant

Pascal

```
SOperation = 'operation';
```

C++

```
SOperation = 'operation';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SOperation.

190.4.33 WSDLIntf.SOutput Constant

Pascal

```
SOutput = 'output';
```

C++

```
SOutput = 'output';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SOutput.

190.4.34 WSDLIntf.SPart Constant

Pascal

```
SPart = 'part';
```

C++

```
SPart = 'part';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SPart.

190.4.35 WSDLIntf.SParts Constant

Pascal

```
SParts = 'parts';
```

C++

```
SParts = 'parts';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SParts.

190.4.36 WSDLIntf.SPort Constant

Pascal

```
SPort = 'port';
```

C++

```
SPort = 'port';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SPort.

190.4.37 WSDLIntf.SPortType Constant

Pascal

```
SPortType = 'portType';
```

C++

```
SPortType = 'portType';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SPortType.

190.4.38 WSDLIntf.SRequest Constant

Pascal

```
SRequest = 'Request';
```

C++

```
SRequest = 'Request';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SRequest.

190.4.39 WSDLIntf.SResponse Constant

Pascal

```
SResponse = 'Response';
```

C++

```
SResponse = 'Response';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SResponse.

190.4.40 WSDLIntf.SReturn Constant

Pascal

```
SReturn = 'return';
```

C++

```
SReturn = 'return';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SReturn.

190.4.41 WSDLIntf.SSchema Constant

Pascal

```
SSchema = 'schema';
```

C++

```
SSchema = 'schema';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SSchema.

190.4.42 WSDLIntf.SService Constant

Pascal

```
SService = 'service';
```

C++

```
SService = 'service';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SService.

190.4.43 WSDLIntf.SSoapAction Constant

Pascal

```
SSoapAction = 'soapAction';
```

C++

```
SSoapAction = 'soapAction';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SSoapAction.

190.4.44 WSDLIntf.SStyle Constant

Pascal

```
SStyle = 'style';
```

C++

```
SStyle = 'style';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SStyle.

190.4.45 WSDLIntf.STns Constant

Pascal

```
STns = 'targetNamespace';
```

C++

```
STns = 'targetNamespace';
```

File

WSDLIntf

Description

This is constant WSDLIntf.STns.

190.4.46 WSDLIntf.STransport Constant

Pascal

```
STransport = 'transport';
```

C++

```
STransport = 'transport';
```

File

WSDLIntf

Description

This is constant WSDLIntf.STransport.

190.4.47 WSDLIntf.SType Constant

Pascal

```
SType = 'type';
```

C++

```
SType = 'type';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SType.

190.4.48 WSDLIntf.STypes Constant

Pascal

```
STypes = 'types';
```

C++

```
STypes = 'types';
```

File

WSDLIntf

Description

This is constant WSDLIntf.STypes.

190.4.49 WSDLIntf.SUnknown Constant

Pascal

```
SUnknown = 'Unknown';
```

C++

```
SUnknown = 'Unknown';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SUnknown.

190.4.50 WSDLIntf.SUse Constant

Pascal

```
SUse = 'use';
```

C++

```
SUse = 'use';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SUse.

190.4.51 WSDLIntf.SVerb Constant

Pascal

```
SVerb = 'verb';
```

C++

```
SVerb = 'verb';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SVerb.

190.4.52 WSDLIntf.tns Constant

Pascal

```
tns = 'http:/'+'/tempuri.org/';
```

C++

```
tns = 'http:/'+'/tempuri.org/';
```

File

WSDLIntf

Description

This is constant WSDLIntf.tns.

190.4.53 WSDLIntf.Wsdlns Constant

Pascal

```
Wsdlns = 'http:/'+'/schemas.xmlsoap.org/wsdl/';
```

C++

```
Wsdlns = 'http:/'+'/schemas.xmlsoap.org/wsdl/';
```

File

WSDLIntf

Description

This is constant WSDLIntf.Wsdlns.

190.4.54 WSDLIntf.xsdns Constant

Pascal

```
xsdns = 'http:/'+'/www.w3.org/1999/XMLSchema';
```

C++

```
xsdns = 'http:/'+'/www.w3.org/1999/XMLSchema';
```

File

WSDLIntf

Description

This is constant WSDLIntf.xsdns.

191 WSDLItems Namespace

This is namespace WSDLItems.

Topics

Name	Description
Classes (see page 25951)	The following table lists classes in this documentation.
Interfaces (see page 25989)	The following table lists interfaces in this documentation.
Functions (see page 26000)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 26002)	The following table lists structs, records, enums in this documentation.
Types (see page 26003)	The following table lists types in this documentation.

Classes

	Name	Description
	TWSDLItems (see page 25951)	TWSDLItems represents the definitions in a WSDL document.

Functions

	Name	Description
	HasDefinition (see page 26000)	This is function WSDLItems.HasDefinition.
	HasDefinition (see page 26000)	This is function WSDLItems.HasDefinition.
	NewQualifiedName (see page 26000)	This is function WSDLItems.NewQualifiedName.

Interfaces

	Name	Description
	IHeaderInfo (see page 25989)	This is class WSDLItems.IHeaderInfo.
	IQualifiedName (see page 25994)	IQualifiedName is the interface for obtaining an information about a name in an XML document.
	IWSDLItems (see page 25998)	IWSDLItems provides access to a WSDL document.

Structs, Records, Enums

	Name	Description
	IterateImportOptions (see page 26002)	This is record WSDLItems.IterateImportOptions.

Types

Name	Description
IHeaderInfoArray (see page 26003)	This is type WSDLItems.IHeaderInfoArray.
IQualifiedNameArray (see page 26003)	This is type WSDLItems.IQualifiedNameArray.

TWSDLIterateProc (see page 26003)	This is type WSDLItems.TWSDLIterateProc.

191.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TWSDLItems (see page 25951)	TWSDLItems represents the definitions in a WSDL document.

191.1.1 TWSDLItems Class

TWSDLItems represents the definitions in a WSDL document.

Class Hierarchy

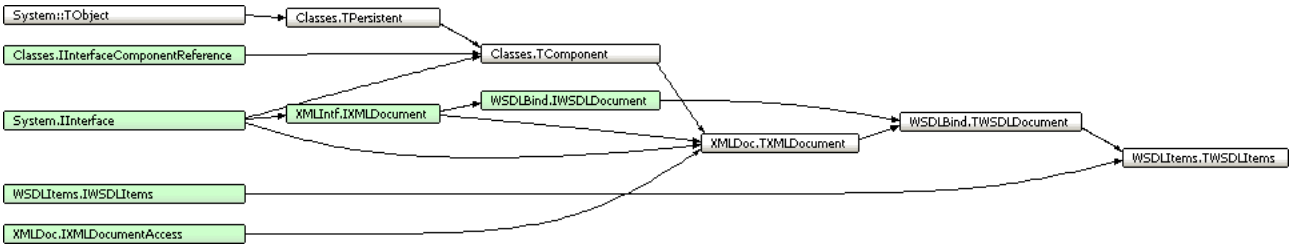

Pascal

```
TWSDLItems = class(TWSDLDocument, IWSDLItems);
```

C++

```
class TWSDLItems : public TWSDLDocument, public IWSDLItems;
```

File

WSDLItems

Description

Use TWSDLItems to represent a WSDL (Web Services Definition Language) document. A WSDL document is a special form of XML document that describes the contents of a Web Service application and how you call the services it implements. TWSDLItems augments its immediate ancestor (TWSDLDocument) by introducing methods that provide easier access to the definitions in the WSDL document.

TWSDLItems can read an existing WSDL document from a file or server, it can be associated with an in-memory string that is the contents of a WSDL document, or it can create a new, empty WSDL document. TWSDLItems uses an external DOM (Document Object Model) parser to analyze the WSDL document. This parser is usually assigned by specifying the DOMVendor property. When you set the Active property to true, TWSDLItems uses the DOM parser to parse the WSDL document so that you can examine or modify it.

When TWSDLItems is created without an Owner, it behaves like an interfaced object. That is, when all references to its interface are released, the TWSDLItems instance is automatically freed. When TWSDLItems is created with an Owner, however, it behaves like any other component, and is freed by its Owner.

See Also

TWSDLView (see page 26013)

191.1.1.1 TWSDLItems Members

The following tables list the members exposed by TWSDLItems.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_AddRef (see page 26106)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	_Release (see page 26106)	Called when an application releases a component interface.
	AssignParseOptions (see page 26108)	This is AssignParseOptions, a member of class TXMLDocument.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckActive (see page 26108)	This is CheckActive, a member of class TXMLDocument.
	CheckAutoSave (see page 26108)	This is CheckAutoSave, a member of class TXMLDocument.
	CheckDOM (see page 26109)	This is CheckDOM, a member of class TXMLDocument.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 26111)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoAfterClose (see page 26111)	This is DoAfterClose, a member of class TXMLDocument.
	DoAfterOpen (see page 26111)	This is DoAfterOpen, a member of class TXMLDocument.
	DoBeforeClose (see page 26112)	This is DoBeforeClose, a member of class TXMLDocument.
	DoBeforeOpen (see page 26112)	This is DoBeforeOpen, a member of class TXMLDocument.
	DoNodeChange (see page 26112)	This is DoNodeChange, a member of class TXMLDocument.
V	GetActive (see page 26113)	Returns the value of the Active property.
	GetAsyncLoadState (see page 26113)	Returns the value of the AsyncLoadState property.
V	GetChildNodeClass (see page 26113)	This is GetChildNodeClass, a member of class TXMLDocument.
	GetChildNodes (see page 26113)	Returns the value of the ChildNodes property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDefinition (see page 25807)	Returns the value of the Definition property.
	GetDocumentElement (see page 26114)	Returns the value of the DocumentElement property.
	GetDocumentNode (see page 26115)	Returns the value of the Node property.
	GetDocumentObject (see page 26115)	This is GetDocumentObject, a member of class TXMLDocument.
	GetDOMDocument (see page 26115)	Returns the value of the DOMDocument property.
	GetDOMPersist (see page 26115)	This is GetDOMPersist, a member of class TXMLDocument.
	GetEncoding (see page 26116)	Returns the value of the Encoding property.
	GetFileName (see page 26116)	Returns the value of the FileName property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetModified (see page 26116)	Returns the value of the Modified property.

	GetName (see page 25967)	Returns the name of the definitions node in the WSDL document.
	GetNodeIndentStr (see page 26117)	Returns the value of the NodeIndentStr property.
	GetOptions (see page 26117)	Returns the value of the Options property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetParseOptions (see page 26117)	Returns the value of the ParseOptions property.
	GetPrologNode (see page 26117)	This is GetPrologNode, a member of class TXMLDocument.
	GetPrologValue (see page 26118)	This is GetPrologValue, a member of class TXMLDocument.
	GetSchemaRef (see page 26118)	Returns the value of the SchemaRef property.
	GetStandAlone (see page 26118)	Returns the value of the StandAlone property.
	GetTargetNamespace (see page 25984)	Returns the value of the TargetNamespace (see page 25988) property.
	GetTargetNamespacePrefix (see page 25985)	Returns the prefix string that represents the target namespace of the WSDL document.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVersion (see page 26118)	Returns the value of the Version property.
	GetXML (see page 26119)	Returns the value of the XML property.
	InternalSetPrologValue (see page 26119)	This is InternalSetPrologValue, a member of class TXMLDocument.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	LoadData (see page 25807)	This is LoadData, a member of class TWSDLDocument.
	LoadData (see page 26120)	This is LoadData, a member of class TXMLDocument.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 26120)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadDOMVendor (see page 26123)	This is ReadDOMVendor, a member of class TXMLDocument.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReleaseDoc (see page 26124)	This is ReleaseDoc, a member of class TXMLDocument.
	SaveToUTF8String (see page 26126)	This is SaveToUTF8String, a member of class TXMLDocument.
	SaveToXMLStrings (see page 26127)	This is SaveToXMLStrings, a member of class TXMLDocument.
V	SetActive (see page 26127)	Sets the value of the Active property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDefinition (see page 25807)	Sets the value of the Definition property.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDocumentElement (see page 26127)	Sets the value of the DocumentElement property.
	SetDOMDocument (see page 26128)	Sets the value of the DOMDocument property.
	SetEncoding (see page 26128)	Sets the value of the Encoding property.
	SetFileName (see page 26128)	Sets the value of the FileName property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetModified (see page 26129)	This is SetModified, a member of class TXMLDocument.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetNodeIndentStr (see page 26129)	Sets the value of the NodeIndentStr property.
	SetOnAsyncLoad (see page 26129)	Supplies an event handler that responds when the AsyncLoadState property changes.
	SetOptions (see page 26130)	Sets the value of the Options property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParseOptions (see page 26130)	Sets the value of the ParseOptions property.
	SetPrologValue (see page 26130)	This is SetPrologValue, a member of class TXMLDocument.
	SetStandAlone (see page 26130)	Sets the value of the StandAlone property.
	SetVersion (see page 26131)	Sets the value of the Version property.

	SetXML (see page 26131)	Sets the value of the XML property.
	SetXMLStrings (see page 26131)	This is SetXMLStrings, a member of class TXMLDocument.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteDOMVendor (see page 26132)	This is WriteDOMVendor, a member of class TXMLDocument.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.
	XMLStringsChanging (see page 26132)	This is XMLStringsChanging, a member of class TXMLDocument.

Protected Properties

	Name	Description
R	DocBindingInfo (see page 26135)	This is DocBindingInfo, a member of class TXMLDocument.
	DocSource (see page 26135)	This is DocSource, a member of class TXMLDocument.
R	DOMParseOptions (see page 26137)	This is DOMParseOptions, a member of class TXMLDocument.
R	DOMPersist (see page 26137)	This is DOMPersist, a member of class TXMLDocument.
	PrefixID (see page 26142)	This is PrefixID, a member of class TXMLDocument.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26107)	This is the overview for the AddChild method overload.
	AddChild (see page 26385)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 25807)	Initializes the TWSDLDocument (see page 25799) object after the last constructor has executed.
V	AfterConstruction (see page 26107)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CompareName (see page 25962)	Indicates whether two node names refer to the same element.
V	Create (see page 25959)	This is the overview for the Create constructor overload.
	Create (see page 26103)	This is the overview for the Create constructor overload.
	CreateElement (see page 26109)	Creates a new element node that is associated with this document.
	CreateElement (see page 26386)	Creates a new element node that is associated with this document.
	CreateNode (see page 26109)	Creates a new node that is associated with this document.
	CreateNode (see page 26387)	Creates a new node that is associated with this document.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.

	Destroy (see page 26104)	Disposes of a TXMLDocument (see page 26097) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GeneratePrefix (see page 26112)	Generates a unique namespace prefix that can be used for a node in the document.
	GeneratePrefix (see page 26388)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 26388)	Returns the value of the Active (see page 26403) property.
	GetAsyncLoadState (see page 26389)	Returns the value of the AsyncLoadState (see page 26404) property.
	GetBindingForServicePort (see page 25962)	Returns the name of the binding associated with a specified service and port.
	GetBindingOfType (see page 25963)	Returns the names of all bindings associated with a specified port type.
	GetBindingOperationNode (see page 25963)	Returns the interface for a specified operation on a specified binding.
	GetBindings (see page 25964)	Fills a WideString list with the names of all bindings defined in the WSDL document.
	GetBindingType (see page 25964)	Returns the port type to which a specified binding applies.
	GetChildNodes (see page 26389)	Returns the value of the ChildNodes (see page 26405) property.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetDefinition (see page 25903)	Returns the value of the Definition (see page 25904) property.
	GetDocBinding (see page 26114)	Replaces the document element with an instance of a specified implementation class.
	GetDocBinding (see page 26389)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 26390)	Returns the value of the DocumentElement (see page 26405) property.
	GetDocumentNode (see page 26390)	Returns the value of the Node (see page 26407) property.
	GetDocumentObject (see page 26229)	This is GetDocumentObject, a member of class IXMLDocumentAccess.
	GetDOMDocument (see page 26390)	Returns the value of the DOMDocument (see page 26405) property.
	GetDOMPersist (see page 26229)	This is GetDOMPersist, a member of class IXMLDocumentAccess.
	GetEncoding (see page 26391)	Returns the value of the Encoding (see page 26406) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFileName (see page 26391)	Returns the value of the FileName (see page 26406) property.
	GetHttpBindingAttribute (see page 25965)	This is the overview for the GetHttpBindingAttribute method overload.
	GetImports (see page 25965)	Fills a WideString list with the namespaces of all documents imported by this one.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetLocationForImport (see page 25966)	Fills a string list with the names of all messages that must be passed in or out when calling a specified operation.
	GetMessageNode (see page 25966)	Returns the interface for the node that describes a message.
	GetMessages (see page 25967)	Fills a WideString list with the names of all messages defined in the WSDL document.
	GetModified (see page 26391)	Returns the value of the Modified (see page 26407) property.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNodeIndentStr (see page 26392)	Returns the value of the NodeIndentStr (see page 26408) property.
	GetOperationNode (see page 25967)	Returns the interface for the node that describes an operation.

	GetOperations (see page 25968)	Fills a WideString list with the names of all operations defined for a specified port type.
	GetOperationsForBinding (see page 25968)	This is GetOperationsForBinding, a member of class TWSDLItems.
	GetOptions (see page 26392)	Returns the value of the Options (see page 26408) property.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParseOptions (see page 26392)	Returns the value of the ParseOptions (see page 26409) property.
	GetPartNode (see page 25969)	Returns the interface for the node that describes a part of a message.
	GetParts (see page 25969)	Fills a list with the names of all parts defined for a specified message.
	GetPartsForOperation (see page 25970)	Fills a list with the names of all parts defined for input messages of a specified operation.
	GetPortsForService (see page 25970)	Fills a list with the names of all ports defined within a specified service.
	GetPortTypeNode (see page 25971)	Returns the interface for the node that describes a port type defined in the WSDL document.
	GetPortTypes (see page 25971)	Fills a list with the names of all port types defined in the WSDL document.
	GetSchemaNode (see page 25972)	Returns the interface for the schema node that defines a type.
	GetSchemaRef (see page 26392)	Returns the value of the SchemaRef (see page 26409) property.
	GetSchemas (see page 25972)	Fills a list with the namespaces for all schema definitions of types in the WSDL document.
	GetServiceAndPortOfBinding (see page 25973)	Retrieves the names of the service and port associated with a specified binding.
	GetServiceNode (see page 25973)	Returns the interface for the node that describes a service defined in the WSDL document.
	GetServices (see page 25974)	Fills a list with the names of all services defined in the WSDL document.
	GetSoapAction (see page 25974)	Returns the value of the soapAction attribute of a specified operation on a specified binding.
	GetSoapActionList (see page 25975)	Fills a list with the soapAction attributes for all operations on a specified SOAP binding.
	GetSoapAddressForServicePort (see page 25975)	Returns the address of a specified port from a specified service.
	GetSoapBindingAttribute (see page 25976)	Returns the value of a specified attribute of a specified SOAP binding.
	GetSoapBindingInputNode (see page 25976)	Returns the interface for the input node on a specified binding and operation.
	GetSoapBindingIONode (see page 25977)	Returns the interface for a node on a specified binding and operation.
	GetSoapBindingOutputNode (see page 25978)	Returns the interface for the output node on a specified binding and operation.
	GetSoapBodyAttribute (see page 25978)	Returns the value of an attribute of a body tag of a message type in an operation on the port type of a SOAP binding.
	GetSoapBodyInputAttribute (see page 25979)	Returns the value of an attribute of a body tag of an input message in an operation on the port type of a SOAP binding.
	GetSoapBodyNamespace (see page 25980)	Returns the value of the first namespace attribute found on a SOAP body node inside a binding with a specified port type.
	GetSoapBodyOutputAttribute (see page 25980)	Returns the value of an attribute of a body tag of an output message in an operation on the port type of a SOAP binding.
	GetSoapHeaders (see page 25981)	Returns an interface for accessing a specified set of header nodes.
	GetSoapInputHeaders (see page 25982)	Returns an interface for accessing the header nodes that belong to the input node of a specified binding and operation.
	GetSoapOperationAttribute (see page 25982)	Returns the value of an attribute on the <soap:operation> node of an operation on a SOAP binding.
	GetSoapOperationStyle (see page 25983)	Returns the value of the style attribute of a specified operation on a specified binding.
	GetSoapOutputHeaders (see page 25983)	Returns an interface for accessing the header nodes that belong to the output node of a specified binding and operation.
	GetStandAlone (see page 26393)	Returns the value of the StandAlone (see page 26410) property.
	GetVersion (see page 26393)	Returns the value of the Version (see page 26410) property.
	GetWSDLDoc (see page 25985)	Returns the interface for this TWSDLItems (see page 25951) instance.
	GetWSDLItems (see page 25985)	Returns a reference to this TWSDLItems (see page 25951) instance.
	GetXML (see page 26393)	Returns the value of the XML (see page 26410) property.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasTypesNode (see page 25986)	Indicates whether the WSDL document includes any type definitions.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.

	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmptyDoc (see page 26119)	Indicates whether the TXMLDocument (see page 26097) instance represents an empty document.
	IsEmptyDoc (see page 26394)	Indicates whether the IXMLDocument (see page 26382) instance represents an empty document.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	IsPortTypeHTTPBound (see page 25986)	Indicates whether a specified port type has an HTTP binding.
	IterateImports (see page 25986)	Executes a callback for every WSDL document that this one imports.
	Load (see page 25987)	Loads a WSDL document and activates it.
	LoadFromFile (see page 26121)	Loads an XML (see page 26144) document and activates it.
	LoadFromFile (see page 26394)	Loads an XML (see page 26410) document from disk.
	LoadFromStream (see page 26121)	Loads an XML (see page 26144) document from a stream and activates it.
	LoadFromStream (see page 26395)	Loads an XML (see page 26410) document from a stream and activates it.
	LoadFromXML (see page 26122)	This is the overview for the LoadFromXML method overload.
	LoadFromXML (see page 26395)	This is the overview for the LoadFromXML method overload.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	NewInstance (see page 26122)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 26123)	Updates the parsed XML (see page 26144) document to reflect external changes.
	Refresh (see page 26396)	Updates the parsed XML (see page 26410) document to reflect external changes.
	RegisterDocBinding (see page 26124)	Registers an implementation class for child nodes with a specified tag name.
	RegisterDocBinding (see page 26396)	Registers an implementation class for child nodes with a specified tag name.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Resync (see page 26125)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	Resync (see page 26397)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 26125)	Saves the XML (see page 26144) document to disk.
	SaveToFile (see page 26397)	Saves the XML (see page 26410) document to disk.
	SaveToStream (see page 26125)	Saves the XML (see page 26144) document to a stream.
	SaveToStream (see page 26398)	Saves the XML (see page 26410) document to a stream.
	SaveToXML (see page 26126)	This is the overview for the SaveToXML method overload.
	SaveToXML (see page 26398)	This is the overview for the SaveToXML method overload.
	SetActive (see page 26399)	Sets the value of the Active (see page 26403) property.
	SetDefinition (see page 25903)	Sets the value of the Definition (see page 25904) property.
	SetDocumentElement (see page 26399)	Sets the value of the DocumentElement (see page 26405) property.
	SetDOMDocument (see page 26399)	Sets the value of the DOMDocument (see page 26405) property.
	SetEncoding (see page 26400)	Sets the value of the Encoding (see page 26406) property.
	SetFileName (see page 26400)	Sets the value of the FileName (see page 26406) property.
	SetNodeIndentStr (see page 26400)	Sets the value of the NodeIndentStr (see page 26408) property.
	SetOnAsyncLoad (see page 26401)	Supplies an event handler that responds when the AsyncLoadState (see page 26404) property changes.
	SetOptions (see page 26401)	Sets the value of the Options (see page 26408) property.
	SetParseOptions (see page 26402)	Sets the value of the ParseOptions (see page 26409) property.
	SetStandAlone (see page 26402)	Sets the value of the StandAlone (see page 26410) property.

	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetVersion (see page 26402)	Sets the value of the Version (see page 26410) property.
	SetXML (see page 26403)	Sets the value of the XML (see page 26410) property.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Active (see page 26403)	Specifies whether the XML (see page 26410) document has been parsed so that the DOM interfaces are available to examine or change its contents.
R	AsyncLoadState (see page 26134)	Indicates the state of the DOM parser when it parses the XML (see page 26144) document asynchronously.
R	AsyncLoadState (see page 26404)	Indicates the state of the DOM parser when it parses the XML (see page 26410) document asynchronously.
R	ChildNodes (see page 26135)	Lists all of the child nodes of the document.
R	ChildNodes (see page 26405)	Lists all of the child nodes of the document.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	Definition (see page 25904)	Provides access to the definitions contained in the WSDL document.
	Definition (see page 25808)	Provides access to the definitions contained in the WSDL document.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DocumentElement (see page 26135)	Provides access to the root node of the XML (see page 26144) document.
	DocumentElement (see page 26405)	Provides access to the root node of the XML (see page 26410) document.
R	DocumentObject (see page 26229)	This is DocumentObject, a member of class IXMLDocumentAccess.
	DOMDocument (see page 26136)	Provides access to the DOM interface for the parsed XML (see page 26144) document.
	DOMDocument (see page 26405)	Provides access to the DOM interface for the parsed XML (see page 26410) document.
	DOMImplementation (see page 26137)	This is DOMImplementation, a member of class TXMLDocument.
R	DOMPersist (see page 26230)	This is DOMPersist, a member of class IXMLDocumentAccess.
	Encoding (see page 26138)	Indicates the character set that the XML (see page 26144) document uses to encode characters.
	Encoding (see page 26406)	Indicates the character set that the XML (see page 26410) document uses to encode characters.
	FileName (see page 26406)	Indicates the XML (see page 26410) document file, if any, that this interface represents.
R	Modified (see page 26139)	Indicates whether the XML (see page 26144) document has been modified.
R	Modified (see page 26407)	Indicates whether the XML (see page 26410) document has been modified.
R	Node (see page 26140)	Provides access to the document node for the XML (see page 26144) document.
R	Node (see page 26407)	Provides access to the document node for the XML (see page 26410) document.
	NodeIndentStr (see page 26408)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 26410) text.
	NSPrefixBase (see page 26141)	Specifies the base string for automatically-generated namespace prefixes.
	Options (see page 26408)	Influences various aspects of the document's behavior.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParseOptions (see page 26409)	Specifies implementation-specific options that are followed when parsing the XML (see page 26410) document.
R	SchemaRef (see page 26142)	Returns the name of the schema associated with this XML (see page 26144) document.
R	SchemaRef (see page 26409)	Returns the name of the schema associated with this XML (see page 26410) document.

	StandAlone (see page 26143)	Indicates whether the XML (see page 26144) document includes any external markup declarations.
	StandAlone (see page 26410)	Indicates whether the XML (see page 26410) document includes any external markup declarations.
R	StreamLoader (see page 25987)	Provides access to an interface for loading the WSDL document.
R	TargetNamespace (see page 25988)	Provides access to an interface for loading the WSDL document.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Version (see page 26143)	Indicates the XML (see page 26144) version that the document uses.
	Version (see page 26410)	Indicates the XML (see page 26410) version that the document uses.
	XML (see page 26410)	Specifies the content of the XML document associated with this IXMLDocument (see page 26382) interface.

Published Events

	Name	Description
	AfterClose (see page 26144)	Occurs after the XML (see page 26144) document is closed.
	AfterNodeChange (see page 26145)	Occurs after a change is made to one of the nodes in the document.
	AfterOpen (see page 26145)	Occurs immediately after the XML (see page 26144) document is opened.
	BeforeClose (see page 26146)	Occurs immediately before the XML (see page 26144) document is closed.
	BeforeNodeChange (see page 26146)	Occurs before a change is made to one of the nodes in the document.
	BeforeOpen (see page 26147)	Occurs immediately before the XML (see page 26144) document is opened.
	OnAsyncLoad (see page 26147)	Occurs at various stages during the asynchronous parsing of an XML (see page 26144) document.

Published Properties

	Name	Description
	Active (see page 26133)	Specifies whether the XML (see page 26144) document has been parsed so that the DOM interfaces are available to examine or change its contents.
	DOMVendor (see page 26137)	Specifies the DOM implementation to use for parsing and manipulating the XML (see page 26144) document.
	FileName (see page 26138)	Specifies the file name for the XML (see page 26144) document associated with this TXMLDocument (see page 26097) object.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NodeIndentStr (see page 26140)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 26144) text.
	Options (see page 26141)	Influences various aspects of the document's behavior.
	ParseOptions (see page 26142)	Specifies implementation-specific options that are followed when parsing the XML (see page 26144) document.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XML (see page 26144)	Specifies the content of the XML document associated with this TXMLDocument (see page 26097) object.

191.1.1.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
V	TWSDLItems.Create (TComponent) (see page 25960)	Creates and initializes a TWSDLItems (see page 25951) object.
V	TWSDLItems.Create (TWSDLItems, IStreamLoader) (see page 25960)	Creates and initializes a TWSDLItems (see page 25951) object.

191.1.1.2.1 TWSDLItems.Create Constructor (TComponent)

Creates and initializes a TWSDLItems (see page 25951) object.

Pascal

```
constructor Create(AOwner: TComponent); override; overload;
constructor Create(const WSDLItems: TWSDLItems; const StreamLoader: IStreamLoader); virtual;
overload;
```

C++

```
virtual __fastcall TWSDLItems(TComponent * AOwner);
virtual __fastcall TWSDLItems(const TWSDLItems WSDLItems, const IStreamLoader StreamLoader);
```

Description

Call Create to instantiate a TWSDLItems (see page 25951) component at runtime. Typically, TWSDLItems (see page 25951) instances are created by other components, such as THTTPRIO, that use them.

AOwner is another component, typically a form or data module, that is responsible for freeing the TWSDLItems (see page 25951) instance. It becomes the value of the Owner property.

WSDLItems (see page 25949) is another TWSDLItems (see page 25951) object that the new instance can use to extract a stream loader if none is specified using the StreamLoader (see page 25987) property,

StreamLoader (see page 25987) is an interface to use for loading the WSDL document from a server. It includes properties for specifying the user name, password, and proxy server needed to connect to the server that hosts the WSDL document.

If you do not specify a stream loader, the TWSDLItems (see page 25951) constructor creates one as follows: Using the first syntax, it creates a default stream loader object. Using the second syntax, it uses the stream loader from the TWSDLItems (see page 25951) instance that is the first parameter. Thus, when using the second syntax, you must supply either the WSDLItems (see page 25949) or StreamLoader (see page 25987) argument.

See Also

THTTPRio

Owner (see page 2807)

191.1.1.3 TWSDLItems Methods

The methods of the TWSDLItems class are listed here.

Protected Methods

	Name	Description
	GetName (see page 25967)	Returns the name of the definitions node in the WSDL document.
	GetTargetNamespace (see page 25984)	Returns the value of the TargetNamespace (see page 25988) property.
	GetTargetNamespacePrefix (see page 25985)	Returns the prefix string that represents the target namespace of the WSDL document.

Public Methods

	Name	Description
	CompareName (see page 25962)	Indicates whether two node names refer to the same element.
	GetBindingForServicePort (see page 25962)	Returns the name of the binding associated with a specified service and port.
	GetBindingOfType (see page 25963)	Returns the names of all bindings associated with a specified port type.
	GetBindingOperationNode (see page 25963)	Returns the interface for a specified operation on a specified binding.
	GetBindings (see page 25964)	Fills a WideString list with the names of all bindings defined in the WSDL document.

	GetBindingType (see page 25964)	Returns the port type to which a specified binding applies.
	GetHttpBindingAttribute (see page 25965)	This is the overview for the GetHttpBindingAttribute method overload.
	GetImports (see page 25965)	Fills a WideString list with the namespaces of all documents imported by this one.
	GetLocationForImport (see page 25966)	Fills a string list with the names of all messages that must be passed in or out when calling a specified operation.
	GetMessageNode (see page 25966)	Returns the interface for the node that describes a message.
	GetMessages (see page 25967)	Fills a WideString list with the names of all messages defined in the WSDL document.
	GetOperationNode (see page 25967)	Returns the interface for the node that describes an operation.
	GetOperations (see page 25968)	Fills a WideString list with the names of all operations defined for a specified port type.
	GetOperationsForBinding (see page 25968)	This is GetOperationsForBinding, a member of class TWSDLItems.
	GetPartNode (see page 25969)	Returns the interface for the node that describes a part of a message.
	GetParts (see page 25969)	Fills a list with the names of all parts defined for a specified message.
	GetPartsForOperation (see page 25970)	Fills a list with the names of all parts defined for input messages of a specified operation.
	GetPortsForService (see page 25970)	Fills a list with the names of all ports defined within a specified service.
	GetPortTypeNode (see page 25971)	Returns the interface for the node that describes a port type defined in the WSDL document.
	GetPortTypes (see page 25971)	Fills a list with the names of all port types defined in the WSDL document.
	GetSchemaNode (see page 25972)	Returns the interface for the schema node that defines a type.
	GetSchemas (see page 25972)	Fills a list with the namespaces for all schema definitions of types in the WSDL document.
	GetServiceAndPortOfBinding (see page 25973)	Retrieves the names of the service and port associated with a specified binding.
	GetServiceNode (see page 25973)	Returns the interface for the node that describes a service defined in the WSDL document.
	GetServices (see page 25974)	Fills a list with the names of all services defined in the WSDL document.
	GetSoapAction (see page 25974)	Returns the value of the soapAction attribute of a specified operation on a specified binding.
	GetSoapActionList (see page 25975)	Fills a list with the soapAction attributes for all operations on a specified SOAP binding.
	GetSoapAddressForServicePort (see page 25975)	Returns the address of a specified port from a specified service.
	GetSoapBindingAttribute (see page 25976)	Returns the value of a specified attribute of a specified SOAP binding.
	GetSoapBindingInputNode (see page 25976)	Returns the interface for the input node on a specified binding and operation.
	GetSoapBindingIONode (see page 25977)	Returns the interface for a node on a specified binding and operation.
	GetSoapBindingOutputNode (see page 25978)	Returns the interface for the output node on a specified binding and operation.
	GetSoapBodyAttribute (see page 25978)	Returns the value of an attribute of a body tag of a message type in an operation on the port type of a SOAP binding.
	GetSoapBodyInputAttribute (see page 25979)	Returns the value of an attribute of a body tag of an input message in an operation on the port type of a SOAP binding.
	GetSoapBodyNamespace (see page 25980)	Returns the value of the first namespace attribute found on a SOAP body node inside a binding with a specified port type.
	GetSoapBodyOutputAttribute (see page 25980)	Returns the value of an attribute of a body tag of an output message in an operation on the port type of a SOAP binding.
	GetSoapHeaders (see page 25981)	Returns an interface for accessing a specified set of header nodes.
	GetSoapInputHeaders (see page 25982)	Returns an interface for accessing the header nodes that belong to the input node of a specified binding and operation.
	GetSoapOperationAttribute (see page 25982)	Returns the value of an attribute on the <soap:operation> node of an operation on a SOAP binding.
	GetSoapOperationStyle (see page 25983)	Returns the value of the style attribute of a specified operation on a specified binding.
	GetSoapOutputHeaders (see page 25983)	Returns an interface for accessing the header nodes that belong to the output node of a specified binding and operation.
	GetWSDLDoc (see page 25985)	Returns the interface for this TWSDLItems (see page 25951) instance.
	GetWSDLItems (see page 25985)	Returns a reference to this TWSDLItems (see page 25951) instance.
	HasTypesNode (see page 25986)	Indicates whether the WSDL document includes any type definitions.
	IsPortTypeHTTPBound (see page 25986)	Indicates whether a specified port type has an HTTP binding.
	IterateImports (see page 25986)	Executes a callback for every WSDL document that this one imports.
	Load (see page 25987)	Loads a WSDL document and activates it.

191.1.1.3.1 TWSDLItems.CompareName Method

Indicates whether two node names refer to the same element.

Pascal

```
function CompareName(const NodeName: WideString; const OtherName: WideString; const TNS:
WideString = ''): boolean;
```

C++

```
__fastcall boolean CompareName(const BSTR NodeName, const BSTR OtherName, const BSTR TNS = '');
```

Description

Call CompareName to compare the names of two nodes to determine whether they match. CompareName compares the node names, taking into account any namespace prefix added to the node names. CompareName uses the namespace prefix to look up the corresponding namespace URI. If either node name does not have a namespace prefix, CompareName uses the TargetNamespace (see page 25988) property for the first node name, and the supplied namespace for the second. CompareName can find a match between two node names that have different namespace prefixes, as long as both prefixes map to the same namespace URI.

NodeName and OtherName are the two node names to compare.

TNS is the target namespace for OtherName. If this is a nonempty string, it is used to replace the namespace of OtherName. If it is an empty string, OtherName uses the namespace associated with its namespace prefix, or the TargetNamespace (see page 25988) property if it has no namespace prefix.

CompareName returns true if the two names match, false if they differ (either in the local name, or in the namespace to which the names belong).

See Also

NodeName (see page 26435)

TargetNamespace (see page 25988)

191.1.1.3.2 TWSDLItems.GetBindingForServicePort Method

Returns the name of the binding associated with a specified service and port.

Pascal

```
function GetBindingForServicePort(const ServiceName: WideString; const PortName: WideString):
IQualifiedName;
```

C++

```
__fastcall IQualifiedName GetBindingForServicePort(const BSTR ServiceName, const BSTR
PortName);
```

Description

Call GetBindingForServicePort to retrieve the name of a binding that is associated with the specified service and port. The binding is defined in the WSDL document or another document that it imports. A binding identifies an encoding format (e.g. soap:rpc) and communications protocol (eg. HTML) that is available for a specific port type (interface).

ServiceName is the name of the service to which the desired binding belongs.

PortName is the name of a port within the specified service that uses the binding.

See Also

GetBindings (see page 25964)

GetBindingType (see page 25964)

GetPortsForService (see page 25970)

GetServices (see page 25974)

GetBindingOfType (see page 25963)

GetServiceAndPortOfBinding (see page 25973)

191.1.1.3.3 TWSDLItems.GetBindingOfType Method

Returns the names of all bindings associated with a specified port type.

Pascal

```
function GetBindingOfType(const BindingTypeName: WideString; const TNS: WideString = ''):
IQualifiedNameArray; overload;
```

C++

```
__fastcall IQualifiedNameArray GetBindingOfType(const BSTR BindingTypeName, const BSTR TNS =
'');
```

Description

Call GetBindingOfType to retrieve the names of the bindings that are associated with the specified port type. Each bindings is defined in the WSDL document or another document that it imports. A binding identifies an encoding format (e.g. soap:rpc) and communications protocol (eg. HTML) that is available for a specific port type (interface).

BindingTypeName is the type of the binding. This is the name of a port type for which the binding describes an encoding format and communications protocol.

TNS is the namespace in which BindingTypeName is defined. If this parameter is omitted, GetBindingOfType assumes the value of the TargetNamespace (see page 25988) property.

GetBindingOfType returns an array of qualified names for each binding defined for the specified type.

See Also

GetBindings (see page 25964)

GetPortTypes (see page 25971)

GetBindingType (see page 25964)

TargetNamespace (see page 25988)

191.1.1.3.4 TWSDLItems.GetBindingOperationNode Method

Returns the interface for a specified operation on a specified binding.

Pascal

```
function GetBindingOperationNode(const BindingName: WideString; const Operation: WideString;
OverloadIndex: Integer): IBindingOperation;
```

C++

```
__fastcall IBindingOperation GetBindingOperationNode(const BSTR BindingName, const BSTR
Operation, int OverloadIndex);
```

Description

Call GetBindingOperationNode to retrieve the operation interface for a SOAP binding. GetBindingOperationNode locates a binding with the specified name, and searches its child nodes for an operation node with the specified name. If there is more

than one operation with the specified name (that is, if the method is overloaded), GetBindingOperationNode locates the desired overload based on its position.

BindingName is the name of the binding. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that binding node. Operations correspond to methods on an invokable interface.

OverloadIndex indicates which operation node to choose if there are overloaded methods. 0 indicates the first operation node with the specified name, 1 indicates the second operation node with that name, and so on.

GetBindingOperationNode returns an IBindingOperation interface, which can be used to access the name, inputs, outputs, and faults associated with the operation. If GetBindingOperationNode is unable to locate an operation node for the specified binding and operation names, it returns nil (Delphi) or NULL (C++).

See Also

GetOperationNode (see page 25967)

191.1.1.3.5 TWSDLItems.GetBindings Method

Fills a WideString list with the names of all bindings defined in the WSDL document.

Pascal

```
procedure GetBindings(BindingNames: TWideStrings; QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetBindings(TWideStrings BindingNames, Boolean QualifiedNames = False);
```

Description

Call GetBindings to retrieve the names of all bindings defined in the WSDL document. A binding identifies an encoding format (e.g. soap:rpc) and communications protocol (eg. HTML) that is available for a specific port type (interface).

BindingNames is an existing TWideStrings object to which the names of any defined bindings are added.

QualifiedNames indicates whether the binding names should be qualified using the target namespace prefix.

Note: GetBindings does not clear the WideString list before adding the names of all defined bindings.

See Also

GetBindingForServicePort (see page 25962)

GetBindingType (see page 25964)

GetPortTypes (see page 25971)

191.1.1.3.6 TWSDLItems.GetBindingType Method

Returns the port type to which a specified binding applies.

Pascal

```
function GetBindingType(const BindingName: WideString): WideString;
```

C++

```
__fastcall BSTR GetBindingType(const BSTR BindingName);
```

Description

Call GetBindingType to retrieve the names of the port type to which a specified binding applies.

BindingName is the name of a binding defined in the WSDL document.

See Also

GetBindingForServicePort (see page 25962)

GetBindings (see page 25964)

GetPortTypes (see page 25971)

GetBindingOfType (see page 25963)

191.1.1.3.7 GetHttpBindingAttribute Method

This is the overview for the GetHttpBindingAttribute method overload.

Overload List

	Name	Description
	TWSDLItems.GetHttpBindingAttribute (IQualifiedName, WideString) (see page 25965)	Returns the value of a specified attribute of a binding's <http:binding> node.
	TWSDLItems.GetHttpBindingAttribute (WideString, WideString) (see page 25965)	Returns the value of a specified attribute of a binding's <http:binding> node.

191.1.1.3.7.1 TWSDLItems.GetHttpBindingAttribute Method (IQualifiedName, WideString)

Returns the value of a specified attribute of a binding's <http:binding> node.

Pascal

```
function GetHttpBindingAttribute(const BindingName: WideString; Attribute: WideString):
WideString; overload;
function GetHttpBindingAttribute(const QualifiedName: IQualifiedName; Attribute: WideString):
WideString; overload;
```

C++

```
__fastcall BSTR GetHttpBindingAttribute(const BSTR BindingName, BSTR Attribute);
__fastcall BSTR GetHttpBindingAttribute(const IQualifiedName QualifiedName, BSTR Attribute);
```

Description

Call GetHttpBindingAttribute to retrieve value of an attribute on the http binding tag of a specified binding. The http binding tag can come from the WSDL document or another document that it imports.

BindingName or QualifiedName is the name of the binding that includes the <http:binding> tag.

Attribute is the name of the desired attribute on the <http:binding> tag.

See Also

GetSoapBindingAttribute (see page 25976)

GetSoapFaultAttribute

GetSoapHeaderAttribute

GetSoapOperationAttribute (see page 25982)

GetSoapBodyAttribute (see page 25978)

191.1.1.3.8 TWSDLItems.GetImports Method

Fills a WideString list with the namespaces of all documents imported by this one.

Pascal

```
procedure GetImports(ImportNames: TWideStrings);
```

C++

```
__fastcall GetImports(TWideStrings ImportNames);
```

Description

Call GetImports to retrieve the namespaces for of all documents that are imported into this one using an <import> tag. The location of these imported documents can be retrieved from the namespace using the GetLocationForImport (see page 25966) method.

ImportNames is an existing TWideStrings object to which the namespaces for any imports are added.

Note: GetImports does not clear the WideString list before adding the namespaces for all imported documents.

See Also

GetLocationForImport (see page 25966)

191.1.1.3.9 TWSDLItems.GetLocationForImport Method

Fills a string list with the names of all messages that must be passed in or out when calling a specified operation.

Pascal

```
function GetLocationForImport(const ImportNameSpace: WideString): WideString;
```

C++

```
__fastcall BSTR GetLocationForImport(const BSTR ImportNameSpace);
```

Description

Call GetLocationForImport to retrieve the URI that indicates the location of an imported document, given its namespace.

ImportNameSpace is the namespace that identifies another document that is imported into this WSDL document. It is one of the strings retrieved by the GetImports (see page 25965) method.

See Also

GetImports (see page 25965)

191.1.1.3.10 TWSDLItems.GetMessageNode Method

Returns the interface for the node that describes a message.

Pascal

```
function GetMessageNode(const MessageName: WideString): IXMLNode;
```

C++

```
__fastcall IXMLNode GetMessageNode(const BSTR MessageName);
```

Description

Call GetMessageNode to obtain information about a message defined in the WSDL document. A message describes a single item of data that is passed in or out when executing an operation (method). The parameters and return value of a method each map onto a message.

MessageName is the name of the message whose node is returned.

See Also

GetMessages (see page 25967)

GetParts (see page 25969)

191.1.1.3.11 TWSDLItems.GetMessages Method

Fills a WideString list with the names of all messages defined in the WSDL document.

Pascal

```
procedure GetMessages(MessageNames: TWideStrings; QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetMessages(TWideStrings MessageNames, Boolean QualifiedNames = False);
```

Description

Call GetMessages to retrieve the names of all messages defined in the WSDL document. Each message describes a single item of data that is passed in or out when executing an operation (method). The parameters and return value of a method each map onto a message.

MessageNames is an existing TWideStrings object to which the names of all messages are added.

QualifiedNames indicates whether the message names added to the list should be qualified using the target namespace prefix.

Note: GetMessages does not clear the WideString list before adding the message names defined in the WSDL document.

See Also

GetMessageNode (see page 25966)

GetParts (see page 25969)

191.1.1.3.12 TWSDLItems.GetName Method

Returns the name of the definitions node in the WSDL document.

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

The root node of a WSDL document is always a definitions node. This node gives a name to the set of definitions in the WSDL document. Call GetName to retrieve that name.

See Also

Definition (see page 25808)

GetTargetNamespace (see page 25984)

191.1.1.3.13 TWSDLItems.GetOperationNode Method

Returns the interface for the node that describes an operation.

Pascal

```
function GetOperationNode(const PortTypeName: WideString; const OperationName: WideString):
IXMLNode;
```

C++

```
__fastcall IXMLNode GetOperationNode(const BSTR PortTypeName, const BSTR OperationName);
```

Description

Call GetOperationNode to obtain information about an operation defined in the WSDL document. An operation corresponds to a method on the interface that corresponds to a specified port type.

PortTypeName is the name of the port type that contains the specified operation.

OperationName is the name of the operation whose node is returned.

See Also

GetOperations (see page 25968)

getoperationsforbinding (see page 25968)

GetPortTypes (see page 25971)

191.1.1.3.14 TWSDLItems.GetOperations Method

Fills a WideString list with the names of all operations defined for a specified port type.

Pascal

```
procedure GetOperations(const PortTypeName: WideString; OperationNames: TWideStrings;
QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetOperations(const BSTR PortTypeName, TWideStrings OperationNames, Boolean
QualifiedNames = False);
```

Description

Call GetOperations to retrieve the names of all operations defined in for the port type specified by PortTypeName. An operation corresponds to a method on the interface that corresponds to the specified port type.

PortTypeName is the name of the port type whose operations are retrieved.

OperationNames is an existing TWideStrings object to which the names of all operations on PortTypeName are added.

QualifiedNames indicates whether the operation names added to the list should be qualified using the target namespace prefix.

Note: GetOperations does not clear the WideString list before adding the names of the operations on the specified port type.

See Also

GetOperationNode (see page 25967)

getoperationsforbinding (see page 25968)

GetPortTypes (see page 25971)

191.1.1.3.15 TWSDLItems.GetOperationsForBinding Method

Pascal

```
procedure GetOperationsForBinding(const BindingName: WideString; OperationNames: TWideStrings;
QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetOperationsForBinding(const BSTR BindingName, TWideStrings OperationNames,
Boolean QualifiedNames = False);
```

Description

This is GetOperationsForBinding, a member of class TWSDLItems.

191.1.1.3.16 TWSDLItems.GetPartNode Method

Returns the interface for the node that describes a part of a message.

Pascal

```
function GetPartNode(const MessageName: WideString; const PartName: WideString): IXMLNode;
```

C++

```
__fastcall IXMLNode GetPartNode(const BSTR MessageName, const BSTR PartName);
```

Description

Call GetPartNode to obtain information about a part that is defined for a message. Every message consists of one or more parts, each of which is associated with a definition from some type system.

MessageName is the name of the message that is defined using the part.

PartName is the name of the part whose node is returned.

See Also

GetParts (see page 25969)

GetMessages (see page 25967)

GetPartsForOperation (see page 25970)

191.1.1.3.17 TWSDLItems.GetParts Method

Fills a list with the names of all parts defined for a specified message.

Pascal

```
procedure GetParts(const MessageName: WideString; PartNames: TWideStrings; QualifiedNames:
Boolean = False);
```

C++

```
__fastcall GetParts(const BSTR MessageName, TWideStrings PartNames, Boolean QualifiedNames =
False);
```

Description

Call GetParts to retrieve the names of all parts that make up a specified message. A part corresponds to definition from some type system.

MessageName is the name of the message whose parts are retrieved.

PartNames is an existing TWideStrings object to which the names of all parts on MessageName are added.

QualifiedNames indicates whether the part names added to the list should be qualified using the target namespace prefix.

Note: GetParts does not clear the WideString list before adding the names of the parts of the specified message.

See Also

GetPartNode (see page 25969)

GetMessages (see page 25967)

GetPartsForOperation (see page 25970)

191.1.1.3.18 TWSDLItems.GetPartsForOperation Method

Fills a list with the names of all parts defined for input messages of a specified operation.

Pascal

```
function GetPartsForOperation(const PortTypeName: WideString; const OperationName: WideString;
OperationIndex: Integer; PartNames: TWideStrings): Boolean;
```

C++

```
__fastcall Boolean GetPartsForOperation(const BSTR PortTypeName, const BSTR OperationName, int
OperationIndex, TWideStrings PartNames);
```

Description

Call GetPartsForOperation to retrieve the names of all parts that make up input messages on a specified operation. The part definition must be in the WSDL document or a document that it imports. A part corresponds to definition from some type system.

PortTypeName is the name of the port type that contains the operation. A port type corresponds to an invokable interface in a Web Service application.

OperationName is the name of the operation from whose input messages the parts are retrieved. An operation corresponds to a method on the interface that corresponds to PortTypeName.

PartNames is an existing TWideStrings object to which the names of all parts on input messages (input parameters) of the operation specified by OperationName.

GetPartsForOperation returns true if it adds any part names to PartNames, false if there are no parts to add.

Note: GetPartsForOperation does not clear the WideString list before adding the names of the parts of the specified message.

See Also

GetPartNode (see page 25969)

GetParts (see page 25969)

GetOperations (see page 25968)

191.1.1.3.19 TWSDLItems.GetPortsForService Method

Fills a list with the names of all ports defined within a specified service.

Pascal

```
procedure GetPortsForService(const ServiceName: WideString; PortNames: TWideStrings;
SkipHttpBindings: Boolean = True; QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetPortsForService(const BSTR ServiceName, TWideStrings PortNames, Boolean
SkipHttpBindings = True, Boolean QualifiedNames = False);
```

Description

Call GetPortsForService to retrieve the names of all ports defined for the specified service. A port represents a binding, plus the

address where that binding can be found.

ServiceName is the name of the service whose ports are listed.

PortNames is an existing TWideStrings object to which the names of all ports in the specified service are added.

SkipHttpBindings indicates whether GetPortsForService should include port types whose definitions include an <http:binding> tag. When false, GetPortsForService retrieves all port type definitions. When true, GetPortsForService ignores port type definitions with simple HTTP bindings.

QualifiedNames indicates whether the port names added to the list should be qualified using the target namespace prefix.

Note: GetPortsForService does not clear the WideString list before adding the names of the ports.

See Also

GetBindingForServicePort (see page 25962)

GetServices (see page 25974)

191.1.1.3.20 TWSDLItems.GetPortTypeNode Method

Returns the interface for the node that describes a port type defined in the WSDL document.

Pascal

```
function GetPortTypeNode(const PortTypeName: WideString): IXMLNode;
```

C++

```
__fastcall IXMLNode GetPortTypeNode(const BSTR PortTypeName);
```

Description

Call GetPortTypeNode to obtain information about a port type that is defined in the WSDL document or a document that it imports. Each port type corresponds to an interface that the Web Service application can execute.

PortTypeName is the name of the port type whose node is returned.

See Also

GetPortTypes (see page 25971)

191.1.1.3.21 TWSDLItems.GetPortTypes Method

Fills a list with the names of all port types defined in the WSDL document.

Pascal

```
procedure GetPortTypes(PortTypeNames: TWideStrings; SkipHttpBindings: Boolean = True;
QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetPortTypes(TWideStrings PortTypeNames, Boolean SkipHttpBindings = True, Boolean
QualifiedNames = False);
```

Description

Call GetPortTypes to retrieve the names of all port types defined in the WSDL document or any document that it imports. Each port type corresponds to an interface that the Web Service application can execute.

PortTypeNames is an existing TWideStrings object to which the names of all port types are added.

SkipHttpBindings indicates whether GetPortTypes should include port types whose definitions include an <http:binding> tag. When false, GetPortTypes retrieves all port type definitions. When true, GetPortTypes ignores port type definitions with simple

HTTP bindings.

QualifiedNames indicates whether the port type names added to the list should be qualified using the target namespace prefix.

Note: GetPortTypes does not clear the WideString list before adding the names of the port types.

See Also

GetPortTypeNode (see page 25971)

GetOperations (see page 25968)

191.1.1.3.22 TWSDLItems.GetSchemaNode Method

Returns the interface for the schema node that defines a type.

Pascal

```
function GetSchemaNode(const SchemaTns: WideString): IXMLNode;
```

C++

```
__fastcall IXMLNode GetSchemaNode(const BSTR SchemaTns);
```

Description

Call GetSchemaNode to obtain information about a schema definition for a type that is defined in the WSDL document.

SchemaTns is the target namespace of the schema definition for a type.

GetSchemaNode returns the corresponding node.

See Also

GetSchemas (see page 25972)

HasTypesNode (see page 25986)

191.1.1.3.23 TWSDLItems.GetSchemas Method

Fills a list with the namespaces for all schema definitions of types in the WSDL document.

Pascal

```
procedure GetSchemas(SchemaNames: TWideStrings);
```

C++

```
__fastcall GetSchemas(TWideStrings SchemaNames);
```

Description

Call GetSchemaTypes to retrieve the namespaces that correspond to schema definitions of types.

SchemaNames is an existing TWideStrings object to which the namespaces for the schema definitions are added.

Note: GetSchemas does not clear the WideString list before adding the namespaces of the schema definitions.

See Also

GetSchemaNode (see page 25972)

HasTypesNode (see page 25986)

191.1.1.3.24 TWSDLItems.GetServiceAndPortOfBinding Method

Retrieves the names of the service and port associated with a specified binding.

Pascal

```
function GetServiceAndPortOfBinding(const BindingName: IQualifiedName; var ServiceName:
WideString; var PortName: WideString): Boolean;
```

C++

```
__fastcall Boolean GetServiceAndPortOfBinding(const IQualifiedName BindingName, BSTR
ServiceName, BSTR PortName);
```

Description

Call GetServiceAndPortOfBinding to retrieve the name of the service and port associated with a specified binding. GetServiceAndPortOfBinding performs the converse lookup from the GetBindingForServicePort (see page 25962) method.

BindingName specifies the qualified name of the binding for which you want to look up the service and port.

ServiceName returns the name of the service to which the binding belongs.

PortName returns the name of a port within the service that uses the binding.

GetServiceAndPortOfBinding returns true if it locates the specified binding and returns a service and port. It returns false if the specified binding is not defined in the WSDL document.

See Also

GetServices (see page 25974)

GetServiceNode (see page 25973)

GetBindingForServicePort (see page 25962)

GetPortsForService (see page 25970)

GetBindingOfType (see page 25963)

GetBindings (see page 25964)

191.1.1.3.25 TWSDLItems.GetServiceNode Method

Returns the interface for the node that describes a service defined in the WSDL document.

Pascal

```
function GetServiceNode(const ServiceName: WideString): IXMLNode;
```

C++

```
__fastcall IXMLNode GetServiceNode(const BSTR ServiceName);
```

Description

Call GetServiceNode to obtain information about a service that is defined in the WSDL document. A service is a collection of related ports.

ServiceName is the name of the service whose node is returned.

See Also

GetServices (see page 25974)

GetPortsForService (see page 25970)

GetServiceAndPortOfBinding (see page 25973)

191.1.1.3.26 TWSDLItems.GetServices Method

Fills a list with the names of all services defined in the WSDL document.

Pascal

```
procedure GetServices(ServiceNames: TWideStrings; QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetServices(TWideStrings ServiceNames, Boolean QualifiedNames = False);
```

Description

Call GetServices to retrieve the names of all services defined in the WSDL document or any document that it imports. Each service is a collection of related ports.

ServiceNames is an existing TWideStrings object to which the names of all services are added.

QualifiedNames indicates whether the service names added to the list should be qualified using the target namespace prefix.

Note: GetServices does not clear the WideString list before adding the names of the services.

See Also

GetServiceNode (see page 25973)

GetPortsForService (see page 25970)

GetServiceAndPortOfBinding (see page 25973)

191.1.1.3.27 TWSDLItems.GetSoapAction Method

Returns the value of the soapAction attribute of a specified operation on a specified binding.

Pascal

```
function GetSoapAction(const BindingName: WideString; const Operation: WideString;
OverloadIndex: Integer): WideString;
```

C++

```
__fastcall BSTR GetSoapAction(const BSTR BindingName, const BSTR Operation, int OverloadIndex);
```

Description

Call GetSoapAction to retrieve the value of the soapAction attribute on the <soap:operation> tag associated with an operation on a specified SOAP binding. The soap operation tag can be in the WSDL document or in another document that it imports.

BindingName is the name of the binding that includes the operation.

Operation is the name of the operation that includes the <soap:operation> tag.

OverloadIndex indicates which operation is desired when the operation is overloaded.

See Also

GetSoapActionList (see page 25975)

GetBindings (see page 25964)

getoperationsforbinding (see page 25968)

GetSoapOperationAttribute (see page 25982)

191.1.1.3.28 TWSDLItems.GetSoapActionList Method

Fills a list with the soapAction attributes for all operations on a specified SOAP binding.

Pascal

```
procedure GetSoapActionList(const BindingName: WideString; ActionList: TWideStrings;
QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetSoapActionList(const BSTR BindingName, TWideStrings ActionList, Boolean
QualifiedNames = False);
```

Description

Call GetSoapActionList to retrieve the value of the soapAction attribute of each operation of a specified SOAP binding. Each soapAction is added as a string with the format

```
OperationName = soapActionName
```

where OperationName is the name of the operation with which the soapAction attribute is associated.

BindingName is the name of the binding associated with the port type whose operations are retrieved along with their SOAP actions.

OperationNames is an existing TWideStrings object to which the names of all operations on the binding's port type are added.

QualifiedNames specifies whether the operation names should be qualified using the target namespace prefix.

Note: GetSoapActionList does not clear the WideString list before adding the names of the operations and their corresponding SOAP actions.

See Also

GetSoapAction (see page 25974)

GetBindings (see page 25964)

getoperationsforbinding (see page 25968)

GetSoapOperationAttribute (see page 25982)

191.1.1.3.29 TWSDLItems.GetSoapAddressForServicePort Method

Returns the address of a specified port from a specified service.

Pascal

```
function GetSoapAddressForServicePort(const ServiceName: WideString; const PortName:
WideString): WideString;
```

C++

```
__fastcall BSTR GetSoapAddressForServicePort(const BSTR ServiceName, const BSTR PortName);
```

Description

Call GetSoapAddressForServicePort to retrieve the address to which a SOAP request should be sent if it wants to access a specified port from a specified service.

ServiceName is the name of the service that contains the desired port. It is defined in the WSDL document or another document that it imports.

PortName is the name of the port whose address is returned.

See Also

GetBindingForServicePort (see page 25962)

GetPortsForService (see page 25970)

191.1.1.3.30 TWSDLItems.GetSoapBindingAttribute Method

Returns the value of a specified attribute of a specified SOAP binding.

Pascal

```
function GetSoapBindingAttribute(const BindingName: WideString; Attribute: WideString):
WideString;
```

C++

```
__fastcall BSTR GetSoapBindingAttribute(const BSTR BindingName, BSTR Attribute);
```

Description

Call GetSoapBindingAttribute to retrieve value of an attribute on a SOAP binding. GetSoapBindingAttribute locates a binding with the specified name that includes a <soap:binding> tag, and returns the specified attribute from that <soap:binding> tag.

BindingName is the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Attribute is the name of the desired attribute on the <soap:binding> tag.

If BindingName does not identify a binding that includes a <soap:binding> tag that has the specified attribute, GetSoapBindingAttribute returns an empty string.

See Also

GetSoapBodyAttribute (see page 25978)

GetSoapFaultAttribute

GetSoapHeaderAttribute

GetSoapOperationAttribute (see page 25982)

GetBindingForServicePort (see page 25962)

191.1.1.3.31 TWSDLItems.GetSoapBindingInputNode Method

Returns the interface for the input node on a specified binding and operation.

Pascal

```
function GetSoapBindingInputNode(const BindingName: WideString; const Operation: WideString;
OverloadIndex: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode GetSoapBindingInputNode(const BSTR BindingName, const BSTR Operation, int
OverloadIndex);
```

Description

Call GetSoapBindingInputNode to retrieve the node for input messages (parameters) on an operation of a SOAP binding. GetSoapBindingInputNode locates a binding with the specified name that includes a <soap:binding> tag, locates the specified operation node that is a child of that binding, and returns the input node for that operation.

BindingName is the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport

protocols for an associated port type.

Operation is the name of the operation of that <soap:binding> tag. Operations correspond to methods on an invokable interface.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

If GetSoapBindingInputNode is unable to locate an input node for the specified binding and operation, it returns nil (Delphi) or NULL (C++).

See Also

GetSoapBindingOutputNode (see page 25978)

GetSoapBodyAttribute (see page 25978)

GetSoapBindingIONode (see page 25977)

GetBindingForServicePort (see page 25962)

191.1.1.3.32 TWSDLItems.GetSoapBindingIONode Method

Returns the interface for a node on a specified binding and operation.

Pascal

```
function GetSoapBindingIONode(const BindingName: WideString; const Operation: WideString;
Input: Boolean; OverloadIndex: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode GetSoapBindingIONode(const BSTR BindingName, const BSTR Operation, Boolean
Input, int OverloadIndex);
```

Description

Call GetSoapBindingIONode to retrieve a specified node on an operation of a SOAP binding. GetSoapBindingIONode locates a binding with the specified name that includes a <soap:binding> tag, locates the specified operation node that is a child of that binding, and returns the child of that node with a specified name. This method provides the underlying implementation for the GetSoapBindingInputNode (see page 25976) and GetSoapBindingOutputNode (see page 25978) methods.

BindingName is the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that <soap:binding> tag. Operations correspond to methods on an invokable interface.

Input indicates whether the child node is an input node (true) or an output node (false).

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

If GetSoapBindingIONode is unable to locate a node for the specified name, binding, and operation, it returns nil (Delphi) or NULL (C++).

See Also

GetSoapBindingOutputNode (see page 25978)

GetSoapBodyAttribute (see page 25978)

GetSoapBindingInputNode (see page 25976)

GetBindingForServicePort (see page 25962)

191.1.1.3.33 TWSDLItems.GetSoapBindingOutputNode Method

Returns the interface for the output node on a specified binding and operation.

Pascal

```
function GetSoapBindingOutputNode(const BindingName: WideString; const Operation: WideString;
OverloadIndex: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode GetSoapBindingOutputNode(const BSTR BindingName, const BSTR Operation, int
OverloadIndex);
```

Description

Call GetSoapBindingOutputNode to retrieve the node for output messages (output parameters or return value) on an operation of a SOAP binding. GetSoapBindingOutputNode locates a binding with the specified name that includes a <soap:binding> tag, locates the specified operation node that is a child of that binding, and returns the output node for that operation.

BindingName is the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that <soap:binding> tag. Operations correspond to methods on an invokable interface.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

If GetSoapBindingOutputNode is unable to locate an output node for the specified binding and operation, it returns nil (Delphi) or NULL (C++).

See Also

GetSoapBindingInputNode (see page 25976)

GetSoapBodyAttribute (see page 25978)

GetSoapBindingIONode (see page 25977)

GetBindingForServicePort (see page 25962)

191.1.1.3.34 TWSDLItems.GetSoapBodyAttribute Method

Returns the value of an attribute of a body tag of a message type in an operation on the port type of a SOAP binding.

Pascal

```
function GetSoapBodyAttribute(const BindingName: WideString; const Operation: WideString;
const IOType: WideString; const Attribute: WideString; OverloadIndex: Integer): WideString;
```

C++

```
__fastcall BSTR GetSoapBodyAttribute(const BSTR BindingName, const BSTR Operation, const BSTR
IOType, const BSTR Attribute, int OverloadIndex);
```

Description

Call GetSoapBodyAttribute to retrieve value of an attribute on the body tag for a message type that is part of an operation of the port type associated with a specified binding. The body tag can come from the WSDL document or another document that it imports.

BindingName is the name of the binding that includes the operation.

Operation is the name of the operation that includes the message.

IOType indicates the type of the message (input or output).

Attribute is the name of the desired attribute on the <soap:body> tag.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

See Also

GetSoapBindingAttribute (see page 25976)

GetSoapFaultAttribute

GetSoapHeaderAttribute

GetSoapOperationAttribute (see page 25982)

GetBindingForServicePort (see page 25962)

getoperationsforbinding (see page 25968)

GetInOutMessagesForOperation

GetSoapBodyInputAttribute (see page 25979)

GetSoapBodyOutputAttribute (see page 25980)

191.1.1.3.35 TWSDLItems.GetSoapBodyInputAttribute Method

Returns the value of an attribute of a body tag of an input message in an operation on the port type of a SOAP binding.

Pascal

```
function GetSoapBodyInputAttribute(const BindingName: WideString; const Operation: WideString;
const Attribute: WideString; OverloadIndex: Integer): WideString;
```

C++

```
__fastcall BSTR GetSoapBodyInputAttribute(const BSTR BindingName, const BSTR Operation, const
BSTR Attribute, int OverloadIndex);
```

Description

Call GetSoapBodyInputAttribute to retrieve value of an attribute on the body tag for an input message that is part of an operation of the port type associated with a specified binding. The body tag can come from the WSDL document or another document that it imports.

BindingName is the name of the binding that includes the operation.

Operation is the name of the operation that includes the input message.

Attribute is the name of the desired attribute on the <soap:body> tag.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

Note: GetSoapBodyInputAttribute returns the same value as GetSoapBodyAttribute (see page 25978) with an IOType of 'input'.

See Also

GetSoapBodyAttribute (see page 25978)

GetSoapBodyOutputAttribute (see page 25980)

GetSoapHeaderAttribute

GetSoapOperationAttribute (see page 25982)

GetBindingForServicePort (see page 25962)

getoperationsforbinding (see page 25968)

GetInOutMessagesForOperation

191.1.1.3.36 TWSDLItems.GetSoapBodyNamespace Method

Returns the value of the first namespace attribute found on a SOAP body node inside a binding with a specified port type.

Pascal

```
function GetSoapBodyNamespace(const BindingPortType: WideString): WideString;
```

C++

```
__fastcall BSTR GetSoapBodyNamespace(const BSTR BindingPortType);
```

Description

GetSoapBodyNamespace searches the bindings in the WSDL document and any documents it imports for one that has a port type that matches BindingPortType. It then searches the operations of that binding to find one whose messages include a SOAP body node. It returns the namespace attribute of that SOAP body node.

PortTypeName is the name of the port type (interface) that must appear in the binding.

See Also

GetSoapBodyAttribute (see page 25978)

GetPortTypes (see page 25971)

191.1.1.3.37 TWSDLItems.GetSoapBodyOutputAttribute Method

Returns the value of an attribute of a body tag of an output message in an operation on the port type of a SOAP binding.

Pascal

```
function GetSoapBodyOutputAttribute(const BindingName: WideString; const Operation:
WideString; const Attribute: WideString; OverloadIndex: Integer): WideString;
```

C++

```
__fastcall BSTR GetSoapBodyOutputAttribute(const BSTR BindingName, const BSTR Operation, const
BSTR Attribute, int OverloadIndex);
```

Description

Call GetSoapBodyOutputAttribute to retrieve value of an attribute on the body tag for an output message that is part of an operation of the port type associated with a specified binding. The body tag can come from the WSDL document or another document that it imports.

BindingName is the name of the binding that includes the operation.

Operation is the name of the operation that includes the output message.

Attribute is the name of the desired attribute on the <soap:body> tag.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

Note: GetSoapBodyOutputAttribute returns the same value as GetSoapBodyAttribute (see page 25978) with an IOType of 'output'.

See Also

GetSoapBodyAttribute (see page 25978)

GetSoapBodyInputAttribute (see page 25979)

GetSoapHeaderAttribute

GetSoapOperationAttribute (see page 25982)

GetBindingForServicePort (see page 25962)

getoperationsforbinding (see page 25968)

GetInOutMessagesForOperation

191.1.1.3.38 TWSDLItems.GetSoapHeaders Method

Returns an interface for accessing a specified set of header nodes.

Pascal

```
function GetSoapHeaders(BindingName: IQualifiedName; const Operation: WideString; Input:
Boolean; OverloadIndex: Integer; const MessageName: WideString = ''): IHeaderInfoArray;
```

C++

```
__fastcall IHeaderInfoArray GetSoapHeaders(IQualifiedName BindingName, const BSTR Operation,
Boolean Input, int OverloadIndex, const BSTR MessageName = '');
```

Description

Call GetSoapHeaders to retrieve a dynamic array of interfaces for accessing information about the header nodes of a specified operation on a specified binding. This method provides the underlying implementation for the GetSoapInputHeaders (see page 25982) and GetSoapOutputHeaders (see page 25983) methods.

BindingName is an interface for reading the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that <soap:binding> tag. Operations correspond to methods on an invokable interface.

Input indicates whether the header node should belong to the operation's input node (true) or output node (false).

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

MessageName is the name of a Message with which the header should be associated. If MessageName is an empty string, GetSoapHeaders does not check the message with which the header is associated.

If GetSoapHeaders is unable to locate a header node for the specified name, binding, and IOType, it returns nil (Delphi) or NULL (C++).

See Also

GetSoapOutputHeaders (see page 25983)

GetSoapBindingIONode (see page 25977)

GetSoapInputHeaders (see page 25982)

GetBindingForServicePort (see page 25962)

GetPartNode (see page 25969)

191.1.1.3.39 TWSDLItems.GetSoapInputHeaders Method

Returns an interface for accessing the header nodes that belong to the input node of a specified binding and operation.

Pascal

```
function GetSoapInputHeaders(BindingName: IQualifiedName; const Operation: WideString;
OverloadIndex: Integer; const MessageName: WideString = ''): IHeaderInfoArray;
```

C++

```
__fastcall IHeaderInfoArray GetSoapInputHeaders(IQualifiedName BindingName, const BSTR
Operation, int OverloadIndex, const BSTR MessageName = '');
```

Description

Call GetSoapInputHeaders to retrieve a dynamic array of interfaces for accessing the header nodes of the input node on a specified binding. GetSoapInputHeaders locates the specified input node using the GetSoapBindingIONode (see page 25977) method, and extracts all of its child header nodes. It then checks the message and part attributes of that input node, and returns a dynamic array of interfaces for the headers.

BindingName is an interface for reading the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that <soap:binding> tag. Operations correspond to methods on an invokable interface.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

MessageName is the name of a Message with which the header should be associated. If MessageName is an empty string, GetSoapInputHeaders does not check the message with which the header is associated.

If GetSoapInputHeaders is unable to locate an input header node for the specified binding and operation, it returns nil (Delphi) or NULL (C++).

See Also

GetSoapOutputHeaders (see page 25983)

GetSoapBindingIONode (see page 25977)

GetSoapHeaders (see page 25981)

GetBindingForServicePort (see page 25962)

GetPartNode (see page 25969)

191.1.1.3.40 TWSDLItems.GetSoapOperationAttribute Method

Returns the value of an attribute on the <soap:operation> node of an operation on a SOAP binding.

Pascal

```
function GetSoapOperationAttribute(const BindingName: WideString; const Operation: WideString;
const Attribute: WideString; OverloadIndex: Integer): WideString;
```

C++

```
__fastcall BSTR GetSoapOperationAttribute(const BSTR BindingName, const BSTR Operation, const
BSTR Attribute, int OverloadIndex);
```

Description

Call GetSoapOperationAttribute to retrieve value of an attribute on the <soap:operation> tag associated with an operation on a specified SOAP binding. The soap operation tag can be in the WSDL document or in another document that it imports.

BindingName is the name of the binding that includes the operation.

Operation is the name of the operation that includes the <soap:operation> tag.

Attribute is the name of the desired attribute on the <soap:operation> tag.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

See Also

GetSoapBindingAttribute (see page 25976)

GetSoapBodyAttribute (see page 25978)

GetSoapHeaderAttribute

GetSoapOperationAttribute

GetSoapFaultsForOperation

191.1.1.3.41 TWSDLItems.GetSoapOperationStyle Method

Returns the value of the style attribute of a specified operation on a specified binding.

Pascal

```
function GetSoapOperationStyle(const BindingName: WideString; const Operation: WideString;
OverloadIndex: Integer): WideString;
```

C++

```
__fastcall BSTR GetSoapOperationStyle(const BSTR BindingName, const BSTR Operation, int
OverloadIndex);
```

Description

Call GetSoapOperationStyle to retrieve the value of the style attribute on the <soap:operation> tag associated with an operation on a specified SOAP binding. The soap operation tag can be in the WSDL document or in another document that it imports.

BindingName is the name of the binding that includes the operation.

Operation is the name of the operation that includes the <soap:operation> tag.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

See Also

GetBindings (see page 25964)

getoperationsforbinding (see page 25968)

GetSoapOperationAttribute (see page 25982)

191.1.1.3.42 TWSDLItems.GetSoapOutputHeaders Method

Returns an interface for accessing the header nodes that belong to the output node of a specified binding and operation.

Pascal

```
function GetSoapOutputHeaders(BindingName: IQualifiedName; const Operation: WideString;
OverloadIndex: Integer; const MessageName: WideString = ''): IHeaderInfoArray;
```

C++

```
__fastcall IHeaderInfoArray GetSoapOutputHeaders(IQualifiedName BindingName, const BSTR
Operation, int OverloadIndex, const BSTR MessageName = '');
```

Description

Call GetSoapOutputHeaders to retrieve a dynamic array of interfaces for accessing the header nodes of the output node on a specified binding. GetSoapOutputHeaders locates the specified output node using the GetSoapBindingIONode (see page 25977) method, and extracts all of its child header nodes. It then checks the message and part attributes of that output node, and returns a dynamic array of interfaces for the headers.

BindingName is an interface for reading the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that <soap:binding> tag. Operations correspond to methods on an invokable interface.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

MessageName is the name of a Message with which the header should be associated. If MessageName is an empty string, GetSoapOutputHeaders does not check the message with which the header is associated.

If GetSoapOutputHeaders is unable to locate an output header node for the specified binding and operation, it returns nil (Delphi) or NULL (C++).

See Also

GetSoapInputHeaders (see page 25982)

GetSoapBindingIONode (see page 25977)

GetSoapHeaders (see page 25981)

GetBindingForServicePort (see page 25962)

GetPartNode (see page 25969)

191.1.1.3.43 TWSDLItems.GetTargetNamespace Method

Returns the value of the TargetNamespace (see page 25988) property.

Pascal

```
function GetTargetNamespace: WideString;
```

C++

```
__fastcall BSTR GetTargetNamespace();
```

Description

GetTargetNamespace is the read access method for the TargetNamespace (see page 25988) property.

See Also

Definition (see page 25808)

GetTargetNamespacePrefix (see page 25985)

TargetNamespace (see page 25988)

191.1.1.3.44 TWSDLItems.GetTargetNamespacePrefix Method

Returns the prefix string that represents the target namespace of the WSDL document.

Pascal

```
function GetTargetNamespacePrefix: WideString;
```

C++

```
__fastcall BSTR GetTargetNamespacePrefix();
```

Description

Call GetTargetNamespacePrefix to retrieve the prefix string that should be added to node names so that they explicitly use the target namespace of the WSDL document. Node names that include this prefix are defined in the namespace identified by the URL that is the value of the TargetNamespace (see page 25988) property.

See Also

TargetNamespace (see page 25988)

191.1.1.3.45 TWSDLItems.GetWSDLDoc Method

Returns the interface for this TWSDLItems (see page 25951) instance.

Pascal

```
function GetWSDLDoc: IWSDLDocument;
```

C++

```
__fastcall IWSDLDocument GetWSDLDoc();
```

Description

GetWSDLDoc returns the IWSDLDocument for this TWSDLItems (see page 25951) instance. This interface can be used to access the Definition property, which supplies uninterpreted access to the definitions in the WSDL document.

See Also

Definition (see page 25808)

191.1.1.3.46 TWSDLItems.GetWSDLItems Method

Returns a reference to this TWSDLItems (see page 25951) instance.

Pascal

```
function GetWSDLItems: TWSDLItems;
```

C++

```
__fastcall TWSDLItems GetWSDLItems();
```

Description

GetWSDLItems returns a reference to the TWSDLItems (see page 25951) instance. This method is mostly used by code that holds an IWSDLItems (see page 25998) interface and needs to use the methods of the object itself.

See Also

IWSDLItems (see page 25998)

191.1.1.3.47 TWSDLItems.HasTypesNode Method

Indicates whether the WSDL document includes any type definitions.

Pascal

```
function HasTypesNode: Boolean;
```

C++

```
__fastcall Boolean HasTypesNode();
```

Description

HasTypesNode returns true if the WSDL document includes any type definitions, false otherwise.

191.1.1.3.48 TWSDLItems.IsPortTypeHTTPBound Method

Indicates whether a specified port type has an HTTP binding.

Pascal

```
function IsPortTypeHTTPBound(const PortType: WideString): Boolean;
```

C++

```
__fastcall Boolean IsPortTypeHTTPBound(const BSTR PortType);
```

Description

Call IsPortTypeHTTPBound to determine whether there is an HTTP binding for a specified port type. A port type corresponds to an invokable interface.

PortType is the port type you want to check.

IsPortTypeHTTPBound returns true if the specified port type has an HTTP binding, false otherwise.

See Also

GetPortTypes (see page 25971)

GetPortTypeNode (see page 25971)

GetBindingOfType (see page 25963)

191.1.1.3.49 TWSDLItems.IterateImports Method

Executes a callback for every WSDL document that this one imports.

Pascal

```
procedure IterateImports(IterateProc: TWSDLIterateProc);
```

C++

```
__fastcall IterateImports(TWSDLIterateProc IterateProc);
```

Description

IterateImports is used to execute a callback before and after loading each WSDL document that this one imports, including those documents that are imported by documents that this one imports.

191.1.1.3.50 TWSDLItems.Load Method

Loads a WSDL document and activates it.

Pascal

```
procedure Load(const WSDLFileName: WideString);
```

C++

```
__fastcall Load(const BSTR WSDLFileName);
```

Description

Load uses the stream loader specified by the StreamLoader (see page 25987) property to load a WSDL document into memory. It then sets itself to represent that WSDL document and activates it.

WSDLFileName is the name of a WSDL document file, an XML schema file, or the URL where such a document can be found.

See Also

StreamLoader (see page 25987)

LoadFromFile (see page 26121)

LoadFromStream (see page 26121)

LoadFromXML (see page 26122)

191.1.1.4 TWSDLItems Properties

The properties of the TWSDLItems class are listed here.

Public Properties

	Name	Description
R	StreamLoader (see page 25987)	Provides access to an interface for loading the WSDL document.
R	TargetNamespace (see page 25988)	Provides access to an interface for loading the WSDL document.

191.1.1.4.1 TWSDLItems.StreamLoader Property

Provides access to an interface for loading the WSDL document.

Pascal

```
property StreamLoader: IStreamLoader;
```

C++

```
__property IStreamLoader StreamLoader;
```

Description

StreamLoader is set by the TWSDLItems (see page 25951) constructor. It is an interface that specifies basic authentication information that can be used to load a WSDL document that is on another server. The IStreamLoader interface specifies the password, user name, and proxy server that can be used to connect to the server that hosts the WSDL document. TWSDLItems (see page 25951) uses this to load the WSDL document.

See Also

Create (see page 25960)

191.1.1.4.2 TWSDLItems.TargetNamespace Property

Provides access to an interface for loading the WSDL document.

Pascal

```
property TargetNamespace: WideString;
```

C++

```
__property BSTR TargetNamespace;
```

Description

The root node of a WSDL document is always a definitions node. This node includes a TargetNamespace attribute that uniquely identifies the set of definitions in the WSDL document. Read TargetNamespace to retrieve that namespace.

See Also

Definition (see page 25808)

GetTargetNamespacePrefix (see page 25985)

191.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IHeaderInfo (see page 25989)	This is class WSDLItems.IHeaderInfo.
	IQualifiedName (see page 25994)	IQualifiedName is the interface for obtaining an information about a name in an XML document.
	IWSDLItems (see page 25998)	IWSDLItems provides access to a WSDL document.

191.2.1 IHeaderInfo Interface

Class Hierarchy

WSDLItems.IHeaderInfo

Pascal

```
IHeaderInfo = interface;
```

C++

```
__interface IHeaderInfo;
```

File

WSDLItems

Description

This is class WSDLItems.IHeaderInfo.

191.2.1.1 IHeaderInfo Members

The following tables list the members exposed by IHeaderInfo.

Public Methods

	Name	Description
	GetEncodingStyle (see page 25990)	This is GetEncodingStyle, a member of class IHeaderInfo.
	GetName (see page 25990)	This is GetName, a member of class IHeaderInfo.
	GetNamespace (see page 25991)	This is GetNamespace, a member of class IHeaderInfo.
	GetPart (see page 25991)	This is GetPart, a member of class IHeaderInfo.
	GetRequired (see page 25991)	This is GetRequired, a member of class IHeaderInfo.
	GetUse (see page 25991)	This is GetUse, a member of class IHeaderInfo.
	SetEncodingStyle (see page 25991)	This is SetEncodingStyle, a member of class IHeaderInfo.
	SetName (see page 25992)	This is SetName, a member of class IHeaderInfo.
	SetNamespace (see page 25992)	This is SetNamespace, a member of class IHeaderInfo.
	SetPart (see page 25992)	This is SetPart, a member of class IHeaderInfo.
	SetRequired (see page 25992)	This is SetRequired, a member of class IHeaderInfo.
	SetUse (see page 25993)	This is SetUse, a member of class IHeaderInfo.

Public Properties

	Name	Description
	EncodingStyle (see page 25993)	This is EncodingStyle, a member of class IHeaderInfo.
	Name (see page 25993)	This is Name, a member of class IHeaderInfo.
	Namespace (see page 25993)	This is Namespace, a member of class IHeaderInfo.
	Part (see page 25994)	This is Part, a member of class IHeaderInfo.
	Required (see page 25994)	This is Required, a member of class IHeaderInfo.
	Use (see page 25994)	This is Use, a member of class IHeaderInfo.

191.2.1.2 IHeaderInfo Methods

The methods of the IHeaderInfo class are listed here.

Public Methods

	Name	Description
	GetEncodingStyle (see page 25990)	This is GetEncodingStyle, a member of class IHeaderInfo.
	GetName (see page 25990)	This is GetName, a member of class IHeaderInfo.
	GetNamespace (see page 25991)	This is GetNamespace, a member of class IHeaderInfo.
	GetPart (see page 25991)	This is GetPart, a member of class IHeaderInfo.
	GetRequired (see page 25991)	This is GetRequired, a member of class IHeaderInfo.
	GetUse (see page 25991)	This is GetUse, a member of class IHeaderInfo.
	SetEncodingStyle (see page 25991)	This is SetEncodingStyle, a member of class IHeaderInfo.
	SetName (see page 25992)	This is SetName, a member of class IHeaderInfo.
	SetNamespace (see page 25992)	This is SetNamespace, a member of class IHeaderInfo.
	SetPart (see page 25992)	This is SetPart, a member of class IHeaderInfo.
	SetRequired (see page 25992)	This is SetRequired, a member of class IHeaderInfo.
	SetUse (see page 25993)	This is SetUse, a member of class IHeaderInfo.

191.2.1.2.1 IHeaderInfo.GetEncodingStyle Method

Pascal

```
function GetEncodingStyle: WideString;
```

C++

```
__fastcall BSTR GetEncodingStyle();
```

Description

This is GetEncodingStyle, a member of class IHeaderInfo.

191.2.1.2.2 IHeaderInfo.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IHeaderInfo.

191.2.1.2.3 IHeaderInfo.GetNamespace Method

Pascal

```
function GetNamespace: WideString;
```

C++

```
__fastcall BSTR GetNamespace();
```

Description

This is GetNamespace, a member of class IHeaderInfo.

191.2.1.2.4 IHeaderInfo.GetPart Method

Pascal

```
function GetPart: IPart;
```

C++

```
__fastcall IPart GetPart();
```

Description

This is GetPart, a member of class IHeaderInfo.

191.2.1.2.5 IHeaderInfo.GetRequired Method

Pascal

```
function GetRequired: Boolean;
```

C++

```
__fastcall Boolean GetRequired();
```

Description

This is GetRequired, a member of class IHeaderInfo.

191.2.1.2.6 IHeaderInfo.GetUse Method

Pascal

```
function GetUse: WideString;
```

C++

```
__fastcall BSTR GetUse();
```

Description

This is GetUse, a member of class IHeaderInfo.

191.2.1.2.7 IHeaderInfo.SetEncodingStyle Method

Pascal

```
procedure SetEncodingStyle(const EncStyl: WideString);
```

C++

```
__fastcall SetEncodingStyle(const BSTR EncStyl);
```

Description

This is SetEncodingStyle, a member of class IHeaderInfo.

191.2.1.2.8 IHeaderInfo.SetName Method

Pascal

```
procedure SetName(const N: WideString);
```

C++

```
__fastcall SetName(const BSTR N);
```

Description

This is SetName, a member of class IHeaderInfo.

191.2.1.2.9 IHeaderInfo.SetNamespace Method

Pascal

```
procedure SetNamespace(const N: WideString);
```

C++

```
__fastcall SetNamespace(const BSTR N);
```

Description

This is SetNamespace, a member of class IHeaderInfo.

191.2.1.2.10 IHeaderInfo.SetPart Method

Pascal

```
procedure SetPart(const P: IPart);
```

C++

```
__fastcall SetPart(const IPart P);
```

Description

This is SetPart, a member of class IHeaderInfo.

191.2.1.2.11 IHeaderInfo.SetRequired Method

Pascal

```
procedure SetRequired(Req: Boolean);
```

C++

```
__fastcall SetRequired(Boolean Req);
```

Description

This is SetRequired, a member of class IHeaderInfo.

191.2.1.2.12 IHeaderInfo.SetUse Method

Pascal

```
procedure SetUse(const U: WideString);
```

C++

```
__fastcall SetUse(const BSTR U);
```

Description

This is SetUse, a member of class IHeaderInfo.

191.2.1.3 IHeaderInfo Properties

The properties of the IHeaderInfo class are listed here.

Public Properties

	Name	Description
	EncodingStyle (see page 25993)	This is EncodingStyle, a member of class IHeaderInfo.
	Name (see page 25993)	This is Name, a member of class IHeaderInfo.
	Namespace (see page 25993)	This is Namespace, a member of class IHeaderInfo.
	Part (see page 25994)	This is Part, a member of class IHeaderInfo.
	Required (see page 25994)	This is Required, a member of class IHeaderInfo.
	Use (see page 25994)	This is Use, a member of class IHeaderInfo.

191.2.1.3.1 IHeaderInfo.EncodingStyle Property

Pascal

```
property EncodingStyle: WideString;
```

C++

```
__property BSTR EncodingStyle;
```

Description

This is EncodingStyle, a member of class IHeaderInfo.

191.2.1.3.2 IHeaderInfo.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IHeaderInfo.

191.2.1.3.3 IHeaderInfo.Namespace Property

Pascal

```
property Namespace: WideString;
```

C++

```
__property BSTR Namespace;
```

Description

This is Namespace, a member of class IHeaderInfo.

191.2.1.3.4 IHeaderInfo.Part Property

Pascal

```
property Part: IPart;
```

C++

```
__property IPart Part;
```

Description

This is Part, a member of class IHeaderInfo.

191.2.1.3.5 IHeaderInfo.Required Property

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;
```

Description

This is Required, a member of class IHeaderInfo.

191.2.1.3.6 IHeaderInfo.Use Property

Pascal

```
property Use: WideString;
```

C++

```
__property BSTR Use;
```

Description

This is Use, a member of class IHeaderInfo.

191.2.2 IQualifiedName Interface

IQualifiedName is the interface for obtaining an information about a name in an XML document.

Class Hierarchy

WSDLItems.IQualifiedName

Pascal

```
IQualifiedName = interface;
```

C++

```
__interface IQualifiedName;
```

File

WSDLItems

Description

IQualifiedName introduces two properties, Name (see page 25997) and Namespace (see page 25997), which represent the local name and namespace URI for a name that is used in an XML document. Using IQualifiedName makes it easier to work with names where it is important to know the namespace in which the name is defined.

Note: C++ method declarations that use IQualifiedName use the _di_IQualifiedName type instead. This type is a DelphiInterface wrapper around the IQualifiedName interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IQualifiedName >  _di_IQualifiedName;
```

See Also

TWSDLItems (see page 25951)

TMemoryStream (see page 2902)

191.2.2.1 IQualifiedName Members

The following tables list the members exposed by IQualifiedName.

Public Methods

	Name	Description
	GetName (see page 25995)	Returns the value of the Name (see page 25997) property.
	GetNamespace (see page 25996)	Returns the value of the Namespace (see page 25997) property.
	SetName (see page 25996)	Sets the value of the Name (see page 25997) property.
	SetNamespace (see page 25996)	Sets the value of the Namespace (see page 25997) property.

Public Properties

	Name	Description
	Name (see page 25997)	Specifies the local name.
	Namespace (see page 25997)	Specifies the namespace in which the name is defined.

191.2.2.2 IQualifiedName Methods

The methods of the IQualifiedName class are listed here.

Public Methods

	Name	Description
	GetName (see page 25995)	Returns the value of the Name (see page 25997) property.
	GetNamespace (see page 25996)	Returns the value of the Namespace (see page 25997) property.
	SetName (see page 25996)	Sets the value of the Name (see page 25997) property.
	SetNamespace (see page 25996)	Sets the value of the Namespace (see page 25997) property.

191.2.2.2.1 IQualifiedName.GetName Method

Returns the value of the Name (see page 25997) property.

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

GetName is the read implementation for the Name (see page 25997) property.

See Also

Name (see page 25997)

191.2.2.2.2 IQualifiedName.GetNamespace Method

Returns the value of the Namespace (see page 25997) property.

Pascal

```
function GetNamespace: WideString;
```

C++

```
__fastcall BSTR GetNamespace();
```

Description

GetNamespace is the read implementation for the Namespace (see page 25997) property.

See Also

Namespace (see page 25997)

191.2.2.2.3 IQualifiedName.SetName Method

Sets the value of the Name (see page 25997) property.

Pascal

```
procedure SetName(const name: WideString);
```

C++

```
__fastcall SetName(const BSTR name);
```

Description

SetName is the write implementation for the Name (see page 25997) property.

Name (see page 25997) is the new name.

See Also

Name (see page 25997)

191.2.2.2.4 IQualifiedName.SetNamespace Method

Sets the value of the Namespace (see page 25997) property.

Pascal

```
procedure SetNamespace(const ns: WideString);
```

C++

```
__fastcall SetNamespace(const BSTR ns);
```

Description

SetNamespace is the write implementation for the Namespace (see page 25997) property.

Namespace (see page 25997) is the new namespace in which to interpret Name (see page 25997).

See Also

Namespace (see page 25997)

191.2.2.3 IQualifiedName Properties

The properties of the IQualifiedName class are listed here.

Public Properties

	Name	Description
	Name (see page 25997)	Specifies the local name.
	Namespace (see page 25997)	Specifies the namespace in which the name is defined.

191.2.2.3.1 IQualifiedName.Name Property

Specifies the local name.

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

Name specifies the local name portion of the qualified name. This name is defined within the context of the namespace that the Namespace (see page 25997) property specifies.

See Also

Namespace (see page 25997)

191.2.2.3.2 IQualifiedName.Namespace Property

Specifies the namespace in which the name is defined.

Pascal

```
property Namespace: WideString;
```

C++

```
__property BSTR Namespace;
```

Description

Namespace specifies the namespace that is used to interpret the value of the Name (see page 25997) property.

See Also

Name (see page 25997)

191.2.3 IWSDLItems Interface

IWSDLItems provides access to a WSDL document.

Class Hierarchy

WSDLItems.IWSDLItems

Pascal

```
IWSDLItems = interface;
```

C++

```
__interface IWSDLItems;
```

File

WSDLItems

Description

IWSDLItems is an interface to the TWSDLItems (see page 25951) class. You can use this interface to load the WSDL document from a file or server, obtain access to the TWSDLItems (see page 25951) object whose methods provide information about the definitions in that document, and obtain an IWSDLDocument interface for low-level access to the definitions in the document.

Note: C++ method declarations that use IWSDLItems use the _di_IWSDLItems type instead. This type is a DelphiInterface wrapper around the IWSDLItems interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IWSDLDocument >  _di_IWSDLDocument;
```

See Also

TWSDLItems (see page 25951)

IWSDLItems

191.2.3.1 IWSDLItems Members

The following tables list the members exposed by IWSDLItems.

Public Methods

	Name	Description
	GetWSDLDoc (see page 25999)	Returns the IWSDLDocument interface for the WSDL document.
	GetWSDLItems (see page 25999)	Returns a reference to the TWSDLItems (see page 25951) instance that implements this interface.
	Load (see page 25999)	Loads a WSDL document so that its definitions are available.

191.2.3.2 IWSDLItems Methods

The methods of the IWSDLItems class are listed here.

Public Methods

	Name	Description
	GetWSDLDoc (see page 25999)	Returns the IWSDLDocument interface for the WSDL document.
	GetWSDLItems (see page 25999)	Returns a reference to the TWSDLItems (see page 25951) instance that implements this interface.
	Load (see page 25999)	Loads a WSDL document so that its definitions are available.

191.2.3.2.1 IWSDLItems.GetWSDLDoc Method

Returns the IWSDLDocument interface for the WSDL document.

Pascal

```
function GetWSDLDoc: IWSDLDocument;
```

C++

```
__fastcall IWSDLDocument GetWSDLDoc();
```

Description

GetWSDLDoc returns a IWSDLDocument interface, which supplies uninterpreted access to the definitions in the WSDL document. To access these definitions more easily, use the GetWSDLItems (see page 25999) method, which returns a reference to the underlying TWSDLItems (see page 25951) object.

See Also

GetWSDLItems (see page 25999)

191.2.3.2.2 IWSDLItems.GetWSDLItems Method

Returns a reference to the TWSDLItems (see page 25951) instance that implements this interface.

Pascal

```
function GetWSDLItems: TWSDLItems;
```

C++

```
__fastcall TWSDLItems GetWSDLItems();
```

Description

GetWSDLItems returns a reference to the underlying TWSDLItems (see page 25951) instance. This object has numerous methods that let you access the definitions in the WSDL document.

191.2.3.2.3 IWSDLItems.Load Method

Loads a WSDL document so that its definitions are available.

Pascal

```
procedure Load(const WSDLFileName: WideString);
```

C++

```
__fastcall Load(const BSTR WSDLFileName);
```

Description

Load reads and interprets a specified WSDL document. This method should be called before you attempt to access the definitions in the WSDL document.

WSDLFileName is the name of a WSDL document file, an XML schema file, or the URL where such a document can be found.

191.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	HasDefinition (see page 26000)	This is function WSDLItems.HasDefinition.
	HasDefinition (see page 26000)	This is function WSDLItems.HasDefinition.
	NewQualifiedName (see page 26000)	This is function WSDLItems.NewQualifiedName.

191.3.1 WSDLItems.HasDefinition Function

Pascal

```
function HasDefinition(const WSDLDoc: IWSDLDocument): Boolean; overload;
```

C++

```
Boolean HasDefinition(const IWSDLDocument WSDLDoc);
```

File

WSDLItems

Description

This is function WSDLItems.HasDefinition.

191.3.2 WSDLItems.HasDefinition Function

Pascal

```
function HasDefinition(const WSDLItems: IWSDLItems): Boolean; overload;
```

C++

```
Boolean HasDefinition(const IWSDLItems WSDLItems);
```

File

WSDLItems

Description

This is function WSDLItems.HasDefinition.

191.3.3 WSDLItems.NewQualifiedName Function

Pascal

```
function NewQualifiedName(const Name: WideString = ''; const Namespace: WideString = ''):
```

```
IQualifiedName;
```

C++

```
IQualifiedName NewQualifiedName(const BSTR Name = '', const BSTR Namespace = '');
```

File

WSDLItems

Description

This is function WSDLItems.NewQualifiedName.

191.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	IterateImportOptions (see page 26002)	This is record WSDLItems.IterateImportOptions.

191.4.1 WSDLItems.IterateImportOptions Enumeration

Pascal

```
IterateImportOptions = (
  ioBeforeLoad,
  ioLoaded
);
```

C++

```
enum IterateImportOptions {
  ioBeforeLoad,
  ioLoaded
};
```

File

WSDLItems

Description

This is record WSDLItems.IterateImportOptions.

191.5 Types

The following table lists types in this documentation.

Types

Name	Description
IHeaderInfoArray (see page 26003)	This is type WSDLItems.IHeaderInfoArray.
IQualifiedNameArray (see page 26003)	This is type WSDLItems.IQualifiedNameArray.
TWSDLIterateProc (see page 26003)	This is type WSDLItems.TWSDLIterateProc.

191.5.1 WSDLItems.IHeaderInfoArray Type

Pascal

```
IHeaderInfoArray = array of IHeaderInfo;
```

C++

```
array of IHeaderInfo IHeaderInfoArray;
```

File

WSDLItems

Description

This is type WSDLItems.IHeaderInfoArray.

191.5.2 WSDLItems.IQualifiedNameArray Type

Pascal

```
IQualifiedNameArray = array of IQualifiedName;
```

C++

```
array of IQualifiedName IQualifiedNameArray;
```

File

WSDLItems

Description

This is type WSDLItems.IQualifiedNameArray.

191.5.3 WSDLItems.TWSDLIterateProc Type

Pascal

```
TWSDLIterateProc = procedure (const Options: IterateImportOptions; const WSDLItems:
```

```
IWSDLItems; const XMLSchemaDoc: IXMLSchemaDoc; const Name: WideString) of object;
```

C++

```
(const Options: IterateImportOptions; const WSDLItems: IWSDLItems; const XMLSchemaDoc:
IXMLSchemaDoc; const Name: WideString) ( TWSDLIterateProc)();
```

File

WSDLItems

Description

This is type WSDLItems.TWSDLIterateProc.

192 WSDLNode Namespace

This is namespace WSDLNode.

Topics

Name	Description
Classes (see page 26006)	The following table lists classes in this documentation.
Functions (see page 26022)	The following table lists functions in this documentation.

Classes

	Name	Description
	EWSDLLoadException (see page 26006)	This is class WSDLNode.EWSDLLoadException.
	TWSDLClientNode (see page 26006)	This is class WSDLNode.TWSDLClientNode.
	TWSDLView (see page 26013)	TWSDLView represents a specific binding and operation in a WSDL (see page 26020) document.

Functions

	Name	Description
	ActivateWSDL (see page 26022)	This is function WSDLNode.ActivateWSDL.

192.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EWSDLLoadException (see page 26006)	This is class WSDLNode.EWSDLLoadException.
	TWSDLClientNode (see page 26006)	This is class WSDLNode.TWSDLClientNode.
	TWSDLView (see page 26013)	TWSDLView represents a specific binding and operation in a WSDL (see page 26020) document.

192.1.1 EWSDLLoadException Class

Class Hierarchy

Pascal

```
EWSDLLoadException = class(Exception);
```

C++

```
class EWSDLLoadException : public Exception;
```

File

WSDLNode

Description

This is class WSDLNode.EWSDLLoadException.

192.1.2 TWSDLClientNode Class

Class Hierarchy

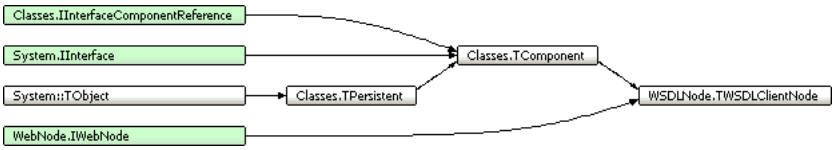

Pascal

```
TWSDLClientNode = class(TComponent, IWebNode);
```

C++

```
class TWSDLClientNode : public TComponent, public IWebNode;
```

File

WSDLNode

Description

This is class WSDLNode.TWSDLClientNode.

192.1.2.1 TWSDLClientNode Members

The following tables list the members exposed by TWSDLClientNode.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetMimeBoundary (see page 26011)	Returns the value of the MimeBoundary property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetMimeBoundary (see page 26012)	Sets the value of the MimeBoundary property.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.

	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	BeforeExecute (see page 26010)	Performs any transport-specific setup before executing an encoded method call.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 2766)	Disposes of the component and its owned components.
	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Execute (see page 25432)	This is the overview for the Execute method overload.
	Execute (see page 26010)	This is the overview for the Execute method overload.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Notification (see page 26012)	Forwards notification messages to all owned components.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	MimeBoundary (see page 25433)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	TransportNode (see page 26013)	This is TransportNode, a member of class TWSDLClientNode.
	WSDLView (see page 26013)	This is WSDLView, a member of class TWSDLClientNode.

192.1.2.2 TWSDLClientNode Methods

The methods of the TWSDLClientNode class are listed here.

Protected Methods

	Name	Description
	GetMimeBoundary (see page 26011)	Returns the value of the MimeBoundary property.
	SetMimeBoundary (see page 26012)	Sets the value of the MimeBoundary property.

Public Methods

	Name	Description
	BeforeExecute (see page 26010)	Performs any transport-specific setup before executing an encoded method call.
	Execute (see page 26010)	This is the overview for the Execute method overload.
	Notification (see page 26012)	Forwards notification messages to all owned components.

192.1.2.2.1 TWSDLClientNode.BeforeExecute Method

Performs any transport-specific setup before executing an encoded method call.

Pascal

```
procedure BeforeExecute(const IntfMetaData: TIntfMetaData; const MethodMetaData:
TIntfMethEntry; MethodIndex: Integer; AttachHandler: IMimeAttachmentHandler);
```

C++

```
__fastcall BeforeExecute(const TIntfMetaData IntfMetaData, const TIntfMethEntry
MethodMetaData, int MethodIndex, IMimeAttachmentHandler AttachHandler);
```

Description

BeforeExecute handles transport-specific tasks such as dynamically querying a WSDL document or setting up serialization options before a call is made to the Execute method. Remote interfaced objects (TRio) call this method after their OnBeforeExecute event, but before delegating the execution of the method to the Execute method.

IntfMD is the runtime type information (RTTI) of the invokable interface that includes the method to execute.

MethMD is describes the method that is about to be executed.

MethodIndex indicates which overload to use when methMD describes an overloaded method. 0 indicates the first overload, 1 indicates the second overload, and so on.

AttachHandler is an interface to use for handling any attachments that are used as parameters of the method.

See Also

Execute (see page 25432)

OnBeforeExecute (see page 18730)

192.1.2.2.2 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
	TWSDLClientNode.Execute (String, TStream) (see page 26010)	Executes an encoded method call and returns the encoded results in a stream.
	TWSDLClientNode.Execute (TStream) (see page 26011)	This is Execute, a member of class TWSDLClientNode.
	TWSDLClientNode.Execute (TStream, TStream) (see page 26011)	This is Execute, a member of class TWSDLClientNode.

192.1.2.2.2.1 TWSDLClientNode.Execute Method (String, TStream)

Executes an encoded method call and returns the encoded results in a stream.

Pascal

```
procedure Execute(const DataMsg: String; Resp: TStream); overload;
```

C++

```
__fastcall Execute(const AnsiString DataMsg, TStream Resp);
```

Description

Execute interprets an encoded method call, executes it, and returns the encoded results using a stream. Typically, this method works in conjunction with an object that implements the IOPConvert interface, which marshals a method call into the DataMsg or Request string and unmarshals the return value.

DataMsg is an encoded method call. It represents the results of marshaling a method call on an invokable interface into a transportable string. The use of this syntax is not recommended.

Request is a stream from which to read the same string that is the value of the DataMsg parameter. This syntax is preferred because the stream need not store this value in memory (for example, it can be a file stream).

Response is a stream object that receives the encoded results that arise from executing the method call.

When using the third syntax, Execute returns a stream from which to read the encoded results from executing the method call. In this case, the caller is responsible for freeing the stream.

See Also

IOPConvert (see page 18122)

BeforeExecute (see page 25431)

192.1.2.2.2.2 TWSDLClientNode.Execute Method (TStream)

Pascal

```
function Execute(const Request: TStream): TStream; overload;
```

C++

```
__fastcall TStream Execute(const TStream Request);
```

Description

This is Execute, a member of class TWSDLClientNode.

192.1.2.2.2.3 TWSDLClientNode.Execute Method (TStream, TStream)

Pascal

```
procedure Execute(const Request: TStream; Response: TStream); overload;
```

C++

```
__fastcall Execute(const TStream Request, TStream Response);
```

Description

This is Execute, a member of class TWSDLClientNode.

192.1.2.2.3 TWSDLClientNode.GetMimeBoundary Method

Returns the value of the MimeBoundary property.

Pascal

```
function GetMimeBoundary: string;
```

C++

```
__fastcall AnsiString GetMimeBoundary();
```

Description

GetMimeBoundary is the read access method for the MimeBoundary property.

See Also

MimeBoundary (see page 25433)

192.1.2.2.4 TWSDLClientNode.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

192.1.2.2.5 TWSDLClientNode.SetMimeBoundary Method

Sets the value of the MimeBoundary property.

Pascal

```
procedure SetMimeBoundary(Value: string);
```

C++

```
__fastcall SetMimeBoundary(AnsiString Value);
```

Description

SetMimeBoundary is the write implementation for the MimeBoundary property.

Value is the new value for MimeBoundary.

See Also

MimeBoundary (see page 25433)

192.1.2.3 TWSDLClientNode Properties

The properties of the TWSDLClientNode class are listed here.

Published Properties

	Name	Description
	TransportNode (see page 26013)	This is TransportNode, a member of class TWSDLClientNode.
	WSDLView (see page 26013)	This is WSDLView, a member of class TWSDLClientNode.

192.1.2.3.1 TWSDLClientNode.TransportNode Property

Pascal

```
property TransportNode: IWebNode;
```

C++

```
__property IWebNode TransportNode;
```

Description

This is TransportNode, a member of class TWSDLClientNode.

192.1.2.3.2 TWSDLClientNode.WSDLView Property

Pascal

```
property WSDLView: TWSDLView;
```

C++

```
__property TWSDLView WSDLView;
```

Description

This is WSDLView, a member of class TWSDLClientNode.

192.1.3 TWSDLView Class

TWSDLView represents a specific binding and operation in a WSDL (see page 26020) document.

Class Hierarchy

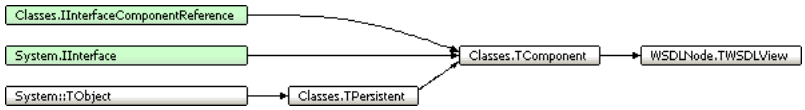

Pascal

```
TWSDLView = class(TComponent);
```

C++

```
class TWSDLView : public TComponent;
```

File

WSDLNode

Description

TWSDLView is used by components that work with WSDL (see page 26020) documents to represent a specific port and/or operation that is defined in a WSDL (see page 26020) document. The WSDL (see page 26020) property provides access to the WSDL (see page 26020) document itself. The other properties give the names of the entities of interest that are defined within that document.

See Also

THTTPReqResp (see page 19961)

TOPToSoapDomConvert (see page 18137)

TWSDLItems (see page 25951)

192.1.3.1 TWSDLView Members

The following tables list the members exposed by TWSDLView.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Fields

	Name	Description
	IntfInfo (see page 26017)	This is IntfInfo, a member of class TWSDLView.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Activate (see page 26017)	This is Activate, a member of class TWSDLView.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.

	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetDesignState (see page 26017)	This is SetDesignState, a member of class TWSDLView.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Operation (see page 26018)	Identifies the target operation that is defined in the WSDL (see page 26020) document.
	Password (see page 26018)	This is Password, a member of class TWSDLView.
	Port (see page 26019)	Specifies the target port that is defined in the WSDL (see page 26020) document.
	PortType (see page 26019)	Specifies the target port type that is defined in the WSDL (see page 26020) document.

	Proxy (see page 26020)	This is Proxy, a member of class TWSDLView.
	Service (see page 26020)	Specifies the target service that is defined in the WSDL (see page 26020) document.
	Tag (see page 2808)	Stores an integer value as part of a component.
	UserName (see page 26020)	This is UserName, a member of class TWSDLView.
	WSDL (see page 26020)	Provides access to the definitions in the WSDL document.

192.1.3.2 TWSDLView Fields

The fields of the TWSDLView class are listed here.

Public Fields

	Name	Description
	IntfInfo (see page 26017)	This is IntfInfo, a member of class TWSDLView.

192.1.3.2.1 TWSDLView.IntfInfo Field

Pascal

```
IntfInfo: PTypeInfo;
```

C++

```
PTypeInfo IntfInfo;
```

Description

This is IntfInfo, a member of class TWSDLView.

192.1.3.3 TWSDLView Methods

The methods of the TWSDLView class are listed here.

Public Methods

	Name	Description
	Activate (see page 26017)	This is Activate, a member of class TWSDLView.
	SetDesignState (see page 26017)	This is SetDesignState, a member of class TWSDLView.

192.1.3.3.1 TWSDLView.Activate Method

Pascal

```
procedure Activate;
```

C++

```
__fastcall Activate();
```

Description

This is Activate, a member of class TWSDLView.

192.1.3.3.2 TWSDLView.SetDesignState Method

Pascal

```
procedure SetDesignState(Designing: Boolean);
```

C++

```
__fastcall SetDesignState(Boolean Designing);
```

Description

This is SetDesignState, a member of class TWSDLView.

192.1.3.4 TWSDLView Properties

The properties of the TWSDLView class are listed here.

Published Properties

	Name	Description
	Operation (see page 26018)	Identifies the target operation that is defined in the WSDL (see page 26020) document.
	Password (see page 26018)	This is Password, a member of class TWSDLView.
	Port (see page 26019)	Specifies the target port that is defined in the WSDL (see page 26020) document.
	PortType (see page 26019)	Specifies the target port type that is defined in the WSDL (see page 26020) document.
	Proxy (see page 26020)	This is Proxy, a member of class TWSDLView.
	Service (see page 26020)	Specifies the target service that is defined in the WSDL (see page 26020) document.
	UserName (see page 26020)	This is UserName, a member of class TWSDLView.
	WSDL (see page 26020)	Provides access to the definitions in the WSDL document.

192.1.3.4.1 TWSDLView.Operation Property

Identifies the target operation that is defined in the WSDL (see page 26020) document.

Pascal

```
property Operation: string;
```

C++

```
__property AnsiString Operation;
```

Description

Operation is the name of an operation that is defined in the WSDL (see page 26020) document identified by the WSDL (see page 26020) property. Operation maps to a method on the interface that corresponds to the PortType (see page 26019) property.

See Also

PortType (see page 26019)

WSDL (see page 26020)

192.1.3.4.2 TWSDLView.Password Property

Pascal

```
property Password: string;
```

C++

```
__property AnsiString Password;
```

Description

This is Password, a member of class TWSDLView.

192.1.3.4.3 TWSDLView.Port Property

Specifies the target port that is defined in the WSDL (see page 26020) document.

Pascal

```
property Port: string;
```

C++

```
__property AnsiString Port;
```

Description

Port is the name of a port that is defined in the WSDL (see page 26020) document identified by the WSDL (see page 26020) property. A Port represents the address of a Web Service (see page 26020) application, a transport protocol that is used for requests to that application, the data format to use for encoding requests, and the port type that defines the operations contained in those requests.

See Also

PortType (see page 26019)

WSDL (see page 26020)

Service (see page 26020)

192.1.3.4.4 TWSDLView.PortType Property

Specifies the target port type that is defined in the WSDL (see page 26020) document.

Pascal

```
property PortType: string;
```

C++

```
__property AnsiString PortType;
```

Description

Port (see page 26019) is the name of a port type that is defined in the WSDL (see page 26020) document identified by the WSDL (see page 26020) property. A port type corresponds to an interface that the Web Service (see page 26020) application implements.

The target method on that interface is identified by the Operation (see page 26018) property. The address and protocols to use when calling this interface are identified by the Port (see page 26019) property.

See Also

WSDL (see page 26020)

Port (see page 26019)

Operation (see page 26018)

192.1.3.4.5 TWSDLView.Proxy Property

Pascal

```
property Proxy: string;
```

C++

```
__property AnsiString Proxy;
```

Description

This is Proxy, a member of class TWSDLView.

192.1.3.4.6 TWSDLView.Service Property

Specifies the target service that is defined in the WSDL (see page 26020) document.

Pascal

```
property Service: string;
```

C++

```
__property AnsiString Service;
```

Description

Service is the name of the service of interest that is defined in the WSDL (see page 26020) document identified by the WSDL (see page 26020) property. A service is a collection of related ports, including the one specified by the Port (see page 26019) property.

See Also

Port (see page 26019)

WSDL (see page 26020)

192.1.3.4.7 TWSDLView.UserName Property

Pascal

```
property UserName: string;
```

C++

```
__property AnsiString UserName;
```

Description

This is UserName, a member of class TWSDLView.

192.1.3.4.8 TWSDLView.WSDL Property

Provides access to the definitions in the WSDL document.

Pascal

```
property WSDL: TWSDLItems;
```

C++

```
__property TWSDLItems WSDL;
```

Description

WSDLItems (see page 25949) is a helper object whose methods return the definitions included in the WSDL document. All the other TWSDLView (see page 26013) properties refer to definitions contained in this WSDL document

192.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	ActivateWSDL (see page 26022)	This is function WSDLNode.ActivateWSDL.

192.2.1 WSDLNode.ActivateWSDL Function

Pascal

```
function ActivateWSDL(WSDL: TWSDLItems; const Name: string; const Password: string; const
Proxy: string): Boolean;
```

C++

```
Boolean ActivateWSDL(TWSDLItems WSDL, const AnsiString Name, const AnsiString Password, const
AnsiString Proxy);
```

File

WSDLNode

Description

This is function WSDLNode.ActivateWSDL.

193 WSDLPub Namespace

This is namespace WSDLPub.

Topics

Name	Description
Classes (see page 26024)	The following table lists classes in this documentation.
Interfaces (see page 26042)	The following table lists interfaces in this documentation.
Functions (see page 26044)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 26045)	The following table lists structs, records, enums in this documentation.
Types (see page 26046)	The following table lists types in this documentation.

Classes

	Name	Description
	TWSDLHTMLPublish (see page 26024)	TWSDLHTMLPublish publishes a list of WSDL documents describing a Web Services application.
	TWSDLPublish (see page 26037)	This is class WSDLPub.TWSDLPublish.

Functions

	Name	Description
	WSDLPubFactory (see page 26044)	This is function WSDLPub.WSDLPubFactory.

Interfaces

	Name	Description
	IWSDLPublish (see page 26042)	This is class WSDLPub.IWSDLPublish.

Structs, Records, Enums

	Name	Description
	SOAPPublishOptions (see page 26045)	This is record WSDLPub.SOAPPublishOptions.

Types

Name	Description
TBeforePublishingWSDLEvent (see page 26046)	This is type WSDLPub.TBeforePublishingWSDLEvent.
TSOAPPublishOptions (see page 26046)	This is type WSDLPub.TSOAPPublishOptions.

193.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TWSDLHTMLPublish (see page 26024)	TWSDLHTMLPublish publishes a list of WSDL documents describing a Web Services application.
	TWSDLPublish (see page 26037)	This is class WSDLPub.TWSDLPublish.

193.1.1 TWSDLHTMLPublish Class

TWSDLHTMLPublish publishes a list of WSDL documents describing a Web Services application.

Class Hierarchy

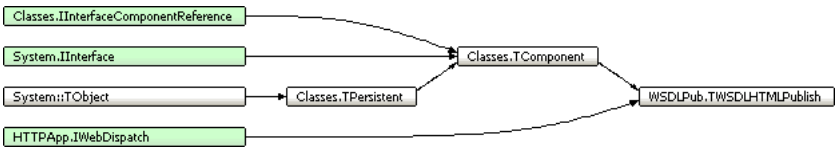

Pascal

```
TWSDLHTMLPublish = class(TComponent, IWebDispatch);
```

C++

```
class TWSDLHTMLPublish : public TComponent, public IWebDispatch;
```

File

WSDLPub

Description

Add TWSDLHTMLPublish to the Web module of a Web Service application to publish a list of WSDL documents that describe how to call that Web Service application. By publishing WSDL documents for your Web Service application, you allow arbitrary clients to call on your Web Services.

TWSDLHTMLPublish publishes definitions it derives from all the invokable interfaces that are registered with the invocation registry (InvRegistry), as well as from all remotable classes and types that are registered with the remotable type registry (RemTypeRegistry). By default, it uses the address where you install the application that contains the TWSDLHTMLPublish component as the address where those services are available. If you deploy the implementation of those Web Services elsewhere, you can enable the WSDL administrator (using the AdminEnabled (see page 26034) property), and then launch the WSDL administrator from a Web browser to change the addresses specified in the generated WSDL documents.

TWSDLHTMLPublish automatically registers itself with the Web module (or Web dispatcher) as an auto-dispatching object. This means that the Web module or Web dispatcher forwards all incoming HTTP messages directed to the WSDL publisher without requiring the use of Web action items. Those messages include requests for the list of WSDL documents and requests for the WSDL administrator.

See Also

TWebModule (see page 15154)

InvRegistry

RemTypeRegistry

Generating WSDL Documents for a Web Service Application

Using the SOAP Application Wizard

Writing Servers that Support Web Services

193.1.1.1 TWSDLHTMLPublish Members

The following tables list the members exposed by TWSDLHTMLPublish.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AddInterfaceList (see page 26029)	This is AddInterfaceList, a member of class TWSDLHTMLPublish.
	AddPortList (see page 26029)	This is AddPortList, a member of class TWSDLHTMLPublish.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetHostScriptBaseURL (see page 26031)	This is GetHostScriptBaseURL, a member of class TWSDLHTMLPublish.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.

	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdatePortList (see page 26033)	This is UpdatePortList, a member of class TWSDLHTMLPublish.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.
	WSILInfo (see page 26033)	Returns the WS inspection document that describes the Web Service.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	Assign (see page 2930)	Copies the contents of another, similar object.
	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 26028)	Creates an instance of TWSDLHTMLPublish (see page 26024).
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 26028)	Destroys the instance of a TWSDLHTMLPublish (see page 26024) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DispatchEnabled (see page 26030)	Indicates whether the WSDL publisher is enabled to respond to incoming messages.
	DispatchMask (see page 26030)	Returns a mask that matches the path portion of any URL to which this object responds.
	DispatchMethodType (see page 26030)	Returns the method type of all HTTP messages to which this component responds.
	DispatchRequest (see page 26031)	Responds to an incoming message and generates the appropriate response message.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.

	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetTargetNamespace (see page 26031)	Returns the value of the TargetNamespace (see page 26035) property.
	HandleRequest (see page 26032)	Generates the response for an incoming request message.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	ServiceInfo (see page 26032)	Responds to requests that are otherwise not handled by the application.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Enabled (see page 15232)	
R	Mask (see page 15233)	
R	MethodType (see page 15233)	This is MethodType, a member of class IWebDispatch.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	OnAfterPublishingWSDL (see page 26035)	Occurs after the publisher generates a WSDL document for the Web Service application.
	OnBeforePublishingTypes (see page 26036)	Occurs before the publisher starts adding type definitions to the WSDL document.
	OnBeforePublishingWSDL (see page 26036)	Occurs before the publisher generates its WSDL document.
	OnPublishingType (see page 26037)	Occurs before the publisher generates the schema for a given type.

Published Properties

	Name	Description
	AdminEnabled (see page 26034)	Enables or disables the WSDL administrator.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	PublishOptions (see page 26034)	Controls (see page 5711) the definitions TWSDLHTMLPublish (see page 26024) creates when it generates its WSDL documents.
	Tag (see page 2808)	Stores an integer value as part of a component.
	TargetNamespace (see page 26035)	Specifies the target namespace URL of the WSDL document.
	WebDispatch (see page 26035)	This is WebDispatch, a member of class TWSDLHTMLPublish.

193.1.1.2 TWSDLHTMLPublish.Create Constructor

Creates an instance of TWSDLHTMLPublish (see page 26024).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWSDLHTMLPublish(TComponent * AOwner);
```

Description

Call Create to instantiate a TWSDLHTMLPublish (see page 26024) object at runtime. TWSDLHTMLPublish (see page 26024) components added to web modules at design time are created automatically.

The AOwner parameter specifies another component (usually the web module) that is responsible for freeing the TWSDLHTMLPublish (see page 26024) instance. It becomes the value of the Owner property.

See Also

Owner (see page 2807)

193.1.1.3 TWSDLHTMLPublish.Destroy Destructor

Destroys the instance of a TWSDLHTMLPublish (see page 26024) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWSDLHTMLPublish();
```

Description

Use the Free method to free a TWSDLHTMLPublish (see page 26024) instance. Unlike the Destroy method, Free guards against a nil reference.

See Also

Free (see page 22300)

193.1.1.4 TWSDLHTMLPublish Methods

The methods of the TWSDLHTMLPublish class are listed here.

Protected Methods

	Name	Description
	AddInterfaceList (see page 26029)	This is AddInterfaceList, a member of class TWSDLHTMLPublish.
	AddPortList (see page 26029)	This is AddPortList, a member of class TWSDLHTMLPublish.
	GetHostScriptBaseURL (see page 26031)	This is GetHostScriptBaseURL, a member of class TWSDLHTMLPublish.
	UpdatePortList (see page 26033)	This is UpdatePortList, a member of class TWSDLHTMLPublish.
	WSILInfo (see page 26033)	Returns the WS inspection document that describes the Web Service.

Public Methods

	Name	Description
	DispatchEnabled (see page 26030)	Indicates whether the WSDL publisher is enabled to respond to incoming messages.
	DispatchMask (see page 26030)	Returns a mask that matches the path portion of any URL to which this object responds.
	DispatchMethodType (see page 26030)	Returns the method type of all HTTP messages to which this component responds.
	DispatchRequest (see page 26031)	Responds to an incoming message and generates the appropriate response message.
	GetTargetNamespace (see page 26031)	Returns the value of the TargetNamespace (see page 26035) property.
	HandleRequest (see page 26032)	Generates the response for an incoming request message.
	ServiceInfo (see page 26032)	Responds to requests that are otherwise not handled by the application.

193.1.1.4.1 TWSDLHTMLPublish.AddInterfaceList Method

Pascal

```
procedure AddInterfaceList(htmldoc: TStringList; const WSDLBaseURL: String);
```

C++

```
__fastcall AddInterfaceList(TStringList htmldoc, const AnsiString WSDLBaseURL);
```

Description

This is AddInterfaceList, a member of class TWSDLHTMLPublish.

193.1.1.4.2 TWSDLHTMLPublish.AddPortList Method

Pascal

```
procedure AddPortList(htmldoc: TStringList; const PortType: String);
```

C++

```
__fastcall AddPortList(TStringList htmldoc, const AnsiString PortType);
```

Description

This is AddPortList, a member of class TWSDLHTMLPublish.

193.1.1.4.3 TWSDLHTMLPublish.DispatchEnabled Method

Indicates whether the WSDL publisher is enabled to respond to incoming messages.

Pascal

```
function DispatchEnabled: Boolean;
```

C++

```
__fastcall Boolean DispatchEnabled();
```

Description

The DispatchEnabled method must be implemented by all auto-dispatching objects to allow Web Module (or Web Dispatcher) to determine whether it can receive incoming messages. In TWSDLHTMLPublish (see page 26024), DispatchEnabled always returns true.

See Also

webdispatch (see page 26035)

193.1.1.4.4 TWSDLHTMLPublish.DispatchMask Method

Returns a mask that matches the path portion of any URL to which this object responds.

Pascal

```
function DispatchMask: TMask;
```

C++

```
__fastcall TMask DispatchMask();
```

Description

The DispatchMask method must be implemented by all auto-dispatching objects to allow Web Module (or Web Dispatcher) to identify the HTTP messages it should forward to the this component. It returns a Mask (see page 15797) derived from the Mask (see page 15797) property of the WebDispatch (see page 26035) property.

See Also

webdispatch (see page 26035)

Mask (see page 1482)

193.1.1.4.5 TWSDLHTMLPublish.DispatchMethodType Method

Returns the method type of all HTTP messages to which this component responds.

Pascal

```
function DispatchMethodType: TMethodType;
```

C++

```
__fastcall TMethodType DispatchMethodType();
```

Description

The DispatchMethodType method must be implemented by all auto-dispatching objects to allow Web Module (or Web Dispatcher) to identify the HTTP messages it should forward to this component. It returns the MethodType properties of WebDispatch (see page 26035).

See Also

MethodType

webdispatch (see page 26035)

193.1.1.4.6 TWSDLHTMLPublish.DispatchRequest Method

Responds to an incoming message and generates the appropriate response message.

Pascal

```
function DispatchRequest(Sender: TObject; Request: TWebRequest; Response: TWebResponse):
Boolean;
```

C++

```
__fastcall Boolean DispatchRequest(TObject * Sender, TWebRequest Request, TWebResponse
Response);
```

Description

The Web Module (or Web Dispatcher) calls DispatchRequest when it identifies an HTTP message that matches the properties of the WebDispatch (see page 26035) property. DispatchRequest responds by generating an HTML page that lists the WSDL documents describing the Web Service, and including a button for the WSDL administrator if the AdminEnabled (see page 26034) property is true.

Sender is not used in the DispatchRequest method.

Request contains information about the incoming request message.

Response is a TWebResponse object that is filled out to indicate the response that should be returned to the Web client.

DispatchRequest returns true if it successfully generates a response. It returns false if it could not handle the request.

See Also

webdispatch (see page 26035)

AdminEnabled (see page 26034)

193.1.1.4.7 TWSDLHTMLPublish.GetHostScriptBaseURL Method

Pascal

```
function GetHostScriptBaseURL(Request: TWebRequest): String;
```

C++

```
__fastcall AnsiString GetHostScriptBaseURL(TWebRequest Request);
```

Description

This is GetHostScriptBaseURL, a member of class TWSDLHTMLPublish.

193.1.1.4.8 TWSDLHTMLPublish.GetTargetNamespace Method

Returns the value of the TargetNamespace (see page 26035) property.

Pascal

```
function GetTargetNamespace: WideString;
```

C++

```
__fastcall BSTR GetTargetNamespace();
```

Description

GetTargetNamespace is the read implementation for the TargetNamespace (see page 26035) property.

See Also

TargetNamespace (see page 26035)

193.1.1.4.9 TWSDLHTMLPublish.HandleRequest Method

Generates the response for an incoming request message.

Pascal

```
function HandleRequest(Resp: TStringList; const Path: String; const HostScriptBaseURL: String;
var ContentType: String; Request: TWebRequest): integer;
```

C++

```
__fastcall int HandleRequest(TStringList Resp, const AnsiString Path, const AnsiString
HostScriptBaseURL, AnsiString ContentType, TWebRequest Request);
```

Description

The DispatchRequest (see page 26031) method calls HandleRequest to generate the content of the HTTP response message it fills out. Typically, applications should not need to call this method.

Resp is a string list to which HandleRequest writes the content of the response.

Path is the path information taken from the incoming request message.

HostScriptBaseURL is the URL for the script name to which the incoming request is addressed.

ContentType returns the content type of the response that is written to Resp. This becomes the value of a ContentType header on the response message.

Request is the request message for which this method generates a response.

HandleRequest returns the StatusCode for the generated response. This becomes the StatusCode property of the response message.

See Also

DispatchRequest (see page 26031)

StatusCode (see page 15215)

193.1.1.4.10 TWSDLHTMLPublish.ServiceInfo Method

Responds to requests that are otherwise not handled by the application.

Pascal

```
procedure ServiceInfo(Sender: TObject; Request: TWebRequest; Response: TWebResponse; var
Handled: Boolean);
```

C++

```
__fastcall ServiceInfo(TObject * Sender, TWebRequest Request, TWebResponse Response, Boolean
Handled);
```

Description

The Web Module (or Web Dispatcher) calls ServiceInfo when there are no other components in the application to handle an incoming request message. ServiceInfo responds by generating a default Web page that includes a link to the WSDL documents.

Sender is not used in the ServiceInfo method.

Request contains information about the incoming request message.

Response is a TWebResponse object that is filled out to indicate the response that should be returned to the Web client.

Handled always returns true, indicating that ServiceInfo successfully generated a response.

See Also

DispatchRequest (see page 26031)

193.1.1.4.11 TWSDLHTMLPublish.UpdatePortList Method

Pascal

```
procedure UpdatePortList(PortList: TStrings; const PortType: String; const Command: String);
```

C++

```
__fastcall UpdatePortList(TStrings PortList, const AnsiString PortType, const AnsiString
Command);
```

Description

This is UpdatePortList, a member of class TWSDLHTMLPublish.

193.1.1.4.12 TWSDLHTMLPublish.WSILInfo Method

Returns the WS inspection document that describes the Web Service.

Pascal

```
procedure WSILInfo(const HostScriptBaseURL: string; Request: TWebRequest; Response:
TWebResponse; var Handled: Boolean);
```

C++

```
__fastcall WSILInfo(const AnsiString HostScriptBaseURL, TWebRequest Request, TWebResponse
Response, Boolean Handled);
```

Description

The Web Module (or Web Dispatcher) calls WSILInfo when a request is made for the WS inspection document for the Web Service (when the path portion of the request ends in the string 'inspection.wisl'.. WSILInfo responds by generating the WSIL description.

HostScriptBaseURL is the base path for accessing the WSDL documents of the Web Service.

Request contains information about the incoming request message.

Response is a TWebResponse object that is filled out to indicate the response that should be returned to the Web client.

Handled always returns true, indicating that WSILInfo successfully generated a response.

See Also

DispatchRequest (see page 26031)

ServiceInfo (see page 26032)

193.1.1.5 TWSDLHTMLPublish Properties

The properties of the TWSDLHTMLPublish class are listed here.

Published Properties

	Name	Description
	AdminEnabled (see page 26034)	Enables or disables the WSDL administrator.
	PublishOptions (see page 26034)	Controls (see page 5711) the definitions TWSDLHTMLPublish (see page 26024) creates when it generates its WSDL documents.
	TargetNamespace (see page 26035)	Specifies the target namespace URL of the WSDL document.
	WebDispatch (see page 26035)	This is WebDispatch, a member of class TWSDLHTMLPublish.

193.1.1.5.1 TWSDLHTMLPublish.AdminEnabled Property

Enables or disables the WSDL administrator.

Pascal

```
property AdminEnabled: Boolean;
```

C++

```
__property Boolean AdminEnabled;
```

Description

Use AdminEnabled to indicate whether this application implements the WSDL administrator. When AdminEnabled is true, pointing a Web browser at the URL that addresses this application using the path info specified by WebDispatch (see page 26035) launches a Web page that includes a button for the WSDL administrator. The WSDL administrator can then adjust the published WSDL documents to indicate an address other than the one where this application is installed as the location or locations where the Web Services are deployed.

When AdminEnabled is false, the URL that includes WebDispatch (see page 26035) publishes a list of WSDL documents but does not provide access to the WSDL administrator.

Tip: You can set AdminEnabled to true, update the URLs in the resulting WSDL documents, and then rebuild your application, setting AdminEnabled to false. The WSDL administrator stores your changes in a .ini file so that they appear in the list of WSDL documents even after you have changed AdminEnabled.

See Also

webdispatch (see page 26035)

TargetNamespace (see page 26035)

Generating WSDL Documents for a Web Service Application

193.1.1.5.2 TWSDLHTMLPublish.PublishOptions Property

Controls (see page 5711) the definitions TWSDLHTMLPublish (see page 26024) creates when it generates its WSDL documents.

Pascal

```
property PublishOptions: TSOAPPublishOptions;
```

C++

```
__property TSOAPPublishOptions PublishOptions;
```

Description

PublishOptions is a set of flags that customize the generated WSDL document.

See Also

TargetNamespace (see page 26035)

193.1.1.5.3 TWSDLHTMLPublish.TargetNamespace Property

Specifies the target namespace URL of the WSDL document.

Pascal

```
property TargetNamespace: WideString;
```

C++

```
__property BSTR TargetNamespace;
```

Description

Use TargetNamespace to specify a namespace for the WSDL documents this component publishes. This is the namespace URL that the WSDL document uses for the definitions it publishes.

See Also

AdminEnabled (see page 26034)

193.1.1.5.4 TWSDLHTMLPublish.WebDispatch Property

Pascal

```
property WebDispatch: TWebDispatch;
```

C++

```
__property TWebDispatch WebDispatch;
```

Description

This is WebDispatch, a member of class TWSDLHTMLPublish.

193.1.1.6 TWSDLHTMLPublish Events

The events of the TWSDLHTMLPublish class are listed here.

Published Events

	Name	Description
	OnAfterPublishingWSDL (see page 26035)	Occurs after the publisher generates a WSDL document for the Web Service application.
	OnBeforePublishingTypes (see page 26036)	Occurs before the publisher starts adding type definitions to the WSDL document.
	OnBeforePublishingWSDL (see page 26036)	Occurs before the publisher generates its WSDL document.
	OnPublishingType (see page 26037)	Occurs before the publisher generates the schema for a given type.

193.1.1.6.1 TWSDLHTMLPublish.OnAfterPublishingWSDL Event

Occurs after the publisher generates a WSDL document for the Web Service application.

Pascal

```
property OnAfterPublishingWSDL: TAfterPublishingWSDLEvent;
```

C++

```
__property TAfterPublishingWSDLEvent OnAfterPublishingWSDL;
```

Description

Use OnAfterPublishingWSDL to perform special actions after the publisher has generated a WSDL document. For example, you can use this event to cache the generated WSDL document and then re-use the cached document in an OnBeforePublishingWSDL (see page 26036) event rather than requiring the publisher to generate a new document for every WSDL request it receives.

See Also

OnBeforePublishingWSDL (see page 26036)

193.1.1.6.2 TWSDLHTMLPublish.OnBeforePublishingTypes Event

Occurs before the publisher starts adding type definitions to the WSDL document.

Pascal

```
property OnBeforePublishingTypes: TBeforePublishingTypesEvent;
```

C++

```
__property TBeforePublishingTypesEvent OnBeforePublishingTypes;
```

Description

Use OnBeforePublishingTypes to add your own type definitions to the WSDL document before the publisher adds definitions for the types that the invokable interfaces use and types that are registered with the remotable type registry. If the OnBeforePublishingTypes event handler adds a schema definition for a type, the publisher uses that definition rather than generating its own version.

See Also

OnBeforePublishingWSDL (see page 26036)

OnPublishingType (see page 26037)

193.1.1.6.3 TWSDLHTMLPublish.OnBeforePublishingWSDL Event

Occurs before the publisher generates its WSDL document.

Pascal

```
property OnBeforePublishingWSDL: TBeforePublishingWSDLEvent;
```

C++

```
__property TBeforePublishingWSDLEvent OnBeforePublishingWSDL;
```

Description

Use OnBeforePublishingWSDL to provide your own WSDL document rather than letting the publisher generate one dynamically. For example, the event handler can supply a cached version of the WSDL document that you saved previously from an OnAfterPublishingWSDL (see page 26035) event handler.

See Also

OnAfterPublishingWSDL (see page 26035)

OnBeforePublishingTypes (see page 26036)

193.1.1.6.4 TWSDLHTMLPublish.OnPublishingType Event

Occurs before the publisher generates the schema for a given type.

Pascal

```
property OnPublishingType: TPublishingTypeEvent;
```

C++

```
__property TPublishingTypeEvent OnPublishingType;
```

Description

Use OnPublishingType to perform specific actions immediately before the publisher adds a type definition to the emerging WSDL document.

See Also

OnBeforePublishingTypes (see page 26036)

193.1.2 TWSDLPublish Class

Class Hierarchy

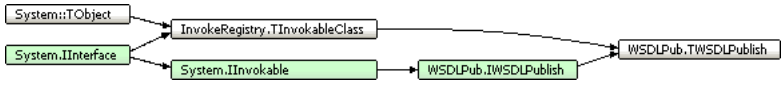

Pascal

```
TWSDLPublish = class(TInvokableClass, IWSDLPublish);
```

C++

```
class TWSDLPublish : public TInvokableClass, public IWSDLPublish;
```

File

WSDLPub

Description

This is class WSDLPub.TWSDLPublish.

193.1.2.1 TWSDLPublish Members

The following tables list the members exposed by TWSDLPublish.

Protected Fields

	Name	Description
	FRefCount (see page 15522)	This is FRefCount, a member of class TInvokableClass.
	FSOAPHeaders (see page 15523)	This is FSOAPHeaders, a member of class TInvokableClass.

Protected Methods

	Name	Description
	_AddRef (see page 15523)	Increments the reference count for this interface.

	_Release (see page 15524)	Decrements the reference count for this interface.
	QueryInterface (see page 15525)	Implements the IInterface QueryInterface method.

Public Events

	Name	Description
	OnAfterPublishingWSDL (see page 26041)	This is OnAfterPublishingWSDL, a member of class TWSDLPublish.
	OnBeforePublishingTypes (see page 26041)	This is OnBeforePublishingTypes, a member of class TWSDLPublish.
	OnBeforePublishingWSDL (see page 26041)	This is OnBeforePublishingWSDL, a member of class TWSDLPublish.
	OnPublishingType (see page 26041)	This is OnPublishingType, a member of class TWSDLPublish.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 15524)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 15524)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 15522)	Creates and initializes a TInvokableClass (see page 15520) object.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetPortTypeEntries (see page 26039)	This is GetPortTypeEntries, a member of class TWSDLPublish.
	GetPortTypeList (see page 26039)	This is GetPortTypeList, a member of class TWSDLPublish.
	GetTypeSystemsList (see page 26039)	This is GetTypeSystemsList, a member of class TWSDLPublish.
	GetWSDLForPortType (see page 26040)	This is GetWSDLForPortType, a member of class TWSDLPublish.
	GetXSDForTypeSystem (see page 26040)	This is GetXSDForTypeSystem, a member of class TWSDLPublish.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 15525)	Allocates memory for each instance of an invokable object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	RefCount (see page 15526)	Indicates the number of interface pointers currently dependent upon the object.
	TargetNamespace (see page 26040)	This is TargetNamespace, a member of class TWSDLPublish.

193.1.2.2 TWSDLPublish Methods

The methods of the TWSDLPublish class are listed here.

Public Methods

	Name	Description
	GetPortTypeEntries (see page 26039)	This is GetPortTypeEntries, a member of class TWSDLPublish.
	GetPortTypeList (see page 26039)	This is GetPortTypeList, a member of class TWSDLPublish.
	GetTypeSystemsList (see page 26039)	This is GetTypeSystemsList, a member of class TWSDLPublish.
	GetWSDLForPortType (see page 26040)	This is GetWSDLForPortType, a member of class TWSDLPublish.
	GetXSDForTypeSystem (see page 26040)	This is GetXSDForTypeSystem, a member of class TWSDLPublish.

193.1.2.2.1 TWSDLPublish.GetPortTypeEntries Method

Pascal

```
procedure GetPortTypeEntries(var Entries: TInvRegIntfEntryArray);
```

C++

```
__fastcall GetPortTypeEntries(TInvRegIntfEntryArray Entries);
```

Description

This is GetPortTypeEntries, a member of class TWSDLPublish.

193.1.2.2.2 TWSDLPublish.GetPortTypeList Method

Pascal

```
function GetPortTypeList: TWideStringDynArray; stdcall;
```

C++

```
__fastcall __stdcall TWideStringDynArray GetPortTypeList();
```

Description

This is GetPortTypeList, a member of class TWSDLPublish.

193.1.2.2.3 TWSDLPublish.GetTypeSystemsList Method

Pascal

```
function GetTypeSystemsList: TWideStringDynArray; stdcall;
```

C++

```
__fastcall __stdcall TWideStringDynArray GetTypeSystemsList();
```

Description

This is GetTypeSystemsList, a member of class TWSDLPublish.

193.1.2.2.4 TWSDLPublish.GetWSDLForPortType Method

Pascal

```
function GetWSDLForPortType(const PortType: WideString): WideString; stdcall;
```

C++

```
__fastcall __stdcall BSTR GetWSDLForPortType(const BSTR PortType);
```

Description

This is GetWSDLForPortType, a member of class TWSDLPublish.

193.1.2.2.5 TWSDLPublish.GetXSDForTypeSystem Method

Pascal

```
function GetXSDForTypeSystem(const TypeSystem: WideString): WideString; stdcall;
```

C++

```
__fastcall __stdcall BSTR GetXSDForTypeSystem(const BSTR TypeSystem);
```

Description

This is GetXSDForTypeSystem, a member of class TWSDLPublish.

193.1.2.3 TWSDLPublish Properties

The properties of the TWSDLPublish class are listed here.

Public Properties

	Name	Description
	TargetNamespace (see page 26040)	This is TargetNamespace, a member of class TWSDLPublish.

193.1.2.3.1 TWSDLPublish.TargetNamespace Property

Pascal

```
property TargetNamespace: WideString;
```

C++

```
__property BSTR TargetNamespace;
```

Description

This is TargetNamespace, a member of class TWSDLPublish.

193.1.2.4 TWSDLPublish Events

The events of the TWSDLPublish class are listed here.

Public Events

	Name	Description
	OnAfterPublishingWSDL (see page 26041)	This is OnAfterPublishingWSDL, a member of class TWSDLPublish.
	OnBeforePublishingTypes (see page 26041)	This is OnBeforePublishingTypes, a member of class TWSDLPublish.
	OnBeforePublishingWSDL (see page 26041)	This is OnBeforePublishingWSDL, a member of class TWSDLPublish.

	OnPublishingType (see page 26041)	This is OnPublishingType, a member of class TWSDLPublish.

193.1.2.4.1 TWSDLPublish.OnAfterPublishingWSDL Event

Pascal

```
property OnAfterPublishingWSDL: TAfterPublishingWSDLEvent;
```

C++

```
__property TAfterPublishingWSDLEvent OnAfterPublishingWSDL;
```

Description

This is OnAfterPublishingWSDL, a member of class TWSDLPublish.

193.1.2.4.2 TWSDLPublish.OnBeforePublishingTypes Event

Pascal

```
property OnBeforePublishingTypes: TBeforePublishingTypesEvent;
```

C++

```
__property TBeforePublishingTypesEvent OnBeforePublishingTypes;
```

Description

This is OnBeforePublishingTypes, a member of class TWSDLPublish.

193.1.2.4.3 TWSDLPublish.OnBeforePublishingWSDL Event

Pascal

```
property OnBeforePublishingWSDL: TBeforePublishingWSDLEvent;
```

C++

```
__property TBeforePublishingWSDLEvent OnBeforePublishingWSDL;
```

Description

This is OnBeforePublishingWSDL, a member of class TWSDLPublish.

193.1.2.4.4 TWSDLPublish.OnPublishingType Event

Pascal

```
property OnPublishingType: TPublishingTypeEvent;
```

C++

```
__property TPublishingTypeEvent OnPublishingType;
```

Description

This is OnPublishingType, a member of class TWSDLPublish.

193.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IWSDLPublish (see page 26042)	This is class WSDLPub.IWSDLPublish.

193.2.1 IWSDLPublish Interface

Class Hierarchy

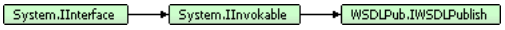

Pascal

```
IWSDLPublish = interface(IInvokable);
```

C++

```
__interface IWSDLPublish : public IInvokable;
```

File

WSDLPub

Description

This is class WSDLPub.IWSDLPublish.

193.2.1.1 IWSDLPublish Members

The following tables list the members exposed by IWSDLPublish.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	GetPortTypeList (see page 26043)	This is GetPortTypeList, a member of class IWSDLPublish.
	GetTypeSystemsList (see page 26043)	This is GetTypeSystemsList, a member of class IWSDLPublish.
	GetWSDLForPortType (see page 26043)	This is GetWSDLForPortType, a member of class IWSDLPublish.
	GetXSDForTypeSystem (see page 26043)	This is GetXSDForTypeSystem, a member of class IWSDLPublish.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.

193.2.1.2 IWSDLPublish Methods

The methods of the IWSDLPublish class are listed here.

Public Methods

	Name	Description
	GetPortTypeList (see page 26043)	This is GetPortTypeList, a member of class IWSDLPublish.
	GetTypeSystemsList (see page 26043)	This is GetTypeSystemsList, a member of class IWSDLPublish.
	GetWSDLForPortType (see page 26043)	This is GetWSDLForPortType, a member of class IWSDLPublish.
	GetXSDForTypeSystem (see page 26043)	This is GetXSDForTypeSystem, a member of class IWSDLPublish.

193.2.1.2.1 IWSDLPublish.GetPortTypeList Method

Pascal

```
function GetPortTypeList: TWideStringDynArray; stdcall;
```

C++

```
__fastcall __stdcall TWideStringDynArray GetPortTypeList();
```

Description

This is GetPortTypeList, a member of class IWSDLPublish.

193.2.1.2.2 IWSDLPublish.GetTypeSystemsList Method

Pascal

```
function GetTypeSystemsList: TWideStringDynArray; stdcall;
```

C++

```
__fastcall __stdcall TWideStringDynArray GetTypeSystemsList();
```

Description

This is GetTypeSystemsList, a member of class IWSDLPublish.

193.2.1.2.3 IWSDLPublish.GetWSDLForPortType Method

Pascal

```
function GetWSDLForPortType(const PortType: WideString): WideString; stdcall;
```

C++

```
__fastcall __stdcall BSTR GetWSDLForPortType(const BSTR PortType);
```

Description

This is GetWSDLForPortType, a member of class IWSDLPublish.

193.2.1.2.4 IWSDLPublish.GetXSDForTypeSystem Method

Pascal

```
function GetXSDForTypeSystem(const TypeSystem: WideString): WideString; stdcall;
```

C++

```
__fastcall __stdcall BSTR GetXSDForTypeSystem(const BSTR TypeSystem);
```

Description

This is GetXSDForTypeSystem, a member of class IWSDLPublish.

193.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	WSDLPubFactory (see page 26044)	This is function WSDLPub.WSDLPubFactory.

193.3.1 WSDLPub.WSDLPubFactory Function

Pascal

```
procedure WSDLPubFactory(out obj: TObject);
```

C++

```
WSDLPubFactory(TObject * obj);
```

File

WSDLPub

Description

This is function WSDLPub.WSDLPubFactory.

193.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	SOAPPublishOptions (see page 26045)	This is record WSDLPub.SOAPPublishOptions.

193.4.1 WSDLPub.SOAPPublishOptions Enumeration

Pascal

```
SOAPPublishOptions = (
  poDefault,
  poPublishLocationAsSecure
);
```

C++

```
enum SOAPPublishOptions {
  poDefault,
  poPublishLocationAsSecure
};
```

File

WSDLPub

Description

This is record WSDLPub.SOAPPublishOptions.

193.5 Types

The following table lists types in this documentation.

Types

Name	Description
TBeforePublishingWSDLEvent (see page 26046)	This is type WSDLPub.TBeforePublishingWSDLEvent.
TSOAPPublishOptions (see page 26046)	This is type WSDLPub.TSOAPPublishOptions.

193.5.1 WSDLPub.TBeforePublishingWSDLEvent Type

Pascal

```
TBeforePublishingWSDLEvent = procedure (const IntfName: WideString; var WSDL: WideString; var
Handled: Boolean) of object;
```

C++

```
(const IntfName: WideString; var WSDL: WideString; var Handled: Boolean) (
TBeforePublishingWSDLEvent)();
```

File

WSDLPub

Description

This is type WSDLPub.TBeforePublishingWSDLEvent.

193.5.2 WSDLPub.TSOAPPublishOptions Type

Pascal

```
TSOAPPublishOptions = set of SOAPPublishOptions;
```

C++

```
SOAPPublishOptions TSOAPPublishOptions;
```

File

WSDLPub

Description

This is type WSDLPub.TSOAPPublishOptions.

194 XMLBrokr Namespace

This is namespace XMLBrokr.

Topics

Name	Description
Classes (see page 26049)	The following table lists classes in this documentation.
Interfaces (see page 26086)	The following table lists interfaces in this documentation.
Functions (see page 26088)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 26089)	The following table lists structs, records, enums in this documentation.
Types (see page 26090)	The following table lists types in this documentation.
Constants (see page 26092)	The following table lists constants in this documentation.

Classes

	Name	Description
	TXMLBroker (see page 26049)	TXMLBroker fetches XML data packets from an application server and applies XML updates to the application server.
	TXMLParams (see page 26081)	TXMLParams manages a list of parameters for an XML broker.

Constants

Name	Description
sPostDelta (see page 26092)	This is constant XMLBrokr.sPostDelta.
sProducer (see page 26092)	This is constant XMLBrokr.sProducer.
sRedirect (see page 26092)	This is constant XMLBrokr.sRedirect.

Functions

	Name	Description
	FormatXML (see page 26088)	This is function XMLBrokr.FormatXML.

Interfaces

	Name	Description
	INotifyConnectionChange (see page 26086)	This is class XMLBrokr.INotifyConnectionChange.

Structs, Records, Enums

	Name	Description
	TXMLOption (see page 26089)	This is record XMLBrokr.TXMLOption.

Types

Name	Description
TGetErrorResponseEvent (see page 26090)	This is type XMLBrokr.TGetErrorResponseEvent.
TGetResponseEvent (see page 26090)	This is type XMLBrokr.TGetResponseEvent.
TRequestRecordsEvent (see page 26091)	This is type XMLBrokr.TRequestRecordsEvent.
TRequestUpdateEvent (see page 26091)	This is type XMLBrokr.TRequestUpdateEvent.
TXMLOptions (see page 26091)	This is type XMLBrokr.TXMLOptions.

194.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TXMLBroker (see page 26049)	TXMLBroker fetches XML data packets from an application server and applies XML updates to the application server.
	TXMLParams (see page 26081)	TXMLParams manages a list of parameters for an XML broker.

194.1.1 TXMLBroker Class

TXMLBroker fetches XML data packets from an application server and applies XML updates to the application server.

Class Hierarchy

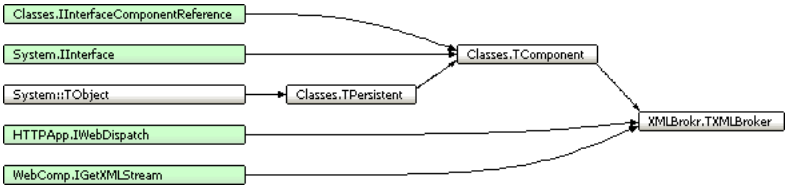

Pascal

```
TXMLBroker = class(TComponent, IWebDispatch, IGetXMLStream);
```

C++

```
class TXMLBroker : public TComponent, public IWebDispatch, public IGetXMLStream;
```

File

XMLBrokr

Description

Use TXMLBroker in an application that is simultaneously the client of a multi-tiered database application and a Web server application. This class serves two major functions:

It fetches XML data packets from a provider using the IAppServer interface.

It receives HTTP messages from Web browsers that contain XML delta packets and applies them to the application server.

Add a TXMLBroker object to a Web module to make database information supplied by a provider on the application server available as XML. An InternetExpress page producer (TInetXPageProducer) in the same Web module uses this XML to build HTML pages that represent that database information.

The XML broker automatically registers itself with the Web module (or Web dispatcher) as an auto-dispatching object. This means that the Web module or Web dispatcher forwards all incoming HTTP messages directed to the XML broker without requiring the use of Web action items.

These incoming messages are assumed to be update messages created by the browser in response to HTML generated by a TApplyUpdatesButton component. The XML broker automatically applies the delta packet (the content of the HTTP request message) to the application server, and passes any update errors to a separate content producer, which can generate a

response message.

See Also

TRemoteDataModule (see page 6585)

IAppServer (see page 16450)

TInetXPageProducer (see page 17145)

TApplyUpdatesButton (see page 16480)

Building an InternetExpress Application

194.1.1.1 TXMLBroker Members

The following tables list the members exposed by TXMLBroker.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	AS_ApplyUpdates (see page 26056)	This is AS_ApplyUpdates, a member of class TXMLBroker.
	AS_FetchParams (see page 26057)	This is AS_FetchParams, a member of class TXMLBroker.
	AS_GetRecords (see page 26057)	This is AS_GetRecords, a member of class TXMLBroker.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckInactive (see page 26057)	This is CheckInactive, a member of class TXMLBroker.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DispatchEnabled (see page 26057)	
	DispatchMask (see page 26058)	
	DispatchMethodType (see page 26058)	
	DispatchRequest (see page 26058)	
	GetAppServer (see page 26059)	This is GetAppServer, a member of class TXMLBroker.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetConnected (see page 26059)	This is GetConnected, a member of class TXMLBroker.
	GetErrorResponse (see page 26060)	This is GetErrorResponse, a member of class TXMLBroker.
	GetHasAppServer (see page 26060)	This is GetHasAppServer, a member of class TXMLBroker.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetNotify (see page 26060)	This is GetNotify, a member of class TXMLBroker.
	GetNotifyCount (see page 26061)	This is GetNotifyCount, a member of class TXMLBroker.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetProducerName (see page 26061)	This is GetProducerName, a member of class TXMLBroker.
	GetRedirect (see page 26061)	This is GetRedirect, a member of class TXMLBroker.

	GetResponse (see page 26061)	This is GetResponse, a member of class TXMLBroker.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetXMLStream (see page 26062)	This is GetXMLStream, a member of class TXMLBroker.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 26064)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SendConnectEvent (see page 26067)	This is SendConnectEvent, a member of class TXMLBroker.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppServer (see page 26067)	This is SetAppServer, a member of class TXMLBroker.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetConnected (see page 26067)	This is SetConnected, a member of class TXMLBroker.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParams (see page 26067)	This is SetParams, a member of class TXMLBroker.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetProviderName (see page 26068)	This is SetProviderName, a member of class TXMLBroker.
	SetReconcileProducer (see page 26068)	This is SetReconcileProducer, a member of class TXMLBroker.
	SetRemoteServer (see page 26069)	This is SetRemoteServer, a member of class TXMLBroker.
	SetWebDispatch (see page 26069)	This is SetWebDispatch, a member of class TXMLBroker.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddNotify (see page 26055)	Adds an object to the Notify (see page 26073) property list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	ApplyXMLUpdates (see page 26056)	Sends an XML delta packet of updated, inserted, and deleted records to the provider on the application server for writing to the database.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 26054)	Creates a new TXMLBroker (see page 26049) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 26054)	Frees the memory associated with the TXMLBroker (see page 26049) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
V	FetchParams (see page 26058)	Retrieves parameter values from the application server.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetDelta (see page 26059)	Extracts an XML delta packet from an HTTP request message.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetErrorCount (see page 26059)	This is GetErrorCount, a member of class TXMLBroker.
	GetErrors (see page 26060)	Returns the XML delta packet (if any) that describes the errors that occurred when applying updates.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetXMLRecords (see page 26062)	Retrieves an XML data packet from the application server.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HTMLSubmitFormName (see page 26063)	Returns the name of the HTML form that submits updates to the XML broker.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MasterRowSetVarName (see page 26063)	Returns the name of the javascript variable for the XML broker's dataset.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	RemoveNotify (see page 26064)	Removes an object to the Notify (see page 26073) property list.
V	RequestRecords (see page 26064)	Returns an XML data packet from the application server.
V	RequestUpdate (see page 26065)	Responds to an HTTP update request.

	RowSetVarName (see page 26066)	Returns the name of the javascript variable for the XML broker's dataset.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetProvider (see page 26068)	Associates the XML broker with a provider in the same process space.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SubmitFormVarName (see page 26069)	Returns the name of the variable that refers to the HTML form named by HTMLSubmitFormName (see page 26063).
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	AppServer (see page 26070)	Provides access to the interface the XML broker uses to communicate with the application server.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Enabled (see page 15232)	
R	HasAppServer (see page 26071)	Indicates whether the XML broker has an IAppServer interface for communicating with its provider.
R	Mask (see page 15233)	
R	MethodType (see page 15233)	This is MethodType, a member of class IWebDispatch.
R	Notify (see page 26073)	Lists the objects that are notified when any properties affecting the XML broker's connection to its provider change.
R	NotifyCount (see page 26073)	Indicates the number of objects in the Notify (see page 26073) property list.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterDispatch (see page 26077)	Occurs after the XML broker generates the content of a response message.
	BeforeDispatch (see page 26078)	Occurs when the XML broker receives an HTTP request message.
	OnGetErrorResponse (see page 26078)	Occurs when there are errors applying updates to the application server.
	OnGetResponse (see page 26079)	Occurs when there are no errors applying updates to the application server.
	OnRequestRecords (see page 26080)	Occurs when the XML broker needs to fetch a data packet from its associated provider.
	OnRequestUpdate (see page 26080)	Occurs immediately before the XML broker applies updates to the application server.

Published Properties

	Name	Description
	Connected (see page 26070)	Determines whether the XML broker has an active connection to the provider that supplies its data and applies its updates.
	MaxErrors (see page 26071)	Specifies the maximum number of update errors allowed before aborting an update operations.
	MaxRecords (see page 26072)	Specifies the maximum number of records that can appear in XML data packets.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Params (see page 26073)	Contains parameter values that are sent to or retrieved from the application server.
	ProviderName (see page 26074)	Specifies the name of a registered provider from the application server.
	ReconcileProducer (see page 26075)	Specifies the producer that generates response message content when an update generates errors.

	RemoteServer (see page 26076)	Specifies the connection component that this XML broker uses to connect to an application server.
	Tag (see page 2808)	Stores an integer value as part of a component.
	WebDispatch (see page 26076)	Lets the Web dispatcher identify what HTTP messages should be forwarded to the XML broker.

194.1.1.2 TXMLBroker.Create Constructor

Creates a new TXMLBroker (see page 26049) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TXMLBroker(TComponent * AOwner);
```

Description

Most applications do not need to call the XML broker constructor. Typically the XML broker is added to a Web module at design time, so that the constructor is called automatically.

The Owner parameter specifies the component (usually the Web module) that is responsible for freeing the XML broker.

See Also

Owner (see page 2807)

194.1.1.3 TXMLBroker.Destroy Destructor

Frees the memory associated with the TXMLBroker (see page 26049) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLBroker();
```

Description

Do not call Destroy directly. Instead,call Free, which verifies that the XML broker is not nil before calling Destroy.

See Also

Free (see page 22300)

194.1.1.4 TXMLBroker Methods

The methods of the TXMLBroker class are listed here.

Protected Methods

	Name	Description
	AS_ApplyUpdates (see page 26056)	This is AS_ApplyUpdates, a member of class TXMLBroker.
	AS_FetchParams (see page 26057)	This is AS_FetchParams, a member of class TXMLBroker.
	AS_GetRecords (see page 26057)	This is AS_GetRecords, a member of class TXMLBroker.
	CheckInactive (see page 26057)	This is CheckInactive, a member of class TXMLBroker.
	DispatchEnabled (see page 26057)	

	DispatchMask (see page 26058)	
	DispatchMethodType (see page 26058)	
	DispatchRequest (see page 26058)	
	GetAppServer (see page 26059)	This is GetAppServer, a member of class TXMLBroker.
	GetConnected (see page 26059)	This is GetConnected, a member of class TXMLBroker.
	GetErrorResponse (see page 26060)	This is GetErrorResponse, a member of class TXMLBroker.
	GetHasAppServer (see page 26060)	This is GetHasAppServer, a member of class TXMLBroker.
	GetNotify (see page 26060)	This is GetNotify, a member of class TXMLBroker.
	GetNotifyCount (see page 26061)	This is GetNotifyCount, a member of class TXMLBroker.
	GetProducerName (see page 26061)	This is GetProducerName, a member of class TXMLBroker.
	GetRedirect (see page 26061)	This is GetRedirect, a member of class TXMLBroker.
	GetResponse (see page 26061)	This is GetResponse, a member of class TXMLBroker.
	GetXMLStream (see page 26062)	This is GetXMLStream, a member of class TXMLBroker.
V	Notification (see page 26064)	Forwards notification messages to all owned components.
	SendConnectEvent (see page 26067)	This is SendConnectEvent, a member of class TXMLBroker.
	SetAppServer (see page 26067)	This is SetAppServer, a member of class TXMLBroker.
	SetConnected (see page 26067)	This is SetConnected, a member of class TXMLBroker.
	SetParams (see page 26067)	This is SetParams, a member of class TXMLBroker.
	SetProviderName (see page 26068)	This is SetProviderName, a member of class TXMLBroker.
	SetReconcileProducer (see page 26068)	This is SetReconcileProducer, a member of class TXMLBroker.
	SetRemoteServer (see page 26069)	This is SetRemoteServer, a member of class TXMLBroker.
	SetWebDispatch (see page 26069)	This is SetWebDispatch, a member of class TXMLBroker.

Public Methods

	Name	Description
	AddNotify (see page 26055)	Adds an object to the Notify (see page 26073) property list.
	ApplyXMLUpdates (see page 26056)	Sends an XML delta packet of updated, inserted, and deleted records to the provider on the application server for writing to the database.
V	FetchParams (see page 26058)	Retrieves parameter values from the application server.
	GetDelta (see page 26059)	Extracts an XML delta packet from an HTTP request message.
	GetErrorCount (see page 26059)	This is GetErrorCount, a member of class TXMLBroker.
	GetErrors (see page 26060)	Returns the XML delta packet (if any) that describes the errors that occurred when applying updates.
	GetXMLRecords (see page 26062)	Retrieves an XML data packet from the application server.
	HTMLSubmitFormName (see page 26063)	Returns the name of the HTML form that submits updates to the XML broker.
	MasterRowSetVarName (see page 26063)	Returns the name of the javascript variable for the XML broker's dataset.
	RemoveNotify (see page 26064)	Removes an object to the Notify (see page 26073) property list.
V	RequestRecords (see page 26064)	Returns an XML data packet from the application server.
V	RequestUpdate (see page 26065)	Responds to an HTTP update request.
	RowSetVarName (see page 26066)	Returns the name of the javascript variable for the XML broker's dataset.
	SetProvider (see page 26068)	Associates the XML broker with a provider in the same process space.
	SubmitFormVarName (see page 26069)	Returns the name of the variable that refers to the HTML form named by HTMLSubmitFormName (see page 26063).

194.1.1.4.1 TXMLBroker.AddNotify Method

Adds an object to the Notify (see page 26073) property list.

Pascal

```
procedure AddNotify(ANotify: TObject);
```

C++

```
__fastcall AddNotify(TObject * ANotify);
```

Description

Call AddNotify to add an object to the list maintained by the Notify (see page 26073) property. Each object in this list is notified

every time a property that affects the metadata of the XML broker's data packets changes. These include changes to the RemoteServer (see page 26076), AppServer (see page 26070), and ProviderName (see page 26074) properties.

ANotify specifies the object that wants to be notified of property changes. This object must support the INotifyConnectionChange (see page 26086) interface, which the XML broker uses to notify it of changes.

See Also

NotifyCount (see page 26073)

Notify (see page 26073)

RemoveNotify (see page 26064)

RemoteServer (see page 26076)

AppServer (see page 26070)

ProviderName (see page 26074)

194.1.1.4.2 TXMLBroker.ApplyXMLUpdates Method

Sends an XML delta packet of updated, inserted, and deleted records to the provider on the application server for writing to the database.

Pascal

```
function ApplyXMLUpdates(const Delta: string; out ErrorCount: Integer): string;
```

C++

```
__fastcall AnsiString ApplyXMLUpdates(const AnsiString Delta, int ErrorCount);
```

Description

Call ApplyXMLUpdates to send the updates in the XML delta packet specified by Delta to the application server. ApplyXMLUpdates is used internally (by the RequestUpdate (see page 26065) method) to apply updates in an XML delta packet and retrieve a delta packet containing any failed updates.

Delta is the XML delta packet that contains the insertions, deletions and modifications from a Web browser. Typically, it is retrieved from an HTTP request message using the GetDelta (see page 26059) method.

ErrorCount returns the number of errors encountered on the application server before it terminated the update operation. If ErrorCount is 0, all updates were applied successfully. ErrorCount can not exceed the value of MaxErrors (see page 26071).

ApplyXMLUpdates returns an XML delta packet containing all updates that were not successfully applied. If ErrorCount is 0, this is an empty string.

See Also

RequestUpdate (see page 26065)

AS_ApplyUpdates (see page 16451)

MaxErrors (see page 26071)

GetDelta (see page 26059)

194.1.1.4.3 TXMLBroker.AS_ApplyUpdates Method

Pascal

```
function AS_ApplyUpdates(Delta: OleVariant; MaxErrors: Integer; out ErrorCount: Integer):
OleVariant;
```

C++

```
__fastcall OleVariant AS_ApplyUpdates(OleVariant Delta, int MaxErrors, int ErrorCount);
```

Description

This is AS_ApplyUpdates, a member of class TXMLBroker.

194.1.1.4.4 TXMLBroker.AS_FetchParams Method

Pascal

```
procedure AS_FetchParams;
```

C++

```
__fastcall AS_FetchParams();
```

Description

This is AS_FetchParams, a member of class TXMLBroker.

194.1.1.4.5 TXMLBroker.AS_GetRecords Method

Pascal

```
function AS_GetRecords(Count: Integer; out RecsOut: Integer; Options: Integer; const
CommandText: WideString; Params: OleVariant; var OwnerData: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant AS_GetRecords(int Count, int RecsOut, int Options, const BSTR
CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

This is AS_GetRecords, a member of class TXMLBroker.

194.1.1.4.6 TXMLBroker.CheckInactive Method

Pascal

```
procedure CheckInactive;
```

C++

```
__fastcall CheckInactive();
```

Description

This is CheckInactive, a member of class TXMLBroker.

194.1.1.4.7 TXMLBroker.DispatchEnabled Method

Pascal

```
function DispatchEnabled: Boolean;
```

C++

```
__fastcall Boolean DispatchEnabled();
```

See Also

Enabled (see page 15232)

194.1.1.4.8 TXMLBroker.DispatchMask Method

Pascal

```
function DispatchMask: TMask;
```

C++

```
__fastcall TMask DispatchMask();
```

See Also

Mask (see page 15233)

194.1.1.4.9 TXMLBroker.DispatchMethodType Method

Pascal

```
function DispatchMethodType: TMethodType;
```

C++

```
__fastcall TMethodType DispatchMethodType();
```

See Also

MethodType (see page 15233)

194.1.1.4.10 TXMLBroker.DispatchRequest Method

Pascal

```
function DispatchRequest(Sender: TObject; Request: TWebRequest; Response: TWebResponse):
Boolean;
```

C++

```
__fastcall Boolean DispatchRequest(TObject * Sender, TWebRequest Request, TWebResponse
Response);
```

194.1.1.4.11 TXMLBroker.FetchParams Method

Retrieves parameter values from the application server.

Pascal

```
procedure FetchParams; virtual;
```

C++

```
virtual __fastcall FetchParams();
```

Description

Call FetchParams to initialize the Params (see page 26073) property with the current parameter values on the application server. FetchParams does nothing unless the provider component on the application server can supply parameter values.

Warning: if the XML broker uses a stateless application server, other clients can change the values returned by FetchParams at any time.

See Also

RowRequest (see page 18472)

Params (see page 26073)

194.1.1.4.12 TXMLBroker.GetAppServer Method

Pascal

```
function GetAppServer: IAppServer;
```

C++

```
__fastcall IAppServer GetAppServer();
```

Description

This is GetAppServer, a member of class TXMLBroker.

194.1.1.4.13 TXMLBroker.GetConnected Method

Pascal

```
function GetConnected: Boolean;
```

C++

```
__fastcall Boolean GetConnected();
```

Description

This is GetConnected, a member of class TXMLBroker.

194.1.1.4.14 TXMLBroker.GetDelta Method

Extracts an XML delta packet from an HTTP request message.

Pascal

```
function GetDelta(Request: TWebRequest): string;
```

C++

```
__fastcall AnsiString GetDelta(TWebRequest Request);
```

Description

Call GetDelta to obtain the XML delta packet contained in the content of the request message specified by Request. This value can be used to apply the updates from the request message by calling ApplyXMLUpdates (see page 26056).

See Also

ApplyXMLUpdates (see page 26056)

194.1.1.4.15 TXMLBroker.GetErrorCount Method

Pascal

```
function GetErrorCount: Integer;
```

C++

```
__fastcall int GetErrorCount();
```

Description

This is GetErrorCount, a member of class TXMLBroker.

194.1.1.4.16 TXMLBroker.GetErrorResponse Method

Pascal

```
function GetErrorResponse(ErrorCount: Integer; XMLErrors: string; Request: TWebRequest;
Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean GetErrorResponse(int ErrorCount, AnsiString XMLErrors, TWebRequest Request,
TWebResponse Response);
```

Description

This is GetErrorResponse, a member of class TXMLBroker.

194.1.1.4.17 TXMLBroker.GetErrors Method

Returns the XML delta packet (if any) that describes the errors that occurred when applying updates.

Pascal

```
function GetErrors: string;
```

C++

```
__fastcall AnsiString GetErrors();
```

Description

Call GetErrors to obtain the XML delta packet returned by the application server when an update operation included some errors.

Note: See the template err.html (included in the WebMidas directory) for one possible use for the XML delta packet of errors. In err.html, the string returned by GetErrors should replace the <#XMLEMBED_VAR_ERR> tag.

See Also

OnGetErrorResponse (see page 26078)

ApplyXMLUpdates (see page 26056)

GetErrorcount (see page 26059)

194.1.1.4.18 TXMLBroker.GetHasAppServer Method

Pascal

```
function GetHasAppServer: Boolean;
```

C++

```
__fastcall Boolean GetHasAppServer();
```

Description

This is GetHasAppServer, a member of class TXMLBroker.

194.1.1.4.19 TXMLBroker.GetNotify Method

Pascal

```
function GetNotify(Index: Integer): TObject;
```

C++

```
__fastcall TObject * GetNotify(int Index);
```

Description

This is GetNotify, a member of class TXMLBroker.

194.1.1.4.20 TXMLBroker.GetNotifyCount Method

Pascal

```
function GetNotifyCount: Integer;
```

C++

```
__fastcall int GetNotifyCount();
```

Description

This is GetNotifyCount, a member of class TXMLBroker.

194.1.1.4.21 TXMLBroker.GetProducerName Method

Pascal

```
function GetProducerName(Request: TWebRequest): string;
```

C++

```
__fastcall AnsiString GetProducerName(TWebRequest Request);
```

Description

This is GetProducerName, a member of class TXMLBroker.

194.1.1.4.22 TXMLBroker.GetRedirect Method

Pascal

```
function GetRedirect(Request: TWebRequest): string;
```

C++

```
__fastcall AnsiString GetRedirect(TWebRequest Request);
```

Description

This is GetRedirect, a member of class TXMLBroker.

194.1.1.4.23 TXMLBroker.GetResponse Method

Pascal

```
function GetResponse(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean GetResponse(TWebRequest Request, TWebResponse Response);
```

Description

This is GetResponse, a member of class TXMLBroker.

194.1.1.4.24 TXMLBroker.GetXMLRecords Method

Retrieves an XML data packet from the application server.

Pascal

```
function GetXMLRecords(var RecsOut: Integer; var OwnerData: OleVariant; XMLOptions:
TXMLOptions): string;
```

C++

```
__fastcall AnsiString GetXMLRecords(int RecsOut, OleVariant OwnerData, TXMLOptions XMLOptions);
```

Description

Call GetXMLRecords to obtain the XML data packet from the application server. GetXMLRecords calls the interface specified by AppServer (see page 26070), requesting an XML data packet with at most MaxRecords (see page 26072) records. It returns the resulting XML data packet in a form that is ready for insertion into an HTML document.

RecsOut returns the number of records in the XML data packet.

On entry, OwnerData contains an application-defined value that is passed to the provider's BeforeGetRecords event handler. OwnerData returns an application-defined value that is set by the provider's AfterGetRecords event handler. Typically, this parameter is used to transmit persistent state information to and from a stateless application server.

XMLOptions is a set of flags that indicate how the broker should format the XML data packet after it is retrieved from the application server.

GetXMLRecords returns the formatted XML data packet.

Note: Unlike the RequestRecords (see page 26064) method, GetXMLRecords bypasses the OnRequestRecords (see page 26080) event.

See Also

RequestRecords (see page 26064)

AS_GetRecords (see page 16455)

MaxRecords (see page 26072)

AppServer (see page 26070)

BeforeGetRecords (see page 18480)

AfterGetRecords (see page 18477)

Using an XML Broker

194.1.1.4.25 TXMLBroker.GetXMLStream Method

Pascal

```
function GetXMLStream(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXMLStream(Boolean Owned);
```

Description

This is GetXMLStream, a member of class TXMLBroker.

194.1.1.4.26 TXMLBroker.HTMLSubmitFormName Method

Returns the name of the HTML form that submits updates to the XML broker.

Pascal

```
function HTMLSubmitFormName: string;
```

C++

```
__fastcall AnsiString HTMLSubmitFormName();
```

Description

When the InternetExpress page producer assembles an HTML document that uses the XML data packets managed by the XML broker, it declares an HTML form that submits the updates to the XML broker. HTMLSubmitFormName is the name of that HTML form.

The HTML form named by this return value sends the HTTP request messages containing an XML delta packet of updates to be applied.

See Also

SubmitFormVarName (see page 26069)

Content (see page 17138)

194.1.1.4.27 TXMLBroker.MasterRowSetVarName Method

Returns the name of the javascript variable for the XML broker's dataset.

Pascal

```
function MasterRowSetVarName(Path: TStrings): string;
```

C++

```
__fastcall AnsiString MasterRowSetVarName(TStrings Path);
```

Description

When the InternetExpress page producer assembles an HTML document that uses the XML data packets managed by the XML broker, it declares a javascript variable of type RowSet for the javascript object that represents the XML broker's dataset. When the variable is first declared, the InternetExpress page producer calls MasterRowSetVarName.

Path is a set of variable names for all the master datasets for which the XML broker's dataset is a detail set. These names come from previous calls to the MasterRowSetVarName method of their XML brokers. The last string in Path refers to the XML broker's dataset itself.

MasterRowSetVarName returns the name of the XML broker, qualified by all strings in Path except for the last one, which is assumed to refer to the XML broker's dataset. Thus, to change the name of the generated variable for an XML broker's dataset, change the Name property of the XML broker.

Note: To retrieve the name of the XML broker preceded by all the datasets in Path, use the RowSetVarName (see page 26066) method instead.

See Also

Name (see page 2805)

Content (see page 17138)

RowSetVarName (see page 26066)

194.1.1.4.28 TXMLBroker.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

194.1.1.4.29 TXMLBroker.RemoveNotify Method

Removes an object to the Notify (see page 26073) property list.

Pascal

```
procedure RemoveNotify(ANotify: TObject);
```

C++

```
__fastcall RemoveNotify(TObject * ANotify);
```

Description

Call RemoveNotify to remove an object from the list maintained by the Notify (see page 26073) property. Each object in this list is notified every time a property that affects the metadata of the XML broker's data packets changes.

See Also

NotifyCount (see page 26073)

Notify (see page 26073)

AddNotify (see page 26055)

194.1.1.4.30 TXMLBroker.RequestRecords Method

Returns an XML data packet from the application server.

Pascal

```
function RequestRecords(Sender: TObject; Request: TWebRequest; out RecCount: Integer; var
OwnerData: OleVariant; XMLOptions: TXMLOptions): string; virtual;
```

C++

```
virtual __fastcall AnsiString RequestRecords(TObject * Sender, TWebRequest Request, int
RecCount, OleVariant OwnerData, TXMLOptions XMLOptions);
```

Description

Call RequestRecords to obtain a current XML data packet, while first allowing the OnRequestRecords (see page 26080) event handler to customize it. After generating the OnRequestRecords (see page 26080) event, RequestRecords fetches the XML data packet by calling GetXMLRecords (see page 26062), unless the event handler supplies the XML data packet first.

Sender is the object that is requesting the XML data packet (usually an InternetExpress page producer).

Request describes an HTTP request message that originated the need for an XML data packet. Typically, this is a request message for which the InternetExpress page producer is creating a response.

RecCount returns the number of records in the XML data packet.

On entry, OwnerData contains an application-defined value that is passed to the OnRequestRecords (see page 26080) event handler, and then forwarded (with any changes provided by the event handler) to the provider's BeforeGetRecords event handler. OwnerData returns an application-defined value that is set by the provider's AfterGetRecords event handler. Typically, this parameter is used to transmit persistent state information to and from a stateless application server.

XMLOptions is a set of flags indicating how the XML broker should format the XML data packet that the application server or OnRequestRecords (see page 26080) event handler supplies.

RequestRecords returns the XML data packet.

See Also

GetXMLRecords (see page 26062)

AS_GetRecords (see page 16455)

OnRequestRecords (see page 26080)

AppServer (see page 26070)

BeforeGetRecords (see page 18480)

AfterGetRecords (see page 18477)

Using an XML Broker

194.1.1.4.31 TXMLBroker.RequestUpdate Method

Responds to an HTTP update request.

Pascal

```
function RequestUpdate(Sender: TObject; Request: TWebRequest; Response: TWebResponse):
Boolean; virtual;
```

C++

```
virtual __fastcall Boolean RequestUpdate(TObject * Sender, TWebRequest Request, TWebResponse
Response);
```

Description

Call RequestUpdate when using a Web action item to respond to update requests from a Web browser. When using a Web

action item to respond to update requests, instead of letting the XML broker receive the update requests directly, you should set the Enabled property of the WebDispatch (see page 26076) to false.

RequestUpdate generates an OnRequestUpdate (see page 26080) event, to allow the event handler to apply the updates in the HTTP request message and generate a response. If the OnRequestUpdate (see page 26080) event handler does not handle the message, RequestUpdate applies the updates using the ApplyXMLUpdates (see page 26056) method. Finally, if there are no update errors, RequestUpdate generates an OnGetResponse (see page 26079) event. If there were update errors, RequestResponse generates an OnGetErrorResponse (see page 26078) event and (if the event handler does not generate a response) assigns the content of ReconcileProducer (see page 26075) to the response message. If there are update errors and the response is not completed by an event handler or the ReconcileProducer (see page 26075), the XML broker raises an exception.

Sender is the object that received the request message.

Request describes an HTTP request message that contains the updates that need to be applied.

Response is the emerging response object that should be completed by RequestUpdate so that the XML broker can send it back to the Web browser.

RequestUpdate returns true if the response message is ready to send back to the Web browser, false if it still needs more processing.

See Also

ApplyXMLUpdates (see page 26056)

AS_ApplyUpdates (see page 16451)

OnRequestUpdate (see page 26080)

AppServer (see page 26070)

BeforeApplyUpdates (see page 18479)

AfterApplyUpdates (see page 18475)

OnGetResponse (see page 26079)

OnGetErrorResponse (see page 26078)

ReconcileProducer (see page 26075)

WebDispatch (see page 26076)

Enabled (see page 1482)

194.1.1.4.32 TXMLBroker.RowSetVarName Method

Returns the name of the javascript variable for the XML broker's dataset.

Pascal

```
function RowSetVarName(Path: TStrings): string;
```

C++

```
__fastcall AnsiString RowSetVarName(TStrings Path);
```

Description

When the InternetExpress page producer assembles an HTML document that uses the XML data packets managed by the XML broker, it declares a javascript variable of type RowSet for the javascript object that represents the XML broker's dataset. When the InternetExpress page producer needs to determine whether a RowSet variable was already declared, it looks up the variable name by calling RowSetVarName.

Path is a set of variable names for all the master datasets for which the XML broker's dataset is a detail set.

RowSetVarName returns the name of the XML broker, qualified by all strings in Path.

Note: Variable names are assigned for an XML broker's dataset using the MasterRowSetVarName (see page 26063) method.

See Also

Name (see page 2805)

Content (see page 17138)

MasterRowSetVarName (see page 26063)

194.1.1.4.33 TXMLBroker.SendConnectEvent Method

Pascal

```
procedure SendConnectEvent(Connecting: Boolean);
```

C++

```
__fastcall SendConnectEvent(Boolean Connecting);
```

Description

This is SendConnectEvent, a member of class TXMLBroker.

194.1.1.4.34 TXMLBroker.SetAppServer Method

Pascal

```
procedure SetAppServer(Value: IAppServer);
```

C++

```
__fastcall SetAppServer(IAppServer Value);
```

Description

This is SetAppServer, a member of class TXMLBroker.

194.1.1.4.35 TXMLBroker.SetConnected Method

Pascal

```
procedure SetConnected(Value: Boolean);
```

C++

```
__fastcall SetConnected(Boolean Value);
```

Description

This is SetConnected, a member of class TXMLBroker.

194.1.1.4.36 TXMLBroker.SetParams Method

Pascal

```
procedure SetParams(const Value: TXMLParams);
```

C++

```
__fastcall SetParams(const TXMLParams Value);
```

Description

This is SetParams, a member of class TXMLBroker.

194.1.1.4.37 TXMLBroker.SetProvider Method

Associates the XML broker with a provider in the same process space.

Pascal

```
procedure SetProvider(Provider: TComponent);
```

C++

```
__fastcall SetProvider(TComponent * Provider);
```

Description

Use SetProvider to specify a local provider component that provides data to and applies updates from the XML broker. The XML broker communicates with this provider using the IAppServer interface specified by the AppServer (see page 26070) property.

Do not use SetProvider for remote providers on an application server. If the provider is not in the same process space as the XML broker, use the ProviderName (see page 26074) property.

The Provider (see page 18419) parameter must be a descendant of TCustomProvider. Otherwise, SetProvider raises an exception.

See Also

AppServer (see page 26070)

ProviderName (see page 26074)

TDataSetProvider (see page 18508)

TCustomProvider (see page 18449)

194.1.1.4.38 TXMLBroker.SetProviderName Method

Pascal

```
procedure SetProviderName(const Value: string);
```

C++

```
__fastcall SetProviderName(const AnsiString Value);
```

Description

This is SetProviderName, a member of class TXMLBroker.

194.1.1.4.39 TXMLBroker.SetReconcileProducer Method

Pascal

```
procedure SetReconcileProducer(const Value: TCustomContentProducer);
```

C++

```
__fastcall SetReconcileProducer(const TCustomContentProducer Value);
```

Description

This is SetReconcileProducer, a member of class TXMLBroker.

194.1.1.4.40 TXMLBroker.SetRemoteServer Method

Pascal

```
procedure SetRemoteServer(Value: TCustomRemoteServer);
```

C++

```
__fastcall SetRemoteServer(TCustomRemoteServer Value);
```

Description

This is SetRemoteServer, a member of class TXMLBroker.

194.1.1.4.41 TXMLBroker.SetWebDispatch Method

Pascal

```
procedure SetWebDispatch(const Value: TWebDispatch);
```

C++

```
__fastcall SetWebDispatch(const TWebDispatch Value);
```

Description

This is SetWebDispatch, a member of class TXMLBroker.

194.1.1.4.42 TXMLBroker.SubmitFormVarName Method

Returns the name of the variable that refers to the HTML form named by HTMLSubmitFormName (see page 26063).

Pascal

```
function SubmitFormVarName: string;
```

C++

```
__fastcall AnsiString SubmitFormVarName();
```

Description

When the InternetExpress page producer assembles an HTML document that uses the XML data packets managed by the XML broker, it declares an HTML form that submits the updates to the XML broker. In addition, it declares a variable that refers to that form. SubmitFormVarName is the name of that variable.

See Also

HTMLSubmitFormName (see page 26063)

Content (see page 17138)

194.1.1.5 TXMLBroker Properties

The properties of the TXMLBroker class are listed here.

Public Properties

	Name	Description
	AppServer (see page 26070)	Provides access to the interface the XML broker uses to communicate with the application server.
R	HasAppServer (see page 26071)	Indicates whether the XML broker has an IAppServer interface for communicating with its provider.

R	Notify (see page 26073)	Lists the objects that are notified when any properties affecting the XML broker's connection to its provider change.
R	NotifyCount (see page 26073)	Indicates the number of objects in the Notify (see page 26073) property list.

Published Properties

	Name	Description
	Connected (see page 26070)	Determines whether the XML broker has an active connection to the provider that supplies its data and applies its updates.
	MaxErrors (see page 26071)	Specifies the maximum number of update errors allowed before aborting an update operations.
	MaxRecords (see page 26072)	Specifies the maximum number of records that can appear in XML data packets.
	Params (see page 26073)	Contains parameter values that are sent to or retrieved from the application server.
	ProviderName (see page 26074)	Specifies the name of a registered provider from the application server.
	ReconcileProducer (see page 26075)	Specifies the producer that generates response message content when an update generates errors.
	RemoteServer (see page 26076)	Specifies the connection component that this XML broker uses to connect to an application server.
	WebDispatch (see page 26076)	Lets the Web dispatcher identify what HTTP messages should be forwarded to the XML broker.

194.1.1.5.1 TXMLBroker.AppServer Property

Provides access to the interface the XML broker uses to communicate with the application server.

Pascal

```
property AppServer: IAppServer;
```

C++

```
__property IAppServer AppServer;
```

Description

Use AppServer to communicate with the providers that supply data to and resolve updates for the XML broker. If the providers are on a remote application server, AppServer has the same value as the AppServer property of the connection component. If the XML broker uses a provider in the same process space, AppServer is the interface to an internally created object that dispatches IAppServer calls to the local provider.

Applications don't need to work directly with the IAppServer interface. The properties and methods of the XML broker are automatically translated into IAppServer calls as needed.

See Also

HasAppServer (see page 26071)

AppServer (see page 8532)

SetProvider (see page 26068)

194.1.1.5.2 TXMLBroker.Connected Property

Determines whether the XML broker has an active connection to the provider that supplies its data and applies its updates.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Use Connected to determine whether the XML broker can supply XML data packets and apply updates. If the XML broker uses a provider on an application server, setting Connected to true sets the Connected property of the connection component to true. If the XML broker uses a local provider component, setting Connected to true causes the XML broker to generate an internal object that communicates with the provider using the IAppServer interface. Thus, the value of Connected indicates whether an IAppServer interface is available for communicating with a provider, and so the value of Connected is the same as the value of HasAppServer (see page 26071).

You can't set Connected to true unless the XML broker has a provider to which it can connect. Trying to set Connected to true before the ProviderName (see page 26074) property is set or a local provider is assigned using the SetProvider (see page 26068) method causes the XML broker to raise an exception.

Note: In the Web Page editor, default fields or parameter values may not appear if the associated XML broker is not connected.

See Also

HasAppServer (see page 26071)

AppServer (see page 26070)

Connected (see page 6868)

TDataSetProvider (see page 18508)

ProviderName (see page 26074)

SetProvider (see page 26068)

194.1.1.5.3 TXMLBroker.HasAppServer Property

Indicates whether the XML broker has an IAppServer interface for communicating with its provider.

Pascal

```
property HasAppServer: Boolean;
```

C++

```
__property Boolean HasAppServer;
```

Description

Read HasAppServer to determine whether the XML broker can get its data from an IAppServer interface. When HasAppServer is true, the XML broker can interact with its provider through the IAppServer interface.

Use HasAppServer to check whether the XML broker is connected rather than reading the AppServer (see page 26070) property. Reading AppServer (see page 26070) causes the XML broker to obtain an IAppServer interface if it does not already have one.

See Also

IAppServer (see page 16450)

AppServer (see page 26070)

Connected (see page 26070)

194.1.1.5.4 TXMLBroker.MaxErrors Property

Specifies the maximum number of update errors allowed before aborting an update operations.

Pascal

```
property MaxErrors: Integer;
```

C++

```
__property int MaxErrors;
```

Description

Set MaxErrors to indicate the maximum number of errors that the associated provider should allow before prematurely stopping an update operation. Set MaxErrors to –1 to indicate that there is no limit to the number of errors.

See Also

AS_ApplyUpdates (see page 16451)

ReconcileProducer (see page 26075)

ApplyXMLUpdates (see page 26056)

194.1.1.5.5 TXMLBroker.MaxRecords Property

Specifies the maximum number of records that can appear in XML data packets.

Pascal

```
property MaxRecords: Integer;
```

C++

```
__property int MaxRecords;
```

Description

Set MaxRecords to limit the size of XML data packets retrieved from the associated provider.

When MaxRecords is –1, XML data packets contain all the records in the provider's dataset.

When MaxRecords is 0, XML data packets contain only metadata.

When MaxRecords is greater than 0, XML data packets contain up to MaxRecords records, depending on the number of records in the provider's dataset.

Tip: Because an InternetExpress application sends the entire data packet to client applications, it is important to limit the size of data packets. Otherwise, the time to download web pages containing data packets can be prohibitively long.

Note: When limiting the number of records that appear in data packets, you may need to fetch several data packets before all data is returned. Because InternetExpress applications are always stateless, you will need to supply information about what record should start the next data packet as the OwnerData parameter of GetXMLRecords (see page 26062) or RequestRecords (see page 26064). You can use a query form to obtain information about the last record seen by a particular Web client.

See Also

AS_GetRecords (see page 16455)

GetXMLRecords (see page 26062)

RequestRecords (see page 26064)

TQueryForm (see page 16745)

Using an XML Broker

194.1.1.5.6 TXMLBroker.Notify Property

Lists the objects that are notified when any properties affecting the XML broker's connection to its provider change.

Pascal

```
property Notify [Index: Integer]: TObject;
```

C++

```
__property TObject * Notify[int Index];
```

Description

Read Notify to access the objects that respond when the XML broker's connection to its provider changes. These objects are sent a notification when the ProviderName (see page 26074), AppServer (see page 26070), or RemoteServer (see page 26076) property changes.

Most applications do not need to use the Notify property. It allows the Web page editor to remain synchronized with property changes made at design time.

See Also

NotifyCount (see page 26073)

AddNotify (see page 26055)

RemoveNotify (see page 26064)

RemoteServer (see page 26076)

AppServer (see page 26070)

ProviderName (see page 26074)

194.1.1.5.7 TXMLBroker.NotifyCount Property

Indicates the number of objects in the Notify (see page 26073) property list.

Pascal

```
property NotifyCount: Integer;
```

C++

```
__property int NotifyCount;
```

Description

NotifyCount specifies the number of objects listed in the Notify (see page 26073) property.

See Also

Notify (see page 26073)

AddNotify (see page 26055)

RemoveNotify (see page 26064)

194.1.1.5.8 TXMLBroker.Params Property

Contains parameter values that are sent to or retrieved from the application server.

Pascal

```
property Params: TXMLParams;
```

C++

```
__property TXMLParams Params;
```

Description

Use Params to specify parameter values that should be passed to a source dataset on the application server. These parameter values are used in one of two ways:

If the associated provider represents a query or stored procedure, they supply parameter values that the application server uses when executing the query or stored procedure.

If the associated provider represents a table, they limit the records that are sent in data packets so that they only include records where each field with the same name as a parameter has the same value as the parameter.

After the application specifies parameters (at design time or runtime), they are sent to the application server automatically when the XML broker fetches data.

To send parameter values to a query or stored procedure, add a parameter object (TParam) to Params for every parameter on the query or stored procedure for which you want to supply a value. Assign each parameter object the same Name, DataType, and ParamType as the corresponding parameter on the query or stored procedure. These values are automatically encoded and sent to the application server when the XML broker fetches data.

To send parameter values to a table, add a parameter object for each field in the table that is used to limit the values sent in data packets. Assign each parameter object the same Name and DataType as the corresponding field component on the table. These parameter values are automatically sent when the client dataset fetches data, and the records returned in data packets from the application server only contain records that match the assigned value on the corresponding fields.

Tip: You can assign parameter values from the content fields supplied to a request message by TSubmitQueryButton using the AssignStrings method of Params.

Note: To initialize Params so that it reflects the parameters of a query or stored procedure on an application server, call the FetchParams (see page 26058) method.

See Also

Params (see page 10087)

Params (see page 10165)

FetchParams (see page 26058)

TSubmitQueryButton (see page 16833)

AssignStrings (see page 26084)

Using an XML Broker

194.1.1.5.9 TXMLBroker.ProviderName Property

Specifies the name of a registered provider from the application server.

Pascal

```
property ProviderName: string;
```

C++

```
__property AnsiString ProviderName;
```

Description

Use ProviderName to specify the provider with which to communicate on the server (or in the same Web module). This property is passed as a parameter to IAppServer interface calls when the XML broker fetches data or applies updates. If an application server contains multiple provider objects, an XML broker can set ProviderName to any of them once the connection component establishes a connection to the application server.

ProviderName can be set from a drop-down in the Object Inspector once the RemoteServer (see page 26076) property is set.

Note: ProviderName can also be used to link the XML broker to a provider in the same process space, as long as the provider component has the same Owner as the XML broker. If the local provider component has a different Owner, use the SetProvider (see page 26068) method instead.

See Also

RemoteServer (see page 26076)

TDataSetProvider (see page 18508)

IAppServer (see page 16450)

SetProvider (see page 26068)

Using an XML Broker

194.1.1.5.10 TXMLBroker.ReconcileProducer Property

Specifies the producer that generates response message content when an update generates errors.

Pascal

```
property ReconcileProducer: TCustomContentProducer;
```

C++

```
__property TCustomContentProducer ReconcileProducer;
```

Description

Use ReconcileProducer to specify the component that generates the content of response messages to update requests when the update operation generated errors.

When the XML broker applies updates, the application server returns a delta packet containing all records that could not be updated because of errors. The XML broker can attempt to respond to those errors or generate its own response message using the OnGetErrorResponse (see page 26078) event. However, unless the OnGetErrorResponse (see page 26078) event handler generates an HTML document to be sent in the response message and indicates that the update is completely handled, the XML broker must still generate a response to the update operation. To do so, it calls the Content method of the ReconcileProducer to generate the content of the response message.

Tip: The Reconcile producer will probably need information about the update errors that the application server returned. You can provide it with this information either by initializing the Reconcile producer in an OnGetErrorResponse (see page 26078) event handler or by having the Reconcile producer call the GetErrors (see page 26060) method.

Note: If there is no ReconcileProducer assigned and an update generates errors, the XML broker raises an exception unless the OnGetErrorResponse (see page 26078) event handler generates a response message.

Tip: The InetXCustom demo contains a TReconcilePageProducer component that you can use as the value of the ReconcileProducer property or as the basis of your own reconcile producer.

See Also

MaxErrors (see page 26071)

OnGetErrorResponse (see page 26078)

ApplyXMLUpdates (see page 26056)

GetErrors (see page 26060)

Using an XML Broker

194.1.1.5.11 TXMLBroker.RemoteServer Property

Specifies the connection component that this XML broker uses to connect to an application server.

Pascal

```
property RemoteServer: TCustomRemoteServer;
```

C++

```
__property TCustomRemoteServer RemoteServer;
```

Description

Use RemoteServer to specify the connection component that establishes a connection to the remote application server. The connection component obtains an IAppServer interface for data exchange and maintains the interface of the remote data module on the application server.

If a Web application uses more than connection component, the application can change which connection an XML broker uses by changing the value for this property.

See Also

ProviderName (see page 26074)

TSocketConnection (see page 18928)

TDCOMConnection (see page 16008)

TWebConnection (see page 18968)

AppServer (see page 8532)

IAppServer (see page 16450)

Using an XML Broker

194.1.1.5.12 TXMLBroker.WebDispatch Property

Lets the Web dispatcher identify what HTTP messages should be forwarded to the XML broker.

Pascal

```
property WebDispatch: TWebDispatch;
```

C++

```
__property TWebDispatch WebDispatch;
```

Description

Use WebDispatch to specify the HTTP messages to which the XML broker responds. Typically, these originate from a button added to HTML forms by TApplyUpdatesButton.

The TWebDispatch object that implements WebDispatch includes properties that let you specify the path portion of URLs to which the XML broker responds, as well as the types of messages it accepts (typically post messages). You can also use this property to disable the XML broker so that the Web dispatcher does not send it any HTTP messages. When the XML broker is disabled in this way, it can still provide XML data packets to an InternetExpress page producer, but any update messages must be handled by a Web action item.

See Also

TWebModule (see page 15154)

TWebDispatcher (see page 15147)

TApplyUpdatesButton (see page 16480)

Using an XML Broker

194.1.1.6 TXMLBroker Events

The events of the TXMLBroker class are listed here.

Published Events

	Name	Description
	AfterDispatch (see page 26077)	Occurs after the XML broker generates the content of a response message.
	BeforeDispatch (see page 26078)	Occurs when the XML broker receives an HTTP request message.
	OnGetErrorResponse (see page 26078)	Occurs when there are errors applying updates to the application server.
	OnGetResponse (see page 26079)	Occurs when there are no errors applying updates to the application server.
	OnRequestRecords (see page 26080)	Occurs when the XML broker needs to fetch a data packet from its associated provider.
	OnRequestUpdate (see page 26080)	Occurs immediately before the XML broker applies updates to the application server.

194.1.1.6.1 TXMLBroker.AfterDispatch Event

Occurs after the XML broker generates the content of a response message.

Pascal

```
property AfterDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent AfterDispatch;
```

Description

Write an AfterDispatch event handler to perform any final actions before sending a response back to the Web browser that sent the XML broker an update message. Usually by the time AfterDispatch occurs, the XML broker has already applied the updates and generated the response message (using the ReconcileProducer (see page 26075) or one of the events that occurs when the XML broker receives an update request).

See Also

WebDispatch (see page 26076)

BeforeDispatch (see page 26078)

OnRequestUpdate (see page 26080)

OnGetResponse (see page 26079)

OnGetErrorResponse (see page 26078)

ReconcileProducer (see page 26075)

Using an XML Broker

194.1.1.6.2 TXMLBroker.BeforeDispatch Event

Occurs when the XML broker receives an HTTP request message.

Pascal

```
property BeforeDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent BeforeDispatch;
```

Description

Write a BeforeDispatch event handler to initialize the response to an update request or to handle any messages sent to the XML broker that are not update requests. If the event handler completely handles the request, filling out the response object so that it is ready to send, it should set the Handled parameter to true, thereby preventing the XML broker from applying the updates.

Unless the BeforeDispatch event handler sets the Handled parameter to true, the XML broker receives an OnRequestUpdate (see page 26080) event after the BeforeDispatch event handler exits. After the OnRequestUpdate (see page 26080) event, the XML broker applies the updates to the application server and generates the contents of the response based on whether there were any update errors. Finally, the XML broker receives an AfterDispatch (see page 26077) event where it can perform any final changes to the response message before it is sent back to the Web browser.

If the BeforeDispatch event handler sets the Handled parameter to true, the XML broker receives an AfterDispatch (see page 26077) event, but does not attempt to apply updates from the request message or receive any of the events that accompany that process.

See Also

WebDispatch (see page 26076)

AfterDispatch (see page 26077)

OnRequestUpdate (see page 26080)

Using an XML Broker

194.1.1.6.3 TXMLBroker.OnGetErrorResponse Event

Occurs when there are errors applying updates to the application server.

Pascal

```
property OnGetErrorResponse: TGetErrorResponseEvent;
```

C++

```
__property TGetErrorResponseEvent OnGetErrorResponse;
```

Description

Write an OnGetErrorResponse event handler to generate the response message after an update request encountered errors or to initialize a ReconcileProducer (see page 26075) before it generates the response.

The Sender parameter is the XML broker that applied updates to the application server and received back a delta packet of update errors.

ErrorCount is the number of update errors found by the application server before it halted the update operation.

XMLErrors is a string of XML returned by the application server. It is a delta packet containing all the updates that could not be applied.

Request is the original request message that contains the delta packet of updates from the Web browser.

Response is the emerging response message that will be sent to the Web browser in response to the update request. Fill in the properties of this object to generate a response (such as an error message or a reconcile form) that indicates the errors in XMLErrors.

Handled indicates whether the event handler completed filling out Response. Set Handled to true if the event handler completes the response for the Web browser. Leave Handled as false if ReconcileProducer (see page 26075) should generate the content of Response.

Note: If the OnGetErrorResponse event handler does not set Handled to true and there is no ReconcileProducer (see page 26075), the XML broker raises an exception.

See Also

MaxErrors (see page 26071)

ReconcileProducer (see page 26075)

OnGetResponse (see page 26079)

ApplyXMLUpdates (see page 26056)

Using an XML Broker

194.1.1.6.4 TXMLBroker.OnGetResponse Event

Occurs when there are no errors applying updates to the application server.

Pascal

```
property OnGetResponse: TGetResponseEvent;
```

C++

```
__property TGetResponseEvent OnGetResponse;
```

Description

Write an OnGetResponse event handler to edit the response message after an update request is successfully applied.

The Sender parameter is the XML broker that applied updates to the application server with no errors.

Request is the original request message that contains the delta packet of updates from the Web browser.

Response is the emerging response message that will be sent to the Web browser in response to the update request. Fill in the properties of this object to generate a response (such as refreshing the data).

Handled indicates whether the event handler completed filling out Response. Set Handled to true if the event handler completes the response for the Web browser. Leave Handled as false if the XML broker should send a redirect response that sends the browser back to the page that generated the update request.

See Also

OnGetErrorResponse (see page 26078)

ApplyXMLUpdates (see page 26056)

Using an XML Broker

194.1.1.6.5 TXMLBroker.OnRequestRecords Event

Occurs when the XML broker needs to fetch a data packet from its associated provider.

Pascal

```
property OnRequestRecords: TRequestRecordsEvent;
```

C++

```
__property TRequestRecordsEvent OnRequestRecords;
```

Description

Write an OnRequestRecords event handler to provide an alternate mechanism for providing XML data packets or to perform other actions whenever the XML broker needs to fetch an XML data packet. For example, the OnRequestRecords event handler might call GetXMLRecords (see page 26062) to fetch an XML data packet from the application server and then edit its contents.

OnRequestRecords occurs when the InternetExpress page producer requests an XML data packet (usually in response to a request message) or when the application calls the XML broker's RequestRecords (see page 26064) method.

The Sender parameter is the object that originates the request (usually an InternetExpress page producer).

Request is the request message that prompted the request for an XML data packet. This value may be nil (Delphi) or NULL (C++) if the request did not arise from a page producer or if the page producer is not processing a request message.

OwnerData contains an application-defined variant that is sent to the provider's BeforeGetRecords event handler. The InternetExpress page producer that is requesting records or the caller of RequestRecords (see page 26064) may initialize this value. OwnerData can be used to pass persistent state information to a stateless application server (for example, the first record that should appear in the data packet when MaxRecords (see page 26072) is greater than 0).

Records optionally returns an XML data packet as a string. If Records is an empty string on exit from the event handler, the XML broker automatically fetches the XML data packet from its associated provider. Otherwise, the XML broker uses the value of Records as the new XML data packet.

See Also

RequestRecords (see page 26064)

GetXMLRecords (see page 26062)

GetXMLData (see page 17124)

BeforeGetRecords (see page 18480)

MaxRecords (see page 26072)

Using an XML Broker

194.1.1.6.6 TXMLBroker.OnRequestUpdate Event

Occurs immediately before the XML broker applies updates to the application server.

Pascal

```
property OnRequestUpdate: TRequestUpdateEvent;
```

C++

```
__property TRequestUpdateEvent OnRequestUpdate;
```

Description

Write an OnRequestUpdate event handler to provide an alternate mechanism for applying updates to the application server. For example, the OnRequestUpdate event handler might call GetDelta (see page 26059) to extract the delta packet from the request message and then preprocess the XML string before applying the updates with a call to ApplyXMLUpdates (see page 26056).

OnRequestUpdate occurs immediately after the BeforeDispatch (see page 26078) event or when the application calls the XML broker's RequestUpdate (see page 26065) method.

The Sender parameter is the object that received the update request (the XML broker or a Web action item).

Request is the request message that contains the updates to be applied. The XML delta packet can be extracted from this request object using the GetDelta (see page 26059) method.

Response is the emerging response message that will be sent back to the Web browser that generated the update request. The event handler can set properties on this response object to partially or completely specify the message will be returned to the Web browser.

Handled indicates whether the event handler applied the updates. If Handled is true on exit from the event handler, the XML broker does not apply the updates, does not receive an OnGetResponse (see page 26079) or OnGetErrorResponse (see page 26078) event, and does not call on the ReconcileProducer (see page 26075) to generate a response message. (It does still receive an AfterDispatch (see page 26077) event). If Handled is false on exit from the event handler, the XML broker proceeds to apply the updates contained in the request message and receives all the associated events.

Note: If the event handler applies the updates, it must also generate the response message.

See Also

BeforeDispatch (see page 26078)

ApplyXMLUpdates (see page 26056)

RequestUpdate (see page 26065)

GetDelta (see page 26059)

OnGetResponse (see page 26079)

OnGetErrorResponse (see page 26078)

ReconcileProducer (see page 26075)

AfterDispatch (see page 26077)

Using an XML Broker

194.1.2 TXMLParams Class

TXMLParams manages a list of parameters for an XML broker.

Class Hierarchy

Pascal

```
TXMLParams = class(TParams);
```

C++

```
class TXMLParams : public TParams;
```

File

XMLBrokr

Description

XML brokers use TXMLParams to manage a list of parameters that it passes to a provider on the application server. These parameters supply values to a query or stored procedure on the application server, or limit the records provided from a table on the application server.

Use the properties and methods of TXMLParams to:

Access a specific parameter.

Add or delete parameters from the list.

Get or set the values of individual parameters.

Assign parameter values from a list such as the one supplied in a request message from a Web browser.

Compare parameters.

Iterate through all parameters.

Copy a set of parameters to another parameter list.

See Also

TParam (see page 7386)

TXMLBroker (see page 26049)

194.1.2.1 TXMLParams Members

The following tables list the members exposed by TXMLParams.

Protected Methods

	Name	Description
V	Added (see page 2736)	Responds when items are added to the collection.
V	AssignTo (see page 26085)	Copies the properties of an object to a destination object.
	Changed (see page 2740)	Responds when the collection or any of its items changes.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 7444)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	Deleting (see page 2741)	Responds when items are deleted from the collection.
	GetAttr (see page 2743)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr (see page 2745) method.
	GetAttrCount (see page 2744)	Returns the number of custom attributes associated with items in the collection.
	GetDataSet (see page 7445)	This is GetDataSet, a member of class TParams.
	GetItem (see page 2744)	Returns a specified item in the collection.
	GetItemAttr (see page 2745)	Returns the value of a custom attribute assigned to one of the collection's items.
	GetOwner (see page 2932)	Returns the owner of an object.
V	GetOwner (see page 7445)	Returns the owner of an object.
V	Notify (see page 2746)	Responds when items are added to or removed from the collection.
	SetItem (see page 2747)	Copies the properties of another item to a specified item in the collection.
V	SetItemName (see page 2747)	Initializes the name of a newly inserted collection item.

V	Update (see page 2747)	Updates the collection to reflect changes to its items.
V	Update (see page 7448)	Updates the collection to reflect changes to its items.

Protected Properties

	Name	Description
R	NextID (see page 2752)	Specifies a unique ID that can be assigned to the next added collection item.
	PropName (see page 2752)	Specifies the name of the property that the collection implements.
R	UpdateCount (see page 2753)	Counts the number of times BeginUpdate (see page 2739) was called without a corresponding call to EndUpdate (see page 2742).

Public Methods

	Name	Description
	Add (see page 2736)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	AddParam (see page 7438)	Inserts a new field parameter into the Items (see page 7449) list.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2736)	Copies the contents of another collection to the object where the method is executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
	AssignStrings (see page 26084)	Assigns values to the parameters from a list of name/value pairs.
	AssignValues (see page 7439)	Assigns new values to the parameters of the Items (see page 7449) property.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
V	BeginUpdate (see page 2739)	Suspends screen repainting.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 2741)	Deletes all items from the collection.
	Create (see page 2734)	Creates and initializes a collection.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7437)	This is the overview for the Create constructor overload.
	CreateParam (see page 7441)	Creates a new field parameter object and inserts it into the Items (see page 7449) list.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 2741)	Deletes a single item from the collection.
V	Destroy (see page 2735)	Destroys the collection and each item in it.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	EndUpdate (see page 2742)	Re-enables screen repainting.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindItemID (see page 2743)	Returns the item with the specified ID.
	FindParam (see page 7445)	Locates the parameter with a given name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetEnumerator (see page 2744)	Returns a TCollection (see page 2731) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2745)	Returns a string used by the Object Inspector.

	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParamList (see page 7446)	Fills a TList object with the TParam (see page 7386) objects specified by name.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	Insert (see page 2746)	Creates a new TCollectionItem (see page 2755) instance and adds it to the Items (see page 2751) array.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEqual (see page 7446)	Indicates whether two TParams (see page 7434) objects have identical Items (see page 7449).
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Owner (see page 2746)	Returns the Owner of the collection.
	ParamByName (see page 7447)	Locates the parameter with a given name.
	ParseSQL (see page 7447)	Locates the parameters in an SQL statement and replaces them with question marks.
	RemoveParam (see page 7448)	Removes the specified parameter from the Items (see page 7449) property of a TParams (see page 7434) object.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	Capacity (see page 2748)	Provides access to the internal TList (see page 2868) Capacity property.
R	Count (see page 2749)	Returns the number of items in the collection.
R	ItemClass (see page 2751)	Indicates the class to which the collection's items belong.
	Items (see page 2751)	Lists the items in the collection.
	Items (see page 7449)	Lists the individual field parameters of a TParams (see page 7434) object.
	ParamValues (see page 7451)	Lists the values of individual field parameters that are specified by name.

194.1.2.2 TXMLParams Methods

The methods of the TXMLParams class are listed here.

Protected Methods

	Name	Description
V	AssignTo (see page 26085)	Copies the properties of an object to a destination object.

Public Methods

	Name	Description
	AssignStrings (see page 26084)	Assigns values to the parameters from a list of name/value pairs.

194.1.2.2.1 TXMLParams.AssignStrings Method

Assigns values to the parameters from a list of name/value pairs.

Pascal

```
procedure AssignStrings(Value: TStrings);
```

C++

```
__fastcall AssignStrings(TStrings Value);
```

Description

Use AssignStrings to update the values of the parameters in the Items list to match the values in the list specified by Value. For

each string in Value of the form Name=Value, AssignStrings locates the parameter (if any) that has the same name as the name portion of the string. It then assigns the Value portion of the string as the parameter's value.

AssignStrings makes it very convenient to assign parameter values that are received as the content of a request message. In particular, TSubmitQueryButton generates an HTML control that sends the values in an HTML query form to the Web application. If the query form displays parameter values that should be sent to a provider on the application server, you can call AssignStrings with the ContentFields property of that Web request message.

See Also

Items (see page 7449)

ParamValues (see page 7451)

ContentFields (see page 15177)

TQueryForm (see page 16745)

TSubmitQueryButton (see page 16833)

194.1.2.2.2 TXMLParams.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign (see page 2930)

EConvertError (see page 22724)

194.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	INotifyConnectionChange (see page 26086)	This is class XMLBrokr.INotifyConnectionChange.

194.2.1 INotifyConnectionChange Interface

Class Hierarchy

XMLBrokr.INotifyConnectionChange

Pascal

```
INotifyConnectionChange = interface;
```

C++

```
__interface INotifyConnectionChange;
```

File

XMLBrokr

Description

This is class XMLBrokr.INotifyConnectionChange.

194.2.1.1 INotifyConnectionChange Members

The following tables list the members exposed by INotifyConnectionChange.

Public Methods

	Name	Description
	ConnectionChange (see page 26087)	This is ConnectionChange, a member of class INotifyConnectionChange.

194.2.1.2 INotifyConnectionChange Methods

The methods of the INotifyConnectionChange class are listed here.

Public Methods

	Name	Description
	ConnectionChange (see page 26087)	This is ConnectionChange, a member of class INotifyConnectionChange.

194.2.1.2.1 INotifyConnectionChange.ConnectionChange Method

Pascal

```
procedure ConnectionChange(Sender: TComponent; Connected: Boolean);
```

C++

```
__fastcall ConnectionChange(TComponent * Sender, Boolean Connected);
```

Description

This is ConnectionChange, a member of class INotifyConnectionChange.

194.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FormatXML (see page 26088)	This is function XMLBrokr.FormatXML.

194.3.1 XMLBrokr.FormatXML Function

Pascal

```
function FormatXML(const Value: string; XMLOptions: TXMLOptions): string;
```

C++

```
AnsiString FormatXML(const AnsiString Value, TXMLOptions XMLOptions);
```

File

XMLBrokr

Description

This is function XMLBrokr.FormatXML.

194.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TXMLOption (see page 26089)	This is record XMLBrokr.TXMLOption.

194.4.1 XMLBrokr.TXMLOption Enumeration

Pascal

```
TXMLOption = (
  xoQuote
);
```

C++

```
enum TXMLOption {
  xoQuote
};
```

File

XMLBrokr

Description

This is record XMLBrokr.TXMLOption.

194.5 Types

The following table lists types in this documentation.

Types

Name	Description
TGetErrorResponseEvent (see page 26090)	This is type XMLBrokr.TGetErrorResponseEvent.
TGetResponseEvent (see page 26090)	This is type XMLBrokr.TGetResponseEvent.
TRequestRecordsEvent (see page 26091)	This is type XMLBrokr.TRequestRecordsEvent.
TRequestUpdateEvent (see page 26091)	This is type XMLBrokr.TRequestUpdateEvent.
TXMLOptions (see page 26091)	This is type XMLBrokr.TXMLOptions.

194.5.1 XMLBrokr.TGetErrorResponseEvent Type

Pascal

```
TGetErrorResponseEvent = procedure (Sender: TObject; ErrorCount: Integer; XMLErrors: string;
Request: TWebRequest; Response: TWebResponse; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; ErrorCount: Integer; XMLErrors: string; Request: TWebRequest; Response:
TWebResponse; var Handled: Boolean) ( TGetErrorResponseEvent)();
```

File

XMLBrokr

Description

This is type XMLBrokr.TGetErrorResponseEvent.

194.5.2 XMLBrokr.TGetResponseEvent Type

Pascal

```
TGetResponseEvent = procedure (Sender: TObject;Request: TWebRequest; Response: TWebResponse;
var Handled: Boolean) of object;
```

C++

```
(Sender: TObject;Request: TWebRequest; Response: TWebResponse; var Handled: Boolean) (
TGetResponseEvent)();
```

File

XMLBrokr

Description

This is type XMLBrokr.TGetResponseEvent.

194.5.3 XMLBrokr.TRequestRecordsEvent Type

Pascal

```
TRequestRecordsEvent = procedure (Sender: TObject; Request: TWebRequest; out RecCount:
Integer; var OwnerData: OleVariant; var Records: string) of object;
```

C++

```
(Sender: TObject; Request: TWebRequest; out RecCount: Integer; var OwnerData: OleVariant; var
Records: string) ( TRequestRecordsEvent)();
```

File

XMLBrokr

Description

This is type XMLBrokr.TRequestRecordsEvent.

194.5.4 XMLBrokr.TRequestUpdateEvent Type

Pascal

```
TRequestUpdateEvent = procedure (Sender: TObject; Request: TWebRequest; Response:
TWebResponse; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; Request: TWebRequest; Response: TWebResponse; var Handled: Boolean) (
TRequestUpdateEvent)();
```

File

XMLBrokr

Description

This is type XMLBrokr.TRequestUpdateEvent.

194.5.5 XMLBrokr.TXMLOptions Type

Pascal

```
TXMLOptions = set of TXMLOption;
```

C++

```
TXMLOption TXMLOptions;
```

File

XMLBrokr

Description

This is type XMLBrokr.TXMLOptions.

194.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
sPostDelta (see page 26092)	This is constant XMLBrokr.sPostDelta.
sProducer (see page 26092)	This is constant XMLBrokr.sProducer.
sRedirect (see page 26092)	This is constant XMLBrokr.sRedirect.

194.6.1 XMLBrokr.sPostDelta Constant

Pascal

```
sPostDelta = 'postdelta';
```

C++

```
sPostDelta = 'postdelta';
```

File

XMLBrokr

Description

This is constant XMLBrokr.sPostDelta.

194.6.2 XMLBrokr.sProducer Constant

Pascal

```
sProducer = 'PRODUCER';
```

C++

```
sProducer = 'PRODUCER';
```

File

XMLBrokr

Description

This is constant XMLBrokr.sProducer.

194.6.3 XMLBrokr.sRedirect Constant

Pascal

```
sRedirect = 'REDIRECT';
```

C++

```
sRedirect = 'REDIRECT';
```

File

XMLBrokr

Description

This is constant XMLBrokr.sRedirect.

195 XMLDoc Namespace

This is namespace XMLDoc.

Topics

Name	Description
Classes (see page 26097)	The following table lists classes in this documentation.
Interfaces (see page 26228)	The following table lists interfaces in this documentation.
Functions (see page 26238)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 26243)	The following table lists structs, records, enums in this documentation.
Types (see page 26247)	The following table lists types in this documentation.
Variables (see page 26251)	The following table lists variables in this documentation.

Classes

	Name	Description
	TXMLDocument (see page 26097)	TXMLDocument represents an XML (see page 26144) document.
	TXMLNode (see page 26147)	TXMLNode represents a node in an XML document.
	TXMLNodeCollection (see page 26196)	TXMLNodeCollection represents a node in an XML document whose children are instances of the same repeating element.
	TXMLNodeList (see page 26211)	TXMLNodeList represents a set of nodes in an XML document.

Functions

	Name	Description
	CheckEncoding (see page 26238)	This is function XMLDoc.CheckEncoding.
	CreateDOMNode (see page 26238)	Creates a new DOM node and returns its interface.
	DetectCharEncoding (see page 26239)	This is function XMLDoc.DetectCharEncoding.
	FormatXMLData (see page 26240)	Formats a string of XML so that it is more readable.
	LoadXMLData (see page 26240)	Creates and returns the interface of an XML document object, given a string of XML.
	LoadXMLData (see page 26240)	Creates and returns the interface of an XML document object, given a string of XML.
	LoadXMLDocument (see page 26241)	Returns the interface for an XML document given the name of an XML file.
	NewXMLDocument (see page 26241)	Returns the interface for a new, empty XML document.
	XMLDocError (see page 26242)	Raises an EXMLDocError exception.
	XMLDocError (see page 26242)	Raises an EXMLDocError exception.
	XMLStringToWideString (see page 26242)	This is function XMLDoc.XMLStringToWideString.

Interfaces

	Name	Description
	IXMLDocumentAccess (see page 26228)	This is class XMLDoc.IXMLDocumentAccess.
	IXMLNodeAccess (see page 26230)	This is class XMLDoc.IXMLNodeAccess.

Structs, Records, Enums

	Name	Description
	TNodeChange (see page 26243)	TNodeChange indicates the type of change made to a node in an XML document.
	TNodeClassInfo (see page 26244)	TNodeClassInfo and TNodeClassArray (see page 26247) represent information about the classes that implement XML node interfaces.
	TNodeListOperation (see page 26244)	TNodeListOperation indicates what type of change is about to occur using an IXMLNodeList interface.
	TXMLDocumentSource (see page 26245)	This is record XMLDoc.TXMLDocumentSource.
	TXMLPrologItem (see page 26246)	This is record XMLDoc.TXMLPrologItem.

Types

Name	Description
TNodeChangeEvent (see page 26247)	TNodeChangeEvent is the type for event handlers that respond when XML nodes are changed.
TNodeClassArray (see page 26247)	This is type XMLDoc.TNodeClassArray.
TNodeListNotification (see page 26248)	TNodeListNotification is the type of the callback the TXMLNodeList (see page 26211) calls when it adds, deletes, or creates XML nodes.
TXMLNodeArray (see page 26249)	TXMLNodeArray is a dynamic array of TXMLNode (see page 26147) objects.
TXMLNodeClass (see page 26249)	TXMLNodeClass is a class reference to a TXMLNode (see page 26147) descendant.
TXMLNodeCollectionClass (see page 26250)	TXMLNodeCollectionClass is a class reference to a TXMLNodeCollection (see page 26196) descendant.

Variables

Name	Description
UseXSDBooleanStrings (see page 26251)	This is variable XMLDoc.UseXSDBooleanStrings.

195.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TXMLDocument (see page 26097)	TXMLDocument represents an XML (see page 26144) document.
	TXMLNode (see page 26147)	TXMLNode represents a node in an XML document.
	TXMLNodeCollection (see page 26196)	TXMLNodeCollection represents a node in an XML document whose children are instances of the same repeating element.
	TXMLNodeList (see page 26211)	TXMLNodeList represents a set of nodes in an XML document.

195.1.1 TXMLDocument Class

TXMLDocument represents an XML (see page 26144) document.

Class Hierarchy

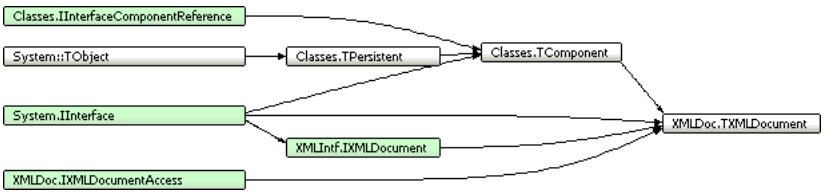

Pascal

```
TXMLDocument = class(TComponent, IInterface, IXMLDocument, IXMLDocumentAccess);
```

C++

```
class TXMLDocument : public TComponent, public IInterface, public IXMLDocument, public
IXMLDocumentAccess;
```

File

XMLDoc

Description

Use TXMLDocument to represent an XML (see page 26144) document. TXMLDocument can read an existing XML (see page 26144) document from a file, it can be associated with an in-memory string that is the contents of an XML (see page 26144) document, or it can create a new, empty XML (see page 26144) document.

You can use TXMLDocument directly to load an XML (see page 26144) document, read and edit it, and save any changes. You can also use TXMLDocument to access the objects generated by the XML (see page 26144) Data Binding wizard.

TXMLDocument uses an external DOM (Document Object Model) parser to analyze the XML (see page 26144) document. This parser is usually assigned by specifying the DOMVendor (see page 26137) property.

When you set the Active (see page 26133) property to true, TXMLDocument uses the DOM parser to parse the XML (see page 26144) document so that you can examine or modify it. In particular, the DocumentElement (see page 26135) property provides an interface to the root node of the document. You can use that interface to access its child nodes and to add or delete child nodes.

TXMLDocument implements the IXMLDocument interface. Applications obtain this interface from TXMLDocument or from the nodes in the document.

When TXMLDocument is created without an Owner, it behaves like an interfaced object. That is, when all references to its interface are released, the TXMLDocument instance is automatically freed. When TXMLDocument is created with an Owner, however, it behaves like any other component, and is freed by its Owner. When you add a TXMLDocument component from the component palette to a form or data module, it is automatically created with an Owner. When the TXMLDocument component is created using the global LoadXMLDocument (see page 26241) function (or by a function that the XML (see page 26144) Data Binding wizard generates to return the root node of the document), the function creates a TXMLDocument instance without an Owner.

See Also

IXMLNode (see page 26411)

EXMLDocError (see page 26381)

IXMLDocument (see page 26382)

LoadXMLDocument (see page 26241)

Using TXMLDocument

Using Code That the XML Data Binding Wizard Generates

195.1.1.1 TXMLDocument Members

The following tables list the members exposed by TXMLDocument.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 26106)	Called when an application uses a component interface.
	_Release (see page 26106)	Called when an application releases a component interface.
	AssignParseOptions (see page 26108)	This is AssignParseOptions, a member of class TXMLDocument.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckActive (see page 26108)	This is CheckActive, a member of class TXMLDocument.
	CheckAutoSave (see page 26108)	This is CheckAutoSave, a member of class TXMLDocument.
	CheckDOM (see page 26109)	This is CheckDOM, a member of class TXMLDocument.
V	DefineProperties (see page 26111)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoAfterClose (see page 26111)	This is DoAfterClose, a member of class TXMLDocument.
	DoAfterOpen (see page 26111)	This is DoAfterOpen, a member of class TXMLDocument.
	DoBeforeClose (see page 26112)	This is DoBeforeClose, a member of class TXMLDocument.
	DoBeforeOpen (see page 26112)	This is DoBeforeOpen, a member of class TXMLDocument.
	DoNodeChange (see page 26112)	This is DoNodeChange, a member of class TXMLDocument.
V	GetActive (see page 26113)	Returns the value of the Active property.
	GetAsyncLoadState (see page 26113)	Returns the value of the AsyncLoadState property.
V	GetChildNodeClass (see page 26113)	This is GetChildNodeClass, a member of class TXMLDocument.
	GetChildNodes (see page 26113)	Returns the value of the ChildNodes property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.

	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDocumentElement (see page 26114)	Returns the value of the DocumentElement property.
	GetDocumentNode (see page 26115)	Returns the value of the Node property.
	GetDocumentObject (see page 26115)	This is GetDocumentObject, a member of class TXMLDocument.
	GetDOMDocument (see page 26115)	Returns the value of the DOMDocument property.
	GetDOMPersist (see page 26115)	This is GetDOMPersist, a member of class TXMLDocument.
	GetEncoding (see page 26116)	Returns the value of the Encoding property.
	GetFileName (see page 26116)	Returns the value of the FileName property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetModified (see page 26116)	Returns the value of the Modified property.
	GetNodeIndentStr (see page 26117)	Returns the value of the NodeIndentStr property.
	GetOptions (see page 26117)	Returns the value of the Options property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetParseOptions (see page 26117)	Returns the value of the ParseOptions property.
	GetPrologNode (see page 26117)	This is GetPrologNode, a member of class TXMLDocument.
	GetPrologValue (see page 26118)	This is GetPrologValue, a member of class TXMLDocument.
	GetSchemaRef (see page 26118)	Returns the value of the SchemaRef property.
	GetStandAlone (see page 26118)	Returns the value of the StandAlone property.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVersion (see page 26118)	Returns the value of the Version property.
	GetXML (see page 26119)	Returns the value of the XML property.
	InternalSetPrologValue (see page 26119)	This is InternalSetPrologValue, a member of class TXMLDocument.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	LoadData (see page 26120)	This is LoadData, a member of class TXMLDocument.
V	Loaded (see page 26120)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadDOMVendor (see page 26123)	This is ReadDOMVendor, a member of class TXMLDocument.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReleaseDoc (see page 26124)	This is ReleaseDoc, a member of class TXMLDocument.
	SaveToUTF8String (see page 26126)	This is SaveToUTF8String, a member of class TXMLDocument.
	SaveToXMLStrings (see page 26127)	This is SaveToXMLStrings, a member of class TXMLDocument.
V	SetActive (see page 26127)	Sets the value of the Active property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDocumentElement (see page 26127)	Sets the value of the DocumentElement property.
	SetDOMDocument (see page 26128)	Sets the value of the DOMDocument property.
	SetEncoding (see page 26128)	Sets the value of the Encoding property.
	SetFileName (see page 26128)	Sets the value of the FileName property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetModified (see page 26129)	This is SetModified, a member of class TXMLDocument.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetNodeIndentStr (see page 26129)	Sets the value of the NodeIndentStr property.
	SetOnAsyncLoad (see page 26129)	Supplies an event handler that responds when the AsyncLoadState property changes.

	SetOptions (see page 26130)	Sets the value of the Options property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParseOptions (see page 26130)	Sets the value of the ParseOptions property.
	SetPrologValue (see page 26130)	This is SetPrologValue, a member of class TXMLDocument.
	SetStandAlone (see page 26130)	Sets the value of the StandAlone property.
	SetVersion (see page 26131)	Sets the value of the Version property.
	SetXML (see page 26131)	Sets the value of the XML property.
	SetXMLStrings (see page 26131)	This is SetXMLStrings, a member of class TXMLDocument.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteDOMVendor (see page 26132)	This is WriteDOMVendor, a member of class TXMLDocument.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.
	XMLStringsChanging (see page 26132)	This is XMLStringsChanging, a member of class TXMLDocument.

Protected Properties

	Name	Description
R	DocBindingInfo (see page 26135)	This is DocBindingInfo, a member of class TXMLDocument.
	DocSource (see page 26135)	This is DocSource, a member of class TXMLDocument.
R	DOMParseOptions (see page 26137)	This is DOMParseOptions, a member of class TXMLDocument.
R	DOMPersist (see page 26137)	This is DOMPersist, a member of class TXMLDocument.
	PrefixID (see page 26142)	This is PrefixID, a member of class TXMLDocument.

Public Methods

	Name	Description
	AddChild (see page 26107)	This is the overview for the AddChild method overload.
	AddChild (see page 26385)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 26107)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 26103)	This is the overview for the Create constructor overload.
	CreateElement (see page 26109)	Creates a new element node that is associated with this document.
	CreateNode (see page 26109)	Creates a new node that is associated with this document.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 26104)	Disposes of a TXMLDocument (see page 26097) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.

	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GeneratePrefix (see page 26112)	Generates a unique namespace prefix that can be used for a node in the document.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetDocBinding (see page 26114)	Replaces the document element with an instance of a specified implementation class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmptyDoc (see page 26119)	Indicates whether the TXMLDocument (see page 26097) instance represents an empty document.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LoadFromFile (see page 26121)	Loads an XML (see page 26144) document and activates it.
	LoadFromStream (see page 26121)	Loads an XML (see page 26144) document from a stream and activates it.
	LoadFromXML (see page 26122)	This is the overview for the LoadFromXML method overload.
	LoadFromXML (see page 26395)	This is the overview for the LoadFromXML method overload.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 26122)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 26123)	Updates the parsed XML (see page 26144) document to reflect external changes.
	RegisterDocBinding (see page 26124)	Registers an implementation class for child nodes with a specified tag name.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Resync (see page 26125)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 26125)	Saves the XML (see page 26144) document to disk.
	SaveToStream (see page 26125)	Saves the XML (see page 26144) document to a stream.
	SaveToXML (see page 26126)	This is the overview for the SaveToXML method overload.
	SaveToXML (see page 26398)	This is the overview for the SaveToXML method overload.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	AsyncLoadState (see page 26134)	Indicates the state of the DOM parser when it parses the XML (see page 26144) document asynchronously.
R	ChildNodes (see page 26135)	Lists all of the child nodes of the document.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	DocumentElement (see page 26135)	Provides access to the root node of the XML (see page 26144) document.
R	DocumentObject (see page 26229)	This is DocumentObject, a member of class IXMLDocumentAccess.
	DOMDocument (see page 26136)	Provides access to the DOM interface for the parsed XML (see page 26144) document.
	DOMImplementation (see page 26137)	This is DOMImplementation, a member of class TXMLDocument.
	Encoding (see page 26138)	Indicates the character set that the XML (see page 26144) document uses to encode characters.
R	Modified (see page 26139)	Indicates whether the XML (see page 26144) document has been modified.
R	Node (see page 26140)	Provides access to the document node for the XML (see page 26144) document.
	NSPrefixBase (see page 26141)	Specifies the base string for automatically-generated namespace prefixes.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	SchemaRef (see page 26142)	Returns the name of the schema associated with this XML (see page 26144) document.
	StandAlone (see page 26143)	Indicates whether the XML (see page 26144) document includes any external markup declarations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Version (see page 26143)	Indicates the XML (see page 26144) version that the document uses.

Published Events

	Name	Description
	AfterClose (see page 26144)	Occurs after the XML (see page 26144) document is closed.
	AfterNodeChange (see page 26145)	Occurs after a change is made to one of the nodes in the document.
	AfterOpen (see page 26145)	Occurs immediately after the XML (see page 26144) document is opened.
	BeforeClose (see page 26146)	Occurs immediately before the XML (see page 26144) document is closed.
	BeforeNodeChange (see page 26146)	Occurs before a change is made to one of the nodes in the document.
	BeforeOpen (see page 26147)	Occurs immediately before the XML (see page 26144) document is opened.
	OnAsyncLoad (see page 26147)	Occurs at various stages during the asynchronous parsing of an XML (see page 26144) document.

Published Properties

	Name	Description
	Active (see page 26133)	Specifies whether the XML (see page 26144) document has been parsed so that the DOM interfaces are available to examine or change its contents.
	DOMVendor (see page 26137)	Specifies the DOM implementation to use for parsing and manipulating the XML (see page 26144) document.
	FileName (see page 26138)	Specifies the file name for the XML (see page 26144) document associated with this TXMLDocument (see page 26097) object.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NodeIndentStr (see page 26140)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 26144) text.
	Options (see page 26141)	Influences various aspects of the document's behavior.

	ParseOptions (see page 26142)	Specifies implementation-specific options that are followed when parsing the XML (see page 26144) document.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XML (see page 26144)	Specifies the content of the XML document associated with this TXMLDocument (see page 26097) object.

195.1.1.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TXMLDocument.Create (DOMString) (see page 26103)	Instantiates a TXMLDocument (see page 26097) object.
	TXMLDocument.Create (TComponent) (see page 26103)	Instantiates a TXMLDocument (see page 26097) object.

195.1.1.2.1 TXMLDocument.Create Constructor (DOMString)

Instantiates a TXMLDocument (see page 26097) object.

Pascal

```
constructor Create(AOwner: TComponent); override; overload;
constructor Create(const AFileName: DOMString); overload; reintroduce;
```

C++

```
virtual __fastcall TXMLDocument(TComponent * AOwner);
__fastcall TXMLDocument(const DOMString AFileName);
```

Description

Call Create to instantiate a TXMLDocument (see page 26097) component at runtime. You do not need to explicitly create TXMLDocument (see page 26097) components that are added to forms or data modules at design time: they are created automatically.

AFileName specifies the file that the new TXMLDocument (see page 26097) instance represents.

AOwner is another component that is responsible for freeing the TXMLDocument (see page 26097) instance. It becomes the value of the Owner property.

TXMLDocument (see page 26097) components that are created at runtime without an owner are freed automatically when all references to their IXMLDocument interface are released. This differs from TXMLDocument (see page 26097) components added to a form or data module at design time or those created with an owner. When added to a form or data module at design time, the object is explicitly freed by its Owner (the form or data module).

See Also

FileName (see page 26138)

IXMLDocument (see page 26382)

Owner (see page 2807)

195.1.1.3 TXMLDocument.Destroy Destructor

Disposes of a TXMLDocument (see page 26097) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLDocument();
```

Description

It is not necessary to explicitly free a TXMLDocument (see page 26097) component. If the component is created with an Owner, that Owner handles the task of freeing the document. If the component is created without an Owner, it is freed automatically when the reference count on its IXMLDocument interface drops to zero.

See Also

Create (see page 26103)

195.1.1.4 TXMLDocument Methods

The methods of the TXMLDocument class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 26106)	Called when an application uses a component interface.
	_Release (see page 26106)	Called when an application releases a component interface.
	AssignParseOptions (see page 26108)	This is AssignParseOptions, a member of class TXMLDocument.
	CheckActive (see page 26108)	This is CheckActive, a member of class TXMLDocument.
	CheckAutoSave (see page 26108)	This is CheckAutoSave, a member of class TXMLDocument.
	CheckDOM (see page 26109)	This is CheckDOM, a member of class TXMLDocument.
V	DefineProperties (see page 26111)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoAfterClose (see page 26111)	This is DoAfterClose, a member of class TXMLDocument.
	DoAfterOpen (see page 26111)	This is DoAfterOpen, a member of class TXMLDocument.
	DoBeforeClose (see page 26112)	This is DoBeforeClose, a member of class TXMLDocument.
	DoBeforeOpen (see page 26112)	This is DoBeforeOpen, a member of class TXMLDocument.
	DoNodeChange (see page 26112)	This is DoNodeChange, a member of class TXMLDocument.
V	GetActive (see page 26113)	Returns the value of the Active property.
	GetAsyncLoadState (see page 26113)	Returns the value of the AsyncLoadState property.
V	GetChildNodeClass (see page 26113)	This is GetChildNodeClass, a member of class TXMLDocument.
	GetChildNodes (see page 26113)	Returns the value of the ChildNodes property.
	GetDocumentElement (see page 26114)	Returns the value of the DocumentElement property.
	GetDocumentNode (see page 26115)	Returns the value of the Node property.
	GetDocumentObject (see page 26115)	This is GetDocumentObject, a member of class TXMLDocument.
	GetDOMDocument (see page 26115)	Returns the value of the DOMDocument property.
	GetDOMPersist (see page 26115)	This is GetDOMPersist, a member of class TXMLDocument.
	GetEncoding (see page 26116)	Returns the value of the Encoding property.
	GetFileName (see page 26116)	Returns the value of the FileName property.
	GetModified (see page 26116)	Returns the value of the Modified property.
	GetNodeIndentStr (see page 26117)	Returns the value of the NodeIndentStr property.
	GetOptions (see page 26117)	Returns the value of the Options property.
	GetParseOptions (see page 26117)	Returns the value of the ParseOptions property.

	GetPrologNode (see page 26117)	This is GetPrologNode, a member of class TXMLDocument.
	GetPrologValue (see page 26118)	This is GetPrologValue, a member of class TXMLDocument.
	GetSchemaRef (see page 26118)	Returns the value of the SchemaRef property.
	GetStandAlone (see page 26118)	Returns the value of the StandAlone property.
	GetVersion (see page 26118)	Returns the value of the Version property.
	GetXML (see page 26119)	Returns the value of the XML property.
	InternalSetPrologValue (see page 26119)	This is InternalSetPrologValue, a member of class TXMLDocument.
	LoadData (see page 26120)	This is LoadData, a member of class TXMLDocument.
V	Loaded (see page 26120)	Initializes the component after the form file has been read into memory.
	ReadDOMVendor (see page 26123)	This is ReadDOMVendor, a member of class TXMLDocument.
	ReleaseDoc (see page 26124)	This is ReleaseDoc, a member of class TXMLDocument.
	SaveToUTF8String (see page 26126)	This is SaveToUTF8String, a member of class TXMLDocument.
	SaveToXMLStrings (see page 26127)	This is SaveToXMLStrings, a member of class TXMLDocument.
V	SetActive (see page 26127)	Sets the value of the Active property.
	SetDocumentElement (see page 26127)	Sets the value of the DocumentElement property.
	SetDOMDocument (see page 26128)	Sets the value of the DOMDocument property.
	SetEncoding (see page 26128)	Sets the value of the Encoding property.
	SetFileName (see page 26128)	Sets the value of the FileName property.
	SetModified (see page 26129)	This is SetModified, a member of class TXMLDocument.
	SetNodeIndentStr (see page 26129)	Sets the value of the NodeIndentStr property.
	SetOnAsyncLoad (see page 26129)	Supplies an event handler that responds when the AsyncLoadState property changes.
	SetOptions (see page 26130)	Sets the value of the Options property.
	SetParseOptions (see page 26130)	Sets the value of the ParseOptions property.
	SetPrologValue (see page 26130)	This is SetPrologValue, a member of class TXMLDocument.
	SetStandAlone (see page 26130)	Sets the value of the StandAlone property.
	SetVersion (see page 26131)	Sets the value of the Version property.
	SetXML (see page 26131)	Sets the value of the XML property.
	SetXMLStrings (see page 26131)	This is SetXMLStrings, a member of class TXMLDocument.
	WriteDOMVendor (see page 26132)	This is WriteDOMVendor, a member of class TXMLDocument.
	XMLStringsChanging (see page 26132)	This is XMLStringsChanging, a member of class TXMLDocument.

Public Methods

	Name	Description
	AddChild (see page 26107)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 26107)	Responds after the last constructor has executed.
	CreateElement (see page 26109)	Creates a new element node that is associated with this document.
	CreateNode (see page 26109)	Creates a new node that is associated with this document.
	GeneratePrefix (see page 26112)	Generates a unique namespace prefix that can be used for a node in the document.
	GetDocBinding (see page 26114)	Replaces the document element with an instance of a specified implementation class.
	IsEmptyDoc (see page 26119)	Indicates whether the TXMLDocument (see page 26097) instance represents an empty document.
	LoadFromFile (see page 26121)	Loads an XML (see page 26144) document and activates it.
	LoadFromStream (see page 26121)	Loads an XML (see page 26144) document from a stream and activates it.
	LoadFromXML (see page 26122)	This is the overview for the LoadFromXML method overload.
V	NewInstance (see page 26122)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Refresh (see page 26123)	Updates the parsed XML (see page 26144) document to reflect external changes.
	RegisterDocBinding (see page 26124)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 26125)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 26125)	Saves the XML (see page 26144) document to disk.
	SaveToStream (see page 26125)	Saves the XML (see page 26144) document to a stream.
	SaveToXML (see page 26126)	This is the overview for the SaveToXML method overload.

195.1.1.4.1 TXMLDocument._AddRef Method

Called when an application uses a component interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef is a basic implementation of the IInterface method, _AddRef.

Note: If the component is a wrapper for a COM object, __AddRef calls the _AddRef method of that COM object, and returns the resulting reference count.

In all other cases, __AddRef simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __AddRef to implement reference counting.

See Also

Invoke (see page 2781)

QueryInterface (see page 2784)

_Release (see page 2769)

ComObject (see page 2793)

_AddRef (see page 22397)

195.1.1.4.2 TXMLDocument._Release Method

Called when an application releases a component interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release is a basic implementation of the IInterface method, _Release.

__Release returns the resulting value of the reference count for the component's interface.

Note:

In all other cases, __Release simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __Release to implement reference counting.

See Also

Invoke (see page 2781)

QueryInterface (see page 2784)

_AddRef (see page 2769)

ComObject (see page 2793)

195.1.1.4.3 AddChild Method

This is the overview for the AddChild method overload.

Overload List

	Name	Description
	TXMLDocument.AddChild (DOMString) (see page 26107)	Creates and adds a node to the child nodes of this document.
	TXMLDocument.AddChild (DOMString, DOMString) (see page 26107)	Creates and adds a node to the child nodes of this document.

195.1.1.4.3.1 TXMLDocument.AddChild Method (DOMString)

Creates and adds a node to the child nodes of this document.

Pascal

```
function AddChild(const TagName: DOMString): IXMLNode; overload;
function AddChild(const TagName: DOMString; const NamespaceURI: DOMString): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode AddChild(const DOMString TagName);
__fastcall IXMLNode AddChild(const DOMString TagName, const DOMString NamespaceURI);
```

Description

Call AddChild to add a new child element node to the document itself. The new node is added to the end of the list maintained by the ChildNodes (see page 26135) property.

TagName is the tag name for the new element node.

NamespaceURI identifies the namespace, if any, that includes the new node's definition.

AddChild returns the interface for the newly created node.

Note: Do not call AddChild to add a child to the document element of this document. When adding data to the XML (see page 26144) document, use the AddChild method of the document element or of the node in the hierarchy that should be the parent of the new node.

See Also

AddChild (see page 26414)

CreateNode (see page 26109)

CreateElement (see page 26109)

ChildNodes (see page 26135)

DocumentElement (see page 26135)

195.1.1.4.4 TXMLDocument.AfterConstruction Method

Responds after the last constructor has executed.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

The System::TObject::AfterConstruction method implemented in System::TObject::TObject does nothing. Override this method when creating a class that takes some action after the object is created. For example, TCustomForm overrides System::TObject::AfterConstruction to generate an OnCreate event.

See Also

System::TObject::BeforeDestruction (see page 22289)

OnCreate (see page 13951)

System::TObject::Create (see page 22287)

195.1.1.4.5 TXMLDocument.AssignParseOptions Method

Pascal

```
procedure AssignParseOptions;
```

C++

```
__fastcall AssignParseOptions();
```

Description

This is AssignParseOptions, a member of class TXMLDocument.

195.1.1.4.6 TXMLDocument.CheckActive Method

Pascal

```
procedure CheckActive;
```

C++

```
__fastcall CheckActive();
```

Description

This is CheckActive, a member of class TXMLDocument.

195.1.1.4.7 TXMLDocument.CheckAutoSave Method

Pascal

```
procedure CheckAutoSave; dynamic;
```

C++

```
__fastcall CheckAutoSave();
```

Description

This is CheckAutoSave, a member of class TXMLDocument.

195.1.1.4.8 TXMLDocument.CheckDOM Method

Pascal

```
procedure CheckDOM; dynamic;
```

C++

```
__fastcall CheckDOM();
```

Description

This is CheckDOM, a member of class TXMLDocument.

195.1.1.4.9 TXMLDocument.CreateElement Method

Creates a new element node that is associated with this document.

Pascal

```
function CreateElement(const TagOrData: DOMString; const NamespaceURI: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateElement(const DOMString TagOrData, const DOMString NamespaceURI);
```

Description

Call CreateElement to create a new generic element node. The resulting node does not have a parent, but can be added to the ChildNodes (see page 26135) list of any node in the document (including this document's ChildNodes (see page 26135) property). It can also be assigned to the DocumentElement (see page 26135) property to replace the entire data hierarchy of this document.

TagOrData provides the tag name of the newly created node.

NamespaceURI identifies the namespace that includes the new node's definition.

CreateElement returns the interface for the new node.

Note: To add a node as the child of another node in this document, it is simpler to call that node's AddChild (see page 26107) method.

See Also

AddChild (see page 26414)

ChildNodes (see page 26135)

DocumentElement (see page 26135)

AddChild (see page 26107)

CreateNode (see page 26109)

195.1.1.4.10 TXMLDocument.CreateNode Method

Creates a new node that is associated with this document.

Pascal

```
function CreateNode(const NameOrData: DOMString; NodeType: TNodeType = ntElement; const
AddlData: DOMString = ''): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateNode(const DOMString NameOrData, TNodeType NodeType = ntElement,
const DOMString AddlData = '');
```

Description

Call CreateNode to create a new generic XML (see page 26144) node. The resulting node does not have a parent, but can be added to the ChildNodes (see page 26135) or AttributeNodes list of any node in the document (including this document's ChildNodes (see page 26135) property). It can also be assigned to the DocumentElement (see page 26135) property to replace the entire data hierarchy of this document.

NameOrData provides the tag name or value of the newly created node. Its interpretation depends on the type of node created, as indicated in the following table:

NodeType	NameOrData
ntElement	The tag name.
ntAttribute	The attribute name.
ntText	The value of the node.
ntCData	The value of the CDATA section.
ntEntityRef	The name of the referenced entity.
ntProcessingInstr	The target of the processing instruction.
ntComment	The value (text) of the comment.
ntDocFragment	Not used.
ntElement	The namespace URI that qualifies the tag name.
ntAttribute	The namespace URI that qualifies the attribute name.
ntProcessingInstr	The content of the processing instruction, except for the target.

NodeType indicates the type of node to create. It can only be one of the types listed in the previous table.

The meaning of AddlData depends on the node type, as indicated in the following table:

NodeType	AddlData
ntElement	The namespace URI that qualifies the tag name.
ntAttribute	The namespace URI that qualifies the attribute name.
ntProcessingInstr	The content of the processing instruction, except for the target.

CreateNode returns the interface for the new node.

Note: To add a node as the child of another node in this document, it is simpler to call that node's AddChild (see page 26107) method.

See Also

AddChild (see page 26414)

ChildNodes (see page 26135)

AttributeNodes (see page 26189)

DocumentElement (see page 26135)

AddChild (see page 26107)

CreateElement (see page 26109)

CreateDOMNode (see page 26238)

195.1.1.4.11 TXMLDocument.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor (see page 2839)

DefineBinaryProperty (see page 2837)

DefineProperties (see page 2931)

DefineProperty (see page 2838)

195.1.1.4.12 TXMLDocument.DoAfterClose Method

Pascal

```
procedure DoAfterClose; dynamic;
```

C++

```
__fastcall DoAfterClose();
```

Description

This is DoAfterClose, a member of class TXMLDocument.

195.1.1.4.13 TXMLDocument.DoAfterOpen Method

Pascal

```
procedure DoAfterOpen; dynamic;
```

C++

```
__fastcall DoAfterOpen();
```

Description

This is DoAfterOpen, a member of class TXMLDocument.

195.1.1.4.14 TXMLDocument.DoBeforeClose Method

Pascal

```
procedure DoBeforeClose; dynamic;
```

C++

```
__fastcall DoBeforeClose();
```

Description

This is DoBeforeClose, a member of class TXMLDocument.

195.1.1.4.15 TXMLDocument.DoBeforeOpen Method

Pascal

```
procedure DoBeforeOpen; dynamic;
```

C++

```
__fastcall DoBeforeOpen();
```

Description

This is DoBeforeOpen, a member of class TXMLDocument.

195.1.1.4.16 TXMLDocument.DoNodeChange Method

Pascal

```
procedure DoNodeChange(const Node: IXMLNode; ChangeType: TNodeChange; BeforeOperation:
Boolean);
```

C++

```
__fastcall DoNodeChange(const IXMLNode Node, TNodeChange ChangeType, Boolean BeforeOperation);
```

Description

This is DoNodeChange, a member of class TXMLDocument.

195.1.1.4.17 TXMLDocument.GeneratePrefix Method

Generates a unique namespace prefix that can be used for a node in the document.

Pascal

```
function GeneratePrefix(const Node: IXMLNode): DOMString;
```

C++

```
__fastcall DOMString GeneratePrefix(const IXMLNode Node);
```

Description

Call GeneratePrefix to generate a new namespace prefix that is not already used in the document to represent a namespace URI. GeneratePrefix uses the NSPrefixBase (see page 26141) property to generate a unique string to act as a namespace prefix. This unique string consists of NSPrefixBase (see page 26141) followed by a number that ensures uniqueness. Thus, for example, if NSPrefixBase (see page 26141) is 'NS', automatically-generated namespace prefixes have the form 'NS1', 'NS2', and so on.

Node (see page 26140) is the interface for the node which will use the newly-generated namespace prefix.

The generated namespace prefixes are only unique within a single document. If two document components have the same value for NSPrefixBase (see page 26141), they will generate the same namespace prefixes.

See Also

NSPrefixBase (see page 26141)

195.1.1.4.18 TXMLDocument.GetActive Method

Returns the value of the Active property.

Pascal

```
function GetActive: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetActive();
```

Description

GetActive is the read implementation of the Active property.

See Also

Active (see page 26403)

195.1.1.4.19 TXMLDocument.GetAsyncLoadState Method

Returns the value of the AsyncLoadState property.

Pascal

```
function GetAsyncLoadState: Integer;
```

C++

```
__fastcall int GetAsyncLoadState();
```

Description

GetAsyncLoadState is the read implementation of the AsyncLoadState property.

See Also

AsyncLoadState (see page 26404)

195.1.1.4.20 TXMLDocument.GetChildNodeClass Method

Pascal

```
function GetChildNodeClass(const Node: IDOMNode): TXMLNodeClass; virtual;
```

C++

```
virtual __fastcall TXMLNodeClass GetChildNodeClass(const IDOMNode Node);
```

Description

This is GetChildNodeClass, a member of class TXMLDocument.

195.1.1.4.21 TXMLDocument.GetChildNodes Method

Returns the value of the ChildNodes property.

Pascal

```
function GetChildNodes: IXMLNodeList;
```

C++

```
__fastcall IXMLNodeList GetChildNodes();
```

Description

GetChildNodes is the read implementation of the ChildNodes property.

See Also

ChildNodes (see page 26405)

195.1.1.4.22 TXMLDocument.GetDocBinding Method

Replaces the document element with an instance of a specified implementation class.

Pascal

```
function GetDocBinding(const TagName: DOMString; DocNodeClass: TClass; NamespaceURI: DOMString
= ''): IXMLNode;
```

C++

```
__fastcall IXMLNode GetDocBinding(const DOMString TagName, TClass DocNodeClass, DOMString
NamespaceURI = '');
```

Description

Call GetDocBinding to specify the implementation class and tag name for the document element. After replacing the specified node with an instance of the specified class, GetDocBinding returns the interface of the document element.

TagName is the tag name of the child node to be replaced. Typically, this identifies the document element.

DocNodeClass is the implementation class to use for all child nodes with the specified tag name.

NamespaceURI is the URI of the namespace in which TagName is defined.

See Also

DocumentElement (see page 26135)

RegisterDocBinding (see page 26124)

ChildNodes (see page 26135)

195.1.1.4.23 TXMLDocument.GetDocumentElement Method

Returns the value of the DocumentElement property.

Pascal

```
function GetDocumentElement: IXMLNode;
```

C++

```
__fastcall IXMLNode GetDocumentElement();
```

Description

GetDocumentElement is the read implementation of the DocumentElement property.

See Also

DocumentElement (see page 26405)

195.1.1.4.24 TXMLDocument.GetDocumentNode Method

Returns the value of the Node property.

Pascal

```
function GetDocumentNode: IXMLNode;
```

C++

```
__fastcall IXMLNode GetDocumentNode();
```

Description

GetDocumentNode is the read implementation of the Node property.

See Also

Node (see page 26407)

195.1.1.4.25 TXMLDocument.GetDocumentObject Method

Pascal

```
function GetDocumentObject: TXMLDocument;
```

C++

```
__fastcall TXMLDocument GetDocumentObject();
```

Description

This is GetDocumentObject, a member of class TXMLDocument.

195.1.1.4.26 TXMLDocument.GetDOMDocument Method

Returns the value of the DOMDocument property.

Pascal

```
function GetDOMDocument: IDOMDocument;
```

C++

```
__fastcall IDOMDocument GetDOMDocument();
```

Description

GetDOMDocument is the read implementation of the DOMDocument property.

See Also

DOMDocument (see page 26405)

195.1.1.4.27 TXMLDocument.GetDOMPersist Method

Pascal

```
function GetDOMPersist: IDOMPersist;
```

C++

```
__fastcall IDOMPersist GetDOMPersist();
```

Description

This is GetDOMPersist, a member of class TXMLDocument.

195.1.1.4.28 TXMLDocument.GetEncoding Method

Returns the value of the Encoding property.

Pascal

```
function GetEncoding: DOMString;
```

C++

```
__fastcall DOMString GetEncoding();
```

Description

GetEncoding is the read implementation of the Encoding property.

See Also

Encoding (see page 26406)

195.1.1.4.29 TXMLDocument.GetFileName Method

Returns the value of the FileName property.

Pascal

```
function GetFileName: DOMString;
```

C++

```
__fastcall DOMString GetFileName();
```

Description

GetFileName is the read implementation of the FileName property.

See Also

FileName (see page 26406)

195.1.1.4.30 TXMLDocument.GetModified Method

Returns the value of the Modified property.

Pascal

```
function GetModified: Boolean;
```

C++

```
__fastcall Boolean GetModified();
```

Description

GetModified is the read implementation of the Modified property.

See Also

Modified (see page 26407)

195.1.1.4.31 TXMLDocument.GetNodeIndentStr Method

Returns the value of the NodeIndentStr property.

Pascal

```
function GetNodeIndentStr: DOMString;
```

C++

```
__fastcall DOMString GetNodeIndentStr();
```

Description

GetNodeIndentStr is the read implementation of the NodeIndentStr property.

See Also

NodeIndentStr (see page 26408)

195.1.1.4.32 TXMLDocument.GetOptions Method

Returns the value of the Options property.

Pascal

```
function GetOptions: TXMLDocOptions;
```

C++

```
__fastcall TXMLDocOptions GetOptions();
```

Description

GetOptions is the read implementation of the Options property.

See Also

Options (see page 26408)

195.1.1.4.33 TXMLDocument.GetParseOptions Method

Returns the value of the ParseOptions property.

Pascal

```
function GetParseOptions: TParseOptions;
```

C++

```
__fastcall TParseOptions GetParseOptions();
```

Description

GetParseOptions is the read implementation of the ParseOptions property.

See Also

ParseOptions (see page 26409)

195.1.1.4.34 TXMLDocument.GetPrologNode Method

Pascal

```
function GetPrologNode: IXMLNode;
```

C++

```
__fastcall IXMLNode GetPrologNode();
```

Description

This is GetPrologNode, a member of class TXMLDocument.

195.1.1.4.35 TXMLDocument.GetPrologValue Method

Pascal

```
function GetPrologValue(PrologItem: TXMLPrologItem; const Default: DOMString = ''): DOMString;
```

C++

```
__fastcall DOMString GetPrologValue(TXMLPrologItem PrologItem, const DOMString Default = '');
```

Description

This is GetPrologValue, a member of class TXMLDocument.

195.1.1.4.36 TXMLDocument.GetSchemaRef Method

Returns the value of the SchemaRef property.

Pascal

```
function GetSchemaRef: DOMString;
```

C++

```
__fastcall DOMString GetSchemaRef();
```

Description

GetSchemaRef is the read implementation of the SchemaRef property.

See Also

SchemaRef (see page 26409)

195.1.1.4.37 TXMLDocument.GetStandAlone Method

Returns the value of the StandAlone property.

Pascal

```
function GetStandAlone: DOMString;
```

C++

```
__fastcall DOMString GetStandAlone();
```

Description

GetStandAlone is the read implementation of the StandAlone property.

See Also

StandAlone (see page 26410)

195.1.1.4.38 TXMLDocument.GetVersion Method

Returns the value of the Version property.

Pascal

```
function GetVersion: DOMString;
```

C++

```
__fastcall DOMString GetVersion();
```

Description

GetVersion is the read implementation of the Version property.

See Also

Version (see page 26410)

195.1.1.4.39 TXMLDocument.GetXML Method

Returns the value of the XML property.

Pascal

```
function GetXML: TSTrings;
```

C++

```
__fastcall TSTrings GetXML();
```

Description

GetXML is the read implementation of the XML property.

See Also

XML (see page 26410)

195.1.1.4.40 TXMLDocument.InternalSetPrologValue Method

Pascal

```
function InternalSetPrologValue(const PrologNode: IXMLNode; const Value: Variant; PrologItem:
TXMLPrologItem): string;
```

C++

```
__fastcall AnsiString InternalSetPrologValue(const IXMLNode PrologNode, const Variant Value,
TXMLPrologItem PrologItem);
```

Description

This is InternalSetPrologValue, a member of class TXMLDocument.

195.1.1.4.41 TXMLDocument.IsEmptyDoc Method

Indicates whether the TXMLDocument (see page 26097) instance represents an empty document.

Pascal

```
function IsEmptyDoc: Boolean;
```

C++

```
__fastcall Boolean IsEmptyDoc();
```

Description

Call IsEmptyDoc to determine whether the TXMLDocument (see page 26097) instance represents an empty document.

IsEmptyDoc returns true if the DOMDocument (see page 26136) property is not set (for example, if there is no DOM implementation) or if this object represents a document with no child nodes.

See Also

XML (see page 26144)

ChildNodes (see page 26135)

DOMDocument (see page 26136)

195.1.1.4.42 TXMLDocument.LoadData Method

Pascal

```
procedure LoadData; dynamic;
```

C++

```
__fastcall LoadData();
```

Description

This is LoadData, a member of class TXMLDocument.

195.1.1.4.43 TXMLDocument.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState (see page 2804)

Create (see page 2766)

TReader (see page 2933)

195.1.1.4.44 TXMLDocument.LoadFromFile Method

Loads an XML (see page 26144) document and activates it.

Pascal

```
procedure LoadFromFile(const AFileName: DOMString = '');
```

C++

```
__fastcall LoadFromFile(const DOMString AFileName = '');
```

Description

Call LoadFromFile to load the XML (see page 26144) document specified by AFileName and set the Active (see page 26133) property to true so that you can examine or modify the document.

AFileName is the name of the XML (see page 26144) document to load from disk. If AFileName is an empty string, TXMLDocument (see page 26097) uses the value of the FileName (see page 26138) property. If AFileName is not an empty string, TXMLDocument (see page 26097) changes the FileName (see page 26138) property to AFileName.

Once you have loaded an XML (see page 26144) document, any changes you make to the document are not saved back to disk until you call the SaveToFile (see page 26125) method.

See Also

SaveToFile (see page 26125)

FileName (see page 26138)

Active (see page 26133)

Refresh (see page 26123)

LoadFromXML (see page 26122)

LoadFromStream (see page 26121)

195.1.1.4.45 TXMLDocument.LoadFromStream Method

Loads an XML (see page 26144) document from a stream and activates it.

Pascal

```
procedure LoadFromStream(const Stream: TStream; EncodingType: TXMLEncodingType = xetUnknown);
```

C++

```
__fastcall LoadFromStream(const TStream Stream, TXMLEncodingType EncodingType = xetUnknown);
```

Description

Call LoadFromStream to load the XML (see page 26144) document from a stream.

Stream is a stream object that can be used to read the string of XML (see page 26144) that makes up the document.

EncodingType indicates the character set that is used for Stream. If EncodingType is xetUnknown (the default), LoadFromStream attempts to infer the encoding type from the XML (see page 26144) encoding declaration at the beginning of the document.

After loading the document from Stream, LoadFromStream sets the Active (see page 26133) property to true.

See Also

SaveToStream (see page 26125)

LoadFromFile (see page 26121)

Active (see page 26133)

LoadFromXML (see page 26122)

195.1.1.4.46 LoadFromXML Method

This is the overview for the LoadFromXML method overload.

Overload List

	Name	Description
	TXMLDocument.LoadFromXML (DOMString) (see page 26122)	Loads a string representation of an XML (see page 26144) document and activates it.
	TXMLDocument.LoadFromXML (string) (see page 26122)	Loads a string representation of an XML (see page 26144) document and activates it.

195.1.1.4.46.1 TXMLDocument.LoadFromXML Method (DOMString)

Loads a string representation of an XML (see page 26144) document and activates it.

Pascal

```
procedure LoadFromXML(const XML: string); overload;
procedure LoadFromXML(const XML: DOMString); overload;
```

C++

```
__fastcall LoadFromXML(const AnsiString XML);
__fastcall LoadFromXML(const DOMString XML);
```

Description

Call LoadFromXML to assign a string as the value of the XML (see page 26144) document. Unlike the XML (see page 26144) property, which lets you assign XML (see page 26144) on a line-by-line basis, LoadFromXML treats the text of the XML (see page 26144) document as a whole.

The XML (see page 26144) parameter is a string containing the text of an XML (see page 26144) document. It should represent the XML (see page 26144) text encoded using UTF-8, Unicode, or UTF-16.

After assigning the XML (see page 26144) property as the contents of the document, LoadFromXML sets the Active (see page 26133) property to true.

See Also

SaveToXML (see page 26126)

LoadFromFile (see page 26121)

Active (see page 26133)

LoadFromStream (see page 26121)

XML (see page 26144)

195.1.1.4.47 TXMLDocument.NewInstance Method

Allocates memory for an instance of an object type and returns a pointer to that new instance.

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
virtual __fastcall TObject * NewInstance();
```

Description

All constructors call System::TObject::NewInstance automatically. System::TObject::NewInstance calls System::TObject::InstanceSize to determine how much memory to allocate from the heap to contain a particular instance. Do not call System::TObject::NewInstance directly.

Override System::TObject::NewInstance only for special memory allocation requirements. For example, when allocating a large number of identical objects that all need to be in memory at the same time, you could allocate a single block of memory for the entire group, then override System::TObject::NewInstance to use part of that larger block for each instance.

If you override System::TObject::NewInstance to allocate memory, you may need to override System::TObject::FreeInstance to deallocate the memory.

Note: By default System::TObject::NewInstance calls System::TObject::InitInstance.

See Also

System::TObject::FreeInstance (see page 22303)

System::TObject::InstanceSize (see page 22306)

System::TObject::InitInstance (see page 22305)

195.1.1.4.48 TXMLDocument.ReadDOMVendor Method

Pascal

```
procedure ReadDOMVendor(Reader: TReader);
```

C++

```
__fastcall ReadDOMVendor(TReader Reader);
```

Description

This is ReadDOMVendor, a member of class TXMLDocument.

195.1.1.4.49 TXMLDocument.Refresh Method

Updates the parsed XML (see page 26144) document to reflect external changes.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh when the XML (see page 26144) document specified by FileName (see page 26138) has changed due to an external cause. Refresh updates the parsed version of the XML (see page 26144) document (which can be traversed starting with DocumentElement (see page 26135)) to reflect the current contents of the file specified by FileName (see page 26138).

The Active (see page 26133) property must be true when you call Refresh. If the XML (see page 26144) document is not active when you call Refresh, TXMLDocument (see page 26097) raises an EXMLDocError exception.

Note: Do not call Refresh after making changes directly to the XML (see page 26144) property, because assigning a value to XML (see page 26144) automatically closes the XML (see page 26144) document. To refresh the XML (see page 26144) document after assigning a new value to XML (see page 26144), set the Active (see page 26133) property to true.

See Also

FileName (see page 26138)

Active (see page 26133)

DOMDocument (see page 26136)

DocumentElement (see page 26135)

Resync (see page 26125)

195.1.1.4.50 TXMLDocument.RegisterDocBinding Method

Registers an implementation class for child nodes with a specified tag name.

Pascal

```
procedure RegisterDocBinding(const TagName: DOMString; DocNodeClass: TClass; NamespaceURI:
DOMString = '');
```

C++

```
__fastcall RegisterDocBinding(const DOMString TagName, TClass DocNodeClass, DOMString
NamespaceURI = '');
```

Description

Call RegisterDocBinding to specify the implementation class for one of the child nodes of the document node. RegisterDocBinding replaces the specified node with an instance of the specified class.

TagName is the tag name of the child node to be replaced. Typically, this identifies the document element.

DocNodeClass is the implementation class to use for all child nodes with the specified tag name.

NamespaceURI is the URI of the namespace in which TagName is defined.

See Also

DocumentElement (see page 26135)

GetDocBinding (see page 26114)

ChildNodes (see page 26135)

195.1.1.4.51 TXMLDocument.ReleaseDoc Method

Pascal

```
procedure ReleaseDoc(const CheckSave: Boolean = True); dynamic;
```

C++

```
__fastcall ReleaseDoc(const Boolean CheckSave = True);
```

Description

This is ReleaseDoc, a member of class TXMLDocument.

195.1.1.4.52 TXMLDocument.Resync Method

Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

Pascal

```
procedure Resync;
```

C++

```
__fastcall Resync();
```

Description

Resync forces the document to reread the values of its child nodes and attribute nodes, discarding any changes that have not been propagated to the underlying DOM implementation.

See Also

ChildNodes (see page 26135)

Refresh (see page 26123)

195.1.1.4.53 TXMLDocument.SaveToFile Method

Saves the XML (see page 26144) document to disk.

Pascal

```
procedure SaveToFile(const AFileName: DOMString = ''); dynamic;
```

C++

```
__fastcall SaveToFile(const DOMString AFileName = '');
```

Description

Call SaveToFile to save any modifications you have made to the parsed XML (see page 26144) document.

AFileName is the name of the file to save. If AFileName is an empty string, TXMLDocument (see page 26097) uses the value of the FileName (see page 26138) property.

See Also

LoadFromFile (see page 26121)

FileName (see page 26138)

SaveToXML (see page 26126)

SaveToStream (see page 26125)

Using Code That the XML Data Binding Wizard Generates

195.1.1.4.54 TXMLDocument.SaveToStream Method

Saves the XML (see page 26144) document to a stream.

Pascal

```
procedure SaveToStream(const Stream: TStream);
```

C++

```
__fastcall SaveToStream(const TStream Stream);
```

Description

Call SaveToStream to save the contents of the XML (see page 26144) document to the stream specified by Stream. Regardless of the encoding system of the original XML (see page 26144) document, SaveToStream always saves the stream in UTF-16.

See Also

LoadFromStream (see page 26121)

SaveToXML (see page 26126)

SaveToFile (see page 26125)

195.1.1.4.55 TXMLDocument.SaveToUTF8String Method

Pascal

```
procedure SaveToUTF8String(var XML: string);
```

C++

```
__fastcall SaveToUTF8String(AnsiString XML);
```

Description

This is SaveToUTF8String, a member of class TXMLDocument.

195.1.1.4.56 SaveToXML Method

This is the overview for the SaveToXML method overload.

Overload List

	Name	Description
	TXMLDocument.SaveToXML (DOMString) (see page 26126)	Saves the XML (see page 26144) document to a string-type variable.
	TXMLDocument.SaveToXML (string) (see page 26126)	Saves the XML (see page 26144) document to a string-type variable.

195.1.1.4.56.1 TXMLDocument.SaveToXML Method (DOMString)

Saves the XML (see page 26144) document to a string-type variable.

Pascal

```
procedure SaveToXML(var XML: DOMString); overload;
procedure SaveToXML(var XML: string); overload;
```

C++

```
__fastcall SaveToXML(DOMString XML);
__fastcall SaveToXML(AnsiString XML);
```

Description

Call SaveToXML to save the contents of the XML (see page 26144) document to the string-type variable specified by XML (see page 26144). SaveToXML writes the contents of XML (see page 26144) document using UTF-8 or UTF-16 as an encoding system, depending on the type of the XML (see page 26144) parameter.

Unlike the XML (see page 26144) property, which lets you write individual lines from the XML (see page 26144) document,

SaveToXML writes the entire text of the XML (see page 26144) document.

See Also

LoadFromXML (see page 26122)

SaveToStream (see page 26125)

SaveToFile (see page 26125)

XML (see page 26144)

195.1.1.4.57 TXMLDocument.SaveToXMLStrings Method

Pascal

```
procedure SaveToXMLStrings;
```

C++

```
__fastcall SaveToXMLStrings();
```

Description

This is SaveToXMLStrings, a member of class TXMLDocument.

195.1.1.4.58 TXMLDocument.SetActive Method

Sets the value of the Active property.

Pascal

```
procedure SetActive(const Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetActive(const Boolean Value);
```

Description

SetActive is the write implementation of the Active property.

Value is the new value to set.

See Also

Active (see page 26403)

195.1.1.4.59 TXMLDocument.SetDocumentElement Method

Sets the value of the DocumentElement property.

Pascal

```
procedure SetDocumentElement(const Value: IXMLNode);
```

C++

```
__fastcall SetDocumentElement(const IXMLNode Value);
```

Description

SetDocumentElement is the write implementation of the DocumentElement property.

Value is the interface for the node tree that replaces the current document element.

See Also

DocumentElement (see page 26405)

195.1.1.4.60 TXMLDocument.SetDOMDocument Method

Sets the value of the DOMDocument property.

Pascal

```
procedure SetDOMDocument(const Value: IDOMDocument);
```

C++

```
__fastcall SetDOMDocument(const IDOMDocument Value);
```

Description

SetDOMDocument is the write implementation of the DOMDocument property.

Value is the DOM interface for the document to assign.

See Also

DOMDocument (see page 26405)

195.1.1.4.61 TXMLDocument.SetEncoding Method

Sets the value of the Encoding property.

Pascal

```
procedure SetEncoding(const Value: DOMString);
```

C++

```
__fastcall SetEncoding(const DOMString Value);
```

Description

SetEncoding is the write implementation of the Encoding property.

Value is the value of the encoding attribute of the document type node.

See Also

Encoding (see page 26406)

195.1.1.4.62 TXMLDocument.SetFileName Method

Sets the value of the FileName property.

Pascal

```
procedure SetFileName(const Value: DOMString);
```

C++

```
__fastcall SetFileName(const DOMString Value);
```

Description

SetFileName is the write implementation of the FileName property.

Value is the name of an XML document or, if the vendor supports it, an URL for the XML document.

See Also

FileName (see page 26406)

195.1.1.4.63 TXMLDocument.SetModified Method

Pascal

```
procedure SetModified(const Value: Boolean);
```

C++

```
__fastcall SetModified(const Boolean Value);
```

Description

This is SetModified, a member of class TXMLDocument.

195.1.1.4.64 TXMLDocument.SetNodeIndentStr Method

Sets the value of the NodeIndentStr property.

Pascal

```
procedure SetNodeIndentStr(const Value: DOMString);
```

C++

```
__fastcall SetNodeIndentStr(const DOMString Value);
```

Description

SetNodeIndentStr is the write implementation of the NodeIndentStr property.

Value is the string that is inserted before nested nodes to indicate a level of nesting.

See Also

NodeIndentStr (see page 26408)

195.1.1.4.65 TXMLDocument.SetOnAsyncLoad Method

Supplies an event handler that responds when the AsyncLoadState property changes.

Pascal

```
procedure SetOnAsyncLoad(const Value: TAsyncEventHandler);
```

C++

```
__fastcall SetOnAsyncLoad(const TAsyncEventHandler Value);
```

Description

Use SetOnAsyncLoad to supply a callback that responds when the asynchronous parsing of an XML document progresses to a certain point.

Value is a method that is called when the AsyncLoadState property changes to reflect a new stage in the parsing process.

See Also

AsyncLoadState (see page 26404)

195.1.1.4.66 TXMLDocument.SetOptions Method

Sets the value of the Options property.

Pascal

```
procedure SetOptions(const Value: TXMLDocOptions);
```

C++

```
__fastcall SetOptions(const TXMLDocOptions Value);
```

Description

GetOptions is the write implementation of the Options property.

Value is the set of options to assign.

See Also

Options (see page 26408)

195.1.1.4.67 TXMLDocument.SetParseOptions Method

Sets the value of the ParseOptions property.

Pascal

```
procedure SetParseOptions(const Value: TParseOptions);
```

C++

```
__fastcall SetParseOptions(const TParseOptions Value);
```

Description

GetParseOptions is the write implementation of the ParseOptions property.

Value is the set of parser options to assign.

See Also

ParseOptions (see page 26409)

195.1.1.4.68 TXMLDocument.SetPrologValue Method

Pascal

```
procedure SetPrologValue(const Value: Variant; PrologItem: TXMLPrologItem);
```

C++

```
__fastcall SetPrologValue(const Variant Value, TXMLPrologItem PrologItem);
```

Description

This is SetPrologValue, a member of class TXMLDocument.

195.1.1.4.69 TXMLDocument.SetStandAlone Method

Sets the value of the StandAlone property.

Pascal

```
procedure SetStandAlone(const Value: DOMString);
```

C++

```
__fastcall SetStandAlone(const DOMString Value);
```

Description

SetStandAlone is the write implementation of the StandAlone property.

Value is the value of the stand-alone attribute of the document type node.

See Also

StandAlone (see page 26410)

195.1.1.4.70 TXMLDocument.SetVersion Method

Sets the value of the Version property.

Pascal

```
procedure SetVersion(const Value: DOMString);
```

C++

```
__fastcall SetVersion(const DOMString Value);
```

Description

SetVersion is the write implementation of the Version property.

Value is the value of the version attribute of the document type node.

See Also

Version (see page 26410)

195.1.1.4.71 TXMLDocument.SetXML Method

Sets the value of the XML property.

Pascal

```
procedure SetXML(const Value: TStrings);
```

C++

```
__fastcall SetXML(const TStrings Value);
```

Description

SetXML is the write implementation of the XML property.

Value contains the raw (unparsed) XML to assign.

See Also

XML (see page 26410)

195.1.1.4.72 TXMLDocument.SetXMLStrings Method

Pascal

```
procedure SetXMLStrings(const Value: string);
```

C++

```
__fastcall SetXMLStrings(const AnsiString Value);
```

Description

This is SetXMLStrings, a member of class TXMLDocument.

195.1.1.4.73 TXMLDocument.WriteDOMVendor Method

Pascal

```
procedure WriteDOMVendor(Writer: TWriter);
```

C++

```
__fastcall WriteDOMVendor(TWriter Writer);
```

Description

This is WriteDOMVendor, a member of class TXMLDocument.

195.1.1.4.74 TXMLDocument.XMLStringsChanging Method

Pascal

```
procedure XMLStringsChanging(Sender: TObject);
```

C++

```
__fastcall XMLStringsChanging(TObject * Sender);
```

Description

This is XMLStringsChanging, a member of class TXMLDocument.

195.1.1.5 TXMLDocument Properties

The properties of the TXMLDocument class are listed here.

Protected Properties

	Name	Description
R	DocBindingInfo (see page 26135)	This is DocBindingInfo, a member of class TXMLDocument.
	DocSource (see page 26135)	This is DocSource, a member of class TXMLDocument.
R	DOMParseOptions (see page 26137)	This is DOMParseOptions, a member of class TXMLDocument.
R	DOMPersist (see page 26137)	This is DOMPersist, a member of class TXMLDocument.
	PrefixID (see page 26142)	This is PrefixID, a member of class TXMLDocument.

Public Properties

	Name	Description
R	AsyncLoadState (see page 26134)	Indicates the state of the DOM parser when it parses the XML (see page 26144) document asynchronously.
R	ChildNodes (see page 26135)	Lists all of the child nodes of the document.
	DocumentElement (see page 26135)	Provides access to the root node of the XML (see page 26144) document.
	DOMDocument (see page 26136)	Provides access to the DOM interface for the parsed XML (see page 26144) document.
	DOMImplementation (see page 26137)	This is DOMImplementation, a member of class TXMLDocument.
	Encoding (see page 26138)	Indicates the character set that the XML (see page 26144) document uses to encode characters.
R	Modified (see page 26139)	Indicates whether the XML (see page 26144) document has been modified.
R	Node (see page 26140)	Provides access to the document node for the XML (see page 26144) document.
	NSPrefixBase (see page 26141)	Specifies the base string for automatically-generated namespace prefixes.

R	SchemaRef (see page 26142)	Returns the name of the schema associated with this XML (see page 26144) document.
	StandAlone (see page 26143)	Indicates whether the XML (see page 26144) document includes any external markup declarations.
	Version (see page 26143)	Indicates the XML (see page 26144) version that the document uses.

Published Properties

	Name	Description
	Active (see page 26133)	Specifies whether the XML (see page 26144) document has been parsed so that the DOM interfaces are available to examine or change its contents.
	DOMVendor (see page 26137)	Specifies the DOM implementation to use for parsing and manipulating the XML (see page 26144) document.
	FileName (see page 26138)	Specifies the file name for the XML (see page 26144) document associated with this TXMLDocument (see page 26097) object.
	NodeIndentStr (see page 26140)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 26144) text.
	Options (see page 26141)	Influences various aspects of the document's behavior.
	ParseOptions (see page 26142)	Specifies implementation-specific options that are followed when parsing the XML (see page 26144) document.
	XML (see page 26144)	Specifies the content of the XML document associated with this TXMLDocument (see page 26097) object.

195.1.1.5.1 TXMLDocument.Active Property

Specifies whether the XML (see page 26144) document has been parsed so that the DOM interfaces are available to examine or change its contents.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before you can use TXMLDocument (see page 26097) to examine or change the contents of an XML (see page 26144) document, it must be active. Setting Active to true causes TXMLDocument (see page 26097) to do the following:

1.Generate a BeforeOpen (see page 26147) event.

2.If necessary, load the DOM parser and assign the DOMImplementation (see page 26137) property.

3.Parse the XML (see page 26144) document specified by FileName (see page 26138) or, if FileName (see page 26138) is an empty string, by the XML (see page 26144) property using the options specified by ParseOptions (see page 26142). If both FileName (see page 26138) and XML (see page 26144) are empty strings, TXMLDocument (see page 26097) creates an empty document. The interface for the parsed document is assigned to the DOMDocument (see page 26136) property.

4. Generate an AfterOpen (see page 26145) event.

Note: Once the DOMDocument (see page 26136) property is assigned, you can also use the DocumentElement (see page 26135) property.

Setting Active to false causes TXMLDocument (see page 26097) to do the following:

1.Generate a BeforeClose (see page 26146) event.

2.Release all of the DOM interfaces, such as DOMDocument (see page 26136) and DocumentElement (see page 26135), that represent the parsed XML (see page 26144) document.

3. Generate an AfterClose (see page 26144) event.

See Also

DOMDocument (see page 26136)

DocumentElement (see page 26135)

FileName (see page 26138)

XML (see page 26144)

ParseOptions (see page 26142)

BeforeOpen (see page 26147)

AfterOpen (see page 26145)

domimplementation (see page 26137)

BeforeClose (see page 26146)

AfterClose (see page 26144)

Using TXMLDocument

195.1.1.5.2 TXMLDocument.AsyncLoadState Property

Indicates the state of the DOM parser when it parses the XML (see page 26144) document asynchronously.

Pascal

```
property AsyncLoadState: Integer;
```

C++

```
__property int AsyncLoadState;
```

Description

Use AsyncLoadState to determine that status of the DOM parser when it is parsing an XML (see page 26144) document asynchronously. AsyncLoadState is only meaningful when the value of ParseOptions (see page 26142) is poAsyncLoad.

The value of AsyncLoadState is vendor-specific. That is, each DOM parser that loads and parses an XML (see page 26144) document asynchronously supplies its own flags to indicate state information.

When the DOM vendor is MSXML, AsyncLoadState can have any of the following values:

Value	Meaning
0	Uninitialized: The AsyncLoadState is not assigned. (Usually indicates that the parser has not tried to parse a document or it is not parsing the document asynchronously.)
1	Loading: The parser is currently parsing the structure of the XML (see page 26144) document.
2	Loaded (see page 26120): The parser has finished reading and parsing the structure of the XML (see page 26144) document, but the corresponding DOM interfaces are not yet available.
3	Interactive: The parser has parsed the structure of the XML (see page 26144) document, the corresponding DOM interfaces are available, but the parser has not finished reading all the node values.
4	Complete: The parser has fully read and parsed the XML (see page 26144) document.

See Also

OnAsyncLoad (see page 26147)

ParseOptions (see page 26142)

195.1.1.5.3 TXMLDocument.ChildNodes Property

Lists all of the child nodes of the document.

Pascal

```
property ChildNodes: IXMLNodeList;
```

C++

```
__property IXMLNodeList ChildNodes;
```

Description

Use ChildNodes to access all of the nodes that are children of the document object. In addition to DocumentElement (see page 26135), which is the root of the data hierarchy, ChildNodes can include nodes for comments, processing instructions, and so on.

Note: ChildNodes is not available unless the XML (see page 26144) document is Active (see page 26133).

See Also

DocumentElement (see page 26135)

Active (see page 26133)

195.1.1.5.4 TXMLDocument.DocBindingInfo Property

Pascal

```
property DocBindingInfo: TNodeClassArray;
```

C++

```
__property TNodeClassArray DocBindingInfo;
```

Description

This is DocBindingInfo, a member of class TXMLDocument.

195.1.1.5.5 TXMLDocument.DocSource Property

Pascal

```
property DocSource: TXMLDocumentSource;
```

C++

```
__property TXMLDocumentSource DocSource;
```

Description

This is DocSource, a member of class TXMLDocument.

195.1.1.5.6 TXMLDocument.DocumentElement Property

Provides access to the root node of the XML (see page 26144) document.

Pascal

```
property DocumentElement: IXMLNode;
```

C++

```
__property IXMLNode DocumentElement;
```

Description

Use DocumentElement to access the root node of the XML (see page 26144) document. Because XML (see page 26144) documents are hierarchical in structure, DocumentElement is the entry point to all data in the XML (see page 26144) document. Setting DocumentElement replaces the entire data hierarchy with a new hierarchy of nodes.

Note: For some DOM implementations, if you set DocumentElement, all of the new nodes must already be associated with this document. That is, the new value should be a node that came from the existing node hierarchy or that was created using the CreateNode (see page 26109) method.

The IXMLNode interface includes properties and methods for examining and changing the text and attributes of a node, for adding and deleting child nodes, and for traversing the hierarchy of nodes rooted at that node.

Note: DocumentElement is not available unless the XML (see page 26144) document is Active (see page 26133).

See Also

DOMDocument (see page 26136)

Active (see page 26133)

DocumentElement

CreateNode (see page 26109)

GetDocBinding (see page 26114)

Using TXMLDocument

Working with XML Nodes

195.1.1.5.7 TXMLDocument.DOMDocument Property

Provides access to the DOM interface for the parsed XML (see page 26144) document.

Pascal

```
property DOMDocument: IDOMDocument;
```

C++

```
__property IDOMDocument DOMDocument;
```

Description

Use DOMDocument to directly access the IDOMDocument interface for the parsed XML (see page 26144) document. This is the interface supported by the DOM parser for the document that is represented in raw (unparsed) form by the FileName (see page 26138) or XML (see page 26144) property.

DOMDocument provides access to the IDOMImplementation and IDOMElement interfaces that are surfaced as the DOMImplementation (see page 26137) and DocumentElement (see page 26135) properties. It also provides access to an IDOMDocumentType interface that describes the structure of the XML (see page 26144) document, and supports a number of methods for creating elements within the document.

Most applications do not need to use the DOMDocument property directly. Instead, it supplies the underlying support for parsing and manipulating the nodes accessed through DocumentElement (see page 26135). If you do want to make use of DOMDocument's low-level support, you must be familiar with the W3C Document Object Model (DOM) Level 1 Specification.

Note: DOMDocument is not available unless the XML (see page 26144) document is Active (see page 26133).

See Also

Active (see page 26133)

DocumentElement (see page 26135)

domimplementation (see page 26137)

XML (see page 26144)

FileName (see page 26138)

195.1.1.5.8 TXMLDocument.DOMImplementation Property

Pascal

```
property DOMImplementation: IDOMImplementation;
```

C++

```
__property IDOMImplementation DOMImplementation;
```

Description

This is DOMImplementation, a member of class TXMLDocument.

195.1.1.5.9 TXMLDocument.DOMParseOptions Property

Pascal

```
property DOMParseOptions: IDOMParseOptions;
```

C++

```
__property IDOMParseOptions DOMParseOptions;
```

Description

This is DOMParseOptions, a member of class TXMLDocument.

195.1.1.5.10 TXMLDocument.DOMPersist Property

Pascal

```
property DOMPersist: IDOMPersist;
```

C++

```
__property IDOMPersist DOMPersist;
```

Description

This is DOMPersist, a member of class TXMLDocument.

195.1.1.5.11 TXMLDocument.DOMVendor Property

Specifies the DOM implementation to use for parsing and manipulating the XML (see page 26144) document.

Pascal

```
property DOMVendor: TDOMVendor;
```

C++

```
__property TDOMVendor DOMVendor;
```

Description

DOMVendor is the object that represents a registered DOM implementation. TXMLDocument (see page 26097) uses this object to obtain an IDOMImplementation interface, which it can use to access all the other DOM interfaces needed for parsing and manipulating the XML (see page 26144) document. Once DOMVendor is assigned, TXMLDocument (see page 26097) obtains the IDOMImplementation interface (when needed) and assigns it to the DOMImplementation (see page 26137) property.

DOMVendor must be registered before it can be used to obtain an IDOMImplementation interface. For a list of all the currently registered DOM implementations, use the global DOMVendors variable. To obtain a specific DOM implementation, given its description, use the global GetDOMVendor function. Use the RegisterDOMVendor procedure to register a new DOM implementation.

You must assign a value to either DOMVendor or DOMImplementation (see page 26137) before you can set the Active (see page 26133) property to true.

See Also

domimplementation (see page 26137)

DOMVendors

Active (see page 26133)

RegisterDOMVendor

Active (see page 26133)

GetDOMVendor

Using TXMLDocument

195.1.1.5.12 TXMLDocument.Encoding Property

Indicates the character set that the XML (see page 26144) document uses to encode characters.

Pascal

```
property Encoding: DOMString;
```

C++

```
__property DOMString Encoding;
```

Description

Encoding is the encoding name that appears in the document type declaration of the XML (see page 26144) document. It is a string such as 'UTF-8', 'UTF-16', or 'ISO-10646-UCS-2' that indicates the character set used in the document.

See Also

FileName (see page 26138)

DOMDocument (see page 26136)

XML (see page 26144)

195.1.1.5.13 TXMLDocument.FileName Property

Specifies the file name for the XML (see page 26144) document associated with this TXMLDocument (see page 26097) object.

Pascal

```
property FileName: DOMString;
```

C++

```
__property DOMString FileName;
```

Description

Set FileName to specify the XML (see page 26144) document you want to examine or manipulate using the TXMLDocument (see page 26097) object. Once you have set FileName to the name of an XML (see page 26144) document, you can activate the XML (see page 26144) document and examine or manipulate it using the DocumentElement (see page 26135) property.

When you set the value of FileName, the XML (see page 26144) property automatically changes (at runtime) to represent the contents of that file.

Note: Setting the value of FileName automatically sets the Active (see page 26133) property to false.

After activating the XML (see page 26144) document, any changes you make to the document are not saved back to disk until you call the SaveToFile (see page 26125) method.

Tip: If the underlying DOM implementation supports it, you can use FileName to specify an URL rather than a file on disk.

See Also

XML (see page 26144)

LoadFromFile (see page 26121)

SaveToFile (see page 26125)

DocumentElement (see page 26135)

Using TXMLDocument

Using Code That the XML Data Binding Wizard Generates

195.1.1.5.14 TXMLDocument.Modified Property

Indicates whether the XML (see page 26144) document has been modified.

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

Read Modified to determine whether the application has made any changes to the nodes in the XML (see page 26144) document. When Modified is true, the value of the XML (see page 26144) property may differ from the contents of the file specified by FileName (see page 26138). You can call the SaveToFile (see page 26125) method to update the XML (see page 26144) document file to reflect the changes.

Note: Modfied changes to false when you call the Refresh (see page 26123) or SaveToFile (see page 26125) method.

See Also

BeforeNodeChange (see page 26146)

AfterNodeChange (see page 26145)

Refresh (see page 26123)

SaveToFile (see page 26125)

XML (see page 26144)

FileName (see page 26138)

195.1.1.5.15 TXMLDocument.Node Property

Provides access to the document node for the XML (see page 26144) document.

Pascal

```
property Node: IXMLNode;
```

C++

```
__property IXMLNode Node;
```

Description

Read Node to access the document node for the XML (see page 26144) document. The document node represents information about the entire document. It may have several child nodes, which represent comments, processing instructions, and which include the DocumentElement (see page 26135) node that acts as the root of the data hierarchy.

See Also

DocumentElement (see page 26135)

ChildNodes (see page 26135)

195.1.1.5.16 TXMLDocument.NodeIndentStr Property

Indicates the string that is inserted before nested nodes in the formatted XML (see page 26144) text.

Pascal

```
property NodeIndentStr: DOMString;
```

C++

```
__property DOMString NodeIndentStr;
```

Description

Use NodeIndentStr to customize how TXMLDocument (see page 26097) formats nodes that are added. When the Options (see page 26141) property includes doNodeAutoIndent, this string is inserted in the generated XML (see page 26144) before any newly added node, once for each nesting level.

If you do not assign a value to NodeIndentStr, TXMLDocument (see page 26097) inserts two spaces to represent each nesting level.

See Also

Options (see page 26141)

NestingLevel (see page 26180)

FormatXMLData (see page 26240)

XML (see page 26144)

195.1.1.5.17 TXMLDocument.NSPrefixBase Property

Specifies the base string for automatically-generated namespace prefixes.

Pascal

```
property NSPrefixBase: DOMString;
```

C++

```
__property DOMString NSPrefixBase;
```

Description

The AddChild (see page 26107) method for each node allows the caller to specify that the new node should use an automatically-generated namespace prefix. When this option is specified, the node uses NSPrefixBase to generate a unique string to act as a namespace prefix. This unique string consists of NSPrefixBase followed by a number that ensures uniqueness. Thus, for example, if NSPrefixBase is 'NS', automatically-generated namespace prefixes have the form 'NS1', 'NS2', and so on.

The generated namespace prefixes are only unique within a single document. If two document components have the same value for NSPrefixBase, they will generate the same namespace prefixes.

See Also

AddChild (see page 26414)

GeneratePrefix (see page 26112)

195.1.1.5.18 TXMLDocument.Options Property

Influences various aspects of the document's behavior.

Pascal

```
property Options: TXMLDocOptions;
```

C++

```
__property TXMLDocOptions Options;
```

Description

Set Options to configure how you want the document to respond when the application tries to read nodes or attributes that do not exist, when it creates a new namespace when adding a node, when it generates XML (see page 26144) text from the nodes in the document, or whether it saves its value automatically when you close the document. Options can include any of the following values:

Value	Meaning
doNodeAutoCreate	If the application tries to read a node by name, using the Nodes property of an IXMLNodeList interface, and that node does not exist, then the application creates a new node using the specified name.
doNodeAutoIndent	When formatting the XML (see page 26144) text from the parsed set of nodes, child nodes are automatically indented from their parent nodes.
doAttrNull	When reading the value of an attribute that does not exist, the value is given as a Null Variant (as opposed to a value of an empty string).
doAutoPrefix	When generating the name of a new node, the value is automatically given a namespace prefix if a namespace URI is supplied.
doNamespaceDecl	When the AddChild (see page 26107) method must create a new namespace for the node, it also adds a namespace attribute to declare that new namespace. If doNamespaceDecl is not included, AddChild (see page 26107) creates the new namespace but does not add an attribute to declare it.

doAutoSave	When you close the document (set Active (see page 26133) to false), any changes are automatically saved back to the file specified by FileName (see page 26138) or to the XML (see page 26144) property.

See Also

Nodes (see page 26456)

Attributes (see page 26430)

AddChild (see page 26414)

Active (see page 26133)

FileName (see page 26138)

XML (see page 26144)

195.1.1.5.19 TXMLDocument.ParseOptions Property

Specifies implementation-specific options that are followed when parsing the XML (see page 26144) document.

Pascal

```
property ParseOptions: TParseOptions;
```

C++

```
__property TParseOptions ParseOptions;
```

Description

Set ParseOptions to influence the way the DOM implementation parses the XML (see page 26144) document. Different DOM implementations support different options. If you attempt to use an option that is not supported by your DOM implementation, TXMLDocument (see page 26097) raises an EXMLDocError exception.

See Also

domimplementation (see page 26137)

EXMLDocError (see page 26381)

Using TXMLDocument

195.1.1.5.20 TXMLDocument.PrefixID Property

Pascal

```
property PrefixID: Integer;
```

C++

```
__property int PrefixID;
```

Description

This is PrefixID, a member of class TXMLDocument.

195.1.1.5.21 TXMLDocument.SchemaRef Property

Returns the name of the schema associated with this XML (see page 26144) document.

Pascal

```
property SchemaRef: DOMString;
```

C++

```
__property DOMString SchemaRef;
```

Description

Read SchemaRef to determine the schema associated with this XML (see page 26144) document. If the document does not have an attribute that identifies an associated schema, the value of SchemaRef is an empty string.

See Also

DOMDocument (see page 26136)

195.1.1.5.22 TXMLDocument.StandAlone Property

Indicates whether the XML (see page 26144) document includes any external markup declarations.

Pascal

```
property StandAlone: DOMString;
```

C++

```
__property DOMString StandAlone;
```

Description

StandAlone is the value of the stand-alone attribute on the document type declaration of the XML (see page 26144) document. StandAlone has a value of 'yes' or 'no'.

When StandAlone is 'yes', the document contains no external markup declarations. It may, however, still contain references to external entities, as long as they are not declarations.

When StandAlone is 'no', the document may or may not contain external markup declarations.

See Also

DOMDocument (see page 26136)

195.1.1.5.23 TXMLDocument.Version Property

Indicates the XML (see page 26144) version that the document uses.

Pascal

```
property Version: DOMString;
```

C++

```
__property DOMString Version;
```

Description

Version indicates the XML (see page 26144) version number that the document uses. An XML (see page 26144) version number is a string of the form 'n.m', where n is the major version number and m is the minor version number.

See Also

XML (see page 26144)

195.1.1.5.24 TXMLDocument.XML Property

Specifies the content of the XML document associated with this TXMLDocument (see page 26097) object.

Pascal

```
property XML: TStrings;
```

C++

```
__property TStrings XML;
```

Description

Read XML to obtain the contents of the associated XML document. Set XML to specify the XML document you want to examine or manipulate. Once XML is set, you can activate the XML document and examine or manipulate it using the DocumentElement (see page 26135) property.

XML is a TStrings object. This means you can supply a value using the Strings editor at design time, or manipulate the XML line by line at runtime. To read or write the XML in the document as a single string, use the LoadFromXML (see page 26122) or SaveToXML (see page 26126) method.

See Also

FileName (see page 26138)

LoadFromXML (see page 26122)

SaveToXML (see page 26126)

DocumentElement (see page 26135)

Using TXMLDocument

Using Code That the XML Data Binding Wizard Generates

195.1.1.6 TXMLDocument Events

The events of the TXMLDocument class are listed here.

Published Events

	Name	Description
	AfterClose (see page 26144)	Occurs after the XML (see page 26144) document is closed.
	AfterNodeChange (see page 26145)	Occurs after a change is made to one of the nodes in the document.
	AfterOpen (see page 26145)	Occurs immediately after the XML (see page 26144) document is opened.
	BeforeClose (see page 26146)	Occurs immediately before the XML (see page 26144) document is closed.
	BeforeNodeChange (see page 26146)	Occurs before a change is made to one of the nodes in the document.
	BeforeOpen (see page 26147)	Occurs immediately before the XML (see page 26144) document is opened.
	OnAsyncLoad (see page 26147)	Occurs at various stages during the asynchronous parsing of an XML (see page 26144) document.

195.1.1.6.1 TXMLDocument.AfterClose Event

Occurs after the XML (see page 26144) document is closed.

Pascal

```
property AfterClose: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterClose;
```

Description

Write an AfterClose event handler to take specific action after the XML (see page 26144) document interfaces are released. For example, you can use an AfterClose event to update your user interface to indicate that the XML (see page 26144) document is no longer available.

AfterClose is the last event to occur when the Active (see page 26133) property is set to false.

See Also

BeforeClose (see page 26146)

Active (see page 26133)

195.1.1.6.2 TXMLDocument.AfterNodeChange Event

Occurs after a change is made to one of the nodes in the document.

Pascal

```
property AfterNodeChange: TNodeChangeEvent;
```

C++

```
__property TNodeChangeEvent AfterNodeChange;
```

Description

Write an AfterNodeChange event handler to take specific action after a change is made to a node in the XML (see page 26144) document. When an application makes a change to one of the nodes, the following steps occur:

1. The Modified (see page 26139) property is set to true.
2. A BeforeNodeChange (see page 26146) event occurs.
3. The change is made to the node.
4. An AfterNodeChange event occurs.

See Also

BeforeNodeChange (see page 26146)

Modified (see page 26139)

ChildNodes (see page 26135)

195.1.1.6.3 TXMLDocument.AfterOpen Event

Occurs immediately after the XML (see page 26144) document is opened.

Pascal

```
property AfterOpen: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterOpen;
```

Description

Write an AfterOpen event handler to take specific action once the XML (see page 26144) document interfaces become

available. For example, you can use an AfterOpen event to update your user interface to reflect the structure of the XML (see page 26144) document once it is opened.

AfterOpen is the last event to occur when the Active (see page 26133) property is set to true.

See Also

BeforeOpen (see page 26147)

Active (see page 26133)

195.1.1.6.4 TXMLDocument.BeforeClose Event

Occurs immediately before the XML (see page 26144) document is closed.

Pascal

```
property BeforeClose: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeClose;
```

Description

Write a BeforeClose event handler to take specific action before the XML (see page 26144) document interfaces are released. For example, you can use an BeforeClose event to cache information about the XML (see page 26144) document before it becomes unavailable.

BeforeClose is the first event to occur when the Active (see page 26133) property is set to false.

See Also

AfterClose (see page 26144)

Active (see page 26133)

195.1.1.6.5 TXMLDocument.BeforeNodeChange Event

Occurs before a change is made to one of the nodes in the document.

Pascal

```
property BeforeNodeChange: TNodeChangeEvent;
```

C++

```
__property TNodeChangeEvent BeforeNodeChange;
```

Description

Write a BeforeNodeChange event handler to take specific action before a change is made to a node in the XML (see page 26144) document. When an application makes a change to one of the nodes, the following steps occur:

1. The Modified (see page 26139) property is set to true.
2. A BeforeNodeChange event occurs.
3. The change is made to the node.
4. An BeforeNodeChange event occurs.

See Also

AfterNodeChange (see page 26145)

Modified (see page 26139)

ChildNodes (see page 26135)

195.1.1.6.6 TXMLDocument.BeforeOpen Event

Occurs immediately before the XML (see page 26144) document is opened.

Pascal

```
property BeforeOpen: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeOpen;
```

Description

Write a BeforeOpen event handler to take specific action before the XML (see page 26144) document interfaces become available. For example, you can use a BeforeOpen event to provide feedback to the user that the XML (see page 26144) document is being parsed (which can be cleared in an AfterOpen (see page 26145) event handler).

BeforeOpen is the first event to occur when the Active (see page 26133) property is set to true.

See Also

AfterOpen (see page 26145)

Active (see page 26133)

195.1.1.6.7 TXMLDocument.OnAsyncLoad Event

Occurs at various stages during the asynchronous parsing of an XML (see page 26144) document.

Pascal

```
property OnAsyncLoad: TAsyncEventHandler;
```

C++

```
__property TAsyncEventHandler OnAsyncLoad;
```

Description

Write a OnAsyncLoad event handler to respond appropriately during the asynchronous parsing of an XML (see page 26144) document. For example, OnAsyncLoad can update a progress indicator, or initiate operations that can't be performed until the document is fully parsed. OnAsyncLoad occurs every time the AsyncLoadState (see page 26134) property changes to reflect a new stage in the parsing process.

See Also

AsyncLoadState (see page 26134)

XML (see page 26144)

195.1.2 TXMLNode Class

TXMLNode represents a node in an XML document.

Class Hierarchy

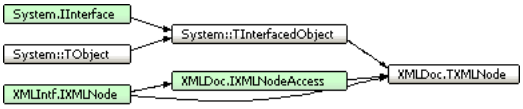

Pascal

```
TXMLNode = class(TInterfacedObject, IXMLNode, IXMLNodeAccess);
```

C++

```
class TXMLNode : public TInterfacedObject, public IXMLNode, public IXMLNodeAccess;
```

File

XMLDoc

Description

The XML Data Binding Wizard uses TXMLNode as a base class for the classes it generates to represent nodes that have child nodes. For example, if the XML document includes the following:

```
<Address country="US">
  <Name>
    <First> John </First>
    <MI> Q. </MI>
    <Last> Public </Last>
  </Name>
  <Street> 123 Easy Street </Street>
  <City> Anytown </City>
  <State> CA </State>
</Address>
```

then the XML Data Binding Wizard generates TXMLNode descendants for the elements Address and Name. Child nodes and node attributes appear as properties of the TXMLNode descendant.

Note: The XML Data Binding Wizard also generates code that can create TXMLNode objects for attributes as well as element nodes. However, there are no special TXMLNode descendants for attribute nodes. Rather, each attribute is represented directly by TXMLNode.

TXMLNode implements the IXMLNode interface. Each TXMLNode descendant implements a class-specific interface that descends from IXMLNode. Typically, applications do not work directly with TXMLNode. Instead, they use the IXMLNode interface or the IXMLNode descendant that is implemented by a TXMLNode descendant.

See Also

TXMLDocument (see page 26097)

TXMLNodeCollection (see page 26196)

IXMLNode (see page 26411)

TXMLNodeList (see page 26211)

195.1.2.1 TXMLNode Members

The following tables list the members exposed by TXMLNode.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 26155)	Decrements the reference count for this node.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.

	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 26152)	Instantiates a Create object.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
	Attributes (see page 26430)	Represents the values of this node's attributes.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.

	NodeValue (see page 26436)	Specifies the value of this node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

195.1.2.2 TXMLNode.Create Constructor

Instantiates a Create object.

Pascal

```
constructor Create(const ADOMNode: IDOMNode; const AParentNode: TXMLNode; const OwnerDoc:
TXMLDocument);
```

C++

```
__fastcall TXMLNode(const IDOMNode ADOMNode, const TXMLNode AParentNode, const TXMLDocument
OwnerDoc);
```

Description

Create instantiates a new Create object to wrap an existing DOM node implementation. Using the first syntax,

ADOMNode is the interface for the DOM implementation of the node.

AParentNode is the implementation of the parent node of this node.

OwnerDoc represents the XML document in which the new node occurs.

Using the second syntax,

HostNode (see page 26193) is another Create object that represents the host node of this node. In this case, this node is a "hosted" node, which means it shares the same underlying parent and DOM node as the host node, but represents a different view of the host node. For example, a node that contains some child nodes that are all of the same type creates a hosted node to represent only the set of repeating child nodes.

Typically, applications do not directly call the Create constructor. Instead, new nodes are created automatically for an XML document as necessary when the application reads their interfaces, starting with the DocumentElement property of the TXMLDocument (see page 26097) object. To add new nodes to a document, applications call the parent node's AddChild (see page 26156) method.

See Also

OwnerDocument (see page 26193)

ParentNode (see page 26194)

DocumentElement (see page 26135)

AddChild (see page 26156)

195.1.2.3 TXMLNode.CreateHosted Constructor

Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.

Pascal

```
constructor CreateHosted(HostNode: TXMLNode);
```

C++

```
__fastcall TXMLNode(TXMLNode HostNode);
```

Description

CreateHosted instantiates a new TXMLNode (see page 26147) object to represent a specific view on an existing TXMLNode (see page 26147) object.

HostNode (see page 26193) is another TXMLNode (see page 26147) object that represents the host node of this node. The new node becomes a "hosted" node, which means it shares the same underlying parent and DOM node as the host node, but represents a different view of the host node. For example, a node that contains some child nodes that are all of the same type creates a hosted node to represent only the set of repeating child nodes.

Typically, applications do not directly call CreateHosted to create a hosted node. Instead, this method is called in code generated by the XML Data Binding Wizard, or by the CreateCollection (see page 26163) method of the host node.

HostNode (see page 26193) (or, if HostNode (see page 26193) is hosted, the host node of HostNode (see page 26193)) becomes the value of the new node's HostNode (see page 26193) property, and the new node is added to the HostedNodes (see page 26192) property list of the new host node.

See Also

HostedNodes (see page 26192)

HostNode (see page 26193)

OnHostAttrNotify (see page 26194)

OnHostChildNotify (see page 26195)

CreateCollection (see page 26163)

195.1.2.4 TXMLNode.Destroy Destructor

Frees the memory associated with the TXMLNode (see page 26147) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLNode();
```

Description

Do not explicitly destroy TXMLNode (see page 26147) objects. As interfaced objects, XML nodes are destroyed automatically when the reference count of their IXMLNode interface drops to zero.

See Also

RefCount (see page 22281)

195.1.2.5 TXMLNode Methods

The methods of the TXMLNode class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 26155)	Increments the reference count for this node.

	_Release (see page 26155)	Decrements the reference count for this node.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26159)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.
	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.

	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 26185)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.
	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.

195.1.2.5.1 TXMLNode._AddRef Method

Increments the reference count for this node.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef implements the IInterface or IUnknown method that is called when an application obtains a reference to this node's interface pointer. __AddRef increments the RefCount property. In addition, to ensure that the document object is alive, it also increments the reference count for the owning document.

When the caller is through with the interface, it calls _Release (see page 26155). This allows the node to free itself when it is no longer used (when the reference count goes to zero).

See Also

_Release (see page 26155)

OwnerDocument (see page 26193)

RefCount (see page 22281)

195.1.2.5.2 TXMLNode._Release Method

Decrements the reference count for this node.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release implements the IInterface or IUnknown method that is called when an application releases a reference to this node's interface pointer. __Release decrements the RefCount property. In addition, it also decrements the reference count for the owning document, which it incremented when the _AddRef (see page 26155) method was called.

When __Release causes the node's reference count to drop to zero, __Release causes the node to free itself.

See Also

_AddRef (see page 26155)

OwnerDocument (see page 26193)

RefCount (see page 22281)

195.1.2.5.3 AddChild Method

This is the overview for the AddChild method overload.

Overload List

	Name	Description
	TXMLNode.AddChild (DOMString, DOMString, Boolean, Integer) (see page 26156)	Adds a new child node to this node.
	TXMLNode.AddChild (DOMString, DOMString, TXMLNodeClass, Integer) (see page 26156)	Adds a new child node to this node.
	TXMLNode.AddChild (DOMString, Integer) (see page 26156)	Adds a new child node to this node.

195.1.2.5.3.1 TXMLNode.AddChild Method (DOMString, DOMString, Boolean, Integer)

Adds a new child node to this node.

Pascal

```
function AddChild(const TagName: DOMString; Index: Integer = -1): IXMLNode; overload;
function AddChild(const TagName: DOMString; const NamespaceURI: DOMString; GenPrefix: Boolean
= False; Index: Integer = -1): IXMLNode; overload;
function AddChild(const TagName: DOMString; const NamespaceURI: DOMString; NodeClass:
TXMLNodeClass; Index: Integer = -1): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode AddChild(const DOMString TagName, int Index = -1);
__fastcall IXMLNode AddChild(const DOMString TagName, const DOMString NamespaceURI, Boolean
GenPrefix = False, int Index = -1);
__fastcall IXMLNode AddChild(const DOMString TagName, const DOMString NamespaceURI,
TXMLNodeClass NodeClass, int Index = -1);
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected AddChild method. Only the first two versions of AddChild are available on the IXMLNode interface. The third syntax is used by TXMLNode (see page 26147)

descendants that the XML data binding wizard creates.

AddChild creates a new element node as the child of this node.

TagName provides the tag name of the newly created node.

Index indicates the position of the child node in this node's list of children, where 0 is the first position, 1 is the second position, and so on. If Index is –1, the new node is added to the end.

NamespaceURI identifies the namespace that includes the new node's definition. If NamespaceURI is omitted (the first syntax), the namespace of the new node is deduced from the namespace prefix of TagName. If TagName has no namespace prefix and there is no NamespaceURI parameter, the namespace is the same as this node's NamespaceURI property.

GenPrefix controls whether AddChild generates a namespace prefix for the namespace URI when the NamespaceURI parameter identifies a URI that is not already declared in the document. Note that setting GenPrefix to true does not cause AddChild to add a declaration of the new namespace prefix unless the document's Options property includes doNamespaceDecl.

NodeClass identifies the implementation class to use for the new child node. It must be TXMLNode (see page 26147) or one of its descendants.

AddChild returns the interface for the newly created child node.

Tip: AddChild only adds element nodes. To add other types of child nodes, use the XML document's CreateNode method instead, and then add the resulting node to the ChildNodes (see page 26190) property array.

See Also

ChildNodes (see page 26190)

AddChild (see page 26414)

CreateNode (see page 26109)

Options (see page 26141)

CreateElement (see page 26109)

NSPrefixBase (see page 26141)

DeclareNamespace (see page 26164)

195.1.2.5.4 TXMLNode.AddHostedNode Method

Adds a node to the HostedNodes (see page 26192) property list.

Pascal

```
procedure AddHostedNode(Node: TXMLNode);
```

C++

```
__fastcall AddHostedNode(TXMLNode Node);
```

Description

AddHostedNode is used internally to add nodes to the HostedNodes (see page 26192) array.

Node is the node to add to the array.

See Also

HostedNodes (see page 26192)

RemoveHostedNode (see page 26183)

195.1.2.5.5 TXMLNode.AttributeListNotify Method

Responds to notifications of changes in the list of attributes.

Pascal

```
procedure AttributeListNotify(Operation: TNodeListOperation; var Node: IXMLNode; const
IndexOrName: OleVariant; BeforeOperation: Boolean);
```

C++

```
__fastcall AttributeListNotify(TNodeListOperation Operation, IXMLNode Node, const OleVariant
IndexOrName, Boolean BeforeOperation);
```

Description

Applications can't call this protected method. The class that implements the AttributeNodes (see page 26189) property calls AttributeListNotify when attribute nodes are inserted or removed, or when a new attribute node needs to be created. AttributeListNotify generates a BeforeNodeChange or AfterNodeChange event on the document object, and, if the change has not yet occurred, updates the underlying DOM implementation to reflect the change.

Operation indicates what type of change occurred.

Node is the interface for the attribute node that was added or removed. If a new attribute node needs to be created, Node returns the new node.

IndexOrName is the name of the attribute when Operation is nlCreateNode.

BeforeOperation indicates whether the call occurs before the attribute node is added or removed.

See Also

AttributeNodes (see page 26189)

TNodeListNotification (see page 26248)

BeforeNodeChange (see page 26146)

AfterNodeChange (see page 26145)

195.1.2.5.6 TXMLNode.CheckNotHosted Method

Raises an EXMLDocError exception if this is a hosted node.

Pascal

```
procedure CheckNotHosted;
```

C++

```
__fastcall CheckNotHosted();
```

Description

Applications can't call this protected method. It is used internally in methods that can only be used by the main node that corresponds to an element. If the node represents a hosted node (that is, if the node is used to represent a subset of the main, or host, node), then CheckNotHosted raises an EXMLDocError exception.

For example, the CreateCollection (see page 26163) method calls CheckNotHosted, because only the main node for an element can host collection nodes.

See Also

HostNode (see page 26193)

HostedNodes (see page 26192)

EXMLDocError (see page 26381)

CreateCollection (see page 26163)

195.1.2.5.7 TXMLNode.CheckReadOnly Method

Raises an EXMLDocError exception if this is a read-only node.

Pascal

```
procedure CheckReadOnly;
```

C++

```
__fastcall CheckReadOnly();
```

Description

Applications can't call this protected method. It is used internally in methods that change the value, children, or attributes of the node, to check whether the change is permitted, and if not, raise an exception.

See Also

GetReadOnly (see page 26177)

SetReadOnly (see page 26187)

ReadOnly (see page 26438)

EXMLDocError (see page 26381)

195.1.2.5.8 TXMLNode.CheckTextNode Method

Generates a DOM text node implementation if it does not yet exist.

Pascal

```
procedure CheckTextNode;
```

C++

```
__fastcall CheckTextNode();
```

Description

Applications can't call this protected method. It is used internally to generate a DOM node implementation for text values.

If the XML node is an element that has child nodes (other than a single DOM text node), CheckTextNode raises an exception.

195.1.2.5.9 TXMLNode.ChildListNotify Method

Responds to notifications of changes in the list of child nodes.

Pascal

```
procedure ChildListNotify(Operation: TNodeListOperation; var Node: IXMLNode; const
IndexOrName: OleVariant; BeforeOperation: Boolean); virtual;
```

C++

```
virtual __fastcall ChildListNotify(TNodeListOperation Operation, IXMLNode Node, const
OleVariant IndexOrName, Boolean BeforeOperation);
```

Description

Applications can't call this protected method. The class that implements the ChildNodes (see page 26190) property calls ChildListNotify when child nodes are inserted or removed, or when a node needs to be created. ChildListNotify generates a BeforeNodeChange or AfterNodeChange event on the document object, and, if the change has not yet occurred, updates the underlying DOM implementation to reflect the change.

Operation indicates what type of change is occurring.

Node is the interface for the child node about which this method is called. When nodes are added or removed, this is the node that is added or removed. When a node needs to be created, this parameter returns the newly created node.

IndexOrName is the index of the node that is added or removed when Operation is nlInsert or nlRemove. When Operation is nlCreateNode, IndexOrName is the name of the child node to create.

BeforeOperation indicates whether the call occurs before the child node is added or removed.

See Also

ChildNodes (see page 26190)

AddChild (see page 26156)

TNodeListNotification (see page 26248)

BeforeNodeChange (see page 26146)

AfterNodeChange (see page 26145)

195.1.2.5.10 TXMLNode.ClearDocumentRef Method

Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).

Pascal

```
procedure ClearDocumentRef;
```

C++

```
__fastcall ClearDocumentRef();
```

Description

Applications can't call this protected method. The document that contains this node calls ClearDocumentRef when it is freed so that the node does not maintain a reference to a non-existent document.

In addition to setting the OwnerDocument (see page 26193) property to nil (Delphi) or NULL (C++), ClearDocumentRef propagates this call to all of this nodes child nodes and attribute nodes.

See Also

OwnerDocument (see page 26193)

ChildNodes (see page 26190)

AttributeNodes (see page 26189)

195.1.2.5.11 TXMLNode.CloneNode Method

Creates a copy of this node and returns its interface.

Pascal

```
function CloneNode(Deep: Boolean): IXMLNode;
```

C++

```
__fastcall IXMLNode CloneNode(Boolean Deep);
```

Description

Applications can't call the protected CloneNode method. To create a copy of this node, use the CloneNode method of this node's IXMLNode interface.

CloneNode returns a copy of this node that has no parent. Any attributes of this node are copied as well, including attribute nodes that represent default values.

Deep indicates whether the child nodes of this node should be cloned as well. For example, if an element node contains any text, CloneNode only copies then text when deep is true, because in the underlying DOM implementation, the text is contained in a child node.

See Also

ChildNodes (see page 26190)

AddChild (see page 26156)

ParentNode (see page 26194)

CloneNode (see page 26415)

CloneNode (see page 26309)

195.1.2.5.12 TXMLNode.CreateAttributeList Method

Creates the object that implements the AttributeNodes (see page 26189) property.

Pascal

```
function CreateAttributeList: IXMLNodeList; dynamic;
```

C++

```
__fastcall IXMLNodeList CreateAttributeList();
```

Description

Applications can't call the protected CreateAttributeList method. This method is used internally the first time an application reads the AttributeNodes (see page 26189) property.

CreateAttributeList creates an implementation class for the AttributeNodes (see page 26189) property, telling it to call the AttributeListNotify (see page 26158) method when it changes the list of attributes for this node, and filling it with nodes based on the current attributes of this node.

See Also

AttributeNodes (see page 26189)

AttributeListNotify (see page 26158)

195.1.2.5.13 TXMLNode.CreateAttributeNode Method

Creates a new TXMLNode (see page 26147) object for an attribute of this node.

Pascal

```
function CreateAttributeNode(const ADOMNode: IDOMNode): IXMLNode; virtual;
```

C++

```
virtual __fastcall IXMLNode CreateAttributeNode(const IDOMNode ADOMNode);
```

Description

Applications can't call the protected CreateAttributeNode method. TXMLNode (see page 26147) uses this method internally to generate the nodes that populate the AttributeNodes (see page 26189) property.

CreateAttributeNode creates a new TXMLNode (see page 26147) object for the specified IDOMNode interface and returns its interface.

ADOMNode is the interface for the DOM implementation's version of the attribute node.

See Also

AttributeNodes (see page 26189)

195.1.2.5.14 TXMLNode.CreateChildList Method

Creates the object that implements the ChildNodes (see page 26190) property.

Pascal

```
function CreateChildList: IXMLNodeList; dynamic;
```

C++

```
__fastcall IXMLNodeList CreateChildList();
```

Description

Applications can't call the protected CreateChildList method. This method is used internally the first time an application reads the ChildNodes (see page 26190) property.

CreateChildList creates an implementation class for the ChildNodes (see page 26190) property, telling it to call the ChildListNotify (see page 26159) method when it changes the list of attributes for this node, and filling it with nodes that are based on the current children of this node.

See Also

ChildNodes (see page 26190)

ChildListNotify (see page 26159)

195.1.2.5.15 TXMLNode.CreateChildNode Method

Creates a new TXMLNode (see page 26147) object for a child node of this node.

Pascal

```
function CreateChildNode(const ADOMNode: IDOMNode): IXMLNode; virtual;
```

C++

```
virtual __fastcall IXMLNode CreateChildNode(const IDOMNode ADOMNode);
```

Description

Applications can't call the protected CreateChildNode method. This method is used internally to create the implementation object for a new child node.

CreateChildNode creates a new TXMLNode (see page 26147) descendant for the specified IDOMNode interface, with this node as the parent. If the node name of the DOM node matches one of the names listed by the ChildNodeClasses (see page 26190) property, the corresponding descendant of TXMLNode (see page 26147) becomes the implementation class of the

new node. Otherwise, the new node is an instance of TXMLNode (see page 26147).

ADOMNode is the interface for the DOM implementation's version of the node.

CreateChildNode returns the interface for the newly created child node.

See Also

Create (see page 26152)

ChildNodeClasses (see page 26190)

CreateDOMNode (see page 26238)

195.1.2.5.16 TXMLNode.CreateCollection Method

Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.

Pascal

```
function CreateCollection(const CollectionClass: TXMLNodeCollectionClass; const ItemInterface:
TGuid; const ItemTag: DOMString; ItemNS: DOMString = ''): TXMLNodeCollection;
```

C++

```
__fastcall TXMLNodeCollection CreateCollection(const TXMLNodeCollectionClass CollectionClass,
const TGuid ItemInterface, const DOMString ItemTag, DOMString ItemNS = '');
```

Description

Applications can't call the protected CreateCollection method. It is used internally to generate the collections that are listed in the HostedNodes (see page 26192) property.

CollectionClass specifies the class (a TXMLNodeCollection (see page 26196) descendant) that implements the collection of child nodes.

ItemInterface specifies the interface for each child node in the collection.

ItemTag specifies the tag name for each child node in the collection.

ItemNS specifies the namespace in which ItemTag is defined.

CreateCollection returns the new hosted collection node. This new node has the same parent node, DOM interface, attribute nodes, and child nodes as this node, but it also maintains a list of only those child nodes with the specified tag and interface. The new collection node lists this node as the value of its HostNode (see page 26193) property.

Note: CreateCollection raises an exception if this is a hosted node.

See Also

HostedNodes (see page 26192)

ChildNodes (see page 26190)

ParentNode (see page 26194)

DOMNode (see page 26192)

HostNode (see page 26193)

ItemInterface (see page 26209)

itemtag (see page 26210)

itemns (see page 26210)

195.1.2.5.17 TXMLNode.DeclareNamespace Method

Adds an attribute to the node that declares a specified namespace URI.

Pascal

```
procedure DeclareNamespace(const Prefix: DOMString; const URI: DOMString);
```

C++

```
__fastcall DeclareNamespace(const DOMString Prefix, const DOMString URI);
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected DeclareNamespace method. DeclareNamespace adds an attribute to this node that declares the specified namespace URI as the value of the specified prefix.

Prefix is the namespace prefix that appears in tag names that reference the namespace.

URI is the corresponding namespace URI.

Note: This method is called automatically for newly created nodes when the document's Options property includes doNamespaceDecl.

See Also

Options (see page 26141)

AttributeNodes (see page 26189)

FindNamespaceDecl (see page 26165)

DeclareNamespace (see page 26416)

195.1.2.5.18 TXMLNode.DOMElement Method

Returns the IDOMElement interface for this node if it exists.

Pascal

```
function DOMElement: IDOMElement;
```

C++

```
__fastcall IDOMElement DOMElement();
```

Description

Application's can't call the protected DOMElement method. TXMLNode (see page 26147) uses this method internally to obtain the DOM element interface, which it uses for working with attributes.

If the TXMLNode (see page 26147) instance does not represent an element node, DOMElement raises an EXMLDocError exception.

See Also

AttributeNodes (see page 26189)

GetAttribute (see page 26166)

SetAttribute (see page 26183)

195.1.2.5.19 TXMLNode.DoNodeChange Method

Generates a BeforeNodeChange or AfterNodeChange event on this node's document.

Pascal

```
procedure DoNodeChange(ChangeType: TNodeChange; BeforeOperation: Boolean); virtual;
```

C++

```
virtual __fastcall DoNodeChange(TNodeChange ChangeType, Boolean BeforeOperation);
```

Description

Applications can't call the protected DoNodeChange method. It is called internally before and after every change that is made to the node. DoNodeChange checks whether the node's OwnerDocument (see page 26193) property is set, and if so, generates a BeforeNodeChange or AfterNodeChange event on that document, as appropriate.

ChangeType indicates the type of change that is occurring to the node.

BeforeOperation is true if the change is about to occur, false if the change has just occurred.

See Also

AfterNodeChange (see page 26145)

BeforeNodeChange (see page 26146)

OwnerDocument (see page 26193)

195.1.2.5.20 TXMLNode.FindHostedNode Method

Returns the IXMLNode interface for a hosted node given its implementation class.

Pascal

```
function FindHostedNode(const NodeClass: TXMLNodeClass): IXMLNode;
```

C++

```
__fastcall IXMLNode FindHostedNode(const TXMLNodeClass NodeClass);
```

Description

Applications can't call the protected FindHostedNode method. Descendant classes call this method to locate a hosted node of a particular class type.

FindHostedNode searches the nodes in the HostedNodes (see page 26192) list and returns the first one it finds that is an instance of descendant of NodeClass. If this node does not have a hosted node of the specified class, FindHostedNode returns nil (Delphi) or NULL (C++).

See Also

HostedNodes (see page 26192)

CreateCollection (see page 26163)

195.1.2.5.21 TXMLNode.FindNamespaceDecl Method

Returns the attribute node that declares a specified Namespace URI.

Pascal

```
function FindNamespaceDecl(const NamespaceURI: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode FindNamespaceDecl(const DOMString NamespaceURI);
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected FindNamespaceDecl method. FindNamespaceDecl searches the node hierarchy that contains this node for an attribute that specifies the mapping between a namespace prefix and the namespace URI specified by NamespaceURI. If it finds such a mapping, FindNamespaceDecl returns the interface for the Attribute node. If it does not find a mapping, FindNamespaceDecl returns nil (Delphi) or NULL (C++).

NamespaceURI is the namespace URI to locate.

See Also

FindNamespaceURI (see page 26166)

DeclareNamespace (see page 26164)

FindNamespaceDecl (see page 26416)

195.1.2.5.22 TXMLNode.FindNamespaceURI Method

Returns the namespace URI for a namespace prefix or qualified tag name.

Pascal

```
function FindNamespaceURI(const TagOrPrefix: DOMString): DOMString;
```

C++

```
__fastcall DOMString FindNamespaceURI(const DOMString TagOrPrefix);
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected FindNamespaceURI method. FindNamespaceDecl (see page 26165) searches the node hierarchy that contains this node for an attribute that specifies the mapping between the namespace prefix identified by TagOrPrefix and a namespace URI. If it finds such a mapping, FindNamespaceURI returns the namespace URI. If it does not find a mapping, FindNamespaceURI returns an empty string.

TagOrPrefix is the namespace prefix or a tag name that includes the namespace prefix.

See Also

GetNamespaceURI (see page 26172)

GetPrefix (see page 26176)

FindNamespaceDecl (see page 26165)

FindNamespaceURI (see page 26416)

195.1.2.5.23 TXMLNode.GetAttribute Method

Returns the value of one of this node's attributes.

Pascal

```
function GetAttribute(const AttrName: DOMString): OleVariant;
```

C++

```
__fastcall OleVariant GetAttribute(const DOMString AttrName);
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected GetAttribute method. GetAttribute returns the value of an attribute of this node.

AttrName is the name of the attribute.

GetAttribute returns the value of the specified attribute. Typically, this value is a string. If the node does not have an attribute with the name specified by AttrName, the value that GetAttribute returns is determined by the document's Options property. When Options include doAttrNull, GetAttribute returns a Null Variant for missing attributes. When Options does not include doAttrNull, GetAttribute returns an empty string as the value of missing attributes.

If this node is not an element node, GetAttribute raises an EXMLDocError exception.

See Also

SetAttribute (see page 26183)

AttributeNodes (see page 26189)

GetAttributeNS (see page 26167)

195.1.2.5.24 TXMLNode.GetAttributeNodes Method

Returns the value of the AttributeNodes (see page 26189) property.

Pascal

```
function GetAttributeNodes: IXMLNodeList;
```

C++

```
__fastcall IXMLNodeList GetAttributeNodes();
```

Description

GetAttributeNodes is the read implementation of the AttributeNodes (see page 26189) property on the IXMLNode interface for this node.

GetAttributeNodes generates the list that implements the AttributeNodes (see page 26189) property the first time an application reads the AttributeNodes (see page 26189) property.

See Also

GetAttribute (see page 26166)

AttributeNodes (see page 26189)

195.1.2.5.25 TXMLNode.GetAttributeNS Method

Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.

Pascal

```
function GetAttributeNS(const AttrName: DOMString; const NamespaceURI: DOMString): OleVariant;
```

C++

```
__fastcall OleVariant GetAttributeNS(const DOMString AttrName, const DOMString NamespaceURI);
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected GetAttributeNS method. GetAttributeNS returns the value of an attribute of this node.

AttrName is the name of the attribute.

NamespaceURI identifies the namespace in which the attribute resides.

GetAttributeNS returns the value of the specified attribute. If AttrName and NamespaceURI do not identify an attribute of this node, GetAttributeNS returns a Null Variant.

If this node is not an element node, GetAttributeNS raises an EXMLDocError exception.

See Also

GetAttribute (see page 26166)

Attributes (see page 26430)

GetNamespaceURI (see page 26172)

SetAttributeNS (see page 26184)

AttributeNodes (see page 26189)

195.1.2.5.26 TXMLNode.GetChildNodeClasses Method

Returns a list of the classes that can implement child nodes of this node.

Pascal

```
function GetChildNodeClasses: TNodeClassArray;
```

C++

```
__fastcall TNodeClassArray GetChildNodeClasses();
```

Description

GetChildNodeClasses is used internally to obtain a list of the classes that can be used to represent child nodes of this node.

If this node is not a hosted node, GetChildNodeClasses returns an array containing every class that has been registered using the RegisterChildNode (see page 26182) method. If this node is a hosted node, GetChildNodeClasses returns the ChildNodeClasses (see page 26190) property of the host node.

See Also

ChildNodeClasses (see page 26190)

RegisterChildNode (see page 26182)

HostNode (see page 26193)

195.1.2.5.27 TXMLNode.GetChildNodes Method

Returns the value of the ChildNodes (see page 26190) property.

Pascal

```
function GetChildNodes: IXMLNodeList;
```

C++

```
__fastcall IXMLNodeList GetChildNodes();
```

Description

GetChildNodes is the protected read implementation of the ChildNodes (see page 26190) property.

GetChildNodes generates the list that implements the ChildNodes (see page 26190) property the first time an application reads the ChildNodes (see page 26190) property.

See Also

GetChildValue (see page 26169)

ChildNodes (see page 26190)

195.1.2.5.28 TXMLNode.GetChildValue Method

Returns the value of a specified child node.

Pascal

```
function GetChildValue(const IndexOrName: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant GetChildValue(const OleVariant IndexOrName);
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected GetChildValue method. GetChildValue returns the value of the node.

IndexOrName identifies the desired child node by either its interface, or by a string that is the local name of the child node.

If IndexOrName does not identify a child of this node, GetChildValue nil (Delphi) or NULL (C++). Otherwise, the return value depends on the type of the child node, as indicated in the following table:

NodeType	Value
ntAttribute	The attribute value
ntElement	If the element contains only text, this is that text value. Otherwise, GetChildValue raises an exception.
ntText	The text
ntCData	The content of the CDATA section.
ntEntityRef	nil (Delphi) or NULL (C++)
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.
ntDocFragment	nil (Delphi) or NULL (C++)

Note: Attributes are not considered child nodes of a node. To get the values of this node's attributes, use the GetAttribute (see page 26166) method instead.

See Also

ChildNodes (see page 26190)

ParentNode (see page 26194)

GetAttribute (see page 26166)

GetNodeValue (see page 26174)

195.1.2.5.29 TXMLNode.GetCollection Method

Returns the value of the Collection (see page 26191) property.

Pascal

```
function GetCollection: IXMLNodeCollection;
```

C++

```
__fastcall IXMLNodeCollection GetCollection();
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected GetCollection method. GetCollection returns the value of the Collection (see page 26191) property. If this node is not a member of a collection (that is, if it does not represent one of a set of child nodes that are all the same type), then GetCollection returns nil (Delphi) or NULL (C++).

See Also

Collection (see page 26191)

195.1.2.5.30 TXMLNode.GetDOMNode Method

Returns the value of the DOMNode (see page 26192) property.

Pascal

```
function GetDOMNode: IDOMNode;
```

C++

```
__fastcall IDOMNode GetDOMNode();
```

Description

GetDOMNode is the protected read implementation of the DOMNode (see page 26192) property.

See Also

DOMNode (see page 26192)

195.1.2.5.31 TXMLNode.GetHasChildNodes Method

Indicates whether this node has any child nodes.

Pascal

```
function GetHasChildNodes: Boolean;
```

C++

```
__fastcall Boolean GetHasChildNodes();
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected GetHasChildNodes method. GetHasChildNodes returns true if the underlying DOM node has any child nodes, false if it doesn't.

Note that even if the underlying DOM node has a child node, there may not be a TXMLNode (see page 26147) object to represent that child. For example, text nodes do not, by default, have corresponding TXMLNode (see page 26147) wrappers. Rather, the text becomes the value of the parent element node.

See Also

ChildNodes (see page 26190)

DOMNode (see page 26192)

GetIsTextElement (see page 26171)

195.1.2.5.32 TXMLNode.GetHostedNodes Method

Returns a list an array of the hosted nodes that share this node's attribute and child node lists.

Pascal

```
function GetHostedNodes: TXMLNodeArray;
```

C++

```
__fastcall TXMLNodeArray GetHostedNodes();
```

Description

GetHostedNodes is the read implementation of the HostedNodes (see page 26192) property. It returns a list an array of all nodes that represent aspects of this node, sharing its attributes and list of child nodes.

See Also

HostedNodes (see page 26192)

195.1.2.5.33 TXMLNode.GetHostNode Method

Returns the value of the HostNode (see page 26193) property.

Pascal

```
function GetHostNode: TXMLNode;
```

C++

```
__fastcall TXMLNode GetHostNode();
```

Description

GetHostNode returns the node that uses this node to represent a particular aspect of it, sharing the same attributes and list of child nodes. If this is not a hosted node, GetHostNode returns nil (Delphi) or NULL (C++).

See Also

HostNode (see page 26193)

195.1.2.5.34 TXMLNode.GetIsTextElement Method

Indicates whether the node has a single text value.

Pascal

```
function GetIsTextElement: Boolean;
```

C++

```
__fastcall Boolean GetIsTextElement();
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected GetIsTextElement method. GetIsTextElement returns true if the node represents an XML fragment such as

```
<Title> Understanding XML </Title>
```

where the node has a tag name (in this case Title) and a single text value (in this case 'Understanding XML').

GetIsTextElement returns false if the node is any other type of node.

Note: In the underlying DOM implementation, nodes for which GetIsTextElement is true are element nodes that have a single

child node that is a text node. TXMLNode (see page 26147) flattens this out so that the child text node does not appear as a child node. Instead, the text can be accessed through the GetText (see page 26177) method.

See Also

GetText (see page 26177)

GetHasChildNodes (see page 26170)

195.1.2.5.35 TXMLNode.GetLocalName Method

Returns the name of the node without any namespace prefix.

Pascal

```
function GetLocalName: DOMString;
```

C++

```
__fastcall DOMString GetLocalName();
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected GetLocalName method. GetLocalName returns the local name of the underlying DOM node.

For element nodes, the local name is derived from the name that appears in the XML tag. For example, the local name for <xsi:Name First="John" Last="Doe"> is 'Name'.

For attribute nodes, the local name is derived from the string that appears to the left of the equals sign in the attribute declaration. For example the element <Name First=John Last=Doe> results in two attribute nodes, with the local names 'First' and 'Last'.

For any other node types, the local name is nil (Delphi) or NULL (C++).

Note: If both GetPrefix (see page 26176) and GetLocalName return a value for a node, the node name (returned by GetNodeName (see page 26173)) is the combination of these two values.

See Also

GetNodeName (see page 26173)

GetNodeType (see page 26174)

GetPrefix (see page 26176)

GetNamespaceURI (see page 26172)

LocalName (see page 26318)

195.1.2.5.36 TXMLNode.GetNamespaceURI Method

Returns the URI for the namespace used in the qualified node name.

Pascal

```
function GetNamespaceURI: DOMString;
```

C++

```
__fastcall DOMString GetNamespaceURI();
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected GetNamespaceURI method.

GetNamespaceURI returns the URI for the namespace that should be used to interpret the node's name.

Only element and attribute nodes can have a namespace URI value. For any other node type, GetNamespaceURI returns nil (Delphi) or NULL (C++).

The namespace URI is only available if it is explicitly assigned when the node is created. By default, TXMLNode (see page 26147) does not assign namespaces to the new nodes it creates.

See Also

GetNodeName (see page 26173)

GetNodeType (see page 26174)

GetPrefix (see page 26176)

GetLocalName (see page 26172)

CreateDOMNode (see page 26238)

NamespaceURI (see page 26318)

195.1.2.5.37 TXMLNode.GetNodeName Method

Returns the node name.

Pascal

```
function GetNodeName: DOMString;
```

C++

```
__fastcall DOMString GetNodeName();
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected GetNodeName method. GetNodeName returns the name of the underlying DOM node.

The node's name depends on the type of the node, as indicated in the following table:

NodeType	NodeName
ntAttribute	The attribute name
ntElement	The tag name
ntText	'#text'
ntCData	'#cdata-section'
ntEntityRef	The name of the entity reference.
ntEntity	The entity name
ntProcessingInstr	the target of the processing instruction
ntComment	'#comment'
ntDocument	'#document'
ntDocType	The document type name
ntDocFragment	'#document-fragment'
ntNotation	The notation name

Note: If GetPrefix (see page 26176) and GetLocalName (see page 26172) both return values for a node, the node name is the combination of these two values.

See Also

GetNodeType (see page 26174)

GetPrefix (see page 26176)

GetLocalName (see page 26172)

195.1.2.5.38 TXMLNode.GetNodeObject Method

Returns a reference to this TXMLNode (see page 26147) object.

Pascal

```
function GetNodeObject: TXMLNode;
```

C++

```
__fastcall TXMLNode GetNodeObject();
```

Description

GetNodeObject returns Self (Delphi) or this (C++).

195.1.2.5.39 TXMLNode.GetNodeType Method

Returns the type of the node.

Pascal

```
function GetNodeType: TNodeType;
```

C++

```
__fastcall TNodeType GetNodeType();
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected GetNodeType method. GetNodeType returns the type of the underlying DOM node that this TXMLNode (see page 26147) object represents.

See Also

GetIsTextElement (see page 26171)

195.1.2.5.40 TXMLNode.GetNodeValue Method

Returns the value of this node.

Pascal

```
function GetNodeValue: OleVariant;
```

C++

```
__fastcall OleVariant GetNodeValue();
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected GetNodeValue method. GetNodeValue returns the value of the node. This value depends on the type of the node, as indicated in the following table:

NodeType	Value
ntAttribute	The attribute value

ntElement	If the element contains only text, this is that text value. Otherwise, GetNodeValue raises an exception.
ntText	The text
ntCData	The content of the CDATA section.
ntEntityRef	nil (Delphi) or NULL (C++)
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.
ntDocFragment	nil (Delphi) or NULL (C++)

See Also

GetChildValue (see page 26169)

GetText (see page 26177)

195.1.2.5.41 TXMLNode.GetOwnerDocument Method

Returns the interface for the document in which this node appears.

Pascal

```
function GetOwnerDocument: IXMLDocument;
```

C++

```
__fastcall IXMLDocument GetOwnerDocument();
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected GetOwnerDocument method. GetOwnerDocument returns the value of the OwnerDocument (see page 26193) property.

See Also

OwnerDocument (see page 26193)

195.1.2.5.42 TXMLNode.GetParentNode Method

Returns the interface for the node's parent node.

Pascal

```
function GetParentNode: IXMLNode;
```

C++

```
__fastcall IXMLNode GetParentNode();
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected GetParentNode method. GetParentNode returns the interface of the node's parent node.

See Also

ParentNode (see page 26194)

195.1.2.5.43 TXMLNode.GetPrefix Method

Returns the namespace prefix of the node's name.

Pascal

```
function GetPrefix: DOMString;
```

C++

```
__fastcall DOMString GetPrefix();
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected GetPrefix method. GetPrefix returns the namespace prefix of an element or attribute node's name. A namespace prefix is a symbolic name for a namespace URI.

For element nodes, the namespace prefix is derived from the name that appears in the XML tag. For example, the namespace prefix for <xsi:Name First="John" Last="Doe"> is 'xsi'.

For attribute nodes, the namespace prefix is derived from the string that appears to the left of the equals sign in the attribute declaration. For example the element <xsi:Name xsn:First="John" Last="Doe"> results in two attribute nodes, with namespace prefixes of 'xsn' and '', respectively. Note that the attribute never inherits a namespace prefix from the node to which it is attached.

For any other node types, the namespace prefix is nil (Delphi) or NULL (C++).

Note: If both GetPrefix and GetLocalName (see page 26172) return a value for a node, the node name (returned by GetNodeName (see page 26173)) is the combination of these two values.

See Also

GetLocalName (see page 26172)

GetNodeType (see page 26174)

GetNamespaceURI (see page 26172)

GetPrefixedName (see page 26176)

195.1.2.5.44 TXMLNode.GetPrefixedName Method

Returns a node name includes a namespace prefix for a specified namespace URI.

Pascal

```
function GetPrefixedName(const Name: DOMString; const NamespaceURI: DOMString): DOMString;
```

C++

```
__fastcall DOMString GetPrefixedName(const DOMString Name, const DOMString NamespaceURI);
```

Description

GetPrefixedName adds a prefix to a local node name based on the specified namespace URI. If there is no corresponding namespace already declared or if the name is already prefixed, GetPrefixedName returns the original node name.

Name is the node name to which the namespace prefix should be added.

NamespaceURI is the URI for the namespace that should be identified by the namespace prefix.

Note: If the document containing this node does not include doAutoPrefix in its Options property, GetPrefixedName always returns the value of Name, without trying to generate or add a namespace prefix.

See Also

GetLocalName (see page 26172)

GetPrefix (see page 26176)

GetNamespaceURI (see page 26172)

Options (see page 26141)

195.1.2.5.45 TXMLNode.GetReadOnly Method

Indicates whether the node can be modified.

Pascal

```
function GetReadOnly: Boolean;
```

C++

```
__fastcall Boolean GetReadOnly();
```

Description

GetReadOnly returns true if the node can't be modified. Read-only nodes raise an EXMLDocError exception when an attempt is made to change their value, child nodes, or attributes.

GetReadOnly returns false if the node can be modified.

Note: Use the ReadOnly property on the IXMLNode interface to specify whether a node is read-only.

See Also

SetReadOnly (see page 26187)

ReadOnly (see page 26438)

EXMLDocError (see page 26381)

195.1.2.5.46 TXMLNode.GetText Method

Returns the text value of the node.

Pascal

```
function GetText: DOMString;
```

C++

```
__fastcall DOMString GetText();
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected GetText method. GetText returns the text of the node. For example, if the node represents an XML fragment such as

```
<Title> Understanding XML </Title>
```

GetText returns the string 'Understanding XML'.

GetText is intended for use when the GetIsTextElement (see page 26171) method returns true. If GetIsTextElement (see page 26171) returns false, GetText returns an empty string if the node has no children, otherwise it raises an exception.

See Also

GetIsTextElement (see page 26171)

GetNodeValue (see page 26174)

SetText (see page 26188)

195.1.2.5.47 TXMLNode.GetXML Method

Returns the XML that corresponds to the subtree rooted at this node.

Pascal

```
function GetXML: DOMString;
```

C++

```
__fastcall DOMString GetXML();
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected GetXML method. GetXML returns the XML that corresponds to this node and any child nodes it contains.

Note: GetXML is only available when the DOMVendor used to parse this node's XML document is MSXML.

See Also

GetText (see page 26177)

ChildNodes (see page 26190)

DOMVendor (see page 26137)

195.1.2.5.48 HasAttribute Method

This is the overview for the HasAttribute method overload.

Overload List

	Name	Description
	TXMLNode.HasAttribute (DOMString) (see page 26178)	Indicates whether the node has an attribute with a specified name.
	TXMLNode.HasAttribute (DOMString, DOMString) (see page 26178)	Indicates whether the node has an attribute with a specified name.

195.1.2.5.48.1 TXMLNode.HasAttribute Method (DOMString)

Indicates whether the node has an attribute with a specified name.

Pascal

```
function HasAttribute(const Name: DOMString): Boolean; overload;
function HasAttribute(const Name: DOMString; const NamespaceURI: DOMString): Boolean; overload;
```

C++

```
__fastcall Boolean HasAttribute(const DOMString Name);
__fastcall Boolean HasAttribute(const DOMString Name, const DOMString NamespaceURI);
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected HasAttribute method.

Name is the name of the attribute about which you want to know.

NamespaceURI is the namespace of the attribute.

HasAttribute returns true if this node has the specified attribute, false otherwise.

See Also

AttributeNodes (see page 26189)

GetAttribute (see page 26166)

GetAttributeNS (see page 26167)

195.1.2.5.49 HasChildNode Method

This is the overview for the HasChildNode method overload.

Overload List

	Name	Description
	TXMLNode.HasChildNode (DOMString) (see page 26179)	Indicates whether the node has a child node with the specified name.
	TXMLNode.HasChildNode (DOMString, DOMString) (see page 26179)	Indicates whether the node has a child node with the specified name.

195.1.2.5.49.1 TXMLNode.HasChildNode Method (DOMString)

Indicates whether the node has a child node with the specified name.

Pascal

```
function HasChildNode(const ChildTag: DOMString): Boolean; overload;
function HasChildNode(const ChildTag: DOMString; const NamespaceURI: DOMString): Boolean;
overload;
```

C++

```
__fastcall Boolean HasChildNode(const DOMString ChildTag);
__fastcall Boolean HasChildNode(const DOMString ChildTag, const DOMString NamespaceURI);
```

Description

Applications can't call the protected HasChildNode method. It is used internally to determine when the underlying DOM node has a child with the specified name.

ChildTag is the local tag name of the child to find.

NamespaceURI identifies the namespace in which ChildTag is defined.

HasChildNode returns true if the underlying DOM node includes the specified child node. Otherwise, it returns false.

See Also

ChildNodes (see page 26190)

DOMNode (see page 26192)

195.1.2.5.50 TXMLNode.InternalAddChild Method

Provides the underlying implementation for the AddChild (see page 26156) method.

Pascal

```
function InternalAddChild(NodeClass: TXMLNodeClass; const NodeName: DOMString; const
```

```
NamespaceURI: DOMString; Index: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode InternalAddChild(TXMLNodeClass NodeClass, const DOMString NodeName, const
DOMString NamespaceURI, int Index);
```

Description

Applications can't call the protected InternalAddChild method. It is used internally to perform all the tasks common to the various overloads of the AddChild (see page 26156) method. InternalAddChild creates a new element node as the child of this node and returns its interface.

NodeClass specifies the implementation class to use for the new child node. It must be TXMLNode (see page 26147) or one of its descendants.

NodeName specifies the tag name of the newly created node.

NamespaceURI identifies the namespace that includes the new node's definition.

Index indicates the position of the child node in this node's list of children, where 0 is the first position, 1 is the second position, and so on. If Index is –1, the new node is added to the end.

See Also

AddChild (see page 26156)

195.1.2.5.51 TXMLNode.NestingLevel Method

Returns the number of parents for this node in the node hierarchy.

Pascal

```
function NestingLevel: Integer;
```

C++

```
__fastcall int NestingLevel();
```

Description

Application's can't call the protected NestingLevel property. It is used internally when formatting the XML text that corresponds to this XML document.

NestingLevel returns the number of ancestors for this node in the node hierarchy.

See Also

ParentNode (see page 26194)

ChildNodes (see page 26190)

195.1.2.5.52 TXMLNode.NextSibling Method

Returns the next child of this node's parent.

Pascal

```
function NextSibling: IXMLNode;
```

C++

```
__fastcall IXMLNode NextSibling();
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected NextSibling method. NextSibling

returns the node that follows this one in the parent node's ChildNodes (see page 26190) property list.

If this node is the last node in its parent's child list, NextSibling raises an exception.

See Also

ParentNode (see page 26194)

PreviousSibling (see page 26181)

ChildNodes (see page 26190)

195.1.2.5.53 TXMLNode.Normalize Method

Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.

Pascal

```
procedure Normalize;
```

C++

```
__fastcall Normalize();
```

Description

Use the IXMLNode interface to call the protected Normalize method. Normalize consolidates adjacent text nodes and remove any empty text nodes in the subtree beneath this node. This method ensures that for all child nodes of this node and of its attributes, only structure nodes (element nodes, comments, processing instructions, CDATA sections, and entity references) separate text nodes and that there are no empty text nodes.

See Also

ChildNodes (see page 26190)

AttributeNodes (see page 26189)

195.1.2.5.54 TXMLNode.PreviousSibling Method

Returns the previous child of this node's parent.

Pascal

```
function PreviousSibling: IXMLNode;
```

C++

```
__fastcall IXMLNode PreviousSibling();
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected PreviousSibling method. PreviousSibling returns the node that precedes this one in the parent node's ChildNodes (see page 26190) property list.

If this node is the first node in its parent's child list, PreviousSibling raises an exception.

See Also

ParentNode (see page 26194)

NextSibling (see page 26180)

ChildNodes (see page 26190)

195.1.2.5.55 TXMLNode.RegisterChildNode Method

Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.

Pascal

```
procedure RegisterChildNode(const TagName: DOMString; ChildNodeClass: TXMLNodeClass;
NamespaceURI: DOMString = '');
```

C++

```
__fastcall RegisterChildNode(const DOMString TagName, TXMLNodeClass ChildNodeClass, DOMString
NamespaceURI = '');
```

Description

Applications can't call the protected RegisterChildNode method. It is called by the RegisterChildNodes (see page 26182) method to register a TXMLNode (see page 26147) descendant so that it can be used as a child of this node. Each TXMLNode (see page 26147) descendant is specific to the structure of the XML document, with properties that correspond to the child nodes and attributes of a node.

RegisterChildNode adds an entry to the ChildNodeClasses (see page 26190) property for the specified node class and tag name.

TagName is the tag name of the child node that the class represents.

NodeClass is the TXMLNode (see page 26147) descendant that implements child nodes that have the specified tag name.

NamespaceURI identifies the namespace in which TagName is defined.

Note: RegisterChildNode raises an exception if this is a hosted node.

See Also

RegisterChildNodes (see page 26182)

ChildNodeClasses (see page 26190)

195.1.2.5.56 TXMLNode.RegisterChildNodes Method

Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.

Pascal

```
procedure RegisterChildNodes(const TagNames: array of DOMString; const NodeClasses: array of
TXMLNodeClass);
```

C++

```
__fastcall RegisterChildNodes(const array of DOMString TagNames, const array of TXMLNodeClass
NodeClasses);
```

Description

Applications can't call the protected RegisterChildNodes method. It is used in code that the XML Data Binding wizard generates to register the special TXMLNode (see page 26147) descendants the are created for nodes in the XML document. Each of these descendant classes is specific to the structure of the XML document, with properties that correspond to the child nodes and attributes of the node.

RegisterChildNodes adds an entry to the ChildNodeClasses (see page 26190) property for each specified node class and tag name.

TagNames is an array containing the names of the child nodes that the classes represent.

TagNames_Size is the index of the last name in TagNames (one less than the number of elements).

NodeClasses is an array containing the TXMLNode (see page 26147) descendants that implement child nodes that have the tag names in the corresponding entries in TagNames.

NodeClasses_Size is the index of the last class in NodeClasses (one less than the number of elements).

See Also

RegisterChildNode (see page 26182)

ChildNodeClasses (see page 26190)

195.1.2.5.57 TXMLNode.RemoveHostedNode Method

Removes a node from the HostedNodes (see page 26192) property list.

Pascal

```
procedure RemoveHostedNode(Node: TXMLNode);
```

C++

```
__fastcall RemoveHostedNode(TXMLNode Node);
```

Description

RemoveHostedNode is used internally to remove nodes from the HostedNodes (see page 26192) array.

Node is the node to remove from the array.

See Also

HostedNodes (see page 26192)

AddHostedNode (see page 26157)

195.1.2.5.58 TXMLNode.Resync Method

Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

Pascal

```
procedure Resync;
```

C++

```
__fastcall Resync();
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected Resync method. Resync forces the node to reread the values of its child nodes and attribute nodes, discarding any changes that have not been propagated to the underlying DOM implementation.

See Also

ChildNodes (see page 26190)

AttributeNodes (see page 26189)

Refresh (see page 26123)

195.1.2.5.59 TXMLNode.SetAttribute Method

Sets the value of one of this node's attributes.

Pascal

```
procedure SetAttribute(const AttrName: DOMString; const Value: OleVariant);
```

C++

```
__fastcall SetAttribute(const DOMString AttrName, const OleVariant Value);
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected SetAttribute method. SetAttribute sets the value of a specified attribute of this node.

AttrName is the name of the attribute. If the attribute already exists for the node, its value is changed. If the attribute does not already exist, a new attribute is created.

Value is the value to assign to the attribute. This value is a string that is not parsed. Any markup (for example, for an entity reference) is treated as literal text, and must include appropriate escape sequences. If Value is NULL, the attribute is removed from this node. If Value is an empty string, the attribute is assigned an empty value.

See Also

GetAttribute (see page 26166)

AttributeNodes (see page 26189)

SetAttributeNS (see page 26184)

195.1.2.5.60 TXMLNode.SetAttributeNodes Method

Sets the value of the AttributeNodes (see page 26189) property.

Pascal

```
procedure SetAttributeNodes(const Value: IXMLNodeList); virtual;
```

C++

```
virtual __fastcall SetAttributeNodes(const IXMLNodeList Value);
```

Description

Applications can't call this protected method. It is called internally the first time an application reads the AttributeNodes (see page 26189) property to supply the interface that implements that property.

See Also

AttributeNodes (see page 26189)

195.1.2.5.61 TXMLNode.SetAttributeNS Method

Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.

Pascal

```
procedure SetAttributeNS(const AttrName: DOMString; const NamespaceURI: DOMString; const
Value: OleVariant);
```

C++

```
__fastcall SetAttributeNS(const DOMString AttrName, const DOMString NamespaceURI, const
OleVariant Value);
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected SetAttributeNS method.

SetAttributeNS sets the value of a specified attribute of this node.

AttrName is the name of the attribute. If the attribute already exists for the node, its value is changed. If the attribute does not already exist, a new attribute is created.

NamespaceURI is the namespace for the attribute.

Value is the value to assign to the attribute. This value is a string that is not parsed. Any markup (for example, for an entity reference) is treated as literal text, and must include appropriate escape sequences. If Value is NULL, the attribute is removed from this node. If Value is an empty string, the attribute is assigned an empty value.

See Also

SetAttribute (see page 26183)

GetAttributeNS (see page 26167)

AttributeNodes (see page 26189)

195.1.2.5.62 TXMLNode.SetChildNodes Method

Sets the value of the ChildNodes (see page 26190) property.

Pascal

```
procedure SetChildNodes(const Value: IXMLNodeList); virtual;
```

C++

```
virtual __fastcall SetChildNodes(const IXMLNodeList Value);
```

Description

Applications can't call this protected method. It is called internally the first time an application reads the ChildNodes (see page 26190) property to supply the interface that implements that property.

See Also

ChildNodes (see page 26190)

195.1.2.5.63 TXMLNode.SetChildValue Method

Sets the value of a specified child node.

Pascal

```
procedure SetChildValue(const IndexOrName: OleVariant; const Value: OleVariant);
```

C++

```
__fastcall SetChildValue(const OleVariant IndexOrName, const OleVariant Value);
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected SetChildValue method. SetChildValue sets the value of a specified child node to a specified value.

IndexOrName identifies the desired child node by either its interface, or by a string that is the local name of the child node. If IndexOrName does not identify a child of this node, SetChildValue raises an exception.

Value is the value to assign. It's interpretation depends on the type of the child node, as indicated in the following table:

NodeType	Value
ntAttribute	The attribute value

ntElement	If the element contains only text, this is that text value. Otherwise, SetChildValue raises an exception.
ntText	The text
ntCData	The content of the CDATA section.
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.
If the child node is any other node type, SetChildValue raises an exception.	

Note: Attributes are not considered child nodes of a node. To set the values of this node's attributes, use the SetAttribute (see page 26183) method instead.

See Also

ChildNodes (see page 26190)

ParentNode (see page 26194)

SetAttribute (see page 26183)

SetNodeValue (see page 26186)

195.1.2.5.64 TXMLNode.SetCollection Method

Sets the value of the Collection (see page 26191) property.

Pascal

```
procedure SetCollection(const Value: TXMLNodeCollection);
```

C++

```
__fastcall SetCollection(const TXMLNodeCollection Value);
```

Description

SetCollection assigns Value as the value of the Collection (see page 26191) property. This protected method is called when this node is added to the collection specified by the Value property.

Value is a TXMLNodeCollection (see page 26196) descendant that represents a repeating set of child nodes, of which this node is a member.

See Also

Collection (see page 26191)

195.1.2.5.65 TXMLNode.SetNodeValue Method

Sets the value of this node.

Pascal

```
procedure SetNodeValue(const Value: OleVariant);
```

C++

```
__fastcall SetNodeValue(const OleVariant Value);
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected SetNodeValue method. SetNodeValue

sets the value of this node.

Value is the value to assign. It's meaning depends on the type of the node, as indicated in the following table:

NodeType	Value
ntAttribute	The attribute value
ntElement	If the element contains only text, this is that text value. Otherwise, SetNodeValue raises an exception.
ntText	The text
ntCData	The content of the CDATA section.
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.
If the node is any other node type, SetNodeValue raises an exception.	

See Also

SetChildValue (see page 26185)

SetText (see page 26188)

195.1.2.5.66 TXMLNode.SetParentNode Method

Sets the value of the ParentNode (see page 26194) property.

Pascal

```
procedure SetParentNode(const Value: TXMLNode); virtual;
```

C++

```
virtual __fastcall SetParentNode(const TXMLNode Value);
```

Description

Applications can't use this protected property. It is used internally when nodes are added to the ChildNodes (see page 26190) or AttributeNodes (see page 26189) property.

See Also

ParentNode (see page 26194)

ChildNodes (see page 26190)

AttributeNodes (see page 26189)

195.1.2.5.67 TXMLNode.SetReadOnly Method

Specifies whether the node can be modified.

Pascal

```
procedure SetReadOnly(const Value: Boolean);
```

C++

```
__fastcall SetReadOnly(const Boolean Value);
```

Description

SetReadOnly assigns the read-only status of the node.

Value is the new read-only status of the node. When Value is true, the node raises an EXMLDocError exception when an attempt is made to change its value, child nodes, or attributes. When Value is false, the node can be modified.

See Also

GetReadOnly (see page 26177)

ReadOnly (see page 26438)

EXMLDocError (see page 26381)

195.1.2.5.68 TXMLNode.SetText Method

Sets the text value of the node.

Pascal

```
procedure SetText(const Value: DOMString);
```

C++

```
__fastcall SetText(const DOMString Value);
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected SetText method. SetText replaces the text of the node.

SetText is intended for use when the GetIsTextElement (see page 26171) method returns true. If GetIsTextElement (see page 26171) returns false, SetText adds a new text value to a node that has no children, otherwise, it raises an exception.

See Also

GetIsTextElement (see page 26171)

SetNodeValue (see page 26186)

GetText (see page 26177)

195.1.2.5.69 TransformNode Method

This is the overview for the TransformNode method overload.

Overload List

	Name	Description
	TXMLNode.TransformNode (IXMLNode, IXMLDocument) (see page 26188)	Transforms the subtree rooted at this node, using a specified XSL interface.
	TXMLNode.TransformNode (IXMLNode, WideString) (see page 26188)	Transforms the subtree rooted at this node, using a specified XSL interface.

195.1.2.5.69.1 TXMLNode.TransformNode Method (IXMLNode, IXMLDocument)

Transforms the subtree rooted at this node, using a specified XSL interface.

Pascal

```
procedure TransformNode(const stylesheet: IXMLNode; var output: WideString); overload;
procedure TransformNode(const stylesheet: IXMLNode; const output: IXMLDocument); overload;
```

C++

```
__fastcall TransformNode(const IXMLNode stylesheet, BSTR output);
__fastcall TransformNode(const IXMLNode stylesheet, const IXMLDocument output);
```

Description

Use the TXMLNode (see page 26147) object's IXMLNode interface to call the protected TransformNode method.

stylesheet is a node in an XML document that represents an XSL transformation. Typically, it is the document element of that document.

output represents the result of the transformation. If output is a WideString, TransformNode returns a string of XML. If output is the interface to an XML document, TransformNode replaces the content of that document with the results of the transformation.

Note: TransformNode is only available when the DOMVendor used to parse this node's XML document is MSXML.

See Also

GetXML (see page 26178)

195.1.2.6 TXMLNode Properties

The properties of the TXMLNode class are listed here.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

195.1.2.6.1 TXMLNode.AttributeNodes Property

Lists the interfaces for nodes that represent attributes of this node.

Pascal

```
property AttributeNodes: IXMLNodeList;
```

C++

```
__property IXMLNodeList AttributeNodes;
```

Description

TXMLNode (see page 26147) uses AttributeNodes to keep track of its attributes. AttributeNodes is an IXMLNodeList interface, which can be used to access individual attribute nodes or to add attributes to or delete them from this node.

Applications can't use this protected property to access the attributes of a node. Instead, they use the AttributeNodes property of the IXMLNode interface for this node.

Attributes modify a node in an XML document. For example, given the node

```
<Movie genre=scifi title="Star Wars"> ... </Movie>
```

both genre and title are attributes of the node, with values of "scifi" and "Star Wars", respectively.

Only element nodes have attributes, and not every element node has them. If the node does not have any attributes, AttributeNodes is the interface for an empty list.

Note: Attributes are not considered child nodes of a node and can't be accessed using the ChildNodes (see page 26190) property.

See Also

IXMLNode (see page 26411)

ChildNodes (see page 26190)

AttributeNodes (see page 26430)

195.1.2.6.2 TXMLNode.ChildNodeClasses Property

Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.

Pascal

```
property ChildNodeClasses: TNodeClassArray;
```

C++

```
__property TNodeClassArray ChildNodeClasses;
```

Description

ChildNodeClasses is used internally when creating child nodes for this node. The XML Data Binding wizard defines TXMLNode (see page 26147) descendants for each node in the XML document that represents a node that has (or can have) children. ChildNodeClasses lists only those TXMLNode (see page 26147) descendants that can represent child nodes of this node. These classes become the types of the corresponding properties on the TXMLNode (see page 26147) descendant that represents this node.

Classes (see page 2685) are added to ChildNodeClasses by the RegisterChildNode (see page 26182) method.

See Also

RegisterChildNode (see page 26182)

RegisterChildNodes (see page 26182)

ChildNodes (see page 26190)

CreateChildNode (see page 26162)

195.1.2.6.3 TXMLNode.ChildNodes Property

Lists the child nodes of this node.

Pascal

```
property ChildNodes: IXMLNodeList;
```

C++

```
__property IXMLNodeList ChildNodes;
```

Description

TXMLNode (see page 26147) uses ChildNodes to keep track of the subnodes of a node. ChildNodes is an IXMLNodeList interface, which can be used to access individual child nodes or to add child nodes to or delete them from this node. Each child node specifies this node as its parent node.

Applications can't use this protected property to access the child nodes of a node. Instead, they use the ChildNodes property of the IXMLNode interface for this node.

Note: Attributes are not considered child nodes of a node. To get the values of this node's attributes, use the Attributes property instead.

See Also

IXMLNode (see page 26411)

ChildNodes (see page 26431)

ParentNode (see page 26194)

195.1.2.6.4 TXMLNode.Collection Property

Indicates the collection in which this node appears.

Pascal

```
property Collection: TXMLNodeCollection;
```

C++

```
__property TXMLNodeCollection Collection;
```

Description

Collection indicates the collection that contains this node if it represents one of a repeating set of child nodes. The node specified by Collection represents the same node in the XML document as this node's ParentNode (see page 26194) property, but it is specialized to work with this node and any other child nodes of the same type.

The node identified by Collection is one of the collections listed by the parent node's HostedNodes (see page 26192) property, and it lists ParentNode (see page 26194) as the value of its HostNode (see page 26193) property.

For example, consider the following XML fragment:

```
<students>
  <student name="John Smith">
    <grade>incomplete</grade>
  </student>
  <student name="Jane Doe">
    <grade>B+</grade>
  </student>
  <auditor name="Bill Jones"></auditor>
</students>
```

If this node represents one of the <student> nodes, then ParentNode (see page 26194) represents the <students> node, and Collection represents a copy of the <students> node that is used for only those children that are <student> nodes, and not the <auditor> nodes.

Applications can't use this protected property to access the Collection that contains this node. Instead, they use the Collection property of the IXMLNode interface for this node.

See Also

ParentNode (see page 26194)

HostedNodes (see page 26192)

HostNode (see page 26193)

Collection (see page 26432)

195.1.2.6.5 TXMLNode.DOMNode Property

Provides access to the IDOMNode interface for this node.

Pascal

```
property DOMNode: IDOMNode;
```

C++

```
__property IDOMNode DOMNode;
```

Description

IDOMNode is the interface for this node provided by the DOM implementation that parsed the XML document. It can be accessed through the DOMNode property of the node's IXMLNode interface.

Typically, applications that contain TXMLNode (see page 26147) descendants use the IXMLNode interface to work with a node rather than the IDOMNode interface. IDOMNode is used more typically in applications that work directly with the DOM interfaces.

The value of DOMNode is set to the IDOMNode interface that is provided to the TXMLNode (see page 26147) constructor.

See Also

DOMNode (see page 26433)

Create (see page 26152)

195.1.2.6.6 TXMLNode.HostedNodes Property

Lists the nodes that share this node's lists of child and attribute nodes.

Pascal

```
property HostedNodes: TXMLNodeArray;
```

C++

```
__property TXMLNodeArray HostedNodes;
```

Description

TXMLNode (see page 26147) uses HostedNodes to keep track of the nodes it uses to represent this same node, restricted to a specialized purpose. For example, any node that represents a subset of this node's child nodes that are all of the same type is a hosted node.

Each object in the HostedNodes list is a representation of a particular aspect of this node. Each hosted node acts as a sibling to this TXMLNode (see page 26147) object but does not represent a separate element in the XML document. Rather, it represents a different view of this element.

For example, consider the following XML fragment:

```
<students>
  <fulltime student>Joe Smith</fulltime student>
  <fulltime student>Jane Doe</fulltime student>
  <auditor>Bill Jones</auditor>
</students>
```

The TXMLNode (see page 26147) that represents students would define two hosted nodes for collections of repeating child elements, one for the fulltime students, and one for the auditors. The two collections share the same underlying DOM node as students, but each represents only those child elements of a particular type.

See Also

CreateCollection (see page 26163)

ChildNodes (see page 26190)

TXMLNodeCollection (see page 26196)

Collection (see page 26191)

HostNode (see page 26193)

195.1.2.6.7 TXMLNode.HostNode Property

Indicates the main object for the node to which this node corresponds when it represents a specialized view.

Pascal

```
property HostNode: TXMLNode;
```

C++

```
__property TXMLNode HostNode;
```

Description

If this node represents a hosted node, HostNode is the main node for which this node represents a subset or specialized view. For example, if this node represents a set of repeating child elements, HostNode is the node that uses this node to represent its repeating child elements.

For example, consider the following XML fragment:

```
<students>
  <student name="John Smith">
    <grade>incomplete</grade>
  </student>
  <student name="Jane Doe">
    <grade>B+</grade>
  </student>
  <auditor name="Bill Jones"></auditor>
  <auditor name="Susan Brown"></auditor>
</students>
```

The <students> node spawns two hosted nodes to represent the collections of repeating child nodes: one for all of the child nodes of type <student>, and one for all of the child nodes of type <auditor>. If this node represents one of those hosted nodes, then HostNode is the <students> node to which this node belongs. The host node lists this node in its HostedNodes (see page 26192) property.

Applications can't use this protected property to access the host node that created this node. Instead, they use the HostNode property of the IXMLNode interface for this node.

See Also

Collection (see page 26191)

HostedNodes (see page 26192)

TXMLNodeCollection (see page 26196)

195.1.2.6.8 TXMLNode.OwnerDocument Property

Provides access to the document that contains this node.

Pascal

```
property OwnerDocument: TXMLDocument;
```

C++

```
__property TXMLDocument OwnerDocument;
```

Description

Applications can't use this protected property to access the document that contains this. Instead, they use the OwnerDocument property of the IXMLNode interface for this node.

The value of OwnerDocument is set to the TXMLDocument (see page 26097) that is provided to the TXMLNode (see page 26147) constructor.

See Also

Create (see page 26152)

OwnerDocument (see page 26437)

195.1.2.6.9 TXMLNode.ParentNode Property

Identifies the object that represents this node's parent node.

Pascal

```
property ParentNode: TXMLNode;
```

C++

```
__property TXMLNode ParentNode;
```

Description

TXMLNode (see page 26147) uses ParentNode to access the implementation of the node's parent, if it exists. Some nodes (for example attribute nodes) do not have parents.

Applications can't use this protected property to access the parent node of a node. Instead, they use the ParentNode property of the IXMLNode interface for this node.

See Also

ChildNodes (see page 26190)

ParentNode (see page 26437)

195.1.2.7 TXMLNode Events

The events of the TXMLNode class are listed here.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

195.1.2.7.1 TXMLNode.OnHostAttrNotify Event

Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).

Pascal

```
property OnHostAttrNotify: TNodeListNotification;
```

C++

```
__property TNodeListNotification OnHostAttrNotify;
```

Description

Use the OnHostAttrNotify event to update a TXMLNode (see page 26147) descendant that represents a hosted node (such as a collection of repeating child elements) when the host node changes its attribute list.

The AttributeListNotify (see page 26158) method of a host node generates an OnHostAttrNotify event on all of its hosted nodes. This allows the hosted nodes (for example, the nodes that represent a subset of the child nodes that are all the same type) to update their own representation of node attributes so that they reflect the attributes of the host node.

See Also

OnHostChildNotify (see page 26195)

AttributeListNotify (see page 26158)

HostNode (see page 26193)

HostedNodes (see page 26192)

AttributeNodes (see page 26189)

195.1.2.7.2 TXMLNode.OnHostChildNotify Event

Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Pascal

```
property OnHostChildNotify: TNodeListNotification;
```

C++

```
__property TNodeListNotification OnHostChildNotify;
```

Description

Use the OnHostChildNotify event to update a TXMLNode (see page 26147) descendant that represents a hosted node (such as a collection of repeating child elements) when the host node changes its list of child nodes. For example, TXMLNodeCollection (see page 26196) assigns its UpdateCollectionList method as an OnHostChildNotify event handler so that its list of repeating child elements reflects any changes to the host node's list of child nodes.

The ChildListNotify (see page 26159) method of a host node generates an OnHostChildNotify event on all of its hosted nodes.

See Also

OnHostAttrNotify (see page 26194)

ChildListNotify (see page 26159)

HostNode (see page 26193)

HostedNodes (see page 26192)

ChildNodes (see page 26190)

UpdateCollectionList (see page 26208)

195.1.3 TXMLNodeCollection Class

TXMLNodeCollection represents a node in an XML document whose children are instances of the same repeating element.

Class Hierarchy

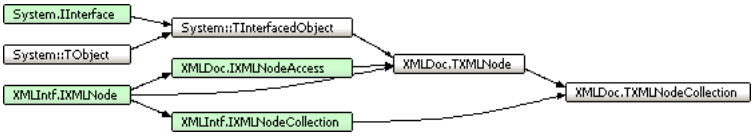

Pascal

```
TXMLNodeCollection = class(TXMLNode, IXMLNodeCollection);
```

C++

```
class TXMLNodeCollection : public TXMLNode, public IXMLNodeCollection;
```

File

XMLDoc

Description

The XML Data Binding Wizard uses TXMLNodeCollection as a base class for the classes it generates to represent nodes that have repeating elements for child nodes. For example, if the XML document includes the following:

```
<students>
  <student name="John Smith">
    <grade>incomplete</grade>
  </student>
  <student name="Jane Doe">
    <grade>B+</grade>
  </student>
  <auditor name="Bill Jones"></auditor>
</students>
```

then the XML Data Binding Wizard generates 3 nodes to represent the <students> node: a TXMLNode (see page 26147) descendant to represent the entire <students> node, and two TXMLNodeCollection descendants to represent the repeating elements: one for the child nodes with the tag <student>, and one for the child nodes with the tag <auditor>.

As with any of the classes the Data Binding Wizard generates to represent nodes in an XML document, child nodes and node attributes appear as properties of the TXMLNodeCollection descendant.

TXMLNodeCollection implements the IXMLNodeCollection interface. Each TXMLNodeCollection descendant implements a class-specific interface that descends from IXMLNodeCollection. Typically, applications do not work directly with TXMLNodeCollection. Instead, they use the IXMLNodeCollection interface or the IXMLNodeCollection descendant that is implemented by a TXMLNodeCollection descendant.

See Also

TXMLDocument (see page 26097)

TXMLNode (see page 26147)

IXMLNodeCollection (see page 26440)

195.1.3.1 TXMLNodeCollection Members

The following tables list the members exposed by TXMLNodeCollection.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 26194)	Occurs when changes occur to the attribute list of this node's HostNode (see page 26193).
	OnHostChildNotify (see page 26195)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 26193).

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 26155)	Increments the reference count for this node.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	_Release (see page 26155)	Decrements the reference count for this node.
	AddChild (see page 26156)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 26157)	Adds a node to the HostedNodes (see page 26192) property list.
V	AddItem (see page 26202)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 26158)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 26158)	Raises an EXMLDocError exception if this is a hosted node.
	CheckReadOnly (see page 26159)	Raises an EXMLDocError exception if this is a read-only node.
	CheckTextNode (see page 26159)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 26203)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 26204)	Removes all child nodes in the List (see page 26210) property from the ChildNodes (see page 26190) and List (see page 26210) properties.
	ClearDocumentRef (see page 26160)	Sets the value of OwnerDocument (see page 26193) to nil (Delphi) or NULL (C++).
	CloneNode (see page 26160)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 26161)	Creates the object that implements the AttributeNodes (see page 26189) property.
V	CreateAttributeNode (see page 26161)	Creates a new TXMLNode (see page 26147) object for an attribute of this node.
	CreateChildList (see page 26162)	Creates the object that implements the ChildNodes (see page 26190) property.
V	CreateChildNode (see page 26162)	Creates a new TXMLNode (see page 26147) object for a child node of this node.
	CreateCollection (see page 26163)	Creates an instance of a TXMLNodeCollection (see page 26196) descendant and returns it.
	CreateItemList (see page 26204)	Creates the object that implements the List (see page 26210) property.
	DeclareNamespace (see page 26164)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26205)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 26164)	Returns the IDOMElement interface for this node if it exists.
V	DoNodeChange (see page 26165)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 26165)	Returns the IXMLNode interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 26165)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26166)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26166)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26167)	Returns the value of the AttributeNodes (see page 26189) property.

	GetAttributeNS (see page 26167)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 26168)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 26168)	Returns the value of the ChildNodes (see page 26190) property.
	GetChildValue (see page 26169)	Returns the value of a specified child node.
	GetCollection (see page 26169)	Returns the value of the Collection (see page 26191) property.
	GetCount (see page 26205)	Returns the value of the Count (see page 26209) property.
	GetDOMNode (see page 26170)	Returns the value of the DOMNode (see page 26192) property.
	GetHasChildNodes (see page 26170)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 26171)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 26171)	Returns the value of the HostNode (see page 26193) property.
	GetIsTextElement (see page 26171)	Indicates whether the node has a single text value.
	GetList (see page 26205)	Returns the value of the List (see page 26210) property.
	GetLocalName (see page 26172)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26172)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 26206)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26173)	Returns the node name.
	GetNodeObject (see page 26174)	Returns a reference to this TXMLNode (see page 26147) object.
	GetNodeType (see page 26174)	Returns the type of the node.
	GetNodeValue (see page 26174)	Returns the value of this node.
	GetOwnerDocument (see page 26175)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26175)	Returns the interface for the node's parent node.
	GetPrefix (see page 26176)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 26176)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 26177)	Indicates whether the node can be modified.
	GetText (see page 26177)	Returns the text value of the node.
	GetXML (see page 26178)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 26178)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26179)	This is the overview for the HasChildNode method overload.
	InsertInCollection (see page 26206)	Copies a child node from the ChildNodes (see page 26190) property to the List (see page 26210) property.
	InternalAddChild (see page 26179)	Provides the underlying implementation for the AddChild (see page 26156) method.
	IsCollectionItem (see page 26207)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 26180)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 26180)	Returns the next child of this node's parent.
	Normalize (see page 26181)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26181)	Returns the previous child of this node's parent.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 26182)	Registers a descendant of TXMLNode (see page 26147) that can represent the children of this node.
	RegisterChildNodes (see page 26182)	Registers a set of TXMLNode (see page 26147) descendants that can represent the children of this node.
	Remove (see page 26207)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 26183)	Removes a node from the HostedNodes (see page 26192) property list.
	Resync (see page 26183)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26183)	Sets the value of one of this node's attributes.
	SetAttributeNodes (see page 26184)	Sets the value of the AttributeNodes (see page 26189) property.
	SetAttributeNS (see page 26184)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildNodes (see page 26208)	Sets the value of the ChildNodes (see page 26190) property.
	SetChildValue (see page 26185)	Sets the value of a specified child node.

	SetCollection (see page 26186)	Sets the value of the Collection (see page 26191) property.
	SetNodeValue (see page 26186)	Sets the value of this node.
V	SetParentNode (see page 26187)	Sets the value of the ParentNode (see page 26194) property.
	SetReadOnly (see page 26187)	Specifies whether the node can be modified.
	SetText (see page 26188)	Sets the text value of the node.
	TransformNode (see page 26188)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 26208)	Updates the List (see page 26210) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 26189)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 26190)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 26190)	Lists the child nodes of this node.
	Collection (see page 26191)	Indicates the collection in which this node appears.
R	Count (see page 26209)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26192)	Provides access to the IDOMNode interface for this node.
R	HostedNodes (see page 26192)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 26193)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 26209)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 26210)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 26210)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 26210)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 26193)	Provides access to the document that contains this node.
R	ParentNode (see page 26194)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 26202)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26152)	Instantiates a Create object.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 26152)	Creates a TXMLNode (see page 26147) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.

	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Destroy (see page 26153)	Frees the memory associated with the TXMLNode (see page 26147) object.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.

	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

195.1.3.2 TXMLNodeCollection Methods

The methods of the TXMLNodeCollection class are listed here.

Protected Methods

	Name	Description
V	AddItem (see page 26202)	Adds a new repetition of the repeating child element to this node.
V	ChildListNotify (see page 26203)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 26204)	Removes all child nodes in the List (see page 26210) property from the ChildNodes (see page 26190) and List (see page 26210) properties.

	CreateItemList (see page 26204)	Creates the object that implements the List (see page 26210) property.
	Delete (see page 26205)	Removes one of the repeating child nodes from this node.
	GetCount (see page 26205)	Returns the value of the Count (see page 26209) property.
V	GetList (see page 26205)	Returns the value of the List (see page 26210) property.
	GetNode (see page 26206)	Returns the interface for one of the repeating child nodes.
	InsertInCollection (see page 26206)	Copies a child node from the ChildNodes (see page 26190) property to the List (see page 26210) property.
	IsCollectionItem (see page 26207)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	Remove (see page 26207)	Removes one of the repeating child nodes from this node.
V	SetChildNodes (see page 26208)	Sets the value of the ChildNodes (see page 26190) property.
	UpdateCollectionList (see page 26208)	Updates the List (see page 26210) property to reflect changes to the set of child nodes.

Public Methods

	Name	Description
V	AfterConstruction (see page 26202)	Initializes the node collection object once it is fully created.

195.1.3.2.1 TXMLNodeCollection.AddItem Method

Adds a new repetition of the repeating child element to this node.

Pascal

```
function AddItem(Index: Integer): IXMLNode; virtual;
```

C++

```
virtual __fastcall IXMLNode AddItem(int Index);
```

Description

Applications can't call the protected AddItem method. It is used internally to create a new child node for this node that is an instance of the repeating element. The new child node is given the first tag name listed by the ItemTag (see page 26210) property.

Index indicates the position of the new child node in this node's list of children, where 0 is the first position, 1 is the second position, and so on. If Index is –1, the new node is added to the end.

Note: Index refers to the list of repeating elements (the List (see page 26210) property), not the list of all children (the ChildNodes (see page 26190) property).

AddItem returns the interface for the newly created child node.

See Also

AddChild (see page 26156)

InsertInCollection (see page 26206)

itemtag (see page 26210)

List (see page 26210)

ChildNodes (see page 26190)

195.1.3.2.2 TXMLNodeCollection.AfterConstruction Method

Initializes the node collection object once it is fully created.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

AfterConstruction sets the UpdateCollectionList (see page 26208) method as the event handler for the collection's OnHostChildNotify (see page 26195) event. This allows the collection to update itself when the list of child nodes (for which this collection manages a subset) changes.

See Also

OnHostChildNotify (see page 26195)

UpdateCollectionList (see page 26208)

195.1.3.2.3 TXMLNodeCollection.ChildListNotify Method

Responds to notifications of changes in the list of child nodes.

Pascal

```
procedure ChildListNotify(Operation: TNodeListOperation; var Node: IXMLNode; const
IndexOrName: OleVariant; BeforeOperation: Boolean); override;
```

C++

```
virtual __fastcall ChildListNotify(TNodeListOperation Operation, IXMLNode Node, const
OleVariant IndexOrName, Boolean BeforeOperation);
```

Description

Applications can't call this protected method. The class that implements the ChildNodes (see page 26190) property calls ChildListNotify when child nodes are inserted, removed, or need to be created. ChildListNotify calls the inherited method to generate a BeforeNodeChange or AfterNodeChange event and to ensure that the DOM implementation for this node reflects the change. After calling the inherited method, ChildListNotify updates the List (see page 26210) property so that it stays in sync with the ChildNodes (see page 26190) property.

Operation indicates what type of change is about to occur.

Node is the interface for the child node that is added or removed. If Operation is nlCreateNode, Node returns a newly created child node.

IndexOrName is the index of the node that is added or removed when Operation is nlInsert or nlRemove. When Operation is nlCreateNode, IndexOrName is the name or value of the new node to create.

BeforeOperation indicates whether the call occurs before or after the change.

See Also

ChildNodes (see page 26190)

AddChild (see page 26156)

List (see page 26210)

BeforeNodeChange (see page 26146)

AfterNodeChange (see page 26145)

195.1.3.2.4 TXMLNodeCollection.Clear Method

Removes all child nodes in the List (see page 26210) property from the ChildNodes (see page 26190) and List (see page 26210) properties.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Use the IXMLNodeCollection interface for this node to call the protected Clear method. Clear deletes every child node from the collection (all the nodes in the List (see page 26210) property), along with the corresponding nodes in the ChildNodes (see page 26190) property.

See Also

List (see page 26210)

Delete (see page 26205)

Remove (see page 26207)

ChildNodes (see page 26190)

Clear (see page 26442)

195.1.3.2.5 TXMLNodeCollection.CreateItemList Method

Creates the object that implements the List (see page 26210) property.

Pascal

```
procedure CreateItemList(CheckFirst: Boolean = False);
```

C++

```
__fastcall CreateItemList(Boolean CheckFirst = False);
```

Description

Applications can't call the protected CreateItemList method. It is used internally when TXMLNodeCollection (see page 26196) needs to create a separate object to implement the List (see page 26210) property. A separate implementation object is only needed if this node contains child nodes that are not in the collection. Otherwise, TXMLNodeCollection (see page 26196) can simply use the ChildNodes (see page 26190) property.

CheckFirst specifies whether CreateItemList should check whether the ChildNodes (see page 26190) property currently includes any nodes that are not in the collection. When CheckFirst is true, CreateItemList first checks the ChildNodes (see page 26190) property, and if all nodes are in the collection, it does not create a separate list to store the nodes of the collection. When CheckFirst is false (or when the ChildNodes (see page 26190) property includes nodes that are not in the collection), CreateItemList instantiates a list to hold the nodes in the collection, and adds to it any nodes listed in ChildNodes (see page 26190) that belong in the collection.

See Also

List (see page 26210)

ChildNodes (see page 26190)

195.1.3.2.6 TXMLNodeCollection.Delete Method

Removes one of the repeating child nodes from this node.

Pascal

```
procedure Delete(Index: Integer);
```

C++

```
__fastcall Delete(int Index);
```

Description

Use the TXMLNodeCollection (see page 26196) object's IXMLNodeCollection interface to call the protected Delete method. Delete removes a specified child node from this node.

Index identifies the child to remove. It is the index of the child in the list of repeating child nodes (The List (see page 26210) property, not the ChildNodes (see page 26190) property). The index 0 represents the first repetition, 1 represents the second repetition, and so on.

See Also

Remove (see page 26207)

Delete (see page 26442)

List (see page 26210)

ChildNodes (see page 26190)

195.1.3.2.7 TXMLNodeCollection.GetCount Method

Returns the value of the Count (see page 26209) property.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the protected read implementation of the Count (see page 26209) property. It returns the number of nodes in the List (see page 26210) property.

See Also

Count (see page 26209)

List (see page 26210)

195.1.3.2.8 TXMLNodeCollection.GetList Method

Returns the value of the List (see page 26210) property.

Pascal

```
function GetList: IXMLNodeList; virtual;
```

C++

```
virtual __fastcall IXMLNodeList GetList();
```

Description

GetList is the protected read implementation of the List (see page 26210) property.

See Also

List (see page 26210)

195.1.3.2.9 TXMLNodeCollection.GetNode Method

Returns the interface for one of the repeating child nodes.

Pascal

```
function GetNode(Index: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode GetNode(int Index);
```

Description

Use the TXMLNodeCollection (see page 26196) object's IXMLNodeCollection interface to call the protected GetNode method. GetNode returns an interface for working with one of the repeating child nodes.

Index identifies which repetition of the child node is desired, where 0 is the first repetition, 1 is the second repetition, and so on. Note that Index is the index of the node in the List (see page 26210) property, not in the index in the ChildNodes (see page 26190) property.

See Also

Nodes (see page 26445)

GetCount (see page 26205)

GetNode (see page 26443)

ChildNodes (see page 26190)

195.1.3.2.10 TXMLNodeCollection.InsertInCollection Method

Copies a child node from the ChildNodes (see page 26190) property to the List (see page 26210) property.

Pascal

```
procedure InsertInCollection(Node: IXMLNode; Index: Integer);
```

C++

```
__fastcall InsertInCollection(IXMLNode Node, int Index);
```

Description

Applications can't call this protected method. It is called automatically when nodes are added to the ChildNodes (see page 26190) property list.

Node is the node that was just added to the ChildNodes (see page 26190) property.

Index is the index of Node in the ChildNodes (see page 26190) property.

InsertInCollection first checks whether Node belongs in the collection. If so, it adds it to the List (see page 26210) property in the position that corresponds to Index (that is, in the index of Node when you ignore all the nodes in ChildNodes (see page 26190) that do not belong in the collection.)

See Also

ChildNodes (see page 26190)

List (see page 26210)

195.1.3.2.11 TXMLNodeCollection.IsCollectionItem Method

Indicates whether a specified node represents the type of node that appears as a repeating child of this node.

Pascal

```
function IsCollectionItem(const Node: IXMLNode): Boolean;
```

C++

```
__fastcall Boolean IsCollectionItem(const IXMLNode Node);
```

Description

Applications can't call the protected IsCollectionItem method. TXMLNodeCollection (see page 26196) uses this method internally to distinguish nodes that represent instances of the repeating child node type.

IsCollectionItem checks whether Node implements the interface specified by the ItemInterface (see page 26209) property . If so, it returns true. Otherwise, it returns false.

See Also

List (see page 26210)

ItemInterface (see page 26209)

195.1.3.2.12 TXMLNodeCollection.Remove Method

Removes one of the repeating child nodes from this node.

Pascal

```
function Remove(const AItem: IXMLNode): Integer;
```

C++

```
__fastcall int Remove(const IXMLNode AItem);
```

Description

Use the TXMLNodeCollection (see page 26196) object's IXMLNodeCollection interface to call the protected Remove method. Remove removes a specified child node from this node.

AItem is the interface of the child to remove. It must represent on of the repeating child nodes.

Remove deletes the child node from both the List (see page 26210) property and from the ChildNodes (see page 26190) property. It returns the index that the child node held in the List (see page 26210) property.

To remove a child that is not one of the repeating child nodes, use the ChildNodes (see page 26190) property instead.

See Also

Delete (see page 26205)

List (see page 26210)

ChildNodes (see page 26190)

195.1.3.2.13 TXMLNodeCollection.SetChildNodes Method

Sets the value of the ChildNodes (see page 26190) property.

Pascal

```
procedure SetChildNodes(const Value: IXMLNodeList); override;
```

C++

```
virtual __fastcall SetChildNodes(const IXMLNodeList Value);
```

Description

Applications can't call this protected method. It is called internally the first time an application reads the ChildNodes (see page 26190) property to supply the interface that implements that property.

TXMLNodeCollection (see page 26196) overrides the inherited method to update the List (see page 26210) property as well.

See Also

ChildNodes (see page 26190)

List (see page 26210)

195.1.3.2.14 TXMLNodeCollection.UpdateCollectionList Method

Updates the List (see page 26210) property to reflect changes to the set of child nodes.

Pascal

```
procedure UpdateCollectionList(Operation: TNodeListOperation; var Node: IXMLNode; const
IndexOrName: OleVariant; BeforeOperation: Boolean);
```

C++

```
__fastcall UpdateCollectionList(TNodeListOperation Operation, IXMLNode Node, const OleVariant
IndexOrName, Boolean BeforeOperation);
```

Description

Applications can't call this protected method. This method is called automatically when the ChildNodes (see page 26190) property changes so that the collection can update the List (see page 26210) property to reflect the repeating elements in the current set of child nodes.

Operation indicates whether the change to ChildNodes (see page 26190) is the addition of a new node (nlInsert) or the deletion of a child node (nlRemove).

Node is the interface for a child node that is added or removed.

IndexOrName is the index or tag name of the node that is added or removed.

BeforeOperation indicates whether this method is called before ChildNodes (see page 26190) has changed or after the change has occurred.

See Also

List (see page 26210)

ChildListNotify (see page 26203)

ChildNodes (see page 26190)

OnHostChildNotify (see page 26195)

195.1.3.3 TXMLNodeCollection Properties

The properties of the TXMLNodeCollection class are listed here.

Protected Properties

	Name	Description
R	Count (see page 26209)	Indicates the number of repeating child nodes.
	ItemInterface (see page 26209)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 26210)	This is ItemNS, a member of class TXMLNodeCollection.
	ItemTag (see page 26210)	This is ItemTag, a member of class TXMLNodeCollection.
R	List (see page 26210)	Lists the interfaces for the children that are repeated elements.

195.1.3.3.1 TXMLNodeCollection.Count Property

Indicates the number of repeating child nodes.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use the IXMLNodeCollection interface to read this protected property. Count is the number of child nodes for this node that are repetitions of the repeating child node type. That is, Count is the number of nodes in the List (see page 26210) property.

Use Count to obtain an upper limit when addressing child nodes by index (for example, when iterating over all nodes in using the GetNode (see page 26206) method).

Count is changed when you add child nodes using the AddChild (see page 26156) method or delete them using the Delete (see page 26205) or Remove (see page 26207) method. You can access the individual repeating child nodes using the GetNode (see page 26206) method.

See Also

GetNode (see page 26206)

AddChild (see page 26414)

Remove (see page 26207)

Delete (see page 26205)

List (see page 26210)

195.1.3.3.2 TXMLNodeCollection.ItemInterface Property

Indicates the interface type for the repeating child nodes.

Pascal

```
property ItemInterface: TGuid;
```

C++

```
__property TGuid ItemInterface;
```

Description

Applications can't use this protected property. It is used internally to identify the interface for all child nodes that represent the repeating child element.

ItemInterface is the GUID for the default interface of the class that implements the nodes in the collection.

See Also

IXMLNode (see page 26411)

IsCollectionItem (see page 26207)

List (see page 26210)

itemtag (see page 26210)

itemns (see page 26210)

195.1.3.3.3 TXMLNodeCollection.ItemNS Property

Pascal

```
property ItemNS: DOMString;
```

C++

```
__property DOMString ItemNS;
```

Description

This is ItemNS, a member of class TXMLNodeCollection.

195.1.3.3.4 TXMLNodeCollection.ItemTag Property

Pascal

```
property ItemTag: DOMString;
```

C++

```
__property DOMString ItemTag;
```

Description

This is ItemTag, a member of class TXMLNodeCollection.

195.1.3.3.5 TXMLNodeCollection.List Property

Lists the interfaces for the children that are repeated elements.

Pascal

```
property List: IXMLNodeList;
```

C++

```
__property IXMLNodeList List;
```

Description

Applications can't use the protected List property. TXMLNodeCollection (see page 26196) uses this property internally to identify those child nodes that are repeated elements. Because TXMLNodeCollection (see page 26196) can correspond to a node in the XML document that has additional children that are not instances of the repeating node type, List is kept separate from the ChildNodes (see page 26190) property, which lists all child nodes.

Applications that need to access the child nodes in this list should use the Nodes property of the IXMLNodeCollection interface for this node or the Items property of an IXMLNodeCollection descendant that is generated by the XML Data Binding wizard.

See Also

Nodes (see page 26445)

ChildNodes (see page 26190)

195.1.4 TXMLNodeList Class

TXMLNodeList represents a set of nodes in an XML document.

Class Hierarchy

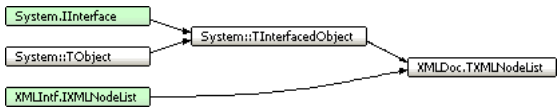

Pascal

```
TXMLNodeList = class(TInterfacedObject, IXMLNodeList);
```

C++

```
class TXMLNodeList : public TInterfacedObject, public IXMLNodeList;
```

File

XMLDoc

Description

TXMLNodeList is used to represent a set of related nodes (IXMLNode interfaces) in an XML document. For example, TXMLNodeList is used to represent all of the children of a node, or all of the attributes of a node, or all of the nodes in a collection of repeating nodes.

TXMLNodeList can be used to add or delete nodes from the collection, or to access specific nodes.

Typically, applications do not use TXMLNodeList directly. Instead, they access it through the IXMLNodeList interface.

See Also

ChildNodes

AttributeNodes

TXMLNodeCollection (see page 26196)

IXMLNodeList (see page 26445)

IXMLNode (see page 26411)

TObjectList (see page 5677)

TInterfaceList (see page 2855)

195.1.4.1 TXMLNodeList Members

The following tables list the members exposed by TXMLNodeList.

Protected Data Members

	Name	Description
	FRefCount (see page 22278)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
R	NotificationProc (see page 26227)	Specifies a callback that TXMLNodeList (see page 26211) calls when nodes are added to or removed from the list, or when new nodes need to be created

Protected Methods

	Name	Description
	_AddRef (see page 22278)	Implements the IInterface (see page 22396) System::TInterfacedObject::_AddRef method.
	_Release (see page 22279)	Implements the IInterface (see page 22396) System::TInterfacedObject::_Release method.
	Add (see page 26215)	Adds a new node to the end of the list.
	BeginUpdate (see page 26215)	Increments the value of UpdateCount (see page 26226).
	Clear (see page 26216)	Removes all nodes from the list.
	Delete (see page 26216)	This is the overview for the Delete method overload.
	DoNotify (see page 26217)	Executes the callback specified by NotificationProc (see page 26227).
	EndUpdate (see page 26218)	Decrements the value of UpdateCount (see page 26226).
	FindNode (see page 26218)	This is the overview for the FindNode method overload.
	FindSibling (see page 26219)	Returns a node that appears a specified amount before or after another node.
	First (see page 26220)	Returns the first node in the list.
	Get (see page 26220)	Returns a specified node in the list.
	GetCount (see page 26220)	Returns the number of nodes in the list.
	GetNode (see page 26221)	Returns a specified node from the list.
	GetUpdateCount (see page 26221)	Returns the value of the UpdateCount (see page 26226) property.
	IndexOf (see page 26222)	This is the overview for the IndexOf method overload.
	Insert (see page 26223)	Inserts a new node into a specified position in the list.
	InternalInsert (see page 26223)	Provides the underlying implementation of the Insert (see page 26223) method
	Last (see page 26223)	Returns the last node in the list.
	QueryInterface (see page 22280)	Implements the IInterface (see page 22396) System::TInterfacedObject::QueryInterface method.
	Remove (see page 26224)	Removes a specified node from the list.
	ReplaceNode (see page 26224)	Replaces a node in the list with another node.

Protected Properties

	Name	Description
R	Count (see page 26225)	Indicates the number of nodes in the collection.
R	DefaultNamespaceURI (see page 26225)	Identifies the default Namespace for the nodes in this collection.
R	List (see page 26226)	Stores the interfaces for every node in the list.
R	Owner (see page 26226)	Specifies the node that uses this TXMLNodeList (see page 26211) object to manage a set of its child or attribute nodes.
R	UpdateCount (see page 26226)	Indicates the number of times an application has called BeginUpdate (see page 26215) without a matching call to EndUpdate (see page 26218).

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22279)	Responds after the last constructor executes.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22279)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.

	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 26213)	Instantiates a TXMLNodeList (see page 26211) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Delete (see page 26448)	This is the overview for the Delete method overload.
V	Destroy (see page 26214)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindNode (see page 26450)	This is the overview for the FindNode method overload.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	IndexOf (see page 26453)	This is the overview for the IndexOf method overload.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22280)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	Nodes (see page 26456)	Provides access to the nodes in the list.
R	RefCount (see page 22281)	Indicates the number of interface pointers currently dependent upon the object.

195.1.4.2 TXMLNodeList.Create Constructor

Instantiates a TXMLNodeList (see page 26211) object.

Pascal

```
constructor Create(Owner: TXMLNode; const DefaultNamespaceURI: DOMString; NotificationProc:
TNodeListNotification);
```

C++

```
__fastcall TXMLNodeList(TXMLNode Owner, const DOMString DefaultNamespaceURI,
TNodeListNotification NotificationProc);
```

Description

Createinstantiates a new TXMLNodeList (see page 26211) object. Typically, TXMLNodeList (see page 26211) is created by another object that represents a node. The TXMLNodeList (see page 26211) represents a set of nodes that are of interest to

the other object that creates it.

AOwner is the node that creates TXMLNodeList (see page 26211) to represent a set of its child nodes or attributes.

DefaultNamespaceURI (see page 26225) identifies the namespace that is used by default for nodes added to this list when no namespace is explicitly specified.

NotificationProc (see page 26227) is a callback that TXMLNodeList (see page 26211) calls when nodes are added to or removed from the list, or when new nodes need to be created.

See Also

DefaultNamespaceURI (see page 26225)

Owner (see page 26226)

NotificationProc (see page 26227)

195.1.4.3 TXMLNodeList.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLNodeList();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 22300)

195.1.4.4 TXMLNodeList Methods

The methods of the TXMLNodeList class are listed here.

Protected Methods

	Name	Description
	Add (see page 26215)	Adds a new node to the end of the list.
	BeginUpdate (see page 26215)	Increments the value of UpdateCount (see page 26226).
	Clear (see page 26216)	Removes all nodes from the list.
	Delete (see page 26216)	This is the overview for the Delete method overload.
	DoNotify (see page 26217)	Executes the callback specified by NotificationProc (see page 26227).
	EndUpdate (see page 26218)	Decrements the value of UpdateCount (see page 26226).
	FindNode (see page 26218)	This is the overview for the FindNode method overload.
	FindSibling (see page 26219)	Returns a node that appears a specified amount before or after another node.
	First (see page 26220)	Returns the first node in the list.
	Get (see page 26220)	Returns a specified node in the list.
	GetCount (see page 26220)	Returns the number of nodes in the list.
	GetNode (see page 26221)	Returns a specified node from the list.
	GetUpdateCount (see page 26221)	Returns the value of the UpdateCount (see page 26226) property.
	IndexOf (see page 26222)	This is the overview for the IndexOf method overload.
	Insert (see page 26223)	Inserts a new node into a specified position in the list.
	InternalInsert (see page 26223)	Provides the underlying implementation of the Insert (see page 26223) method
	Last (see page 26223)	Returns the last node in the list.
	Remove (see page 26224)	Removes a specified node from the list.
	ReplaceNode (see page 26224)	Replaces a node in the list with another node.

195.1.4.4.1 TXMLNodeList.Add Method

Adds a new node to the end of the list.

Pascal

```
function Add(const Node: IXMLNode): Integer;
```

C++

```
__fastcall int Add(const IXMLNode Node);
```

Description

Call Add to add a node to the end of the list. Add returns the index of the node once it is added, where 0 is the index of the first node in the list, 1 is the index of the second node, and so on.

Node is the node to add to the list.

See Also

Delete (see page 26217)

Remove (see page 26224)

IndexOf (see page 26222)

Insert (see page 26223)

List (see page 26226)

195.1.4.4.2 TXMLNodeList.BeginUpdate Method

Increments the value of UpdateCount (see page 26226).

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

Call BeginUpdate before making several changes to the nodes in the list (such as adding or removing several nodes). BeginUpdate increments the UpdateCount (see page 26226) property, to keep track of the fact that the list is in the middle of an update. When the update is complete, call EndUpdate (see page 26218) to decrement the UpdateCount (see page 26226) property.

Every call to BeginUpdate should be matched by a call to EndUpdate (see page 26218), so that the UpdateCount (see page 26226) property accurately reflects whether an update operation is underway.

See Also

UpdateCount (see page 26226)

EndUpdate (see page 26218)

195.1.4.4.3 TXMLNodeList.Clear Method

Removes all nodes from the list.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the list.

Note: Clear does not call the BeginUpdate (see page 26215) and EndUpdate (see page 26218) methods. NotificationProc (see page 26227) is called for every node in the list as it is removed.

See Also

Delete (see page 26217)

Remove (see page 26224)

List (see page 26226)

NotificationProc

195.1.4.4.4 Delete Method

This is the overview for the Delete method overload.

Overload List

	Name	Description
	TXMLNodeList.Delete (DOMString) (see page 26217)	Removes a specified node from the list.
	TXMLNodeList.Delete (DOMString, DOMString) (see page 26217)	Removes a specified node from the list.
	TXMLNodeList.Delete (Integer) (see page 26217)	Removes a specified node from the list.

195.1.4.4.4.1 TXMLNodeList.Delete Method (DOMString)

Removes a specified node from the list.

Pascal

```
function Delete(const Index: Integer): Integer; overload;
function Delete(const Name: DOMString): Integer; overload;
function Delete(const Name: DOMString; const NamespaceURI: DOMString): Integer; overload;
```

C++

```
__fastcall int Delete(const int Index);
__fastcall int Delete(const DOMString Name);
__fastcall int Delete(const DOMString Name, const DOMString NamespaceURI);
```

Description

Delete removes the node specified by the Index or Name parameter.

Name identifies the node to remove from the list. This is the local name of the node to remove.

NamespaceURI identifies the namespace to use when interpreting Name. If the node is identified by Name, but no NamespaceURI parameter is provided, Delete uses DefaultNamespaceURI (see page 26225).

Index identifies the node to remove by index rather than name. Index ranges from 0 to one less than the value of the Count (see page 26225) property.

Delete returns the index of the node that was removed. If there was no node that matched the value of Index or Name, Delete returns –1.

See Also

Remove (see page 26224)

Clear (see page 26216)

IndexOf (see page 26222)

LocalName (see page 26434)

DefaultNamespaceURI (see page 26225)

NamespaceURI (see page 26435)

195.1.4.4.5 TXMLNodeList.DoNotify Method

Executes the callback specified by NotificationProc (see page 26227).

Pascal

```
function DoNotify(Operation: TNodeListOperation; const Node: IXMLNode; const IndexOrName:
OleVariant; BeforeOperation: Boolean): IXMLNode;
```

C++

```
__fastcall IXMLNode DoNotify(TNodeListOperation Operation, const IXMLNode Node, const
OleVariant IndexOrName, Boolean BeforeOperation);
```

Description

DoNotify executes the callback that is assigned to NotificationProc (see page 26227) when the list adds, deletes, or needs to create nodes in the list. The parameters are supplied as parameters to the callback, and DoNotify returns the node that was added, deleted, or created.

Operation indicates what change is about to occur or has just occurred.

Node is the node that is being added or removed from the list, or that acts as a parent if the callback must create a node (when Operation is nlCreateNode).

IndexOrName is the name of the node to be created when Operation is nlCreateNode. Otherwise, it is the name or index of the node being deleted or added.

BeforeOperation indicates whether the call occurs before the indicated operation or after. When nodes are inserted and removed, a call is made first with BeforeOperation set to true, then the operation occurs, and then a second callback is made with BeforeOperation set to false. When nodes must be created, only one call is made, with BeforeOperation set to true.

See Also

NotificationProc (see page 26227)

195.1.4.4.6 TXMLNodeList.EndUpdate Method

Decrements the value of UpdateCount (see page 26226).

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

Call EndUpdate after making several changes to the nodes in the list (such as adding or removing several nodes). EndUpdate decrements the UpdateCount (see page 26226) property, which was incremented by a previous call to BeginUpdate (see page 26215).

Every call to BeginUpdate (see page 26215) should be matched by a call to EndUpdate, so that the UpdateCount (see page 26226) property accurately reflects whether an update operation is underway.

See Also

UpdateCount (see page 26226)

BeginUpdate (see page 26215)

195.1.4.4.7 FindNode Method

This is the overview for the FindNode method overload.

Overload List

	Name	Description
	TXMLNodeList.FindNode (DOMString) (see page 26218)	Returns a specified node from the list.
	TXMLNodeList.FindNode (DOMString, DOMString) (see page 26218)	Returns a specified node from the list.
	TXMLNodeList.FindNode (TGuid) (see page 26218)	Returns a specified node from the list.

195.1.4.4.7.1 TXMLNodeList.FindNode Method (DOMString)

Returns a specified node from the list.

Pascal

```
function FindNode(NodeName: DOMString): IXMLNode; overload;
function FindNode(NodeName: DOMString; NamespaceURI: DOMString): IXMLNode; overload;
function FindNode(ChildNodeType: TGuid): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode FindNode(DOMString NodeName);
__fastcall IXMLNode FindNode(DOMString NodeName, DOMString NamespaceURI);
__fastcall IXMLNode FindNode(TGuid ChildNodeType);
```

Description

Call FindNode to access a particular node in the list.

NodeName is the node to access. It specifies the LocalName property of the desired node.

NamespaceURI identifies the namespace that qualifies NodeName. It is the same as the NamespaceURI property of the desired node. If the first syntax is used (omitting NamespaceURI), FindNode uses the value of the DefaultNamespaceURI (see page 26225) property.

ChildNodeType identifies the node to access by its interface type (GUID), rather than by name. When using ChildNodeType, FindNode returns the first child node it finds that implements the specified interface.

FindNode returns the interface for the node if it is in the list. If NodeName or ChildNodeType does not specify a node in the list, FindNode returns nil (Delphi) or NULL (C++).

See Also

FindSibling (see page 26219)

First (see page 26220)

Last (see page 26223)

IndexOf (see page 26222)

LocalName (see page 26434)

NamespaceURI (see page 26435)

DefaultNamespaceURI (see page 26225)

195.1.4.4.8 TXMLNodeList.FindSibling Method

Returns a node that appears a specified amount before or after another node.

Pascal

```
function FindSibling(const Node: IXMLNode; Delta: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode FindSibling(const IXMLNode Node, int Delta);
```

Description

Call FindSibling to access the node whose position has a specified relationship to another node.

Node is a node in the list to use as a reference point.

Delta indicates where the desired node appears, relative to Node. If Delta is positive, FindSibling returns the node that appears Delta positions after Node. If Delta is negative, FindSibling returns a node that appears before Node.

FindSibling returns the node that appears at the position offset by Delta, relative to the position of Node. If Delta specifies a position before the first node or after the last node in the list, FindSibling returns nil (Delphi) or NULL (C++).

See Also

FindNode (see page 26218)

First (see page 26220)

Last (see page 26223)

IndexOf (see page 26222)

195.1.4.4.9 TXMLNodeList.First Method

Returns the first node in the list.

Pascal

```
function First: IXMLNode;
```

C++

```
__fastcall IXMLNode First();
```

Description

Call First to access the first node in the list. If the list is empty, First raises an exception.

See Also

FindNode (see page 26218)

FindSibling (see page 26219)

Last (see page 26223)

195.1.4.4.10 TXMLNodeList.Get Method

Returns a specified node in the list.

Pascal

```
function Get(Index: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode Get(int Index);
```

Description

Call Get to retrieve a node from the list, given its index.

Index specifies the node to fetch, where 0 identifies the first node, 1 identifies the second node, and so on. Index should be less than the value of the Count (see page 26225) property.

See Also

List (see page 26226)

GetNode (see page 26221)

Count (see page 26225)

195.1.4.4.11 TXMLNodeList.GetCount Method

Returns the number of nodes in the list.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the read implementation of the Count (see page 26225) property.

See Also

Count (see page 26225)

195.1.4.4.12 TXMLNodeList.GetNode Method

Returns a specified node from the list.

Pascal

```
function GetNode(const IndexOrName: OleVariant): IXMLNode;
```

C++

```
__fastcall IXMLNode GetNode(const OleVariant IndexOrName);
```

Description

GetNode returns the interface to a node that is identified by name or by position.

IndexOrName identifies the desired node. It can be

The index of the node, where 0 is the index of the first node, 1 is the index of the second node, and so on.

The LocalName property of a node in the list.

If IndexOrName does not identify a node in the list, GetNode tries to create a new node with the name specified by IndexOrName. If it can't create the new node, GetNode raises an exception.

See Also

FindNode (see page 26218)

List (see page 26226)

LocalName (see page 26434)

Get (see page 26220)

195.1.4.4.13 TXMLNodeList.GetUpdateCount Method

Returns the value of the UpdateCount (see page 26226) property.

Pascal

```
function GetUpdateCount: Integer;
```

C++

```
__fastcall int GetUpdateCount();
```

Description

GetUpdateCount is the read implementation of the UpdateCount (see page 26226) property.

See Also

UpdateCount (see page 26226)

195.1.4.4.14 IndexOf Method

This is the overview for the IndexOf method overload.

Overload List

	Name	Description
	TXMLNodeList.IndexOf (DOMString) (see page 26222)	Returns the index of a specified node.
	TXMLNodeList.IndexOf (DOMString, DOMString) (see page 26222)	Returns the index of a specified node.
	TXMLNodeList.IndexOf (IXMLNode) (see page 26222)	Returns the index of a specified node.

195.1.4.4.14.1 TXMLNodeList.IndexOf Method (DOMString)

Returns the index of a specified node.

Pascal

```
function IndexOf(const Node: IXMLNode): Integer; overload;
function IndexOf(const Name: DOMString): Integer; overload;
function IndexOf(const Name: DOMString; const NamespaceURI: DOMString): Integer; overload;
```

C++

```
__fastcall int IndexOf(const IXMLNode Node);
__fastcall int IndexOf(const DOMString Name);
__fastcall int IndexOf(const DOMString Name, const DOMString NamespaceURI);
```

Description

Call IndexOf to locate a node in the list. You can specify which node to locate using its name or its interface.

Name is the LocalName property of the node to locate.

NamespaceURI identifies the namespace to use when interpreting LocalName. If Name is used without a NamespaceURI parameter (the first syntax), IndexOf uses the value of the DefaultNamespaceURI (see page 26225) property.

Node is the interface of the node to locate.

IndexOf returns the index of the specified node, where 0 is the index of the first node, 1 is the index of the second node, and so on. If the specified node is not in the list, IndexOf returns -1.

See Also

FindNode (see page 26218)

LocalName (see page 26434)

NamespaceURI (see page 26435)

DefaultNamespaceURI (see page 26225)

195.1.4.4.15 TXMLNodeList.Insert Method

Inserts a new node into a specified position in the list.

Pascal

```
procedure Insert(Index: Integer; const Node: IXMLNode);
```

C++

```
__fastcall Insert(int Index, const IXMLNode Node);
```

Description

Call Insert to add a node at the position specified by Index.

Index specifies where to insert the node, where 0 is the first position, 1 is second position, and so on. If Index does not specify a valid index, Insert raises an exception.

Node is the node to add to the list.

See Also

Delete (see page 26217)

Remove (see page 26224)

IndexOf (see page 26222)

Add (see page 26215)

195.1.4.4.16 TXMLNodeList.InternalInsert Method

Provides the underlying implementation of the Insert (see page 26223) method

Pascal

```
function InternalInsert(Index: Integer; const Node: IXMLNode): Integer;
```

C++

```
__fastcall int InternalInsert(int Index, const IXMLNode Node);
```

Description

The Insert (see page 26223) method calls InternalInsert to add a node at the position specified by Index.

Index specifies where to insert the node, where 0 is the first position, 1 is second position, and so on. If Index is –1, the node is added to the end of the list.

Node is the node to add to the list.

InternalInsert returns the index of the node after it is inserted. This is the same as the Index parameter when Index is not –1.

See Also

Insert (see page 26223)

195.1.4.4.17 TXMLNodeList.Last Method

Returns the last node in the list.

Pascal

```
function Last: IXMLNode;
```

C++

```
__fastcall IXMLNode Last();
```

Description

Call Last to access the last node in the list. If the list is empty, Last raises an exception.

See Also

FindNode (see page 26218)

FindSibling (see page 26219)

First (see page 26220)

195.1.4.4.18 TXMLNodeList.Remove Method

Removes a specified node from the list.

Pascal

```
function Remove(const Node: IXMLNode): Integer;
```

C++

```
__fastcall int Remove(const IXMLNode Node);
```

Description

Remove removes the specified node from the list.

Node is the node to remove from the list.

Remove returns the index of Node before it was removed. If node is not a node in the list, Remove returns -1.

See Also

Delete (see page 26217)

Clear (see page 26216)

195.1.4.4.19 TXMLNodeList.ReplaceNode Method

Replaces a node in the list with another node.

Pascal

```
function ReplaceNode(const OldNode: IXMLNode; const NewNode: IXMLNode): IXMLNode;
```

C++

```
__fastcall IXMLNode ReplaceNode(const IXMLNode OldNode, const IXMLNode NewNode);
```

Description

Call ReplaceNode to replace the node specified by OldNode with the node specified by NewNode.

OldNode is the node to replace. If OldNode does not appear in the list, then ReplaceNode adds the new node to the end of the list.

NewNode is the node to add to the list in place of OldNode.

ReplaceNode returns OldNode (even if OldNode did not appear in the list).

See Also

List (see page 26226)

195.1.4.5 TXMLNodeList Properties

The properties of the TXMLNodeList class are listed here.

Protected Properties

	Name	Description
R	Count (see page 26225)	Indicates the number of nodes in the collection.
R	DefaultNamespaceURI (see page 26225)	Identifies the default Namespace for the nodes in this collection.
R	List (see page 26226)	Stores the interfaces for every node in the list.
R	Owner (see page 26226)	Specifies the node that uses this TXMLNodeList (see page 26211) object to manage a set of its child or attribute nodes.
R	UpdateCount (see page 26226)	Indicates the number of times an application has called BeginUpdate (see page 26215) without a matching call to EndUpdate (see page 26218).

195.1.4.5.1 TXMLNodeList.Count Property

Indicates the number of nodes in the collection.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read Count to determine the number of nodes in the list. To change the value of Count, use the IXMLNodeList interface. Add (see page 26215) nodes using the Add (see page 26215) or Insert (see page 26223) method, or remove them using the Delete (see page 26217) or Remove (see page 26224) method.

See Also

Nodes (see page 26456)

Add (see page 26447)

Delete (see page 26449)

Remove (see page 26455)

Insert (see page 26454)

195.1.4.5.2 TXMLNodeList.DefaultNamespaceURI Property

Identifies the default Namespace for the nodes in this collection.

Pascal

```
property DefaultNamespaceURI: DOMString;
```

C++

```
__property DOMString DefaultNamespaceURI;
```

Description

Read DefaultNamespaceURI to determine the namespace that is used by default when locating nodes in the list. The IndexOf (see page 26222), FindNode (see page 26218), and Delete (see page 26217) methods all use this namespace when trying to match the name of a node if a namespace is not explicitly supplied.

The value of DefaultNamespaceURI is set by the TXMLNodeList (see page 26211) constructor.

See Also

NamespaceURI (see page 26435)

Create (see page 26213)

IndexOf (see page 26222)

FindNode (see page 26218)

Delete (see page 26217)

195.1.4.5.3 TXMLNodeList.List Property

Stores the interfaces for every node in the list.

Pascal

```
property List: IInterfaceList;
```

C++

```
__property IInterfaceList List;
```

Description

TXMLNodeList (see page 26211) uses List internally to store and manage the interfaces to the nodes it represents.

See Also

Count (see page 26225)

Nodes (see page 26456)

195.1.4.5.4 TXMLNodeList.Owner Property

Specifies the node that uses this TXMLNodeList (see page 26211) object to manage a set of its child or attribute nodes.

Pascal

```
property Owner: TXMLNode;
```

C++

```
__property TXMLNode Owner;
```

Description

Owner designates the node that instantiates TXMLNodeList (see page 26211) to manage a set of its child or attribute nodes. This value is supplied to the TXMLNodeList (see page 26211) constructor.

See Also

ChildNodes (see page 26190)

Create (see page 26213)

AttributeNodes (see page 26189)

195.1.4.5.5 TXMLNodeList.UpdateCount Property

Indicates the number of times an application has called BeginUpdate (see page 26215) without a matching call to EndUpdate (see page 26218).

Pascal

```
property UpdateCount: Integer;
```

C++

```
__property int UpdateCount;
```

Description

UpdateCount keeps track of whether all update operations have been completed. When an application starts making changes to the list, it calls BeginUpdate (see page 26215), which increments UpdateCount. When those changes are complete, it calls EndUpdate (see page 26218), which decrements UpdateCount. Thus, UpdateCount allows calls to BeginUpdate (see page 26215) and EndUpdate (see page 26218) to be nested.

Applications can avoid responding to updates in a piecemeal fashion by checking UpdateCount and only responding when it is 0. For example, the callback that is supplied to the TXMLNodeList (see page 26211) constructor typically checks UpdateCount before responding to change notifications.

See Also

BeginUpdate (see page 26215)

EndUpdate (see page 26218)

Create (see page 26213)

195.1.4.6 TXMLNodeList Events

The events of the TXMLNodeList class are listed here.

Protected Events

	Name	Description
R	NotificationProc (see page 26227)	Specifies a callback that TXMLNodeList (see page 26211) calls when nodes are added to or removed from the list, or when new nodes need to be created

195.1.4.6.1 TXMLNodeList.NotificationProc Event

Specifies a callback that TXMLNodeList (see page 26211) calls when nodes are added to or removed from the list, or when new nodes need to be created

Pascal

```
property NotificationProc: TNodeListNotification;
```

C++

```
__property TNodeListNotification NotificationProc;
```

Description

The node that instantiates TXMLNodeList (see page 26211) to manage a set of its child or attribute nodes supplies the value of this callback to the TXMLNodeList (see page 26211) constructor. TXMLNodeList (see page 26211) then calls this callback when its methods are used to add, create, or remove nodes.

See Also

ChildListNotify (see page 26159)

Create (see page 26213)

AttributeListNotify (see page 26158)

195.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IXMLDocumentAccess (see page 26228)	This is class XMLDoc.IXMLDocumentAccess.
	IXMLNodeAccess (see page 26230)	This is class XMLDoc.IXMLNodeAccess.

195.2.1 IXMLDocumentAccess Interface

Class Hierarchy

XMLDoc.IXMLDocumentAccess

Pascal

```
IXMLDocumentAccess = interface;
```

C++

```
__interface IXMLDocumentAccess;
```

File

XMLDoc

Description

This is class XMLDoc.IXMLDocumentAccess.

195.2.1.1 IXMLDocumentAccess Members

The following tables list the members exposed by IXMLDocumentAccess.

Public Methods

	Name	Description
	GetDocumentObject (see page 26229)	This is GetDocumentObject, a member of class IXMLDocumentAccess.
	GetDOMPersist (see page 26229)	This is GetDOMPersist, a member of class IXMLDocumentAccess.

Public Properties

	Name	Description
R	DocumentObject (see page 26229)	This is DocumentObject, a member of class IXMLDocumentAccess.
R	DOMPersist (see page 26230)	This is DOMPersist, a member of class IXMLDocumentAccess.

195.2.1.2 IXMLDocumentAccess Methods

The methods of the IXMLDocumentAccess class are listed here.

Public Methods

	Name	Description
	GetDocumentObject (see page 26229)	This is GetDocumentObject, a member of class IXMLDocumentAccess.
	GetDOMPersist (see page 26229)	This is GetDOMPersist, a member of class IXMLDocumentAccess.

195.2.1.2.1 IXMLDocumentAccess.GetDocumentObject Method

Pascal

```
function GetDocumentObject: TXMLDocument;
```

C++

```
__fastcall TXMLDocument GetDocumentObject();
```

Description

This is GetDocumentObject, a member of class IXMLDocumentAccess.

195.2.1.2.2 IXMLDocumentAccess.GetDOMPersist Method

Pascal

```
function GetDOMPersist: IDOMPersist;
```

C++

```
__fastcall IDOMPersist GetDOMPersist();
```

Description

This is GetDOMPersist, a member of class IXMLDocumentAccess.

195.2.1.3 IXMLDocumentAccess Properties

The properties of the IXMLDocumentAccess class are listed here.

Public Properties

	Name	Description
R	DocumentObject (see page 26229)	This is DocumentObject, a member of class IXMLDocumentAccess.
R	DOMPersist (see page 26230)	This is DOMPersist, a member of class IXMLDocumentAccess.

195.2.1.3.1 IXMLDocumentAccess.DocumentObject Property

Pascal

```
property DocumentObject: TXMLDocument;
```

C++

```
__property TXMLDocument DocumentObject;
```

Description

This is DocumentObject, a member of class IXMLDocumentAccess.

195.2.1.3.2 IXMLDocumentAccess.DOMPersist Property

Pascal

```
property DOMPersist: IDOMPersist;
```

C++

```
__property IDOMPersist DOMPersist;
```

Description

This is DOMPersist, a member of class IXMLDocumentAccess.

195.2.2 IXMLNodeAccess Interface

Class Hierarchy

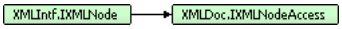

Pascal

```
IXMLNodeAccess = interface(IXMLNode);
```

C++

```
__interface IXMLNodeAccess : public IXMLNode;
```

File

XMLDoc

Description

This is class XMLDoc.IXMLNodeAccess.

195.2.2.1 IXMLNodeAccess Members

The following tables list the members exposed by IXMLNodeAccess.

Public Methods

	Name	Description
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.

	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.

R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

195.2.2.2 IXMLNodeAccess Methods

The methods of the IXMLNodeAccess class are listed here.

Public Methods

	Name	Description
	AddChild (see page 26232)	This is AddChild, a member of class IXMLNodeAccess.
	CheckTextNode (see page 26233)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClearDocumentRef (see page 26233)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CreateAttributeNode (see page 26233)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 26233)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 26233)	This is CreateCollection, a member of class IXMLNodeAccess.
	DOMElement (see page 26234)	This is DOMElement, a member of class IXMLNodeAccess.
	FindHostedNode (see page 26234)	This is FindHostedNode, a member of class IXMLNodeAccess.
	GetChildNodeClasses (see page 26234)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetHostNode (see page 26234)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetNodeObject (see page 26235)	This is GetNodeObject, a member of class IXMLNodeAccess.
	HasChildNode (see page 26235)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 26235)	This is InternalAddChild, a member of class IXMLNodeAccess.
	NestingLevel (see page 26236)	This is NestingLevel, a member of class IXMLNodeAccess.
	RegisterChildNode (see page 26236)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 26236)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	SetCollection (see page 26236)	This is SetCollection, a member of class IXMLNodeAccess.
	SetParentNode (see page 26237)	This is SetParentNode, a member of class IXMLNodeAccess.

195.2.2.2.1 IXMLNodeAccess.AddChild Method

Pascal

```
function AddChild(const TagName: DOMString; const NamespaceURI: DOMString; NodeClass:
TXMLNodeClass; Index: Integer = -1): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode AddChild(const DOMString TagName, const DOMString NamespaceURI,
TXMLNodeClass NodeClass, int Index = -1);
```

Description

This is AddChild, a member of class IXMLNodeAccess.

195.2.2.2.2 IXMLNodeAccess.CheckTextNode Method

Pascal

```
procedure CheckTextNode;
```

C++

```
__fastcall CheckTextNode();
```

Description

This is CheckTextNode, a member of class IXMLNodeAccess.

195.2.2.2.3 IXMLNodeAccess.ClearDocumentRef Method

Pascal

```
procedure ClearDocumentRef;
```

C++

```
__fastcall ClearDocumentRef();
```

Description

This is ClearDocumentRef, a member of class IXMLNodeAccess.

195.2.2.2.4 IXMLNodeAccess.CreateAttributeNode Method

Pascal

```
function CreateAttributeNode(const ADOMNode: IDOMNode): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateAttributeNode(const IDOMNode ADOMNode);
```

Description

This is CreateAttributeNode, a member of class IXMLNodeAccess.

195.2.2.2.5 IXMLNodeAccess.CreateChildNode Method

Pascal

```
function CreateChildNode(const ADOMNode: IDOMNode): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateChildNode(const IDOMNode ADOMNode);
```

Description

This is CreateChildNode, a member of class IXMLNodeAccess.

195.2.2.2.6 IXMLNodeAccess.CreateCollection Method

Pascal

```
function CreateCollection(const CollectionClass: TXMLNodeCollectionClass; const ItemIterface:
TGuid; const ItemTag: DOMString; ItemNS: DOMString = ''): TXMLNodeCollection;
```

C++

```
__fastcall TXMLNodeCollection CreateCollection(const TXMLNodeCollectionClass CollectionClass,
const TGuid ItemIterface, const DOMString ItemTag, DOMString ItemNS = '');
```

Description

This is CreateCollection, a member of class IXMLNodeAccess.

195.2.2.2.7 IXMLNodeAccess.DOMElement Method

Pascal

```
function DOMElement: IDOMElement;
```

C++

```
__fastcall IDOMElement DOMElement();
```

Description

This is DOMElement, a member of class IXMLNodeAccess.

195.2.2.2.8 IXMLNodeAccess.FindHostedNode Method

Pascal

```
function FindHostedNode(const NodeClass: TXMLNodeClass): IXMLNode;
```

C++

```
__fastcall IXMLNode FindHostedNode(const TXMLNodeClass NodeClass);
```

Description

This is FindHostedNode, a member of class IXMLNodeAccess.

195.2.2.2.9 IXMLNodeAccess.GetChildNodeClasses Method

Pascal

```
function GetChildNodeClasses: TNodeClassArray;
```

C++

```
__fastcall TNodeClassArray GetChildNodeClasses();
```

Description

This is GetChildNodeClasses, a member of class IXMLNodeAccess.

195.2.2.2.10 IXMLNodeAccess.GetHostNode Method

Pascal

```
function GetHostNode: TXMLNode;
```

C++

```
__fastcall TXMLNode GetHostNode();
```

Description

This is GetHostNode, a member of class IXMLNodeAccess.

195.2.2.2.11 IXMLNodeAccess.GetNodeObject Method

Pascal

```
function GetNodeObject: TXMLNode;
```

C++

```
__fastcall TXMLNode GetNodeObject();
```

Description

This is GetNodeObject, a member of class IXMLNodeAccess.

195.2.2.2.12 HasChildNode Method

This is the overview for the HasChildNode method overload.

Overload List

	Name	Description
	IXMLNodeAccess.HasChildNode (DOMString) (see page 26235)	This is HasChildNode, a member of class IXMLNodeAccess.
	IXMLNodeAccess.HasChildNode (DOMString, DOMString) (see page 26235)	This is HasChildNode, a member of class IXMLNodeAccess.

195.2.2.2.12.1 IXMLNodeAccess.HasChildNode Method (DOMString)

Pascal

```
function HasChildNode(const ChildTag: DOMString): Boolean; overload;
```

C++

```
__fastcall Boolean HasChildNode(const DOMString ChildTag);
```

Description

This is HasChildNode, a member of class IXMLNodeAccess.

195.2.2.2.12.2 IXMLNodeAccess.HasChildNode Method (DOMString, DOMString)

Pascal

```
function HasChildNode(const ChildTag: DOMString; const NamespaceURI: DOMString): Boolean;
overload;
```

C++

```
__fastcall Boolean HasChildNode(const DOMString ChildTag, const DOMString NamespaceURI);
```

Description

This is HasChildNode, a member of class IXMLNodeAccess.

195.2.2.2.13 IXMLNodeAccess.InternalAddChild Method

Pascal

```
function InternalAddChild(NodeClass: TXMLNodeClass; const NodeName: DOMString; const
NamespaceURI: DOMString; Index: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode InternalAddChild(TXMLNodeClass NodeClass, const DOMString NodeName, const
DOMString NamespaceURI, int Index);
```

Description

This is InternalAddChild, a member of class IXMLNodeAccess.

195.2.2.2.14 IXMLNodeAccess.NestingLevel Method

Pascal

```
function NestingLevel: Integer;
```

C++

```
__fastcall int NestingLevel();
```

Description

This is NestingLevel, a member of class IXMLNodeAccess.

195.2.2.2.15 IXMLNodeAccess.RegisterChildNode Method

Pascal

```
procedure RegisterChildNode(const TagName: DOMString; ChildNodeClass: TXMLNodeClass;
NamespaceURI: DOMString = '');
```

C++

```
__fastcall RegisterChildNode(const DOMString TagName, TXMLNodeClass ChildNodeClass, DOMString
NamespaceURI = '');
```

Description

This is RegisterChildNode, a member of class IXMLNodeAccess.

195.2.2.2.16 IXMLNodeAccess.RegisterChildNodes Method

Pascal

```
procedure RegisterChildNodes(const TagNames: array of DOMString; const NodeClasses: array of
TXMLNodeClass);
```

C++

```
__fastcall RegisterChildNodes(const array of DOMString TagNames, const array of TXMLNodeClass
NodeClasses);
```

Description

This is RegisterChildNodes, a member of class IXMLNodeAccess.

195.2.2.2.17 IXMLNodeAccess.SetCollection Method

Pascal

```
procedure SetCollection(const Value: TXMLNodeCollection);
```

C++

```
__fastcall SetCollection(const TXMLNodeCollection Value);
```

Description

This is SetCollection, a member of class IXMLNodeAccess.

195.2.2.2.18 IXMLNodeAccess.SetParentNode Method

Pascal

```
procedure SetParentNode(const Value: TXMLNode);
```

C++

```
__fastcall SetParentNode(const TXMLNode Value);
```

Description

This is SetParentNode, a member of class IXMLNodeAccess.

195.2.2.3 IXMLNodeAccess Properties

The properties of the IXMLNodeAccess class are listed here.

Public Properties

	Name	Description
R	ChildNodeClasses (see page 26237)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	HostNode (see page 26237)	This is HostNode, a member of class IXMLNodeAccess.

195.2.2.3.1 IXMLNodeAccess.ChildNodeClasses Property

Pascal

```
property ChildNodeClasses: TNodeClassArray;
```

C++

```
__property TNodeClassArray ChildNodeClasses;
```

Description

This is ChildNodeClasses, a member of class IXMLNodeAccess.

195.2.2.3.2 IXMLNodeAccess.HostNode Property

Pascal

```
property HostNode: TXMLNode;
```

C++

```
__property TXMLNode HostNode;
```

Description

This is HostNode, a member of class IXMLNodeAccess.

195.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CheckEncoding (see page 26238)	This is function XMLDoc.CheckEncoding.
	CreateDOMNode (see page 26238)	Creates a new DOM node and returns its interface.
	DetectCharEncoding (see page 26239)	This is function XMLDoc.DetectCharEncoding.
	FormatXMLData (see page 26240)	Formats a string of XML so that it is more readable.
	LoadXMLData (see page 26240)	Creates and returns the interface of an XML document object, given a string of XML.
	LoadXMLData (see page 26240)	Creates and returns the interface of an XML document object, given a string of XML.
	LoadXMLDocument (see page 26241)	Returns the interface for an XML document given the name of an XML file.
	NewXMLDocument (see page 26241)	Returns the interface for a new, empty XML document.
	XMLDocError (see page 26242)	Raises an EXMLDocError exception.
	XMLDocError (see page 26242)	Raises an EXMLDocError exception.
	XMLStringToWideString (see page 26242)	This is function XMLDoc.XMLStringToWideString.

195.3.1 XMLDoc.CheckEncoding Function

Pascal

```
procedure CheckEncoding(var XMLData: DOMString; const ValidEncodings: array of string);
```

C++

```
CheckEncoding(DOMString XMLData, const array of string ValidEncodings);
```

File

XMLDoc

Description

This is function XMLDoc.CheckEncoding.

195.3.2 XMLDoc.CreateDOMNode Function

Creates a new DOM node and returns its interface.

Pascal

```
function CreateDOMNode(Doc: IDOMDocument; const NameOrData: DOMString; NodeType: TNodeType =
ntElement; const AddlData: DOMString = ''): IDOMNode;
```

C++

```
IDOMNode CreateDOMNode(IDOMDocument Doc, const DOMString NameOrData, TNodeType NodeType =
ntElement, const DOMString AddlData = '');
```

File

XMLDoc

Description

CreateDOMNode generates a new DOM node.

Doc is the DOM interface for the document to which the new node will belong.

NameOrData provides the tag name or value of the newly created node. Its interpretation depends on the type of node created, as indicated in the following table:

NodeType	NameOrData
ntElement	The tag name.
ntAttribute	The attribute name.
ntText	The value of the node.
ntCData	The value of the CDATA section.
ntEntityRef	The name of the referenced entity.
ntProcessingInstr	The target of the processing instruction.
ntComment	The value (text) of the comment.
ntDocFragment	Not used.

NodeType indicates the type of node to create. It can only be one of the types listed in the previous table.

The meaning of AddlData depends on the node type, as indicated in the following table:

NodeType	AddlData
ntElement	The namespace URI that qualifies the tag name.
ntAttribute	The namespace URI that qualifies the attribute name.
ntProcessingInstr	The content of the processing instruction, except for the target.

CreateDOMNode returns the interface for the newly created DOM node.

195.3.3 XMLDoc.DetectCharEncoding Function

Pascal

```
function DetectCharEncoding(S: TStream): TXmlEncodingType;
```

C++

```
TXmlEncodingType DetectCharEncoding(TStream S);
```

File

XMLDoc

Description

This is function XMLDoc.DetectCharEncoding.

195.3.4 XMLDoc.FormatXMLData Function

Formats a string of XML so that it is more readable.

Pascal

```
function FormatXMLData(const XMLData: DOMString): DOMString;
```

C++

```
DOMString FormatXMLData(const DOMString XMLData);
```

File

XMLDoc

Description

Use FormatXMLData to convert a string of XML into a format that represents its structure. FormatXMLData changes the input string (XMLData) so that each element node appears on its own line, indented appropriately to reflect its nesting in the node hierarchy.

195.3.5 XMLDoc.LoadXMLData Function

Creates and returns the interface of an XML document object, given a string of XML.

Pascal

```
function LoadXMLData(const XMLData: DOMString): IXMLDocument; overload;
function LoadXMLData(const XMLData: string): IXMLDocument; overload;
```

C++

```
IXMLDocument LoadXMLData(const DOMString XMLData);
IXMLDocument LoadXMLData(const AnsiString XMLData);
```

File

XMLDoc

Description

Call LoadXMLData to create an XML document that represents a specified XML string and obtain its interface.

XMLData is the XML for which you want an IXMLDocument interface.

LoadXMLData returns an interface for the parsed value of XMLData.

LoadXMLData creates a new TXMLDocument (see page 26097) object to parse the XML and returns its IXMLDocument. The TXMLDocument (see page 26097) component is not given an owner, which means that it behaves like an interfaced object. When the application releases the interface that LoadXMLData returns, the TXMLDocument (see page 26097) instance is automatically freed.

See Also

TXMLDocument (see page 26097)

NewXMLDocument (see page 26241)

LoadXMLDocument (see page 26241)

TXMLDocument (see page 26097)

195.3.6 XMLDoc.LoadXMLDocument Function

Returns the interface for an XML document given the name of an XML file.

Pascal

```
function LoadXMLDocument(const FileName: DOMString): IXMLDocument;
```

C++

```
IXMLDocument LoadXMLDocument(const DOMString FileName);
```

File

XMLDoc

Description

Call LoadXMLDocument to load an XML document from a file on disk and obtain its interface. LoadXMLDocument creates a new TXMLDocument (see page 26097) object and uses it to parse the file specified by the FileName parameter. The TXMLDocument (see page 26097) component is not given an owner, which means that it behaves like an interfaced object. When the application releases the interface that LoadXMLDocument returns, the TXMLDocument (see page 26097) instance is automatically freed.

See Also

TXMLDocument (see page 26097)

NewXMLDocument (see page 26241)

LoadXMLData (see page 26240)

195.3.7 XMLDoc.NewXMLDocument Function

Returns the interface for a new, empty XML document.

Pascal

```
function NewXMLDocument(Version: DOMString = '1.0'): IXMLDocument;
```

C++

```
IXMLDocument NewXMLDocument(DOMString Version = '1.0');
```

File

XMLDoc

Description

Call NewXMLDocument to create a TXMLDocument (see page 26097) object for a new, empty XML document.

Version specifies the XML version that the document should use. It becomes the value of the new document's Version property.

See Also

TXMLDocument (see page 26097)

LoadXMLDocument (see page 26241)

IXMLDocument

195.3.8 XMLDoc.XMLDocError Function

Raises an EXMLDocError exception.

Pascal

```
procedure XMLDocError(const Msg: string); overload;
procedure XMLDocError(const Msg: string; const Args: array of const); overload;
```

C++

```
XMLDocError(const AnsiString Msg);
XMLDocError(const AnsiString Msg, const array of const Args);
```

File

XMLDoc

Description

Call XMLDocError to raise an EXMLDocError exception. Using XMLDocError rather than explicitly creating the EXMLDocError instance can result in more space-efficient code.

Msg is the error message associated with the EXMLDocError instance.

Args supplies arguments for any format specifiers embedded in Msg.

Note: Args_Size is the index of the last entry in the Args array. That is one less than the total number of arguments.

See Also

EXMLDocError

195.3.9 XMLDoc.XMLStringToWideString Function

Pascal

```
function XMLStringToWideString(const XMLString: string): WideString;
```

C++

```
BSTR XMLStringToWideString(const AnsiString XMLString);
```

File

XMLDoc

Description

This is function XMLDoc.XMLStringToWideString.

195.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TNodeChange (see page 26243)	TNodeChange indicates the type of change made to a node in an XML document.
	TNodeListOperation (see page 26244)	TNodeListOperation indicates what type of change is about to occur using an IXMLNodeList interface.
	TXMLDocumentSource (see page 26245)	This is record XMLDoc.TXMLDocumentSource.
	TXMLPrologItem (see page 26246)	This is record XMLDoc.TXMLPrologItem.

Records

	Name	Description
	TNodeClassInfo (see page 26244)	TNodeClassInfo and TNodeClassArray (see page 26247) represent information about the classes that implement XML node interfaces.

195.4.1 XMLDoc.TNodeChange Enumeration

TNodeChange indicates the type of change made to a node in an XML document.

Pascal

```
TNodeChange = (
  ncUpdateValue,
  ncInsertChild,
  ncRemoveChild,
  ncAddAttribute,
  ncRemoveAttribute
);
```

C++

```
enum TNodeChange {
  ncUpdateValue,
  ncInsertChild,
  ncRemoveChild,
  ncAddAttribute,
  ncRemoveAttribute
};
```

File

XMLDoc

Description

The following table lists the possible values:

Value	Meaning
ncUpdateValue	The value of the node is changed.
ncInsertChild	A new child node is added to the node.
ncRemoveChild	A child node is removed from the node.

ncAddAttribute	A new attribute is added to the node.
ncRemoveAttribute	An attribute is removed from the node.

See Also

TNodeChangeEvent

195.4.2 XMLDoc.TNodeClassInfo Record

TNodeClassInfo and TNodeClassArray (see page 26247) represent information about the classes that implement XML node interfaces.

Pascal

```
TNodeClassInfo = record
  NodeName: DOMString;
  NamespaceURI: DOMString;
  NodeClass: TXMLNodeClass;
end;
```

C++

```
struct TNodeClassInfo {
  DOMString NodeName;
  DOMString NamespaceURI;
  TXMLNodeClass NodeClass;
};
```

File

XMLDoc

Description

TNodeClassArray (see page 26247) is a dynamic array of TNodeClassInfo values. Each element of the array describes the implementation of a node.

TNodeClassInfo describes the name and implementation class for a node in an XML document. It contains the following fields:

Field	Meaning
NodeName	The name of the node. This string describes the identifier or use of the node. For example, on element nodes, this is the tag name. On attribute nodes, this is the name of the attribute. On comment nodes, this is the string "#comment". This is the value returned by a node's NodeName property, and is used as a property name in classes generated by the XML Data Binding Wizard.
NamespaceURI	The namespace of the node's name. This string is a Uniform Resource Identifier that identifies the namespace that contains NodeName. It is used to interpret the value of NodeName.
NodeClass	The class of a TXMLNode (see page 26147) descendant that represents the node.

195.4.3 XMLDoc.TNodeListOperation Enumeration

TNodeListOperation indicates what type of change is about to occur using an IXMLNodeList interface.

Pascal

```
TNodeListOperation = (
  nlInsert,
  nlRemove,
  nlCreateNode
);
```

C++

```
enum TNodeListOperation {
  nlInsert,
  nlRemove,
  nlCreateNode
};
```

File

XMLDoc

Description

When an application makes a change using the IXMLNodeList interface, the implementing object sends its owner a notification of that change. TNodeListOperation identifies the type of change that is about to occur. The following table lists the possible values:

Value	Meaning
nlInsert	An existing node is about to be added to the list.
nlRemove	A node is about to be removed from the list.
nlCreateNode	A new node needs to be created so that it can be added to the list.

195.4.4 XMLDoc.TXMLDocumentSource Enumeration

Pascal

```
TXMLDocumentSource = (
  xdsNone,
  xdsXMLProperty,
  xdsXMLData,
  xdsFile,
  xdsStream
);
```

C++

```
enum TXMLDocumentSource {
  xdsNone,
  xdsXMLProperty,
  xdsXMLData,
  xdsFile,
  xdsStream
};
```

File

XMLDoc

Description

This is record XMLDoc.TXMLDocumentSource.

195.4.5 XMLDoc.TXMLPrologItem Enumeration

Pascal

```
TXMLPrologItem = (
  xpVersion,
  xpEncoding,
  xpStandalone
);
```

C++

```
enum TXMLPrologItem {
  xpVersion,
  xpEncoding,
  xpStandalone
};
```

File

XMLDoc

Description

This is record XMLDoc.TXMLPrologItem.

195.5 Types

The following table lists types in this documentation.

Types

Name	Description
TNodeChangeEvent (see page 26247)	TNodeChangeEvent is the type for event handlers that respond when XML nodes are changed.
TNodeClassArray (see page 26247)	This is type XMLDoc.TNodeClassArray.
TNodeListNotification (see page 26248)	TNodeListNotification is the type of the callback the TXMLNodeList (see page 26211) calls when it adds, deletes, or creates XML nodes.
TXMLNodeArray (see page 26249)	TXMLNodeArray is a dynamic array of TXMLNode (see page 26147) objects.
TXMLNodeClass (see page 26249)	TXMLNodeClass is a class reference to a TXMLNode (see page 26147) descendant.
TXMLNodeCollectionClass (see page 26250)	TXMLNodeCollectionClass is a class reference to a TXMLNodeCollection (see page 26196) descendant.

195.5.1 XMLDoc.TNodeChangeEvent Type

TNodeChangeEvent is the type for event handlers that respond when XML nodes are changed.

Pascal

```
TNodeChangeEvent = procedure (const Node: IXMLNode; ChangeType: TNodeChange) of object;
```

C++

```
(const Node: IXMLNode; ChangeType: TNodeChange) ( TNodeChangeEvent)();
```

File

XMLDoc

Description

TNodeChangeEvent is the type for the BeforeNodeChange and AfterNodeChange event handlers. These events occur before and after changes are made to the nodes in an XML document.

Node identifies the node to which the change occurs.

ChangeType indicates the type of change made to Node.

See Also

BeforeNodeChange

AfterNodeChange

195.5.2 XMLDoc.TNodeClassArray Type

Pascal

```
TNodeClassArray = array of TNodeClassInfo;
```

C++

```
array of TNodeClassInfo TNodeClassArray;
```

File

XMLDoc

Description

This is type XMLDoc.TNodeClassArray.

195.5.3 XMLDoc.TNodeListNotification Type

TNodeListNotification is the type of the callback the TXMLNodeList (see page 26211) calls when it adds, deletes, or creates XML nodes.

Pascal

```
TNodeListNotification = procedure (Operation: TNodeListOperation; var Node: IXMLNode; const
IndexOrName: OleVariant; BeforeOperation: Boolean) of object;
```

C++

```
(Operation: TNodeListOperation; var Node: IXMLNode; const IndexOrName: OleVariant;
BeforeOperation: Boolean) ( TNodeListNotification)();
```

File

XMLDoc

Description

TNodeListNotification is the type for the callback supplied to TXMLNodeList (see page 26211) in its constructor. TXMLNodeList (see page 26211) calls this procedure before and after it adds or deletes nodes in the list and when it needs to create a new node.

Operation indicates what change is about to occur or has just occurred.

Node specifies the node that is the focus of that change. It's use depends on the operation, as indicated in the following table:

Operation	Node
nlInsert	The node that is about to be added to the list or that has just been added.
nlRemove	The node that is about to be removed from the list or that has just been removed.
nlCreateNode	The newly-created node. This should be created by the callback and returned as the Node parameter.
ntElement	The tag name.
ntAttribute	The name of the attribute.
ntText	The value of the node.
ntCData	The value of the CDATA section.
ntEntityRef	The name of the referenced entity.
ntProcessingInstr	The target of the processing instruction.
ntComment	The value (text) of the comment.
ntDocFragment	Not used.

IndexOrName is used only when Operation is nlCreateNode. It is the value assigned to the Nodes property using the index of a node that does not yet exist. The interpretation of IndexOrName depends on the type of node that the callback creates, as indicated in the following table:

NodeType	IndexOrName
ntElement	The tag name.
ntAttribute	The name of the attribute.
ntText	The value of the node.
ntCData	The value of the CDATA section.
ntEntityRef	The name of the referenced entity.
ntProcessingInstr	The target of the processing instruction.
ntComment	The value (text) of the comment.
ntDocFragment	Not used.

BeforeOperation indicates whether the callback occurs before the indicated operation or after. When nodes are inserted and removed, the callback is called first with BeforeOperation set to true, then the operation occurs, and then a second callback is made with BeforeOperation set to false. When nodes must be created, only one call is made, with BeforeOperation set to true.

See Also

TXMLNodeList

TXMLNode

TXMLNode

IXMLNodeList

195.5.4 XMLDoc.TXMLNodeArray Type

TXMLNodeArray is a dynamic array of TXMLNode (see page 26147) objects.

Pascal

```
TXMLNodeArray = array of TXMLNode;
```

C++

```
array of TXMLNode TXMLNodeArray;
```

File

XMLDoc

Description

TXMLNodeArray is a dynamic array of TXMLNode (see page 26147) objects, each of which represents a node in an XML document.

195.5.5 XMLDoc.TXMLNodeClass Type

TXMLNodeClass is a class reference to a TXMLNode (see page 26147) descendant.

Pascal

```
TXMLNodeClass = class of TXMLNode;
```

C++

```
TXMLNode TXMLNodeClass;
```

File

XMLDoc

Description

TXMLNodeClass describes a class that descends from TXMLNode (see page 26147).

195.5.6 XMLDoc.TXMLNodeCollectionClass Type

TXMLNodeCollectionClass is a class reference to a TXMLNodeCollection (see page 26196) descendant.

Pascal

```
TXMLNodeCollectionClass = class of TXMLNodeCollection;
```

C++

```
TXMLNodeCollection TXMLNodeCollectionClass;
```

File

XMLDoc

Description

TXMLNodeCollectionClass describes a class that descends from TXMLNodeCollection (see page 26196).

195.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
UseXSDBooleanStrings (see page 26251)	This is variable XMLDoc.UseXSDBooleanStrings.

195.6.1 XMLDoc.UseXSDBooleanStrings Variable

Pascal

```
UseXSDBooleanStrings: Boolean = True;
```

C++

```
Boolean UseXSDBooleanStrings = True;
```

File

XMLDoc

Description

This is variable XMLDoc.UseXSDBooleanStrings.

196 xmldom Namespace

This is namespace xmldom.

Topics

Name	Description
Classes (see page 26256)	The following table lists classes in this documentation.
Interfaces (see page 26270)	The following table lists interfaces in this documentation.
Functions (see page 26354)	The following table lists functions in this documentation.
Types (see page 26362)	The following table lists types in this documentation.
Variables (see page 26365)	The following table lists variables in this documentation.
Constants (see page 26366)	The following table lists constants in this documentation.

Classes

	Name	Description
	DOMException (see page 26256)	This is class xmldom.DOMException.
	EDOMParseError (see page 26258)	EDOMParseError is the exception class for errors that occur when a DOM implementation parses an XML document.
	TDOMVendor (see page 26263)	TDOMVendor is the base class for components that provide access to a DOM implementation.
	TDOMVendorList (see page 26265)	TDOMVendorList represents a collection of available DOM implementations.

Constants

Name	Description
ATTRIBUTE_NODE (see page 26366)	This is constant xmldom.ATTRIBUTE_NODE.
CDATA_SECTION_NODE (see page 26367)	This is constant xmldom.CDATA_SECTION_NODE.
COMMENT_NODE (see page 26367)	This is constant xmldom.COMMENT_NODE.
DOCUMENT_FRAGMENT_NODE (see page 26367)	This is constant xmldom.DOCUMENT_FRAGMENT_NODE.
DOCUMENT_NODE (see page 26368)	This is constant xmldom.DOCUMENT_NODE.
DOCUMENT_TYPE_NODE (see page 26368)	This is constant xmldom.DOCUMENT_TYPE_NODE.
DOMSTRING_SIZE_ERR (see page 26368)	This is constant xmldom.DOMSTRING_SIZE_ERR.
DOMWrapperVersion (see page 26369)	This is constant xmldom.DOMWrapperVersion.
ELEMENT_NODE (see page 26369)	This is constant xmldom.ELEMENT_NODE.
ENTITY_NODE (see page 26369)	This is constant xmldom.ENTITY_NODE.
ENTITY_REFERENCE_NODE (see page 26370)	This is constant xmldom.ENTITY_REFERENCE_NODE.
HIERARCHY_REQUEST_ERR (see page 26370)	This is constant xmldom.HIERARCHY_REQUEST_ERR.
INDEX_SIZE_ERR (see page 26370)	This is constant xmldom.INDEX_SIZE_ERR.

INUSE_ATTRIBUTE_ERR (see page 26371)	This is constant xmldom.INUSE_ATTRIBUTE_ERR.
INVALID_ACCESS_ERR (see page 26371)	This is constant xmldom.INVALID_ACCESS_ERR.
INVALID_CHARACTER_ERR (see page 26371)	This is constant xmldom.INVALID_CHARACTER_ERR.
INVALID_MODIFICATION_ERR (see page 26372)	This is constant xmldom.INVALID_MODIFICATION_ERR.
INVALID_STATE_ERR (see page 26372)	This is constant xmldom.INVALID_STATE_ERR.
NAMESPACE_ERR (see page 26372)	This is constant xmldom.NAMESPACE_ERR.
NO_DATA_ALLOWED_ERR (see page 26373)	This is constant xmldom.NO_DATA_ALLOWED_ERR.
NO_MODIFICATION_ALLOWED_ERR (see page 26373)	This is constant xmldom.NO_MODIFICATION_ALLOWED_ERR.
NOT_FOUND_ERR (see page 26373)	This is constant xmldom.NOT_FOUND_ERR.
NOT_SUPPORTED_ERR (see page 26374)	This is constant xmldom.NOT_SUPPORTED_ERR.
NOTATION_NODE (see page 26374)	This is constant xmldom.NOTATION_NODE.
NSDelim (see page 26374)	This is constant xmldom.NSDelim.
PROCESSING_INSTRUCTION_NODE (see page 26375)	This is constant xmldom.PROCESSING_INSTRUCTION_NODE.
SEncoding (see page 26375)	This is constant xmldom.SEncoding.
SHttp (see page 26375)	This is constant xmldom.SHttp.
SStandalone (see page 26376)	This is constant xmldom.SStandalone.
SVersion (see page 26376)	This is constant xmldom.SVersion.
SXML (see page 26376)	This is constant xmldom.SXML.
SXMLNamespaceURI (see page 26377)	This is constant xmldom.SXMLNamespaceURI.
SXMLNS (see page 26377)	This is constant xmldom.SXMLNS.
SXMLPrefixNamespaceURI (see page 26377)	This is constant xmldom.SXMLPrefixNamespaceURI.
SYNTAX_ERR (see page 26378)	This is constant xmldom.SYNTAX_ERR.
TEXT_NODE (see page 26378)	This is constant xmldom.TEXT_NODE.
WRONG_DOCUMENT_ERR (see page 26378)	This is constant xmldom.WRONG_DOCUMENT_ERR.

Functions

	Name	Description
	DOMVendorNotSupported (see page 26354)	This is function xmldom.DOMVendorNotSupported.
	ExtractLocalName (see page 26354)	Strips the namespace prefix, if present, from the tag name of an XML node.
	ExtractPrefix (see page 26355)	Returns the namespace prefix of an XML node's tag name.
	GetDOM (see page 26356)	Returns the top-level DOM interface for a registered DOM implementation.
	GetDOMNodeEx (see page 26356)	This is function xmldom.GetDOMNodeEx.
	GetDOMVendor (see page 26357)	Returns the TDOMVendor (see page 26263) object for a specified DOM vendor.
	IsPrefixed (see page 26357)	Indicates whether an XML tag name includes a namespace prefix.
	MakeNodeName (see page 26358)	Assembles a full tag name from a local name and namespace prefix.
	NodeMatches (see page 26358)	Indicates whether a specified node matches a given namespace and tag name.
	RegisterDOMVendor (see page 26359)	Registers an implementation of the DOM interfaces so that it is available to the TXMLDocument class.
	SameNamespace (see page 26359)	Indicates whether a specified node is defined within a specified namespace.
	SameNamespace (see page 26359)	Indicates whether a specified node is defined within a specified namespace.
	UnRegisterDOMVendor (see page 26360)	Unregisters a DOM implementation that was registered using RegisterDOMVendor (see page 26359) .

Interfaces

	Name	Description
	IDOMAttr (see page 26270)	This is class xmldom.IDOMAttr.
	IDOMCDATASection (see page 26274)	This is class xmldom.IDOMCDATASection.
	IDOMCharacterData (see page 26275)	This is class xmldom.IDOMCharacterData.
	IDOMComment (see page 26279)	This is class xmldom.IDOMComment.
	IDOMDocument (see page 26279)	This is class xmldom.IDOMDocument.
	IDOMDocumentFragment (see page 26286)	This is class xmldom.IDOMDocumentFragment.
	IDOMDocumentType (see page 26287)	This is class xmldom.IDOMDocumentType.
	IDOMElement (see page 26291)	This is class xmldom.IDOMElement.
	IDOMEntity (see page 26298)	This is class xmldom.IDOMEntity.

	IDOMEntityReference (see page 26301)	This is class xmldom.IDOMEntityReference.
	IDOMImplementation (see page 26301)	This is class xmldom.IDOMImplementation.
	IDOMNamedNodeMap (see page 26303)	This is class xmldom.IDOMNamedNodeMap.
	IDOMNode (see page 26306)	IDOMNode is the DOM interface for a node in an XML document.
	IDOMNodeEx (see page 26320)	This is class xmldom.IDOMNodeEx.
	IDOMNodeList (see page 26323)	This is class xmldom.IDOMNodeList.
	IDOMNodeSelect (see page 26325)	This is class xmldom.IDOMNodeSelect.
	IDOMNotation (see page 26327)	This is class xmldom.IDOMNotation.
	IDOMParseError (see page 26329)	This is class xmldom.IDOMParseError.
	IDOMParseOptions (see page 26334)	This is class xmldom.IDOMParseOptions.
	IDOMPersist (see page 26338)	This is class xmldom.IDOMPersist.
	IDOMProcessingInstruction (see page 26342)	This is class xmldom.IDOMProcessingInstruction.
	IDOMText (see page 26345)	This is class xmldom.IDOMText.
	IDOMXMLProlog (see page 26347)	This is class xmldom.IDOMXMLProlog.
	IDOMXSLProcessor (see page 26350)	This is class xmldom.IDOMXSLProcessor.

Types

Name	Description
DOMNodeType (see page 26362)	DOMNodeType indicates the type of a node in an XML document, as parsed by a DOM parser.
DOMString (see page 26362)	DOMString is the type for strings that are used by a DOM parser.
DOMTimeStamp (see page 26363)	This is type xmldom.DOMTimeStamp.
TAsyncEventHandler (see page 26363)	TAsyncEventHandler is the type of event handlers that respond when the AsyncLoadState of an XML document changes.
TDOMVendorArray (see page 26364)	This is type xmldom.TDOMVendorArray.

Variables

Name	Description
DefaultDOMVendor (see page 26365)	This is variable xmldom.DefaultDOMVendor.
DOMVendors (see page 26365)	This is variable xmldom.DOMVendors.

196.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	DOMException (see page 26256)	This is class xmldom.DOMException.
	EDOMParseError (see page 26258)	EDOMParseError is the exception class for errors that occur when a DOM implementation parses an XML document.
	TDOMVendor (see page 26263)	TDOMVendor is the base class for components that provide access to a DOM implementation.
	TDOMVendorList (see page 26265)	TDOMVendorList represents a collection of available DOM implementations.

196.1.1 DOMException Class

Class Hierarchy

Pascal

```
DOMException = class(Exception);
```

C++

```
class DOMException : public Exception;
```

File

xmldom

Description

This is class xmldom.DOMException.

196.1.1.1 DOMException Members

The following tables list the members exposed by DOMException.

Public Fields

	Name	Description
	code (see page 26257)	This is code, a member of class DOMException.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.

	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 22752)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.

196.1.1.2 DOMException Fields

The fields of the DOMException class are listed here.

Public Fields

	Name	Description
	code (see page 26257)	This is code, a member of class DOMException.

196.1.1.2.1 DOMException.code Field

Pascal

```
code: Word;
```

C++

```
Word code;
```

Description

This is code, a member of class DOMException.

196.1.2 EDOMParseError Class

EDOMParseError is the exception class for errors that occur when a DOM implementation parses an XML document.

Class Hierarchy

System::TObject → SysUtils.Exception → xmldom.EDOMParseError

Pascal

```
EDOMParseError = class(Exception);
```

C++

```
class EDOMParseError : public Exception;
```

File

xmldom

Description

EDOMParseError is raised when TXMLDocument encounters an error while parsing an XML document. It wraps the information that is available from the DOM implementation's IDOMParseError (see page 26329) interface.

See Also

TXMLDocument (see page 26097)

196.1.2.1 EDOMParseError Members

The following tables list the members exposed by EDOMParseError.

Protected Properties

	Name	Description
R	ParseError (see page 26261)	This is ParseError, a member of class EDOMParseError.

Public Methods

	Name	Description
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 26259)	This is Create, a member of class EDOMParseError.

	CreateFmt (see page 22754)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 22755)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 22755)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 22756)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 22756)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 22757)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 22759)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	ErrorCode (see page 26260)	Indicates the error code that the DOM parser returned.
R	FilePos (see page 26260)	Indicates the offset in the XML document where the parser encountered an error.
	HelpContext (see page 22760)	Indicates the context ID number for context-sensitive online help associated with the exception.
R	Line (see page 26261)	Indicates the line of the XML document where the parser encountered an error.
R	LinePos (see page 26261)	Indicates the position within the problem line where the parser encountered an error.
	Message (see page 22760)	Message contains the text string to display in the exception dialog box when the exception is raised.
R	Reason (see page 26262)	Indicates the error code that the DOM parser returned.
R	SrcText (see page 26262)	Indicates the text of the line where the parser encountered an error.
R	URL (see page 26262)	Identifies the source that could not be parsed by its URL.

196.1.2.2 EDOMParseError.Create Constructor

Pascal

```
constructor Create(const ParseError: IDOMParseError; const Msg: string);
```

C++

```
__fastcall EDOMParseError(const IDOMParseError ParseError, const AnsiString Msg);
```

Description

This is Create, a member of class EDOMParseError.

196.1.2.3 EDOMParseError Properties

The properties of the EDOMParseError class are listed here.

Protected Properties

	Name	Description
R	ParseError (see page 26261)	This is ParseError, a member of class EDOMParseError.

Public Properties

	Name	Description
R	ErrorCode (see page 26260)	Indicates the error code that the DOM parser returned.
R	FilePos (see page 26260)	Indicates the offset in the XML document where the parser encountered an error.
R	Line (see page 26261)	Indicates the line of the XML document where the parser encountered an error.
R	LinePos (see page 26261)	Indicates the position within the problem line where the parser encountered an error.
R	Reason (see page 26262)	Indicates the error code that the DOM parser returned.
R	SrcText (see page 26262)	Indicates the text of the line where the parser encountered an error.
R	URL (see page 26262)	Identifies the source that could not be parsed by its URL.

196.1.2.3.1 EDOMParseError.ErrorCode Property

Indicates the error code that the DOM parser returned.

Pascal

```
property ErrorCode: Integer;
```

C++

```
__property int ErrorCode;
```

Description

ErrorCode is the error code supplied by the DOM parser when it encountered a problem parsing the XML document.

See Also

Reason (see page 26262)

196.1.2.3.2 EDOMParseError.FilePos Property

Indicates the offset in the XML document where the parser encountered an error.

Pascal

```
property FilePos: Integer;
```

C++

```
__property int FilePos;
```

Description

FilePos is the offset in the XML document where the parser detected the error.

See Also

Line (see page 26261)

LinePos (see page 26261)

SrcText (see page 26262)

URL (see page 26262)

196.1.2.3.3 EDOMParseError.Line Property

Indicates the line of the XML document where the parser encountered an error.

Pascal

```
property Line: Integer;
```

C++

```
__property int Line;
```

Description

Line is the line of the XML document where the parser detected the error.

See Also

FilePos (see page 26260)

LinePos (see page 26261)

SrcText (see page 26262)

URL (see page 26262)

196.1.2.3.4 EDOMParseError.LinePos Property

Indicates the position within the problem line where the parser encountered an error.

Pascal

```
property LinePos: Integer;
```

C++

```
__property int LinePos;
```

Description

LinePos is the offset in the line specified by the Line (see page 26261) property where the DOM parser detected an error.

See Also

Line (see page 26261)

FilePos (see page 26260)

SrcText (see page 26262)

URL (see page 26262)

196.1.2.3.5 EDOMParseError.ParseError Property

Pascal

```
property ParseError: IDOMParseError;
```

C++

```
__property IDOMParseError ParseError;
```

Description

This is ParseError, a member of class EDOMParseError.

196.1.2.3.6 EDOMParseError.Reason Property

Indicates the error code that the DOM parser returned.

Pascal

```
property Reason: DOMString;
```

C++

```
__property DOMString Reason;
```

Description

Reason is a short text string that describes the error that was encountered. It corresponds to the error code specified by ErrorCode (see page 26260).

See Also

ErrorCode (see page 26260)

Message (see page 22760)

196.1.2.3.7 EDOMParseError.SrcText Property

Indicates the text of the line where the parser encountered an error.

Pascal

```
property SrcText: DOMString;
```

C++

```
__property DOMString SrcText;
```

Description

SrcText is the text from the XML document that the DOM parser could not interpret.

See Also

Line (see page 26261)

196.1.2.3.8 EDOMParseError.URL Property

Identifies the source that could not be parsed by its URL.

Pascal

```
property URL: DOMString;
```

C++

```
__property DOMString URL;
```

Description

URL identifies the problem the DOM parser encountered by its URL.

See Also

SrcText (see page 26262)

196.1.3 TDOMVendor Class

TDOMVendor is the base class for components that provide access to a DOM implementation.

Class Hierarchy

xmldom.TDOMVendor

Pascal

```
TDOMVendor = class;
```

C++

```
class TDOMVendor;
```

File

xmldom

Description

Note: Use TDOMVendor as a base class for a class that can be used to represent a DOM implementation. TDOMVendor introduces two abstract (pure virtual) methods, which provide information about a specific DOM implementation. One returns the IDOMImplementation (see page 26301) interface that acts as the top-level interface for a DOM vendor's implementation. The other returns a unique string that identifies the DOM implementation.

Once a TDOMVendor object has been registered, TXMLDocument objects can assign the DOM implementation as the value of their DOMVendor property by choosing the identifying string in the Object Inspector.

See Also

RegisterDOMVendor

UnregisterDOMVendor

TXMLDocument (see page 26097)

GetDOMVendor

Using the Document Object Model

196.1.3.1 TDOMVendor Members

The following tables list the members exposed by TDOMVendor.

Public Methods

	Name	Description
	Description (see page 26264)	Returns a string that uniquely identifies the associated DOM implementation.
	DOMImplementation (see page 26264)	Returns the IDOMImplementation (see page 26301) interface for the associated DOM implementation.

196.1.3.2 TDOMVendor Methods

The methods of the TDOMVendor class are listed here.

Public Methods

	Name	Description
	Description (see page 26264)	Returns a string that uniquely identifies the associated DOM implementation.
	DOMImplementation (see page 26264)	Returns the IDOMImplementation (see page 26301) interface for the associated DOM implementation.

196.1.3.2.1 TDOMVendor.Description Method

Returns a string that uniquely identifies the associated DOM implementation.

Pascal

```
function Description: string; virtual; abstract;
```

C++

```
virtual __fastcall AnsiString Description() = 0;
```

Description

Description returns the name, or description, of the associated DOM implementation. This is the string that appears in the Object Inspector for a TXMLDocument object's DOMVendor property.

Description is an abstract (pure virtual) method in TDOMVendor (see page 26263). Descendant classes override this method to provide a unique identifier for the DOM implementation.

See Also

DOMVendor (see page 26137)

GetDOMVendor

196.1.3.2.2 TDOMVendor.DOMImplementation Method

Returns the IDOMImplementation (see page 26301) interface for the associated DOM implementation.

Pascal

```
function DOMImplementation: IDOMImplementation; virtual; abstract;
```

C++

```
virtual __fastcall IDOMImplementation DOMImplementation() = 0;
```

Description

DOMImplementation returns the top-level interface for a DOM implementation. This interface is the starting point for any application that uses the DOM implementation to parse and edit an XML document.

DOMImplementation is an abstract (pure virtual) method in TDOMVendor (see page 26263). Descendant classes override this method to return the IDOMImplementation (see page 26301) interface for the associated DOM implementation.

Note: For more information about the IDOMImplementation (see page 26301), see the W3C Document Object Model (DOM) Level 2 Specification or the documentation provided by your DOM vendor.

See Also

domimplementation (see page 26137)

Description (see page 26264)

196.1.4 TDOMVendorList Class

TDOMVendorList represents a collection of available DOM implementations.

Class Hierarchy

xmldom.TDOMVendorList

Pascal

```
TDOMVendorList = class;
```

C++

```
class TDOMVendorList;
```

File

xmldom

Description

TDOMVendorList is the type of the global DOMVendors (see page 26365) variable. It represents a collection of DOM implementations (TDOMVendor (see page 26263) objects).

TDOMVendorList introduces a single property, Vendors (see page 26268), that can be used to iterate through all the DOM vendors in the collection. It also introduces methods that can be used to add, delete, or locate vendors.

Note: Do not explicitly use TDOMVendorList methods to add or delete vendors from the global DOMVendors (see page 26365) variable. Instead, add or remove vendors from this global list by calling the global RegisterDOMVendor (see page 26359) and UnRegisterDOMVendor (see page 26360) procedures.

See Also

DOMVendors

TXMLDocument (see page 26097)

RegisterDOMVendor

TDOMVendor (see page 26263)

UnregisterDOMVendor

196.1.4.1 TDOMVendorList Members

The following tables list the members exposed by TDOMVendorList.

Protected Methods

	Name	Description
	GetVendors (see page 26267)	This is GetVendors, a member of class TDOMVendorList.

Public Methods

	Name	Description
	Add (see page 26266)	Adds a new DOM implementation to the end of the list.
	Count (see page 26266)	Returns the number of DOM vendors in the collection.
	Find (see page 26267)	Returns a vendor from the list, given its description.
	Remove (see page 26268)	Removes a specified vendor from the list.

Public Properties

	Name	Description
	Vendors (see page 26268)	Lists the DOM vendors maintained by this TDOMVendorList (see page 26265) object.

196.1.4.2 TDOMVendorList Methods

The methods of the TDOMVendorList class are listed here.

Protected Methods

	Name	Description
	GetVendors (see page 26267)	This is GetVendors, a member of class TDOMVendorList.

Public Methods

	Name	Description
	Add (see page 26266)	Adds a new DOM implementation to the end of the list.
	Count (see page 26266)	Returns the number of DOM vendors in the collection.
	Find (see page 26267)	Returns a vendor from the list, given its description.
	Remove (see page 26268)	Removes a specified vendor from the list.

196.1.4.2.1 TDOMVendorList.Add Method

Adds a new DOM implementation to the end of the list.

Pascal

```
procedure Add(const Vendor: TDOMVendor);
```

C++

```
__fastcall Add(const TDOMVendor Vendor);
```

Description

Call Add to add a DOM vendor to the end of the list.

Vendor is an object that provides access to the IDOMImplementation (see page 26301) interface for the new DOM vendor.

If the list already includes a DOM vendor with the same description as that supplied by Vendor, Add raises an exception.

See Also

Count

Vendors

Find

Remove

Description (see page 26264)

196.1.4.2.2 TDOMVendorList.Count Method

Returns the number of DOM vendors in the collection.

Pascal

```
function Count: Integer;
```

C++

```
__fastcall int Count();
```

Description

Call Count to determine the number of DOM implementations stored in the list. Count returns a value that can be used as an upper bound when iterating over the vendors using the Vendors (see page 26268) property.

Count increases when the Add (see page 26266) method adds a vendor to the list and decreases when the Remove (see page 26268) method removes a vendor from the list.

See Also

Vendors

Add

Remove

196.1.4.2.3 TDOMVendorList.Find Method

Returns a vendor from the list, given its description.

Pascal

```
function Find(const VendorDesc: string): TDOMVendor;
```

C++

```
__fastcall TDOMVendor Find(const AnsiString VendorDesc);
```

Description

Call Find to access a vendor in the list, given its description. If there is no vendor in the list with the specified description, Find returns nil (Delphi) or NULL (C++).

VendorDesc is the string that is returned by the TDOMVendor (see page 26263) object's Description method.

Find returns a TDOMVendor (see page 26263) object, which can be used as a value for the DOMVendor property of a TXMLDocument component.

See Also

Vendors

Description (see page 26264)

DOMVendor (see page 26137)

196.1.4.2.4 TDOMVendorList.GetVendors Method

Pascal

```
function GetVendors(Index: Integer): TDOMVendor;
```

C++

```
__fastcall TDOMVendor GetVendors(int Index);
```

Description

This is GetVendors, a member of class TDOMVendorList.

196.1.4.2.5 TDOMVendorList.Remove Method

Removes a specified vendor from the list.

Pascal

```
procedure Remove(const Vendor: TDOMVendor);
```

C++

```
__fastcall Remove(const TDOMVendor Vendor);
```

Description

Remove removes the specified vendor from the list.

Vendor is the DOM vendor to remove from the list. If the list does not include a vendor with the same description as this parameter, Remove does nothing.

See Also

Count

Vendors

Find

Add

Description (see page 26264)

196.1.4.3 TDOMVendorList Properties

The properties of the TDOMVendorList class are listed here.

Public Properties

	Name	Description
R	Vendors (see page 26268)	Lists the DOM vendors maintained by this TDOMVendorList (see page 26265) object.

196.1.4.3.1 TDOMVendorList.Vendors Property

Lists the DOM vendors maintained by this TDOMVendorList (see page 26265) object.

Pascal

```
property Vendors [Index: Integer]: TDOMVendor;
```

C++

```
__property TDOMVendor Vendors[int Index];
```

Description

Use Vendors to access the DOM vendors in this collection. Index is the index of the vendor. Its value ranges from 0 to one less than the value returned by the Count (see page 26266) method.

Note: In Delphi, Vendors is the default property for TDOMVendorList (see page 26265). This means that the property name can be omitted in code. That is, the statement

for i := 0 to DOMVendors. Count (see page 26266) do

StringList1.Add (see page 26266)(DOMVendors.Vendors[i].Description);

Can be written

for i := 0 to DOMVendors.Count (see page 26266) do

StringList1.Add (see page 26266)(DOMVendors (see page 26365)[i].Description);

Note: In C++, Vendors can be accessed using the [] operator, to achieve an affect similar to the default property in Delphi.

See Also

Count

Add

Find

Remove

operator_sb

196.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IDOMAttr (see page 26270)	This is class xmldom.IDOMAttr.
	IDOMCDATASection (see page 26274)	This is class xmldom.IDOMCDATASection.
	IDOMCharacterData (see page 26275)	This is class xmldom.IDOMCharacterData.
	IDOMComment (see page 26279)	This is class xmldom.IDOMComment.
	IDOMDocument (see page 26279)	This is class xmldom.IDOMDocument.
	IDOMDocumentFragment (see page 26286)	This is class xmldom.IDOMDocumentFragment.
	IDOMDocumentType (see page 26287)	This is class xmldom.IDOMDocumentType.
	IDOMElement (see page 26291)	This is class xmldom.IDOMElement.
	IDOMEntity (see page 26298)	This is class xmldom.IDOMEntity.
	IDOMEntityReference (see page 26301)	This is class xmldom.IDOMEntityReference.
	IDOMImplementation (see page 26301)	This is class xmldom.IDOMImplementation.
	IDOMNamedNodeMap (see page 26303)	This is class xmldom.IDOMNamedNodeMap.
	IDOMNode (see page 26306)	IDOMNode is the DOM interface for a node in an XML document.
	IDOMNodeEx (see page 26320)	This is class xmldom.IDOMNodeEx.
	IDOMNodeList (see page 26323)	This is class xmldom.IDOMNodeList.
	IDOMNodeSelect (see page 26325)	This is class xmldom.IDOMNodeSelect.
	IDOMNotation (see page 26327)	This is class xmldom.IDOMNotation.
	IDOMParseError (see page 26329)	This is class xmldom.IDOMParseError.
	IDOMParseOptions (see page 26334)	This is class xmldom.IDOMParseOptions.
	IDOMPersist (see page 26338)	This is class xmldom.IDOMPersist.
	IDOMProcessingInstruction (see page 26342)	This is class xmldom.IDOMProcessingInstruction.
	IDOMText (see page 26345)	This is class xmldom.IDOMText.
	IDOMXMLProlog (see page 26347)	This is class xmldom.IDOMXMLProlog.
	IDOMXSLProcessor (see page 26350)	This is class xmldom.IDOMXSLProcessor.

196.2.1 IDOMAttr Interface

Class Hierarchy

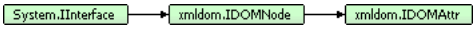

Pascal

```
IDOMAttr = interface(IDOMNode);
```

C++

```
__interface IDOMAttr : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMAttr.

196.2.1.1 IDOMAttr Members

The following tables list the members exposed by IDOMAttr.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	appendChild (see page 26308)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 26309)	This is cloneNode, a member of class IDOMNode.
	get_attributes (see page 26309)	Returns the value of the get_attributes property.
	get_childNodes (see page 26309)	Returns the value of the get_childNodes property.
	get_firstChild (see page 26309)	Returns the value of the get_firstChild property.
	get_lastChild (see page 26310)	Returns the value of the get_lastChild property.
	get_localName (see page 26310)	Returns the value of the get_localName property.
	get_name (see page 26272)	This is get_name, a member of class IDOMAttr.
	get_namespaceURI (see page 26310)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 26311)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 26311)	Returns the value of the get_nodeName property.
	get_nodeType (see page 26311)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 26312)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 26312)	Returns the value of the get_ownerDocument property.
	get_ownerElement (see page 26272)	This is get_ownerElement, a member of class IDOMAttr.
	get_parentNode (see page 26312)	Returns the value of the get_parentNode property.
	get_prefix (see page 26312)	Returns the value of the get_prefix property.
	get_previousSibling (see page 26313)	Returns the value of the get_previousSibling property.
	get_specified (see page 26272)	This is get_specified, a member of class IDOMAttr.
	get_value (see page 26273)	This is get_value, a member of class IDOMAttr.
	hasChildNodes (see page 26313)	Indicates whether the node has any child nodes.
	insertBefore (see page 26313)	Inserts a new child node before a specified child of the node.
	normalize (see page 26314)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 26315)	Removes a child node from this node.
	replaceChild (see page 26315)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 26316)	Sets the value of the node.
	set_value (see page 26273)	This is set_value, a member of class IDOMAttr.
	supports (see page 26316)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 26317)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 26317)	This is childNodes, a member of class IDOMNode.
R	firstChild (see page 26317)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 26317)	This is lastChild, a member of class IDOMNode.
R	localName (see page 26318)	This is localName, a member of class IDOMNode.
R	name (see page 26273)	This is name, a member of class IDOMAttr.
R	namespaceURI (see page 26318)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 26318)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 26318)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 26318)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 26319)	This is nodeValue, a member of class IDOMNode.
R	ownerDocument (see page 26319)	This is ownerDocument, a member of class IDOMNode.

R	ownerElement (see page 26273)	This is ownerElement, a member of class IDOMAttr.
R	parentNode (see page 26319)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 26319)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 26320)	This is previousSibling, a member of class IDOMNode.
R	specified (see page 26274)	This is specified, a member of class IDOMAttr.
	value (see page 26274)	This is value, a member of class IDOMAttr.

196.2.1.2 IDOMAttr Methods

The methods of the IDOMAttr class are listed here.

Public Methods

	Name	Description
	get_name (see page 26272)	This is get_name, a member of class IDOMAttr.
	get_ownerElement (see page 26272)	This is get_ownerElement, a member of class IDOMAttr.
	get_specified (see page 26272)	This is get_specified, a member of class IDOMAttr.
	get_value (see page 26273)	This is get_value, a member of class IDOMAttr.
	set_value (see page 26273)	This is set_value, a member of class IDOMAttr.

196.2.1.2.1 IDOMAttr.get_name Method

Pascal

```
function get_name: DOMString; safecall;
```

C++

```
__fastcall DOMString get_name();
```

Description

This is get_name, a member of class IDOMAttr.

196.2.1.2.2 IDOMAttr.get_ownerElement Method

Pascal

```
function get_ownerElement: IDOMElement; safecall;
```

C++

```
__fastcall IDOMElement get_ownerElement();
```

Description

This is get_ownerElement, a member of class IDOMAttr.

196.2.1.2.3 IDOMAttr.get_specified Method

Pascal

```
function get_specified: WordBool; safecall;
```

C++

```
__fastcall WordBool get_specified();
```

Description

This is get_specified, a member of class IDOMAttr.

196.2.1.2.4 IDOMAttr.get_value Method

Pascal

```
function get_value: DOMString; safecall;
```

C++

```
__fastcall DOMString get_value();
```

Description

This is get_value, a member of class IDOMAttr.

196.2.1.2.5 IDOMAttr.set_value Method

Pascal

```
procedure set_value(const attributeValue: DOMString); safecall;
```

C++

```
__fastcall set_value(const DOMString attributeValue);
```

Description

This is set_value, a member of class IDOMAttr.

196.2.1.3 IDOMAttr Properties

The properties of the IDOMAttr class are listed here.

Public Properties

	Name	Description
R	name (see page 26273)	This is name, a member of class IDOMAttr.
R	ownerElement (see page 26273)	This is ownerElement, a member of class IDOMAttr.
R	specified (see page 26274)	This is specified, a member of class IDOMAttr.
	value (see page 26274)	This is value, a member of class IDOMAttr.

196.2.1.3.1 IDOMAttr.name Property

Pascal

```
property name: DOMString;
```

C++

```
__property DOMString name;
```

Description

This is name, a member of class IDOMAttr.

196.2.1.3.2 IDOMAttr.ownerElement Property

Pascal

```
property ownerElement: IDOMElement;
```

C++

```
__property IDOMElement ownerElement;
```

Description

This is ownerElement, a member of class IDOMAttr.

196.2.1.3.3 IDOMAttr.specified Property

Pascal

```
property specified: WordBool;
```

C++

```
__property WordBool specified;
```

Description

This is specified, a member of class IDOMAttr.

196.2.1.3.4 IDOMAttr.value Property

Pascal

```
property value: DOMString;
```

C++

```
__property DOMString value;
```

Description

This is value, a member of class IDOMAttr.

196.2.2 IDOMCDATASection Interface

Class Hierarchy

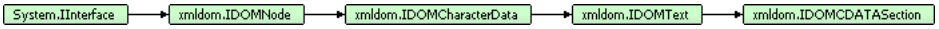

Pascal

```
IDOMCDATASection = interface(IDOMText);
```

C++

```
__interface IDOMCDATASection : public IDOMText;
```

File

xmldom

Description

This is class xmldom.IDOMCDATASection.

196.2.3 IDOMCharacterData Interface

Class Hierarchy

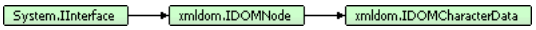

Pascal

```
IDOMCharacterData = interface(IDOMNode);
```

C++

```
__interface IDOMCharacterData : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMCharacterData.

196.2.3.1 IDOMCharacterData Members

The following tables list the members exposed by IDOMCharacterData.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	appendChild (see page 26308)	This is appendChild, a member of class IDOMNode.
	appendData (see page 26276)	This is appendData, a member of class IDOMCharacterData.
	cloneNode (see page 26309)	This is cloneNode, a member of class IDOMNode.
	deleteData (see page 26277)	This is deleteData, a member of class IDOMCharacterData.
	get_attributes (see page 26309)	Returns the value of the get_attributes property.
	get_childNodes (see page 26309)	Returns the value of the get_childNodes property.
	get_data (see page 26277)	This is get_data, a member of class IDOMCharacterData.
	get_firstChild (see page 26309)	Returns the value of the get_firstChild property.
	get_lastChild (see page 26310)	Returns the value of the get_lastChild property.
	get_length (see page 26277)	This is get_length, a member of class IDOMCharacterData.
	get_localName (see page 26310)	Returns the value of the get_localName property.
	get_namespaceURI (see page 26310)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 26311)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 26311)	Returns the value of the get_nodeName property.
	get_nodeType (see page 26311)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 26312)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 26312)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 26312)	Returns the value of the get_parentNode property.
	get_prefix (see page 26312)	Returns the value of the get_prefix property.
	get_previousSibling (see page 26313)	Returns the value of the get_previousSibling property.
	hasChildNodes (see page 26313)	Indicates whether the node has any child nodes.
	insertBefore (see page 26313)	Inserts a new child node before a specified child of the node.
	insertData (see page 26277)	This is insertData, a member of class IDOMCharacterData.
	normalize (see page 26314)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 26315)	Removes a child node from this node.
	replaceChild (see page 26315)	Replaces one of this node's children with a specified node.
	replaceData (see page 26277)	This is replaceData, a member of class IDOMCharacterData.
	set_data (see page 26278)	This is set_data, a member of class IDOMCharacterData.
	set_nodeValue (see page 26316)	Sets the value of the node.
	substringData (see page 26278)	This is substringData, a member of class IDOMCharacterData.
	supports (see page 26316)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 26317)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 26317)	This is childNodes, a member of class IDOMNode.
	data (see page 26278)	This is data, a member of class IDOMCharacterData.
R	firstChild (see page 26317)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 26317)	This is lastChild, a member of class IDOMNode.
R	length (see page 26278)	This is length, a member of class IDOMCharacterData.
R	localName (see page 26318)	This is localName, a member of class IDOMNode.
R	namespaceURI (see page 26318)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 26318)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 26318)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 26318)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 26319)	This is nodeValue, a member of class IDOMNode.
R	ownerDocument (see page 26319)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 26319)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 26319)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 26320)	This is previousSibling, a member of class IDOMNode.

196.2.3.2 IDOMCharacterData Methods

The methods of the IDOMCharacterData class are listed here.

Public Methods

	Name	Description
	appendData (see page 26276)	This is appendData, a member of class IDOMCharacterData.
	deleteData (see page 26277)	This is deleteData, a member of class IDOMCharacterData.
	get_data (see page 26277)	This is get_data, a member of class IDOMCharacterData.
	get_length (see page 26277)	This is get_length, a member of class IDOMCharacterData.
	insertData (see page 26277)	This is insertData, a member of class IDOMCharacterData.
	replaceData (see page 26277)	This is replaceData, a member of class IDOMCharacterData.
	set_data (see page 26278)	This is set_data, a member of class IDOMCharacterData.
	substringData (see page 26278)	This is substringData, a member of class IDOMCharacterData.

196.2.3.2.1 IDOMCharacterData.appendData Method

Pascal

```
procedure appendData(const data: DOMString); safecall;
```

C++

```
__fastcall appendData(const DOMString data);
```

Description

This is appendData, a member of class IDOMCharacterData.

196.2.3.2.2 IDOMCharacterData.deleteData Method

Pascal

```
procedure deleteData(offset: Integer; count: Integer); safecall;
```

C++

```
__fastcall deleteData(int offset, int count);
```

Description

This is deleteData, a member of class IDOMCharacterData.

196.2.3.2.3 IDOMCharacterData.get_data Method

Pascal

```
function get_data: DOMString; safecall;
```

C++

```
__fastcall DOMString get_data();
```

Description

This is get_data, a member of class IDOMCharacterData.

196.2.3.2.4 IDOMCharacterData.get_length Method

Pascal

```
function get_length: Integer; safecall;
```

C++

```
__fastcall int get_length();
```

Description

This is get_length, a member of class IDOMCharacterData.

196.2.3.2.5 IDOMCharacterData.insertData Method

Pascal

```
procedure insertData(offset: Integer; const data: DOMString); safecall;
```

C++

```
__fastcall insertData(int offset, const DOMString data);
```

Description

This is insertData, a member of class IDOMCharacterData.

196.2.3.2.6 IDOMCharacterData.replaceData Method

Pascal

```
procedure replaceData(offset: Integer; count: Integer; const data: DOMString); safecall;
```

C++

```
__fastcall replaceData(int offset, int count, const DOMString data);
```

Description

This is replaceData, a member of class IDOMCharacterData.

196.2.3.2.7 IDOMCharacterData.set_data Method

Pascal

```
procedure set_data(const data: DOMString); safecall;
```

C++

```
__fastcall set_data(const DOMString data);
```

Description

This is set_data, a member of class IDOMCharacterData.

196.2.3.2.8 IDOMCharacterData.substringData Method

Pascal

```
function substringData(offset: Integer; count: Integer): DOMString; safecall;
```

C++

```
__fastcall DOMString substringData(int offset, int count);
```

Description

This is substringData, a member of class IDOMCharacterData.

196.2.3.3 IDOMCharacterData Properties

The properties of the IDOMCharacterData class are listed here.

Public Properties

	Name	Description
	data (see page 26278)	This is data, a member of class IDOMCharacterData.
R	length (see page 26278)	This is length, a member of class IDOMCharacterData.

196.2.3.3.1 IDOMCharacterData.data Property

Pascal

```
property data: DOMString;
```

C++

```
__property DOMString data;
```

Description

This is data, a member of class IDOMCharacterData.

196.2.3.3.2 IDOMCharacterData.length Property

Pascal

```
property length: Integer;
```

C++

```
__property int length;
```

Description

This is length, a member of class IDOMCharacterData.

196.2.4 IDOMComment Interface

Class Hierarchy

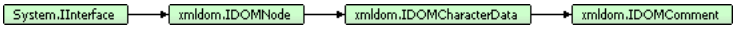

Pascal

```
IDOMComment = interface(IDOMCharacterData);
```

C++

```
__interface IDOMComment : public IDOMCharacterData;
```

File

xmldom

Description

This is class xmldom.IDOMComment.

196.2.5 IDOMDocument Interface

Class Hierarchy

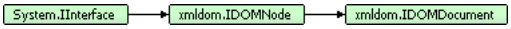

Pascal

```
IDOMDocument = interface(IDOMNode);
```

C++

```
__interface IDOMDocument : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMDocument.

196.2.5.1 IDOMDocument Members

The following tables list the members exposed by IDOMDocument.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	appendChild (see page 26308)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 26309)	This is cloneNode, a member of class IDOMNode.
	createAttribute (see page 26281)	This is createAttribute, a member of class IDOMDocument.
	createAttributeNS (see page 26282)	This is createAttributeNS, a member of class IDOMDocument.
	createCDATASection (see page 26282)	This is createCDATASection, a member of class IDOMDocument.
	createComment (see page 26282)	This is createComment, a member of class IDOMDocument.
	createDocumentFragment (see page 26282)	This is createDocumentFragment, a member of class IDOMDocument.
	createElement (see page 26282)	This is createElement, a member of class IDOMDocument.
	createElementNS (see page 26283)	This is createElementNS, a member of class IDOMDocument.
	createEntityReference (see page 26283)	This is createEntityReference, a member of class IDOMDocument.
	createProcessingInstruction (see page 26283)	This is createProcessingInstruction, a member of class IDOMDocument.
	createTextNode (see page 26283)	This is createTextNode, a member of class IDOMDocument.
	get_attributes (see page 26309)	Returns the value of the get_attributes property.
	get_childNodes (see page 26309)	Returns the value of the get_childNodes property.
	get_doctype (see page 26284)	This is get_doctype, a member of class IDOMDocument.
	get_documentElement (see page 26284)	This is get_documentElement, a member of class IDOMDocument.
	get_domImplementation (see page 26284)	This is get_domImplementation, a member of class IDOMDocument.
	get_firstChild (see page 26309)	Returns the value of the get_firstChild property.
	get_lastChild (see page 26310)	Returns the value of the get_lastChild property.
	get_localName (see page 26310)	Returns the value of the get_localName property.
	get_namespaceURI (see page 26310)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 26311)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 26311)	Returns the value of the get_nodeName property.
	get_nodeType (see page 26311)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 26312)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 26312)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 26312)	Returns the value of the get_parentNode property.
	get_prefix (see page 26312)	Returns the value of the get_prefix property.
	get_previousSibling (see page 26313)	Returns the value of the get_previousSibling property.
	getElementById (see page 26284)	This is getElementById, a member of class IDOMDocument.
	getElementsByTagName (see page 26285)	This is getElementsByTagName, a member of class IDOMDocument.
	getElementsByTagNameNS (see page 26285)	This is getElementsByTagNameNS, a member of class IDOMDocument.
	hasChildNodes (see page 26313)	Indicates whether the node has any child nodes.
	importNode (see page 26285)	This is importNode, a member of class IDOMDocument.
	insertBefore (see page 26313)	Inserts a new child node before a specified child of the node.
	normalize (see page 26314)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 26315)	Removes a child node from this node.
	replaceChild (see page 26315)	Replaces one of this node's children with a specified node.
	set_documentElement (see page 26285)	This is set_documentElement, a member of class IDOMDocument.
	set_nodeValue (see page 26316)	Sets the value of the node.
	supports (see page 26316)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 26317)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 26317)	This is childNodes, a member of class IDOMNode.
R	doctype (see page 26286)	This is doctype, a member of class IDOMDocument.
	documentElement (see page 26286)	This is documentElement, a member of class IDOMDocument.

R	domImplementation (see page 26286)	This is domImplementation, a member of class IDOMDocument.
R	firstChild (see page 26317)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 26317)	This is lastChild, a member of class IDOMNode.
R	localName (see page 26318)	This is localName, a member of class IDOMNode.
R	namespaceURI (see page 26318)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 26318)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 26318)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 26318)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 26319)	This is nodeValue, a member of class IDOMNode.
R	ownerDocument (see page 26319)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 26319)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 26319)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 26320)	This is previousSibling, a member of class IDOMNode.

196.2.5.2 IDOMDocument Methods

The methods of the IDOMDocument class are listed here.

Public Methods

	Name	Description
	createAttribute (see page 26281)	This is createAttribute, a member of class IDOMDocument.
	createAttributeNS (see page 26282)	This is createAttributeNS, a member of class IDOMDocument.
	createCDATASection (see page 26282)	This is createCDATASection, a member of class IDOMDocument.
	createComment (see page 26282)	This is createComment, a member of class IDOMDocument.
	createDocumentFragment (see page 26282)	This is createDocumentFragment, a member of class IDOMDocument.
	createElement (see page 26282)	This is createElement, a member of class IDOMDocument.
	createElementNS (see page 26283)	This is createElementNS, a member of class IDOMDocument.
	createEntityReference (see page 26283)	This is createEntityReference, a member of class IDOMDocument.
	createProcessingInstruction (see page 26283)	This is createProcessingInstruction, a member of class IDOMDocument.
	createTextNode (see page 26283)	This is createTextNode, a member of class IDOMDocument.
	get_doctype (see page 26284)	This is get_doctype, a member of class IDOMDocument.
	get_documentElement (see page 26284)	This is get_documentElement, a member of class IDOMDocument.
	get_domImplementation (see page 26284)	This is get_domImplementation, a member of class IDOMDocument.
	getElementById (see page 26284)	This is getElementById, a member of class IDOMDocument.
	getElementsByTagName (see page 26285)	This is getElementsByTagName, a member of class IDOMDocument.
	getElementsByTagNameNS (see page 26285)	This is getElementsByTagNameNS, a member of class IDOMDocument.
	importNode (see page 26285)	This is importNode, a member of class IDOMDocument.
	set_documentElement (see page 26285)	This is set_documentElement, a member of class IDOMDocument.

196.2.5.2.1 IDOMDocument.createAttribute Method

Pascal

```
function createAttribute(const name: DOMString): IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr createAttribute(const DOMString name);
```

Description

This is createAttribute, a member of class IDOMDocument.

196.2.5.2.2 IDOMDocument.createAttributeNS Method

Pascal

```
function createAttributeNS(const namespaceURI: DOMString; const qualifiedName: DOMString):
IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr createAttributeNS(const DOMString namespaceURI, const DOMString
qualifiedName);
```

Description

This is createAttributeNS, a member of class IDOMDocument.

196.2.5.2.3 IDOMDocument.createCDATASection Method

Pascal

```
function createCDATASection(const data: DOMString): IDOMCDATASection; safecall;
```

C++

```
__fastcall IDOMCDATASection createCDATASection(const DOMString data);
```

Description

This is createCDATASection, a member of class IDOMDocument.

196.2.5.2.4 IDOMDocument.createComment Method

Pascal

```
function createComment(const data: DOMString): IDOMComment; safecall;
```

C++

```
__fastcall IDOMComment createComment(const DOMString data);
```

Description

This is createComment, a member of class IDOMDocument.

196.2.5.2.5 IDOMDocument.createDocumentFragment Method

Pascal

```
function createDocumentFragment: IDOMDocumentFragment; safecall;
```

C++

```
__fastcall IDOMDocumentFragment createDocumentFragment();
```

Description

This is createDocumentFragment, a member of class IDOMDocument.

196.2.5.2.6 IDOMDocument.createElement Method

Pascal

```
function createElement(const tagName: DOMString): IDOMElement; safecall;
```

C++

```
__fastcall IDOMElement createElement(const DOMString tagName);
```

Description

This is createElement, a member of class IDOMDocument.

196.2.5.2.7 IDOMDocument.createElementNS Method

Pascal

```
function createElementNS(const namespaceURI: DOMString; const qualifiedName: DOMString):
IDOMElement; safecall;
```

C++

```
__fastcall IDOMElement createElementNS(const DOMString namespaceURI, const DOMString
qualifiedName);
```

Description

This is createElementNS, a member of class IDOMDocument.

196.2.5.2.8 IDOMDocument.createEntityReference Method

Pascal

```
function createEntityReference(const name: DOMString): IDOMEntityReference; safecall;
```

C++

```
__fastcall IDOMEntityReference createEntityReference(const DOMString name);
```

Description

This is createEntityReference, a member of class IDOMDocument.

196.2.5.2.9 IDOMDocument.createProcessingInstruction Method

Pascal

```
function createProcessingInstruction(const target: DOMString; const data: DOMString):
IDOMProcessingInstruction; safecall;
```

C++

```
__fastcall IDOMProcessingInstruction createProcessingInstruction(const DOMString target, const
DOMString data);
```

Description

This is createProcessingInstruction, a member of class IDOMDocument.

196.2.5.2.10 IDOMDocument.createTextNode Method

Pascal

```
function createTextNode(const data: DOMString): IDOMText; safecall;
```

C++

```
__fastcall IDOMText createTextNode(const DOMString data);
```

Description

This is createTextNode, a member of class IDOMDocument.

196.2.5.2.11 IDOMDocument.get_doctype Method

Pascal

```
function get_doctype: IDOMDocumentType; safecall;
```

C++

```
__fastcall IDOMDocumentType get_doctype();
```

Description

This is get_doctype, a member of class IDOMDocument.

196.2.5.2.12 IDOMDocument.get_documentElement Method

Pascal

```
function get_documentElement: IDOMElement; safecall;
```

C++

```
__fastcall IDOMElement get_documentElement();
```

Description

This is get_documentElement, a member of class IDOMDocument.

196.2.5.2.13 IDOMDocument.get_domImplementation Method

Pascal

```
function get_domImplementation: IDOMImplementation; safecall;
```

C++

```
__fastcall IDOMImplementation get_domImplementation();
```

Description

This is get_domImplementation, a member of class IDOMDocument.

196.2.5.2.14 IDOMDocument.getElementById Method

Pascal

```
function getElementById(const elementId: DOMString): IDOMElement; safecall;
```

C++

```
__fastcall IDOMElement getElementById(const DOMString elementId);
```

Description

This is getElementById, a member of class IDOMDocument.

196.2.5.2.15 IDOMDocument.getElementsByTagName Method

Pascal

```
function getElementsByTagName(const tagName: DOMString): IDOMNodeList; safecall;
```

C++

```
__fastcall IDOMNodeList getElementsByTagName(const DOMString tagName);
```

Description

This is getElementsByTagName, a member of class IDOMDocument.

196.2.5.2.16 IDOMDocument.getElementsByTagNameNS Method

Pascal

```
function getElementsByTagNameNS(const namespaceURI: DOMString; const localName: DOMString):
IDOMNodeList; safecall;
```

C++

```
__fastcall IDOMNodeList getElementsByTagNameNS(const DOMString namespaceURI, const DOMString
localName);
```

Description

This is getElementsByTagNameNS, a member of class IDOMDocument.

196.2.5.2.17 IDOMDocument.importNode Method

Pascal

```
function importNode(importedNode: IDOMNode; deep: WordBool): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode importNode(IDOMNode importedNode, WordBool deep);
```

Description

This is importNode, a member of class IDOMDocument.

196.2.5.2.18 IDOMDocument.set_documentElement Method

Pascal

```
procedure set_documentElement(const Element: IDOMElement); safecall;
```

C++

```
__fastcall set_documentElement(const IDOMElement Element);
```

Description

This is set_documentElement, a member of class IDOMDocument.

196.2.5.3 IDOMDocument Properties

The properties of the IDOMDocument class are listed here.

Public Properties

	Name	Description
R	doctype (see page 26286)	This is doctype, a member of class IDOMDocument.
	documentElement (see page 26286)	This is documentElement, a member of class IDOMDocument.
R	domImplementation (see page 26286)	This is domImplementation, a member of class IDOMDocument.

196.2.5.3.1 IDOMDocument.doctype Property

Pascal

```
property doctype: IDOMDocumentType;
```

C++

```
__property IDOMDocumentType doctype;
```

Description

This is doctype, a member of class IDOMDocument.

196.2.5.3.2 IDOMDocument.documentElement Property

Pascal

```
property documentElement: IDOMElement;
```

C++

```
__property IDOMElement documentElement;
```

Description

This is documentElement, a member of class IDOMDocument.

196.2.5.3.3 IDOMDocument.domImplementation Property

Pascal

```
property domImplementation: IDOMImplementation;
```

C++

```
__property IDOMImplementation domImplementation;
```

Description

This is domImplementation, a member of class IDOMDocument.

196.2.6 IDOMDocumentFragment Interface

Class Hierarchy

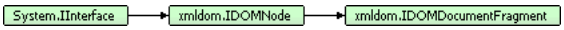

Pascal

```
IDOMDocumentFragment = interface(IDOMNode);
```

C++

```
__interface IDOMDocumentFragment : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMDocumentFragment.

196.2.7 IDOMDocumentType Interface

Class Hierarchy

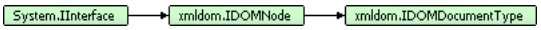

Pascal

```
IDOMDocumentType = interface(IDOMNode);
```

C++

```
__interface IDOMDocumentType : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMDocumentType.

196.2.7.1 IDOMDocumentType Members

The following tables list the members exposed by IDOMDocumentType.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	appendChild (see page 26308)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 26309)	This is cloneNode, a member of class IDOMNode.
	get_attributes (see page 26309)	Returns the value of the get_attributes property.
	get_childNodes (see page 26309)	Returns the value of the get_childNodes property.
	get_entities (see page 26289)	This is get_entities, a member of class IDOMDocumentType.
	get_firstChild (see page 26309)	Returns the value of the get_firstChild property.
	get_internalSubset (see page 26289)	This is get_internalSubset, a member of class IDOMDocumentType.
	get_lastChild (see page 26310)	Returns the value of the get_lastChild property.
	get_localName (see page 26310)	Returns the value of the get_localName property.
	get_name (see page 26289)	This is get_name, a member of class IDOMDocumentType.
	get_namespaceURI (see page 26310)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 26311)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 26311)	Returns the value of the get_nodeName property.
	get_nodeType (see page 26311)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 26312)	Returns the value of the get_nodeValue property.
	get_notations (see page 26289)	This is get_notations, a member of class IDOMDocumentType.
	get_ownerDocument (see page 26312)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 26312)	Returns the value of the get_parentNode property.

	get_prefix (see page 26312)	Returns the value of the get_prefix property.
	get_previousSibling (see page 26313)	Returns the value of the get_previousSibling property.
	get_publicId (see page 26289)	This is get_publicId, a member of class IDOMDocumentType.
	get_systemId (see page 26290)	This is get_systemId, a member of class IDOMDocumentType.
	hasChildNodes (see page 26313)	Indicates whether the node has any child nodes.
	insertBefore (see page 26313)	Inserts a new child node before a specified child of the node.
	normalize (see page 26314)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 26315)	Removes a child node from this node.
	replaceChild (see page 26315)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 26316)	Sets the value of the node.
	supports (see page 26316)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 26317)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 26317)	This is childNodes, a member of class IDOMNode.
R	entities (see page 26290)	This is entities, a member of class IDOMDocumentType.
R	firstChild (see page 26317)	This is firstChild, a member of class IDOMNode.
R	internalSubset (see page 26290)	This is internalSubset, a member of class IDOMDocumentType.
R	lastChild (see page 26317)	This is lastChild, a member of class IDOMNode.
R	localName (see page 26318)	This is localName, a member of class IDOMNode.
R	name (see page 26291)	This is name, a member of class IDOMDocumentType.
R	namespaceURI (see page 26318)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 26318)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 26318)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 26318)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 26319)	This is nodeValue, a member of class IDOMNode.
R	notations (see page 26291)	This is notations, a member of class IDOMDocumentType.
R	ownerDocument (see page 26319)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 26319)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 26319)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 26320)	This is previousSibling, a member of class IDOMNode.
R	publicId (see page 26291)	This is publicId, a member of class IDOMDocumentType.
R	systemId (see page 26291)	This is systemId, a member of class IDOMDocumentType.

196.2.7.2 IDOMDocumentType Methods

The methods of the IDOMDocumentType class are listed here.

Public Methods

	Name	Description
	get_entities (see page 26289)	This is get_entities, a member of class IDOMDocumentType.
	get_internalSubset (see page 26289)	This is get_internalSubset, a member of class IDOMDocumentType.
	get_name (see page 26289)	This is get_name, a member of class IDOMDocumentType.
	get_notations (see page 26289)	This is get_notations, a member of class IDOMDocumentType.
	get_publicId (see page 26289)	This is get_publicId, a member of class IDOMDocumentType.
	get_systemId (see page 26290)	This is get_systemId, a member of class IDOMDocumentType.

196.2.7.2.1 IDOMDocumentType.get_entities Method

Pascal

```
function get_entities: IDOMNamedNodeMap; safecall;
```

C++

```
__fastcall IDOMNamedNodeMap get_entities();
```

Description

This is get_entities, a member of class IDOMDocumentType.

196.2.7.2.2 IDOMDocumentType.get_internalSubset Method

Pascal

```
function get_internalSubset: DOMString; safecall;
```

C++

```
__fastcall DOMString get_internalSubset();
```

Description

This is get_internalSubset, a member of class IDOMDocumentType.

196.2.7.2.3 IDOMDocumentType.get_name Method

Pascal

```
function get_name: DOMString; safecall;
```

C++

```
__fastcall DOMString get_name();
```

Description

This is get_name, a member of class IDOMDocumentType.

196.2.7.2.4 IDOMDocumentType.get_notations Method

Pascal

```
function get_notations: IDOMNamedNodeMap; safecall;
```

C++

```
__fastcall IDOMNamedNodeMap get_notations();
```

Description

This is get_notations, a member of class IDOMDocumentType.

196.2.7.2.5 IDOMDocumentType.get_publicId Method

Pascal

```
function get_publicId: DOMString; safecall;
```

C++

```
__fastcall DOMString get_publicId();
```

Description

This is get_publicId, a member of class IDOMDocumentType.

196.2.7.2.6 IDOMDocumentType.get_systemId Method

Pascal

```
function get_systemId: DOMString; safecall;
```

C++

```
__fastcall DOMString get_systemId();
```

Description

This is get_systemId, a member of class IDOMDocumentType.

196.2.7.3 IDOMDocumentType Properties

The properties of the IDOMDocumentType class are listed here.

Public Properties

	Name	Description
R	entities (see page 26290)	This is entities, a member of class IDOMDocumentType.
R	internalSubset (see page 26290)	This is internalSubset, a member of class IDOMDocumentType.
R	name (see page 26291)	This is name, a member of class IDOMDocumentType.
R	notations (see page 26291)	This is notations, a member of class IDOMDocumentType.
R	publicId (see page 26291)	This is publicId, a member of class IDOMDocumentType.
R	systemId (see page 26291)	This is systemId, a member of class IDOMDocumentType.

196.2.7.3.1 IDOMDocumentType.entities Property

Pascal

```
property entities: IDOMNamedNodeMap;
```

C++

```
__property IDOMNamedNodeMap entities;
```

Description

This is entities, a member of class IDOMDocumentType.

196.2.7.3.2 IDOMDocumentType.internalSubset Property

Pascal

```
property internalSubset: DOMString;
```

C++

```
__property DOMString internalSubset;
```

Description

This is internalSubset, a member of class IDOMDocumentType.

196.2.7.3.3 IDOMDocumentType.name Property

Pascal

```
property name: DOMString;
```

C++

```
__property DOMString name;
```

Description

This is name, a member of class IDOMDocumentType.

196.2.7.3.4 IDOMDocumentType.notations Property

Pascal

```
property notations: IDOMNamedNodeMap;
```

C++

```
__property IDOMNamedNodeMap notations;
```

Description

This is notations, a member of class IDOMDocumentType.

196.2.7.3.5 IDOMDocumentType.publicId Property

Pascal

```
property publicId: DOMString;
```

C++

```
__property DOMString publicId;
```

Description

This is publicId, a member of class IDOMDocumentType.

196.2.7.3.6 IDOMDocumentType.systemId Property

Pascal

```
property systemId: DOMString;
```

C++

```
__property DOMString systemId;
```

Description

This is systemId, a member of class IDOMDocumentType.

196.2.8 IDOMElement Interface

Class Hierarchy

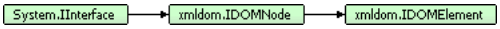

Pascal

```
IDOMElement = interface(IDOMNode);
```

C++

```
__interface IDOMElement : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMElement.

196.2.8.1 IDOMElement Members

The following tables list the members exposed by IDOMElement.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	appendChild (see page 26308)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 26309)	This is cloneNode, a member of class IDOMNode.
	get_attributes (see page 26309)	Returns the value of the get_attributes property.
	get_childNodes (see page 26309)	Returns the value of the get_childNodes property.
	get_firstChild (see page 26309)	Returns the value of the get_firstChild property.
	get_lastChild (see page 26310)	Returns the value of the get_lastChild property.
	get_localName (see page 26310)	Returns the value of the get_localName property.
	get_namespaceURI (see page 26310)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 26311)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 26311)	Returns the value of the get_nodeName property.
	get_nodeType (see page 26311)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 26312)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 26312)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 26312)	Returns the value of the get_parentNode property.
	get_prefix (see page 26312)	Returns the value of the get_prefix property.
	get_previousSibling (see page 26313)	Returns the value of the get_previousSibling property.
	get_tagName (see page 26294)	This is get_tagName, a member of class IDOMElement.
	getAttribute (see page 26294)	This is getAttribute, a member of class IDOMElement.
	getAttributeNode (see page 26294)	This is getAttributeNode, a member of class IDOMElement.
	getAttributeNodeNS (see page 26294)	This is getAttributeNodeNS, a member of class IDOMElement.
	getAttributeNS (see page 26294)	This is getAttributeNS, a member of class IDOMElement.
	getElementsByTagName (see page 26295)	This is getElementsByTagName, a member of class IDOMElement.
	getElementsByTagNameNS (see page 26295)	This is getElementsByTagNameNS, a member of class IDOMElement.
	hasAttribute (see page 26295)	This is hasAttribute, a member of class IDOMElement.
	hasAttributeNS (see page 26295)	This is hasAttributeNS, a member of class IDOMElement.
	hasChildNodes (see page 26313)	Indicates whether the node has any child nodes.
	insertBefore (see page 26313)	Inserts a new child node before a specified child of the node.
	normalize (see page 26314)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	removeAttribute (see page 26296)	This is removeAttribute, a member of class IDOMElement.
	removeAttributeNode (see page 26296)	This is removeAttributeNode, a member of class IDOMElement.
	removeAttributeNS (see page 26296)	This is removeAttributeNS, a member of class IDOMElement.

	removeChild (see page 26315)	Removes a child node from this node.
	replaceChild (see page 26315)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 26316)	Sets the value of the node.
	setAttribute (see page 26296)	This is setAttribute, a member of class IDOMElement.
	setAttributeNode (see page 26296)	This is setAttributeNode, a member of class IDOMElement.
	setAttributeNodeNS (see page 26297)	This is setAttributeNodeNS, a member of class IDOMElement.
	setAttributeNS (see page 26297)	This is setAttributeNS, a member of class IDOMElement.
	supports (see page 26316)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 26317)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 26317)	This is childNodes, a member of class IDOMNode.
R	firstChild (see page 26317)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 26317)	This is lastChild, a member of class IDOMNode.
R	localName (see page 26318)	This is localName, a member of class IDOMNode.
R	namespaceURI (see page 26318)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 26318)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 26318)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 26318)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 26319)	This is nodeValue, a member of class IDOMNode.
R	ownerDocument (see page 26319)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 26319)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 26319)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 26320)	This is previousSibling, a member of class IDOMNode.
R	tagName (see page 26297)	This is tagName, a member of class IDOMElement.

196.2.8.2 IDOMElement Methods

The methods of the IDOMElement class are listed here.

Public Methods

	Name	Description
	get_tagName (see page 26294)	This is get_tagName, a member of class IDOMElement.
	getAttribute (see page 26294)	This is getAttribute, a member of class IDOMElement.
	getAttributeNode (see page 26294)	This is getAttributeNode, a member of class IDOMElement.
	getAttributeNodeNS (see page 26294)	This is getAttributeNodeNS, a member of class IDOMElement.
	getAttributeNS (see page 26294)	This is getAttributeNS, a member of class IDOMElement.
	getElementsByTagName (see page 26295)	This is getElementsByTagName, a member of class IDOMElement.
	getElementsByTagNameNS (see page 26295)	This is getElementsByTagNameNS, a member of class IDOMElement.
	hasAttribute (see page 26295)	This is hasAttribute, a member of class IDOMElement.
	hasAttributeNS (see page 26295)	This is hasAttributeNS, a member of class IDOMElement.
	removeAttribute (see page 26296)	This is removeAttribute, a member of class IDOMElement.
	removeAttributeNode (see page 26296)	This is removeAttributeNode, a member of class IDOMElement.
	removeAttributeNS (see page 26296)	This is removeAttributeNS, a member of class IDOMElement.
	setAttribute (see page 26296)	This is setAttribute, a member of class IDOMElement.
	setAttributeNode (see page 26296)	This is setAttributeNode, a member of class IDOMElement.
	setAttributeNodeNS (see page 26297)	This is setAttributeNodeNS, a member of class IDOMElement.
	setAttributeNS (see page 26297)	This is setAttributeNS, a member of class IDOMElement.

196.2.8.2.1 IDOMElement.get_tagName Method

Pascal

```
function get_tagName: DOMString; safecall;
```

C++

```
__fastcall DOMString get_tagName();
```

Description

This is get_tagName, a member of class IDOMElement.

196.2.8.2.2 IDOMElement.getAttribute Method

Pascal

```
function getAttribute(const name: DOMString): DOMString; safecall;
```

C++

```
__fastcall DOMString getAttribute(const DOMString name);
```

Description

This is getAttribute, a member of class IDOMElement.

196.2.8.2.3 IDOMElement.getAttributeNode Method

Pascal

```
function getAttributeNode(const name: DOMString): IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr getAttributeNode(const DOMString name);
```

Description

This is getAttributeNode, a member of class IDOMElement.

196.2.8.2.4 IDOMElement.getAttributeNodeNS Method

Pascal

```
function getAttributeNodeNS(const namespaceURI: DOMString; const localName: DOMString):
IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr getAttributeNodeNS(const DOMString namespaceURI, const DOMString
localName);
```

Description

This is getAttributeNodeNS, a member of class IDOMElement.

196.2.8.2.5 IDOMElement.getAttributeNS Method

Pascal

```
function getAttributeNS(const namespaceURI: DOMString; const localName: DOMString): DOMString;
safecall;
```

C++

```
__fastcall DOMString getAttributeNS(const DOMString namespaceURI, const DOMString localName);
```

Description

This is getAttributeNS, a member of class IDOMElement.

196.2.8.2.6 IDOMElement.getElementsByTagName Method

Pascal

```
function getElementsByTagName(const name: DOMString): IDOMNodeList; safecall;
```

C++

```
__fastcall IDOMNodeList getElementsByTagName(const DOMString name);
```

Description

This is getElementsByTagName, a member of class IDOMElement.

196.2.8.2.7 IDOMElement.getElementsByTagNameNS Method

Pascal

```
function getElementsByTagNameNS(const namespaceURI: DOMString; const localName: DOMString):
IDOMNodeList; safecall;
```

C++

```
__fastcall IDOMNodeList getElementsByTagNameNS(const DOMString namespaceURI, const DOMString
localName);
```

Description

This is getElementsByTagNameNS, a member of class IDOMElement.

196.2.8.2.8 IDOMElement.hasAttribute Method

Pascal

```
function hasAttribute(const name: DOMString): WordBool; safecall;
```

C++

```
__fastcall WordBool hasAttribute(const DOMString name);
```

Description

This is hasAttribute, a member of class IDOMElement.

196.2.8.2.9 IDOMElement.hasAttributeNS Method

Pascal

```
function hasAttributeNS(const namespaceURI: DOMString; const localName: DOMString): WordBool;
safecall;
```

C++

```
__fastcall WordBool hasAttributeNS(const DOMString namespaceURI, const DOMString localName);
```

Description

This is hasAttributeNS, a member of class IDOMElement.

196.2.8.2.10 IDOMElement.removeAttribute Method

Pascal

```
procedure removeAttribute(const name: DOMString); safecall;
```

C++

```
__fastcall removeAttribute(const DOMString name);
```

Description

This is removeAttribute, a member of class IDOMElement.

196.2.8.2.11 IDOMElement.removeAttributeNode Method

Pascal

```
function removeAttributeNode(const oldAttr: IDOMAttr): IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr removeAttributeNode(const IDOMAttr oldAttr);
```

Description

This is removeAttributeNode, a member of class IDOMElement.

196.2.8.2.12 IDOMElement.removeAttributeNS Method

Pascal

```
procedure removeAttributeNS(const namespaceURI: DOMString; const localName: DOMString);
safecall;
```

C++

```
__fastcall removeAttributeNS(const DOMString namespaceURI, const DOMString localName);
```

Description

This is removeAttributeNS, a member of class IDOMElement.

196.2.8.2.13 IDOMElement.setAttribute Method

Pascal

```
procedure setAttribute(const name: DOMString; const value: DOMString); safecall;
```

C++

```
__fastcall setAttribute(const DOMString name, const DOMString value);
```

Description

This is setAttribute, a member of class IDOMElement.

196.2.8.2.14 IDOMElement.setAttributeNode Method

Pascal

```
function setAttributeNode(const newAttr: IDOMAttr): IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr setAttributeNode(const IDOMAttr newAttr);
```

Description

This is setAttributeNode, a member of class IDOMElement.

196.2.8.2.15 IDOMElement.setAttributeNodeNS Method

Pascal

```
function setAttributeNodeNS(const newAttr: IDOMAttr): IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr setAttributeNodeNS(const IDOMAttr newAttr);
```

Description

This is setAttributeNodeNS, a member of class IDOMElement.

196.2.8.2.16 IDOMElement.setAttributeNS Method

Pascal

```
procedure setAttributeNS(const namespaceURI: DOMString; const qulifiedName: DOMString; const
value: DOMString); safecall;
```

C++

```
__fastcall setAttributeNS(const DOMString namespaceURI, const DOMString qulifiedName, const
DOMString value);
```

Description

This is setAttributeNS, a member of class IDOMElement.

196.2.8.3 IDOMElement Properties

The properties of the IDOMElement class are listed here.

Public Properties

	Name	Description
R	tagName (see page 26297)	This is tagName, a member of class IDOMElement.

196.2.8.3.1 IDOMElement.tagName Property

Pascal

```
property tagName: DOMString;
```

C++

```
__property DOMString tagName;
```

Description

This is tagName, a member of class IDOMElement.

196.2.9 IDOMEntity Interface

Class Hierarchy

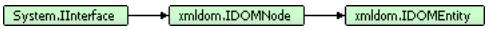

Pascal

```
IDOMEntity = interface(IDOMNode);
```

C++

```
__interface IDOMEntity : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMEntity.

196.2.9.1 IDOMEntity Members

The following tables list the members exposed by IDOMEntity.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	appendChild (see page 26308)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 26309)	This is cloneNode, a member of class IDOMNode.
	get_attributes (see page 26309)	Returns the value of the get_attributes property.
	get_childNodes (see page 26309)	Returns the value of the get_childNodes property.
	get_firstChild (see page 26309)	Returns the value of the get_firstChild property.
	get_lastChild (see page 26310)	Returns the value of the get_lastChild property.
	get_localName (see page 26310)	Returns the value of the get_localName property.
	get_namespaceURI (see page 26310)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 26311)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 26311)	Returns the value of the get_nodeName property.
	get_nodeType (see page 26311)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 26312)	Returns the value of the get_nodeValue property.
	get_notationName (see page 26299)	This is get_notationName, a member of class IDOMEntity.
	get_ownerDocument (see page 26312)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 26312)	Returns the value of the get_parentNode property.
	get_prefix (see page 26312)	Returns the value of the get_prefix property.
	get_previousSibling (see page 26313)	Returns the value of the get_previousSibling property.
	get_publicId (see page 26299)	This is get_publicId, a member of class IDOMEntity.
	get_systemId (see page 26300)	This is get_systemId, a member of class IDOMEntity.
	hasChildNodes (see page 26313)	Indicates whether the node has any child nodes.
	insertBefore (see page 26313)	Inserts a new child node before a specified child of the node.
	normalize (see page 26314)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 26315)	Removes a child node from this node.

	replaceChild (see page 26315)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 26316)	Sets the value of the node.
	supports (see page 26316)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 26317)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 26317)	This is childNodes, a member of class IDOMNode.
R	firstChild (see page 26317)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 26317)	This is lastChild, a member of class IDOMNode.
R	localName (see page 26318)	This is localName, a member of class IDOMNode.
R	namespaceURI (see page 26318)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 26318)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 26318)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 26318)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 26319)	This is nodeValue, a member of class IDOMNode.
R	notationName (see page 26300)	This is notationName, a member of class IDOMEntity.
R	ownerDocument (see page 26319)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 26319)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 26319)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 26320)	This is previousSibling, a member of class IDOMNode.
R	publicId (see page 26300)	This is publicId, a member of class IDOMEntity.
R	systemId (see page 26300)	This is systemId, a member of class IDOMEntity.

196.2.9.2 IDOMEntity Methods

The methods of the IDOMEntity class are listed here.

Public Methods

	Name	Description
	get_notationName (see page 26299)	This is get_notationName, a member of class IDOMEntity.
	get_publicId (see page 26299)	This is get_publicId, a member of class IDOMEntity.
	get_systemId (see page 26300)	This is get_systemId, a member of class IDOMEntity.

196.2.9.2.1 IDOMEntity.get_notationName Method

Pascal

```
function get_notationName: DOMString; safecall;
```

C++

```
__fastcall DOMString get_notationName();
```

Description

This is get_notationName, a member of class IDOMEntity.

196.2.9.2.2 IDOMEntity.get_publicId Method

Pascal

```
function get_publicId: DOMString; safecall;
```

C++

```
__fastcall DOMString get_publicId();
```

Description

This is get_publicId, a member of class IDOMEntity.

196.2.9.2.3 IDOMEntity.get_systemId Method

Pascal

```
function get_systemId: DOMString; safecall;
```

C++

```
__fastcall DOMString get_systemId();
```

Description

This is get_systemId, a member of class IDOMEntity.

196.2.9.3 IDOMEntity Properties

The properties of the IDOMEntity class are listed here.

Public Properties

	Name	Description
R	notationName (see page 26300)	This is notationName, a member of class IDOMEntity.
R	publicId (see page 26300)	This is publicId, a member of class IDOMEntity.
R	systemId (see page 26300)	This is systemId, a member of class IDOMEntity.

196.2.9.3.1 IDOMEntity.notationName Property

Pascal

```
property notationName: DOMString;
```

C++

```
__property DOMString notationName;
```

Description

This is notationName, a member of class IDOMEntity.

196.2.9.3.2 IDOMEntity.publicId Property

Pascal

```
property publicId: DOMString;
```

C++

```
__property DOMString publicId;
```

Description

This is publicId, a member of class IDOMEntity.

196.2.9.3.3 IDOMEntity.systemId Property

Pascal

```
property systemId: DOMString;
```

C++

```
__property DOMString systemId;
```

Description

This is systemId, a member of class IDOMEntity.

196.2.10 IDOMEntityReference Interface

Class Hierarchy

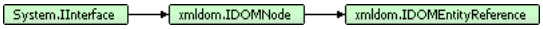

Pascal

```
IDOMEntityReference = interface(IDOMNode);
```

C++

```
__interface IDOMEntityReference : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMEntityReference.

196.2.11 IDOMImplementation Interface

Class Hierarchy

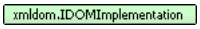

Pascal

```
IDOMImplementation = interface;
```

C++

```
__interface IDOMImplementation;
```

File

xmldom

Description

This is class xmldom.IDOMImplementation.

196.2.11.1 IDOMImplementation Members

The following tables list the members exposed by IDOMImplementation.

Public Methods

	Name	Description
	createDocument (see page 26302)	This is createDocument, a member of class IDOMImplementation.
	createDocumentType (see page 26302)	This is createDocumentType, a member of class IDOMImplementation.
	hasFeature (see page 26302)	This is hasFeature, a member of class IDOMImplementation.

196.2.11.2 IDOMImplementation Methods

The methods of the IDOMImplementation class are listed here.

Public Methods

	Name	Description
	createDocument (see page 26302)	This is createDocument, a member of class IDOMImplementation.
	createDocumentType (see page 26302)	This is createDocumentType, a member of class IDOMImplementation.
	hasFeature (see page 26302)	This is hasFeature, a member of class IDOMImplementation.

196.2.11.2.1 IDOMImplementation.createDocument Method

Pascal

```
function createDocument(const namespaceURI: DOMString; const qualifiedName: DOMString;
doctype: IDOMDocumentType): IDOMDocument; safecall;
```

C++

```
__fastcall IDOMDocument createDocument(const DOMString namespaceURI, const DOMString
qualifiedName, IDOMDocumentType doctype);
```

Description

This is createDocument, a member of class IDOMImplementation.

196.2.11.2.2 IDOMImplementation.createDocumentType Method

Pascal

```
function createDocumentType(const qualifiedName: DOMString; const publicId: DOMString; const
systemId: DOMString): IDOMDocumentType; safecall;
```

C++

```
__fastcall IDOMDocumentType createDocumentType(const DOMString qualifiedName, const DOMString
publicId, const DOMString systemId);
```

Description

This is createDocumentType, a member of class IDOMImplementation.

196.2.11.2.3 IDOMImplementation.hasFeature Method

Pascal

```
function hasFeature(const feature: DOMString; const version: DOMString): WordBool;
```

C++

```
__fastcall WordBool hasFeature(const DOMString feature, const DOMString version);
```

Description

This is hasFeature, a member of class IDOMImplementation.

196.2.12 IDOMNamedNodeMap Interface

Class Hierarchy

xmldom.IDOMNamedNodeMap

Pascal

```
IDOMNamedNodeMap = interface;
```

C++

```
__interface IDOMNamedNodeMap;
```

File

xmldom

Description

This is class xmldom.IDOMNamedNodeMap.

196.2.12.1 IDOMNamedNodeMap Members

The following tables list the members exposed by IDOMNamedNodeMap.

Public Methods

	Name	Description
	get_item (see page 26304)	This is get_item, a member of class IDOMNamedNodeMap.
	get_length (see page 26304)	This is get_length, a member of class IDOMNamedNodeMap.
	getNamedItem (see page 26304)	This is getNamedItem, a member of class IDOMNamedNodeMap.
	getNamedItemNS (see page 26304)	This is getNamedItemNS, a member of class IDOMNamedNodeMap.
	removeNamedItem (see page 26305)	This is removeNamedItem, a member of class IDOMNamedNodeMap.
	removeNamedItemNS (see page 26305)	This is removeNamedItemNS, a member of class IDOMNamedNodeMap.
	setNamedItem (see page 26305)	This is setNamedItem, a member of class IDOMNamedNodeMap.
	setNamedItemNS (see page 26305)	This is setNamedItemNS, a member of class IDOMNamedNodeMap.

Public Properties

	Name	Description
R	item (see page 26306)	This is item, a member of class IDOMNamedNodeMap.
R	length (see page 26306)	This is length, a member of class IDOMNamedNodeMap.

196.2.12.2 IDOMNamedNodeMap Methods

The methods of the IDOMNamedNodeMap class are listed here.

Public Methods

	Name	Description
	get_item (see page 26304)	This is get_item, a member of class IDOMNamedNodeMap.
	get_length (see page 26304)	This is get_length, a member of class IDOMNamedNodeMap.
	getNamedItem (see page 26304)	This is getNamedItem, a member of class IDOMNamedNodeMap.
	getNamedItemNS (see page 26304)	This is getNamedItemNS, a member of class IDOMNamedNodeMap.
	removeNamedItem (see page 26305)	This is removeNamedItem, a member of class IDOMNamedNodeMap.
	removeNamedItemNS (see page 26305)	This is removeNamedItemNS, a member of class IDOMNamedNodeMap.

	setNamedItem (see page 26305)	This is setNamedItem, a member of class IDOMNamedNodeMap.
	setNamedItemNS (see page 26305)	This is setNamedItemNS, a member of class IDOMNamedNodeMap.

196.2.12.2.1 IDOMNamedNodeMap.get_item Method

Pascal

```
function get_item(index: Integer): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_item(int index);
```

Description

This is get_item, a member of class IDOMNamedNodeMap.

196.2.12.2.2 IDOMNamedNodeMap.get_length Method

Pascal

```
function get_length: Integer; safecall;
```

C++

```
__fastcall int get_length();
```

Description

This is get_length, a member of class IDOMNamedNodeMap.

196.2.12.2.3 IDOMNamedNodeMap.getNamedItem Method

Pascal

```
function getNamedItem(const name: DOMString): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode getNamedItem(const DOMString name);
```

Description

This is getNamedItem, a member of class IDOMNamedNodeMap.

196.2.12.2.4 IDOMNamedNodeMap.getNamedItemNS Method

Pascal

```
function getNamedItemNS(const namespaceURI: DOMString; const localName: DOMString): IDOMNode;
safecall;
```

C++

```
__fastcall IDOMNode getNamedItemNS(const DOMString namespaceURI, const DOMString localName);
```

Description

This is getNamedItemNS, a member of class IDOMNamedNodeMap.

196.2.12.2.5 IDOMNamedNodeMap.removeNamedItem Method

Pascal

```
function removeNamedItem(const name: DOMString): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode removeNamedItem(const DOMString name);
```

Description

This is removeNamedItem, a member of class IDOMNamedNodeMap.

196.2.12.2.6 IDOMNamedNodeMap.removeNamedItemNS Method

Pascal

```
function removeNamedItemNS(const namespaceURI: DOMString; const localName: DOMString):
IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode removeNamedItemNS(const DOMString namespaceURI, const DOMString localName);
```

Description

This is removeNamedItemNS, a member of class IDOMNamedNodeMap.

196.2.12.2.7 IDOMNamedNodeMap.setNamedItem Method

Pascal

```
function setNamedItem(const arg: IDOMNode): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode setNamedItem(const IDOMNode arg);
```

Description

This is setNamedItem, a member of class IDOMNamedNodeMap.

196.2.12.2.8 IDOMNamedNodeMap.setNamedItemNS Method

Pascal

```
function setNamedItemNS(const arg: IDOMNode): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode setNamedItemNS(const IDOMNode arg);
```

Description

This is setNamedItemNS, a member of class IDOMNamedNodeMap.

196.2.12.3 IDOMNamedNodeMap Properties

The properties of the IDOMNamedNodeMap class are listed here.

Public Properties

	Name	Description
R	item (see page 26306)	This is item, a member of class IDOMNamedNodeMap.
R	length (see page 26306)	This is length, a member of class IDOMNamedNodeMap.

196.2.12.3.1 IDOMNamedNodeMap.item Property

Pascal

```
property item [index: Integer]: IDOMNode;
```

C++

```
__property IDOMNode item[int index];
```

Description

This is item, a member of class IDOMNamedNodeMap.

196.2.12.3.2 IDOMNamedNodeMap.length Property

Pascal

```
property length: Integer;
```

C++

```
__property int length;
```

Description

This is length, a member of class IDOMNamedNodeMap.

196.2.13 IDOMNode Interface

IDOMNode is the DOM interface for a node in an XML document.

Class Hierarchy

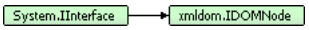

Pascal

```
IDOMNode = interface(IInterface);
```

C++

```
__interface IDOMNode : public IInterface;
```

File

xmldom

Description

IDOMNode is a vendor-neutral version of the DOM interface for a node in an XML document. It conforms to the W3CXML DOM level 2 specification. Use IDOMNode to work with a node in an XML document when you are using standard DOM interfaces rather than the IXMLNode interface. (IXMLNode is the node interface used by TXMLDocument document.)

IDOMNode represents an element in an XML document that has been parsed by a DOM parser. It includes properties and methods for navigating the node hierarchy of an XML document, for accessing the attributes (see page 26317) of a node, and

for drilling down to obtain the node's value.

For more information about IDOMNode and the other standard DOM interfaces, see the DOM specification on the W3 website or the documentation provided by your DOM vendor.

Note: C++ method declarations that use IDOMNode use the _di_IDOMNode type instead. This type is a DelphiInterface wrapper around the IDOMNode interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IDOMNode >  _di_IDOMNode;
```

See Also

IXMLNode (see page 26411)

TXMLDocument (see page 26097)

CreateDOMNode

196.2.13.1 IDOMNode Members

The following tables list the members exposed by IDOMNode.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	appendChild (see page 26308)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 26309)	This is cloneNode, a member of class IDOMNode.
	get_attributes (see page 26309)	Returns the value of the get_attributes property.
	get_childNodes (see page 26309)	Returns the value of the get_childNodes property.
	get_firstChild (see page 26309)	Returns the value of the get_firstChild property.
	get_lastChild (see page 26310)	Returns the value of the get_lastChild property.
	get_localName (see page 26310)	Returns the value of the get_localName property.
	get_namespaceURI (see page 26310)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 26311)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 26311)	Returns the value of the get_nodeName property.
	get_nodeType (see page 26311)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 26312)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 26312)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 26312)	Returns the value of the get_parentNode property.
	get_prefix (see page 26312)	Returns the value of the get_prefix property.
	get_previousSibling (see page 26313)	Returns the value of the get_previousSibling property.
	hasChildNodes (see page 26313)	Indicates whether the node has any child nodes.
	insertBefore (see page 26313)	Inserts a new child node before a specified child of the node.
	normalize (see page 26314)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 26315)	Removes a child node from this node.
	replaceChild (see page 26315)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 26316)	Sets the value of the node.
	supports (see page 26316)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 26317)	This is attributes, a member of class IDOMNode.

R	childNodes (see page 26317)	This is childNodes, a member of class IDOMNode.
R	firstChild (see page 26317)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 26317)	This is lastChild, a member of class IDOMNode.
R	localName (see page 26318)	This is localName, a member of class IDOMNode.
R	namespaceURI (see page 26318)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 26318)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 26318)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 26318)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 26319)	This is nodeValue, a member of class IDOMNode.
R	ownerDocument (see page 26319)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 26319)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 26319)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 26320)	This is previousSibling, a member of class IDOMNode.

196.2.13.2 IDOMNode Methods

The methods of the IDOMNode class are listed here.

Public Methods

	Name	Description
	appendChild (see page 26308)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 26309)	This is cloneNode, a member of class IDOMNode.
	get_attributes (see page 26309)	Returns the value of the get_attributes property.
	get_childNodes (see page 26309)	Returns the value of the get_childNodes property.
	get_firstChild (see page 26309)	Returns the value of the get_firstChild property.
	get_lastChild (see page 26310)	Returns the value of the get_lastChild property.
	get_localName (see page 26310)	Returns the value of the get_localName property.
	get_namespaceURI (see page 26310)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 26311)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 26311)	Returns the value of the get_nodeName property.
	get_nodeType (see page 26311)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 26312)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 26312)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 26312)	Returns the value of the get_parentNode property.
	get_prefix (see page 26312)	Returns the value of the get_prefix property.
	get_previousSibling (see page 26313)	Returns the value of the get_previousSibling property.
	hasChildNodes (see page 26313)	Indicates whether the node has any child nodes.
	insertBefore (see page 26313)	Inserts a new child node before a specified child of the node.
	normalize (see page 26314)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	removeChild (see page 26315)	Removes a child node from this node.
	replaceChild (see page 26315)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 26316)	Sets the value of the node.
	supports (see page 26316)	Indicates whether the node supports a specified feature.

196.2.13.2.1 IDOMNode.appendChild Method

Pascal

```
function appendChild(const newChild: IDOMNode): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode appendChild(const IDOMNode newChild);
```

Description

This is appendChild, a member of class IDOMNode.

196.2.13.2.2 IDOMNode.cloneNode Method

Pascal

```
function cloneNode(deep: WordBool): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode cloneNode(WordBool deep);
```

Description

This is cloneNode, a member of class IDOMNode.

196.2.13.2.3 IDOMNode.get_attributes Method

Returns the value of the get_attributes property.

Pascal

```
function get_attributes: IDOMNamedNodeMap; safecall;
```

C++

```
__fastcall IDOMNamedNodeMap get_attributes();
```

Description

get_get_attributes is the read implementation of the get_attributes property.

See Also

get_attributes

196.2.13.2.4 IDOMNode.get_childNodes Method

Returns the value of the get_childNodes property.

Pascal

```
function get_childNodes: IDOMNodeList; safecall;
```

C++

```
__fastcall IDOMNodeList get_childNodes();
```

Description

get_get_childNodes is the read implementation of the get_childNodes property.

See Also

get_childNodes

196.2.13.2.5 IDOMNode.get_firstChild Method

Returns the value of the get_firstChild property.

Pascal

```
function get_firstChild: IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_firstChild();
```

Description

get_get_firstChild is the read implementation of the get_firstChild property.

See Also

get_firstChild

196.2.13.2.6 IDOMNode.get_lastChild Method

Returns the value of the get_lastChild property.

Pascal

```
function get_lastChild: IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_lastChild();
```

Description

get_get_lastChild is the read implementation of the get_lastChild property.

See Also

get_lastChild

196.2.13.2.7 IDOMNode.get_localName Method

Returns the value of the get_localName property.

Pascal

```
function get_localName: DOMString; safecall;
```

C++

```
__fastcall DOMString get_localName();
```

Description

get_get_localName is the read implementation of the get_localName property.

See Also

get_localName

196.2.13.2.8 IDOMNode.get_namespaceURI Method

Returns the value of the get_namespaceURI property.

Pascal

```
function get_namespaceURI: DOMString; safecall;
```

C++

```
__fastcall DOMString get_namespaceURI();
```

Description

get_get_namespaceURI is the read implementation of the get_namespaceURI property.

See Also

get_namespaceURI

196.2.13.2.9 IDOMNode.get_nextSibling Method

Returns the value of the get_nextSibling property.

Pascal

```
function get_nextSibling: IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_nextSibling();
```

Description

get_get_nextSibling is the read implementation of the get_nextSibling property.

See Also

get_nextSibling

196.2.13.2.10 IDOMNode.get_nodeName Method

Returns the value of the get_nodeName property.

Pascal

```
function get_nodeName: DOMString; safecall;
```

C++

```
__fastcall DOMString get_nodeName();
```

Description

get_get_nodeName is the read implementation of the get_nodeName property.

See Also

get_nodeName

196.2.13.2.11 IDOMNode.get_nodeType Method

Returns the value of the get_nodeType property.

Pascal

```
function get_nodeType: DOMNodeType; safecall;
```

C++

```
__fastcall DOMNodeType get_nodeType();
```

Description

get_get_nodeType is the read implementation of the get_nodeType property.

See Also

get_nodeType

196.2.13.2.12 IDOMNode.get_nodeValue Method

Returns the value of the get_nodeValue property.

Pascal

```
function get_nodeValue: DOMString; safecall;
```

C++

```
__fastcall DOMString get_nodeValue();
```

Description

get_get_nodeValue is the read implementation of the get_nodeValue property.

See Also

get_nodeValue

196.2.13.2.13 IDOMNode.get_ownerDocument Method

Returns the value of the get_ownerDocument property.

Pascal

```
function get_ownerDocument: IDOMDocument; safecall;
```

C++

```
__fastcall IDOMDocument get_ownerDocument();
```

Description

get_get_ownerDocument is the read implementation of the get_ownerDocument property.

See Also

get_ownerDocument

196.2.13.2.14 IDOMNode.get_parentNode Method

Returns the value of the get_parentNode property.

Pascal

```
function get_parentNode: IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_parentNode();
```

Description

get_get_parentNode is the read implementation of the get_parentNode property.

See Also

get_parentNode

196.2.13.2.15 IDOMNode.get_prefix Method

Returns the value of the get_prefix property.

Pascal

```
function get_prefix: DOMString; safecall;
```

C++

```
__fastcall DOMString get_prefix();
```

Description

get_get_prefix is the read implementation of the get_prefix property.

See Also

get_prefix

196.2.13.2.16 IDOMNode.get_previousSibling Method

Returns the value of the get_previousSibling property.

Pascal

```
function get_previousSibling: IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_previousSibling();
```

Description

get_get_previousSibling is the read implementation of the get_previousSibling property.

See Also

get_previousSibling

196.2.13.2.17 IDOMNode.hasChildNodes Method

Indicates whether the node has any child nodes.

Pascal

```
function hasChildNodes: WordBool; safecall;
```

C++

```
__fastcall WordBool hasChildNodes();
```

Description

hasChildNodes returns true if this node has any child nodes, false if it has no children.

See Also

ChildNodes (see page 26317)

FirstChild (see page 26317)

LastChild (see page 26317)

196.2.13.2.18 IDOMNode.insertBefore Method

Inserts a new child node before a specified child of the node.

Pascal

```
function insertBefore(const newChild: IDOMNode; const refChild: IDOMNode): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode insertBefore(const IDOMNode newChild, const IDOMNode refChild);
```

Description

Use insertBefore to add a child node at a specified position in this node's list of children. If the new node is already in the document's node hierarchy, it is first removed from its parent node. If this node is read-only, refChild is a node that is not in this node's child list, newChild was created using a different document, or if newChild is of a nodeType (see page 26318) that is not permitted as a child of this node, insertBefore raises an exception.

newChild is the child node to add. It can be a node that currently has another parent node, or a new node created using the methods of the document interface. If it is of nodeType (see page 26318) DOCUMENT_FRAGMENT_NODE (see page 26367), the entire contents of the document fragment are moved into this node's childNodes (see page 26317) property list.

refChild is a child of this node before which to insert the new node. If refChild is nil (Delphi) or NULL (C++), the new node is added to the end of this node's child list.

insertBefore returns the interface for the newly added child node.

See Also

AppendChild (see page 26308)

ChildNodes (see page 26317)

replaceChild (see page 26315)

removeChild (see page 26315)

NodeType (see page 26318)

OwnerDocument (see page 26319)

196.2.13.2.19 IDOMNode.normalize Method

Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.

Pascal

```
procedure normalize; safecall;
```

C++

```
__fastcall normalize();
```

Description

Call normalize to consolidate adjacent text nodes and remove any empty text nodes in the subtree beneath this node. This method ensures that for all child nodes of this node and of its attributes (see page 26317), only structure nodes (element nodes, comments, processing instructions, CDATA sections, and entity references) separate text nodes and that there are no empty text nodes. This is useful for operations (such as XPointer lookups) that depend on a particular structure in the node hierarchy.

See Also

ChildNodes (see page 26317)

NodeType (see page 26318)

Attributes (see page 26317)

196.2.13.2.20 IDOMNode.removeChild Method

Removes a child node from this node.

Pascal

```
function removeChild(const childNode: IDOMNode): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode removeChild(const IDOMNode childNode);
```

Description

removeChild removes a specified child node from this node, and returns its interface. If this node is read-only, or if the specified child node is not a child of this node, removeChild raises an exception.

childNode the node to remove.

See Also

AppendChild (see page 26308)

ChildNodes (see page 26317)

replaceChild (see page 26315)

196.2.13.2.21 IDOMNode.replaceChild Method

Replaces one of this node's children with a specified node.

Pascal

```
function replaceChild(const newChild: IDOMNode; const oldChild: IDOMNode): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode replaceChild(const IDOMNode newChild, const IDOMNode oldChild);
```

Description

Use replaceChild to remove a child of this node and replace it with the node specified by newChild. If the new node is already in the document's node hierarchy, it is first removed from its parent node.

If this node is read-only, refChild is a node that is not in this node's child list, newChild was created using a different document, or if newChild is of a nodeType (see page 26318) that is not permitted as a child of this node, replaceChild raises an exception.

newChild is the child node that replaces oldChild. It can be a node that currently has another parent node, or a new node created using the methods of the document interface. If it is of nodeType (see page 26318) DOCUMENT_FRAGMENT_NODE (see page 26367), the entire contents of the document fragment are moved into this node's childNodes (see page 26317) property list.

refChild is a child that newChild replaces. If refChild is nil (Delphi) or NULL (C++), the new node is added to the end of this node's child list.

replaceChild returns the interface for refChild (the replaced node).

See Also

AppendChild (see page 26308)

ChildNodes (see page 26317)

replaceChild

insertBefore (see page 26313)

196.2.13.2.22 IDOMNode.set_nodeValue Method

Sets the value of the node.

Pascal

```
procedure set_nodeValue(value: DOMString); safecall;
```

C++

```
__fastcall set_nodeValue(DOMString value);
```

Description

set_set_nodeValue is the write implementation of the set_nodeValue property.

value is the new value for the node.

See Also

set_nodeValue

196.2.13.2.23 IDOMNode.supports Method

Indicates whether the node supports a specified feature.

Pascal

```
function supports(const feature: DOMString; const version: DOMString): WordBool;
```

C++

```
__fastcall WordBool supports(const DOMString feature, const DOMString version);
```

Description

supports returns true if the DOM implementation implements the specified feature and this node supports that feature. Otherwise, supports returns false.

feature is the name of the feature in question.

version is the version of the feature. If no version is specified, supports returns true if any version of the feature is supported.

196.2.13.3 IDOMNode Properties

The properties of the IDOMNode class are listed here.

Public Properties

	Name	Description
R	attributes (see page 26317)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 26317)	This is childNodes, a member of class IDOMNode.
R	firstChild (see page 26317)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 26317)	This is lastChild, a member of class IDOMNode.
R	localName (see page 26318)	This is localName, a member of class IDOMNode.
R	namespaceURI (see page 26318)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 26318)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 26318)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 26318)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 26319)	This is nodeValue, a member of class IDOMNode.

R	ownerDocument (see page 26319)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 26319)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 26319)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 26320)	This is previousSibling, a member of class IDOMNode.

196.2.13.3.1 IDOMNode.attributes Property

Pascal

```
property attributes: IDOMNamedNodeMap;
```

C++

```
__property IDOMNamedNodeMap attributes;
```

Description

This is attributes, a member of class IDOMNode.

196.2.13.3.2 IDOMNode.childNodes Property

Pascal

```
property childNodes: IDOMNodeList;
```

C++

```
__property IDOMNodeList childNodes;
```

Description

This is childNodes, a member of class IDOMNode.

196.2.13.3.3 IDOMNode.firstChild Property

Pascal

```
property firstChild: IDOMNode;
```

C++

```
__property IDOMNode firstChild;
```

Description

This is firstChild, a member of class IDOMNode.

196.2.13.3.4 IDOMNode.lastChild Property

Pascal

```
property lastChild: IDOMNode;
```

C++

```
__property IDOMNode lastChild;
```

Description

This is lastChild, a member of class IDOMNode.

196.2.13.3.5 IDOMNode.localName Property

Pascal

```
property localName: DOMString;
```

C++

```
__property DOMString localName;
```

Description

This is localName, a member of class IDOMNode.

196.2.13.3.6 IDOMNode.namespaceURI Property

Pascal

```
property namespaceURI: DOMString;
```

C++

```
__property DOMString namespaceURI;
```

Description

This is namespaceURI, a member of class IDOMNode.

196.2.13.3.7 IDOMNode.nextSibling Property

Pascal

```
property nextSibling: IDOMNode;
```

C++

```
__property IDOMNode nextSibling;
```

Description

This is nextSibling, a member of class IDOMNode.

196.2.13.3.8 IDOMNode.nodeName Property

Pascal

```
property nodeName: DOMString;
```

C++

```
__property DOMString nodeName;
```

Description

This is nodeName, a member of class IDOMNode.

196.2.13.3.9 IDOMNode.nodeType Property

Pascal

```
property nodeType: DOMNodeType;
```

C++

```
__property DOMNodeType nodeType;
```

Description

This is nodeType, a member of class IDOMNode.

196.2.13.3.10 IDOMNode.nodeValue Property

Pascal

```
property nodeValue: DOMString;
```

C++

```
__property DOMString nodeValue;
```

Description

This is nodeValue, a member of class IDOMNode.

196.2.13.3.11 IDOMNode.ownerDocument Property

Pascal

```
property ownerDocument: IDOMDocument;
```

C++

```
__property IDOMDocument ownerDocument;
```

Description

This is ownerDocument, a member of class IDOMNode.

196.2.13.3.12 IDOMNode.parentNode Property

Pascal

```
property parentNode: IDOMNode;
```

C++

```
__property IDOMNode parentNode;
```

Description

This is parentNode, a member of class IDOMNode.

196.2.13.3.13 IDOMNode.prefix Property

Pascal

```
property prefix: DOMString;
```

C++

```
__property DOMString prefix;
```

Description

This is prefix, a member of class IDOMNode.

196.2.13.3.14 IDOMNode.previousSibling Property

Pascal

```
property previousSibling: IDOMNode;
```

C++

```
__property IDOMNode previousSibling;
```

Description

This is previousSibling, a member of class IDOMNode.

196.2.14 IDOMNodeEx Interface

Class Hierarchy

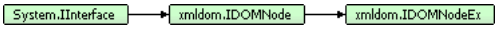

Pascal

```
IDOMNodeEx = interface(IDOMNode);
```

C++

```
__interface IDOMNodeEx : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMNodeEx.

196.2.14.1 IDOMNodeEx Members

The following tables list the members exposed by IDOMNodeEx.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	appendChild (see page 26308)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 26309)	This is cloneNode, a member of class IDOMNode.
	get_attributes (see page 26309)	Returns the value of the get_attributes property.
	get_childNodes (see page 26309)	Returns the value of the get_childNodes property.
	get_firstChild (see page 26309)	Returns the value of the get_firstChild property.
	get_lastChild (see page 26310)	Returns the value of the get_lastChild property.
	get_localName (see page 26310)	Returns the value of the get_localName property.
	get_namespaceURI (see page 26310)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 26311)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 26311)	Returns the value of the get_nodeName property.
	get_nodeType (see page 26311)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 26312)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 26312)	Returns the value of the get_ownerDocument property.

	get_parentNode (see page 26312)	Returns the value of the get_parentNode property.
	get_prefix (see page 26312)	Returns the value of the get_prefix property.
	get_previousSibling (see page 26313)	Returns the value of the get_previousSibling property.
	get_text (see page 26321)	This is get_text, a member of class IDOMNodeEx.
	get_xml (see page 26322)	This is get_xml, a member of class IDOMNodeEx.
	hasChildNodes (see page 26313)	Indicates whether the node has any child nodes.
	insertBefore (see page 26313)	Inserts a new child node before a specified child of the node.
	normalize (see page 26314)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 26315)	Removes a child node from this node.
	replaceChild (see page 26315)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 26316)	Sets the value of the node.
	set_text (see page 26322)	This is set_text, a member of class IDOMNodeEx.
	supports (see page 26316)	Indicates whether the node supports a specified feature.
	transformNode (see page 26322)	This is the overview for the transformNode method overload.

Public Properties

	Name	Description
R	attributes (see page 26317)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 26317)	This is childNodes, a member of class IDOMNode.
R	firstChild (see page 26317)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 26317)	This is lastChild, a member of class IDOMNode.
R	localName (see page 26318)	This is localName, a member of class IDOMNode.
R	namespaceURI (see page 26318)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 26318)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 26318)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 26318)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 26319)	This is nodeValue, a member of class IDOMNode.
R	ownerDocument (see page 26319)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 26319)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 26319)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 26320)	This is previousSibling, a member of class IDOMNode.
	text (see page 26323)	This is text, a member of class IDOMNodeEx.
R	xml (see page 26323)	This is xml, a member of class IDOMNodeEx.

196.2.14.2 IDOMNodeEx Methods

The methods of the IDOMNodeEx class are listed here.

Public Methods

	Name	Description
	get_text (see page 26321)	This is get_text, a member of class IDOMNodeEx.
	get_xml (see page 26322)	This is get_xml, a member of class IDOMNodeEx.
	set_text (see page 26322)	This is set_text, a member of class IDOMNodeEx.
	transformNode (see page 26322)	This is the overview for the transformNode method overload.

196.2.14.2.1 IDOMNodeEx.get_text Method

Pascal

```
function get_text: DOMString; safecall;
```

C++

```
__fastcall DOMString get_text();
```

Description

This is get_text, a member of class IDOMNodeEx.

196.2.14.2.2 IDOMNodeEx.get_xml Method

Pascal

```
function get_xml: DOMString; safecall;
```

C++

```
__fastcall DOMString get_xml();
```

Description

This is get_xml, a member of class IDOMNodeEx.

196.2.14.2.3 IDOMNodeEx.set_text Method

Pascal

```
procedure set_text(const Value: DOMString); safecall;
```

C++

```
__fastcall set_text(const DOMString Value);
```

Description

This is set_text, a member of class IDOMNodeEx.

196.2.14.2.4 transformNode Method

This is the overview for the transformNode method overload.

Overload List

	Name	Description
	IDOMNodeEx.transformNode (IDOMNode, IDOMDocument) (see page 26322)	This is transformNode, a member of class IDOMNodeEx.
	IDOMNodeEx.transformNode (IDOMNode, WideString) (see page 26323)	This is transformNode, a member of class IDOMNodeEx.

196.2.14.2.4.1 IDOMNodeEx.transformNode Method (IDOMNode, IDOMDocument)

Pascal

```
procedure transformNode(const stylesheet: IDOMNode; const output: IDOMDocument); safecall;
overload;
```

C++

```
__fastcall transformNode(const IDOMNode stylesheet, const IDOMDocument output);
```

Description

This is transformNode, a member of class IDOMNodeEx.

196.2.14.2.4.2 IDOMNodeEx.transformNode Method (IDOMNode, WideString)

Pascal

```
procedure transformNode(const stylesheet: IDOMNode; var output: WideString); safecall;
overload;
```

C++

```
__fastcall transformNode(const IDOMNode stylesheet, BSTR output);
```

Description

This is transformNode, a member of class IDOMNodeEx.

196.2.14.3 IDOMNodeEx Properties

The properties of the IDOMNodeEx class are listed here.

Public Properties

	Name	Description
	text (see page 26323)	This is text, a member of class IDOMNodeEx.
R	xml (see page 26323)	This is xml, a member of class IDOMNodeEx.

196.2.14.3.1 IDOMNodeEx.text Property

Pascal

```
property text: DOMString;
```

C++

```
__property DOMString text;
```

Description

This is text, a member of class IDOMNodeEx.

196.2.14.3.2 IDOMNodeEx.xml Property

Pascal

```
property xml: DOMString;
```

C++

```
__property DOMString xml;
```

Description

This is xml, a member of class IDOMNodeEx.

196.2.15 IDOMNodeList Interface

Class Hierarchy

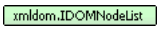

Pascal

```
IDOMNodeList = interface;
```

C++

```
__interface IDOMNodeList;
```

File

xmldom

Description

This is class xmldom.IDOMNodeList.

196.2.15.1 IDOMNodeList Members

The following tables list the members exposed by IDOMNodeList.

Public Methods

	Name	Description
	get_item (see page 26324)	This is get_item, a member of class IDOMNodeList.
	get_length (see page 26324)	This is get_length, a member of class IDOMNodeList.

Public Properties

	Name	Description
R	item (see page 26325)	This is item, a member of class IDOMNodeList.
R	length (see page 26325)	This is length, a member of class IDOMNodeList.

196.2.15.2 IDOMNodeList Methods

The methods of the IDOMNodeList class are listed here.

Public Methods

	Name	Description
	get_item (see page 26324)	This is get_item, a member of class IDOMNodeList.
	get_length (see page 26324)	This is get_length, a member of class IDOMNodeList.

196.2.15.2.1 IDOMNodeList.get_item Method

Pascal

```
function get_item(index: Integer): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_item(int index);
```

Description

This is get_item, a member of class IDOMNodeList.

196.2.15.2.2 IDOMNodeList.get_length Method

Pascal

```
function get_length: Integer; safecall;
```

C++

```
__fastcall int get_length();
```

Description

This is get_length, a member of class IDOMNodeList.

196.2.15.3 IDOMNodeList Properties

The properties of the IDOMNodeList class are listed here.

Public Properties

	Name	Description
R	item (see page 26325)	This is item, a member of class IDOMNodeList.
R	length (see page 26325)	This is length, a member of class IDOMNodeList.

196.2.15.3.1 IDOMNodeList.item Property

Pascal

```
property item [index: Integer]: IDOMNode;
```

C++

```
__property IDOMNode item[int index];
```

Description

This is item, a member of class IDOMNodeList.

196.2.15.3.2 IDOMNodeList.length Property

Pascal

```
property length: Integer;
```

C++

```
__property int length;
```

Description

This is length, a member of class IDOMNodeList.

196.2.16 IDOMNodeSelect Interface

Class Hierarchy

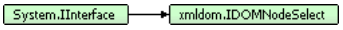

Pascal

```
IDOMNodeSelect = interface(IInterface);
```

C++

```
__interface IDOMNodeSelect : public IInterface;
```

File

xmldom

Description

This is class xmldom.IDOMNodeSelect.

196.2.16.1 IDOMNodeSelect Members

The following tables list the members exposed by IDOMNodeSelect.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	selectNode (see page 26326)	This is selectNode, a member of class IDOMNodeSelect.
	selectNodes (see page 26326)	This is selectNodes, a member of class IDOMNodeSelect.

196.2.16.2 IDOMNodeSelect Methods

The methods of the IDOMNodeSelect class are listed here.

Public Methods

	Name	Description
	selectNode (see page 26326)	This is selectNode, a member of class IDOMNodeSelect.
	selectNodes (see page 26326)	This is selectNodes, a member of class IDOMNodeSelect.

196.2.16.2.1 IDOMNodeSelect.selectNode Method

Pascal

```
function selectNode(const nodePath: WideString): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode selectNode(const BSTR nodePath);
```

Description

This is selectNode, a member of class IDOMNodeSelect.

196.2.16.2.2 IDOMNodeSelect.selectNodes Method

Pascal

```
function selectNodes(const nodePath: WideString): IDOMNodeList; safecall;
```

C++

```
__fastcall IDOMNodeList selectNodes(const BSTR nodePath);
```

Description

This is selectNodes, a member of class IDOMNodeSelect.

196.2.17 IDOMNotation Interface

Class Hierarchy

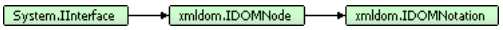

Pascal

```
IDOMNotation = interface(IDOMNode);
```

C++

```
__interface IDOMNotation : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMNotation.

196.2.17.1 IDOMNotation Members

The following tables list the members exposed by IDOMNotation.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	appendChild (see page 26308)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 26309)	This is cloneNode, a member of class IDOMNode.
	get_attributes (see page 26309)	Returns the value of the get_attributes property.
	get_childNodes (see page 26309)	Returns the value of the get_childNodes property.
	get_firstChild (see page 26309)	Returns the value of the get_firstChild property.
	get_lastChild (see page 26310)	Returns the value of the get_lastChild property.
	get_localName (see page 26310)	Returns the value of the get_localName property.
	get_namespaceURI (see page 26310)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 26311)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 26311)	Returns the value of the get_nodeName property.
	get_nodeType (see page 26311)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 26312)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 26312)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 26312)	Returns the value of the get_parentNode property.
	get_prefix (see page 26312)	Returns the value of the get_prefix property.
	get_previousSibling (see page 26313)	Returns the value of the get_previousSibling property.
	get_publicId (see page 26328)	This is get_publicId, a member of class IDOMNotation.
	get_systemId (see page 26328)	This is get_systemId, a member of class IDOMNotation.
	hasChildNodes (see page 26313)	Indicates whether the node has any child nodes.
	insertBefore (see page 26313)	Inserts a new child node before a specified child of the node.
	normalize (see page 26314)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 26315)	Removes a child node from this node.
	replaceChild (see page 26315)	Replaces one of this node's children with a specified node.

	set_nodeValue (see page 26316)	Sets the value of the node.
	supports (see page 26316)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 26317)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 26317)	This is childNodes, a member of class IDOMNode.
R	firstChild (see page 26317)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 26317)	This is lastChild, a member of class IDOMNode.
R	localName (see page 26318)	This is localName, a member of class IDOMNode.
R	namespaceURI (see page 26318)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 26318)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 26318)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 26318)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 26319)	This is nodeValue, a member of class IDOMNode.
R	ownerDocument (see page 26319)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 26319)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 26319)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 26320)	This is previousSibling, a member of class IDOMNode.
R	publicId (see page 26329)	This is publicId, a member of class IDOMNotation.
R	systemId (see page 26329)	This is systemId, a member of class IDOMNotation.

196.2.17.2 IDOMNotation Methods

The methods of the IDOMNotation class are listed here.

Public Methods

	Name	Description
	get_publicId (see page 26328)	This is get_publicId, a member of class IDOMNotation.
	get_systemId (see page 26328)	This is get_systemId, a member of class IDOMNotation.

196.2.17.2.1 IDOMNotation.get_publicId Method

Pascal

```
function get_publicId: DOMString; safecall;
```

C++

```
__fastcall DOMString get_publicId();
```

Description

This is get_publicId, a member of class IDOMNotation.

196.2.17.2.2 IDOMNotation.get_systemId Method

Pascal

```
function get_systemId: DOMString; safecall;
```

C++

```
__fastcall DOMString get_systemId();
```

Description

This is get_systemId, a member of class IDOMNotation.

196.2.17.3 IDOMNotation Properties

The properties of the IDOMNotation class are listed here.

Public Properties

	Name	Description
R	publicId (see page 26329)	This is publicId, a member of class IDOMNotation.
R	systemId (see page 26329)	This is systemId, a member of class IDOMNotation.

196.2.17.3.1 IDOMNotation.publicId Property

Pascal

```
property publicId: DOMString;
```

C++

```
__property DOMString publicId;
```

Description

This is publicId, a member of class IDOMNotation.

196.2.17.3.2 IDOMNotation.systemId Property

Pascal

```
property systemId: DOMString;
```

C++

```
__property DOMString systemId;
```

Description

This is systemId, a member of class IDOMNotation.

196.2.18 IDOMParseError Interface

Class Hierarchy

xmldom.IDOMParseError

Pascal

```
IDOMParseError = interface;
```

C++

```
__interface IDOMParseError;
```

File

xmldom

Description

This is class xmldom.IDOMParseError.

196.2.18.1 IDOMParseError Members

The following tables list the members exposed by IDOMParseError.

Public Methods

	Name	Description
	get_errorCode (see page 26330)	This is get_errorCode, a member of class IDOMParseError.
	get_filepos (see page 26331)	This is get_filepos, a member of class IDOMParseError.
	get_line (see page 26331)	This is get_line, a member of class IDOMParseError.
	get_linepos (see page 26331)	This is get_linepos, a member of class IDOMParseError.
	get_reason (see page 26331)	This is get_reason, a member of class IDOMParseError.
	get_srcText (see page 26331)	This is get_srcText, a member of class IDOMParseError.
	get_url (see page 26332)	This is get_url, a member of class IDOMParseError.

Public Properties

	Name	Description
R	errorCode (see page 26332)	This is errorCode, a member of class IDOMParseError.
R	filePos (see page 26332)	This is filePos, a member of class IDOMParseError.
R	line (see page 26333)	This is line, a member of class IDOMParseError.
R	linePos (see page 26333)	This is linePos, a member of class IDOMParseError.
R	reason (see page 26333)	This is reason, a member of class IDOMParseError.
R	srcText (see page 26333)	This is srcText, a member of class IDOMParseError.
R	url (see page 26333)	This is url, a member of class IDOMParseError.

196.2.18.2 IDOMParseError Methods

The methods of the IDOMParseError class are listed here.

Public Methods

	Name	Description
	get_errorCode (see page 26330)	This is get_errorCode, a member of class IDOMParseError.
	get_filepos (see page 26331)	This is get_filepos, a member of class IDOMParseError.
	get_line (see page 26331)	This is get_line, a member of class IDOMParseError.
	get_linepos (see page 26331)	This is get_linepos, a member of class IDOMParseError.
	get_reason (see page 26331)	This is get_reason, a member of class IDOMParseError.
	get_srcText (see page 26331)	This is get_srcText, a member of class IDOMParseError.
	get_url (see page 26332)	This is get_url, a member of class IDOMParseError.

196.2.18.2.1 IDOMParseError.get_errorCode Method

Pascal

```
function get_errorCode: Integer; safecall;
```

C++

```
__fastcall int get_errorCode();
```

Description

This is get_errorCode, a member of class IDOMParseError.

196.2.18.2.2 IDOMParseError.get_filepos Method

Pascal

```
function get_filepos: Integer; safecall;
```

C++

```
__fastcall int get_filepos();
```

Description

This is get_filepos, a member of class IDOMParseError.

196.2.18.2.3 IDOMParseError.get_line Method

Pascal

```
function get_line: Integer; safecall;
```

C++

```
__fastcall int get_line();
```

Description

This is get_line, a member of class IDOMParseError.

196.2.18.2.4 IDOMParseError.get_linepos Method

Pascal

```
function get_linepos: Integer; safecall;
```

C++

```
__fastcall int get_linepos();
```

Description

This is get_linepos, a member of class IDOMParseError.

196.2.18.2.5 IDOMParseError.get_reason Method

Pascal

```
function get_reason: DOMString; safecall;
```

C++

```
__fastcall DOMString get_reason();
```

Description

This is get_reason, a member of class IDOMParseError.

196.2.18.2.6 IDOMParseError.get_srcText Method

Pascal

```
function get_srcText: DOMString; safecall;
```

C++

```
__fastcall DOMString get_srcText();
```

Description

This is get_srcText, a member of class IDOMParseError.

196.2.18.2.7 IDOMParseError.get_url Method

Pascal

```
function get_url: DOMString; safecall;
```

C++

```
__fastcall DOMString get_url();
```

Description

This is get_url, a member of class IDOMParseError.

196.2.18.3 IDOMParseError Properties

The properties of the IDOMParseError class are listed here.

Public Properties

	Name	Description
R	errorCode (see page 26332)	This is errorCode, a member of class IDOMParseError.
R	filePos (see page 26332)	This is filePos, a member of class IDOMParseError.
R	line (see page 26333)	This is line, a member of class IDOMParseError.
R	linePos (see page 26333)	This is linePos, a member of class IDOMParseError.
R	reason (see page 26333)	This is reason, a member of class IDOMParseError.
R	srcText (see page 26333)	This is srcText, a member of class IDOMParseError.
R	url (see page 26333)	This is url, a member of class IDOMParseError.

196.2.18.3.1 IDOMParseError.errorCode Property

Pascal

```
property errorCode: Integer;
```

C++

```
__property int errorCode;
```

Description

This is errorCode, a member of class IDOMParseError.

196.2.18.3.2 IDOMParseError.filePos Property

Pascal

```
property filePos: Integer;
```

C++

```
__property int filePos;
```

Description

This is filePos, a member of class IDOMParseError.

196.2.18.3.3 IDOMParseError.line Property

Pascal

```
property line: Integer;
```

C++

```
__property int line;
```

Description

This is line, a member of class IDOMParseError.

196.2.18.3.4 IDOMParseError.linePos Property

Pascal

```
property linePos: Integer;
```

C++

```
__property int linePos;
```

Description

This is linePos, a member of class IDOMParseError.

196.2.18.3.5 IDOMParseError.reason Property

Pascal

```
property reason: DOMString;
```

C++

```
__property DOMString reason;
```

Description

This is reason, a member of class IDOMParseError.

196.2.18.3.6 IDOMParseError.srcText Property

Pascal

```
property srcText: DOMString;
```

C++

```
__property DOMString srcText;
```

Description

This is srcText, a member of class IDOMParseError.

196.2.18.3.7 IDOMParseError.url Property

Pascal

```
property url: DOMString;
```

C++

```
__property DOMString url;
```

Description

This is url, a member of class IDOMParseError.

196.2.19 IDOMParseOptions Interface

Class Hierarchy

xmldom.IDOMParseOptions

Pascal

```
IDOMParseOptions = interface;
```

C++

```
__interface IDOMParseOptions;
```

File

xmldom

Description

This is class xmldom.IDOMParseOptions.

196.2.19.1 IDOMParseOptions Members

The following tables list the members exposed by IDOMParseOptions.

Public Methods

	Name	Description
	get_async (see page 26335)	This is get_async, a member of class IDOMParseOptions.
	get_preserveWhiteSpace (see page 26335)	This is get_preserveWhiteSpace, a member of class IDOMParseOptions.
	get_resolveExternals (see page 26335)	This is get_resolveExternals, a member of class IDOMParseOptions.
	get_validate (see page 26335)	This is get_validate, a member of class IDOMParseOptions.
	set_async (see page 26336)	This is set_async, a member of class IDOMParseOptions.
	set_preserveWhiteSpace (see page 26336)	This is set_preserveWhiteSpace, a member of class IDOMParseOptions.
	set_resolveExternals (see page 26336)	This is set_resolveExternals, a member of class IDOMParseOptions.
	set_validate (see page 26336)	This is set_validate, a member of class IDOMParseOptions.

Public Properties

	Name	Description
	async (see page 26337)	This is async, a member of class IDOMParseOptions.
	preserveWhiteSpace (see page 26337)	This is preserveWhiteSpace, a member of class IDOMParseOptions.
	resolveExternals (see page 26337)	This is resolveExternals, a member of class IDOMParseOptions.
	validate (see page 26337)	This is validate, a member of class IDOMParseOptions.

196.2.19.2 IDOMParseOptions Methods

The methods of the IDOMParseOptions class are listed here.

Public Methods

	Name	Description
	get_async (see page 26335)	This is get_async, a member of class IDOMParseOptions.
	get_preserveWhiteSpace (see page 26335)	This is get_preserveWhiteSpace, a member of class IDOMParseOptions.
	get_resolveExternals (see page 26335)	This is get_resolveExternals, a member of class IDOMParseOptions.
	get_validate (see page 26335)	This is get_validate, a member of class IDOMParseOptions.
	set_async (see page 26336)	This is set_async, a member of class IDOMParseOptions.
	set_preserveWhiteSpace (see page 26336)	This is set_preserveWhiteSpace, a member of class IDOMParseOptions.
	set_resolveExternals (see page 26336)	This is set_resolveExternals, a member of class IDOMParseOptions.
	set_validate (see page 26336)	This is set_validate, a member of class IDOMParseOptions.

196.2.19.2.1 IDOMParseOptions.get_async Method

Pascal

```
function get_async: Boolean; safecall;
```

C++

```
__fastcall Boolean get_async();
```

Description

This is get_async, a member of class IDOMParseOptions.

196.2.19.2.2 IDOMParseOptions.get_preserveWhiteSpace Method

Pascal

```
function get_preserveWhiteSpace: Boolean; safecall;
```

C++

```
__fastcall Boolean get_preserveWhiteSpace();
```

Description

This is get_preserveWhiteSpace, a member of class IDOMParseOptions.

196.2.19.2.3 IDOMParseOptions.get_resolveExternals Method

Pascal

```
function get_resolveExternals: Boolean; safecall;
```

C++

```
__fastcall Boolean get_resolveExternals();
```

Description

This is get_resolveExternals, a member of class IDOMParseOptions.

196.2.19.2.4 IDOMParseOptions.get_validate Method

Pascal

```
function get_validate: Boolean; safecall;
```

C++

```
__fastcall Boolean get_validate();
```

Description

This is get_validate, a member of class IDOMParseOptions.

196.2.19.2.5 IDOMParseOptions.set_async Method

Pascal

```
procedure set_async(Value: Boolean); safecall;
```

C++

```
__fastcall set_async(Boolean Value);
```

Description

This is set_async, a member of class IDOMParseOptions.

196.2.19.2.6 IDOMParseOptions.set_preserveWhiteSpace Method

Pascal

```
procedure set_preserveWhiteSpace(Value: Boolean); safecall;
```

C++

```
__fastcall set_preserveWhiteSpace(Boolean Value);
```

Description

This is set_preserveWhiteSpace, a member of class IDOMParseOptions.

196.2.19.2.7 IDOMParseOptions.set_resolveExternals Method

Pascal

```
procedure set_resolveExternals(Value: Boolean); safecall;
```

C++

```
__fastcall set_resolveExternals(Boolean Value);
```

Description

This is set_resolveExternals, a member of class IDOMParseOptions.

196.2.19.2.8 IDOMParseOptions.set_validate Method

Pascal

```
procedure set_validate(Value: Boolean); safecall;
```

C++

```
__fastcall set_validate(Boolean Value);
```

Description

This is set_validate, a member of class IDOMParseOptions.

196.2.19.3 IDOMParseOptions Properties

The properties of the IDOMParseOptions class are listed here.

Public Properties

	Name	Description
	async (see page 26337)	This is async, a member of class IDOMParseOptions.
	preserveWhiteSpace (see page 26337)	This is preserveWhiteSpace, a member of class IDOMParseOptions.
	resolveExternals (see page 26337)	This is resolveExternals, a member of class IDOMParseOptions.
	validate (see page 26337)	This is validate, a member of class IDOMParseOptions.

196.2.19.3.1 IDOMParseOptions.async Property

Pascal

```
property async: Boolean;
```

C++

```
__property Boolean async;
```

Description

This is async, a member of class IDOMParseOptions.

196.2.19.3.2 IDOMParseOptions.preserveWhiteSpace Property

Pascal

```
property preserveWhiteSpace: Boolean;
```

C++

```
__property Boolean preserveWhiteSpace;
```

Description

This is preserveWhiteSpace, a member of class IDOMParseOptions.

196.2.19.3.3 IDOMParseOptions.resolveExternals Property

Pascal

```
property resolveExternals: Boolean;
```

C++

```
__property Boolean resolveExternals;
```

Description

This is resolveExternals, a member of class IDOMParseOptions.

196.2.19.3.4 IDOMParseOptions.validate Property

Pascal

```
property validate: Boolean;
```

C++

```
__property Boolean validate;
```

Description

This is validate, a member of class IDOMParseOptions.

196.2.20 IDOMPersist Interface

Class Hierarchy

xmldom.IDOMPersist

Pascal

```
IDOMPersist = interface;
```

C++

```
__interface IDOMPersist;
```

File

xmldom

Description

This is class xmldom.IDOMPersist.

196.2.20.1 IDOMPersist Members

The following tables list the members exposed by IDOMPersist.

Public Methods

	Name	Description
	asyncLoadState (see page 26339)	This is asyncLoadState, a member of class IDOMPersist.
	get_xml (see page 26339)	This is get_xml, a member of class IDOMPersist.
	load (see page 26339)	This is load, a member of class IDOMPersist.
	loadFromStream (see page 26339)	This is the overview for the loadFromStream method overload.
	loadxml (see page 26340)	This is loadxml, a member of class IDOMPersist.
	save (see page 26340)	This is save, a member of class IDOMPersist.
	saveToStream (see page 26340)	This is the overview for the saveToStream method overload.
	set_OnAsyncLoad (see page 26341)	This is set_OnAsyncLoad, a member of class IDOMPersist.

Public Properties

	Name	Description
R	xml (see page 26341)	This is xml, a member of class IDOMPersist.

196.2.20.2 IDOMPersist Methods

The methods of the IDOMPersist class are listed here.

Public Methods

	Name	Description
	asyncLoadState (see page 26339)	This is asyncLoadState, a member of class IDOMPersist.
	get_xml (see page 26339)	This is get_xml, a member of class IDOMPersist.
	load (see page 26339)	This is load, a member of class IDOMPersist.
	loadFromStream (see page 26339)	This is the overview for the loadFromStream method overload.
	loadxml (see page 26340)	This is loadxml, a member of class IDOMPersist.
	save (see page 26340)	This is save, a member of class IDOMPersist.
	saveToStream (see page 26340)	This is the overview for the saveToStream method overload.

	set_OnAsyncLoad (see page 26341)	This is set_OnAsyncLoad, a member of class IDOMPersist.

196.2.20.2.1 IDOMPersist.asyncLoadState Method

Pascal

```
function asyncLoadState: Integer; safecall;
```

C++

```
__fastcall int asyncLoadState();
```

Description

This is asyncLoadState, a member of class IDOMPersist.

196.2.20.2.2 IDOMPersist.get_xml Method

Pascal

```
function get_xml: DOMString; safecall;
```

C++

```
__fastcall DOMString get_xml();
```

Description

This is get_xml, a member of class IDOMPersist.

196.2.20.2.3 IDOMPersist.load Method

Pascal

```
function load(source: OleVariant): WordBool; safecall;
```

C++

```
__fastcall WordBool load(OleVariant source);
```

Description

This is load, a member of class IDOMPersist.

196.2.20.2.4 loadFromStream Method

This is the overview for the loadFromStream method overload.

Overload List

	Name	Description
	IDOMPersist.loadFromStream (IStream) (see page 26339)	This is loadFromStream, a member of class IDOMPersist.
	IDOMPersist.loadFromStream (TStream) (see page 26340)	This is loadFromStream, a member of class IDOMPersist.

196.2.20.2.4.1 IDOMPersist.loadFromStream Method (IStream)

Pascal

```
function loadFromStream(const stream: IStream): WordBool; safecall; overload;
```

C++

```
__fastcall WordBool loadFromStream(const IStream stream);
```

Description

This is loadFromStream, a member of class IDOMPersist.

196.2.20.2.4.2 IDOMPersist.loadFromStream Method (TStream)

Pascal

```
function loadFromStream(const stream: TStream): WordBool; safecall; overload;
```

C++

```
__fastcall WordBool loadFromStream(const TStream stream);
```

Description

This is loadFromStream, a member of class IDOMPersist.

196.2.20.2.5 IDOMPersist.loadxml Method

Pascal

```
function loadxml(const Value: DOMString): WordBool; safecall;
```

C++

```
__fastcall WordBool loadxml(const DOMString Value);
```

Description

This is loadxml, a member of class IDOMPersist.

196.2.20.2.6 IDOMPersist.save Method

Pascal

```
procedure save(destination: OleVariant); safecall;
```

C++

```
__fastcall save(OleVariant destination);
```

Description

This is save, a member of class IDOMPersist.

196.2.20.2.7 saveToStream Method

This is the overview for the saveToStream method overload.

Overload List

	Name	Description
	IDOMPersist.saveToStream (IStream) (see page 26341)	This is saveToStream, a member of class IDOMPersist.
	IDOMPersist.saveToStream (TStream) (see page 26341)	This is saveToStream, a member of class IDOMPersist.

196.2.20.2.7.1 IDOMPersist.saveToStream Method (IStream)

Pascal

```
procedure saveToStream(const stream: IStream); safecall; overload;
```

C++

```
__fastcall saveToStream(const IStream stream);
```

Description

This is saveToStream, a member of class IDOMPersist.

196.2.20.2.7.2 IDOMPersist.saveToStream Method (TStream)

Pascal

```
procedure saveToStream(const stream: TStream); safecall; overload;
```

C++

```
__fastcall saveToStream(const TStream stream);
```

Description

This is saveToStream, a member of class IDOMPersist.

196.2.20.2.8 IDOMPersist.set_OnAsyncLoad Method

Pascal

```
procedure set_OnAsyncLoad(const Sender: TObject; EventHandler: TAsyncEventHandler); safecall;
```

C++

```
__fastcall set_OnAsyncLoad(const TObject * Sender, TAsyncEventHandler EventHandler);
```

Description

This is set_OnAsyncLoad, a member of class IDOMPersist.

196.2.20.3 IDOMPersist Properties

The properties of the IDOMPersist class are listed here.

Public Properties

	Name	Description
R	xml (see page 26341)	This is xml, a member of class IDOMPersist.

196.2.20.3.1 IDOMPersist.xml Property

Pascal

```
property xml: DOMString;
```

C++

```
__property DOMString xml;
```

Description

This is xml, a member of class IDOMPersist.

196.2.21 IDOMProcessingInstruction Interface

Class Hierarchy

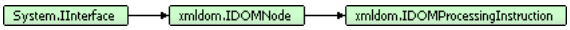

Pascal

```
IDOMProcessingInstruction = interface(IDOMNode);
```

C++

```
__interface IDOMProcessingInstruction : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMProcessingInstruction.

196.2.21.1 IDOMProcessingInstruction Members

The following tables list the members exposed by IDOMProcessingInstruction.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	appendChild (see page 26308)	This is appendChild, a member of class IDOMNode.
	cloneNode (see page 26309)	This is cloneNode, a member of class IDOMNode.
	get_attributes (see page 26309)	Returns the value of the get_attributes property.
	get_childNodes (see page 26309)	Returns the value of the get_childNodes property.
	get_data (see page 26343)	This is get_data, a member of class IDOMProcessingInstruction.
	get_firstChild (see page 26309)	Returns the value of the get_firstChild property.
	get_lastChild (see page 26310)	Returns the value of the get_lastChild property.
	get_localName (see page 26310)	Returns the value of the get_localName property.
	get_namespaceURI (see page 26310)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 26311)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 26311)	Returns the value of the get_nodeName property.
	get_nodeType (see page 26311)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 26312)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 26312)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 26312)	Returns the value of the get_parentNode property.
	get_prefix (see page 26312)	Returns the value of the get_prefix property.
	get_previousSibling (see page 26313)	Returns the value of the get_previousSibling property.
	get_target (see page 26343)	This is get_target, a member of class IDOMProcessingInstruction.
	hasChildNodes (see page 26313)	Indicates whether the node has any child nodes.
	insertBefore (see page 26313)	Inserts a new child node before a specified child of the node.
	normalize (see page 26314)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 26315)	Removes a child node from this node.
	replaceChild (see page 26315)	Replaces one of this node's children with a specified node.

	set_data (see page 26344)	This is set_data, a member of class IDOMProcessingInstruction.
	set_nodeValue (see page 26316)	Sets the value of the node.
	supports (see page 26316)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 26317)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 26317)	This is childNodes, a member of class IDOMNode.
	data (see page 26344)	This is data, a member of class IDOMProcessingInstruction.
R	firstChild (see page 26317)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 26317)	This is lastChild, a member of class IDOMNode.
R	localName (see page 26318)	This is localName, a member of class IDOMNode.
R	namespaceURI (see page 26318)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 26318)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 26318)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 26318)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 26319)	This is nodeValue, a member of class IDOMNode.
R	ownerDocument (see page 26319)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 26319)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 26319)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 26320)	This is previousSibling, a member of class IDOMNode.
R	target (see page 26344)	This is target, a member of class IDOMProcessingInstruction.

196.2.21.2 IDOMProcessingInstruction Methods

The methods of the IDOMProcessingInstruction class are listed here.

Public Methods

	Name	Description
	get_data (see page 26343)	This is get_data, a member of class IDOMProcessingInstruction.
	get_target (see page 26343)	This is get_target, a member of class IDOMProcessingInstruction.
	set_data (see page 26344)	This is set_data, a member of class IDOMProcessingInstruction.

196.2.21.2.1 IDOMProcessingInstruction.get_data Method

Pascal

```
function get_data: DOMString; safecall;
```

C++

```
__fastcall DOMString get_data();
```

Description

This is get_data, a member of class IDOMProcessingInstruction.

196.2.21.2.2 IDOMProcessingInstruction.get_target Method

Pascal

```
function get_target: DOMString; safecall;
```

C++

```
__fastcall DOMString get_target();
```

Description

This is get_target, a member of class IDOMProcessingInstruction.

196.2.21.2.3 IDOMProcessingInstruction.set_data Method

Pascal

```
procedure set_data(const value: DOMString); safecall;
```

C++

```
__fastcall set_data(const DOMString value);
```

Description

This is set_data, a member of class IDOMProcessingInstruction.

196.2.21.3 IDOMProcessingInstruction Properties

The properties of the IDOMProcessingInstruction class are listed here.

Public Properties

	Name	Description
	data (see page 26344)	This is data, a member of class IDOMProcessingInstruction.
R	target (see page 26344)	This is target, a member of class IDOMProcessingInstruction.

196.2.21.3.1 IDOMProcessingInstruction.data Property

Pascal

```
property data: DOMString;
```

C++

```
__property DOMString data;
```

Description

This is data, a member of class IDOMProcessingInstruction.

196.2.21.3.2 IDOMProcessingInstruction.target Property

Pascal

```
property target: DOMString;
```

C++

```
__property DOMString target;
```

Description

This is target, a member of class IDOMProcessingInstruction.

196.2.22 IDOMText Interface

Class Hierarchy

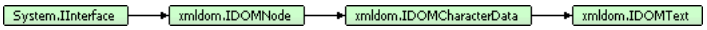

Pascal

```
IDOMText = interface(IDOMCharacterData);
```

C++

```
__interface IDOMText : public IDOMCharacterData;
```

File

xmldom

Description

This is class xmldom.IDOMText.

196.2.22.1 IDOMText Members

The following tables list the members exposed by IDOMText.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	appendChild (see page 26308)	This is appendChild, a member of class IDOMNode.
	appendData (see page 26276)	This is appendData, a member of class IDOMCharacterData.
	cloneNode (see page 26309)	This is cloneNode, a member of class IDOMNode.
	deleteData (see page 26277)	This is deleteData, a member of class IDOMCharacterData.
	get_attributes (see page 26309)	Returns the value of the get_attributes property.
	get_childNodes (see page 26309)	Returns the value of the get_childNodes property.
	get_data (see page 26277)	This is get_data, a member of class IDOMCharacterData.
	get_firstChild (see page 26309)	Returns the value of the get_firstChild property.
	get_lastChild (see page 26310)	Returns the value of the get_lastChild property.
	get_length (see page 26277)	This is get_length, a member of class IDOMCharacterData.
	get_localName (see page 26310)	Returns the value of the get_localName property.
	get_namespaceURI (see page 26310)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 26311)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 26311)	Returns the value of the get_nodeName property.
	get_nodeType (see page 26311)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 26312)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 26312)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 26312)	Returns the value of the get_parentNode property.
	get_prefix (see page 26312)	Returns the value of the get_prefix property.
	get_previousSibling (see page 26313)	Returns the value of the get_previousSibling property.
	hasChildNodes (see page 26313)	Indicates whether the node has any child nodes.
	insertBefore (see page 26313)	Inserts a new child node before a specified child of the node.
	insertData (see page 26277)	This is insertData, a member of class IDOMCharacterData.
	normalize (see page 26314)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.

	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 26315)	Removes a child node from this node.
	replaceChild (see page 26315)	Replaces one of this node's children with a specified node.
	replaceData (see page 26277)	This is replaceData, a member of class IDOMCharacterData.
	set_data (see page 26278)	This is set_data, a member of class IDOMCharacterData.
	set_nodeValue (see page 26316)	Sets the value of the node.
	splitText (see page 26346)	This is splitText, a member of class IDOMText.
	substringData (see page 26278)	This is substringData, a member of class IDOMCharacterData.
	supports (see page 26316)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 26317)	This is attributes, a member of class IDOMNode.
R	childNodes (see page 26317)	This is childNodes, a member of class IDOMNode.
	data (see page 26278)	This is data, a member of class IDOMCharacterData.
R	firstChild (see page 26317)	This is firstChild, a member of class IDOMNode.
R	lastChild (see page 26317)	This is lastChild, a member of class IDOMNode.
R	length (see page 26278)	This is length, a member of class IDOMCharacterData.
R	localName (see page 26318)	This is localName, a member of class IDOMNode.
R	namespaceURI (see page 26318)	This is namespaceURI, a member of class IDOMNode.
R	nextSibling (see page 26318)	This is nextSibling, a member of class IDOMNode.
R	nodeName (see page 26318)	This is nodeName, a member of class IDOMNode.
R	nodeType (see page 26318)	This is nodeType, a member of class IDOMNode.
	nodeValue (see page 26319)	This is nodeValue, a member of class IDOMNode.
R	ownerDocument (see page 26319)	This is ownerDocument, a member of class IDOMNode.
R	parentNode (see page 26319)	This is parentNode, a member of class IDOMNode.
R	prefix (see page 26319)	This is prefix, a member of class IDOMNode.
R	previousSibling (see page 26320)	This is previousSibling, a member of class IDOMNode.

196.2.22.2 IDOMText Methods

The methods of the IDOMText class are listed here.

Public Methods

	Name	Description
	splitText (see page 26346)	This is splitText, a member of class IDOMText.

196.2.22.2.1 IDOMText.splitText Method

Pascal

```
function splitText(offset: Integer): IDOMText; safecall;
```

C++

```
__fastcall IDOMText splitText(int offset);
```

Description

This is splitText, a member of class IDOMText.

196.2.23 IDOMXMLProlog Interface

Class Hierarchy

xmldom.IDOMXMLProlog

Pascal

```
IDOMXMLProlog = interface;
```

C++

```
__interface IDOMXMLProlog;
```

File

xmldom

Description

This is class xmldom.IDOMXMLProlog.

196.2.23.1 IDOMXMLProlog Members

The following tables list the members exposed by IDOMXMLProlog.

Public Methods

	Name	Description
	get_Encoding (see page 26348)	This is get_Encoding, a member of class IDOMXMLProlog.
	get_Standalone (see page 26348)	This is get_Standalone, a member of class IDOMXMLProlog.
	get_Version (see page 26348)	This is get_Version, a member of class IDOMXMLProlog.
	set_Encoding (see page 26348)	This is set_Encoding, a member of class IDOMXMLProlog.
	set_Standalone (see page 26348)	This is set_Standalone, a member of class IDOMXMLProlog.
	set_Version (see page 26349)	This is set_Version, a member of class IDOMXMLProlog.

Public Properties

	Name	Description
	Encoding (see page 26349)	This is Encoding, a member of class IDOMXMLProlog.
	Standalone (see page 26349)	This is Standalone, a member of class IDOMXMLProlog.
	Version (see page 26349)	This is Version, a member of class IDOMXMLProlog.

196.2.23.2 IDOMXMLProlog Methods

The methods of the IDOMXMLProlog class are listed here.

Public Methods

	Name	Description
	get_Encoding (see page 26348)	This is get_Encoding, a member of class IDOMXMLProlog.
	get_Standalone (see page 26348)	This is get_Standalone, a member of class IDOMXMLProlog.
	get_Version (see page 26348)	This is get_Version, a member of class IDOMXMLProlog.
	set_Encoding (see page 26348)	This is set_Encoding, a member of class IDOMXMLProlog.
	set_Standalone (see page 26348)	This is set_Standalone, a member of class IDOMXMLProlog.
	set_Version (see page 26349)	This is set_Version, a member of class IDOMXMLProlog.

196.2.23.2.1 IDOMXMLProlog.get_Encoding Method

Pascal

```
function get_Encoding: DOMString; safecall;
```

C++

```
__fastcall DOMString get_Encoding();
```

Description

This is get_Encoding, a member of class IDOMXMLProlog.

196.2.23.2.2 IDOMXMLProlog.get_Standalone Method

Pascal

```
function get_Standalone: DOMString; safecall;
```

C++

```
__fastcall DOMString get_Standalone();
```

Description

This is get_Standalone, a member of class IDOMXMLProlog.

196.2.23.2.3 IDOMXMLProlog.get_Version Method

Pascal

```
function get_Version: DOMString; safecall;
```

C++

```
__fastcall DOMString get_Version();
```

Description

This is get_Version, a member of class IDOMXMLProlog.

196.2.23.2.4 IDOMXMLProlog.set_Encoding Method

Pascal

```
procedure set_Encoding(const Value: DOMString); safecall;
```

C++

```
__fastcall set_Encoding(const DOMString Value);
```

Description

This is set_Encoding, a member of class IDOMXMLProlog.

196.2.23.2.5 IDOMXMLProlog.set_Standalone Method

Pascal

```
procedure set_Standalone(const Value: DOMString); safecall;
```

C++

```
__fastcall set_Standalone(const DOMString Value);
```

Description

This is set_Standalone, a member of class IDOMXMLProlog.

196.2.23.2.6 IDOMXMLProlog.set_Version Method

Pascal

```
procedure set_Version(const Value: DOMString); safecall;
```

C++

```
__fastcall set_Version(const DOMString Value);
```

Description

This is set_Version, a member of class IDOMXMLProlog.

196.2.23.3 IDOMXMLProlog Properties

The properties of the IDOMXMLProlog class are listed here.

Public Properties

	Name	Description
	Encoding (see page 26349)	This is Encoding, a member of class IDOMXMLProlog.
	Standalone (see page 26349)	This is Standalone, a member of class IDOMXMLProlog.
	Version (see page 26349)	This is Version, a member of class IDOMXMLProlog.

196.2.23.3.1 IDOMXMLProlog.Encoding Property

Pascal

```
property Encoding: DOMString;
```

C++

```
__property DOMString Encoding;
```

Description

This is Encoding, a member of class IDOMXMLProlog.

196.2.23.3.2 IDOMXMLProlog.Standalone Property

Pascal

```
property Standalone: DOMString;
```

C++

```
__property DOMString Standalone;
```

Description

This is Standalone, a member of class IDOMXMLProlog.

196.2.23.3.3 IDOMXMLProlog.Version Property

Pascal

```
property Version: DOMString;
```

C++

```
__property DOMString Version;
```

Description

This is Version, a member of class IDOMXMLProlog.

196.2.24 IDOMXSLProcessor Interface

Class Hierarchy

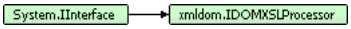

Pascal

```
IDOMXSLProcessor = interface(IInterface);
```

C++

```
__interface IDOMXSLProcessor : public IInterface;
```

File

xmldom

Description

This is class xmldom.IDOMXSLProcessor.

196.2.24.1 IDOMXSLProcessor Members

The following tables list the members exposed by IDOMXSLProcessor.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	Get_input (see page 26351)	This is Get_input, a member of class IDOMXSLProcessor.
	Get_output (see page 26351)	This is Get_output, a member of class IDOMXSLProcessor.
	Get_stylesheet (see page 26351)	This is Get_stylesheet, a member of class IDOMXSLProcessor.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	reset (see page 26352)	This is reset, a member of class IDOMXSLProcessor.
	Set_input (see page 26352)	This is Set_input, a member of class IDOMXSLProcessor.
	Set_output (see page 26352)	This is Set_output, a member of class IDOMXSLProcessor.
	setParameter (see page 26352)	This is setParameter, a member of class IDOMXSLProcessor.
	transform (see page 26352)	This is transform, a member of class IDOMXSLProcessor.

Public Properties

	Name	Description
	input (see page 26353)	This is input, a member of class IDOMXSLProcessor.
	output (see page 26353)	This is output, a member of class IDOMXSLProcessor.
R	stylesheet (see page 26353)	This is stylesheet, a member of class IDOMXSLProcessor.

196.2.24.2 IDOMXSLProcessor Methods

The methods of the IDOMXSLProcessor class are listed here.

Public Methods

	Name	Description
	Get_input (see page 26351)	This is Get_input, a member of class IDOMXSLProcessor.
	Get_output (see page 26351)	This is Get_output, a member of class IDOMXSLProcessor.
	Get_stylesheet (see page 26351)	This is Get_stylesheet, a member of class IDOMXSLProcessor.
	reset (see page 26352)	This is reset, a member of class IDOMXSLProcessor.
	Set_input (see page 26352)	This is Set_input, a member of class IDOMXSLProcessor.
	Set_output (see page 26352)	This is Set_output, a member of class IDOMXSLProcessor.
	setParameter (see page 26352)	This is setParameter, a member of class IDOMXSLProcessor.
	transform (see page 26352)	This is transform, a member of class IDOMXSLProcessor.

196.2.24.2.1 IDOMXSLProcessor.Get_input Method

Pascal

```
function Get_input: OleVariant; safecall;
```

C++

```
__fastcall OleVariant Get_input();
```

Description

This is Get_input, a member of class IDOMXSLProcessor.

196.2.24.2.2 IDOMXSLProcessor.Get_output Method

Pascal

```
function Get_output: OleVariant; safecall;
```

C++

```
__fastcall OleVariant Get_output();
```

Description

This is Get_output, a member of class IDOMXSLProcessor.

196.2.24.2.3 IDOMXSLProcessor.Get_stylesheet Method

Pascal

```
function Get_stylesheet: IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode Get_stylesheet();
```

Description

This is Get_stylesheet, a member of class IDOMXSLProcessor.

196.2.24.2.4 IDOMXSLProcessor.reset Method

Pascal

```
procedure reset; safecall;
```

C++

```
__fastcall reset();
```

Description

This is reset, a member of class IDOMXSLProcessor.

196.2.24.2.5 IDOMXSLProcessor.Set_input Method

Pascal

```
procedure Set_input(const value: OleVariant); safecall;
```

C++

```
__fastcall Set_input(const OleVariant value);
```

Description

This is Set_input, a member of class IDOMXSLProcessor.

196.2.24.2.6 IDOMXSLProcessor.Set_output Method

Pascal

```
procedure Set_output(const value: OleVariant); safecall;
```

C++

```
__fastcall Set_output(const OleVariant value);
```

Description

This is Set_output, a member of class IDOMXSLProcessor.

196.2.24.2.7 IDOMXSLProcessor.setParameter Method

Pascal

```
procedure setParameter(const Name: DOMString; Value: OleVariant; const namespaceURI:
DOMString); safecall;
```

C++

```
__fastcall setParameter(const DOMString Name, OleVariant Value, const DOMString namespaceURI);
```

Description

This is setParameter, a member of class IDOMXSLProcessor.

196.2.24.2.8 IDOMXSLProcessor.transform Method

Pascal

```
function transform: WordBool; safecall;
```

C++

```
__fastcall WordBool transform();
```

Description

This is transform, a member of class IDOMXSLProcessor.

196.2.24.3 IDOMXSLProcessor Properties

The properties of the IDOMXSLProcessor class are listed here.

Public Properties

	Name	Description
	input (see page 26353)	This is input, a member of class IDOMXSLProcessor.
	output (see page 26353)	This is output, a member of class IDOMXSLProcessor.
R	stylesheet (see page 26353)	This is stylesheet, a member of class IDOMXSLProcessor.

196.2.24.3.1 IDOMXSLProcessor.input Property

Pascal

```
property input: OleVariant;
```

C++

```
__property OleVariant input;
```

Description

This is input, a member of class IDOMXSLProcessor.

196.2.24.3.2 IDOMXSLProcessor.output Property

Pascal

```
property output: OleVariant;
```

C++

```
__property OleVariant output;
```

Description

This is output, a member of class IDOMXSLProcessor.

196.2.24.3.3 IDOMXSLProcessor.stylesheet Property

Pascal

```
property stylesheet: IDOMNode;
```

C++

```
__property IDOMNode stylesheet;
```

Description

This is stylesheet, a member of class IDOMXSLProcessor.

196.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DOMVendorNotSupported (see page 26354)	This is function xmldom.DOMVendorNotSupported.
	ExtractLocalName (see page 26354)	Strips the namespace prefix, if present, from the tag name of an XML node.
	ExtractPrefix (see page 26355)	Returns the namespace prefix of an XML node's tag name.
	GetDOM (see page 26356)	Returns the top-level DOM interface for a registered DOM implementation.
	GetDOMNodeEx (see page 26356)	This is function xmldom.GetDOMNodeEx.
	GetDOMVendor (see page 26357)	Returns the TDOMVendor (see page 26263) object for a specified DOM vendor.
	IsPrefixed (see page 26357)	Indicates whether an XML tag name includes a namespace prefix.
	MakeNodeName (see page 26358)	Assembles a full tag name from a local name and namespace prefix.
	NodeMatches (see page 26358)	Indicates whether a specified node matches a given namespace and tag name.
	RegisterDOMVendor (see page 26359)	Registers an implementation of the DOM interfaces so that it is available to the TXMLDocument class.
	SameNamespace (see page 26359)	Indicates whether a specified node is defined within a specified namespace.
	SameNamespace (see page 26359)	Indicates whether a specified node is defined within a specified namespace.
	UnRegisterDOMVendor (see page 26360)	Unregisters a DOM implementation that was registered using RegisterDOMVendor (see page 26359) .

196.3.1 xmldom.DOMVendorNotSupported Function

Pascal

```
procedure DOMVendorNotSupported(const PropOrMethod: string; const VendorName: string);
```

C++

```
DOMVendorNotSupported(const AnsiString PropOrMethod, const AnsiString VendorName);
```

File

xmldom

Description

This is function xmldom.DOMVendorNotSupported.

196.3.2 xmldom.ExtractLocalName Function

Strips the namespace prefix, if present, from the tag name of an XML node.

Pascal

```
function ExtractLocalName(const AName: DOMString): DOMString;
```

C++

```
DOMString ExtractLocalName(const DOMString AName);
```

File

xmldom

Description

Call ExtractLocalName to convert a full tag name for an XML node into the corresponding local name.

AName is the tag name to be converted.

ExtractLocalName returns the value of AName with any namespace prefix stripped away.

See Also

ExtractPrefix (see page 26355)

IsPrefixed (see page 26357)

NodeMatches (see page 26358)

IXMLNode

SameNamespace (see page 26359)

196.3.3 xmldom.ExtractPrefix Function

Returns the namespace prefix of an XML node's tag name.

Pascal

```
function ExtractPrefix(const AName: DOMString): DOMString;
```

C++

```
DOMString ExtractPrefix(const DOMString AName);
```

File

xmldom

Description

Call ExtractPrefix to determine the namespace prefix, if any, of a full tag name for an XML node.

AName is the tag name of the XML node.

ExtractPrefix returns the namespace prefix, not including the colon (:) that separates the namespace from the local name of the XML node. If the tag name does not include a namespace prefix, ExtractPrefix returns an empty string.

See Also

ExtractLocalName (see page 26354)

IsPrefixed (see page 26357)

SameNamespace (see page 26359)

NodeMatches (see page 26358)

196.3.4 xmldom.GetDOM Function

Returns the top-level DOM interface for a registered DOM implementation.

Pascal

```
function GetDOM(const VendorDesc: string = ''): IDOMImplementation;
```

C++

```
IDOMImplementation GetDOM(const AnsiString VendorDesc = '');
```

File

xmldom

Description

Call GetDOM to obtain an IDOMImplementation (see page 26301) interface that you can use for parsing and editing XML documents. The IDOMImplementation (see page 26301) interface is the top-level DOM interface, which you can use to check for the presence of a specific feature or to obtain an interface for working with a DOM document.

VendorDesc is a short descriptive name that identifies the DOM implementation. This name is returned by the Description method of the DOM implementation factory that is used to register the implementation.

Note: The DOM implementation must be registered before you can access it using GetDOM. To register a DOM implementation, use the RegisterDOMVendor (see page 26359) procedure.

See Also

RegisterDOMVendor (see page 26359)

DOMVendors (see page 26365)

GetDOMVendor (see page 26357)

TXMLDocument

Using the Document Object Model

196.3.5 xmldom.GetDOMNodeEx Function

Pascal

```
function GetDOMNodeEx(const Node: IDOMNode): IDOMNodeEx;
```

C++

```
IDOMNodeEx GetDOMNodeEx(const IDOMNode Node);
```

File

xmldom

Description

This is function xmldom.GetDOMNodeEx.

196.3.6 xmldom.GetDOMVendor Function

Returns the TDOMVendor (see page 26263) object for a specified DOM vendor.

Pascal

```
function GetDOMVendor(VendorDesc: string): TDOMVendor;
```

C++

```
TDOMVendor GetDOMVendor(AnsiString VendorDesc);
```

File

xmldom

Description

Call GetDOMVendor to obtain the TDOMVendor (see page 26263) object that represents a registered DOM implementation. You can then assign this return value as the DOMVendor property of TXMLDocument, or call this object's DOMImplementation method to access the IDOMImplementation (see page 26301) interface for the DOM implementation.

VendorDesc is the descriptive string that identifies the particular DOM implementation that is desired. This is the value returned by the TDOMVendor (see page 26263) object's Description method.

See Also

RegisterDOMVendor (see page 26359)

DOMVendors (see page 26365)

TXMLDocument

196.3.7 xmldom.IsPrefixed Function

Indicates whether an XML tag name includes a namespace prefix.

Pascal

```
function IsPrefixed(const AName: DOMString): Boolean;
```

C++

```
Boolean IsPrefixed(const DOMString AName);
```

File

xmldom

Description

Call IsPrefixed to determine whether a tag name includes a namespace prefix.

AName is the tag name of the XML node.

IsPrefixed returns true if AName includes a namespace prefix followed by the colon (:) separator character, false otherwise.

See Also

ExtractLocalName (see page 26354)

ExtractPrefix (see page 26355)

IXMLNode

196.3.8 xmldom.MakeNodeName Function

Assembles a full tag name from a local name and namespace prefix.

Pascal

```
function MakeNodeName(const Prefix: DOMString; const LocalName: DOMString): DOMString;
```

C++

```
DOMString MakeNodeName(const DOMString Prefix, const DOMString LocalName);
```

File

xmldom

Description

Call MakeNodeName to create a tag name from a namespace prefix and a local tag name.

Prefix is the namespace prefix.

LocalName is the local name for the node.

MakeNodeName returns the full tag name that consists of the namespace prefix, followed by a colon and then the local name.

See Also

ExtractPrefix (see page 26355)

ExtractLocalName (see page 26354)

196.3.9 xmldom.NodeMatches Function

Indicates whether a specified node matches a given namespace and tag name.

Pascal

```
function NodeMatches(const Node: IDOMNode; const TagName: DOMString; const NamespaceURI:
DOMString): Boolean; overload;
```

C++

```
Boolean NodeMatches(const IDOMNode Node, const DOMString TagName, const DOMString
NamespaceURI);
```

File

xmldom

Description

Call NodeMatches to determine whether the node with the interface specified by Node refers to the same node as is described by the TagName and NamespaceURI parameters.

NodeMatches returns true if

The value of Node's NamespaceURI property is the same as the NamespaceURI parameter.

The local name or tag name of Node is the same as the TagName parameter.

Tip: Given an IXMLNode interface, you can obtain the IDOMNode (see page 26306) interface required for the Node parameter by reading its DOMNode property.

See Also

ExtractLocalName (see page 26354)

ExtractPrefix (see page 26355)

IXMLNode

SameNamespace (see page 26359)

IXMLNode

196.3.10 xmldom.RegisterDOMVendor Function

Registers an implementation of the DOM interfaces so that it is available to the TXMLDocument class.

Pascal

```
procedure RegisterDOMVendor(const Vendor: TDOMVendor);
```

C++

```
RegisterDOMVendor(const TDOMVendor Vendor);
```

File

xmldom

Description

Call RegisterDOMVendor to register a DOM implementation that you want to use for parsing XML documents. A DOM (Document Object Model) implementation is a set of objects that implement the standard DOM interfaces.

If you are using another DOM implementation, you must create a TDOMVendor (see page 26263) descendant that can supply the IDOMImplementation (see page 26301) interface and description string for the DOM implementation. Pass an instance of this object as the Vendor parameter to RegisterDOMVendor.

See Also

UnRegisterDOMVendor (see page 26360)

TXMLDocument

TXMLDocument

DOMVendors (see page 26365)

Using the Document Object Model

196.3.11 xmldom.SameNamespace Function

Indicates whether a specified node is defined within a specified namespace.

Pascal

```
function SameNamespace(const Node: IDOMNode; const namespaceURI: WideString): Boolean;
overload;
```

```
function SameNamespace(const URI1: WideString; const URI2: WideString): Boolean; overload;
```

C++

```
Boolean SameNamespace(const IDOMNode Node, const BSTR namespaceURI);
Boolean SameNamespace(const BSTR URI1, const BSTR URI2);
```

File

xmldom

Description

Call SameNamespace to determine whether the two parameters identify the same namespace.

Using the first syntax, the Node parameter identifies a node and the NamespaceURI parameter a namespace. SameNamespace checks whether the node is defined in the specified namespace.

Using the second syntax, both parameters are namespace URI values. SameNamespace checks whether they both identify the same namespace.

Tip: Given an IXMLNode interface, you can obtain the IDOMNode (see page 26306) interface required for the Node parameter by reading its DOMNode property.

See Also

ExtractLocalName (see page 26354)

ExtractPrefix (see page 26355)

IXMLNode

SameNamespace

IXMLNode

IXMLNode

196.3.12 xmldom.UnRegisterDOMVendor Function

Unregisters a DOM implementation that was registered using RegisterDOMVendor (see page 26359) .

Pascal

```
procedure UnRegisterDOMVendor(const Vendor: TDOMVendor);
```

C++

```
UnRegisterDOMVendor(const TDOMVendor Vendor);
```

File

xmldom

Description

Call UnRegisterDOMVendor to unregister a DOM implementation when it is no longer available. After unregistering a DOM implementation, it can't be used by TXMLDocument for parsing XML documents.

Vendor is the TDOMVendor (see page 26263) descendant the represents the DOM implementation that is to be unregistered. It must be the same object used when registering the DOM implementation when you called RegisterDOMVendor (see page 26359).

See Also

RegisterDOMVendor (see page 26359)

TXMLDocument

TXMLDocument

DOMVendors (see page 26365)

Using the Document Object Model

196.4 Types

The following table lists types in this documentation.

Types

Name	Description
DOMNodeType (see page 26362)	DOMNodeType indicates the type of a node in an XML document, as parsed by a DOM parser.
DOMString (see page 26362)	DOMString is the type for strings that are used by a DOM parser.
DOMTimeStamp (see page 26363)	This is type xmldom.DOMTimeStamp.
TAsyncEventHandler (see page 26363)	TAsyncEventHandler is the type of event handlers that respond when the AsyncLoadState of an XML document changes.
TDOMVendorArray (see page 26364)	This is type xmldom.TDOMVendorArray.

196.4.1 xmldom.DOMNodeType Type

DOMNodeType indicates the type of a node in an XML document, as parsed by a DOM parser.

Pascal

```
DOMNodeType = Word;
```

C++

```
Word DOMNodeType;
```

File

xmldom

Description

DOMNodeType is the type of the nodeType property on the IDOMNode (see page 26306) interface. It is a flag, defined by W3C, that indicates the role played by a node in the parsed version of an XML document.

See Also

IDOMNode

196.4.2 xmldom.DOMString Type

DOMString is the type for strings that are used by a DOM parser.

Pascal

```
DOMString = WideString;
```

C++

```
BSTR DOMString;
```

File

xmldom

Description

DOM implementations work with wide characters. The strings that are used, therefore, when working with DOM interfaces, are of type DOMString, which simply maps to the WideString type.

196.4.3 xmldom.DOMTimeStamp Type

Pascal

```
DOMTimeStamp = Int64;
```

C++

```
Int64 DOMTimeStamp;
```

File

xmldom

Description

This is type xmldom.DOMTimeStamp.

196.4.4 xmldom.TAsyncEventHandler Type

TAsyncEventHandler is the type of event handlers that respond when the AsyncLoadState of an XML document changes.

Pascal

```
TAsyncEventHandler = procedure (Sender: TObject; AsyncLoadState: Integer) of object;
```

C++

```
(Sender: TObject; AsyncLoadState: Integer) ( TAsyncEventHandler)();
```

File

xmldom

Description

TAsyncEventHandler is the type for event handlers that respond at various stages during the asynchronous parsing of an XML document.

Sender is the TXMLDocument object that receives the event.

AsyncLoadState is an integer that indicates the current stage of the parsing process. Its value is vendor-specific, and corresponds to the document's AsyncLoadState property.

See Also

TXMLDocument

IXMLDocument

AsyncLoadState

196.4.5 xmldom.TDOMVendorArray Type

Pascal

```
TDOMVendorArray = array of TDOMVendor;
```

C++

```
array of TDOMVendor TDOMVendorArray;
```

File

xmldom

Description

This is type xmldom.TDOMVendorArray.

196.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
DefaultDOMVendor (see page 26365)	This is variable xmldom.DefaultDOMVendor.
DOMVendors (see page 26365)	This is variable xmldom.DOMVendors.

196.5.1 xmldom.DefaultDOMVendor Variable

Pascal

```
DefaultDOMVendor: string;
```

C++

```
AnsiString DefaultDOMVendor;
```

File

xmldom

Description

This is variable xmldom.DefaultDOMVendor.

196.5.2 xmldom.DOMVendors Variable

Pascal

```
DOMVendors: TDOMVendorList;
```

C++

```
TDOMVendorList DOMVendors;
```

File

xmldom

Description

This is variable xmldom.DOMVendors.

196.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
ATTRIBUTE_NODE (see page 26366)	This is constant xmldom.ATTRIBUTE_NODE.
CDATA_SECTION_NODE (see page 26367)	This is constant xmldom.CDATA_SECTION_NODE.
COMMENT_NODE (see page 26367)	This is constant xmldom.COMMENT_NODE.
DOCUMENT_FRAGMENT_NODE (see page 26367)	This is constant xmldom.DOCUMENT_FRAGMENT_NODE.
DOCUMENT_NODE (see page 26368)	This is constant xmldom.DOCUMENT_NODE.
DOCUMENT_TYPE_NODE (see page 26368)	This is constant xmldom.DOCUMENT_TYPE_NODE.
DOMSTRING_SIZE_ERR (see page 26368)	This is constant xmldom.DOMSTRING_SIZE_ERR.
DOMWrapperVersion (see page 26369)	This is constant xmldom.DOMWrapperVersion.
ELEMENT_NODE (see page 26369)	This is constant xmldom.ELEMENT_NODE.
ENTITY_NODE (see page 26369)	This is constant xmldom.ENTITY_NODE.
ENTITY_REFERENCE_NODE (see page 26370)	This is constant xmldom.ENTITY_REFERENCE_NODE.
HIERARCHY_REQUEST_ERR (see page 26370)	This is constant xmldom.HIERARCHY_REQUEST_ERR.
INDEX_SIZE_ERR (see page 26370)	This is constant xmldom.INDEX_SIZE_ERR.
INUSE_ATTRIBUTE_ERR (see page 26371)	This is constant xmldom.INUSE_ATTRIBUTE_ERR.
INVALID_ACCESS_ERR (see page 26371)	This is constant xmldom.INVALID_ACCESS_ERR.
INVALID_CHARACTER_ERR (see page 26371)	This is constant xmldom.INVALID_CHARACTER_ERR.
INVALID_MODIFICATION_ERR (see page 26372)	This is constant xmldom.INVALID_MODIFICATION_ERR.
INVALID_STATE_ERR (see page 26372)	This is constant xmldom.INVALID_STATE_ERR.
NAMESPACE_ERR (see page 26372)	This is constant xmldom.NAMESPACE_ERR.
NO_DATA_ALLOWED_ERR (see page 26373)	This is constant xmldom.NO_DATA_ALLOWED_ERR.
NO_MODIFICATION_ALLOWED_ERR (see page 26373)	This is constant xmldom.NO_MODIFICATION_ALLOWED_ERR.
NOT_FOUND_ERR (see page 26373)	This is constant xmldom.NOT_FOUND_ERR.
NOT_SUPPORTED_ERR (see page 26374)	This is constant xmldom.NOT_SUPPORTED_ERR.
NOTATION_NODE (see page 26374)	This is constant xmldom.NOTATION_NODE.
NSDelim (see page 26374)	This is constant xmldom.NSDelim.
PROCESSING_INSTRUCTION_NODE (see page 26375)	This is constant xmldom.PROCESSING_INSTRUCTION_NODE.
SEncoding (see page 26375)	This is constant xmldom.SEncoding.
SHttp (see page 26375)	This is constant xmldom.SHttp.
SStandalone (see page 26376)	This is constant xmldom.SStandalone.
SVersion (see page 26376)	This is constant xmldom.SVersion.
SXML (see page 26376)	This is constant xmldom.SXML.
SXMLNamespaceURI (see page 26377)	This is constant xmldom.SXMLNamespaceURI.
SXMLNS (see page 26377)	This is constant xmldom.SXMLNS.
SXMLPrefixNamespaceURI (see page 26377)	This is constant xmldom.SXMLPrefixNamespaceURI.
SYNTAX_ERR (see page 26378)	This is constant xmldom.SYNTAX_ERR.
TEXT_NODE (see page 26378)	This is constant xmldom.TEXT_NODE.
WRONG_DOCUMENT_ERR (see page 26378)	This is constant xmldom.WRONG_DOCUMENT_ERR.

196.6.1 xmldom.ATTRIBUTE_NODE Constant

Pascal

```
ATTRIBUTE_NODE = 2;
```

C++

```
ATTRIBUTE_NODE = 2;
```

File

xmldom

Description

This is constant xmldom.ATTRIBUTE_NODE.

196.6.2 xmldom.CDATA_SECTION_NODE Constant

Pascal

```
CDATA_SECTION_NODE = 4;
```

C++

```
CDATA_SECTION_NODE = 4;
```

File

xmldom

Description

This is constant xmldom.CDATA_SECTION_NODE.

196.6.3 xmldom.COMMENT_NODE Constant

Pascal

```
COMMENT_NODE = 8;
```

C++

```
COMMENT_NODE = 8;
```

File

xmldom

Description

This is constant xmldom.COMMENT_NODE.

196.6.4 xmldom.DOCUMENT_FRAGMENT_NODE Constant

Pascal

```
DOCUMENT_FRAGMENT_NODE = 11;
```

C++

```
DOCUMENT_FRAGMENT_NODE = 11;
```

File

xmldom

Description

This is constant xmldom.DOCUMENT_FRAGMENT_NODE.

196.6.5 xmldom.DOCUMENT_NODE Constant

Pascal

```
DOCUMENT_NODE = 9;
```

C++

```
DOCUMENT_NODE = 9;
```

File

xmldom

Description

This is constant xmldom.DOCUMENT_NODE.

196.6.6 xmldom.DOCUMENT_TYPE_NODE Constant

Pascal

```
DOCUMENT_TYPE_NODE = 10;
```

C++

```
DOCUMENT_TYPE_NODE = 10;
```

File

xmldom

Description

This is constant xmldom.DOCUMENT_TYPE_NODE.

196.6.7 xmldom.DOMSTRING_SIZE_ERR Constant

Pascal

```
DOMSTRING_SIZE_ERR = 2;
```

C++

```
DOMSTRING_SIZE_ERR = 2;
```

File

xmldom

Description

This is constant xmldom.DOMSTRING_SIZE_ERR.

196.6.8 xmldom.DOMWrapperVersion Constant

Pascal

```
DOMWrapperVersion = 1.2;
```

C++

```
DOMWrapperVersion = 1.2;
```

File

xmldom

Description

This is constant xmldom.DOMWrapperVersion.

196.6.9 xmldom.ELEMENT_NODE Constant

Pascal

```
ELEMENT_NODE = 1;
```

C++

```
ELEMENT_NODE = 1;
```

File

xmldom

Description

This is constant xmldom.ELEMENT_NODE.

196.6.10 xmldom.ENTITY_NODE Constant

Pascal

```
ENTITY_NODE = 6;
```

C++

```
ENTITY_NODE = 6;
```

File

xmldom

Description

This is constant xmldom.ENTITY_NODE.

196.6.11 xmldom.ENTITY_REFERENCE_NODE Constant

Pascal

```
ENTITY_REFERENCE_NODE = 5;
```

C++

```
ENTITY_REFERENCE_NODE = 5;
```

File

xmldom

Description

This is constant xmldom.ENTITY_REFERENCE_NODE.

196.6.12 xmldom.HIERARCHY_REQUEST_ERR Constant

Pascal

```
HIERARCHY_REQUEST_ERR = 3;
```

C++

```
HIERARCHY_REQUEST_ERR = 3;
```

File

xmldom

Description

This is constant xmldom.HIERARCHY_REQUEST_ERR.

196.6.13 xmldom.INDEX_SIZE_ERR Constant

Pascal

```
INDEX_SIZE_ERR = 1;
```

C++

```
INDEX_SIZE_ERR = 1;
```

File

xmldom

Description

This is constant xmldom.INDEX_SIZE_ERR.

196.6.14 xmldom.INUSE_ATTRIBUTE_ERR Constant

Pascal

```
INUSE_ATTRIBUTE_ERR = 10;
```

C++

```
INUSE_ATTRIBUTE_ERR = 10;
```

File

xmldom

Description

This is constant xmldom.INUSE_ATTRIBUTE_ERR.

196.6.15 xmldom.INVALID_ACCESS_ERR Constant

Pascal

```
INVALID_ACCESS_ERR = 15;
```

C++

```
INVALID_ACCESS_ERR = 15;
```

File

xmldom

Description

This is constant xmldom.INVALID_ACCESS_ERR.

196.6.16 xmldom.INVALID_CHARACTER_ERR Constant

Pascal

```
INVALID_CHARACTER_ERR = 5;
```

C++

```
INVALID_CHARACTER_ERR = 5;
```

File

xmldom

Description

This is constant xmldom.INVALID_CHARACTER_ERR.

196.6.17 xmldom.INVALID_MODIFICATION_ERR Constant

Pascal

```
INVALID_MODIFICATION_ERR = 13;
```

C++

```
INVALID_MODIFICATION_ERR = 13;
```

File

xmldom

Description

This is constant xmldom.INVALID_MODIFICATION_ERR.

196.6.18 xmldom.INVALID_STATE_ERR Constant

Pascal

```
INVALID_STATE_ERR = 11;
```

C++

```
INVALID_STATE_ERR = 11;
```

File

xmldom

Description

This is constant xmldom.INVALID_STATE_ERR.

196.6.19 xmldom.NAMESPACE_ERR Constant

Pascal

```
NAMESPACE_ERR = 14;
```

C++

```
NAMESPACE_ERR = 14;
```

File

xmldom

Description

This is constant xmldom.NAMESPACE_ERR.

196.6.20 xmldom.NO_DATA_ALLOWED_ERR Constant

Pascal

```
NO_DATA_ALLOWED_ERR = 6;
```

C++

```
NO_DATA_ALLOWED_ERR = 6;
```

File

xmldom

Description

This is constant xmldom.NO_DATA_ALLOWED_ERR.

196.6.21 xmldom.NO_MODIFICATION_ALLOWED_ERR Constant

Pascal

```
NO_MODIFICATION_ALLOWED_ERR = 7;
```

C++

```
NO_MODIFICATION_ALLOWED_ERR = 7;
```

File

xmldom

Description

This is constant xmldom.NO_MODIFICATION_ALLOWED_ERR.

196.6.22 xmldom.NOT_FOUND_ERR Constant

Pascal

```
NOT_FOUND_ERR = 8;
```

C++

```
NOT_FOUND_ERR = 8;
```

File

xmldom

Description

This is constant xmldom.NOT_FOUND_ERR.

196.6.23 xmldom.NOT_SUPPORTED_ERR Constant

Pascal

```
NOT_SUPPORTED_ERR = 9;
```

C++

```
NOT_SUPPORTED_ERR = 9;
```

File

xmldom

Description

This is constant xmldom.NOT_SUPPORTED_ERR.

196.6.24 xmldom.NOTATION_NODE Constant

Pascal

```
NOTATION_NODE = 12;
```

C++

```
NOTATION_NODE = 12;
```

File

xmldom

Description

This is constant xmldom.NOTATION_NODE.

196.6.25 xmldom.NSDelim Constant

Pascal

```
NSDelim = ':';
```

C++

```
NSDelim = ':';
```

File

xmldom

Description

This is constant xmldom.NSDelim.

196.6.26 xmldom.PROCESSING_INSTRUCTION_NODE Constant

Pascal

```
PROCESSING_INSTRUCTION_NODE = 7;
```

C++

```
PROCESSING_INSTRUCTION_NODE = 7;
```

File

xmldom

Description

This is constant xmldom.PROCESSING_INSTRUCTION_NODE.

196.6.27 xmldom.SEncoding Constant

Pascal

```
SEncoding = 'encoding';
```

C++

```
SEncoding = 'encoding';
```

File

xmldom

Description

This is constant xmldom.SEncoding.

196.6.28 xmldom.SHttp Constant

Pascal

```
SHttp = 'http:/';
```

C++

```
SHttp = 'http:/';
```

File

xmldom

Description

This is constant xmldom.SHttp.

196.6.29 xmldom.SStandalone Constant

Pascal

```
SStandalone = 'standalone';
```

C++

```
SStandalone = 'standalone';
```

File

xmldom

Description

This is constant xmldom.SStandalone.

196.6.30 xmldom.SVersion Constant

Pascal

```
SVersion = 'version';
```

C++

```
SVersion = 'version';
```

File

xmldom

Description

This is constant xmldom.SVersion.

196.6.31 xmldom.SXML Constant

Pascal

```
SXML = 'xml';
```

C++

```
SXML = 'xml';
```

File

xmldom

Description

This is constant xmldom.SXML.

196.6.32 xmldom.SXMLNamespaceURI Constant

Pascal

```
SXMLNamespaceURI = SHttp+'/www.w3.org/2000/xmlns/';
```

C++

```
SXMLNamespaceURI = SHttp+'/www.w3.org/2000/xmlns/';
```

File

xmldom

Description

This is constant xmldom.SXMLNamespaceURI.

196.6.33 xmldom.SXMLNS Constant

Pascal

```
SXMLNS = 'xmlns';
```

C++

```
SXMLNS = 'xmlns';
```

File

xmldom

Description

This is constant xmldom.SXMLNS.

196.6.34 xmldom.SXMLPrefixNamespaceURI Constant

Pascal

```
SXMLPrefixNamespaceURI = SHttp+'/www.w3.org/XML/1998/namespace';
```

C++

```
SXMLPrefixNamespaceURI = SHttp+'/www.w3.org/XML/1998/namespace';
```

File

xmldom

Description

This is constant xmldom.SXMLPrefixNamespaceURI.

196.6.35 xmldom.SYNTAX_ERR Constant

Pascal

```
SYNTAX_ERR = 12;
```

C++

```
SYNTAX_ERR = 12;
```

File

xmldom

Description

This is constant xmldom.SYNTAX_ERR.

196.6.36 xmldom.TEXT_NODE Constant

Pascal

```
TEXT_NODE = 3;
```

C++

```
TEXT_NODE = 3;
```

File

xmldom

Description

This is constant xmldom.TEXT_NODE.

196.6.37 xmldom.WRONG_DOCUMENT_ERR Constant

Pascal

```
WRONG_DOCUMENT_ERR = 4;
```

C++

```
WRONG_DOCUMENT_ERR = 4;
```

File

xmldom

Description

This is constant xmldom.WRONG_DOCUMENT_ERR.

197 XMLIntf Namespace

This is namespace XMLIntf.

Topics

Name	Description
Classes (see page 26381)	The following table lists classes in this documentation.
Interfaces (see page 26382)	The following table lists interfaces in this documentation.
Structs, Records, Enums (see page 26458)	The following table lists structs, records, enums in this documentation.
Types (see page 26463)	The following table lists types in this documentation.
Constants (see page 26464)	The following table lists constants in this documentation.

Classes

	Name	Description
	EXMLDocError (see page 26381)	EXMLDocError is the exception class for errors editing or parsing an XML document.

Constants

Name	Description
NodeTypeNames (see page 26464)	This is constant XMLIntf.NodeTypeNames.

Interfaces

	Name	Description
	IXMLDocument (see page 26382)	IXMLDocument represents an XML (see page 26410) document.
	IXMLNode (see page 26411)	IXMLNode is the interface for a node in an XML (see page 26439) document.
	IXMLNodeCollection (see page 26440)	IXMLNodeCollection is the interface for a set of child nodes that are all the same type.
	IXMLNodeList (see page 26445)	IXMLNodeList represents a set of nodes in an XML document.

Structs, Records, Enums

	Name	Description
	TNodeType (see page 26458)	TNodeType identifies the type of a node in an XML document.
	TParseOption (see page 26460)	TParseOption and TParseOptions (see page 26463) represent implementation-specific options that are followed when parsing the XML document.
	TXMLDocOption (see page 26460)	TXMLDocOption and TXMLDocOptions (see page 26463) indicate types of information that an XML document automatically supplies.
	TXMLEncodingType (see page 26461)	TXMLEncodingType indicates the character set used to encode characters in an XML document.

Types

Name	Description
TParseOptions (see page 26463)	This is type XMLIntf.TParseOptions.
TXMLDocOptions (see page 26463)	This is type XMLIntf.TXMLDocOptions.

197.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EXMLDocError (see page 26381)	EXMLDocError is the exception class for errors editing or parsing an XML document.

197.1.1 EXMLDocError Class

EXMLDocError is the exception class for errors editing or parsing an XML document.

Class Hierarchy

Pascal

```
EXMLDocError = class(Exception);
```

C++

```
class EXMLDocError : public Exception;
```

File

XMLIntf

Description

EXMLDocError is raised by TXMLDocument or TXMLNode when

An error occurs when loading an XML document.

An error occurs when creating a node in an XML document.

A required DOM interface is not found.

An option is requested that is not supported by the DOM implementation.

An attempt is made to perform an action that is not consistent with the specified node type.

See Also

XMLDocError

TXMLDocument (see page 26097)

TXMLNode (see page 26147)

197.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IXMLDocument (see page 26382)	IXMLDocument represents an XML (see page 26410) document.
	IXMLNode (see page 26411)	IXMLNode is the interface for a node in an XML (see page 26439) document.
	IXMLNodeCollection (see page 26440)	IXMLNodeCollection is the interface for a set of child nodes that are all the same type.
	IXMLNodeList (see page 26445)	IXMLNodeList represents a set of nodes in an XML document.

197.2.1 IXMLDocument Interface

IXMLDocument represents an XML (see page 26410) document.

Class Hierarchy

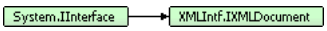

Pascal

```
IXMLDocument = interface(IInterface);
```

C++

```
__interface IXMLDocument : public IInterface;
```

File

XMLIntf

Description

Use IXMLDocument to work with an XML (see page 26410) document. Applications can obtain an IXMLDocument interface through a TXMLDocument component, using the global LoadXMLDocument function, or through the OwnerDocument property of any node in the document.

Like the related DOM interface IDOMDocument, you can use IXMLDocument to access the contents of an XML (see page 26410) document and to create new nodes. In addition, IXMLDocument lets you load an XML (see page 26410) document that is stored on disk or assign a string of raw (unparsed) XML (see page 26410) as the contents of the document.

TXMLDocument implements the IXMLDocument interface. Typically, the TXMLDocument component is the starting point for working with an XML (see page 26410) document.

Note: C++ method declarations that use IXMLDocument use the _di_IXMLDocument type instead. This type is a DelphiInterface wrapper around the IXMLDocument interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IXMLDocument >  _di_IXMLDocument;
```

See Also

IXMLNode (see page 26411)

EXMLDocError (see page 26381)

TXMLDocument (see page 26097)

IXMLNode (see page 26411)

LoadXMLDocument

197.2.1.1 IXMLDocument Members

The following tables list the members exposed by IXMLDocument.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26385)	This is the overview for the AddChild method overload.
	CreateElement (see page 26386)	Creates a new element node that is associated with this document.
	CreateNode (see page 26387)	Creates a new node that is associated with this document.
	GeneratePrefix (see page 26388)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 26388)	Returns the value of the Active (see page 26403) property.
	GetAsyncLoadState (see page 26389)	Returns the value of the AsyncLoadState (see page 26404) property.
	GetChildNodes (see page 26389)	Returns the value of the ChildNodes (see page 26405) property.
	GetDocBinding (see page 26389)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 26390)	Returns the value of the DocumentElement (see page 26405) property.
	GetDocumentNode (see page 26390)	Returns the value of the Node (see page 26407) property.
	GetDOMDocument (see page 26390)	Returns the value of the DOMDocument (see page 26405) property.
	GetEncoding (see page 26391)	Returns the value of the Encoding (see page 26406) property.
	GetFileName (see page 26391)	Returns the value of the FileName (see page 26406) property.
	GetModified (see page 26391)	Returns the value of the Modified (see page 26407) property.
	GetNodeIndentStr (see page 26392)	Returns the value of the NodeIndentStr (see page 26408) property.
	GetOptions (see page 26392)	Returns the value of the Options (see page 26408) property.
	GetParseOptions (see page 26392)	Returns the value of the ParseOptions (see page 26409) property.
	GetSchemaRef (see page 26392)	Returns the value of the SchemaRef (see page 26409) property.
	GetStandAlone (see page 26393)	Returns the value of the StandAlone (see page 26410) property.
	GetVersion (see page 26393)	Returns the value of the Version (see page 26410) property.
	GetXML (see page 26393)	Returns the value of the XML (see page 26410) property.
	IsEmptyDoc (see page 26394)	Indicates whether the IXMLDocument (see page 26382) instance represents an empty document.
	LoadFromFile (see page 26394)	Loads an XML (see page 26410) document from disk.
	LoadFromStream (see page 26395)	Loads an XML (see page 26410) document from a stream and activates it.
	LoadFromXML (see page 26395)	This is the overview for the LoadFromXML method overload.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	Refresh (see page 26396)	Updates the parsed XML (see page 26410) document to reflect external changes.
	RegisterDocBinding (see page 26396)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 26397)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 26397)	Saves the XML (see page 26410) document to disk.
	SaveToStream (see page 26398)	Saves the XML (see page 26410) document to a stream.
	SaveToXML (see page 26398)	This is the overview for the SaveToXML method overload.
	SetActive (see page 26399)	Sets the value of the Active (see page 26403) property.
	SetDocumentElement (see page 26399)	Sets the value of the DocumentElement (see page 26405) property.
	SetDOMDocument (see page 26399)	Sets the value of the DOMDocument (see page 26405) property.
	SetEncoding (see page 26400)	Sets the value of the Encoding (see page 26406) property.

	SetFileName (see page 26400)	Sets the value of the FileName (see page 26406) property.
	SetNodeIndentStr (see page 26400)	Sets the value of the NodeIndentStr (see page 26408) property.
	SetOnAsyncLoad (see page 26401)	Supplies an event handler that responds when the AsyncLoadState (see page 26404) property changes.
	SetOptions (see page 26401)	Sets the value of the Options (see page 26408) property.
	SetParseOptions (see page 26402)	Sets the value of the ParseOptions (see page 26409) property.
	SetStandAlone (see page 26402)	Sets the value of the StandAlone (see page 26410) property.
	SetVersion (see page 26402)	Sets the value of the Version (see page 26410) property.
	SetXML (see page 26403)	Sets the value of the XML (see page 26410) property.

Public Properties

	Name	Description
	Active (see page 26403)	Specifies whether the XML (see page 26410) document has been parsed so that the DOM interfaces are available to examine or change its contents.
R	AsyncLoadState (see page 26404)	Indicates the state of the DOM parser when it parses the XML (see page 26410) document asynchronously.
R	ChildNodes (see page 26405)	Lists all of the child nodes of the document.
	DocumentElement (see page 26405)	Provides access to the root node of the XML (see page 26410) document.
	DOMDocument (see page 26405)	Provides access to the DOM interface for the parsed XML (see page 26410) document.
	Encoding (see page 26406)	Indicates the character set that the XML (see page 26410) document uses to encode characters.
	FileName (see page 26406)	Indicates the XML (see page 26410) document file, if any, that this interface represents.
R	Modified (see page 26407)	Indicates whether the XML (see page 26410) document has been modified.
R	Node (see page 26407)	Provides access to the document node for the XML (see page 26410) document.
	NodeIndentStr (see page 26408)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 26410) text.
	Options (see page 26408)	Influences various aspects of the document's behavior.
	ParseOptions (see page 26409)	Specifies implementation-specific options that are followed when parsing the XML (see page 26410) document.
R	SchemaRef (see page 26409)	Returns the name of the schema associated with this XML (see page 26410) document.
	StandAlone (see page 26410)	Indicates whether the XML (see page 26410) document includes any external markup declarations.
	Version (see page 26410)	Indicates the XML (see page 26410) version that the document uses.
	XML (see page 26410)	Specifies the content of the XML document associated with this IXMLDocument (see page 26382) interface.

197.2.1.2 IXMLDocument Methods

The methods of the IXMLDocument class are listed here.

Public Methods

	Name	Description
	AddChild (see page 26385)	This is the overview for the AddChild method overload.
	CreateElement (see page 26386)	Creates a new element node that is associated with this document.
	CreateNode (see page 26387)	Creates a new node that is associated with this document.
	GeneratePrefix (see page 26388)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 26388)	Returns the value of the Active (see page 26403) property.
	GetAsyncLoadState (see page 26389)	Returns the value of the AsyncLoadState (see page 26404) property.
	GetChildNodes (see page 26389)	Returns the value of the ChildNodes (see page 26405) property.
	GetDocBinding (see page 26389)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 26390)	Returns the value of the DocumentElement (see page 26405) property.

	GetDocumentNode (see page 26390)	Returns the value of the Node (see page 26407) property.
	GetDOMDocument (see page 26390)	Returns the value of the DOMDocument (see page 26405) property.
	GetEncoding (see page 26391)	Returns the value of the Encoding (see page 26406) property.
	GetFileName (see page 26391)	Returns the value of the FileName (see page 26406) property.
	GetModified (see page 26391)	Returns the value of the Modified (see page 26407) property.
	GetNodeIndentStr (see page 26392)	Returns the value of the NodeIndentStr (see page 26408) property.
	GetOptions (see page 26392)	Returns the value of the Options (see page 26408) property.
	GetParseOptions (see page 26392)	Returns the value of the ParseOptions (see page 26409) property.
	GetSchemaRef (see page 26392)	Returns the value of the SchemaRef (see page 26409) property.
	GetStandAlone (see page 26393)	Returns the value of the StandAlone (see page 26410) property.
	GetVersion (see page 26393)	Returns the value of the Version (see page 26410) property.
	GetXML (see page 26393)	Returns the value of the XML (see page 26410) property.
	IsEmptyDoc (see page 26394)	Indicates whether the IXMLDocument (see page 26382) instance represents an empty document.
	LoadFromFile (see page 26394)	Loads an XML (see page 26410) document from disk.
	LoadFromStream (see page 26395)	Loads an XML (see page 26410) document from a stream and activates it.
	LoadFromXML (see page 26395)	This is the overview for the LoadFromXML method overload.
	Refresh (see page 26396)	Updates the parsed XML (see page 26410) document to reflect external changes.
	RegisterDocBinding (see page 26396)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 26397)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 26397)	Saves the XML (see page 26410) document to disk.
	SaveToStream (see page 26398)	Saves the XML (see page 26410) document to a stream.
	SaveToXML (see page 26398)	This is the overview for the SaveToXML method overload.
	SetActive (see page 26399)	Sets the value of the Active (see page 26403) property.
	SetDocumentElement (see page 26399)	Sets the value of the DocumentElement (see page 26405) property.
	SetDOMDocument (see page 26399)	Sets the value of the DOMDocument (see page 26405) property.
	SetEncoding (see page 26400)	Sets the value of the Encoding (see page 26406) property.
	SetFileName (see page 26400)	Sets the value of the FileName (see page 26406) property.
	SetNodeIndentStr (see page 26400)	Sets the value of the NodeIndentStr (see page 26408) property.
	SetOnAsyncLoad (see page 26401)	Supplies an event handler that responds when the AsyncLoadState (see page 26404) property changes.
	SetOptions (see page 26401)	Sets the value of the Options (see page 26408) property.
	SetParseOptions (see page 26402)	Sets the value of the ParseOptions (see page 26409) property.
	SetStandAlone (see page 26402)	Sets the value of the StandAlone (see page 26410) property.
	SetVersion (see page 26402)	Sets the value of the Version (see page 26410) property.
	SetXML (see page 26403)	Sets the value of the XML (see page 26410) property.

197.2.1.2.1 AddChild Method

This is the overview for the AddChild method overload.

Overload List

	Name	Description
	IXMLDocument.AddChild (DOMString) (see page 26385)	Creates and adds a node to the child nodes of this document.
	IXMLDocument.AddChild (DOMString, DOMString) (see page 26385)	Creates and adds a node to the child nodes of this document.

197.2.1.2.1.1 IXMLDocument.AddChild Method (DOMString)

Creates and adds a node to the child nodes of this document.

Pascal

```
function AddChild(const TagName: DOMString): IXMLNode; overload;
function AddChild(const TagName: DOMString; const NamespaceURI: DOMString): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode AddChild(const DOMString TagName);
__fastcall IXMLNode AddChild(const DOMString TagName, const DOMString NamespaceURI);
```

Description

Call AddChild to add a new child element node to the document itself. The new node is added to the end of the list maintained by the ChildNodes (see page 26405) property.

TagName is the tag name for the new element node.

NamespaceURI identifies the namespace, if any, that includes the new node's definition.

AddChild returns the interface for the newly created node.

Note: Do not call AddChild to add a child to the document element of this document. When adding data to the XML (see page 26410) document, use the AddChild method of the document element or of the node in the hierarchy that should be the parent of the new node.

See Also

AddChild (see page 26414)

CreateNode (see page 26387)

CreateElement (see page 26386)

ChildNodes (see page 26405)

DocumentElement (see page 26405)

197.2.1.2.2 IXMLDocument.CreateElement Method

Creates a new element node that is associated with this document.

Pascal

```
function CreateElement(const TagOrData: DOMString; const NamespaceURI: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateElement(const DOMString TagOrData, const DOMString NamespaceURI);
```

Description

Call CreateElement to create a new generic element node. The resulting node does not have a parent, but can be added to the ChildNodes (see page 26405) list of any node in the document (including this document's ChildNodes (see page 26405) property). It can also be assigned to the DocumentElement (see page 26405) property to replace the entire data hierarchy of this document.

TagOrData provides the tag name of the newly created node.

NamespaceURI identifies the namespace that includes the new node's definition.

CreateElement returns the interface for the new node.

Note: To add a node as the child of another node in this document, it is simpler to call that node's AddChild (see page 26385) method.

See Also

AddChild (see page 26414)

ChildNodes (see page 26405)

DocumentElement (see page 26405)

AddChild (see page 26385)

CreateNode (see page 26387)

197.2.1.2.3 IXMLDocument.CreateNode Method

Creates a new node that is associated with this document.

Pascal

```
function CreateNode(const NameOrData: DOMString; NodeType: TNodeType = ntElement; const
AddlData: DOMString = ''): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateNode(const DOMString NameOrData, TNodeType NodeType = ntElement,
const DOMString AddlData = '');
```

Description

Call CreateNode to create a new generic XML (see page 26410) node. The resulting node does not have a parent, but can be added to the ChildNodes (see page 26405) or AttributeNodes list of any node in the document (including this document's ChildNodes (see page 26405) property).

NameOrData provides the tag name or value of the newly created node. Its interpretation depends on the type of node created, as indicated in the following table:

NodeType	NameOrData
ntElement	The tag name.
ntAttribute	The attribute name.
ntText	The value of the node.
ntCData	The value of the CDATA section.
ntEntityRef	The name of the referenced entity.
ntProcessingInstr	The target of the processing instruction.
ntComment	The value (text) of the comment.
ntDocFragment	Not used.
ntElement	The namespace URI that qualifies the tag name.
ntAttribute	The namespace URI that qualifies the attribute name.
ntProcessingInstr	The content of the processing instruction, except for the target.

NodeType indicates the type of node to create. It can only be one of the types listed in the previous table.

The meaning of AddlData depends on the node type, as indicated in the following table:

NodeType	AddlData
ntElement	The namespace URI that qualifies the tag name.
ntAttribute	The namespace URI that qualifies the attribute name.

ntProcessingInstr	The content of the processing instruction, except for the target.

CreateNode returns the interface for the new node.

Note: To add a node as the child of another node in this document, it is simpler to call that node's AddChild (see page 26385) method.

See Also

AddChild (see page 26414)

ChildNodes (see page 26405)

AttributeNodes (see page 26430)

DocumentElement (see page 26405)

CreateElement (see page 26386)

AddChild (see page 26385)

197.2.1.2.4 IXMLDocument.GeneratePrefix Method

Generates a unique namespace prefix that can be used for a node in the document.

Pascal

```
function GeneratePrefix(const Node: IXMLNode): DOMString;
```

C++

```
__fastcall DOMString GeneratePrefix(const IXMLNode Node);
```

Description

The AddChild (see page 26385) method for each node allows the caller to specify that the new node should use an automatically-generated namespace prefix. When this option is specified, the node calls GeneratePrefix to generate a unique string to act as a namespace prefix.

Node (see page 26407) is the interface for the node which will use the newly-generated namespace prefix.

GeneratePrefix returns a unique string that is not already used in the document to represent a namespace URI. The generated namespace prefixes are only unique within a single document.

See Also

AddChild (see page 26414)

197.2.1.2.5 IXMLDocument.GetActive Method

Returns the value of the Active (see page 26403) property.

Pascal

```
function GetActive: Boolean;
```

C++

```
__fastcall Boolean GetActive();
```

Description

GetActive is the read implementation of the Active (see page 26403) property.

See Also

Active (see page 26403)

197.2.1.2.6 IXMLDocument.GetAsyncLoadState Method

Returns the value of the AsyncLoadState (see page 26404) property.

Pascal

```
function GetAsyncLoadState: Integer;
```

C++

```
__fastcall int GetAsyncLoadState();
```

Description

GetAsyncLoadState is the read implementation of the AsyncLoadState (see page 26404) property.

See Also

AsyncLoadState (see page 26404)

197.2.1.2.7 IXMLDocument.GetChildNodes Method

Returns the value of the ChildNodes (see page 26405) property.

Pascal

```
function GetChildNodes: IXMLNodeList;
```

C++

```
__fastcall IXMLNodeList GetChildNodes();
```

Description

GetChildNodes is the read implementation of the ChildNodes (see page 26405) property.

See Also

ChildNodes (see page 26405)

197.2.1.2.8 IXMLDocument.GetDocBinding Method

Replaces a specified child node with an instance of a specified implementation class.

Pascal

```
function GetDocBinding(const TagName: DOMString; DocNodeClass: TClass; NamespaceURI: DOMString
= ''): IXMLNode;
```

C++

```
__fastcall IXMLNode GetDocBinding(const DOMString TagName, TClass DocNodeClass, DOMString
NamespaceURI = '');
```

Description

Call GetDocBinding to specify the implementation class for one of the child nodes of the document node. After replacing the specified node with an instance of the specified class, GetDocBinding returns the interface of the document element.

TagName is the tag name of the child node to be replaced. Typically, this identifies the document element.

DocNodeClass is the implementation class to use for all child nodes with the specified tag name.

NamespaceURI is the URI of the namespace in which TagName is defined.

See Also

DocumentElement (see page 26405)

RegisterDocBinding (see page 26396)

ChildNodes (see page 26405)

197.2.1.2.9 IXMLDocument.GetDocumentElement Method

Returns the value of the DocumentElement (see page 26405) property.

Pascal

```
function GetDocumentElement: IXMLNode;
```

C++

```
__fastcall IXMLNode GetDocumentElement();
```

Description

GetDocumentElement is the read implementation of the DocumentElement (see page 26405) property.

See Also

DocumentElement (see page 26405)

197.2.1.2.10 IXMLDocument.GetDocumentNode Method

Returns the value of the Node (see page 26407) property.

Pascal

```
function GetDocumentNode: IXMLNode;
```

C++

```
__fastcall IXMLNode GetDocumentNode();
```

Description

GetDocumentNode is the read implementation of the Node (see page 26407) property.

See Also

Node (see page 26407)

197.2.1.2.11 IXMLDocument.GetDOMDocument Method

Returns the value of the DOMDocument (see page 26405) property.

Pascal

```
function GetDOMDocument: IDOMDocument;
```

C++

```
__fastcall IDOMDocument GetDOMDocument();
```

Description

GetDOMDocument is the read implementation of the DOMDocument (see page 26405) property.

See Also

DOMDocument (see page 26405)

197.2.1.2.12 IXMLDocument.GetEncoding Method

Returns the value of the Encoding (see page 26406) property.

Pascal

```
function GetEncoding: DOMString;
```

C++

```
__fastcall DOMString GetEncoding();
```

Description

GetEncoding is the read implementation of the Encoding (see page 26406) property.

See Also

Encoding (see page 26406)

197.2.1.2.13 IXMLDocument.GetFileName Method

Returns the value of the FileName (see page 26406) property.

Pascal

```
function GetFileName: DOMString;
```

C++

```
__fastcall DOMString GetFileName();
```

Description

GetFileName is the read implementation of the FileName (see page 26406) property.

See Also

FileName (see page 26406)

197.2.1.2.14 IXMLDocument.GetModified Method

Returns the value of the Modified (see page 26407) property.

Pascal

```
function GetModified: Boolean;
```

C++

```
__fastcall Boolean GetModified();
```

Description

GetModified is the read implementation of the Modified (see page 26407) property.

See Also

Modified (see page 26407)

197.2.1.2.15 IXMLDocument.GetNodeIndentStr Method

Returns the value of the NodeIndentStr (see page 26408) property.

Pascal

```
function GetNodeIndentStr: DOMString;
```

C++

```
__fastcall DOMString GetNodeIndentStr();
```

Description

GetNodeIndentStr is the read implementation of the NodeIndentStr (see page 26408) property.

See Also

NodeIndentStr (see page 26408)

197.2.1.2.16 IXMLDocument.GetOptions Method

Returns the value of the Options (see page 26408) property.

Pascal

```
function GetOptions: TXMLDocOptions;
```

C++

```
__fastcall TXMLDocOptions GetOptions();
```

Description

GetOptions is the read implementation of the Options (see page 26408) property.

See Also

Options (see page 26408)

197.2.1.2.17 IXMLDocument.GetParseOptions Method

Returns the value of the ParseOptions (see page 26409) property.

Pascal

```
function GetParseOptions: TParseOptions;
```

C++

```
__fastcall TParseOptions GetParseOptions();
```

Description

GetParseOptions is the read implementation of the ParseOptions (see page 26409) property.

See Also

ParseOptions (see page 26409)

197.2.1.2.18 IXMLDocument.GetSchemaRef Method

Returns the value of the SchemaRef (see page 26409) property.

Pascal

```
function GetSchemaRef: DOMString;
```

C++

```
__fastcall DOMString GetSchemaRef();
```

Description

GetSchemaRef is the read implementation of the SchemaRef (see page 26409) property.

See Also

SchemaRef (see page 26409)

197.2.1.2.19 IXMLDocument.GetStandAlone Method

Returns the value of the StandAlone (see page 26410) property.

Pascal

```
function GetStandAlone: DOMString;
```

C++

```
__fastcall DOMString GetStandAlone();
```

Description

GetStandAlone is the read implementation of the StandAlone (see page 26410) property.

See Also

StandAlone (see page 26410)

197.2.1.2.20 IXMLDocument.GetVersion Method

Returns the value of the Version (see page 26410) property.

Pascal

```
function GetVersion: DOMString;
```

C++

```
__fastcall DOMString GetVersion();
```

Description

GetVersion is the read implementation of the Version (see page 26410) property.

See Also

Version (see page 26410)

197.2.1.2.21 IXMLDocument.GetXML Method

Returns the value of the XML (see page 26410) property.

Pascal

```
function GetXML: TStrings;
```

C++

```
__fastcall TStrings GetXML();
```

Description

GetXML is the read implementation of the XML (see page 26410) property.

See Also

XML (see page 26410)

197.2.1.2.22 IXMLDocument.IsEmptyDoc Method

Indicates whether the IXMLDocument (see page 26382) instance represents an empty document.

Pascal

```
function IsEmptyDoc: Boolean;
```

C++

```
__fastcall Boolean IsEmptyDoc();
```

Description

Call IsEmptyDoc to determine whether the IXMLDocument (see page 26382) instance represents an empty document. IsEmptyDoc returns true if the DOMDocument (see page 26405) property is not set (for example, if there is no DOM implementation) or if this interface represents a document with no child nodes.

See Also

XML (see page 26410)

ChildNodes (see page 26405)

DOMDocument (see page 26405)

197.2.1.2.23 IXMLDocument.LoadFromFile Method

Loads an XML (see page 26410) document from disk.

Pascal

```
procedure LoadFromFile(const AFileName: DOMString);
```

C++

```
__fastcall LoadFromFile(const DOMString AFileName);
```

Description

Call LoadFromFile to load the XML (see page 26410) document specified by AFileName so that you can examine or modify it.

AFileName is the name of the file to load.

Once you have loaded an XML (see page 26410) document, any changes you make to the document are not saved back to disk until you call the SaveToFile (see page 26397) method.

See Also

SaveToFile (see page 26397)

XML (see page 26410)

FileName (see page 26406)

197.2.1.2.24 IXMLDocument.LoadFromStream Method

Loads an XML (see page 26410) document from a stream and activates it.

Pascal

```
procedure LoadFromStream(const Stream: TStream; EncodingType: TXMLEncodingType = xetUnknown);
```

C++

```
__fastcall LoadFromStream(const TStream Stream, TXMLEncodingType EncodingType = xetUnknown);
```

Description

Call LoadFromStream to load the XML (see page 26410) document from a stream.

Stream is a stream object that can be used to read the string of XML (see page 26410) that makes up the document.

EncodingType indicates the character set that is used for Stream. If EncodingType is xetUnknown (the default), LoadFromStream attempts to infer the encoding type from the XML (see page 26410) encoding declaration at the beginning of the document.

After loading the document from Stream, LoadFromStream sets the Active (see page 26403) property to true.

See Also

SaveToStream (see page 26398)

LoadFromFile (see page 26394)

LoadFromXML (see page 26395)

Active (see page 26403)

197.2.1.2.25 LoadFromXML Method

This is the overview for the LoadFromXML method overload.

Overload List

	Name	Description
	IXMLDocument.LoadFromXML (DOMString) (see page 26395)	Loads a string representation of an XML (see page 26410) document and activates it.
	IXMLDocument.LoadFromXML (string) (see page 26395)	Loads a string representation of an XML (see page 26410) document and activates it.

197.2.1.2.25.1 IXMLDocument.LoadFromXML Method (DOMString)

Loads a string representation of an XML (see page 26410) document and activates it.

Pascal

```
procedure LoadFromXML(const XML: string); overload;
procedure LoadFromXML(const XML: DOMString); overload;
```

C++

```
__fastcall LoadFromXML(const AnsiString XML);
__fastcall LoadFromXML(const DOMString XML);
```

Description

Call LoadFromXML to assign a string as the value of the XML (see page 26410) document. Unlike the XML (see page

26410) property, which lets you assign XML (see page 26410) on a line-by-line basis, LoadFromXML treats the text of the XML (see page 26410) document as a whole.

After assigning the XML (see page 26410) property as the contents of the document, LoadFromXML sets the Active (see page 26403) property to true.

See Also

SaveToXML (see page 26398)

LoadFromFile (see page 26394)

Active (see page 26403)

LoadFromStream (see page 26395)

XML (see page 26410)

197.2.1.2.26 IXMLDocument.Refresh Method

Updates the parsed XML (see page 26410) document to reflect external changes.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh when the XML (see page 26410) document specified by FileName (see page 26406) has changed due to an external cause. Refresh updates the parsed version of the XML (see page 26410) document (which can be traversed starting with DocumentElement (see page 26405)) to reflect the current contents of the file specified by FileName (see page 26406).

The Active (see page 26403) property must be true when you call Refresh.

Note: Do not call Refresh after making changes directly to the XML (see page 26410) property, because assigning a value to XML (see page 26410) automatically closes the XML (see page 26410) document. To refresh the XML (see page 26410) document after assigning a new value to XML (see page 26410), set the Active (see page 26403) property to true.

See Also

FileName (see page 26406)

Active (see page 26403)

DOMDocument (see page 26405)

DocumentElement (see page 26405)

Resync (see page 26397)

197.2.1.2.27 IXMLDocument.RegisterDocBinding Method

Registers an implementation class for child nodes with a specified tag name.

Pascal

```
procedure RegisterDocBinding(const TagName: DOMString; DocNodeClass: TClass; NamespaceURI:
DOMString = '');
```

C++

```
__fastcall RegisterDocBinding(const DOMString TagName, TClass DocNodeClass, DOMString
NamespaceURI = '');
```

Description

Call RegisterDocBinding to specify the implementation class for one of the child nodes of the document node. RegisterDocBinding replaces the specified node with an instance of the specified class.

TagName is the tag name of the child node to be replaced. Typically, this identifies the document element.

DocNodeClass is the implementation class to use for all child nodes with the specified tag name.

NamespaceURI is the URI of the namespace in which TagName is defined.

See Also

DocumentElement (see page 26405)

GetDocBinding (see page 26389)

ChildNodes (see page 26405)

197.2.1.2.28 IXMLDocument.Resync Method

Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

Pascal

```
procedure Resync;
```

C++

```
__fastcall Resync();
```

Description

Resync forces the document to reread the values of its child nodes and attribute nodes, discarding any changes that have not been propagated to the underlying DOM implementation.

See Also

ChildNodes (see page 26405)

197.2.1.2.29 IXMLDocument.SaveToFile Method

Saves the XML (see page 26410) document to disk.

Pascal

```
procedure SaveToFile(const AFileName: DOMString);
```

C++

```
__fastcall SaveToFile(const DOMString AFileName);
```

Description

Call SaveToFile to save any modifications you have made to the parsed XML (see page 26410) document.

AFileName is the name of the file to save.

Tip: To save the modifications back to the file from which the document was originally read, use the value of the FileName (see page 26406) property for the AFileName parameter.

See Also

LoadFromFile (see page 26394)

XML (see page 26410)

FileName (see page 26406)

197.2.1.2.30 IXMLDocument.SaveToStream Method

Saves the XML (see page 26410) document to a stream.

Pascal

```
procedure SaveToStream(const Stream: TStream);
```

C++

```
__fastcall SaveToStream(const TStream Stream);
```

Description

Call SaveToStream to save the contents of the XML (see page 26410) document to the stream specified by Stream. Regardless of the encoding system of the original XML (see page 26410) document, SaveToStream always saves the stream in UTF-16.

See Also

LoadFromStream (see page 26395)

SaveToXML (see page 26398)

SaveToFile (see page 26397)

197.2.1.2.31 SaveToXML Method

This is the overview for the SaveToXML method overload.

Overload List

	Name	Description
	IXMLDocument.SaveToXML (DOMString) (see page 26398)	Saves the XML (see page 26410) document to a string-type variable.
	IXMLDocument.SaveToXML (string) (see page 26398)	Saves the XML (see page 26410) document to a string-type variable.

197.2.1.2.31.1 IXMLDocument.SaveToXML Method (DOMString)

Saves the XML (see page 26410) document to a string-type variable.

Pascal

```
procedure SaveToXML(var XML: DOMString); overload;
procedure SaveToXML(var XML: string); overload;
```

C++

```
__fastcall SaveToXML(DOMString XML);
__fastcall SaveToXML(AnsiString XML);
```

Description

Call SaveToXML to save the contents of the XML (see page 26410) document to the string-type variable specified by XML (

see page 26410). SaveToXML writes the contents of XML (see page 26410) document using UTF-8 or UTF-16 as an encoding system, depending on the type of the XML (see page 26410) parameter.

Unlike the XML (see page 26410) property, which lets you write individual lines from the XML (see page 26410) document, SaveToXML writes the entire text of the XML (see page 26410) document.

See Also

LoadFromXML (see page 26395)

SaveToStream (see page 26398)

SaveToFile (see page 26397)

XML (see page 26410)

197.2.1.2.32 IXMLDocument.SetActive Method

Sets the value of the Active (see page 26403) property.

Pascal

```
procedure SetActive(const Value: Boolean);
```

C++

```
__fastcall SetActive(const Boolean Value);
```

Description

SetActive is the write implementation of the Active (see page 26403) property.

Value is the new value to set.

See Also

Active (see page 26403)

197.2.1.2.33 IXMLDocument.SetDocumentElement Method

Sets the value of the DocumentElement (see page 26405) property.

Pascal

```
procedure SetDocumentElement(const Value: IXMLNode);
```

C++

```
__fastcall SetDocumentElement(const IXMLNode Value);
```

Description

SetDocumentElement is the write implementation of the DocumentElement (see page 26405) property.

Value is the interface for the node tree that replaces the current document element.

See Also

DocumentElement (see page 26405)

197.2.1.2.34 IXMLDocument.SetDOMDocument Method

Sets the value of the DOMDocument (see page 26405) property.

Pascal

```
procedure SetDOMDocument(const Value: IDOMDocument);
```

C++

```
__fastcall SetDOMDocument(const IDOMDocument Value);
```

Description

SetDOMDocument is the write implementation of the DOMDocument (see page 26405) property.

Value is the DOM interface for the document to assign.

See Also

DOMDocument (see page 26405)

197.2.1.2.35 IXMLDocument.SetEncoding Method

Sets the value of the Encoding (see page 26406) property.

Pascal

```
procedure SetEncoding(const Value: DOMString);
```

C++

```
__fastcall SetEncoding(const DOMString Value);
```

Description

SetEncoding is the write implementation of the Encoding (see page 26406) property.

Value is the value of the encoding attribute of the document type node.

See Also

Encoding (see page 26406)

197.2.1.2.36 IXMLDocument.SetFileName Method

Sets the value of the FileName (see page 26406) property.

Pascal

```
procedure SetFileName(const Value: DOMString);
```

C++

```
__fastcall SetFileName(const DOMString Value);
```

Description

SetFileName is the write implementation of the FileName (see page 26406) property.

Value is the name of an XML (see page 26410) document or, if the vendor supports it, an URL for the XML (see page 26410) document.

See Also

FileName (see page 26406)

197.2.1.2.37 IXMLDocument.SetNodeIndentStr Method

Sets the value of the NodeIndentStr (see page 26408) property.

Pascal

```
procedure SetNodeIndentStr(const Value: DOMString);
```

C++

```
__fastcall SetNodeIndentStr(const DOMString Value);
```

Description

SetNodeIndentStr is the write implementation of the NodeIndentStr (see page 26408) property.

Value is the string that is inserted before nested nodes to indicate a level of nesting.

See Also

NodeIndentStr (see page 26408)

197.2.1.2.38 IXMLDocument.SetOnAsyncLoad Method

Supplies an event handler that responds when the AsyncLoadState (see page 26404) property changes.

Pascal

```
procedure SetOnAsyncLoad(const Value: TAsyncEventHandler);
```

C++

```
__fastcall SetOnAsyncLoad(const TAsyncEventHandler Value);
```

Description

Use SetOnAsyncLoad to supply a callback that responds when the asynchronous parsing of an XML (see page 26410) document progresses to a certain point.

Value is a method that is called when the AsyncLoadState (see page 26404) property changes to reflect a new stage in the parsing process.

See Also

AsyncLoadState (see page 26404)

197.2.1.2.39 IXMLDocument.SetOptions Method

Sets the value of the Options (see page 26408) property.

Pascal

```
procedure SetOptions(const Value: TXMLDocOptions);
```

C++

```
__fastcall SetOptions(const TXMLDocOptions Value);
```

Description

GetOptions (see page 26392) is the write implementation of the Options (see page 26408) property.

Value is the set of options to assign.

See Also

Options (see page 26408)

197.2.1.2.40 IXMLDocument.SetParseOptions Method

Sets the value of the ParseOptions (see page 26409) property.

Pascal

```
procedure SetParseOptions(const Value: TParseOptions);
```

C++

```
__fastcall SetParseOptions(const TParseOptions Value);
```

Description

GetParseOptions (see page 26392) is the write implementation of the ParseOptions (see page 26409) property.

Value is the set of parser options to assign.

See Also

ParseOptions (see page 26409)

197.2.1.2.41 IXMLDocument.SetStandAlone Method

Sets the value of the StandAlone (see page 26410) property.

Pascal

```
procedure SetStandAlone(const Value: DOMString);
```

C++

```
__fastcall SetStandAlone(const DOMString Value);
```

Description

SetStandAlone is the write implementation of the StandAlone (see page 26410) property.

Value is the value of the stand-alone attribute of the document type node.

See Also

StandAlone (see page 26410)

197.2.1.2.42 IXMLDocument.SetVersion Method

Sets the value of the Version (see page 26410) property.

Pascal

```
procedure SetVersion(const Value: DOMString);
```

C++

```
__fastcall SetVersion(const DOMString Value);
```

Description

SetVersion is the write implementation of the Version (see page 26410) property.

Value is the value of the version attribute of the document type node.

See Also

Version (see page 26410)

197.2.1.2.43 IXMLDocument.SetXML Method

Sets the value of the XML (see page 26410) property.

Pascal

```
procedure SetXML(const Value: TStrings);
```

C++

```
__fastcall SetXML(const TStrings Value);
```

Description

SetXML is the write implementation of the XML (see page 26410) property.

Value contains the raw (unparsed) XML (see page 26410) to assign.

See Also

XML (see page 26410)

197.2.1.3 IXMLDocument Properties

The properties of the IXMLDocument class are listed here.

Public Properties

	Name	Description
	Active (see page 26403)	Specifies whether the XML (see page 26410) document has been parsed so that the DOM interfaces are available to examine or change its contents.
R	AsyncLoadState (see page 26404)	Indicates the state of the DOM parser when it parses the XML (see page 26410) document asynchronously.
R	ChildNodes (see page 26405)	Lists all of the child nodes of the document.
	DocumentElement (see page 26405)	Provides access to the root node of the XML (see page 26410) document.
	DOMDocument (see page 26405)	Provides access to the DOM interface for the parsed XML (see page 26410) document.
	Encoding (see page 26406)	Indicates the character set that the XML (see page 26410) document uses to encode characters.
	FileName (see page 26406)	Indicates the XML (see page 26410) document file, if any, that this interface represents.
R	Modified (see page 26407)	Indicates whether the XML (see page 26410) document has been modified.
R	Node (see page 26407)	Provides access to the document node for the XML (see page 26410) document.
	NodeIndentStr (see page 26408)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 26410) text.
	Options (see page 26408)	Influences various aspects of the document's behavior.
	ParseOptions (see page 26409)	Specifies implementation-specific options that are followed when parsing the XML (see page 26410) document.
R	SchemaRef (see page 26409)	Returns the name of the schema associated with this XML (see page 26410) document.
	StandAlone (see page 26410)	Indicates whether the XML (see page 26410) document includes any external markup declarations.
	Version (see page 26410)	Indicates the XML (see page 26410) version that the document uses.
	XML (see page 26410)	Specifies the content of the XML document associated with this IXMLDocument (see page 26382) interface.

197.2.1.3.1 IXMLDocument.Active Property

Specifies whether the XML (see page 26410) document has been parsed so that the DOM interfaces are available to examine or change its contents.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before you can use IXMLDocument (see page 26382) to examine or change the contents of an XML (see page 26410) document, it must be active. Setting Active to true results in the parsing of the XML (see page 26410) document specified by the FileName (see page 26406) property or by the XML (see page 26410) property. If both FileName (see page 26406) and XML (see page 26410) are empty strings, Setting Active to true causes IXMLDocument (see page 26382) to represent an empty document.

Active must be true before you can access the DOMDocument (see page 26405) or DocumentElement (see page 26405) properties.

See Also

DOMDocument (see page 26405)

DocumentElement (see page 26405)

FileName (see page 26406)

XML (see page 26410)

197.2.1.3.2 IXMLDocument.AsyncLoadState Property

Indicates the state of the DOM parser when it parses the XML (see page 26410) document asynchronously.

Pascal

```
property AsyncLoadState: Integer;
```

C++

```
__property int AsyncLoadState;
```

Description

Use AsyncLoadState to determine that status of the DOM parser when it is parsing an XML (see page 26410) document asynchronously. The value of AsyncLoadState is vendor-specific. That is, each DOM parser that loads and parses an XML (see page 26410) document asynchronously supplies its own flags to indicate state information.

When the DOM vendor is MSXML, AsyncLoadState can have any of the following values:

Value	Meaning
0	Uninitialized: The AsyncLoadState is not assigned. (Usually indicates that the parser has not tried to parse a document or it is not parsing the document asynchronously.)
1	Loading: The parser is currently parsing the structure of the XML (see page 26410) document.
2	Loaded: The parser has finished reading and parsing the structure of the XML (see page 26410) document, but the corresponding DOM interfaces are not yet available.
3	Interactive: The parser has parsed the structure of the XML (see page 26410) document, the corresponding DOM interfaces are available, but the parser has not finished reading all the node values.
4	Complete: The parser has fully read and parsed the XML (see page 26410) document.

See Also

SetOnAsyncLoad (see page 26401)

197.2.1.3.3 IXMLDocument.ChildNodes Property

Lists all of the child nodes of the document.

Pascal

```
property ChildNodes: IXMLNodeList;
```

C++

```
__property IXMLNodeList ChildNodes;
```

Description

Use ChildNodes to access all of the nodes that are children of the document object. In addition to DocumentElement (see page 26405), which is the root of the data hierarchy, ChildNodes can include nodes for comments, processing instructions, and so on.

See Also

DocumentElement (see page 26405)

Node (see page 26407)

197.2.1.3.4 IXMLDocument.DocumentElement Property

Provides access to the root node of the XML (see page 26410) document.

Pascal

```
property DocumentElement: IXMLNode;
```

C++

```
__property IXMLNode DocumentElement;
```

Description

Use DocumentElement to access the root node of the XML (see page 26410) document. Because XML (see page 26410) documents are hierarchical in structure, DocumentElement is the entry point to all data in the XML (see page 26410) document. Setting DocumentElement replaces the entire data hierarchy with a new hierarchy of nodes.

The IXMLNode (see page 26411) interface includes properties and methods for examining and changing the text and attributes of a node, for adding and deleting child nodes, and for traversing the hierarchy of nodes rooted at that node.

See Also

DOMDocument (see page 26405)

ChildNodes (see page 26405)

197.2.1.3.5 IXMLDocument.DOMDocument Property

Provides access to the DOM interface for the parsed XML (see page 26410) document.

Pascal

```
property DOMDocument: IDOMDocument;
```

C++

```
__property IDOMDocument DOMDocument;
```

Description

Use DOMDocument to directly access the IDOMDocument interface for the parsed XML (see page 26410) document. This is the interface supported by the DOM parser for the document.

Most applications do not need to use the DOMDocument property directly. Instead, it supplies the underlying support for parsing and manipulating the nodes accessed through DocumentElement (see page 26405). If you do want to make use of DOMDocument's low-level support, you must be familiar with the W3C Document Object Model (DOM) Specification.

See Also

DocumentElement (see page 26405)

197.2.1.3.6 IXMLDocument.Encoding Property

Indicates the character set that the XML (see page 26410) document uses to encode characters.

Pascal

```
property Encoding: DOMString;
```

C++

```
__property DOMString Encoding;
```

Description

Encoding is the encoding name that appears in the document type declaration of the XML (see page 26410) document. It is a string such as 'UTF-8', 'UTF-16', or 'ISO-10646-UCS-2' that indicates the character set used in the document.

See Also

FileName (see page 26406)

DOMDocument (see page 26405)

XML (see page 26410)

197.2.1.3.7 IXMLDocument.FileName Property

Indicates the XML (see page 26410) document file, if any, that this interface represents.

Pascal

```
property FileName: DOMString;
```

C++

```
__property DOMString FileName;
```

Description

Use FileName to get or set the name of the XML (see page 26410) file that this document represents. FileName is set automatically when an application calls the LoadFromFile (see page 26394) method or when this interface is obtained from the global LoadXMLDocument function.

When you set the value of FileName, the XML (see page 26410) property automatically changes (at runtime) to represent the contents of that file.

Note: The content of this XML (see page 26410) document, which is available as the value of the XML (see page 26410) property, may not be the same as the content of the specified file. If an application uses IXMLDocument (see page 26382) to

edit the document, those changes are not saved back to the XML (see page 26410) document file until you call the SaveToFile (see page 26397) method.

Tip: If the underlying DOM implementation supports it, you can use FileName to specify an URL rather than a file on disk.

See Also

SaveToFile (see page 26397)

LoadFromFile (see page 26394)

LoadXMLDocument (see page 26241)

XML (see page 26410)

197.2.1.3.8 IXMLDocument.Modified Property

Indicates whether the XML (see page 26410) document has been modified.

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

Read Modified to determine whether the application has made any changes to the nodes in the XML (see page 26410) document. When Modified is true, the value of the XML (see page 26410) property may differ from the contents of the file specified by FileName (see page 26406). You can call the SaveToFile (see page 26397) method to update the XML (see page 26410) document file to reflect the changes.

See Also

Resync (see page 26397)

SaveToFile (see page 26397)

XML (see page 26410)

FileName (see page 26406)

197.2.1.3.9 IXMLDocument.Node Property

Provides access to the document node for the XML (see page 26410) document.

Pascal

```
property Node: IXMLNode;
```

C++

```
__property IXMLNode Node;
```

Description

Read Node to access the document node for the XML (see page 26410) document. The document node represents information about the entire document. It may have several child nodes, which represent comments, processing instructions, and which include the DocumentElement (see page 26405) node that acts as the root of the data hierarchy.

See Also

DocumentElement (see page 26405)

ChildNodes (see page 26405)

197.2.1.3.10 IXMLDocument.NodeIndentStr Property

Indicates the string that is inserted before nested nodes in the formatted XML (see page 26410) text.

Pascal

```
property NodeIndentStr: DOMString;
```

C++

```
__property DOMString NodeIndentStr;
```

Description

Use NodeIndentStr to customize how the document formats nodes that are added. When the Options (see page 26408) property includes doNodeAutoIndent, this string is inserted in the generated XML (see page 26410) before any newly added node, once for each nesting level.

If you do not assign a value to NodeIndentStr, IXMLDocument (see page 26382) inserts two spaces to represent each nesting level.

See Also

Options (see page 26408)

NestingLevel (see page 26180)

FormatXMLData (see page 26240)

XML (see page 26410)

197.2.1.3.11 IXMLDocument.Options Property

Influences various aspects of the document's behavior.

Pascal

```
property Options: TXMLDocOptions;
```

C++

```
__property TXMLDocOptions Options;
```

Description

Set Options to configure how you want the document to respond when the application tries to read nodes or attributes that do not exist, when it creates a new namespace when adding a node, when it generates XML (see page 26410) text from the nodes in the document, or whether it saves its value automatically when you close the document. Options can include any of the following values:

Value	Meaning
doNodeAutoCreate	If the application tries to read a node by name, using the Nodes property of an IXMLNodeList (see page 26445) interface, and that node does not exist, then the application creates a new node using the specified name.
doNodeAutoIndent	When formatting the XML (see page 26410) text from the parsed set of nodes, child nodes are automatically indented from their parent nodes.

doAttrNull	When reading the value of an attribute that does not exist, the value is given as a Null Variant (as opposed to a value of an empty string).
doAutoPrefix	When generating the name of a new node, the value is automatically given a namespace prefix if a namespace URI is supplied.
doNamespaceDecl	When the AddChild (see page 26385) method must create a new namespace for the node, it also adds a namespace attribute to declare that new namespace. If doNamespaceDecl is not included, AddChild (see page 26385) creates the new namespace but does not add an attribute to declare it.
doAutoSave	When you close the document (set Active (see page 26403) to false), any changes are automatically saved back to the file specified by FileName (see page 26406) or to the XML (see page 26410) property.

See Also

Nodes (see page 26456)

Attributes (see page 26430)

AddChild (see page 26414)

197.2.1.3.12 IXMLDocument.ParseOptions Property

Specifies implementation-specific options that are followed when parsing the XML (see page 26410) document.

Pascal

```
property ParseOptions: TParseOptions;
```

C++

```
__property TParseOptions ParseOptions;
```

Description

Set ParseOptions to influence the way the DOM implementation parses the XML (see page 26410) document. Different DOM implementations support different options. If you attempt to use an option that is not supported by your DOM implementation, IXMLDocument (see page 26382) raises an EXMLDocError (see page 26381) exception.

See Also

domimplementation (see page 26137)

EXMLDocError (see page 26381)

197.2.1.3.13 IXMLDocument.SchemaRef Property

Returns the name of the schema associated with this XML (see page 26410) document.

Pascal

```
property SchemaRef: DOMString;
```

C++

```
__property DOMString SchemaRef;
```

Description

Read SchemaRef to determine the schema associated with this XML (see page 26410) document. If the document does not have an attribute that identifies an associated schema, the value of SchemaRef is an empty string.

See Also

DOMDocument (see page 26405)

197.2.1.3.14 IXMLDocument.StandAlone Property

Indicates whether the XML (see page 26410) document includes any external markup declarations.

Pascal

```
property StandAlone: DOMString;
```

C++

```
__property DOMString StandAlone;
```

Description

StandAlone is the value of the stand-alone attribute on the document type declaration of the XML (see page 26410) document. StandAlone has a value of 'yes' or 'no'.

When StandAlone is 'yes', the document contains no external markup declarations. It may, however, still contain references to external entities, as long as they are not declarations.

When StandAlone is 'no', the document may or may not contain external markup declarations.

See Also

DOMDocument (see page 26405)

197.2.1.3.15 IXMLDocument.Version Property

Indicates the XML (see page 26410) version that the document uses.

Pascal

```
property Version: DOMString;
```

C++

```
__property DOMString Version;
```

Description

Version indicates the XML (see page 26410) version number that the document uses. An XML (see page 26410) version number is a string of the form 'n.m', where n is the major version number and m is the minor version number.

See Also

XML (see page 26410)

197.2.1.3.16 IXMLDocument.XML Property

Specifies the content of the XML document associated with this IXMLDocument (see page 26382) interface.

Pascal

```
property XML: TStrings;
```

C++

```
__property TStrings XML;
```

Description

Read XML to obtain the contents of the associated XML document. Set XML to specify the raw (unparsed) XML you want to

examine or manipulate.

XML is a TStrings object. This means you can manipulate the XML line by line. To read or write the XML in the document as a single string, use the LoadFromXML (see page 26395) or SaveToXML (see page 26398) method.

Note: If you call the LoadFromFile (see page 26394), LoadFromStream (see page 26395), or LoadFromXML (see page 26395) method, XML is automatically updated to reflect the contents of that file.

See Also

LoadFromFile (see page 26394)

DocumentElement (see page 26405)

TXMLDocument (see page 26097)

LoadFromXML (see page 26395)

SaveToXML (see page 26398)

LoadFromStream (see page 26395)

197.2.2 IXMLNode Interface

IXMLNode is the interface for a node in an XML (see page 26439) document.

Class Hierarchy

XMLIntf.IXMLNode

Pascal

```
IXMLNode = interface;
```

C++

```
__interface IXMLNode;
```

File

XMLIntf

Description

Use IXMLNode to work with a node in an XML (see page 26439) document. The IXMLNode interface is designed to work with TXMLDocument and the code generated by the Data Binding wizard.

IXMLNode is similar to the IDOMNode interface that is defined as part of the Document Object Model (DOM) specification. However, IXMLNode is designed to simplify the task of navigating and editing XML (see page 26439) documents by

including some of the properties and methods from the IDOMElement interface, resulting in a simpler model for examining an XML (see page 26439) node.

including additional properties and methods to simplify access to data in common XML (see page 26439) structures.

The Data Binding wizard generates descendants of IXMLNode that further simplify the task of working with XML (see page 26439) documents by converting the attributes and child nodes of a node into properties of the node interface.

Note: C++ method declarations that use IXMLNode use the _di_IXMLNode type instead. This type is a DelphiInterface wrapper around the IXMLNode interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IXMLNode >  _di_IXMLNode;
```

See Also

TXMLNode (see page 26147)

TXMLDocument (see page 26097)

IXMLNodeCollection (see page 26440)

IDOMNode (see page 26306)

IXMLNodeList (see page 26445)

Working with XML Nodes

197.2.2.1 IXMLNode Members

The following tables list the members exposed by IXMLNode.

Public Methods

	Name	Description
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.

	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

197.2.2.2 IXMLNode Methods

The methods of the IXMLNode class are listed here.

Public Methods

	Name	Description
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.

	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

197.2.2.2.1 AddChild Method

This is the overview for the AddChild method overload.

Overload List

	Name	Description
	IXMLNode.AddChild (DOMString, DOMString, Boolean, Integer) (see page 26414)	Adds a new child node to this node.
	IXMLNode.AddChild (DOMString, Integer) (see page 26414)	Adds a new child node to this node.

197.2.2.2.1.1 IXMLNode.AddChild Method (DOMString, DOMString, Boolean, Integer)

Adds a new child node to this node.

Pascal

```
function AddChild(const TagName: DOMString; Index: Integer = -1): IXMLNode; overload;
function AddChild(const TagName: DOMString; const NamespaceURI: DOMString; GenPrefix: Boolean
= False; Index: Integer = -1): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode AddChild(const DOMString TagName, int Index = -1);
__fastcall IXMLNode AddChild(const DOMString TagName, const DOMString NamespaceURI, Boolean
GenPrefix = False, int Index = -1);
```

Description

AddChild creates a new element node as the child of this node.

TagName provides the tag name of the newly created node.

NamespaceURI (see page 26435) identifies the namespace that includes the new node's definition. If NamespaceURI (see page 26435) is omitted (the second syntax), the namespace of the new node is deduced from the namespace prefix of TagName. If TagName has no namespace prefix and there is no NamespaceURI (see page 26435) parameter, the

namespace is the same as this node's NamespaceURI (see page 26435) property.

GenPrefix controls whether AddChild generates a namespace prefix for the namespace URI when the NamespaceURI (see page 26435) parameter identifies a URI that is not already declared in the document. Note that setting GenPrefix to true does not cause AddChild to add a declaration of the new namespace prefix unless the document's Options property includes doNamespaceDecl.

Index indicates the position of the child node in this node's list of children, where 0 is the first position, 1 is the second position, and so on. If Index is –1, the new node is added to the end.

AddChild returns the interface for the newly created child node.

Tip: AddChild only adds element nodes. To add other types of child nodes, use the XML (see page 26439) document's CreateNode method instead, and then add the resulting node to the ChildNodes (see page 26431) property array.

See Also

Delete (see page 26449)

ChildNodes (see page 26431)

CreateNode (see page 26387)

NamespaceURI (see page 26435)

Options (see page 26141)

Working with XML Nodes

197.2.2.2.2 IXMLNode.CloneNode Method

Creates a copy of this node and returns its interface.

Pascal

```
function CloneNode(Deep: Boolean): IXMLNode;
```

C++

```
__fastcall IXMLNode CloneNode(Boolean Deep);
```

Description

CloneNode returns a copy of this node that has no parent. Any attributes of this node are copied as well, including attribute nodes that represent default values.

Deep indicates whether the child nodes of this node should be cloned as well. For example, if an element node contains any text, CloneNode only copies then text when deep is true, because in the underlying DOM implementation, the text is contained in a child node.

After calling CloneNode to create a new node, you can add that node into the ChildNodes (see page 26431) property of another node (or, if this node represents an attribute, into the AttributeNodes (see page 26430) property) by calling ChildNodes (see page 26431). ->Add or ChildNodes (see page 26431). ->Insert.

See Also

ChildNodes (see page 26431)

AddChild (see page 26414)

ParentNode (see page 26437)

CloneNode (see page 26309)

Add (see page 26447)

Insert (see page 26454)

197.2.2.2.3 IXMLNode.DeclareNamespace Method

Adds an attribute to the node that declares a specified namespace URI.

Pascal

```
procedure DeclareNamespace(const Prefix: DOMString; const URI: DOMString);
```

C++

```
__fastcall DeclareNamespace(const DOMString Prefix, const DOMString URI);
```

Description

DeclareNamespace adds an attribute to this node that declares the specified namespace URI as the value of the specified prefix.

Prefix (see page 26438) is the namespace prefix that appears in tag names that reference the namespace.

URI is the corresponding namespace URI.

Note: This method is called automatically for newly created nodes when the document's Options property includes doNamespaceDecl.

See Also

Options (see page 26408)

Attributes (see page 26430)

FindNamespaceDecl (see page 26416)

197.2.2.2.4 IXMLNode.FindNamespaceDecl Method

Returns the attribute node that declares a specified Namespace URI.

Pascal

```
function FindNamespaceDecl(const NamespaceURI: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode FindNamespaceDecl(const DOMString NamespaceURI);
```

Description

FindNamespaceDecl searches the node hierarchy that contains this node for an attribute that specifies the mapping between a namespace prefix and the namespace URI identified by NamespaceURI (see page 26435). If it finds such a mapping, FindNamespaceDecl returns the interface for the Attribute node. If it does not find a mapping, FindNamespaceDecl returns nil (Delphi) or NULL (C++).

NamespaceURI (see page 26435) is the namespace URI to locate.

See Also

FindNamespaceURI (see page 26416)

DeclareNamespace (see page 26416)

197.2.2.2.5 IXMLNode.FindNamespaceURI Method

Returns the namespace URI for a namespace prefix or qualified tag name.

Pascal

```
function FindNamespaceURI(const TagOrPrefix: DOMString): DOMString;
```

C++

```
__fastcall DOMString FindNamespaceURI(const DOMString TagOrPrefix);
```

Description

FindNamespaceDecl (see page 26416) searches the node hierarchy that contains this node for an attribute that specifies the mapping between the namespace prefix identified by TagOrPrefix and a namespace URI. If it finds such a mapping, FindNamespaceURI returns the namespace URI. If it does not find a mapping, FindNamespaceURI returns an empty string.

TagOrPrefix is the namespace prefix or a tag name that includes the namespace prefix.

See Also

GetNamespaceURI (see page 26421)

GetPrefix (see page 26422)

FindNamespaceDecl (see page 26416)

197.2.2.2.6 IXMLNode.GetAttribute Method

Returns the value of one of this node's attributes.

Pascal

```
function GetAttribute(const AttrName: DOMString): OleVariant;
```

C++

```
__fastcall OleVariant GetAttribute(const DOMString AttrName);
```

Description

GetAttribute is the read implementation of the Attributes (see page 26430) property.

AttrName is the name of the attribute.

GetAttribute returns the value of the specified attribute. Typically, this value is a string. If the node does not have an attribute with the name specified by AttrName, GetAttribute returns a Null Variant.

Note: Do not call GetAttribute unless this node is an element node. Only element nodes have attributes.

See Also

Attributes (see page 26430)

GetAttributeNS (see page 26418)

197.2.2.2.7 IXMLNode.GetAttributeNodes Method

Returns the value of the AttributeNodes (see page 26430) property.

Pascal

```
function GetAttributeNodes: IXMLNodeList;
```

C++

```
__fastcall IXMLNodeList GetAttributeNodes();
```

Description

GetAttributeNodes is the read implementation of the AttributeNodes (see page 26430) property.

See Also

AttributeNodes (see page 26430)

197.2.2.2.8 IXMLNode.GetAttributeNS Method

Returns the value of a specified attribute.

Pascal

```
function GetAttributeNS(const AttrName: DOMString; const NamespaceURI: DOMString): OleVariant;
```

C++

```
__fastcall OleVariant GetAttributeNS(const DOMString AttrName, const DOMString NamespaceURI);
```

Description

Call GetAttributeNS to determine the value of an attribute when you must explicitly specify the namespace for the attribute.

AttrName is the name of the attribute whose value you want to fetch.

NamespaceURI (see page 26435) identifies the namespace in which the attribute resides.

GetAttributeNS returns the value of the attribute. If AttrName and NamespaceURI (see page 26435) do not identify an attribute of this node, GetAttributeNS returns a Null Variant.

See Also

AttributeNodes (see page 26430)

Attributes (see page 26430)

NamespaceURI (see page 26435)

SetAttributeNS (see page 26426)

197.2.2.2.9 IXMLNode.GetChildNodes Method

Returns the value of the ChildNodes (see page 26431) property.

Pascal

```
function GetChildNodes: IXMLNodeList;
```

C++

```
__fastcall IXMLNodeList GetChildNodes();
```

Description

GetChildNodes is the read implementation of the ChildNodes (see page 26431) property.

See Also

ChildNodes (see page 26431)

197.2.2.2.10 IXMLNode.GetChildValue Method

Returns the value of a specified child node.

Pascal

```
function GetChildValue(const IndexOrName: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant GetChildValue(const OleVariant IndexOrName);
```

Description

GetChildValue is the read implementation of the ChildValues (see page 26432) property.

IndexOrName identifies the desired child node by either its interface, or by a string that is the local name of the child node.

The return value depends on the type of the child node, as indicated in the following table:

NodeType (see page 26436)	Value
ntAttribute	The attribute value
ntElement	If the element contains only text, this is that text value. Otherwise, reading the child's value causes an exception.
ntText	The text
ntCData	The content of the CDATA section.
ntEntityRef	nil (Delphi) or NULL (C++)
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.
ntDocFragment	nil (Delphi) or NULL (C++)

See Also

ChildValues (see page 26432)

197.2.2.2.11 IXMLNode.GetCollection Method

Returns the value of the Collection (see page 26432) property.

Pascal

```
function GetCollection: IXMLNodeCollection;
```

C++

```
__fastcall IXMLNodeCollection GetCollection();
```

Description

GetCollection is the read implementation of the Collection (see page 26432) property.

See Also

Collection (see page 26432)

197.2.2.2.12 IXMLNode.GetDOMNode Method

Returns the value of the DOMNode (see page 26433) property.

Pascal

```
function GetDOMNode: IDOMNode;
```

C++

```
__fastcall IDOMNode GetDOMNode();
```

Description

GetDOMNode is the read implementation of the DOMNode (see page 26433) property.

See Also

DOMNode (see page 26433)

197.2.2.2.13 IXMLNode.GetHasChildNodes Method

Indicates whether the node has any child nodes.

Pascal

```
function GetHasChildNodes: Boolean;
```

C++

```
__fastcall Boolean GetHasChildNodes();
```

Description

GetHasChildNodes is the read implementation of the HasChildNodes (see page 26433) property.

See Also

HasChildNodes (see page 26433)

197.2.2.2.14 IXMLNode.GetIsTextElement Method

Indicates whether the node has a single text value.

Pascal

```
function GetIsTextElement: Boolean;
```

C++

```
__fastcall Boolean GetIsTextElement();
```

Description

GetIsTextElement is the read implementation of the IsTextElement (see page 26434) property.

See Also

IsTextElement (see page 26434)

197.2.2.2.15 IXMLNode.GetLocalName Method

Returns the name of the node without any namespace prefix.

Pascal

```
function GetLocalName: DOMString;
```

C++

```
__fastcall DOMString GetLocalName();
```

Description

GetLocalName is the read implementation of the LocalName (see page 26434) property.

See Also

LocalName (see page 26434)

197.2.2.2.16 IXMLNode.GetNamespaceURI Method

Returns the namespace URI for the node's qualified name.

Pascal

```
function GetNamespaceURI: DOMString;
```

C++

```
__fastcall DOMString GetNamespaceURI();
```

Description

GetNamespaceURI is the read implementation of the NamespaceURI (see page 26435) property.

See Also

NamespaceURI (see page 26435)

197.2.2.2.17 IXMLNode.GetNodeName Method

Returns the name of the node.

Pascal

```
function GetNodeName: DOMString;
```

C++

```
__fastcall DOMString GetNodeName();
```

Description

GetNodeName is the read implementation of the NodeName (see page 26435) property.

See Also

NodeName (see page 26435)

197.2.2.2.18 IXMLNode.GetNodeType Method

Returns the type of the node.

Pascal

```
function GetNodeType: TNodeType;
```

C++

```
__fastcall TNodeType GetNodeType();
```

Description

GetNodeType is the read implementation of the NodeType (see page 26436) property.

See Also

NodeType (see page 26436)

197.2.2.2.19 IXMLNode.GetNodeValue Method

Returns the value of the node.

Pascal

```
function GetNodeValue: OleVariant;
```

C++

```
__fastcall OleVariant GetNodeValue();
```

Description

GetNodeValue is the read implementation of the NodeValue (see page 26436) property.

See Also

NodeValue (see page 26436)

197.2.2.2.20 IXMLNode.GetOwnerDocument Method

Returns the interface for the document in which this node appears.

Pascal

```
function GetOwnerDocument: IXMLDocument;
```

C++

```
__fastcall IXMLDocument GetOwnerDocument();
```

Description

GetOwnerDocument is the read implementation of the OwnerDocument (see page 26437) property.

See Also

OwnerDocument (see page 26437)

197.2.2.2.21 IXMLNode.GetParentNode Method

Returns the interface for the node's parent node.

Pascal

```
function GetParentNode: IXMLNode;
```

C++

```
__fastcall IXMLNode GetParentNode();
```

Description

GetParentNode is the read implementation of the ParentNode (see page 26437) property.

See Also

ParentNode (see page 26437)

197.2.2.2.22 IXMLNode.GetPrefix Method

Returns the namespace prefix from the node's name.

Pascal

```
function GetPrefix: DOMString;
```

C++

```
__fastcall DOMString GetPrefix();
```

Description

GetPrefix is the read implementation of the Prefix (see page 26438) property.

See Also

Prefix (see page 26438)

197.2.2.2.23 IXMLNode.GetReadOnly Method

Returns the value of the ReadOnly (see page 26438) property.

Pascal

```
function GetReadOnly: Boolean;
```

C++

```
__fastcall Boolean GetReadOnly();
```

Description

GetReadOnly is the read implementation of the ReadOnly (see page 26438) property.

See Also

ReadOnly (see page 26438)

197.2.2.2.24 IXMLNode.GetText Method

Returns the text value of the node.

Pascal

```
function GetText: DOMString;
```

C++

```
__fastcall DOMString GetText();
```

Description

GetText is the read implementation for the Text (see page 26439) property.

See Also

Text (see page 26439)

197.2.2.2.25 IXMLNode.GetXML Method

Returns the value of the XML (see page 26439) property.

Pascal

```
function GetXML: DOMString;
```

C++

```
__fastcall DOMString GetXML();
```

Description

GetXML is the read implementation of this node's XML (see page 26439) property.

See Also

XML (see page 26439)

197.2.2.2.26 HasAttribute Method

This is the overview for the HasAttribute method overload.

Overload List

	Name	Description
	IXMLNode.HasAttribute (DOMString) (see page 26424)	Indicates whether the node has an attribute with a specified name.
	IXMLNode.HasAttribute (DOMString, DOMString) (see page 26424)	Indicates whether the node has an attribute with a specified name.

197.2.2.2.26.1 IXMLNode.HasAttribute Method (DOMString)

Indicates whether the node has an attribute with a specified name.

Pascal

```
function HasAttribute(const Name: DOMString): Boolean; overload;
function HasAttribute(const Name: DOMString; const NamespaceURI: DOMString): Boolean; overload;
```

C++

```
__fastcall Boolean HasAttribute(const DOMString Name);
__fastcall Boolean HasAttribute(const DOMString Name, const DOMString NamespaceURI);
```

Description

Call HasAttribute to determine whether this node has a specified attribute.

Name is the name of the attribute about which you want to know.

NamespaceURI (see page 26435) is the namespace of the attribute.

HasAttribute returns true if this node has the specified attribute, false otherwise.

See Also

AttributeNodes (see page 26430)

Attributes (see page 26430)

197.2.2.2.27 IXMLNode.NextSibling Method

Returns the next child of this node's parent.

Pascal

```
function NextSibling: IXMLNode;
```

C++

```
__fastcall IXMLNode NextSibling();
```

Description

NextSibling returns the node that follows this one in the parent node's ChildNodes (see page 26431) property list.

If this node is the last node in its parent's child list, NextSibling raises an exception.

See Also

ParentNode (see page 26437)

PreviousSibling (see page 26425)

ChildNodes (see page 26431)

197.2.2.2.28 IXMLNode.Normalize Method

Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.

Pascal

```
procedure Normalize;
```

C++

```
__fastcall Normalize();
```

Description

Use the IXMLNode (see page 26411) interface to call the protected Normalize method. Normalize consolidates adjacent text nodes and remove any empty text nodes in the subtree beneath this node. This method ensures that for all child nodes of this node and of its attributes, only structure nodes (element nodes, comments, processing instructions, CDATA sections, and entity references) separate text nodes and that there are no empty text nodes.

See Also

ChildNodes (see page 26431)

AttributeNodes (see page 26430)

197.2.2.2.29 IXMLNode.PreviousSibling Method

Returns the previous child of this node's parent.

Pascal

```
function PreviousSibling: IXMLNode;
```

C++

```
__fastcall IXMLNode PreviousSibling();
```

Description

PreviousSibling returns the node that precedes this one in the parent node's ChildNodes (see page 26431) property list.

If this node is the first node in its parent's child list, PreviousSibling raises an exception.

See Also

ParentNode (see page 26437)

NextSibling (see page 26424)

ChildNodes (see page 26431)

197.2.2.2.30 IXMLNode.Resync Method

Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

Pascal

```
procedure Resync;
```

C++

```
__fastcall Resync();
```

Description

Resync forces the node to reread the values of its child nodes and attribute nodes, discarding any changes that have not been propagated to the underlying DOM implementation.

See Also

ChildNodes (see page 26431)

AttributeNodes (see page 26430)

Refresh (see page 26123)

197.2.2.2.31 IXMLNode.SetAttribute Method

Sets the value of one of this node's attributes.

Pascal

```
procedure SetAttribute(const AttrName: DOMString; const Value: OleVariant);
```

C++

```
__fastcall SetAttribute(const DOMString AttrName, const OleVariant Value);
```

Description

SetAttribute is the write implementation for the Attributes (see page 26430) property.

AttrName is the name of the attribute. If the attribute already exists for the node, its value is changed. If the attribute does not already exist, a new attribute is created.

Value is the value to assign to the attribute. This value is a string that is not parsed. Any markup (for example, for an entity reference) is treated as literal text, and must include appropriate escape sequences. If Value is NULL, the attribute is removed from this node. If Value is an empty string, the attribute is assigned an empty value.

See Also

Attributes (see page 26430)

SetAttributeNS (see page 26426)

197.2.2.2.32 IXMLNode.SetAttributeNS Method

Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.

Pascal

```
procedure SetAttributeNS(const AttrName: DOMString; const NamespaceURI: DOMString; const
Value: OleVariant);
```

C++

```
__fastcall SetAttributeNS(const DOMString AttrName, const DOMString NamespaceURI, const
OleVariant Value);
```

Description

SetAttributeNS sets the value of a specified attribute of this node.

AttrName is the name of the attribute. If the attribute already exists for the node, its value is changed. If the attribute does not already exist, a new attribute is created.

NamespaceURI (see page 26435) is the namespace for the attribute.

Value is the value to assign to the attribute. This value is a string that is not parsed. Any markup (for example, for an entity reference) is treated as literal text, and must include appropriate escape sequences. If Value is NULL, the attribute is removed from this node. If Value is an empty string, the attribute is assigned an empty value.

See Also

SetAttribute (see page 26426)

GetAttributeNS (see page 26418)

AttributeNodes (see page 26430)

197.2.2.2.33 IXMLNode.SetChildValue Method

Sets the value of a specified child node.

Pascal

```
procedure SetChildValue(const IndexOrName: OleVariant; const Value: OleVariant);
```

C++

```
__fastcall SetChildValue(const OleVariant IndexOrName, const OleVariant Value);
```

Description

SetChildValue is the write implementation of the ChildValues (see page 26432) property.

IndexOrName identifies the desired child node by either its interface, or by a string that is the local name of the child node.

Value is the value to assign.

See Also

ChildValues (see page 26432)

197.2.2.2.34 IXMLNode.SetNodeValue Method

Sets the value of this node.

Pascal

```
procedure SetNodeValue(const Value: OleVariant);
```

C++

```
__fastcall SetNodeValue(const OleVariant Value);
```

Description

SetNodeValue is the write implementation of the NodeValue (see page 26436) property.

Value is the value to assign.

See Also

NodeValue (see page 26436)

197.2.2.2.35 IXMLNode.SetReadOnly Method

Specifies whether the node can be modified.

Pascal

```
procedure SetReadOnly(const Value: Boolean);
```

C++

```
__fastcall SetReadOnly(const Boolean Value);
```

Description

SetReadOnly is the write implementation of the ReadOnly (see page 26438) property.

Value is the new read-only status of the node.

See Also

ReadOnly (see page 26438)

197.2.2.2.36 IXMLNode.SetText Method

Sets the text value of the node.

Pascal

```
procedure SetText(const Value: DOMString);
```

C++

```
__fastcall SetText(const DOMString Value);
```

Description

SetText is the write implementation for the Text (see page 26439) property.

Value is the new text value for the node.

See Also

Text (see page 26439)

197.2.2.2.37 TransformNode Method

This is the overview for the TransformNode method overload.

Overload List

	Name	Description
	IXMLNode.TransformNode (IXMLNode, IXMLDocument) (see page 26429)	Transforms the subtree rooted at this node, using a specified XSL interface.
	IXMLNode.TransformNode (IXMLNode, WideString) (see page 26429)	Transforms the subtree rooted at this node, using a specified XSL interface.

197.2.2.2.37.1 IXMLNode.TransformNode Method (IXMLNode, IXMLDocument)

Transforms the subtree rooted at this node, using a specified XSL interface.

Pascal

```
procedure TransformNode(const stylesheet: IXMLNode; var output: WideString); overload;
procedure TransformNode(const stylesheet: IXMLNode; const output: IXMLDocument); overload;
```

C++

```
__fastcall TransformNode(const IXMLNode stylesheet, BSTR output);
__fastcall TransformNode(const IXMLNode stylesheet, const IXMLDocument output);
```

Description

Call TransformNode to transform the XML (see page 26439) that is the value of the XML (see page 26439) property using the transformation specified by an XSL document.

stylesheet is a node in an XML (see page 26439) document that represents an XSL transformation. Typically, it is the document element of that document.

output represents the result of the transformation. If output is a WideString, TransformNode returns a string of XML (see page 26439). If output is the interface to an XML (see page 26439) document, TransformNode replaces the content of that document with the results of the transformation.

Note: TransformNode is only available when the DOMVendor used to parse this node's XML (see page 26439) document is MSXML.

See Also

XML (see page 26439)

197.2.2.3 IXMLNode Properties

The properties of the IXMLNode class are listed here.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

197.2.2.3.1 IXMLNode.AttributeNodes Property

Lists the interfaces for nodes that represent attributes of this node.

Pascal

```
property AttributeNodes: IXMLNodeList;
```

C++

```
__property IXMLNodeList AttributeNodes;
```

Description

Use AttributeNodes to access the attributes of this node using an IXMLNode (see page 26411) interface. Attributes (see page 26430) modify a node in an XML (see page 26439) document. For example, given the node for

```
<Movie genre="scifi" title="Star Wars"> ... </Movie>
```

both genre and title are attributes of the node, with values of "scifi" and "Star Wars", respectively.

If an attribute consists of a simple Name=Value pair, it is simpler to use the Attributes (see page 26430) property to read or set attribute values. AttributeNodes is an IXMLNodeList (see page 26445) interface, which can be used to

Read the count of attributes before iterating over all the attributes of this node.

Add or delete attribute nodes from this node.

Access attributes through an XML (see page 26439) node interface (for example, to read the attribute names from the nodes).

Only element nodes have attributes, and not every element node has them. If the node does not have any attributes, AttributeNodes is the interface for an empty list.

Note: Attributes (see page 26430) are not considered child nodes of a node and can't be accessed using the ChildNodes (see page 26431) property.

See Also

ChildNodes (see page 26431)

Attributes (see page 26430)

197.2.2.3.2 IXMLNode.Attributes Property

Represents the values of this node's attributes.

Pascal

```
property Attributes [const AttrName: DOMString]: OleVariant;
```

C++

```
__property OleVariant Attributes[DOMString const AttrName];
```

Description

Use Attributes to read or set the values of this node's attributes. Attributes modify a node in an XML (see page 26439) document. For example, given the node for

```
<Movie genre="scifi" title="Star Wars"> ... </Movie>
```

both genre and title are attributes of the node, with values of "scifi" and "Star Wars", respectively.

The AttrName index gives the name of the attribute to access. In the previous example, "genre" and "title" are the names of the attributes. When setting Attributes, if you specify the name of an attribute that already exists for a node, you change that attribute's value. If you specify an attribute name that does not correspond to an existing attribute of the node, a new attribute is

created.

When reading Attributes, if you specify the name of an attribute that does not exist for the node, the value depends on the Options property of the document that contains the node. If Options includes doAttrNull, reading an attribute that does not exist for the node results in a Null Variant. When Options does not include doAttrNull, reading an attribute that does not exist results in an empty string.

The value of Attributes is the text of the specified attribute. When setting attributes, the value string is not parsed. Any markup in the supplied value (for example, an entity reference) is treated as literal text. If you set an attribute to nil (Delphi) or NULL (C++), it is removed from this node. Note that you can set an attribute's value to an empty string.

See Also

AttributeNodes (see page 26430)

GetAttributeNS (see page 26418)

SetAttributeNS (see page 26426)

Options (see page 26141)

Working with XML Nodes

197.2.2.3.3 IXMLNode.ChildNodes Property

Represents the child nodes of the node.

Pascal

```
property ChildNodes: IXMLNodeList;
```

C++

```
__property IXMLNodeList ChildNodes;
```

Description

Use ChildNodes to access the children of a node. ChildNodes is an IXMLNodeList (see page 26445) interface, which can be used to access individual child nodes or to add child nodes to or delete them from this node. Each child node specifies this node as its parent node.

Tip: To get or set the values of the child nodes rather than their interfaces, use the ChildValues (see page 26432) property instead.

Note: Attributes (see page 26430) are not considered child nodes of a node. To get the values of this node's attributes, use the Attributes (see page 26430) or AttributeNodes (see page 26430) property instead.

See Also

ChildValues (see page 26432)

AttributeNodes (see page 26430)

ParentNode (see page 26437)

Attributes (see page 26430)

Working with XML Nodes

197.2.2.3.4 IXMLNode.ChildValues Property

Represents the values of this node's child nodes.

Pascal

```
property ChildValues [const IndexOrName: OleVariant]: OleVariant;
```

C++

```
__property OleVariant ChildValues[OleVariant const IndexOrName];
```

Description

Use ChildValues to access the values of this node's children.

IndexOrName identifies the desired child node either by its interface, or by a string that is the local name of the child node. If IndexOrName does not identify a child of this node, trying to get or set ChildValues causes an exception.

The value of a child node depends on the type of the child node, as indicated in the following table:

NodeType (see page 26436)	Value
ntElement	If the element contains only text, this is that text value. Otherwise, it is the Value of the first child node.
ntText	The text
ntCData	The content of the CDATA section.
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.
If the child node is any other node type, reading or setting ChildValues causes an exception.	

Note: In Delphi, ChildValues is the default property for IXMLNode (see page 26411). That means that you can omit the property name when using this property. For example

```
Edit1.Text := XMLDocument1.DocumentElement.ChildValues['Name'];
```

can also be written

```
Edit1.Text := XMLDocument1.DocumentElement['Name'];
```

Note: Attributes (see page 26430) are not considered child nodes of a node. To get or set the values of this node's attributes, use the Attributes (see page 26430) property instead.

See Also

ChildNodes (see page 26431)

ParentNode (see page 26437)

Attributes (see page 26430)

197.2.2.3.5 IXMLNode.Collection Property

Indicates the collection in which this node appears.

Pascal

```
property Collection: IXMLNodeCollection;
```

C++

```
__property IXMLNodeCollection Collection;
```

Description

Read Collection to determine whether this node is one of a repeating set of child nodes. The node specified by Collection represents the same node in the XML (see page 26439) document as this node's ParentNode (see page 26437) property, but it is specialized to work with this node and any other child nodes of the same type.

For example, consider the following XML (see page 26439) fragment:

```
<students>
  <student name="John Smith">
    <grade>incomplete</grade>
  </student>
  <student name="Jane Doe">
    <grade>B+</grade>
  </student>
  <auditor name="Bill Jones"></auditor>
</students>
```

If this node represents one of the <student> nodes, then ParentNode (see page 26437) represents the <students> node, and Collection represents a copy of the <students> node that is used for only those children that are <student> nodes, and not the <auditor> nodes.

If this node does not represent one of a set of repeating child elements, Collection is nil (Delphi) or NULL (C++).

See Also

ParentNode (see page 26437)

197.2.2.3.6 IXMLNode.DOMNode Property

Provides access to the IDOMNode interface for this node.

Pascal

```
property DOMNode: IDOMNode;
```

C++

```
__property IDOMNode DOMNode;
```

Description

IDOMNode is the interface for this node provided by the DOM implementation that parsed the XML (see page 26439) document. You can use this interface to work with the node using the lower-level properties and methods of the DOM.

Typically, applications use the IXMLNode (see page 26411) interface to work with a node rather than the IDOMNode interface. For more information on DOM interfaces, see the DOM specification on the W3 website or the documentation provided by your DOM vendor.

197.2.2.3.7 IXMLNode.HasChildNodes Property

Indicates whether this node has any child nodes.

Pascal

```
property HasChildNodes: Boolean;
```

C++

```
__property Boolean HasChildNodes;
```

Description

HasChildNodes is true if the underlying DOM node has any child nodes, false if it has no children.

Note that in the underlying DOM implementation, given the following XML (see page 26439)

```
<Title> Understanding XML </Title>
```

the "Title" node has a single child (a text node with the value 'Understanding XML (see page 26439)').

See Also

ChildNodes (see page 26431)

DOMNode (see page 26433)

197.2.2.3.8 IXMLNode.IsTextElement Property

Indicates whether the node has a single text value.

Pascal

```
property IsTextElement: Boolean;
```

C++

```
__property Boolean IsTextElement;
```

Description

Read IsTextElement to determine whether this node represents a single text value. For example, if the node represents the following:

```
<Title> Understanding XML </Title>
```

IsTextElement is true, because the node represents a tagged element and the text that is its value.

Note: In the underlying DOM implementation, nodes for which IsTextElement is true are element nodes that have a single child node that is a text node. IXMLNode (see page 26411) flattens this out so that you can use the element node to work with the text value (through the Text (see page 26439) property), rather than requiring you to use a separate node for the text value.

See Also

NodeType (see page 26436)

Text (see page 26439)

Working with XML Nodes

197.2.2.3.9 IXMLNode.LocalName Property

Indicates the name of the node without any namespace prefix.

Pascal

```
property LocalName: DOMString;
```

C++

```
__property DOMString LocalName;
```

Description

Read LocalName to determine the local name of the underlying DOM node. LocalName is only defined for nodes of type ntElement or ntAttrribute.

For element nodes, LocalName is derived from the name that appears in the XML (see page 26439) tag. For example, the

local name for <xsi:Name First="John" Last="Doe"> is 'Name'. (The string "xsi" is called the prefix).

For attribute nodes, the local name is derived from the string that appears to the left of the equals sign in the attribute declaration. For example, the element <xsi:Name xsi:First="John" xsi:Last="Doe"> results in two attribute nodes, with the local names 'First' and 'Last'.

For any other node types, localName is nil (Delphi) or NULL (C++).

To get the name of a node when localName is not defined, use the NodeName (see page 26435) property instead. If the node's name has no namespace prefix, LocalName has the same value as NodeName (see page 26435).

See Also

NamespaceURI (see page 26435)

Prefix (see page 26438)

NodeName (see page 26435)

197.2.2.3.10 IXMLNode.NamespaceURI Property

Identifies the URI for the namespace used in the qualified node name.

Pascal

```
property NamespaceURI: DOMString;
```

C++

```
__property DOMString NamespaceURI;
```

Description

NamespaceURI is the URI for the namespace that should be used to interpret the node's name.

Only element and attribute nodes can have a namespace URI value. For any other node type, NamespaceURI is an empty string.

The namespace URI is only available if it is explicitly assigned when the node is created. By default, IXMLNode (see page 26411) does not assign namespaces to the new nodes it creates.

See Also

NodeName (see page 26435)

NodeType (see page 26436)

Prefix (see page 26438)

LocalName (see page 26434)

CreateDOMNode (see page 26238)

NamespaceURI (see page 26318)

197.2.2.3.11 IXMLNode.NodeName Property

Indicates the node name.

Pascal

```
property NodeName: DOMstring;
```

C++

```
__property DOMstring NodeName;
```

Description

NodeName is the name of the underlying DOM node.

The node's name depends on the type of the node, as indicated in the following table:

NodeType (see page 26436)	NodeName
ntAttribute	The attribute name
ntElement	The tag name
ntText	'#text'
ntCData	'#cdata-section'
ntEntityRef	The name of the entity reference.
ntEntity	The entity name
ntProcessingInstr	the target of the processing instruction
ntComment	'#comment'
ntDocument	'#document'
ntDocType	The document type name
ntDocFragment	'#document-fragment'
ntNotation	The notation name

Note: If Prefix (see page 26438) and LocalName (see page 26434) are both defined for a node, the node name is the combination of these two values.

See Also

NodeType (see page 26436)

Prefix (see page 26438)

LocalName (see page 26434)

197.2.2.3.12 IXMLNode.NodeType Property

Indicates the type of the node.

Pascal

```
property NodeType: TNodeType;
```

C++

```
__property TNodeType NodeType;
```

Description

Read NodeType to determine the type of the node. The type reflects the way the node is used in the XML (see page 26439) document.

See Also

IsTextElement (see page 26434)

197.2.2.3.13 IXMLNode.NodeValue Property

Specifies the value of this node.

Pascal

```
property NodeValue: OleVariant;
```

C++

```
__property OleVariant NodeValue;
```

Description

Use NodeValue to get or set the value of the node. The meaning of a node's value depends on the type of the node, as indicated in the following table:

NodeType (see page 26436)	Value
ntAttribute	The attribute value
ntElement	If the element contains only text, this is that text value. Otherwise, trying to read or set NodeValue causes an exception.
ntText	The text
ntCData	The content of the CDATA section.
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.

Trying to read or set the value of any other node type causes an exception.

See Also

ChildValues (see page 26432)

Text (see page 26439)

197.2.2.3.14 IXMLNode.OwnerDocument Property

Identifies the document in which this node appears.

Pascal

```
property OwnerDocument: IXMLDocument;
```

C++

```
__property IXMLDocument OwnerDocument;
```

Description

Read OwnerDocument to obtain the IXMLDocument (see page 26382) interface of the document to which this node belongs.

See Also

ParentNode (see page 26437)

197.2.2.3.15 IXMLNode.ParentNode Property

Identifies the interface of the node's parent node.

Pascal

```
property ParentNode: IXMLNode;
```

C++

```
__property IXMLNode ParentNode;
```

Description

Read ParentNode to access the node's parent node.

See Also

ChildNodes (see page 26431)

197.2.2.3.16 IXMLNode.Prefix Property

Identifies the namespace prefix of the node's name.

Pascal

```
property Prefix: DOMString;
```

C++

```
__property DOMString Prefix;
```

Description

Prefix is the namespace prefix of an element or attribute node's name. A namespace prefix is a symbolic name for a namespace URI.

For element nodes, the namespace prefix is derived from the name that appears in the XML (see page 26439) tag. For example, the namespace prefix for <xsi:Name First="John" Last="Doe"> is 'xsi'.

For attribute nodes, the namespace prefix is derived from the string that appears to the left of the equals sign in the attribute declaration. For example the element <xsi:Name xsn:First="John" Last="Doe"> results in two attribute nodes, with namespace prefixes of 'xsn' and '', respectively. Note that the attribute never inherits a namespace prefix from the node to which it is attached.

For any other node types, the namespace prefix is nil (Delphi) or NULL (C++).

See Also

LocalName (see page 26434)

NodeType (see page 26436)

NamespaceURI (see page 26435)

197.2.2.3.17 IXMLNode.ReadOnly Property

Specifies whether the node can be modified.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

ReadOnly specifies whether the value, child nodes, or attributes of the node can be changed. When ReadOnly is true, the node can't be modified. Read-only nodes raise an exception when an attempt is made to modify them.

When ReadOnly is false, the node can be modified.

See Also

AttributeNodes (see page 26430)

ChildNodes (see page 26431)

NodeValue (see page 26436)

197.2.2.3.18 IXMLNode.Text Property

Represents the Text of the node.

Pascal

```
property Text: DOMString;
```

C++

```
__property DOMString Text;
```

Description

Text represents the text of the node. For example, if the node corresponds to

```
<Title> Understanding XML </Title>
```

The text of the node is the string 'Understanding XML (see page 26439)'.

Text is intended for use nodes where the IsTextElement (see page 26434) property is true. If IsTextElement (see page 26434) is false, then

If the node has no children, the value of Text is an empty string. Setting Text is this case result in a node where IsTextElement (see page 26434) istrue.

If the node has children (other than a single DOM text node), reading or setting Text causes an exception.

See Also

IsTextElement (see page 26434)

NodeValue (see page 26436)

Working with XML Nodes

197.2.2.3.19 IXMLNode.XML Property

Indicates the XML that corresponds to the subtree rooted at this node.

Pascal

```
property XML: DOMString;
```

C++

```
__property DOMString XML;
```

Description

XML is the XML that corresponds to the document fragment that consists of this node and any child nodes it contains.

Note: XML is only available when the DOMVendor used to parse this node's XML document is MSXML.

See Also

Text (see page 26439)

ChildValues (see page 26432)

DOMVendor (see page 26137)

197.2.3 IXMLNodeCollection Interface

IXMLNodeCollection is the interface for a set of child nodes that are all the same type.

Class Hierarchy

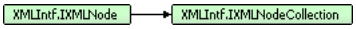

Pascal

```
IXMLNodeCollection = interface(IXMLNode);
```

C++

```
__interface IXMLNodeCollection : public IXMLNode;
```

File

XMLIntf

Description

Use IXMLNodeCollection to work with repeating elements in an XML (see page 26439) document. The XML (see page 26439) Data Binding wizard generates descendants of IXMLNodeCollection interface to represent sets of child nodes that are all of a particular type.

IXMLNodeCollection is a descendant of IXMLNode (see page 26411). As such, it provides generic support for working with a node in an XML (see page 26439) document. In addition, IXMLNodeCollection introduces properties and methods that manage a set of child nodes that are all the same type of node (that all have the same tag names or that are defined as the same type in an XML (see page 26439) schema).

Note: C++ method declarations that use IXMLNodeCollection use the _di_IXMLNodeCollection type instead. This type is a DelphiInterface wrapper around the IXMLNodeCollection interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IXMLNodeCollection >
_di_IXMLNodeCollection;
```

See Also

TXMLDocument (see page 26097)

IXMLNode (see page 26411)

TXMLNodeCollection (see page 26196)

197.2.3.1 IXMLNodeCollection Members

The following tables list the members exposed by IXMLNodeCollection.

Public Methods

	Name	Description
	AddChild (see page 26414)	This is the overview for the AddChild method overload.
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	CloneNode (see page 26415)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 26416)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.

	FindNamespaceDecl (see page 26416)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 26416)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 26417)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 26417)	Returns the value of the AttributeNodes (see page 26430) property.
	GetAttributeNS (see page 26418)	Returns the value of a specified attribute.
	GetChildNodes (see page 26418)	Returns the value of the ChildNodes (see page 26431) property.
	GetChildValue (see page 26418)	Returns the value of a specified child node.
	GetCollection (see page 26419)	Returns the value of the Collection (see page 26432) property.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetDOMNode (see page 26419)	Returns the value of the DOMNode (see page 26433) property.
	GetHasChildNodes (see page 26420)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 26420)	Indicates whether the node has a single text value.
	GetLocalName (see page 26420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 26421)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 26421)	Returns the name of the node.
	GetNodeType (see page 26421)	Returns the type of the node.
	GetNodeValue (see page 26422)	Returns the value of the node.
	GetOwnerDocument (see page 26422)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 26422)	Returns the interface for the node's parent node.
	GetPrefix (see page 26422)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 26423)	Returns the value of the ReadOnly (see page 26438) property.
	GetText (see page 26423)	Returns the text value of the node.
	GetXML (see page 26423)	Returns the value of the XML (see page 26439) property.
	HasAttribute (see page 26424)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 26424)	Returns the next child of this node's parent.
	Normalize (see page 26425)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 26425)	Returns the previous child of this node's parent.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.
	Resync (see page 26426)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 26426)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 26426)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 26427)	Sets the value of a specified child node.
	SetNodeValue (see page 26427)	Sets the value of this node.
	SetReadOnly (see page 26428)	Specifies whether the node can be modified.
	SetText (see page 26428)	Sets the text value of the node.
	TransformNode (see page 26428)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 26430)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 26430)	Represents the values of this node's attributes.
R	ChildNodes (see page 26431)	Represents the child nodes of the node.
	ChildValues (see page 26432)	Represents the values of this node's child nodes.
R	Collection (see page 26432)	Indicates the collection in which this node appears.
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	DOMNode (see page 26433)	Provides access to the IDOMNode interface for this node.
R	HasChildNodes (see page 26433)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 26434)	Indicates whether the node has a single text value.
R	LocalName (see page 26434)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 26435)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 26435)	Indicates the node name.

R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 26436)	Indicates the type of the node.
	NodeValue (see page 26436)	Specifies the value of this node.
R	OwnerDocument (see page 26437)	Identifies the document in which this node appears.
R	ParentNode (see page 26437)	Identifies the interface of the node's parent node.
R	Prefix (see page 26438)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 26438)	Specifies whether the node can be modified.
	Text (see page 26439)	Represents the Text of the node.
R	XML (see page 26439)	Indicates the XML that corresponds to the subtree rooted at this node.

197.2.3.2 IXMLNodeCollection Methods

The methods of the IXMLNodeCollection class are listed here.

Public Methods

	Name	Description
	Clear (see page 26442)	Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.
	Delete (see page 26442)	Removes one of the repeating child nodes from this node.
	GetCount (see page 26443)	Returns the value of the Count (see page 26444) property.
	GetNode (see page 26443)	Returns the interface for one of the repeating child nodes.
	Remove (see page 26443)	Removes one of the repeating child nodes from this node.

197.2.3.2.1 IXMLNodeCollection.Clear Method

Removes all child nodes in the Nodes (see page 26445) property from the ChildList and Nodes (see page 26445) properties.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Clear deletes every child node from the collection (all the nodes in the Nodes (see page 26445) property), along with the corresponding nodes in the ChildNodes (see page 26431) property.

See Also

Nodes (see page 26445)

Delete (see page 26442)

Remove (see page 26443)

ChildNodes (see page 26431)

197.2.3.2.2 IXMLNodeCollection.Delete Method

Removes one of the repeating child nodes from this node.

Pascal

```
procedure Delete(Index: Integer);
```

C++

```
__fastcall Delete(int Index);
```

Description

Delete removes a specified child node from this node.

Index identifies the child to remove. It is the index of the child in the Nodes (see page 26445) property list of repeating child nodes. The index 0 represents the first repetition, 1 represents the second repetition, and so on.

See Also

Remove (see page 26443)

AddChild (see page 26414)

Nodes (see page 26445)

Clear (see page 26442)

197.2.3.2.3 IXMLNodeCollection.GetCount Method

Returns the value of the Count (see page 26444) property.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the read implementation of the Count (see page 26444) property.

See Also

Count (see page 26444)

197.2.3.2.4 IXMLNodeCollection.GetNode Method

Returns the interface for one of the repeating child nodes.

Pascal

```
function GetNode(Index: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode GetNode(int Index);
```

Description

GetNode is the read implementation of the Nodes (see page 26445) property.

Index identifies which repetition of the child node is desired, where 0 is the first repetition, 1 is the second repetition, and so on.

See Also

Nodes (see page 26445)

197.2.3.2.5 IXMLNodeCollection.Remove Method

Removes one of the repeating child nodes from this node.

Pascal

```
function Remove(const Node: IXMLNode): Integer;
```

C++

```
__fastcall int Remove(const IXMLNode Node);
```

Description

Remove removes a specified child node from the collection of repeating child nodes. It removes the node from both the collection, and from the ChildNodes (see page 26431) property, which lists all children of this node.

Node is the interface of the child to remove. It must represent on of the repeating child nodes.

To remove a child that is not one of the repeating child nodes, use the ChildNodes (see page 26431) property instead.

See Also

Delete (see page 26442)

ChildNodes (see page 26431)

Nodes (see page 26445)

197.2.3.3 IXMLNodeCollection Properties

The properties of the IXMLNodeCollection class are listed here.

Public Properties

	Name	Description
R	Count (see page 26444)	Indicates the number of repeating child nodes.
R	Nodes (see page 26445)	Lists the interfaces of the repeating child nodes.

197.2.3.3.1 IXMLNodeCollection.Count Property

Indicates the number of repeating child nodes.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read Count to obtain the number of repeating children of this node. Use Count to obtain an upper limit when addressing child nodes by index (for example, when iterating over all nodes in the Nodes (see page 26445) property list).

Count is changed when you add child nodes using the AddChild (see page 26414) method or delete them using the Delete (see page 26442) or Remove (see page 26443) method. You can access the individual repeating child nodes using the Nodes (see page 26445) property.

See Also

Nodes (see page 26445)

AddChild (see page 26414)

Remove (see page 26443)

Delete (see page 26442)

197.2.3.3.2 IXMLNodeCollection.Nodes Property

Lists the interfaces of the repeating child nodes.

Pascal

```
property Nodes [Index: Integer]: IXMLNode;
```

C++

```
__property IXMLNode Nodes[int Index];
```

Description

Use Nodes to access nodes from the set of repeating child nodes. The value of Nodes is the IXMLNode (see page 26411) interface that represents a specified child node.

Index identifies which repetition of the child node is desired, where 0 is the first repetition, 1 is the second repetition, and so on.

Note: The XML (see page 26439) Data Binding wizard generates a nearly identical property in IXMLNodeCollection (see page 26440) descendants called Items. Items differs from Nodes in that the value is an IXMLNode (see page 26411) descendant specific to the type of the repeating element, rather than the generic IXMLNode (see page 26411) interface.

Note: In Delphi, ChildNodes (see page 26431) is the default property for IXMLNodeCollection (see page 26440). That means that you can omit the property name when using this property. In IXMLNodeCollection (see page 26440) descendants that are generated by the XML (see page 26439) Data Binding wizard, the Items property is the default property instead.

Note: As a descendant of IXMLNode (see page 26411), IXMLNodeCollection (see page 26440) also inherits the ChildNodes (see page 26431) property, which lists all the child nodes for this node. Nodes differs from ChildNodes (see page 26431) in that it contains only those child nodes that are in the repeating collection.

See Also

Count (see page 26444)

ChildNodes (see page 26431)

197.2.4 IXMLNodeList Interface

IXMLNodeList represents a set of nodes in an XML document.

Class Hierarchy

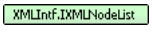

Pascal

```
IXMLNodeList = interface;
```

C++

```
__interface IXMLNodeList;
```

File

XMLIntf

Description

Use IXMLNodeList to work with a set of related nodes in an XML document. IXMLNodeList can be used to add or delete nodes from the set, or to access specific nodes.

Note: C++ method declarations that use IXMLNodeList use the _di_IXMLNodeList type instead. This type is a DelphiInterface wrapper around the IXMLNodeList interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IXMLNodeList >  _di_IXMLNodeList;
```

See Also

TXMLNodeList (see page 26211)

IXMLNode (see page 26411)

IXMLNode (see page 26411)

TXMLDocument (see page 26097)

197.2.4.1 IXMLNodeList Members

The following tables list the members exposed by IXMLNodeList.

Public Methods

	Name	Description
	Add (see page 26447)	Adds a new node to the end of the list.
	BeginUpdate (see page 26447)	Increments the value of UpdateCount (see page 26457).
	Clear (see page 26448)	Removes all nodes from the list.
	Delete (see page 26448)	This is the overview for the Delete method overload.
	EndUpdate (see page 26449)	Decrements the value of UpdateCount (see page 26457).
	FindNode (see page 26450)	This is the overview for the FindNode method overload.
	FindSibling (see page 26451)	Returns a node that appears a specified amount before or after another node.
	First (see page 26451)	Returns the first node in the list.
	Get (see page 26452)	Returns a specified node in the list.
	GetCount (see page 26452)	Returns the number of nodes in the list.
	GetNode (see page 26452)	Returns a specified node from the list.
	GetUpdateCount (see page 26453)	Returns the value of the UpdateCount (see page 26457) property.
	IndexOf (see page 26453)	This is the overview for the IndexOf method overload.
	Insert (see page 26454)	Inserts a new node into a specified position in the list.
	Last (see page 26454)	Returns the last node in the list.
	Remove (see page 26455)	Removes a specified node from the list.
	ReplaceNode (see page 26455)	Replaces a specified node in the list with another node.

Public Properties

	Name	Description
R	Count (see page 26456)	Indicates the number of nodes in the list.
R	Nodes (see page 26456)	Provides access to the nodes in the list.
R	UpdateCount (see page 26457)	Indicates the number of times an application has called BeginUpdate (see page 26447) without a matching call to EndUpdate (see page 26449).

197.2.4.2 IXMLNodeList Methods

The methods of the IXMLNodeList class are listed here.

Public Methods

	Name	Description
	Add (see page 26447)	Adds a new node to the end of the list.
	BeginUpdate (see page 26447)	Increments the value of UpdateCount (see page 26457).
	Clear (see page 26448)	Removes all nodes from the list.
	Delete (see page 26448)	This is the overview for the Delete method overload.
	EndUpdate (see page 26449)	Decrements the value of UpdateCount (see page 26457).
	FindNode (see page 26450)	This is the overview for the FindNode method overload.
	FindSibling (see page 26451)	Returns a node that appears a specified amount before or after another node.
	First (see page 26451)	Returns the first node in the list.
	Get (see page 26452)	Returns a specified node in the list.
	GetCount (see page 26452)	Returns the number of nodes in the list.
	GetNode (see page 26452)	Returns a specified node from the list.
	GetUpdateCount (see page 26453)	Returns the value of the UpdateCount (see page 26457) property.
	IndexOf (see page 26453)	This is the overview for the IndexOf method overload.
	Insert (see page 26454)	Inserts a new node into a specified position in the list.
	Last (see page 26454)	Returns the last node in the list.
	Remove (see page 26455)	Removes a specified node from the list.
	ReplaceNode (see page 26455)	Replaces a specified node in the list with another node.

197.2.4.2.1 IXMLNodeList.Add Method

Adds a new node to the end of the list.

Pascal

```
function Add(const Node: IXMLNode): Integer;
```

C++

```
__fastcall int Add(const IXMLNode Node);
```

Description

Call Add to add a node to the end of the list. Add returns the index of the node once it is added, where 0 is the index of the first node in the list, 1 is the index of the second node, and so on.

Node is the node to add to the list.

See Also

Delete (see page 26449)

Remove (see page 26455)

IndexOf (see page 26453)

Insert (see page 26454)

Nodes (see page 26456)

197.2.4.2.2 IXMLNodeList.BeginUpdate Method

Increments the value of UpdateCount (see page 26457).

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

Call BeginUpdate before making several changes to the nodes in the list (such as adding or removing several nodes). BeginUpdate increments the UpdateCount (see page 26457) property, to keep track of the fact that the list is in the middle of an update. When the update is complete, call EndUpdate (see page 26449) to decrement the UpdateCount (see page 26457) property.

Every call to BeginUpdate should be matched by a call to EndUpdate (see page 26449), so that the UpdateCount (see page 26457) property accurately reflects whether an update operation is underway.

See Also

UpdateCount (see page 26457)

EndUpdate (see page 26449)

197.2.4.2.3 IXMLNodeList.Clear Method

Removes all nodes from the list.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the list.

Note: Clear does not call the BeginUpdate (see page 26447) and EndUpdate (see page 26449) methods, even though it may result in the deletion of more than one node.

See Also

Delete (see page 26449)

Remove (see page 26455)

Nodes (see page 26456)

197.2.4.2.4 Delete Method

This is the overview for the Delete method overload.

Overload List

	Name	Description
	IXMLNodeList.Delete (DOMString) (see page 26449)	Removes a specified node from the list.
	IXMLNodeList.Delete (DOMString, DOMString) (see page 26449)	Removes a specified node from the list.
	IXMLNodeList.Delete (Integer) (see page 26449)	Removes a specified node from the list.

197.2.4.2.4.1 IXMLNodeList.Delete Method (DOMString)

Removes a specified node from the list.

Pascal

```
function Delete(const Index: Integer): Integer; overload;
function Delete(const Name: DOMString): Integer; overload;
function Delete(const Name: DOMString; const NamespaceURI: DOMString): Integer; overload;
```

C++

```
__fastcall int Delete(const int Index);
__fastcall int Delete(const DOMString Name);
__fastcall int Delete(const DOMString Name, const DOMString NamespaceURI);
```

Description

Delete removes the node specified by the Index or Name parameter.

Name identifies the node to remove from the list. This is the local name of the node to remove.

NamespaceURI identifies the namespace to use when interpreting Name (if necessary).

Index identifies the node to remove by index rather than name. Index ranges from 0 to one less than the value of the Count (see page 26456) property.

Delete returns the index of the node that was removed. If there was no node that matched the value of Index or Name, Delete returns –1.

See Also

Remove (see page 26455)

Clear (see page 26448)

IndexOf (see page 26453)

LocalName (see page 26434)

NamespaceURI (see page 26435)

Working with XML Nodes

197.2.4.2.5 IXMLNodeList.EndUpdate Method

Decrements the value of UpdateCount (see page 26457).

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

Call EndUpdate after making several changes to the nodes in the list (such as adding or removing several nodes). EndUpdate decrements the UpdateCount (see page 26457) property, which was incremented by a previous call to BeginUpdate (see page 26447).

Every call to BeginUpdate (see page 26447) should be matched by a call to EndUpdate, so that the UpdateCount (see page 26457) property accurately reflects whether an update operation is underway.

See Also

UpdateCount (see page 26457)

BeginUpdate (see page 26447)

197.2.4.2.6 FindNode Method

This is the overview for the FindNode method overload.

Overload List

	Name	Description
	IXMLNodeList.FindNode (DOMString) (see page 26450)	Returns a specified node from the list.
	IXMLNodeList.FindNode (DOMString, DOMString) (see page 26450)	Returns a specified node from the list.
	IXMLNodeList.FindNode (TGuid) (see page 26450)	Returns a specified node from the list.

197.2.4.2.6.1 IXMLNodeList.FindNode Method (DOMString)

Returns a specified node from the list.

Pascal

```
function FindNode(NodeName: DOMString): IXMLNode; overload;
function FindNode(NodeName: DOMString; NamespaceURI: DOMString): IXMLNode; overload;
function FindNode(ChildNodeType: TGuid): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode FindNode(DOMString NodeName);
__fastcall IXMLNode FindNode(DOMString NodeName, DOMString NamespaceURI);
__fastcall IXMLNode FindNode(TGuid ChildNodeType);
```

Description

Call FindNode to access a particular node in the list.

NodeName is the node to access. It specifies the LocalName property of the desired node.

NamespaceURI identifies the namespace that qualifies NodeName. It is the same as the NamespaceURI property of the desired node.

ChildNodeType identifies the node to access by its interface type (GUID), rather than by name.

FindNode returns the interface for the node if it is in the list. If NodeName or ChildNodeType does not specify a node in the list, FindNode returns nil (Delphi) or NULL (C++).

See Also

FindSibling (see page 26451)

First (see page 26451)

Last (see page 26454)

IndexOf (see page 26453)

LocalName (see page 26434)

Nodes (see page 26456)

NamespaceURI (see page 26435)

197.2.4.2.7 IXMLNodeList.FindSibling Method

Returns a node that appears a specified amount before or after another node.

Pascal

```
function FindSibling(const Node: IXMLNode; Delta: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode FindSibling(const IXMLNode Node, int Delta);
```

Description

Call FindSibling to access the node whose position has a specified relationship to another node.

Node is a node in the list to use as a reference point.

Delta indicates where the desired node appears, relative to Node. If Delta is positive, FindSibling returns the node that appears Delta positions after Node. If Delta is negative, FindSibling returns a node that appears before Node.

FindSibling returns the node that appears at the position offset by Delta, relative to the position of Node. If Delta specifies a position before the first node or after the last node in the list, FindSibling returns nil (Delphi) or NULL (C++).

See Also

FindNode (see page 26450)

First (see page 26451)

Last (see page 26454)

IndexOf (see page 26453)

197.2.4.2.8 IXMLNodeList.First Method

Returns the first node in the list.

Pascal

```
function First: IXMLNode;
```

C++

```
__fastcall IXMLNode First();
```

Description

Call First to access the first node in the list. If the list is empty, First raises an exception.

See Also

FindNode (see page 26450)

FindSibling (see page 26451)

Last (see page 26454)

Nodes (see page 26456)

197.2.4.2.9 IXMLNodeList.Get Method

Returns a specified node in the list.

Pascal

```
function Get(Index: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode Get(int Index);
```

Description

Call Get to retrieve a node from the list, given its index.

Index specifies the node to fetch, where 0 identifies the first node, 1 identifies the second node, and so on. Index should be less than the value of the Count (see page 26456) property.

See Also

Nodes (see page 26456)

GetNode (see page 26452)

Count (see page 26456)

197.2.4.2.10 IXMLNodeList.GetCount Method

Returns the number of nodes in the list.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the read implementation of the Count (see page 26456) property.

See Also

Count (see page 26456)

197.2.4.2.11 IXMLNodeList.GetNode Method

Returns a specified node from the list.

Pascal

```
function GetNode(const IndexOrName: OleVariant): IXMLNode;
```

C++

```
__fastcall IXMLNode GetNode(const OleVariant IndexOrName);
```

Description

GetNode is the read implementation of the Nodes (see page 26456) property.

IndexOrName identifies the desired node. It can be

The index of the node, where 0 is the index of the first node, 1 is the index of the second node, and so on.

The LocalName property of a node in the list.

If IndexOrName does not identify a node in the list, GetNode tries to create a new node with the name specified by IndexOrName. If it can't create the new node, GetNode raises an exception.

See Also

FindNode (see page 26450)

Nodes (see page 26456)

LocalName (see page 26434)

Get (see page 26452)

197.2.4.2.12 IXMLNodeList.GetUpdateCount Method

Returns the value of the UpdateCount (see page 26457) property.

Pascal

```
function GetUpdateCount: Integer;
```

C++

```
__fastcall int GetUpdateCount();
```

Description

GetUpdateCount is the read implementation of the UpdateCount (see page 26457) property.

See Also

UpdateCount (see page 26457)

197.2.4.2.13 IndexOf Method

This is the overview for the IndexOf method overload.

Overload List

	Name	Description
	IXMLNodeList.IndexOf (DOMString) (see page 26453)	Returns the index of a specified node.
	IXMLNodeList.IndexOf (DOMString, DOMString) (see page 26453)	Returns the index of a specified node.
	IXMLNodeList.IndexOf (IXMLNode) (see page 26453)	Returns the index of a specified node.

197.2.4.2.13.1 IXMLNodeList.IndexOf Method (DOMString)

Returns the index of a specified node.

Pascal

```
function IndexOf(const Node: IXMLNode): Integer; overload;
function IndexOf(const Name: DOMString): Integer; overload;
function IndexOf(const Name: DOMString; const NamespaceURI: DOMString): Integer; overload;
```

C++

```
__fastcall int IndexOf(const IXMLNode Node);
```

```
__fastcall int IndexOf(const DOMString Name);
__fastcall int IndexOf(const DOMString Name, const DOMString NamespaceURI);
```

Description

Call IndexOf to locate a node in the list. You can specify which node to locate using its name or its interface.

Name is the LocalName property of the node to locate.

NamespaceURI identifies the namespace to use when interpreting LocalName (if necessary).

Node is the interface of the node to locate.

IndexOf returns the index of the specified node, where 0 is the index of the first node, 1 is the index of the second node, and so on. If the specified node is not in the list, IndexOf returns -1.

See Also

FindNode (see page 26450)

LocalName (see page 26434)

Nodes (see page 26456)

NamespaceURI (see page 26435)

197.2.4.2.14 IXMLNodeList.Insert Method

Inserts a new node into a specified position in the list.

Pascal

```
procedure Insert(Index: Integer; const Node: IXMLNode);
```

C++

```
__fastcall Insert(int Index, const IXMLNode Node);
```

Description

Call Insert to add a node at the position specified by Index.

Index specifies where to insert the node, where 0 is the first position, 1 is second position, and so on. If Index does not specify a valid index, Insert raises an exception.

Node is the node to add to the list.

See Also

Delete (see page 26449)

Remove (see page 26455)

IndexOf (see page 26453)

Add (see page 26447)

Nodes (see page 26456)

197.2.4.2.15 IXMLNodeList.Last Method

Returns the last node in the list.

Pascal

```
function Last: IXMLNode;
```

C++

```
__fastcall IXMLNode Last();
```

Description

Call Last to access the last node in the list. If the list is empty, Last raises an exception.

See Also

FindNode (see page 26450)

FindSibling (see page 26451)

First (see page 26451)

Nodes (see page 26456)

197.2.4.2.16 IXMLNodeList.Remove Method

Removes a specified node from the list.

Pascal

```
function Remove(const Node: IXMLNode): Integer;
```

C++

```
__fastcall int Remove(const IXMLNode Node);
```

Description

Remove removes the specified node from the list.

Node is the node to remove from the list.

Remove returns the index of Node before it was removed. If node is not a node in the list, Remove returns -1.

See Also

Delete (see page 26449)

Clear (see page 26448)

197.2.4.2.17 IXMLNodeList.ReplaceNode Method

Replaces a specified node in the list with another node.

Pascal

```
function ReplaceNode(const OldNode: IXMLNode; const NewNode: IXMLNode): IXMLNode;
```

C++

```
__fastcall IXMLNode ReplaceNode(const IXMLNode OldNode, const IXMLNode NewNode);
```

Description

Call ReplaceNode to replace the node specified by OldNode with the node specified by NewNode.

OldNode is the node to replace. If OldNode does not appear in the list, then ReplaceNode adds the new node to the end of the list.

NewNode is the node to add to the list in place of OldNode.

ReplaceNode returns OldNode (even if OldNode did not appear in the list).

See Also

Remove (see page 26455)

Insert (see page 26454)

197.2.4.3 IXMLNodeList Properties

The properties of the IXMLNodeList class are listed here.

Public Properties

	Name	Description
R	Count (see page 26456)	Indicates the number of nodes in the list.
R	Nodes (see page 26456)	Provides access to the nodes in the list.
R	UpdateCount (see page 26457)	Indicates the number of times an application has called BeginUpdate (see page 26447) without a matching call to EndUpdate (see page 26449).

197.2.4.3.1 IXMLNodeList.Count Property

Indicates the number of nodes in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read Count to determine the number of nodes in the list. Count provides an upper limit to the indexes that can be used with the Nodes (see page 26456) property.

To change the value of Count, add nodes using the Add (see page 26447) or Insert (see page 26454) method, or remove them using the Delete (see page 26449) or Remove (see page 26455) method.

See Also

Nodes (see page 26456)

Add (see page 26447)

Delete (see page 26449)

Remove (see page 26455)

Insert (see page 26454)

197.2.4.3.2 IXMLNodeList.Nodes Property

Provides access to the nodes in the list.

Pascal

```
property Nodes [const IndexOrName: OleVariant]: IXMLNode;
```

C++

```
__property IXMLNode Nodes[OleVariant const IndexOrName];
```

Description

Read Nodes to access a specified node in the list.

IndexOrName identifies the desired node. It can be

The index of the node, where 0 is the index of the first node, 1 is the index of the second node, and so on. The Count (see page 26456) property provides an upper bound on the indexes you can specify.

The LocalName property of a node in the list.

If IndexOrName does not identify a node in the list, and if the document that contains this node list's parent includes doNodeAutoCreate in its Options property, then the node list tries to create a new node with the name specified by IndexOrName. If the node list can't create the new node, it raises an exception.

See Also

Count (see page 26456)

FindNode (see page 26450)

LocalName (see page 26434)

Options (see page 26141)

197.2.4.3.3 IXMLNodeList.UpdateCount Property

Indicates the number of times an application has called BeginUpdate (see page 26447) without a matching call to EndUpdate (see page 26449).

Pascal

```
property UpdateCount: Integer;
```

C++

```
__property int UpdateCount;
```

Description

UpdateCount keeps track of whether all update operations have been completed. When an application starts making changes to the list, it calls BeginUpdate (see page 26447), which increments UpdateCount. When those changes are complete, it calls EndUpdate (see page 26449), which decrements UpdateCount. Thus, UpdateCount allows calls to BeginUpdate (see page 26447) and EndUpdate (see page 26449) to be nested.

Applications can avoid responding to updates in a piecemeal fashion by checking UpdateCount and only responding when it is 0.

See Also

BeginUpdate (see page 26447)

EndUpdate (see page 26449)

197.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TNodeType (see page 26458)	TNodeType identifies the type of a node in an XML document.
	TParseOption (see page 26460)	TParseOption and TParseOptions (see page 26463) represent implementation-specific options that are followed when parsing the XML document.
	TXMLDocOption (see page 26460)	TXMLDocOption and TXMLDocOptions (see page 26463) indicate types of information that an XML document automatically supplies.
	TXMLEncodingType (see page 26461)	TXMLEncodingType indicates the character set used to encode characters in an XML document.

197.3.1 XMLIntf.TNodeType Enumeration

TNodeType identifies the type of a node in an XML document.

Pascal

```
TNodeType = (
  ntReserved,
  ntElement,
  ntAttribute,
  ntText,
  ntCData,
  ntEntityRef,
  ntEntity,
  ntProcessingInstr,
  ntComment,
  ntDocument,
  ntDocType,
  ntDocFragment,
  ntNotation
);
```

C++

```
enum TNodeType {
  ntReserved,
  ntElement,
  ntAttribute,
  ntText,
  ntCData,
  ntEntityRef,
  ntEntity,
  ntProcessingInstr,
  ntComment,
  ntDocument,
  ntDocType,
  ntDocFragment,
  ntNotation
};
```

File

XMLIntf

Description

TNodeType identifies the type of a node in an XML document. The following table lists the possible values:

Value	Meaning
ntReserved	Not used
ntElement	The node represents an element. Element nodes represent simple tags that have child nodes. (Note that sometimes these child nodes do not appear when using IXMLNode (see page 26411). For example child nodes of type ntText are typically hidden by IXMLNode (see page 26411) and appear only as the value of the Text property). The child nodes of an element node can have the following node types: ntElement, ntText, ntCData, ntEntityRef, ntProcessingInstr, and ntComment. Element nodes can also have attributes (ntAttribute). Element nodes can be the child of a node of type ntDocument, ntDocFragment, ntEntityRef, and ntElement.
ntAttribute	The node represents an attribute of an element. It is not the child of another node, but its value can be accessed using the Attribute property of the element node's IXMLNode (see page 26411) interface. An attribute node can have child nodes of type ntText and ntEntityRef.
ntText	The node represents the text content of a tag. A text node cannot have any child nodes, but can appear as the child node of a node of type ntAttribute, ntDocFragment, ntElement, or ntEntityRef.
ntCData	The node represents a CDATA section in the XML source. CDATA sections identify blocks of text that would otherwise be interpreted as markup. An ntCData node can't have any child nodes. They can appear as the child of an ntDocFragment, ntEntityRef, or ntElement node.
ntEntityRef	The node represents a reference to an entity in the XML document. This can be any type of entity, including character entity references. The children of an entity reference node can be of the following types: ntElement, ntProcessingInstr, ntComment, ntText, ntCData, and ntEntityRef. The entity reference node can appear as the child of an ntAttribute, ntDocFragment, ntElement, or ntEntityRef node.
ntEntity	The node represents an expanded entity. Entity nodes can have child nodes that represent the expanded entity (for example, ntText and ntEntityRef nodes). Entity nodes only appear as the child of an ntDocType node.
ntProcessingInstr	The node represents a processing instruction (PI) from the XML document. A PI node cannot have any child nodes, but can appear as the child node of a node of type ntDocument, ntDocFragment, ntElement, or ntEntityRef.
ntComment	The node represents a comment in the XML document. Comment nodes do not have child nodes. They appear as the child of an ntDocument, ntDocFragment, ntElement, or ntEntityRef node.
ntDocument	The node represents a document object, which is the root of the entire XML document. Document nodes have a single ntElement node as a child (the DocumentElement). In addition, they can have child nodes of type ntProcessingInstr, ntComment, and ntDocType. Because the document is the root of the XML document, it never appears as a child node.
ntDocType	The node represents the document type declaration, indicated by the <!DOCTYPE > tag. The document type node can child nodes of type ntNotation and ntEntity. It always appears as the child of the document node.
ntDocFragment	The node represents a document fragment. A document fragment node associates a node or subtree with a document without actually being contained in the document. Document fragment nodes can have child nodes of type ntElement, ntProcessingInstr, ntComment, ntText, ntCData, and ntEntityRef. It never appears as the child of another node.
ntNotation	A node represents a notation in the document type declaration. It always appears as the child of an ntDocType node and never has any child nodes.

197.3.2 XMLIntf.TParseOption Enumeration

TParseOption and TParseOptions (see page 26463) represent implementation-specific options that are followed when parsing the XML document.

Pascal

```
TParseOption = (
  poResolveExternals,
  poValidateOnParse,
  poPreserveWhiteSpace,
  poAsyncLoad
);
```

C++

```
enum TParseOption {
  poResolveExternals,
  poValidateOnParse,
  poPreserveWhiteSpace,
  poAsyncLoad
};
```

File

XMLIntf

Description

TParseOptions (see page 26463) is a set of TParseOption values. Each TParseOption value provides information about how a DOM parser should parse an XML document. Different DOM implementations support different options.

The following table lists the possible values:

OptionDOM VendorMeaning

Option	DOM Vendor	Meaning
poResolveExternals	MSXML	External definitions (resolvable namespaces, DTD external subsets, and external entity references) are resolved at parse time.
poValidateOnParse	MSXML	The DOM parser validates the XML document against its schema information as well as verifying that it is well-formed XML.
poPreserveWhiteSpace	MSXML	White space in the text of the XML document is not stripped off.
poAsyncLoad	MSXML	The DOM parser works asynchronously. This can speed up performance, especially when parsing a large XML document, but can also lead to exceptions if the application tries to access nodes in the document before it is completely parsed.

197.3.3 XMLIntf.TXMLDocOption Enumeration

TXMLDocOption and TXMLDocOptions (see page 26463) indicate types of information that an XML document automatically supplies.

Pascal

```
TXMLDocOption = (
```

```
    doNodeAutoCreate,
    doNodeAutoIndent,
    doAttrNull,
    doAutoPrefix,
    doNamespaceDecl,
    doAutoSave
  );
```

C++

```
  enum TXMLDocOption {
    doNodeAutoCreate,
    doNodeAutoIndent,
    doAttrNull,
    doAutoPrefix,
    doNamespaceDecl,
    doAutoSave
  };
```

File

XMLIntf

Description

TXMLDocOptions (see page 26463) is a set of TXMLDocOption values. The following table lists the TXMLDocOption values:

Value	Meaning
doNodeAutoCreate	If the application tries to read a node by name, using the Nodes property of an IXMLNodeList (see page 26445) interface, and that node does not exist, then the application creates a new node using the specified name.
doNodeAutoIndent	When formatting the XML text from the parsed set of nodes, child nodes are automatically indented from their parent nodes.
doAttrNull	When reading the value of an attribute that does not exist, the value is given as a Null Variant (as opposed to a value of an empty string).
doAutoPrefix	When generating the name of a new node, the value is automatically given a namespace prefix if a namespace URI is supplied.
doNamespaceDecl	When the AddChild method must create a new namespace for the node, it also adds a namespace attribute to declare that new namespace. If doNamespaceDecl is not included, AddChild creates the new namespace but does not add an attribute to declare it.
doAutoSave	When the XML document closes, any changes are automatically saved back to the XML document file or to the XML property.

197.3.4 XMLIntf.TXMLEncodingType Enumeration

TXMLEncodingType indicates the character set used to encode characters in an XML document.

Pascal

```
  TXMLEncodingType = (
    xetUnknown,
    xetUCS_4BE,
    xetUCS_4LE,
    xetUCS_4Order2134,
    xetUCS_4Order3412,
    xetUTF_16BE,
    xetUTF_16LE,
    xetUTF_8,
```

```
    xetUCS_4Like,
    xetUTF_16BELike,
    xetUTF_16LELike,
    xetUTF_8Like,
    xetEBCDICLike
  );
```

C++

```
  enum TXMLEncodingType {
    xetUnknown,
    xetUCS_4BE,
    xetUCS_4LE,
    xetUCS_4Order2134,
    xetUCS_4Order3412,
    xetUTF_16BE,
    xetUTF_16LE,
    xetUTF_8,
    xetUCS_4Like,
    xetUTF_16BELike,
    xetUTF_16LELike,
    xetUTF_8Like,
    xetEBCDICLike
  };
```

File

XMLIntf

Description

The following table lists the possible TXMLEncodingType values:

Value	Character encoding
xetUnknown	The character set is unknown.
xetUCS_4BE	UCS-4, big-endian (1234).
xetUCS_4LE	UCS-4, little-endian (4321).
xetUCS_4Order2134	UCS-4, unusual octet order (2134)
xetUCS_4Order3412	UCS-4, unusual octet order (3412)
xetUTF_16BE	UTF-16, big-endian order
xetUTF_16LE	UTF-16, little-endian order
xetUTF_8	UTF-8
xetUCS_4Like	Any UCS-4 encoding
xetUTF_16BELike	Any UTF-16, big-endian encoding, may be without byte order mark
xetUTF_16LELike	Any UTF-16, little-endian encoding, my be without byte order mark
xetUTF_8Like	UTF-8, ISO 646, ASCII, or other 7- or 8-bit encoding, including multi-byte character sets such as Shift-JIS.
xetEBCDICLike	EBCDIC

197.4 Types

The following table lists types in this documentation.

Types

Name	Description
TParseOptions (see page 26463)	This is type XMLIntf.TParseOptions.
TXMLDocOptions (see page 26463)	This is type XMLIntf.TXMLDocOptions.

197.4.1 XMLIntf.TParseOptions Type

Pascal

```
TParseOptions = set of TParseOption;
```

C++

```
TParseOption TParseOptions;
```

File

XMLIntf

Description

This is type XMLIntf.TParseOptions.

197.4.2 XMLIntf.TXMLDocOptions Type

Pascal

```
TXMLDocOptions = set of TXMLDocOption;
```

C++

```
TXMLDocOption TXMLDocOptions;
```

File

XMLIntf

Description

This is type XMLIntf.TXMLDocOptions.

197.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
NodeTypeNames (see page 26464)	This is constant XMLIntf.NodeTypeNames.

197.5.1 XMLIntf.NodeTypeNames Constant

Pascal

```
NodeTypeNames: array[TNodeType] of string = ('', 'Element','Attribute','Text','CDATASection',
'EntityRef','Entity', 'ProcessingInstr', 'Comment','Document','DocumentType',
'DocumentFragment','Notation');
```

C++

```
array[TNodeType] of string NodeTypeNames = ('', 'Element','Attribute','Text','CDATASection',
'EntityRef','Entity', 'ProcessingInstr', 'Comment','Document','DocumentType',
'DocumentFragment','Notation');
```

File

XMLIntf

Description

This is constant XMLIntf.NodeTypeNames.

198 Xmlxform Namespace

This is namespace Xmlxform.

Topics

Name	Description
Classes (see page 26466)	The following table lists classes in this documentation.
Types (see page 26509)	The following table lists types in this documentation.

Classes

	Name	Description
	TranslateException (see page 26466)	TranslateException is the exception class for errors that occur when converting an XML document to a data packet or a data packet to an XML document.
	TXMLTransform (see page 26467)	TXMLTransform converts an XML document into a data packet or an XML data packet into an XML document with another structure.
	TXMLTransformClient (see page 26482)	TXMLTransformClient acts as an adapter between an XML document and a provider component.
	TXMLTransformProvider (see page 26492)	TXMLTransformProvider provides data from and resolves updates to an XML document.

Types

Name	Description
TRowEvent (see page 26509)	This is type Xmlxform.TRowEvent.
TTranslateEvent (see page 26509)	This is type Xmlxform.TTranslateEvent.

198.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TranslateException (see page 26466)	TranslateException is the exception class for errors that occur when converting an XML document to a data packet or a data packet to an XML document.
	TXMLTransform (see page 26467)	TXMLTransform converts an XML document into a data packet or an XML data packet into an XML document with another structure.
	TXMLTransformClient (see page 26482)	TXMLTransformClient acts as an adapter between an XML document and a provider component.
	TXMLTransformProvider (see page 26492)	TXMLTransformProvider provides data from and resolves updates to an XML document.

198.1.1 TranslateException Class

TranslateException is the exception class for errors that occur when converting an XML document to a data packet or a data packet to an XML document.

Class Hierarchy

Pascal

```
TranslateException = class(Exception);
```

C++

```
class TranslateException : public Exception;
```

File

Xmlxform

Description

TranslateException is raised by the transform components (TXMLTransform (see page 26467), TXMLTransformClient (see page 26482), and TXMLTransformProvider (see page 26492)) when they can't find a file that is needed to perform a transformation. The missing file can be the XML document that is to be translated into a data packet, or a transformation file that contains instructions on how to map between a data packet and an XML document.

See Also

TXMLTransform (see page 26467)

TXMLTransformClient (see page 26482)

TXMLTransformProvider (see page 26492)

198.1.2 TXMLTransform Class

TXMLTransform converts an XML document into a data packet or an XML data packet into an XML document with another structure.

Class Hierarchy

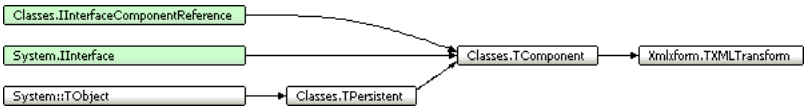

Pascal

```
TXMLTransform = class(TComponent);
```

C++

```
class TXMLTransform : public TComponent;
```

File

Xmlxform

Description

Use TXMLTransform to convert an XML document into a data packet or an XML data packet into an XML document that is not a data packet. You can use a TXMLTransform component by itself, or specify the transformation to be performed on a TXMLTransform component that is part of another component such as TXMLTransformClient (see page 26482) or TXMLTransformProvider (see page 26492).

TXMLTransform uses a special XML document called a transformation, which indicates how to perform the conversion. This transformation is specified by the TransformationFile (see page 26478) or TransformationDocument (see page 26478) property. You can generate a transformation file using the xmlmapper.exe utility.

There are three ways to specify the source document or data packet that TXMLTransform converts:

Use the SourceXmlFile (see page 26477) property if the source document is a file on disk.

Use the SourceXml (see page 26476) property if the source document is an in-memory string of XML.

Use the SourceXmlDocument (see page 26476) property if the source document is available through a DOM interface (IDOMDocument).

TXMLTransform performs its transformation when you read the Data (see page 26473) property or call the TransformXML (see page 26472) method. Once the source XML has been transformed, you can access the DOM interface for the resulting XML document or data packet using the ResultDocument (see page 26475) property, or read the Data (see page 26473) property to directly access the string of XML.

See Also

TXMLTransformClient (see page 26482)

TXMLTransformProvider (see page 26492)

Converting XML Documents into Data Packets

198.1.2.1 TXMLTransform Members

The following tables list the members exposed by TXMLTransform.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	DoTransform (see page 26471)	This is DoTransform, a member of class TXMLTransform.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetData (see page 26471)	This is GetData, a member of class TXMLTransform.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetResultString (see page 26471)	This is GetResultString, a member of class TXMLTransform.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Transform (see page 26472)	This is Transform, a member of class TXMLTransform.
	Translate (see page 26473)	This is Translate, a member of class TXMLTransform.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.

	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	TransformXML (see page 26472)	Converts an XML document using the specifications in a transformation file.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Data (see page 26473)	Represents the translated string of XML.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	EmptyDestinationDocument (see page 26474)	Specifies the DOM interface for an XML document that defines the structure of the transformation result.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
R	ResultDocument (see page 26475)	Specifies the DOM interface for the results of the transformation.
R	ResultString (see page 26475)	Specifies the results of the transformation.
	SourceXml (see page 26476)	Specifies a string of XML to be translated.
	SourceXmlDocument (see page 26476)	Specifies the DOM interface for the XML document to be converted.
	SourceXmlFile (see page 26477)	Specifies an XML document or XML data packet file to be translated.
	TransformationDocument (see page 26478)	Specifies the transformation that indicates how to convert the source XML document.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterEachRow (see page 26479)	Occurs after TXMLTransform (see page 26467) has finished transforming a record in the XML data packet.
	AfterEachRowSet (see page 26480)	Occurs after TXMLTransform (see page 26467) has finished transforming the entire XML data packet or a nested detail set.
	BeforeEachRow (see page 26480)	Occurs immediately before TXMLTransform (see page 26467) starts transforming a record in the XML data packet.
	BeforeEachRowSet (see page 26481)	Occurs immediately before TXMLTransform (see page 26467) starts transforming the entire XML data packet or a nested detail set.
	OnTranslate (see page 26481)	Occurs when the TXMLTransform (see page 26467) component needs to translate a user-defined node.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
	TransformationFile (see page 26478)	Specifies the transformation file that indicates how to convert the source XML document.

198.1.2.2 TXMLTransform Methods

The methods of the TXMLTransform class are listed here.

Protected Methods

	Name	Description
	DoTransform (see page 26471)	This is DoTransform, a member of class TXMLTransform.
	GetData (see page 26471)	This is GetData, a member of class TXMLTransform.
	GetResultString (see page 26471)	This is GetResultString, a member of class TXMLTransform.
	Transform (see page 26472)	This is Transform, a member of class TXMLTransform.
	Translate (see page 26473)	This is Translate, a member of class TXMLTransform.

Public Methods

	Name	Description
	TransformXML (see page 26472)	Converts an XML document using the specifications in a transformation file.

198.1.2.2.1 TXMLTransform.DoTransform Method

Pascal

```
function DoTransform(const XMLSrc: IDOMDocument; const XMLExtract: IDOMDocument; const XMLOut:
IDOMDocument): string;
```

C++

```
__fastcall AnsiString DoTransform(const IDOMDocument XMLSrc, const IDOMDocument XMLExtract,
const IDOMDocument XMLOut);
```

Description

This is DoTransform, a member of class TXMLTransform.

198.1.2.2.2 TXMLTransform.GetData Method

Pascal

```
function GetData: string;
```

C++

```
__fastcall AnsiString GetData();
```

Description

This is GetData, a member of class TXMLTransform.

198.1.2.2.3 TXMLTransform.GetResultString Method

Pascal

```
function GetResultString: string;
```

C++

```
__fastcall AnsiString GetResultString();
```

Description

This is GetResultString, a member of class TXMLTransform.

198.1.2.2.4 TXMLTransform.Transform Method

Pascal

```
procedure Transform(TransNode: IDOMNode; SrcNode: IDOMNode; DestNode: IDOMNode; Count:
Integer; InFromList: TStringList; InDestList: TStringList; InIdStrList: TStringList;
InValueList: TStringList; InOptionalList: TStringList; InDateFormatList: TStringList;
InDateFormatTypeList: TStringList; InMapValuesList: TStringList);
```

C++

```
__fastcall Transform(IDOMNode TransNode, IDOMNode SrcNode, IDOMNode DestNode, int Count,
TStringList InFromList, TStringList InDestList, TStringList InIdStrList, TStringList
InValueList, TStringList InOptionalList, TStringList InDateFormatList, TStringList
InDateFormatTypeList, TStringList InMapValuesList);
```

Description

This is Transform, a member of class TXMLTransform.

198.1.2.2.5 TXMLTransform.TransformXML Method

Converts an XML document using the specifications in a transformation file.

Pascal

```
function TransformXML(const SourceXml: string; const ATransformationFile: string = ''): string;
```

C++

```
__fastcall AnsiString TransformXML(const AnsiString SourceXml, const AnsiString
ATransformationFile = '');
```

Description

Call TransformXML to convert an XML document when the source XML is an in-memory string.

SourceXml (see page 26476) is a string of XML to be transformed. It can be any type of XML document, including an XML data packet. It must, however, conform to the schema expected by the transformation document used to perform the transformation. If the transformation converts a specific type of XML document into a data packet, SourceXml (see page 26476) must conform to the expected schema. If the transformation converts a data packet into an XML document, SourceXml (see page 26476) must be an XML data packet with the expected metadata.

ATransformationFile is the transformation file to use when performing the translation. A transformation file is a special type of XML file with the .xtr extension. Each transformation file is specific to a particular XML schema and data packet format and represents a one-way transformation. Use the xmlmapper utility to create transformation files.

If the ATransformationFile parameter is omitted, TransformXML uses the file specified by the TransformationFile (see page 26478) property, or the document specified by the TransformationDocument (see page 26478) property.

TransformXML returns the results of the conversion as a string of XML.

See Also

Data (see page 26473)

TransformationFile (see page 26478)

TransformationDocument (see page 26478)

198.1.2.2.6 TXMLTransform.Translate Method

Pascal

```
procedure Translate(const Id: string; const SrcNode: IDOMNode; var SrcValue: string; const
DestNode: IDOMNode); dynamic;
```

C++

```
__fastcall Translate(const AnsiString Id, const IDOMNode SrcNode, AnsiString SrcValue, const
IDOMNode DestNode);
```

Description

This is Translate, a member of class TXMLTransform.

198.1.2.3 TXMLTransform Properties

The properties of the TXMLTransform class are listed here.

Public Properties

	Name	Description
R	Data (see page 26473)	Represents the translated string of XML.
	EmptyDestinationDocument (see page 26474)	Specifies the DOM interface for an XML document that defines the structure of the transformation result.
R	ResultDocument (see page 26475)	Specifies the DOM interface for the results of the transformation.
R	ResultString (see page 26475)	Specifies the results of the transformation.
	SourceXml (see page 26476)	Specifies a string of XML to be translated.
	SourceXmlDocument (see page 26476)	Specifies the DOM interface for the XML document to be converted.
	SourceXmlFile (see page 26477)	Specifies an XML document or XML data packet file to be translated.
	TransformationDocument (see page 26478)	Specifies the transformation that indicates how to convert the source XML document.

Published Properties

	Name	Description
	TransformationFile (see page 26478)	Specifies the transformation file that indicates how to convert the source XML document.

198.1.2.3.1 TXMLTransform.Data Property

Represents the translated string of XML.

Pascal

```
property Data: string;
```

C++

```
__property AnsiString Data;
```

Description

Read Data to obtain the results of the transformation. Reading the Data property causes TXMLTransform (see page 26467) to translate its source XML document or data packet into the destination format, which becomes the value of the Data property. The resulting XML string is not cached in memory: reading Data a second time causes TXMLTransform (see page 26467) to perform its transformation a second time.

If the source XML is an XML document and the transformation indicates how to convert it into a data packet, Data is the contents

of a data packet in XML format.

If the source XML is an XML data packet and the transformation indicates how to convert it into another type of XML document, Data is the contents of the resulting document.

There are three ways to specify the source document or data packet that is converted to produce Data:

- Use the SourceXmlFile (see page 26477) property if the source document is a file on disk.
- Use the SourceXml (see page 26476) property if the source document is an in-memory string of XML.
- Use the SourceXmlDocument (see page 26476) property if the source document is available through a DOM interface (IDOMDocument).
- If SourceXMLFile is set to an invalid file name or SourceXMLFile is an empty string and SourceXml (see page 26476) is not a valid string of XML, or SourceXMLFile and SourceXML are both empty strings and SourceXmlDocument (see page 26476) is not set, TXMLTransform (see page 26467) raises a TranslateException (see page 26466) when you try to read Data.
- There are two ways to specify the transformation that converts the source XML into Data:
- Use the TransformationFile (see page 26478) property if the transformation is stored in a file on disk.
- Use the TransformationDocument (see page 26478) method if the transformation is available through a DOM interface (IDOMDocument).
- If TransformationFile (see page 26478) is set to an invalid file name or TransformationFile (see page 26478) is an empty string and TransformationDocument (see page 26478) is not set, TXMLTransform (see page 26467) raises a TranslateException (see page 26466) when you try to read Data.
- You can augment the transformation by using the EmptyDestinationDocument (see page 26474) property to supply a skeleton document that gives the form for the resulting XML document.
- When you read Data, for every user-defined node in the source XML document, TXMLTransform (see page 26467) generates an OnTranslate (see page 26481) event to convert the node.

Tip: When converting an XML document into a data packet, the value of Data can be assigned to a client dataset's XMLData property.

See Also

SourceXmlFile (see page 26477)

TransformationFile (see page 26478)

TranslateException (see page 26466)

OnTranslate (see page 26481)

XMLData (see page 8496)

ResultDocument (see page 26475)

EmptyDestinationDocument (see page 26474)

TransformationDocument (see page 26478)

SourceXml (see page 26476)

Converting XML Documents into Data Packets

198.1.2.3.2 TXMLTransform.EmptyDestinationDocument Property

Specifies the DOM interface for an XML document that defines the structure of the transformation result.

Pascal

```
property EmptyDestinationDocument: IDOMDocument;
```

C++

```
__property IDOMDocument EmptyDestinationDocument;
```

Description

The transformation defined by TransformationFile (see page 26478) or TransformationDocument (see page 26478) contains a default structure for the results of transforming the source document in addition to the mappings between nodes in the source document and nodes in the destination. You can override this default structure, however, by setting EmptyDestinationDocument. This is the DOM interface for an XML document that has the desired structure, but with no values in the nodes that should be filled in using values in the source document. This is useful, for example, when you want to add additional information that does not come from the source document.

EmptyDestinationDocument is not changed when TXMLTransform (see page 26467) performs its transformation. To obtain a DOM interface for the resulting document, use the ResultDocument (see page 26475) property.

See Also

ResultDocument (see page 26475)

Data (see page 26473)

TransformXML (see page 26472)

TransformationFile (see page 26478)

TransformationDocument (see page 26478)

198.1.2.3.3 TXMLTransform.ResultDocument Property

Specifies the DOM interface for the results of the transformation.

Pascal

```
property ResultDocument: IDOMDocument;
```

C++

```
__property IDOMDocument ResultDocument;
```

Description

Read ResultDocument to work with the results of the transformation using a DOM interface. ResultDocument is created when you call the TransformXML (see page 26472) method to perform the transformation.

See Also

SourceXmlDocument (see page 26476)

Data (see page 26473)

TransformXML (see page 26472)

EmptyDestinationDocument (see page 26474)

ResultString (see page 26475)

198.1.2.3.4 TXMLTransform.ResultString Property

Specifies the results of the transformation.

Pascal

```
property ResultString: string;
```

C++

```
__property AnsiString ResultString;
```

Description

Read ResultString to obtain the results of the transformation as a string of XML. ResultString is created when you call the TransformXML (see page 26472) method to perform the transformation.

See Also

SourceXmlDocument (see page 26476)

Data (see page 26473)

TransformXML (see page 26472)

EmptyDestinationDocument (see page 26474)

ResultString

198.1.2.3.5 TXMLTransform.SourceXml Property

Specifies a string of XML to be translated.

Pascal

```
property SourceXml: string;
```

C++

```
__property AnsiString SourceXml;
```

Description

Specify the XML to be translated as the value of SourceXml. SourceXml can be the text of any XML document, including an XML data packet. It must, however, conform to the schema expected by the transformation specified by TransformationFile (see page 26478) or TransformationDocument (see page 26478). If the transformation converts a specific type of XML document into a data packet, SourceXml must conform to the expected schema. If the transformation converts a data packet into an XML document, SourceXml is an XML data packet with the expected metadata.

SourceXmlDocument (see page 26476) is used only if the SourceXMLFile property has not been set. If both SourceXMLFile and SourceXml are empty strings, TXMLTransform (see page 26467) uses the SourceXmlDocument (see page 26476) property instead. SourceXml takes precedence over SourceXmlDocument (see page 26476): If SourceXmlDocument (see page 26476) is set, TXMLTransform (see page 26467) still uses SourceXml (even if its value is not a valid string of XML).

See Also

SourceXmlDocument (see page 26476)

TransformationFile (see page 26478)

TransformationDocument (see page 26478)

SourceXmlFile (see page 26477)

Converting XML Documents into Data Packets

198.1.2.3.6 TXMLTransform.SourceXmlDocument Property

Specifies the DOM interface for the XML document to be converted.

Pascal

```
property SourceXmlDocument: IDOMDocument;
```

C++

```
__property IDOMDocument SourceXmlDocument;
```

Description

Specify the document to be translated as the value of SourceXmlDocument. SourceXmlDocument can be any type of XML document, including an XML data packet. It must, however, conform to the schema expected by the transformation specified by TransformationFile (see page 26478) or TransformationDocument (see page 26478). If the transformation converts a specific type of XML document into a data packet, then SourceXmlDocument is a document with the expected schema. If the transformation converts a data packet into an XML document, SourceXmlDocument is the interface to an XML data packet with the expected metadata.

SourceXmlDocument is used only if the SourceXMLFile and SourceXml (see page 26476) properties have not been set.

See Also

SourceXmlFile (see page 26477)

TransformationFile (see page 26478)

TransformationDocument (see page 26478)

SourceXml (see page 26476)

Converting XML Documents into Data Packets

198.1.2.3.7 TXMLTransform.SourceXmlFile Property

Specifies an XML document or XML data packet file to be translated.

Pascal

```
property SourceXmlFile: string;
```

C++

```
__property AnsiString SourceXmlFile;
```

Description

Specify the file to be translated as the value of SourceXmlFile. SourceXmlFile can be any type of XML document, including an XML data packet. It must, however, conform to the schema expected by the transformation specified by TransformationFile (see page 26478) or TransformationDocument (see page 26478). If the transformation converts a specific type of XML document into a data packet, SourceXmlFile is a document with the expected schema. If the transformation converts a data packet into an XML document, SourceXmlFile is an XML data packet with the expected metadata.

If SourceXmlFile is an empty string, TXMLTransform (see page 26467) uses the SourceXml (see page 26476) or SourceXmlDocument (see page 26476) property instead. SourceXmlFile takes precedence over both of these other properties. If SourceXml (see page 26476) or SourceXmlDocument (see page 26476) is set, TXMLTransform (see page 26467) still uses SourceXmlFile (even if it does not specify a valid file name).

See Also

SourceXmlDocument (see page 26476)

TransformationFile (see page 26478)

TransformationDocument (see page 26478)

SourceXml (see page 26476)

Converting XML Documents into Data Packets

198.1.2.3.8 TXMLTransform.TransformationDocument Property

Specifies the transformation that indicates how to convert the source XML document.

Pascal

```
property TransformationDocument: IDOMDocument;
```

C++

```
__property IDOMDocument TransformationDocument;
```

Description

TransformationDocument is the DOM interface for the transformation that contains instructions for converting the source XML document into the XML that is the value of Data (see page 26473) and ResultDocument (see page 26475).

A transformation is a special type of XML document that defines the mapping between the nodes of a source document and a destination document and the structure of the destination document. Each transformation is specific to a particular XML schema and data packet format and represents a one-way transformation. Use the xmlmapper utility to create transformations.

If the EmptyDestinationDocument (see page 26474) property is set, TXMLTransform (see page 26467) uses that to override the default structure for the destination document that is defined in TransformationDocument.

TransformationDocument is used only if the value of TransformationFile (see page 26478) is an empty string.

See Also

TransformXML (see page 26472)

TransformationFile (see page 26478)

Data (see page 26473)

EmptyDestinationDocument (see page 26474)

ResultDocument (see page 26475)

Converting XML Documents into Data Packets

Using an XML Document as the Client of a Provider

Using an XML Document as the Source for a Provider

198.1.2.3.9 TXMLTransform.TransformationFile Property

Specifies the transformation file that indicates how to convert the source XML document.

Pascal

```
property TransformationFile: string;
```

C++

```
__property AnsiString TransformationFile;
```

Description

TransformationFile is a file that contains instructions for converting the source XML document into the XML that is the value of Data (see page 26473) and ResultDocument (see page 26475). If the source document is a general XML document, TransformationFile represents a transformation from that document's schema into a data packet. If the source document is an XML data packet, TransformationFile represents a transformation from that data packet into another type of XML document.

A transformation file is a special type of XML file with the .xtr extension. Each transformation file is specific to a particular XML schema and data packet format and represents a one-way transformation. Use the xmlmapper utility to create transformation files.

If the EmptyDestinationDocument (see page 26474) property is set, TXMLTransform (see page 26467) uses that to override the default structure for the destination document that is defined in the transformation file.

TransformationFile takes precedence over the TransformationDocument (see page 26478) property. That is, TXMLTransform (see page 26467) only uses TransformationDocument (see page 26478) if TransformationFile is an empty string.

See Also

TransformXML (see page 26472)

Data (see page 26473)

TransformationDocument (see page 26478)

ResultDocument (see page 26475)

Converting XML Documents into Data Packets

Using an XML Document as the Client of a Provider

Using an XML Document as the Source for a Provider

198.1.2.4 TXMLTransform Events

The events of the TXMLTransform class are listed here.

Published Events

	Name	Description
	AfterEachRow (see page 26479)	Occurs after TXMLTransform (see page 26467) has finished transforming a record in the XML data packet.
	AfterEachRowSet (see page 26480)	Occurs after TXMLTransform (see page 26467) has finished transforming the entire XML data packet or a nested detail set.
	BeforeEachRow (see page 26480)	Occurs immediately before TXMLTransform (see page 26467) starts transforming a record in the XML data packet.
	BeforeEachRowSet (see page 26481)	Occurs immediately before TXMLTransform (see page 26467) starts transforming the entire XML data packet or a nested detail set.
	OnTranslate (see page 26481)	Occurs when the TXMLTransform (see page 26467) component needs to translate a user-defined node.

198.1.2.4.1 TXMLTransform.AfterEachRow Event

Occurs after TXMLTransform (see page 26467) has finished transforming a record in the XML data packet.

Pascal

```
property AfterEachRow: TRowEvent;
```

C++

```
__property TRowEvent AfterEachRow;
```

Description

Write an AfterEachRow event handler to take specific action after the XML transform component has translated each row in the data packet. For example, you can use AfterEachRow to read the data from the generated row and use it to compute the value of an internally calculated field.

See Also

TransformXML (see page 26472)

AfterEachRowSet (see page 26480)

BeforeEachRow (see page 26480)

Data (see page 26473)

198.1.2.4.2 TXMLTransform.AfterEachRowSet Event

Occurs after TXMLTransform (see page 26467) has finished transforming the entire XML data packet or a nested detail set.

Pascal

```
property AfterEachRowSet: TRowEvent;
```

C++

```
__property TRowEvent AfterEachRowSet;
```

Description

Write an AfterEachRowSet event handler to take specific action after the XML transform component has translated an entire dataset. This can be a nested detail set, or the entire XML data packet.

For example, you can use AfterEachRowSet with the BeforeEachRowSet (see page 26481) and OnTranslate (see page 26481) methods to compute summary values: initialize a variable in the BeforeEachRowSet (see page 26481) method, update it for every OnTranslate (see page 26481) event, and use the result in the AfterEachRowSet event.

See Also

TransformXML (see page 26472)

AfterEachRow (see page 26479)

BeforeEachRow (see page 26480)

Data (see page 26473)

198.1.2.4.3 TXMLTransform.BeforeEachRow Event

Occurs immediately before TXMLTransform (see page 26467) starts transforming a record in the XML data packet.

Pascal

```
property BeforeEachRow: TRowEvent;
```

C++

```
__property TRowEvent BeforeEachRow;
```

Description

Write a BeforeEachRow event handler to take specific action before the XML transform component starts translating each row in the data packet.

See Also

TransformXML (see page 26472)

BeforeEachRowSet (see page 26481)

AfterEachRow (see page 26479)

Data (see page 26473)

198.1.2.4.4 TXMLTransform.BeforeEachRowSet Event

Occurs immediately before TXMLTransform (see page 26467) starts transforming the entire XML data packet or a nested detail set.

Pascal

```
property BeforeEachRowSet: TRowEvent;
```

C++

```
__property TRowEvent BeforeEachRowSet;
```

Description

Write an BeforeEachRowSet event handler to take specific action before the XML transform component starts translating a dataset. This can be a nested detail set, or the entire XML data packet.

For example, you can use BeforeEachRowSet with the AfterEachRowSet (see page 26480) and OnTranslate (see page 26481) methods to compute summary values: initialize a variable in the BeforeEachRowSet method, update it for every OnTranslate (see page 26481) event, and use the result in the AfterEachRowSet (see page 26480) event.

See Also

TransformXML (see page 26472)

AfterEachRowSet (see page 26480)

BeforeEachRow (see page 26480)

Data (see page 26473)

198.1.2.4.5 TXMLTransform.OnTranslate Event

Occurs when the TXMLTransform (see page 26467) component needs to translate a user-defined node.

Pascal

```
property OnTranslate: TTranslateEvent;
```

C++

```
__property TTranslateEvent OnTranslate;
```

Description

Write an OnTranslate event handler to provide the translation for user-defined nodes.

TXMLTransform (see page 26467) performs a translation when an application reads the Data (see page 26473) property or calls the TransformXML (see page 26472) method. If the transformation document that governs how to perform that translation indicates that a node in the XML document is user defined, TXMLTransform (see page 26467) generates an OnTranslate event when it needs to translate to or from that node. OnTranslate only occurs for nodes that are marked as user defined.

If you do not translate a user-defined node in an OnTranslate event handler, the corresponding node in the translated XML is assigned the source XML node's value, if available. Otherwise, it is left blank.

See Also

TransformXML (see page 26472)

TransformationFile (see page 26478)

TransformationDocument (see page 26478)

Data (see page 26473)

Converting XML Documents into Data Packets

Using an XML Document as the Client of a Provider

Using an XML Document as the Source for a Provider

198.1.3 TXMLTransformClient Class

TXMLTransformClient acts as an adapter between an XML document and a provider component.

Class Hierarchy

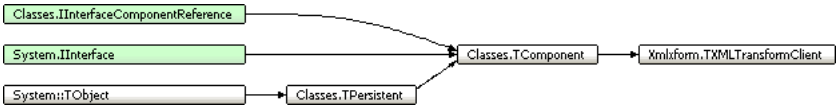

Pascal

```
TXMLTransformClient = class(TComponent);
```

C++

```
class TXMLTransformClient : public TComponent;
```

File

Xmlxform

Description

Use TXMLTransformClient to convert the data from a provider into an XML document or to apply updates in the form of an XML document to a provider component. TXMLTransformClient can act as the client of any provider component, local or remote. It uses a transformation file that is generated using the xmlmapper utility to convert between XML documents and the data packets and delta packets used by the provider.

See Also

TXMLTransform (see page 26467)

TXMLTransformProvider (see page 26492)

Using an XML Document as the Client of a Provider

198.1.3.1 TXMLTransformClient Members

The following tables list the members exposed by TXMLTransformClient.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.

	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetupAppServer (see page 26489)	This is SetupAppServer, a member of class TXMLTransformClient.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	ApplyUpdates (see page 26486)	Inserts or deletes the records corresponding to the data in an XML document.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.

	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 26485)	Creates and initializes a TXMLTransformClient (see page 26482) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 26485)	Destroys the instance of a TXMLTransformClient (see page 26482) component.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
V	GetDataAsXml (see page 26487)	Fetches a data packet from the provider and converts it to an XML document.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetParentComponent (see page 2778)	Returns the containing Component.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetParams (see page 26488)	Sets input parameters to values supplied in an XML document.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Properties

	Name	Description
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	ProviderName (see page 26489)	Specifies the name of a provider component that supplies data to and applies updates from the XML transform client.
	RemoteServer (see page 26490)	Specifies the connection component that this XML transform client uses to connect to an application server.
	Tag (see page 2808)	Stores an integer value as part of a component.
R	TransformApplyUpdates (see page 26490)	Specifies the TXMLTransform (see page 26467) component that converts XML into delta packets that are then sent to the provider to apply updates.
R	TransformGetData (see page 26491)	Specifies the TXMLTransform (see page 26467) component that converts data packets from the provider into XML.
R	TransformSetParams (see page 26491)	Specifies the TXMLTransform (see page 26467) component that converts the XML supplied to the SetParams (see page 26488) method into a data packet format.

198.1.3.2 TXMLTransformClient.Create Constructor

Creates and initializes a TXMLTransformClient (see page 26482) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TXMLTransformClient(TComponent * AOwner);
```

Description

Call Create to instantiate an XML transform client at runtime. XML transform clients placed in forms or data modules at design time are created automatically.

The AOwner parameter specifies another component (typically a form or data module) that is responsible for freeing the TXMLTransformClient (see page 26482) object.

See Also

Owner (see page 2807)

198.1.3.3 TXMLTransformClient.Destroy Destructor

Destroys the instance of a TXMLTransformClient (see page 26482) component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLTransformClient();
```

Description

Do not call Destroy directly in an application. Usually destruction of TXMLTransformClient (see page 26482) objects is handled automatically. If an application creates its own instances of TXMLTransformClient (see page 26482) without assigning an Owner, however, it should call Free, which verifies that the dataset component is not nil before calling Destroy.

See Also

Owner (see page 2807)

Create (see page 26485)

Free (see page 22300)

198.1.3.4 TXMLTransformClient Methods

The methods of the TXMLTransformClient class are listed here.

Protected Methods

	Name	Description
	SetupAppServer (see page 26489)	This is SetupAppServer, a member of class TXMLTransformClient.

Public Methods

	Name	Description
	ApplyUpdates (see page 26486)	Inserts or deletes the records corresponding to the data in an XML document.
	GetDataAsXml (see page 26487)	Fetches a data packet from the provider and converts it to an XML document.
	SetParams (see page 26488)	Sets input parameters to values supplied in an XML document.

198.1.3.4.1 TXMLTransformClient.ApplyUpdates Method

Inserts or deletes the records corresponding to the data in an XML document.

Pascal

```
function ApplyUpdates(const UpdateXML: string; const UpdateTransformFile: string; MaxErrors:
Integer): Integer; virtual;
```

C++

```
virtual __fastcall int ApplyUpdates(const AnsiString UpdateXML, const AnsiString
UpdateTransformFile, int MaxErrors);
```

Description

Call ApplyUpdates to transform an XML document into a delta packet that inserts or deletes all of the resulting records and send it to the associated provider so that it can be applied. Before you can call ApplyUpdates, you must define the transformation to use by setting properties on the XML transform component specified by TransformApplyUpdates (see page 26490).

UpdateXML is the XML document that can be transformed into a delta packet. Typically, it contains the contents of an XML document file.

UpdateTransformFile is the name of a transformation file that converts UpdateXML into an insert or delete delta packet. A transformation file is a special type of XML file with the .xtr extension. The transformation file is specific to a particular XML

schema, data packet format, and update operation (insert or delete). Use the xmlmapper utility to create the transformation file.

Note: If UpdateTransformFile includes any user-defined conversions, ApplyUpdates generates an OnTranslate event on TransformApplyUpdates (see page 26490) when it needs to transform user-defined nodes in the UpdateXML string into corresponding field values.

MaxErrors indicates the maximum number of errors that the provider should allow before prematurely stopping the update operation. Set MaxErrors to –1 to indicate that there is no limit to the number of errors.

ApplyUpdates returns the number of errors it encountered.

See Also

ProviderName (see page 26489)

TransformApplyUpdates (see page 26490)

ApplyUpdates

Using an XML Document as the Client of a Provider

198.1.3.4.2 TXMLTransformClient.GetDataAsXml Method

Fetches a data packet from the provider and converts it to an XML document.

Pascal

```
function GetDataAsXml(const PublishTransformFile: string): string; virtual;
```

C++

```
virtual __fastcall AnsiString GetDataAsXml(const AnsiString PublishTransformFile);
```

Description

Call GetDataAsXml to obtain an XML document that is the translation of a data packet from the associated provider. GetDataAsXml

1. Passes any parameters supplied to the SetParams (see page 26488) method to the provider specified by ProviderName (see page 26489).
2. Obtains a data packet that the provider returns.
3. Uses the transform component specified by TransformGetData (see page 26491) to convert the data packet to an XML document. If the PublishTransformFile parameter is not an empty string, GetDataAsXml uses the file specified by this parameter as the transformation file. Otherwise, it uses the transformation file specified on the TransformGetData (see page 26491) property.
4. Generates an OnTranslate event for every user-defined conversion in its transformation. The application can use this event to perform the user-defined conversion that converts field values into XML nodes.
5. Returns the resulting XML document as a string.

Once you have obtained the resulting, translated XML document, you can assign it to a TXMLDocument component or save it to a file.

Note: Before you can call GetDataAsXml, you must supply any input parameters using the SetParams (see page 26488) method and set properties on TransformGetData (see page 26491) to indicate what transformation to perform.

See Also

ProviderName (see page 26489)

TransformGetData (see page 26491)

XML (see page 26144)

OnTranslate (see page 26481)

SetParams (see page 26488)

Using an XML Document as the Client of a Provider

198.1.3.4.3 TXMLTransformClient.SetParams Method

Sets input parameters to values supplied in an XML document.

Pascal

```
procedure SetParams(const ParamsXml: string; const ParamsTransformFile: string);
```

C++

```
__fastcall SetParams(const AnsiString ParamsXml, const AnsiString ParamsTransformFile);
```

Description

Call SetParams to supply input parameter values before fetching data from the provider. Parameter values can represent the input parameters to a query or stored procedure, or they can represent field values that filter the data when the provider represents the data in a TTable or TSQLTable component.

ParamsXML is the XML document that represents the parameter values to set. Typically, it contains the contents of an XML document file.

ParamsTransformFile is the name of a transformation file that converts ParamsXml into a data packet. A transformation file is a special type of XML file with the .xtr extension. The transformation file is specific to a particular XML schema and data packet format. Use the xmlmapper.exe utility to create the transformation file.

Note: If ParamsTransformFile includes any user-defined conversions, the component specified by TransformSetParams (see page 26491) generates an OnTranslate event when it needs to transform user-defined nodes in the ParamsXml string into corresponding parameter values.

The TXMLTransformClient (see page 26482) component stores parameter values internally. Each time you call SetParams, you update those internally stored values. Parameter values from previous calls are not cleared, so be sure to set every parameter value you want to change.

The next the GetDataAsXml (see page 26487) method is called, TXMLTransformClient (see page 26482) sends the currently stored parameter values to the provider.

See Also

GetDataAsXml (see page 26487)

OnTranslate (see page 26481)

ProviderName (see page 26489)

TTable (see page 10168)

TSQLTable (see page 20337)

TransformSetParams (see page 26491)

Using an XML Document as the Client of a Provider

198.1.3.4.4 TXMLTransformClient.SetupAppServer Method

Pascal

```
procedure SetupAppServer;
```

C++

```
__fastcall SetupAppServer();
```

Description

This is SetupAppServer, a member of class TXMLTransformClient.

198.1.3.5 TXMLTransformClient Properties

The properties of the TXMLTransformClient class are listed here.

Published Properties

	Name	Description
	ProviderName (see page 26489)	Specifies the name of a provider component that supplies data to and applies updates from the XML transform client.
	RemoteServer (see page 26490)	Specifies the connection component that this XML transform client uses to connect to an application server.
R	TransformApplyUpdates (see page 26490)	Specifies the TXMLTransform (see page 26467) component that converts XML into delta packets that are then sent to the provider to apply updates.
R	TransformGetData (see page 26491)	Specifies the TXMLTransform (see page 26467) component that converts data packets from the provider into XML.
R	TransformSetParams (see page 26491)	Specifies the TXMLTransform (see page 26467) component that converts the XML supplied to the SetParams (see page 26488) method into a data packet format.

198.1.3.5.1 TXMLTransformClient.ProviderName Property

Specifies the name of a provider component that supplies data to and applies updates from the XML transform client.

Pascal

```
property ProviderName: string;
```

C++

```
__property AnsiString ProviderName;
```

Description

Use ProviderName to specify the provider object that supplies the XML transform client with data packets and applies updates on its behalf. This property is passed as a parameter to IAppServer interface calls when the client fetches data from or applies updates to the source dataset, its database server, or the source XML document.

The provider specified by ProviderName can reside in the same application as the XML transform client or in an application server running on another system.

If the provider is in the same application as the XML transform client, it must have the same Owner as the client.

If the provider is on a remote application server, it must be exported by that application server. If the application server contains multiple provider objects, you can set ProviderName to any of them once the XML transform client establishes an initial connection to the application server through its RemoteServer (see page 26490) property.

ProviderName can be set from a drop-down in the Object Inspector once the RemoteServer (see page 26490) property is set if

The connection component is a SOAP connection.

The connection component is a Web connection or socket connection (Windows only).

The application server is COM-based and registered on the client system using TRegsvr.exe (Windows only).

See Also

RemoteServer (see page 26490)

GetDataAsXml (see page 26487)

IAppServer (see page 16450)

ApplyUpdates (see page 26486)

Using an XML Document as the Client of a Provider

198.1.3.5.2 TXMLTransformClient.RemoteServer Property

Specifies the connection component that this XML transform client uses to connect to an application server.

Pascal

```
property RemoteServer: TCustomRemoteServer;
```

C++

```
__property TCustomRemoteServer RemoteServer;
```

Description

If the XML transform client is in the client portion of a multi-tiered application, use RemoteServer to specify the connection component that connects to the application server. The connection component locates the application server and obtains its IAppServer interface, which the XML transform client can use for communicating with a remote provider.

If a client application uses more than connection component, the application can change which connection an XML transform client uses by changing the value for this property.

See Also

ProviderName (see page 26489)

TSocketConnection (see page 18928)

TDCOMConnection (see page 16008)

TWebConnection (see page 18968)

TSOAPConnection

AppServer (see page 8532)

IAppServer (see page 16450)

Using an XML Document as the Client of a Provider

198.1.3.5.3 TXMLTransformClient.TransformApplyUpdates Property

Specifies the TXMLTransform (see page 26467) component that converts XML into delta packets that are then sent to the provider to apply updates.

Pascal

```
property TransformApplyUpdates: TXMLTransform;
```

C++

```
__property TXMLTransform TransformApplyUpdates;
```

Description

Use TransformApplyUpdates to access the subcomponent that transforms the XML passed as the UpdateXML parameter of the ApplyUpdates (see page 26486) method into a delta packet, which is then forwarded to the provider. If the transformation passed to the ApplyUpdates (see page 26486) method includes any user-defined nodes, you must supply TransformApplyUpdates with an OnTranslate event handler.

See Also

ApplyUpdates (see page 26486)

ProviderName (see page 26489)

OnTranslate (see page 26481)

198.1.3.5.4 TXMLTransformClient.TransformGetData Property

Specifies the TXMLTransform (see page 26467) component that converts data packets from the provider into XML.

Pascal

```
property TransformGetData: TXMLTransform;
```

C++

```
__property TXMLTransform TransformGetData;
```

Description

Use TransformGetData to access the subcomponent that performs the transformation of data packets from the provider into another XML format. When an application calls the GetDataAsXml (see page 26487) method, this subcomponent converts the data packet from the provider into XML.

Before you can call GetDataAsXml (see page 26487), you must specify the transformation that this subcomponent uses by setting its TransformationFile or TransformationDocument property. In addition, if the transformation includes any user-defined nodes, you must supply TransformGetData with an OnTranslate event handler.

See Also

GetDataAsXml (see page 26487)

TransformationFile (see page 26478)

TransformationDocument (see page 26478)

OnTranslate (see page 26481)

Using an XML Document as the Client of a Provider

198.1.3.5.5 TXMLTransformClient.TransformSetParams Property

Specifies the TXMLTransform (see page 26467) component that converts the XML supplied to the SetParams (see page 26488) method into a data packet format.

Pascal

```
property TransformSetParams: TXMLTransform;
```

C++

```
__property TXMLTransform TransformSetParams;
```

Description

Use TransformSetParams to directly access the subcomponent that performs the transformation of an XML document into a set of parameter values. When an application calls the SetParams (see page 26488) method, this subcomponent converts the XML supplied as the ParamsXml parameter into a data packet, which it then uses to read and store parameter values. These parameter values are later passed to the provider when you call GetDataAsXml (see page 26487).

If the transformation passed to the SetParams (see page 26488) method includes any user-defined nodes, you must supply TransformSetParams with an OnTranslate event handler.

See Also

SetParams (see page 26488)

ProviderName (see page 26489)

OnTranslate (see page 26481)

GetDataAsXml (see page 26487)

Using an XML Document as the Client of a Provider

198.1.4 TXMLTransformProvider Class

TXMLTransformProvider provides data from and resolves updates to an XML document.

Class Hierarchy

Pascal

```
TXMLTransformProvider = class(TCustomProvider);
```

C++

```
class TXMLTransformProvider : public TCustomProvider;
```

File

Xmlxform

Description

Use TXMLTransformProvider to provide data from an XML document to a client dataset or XML broker and to resolve updates from that client dataset or XML broker back to the source XML document. TXMLTransformProvider can be a part of the same application as the client dataset or XML broker, or it can be placed in the application server of a multi-tiered application. It serves as a translator between an XML document and a client dataset.

TDataSetProvider packages data from the source XML document and passes it in one or more transportable data packets to the client dataset or XML broker.

When the provider supplies data to a client dataset, the client dataset reconstructs the data in the data packet to create a local, in-memory copy for user access. When user access is complete, the client dataset repackages any changed data and sends the updates back to the provider. The provider applies the updates to the source XML document.

When the provider supplies data to an XML broker, the XML broker adds the data packet, in XML format, to an HTML document that is downloaded by a Web client. When the XML broker receives updates from the Web client, it sends the updates to the

provider, which applies them to the source XML document.

Client datasets and XML brokers communicate with a provider through the IAppServer interface. In multi-tiered applications, this is the interface of the remote data module that contains the provider. To enable the remote data module to pass calls to the provider, the provider's Exported property must be true, and its Owner property must specify the remote data module.

See Also

TClientDataSet (see page 8289)

TDataSetProvider (see page 18508)

TXMLTransform (see page 26467)

TRemoteDataModule (see page 6585)

TCRemoteDataModule (see page 6576)

TSoapDataModule (see page 19884)

TXMLTransformClient (see page 26482)

TCustomProvider (see page 18449)

Determining the Source of Data

Using Provider Components

Using an XML Document as the Source for a Provider

198.1.4.1 TXMLTransformProvider Members

The following tables list the members exposed by TXMLTransformProvider.

Protected Events

	Name	Description
	AfterExecute (see page 18476)	Occurs after the provider passes an execute command to its dataset.
	AfterGetParams (see page 18477)	This is AfterGetParams, a member of class TCustomProvider.
	BeforeExecute (see page 18479)	This is BeforeExecute, a member of class TCustomProvider.
	BeforeGetParams (see page 18480)	Occurs before the provider creates an OleVariant that encodes the current parameter values and sends them to a client dataset.
	OnValidate (see page 18483)	This is OnValidate, a member of class TCustomProvider.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DoAfterApplyUpdates (see page 18460)	Generates an AfterApplyUpdates (see page 18475) event.
V	DoAfterExecute (see page 18461)	Generates an AfterExecute (see page 18476) event.

	DoAfterGetParams (see page 18461)	Generates an AfterGetParams (see page 18477) event.
	DoAfterGetRecords (see page 18462)	Generates an AfterGetRecords (see page 18477) event.
	DoAfterRowRequest (see page 18462)	Generates an AfterRowRequest (see page 18478) event.
	DoApplyUpdates (see page 18463)	This is DoApplyUpdates, a member of class TCustomProvider.
	DoBeforeApplyUpdates (see page 18463)	Generates a BeforeApplyUpdates (see page 18479) event.
	DoBeforeExecute (see page 18464)	Generates a BeforeExecute (see page 18479) event.
	DoBeforeGetParams (see page 18464)	Generates a BeforeGetParams (see page 18480) event.
	DoBeforeGetRecords (see page 18465)	Generates a BeforeGetRecords (see page 18480) event.
	DoBeforeRowRequest (see page 18465)	Generates a BeforeRowRequest (see page 18482) event.
	DoGetRecords (see page 18466)	This is DoGetRecords, a member of class TCustomProvider.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (displDs).
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	InternalApplyUpdates (see page 26498)	Provides the underlying implementation of the ApplyUpdates method.
	InternalExecute (see page 18470)	Provides the underlying implementation of the Execute (see page 18466) method.
	InternalGetParams (see page 18471)	Returns the requested parameter values as a Variant array.
	InternalGetRecords (see page 26498)	Provides the underlying implementation of the GetRecords method.
	InternalRowRequest (see page 18472)	Provides the underlying implementation of the RowRequest (see page 18472) method.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
	Notification (see page 2783)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	SetActiveUpdateException (see page 18473)	This is SetActiveUpdateException, a member of class TCustomProvider.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteState (see page 2793)	This is WriteState, a member of class TComponent.

Protected Properties

	Name	Description
R	ActiveUpdateException (see page 18474)	This is ActiveUpdateException, a member of class TCustomProvider.
	OwnerData (see page 18475)	This is OwnerData, a member of class TCustomProvider.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	ApplyUpdates (see page 18459)	This is the overview for the ApplyUpdates method overload.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 26497)	Creates and initializes a TXMLTransformProvider (see page 26492) object.
V	DataRequest (see page 18460)	Generates an OnDataRequest (see page 18482) event.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 26497)	Frees an instance of a provider object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Execute (see page 18466)	Executes a query, stored procedure, or custom SQL statement.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
V	GetParams (see page 18467)	Returns the current parameter values.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetRecords (see page 18468)	This is the overview for the GetRecords method overload.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
V	RowRequest (see page 18472)	Returns information from a specified record.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
R	Data (see page 18474)	Denotes a data packet created by the provider.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
	Exported (see page 18474)	Allows client applications to pass calls to the provider when it resides on an application server.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.

Published Events

	Name	Description
	AfterApplyUpdates (see page 26502)	Occurs after the provider applies updates.
	AfterGetRecords (see page 26503)	Occurs after the provider creates a data packet to send to the client dataset.
	AfterRowRequest (see page 26504)	Occurs after the provider fetches new information about the current record.
	BeforeApplyUpdates (see page 26504)	Occurs before the provider applies updates from a client dataset.
	BeforeGetRecords (see page 26505)	Occurs before the provider creates a data packet and sends it to a client dataset.
	BeforeRowRequest (see page 26506)	Occurs before the provider creates a delta packet with information about the current record.
	OnDataRequest (see page 26507)	Occurs when a provider calls its DataRequest method.

Published Properties

	Name	Description
	CacheData (see page 26499)	Specifies whether the provider caches the data packet internally in memory.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	Tag (see page 2808)	Stores an integer value as part of a component.
R	TransformRead (see page 26500)	Specifies the TXMLTransform (see page 26467) component that converts the source XML document into data packets.
R	TransformWrite (see page 26500)	Specifies the TXMLTransform (see page 26467) component that generates the new value of the source XML document when updates are applied.

	XMLDataFile (see page 26501)	Specifies the XML document from which the provider creates data packets and to which it applies updates.

Strict private Fields

	Name	Description
	FActiveUpdateAbortException (see page 18455)	This is FActiveUpdateAbortException, a member of class TCustomProvider.
	FAfterApplyUpdates (see page 18455)	This is FAfterApplyUpdates, a member of class TCustomProvider.
	FAfterExecute (see page 18456)	This is FAfterExecute, a member of class TCustomProvider.
	FAfterGetParams (see page 18456)	This is FAfterGetParams, a member of class TCustomProvider.
	FAfterGetRecords (see page 18456)	This is FAfterGetRecords, a member of class TCustomProvider.
	FAfterRowRequest (see page 18456)	This is FAfterRowRequest, a member of class TCustomProvider.
	FBeforeApplyUpdates (see page 18456)	This is FBeforeApplyUpdates, a member of class TCustomProvider.
	FBeforeExecute (see page 18457)	This is FBeforeExecute, a member of class TCustomProvider.
	FBeforeGetParams (see page 18457)	This is FBeforeGetParams, a member of class TCustomProvider.
	FBeforeGetRecords (see page 18457)	This is FBeforeGetRecords, a member of class TCustomProvider.
	FBeforeRowRequest (see page 18457)	This is FBeforeRowRequest, a member of class TCustomProvider.
	FExported (see page 18458)	This is FExported, a member of class TCustomProvider.
	FOnDataRequest (see page 18458)	This is FOnDataRequest, a member of class TCustomProvider.
	FOnValidate (see page 18458)	This is FOnValidate, a member of class TCustomProvider.
	FOwnerData (see page 18458)	This is FOwnerData, a member of class TCustomProvider.

Strict private Methods

	Name	Description
	GetData (see page 18467)	This is GetData, a member of class TCustomProvider.

198.1.4.2 TXMLTransformProvider.Create Constructor

Creates and initializes a TXMLTransformProvider (see page 26492) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TXMLTransformProvider(TComponent * AOwner);
```

Description

Call Create to instantiate an XML transform provider at runtime. Providers placed in forms or data modules at design time are created automatically.

The AOwner parameter specifies another component (typically a form or data module) that is responsible for freeing the provider.

See Also

Owner (see page 2807)

198.1.4.3 TXMLTransformProvider.Destroy Destructor

Frees an instance of a provider object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLTransformProvider();
```

Description

Do not call Destroy directly. Instead, use the delete keyword, which indirectly calls Destroy.

See Also

delete

198.1.4.4 TXMLTransformProvider Methods

The methods of the TXMLTransformProvider class are listed here.

Protected Methods

	Name	Description
	InternalApplyUpdates (see page 26498)	Provides the underlying implementation of the ApplyUpdates method.
	InternalGetRecords (see page 26498)	Provides the underlying implementation of the GetRecords method.

198.1.4.4.1 TXMLTransformProvider.InternalApplyUpdates Method

Provides the underlying implementation of the ApplyUpdates method.

Pascal

```
function InternalApplyUpdates(const Delta: OleVariant; MaxErrors: Integer; out ErrorCount:
Integer): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalApplyUpdates(const OleVariant Delta, int MaxErrors, int
ErrorCount);
```

Description

The ApplyUpdates method calls InternalApplyUpdates to apply the updates contained in a delta packet after generating the BeforeApplyUpdates event and before generating the AfterApplyUpdates event. The InternalApplyUpdates method of TCustomProvider is abstract (Delphi) or pure virtual (C++), meaning that it has no implementation. Descendant classes override this method to apply updates, resolve update errors, and return any updates that could not be applied.

Delta is the delta packet containing insertions, deletions, and modifications to be applied.

MaxErrors indicates the maximum number of errors that can be applied before stopping the update operation. If MaxErrors is –1, the provider tries to apply all updates.

ErrorCount returns the number of errors encountered during the update operation.

InternalApplyUpdates returns a data packet of updates that could not be applied to the database as an OleVariant.

See Also

applyupdates (see page 18459)

198.1.4.4.2 TXMLTransformProvider.InternalGetRecords Method

Provides the underlying implementation of the GetRecords method.

Pascal

```
function InternalGetRecords(Count: Integer; out RecsOut: Integer; Options: TGetRecordOptions;
const CommandText: WideString; var Params: OleVariant): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalGetRecords(int Count, int RecsOut, TGetRecordOptions
Options, const BSTR CommandText, OleVariant Params);
```

Description

The GetRecords method calls InternalGetRecords to assemble a data packet after calling DoBeforeGetRecords to apply the CommandText and Params and generate a BeforeGetRecords event. As implemented in TCustomProvider, InternalGetRecords always returns NULL. Descendant classes override this method to fetch the requested records and assemble them into a data packet.

Count indicates the number of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only the specified number of records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It can include grMetaData (include metadata), grReset (start providing records with the first one in the dataset), and grXML (encode the datapacket in XML rather than as an OleVariant) or grXMLUTF8 (same as grXML except extended characters are encoded using UTF8).

CommandText specifies an optional SQL statement that replaces the SQL of an associated query, or the name of a table or stored procedure that replaces the associated table or stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText or if the provider does not have an Options property.

Params specifies any parameter values that should be applied to CommandText or to an associated query or stored procedure. If the provider is associated with a TTable object, Params specifies field values that filter the result set. On exit from GetRecords, Params returns any output parameters.

Records are returned as a data packet in a Variant.

See Also

GetRecords (see page 18468)

DoBeforeGetRecords (see page 18465)

198.1.4.5 TXMLTransformProvider Properties

The properties of the TXMLTransformProvider class are listed here.

Published Properties

	Name	Description
	CacheData (see page 26499)	Specifies whether the provider caches the data packet internally in memory.
R	TransformRead (see page 26500)	Specifies the TXMLTransform (see page 26467) component that converts the source XML document into data packets.
R	TransformWrite (see page 26500)	Specifies the TXMLTransform (see page 26467) component that generates the new value of the source XML document when updates are applied.
	XMLDataFile (see page 26501)	Specifies the XML document from which the provider creates data packets and to which it applies updates.

198.1.4.5.1 TXMLTransformProvider.CacheData Property

Specifies whether the provider caches the data packet internally in memory.

Pascal

```
property CacheData: Boolean;
```

C++

```
__property Boolean CacheData;
```

Description

Set CacheData to true to specify that the provider should cache its data packet in memory after the first time it is read from the source XML document. When CacheData is false, every time the Data property is read, the GetRecords method is called, or updates are applied, the provider must reread the source document specified by XMLDataFile (see page 26501).

Caching the data improves performance on subsequent data fetches or when applying updates, but consumes additional resources.

See Also

Data (see page 18474)

TransformRead (see page 26500)

GetRecords (see page 18468)

TransformRead (see page 26500)

XMLDataFile (see page 26501)

198.1.4.5.2 TXMLTransformProvider.TransformRead Property

Specifies the TXMLTransform (see page 26467) component that converts the source XML document into data packets.

Pascal

```
property TransformRead: TXMLTransform;
```

C++

```
__property TXMLTransform TransformRead;
```

Description

Use TransformRead to access the subcomponent that transforms the source XML document into the data packet that is the value of the Data property. Before you can use TXMLTransformProvider (see page 26492), you must specify the transformation that this subcomponent uses by setting its TransformationFile or TransformationDocument property. In addition, if the transformation includes any user-defined nodes, you must supply TransformRead with an OnTranslate event handler.

See Also

XMLDataFile (see page 26501)

TransformWrite (see page 26500)

Data (see page 18474)

TransformationFile (see page 26478)

TransformationDocument (see page 26478)

OnTranslate (see page 26481)

Determining the Source of Data

Using an XML Document as the Source for a Provider

198.1.4.5.3 TXMLTransformProvider.TransformWrite Property

Specifies the TXMLTransform (see page 26467) component that generates the new value of the source XML document when

updates are applied.

Pascal

```
property TransformWrite: TXMLTransform;
```

C++

```
__property TXMLTransform TransformWrite;
```

Description

Use TransformWrite to access the subcomponent that transforms a data packet back into the format required by the source XML document. When a client calls the ApplyUpdates method, the delta packet is merged into the transformed data from the XML document, and then TransformWrite converts the resulting dataset into the format of the source XML document.

Before you can use TXMLTransformProvider (see page 26492) to apply updates, you must specify the transformation that this subcomponent uses by setting its TransformationFile or TransformationDocument property. In addition, if the transformation includes any user-defined nodes, you must supply TransformWrite with an OnTranslate event handler.

See Also

applyupdates (see page 18459)

TransformRead (see page 26500)

TransformationFile (see page 26478)

TransformationDocument (see page 26478)

OnTranslate (see page 26481)

Using an XML Document as the Source for a Provider

198.1.4.5.4 TXMLTransformProvider.XMLDataFile Property

Specifies the XML document from which the provider creates data packets and to which it applies updates.

Pascal

```
property XMLDataFile: string;
```

C++

```
__property AnsiString XMLDataFile;
```

Description

Use XMLDataFile to specify the XML document whose data the provider represents. When the Data property is read or the GetRecords method is called, TXMLTransformProvider (see page 26492) reads the contents of XMLDataFile and converts it into a data packet using the transform component specified by TransformRead (see page 26500). When the ApplyUpdates method is called, TXMLTransformProvider (see page 26492) uses the transform component specified by TransformWrite (see page 26500) to convert the data into the proper format for XMLDataFile and updates this XML document accordingly.

XMLDataFile is the same as the SourceXmlFile property of TransformRead (see page 26500).

See Also

Data (see page 18474)

GetRecords (see page 18468)

TransformRead (see page 26500)

TransformWrite (see page 26500)

applyupdates (see page 18459)

SourceXmlFile (see page 26477)

Determining the Source of Data

Using an XML Document as the Source for a Provider

198.1.4.6 TXMLTransformProvider Events

The events of the TXMLTransformProvider class are listed here.

Published Events

	Name	Description
	AfterApplyUpdates (see page 26502)	Occurs after the provider applies updates.
	AfterGetRecords (see page 26503)	Occurs after the provider creates a data packet to send to the client dataset.
	AfterRowRequest (see page 26504)	Occurs after the provider fetches new information about the current record.
	BeforeApplyUpdates (see page 26504)	Occurs before the provider applies updates from a client dataset.
	BeforeGetRecords (see page 26505)	Occurs before the provider creates a data packet and sends it to a client dataset.
	BeforeRowRequest (see page 26506)	Occurs before the provider creates a delta packet with information about the current record.
	OnDataRequest (see page 26507)	Occurs when a provider calls its DataRequest method.

198.1.4.6.1 TXMLTransformProvider.AfterApplyUpdates Event

Occurs after the provider applies updates.

Pascal

```
property AfterApplyUpdates: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterApplyUpdates;
```

Description

Write an AfterApplyUpdates event handler to return custom information to a client dataset after applying updates. AfterApplyUpdates is part of the mechanism by which a client dataset and a provider communicate information about updates. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_ApplyUpdates method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeApplyUpdates event, where it can respond to or change custom information that the client dataset sends.

2.If it is a dataset provider, the provider receives an OnUpdateData event where it can edit the updates before they are applied.

3. The provider applies the updates, and attempts to resolve problems.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider in its AfterApplyUpdates event handler.

Note: If ApplyUpdates is called without the OwnerData parameter, on entry to the event handler the OwnerData parameter contains the value set in the BeforeApplyUpdates event handler and it is ignored on exit from the AfterApplyUpdates event handler.

See Also

applyupdates (see page 18459)

AS_ApplyUpdates (see page 16451)

ApplyUpdates (see page 8394)

AfterApplyUpdates (see page 8497)

BeforeApplyUpdates (see page 18479)

OnUpdateData (see page 18443)

Responding to Client Update Requests

Communicating with the Client Dataset

198.1.4.6.2 TXMLTransformProvider.AfterGetRecords Event

Occurs after the provider creates a data packet to send to the client dataset.

Pascal

```
property AfterGetRecords: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterGetRecords;
```

Description

Write an AfterGetRecords event handler to send custom information to a client dataset when providing a data packet. AfterGetRecords is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_GetRecords method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeGetRecords event, where it can respond to or change custom information that the client dataset sends.

2.If it is a dataset provider, the provider receives an OnGetData event where it can edit the data before creating a data packet, and then generates the data packet.

3.The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.

4.The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider in its AfterGetRecords event handler.

See Also

AS_GetRecords (see page 16455)

OnGetData (see page 18443)

GetRecords (see page 18468)

AfterGetRecords (see page 8506)

BeforeGetRecords (see page 18480)

Communicating with the Client Dataset

Responding to Client Data Requests

198.1.4.6.3 TXMLTransformProvider.AfterRowRequest Event

Occurs after the provider fetches new information about the current record.

Pascal

```
property AfterRowRequest: TRemoteEvent;
```

C++

```
__property TRemoteEvent AfterRowRequest;
```

Description

Write an AfterRowRequest event handler to send custom information to a client dataset along with information about the current record. AfterRowRequest is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_RowRequest method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeRowRequest event, where it can respond to or change custom information that the client dataset sends.

2.The provider creates a delta packet that contains the requested information.

3.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeRowRequest event handler.

4.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider in its AfterRowRequest event handler.

See Also

AS_RowRequest (see page 16456)

RowRequest (see page 18472)

AfterRowRequest (see page 8509)

BeforeRowRequest (see page 18482)

FetchBlobs (see page 8414)

FetchParams (see page 8415)

RefreshRecord (see page 8442)

Communicating with the Client Dataset

198.1.4.6.4 TXMLTransformProvider.BeforeApplyUpdates Event

Occurs before the provider applies updates from a client dataset.

Pascal

```
property BeforeApplyUpdates: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeApplyUpdates;
```

Description

Write a BeforeApplyUpdates event handler to respond to custom information from a client dataset. BeforeApplyUpdates is part of

the mechanism by which a provider and a client dataset communicate information about updates. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset's ApplyUpdates method is called, the following events occur:

1.The client dataset receives a BeforeApplyUpdates event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeApplyUpdates event, where the OleVariant from the client dataset appears as the OwnerData parameter. The provider can respond to or change that information before applying the updates.

3.If it is a dataset provider, the provider applies the updates, and receives an OnUpdateError event for any problem records.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider in its AfterApplyUpdates event handler.

Note: If ApplyUpdates is called without the OwnerData parameter, the OwnerData parameter is nil (Delphi) or NULL (C++) on entry to the event handler and its value (on exit) is only passed to the AfterApplyUpdates event handler but not returned to the caller.

See Also

applyupdates (see page 18459)

AS_ApplyUpdates (see page 16451)

ApplyUpdates (see page 8394)

BeforeApplyUpdates (see page 8511)

AfterApplyUpdates (see page 18475)

Responding to Client Update Requests

Communicating with the Client Dataset

198.1.4.6.5 TXMLTransformProvider.BeforeGetRecords Event

Occurs before the provider creates a data packet and sends it to a client dataset.

Pascal

```
property BeforeGetRecords: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeGetRecords;
```

Description

Write a BeforeGetRecords event handler to respond to custom information from a client dataset. BeforeGetRecords is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset fetches data from the provider, the following events occur:

1.The client dataset receives a BeforeGetRecords event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeGetRecords event, where the OleVariant from the client dataset appears as the OwnerData parameter. The provider can respond to or change that information before it creates a data packet.

3.The provider generates the data packet.

4.The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.

5.The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider in its AfterGetRecords event handler.

Delphi Examples:

```
{
This example shows the implementation of a provider BeforeGetRecords
event handler and a RemoteDataModule UpdateRegistry method.  The
complete project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TPooledRDM.DataSetProvider1BeforeGetRecords(Sender: TObject;
  var OwnerData: OleVariant);
begin
  try
    Query1.Close;
    Query1.SQL.Text := OwnerData[0];
    if not VarIsNull(OwnerData[1]) and not VarIsEmpty(OwnerData[1]) then
    begin
      Query1.Open;
      if not Query1.Locate(Query1.Fields[0].FieldName, OwnerData[1], []) then
        raise Exception.Create('Record not found');
      Query1.Next;
    end;
  finally
    OwnerData := NULL;
  end;
end;
```

See Also

AS_GetRecords (see page 16455)

GetRecords (see page 18468)

BeforeGetRecords (see page 8515)

AfterGetRecords (see page 18477)

Communicating with the Client Dataset

Responding to Client Data Requests

198.1.4.6.6 TXMLTransformProvider.BeforeRowRequest Event

Occurs before the provider creates a delta packet with information about the current record.

Pascal

```
property BeforeRowRequest: TRemoteEvent;
```

C++

```
__property TRemoteEvent BeforeRowRequest;
```

Description

Write a BeforeRowRequest event handler to respond to custom information from a client dataset. BeforeRowRequest is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset fetches detail information about the current record or refreshes the current record, the following events occur:

1.The client dataset receives a BeforeRowRequest event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeRowRequest event, where the custom information appears as the OwnerData parameter. The provider can respond to or change that information before it looks up the requested information.

3.The provider creates a delta packet that contains the requested information.

4.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeRowRequest event handler.

5.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider in its AfterRowRequest event handler.

See Also

AS_RowRequest (see page 16456)

RowRequest (see page 18472)

AfterRowRequest (see page 18478)

BeforeRowRequest (see page 8520)

FetchBlobs (see page 8414)

FetchParams (see page 8415)

RefreshRecord (see page 8442)

Communicating with the Client Dataset

198.1.4.6.7 TXMLTransformProvider.OnDataRequest Event

Occurs when a provider calls its DataRequest method.

Pascal

```
property OnDataRequest: TDataRequestEvent;
```

C++

```
__property TDataRequestEvent OnDataRequest;
```

Description

Write an OnDataRequest event handler to do custom processing of data. OnDataRequest has no built-in use. Instead, it allows developers to augment the IAppServer interface so that clients in a multi-tiered application can communicate directly with an individual provider.

If an application does not provide an OnDataRequest event handler, calls to DataRequest return NULL.

Note: To provide the same sort of custom processing on the client side, use an automation method on the connection component.

See Also

DataRequest (see page 18460)

IAppServer (see page 16450)

Communicating with Providers Using Custom Events

Communicating with the Client Dataset

Responding to Client-generated Events

198.2 Types

The following table lists types in this documentation.

Types

Name	Description
TRowEvent (see page 26509)	This is type Xmlxform.TRowEvent.
TTranslateEvent (see page 26509)	This is type Xmlxform.TTranslateEvent.

198.2.1 Xmlxform.TRowEvent Type

Pascal

```
TRowEvent = procedure (Sender: TObject; Id: string; SrcNode: IDOMNode; DestNode: IDOMNode) of
object;
```

C++

```
(Sender: TObject; Id: string; SrcNode: IDOMNode; DestNode: IDOMNode) ( TRowEvent)();
```

File

Xmlxform

Description

This is type Xmlxform.TRowEvent.

198.2.2 Xmlxform.TTranslateEvent Type

Pascal

```
TTranslateEvent = procedure (Sender: TObject; Id: string; SrcNode: IDOMNode; var Value:
string; DestNode: IDOMNode) of object;
```

C++

```
(Sender: TObject; Id: string; SrcNode: IDOMNode; var Value: string; DestNode: IDOMNode) (
TTranslateEvent)();
```

File

Xmlxform

Description

This is type Xmlxform.TTranslateEvent.

199 XSBuiltIns Namespace

This is namespace XSBuiltIns.

Topics

Name	Description
Classes (see page 26513)	The following table lists classes in this documentation.
Functions (see page 26571)	The following table lists functions in this documentation.
Types (see page 26575)	The following table lists types in this documentation.
Constants (see page 26576)	The following table lists constants in this documentation.

Classes

	Name	Description
	EXSDateTimeException (see page 26513)	EXSDateTimeException is the exception class for errors encountered by the remotable objects that represent XML schema date and time values.
	EXSDecimalException (see page 26514)	EXSDecimalException is the exception class for errors encountered by the remotable objects that represent decimal values.
	EXSHexBinaryException (see page 26514)	EXSHexBinaryException is the exception class for errors encountered by the remotable objects that represent hexadecimal values.
	TXSCustomDateTime (see page 26515)	TXSCustomDateTime is the base class for classes that represent both date and time values for an invokable interface.
	TXSDate (see page 26534)	TXSDate implements the XML schema date type so that it can be used on invokable interfaces.
	TXSDateTime (see page 26541)	TXSDateTime implements the XML schema datetime type so that it can be used on invokable interfaces.
	TXSDecimal (see page 26545)	TXSDecimal implements the XML schema decimal type so that it can be used on invokable interfaces.
	TXSDuration (see page 26548)	TXSDuration implements the XML schema duration type so that it can be used on invokable interfaces.
	TXSHexBinary (see page 26553)	TXSHexBinary implements the XML schema hexBinary type so that it can be used on invokable interfaces.
	TXSString (see page 26556)	This is class XSBuiltIns.TXSString.
	TXSTime (see page 26558)	TXSTime implements the XML schema time type so that it can be used on invokable interfaces.

Constants

Name	Description
IncrementAmount (see page 26576)	This is constant XSBuiltIns.IncrementAmount.
MaxDay (see page 26577)	This is constant XSBuiltIns.MaxDay.
MaxMonth (see page 26577)	This is constant XSBuiltIns.MaxMonth.

MinDay (see page 26577)	This is constant XSBuiltIns.MinDay.
MinMonth (see page 26578)	This is constant XSBuiltIns.MinMonth.
SDecimal (see page 26578)	This is constant XSBuiltIns.SDecimal.
SHexMarker (see page 26578)	This is constant XSBuiltIns.SHexMarker.
SLocalTimeMarker (see page 26579)	This is constant XSBuiltIns.SLocalTimeMarker.
SNAN (see page 26579)	This is constant XSBuiltIns.SNAN.
SNegative (see page 26579)	This is constant XSBuiltIns.SNegative.
SoapDecimalSeparator (see page 26580)	This is constant XSBuiltIns.SoapDecimalSeparator.
SoapTimePrefix (see page 26580)	This is constant XSBuiltIns.SoapTimePrefix.
SPlus (see page 26580)	This is constant XSBuiltIns.SPlus.
SSciNotationMarker (see page 26581)	This is constant XSBuiltIns.SSciNotationMarker.
XMLDateSeparator (see page 26581)	This is constant XSBuiltIns.XMLDateSeparator.
XMLDayPos (see page 26581)	This is constant XSBuiltIns.XMLDayPos.
XMLDefaultYearDigits (see page 26582)	This is constant XSBuiltIns.XMLDefaultYearDigits.
XMLDurationDay (see page 26582)	This is constant XSBuiltIns.XMLDurationDay.
XMLDurationHour (see page 26582)	This is constant XSBuiltIns.XMLDurationHour.
XMLDurationMinute (see page 26583)	This is constant XSBuiltIns.XMLDurationMinute.
XMLDurationMonth (see page 26583)	This is constant XSBuiltIns.XMLDurationMonth.
XMLDurationSecond (see page 26583)	This is constant XSBuiltIns.XMLDurationSecond.
XMLDurationStart (see page 26584)	This is constant XSBuiltIns.XMLDurationStart.
XMLDurationYear (see page 26584)	This is constant XSBuiltIns.XMLDurationYear.
XMLHourOffsetMinusMarker (see page 26584)	This is constant XSBuiltIns.XMLHourOffsetMinusMarker.
XMLHourOffsetPlusMarker (see page 26585)	This is constant XSBuiltIns.XMLHourOffsetPlusMarker.
XMLMilSecPos (see page 26585)	This is constant XSBuiltIns.XMLMilSecPos.
XMLMonthPos (see page 26585)	This is constant XSBuiltIns.XMLMonthPos.
XMLTimeSeparator (see page 26586)	This is constant XSBuiltIns.XMLTimeSeparator.
XMLYearPos (see page 26586)	This is constant XSBuiltIns.XMLYearPos.

Functions

	Name	Description
	DateTimeToXMLTime (see page 26571)	Returns the XML string that represents a TDateTime value.
	DateTimeToXSDateTime (see page 26571)	Returns a TXSDateTime (see page 26541) object that encodes a TDateTime value.
	GetDataFromFile (see page 26572)	This is function XSBuiltIns.GetDataFromFile.
	InitXSTypes (see page 26572)	This is function XSBuiltIns.InitXSTypes.
	SoapFloatToStr (see page 26573)	This is function XSBuiltIns.SoapFloatToStr.
	SoapStrToFloat (see page 26573)	This is function XSBuiltIns.SoapStrToFloat.
	XMLTimeToDateTime (see page 26573)	Returns a TDateTime value to represent an XML datetime value.

Types

Name	Description
TXSTimeInstant (see page 26575)	This is type XSBuiltIns.TXSTimeInstant.

199.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EXSDateTimeException (see page 26513)	EXSDateTimeException is the exception class for errors encountered by the remotable objects that represent XML schema date and time values.
	EXSDecimalException (see page 26514)	EXSDecimalException is the exception class for errors encountered by the remotable objects that represent decimal values.
	EXSHexBinaryException (see page 26514)	EXSHexBinaryException is the exception class for errors encountered by the remotable objects that represent hexadecimal values.
	TXSCustomDateTime (see page 26515)	TXSCustomDateTime is the base class for classes that represent both date and time values for an invokable interface.
	TXSDate (see page 26534)	TXSDate implements the XML schema date type so that it can be used on invokable interfaces.
	TXSDateTime (see page 26541)	TXSDateTime implements the XML schema datetime type so that it can be used on invokable interfaces.
	TXSDecimal (see page 26545)	TXSDecimal implements the XML schema decimal type so that it can be used on invokable interfaces.
	TXSDuration (see page 26548)	TXSDuration implements the XML schema duration type so that it can be used on invokable interfaces.
	TXSHexBinary (see page 26553)	TXSHexBinary implements the XML schema hexBinary type so that it can be used on invokable interfaces.
	TXSString (see page 26556)	This is class XSBuiltIns.TXSString.
	TXSTime (see page 26558)	TXSTime implements the XML schema time type so that it can be used on invokable interfaces.

199.1.1 EXSDateTimeException Class

EXSDateTimeException is the exception class for errors encountered by the remotable objects that represent XML schema date and time values.

Class Hierarchy

Pascal

```
EXSDateTimeException = class(Exception);
```

C++

```
class EXSDateTimeException : public Exception;
```

File

XSBuiltIns

Description

EXSDateTimeException is raised by the TXSDate (see page 26534), TXSTime (see page 26558), TXSDateTime (see page 26541), and TXSDuration (see page 26548) objects when

An application assigns a value that is out of range to one of their properties. Such values include Month values greater than 12, Hour values greater than 23, Minute values greater than 59, and so on.

The string that represents the object's value does not have the proper format and can't be parsed.

See Also

TXSTime (see page 26558)

TXSDate (see page 26534)

TXSDateTime (see page 26541)

TXSDuration (see page 26548)

199.1.2 EXSDecimalException Class

EXSDecimalException is the exception class for errors encountered by the remotable objects that represent decimal values.

Class Hierarchy

Pascal

```
EXSDecimalException = class(Exception);
```

C++

```
class EXSDecimalException : public Exception;
```

File

XSBuiltIns

Description

EXSDecimalException is raised by TXSDecimal (see page 26545) when it is unable to convert between its string value and a TBcd value.

See Also

TXSDecimal (see page 26545)

TBcd

199.1.3 EXSHexBinaryException Class

EXSHexBinaryException is the exception class for errors encountered by the remotable objects that represent hexadecimal values.

Class Hierarchy

Pascal

```
EXSHexBinaryException = class(Exception);
```

C++

```
class EXSHexBinaryException : public Exception;
```

File

XSBuiltIns

Description

EXSHexBinaryException is raised by TXSHexBinary (see page 26553) when it encounters a string that contains invalid characters for a hexadecimal value.

See Also

TXSDecimal (see page 26545)

TBcd

199.1.4 TXSCustomDateTime Class

TXSCustomDateTime is the base class for classes that represent both date and time values for an invokable interface.

Class Hierarchy

Pascal

```
TXSCustomDateTime = class(TRemotableXS);
```

C++

```
class TXSCustomDateTime : public TRemotableXS;
```

File

XSBuiltIns

Description

Use TXSCustomDateTime as a base class when defining a class to represent a type that includes both date and time portions and that represents a scalar value in a WSDL document. When creating a TXSCustomDateTime descendant, override the XSToNative and NativeToXS methods to indicate how the value of the class can be converted to and from the string representation that is used to encode the value for transport.

On server applications, instances of TXSCustomDateTime descendants that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSCustomDateTime descendant is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSCustomDateTime instances that it passes as input parameters, and for freeing any TXSCustomDateTime instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 22398)

TXSDateTime (see page 26541)

TXSDuration (see page 26548)

199.1.4.1 TXSCustomDateTime Members

The following tables list the members exposed by TXSCustomDateTime.

Protected Fields

	Name	Description
	FDateTime (see page 26518)	This is FDateTime, a member of class TXSCustomDateTime.
	FSerializationOptions (see page 15559)	This is FSerializationOptions, a member of class TRemotable.

Protected Methods

	Name	Description
	GetAsDateTime (see page 26519)	Returns the value of the AsDateTime (see page 26527) property.
	GetAsUTCDateTime (see page 26519)	This is GetAsUTCDateTime, a member of class TXSCustomDateTime.
	GetDay (see page 26519)	Returns the value of the Day (see page 26528) property.
	GetFractionalSeconds (see page 26520)	Returns the value of the FractionalSeconds (see page 26529) property.
	GetHour (see page 26520)	Returns the value of the Hour (see page 26529) property.
	GetHourOffset (see page 26520)	Returns the value of the HourOffset (see page 26530) property.
	GetMillisecond (see page 26521)	Returns the value of the Millisecond (see page 26530) property.
	GetMinute (see page 26521)	Returns the value of the Minute (see page 26531) property.
	GetMinuteOffset (see page 26521)	Returns the value of the MinuteOffset (see page 26531) property.
	GetMonth (see page 26521)	Returns the value of the Month (see page 26532) property.
	GetSecond (see page 26522)	Returns the value of the Second (see page 26532) property.
	GetUseZeroMilliseconds (see page 26522)	Returns the value of the UseZeroMilliseconds (see page 26533) property.
	GetYear (see page 26522)	Returns the value of the Year (see page 26533) property.
	SetAsDateTime (see page 26523)	Sets the value of the AsDateTime (see page 26527) property.
	SetAsUTCDateTime (see page 26523)	Sets the value of the AsUTCDateTime (see page 26528) property.
V	SetDay (see page 26523)	Sets the value of the Day (see page 26528) property.
	SetFractionalSeconds (see page 26524)	Sets the value of the FractionalSeconds (see page 26529) property.
V	SetHour (see page 26524)	Sets the value of the Hour (see page 26529) property.
V	SetHourOffset (see page 26524)	Sets the value of the HourOffset (see page 26530) property.
V	SetMillisecond (see page 26524)	Sets the value of the Millisecond (see page 26530) property.
V	SetMinute (see page 26525)	Sets the value of the Minute (see page 26531) property.
V	SetMinuteOffset (see page 26525)	Sets the value of the MinuteOffset (see page 26531) property.
V	SetMonth (see page 26525)	Sets the value of the Month (see page 26532) property.
V	SetSecond (see page 26526)	Sets the value of the Second (see page 26532) property.
	SetUseZeroMilliseconds (see page 26526)	Sets the value of the UseZeroMilliseconds (see page 26533) property.
V	SetYear (see page 26526)	Sets the value of the Year (see page 26533) property.

Protected Properties

	Name	Description
	Millisecond (see page 26530)	Indicates the Millisecond of the second that the object represents.

Public Methods

	Name	Description
V	Create (see page 26517)	Creates and initializes a TXSCustomDateTime (see page 26515) object.
V	Destroy (see page 26517)	Frees the memory associated with the TXSCustomDateTime (see page 26515) object.
A	NativeToXS (see page 15581)	Returns a string that represents the value of a TRemotableXS (see page 15580) instance.
V	ObjectToSOAP (see page 15559)	Converts this remotable object to its SOAP representation.
V	ObjectToSOAP (see page 15582)	Converts this remotable object to its SOAP representation.
V	SOAPToObject (see page 15560)	Initializes this TRemotable (see page 15556) instance to represent the value of its SOAP representation.

V	SOAPToObject (see page 15582)	Initializes this TRemotableXS (see page 15580) instance to represent the value of its SOAP representation.
A	XSToNative (see page 15583)	Assigns the value of a string to the TRemotableXS (see page 15580) instance.

Public Properties

	Name	Description
	AsDateTime (see page 26527)	Represents the object's value as a TDateTime value.
	AsUTCDateTime (see page 26528)	Represents the object's value as a TDateTime value using UTC time.
	DataContext (see page 15561)	Specifies the object that controls the lifetime of a TRemotable (see page 15556) instance.
	Day (see page 26528)	Indicates the Day of the month that the object represents.
	FractionalSeconds (see page 26529)	Indicates any factional portion of a second beyond the seconds specified in the Second (see page 26532) property.
	Hour (see page 26529)	Indicates the hour of the day that the object represents.
	HourOffset (see page 26530)	Indicates the difference between the Hour (see page 26529) property and the hour in the local time zone.
	Minute (see page 26531)	Indicates the minute of the hour that the object represents.
	MinuteOffset (see page 26531)	Indicates the difference between the Minute (see page 26531) property and the minute of the hour in local time.
	Month (see page 26532)	Indicates the month of the year that the object represents.
	Second (see page 26532)	Indicates the second of the minute that the object represents.
R	SerializationOptions (see page 15561)	Indicates how the value of the TRemotable (see page 15556) descendant encodes its value.
	UseZeroMilliseconds (see page 26533)	Indicates whether the string representation of this object's value includes a portion for milliseconds when the value of Millisecond (see page 26530) is zero.
	Year (see page 26533)	Indicates the year that the object represents.

199.1.4.2 TXSCustomDateTime.Create Constructor

Creates and initializes a TXSCustomDateTime (see page 26515) object.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TXSCustomDateTime();
```

Description

Call Create to create a TXSCustomDateTime (see page 26515) object at runtime.

See Also

Destroy (see page 26517)

199.1.4.3 TXSCustomDateTime.Destroy Destructor

Frees the memory associated with the TXSCustomDateTime (see page 26515) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXSCustomDateTime();
```

Description

Do not call Destroy to free a TXSCustomDateTime (see page 26515) object. Instead, use the Free method, which checks that the TXSCustomDatetime reference is not nil before calling Destroy.

On server applications, there is typically no need to explicitly free a TXSCustomDateTime (see page 26515) instance (for example, one created as a return value or output parameter). By default, when a TXSCustomDateTime (see page 26515) descendant is created in a method that was called remotely using an invokable interface, it is added to a data context (the value of the DataContext property). As long as the remotable object belongs to the data context, the data context handles freeing the object. Similarly, TCustomXSDateTime objects that are passed in as a parameter belong to the data context.

On client applications, TXSCustomDateTime (see page 26515) instances created and passed to an invokable interface or returned as a parameter or result of an invokable interface must be freed by the calling application.

See Also

DataContext (see page 15561)

199.1.4.4 TXSCustomDateTime Fields

The fields of the TXSCustomDateTime class are listed here.

Protected Fields

	Name	Description
	FDateTime (see page 26518)	This is FDateTime, a member of class TXSCustomDateTime.

199.1.4.4.1 TXSCustomDateTime.FDateTime Field

Pascal

```
FDateTime: WideString;
```

C++

```
BSTR FDateTime;
```

Description

This is FDateTime, a member of class TXSCustomDateTime.

199.1.4.5 TXSCustomDateTime Methods

The methods of the TXSCustomDateTime class are listed here.

Protected Methods

	Name	Description
	GetAsDateTime (see page 26519)	Returns the value of the AsDateTime (see page 26527) property.
	GetAsUTCDateTime (see page 26519)	This is GetAsUTCDateTime, a member of class TXSCustomDateTime.
	GetDay (see page 26519)	Returns the value of the Day (see page 26528) property.
	GetFractionalSeconds (see page 26520)	Returns the value of the FractionalSeconds (see page 26529) property.
	GetHour (see page 26520)	Returns the value of the Hour (see page 26529) property.
	GetHourOffset (see page 26520)	Returns the value of the HourOffset (see page 26530) property.
	GetMillisecond (see page 26521)	Returns the value of the Millisecond (see page 26530) property.
	GetMinute (see page 26521)	Returns the value of the Minute (see page 26531) property.
	GetMinuteOffset (see page 26521)	Returns the value of the MinuteOffset (see page 26531) property.
	GetMonth (see page 26521)	Returns the value of the Month (see page 26532) property.

	GetSecond (see page 26522)	Returns the value of the Second (see page 26532) property.
	GetUseZeroMilliseconds (see page 26522)	Returns the value of the UseZeroMilliseconds (see page 26533) property.
	GetYear (see page 26522)	Returns the value of the Year (see page 26533) property.
	SetAsDateTime (see page 26523)	Sets the value of the AsDateTime (see page 26527) property.
	SetAsUTCDateTime (see page 26523)	Sets the value of the AsUTCDateTime (see page 26528) property.
V	SetDay (see page 26523)	Sets the value of the Day (see page 26528) property.
	SetFractionalSeconds (see page 26524)	Sets the value of the FractionalSeconds (see page 26529) property.
V	SetHour (see page 26524)	Sets the value of the Hour (see page 26529) property.
V	SetHourOffset (see page 26524)	Sets the value of the HourOffset (see page 26530) property.
V	SetMillisecond (see page 26524)	Sets the value of the Millisecond (see page 26530) property.
V	SetMinute (see page 26525)	Sets the value of the Minute (see page 26531) property.
V	SetMinuteOffset (see page 26525)	Sets the value of the MinuteOffset (see page 26531) property.
V	SetMonth (see page 26525)	Sets the value of the Month (see page 26532) property.
V	SetSecond (see page 26526)	Sets the value of the Second (see page 26532) property.
	SetUseZeroMilliseconds (see page 26526)	Sets the value of the UseZeroMilliseconds (see page 26533) property.
V	SetYear (see page 26526)	Sets the value of the Year (see page 26533) property.

199.1.4.5.1 TXSCustomDateTime.GetAsDateTime Method

Returns the value of the AsDateTime (see page 26527) property.

Pascal

```
function GetAsDateTime: TDateTime;
```

C++

```
__fastcall TDateTime GetAsDateTime();
```

Description

GetAsDateTime is the protected read implementation of the AsDateTime (see page 26527) property.

See Also

AsDateTime (see page 26527)

199.1.4.5.2 TXSCustomDateTime.GetAsUTCDateTime Method

Pascal

```
function GetAsUTCDateTime: TDateTime;
```

C++

```
__fastcall TDateTime GetAsUTCDateTime();
```

Description

This is GetAsUTCDateTime, a member of class TXSCustomDateTime.

199.1.4.5.3 TXSCustomDateTime.GetDay Method

Returns the value of the Day (see page 26528) property.

Pascal

```
function GetDay: Word;
```

C++

```
__fastcall Word GetDay();
```

Description

GetDay is the protected read implementation of the Day (see page 26528) property.

See Also

Day (see page 26528)

199.1.4.5.4 TXSCustomDateTime.GetFractionalSeconds Method

Returns the value of the FractionalSeconds (see page 26529) property.

Pascal

```
function GetFractionalSeconds: Double;
```

C++

```
__fastcall Double GetFractionalSeconds();
```

Description

GetFractionalSeconds is the protected read implementation of the FractionalSeconds (see page 26529) property.

See Also

FractionalSeconds (see page 26529)

199.1.4.5.5 TXSCustomDateTime.GetHour Method

Returns the value of the Hour (see page 26529) property.

Pascal

```
function GetHour: Word;
```

C++

```
__fastcall Word GetHour();
```

Description

GetHour is the protected read implementation of the Hour (see page 26529) property.

See Also

Hour (see page 26529)

199.1.4.5.6 TXSCustomDateTime.GetHourOffset Method

Returns the value of the HourOffset (see page 26530) property.

Pascal

```
function GetHourOffset: SmallInt;
```

C++

```
__fastcall SmallInt GetHourOffset();
```

Description

GetHourOffset is the protected read implementation of the HourOffset (see page 26530) property.

See Also

HourOffset (see page 26530)

199.1.4.5.7 TXSCustomDateTime.GetMillisecond Method

Returns the value of the Millisecond (see page 26530) property.

Pascal

```
function GetMillisecond: Word;
```

C++

```
__fastcall Word GetMillisecond();
```

Description

GetMillisecond is the protected read implementation of the Millisecond (see page 26530) property.

See Also

Millisecond (see page 26530)

199.1.4.5.8 TXSCustomDateTime.GetMinute Method

Returns the value of the Minute (see page 26531) property.

Pascal

```
function GetMinute: Word;
```

C++

```
__fastcall Word GetMinute();
```

Description

GetMinute is the protected read implementation of the Minute (see page 26531) property.

See Also

Minute (see page 26531)

199.1.4.5.9 TXSCustomDateTime.GetMinuteOffset Method

Returns the value of the MinuteOffset (see page 26531) property.

Pascal

```
function GetMinuteOffset: SmallInt;
```

C++

```
__fastcall SmallInt GetMinuteOffset();
```

Description

GetMinuteOffset is the protected read implementation of the MinuteOffset (see page 26531) property.

See Also

MinuteOffset (see page 26531)

199.1.4.5.10 TXSCustomDateTime.GetMonth Method

Returns the value of the Month (see page 26532) property.

Pascal

```
function GetMonth: Word;
```

C++

```
__fastcall Word GetMonth();
```

Description

GetMonth is the protected read implementation of the Month (see page 26532) property.

See Also

Month (see page 26532)

199.1.4.5.11 TXSCustomDateTime.GetSecond Method

Returns the value of the Second (see page 26532) property.

Pascal

```
function GetSecond: Word;
```

C++

```
__fastcall Word GetSecond();
```

Description

GetSecond is the protected read implementation of the Second (see page 26532) property.

See Also

Second (see page 26532)

199.1.4.5.12 TXSCustomDateTime.GetUseZeroMilliseconds Method

Returns the value of the UseZeroMilliseconds (see page 26533) property.

Pascal

```
function GetUseZeroMilliseconds: Boolean;
```

C++

```
__fastcall Boolean GetUseZeroMilliseconds();
```

Description

GetUseZeroMilliseconds is the protected read implementation of the UseZeroMilliseconds (see page 26533) property.

See Also

UseZeroMilliseconds (see page 26533)

199.1.4.5.13 TXSCustomDateTime.GetYear Method

Returns the value of the Year (see page 26533) property.

Pascal

```
function GetYear: Integer;
```

C++

```
__fastcall int GetYear();
```

Description

GetYear is the protected read implementation of the Year (see page 26533) property.

See Also

Year (see page 26533)

199.1.4.5.14 TXSCustomDateTime.SetAsDateTime Method

Sets the value of the AsDateTime (see page 26527) property.

Pascal

```
procedure SetAsDateTime(const Value: TDateTime);
```

C++

```
__fastcall SetAsDateTime(const TDateTime Value);
```

Description

SetAsDateTime is the protected write implementation of the AsDateTime (see page 26527) property.

See Also

AsDateTime (see page 26527)

199.1.4.5.15 TXSCustomDateTime.SetAsUTCDateTime Method

Sets the value of the AsUTCDateTime (see page 26528) property.

Pascal

```
procedure SetAsUTCDateTime(const Value: TDateTime);
```

C++

```
__fastcall SetAsUTCDateTime(const TDateTime Value);
```

Description

SetAsUTCDateTime is the protected write implementation of the AsUTCDateTime (see page 26528) property.

See Also

AsUTCDateTime (see page 26528)

199.1.4.5.16 TXSCustomDateTime.SetDay Method

Sets the value of the Day (see page 26528) property.

Pascal

```
procedure SetDay(const Value: Word); virtual;
```

C++

```
virtual __fastcall SetDay(const Word Value);
```

Description

SetDay is the protected write implementation of the Day (see page 26528) property.

See Also

Day (see page 26528)

199.1.4.5.17 TXSCustomDateTime.SetFractionalSeconds Method

Sets the value of the FractionalSeconds (see page 26529) property.

Pascal

```
procedure SetFractionalSeconds(const Value: Double);
```

C++

```
__fastcall SetFractionalSeconds(const Double Value);
```

Description

SetDay (see page 26523) is the protected write implementation of the FractionalSeconds (see page 26529) property.

Value is the fraction of a second. It is a value between 0 and 1.

See Also

FractionalSeconds (see page 26529)

199.1.4.5.18 TXSCustomDateTime.SetHour Method

Sets the value of the Hour (see page 26529) property.

Pascal

```
procedure SetHour(const Value: Word); virtual;
```

C++

```
virtual __fastcall SetHour(const Word Value);
```

Description

SetHour is the protected write implementation of the Hour (see page 26529) property.

See Also

Hour (see page 26529)

199.1.4.5.19 TXSCustomDateTime.SetHourOffset Method

Sets the value of the HourOffset (see page 26530) property.

Pascal

```
procedure SetHourOffset(const Value: SmallInt); virtual;
```

C++

```
virtual __fastcall SetHourOffset(const SmallInt Value);
```

Description

SetHourOffset is the protected write implementation of the HourOffset (see page 26530) property.

See Also

HourOffset (see page 26530)

199.1.4.5.20 TXSCustomDateTime.SetMillisecond Method

Sets the value of the Millisecond (see page 26530) property.

Pascal

```
procedure SetMillisecond(const Value: Word); virtual;
```

C++

```
virtual __fastcall SetMillisecond(const Word Value);
```

Description

SetMillisecond is the protected write implementation for the Millisecond (see page 26530) property.

See Also

Millisecond (see page 26530)

199.1.4.5.21 TXSCustomDateTime.SetMinute Method

Sets the value of the Minute (see page 26531) property.

Pascal

```
procedure SetMinute(const Value: Word); virtual;
```

C++

```
virtual __fastcall SetMinute(const Word Value);
```

Description

SetMinute is the protected write implementation of the Minute (see page 26531) property.

See Also

Minute (see page 26531)

199.1.4.5.22 TXSCustomDateTime.SetMinuteOffset Method

Sets the value of the MinuteOffset (see page 26531) property.

Pascal

```
procedure SetMinuteOffset(const Value: SmallInt); virtual;
```

C++

```
virtual __fastcall SetMinuteOffset(const SmallInt Value);
```

Description

SetMinuteOffest is the protected write implementation of the MinuteOffset (see page 26531) property.

See Also

MinuteOffset (see page 26531)

199.1.4.5.23 TXSCustomDateTime.SetMonth Method

Sets the value of the Month (see page 26532) property.

Pascal

```
procedure SetMonth(const Value: Word); virtual;
```

C++

```
virtual __fastcall SetMonth(const Word Value);
```

Description

SetMonth is the protected write implementation of the Month (see page 26532) property.

See Also

Month (see page 26532)

199.1.4.5.24 TXSCustomDateTime.SetSecond Method

Sets the value of the Second (see page 26532) property.

Pascal

```
procedure SetSecond(const Value: Word); virtual;
```

C++

```
virtual __fastcall SetSecond(const Word Value);
```

Description

SetSecond is the protected write implementation of the Second (see page 26532) property.

See Also

Second (see page 26532)

199.1.4.5.25 TXSCustomDateTime.SetUseZeroMilliseconds Method

Sets the value of the UseZeroMilliseconds (see page 26533) property.

Pascal

```
procedure SetUseZeroMilliseconds(const Value: Boolean);
```

C++

```
__fastcall SetUseZeroMilliseconds(const Boolean Value);
```

Description

SetUseZeroMilliseconds is the protected write implementation of the UseZeroMilliseconds (see page 26533) property.

See Also

UseZeroMilliseconds (see page 26533)

199.1.4.5.26 TXSCustomDateTime.SetYear Method

Sets the value of the Year (see page 26533) property.

Pascal

```
procedure SetYear(const Value: Integer); virtual;
```

C++

```
virtual __fastcall SetYear(const int Value);
```

Description

SetYear is the protected write implementation of the Year (see page 26533) property.

See Also

Year (see page 26533)

199.1.4.6 TXSCustomDateTime Properties

The properties of the TXSCustomDateTime class are listed here.

Protected Properties

	Name	Description
	Millisecond (see page 26530)	Indicates the Millisecond of the second that the object represents.

Public Properties

	Name	Description
	AsDateTime (see page 26527)	Represents the object's value as a TDateTime value.
	AsUTCDateTime (see page 26528)	Represents the object's value as a TDateTime value using UTC time.
	Day (see page 26528)	Indicates the Day of the month that the object represents.
	FractionalSeconds (see page 26529)	Indicates any factional portion of a second beyond the seconds specified in the Second (see page 26532) property.
	Hour (see page 26529)	Indicates the hour of the day that the object represents.
	HourOffset (see page 26530)	Indicates the difference between the Hour (see page 26529) property and the hour in the local time zone.
	Minute (see page 26531)	Indicates the minute of the hour that the object represents.
	MinuteOffset (see page 26531)	Indicates the difference between the Minute (see page 26531) property and the minute of the hour in local time.
	Month (see page 26532)	Indicates the month of the year that the object represents.
	Second (see page 26532)	Indicates the second of the minute that the object represents.
	UseZeroMilliseconds (see page 26533)	Indicates whether the string representation of this object's value includes a portion for milliseconds when the value of Millisecond (see page 26530) is zero.
	Year (see page 26533)	Indicates the year that the object represents.

199.1.4.6.1 TXSCustomDateTime.AsDateTime Property

Represents the object's value as a TDateTime value.

Pascal

```
property AsDateTime: TDateTime;
```

C++

```
__property TDateTime AsDateTime;
```

Description

Use AsDateTime to read the value of the TXSCustomDateTime (see page 26515) object into a variable of type TDateTime or set its value from a TDateTime value. AsDateTime handles the mapping between a TDateTime value and the Year (see page 26533), Month (see page 26532), Day (see page 26528), Hour (see page 26529), Minute (see page 26531), and Second (see page 26532) values.

Note: AsDateTime represents the local date-and-time, without using Coordinated Universal Time (UTC). That is, AsDateTime ignores the HourOffset (see page 26530) and MinuteOffset (see page 26531) properties. To get or set the value taking the HourOffset (see page 26530) and MinuteOffset (see page 26531) properties into account, use the AsUTCDateTime (see page 26528) property instead.

See Also

Day (see page 26528)

Hour (see page 26529)

Minute (see page 26531)

Month (see page 26532)

Second (see page 26532)

Year (see page 26533)

HourOffset (see page 26530)

MinuteOffset (see page 26531)

Millisecond (see page 26530)

AsUTCDateTime (see page 26528)

199.1.4.6.2 TXSCustomDateTime.AsUTCDateTime Property

Represents the object's value as a TDateTime value using UTC time.

Pascal

```
property AsUTCDateTime: TDateTime;
```

C++

```
__property TDateTime AsUTCDateTime;
```

Description

Use AsUTCDateTime to read the value of the TXSCustomDateTime (see page 26515) object into a variable of type TDateTime or set its value from a TDateTime value. AsUTCDateTime handles the mapping between a TDateTime value and the Year (see page 26533), Month (see page 26532), Day (see page 26528), Hour (see page 26529), Minute (see page 26531), and Second (see page 26532) values. It also makes an adjustment for the HourOffset (see page 26530) and MinuteOffset (see page 26531) properties, so that the value of AsUTCDateTime is a Coordinated Universal Time (UTC) value rather than a local time value.

Note: To get or set the value as a local time value ignoring the HourOffset (see page 26530) and MinuteOffset (see page 26531) properties, use the AsDateTime (see page 26527) property instead.

See Also

Day (see page 26528)

Hour (see page 26529)

Minute (see page 26531)

Month (see page 26532)

Second (see page 26532)

Year (see page 26533)

HourOffset (see page 26530)

MinuteOffset (see page 26531)

Millisecond (see page 26530)

AsDateTime (see page 26527)

199.1.4.6.3 TXSCustomDateTime.Day Property

Indicates the Day of the month that the object represents.

Pascal

```
property Day: Word;
```

C++

```
__property Word Day;
```

Description

Use Day to get or set the day portion of the date-and-time value. Valid Day values are 1 through 28, 29, 30, or 31, depending on the Month (see page 26532) value.

See Also

AsDateTime (see page 26527)

Hour (see page 26529)

Minute (see page 26531)

Month (see page 26532)

Second (see page 26532)

Year (see page 26533)

Millisecond (see page 26530)

199.1.4.6.4 TXSCustomDateTime.FractionalSeconds Property

Indicates any factional portion of a second beyond the seconds specified in the Second (see page 26532) property.

Pascal

```
property FractionalSeconds: Double;
```

C++

```
__property Double FractionalSeconds;
```

Description

Use FractionalSeconds to get or set the fractional seconds of the time value. FractionalSeconds represents the same information as the Millisecond (see page 26530) property, but can provide greater precision.

See Also

AsDateTime (see page 26527)

Minute (see page 26531)

Second (see page 26532)

Millisecond (see page 26530)

UseZeroMilliseconds (see page 26533)

199.1.4.6.5 TXSCustomDateTime.Hour Property

Indicates the hour of the day that the object represents.

Pascal

```
property Hour: Word;
```

C++

```
__property Word Hour;
```

Description

Use Hour to get or set the hour portion of the date-and-time value. Valid Hour values are 0 through 23.

Note: Hour represents the hour using Coordinated Universal Time (UTC). To convert the hour to local time, you must add the hour offset.

See Also

AsDateTime (see page 26527)

Day (see page 26528)

Minute (see page 26531)

Month (see page 26532)

Second (see page 26532)

Year (see page 26533)

HourOffset (see page 26530)

Millisecond (see page 26530)

199.1.4.6.6 TXSCustomDateTime.HourOffset Property

Indicates the difference between the Hour (see page 26529) property and the hour in the local time zone.

Pascal

```
property HourOffset: SmallInt;
```

C++

```
__property SmallInt HourOffset;
```

Description

The Hour (see page 26529) property represents the hour of the day using Coordinated Universal Time (UTC). HourOffset is the correction factor you can add to Hour (see page 26529) to obtain the hour of the day in local time.

See Also

Hour (see page 26529)

199.1.4.6.7 TXSCustomDateTime.Millisecond Property

Indicates the Millisecond of the second that the object represents.

Pascal

```
property Millisecond: Word;
```

C++

```
__property Word Millisecond;
```

Description

Use Millisecond to get or set the millisecond portion of the date-and-time value. Valid Millisecond values are 1 through 59.

Note: Millisecond values are not used by TXSDuration (see page 26548).

Note: You can also use the FractionalSeconds (see page 26529) property to obtain fractional portions of the seconds. Unlike Millisecond, which gives the value in milliseconds, FractionalSeconds (see page 26529) returns a fraction of a second.

See Also

AsDateTime (see page 26527)

Day (see page 26528)

Hour (see page 26529)

Minute (see page 26531)

Second (see page 26532)

Year (see page 26533)

Month (see page 26532)

UseZeroMilliseconds (see page 26533)

FractionalSeconds (see page 26529)

199.1.4.6.8 TXSCustomDateTime.Minute Property

Indicates the minute of the hour that the object represents.

Pascal

```
property Minute: Word;
```

C++

```
__property Word Minute;
```

Description

Use Minute to get or set the minute portion of the date-and-time value. Valid Minute values are 0 through 59.

Note: Minute represents the minute using Coordinated Universal Time (UTC). To convert the minutes to local time, you must add the minute offset.

See Also

AsDateTime (see page 26527)

Day (see page 26528)

Hour (see page 26529)

Month (see page 26532)

Second (see page 26532)

Year (see page 26533)

MinuteOffset (see page 26531)

Millisecond (see page 26530)

199.1.4.6.9 TXSCustomDateTime.MinuteOffset Property

Indicates the difference between the Minute (see page 26531) property and the minute of the hour in local time.

Pascal

```
property MinuteOffset: SmallInt;
```

C++

```
__property SmallInt MinuteOffset;
```

Description

The Minute (see page 26531) property represents the minute of the hour using Coordinated Universal Time (UTC). MinuteOffset is the correction factor you can add to Minute (see page 26531) to obtain the minute of the hour in local time.

See Also

Minute (see page 26531)

199.1.4.6.10 TXSCustomDateTime.Month Property

Indicates the month of the year that the object represents.

Pascal

```
property Month: Word;
```

C++

```
__property Word Month;
```

Description

Use Month to get or set the month portion of the date-and-time value. Valid Month values are 1 through 12.

See Also

AsDateTime (see page 26527)

Day (see page 26528)

Hour (see page 26529)

Minute (see page 26531)

Second (see page 26532)

Year (see page 26533)

Millisecond (see page 26530)

199.1.4.6.11 TXSCustomDateTime.Second Property

Indicates the second of the minute that the object represents.

Pascal

```
property Second: Word;
```

C++

```
__property Word Second;
```

Description

Use Second to get or set the seconds portion of the date-and-time value. Valid Second values are 0 through 59.

See Also

AsDateTime (see page 26527)

Day (see page 26528)

Hour (see page 26529)

Minute (see page 26531)

Second

Year (see page 26533)

Millisecond (see page 26530)

199.1.4.6.12 TXSCustomDateTime.UseZeroMilliseconds Property

Indicates whether the string representation of this object's value includes a portion for milliseconds when the value of Millisecond (see page 26530) is zero.

Pascal

```
property UseZeroMilliseconds: Boolean;
```

C++

```
__property Boolean UseZeroMilliseconds;
```

Description

By default, the NativeToXS method does not include the milliseconds in the string encoding of this object's value when the value of the Millisecond (see page 26530) property is zero. If you set UseZeroMilliseconds, NativeToXS changes its encoding to include the milliseconds even when the value is zero. This is necessary for some Web Services that do not support the full schema defined by W3C.

See Also

Millisecond (see page 26530)

NativeToXS (see page 15581)

199.1.4.6.13 TXSCustomDateTime.Year Property

Indicates the year that the object represents.

Pascal

```
property Year: Integer;
```

C++

```
__property int Year;
```

Description

Use Year to get or set the year portion of the date-and-time value. Valid Year values are 1 through 9999.

See Also

AsDateTime (see page 26527)

Day (see page 26528)

Hour (see page 26529)

Minute (see page 26531)

Second (see page 26532)

Month (see page 26532)

Millisecond (see page 26530)

199.1.5 TXSDate Class

TXSDate implements the XML schema date type so that it can be used on invokable interfaces.

Class Hierarchy

Pascal

```
TXSDate = class(TRemotableXS);
```

C++

```
class TXSDate : public TRemotableXS;
```

File

XSBuiltIns

Description

TXSDate acts as a wrapper for the simple scalar type date that W3C defines for XML schemas. Because this type does not correspond directly to a Delphi or C++ type, it requires special marshaling when used in an invokable interface. TXSDate enables this marshaling because it is a descendant of TRemotable.

To convert a TXSDate value to a TDateTime value, use the AsDate (see page 26539) property. Other properties give the Year (see page 26540), Month (see page 26540), and Day (see page 26539) that make up that value.

On server applications, instances of TXSDate that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSDate instance is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSDate instances that it passes as input parameters, and for freeing any TXSDate instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 22398)

TXSDate

TXSDuration (see page 26548)

TXSTime (see page 26558)

199.1.5.1 TXSDate Members

The following tables list the members exposed by TXSDate.

Protected Fields

	Name	Description
	FSerializationOptions (see page 15559)	This is FSerializationOptions, a member of class TRemotable.

Protected Methods

	Name	Description
	GetAsDate (see page 26536)	This is GetAsDate, a member of class TXSDate.
	GetDay (see page 26537)	This is GetDay, a member of class TXSDate.
	GetMonth (see page 26537)	This is GetMonth, a member of class TXSDate.
	GetYear (see page 26537)	This is GetYear, a member of class TXSDate.
	SetAsDate (see page 26538)	This is SetAsDate, a member of class TXSDate.
	SetDay (see page 26538)	This is SetDay, a member of class TXSDate.
	SetMonth (see page 26538)	This is SetMonth, a member of class TXSDate.
	SetYear (see page 26538)	This is SetYear, a member of class TXSDate.

Public Methods

	Name	Description
	Clone (see page 26536)	Returns a new TXSDate (see page 26534) object that represents the same value as this one.
V	Create (see page 26535)	Creates and initializes a Create object.
V	Destroy (see page 15558)	Frees the memory associated with the TRemotable (see page 15556) object.
V	NativeToXS (see page 26537)	Returns a string that represents the value of a TXSDate (see page 26534) instance.
V	ObjectToSOAP (see page 15559)	Converts this remotable object to its SOAP representation.
V	ObjectToSOAP (see page 15582)	Converts this remotable object to its SOAP representation.
V	SOAPToObject (see page 15560)	Initializes this TRemotable (see page 15556) instance to represent the value of its SOAP representation.
V	SOAPToObject (see page 15582)	Initializes this TRemotableXS (see page 15580) instance to represent the value of its SOAP representation.
V	XSToNative (see page 26538)	Assigns the properties of the TXSDate (see page 26534) instance from the string representation of its value.

Public Properties

	Name	Description
	AsDate (see page 26539)	Represents the object's value as a TDateTime value.
	DataContext (see page 15561)	Specifies the object that controls the lifetime of a TRemotable (see page 15556) instance.
	Day (see page 26539)	Indicates the Day of the month that the object represents.
	Month (see page 26540)	Indicates the month of the year that the object represents.
R	SerializationOptions (see page 15561)	Indicates how the value of the TRemotable (see page 15556) descendant encodes its value.
	Year (see page 26540)	Indicates the year that the object represents.

199.1.5.2 TXSDate.Create Constructor

Creates and initializes a Create object.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TXSDate();
```

Description

Call Create indirectly, using the new keyword, to create a Create object at runtime.

199.1.5.3 TXSDate Methods

The methods of the TXSDate class are listed here.

Protected Methods

	Name	Description
	GetAsDate (see page 26536)	This is GetAsDate, a member of class TXSDate.
	GetDay (see page 26537)	This is GetDay, a member of class TXSDate.
	GetMonth (see page 26537)	This is GetMonth, a member of class TXSDate.
	GetYear (see page 26537)	This is GetYear, a member of class TXSDate.
	SetAsDate (see page 26538)	This is SetAsDate, a member of class TXSDate.
	SetDay (see page 26538)	This is SetDay, a member of class TXSDate.
	SetMonth (see page 26538)	This is SetMonth, a member of class TXSDate.
	SetYear (see page 26538)	This is SetYear, a member of class TXSDate.

Public Methods

	Name	Description
	Clone (see page 26536)	Returns a new TXSDate (see page 26534) object that represents the same value as this one.
V	NativeToXS (see page 26537)	Returns a string that represents the value of a TXSDate (see page 26534) instance.
V	XSToNative (see page 26538)	Assigns the properties of the TXSDate (see page 26534) instance from the string representation of its value.

199.1.5.3.1 TXSDate.Clone Method

Returns a new TXSDate (see page 26534) object that represents the same value as this one.

Pascal

```
function Clone: TXSDate;
```

C++

```
__fastcall TXSDate Clone();
```

Description

Call Clone to obtain a new TXSDate (see page 26534) instance that represents the same time as this one. Clone returns the new TXSDate (see page 26534) object.

199.1.5.3.2 TXSDate.GetAsDate Method

Pascal

```
function GetAsDate: TDateTime;
```

C++

```
__fastcall TDateTime GetAsDate();
```

Description

This is GetAsDate, a member of class TXSDate.

199.1.5.3.3 TXSDate.GetDay Method

Pascal

```
function GetDay: Word;
```

C++

```
__fastcall Word GetDay();
```

Description

This is GetDay, a member of class TXSDate.

199.1.5.3.4 TXSDate.GetMonth Method

Pascal

```
function GetMonth: Word;
```

C++

```
__fastcall Word GetMonth();
```

Description

This is GetMonth, a member of class TXSDate.

199.1.5.3.5 TXSDate.GetYear Method

Pascal

```
function GetYear: Integer;
```

C++

```
__fastcall int GetYear();
```

Description

This is GetYear, a member of class TXSDate.

199.1.5.3.6 TXSDate.NativeToXS Method

Returns a string that represents the value of a TXSDate (see page 26534) instance.

Pascal

```
function NativeToXS: WideString; override;
```

C++

```
virtual __fastcall BSTR NativeToXS();
```

Description

NativeToXS returns the string that represents the value of this TXSDate (see page 26534) in a SOAP packet.

See Also

XSToNative (see page 26538)

199.1.5.3.7 TXSDate.SetAsDate Method

Pascal

```
procedure SetAsDate(const Value: TDateTime);
```

C++

```
__fastcall SetAsDate(const TDateTime Value);
```

Description

This is SetAsDate, a member of class TXSDate.

199.1.5.3.8 TXSDate.SetDay Method

Pascal

```
procedure SetDay(const Value: Word);
```

C++

```
__fastcall SetDay(const Word Value);
```

Description

This is SetDay, a member of class TXSDate.

199.1.5.3.9 TXSDate.SetMonth Method

Pascal

```
procedure SetMonth(const Value: Word);
```

C++

```
__fastcall SetMonth(const Word Value);
```

Description

This is SetMonth, a member of class TXSDate.

199.1.5.3.10 TXSDate.SetYear Method

Pascal

```
procedure SetYear(const Value: Integer);
```

C++

```
__fastcall SetYear(const int Value);
```

Description

This is SetYear, a member of class TXSDate.

199.1.5.3.11 TXSDate.XSToNative Method

Assigns the properties of the TXSDate (see page 26534) instance from the string representation of its value.

Pascal

```
procedure XSToNative(Value: WideString); override;
```

C++

```
virtual __fastcall XSToNative(BSTR Value);
```

Description

XSToNative parses the string specified by Data, which is the SOAP encoding of a date value. It uses this information to set the properties of the TXSDate (see page 26534) object.

See Also

NativeToXS (see page 26537)

199.1.5.4 TXSDate Properties

The properties of the TXSDate class are listed here.

Public Properties

	Name	Description
	AsDate (see page 26539)	Represents the object's value as a TDateTime value.
	Day (see page 26539)	Indicates the Day of the month that the object represents.
	Month (see page 26540)	Indicates the month of the year that the object represents.
	Year (see page 26540)	Indicates the year that the object represents.

199.1.5.4.1 TXSDate.AsDate Property

Represents the object's value as a TDateTime value.

Pascal

```
property AsDate: TDateTime;
```

C++

```
__property TDateTime AsDate;
```

Description

Use AsDate to read the value of the TXSDate (see page 26534) object into a variable of type TDateTime or set this objects value from a TDateTime value. AsDate handles the mapping between a TDateTime value and the Year (see page 26540), Month (see page 26540), and Day (see page 26539) properties. When reading AsDate, the resulting TDateTime value has a time portion of 0. When setting AsDate, the time portion is ignored.

See Also

Day (see page 26539)

Month (see page 26540)

Year (see page 26540)

199.1.5.4.2 TXSDate.Day Property

Indicates the Day of the month that the object represents.

Pascal

```
property Day: Word;
```

C++

```
__property Word Day;
```

Description

Use Day to get or set the day portion of the date-and-time value. Valid Day values are 1 through 28, 29, 30, or 31, depending on the Month (see page 26540) value.

See Also

AsDate (see page 26539)

Month (see page 26540)

Year (see page 26540)

199.1.5.4.3 TXSDate.Month Property

Indicates the month of the year that the object represents.

Pascal

```
property Month: Word;
```

C++

```
__property Word Month;
```

Description

Use Month to get or set the month portion of the date-and-time value. Valid Month values are 1 through 12.

See Also

AsDate (see page 26539)

Day (see page 26539)

Year (see page 26540)

199.1.5.4.4 TXSDate.Year Property

Indicates the year that the object represents.

Pascal

```
property Year: Integer;
```

C++

```
__property int Year;
```

Description

Use Year to get or set the year portion of the date-and-time value. Valid Year values are 1 through 9999.

See Also

AsDate (see page 26539)

Day (see page 26539)

Month (see page 26540)

199.1.6 TXSDateTime Class

TXSDateTime implements the XML schema datetime type so that it can be used on invokable interfaces.

Class Hierarchy

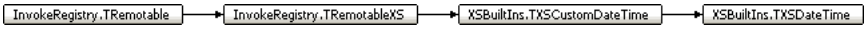

Pascal

```
TXSDateTime = class(TXSCustomDateTime);
```

C++

```
class TXSDateTime : public TXSCustomDateTime;
```

File

XSBuiltIns

Description

TXSDateTime acts as a wrapper for the simple scalar type datetime that W3C defines for XML schemas. Because this type does not correspond directly to a Delphi or C++ type, it requires special marshaling when used in an invokable interface. TXSDateTime enables this marshaling because it is a descendant of TRemotable.

To convert a TXSDateTime value to a TDateTime value, use the AsDateTime (see page 26527) property. The resulting value represents the time portion using Coordinated Universal Time (UTC). To convert that to a local time zone, adjust the hour and minute portions using the HourOffset (see page 26530) and MinuteOffset (see page 26531) properties.

On server applications, instances of TXSDateTime that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSDateTime instance is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSDateTime instances that it passes as input parameters, and for freeing any TXSDateTime instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 22398)

TXSDate (see page 26534)

TXSDuration (see page 26548)

TXSTime (see page 26558)

199.1.6.1 TXSDateTime Members

The following tables list the members exposed by TXSDateTime.

Protected Fields

	Name	Description
	FDateTime (see page 26518)	This is FDateTime, a member of class TXSCustomDateTime.
	FSerializationOptions (see page 15559)	This is FSerializationOptions, a member of class TRemotable.

Protected Methods

	Name	Description
	GetAsDateTime (see page 26519)	Returns the value of the AsDateTime (see page 26527) property.
	GetAsUTCDateTime (see page 26519)	This is GetAsUTCDateTime, a member of class TXSCustomDateTime.
	GetDay (see page 26519)	Returns the value of the Day (see page 26528) property.
	GetFractionalSeconds (see page 26520)	Returns the value of the FractionalSeconds (see page 26529) property.
	GetHour (see page 26520)	Returns the value of the Hour (see page 26529) property.
	GetHourOffset (see page 26520)	Returns the value of the HourOffset (see page 26530) property.
	GetMillisecond (see page 26521)	Returns the value of the Millisecond (see page 26530) property.
	GetMinute (see page 26521)	Returns the value of the Minute (see page 26531) property.
	GetMinuteOffset (see page 26521)	Returns the value of the MinuteOffset (see page 26531) property.
	GetMonth (see page 26521)	Returns the value of the Month (see page 26532) property.
	GetSecond (see page 26522)	Returns the value of the Second (see page 26532) property.
	GetUseZeroMilliseconds (see page 26522)	Returns the value of the UseZeroMilliseconds (see page 26533) property.
	GetYear (see page 26522)	Returns the value of the Year (see page 26533) property.
	SetAsDateTime (see page 26523)	Sets the value of the AsDateTime (see page 26527) property.
	SetAsUTCDateTime (see page 26523)	Sets the value of the AsUTCDateTime (see page 26528) property.
	SetDay (see page 26523)	Sets the value of the Day (see page 26528) property.
	SetFractionalSeconds (see page 26524)	Sets the value of the FractionalSeconds (see page 26529) property.
	SetHour (see page 26524)	Sets the value of the Hour (see page 26529) property.
	SetHourOffset (see page 26524)	Sets the value of the HourOffset (see page 26530) property.
	SetMillisecond (see page 26524)	Sets the value of the Millisecond (see page 26530) property.
	SetMinute (see page 26525)	Sets the value of the Minute (see page 26531) property.
	SetMinuteOffset (see page 26525)	Sets the value of the MinuteOffset (see page 26531) property.
	SetMonth (see page 26525)	Sets the value of the Month (see page 26532) property.
	SetSecond (see page 26526)	Sets the value of the Second (see page 26532) property.
	SetUseZeroMilliseconds (see page 26526)	Sets the value of the UseZeroMilliseconds (see page 26533) property.
	SetYear (see page 26526)	Sets the value of the Year (see page 26533) property.

Public Methods

	Name	Description
	Clone (see page 26543)	Returns a new TXSDateTime (see page 26541) object that represents the same value as this one.
	CompareDateTimeParam (see page 26543)	This is CompareDateTimeParam, a member of class TXSDateTime.
	Create (see page 15558)	Creates and initializes a TRemotable (see page 15556) object.
	Create (see page 26517)	Creates and initializes a TXSCustomDateTime (see page 26515) object.
	Destroy (see page 15558)	Frees the memory associated with the TRemotable (see page 15556) object.
	Destroy (see page 26517)	Frees the memory associated with the TXSCustomDateTime (see page 26515) object.
	NativeToXS (see page 26544)	Returns a string that represents the value of a TXSDateTime (see page 26541) instance.
	ObjectToSOAP (see page 15559)	Converts this remotable object to its SOAP representation.
	ObjectToSOAP (see page 15582)	Converts this remotable object to its SOAP representation.
	SOAPToObject (see page 15560)	Initializes this TRemotable (see page 15556) instance to represent the value of its SOAP representation.
	SOAPToObject (see page 15582)	Initializes this TRemotableXS (see page 15580) instance to represent the value of its SOAP representation.
	XSToNative (see page 26544)	Assigns the properties of the TXSDateTime (see page 26541) instance from the string representation of its value.

Public Properties

	Name	Description
	AsDateTime (see page 26527)	Represents the object's value as a TDateTime value.
	AsUTCDateTime (see page 26528)	Represents the object's value as a TDateTime value using UTC time.
	DataContext (see page 15561)	Specifies the object that controls the lifetime of a TRemotable (see page 15556) instance.

	Day (see page 26528)	Indicates the Day of the month that the object represents.
	FractionalSeconds (see page 26529)	Indicates any factional portion of a second beyond the seconds specified in the Second (see page 26532) property.
	Hour (see page 26529)	Indicates the hour of the day that the object represents.
	HourOffset (see page 26530)	Indicates the difference between the Hour (see page 26529) property and the hour in the local time zone.
	Millisecond (see page 26544)	Indicates the Millisecond of the second that the object represents.
	Minute (see page 26531)	Indicates the minute of the hour that the object represents.
	MinuteOffset (see page 26531)	Indicates the difference between the Minute (see page 26531) property and the minute of the hour in local time.
	Month (see page 26532)	Indicates the month of the year that the object represents.
	Second (see page 26532)	Indicates the second of the minute that the object represents.
R	SerializationOptions (see page 15561)	Indicates how the value of the TRemotable (see page 15556) descendant encodes its value.
	UseZeroMilliseconds (see page 26533)	Indicates whether the string representation of this object's value includes a portion for milliseconds when the value of Millisecond (see page 26530) is zero.
	Year (see page 26533)	Indicates the year that the object represents.

199.1.6.2 TXSDateTime Methods

The methods of the TXSDateTime class are listed here.

Public Methods

	Name	Description
	Clone (see page 26543)	Returns a new TXSDateTime (see page 26541) object that represents the same value as this one.
	CompareDateTimeParam (see page 26543)	This is CompareDateTimeParam, a member of class TXSDateTime.
V	NativeToXS (see page 26544)	Returns a string that represents the value of a TXSDateTime (see page 26541) instance.
V	XSToNative (see page 26544)	Assigns the properties of the TXSDateTime (see page 26541) instance from the string representation of its value.

199.1.6.2.1 TXSDateTime.Clone Method

Returns a new TXSDateTime (see page 26541) object that represents the same value as this one.

Pascal

```
function Clone: TXSDateTime;
```

C++

```
__fastcall TXSDateTime Clone();
```

Description

Call Clone to obtain a new TXSDateTime (see page 26541) instance that represents the same date and time as this one. Clone returns the new TXSDateTime (see page 26541) object.

199.1.6.2.2 TXSDateTime.CompareDateTimeParam Method

Pascal

```
function CompareDateTimeParam(const Value1: TXSDateTime; const Value2: TXSDateTime):
TXSDuration;
```

C++

```
__fastcall TXSDuration CompareDateTimeParam(const TXSDateTime Value1, const TXSDateTime
Value2);
```

Description

This is CompareDateTimeParam, a member of class TXSDateTime.

199.1.6.2.3 TXSDateTime.NativeToXS Method

Returns a string that represents the value of a TXSDateTime (see page 26541) instance.

Pascal

```
function NativeToXS: WideString; override;
```

C++

```
virtual __fastcall BSTR NativeToXS();
```

Description

NativeToXS returns the string that represents the value of this TXSDateTime (see page 26541) in a SOAP packet.

See Also

XSToNative (see page 26544)

UseZeroMilliseconds (see page 26533)

199.1.6.2.4 TXSDateTime.XSToNative Method

Assigns the properties of the TXSDateTime (see page 26541) instance from the string representation of its value.

Pascal

```
procedure XSToNative(Value: WideString); override;
```

C++

```
virtual __fastcall XSToNative(BSTR Value);
```

Description

XSToNative parses the string specified by Data, which is the SOAP encoding of a datetime value. It uses this information to set the properties of the TXSDateTime (see page 26541) object.

See Also

NativeToXS (see page 26544)

199.1.6.3 TXSDateTime Properties

The properties of the TXSDateTime class are listed here.

Public Properties

	Name	Description
	Millisecond (see page 26544)	Indicates the Millisecond of the second that the object represents.

199.1.6.3.1 TXSDateTime.Millisecond Property

Indicates the Millisecond of the second that the object represents.

Pascal

```
property Millisecond: Word;
```

C++

```
__property Word Millisecond;
```

Description

Use Millisecond to get or set the millisecond portion of the date-and-time value. Valid Millisecond values are 1 through 59.

Note: Millisecond values are not used by TXSDuration.

Note: You can also use the FractionalSeconds property to obtain fractional portions of the seconds. Unlike Millisecond, which gives the value in milliseconds, FractionalSeconds returns a fraction of a second.

See Also

AsDateTime (see page 26527)

Day (see page 26528)

Hour (see page 26529)

Minute (see page 26531)

Second (see page 26532)

Year (see page 26533)

Month (see page 26532)

UseZeroMilliseconds (see page 26533)

FractionalSeconds (see page 26529)

199.1.7 TXSDecimal Class

TXSDecimal implements the XML schema decimal type so that it can be used on invokable interfaces.

Class Hierarchy

Pascal

```
TXSDecimal = class(TRemotableXS);
```

C++

```
class TXSDecimal : public TRemotableXS;
```

File

XSBuiltIns

Description

TXSDecimal acts as a wrapper for the simple scalar decimal type that W3C defines for XML schemas. Because this type does not correspond directly to a Delphi or C++ type, it requires special marshaling when used in an invokable interface. TXSDecimal enables this marshaling because it is a descendant of TRemotable.

To convert a TXSDecimal value to a TBcd value, use the AsBcd (see page 26547) property.

On server applications, instances of TXSDecimal that are input parameters on an invokable interface are automatically created

when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSDecimal instance is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSDecimal instances that it passes as input parameters, and for freeing any TXSDecimal instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 22398)

TXSHexBinary (see page 26553)

EXSDecimalException (see page 26514)

199.1.7.1 TXSDecimal Members

The following tables list the members exposed by TXSDecimal.

Protected Fields

	Name	Description
	FSerializationOptions (see page 15559)	This is FSerializationOptions, a member of class TRemotable.

Public Methods

	Name	Description
	Create (see page 15558)	Creates and initializes a TRemotable (see page 15556) object.
	Destroy (see page 15558)	Frees the memory associated with the TRemotable (see page 15556) object.
	NativeToXS (see page 26547)	Returns a string that represents the value of a TXSDecimal (see page 26545) instance.
	ObjectToSOAP (see page 15559)	Converts this remotable object to its SOAP representation.
	ObjectToSOAP (see page 15582)	Converts this remotable object to its SOAP representation.
	SOAPToObject (see page 15560)	Initializes this TRemotable (see page 15556) instance to represent the value of its SOAP representation.
	SOAPToObject (see page 15582)	Initializes this TRemotableXS (see page 15580) instance to represent the value of its SOAP representation.
	XSToNative (see page 26547)	Assigns a string as the value of the TXSDecimal (see page 26545) instance.

Public Properties

	Name	Description
	AsBcd (see page 26547)	Represents the object's value as a TBcd value.
	DataContext (see page 15561)	Specifies the object that controls the lifetime of a TRemotable (see page 15556) instance.
	DecimalString (see page 26548)	Represents the object's value as a string.
	SerializationOptions (see page 15561)	Indicates how the value of the TRemotable (see page 15556) descendant encodes its value.

199.1.7.2 TXSDecimal Methods

The methods of the TXSDecimal class are listed here.

Public Methods

	Name	Description
	NativeToXS (see page 26547)	Returns a string that represents the value of a TXSDecimal (see page 26545) instance.
	XSToNative (see page 26547)	Assigns a string as the value of the TXSDecimal (see page 26545) instance.

199.1.7.2.1 TXSDecimal.NativeToXS Method

Returns a string that represents the value of a TXSDecimal (see page 26545) instance.

Pascal

```
function NativeToXS: WideString; override;
```

C++

```
virtual __fastcall BSTR NativeToXS();
```

Description

NativeToXS returns the string that represents the value of this TXSDecimal (see page 26545) object in a SOAP packet. That is, NativeToXS converts the DecimalString (see page 26548) property to a WideString and returns the result.

See Also

XSToNative (see page 26547)

DecimalString (see page 26548)

199.1.7.2.2 TXSDecimal.XSToNative Method

Assigns a string as the value of the TXSDecimal (see page 26545) instance.

Pascal

```
procedure XSToNative(Value: WideString); override;
```

C++

```
virtual __fastcall XSToNative(BSTR Value);
```

Description

XSToNative sets the value of this TXSDecimal (see page 26545) object to a string that represents a formatted decimal value.

Data is the SOAP encoding of a decimal value. It becomes the value of the DecimalString (see page 26548) property.

See Also

NativeToXS (see page 26547)

DecimalString (see page 26548)

199.1.7.3 TXSDecimal Properties

The properties of the TXSDecimal class are listed here.

Public Properties

	Name	Description
	AsBcd (see page 26547)	Represents the object's value as a TBcd value.
	DecimalString (see page 26548)	Represents the object's value as a string.

199.1.7.3.1 TXSDecimal.AsBcd Property

Represents the object's value as a TBcd value.

Pascal

```
property AsBcd: TBcd;
```

C++

```
__property TBcd AsBcd;
```

Description

Use AsBcd to get or set this object's value as a TBcd type. TXSDecimal (see page 26545) converts between the internal string representation that is the value of the DecimalString (see page 26548) property and a TBcd value when the application gets or sets this property. Thus, if the string representation is ill-formed, reading AsBcd can cause an EXSDecimalException (see page 26514) exception. Similarly, setting AsBcd can cause an EXSDecimalException (see page 26514) exception if the TBcd value has an invalid scale or precision.

See Also

BcdToStr (see page 13639)

TryStrToBCD

DecimalString (see page 26548)

EXSDecimalException (see page 26514)

199.1.7.3.2 TXSDecimal.DecimalString Property

Represents the object's value as a string.

Pascal

```
property DecimalString: string;
```

C++

```
__property AnsiString DecimalString;
```

Description

Use DecimalString to obtain the value of the TXSDecimal (see page 26545) object as a string. This value is the same as the string that appears in XML documents to represent the decimal value. To work with this object's value as a TBcd value, use the AsBcd (see page 26547) property instead.

Note: If you do not need the precision of binary-coded decimal values, you can use global functions such as StrToFloat and FloatToStr or StrToCurr and CurrToStr to convert this string to a floating-point representation.

See Also

StrToCurr (see page 22962)

CurrToStr (see page 22819)

StrToFloat (see page 22967)

FloatToStr (see page 22890)

AsBcd (see page 26547)

199.1.8 TXSDuration Class

TXSDuration implements the XML schema duration type so that it can be used on invokable interfaces.

Class Hierarchy

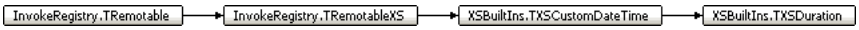

Pascal

```
TXSDuration = class(TXSCustomDateTime);
```

C++

```
class TXSDuration : public TXSCustomDateTime;
```

File

XSBuiltIns

Description

TXSDuration acts as a wrapper for the simple scalar type duration that W3C defines for XML schemas. That is, TXSDuration represents the interval between two points in time. Because this type does not correspond directly to a Delphi or C++ type, it requires special marshaling when used in an invokable interface. TXSDuration enables this marshaling because it is a descendant of TRemotable.

To convert a TXSDuration value to a TDateTime value, use the AsDateTime (see page 26527) property.

On server applications, instances of TXSDuration that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSDuration instance is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSDuration instances that it passes as input parameters, and for freeing any TXSDuration instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 22398)

TXSDate (see page 26534)

TXSDuration

TXSTime (see page 26558)

199.1.8.1 TXSDuration Members

The following tables list the members exposed by TXSDuration.

Protected Fields

	Name	Description
	FDateTime (see page 26518)	This is FDateTime, a member of class TXSCustomDateTime.
	FSerializationOptions (see page 15559)	This is FSerializationOptions, a member of class TRemotable.

Protected Methods

	Name	Description
	GetAsDateTime (see page 26519)	Returns the value of the AsDateTime (see page 26527) property.
	GetAsUTCDateTime (see page 26519)	This is GetAsUTCDateTime, a member of class TXSCustomDateTime.
	GetDay (see page 26519)	Returns the value of the Day (see page 26528) property.
	GetDecimalSecond (see page 26551)	This is GetDecimalSecond, a member of class TXSDuration.
	GetFractionalSeconds (see page 26520)	Returns the value of the FractionalSeconds (see page 26529) property.

	GetHour (see page 26520)	Returns the value of the Hour (see page 26529) property.
	GetHourOffset (see page 26520)	Returns the value of the HourOffset (see page 26530) property.
	GetMillisecond (see page 26521)	Returns the value of the Millisecond (see page 26530) property.
	GetMinute (see page 26521)	Returns the value of the Minute (see page 26531) property.
	GetMinuteOffset (see page 26521)	Returns the value of the MinuteOffset (see page 26531) property.
	GetMonth (see page 26521)	Returns the value of the Month (see page 26532) property.
	GetSecond (see page 26522)	Returns the value of the Second (see page 26532) property.
	GetUseZeroMilliseconds (see page 26522)	Returns the value of the UseZeroMilliseconds (see page 26533) property.
	GetYear (see page 26522)	Returns the value of the Year (see page 26533) property.
	SetAsDateTime (see page 26523)	Sets the value of the AsDateTime (see page 26527) property.
	SetAsUTCDateTime (see page 26523)	Sets the value of the AsUTCDateTime (see page 26528) property.
V	SetDay (see page 26523)	Sets the value of the Day (see page 26528) property.
	SetDecimalSecond (see page 26552)	This is SetDecimalSecond, a member of class TXSDuration.
	SetFractionalSeconds (see page 26524)	Sets the value of the FractionalSeconds (see page 26529) property.
V	SetHour (see page 26524)	Sets the value of the Hour (see page 26529) property.
V	SetHourOffset (see page 26524)	Sets the value of the HourOffset (see page 26530) property.
V	SetMillisecond (see page 26524)	Sets the value of the Millisecond (see page 26530) property.
V	SetMinute (see page 26525)	Sets the value of the Minute (see page 26531) property.
V	SetMinuteOffset (see page 26525)	Sets the value of the MinuteOffset (see page 26531) property.
V	SetMonth (see page 26525)	Sets the value of the Month (see page 26532) property.
V	SetSecond (see page 26526)	Sets the value of the Second (see page 26532) property.
	SetUseZeroMilliseconds (see page 26526)	Sets the value of the UseZeroMilliseconds (see page 26533) property.
V	SetYear (see page 26526)	Sets the value of the Year (see page 26533) property.

Protected Properties

	Name	Description
	Millisecond (see page 26530)	Indicates the Millisecond of the second that the object represents.

Public Methods

	Name	Description
V	Create (see page 26551)	Creates and initializes a TXSDuration (see page 26548) object.
V	Destroy (see page 15558)	Frees the memory associated with the TRemotable (see page 15556) object.
V	Destroy (see page 26517)	Frees the memory associated with the TXSCustomDateTime (see page 26515) object.
V	NativeToXS (see page 26552)	Returns a string that represents the value of a TXSDuration (see page 26548) instance.
V	ObjectToSOAP (see page 15559)	Converts this remotable object to its SOAP representation.
V	ObjectToSOAP (see page 15582)	Converts this remotable object to its SOAP representation.
V	SOAPToObject (see page 15560)	Initializes this TRemotable (see page 15556) instance to represent the value of its SOAP representation.
V	SOAPToObject (see page 15582)	Initializes this TRemotableXS (see page 15580) instance to represent the value of its SOAP representation.
V	XSToNative (see page 26552)	Assigns the properties of the TXSDuration (see page 26548) instance from the string representation of its value.

Public Properties

	Name	Description
	AsDateTime (see page 26527)	Represents the object's value as a TDateTime value.
	AsUTCDateTime (see page 26528)	Represents the object's value as a TDateTime value using UTC time.
	DataContext (see page 15561)	Specifies the object that controls the lifetime of a TRemotable (see page 15556) instance.
	Day (see page 26528)	Indicates the Day of the month that the object represents.
	DecimalSecond (see page 26553)	Indicates the number of seconds, including fractional seconds, in the duration.
	FractionalSeconds (see page 26529)	Indicates any factional portion of a second beyond the seconds specified in the Second (see page 26532) property.
	Hour (see page 26529)	Indicates the hour of the day that the object represents.

	HourOffset (see page 26530)	Indicates the difference between the Hour (see page 26529) property and the hour in the local time zone.
	Minute (see page 26531)	Indicates the minute of the hour that the object represents.
	MinuteOffset (see page 26531)	Indicates the difference between the Minute (see page 26531) property and the minute of the hour in local time.
	Month (see page 26532)	Indicates the month of the year that the object represents.
	Second (see page 26532)	Indicates the second of the minute that the object represents.
R	SerializationOptions (see page 15561)	Indicates how the value of the TRemotable (see page 15556) descendant encodes its value.
	UseZeroMilliseconds (see page 26533)	Indicates whether the string representation of this object's value includes a portion for milliseconds when the value of Millisecond (see page 26530) is zero.
	Year (see page 26533)	Indicates the year that the object represents.

199.1.8.2 TXSDuration.Create Constructor

Creates and initializes a TXSDuration (see page 26548) object.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TXSDuration();
```

Description

Call Create to create a TXSDuration (see page 26548) object at runtime.

199.1.8.3 TXSDuration Methods

The methods of the TXSDuration class are listed here.

Protected Methods

	Name	Description
	GetDecimalSecond (see page 26551)	This is GetDecimalSecond, a member of class TXSDuration.
	SetDecimalSecond (see page 26552)	This is SetDecimalSecond, a member of class TXSDuration.

Public Methods

	Name	Description
V	NativeToXS (see page 26552)	Returns a string that represents the value of a TXSDuration (see page 26548) instance.
V	XSToNative (see page 26552)	Assigns the properties of the TXSDuration (see page 26548) instance from the string representation of its value.

199.1.8.3.1 TXSDuration.GetDecimalSecond Method

Pascal

```
function GetDecimalSecond: Double;
```

C++

```
__fastcall Double GetDecimalSecond();
```

Description

This is GetDecimalSecond, a member of class TXSDuration.

199.1.8.3.2 TXSDuration.NativeToXS Method

Returns a string that represents the value of a TXSDuration (see page 26548) instance.

Pascal

```
function NativeToXS: WideString; override;
```

C++

```
virtual __fastcall BSTR NativeToXS();
```

Description

NativeToXS returns the string that represents the value of this TXSDuration (see page 26548) in a SOAP packet.

See Also

XSToNative (see page 26552)

199.1.8.3.3 TXSDuration.SetDecimalSecond Method

Pascal

```
procedure SetDecimalSecond(const Value: Double);
```

C++

```
__fastcall SetDecimalSecond(const Double Value);
```

Description

This is SetDecimalSecond, a member of class TXSDuration.

199.1.8.3.4 TXSDuration.XSToNative Method

Assigns the properties of the TXSDuration (see page 26548) instance from the string representation of its value.

Pascal

```
procedure XSToNative(Value: WideString); override;
```

C++

```
virtual __fastcall XSToNative(BSTR Value);
```

Description

XSToNative parses the string specified by Data, which is the SOAP encoding of a duration value. It uses this information to set the properties of the TXSDuration (see page 26548) object.

See Also

NativeToXS (see page 26552)

199.1.8.4 TXSDuration Properties

The properties of the TXSDuration class are listed here.

Public Properties

	Name	Description
	DecimalSecond (see page 26553)	Indicates the number of seconds, including fractional seconds, in the duration.

199.1.8.4.1 TXSDuration.DecimalSecond Property

Indicates the number of seconds, including fractional seconds, in the duration.

Pascal

```
property DecimalSecond: Double;
```

C++

```
__property Double DecimalSecond;
```

Description

Use the DecimalSecond property to determine the size of the duration with a granularity finer than the Seconds property provides. DecimalSecond represents the number of seconds in the duration (above and beyond the years, months, days, hours and minutes represented by the other TXSDuration (see page 26548) properties), including any fractional seconds.

See Also

Second (see page 26532)

FractionalSeconds (see page 26529)

199.1.9 TXSHexBinary Class

TXSHexBinary implements the XML schema hexBinary type so that it can be used on invokable interfaces.

Class Hierarchy

Pascal

```
TXSHexBinary = class(TRemotableXS);
```

C++

```
class TXSHexBinary : public TRemotableXS;
```

File

XSBuiltIns

Description

TXSHexBinary acts as a wrapper for the simple scalar hexBinary type that W3C defines for XML schemas. Because this type does not correspond directly to a Delphi or C++ type, it requires special marshaling when used in an invokable interface. TXSHexBinary enables this marshaling because it is a descendant of TRemotable.

Use the AsByteArray (see page 26555) property to get or set the value of TXSHexBinary as an array of bytes. You can also set the value of TXSHexBinary from an integer using the global IntToHex function to convert an integer into a hexadecimal string and setting the HexBinaryString (see page 26556) property.

On server applications, instances of TXSHexBinary that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSHexBinary instance is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSHexBinary instances that it passes as input

parameters, and for freeing any TXSHexBinary instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 22398)

TXSDecimal (see page 26545)

HexToBin

BinToHex

IntToHex

EXSHexBinaryException (see page 26514)

199.1.9.1 TXSHexBinary Members

The following tables list the members exposed by TXSHexBinary.

Protected Fields

	Name	Description
	FSerializationOptions (see page 15559)	This is FSerializationOptions, a member of class TRemotable.

Public Methods

	Name	Description
V	Create (see page 15558)	Creates and initializes a TRemotable (see page 15556) object.
V	Destroy (see page 15558)	Frees the memory associated with the TRemotable (see page 15556) object.
V	NativeToXS (see page 26555)	Returns a string that represents the value of a TXSHexBinary (see page 26553) instance.
V	ObjectToSOAP (see page 15559)	Converts this remotable object to its SOAP representation.
V	ObjectToSOAP (see page 15582)	Converts this remotable object to its SOAP representation.
V	SOAPToObject (see page 15560)	Initializes this TRemotable (see page 15556) instance to represent the value of its SOAP representation.
V	SOAPToObject (see page 15582)	Initializes this TRemotableXS (see page 15580) instance to represent the value of its SOAP representation.
V	XSToNative (see page 26555)	Assigns a string as the value of the TXSHexBinary (see page 26553) instance.

Public Properties

	Name	Description
	AsByteArray (see page 26555)	Represents the object's value as an array of bytes.
	DataContext (see page 15561)	Specifies the object that controls the lifetime of a TRemotable (see page 15556) instance.
	HexBinaryString (see page 26556)	This is HexBinaryString, a member of class TXSHexBinary.
R	SerializationOptions (see page 15561)	Indicates how the value of the TRemotable (see page 15556) descendant encodes its value.

199.1.9.2 TXSHexBinary Methods

The methods of the TXSHexBinary class are listed here.

Public Methods

	Name	Description
V	NativeToXS (see page 26555)	Returns a string that represents the value of a TXSHexBinary (see page 26553) instance.
V	XSToNative (see page 26555)	Assigns a string as the value of the TXSHexBinary (see page 26553) instance.

199.1.9.2.1 TXSHexBinary.NativeToXS Method

Returns a string that represents the value of a TXSHexBinary (see page 26553) instance.

Pascal

```
function NativeToXS: WideString; override;
```

C++

```
virtual __fastcall BSTR NativeToXS();
```

Description

NativeToXS returns the string that represents the value of this TXSHexBinary (see page 26553) object in a SOAP packet. That is, NativeToXS converts the HexBinaryString (see page 26556) property to a WideString and returns the result.

See Also

XSToNative (see page 26555)

hexbinarystring (see page 26556)

199.1.9.2.2 TXSHexBinary.XSToNative Method

Assigns a string as the value of the TXSHexBinary (see page 26553) instance.

Pascal

```
procedure XSToNative(Value: WideString); override;
```

C++

```
virtual __fastcall XSToNative(BSTR Value);
```

Description

XSToNative sets the value of this TXSHexBinary (see page 26553) object to a string that represents a formatted hexadecimal value.

Data is the SOAP encoding of a hexadecimal value. It becomes the value of the HexBinaryString (see page 26556) property.

See Also

NativeToXS (see page 26555)

hexbinarystring (see page 26556)

199.1.9.3 TXSHexBinary Properties

The properties of the TXSHexBinary class are listed here.

Public Properties

	Name	Description
	AsByteArray (see page 26555)	Represents the object's value as an array of bytes.
	HexBinaryString (see page 26556)	This is HexBinaryString, a member of class TXSHexBinary.

199.1.9.3.1 TXSHexBinary.AsByteArray Property

Represents the object's value as an array of bytes.

Pascal

```
property AsByteArray: TByteDynArray;
```

C++

```
__property TByteDynArray AsByteArray;
```

Description

Use AsByteArray to obtain the value of the TXSHexBinary (see page 26553) object as an array of bytes. This value is obtained from the HexBinaryString (see page 26556) property using the global HexToBin function. When you set AsByteArray, TXSHexBinary (see page 26553) uses the global BinToHex function to convert the binary value to a string and sets the HexBinaryString (see page 26556) property.

See Also

BinToHex

HexToBin

hexbinarystring (see page 26556)

199.1.9.3.2 TXSHexBinary.HexBinaryString Property

Pascal

```
property HexBinaryString: string;
```

C++

```
__property AnsiString HexBinaryString;
```

Description

This is HexBinaryString, a member of class TXSHexBinary.

199.1.10 TXSString Class

Class Hierarchy

Pascal

```
TXSString = class(TRemotableXS);
```

C++

```
class TXSString : public TRemotableXS;
```

File

XSBuiltIns

Description

This is class XSBuiltIns.TXSString.

199.1.10.1 TXSString Members

The following tables list the members exposed by TXSString.

Protected Fields

	Name	Description
	FSerializationOptions (see page 15559)	This is FSerializationOptions, a member of class TRemotable.

Public Methods

	Name	Description
	Create (see page 15558)	Creates and initializes a TRemotable (see page 15556) object.
	Destroy (see page 15558)	Frees the memory associated with the TRemotable (see page 15556) object.
	NativeToXS (see page 26557)	Returns a string that represents the value of a TRemotableXS instance.
	ObjectToSOAP (see page 15559)	Converts this remotable object to its SOAP representation.
	ObjectToSOAP (see page 15582)	Converts this remotable object to its SOAP representation.
	SOAPToObject (see page 15560)	Initializes this TRemotable (see page 15556) instance to represent the value of its SOAP representation.
	SOAPToObject (see page 15582)	Initializes this TRemotableXS (see page 15580) instance to represent the value of its SOAP representation.
	XSToNative (see page 26558)	Assigns the value of a string to the TRemotableXS instance.

Public Properties

	Name	Description
	DataContext (see page 15561)	Specifies the object that controls the lifetime of a TRemotable (see page 15556) instance.
	SerializationOptions (see page 15561)	Indicates how the value of the TRemotable (see page 15556) descendant encodes its value.

199.1.10.2 TXSString Methods

The methods of the TXSString class are listed here.

Public Methods

	Name	Description
	NativeToXS (see page 26557)	Returns a string that represents the value of a TRemotableXS instance.
	XSToNative (see page 26558)	Assigns the value of a string to the TRemotableXS instance.

199.1.10.2.1 TXSString.NativeToXS Method

Returns a string that represents the value of a TRemotableXS instance.

Pascal

```
function NativeToXS: WideString; override;
```

C++

```
virtual __fastcall BSTR NativeToXS();
```

Description

Override NativeToXS when implementing a TRemotableXS descendant that represents a non-native scalar type. NativeToXS returns a string that represents the value of the TRemotableXS instance. This string is used when marshaling the value of the TRemotableXS descendant.

Classes that implement NativeToXS must also override the XSToNative method so that it performs the converse transformation.

See Also

XSToNative (see page 15583)

Using Remotable Objects

199.1.10.2.2 TXSString.XSToNative Method

Assigns the value of a string to the TRemotableXS instance.

Pascal

```
procedure XSToNative(Value: WideString); override;
```

C++

```
virtual __fastcall XSToNative(BSTR Value);
```

Description

Override XSToNative when implementing a TRemotableXS descendant that represents a non-native scalar type. NativeToXS interprets a string that represents the value of the TRemotableXS instance and uses it to make any necessary changes so that the TRemotableXS object reflects that value. It reverses the transformation that the NativeToXS method performs.

Data is a string that represents the value of the TRemotableXS instance.

See Also

NativeToXS (see page 15581)

Using Remotable Objects

199.1.11 TXSTime Class

TXSTime implements the XML schema time type so that it can be used on invokable interfaces.

Class Hierarchy

Pascal

```
TXSTime = class(TRemotableXS);
```

C++

```
class TXSTime : public TRemotableXS;
```

File

XSBuiltIns

Description

TXSTime acts as a wrapper for the simple scalar type time that W3C defines for XML schemas. Because this type does not correspond directly to a Delphi or C++ type, it requires special marshaling when used in an invokable interface. TXSTime enables this marshaling because it is a descendant of TRemotable.

To convert a TXSTime value to a TDateTime value, use the AsTime (see page 26566) property. Other properties give the Hour (see page 26567), Minute (see page 26568), Second (see page 26569), and Millisecond (see page 26568) that make up that value. These properties represent the time using Coordinated Universal Time (UTC). To convert that to a local time zone, adjust the hour and minute portions using the HourOffset (see page 26567) and MinuteOffset (see page 26569) properties.

On server applications, instances of TXSTime that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSTime instance is created in a method that was called remotely using an invokable

interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSTime instances that it passes as input parameters, and for freeing any TXSTime instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 22398)

TXSDate (see page 26534)

TXSDuration (see page 26548)

TXSDateTime (see page 26541)

199.1.11.1 TXSTime Members

The following tables list the members exposed by TXSTime.

Protected Fields

	Name	Description
	FSerializationOptions (see page 15559)	This is FSerializationOptions, a member of class TRemotable.

Protected Methods

	Name	Description
	GetAsTime (see page 26561)	This is GetAsTime, a member of class TXSTime.
	GetFractionalSeconds (see page 26561)	This is GetFractionalSeconds, a member of class TXSTime.
	GetFractionalSecondString (see page 26561)	This is GetFractionalSecondString, a member of class TXSTime.
	GetHour (see page 26562)	This is GetHour, a member of class TXSTime.
	GetHourOffset (see page 26562)	This is GetHourOffset, a member of class TXSTime.
	GetMillisecond (see page 26562)	This is GetMillisecond, a member of class TXSTime.
	GetMinute (see page 26562)	This is GetMinute, a member of class TXSTime.
	GetMinuteOffset (see page 26563)	This is GetMinuteOffset, a member of class TXSTime.
	GetSecond (see page 26563)	This is GetSecond, a member of class TXSTime.
	SetAsTime (see page 26563)	This is SetAsTime, a member of class TXSTime.
	SetFractionalSeconds (see page 26564)	This is SetFractionalSeconds, a member of class TXSTime.
	SetHour (see page 26564)	This is SetHour, a member of class TXSTime.
	SetHourOffset (see page 26564)	This is SetHourOffset, a member of class TXSTime.
	SetMillisecond (see page 26564)	This is SetMillisecond, a member of class TXSTime.
	SetMinute (see page 26564)	This is SetMinute, a member of class TXSTime.
	SetMinuteOffset (see page 26565)	This is SetMinuteOffset, a member of class TXSTime.
	SetSecond (see page 26565)	This is SetSecond, a member of class TXSTime.

Protected Properties

	Name	Description
R	FractionalSecondString (see page 26567)	This is FractionalSecondString, a member of class TXSTime.

Public Methods

	Name	Description
	Clone (see page 26561)	Returns a new TXSTime (see page 26558) object that represents the same value as this one.
V	Create (see page 15558)	Creates and initializes a TRemotable (see page 15556) object.
V	Destroy (see page 15558)	Frees the memory associated with the TRemotable (see page 15556) object.
V	NativeToXS (see page 26563)	Returns a string that represents the value of a TXSTime (see page 26558) instance.

V	ObjectToSOAP (see page 15559)	Converts this remotable object to its SOAP representation.
V	ObjectToSOAP (see page 15582)	Converts this remotable object to its SOAP representation.
V	SOAPToObject (see page 15560)	Initializes this TRemotable (see page 15556) instance to represent the value of its SOAP representation.
V	SOAPToObject (see page 15582)	Initializes this TRemotableXS (see page 15580) instance to represent the value of its SOAP representation.
V	XSToNative (see page 26565)	Assigns the properties of the TXSTime (see page 26558) instance from the string representation of its value.

Public Properties

	Name	Description
	AsTime (see page 26566)	Represents the object's value as a TDateTime value.
	DataContext (see page 15561)	Specifies the object that controls the lifetime of a TRemotable (see page 15556) instance.
	FractionalSeconds (see page 26566)	Indicates any factional portion of a second beyond the seconds specified in the Second (see page 26569) property.
	Hour (see page 26567)	Indicates the hour of the day that the object represents.
	HourOffset (see page 26567)	Indicates the difference between the Hour (see page 26567) property and the hour in the local time zone.
	Millisecond (see page 26568)	Indicates the Millisecond of the second that the object represents.
	Minute (see page 26568)	Indicates the minute of the hour that the object represents.
	MinuteOffset (see page 26569)	Indicates the difference between the Minute (see page 26568) property and the minute of the hour in local time.
	Second (see page 26569)	Indicates the second of the minute that the object represents.
R	SerializationOptions (see page 15561)	Indicates how the value of the TRemotable (see page 15556) descendant encodes its value.
	UseZeroMilliseconds (see page 26569)	Indicates whether the string representation of this object's value includes a portion for fractions of a second when the value of Millisecond (see page 26568) and FractionalSeconds (see page 26566) is zero.

199.1.11.2 TXSTime Methods

The methods of the TXSTime class are listed here.

Protected Methods

	Name	Description
	GetAsTime (see page 26561)	This is GetAsTime, a member of class TXSTime.
	GetFractionalSeconds (see page 26561)	This is GetFractionalSeconds, a member of class TXSTime.
	GetFractionalSecondString (see page 26561)	This is GetFractionalSecondString, a member of class TXSTime.
	GetHour (see page 26562)	This is GetHour, a member of class TXSTime.
	GetHourOffset (see page 26562)	This is GetHourOffset, a member of class TXSTime.
	GetMillisecond (see page 26562)	This is GetMillisecond, a member of class TXSTime.
	GetMinute (see page 26562)	This is GetMinute, a member of class TXSTime.
	GetMinuteOffset (see page 26563)	This is GetMinuteOffset, a member of class TXSTime.
	GetSecond (see page 26563)	This is GetSecond, a member of class TXSTime.
	SetAsTime (see page 26563)	This is SetAsTime, a member of class TXSTime.
	SetFractionalSeconds (see page 26564)	This is SetFractionalSeconds, a member of class TXSTime.
	SetHour (see page 26564)	This is SetHour, a member of class TXSTime.
	SetHourOffset (see page 26564)	This is SetHourOffset, a member of class TXSTime.
	SetMillisecond (see page 26564)	This is SetMillisecond, a member of class TXSTime.
	SetMinute (see page 26564)	This is SetMinute, a member of class TXSTime.
	SetMinuteOffset (see page 26565)	This is SetMinuteOffset, a member of class TXSTime.
	SetSecond (see page 26565)	This is SetSecond, a member of class TXSTime.

Public Methods

	Name	Description
	Clone (see page 26561)	Returns a new TXSTime (see page 26558) object that represents the same value as this one.
V	NativeToXS (see page 26563)	Returns a string that represents the value of a TXSTime (see page 26558) instance.
V	XSToNative (see page 26565)	Assigns the properties of the TXSTime (see page 26558) instance from the string representation of its value.

199.1.11.2.1 TXSTime.Clone Method

Returns a new TXSTime (see page 26558) object that represents the same value as this one.

Pascal

```
function Clone: TXSTime;
```

C++

```
__fastcall TXSTime Clone();
```

Description

Call Clone to obtain a new TXSTime (see page 26558) instance that represents the same time as this one. Clone returns the new TXSTime (see page 26558) object.

199.1.11.2.2 TXSTime.GetAsTime Method

Pascal

```
function GetAsTime: TDateTime;
```

C++

```
__fastcall TDateTime GetAsTime();
```

Description

This is GetAsTime, a member of class TXSTime.

199.1.11.2.3 TXSTime.GetFractionalSeconds Method

Pascal

```
function GetFractionalSeconds: Double;
```

C++

```
__fastcall Double GetFractionalSeconds();
```

Description

This is GetFractionalSeconds, a member of class TXSTime.

199.1.11.2.4 TXSTime.GetFractionalSecondString Method

Pascal

```
function GetFractionalSecondString: string;
```

C++

```
__fastcall AnsiString GetFractionalSecondString();
```

Description

This is GetFractionalSecondString, a member of class TXSTime.

199.1.11.2.5 TXSTime.GetHour Method

Pascal

```
function GetHour: Word;
```

C++

```
__fastcall Word GetHour();
```

Description

This is GetHour, a member of class TXSTime.

199.1.11.2.6 TXSTime.GetHourOffset Method

Pascal

```
function GetHourOffset: SmallInt;
```

C++

```
__fastcall SmallInt GetHourOffset();
```

Description

This is GetHourOffset, a member of class TXSTime.

199.1.11.2.7 TXSTime.GetMillisecond Method

Pascal

```
function GetMillisecond: Word;
```

C++

```
__fastcall Word GetMillisecond();
```

Description

This is GetMillisecond, a member of class TXSTime.

199.1.11.2.8 TXSTime.GetMinute Method

Pascal

```
function GetMinute: Word;
```

C++

```
__fastcall Word GetMinute();
```

Description

This is GetMinute, a member of class TXSTime.

199.1.11.2.9 TXSTime.GetMinuteOffset Method

Pascal

```
function GetMinuteOffset: SmallInt;
```

C++

```
__fastcall SmallInt GetMinuteOffset();
```

Description

This is GetMinuteOffset, a member of class TXSTime.

199.1.11.2.10 TXSTime.GetSecond Method

Pascal

```
function GetSecond: Word;
```

C++

```
__fastcall Word GetSecond();
```

Description

This is GetSecond, a member of class TXSTime.

199.1.11.2.11 TXSTime.NativeToXS Method

Returns a string that represents the value of a TXSTime (see page 26558) instance.

Pascal

```
function NativeToXS: WideString; override;
```

C++

```
virtual __fastcall BSTR NativeToXS();
```

Description

NativeToXS returns the string that represents the value of this TXSTime (see page 26558) object in a SOAP packet.

See Also

XSToNative (see page 26565)

UseZeroMilliseconds (see page 26569)

199.1.11.2.12 TXSTime.SetAsTime Method

Pascal

```
procedure SetAsTime(const Value: TDateTime);
```

C++

```
__fastcall SetAsTime(const TDateTime Value);
```

Description

This is SetAsTime, a member of class TXSTime.

199.1.11.2.13 TXSTime.SetFractionalSeconds Method

Pascal

```
procedure SetFractionalSeconds(const Value: Double);
```

C++

```
__fastcall SetFractionalSeconds(const Double Value);
```

Description

This is SetFractionalSeconds, a member of class TXSTime.

199.1.11.2.14 TXSTime.SetHour Method

Pascal

```
procedure SetHour(const Value: Word);
```

C++

```
__fastcall SetHour(const Word Value);
```

Description

This is SetHour, a member of class TXSTime.

199.1.11.2.15 TXSTime.SetHourOffset Method

Pascal

```
procedure SetHourOffset(const Value: SmallInt);
```

C++

```
__fastcall SetHourOffset(const SmallInt Value);
```

Description

This is SetHourOffset, a member of class TXSTime.

199.1.11.2.16 TXSTime.SetMillisecond Method

Pascal

```
procedure SetMillisecond(const Value: Word);
```

C++

```
__fastcall SetMillisecond(const Word Value);
```

Description

This is SetMillisecond, a member of class TXSTime.

199.1.11.2.17 TXSTime.SetMinute Method

Pascal

```
procedure SetMinute(const Value: Word);
```

C++

```
__fastcall SetMinute(const Word Value);
```

Description

This is SetMinute, a member of class TXSTime.

199.1.11.2.18 TXSTime.SetMinuteOffset Method

Pascal

```
procedure SetMinuteOffset(const Value: SmallInt);
```

C++

```
__fastcall SetMinuteOffset(const SmallInt Value);
```

Description

This is SetMinuteOffset, a member of class TXSTime.

199.1.11.2.19 TXSTime.SetSecond Method

Pascal

```
procedure SetSecond(const Value: Word);
```

C++

```
__fastcall SetSecond(const Word Value);
```

Description

This is SetSecond, a member of class TXSTime.

199.1.11.2.20 TXSTime.XSToNative Method

Assigns the properties of the TXSTime (see page 26558) instance from the string representation of its value.

Pascal

```
procedure XSToNative(Value: WideString); override;
```

C++

```
virtual __fastcall XSToNative(BSTR Value);
```

Description

XSToNative parses the string specified by Data, which is the SOAP encoding of a time value. It uses this information to set the properties of the TXSTime (see page 26558) object.

See Also

NativeToXS (see page 26563)

199.1.11.3 TXSTime Properties

The properties of the TXSTime class are listed here.

Protected Properties

	Name	Description
R	FractionalSecondString (see page 26567)	This is FractionalSecondString, a member of class TXSTime.

Public Properties

	Name	Description
	AsTime (see page 26566)	Represents the object's value as a TDateTime value.
	FractionalSeconds (see page 26566)	Indicates any factional portion of a second beyond the seconds specified in the Second (see page 26569) property.
	Hour (see page 26567)	Indicates the hour of the day that the object represents.
	HourOffset (see page 26567)	Indicates the difference between the Hour (see page 26567) property and the hour in the local time zone.
	Millisecond (see page 26568)	Indicates the Millisecond of the second that the object represents.
	Minute (see page 26568)	Indicates the minute of the hour that the object represents.
	MinuteOffset (see page 26569)	Indicates the difference between the Minute (see page 26568) property and the minute of the hour in local time.
	Second (see page 26569)	Indicates the second of the minute that the object represents.
	UseZeroMilliseconds (see page 26569)	Indicates whether the string representation of this object's value includes a portion for fractions of a second when the value of Millisecond (see page 26568) and FractionalSeconds (see page 26566) is zero.

199.1.11.3.1 TXSTime.AsTime Property

Represents the object's value as a TDateTime value.

Pascal

```
property AsTime: TDateTime;
```

C++

```
__property TDateTime AsTime;
```

Description

Use AsTime to read the value of the TXSTime (see page 26558) object into a variable of type TDateTime or set its value from a TDateTime value. AsTime handles the mapping between a TDateTime value and the Hour (see page 26567), Minute (see page 26568), and Second (see page 26569) properties. When reading AsTime, the resulting TDateTime value is given a data portion of 0. When setting AsTime, the data portion is ignored.

Note: AsTime represents the time using Coordinated Universal Time (UTC). To convert this to a local time, add the offsets specified by HourOffset (see page 26567) and MinuteOffset (see page 26569) to the hours and minutes.

See Also

Hour (see page 26567)

Minute (see page 26568)

Second (see page 26569)

HourOffset (see page 26567)

MinuteOffset (see page 26569)

199.1.11.3.2 TXSTime.FractionalSeconds Property

Indicates any factional portion of a second beyond the seconds specified in the Second (see page 26569) property.

Pascal

```
property FractionalSeconds: Double;
```

C++

```
__property Double FractionalSeconds;
```

Description

Use FractionalSeconds to get or set the fractional seconds of the time value. FractionalSeconds represents the same information as the Millisecond (see page 26568) property, but can provide greater precision.

See Also

AsTime (see page 26566)

Minute (see page 26568)

Second (see page 26569)

Millisecond (see page 26568)

UseZeroMilliseconds (see page 26569)

199.1.11.3.3 TXSTime.FractionalSecondString Property

Pascal

```
property FractionalSecondString: string;
```

C++

```
__property AnsiString FractionalSecondString;
```

Description

This is FractionalSecondString, a member of class TXSTime.

199.1.11.3.4 TXSTime.Hour Property

Indicates the hour of the day that the object represents.

Pascal

```
property Hour: Word;
```

C++

```
__property Word Hour;
```

Description

Use Hour to get or set the hour portion of the time value. Valid Hour values are 0 through 23.

Note: Hour represents the hour using Coordinated Universal Time (UTC). To convert the hour to local time, you must add the value of the HourOffset (see page 26567) property.

See Also

AsTime (see page 26566)

Minute (see page 26568)

Second (see page 26569)

Millisecond (see page 26568)

HourOffset (see page 26567)

199.1.11.3.5 TXSTime.HourOffset Property

Indicates the difference between the Hour (see page 26567) property and the hour in the local time zone.

Pascal

```
property HourOffset: SmallInt;
```

C++

```
__property SmallInt HourOffset;
```

Description

The Hour (see page 26567) property represents the hour of the day using Coordinated Universal Time (UTC). HourOffset is the correction factor you can add to Hour (see page 26567) to obtain the hour of the day in local time.

See Also

Hour (see page 26567)

199.1.11.3.6 TXSTime.Millisecond Property

Indicates the Millisecond of the second that the object represents.

Pascal

```
property Millisecond: Word;
```

C++

```
__property Word Millisecond;
```

Description

Use Millisecond to get or set the millisecond portion of the time value. Valid Millisecond values are 1 through 59.

Note: You can also use the FractionalSeconds (see page 26566) property to obtain fractional portions of the seconds. Unlike Millisecond, which gives the value in milliseconds, FractionalSeconds (see page 26566) returns a fraction of a second.

See Also

AsTime (see page 26566)

Hour (see page 26567)

Minute (see page 26568)

Second (see page 26569)

UseZeroMilliseconds (see page 26569)

FractionalSeconds (see page 26566)

199.1.11.3.7 TXSTime.Minute Property

Indicates the minute of the hour that the object represents.

Pascal

```
property Minute: Word;
```

C++

```
__property Word Minute;
```

Description

Use Minute to get or set the minute portion of the time value. Valid Minute values are 0 through 59.

Note: Minute represents the minute using Coordinated Universal Time (UTC). To convert the minutes to local time, you must

add the value of the MinuteOffset (see page 26569) property.

See Also

AsTime (see page 26566)

Hour (see page 26567)

Second (see page 26569)

Millisecond (see page 26568)

MinuteOffset (see page 26569)

199.1.11.3.8 TXSTime.MinuteOffset Property

Indicates the difference between the Minute (see page 26568) property and the minute of the hour in local time.

Pascal

```
property MinuteOffset: SmallInt;
```

C++

```
__property SmallInt MinuteOffset;
```

Description

The Minute (see page 26568) property represents the minute of the hour using Coordinated Universal Time (UTC). MinuteOffset is the correction factor you can add to Minute (see page 26568) to obtain the minute of the hour in local time.

See Also

Minute (see page 26568)

199.1.11.3.9 TXSTime.Second Property

Indicates the second of the minute that the object represents.

Pascal

```
property Second: Word;
```

C++

```
__property Word Second;
```

Description

Use Second to get or set the seconds portion of the time value. Valid Second values are 0 through 59.

See Also

AsTime (see page 26566)

Millisecond (see page 26568)

Hour (see page 26567)

Minute (see page 26568)

Second

199.1.11.3.10 TXSTime.UseZeroMilliseconds Property

Indicates whether the string representation of this object's value includes a portion for fractions of a second when the value of

Millisecond (see page 26568) and FractionalSeconds (see page 26566) is zero.

Pascal

```
property UseZeroMilliseconds: Boolean;
```

C++

```
__property Boolean UseZeroMilliseconds;
```

Description

By default, the NativeToXS (see page 26563) method does not include the fractional portion of a second in the string encoding of this object's value when the value represents an exact number of seconds. If you set UseZeroMilliseconds, NativeToXS (see page 26563) changes its encoding to include milliseconds even when the value is zero. This is necessary for some Web Services that do not support the full time schema defined by W3C.

See Also

Millisecond (see page 26568)

NativeToXS (see page 26563)

FractionalSeconds (see page 26566)

199.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DateTimeToXMLTime (see page 26571)	Returns the XML string that represents a TDateTime value.
	DateTimeToXSDateTime (see page 26571)	Returns a TXSDateTime (see page 26541) object that encodes a TDateTime value.
	GetDataFromFile (see page 26572)	This is function XSBuiltIns.GetDataFromFile.
	InitXSTypes (see page 26572)	This is function XSBuiltIns.InitXSTypes.
	SoapFloatToStr (see page 26573)	This is function XSBuiltIns.SoapFloatToStr.
	SoapStrToFloat (see page 26573)	This is function XSBuiltIns.SoapStrToFloat.
	XMLTimeToDateTime (see page 26573)	Returns a TDateTime value to represent an XML datetime value.

199.2.1 XSBuiltIns.DateTimeToXMLTime Function

Returns the XML string that represents a TDateTime value.

Pascal

```
function DateTimeToXMLTime(Value: TDateTime; ApplyLocalBias: Boolean = True): WideString;
```

C++

```
BSTR DateTimeToXMLTime(TDateTime Value, Boolean ApplyLocalBias = True);
```

File

XSBuiltIns

Description

DateTimeToXMLTime returns the XML representation of a specified TDateTime value.

Value is the TDateTime value to convert. If the time zone information is important, this value is assumed to be given in Coordinated Universal Time (UTC).

CalcLocalBias indicates whether the XML string should include offsets for converting Value to the local time.

See Also

DateTimeToXSDateTime (see page 26571)

XMLTimeToDateTime (see page 26573)

199.2.2 XSBuiltIns.DateTimeToXSDateTime Function

Returns a TXSDateTime (see page 26541) object that encodes a TDateTime value.

Pascal

```
function DateTimeToXSDateTime(const Value: TDateTime; ApplyLocalBias: Boolean = False):
```

```
TXSDateTime;
```

C++

```
TXSDateTime DateTimeToXSDateTime(const TDateTime Value, Boolean ApplyLocalBias = False);
```

File

XSBuiltIns

Description

DateTimeToXSDateTime returns a TXSDateTime (see page 26541) object that corresponds to a specified TDateTime value. The TXSDateTime (see page 26541) object can be used to represent a date-and-time value on an invokable interface. As a remotable object, it can encoded in SOAP packets.

Value is the TDateTime value to convert. If the time zone information is important, this value is assumed to be given in Coordinated Universal Time (UTC).

CalcLocalBias indicates whether the TXSDateTime (see page 26541) object should include offsets for converting Value to the local time zone. If CalcLocalBias is true, DateTimeToXSDateTime looks up the local time zone and uses it to set the HourOffset and MinuteOffset properties of the return value.

See Also

IInvokable

TXSCustomDateTime (see page 26515)

TXSCustomDateTime (see page 26515)

199.2.3 XSBuiltIns.GetDataFromFile Function

Pascal

```
function GetDataFromFile(AFileName: string): string;
```

C++

```
AnsiString GetDataFromFile(AnsiString AFileName);
```

File

XSBuiltIns

Description

This is function XSBuiltIns.GetDataFromFile.

199.2.4 XSBuiltIns.InitXSTypes Function

Pascal

```
procedure InitXSTypes;
```

C++

```
InitXSTypes();
```

File

XSBuiltIns

Description

This is function XSBuiltIns.InitXSTypes.

199.2.5 XSBuiltIns.SoapFloatToStr Function

Pascal

```
function SoapFloatToStr(Value: double): string;
```

C++

```
AnsiString SoapFloatToStr(double Value);
```

File

XSBuiltIns

Description

This is function XSBuiltIns.SoapFloatToStr.

199.2.6 XSBuiltIns.SoapStrToFloat Function

Pascal

```
function SoapStrToFloat(Value: string): double;
```

C++

```
double SoapStrToFloat(AnsiString Value);
```

File

XSBuiltIns

Description

This is function XSBuiltIns.SoapStrToFloat.

199.2.7 XSBuiltIns.XMLTimeToDateTime Function

Returns a TDateTime value to represent an XML datetime value.

Pascal

```
function XMLTimeToDateTime(const XMLDateTime: WideString; AsUTCTime: Boolean = False):
TDateTime;
```

C++

```
TDateTime XMLTimeToDateTime(const BSTR XMLDateTime, Boolean AsUTCTime = False);
```

File

XSBuiltIns

Description

XMLTimeToDateTime returns a TDateTime value for an XML date-and time value. This function is the inverse to the DateTimeToXMLTime (see page 26571) function.

Value is the XML string to convert.

See Also

DateTimeToXSDateTime (see page 26571)

DateTimeToXMLTime (see page 26571)

199.3 Types

The following table lists types in this documentation.

Types

Name	Description
TXSTimeInstant (see page 26575)	This is type XSBuiltIns.TXSTimeInstant.

199.3.1 XSBuiltIns.TXSTimeInstant Type

Pascal

```
TXSTimeInstant = TXSDateTime;
```

C++

```
TXSDateTime TXSTimeInstant;
```

File

XSBuiltIns

Description

This is type XSBuiltIns.TXSTimeInstant.

199.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
IncrementAmount (see page 26576)	This is constant XSBuiltIns.IncrementAmount.
MaxDay (see page 26577)	This is constant XSBuiltIns.MaxDay.
MaxMonth (see page 26577)	This is constant XSBuiltIns.MaxMonth.
MinDay (see page 26577)	This is constant XSBuiltIns.MinDay.
MinMonth (see page 26578)	This is constant XSBuiltIns.MinMonth.
SDecimal (see page 26578)	This is constant XSBuiltIns.SDecimal.
SHexMarker (see page 26578)	This is constant XSBuiltIns.SHexMarker.
SLocalTimeMarker (see page 26579)	This is constant XSBuiltIns.SLocalTimeMarker.
SNAN (see page 26579)	This is constant XSBuiltIns.SNAN.
SNegative (see page 26579)	This is constant XSBuiltIns.SNegative.
SoapDecimalSeparator (see page 26580)	This is constant XSBuiltIns.SoapDecimalSeparator.
SoapTimePrefix (see page 26580)	This is constant XSBuiltIns.SoapTimePrefix.
SPlus (see page 26580)	This is constant XSBuiltIns.SPlus.
SSciNotationMarker (see page 26581)	This is constant XSBuiltIns.SSciNotationMarker.
XMLDateSeparator (see page 26581)	This is constant XSBuiltIns.XMLDateSeparator.
XMLDayPos (see page 26581)	This is constant XSBuiltIns.XMLDayPos.
XMLDefaultYearDigits (see page 26582)	This is constant XSBuiltIns.XMLDefaultYearDigits.
XMLDurationDay (see page 26582)	This is constant XSBuiltIns.XMLDurationDay.
XMLDurationHour (see page 26582)	This is constant XSBuiltIns.XMLDurationHour.
XMLDurationMinute (see page 26583)	This is constant XSBuiltIns.XMLDurationMinute.
XMLDurationMonth (see page 26583)	This is constant XSBuiltIns.XMLDurationMonth.
XMLDurationSecond (see page 26583)	This is constant XSBuiltIns.XMLDurationSecond.
XMLDurationStart (see page 26584)	This is constant XSBuiltIns.XMLDurationStart.
XMLDurationYear (see page 26584)	This is constant XSBuiltIns.XMLDurationYear.
XMLHourOffsetMinusMarker (see page 26584)	This is constant XSBuiltIns.XMLHourOffsetMinusMarker.
XMLHourOffsetPlusMarker (see page 26585)	This is constant XSBuiltIns.XMLHourOffsetPlusMarker.
XMLMilSecPos (see page 26585)	This is constant XSBuiltIns.XMLMilSecPos.
XMLMonthPos (see page 26585)	This is constant XSBuiltIns.XMLMonthPos.
XMLTimeSeparator (see page 26586)	This is constant XSBuiltIns.XMLTimeSeparator.
XMLYearPos (see page 26586)	This is constant XSBuiltIns.XMLYearPos.

199.4.1 XSBuiltIns.IncrementAmount Constant

Pascal

```
IncrementAmount: array[Boolean] of Integer = (1, -1);
```

C++

```
array[Boolean] of Integer IncrementAmount = (1, -1);
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.IncrementAmount.

199.4.2 XSBuiltIns.MaxDay Constant

Pascal

```
MaxDay = 31;
```

C++

```
MaxDay = 31;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.MaxDay.

199.4.3 XSBuiltIns.MaxMonth Constant

Pascal

```
MaxMonth = 12;
```

C++

```
MaxMonth = 12;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.MaxMonth.

199.4.4 XSBuiltIns.MinDay Constant

Pascal

```
MinDay = 1;
```

C++

```
MinDay = 1;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.MinDay.

199.4.5 XSBuiltIns.MinMonth Constant

Pascal

```
MinMonth = 1;
```

C++

```
MinMonth = 1;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.MinMonth.

199.4.6 XSBuiltIns.SDecimal Constant

Pascal

```
SDecimal: Char = '.';
```

C++

```
Char SDecimal = '.';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SDecimal.

199.4.7 XSBuiltIns.SHexMarker Constant

Pascal

```
SHexMarker = '$';
```

C++

```
SHexMarker = '$';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SHexMarker.

199.4.8 XSBuiltIns.SLocalTimeMarker Constant

Pascal

```
SLocalTimeMarker = 'Z';
```

C++

```
SLocalTimeMarker = 'Z';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SLocalTimeMarker.

199.4.9 XSBuiltIns.SNAN Constant

Pascal

```
SNAN = 'NAN';
```

C++

```
SNAN = 'NAN';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SNAN.

199.4.10 XSBuiltIns.SNegative Constant

Pascal

```
SNegative: Char = '-';
```

C++

```
Char SNegative = '-';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SNegative.

199.4.11 XSBuiltIns.SoapDecimalSeparator Constant

Pascal

```
SoapDecimalSeparator = '.';
```

C++

```
SoapDecimalSeparator = '.';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SoapDecimalSeparator.

199.4.12 XSBuiltIns.SoapTimePrefix Constant

Pascal

```
SoapTimePrefix = 'T';
```

C++

```
SoapTimePrefix = 'T';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SoapTimePrefix.

199.4.13 XSBuiltIns.SPlus Constant

Pascal

```
SPlus: Char = '+';
```

C++

```
Char SPlus = '+';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SPlus.

199.4.14 XSBuiltIns.SSciNotationMarker Constant

Pascal

```
SSciNotationMarker: Char = 'E';
```

C++

```
Char SSciNotationMarker = 'E';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SSciNotationMarker.

199.4.15 XSBuiltIns.XMLDateSeparator Constant

Pascal

```
XMLDateSeparator = '-';
```

C++

```
XMLDateSeparator = '-';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDateSeparator.

199.4.16 XSBuiltIns.XMLDayPos Constant

Pascal

```
XMLDayPos = 9;
```

C++

```
XMLDayPos = 9;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDayPos.

199.4.17 XSBuiltIns.XMLDefaultYearDigits Constant

Pascal

```
XMLDefaultYearDigits = 4;
```

C++

```
XMLDefaultYearDigits = 4;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDefaultYearDigits.

199.4.18 XSBuiltIns.XMLDurationDay Constant

Pascal

```
XMLDurationDay = 'D';
```

C++

```
XMLDurationDay = 'D';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationDay.

199.4.19 XSBuiltIns.XMLDurationHour Constant

Pascal

```
XMLDurationHour = 'H';
```

C++

```
XMLDurationHour = 'H';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationHour.

199.4.20 XSBuiltIns.XMLDurationMinute Constant

Pascal

```
XMLDurationMinute = 'M';
```

C++

```
XMLDurationMinute = 'M';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationMinute.

199.4.21 XSBuiltIns.XMLDurationMonth Constant

Pascal

```
XMLDurationMonth = 'M';
```

C++

```
XMLDurationMonth = 'M';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationMonth.

199.4.22 XSBuiltIns.XMLDurationSecond Constant

Pascal

```
XMLDurationSecond = 'S';
```

C++

```
XMLDurationSecond = 'S';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationSecond.

199.4.23 XSBuiltIns.XMLDurationStart Constant

Pascal

```
XMLDurationStart = 'P';
```

C++

```
XMLDurationStart = 'P';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationStart.

199.4.24 XSBuiltIns.XMLDurationYear Constant

Pascal

```
XMLDurationYear = 'Y';
```

C++

```
XMLDurationYear = 'Y';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationYear.

199.4.25 XSBuiltIns.XMLHourOffsetMinusMarker Constant

Pascal

```
XMLHourOffsetMinusMarker = '-';
```

C++

```
XMLHourOffsetMinusMarker = '-';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLHourOffsetMinusMarker.

199.4.26 XSBuiltIns.XMLHourOffsetPlusMarker Constant

Pascal

```
XMLHourOffsetPlusMarker = '+';
```

C++

```
XMLHourOffsetPlusMarker = '+';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLHourOffsetPlusMarker.

199.4.27 XSBuiltIns.XMLMilSecPos Constant

Pascal

```
XMLMilSecPos = 10;
```

C++

```
XMLMilSecPos = 10;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLMilSecPos.

199.4.28 XSBuiltIns.XMLMonthPos Constant

Pascal

```
XMLMonthPos = 6;
```

C++

```
XMLMonthPos = 6;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLMonthPos.

199.4.29 XSBuiltIns.XMLTimeSeparator Constant

Pascal

```
XMLTimeSeparator = ':';
```

C++

```
XMLTimeSeparator = ':';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLTimeSeparator.

199.4.30 XSBuiltIns.XMLYearPos Constant

Pascal

```
XMLYearPos = 1;
```

C++

```
XMLYearPos = 1;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLYearPos.

200 XSLProd Namespace

This is namespace XSLProd.

Topics

Name	Description
Classes (see page 26588)	The following table lists classes in this documentation.

Classes

	Name	Description
	EXSLPageProducerError (see page 26588)	EXSLPageProducerError is the exception that is raised when TXSLPageProducer (see page 26605) encounters a problem.
	TCustomXSLPageProducer (see page 26588)	TCustomXSLPageProducer generates web page content by transforming data described with XML (Extensible Markup Language) using an XSL (Extensible Stylesheet Language) template.
	TXSLPageProducer (see page 26605)	TXSLPageProducer generates web page content by transforming data described with XML (Extensible Markup Language) using an XSL (Extensible Stylesheet Language) template.

200.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EXSLPageProducerError (see page 26588)	EXSLPageProducerError is the exception that is raised when TXSLPageProducer (see page 26605) encounters a problem.
	TCustomXSLPageProducer (see page 26588)	TCustomXSLPageProducer generates web page content by transforming data described with XML (Extensible Markup Language) using an XSL (Extensible Stylesheet Language) template.
	TXSLPageProducer (see page 26605)	TXSLPageProducer generates web page content by transforming data described with XML (Extensible Markup Language) using an XSL (Extensible Stylesheet Language) template.

200.1.1 EXSLPageProducerError Class

EXSLPageProducerError is the exception that is raised when TXSLPageProducer (see page 26605) encounters a problem.

Class Hierarchy

Pascal

```
EXSLPageProducerError = class(EWebBrokerException);
```

C++

```
class EXSLPageProducerError : public EWebBrokerException;
```

File

XSLProd

Description

TXSLPageProducer (see page 26605) raises an EXSLPageProducerError exception when it tries to produce content and the WebSnap application has not yet specified the source XML document or the XSL template that transforms that source into the final content.

See Also

TXSLPageProducer (see page 26605)

200.1.2 TCustomXSLPageProducer Class

TCustomXSLPageProducer generates web page content by transforming data described with XML (Extensible Markup Language) using an XSL (Extensible Stylesheet Language) template.

Class Hierarchy

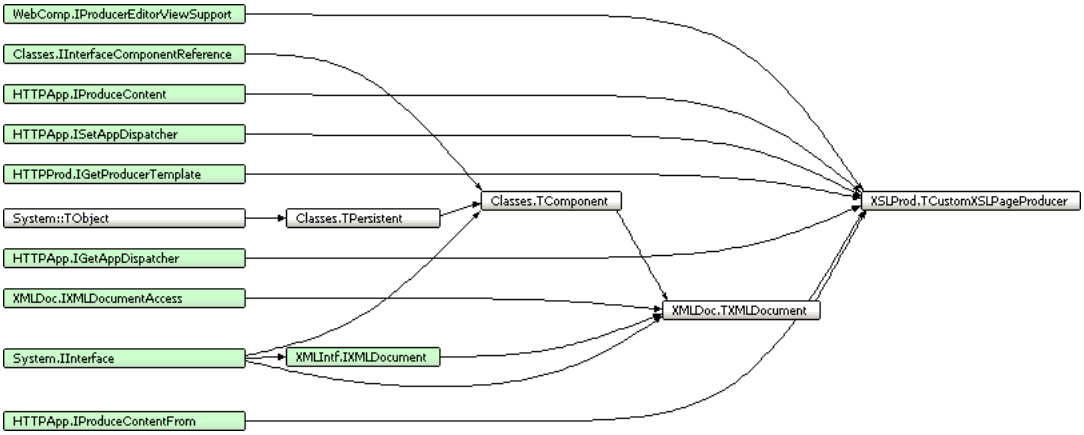

Pascal

```
TCustomXSLPageProducer = class(TXMLDocument, IProduceContent, IProduceContentFrom,
ISetAppDispatcher, IGetAppDispatcher, IProducerEditorViewSupport, IGetProducerTemplate);
```

C++

```
class TCustomXSLPageProducer : public TXMLDocument, public IProduceContent, public
IProduceContentFrom, public ISetAppDispatcher, public IGetAppDispatcher, public
IProducerEditorViewSupport, public IGetProducerTemplate;
```

File

XSLProd

Description

TCustomXSLPageProducer is the base class for components that transform the data in an XML document using an XSL template. This class introduces all the properties and methods necessary for specifying the XML document to transform, the XSL template that describes the transformation, and for performing the transformation.

The XMLData (see page 26605) property identifies the XML data that this component transforms. XMLData (see page 26605) must be set to a component that supports either the IXMLDocument interface (for example, TXMLDocument or one of its descendants) or the IGetXMLStream interface (for example, TXMLBroker).

To obtain the results of the transformation, you can use any of the following methods:

The Content (see page 26597) method performs the transformation using the XSL template that is specified by the FileName or XML property. When using the Content (see page 26597) method, you can also read an XSL document by calling the LoadFromFile, LoadFromStream, or LoadFromXML method first.

The ContentFromString (see page 26598) and ContentFromWideString (see page 26599) methods perform the transformation using an XSL template that is supplied as a parameter.

The ContentFromStream (see page 26598) method performs the transformation using an XSL template that it reads from a specified stream.

200.1.2.1 TCustomXSLPageProducer Members

The following tables list the members exposed by TCustomXSLPageProducer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_AddRef (see page 26106)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	_Release (see page 26106)	Called when an application releases a component interface.
	AssignParseOptions (see page 26108)	This is AssignParseOptions, a member of class TXMLDocument.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckActive (see page 26108)	This is CheckActive, a member of class TXMLDocument.
	CheckAutoSave (see page 26108)	This is CheckAutoSave, a member of class TXMLDocument.
	CheckDOM (see page 26109)	This is CheckDOM, a member of class TXMLDocument.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.
V	DefineProperties (see page 26111)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoAfterClose (see page 26111)	This is DoAfterClose, a member of class TXMLDocument.
	DoAfterOpen (see page 26111)	This is DoAfterOpen, a member of class TXMLDocument.
	DoBeforeClose (see page 26112)	This is DoBeforeClose, a member of class TXMLDocument.
	DoBeforeOpen (see page 26112)	This is DoBeforeOpen, a member of class TXMLDocument.
	DoNodeChange (see page 26112)	This is DoNodeChange, a member of class TXMLDocument.
V	GetActive (see page 26113)	Returns the value of the Active property.
	GetAppDispatcher (see page 26599)	This is GetAppDispatcher, a member of class TCustomXSLPageProducer.
	GetAsyncLoadState (see page 26113)	Returns the value of the AsyncLoadState property.
V	GetChildNodeClass (see page 26113)	This is GetChildNodeClass, a member of class TXMLDocument.
	GetChildNodes (see page 26113)	Returns the value of the ChildNodes property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDocumentElement (see page 26114)	Returns the value of the DocumentElement property.
	GetDocumentNode (see page 26115)	Returns the value of the Node property.
	GetDocumentObject (see page 26115)	This is GetDocumentObject, a member of class TXMLDocument.
	GetDOMDocument (see page 26115)	Returns the value of the DOMDocument property.
	GetDOMPersist (see page 26115)	This is GetDOMPersist, a member of class TXMLDocument.
	GetEncoding (see page 26116)	Returns the value of the Encoding property.
	GetFileName (see page 26116)	Returns the value of the FileName property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetModified (see page 26116)	Returns the value of the Modified property.
	GetNodeIndentStr (see page 26117)	Returns the value of the NodeIndentStr property.
	GetOptions (see page 26117)	Returns the value of the Options property.
V	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetParseOptions (see page 26117)	Returns the value of the ParseOptions property.
	GetProducerTemplateFileName (see page 26599)	This is GetProducerTemplateFileName, a member of class TCustomXSLPageProducer.
	GetProducerTemplateStream (see page 26599)	This is GetProducerTemplateStream, a member of class TCustomXSLPageProducer.

	GetPrologNode (see page 26117)	This is GetPrologNode, a member of class TXMLDocument.
	GetPrologValue (see page 26118)	This is GetPrologValue, a member of class TXMLDocument.
	GetSchemaRef (see page 26118)	Returns the value of the SchemaRef property.
	GetStandAlone (see page 26118)	Returns the value of the StandAlone property.
	GetTemplateFileType (see page 26600)	This is GetTemplateFileType, a member of class TCustomXSLPageProducer.
	GetTemplateStream (see page 26600)	This is GetTemplateStream, a member of class TCustomXSLPageProducer.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVersion (see page 26118)	Returns the value of the Version property.
	GetXML (see page 26119)	Returns the value of the XML property.
	GetXMLData (see page 26600)	This is GetXMLData, a member of class TCustomXSLPageProducer.
	GetXMLDocument (see page 26600)	This is GetXMLDocument, a member of class TCustomXSLPageProducer.
	GetXSLData (see page 26601)	This is GetXSLData, a member of class TCustomXSLPageProducer.
	HasHTMLBrowserView (see page 26601)	This is HasHTMLBrowserView, a member of class TCustomXSLPageProducer.
	HasScriptView (see page 26601)	This is HasScriptView, a member of class TCustomXSLPageProducer.
	HasXMLBrowserView (see page 26601)	This is HasXMLBrowserView, a member of class TCustomXSLPageProducer.
	HasXSLBrowserView (see page 26601)	This is HasXSLBrowserView, a member of class TCustomXSLPageProducer.
	InternalSetPrologValue (see page 26119)	This is InternalSetPrologValue, a member of class TXMLDocument.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.
	LoadData (see page 26120)	This is LoadData, a member of class TXMLDocument.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 26120)	Initializes the component after the form file has been read into memory.
V	Notification (see page 26602)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 26602)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 26602)	This is ProduceContentFromStream, a member of class TCustomXSLPageProducer.
	ProduceContentFromString (see page 26603)	This is ProduceContentFromString, a member of class TCustomXSLPageProducer.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadDOMVendor (see page 26123)	This is ReadDOMVendor, a member of class TXMLDocument.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReleaseDoc (see page 26124)	This is ReleaseDoc, a member of class TXMLDocument.
	SaveToUTF8String (see page 26126)	This is SaveToUTF8String, a member of class TXMLDocument.
	SaveToXMLStrings (see page 26127)	This is SaveToXMLStrings, a member of class TXMLDocument.
V	SetActive (see page 26127)	Sets the value of the Active property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 26603)	This is SetAppDispatcher, a member of class TCustomXSLPageProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDocumentElement (see page 26127)	Sets the value of the DocumentElement property.
	SetDOMDocument (see page 26128)	Sets the value of the DOMDocument property.
	SetEncoding (see page 26128)	Sets the value of the Encoding property.
	SetFileName (see page 26128)	Sets the value of the FileName property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetModified (see page 26129)	This is SetModified, a member of class TXMLDocument.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetNodeIndentStr (see page 26129)	Sets the value of the NodeIndentStr property.
	SetOnAsyncLoad (see page 26129)	Supplies an event handler that responds when the AsyncLoadState property changes.
	SetOptions (see page 26130)	Sets the value of the Options property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.

	SetParseOptions (see page 26130)	Sets the value of the ParseOptions property.
	SetPrologValue (see page 26130)	This is SetPrologValue, a member of class TXMLDocument.
	SetStandAlone (see page 26130)	Sets the value of the StandAlone property.
	SetVersion (see page 26131)	Sets the value of the Version property.
	SetXML (see page 26131)	Sets the value of the XML property.
	SetXMLStrings (see page 26131)	This is SetXMLStrings, a member of class TXMLDocument.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteDOMVendor (see page 26132)	This is WriteDOMVendor, a member of class TXMLDocument.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.
	XMLStringsChanging (see page 26132)	This is XMLStringsChanging, a member of class TXMLDocument.

Protected Properties

	Name	Description
R	DocBindingInfo (see page 26135)	This is DocBindingInfo, a member of class TXMLDocument.
	DocSource (see page 26135)	This is DocSource, a member of class TXMLDocument.
R	DOMParseOptions (see page 26137)	This is DOMParseOptions, a member of class TXMLDocument.
R	DOMPersist (see page 26137)	This is DOMPersist, a member of class TXMLDocument.
	PrefixID (see page 26142)	This is PrefixID, a member of class TXMLDocument.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26107)	This is the overview for the AddChild method overload.
	AddChild (see page 26385)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 26107)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Content (see page 26597)	Returns the transformed value of the XML supplied by the XMLData (see page 26605) property.
V	ContentFromStream (see page 26598)	Returns the transformed value of the XML supplied by the XMLData (see page 26605) property using an XSL template that can be read from a specified stream.
	ContentFromString (see page 26598)	Returns the transformed value of the XML supplied by the XMLData (see page 26605) property using a specified XSL template.
V	ContentFromWideString (see page 26599)	Returns the transformed value of the XML supplied by the XMLData (see page 26605) property using a specified XSL template.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.

	Create (see page 26103)	This is the overview for the Create constructor overload.
	CreateElement (see page 26109)	Creates a new element node that is associated with this document.
	CreateElement (see page 26386)	Creates a new element node that is associated with this document.
	CreateNode (see page 26109)	Creates a new node that is associated with this document.
	CreateNode (see page 26387)	Creates a new node that is associated with this document.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26104)	Disposes of a TXMLDocument (see page 26097) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GeneratePrefix (see page 26112)	Generates a unique namespace prefix that can be used for a node in the document.
	GeneratePrefix (see page 26388)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 26388)	Returns the value of the Active (see page 26403) property.
	GetAsyncLoadState (see page 26389)	Returns the value of the AsyncLoadState (see page 26404) property.
	GetChildNodes (see page 26389)	Returns the value of the ChildNodes (see page 26405) property.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetDocBinding (see page 26114)	Replaces the document element with an instance of a specified implementation class.
	GetDocBinding (see page 26389)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 26390)	Returns the value of the DocumentElement (see page 26405) property.
	GetDocumentNode (see page 26390)	Returns the value of the Node (see page 26407) property.
	GetDocumentObject (see page 26229)	This is GetDocumentObject, a member of class IXMLDocumentAccess.
	GetDOMDocument (see page 26390)	Returns the value of the DOMDocument (see page 26405) property.
	GetDOMPersist (see page 26229)	This is GetDOMPersist, a member of class IXMLDocumentAccess.
	GetEncoding (see page 26391)	Returns the value of the Encoding (see page 26406) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFileName (see page 26391)	Returns the value of the FileName (see page 26406) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetModified (see page 26391)	Returns the value of the Modified (see page 26407) property.
V	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNodeIndentStr (see page 26392)	Returns the value of the NodeIndentStr (see page 26408) property.
	GetOptions (see page 26392)	Returns the value of the Options (see page 26408) property.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParseOptions (see page 26392)	Returns the value of the ParseOptions (see page 26409) property.
	GetSchemaRef (see page 26392)	Returns the value of the SchemaRef (see page 26409) property.
	GetStandAlone (see page 26393)	Returns the value of the StandAlone (see page 26410) property.

	GetVersion (see page 26393)	Returns the value of the Version (see page 26410) property.
	GetXML (see page 26393)	Returns the value of the XML (see page 26410) property.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	IsEmptyDoc (see page 26119)	Indicates whether the TXMLDocument (see page 26097) instance represents an empty document.
	IsEmptyDoc (see page 26394)	Indicates whether the IXMLDocument (see page 26382) instance represents an empty document.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LoadFromFile (see page 26121)	Loads an XML (see page 26144) document and activates it.
	LoadFromFile (see page 26394)	Loads an XML (see page 26410) document from disk.
	LoadFromStream (see page 26121)	Loads an XML (see page 26144) document from a stream and activates it.
	LoadFromStream (see page 26395)	Loads an XML (see page 26410) document from a stream and activates it.
	LoadFromXML (see page 26122)	This is the overview for the LoadFromXML method overload.
	LoadFromXML (see page 26395)	This is the overview for the LoadFromXML method overload.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	NewInstance (see page 26122)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 26123)	Updates the parsed XML (see page 26144) document to reflect external changes.
	Refresh (see page 26396)	Updates the parsed XML (see page 26410) document to reflect external changes.
	RegisterDocBinding (see page 26124)	Registers an implementation class for child nodes with a specified tag name.
	RegisterDocBinding (see page 26396)	Registers an implementation class for child nodes with a specified tag name.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Resync (see page 26125)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	Resync (see page 26397)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 26125)	Saves the XML (see page 26144) document to disk.
	SaveToFile (see page 26397)	Saves the XML (see page 26410) document to disk.
	SaveToStream (see page 26125)	Saves the XML (see page 26144) document to a stream.
	SaveToStream (see page 26398)	Saves the XML (see page 26410) document to a stream.
	SaveToXML (see page 26126)	This is the overview for the SaveToXML method overload.
	SaveToXML (see page 26398)	This is the overview for the SaveToXML method overload.
	SetActive (see page 26399)	Sets the value of the Active (see page 26403) property.
	SetDocumentElement (see page 26399)	Sets the value of the DocumentElement (see page 26405) property.
	SetDOMDocument (see page 26399)	Sets the value of the DOMDocument (see page 26405) property.
	SetEncoding (see page 26400)	Sets the value of the Encoding (see page 26406) property.
	SetFileName (see page 26400)	Sets the value of the FileName (see page 26406) property.
	SetNodeIndentStr (see page 26400)	Sets the value of the NodeIndentStr (see page 26408) property.
	SetOnAsyncLoad (see page 26401)	Supplies an event handler that responds when the AsyncLoadState (see page 26404) property changes.

	SetOptions (see page 26401)	Sets the value of the Options (see page 26408) property.
	SetParseOptions (see page 26402)	Sets the value of the ParseOptions (see page 26409) property.
	SetStandAlone (see page 26402)	Sets the value of the StandAlone (see page 26410) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.
	SetVersion (see page 26402)	Sets the value of the Version (see page 26410) property.
	SetXML (see page 26403)	Sets the value of the XML (see page 26410) property.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Active (see page 26403)	Specifies whether the XML (see page 26410) document has been parsed so that the DOM interfaces are available to examine or change its contents.
R	AsyncLoadState (see page 26134)	Indicates the state of the DOM parser when it parses the XML (see page 26144) document asynchronously.
R	AsyncLoadState (see page 26404)	Indicates the state of the DOM parser when it parses the XML (see page 26410) document asynchronously.
R	ChildNodes (see page 26135)	Lists all of the child nodes of the document.
R	ChildNodes (see page 26405)	Lists all of the child nodes of the document.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 26603)	Specifies the interface of a component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.
R	DispatcherComponent (see page 26604)	Specifies the component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.
	DocumentElement (see page 26135)	Provides access to the root node of the XML (see page 26144) document.
	DocumentElement (see page 26405)	Provides access to the root node of the XML (see page 26410) document.
R	DocumentObject (see page 26229)	This is DocumentObject, a member of class IXMLDocumentAccess.
	DOMDocument (see page 26136)	Provides access to the DOM interface for the parsed XML (see page 26144) document.
	DOMDocument (see page 26405)	Provides access to the DOM interface for the parsed XML (see page 26410) document.
	DOMImplementation (see page 26137)	This is DOMImplementation, a member of class TXMLDocument.
R	DOMPersist (see page 26230)	This is DOMPersist, a member of class IXMLDocumentAccess.
	Encoding (see page 26138)	Indicates the character set that the XML (see page 26144) document uses to encode characters.
	Encoding (see page 26406)	Indicates the character set that the XML (see page 26410) document uses to encode characters.
	FileName (see page 26406)	Indicates the XML (see page 26410) document file, if any, that this interface represents.
R	Modified (see page 26139)	Indicates whether the XML (see page 26144) document has been modified.
R	Modified (see page 26407)	Indicates whether the XML (see page 26410) document has been modified.
R	Node (see page 26140)	Provides access to the document node for the XML (see page 26144) document.
R	Node (see page 26407)	Provides access to the document node for the XML (see page 26410) document.
	NodeIndentStr (see page 26408)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 26410) text.
	NSPrefixBase (see page 26141)	Specifies the base string for automatically-generated namespace prefixes.
	Options (see page 26408)	Influences various aspects of the document's behavior.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.

	ParseOptions (see page 26409)	Specifies implementation-specific options that are followed when parsing the XML (see page 26410) document.
R	SchemaRef (see page 26142)	Returns the name of the schema associated with this XML (see page 26144) document.
R	SchemaRef (see page 26409)	Returns the name of the schema associated with this XML (see page 26410) document.
	StandAlone (see page 26143)	Indicates whether the XML (see page 26144) document includes any external markup declarations.
	StandAlone (see page 26410)	Indicates whether the XML (see page 26410) document includes any external markup declarations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Version (see page 26143)	Indicates the XML (see page 26144) version that the document uses.
	Version (see page 26410)	Indicates the XML (see page 26410) version that the document uses.
	XML (see page 26410)	Specifies the content of the XML document associated with this IXMLDocument (see page 26382) interface.
	XMLData (see page 26605)	Specifies the component that supplies the XML data that this component transforms.

Published Events

	Name	Description
	AfterClose (see page 26144)	Occurs after the XML (see page 26144) document is closed.
	AfterNodeChange (see page 26145)	Occurs after a change is made to one of the nodes in the document.
	AfterOpen (see page 26145)	Occurs immediately after the XML (see page 26144) document is opened.
	BeforeClose (see page 26146)	Occurs immediately before the XML (see page 26144) document is closed.
	BeforeNodeChange (see page 26146)	Occurs before a change is made to one of the nodes in the document.
	BeforeOpen (see page 26147)	Occurs immediately before the XML (see page 26144) document is opened.
	OnAsyncLoad (see page 26147)	Occurs at various stages during the asynchronous parsing of an XML (see page 26144) document.

Published Properties

	Name	Description
	Active (see page 26133)	Specifies whether the XML (see page 26144) document has been parsed so that the DOM interfaces are available to examine or change its contents.
	DOMVendor (see page 26137)	Specifies the DOM implementation to use for parsing and manipulating the XML (see page 26144) document.
	FileName (see page 26138)	Specifies the file name for the XML (see page 26144) document associated with this TXMLDocument (see page 26097) object.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NodeIndentStr (see page 26140)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 26144) text.
	Options (see page 26141)	Influences various aspects of the document's behavior.
	ParseOptions (see page 26142)	Specifies implementation-specific options that are followed when parsing the XML (see page 26144) document.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XML (see page 26144)	Specifies the content of the XML document associated with this TXMLDocument (see page 26097) object.

200.1.2.2 TCustomXSLPageProducer Methods

The methods of the TCustomXSLPageProducer class are listed here.

Protected Methods

	Name	Description
	GetAppDispatcher (see page 26599)	This is GetAppDispatcher, a member of class TCustomXSLPageProducer.
	GetProducerTemplateFileName (see page 26599)	This is GetProducerTemplateFileName, a member of class TCustomXSLPageProducer.
	GetProducerTemplateStream (see page 26599)	This is GetProducerTemplateStream, a member of class TCustomXSLPageProducer.

	GetTemplateFileType (see page 26600)	This is GetTemplateFileType, a member of class TCustomXSLPageProducer.
	GetTemplateStream (see page 26600)	This is GetTemplateStream, a member of class TCustomXSLPageProducer.
	GetXMLData (see page 26600)	This is GetXMLData, a member of class TCustomXSLPageProducer.
	GetXMLDocument (see page 26600)	This is GetXMLDocument, a member of class TCustomXSLPageProducer.
	GetXSLData (see page 26601)	This is GetXSLData, a member of class TCustomXSLPageProducer.
	HasHTMLBrowserView (see page 26601)	This is HasHTMLBrowserView, a member of class TCustomXSLPageProducer.
	HasScriptView (see page 26601)	This is HasScriptView, a member of class TCustomXSLPageProducer.
	HasXMLBrowserView (see page 26601)	This is HasXMLBrowserView, a member of class TCustomXSLPageProducer.
	HasXSLBrowserView (see page 26601)	This is HasXSLBrowserView, a member of class TCustomXSLPageProducer.
V	Notification (see page 26602)	Forwards notification messages to all owned components.
	ProduceContent (see page 26602)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 26602)	This is ProduceContentFromStream, a member of class TCustomXSLPageProducer.
	ProduceContentFromString (see page 26603)	This is ProduceContentFromString, a member of class TCustomXSLPageProducer.
	SetAppDispatcher (see page 26603)	This is SetAppDispatcher, a member of class TCustomXSLPageProducer.

Public Methods

	Name	Description
V	Content (see page 26597)	Returns the transformed value of the XML supplied by the XMLData (see page 26605) property.
V	ContentFromStream (see page 26598)	Returns the transformed value of the XML supplied by the XMLData (see page 26605) property using an XSL template that can be read from a specified stream.
	ContentFromString (see page 26598)	Returns the transformed value of the XML supplied by the XMLData (see page 26605) property using a specified XSL template.
V	ContentFromWideString (see page 26599)	Returns the transformed value of the XML supplied by the XMLData (see page 26605) property using a specified XSL template.

200.1.2.2.1 TCustomXSLPageProducer.Content Method

Returns the transformed value of the XML supplied by the XMLData (see page 26605) property.

Pascal

```
function Content: string; virtual;
```

C++

```
virtual __fastcall AnsiString Content();
```

Description

Call Content to retrieve the content for using in an HTTP response message. Content transforms the XML data that is supplied by the XMLData (see page 26605) property using the XSL template associated with this component. The XSL template can be specified in several ways:

If the FileName property is set, it specifies a file that contains the XSL template.

If FileName is not set, you can assign an XSL template using the XML property.

At runtime, you can supply an XSL template by calling the LoadFromFile, LoadFromStream, or LoadFromXML method first.

See Also

ContentFromStream (see page 26598)

ContentFromString (see page 26598)

ContentFromWideString (see page 26599)

XMLData (see page 26605)

FileName (see page 26138)

XML (see page 26144)

LoadFromFile (see page 26121)

LoadFromStream (see page 26121)

LoadFromXML (see page 26122)

200.1.2.2.2 TCustomXSLPageProducer.ContentFromStream Method

Returns the transformed value of the XML supplied by the XMLData (see page 26605) property using an XSL template that can be read from a specified stream.

Pascal

```
function ContentFromStream(InStream: TStream): string; virtual;
```

C++

```
virtual __fastcall AnsiString ContentFromStream(TStream InStream);
```

Description

Call Content (see page 26597) to retrieve the content for using in an HTTP response message. Content (see page 26597) transforms the XML data that is supplied by the XMLData (see page 26605) property using the XSL template that is reads from InStream.

See Also

Content (see page 26597)

ContentFromString (see page 26598)

ContentFromWideString (see page 26599)

XMLData (see page 26605)

200.1.2.2.3 TCustomXSLPageProducer.ContentFromString Method

Returns the transformed value of the XML supplied by the XMLData (see page 26605) property using a specified XSL template.

Pascal

```
function ContentFromString(const S: string): string;
```

C++

```
__fastcall AnsiString ContentFromString(const AnsiString S);
```

Description

Call Content (see page 26597) to retrieve the content for using in an HTTP response message. Content (see page 26597) transforms the XML data that is supplied by the XMLData (see page 26605) property using the XSL template specified by S.

See Also

Content (see page 26597)

ContentFromStream (see page 26598)

ContentFromWideString (see page 26599)

XMLData (see page 26605)

200.1.2.2.4 TCustomXSLPageProducer.ContentFromWideString Method

Returns the transformed value of the XML supplied by the XMLData (see page 26605) property using a specified XSL template.

Pascal

```
function ContentFromWideString(const S: WideString): string; virtual;
```

C++

```
virtual __fastcall AnsiString ContentFromWideString(const BSTR S);
```

Description

Call Content (see page 26597) to retrieve the content for using in an HTTP response message. Content (see page 26597) transforms the XML data that is supplied by the XMLData (see page 26605) property using the XSL template specified by S.

See Also

Content (see page 26597)

ContentFromStream (see page 26598)

ContentFromString (see page 26598)

XMLData (see page 26605)

200.1.2.2.5 TCustomXSLPageProducer.GetAppDispatcher Method

Pascal

```
function GetAppDispatcher: TComponent;
```

C++

```
__fastcall TComponent * GetAppDispatcher();
```

Description

This is GetAppDispatcher, a member of class TCustomXSLPageProducer.

200.1.2.2.6 TCustomXSLPageProducer.GetProducerTemplateFileName Method

Pascal

```
function GetProducerTemplateFileName: string;
```

C++

```
__fastcall AnsiString GetProducerTemplateFileName();
```

Description

This is GetProducerTemplateFileName, a member of class TCustomXSLPageProducer.

200.1.2.2.7 TCustomXSLPageProducer.GetProducerTemplateStream Method

Pascal

```
function GetProducerTemplateStream(out AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream GetProducerTemplateStream(Boolean AOwned);
```

Description

This is GetProducerTemplateStream, a member of class TCustomXSLPageProducer.

200.1.2.2.8 TCustomXSLPageProducer.GetTemplateFileType Method

Pascal

```
function GetTemplateFileType: string;
```

C++

```
__fastcall AnsiString GetTemplateFileType();
```

Description

This is GetTemplateFileType, a member of class TCustomXSLPageProducer.

200.1.2.2.9 TCustomXSLPageProducer.GetTemplateStream Method

Pascal

```
function GetTemplateStream(out AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream GetTemplateStream(Boolean AOwned);
```

Description

This is GetTemplateStream, a member of class TCustomXSLPageProducer.

200.1.2.2.10 TCustomXSLPageProducer.GetXMLData Method

Pascal

```
function GetXMLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXMLData(Boolean Owned);
```

Description

This is GetXMLData, a member of class TCustomXSLPageProducer.

200.1.2.2.11 TCustomXSLPageProducer.GetXMLDocument Method

Pascal

```
function GetXMLDocument(out AXMLDocument: IXMLDocument): Boolean;
```

C++

```
__fastcall Boolean GetXMLDocument(IXMLDocument AXMLDocument);
```

Description

This is GetXMLDocument, a member of class TCustomXSLPageProducer.

200.1.2.2.12 TCustomXSLPageProducer.GetXSLData Method

Pascal

```
function GetXSLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXSLData(Boolean Owned);
```

Description

This is GetXSLData, a member of class TCustomXSLPageProducer.

200.1.2.2.13 TCustomXSLPageProducer.HasHTMLBrowserView Method

Pascal

```
function HasHTMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasHTMLBrowserView();
```

Description

This is HasHTMLBrowserView, a member of class TCustomXSLPageProducer.

200.1.2.2.14 TCustomXSLPageProducer.HasScriptView Method

Pascal

```
function HasScriptView: Boolean;
```

C++

```
__fastcall Boolean HasScriptView();
```

Description

This is HasScriptView, a member of class TCustomXSLPageProducer.

200.1.2.2.15 TCustomXSLPageProducer.HasXMLBrowserView Method

Pascal

```
function HasXMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXMLBrowserView();
```

Description

This is HasXMLBrowserView, a member of class TCustomXSLPageProducer.

200.1.2.2.16 TCustomXSLPageProducer.HasXSLBrowserView Method

Pascal

```
function HasXSLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXSLBrowserView();
```

Description

This is HasXSLBrowserView, a member of class TCustomXSLPageProducer.

200.1.2.2.17 TCustomXSLPageProducer.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification (see page 2775)

InsertComponent (see page 2780)

RemoveComponent (see page 2785)

Owner (see page 2807)

200.1.2.2.18 TCustomXSLPageProducer.ProduceContent Method

Returns the content that the producer creates.

Pascal

```
function ProduceContent: string;
```

C++

```
__fastcall AnsiString ProduceContent();
```

Description

Call ProduceContent to retrieve the content for using in an HTTP response message. In most cases, this content is a string of HTML commands defining a Web page, although it can be any valid MIME type.

See Also

Content (see page 15106)

200.1.2.2.19 TCustomXSLPageProducer.ProduceContentFromStream Method

Pascal

```
function ProduceContentFromStream(Stream: TStream): string;
```

C++

```
__fastcall AnsiString ProduceContentFromStream(TStream Stream);
```

Description

This is ProduceContentFromStream, a member of class TCustomXSLPageProducer.

200.1.2.2.20 TCustomXSLPageProducer.ProduceContentFromString Method

Pascal

```
function ProduceContentFromString(const S: string): string;
```

C++

```
__fastcall AnsiString ProduceContentFromString(const AnsiString S);
```

Description

This is ProduceContentFromString, a member of class TCustomXSLPageProducer.

200.1.2.2.21 TCustomXSLPageProducer.SetAppDispatcher Method

Pascal

```
procedure SetAppDispatcher(const ADispatcher: TComponent);
```

C++

```
__fastcall SetAppDispatcher(const TComponent * ADispatcher);
```

Description

This is SetAppDispatcher, a member of class TCustomXSLPageProducer.

200.1.2.3 TCustomXSLPageProducer Properties

The properties of the TCustomXSLPageProducer class are listed here.

Public Properties

	Name	Description
R	Dispatcher (see page 26603)	Specifies the interface of a component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.
R	DispatcherComponent (see page 26604)	Specifies the component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.
	XMLData (see page 26605)	Specifies the component that supplies the XML data that this component transforms.

200.1.2.3.1 TCustomXSLPageProducer.Dispatcher Property

Specifies the interface of a component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.

Pascal

```
property Dispatcher: IWebDispatcherAccess;
```

C++

```
__property IWebDispatcherAccess Dispatcher;
```

Description

Read Dispatcher to access the interface of the dispatcher. In Web Broker applications, this is the interface of a TWebDispatcher or TWebModule component. In WebSnap applications, this is the interface of a TWebAppComponents component.

The IWebDispatcherAccess interface has two methods:

Request, which returns the HTTP request message to which this page producer responds.

Response, which returns the HTTP response message for which this page producer generates content.

See Also

TWebModule (see page 15154)

TWebDispatcher (see page 15147)

TWebAppComponents (see page 24958)

TWebRequest (see page 15161)

TWebResponse (see page 15191)

DispatcherComponent (see page 26604)

200.1.2.3.2 TCustomXSLPageProducer.DispatcherComponent Property

Specifies the component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.

Pascal

```
property DispatcherComponent: TComponent;
```

C++

```
__property TComponent * DispatcherComponent;
```

Description

Read DispatcherComponent to access the dispatcher that handles the HTTP request and response messages for this Web Server application. In Web Broker applications, this is the interface of a TWebDispatcherComponent or TWebModule component. In WebSnap applications, this is the interface of a TWebAppComponents component.

Dispatcher (see page 26603) implements the interface that is the value of the Dispatcher (see page 26603) property. The XSL page producer uses this property so that it can respond to notifications when the component is freed. To use the dispatcher, use the Dispatcher (see page 26603) property instead.

See Also

TWebModule (see page 15154)

TWebDispatcher (see page 15147)

TWebAppComponents (see page 24958)

TWebRequest (see page 15161)

TWebResponse (see page 15191)

Dispatcher (see page 26603)

200.1.2.3.3 TCustomXSLPageProducer.XMLData Property

Specifies the component that supplies the XML data that this component transforms.

Pascal

```
property XMLData: TComponent;
```

C++

```
__property TComponent * XMLData;
```

Description

Use the XMLData property to specify the XML data that this component transforms. XMLData must be set to a component that supports either the IXMLDocument interface (for example, TXMLDocument or one of its descendants) or the IGetXMLStream interface (for example, TXMLBroker).

See Also

TXMLDocument (see page 26097)

TXMLBroker (see page 26049)

200.1.3 TXSLPageProducer Class

TXSLPageProducer generates web page content by transforming data described with XML (Extensible Markup Language) using an XSL (Extensible Stylesheet Language) template.

Class Hierarchy

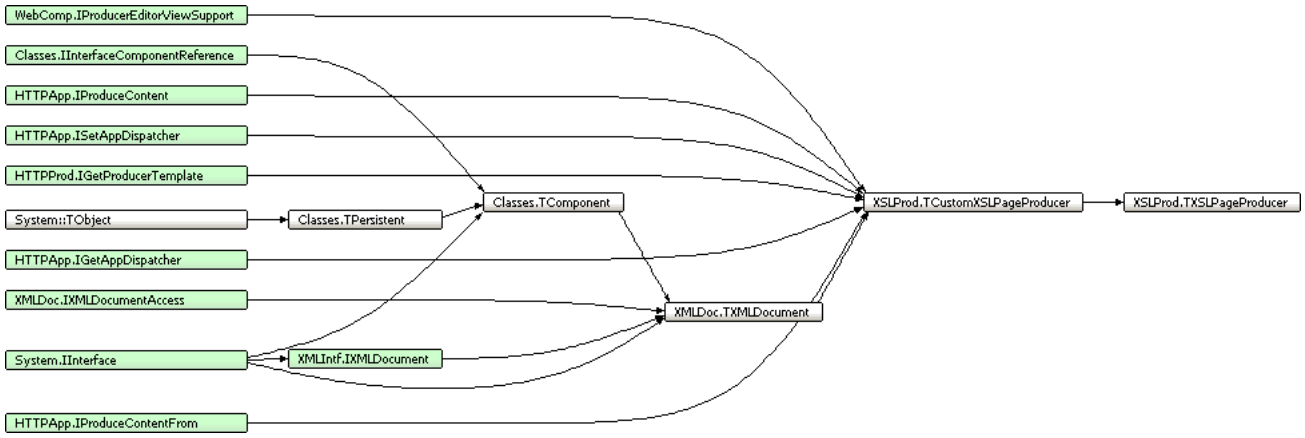

Pascal

```
TXSLPageProducer = class(TCustomXSLPageProducer);
```

C++

```
class TXSLPageProducer : public TCustomXSLPageProducer;
```

File

XSLProd

Description

Use the Web Page Module wizard to create a web page that uses TXSLPageProducer. To produce content, TXSLPageProducer uses an XSL template to transform the data in an XML document. Typically, the XSL transforms the XML document into another XML document that is HTML compliant (that is, into a document whose tags are actually HTML tags). However, as specified by the XSL template, TXSLPageProducer can return any arbitrary output as a result of the transformation.

Use the XMLData (see page 26613) property to specify the XML data that this component transforms. XMLData (see page 26613) must be set to a component that supports either the IXMLDocument interface (for example, TXMLDocument or one of its descendants) or the IGetXMLStream interface (for example, TXMLBroker).

To obtain the results of the transformation, you can use any of the following methods:

The Content (see page 26597) method performs the transformation using the XSL template that is specified by the FileName or XML property. When using the Content (see page 26597) method, you can also read an XSL document by calling the LoadFromFile, LoadFromStream, or LoadFromXML method first.

The ContentFromString (see page 26598) and ContentFromWideString (see page 26599) methods perform the transformation using an XSL template that is supplied as a parameter.

The ContentFromStream (see page 26598) method performs the transformation using an XSL template that it reads from a specified stream.

At design time, Web page modules that use the TXMLPageProducer have editor views called XML Tree and XSL Tree. These views provide a read only tree of the XML and XSL respectively.

See Also

TXMLDocument (see page 26097)

TXMLBroker (see page 26049)

200.1.3.1 TXSLPageProducer Members

The following tables list the members exposed by TXSLPageProducer.

Protected Fields

	Name	Description
	FComponentStyle (see page 2767)	This is FComponentStyle, a member of class TComponent.

Protected Methods

	Name	Description
	_AddRef (see page 2769)	Called when an application uses a component interface.
	_AddRef (see page 26106)	Called when an application uses a component interface.
	_Release (see page 2769)	Called when an application releases a component interface.
	_Release (see page 26106)	Called when an application releases a component interface.
	AssignParseOptions (see page 26108)	This is AssignParseOptions, a member of class TXMLDocument.
V	AssignTo (see page 2931)	Copies the properties of an object to a destination object.
	ChangeName (see page 2770)	Sets the private, internal storage for the Name (see page 2805) property to the string passed in NewName.
	CheckActive (see page 26108)	This is CheckActive, a member of class TXMLDocument.
	CheckAutoSave (see page 26108)	This is CheckAutoSave, a member of class TXMLDocument.
	CheckDOM (see page 26109)	This is CheckDOM, a member of class TXMLDocument.
V	DefineProperties (see page 2771)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	DefineProperties (see page 2931)	Provides an interface for a method that reads and writes otherwise unpublished data.

	DefineProperties (see page 26111)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoAfterClose (see page 26111)	This is DoAfterClose, a member of class TXMLDocument.
	DoAfterOpen (see page 26111)	This is DoAfterOpen, a member of class TXMLDocument.
	DoBeforeClose (see page 26112)	This is DoBeforeClose, a member of class TXMLDocument.
	DoBeforeOpen (see page 26112)	This is DoBeforeOpen, a member of class TXMLDocument.
	DoNodeChange (see page 26112)	This is DoNodeChange, a member of class TXMLDocument.
	GetActive (see page 26113)	Returns the value of the Active property.
	GetAppDispatcher (see page 26599)	This is GetAppDispatcher, a member of class TCustomXSLPageProducer.
	GetAsyncLoadState (see page 26113)	Returns the value of the AsyncLoadState property.
	GetChildNodeClass (see page 26113)	This is GetChildNodeClass, a member of class TXMLDocument.
	GetChildNodes (see page 26113)	Returns the value of the ChildNodes property.
	GetChildOwner (see page 2776)	Returns the owner of a child component being read from a stream.
	GetChildParent (see page 2776)	Returns the parent, or if there is no parent, returns the owner of a child component being read from a stream.
	GetChildren (see page 2777)	This is GetChildren, a member of class TComponent.
	GetDocumentElement (see page 26114)	Returns the value of the DocumentElement property.
	GetDocumentNode (see page 26115)	Returns the value of the Node property.
	GetDocumentObject (see page 26115)	This is GetDocumentObject, a member of class TXMLDocument.
	GetDOMDocument (see page 26115)	Returns the value of the DOMDocument property.
	GetDOMPersist (see page 26115)	This is GetDOMPersist, a member of class TXMLDocument.
	GetEncoding (see page 26116)	Returns the value of the Encoding property.
	GetFileName (see page 26116)	Returns the value of the FileName property.
	GetIDsOfNames (see page 2777)	Maps a single member and an optional set of argument names to a corresponding set of integer dispatch identifiers (dispIDs).
	GetModified (see page 26116)	Returns the value of the Modified property.
	GetNodeIndentStr (see page 26117)	Returns the value of the NodeIndentStr property.
	GetOptions (see page 26117)	Returns the value of the Options property.
	GetOwner (see page 2778)	Returns the owner of a component.
	GetOwner (see page 2932)	Returns the owner of an object.
	GetParseOptions (see page 26117)	Returns the value of the ParseOptions property.
	GetProducerTemplateFileName (see page 26599)	This is GetProducerTemplateFileName, a member of class TCustomXSLPageProducer.
	GetProducerTemplateStream (see page 26599)	This is GetProducerTemplateStream, a member of class TCustomXSLPageProducer.
	GetPrologNode (see page 26117)	This is GetPrologNode, a member of class TXMLDocument.
	GetPrologValue (see page 26118)	This is GetPrologValue, a member of class TXMLDocument.
	GetSchemaRef (see page 26118)	Returns the value of the SchemaRef property.
	GetStandAlone (see page 26118)	Returns the value of the StandAlone property.
	GetTemplateFileType (see page 26600)	This is GetTemplateFileType, a member of class TCustomXSLPageProducer.
	GetTemplateStream (see page 26600)	This is GetTemplateStream, a member of class TCustomXSLPageProducer.
	GetTypeInfo (see page 2779)	Retrieves the type information for an object.
	GetTypeInfoCount (see page 2779)	Returns the number of type information interfaces that an object provides (either 0 or 1).
	GetVersion (see page 26118)	Returns the value of the Version property.
	GetXML (see page 26119)	Returns the value of the XML property.
	GetXMLData (see page 26600)	This is GetXMLData, a member of class TCustomXSLPageProducer.
	GetXMLDocument (see page 26600)	This is GetXMLDocument, a member of class TCustomXSLPageProducer.
	GetXSLData (see page 26601)	This is GetXSLData, a member of class TCustomXSLPageProducer.
	HasHTMLBrowserView (see page 26601)	This is HasHTMLBrowserView, a member of class TCustomXSLPageProducer.
	HasScriptView (see page 26601)	This is HasScriptView, a member of class TCustomXSLPageProducer.
	HasXMLBrowserView (see page 26601)	This is HasXMLBrowserView, a member of class TCustomXSLPageProducer.
	HasXSLBrowserView (see page 26601)	This is HasXSLBrowserView, a member of class TCustomXSLPageProducer.
	InternalSetPrologValue (see page 26119)	This is InternalSetPrologValue, a member of class TXMLDocument.
	Invoke (see page 2781)	Provides access to Automation properties and methods when the component wraps an Automation object.

	LoadData (see page 26120)	This is LoadData, a member of class TXMLDocument.
V	Loaded (see page 2782)	Initializes the component after the form file has been read into memory.
V	Loaded (see page 26120)	Initializes the component after the form file has been read into memory.
V	Notification (see page 2783)	Forwards notification messages to all owned components.
V	Notification (see page 26602)	Forwards notification messages to all owned components.
	PaletteCreated (see page 2783)	Responds when the component is created from the component palette.
	ProduceContent (see page 26602)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 26602)	This is ProduceContentFromStream, a member of class TCustomXSLPageProducer.
	ProduceContentFromString (see page 26603)	This is ProduceContentFromString, a member of class TCustomXSLPageProducer.
V	QueryInterface (see page 2784)	Returns a reference to a specified interface if the object supports that interface.
	ReadDOMVendor (see page 26123)	This is ReadDOMVendor, a member of class TXMLDocument.
V	ReadState (see page 2784)	This is ReadState, a member of class TComponent.
	ReleaseDoc (see page 26124)	This is ReleaseDoc, a member of class TXMLDocument.
	SaveToUTF8String (see page 26126)	This is SaveToUTF8String, a member of class TXMLDocument.
	SaveToXMLStrings (see page 26127)	This is SaveToXMLStrings, a member of class TXMLDocument.
V	SetActive (see page 26127)	Sets the value of the Active property.
	SetAncestor (see page 2787)	This is SetAncestor, a member of class TComponent.
	SetAppDispatcher (see page 26603)	This is SetAppDispatcher, a member of class TCustomXSLPageProducer.
	SetChildOrder (see page 2788)	This is SetChildOrder, a member of class TComponent.
	SetDesigning (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDesignInstance (see page 2788)	Ensures that components inserted at design time have their design-mode flag set.
	SetDocumentElement (see page 26127)	Sets the value of the DocumentElement property.
	SetDOMDocument (see page 26128)	Sets the value of the DOMDocument property.
	SetEncoding (see page 26128)	Sets the value of the Encoding property.
	SetFileName (see page 26128)	Sets the value of the FileName property.
	SetInline (see page 2789)	Sets the csInline bit of the component's ComponentState (see page 2804) property
	SetModified (see page 26129)	This is SetModified, a member of class TXMLDocument.
V	SetName (see page 2789)	Sets the value of the Name (see page 2805) property.
	SetNodeIndentStr (see page 26129)	Sets the value of the NodeIndentStr property.
	SetOnAsyncLoad (see page 26129)	Supplies an event handler that responds when the AsyncLoadState property changes.
	SetOptions (see page 26130)	Sets the value of the Options property.
	SetParentComponent (see page 2790)	This is SetParentComponent, a member of class TComponent.
	SetParseOptions (see page 26130)	Sets the value of the ParseOptions property.
	SetPrologValue (see page 26130)	This is SetPrologValue, a member of class TXMLDocument.
	SetStandAlone (see page 26130)	Sets the value of the StandAlone property.
	SetVersion (see page 26131)	Sets the value of the Version property.
	SetXML (see page 26131)	Sets the value of the XML property.
	SetXMLStrings (see page 26131)	This is SetXMLStrings, a member of class TXMLDocument.
	Updated (see page 2791)	Clears the csUpdating state in the component's ComponentState (see page 2804) property when the component finishes updating.
V	UpdateRegistry (see page 2791)	Provides the interface for a method that adds type library and version information to the Registry (see page 18633) on components that implement COM interfaces.
	Updating (see page 2791)	Sets the csUpdating state in the component's ComponentState (see page 2804) property.
	ValidateContainer (see page 2792)	Determines whether an object can be inserted into a container.
	ValidateInsert (see page 2792)	Provides the interface for a method that validates an child component before it is inserted.
V	ValidateRename (see page 2792)	Ensures that renaming an owned component does not create name conflict.
	WriteDOMVendor (see page 26132)	This is WriteDOMVendor, a member of class TXMLDocument.
V	WriteState (see page 2793)	This is WriteState, a member of class TComponent.
	XMLStringsChanging (see page 26132)	This is XMLStringsChanging, a member of class TXMLDocument.

Protected Properties

	Name	Description
R	DocBindingInfo (see page 26135)	This is DocBindingInfo, a member of class TXMLDocument.
	DocSource (see page 26135)	This is DocSource, a member of class TXMLDocument.
R	DOMParseOptions (see page 26137)	This is DOMParseOptions, a member of class TXMLDocument.
R	DOMPersist (see page 26137)	This is DOMPersist, a member of class TXMLDocument.
	PrefixID (see page 26142)	This is PrefixID, a member of class TXMLDocument.

Public Methods

	Name	Description
	_AddRef (see page 22397)	Increments the reference count for this interface.
	_Release (see page 22397)	Decrements the reference count for this interface.
	AddChild (see page 26107)	This is the overview for the AddChild method overload.
	AddChild (see page 26385)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
V	AfterConstruction (see page 26107)	Responds after the last constructor has executed.
V	Assign (see page 2930)	Copies the contents of another, similar object.
V	BeforeDestruction (see page 2770)	Performs any necessary actions before the first destructor is called.
V	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
V	Content (see page 26597)	Returns the transformed value of the XML supplied by the XMLData (see page 26605) property.
V	ContentFromStream (see page 26598)	Returns the transformed value of the XML supplied by the XMLData (see page 26605) property using an XSL template that can be read from a specified stream.
	ContentFromString (see page 26598)	Returns the transformed value of the XML supplied by the XMLData (see page 26605) property using a specified XSL template.
V	ContentFromWideString (see page 26599)	Returns the transformed value of the XML supplied by the XMLData (see page 26605) property using a specified XSL template.
V	Create (see page 2766)	Allocates memory and constructs a safely initialized instance of a component.
	Create (see page 22287)	Constructs an object and initializes its data before the object is first used.
	Create (see page 26103)	This is the overview for the Create constructor overload.
	CreateElement (see page 26109)	Creates a new element node that is associated with this document.
	CreateElement (see page 26386)	Creates a new element node that is associated with this document.
	CreateNode (see page 26109)	Creates a new node that is associated with this document.
	CreateNode (see page 26387)	Creates a new node that is associated with this document.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 2766)	Disposes of the component and its owned components.
V	Destroy (see page 2929)	Destroys the TPersistent (see page 2928) instance and frees its memory.
V	Destroy (see page 22287)	Disposes of an object instance.
V	Destroy (see page 26104)	Disposes of a TXMLDocument (see page 26097) object.
	DestroyComponents (see page 2771)	Destroys all owned components.
	Destroying (see page 2772)	Indicates that the component and its owned components are about to be destroyed.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	ExecuteAction (see page 2772)	This is ExecuteAction, a member of class TComponent.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FindComponent (see page 2772)	Indicates whether a given component is owned by the component.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	FreeNotification (see page 2775)	Ensures that AComponent is notified that the component is going to be destroyed.
	FreeOnRelease (see page 2776)	Frees the interface reference for components that were created from COM classes.
	GeneratePrefix (see page 26112)	Generates a unique namespace prefix that can be used for a node in the document.
	GeneratePrefix (see page 26388)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 26388)	Returns the value of the Active (see page 26403) property.
	GetAppDispatcher (see page 15220)	This is GetAppDispatcher, a member of class IGetAppDispatcher.
	GetAsyncLoadState (see page 26389)	Returns the value of the AsyncLoadState (see page 26404) property.
	GetChildNodes (see page 26389)	Returns the value of the ChildNodes (see page 26405) property.
	GetComponent (see page 3197)	Returns a reference to the component associated by the implementing class.
	GetDocBinding (see page 26114)	Replaces the document element with an instance of a specified implementation class.
	GetDocBinding (see page 26389)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 26390)	Returns the value of the DocumentElement (see page 26405) property.
	GetDocumentNode (see page 26390)	Returns the value of the Node (see page 26407) property.
	GetDocumentObject (see page 26229)	This is GetDocumentObject, a member of class IXMLDocumentAccess.
	GetDOMDocument (see page 26390)	Returns the value of the DOMDocument (see page 26405) property.
	GetDOMPersist (see page 26229)	This is GetDOMPersist, a member of class IXMLDocumentAccess.
	GetEncoding (see page 26391)	Returns the value of the Encoding (see page 26406) property.
	GetEnumerator (see page 2777)	Returns a TComponent (see page 2762) enumerator.
	GetFileName (see page 26391)	Returns the value of the FileName (see page 26406) property.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetModified (see page 26391)	Returns the value of the Modified (see page 26407) property.
	GetNamePath (see page 2778)	Returns a string used by the Object Inspector.
	GetNamePath (see page 2932)	Returns the name of the object as it appears in the Object Inspector.
	GetNodeIndentStr (see page 26392)	Returns the value of the NodeIndentStr (see page 26408) property.
	GetOptions (see page 26392)	Returns the value of the Options (see page 26408) property.
	GetParentComponent (see page 2778)	Returns the containing Component.
	GetParseOptions (see page 26392)	Returns the value of the ParseOptions (see page 26409) property.
	GetProducerTemplateFileName (see page 15320)	This is GetProducerTemplateFileName, a member of class IGetProducerTemplate.
	GetProducerTemplateStream (see page 15320)	This is GetProducerTemplateStream, a member of class IGetProducerTemplate.
	GetSchemaRef (see page 26392)	Returns the value of the SchemaRef (see page 26409) property.
	GetStandAlone (see page 26393)	Returns the value of the StandAlone (see page 26410) property.
	GetTemplateFileType (see page 24780)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetVersion (see page 26393)	Returns the value of the Version (see page 26410) property.
	GetXML (see page 26393)	Returns the value of the XML (see page 26410) property.
	GetXMLData (see page 24780)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 24780)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 24780)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasParent (see page 2780)	This is HasParent, a member of class TComponent.
	HasScriptView (see page 24780)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 24781)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 24781)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InsertComponent (see page 2780)	Establishes the component as the owner of a specified component.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.

	IsEmptyDoc (see page 26119)	Indicates whether the TXMLDocument (see page 26097) instance represents an empty document.
	IsEmptyDoc (see page 26394)	Indicates whether the IXMLDocument (see page 26382) instance represents an empty document.
	IsImplementorOf (see page 2782)	Indicates whether the component implements a specified interface.
	LoadFromFile (see page 26121)	Loads an XML (see page 26144) document and activates it.
	LoadFromFile (see page 26394)	Loads an XML (see page 26410) document from disk.
	LoadFromStream (see page 26121)	Loads an XML (see page 26144) document from a stream and activates it.
	LoadFromStream (see page 26395)	Loads an XML (see page 26410) document from a stream and activates it.
	LoadFromXML (see page 26122)	This is the overview for the LoadFromXML method overload.
	LoadFromXML (see page 26395)	This is the overview for the LoadFromXML method overload.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	NewInstance (see page 26122)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	ProduceContent (see page 15224)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 15225)	This is ProduceContentFromStream, a member of class IProduceContentFrom.
	ProduceContentFromString (see page 15226)	This is ProduceContentFromString, a member of class IProduceContentFrom.
	QueryInterface (see page 22398)	Returns a reference to a specified interface if the object supports that interface.
	ReferenceInterface (see page 2785)	Establishes or removes internal links that cause this component to be notified when the implementer of a specified interface is destroyed.
	Refresh (see page 26123)	Updates the parsed XML (see page 26144) document to reflect external changes.
	Refresh (see page 26396)	Updates the parsed XML (see page 26410) document to reflect external changes.
	RegisterDocBinding (see page 26124)	Registers an implementation class for child nodes with a specified tag name.
	RegisterDocBinding (see page 26396)	Registers an implementation class for child nodes with a specified tag name.
	RemoveComponent (see page 2785)	Removes a specified component specified from the component's Components (see page 2799) list.
	RemoveFreeNotification (see page 2787)	Disables destruction notification that was enabled by FreeNotification (see page 2775)
	Resync (see page 26125)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	Resync (see page 26397)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 2787)	Handles exceptions in methods declared using the safecall calling convention.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
	SaveToFile (see page 26125)	Saves the XML (see page 26144) document to disk.
	SaveToFile (see page 26397)	Saves the XML (see page 26410) document to disk.
	SaveToStream (see page 26125)	Saves the XML (see page 26144) document to a stream.
	SaveToStream (see page 26398)	Saves the XML (see page 26410) document to a stream.
	SaveToXML (see page 26126)	This is the overview for the SaveToXML method overload.
	SaveToXML (see page 26398)	This is the overview for the SaveToXML method overload.
	SetActive (see page 26399)	Sets the value of the Active (see page 26403) property.
	SetAppDispatcher (see page 15227)	This is SetAppDispatcher, a member of class ISetAppDispatcher.
	SetDocumentElement (see page 26399)	Sets the value of the DocumentElement (see page 26405) property.
	SetDOMDocument (see page 26399)	Sets the value of the DOMDocument (see page 26405) property.
	SetEncoding (see page 26400)	Sets the value of the Encoding (see page 26406) property.
	SetFileName (see page 26400)	Sets the value of the FileName (see page 26406) property.
	SetNodeIndentStr (see page 26400)	Sets the value of the NodeIndentStr (see page 26408) property.
	SetOnAsyncLoad (see page 26401)	Supplies an event handler that responds when the AsyncLoadState (see page 26404) property changes.
	SetOptions (see page 26401)	Sets the value of the Options (see page 26408) property.
	SetParseOptions (see page 26402)	Sets the value of the ParseOptions (see page 26409) property.
	SetStandAlone (see page 26402)	Sets the value of the StandAlone (see page 26410) property.
	SetSubComponent (see page 2790)	Identifies whether the component is a subcomponent.

	SetVersion (see page 26402)	Sets the value of the Version (see page 26410) property.
	SetXML (see page 26403)	Sets the value of the XML (see page 26410) property.
	UpdateAction (see page 2790)	This is UpdateAction, a member of class TComponent.

Public Properties

	Name	Description
	Active (see page 26403)	Specifies whether the XML (see page 26410) document has been parsed so that the DOM interfaces are available to examine or change its contents.
R	AsyncLoadState (see page 26134)	Indicates the state of the DOM parser when it parses the XML (see page 26144) document asynchronously.
R	AsyncLoadState (see page 26404)	Indicates the state of the DOM parser when it parses the XML (see page 26410) document asynchronously.
R	ChildNodes (see page 26135)	Lists all of the child nodes of the document.
R	ChildNodes (see page 26405)	Lists all of the child nodes of the document.
R	ComObject (see page 2793)	Specifies the interface reference implemented by the component.
R	ComponentCount (see page 2794)	Indicates the number of components owned by the component.
	ComponentIndex (see page 2799)	Indicates the position of the component in its owner's Components (see page 2799) property array.
R	Components (see page 2799)	Lists all components owned by the component.
R	ComponentState (see page 2804)	Describes the current state of the component, indicating when a component needs to avoid certain actions.
R	ComponentStyle (see page 2805)	Governs the behavior of the component.
	DesignInfo (see page 2805)	Contains information used by the Form designer.
R	Dispatcher (see page 26603)	Specifies the interface of a component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.
R	DispatcherComponent (see page 26604)	Specifies the component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.
	DocumentElement (see page 26135)	Provides access to the root node of the XML (see page 26144) document.
	DocumentElement (see page 26405)	Provides access to the root node of the XML (see page 26410) document.
R	DocumentObject (see page 26229)	This is DocumentObject, a member of class IXMLDocumentAccess.
	DOMDocument (see page 26136)	Provides access to the DOM interface for the parsed XML (see page 26144) document.
	DOMDocument (see page 26405)	Provides access to the DOM interface for the parsed XML (see page 26410) document.
	DOMImplementation (see page 26137)	This is DOMImplementation, a member of class TXMLDocument.
R	DOMPersist (see page 26230)	This is DOMPersist, a member of class IXMLDocumentAccess.
	Encoding (see page 26138)	Indicates the character set that the XML (see page 26144) document uses to encode characters.
	Encoding (see page 26406)	Indicates the character set that the XML (see page 26410) document uses to encode characters.
	FileName (see page 26406)	Indicates the XML (see page 26410) document file, if any, that this interface represents.
R	Modified (see page 26139)	Indicates whether the XML (see page 26144) document has been modified.
R	Modified (see page 26407)	Indicates whether the XML (see page 26410) document has been modified.
R	Node (see page 26140)	Provides access to the document node for the XML (see page 26144) document.
R	Node (see page 26407)	Provides access to the document node for the XML (see page 26410) document.
	NodeIndentStr (see page 26408)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 26410) text.
	NSPrefixBase (see page 26141)	Specifies the base string for automatically-generated namespace prefixes.
	Options (see page 26408)	Influences various aspects of the document's behavior.
R	Owner (see page 2807)	Indicates the component that is responsible for streaming and freeing this component.
	ParseOptions (see page 26409)	Specifies implementation-specific options that are followed when parsing the XML (see page 26410) document.
R	SchemaRef (see page 26142)	Returns the name of the schema associated with this XML (see page 26144) document.

	SchemaRef (see page 26409)	Returns the name of the schema associated with this XML (see page 26410) document.
	StandAlone (see page 26143)	Indicates whether the XML (see page 26144) document includes any external markup declarations.
	StandAlone (see page 26410)	Indicates whether the XML (see page 26410) document includes any external markup declarations.
	VCLComObject (see page 2812)	Represents information used internally by components that support COM.
	Version (see page 26143)	Indicates the XML (see page 26144) version that the document uses.
	Version (see page 26410)	Indicates the XML (see page 26410) version that the document uses.
	XML (see page 26410)	Specifies the content of the XML document associated with this IXMLDocument (see page 26382) interface.

Published Events

	Name	Description
	AfterClose (see page 26144)	Occurs after the XML (see page 26144) document is closed.
	AfterNodeChange (see page 26145)	Occurs after a change is made to one of the nodes in the document.
	AfterOpen (see page 26145)	Occurs immediately after the XML (see page 26144) document is opened.
	BeforeClose (see page 26146)	Occurs immediately before the XML (see page 26144) document is closed.
	BeforeNodeChange (see page 26146)	Occurs before a change is made to one of the nodes in the document.
	BeforeOpen (see page 26147)	Occurs immediately before the XML (see page 26144) document is opened.
	OnAsyncLoad (see page 26147)	Occurs at various stages during the asynchronous parsing of an XML (see page 26144) document.

Published Properties

	Name	Description
	Active (see page 26133)	Specifies whether the XML (see page 26144) document has been parsed so that the DOM interfaces are available to examine or change its contents.
	DOMVendor (see page 26137)	Specifies the DOM implementation to use for parsing and manipulating the XML (see page 26144) document.
	FileName (see page 26138)	Specifies the file name for the XML (see page 26144) document associated with this TXMLDocument (see page 26097) object.
	Name (see page 2805)	Specifies the name of the component as referenced in code.
	NodeIndentStr (see page 26140)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 26144) text.
	Options (see page 26141)	Influences various aspects of the document's behavior.
	ParseOptions (see page 26142)	Specifies implementation-specific options that are followed when parsing the XML (see page 26144) document.
	Tag (see page 2808)	Stores an integer value as part of a component.
	XML (see page 26144)	Specifies the content of the XML document associated with this TXMLDocument (see page 26097) object.
	XMLData (see page 26613)	Specifies the component that supplies the XML data that this component transforms.

200.1.3.2 TXSLPageProducer Properties

The properties of the TXSLPageProducer class are listed here.

Published Properties

	Name	Description
	XMLData (see page 26613)	Specifies the component that supplies the XML data that this component transforms.

200.1.3.2.1 TXSLPageProducer.XMLData Property

Specifies the component that supplies the XML data that this component transforms.

Pascal

```
property XMLData: TComponent;
```

C++

```
__property TComponent * XMLData;
```

Description

Use the XMLData property to specify the XML data that this component transforms. XMLData must be set to a component that supports either the IXMLDocument interface (for example, TXMLDocument or one of its descendants) or the IGetXMLStream interface (for example, TXMLBroker).

See Also

TXMLDocument (see page 26097)

TXMLBroker (see page 26049)

201 ZLib Namespace

This is namespace ZLib.

Topics

Name	Description
Classes (see page 26617)	The following table lists classes in this documentation.
Functions (see page 26632)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 26637)	The following table lists structs, records, enums in this documentation.
Types (see page 26639)	The following table lists types in this documentation.
Constants (see page 26640)	The following table lists constants in this documentation.

Classes

	Name	Description
	ECompressionError (see page 26617)	This is class ZLib.ECompressionError.
	EDecompressionError (see page 26617)	This is class ZLib.EDecompressionError.
	EZlibError (see page 26618)	This is class ZLib.EZlibError.
	TCompressionStream (see page 26618)	Compresses data as it is written to the stream.
	TCustomZlibStream (see page 26623)	Abstract ancestor class for compression streams.
	TDecompressionStream (see page 26627)	Decompresses data that was written to a TCompressionStream (see page 26618) object.

Constants

Name	Description
Z_ASCII (see page 26640)	This is constant ZLib.Z_ASCII.
Z_BEST_COMPRESSION (see page 26641)	This is constant ZLib.Z_BEST_COMPRESSION.
Z_BEST_SPEED (see page 26641)	This is constant ZLib.Z_BEST_SPEED.
Z_BINARY (see page 26641)	This is constant ZLib.Z_BINARY.
Z_BUF_ERROR (see page 26642)	This is constant ZLib.Z_BUF_ERROR.
Z_DATA_ERROR (see page 26642)	This is constant ZLib.Z_DATA_ERROR.
Z_DEFAULT_COMPRESSION (see page 26642)	This is constant ZLib.Z_DEFAULT_COMPRESSION.
Z_DEFAULT_STRATEGY (see page 26643)	This is constant ZLib.Z_DEFAULT_STRATEGY.
Z_DEFLATED (see page 26643)	This is constant ZLib.Z_DEFLATED.
Z_ERRNO (see page 26643)	This is constant ZLib.Z_ERRNO.
Z_FILTERED (see page 26644)	This is constant ZLib.Z_FILTERED.
Z_FINISH (see page 26644)	This is constant ZLib.Z_FINISH.
Z_FULL_FLUSH (see page 26644)	This is constant ZLib.Z_FULL_FLUSH.

Z_HUFFMAN_ONLY (see page 26645)	This is constant ZLib.Z_HUFFMAN_ONLY.
Z_MEM_ERROR (see page 26645)	This is constant ZLib.Z_MEM_ERROR.
Z_NEED_DICT (see page 26645)	This is constant ZLib.Z_NEED_DICT.
Z_NO_COMPRESSION (see page 26646)	This is constant ZLib.Z_NO_COMPRESSION.
Z_NO_FLUSH (see page 26646)	This is constant ZLib.Z_NO_FLUSH.
Z_OK (see page 26646)	This is constant ZLib.Z_OK.
Z_PARTIAL_FLUSH (see page 26647)	This is constant ZLib.Z_PARTIAL_FLUSH.
Z_STREAM_END (see page 26647)	This is constant ZLib.Z_STREAM_END.
Z_STREAM_ERROR (see page 26647)	This is constant ZLib.Z_STREAM_ERROR.
Z_SYNC_FLUSH (see page 26648)	This is constant ZLib.Z_SYNC_FLUSH.
Z_UNKNOWN (see page 26648)	This is constant ZLib.Z_UNKNOWN.
Z_VERSION_ERROR (see page 26648)	This is constant ZLib.Z_VERSION_ERROR.
zlib_Version (see page 26649)	This is constant ZLib.zlib_Version.

Functions

	Name	Description
	CompressBuf (see page 26632)	This is function ZLib.CompressBuf.
	DecompressBuf (see page 26632)	This is function ZLib.DecompressBuf.
	DecompressToUserBuf (see page 26633)	This is function ZLib.DecompressToUserBuf.
	deflate (see page 26633)	This is function ZLib.deflate.
	deflateEnd (see page 26633)	This is function ZLib.deflateEnd.
	deflateInit_ (see page 26634)	This is function ZLib.deflateInit_.
	inflate (see page 26634)	This is function ZLib.inflate.
	inflateEnd (see page 26634)	This is function ZLib.inflateEnd.
	inflateInit_ (see page 26635)	This is function ZLib.inflateInit_.
	inflateReset (see page 26635)	This is function ZLib.inflateReset.
	zlibAllocMem (see page 26635)	This is function ZLib.zlibAllocMem.
	zlibFreeMem (see page 26636)	This is function ZLib.zlibFreeMem.

Structs, Records, Enums

	Name	Description
	TCompressionLevel (see page 26637)	TCompressionLevel indicates the type of compression algorithm to be used when creating a compressed data stream.
	TZStreamRec (see page 26638)	This is record ZLib.TZStreamRec.

Types

Name	Description
TAlloc (see page 26639)	This is type ZLib.TAlloc.
TFree (see page 26639)	This is type ZLib.TFree.

201.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ECompressionError (see page 26617)	This is class ZLib.ECompressionError.
	EDecompressionError (see page 26617)	This is class ZLib.EDecompressionError.
	EZlibError (see page 26618)	This is class ZLib.EZlibError.
	TCompressionStream (see page 26618)	Compresses data as it is written to the stream.
	TCustomZlibStream (see page 26623)	Abstract ancestor class for compression streams.
	TDecompressionStream (see page 26627)	Decompresses data that was written to a TCompressionStream (see page 26618) object.

201.1.1 ECompressionError Class

Class Hierarchy

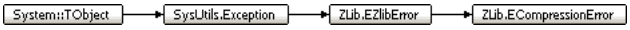

Pascal

```
ECompressionError = class(EZlibError);
```

C++

```
class ECompressionError : public EZlibError;
```

File

ZLib

Description

This is class ZLib.ECompressionError.

201.1.2 EDecompressionError Class

Class Hierarchy

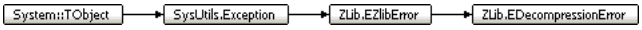

Pascal

```
EDecompressionError = class(EZlibError);
```

C++

```
class EDecompressionError : public EZlibError;
```

File

ZLib

Description

This is class ZLib.EDecompressionError.

201.1.3 EZlibError Class

Class Hierarchy

Pascal

```
EZlibError = class(Exception);
```

C++

```
class EZlibError : public Exception;
```

File

ZLib

Description

This is class ZLib.EZlibError.

201.1.4 TCompressionStream Class

Compresses data as it is written to the stream.

Class Hierarchy

Pascal

```
TCompressionStream = class(TCustomZlibStream);
```

C++

```
class TCompressionStream : public TCustomZlibStream;
```

File

ZLib

Description

TCompressionStream is a write-only, sequential access stream that compresses data as it is written. The compressed data is written to a separate stream, which is passed to the TCompressionStream constructor. TCompressionStream does not take ownership of the output stream; you are responsible for creating, initializing, and destroying the stream when it is no longer needed.

Reading from a TCompressionStream object will raise an exception of type ECompressionError (see page 26617), with the message string set to "Invalid Stream Operation." Similarly, a Seek (see page 26621) operation causes an ECompressionError (see page 26617) exception, however, you can perform a Seek (see page 26621) operation with a zero offset from the current position. This will return the number of raw, uncompressed bytes that have been written to the stream.

You set the compression level to be used when you create the stream object. The possible compression levels are: clNone,

clFastest, clDefault, and clMax.

When the stream's output buffer is filled, a Write (see page 26622) operation on TCompressionStream will write the contents of the buffer to the output stream, and then call the OnProgress (see page 26623) event. You can use the OnProgress (see page 26623) event to update a user interface control such as a progress indicator.

See Also

TCustomZlibStream (see page 26623)

TDecompressionStream (see page 26627)

TStream (see page 2973)

201.1.4.1 TCompressionStream Members

The following tables list the members exposed by TCompressionStream.

Protected Methods

	Name	Description
	GetSize (see page 2982)	This is GetSize, a member of class TStream.
	Progress (see page 26626)	Calls the OnProgress (see page 26626) method, if assigned.
	SetSize (see page 2989)	This is the overview for the SetSize method overload.

Public Events

	Name	Description
	OnProgress (see page 26623)	Occurs after the stream's output buffer has been written.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CopyFrom (see page 2979)	Copies a specified number of bytes from one stream to another.
	Create (see page 26620)	Creates and initializes an instance of a TCompressionStream (see page 26618) object.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 26620)	Destroys the TCompressionStream (see page 26618) object.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FixupResourceHeader (see page 2981)	Patches the resource header for a resource that has been written to the stream.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Read (see page 26621)	Raises an exception of type ECompressionError (see page 26617).
	ReadBuffer (see page 2983)	Reads bytes from the stream into Buffer.
	ReadComponent (see page 2983)	Initiates streaming of components and their properties.
	ReadComponentRes (see page 2987)	Reads components and their properties from a stream in a resource file format.
	ReadResHeader (see page 2987)	Reads a resource-file header from the stream.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Seek (see page 26621)	Raises an exception of type ECompressionErrror.
V	Seek (see page 2988)	This is the overview for the Seek method overload.
V	Write (see page 26622)	Writes compressed data to the destination output stream.
	WriteBuffer (see page 2990)	Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.
	WriteComponent (see page 2990)	Initiates the writing of components and their properties to a stream.
	WriteComponentRes (see page 2994)	Writes components and their properties to a stream from a resource file format.
	WriteDescendent (see page 2994)	Streams components and their properties in inherited forms.
	WriteDescendentRes (see page 2995)	Streams components and their properties in inherited forms.
	WriteResourceHeader (see page 2995)	Writes a resource-file header to the stream.

Public Properties

	Name	Description
R	CompressionRate (see page 26622)	Returns the percentage by which the data has been compressed.
	Position (see page 2996)	Indicates the current offset into the stream for reading and writing.
	Size (see page 2997)	Indicates the size in bytes of the stream.

201.1.4.2 TCompressionStream.Create Constructor

Creates and initializes an instance of a TCompressionStream (see page 26618) object.

Pascal

```
constructor Create(CompressionLevel: TCompressionLevel; Dest: TStream);
```

C++

```
__fastcall TCompressionStream(TCompressionLevel CompressionLevel, TStream Dest);
```

Description

Call Create to instantiate a TCompressionStream (see page 26618) object.

The CompressionLevel parameter determines the type of compression algorithm to be used, and must be one of the following: clNone, clFastest, clDefault, or clMax.

The Dest parameter is a TStream object; compressed data will be written to this stream.

201.1.4.3 TCompressionStream.Destroy Destructor

Destroys the TCompressionStream (see page 26618) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCompressionStream();
```

Description

Do not call the Destroy method directly. Instead, use the Free method, as this will check the validity of the object before calling Destroy.

All data in the TCompressionStream (see page 26618)'s cache is flushed to the output stream when the object is destroyed.

201.1.4.4 TCompressionStream Methods

The methods of the TCompressionStream class are listed here.

Public Methods

	Name	Description
	Read (see page 26621)	Raises an exception of type ECompressionError (see page 26617).
	Seek (see page 26621)	Raises an exception of type ECompressionErrror.
	Write (see page 26622)	Writes compressed data to the destination output stream.

201.1.4.4.1 TCompressionStream.Read Method

Raises an exception of type ECompressionError (see page 26617).

Pascal

```
function Read(var Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Read( Buffer, Longint Count);
```

Description

Read operations are not permitted on a TCompressionStream (see page 26618) object. If a Read is attempted, an exception of type ECompressionError (see page 26617) will be raised, with the message string "Invalid Stream Operation."

See Also

Write (see page 26622)

Seek (see page 26621)

Read (see page 2982)

201.1.4.4.2 TCompressionStream.Seek Method

Raises an exception of type ECompressionErrror.

Pascal

```
function Seek(Offset: Longint; Origin: Word): Longint; override;
```

C++

```
virtual __fastcall Longint Seek(Longint Offset, Word Origin);
```

Description

A TCompressionStream (see page 26618) is a write-only, sequential access data stream. Read (see page 26621) operations are not permitted, and will raise an exception of type ECompressionError (see page 26617). Similarly, Seek

operations will also raise an ECompressionError (see page 26617), with the message string "Invalid Stream Operation."

However, you can call the Seek method with an offset of zero, and the Origin set to soFromCurrent. In this special case, the Seek method will return the number of raw (uncompressed) bytes that have been written to the stream so far.

See Also

Read (see page 26621)

Write (see page 26622)

Seek (see page 2988)

201.1.4.4.3 TCompressionStream.Write Method

Writes compressed data to the destination output stream.

Pascal

```
function Write(const Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  Buffer, Longint Count);
```

Description

The Write method causes the data in the supplied buffer to be written the output stream. Buffer contains the raw (uncompressed) data to be written, and Count contains the number of bytes to write.

The TCompressionStream (see page 26618) object caches data internally; data is flushed to the output stream when the cache becomes full, and when the object is destroyed (if pending data remains in the cache).

After data is written to the output stream, the OnProgress (see page 26623) event is called. This gives you an opportunity to update a user interface control such as a progress indicator.

The result of the Write method is the number of bytes transferred from the buffer

See Also

Read (see page 26621)

Seek (see page 26621)

Write (see page 2989)

201.1.4.5 TCompressionStream Properties

The properties of the TCompressionStream class are listed here.

Public Properties

	Name	Description
R	CompressionRate (see page 26622)	Returns the percentage by which the data has been compressed.

201.1.4.5.1 TCompressionStream.CompressionRate Property

Returns the percentage by which the data has been compressed.

Pascal

```
property CompressionRate: Single;
```

C++

```
__property Single CompressionRate;
```

Description

The compression rate is a read-only property. Its value is calculated and updated as data is written to the stream. For example, if a total of 100 raw (uncompressed) bytes have been written to the stream, and only 25 compressed bytes have been written out, then reading the CompressionRate property will return a value of 75.

201.1.4.6 TCompressionStream Events

The events of the TCompressionStream class are listed here.

Public Events

	Name	Description
	OnProgress (see page 26623)	Occurs after the stream's output buffer has been written.

201.1.4.6.1 TCompressionStream.OnProgress Event

Occurs after the stream's output buffer has been written.

Pascal

```
property OnProgress: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnProgress;
```

Description

The OnProgress event is automatically called from a Write (see page 26622) operation, after the object's internal buffer has been filled and written to the output stream.

The OnProgress event is of type TNotifierEvent. When your procedure is called, the Sender parameter will be set to the object itself. You can use the OnProgress event to update a user interface control, such as a progress indicator. You can use the Position property, inherited from TStream, to determine the number of raw (uncompressed) bytes written to the output stream so far.

201.1.5 TCustomZlibStream Class

Abstract ancestor class for compression streams.

Class Hierarchy

Pascal

```
TCustomZlibStream = class(TStream);
```

C++

```
class TCustomZlibStream : public TStream;
```

File

ZLib

Description

TCustomZlibStream is an abstract class, which forms the base for TCompressionStream (see page 26618) and TDecompressionStream (see page 26627). Since it is an abstract class, instances of TCustomZlibStream cannot be directly instantiated. TCustomZlibStream serves as the starting point for your own custom data compression streams.

See Also

TCompressionStream (see page 26618)

TDecompressionStream (see page 26627)

TStream (see page 2973)

201.1.5.1 TCustomZlibStream Members

The following tables list the members exposed by TCustomZlibStream.

Protected Events

	Name	Description
	OnProgress (see page 26626)	Occurs when reading or writing a large block of data in a single call.

Protected Methods

	Name	Description
	Create (see page 26625)	Initializes an instance of a TCustomZlibStream (see page 26623)-derived object.
	GetSize (see page 2982)	This is GetSize, a member of class TStream.
	Progress (see page 26626)	Calls the OnProgress (see page 26626) method, if assigned.
	SetSize (see page 2989)	This is the overview for the SetSize method overload.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CopyFrom (see page 2979)	Copies a specified number of bytes from one stream to another.
	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
	Destroy (see page 22287)	Disposes of an object instance.
	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FixupResourceHeader (see page 2981)	Patches the resource header for a resource that has been written to the stream.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.

	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
A	Read (see page 2982)	Introduces an abstract or pure virtual method responsible for reading from the stream.
	ReadBuffer (see page 2983)	Reads bytes from the stream into Buffer.
	ReadComponent (see page 2983)	Initiates streaming of components and their properties.
	ReadComponentRes (see page 2987)	Reads components and their properties from a stream in a resource file format.
	ReadResHeader (see page 2987)	Reads a resource-file header from the stream.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Seek (see page 2988)	This is the overview for the Seek method overload.
A	Write (see page 2989)	Introduces a pure virtual method for writing to the stream.
	WriteBuffer (see page 2990)	Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.
	WriteComponent (see page 2990)	Initiates the writing of components and their properties to a stream.
	WriteComponentRes (see page 2994)	Writes components and their properties to a stream from a resource file format.
	WriteDescendent (see page 2994)	Streams components and their properties in inherited forms.
	WriteDescendentRes (see page 2995)	Streams components and their properties in inherited forms.
	WriteResourceHeader (see page 2995)	Writes a resource-file header to the stream.

Public Properties

	Name	Description
	Position (see page 2996)	Indicates the current offset into the stream for reading and writing.
	Size (see page 2997)	Indicates the size in bytes of the stream.

201.1.5.2 TCustomZlibStream.Create Constructor

Initializes an instance of a TCustomZlibStream (see page 26623)-derived object.

Pascal

```
constructor Create(Strm: TStream);
```

C++

```
__fastcall TCustomZlibStream(TStream Strm);
```

Description

TCustomZlibStream (see page 26623) is intended to be an abstract base class for your own custom data compression streams. Its constructor is protected; therefore you cannot directly instantiate an instance of this class.

A concrete class derived from TCustomZlibStream (see page 26623) should, in its own public constructor, call the constructor inherited from TCustomZlibStream (see page 26623).

201.1.5.3 TCustomZlibStream Methods

The methods of the TCustomZlibStream class are listed here.

Protected Methods

	Name	Description
	Progress (see page 26626)	Calls the OnProgress (see page 26626) method, if assigned.

201.1.5.3.1 TCustomZlibStream.Progress Method

Calls the OnProgress (see page 26626) method, if assigned.

Pascal

```
procedure Progress(Sender: TObject); dynamic;
```

C++

```
__fastcall Progress(TObject * Sender);
```

Description

Derived compression stream classes should call the Progress method when reading or writing a large block of data in a single call. This gives the derived class an opportunity to update a user interface control such as a progress indicator.

See Also

OnProgress (see page 26623)

OnProgress (see page 26631)

201.1.5.4 TCustomZlibStream Events

The events of the TCustomZlibStream class are listed here.

Protected Events

	Name	Description
	OnProgress (see page 26626)	Occurs when reading or writing a large block of data in a single call.

201.1.5.4.1 TCustomZlibStream.OnProgress Event

Occurs when reading or writing a large block of data in a single call.

Pascal

```
property OnProgress: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnProgress;
```

Description

Classes (see page 2685) derived from TCustomZlibStream (see page 26623) can use the OnProgress event to perform some action during a long read or write operation.

For example, the TCompressionStream (see page 26618) class calls the OnProgress event handler, after the stream's output buffer has been filled and subsequently written. Similarly, the TDecompressionStream (see page 26627) class calls the OnProgress event handler when its input buffer has been filled and is ready to be decompressed.

See Also

Progress (see page 26626)

OnProgress (see page 26623)

OnProgress (see page 26631)

201.1.6 TDecompressionStream Class

Decompresses data that was written to a TCompressionStream (see page 26618) object.

Class Hierarchy

Pascal

```
TDecompressionStream = class(TCustomZlibStream);
```

C++

```
class TDecompressionStream : public TCustomZlibStream;
```

File

ZLib

Description

TDecompressionStream is the analog of the TCompressionStream (see page 26618) class. It is a read-only, unidirectional stream; you can seek forward in the stream, but not backwards. A separate input stream provides the source data. The input stream will be a stream that was previously written to using a TCompressionStream (see page 26618) object. The input stream is passed to the constructor, however, TDecompressionStream does not take ownership of this object. You are responsible for creating, initializing, and destroying the input stream when it is no longer needed.

Writing to the stream will raise an exception, as will requesting the size of the stream, and attempting to perform a Seek (see page 26630) operation backwards or relative to the current position.

The TDecompressionStream object's OnProgress (see page 26631) event is called when a block of data is read from the input stream. You can read the TDecompressionStream object's Position property to determine the number of raw, uncompressed bytes that have been read from the input stream so far.

201.1.6.1 TDecompressionStream Members

The following tables list the members exposed by TDecompressionStream.

Protected Methods

	Name	Description
	GetSize (see page 2982)	This is GetSize, a member of class TStream.
	Progress (see page 26626)	Calls the OnProgress (see page 26626) method, if assigned.
	SetSize (see page 2989)	This is the overview for the SetSize method overload.

Public Events

	Name	Description
	OnProgress (see page 26631)	Occurs after a block of data has been read from the input stream.

Public Methods

	Name	Description
	AfterConstruction (see page 22289)	Responds after the last constructor has executed.
	BeforeDestruction (see page 22289)	Responds before the first destructor executes.
	ClassInfo (see page 22290)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 22290)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 22293)	Determines whether an object is of a specific type.
	ClassParent (see page 22293)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 22295)	Returns the class reference for the object's class.
	CleanupInstance (see page 22298)	Performs finalization on long strings, variants, and interface variables within a class.
	CopyFrom (see page 2979)	Copies a specified number of bytes from one stream to another.
	Create (see page 26628)	Creates and initializes an instance of a TDecompressionStream (see page 26627) object.
V	DefaultHandler (see page 22299)	Provides the interface for a method that processes message records.
V	Destroy (see page 26629)	Destroys the TDecompressionStream (see page 26627) object.
V	Dispatch (see page 22299)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	FieldAddress (see page 22299)	Returns the address of a published object field.
	FixupResourceHeader (see page 2981)	Patches the resource header for a resource that has been written to the stream.
	Free (see page 22300)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 22303)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 22306) method.
	GetInterface (see page 22303)	Retrieves a specified interface.
	GetInterfaceEntry (see page 22304)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 22304)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 22305)	Determines the relationship of two object types.
	InitInstance (see page 22305)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 22306)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 22306)	This is MethodAddress, a member of class TObject.
	MethodName (see page 22306)	This is MethodName, a member of class TObject.
V	NewInstance (see page 22306)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Read (see page 26629)	Reads a block of data from the input stream, and decompresses it.
	ReadBuffer (see page 2983)	Reads bytes from the stream into Buffer.
	ReadComponent (see page 2983)	Initiates streaming of components and their properties.
	ReadComponentRes (see page 2987)	Reads components and their properties from a stream in a resource file format.
	ReadResHeader (see page 2987)	Reads a resource-file header from the stream.
V	SafeCallException (see page 22307)	Handles exceptions in methods declared using the safecall calling convention.
V	Seek (see page 26630)	Resets the input stream, or moves the position forward.
V	Seek (see page 2988)	This is the overview for the Seek method overload.
V	Write (see page 26630)	Raises an exception of type ECompressionError (see page 26617).
	WriteBuffer (see page 2990)	Writes Count bytes from Buffer onto the stream and advances the current position of the stream by Count bytes.
	WriteComponent (see page 2990)	Initiates the writing of components and their properties to a stream.
	WriteComponentRes (see page 2994)	Writes components and their properties to a stream from a resource file format.
	WriteDescendent (see page 2994)	Streams components and their properties in inherited forms.
	WriteDescendentRes (see page 2995)	Streams components and their properties in inherited forms.
	WriteResourceHeader (see page 2995)	Writes a resource-file header to the stream.

Public Properties

	Name	Description
	Position (see page 2996)	Indicates the current offset into the stream for reading and writing.
	Size (see page 2997)	Indicates the size in bytes of the stream.

201.1.6.2 TDecompressionStream.Create Constructor

Creates and initializes an instance of a TDecompressionStream (see page 26627) object.

Pascal

```
constructor Create(Source: TStream);
```

C++

```
__fastcall TDecompressionStream(TStream Source);
```

Description

Call Create to instantiate a TDecompressionStream (see page 26627) object.

The Source parameter is the source input stream, which was previously created with a TCompressionStream (see page 26618) object. Subsequent Read (see page 26629) operations will read data from the Source stream, and decompress the data on the fly.

201.1.6.3 TDecompressionStream.Destroy Destructor

Destroys the TDecompressionStream (see page 26627) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDecompressionStream();
```

Description

Do not call the Destroy method directly. Instead, use the Free method, as this will check the validity of the object before calling Destroy.

If there is data remaining in the object's internal buffer, the input stream position is moved backwards by the number of bytes remaining.

201.1.6.4 TDecompressionStream Methods

The methods of the TDecompressionStream class are listed here.

Public Methods

	Name	Description
	Read (see page 26629)	Reads a block of data from the input stream, and decompresses it.
	Seek (see page 26630)	Resets the input stream, or moves the position forward.
	Write (see page 26630)	Raises an exception of type ECompressionError (see page 26617).

201.1.6.4.1 TDecompressionStream.Read Method

Reads a block of data from the input stream, and decompresses it.

Pascal

```
function Read(var Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Read( Buffer, Longint Count);
```

Description

The Read method reads a block of data from the input stream, decompresses it, and stores it in the supplied buffer.

The Buffer parameter is the destination for the decompressed data.

Count is size of the destination buffer. The Read operation will end when the destination buffer is full, or when there are no more bytes available to be read from the input stream.

After reading a block of data, the OnProgress (see page 26631) event is fired by the object. This gives you a chance to update a user interface control such as a progress indicator during a long Read operation.

The result of the Read operation is the number of decompressed bytes placed into the destination buffer.

See Also

Write (see page 26630)

Seek (see page 26630)

Read (see page 2982)

201.1.6.4.2 TDecompressionStream.Seek Method

Resets the input stream, or moves the position forward.

Pascal

```
function Seek(Offset: Longint; Origin: Word): Longint; override;
```

C++

```
virtual __fastcall Longint Seek(Longint Offset, Word Origin);
```

Description

A TDecompressionStream (see page 26627) is a read-only, unidirectional input stream. Write (see page 26630) operations are not permitted, and will raise an exception of type ECompressionError (see page 26617). Seek operations can be used to move the stream position forward, but a Seek operation that attempts to move the position backwards will raise an ECompressionError (see page 26617) exception, with the message string set to "Invalid Stream Operation."

You can call the Seek method with an Offset of zero, and the Origin set to soFromBeginning. This has the effect of resetting the input stream, effectively rewinding the stream back to the beginning.

Otherwise, the Offset parameter must be a positive Longint, and the Origin parameter must be either soFromCurrent, or soFromBeginning. This has the effect of moving the input stream position forward, decompressing data as it goes.

See Also

Read (see page 26629)

Write (see page 26622)

Seek (see page 2988)

201.1.6.4.3 TDecompressionStream.Write Method

Raises an exception of type ECompressionError (see page 26617).

Pascal

```
function Write(const Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  Buffer, Longint Count);
```

Description

Write operations are not permitted on a TDecompressionStream (see page 26627) object. If a Write operation is attempted, an exception of type ECompressionError (see page 26617) will be raised, with the message string set to "Invalid Stream Operation."

See Also

Read (see page 26629)

Seek (see page 26630)

Write (see page 2989)

201.1.6.5 TDecompressionStream Events

The events of the TDecompressionStream class are listed here.

Public Events

	Name	Description
	OnProgress (see page 26631)	Occurs after a block of data has been read from the input stream.

201.1.6.5.1 TDecompressionStream.OnProgress Event

Occurs after a block of data has been read from the input stream.

Pascal

```
property OnProgress: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnProgress;
```

Description

The OnProgress event is automatically called from a Read (see page 26629) operation. The event occurs after a block of data has been read from the input stream, and before the data has been decompressed.

The OnProgress event is of type TNotifierEvent. When your procedure is called, the Sender parameter will be set to the object itself. You can use the OnProgress event to update a user interface control, such as a progress indicator. You can use the Position property, inherited from TStream, to determine the number of raw (uncompressed) bytes read from the input stream so far.

201.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CompressBuf (see page 26632)	This is function ZLib.CompressBuf.
	DecompressBuf (see page 26632)	This is function ZLib.DecompressBuf.
	DecompressToUserBuf (see page 26633)	This is function ZLib.DecompressToUserBuf.
	deflate (see page 26633)	This is function ZLib.deflate.
	deflateEnd (see page 26633)	This is function ZLib.deflateEnd.
	deflateInit_ (see page 26634)	This is function ZLib.deflateInit_.
	inflate (see page 26634)	This is function ZLib.inflate.
	inflateEnd (see page 26634)	This is function ZLib.inflateEnd.
	inflateInit_ (see page 26635)	This is function ZLib.inflateInit_.
	inflateReset (see page 26635)	This is function ZLib.inflateReset.
	zlibAllocMem (see page 26635)	This is function ZLib.zlibAllocMem.
	zlibFreeMem (see page 26636)	This is function ZLib.zlibFreeMem.

201.2.1 ZLib.CompressBuf Function

Pascal

```
procedure CompressBuf(const InBuf: Pointer; InBytes: Integer; out OutBuf: Pointer; out
OutBytes: Integer);
```

C++

```
CompressBuf(const void * InBuf, int InBytes, void * OutBuf, int OutBytes);
```

File

ZLib

Description

This is function ZLib.CompressBuf.

201.2.2 ZLib.DecompressBuf Function

Pascal

```
procedure DecompressBuf(const InBuf: Pointer; InBytes: Integer; OutEstimate: Integer; out
OutBuf: Pointer; out OutBytes: Integer);
```

C++

```
DecompressBuf(const void * InBuf, int InBytes, int OutEstimate, void * OutBuf, int OutBytes);
```

File

ZLib

Description

This is function ZLib.DecompressBuf.

201.2.3 ZLib.DecompressToUserBuf Function

Pascal

```
procedure DecompressToUserBuf(const InBuf: Pointer; InBytes: Integer; const OutBuf: Pointer;
BufSize: Integer);
```

C++

```
DecompressToUserBuf(const void * InBuf, int InBytes, const void * OutBuf, int BufSize);
```

File

ZLib

Description

This is function ZLib.DecompressToUserBuf.

201.2.4 ZLib.deflate Function

Pascal

```
function deflate(var strm: TZStreamRec; flush: Integer): Integer;
```

C++

```
int deflate(TZStreamRec strm, int flush);
```

File

ZLib

Description

This is function ZLib.deflate.

201.2.5 ZLib.deflateEnd Function

Pascal

```
function deflateEnd(var strm: TZStreamRec): Integer;
```

C++

```
int deflateEnd(TZStreamRec strm);
```

File

ZLib

Description

This is function ZLib.deflateEnd.

201.2.6 ZLib.deflateInit_ Function

Pascal

```
function deflateInit_(var strm: TZStreamRec; level: Integer; version: PChar; recsize:
Integer): Integer;
```

C++

```
int deflateInit_(TZStreamRec strm, int level, const char * version, int recsize);
```

File

ZLib

Description

This is function ZLib.deflateInit_.

201.2.7 ZLib.inflate Function

Pascal

```
function inflate(var strm: TZStreamRec; flush: Integer): Integer;
```

C++

```
int inflate(TZStreamRec strm, int flush);
```

File

ZLib

Description

This is function ZLib.inflate.

201.2.8 ZLib.inflateEnd Function

Pascal

```
function inflateEnd(var strm: TZStreamRec): Integer;
```

C++

```
int inflateEnd(TZStreamRec strm);
```

File

ZLib

Description

This is function ZLib.inflateEnd.

201.2.9 ZLib.inflateInit_ Function

Pascal

```
function inflateInit_(var strm: TZStreamRec; version: PChar; recsize: Integer): Integer;
```

C++

```
int inflateInit_(TZStreamRec strm, const char * version, int recsize);
```

File

ZLib

Description

This is function ZLib.inflateInit_.

201.2.10 ZLib.inflateReset Function

Pascal

```
function inflateReset(var strm: TZStreamRec): Integer;
```

C++

```
int inflateReset(TZStreamRec strm);
```

File

ZLib

Description

This is function ZLib.inflateReset.

201.2.11 ZLib.zlibAllocMem Function

Pascal

```
function zlibAllocMem(AppData: Pointer; Items: Integer; Size: Integer): Pointer;
```

C++

```
void * zlibAllocMem(void * AppData, int Items, int Size);
```

File

ZLib

Description

This is function ZLib.zlibAllocMem.

201.2.12 ZLib.zlibFreeMem Function

Pascal

```
procedure zlibFreeMem(AppData: Pointer; Block: Pointer);
```

C++

```
zlibFreeMem(void * AppData, void * Block);
```

File

ZLib

Description

This is function ZLib.zlibFreeMem.

201.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCompressionLevel (see page 26637)	TCompressionLevel indicates the type of compression algorithm to be used when creating a compressed data stream.

Records

	Name	Description
	TZStreamRec (see page 26638)	This is record ZLib.TZStreamRec.

201.3.1 ZLib.TCompressionLevel Enumeration

TCompressionLevel indicates the type of compression algorithm to be used when creating a compressed data stream.

Pascal

```
TCompressionLevel = (
  clNone,
  clFastest,
  clDefault,
  clMax
);
```

C++

```
enum TCompressionLevel {
  clNone,
  clFastest,
  clDefault,
  clMax
};
```

File

ZLib

Description

The following table lists the possible values:

Value	Meaning
clNone	Specifies no compression; data is merely copied to the output stream.
clFastest	Specifies fastest compression, resulting in a larger archive.
clDefault	Compromise between speed and amount of compression.
clMax	Specifies maximum compression, resulting in a longer time to perform the operation.

See Also

TCustomZlibStream (see page 26623)

TDecompressionStream (see page 26627)

TCompressionStream (see page 26618)

201.3.2 ZLib.TZStreamRec Record

Pascal

```
TZStreamRec = packed record
  next_in: PChar;
  avail_in: Integer;
  total_in: Integer;
  next_out: PChar;
  avail_out: Integer;
  total_out: Integer;
  msg: PChar;
  internal: Pointer;
  zalloc: TAlloc;
  zfree: TFree;
  AppData: Pointer;
  data_type: Integer;
  adler: Integer;
  reserved: Integer;
end;
```

C++

```
struct TZStreamRec {
  const char * next_in;
  int avail_in;
  int total_in;
  const char * next_out;
  int avail_out;
  int total_out;
  const char * msg;
  void * internal;
  TAlloc zalloc;
  TFree zfree;
  void * AppData;
  int data_type;
  int adler;
  int reserved;
};
```

File

ZLib

Description

This is record ZLib.TZStreamRec.

201.4 Types

The following table lists types in this documentation.

Types

Name	Description
TAlloc (see page 26639)	This is type ZLib.TAlloc.
TFree (see page 26639)	This is type ZLib.TFree.

201.4.1 ZLib.TAlloc Type

Pascal

```
TAlloc = function (AppData: Pointer; Items, Size: Integer): Pointer;
```

C++

```
(AppData: Pointer; Items, Size: Integer): Pointer ( TAlloc)();
```

File

ZLib

Description

This is type ZLib.TAlloc.

201.4.2 ZLib.TFree Type

Pascal

```
TFree = procedure (AppData, Block: Pointer);
```

C++

```
(AppData, Block: Pointer) ( TFree)();
```

File

ZLib

Description

This is type ZLib.TFree.

201.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
Z_ASCII (see page 26640)	This is constant ZLib.Z_ASCII.
Z_BEST_COMPRESSION (see page 26641)	This is constant ZLib.Z_BEST_COMPRESSION.
Z_BEST_SPEED (see page 26641)	This is constant ZLib.Z_BEST_SPEED.
Z_BINARY (see page 26641)	This is constant ZLib.Z_BINARY.
Z_BUF_ERROR (see page 26642)	This is constant ZLib.Z_BUF_ERROR.
Z_DATA_ERROR (see page 26642)	This is constant ZLib.Z_DATA_ERROR.
Z_DEFAULT_COMPRESSION (see page 26642)	This is constant ZLib.Z_DEFAULT_COMPRESSION.
Z_DEFAULT_STRATEGY (see page 26643)	This is constant ZLib.Z_DEFAULT_STRATEGY.
Z_DEFLATED (see page 26643)	This is constant ZLib.Z_DEFLATED.
Z_ERRNO (see page 26643)	This is constant ZLib.Z_ERRNO.
Z_FILTERED (see page 26644)	This is constant ZLib.Z_FILTERED.
Z_FINISH (see page 26644)	This is constant ZLib.Z_FINISH.
Z_FULL_FLUSH (see page 26644)	This is constant ZLib.Z_FULL_FLUSH.
Z_HUFFMAN_ONLY (see page 26645)	This is constant ZLib.Z_HUFFMAN_ONLY.
Z_MEM_ERROR (see page 26645)	This is constant ZLib.Z_MEM_ERROR.
Z_NEED_DICT (see page 26645)	This is constant ZLib.Z_NEED_DICT.
Z_NO_COMPRESSION (see page 26646)	This is constant ZLib.Z_NO_COMPRESSION.
Z_NO_FLUSH (see page 26646)	This is constant ZLib.Z_NO_FLUSH.
Z_OK (see page 26646)	This is constant ZLib.Z_OK.
Z_PARTIAL_FLUSH (see page 26647)	This is constant ZLib.Z_PARTIAL_FLUSH.
Z_STREAM_END (see page 26647)	This is constant ZLib.Z_STREAM_END.
Z_STREAM_ERROR (see page 26647)	This is constant ZLib.Z_STREAM_ERROR.
Z_SYNC_FLUSH (see page 26648)	This is constant ZLib.Z_SYNC_FLUSH.
Z_UNKNOWN (see page 26648)	This is constant ZLib.Z_UNKNOWN.
Z_VERSION_ERROR (see page 26648)	This is constant ZLib.Z_VERSION_ERROR.
zlib_Version (see page 26649)	This is constant ZLib.zlib_Version.

201.5.1 ZLib.Z_ASCII Constant

Pascal

```
Z_ASCII = 1;
```

C++

```
Z_ASCII = 1;
```

File

ZLib

Description

This is constant ZLib.Z_ASCII.

201.5.2 ZLib.Z_BEST_COMPRESSION Constant

Pascal

```
Z_BEST_COMPRESSION = 9;
```

C++

```
Z_BEST_COMPRESSION = 9;
```

File

ZLib

Description

This is constant ZLib.Z_BEST_COMPRESSION.

201.5.3 ZLib.Z_BEST_SPEED Constant

Pascal

```
Z_BEST_SPEED = 1;
```

C++

```
Z_BEST_SPEED = 1;
```

File

ZLib

Description

This is constant ZLib.Z_BEST_SPEED.

201.5.4 ZLib.Z_BINARY Constant

Pascal

```
Z_BINARY = 0;
```

C++

```
Z_BINARY = 0;
```

File

ZLib

Description

This is constant ZLib.Z_BINARY.

201.5.5 ZLib.Z_BUF_ERROR Constant

Pascal

```
Z_BUF_ERROR = (-5);
```

C++

```
Z_BUF_ERROR = (-5);
```

File

ZLib

Description

This is constant ZLib.Z_BUF_ERROR.

201.5.6 ZLib.Z_DATA_ERROR Constant

Pascal

```
Z_DATA_ERROR = (-3);
```

C++

```
Z_DATA_ERROR = (-3);
```

File

ZLib

Description

This is constant ZLib.Z_DATA_ERROR.

201.5.7 ZLib.Z_DEFAULT_COMPRESSION Constant

Pascal

```
Z_DEFAULT_COMPRESSION = (-1);
```

C++

```
Z_DEFAULT_COMPRESSION = (-1);
```

File

ZLib

Description

This is constant ZLib.Z_DEFAULT_COMPRESSION.

201.5.8 ZLib.Z_DEFAULT_STRATEGY Constant

Pascal

```
Z_DEFAULT_STRATEGY = 0;
```

C++

```
Z_DEFAULT_STRATEGY = 0;
```

File

ZLib

Description

This is constant ZLib.Z_DEFAULT_STRATEGY.

201.5.9 ZLib.Z_DEFLATED Constant

Pascal

```
Z_DEFLATED = 8;
```

C++

```
Z_DEFLATED = 8;
```

File

ZLib

Description

This is constant ZLib.Z_DEFLATED.

201.5.10 ZLib.Z_ERRNO Constant

Pascal

```
Z_ERRNO = (-1);
```

C++

```
Z_ERRNO = (-1);
```

File

ZLib

Description

This is constant ZLib.Z_ERRNO.

201.5.11 ZLib.Z_FILTERED Constant

Pascal

```
Z_FILTERED = 1;
```

C++

```
Z_FILTERED = 1;
```

File

ZLib

Description

This is constant ZLib.Z_FILTERED.

201.5.12 ZLib.Z_FINISH Constant

Pascal

```
Z_FINISH = 4;
```

C++

```
Z_FINISH = 4;
```

File

ZLib

Description

This is constant ZLib.Z_FINISH.

201.5.13 ZLib.Z_FULL_FLUSH Constant

Pascal

```
Z_FULL_FLUSH = 3;
```

C++

```
Z_FULL_FLUSH = 3;
```

File

ZLib

Description

This is constant ZLib.Z_FULL_FLUSH.

201.5.14 ZLib.Z_HUFFMAN_ONLY Constant

Pascal

```
Z_HUFFMAN_ONLY = 2;
```

C++

```
Z_HUFFMAN_ONLY = 2;
```

File

ZLib

Description

This is constant ZLib.Z_HUFFMAN_ONLY.

201.5.15 ZLib.Z_MEM_ERROR Constant

Pascal

```
Z_MEM_ERROR = (-4);
```

C++

```
Z_MEM_ERROR = (-4);
```

File

ZLib

Description

This is constant ZLib.Z_MEM_ERROR.

201.5.16 ZLib.Z_NEED_DICT Constant

Pascal

```
Z_NEED_DICT = 2;
```

C++

```
Z_NEED_DICT = 2;
```

File

ZLib

Description

This is constant ZLib.Z_NEED_DICT.

201.5.17 ZLib.Z_NO_COMPRESSION Constant

Pascal

```
Z_NO_COMPRESSION = 0;
```

C++

```
Z_NO_COMPRESSION = 0;
```

File

ZLib

Description

This is constant ZLib.Z_NO_COMPRESSION.

201.5.18 ZLib.Z_NO_FLUSH Constant

Pascal

```
Z_NO_FLUSH = 0;
```

C++

```
Z_NO_FLUSH = 0;
```

File

ZLib

Description

This is constant ZLib.Z_NO_FLUSH.

201.5.19 ZLib.Z_OK Constant

Pascal

```
Z_OK = 0;
```

C++

```
Z_OK = 0;
```

File

ZLib

Description

This is constant ZLib.Z_OK.

201.5.20 ZLib.Z_PARTIAL_FLUSH Constant

Pascal

```
Z_PARTIAL_FLUSH = 1;
```

C++

```
Z_PARTIAL_FLUSH = 1;
```

File

ZLib

Description

This is constant ZLib.Z_PARTIAL_FLUSH.

201.5.21 ZLib.Z_STREAM_END Constant

Pascal

```
Z_STREAM_END = 1;
```

C++

```
Z_STREAM_END = 1;
```

File

ZLib

Description

This is constant ZLib.Z_STREAM_END.

201.5.22 ZLib.Z_STREAM_ERROR Constant

Pascal

```
Z_STREAM_ERROR = (-2);
```

C++

```
Z_STREAM_ERROR = (-2);
```

File

ZLib

Description

This is constant ZLib.Z_STREAM_ERROR.

201.5.23 ZLib.Z_SYNC_FLUSH Constant

Pascal

```
Z_SYNC_FLUSH = 2;
```

C++

```
Z_SYNC_FLUSH = 2;
```

File

ZLib

Description

This is constant ZLib.Z_SYNC_FLUSH.

201.5.24 ZLib.Z_UNKNOWN Constant

Pascal

```
Z_UNKNOWN = 2;
```

C++

```
Z_UNKNOWN = 2;
```

File

ZLib

Description

This is constant ZLib.Z_UNKNOWN.

201.5.25 ZLib.Z_VERSION_ERROR Constant

Pascal

```
Z_VERSION_ERROR = (-6);
```

C++

```
Z_VERSION_ERROR = (-6);
```

File

ZLib

Description

This is constant ZLib.Z_VERSION_ERROR.

201.5.26 ZLib.zlib_Version Constant

Pascal

```
zlib_Version = '1.0.4';
```

C++

```
zlib_Version = '1.0.4';
```

File

ZLib

Description

This is constant ZLib.zlib_Version.

Index

_

_SearchAssistantEvents interface 19276
- _SearchAssistantEvents Members 19277
- _SearchAssistantEvents methods 19277
- about _SearchAssistantEvents interface 19276
- OnNewSearch 19277
- OnNextMenuSelect 19277

A

aaDoubleClick enumeration member 18012
aaGetFocus enumeration member 18012
aaManual enumeration member 18012
acInternal enumeration member 17913
acMultiInstance enumeration member 17913
acoAutoAppend enumeration member 5440
acoAutoSuggest enumeration member 5440
acoFilterPrefixes enumeration member 5440
acoRtlReading enumeration member 5440
acoSearch enumeration member 5440
acoUpDownKeyDropsList enumeration member 5440
acoUseTab enumeration member 5440
acSingleInstance enumeration member 17913
ActnColorMaps 1
ActnColorMaps namespace 1
- Classes 2
- Constants 35

ActnColorMaps.cXPBtnFrameColor 35
ActnColorMaps.cXPBtnFrameColor constant 35
ActnColorMaps.cXPFrameOuter 35
ActnColorMaps.cXPFrameOuter constant 35
ActnColorMaps.cXPSelectedColor 35
ActnColorMaps.cXPSelectedColor constant 35
ActnColorMaps.TStandardColorMap 2
ActnColorMaps.TStandardColorMap.BtnFrameColor 7
ActnColorMaps.TStandardColorMap.BtnSelectedColor 7
ActnColorMaps.TStandardColorMap.BtnSelectedFont 7
ActnColorMaps.TStandardColorMap.Color 8
ActnColorMaps.TStandardColorMap.DisabledColor 8
ActnColorMaps.TStandardColorMap.DisabledFontColor 8
ActnColorMaps.TStandardColorMap.DisabledFontShadow 8
ActnColorMaps.TStandardColorMap.FontColor 9
ActnColorMaps.TStandardColorMap.FrameBottomRightInner 9
ActnColorMaps.TStandardColorMap.FrameBottomRightOuter 9
ActnColorMaps.TStandardColorMap.FrameTopLeftInner 10
ActnColorMaps.TStandardColorMap.FrameTopLeftOuter 10
ActnColorMaps.TStandardColorMap.HighlightColor 10
ActnColorMaps.TStandardColorMap.HotColor 10
ActnColorMaps.TStandardColorMap.HotFontColor 11
ActnColorMaps.TStandardColorMap.MenuColor 11
ActnColorMaps.TStandardColorMap.OnColorChange 12
ActnColorMaps.TStandardColorMap.SelectedColor 11
ActnColorMaps.TStandardColorMap.SelectedFontColor 11
ActnColorMaps.TStandardColorMap.SetColor 6
ActnColorMaps.TStandardColorMap.ShadowColor 12
ActnColorMaps.TStandardColorMap.UnusedColor 12
ActnColorMaps.TStandardColorMap.UpdateColors 6
ActnColorMaps.TTwilightColorMap 13
ActnColorMaps.TTwilightColorMap.BtnFrameColor 17
ActnColorMaps.TTwilightColorMap.BtnSelectedColor 18
ActnColorMaps.TTwilightColorMap.BtnSelectedFont 18
ActnColorMaps.TTwilightColorMap.Color 18
ActnColorMaps.TTwilightColorMap.DisabledColor 19
ActnColorMaps.TTwilightColorMap.DisabledFontColor 19
ActnColorMaps.TTwilightColorMap.DisabledFontShadow 19
ActnColorMaps.TTwilightColorMap.FontColor 19
ActnColorMaps.TTwilightColorMap.FrameBottomRightInner 20
ActnColorMaps.TTwilightColorMap.FrameBottomRightOuter 20
ActnColorMaps.TTwilightColorMap.FrameTopLeftInner 20
ActnColorMaps.TTwilightColorMap.FrameTopLeftOuter 20
ActnColorMaps.TTwilightColorMap.HighlightColor 21
ActnColorMaps.TTwilightColorMap.HotColor 21
ActnColorMaps.TTwilightColorMap.HotFontColor 21
ActnColorMaps.TTwilightColorMap.MenuColor 21
ActnColorMaps.TTwilightColorMap.OnColorChange 23
ActnColorMaps.TTwilightColorMap.SelectedColor 22

ActnColorMaps.TTwilightColorMap.SelectedFontColor 22
ActnColorMaps.TTwilightColorMap.ShadowColor 22
ActnColorMaps.TTwilightColorMap.UnusedColor 23
ActnColorMaps.TTwilightColorMap.UpdateColors 17
ActnColorMaps.TXPColorMap 23
ActnColorMaps.TXPColorMap.BtnFrameColor 28
ActnColorMaps.TXPColorMap.BtnSelectedColor 28
ActnColorMaps.TXPColorMap.BtnSelectedFont 29
ActnColorMaps.TXPColorMap.Color 29
ActnColorMaps.TXPColorMap.DisabledColor 29
ActnColorMaps.TXPColorMap.DisabledFontColor 29
ActnColorMaps.TXPColorMap.DisabledFontShadow 30
ActnColorMaps.TXPColorMap.FontColor 30
ActnColorMaps.TXPColorMap.FrameBottomRightInner 30
ActnColorMaps.TXPColorMap.FrameBottomRightOuter 30
ActnColorMaps.TXPColorMap.FrameTopLeftInner 31
ActnColorMaps.TXPColorMap.FrameTopLeftOuter 31
ActnColorMaps.TXPColorMap.HighlightColor 31
ActnColorMaps.TXPColorMap.HotColor 31
ActnColorMaps.TXPColorMap.HotFontColor 32
ActnColorMaps.TXPColorMap.MenuColor 32
ActnColorMaps.TXPColorMap.OnColorChange 34
ActnColorMaps.TXPColorMap.SelectedColor 32
ActnColorMaps.TXPColorMap.SelectedFontColor 33
ActnColorMaps.TXPColorMap.ShadowColor 33
ActnColorMaps.TXPColorMap.UnusedColor 33
ActnColorMaps.TXPColorMap.UpdateColors 27
ActnCtrls 37
ActnCtrls namespace 37
    Classes 38
    Types 245
ActnCtrls.TActionToolBar 38
ActnCtrls.TActionToolBar.ActionManager 52
ActnCtrls.TActionToolBar.Align 53
ActnCtrls.TActionToolBar.AllowHiding 56
ActnCtrls.TActionToolBar.Anchors 57
ActnCtrls.TActionToolBar.BiDiMode 57
ActnCtrls.TActionToolBar.Canvas 58
ActnCtrls.TActionToolBar.Caption 58
ActnCtrls.TActionToolBar.Color 61
ActnCtrls.TActionToolBar.ColorMap 61
ActnCtrls.TActionToolBar.Constraints 62
ActnCtrls.TActionToolBar.Cursor 63
ActnCtrls.TActionToolBar.DragCursor 65
ActnCtrls.TActionToolBar.DragKind 66
ActnCtrls.TActionToolBar.DragMode 66
ActnCtrls.TActionToolBar.EdgeBorders 68
ActnCtrls.TActionToolBar.EdgeInner 68
ActnCtrls.TActionToolBar.EdgeOuter 68
ActnCtrls.TActionToolBar.Enabled 69
ActnCtrls.TActionToolBar.Font 69
ActnCtrls.TActionToolBar.HorzMargin 70
ActnCtrls.TActionToolBar.HorzSeparator 71
ActnCtrls.TActionToolBar.OnAlignInsertBefore 83
ActnCtrls.TActionToolBar.OnAlignPosition 84
ActnCtrls.TActionToolBar.OnCanResize 84
ActnCtrls.TActionToolBar.OnClick 85
ActnCtrls.TActionToolBar.OnConstrainedResize 87
ActnCtrls.TActionToolBar.OnContextPopup 87
ActnCtrls.TActionToolBar.OnControlCreated 88
ActnCtrls.TActionToolBar.OnDblClick 88
ActnCtrls.TActionToolBar.OnDockDrop 89
ActnCtrls.TActionToolBar.OnDockOver 91
ActnCtrls.TActionToolBar.OnDragDrop 93
ActnCtrls.TActionToolBar.OnDragOver 95
ActnCtrls.TActionToolBar.OnEndDock 97
ActnCtrls.TActionToolBar.OnEndDrag 98
ActnCtrls.TActionToolBar.OnEnter 99
ActnCtrls.TActionToolBar.OnExit 101
ActnCtrls.TActionToolBar.OnGetControlClass 103
ActnCtrls.TActionToolBar.OnGetSiteInfo 103
ActnCtrls.TActionToolBar.OnMouseActivate 105
ActnCtrls.TActionToolBar.OnMouseDown 106
ActnCtrls.TActionToolBar.OnMouseMove 109
ActnCtrls.TActionToolBar.OnMouseUp 111
ActnCtrls.TActionToolBar.OnPaint 113
ActnCtrls.TActionToolBar.OnResize 113
ActnCtrls.TActionToolBar.OnStartDock 114

ActnCtrls.TActionToolBar.OnStartDrag 115
ActnCtrls.TActionToolBar.OnUnDock 115
ActnCtrls.TActionToolBar.Orientation 71
ActnCtrls.TActionToolBar.ParentBackground 71
ActnCtrls.TActionToolBar.ParentBiDiMode 72
ActnCtrls.TActionToolBar.ParentColor 72
ActnCtrls.TActionToolBar.ParentFont 73
ActnCtrls.TActionToolBar.ParentShowHint 73
ActnCtrls.TActionToolBar.PersistentHotKeys 74
ActnCtrls.TActionToolBar.PopupMenu 74
ActnCtrls.TActionToolBar.ShowHint 79
ActnCtrls.TActionToolBar.Spacing 80
ActnCtrls.TActionToolBar.VertMargin 81
ActnCtrls.TActionToolBar.Visible 81
ActnCtrls.TCustomActionCombo 118
ActnCtrls.TCustomActionCombo.BeginAutoDrag 131
ActnCtrls.TCustomActionCombo.ComboWndProc 131
ActnCtrls.TCustomActionCombo.DesignWndProc 132
ActnCtrls.TCustomActionCombo.DragDrop 132
ActnCtrls.TCustomActionCombo.DragOver 132
ActnCtrls.TCustomActionDockBar 133
ActnCtrls.TCustomActionDockBar.CalcDockedWidth 146
ActnCtrls.TCustomActionDockBar.Create 145
ActnCtrls.TCustomActionDockBar.Dock 146
ActnCtrls.TCustomActionDockBar.DoEndDock 146
ActnCtrls.TCustomActionDockBar.DoStartDock 147
ActnCtrls.TCustomActionDockBar.GetFloatingDockSiteClass 147
ActnCtrls.TCustomActionDockBar.GetSiteInfo 147
ActnCtrls.TCustomActionDockBar.Loaded 148
ActnCtrls.TCustomActionToolBar 148
ActnCtrls.TCustomActionToolBar.AutoSizingChanged 163
ActnCtrls.TCustomActionToolBar.CalcButtonWidth 163
ActnCtrls.TCustomActionToolBar.CalcDockedWidth 163
ActnCtrls.TCustomActionToolBar.CMColorchanged 163
ActnCtrls.TCustomActionToolBar.Create 161
ActnCtrls.TCustomActionToolBar.CreateControl 163
ActnCtrls.TCustomActionToolBar.CreateControls 164
ActnCtrls.TCustomActionToolBar.Destroy 162
ActnCtrls.TCustomActionToolBar.DisableHiding 164
ActnCtrls.TCustomActionToolBar.DoDropCategory 164
ActnCtrls.TCustomActionToolBar.DragDrop 164
ActnCtrls.TCustomActionToolBar.DrawBackground 165
ActnCtrls.TCustomActionToolBar.DrawSeparator 165
ActnCtrls.TCustomActionToolBar.EnableHiding 165
ActnCtrls.TCustomActionToolBar.GetControlClass 165
ActnCtrls.TCustomActionToolBar.GetPopupClass 166
ActnCtrls.TCustomActionToolBar.GetScrollBtnClass 166
ActnCtrls.TCustomActionToolBar.HiddenCount 168
ActnCtrls.TCustomActionToolBar.HideUnusedItems 166
ActnCtrls.TCustomActionToolBar.Reset 166
ActnCtrls.TCustomActionToolBar.ScrollBtnClick 166
ActnCtrls.TCustomActionToolBar.SetBounds 167
ActnCtrls.TCustomActionToolBar.SetOrientation 167
ActnCtrls.TCustomActionToolBar.SetupDropDownBtn 167
ActnCtrls.TCustomButtonControl 168
ActnCtrls.TCustomButtonControl.AllowAllUp 186
ActnCtrls.TCustomButtonControl.CalcBounds 179
ActnCtrls.TCustomButtonControl.Caption 186
ActnCtrls.TCustomButtonControl.Click 179
ActnCtrls.TCustomButtonControl.CMDialogChar 180
ActnCtrls.TCustomButtonControl.CMEnabledChanged 180
ActnCtrls.TCustomButtonControl.CMMouseEnter 180
ActnCtrls.TCustomButtonControl.CMMouseLeave 180
ActnCtrls.TCustomButtonControl.CMTextChanged 180
ActnCtrls.TCustomButtonControl.Create 177
ActnCtrls.TCustomButtonControl.Down 189
ActnCtrls.TCustomButtonControl.DrawBackground 181
ActnCtrls.TCustomButtonControl.DrawFrame 181
ActnCtrls.TCustomButtonControl.DrawLargeGlyph 181
ActnCtrls.TCustomButtonControl.FFlat 178
ActnCtrls.TCustomButtonControl.Flat 189
ActnCtrls.TCustomButtonControl.FState 178
ActnCtrls.TCustomButtonControl.GetShowShortCut 181
ActnCtrls.TCustomButtonControl.Loaded 182
ActnCtrls.TCustomButtonControl.MouseDown 182
ActnCtrls.TCustomButtonControl.MouseInControl 189
ActnCtrls.TCustomButtonControl.MouseMove 183
ActnCtrls.TCustomButtonControl.MouseUp 183

ActnCtrls.TCustomButtonControl.Paint 184
ActnCtrls.TCustomButtonControl.SetActionClient 184
ActnCtrls.TCustomButtonControl.SetBounds 184
ActnCtrls.TCustomButtonControl.SetGlyphLayout 185
ActnCtrls.TCustomButtonControl.SetSelected 185
ActnCtrls.TCustomButtonControl.ShowCaption 189
ActnCtrls.TCustomButtonControl.ShowShortCut 190
ActnCtrls.TCustomButtonControl.State 190
ActnCtrls.TCustomButtonControl.TrackButton 190
ActnCtrls.TCustomButtonControl.WMLButtonDblClk 185
ActnCtrls.TCustomComboControl 191
ActnCtrls.TCustomComboControl.CalcBounds 201
ActnCtrls.TCustomComboControl.Click 201
ActnCtrls.TCustomComboControl.CMVisiblechanged 201
ActnCtrls.TCustomComboControl.ComboBox 205
ActnCtrls.TCustomComboControl.ComboClick 202
ActnCtrls.TCustomComboControl.Create 199
ActnCtrls.TCustomComboControl.Destroy 200
ActnCtrls.TCustomComboControl.Paint 202
ActnCtrls.TCustomComboControl.SetActionClient 202
ActnCtrls.TCustomComboControl.SetBounds 202
ActnCtrls.TCustomComboControl.SetDragMode 203
ActnCtrls.TCustomComboControl.SetParent 204
ActnCtrls.TCustomComboControl.VisibleChanging 204
ActnCtrls.TCustomDropDownButton 205
ActnCtrls.TCustomDropDownButton.CalcBounds 214
ActnCtrls.TCustomDropDownButton.Click 215
ActnCtrls.TCustomDropDownButton.DrawFrame 215
ActnCtrls.TCustomDropDownButton.DropDownClick 215
ActnCtrls.TCustomDropDownButton.DroppedDown 217
ActnCtrls.TCustomDropDownButton.GetPopupClass 215
ActnCtrls.TCustomDropDownButton.MouseDown 216
ActnCtrls.TCustomDropDownButton.MouseOverDropDown 216
ActnCtrls.TCustomDropDownButton.Paint 216
ActnCtrls.TCustomToolScrollBtn 217
ActnCtrls.TCustomToolScrollBtn.Create 227
ActnCtrls.TCustomToolScrollBtn.DrawArrows 227
ActnCtrls.TCustomToolScrollBtnClass 245
ActnCtrls.TCustomToolScrollBtnClass type 245
ActnCtrls.TCustomUtilityButton 228
ActnCtrls.TCustomUtilityButton.ArrowSize 240
ActnCtrls.TCustomUtilityButton.ArrowType 241
ActnCtrls.TCustomUtilityButton.AutoScroll 241
ActnCtrls.TCustomUtilityButton.Click 238
ActnCtrls.TCustomUtilityButton.CMMouseEnter 238
ActnCtrls.TCustomUtilityButton.CMMouseLeave 239
ActnCtrls.TCustomUtilityButton.Color 241
ActnCtrls.TCustomUtilityButton.Create 237
ActnCtrls.TCustomUtilityButton.Destroy 237
ActnCtrls.TCustomUtilityButton.Direction 242
ActnCtrls.TCustomUtilityButton.DrawArrows 239
ActnCtrls.TCustomUtilityButton.MouseDown 239
ActnCtrls.TCustomUtilityButton.OnClick 243
ActnCtrls.TCustomUtilityButton.OnDelay 240
ActnCtrls.TCustomUtilityButton.Paint 240
ActnCtrls.TCustomUtilityButton.RepeatRate 242
ActnList 247
ActnList namespace 247
    Classes 249
    Constants 341
    Functions 334
    Structs, Records, Enums 337
    Types 338
ActnList.CreateAction 334
ActnList.CreateAction function 334
ActnList.CreateActionProc 341
ActnList.CreateActionProc constant 341
ActnList.EnumRegisteredActions 334
ActnList.EnumRegisteredActions function 334
ActnList.EnumRegisteredActionsProc 341
ActnList.EnumRegisteredActionsProc constant 341
ActnList.RegisterActions 335
ActnList.RegisterActions function 335
ActnList.RegisterActionsProc 342
ActnList.RegisterActionsProc constant 342
ActnList.TAction 249
ActnList.TAction.AutoCheck 255
ActnList.TAction.Caption 255

ActnList.TAction.Checked 256
ActnList.TAction.Create 254
ActnList.TAction.Enabled 257
ActnList.TAction.GroupIndex 257
ActnList.TAction.HelpContext 258
ActnList.TAction.HelpKeyword 258
ActnList.TAction.HelpType 259
ActnList.TAction.Hint 259
ActnList.TAction.ImageIndex 259
ActnList.TAction.OnExecute 261
ActnList.TAction.OnHint 263
ActnList.TAction.OnUpdate 263
ActnList.TAction.SecondaryShortCuts 260
ActnList.TAction.ShortCut 260
ActnList.TAction.Visible 261
ActnList.TActionEvent 338
ActnList.TActionEvent type 338
ActnList.TActionLink 265
ActnList.TActionLink.IsCaptionLinked 268
ActnList.TActionLink.IsCheckedLinked 269
ActnList.TActionLink.IsEnabledLinked 269
ActnList.TActionLink.IsGroupIndexLinked 270
ActnList.TActionLink.IsHelpContextLinked 270
ActnList.TActionLink.IsHelpLinked 271
ActnList.TActionLink.IsHintLinked 271
ActnList.TActionLink.IsImageIndexLinked 272
ActnList.TActionLink.IsShortCutLinked 272
ActnList.TActionLink.IsVisibleLinked 273
ActnList.TActionLink.SetAutoCheck 274
ActnList.TActionLink.SetCaption 274
ActnList.TActionLink.SetChecked 274
ActnList.TActionLink.SetEnabled 275
ActnList.TActionLink.SetGroupIndex 275
ActnList.TActionLink.SetHelpContext 276
ActnList.TActionLink.SetHelpKeyword 276
ActnList.TActionLink.SetHelpType 277
ActnList.TActionLink.SetHint 278
ActnList.TActionLink.SetImageIndex 278
ActnList.TActionLink.SetShortCut 279
ActnList.TActionLink.SetVisible 279
ActnList.TActionLinkClass 338
ActnList.TActionLinkClass type 338
ActnList.TActionList 280
ActnList.TActionList.Images 284
ActnList.TActionList.OnChange 285
ActnList.TActionList.OnExecute 285
ActnList.TActionList.OnStateChange 286
ActnList.TActionList.OnUpdate 286
ActnList.TActionList.State 284
ActnList.TActionListEnumerator 287
ActnList.TActionListEnumerator.Create 287
ActnList.TActionListEnumerator.Current 288
ActnList.TActionListEnumerator.GetCurrent 288
ActnList.TActionListEnumerator.MoveNext 288
ActnList.TActionListState 337
ActnList.TActionListState enumeration 337
ActnList.TContainedAction 289
ActnList.TContainedAction.ActionList 295
ActnList.TContainedAction.Category 296
ActnList.TContainedAction.Destroy 292
ActnList.TContainedAction.Execute 293
ActnList.TContainedAction.GetParentComponent 293
ActnList.TContainedAction.HasParent 294
ActnList.TContainedAction.Index 296
ActnList.TContainedAction.ReadState 294
ActnList.TContainedAction.SetParentComponent 294
ActnList.TContainedAction.Update 295
ActnList.TContainedActionClass 339
ActnList.TContainedActionClass type 339
ActnList.TCustomAction 297
ActnList.TCustomAction.AssignTo 303
ActnList.TCustomAction.AutoCheck 308
ActnList.TCustomAction.Caption 308
ActnList.TCustomAction.Checked 308
ActnList.TCustomAction.Create 301
ActnList.TCustomAction.Destroy 302
ActnList.TCustomAction.DisableIfNoHandler 310
ActnList.TCustomAction.DoHint 303

ActnList.TCustomAction.Enabled 310
ActnList.TCustomAction.Execute 304
ActnList.TCustomAction.FImage 302
ActnList.TCustomAction.FMask 303
ActnList.TCustomAction.GroupIndex 311
ActnList.TCustomAction.HandleShortCut 306
ActnList.TCustomAction.HelpContext 311
ActnList.TCustomAction.HelpKeyword 311
ActnList.TCustomAction.HelpType 312
ActnList.TCustomAction.Hint 312
ActnList.TCustomAction.ImageIndex 313
ActnList.TCustomAction.OnHint 315
ActnList.TCustomAction.SavedEnabledState 313
ActnList.TCustomAction.SecondaryShortCuts 314
ActnList.TCustomAction.SetHelpContext 306
ActnList.TCustomAction.SetHelpKeyword 306
ActnList.TCustomAction.SetName 307
ActnList.TCustomAction.ShortCut 314
ActnList.TCustomAction.Visible 314
ActnList.TCustomActionList 315
ActnList.TCustomActionList.ActionCount 324
ActnList.TCustomActionList.Actions 325
ActnList.TCustomActionList.AddAction 320
ActnList.TCustomActionList.Change 320
ActnList.TCustomActionList.Create 319
ActnList.TCustomActionList.Destroy 319
ActnList.TCustomActionList.ExecuteAction 321
ActnList.TCustomActionList.GetChildren 321
ActnList.TCustomActionList.GetEnumerator 321
ActnList.TCustomActionList.Images 325
ActnList.TCustomActionList.IsShortCut 322
ActnList.TCustomActionList.Notification 322
ActnList.TCustomActionList.OnChange 326
ActnList.TCustomActionList.OnExecute 327
ActnList.TCustomActionList.OnStateChange 327
ActnList.TCustomActionList.OnUpdate 327
ActnList.TCustomActionList.RemoveAction 323
ActnList.TCustomActionList.SetChildOrder 323
ActnList.TCustomActionList.SetImages 323
ActnList.TCustomActionList.SetState 324
ActnList.TCustomActionList.State 326
ActnList.TCustomActionList.UpdateAction 324
ActnList.TEnumActionProc 339
ActnList.TEnumActionProc type 339
ActnList.THintEvent 339
ActnList.THintEvent type 339
ActnList.TShortCutList 328
ActnList.TShortCutList.Add 332
ActnList.TShortCutList.IndexOfShortCut 332
ActnList.TShortCutList.ShortCuts 333
ActnList.UnRegisterActions 336
ActnList.UnRegisterActions function 336
ActnList.UnRegisterActionsProc 342
ActnList.UnRegisterActionsProc constant 342
ActnMan 343
ActnMan namespace 343
    Classes 346
    Constants 577
    Functions 563
    Interfaces 560
    Structs, Records, Enums 564
    Types 568
    Variables 575
ActnMan.ActionBarDesigner 575
ActnMan.ActionBarDesigner variable 575
ActnMan.ActionBarStyles 575
ActnMan.ActionBarStyles variable 575
ActnMan.caAllChanges 577
ActnMan.caAllChanges constant 577
ActnMan.cDefaultSchedule 577
ActnMan.cDefaultSchedule constant 577
ActnMan.CM_ENTERMENULOOP 578
ActnMan.CM_ENTERMENULOOP constant 578
ActnMan.CM_ITEMCLICKED 578
ActnMan.CM_ITEMCLICKED constant 578
ActnMan.CM_ITEMDROPPOINT 578
ActnMan.CM_ITEMDROPPOINT constant 578
ActnMan.CM_ITEMKEYED 579

ActnMan.CM_ITEMKEYED constant 579
ActnMan.CM_ITEMSELECTED 579
ActnMan.CM_ITEMSELECTED constant 579
ActnMan.CM_RESETBAR 579
ActnMan.CM_RESETBAR constant 579
ActnMan.CM_SCROLLCLICKED 580
ActnMan.CM_SCROLLCLICKED constant 580
ActnMan.ControlDragMode 580
ActnMan.ControlDragMode constant 580
ActnMan.DefaultActnBarStyle 575
ActnMan.DefaultActnBarStyle variable 575
ActnMan.IActionBarDesigner 560
ActnMan.IActionBarDesigner.CreateAction 561
ActnMan.IActionBarDesigner.EditAction 561
ActnMan.IActionBarDesigner.Modified 561
ActnMan.IActionBarDesigner.SetActiveMenu 561
ActnMan.IActionBarDesigner.SetItemSelection 562
ActnMan.IActionBarDesigner.SetSelection 562
ActnMan.NotifyDesigner 563
ActnMan.NotifyDesigner function 563
ActnMan.TActionBarItem 346
ActnMan.TActionBarItem.ActionBar 351
ActnMan.TActionBarItem.ActionBars 351
ActnMan.TActionBarItem.AutoSize 351
ActnMan.TActionBarItem.ColorChanged 349
ActnMan.TActionBarItem.Create 349
ActnMan.TActionBarItem.GetDisplayName 350
ActnMan.TActionBarItem.GlyphLayout 352
ActnMan.TActionBarItem.Refresh 350
ActnMan.TActionBarItem.SetVisible 350
ActnMan.TActionBarItemClass 568
ActnMan.TActionBarItemClass type 568
ActnMan.TActionBars 352
ActnMan.TActionBars.ActionBars 357
ActnMan.TActionBars.Add 355
ActnMan.TActionBars.Create 355
ActnMan.TActionBars.Customizable 357
ActnMan.TActionBars.DoShowHint 355
ActnMan.TActionBars.HintShortCuts 357
ActnMan.TActionBars.Loaded 356
ActnMan.TActionBars.Notify 356
ActnMan.TActionBars.SessionCount 358
ActnMan.TActionBars.ShowHints 358
ActnMan.TActionBarsClass 568
ActnMan.TActionBarsClass type 568
ActnMan.TActionBarStyle 358
ActnMan.TActionBarStyle.GetColorMapClass 359
ActnMan.TActionBarStyle.GetControlClass 359
ActnMan.TActionBarStyle.GetStyleName 360
ActnMan.TActionBarStyleList 360
ActnMan.TActionBarStyleList.Style 363
ActnMan.TActionClient 364
ActnMan.TActionClient.Accessible 370
ActnMan.TActionClient.ActionBar 370
ActnMan.TActionClient.Assign 367
ActnMan.TActionClient.Background 370
ActnMan.TActionClient.BackgroundLayout 371
ActnMan.TActionClient.ChangesAllowed 371
ActnMan.TActionClient.ChildActionBar 372
ActnMan.TActionClient.ClearActionBar 367
ActnMan.TActionClient.Color 372
ActnMan.TActionClient.ColorChanged 367
ActnMan.TActionClient.ContextItems 372
ActnMan.TActionClient.Create 366
ActnMan.TActionClient.Destroy 366
ActnMan.TActionClient.FreeChildren 368
ActnMan.TActionClient.HasBackground 368
ActnMan.TActionClient.HasItems 373
ActnMan.TActionClient.HasVisibleItems 368
ActnMan.TActionClient.IsChildItem 368
ActnMan.TActionClient.Items 373
ActnMan.TActionClient.OwningCollection 373
ActnMan.TActionClient.Refresh 369
ActnMan.TActionClient.SetChangesAllowed 369
ActnMan.TActionClient.SetColor 369
ActnMan.TActionClient.SetVisible 369
ActnMan.TActionClient.Tag 374
ActnMan.TActionClient.Visible 374

ActnMan.TActionClientClass 569
ActnMan.TActionClientClass type 569
ActnMan.TActionClientItem 374
ActnMan.TActionClientItem.Action 382
ActnMan.TActionClientItem.ActionChange 378
ActnMan.TActionClientItem.ActionClients 383
ActnMan.TActionClientItem.ActionLink 383
ActnMan.TActionClientItem.Assign 378
ActnMan.TActionClientItem.Caption 384
ActnMan.TActionClientItem.CheckUnused 384
ActnMan.TActionClientItem.ColorChanged 379
ActnMan.TActionClientItem.Control 384
ActnMan.TActionClientItem.Create 377
ActnMan.TActionClientItem.Destroy 377
ActnMan.TActionClientItem.GetActionLinkClass 379
ActnMan.TActionClientItem.GetDisplayName 379
ActnMan.TActionClientItem.HasGlyph 379
ActnMan.TActionClientItem.HelpContext 385
ActnMan.TActionClientItem.ImageIndex 385
ActnMan.TActionClientItem.InitiateAction 380
ActnMan.TActionClientItem.LastSession 385
ActnMan.TActionClientItem.MergeIndex 386
ActnMan.TActionClientItem.ParentItem 386
ActnMan.TActionClientItem.Refresh 380
ActnMan.TActionClientItem.ResetUsageData 380
ActnMan.TActionClientItem.Separator 386
ActnMan.TActionClientItem.SetChangesAllowed 380
ActnMan.TActionClientItem.SetCollection 381
ActnMan.TActionClientItem.SetIndex 381
ActnMan.TActionClientItem.SetVisible 381
ActnMan.TActionClientItem.ShortCut 387
ActnMan.TActionClientItem.ShortCutText 387
ActnMan.TActionClientItem.ShowCaption 387
ActnMan.TActionClientItem.ShowGlyph 388
ActnMan.TActionClientItem.ShowShortCut 388
ActnMan.TActionClientItem.Unused 382
ActnMan.TActionClientItem.UsageCount 388
ActnMan.TActionClientItemClass 569
ActnMan.TActionClientItemClass type 569
ActnMan.TActionClientLink 389
ActnMan.TActionClientLink.AssignClient 392
ActnMan.TActionClientLink.DoShowHint 392
ActnMan.TActionClientLink.FClient 391
ActnMan.TActionClientLink.IsCaptionLinked 392
ActnMan.TActionClientLink.IsEnabledLinked 393
ActnMan.TActionClientLink.IsHelpContextLinked 393
ActnMan.TActionClientLink.IsHintLinked 394
ActnMan.TActionClientLink.IsImageIndexLinked 394
ActnMan.TActionClientLink.IsOnExecuteLinked 395
ActnMan.TActionClientLink.IsShortCutLinked 395
ActnMan.TActionClientLink.IsVisibleLinked 396
ActnMan.TActionClientLink.SetCaption 396
ActnMan.TActionClientLink.SetChecked 397
ActnMan.TActionClientLink.SetEnabled 397
ActnMan.TActionClientLink.SetHelpContext 398
ActnMan.TActionClientLink.SetHint 398
ActnMan.TActionClientLink.SetImageIndex 399
ActnMan.TActionClientLink.SetOnExecute 399
ActnMan.TActionClientLink.SetShortCut 399
ActnMan.TActionClientLink.SetVisible 400
ActnMan.TActionClientLinkClass 569
ActnMan.TActionClientLinkClass type 569
ActnMan.TActionClients 400
ActnMan.TActionClients.ActionClients 406
ActnMan.TActionClients.ActionManager 406
ActnMan.TActionClients.Add 404
ActnMan.TActionClients.Assign 404
ActnMan.TActionClients.AutoHotKeys 407
ActnMan.TActionClients.CaptionOptions 407
ActnMan.TActionClients.Create 403
ActnMan.TActionClients.Customizable 407
ActnMan.TActionClients.DoHideUnused 404
ActnMan.TActionClients.DoShowCaptions 404
ActnMan.TActionClients.HideUnused 408
ActnMan.TActionClients.Notify 405
ActnMan.TActionClients.SmallIcons 408
ActnMan.TActionClients.ToggleSmallIcons 405
ActnMan.TActionClients.VisibleCount 405

ActnMan.TActionClientsClass 570
ActnMan.TActionClientsClass type 570
ActnMan.TActionClientsCollection 408
ActnMan.TActionClientsCollection.ActionClients 413
ActnMan.TActionClientsCollection.ActionManager 414
ActnMan.TActionClientsCollection.AfterConstruction 411
ActnMan.TActionClientsCollection.AutoHotKeys 414
ActnMan.TActionClientsCollection.Customizable 415
ActnMan.TActionClientsCollection.GetActionManager 412
ActnMan.TActionClientsCollection.InternalRethinkHotkeys 412
ActnMan.TActionClientsCollection.IterateClients 412
ActnMan.TActionClientsCollection.ParentItem 415
ActnMan.TActionDragBaseClass 415
ActnMan.TActionDragBaseClass.ActionManager 417
ActnMan.TActionDragObject 418
ActnMan.TActionDragObject.ActionCount 422
ActnMan.TActionDragObject.Actions 422
ActnMan.TActionDragObject.AddAction 421
ActnMan.TActionDragObject.AddActionClass 421
ActnMan.TActionDragObject.Create 419
ActnMan.TActionDragObject.Destroy 420
ActnMan.TActionDragObject.FActionClasses 421
ActnMan.TActionItemDragObject 422
ActnMan.TActionItemDragObject.ClientItem 425
ActnMan.TActionItemDragObject.Finished 424
ActnMan.TActionListCollection 425
ActnMan.TActionListCollection.ActionManager 427
ActnMan.TActionListCollection.ListItems 428
ActnMan.TActionListItem 428
ActnMan.TActionListItem.ActionList 431
ActnMan.TActionListItem.Assign 430
ActnMan.TActionListItem.Caption 432
ActnMan.TActionListItem.GetDisplayName 430
ActnMan.TActionListItem.Owner 431
ActnMan.TActionManager 432
ActnMan.TActionManager.ActionBars 438
ActnMan.TActionManager.FileName 438
ActnMan.TActionManager.Images 438
ActnMan.TActionManager.LinkedActionLists 439
ActnMan.TActionManager.OnChange 441
ActnMan.TActionManager.OnExecute 442
ActnMan.TActionManager.OnGetBarItemClass 442
ActnMan.TActionManager.OnGetBarsClass 442
ActnMan.TActionManager.OnGetClientItemClass 443
ActnMan.TActionManager.OnGetClientsClass 443
ActnMan.TActionManager.OnStateChange 443
ActnMan.TActionManager.OnStyleChanged 444
ActnMan.TActionManager.OnUpdate 444
ActnMan.TActionManager.PrioritySchedule 439
ActnMan.TActionManager.State 440
ActnMan.TActionManager.Style 440
ActnMan.TActionProc 570
ActnMan.TActionProc type 570
ActnMan.TBackgroundLayout 564
ActnMan.TBackgroundLayout enumeration 564
ActnMan.TBarEdge 564
ActnMan.TBarEdge enumeration 564
ActnMan.TBarOrientation 565
ActnMan.TBarOrientation enumeration 565
ActnMan.TCaptionOptions 565
ActnMan.TCaptionOptions enumeration 565
ActnMan.TCategoryDragObject 444
ActnMan.TCategoryDragObject.Category 447
ActnMan.TCategoryDragObject.Create 446
ActnMan.TChangesAllowed 566
ActnMan.TChangesAllowed enumeration 566
ActnMan.TChangesAllowedSet 570
ActnMan.TChangesAllowedSet type 570
ActnMan.TCMItemMsg 566
ActnMan.TCMItemMsg record 566
ActnMan.TColorMapUpdateProc 571
ActnMan.TColorMapUpdateProc type 571
ActnMan.TControlCreatedEvent 571
ActnMan.TControlCreatedEvent type 571
ActnMan.TCustomActionBar 447
ActnMan.TCustomActionBar.ActionClient 479
ActnMan.TCustomActionBar.ActionControls 479
ActnMan.TCustomActionBar.ActionManager 480

ActnMan.TCustomActionBar.AlignControls 462
ActnMan.TCustomActionBar.AllowHiding 480
ActnMan.TCustomActionBar.AutoSizing 480
ActnMan.TCustomActionBar.AutoSizingChanged 462
ActnMan.TCustomActionBar.BackgroundSize 462
ActnMan.TCustomActionBar.CalcInsertPosition 462
ActnMan.TCustomActionBar.Canvas 481
ActnMan.TCustomActionBar.Caption 481
ActnMan.TCustomActionBar.Clear 463
ActnMan.TCustomActionBar.Color 483
ActnMan.TCustomActionBar.ColorMap 484
ActnMan.TCustomActionBar.ContextBar 485
ActnMan.TCustomActionBar.Create 459
ActnMan.TCustomActionBar.CreateControl 463
ActnMan.TCustomActionBar.CreateControls 463
ActnMan.TCustomActionBar.CustomAlignInsertBefore 463
ActnMan.TCustomActionBar.CustomAlignPosition 464
ActnMan.TCustomActionBar.Designable 485
ActnMan.TCustomActionBar.DesignMode 486
ActnMan.TCustomActionBar.DesignWndProc 465
ActnMan.TCustomActionBar.Destroy 459
ActnMan.TCustomActionBar.DoControlCreated 465
ActnMan.TCustomActionBar.DoDropActions 465
ActnMan.TCustomActionBar.DoDropCategory 465
ActnMan.TCustomActionBar.DoDropItem 466
ActnMan.TCustomActionBar.DoGetControlClass 466
ActnMan.TCustomActionBar.DragDrop 466
ActnMan.TCustomActionBar.DragOver 466
ActnMan.TCustomActionBar.DrawBackground 467
ActnMan.TCustomActionBar.FDefaultColorMap 460
ActnMan.TCustomActionBar.FindAccelItem 467
ActnMan.TCustomActionBar.FindFirst 467
ActnMan.TCustomActionBar.FindFirstVisibleItem 468
ActnMan.TCustomActionBar.FindItemAt 468
ActnMan.TCustomActionBar.FindItemCaption 468
ActnMan.TCustomActionBar.FindLast 468
ActnMan.TCustomActionBar.FindLastVisibleItem 469
ActnMan.TCustomActionBar.FindLeastUsedItem 469
ActnMan.TCustomActionBar.FindNearestControl 469
ActnMan.TCustomActionBar.FindNext 470
ActnMan.TCustomActionBar.FindNextVisibleItem 470
ActnMan.TCustomActionBar.FindPrevious 470
ActnMan.TCustomActionBar.FindPreviousVisibleItem 470
ActnMan.TCustomActionBar.Font 486
ActnMan.TCustomActionBar.FSavedWrapState 460
ActnMan.TCustomActionBar.GetActionControl 471
ActnMan.TCustomActionBar.GetBannerWidth 471
ActnMan.TCustomActionBar.GetBarHeight 471
ActnMan.TCustomActionBar.GetBarWidth 471
ActnMan.TCustomActionBar.GetControlClass 471
ActnMan.TCustomActionBar.GetDefaultColorMapClass 472
ActnMan.TCustomActionBar.GetDesignMode 472
ActnMan.TCustomActionBar.GetPopupMenuClass 472
ActnMan.TCustomActionBar.HasItems 472
ActnMan.TCustomActionBar.HorzMargin 487
ActnMan.TCustomActionBar.HorzSeparator 487
ActnMan.TCustomActionBar.HRowCount 488
ActnMan.TCustomActionBar.ItemCount 473
ActnMan.TCustomActionBar.Items 488
ActnMan.TCustomActionBar.Loaded 473
ActnMan.TCustomActionBar.NCPaint 473
ActnMan.TCustomActionBar.Notification 473
ActnMan.TCustomActionBar.OnControlCreated 491
ActnMan.TCustomActionBar.OnGetControlClass 492
ActnMan.TCustomActionBar.OnPaint 492
ActnMan.TCustomActionBar.Orientation 488
ActnMan.TCustomActionBar.Paint 474
ActnMan.TCustomActionBar.PaintWindow 474
ActnMan.TCustomActionBar.PersistentHotKeys 489
ActnMan.TCustomActionBar.RecreateControls 475
ActnMan.TCustomActionBar.Reset 475
ActnMan.TCustomActionBar.SetActionClient 475
ActnMan.TCustomActionBar.SetAutoSizing 475
ActnMan.TCustomActionBar.SetBiDiMode 475
ActnMan.TCustomActionBar.SetBounds 476
ActnMan.TCustomActionBar.SetColorMap 476
ActnMan.TCustomActionBar.SetDesignMode 476
ActnMan.TCustomActionBar.SetOrientation 477

ActnMan.TCustomActionBar.SetPersistentHotKeys 477
ActnMan.TCustomActionBar.SetSpacing 477
ActnMan.TCustomActionBar.SetupDefaultColorMap 477
ActnMan.TCustomActionBar.Spacing 489
ActnMan.TCustomActionBar.Style 477
ActnMan.TCustomActionBar.VertMargin 490
ActnMan.TCustomActionBar.VertSeparator 490
ActnMan.TCustomActionBar.VisibleChanging 478
ActnMan.TCustomActionBar.VRowCount 490
ActnMan.TCustomActionBar.Widest 491
ActnMan.TCustomActionBarClass 571
ActnMan.TCustomActionBarClass type 571
ActnMan.TCustomActionBarColorMap 492
ActnMan.TCustomActionBarColorMap.Add 497
ActnMan.TCustomActionBarColorMap.Assign 497
ActnMan.TCustomActionBarColorMap.BtnFrameColor 500
ActnMan.TCustomActionBarColorMap.BtnSelectedColor 500
ActnMan.TCustomActionBarColorMap.BtnSelectedFont 501
ActnMan.TCustomActionBarColorMap.Color 501
ActnMan.TCustomActionBarColorMap.Create 496
ActnMan.TCustomActionBarColorMap.DesignFocus 501
ActnMan.TCustomActionBarColorMap.Destroy 496
ActnMan.TCustomActionBarColorMap.DisabledColor 501
ActnMan.TCustomActionBarColorMap.DisabledFontColor 502
ActnMan.TCustomActionBarColorMap.DisabledFontShadow 502
ActnMan.TCustomActionBarColorMap.FontColor 502
ActnMan.TCustomActionBarColorMap.FrameBottomRightInner 502
ActnMan.TCustomActionBarColorMap.FrameBottomRightOuter 503
ActnMan.TCustomActionBarColorMap.FrameTopLeftInner 503
ActnMan.TCustomActionBarColorMap.FrameTopLeftOuter 503
ActnMan.TCustomActionBarColorMap.HighlightColor 503
ActnMan.TCustomActionBarColorMap.HotColor 504
ActnMan.TCustomActionBarColorMap.HotFontColor 504
ActnMan.TCustomActionBarColorMap.MenuColor 504
ActnMan.TCustomActionBarColorMap.Notification 498
ActnMan.TCustomActionBarColorMap.OnColorChange 506
ActnMan.TCustomActionBarColorMap.Remove 498
ActnMan.TCustomActionBarColorMap.SelectedColor 505
ActnMan.TCustomActionBarColorMap.SelectedFontColor 505
ActnMan.TCustomActionBarColorMap.SetColor 499
ActnMan.TCustomActionBarColorMap.SetFontColor 499
ActnMan.TCustomActionBarColorMap.ShadowColor 505
ActnMan.TCustomActionBarColorMap.UnusedColor 505
ActnMan.TCustomActionBarColorMap.UpdateActionBars 499
ActnMan.TCustomActionBarColorMap.UpdateColors 499
ActnMan.TCustomActionClass 506
ActnMan.TCustomActionControl 507
ActnMan.TCustomActionControl.ActionBar 528
ActnMan.TCustomActionControl.ActionClient 529
ActnMan.TCustomActionControl.BeginAutoDrag 517
ActnMan.TCustomActionControl.CalcBounds 518
ActnMan.TCustomActionControl.CalcLayout 518
ActnMan.TCustomActionControl.Caption 529
ActnMan.TCustomActionControl.Click 518
ActnMan.TCustomActionControl.CMDesignHitTest 519
ActnMan.TCustomActionControl.CMFontChanged 519
ActnMan.TCustomActionControl.CMItemDropPoint 519
ActnMan.TCustomActionControl.CMItemSelected 519
ActnMan.TCustomActionControl.Create 515
ActnMan.TCustomActionControl.DesignWndProc 519
ActnMan.TCustomActionControl.Destroy 516
ActnMan.TCustomActionControl.DoDragDrop 520
ActnMan.TCustomActionControl.DoStartDrag 520
ActnMan.TCustomActionControl.DragDrop 520
ActnMan.TCustomActionControl.DragOver 521
ActnMan.TCustomActionControl.DrawBackground 521
ActnMan.TCustomActionControl.DrawDesignFocus 521
ActnMan.TCustomActionControl.DrawDragDropPoint 522
ActnMan.TCustomActionControl.DrawGlyph 522
ActnMan.TCustomActionControl.DrawLargeGlyph 522
ActnMan.TCustomActionControl.DrawSeparator 522
ActnMan.TCustomActionControl.DrawShadowedText 523
ActnMan.TCustomActionControl.DrawText 523
ActnMan.TCustomActionControl.DropPoint 532
ActnMan.TCustomActionControl.GetAction 523
ActnMan.TCustomActionControl.GetImageSize 523

ActnMan.TCustomActionControl.GetSeparator 524
ActnMan.TCustomActionControl.GetShowCaption 524
ActnMan.TCustomActionControl.GetShowShortCut 524
ActnMan.TCustomActionControl.GlyphLayout 532
ActnMan.TCustomActionControl.GlyphPos 532
ActnMan.TCustomActionControl.HasGlyph 524
ActnMan.TCustomActionControl.InitiateAction 524
ActnMan.TCustomActionControl.IsChecked 525
ActnMan.TCustomActionControl.IsGrouped 525
ActnMan.TCustomActionControl.Keyed 525
ActnMan.TCustomActionControl.Margins 532
ActnMan.TCustomActionControl.OnClick 535
ActnMan.TCustomActionControl.Paint 526
ActnMan.TCustomActionControl.PositionChanged 526
ActnMan.TCustomActionControl.ResetUsageData 526
ActnMan.TCustomActionControl.Selected 533
ActnMan.TCustomActionControl.Separator 533
ActnMan.TCustomActionControl.SetActionClient 526
ActnMan.TCustomActionControl.SetDropPoint 527
ActnMan.TCustomActionControl.SetGlyphLayout 527
ActnMan.TCustomActionControl.SetSelected 527
ActnMan.TCustomActionControl.ShowCaption 533
ActnMan.TCustomActionControl.ShowShortCut 533
ActnMan.TCustomActionControl.SmallIcon 534
ActnMan.TCustomActionControl.Spacing 534
ActnMan.TCustomActionControl.TextBounds 534
ActnMan.TCustomActionControl.Transparent 534
ActnMan.TCustomActionControl.UpdateSelection 527
ActnMan.TCustomActionControl.UpdateTextBounds 527
ActnMan.TCustomActionControl.WMSetText 528
ActnMan.TCustomActionControlClass 572
ActnMan.TCustomActionControlClass type 572
ActnMan.TCustomActionManager 537
ActnMan.TCustomActionManager.ActionBars 553
ActnMan.TCustomActionManager.AddAction 543
ActnMan.TCustomActionManager.AddCategory 543
ActnMan.TCustomActionManager.AddSeparator 544
ActnMan.TCustomActionManager.AfterConstruction 544
ActnMan.TCustomActionManager.Assign 544
ActnMan.TCustomActionManager.Change 545
ActnMan.TCustomActionManager.Create 541
ActnMan.TCustomActionManager.CreateActionBars 545
ActnMan.TCustomActionManager.DefaultActionBars 554
ActnMan.TCustomActionManager.DefineProperties 545
ActnMan.TCustomActionManager.DeleteActionItems 546
ActnMan.TCustomActionManager.DeleteItem 546
ActnMan.TCustomActionManager.Destroy 542
ActnMan.TCustomActionManager.FileName 554
ActnMan.TCustomActionManager.FindActionBar 547
ActnMan.TCustomActionManager.FindActionClient 547
ActnMan.TCustomActionManager.FindClient 547
ActnMan.TCustomActionManager.FindItemByAction 547
ActnMan.TCustomActionManager.FindItemByCaption 548
ActnMan.TCustomActionManager.GetActionBarItemClass 548
ActnMan.TCustomActionManager.GetActionBarsClass 548
ActnMan.TCustomActionManager.GetActionClientItemClass 548
ActnMan.TCustomActionManager.GetActionClientsClass 549
ActnMan.TCustomActionManager.GetUsage 549
ActnMan.TCustomActionManager.HasLinkedActionLists 549
ActnMan.TCustomActionManager.Images 554
ActnMan.TCustomActionManager.LinkedActionLists 555
ActnMan.TCustomActionManager.Loaded 549
ActnMan.TCustomActionManager.LoadFromFile 550
ActnMan.TCustomActionManager.LoadFromStream 550
ActnMan.TCustomActionManager.Loading 555
ActnMan.TCustomActionManager.LoadMenu 551
ActnMan.TCustomActionManager.Notification 551
ActnMan.TCustomActionManager.OnGetBarItemClass 557
ActnMan.TCustomActionManager.OnGetBarsClass 558
ActnMan.TCustomActionManager.OnGetClientItemClass 558
ActnMan.TCustomActionManager.OnGetClientsClass 558
ActnMan.TCustomActionManager.OnStyleChanged 558
ActnMan.TCustomActionManager.PrioritySchedule 555
ActnMan.TCustomActionManager.ResetActionBar 551
ActnMan.TCustomActionManager.ResetUsageData 551
ActnMan.TCustomActionManager.SaveToFile 552
ActnMan.TCustomActionManager.SaveToStream 552
ActnMan.TCustomActionManager.SetImages 552

ActnMan.TCustomActionManager.SetState 553
ActnMan.TCustomActionManager.SetupActionBars 553
ActnMan.TCustomActionManager.Style 556
ActnMan.TCustomColorMapClass 572
ActnMan.TCustomColorMapClass type 572
ActnMan.TGetBarItemClassEvent 572
ActnMan.TGetBarItemClassEvent type 572
ActnMan.TGetBarsClassEvent 573
ActnMan.TGetBarsClassEvent type 573
ActnMan.TGetClientItemClassEvent 573
ActnMan.TGetClientItemClassEvent type 573
ActnMan.TGetClientsClassEvent 573
ActnMan.TGetClientsClassEvent type 573
ActnMan.TGetControlClassEvent 574
ActnMan.TGetControlClassEvent type 574
ActnMan.TMergeRange 574
ActnMan.TMergeRange type 574
ActnMan.TStyleChanged 574
ActnMan.TStyleChanged type 574
ActnMan.TXToolDockForm 559
ActnMenus 581
ActnMenus namespace 581
    Classes 583
    Functions 837
    Structs, Records, Enums 839
    Types 841
    Variables 846
ActnMenus.MenuAddRemoveItemClass 846
ActnMenus.MenuAddRemoveItemClass variable 846
ActnMenus.MenuButtonControlClass 846
ActnMenus.MenuButtonControlClass variable 846
ActnMenus.MenuCustomizePopupClass 847
ActnMenus.MenuCustomizePopupClass variable 847
ActnMenus.MenuItemControlClass 847
ActnMenus.MenuItemControlClass variable 847
ActnMenus.MenuPopupClass 847
ActnMenus.MenuPopupClass variable 847
ActnMenus.RegisterActnBarStyle 837
ActnMenus.RegisterActnBarStyle function 837
ActnMenus.TActionBarStyleEx 583
ActnMenus.TActionBarStyleEx.GetAddRemoveItemClass 584
ActnMenus.TActionBarStyleEx.GetPopupClass 584
ActnMenus.TActionBarStyleEx.GetScrollBtnClass 585
ActnMenus.TActionMainMenuBar 585
ActnMenus.TActionMainMenuBar.ActionManager 601
ActnMenus.TActionMainMenuBar.Align 601
ActnMenus.TActionMainMenuBar.AllowHiding 605
ActnMenus.TActionMainMenuBar.Anchors 605
ActnMenus.TActionMainMenuBar.AnimateDuration 605
ActnMenus.TActionMainMenuBar.AnimationStyle 606
ActnMenus.TActionMainMenuBar.AutoFocus 606
ActnMenus.TActionMainMenuBar.BiDiMode 606
ActnMenus.TActionMainMenuBar.Canvas 607
ActnMenus.TActionMainMenuBar.Caption 607
ActnMenus.TActionMainMenuBar.Color 610
ActnMenus.TActionMainMenuBar.ColorMap 610
ActnMenus.TActionMainMenuBar.Constraints 611
ActnMenus.TActionMainMenuBar.DragCursor 612
ActnMenus.TActionMainMenuBar.DragKind 612
ActnMenus.TActionMainMenuBar.DragMode 612
ActnMenus.TActionMainMenuBar.EdgeBorders 614
ActnMenus.TActionMainMenuBar.EdgeInner 614
ActnMenus.TActionMainMenuBar.EdgeOuter 615
ActnMenus.TActionMainMenuBar.Enabled 615
ActnMenus.TActionMainMenuBar.ExpandDelay 616
ActnMenus.TActionMainMenuBar.Font 616
ActnMenus.TActionMainMenuBar.HorzMargin 617
ActnMenus.TActionMainMenuBar.OnAlignInsertBefore 625
ActnMenus.TActionMainMenuBar.OnAlignPosition 626
ActnMenus.TActionMainMenuBar.OnCanResize 627
ActnMenus.TActionMainMenuBar.OnClick 627
ActnMenus.TActionMainMenuBar.OnConstrainedResize 629
ActnMenus.TActionMainMenuBar.OnContextPopup 629
ActnMenus.TActionMainMenuBar.OnControlCreated 630
ActnMenus.TActionMainMenuBar.OnDblClick 631
ActnMenus.TActionMainMenuBar.OnDockDrop 631
ActnMenus.TActionMainMenuBar.OnDockOver 633
ActnMenus.TActionMainMenuBar.OnDragDrop 635

ActnMenus.TActionMainMenuBar.OnDragOver 637
ActnMenus.TActionMainMenuBar.OnEndDock 640
ActnMenus.TActionMainMenuBar.OnEndDrag 640
ActnMenus.TActionMainMenuBar.OnEnter 641
ActnMenus.TActionMainMenuBar.OnEnterMenuLoop 643
ActnMenus.TActionMainMenuBar.OnExit 644
ActnMenus.TActionMainMenuBar.OnExitMenuLoop 645
ActnMenus.TActionMainMenuBar.OnGetControlClass 645
ActnMenus.TActionMainMenuBar.OnGetPopupClass 646
ActnMenus.TActionMainMenuBar.OnGetSiteInfo 646
ActnMenus.TActionMainMenuBar.OnMouseActivate 648
ActnMenus.TActionMainMenuBar.OnMouseDown 649
ActnMenus.TActionMainMenuBar.OnMouseMove 651
ActnMenus.TActionMainMenuBar.OnMouseUp 654
ActnMenus.TActionMainMenuBar.OnPaint 656
ActnMenus.TActionMainMenuBar.OnPopup 656
ActnMenus.TActionMainMenuBar.OnResize 657
ActnMenus.TActionMainMenuBar.OnStartDock 657
ActnMenus.TActionMainMenuBar.OnStartDrag 658
ActnMenus.TActionMainMenuBar.OnUnDock 658
ActnMenus.TActionMainMenuBar.ParentBackground 617
ActnMenus.TActionMainMenuBar.ParentBiDiMode 617
ActnMenus.TActionMainMenuBar.ParentColor 618
ActnMenus.TActionMainMenuBar.ParentFont 618
ActnMenus.TActionMainMenuBar.ParentShowHint 619
ActnMenus.TActionMainMenuBar.PersistentHotKeys 619
ActnMenus.TActionMainMenuBar.Shadows 620
ActnMenus.TActionMainMenuBar.ShowHint 620
ActnMenus.TActionMainMenuBar.Spacing 622
ActnMenus.TActionMainMenuBar.UseSystemFont 622
ActnMenus.TActionMainMenuBar.VertMargin 622
ActnMenus.TActionMainMenuBar.Visible 623
ActnMenus.TActionMainMenuBar.WindowMenu 624
ActnMenus.TAddRemoveItemClass 841
ActnMenus.TAddRemoveItemClass type 841
ActnMenus.TAnimationStyle 839
ActnMenus.TAnimationStyle enumeration 839
ActnMenus.TCustomActionMainMenuBar 661
ActnMenus.TCustomActionMainMenuBar.CanFocus 677
ActnMenus.TCustomActionMainMenuBar.CloseMenu 677
ActnMenus.TCustomActionMainMenuBar.CMWininichange 678
ActnMenus.TCustomActionMainMenuBar.Create 675
ActnMenus.TCustomActionMainMenuBar.CreateControl 678
ActnMenus.TCustomActionMainMenuBar.CreateControls 678
ActnMenus.TCustomActionMainMenuBar.CustomAlignPosition 678
ActnMenus.TCustomActionMainMenuBar.DesignWndProc 679
ActnMenus.TCustomActionMainMenuBar.Destroy 676
ActnMenus.TCustomActionMainMenuBar.DoneMenuLoop 679
ActnMenus.TCustomActionMainMenuBar.FWindowMenuItem 676
ActnMenus.TCustomActionMainMenuBar.GetActiveMDIChild 679
ActnMenus.TCustomActionMainMenuBar.Inactive 683
ActnMenus.TCustomActionMainMenuBar.MainWndHook 680
ActnMenus.TCustomActionMainMenuBar.MDIChildMaximized 680
ActnMenus.TCustomActionMainMenuBar.MDIChildRestored 680
ActnMenus.TCustomActionMainMenuBar.NewPopup 680
ActnMenus.TCustomActionMainMenuBar.Notification 681
ActnMenus.TCustomActionMainMenuBar.PersistentHotKeys 684
ActnMenus.TCustomActionMainMenuBar.ProcessMouseMsg 681
ActnMenus.TCustomActionMainMenuBar.RefreshMDIMenu 681
ActnMenus.TCustomActionMainMenuBar.SetDesignMode 681
ActnMenus.TCustomActionMainMenuBar.SetupWindowMenu 682
ActnMenus.TCustomActionMainMenuBar.Shadows 684
ActnMenus.TCustomActionMainMenuBar.TrackMenu 682
ActnMenus.TCustomActionMainMenuBar.WindowMenu 684
ActnMenus.TCustomActionMainMenuBar.WMKeyDown 682
ActnMenus.TCustomActionMainMenuBar.WMNCHitTest 682
ActnMenus.TCustomActionMainMenuBar.WMSysCommand 683
ActnMenus.TCustomActionMainMenuBar.WMSysKeyDown 683
ActnMenus.TCustomActionMainMenuBar.WMSysKeyUp 683
ActnMenus.TCustomActionMenuBar 685
ActnMenus.TCustomActionMenuBar.Animate 701
ActnMenus.TCustomActionMenuBar.AnimateDuration 716
ActnMenus.TCustomActionMenuBar.AnimationStyle 716

ActnMenus.TCustomActionMenuBar.AutoFocus 717
ActnMenus.TCustomActionMenuBar.CleanupStack 701
ActnMenus.TCustomActionMenuBar.ClearSubMenus 702
ActnMenus.TCustomActionMenuBar.CloseMenu 702
ActnMenus.TCustomActionMenuBar.Create 698
ActnMenus.TCustomActionMenuBar.CreateControl 702
ActnMenus.TCustomActionMenuBar.CreatePopup 702
ActnMenus.TCustomActionMenuBar.DelayItem 717
ActnMenus.TCustomActionMenuBar.Destroy 698
ActnMenus.TCustomActionMenuBar.DoGetPopupClass 703
ActnMenus.TCustomActionMenuBar.DoItemClicked 703
ActnMenus.TCustomActionMenuBar.DoItemKeyed 703
ActnMenus.TCustomActionMenuBar.DoItemSelected 703
ActnMenus.TCustomActionMenuBar.DoMenuDelay 703
ActnMenus.TCustomActionMenuBar.DoMouseIdle 704
ActnMenus.TCustomActionMenuBar.DoneMenuLoop 704
ActnMenus.TCustomActionMenuBar.DoPopup 704
ActnMenus.TCustomActionMenuBar.ExecAction 704
ActnMenus.TCustomActionMenuBar.Expand 704
ActnMenus.TCustomActionMenuBar.Expandable 717
ActnMenus.TCustomActionMenuBar.ExpandDelay 717
ActnMenus.TCustomActionMenuBar.Expanded 718
ActnMenus.TCustomActionMenuBar.ExpandTimer 705
ActnMenus.TCustomActionMenuBar.FCachedMenu 699
ActnMenus.TCustomActionMenuBar.FChildMenu 699
ActnMenus.TCustomActionMenuBar.FExpandTimer 700
ActnMenus.TCustomActionMenuBar.FPopupTimer 700
ActnMenus.TCustomActionMenuBar.FSelectedItem 700
ActnMenus.TCustomActionMenuBar.GetDefaultSounds 705
ActnMenus.TCustomActionMenuBar.GetDesignMode 705
ActnMenus.TCustomActionMenuBar.GetMouseHoverItem 705
ActnMenus.TCustomActionMenuBar.GetPopupClass 706
ActnMenus.TCustomActionMenuBar.GetSelected 706
ActnMenus.TCustomActionMenuBar.Idle 706
ActnMenus.TCustomActionMenuBar.InitMenuLoop 706
ActnMenus.TCustomActionMenuBar.InMenuLoop 718
ActnMenus.TCustomActionMenuBar.IsDesignMsg 707
ActnMenus.TCustomActionMenuBar.Loaded 707
ActnMenus.TCustomActionMenuBar.MouseActivate 707
ActnMenus.TCustomActionMenuBar.MouseControl 718
ActnMenus.TCustomActionMenuBar.NewPopup 707
ActnMenus.TCustomActionMenuBar.Notification 708
ActnMenus.TCustomActionMenuBar.OnEnterMenuLoop 720
ActnMenus.TCustomActionMenuBar.OnExitMenuLoop 721
ActnMenus.TCustomActionMenuBar.OnGetPopupClass 721
ActnMenus.TCustomActionMenuBar.OnPopup 721
ActnMenus.TCustomActionMenuBar.ParentControl 718
ActnMenus.TCustomActionMenuBar.ParentMenu 719
ActnMenus.TCustomActionMenuBar.PopupStack 719
ActnMenus.TCustomActionMenuBar.ProcessMenuLoop 708
ActnMenus.TCustomActionMenuBar.ProcessMessage 708
ActnMenus.TCustomActionMenuBar.ProcessMessages 709
ActnMenus.TCustomActionMenuBar.ProcessMouseMsg 709
ActnMenus.TCustomActionMenuBar.RecreateControls 709
ActnMenus.TCustomActionMenuBar.RootMenu 719
ActnMenus.TCustomActionMenuBar.Select 709
ActnMenus.TCustomActionMenuBar.Selected 719
ActnMenus.TCustomActionMenuBar.SelectItem 710
ActnMenus.TCustomActionMenuBar.SetDesignMode 710
ActnMenus.TCustomActionMenuBar.SetParent 710
ActnMenus.TCustomActionMenuBar.SetPersistentHotKeys 711
ActnMenus.TCustomActionMenuBar.Style 711
ActnMenus.TCustomActionMenuBar.TrackMenu 711
ActnMenus.TCustomActionMenuBar.UseSystemFont 720
ActnMenus.TCustomActionMenuBar.WndProc 711
ActnMenus.TCustomActionPopupMenu 721
ActnMenus.TCustomActionPopupMenu.AddEdges 737
ActnMenus.TCustomActionPopupMenu.CanAutoSize 737
ActnMenus.TCustomActionPopupMenu.CloseMenu 737
ActnMenus.TCustomActionPopupMenu.CMMouseLeave 738
ActnMenus.TCustomActionPopupMenu.CMVisibleChanged 738
ActnMenus.TCustomActionPopupMenu.Create 735
ActnMenus.TCustomActionPopupMenu.CreateControl 738
ActnMenus.TCustomActionPopupMenu.CreateControls 738
ActnMenus.TCustomActionPopupMenu.CreateParams 739
ActnMenus.TCustomActionPopupMenu.DesignWndProc 739
ActnMenus.TCustomActionPopupMenu.Destroy 736
ActnMenus.TCustomActionPopupMenu.DisplayShadow 739

ActnMenus.TCustomActionPopupMenu.DrawBackground 740
ActnMenus.TCustomActionPopupMenu.ExecAction 740
ActnMenus.TCustomActionPopupMenu.Expand 740
ActnMenus.TCustomActionPopupMenu.ExpandBtn 744
ActnMenus.TCustomActionPopupMenu.ExpandClick 740
ActnMenus.TCustomActionPopupMenu.GetClientRect 741
ActnMenus.TCustomActionPopupMenu.GetExpandBtnClass 741
ActnMenus.TCustomActionPopupMenu.HideShadow 741
ActnMenus.TCustomActionPopupMenu.NewPopup 741
ActnMenus.TCustomActionPopupMenu.Notification 742
ActnMenus.TCustomActionPopupMenu.Popup 742
ActnMenus.TCustomActionPopupMenu.PositionPopup 742
ActnMenus.TCustomActionPopupMenu.SetColorMap 743
ActnMenus.TCustomActionPopupMenu.SetOrientation 743
ActnMenus.TCustomActionPopupMenu.SetupExpandBtn 743
ActnMenus.TCustomActionPopupMenu.Shadow 745
ActnMenus.TCustomActionPopupMenu.VisibleChanging 743
ActnMenus.TCustomActionPopupMenu.WMKeyDown 744
ActnMenus.TCustomActionPopupMenu.WMPrint 744
ActnMenus.TCustomActionPopupMenu.WMPrintClient 744
ActnMenus.TCustomAddRemoveItem 745
ActnMenus.TCustomAddRemoveItem.CalcBounds 754
ActnMenus.TCustomAddRemoveItem.IsActionVisible 755
ActnMenus.TCustomAddRemoveItem.MouseUp 755
ActnMenus.TCustomAddRemoveItem.SetEnabled 756
ActnMenus.TCustomAddRemoveItemClass 841
ActnMenus.TCustomAddRemoveItemClass type 841
ActnMenus.TCustomizeActionClientItem 756
ActnMenus.TCustomizeActionClientItem.GetActionLinkClass 759
ActnMenus.TCustomizeActionLink 759
ActnMenus.TCustomizeActionLink.SetEnabled 762
ActnMenus.TCustomizeActionToolBar 762
ActnMenus.TCustomizeActionToolBar.AdditionalItem 780
ActnMenus.TCustomizeActionToolBar.AddlItem 780
ActnMenus.TCustomizeActionToolBar.CanAutoSize 778
ActnMenus.TCustomizeActionToolBar.Clear 778
ActnMenus.TCustomizeActionToolBar.Create 777
ActnMenus.TCustomizeActionToolBar.Destroy 777
ActnMenus.TCustomizeActionToolBar.DoItemClicked 778
ActnMenus.TCustomizeActionToolBar.GetActionControl 779
ActnMenus.TCustomizeActionToolBar.GetAddRemoveItemClass 779
ActnMenus.TCustomizeActionToolBar.GetSubMenuControlClass 779
ActnMenus.TCustomizeActionToolBar.ItemCount 779
ActnMenus.TCustomizeActionToolBar.NewPopup 780
ActnMenus.TCustomizeActionToolBar.SetActionClient 780
ActnMenus.TCustomizeActionToolBarClass 842
ActnMenus.TCustomizeActionToolBarClass type 842
ActnMenus.TCustomMDIMenuButton 781
ActnMenus.TCustomMDIMenuButton.ButtonStyle 790
ActnMenus.TCustomMDIMenuButton.Paint 789
ActnMenus.TCustomMenuButton 790
ActnMenus.TCustomMenuButton.BeginAutoDrag 801
ActnMenus.TCustomMenuButton.Click 801
ActnMenus.TCustomMenuButton.CMMouseEnter 801
ActnMenus.TCustomMenuButton.Create 799
ActnMenus.TCustomMenuButton.Destroy 800
ActnMenus.TCustomMenuButton.DoDragDrop 802
ActnMenus.TCustomMenuButton.DragOver 802
ActnMenus.TCustomMenuButton.DrawText 802
ActnMenus.TCustomMenuButton.Keyed 803
ActnMenus.TCustomMenuButton.Menu 805
ActnMenus.TCustomMenuButton.MouseDown 803
ActnMenus.TCustomMenuButton.MouseUp 804
ActnMenus.TCustomMenuButton.Paint 804
ActnMenus.TCustomMenuExpandBtn 805
ActnMenus.TCustomMenuExpandBtn.Create 814
ActnMenus.TCustomMenuExpandBtn.DragOver 815
ActnMenus.TCustomMenuExpandBtn.DrawBackground 815
ActnMenus.TCustomMenuExpandBtn.DrawFrame 816
ActnMenus.TCustomMenuExpandBtn.Menu 816
ActnMenus.TCustomMenuExpandBtnClass 842
ActnMenus.TCustomMenuExpandBtnClass type 842
ActnMenus.TCustomMenuItem 816
ActnMenus.TCustomMenuItem.CalcBounds 827
ActnMenus.TCustomMenuItem.CalcLayout 827

ActnMenus.TCustomMenuItem.Click 827
ActnMenus.TCustomMenuItem.Create 825
ActnMenus.TCustomMenuItem.Destroy 825
ActnMenus.TCustomMenuItem.DoDragDrop 828
ActnMenus.TCustomMenuItem.DragDrop 828
ActnMenus.TCustomMenuItem.DragOver 828
ActnMenus.TCustomMenuItem.DrawEdge 829
ActnMenus.TCustomMenuItem.DrawShadowedText 829
ActnMenus.TCustomMenuItem.DrawSubMenuGlyph 829
ActnMenus.TCustomMenuItem.DrawText 829
ActnMenus.TCustomMenuItem.DrawUnusedEdges 830
ActnMenus.TCustomMenuItem.Edges 832
ActnMenus.TCustomMenuItem.FNoPrefix 826
ActnMenus.TCustomMenuItem.Keyed 830
ActnMenus.TCustomMenuItem.Menu 832
ActnMenus.TCustomMenuItem.MouseSelected 832
ActnMenus.TCustomMenuItem.MouseUp 830
ActnMenus.TCustomMenuItem.Paint 831
ActnMenus.TCustomMenuItem.PositionChanged 831
ActnMenus.TCustomMenuItem.SetSelected 831
ActnMenus.TCustomMenuItem.ShortCutBounds 832
ActnMenus.TCustomPopupClass 842
ActnMenus.TCustomPopupClass type 842
ActnMenus.TExitMenuEvent 843
ActnMenus.TExitMenuEvent type 843
ActnMenus.TGetPopupClassEvent 843
ActnMenus.TGetPopupClassEvent type 843
ActnMenus.TMDIButtonStyle 839
ActnMenus.TMDIButtonStyle enumeration 839
ActnMenus.TMenuButtonControlClass 843
ActnMenus.TMenuButtonControlClass type 843
ActnMenus.TMenuEdges 844
ActnMenus.TMenuEdges type 844
ActnMenus.TMenuItemControlClass 844
ActnMenus.TMenuItemControlClass type 844
ActnMenus.TMenuPopupEvent 844
ActnMenus.TMenuPopupEvent type 844
ActnMenus.TMenuStack 833
ActnMenus.TMenuStack.Bars 836
ActnMenus.TMenuStack.Create 834
ActnMenus.TMenuStack.Peek 834
ActnMenus.TMenuStack.Pop 835
ActnMenus.TMenuStack.Push 835
ActnMenus.TUpdateActnMenusProc 845
ActnMenus.TUpdateActnMenusProc type 845
ActnMenus.UnRegisterActnBarStyle 837
ActnMenus.UnRegisterActnBarStyle function 837
ActnMenus.UpdateActnMenusProc 848
ActnMenus.UpdateActnMenusProc variable 848
ActnPopup 849
ActnPopup namespace 849
    Classes 850
    Functions 859
ActnPopup.NewPopupMenu 859
ActnPopup.NewPopupMenu function 859
ActnPopup.TCustomActionPopupMenuEx 850
ActnPopup.TCustomActionPopupMenuEx.DoPopup 851
ActnPopup.TCustomActionPopupMenuEx.ExecAction 851
ActnPopup.TCustomActionPopupMenuEx.GetPopupClass 851
ActnPopup.TCustomActionPopupMenuEx.LoadMenu 851
ActnPopup.TPopupActionBar 852
ActnPopup.TPopupActionBar.Create 856
ActnPopup.TPopupActionBar.MenuActive 857
ActnPopup.TPopupActionBar.OnGetControlClass 858
ActnPopup.TPopupActionBar.Popup 856
ActnPopup.TPopupActionBar.PopupMenu 858
AdaptReq 861
AdaptReq namespace 861
    Classes 863
    Constants 972
    Functions 968
    Interfaces 920
    Structs, Records, Enums 969
    Types 971
AdaptReq.EAdapterRequestException 863
AdaptReq.ExtractStringValue 968
AdaptReq.ExtractStringValue function 968
AdaptReq.IActionFieldValue 920

AdaptReq.IActionFieldValue.FieldName 923
AdaptReq.IActionFieldValue.FileCount 924
AdaptReq.IActionFieldValue.Files 924
AdaptReq.IActionFieldValue.GetFieldName 922
AdaptReq.IActionFieldValue.GetFile 922
AdaptReq.IActionFieldValue.GetFileCount 922
AdaptReq.IActionFieldValue.GetValue 923
AdaptReq.IActionFieldValue.GetValueCount 923
AdaptReq.IActionFieldValue.ValueCount 925
AdaptReq.IActionFieldValue.Values 925
AdaptReq.IActionFieldValues 925
AdaptReq.IActionFieldValues.AddValue 927
AdaptReq.IActionFieldValues.FieldCount 930
AdaptReq.IActionFieldValues.FieldNames 931
AdaptReq.IActionFieldValues.GetFieldCount 927
AdaptReq.IActionFieldValues.GetFieldName 928
AdaptReq.IActionFieldValues.GetOrigValue 928
AdaptReq.IActionFieldValues.GetValue 928
AdaptReq.IActionFieldValues.IndexOfField 929
AdaptReq.IActionFieldValues.OrigValueOfField 929
AdaptReq.IActionFieldValues.OrigValues 931
AdaptReq.IActionFieldValues.ValueOfField 930
AdaptReq.IActionFieldValues.Values 932
AdaptReq.IActionFieldValuesOfAdapter 933
AdaptReq.IActionFieldValuesOfAdapter.LocateActionFieldValuesOfAdapter 934
AdaptReq.IActionHandleResponse 934
AdaptReq.IActionHandleResponse.HandleResponse 935
AdaptReq.IActionRequest 935
AdaptReq.IActionRequest.ActionName 938
AdaptReq.IActionRequest.GetActionName 937
AdaptReq.IActionRequest.GetHandled 937
AdaptReq.IActionRequest.Handled 938
AdaptReq.IActionRequest.SetHandled 937
AdaptReq.IActionRespondWith 938
AdaptReq.IActionRespondWith.GetRedirectOptions 939
AdaptReq.IActionRespondWith.GetRespondWith 939
AdaptReq.IActionRespondWith.RedirectOptions 941
AdaptReq.IActionRespondWith.RespondWith 941
AdaptReq.IActionRespondWith.RespondWithComponentPage 940
AdaptReq.IActionRespondWith.RespondWithPage 940
AdaptReq.IActionRespondWith.RespondWithURL 940
AdaptReq.IActionRespondWith.SetExecuteStatus 940
AdaptReq.IActionRespondWith.SetRedirectOptions 941
AdaptReq.IActionResponse 941
AdaptReq.IActionResponse.ExecuteStatus 944
AdaptReq.IActionResponse.GetExecuteStatus 942
AdaptReq.IActionResponse.GetHandled 943
AdaptReq.IActionResponse.Handled 944
AdaptReq.IActionResponse.SetExecuteStatus 943
AdaptReq.IActionResponse.SetHandled 943
AdaptReq.IAdapterModes 944
AdaptReq.IAdapterModes.AdapterCount 947
AdaptReq.IAdapterModes.AdapterModes 948
AdaptReq.IAdapterModes.AdapterNames 948
AdaptReq.IAdapterModes.FindAdapterMode 946
AdaptReq.IAdapterModes.GetAdapterCount 946
AdaptReq.IAdapterModes.GetAdapterMode 947
AdaptReq.IAdapterModes.GetAdapterName 947
AdaptReq.IAdapterRecordKeys 949
AdaptReq.IAdapterRecordKeys.GetRecordKeyCount 950
AdaptReq.IAdapterRecordKeys.GetRecordKeyName 950
AdaptReq.IAdapterRecordKeys.GetRecordKeyValue 951
AdaptReq.IAdapterRecordKeys.RecordKeyNames 951
AdaptReq.IAdapterRecordKeys.RecordKeyValues 952
AdaptReq.IAdapterRequestParams 952
AdaptReq.IAdapterRequestParams.AssignTo 954
AdaptReq.IAdapterRequestParams.GetParamCount 954
AdaptReq.IAdapterRequestParams.GetParamName 954
AdaptReq.IAdapterRequestParams.GetParamValue 955
AdaptReq.IAdapterRequestParams.IndexOfParam 955
AdaptReq.IAdapterRequestParams.ParamCount 956
AdaptReq.IAdapterRequestParams.ParamNames 956
AdaptReq.IAdapterRequestParams.ParamValues 957
AdaptReq.IAdapterRequestParams.ValueOfParam 955
AdaptReq.IEnumActionFieldValueRecords 957
AdaptReq.IEnumActionFieldValueRecords.EndIterator 959
AdaptReq.IEnumActionFieldValueRecords.Eof 962

AdaptReq.IEnumActionFieldValueRecords.FindPopulatedRecord 959
AdaptReq.IEnumActionFieldValueRecords.GetEOF 960
AdaptReq.IEnumActionFieldValueRecords.GetFirstRecord 960
AdaptReq.IEnumActionFieldValueRecords.GetInIterator 960
AdaptReq.IEnumActionFieldValueRecords.GetNextRecord 961
AdaptReq.IEnumActionFieldValueRecords.GetRecord 961
AdaptReq.IEnumActionFieldValueRecords.GetRecordCount 962
AdaptReq.IEnumActionFieldValueRecords.InIterator 962
AdaptReq.IEnumActionFieldValueRecords.RecordCount 963
AdaptReq.IGetAdapterActionRedirectOptions 963
AdaptReq.IGetAdapterActionRedirectOptions.FailureOptions 965
AdaptReq.IGetAdapterActionRedirectOptions.GetFailureOptions 964
AdaptReq.IGetAdapterActionRedirectOptions.GetSuccessOptions 964
AdaptReq.IGetAdapterActionRedirectOptions.SuccessOptions 965
AdaptReq.IImageRequest 965
AdaptReq.IImageRequest.GetImageName 966
AdaptReq.IImageRequest.ImageName 967
AdaptReq.IImageResponse 967
AdaptReq.roRedirect 972
AdaptReq.roRedirect constant 972
AdaptReq.sActionRequest 972
AdaptReq.sActionRequest constant 972
AdaptReq.sDelimAdapterMode 973
AdaptReq.sDelimAdapterMode constant 973
AdaptReq.sDelimOrigFieldValue 973
AdaptReq.sDelimOrigFieldValue constant 973
AdaptReq.sDelimRecordKeys 973
AdaptReq.sDelimRecordKeys constant 973
AdaptReq.TActionFieldValueImpl 864
AdaptReq.TActionFieldValueImpl.Create 865
AdaptReq.TActionFieldValueImpl.Destroy 866
AdaptReq.TActionFieldValueImpl.GetFieldName 866
AdaptReq.TActionFieldValueImpl.GetFile 867
AdaptReq.TActionFieldValueImpl.GetFileCount 867
AdaptReq.TActionFieldValueImpl.GetValue 867
AdaptReq.TActionFieldValueImpl.GetValueCount 868
AdaptReq.TActionFieldValuesImpl 868
AdaptReq.TActionFieldValuesImpl.AddValue 870
AdaptReq.TActionFieldValuesImpl.Create 869
AdaptReq.TActionFieldValuesImpl.Destroy 870
AdaptReq.TActionFieldValuesImpl.FieldCount 872
AdaptReq.TActionFieldValuesImpl.FieldNames 873
AdaptReq.TActionFieldValuesImpl.GetFieldName 871
AdaptReq.TActionFieldValuesImpl.GetRecordKeyCount 871
AdaptReq.TActionFieldValuesImpl.GetRecordKeyName 871
AdaptReq.TActionFieldValuesImpl.GetRecordKeyValue 871
AdaptReq.TActionFieldValuesImpl.IndexOfField 872
AdaptReq.TActionFieldValuesImpl.OrigValueOfField 872
AdaptReq.TActionFieldValuesImpl.OrigValues 873
AdaptReq.TActionFieldValuesImpl.ValueOfField 872
AdaptReq.TActionFieldValuesImpl.Values 873
AdaptReq.TAdapterActionFieldValuesList 873
AdaptReq.TAdapterActionFieldValuesList.Create 875
AdaptReq.TAdapterActionFieldValuesList.CurrentActionFieldValuesList 876
AdaptReq.TAdapterActionFieldValuesList.Destroy 875
AdaptReq.TAdapterActionFieldValuesList.LocateActionFieldValuesOfAdapter 876
AdaptReq.TAdapterActionRedirectOption 969
AdaptReq.TAdapterActionRedirectOption enumeration 969
AdaptReq.TAdapterActionRedirectOptions 971
AdaptReq.TAdapterActionRedirectOptions type 971
AdaptReq.TAdapterModes 876
AdaptReq.TAdapterModes.AdapterCount 880
AdaptReq.TAdapterModes.Create 878
AdaptReq.TAdapterModes.Destroy 878
AdaptReq.TAdapterModes.FindAdapterMode 879
AdaptReq.TAdapterModes.GetAdapterCount 879
AdaptReq.TAdapterModes.GetAdapterMode 879
AdaptReq.TAdapterModes.GetAdapterName 879
AdaptReq.TAdapterRequestParamsImpl 880
AdaptReq.TAdapterRequestParamsImpl.AssignTo 882
AdaptReq.TAdapterRequestParamsImpl.Create 881

AdaptReq.TAdapterRequestParamsImpl.GetParamCount 882
AdaptReq.TAdapterRequestParamsImpl.GetParamName 882
AdaptReq.TAdapterRequestParamsImpl.GetParamValue 882
AdaptReq.TAdapterRequestParamsImpl.IndexOfParam 883
AdaptReq.TAdapterRequestParamsImpl.ParamCount 883
AdaptReq.TAdapterRequestParamsImpl.ParamNames 883
AdaptReq.TAdapterRequestParamsImpl.ParamValues 884
AdaptReq.TAdapterRequestParamsImpl.ValueOfParam 883
AdaptReq.TBasicActionRequestImpl 884
AdaptReq.TBasicActionRequestImpl.ActionName 889
AdaptReq.TBasicActionRequestImpl.ActionParams 889
AdaptReq.TBasicActionRequestImpl.Create 886
AdaptReq.TBasicActionRequestImpl.Destroy 886
AdaptReq.TBasicActionRequestImpl.DispatchParams 889
AdaptReq.TBasicActionRequestImpl.FailurePage 889
AdaptReq.TBasicActionRequestImpl.GetActionName 887
AdaptReq.TBasicActionRequestImpl.GetFailurePage 887
AdaptReq.TBasicActionRequestImpl.GetHandled 888
AdaptReq.TBasicActionRequestImpl.GetSuccessPage 888
AdaptReq.TBasicActionRequestImpl.SetHandled 888
AdaptReq.TBasicActionRequestImpl.SuccessPage 890
AdaptReq.TBasicActionResponseImpl 890
AdaptReq.TBasicActionResponseImpl.Create 892
AdaptReq.TBasicActionResponseImpl.ExecuteStatus 896
AdaptReq.TBasicActionResponseImpl.FailurePage 896
AdaptReq.TBasicActionResponseImpl.GetExecuteStatus 892
AdaptReq.TBasicActionResponseImpl.GetHandled 893
AdaptReq.TBasicActionResponseImpl.GetRedirectOptions 893
AdaptReq.TBasicActionResponseImpl.GetRespondWith 893
AdaptReq.TBasicActionResponseImpl.Handled 896
AdaptReq.TBasicActionResponseImpl.HandleResponse 894
AdaptReq.TBasicActionResponseImpl.RedirectOptions 897
AdaptReq.TBasicActionResponseImpl.RespondWith 897
AdaptReq.TBasicActionResponseImpl.RespondWithComponentPage 894
AdaptReq.TBasicActionResponseImpl.RespondWithFailurePage 894
AdaptReq.TBasicActionResponseImpl.RespondWithPage 894
AdaptReq.TBasicActionResponseImpl.RespondWithSuccessPage 894
AdaptReq.TBasicActionResponseImpl.RespondWithURL 895
AdaptReq.TBasicActionResponseImpl.SetExecuteStatus 895
AdaptReq.TBasicActionResponseImpl.SetHandled 895
AdaptReq.TBasicActionResponseImpl.SetRedirectOptions 895
AdaptReq.TBasicActionResponseImpl.SuccessPage 897
AdaptReq.TBasicImageRequestImpl 898
AdaptReq.TBasicImageRequestImpl.ActionRequestParams 901
AdaptReq.TBasicImageRequestImpl.Create 900
AdaptReq.TBasicImageRequestImpl.Destroy 900
AdaptReq.TBasicImageRequestImpl.DispatchParams 901
AdaptReq.TBasicImageRequestImpl.GetImageName 901
AdaptReq.TBasicImageRequestImpl.ImageName 902
AdaptReq.TBasicImageResponseImpl 902
AdaptReq.TBasicImageResponseImpl.Create 903
AdaptReq.TEnumActionFieldValueRecordsImpl 904
AdaptReq.TEnumActionFieldValueRecordsImpl.AdapterName 908
AdaptReq.TEnumActionFieldValueRecordsImpl.Create 905
AdaptReq.TEnumActionFieldValueRecordsImpl.CurrentFieldValues 909
AdaptReq.TEnumActionFieldValueRecordsImpl.Destroy 905
AdaptReq.TEnumActionFieldValueRecordsImpl.EndIterator 906
AdaptReq.TEnumActionFieldValueRecordsImpl.Eof 909
AdaptReq.TEnumActionFieldValueRecordsImpl.FindPopulatedRecord 907
AdaptReq.TEnumActionFieldValueRecordsImpl.GetEOF 907
AdaptReq.TEnumActionFieldValueRecordsImpl.GetFirstRecord 907
AdaptReq.TEnumActionFieldValueRecordsImpl.GetInIterator 907
AdaptReq.TEnumActionFieldValueRecordsImpl.GetNextRecord 907
AdaptReq.TEnumActionFieldValueRecordsImpl.GetRecord 908
AdaptReq.TEnumActionFieldValueRecordsImpl.GetRecordCount 908
AdaptReq.TestRedirectOptions 968
AdaptReq.TestRedirectOptions function 968
AdaptReq.TExecuteStatus 969

AdaptReq.TExecuteStatus enumeration 969
AdaptReq.TMultiRecordActionRequestImpl 909
AdaptReq.TMultiRecordActionRequestImpl.ActionFieldValues 913
AdaptReq.TMultiRecordActionRequestImpl.ActionFieldValues2 914
AdaptReq.TMultiRecordActionRequestImpl.ActionFieldValuesList 914
AdaptReq.TMultiRecordActionRequestImpl.AdapterActionFieldValuesList 914
AdaptReq.TMultiRecordActionRequestImpl.AdapterModes 914
AdaptReq.TMultiRecordActionRequestImpl.Create 912
AdaptReq.TMultiRecordActionRequestImpl.Destroy 912
AdaptReq.TRespondWith 970
AdaptReq.TRespondWith enumeration 970
AdaptReq.TSingleRecordActionRequestImpl 915
AdaptReq.TSingleRecordActionRequestImpl.ActionFieldValues 918
AdaptReq.TSingleRecordActionRequestImpl.AdapterModes 919
AdaptReq.TSingleRecordActionRequestImpl.Create 917
AdaptReq.TSingleRecordActionRequestImpl.Destroy 918
adCantView enumeration member 24983
adLoginRequired enumeration member 24983
ADODB 975
ADODB namespace 975
Classes 978
Functions 1286
Structs, Records, Enums 1289
Types 1310
ADODB._Command 1310
ADODB._Command type 1310
ADODB._Connection 1311
ADODB._Connection type 1311
ADODB._Parameter 1311
ADODB._Parameter type 1311
ADODB._Recordset 1311
ADODB._Recordset type 1311
ADODB.CreateUDLFile 1286
ADODB.CreateUDLFile function 1286
ADODB.DataLinkDir 1286
ADODB.DataLinkDir function 1286
ADODB.EADOError 978
ADODB.Error 1312
ADODB.Error type 1312
ADODB.Errors 1312
ADODB.Errors type 1312
ADODB.GetDataLinkFiles 1287
ADODB.GetDataLinkFiles function 1287
ADODB.GetProviderNames 1287, 1288
ADODB.GetProviderNames function 1287, 1288
ADODB.Parameters 1312
ADODB.Parameters type 1312
ADODB.PromptDataLinkFile 1288
ADODB.PromptDataLinkFile function 1288
ADODB.PromptDataSource 1288
ADODB.PromptDataSource function 1288
ADODB.Properties 1313
ADODB.Properties type 1313
ADODB.Property_ 1313
ADODB.Property_ type 1313
ADODB.PVariantList 1313
ADODB.PVariantList type 1313
ADODB.TADOBlobStream 979
ADODB.TADOBlobStream.Create 981
ADODB.TADOBlobStream.Destroy 981
ADODB.TADOBlobStream.ReadBlobData 982
ADODB.TADOBlobStream.Realloc 982
ADODB.TADOBlobStream.Truncate 983
ADODB.TADOBlobStream.Write 983
ADODB.TADOCommand 984
ADODB.TADOCommand.ActiveConnection 995
ADODB.TADOCommand.Assign 990
ADODB.TADOCommand.AssignCommandText 990
ADODB.TADOCommand.Cancel 991
ADODB.TADOCommand.CheckCommandText 991
ADODB.TADOCommand.ClearActiveConnection 991
ADODB.TADOCommand.CommandObject 995
ADODB.TADOCommand.CommandText 995
ADODB.TADOCommand.CommandTextAlias 996

ADODB.TADOCommand.CommandTimeout 996
ADODB.TADOCommand.CommandType 997
ADODB.TADOCommand.ComponentLoading 992
ADODB.TADOCommand.ComponentRef 997
ADODB.TADOCommand.Connection 997
ADODB.TADOCommand.ConnectionStateChange 992
ADODB.TADOCommand.ConnectionString 998
ADODB.TADOCommand.Create 988
ADODB.TADOCommand.Destroy 989
ADODB.TADOCommand.Execute 992
ADODB.TADOCommand.ExecuteOptions 999
ADODB.TADOCommand.Initialize 993
ADODB.TADOCommand.OpenConnection 993
ADODB.TADOCommand.ParamCheck 999
ADODB.TADOCommand.Parameters 999
ADODB.TADOCommand.Prepared 1000
ADODB.TADOCommand.Properties 1000
ADODB.TADOCommand.SetCommandText 994
ADODB.TADOCommand.SetConnectionFlag 994
ADODB.TADOCommand.States 1001
ADODB.TADOConnection 1001
ADODB.TADOConnection.AfterConnect 1034
ADODB.TADOConnection.AfterDisconnect 1034
ADODB.TADOConnection.Attributes 1020
ADODB.TADOConnection.BeforeConnect 1034
ADODB.TADOConnection.BeforeDisconnect 1035
ADODB.TADOConnection.BeginTrans 1008
ADODB.TADOConnection.BeginTransComplete 1008
ADODB.TADOConnection.Cancel 1009
ADODB.TADOConnection.CheckActive 1009
ADODB.TADOConnection.CheckDisconnect 1009
ADODB.TADOConnection.CheckInactive 1010
ADODB.TADOConnection.CommandCount 1021
ADODB.TADOConnection.Commands 1021
ADODB.TADOConnection.CommandTimeout 1022
ADODB.TADOConnection.CommitTrans 1010
ADODB.TADOConnection.CommitTransComplete 1010
ADODB.TADOConnection.ConnectComplete 1010
ADODB.TADOConnection.Connected 1022
ADODB.TADOConnection.ConnectionObject 1023
ADODB.TADOConnection.ConnectionPoint 1011
ADODB.TADOConnection.ConnectionString 1024
ADODB.TADOConnection.ConnectionTimeout 1025
ADODB.TADOConnection.ConnectOptions 1026
ADODB.TADOConnection.Create 1006
ADODB.TADOConnection.CursorLocation 1026
ADODB.TADOConnection.DataSets 1027
ADODB.TADOConnection.DefaultDatabase 1027
ADODB.TADOConnection.Destroy 1006
ADODB.TADOConnection.Disconnect 1011
ADODB.TADOConnection.DoConnect 1011
ADODB.TADOConnection.DoDisconnect 1011
ADODB.TADOConnection.Errors 1028
ADODB.TADOConnection.Execute 1012
ADODB.TADOConnection.ExecuteComplete 1013
ADODB.TADOConnection.GetConnected 1013
ADODB.TADOConnection.GetErrors 1013
ADODB.TADOConnection.GetFieldNames 1014
ADODB.TADOConnection.GetProcedureNames 1014
ADODB.TADOConnection.GetTableNames 1015
ADODB.TADOConnection.InfoMessage 1016
ADODB.TADOConnection.InTransaction 1028
ADODB.TADOConnection.IsolationLevel 1029
ADODB.TADOConnection.KeepConnection 1029
ADODB.TADOConnection.Loaded 1016
ADODB.TADOConnection.LoginPrompt 1029
ADODB.TADOConnection.Mode 1031
ADODB.TADOConnection.OnBeginTransComplete 1035
ADODB.TADOConnection.OnCommitTransComplete 1036
ADODB.TADOConnection.OnConnectComplete 1036
ADODB.TADOConnection.OnDisconnect 1037
ADODB.TADOConnection.OnExecuteComplete 1038
ADODB.TADOConnection.OnInfoMessage 1038
ADODB.TADOConnection.OnLogin 1031
ADODB.TADOConnection.OnRollbackTransComplete 1039
ADODB.TADOConnection.OnWillConnect 1039
ADODB.TADOConnection.OnWillExecute 1040
ADODB.TADOConnection.Open 1016

ADODB.TADOConnection.OpenSchema 1017
ADODB.TADOConnection.Properties 1032
ADODB.TADOConnection.Provider 1032
ADODB.TADOConnection.RegisterClient 1017
ADODB.TADOConnection.RollbackTrans 1018
ADODB.TADOConnection.RollbackTransComplete 1018
ADODB.TADOConnection.State 1032
ADODB.TADOConnection.UnRegisterClient 1019
ADODB.TADOConnection.Version 1033
ADODB.TADOConnection.WaitForConnectComplete 1019
ADODB.TADOConnection.WillConnect 1019
ADODB.TADOConnection.WillExecute 1020
ADODB.TADODataSet 1041
ADODB.TADODataSet.AfterCancel 1067
ADODB.TADODataSet.AfterClose 1069
ADODB.TADODataSet.AfterDelete 1071
ADODB.TADODataSet.AfterEdit 1073
ADODB.TADODataSet.AfterInsert 1074
ADODB.TADODataSet.AfterOpen 1075
ADODB.TADODataSet.AfterPost 1076
ADODB.TADODataSet.AfterScroll 1076
ADODB.TADODataSet.BeforeCancel 1077
ADODB.TADODataSet.BeforeClose 1077
ADODB.TADODataSet.BeforeDelete 1078
ADODB.TADODataSet.BeforeEdit 1078
ADODB.TADODataSet.BeforeInsert 1079
ADODB.TADODataSet.BeforeOpen 1080
ADODB.TADODataSet.BeforePost 1081
ADODB.TADODataSet.BeforeScroll 1082
ADODB.TADODataSet.CommandText 1057
ADODB.TADODataSet.CommandTimeout 1057
ADODB.TADODataSet.CommandType 1057
ADODB.TADODataSet.CreateDataSet 1055
ADODB.TADODataSet.DataEvent 1055
ADODB.TADODataSet.DataSetField 1058
ADODB.TADODataSet.DataSource 1059
ADODB.TADODataSet.EnableBCD 1059
ADODB.TADODataSet.FieldDefs 1060
ADODB.TADODataSet.GetIndexNames 1055
ADODB.TADODataSet.IndexDefs 1062
ADODB.TADODataSet.IndexFieldNames 1063
ADODB.TADODataSet.IndexName 1063
ADODB.TADODataSet.MasterFields 1064
ADODB.TADODataSet.OnCalcFields 1082
ADODB.TADODataSet.OnDeleteError 1083
ADODB.TADODataSet.OnEditError 1083
ADODB.TADODataSet.OnNewRecord 1084
ADODB.TADODataSet.OnPostError 1084
ADODB.TADODataSet.OpenCursor 1056
ADODB.TADODataSet.ParamCheck 1065
ADODB.TADODataSet.Parameters 1065
ADODB.TADODataSet.Prepared 1066
ADODB.TADODataSet.RDSConnection 1066
ADODB.TADODataSet.SetConnection 1056
ADODB.TADODataSet.SetDataSetField 1056
ADODB.TADODataSet.StoreDefs 1066
ADODB.TADOLockType 1289
ADODB.TADOLockType enumeration 1289
ADODB.TADOQuery 1084
ADODB.TADOQuery.CommandTimeout 1101
ADODB.TADOQuery.Create 1099
ADODB.TADOQuery.DataSource 1101
ADODB.TADOQuery.Destroy 1099
ADODB.TADOQuery.EnableBCD 1103
ADODB.TADOQuery.ExecSQL 1100
ADODB.TADOQuery.ParamCheck 1103
ADODB.TADOQuery.Parameters 1104
ADODB.TADOQuery.Prepared 1104
ADODB.TADOQuery.QueryChanged 1100
ADODB.TADOQuery.RowsAffected 1105
ADODB.TADOQuery.SQL 1105
ADODB.TADOStoredProc 1106
ADODB.TADOStoredProc.CommandTimeout 1122
ADODB.TADOStoredProc.Create 1120
ADODB.TADOStoredProc.DataSource 1122
ADODB.TADOStoredProc.EnableBCD 1123
ADODB.TADOStoredProc.ExecProc 1121
ADODB.TADOStoredProc.Parameters 1123

ADODB.TADOStoredProc.Prepared 1124
ADODB.TADOStoredProc.ProcedureName 1124
ADODB.TADOTable 1125
ADODB.TADOTable.CommandTimeout 1140
ADODB.TADOTable.Create 1139
ADODB.TADOTable.EnableBCD 1141
ADODB.TADOTable.GetIndexNames 1140
ADODB.TADOTable.IndexDefs 1141
ADODB.TADOTable.IndexFieldNames 1142
ADODB.TADOTable.IndexName 1143
ADODB.TADOTable.MasterFields 1143
ADODB.TADOTable.MasterSource 1144
ADODB.TADOTable.ReadOnly 1144
ADODB.TADOTable.TableDirect 1145
ADODB.TADOTable.TableName 1145
ADODB.TAffectRecords 1290
ADODB.TAffectRecords enumeration 1290
ADODB.TBeginTransCompleteEvent 1314
ADODB.TBeginTransCompleteEvent type 1314
ADODB.TCommandType 1291
ADODB.TCommandType enumeration 1291
ADODB.TConnectErrorEvent 1314
ADODB.TConnectErrorEvent type 1314
ADODB.TConnectMode 1292
ADODB.TConnectMode enumeration 1292
ADODB.TConnectOption 1293
ADODB.TConnectOption enumeration 1293
ADODB.TCursorLocation 1293
ADODB.TCursorLocation enumeration 1293
ADODB.TCursorOption 1294
ADODB.TCursorOption enumeration 1294
ADODB.TCursorOptions 1314
ADODB.TCursorOptions type 1314
ADODB.TCursorType 1295
ADODB.TCursorType enumeration 1295
ADODB.TCustomADODataSet 1145
ADODB.TCustomADODataSet.ActivateTextFilter 1160
ADODB.TCustomADODataSet.Active 1201
ADODB.TCustomADODataSet.AfterCancel 1224
ADODB.TCustomADODataSet.AfterClose 1226
ADODB.TCustomADODataSet.AfterDelete 1227
ADODB.TCustomADODataSet.AfterEdit 1230
ADODB.TCustomADODataSet.AfterInsert 1231
ADODB.TCustomADODataSet.AfterOpen 1231
ADODB.TCustomADODataSet.AfterPost 1233
ADODB.TCustomADODataSet.AfterRefresh 1233
ADODB.TCustomADODataSet.AfterScroll 1233
ADODB.TCustomADODataSet.AllocRecordBuffer 1160
ADODB.TCustomADODataSet.AutoCalcFields 1203
ADODB.TCustomADODataSet.BeforeCancel 1234
ADODB.TCustomADODataSet.BeforeClose 1234
ADODB.TCustomADODataSet.BeforeDelete 1235
ADODB.TCustomADODataSet.BeforeEdit 1235
ADODB.TCustomADODataSet.BeforeInsert 1236
ADODB.TCustomADODataSet.BeforeOpen 1237
ADODB.TCustomADODataSet.BeforePost 1238
ADODB.TCustomADODataSet.BeforeRefresh 1239
ADODB.TCustomADODataSet.BeforeScroll 1239
ADODB.TCustomADODataSet.BookmarkValid 1160
ADODB.TCustomADODataSet.CacheSize 1203
ADODB.TCustomADODataSet.CancelBatch 1161
ADODB.TCustomADODataSet.CancelUpdates 1162
ADODB.TCustomADODataSet.CheckActive 1162
ADODB.TCustomADODataSet.CheckFieldCompatibility 1162
ADODB.TCustomADODataSet.ClearCalcFields 1162
ADODB.TCustomADODataSet.Clone 1163
ADODB.TCustomADODataSet.Command 1204
ADODB.TCustomADODataSet.CommandText 1204
ADODB.TCustomADODataSet.CommandTimeout 1204
ADODB.TCustomADODataSet.CommandType 1205
ADODB.TCustomADODataSet.CompareBookmarks 1163
ADODB.TCustomADODataSet.Connection 1205
ADODB.TCustomADODataSet.ConnectionPoint 1163
ADODB.TCustomADODataSet.ConnectionString 1206
ADODB.TCustomADODataSet.Create 1156
ADODB.TCustomADODataSet.CreateBlobStream 1164
ADODB.TCustomADODataSet.CursorLocation 1207
ADODB.TCustomADODataSet.CursorType 1208

ADODB.TCustomADODataSet.DataEvent 1164
ADODB.TCustomADODataSet.DataSource 1208
ADODB.TCustomADODataSet.DeactivateFilters 1164
ADODB.TCustomADODataSet.DefChanged 1165
ADODB.TCustomADODataSet.DefineProperties 1165
ADODB.TCustomADODataSet.DeleteRecords 1165
ADODB.TCustomADODataSet.DesignerData 1208
ADODB.TCustomADODataSet.Destroy 1157
ADODB.TCustomADODataSet.DestroyLookupCursor 1166
ADODB.TCustomADODataSet.DoOnNewRecord 1166
ADODB.TCustomADODataSet.EnableBCD 1209
ADODB.TCustomADODataSet.EnableEvents 1166
ADODB.TCustomADODataSet.EndOfRecordset 1167
ADODB.TCustomADODataSet.ExecuteOptions 1209
ADODB.TCustomADODataSet.FetchComplete 1167
ADODB.TCustomADODataSet.FetchProgress 1167
ADODB.TCustomADODataSet.FieldChangeComplete 1167
ADODB.TCustomADODataSet.FieldDefs 1210
ADODB.TCustomADODataSet.Filter 1212
ADODB.TCustomADODataSet.Filtered 1214
ADODB.TCustomADODataSet.FilterGroup 1214
ADODB.TCustomADODataSet.FilterOnBookmarks 1168
ADODB.TCustomADODataSet.FindRecord 1168
ADODB.TCustomADODataSet.FreeRecordBuffer 1168
ADODB.TCustomADODataSet.GetActiveRecBuf 1168
ADODB.TCustomADODataSet.GetBlobFieldData 1169
ADODB.TCustomADODataSet.GetBookmarkData 1169
ADODB.TCustomADODataSet.GetBookmarkFlag 1169
ADODB.TCustomADODataSet.GetCanModify 1169
ADODB.TCustomADODataSet.GetCommandText 1170
ADODB.TCustomADODataSet.GetDataSource 1170
ADODB.TCustomADODataSet.GetDetailLinkFields 1170
ADODB.TCustomADODataSet.GetFieldData 1171
ADODB.TCustomADODataSet.GetRecNo 1171
ADODB.TCustomADODataSet.GetRecord 1172
ADODB.TCustomADODataSet.GetRecordCount 1172
ADODB.TCustomADODataSet.GetRecordSize 1172
ADODB.TCustomADODataSet.GetStateFieldValue 1172
ADODB.TCustomADODataSet.IndexDefs 1215
ADODB.TCustomADODataSet.IndexFieldCount 1216
ADODB.TCustomADODataSet.IndexFieldNames 1216
ADODB.TCustomADODataSet.IndexFields 1217
ADODB.TCustomADODataSet.IndexName 1217
ADODB.TCustomADODataSet.InitOleDBAccess 1173
ADODB.TCustomADODataSet.InternalAddRecord 1173
ADODB.TCustomADODataSet.InternalCancel 1173
ADODB.TCustomADODataSet.InternalClose 1173
ADODB.TCustomADODataSet.InternalDelete 1173
ADODB.TCustomADODataSet.InternalEdit 1174
ADODB.TCustomADODataSet.InternalFirst 1174
ADODB.TCustomADODataSet.InternalGetRecord 1174
ADODB.TCustomADODataSet.InternalGotoBookmark 1174
ADODB.TCustomADODataSet.InternalHandleException 1175
ADODB.TCustomADODataSet.InternalInitFieldDefs 1175
ADODB.TCustomADODataSet.InternalInitRecord 1175
ADODB.TCustomADODataSet.InternalInsert 1175
ADODB.TCustomADODataSet.InternalLast 1175
ADODB.TCustomADODataSet.InternalOpen 1176
ADODB.TCustomADODataSet.InternalPost 1176
ADODB.TCustomADODataSet.InternalRefresh 1176
ADODB.TCustomADODataSet.InternalRequery 1176
ADODB.TCustomADODataSet.InternalSetSort 1177
ADODB.TCustomADODataSet.InternalSetToRecord 1177
ADODB.TCustomADODataSet.IsCursorOpen 1177
ADODB.TCustomADODataSet.IsSequenced 1177
ADODB.TCustomADODataSet.Loaded 1177
ADODB.TCustomADODataSet.LoadFromFile 1178
ADODB.TCustomADODataSet.Locate 1179
ADODB.TCustomADODataSet.LocateRecord 1179
ADODB.TCustomADODataSet.LockType 1218
ADODB.TCustomADODataSet.Lookup 1180
ADODB.TCustomADODataSet.MarshalOptions 1218
ADODB.TCustomADODataSet.MasterChanged 1180
ADODB.TCustomADODataSet.MasterDataLink 1218
ADODB.TCustomADODataSet.MasterDisabled 1180
ADODB.TCustomADODataSet.MasterFields 1219
ADODB.TCustomADODataSet.MaxRecords 1219
ADODB.TCustomADODataSet.MoveComplete 1180

ADODB.TCustomADODataSet.NextRecordset 1181
ADODB.TCustomADODataSet.OnCalcFields 1239
ADODB.TCustomADODataSet.OnDeleteError 1240
ADODB.TCustomADODataSet.OnEditError 1241
ADODB.TCustomADODataSet.OnEndOfRecordset 1241
ADODB.TCustomADODataSet.OnFetchComplete 1242
ADODB.TCustomADODataSet.OnFetchProgress 1242
ADODB.TCustomADODataSet.OnFieldChangeComplete 1243
ADODB.TCustomADODataSet.OnFilterRecord 1244
ADODB.TCustomADODataSet.OnMoveComplete 1244
ADODB.TCustomADODataSet.OnNewRecord 1245
ADODB.TCustomADODataSet.OnPostError 1245
ADODB.TCustomADODataSet.OnRecordChangeComplete 1246
ADODB.TCustomADODataSet.OnRecordsetChangeComplete 1247
ADODB.TCustomADODataSet.OnRecordsetCreate 1247
ADODB.TCustomADODataSet.OnWillChangeField 1248
ADODB.TCustomADODataSet.OnWillChangeRecord 1248
ADODB.TCustomADODataSet.OnWillChangeRecordset 1249
ADODB.TCustomADODataSet.OnWillMove 1250
ADODB.TCustomADODataSet.OpenCursor 1181
ADODB.TCustomADODataSet.ParamCheck 1220
ADODB.TCustomADODataSet.Parameters 1220
ADODB.TCustomADODataSet.PrepareCursor 1181
ADODB.TCustomADODataSet.Prepared 1221
ADODB.TCustomADODataSet.Properties 1221
ADODB.TCustomADODataSet.PSEndTransaction 1181
ADODB.TCustomADODataSet.PSExecute 1182
ADODB.TCustomADODataSet.PSExecuteStatement 1182
ADODB.TCustomADODataSet.PSGetAttributes 1183
ADODB.TCustomADODataSet.PSGetCommandText 1183
ADODB.TCustomADODataSet.PSGetCommandType 1184
ADODB.TCustomADODataSet.PSGetDefaultOrder 1184
ADODB.TCustomADODataSet.PSGetIndexDefs 1185
ADODB.TCustomADODataSet.PSGetKeyFieldsW 1185
ADODB.TCustomADODataSet.PSGetParams 1185
ADODB.TCustomADODataSet.PSGetQuoteCharW 1186
ADODB.TCustomADODataSet.PSGetTableNameW 1186
ADODB.TCustomADODataSet.PSGetUpdateException 1186
ADODB.TCustomADODataSet.PSInTransaction 1187
ADODB.TCustomADODataSet.PSIsSQLBased 1187
ADODB.TCustomADODataSet.PSIsSQLSupported 1188
ADODB.TCustomADODataSet.PSReset 1188
ADODB.TCustomADODataSet.PSSetCommandText 1188
ADODB.TCustomADODataSet.PSSetParams 1189
ADODB.TCustomADODataSet.PSStartTransaction 1189
ADODB.TCustomADODataSet.PSUpdateRecord 1190
ADODB.TCustomADODataSet.RecordChangeComplete 1190
ADODB.TCustomADODataSet.Recordset 1221
ADODB.TCustomADODataSet.RecordsetChangeComplete 1190
ADODB.TCustomADODataSet.RecordsetState 1222
ADODB.TCustomADODataSet.RecordStatus 1222
ADODB.TCustomADODataSet.RefreshParams 1191
ADODB.TCustomADODataSet.ReleaseLock 1191
ADODB.TCustomADODataSet.Requery 1191
ADODB.TCustomADODataSet.SaveToFile 1192
ADODB.TCustomADODataSet.Seek 1192
ADODB.TCustomADODataSet.SetBlockReadSize 1194
ADODB.TCustomADODataSet.SetBookmarkData 1194
ADODB.TCustomADODataSet.SetBookmarkFlag 1194
ADODB.TCustomADODataSet.SetCommandText 1194
ADODB.TCustomADODataSet.SetConnection 1194
ADODB.TCustomADODataSet.SetConnectionFlag 1195
ADODB.TCustomADODataSet.SetDataSource 1195
ADODB.TCustomADODataSet.SetDetailFilter 1195
ADODB.TCustomADODataSet.SetFieldData 1196
ADODB.TCustomADODataSet.SetFiltered 1196
ADODB.TCustomADODataSet.SetFilterOptions 1196
ADODB.TCustomADODataSet.SetFilterText 1196
ADODB.TCustomADODataSet.SetParamsFromCursor 1197
ADODB.TCustomADODataSet.SetRecNo 1197
ADODB.TCustomADODataSet.Sort 1222
ADODB.TCustomADODataSet.StoreDefs 1223
ADODB.TCustomADODataSet.Supports 1197
ADODB.TCustomADODataSet.UpdateBatch 1197
ADODB.TCustomADODataSet.UpdateIndexDefs 1198
ADODB.TCustomADODataSet.UpdateRecordSetPosition 1198
ADODB.TCustomADODataSet.UpdateStatus 1199

ADODB.TCustomADODataSet.WillChangeField 1199
ADODB.TCustomADODataSet.WillChangeRecord 1199
ADODB.TCustomADODataSet.WillChangeRecordset 1200
ADODB.TCustomADODataSet.WillMove 1200
ADODB.TDataType 1315
ADODB.TDataType type 1315
ADODB.TDisconnectEvent 1315
ADODB.TDisconnectEvent type 1315
ADODB.TEndOfRecordsetEvent 1315
ADODB.TEndOfRecordsetEvent type 1315
ADODB.TEventReason 1296
ADODB.TEventReason enumeration 1296
ADODB.TEventStatus 1298
ADODB.TEventStatus enumeration 1298
ADODB.TExecuteCompleteEvent 1316
ADODB.TExecuteCompleteEvent type 1316
ADODB.TExecuteOption 1299
ADODB.TExecuteOption enumeration 1299
ADODB.TExecuteOptions 1316
ADODB.TExecuteOptions type 1316
ADODB.TFetchProgressEvent 1317
ADODB.TFetchProgressEvent type 1317
ADODB.TFieldChangeCompleteEvent 1317
ADODB.TFieldChangeCompleteEvent type 1317
ADODB.TFilterGroup 1299
ADODB.TFilterGroup enumeration 1299
ADODB.TInfoMessageEvent 1317
ADODB.TInfoMessageEvent type 1317
ADODB.TIsolationLevel 1300
ADODB.TIsolationLevel enumeration 1300
ADODB.TMarshalOption 1301
ADODB.TMarshalOption enumeration 1301
ADODB.TObjectState 1302
ADODB.TObjectState enumeration 1302
ADODB.TObjectStates 1318
ADODB.TObjectStates type 1318
ADODB.TParameter 1250
ADODB.TParameter.AppendChunk 1253
ADODB.TParameter.Assign 1253
ADODB.TParameter.AssignTo 1254
ADODB.TParameter.Attributes 1257
ADODB.TParameter.DataType 1257
ADODB.TParameter.Direction 1257
ADODB.TParameter.GetDisplayName 1255
ADODB.TParameter.IsEqual 1255
ADODB.TParameter.LoadFromFile 1255
ADODB.TParameter.LoadFromStream 1255
ADODB.TParameter.Name 1258
ADODB.TParameter.NumericScale 1258
ADODB.TParameter.ParameterObject 1259
ADODB.TParameter.Parameters 1259
ADODB.TParameter.Precision 1259
ADODB.TParameter.Properties 1260
ADODB.TParameter.Size 1260
ADODB.TParameter.Value 1260
ADODB.TParameterAttribute 1302
ADODB.TParameterAttribute enumeration 1302
ADODB.TParameterAttributes 1318
ADODB.TParameterAttributes type 1318
ADODB.TParameterDirection 1303
ADODB.TParameterDirection enumeration 1303
ADODB.TParameters 1261
ADODB.TParameters.AddParameter 1264
ADODB.TParameters.AppendParameters 1264
ADODB.TParameters.AssignValues 1265
ADODB.TParameters.Command 1272
ADODB.TParameters.Create_Parameter 1265
ADODB.TParameters.CreateParameter 1266
ADODB.TParameters.FindParam 1266
ADODB.TParameters.GetAttr 1267
ADODB.TParameters.GetAttrCount 1267
ADODB.TParameters.GetItemAttr 1268
ADODB.TParameters.GetParamList 1268
ADODB.TParameters.InternalRefresh 1269
ADODB.TParameters.IsEqual 1269
ADODB.TParameters.Items 1272
ADODB.TParameters.Modified 1272
ADODB.TParameters.ParamByName 1269

ADODB.TParameters.ParameterCollection 1273
ADODB.TParameters.ParamValues 1273
ADODB.TParameters.ParseSQL 1270
ADODB.TParameters.Refresh 1271
ADODB.TParameters.Update 1271
ADODB.TPersistFormat 1303
ADODB.TPersistFormat enumeration 1303
ADODB.TPropList 1319
ADODB.TPropList type 1319
ADODB.TRDSConnection 1274
ADODB.TRDSConnection.AfterConnect 1284
ADODB.TRDSConnection.AfterDisconnect 1284
ADODB.TRDSConnection.AppServer 1281
ADODB.TRDSConnection.BeforeConnect 1284
ADODB.TRDSConnection.BeforeDisconnect 1285
ADODB.TRDSConnection.ComputerName 1281
ADODB.TRDSConnection.Connected 1282
ADODB.TRDSConnection.Create 1278
ADODB.TRDSConnection.DataSpaceObject 1282
ADODB.TRDSConnection.Destroy 1278
ADODB.TRDSConnection.DoConnect 1279
ADODB.TRDSConnection.DoDisconnect 1279
ADODB.TRDSConnection.GetConnected 1280
ADODB.TRDSConnection.GetRecordset 1280
ADODB.TRDSConnection.InternetTimeout 1283
ADODB.TRDSConnection.ServerName 1283
ADODB.TRecordChangeCompleteEvent 1319
ADODB.TRecordChangeCompleteEvent type 1319
ADODB.TRecordsetCreate 1320
ADODB.TRecordsetCreate type 1320
ADODB.TRecordsetErrorEvent 1320
ADODB.TRecordsetErrorEvent type 1320
ADODB.TRecordsetEvent 1320
ADODB.TRecordsetEvent type 1320
ADODB.TRecordsetReasonEvent 1321
ADODB.TRecordsetReasonEvent type 1321
ADODB.TRecordStatus 1304
ADODB.TRecordStatus enumeration 1304
ADODB.TRecordStatusSet 1321
ADODB.TRecordStatusSet type 1321
ADODB.TSchemaInfo 1305
ADODB.TSchemaInfo enumeration 1305
ADODB.TSeekOption 1308
ADODB.TSeekOption enumeration 1308
ADODB.TVariantList 1322
ADODB.TVariantList type 1322
ADODB.TWillChangeFieldEvent 1322
ADODB.TWillChangeFieldEvent type 1322
ADODB.TWillChangeRecordEvent 1323
ADODB.TWillChangeRecordEvent type 1323
ADODB.TWillConnectEvent 1323
ADODB.TWillConnectEvent type 1323
ADODB.TWillExecuteEvent 1323
ADODB.TWillExecuteEvent type 1323
ADODB.TXactAttribute 1308
ADODB.TXactAttribute enumeration 1308
ADODB.TXactAttributes 1324
ADODB.TXactAttributes type 1324
afAutoSize enumeration member 17387
afbNone enumeration member 14353
afbRaised enumeration member 14353
afbSingle enumeration member 14353
afbSunken enumeration member 14353
afCanCompare enumeration member 17387
afOwnsData enumeration member 17387
afSortUnique enumeration member 17387
aggSTATEBLANK enumeration member 11705
aggSTATEERROR enumeration member 11705
aggSTATEOK enumeration member 11705
aggSTATEUNDEF enumeration member 11705
akBottom enumeration member 6269
akLeft enumeration member 6269
akRight enumeration member 6269
akTop enumeration member 6269
alBottom enumeration member 6268
alClient enumeration member 6268
alCustom enumeration member 6268
alLeft enumeration member 6268

alNone enumeration member 6268
alRight enumeration member 6268
alTop enumeration member 6268
amBrowse enumeration member 8111
amEdit enumeration member 8111
amInsert enumeration member 8111
amQuery enumeration member 8111
AnsiString class 22093
- != 22123
- [] 22123
- [] operator 22123
- ~AnsiString 22096
- + 22124
- += 22124
- < 22124
- <= 22125
- = 22125
- == 22126
- > 22126
- >= 22126
- about AnsiString class 22093
- AnsiCompare 22103
- AnsiCompareIC 22104
- AnsiLastChar 22104
- AnsiPos 22104
- AnsiString 22097, 22098, 22099, 22100
- AnsiString __fastcall operator +(const char*, const AnsiString& rhs) 22127
- AnsiString constructor 22096
- AnsiString enumerations 22101
- AnsiString friends 22127
- AnsiString Members 22094
- AnsiString methods 22102
- AnsiString operators 22123
- AnsiString structures 22100
- ByteType 22105
- c_str 22105
- cat_printf 22106
- cat_sprintf 22106
- cat_vprintf 22107
- CurrToStr 22107
- CurrToStrF 22108
- data 22108
- Delete 22108
- FloatToStrF 22109
- FmtLoadStr 22110
- Format 22110
- FormatFloat 22110
- GetRec 22112
- GetRec method 22112
- Insert 22112
- IntToHex 22113
- IsDelimiter 22113
- IsEmpty 22113
- IsLeadByte 22114
- IsPathDelimiter 22114
- IsTrailByte 22114
- LastDelimiter 22115
- Length 22115
- LoadStr 22115
- LoadStr method 22115
- LoadString 22116
- LowerCase 22116
- Pos 22117
- printf 22117
- SetLength 22117
- sprintf 22118
- StringOfChar 22118
- SubString 22118
- ThrowIfOutOfRange 22119
- ToDouble 22119
- ToInt 22119
- ToIntDef 22120
- Trim 22120
- TrimLeft 22120
- TrimRight 22120
- Unique 22121
- UpperCase 22121
- vprintf 22121

WideChar 22122
WideCharBufSize 22122
AppEvnts 1325
AppEvnts namespace 1325
Classes 1326
AppEvnts.TApplicationEvents 1326
AppEvnts.TApplicationEvents.OnActionExecute 1330
AppEvnts.TApplicationEvents.OnActionUpdate 1331
AppEvnts.TApplicationEvents.OnActivate 1333
AppEvnts.TApplicationEvents.OnDeactivate 1334
AppEvnts.TApplicationEvents.OnException 1335
AppEvnts.TApplicationEvents.OnHelp 1336
AppEvnts.TApplicationEvents.OnHint 1337
AppEvnts.TApplicationEvents.OnIdle 1339
AppEvnts.TApplicationEvents.OnMessage 1340
AppEvnts.TApplicationEvents.OnMinimize 1341
AppEvnts.TApplicationEvents.OnModalBegin 1342
AppEvnts.TApplicationEvents.OnModalEnd 1342
AppEvnts.TApplicationEvents.OnRestore 1342
AppEvnts.TApplicationEvents.OnSettingChange 1343
AppEvnts.TApplicationEvents.OnShortCut 1344
AppEvnts.TApplicationEvents.OnShowHint 1345
AppEvnts.TCustomApplicationEvents 1347
AppEvnts.TCustomApplicationEvents.Activate 1351
AppEvnts.TCustomApplicationEvents.CancelDispatch 1351
AppEvnts.TCustomApplicationEvents.Create 1350
AppEvnts.TCustomApplicationEvents.OnActionExecute 1352
AppEvnts.TCustomApplicationEvents.OnActionUpdate 1354
AppEvnts.TCustomApplicationEvents.OnActivate 1355
AppEvnts.TCustomApplicationEvents.OnDeactivate 1356
AppEvnts.TCustomApplicationEvents.OnException 1357
AppEvnts.TCustomApplicationEvents.OnHelp 1358
AppEvnts.TCustomApplicationEvents.OnHint 1359
AppEvnts.TCustomApplicationEvents.OnIdle 1361
AppEvnts.TCustomApplicationEvents.OnMessage 1362
AppEvnts.TCustomApplicationEvents.OnMinimize 1363
AppEvnts.TCustomApplicationEvents.OnModalBegin 1364
AppEvnts.TCustomApplicationEvents.OnModalEnd 1364
AppEvnts.TCustomApplicationEvents.OnRestore 1364
AppEvnts.TCustomApplicationEvents.OnSettingChange 1365
AppEvnts.TCustomApplicationEvents.OnShortCut 1366
AppEvnts.TCustomApplicationEvents.OnShowHint 1367
arAlignBottom enumeration member 5455
arAlignLeft enumeration member 5455
arAlignRight enumeration member 5455
arAlignTop enumeration member 5455
arAll enumeration member 1290
arAllChapters enumeration member 1290
arAutoInc enumeration member 7604
arCurrent enumeration member 1290
arDefault enumeration member 5455, 7604
arFiltered enumeration member 1290
argIn enumeration member 25564
argInOut enumeration member 25564
argOut enumeration member 25564
argReturn enumeration member 25564
arNone enumeration member 7604
arSnapToGrid enumeration member 5455
asAccept enumeration member 18884
asClose enumeration member 18884
asConnect enumeration member 18884
asDefault enumeration member 839
asFade enumeration member 839
asNone enumeration member 839
asNormal enumeration member 337
asOOB enumeration member 18884
asRead enumeration member 18884
asSlide enumeration member 839
asSuspended enumeration member 337
asSuspendedEnabled enumeration member 337
asUnFold enumeration member 839
asWrite enumeration member 18884
atArrows enumeration member 14756
atDefaultText enumeration member 5440
atSelected enumeration member 5440
atSolid enumeration member 14756
AutoAdap 1371
AutoAdap namespace 1371

Classes 1373
Interfaces 1434
Types 1475
AutoAdap.IAdapterErrorsStub 1434
AutoAdap.IAdapterErrorsStub.DefineLabel 1435
AutoAdap.IAdapterErrorsStub.FieldErrors 1435
AutoAdap.IAdapterErrorStub 1435
AutoAdap.IAdapterErrorStub.Field 1437
AutoAdap.IAdapterErrorStub.Get_Field 1436
AutoAdap.IAdapterErrorStub.Get_ID 1436
AutoAdap.IAdapterErrorStub.Get_Message 1437
AutoAdap.IAdapterErrorStub.ID 1437
AutoAdap.IAdapterErrorStub.Message 1437
AutoAdap.IAdapterFieldStub 1438
AutoAdap.IAdapterFieldStub.CanModify 1444
AutoAdap.IAdapterFieldStub.CanView 1444
AutoAdap.IAdapterFieldStub.DisplayLabel 1445
AutoAdap.IAdapterFieldStub.DisplayStyle 1445
AutoAdap.IAdapterFieldStub.DisplayText 1445
AutoAdap.IAdapterFieldStub.DisplayWidth 1445
AutoAdap.IAdapterFieldStub.EditText 1445
AutoAdap.IAdapterFieldStub.Get_CanModify 1439
AutoAdap.IAdapterFieldStub.Get_CanView 1440
AutoAdap.IAdapterFieldStub.Get_DisplayLabel 1440
AutoAdap.IAdapterFieldStub.Get_DisplayStyle 1440
AutoAdap.IAdapterFieldStub.Get_DisplayText 1440
AutoAdap.IAdapterFieldStub.Get_DisplayWidth 1441
AutoAdap.IAdapterFieldStub.Get_EditText 1441
AutoAdap.IAdapterFieldStub.Get_Image 1441
AutoAdap.IAdapterFieldStub.Get_InputName 1441
AutoAdap.IAdapterFieldStub.Get_InputStyle 1441
AutoAdap.IAdapterFieldStub.Get_MaxLength 1442
AutoAdap.IAdapterFieldStub.Get_Name 1442
AutoAdap.IAdapterFieldStub.Get_Required 1442
AutoAdap.IAdapterFieldStub.Get_Value 1442
AutoAdap.IAdapterFieldStub.Get_Values 1443
AutoAdap.IAdapterFieldStub.Get_ValuesList 1443
AutoAdap.IAdapterFieldStub.Get_ViewMode 1443
AutoAdap.IAdapterFieldStub.Get_Visible 1443
AutoAdap.IAdapterFieldStub.Image 1446
AutoAdap.IAdapterFieldStub.InputName 1446
AutoAdap.IAdapterFieldStub.InputStyle 1446
AutoAdap.IAdapterFieldStub.IsEqual 1443
AutoAdap.IAdapterFieldStub.MaxLength 1446
AutoAdap.IAdapterFieldStub.Name 1447
AutoAdap.IAdapterFieldStub.Required 1447
AutoAdap.IAdapterFieldStub.Value 1447
AutoAdap.IAdapterFieldStub.Values 1447
AutoAdap.IAdapterFieldStub.ValuesList 1447
AutoAdap.IAdapterFieldStub.ViewMode 1448
AutoAdap.IAdapterFieldStub.Visible 1448
AutoAdap.IAdapterFieldValuesStub 1448
AutoAdap.IAdapterFieldValuesStub.Get_Value 1449
AutoAdap.IAdapterFieldValuesStub.HasValue 1449
AutoAdap.IAdapterFieldValuesStub.Value 1450
AutoAdap.IAdapterHiddenFieldsStub 1450
AutoAdap.IAdapterHiddenFieldsStub.WriteFields 1451
AutoAdap.IAdapterHiddenFieldStub 1451
AutoAdap.IAdapterHiddenFieldStub.Get_Name 1452
AutoAdap.IAdapterHiddenFieldStub.Get_Value 1452
AutoAdap.IAdapterHiddenFieldStub.Name 1452
AutoAdap.IAdapterHiddenFieldStub.Value 1453
AutoAdap.IAdapterImageStub 1453
AutoAdap.IAdapterImageStub.AsHREF 1454
AutoAdap.IAdapterImageStub.Get_AsHREF 1454
AutoAdap.IAdapterRecordStub 1454
AutoAdap.IAdapterStub 1455
AutoAdap.IAdapterStub.CanModify 1458
AutoAdap.IAdapterStub.CanView 1458
AutoAdap.IAdapterStub.Errors 1458
AutoAdap.IAdapterStub.Get_CanModify 1456
AutoAdap.IAdapterStub.Get_CanView 1456
AutoAdap.IAdapterStub.Get_Errors 1456
AutoAdap.IAdapterStub.Get_HiddenFields 1457
AutoAdap.IAdapterStub.Get_HiddenRecordFields 1457
AutoAdap.IAdapterStub.Get_Mode 1457
AutoAdap.IAdapterStub.HiddenFields 1458
AutoAdap.IAdapterStub.HiddenRecordFields 1459

AutoAdap.IAdapterStub.Mode 1459
AutoAdap.IAdapterStub.Set_Mode 1457
AutoAdap.IAdapterValuesListStub 1459
AutoAdap.IAdapterValuesListStub.Get_Image 1460
AutoAdap.IAdapterValuesListStub.Get_Value 1460
AutoAdap.IAdapterValuesListStub.Get_ValueName 1461
AutoAdap.IAdapterValuesListStub.Image 1461
AutoAdap.IAdapterValuesListStub.ImageOfValue 1461
AutoAdap.IAdapterValuesListStub.NameOfValue 1461
AutoAdap.IAdapterValuesListStub.Value 1462
AutoAdap.IAdapterValuesListStub.ValueName 1462
AutoAdap.IApplicationStub 1462
AutoAdap.IApplicationStub.Designing 1465
AutoAdap.IApplicationStub.Get_Designing 1463
AutoAdap.IApplicationStub.Get_ModuleFileName 1464
AutoAdap.IApplicationStub.Get_ModulePath 1464
AutoAdap.IApplicationStub.Get_Title 1464
AutoAdap.IApplicationStub.ModuleFileName 1465
AutoAdap.IApplicationStub.ModulePath 1465
AutoAdap.IApplicationStub.QualifyFileName 1464
AutoAdap.IApplicationStub.Title 1465
AutoAdap.IBaseAdapterStub 1466
AutoAdap.IBaseAdapterStub.Actions 1468
AutoAdap.IBaseAdapterStub.Fields 1468
AutoAdap.IBaseAdapterStub.Get_Actions 1467
AutoAdap.IBaseAdapterStub.Get_ClassName_ 1467
AutoAdap.IBaseAdapterStub.Get_Fields 1467
AutoAdap.IBaseAdapterStub.Get_Name_ 1467
AutoAdap.IBaseRecordsAdapterStub 1468
AutoAdap.IBaseRecordsAdapterStub.Get_Records 1469
AutoAdap.IBaseRecordsAdapterStub.Records 1469
AutoAdap.IEndUserStub 1470
AutoAdap.IEndUserStub.DisplayName 1472
AutoAdap.IEndUserStub.Get_DisplayName 1471
AutoAdap.IEndUserStub.Get_LoggedIn 1471
AutoAdap.IEndUserStub.Get_LoginFormAction 1471
AutoAdap.IEndUserStub.Get_LogoutAction 1472
AutoAdap.IEndUserStub.LoggedIn 1472
AutoAdap.IEndUserStub.LoginFormAction 1472
AutoAdap.IEndUserStub.LogoutAction 1473
AutoAdap.IImageProducerStub 1473
AutoAdap.IImageProducerStub.GetDisplayStyle 1474
AutoAdap.IImageProducerStub.GetEventImage 1474
AutoAdap.IImageProducerStub.GetImage 1474
AutoAdap.TActionLink 1373
AutoAdap.TActionLink.Create 1375
AutoAdap.TActionLink.GetAdapterItemRequestParams 1375
AutoAdap.TActionLink.GetComponent 1376
AutoAdap.TActionLink.GetFailurePage 1376
AutoAdap.TActionLink.GetSuccessPage 1376
AutoAdap.TAdapterActionArrayStub 1376
AutoAdap.TAdapterActionArrayStub.Action 1378
AutoAdap.TAdapterActionArrayStub.Create 1378
AutoAdap.TAdapterActionStub 1379
AutoAdap.TAdapterActionStub.CreateActionArrayStub 1381
AutoAdap.TAdapterActionStub.CreateActionStub 1381
AutoAdap.TAdapterActionStub.Get_Array_ 1381
AutoAdap.TAdapterActionStub.Get_AsFieldValue 1382
AutoAdap.TAdapterActionStub.Get_AsHREF 1382
AutoAdap.TAdapterActionStub.Get_CanExecute 1382
AutoAdap.TAdapterActionStub.Get_DisplayLabel 1382
AutoAdap.TAdapterActionStub.Get_DisplayStyle 1382
AutoAdap.TAdapterActionStub.Get_Enabled 1383
AutoAdap.TAdapterActionStub.Get_Visible 1383
AutoAdap.TAdapterActionStub.LinkToPage 1383
AutoAdap.TAdapterComponentsStub 1383
AutoAdap.TAdapterComponentsStub.FindObject 1384
AutoAdap.TAdapterErrorsStub 1384
AutoAdap.TAdapterErrorsStub.Create 1386
AutoAdap.TAdapterErrorsStub.CreateErrorsStub 1386
AutoAdap.TAdapterErrorsStub.DefineLabel 1387
AutoAdap.TAdapterErrorsStub.Errors 1387
AutoAdap.TAdapterErrorsStub.FieldErrors 1387
AutoAdap.TAdapterErrorStub 1387
AutoAdap.TAdapterErrorStub.Create 1389
AutoAdap.TAdapterErrorStub.Get_Field 1389
AutoAdap.TAdapterErrorStub.Get_ID 1390
AutoAdap.TAdapterErrorStub.Get_Message 1390

AutoAdap.TAdapterFieldStub 1390
AutoAdap.TAdapterFieldStub.CreateFieldValuesStub 1393
AutoAdap.TAdapterFieldStub.CreateImageStub 1393
AutoAdap.TAdapterFieldStub.CreateValuesListStub 1394
AutoAdap.TAdapterFieldStub.Get_CanModify 1394
AutoAdap.TAdapterFieldStub.Get_CanView 1394
AutoAdap.TAdapterFieldStub.Get_DisplayLabel 1394
AutoAdap.TAdapterFieldStub.Get_DisplayStyle 1394
AutoAdap.TAdapterFieldStub.Get_DisplayText 1395
AutoAdap.TAdapterFieldStub.Get_DisplayWidth 1395
AutoAdap.TAdapterFieldStub.Get_EditText 1395
AutoAdap.TAdapterFieldStub.Get_Image 1395
AutoAdap.TAdapterFieldStub.Get_InputName 1396
AutoAdap.TAdapterFieldStub.Get_InputStyle 1396
AutoAdap.TAdapterFieldStub.Get_MaxLength 1396
AutoAdap.TAdapterFieldStub.Get_Required 1396
AutoAdap.TAdapterFieldStub.Get_Value 1396
AutoAdap.TAdapterFieldStub.Get_Values 1397
AutoAdap.TAdapterFieldStub.Get_ValuesList 1397
AutoAdap.TAdapterFieldStub.Get_ViewMode 1397
AutoAdap.TAdapterFieldStub.Get_Visible 1397
AutoAdap.TAdapterFieldStub.IsEqual 1398
AutoAdap.TAdapterFieldValuesStub 1398
AutoAdap.TAdapterFieldValuesStub.FindObject 1399
AutoAdap.TAdapterFieldValuesStub.Get_Records 1399
AutoAdap.TAdapterFieldValuesStub.Get_Value 1400
AutoAdap.TAdapterFieldValuesStub.HasValue 1400
AutoAdap.TAdapterHiddenFieldsStub 1400
AutoAdap.TAdapterHiddenFieldsStub.Create 1402
AutoAdap.TAdapterHiddenFieldsStub.HiddenFields 1402
AutoAdap.TAdapterHiddenFieldsStub.WriteFields 1402
AutoAdap.TAdapterHiddenFieldStub 1403
AutoAdap.TAdapterHiddenFieldStub.Create 1404
AutoAdap.TAdapterHiddenFieldStub.Get_Name 1405
AutoAdap.TAdapterHiddenFieldStub.Get_Value 1405
AutoAdap.TAdapterImageStub 1405
AutoAdap.TAdapterImageStub.Create 1407
AutoAdap.TAdapterImageStub.Get_AsHREF 1407
AutoAdap.TAdapterRecordsStub 1407
AutoAdap.TAdapterRecordsStub.Adapter 1409
AutoAdap.TAdapterRecordsStub.Create 1409
AutoAdap.TAdapterRecordStub 1409
AutoAdap.TAdapterRecordStub.Create 1411
AutoAdap.TAdapterRecordStub.Impl 1411
AutoAdap.TAdapterStub 1412
AutoAdap.TAdapterStub.CreateErrorsStub 1413
AutoAdap.TAdapterStub.CreateHiddenFieldsStub 1414
AutoAdap.TAdapterStub.FindObject 1414
AutoAdap.TAdapterStub.Get_CanModify 1414
AutoAdap.TAdapterStub.Get_CanView 1414
AutoAdap.TAdapterStub.Get_Errors 1414
AutoAdap.TAdapterStub.Get_HiddenFields 1415
AutoAdap.TAdapterStub.Get_HiddenRecordFields 1415
AutoAdap.TAdapterStub.Get_Mode 1415
AutoAdap.TAdapterStub.Set_Mode 1415
AutoAdap.TAdapterStubClass 1475
AutoAdap.TAdapterStubClass type 1475
AutoAdap.TAdapterSubComponentStub 1416
AutoAdap.TAdapterSubComponentStub.Create 1417, 1418
AutoAdap.TAdapterSubComponentStub.Get_Name 1418
AutoAdap.TAdapterSubComponentStub.GetComponent 1418
AutoAdap.TAdapterSubComponentStub.ObjectIntf 1419
AutoAdap.TAdapterValuesListStub 1419
AutoAdap.TAdapterValuesListStub.CreateImageStub 1421
AutoAdap.TAdapterValuesListStub.FindObject 1421
AutoAdap.TAdapterValuesListStub.Get_Image 1421
AutoAdap.TAdapterValuesListStub.Get_Records 1421
AutoAdap.TAdapterValuesListStub.Get_Value 1421
AutoAdap.TAdapterValuesListStub.Get_ValueName 1422
AutoAdap.TAdapterValuesListStub.ImageOfValue 1422
AutoAdap.TAdapterValuesListStub.NameOfValue 1422
AutoAdap.TBaseAdapterStub 1422
AutoAdap.TBaseAdapterStub.CreateComponentsStub 1424
AutoAdap.TBaseAdapterStub.FActions 1423
AutoAdap.TBaseAdapterStub.FFields 1424
AutoAdap.TBaseAdapterStub.FindObject 1424
AutoAdap.TBaseAdapterStub.Get_Actions 1424
AutoAdap.TBaseAdapterStub.Get_ClassName_ 1425

AutoAdap.TBaseAdapterStub.Get_Fields 1425
AutoAdap.TBaseAdapterStub.Get_Name_ 1425
AutoAdap.TBaseRecordsAdapterStub 1425
AutoAdap.TBaseRecordsAdapterStub.CreateRecordsStub 1427
AutoAdap.TBaseRecordsAdapterStub.Get_Records 1427
AutoAdap.TWebApplicationStub 1427
AutoAdap.TWebApplicationStub.Create 1429
AutoAdap.TWebApplicationStub.Get_Designing 1429
AutoAdap.TWebApplicationStub.Get_ModuleFileName 1429
AutoAdap.TWebApplicationStub.Get_ModulePath 1430
AutoAdap.TWebApplicationStub.Get_Title 1430
AutoAdap.TWebApplicationStub.QualifyFileName 1430
AutoAdap.TWebEndUserStub 1430
AutoAdap.TWebEndUserStub.Create 1432
AutoAdap.TWebEndUserStub.CreateActionStub 1432
AutoAdap.TWebEndUserStub.Get_DisplayName 1433
AutoAdap.TWebEndUserStub.Get_LoggedIn 1433
AutoAdap.TWebEndUserStub.Get_LoginFormAction 1433
AutoAdap.TWebEndUserStub.Get_LogoutAction 1433
AutoCmd class 22127
~AutoCmd 22129
<< 22139, 22140, 22141, 22142, 22143, 22144
<< operator 22138
about AutoCmd class 22127
AddElement 22132
AutoCmd 22129
AutoCmd constructor 22129
AutoCmd data members 22130
AutoCmd Members 22128
AutoCmd methods 22131
AutoCmd operators 22138
Clear 22132
ClearArgs 22132
ClearName 22133
GetArgCount 22133
GetArgType 22134
GetCallType 22134
GetName 22134
GetNamedArgCount 22135
GetNamedArgType 22135
GetNamedParm 22136
GetNamedParmName 22136
GetParm 22137
Name 22130
NamedParmNames 22130
NamedParms 22130
NamedParmTypes 22131
Parms 22131
ParmTypes 22131
RequestResult 22137
SetName 22138
AutoDisp 1477
AutoDisp namespace 1477
Classes 1478
AutoDisp.TWebDispatch 1478
AutoDisp.TWebDispatch.AssignTo 1480
AutoDisp.TWebDispatch.Create 1480
AutoDisp.TWebDispatch.Destroy 1480
AutoDisp.TWebDispatch.Enabled 1482
AutoDisp.TWebDispatch.GetPathInfo 1481
AutoDisp.TWebDispatch.Mask 1482
AutoDisp.TWebDispatch.MethodType 1483
AutoDisp.TWebDispatch.PathInfo 1483
AutoDisp.TWebDispatch.SetPathInfo 1481
AutoDisp.TWebPathInfo 1483
AutoDisp.TWebPathInfo.Create 1484
AutoDisp.TWebPathInfo.Destroy 1484
AutoDisp.TWebPathInfo.Mask 1485
AutoDisp.TWebPathInfo.PathInfo 1485
aviCopyFile enumeration member 5443
aviCopyFiles enumeration member 5443
aviDeleteFile enumeration member 5443
aviEmptyRecycle enumeration member 5443
aviFindComputer enumeration member 5443
aviFindFile enumeration member 5443
aviFindFolder enumeration member 5443
aviNone enumeration member 5443
aviRecycleFile enumeration member 5443

AxCtrls 1487
AxCtrls namespace 1487
Classes 1489
Constants 1717
Functions 1709
Interfaces 1702
Structs, Records, Enums 1714
Types 1715
AxCtrls.Class_DColorPropPage 1717
AxCtrls.Class_DColorPropPage constant 1717
AxCtrls.Class_DFontPropPage 1717
AxCtrls.Class_DFontPropPage constant 1717
AxCtrls.Class_DPicturePropPage 1718
AxCtrls.Class_DPicturePropPage constant 1718
AxCtrls.Class_DStringPropPage 1718
AxCtrls.Class_DStringPropPage constant 1718
AxCtrls.EnumDispatchProperties 1709
AxCtrls.EnumDispatchProperties function 1709
AxCtrls.GetOleFont 1710
AxCtrls.GetOleFont function 1710
AxCtrls.GetOlePicture 1710
AxCtrls.GetOlePicture function 1710
AxCtrls.GetOleStrings 1711
AxCtrls.GetOleStrings function 1711
AxCtrls.IAmbientDispatch 1702
AxCtrls.IAmbientDispatch.AutoClip 1703
AxCtrls.IAmbientDispatch.BackColor 1703
AxCtrls.IAmbientDispatch.DisplayAsDefault 1703
AxCtrls.IAmbientDispatch.DisplayName 1704
AxCtrls.IAmbientDispatch.Font 1704
AxCtrls.IAmbientDispatch.ForeColor 1704
AxCtrls.IAmbientDispatch.LocaleID 1704
AxCtrls.IAmbientDispatch.MessageReflect 1705
AxCtrls.IAmbientDispatch.ScaleUnits 1705
AxCtrls.IAmbientDispatch.ShowGrabHandles 1705
AxCtrls.IAmbientDispatch.ShowHatching 1705
AxCtrls.IAmbientDispatch.SupportsMnemonics 1705
AxCtrls.IAmbientDispatch.TextAlign 1706
AxCtrls.IAmbientDispatch.UIDead 1706
AxCtrls.IAmbientDispatch.UserMode 1706
AxCtrls.IFontAccess 1706
AxCtrls.IFontAccess.GetOleFont 1707
AxCtrls.IFontAccess.SetOleFont 1707
AxCtrls.IPictureAccess 1707
AxCtrls.IPictureAccess.GetOlePicture 1708
AxCtrls.IPictureAccess.SetOlePicture 1708
AxCtrls.ParkingWindow 1711
AxCtrls.ParkingWindow function 1711
AxCtrls.SetOleFont 1712
AxCtrls.SetOleFont function 1712
AxCtrls.SetOlePicture 1712
AxCtrls.SetOlePicture function 1712
AxCtrls.SetOleStrings 1713
AxCtrls.SetOleStrings function 1713
AxCtrls.TActiveForm 1489
AxCtrls.TActiveForm.ActiveFormControl 1506
AxCtrls.TActiveForm.DefinePropertyPages 1505
AxCtrls.TActiveForm.DoDestroy 1505
AxCtrls.TActiveForm.EventSinkChanged 1506
AxCtrls.TActiveForm.Initialize 1506
AxCtrls.TActiveFormClass 1715
AxCtrls.TActiveFormClass type 1715
AxCtrls.TActiveFormControl 1507
AxCtrls.TActiveFormControl.DefinePropertyPages 1512
AxCtrls.TActiveFormControl.EventSinkChanged 1512
AxCtrls.TActiveFormControl.FreeOnRelease 1512
AxCtrls.TActiveFormControl.InitializeControl 1513
AxCtrls.TActiveFormControl.Invoke 1513
AxCtrls.TActiveFormControl.ObjQueryInterface 1513
AxCtrls.TActiveFormFactory 1514
AxCtrls.TActiveFormFactory.GetIntfEntry 1516
AxCtrls.TActiveXControl 1517
AxCtrls.TActiveXControl.Advise 1525
AxCtrls.TActiveXControl.ClientSite 1553
AxCtrls.TActiveXControl.Close 1525
AxCtrls.TActiveXControl.ConnectionPoints 1553
AxCtrls.TActiveXControl.ContextSensitiveHelp 1525
AxCtrls.TActiveXControl.Control 1553

AxCtrls.TActiveXControl.DAdvise 1525
AxCtrls.TActiveXControl.DefinePropertyPages 1526
AxCtrls.TActiveXControl.Destroy 1522
AxCtrls.TActiveXControl.DoVerb 1526
AxCtrls.TActiveXControl.Draw 1526
AxCtrls.TActiveXControl.DUnadvise 1526
AxCtrls.TActiveXControl.EnableModeless 1527
AxCtrls.TActiveXControl.EnumAdvise 1527
AxCtrls.TActiveXControl.EnumDAdvise 1527
AxCtrls.TActiveXControl.EnumFormatEtc 1527
AxCtrls.TActiveXControl.EnumVerbs 1527
AxCtrls.TActiveXControl.Freeze 1528
AxCtrls.TActiveXControl.FreezeEvents 1528
AxCtrls.TActiveXControl.Get_BackColor 1528
AxCtrls.TActiveXControl.Get_Caption 1528
AxCtrls.TActiveXControl.Get_Enabled 1529
AxCtrls.TActiveXControl.Get_Font 1529
AxCtrls.TActiveXControl.Get_ForeColor 1529
AxCtrls.TActiveXControl.Get_HWnd 1529
AxCtrls.TActiveXControl.Get_TabStop 1529
AxCtrls.TActiveXControl.Get_Text 1530
AxCtrls.TActiveXControl.GetAdvise 1530
AxCtrls.TActiveXControl.GetCanonicalFormatEtc 1530
AxCtrls.TActiveXControl.GetClassID 1530
AxCtrls.TActiveXControl.GetClientSite 1531
AxCtrls.TActiveXControl.GetClipboardData 1531
AxCtrls.TActiveXControl.GetColorSet 1531
AxCtrls.TActiveXControl.GetContentExtent 1531
AxCtrls.TActiveXControl.GetControlInfo 1531
AxCtrls.TActiveXControl.GetData 1532
AxCtrls.TActiveXControl.GetDataHere 1532
AxCtrls.TActiveXControl.GetDisplayString 1532
AxCtrls.TActiveXControl.GetExtent 1532
AxCtrls.TActiveXControl.GetInterfaceSafetyOptions 1533
AxCtrls.TActiveXControl.GetMiscStatus 1533
AxCtrls.TActiveXControl.GetMoniker 1533
AxCtrls.TActiveXControl.GetPages 1533
AxCtrls.TActiveXControl.GetPredefinedStrings 1533
AxCtrls.TActiveXControl.GetPredefinedValue 1534
AxCtrls.TActiveXControl.GetPropertyString 1534
AxCtrls.TActiveXControl.GetPropertyStrings 1534
AxCtrls.TActiveXControl.GetPropertyValue 1534
AxCtrls.TActiveXControl.GetPropFromBag 1535
AxCtrls.TActiveXControl.GetSizeMax 1535
AxCtrls.TActiveXControl.GetUserClassID 1535
AxCtrls.TActiveXControl.GetUserType 1535
AxCtrls.TActiveXControl.GetWindow 1536
AxCtrls.TActiveXControl.HandsOffStorage 1536
AxCtrls.TActiveXControl.InitFromData 1536
AxCtrls.TActiveXControl.Initialize 1536
AxCtrls.TActiveXControl.InitializeControl 1537
AxCtrls.TActiveXControl.InitNew 1537
AxCtrls.TActiveXControl.InPlaceActivate 1537
AxCtrls.TActiveXControl.InPlaceDeactivate 1537
AxCtrls.TActiveXControl.InPlaceSite 1554
AxCtrls.TActiveXControl.IsDirty 1537
AxCtrls.TActiveXControl.IsUpToDate 1538
AxCtrls.TActiveXControl.LoadFromStream 1538
AxCtrls.TActiveXControl.MapPropertyToPage 1538
AxCtrls.TActiveXControl.ObjQueryInterface 1538
AxCtrls.TActiveXControl.OnAmbientPropertyChange 1539
AxCtrls.TActiveXControl.OnDocWindowActivate 1539
AxCtrls.TActiveXControl.OnFrameWindowActivate 1539
AxCtrls.TActiveXControl.OnMnemonic 1539
AxCtrls.TActiveXControl.PerformVerb 1540
AxCtrls.TActiveXControl.PersistPropBagInitNew 1540
AxCtrls.TActiveXControl.PersistPropBagLoad 1540
AxCtrls.TActiveXControl.PersistPropBagSave 1540
AxCtrls.TActiveXControl.PersistStorageInitNew 1541
AxCtrls.TActiveXControl.PersistStorageLoad 1541
AxCtrls.TActiveXControl.PersistStorageSave 1541
AxCtrls.TActiveXControl.PersistStreamLoad 1541
AxCtrls.TActiveXControl.PersistStreamSave 1541
AxCtrls.TActiveXControl.PostMessageFilter 1542
AxCtrls.TActiveXControl.PreMessageFilter 1542
AxCtrls.TActiveXControl.PropChanged 1542
AxCtrls.TActiveXControl.PropRequestEdit 1543
AxCtrls.TActiveXControl.PutPropInBag 1543

AxCtrls.TActiveXControl.QueryGetData 1544
AxCtrls.TActiveXControl.QuickActivate 1544
AxCtrls.TActiveXControl.ReactivateAndUndo 1544
AxCtrls.TActiveXControl.ResizeBorder 1544
AxCtrls.TActiveXControl.SaveCompleted 1544
AxCtrls.TActiveXControl.SaveToStream 1545
AxCtrls.TActiveXControl.Set_BackColor 1545
AxCtrls.TActiveXControl.Set_Caption 1545
AxCtrls.TActiveXControl.Set_Enabled 1545
AxCtrls.TActiveXControl.Set_Font 1546
AxCtrls.TActiveXControl.Set_ForeColor 1546
AxCtrls.TActiveXControl.Set_TabStop 1546
AxCtrls.TActiveXControl.Set_Text 1546
AxCtrls.TActiveXControl.SetAdvise 1546
AxCtrls.TActiveXControl.SetClientSite 1547
AxCtrls.TActiveXControl.SetColorScheme 1547
AxCtrls.TActiveXControl.SetContentExtent 1547
AxCtrls.TActiveXControl.SetData 1547
AxCtrls.TActiveXControl.SetExtent 1548
AxCtrls.TActiveXControl.SetHostNames 1548
AxCtrls.TActiveXControl.SetInPlaceSite 1548
AxCtrls.TActiveXControl.SetInterfaceSafetyOptions 1548
AxCtrls.TActiveXControl.SetMoniker 1548
AxCtrls.TActiveXControl.SetObjectRects 1549
AxCtrls.TActiveXControl.ShowPropertyDialog 1549
AxCtrls.TActiveXControl.StdClickEvent 1549
AxCtrls.TActiveXControl.StdDblClickEvent 1549
AxCtrls.TActiveXControl.StdKeyDownEvent 1550
AxCtrls.TActiveXControl.StdKeyPressEvent 1550
AxCtrls.TActiveXControl.StdKeyUpEvent 1550
AxCtrls.TActiveXControl.StdMouseDownEvent 1550
AxCtrls.TActiveXControl.StdMouseMoveEvent 1550
AxCtrls.TActiveXControl.StdMouseUpEvent 1551
AxCtrls.TActiveXControl.TranslateAccelerator 1551
AxCtrls.TActiveXControl.UIDeactivate 1551
AxCtrls.TActiveXControl.Unadvise 1551
AxCtrls.TActiveXControl.Unfreeze 1552
AxCtrls.TActiveXControl.Update 1552
AxCtrls.TActiveXControl.ViewObjectGetExtent 1552
AxCtrls.TActiveXControl.WndProc 1552
AxCtrls.TActiveXControlClass 1715
AxCtrls.TActiveXControlClass type 1715
AxCtrls.TActiveXControlFactory 1554
AxCtrls.TActiveXControlFactory.AddVerb 1557
AxCtrls.TActiveXControlFactory.Create 1556
AxCtrls.TActiveXControlFactory.Destroy 1557
AxCtrls.TActiveXControlFactory.GetLicenseFileName 1558
AxCtrls.TActiveXControlFactory.HasMachineLicense 1558
AxCtrls.TActiveXControlFactory.MiscStatus 1559
AxCtrls.TActiveXControlFactory.ToolboxBitmapID 1560
AxCtrls.TActiveXControlFactory.UpdateRegistry 1559
AxCtrls.TActiveXControlFactory.WinControlClass 1560
AxCtrls.TActiveXPropertyPage 1561
AxCtrls.TActiveXPropertyPage.Destroy 1563
AxCtrls.TActiveXPropertyPage.Initialize 1563
AxCtrls.TActiveXPropertyPage.PropertyPageImpl 1564
AxCtrls.TActiveXPropertyPageFactory 1564
AxCtrls.TActiveXPropertyPageFactory.Create 1566
AxCtrls.TActiveXPropertyPageFactory.CreateComObject 1567
AxCtrls.TAdapterNotifier 1567
AxCtrls.TAdapterNotifier.Create 1569
AxCtrls.TAdapterNotifier.OnChanged 1569
AxCtrls.TAdapterNotifier.OnRequestEdit 1570
AxCtrls.TConnectionKind 1714
AxCtrls.TConnectionKind enumeration 1714
AxCtrls.TConnectionPoint 1570
AxCtrls.TConnectionPoint.Advise 1573
AxCtrls.TConnectionPoint.Create 1572
AxCtrls.TConnectionPoint.Destroy 1572
AxCtrls.TConnectionPoint.EnumConnections 1573
AxCtrls.TConnectionPoint.GetConnectionInterface 1573
AxCtrls.TConnectionPoint.GetConnectionPointContainer 1574
AxCtrls.TConnectionPoint.SinkList 1574
AxCtrls.TConnectionPoint.Unadvise 1574
AxCtrls.TConnectionPoints 1575
AxCtrls.TConnectionPoints.Controller 1578
AxCtrls.TConnectionPoints.Create 1576
AxCtrls.TConnectionPoints.CreateConnectionPoint 1577

AxCtrls.TConnectionPoints.Destroy 1576
AxCtrls.TConnectionPoints.EnumConnectionPoints 1577
AxCtrls.TConnectionPoints.FindConnectionPoint 1578
AxCtrls.TCustomAdapter 1578
AxCtrls.TCustomAdapter.Changed 1582
AxCtrls.TCustomAdapter.ConnectOleObject 1582
AxCtrls.TCustomAdapter.Create 1580
AxCtrls.TCustomAdapter.Destroy 1581
AxCtrls.TCustomAdapter.ReleaseOleObject 1582
AxCtrls.TCustomAdapter.Update 1583
AxCtrls.TCustomAdapter.Updating 1581
AxCtrls.TDefinePropertyPage 1716
AxCtrls.TDefinePropertyPage type 1716
AxCtrls.TFontAdapter 1583
AxCtrls.TFontAdapter.Changed 1585
AxCtrls.TFontAdapter.Create 1585
AxCtrls.TFontAdapter.GetOleFont 1586
AxCtrls.TFontAdapter.SetOleFont 1586
AxCtrls.TFontAdapter.Update 1587
AxCtrls.TOleGraphic 1587
AxCtrls.TOleGraphic.Assign 1590
AxCtrls.TOleGraphic.Changed 1590
AxCtrls.TOleGraphic.Draw 1590
AxCtrls.TOleGraphic.GetEmpty 1591
AxCtrls.TOleGraphic.GetHeight 1591
AxCtrls.TOleGraphic.GetPalette 1591
AxCtrls.TOleGraphic.GetTransparent 1591
AxCtrls.TOleGraphic.GetWidth 1591
AxCtrls.TOleGraphic.LoadFromClipboardFormat 1592
AxCtrls.TOleGraphic.LoadFromFile 1592
AxCtrls.TOleGraphic.LoadFromStream 1592
AxCtrls.TOleGraphic.MMHeight 1594
AxCtrls.TOleGraphic.MMWidth 1595
AxCtrls.TOleGraphic.Picture 1595
AxCtrls.TOleGraphic.SaveToClipboardFormat 1593
AxCtrls.TOleGraphic.SaveToStream 1593
AxCtrls.TOleGraphic.SetHeight 1593
AxCtrls.TOleGraphic.SetPalette 1594
AxCtrls.TOleGraphic.SetWidth 1594
AxCtrls.TOleStream 1595
AxCtrls.TOleStream.Create 1597
AxCtrls.TOleStream.GetIStream 1598
AxCtrls.TOleStream.Read 1598
AxCtrls.TOleStream.Seek 1598
AxCtrls.TOleStream.Write 1599
AxCtrls.TPictureAdapter 1599
AxCtrls.TPictureAdapter.Create 1601
AxCtrls.TPictureAdapter.GetOlePicture 1602
AxCtrls.TPictureAdapter.SetOlePicture 1602
AxCtrls.TPictureAdapter.Update 1602
AxCtrls.TPropertyPage 1603
AxCtrls.TPropertyPage.ActiveControl 1620
AxCtrls.TPropertyPage.AutoScroll 1621
AxCtrls.TPropertyPage.Caption 1621
AxCtrls.TPropertyPage.ClientHeight 1624
AxCtrls.TPropertyPage.ClientWidth 1624
AxCtrls.TPropertyPage.Color 1626
AxCtrls.TPropertyPage.Create 1616
AxCtrls.TPropertyPage.Ctl3D 1627
AxCtrls.TPropertyPage.Destroy 1617
AxCtrls.TPropertyPage.Enabled 1628
AxCtrls.TPropertyPage.EnumCtlProps 1617
AxCtrls.TPropertyPage.Font 1628
AxCtrls.TPropertyPage.Height 1629
AxCtrls.TPropertyPage.HorzScrollBar 1630
AxCtrls.TPropertyPage.KeyPreview 1631
AxCtrls.TPropertyPage.Modified 1618
AxCtrls.TPropertyPage.OldCreateOrder 1634
AxCtrls.TPropertyPage.OleObject 1635
AxCtrls.TPropertyPage.OleObjects 1635
AxCtrls.TPropertyPage.OnActivate 1646
AxCtrls.TPropertyPage.OnClick 1650
AxCtrls.TPropertyPage.OnClose 1652
AxCtrls.TPropertyPage.OnContextPopup 1653
AxCtrls.TPropertyPage.OnCreate 1654
AxCtrls.TPropertyPage.OnDblClick 1655
AxCtrls.TPropertyPage.OnDeactivate 1656
AxCtrls.TPropertyPage.OnDestroy 1656

AxCtrls.TPropertyPage.OnDragDrop 1657
AxCtrls.TPropertyPage.OnDragOver 1659
AxCtrls.TPropertyPage.OnHide 1662
AxCtrls.TPropertyPage.OnKeyDown 1664
AxCtrls.TPropertyPage.OnKeyPress 1667
AxCtrls.TPropertyPage.OnKeyUp 1668
AxCtrls.TPropertyPage.OnMouseDown 1669
AxCtrls.TPropertyPage.OnMouseMove 1672
AxCtrls.TPropertyPage.OnMouseUp 1674
AxCtrls.TPropertyPage.OnPaint 1676
AxCtrls.TPropertyPage.OnResize 1677
AxCtrls.TPropertyPage.OnShow 1678
AxCtrls.TPropertyPage.ParentFont 1636
AxCtrls.TPropertyPage.PixelsPerInch 1636
AxCtrls.TPropertyPage.PopupMenu 1636
AxCtrls.TPropertyPage.PrintScale 1641
AxCtrls.TPropertyPage.Scaled 1641
AxCtrls.TPropertyPage.ShowHint 1642
AxCtrls.TPropertyPage.UpdateObject 1618
AxCtrls.TPropertyPage.UpdatePropertyPage 1619
AxCtrls.TPropertyPage.VertScrollBar 1643
AxCtrls.TPropertyPage.Visible 1644
AxCtrls.TPropertyPage.Width 1645
AxCtrls.TPropertyPageClass 1716
AxCtrls.TPropertyPageClass type 1716
AxCtrls.TPropertyPageImpl 1679
AxCtrls.TPropertyPageImpl.Activate 1681
AxCtrls.TPropertyPageImpl.Apply 1681
AxCtrls.TPropertyPageImpl.Deactivate 1681
AxCtrls.TPropertyPageImpl.EditProperty 1682
AxCtrls.TPropertyPageImpl.GetPageInfo 1682
AxCtrls.TPropertyPageImpl.Help 1682
AxCtrls.TPropertyPageImpl.InitPropertyPage 1682
AxCtrls.TPropertyPageImpl.IsPageDirty 1683
AxCtrls.TPropertyPageImpl.Move 1683
AxCtrls.TPropertyPageImpl.PropertyPage 1684
AxCtrls.TPropertyPageImpl.SetObjects 1683
AxCtrls.TPropertyPageImpl.SetPageSite 1683
AxCtrls.TPropertyPageImpl.Show 1684
AxCtrls.TPropertyPageImpl.TranslateAccelerator 1684
AxCtrls.TReflectorWindow 1684
AxCtrls.TReflectorWindow.Create 1695
AxCtrls.TReflectorWindow.CreateParams 1695
AxCtrls.TStringsAdapter 1696
AxCtrls.TStringsAdapter._NewEnum 1699
AxCtrls.TStringsAdapter.Add 1699
AxCtrls.TStringsAdapter.Clear 1699
AxCtrls.TStringsAdapter.Count 1700
AxCtrls.TStringsAdapter.Create 1698
AxCtrls.TStringsAdapter.Get_ControlDefault 1700
AxCtrls.TStringsAdapter.Get_Item 1700
AxCtrls.TStringsAdapter.ReferenceStrings 1700
AxCtrls.TStringsAdapter.ReleaseStrings 1700
AxCtrls.TStringsAdapter.Remove 1701
AxCtrls.TStringsAdapter.Set_ControlDefault 1701
AxCtrls.TStringsAdapter.Set_Item 1701
axrComponentOnly enumeration member 3268
axrIncludeDescendants enumeration member 3268

B

BandActn 1719
BandActn namespace 1719
    Classes 1720
BandActn.TCustomizeActionBars 1720
BandActn.TCustomizeActionBars.ActionManager 1726
BandActn.TCustomizeActionBars.Create 1725
BandActn.TCustomizeActionBars.CustomizeDlg 1726
BandActn.TCustomizeActionBars.ExecuteTarget 1725
BandActn.TCustomizeActionBars.HandlesTarget 1726
batAppend enumeration member 10251
batAppendUpdate enumeration member 10251
batCopy enumeration member 10251
batDelete enumeration member 10251
batUpdate enumeration member 10251
BDEReg 1729
BDEReg namespace 1729
    Classes 1731
    Constants 1758

Functions 1756
BDEReg.cDefaultSessionSprigName 1758
BDEReg.cDefaultSessionSprigName constant 1758
BDEReg.cImpliedDatabaseSprigPrefix 1758
BDEReg.cImpliedDatabaseSprigPrefix constant 1758
BDEReg.cImpliedSessionSprigPrefix 1758
BDEReg.cImpliedSessionSprigPrefix constant 1758
BDEReg.Register 1756
BDEReg.Register function 1756
BDEReg.SprigBDEImpliedDatabaseName 1756
BDEReg.SprigBDEImpliedDatabaseName function 1756
BDEReg.SprigBDEImpliedSessionName 1757
BDEReg.SprigBDEImpliedSessionName function 1757
BDEReg.SprigBDESessionName 1757
BDEReg.SprigBDESessionName function 1757
BDEReg.TBatchMoveSprig 1731
BDEReg.TBatchMoveSprig.AnyProblems 1732
BDEReg.TBatchMoveSprig.Caption 1732
BDEReg.TBDEDataSetSprig 1732
BDEReg.TBDEDataSetSprig.GetDSDesignerClass 1733
BDEReg.TDatabaseSprig 1733
BDEReg.TDatabaseSprig.AnyProblems 1734
BDEReg.TDatabaseSprig.Caption 1734
BDEReg.TDatabaseSprig.DragDropTo 1734
BDEReg.TDatabaseSprig.DragOverTo 1735
BDEReg.TDatabaseSprig.FigureParent 1735
BDEReg.TDatabaseSprig.Name 1735
BDEReg.TDatabaseSprig.PaletteOverTo 1735
BDEReg.TDBDataSetSprig 1736
BDEReg.TDBDataSetSprig.AnyProblems 1736
BDEReg.TDBDataSetSprig.DragDropTo 1737
BDEReg.TDBDataSetSprig.DragOverTo 1737
BDEReg.TDBDataSetSprig.FigureParent 1737
BDEReg.TDBDataSetSprig.PaletteOverTo 1737
BDEReg.TDBDataSetSprig.Reparent 1738
BDEReg.TDefaultSessionSprig 1738
BDEReg.TDefaultSessionSprig.Caption 1739
BDEReg.TDefaultSessionSprig.ItemClass 1739
BDEReg.TDefaultSessionSprig.UniqueName 1739
BDEReg.TImpliedDatabaseSprig 1739
BDEReg.TImpliedDatabaseSprig.AnyProblems 1740
BDEReg.TImpliedDatabaseSprig.Caption 1740
BDEReg.TImpliedDatabaseSprig.DragDropTo 1741
BDEReg.TImpliedDatabaseSprig.DragOverTo 1741
BDEReg.TImpliedDatabaseSprig.FigureParent 1741
BDEReg.TImpliedDatabaseSprig.ItemClass 1741
BDEReg.TImpliedDatabaseSprig.UniqueName 1741
BDEReg.TImpliedSessionSprig 1742
BDEReg.TImpliedSessionSprig.Caption 1742
BDEReg.TImpliedSessionSprig.ItemClass 1743
BDEReg.TImpliedSessionSprig.UniqueName 1743
BDEReg.TNestedTableSprig 1743
BDEReg.TNestedTableSprig.ParentProperty 1744
BDEReg.TQuerySprig 1744
BDEReg.TQuerySprig.AnyProblems 1745
BDEReg.TSessionSprig 1745
BDEReg.TSessionSprig.AnyProblems 1746
BDEReg.TSessionSprig.Caption 1746
BDEReg.TSessionSprig.Name 1746
BDEReg.TStoredProcSprig 1747
BDEReg.TStoredProcSprig.AnyProblems 1747
BDEReg.TStoredProcSprig.Caption 1748
BDEReg.TTableFieldLinkProperty 1748
BDEReg.TTableFieldLinkProperty.Edit 1749
BDEReg.TTableFieldLinkProperty.GetFieldNamesForIndex 1749
BDEReg.TTableFieldLinkProperty.GetIndexBased 1750
BDEReg.TTableFieldLinkProperty.GetIndexDefs 1750
BDEReg.TTableFieldLinkProperty.GetIndexFieldNames 1750
BDEReg.TTableFieldLinkProperty.GetIndexName 1750
BDEReg.TTableFieldLinkProperty.GetMasterFields 1751
BDEReg.TTableFieldLinkProperty.IndexBased 1752
BDEReg.TTableFieldLinkProperty.IndexDefs 1752
BDEReg.TTableFieldLinkProperty.IndexFieldNames 1752
BDEReg.TTableFieldLinkProperty.IndexName 1752
BDEReg.TTableFieldLinkProperty.MasterFields 1753
BDEReg.TTableFieldLinkProperty.SetIndexFieldNames 1751
BDEReg.TTableFieldLinkProperty.SetIndexName 1751
BDEReg.TTableFieldLinkProperty.SetMasterFields 1751

BDEReg.TTableSprig 1753
BDEReg.TTableSprig.AnyProblems 1754
BDEReg.TTableSprig.Caption 1754
BDEReg.TUpdateSQLSprig 1754
BDEReg.TUpdateSQLSprig.AnyProblems 1755
bdLeftToRight enumeration member 3269
bdRightToLeft enumeration member 3269
bdRightToLeftNoAlign enumeration member 3269
bdRightToLeftReadingOnly enumeration member 3269
beBottom enumeration member 6270
beEither enumeration member 564
beLeft enumeration member 564, 6270
beRight enumeration member 564, 6270
beTop enumeration member 6270
bfBOF enumeration member 7606
bfCurrent enumeration member 7606
bfEOF enumeration member 7606
bfError enumeration member 13264
bfInfo enumeration member 13264
bfInserted enumeration member 7606
bfNone enumeration member 13264
bfWarning enumeration member 13264
bhoHideAlways enumeration member 25315
bhoHideOnActionNotVisible enumeration member 25315
bhoHideOnDisabledAction enumeration member 25315
bhoHideOnNoExecuteAccess enumeration member 25315
bifBrowseForComputer enumeration member 20822
bifBrowseForPrinter enumeration member 20822
bifBrowseIncludeFiles enumeration member 20822
bifBrowseIncludeURLS enumeration member 20822
bifDontGoBelowDomain enumeration member 20822
bifEditBox enumeration member 20822
bifNewDialogStyle enumeration member 20822
bifNoNewFolderButton enumeration member 20822
bifNoTranslateTargets enumeration member 20822
bifReturnFSAncestors enumeration member 20822
bifReturnOnlyFSDirs enumeration member 20822
bifShareable enumeration member 20822
bifStatusText enumeration member 20822
bifUAHint enumeration member 20822
bifUseNewUI enumeration member 20822
bifValidate enumeration member 20822
biHelp enumeration member 14354
biMaximize enumeration member 14354
biMinimize enumeration member 14354
binCustom enumeration member 17432
binMonth enumeration member 17432
binNone enumeration member 17432
binQuarter enumeration member 17432
binSet enumeration member 17432
binYear enumeration member 17432
biSystemMenu enumeration member 14354
bkAbort enumeration member 2392
bkAll enumeration member 2392
bkCancel enumeration member 2392
bkClose enumeration member 2392
bkCustom enumeration member 2392
bkFlat enumeration member 6271
bkHelp enumeration member 2392
bkIgnore enumeration member 2392
bkNo enumeration member 2392
bkNone enumeration member 6271
bkOK enumeration member 2392
bkRetry enumeration member 2392
bkSoft enumeration member 6271
bkTile enumeration member 6271
bkYes enumeration member 2392
bl128 enumeration member 5707
bl16 enumeration member 5707
bl2 enumeration member 5707
bl256 enumeration member 5707
bl32 enumeration member 5707
bl4 enumeration member 5707
bl64 enumeration member 5707
bl8 enumeration member 5707
blGlyphBottom enumeration member 2393
blGlyphLeft enumeration member 2393
blGlyphRight enumeration member 2393

blGlyphTop enumeration member 2393
blLeftBanner enumeration member 564
blNormal enumeration member 564
blRightBanner enumeration member 564
blStretch enumeration member 564
blTile enumeration member 564
bmBlocking enumeration member 20136
bmClick enumeration member 5445
bmDblClick enumeration member 5445
bmDDB enumeration member 14630
bmDIB enumeration member 14630
bmNonBlocking enumeration member 20136
bmNone enumeration member 5445
bmRead enumeration member 7605
bmReadWrite enumeration member 7605
bmThreadBlocking enumeration member 20136
bmWrite enumeration member 7605
boBottomToTop enumeration member 565
boLeftToRight enumeration member 565
boRightToLeft enumeration member 565
Borland 1761
Borland namespace 1761
Borland.Data 1762
Borland.Data namespace 1762
Borland.Data.AdoDbxClientProvider 1762
Borland.Data.AdoDbxClientProvider namespace 1762
    Classes 1763
    Constants 1891
    Types 1890
Borland.Data.AdoDbxClientProvider.SCommandAlreadyPrepared 1891
Borland.Data.AdoDbxClientProvider.SCommandAlreadyPrepared constant 1891
Borland.Data.AdoDbxClientProvider.SInvalidFieldType 1892
Borland.Data.AdoDbxClientProvider.SInvalidFieldType constant 1892
Borland.Data.AdoDbxClientProvider.SUnSupported 1892
Borland.Data.AdoDbxClientProvider.SUnSupported constant 1892
Borland.Data.AdoDbxClientProvider.TAdoDbxCommand 1763
Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.Cancel 1767
Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.Clone 1767
Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.CommandText 1772
Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.CommandTimeout 1773
Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.CommandType 1773
Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.CreateDbParameter 1767
Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.DesignTimeVisible 1774
Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.Destroy 1766
Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.ExecuteDbDataReader 1767
Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.ExecuteNonQuery 1768
Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.ExecuteScalar 1768
Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.get_CommandText 1768
Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.get_CommandTimeout 1769
Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.get_CommandType 1769
Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.get_DbConnection 1769
Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.get_DbParameterCollection 1769
Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.get_DbTransaction 1769

Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.get_DesignTimeVisible
1770

Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.get_UpdatedRowSource
1770

Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.Parameters
1774

Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.Prepare
1770

Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.set_CommandText
1770

Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.set_CommandTimeout
1771

Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.set_CommandType
1771

Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.set_DbConnection
1771

Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.set_DbTransaction
1771

Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.set_DesignTimeVisible
1771

Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.set_UpdatedRowSource
1772

Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.Transaction
1774

Borland.Data.AdoDbxClientProvider.TAdoDbxCommand.UpdateRowSource
1775

Borland.Data.AdoDbxClientProvider.TAdoDbxCommandBuilder
1776

Borland.Data.AdoDbxClientProvider.TAdoDbxCommandBuilder.AdoDbxRowUpdatingHandler
1778

Borland.Data.AdoDbxClientProvider.TAdoDbxCommandBuilder.ApplyParameterInfo
1778

Borland.Data.AdoDbxClientProvider.TAdoDbxCommandBuilder.Create
1777

Borland.Data.AdoDbxClientProvider.TAdoDbxCommandBuilder.GetParameterName
1778, 1779

Borland.Data.AdoDbxClientProvider.TAdoDbxCommandBuilder.GetParameterPlaceholder
1779

Borland.Data.AdoDbxClientProvider.TAdoDbxCommandBuilder.SetRowUpdatingHandler
1779

Borland.Data.AdoDbxClientProvider.TAdoDbxConnection 1779

Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.AdoDbxIsolation
1784

Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.BeginDbTransaction
1784

Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.ChangeDatabase
1784

Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.Clone
1785

Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.Close
1785

Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.ConnectionString
1788

Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.ConnectionTimeout
1789

Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.Create
1781

Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.CreateDbCommand
1785

Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.Database
1789

Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.DataSource
1789

Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.Dbx
1790

Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.Destroy
1782

Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.FConnectionString
1782

Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.FConnectionTimeout
1783

Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.FDbxExtension
1783
Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.FDriverName
1783
Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.FSchemaItems
1783
Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.get_ConnectionString
1785
Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.get_ConnectionTimeout
1786
Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.get_Database
1786
Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.get_DataSource
1786
Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.get_ServerVersion
1786
Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.get_State
1786
Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.GetSchema
1787
Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.Open
1787
Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.ServerVersion
1790
Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.set_ConnectionString
1788
Borland.Data.AdoDbxClientProvider.TAdoDbxConnection.State
1790
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder
1790
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.ConnectionName
1806
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.Create
1793
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.Database
1806
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.DelegateConnection
1807
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.DelegateDriver
1807
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.DriverAssembly
1807
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.DriverAssemblyLoader
1808
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.DriverName
1808
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.DriverPackage
1808
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.DriverPackageLoader
1809
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.DriverUnit
1809
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.ErrorResourceFile
1809
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_ConnectionName
1795
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_Database
1795
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_DelegateConnection
1795
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_DelegateDriver
1795
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_DriverAssembly
1795
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_DriverAssemblyLoader
1796
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_DriverName
1796
Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_DriverPackage

1796

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_DriverPackageLoader
1796

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_DriverUnit
1796

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_ErrorResourceFile
1797

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_GetDriverFunc
1797

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_HostName
1797

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_IsolationLevel
1797

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_Item
1797

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_LibraryName
1798

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_MaxBlobSize
1798

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_MetaDataAssemblyLoader
1798

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_MetaDataPackageLoader
1798

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_Password
1798

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_Port
1799

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_ProductVersion
1799

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_Role
1799

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_SchemaOverride
1799

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_User_Name
1799

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.get_VendorLib
1800

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.GetDriverFunc
1809

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.HostName
1810

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.IsolationLevel
1810

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.LibraryName
1811

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.MaxBlobSize
1811

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.MetaDataAssemblyLoader
1811

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.MetaDataPackageLoader
1811

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.Password
1812

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.Port
1812

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.ProductVersion
1812

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.Role
1813

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.SchemaOverride
1813

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_ConnectionName
1800

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_Database
1800

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_DelegateConnection
1800

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_DelegateDriver

1800

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_DriverAssembly
1801

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_DriverAssemblyLoader
1801

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_DriverName
1801

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_DriverPackage
1801

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_DriverPackageLoader
1801

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_DriverUnit
1802

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_ErrorResourceFile
1802

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_GetDriverFunc
1802

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_HostName
1802

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_IsolationLevel
1802

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_Item
1803

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_LibraryName
1803

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_MaxBlobSize
1803

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_MetaDataAssemblyLoader
1803

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_MetaDataPackageLoader
1803

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_Password
1804

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_Port
1804

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_ProductVersion
1804

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_Role
1804

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_SchemaOverride
1804

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_User_Name
1805

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.set_VendorLib
1805

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.TryGetValue
1805

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.User_Name
1814

Borland.Data.AdoDbxClientProvider.TAdoDbxConnectionStringBuilder.VendorLib
1814

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter 1814

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.add_RowUpdated
1819

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.add_RowUpdating
1820

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.Create
1818

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.DeleteCommand
1823

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.FEventRowUpdated
1819

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.FEventRowUpdating
1819

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.get_DeleteCommand
1820

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.get_InsertCommand
1820

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.get_

SelectCommand
1821

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.get_UpdateCommand
1821

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.InsertCommand
1824

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.OnRowUpdated
1821

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.OnRowUpdating
1821

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.remove_RowUpdated
1821

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.remove_RowUpdating
1822

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.RowUpdated
1826

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.RowUpdating
1826

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.SelectCommand
1825

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.set_DeleteCommand
1822

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.set_InsertCommand
1822

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.set_SelectCommand
1823

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.set_UpdateCommand
1823

Borland.Data.AdoDbxClientProvider.TAdoDbxDataAdapter.UpdateCommand
1825

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader 1827

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.Close
1832

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.Depth
1843

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.Destroy
1830

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.FCommandBehavior
1831

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.FDbxCommand
1831

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.FDbxReader
1831

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.FieldCount
1843

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.FPeekAheadRead
1832

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.get_Depth
1833

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.get_FieldCount
1833

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.get_HasRows
1833

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.get_IsClosed
1833

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.get_Item
1834

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.get_RecordsAffected
1834

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetBoolean
1834

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetByte
1835

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetBytes
1835

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetChar
1835

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetChars
1836

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetData
1836

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetDataTypeName
1837

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetDateTime
1837

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetDecimal
1837

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetDouble
1837

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetEnumerator
1838

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetFieldType
1838

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetFloat
1838

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetGuid
1839

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetInt16
1839

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetInt32
1839

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetInt64
1839

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetName
1840

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetOrdinal
1840

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetSchemaTable
1840

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetString
1841

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetValue
1841

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.GetValues
1841

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.HasRows
1843

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.IsClosed
1844

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.IsDBNull
1841

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.Item
1844

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.NextResult
1842

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.Read
1842

Borland.Data.AdoDbxClientProvider.TAdoDbxDataReader.RecordsAffected
1844

Borland.Data.AdoDbxClientProvider.TAdoDbxDataSourceEnumerator
1845

Borland.Data.AdoDbxClientProvider.TAdoDbxException 1845

Borland.Data.AdoDbxClientProvider.TAdoDbxException.Create
1846

Borland.Data.AdoDbxClientProvider.TAdoDbxExtension 1846

Borland.Data.AdoDbxClientProvider.TAdoDbxExtension.Create
1847

Borland.Data.AdoDbxClientProvider.TAdoDbxExtension.FConnection
1847

Borland.Data.AdoDbxClientProvider.TAdoDbxExtension.GetVendorProperty
1848

Borland.Data.AdoDbxClientProvider.TAdoDbxMetaDataColumnNames
1848

Borland.Data.AdoDbxClientProvider.TAdoDbxMetaDataColumnNames.RestrictionDefault
1849

Borland.Data.AdoDbxClientProvider.TAdoDbxMetaDataColumnNames.RestrictionName
1849

Borland.Data.AdoDbxClientProvider.TAdoDbxMetaDataColumnNames.RestrictionNumber
1849

Borland.Data.AdoDbxClientProvider.TAdoDbxParameter 1850
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.Clone 1855
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.Create 1852
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.DbType 1860
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.DbxDataType 1860
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.DbxSubType 1861
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.Destroy 1853
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.Direction 1861
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.FConnection 1853
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.FDbxContext 1853
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.FSourceColumn 1854
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.FSourceColumnNullMapping 1854
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.FSourceVersion 1854
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.get_DbType 1855
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.get_Direction 1855
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.get_IsNullable 1855
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.get_ParameterName 1856
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.get_Precision 1856
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.get_Scale 1856
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.get_Size 1856
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.get_SourceColumn 1856
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.get_SourceColumnNullMapping 1857
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.get_SourceVersion 1857
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.get_Value 1857
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.IsNullable 1862
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.ParameterName 1862
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.Precision 1862
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.ResetDbType 1857
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.Scale 1863
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.set_DbType 1857
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.set_Direction 1858
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.set_IsNullable 1858
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.set_ParameterName 1858
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.set_Precision 1858
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.set_Scale 1858
Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.set_Size

1859

Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.set_SourceColumn
1859

Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.set_SourceColumnNullMapping
1859

Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.set_SourceVersion
1859

Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.set_Value
1859

Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.Size
1863

Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.SourceColumn
1863

Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.SourceColumnNullMapping
1864

Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.SourceVersion
1864

Borland.Data.AdoDbxClientProvider.TAdoDbxParameter.Value
1865

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection
1865

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.Add
1868

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.AddRange
1868

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.Clear
1869

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.Contains
1869

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.CopyTo
1869

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.Destroy
1867

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.FDBXParameters
1867

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.FParameters
1867

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.get_Count
1870

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.get_IsFixedSize
1870

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.get_IsReadOnly
1870

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.get_IsSynchronized
1870

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.get_SyncRoot
1871

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.GetEnumerator
1871

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.GetParameter
1871, 1872

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.IndexOf
1872

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.Insert
1872

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.Remove
1873

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.RemoveAt
1873

Borland.Data.AdoDbxClientProvider.TAdoDbxParameterCollection.SetParameter
1874

Borland.Data.AdoDbxClientProvider.TAdoDbxProviderFactory
1874

Borland.Data.AdoDbxClientProvider.TAdoDbxProviderFactory.AdoDbxException
1877

Borland.Data.AdoDbxClientProvider.TAdoDbxProviderFactory.CanCreateDataSourceEnumerator
1880

Borland.Data.AdoDbxClientProvider.TAdoDbxProviderFactory.Create
1875, 1876

Borland.Data.AdoDbxClientProvider.TAdoDbxProviderFactory.CreateCommand

1877

Borland.Data.AdoDbxClientProvider.TAdoDbxProviderFactory.CreateCommandBuilder
1877

Borland.Data.AdoDbxClientProvider.TAdoDbxProviderFactory.CreateConnection
1878

Borland.Data.AdoDbxClientProvider.TAdoDbxProviderFactory.CreateConnectionStringBuilder
1878

Borland.Data.AdoDbxClientProvider.TAdoDbxProviderFactory.CreateDataAdapter
1878

Borland.Data.AdoDbxClientProvider.TAdoDbxProviderFactory.CreateDataSourceEnumerator
1879

Borland.Data.AdoDbxClientProvider.TAdoDbxProviderFactory.CreateParameter
1879

Borland.Data.AdoDbxClientProvider.TAdoDbxProviderFactory.DbxToAdoType
1879

Borland.Data.AdoDbxClientProvider.TAdoDbxProviderFactory.Destroy
1876

Borland.Data.AdoDbxClientProvider.TAdoDbxProviderFactory.Instance
1876

Borland.Data.AdoDbxClientProvider.TAdoDbxProviderFactory.ReadCanCreateDataSourceEnumerator
1879

Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatedEventArgs
1880

Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatedEventArgs.Command
1881

Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatedEventArgs.get_Command
1881

Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatedEventHandler
1890

Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatedEventHandler
type 1890

Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatingEventArgs
1881

Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatingEventArgs.Command
1883

Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatingEventArgs.get_Command
1882

Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatingEventArgs.set_Command
1882

Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatingEventHandler
1890

Borland.Data.AdoDbxClientProvider.TAdoDbxRowUpdatingEventHandler
type 1890

Borland.Data.AdoDbxClientProvider.TAdoDbxSchemaTableColumnNames
1883

Borland.Data.AdoDbxClientProvider.TAdoDbxSchemaTableColumnNames.DbxDataType
1884

Borland.Data.AdoDbxClientProvider.TAdoDbxSchemaTableColumnNames.DbxSubType
1884

Borland.Data.AdoDbxClientProvider.TAdoDbxTransaction 1884

Borland.Data.AdoDbxClientProvider.TAdoDbxTransaction.Commit
1887

Borland.Data.AdoDbxClientProvider.TAdoDbxTransaction.Destroy
1886

Borland.Data.AdoDbxClientProvider.TAdoDbxTransaction.FConnection
1886

Borland.Data.AdoDbxClientProvider.TAdoDbxTransaction.FIsolation
1886

Borland.Data.AdoDbxClientProvider.TAdoDbxTransaction.FTransaction
1887

Borland.Data.AdoDbxClientProvider.TAdoDbxTransaction.get_DbConnection
1887

Borland.Data.AdoDbxClientProvider.TAdoDbxTransaction.get_IsolationLevel
1888

Borland.Data.AdoDbxClientProvider.TAdoDbxTransaction.Rollback
1888

Borland.Data.AdoDbxClientProvider.TConnectionStringCategories
1888

Borland.Data.AdoDbxClientProvider.TConnectionStringCategori

es.DataBase
1889

Borland.Data.AdoDbxClientProvider.TConnectionStringCategories.DriverConfiguration
1889

Borland.Data.AdoDbxClientProvider.TConnectionStringCategories.Dynalink
1889

Borland.Data.AdoDbxClientProvider.TObjectArray 1891

Borland.Data.AdoDbxClientProvider.TObjectArray type 1891

Borland.Data.DataStore 1892

Borland.Data.DataStore namespace 1892

Classes 1893

Types 2129

Borland.Data.DataStore.ConColumns 1894

Borland.Data.DataStore.ConColumns.Birth 1896

Borland.Data.DataStore.ConColumns.FileName 1896

Borland.Data.DataStore.ConColumns.Host 1896

Borland.Data.DataStore.ConColumns.Id 1896

Borland.Data.DataStore.ConColumns.Ip 1896

Borland.Data.DataStore.ConColumns.Remarks 1897

Borland.Data.DataStore.ConColumns.RemotePort 1897

Borland.Data.DataStore.ConColumns.User 1897

Borland.Data.DataStore.ConnectionProperties 1897

Borland.Data.DataStore.ConnectionProperties.ClientDataDirectory
1901

Borland.Data.DataStore.ConnectionProperties.CommitOnClose
1901

Borland.Data.DataStore.ConnectionProperties.ConnectionTimeout
1901

Borland.Data.DataStore.ConnectionProperties.CreateDb 1901

Borland.Data.DataStore.ConnectionProperties.Database 1901

Borland.Data.DataStore.ConnectionProperties.ExtendedMetadata
1902

Borland.Data.DataStore.ConnectionProperties.Fileio 1902

Borland.Data.DataStore.ConnectionProperties.HeuristicCompletion
1902

Borland.Data.DataStore.ConnectionProperties.Host 1902

Borland.Data.DataStore.ConnectionProperties.LockWaitTime
1902

Borland.Data.DataStore.ConnectionProperties.LogFilters 1903

Borland.Data.DataStore.ConnectionProperties.MaxPoolSize
1903

Borland.Data.DataStore.ConnectionProperties.MaxRowLocks
1903

Borland.Data.DataStore.ConnectionProperties.MinPoolSize
1903

Borland.Data.DataStore.ConnectionProperties.Password 1903

Borland.Data.DataStore.ConnectionProperties.PingServer 1904

Borland.Data.DataStore.ConnectionProperties.Pooling 1904

Borland.Data.DataStore.ConnectionProperties.Port 1904

Borland.Data.DataStore.ConnectionProperties.PriorFileName
1904

Borland.Data.DataStore.ConnectionProperties.Protocol 1904

Borland.Data.DataStore.ConnectionProperties.ReadOnly 1905

Borland.Data.DataStore.ConnectionProperties.ReadOnlyTx
1905

Borland.Data.DataStore.ConnectionProperties.ReadOnlyTxDelay
1905

Borland.Data.DataStore.ConnectionProperties.Remarks 1905

Borland.Data.DataStore.ConnectionProperties.ShutdownServer
1905

Borland.Data.DataStore.ConnectionProperties.TableLockTables

1906

Borland.Data.DataStore.ConnectionProperties.User 1906

Borland.Data.DataStore.DatabaseColumns 1906

Borland.Data.DataStore.DatabaseColumns.ALogDir 1908

Borland.Data.DataStore.DatabaseColumns.BlockSize 1908

Borland.Data.DataStore.DatabaseColumns.CheckpointFrequency
1908

Borland.Data.DataStore.DatabaseColumns.DefaultSchemaName
1908

Borland.Data.DataStore.DatabaseColumns.LogBlockSize 1908

Borland.Data.DataStore.DatabaseColumns.LogEnabled 1909

Borland.Data.DataStore.DatabaseColumns.LogPreallocateCount
1909

Borland.Data.DataStore.DatabaseColumns.MaxLogSize 1909

Borland.Data.DataStore.DatabaseColumns.MaxOpenLogs 1909

Borland.Data.DataStore.DatabaseColumns.RecordStatus 1909

Borland.Data.DataStore.DatabaseColumns.SoftCommit 1910

Borland.Data.DataStore.DatabaseColumns.StoreVersion 1910

Borland.Data.DataStore.DatabaseStatusTable 1910
Borland.Data.DataStore.DatabaseStatusTable.HighestLsn 1911
Borland.Data.DataStore.DatabaseStatusTable.LastCheckpointLsn 1911
Borland.Data.DataStore.DatabaseStatusTable.LowestLsn 1911
Borland.Data.DataStore.DatabaseStatusTable.LowestLsnRemarks 1912
Borland.Data.DataStore.DataStoreCommand 1912
Borland.Data.DataStore.DataStoreCommand.Cancel 1915
Borland.Data.DataStore.DataStoreCommand.Clone 1915
Borland.Data.DataStore.DataStoreCommand.Close 1916
Borland.Data.DataStore.DataStoreCommand.CommandText 1919
Borland.Data.DataStore.DataStoreCommand.CommandTimeout 1920
Borland.Data.DataStore.DataStoreCommand.CommandType 1920
Borland.Data.DataStore.DataStoreCommand.Connection 1921
Borland.Data.DataStore.DataStoreCommand.CreateDbParameter 1916
Borland.Data.DataStore.DataStoreCommand.CreateParameter 1916
Borland.Data.DataStore.DataStoreCommand.DataReaderClosed 1916
Borland.Data.DataStore.DataStoreCommand.DataStoreCommand 1914
Borland.Data.DataStore.DataStoreCommand.DbConnection 1921
Borland.Data.DataStore.DataStoreCommand.DbParameterCollection 1921
Borland.Data.DataStore.DataStoreCommand.DbTransaction 1922
Borland.Data.DataStore.DataStoreCommand.DesignTimeVisible 1922
Borland.Data.DataStore.DataStoreCommand.Dispose 1917
Borland.Data.DataStore.DataStoreCommand.ExecuteDbDataReader 1917
Borland.Data.DataStore.DataStoreCommand.ExecuteNonQuery 1917
Borland.Data.DataStore.DataStoreCommand.ExecuteReader 1918
Borland.Data.DataStore.DataStoreCommand.ExecuteScalar 1918
Borland.Data.DataStore.DataStoreCommand.FailIfNotOpen 1919
Borland.Data.DataStore.DataStoreCommand.Parameters 1922
Borland.Data.DataStore.DataStoreCommand.Prepare 1919
Borland.Data.DataStore.DataStoreCommand.RecordsAffected 1922
Borland.Data.DataStore.DataStoreCommand.Transaction 1923
Borland.Data.DataStore.DataStoreCommand.UpdatedRowSource 1923
Borland.Data.DataStore.DataStoreConnection 1923
Borland.Data.DataStore.DataStoreConnection.BeginDbTransaction 1926
Borland.Data.DataStore.DataStoreConnection.BeginTransaction 1926
Borland.Data.DataStore.DataStoreConnection.ChangeDatabase 1927
Borland.Data.DataStore.DataStoreConnection.Clone 1927
Borland.Data.DataStore.DataStoreConnection.Close 1927
Borland.Data.DataStore.DataStoreConnection.ConnectionString 1930
Borland.Data.DataStore.DataStoreConnection.ConnectionTimeout 1931
Borland.Data.DataStore.DataStoreConnection.CreateCommand 1928
Borland.Data.DataStore.DataStoreConnection.CreateDbCommand 1928
Borland.Data.DataStore.DataStoreConnection.Database 1931
Borland.Data.DataStore.DataStoreConnection.DataSource 1931
Borland.Data.DataStore.DataStoreConnection.DataStoreConnection 1925
Borland.Data.DataStore.DataStoreConnection.Dispose 1928
Borland.Data.DataStore.DataStoreConnection.EndTransaction 1929
Borland.Data.DataStore.DataStoreConnection.FailIfNotOpen

1929
Borland.Data.DataStore.DataStoreConnection.GetSchema 1929
Borland.Data.DataStore.DataStoreConnection.InternalState 1932
Borland.Data.DataStore.DataStoreConnection.MetaDataGenerator 1932
Borland.Data.DataStore.DataStoreConnection.Open 1930
Borland.Data.DataStore.DataStoreConnection.ServerVersion 1932
Borland.Data.DataStore.DataStoreConnection.State 1932
Borland.Data.DataStore.DataStoreDataAdapter 1933
Borland.Data.DataStore.DataStoreDataAdapter.BeginInit 1935
Borland.Data.DataStore.DataStoreDataAdapter.CreateRowUpdatedEvent 1935
Borland.Data.DataStore.DataStoreDataAdapter.CreateRowUpdatingEvent 1936
Borland.Data.DataStore.DataStoreDataAdapter.DataSet 1937
Borland.Data.DataStore.DataStoreDataAdapter.DataStoreDataAdapter 1934
Borland.Data.DataStore.DataStoreDataAdapter.DeleteCommand 1937
Borland.Data.DataStore.DataStoreDataAdapter.EndInit 1936
Borland.Data.DataStore.DataStoreDataAdapter.InsertCommand 1937
Borland.Data.DataStore.DataStoreDataAdapter.OnRowUpdated 1936
Borland.Data.DataStore.DataStoreDataAdapter.OnRowUpdating 1936
Borland.Data.DataStore.DataStoreDataAdapter.RowUpdated 1939
Borland.Data.DataStore.DataStoreDataAdapter.RowUpdating 1939
Borland.Data.DataStore.DataStoreDataAdapter.SelectCommand 1938
Borland.Data.DataStore.DataStoreDataAdapter.UpdateCommand 1938
Borland.Data.DataStore.DataStoreErrorCode 1939
Borland.Data.DataStore.DataStoreErrorCode.ActionComplete 1965
Borland.Data.DataStore.DataStoreErrorCode.AfterLast 1966
Borland.Data.DataStore.DataStoreErrorCode.AggNotValidHere 1966
Borland.Data.DataStore.DataStoreErrorCode.AggOperatorNotFound 1966
Borland.Data.DataStore.DataStoreErrorCode.AlreadyLoading 1966
Borland.Data.DataStore.DataStoreErrorCode.AmbiguousLogdir 1966
Borland.Data.DataStore.DataStoreErrorCode.AmbiguousType 1967
Borland.Data.DataStore.DataStoreErrorCode.AnchorTooOld 1967
Borland.Data.DataStore.DataStoreErrorCode.AppendFailed 1967
Borland.Data.DataStore.DataStoreErrorCode.ApplicationError 1967
Borland.Data.DataStore.DataStoreErrorCode.BadColumnIndex 1967
Borland.Data.DataStore.DataStoreErrorCode.BadDirection 1968
Borland.Data.DataStore.DataStoreErrorCode.BadFetchsize 1968
Borland.Data.DataStore.DataStoreErrorCode.BadFieldlist 1968
Borland.Data.DataStore.DataStoreErrorCode.BadMaxfieldsize 1968
Borland.Data.DataStore.DataStoreErrorCode.BadMaxrows 1968
Borland.Data.DataStore.DataStoreErrorCode.BadOrdinal 1969
Borland.Data.DataStore.DataStoreErrorCode.BadProcedureProperties 1969
Borland.Data.DataStore.DataStoreErrorCode.BadQueryProperties 1969
Borland.Data.DataStore.DataStoreErrorCode.BadUrl 1969
Borland.Data.DataStore.DataStoreErrorCode.BeforeFirst 1969
Borland.Data.DataStore.DataStoreErrorCode.BigdecimalPrecisionError 1970
Borland.Data.DataStore.DataStoreErrorCode.BirthstampMismatch 1970
Borland.Data.DataStore.DataStoreErrorCode.CannotAssignTypes 1970
Borland.Data.DataStore.DataStoreErrorCode.CannotCast 1970

Borland.Data.DataStore.DataStoreErrorCode.CannotChangeColumn
1970

Borland.Data.DataStore.DataStoreErrorCode.CannotChangeColumnDataType
1971

Borland.Data.DataStore.DataStoreErrorCode.CannotDittoExisting
1971

Borland.Data.DataStore.DataStoreErrorCode.CannotFindTableName
1971

Borland.Data.DataStore.DataStoreErrorCode.CannotImportNullDataset
1971

Borland.Data.DataStore.DataStoreErrorCode.CannotLockDir
1971

Borland.Data.DataStore.DataStoreErrorCode.CannotMixNamedParameters
1972

Borland.Data.DataStore.DataStoreErrorCode.CannotOpen 1972

Borland.Data.DataStore.DataStoreErrorCode.CannotOrphanDetails
1972

Borland.Data.DataStore.DataStoreErrorCode.CannotRefresh
1972

Borland.Data.DataStore.DataStoreErrorCode.CannotRestructure
1972

Borland.Data.DataStore.DataStoreErrorCode.CannotSaveChanges
1973

Borland.Data.DataStore.DataStoreErrorCode.CannotStartServer

1973

Borland.Data.DataStore.DataStoreErrorCode.CannotUpdateScopedDataRow
1973

Borland.Data.DataStore.DataStoreErrorCode.CannotUseInternalrow
1973

Borland.Data.DataStore.DataStoreErrorCode.CannotUseViewHere
1973

Borland.Data.DataStore.DataStoreErrorCode.CantCreateOpenFile
1974

Borland.Data.DataStore.DataStoreErrorCode.CantCreateOpenStream
1974

Borland.Data.DataStore.DataStoreErrorCode.ClassNotFoundError
1974

Borland.Data.DataStore.DataStoreErrorCode.ColumnAlreadyBound
1974

Borland.Data.DataStore.DataStoreErrorCode.ColumnIsReadonly
1974

Borland.Data.DataStore.DataStoreErrorCode.ColumnNeedsRestructure
1975

Borland.Data.DataStore.DataStoreErrorCode.ColumnNotInLeftRef
1975

Borland.Data.DataStore.DataStoreErrorCode.ColumnNotInRightRef
1975

Borland.Data.DataStore.DataStoreErrorCode.ColumnSpecifiedTwice
1975

Borland.Data.DataStore.DataStoreErrorCode.ColumnTypeConflict
1975

Borland.Data.DataStore.DataStoreErrorCode.ConflictingMethodDef
1976

Borland.Data.DataStore.DataStoreErrorCode.ConnectionClosed

1976

Borland.Data.DataStore.DataStoreErrorCode.ConnectionCountExceeded
1976

Borland.Data.DataStore.DataStoreErrorCode.ConnectionDescriptorNotSet
1976

Borland.Data.DataStore.DataStoreErrorCode.ConnectionError
1976

Borland.Data.DataStore.DataStoreErrorCode.ConnectionNotClosed
1977

Borland.Data.DataStore.DataStoreErrorCode.ConnectionPoolClosed
1977

Borland.Data.DataStore.DataStoreErrorCode.ConnectionTimeout
1977

Borland.Data.DataStore.DataStoreErrorCode.ConstraintNameExists
1977

Borland.Data.DataStore.DataStoreErrorCode.CreateErrorMethodExists

1977

Borland.Data.DataStore.DataStoreErrorCode.CyclesInViewDefinition
1978

Borland.Data.DataStore.DataStoreErrorCode.DataFileLoadFailed
1978

Borland.Data.DataStore.DataStoreErrorCode.DataHasDuplicates
1978

Borland.Data.DataStore.DataStoreErrorCode.DatasetAlreadyOpen
1978

Borland.Data.DataStore.DataStoreErrorCode.DatasetCorrupt
1978

Borland.Data.DataStore.DataStoreErrorCode.DatasetExists
1979

Borland.Data.DataStore.DataStoreErrorCode.DatasetHasNoRows
1979

Borland.Data.DataStore.DataStoreErrorCode.DatasetHasNoTables
1979

Borland.Data.DataStore.DataStoreErrorCode.DatasetNotOpen
1979

Borland.Data.DataStore.DataStoreErrorCode.DatasetOpen
1979

Borland.Data.DataStore.DataStoreErrorCode.DatastoreAlreadyOpen
1980

Borland.Data.DataStore.DataStoreErrorCode.DatastoreExists
1980

Borland.Data.DataStore.DataStoreErrorCode.DatastoreInvalid
1980

Borland.Data.DataStore.DataStoreErrorCode.DatastoreIsUnlicensed
1980

Borland.Data.DataStore.DataStoreErrorCode.DatastoreNotFound
1980

Borland.Data.DataStore.DataStoreErrorCode.DatastoreNotOpen

1981

Borland.Data.DataStore.DataStoreErrorCode.DatastoreOpen
1981

Borland.Data.DataStore.DataStoreErrorCode.DbAdminError
1981

Borland.Data.DataStore.DataStoreErrorCode.Deadlock 1981

Borland.Data.DataStore.DataStoreErrorCode.DeleteDuplicates
1981

Borland.Data.DataStore.DataStoreErrorCode.DeleteFailed 1982

Borland.Data.DataStore.DataStoreErrorCode.DeleteNotAllowed
1982

Borland.Data.DataStore.DataStoreErrorCode.DesktopLicense
1982

Borland.Data.DataStore.DataStoreErrorCode.DifferentDegree
1982

Borland.Data.DataStore.DataStoreErrorCode.DifferentSymbol
1982

Borland.Data.DataStore.DataStoreErrorCode.DivideByZero
1983

Borland.Data.DataStore.DataStoreErrorCode.DriverNotLoadedAtRuntime
1983

Borland.Data.DataStore.DataStoreErrorCode.DriverNotLoadedInDesign
1983

Borland.Data.DataStore.DataStoreErrorCode.DropErrorDependentView
1983

Borland.Data.DataStore.DataStoreErrorCode.DropErrorForeignKey
1983

Borland.Data.DataStore.DataStoreErrorCode.DropErrorRoleGrant
1984

Borland.Data.DataStore.DataStoreErrorCode.DropSchemaError
1984

Borland.Data.DataStore.DataStoreErrorCode.DuplicateColumnName
1984

Borland.Data.DataStore.DataStoreErrorCode.DuplicateKey
1984

Borland.Data.DataStore.DataStoreErrorCode.DuplicatePrimary
1984

Borland.Data.DataStore.DataStoreErrorCode.EmptyColumnNames
1985

Borland.Data.DataStore.DataStoreErrorCode.EofExpected 1985

Borland.Data.DataStore.DataStoreErrorCode.ExceptionChain
1985

Borland.Data.DataStore.DataStoreErrorCode.FailoverError 1985

Borland.Data.DataStore.DataStoreErrorCode.FeatureNotSupp
1985

Borland.Data.DataStore.DataStoreErrorCode.FieldPostError
1986

Borland.Data.DataStore.DataStoreErrorCode.FileExists 1986

Borland.Data.DataStore.DataStoreErrorCode.FileNotFound

1986

Borland.Data.DataStore.DataStoreErrorCode.FloatPrecision
1986

Borland.Data.DataStore.DataStoreErrorCode.ForeignKeyError
1986

Borland.Data.DataStore.DataStoreErrorCode.ForeignKeyViolation
1987

Borland.Data.DataStore.DataStoreErrorCode.ForwardOnly 1987

Borland.Data.DataStore.DataStoreErrorCode.GenericError 1987

Borland.Data.DataStore.DataStoreErrorCode.GreaterThanMax
1987

Borland.Data.DataStore.DataStoreErrorCode.HasResultset
1987

Borland.Data.DataStore.DataStoreErrorCode.IllegalCatalog
1988

Borland.Data.DataStore.DataStoreErrorCode.IllegalCharacter
1988

Borland.Data.DataStore.DataStoreErrorCode.IllegalExprStart
1988

Borland.Data.DataStore.DataStoreErrorCode.IllegalMethodCall
1988

Borland.Data.DataStore.DataStoreErrorCode.IncompatibleBlockSize
1988

Borland.Data.DataStore.DataStoreErrorCode.IncompatibleDataRow
1989

Borland.Data.DataStore.DataStoreErrorCode.IncompatibleTypes
1989

Borland.Data.DataStore.DataStoreErrorCode.IncompatibleTypesForOp
1989

Borland.Data.DataStore.DataStoreErrorCode.IndexAlreadyExists
1989

Borland.Data.DataStore.DataStoreErrorCode.InlineTooSmall
1989

Borland.Data.DataStore.DataStoreErrorCode.InsertFailed 1990

Borland.Data.DataStore.DataStoreErrorCode.InsertNotAllowed
1990

Borland.Data.DataStore.DataStoreErrorCode.InsufficientRowid
1990

Borland.Data.DataStore.DataStoreErrorCode.IntegerRequired
1990

Borland.Data.DataStore.DataStoreErrorCode.InternalDuplicateKey
1990

Borland.Data.DataStore.DataStoreErrorCode.InternalError 1991

Borland.Data.DataStore.DataStoreErrorCode.InternalUpdateFailed
1991

Borland.Data.DataStore.DataStoreErrorCode.InvalidAggDescriptor
1991

Borland.Data.DataStore.DataStoreErrorCode.InvalidAggExpr
1991

Borland.Data.DataStore.DataStoreErrorCode.InvalidAnchorFile
1991

Borland.Data.DataStore.DataStoreErrorCode.InvalidAnchorLength
1992

Borland.Data.DataStore.DataStoreErrorCode.InvalidBinaryString
1992

Borland.Data.DataStore.DataStoreErrorCode.InvalidClass 1992

Borland.Data.DataStore.DataStoreErrorCode.InvalidColumnFormat
1992

Borland.Data.DataStore.DataStoreErrorCode.InvalidColumnname
1992

Borland.Data.DataStore.DataStoreErrorCode.InvalidColumnPosition
1993

Borland.Data.DataStore.DataStoreErrorCode.InvalidColumnRef
1993

Borland.Data.DataStore.DataStoreErrorCode.InvalidColumnType
1993

Borland.Data.DataStore.DataStoreErrorCode.InvalidColumnValue
1993

Borland.Data.DataStore.DataStoreErrorCode.InvalidConstraintName
1993

Borland.Data.DataStore.DataStoreErrorCode.InvalidDataFileFormat
1994

Borland.Data.DataStore.DataStoreErrorCode.InvalidDirectoryAttributes
1994

Borland.Data.DataStore.DataStoreErrorCode.InvalidEscapeCharInLike
1994

Borland.Data.DataStore.DataStoreErrorCode.InvalidExtractField

1994

Borland.Data.DataStore.DataStoreErrorCode.InvalidFormat 1994

Borland.Data.DataStore.DataStoreErrorCode.InvalidIndexname 1995

Borland.Data.DataStore.DataStoreErrorCode.InvalidIteratorUse 1995

Borland.Data.DataStore.DataStoreErrorCode.InvalidJoin 1995

Borland.Data.DataStore.DataStoreErrorCode.InvalidLikePattern 1995

Borland.Data.DataStore.DataStoreErrorCode.InvalidLsn 1995

Borland.Data.DataStore.DataStoreErrorCode.InvalidPattern 1996

Borland.Data.DataStore.DataStoreErrorCode.InvalidPrecision 1996

Borland.Data.DataStore.DataStoreErrorCode.InvalidRowValues 1996

Borland.Data.DataStore.DataStoreErrorCode.InvalidSavepoint 1996

Borland.Data.DataStore.DataStoreErrorCode.InvalidSchemaFile 1996

Borland.Data.DataStore.DataStoreErrorCode.InvalidSort 1997

Borland.Data.DataStore.DataStoreErrorCode.InvalidSortAsInserted 1997

Borland.Data.DataStore.DataStoreErrorCode.InvalidSortColumn 1997

Borland.Data.DataStore.DataStoreErrorCode.InvalidStoreClass 1997

Borland.Data.DataStore.DataStoreErrorCode.InvalidStoreName 1997

Borland.Data.DataStore.DataStoreErrorCode.InvalidTablename 1998

Borland.Data.DataStore.DataStoreErrorCode.InvalidTableRef 1998

Borland.Data.DataStore.DataStoreErrorCode.InvalidTrimChar 1998

Borland.Data.DataStore.DataStoreErrorCode.InvalidUserName 1998

Borland.Data.DataStore.DataStoreErrorCode.InvalidViewDefinition 1998

Borland.Data.DataStore.DataStoreErrorCode.IoError 1999

Borland.Data.DataStore.DataStoreErrorCode.IoNetError 1999

Borland.Data.DataStore.DataStoreErrorCode.IoProtocolError 1999

Borland.Data.DataStore.DataStoreErrorCode.IsInsertRow 1999

Borland.Data.DataStore.DataStoreErrorCode.LessThanMin 1999

Borland.Data.DataStore.DataStoreErrorCode.LinkColumnsError 2000

Borland.Data.DataStore.DataStoreErrorCode.LinkfieldInUserparameters 2000

Borland.Data.DataStore.DataStoreErrorCode.LoadingNotStarted 2000

Borland.Data.DataStore.DataStoreErrorCode.LockTimeout 2000

Borland.Data.DataStore.DataStoreErrorCode.LogdirMismatch 2000

Borland.Data.DataStore.DataStoreErrorCode.LogExists 2001

Borland.Data.DataStore.DataStoreErrorCode.MalformedDateOrTime 2001

Borland.Data.DataStore.DataStoreErrorCode.MalformedDecimal 2001

Borland.Data.DataStore.DataStoreErrorCode.MalformedFloating 2001

Borland.Data.DataStore.DataStoreErrorCode.MasterDetailViewError 2001

Borland.Data.DataStore.DataStoreErrorCode.MasterNavigationError 2002

Borland.Data.DataStore.DataStoreErrorCode.MaxResolveFailed 2002

Borland.Data.DataStore.DataStoreErrorCode.MethodArgMismatch 2002

Borland.Data.DataStore.DataStoreErrorCode.MethodClassMissing 2002

Borland.Data.DataStore.DataStoreErrorCode.MethodNameMissing 2002

Borland.Data.DataStore.DataStoreErrorCode.MethodNonexistent 2003

Borland.Data.DataStore.DataStoreErrorCode.MethodNotFound 2003

Borland.Data.DataStore.DataStoreErrorCode.MirrorError 2003

Borland.Data.DataStore.DataStoreErrorCode.MismatchColumnlistCount
2003

Borland.Data.DataStore.DataStoreErrorCode.MismatchedParameterFormat
2003

Borland.Data.DataStore.DataStoreErrorCode.MismatchInsertColumnCount
2004

Borland.Data.DataStore.DataStoreErrorCode.MismatchIntoParamCount
2004

Borland.Data.DataStore.DataStoreErrorCode.MismatchParamResult
2004

Borland.Data.DataStore.DataStoreErrorCode.MissingCheckpointFile
2004

Borland.Data.DataStore.DataStoreErrorCode.MissingLogFile
2004

Borland.Data.DataStore.DataStoreErrorCode.MissingMasterDataset
2005

Borland.Data.DataStore.DataStoreErrorCode.MissingReplacestorerow
2005

Borland.Data.DataStore.DataStoreErrorCode.MissingResolver
2005

Borland.Data.DataStore.DataStoreErrorCode.MoreThanOneRow
2005

Borland.Data.DataStore.DataStoreErrorCode.MultipleRowsAffected
2005

Borland.Data.DataStore.DataStoreErrorCode.MustBeSingleColumn
2006

Borland.Data.DataStore.DataStoreErrorCode.NameNotUnique
2006

Borland.Data.DataStore.DataStoreErrorCode.NeedALogdir 2006

Borland.Data.DataStore.DataStoreErrorCode.NeedLocateStartOption
2006

Borland.Data.DataStore.DataStoreErrorCode.NeedProcedureprovider
2006

Borland.Data.DataStore.DataStoreErrorCode.NeedQueryprovider
2007

Borland.Data.DataStore.DataStoreErrorCode.NeedsRecalc
2007

Borland.Data.DataStore.DataStoreErrorCode.NeedStoragedataset
2007

Borland.Data.DataStore.DataStoreErrorCode.NegativeSqrtArgument
2007

Borland.Data.DataStore.DataStoreErrorCode.NegativeSubstringLength
2007

Borland.Data.DataStore.DataStoreErrorCode.NestedAggregators
2008

Borland.Data.DataStore.DataStoreErrorCode.NewerVersion
2008

Borland.Data.DataStore.DataStoreErrorCode.NewerVersionForOperation
2008

Borland.Data.DataStore.DataStoreErrorCode.NoCalcAggFields
2008

Borland.Data.DataStore.DataStoreErrorCode.NoCalcFields
2008

Borland.Data.DataStore.DataStoreErrorCode.NoDatabaseToResolve
2009

Borland.Data.DataStore.DataStoreErrorCode.NodefaultForBinary
2009

Borland.Data.DataStore.DataStoreErrorCode.NoGroupingColumn
2009

Borland.Data.DataStore.DataStoreErrorCode.NonExistentRowid
2009

Borland.Data.DataStore.DataStoreErrorCode.NoNonBlobColumns
2009

Borland.Data.DataStore.DataStoreErrorCode.NonPrimitiveRequired
2010

Borland.Data.DataStore.DataStoreErrorCode.NonTransactional
2010

Borland.Data.DataStore.DataStoreErrorCode.NoPrimaryKey
2010

Borland.Data.DataStore.DataStoreErrorCode.NoPriorOriginalRow
2010

Borland.Data.DataStore.DataStoreErrorCode.NoReadonly 2010

Borland.Data.DataStore.DataStoreErrorCode.NoResultSet 2011

Borland.Data.DataStore.DataStoreErrorCode.NoResultsetProduced

2011

Borland.Data.DataStore.DataStoreErrorCode.NoRowsAffected
2011

Borland.Data.DataStore.DataStoreErrorCode.NoRowsToDelete
2011

Borland.Data.DataStore.DataStoreErrorCode.NoServerFound
2011

Borland.Data.DataStore.DataStoreErrorCode.NoSuchParameter

2012

Borland.Data.DataStore.DataStoreErrorCode.NotAnOutputParameter
2012

Borland.Data.DataStore.DataStoreErrorCode.NotDatabaseResolver
2012

Borland.Data.DataStore.DataStoreErrorCode.NotImplemented
2012

Borland.Data.DataStore.DataStoreErrorCode.NotInsertRow
2012

Borland.Data.DataStore.DataStoreErrorCode.NotSelectQuery
2013

Borland.Data.DataStore.DataStoreErrorCode.NotTransactional
2013

Borland.Data.DataStore.DataStoreErrorCode.NotUpdateable
2013

Borland.Data.DataStore.DataStoreErrorCode.NoUpdatableColumns
2013

Borland.Data.DataStore.DataStoreErrorCode.NoWhereClause
2013

Borland.Data.DataStore.DataStoreErrorCode.NowReadonly
2014

Borland.Data.DataStore.DataStoreErrorCode.NullColumnName
2014

Borland.Data.DataStore.DataStoreErrorCode.NumericOverflow
2014

Borland.Data.DataStore.DataStoreErrorCode.NumericUnderflow

2014

Borland.Data.DataStore.DataStoreErrorCode.OlderVersion 2014

Borland.Data.DataStore.DataStoreErrorCode.OldLog 2015

Borland.Data.DataStore.DataStoreErrorCode.OnepassInputStream
2015

Borland.Data.DataStore.DataStoreErrorCode.OperationCanceled
2015

Borland.Data.DataStore.DataStoreErrorCode.OutParameterRequired
2015

Borland.Data.DataStore.DataStoreErrorCode.OverloadedDiffParams
2015

Borland.Data.DataStore.DataStoreErrorCode.ParameterCountMismatch
2016

Borland.Data.DataStore.DataStoreErrorCode.ParameterValueNotSet
2016

Borland.Data.DataStore.DataStoreErrorCode.PartialSearchForString
2016

Borland.Data.DataStore.DataStoreErrorCode.PhysUndoNeeded

2016

Borland.Data.DataStore.DataStoreErrorCode.PrivilegeMix 2016

Borland.Data.DataStore.DataStoreErrorCode.ProcedureFailed
2017

Borland.Data.DataStore.DataStoreErrorCode.ProcedureInProcess
2017

Borland.Data.DataStore.DataStoreErrorCode.ProtocolVersion
2017

Borland.Data.DataStore.DataStoreErrorCode.ProviderFailed
2017

Borland.Data.DataStore.DataStoreErrorCode.ProviderOwned
2017

Borland.Data.DataStore.DataStoreErrorCode.QueryFailed 2018

Borland.Data.DataStore.DataStoreErrorCode.QueryInProcess
2018

Borland.Data.DataStore.DataStoreErrorCode.ReadBlockError
2018

Borland.Data.DataStore.DataStoreErrorCode.ReadOnly 2018

Borland.Data.DataStore.DataStoreErrorCode.ReadOnlyColumn
2018

Borland.Data.DataStore.DataStoreErrorCode.ReadOnlyDataset
2019

Borland.Data.DataStore.DataStoreErrorCode.ReadOnlyStore
2019

Borland.Data.DataStore.DataStoreErrorCode.ReadonlyTimeout
2019

Borland.Data.DataStore.DataStoreErrorCode.ReadonlytxStartFailed
2019

Borland.Data.DataStore.DataStoreErrorCode.RecoveryDenied
2019

Borland.Data.DataStore.DataStoreErrorCode.RefreshrowNotSupported 2020
Borland.Data.DataStore.DataStoreErrorCode.RenamedTwice 2020
Borland.Data.DataStore.DataStoreErrorCode.ReopenFailure 2020
Borland.Data.DataStore.DataStoreErrorCode.ResolveFailed 2020
Borland.Data.DataStore.DataStoreErrorCode.ResolveInProgress 2020
Borland.Data.DataStore.DataStoreErrorCode.ResolvePartial 2021
Borland.Data.DataStore.DataStoreErrorCode.RestructureDataLoss 2021
Borland.Data.DataStore.DataStoreErrorCode.RestructureInProgress 2021
Borland.Data.DataStore.DataStoreErrorCode.RestructureParseError 2021
Borland.Data.DataStore.DataStoreErrorCode.RestructurePrecisionLoss 2021
Borland.Data.DataStore.DataStoreErrorCode.ResultNotUpdateable 2022
Borland.Data.DataStore.DataStoreErrorCode.ResultsetClosed 2022
Borland.Data.DataStore.DataStoreErrorCode.RevokeErrorDependentPrivilege 2022
Borland.Data.DataStore.DataStoreErrorCode.RevokeErrorDependentView 2022
Borland.Data.DataStore.DataStoreErrorCode.RevokeErrorForeignKey 2022
Borland.Data.DataStore.DataStoreErrorCode.RowLockTimeout 2023
Borland.Data.DataStore.DataStoreErrorCode.RowNotFound 2023
Borland.Data.DataStore.DataStoreErrorCode.RowTooWide 2023
Borland.Data.DataStore.DataStoreErrorCode.RuntimeError 2023
Borland.Data.DataStore.DataStoreErrorCode.SecurityError 2023
Borland.Data.DataStore.DataStoreErrorCode.SetCalculatedFailure 2024
Borland.Data.DataStore.DataStoreErrorCode.SingleRowHasMoreRows 2024
Borland.Data.DataStore.DataStoreErrorCode.SqlError 2024
Borland.Data.DataStore.DataStoreErrorCode.StatementClosed 2024
Borland.Data.DataStore.DataStoreErrorCode.StoreNameNotSet 2024
Borland.Data.DataStore.DataStoreErrorCode.StoreOperationUnsupported 2025
Borland.Data.DataStore.DataStoreErrorCode.StreamClosed 2025
Borland.Data.DataStore.DataStoreErrorCode.StreamNotFound 2025
Borland.Data.DataStore.DataStoreErrorCode.StreamOpenTwice 2025
Borland.Data.DataStore.DataStoreErrorCode.StringTooLong 2025
Borland.Data.DataStore.DataStoreErrorCode.SyntaxError 2026
Borland.Data.DataStore.DataStoreErrorCode.TableAlreadyExists 2026
Borland.Data.DataStore.DataStoreErrorCode.TableNonexistent 2026
Borland.Data.DataStore.DataStoreErrorCode.TooManyAutoIncrements 2026
Borland.Data.DataStore.DataStoreErrorCode.TooManyErrors 2026
Borland.Data.DataStore.DataStoreErrorCode.TooManyOpenFiles 2027
Borland.Data.DataStore.DataStoreErrorCode.Toomanyspecs 2027
Borland.Data.DataStore.DataStoreErrorCode.TransactionIsolationLevelNotSupported 2027
Borland.Data.DataStore.DataStoreErrorCode.TxAlreadyStarted 2027
Borland.Data.DataStore.DataStoreErrorCode.TxManagerInuse 2027
Borland.Data.DataStore.DataStoreErrorCode.TxNotAllowed 2028
Borland.Data.DataStore.DataStoreErrorCode.TxRequired 2028

Borland.Data.DataStore.DataStoreErrorCode.TypenameExpected 2028
Borland.Data.DataStore.DataStoreErrorCode.UnclosedComment 2028
Borland.Data.DataStore.DataStoreErrorCode.UnclosedIdentifier 2028
Borland.Data.DataStore.DataStoreErrorCode.UnclosedStringLiteral 2029
Borland.Data.DataStore.DataStoreErrorCode.UnexpectedEndOfQuery 2029
Borland.Data.DataStore.DataStoreErrorCode.UnknownColumn 2029
Borland.Data.DataStore.DataStoreErrorCode.UnknownColumnName 2029
Borland.Data.DataStore.DataStoreErrorCode.UnknownDetailName 2029
Borland.Data.DataStore.DataStoreErrorCode.UnknownParameter 2030
Borland.Data.DataStore.DataStoreErrorCode.UnknownParamName 2030
Borland.Data.DataStore.DataStoreErrorCode.UnrecognizedDataType 2030
Borland.Data.DataStore.DataStoreErrorCode.UnrecognizedEscape 2030
Borland.Data.DataStore.DataStoreErrorCode.UnsupportedColumntype 2030
Borland.Data.DataStore.DataStoreErrorCode.UnsupportedDbxType 2031
Borland.Data.DataStore.DataStoreErrorCode.UpdateFailed 2031
Borland.Data.DataStore.DataStoreErrorCode.UpdateNotAllowed 2031
Borland.Data.DataStore.DataStoreErrorCode.UrlNotFound 2031
Borland.Data.DataStore.DataStoreErrorCode.UrlNotFoundInDesign 2031
Borland.Data.DataStore.DataStoreErrorCode.ViewNotUpdatable 2032
Borland.Data.DataStore.DataStoreErrorCode.WithCheckOptionViolation 2032
Borland.Data.DataStore.DataStoreErrorCode.WriteBlockError 2032
Borland.Data.DataStore.DataStoreErrorCode.WrongCreateschemaname 2032
Borland.Data.DataStore.DataStoreErrorCode.WrongDatabase 2032
Borland.Data.DataStore.DataStoreErrorCode.WrongSchemaname 2033
Borland.Data.DataStore.DataStoreErrorCode.WrongTableType 2033
Borland.Data.DataStore.DataStoreErrorCode.XaActive 2033
Borland.Data.DataStore.DataStoreParameter 2033
Borland.Data.DataStore.DataStoreParameter.Clone 2036
Borland.Data.DataStore.DataStoreParameter.DataStoreParameter 2035
Borland.Data.DataStore.DataStoreParameter.DbType 2037
Borland.Data.DataStore.DataStoreParameter.Direction 2037
Borland.Data.DataStore.DataStoreParameter.IsNullable 2037
Borland.Data.DataStore.DataStoreParameter.JDSType 2038
Borland.Data.DataStore.DataStoreParameter.ParameterName 2038
Borland.Data.DataStore.DataStoreParameter.Precision 2038
Borland.Data.DataStore.DataStoreParameter.ResetDbType 2036
Borland.Data.DataStore.DataStoreParameter.Scale 2039
Borland.Data.DataStore.DataStoreParameter.Size 2039
Borland.Data.DataStore.DataStoreParameter.SourceColumn 2039
Borland.Data.DataStore.DataStoreParameter.SourceColumnNullMapping 2040
Borland.Data.DataStore.DataStoreParameter.SourceVersion 2040
Borland.Data.DataStore.DataStoreParameter.Value 2041
Borland.Data.DataStore.DataStoreParameter.WasPrepared 2041
Borland.Data.DataStore.DataStoreParameterCollection 2041
Borland.Data.DataStore.DataStoreParameterCollection.Add 2044

Borland.Data.DataStore.DataStoreParameterCollection.AddRange 2045
Borland.Data.DataStore.DataStoreParameterCollection.Clear 2045
Borland.Data.DataStore.DataStoreParameterCollection.Contains 2046
Borland.Data.DataStore.DataStoreParameterCollection.CopyTo 2046
Borland.Data.DataStore.DataStoreParameterCollection.Count 2051
Borland.Data.DataStore.DataStoreParameterCollection.DataStoreParameterCollection 2043
Borland.Data.DataStore.DataStoreParameterCollection.GetEnumerator 2046
Borland.Data.DataStore.DataStoreParameterCollection.GetParameter 2047
Borland.Data.DataStore.DataStoreParameterCollection.IndexOf 2048
Borland.Data.DataStore.DataStoreParameterCollection.Insert 2048
Borland.Data.DataStore.DataStoreParameterCollection.InvalidatePreparation 2048
Borland.Data.DataStore.DataStoreParameterCollection.IsFixedSize 2051
Borland.Data.DataStore.DataStoreParameterCollection.IsReadOnly 2052
Borland.Data.DataStore.DataStoreParameterCollection.IsSynchronized 2052
Borland.Data.DataStore.DataStoreParameterCollection.Remove 2049
Borland.Data.DataStore.DataStoreParameterCollection.RemoveAt 2049
Borland.Data.DataStore.DataStoreParameterCollection.SetParameter 2050
Borland.Data.DataStore.DataStoreParameterCollection.SyncRoot 2052
Borland.Data.DataStore.DataStoreParameterCollection.this 2050, 2051
Borland.Data.DataStore.DataStoreRowUpdatedEventArgs 2053
Borland.Data.DataStore.DataStoreRowUpdatedEventArgs.Command 2054
Borland.Data.DataStore.DataStoreRowUpdatedEventArgs.DataStoreRowUpdatedEventArgs 2054
Borland.Data.DataStore.DataStoreRowUpdatedEventHandler 2129
Borland.Data.DataStore.DataStoreRowUpdatedEventHandler type 2129
Borland.Data.DataStore.DataStoreRowUpdatingEventArgs 2054
Borland.Data.DataStore.DataStoreRowUpdatingEventArgs.Command 2055
Borland.Data.DataStore.DataStoreRowUpdatingEventArgs.DataStoreRowUpdatingEventArgs 2055
Borland.Data.DataStore.DataStoreRowUpdatingEventHandler 2129
Borland.Data.DataStore.DataStoreRowUpdatingEventHandler type 2129
Borland.Data.DataStore.DataStoreTransaction 2056
Borland.Data.DataStore.DataStoreTransaction.Commit 2057
Borland.Data.DataStore.DataStoreTransaction.Connection 2058
Borland.Data.DataStore.DataStoreTransaction.DataStoreTransaction 2056
Borland.Data.DataStore.DataStoreTransaction.DbConnection 2058
Borland.Data.DataStore.DataStoreTransaction.Dispose 2057
Borland.Data.DataStore.DataStoreTransaction.IsolationLevel 2058
Borland.Data.DataStore.DataStoreTransaction.Rollback 2058
Borland.Data.DataStore.LicenseColumns 2059
Borland.Data.DataStore.LicenseColumns.CacheSizeLimit 2060
Borland.Data.DataStore.LicenseColumns.DeploymentLimit 2060
Borland.Data.DataStore.LicenseColumns.Desktop 2060
Borland.Data.DataStore.LicenseColumns.Expired 2061
Borland.Data.DataStore.LicenseColumns.FileName 2061
Borland.Data.DataStore.LicenseColumns.FileSizeLimit 2061
Borland.Data.DataStore.LicenseColumns.ProductId 2061
Borland.Data.DataStore.LicenseColumns.SerialNumber 2061
Borland.Data.DataStore.LicenseColumns.SkuId 2062
Borland.Data.DataStore.LicenseColumns.UserLimit 2062

Borland.Data.DataStore.LockColumns 2062
Borland.Data.DataStore.LockColumns.CompMask 2063
Borland.Data.DataStore.LockColumns.Granted 2064
Borland.Data.DataStore.LockColumns.LockMask 2064
Borland.Data.DataStore.LockColumns.Name 2064
Borland.Data.DataStore.LockColumns.Row 2064
Borland.Data.DataStore.LockColumns.RowLock 2064
Borland.Data.DataStore.LockColumns.Shared 2065
Borland.Data.DataStore.LockColumns.TableLock 2065
Borland.Data.DataStore.LockColumns.Type 2065
Borland.Data.DataStore.LockColumns.User 2065
Borland.Data.DataStore.LogFilter 2065
Borland.Data.DataStore.LogFilter.Connect 2067
Borland.Data.DataStore.LogFilter.Errors 2067
Borland.Data.DataStore.LogFilter.Execute 2067
Borland.Data.DataStore.LogFilter.Init 2068
Borland.Data.DataStore.LogFilter.LockErrors 2068
Borland.Data.DataStore.LogFilter.Mirror 2068
Borland.Data.DataStore.LogFilter.Prepare 2068
Borland.Data.DataStore.LogFilter.SetParameter 2068
Borland.Data.DataStore.LogFilter.System 2069
Borland.Data.DataStore.LogFilter.Transactions 2069
Borland.Data.DataStore.LogFilter.Xa 2069
Borland.Data.DataStore.MirrorStatusColumns 2069
Borland.Data.DataStore.MirrorStatusColumns.AttachedAutoFailOverCount 2071
Borland.Data.DataStore.MirrorStatusColumns.AttachedInstantSynchCount 2071
Borland.Data.DataStore.MirrorStatusColumns.ConnectError 2071
Borland.Data.DataStore.MirrorStatusColumns.HighestRedoStamp 2071
Borland.Data.DataStore.MirrorStatusColumns.MirrorReplayStamp 2072
Borland.Data.DataStore.MirrorStatusColumns.TotalAutoFailOverCount 2072
Borland.Data.DataStore.MirrorStatusColumns.TotalInstantSynchCount 2072
Borland.Data.DataStore.MirrorTypes 2072
Borland.Data.DataStore.MirrorTypes.Directory 2073
Borland.Data.DataStore.MirrorTypes.None 2074
Borland.Data.DataStore.MirrorTypes.Primary 2074
Borland.Data.DataStore.MirrorTypes.Readonly 2074
Borland.Data.DataStore.ProtocolValue 2074
Borland.Data.DataStore.ProtocolValue.InProcess 2075
Borland.Data.DataStore.ProtocolValue.Tcp 2075
Borland.Data.DataStore.ScheduleColumns 2076
Borland.Data.DataStore.ScheduleColumns.Day 2077
Borland.Data.DataStore.ScheduleColumns.JavaMethod 2077
Borland.Data.DataStore.ScheduleColumns.Millis 2077
Borland.Data.DataStore.ScheduleColumns.Period 2077
Borland.Data.DataStore.ScheduleColumns.Time 2078
Borland.Data.DataStore.SchedulePeriod 2078
Borland.Data.DataStore.SchedulePeriod.Day 2079
Borland.Data.DataStore.SchedulePeriod.Millis 2079
Borland.Data.DataStore.SchedulePeriod.Week 2079
Borland.Data.DataStore.SqlStateCode 2079
Borland.Data.DataStore.SqlStateCode.Sql02000 2082
Borland.Data.DataStore.SqlStateCode.Sql08001 2082
Borland.Data.DataStore.SqlStateCode.Sql08003 2082
Borland.Data.DataStore.SqlStateCode.Sql08004 2083
Borland.Data.DataStore.SqlStateCode.Sql08006 2083
Borland.Data.DataStore.SqlStateCode.Sql0A000 2083
Borland.Data.DataStore.SqlStateCode.Sql21000 2083
Borland.Data.DataStore.SqlStateCode.Sql22000 2083
Borland.Data.DataStore.SqlStateCode.Sql22001 2084
Borland.Data.DataStore.SqlStateCode.Sql22003 2084
Borland.Data.DataStore.SqlStateCode.Sql22005 2084
Borland.Data.DataStore.SqlStateCode.Sql22007 2084
Borland.Data.DataStore.SqlStateCode.Sql22011 2084
Borland.Data.DataStore.SqlStateCode.Sql22012 2085
Borland.Data.DataStore.SqlStateCode.Sql22019 2085
Borland.Data.DataStore.SqlStateCode.Sql22023 2085
Borland.Data.DataStore.SqlStateCode.Sql22025 2085
Borland.Data.DataStore.SqlStateCode.Sql22027 2085
Borland.Data.DataStore.SqlStateCode.Sql23000 2086
Borland.Data.DataStore.SqlStateCode.Sql24000 2086

Borland.Data.DataStore.SqlStateCode.Sql42000 2086
Borland.Data.DataStore.SqlStateCode.Sql44000 2086
Borland.Data.DataStore.SqlStateCode.SqlHz309 2086
Borland.Data.DataStore.SqlStateCode.SqlHz316 2087
Borland.Data.DataStore.SqlStateCode.SqlHz320 2087
Borland.Data.DataStore.SqlStateCode.SqlIm000 2087
Borland.Data.DataStore.StatusLogColumns 2087
Borland.Data.DataStore.StatusLogColumns.Id 2088
Borland.Data.DataStore.StatusLogColumns.LastModified 2088
Borland.Data.DataStore.StatusLogColumns.Length 2089
Borland.Data.DataStore.StatusLogColumns.Name 2089
Borland.Data.DataStore.SysMirrors 2089
Borland.Data.DataStore.SysMirrors.AutoFailOver 2092
Borland.Data.DataStore.SysMirrors.BirthStamp 2092
Borland.Data.DataStore.SysMirrors.ClientHashCode 2092
Borland.Data.DataStore.SysMirrors.FailOverPriority 2092
Borland.Data.DataStore.SysMirrors.FileName 2093
Borland.Data.DataStore.SysMirrors.Host 2093
Borland.Data.DataStore.SysMirrors.Id 2093
Borland.Data.DataStore.SysMirrors.InstantSynch 2093
Borland.Data.DataStore.SysMirrors.LastKnownReplay 2093
Borland.Data.DataStore.SysMirrors.Name 2094
Borland.Data.DataStore.SysMirrors.Port 2094
Borland.Data.DataStore.SysMirrors.SynchMirrorName 2094
Borland.Data.DataStore.SysMirrors.Type 2094
Borland.Data.DataStore.SysMirrors.ValidatedPrimary 2094
Borland.Data.DataStore.SysSchedule 2095
Borland.Data.DataStore.SysSchedule.Day 2096
Borland.Data.DataStore.SysSchedule.Id 2096
Borland.Data.DataStore.SysSchedule.Millis 2096
Borland.Data.DataStore.SysSchedule.Period 2096
Borland.Data.DataStore.SysSchedule.Ref 2097
Borland.Data.DataStore.SysSchedule.ScheduleItemClass 2097
Borland.Data.DataStore.SysSchedule.Time 2097
Borland.Data.DataStore.SystemProperties 2097
Borland.Data.DataStore.SystemProperties.AssemblyPath 2099
Borland.Data.DataStore.SystemProperties.BlackfishSqlNamespace 2100
Borland.Data.DataStore.SystemProperties.DataDirectory 2100
Borland.Data.DataStore.SystemProperties.LicenseDirectory 2100
Borland.Data.DataStore.SystemProperties.LogFile 2100
Borland.Data.DataStore.SystemProperties.LogFilters 2100
Borland.Data.DataStore.SystemProperties.MaxSortBuffer 2101
Borland.Data.DataStore.SystemProperties.MinCacheSize 2101
Borland.Data.DataStore.SystemProperties.Port 2101
Borland.Data.DataStore.SystemProperties.TempDir 2101
Borland.Data.DataStore.TriggerContext 2101
Borland.Data.DataStore.TriggerContext.Connection 2103
Borland.Data.DataStore.TriggerContext.method 2102
Borland.Data.DataStore.TriggerContext.NewRow 2103
Borland.Data.DataStore.TriggerContext.OldRow 2103
Borland.Data.DataStore.TriggerReadRow 2103
Borland.Data.DataStore.TriggerReadRow.FieldCount 2117
Borland.Data.DataStore.TriggerReadRow.FindModified 2105
Borland.Data.DataStore.TriggerReadRow.GetArrayLength 2106
Borland.Data.DataStore.TriggerReadRow.GetBoolean 2106
Borland.Data.DataStore.TriggerReadRow.GetByte 2107
Borland.Data.DataStore.TriggerReadRow.GetBytes 2107
Borland.Data.DataStore.TriggerReadRow.GetColumnNames 2108
Borland.Data.DataStore.TriggerReadRow.GetDateTime 2108
Borland.Data.DataStore.TriggerReadRow.GetDecimal 2109
Borland.Data.DataStore.TriggerReadRow.GetDouble 2109
Borland.Data.DataStore.TriggerReadRow.GetFloat 2110
Borland.Data.DataStore.TriggerReadRow.GetInt16 2111
Borland.Data.DataStore.TriggerReadRow.GetInt32 2111
Borland.Data.DataStore.TriggerReadRow.GetInt64 2112
Borland.Data.DataStore.TriggerReadRow.GetObject 2113
Borland.Data.DataStore.TriggerReadRow.GetOrdinal 2113
Borland.Data.DataStore.TriggerReadRow.GetString 2113
Borland.Data.DataStore.TriggerReadRow.IsAssignedNull 2114
Borland.Data.DataStore.TriggerReadRow.IsModified 2115
Borland.Data.DataStore.TriggerReadRow.IsNull 2115
Borland.Data.DataStore.TriggerReadRow.IsUnassignedNull 2116
Borland.Data.DataStore.TriggerReadRow.method 2105
Borland.Data.DataStore.TriggerReadRow.ToString 2116
Borland.Data.DataStore.TriggerReadWriteRow 2117

Borland.Data.DataStore.TriggerReadWriteRow.ClearValues 2119
Borland.Data.DataStore.TriggerReadWriteRow.SetAssignedNull 2119
Borland.Data.DataStore.TriggerReadWriteRow.SetBoolean 2120
Borland.Data.DataStore.TriggerReadWriteRow.SetByte 2121
Borland.Data.DataStore.TriggerReadWriteRow.SetBytes 2121
Borland.Data.DataStore.TriggerReadWriteRow.SetDateTime 2122
Borland.Data.DataStore.TriggerReadWriteRow.SetDecimal 2122
Borland.Data.DataStore.TriggerReadWriteRow.SetDefault 2123
Borland.Data.DataStore.TriggerReadWriteRow.SetDefaultValues 2123
Borland.Data.DataStore.TriggerReadWriteRow.SetDouble 2124
Borland.Data.DataStore.TriggerReadWriteRow.SetFloat 2125
Borland.Data.DataStore.TriggerReadWriteRow.SetInt16 2125
Borland.Data.DataStore.TriggerReadWriteRow.SetInt32 2126
Borland.Data.DataStore.TriggerReadWriteRow.SetInt64 2127
Borland.Data.DataStore.TriggerReadWriteRow.SetObject 2127
Borland.Data.DataStore.TriggerReadWriteRow.SetString 2128
Borland.Data.DataStore.TriggerReadWriteRow.SetUnassignedNull 2129
Borland.Data.MetaData 2130
Borland.Data.MetaData namespace 2130
Classes 2131
Borland.Data.MetaData.CatalogsColumns 2132
Borland.Data.MetaData.CatalogsColumns.CatalogName 2133
Borland.Data.MetaData.ColumnConstraintsColumns 2133
Borland.Data.MetaData.ColumnConstraintsColumns.CatalogName 2134
Borland.Data.MetaData.ColumnConstraintsColumns.ColumnName 2134
Borland.Data.MetaData.ColumnConstraintsColumns.ConstraintName 2134
Borland.Data.MetaData.ColumnConstraintsColumns.SchemaName 2134
Borland.Data.MetaData.ColumnConstraintsColumns.TableName 2134
Borland.Data.MetaData.ColumnsColumns 2135
Borland.Data.MetaData.ColumnsColumns.CatalogName 2137
Borland.Data.MetaData.ColumnsColumns.ColumnName 2137
Borland.Data.MetaData.ColumnsColumns.DbxDataType 2137
Borland.Data.MetaData.ColumnsColumns.DefaultValue 2138
Borland.Data.MetaData.ColumnsColumns.IsAutoIncrement 2138
Borland.Data.MetaData.ColumnsColumns.IsFixedLength 2138
Borland.Data.MetaData.ColumnsColumns.IsLong 2138
Borland.Data.MetaData.ColumnsColumns.IsNullable 2138
Borland.Data.MetaData.ColumnsColumns.IsUnicode 2139
Borland.Data.MetaData.ColumnsColumns.IsUnsigned 2139
Borland.Data.MetaData.ColumnsColumns.MaxInline 2139
Borland.Data.MetaData.ColumnsColumns.Ordinal 2139
Borland.Data.MetaData.ColumnsColumns.Precision 2139
Borland.Data.MetaData.ColumnsColumns.Scale 2140
Borland.Data.MetaData.ColumnsColumns.SchemaName 2140
Borland.Data.MetaData.ColumnsColumns.TableName 2140
Borland.Data.MetaData.ColumnsColumns.TypeName 2140
Borland.Data.MetaData.DataTypesColumns 2140
Borland.Data.MetaData.DataTypesColumns.ColumnSize 2143
Borland.Data.MetaData.DataTypesColumns.CreateFormat 2144
Borland.Data.MetaData.DataTypesColumns.CreateParameters 2144
Borland.Data.MetaData.DataTypesColumns.DataType 2144
Borland.Data.MetaData.DataTypesColumns.DbxDataType 2144
Borland.Data.MetaData.DataTypesColumns.IsAutoIncrementable 2144
Borland.Data.MetaData.DataTypesColumns.IsBestMatch 2145
Borland.Data.MetaData.DataTypesColumns.IsCaseSensitive 2145
Borland.Data.MetaData.DataTypesColumns.IsConcurrencyType 2145
Borland.Data.MetaData.DataTypesColumns.IsFixedLength 2145
Borland.Data.MetaData.DataTypesColumns.IsFixedPrecisionScale 2145
Borland.Data.MetaData.DataTypesColumns.IsLiteralSupported 2146

Borland.Data.MetaData.DataTypesColumns.IsLong 2146
Borland.Data.MetaData.DataTypesColumns.IsNullable 2146
Borland.Data.MetaData.DataTypesColumns.IsSearchable 2146
Borland.Data.MetaData.DataTypesColumns.IsSearchableWithLike 2146
Borland.Data.MetaData.DataTypesColumns.IsUnicode 2147
Borland.Data.MetaData.DataTypesColumns.IsUnsigned 2147
Borland.Data.MetaData.DataTypesColumns.LiteralPrefix 2147
Borland.Data.MetaData.DataTypesColumns.LiteralSuffix 2147
Borland.Data.MetaData.DataTypesColumns.MaximumScale 2147
Borland.Data.MetaData.DataTypesColumns.MaximumVersion 2148
Borland.Data.MetaData.DataTypesColumns.MinimumScale 2148
Borland.Data.MetaData.DataTypesColumns.MinimumVersion 2148
Borland.Data.MetaData.DataTypesColumns.ProviderDbType 2148
Borland.Data.MetaData.DataTypesColumns.TypeName 2148
Borland.Data.MetaData.ForeignKeyColumnsColumns 2149
Borland.Data.MetaData.ForeignKeyColumnsColumns.CatalogName 2150
Borland.Data.MetaData.ForeignKeyColumnsColumns.ColumnName 2151
Borland.Data.MetaData.ForeignKeyColumnsColumns.ForeignKeyName 2151
Borland.Data.MetaData.ForeignKeyColumnsColumns.Ordinal 2151
Borland.Data.MetaData.ForeignKeyColumnsColumns.PrimaryCatalogName 2151
Borland.Data.MetaData.ForeignKeyColumnsColumns.PrimaryColumnName 2151
Borland.Data.MetaData.ForeignKeyColumnsColumns.PrimaryKeyName 2152
Borland.Data.MetaData.ForeignKeyColumnsColumns.PrimarySchemaName 2152
Borland.Data.MetaData.ForeignKeyColumnsColumns.PrimaryTableName 2152
Borland.Data.MetaData.ForeignKeyColumnsColumns.SchemaName 2152
Borland.Data.MetaData.ForeignKeyColumnsColumns.TableName 2152
Borland.Data.MetaData.ForeignKeysColumns 2153
Borland.Data.MetaData.ForeignKeysColumns.CatalogName 2154
Borland.Data.MetaData.ForeignKeysColumns.ForeignKeyName 2154
Borland.Data.MetaData.ForeignKeysColumns.SchemaName 2154
Borland.Data.MetaData.ForeignKeysColumns.TableName 2154
Borland.Data.MetaData.IndexColumnsColumns 2154
Borland.Data.MetaData.IndexColumnsColumns.CatalogName 2156
Borland.Data.MetaData.IndexColumnsColumns.ColumnName 2156
Borland.Data.MetaData.IndexColumnsColumns.IndexName 2156
Borland.Data.MetaData.IndexColumnsColumns.IsAscending 2157
Borland.Data.MetaData.IndexColumnsColumns.Ordinal 2157
Borland.Data.MetaData.IndexColumnsColumns.SchemaName 2157
Borland.Data.MetaData.IndexColumnsColumns.TableName 2157
Borland.Data.MetaData.IndexesColumns 2157
Borland.Data.MetaData.IndexesColumns.CatalogName 2159
Borland.Data.MetaData.IndexesColumns.ConstraintName 2159
Borland.Data.MetaData.IndexesColumns.IndexName 2159
Borland.Data.MetaData.IndexesColumns.IsAscending 2159
Borland.Data.MetaData.IndexesColumns.IsPrimary 2160
Borland.Data.MetaData.IndexesColumns.IsUnique 2160
Borland.Data.MetaData.IndexesColumns.SchemaName 2160
Borland.Data.MetaData.IndexesColumns.TableName 2160
Borland.Data.MetaData.MetaDataCollectionName 2160
Borland.Data.MetaData.MetaDataCollectionName.Catalogs 2163
Borland.Data.MetaData.MetaDataCollectionName.ColumnConstraints 2163
Borland.Data.MetaData.MetaDataCollectionName.Columns 2163

Borland.Data.MetaData.MetaDataCollectionName.DataTypes 2164
Borland.Data.MetaData.MetaDataCollectionName.ForeignKeyColumns 2165
Borland.Data.MetaData.MetaDataCollectionName.ForeignKeys 2166
Borland.Data.MetaData.MetaDataCollectionName.IndexColumns 2167
Borland.Data.MetaData.MetaDataCollectionName.Indexes 2167
Borland.Data.MetaData.MetaDataCollectionName.PackageProcedureParameters 2168
Borland.Data.MetaData.MetaDataCollectionName.PackageProcedures 2168
Borland.Data.MetaData.MetaDataCollectionName.Packages 2169
Borland.Data.MetaData.MetaDataCollectionName.PackageSources 2169
Borland.Data.MetaData.MetaDataCollectionName.ProcedureParameters 2170
Borland.Data.MetaData.MetaDataCollectionName.Procedures 2171
Borland.Data.MetaData.MetaDataCollectionName.ProcedureSources 2171
Borland.Data.MetaData.MetaDataCollectionName.ReservedWords 2172
Borland.Data.MetaData.MetaDataCollectionName.Roles 2172
Borland.Data.MetaData.MetaDataCollectionName.Schemas 2172
Borland.Data.MetaData.MetaDataCollectionName.Synonyms 2173
Borland.Data.MetaData.MetaDataCollectionName.Tables 2173
Borland.Data.MetaData.MetaDataCollectionName.Users 2174
Borland.Data.MetaData.MetaDataCollectionName.Views 2174
Borland.Data.MetaData.ProcedureParametersColumns 2175
Borland.Data.MetaData.ProcedureParametersColumns.CatalogName 2177
Borland.Data.MetaData.ProcedureParametersColumns.DbxDataType 2177
Borland.Data.MetaData.ProcedureParametersColumns.IsFixedLength 2177
Borland.Data.MetaData.ProcedureParametersColumns.IsLong 2177
Borland.Data.MetaData.ProcedureParametersColumns.IsNullable 2178
Borland.Data.MetaData.ProcedureParametersColumns.IsUnicode 2178
Borland.Data.MetaData.ProcedureParametersColumns.IsUnsigned 2178
Borland.Data.MetaData.ProcedureParametersColumns.Ordinal 2178
Borland.Data.MetaData.ProcedureParametersColumns.ParameterMode 2178
Borland.Data.MetaData.ProcedureParametersColumns.ParameterName 2179
Borland.Data.MetaData.ProcedureParametersColumns.Precision 2179
Borland.Data.MetaData.ProcedureParametersColumns.ProcedureName 2179
Borland.Data.MetaData.ProcedureParametersColumns.Scale 2179
Borland.Data.MetaData.ProcedureParametersColumns.SchemaName 2179
Borland.Data.MetaData.ProcedureParametersColumns.TypeName 2180
Borland.Data.MetaData.ProceduresColumns 2180
Borland.Data.MetaData.ProceduresColumns.CatalogName 2181
Borland.Data.MetaData.ProceduresColumns.ProcedureName 2181
Borland.Data.MetaData.ProceduresColumns.ProcedureType 2181
Borland.Data.MetaData.ProceduresColumns.SchemaName 2182
Borland.Data.MetaData.ProcedureSourcesColumns 2182
Borland.Data.MetaData.ProcedureSourcesColumns.CatalogName 2183
Borland.Data.MetaData.ProcedureSourcesColumns.Definition

2183
Borland.Data.MetaData.ProcedureSourcesColumns.ExternalDefinition 2184
Borland.Data.MetaData.ProcedureSourcesColumns.ProcedureName 2184
Borland.Data.MetaData.ProcedureSourcesColumns.ProcedureType 2184
Borland.Data.MetaData.ProcedureSourcesColumns.SchemaName 2184
Borland.Data.MetaData.ReservedWordsColumns 2184
Borland.Data.MetaData.ReservedWordsColumns.ReservedWord 2185
Borland.Data.MetaData.RolesColumns 2185
Borland.Data.MetaData.RolesColumns.RoleName 2186
Borland.Data.MetaData.SchemasColumns 2187
Borland.Data.MetaData.SchemasColumns.CatalogName 2188
Borland.Data.MetaData.SchemasColumns.SchemaName 2188
Borland.Data.MetaData.SynonymsColumns 2188
Borland.Data.MetaData.SynonymsColumns.CatalogName 2189
Borland.Data.MetaData.SynonymsColumns.SchemaName 2189
Borland.Data.MetaData.SynonymsColumns.SynonymName 2189
Borland.Data.MetaData.SynonymsColumns.TableCatalogName 2189
Borland.Data.MetaData.SynonymsColumns.TableName 2190
Borland.Data.MetaData.SynonymsColumns.TableSchemaName 2190
Borland.Data.MetaData.TablesColumns 2190
Borland.Data.MetaData.TablesColumns.CatalogName 2191
Borland.Data.MetaData.TablesColumns.SchemaName 2191
Borland.Data.MetaData.TablesColumns.TableName 2192
Borland.Data.MetaData.TablesColumns.TableType 2192
Borland.Data.MetaData.UsersColumns 2192
Borland.Data.MetaData.UsersColumns.UserName 2193
Borland.Data.MetaData.ViewsColumns 2193
Borland.Data.MetaData.ViewsColumns.CatalogName 2194
Borland.Data.MetaData.ViewsColumns.Definition 2195
Borland.Data.MetaData.ViewsColumns.SchemaName 2195
Borland.Data.MetaData.ViewsColumns.ViewName 2195
boTopToBottom enumeration member 565
bpLeft enumeration member 17672
bpMiddle enumeration member 17672
bpoFrame enumeration member 13265
bpoGrabber enumeration member 13265
bpoGradient enumeration member 13265
bpoRoundRect enumeration member 13265
bpRight enumeration member 17672
bsAutoDetect enumeration member 2395
bsBDiagonal enumeration member 14631
bsBottomLine enumeration member 13266
bsBox enumeration member 13266
bsClear enumeration member 14631
bsCross enumeration member 14631
bsDepressed enumeration member 5460
bsDiagCross enumeration member 14631
bsDialog enumeration member 14357
bsDisabled enumeration member 2394
bsDown enumeration member 2394
bsExclusive enumeration member 2394
bsFDiagonal enumeration member 14631
bsFrame enumeration member 13266
bsGrayed enumeration member 5460
bsHorizontal enumeration member 14631
bsHot enumeration member 5460
bsInvisible enumeration member 5460
bsLeftLine enumeration member 13266
bsLowered enumeration member 13267
bsNew enumeration member 2395
bsNone enumeration member 14357
bsNormal enumeration member 5460
bsRaised enumeration member 13267
bsrAsIs enumeration member 23671
bsRightLine enumeration member 13266
bsrLower enumeration member 23671
bsrUpper enumeration member 23671
bsSingle enumeration member 14357
bsSizeable enumeration member 14357

bsSizeToolWin enumeration member 14357
bsSolid enumeration member 14631
bsSpacer enumeration member 13266
bsToolWindow enumeration member 14357
bsTopLine enumeration member 13266
bsUp enumeration member 2394
bsVertical enumeration member 14631
bsWin31 enumeration member 2395
btBack enumeration member 17229
btDIME enumeration member 25925
btEject enumeration member 17229
btHardRebuild enumeration member 17837
btHttp enumeration member 25925
btMime enumeration member 25925
btNext enumeration member 5472, 17229
btNoRebuild enumeration member 17837
btPause enumeration member 17229
btPlay enumeration member 17229
btPrev enumeration member 5472, 17229
btRecord enumeration member 17229
btSoap enumeration member 25925
btSoftRebuild enumeration member 17837
btStep enumeration member 17229
btStop enumeration member 17229
btUnknown enumeration member 25925
ButtonGroup 2197
ButtonGroup namespace 2197
    Classes 2198
    Types 2284
ButtonGroup.TButtonGroup 2198
ButtonGroup.TButtonGroup.Align 2221
ButtonGroup.TButtonGroup.Anchors 2225
ButtonGroup.TButtonGroup.Assign 2210
ButtonGroup.TButtonGroup.BevelEdges 2225
ButtonGroup.TButtonGroup.BevelInner 2226
ButtonGroup.TButtonGroup.BevelKind 2226
ButtonGroup.TButtonGroup.BevelOuter 2227
ButtonGroup.TButtonGroup.BevelWidth 2227
ButtonGroup.TButtonGroup.BorderStyle 2227
ButtonGroup.TButtonGroup.BorderWidth 2228
ButtonGroup.TButtonGroup.ButtonHeight 2228
ButtonGroup.TButtonGroup.ButtonOptions 2228
ButtonGroup.TButtonGroup.ButtonWidth 2229
ButtonGroup.TButtonGroup.Canvas 2229
ButtonGroup.TButtonGroup.Create 2209
ButtonGroup.TButtonGroup.CreateButton 2211
ButtonGroup.TButtonGroup.CreateHandle 2211
ButtonGroup.TButtonGroup.CreateParams 2211
ButtonGroup.TButtonGroup.Destroy 2209
ButtonGroup.TButtonGroup.DockSite 2231
ButtonGroup.TButtonGroup.DoEndDrag 2212
ButtonGroup.TButtonGroup.DoHotButton 2212
ButtonGroup.TButtonGroup.DoItemClicked 2212
ButtonGroup.TButtonGroup.DoMouseWheelDown 2213
ButtonGroup.TButtonGroup.DoMouseWheelUp 2213
ButtonGroup.TButtonGroup.DoReorderButton 2214
ButtonGroup.TButtonGroup.DoStartDrag 2214
ButtonGroup.TButtonGroup.DragCursor 2232
ButtonGroup.TButtonGroup.DragDrop 2214
ButtonGroup.TButtonGroup.DragImageList 2232
ButtonGroup.TButtonGroup.DragIndex 2233
ButtonGroup.TButtonGroup.DragKind 2233
ButtonGroup.TButtonGroup.DragMode 2233
ButtonGroup.TButtonGroup.DragOver 2215
ButtonGroup.TButtonGroup.DrawButton 2215
ButtonGroup.TButtonGroup.Enabled 2235
ButtonGroup.TButtonGroup.Font 2235
ButtonGroup.TButtonGroup.GetButtonClass 2215
ButtonGroup.TButtonGroup.GetButtonRect 2215
ButtonGroup.TButtonGroup.GetButtonsClass 2216
ButtonGroup.TButtonGroup.GetDragImages 2216
ButtonGroup.TButtonGroup.Height 2237
ButtonGroup.TButtonGroup.Images 2238
ButtonGroup.TButtonGroup.IndexOfButtonAt 2216
ButtonGroup.TButtonGroup.ItemIndex 2238
ButtonGroup.TButtonGroup.Items 2238
ButtonGroup.TButtonGroup.MouseDown 2216
ButtonGroup.TButtonGroup.MouseMove 2217

ButtonGroup.TButtonGroup.MouseUp 2217
ButtonGroup.TButtonGroup.Notification 2218
ButtonGroup.TButtonGroup.OnAfterDrawButton 2248
ButtonGroup.TButtonGroup.OnAlignInsertBefore 2248
ButtonGroup.TButtonGroup.OnAlignPosition 2249
ButtonGroup.TButtonGroup.OnBeforeDrawButton 2249
ButtonGroup.TButtonGroup.OnButtonClicked 2249
ButtonGroup.TButtonGroup.OnClick 2249
ButtonGroup.TButtonGroup.OnContextPopup 2251
ButtonGroup.TButtonGroup.OnDockDrop 2251
ButtonGroup.TButtonGroup.OnDockOver 2251
ButtonGroup.TButtonGroup.OnDragDrop 2252
ButtonGroup.TButtonGroup.OnDragOver 2254
ButtonGroup.TButtonGroup.OnDrawButton 2256
ButtonGroup.TButtonGroup.OnDrawIcon 2256
ButtonGroup.TButtonGroup.OnEndDock 2257
ButtonGroup.TButtonGroup.OnEndDrag 2257
ButtonGroup.TButtonGroup.OnEnter 2258
ButtonGroup.TButtonGroup.OnExit 2258
ButtonGroup.TButtonGroup.OnHotButton 2258
ButtonGroup.TButtonGroup.OnKeyDown 2259
ButtonGroup.TButtonGroup.OnKeyPress 2261
ButtonGroup.TButtonGroup.OnKeyUp 2262
ButtonGroup.TButtonGroup.OnMouseActivate 2264
ButtonGroup.TButtonGroup.OnMouseDown 2265
ButtonGroup.TButtonGroup.OnMouseEnter 2267
ButtonGroup.TButtonGroup.OnMouseLeave 2268
ButtonGroup.TButtonGroup.OnMouseMove 2268
ButtonGroup.TButtonGroup.OnMouseUp 2270
ButtonGroup.TButtonGroup.OnMouseWheel 2273
ButtonGroup.TButtonGroup.OnMouseWheelDown 2273
ButtonGroup.TButtonGroup.OnMouseWheelUp 2273
ButtonGroup.TButtonGroup.OnReorderButton 2274
ButtonGroup.TButtonGroup.OnStartDock 2274
ButtonGroup.TButtonGroup.OnStartDrag 2274
ButtonGroup.TButtonGroup.Paint 2218
ButtonGroup.TButtonGroup.PopupMenu 2238
ButtonGroup.TButtonGroup.RemoveInsertionPoints 2219
ButtonGroup.TButtonGroup.Resize 2219
ButtonGroup.TButtonGroup.ScrollIntoView 2219
ButtonGroup.TButtonGroup.SetInsertionPoints 2220
ButtonGroup.TButtonGroup.ShowHint 2243
ButtonGroup.TButtonGroup.TabOrder 2244
ButtonGroup.TButtonGroup.TabStop 2245
ButtonGroup.TButtonGroup.TargetIndexAt 2220
ButtonGroup.TButtonGroup.UpdateAllButtons 2220
ButtonGroup.TButtonGroup.UpdateButton 2220
ButtonGroup.TButtonGroup.Visible 2245
ButtonGroup.TButtonGroup.Width 2247
ButtonGroup.TGrpButtonDrawEvent 2284
ButtonGroup.TGrpButtonDrawEvent type 2284
ButtonGroup.TGrpButtonDrawIconEvent 2285
ButtonGroup.TGrpButtonDrawIconEvent type 2285
ButtonGroup.TGrpButtonEvent 2285
ButtonGroup.TGrpButtonEvent type 2285
ButtonGroup.TGrpButtonItem 2275
ButtonGroup.TGrpButtonItem.ButtonGroup 2278
ButtonGroup.TGrpButtonItem.Collection 2278
ButtonGroup.TGrpButtonItem.GetNotifyTarget 2277
ButtonGroup.TGrpButtonItem.ScrollIntoView 2278
ButtonGroup.TGrpButtonItemClass 2286
ButtonGroup.TGrpButtonItemClass type 2286
ButtonGroup.TGrpButtonItems 2278
ButtonGroup.TGrpButtonItems.Add 2281
ButtonGroup.TGrpButtonItems.AddItem 2281
ButtonGroup.TGrpButtonItems.BeginUpdate 2281
ButtonGroup.TGrpButtonItems.ButtonGroup 2283
ButtonGroup.TGrpButtonItems.Create 2280
ButtonGroup.TGrpButtonItems.GetOwner 2281
ButtonGroup.TGrpButtonItems.Insert 2282
ButtonGroup.TGrpButtonItems.Items 2283
ButtonGroup.TGrpButtonItems.Update 2282
ButtonGroup.TGrpButtonItemsClass 2286
ButtonGroup.TGrpButtonItemsClass type 2286
ButtonGroup.TGrpButtonOptions 2286
ButtonGroup.TGrpButtonOptions type 2286
ButtonGroup.TGrpButtonReorderEvent 2287
ButtonGroup.TGrpButtonReorderEvent type 2287

Buttons 2289
Buttons namespace 2289
Classes 2290
Functions 2391
Structs, Records, Enums 2392
Types 2396
Buttons.DrawButtonFace 2391
Buttons.DrawButtonFace function 2391
Buttons.TBitBtn 2290
Buttons.TBitBtn.Action 2307
Buttons.TBitBtn.ActionChange 2303
Buttons.TBitBtn.Align 2307
Buttons.TBitBtn.Anchors 2310
Buttons.TBitBtn.BiDiMode 2311
Buttons.TBitBtn.Cancel 2312
Buttons.TBitBtn.Caption 2312
Buttons.TBitBtn.Click 2303
Buttons.TBitBtn.Constraints 2315
Buttons.TBitBtn.CopyImage 2304
Buttons.TBitBtn.Create 2302
Buttons.TBitBtn.CreateHandle 2304
Buttons.TBitBtn.CreateParams 2304
Buttons.TBitBtn.Default 2315
Buttons.TBitBtn.Destroy 2302
Buttons.TBitBtn.Enabled 2316
Buttons.TBitBtn.GetActionLinkClass 2305
Buttons.TBitBtn.GetPalette 2305
Buttons.TBitBtn.Glyph 2317
Buttons.TBitBtn.Kind 2317
Buttons.TBitBtn.Layout 2318
Buttons.TBitBtn.Margin 2318
Buttons.TBitBtn.ModalResult 2319
Buttons.TBitBtn.NumGlyphs 2320
Buttons.TBitBtn.OnEnter 2326
Buttons.TBitBtn.OnExit 2328
Buttons.TBitBtn.ParentBiDiMode 2320
Buttons.TBitBtn.ParentShowHint 2320
Buttons.TBitBtn.SetButtonStyle 2306
Buttons.TBitBtn.ShowHint 2321
Buttons.TBitBtn.Spacing 2322
Buttons.TBitBtn.Style 2323
Buttons.TBitBtn.TabOrder 2323
Buttons.TBitBtn.TabStop 2324
Buttons.TBitBtn.Visible 2324
Buttons.TBitBtnActionLink 2330
Buttons.TBitBtnActionLink.AssignClient 2334
Buttons.TBitBtnActionLink.Create 2332
Buttons.TBitBtnActionLink.FClient 2333
Buttons.TBitBtnActionLink.FImageIndex 2333
Buttons.TBitBtnActionLink.IsImageIndexLinked 2334
Buttons.TBitBtnActionLink.SetImageIndex 2335
Buttons.TBitBtnKind 2392
Buttons.TBitBtnKind enumeration 2392
Buttons.TButtonLayout 2393
Buttons.TButtonLayout enumeration 2393
Buttons.TButtonState 2394
Buttons.TButtonState enumeration 2394
Buttons.TButtonStyle 2395
Buttons.TButtonStyle enumeration 2395
Buttons.TNumGlyphs 2396
Buttons.TNumGlyphs type 2396
Buttons.TSpeedButton 2335
Buttons.TSpeedButton.Action 2350
Buttons.TSpeedButton.ActionChange 2345
Buttons.TSpeedButton.Align 2350
Buttons.TSpeedButton.AllowAllUp 2354
Buttons.TSpeedButton.Anchors 2355
Buttons.TSpeedButton.BiDiMode 2355
Buttons.TSpeedButton.Caption 2356
Buttons.TSpeedButton.Click 2346
Buttons.TSpeedButton.Constraints 2356
Buttons.TSpeedButton.CopyImage 2346
Buttons.TSpeedButton.Create 2344
Buttons.TSpeedButton.Destroy 2344
Buttons.TSpeedButton.Down 2357
Buttons.TSpeedButton.Enabled 2357
Buttons.TSpeedButton.Flat 2357
Buttons.TSpeedButton.Font 2358

Buttons.TSpeedButton.FState 2345
Buttons.TSpeedButton.GetActionLinkClass 2346
Buttons.TSpeedButton.GetPalette 2347
Buttons.TSpeedButton.Glyph 2359
Buttons.TSpeedButton.GroupIndex 2360
Buttons.TSpeedButton.Layout 2360
Buttons.TSpeedButton.Loaded 2347
Buttons.TSpeedButton.Margin 2361
Buttons.TSpeedButton.MouseDown 2348
Buttons.TSpeedButton.MouseInControl 2361
Buttons.TSpeedButton.MouseMove 2348
Buttons.TSpeedButton.MouseUp 2349
Buttons.TSpeedButton.NumGlyphs 2361
Buttons.TSpeedButton.OnClick 2372
Buttons.TSpeedButton.OnDblClick 2374
Buttons.TSpeedButton.OnMouseActivate 2374
Buttons.TSpeedButton.OnMouseDown 2375
Buttons.TSpeedButton.OnMouseEnter 2377
Buttons.TSpeedButton.OnMouseLeave 2378
Buttons.TSpeedButton.OnMouseMove 2378
Buttons.TSpeedButton.OnMouseUp 2380
Buttons.TSpeedButton.Paint 2349
Buttons.TSpeedButton.ParentBiDiMode 2362
Buttons.TSpeedButton.ParentFont 2362
Buttons.TSpeedButton.ParentShowHint 2363
Buttons.TSpeedButton.PopupMenu 2363
Buttons.TSpeedButton.ShowHint 2368
Buttons.TSpeedButton.Spacing 2369
Buttons.TSpeedButton.Transparent 2370
Buttons.TSpeedButton.Visible 2370
Buttons.TSpeedButtonActionLink 2382
Buttons.TSpeedButtonActionLink.AssignClient 2386
Buttons.TSpeedButtonActionLink.Create 2385
Buttons.TSpeedButtonActionLink.FClient 2386
Buttons.TSpeedButtonActionLink.FImageIndex 2386
Buttons.TSpeedButtonActionLink.IsCheckedLinked 2387
Buttons.TSpeedButtonActionLink.IsGroupIndexLinked 2387
Buttons.TSpeedButtonActionLink.IsImageIndexLinked 2388
Buttons.TSpeedButtonActionLink.SetChecked 2388
Buttons.TSpeedButtonActionLink.SetGroupIndex 2389
Buttons.TSpeedButtonActionLink.SetImageIndex 2389
bvLowered enumeration member 6270
bvNone enumeration member 6270
bvRaised enumeration member 6270
bvSpace enumeration member 6270

C

caBold enumeration member 5444
caBottom enumeration member 10347
caCache enumeration member 15246
caColor enumeration member 5444
caDefault enumeration member 10347
caDelete enumeration member 566
caDestroy enumeration member 15246
caFace enumeration member 5444
caFree enumeration member 14355
caHide enumeration member 14355
caItalic enumeration member 5444
caMinimize enumeration member 14355
caModify enumeration member 566
caMove enumeration member 566
caNone enumeration member 14355
capAbove enumeration member 17090
capBelow enumeration member 17090
capLeft enumeration member 17090
capRight enumeration member 17090
caProtected enumeration member 5444
CaptionedDockTree 2397
CaptionedDockTree namespace 2397
- Classes 2398
- Constants 2413
- Structs, Records, Enums 2409
- Types 2411

CaptionedDockTree.dchtCaption 2413
CaptionedDockTree.dchtCaption constant 2413
CaptionedDockTree.dchtClose 2413
CaptionedDockTree.dchtClose constant 2413
CaptionedDockTree.dchtCustom 2414

CaptionedDockTree.dchtCustom constant 2414
CaptionedDockTree.dchtNone 2414
CaptionedDockTree.dchtNone constant 2414
CaptionedDockTree.dchtPin 2414
CaptionedDockTree.dchtPin constant 2414
CaptionedDockTree.TCaptionedDockTree 2398
CaptionedDockTree.TCaptionedDockTree.AdjustCaptionRect 2402
CaptionedDockTree.TCaptionedDockTree.AdjustDockRect 2402
CaptionedDockTree.TCaptionedDockTree.Create 2401
CaptionedDockTree.TCaptionedDockTree.Destroy 2401
CaptionedDockTree.TCaptionedDockTree.DockCaptionDrawer 2405
CaptionedDockTree.TCaptionedDockTree.DockCaptionOrientation 2405
CaptionedDockTree.TCaptionedDockTree.GetDockCaptionDrawer 2403
CaptionedDockTree.TCaptionedDockTree.GetParentFormState 2403
CaptionedDockTree.TCaptionedDockTree.InternalCaptionHitTest 2403
CaptionedDockTree.TCaptionedDockTree.PaintDockFrame 2404
CaptionedDockTree.TCaptionedDockTree.WndProc 2404
CaptionedDockTree.TCaptionedDockTree.ZoneCaptionHitTest 2404
CaptionedDockTree.TCaptionedDockTreeClass 2411
CaptionedDockTree.TCaptionedDockTreeClass type 2411
CaptionedDockTree.TDockCaptionDrawer 2405
CaptionedDockTree.TDockCaptionDrawer.Create 2406
CaptionedDockTree.TDockCaptionDrawer.DockCaptionHitTest 2407
CaptionedDockTree.TDockCaptionDrawer.DockCaptionPinButton 2408
CaptionedDockTree.TDockCaptionDrawer.DrawDockCaption 2407
CaptionedDockTree.TDockCaptionDrawerClass 2411
CaptionedDockTree.TDockCaptionDrawerClass type 2411
CaptionedDockTree.TDockCaptionHitTest 2411
CaptionedDockTree.TDockCaptionHitTest type 2411
CaptionedDockTree.TDockCaptionOrientation 2409
CaptionedDockTree.TDockCaptionOrientation enumeration 2409
CaptionedDockTree.TDockCaptionPinButton 2409
CaptionedDockTree.TDockCaptionPinButton enumeration 2409
CaptionedDockTree.TParentFormState 2410
CaptionedDockTree.TParentFormState record 2410
caSize enumeration member 5444
caStrikeOut enumeration member 5444
CatalogsColumns class 2132
about CatalogsColumns class 2132
CatalogName 2133
CatalogsColumns fields 2132
CatalogsColumns Members 2132
CategoryButtons 2415
CategoryButtons namespace 2415
Classes 2417
Constants 2551
Structs, Records, Enums 2543
Types 2544
CategoryButtons.crDragCopy 2551
CategoryButtons.crDragCopy constant 2551
CategoryButtons.TBaseButtonItem 2417
CategoryButtons.TBaseButtonItem.Action 2422
CategoryButtons.TBaseButtonItem.ActionChange 2420
CategoryButtons.TBaseButtonItem.ActionLink 2422
CategoryButtons.TBaseButtonItem.Assign 2420
CategoryButtons.TBaseButtonItem.Caption 2422
CategoryButtons.TBaseButtonItem.Create 2419
CategoryButtons.TBaseButtonItem.Data 2423
CategoryButtons.TBaseButtonItem.GetActionLinkClass 2421
CategoryButtons.TBaseButtonItem.GetDisplayName 2421
CategoryButtons.TBaseButtonItem.GetNotifyTarget 2421
CategoryButtons.TBaseButtonItem.Hint 2423
CategoryButtons.TBaseButtonItem.ImageIndex 2423
CategoryButtons.TBaseButtonItem.OnClick 2423
CategoryButtons.TBaseButtonItem.ScrollIntoView 2421
CategoryButtons.TBaseButtonItemClass 2544
CategoryButtons.TBaseButtonItemClass type 2544
CategoryButtons.TButtonCategories 2424
CategoryButtons.TButtonCategories.Add 2426

CategoryButtons.TButtonCategories.AddItem 2426
CategoryButtons.TButtonCategories.BeginUpdate 2427
CategoryButtons.TButtonCategories.ButtonGroup 2428
CategoryButtons.TButtonCategories.Create 2426
CategoryButtons.TButtonCategories.GetOwner 2427
CategoryButtons.TButtonCategories.IndexOf 2427
CategoryButtons.TButtonCategories.Insert 2428
CategoryButtons.TButtonCategories.Items 2429
CategoryButtons.TButtonCategories.Update 2428
CategoryButtons.TButtonCategoriesClass 2544
CategoryButtons.TButtonCategoriesClass type 2544
CategoryButtons.TButtonCategory 2429
CategoryButtons.TButtonCategory.Assign 2432
CategoryButtons.TButtonCategory.Caption 2434
CategoryButtons.TButtonCategory.Categories 2434
CategoryButtons.TButtonCategory.Collapsed 2434
CategoryButtons.TButtonCategory.Color 2434
CategoryButtons.TButtonCategory.Create 2431
CategoryButtons.TButtonCategory.Data 2434
CategoryButtons.TButtonCategory.Destroy 2431
CategoryButtons.TButtonCategory.GradientColor 2435
CategoryButtons.TButtonCategory.IndexOf 2432
CategoryButtons.TButtonCategory.InterfaceData 2435
CategoryButtons.TButtonCategory.Items 2435
CategoryButtons.TButtonCategory.ScrollIntoView 2433
CategoryButtons.TButtonCategory.SetIndex 2433
CategoryButtons.TButtonCategory.TextColor 2435
CategoryButtons.TButtonCategoryClass 2545
CategoryButtons.TButtonCategoryClass type 2545
CategoryButtons.TButtonCollection 2436
CategoryButtons.TButtonCollection.Add 2438
CategoryButtons.TButtonCollection.AddItem 2438
CategoryButtons.TButtonCollection.Category 2440
CategoryButtons.TButtonCollection.Create 2438
CategoryButtons.TButtonCollection.GetOwner 2438
CategoryButtons.TButtonCollection.Insert 2439
CategoryButtons.TButtonCollection.Items 2440
CategoryButtons.TButtonCollection.Notify 2439
CategoryButtons.TButtonCollection.Update 2440
CategoryButtons.TButtonDrawState 2545
CategoryButtons.TButtonDrawState type 2545
CategoryButtons.TButtonItem 2441
CategoryButtons.TButtonItem.Assign 2443
CategoryButtons.TButtonItem.ButtonGroup 2444
CategoryButtons.TButtonItem.Category 2445
CategoryButtons.TButtonItem.GetNotifyTarget 2444
CategoryButtons.TButtonItem.InterfaceData 2445
CategoryButtons.TButtonItem.ScrollIntoView 2444
CategoryButtons.TButtonItemActionLink 2445
CategoryButtons.TButtonItemActionLink.AssignClient 2448
CategoryButtons.TButtonItemActionLink.DoShowHint 2448
CategoryButtons.TButtonItemActionLink.FClient 2447
CategoryButtons.TButtonItemActionLink.IsCaptionLinked 2448
CategoryButtons.TButtonItemActionLink.IsHintLinked 2449
CategoryButtons.TButtonItemActionLink.IsImageIndexLinked 2449
CategoryButtons.TButtonItemActionLink.IsOnExecuteLinked 2450
CategoryButtons.TButtonItemActionLink.SetCaption 2450
CategoryButtons.TButtonItemActionLink.SetHint 2450
CategoryButtons.TButtonItemActionLink.SetImageIndex 2451
CategoryButtons.TButtonItemActionLink.SetOnExecute 2452
CategoryButtons.TButtonItemActionLinkClass 2546
CategoryButtons.TButtonItemActionLinkClass type 2546
CategoryButtons.TButtonItemClass 2546
CategoryButtons.TButtonItemClass type 2546
CategoryButtons.TCatButtonCopyEvent 2546
CategoryButtons.TCatButtonCopyEvent type 2546
CategoryButtons.TCatButtonDrawEvent 2547
CategoryButtons.TCatButtonDrawEvent type 2547
CategoryButtons.TCatButtonDrawIconEvent 2547
CategoryButtons.TCatButtonDrawIconEvent type 2547
CategoryButtons.TCatButtonEvent 2547
CategoryButtons.TCatButtonEvent type 2547
CategoryButtons.TCatButtonFlow 2543
CategoryButtons.TCatButtonFlow enumeration 2543
CategoryButtons.TCatButtonGetHint 2548
CategoryButtons.TCatButtonGetHint type 2548
CategoryButtons.TCatButtonOptions 2548

CategoryButtons.TCatButtonOptions type 2548
CategoryButtons.TCatButtonReorderEvent 2549
CategoryButtons.TCatButtonReorderEvent type 2549
CategoryButtons.TCategoryButtons 2452
CategoryButtons.TCategoryButtons.Align 2480
CategoryButtons.TCategoryButtons.Anchors 2480
CategoryButtons.TCategoryButtons.Assign 2465
CategoryButtons.TCategoryButtons.BackgroundGradientColor 2481
CategoryButtons.TCategoryButtons.BackgroundGradientDirection 2481
CategoryButtons.TCategoryButtons.BeginAutoDrag 2466
CategoryButtons.TCategoryButtons.BevelEdges 2481
CategoryButtons.TCategoryButtons.BevelInner 2482
CategoryButtons.TCategoryButtons.BevelKind 2482
CategoryButtons.TCategoryButtons.BevelOuter 2483
CategoryButtons.TCategoryButtons.BevelWidth 2483
CategoryButtons.TCategoryButtons.BorderStyle 2484
CategoryButtons.TCategoryButtons.BorderWidth 2484
CategoryButtons.TCategoryButtons.ButtonFlow 2484
CategoryButtons.TCategoryButtons.ButtonHeight 2485
CategoryButtons.TCategoryButtons.ButtonOptions 2485
CategoryButtons.TCategoryButtons.ButtonWidth 2486
CategoryButtons.TCategoryButtons.Categories 2486
CategoryButtons.TCategoryButtons.CMFontchanged 2466
CategoryButtons.TCategoryButtons.Color 2486
CategoryButtons.TCategoryButtons.Create 2464
CategoryButtons.TCategoryButtons.CreateHandle 2466
CategoryButtons.TCategoryButtons.CreateParams 2467
CategoryButtons.TCategoryButtons.Cursor 2486
CategoryButtons.TCategoryButtons.Destroy 2464
CategoryButtons.TCategoryButtons.DoBeginDrag 2467
CategoryButtons.TCategoryButtons.DockSite 2489
CategoryButtons.TCategoryButtons.DoCopyButton 2468
CategoryButtons.TCategoryButtons.DoEndDrag 2468
CategoryButtons.TCategoryButtons.DoHotButton 2468
CategoryButtons.TCategoryButtons.DoItemClicked 2469
CategoryButtons.TCategoryButtons.DoMouseWheelDown 2469
CategoryButtons.TCategoryButtons.DoMouseWheelUp 2469
CategoryButtons.TCategoryButtons.DoReorderButton 2470
CategoryButtons.TCategoryButtons.DoReorderCategory 2470
CategoryButtons.TCategoryButtons.DoSelectedItemChanged 2470
CategoryButtons.TCategoryButtons.DoStartDrag 2470
CategoryButtons.TCategoryButtons.DragButton 2490
CategoryButtons.TCategoryButtons.DragCategory 2490
CategoryButtons.TCategoryButtons.DragCursor 2491
CategoryButtons.TCategoryButtons.DragDrop 2471
CategoryButtons.TCategoryButtons.DragImageList 2491
CategoryButtons.TCategoryButtons.DragKind 2492
CategoryButtons.TCategoryButtons.DragMode 2492
CategoryButtons.TCategoryButtons.DragOver 2471
CategoryButtons.TCategoryButtons.DrawButton 2472
CategoryButtons.TCategoryButtons.Enabled 2494
CategoryButtons.TCategoryButtons.FocusedItem 2494
CategoryButtons.TCategoryButtons.Font 2494
CategoryButtons.TCategoryButtons.GenerateDragImage 2472
CategoryButtons.TCategoryButtons.GetAllowReorder 2472
CategoryButtons.TCategoryButtons.GetButtonAt 2472
CategoryButtons.TCategoryButtons.GetButtonCategoriesClass 2473
CategoryButtons.TCategoryButtons.GetButtonCategoryClass 2473
CategoryButtons.TCategoryButtons.GetButtonItemClass 2473
CategoryButtons.TCategoryButtons.GetButtonRect 2473
CategoryButtons.TCategoryButtons.GetCategoryAt 2473
CategoryButtons.TCategoryButtons.GetCategoryRect 2474
CategoryButtons.TCategoryButtons.GetDragImages 2474
CategoryButtons.TCategoryButtons.GetScrollPos 2474
CategoryButtons.TCategoryButtons.GetTargetAt 2474
CategoryButtons.TCategoryButtons.GradientDirection 2495
CategoryButtons.TCategoryButtons.Height 2496
CategoryButtons.TCategoryButtons.HotButtonColor 2497
CategoryButtons.TCategoryButtons.Images 2497
CategoryButtons.TCategoryButtons.MouseDown 2475
CategoryButtons.TCategoryButtons.MouseMove 2475
CategoryButtons.TCategoryButtons.MouseUp 2476
CategoryButtons.TCategoryButtons.Notification 2476
CategoryButtons.TCategoryButtons.OnAfterDrawButton 2508

CategoryButtons.TCategoryButtons.OnAlignInsertBefore 2508
CategoryButtons.TCategoryButtons.OnAlignPosition 2508
CategoryButtons.TCategoryButtons.OnBeforeDrawButton 2509
CategoryButtons.TCategoryButtons.OnButtonClicked 2509
CategoryButtons.TCategoryButtons.OnCategoryCollapase 2509
CategoryButtons.TCategoryButtons.OnClick 2510
CategoryButtons.TCategoryButtons.OnContextPopup 2510
CategoryButtons.TCategoryButtons.OnCopyButton 2511
CategoryButtons.TCategoryButtons.OnDockDrop 2511
CategoryButtons.TCategoryButtons.OnDockOver 2513
CategoryButtons.TCategoryButtons.OnDragDrop 2515
CategoryButtons.TCategoryButtons.OnDragOver 2517
CategoryButtons.TCategoryButtons.OnDrawButton 2519
CategoryButtons.TCategoryButtons.OnDrawIcon 2520
CategoryButtons.TCategoryButtons.OnDrawText 2520
CategoryButtons.TCategoryButtons.OnEndDock 2520
CategoryButtons.TCategoryButtons.OnEndDrag 2520
CategoryButtons.TCategoryButtons.OnEnter 2521
CategoryButtons.TCategoryButtons.OnExit 2524
CategoryButtons.TCategoryButtons.OnGetHint 2525
CategoryButtons.TCategoryButtons.OnHotButton 2526
CategoryButtons.TCategoryButtons.OnKeyDown 2526
CategoryButtons.TCategoryButtons.OnKeyPress 2529
CategoryButtons.TCategoryButtons.OnKeyUp 2530
CategoryButtons.TCategoryButtons.OnMouseActivate 2531
CategoryButtons.TCategoryButtons.OnMouseDown 2532
CategoryButtons.TCategoryButtons.OnMouseEnter 2535
CategoryButtons.TCategoryButtons.OnMouseLeave 2535
CategoryButtons.TCategoryButtons.OnMouseMove 2535
CategoryButtons.TCategoryButtons.OnMouseUp 2537
CategoryButtons.TCategoryButtons.OnMouseWheel 2540
CategoryButtons.TCategoryButtons.OnMouseWheelDown 2540
CategoryButtons.TCategoryButtons.OnMouseWheelUp 2540
CategoryButtons.TCategoryButtons.OnReorderButton 2541
CategoryButtons.TCategoryButtons.OnReorderCategory 2541
CategoryButtons.TCategoryButtons.OnSelectedItemChange 2541
CategoryButtons.TCategoryButtons.OnStartDock 2541
CategoryButtons.TCategoryButtons.OnStartDrag 2542
CategoryButtons.TCategoryButtons.Paint 2477
CategoryButtons.TCategoryButtons.PopupMenu 2497
CategoryButtons.TCategoryButtons.RegularButtonColor 2502
CategoryButtons.TCategoryButtons.RemoveInsertionPoints 2477
CategoryButtons.TCategoryButtons.Resize 2477
CategoryButtons.TCategoryButtons.ScrollIntoView 2478
CategoryButtons.TCategoryButtons.ScrollRectIntoView 2478
CategoryButtons.TCategoryButtons.SelectedButtonColor 2502
CategoryButtons.TCategoryButtons.SelectedItem 2502
CategoryButtons.TCategoryButtons.SetInsertionButton 2478
CategoryButtons.TCategoryButtons.SetScrollPos 2478
CategoryButtons.TCategoryButtons.ShowHint 2502
CategoryButtons.TCategoryButtons.TabOrder 2504
CategoryButtons.TCategoryButtons.TabStop 2504
CategoryButtons.TCategoryButtons.UpdateAllButtons 2479
CategoryButtons.TCategoryButtons.UpdateButton 2479
CategoryButtons.TCategoryButtons.Visible 2505
CategoryButtons.TCategoryButtons.Width 2506
CategoryButtons.TCategoryCollapseEvent 2549
CategoryButtons.TCategoryCollapseEvent type 2549
CategoryButtons.TCategoryReorderEvent 2549
CategoryButtons.TCategoryReorderEvent type 2549
caTop enumeration member 10347
caUnderline enumeration member 5444
cbChecked enumeration member 21945
cbCustomColor enumeration member 13267
cbCustomColors enumeration member 13267
cbExtendedColors enumeration member 13267
cbfHorizontal enumeration member 2543
cbfVertical enumeration member 2543
cbGrayed enumeration member 21945
cbIncludeDefault enumeration member 13267
cbIncludeNone enumeration member 13267
cbPrettyNames enumeration member 13267
cbsAuto enumeration member 9765
cbsEllipsis enumeration member 9765
cbsNone enumeration member 9765
cbStandardColors enumeration member 13267
cbSystemColors enumeration member 13267
cbUnchecked enumeration member 21945

ccCdecl enumeration member 15485
ccPascal enumeration member 15485
ccReg enumeration member 15485
ccSafeCall enumeration member 15485
ccStdCall enumeration member 15485
cdAnyColor enumeration member 11548
cdFullOpen enumeration member 11548
cdPostErase enumeration member 5445
cdPostPaint enumeration member 5445
cdPreErase enumeration member 5445
cdPrePaint enumeration member 5445
cdPreventFullOpen enumeration member 11548
cdShowHelp enumeration member 11548
cdSolidColor enumeration member 11548
ceLarge enumeration member 13268
ceMedium enumeration member 13268
ceNone enumeration member 13268
ceSmall enumeration member 13268
cfmPersistent enumeration member 10253
cfmSession enumeration member 10253
cfmVirtual enumeration member 10253
cgGridLines enumeration member 17674
CGIApp 2553
CGIApp namespace 2553
    Classes 2554
CGIApp.TCGIApplication 2554
CGIApp.TCGIApplication.Destroy 2557
CGIApp.TCGIApplication.Run 2558
CGIApp.TCGIFactory 2559
CGIApp.TCGIFactory.Create 2560
CGIApp.TCGIFactory.NewRequest 2560
CGIApp.TCGIFactory.NewResponse 2561
CGIApp.TCGIFactory.OutputFileName 2561
CGIHTTP 2563
CGIHTTP namespace 2563
    Classes 2564
    Types 2591
CGIHTTP.TCGIRequest 2564
CGIHTTP.TCGIRequest.Create 2567
CGIHTTP.TCGIRequest.GetDateVariable 2567
CGIHTTP.TCGIRequest.GetFieldByName 2568
CGIHTTP.TCGIRequest.GetIntegerVariable 2568
CGIHTTP.TCGIRequest.GetStringVariable 2569
CGIHTTP.TCGIRequest.ReadClient 2570
CGIHTTP.TCGIRequest.ReadString 2570
CGIHTTP.TCGIRequest.TranslateURI 2570
CGIHTTP.TCGIRequest.WriteClient 2571
CGIHTTP.TCGIRequest.WriteHeaders 2571
CGIHTTP.TCGIRequest.WriteString 2572
CGIHTTP.TCGIRequestClass 2591
CGIHTTP.TCGIRequestClass type 2591
CGIHTTP.TCGIResponse 2572
CGIHTTP.TCGIResponse.Create 2575
CGIHTTP.TCGIResponse.GetContent 2576
CGIHTTP.TCGIResponse.GetDateVariable 2576
CGIHTTP.TCGIResponse.GetIntegerVariable 2577
CGIHTTP.TCGIResponse.GetLogMessage 2577
CGIHTTP.TCGIResponse.GetStatusCode 2578
CGIHTTP.TCGIResponse.GetStringVariable 2578
CGIHTTP.TCGIResponse.InitResponse 2579
CGIHTTP.TCGIResponse.SendRedirect 2579
CGIHTTP.TCGIResponse.SendResponse 2579
CGIHTTP.TCGIResponse.SendStream 2580
CGIHTTP.TCGIResponse.Sent 2580
CGIHTTP.TCGIResponse.SetContent 2580
CGIHTTP.TCGIResponse.SetDateVariable 2581
CGIHTTP.TCGIResponse.SetIntegerVariable 2581
CGIHTTP.TCGIResponse.SetLogMessage 2582
CGIHTTP.TCGIResponse.SetStatusCode 2582
CGIHTTP.TCGIResponse.SetStringVariable 2582
CGIHTTP.TCGIResponseClass 2591
CGIHTTP.TCGIResponseClass type 2591
CGIHTTP.TWinCGIRequest 2583
CGIHTTP.TWinCGIRequest.Create 2586
CGIHTTP.TWinCGIRequest.Destroy 2586
CGIHTTP.TWinCGIRequest.GetFieldByName 2587
CGIHTTP.TWinCGIRequest.GetStringVariable 2587
CGIHTTP.TWinCGIRequest.ReadClient 2588

CGIHTTP.TWinCGIRequest.ReadString 2589
CGIHTTP.TWinCGIRequest.WriteClient 2589
CGIHTTP.TWinCGIRequest.WriteString 2589
CGIHTTP.TWinCGIRequestClass 2592
CGIHTTP.TWinCGIRequestClass type 2592
CGIHTTP.TWinCGIResponse 2590
CGIHTTP.TWinCGIResponseClass 2592
CGIHTTP.TWinCGIResponseClass type 2592
cgOutliner enumeration member 17674
cgPivotable enumeration member 17674
CheckLst 2593
CheckLst namespace 2593
    Classes 2594
CheckLst.TCheckListBox 2594
CheckLst.TCheckListBox.Align 2614
CheckLst.TCheckListBox.AllowGrayed 2617
CheckLst.TCheckListBox.Anchors 2618
CheckLst.TCheckListBox.AutoComplete 2618
CheckLst.TCheckListBox.BevelEdges 2619
CheckLst.TCheckListBox.BevelInner 2619
CheckLst.TCheckListBox.BevelKind 2619
CheckLst.TCheckListBox.BevelOuter 2620
CheckLst.TCheckListBox.BevelWidth 2620
CheckLst.TCheckListBox.BiDiMode 2621
CheckLst.TCheckListBox.BorderStyle 2621
CheckLst.TCheckListBox.Checked 2621
CheckLst.TCheckListBox.ClickCheck 2607
CheckLst.TCheckListBox.Color 2622
CheckLst.TCheckListBox.Columns 2623
CheckLst.TCheckListBox.Constraints 2623
CheckLst.TCheckListBox.Create 2606
CheckLst.TCheckListBox.CreateParams 2607
CheckLst.TCheckListBox.CreateWnd 2608
CheckLst.TCheckListBox.Ctl3D 2624
CheckLst.TCheckListBox.DeleteString 2609
CheckLst.TCheckListBox.Destroy 2606
CheckLst.TCheckListBox.DragCursor 2624
CheckLst.TCheckListBox.DragKind 2625
CheckLst.TCheckListBox.DragMode 2625
CheckLst.TCheckListBox.DrawItem 2609
CheckLst.TCheckListBox.Enabled 2627
CheckLst.TCheckListBox.Flat 2627
CheckLst.TCheckListBox.Font 2627
CheckLst.TCheckListBox.GetCheckWidth 2609
CheckLst.TCheckListBox.GetItemData 2610
CheckLst.TCheckListBox.Header 2628
CheckLst.TCheckListBox.HeaderBackgroundColor 2629
CheckLst.TCheckListBox.HeaderColor 2629
CheckLst.TCheckListBox.ImeMode 2630
CheckLst.TCheckListBox.ImeName 2630
CheckLst.TCheckListBox.IntegralHeight 2631
CheckLst.TCheckListBox.InternalGetItemData 2610
CheckLst.TCheckListBox.InternalSetItemData 2610
CheckLst.TCheckListBox.ItemEnabled 2631
CheckLst.TCheckListBox.ItemHeight 2632
CheckLst.TCheckListBox.Items 2632
CheckLst.TCheckListBox.KeyPress 2611
CheckLst.TCheckListBox.LoadRecreateItems 2611
CheckLst.TCheckListBox.MouseDown 2611
CheckLst.TCheckListBox.OnClick 2649
CheckLst.TCheckListBox.OnClickCheck 2651
CheckLst.TCheckListBox.OnContextPopup 2652
CheckLst.TCheckListBox.OnData 2652
CheckLst.TCheckListBox.OnDataFind 2653
CheckLst.TCheckListBox.OnDataObject 2653
CheckLst.TCheckListBox.OnDblClick 2654
CheckLst.TCheckListBox.OnDragDrop 2654
CheckLst.TCheckListBox.OnDragOver 2656
CheckLst.TCheckListBox.OnDrawItem 2659
CheckLst.TCheckListBox.OnEndDock 2661
CheckLst.TCheckListBox.OnEndDrag 2661
CheckLst.TCheckListBox.OnEnter 2662
CheckLst.TCheckListBox.OnExit 2665
CheckLst.TCheckListBox.OnKeyDown 2666
CheckLst.TCheckListBox.OnKeyPress 2669
CheckLst.TCheckListBox.OnKeyUp 2670
CheckLst.TCheckListBox.OnMeasureItem 2672
CheckLst.TCheckListBox.OnMouseActivate 2673

CheckLst.TCheckListBox.OnMouseDown 2674
CheckLst.TCheckListBox.OnMouseEnter 2677
CheckLst.TCheckListBox.OnMouseLeave 2677
CheckLst.TCheckListBox.OnMouseMove 2677
CheckLst.TCheckListBox.OnMouseUp 2679
CheckLst.TCheckListBox.OnStartDock 2682
CheckLst.TCheckListBox.OnStartDrag 2682
CheckLst.TCheckListBox.ParentBiDiMode 2635
CheckLst.TCheckListBox.ParentColor 2635
CheckLst.TCheckListBox.ParentCtl3D 2636
CheckLst.TCheckListBox.ParentFont 2636
CheckLst.TCheckListBox.ParentShowHint 2637
CheckLst.TCheckListBox.PopupMenu 2637
CheckLst.TCheckListBox.ResetContent 2612
CheckLst.TCheckListBox.SaveRecreateItems 2612
CheckLst.TCheckListBox.SetItemData 2612
CheckLst.TCheckListBox.ShowHint 2642
CheckLst.TCheckListBox.Sorted 2643
CheckLst.TCheckListBox.State 2645
CheckLst.TCheckListBox.Style 2645
CheckLst.TCheckListBox.TabOrder 2646
CheckLst.TCheckListBox.TabStop 2647
CheckLst.TCheckListBox.TabWidth 2647
CheckLst.TCheckListBox.Visible 2647
ciInternal enumeration member 5604
ciMultiInstance enumeration member 5604
ciSingleInstance enumeration member 5604
ckAttachToInterface enumeration member 18117
ckMulti enumeration member 1714
ckNewInstance enumeration member 18117
ckRemote enumeration member 18117
ckRunningInstance enumeration member 18117
ckRunningOrNew enumeration member 18117
ckSingle enumeration member 1714
Classes 2685
Classes namespace 2685
    Classes 2692
    Constants 3316
    Functions 3214
    Interfaces 3195
    Structs, Records, Enums 3268
    Types 3282
    Variables 3309
Classes.ActivateClassGroup 3215
Classes.ActivateClassGroup function 3215
Classes.ActiveClassGroup 3216
Classes.ActiveClassGroup function 3216
Classes.AddDataModule 3309
Classes.AddDataModule variable 3309
Classes.AllocateHWnd 3217
Classes.AllocateHWnd function 3217
Classes.AncestorIsValid 3217
Classes.AncestorIsValid function 3217
Classes.ApplicationHandleException 3310
Classes.ApplicationHandleException variable 3310
Classes.ApplicationShowException 3310
Classes.ApplicationShowException variable 3310
Classes.BeginGlobalLoading 3218
Classes.BeginGlobalLoading function 3218
Classes.BinToHex 3218
Classes.BinToHex function 3218
Classes.Bounds 3218
Classes.Bounds function 3218
Classes.CheckSynchronize 3219
Classes.CheckSynchronize function 3219
Classes.ClassGroupOf 3220
Classes.ClassGroupOf function 3220
Classes.CollectionsEqual 3220
Classes.CollectionsEqual function 3220
Classes.CountGenerations 3221
Classes.CountGenerations function 3221
Classes.CreateVCLComObjectProc 3310
Classes.CreateVCLComObjectProc variable 3310
Classes.CurrentGroup 3311
Classes.CurrentGroup variable 3311
Classes.DeallocateHWnd 3221
Classes.DeallocateHWnd function 3221
Classes.EBitsError 2693

Classes.EClassNotFound 2694
Classes.EComponentError 2694
Classes.EFCreateError 2695
Classes.EFilerError 2696
Classes.EFileStreamError 2697
Classes.EFileStreamError.Create 2698
Classes.EFOpenError 2699
Classes.EInvalidImage 2699
Classes.EInvalidOperation 2700
Classes.EListError 2701
Classes.EMethodNotFound 2701
Classes.EndGlobalLoading 3222
Classes.EndGlobalLoading function 3222
Classes.EOutOfResources 2702
Classes.EParserError 2702
Classes.EReadError 2703
Classes.EResNotFound 2704
Classes.EStreamError 2704
Classes.EStringListError 2705
Classes.EThread 2706
Classes.EWriteError 2706
Classes.ExtractStrings 3222
Classes.ExtractStrings function 3222
Classes.FindClass 3223
Classes.FindClass function 3223
Classes.FindGlobalComponent 3223
Classes.FindGlobalComponent function 3223
Classes.FindIdentToInt 3224
Classes.FindIdentToInt function 3224
Classes.FindIntToIdent 3224
Classes.FindIntToIdent function 3224
Classes.FindNestedComponent 3225
Classes.FindNestedComponent function 3225
Classes.FindRootDesigner 3225
Classes.FindRootDesigner function 3225
Classes.fmCreate 3316
Classes.fmCreate constant 3316
Classes.FreeObjectInstance 3226
Classes.FreeObjectInstance function 3226
Classes.GetClass 3226
Classes.GetClass function 3226
Classes.GetFixupInstanceNames 3228
Classes.GetFixupInstanceNames function 3228
Classes.GetFixupReferenceNames 3228
Classes.GetFixupReferenceNames function 3228
Classes.GetUltimateOwner 3228
Classes.GetUltimateOwner function 3228
Classes.GlobalFixupReferences 3229
Classes.GlobalFixupReferences function 3229
Classes.GlobalNameSpace 3311
Classes.GlobalNameSpace variable 3311
Classes.GroupDescendentsWith 3229
Classes.GroupDescendentsWith function 3229
Classes.HexToBin 3230
Classes.HexToBin function 3230
Classes.IdentToInt 3230
Classes.IdentToInt function 3230
Classes.IDesignerNotify 3195
Classes.IDesignerNotify.Modified 3196
Classes.IDesignerNotify.Notification 3196
Classes.IInterfaceComponentReference 3197
Classes.IInterfaceComponentReference.GetComponent 3197
Classes.IInterfaceList 3198
Classes.IInterfaceList.Add 3199
Classes.IInterfaceList.Capacity 3205
Classes.IInterfaceList.Clear 3200
Classes.IInterfaceList.Count 3206
Classes.IInterfaceList.Delete 3200
Classes.IInterfaceList.Exchange 3201
Classes.IInterfaceList.First 3201
Classes.IInterfaceList.Get 3201
Classes.IInterfaceList.GetCapacity 3202
Classes.IInterfaceList.GetCount 3202
Classes.IInterfaceList.IndexOf 3202
Classes.IInterfaceList.Insert 3202
Classes.IInterfaceList.Items 3206
Classes.IInterfaceList.Last 3203
Classes.IInterfaceList.Lock 3203

Classes.IInterfaceList.Put 3204
Classes.IInterfaceList.Remove 3204
Classes.IInterfaceList.SetCapacity 3204
Classes.IInterfaceList.SetCount 3205
Classes.IInterfaceList.Unlock 3205
Classes.InitComponentRes 3231
Classes.InitComponentRes function 3231
Classes.InitInheritedComponent 3231
Classes.InitInheritedComponent function 3231
Classes.IntToIdent 3232
Classes.IntToIdent function 3232
Classes.InvalidPoint 3233
Classes.InvalidPoint function 3233
Classes.IsDefaultPropertyValue 3233
Classes.IsDefaultPropertyValue function 3233
Classes.IStreamPersist 3207
Classes.IStreamPersist.LoadFromStream 3208
Classes.IStreamPersist.SaveToStream 3208
Classes.IStringsAdapter 3208
Classes.IStringsAdapter.ReferenceStrings 3209
Classes.IStringsAdapter.ReleaseStrings 3209
Classes.IsUniqueGlobalComponentName 3234
Classes.IsUniqueGlobalComponentName function 3234
Classes.IsUniqueGlobalComponentNameProc 3311
Classes.IsUniqueGlobalComponentNameProc variable 3311
Classes.IVarStreamable 3210
Classes.IVarStreamable.StreamIn 3211
Classes.IVarStreamable.StreamOut 3211
Classes.IVCLComObject 3211
Classes.IVCLComObject.FreeOnRelease 3212
Classes.IVCLComObject.GetIDsOfNames 3212
Classes.IVCLComObject.GetTypeInfo 3213
Classes.IVCLComObject.GetTypeInfoCount 3213
Classes.IVCLComObject.Invoke 3213
Classes.IVCLComObject.SafeCallException 3213
Classes.LineStart 3234
Classes.LineStart function 3234
Classes.MakeObjectInstance 3235
Classes.MakeObjectInstance function 3235
Classes.MaxListSize 3317
Classes.MaxListSize constant 3317
Classes.NotifyGlobalLoading 3235
Classes.NotifyGlobalLoading function 3235
Classes.ObjectBinaryToText 3235
Classes.ObjectBinaryToText function 3235
Classes.ObjectResourceToText 3239
Classes.ObjectResourceToText function 3239
Classes.ObjectTextToBinary 3240
Classes.ObjectTextToBinary function 3240
Classes.ObjectTextToResource 3244
Classes.ObjectTextToResource function 3244
Classes.Point 3244
Classes.Point function 3244
Classes.PointsEqual 3245
Classes.PointsEqual function 3245
Classes.PPointerList 3283
Classes.PPointerList type 3283
Classes.PStringItem 3284
Classes.PStringItem type 3284
Classes.PStringItemList 3284
Classes.PStringItemList type 3284
Classes.PSynchronizeRecord 3285
Classes.PSynchronizeRecord type 3285
Classes.ReadComponentRes 3245
Classes.ReadComponentRes function 3245
Classes.ReadComponentResEx 3246
Classes.ReadComponentResEx function 3246
Classes.ReadComponentResFile 3247
Classes.ReadComponentResFile function 3247
Classes.Rect 3249
Classes.Rect function 3249
Classes.RedirectFixupReferences 3250
Classes.RedirectFixupReferences function 3250
Classes.RegisterClass 3250
Classes.RegisterClass function 3250
Classes.RegisterClassAlias 3254
Classes.RegisterClassAlias function 3254
Classes.RegisterClasses 3255

Classes.RegisterClasses function 3255
Classes.RegisterComponents 3256
Classes.RegisterComponents function 3256
Classes.RegisterComponentsProc 3312
Classes.RegisterComponentsProc variable 3312
Classes.RegisterFindGlobalComponentProc 3259
Classes.RegisterFindGlobalComponentProc function 3259
Classes.RegisterIntegerConsts 3259
Classes.RegisterIntegerConsts function 3259
Classes.RegisterNoIcon 3260
Classes.RegisterNoIcon function 3260
Classes.RegisterNoIconProc 3313
Classes.RegisterNoIconProc variable 3313
Classes.RegisterNonActiveX 3260
Classes.RegisterNonActiveX function 3260
Classes.RegisterNonActiveXProc 3313
Classes.RegisterNonActiveXProc variable 3313
Classes.RemoveDataModule 3314
Classes.RemoveDataModule variable 3314
Classes.RemoveFixupReferences 3261
Classes.RemoveFixupReferences function 3261
Classes.RemoveFixups 3261
Classes.RemoveFixups function 3261
Classes.scAlt 3317
Classes.scAlt constant 3317
Classes.scCtrl 3318
Classes.scCtrl constant 3318
Classes.scNone 3318
Classes.scNone constant 3318
Classes.scShift 3319
Classes.scShift constant 3319
Classes.SmallPoint 3262
Classes.SmallPoint function 3262
Classes.soFromBeginning 3319
Classes.soFromBeginning constant 3319
Classes.soFromCurrent 3320
Classes.soFromCurrent constant 3320
Classes.soFromEnd 3320
Classes.soFromEnd constant 3320
Classes.StartClassGroup 3262
Classes.StartClassGroup function 3262
Classes.TActiveXRegType 3268
Classes.TActiveXRegType enumeration 3268
Classes.TAlignment 3269
Classes.TAlignment enumeration 3269
Classes.TAncestorNotFoundEvent 3285
Classes.TAncestorNotFoundEvent type 3285
Classes.TBasicAction 2707
Classes.TBasicAction.ActionComponent 2715
Classes.TBasicAction.Change 2712
Classes.TBasicAction.Create 2710
Classes.TBasicAction.Destroy 2711
Classes.TBasicAction.Execute 2712
Classes.TBasicAction.ExecuteTarget 2712
Classes.TBasicAction.FClients 2711
Classes.TBasicAction.HandlesTarget 2713
Classes.TBasicAction.Notification 2713
Classes.TBasicAction.OnChange 2716
Classes.TBasicAction.OnExecute 2716
Classes.TBasicAction.OnUpdate 2718
Classes.TBasicAction.RegisterChanges 2713
Classes.TBasicAction.SetOnExecute 2714
Classes.TBasicAction.UnRegisterChanges 2714
Classes.TBasicAction.Update 2714
Classes.TBasicAction.UpdateTarget 2715
Classes.TBasicActionClass 3285
Classes.TBasicActionClass type 3285
Classes.TBasicActionLink 2720
Classes.TBasicActionLink.Action 2725
Classes.TBasicActionLink.AssignClient 2723
Classes.TBasicActionLink.Change 2723
Classes.TBasicActionLink.Create 2721
Classes.TBasicActionLink.Destroy 2722
Classes.TBasicActionLink.Execute 2723
Classes.TBasicActionLink.FAction 2722
Classes.TBasicActionLink.IsOnExecuteLinked 2723
Classes.TBasicActionLink.OnChange 2725
Classes.TBasicActionLink.SetAction 2724

Classes.TBasicActionLink.SetOnExecute 2724
Classes.TBasicActionLink.Update 2724
Classes.TBasicActionLinkClass 3286
Classes.TBasicActionLinkClass type 3286
Classes.TBiDiMode 3269
Classes.TBiDiMode enumeration 3269
Classes.TBits 2725
Classes.TBits.Bits 2727
Classes.TBits.Destroy 2726
Classes.TBits.OpenBit 2727
Classes.TBits.Size 2728
Classes.TClassFinder 2728
Classes.TClassFinder.Create 2729
Classes.TClassFinder.Destroy 2729
Classes.TClassFinder.GetClass 2730
Classes.TClassFinder.GetClasses 2731
Classes.TCollection 2731
Classes.TCollection.Add 2736
Classes.TCollection.Added 2736
Classes.TCollection.Assign 2736
Classes.TCollection.BeginUpdate 2739
Classes.TCollection.Capacity 2748
Classes.TCollection.Changed 2740
Classes.TCollection.Clear 2741
Classes.TCollection.Count 2749
Classes.TCollection.Create 2734
Classes.TCollection.Delete 2741
Classes.TCollection.Deleting 2741
Classes.TCollection.Destroy 2735
Classes.TCollection.EndUpdate 2742
Classes.TCollection.FindItemID 2743
Classes.TCollection.GetAttr 2743
Classes.TCollection.GetAttrCount 2744
Classes.TCollection.GetEnumerator 2744
Classes.TCollection.GetItem 2744
Classes.TCollection.GetItemAttr 2745
Classes.TCollection.GetNamePath 2745
Classes.TCollection.Insert 2746
Classes.TCollection.ItemClass 2751
Classes.TCollection.Items 2751
Classes.TCollection.NextID 2752
Classes.TCollection.Notify 2746
Classes.TCollection.Owner 2746
Classes.TCollection.PropName 2752
Classes.TCollection.SetItem 2747
Classes.TCollection.SetItemName 2747
Classes.TCollection.Update 2747
Classes.TCollection.UpdateCount 2753
Classes.TCollectionEnumerator 2753
Classes.TCollectionEnumerator.Create 2754
Classes.TCollectionEnumerator.Current 2755
Classes.TCollectionEnumerator.GetCurrent 2754
Classes.TCollectionEnumerator.MoveNext 2754
Classes.TCollectionItem 2755
Classes.TCollectionItem.Changed 2758
Classes.TCollectionItem.Collection 2761
Classes.TCollectionItem.Create 2757
Classes.TCollectionItem.Destroy 2757
Classes.TCollectionItem.DisplayName 2761
Classes.TCollectionItem.GetDisplayName 2758
Classes.TCollectionItem.GetNamePath 2759
Classes.TCollectionItem.GetOwner 2759
Classes.TCollectionItem.ID 2761
Classes.TCollectionItem.Index 2762
Classes.TCollectionItem.SetCollection 2759
Classes.TCollectionItem.SetDisplayName 2760
Classes.TCollectionItem.SetIndex 2760
Classes.TCollectionItemClass 3286
Classes.TCollectionItemClass type 3286
Classes.TCollectionNotification 3270
Classes.TCollectionNotification enumeration 3270
Classes.TComponent 2762
Classes.TComponent._AddRef 2769
Classes.TComponent._Release 2769
Classes.TComponent.BeforeDestruction 2770
Classes.TComponent.ChangeName 2770
Classes.TComponent.ComObject 2793
Classes.TComponent.ComponentCount 2794

Classes.TComponent.ComponentIndex 2799
Classes.TComponent.Components 2799
Classes.TComponent.ComponentState 2804
Classes.TComponent.ComponentStyle 2805
Classes.TComponent.Create 2766
Classes.TComponent.DefineProperties 2771
Classes.TComponent.DesignInfo 2805
Classes.TComponent.Destroy 2766
Classes.TComponent.DestroyComponents 2771
Classes.TComponent.Destroying 2772
Classes.TComponent.ExecuteAction 2772
Classes.TComponent.FComponentStyle 2767
Classes.TComponent.FindComponent 2772
Classes.TComponent.FreeNotification 2775
Classes.TComponent.FreeOnRelease 2776
Classes.TComponent.GetChildOwner 2776
Classes.TComponent.GetChildParent 2776
Classes.TComponent.GetChildren 2777
Classes.TComponent.GetEnumerator 2777
Classes.TComponent.GetIDsOfNames 2777
Classes.TComponent.GetNamePath 2778
Classes.TComponent.GetOwner 2778
Classes.TComponent.GetParentComponent 2778
Classes.TComponent.GetTypeInfo 2779
Classes.TComponent.GetTypeInfoCount 2779
Classes.TComponent.HasParent 2780
Classes.TComponent.InsertComponent 2780
Classes.TComponent.Invoke 2781
Classes.TComponent.IsImplementorOf 2782
Classes.TComponent.Loaded 2782
Classes.TComponent.Name 2805
Classes.TComponent.Notification 2783
Classes.TComponent.Owner 2807
Classes.TComponent.PaletteCreated 2783
Classes.TComponent.QueryInterface 2784
Classes.TComponent.ReadState 2784
Classes.TComponent.ReferenceInterface 2785
Classes.TComponent.RemoveComponent 2785
Classes.TComponent.RemoveFreeNotification 2787
Classes.TComponent.SafeCallException 2787
Classes.TComponent.SetAncestor 2787
Classes.TComponent.SetChildOrder 2788
Classes.TComponent.SetDesigning 2788
Classes.TComponent.SetDesignInstance 2788
Classes.TComponent.SetInline 2789
Classes.TComponent.SetName 2789
Classes.TComponent.SetParentComponent 2790
Classes.TComponent.SetSubComponent 2790
Classes.TComponent.Tag 2808
Classes.TComponent.UpdateAction 2790
Classes.TComponent.Updated 2791
Classes.TComponent.UpdateRegistry 2791
Classes.TComponent.Updating 2791
Classes.TComponent.ValidateContainer 2792
Classes.TComponent.ValidateInsert 2792
Classes.TComponent.ValidateRename 2792
Classes.TComponent.VCLComObject 2812
Classes.TComponent.WriteState 2793
Classes.TComponentClass 3287
Classes.TComponentClass type 3287
Classes.TComponentEnumerator 2812
Classes.TComponentEnumerator.Create 2813
Classes.TComponentEnumerator.Current 2814
Classes.TComponentEnumerator.GetCurrent 2813
Classes.TComponentEnumerator.MoveNext 2814
Classes.TComponentName 3287
Classes.TComponentName type 3287
Classes.TComponentState 3287
Classes.TComponentState type 3287
Classes.TComponentStyle 3288
Classes.TComponentStyle type 3288
Classes.TCreateComponentEvent 3289
Classes.TCreateComponentEvent type 3289
Classes.TCustomMemoryStream 2815
Classes.TCustomMemoryStream.Memory 2823
Classes.TCustomMemoryStream.Read 2817
Classes.TCustomMemoryStream.SaveToFile 2818
Classes.TCustomMemoryStream.SaveToStream 2818

Classes.TCustomMemoryStream.Seek 2819
Classes.TCustomMemoryStream.SetPointer 2823
Classes.TDataModule 2824
Classes.TDataModule.AfterConstruction 2829
Classes.TDataModule.BeforeDestruction 2830
Classes.TDataModule.Create 2827
Classes.TDataModule.CreateNew 2828
Classes.TDataModule.DefineProperties 2830
Classes.TDataModule.DesignOffset 2832
Classes.TDataModule.DesignSize 2832
Classes.TDataModule.Destroy 2829
Classes.TDataModule.DoCreate 2831
Classes.TDataModule.DoDestroy 2831
Classes.TDataModule.GetChildren 2831
Classes.TDataModule.HandleCreateException 2831
Classes.TDataModule.OldCreateOrder 2833
Classes.TDataModule.OnCreate 2833
Classes.TDataModule.OnDestroy 2834
Classes.TDataModule.ReadState 2832
Classes.TDuplicates 3271
Classes.TDuplicates enumeration 3271
Classes.TestStreamFormat 3263
Classes.TestStreamFormat function 3263
Classes.TFiler 2834
Classes.TFiler.Ancestor 2839
Classes.TFiler.Create 2836
Classes.TFiler.DefineBinaryProperty 2837
Classes.TFiler.DefineProperty 2838
Classes.TFiler.Destroy 2837
Classes.TFiler.FlushBuffer 2838
Classes.TFiler.IgnoreChildren 2839
Classes.TFiler.LookupRoot 2840
Classes.TFiler.Root 2840
Classes.TFiler.SetRoot 2838
Classes.TFilerFlag 3271
Classes.TFilerFlag enumeration 3271
Classes.TFilerFlags 3289
Classes.TFilerFlags type 3289
Classes.TFileStream 2841
Classes.TFileStream.Create 2843
Classes.TFileStream.Destroy 2844
Classes.TFileStream.FFileName 2844
Classes.TFileStream.FileName 2845
Classes.TFindAncestorEvent 3290
Classes.TFindAncestorEvent type 3290
Classes.TFindComponentClassEvent 3290
Classes.TFindComponentClassEvent type 3290
Classes.TFindComponentInstanceEvent 3291
Classes.TFindComponentInstanceEvent type 3291
Classes.TFindGlobalComponent 3291
Classes.TFindGlobalComponent type 3291
Classes.TFindMethodEvent 3291
Classes.TFindMethodEvent type 3291
Classes.TFindMethodInstanceEvent 3292
Classes.TFindMethodInstanceEvent type 3292
Classes.TFindMethodNameEvent 3292
Classes.TFindMethodNameEvent type 3292
Classes.TGetChildProc 3292
Classes.TGetChildProc type 3292
Classes.TGetClass 3293
Classes.TGetClass type 3293
Classes.TGetLookupInfoEvent 3293
Classes.TGetLookupInfoEvent type 3293
Classes.TGetModuleProc 3294
Classes.TGetModuleProc type 3294
Classes.TGetStrProc 3294
Classes.TGetStrProc type 3294
Classes.THandleStream 2845
Classes.THandleStream.Create 2847
Classes.THandleStream.FHandle 2848
Classes.THandleStream.Handle 2850
Classes.THandleStream.Read 2848
Classes.THandleStream.Seek 2849
Classes.THandleStream.SetSize 2849
Classes.THandleStream.Write 2850
Classes.THelpContext 3295
Classes.THelpContext type 3295
Classes.THelpType 3272

Classes.THelpType enumeration 3272
Classes.TIdentMapEntry 3273
Classes.TIdentMapEntry record 3273
Classes.TIdentToInt 3295
Classes.TIdentToInt type 3295
Classes.TInterfacedPersistent 2851
Classes.TInterfacedPersistent._AddRef 2853
Classes.TInterfacedPersistent._Release 2854
Classes.TInterfacedPersistent.AfterConstruction 2854
Classes.TInterfacedPersistent.QueryInterface 2854
Classes.TInterfaceList 2855
Classes.TInterfaceList.Add 2858
Classes.TInterfaceList.Capacity 2865
Classes.TInterfaceList.Clear 2858
Classes.TInterfaceList.Count 2865
Classes.TInterfaceList.Create 2857
Classes.TInterfaceList.Delete 2859
Classes.TInterfaceList.Destroy 2857
Classes.TInterfaceList.Exchange 2859
Classes.TInterfaceList.Expand 2859
Classes.TInterfaceList.First 2860
Classes.TInterfaceList.Get 2860
Classes.TInterfaceList.GetCapacity 2860
Classes.TInterfaceList.GetCount 2861
Classes.TInterfaceList.GetEnumerator 2861
Classes.TInterfaceList.IndexOf 2861
Classes.TInterfaceList.Insert 2862
Classes.TInterfaceList.Items 2866
Classes.TInterfaceList.Last 2862
Classes.TInterfaceList.Lock 2862
Classes.TInterfaceList.Put 2863
Classes.TInterfaceList.Remove 2863
Classes.TInterfaceList.SetCapacity 2864
Classes.TInterfaceList.SetCount 2864
Classes.TInterfaceList.Unlock 2864
Classes.TInterfaceListEnumerator 2866
Classes.TInterfaceListEnumerator.Create 2867
Classes.TInterfaceListEnumerator.Current 2868
Classes.TInterfaceListEnumerator.GetCurrent 2867
Classes.TInterfaceListEnumerator.MoveNext 2868
Classes.TIntToIdent 3296
Classes.TIntToIdent type 3296
Classes.TIsUniqueGlobalComponentName 3296
Classes.TIsUniqueGlobalComponentName type 3296
Classes.TLeftRight 3297
Classes.TLeftRight type 3297
Classes.TList 2868
Classes.TList.Add 2871
Classes.TList.Assign 2874
Classes.TList.Capacity 2895
Classes.TList.Clear 2874
Classes.TList.Count 2896
Classes.TList.Delete 2874
Classes.TList.Destroy 2870
Classes.TList.Error 2875
Classes.TList.Exchange 2876
Classes.TList.Expand 2877
Classes.TList.Extract 2877
Classes.TList.First 2878
Classes.TList.Get 2878
Classes.TList.GetEnumerator 2878
Classes.TList.Grow 2879
Classes.TList.IndexOf 2879
Classes.TList.Insert 2882
Classes.TList.Items 2898
Classes.TList.Last 2882
Classes.TList.List 2900
Classes.TList.Move 2884
Classes.TList.Notify 2884
Classes.TList.Pack 2885
Classes.TList.Put 2887
Classes.TList.Remove 2887
Classes.TList.SetCapacity 2891
Classes.TList.SetCount 2892
Classes.TList.Sort 2892
Classes.TListAssignOp 3273
Classes.TListAssignOp enumeration 3273
Classes.TListEnumerator 2900

Classes.TListEnumerator.Create 2901
Classes.TListEnumerator.Current 2902
Classes.TListEnumerator.GetCurrent 2901
Classes.TListEnumerator.MoveNext 2902
Classes.TListNotification 3274
Classes.TListNotification enumeration 3274
Classes.TListSortCompare 3297
Classes.TListSortCompare type 3297
Classes.TMemoryStream 2902
Classes.TMemoryStream.Capacity 2912
Classes.TMemoryStream.Clear 2909
Classes.TMemoryStream.Destroy 2908
Classes.TMemoryStream.LoadFromFile 2909
Classes.TMemoryStream.LoadFromStream 2909
Classes.TMemoryStream.Realloc 2910
Classes.TMemoryStream.SetSize 2910
Classes.TMemoryStream.Write 2911
Classes.TNotifyEvent 3298
Classes.TNotifyEvent type 3298
Classes.toEOF 3321
Classes.toEOF constant 3321
Classes.toFloat 3322
Classes.toFloat constant 3322
Classes.toInteger 3322
Classes.toInteger constant 3322
Classes.TOperation 3275
Classes.TOperation enumeration 3275
Classes.toString 3323
Classes.toString constant 3323
Classes.toSymbol 3324
Classes.toSymbol constant 3324
Classes.TOwnedCollection 2912
Classes.TOwnedCollection.Create 2914
Classes.TOwnedCollection.GetOwner 2915
Classes.toWString 3324
Classes.toWString constant 3324
Classes.TParser 2915
Classes.TParser.CheckToken 2919
Classes.TParser.CheckTokenSymbol 2919
Classes.TParser.Create 2917
Classes.TParser.Destroy 2917
Classes.TParser.Error 2920
Classes.TParser.ErrorFmt 2920
Classes.TParser.ErrorStr 2921
Classes.TParser.FloatType 2925
Classes.TParser.GetLinePos 2921
Classes.TParser.HexToBinary 2921
Classes.TParser.LinePos 2926
Classes.TParser.NextToken 2922
Classes.TParser.OnError 2927
Classes.TParser.SourceLine 2926
Classes.TParser.SourcePos 2922
Classes.TParser.Token 2926
Classes.TParser.TokenComponentIdent 2923
Classes.TParser.TokenFloat 2923
Classes.TParser.TokenInt 2923
Classes.TParser.TokenString 2924
Classes.TParser.TokenSymbolIs 2924
Classes.TParser.TokenWideString 2925
Classes.TParserErrorEvent 3298
Classes.TParserErrorEvent type 3298
Classes.TPersistent 2928
Classes.TPersistent.Assign 2930
Classes.TPersistent.AssignTo 2931
Classes.TPersistent.DefineProperties 2931
Classes.TPersistent.Destroy 2929
Classes.TPersistent.GetNamePath 2932
Classes.TPersistent.GetOwner 2932
Classes.TPersistentClass 3299
Classes.TPersistentClass type 3299
Classes.TPointerList 3299
Classes.TPointerList type 3299
Classes.TReadComponentsProc 3300
Classes.TReadComponentsProc type 3300
Classes.TReader 2933
Classes.TReader.BeginReferences 2938
Classes.TReader.CanHandleExceptions 2958
Classes.TReader.CheckValue 2939

Classes.TReader.CopyValue 2939
Classes.TReader.DefineBinaryProperty 2939
Classes.TReader.DefineProperty 2940
Classes.TReader.Destroy 2936
Classes.TReader.EndOfList 2941
Classes.TReader.EndReferences 2941
Classes.TReader.Error 2942
Classes.TReader.FindAncestorComponent 2942
Classes.TReader.FindMethod 2942
Classes.TReader.FindMethodInstance 2942
Classes.TReader.FixupReferences 2942
Classes.TReader.FlushBuffer 2943
Classes.TReader.NextValue 2943
Classes.TReader.OnAncestorNotFound 2961
Classes.TReader.OnCreateComponent 2961
Classes.TReader.OnError 2961
Classes.TReader.OnFindComponentClass 2962
Classes.TReader.OnFindComponentInstance 2962
Classes.TReader.OnFindMethod 2962
Classes.TReader.OnFindMethodInstance 2963
Classes.TReader.OnReferenceName 2963
Classes.TReader.OnSetName 2963
Classes.TReader.Owner 2959
Classes.TReader.Parent 2959
Classes.TReader.Position 2960
Classes.TReader.PropertyError 2944
Classes.TReader.PropName 2960
Classes.TReader.Read 2944
Classes.TReader.ReadBoolean 2944
Classes.TReader.ReadChar 2945
Classes.TReader.ReadCollection 2945
Classes.TReader.ReadComponent 2945
Classes.TReader.ReadComponents 2946
Classes.TReader.ReadCurrency 2946
Classes.TReader.ReadData 2947
Classes.TReader.ReadDate 2947
Classes.TReader.ReadDouble 2947
Classes.TReader.ReadFloat 2948
Classes.TReader.ReadIdent 2948
Classes.TReader.ReadInt64 2949
Classes.TReader.ReadInteger 2949
Classes.TReader.ReadListBegin 2950
Classes.TReader.ReadListEnd 2950
Classes.TReader.ReadPrefix 2951
Classes.TReader.ReadProperty 2951
Classes.TReader.ReadPropValue 2952
Classes.TReader.ReadRootComponent 2952
Classes.TReader.ReadSet 2952
Classes.TReader.ReadSignature 2953
Classes.TReader.ReadSingle 2953
Classes.TReader.ReadStr 2953
Classes.TReader.ReadString 2954
Classes.TReader.ReadValue 2954
Classes.TReader.ReadVariant 2955
Classes.TReader.ReadWideChar 2955
Classes.TReader.ReadWideString 2956
Classes.TReader.ReferenceName 2956
Classes.TReader.SetName 2956
Classes.TReader.SetPosition 2957
Classes.TReader.SkipBytes 2957
Classes.TReader.SkipComponent 2957
Classes.TReader.SkipProperty 2957
Classes.TReader.SkipSetBody 2958
Classes.TReader.SkipValue 2958
Classes.TReaderError 3300
Classes.TReaderError type 3300
Classes.TReaderProc 3301
Classes.TReaderProc type 3301
Classes.TRecall 2964
Classes.TRecall.Create 2965
Classes.TRecall.Destroy 2966
Classes.TRecall.Forget 2967
Classes.TRecall.Reference 2968
Classes.TRecall.Store 2967
Classes.TReferenceNameEvent 3301
Classes.TReferenceNameEvent type 3301
Classes.TResourceStream 2968
Classes.TResourceStream.Create 2970

Classes.TResourceStream.CreateFromID 2971
Classes.TResourceStream.Destroy 2972
Classes.TResourceStream.Write 2973
Classes.TSeekOrigin 3276
Classes.TSeekOrigin enumeration 3276
Classes.TSetNameEvent 3302
Classes.TSetNameEvent type 3302
Classes.TShiftState 3302
Classes.TShiftState type 3302
Classes.TShortCut 3303
Classes.TShortCut type 3303
Classes.TStream 2973
Classes.TStream.CopyFrom 2979
Classes.TStream.FixupResourceHeader 2981
Classes.TStream.GetSize 2982
Classes.TStream.Position 2996
Classes.TStream.Read 2982
Classes.TStream.ReadBuffer 2983
Classes.TStream.ReadComponent 2983
Classes.TStream.ReadComponentRes 2987
Classes.TStream.ReadResHeader 2987
Classes.TStream.Seek 2988
Classes.TStream.SetSize 2989
Classes.TStream.Size 2997
Classes.TStream.Write 2989
Classes.TStream.WriteBuffer 2990
Classes.TStream.WriteComponent 2990
Classes.TStream.WriteComponentRes 2994
Classes.TStream.WriteDescendent 2994
Classes.TStream.WriteDescendentRes 2995
Classes.TStream.WriteResourceHeader 2995
Classes.TStreamAdapter 2997
Classes.TStreamAdapter.Clone 3000
Classes.TStreamAdapter.Commit 3001
Classes.TStreamAdapter.CopyTo 3001
Classes.TStreamAdapter.Create 2999
Classes.TStreamAdapter.Destroy 3000
Classes.TStreamAdapter.LockRegion 3002
Classes.TStreamAdapter.Read 3002
Classes.TStreamAdapter.Revert 3002
Classes.TStreamAdapter.Seek 3003
Classes.TStreamAdapter.SetSize 3003
Classes.TStreamAdapter.Stat 3004
Classes.TStreamAdapter.Stream 3005
Classes.TStreamAdapter.StreamOwnership 3005
Classes.TStreamAdapter.UnlockRegion 3004
Classes.TStreamAdapter.Write 3004
Classes.TStreamOriginalFormat 3276
Classes.TStreamOriginalFormat enumeration 3276
Classes.TStreamOwnership 3277
Classes.TStreamOwnership enumeration 3277
Classes.TStreamProc 3304
Classes.TStreamProc type 3304
Classes.TStringItem 3277
Classes.TStringItem record 3277
Classes.TStringItemList 3304
Classes.TStringItemList type 3304
Classes.TStringList 3006
Classes.TStringList.Add 3010
Classes.TStringList.AddObject 3012
Classes.TStringList.CaseSensitive 3025
Classes.TStringList.Changed 3013
Classes.TStringList.Changing 3013
Classes.TStringList.Clear 3013
Classes.TStringList.CompareStrings 3014
Classes.TStringList.CustomSort 3014
Classes.TStringList.Delete 3015
Classes.TStringList.Destroy 3009
Classes.TStringList.Duplicates 3025
Classes.TStringList.Exchange 3015
Classes.TStringList.Find 3016
Classes.TStringList.Get 3017
Classes.TStringList.GetCapacity 3017
Classes.TStringList.GetCount 3018
Classes.TStringList.GetObject 3018
Classes.TStringList.IndexOf 3019
Classes.TStringList.Insert 3019
Classes.TStringList.InsertItem 3020

Classes.TStringList.InsertObject 3020
Classes.TStringList.OnChange 3028
Classes.TStringList.OnChanging 3028
Classes.TStringList.Put 3022
Classes.TStringList.PutObject 3022
Classes.TStringList.SetCapacity 3022
Classes.TStringList.SetUpdateState 3023
Classes.TStringList.Sort 3023
Classes.TStringList.Sorted 3026
Classes.TStringListSortCompare 3305
Classes.TStringListSortCompare type 3305
Classes.TStrings 3029
Classes.TStrings.Add 3033
Classes.TStrings.AddObject 3036
Classes.TStrings.AddStrings 3039
Classes.TStrings.Append 3041
Classes.TStrings.Assign 3043
Classes.TStrings.BeginUpdate 3045
Classes.TStrings.Capacity 3070
Classes.TStrings.Clear 3047
Classes.TStrings.CommaText 3070
Classes.TStrings.CompareStrings 3048
Classes.TStrings.Count 3071
Classes.TStrings.DefineProperties 3049
Classes.TStrings.Delete 3049
Classes.TStrings.DelimitedText 3074
Classes.TStrings.Delimiter 3074
Classes.TStrings.Destroy 3032
Classes.TStrings.EndUpdate 3050
Classes.TStrings.Equals 3051
Classes.TStrings.Error 3052
Classes.TStrings.Exchange 3053
Classes.TStrings.ExtractName 3053
Classes.TStrings.Get 3053
Classes.TStrings.GetCapacity 3054
Classes.TStrings.GetCount 3054
Classes.TStrings.GetEnumerator 3054
Classes.TStrings.GetObject 3055
Classes.TStrings.GetText 3055
Classes.TStrings.GetTextStr 3056
Classes.TStrings.IndexOf 3056
Classes.TStrings.IndexOfName 3057
Classes.TStrings.IndexOfObject 3059
Classes.TStrings.Insert 3059
Classes.TStrings.InsertObject 3060
Classes.TStrings.LineBreak 3075
Classes.TStrings.LoadFromFile 3061
Classes.TStrings.LoadFromStream 3062
Classes.TStrings.Move 3063
Classes.TStrings.Names 3075
Classes.TStrings.NameValueSeparator 3077
Classes.TStrings.Objects 3077
Classes.TStrings.Put 3064
Classes.TStrings.PutObject 3065
Classes.TStrings.QuoteChar 3080
Classes.TStrings.SaveToFile 3065
Classes.TStrings.SaveToStream 3067
Classes.TStrings.SetCapacity 3068
Classes.TStrings.SetText 3068
Classes.TStrings.SetTextStr 3069
Classes.TStrings.SetUpdateState 3069
Classes.TStrings.StrictDelimiter 3080
Classes.TStrings.Strings 3080
Classes.TStrings.StringsAdapter 3084
Classes.TStrings.Text 3084
Classes.TStrings.UpdateCount 3085
Classes.TStrings.ValueFromIndex 3085
Classes.TStrings.Values 3086
Classes.TStringsDefined 3306
Classes.TStringsDefined type 3306
Classes.TStringsEnumerator 3086
Classes.TStringsEnumerator.Create 3087
Classes.TStringsEnumerator.Current 3088
Classes.TStringsEnumerator.GetCurrent 3087
Classes.TStringsEnumerator.MoveNext 3087
Classes.TStringStream 3088
Classes.TStringStream.Create 3093
Classes.TStringStream.DataString 3104

Classes.TStringStream.Read 3097
Classes.TStringStream.ReadString 3098
Classes.TStringStream.Seek 3098
Classes.TStringStream.SetSize 3102
Classes.TStringStream.Write 3103
Classes.TStringStream.WriteString 3103
Classes.TSynchronizeRecord 3278
Classes.TSynchronizeRecord record 3278
Classes.TThread 3108
Classes.TThread.AfterConstruction 3114
Classes.TThread.CheckThreadError 3115
Classes.TThread.Create 3110
Classes.TThread.Destroy 3113
Classes.TThread.DoTerminate 3115
Classes.TThread.Execute 3115
Classes.TThread.FatalException 3133
Classes.TThread.FreeOnTerminate 3134
Classes.TThread.Handle 3137
Classes.TThread.OnTerminate 3148
Classes.TThread.Priority 3138
Classes.TThread.Queue 3119
Classes.TThread.RemoveQueuedEvents 3120
Classes.TThread.Resume 3120
Classes.TThread.ReturnValue 3141
Classes.TThread.StaticQueue 3124
Classes.TThread.StaticSynchronize 3125
Classes.TThread.Suspend 3125
Classes.TThread.Suspended 3142
Classes.TThread.Synchronize 3126
Classes.TThread.Terminate 3126
Classes.TThread.Terminated 3142
Classes.TThread.ThreadID 3148
Classes.TThread.WaitFor 3132
Classes.TThreadList 3149
Classes.TThreadList.Add 3151
Classes.TThreadList.Clear 3157
Classes.TThreadList.Create 3150
Classes.TThreadList.Destroy 3150
Classes.TThreadList.Duplicates 3175
Classes.TThreadList.LockList 3157
Classes.TThreadList.Remove 3163
Classes.TThreadList.UnlockList 3169
Classes.TThreadMethod 3306
Classes.TThreadMethod type 3306
Classes.TThreadPriority 3278
Classes.TThreadPriority enumeration 3278
Classes.TTopBottom 3306
Classes.TTopBottom type 3306
Classes.TValueType 3279
Classes.TValueType enumeration 3279
Classes.TVerticalAlignment 3280
Classes.TVerticalAlignment enumeration 3280
Classes.TWndMethod 3307
Classes.TWndMethod type 3307
Classes.TWriter 3175
Classes.TWriter.DefineBinaryProperty 3179
Classes.TWriter.DefineProperty 3180
Classes.TWriter.Destroy 3178
Classes.TWriter.FindMethodName 3181
Classes.TWriter.FlushBuffer 3181
Classes.TWriter.OnFindAncestor 3193
Classes.TWriter.OnFindMethodName 3194
Classes.TWriter.Position 3192
Classes.TWriter.RootAncestor 3192
Classes.TWriter.SetRoot 3181
Classes.TWriter.UseQualifiedNames 3193
Classes.TWriter.Write 3181
Classes.TWriter.WriteBinary 3182
Classes.TWriter.WriteBoolean 3182
Classes.TWriter.WriteChar 3182
Classes.TWriter.WriteCollection 3183
Classes.TWriter.WriteComponent 3183
Classes.TWriter.WriteCurrency 3184
Classes.TWriter.WriteDate 3184
Classes.TWriter.WriteDescendent 3184
Classes.TWriter.WriteDouble 3185
Classes.TWriter.WriteFloat 3185
Classes.TWriter.WriteIdent 3186

Classes.TWriter.WriteInteger 3186
Classes.TWriter.WriteListBegin 3187
Classes.TWriter.WriteListEnd 3187
Classes.TWriter.WritePrefix 3187
Classes.TWriter.WriteProperties 3188
Classes.TWriter.WriteProperty 3188
Classes.TWriter.WritePropName 3188
Classes.TWriter.WriteRootComponent 3188
Classes.TWriter.WriteSignature 3189
Classes.TWriter.WriteSingle 3189
Classes.TWriter.WriteStr 3189
Classes.TWriter.WriteString 3190
Classes.TWriter.WriteValue 3190
Classes.TWriter.WriteVariant 3190
Classes.TWriter.WriteWideChar 3191
Classes.TWriter.WriteWideString 3191
Classes.TWriterProc 3307
Classes.TWriterProc type 3307
Classes.UnRegisterClass 3263
Classes.UnRegisterClass function 3263
Classes.UnRegisterClasses 3264
Classes.UnRegisterClasses function 3264
Classes.UnregisterFindGlobalComponentProc 3264
Classes.UnregisterFindGlobalComponentProc function 3264
Classes.UnregisterIntegerConsts 3264
Classes.UnregisterIntegerConsts function 3264
Classes.UnRegisterModuleClasses 3265
Classes.UnRegisterModuleClasses function 3265
Classes.WakeMainThread 3314
Classes.WakeMainThread variable 3314
Classes.WriteComponentResFile 3266
Classes.WriteComponentResFile function 3266
clDefault enumeration member 26637
clFastest enumeration member 26637
Clipbrd 3327
Clipbrd namespace 3327
    Classes 3328
    Functions 3351
    Variables 3354
Clipbrd.CF_COMPONENT 3354
Clipbrd.CF_COMPONENT variable 3354
Clipbrd.CF_PICTURE 3354
Clipbrd.CF_PICTURE variable 3354
Clipbrd.Clipboard 3351
Clipbrd.Clipboard function 3351
Clipbrd.SetClipboard 3353
Clipbrd.SetClipboard function 3353
Clipbrd.TClipboard 3328
Clipbrd.TClipboard.Assign 3331
Clipbrd.TClipboard.AssignTo 3333
Clipbrd.TClipboard.AsText 3347
Clipbrd.TClipboard.Clear 3333
Clipbrd.TClipboard.Close 3336
Clipbrd.TClipboard.Destroy 3330
Clipbrd.TClipboard.FormatCount 3348
Clipbrd.TClipboard.Formats 3348
Clipbrd.TClipboard.GetAsHandle 3336
Clipbrd.TClipboard.GetComponent 3339
Clipbrd.TClipboard.GetTextBuf 3340
Clipbrd.TClipboard.Handle 3349
Clipbrd.TClipboard.HasFormat 3340
Clipbrd.TClipboard.MainWndProc 3342
Clipbrd.TClipboard.Open 3342
Clipbrd.TClipboard.OpenRefCount 3349
Clipbrd.TClipboard.SetAsHandle 3343
Clipbrd.TClipboard.SetBuffer 3345
Clipbrd.TClipboard.SetComponent 3345
Clipbrd.TClipboard.SetTextBuf 3346
Clipbrd.TClipboard.WndProc 3346
clMax enumeration member 26637
clNone enumeration member 26637
clUseClient enumeration member 1293
clUseServer enumeration member 1293
CmAdmCtl 3355
CmAdmCtl namespace 3355
    Classes 3356
CmAdmCtl.CoCOMAdminCatalog 3356
CmAdmCtl.CoCOMAdminCatalog.Create 3357

CmAdmCtl.CoCOMAdminCatalogCollection 3357
CmAdmCtl.CoCOMAdminCatalogCollection.Create 3358
CmAdmCtl.CoCOMAdminCatalogObject 3358
CmAdmCtl.CoCOMAdminCatalogObject.Create 3359
CmAdmCtl.TCOMAdminCatalog 3359
CmAdmCtl.TCOMAdminCatalog.BackupREGDB 3365
CmAdmCtl.TCOMAdminCatalog.Connect 3365
CmAdmCtl.TCOMAdminCatalog.ConnectTo 3366
CmAdmCtl.TCOMAdminCatalog.Create 3363
CmAdmCtl.TCOMAdminCatalog.DefaultInterface 3379
CmAdmCtl.TCOMAdminCatalog.Destroy 3364
CmAdmCtl.TCOMAdminCatalog.Disconnect 3366
CmAdmCtl.TCOMAdminCatalog.ExportApplication 3367
CmAdmCtl.TCOMAdminCatalog.Get_MajorVersion 3367
CmAdmCtl.TCOMAdminCatalog.Get_MinorVersion 3368
CmAdmCtl.TCOMAdminCatalog.GetCollection 3368
CmAdmCtl.TCOMAdminCatalog.GetCollectionByQuery 3368
CmAdmCtl.TCOMAdminCatalog.GetEventClassesForIID 3369
CmAdmCtl.TCOMAdminCatalog.GetMultipleComponentsInfo 3370
CmAdmCtl.TCOMAdminCatalog.ICOMAdminCatalog_Connect 3371
CmAdmCtl.TCOMAdminCatalog.ImportComponent 3371
CmAdmCtl.TCOMAdminCatalog.InitServerData 3372
CmAdmCtl.TCOMAdminCatalog.InstallApplication 3372
CmAdmCtl.TCOMAdminCatalog.InstallComponent 3373
CmAdmCtl.TCOMAdminCatalog.InstallEventClass 3373
CmAdmCtl.TCOMAdminCatalog.InstallMultipleComponents 3374
CmAdmCtl.TCOMAdminCatalog.InstallMultipleEventClasses 3375
CmAdmCtl.TCOMAdminCatalog.MajorVersion 3379
CmAdmCtl.TCOMAdminCatalog.MinorVersion 3380
CmAdmCtl.TCOMAdminCatalog.QueryApplicationFile 3375
CmAdmCtl.TCOMAdminCatalog.RefreshComponents 3376
CmAdmCtl.TCOMAdminCatalog.RefreshRouter 3376
CmAdmCtl.TCOMAdminCatalog.RestoreREGDB 3376
CmAdmCtl.TCOMAdminCatalog.ServiceCheck 3377
CmAdmCtl.TCOMAdminCatalog.ShutdownApplication 3377
CmAdmCtl.TCOMAdminCatalog.StartApplication 3378
CmAdmCtl.TCOMAdminCatalog.StartRouter 3378
CmAdmCtl.TCOMAdminCatalog.StopRouter 3378
CmAdmCtl.TCOMAdminCatalogCollection 3380
CmAdmCtl.TCOMAdminCatalogCollection.Add 3386
CmAdmCtl.TCOMAdminCatalogCollection.AddEnabled 3393
CmAdmCtl.TCOMAdminCatalogCollection.Connect 3386
CmAdmCtl.TCOMAdminCatalogCollection.ConnectTo 3386
CmAdmCtl.TCOMAdminCatalogCollection.Count 3393
CmAdmCtl.TCOMAdminCatalogCollection.Create 3384
CmAdmCtl.TCOMAdminCatalogCollection.DataStoreMajorVersion 3394
CmAdmCtl.TCOMAdminCatalogCollection.DataStoreMinorVersion 3394
CmAdmCtl.TCOMAdminCatalogCollection.DefaultInterface 3395
CmAdmCtl.TCOMAdminCatalogCollection.Destroy 3385
CmAdmCtl.TCOMAdminCatalogCollection.Disconnect 3387
CmAdmCtl.TCOMAdminCatalogCollection.Get_AddEnabled 3387
CmAdmCtl.TCOMAdminCatalogCollection.Get_Count 3388
CmAdmCtl.TCOMAdminCatalogCollection.Get_DataStoreMajorVersion 3388
CmAdmCtl.TCOMAdminCatalogCollection.Get_DataStoreMinorVersion 3388
CmAdmCtl.TCOMAdminCatalogCollection.Get_Item 3389
CmAdmCtl.TCOMAdminCatalogCollection.Get_Name 3389
CmAdmCtl.TCOMAdminCatalogCollection.Get_RemoveEnabled 3389
CmAdmCtl.TCOMAdminCatalogCollection.GetCollection 3389
CmAdmCtl.TCOMAdminCatalogCollection.GetUtilInterface 3390
CmAdmCtl.TCOMAdminCatalogCollection.InitServerData 3390
CmAdmCtl.TCOMAdminCatalogCollection.Name 3395
CmAdmCtl.TCOMAdminCatalogCollection.Populate 3391
CmAdmCtl.TCOMAdminCatalogCollection.PopulateByKey 3391
CmAdmCtl.TCOMAdminCatalogCollection.PopulateByQuery 3392
CmAdmCtl.TCOMAdminCatalogCollection.Remove 3392
CmAdmCtl.TCOMAdminCatalogCollection.RemoveEnabled 3395
CmAdmCtl.TCOMAdminCatalogCollection.SaveChanges 3392
CmAdmCtl.TCOMAdminCatalogObject 3396

CmAdmCtl.TCOMAdminCatalogObject.Connect 3401
CmAdmCtl.TCOMAdminCatalogObject.ConnectTo 3401
CmAdmCtl.TCOMAdminCatalogObject.Create 3400
CmAdmCtl.TCOMAdminCatalogObject.DefaultInterface 3405
CmAdmCtl.TCOMAdminCatalogObject.Destroy 3400
CmAdmCtl.TCOMAdminCatalogObject.Disconnect 3402
CmAdmCtl.TCOMAdminCatalogObject.Get_Key 3402
CmAdmCtl.TCOMAdminCatalogObject.Get_Name 3402
CmAdmCtl.TCOMAdminCatalogObject.Get_Valid 3403
CmAdmCtl.TCOMAdminCatalogObject.Get_Value 3403
CmAdmCtl.TCOMAdminCatalogObject.InitServerData 3403
CmAdmCtl.TCOMAdminCatalogObject.IsPropertyReadOnly 3404
CmAdmCtl.TCOMAdminCatalogObject.IsPropertyWriteOnly 3404
CmAdmCtl.TCOMAdminCatalogObject.Key 3405
CmAdmCtl.TCOMAdminCatalogObject.Name 3406
CmAdmCtl.TCOMAdminCatalogObject.Set_Value 3404
CmAdmCtl.TCOMAdminCatalogObject.Valid 3406
CmAdmCtl.TCOMAdminCatalogObject.Value 3407
cmaVirtualSize enumeration member 11233
cmdFile enumeration member 1291
cmdStoredProc enumeration member 1291
cmdTable enumeration member 1291
cmdTableDirect enumeration member 1291
cmdText enumeration member 1291
cmdUnknown enumeration member 1291
cmRead enumeration member 1292
cmReadWrite enumeration member 1292
cmShareDenyNone enumeration member 1292
cmShareDenyRead enumeration member 1292
cmShareDenyWrite enumeration member 1292
cmShareExclusive enumeration member 1292
cmUnknown enumeration member 1292
cmWrite enumeration member 1292
cnAdded enumeration member 3270
cnDeleting enumeration member 3270
cnExtracting enumeration member 3270
coAddNew enumeration member 1294
coAll enumeration member 565
coApproxPosition enumeration member 1294
coAsyncConnect enumeration member 1293
coBookmark enumeration member 1294
CoCOMAdminCatalog class 3356
about CoCOMAdminCatalog class 3356
CoCOMAdminCatalog Members 3356
CoCOMAdminCatalog methods 3356
Create 3357
CoCOMAdminCatalogCollection class 3357
about CoCOMAdminCatalogCollection class 3357
CoCOMAdminCatalogCollection Members 3357
CoCOMAdminCatalogCollection methods 3357
Create 3358
CoCOMAdminCatalogObject class 3358
about CoCOMAdminCatalogObject class 3358
CoCOMAdminCatalogObject Members 3358
CoCOMAdminCatalogObject methods 3358
Create 3359
coConnectUnspecified enumeration member 1293
CoCScriptErrorList class 19045
about CoCScriptErrorList class 19045
CoCScriptErrorList Members 19045
CoCScriptErrorList methods 19046
Create 19046
CreateRemote 19046
coDelete enumeration member 1294
coFind enumeration member 1294
coHoldRecords enumeration member 1294
coIndex enumeration member 1294
CoInternetExplorer class 19046
about CoInternetExplorer class 19046
CoInternetExplorer Members 19047
CoInternetExplorer methods 19047
Create 19047
CreateRemote 19047
coLeftToRight enumeration member 14632
coLocalPaths enumeration member 24799
coltDefault enumeration member 25316
coltImage enumeration member 25316

coltList enumeration member 25316
coltText enumeration member 25316
ColumnConstraintsColumns class 2133
about ColumnConstraintsColumns class 2133
CatalogName 2134
ColumnConstraintsColumns fields 2133
ColumnConstraintsColumns Members 2133
ColumnName 2134
ConstraintName 2134
SchemaName 2134
TableName 2134
ColumnsColumns class 2135
about ColumnsColumns class 2135
CatalogName 2137
ColumnName 2137
ColumnsColumns fields 2137
ColumnsColumns Members 2136
DbxDataType 2137
DefaultValue 2138
IsAutoIncrement 2138
IsFixedLength 2138
IsLong 2138
IsNullable 2138
IsUnicode 2139
IsUnsigned 2139
MaxInline 2139
Ordinal 2139
Precision 2139
Scale 2140
SchemaName 2140
TableName 2140
TypeName 2140
ComApp 3409
ComApp namespace 3409
Classes 3410
ComApp.TWebAppAutoObjectFactory 3410
ComApp.TWebAppAutoObjectFactory.Create 3410
ComCtrls 3413
ComCtrls namespace 3413
Classes 3419
Constants 5521
Functions 5435
Structs, Records, Enums 5438
Types 5477
ComCtrls.CheckCommonControl 5435
ComCtrls.CheckCommonControl function 5435
ComCtrls.CheckToolMenuDropdown 5435
ComCtrls.CheckToolMenuDropdown function 5435
ComCtrls.CN_BANDCHANGE 5521
ComCtrls.CN_BANDCHANGE constant 5521
ComCtrls.CN_DROPDOWNCLOSED 5521
ComCtrls.CN_DROPDOWNCLOSED constant 5521
ComCtrls.ColumnHeaderWidth 5522
ComCtrls.ColumnHeaderWidth constant 5522
ComCtrls.ColumnTextWidth 5522
ComCtrls.ColumnTextWidth constant 5522
ComCtrls.ComCtlVersionIE3 5522
ComCtrls.ComCtlVersionIE3 constant 5522
ComCtrls.ComCtlVersionIE4 5523
ComCtrls.ComCtlVersionIE4 constant 5523
ComCtrls.ComCtlVersionIE401 5523
ComCtrls.ComCtlVersionIE401 constant 5523
ComCtrls.ComCtlVersionIE5 5523
ComCtrls.ComCtlVersionIE5 constant 5523
ComCtrls.ComCtlVersionIE501 5524
ComCtrls.ComCtlVersionIE501 constant 5524
ComCtrls.ComCtlVersionIE6 5524
ComCtrls.ComCtlVersionIE6 constant 5524
ComCtrls.ECommonCalendarError 3420
ComCtrls.EDateTimeError 3421
ComCtrls.EMonthCalError 3421
ComCtrls.ETreeViewError 3422
ComCtrls.GetComCtlVersion 5436
ComCtrls.GetComCtlVersion function 5436
ComCtrls.InitCommonControl 5437
ComCtrls.InitCommonControl function 5437
ComCtrls.PConversionFormat 5479
ComCtrls.PConversionFormat type 5479

ComCtrls.PNodeCache 5479
ComCtrls.PNodeCache type 5479
ComCtrls.PNodeInfo 5480
ComCtrls.PNodeInfo type 5480
ComCtrls.PRichEditStreamInfo 5480
ComCtrls.PRichEditStreamInfo type 5480
ComCtrls.TAddMode 5439
ComCtrls.TAddMode enumeration 5439
ComCtrls.TAnimate 3422
ComCtrls.TAnimate.Active 3441
ComCtrls.TAnimate.Align 3442
ComCtrls.TAnimate.Anchors 3446
ComCtrls.TAnimate.AutoSize 3446
ComCtrls.TAnimate.BorderWidth 3447
ComCtrls.TAnimate.CanAutoSize 3435
ComCtrls.TAnimate.Center 3447
ComCtrls.TAnimate.Color 3447
ComCtrls.TAnimate.CommonAVI 3448
ComCtrls.TAnimate.Constraints 3450
ComCtrls.TAnimate.Create 3433
ComCtrls.TAnimate.CreateParams 3436
ComCtrls.TAnimate.CreateWnd 3436
ComCtrls.TAnimate.DestroyWnd 3437
ComCtrls.TAnimate.DoClose 3437
ComCtrls.TAnimate.DoOpen 3437
ComCtrls.TAnimate.DoStart 3437
ComCtrls.TAnimate.DoStop 3438
ComCtrls.TAnimate.FileName 3450
ComCtrls.TAnimate.FrameCount 3451
ComCtrls.TAnimate.FrameHeight 3451
ComCtrls.TAnimate.FrameWidth 3451
ComCtrls.TAnimate.Loaded 3438
ComCtrls.TAnimate.OnClose 3461
ComCtrls.TAnimate.OnOpen 3461
ComCtrls.TAnimate.OnStart 3462
ComCtrls.TAnimate.OnStop 3462
ComCtrls.TAnimate.Open 3452
ComCtrls.TAnimate.ParentColor 3453
ComCtrls.TAnimate.ParentShowHint 3453
ComCtrls.TAnimate.Play 3438
ComCtrls.TAnimate.Repetitions 3454
ComCtrls.TAnimate.Reset 3439
ComCtrls.TAnimate.ResHandle 3454
ComCtrls.TAnimate.ResId 3455
ComCtrls.TAnimate.ResName 3455
ComCtrls.TAnimate.Seek 3439
ComCtrls.TAnimate.ShowHint 3456
ComCtrls.TAnimate.StartFrame 3457
ComCtrls.TAnimate.Stop 3440
ComCtrls.TAnimate.StopFrame 3458
ComCtrls.TAnimate.Timers 3458
ComCtrls.TAnimate.Transparent 3459
ComCtrls.TAnimate.Visible 3459
ComCtrls.TAttributeType 5440
ComCtrls.TAttributeType enumeration 5440
ComCtrls.TAutoCompleteOption 5440
ComCtrls.TAutoCompleteOption enumeration 5440
ComCtrls.TAutoCompleteOptions 5480
ComCtrls.TAutoCompleteOptions type 5480
ComCtrls.TCalDayOfWeek 5441
ComCtrls.TCalDayOfWeek enumeration 5441
ComCtrls.TComboBoxEx 3463
ComCtrls.TComboBoxEx.Action 3476
ComCtrls.TComboBoxEx.Align 3477
ComCtrls.TComboBoxEx.Anchors 3480
ComCtrls.TComboBoxEx.AutoCompleteOptions 3481
ComCtrls.TComboBoxEx.BiDiMode 3481
ComCtrls.TComboBoxEx.Color 3482
ComCtrls.TComboBoxEx.Constraints 3482
ComCtrls.TComboBoxEx.Ctl3D 3483
ComCtrls.TComboBoxEx.DragCursor 3483
ComCtrls.TComboBoxEx.DragKind 3484
ComCtrls.TComboBoxEx.DragMode 3484
ComCtrls.TComboBoxEx.DropDownCount 3486
ComCtrls.TComboBoxEx.Enabled 3486
ComCtrls.TComboBoxEx.Font 3487
ComCtrls.TComboBoxEx.Images 3488
ComCtrls.TComboBoxEx.ImeMode 3488

ComCtrls.TComboBoxEx.ImeName 3489
ComCtrls.TComboBoxEx.ItemHeight 3489
ComCtrls.TComboBoxEx.ItemsEx 3490
ComCtrls.TComboBoxEx.MaxLength 3490
ComCtrls.TComboBoxEx.OnBeginEdit 3504
ComCtrls.TComboBoxEx.OnChange 3504
ComCtrls.TComboBoxEx.OnClick 3507
ComCtrls.TComboBoxEx.OnContextPopup 3509
ComCtrls.TComboBoxEx.OnDblClick 3509
ComCtrls.TComboBoxEx.OnDragDrop 3510
ComCtrls.TComboBoxEx.OnDragOver 3512
ComCtrls.TComboBoxEx.OnDropDown 3514
ComCtrls.TComboBoxEx.OnEndDock 3516
ComCtrls.TComboBoxEx.OnEndDrag 3516
ComCtrls.TComboBoxEx.OnEndEdit 3517
ComCtrls.TComboBoxEx.OnEnter 3517
ComCtrls.TComboBoxEx.OnExit 3520
ComCtrls.TComboBoxEx.OnKeyDown 3521
ComCtrls.TComboBoxEx.OnKeyPress 3524
ComCtrls.TComboBoxEx.OnKeyUp 3525
ComCtrls.TComboBoxEx.OnMouseMove 3527
ComCtrls.TComboBoxEx.OnSelect 3529
ComCtrls.TComboBoxEx.OnStartDock 3530
ComCtrls.TComboBoxEx.OnStartDrag 3530
ComCtrls.TComboBoxEx.ParentBiDiMode 3491
ComCtrls.TComboBoxEx.ParentColor 3491
ComCtrls.TComboBoxEx.ParentCtl3D 3492
ComCtrls.TComboBoxEx.ParentFont 3492
ComCtrls.TComboBoxEx.ParentShowHint 3492
ComCtrls.TComboBoxEx.PopupMenu 3493
ComCtrls.TComboBoxEx.ShowHint 3498
ComCtrls.TComboBoxEx.Style 3499
ComCtrls.TComboBoxEx.StyleEx 3499
ComCtrls.TComboBoxEx.TabOrder 3500
ComCtrls.TComboBoxEx.TabStop 3500
ComCtrls.TComboBoxEx.Text 3501
ComCtrls.TComboBoxEx.Visible 3502
ComCtrls.TComboBoxExActionLink 3531
ComCtrls.TComboBoxExActionLink.AddItem 3534
ComCtrls.TComboBoxExStrings 3535
ComCtrls.TComboBoxExStrings.Add 3538
ComCtrls.TComboBoxExStrings.AddItem 3539
ComCtrls.TComboBoxExStrings.AddObject 3539
ComCtrls.TComboBoxExStrings.Clear 3539
ComCtrls.TComboBoxExStrings.Create 3537
ComCtrls.TComboBoxExStrings.Delete 3539
ComCtrls.TComboBoxExStrings.Destroy 3537
ComCtrls.TComboBoxExStrings.Exchange 3540
ComCtrls.TComboBoxExStrings.Get 3540
ComCtrls.TComboBoxExStrings.GetCapacity 3541
ComCtrls.TComboBoxExStrings.GetCount 3541
ComCtrls.TComboBoxExStrings.GetItemClass 3542
ComCtrls.TComboBoxExStrings.GetItemsClass 3542
ComCtrls.TComboBoxExStrings.GetObject 3542
ComCtrls.TComboBoxExStrings.IndexOf 3542
ComCtrls.TComboBoxExStrings.IndexOfName 3543
ComCtrls.TComboBoxExStrings.Insert 3543
ComCtrls.TComboBoxExStrings.ItemsEx 3545
ComCtrls.TComboBoxExStrings.Move 3543
ComCtrls.TComboBoxExStrings.PutObject 3544
ComCtrls.TComboBoxExStrings.SetUpdateState 3545
ComCtrls.TComboBoxExStrings.SortType 3545
ComCtrls.TComboBoxExStyle 5442
ComCtrls.TComboBoxExStyle enumeration 5442
ComCtrls.TComboBoxExStyleEx 5442
ComCtrls.TComboBoxExStyleEx enumeration 5442
ComCtrls.TComboBoxExStyles 5481
ComCtrls.TComboBoxExStyles type 5481
ComCtrls.TComboExItem 3546
ComCtrls.TComboExItem.Assign 3548
ComCtrls.TComboExItem.Indent 3551
ComCtrls.TComboExItem.OverlayImageIndex 3551
ComCtrls.TComboExItem.SelectedImageIndex 3552
ComCtrls.TComboExItem.SetCaption 3549
ComCtrls.TComboExItem.SetData 3549
ComCtrls.TComboExItem.SetDisplayName 3549
ComCtrls.TComboExItem.SetImageIndex 3550
ComCtrls.TComboExItem.SetIndex 3550

ComCtrls.TComboExItem.SetOverlayImageIndex 3550
ComCtrls.TComboExItem.SetSelectedImageIndex 3551
ComCtrls.TComboExItemClass 5481
ComCtrls.TComboExItemClass type 5481
ComCtrls.TComboExItems 3552
ComCtrls.TComboExItems.Add 3555
ComCtrls.TComboExItems.AddItem 3556
ComCtrls.TComboExItems.ComboItems 3557
ComCtrls.TComboExItems.Insert 3556
ComCtrls.TComboExItems.Notify 3557
ComCtrls.TComboExItems.SetItem 3557
ComCtrls.TComboExItemsClass 5482
ComCtrls.TComboExItemsClass type 5482
ComCtrls.TCommonAVI 5443
ComCtrls.TCommonAVI enumeration 5443
ComCtrls.TCommonCalendar 3558
ComCtrls.TCommonCalendar.BoldDays 3570
ComCtrls.TCommonCalendar.CalColors 3575
ComCtrls.TCommonCalendar.CalendarHandle 3576
ComCtrls.TCommonCalendar.CalExceptionClass 3576
ComCtrls.TCommonCalendar.CheckEmptyDate 3571
ComCtrls.TCommonCalendar.CheckValidDate 3571
ComCtrls.TCommonCalendar.Create 3569
ComCtrls.TCommonCalendar.CreateWnd 3572
ComCtrls.TCommonCalendar.Date 3576
ComCtrls.TCommonCalendar.DateTime 3577
ComCtrls.TCommonCalendar.Destroy 3570
ComCtrls.TCommonCalendar.EndDate 3577
ComCtrls.TCommonCalendar.FirstDayOfWeek 3578
ComCtrls.TCommonCalendar.GetCalendarHandle 3572
ComCtrls.TCommonCalendar.GetCalStyles 3573
ComCtrls.TCommonCalendar.MaxDate 3578
ComCtrls.TCommonCalendar.MaxSelectRange 3579
ComCtrls.TCommonCalendar.MinDate 3579
ComCtrls.TCommonCalendar.MonthDelta 3579
ComCtrls.TCommonCalendar.MsgSetCalColors 3573
ComCtrls.TCommonCalendar.MsgSetDateTime 3574
ComCtrls.TCommonCalendar.MsgSetRange 3574
ComCtrls.TCommonCalendar.MultiSelect 3580
ComCtrls.TCommonCalendar.OnGetMonthInfo 3581
ComCtrls.TCommonCalendar.ShowToday 3580
ComCtrls.TCommonCalendar.ShowTodayCircle 3580
ComCtrls.TCommonCalendar.WeekNumbers 3581
ComCtrls.TConsistentAttribute 5444
ComCtrls.TConsistentAttribute enumeration 5444
ComCtrls.TConsistentAttributes 5482
ComCtrls.TConsistentAttributes type 5482
ComCtrls.TConversion 3582
ComCtrls.TConversion.ConvertReadStream 3583
ComCtrls.TConversion.ConvertWriteStream 3584
ComCtrls.TConversionClass 5482
ComCtrls.TConversionClass type 5482
ComCtrls.TConversionFormat 5444
ComCtrls.TConversionFormat record 5444
ComCtrls.TCoolBand 3584
ComCtrls.TCoolBand.Assign 3587
ComCtrls.TCoolBand.Bitmap 3589
ComCtrls.TCoolBand.BorderStyle 3589
ComCtrls.TCoolBand.Break 3590
ComCtrls.TCoolBand.Color 3590
ComCtrls.TCoolBand.Control 3590
ComCtrls.TCoolBand.Create 3586
ComCtrls.TCoolBand.Destroy 3586
ComCtrls.TCoolBand.FixedBackground 3591
ComCtrls.TCoolBand.FixedSize 3591
ComCtrls.TCoolBand.GetDisplayName 3587
ComCtrls.TCoolBand.Height 3591
ComCtrls.TCoolBand.HorizontalOnly 3592
ComCtrls.TCoolBand.ImageIndex 3592
ComCtrls.TCoolBand.MinHeight 3592
ComCtrls.TCoolBand.MinWidth 3593
ComCtrls.TCoolBand.ParentBitmap 3593
ComCtrls.TCoolBand.ParentBitmapChanged 3588
ComCtrls.TCoolBand.ParentColor 3593
ComCtrls.TCoolBand.ParentColorChanged 3588
ComCtrls.TCoolBand.Text 3594
ComCtrls.TCoolBand.Visible 3595
ComCtrls.TCoolBand.Width 3596

ComCtrls.TCoolBandMaximize 5445
ComCtrls.TCoolBandMaximize enumeration 5445
ComCtrls.TCoolBands 3596
ComCtrls.TCoolBands.Add 3599
ComCtrls.TCoolBands.CoolBar 3601
ComCtrls.TCoolBands.Create 3598
ComCtrls.TCoolBands.FindBand 3599
ComCtrls.TCoolBands.GetOwner 3599
ComCtrls.TCoolBands.HaveGraphic 3600
ComCtrls.TCoolBands.Items 3601
ComCtrls.TCoolBands.Update 3600
ComCtrls.TCoolBar 3603
ComCtrls.TCoolBar.Align 3626
ComCtrls.TCoolBar.AlignControls 3616
ComCtrls.TCoolBar.Anchors 3626
ComCtrls.TCoolBar.AutoSize 3627
ComCtrls.TCoolBar.BandBorderStyle 3627
ComCtrls.TCoolBar.BandMaximize 3628
ComCtrls.TCoolBar.Bands 3628
ComCtrls.TCoolBar.Bitmap 3630
ComCtrls.TCoolBar.BorderWidth 3630
ComCtrls.TCoolBar.CanAutoSize 3617
ComCtrls.TCoolBar.Change 3617
ComCtrls.TCoolBar.Color 3631
ComCtrls.TCoolBar.Constraints 3631
ComCtrls.TCoolBar.Create 3614
ComCtrls.TCoolBar.CreateParams 3617
ComCtrls.TCoolBar.CreateWnd 3618
ComCtrls.TCoolBar.Ctl3D 3632
ComCtrls.TCoolBar.Destroy 3615
ComCtrls.TCoolBar.DockSite 3632
ComCtrls.TCoolBar.DragCursor 3633
ComCtrls.TCoolBar.DragKind 3634
ComCtrls.TCoolBar.DragMode 3634
ComCtrls.TCoolBar.EdgeBorders 3636
ComCtrls.TCoolBar.EdgeInner 3636
ComCtrls.TCoolBar.EdgeOuter 3637
ComCtrls.TCoolBar.Enabled 3637
ComCtrls.TCoolBar.FixedOrder 3637
ComCtrls.TCoolBar.FixedSize 3638
ComCtrls.TCoolBar.FlipChildren 3618
ComCtrls.TCoolBar.Font 3638
ComCtrls.TCoolBar.GetPalette 3619
ComCtrls.TCoolBar.HitTest 3619
ComCtrls.TCoolBar.Images 3639
ComCtrls.TCoolBar.Loaded 3619
ComCtrls.TCoolBar.Notification 3620
ComCtrls.TCoolBar.OnChange 3650
ComCtrls.TCoolBar.OnClick 3650
ComCtrls.TCoolBar.OnContextPopup 3652
ComCtrls.TCoolBar.OnDblClick 3653
ComCtrls.TCoolBar.OnDockDrop 3653
ComCtrls.TCoolBar.OnDockOver 3655
ComCtrls.TCoolBar.OnDragDrop 3657
ComCtrls.TCoolBar.OnDragOver 3660
ComCtrls.TCoolBar.OnEndDock 3662
ComCtrls.TCoolBar.OnEndDrag 3662
ComCtrls.TCoolBar.OnGetSiteInfo 3663
ComCtrls.TCoolBar.OnMouseActivate 3665
ComCtrls.TCoolBar.OnMouseDown 3666
ComCtrls.TCoolBar.OnMouseEnter 3669
ComCtrls.TCoolBar.OnMouseLeave 3669
ComCtrls.TCoolBar.OnMouseMove 3669
ComCtrls.TCoolBar.OnMouseUp 3672
ComCtrls.TCoolBar.OnResize 3674
ComCtrls.TCoolBar.OnStartDock 3674
ComCtrls.TCoolBar.OnStartDrag 3675
ComCtrls.TCoolBar.OnUnDock 3676
ComCtrls.TCoolBar.PaintWindow 3620
ComCtrls.TCoolBar.ParentColor 3640
ComCtrls.TCoolBar.ParentFont 3640
ComCtrls.TCoolBar.ParentShowHint 3641
ComCtrls.TCoolBar.PopupMenu 3641
ComCtrls.TCoolBar.ShowHint 3646
ComCtrls.TCoolBar.ShowText 3647
ComCtrls.TCoolBar.Vertical 3647
ComCtrls.TCoolBar.Visible 3648
ComCtrls.TCoolBar.WndProc 3621

ComCtrls.TCustomComboBoxEx 3678
ComCtrls.TCustomComboBoxEx.ActionChange 3692
ComCtrls.TCustomComboBoxEx.AutoCompleteOptions 3699
ComCtrls.TCustomComboBoxEx.CMColorChanged 3693
ComCtrls.TCustomComboBoxEx.CMParentColorChanged 3693
ComCtrls.TCustomComboBoxEx.CNNotify 3693
ComCtrls.TCustomComboBoxEx.ComboExWndProc 3694
ComCtrls.TCustomComboBoxEx.Create 3691
ComCtrls.TCustomComboBoxEx.CreateParams 3694
ComCtrls.TCustomComboBoxEx.CreateWnd 3694
ComCtrls.TCustomComboBoxEx.Destroy 3691
ComCtrls.TCustomComboBoxEx.DestroyWnd 3695
ComCtrls.TCustomComboBoxEx.DropDownCount 3699
ComCtrls.TCustomComboBoxEx.Focused 3695
ComCtrls.TCustomComboBoxEx.GetActionLinkClass 3695
ComCtrls.TCustomComboBoxEx.GetItemCount 3696
ComCtrls.TCustomComboBoxEx.GetItemHt 3696
ComCtrls.TCustomComboBoxEx.GetItemsClass 3696
ComCtrls.TCustomComboBoxEx.Images 3700
ComCtrls.TCustomComboBoxEx.ItemsEx 3700
ComCtrls.TCustomComboBoxEx.Notification 3697
ComCtrls.TCustomComboBoxEx.OnBeginEdit 3702
ComCtrls.TCustomComboBoxEx.OnEndEdit 3702
ComCtrls.TCustomComboBoxEx.SelText 3701
ComCtrls.TCustomComboBoxEx.SetDropDownCount 3697
ComCtrls.TCustomComboBoxEx.Style 3701
ComCtrls.TCustomComboBoxEx.StyleEx 3701
ComCtrls.TCustomComboBoxEx.WMHelp 3697
ComCtrls.TCustomComboBoxEx.WMLButtonDown 3698
ComCtrls.TCustomComboBoxEx.WndProc 3698
ComCtrls.TCustomDrawPanelEvent 5483
ComCtrls.TCustomDrawPanelEvent type 5483
ComCtrls.TCustomDrawSectionEvent 5483
ComCtrls.TCustomDrawSectionEvent type 5483
ComCtrls.TCustomDrawStage 5445
ComCtrls.TCustomDrawStage enumeration 5445
ComCtrls.TCustomDrawState 5483
ComCtrls.TCustomDrawState type 5483
ComCtrls.TCustomDrawTarget 5446
ComCtrls.TCustomDrawTarget enumeration 5446
ComCtrls.TCustomHCCreateSectionClassEvent 5484
ComCtrls.TCustomHCCreateSectionClassEvent type 5484
ComCtrls.TCustomHeaderControl 3702
ComCtrls.TCustomHeaderControl.Align 3723
ComCtrls.TCustomHeaderControl.Anchors 3727
ComCtrls.TCustomHeaderControl.BiDiMode 3728
ComCtrls.TCustomHeaderControl.BorderWidth 3728
ComCtrls.TCustomHeaderControl.Canvas 3728
ComCtrls.TCustomHeaderControl.Constraints 3730
ComCtrls.TCustomHeaderControl.Create 3714
ComCtrls.TCustomHeaderControl.CreateParams 3715
ComCtrls.TCustomHeaderControl.CreateSection 3715
ComCtrls.TCustomHeaderControl.CreateSections 3715
ComCtrls.TCustomHeaderControl.CreateWnd 3716
ComCtrls.TCustomHeaderControl.Destroy 3714
ComCtrls.TCustomHeaderControl.DestroyWnd 3716
ComCtrls.TCustomHeaderControl.DragCursor 3730
ComCtrls.TCustomHeaderControl.DragKind 3731
ComCtrls.TCustomHeaderControl.DragMode 3731
ComCtrls.TCustomHeaderControl.DragReorder 3733
ComCtrls.TCustomHeaderControl.DrawSection 3716
ComCtrls.TCustomHeaderControl.Enabled 3733
ComCtrls.TCustomHeaderControl.FlipChildren 3717
ComCtrls.TCustomHeaderControl.Font 3733
ComCtrls.TCustomHeaderControl.FullDrag 3735
ComCtrls.TCustomHeaderControl.HotTrack 3735
ComCtrls.TCustomHeaderControl.Images 3735
ComCtrls.TCustomHeaderControl.Notification 3717
ComCtrls.TCustomHeaderControl.OnCreateSectionClass 3736
ComCtrls.TCustomHeaderControl.OnDrawSection 3737
ComCtrls.TCustomHeaderControl.OnSectionClick 3737
ComCtrls.TCustomHeaderControl.OnSectionDrag 3737
ComCtrls.TCustomHeaderControl.OnSectionEndDrag 3738
ComCtrls.TCustomHeaderControl.OnSectionResize 3738
ComCtrls.TCustomHeaderControl.OnSectionTrack 3738
ComCtrls.TCustomHeaderControl.SectionClick 3717
ComCtrls.TCustomHeaderControl.SectionDrag 3717
ComCtrls.TCustomHeaderControl.SectionEndDrag 3718

ComCtrls.TCustomHeaderControl.SectionResize 3718
ComCtrls.TCustomHeaderControl.Sections 3735
ComCtrls.TCustomHeaderControl.SectionTrack 3718
ComCtrls.TCustomHeaderControl.SetImages 3718
ComCtrls.TCustomHeaderControl.Style 3736
ComCtrls.TCustomHeaderControl.WndProc 3719
ComCtrls.TCustomHotKey 3739
ComCtrls.TCustomHotKey.AutoSize 3752
ComCtrls.TCustomHotKey.CNCommand 3750
ComCtrls.TCustomHotKey.Create 3750
ComCtrls.TCustomHotKey.CreateParams 3751
ComCtrls.TCustomHotKey.CreateWnd 3751
ComCtrls.TCustomHotKey.HotKey 3752
ComCtrls.TCustomHotKey.InvalidKeys 3753
ComCtrls.TCustomHotKey.Modifiers 3755
ComCtrls.TCustomHotKey.OnChange 3757
ComCtrls.TCustomHotKey.SetAutoSize 3752
ComCtrls.TCustomHotKey.TabStop 3756
ComCtrls.TCustomListView 3757
ComCtrls.TCustomListView.ActionChange 3773
ComCtrls.TCustomListView.AddItem 3773
ComCtrls.TCustomListView.AllocBy 3803
ComCtrls.TCustomListView.AlphaSort 3774
ComCtrls.TCustomListView.Arrange 3777
ComCtrls.TCustomListView.BorderStyle 3803
ComCtrls.TCustomListView.BoundingRect 3804
ComCtrls.TCustomListView.CanChange 3777
ComCtrls.TCustomListView.CanEdit 3778
ComCtrls.TCustomListView.Canvas 3804
ComCtrls.TCustomListView.Change 3778
ComCtrls.TCustomListView.ChangeScale 3778
ComCtrls.TCustomListView.Checkboxes 3805
ComCtrls.TCustomListView.Clear 3779
ComCtrls.TCustomListView.ClearSelection 3779
ComCtrls.TCustomListView.ColClick 3780
ComCtrls.TCustomListView.ColRightClick 3780
ComCtrls.TCustomListView.Column 3806
ComCtrls.TCustomListView.ColumnClick 3808
ComCtrls.TCustomListView.Columns 3808
ComCtrls.TCustomListView.ColumnsShowing 3780
ComCtrls.TCustomListView.CopySelection 3781
ComCtrls.TCustomListView.Create 3770
ComCtrls.TCustomListView.CreateListItem 3781
ComCtrls.TCustomListView.CreateListItems 3782
ComCtrls.TCustomListView.CreateParams 3782
ComCtrls.TCustomListView.CreateWnd 3782
ComCtrls.TCustomListView.CustomDraw 3783
ComCtrls.TCustomListView.CustomDrawItem 3783
ComCtrls.TCustomListView.CustomDrawSubItem 3783
ComCtrls.TCustomListView.CustomSort 3784
ComCtrls.TCustomListView.Delete 3785
ComCtrls.TCustomListView.DeleteSelected 3786
ComCtrls.TCustomListView.Destroy 3771
ComCtrls.TCustomListView.DestroyWnd 3786
ComCtrls.TCustomListView.DoEndDrag 3787
ComCtrls.TCustomListView.DoInfoTip 3787
ComCtrls.TCustomListView.DoStartDrag 3787
ComCtrls.TCustomListView.DrawItem 3788
ComCtrls.TCustomListView.DropTarget 3810
ComCtrls.TCustomListView.Edit 3788
ComCtrls.TCustomListView.FindCaption 3788
ComCtrls.TCustomListView.FindData 3789
ComCtrls.TCustomListView.FlatScrollBars 3810
ComCtrls.TCustomListView.FullDrag 3811
ComCtrls.TCustomListView.GetActionLinkClass 3789
ComCtrls.TCustomListView.GetCount 3789
ComCtrls.TCustomListView.GetDragImages 3790
ComCtrls.TCustomListView.GetHitTestInfoAt 3790
ComCtrls.TCustomListView.GetItemAt 3790
ComCtrls.TCustomListView.GetItemIndex 3791
ComCtrls.TCustomListView.GetNearestItem 3791
ComCtrls.TCustomListView.GetNextItem 3792
ComCtrls.TCustomListView.GetSearchString 3793
ComCtrls.TCustomListView.GetSelCount 3794
ComCtrls.TCustomListView.GridLines 3811
ComCtrls.TCustomListView.HideSelection 3812
ComCtrls.TCustomListView.HotTrack 3812
ComCtrls.TCustomListView.HotTrackStyles 3812

ComCtrls.TCustomListView.HoverTime 3813
ComCtrls.TCustomListView.IconOptions 3813
ComCtrls.TCustomListView.InsertItem 3794
ComCtrls.TCustomListView.IsCustomDrawn 3794
ComCtrls.TCustomListView.IsEditing 3795
ComCtrls.TCustomListView.ItemFocused 3813
ComCtrls.TCustomListView.Items 3814
ComCtrls.TCustomListView.LargeImages 3817
ComCtrls.TCustomListView.MouseUp 3795
ComCtrls.TCustomListView.MultiSelect 3820
ComCtrls.TCustomListView.Notification 3796
ComCtrls.TCustomListView.OnAdvancedCustomDraw 3834
ComCtrls.TCustomListView.OnAdvancedCustomDrawItem 3835
ComCtrls.TCustomListView.OnAdvancedCustomDrawSubItem 3835
ComCtrls.TCustomListView.OnChange 3836
ComCtrls.TCustomListView.OnChanging 3837
ComCtrls.TCustomListView.OnColumnClick 3837
ComCtrls.TCustomListView.OnColumnDragged 3840
ComCtrls.TCustomListView.OnColumnRightClick 3841
ComCtrls.TCustomListView.OnCompare 3841
ComCtrls.TCustomListView.OnCreateItemClass 3844
ComCtrls.TCustomListView.OnCustomDraw 3844
ComCtrls.TCustomListView.OnCustomDrawItem 3845
ComCtrls.TCustomListView.OnCustomDrawSubItem 3846
ComCtrls.TCustomListView.OnData 3846
ComCtrls.TCustomListView.OnDataFind 3847
ComCtrls.TCustomListView.OnDataHint 3847
ComCtrls.TCustomListView.OnDataStateChange 3848
ComCtrls.TCustomListView.OnDeletion 3848
ComCtrls.TCustomListView.OnDrawItem 3848
ComCtrls.TCustomListView.OnEdited 3849
ComCtrls.TCustomListView.OnEditing 3850
ComCtrls.TCustomListView.OnGetImageIndex 3850
ComCtrls.TCustomListView.OnGetSubItemImage 3850
ComCtrls.TCustomListView.OnInfoTip 3851
ComCtrls.TCustomListView.OnInsert 3851
ComCtrls.TCustomListView.OnSelectItem 3852
ComCtrls.TCustomListView.OwnerData 3820
ComCtrls.TCustomListView.OwnerDataFetch 3796
ComCtrls.TCustomListView.OwnerDataFind 3797
ComCtrls.TCustomListView.OwnerDataHint 3797
ComCtrls.TCustomListView.OwnerDataStateChange 3798
ComCtrls.TCustomListView.OwnerDraw 3821
ComCtrls.TCustomListView.ReadOnly 3821
ComCtrls.TCustomListView.RowSelect 3821
ComCtrls.TCustomListView.Scroll 3798
ComCtrls.TCustomListView.SelCount 3822
ComCtrls.TCustomListView.SelectAll 3799
ComCtrls.TCustomListView.Selected 3822
ComCtrls.TCustomListView.SetItemIndex 3799
ComCtrls.TCustomListView.SetMultiSelect 3800
ComCtrls.TCustomListView.SetViewStyle 3800
ComCtrls.TCustomListView.ShowColumnHeaders 3824
ComCtrls.TCustomListView.ShowWorkAreas 3824
ComCtrls.TCustomListView.SmallImages 3825
ComCtrls.TCustomListView.SortType 3827
ComCtrls.TCustomListView.StateImages 3828
ComCtrls.TCustomListView.StringWidth 3800
ComCtrls.TCustomListView.TopItem 3828
ComCtrls.TCustomListView.UpdateColumn 3801
ComCtrls.TCustomListView.UpdateColumns 3801
ComCtrls.TCustomListView.UpdateItems 3801
ComCtrls.TCustomListView.ViewOrigin 3829
ComCtrls.TCustomListView.ViewStyle 3829
ComCtrls.TCustomListView.VisibleRowCount 3832
ComCtrls.TCustomListView.WndProc 3802
ComCtrls.TCustomListView.WorkAreas 3833
ComCtrls.TCustomRichEdit 3854
ComCtrls.TCustomRichEdit.Clear 3867
ComCtrls.TCustomRichEdit.Create 3866
ComCtrls.TCustomRichEdit.CreateParams 3868
ComCtrls.TCustomRichEdit.CreateWnd 3868
ComCtrls.TCustomRichEdit.DefAttributes 3877
ComCtrls.TCustomRichEdit.DefaultConverter 3878
ComCtrls.TCustomRichEdit.Destroy 3866
ComCtrls.TCustomRichEdit.DestroyWnd 3868
ComCtrls.TCustomRichEdit.DoSetMaxLength 3869
ComCtrls.TCustomRichEdit.FindText 3869

ComCtrls.TCustomRichEdit.GetCaretPos 3871
ComCtrls.TCustomRichEdit.GetSelLength 3871
ComCtrls.TCustomRichEdit.GetSelStart 3872
ComCtrls.TCustomRichEdit.GetSelText 3872
ComCtrls.TCustomRichEdit.GetSelTextBuf 3872
ComCtrls.TCustomRichEdit.HideScrollBars 3879
ComCtrls.TCustomRichEdit.HideSelection 3881
ComCtrls.TCustomRichEdit.Lines 3882
ComCtrls.TCustomRichEdit.OnProtectChange 3888
ComCtrls.TCustomRichEdit.OnResizeRequest 3889
ComCtrls.TCustomRichEdit.OnSaveClipboard 3890
ComCtrls.TCustomRichEdit.OnSelectionChange 3890
ComCtrls.TCustomRichEdit.PageRect 3884
ComCtrls.TCustomRichEdit.Paragraph 3884
ComCtrls.TCustomRichEdit.PlainText 3886
ComCtrls.TCustomRichEdit.Print 3873
ComCtrls.TCustomRichEdit.RegisterConversionFormat 3874
ComCtrls.TCustomRichEdit.RequestSize 3874
ComCtrls.TCustomRichEdit.SelAttributes 3887
ComCtrls.TCustomRichEdit.SelectionChange 3875
ComCtrls.TCustomRichEdit.SetCaretPos 3875
ComCtrls.TCustomRichEdit.SetSelLength 3876
ComCtrls.TCustomRichEdit.SetSelStart 3876
ComCtrls.TCustomSectionNotifyEvent 5484
ComCtrls.TCustomSectionNotifyEvent type 5484
ComCtrls.TCustomSectionTrackEvent 5485
ComCtrls.TCustomSectionTrackEvent type 5485
ComCtrls.TCustomStatusBar 3892
ComCtrls.TCustomStatusBar.AutoHint 3910
ComCtrls.TCustomStatusBar.Canvas 3910
ComCtrls.TCustomStatusBar.ChangeScale 3904
ComCtrls.TCustomStatusBar.Create 3903
ComCtrls.TCustomStatusBar.CreatePanel 3905
ComCtrls.TCustomStatusBar.CreatePanels 3905
ComCtrls.TCustomStatusBar.CreateParams 3905
ComCtrls.TCustomStatusBar.CreateWnd 3906
ComCtrls.TCustomStatusBar.Destroy 3903
ComCtrls.TCustomStatusBar.DoHint 3906
ComCtrls.TCustomStatusBar.DrawPanel 3907
ComCtrls.TCustomStatusBar.ExecuteAction 3907
ComCtrls.TCustomStatusBar.FlipChildren 3908
ComCtrls.TCustomStatusBar.GetPanelClass 3908
ComCtrls.TCustomStatusBar.IsFontStored 3908
ComCtrls.TCustomStatusBar.OnCreatePanelClass 3919
ComCtrls.TCustomStatusBar.OnDrawPanel 3919
ComCtrls.TCustomStatusBar.OnHint 3920
ComCtrls.TCustomStatusBar.Panels 3912
ComCtrls.TCustomStatusBar.SetBounds 3909
ComCtrls.TCustomStatusBar.SetParent 3909
ComCtrls.TCustomStatusBar.SimplePanel 3915
ComCtrls.TCustomStatusBar.SimpleText 3916
ComCtrls.TCustomStatusBar.SizeGrip 3918
ComCtrls.TCustomStatusBar.UseSystemFont 3918
ComCtrls.TCustomTabControl 3921
ComCtrls.TCustomTabControl.AdjustClientRect 3933
ComCtrls.TCustomTabControl.CanChange 3933
ComCtrls.TCustomTabControl.CanShowTab 3934
ComCtrls.TCustomTabControl.Canvas 3940
ComCtrls.TCustomTabControl.Change 3934
ComCtrls.TCustomTabControl.Create 3932
ComCtrls.TCustomTabControl.CreateParams 3935
ComCtrls.TCustomTabControl.CreateWnd 3935
ComCtrls.TCustomTabControl.Destroy 3932
ComCtrls.TCustomTabControl.DisplayRect 3941
ComCtrls.TCustomTabControl.DrawTab 3935
ComCtrls.TCustomTabControl.GetHitTestInfoAt 3936
ComCtrls.TCustomTabControl.GetImageIndex 3936
ComCtrls.TCustomTabControl.HotTrack 3941
ComCtrls.TCustomTabControl.Images 3941
ComCtrls.TCustomTabControl.IndexOfTabAt 3936
ComCtrls.TCustomTabControl.Loaded 3937
ComCtrls.TCustomTabControl.MultiLine 3942
ComCtrls.TCustomTabControl.MultiSelect 3943
ComCtrls.TCustomTabControl.Notification 3937
ComCtrls.TCustomTabControl.OnChange 3952
ComCtrls.TCustomTabControl.OnChanging 3955
ComCtrls.TCustomTabControl.OnDrawTab 3956
ComCtrls.TCustomTabControl.OnGetImageIndex 3956

ComCtrls.TCustomTabControl.OwnerDraw 3944
ComCtrls.TCustomTabControl.RaggedRight 3944
ComCtrls.TCustomTabControl.RowCount 3938
ComCtrls.TCustomTabControl.ScrollOpposite 3944
ComCtrls.TCustomTabControl.ScrollTabs 3938
ComCtrls.TCustomTabControl.SetTabIndex 3938
ComCtrls.TCustomTabControl.Style 3945
ComCtrls.TCustomTabControl.TabHeight 3945
ComCtrls.TCustomTabControl.TabIndex 3946
ComCtrls.TCustomTabControl.TabPosition 3948
ComCtrls.TCustomTabControl.TabRect 3939
ComCtrls.TCustomTabControl.Tabs 3948
ComCtrls.TCustomTabControl.TabStop 3950
ComCtrls.TCustomTabControl.TabWidth 3950
ComCtrls.TCustomTabControl.UpdateTabImages 3939
ComCtrls.TCustomTreeView 3957
ComCtrls.TCustomTreeView.Added 3972
ComCtrls.TCustomTreeView.AlphaSort 3972
ComCtrls.TCustomTreeView.AutoExpand 3996
ComCtrls.TCustomTreeView.BorderStyle 3997
ComCtrls.TCustomTreeView.CanChange 3972
ComCtrls.TCustomTreeView.CanCollapse 3973
ComCtrls.TCustomTreeView.CanEdit 3973
ComCtrls.TCustomTreeView.CanExpand 3974
ComCtrls.TCustomTreeView.Canvas 3997
ComCtrls.TCustomTreeView.Change 3974
ComCtrls.TCustomTreeView.ChangeDelay 3999
ComCtrls.TCustomTreeView.ClearSelection 3974
ComCtrls.TCustomTreeView.Collapse 3975
ComCtrls.TCustomTreeView.Create 3969
ComCtrls.TCustomTreeView.CreateNode 3975
ComCtrls.TCustomTreeView.CreateNodes 3976
ComCtrls.TCustomTreeView.CreateParams 3976
ComCtrls.TCustomTreeView.CreateWnd 3976
ComCtrls.TCustomTreeView.CreateWndRestores 4000
ComCtrls.TCustomTreeView.CustomDraw 3977
ComCtrls.TCustomTreeView.CustomDrawItem 3977
ComCtrls.TCustomTreeView.CustomSort 3978
ComCtrls.TCustomTreeView.Delete 3979
ComCtrls.TCustomTreeView.Deselect 3980
ComCtrls.TCustomTreeView.Destroy 3970
ComCtrls.TCustomTreeView.DestroyWnd 3980
ComCtrls.TCustomTreeView.DoEndDrag 3981
ComCtrls.TCustomTreeView.DoEnter 3981
ComCtrls.TCustomTreeView.DoExit 3981
ComCtrls.TCustomTreeView.DoStartDrag 3982
ComCtrls.TCustomTreeView.DropTarget 4000
ComCtrls.TCustomTreeView.Edit 3982
ComCtrls.TCustomTreeView.Expand 3983
ComCtrls.TCustomTreeView.FChangeTimer 3970
ComCtrls.TCustomTreeView.FindNextToSelect 3983
ComCtrls.TCustomTreeView.FullCollapse 3983
ComCtrls.TCustomTreeView.FullExpand 3984
ComCtrls.TCustomTreeView.GetDragImages 3984
ComCtrls.TCustomTreeView.GetHitTestInfoAt 3985
ComCtrls.TCustomTreeView.GetImageIndex 3986
ComCtrls.TCustomTreeView.GetNodeAt 3987
ComCtrls.TCustomTreeView.GetSelectedIndex 3988
ComCtrls.TCustomTreeView.GetSelections 3989
ComCtrls.TCustomTreeView.HideSelection 4000
ComCtrls.TCustomTreeView.HotTrack 4001
ComCtrls.TCustomTreeView.Images 4001
ComCtrls.TCustomTreeView.Indent 4001
ComCtrls.TCustomTreeView.InvalidateSelectionsRects 3989
ComCtrls.TCustomTreeView.IsCustomDrawn 3989
ComCtrls.TCustomTreeView.IsEditing 3990
ComCtrls.TCustomTreeView.Items 4004
ComCtrls.TCustomTreeView.Loaded 3990
ComCtrls.TCustomTreeView.LoadFromFile 3990
ComCtrls.TCustomTreeView.LoadFromStream 3991
ComCtrls.TCustomTreeView.MouseDown 3991
ComCtrls.TCustomTreeView.MultiSelect 4005
ComCtrls.TCustomTreeView.MultiSelectStyle 4005
ComCtrls.TCustomTreeView.Notification 3992
ComCtrls.TCustomTreeView.OnAddition 4016
ComCtrls.TCustomTreeView.OnAdvancedCustomDraw 4016
ComCtrls.TCustomTreeView.OnAdvancedCustomDrawItem 4016
ComCtrls.TCustomTreeView.OnCancelEdit 4017

ComCtrls.TCustomTreeView.OnChange 4017
ComCtrls.TCustomTreeView.OnChanging 4018
ComCtrls.TCustomTreeView.OnCollapsed 4018
ComCtrls.TCustomTreeView.OnCollapsing 4018
ComCtrls.TCustomTreeView.OnCompare 4019
ComCtrls.TCustomTreeView.OnCreateNodeClass 4019
ComCtrls.TCustomTreeView.OnCustomDraw 4019
ComCtrls.TCustomTreeView.OnCustomDrawItem 4021
ComCtrls.TCustomTreeView.OnDeletion 4024
ComCtrls.TCustomTreeView.OnEdited 4024
ComCtrls.TCustomTreeView.OnEditing 4024
ComCtrls.TCustomTreeView.OnExpanded 4025
ComCtrls.TCustomTreeView.OnExpanding 4025
ComCtrls.TCustomTreeView.OnGetImageIndex 4026
ComCtrls.TCustomTreeView.OnGetSelectedIndex 4026
ComCtrls.TCustomTreeView.ReadOnly 4006
ComCtrls.TCustomTreeView.RightClickSelect 4006
ComCtrls.TCustomTreeView.RowSelect 4007
ComCtrls.TCustomTreeView.SaveToFile 3992
ComCtrls.TCustomTreeView.SaveToStream 3993
ComCtrls.TCustomTreeView.Select 3994
ComCtrls.TCustomTreeView.Selected 4007
ComCtrls.TCustomTreeView.SelectionCount 4012
ComCtrls.TCustomTreeView.Selections 4012
ComCtrls.TCustomTreeView.SetDragMode 3994
ComCtrls.TCustomTreeView.ShowButtons 4013
ComCtrls.TCustomTreeView.ShowLines 4013
ComCtrls.TCustomTreeView.ShowRoot 4013
ComCtrls.TCustomTreeView.SortType 4014
ComCtrls.TCustomTreeView.StateImages 4014
ComCtrls.TCustomTreeView.Subselect 3994
ComCtrls.TCustomTreeView.ToolTips 4014
ComCtrls.TCustomTreeView.TopItem 4015
ComCtrls.TCustomTreeView.ValidateSelection 3995
ComCtrls.TCustomTreeView.WndProc 3995
ComCtrls.TCustomUpDown 4027
ComCtrls.TCustomUpDown.AlignButton 4042
ComCtrls.TCustomUpDown.ArrowKeys 4042
ComCtrls.TCustomUpDown.Associate 4043
ComCtrls.TCustomUpDown.CanChange 4039
ComCtrls.TCustomUpDown.Click 4039
ComCtrls.TCustomUpDown.Create 4038
ComCtrls.TCustomUpDown.CreateParams 4039
ComCtrls.TCustomUpDown.CreateWnd 4040
ComCtrls.TCustomUpDown.DoCanChange 4040
ComCtrls.TCustomUpDown.Increment 4043
ComCtrls.TCustomUpDown.Max 4043
ComCtrls.TCustomUpDown.Min 4044
ComCtrls.TCustomUpDown.Notification 4041
ComCtrls.TCustomUpDown.OnChanging 4046
ComCtrls.TCustomUpDown.OnChangingEx 4047
ComCtrls.TCustomUpDown.OnClick 4047
ComCtrls.TCustomUpDown.Orientation 4044
ComCtrls.TCustomUpDown.Position 4045
ComCtrls.TCustomUpDown.Thousands 4045
ComCtrls.TCustomUpDown.Wrap 4045
ComCtrls.TDateTimeColors 5485
ComCtrls.TDateTimeColors type 5485
ComCtrls.TDateTimeKind 5447
ComCtrls.TDateTimeKind enumeration 5447
ComCtrls.TDateTimePicker 4049
ComCtrls.TDateTimePicker.Align 4066
ComCtrls.TDateTimePicker.Anchors 4069
ComCtrls.TDateTimePicker.BevelEdges 4070
ComCtrls.TDateTimePicker.BevelInner 4070
ComCtrls.TDateTimePicker.BevelKind 4071
ComCtrls.TDateTimePicker.BevelOuter 4071
ComCtrls.TDateTimePicker.BevelWidth 4072
ComCtrls.TDateTimePicker.BiDiMode 4072
ComCtrls.TDateTimePicker.CalAlignment 4073
ComCtrls.TDateTimePicker.CalColors 4073
ComCtrls.TDateTimePicker.Change 4061
ComCtrls.TDateTimePicker.Checked 4073
ComCtrls.TDateTimePicker.CheckEmptyDate 4061
ComCtrls.TDateTimePicker.Color 4074
ComCtrls.TDateTimePicker.Constraints 4074
ComCtrls.TDateTimePicker.Create 4060
ComCtrls.TDateTimePicker.CreateParams 4062

ComCtrls.TDateTimePicker.CreateWnd 4062
ComCtrls.TDateTimePicker.Date 4075
ComCtrls.TDateTimePicker.DateFormat 4076
ComCtrls.TDateTimePicker.DateMode 4076
ComCtrls.TDateTimePicker.DateTime 4076
ComCtrls.TDateTimePicker.DragCursor 4077
ComCtrls.TDateTimePicker.DragKind 4077
ComCtrls.TDateTimePicker.DragMode 4077
ComCtrls.TDateTimePicker.DroppedDown 4079
ComCtrls.TDateTimePicker.Enabled 4079
ComCtrls.TDateTimePicker.Font 4080
ComCtrls.TDateTimePicker.Format 4081
ComCtrls.TDateTimePicker.GetCalendarHandle 4063
ComCtrls.TDateTimePicker.ImeMode 4082
ComCtrls.TDateTimePicker.ImeName 4082
ComCtrls.TDateTimePicker.Kind 4083
ComCtrls.TDateTimePicker.MaxDate 4084
ComCtrls.TDateTimePicker.MinDate 4084
ComCtrls.TDateTimePicker.MsgSetCalColors 4063
ComCtrls.TDateTimePicker.MsgSetDateTime 4064
ComCtrls.TDateTimePicker.MsgSetRange 4064
ComCtrls.TDateTimePicker.OnChange 4096
ComCtrls.TDateTimePicker.OnClick 4097
ComCtrls.TDateTimePicker.OnCloseUp 4099
ComCtrls.TDateTimePicker.OnContextPopup 4099
ComCtrls.TDateTimePicker.OnDblClick 4100
ComCtrls.TDateTimePicker.OnDragDrop 4100
ComCtrls.TDateTimePicker.OnDragOver 4102
ComCtrls.TDateTimePicker.OnDropDown 4105
ComCtrls.TDateTimePicker.OnEndDock 4105
ComCtrls.TDateTimePicker.OnEndDrag 4105
ComCtrls.TDateTimePicker.OnEnter 4106
ComCtrls.TDateTimePicker.OnExit 4109
ComCtrls.TDateTimePicker.OnKeyDown 4110
ComCtrls.TDateTimePicker.OnKeyPress 4113
ComCtrls.TDateTimePicker.OnKeyUp 4114
ComCtrls.TDateTimePicker.OnMouseEnter 4116
ComCtrls.TDateTimePicker.OnMouseLeave 4116
ComCtrls.TDateTimePicker.OnStartDock 4116
ComCtrls.TDateTimePicker.OnStartDrag 4117
ComCtrls.TDateTimePicker.OnUserInput 4118
ComCtrls.TDateTimePicker.ParentBiDiMode 4084
ComCtrls.TDateTimePicker.ParentColor 4085
ComCtrls.TDateTimePicker.ParentFont 4085
ComCtrls.TDateTimePicker.ParentShowHint 4086
ComCtrls.TDateTimePicker.ParseInput 4086
ComCtrls.TDateTimePicker.PopupMenu 4086
ComCtrls.TDateTimePicker.ShowCheckbox 4091
ComCtrls.TDateTimePicker.ShowHint 4091
ComCtrls.TDateTimePicker.TabOrder 4093
ComCtrls.TDateTimePicker.TabStop 4093
ComCtrls.TDateTimePicker.Time 4094
ComCtrls.TDateTimePicker.Visible 4094
ComCtrls.TDisplayCode 5447
ComCtrls.TDisplayCode enumeration 5447
ComCtrls.TDrawPanelEvent 5485
ComCtrls.TDrawPanelEvent type 5485
ComCtrls.TDrawSectionEvent 5486
ComCtrls.TDrawSectionEvent type 5486
ComCtrls.TDrawTabEvent 5486
ComCtrls.TDrawTabEvent type 5486
ComCtrls.TDTCalAlignment 5447
ComCtrls.TDTCalAlignment enumeration 5447
ComCtrls.TDTDateFormat 5448
ComCtrls.TDTDateFormat enumeration 5448
ComCtrls.TDTDateMode 5448
ComCtrls.TDTDateMode enumeration 5448
ComCtrls.TDTParseInputEvent 5487
ComCtrls.TDTParseInputEvent type 5487
ComCtrls.THCCreateSectionClassEvent 5487
ComCtrls.THCCreateSectionClassEvent type 5487
ComCtrls.THeaderControl 4118
ComCtrls.THeaderControl.Align 4130
ComCtrls.THeaderControl.Anchors 4134
ComCtrls.THeaderControl.BiDiMode 4134
ComCtrls.THeaderControl.BorderWidth 4135
ComCtrls.THeaderControl.Constraints 4135
ComCtrls.THeaderControl.DragCursor 4136

ComCtrls.THeaderControl.DragKind 4136
ComCtrls.THeaderControl.DragMode 4137
ComCtrls.THeaderControl.DragReorder 4138
ComCtrls.THeaderControl.Enabled 4138
ComCtrls.THeaderControl.Font 4139
ComCtrls.THeaderControl.FullDrag 4140
ComCtrls.THeaderControl.HotTrack 4140
ComCtrls.THeaderControl.Images 4141
ComCtrls.THeaderControl.OnContextPopup 4152
ComCtrls.THeaderControl.OnCreateSectionClass 4153
ComCtrls.THeaderControl.OnDragDrop 4153
ComCtrls.THeaderControl.OnDragOver 4155
ComCtrls.THeaderControl.OnDrawSection 4158
ComCtrls.THeaderControl.OnEndDock 4159
ComCtrls.THeaderControl.OnEndDrag 4160
ComCtrls.THeaderControl.OnMouseActivate 4161
ComCtrls.THeaderControl.OnMouseDown 4162
ComCtrls.THeaderControl.OnMouseEnter 4164
ComCtrls.THeaderControl.OnMouseLeave 4164
ComCtrls.THeaderControl.OnMouseMove 4164
ComCtrls.THeaderControl.OnMouseUp 4167
ComCtrls.THeaderControl.OnResize 4169
ComCtrls.THeaderControl.OnSectionClick 4170
ComCtrls.THeaderControl.OnSectionDrag 4170
ComCtrls.THeaderControl.OnSectionEndDrag 4170
ComCtrls.THeaderControl.OnSectionResize 4171
ComCtrls.THeaderControl.OnSectionTrack 4171
ComCtrls.THeaderControl.OnStartDock 4172
ComCtrls.THeaderControl.OnStartDrag 4172
ComCtrls.THeaderControl.ParentBiDiMode 4141
ComCtrls.THeaderControl.ParentFont 4141
ComCtrls.THeaderControl.ParentShowHint 4142
ComCtrls.THeaderControl.PopupMenu 4142
ComCtrls.THeaderControl.Sections 4147
ComCtrls.THeaderControl.ShowHint 4148
ComCtrls.THeaderControl.Style 4150
ComCtrls.THeaderControl.Visible 4150
ComCtrls.THeaderSection 4173
ComCtrls.THeaderSection.Alignment 4178
ComCtrls.THeaderSection.AllowClick 4178
ComCtrls.THeaderSection.Assign 4176
ComCtrls.THeaderSection.AutoSize 4179
ComCtrls.THeaderSection.BiDiMode 4179
ComCtrls.THeaderSection.Create 4175
ComCtrls.THeaderSection.GetDisplayName 4176
ComCtrls.THeaderSection.ImageIndex 4179
ComCtrls.THeaderSection.Left 4180
ComCtrls.THeaderSection.MaxWidth 4180
ComCtrls.THeaderSection.MinWidth 4180
ComCtrls.THeaderSection.ParentBiDiMode 4181
ComCtrls.THeaderSection.ParentBiDiModeChanged 4176
ComCtrls.THeaderSection.Right 4181
ComCtrls.THeaderSection.Style 4181
ComCtrls.THeaderSection.Text 4183
ComCtrls.THeaderSection.UseRightToLeftAlignment 4177
ComCtrls.THeaderSection.UseRightToLeftReading 4177
ComCtrls.THeaderSection.Width 4184
ComCtrls.THeaderSectionClass 5487
ComCtrls.THeaderSectionClass type 5487
ComCtrls.THeaderSections 4185
ComCtrls.THeaderSections.Add 4188
ComCtrls.THeaderSections.AddItem 4188
ComCtrls.THeaderSections.Create 4187
ComCtrls.THeaderSections.GetOwner 4188
ComCtrls.THeaderSections.Insert 4189
ComCtrls.THeaderSections.Items 4190
ComCtrls.THeaderSections.Update 4189
ComCtrls.THeaderSectionStyle 5449
ComCtrls.THeaderSectionStyle enumeration 5449
ComCtrls.THeaderStyle 5449
ComCtrls.THeaderStyle enumeration 5449
ComCtrls.THitTest 5450
ComCtrls.THitTest enumeration 5450
ComCtrls.THitTests 5488
ComCtrls.THitTests type 5488
ComCtrls.THKInvalidKey 5451
ComCtrls.THKInvalidKey enumeration 5451
ComCtrls.THKInvalidKeys 5488

ComCtrls.THKInvalidKeys type 5488
ComCtrls.THKModifier 5451
ComCtrls.THKModifier enumeration 5451
ComCtrls.THKModifiers 5488
ComCtrls.THKModifiers type 5488
ComCtrls.THotKey 4190
ComCtrls.THotKey.Anchors 4202
ComCtrls.THotKey.AutoSize 4202
ComCtrls.THotKey.BiDiMode 4203
ComCtrls.THotKey.Constraints 4203
ComCtrls.THotKey.Enabled 4204
ComCtrls.THotKey.Hint 4204
ComCtrls.THotKey.HotKey 4206
ComCtrls.THotKey.InvalidKeys 4207
ComCtrls.THotKey.Modifiers 4209
ComCtrls.THotKey.OnChange 4220
ComCtrls.THotKey.OnContextPopup 4220
ComCtrls.THotKey.OnEnter 4221
ComCtrls.THotKey.OnExit 4223
ComCtrls.THotKey.OnMouseActivate 4225
ComCtrls.THotKey.OnMouseDown 4226
ComCtrls.THotKey.OnMouseEnter 4228
ComCtrls.THotKey.OnMouseLeave 4228
ComCtrls.THotKey.OnMouseMove 4228
ComCtrls.THotKey.OnMouseUp 4231
ComCtrls.THotKey.ParentBiDiMode 4210
ComCtrls.THotKey.ParentShowHint 4210
ComCtrls.THotKey.PopupMenu 4211
ComCtrls.THotKey.ShowHint 4215
ComCtrls.THotKey.TabOrder 4217
ComCtrls.THotKey.TabStop 4217
ComCtrls.THotKey.Visible 4218
ComCtrls.TIconArrangement 5452
ComCtrls.TIconArrangement enumeration 5452
ComCtrls.TIconOptions 4233
ComCtrls.TIconOptions.Arrangement 4235
ComCtrls.TIconOptions.AutoArrange 4236
ComCtrls.TIconOptions.Create 4235
ComCtrls.TIconOptions.WrapText 4236
ComCtrls.TItemChange 5452
ComCtrls.TItemChange enumeration 5452
ComCtrls.TItemFind 5453
ComCtrls.TItemFind enumeration 5453
ComCtrls.TItemRequest 5489
ComCtrls.TItemRequest type 5489
ComCtrls.TItemRequests 5454
ComCtrls.TItemRequests enumeration 5454
ComCtrls.TItemState 5454
ComCtrls.TItemState enumeration 5454
ComCtrls.TItemStates 5489
ComCtrls.TItemStates type 5489
ComCtrls.TListArrangement 5455
ComCtrls.TListArrangement enumeration 5455
ComCtrls.TListColumn 4236
ComCtrls.TListColumn.Alignment 4242
ComCtrls.TListColumn.Assign 4239
ComCtrls.TListColumn.AutoSize 4242
ComCtrls.TListColumn.Caption 4242
ComCtrls.TListColumn.Create 4238
ComCtrls.TListColumn.DefineProperties 4240
ComCtrls.TListColumn.Destroy 4239
ComCtrls.TListColumn.GetDisplayName 4241
ComCtrls.TListColumn.ImageIndex 4244
ComCtrls.TListColumn.MaxWidth 4244
ComCtrls.TListColumn.MinWidth 4244
ComCtrls.TListColumn.SetIndex 4241
ComCtrls.TListColumn.Tag 4244
ComCtrls.TListColumn.Width 4245
ComCtrls.TListColumn.WidthType 4245
ComCtrls.TListColumns 4245
ComCtrls.TListColumns.Add 4248
ComCtrls.TListColumns.Create 4247
ComCtrls.TListColumns.GetOwner 4248
ComCtrls.TListColumns.Items 4250
ComCtrls.TListColumns.Owner 4249
ComCtrls.TListColumns.Update 4249
ComCtrls.TListHotTrackStyle 5456
ComCtrls.TListHotTrackStyle enumeration 5456

ComCtrls.TListHotTrackStyles 5489
ComCtrls.TListHotTrackStyles type 5489
ComCtrls.TListItem 4250
ComCtrls.TListItem.Assign 4253
ComCtrls.TListItem.CancelEdit 4254
ComCtrls.TListItem.Caption 4258
ComCtrls.TListItem.Checked 4258
ComCtrls.TListItem.Create 4252
ComCtrls.TListItem.Cut 4259
ComCtrls.TListItem.Data 4259
ComCtrls.TListItem.Delete 4254
ComCtrls.TListItem.Deleting 4260
ComCtrls.TListItem.Destroy 4252
ComCtrls.TListItem.DisplayRect 4254
ComCtrls.TListItem.DropTarget 4260
ComCtrls.TListItem.EditCaption 4255
ComCtrls.TListItem.Focused 4260
ComCtrls.TListItem.GetPosition 4255
ComCtrls.TListItem.Handle 4261
ComCtrls.TListItem.ImageIndex 4261
ComCtrls.TListItem.Indent 4263
ComCtrls.TListItem.Index 4264
ComCtrls.TListItem.Left 4265
ComCtrls.TListItem.ListView 4265
ComCtrls.TListItem.MakeVisible 4256
ComCtrls.TListItem.OverlayIndex 4266
ComCtrls.TListItem.Owner 4266
ComCtrls.TListItem.Position 4267
ComCtrls.TListItem.Selected 4267
ComCtrls.TListItem.SetPosition 4256
ComCtrls.TListItem.StateIndex 4267
ComCtrls.TListItem.SubItemImages 4268
ComCtrls.TListItem.SubItems 4268
ComCtrls.TListItem.Top 4270
ComCtrls.TListItem.Update 4256
ComCtrls.TListItem.WorkArea 4257
ComCtrls.TListItemClass 5490
ComCtrls.TListItemClass type 5490
ComCtrls.TListItems 4270
ComCtrls.TListItems.Add 4273
ComCtrls.TListItems.AddItem 4275
ComCtrls.TListItems.Assign 4275
ComCtrls.TListItems.BeginUpdate 4276
ComCtrls.TListItems.Clear 4276
ComCtrls.TListItems.Count 4281
ComCtrls.TListItems.Create 4272
ComCtrls.TListItems.CreateItem 4276
ComCtrls.TListItems.DefineProperties 4277
ComCtrls.TListItems.Delete 4277
ComCtrls.TListItems.Destroy 4272
ComCtrls.TListItems.EndUpdate 4277
ComCtrls.TListItems.GetCount 4278
ComCtrls.TListItems.GetEnumerator 4278
ComCtrls.TListItems.GetHandle 4278
ComCtrls.TListItems.GetItem 4278
ComCtrls.TListItems.Handle 4281
ComCtrls.TListItems.IndexOf 4278
ComCtrls.TListItems.Insert 4279
ComCtrls.TListItems.Item 4282
ComCtrls.TListItems.Owner 4283
ComCtrls.TListItems.SetCount 4280
ComCtrls.TListItems.SetItem 4280
ComCtrls.TListItems.SetUpdateState 4281
ComCtrls.TListItemsEnumerator 4283
ComCtrls.TListItemsEnumerator.Create 4284
ComCtrls.TListItemsEnumerator.Current 4285
ComCtrls.TListItemsEnumerator.GetCurrent 4284
ComCtrls.TListItemsEnumerator.MoveNext 4284
ComCtrls.TListView 4285
ComCtrls.TListView.Action 4300
ComCtrls.TListView.Align 4300
ComCtrls.TListView.AllocBy 4304
ComCtrls.TListView.Anchors 4304
ComCtrls.TListView.BevelEdges 4304
ComCtrls.TListView.BevelInner 4305
ComCtrls.TListView.BevelKind 4305
ComCtrls.TListView.BevelOuter 4306
ComCtrls.TListView.BevelWidth 4306

ComCtrls.TListView.BiDiMode 4307
ComCtrls.TListView.BorderStyle 4307
ComCtrls.TListView.BorderWidth 4307
ComCtrls.TListView.Checkboxes 4308
ComCtrls.TListView.Color 4308
ComCtrls.TListView.ColumnClick 4309
ComCtrls.TListView.Columns 4309
ComCtrls.TListView.Constraints 4311
ComCtrls.TListView.Ctl3D 4312
ComCtrls.TListView.DragCursor 4312
ComCtrls.TListView.DragKind 4312
ComCtrls.TListView.DragMode 4313
ComCtrls.TListView.Enabled 4314
ComCtrls.TListView.FlatScrollBars 4315
ComCtrls.TListView.Font 4315
ComCtrls.TListView.FullDrag 4316
ComCtrls.TListView.GridLines 4317
ComCtrls.TListView.HideSelection 4317
ComCtrls.TListView.HotTrack 4317
ComCtrls.TListView.HotTrackStyles 4318
ComCtrls.TListView.HoverTime 4318
ComCtrls.TListView.IconOptions 4318
ComCtrls.TListView.Items 4319
ComCtrls.TListView.LargeImages 4322
ComCtrls.TListView.MultiSelect 4324
ComCtrls.TListView.OnAdvancedCustomDraw 4346
ComCtrls.TListView.OnAdvancedCustomDrawItem 4347
ComCtrls.TListView.OnAdvancedCustomDrawSubItem 4347
ComCtrls.TListView.OnChange 4348
ComCtrls.TListView.OnChanging 4349
ComCtrls.TListView.OnClick 4349
ComCtrls.TListView.OnColumnClick 4351
ComCtrls.TListView.OnColumnDragged 4352
ComCtrls.TListView.OnColumnRightClick 4352
ComCtrls.TListView.OnCompare 4353
ComCtrls.TListView.OnContextPopup 4353
ComCtrls.TListView.OnCustomDraw 4354
ComCtrls.TListView.OnCustomDrawItem 4355
ComCtrls.TListView.OnCustomDrawSubItem 4355
ComCtrls.TListView.OnData 4356
ComCtrls.TListView.OnDataFind 4356
ComCtrls.TListView.OnDataHint 4357
ComCtrls.TListView.OnDataStateChange 4357
ComCtrls.TListView.OnDblClick 4358
ComCtrls.TListView.OnDeletion 4358
ComCtrls.TListView.OnDragDrop 4359
ComCtrls.TListView.OnDragOver 4361
ComCtrls.TListView.OnDrawItem 4363
ComCtrls.TListView.OnEdited 4364
ComCtrls.TListView.OnEditing 4364
ComCtrls.TListView.OnEndDock 4365
ComCtrls.TListView.OnEndDrag 4365
ComCtrls.TListView.OnEnter 4366
ComCtrls.TListView.OnExit 4369
ComCtrls.TListView.OnGetImageIndex 4370
ComCtrls.TListView.OnGetSubItemImage 4371
ComCtrls.TListView.OnInfoTip 4371
ComCtrls.TListView.OnInsert 4372
ComCtrls.TListView.OnKeyDown 4372
ComCtrls.TListView.OnKeyPress 4375
ComCtrls.TListView.OnKeyUp 4376
ComCtrls.TListView.OnMouseActivate 4378
ComCtrls.TListView.OnMouseDown 4379
ComCtrls.TListView.OnMouseEnter 4381
ComCtrls.TListView.OnMouseLeave 4381
ComCtrls.TListView.OnMouseMove 4381
ComCtrls.TListView.OnMouseUp 4384
ComCtrls.TListView.OnResize 4386
ComCtrls.TListView.OnSelectItem 4387
ComCtrls.TListView.OnStartDock 4387
ComCtrls.TListView.OnStartDrag 4387
ComCtrls.TListView.OwnerData 4325
ComCtrls.TListView.OwnerDraw 4325
ComCtrls.TListView.ParentBiDiMode 4326
ComCtrls.TListView.ParentColor 4326
ComCtrls.TListView.ParentFont 4327
ComCtrls.TListView.ParentShowHint 4327
ComCtrls.TListView.PopupMenu 4328

ComCtrls.TListView.ReadOnly 4332
ComCtrls.TListView.RowSelect 4333
ComCtrls.TListView.ShowColumnHeaders 4333
ComCtrls.TListView.ShowHint 4334
ComCtrls.TListView.ShowWorkAreas 4335
ComCtrls.TListView.SmallImages 4336
ComCtrls.TListView.SortType 4338
ComCtrls.TListView.StateImages 4339
ComCtrls.TListView.TabOrder 4339
ComCtrls.TListView.TabStop 4340
ComCtrls.TListView.ViewStyle 4340
ComCtrls.TListView.Visible 4343
ComCtrls.TListViewActionLink 4388
ComCtrls.TListViewActionLink.AddItem 4391
ComCtrls.TListViewActionLink.SetImages 4392
ComCtrls.TLVAdvancedCustomDrawEvent 5490
ComCtrls.TLVAdvancedCustomDrawEvent type 5490
ComCtrls.TLVAdvancedCustomDrawItemEvent 5491
ComCtrls.TLVAdvancedCustomDrawItemEvent type 5491
ComCtrls.TLVAdvancedCustomDrawSubItemEvent 5491
ComCtrls.TLVAdvancedCustomDrawSubItemEvent type 5491
ComCtrls.TLVChangeEvent 5491
ComCtrls.TLVChangeEvent type 5491
ComCtrls.TLVChangingEvent 5492
ComCtrls.TLVChangingEvent type 5492
ComCtrls.TLVColumnClickEvent 5492
ComCtrls.TLVColumnClickEvent type 5492
ComCtrls.TLVColumnRClickEvent 5492
ComCtrls.TLVColumnRClickEvent type 5492
ComCtrls.TLVCompareEvent 5493
ComCtrls.TLVCompareEvent type 5493
ComCtrls.TLVCreateItemClassEvent 5493
ComCtrls.TLVCreateItemClassEvent type 5493
ComCtrls.TLVCustomDrawEvent 5493
ComCtrls.TLVCustomDrawEvent type 5493
ComCtrls.TLVCustomDrawItemEvent 5494
ComCtrls.TLVCustomDrawItemEvent type 5494
ComCtrls.TLVCustomDrawSubItemEvent 5494
ComCtrls.TLVCustomDrawSubItemEvent type 5494
ComCtrls.TLVDeletedEvent 5494
ComCtrls.TLVDeletedEvent type 5494
ComCtrls.TLVDrawItemEvent 5495
ComCtrls.TLVDrawItemEvent type 5495
ComCtrls.TLVEditedEvent 5495
ComCtrls.TLVEditedEvent type 5495
ComCtrls.TLVEditingEvent 5496
ComCtrls.TLVEditingEvent type 5496
ComCtrls.TLVInfoTipEvent 5496
ComCtrls.TLVInfoTipEvent type 5496
ComCtrls.TLVNotifyEvent 5496
ComCtrls.TLVNotifyEvent type 5496
ComCtrls.TLVOwnerDataEvent 5497
ComCtrls.TLVOwnerDataEvent type 5497
ComCtrls.TLVOwnerDataFindEvent 5497
ComCtrls.TLVOwnerDataFindEvent type 5497
ComCtrls.TLVOwnerDataHintEvent 5498
ComCtrls.TLVOwnerDataHintEvent type 5498
ComCtrls.TLVOwnerDataStateChangeEvent 5498
ComCtrls.TLVOwnerDataStateChangeEvent type 5498
ComCtrls.TLVSelectItemEvent 5499
ComCtrls.TLVSelectItemEvent type 5499
ComCtrls.TLVSubItemImageEvent 5499
ComCtrls.TLVSubItemImageEvent type 5499
ComCtrls.TMonthCalColors 4392
ComCtrls.TMonthCalColors.Assign 4394
ComCtrls.TMonthCalColors.BackColor 4395
ComCtrls.TMonthCalColors.Create 4394
ComCtrls.TMonthCalColors.MonthBackColor 4395
ComCtrls.TMonthCalColors.TextColor 4395
ComCtrls.TMonthCalColors.TitleBackColor 4396
ComCtrls.TMonthCalColors.TitleTextColor 4396
ComCtrls.TMonthCalColors.TrailingTextColor 4396
ComCtrls.TMonthCalendar 4397
ComCtrls.TMonthCalendar.Align 4412
ComCtrls.TMonthCalendar.Anchors 4416
ComCtrls.TMonthCalendar.AutoSize 4416
ComCtrls.TMonthCalendar.BiDiMode 4417
ComCtrls.TMonthCalendar.BorderWidth 4417

ComCtrls.TMonthCalendar.CalColors 4417
ComCtrls.TMonthCalendar.CanAutoSize 4408
ComCtrls.TMonthCalendar.ConstrainedResize 4409
ComCtrls.TMonthCalendar.Constraints 4418
ComCtrls.TMonthCalendar.Create 4408
ComCtrls.TMonthCalendar.CreateParams 4409
ComCtrls.TMonthCalendar.Date 4418
ComCtrls.TMonthCalendar.DragCursor 4419
ComCtrls.TMonthCalendar.DragKind 4419
ComCtrls.TMonthCalendar.DragMode 4420
ComCtrls.TMonthCalendar.Enabled 4421
ComCtrls.TMonthCalendar.EndDate 4422
ComCtrls.TMonthCalendar.FirstDayOfWeek 4422
ComCtrls.TMonthCalendar.Font 4422
ComCtrls.TMonthCalendar.GetCalendarHandle 4410
ComCtrls.TMonthCalendar.ImeMode 4423
ComCtrls.TMonthCalendar.ImeName 4424
ComCtrls.TMonthCalendar.MaxDate 4425
ComCtrls.TMonthCalendar.MaxSelectRange 4425
ComCtrls.TMonthCalendar.MinDate 4425
ComCtrls.TMonthCalendar.MsgSetCalColors 4410
ComCtrls.TMonthCalendar.MsgSetDateTime 4410
ComCtrls.TMonthCalendar.MsgSetRange 4411
ComCtrls.TMonthCalendar.MultiSelect 4426
ComCtrls.TMonthCalendar.OnClick 4438
ComCtrls.TMonthCalendar.OnContextPopup 4440
ComCtrls.TMonthCalendar.OnDblClick 4440
ComCtrls.TMonthCalendar.OnDragDrop 4441
ComCtrls.TMonthCalendar.OnDragOver 4443
ComCtrls.TMonthCalendar.OnEndDock 4445
ComCtrls.TMonthCalendar.OnEndDrag 4446
ComCtrls.TMonthCalendar.OnEnter 4447
ComCtrls.TMonthCalendar.OnExit 4449
ComCtrls.TMonthCalendar.OnGetMonthInfo 4451
ComCtrls.TMonthCalendar.OnKeyDown 4451
ComCtrls.TMonthCalendar.OnKeyPress 4454
ComCtrls.TMonthCalendar.OnKeyUp 4455
ComCtrls.TMonthCalendar.OnMouseEnter 4457
ComCtrls.TMonthCalendar.OnMouseLeave 4457
ComCtrls.TMonthCalendar.OnStartDock 4457
ComCtrls.TMonthCalendar.OnStartDrag 4458
ComCtrls.TMonthCalendar.ParentBiDiMode 4426
ComCtrls.TMonthCalendar.ParentFont 4427
ComCtrls.TMonthCalendar.ParentShowHint 4427
ComCtrls.TMonthCalendar.PopupMenu 4428
ComCtrls.TMonthCalendar.ShowHint 4432
ComCtrls.TMonthCalendar.ShowToday 4434
ComCtrls.TMonthCalendar.ShowTodayCircle 4434
ComCtrls.TMonthCalendar.TabOrder 4434
ComCtrls.TMonthCalendar.TabStop 4435
ComCtrls.TMonthCalendar.Visible 4435
ComCtrls.TMonthCalendar.WeekNumbers 4437
ComCtrls.TMultiSelectStyle 5499
ComCtrls.TMultiSelectStyle type 5499
ComCtrls.TMultiSelectStyles 5456
ComCtrls.TMultiSelectStyles enumeration 5456
ComCtrls.TNodeAttachMode 5457
ComCtrls.TNodeAttachMode enumeration 5457
ComCtrls.TNodeCache 5457
ComCtrls.TNodeCache record 5457
ComCtrls.TNodeDataInfo 5458
ComCtrls.TNodeDataInfo record 5458
ComCtrls.TNodeInfo 5458
ComCtrls.TNodeInfo record 5458
ComCtrls.TNodeState 5459
ComCtrls.TNodeState enumeration 5459
ComCtrls.TNumberingStyle 5459
ComCtrls.TNumberingStyle enumeration 5459
ComCtrls.TOnGetMonthInfoEvent 5500
ComCtrls.TOnGetMonthInfoEvent type 5500
ComCtrls.TOwnerDrawState 5500
ComCtrls.TOwnerDrawState type 5500
ComCtrls.TPageControl 4459
ComCtrls.TPageControl.ActivePage 4482
ComCtrls.TPageControl.ActivePageIndex 4484
ComCtrls.TPageControl.Align 4485
ComCtrls.TPageControl.Anchors 4488
ComCtrls.TPageControl.BiDiMode 4489

ComCtrls.TPageControl.CanShowTab 4473
ComCtrls.TPageControl.Change 4473
ComCtrls.TPageControl.Constraints 4489
ComCtrls.TPageControl.Create 4470
ComCtrls.TPageControl.Destroy 4472
ComCtrls.TPageControl.DoAddDockClient 4474
ComCtrls.TPageControl.DockOver 4474
ComCtrls.TPageControl.DockSite 4490
ComCtrls.TPageControl.DoRemoveDockClient 4474
ComCtrls.TPageControl.DragCursor 4491
ComCtrls.TPageControl.DragKind 4491
ComCtrls.TPageControl.DragMode 4491
ComCtrls.TPageControl.Enabled 4493
ComCtrls.TPageControl.FindNextPage 4475
ComCtrls.TPageControl.Font 4493
ComCtrls.TPageControl.GetChildren 4476
ComCtrls.TPageControl.GetImageIndex 4477
ComCtrls.TPageControl.GetPageFromDockClient 4477
ComCtrls.TPageControl.GetSiteInfo 4477
ComCtrls.TPageControl.HotTrack 4495
ComCtrls.TPageControl.Images 4495
ComCtrls.TPageControl.Loaded 4478
ComCtrls.TPageControl.MultiLine 4495
ComCtrls.TPageControl.OnChange 4516
ComCtrls.TPageControl.OnChanging 4516
ComCtrls.TPageControl.OnContextPopup 4517
ComCtrls.TPageControl.OnDockDrop 4518
ComCtrls.TPageControl.OnDockOver 4520
ComCtrls.TPageControl.OnDragDrop 4522
ComCtrls.TPageControl.OnDragOver 4524
ComCtrls.TPageControl.OnDrawTab 4527
ComCtrls.TPageControl.OnEndDock 4527
ComCtrls.TPageControl.OnEndDrag 4527
ComCtrls.TPageControl.OnEnter 4528
ComCtrls.TPageControl.OnExit 4531
ComCtrls.TPageControl.OnGetImageIndex 4532
ComCtrls.TPageControl.OnGetSiteInfo 4533
ComCtrls.TPageControl.OnMouseActivate 4535
ComCtrls.TPageControl.OnMouseDown 4536
ComCtrls.TPageControl.OnMouseEnter 4538
ComCtrls.TPageControl.OnMouseLeave 4538
ComCtrls.TPageControl.OnMouseMove 4539
ComCtrls.TPageControl.OnMouseUp 4541
ComCtrls.TPageControl.OnResize 4543
ComCtrls.TPageControl.OnStartDock 4544
ComCtrls.TPageControl.OnStartDrag 4544
ComCtrls.TPageControl.OnUnDock 4545
ComCtrls.TPageControl.OwnerDraw 4497
ComCtrls.TPageControl.PageCount 4497
ComCtrls.TPageControl.Pages 4498
ComCtrls.TPageControl.ParentBiDiMode 4500
ComCtrls.TPageControl.ParentFont 4500
ComCtrls.TPageControl.ParentShowHint 4501
ComCtrls.TPageControl.PopupMenu 4501
ComCtrls.TPageControl.RaggedRight 4506
ComCtrls.TPageControl.ScrollOpposite 4506
ComCtrls.TPageControl.SelectNextPage 4478
ComCtrls.TPageControl.SetActivePage 4480
ComCtrls.TPageControl.SetChildOrder 4480
ComCtrls.TPageControl.SetTabIndex 4480
ComCtrls.TPageControl.ShowControl 4481
ComCtrls.TPageControl.ShowHint 4506
ComCtrls.TPageControl.Style 4508
ComCtrls.TPageControl.TabHeight 4508
ComCtrls.TPageControl.TabIndex 4509
ComCtrls.TPageControl.TabOrder 4511
ComCtrls.TPageControl.TabPosition 4512
ComCtrls.TPageControl.TabStop 4512
ComCtrls.TPageControl.TabWidth 4512
ComCtrls.TPageControl.UpdateActivePage 4481
ComCtrls.TPageControl.Visible 4514
ComCtrls.TPageScroller 4548
ComCtrls.TPageScroller.Align 4562
ComCtrls.TPageScroller.AlignControls 4559
ComCtrls.TPageScroller.Anchors 4565
ComCtrls.TPageScroller.AutoScroll 4566
ComCtrls.TPageScroller.BorderWidth 4566
ComCtrls.TPageScroller.ButtonSize 4567

ComCtrls.TPageScroller.Color 4567
ComCtrls.TPageScroller.Constraints 4568
ComCtrls.TPageScroller.Control 4568
ComCtrls.TPageScroller.Create 4558
ComCtrls.TPageScroller.CreateParams 4559
ComCtrls.TPageScroller.CreateWnd 4560
ComCtrls.TPageScroller.DockSite 4568
ComCtrls.TPageScroller.DragCursor 4569
ComCtrls.TPageScroller.DragKind 4570
ComCtrls.TPageScroller.DragMode 4570
ComCtrls.TPageScroller.DragScroll 4572
ComCtrls.TPageScroller.Enabled 4572
ComCtrls.TPageScroller.Font 4572
ComCtrls.TPageScroller.GetButtonState 4560
ComCtrls.TPageScroller.Margin 4574
ComCtrls.TPageScroller.Notification 4560
ComCtrls.TPageScroller.OnClick 4585
ComCtrls.TPageScroller.OnContextPopup 4587
ComCtrls.TPageScroller.OnDblClick 4588
ComCtrls.TPageScroller.OnDragDrop 4588
ComCtrls.TPageScroller.OnDragOver 4590
ComCtrls.TPageScroller.OnEndDock 4593
ComCtrls.TPageScroller.OnEndDrag 4593
ComCtrls.TPageScroller.OnEnter 4594
ComCtrls.TPageScroller.OnExit 4597
ComCtrls.TPageScroller.OnKeyDown 4598
ComCtrls.TPageScroller.OnKeyPress 4601
ComCtrls.TPageScroller.OnKeyUp 4602
ComCtrls.TPageScroller.OnMouseWheel 4604
ComCtrls.TPageScroller.OnResize 4604
ComCtrls.TPageScroller.OnScroll 4605
ComCtrls.TPageScroller.OnStartDock 4605
ComCtrls.TPageScroller.OnStartDrag 4605
ComCtrls.TPageScroller.Orientation 4574
ComCtrls.TPageScroller.ParentBackground 4574
ComCtrls.TPageScroller.ParentColor 4574
ComCtrls.TPageScroller.ParentFont 4575
ComCtrls.TPageScroller.ParentShowHint 4575
ComCtrls.TPageScroller.PopupMenu 4576
ComCtrls.TPageScroller.Position 4580
ComCtrls.TPageScroller.Scroll 4561
ComCtrls.TPageScroller.ShowHint 4581
ComCtrls.TPageScroller.TabOrder 4582
ComCtrls.TPageScroller.TabStop 4583
ComCtrls.TPageScroller.Visible 4583
ComCtrls.TPageScrollerButton 5460
ComCtrls.TPageScrollerButton enumeration 5460
ComCtrls.TPageScrollerButtonState 5460
ComCtrls.TPageScrollerButtonState enumeration 5460
ComCtrls.TPageScrollerOrientation 5461
ComCtrls.TPageScrollerOrientation enumeration 5461
ComCtrls.TPageScrollEvent 5501
ComCtrls.TPageScrollEvent type 5501
ComCtrls.TParaAttributes 4606
ComCtrls.TParaAttributes.Alignment 4609
ComCtrls.TParaAttributes.Assign 4608
ComCtrls.TParaAttributes.Create 4608
ComCtrls.TParaAttributes.FirstIndent 4610
ComCtrls.TParaAttributes.LeftIndent 4612
ComCtrls.TParaAttributes.Numbering 4612
ComCtrls.TParaAttributes.RightIndent 4614
ComCtrls.TParaAttributes.Tab 4614
ComCtrls.TParaAttributes.TabCount 4615
ComCtrls.TPositionToolTip 5462
ComCtrls.TPositionToolTip enumeration 5462
ComCtrls.TProgressBar 4615
ComCtrls.TProgressBar.Align 4633
ComCtrls.TProgressBar.Anchors 4637
ComCtrls.TProgressBar.BorderWidth 4637
ComCtrls.TProgressBar.Constraints 4637
ComCtrls.TProgressBar.Create 4626
ComCtrls.TProgressBar.CreateParams 4627
ComCtrls.TProgressBar.CreateWnd 4628
ComCtrls.TProgressBar.DestroyWnd 4628
ComCtrls.TProgressBar.DragCursor 4638
ComCtrls.TProgressBar.DragKind 4638
ComCtrls.TProgressBar.DragMode 4639
ComCtrls.TProgressBar.Enabled 4640

ComCtrls.TProgressBar.Hint 4641
ComCtrls.TProgressBar.Max 4643
ComCtrls.TProgressBar.Min 4646
ComCtrls.TProgressBar.OnContextPopup 4665
ComCtrls.TProgressBar.OnDragDrop 4665
ComCtrls.TProgressBar.OnDragOver 4668
ComCtrls.TProgressBar.OnEndDock 4670
ComCtrls.TProgressBar.OnEndDrag 4670
ComCtrls.TProgressBar.OnEnter 4671
ComCtrls.TProgressBar.OnExit 4674
ComCtrls.TProgressBar.OnMouseActivate 4675
ComCtrls.TProgressBar.OnMouseDown 4676
ComCtrls.TProgressBar.OnMouseEnter 4679
ComCtrls.TProgressBar.OnMouseLeave 4679
ComCtrls.TProgressBar.OnMouseMove 4679
ComCtrls.TProgressBar.OnMouseUp 4681
ComCtrls.TProgressBar.OnStartDock 4684
ComCtrls.TProgressBar.OnStartDrag 4684
ComCtrls.TProgressBar.Orientation 4649
ComCtrls.TProgressBar.ParentShowHint 4649
ComCtrls.TProgressBar.PopupMenu 4650
ComCtrls.TProgressBar.Position 4655
ComCtrls.TProgressBar.ShowHint 4658
ComCtrls.TProgressBar.Smooth 4660
ComCtrls.TProgressBar.Step 4660
ComCtrls.TProgressBar.StepBy 4629
ComCtrls.TProgressBar.StepIt 4631
ComCtrls.TProgressBar.TabOrder 4662
ComCtrls.TProgressBar.TabStop 4662
ComCtrls.TProgressBar.Visible 4663
ComCtrls.TProgressBarOrientation 5462
ComCtrls.TProgressBarOrientation enumeration 5462
ComCtrls.TProgressRange 5502
ComCtrls.TProgressRange type 5502
ComCtrls.TRichEdit 4685
ComCtrls.TRichEdit.Align 4699
ComCtrls.TRichEdit.Alignment 4703
ComCtrls.TRichEdit.Anchors 4703
ComCtrls.TRichEdit.BevelEdges 4704
ComCtrls.TRichEdit.BevelInner 4704
ComCtrls.TRichEdit.BevelKind 4705
ComCtrls.TRichEdit.BevelOuter 4705
ComCtrls.TRichEdit.BevelWidth 4705
ComCtrls.TRichEdit.BiDiMode 4706
ComCtrls.TRichEdit.BorderStyle 4706
ComCtrls.TRichEdit.BorderWidth 4707
ComCtrls.TRichEdit.Color 4707
ComCtrls.TRichEdit.Constraints 4708
ComCtrls.TRichEdit.Ctl3D 4708
ComCtrls.TRichEdit.DragCursor 4709
ComCtrls.TRichEdit.DragKind 4709
ComCtrls.TRichEdit.DragMode 4710
ComCtrls.TRichEdit.Enabled 4711
ComCtrls.TRichEdit.Font 4712
ComCtrls.TRichEdit.HideScrollBars 4713
ComCtrls.TRichEdit.HideSelection 4715
ComCtrls.TRichEdit.ImeMode 4716
ComCtrls.TRichEdit.ImeName 4716
ComCtrls.TRichEdit.Lines 4717
ComCtrls.TRichEdit.MaxLength 4719
ComCtrls.TRichEdit.OnChange 4738
ComCtrls.TRichEdit.OnClick 4738
ComCtrls.TRichEdit.OnContextPopup 4740
ComCtrls.TRichEdit.OnDblClick 4741
ComCtrls.TRichEdit.OnDragDrop 4741
ComCtrls.TRichEdit.OnDragOver 4743
ComCtrls.TRichEdit.OnEndDock 4746
ComCtrls.TRichEdit.OnEndDrag 4746
ComCtrls.TRichEdit.OnEnter 4747
ComCtrls.TRichEdit.OnExit 4749
ComCtrls.TRichEdit.OnKeyDown 4751
ComCtrls.TRichEdit.OnKeyPress 4754
ComCtrls.TRichEdit.OnKeyUp 4755
ComCtrls.TRichEdit.OnMouseActivate 4757
ComCtrls.TRichEdit.OnMouseDown 4758
ComCtrls.TRichEdit.OnMouseEnter 4760
ComCtrls.TRichEdit.OnMouseLeave 4760
ComCtrls.TRichEdit.OnMouseMove 4760

ComCtrls.TRichEdit.OnMouseUp 4763
ComCtrls.TRichEdit.OnMouseWheel 4765
ComCtrls.TRichEdit.OnMouseWheelDown 4765
ComCtrls.TRichEdit.OnMouseWheelUp 4766
ComCtrls.TRichEdit.OnProtectChange 4766
ComCtrls.TRichEdit.OnResizeRequest 4766
ComCtrls.TRichEdit.OnSaveClipboard 4767
ComCtrls.TRichEdit.OnSelectionChange 4767
ComCtrls.TRichEdit.OnStartDock 4768
ComCtrls.TRichEdit.OnStartDrag 4768
ComCtrls.TRichEdit.ParentBiDiMode 4719
ComCtrls.TRichEdit.ParentColor 4720
ComCtrls.TRichEdit.ParentCtl3D 4720
ComCtrls.TRichEdit.ParentFont 4721
ComCtrls.TRichEdit.ParentShowHint 4721
ComCtrls.TRichEdit.PlainText 4722
ComCtrls.TRichEdit.PopupMenu 4723
ComCtrls.TRichEdit.ReadOnly 4727
ComCtrls.TRichEdit.ScrollBars 4728
ComCtrls.TRichEdit.ShowHint 4730
ComCtrls.TRichEdit.TabOrder 4731
ComCtrls.TRichEdit.TabStop 4732
ComCtrls.TRichEdit.Visible 4732
ComCtrls.TRichEdit.WantReturns 4734
ComCtrls.TRichEdit.WantTabs 4734
ComCtrls.TRichEdit.WordWrap 4735
ComCtrls.TRichEditProtectChange 5502
ComCtrls.TRichEditProtectChange type 5502
ComCtrls.TRichEditResizeEvent 5502
ComCtrls.TRichEditResizeEvent type 5502
ComCtrls.TRichEditSaveClipboard 5503
ComCtrls.TRichEditSaveClipboard type 5503
ComCtrls.TRichEditStreamInfo 5463
ComCtrls.TRichEditStreamInfo record 5463
ComCtrls.TSBCreatePanelClassEvent 5503
ComCtrls.TSBCreatePanelClassEvent type 5503
ComCtrls.TSearchDirection 5463
ComCtrls.TSearchDirection enumeration 5463
ComCtrls.TSearchType 5464
ComCtrls.TSearchType enumeration 5464
ComCtrls.TSearchTypes 5503
ComCtrls.TSearchTypes type 5503
ComCtrls.TSectionDragEvent 5505
ComCtrls.TSectionDragEvent type 5505
ComCtrls.TSectionNotifyEvent 5506
ComCtrls.TSectionNotifyEvent type 5506
ComCtrls.TSectionTrackEvent 5506
ComCtrls.TSectionTrackEvent type 5506
ComCtrls.TSectionTrackState 5464
ComCtrls.TSectionTrackState enumeration 5464
ComCtrls.TSortType 5465
ComCtrls.TSortType enumeration 5465
ComCtrls.TStatusBar 4769
ComCtrls.TStatusBar.Action 4781
ComCtrls.TStatusBar.Align 4781
ComCtrls.TStatusBar.Anchors 4785
ComCtrls.TStatusBar.AutoHint 4786
ComCtrls.TStatusBar.BiDiMode 4786
ComCtrls.TStatusBar.BorderWidth 4786
ComCtrls.TStatusBar.Color 4787
ComCtrls.TStatusBar.Constraints 4787
ComCtrls.TStatusBar.DragCursor 4788
ComCtrls.TStatusBar.DragKind 4788
ComCtrls.TStatusBar.DragMode 4789
ComCtrls.TStatusBar.Enabled 4790
ComCtrls.TStatusBar.Font 4791
ComCtrls.TStatusBar.OnClick 4804
ComCtrls.TStatusBar.OnContextPopup 4806
ComCtrls.TStatusBar.OnCreatePanelClass 4806
ComCtrls.TStatusBar.OnDblClick 4807
ComCtrls.TStatusBar.OnDragDrop 4807
ComCtrls.TStatusBar.OnDragOver 4809
ComCtrls.TStatusBar.OnDrawPanel 4812
ComCtrls.TStatusBar.OnEndDock 4813
ComCtrls.TStatusBar.OnEndDrag 4813
ComCtrls.TStatusBar.OnHint 4814
ComCtrls.TStatusBar.OnMouseActivate 4814
ComCtrls.TStatusBar.OnMouseDown 4815

ComCtrls.TStatusBar.OnMouseEnter 4818
ComCtrls.TStatusBar.OnMouseLeave 4818
ComCtrls.TStatusBar.OnMouseMove 4818
ComCtrls.TStatusBar.OnMouseUp 4820
ComCtrls.TStatusBar.OnResize 4823
ComCtrls.TStatusBar.OnStartDock 4823
ComCtrls.TStatusBar.OnStartDrag 4824
ComCtrls.TStatusBar.Panels 4792
ComCtrls.TStatusBar.ParentBiDiMode 4792
ComCtrls.TStatusBar.ParentColor 4793
ComCtrls.TStatusBar.ParentFont 4793
ComCtrls.TStatusBar.ParentShowHint 4794
ComCtrls.TStatusBar.PopupMenu 4794
ComCtrls.TStatusBar.ShowHint 4799
ComCtrls.TStatusBar.SimplePanel 4800
ComCtrls.TStatusBar.SimpleText 4801
ComCtrls.TStatusBar.SizeGrip 4801
ComCtrls.TStatusBar.UseSystemFont 4801
ComCtrls.TStatusBar.Visible 4802
ComCtrls.TStatusPanel 4825
ComCtrls.TStatusPanel.Alignment 4829
ComCtrls.TStatusPanel.Assign 4827
ComCtrls.TStatusPanel.Bevel 4829
ComCtrls.TStatusPanel.BiDiMode 4830
ComCtrls.TStatusPanel.Create 4827
ComCtrls.TStatusPanel.GetDisplayName 4828
ComCtrls.TStatusPanel.ParentBiDiMode 4830
ComCtrls.TStatusPanel.ParentBiDiModeChanged 4828
ComCtrls.TStatusPanel.Style 4830
ComCtrls.TStatusPanel.Text 4831
ComCtrls.TStatusPanel.UseRightToLeftAlignment 4828
ComCtrls.TStatusPanel.UseRightToLeftReading 4828
ComCtrls.TStatusPanel.Width 4834
ComCtrls.TStatusPanelBevel 5466
ComCtrls.TStatusPanelBevel enumeration 5466
ComCtrls.TStatusPanelClass 5507
ComCtrls.TStatusPanelClass type 5507
ComCtrls.TStatusPanels 4834
ComCtrls.TStatusPanels.Add 4837
ComCtrls.TStatusPanels.AddItem 4838
ComCtrls.TStatusPanels.Create 4836
ComCtrls.TStatusPanels.GetOwner 4839
ComCtrls.TStatusPanels.Insert 4839
ComCtrls.TStatusPanels.Items 4840
ComCtrls.TStatusPanels.Update 4839
ComCtrls.TStatusPanelStyle 5466
ComCtrls.TStatusPanelStyle enumeration 5466
ComCtrls.TTabChangingEvent 5507
ComCtrls.TTabChangingEvent type 5507
ComCtrls.TTabControl 4841
ComCtrls.TTabControl.Align 4853
ComCtrls.TTabControl.Anchors 4857
ComCtrls.TTabControl.BiDiMode 4858
ComCtrls.TTabControl.Constraints 4858
ComCtrls.TTabControl.DisplayRect 4859
ComCtrls.TTabControl.DockSite 4859
ComCtrls.TTabControl.DragCursor 4860
ComCtrls.TTabControl.DragKind 4860
ComCtrls.TTabControl.DragMode 4861
ComCtrls.TTabControl.Enabled 4862
ComCtrls.TTabControl.Font 4863
ComCtrls.TTabControl.HotTrack 4864
ComCtrls.TTabControl.Images 4864
ComCtrls.TTabControl.MultiLine 4865
ComCtrls.TTabControl.MultiSelect 4866
ComCtrls.TTabControl.OnChange 4884
ComCtrls.TTabControl.OnChanging 4888
ComCtrls.TTabControl.OnContextPopup 4889
ComCtrls.TTabControl.OnDockDrop 4890
ComCtrls.TTabControl.OnDockOver 4892
ComCtrls.TTabControl.OnDragDrop 4894
ComCtrls.TTabControl.OnDragOver 4896
ComCtrls.TTabControl.OnDrawTab 4898
ComCtrls.TTabControl.OnEndDock 4899
ComCtrls.TTabControl.OnEndDrag 4899
ComCtrls.TTabControl.OnEnter 4900
ComCtrls.TTabControl.OnExit 4903
ComCtrls.TTabControl.OnGetImageIndex 4904

ComCtrls.TTabControl.OnGetSiteInfo 4904
ComCtrls.TTabControl.OnMouseActivate 4907
ComCtrls.TTabControl.OnMouseDown 4908
ComCtrls.TTabControl.OnMouseEnter 4910
ComCtrls.TTabControl.OnMouseLeave 4910
ComCtrls.TTabControl.OnMouseMove 4910
ComCtrls.TTabControl.OnMouseUp 4913
ComCtrls.TTabControl.OnResize 4915
ComCtrls.TTabControl.OnStartDock 4915
ComCtrls.TTabControl.OnStartDrag 4916
ComCtrls.TTabControl.OnUnDock 4917
ComCtrls.TTabControl.OwnerDraw 4866
ComCtrls.TTabControl.ParentBiDiMode 4866
ComCtrls.TTabControl.ParentFont 4867
ComCtrls.TTabControl.ParentShowHint 4867
ComCtrls.TTabControl.PopupMenu 4868
ComCtrls.TTabControl.RaggedRight 4873
ComCtrls.TTabControl.ScrollOpposite 4873
ComCtrls.TTabControl.ShowHint 4873
ComCtrls.TTabControl.Style 4875
ComCtrls.TTabControl.TabHeight 4875
ComCtrls.TTabControl.TabIndex 4876
ComCtrls.TTabControl.TabOrder 4878
ComCtrls.TTabControl.TabPosition 4879
ComCtrls.TTabControl.Tabs 4879
ComCtrls.TTabControl.TabStop 4881
ComCtrls.TTabControl.TabWidth 4881
ComCtrls.TTabControl.Visible 4882
ComCtrls.TTabGetImageEvent 5507
ComCtrls.TTabGetImageEvent type 5507
ComCtrls.TTabPosition 5467
ComCtrls.TTabPosition enumeration 5467
ComCtrls.TTabSheet 4919
ComCtrls.TTabSheet.BorderWidth 4934
ComCtrls.TTabSheet.Caption 4935
ComCtrls.TTabSheet.Constraints 4937
ComCtrls.TTabSheet.Create 4930
ComCtrls.TTabSheet.CreateParams 4932
ComCtrls.TTabSheet.Destroy 4932
ComCtrls.TTabSheet.DoHide 4933
ComCtrls.TTabSheet.DoShow 4933
ComCtrls.TTabSheet.DragMode 4938
ComCtrls.TTabSheet.Enabled 4940
ComCtrls.TTabSheet.Font 4940
ComCtrls.TTabSheet.Height 4941
ComCtrls.TTabSheet.Highlighted 4942
ComCtrls.TTabSheet.ImageIndex 4943
ComCtrls.TTabSheet.Left 4943
ComCtrls.TTabSheet.OnContextPopup 4961
ComCtrls.TTabSheet.OnDragDrop 4962
ComCtrls.TTabSheet.OnDragOver 4964
ComCtrls.TTabSheet.OnEndDrag 4966
ComCtrls.TTabSheet.OnEnter 4967
ComCtrls.TTabSheet.OnExit 4970
ComCtrls.TTabSheet.OnHide 4971
ComCtrls.TTabSheet.OnMouseActivate 4972
ComCtrls.TTabSheet.OnMouseDown 4973
ComCtrls.TTabSheet.OnMouseEnter 4975
ComCtrls.TTabSheet.OnMouseLeave 4975
ComCtrls.TTabSheet.OnMouseMove 4975
ComCtrls.TTabSheet.OnMouseUp 4978
ComCtrls.TTabSheet.OnResize 4980
ComCtrls.TTabSheet.OnShow 4980
ComCtrls.TTabSheet.OnStartDrag 4981
ComCtrls.TTabSheet.PageControl 4944
ComCtrls.TTabSheet.PageIndex 4946
ComCtrls.TTabSheet.ParentFont 4948
ComCtrls.TTabSheet.ParentShowHint 4948
ComCtrls.TTabSheet.PopupMenu 4949
ComCtrls.TTabSheet.ReadState 4933
ComCtrls.TTabSheet.ShowHint 4953
ComCtrls.TTabSheet.TabIndex 4955
ComCtrls.TTabSheet.TabVisible 4956
ComCtrls.TTabSheet.Top 4957
ComCtrls.TTabSheet.UpdateControlOriginalParentSize 4934
ComCtrls.TTabSheet.Visible 4958
ComCtrls.TTabSheet.Width 4960
ComCtrls.TTabStyle 5467

ComCtrls.TTabStyle enumeration 5467
ComCtrls.TTBAdvancedCustomDrawBtnEvent 5508
ComCtrls.TTBAdvancedCustomDrawBtnEvent type 5508
ComCtrls.TTBAdvancedCustomDrawEvent 5508
ComCtrls.TTBAdvancedCustomDrawEvent type 5508
ComCtrls.TTBButtonEvent 5508
ComCtrls.TTBButtonEvent type 5508
ComCtrls.TTBCustomDrawBtnEvent 5509
ComCtrls.TTBCustomDrawBtnEvent type 5509
ComCtrls.TTBCustomDrawEvent 5509
ComCtrls.TTBCustomDrawEvent type 5509
ComCtrls.TTBCustomDrawFlags 5509
ComCtrls.TTBCustomDrawFlags type 5509
ComCtrls.TTBCustomizeQueryEvent 5510
ComCtrls.TTBCustomizeQueryEvent type 5510
ComCtrls.TTBDrawingStyle 5468
ComCtrls.TTBDrawingStyle enumeration 5468
ComCtrls.TTBGradientDrawingOptions 5511
ComCtrls.TTBGradientDrawingOptions type 5511
ComCtrls.TTBNewButtonEvent 5511
ComCtrls.TTBNewButtonEvent type 5511
ComCtrls.TTextAttributes 4982
ComCtrls.TTextAttributes.Assign 4984
ComCtrls.TTextAttributes.AssignTo 4986
ComCtrls.TTextAttributes.Charset 4987
ComCtrls.TTextAttributes.Color 4987
ComCtrls.TTextAttributes.ConsistentAttributes 4989
ComCtrls.TTextAttributes.Create 4983
ComCtrls.TTextAttributes.Height 4989
ComCtrls.TTextAttributes.InitFormat 4986
ComCtrls.TTextAttributes.Name 4991
ComCtrls.TTextAttributes.Pitch 4991
ComCtrls.TTextAttributes.Protected 4992
ComCtrls.TTextAttributes.Size 4992
ComCtrls.TTextAttributes.Style 4992
ComCtrls.TTickMark 5468
ComCtrls.TTickMark enumeration 5468
ComCtrls.TTickStyle 5469
ComCtrls.TTickStyle enumeration 5469
ComCtrls.TToolBar 4994
ComCtrls.TToolBar.Align 5022
ComCtrls.TToolBar.AlignControls 5009
ComCtrls.TToolBar.AllowTextButtons 5025
ComCtrls.TToolBar.Anchors 5026
ComCtrls.TToolBar.AutoSize 5026
ComCtrls.TToolBar.BorderWidth 5026
ComCtrls.TToolBar.ButtonCount 5027
ComCtrls.TToolBar.ButtonHeight 5028
ComCtrls.TToolBar.Buttons 5029
ComCtrls.TToolBar.ButtonWidth 5029
ComCtrls.TToolBar.CanAutoSize 5010
ComCtrls.TToolBar.CancelMenu 5010
ComCtrls.TToolBar.Canvas 5029
ComCtrls.TToolBar.Caption 5030
ComCtrls.TToolBar.ChangeScale 5010
ComCtrls.TToolBar.CheckMenuDropdown 5011
ComCtrls.TToolBar.ClickButton 5011
ComCtrls.TToolBar.Color 5033
ComCtrls.TToolBar.Constraints 5034
ComCtrls.TToolBar.Create 5006
ComCtrls.TToolBar.CreateParams 5012
ComCtrls.TToolBar.CreateWnd 5012
ComCtrls.TToolBar.Ctl3D 5035
ComCtrls.TToolBar.CustomDraw 5012
ComCtrls.TToolBar.CustomDrawButton 5013
ComCtrls.TToolBar.Customizable 5035
ComCtrls.TToolBar.CustomizeKeyName 5036
ComCtrls.TToolBar.CustomizeValueName 5036
ComCtrls.TToolBar.Destroy 5008
ComCtrls.TToolBar.DisabledImages 5037
ComCtrls.TToolBar.DockSite 5037
ComCtrls.TToolBar.DoQueryDelete 5013
ComCtrls.TToolBar.DoQueryInsert 5014
ComCtrls.TToolBar.DragCursor 5038
ComCtrls.TToolBar.DragKind 5039
ComCtrls.TToolBar.DragMode 5039
ComCtrls.TToolBar.DrawingStyle 5040
ComCtrls.TToolBar.EdgeBorders 5041

ComCtrls.TToolBar.EdgeInner 5041
ComCtrls.TToolBar.EdgeOuter 5041
ComCtrls.TToolBar.Enabled 5042
ComCtrls.TToolBar.FindButtonFromAccel 5014
ComCtrls.TToolBar.Flat 5042
ComCtrls.TToolBar.FlipChildren 5014
ComCtrls.TToolBar.Font 5043
ComCtrls.TToolBar.GetChildren 5015
ComCtrls.TToolBar.GetEnumerator 5015
ComCtrls.TToolBar.GradientDirection 5044
ComCtrls.TToolBar.GradientDrawButton 5016
ComCtrls.TToolBar.GradientDrawingOptions 5044
ComCtrls.TToolBar.GradientDrawToolBar 5016
ComCtrls.TToolBar.GradientEndColor 5044
ComCtrls.TToolBar.GradientStartColor 5044
ComCtrls.TToolBar.Height 5045
ComCtrls.TToolBar.HideClippedButtons 5046
ComCtrls.TToolBar.HotImages 5046
ComCtrls.TToolBar.HotTrackColor 5047
ComCtrls.TToolBar.Images 5047
ComCtrls.TToolBar.Indent 5047
ComCtrls.TToolBar.InitMenu 5016
ComCtrls.TToolBar.IsCustomDrawn 5016
ComCtrls.TToolBar.List 5048
ComCtrls.TToolBar.Loaded 5017
ComCtrls.TToolBar.Menu 5048
ComCtrls.TToolBar.Notification 5017
ComCtrls.TToolBar.OnAdvancedCustomDraw 5062
ComCtrls.TToolBar.OnAdvancedCustomDrawButton 5063
ComCtrls.TToolBar.OnClick 5064
ComCtrls.TToolBar.OnContextPopup 5066
ComCtrls.TToolBar.OnCustomDraw 5066
ComCtrls.TToolBar.OnCustomDrawButton 5067
ComCtrls.TToolBar.OnCustomizeAdded 5068
ComCtrls.TToolBar.OnCustomizeCanDelete 5068
ComCtrls.TToolBar.OnCustomizeCanInsert 5069
ComCtrls.TToolBar.OnCustomized 5069
ComCtrls.TToolBar.OnCustomizeDelete 5070
ComCtrls.TToolBar.OnCustomizeNewButton 5070
ComCtrls.TToolBar.OnCustomizeReset 5071
ComCtrls.TToolBar.OnCustomizing 5072
ComCtrls.TToolBar.OnDblClick 5072
ComCtrls.TToolBar.OnDockDrop 5073
ComCtrls.TToolBar.OnDockOver 5075
ComCtrls.TToolBar.OnDragDrop 5077
ComCtrls.TToolBar.OnDragOver 5079
ComCtrls.TToolBar.OnEndDock 5081
ComCtrls.TToolBar.OnEndDrag 5082
ComCtrls.TToolBar.OnEnter 5083
ComCtrls.TToolBar.OnExit 5085
ComCtrls.TToolBar.OnGetSiteInfo 5087
ComCtrls.TToolBar.OnMouseActivate 5089
ComCtrls.TToolBar.OnMouseDown 5090
ComCtrls.TToolBar.OnMouseEnter 5092
ComCtrls.TToolBar.OnMouseLeave 5092
ComCtrls.TToolBar.OnMouseMove 5093
ComCtrls.TToolBar.OnMouseUp 5095
ComCtrls.TToolBar.OnResize 5097
ComCtrls.TToolBar.OnStartDock 5098
ComCtrls.TToolBar.OnStartDrag 5098
ComCtrls.TToolBar.OnUnDock 5099
ComCtrls.TToolBar.ParentColor 5048
ComCtrls.TToolBar.ParentFont 5049
ComCtrls.TToolBar.ParentShowHint 5049
ComCtrls.TToolBar.PopupMenu 5050
ComCtrls.TToolBar.RepositionButton 5018
ComCtrls.TToolBar.RepositionButtons 5018
ComCtrls.TToolBar.Resize 5019
ComCtrls.TToolBar.RowCount 5054
ComCtrls.TToolBar.SetAutoSize 5019
ComCtrls.TToolBar.ShowCaptions 5055
ComCtrls.TToolBar.ShowHint 5057
ComCtrls.TToolBar.TabOrder 5058
ComCtrls.TToolBar.TabStop 5059
ComCtrls.TToolBar.TrackMenu 5019
ComCtrls.TToolBar.Transparent 5059
ComCtrls.TToolBar.Visible 5060
ComCtrls.TToolBar.WndProc 5020

ComCtrls.TToolBar.Wrapable 5061
ComCtrls.TToolBar.WrapButtons 5020
ComCtrls.TToolBarDockObject 5102
ComCtrls.TToolBarDockObject.AdjustDockRect 5104
ComCtrls.TToolBarDockObject.DrawDragDockImage 5104
ComCtrls.TToolBarDockObject.EraseDragDockImage 5104
ComCtrls.TToolBarDockObject.GetDragCursor 5105
ComCtrls.TToolBarDockObject.ToolDockImage 5105
ComCtrls.TToolBarEnumerator 5105
ComCtrls.TToolBarEnumerator.Create 5106
ComCtrls.TToolBarEnumerator.Current 5107
ComCtrls.TToolBarEnumerator.GetCurrent 5106
ComCtrls.TToolBarEnumerator.MoveNext 5106
ComCtrls.TToolButton 5107
ComCtrls.TToolButton.Action 5127
ComCtrls.TToolButton.ActionChange 5118
ComCtrls.TToolButton.AllowAllUp 5127
ComCtrls.TToolButton.AssignTo 5119
ComCtrls.TToolButton.AutoSize 5128
ComCtrls.TToolButton.BeginUpdate 5119
ComCtrls.TToolButton.Caption 5128
ComCtrls.TToolButton.CheckMenuDropdown 5120
ComCtrls.TToolButton.Click 5120
ComCtrls.TToolButton.Create 5116
ComCtrls.TToolButton.Down 5128
ComCtrls.TToolButton.DragCursor 5129
ComCtrls.TToolButton.DragKind 5129
ComCtrls.TToolButton.DragMode 5130
ComCtrls.TToolButton.DropdownMenu 5131
ComCtrls.TToolButton.Enabled 5132
ComCtrls.TToolButton.EnableDropdown 5132
ComCtrls.TToolButton.EndUpdate 5120
ComCtrls.TToolButton.FToolBar 5117
ComCtrls.TToolButton.GetActionLinkClass 5121
ComCtrls.TToolButton.Grouped 5132
ComCtrls.TToolButton.Height 5133
ComCtrls.TToolButton.ImageIndex 5135
ComCtrls.TToolButton.Indeterminate 5135
ComCtrls.TToolButton.Index 5135
ComCtrls.TToolButton.Marked 5136
ComCtrls.TToolButton.MenuItem 5136
ComCtrls.TToolButton.MouseDown 5121
ComCtrls.TToolButton.MouseMove 5122
ComCtrls.TToolButton.MouseUp 5122
ComCtrls.TToolButton.Notification 5123
ComCtrls.TToolButton.OnClick 5148
ComCtrls.TToolButton.OnContextPopup 5150
ComCtrls.TToolButton.OnDragDrop 5151
ComCtrls.TToolButton.OnDragOver 5153
ComCtrls.TToolButton.OnEndDock 5155
ComCtrls.TToolButton.OnEndDrag 5156
ComCtrls.TToolButton.OnMouseActivate 5157
ComCtrls.TToolButton.OnMouseDown 5158
ComCtrls.TToolButton.OnMouseEnter 5160
ComCtrls.TToolButton.OnMouseLeave 5160
ComCtrls.TToolButton.OnMouseMove 5161
ComCtrls.TToolButton.OnMouseUp 5163
ComCtrls.TToolButton.OnStartDock 5165
ComCtrls.TToolButton.OnStartDrag 5166
ComCtrls.TToolButton.Paint 5123
ComCtrls.TToolButton.ParentShowHint 5137
ComCtrls.TToolButton.PopupMenu 5137
ComCtrls.TToolButton.RefreshControl 5124
ComCtrls.TToolButton.SetAutoSize 5124
ComCtrls.TToolButton.SetBounds 5124
ComCtrls.TToolButton.SetToolBar 5125
ComCtrls.TToolButton.ShowHint 5142
ComCtrls.TToolButton.Style 5144
ComCtrls.TToolButton.UpdateControl 5125
ComCtrls.TToolButton.ValidateContainer 5126
ComCtrls.TToolButton.Visible 5145
ComCtrls.TToolButton.Width 5147
ComCtrls.TToolButton.Wrap 5147
ComCtrls.TToolButtonActionLink 5166
ComCtrls.TToolButtonActionLink.AssignClient 5170
ComCtrls.TToolButtonActionLink.FClient 5169
ComCtrls.TToolButtonActionLink.IsCheckedLinked 5170
ComCtrls.TToolButtonActionLink.IsDropdownMenuLinked 5170

ComCtrls.TToolButtonActionLink.IsEnableDropdownLinked 5171
ComCtrls.TToolButtonActionLink.IsImageIndexLinked 5171
ComCtrls.TToolButtonActionLink.SetChecked 5171
ComCtrls.TToolButtonActionLink.SetDropdownMenu 5172
ComCtrls.TToolButtonActionLink.SetEnableDropdown 5172
ComCtrls.TToolButtonActionLink.SetImageIndex 5172
ComCtrls.TToolButtonActionLinkClass 5511
ComCtrls.TToolButtonActionLinkClass type 5511
ComCtrls.TToolButtonState 5470
ComCtrls.TToolButtonState enumeration 5470
ComCtrls.TToolButtonStyle 5470
ComCtrls.TToolButtonStyle enumeration 5470
ComCtrls.TTrackBar 5172
ComCtrls.TTrackBar.Align 5188
ComCtrls.TTrackBar.Anchors 5191
ComCtrls.TTrackBar.BorderWidth 5192
ComCtrls.TTrackBar.Changed 5185
ComCtrls.TTrackBar.Constraints 5192
ComCtrls.TTrackBar.Create 5184
ComCtrls.TTrackBar.CreateParams 5185
ComCtrls.TTrackBar.CreateWnd 5186
ComCtrls.TTrackBar.Ctl3D 5193
ComCtrls.TTrackBar.DestroyWnd 5186
ComCtrls.TTrackBar.DragCursor 5193
ComCtrls.TTrackBar.DragKind 5194
ComCtrls.TTrackBar.DragMode 5194
ComCtrls.TTrackBar.Enabled 5196
ComCtrls.TTrackBar.Frequency 5196
ComCtrls.TTrackBar.LineSize 5196
ComCtrls.TTrackBar.Max 5197
ComCtrls.TTrackBar.Min 5197
ComCtrls.TTrackBar.OnChange 5212
ComCtrls.TTrackBar.OnContextPopup 5212
ComCtrls.TTrackBar.OnDragDrop 5213
ComCtrls.TTrackBar.OnDragOver 5215
ComCtrls.TTrackBar.OnEndDock 5217
ComCtrls.TTrackBar.OnEndDrag 5218
ComCtrls.TTrackBar.OnEnter 5219
ComCtrls.TTrackBar.OnExit 5221
ComCtrls.TTrackBar.OnKeyDown 5223
ComCtrls.TTrackBar.OnKeyPress 5226
ComCtrls.TTrackBar.OnKeyUp 5227
ComCtrls.TTrackBar.OnStartDock 5228
ComCtrls.TTrackBar.OnStartDrag 5229
ComCtrls.TTrackBar.Orientation 5198
ComCtrls.TTrackBar.PageSize 5198
ComCtrls.TTrackBar.ParentCtl3D 5198
ComCtrls.TTrackBar.ParentShowHint 5199
ComCtrls.TTrackBar.PopupMenu 5199
ComCtrls.TTrackBar.Position 5204
ComCtrls.TTrackBar.PositionToolTip 5204
ComCtrls.TTrackBar.SelEnd 5205
ComCtrls.TTrackBar.SelStart 5205
ComCtrls.TTrackBar.SetTick 5187
ComCtrls.TTrackBar.ShowHint 5206
ComCtrls.TTrackBar.ShowSelRange 5207
ComCtrls.TTrackBar.SliderVisible 5207
ComCtrls.TTrackBar.TabOrder 5208
ComCtrls.TTrackBar.TabStop 5208
ComCtrls.TTrackBar.ThumbLength 5209
ComCtrls.TTrackBar.TickMarks 5209
ComCtrls.TTrackBar.TickStyle 5209
ComCtrls.TTrackBar.Visible 5210
ComCtrls.TTrackBarOrientation 5471
ComCtrls.TTrackBarOrientation enumeration 5471
ComCtrls.TTreeNode 5230
ComCtrls.TTreeNode.AbsoluteIndex 5252
ComCtrls.TTreeNode.AlphaSort 5234
ComCtrls.TTreeNode.Assign 5234
ComCtrls.TTreeNode.Collapse 5235
ComCtrls.TTreeNode.Count 5253
ComCtrls.TTreeNode.Create 5232
ComCtrls.TTreeNode.CustomSort 5236
ComCtrls.TTreeNode.Cut 5254
ComCtrls.TTreeNode.Data 5254
ComCtrls.TTreeNode.Delete 5237
ComCtrls.TTreeNode.DeleteChildren 5238
ComCtrls.TTreeNode.Deleting 5256

ComCtrls.TTreeNode.Destroy 5233
ComCtrls.TTreeNode.DisplayRect 5238
ComCtrls.TTreeNode.DropTarget 5257
ComCtrls.TTreeNode.EditText 5240
ComCtrls.TTreeNode.EndEdit 5241
ComCtrls.TTreeNode.Expand 5242
ComCtrls.TTreeNode.Expanded 5257
ComCtrls.TTreeNode.Focused 5257
ComCtrls.TTreeNode.getFirstChild 5242
ComCtrls.TTreeNode.GetHandle 5243
ComCtrls.TTreeNode.GetLastChild 5243
ComCtrls.TTreeNode.GetNext 5243
ComCtrls.TTreeNode.GetNextChild 5244
ComCtrls.TTreeNode.getNextSibling 5245
ComCtrls.TTreeNode.GetNextVisible 5245
ComCtrls.TTreeNode.GetPrev 5246
ComCtrls.TTreeNode.GetPrevChild 5246
ComCtrls.TTreeNode.getPrevSibling 5247
ComCtrls.TTreeNode.GetPrevVisible 5247
ComCtrls.TTreeNode.GetState 5248
ComCtrls.TTreeNode.Handle 5258
ComCtrls.TTreeNode.HasAsParent 5248
ComCtrls.TTreeNode.HasChildren 5258
ComCtrls.TTreeNode.ImageIndex 5259
ComCtrls.TTreeNode.Index 5259
ComCtrls.TTreeNode.IndexOf 5248
ComCtrls.TTreeNode.IsFirstNode 5249
ComCtrls.TTreeNode.IsVisible 5259
ComCtrls.TTreeNode.Item 5260
ComCtrls.TTreeNode.ItemId 5260
ComCtrls.TTreeNode.Level 5260
ComCtrls.TTreeNode.MakeVisible 5249
ComCtrls.TTreeNode.MoveTo 5249
ComCtrls.TTreeNode.OverlayIndex 5263
ComCtrls.TTreeNode.Owner 5263
ComCtrls.TTreeNode.Parent 5263
ComCtrls.TTreeNode.Selected 5264
ComCtrls.TTreeNode.SelectedIndex 5265
ComCtrls.TTreeNode.SetSelectedBit 5251
ComCtrls.TTreeNode.SetState 5251
ComCtrls.TTreeNode.StateIndex 5265
ComCtrls.TTreeNode.Text 5265
ComCtrls.TTreeNode.TreeView 5266
ComCtrls.TTreeNodeClass 5512
ComCtrls.TTreeNodeClass type 5512
ComCtrls.TTreeNodes 5266
ComCtrls.TTreeNodes.Add 5270
ComCtrls.TTreeNodes.AddChild 5271
ComCtrls.TTreeNodes.AddChildFirst 5273
ComCtrls.TTreeNodes.AddChildObject 5273
ComCtrls.TTreeNodes.AddChildObjectFirst 5274
ComCtrls.TTreeNodes.AddFirst 5275
ComCtrls.TTreeNodes.AddItem 5275
ComCtrls.TTreeNodes.AddNode 5276
ComCtrls.TTreeNodes.AddObject 5276
ComCtrls.TTreeNodes.AddObjectFirst 5279
ComCtrls.TTreeNodes.AlphaSort 5279
ComCtrls.TTreeNodes.Assign 5279
ComCtrls.TTreeNodes.BeginUpdate 5280
ComCtrls.TTreeNodes.Clear 5280
ComCtrls.TTreeNodes.Count 5287
ComCtrls.TTreeNodes.Create 5268
ComCtrls.TTreeNodes.CreateItem 5281
ComCtrls.TTreeNodes.CustomSort 5281
ComCtrls.TTreeNodes.DefineProperties 5281
ComCtrls.TTreeNodes.Delete 5282
ComCtrls.TTreeNodes.Destroy 5269
ComCtrls.TTreeNodes.EndUpdate 5283
ComCtrls.TTreeNodes.GetCount 5283
ComCtrls.TTreeNodes.GetEnumerator 5283
ComCtrls.TTreeNodes.GetFirstNode 5284
ComCtrls.TTreeNodes.GetNode 5285
ComCtrls.TTreeNodes.Handle 5289
ComCtrls.TTreeNodes.Insert 5285
ComCtrls.TTreeNodes.InsertNode 5285
ComCtrls.TTreeNodes.InsertObject 5286
ComCtrls.TTreeNodes.Item 5289
ComCtrls.TTreeNodes.Owner 5290

ComCtrls.TTreeNodes.Reading 5290
ComCtrls.TTreeNodes.SetItem 5286
ComCtrls.TTreeNodes.SetUpdateState 5286
ComCtrls.TTreeNodesEnumerator 5291
ComCtrls.TTreeNodesEnumerator.Create 5291
ComCtrls.TTreeNodesEnumerator.Current 5292
ComCtrls.TTreeNodesEnumerator.GetCurrent 5292
ComCtrls.TTreeNodesEnumerator.MoveNext 5292
ComCtrls.TTreeView 5293
ComCtrls.TTreeView.Align 5307
ComCtrls.TTreeView.Anchors 5310
ComCtrls.TTreeView.AutoExpand 5311
ComCtrls.TTreeView.BevelEdges 5311
ComCtrls.TTreeView.BevelInner 5312
ComCtrls.TTreeView.BevelKind 5312
ComCtrls.TTreeView.BevelOuter 5312
ComCtrls.TTreeView.BevelWidth 5313
ComCtrls.TTreeView.BiDiMode 5313
ComCtrls.TTreeView.BorderStyle 5314
ComCtrls.TTreeView.BorderWidth 5314
ComCtrls.TTreeView.ChangeDelay 5315
ComCtrls.TTreeView.Color 5315
ComCtrls.TTreeView.Constraints 5316
ComCtrls.TTreeView.Ctl3D 5316
ComCtrls.TTreeView.DragCursor 5317
ComCtrls.TTreeView.DragKind 5317
ComCtrls.TTreeView.DragMode 5318
ComCtrls.TTreeView.Enabled 5319
ComCtrls.TTreeView.Font 5320
ComCtrls.TTreeView.HideSelection 5321
ComCtrls.TTreeView.HotTrack 5321
ComCtrls.TTreeView.Images 5321
ComCtrls.TTreeView.Indent 5322
ComCtrls.TTreeView.Items 5324
ComCtrls.TTreeView.MultiSelect 5325
ComCtrls.TTreeView.MultiSelectStyle 5326
ComCtrls.TTreeView.OnAddition 5342
ComCtrls.TTreeView.OnAdvancedCustomDraw 5342
ComCtrls.TTreeView.OnAdvancedCustomDrawItem 5342
ComCtrls.TTreeView.OnChange 5343
ComCtrls.TTreeView.OnChanging 5343
ComCtrls.TTreeView.OnClick 5344
ComCtrls.TTreeView.OnCollapsed 5345
ComCtrls.TTreeView.OnCollapsing 5346
ComCtrls.TTreeView.OnCompare 5346
ComCtrls.TTreeView.OnContextPopup 5347
ComCtrls.TTreeView.OnCreateNodeClass 5347
ComCtrls.TTreeView.OnCustomDraw 5348
ComCtrls.TTreeView.OnCustomDrawItem 5349
ComCtrls.TTreeView.OnDblClick 5352
ComCtrls.TTreeView.OnDeletion 5352
ComCtrls.TTreeView.OnDragDrop 5352
ComCtrls.TTreeView.OnDragOver 5355
ComCtrls.TTreeView.OnEdited 5357
ComCtrls.TTreeView.OnEditing 5357
ComCtrls.TTreeView.OnEndDock 5358
ComCtrls.TTreeView.OnEndDrag 5358
ComCtrls.TTreeView.OnEnter 5359
ComCtrls.TTreeView.OnExit 5362
ComCtrls.TTreeView.OnExpanded 5363
ComCtrls.TTreeView.OnExpanding 5364
ComCtrls.TTreeView.OnGetImageIndex 5364
ComCtrls.TTreeView.OnGetSelectedIndex 5365
ComCtrls.TTreeView.OnKeyDown 5365
ComCtrls.TTreeView.OnKeyPress 5368
ComCtrls.TTreeView.OnKeyUp 5369
ComCtrls.TTreeView.OnMouseActivate 5371
ComCtrls.TTreeView.OnMouseDown 5372
ComCtrls.TTreeView.OnMouseEnter 5374
ComCtrls.TTreeView.OnMouseLeave 5374
ComCtrls.TTreeView.OnMouseMove 5375
ComCtrls.TTreeView.OnMouseUp 5377
ComCtrls.TTreeView.OnStartDock 5379
ComCtrls.TTreeView.OnStartDrag 5380
ComCtrls.TTreeView.ParentBiDiMode 5326
ComCtrls.TTreeView.ParentColor 5327
ComCtrls.TTreeView.ParentCtl3D 5327
ComCtrls.TTreeView.ParentFont 5328

ComCtrls.TTreeView.ParentShowHint 5328
ComCtrls.TTreeView.PopupMenu 5329
ComCtrls.TTreeView.ReadOnly 5333
ComCtrls.TTreeView.RightClickSelect 5333
ComCtrls.TTreeView.RowSelect 5334
ComCtrls.TTreeView.ShowButtons 5334
ComCtrls.TTreeView.ShowHint 5335
ComCtrls.TTreeView.ShowLines 5336
ComCtrls.TTreeView.ShowRoot 5336
ComCtrls.TTreeView.SortType 5337
ComCtrls.TTreeView.StateImages 5337
ComCtrls.TTreeView.TabOrder 5338
ComCtrls.TTreeView.TabStop 5338
ComCtrls.TTreeView.ToolTips 5339
ComCtrls.TTreeView.Visible 5339
ComCtrls.TTVAdvancedCustomDrawEvent 5512
ComCtrls.TTVAdvancedCustomDrawEvent type 5512
ComCtrls.TTVAdvancedCustomDrawItemEvent 5513
ComCtrls.TTVAdvancedCustomDrawItemEvent type 5513
ComCtrls.TTVChangedEvent 5513
ComCtrls.TTVChangedEvent type 5513
ComCtrls.TTVChangingEvent 5514
ComCtrls.TTVChangingEvent type 5514
ComCtrls.TTVCollapsingEvent 5514
ComCtrls.TTVCollapsingEvent type 5514
ComCtrls.TTVCompareEvent 5515
ComCtrls.TTVCompareEvent type 5515
ComCtrls.TTVCreateNodeClassEvent 5516
ComCtrls.TTVCreateNodeClassEvent type 5516
ComCtrls.TTVCustomDrawEvent 5516
ComCtrls.TTVCustomDrawEvent type 5516
ComCtrls.TTVCustomDrawItemEvent 5516
ComCtrls.TTVCustomDrawItemEvent type 5516
ComCtrls.TTVEditedEvent 5517
ComCtrls.TTVEditedEvent type 5517
ComCtrls.TTVEditingEvent 5518
ComCtrls.TTVEditingEvent type 5518
ComCtrls.TTVExpandedEvent 5518
ComCtrls.TTVExpandedEvent type 5518
ComCtrls.TTVExpandingEvent 5519
ComCtrls.TTVExpandingEvent type 5519
ComCtrls.TUDAlignButton 5471
ComCtrls.TUDAlignButton enumeration 5471
ComCtrls.TUDBtnType 5472
ComCtrls.TUDBtnType enumeration 5472
ComCtrls.TUDChangingEvent 5519
ComCtrls.TUDChangingEvent type 5519
ComCtrls.TUDChangingEventEx 5520
ComCtrls.TUDChangingEventEx type 5520
ComCtrls.TUDClickEvent 5520
ComCtrls.TUDClickEvent type 5520
ComCtrls.TUDOrientation 5472
ComCtrls.TUDOrientation enumeration 5472
ComCtrls.TUpDown 5380
ComCtrls.TUpDown.AlignButton 5392
ComCtrls.TUpDown.Anchors 5393
ComCtrls.TUpDown.ArrowKeys 5393
ComCtrls.TUpDown.Associate 5394
ComCtrls.TUpDown.Constraints 5394
ComCtrls.TUpDown.Enabled 5395
ComCtrls.TUpDown.Hint 5395
ComCtrls.TUpDown.Increment 5397
ComCtrls.TUpDown.Max 5397
ComCtrls.TUpDown.Min 5398
ComCtrls.TUpDown.OnChanging 5409
ComCtrls.TUpDown.OnChangingEx 5410
ComCtrls.TUpDown.OnClick 5410
ComCtrls.TUpDown.OnContextPopup 5412
ComCtrls.TUpDown.OnEnter 5412
ComCtrls.TUpDown.OnExit 5415
ComCtrls.TUpDown.OnMouseActivate 5416
ComCtrls.TUpDown.OnMouseDown 5417
ComCtrls.TUpDown.OnMouseEnter 5420
ComCtrls.TUpDown.OnMouseLeave 5420
ComCtrls.TUpDown.OnMouseMove 5420
ComCtrls.TUpDown.OnMouseUp 5422
ComCtrls.TUpDown.Orientation 5398
ComCtrls.TUpDown.ParentShowHint 5399

ComCtrls.TUpDown.PopupMenu 5399
ComCtrls.TUpDown.Position 5404
ComCtrls.TUpDown.ShowHint 5404
ComCtrls.TUpDown.TabOrder 5406
ComCtrls.TUpDown.TabStop 5406
ComCtrls.TUpDown.Thousands 5407
ComCtrls.TUpDown.Visible 5407
ComCtrls.TUpDown.Wrap 5408
ComCtrls.TUpDownDirection 5473
ComCtrls.TUpDownDirection enumeration 5473
ComCtrls.TViewStyle 5473
ComCtrls.TViewStyle enumeration 5473
ComCtrls.TWidth 5520
ComCtrls.TWidth type 5520
ComCtrls.TWorkArea 5425
ComCtrls.TWorkArea.Color 5428
ComCtrls.TWorkArea.Create 5427
ComCtrls.TWorkArea.GetDisplayName 5427
ComCtrls.TWorkArea.Rect 5428
ComCtrls.TWorkArea.SetDisplayName 5427
ComCtrls.TWorkAreas 5429
ComCtrls.TWorkAreas.Add 5431
ComCtrls.TWorkAreas.Changed 5432
ComCtrls.TWorkAreas.Delete 5432
ComCtrls.TWorkAreas.Insert 5433
ComCtrls.TWorkAreas.Items 5434
ComCtrls.TWorkAreas.Update 5433
ComObj 5525
ComObj namespace 5525
- Classes 5528
- Functions 5590
- Interfaces 5588
- Structs, Records, Enums 5604
- Types 5606
- Variables 5610

ComObj.ClassIDToProgID 5590
ComObj.ClassIDToProgID function 5590
ComObj.CoAddRefServerProcess 5610
ComObj.CoAddRefServerProcess variable 5610
ComObj.CoCreateInstanceEx 5610
ComObj.CoCreateInstanceEx variable 5610
ComObj.CoInitFlags 5611
ComObj.CoInitFlags variable 5611
ComObj.CoInitializeEx 5611
ComObj.CoInitializeEx variable 5611
ComObj.ComClassManager 5591
ComObj.ComClassManager function 5591
ComObj.CoReleaseServerProcess 5612
ComObj.CoReleaseServerProcess variable 5612
ComObj.CoResumeClassObjects 5612
ComObj.CoResumeClassObjects variable 5612
ComObj.CoSuspendClassObjects 5612
ComObj.CoSuspendClassObjects variable 5612
ComObj.CreateClassID 5591
ComObj.CreateClassID function 5591
ComObj.CreateComObject 5592
ComObj.CreateComObject function 5592
ComObj.CreateOleObject 5592
ComObj.CreateOleObject function 5592
ComObj.CreateRegKey 5593
ComObj.CreateRegKey function 5593
ComObj.CreateRemoteComObject 5594
ComObj.CreateRemoteComObject function 5594
ComObj.DeleteRegKey 5595
ComObj.DeleteRegKey function 5595
ComObj.DispatchInvoke 5595
ComObj.DispatchInvoke function 5595
ComObj.DispatchInvokeError 5596
ComObj.DispatchInvokeError function 5596
ComObj.EOleError 5528
ComObj.EOleException 5529
ComObj.EOleException.Create 5530
ComObj.EOleException.HelpFile 5531
ComObj.EOleException.Source 5531
ComObj.EOleRegistrationError 5531
ComObj.EOleSysError 5532
ComObj.EOleSysError.Create 5533
ComObj.EOleSysError.ErrorCode 5534

ComObj.GetActiveOleObject 5596
ComObj.GetActiveOleObject function 5596
ComObj.GetDispatchPropValue 5596
ComObj.GetDispatchPropValue function 5596
ComObj.GetRegStringValue 5597
ComObj.GetRegStringValue function 5597
ComObj.GUIDToString 5597
ComObj.GUIDToString function 5597
ComObj.HandleSafeCallException 5598
ComObj.HandleSafeCallException function 5598
ComObj.InterfaceConnect 5598
ComObj.InterfaceConnect function 5598
ComObj.InterfaceDisconnect 5599
ComObj.InterfaceDisconnect function 5599
ComObj.IServerExceptionHandler 5588
ComObj.IServerExceptionHandler.OnException 5589
ComObj.OleCheck 5600
ComObj.OleCheck function 5600
ComObj.OleError 5600
ComObj.OleError function 5600
ComObj.ProgIDToClassID 5601
ComObj.ProgIDToClassID function 5601
ComObj.RegisterAsService 5601
ComObj.RegisterAsService function 5601
ComObj.RegisterComServer 5602
ComObj.RegisterComServer function 5602
ComObj.SetDispatchPropValue 5602
ComObj.SetDispatchPropValue function 5602
ComObj.StringToGUID 5603
ComObj.StringToGUID function 5603
ComObj.StringToLPOLESTR 5603
ComObj.StringToLPOLESTR function 5603
ComObj.TAutoClass 5606
ComObj.TAutoClass type 5606
ComObj.TAutoIntfObject 5534
ComObj.TAutoIntfObject.Create 5536
ComObj.TAutoIntfObject.DispIID 5539
ComObj.TAutoIntfObject.DispIntfEntry 5539
ComObj.TAutoIntfObject.DispTypeInfo 5539
ComObj.TAutoIntfObject.GetIDsOfNames 5537
ComObj.TAutoIntfObject.GetTypeInfo 5537
ComObj.TAutoIntfObject.GetTypeInfoCount 5537
ComObj.TAutoIntfObject.InterfaceSupportsErrorInfo 5538
ComObj.TAutoIntfObject.Invoke 5538
ComObj.TAutoIntfObject.SafeCallException 5538
ComObj.TAutoObject 5540
ComObj.TAutoObject.AutoFactory 5545
ComObj.TAutoObject.EventConnect 5542
ComObj.TAutoObject.EventSink 5545
ComObj.TAutoObject.EventSinkChanged 5543
ComObj.TAutoObject.GetIDsOfNames 5543
ComObj.TAutoObject.GetTypeInfo 5543
ComObj.TAutoObject.GetTypeInfoCount 5544
ComObj.TAutoObject.Initialize 5544
ComObj.TAutoObject.Invoke 5544
ComObj.TAutoObjectFactory 5545
ComObj.TAutoObjectFactory.Create 5548
ComObj.TAutoObjectFactory.DispIntfEntry 5549
ComObj.TAutoObjectFactory.DispTypeInfo 5549
ComObj.TAutoObjectFactory.EventIID 5549
ComObj.TAutoObjectFactory.EventTypeInfo 5550
ComObj.TAutoObjectFactory.GetIntfEntry 5548
ComObj.TClassInstancing 5604
ComObj.TClassInstancing enumeration 5604
ComObj.TCoAddRefServerProcessProc 5606
ComObj.TCoAddRefServerProcessProc type 5606
ComObj.TCoCreateInstanceExProc 5607
ComObj.TCoCreateInstanceExProc type 5607
ComObj.TCoInitializeExProc 5607
ComObj.TCoInitializeExProc type 5607
ComObj.TComClass 5607
ComObj.TComClass type 5607
ComObj.TComClassManager 5550
ComObj.TComClassManager.Create 5551
ComObj.TComClassManager.Destroy 5552
ComObj.TComClassManager.ForEachFactory 5552
ComObj.TComClassManager.GetFactoryFromClass 5553
ComObj.TComClassManager.GetFactoryFromClassID 5553

ComObj.TComObject 5553
ComObj.TComObject._AddRef 5558
ComObj.TComObject._Release 5558
ComObj.TComObject.Controller 5561
ComObj.TComObject.Create 5555
ComObj.TComObject.CreateAggregated 5556
ComObj.TComObject.CreateFromFactory 5556
ComObj.TComObject.Destroy 5557
ComObj.TComObject.Factory 5561
ComObj.TComObject.Initialize 5558
ComObj.TComObject.InterfaceSupportsErrorInfo 5559
ComObj.TComObject.ObjAddRef 5559
ComObj.TComObject.ObjQueryInterface 5559
ComObj.TComObject.ObjRelease 5559
ComObj.TComObject.QueryInterface 5560
ComObj.TComObject.RefCount 5562
ComObj.TComObject.SafeCallException 5560
ComObj.TComObject.ServerExceptionHandler 5562
ComObj.TComObjectFactory 5563
ComObj.TComObjectFactory._AddRef 5567
ComObj.TComObjectFactory._Release 5567
ComObj.TComObjectFactory.ClassID 5571
ComObj.TComObjectFactory.ClassName 5572
ComObj.TComObjectFactory.ComClass 5572
ComObj.TComObjectFactory.ComServer 5572
ComObj.TComObjectFactory.Create 5565
ComObj.TComObjectFactory.CreateComObject 5567
ComObj.TComObjectFactory.CreateInstance 5568
ComObj.TComObjectFactory.CreateInstanceLic 5568
ComObj.TComObjectFactory.Description 5572
ComObj.TComObjectFactory.Destroy 5566
ComObj.TComObjectFactory.ErrorIID 5573
ComObj.TComObjectFactory.GetLicenseString 5568
ComObj.TComObjectFactory.GetLicInfo 5568
ComObj.TComObjectFactory.GetProgID 5569
ComObj.TComObjectFactory.HasMachineLicense 5569
ComObj.TComObjectFactory.Instancing 5573
ComObj.TComObjectFactory.LicString 5573
ComObj.TComObjectFactory.LockServer 5569
ComObj.TComObjectFactory.ProgID 5574
ComObj.TComObjectFactory.QueryInterface 5569
ComObj.TComObjectFactory.RegisterClassObject 5569
ComObj.TComObjectFactory.RequestLicKey 5570
ComObj.TComObjectFactory.ShowErrors 5574
ComObj.TComObjectFactory.SupportsLicensing 5574
ComObj.TComObjectFactory.ThreadingModel 5575
ComObj.TComObjectFactory.UpdateRegistry 5570
ComObj.TComObjectFactory.ValidateUserLicense 5571
ComObj.TComServerObject 5575
ComObj.TComServerObject.CountFactory 5577
ComObj.TComServerObject.CountObject 5577
ComObj.TComServerObject.GetHelpFileName 5577
ComObj.TComServerObject.GetServerFileName 5578
ComObj.TComServerObject.GetServerKey 5578
ComObj.TComServerObject.GetServerName 5578
ComObj.TComServerObject.GetStartSuspended 5578
ComObj.TComServerObject.GetTypeLib 5579
ComObj.TComServerObject.HelpFileName 5579
ComObj.TComServerObject.ServerFileName 5580
ComObj.TComServerObject.ServerKey 5580
ComObj.TComServerObject.ServerName 5580
ComObj.TComServerObject.SetHelpFileName 5579
ComObj.TComServerObject.StartSuspended 5580
ComObj.TComServerObject.TypeLib 5581
ComObj.TConnectEvent 5608
ComObj.TConnectEvent type 5608
ComObj.TCoReleaseServerProcessProc 5608
ComObj.TCoReleaseServerProcessProc type 5608
ComObj.TCoResumeClassObjectsProc 5608
ComObj.TCoResumeClassObjectsProc type 5608
ComObj.TCoSuspendClassObjectsProc 5609
ComObj.TCoSuspendClassObjectsProc type 5609
ComObj.TFactoryProc 5609
ComObj.TFactoryProc type 5609
ComObj.TThreadingModel 5604
ComObj.TThreadingModel enumeration 5604
ComObj.TTypedComClass 5609
ComObj.TTypedComClass type 5609

ComObj.TTypedComObject 5581
ComObj.TTypedComObject.GetClassInfo 5583
ComObj.TTypedComObjectFactory 5583
ComObj.TTypedComObjectFactory.ClassInfo 5587
ComObj.TTypedComObjectFactory.Create 5585
ComObj.TTypedComObjectFactory.GetInterfaceTypeInfo 5586
ComObj.TTypedComObjectFactory.UpdateRegistry 5586
coMovePrevious enumeration member 1294
CompProd 5613
CompProd namespace 5613
Classes 5614
CompProd.TComponentsPageProducer 5614
CompProd.TComponentsPageProducer.DoTagEvent 5618
CompProd.TComponentsPageProducer.FindComponent 5619
CompProd.TComponentsPageProducer.FindComponentName 5619
CompProd.TComponentsPageProducer.GetComponentContent 5619
CompProd.TComponentsPageProducer.GetContentOptions 5619
CompProd.TComponentsPageProducer.HandleTag 5620
CompProd.TComponentsPageProducer.OnHTMLTag 5620
ComServ 5621
ComServ namespace 5621
Classes 5623
Functions 5634
Structs, Records, Enums 5638
Types 5639
Variables 5640
ComServ.ComServer 5640
ComServ.ComServer variable 5640
ComServ.DllCanUnloadNow 5634
ComServ.DllCanUnloadNow function 5634
ComServ.DllGetClassObject 5635
ComServ.DllGetClassObject function 5635
ComServ.DllRegisterServer 5636
ComServ.DllRegisterServer function 5636
ComServ.DllUnregisterServer 5636
ComServ.DllUnregisterServer function 5636
ComServ.LoadTypeLibrary 5637
ComServ.LoadTypeLibrary function 5637
ComServ.TComServer 5623
ComServ.TComServer.CountFactory 5626
ComServ.TComServer.CountObject 5627
ComServ.TComServer.Create 5625
ComServ.TComServer.Destroy 5626
ComServ.TComServer.GetHelpFileName 5627
ComServ.TComServer.GetServerFileName 5627
ComServ.TComServer.GetServerKey 5627
ComServ.TComServer.GetServerName 5628
ComServ.TComServer.GetStartSuspended 5628
ComServ.TComServer.GetTypeLib 5628
ComServ.TComServer.Initialize 5628
ComServ.TComServer.IsInprocServer 5630
ComServ.TComServer.LoadTypeLib 5629
ComServ.TComServer.ObjectCount 5631
ComServ.TComServer.OnLastRelease 5632
ComServ.TComServer.SetHelpFileName 5629
ComServ.TComServer.SetServerName 5629
ComServ.TComServer.StartMode 5631
ComServ.TComServer.UIInteractive 5632
ComServ.TComServer.UpdateRegistry 5630
ComServ.TLastReleaseEvent 5639
ComServ.TLastReleaseEvent type 5639
ComServ.TStartMode 5638
ComServ.TStartMode enumeration 5638
ConColumns class 1894
about ConColumns class 1894
Birth 1896
ConColumns fields 1895
ConColumns Members 1895
FileName 1896
Host 1896
Id 1896
Ip 1896
Remarks 1897
RemotePort 1897
User 1897
ConnectionProperties class 1897

about ConnectionProperties class 1897
ClientDataDirectory 1901
CommitOnClose 1901
ConnectionProperties fields 1900
ConnectionProperties Members 1899
ConnectionTimeout 1901
CreateDb 1901
Database 1901
ExtendedMetadata 1902
Fileio 1902
HeuristicCompletion 1902
Host 1902
LockWaitTime 1902
LogFilters 1903
MaxPoolSize 1903
MaxRowLocks 1903
MinPoolSize 1903
Password 1903
PingServer 1904
Pooling 1904
Port 1904
PriorFileName 1904
Protocol 1904
ReadOnly 1905
ReadOnlyTx 1905
ReadOnlyTxDelay 1905
Remarks 1905
ShutdownServer 1905
TableLockTables 1906
User 1906
coNoExecuteScript enumeration member 24799
coNone enumeration member 565
coNoScript enumeration member 24799
coNotify enumeration member 1294
Contnrs 5641
Contnrs namespace 5641
Classes 5643
Functions 5705
Structs, Records, Enums 5706
Types 5709
Contnrs.RaiseListError 5705
Contnrs.RaiseListError function 5705
Contnrs.TBucket 5706
Contnrs.TBucket record 5706
Contnrs.TBucketArray 5709
Contnrs.TBucketArray type 5709
Contnrs.TBucketEvent 5709
Contnrs.TBucketEvent type 5709
Contnrs.TBucketItem 5707
Contnrs.TBucketItem record 5707
Contnrs.TBucketItemArray 5710
Contnrs.TBucketItemArray type 5710
Contnrs.TBucketList 5643
Contnrs.TBucketList.BucketFor 5645
Contnrs.TBucketList.Create 5645
Contnrs.TBucketListSizes 5707
Contnrs.TBucketListSizes enumeration 5707
Contnrs.TBucketProc 5710
Contnrs.TBucketProc type 5710
Contnrs.TClassList 5646
Contnrs.TClassList.Add 5648
Contnrs.TClassList.Extract 5649
Contnrs.TClassList.First 5649
Contnrs.TClassList.GetItems 5649
Contnrs.TClassList.IndexOf 5650
Contnrs.TClassList.Insert 5650
Contnrs.TClassList.Items 5652
Contnrs.TClassList.Last 5651
Contnrs.TClassList.Remove 5651
Contnrs.TClassList.SetItems 5652
Contnrs.TComponentList 5653
Contnrs.TComponentList.Add 5656
Contnrs.TComponentList.Destroy 5655
Contnrs.TComponentList.Extract 5656
Contnrs.TComponentList.First 5657
Contnrs.TComponentList.GetItems 5657
Contnrs.TComponentList.HandleFreeNotify 5657
Contnrs.TComponentList.IndexOf 5658

Contnrs.TComponentList.Insert 5658
Contnrs.TComponentList.Items 5660
Contnrs.TComponentList.Last 5659
Contnrs.TComponentList.Notify 5659
Contnrs.TComponentList.Remove 5659
Contnrs.TComponentList.SetItems 5660
Contnrs.TCustomBucketList 5661
Contnrs.TCustomBucketList.Add 5664
Contnrs.TCustomBucketList.AddItem 5664
Contnrs.TCustomBucketList.Assign 5665
Contnrs.TCustomBucketList.BucketCount 5668
Contnrs.TCustomBucketList.BucketFor 5665
Contnrs.TCustomBucketList.Buckets 5669
Contnrs.TCustomBucketList.Clear 5665
Contnrs.TCustomBucketList.Data 5669
Contnrs.TCustomBucketList.DeleteItem 5666
Contnrs.TCustomBucketList.Destroy 5663
Contnrs.TCustomBucketList.Exists 5666
Contnrs.TCustomBucketList.Find 5666
Contnrs.TCustomBucketList.FindItem 5667
Contnrs.TCustomBucketList.ForEach 5667
Contnrs.TCustomBucketList.Remove 5668
Contnrs.TIntegerBucketList 5670
Contnrs.TIntegerBucketList.Add 5672
Contnrs.TIntegerBucketList.Data 5673
Contnrs.TIntegerBucketList.GetData 5672
Contnrs.TIntegerBucketList.Remove 5672
Contnrs.TIntegerBucketList.SetData 5672
Contnrs.TObjectBucketList 5673
Contnrs.TObjectBucketList.Add 5675
Contnrs.TObjectBucketList.Data 5677
Contnrs.TObjectBucketList.GetData 5676
Contnrs.TObjectBucketList.Remove 5676
Contnrs.TObjectBucketList.SetData 5676
Contnrs.TObjectList 5677
Contnrs.TObjectList.Add 5680
Contnrs.TObjectList.Create 5680
Contnrs.TObjectList.Extract 5681
Contnrs.TObjectList.FindInstanceOf 5682
Contnrs.TObjectList.First 5682
Contnrs.TObjectList.GetItem 5682
Contnrs.TObjectList.IndexOf 5683
Contnrs.TObjectList.Insert 5683
Contnrs.TObjectList.Items 5685
Contnrs.TObjectList.Last 5684
Contnrs.TObjectList.Notify 5684
Contnrs.TObjectList.OwnsObjects 5686
Contnrs.TObjectList.Remove 5684
Contnrs.TObjectList.SetItem 5685
Contnrs.TObjectQueue 5687
Contnrs.TObjectQueue.Peek 5689
Contnrs.TObjectQueue.Pop 5689
Contnrs.TObjectQueue.Push 5689
Contnrs.TObjectStack 5690
Contnrs.TObjectStack.Peek 5692
Contnrs.TObjectStack.Pop 5692
Contnrs.TObjectStack.Push 5693
Contnrs.TOrderedList 5693
Contnrs.TOrderedList.AtLeast 5696
Contnrs.TOrderedList.Count 5696
Contnrs.TOrderedList.Create 5695
Contnrs.TOrderedList.Destroy 5695
Contnrs.TOrderedList.List 5699
Contnrs.TOrderedList.Peek 5696
Contnrs.TOrderedList.PeekItem 5697
Contnrs.TOrderedList.Pop 5697
Contnrs.TOrderedList.PopItem 5697
Contnrs.TOrderedList.Push 5698
Contnrs.TOrderedList.PushItem 5698
Contnrs.TQueue 5699
Contnrs.TQueue.PushItem 5701
Contnrs.TStack 5701
Contnrs.TStack.PushItem 5703
Controls 5711
Controls namespace 5711
Classes 5719
Constants 6323
Functions 6246

Interfaces 6238
Structs, Records, Enums 6267
Types 6290
Variables 6320
Controls.AnchorAlign 6325
Controls.AnchorAlign constant 6325
Controls.AnimateWindowProc 6320
Controls.AnimateWindowProc variable 6320
Controls.CancelDrag 6247
Controls.CancelDrag function 6247
Controls.ChangeBiDiModeAlignment 6247
Controls.ChangeBiDiModeAlignment function 6247
Controls.CM_ACTIONEXECUTE 6326
Controls.CM_ACTIONEXECUTE constant 6326
Controls.CM_ACTIONUPDATE 6327
Controls.CM_ACTIONUPDATE constant 6327
Controls.CM_ACTIVATE 6327
Controls.CM_ACTIVATE constant 6327
Controls.CM_ALLCHILDRENFLIPPED 6327
Controls.CM_ALLCHILDRENFLIPPED constant 6327
Controls.CM_APPKEYDOWN 6328
Controls.CM_APPKEYDOWN constant 6328
Controls.CM_APPSYSCOMMAND 6328
Controls.CM_APPSYSCOMMAND constant 6328
Controls.CM_BASE 6328
Controls.CM_BASE constant 6328
Controls.CM_BIDIMODECHANGED 6329
Controls.CM_BIDIMODECHANGED constant 6329
Controls.CM_BORDERCHANGED 6329
Controls.CM_BORDERCHANGED constant 6329
Controls.CM_BUTTONPRESSED 6329
Controls.CM_BUTTONPRESSED constant 6329
Controls.CM_CANCELMODE 6330
Controls.CM_CANCELMODE constant 6330
Controls.CM_CHANGED 6330
Controls.CM_CHANGED constant 6330
Controls.CM_CHILDKEY 6330
Controls.CM_CHILDKEY constant 6330
Controls.CM_COLORCHANGED 6331
Controls.CM_COLORCHANGED constant 6331
Controls.CM_CONTROLCHANGE 6331
Controls.CM_CONTROLCHANGE constant 6331
Controls.CM_CONTROLLISTCHANGE 6331
Controls.CM_CONTROLLISTCHANGE constant 6331
Controls.CM_CONTROLLISTCHANGING 6332
Controls.CM_CONTROLLISTCHANGING constant 6332
Controls.CM_CREATEPOPUP 6332
Controls.CM_CREATEPOPUP constant 6332
Controls.CM_CTL3DCHANGED 6332
Controls.CM_CTL3DCHANGED constant 6332
Controls.CM_CTLCOLOREDITGLASS 6333
Controls.CM_CTLCOLOREDITGLASS constant 6333
Controls.CM_CURSORCHANGED 6333
Controls.CM_CURSORCHANGED constant 6333
Controls.CM_DEACTIVATE 6333
Controls.CM_DEACTIVATE constant 6333
Controls.CM_DESIGNHITTEST 6334
Controls.CM_DESIGNHITTEST constant 6334
Controls.CM_DESTROYHANDLE 6334
Controls.CM_DESTROYHANDLE constant 6334
Controls.CM_DIALOGCHAR 6334
Controls.CM_DIALOGCHAR constant 6334
Controls.CM_DIALOGHANDLE 6335
Controls.CM_DIALOGHANDLE constant 6335
Controls.CM_DIALOGKEY 6335
Controls.CM_DIALOGKEY constant 6335
Controls.CM_DOCKCLIENT 6335
Controls.CM_DOCKCLIENT constant 6335
Controls.CM_DOCKNOTIFICATION 6336
Controls.CM_DOCKNOTIFICATION constant 6336
Controls.CM_DOCWINDOWACTIVATE 6336
Controls.CM_DOCWINDOWACTIVATE constant 6336
Controls.CM_DRAG 6336
Controls.CM_DRAG constant 6336
Controls.CM_ENABLEDCHANGED 6337
Controls.CM_ENABLEDCHANGED constant 6337
Controls.CM_ENTER 6337
Controls.CM_ENTER constant 6337

Controls.CM_EXIT 6337
Controls.CM_EXIT constant 6337
Controls.CM_FLOAT 6338
Controls.CM_FLOAT constant 6338
Controls.CM_FOCUSCHANGED 6338
Controls.CM_FOCUSCHANGED constant 6338
Controls.CM_FONTCHANGE 6338
Controls.CM_FONTCHANGE constant 6338
Controls.CM_FONTCHANGED 6339
Controls.CM_FONTCHANGED constant 6339
Controls.CM_GETDATALINK 6339
Controls.CM_GETDATALINK constant 6339
Controls.CM_GOTFOCUS 6339
Controls.CM_GOTFOCUS constant 6339
Controls.CM_HINTSHOW 6340
Controls.CM_HINTSHOW constant 6340
Controls.CM_HINTSHOWPAUSE 6340
Controls.CM_HINTSHOWPAUSE constant 6340
Controls.CM_HITTEST 6340
Controls.CM_HITTEST constant 6340
Controls.CM_ICONCHANGED 6341
Controls.CM_ICONCHANGED constant 6341
Controls.CM_INVALIDATE 6341
Controls.CM_INVALIDATE constant 6341
Controls.CM_INVALIDATEDOCKHOST 6341
Controls.CM_INVALIDATEDOCKHOST constant 6341
Controls.CM_INVOKEHELP 6342
Controls.CM_INVOKEHELP constant 6342
Controls.CM_ISSHORTCUT 6342
Controls.CM_ISSHORTCUT constant 6342
Controls.CM_ISTOOLCONTROL 6342
Controls.CM_ISTOOLCONTROL constant 6342
Controls.CM_LOSTFOCUS 6343
Controls.CM_LOSTFOCUS constant 6343
Controls.CM_MENUCHANGED 6343
Controls.CM_MENUCHANGED constant 6343
Controls.CM_MOUSEACTIVATE 6343
Controls.CM_MOUSEACTIVATE constant 6343
Controls.CM_MOUSEENTER 6344
Controls.CM_MOUSEENTER constant 6344
Controls.CM_MOUSELEAVE 6344
Controls.CM_MOUSELEAVE constant 6344
Controls.CM_MOUSEWHEEL 6344
Controls.CM_MOUSEWHEEL constant 6344
Controls.CM_PARENTBIDIMODECHANGED 6345
Controls.CM_PARENTBIDIMODECHANGED constant 6345
Controls.CM_PARENTCOLORCHANGED 6345
Controls.CM_PARENTCOLORCHANGED constant 6345
Controls.CM_PARENTCTL3DCHANGED 6345
Controls.CM_PARENTCTL3DCHANGED constant 6345
Controls.CM_PARENTFONTCHANGED 6346
Controls.CM_PARENTFONTCHANGED constant 6346
Controls.CM_PARENTSHOWHINTCHANGED 6346
Controls.CM_PARENTSHOWHINTCHANGED constant 6346
Controls.CM_POPUPHWNDDESTROY 6346
Controls.CM_POPUPHWNDDESTROY constant 6346
Controls.CM_RECREATEWND 6347
Controls.CM_RECREATEWND constant 6347
Controls.CM_RELEASE 6347
Controls.CM_RELEASE constant 6347
Controls.CM_SETACTIVECONTROL 6347
Controls.CM_SETACTIVECONTROL constant 6347
Controls.CM_SHOWHINTCHANGED 6348
Controls.CM_SHOWHINTCHANGED constant 6348
Controls.CM_SHOWINGCHANGED 6348
Controls.CM_SHOWINGCHANGED constant 6348
Controls.CM_SYSCOLORCHANGE 6348
Controls.CM_SYSCOLORCHANGE constant 6348
Controls.CM_SYSFONTCHANGED 6349
Controls.CM_SYSFONTCHANGED constant 6349
Controls.CM_TABSTOPCHANGED 6349
Controls.CM_TABSTOPCHANGED constant 6349
Controls.CM_TEXTCHANGED 6349
Controls.CM_TEXTCHANGED constant 6349
Controls.CM_TIMECHANGE 6350
Controls.CM_TIMECHANGE constant 6350
Controls.CM_UIACTIVATE 6350
Controls.CM_UIACTIVATE constant 6350

Controls.CM_UIDEACTIVATE 6350
Controls.CM_UIDEACTIVATE constant 6350
Controls.CM_UNDOCKCLIENT 6351
Controls.CM_UNDOCKCLIENT constant 6351
Controls.CM_VISIBLECHANGED 6351
Controls.CM_VISIBLECHANGED constant 6351
Controls.CM_WANTSPECIALKEY 6351
Controls.CM_WANTSPECIALKEY constant 6351
Controls.CM_WINDOWHOOK 6352
Controls.CM_WINDOWHOOK constant 6352
Controls.CM_WININICHANGE 6352
Controls.CM_WININICHANGE constant 6352
Controls.CN_BASE 6352
Controls.CN_BASE constant 6352
Controls.CN_CHAR 6353
Controls.CN_CHAR constant 6353
Controls.CN_CHARTOITEM 6353
Controls.CN_CHARTOITEM constant 6353
Controls.CN_COMMAND 6353
Controls.CN_COMMAND constant 6353
Controls.CN_COMPAREITEM 6354
Controls.CN_COMPAREITEM constant 6354
Controls.CN_CTLCOLORBTN 6354
Controls.CN_CTLCOLORBTN constant 6354
Controls.CN_CTLCOLORDLG 6354
Controls.CN_CTLCOLORDLG constant 6354
Controls.CN_CTLCOLOREDIT 6355
Controls.CN_CTLCOLOREDIT constant 6355
Controls.CN_CTLCOLORLISTBOX 6355
Controls.CN_CTLCOLORLISTBOX constant 6355
Controls.CN_CTLCOLORMSGBOX 6355
Controls.CN_CTLCOLORMSGBOX constant 6355
Controls.CN_CTLCOLORSCROLLBAR 6356
Controls.CN_CTLCOLORSCROLLBAR constant 6356
Controls.CN_CTLCOLORSTATIC 6356
Controls.CN_CTLCOLORSTATIC constant 6356
Controls.CN_DELETEITEM 6356
Controls.CN_DELETEITEM constant 6356
Controls.CN_DRAWITEM 6357
Controls.CN_DRAWITEM constant 6357
Controls.CN_HSCROLL 6357
Controls.CN_HSCROLL constant 6357
Controls.CN_KEYDOWN 6357
Controls.CN_KEYDOWN constant 6357
Controls.CN_KEYUP 6358
Controls.CN_KEYUP constant 6358
Controls.CN_MEASUREITEM 6358
Controls.CN_MEASUREITEM constant 6358
Controls.CN_NOTIFY 6358
Controls.CN_NOTIFY constant 6358
Controls.CN_PARENTNOTIFY 6359
Controls.CN_PARENTNOTIFY constant 6359
Controls.CN_SYSCHAR 6359
Controls.CN_SYSCHAR constant 6359
Controls.CN_SYSKEYDOWN 6359
Controls.CN_SYSKEYDOWN constant 6359
Controls.CN_VKEYTOITEM 6360
Controls.CN_VKEYTOITEM constant 6360
Controls.CN_VSCROLL 6360
Controls.CN_VSCROLL constant 6360
Controls.crAppStart 6360
Controls.crAppStart constant 6360
Controls.crArrow 6361
Controls.crArrow constant 6361
Controls.crCross 6361
Controls.crCross constant 6361
Controls.crDefault 6362
Controls.crDefault constant 6362
Controls.crDrag 6362
Controls.crDrag constant 6362
Controls.CreationControl 6320
Controls.CreationControl variable 6320
Controls.crHandPoint 6363
Controls.crHandPoint constant 6363
Controls.crHelp 6363
Controls.crHelp constant 6363
Controls.crHourGlass 6364
Controls.crHourGlass constant 6364

Controls.crHSplit 6364
Controls.crHSplit constant 6364
Controls.crIBeam 6365
Controls.crIBeam constant 6365
Controls.crMultiDrag 6365
Controls.crMultiDrag constant 6365
Controls.crNo 6366
Controls.crNo constant 6366
Controls.crNoDrop 6366
Controls.crNoDrop constant 6366
Controls.crNone 6367
Controls.crNone constant 6367
Controls.crSize 6367
Controls.crSize constant 6367
Controls.crSizeAll 6368
Controls.crSizeAll constant 6368
Controls.crSizeNESW 6368
Controls.crSizeNESW constant 6368
Controls.crSizeNS 6369
Controls.crSizeNS constant 6369
Controls.crSizeNWSE 6369
Controls.crSizeNWSE constant 6369
Controls.crSizeWE 6370
Controls.crSizeWE constant 6370
Controls.crSQLWait 6370
Controls.crSQLWait constant 6370
Controls.crUpArrow 6371
Controls.crUpArrow constant 6371
Controls.crVSplit 6371
Controls.crVSplit constant 6371
Controls.CTL3D_ALL 6372
Controls.CTL3D_ALL constant 6372
Controls.CursorToIdent 6247
Controls.CursorToIdent function 6247
Controls.CursorToString 6248
Controls.CursorToString function 6248
Controls.DefaultDockTreeClass 6321
Controls.DefaultDockTreeClass variable 6321
Controls.DragDone 6249
Controls.DragDone function 6249
Controls.FindControl 6249
Controls.FindControl function 6249
Controls.FindDragTarget 6249
Controls.FindDragTarget function 6249
Controls.FindVCLWindow 6250
Controls.FindVCLWindow function 6250
Controls.GetCaptureControl 6251
Controls.GetCaptureControl function 6251
Controls.GetCursorValues 6251
Controls.GetCursorValues function 6251
Controls.GetLongHint 6252
Controls.GetLongHint function 6252
Controls.GetShortHint 6253
Controls.GetShortHint function 6253
Controls.IdentToCursor 6254
Controls.IdentToCursor function 6254
Controls.IDockManager 6238
Controls.IDockManager.BeginUpdate 6239
Controls.IDockManager.EndUpdate 6240
Controls.IDockManager.GetControlBounds 6240
Controls.IDockManager.InsertControl 6240
Controls.IDockManager.LoadFromStream 6241
Controls.IDockManager.PaintSite 6241
Controls.IDockManager.PositionDockRect 6242
Controls.IDockManager.RemoveControl 6242
Controls.IDockManager.ResetBounds 6243
Controls.IDockManager.SaveToStream 6245
Controls.IDockManager.SetReplacingControl 6245
Controls.Imm32GetCompositionString 6254
Controls.Imm32GetCompositionString function 6254
Controls.Imm32GetContext 6255
Controls.Imm32GetContext function 6255
Controls.Imm32GetConversionStatus 6255
Controls.Imm32GetConversionStatus function 6255
Controls.Imm32IsIME 6255
Controls.Imm32IsIME function 6255
Controls.Imm32NotifyIME 6256
Controls.Imm32NotifyIME function 6256

Controls.Imm32ReleaseContext 6256
Controls.Imm32ReleaseContext function 6256
Controls.Imm32SetCompositionFont 6256
Controls.Imm32SetCompositionFont function 6256
Controls.Imm32SetCompositionWindow 6257
Controls.Imm32SetCompositionWindow function 6257
Controls.Imm32SetConversionStatus 6257
Controls.Imm32SetConversionStatus function 6257
Controls.Imm32SetOpenStatus 6257
Controls.Imm32SetOpenStatus function 6257
Controls.InitWndProc 6258
Controls.InitWndProc function 6258
Controls.IsAbortResult 6258
Controls.IsAbortResult function 6258
Controls.IsAnAllResult 6259
Controls.IsAnAllResult function 6259
Controls.IsDragObject 6259
Controls.IsDragObject function 6259
Controls.IsNegativeResult 6260
Controls.IsNegativeResult function 6260
Controls.IsPositiveResult 6260
Controls.IsPositiveResult function 6260
Controls.IsVCLControl 6261
Controls.IsVCLControl function 6261
Controls.IsVCLControlHook 6321
Controls.IsVCLControlHook variable 6321
Controls.Mouse 6321
Controls.Mouse variable 6321
Controls.MoveWindowOrg 6262
Controls.MoveWindowOrg function 6262
Controls.mrAbort 6372
Controls.mrAbort constant 6372
Controls.mrAll 6372
Controls.mrAll constant 6372
Controls.mrCancel 6373
Controls.mrCancel constant 6373
Controls.mrIgnore 6373
Controls.mrIgnore constant 6373
Controls.mrNo 6374
Controls.mrNo constant 6374
Controls.mrNone 6374
Controls.mrNone constant 6374
Controls.mrNoToAll 6375
Controls.mrNoToAll constant 6375
Controls.mrOk 6375
Controls.mrOk constant 6375
Controls.mrRetry 6375
Controls.mrRetry constant 6375
Controls.mrYes 6376
Controls.mrYes constant 6376
Controls.mrYesToAll 6376
Controls.mrYesToAll constant 6376
Controls.NewStyleControls 6322
Controls.NewStyleControls variable 6322
Controls.NullDockSite 6377
Controls.NullDockSite constant 6377
Controls.PControlListItem 6291
Controls.PControlListItem type 6291
Controls.PDockNotifyRec 6291
Controls.PDockNotifyRec type 6291
Controls.PDragRec 6292
Controls.PDragRec type 6292
Controls.PerformEraseBackground 6262
Controls.PerformEraseBackground function 6262
Controls.PMouseActivateRec 6292
Controls.PMouseActivateRec type 6292
Controls.PPopupFormInfo 6292
Controls.PPopupFormInfo type 6292
Controls.SendAppMessage 6262
Controls.SendAppMessage function 6262
Controls.SetCaptureControl 6263
Controls.SetCaptureControl function 6263
Controls.SetImeMode 6264
Controls.SetImeMode function 6264
Controls.SetImeName 6264
Controls.SetImeName function 6264
Controls.StringToCursor 6264
Controls.StringToCursor function 6264

Controls.StripAllFromResult 6265
Controls.StripAllFromResult function 6265
Controls.TAlign 6268
Controls.TAlign enumeration 6268
Controls.TAlignInfo 6269
Controls.TAlignInfo record 6269
Controls.TAlignInsertBeforeEvent 6293
Controls.TAlignInsertBeforeEvent type 6293
Controls.TAlignPositionEvent 6293
Controls.TAlignPositionEvent type 6293
Controls.TAlignSet 6294
Controls.TAlignSet type 6294
Controls.TAnchorKind 6269
Controls.TAnchorKind enumeration 6269
Controls.TAnchors 6294
Controls.TAnchors type 6294
Controls.TAnimateWindowProc 6295
Controls.TAnimateWindowProc type 6295
Controls.TBaseDragControlObject 5719
Controls.TBaseDragControlObject.Assign 5722
Controls.TBaseDragControlObject.Control 5724
Controls.TBaseDragControlObject.Create 5721
Controls.TBaseDragControlObject.EndDrag 5723
Controls.TBaseDragControlObject.Finished 5723
Controls.TBevelCut 6270
Controls.TBevelCut enumeration 6270
Controls.TBevelEdge 6270
Controls.TBevelEdge enumeration 6270
Controls.TBevelEdges 6295
Controls.TBevelEdges type 6295
Controls.TBevelKind 6271
Controls.TBevelKind enumeration 6271
Controls.TBevelWidth 6296
Controls.TBevelWidth type 6296
Controls.TBorderWidth 6296
Controls.TBorderWidth type 6296
Controls.TCanResizeEvent 6296
Controls.TCanResizeEvent type 6296
Controls.TCaption 6297
Controls.TCaption type 6297
Controls.TCMActivate 6297
Controls.TCMActivate type 6297
Controls.TCMCancelMode 6272
Controls.TCMCancelMode record 6272
Controls.TCMChanged 6272
Controls.TCMChanged record 6272
Controls.TCMChildKey 6273
Controls.TCMChildKey record 6273
Controls.TCMControlChange 6273
Controls.TCMControlChange record 6273
Controls.TCMControlListChange 6274
Controls.TCMControlListChange record 6274
Controls.TCMControlListChanging 6274
Controls.TCMControlListChanging record 6274
Controls.TCMCreatePopup 6275
Controls.TCMCreatePopup record 6275
Controls.TCMDeactivate 6297
Controls.TCMDeactivate type 6297
Controls.TCMDesignHitTest 6298
Controls.TCMDesignHitTest type 6298
Controls.TCMDialogChar 6298
Controls.TCMDialogChar type 6298
Controls.TCMDialogKey 6298
Controls.TCMDialogKey type 6298
Controls.TCMDockClient 6275
Controls.TCMDockClient record 6275
Controls.TCMDockNotification 6276
Controls.TCMDockNotification record 6276
Controls.TCMDrag 6276
Controls.TCMDrag record 6276
Controls.TCMEnter 6299
Controls.TCMEnter type 6299
Controls.TCMExit 6299
Controls.TCMExit type 6299
Controls.TCMFloat 6277
Controls.TCMFloat record 6277
Controls.TCMFocusChanged 6277
Controls.TCMFocusChanged record 6277

Controls.TCMGotFocus 6299
Controls.TCMGotFocus type 6299
Controls.TCMHitTest 6300
Controls.TCMHitTest type 6300
Controls.TCMLostFocus 6300
Controls.TCMLostFocus type 6300
Controls.TCMMouseActivate 6278
Controls.TCMMouseActivate record 6278
Controls.TCMMouseWheel 6278
Controls.TCMMouseWheel record 6278
Controls.TCMPopupHWndDestroy 6279
Controls.TCMPopupHWndDestroy record 6279
Controls.TCMUnDockClient 6279
Controls.TCMUnDockClient record 6279
Controls.TCMWantSpecialKey 6300
Controls.TCMWantSpecialKey type 6300
Controls.TConstrainedResizeEvent 6301
Controls.TConstrainedResizeEvent type 6301
Controls.TConstraintSize 6301
Controls.TConstraintSize type 6301
Controls.TContextPopupEvent 6301
Controls.TContextPopupEvent type 6301
Controls.TControl 5724
Controls.TControl.Action 5797
Controls.TControl.ActionChange 5737
Controls.TControl.ActionLink 5798
Controls.TControl.AdjustSize 5738
Controls.TControl.Align 5798
Controls.TControl.AlignWithMargins 5802
Controls.TControl.Anchors 5802
Controls.TControl.AssignTo 5739
Controls.TControl.AutoSize 5803
Controls.TControl.BeginAutoDrag 5739
Controls.TControl.BeginDrag 5740
Controls.TControl.BiDiMode 5803
Controls.TControl.BoundsRect 5803
Controls.TControl.BringToFront 5740
Controls.TControl.CalcCursorPos 5741
Controls.TControl.CanAutoSize 5741
Controls.TControl.CanResize 5742
Controls.TControl.Caption 5806
Controls.TControl.Changed 5742
Controls.TControl.ChangeScale 5743
Controls.TControl.Click 5743
Controls.TControl.ClientHeight 5808
Controls.TControl.ClientOrigin 5809
Controls.TControl.ClientRect 5810
Controls.TControl.ClientToParent 5744
Controls.TControl.ClientToScreen 5744
Controls.TControl.ClientWidth 5811
Controls.TControl.Color 5812
Controls.TControl.ConstrainedResize 5746
Controls.TControl.Constraints 5812
Controls.TControl.ControlState 5813
Controls.TControl.ControlStyle 5814
Controls.TControl.Create 5731
Controls.TControl.Cursor 5815
Controls.TControl.DblClick 5747
Controls.TControl.DefaultDockImage 5747
Controls.TControl.DefaultHandler 5748
Controls.TControl.DefineProperties 5748
Controls.TControl.DesignWndProc 5749
Controls.TControl.DesktopFont 5818
Controls.TControl.Destroy 5732
Controls.TControl.Dock 5749
Controls.TControl.DockOrientation 5818
Controls.TControl.DockTrackNoTarget 5749
Controls.TControl.DoContextPopup 5750
Controls.TControl.DoDock 5750
Controls.TControl.DoEndDock 5750
Controls.TControl.DoEndDrag 5751
Controls.TControl.DoMouseWheel 5751
Controls.TControl.DoMouseWheelDown 5752
Controls.TControl.DoMouseWheelUp 5752
Controls.TControl.DoStartDock 5753
Controls.TControl.DoStartDrag 5753
Controls.TControl.DragCanceled 5754
Controls.TControl.DragCursor 5818

Controls.TControl.DragDrop 5754
Controls.TControl.Dragging 5754
Controls.TControl.DragKind 5819
Controls.TControl.DragMode 5819
Controls.TControl.DragOver 5755
Controls.TControl.DrawDragDockImage 5755
Controls.TControl.DrawTextBiDiModeFlags 5756
Controls.TControl.DrawTextBiDiModeFlagsReadingOnly 5756
Controls.TControl.Enabled 5821
Controls.TControl.EndDrag 5757
Controls.TControl.EraseDragDockImage 5757
Controls.TControl.ExplicitHeight 5821
Controls.TControl.ExplicitLeft 5822
Controls.TControl.ExplicitTop 5822
Controls.TControl.ExplicitWidth 5822
Controls.TControl.FAnchorMove 5733
Controls.TControl.FAnchorOrigin 5734
Controls.TControl.FAnchorRules 5734
Controls.TControl.FExplicitHeight 5734
Controls.TControl.FExplicitLeft 5734
Controls.TControl.FExplicitTop 5734
Controls.TControl.FExplicitWidth 5735
Controls.TControl.Floating 5823
Controls.TControl.FloatingDockSiteClass 5823
Controls.TControl.Font 5823
Controls.TControl.FOriginalParentSize 5735
Controls.TControl.GetAction 5758
Controls.TControl.GetActionLinkClass 5758
Controls.TControl.GetClientOrigin 5758
Controls.TControl.GetClientRect 5759
Controls.TControl.GetControlsAlignment 5759
Controls.TControl.GetDeviceContext 5760
Controls.TControl.GetDockEdge 5763
Controls.TControl.GetDragImages 5763
Controls.TControl.GetEnabled 5763
Controls.TControl.GetFloating 5764
Controls.TControl.GetFloatingDockSiteClass 5764
Controls.TControl.GetPalette 5765
Controls.TControl.GetParentComponent 5765
Controls.TControl.GetPopupMenu 5765
Controls.TControl.GetTextBuf 5766
Controls.TControl.GetTextLen 5767
Controls.TControl.HasParent 5768
Controls.TControl.Height 5825
Controls.TControl.HelpContext 5826
Controls.TControl.HelpKeyword 5826
Controls.TControl.HelpType 5827
Controls.TControl.Hide 5769
Controls.TControl.Hint 5827
Controls.TControl.HostDockSite 5829
Controls.TControl.InitiateAction 5769
Controls.TControl.Invalidate 5770
Controls.TControl.IsControl 5829
Controls.TControl.IsRightToLeft 5770
Controls.TControl.Left 5830
Controls.TControl.Loaded 5771
Controls.TControl.LRDockWidth 5831
Controls.TControl.ManualDock 5771
Controls.TControl.ManualFloat 5772
Controls.TControl.Margins 5831
Controls.TControl.MouseActivate 5773
Controls.TControl.MouseCapture 5831
Controls.TControl.MouseDown 5773
Controls.TControl.MouseMove 5773
Controls.TControl.MouseUp 5774
Controls.TControl.MouseWheelHandler 5774
Controls.TControl.Notification 5775
Controls.TControl.OnCanResize 5851
Controls.TControl.OnClick 5851
Controls.TControl.OnConstrainedResize 5853
Controls.TControl.OnContextPopup 5854
Controls.TControl.OnDblClick 5854
Controls.TControl.OnDragDrop 5855
Controls.TControl.OnDragOver 5857
Controls.TControl.OnEndDock 5859
Controls.TControl.OnEndDrag 5860
Controls.TControl.OnMouseActivate 5861
Controls.TControl.OnMouseDown 5862

Controls.TControl.OnMouseEnter 5864
Controls.TControl.OnMouseLeave 5864
Controls.TControl.OnMouseMove 5865
Controls.TControl.OnMouseUp 5867
Controls.TControl.OnMouseWheel 5869
Controls.TControl.OnMouseWheelDown 5869
Controls.TControl.OnMouseWheelUp 5870
Controls.TControl.OnResize 5870
Controls.TControl.OnStartDock 5871
Controls.TControl.OnStartDrag 5871
Controls.TControl.PaletteChanged 5775
Controls.TControl.Parent 5832
Controls.TControl.ParentBiDiMode 5833
Controls.TControl.ParentColor 5834
Controls.TControl.ParentFont 5834
Controls.TControl.ParentShowHint 5835
Controls.TControl.ParentToClient 5776
Controls.TControl.Perform 5776
Controls.TControl.PopupMenu 5835
Controls.TControl.PositionDockRect 5777
Controls.TControl.ReadState 5778
Controls.TControl.Refresh 5779
Controls.TControl.Repaint 5779
Controls.TControl.ReplaceDockedControl 5779
Controls.TControl.RequestAlign 5780
Controls.TControl.Resize 5780
Controls.TControl.ScaleConstraints 5781
Controls.TControl.ScalingFlags 5840
Controls.TControl.ScreenToClient 5781
Controls.TControl.SendCancelMode 5781
Controls.TControl.SendDockNotification 5782
Controls.TControl.SendToBack 5782
Controls.TControl.SetAutoSize 5783
Controls.TControl.SetBiDiMode 5783
Controls.TControl.SetBounds 5783
Controls.TControl.SetDesignVisible 5784
Controls.TControl.SetDragMode 5785
Controls.TControl.SetEnabled 5785
Controls.TControl.SetName 5785
Controls.TControl.SetParent 5786
Controls.TControl.SetParentBiDiMode 5786
Controls.TControl.SetParentComponent 5787
Controls.TControl.SetTextBuf 5787
Controls.TControl.SetZOrder 5788
Controls.TControl.Show 5788
Controls.TControl.ShowHint 5840
Controls.TControl.TBDockHeight 5842
Controls.TControl.Text 5842
Controls.TControl.Top 5844
Controls.TControl.UndockHeight 5845
Controls.TControl.UndockWidth 5846
Controls.TControl.Update 5789
Controls.TControl.UpdateBoundsRect 5789
Controls.TControl.UpdateExplicitBounds 5789
Controls.TControl.UseRightToLeftAlignment 5790
Controls.TControl.UseRightToLeftReading 5790
Controls.TControl.UseRightToLeftScrollBar 5791
Controls.TControl.Visible 5847
Controls.TControl.VisibleChanging 5791
Controls.TControl.WheelAccumulator 5849
Controls.TControl.Width 5849
Controls.TControl.WindowProc 5872
Controls.TControl.WindowText 5850
Controls.TControl.WndProc 5791
Controls.TControlAction 5877
Controls.TControlAction.AutoCheck 5881
Controls.TControlAction.Caption 5882
Controls.TControlAction.Checked 5882
Controls.TControlAction.DropdownMenu 5883
Controls.TControlAction.Enabled 5884
Controls.TControlAction.EnableDropdown 5884
Controls.TControlAction.GroupIndex 5884
Controls.TControlAction.HelpContext 5885
Controls.TControlAction.HelpKeyword 5885
Controls.TControlAction.HelpType 5886
Controls.TControlAction.Hint 5886
Controls.TControlAction.ImageIndex 5886
Controls.TControlAction.OnExecute 5888

Controls.TControlAction.OnHint 5890
Controls.TControlAction.OnUpdate 5890
Controls.TControlAction.PopupMenu 5887
Controls.TControlAction.SecondaryShortCuts 5887
Controls.TControlAction.ShortCut 5887
Controls.TControlAction.Visible 5888
Controls.TControlActionLink 5892
Controls.TControlActionLink.AssignClient 5895
Controls.TControlActionLink.DoShowHint 5896
Controls.TControlActionLink.FClient 5894
Controls.TControlActionLink.IsCaptionLinked 5896
Controls.TControlActionLink.IsDropdownMenuLinked 5896
Controls.TControlActionLink.IsEnabledLinked 5897
Controls.TControlActionLink.IsEnableDropdownLinked 5897
Controls.TControlActionLink.IsHelpLinked 5897
Controls.TControlActionLink.IsHintLinked 5898
Controls.TControlActionLink.IsOnExecuteLinked 5898
Controls.TControlActionLink.IsPopupMenuLinked 5898
Controls.TControlActionLink.IsVisibleLinked 5899
Controls.TControlActionLink.SetCaption 5899
Controls.TControlActionLink.SetDropdownMenu 5899
Controls.TControlActionLink.SetEnabled 5900
Controls.TControlActionLink.SetEnableDropdown 5900
Controls.TControlActionLink.SetHelpContext 5900
Controls.TControlActionLink.SetHelpKeyword 5901
Controls.TControlActionLink.SetHelpType 5901
Controls.TControlActionLink.SetHint 5901
Controls.TControlActionLink.SetOnExecute 5902
Controls.TControlActionLink.SetPopupMenu 5902
Controls.TControlActionLink.SetVisible 5902
Controls.TControlActionLinkClass 6302
Controls.TControlActionLinkClass type 6302
Controls.TControlCanvas 5903
Controls.TControlCanvas.Control 5907
Controls.TControlCanvas.CreateHandle 5906
Controls.TControlCanvas.Destroy 5906
Controls.TControlCanvas.FreeHandle 5906
Controls.TControlCanvas.UpdateTextFlags 5907
Controls.TControlClass 6302
Controls.TControlClass type 6302
Controls.TControlListItem 6280
Controls.TControlListItem record 6280
Controls.TControlState 6302
Controls.TControlState type 6302
Controls.TControlStyle 6303
Controls.TControlStyle type 6303
Controls.TCreateParams 6280
Controls.TCreateParams record 6280
Controls.TCursor 6305
Controls.TCursor type 6305
Controls.TCustomControl 5908
Controls.TCustomControl.Canvas 5921
Controls.TCustomControl.Create 5918
Controls.TCustomControl.Destroy 5919
Controls.TCustomControl.Paint 5920
Controls.TCustomControl.PaintWindow 5920
Controls.TCustomControlAction 5923
Controls.TCustomControlAction.DropdownMenu 5927
Controls.TCustomControlAction.EnableDropdown 5928
Controls.TCustomControlAction.PopupMenu 5928
Controls.TCustomListControl 5928
Controls.TCustomListControl.AddItem 5940
Controls.TCustomListControl.Clear 5940
Controls.TCustomListControl.ClearSelection 5940
Controls.TCustomListControl.CopySelection 5941
Controls.TCustomListControl.DeleteSelected 5941
Controls.TCustomListControl.GetCount 5942
Controls.TCustomListControl.GetItemIndex 5942
Controls.TCustomListControl.ItemIndex 5944
Controls.TCustomListControl.MoveSelection 5942
Controls.TCustomListControl.SelectAll 5943
Controls.TCustomListControl.SetItemIndex 5943
Controls.TCustomMultiSelectListControl 5944
Controls.TCustomMultiSelectListControl.FMultiSelect 5956
Controls.TCustomMultiSelectListControl.GetSelCount 5956
Controls.TCustomMultiSelectListControl.MultiSelect 5957
Controls.TCustomMultiSelectListControl.SelCount 5957
Controls.TCustomMultiSelectListControl.SetMultiSelect 5956

Controls.TCustomPanningWindow 5958
Controls.TCustomPanningWindow.GetIsPanning 5969
Controls.TCustomPanningWindow.StartPanning 5969
Controls.TCustomPanningWindow.StopPanning 5969
Controls.TCustomTransparentControl 5970
Controls.TCustomTransparentControl.Create 5980
Controls.TCustomTransparentControl.CreateParams 5981
Controls.TCustomTransparentControl.InterceptMouse 5983
Controls.TCustomTransparentControl.Invalidate 5982
Controls.TCustomTransparentControl.InvalidateControlsUnderneath 5982
Controls.TCustomTransparentControl.WMNCHitTest 5982
Controls.TDate 6306
Controls.TDate type 6306
Controls.TDockDropEvent 6307
Controls.TDockDropEvent type 6307
Controls.TDockNotifyRec 6281
Controls.TDockNotifyRec record 6281
Controls.TDockOrientation 6282
Controls.TDockOrientation enumeration 6282
Controls.TDockOverEvent 6307
Controls.TDockOverEvent type 6307
Controls.TDockTree 5983
Controls.TDockTree.AdjustDockRect 5987
Controls.TDockTree.AdjustFrameRect 5988
Controls.TDockTree.BeginUpdate 5988
Controls.TDockTree.Create 5985
Controls.TDockTree.Destroy 5986
Controls.TDockTree.DockSite 5996
Controls.TDockTree.EndUpdate 5988
Controls.TDockTree.FindControlAtPos 5989
Controls.TDockTree.GetControlBounds 5989
Controls.TDockTree.HitTest 5990
Controls.TDockTree.InsertControl 5990
Controls.TDockTree.LoadFromStream 5991
Controls.TDockTree.MouseDown 5991
Controls.TDockTree.MouseMove 5991
Controls.TDockTree.MouseUp 5992
Controls.TDockTree.PaintDockFrame 5992
Controls.TDockTree.PaintSite 5992
Controls.TDockTree.PositionDockRect 5993
Controls.TDockTree.ReferenceFromOrient 5993
Controls.TDockTree.RelativeSizes 5997
Controls.TDockTree.RemoveControl 5993
Controls.TDockTree.ResetBounds 5994
Controls.TDockTree.SaveToStream 5994
Controls.TDockTree.SetReplacingControl 5995
Controls.TDockTree.ShowHint 5995
Controls.TDockTree.TopZone 5997
Controls.TDockTree.UpdateAll 5995
Controls.TDockTree.WndProc 5996
Controls.TDockTree.ZoneCaptionHitTest 5996
Controls.TDockTreeClass 6308
Controls.TDockTreeClass type 6308
Controls.TDockZone 5997
Controls.TDockZone.ChildControl 6001
Controls.TDockZone.ChildCount 6001
Controls.TDockZone.Create 5998
Controls.TDockZone.ExpandZoneLimit 5999
Controls.TDockZone.FirstVisibleChild 5999
Controls.TDockZone.Height 6001
Controls.TDockZone.Left 6001
Controls.TDockZone.LimitBegin 6002
Controls.TDockZone.LimitSize 6002
Controls.TDockZone.NextVisible 5999
Controls.TDockZone.PrevVisible 5999
Controls.TDockZone.ResetChildren 6000
Controls.TDockZone.ResetZoneLimits 6000
Controls.TDockZone.Top 6002
Controls.TDockZone.Update 6000
Controls.TDockZone.Visible 6002
Controls.TDockZone.VisibleChildCount 6002
Controls.TDockZone.Width 6003
Controls.TDockZone.ZoneLimit 6003
Controls.TDragControlObject 6003
Controls.TDragControlObject.GetDragCursor 6005
Controls.TDragControlObject.GetDragImages 6006
Controls.TDragControlObject.HideDragImage 6006

Controls.TDragControlObject.ShowDragImage 6006
Controls.TDragControlObjectEx 6007
Controls.TDragControlObjectEx.BeforeDestruction 6009
Controls.TDragDockObject 6009
Controls.TDragDockObject.AdjustDockRect 6012
Controls.TDragDockObject.Assign 6012
Controls.TDragDockObject.Brush 6015
Controls.TDragDockObject.Create 6011
Controls.TDragDockObject.Destroy 6012
Controls.TDragDockObject.DockRect 6015
Controls.TDragDockObject.DrawDragDockImage 6013
Controls.TDragDockObject.DropAlign 6017
Controls.TDragDockObject.DropOnControl 6017
Controls.TDragDockObject.EndDrag 6013
Controls.TDragDockObject.EraseDockRect 6017
Controls.TDragDockObject.EraseDragDockImage 6013
Controls.TDragDockObject.EraseWhenMoving 6018
Controls.TDragDockObject.Floating 6018
Controls.TDragDockObject.FrameWidth 6018
Controls.TDragDockObject.GetDragCursor 6014
Controls.TDragDockObject.GetEraseWhenMoving 6014
Controls.TDragDockObject.GetFrameWidth 6014
Controls.TDragDockObjectEx 6018
Controls.TDragDockObjectEx.BeforeDestruction 6021
Controls.TDragDropEvent 6308
Controls.TDragDropEvent type 6308
Controls.TDragImageList 6021
Controls.TDragImageList.BeginDrag 6026
Controls.TDragImageList.DragCursor 6030
Controls.TDragImageList.Dragging 6030
Controls.TDragImageList.DragHotspot 6030
Controls.TDragImageList.DragLock 6026
Controls.TDragImageList.DragMove 6027
Controls.TDragImageList.DragUnlock 6027
Controls.TDragImageList.EndDrag 6028
Controls.TDragImageList.GetHotSpot 6028
Controls.TDragImageList.HideDragImage 6028
Controls.TDragImageList.Initialize 6028
Controls.TDragImageList.SetDragImage 6029
Controls.TDragImageList.ShowDragImage 6029
Controls.TDragKind 6282
Controls.TDragKind enumeration 6282
Controls.TDragMessage 6283
Controls.TDragMessage enumeration 6283
Controls.TDragMode 6283
Controls.TDragMode enumeration 6283
Controls.TDragObject 6031
Controls.TDragObject.AfterConstruction 6033
Controls.TDragObject.AlwaysShowDragImages 6038
Controls.TDragObject.Assign 6034
Controls.TDragObject.BeforeDestruction 6034
Controls.TDragObject.Cancelling 6038
Controls.TDragObject.DragHandle 6039
Controls.TDragObject.DragPos 6039
Controls.TDragObject.DragTarget 6039
Controls.TDragObject.DragTargetPos 6040
Controls.TDragObject.Dropped 6040
Controls.TDragObject.Finished 6035
Controls.TDragObject.GetDragCursor 6035
Controls.TDragObject.GetDragImages 6035
Controls.TDragObject.GetName 6036
Controls.TDragObject.HideDragImage 6036
Controls.TDragObject.Instance 6037
Controls.TDragObject.MainWndProc 6037
Controls.TDragObject.MouseDeltaX 6040
Controls.TDragObject.MouseDeltaY 6041
Controls.TDragObject.RightClickCancels 6041
Controls.TDragObject.ShowDragImage 6037
Controls.TDragObject.WndProc 6038
Controls.TDragObjectClass 6308
Controls.TDragObjectClass type 6308
Controls.TDragObjectEx 6041
Controls.TDragObjectEx.BeforeDestruction 6043
Controls.TDragOverEvent 6309
Controls.TDragOverEvent type 6309
Controls.TDragRec 6284
Controls.TDragRec record 6284
Controls.TDragState 6284

Controls.TDragState enumeration 6284
Controls.TEndDragEvent 6309
Controls.TEndDragEvent type 6309
Controls.TForEachZoneProc 6309
Controls.TForEachZoneProc type 6309
Controls.TGetSiteInfoEvent 6310
Controls.TGetSiteInfoEvent type 6310
Controls.TGraphicControl 6044
Controls.TGraphicControl.Canvas 6053
Controls.TGraphicControl.Create 6051
Controls.TGraphicControl.Destroy 6052
Controls.TGraphicControl.Paint 6052
Controls.THintWindow 6055
Controls.THintWindow.ActivateHint 6067
Controls.THintWindow.ActivateHintData 6067
Controls.THintWindow.BiDiMode 6071
Controls.THintWindow.CalcHintRect 6068
Controls.THintWindow.Canvas 6072
Controls.THintWindow.Caption 6073
Controls.THintWindow.Color 6076
Controls.THintWindow.Create 6066
Controls.THintWindow.CreateParams 6069
Controls.THintWindow.Font 6077
Controls.THintWindow.IsHintMsg 6069
Controls.THintWindow.NCPaint 6069
Controls.THintWindow.Paint 6070
Controls.THintWindow.ReleaseHandle 6070
Controls.THintWindow.ShouldHideHint 6070
Controls.THintWindow.WMPrint 6071
Controls.THintWindowClass 6310
Controls.THintWindowClass type 6310
Controls.TImageList 6078
Controls.TImageList.AllocBy 6083
Controls.TImageList.BkColor 6083
Controls.TImageList.BlendColor 6084
Controls.TImageList.DrawingStyle 6084
Controls.TImageList.Height 6085
Controls.TImageList.ImageType 6085
Controls.TImageList.Masked 6086
Controls.TImageList.OnChange 6087
Controls.TImageList.ShareImages 6086
Controls.TImageList.Width 6086
Controls.TImeMode 6285
Controls.TImeMode enumeration 6285
Controls.TImeName 6311
Controls.TImeName type 6311
Controls.TKeyEvent 6311
Controls.TKeyEvent type 6311
Controls.TKeyPressEvent 6311
Controls.TKeyPressEvent type 6311
Controls.TMargins 6087
Controls.TMargins.AssignTo 6090
Controls.TMargins.Bottom 6093
Controls.TMargins.Change 6091
Controls.TMargins.Control 6093
Controls.TMargins.ControlHeight 6093
Controls.TMargins.ControlLeft 6093
Controls.TMargins.ControlTop 6094
Controls.TMargins.ControlWidth 6094
Controls.TMargins.Create 6089
Controls.TMargins.ExplicitHeight 6094
Controls.TMargins.ExplicitLeft 6094
Controls.TMargins.ExplicitTop 6095
Controls.TMargins.ExplicitWidth 6095
Controls.TMargins.GetControlBound 6091
Controls.TMargins.InitDefaults 6091
Controls.TMargins.Left 6095
Controls.TMargins.OnChange 6096
Controls.TMargins.Right 6095
Controls.TMargins.SetBounds 6091
Controls.TMargins.SetControlBounds 6092
Controls.TMargins.Top 6095
Controls.TMarginSize 6312
Controls.TMarginSize type 6312
Controls.TModalResult 6312
Controls.TModalResult type 6312
Controls.TMouse 6096
Controls.TMouse.Capture 6099

Controls.TMouse.Create 6097
Controls.TMouse.CreatePanningWindow 6098
Controls.TMouse.CursorPos 6100
Controls.TMouse.Destroy 6098
Controls.TMouse.DragImmediate 6100
Controls.TMouse.DragThreshold 6101
Controls.TMouse.IsDragging 6101
Controls.TMouse.IsPanning 6101
Controls.TMouse.MousePresent 6101
Controls.TMouse.PanningWindow 6102
Controls.TMouse.PanningWindowClass 6102
Controls.TMouse.RegWheelMessage 6102
Controls.TMouse.SettingChanged 6098
Controls.TMouse.WheelPresent 6103
Controls.TMouse.WheelScrollLines 6103
Controls.TMouseActivate 6287
Controls.TMouseActivate enumeration 6287
Controls.TMouseActivateEvent 6313
Controls.TMouseActivateEvent type 6313
Controls.TMouseActivateRec 6288
Controls.TMouseActivateRec record 6288
Controls.TMouseButton 6288
Controls.TMouseButton enumeration 6288
Controls.TMouseEvent 6313
Controls.TMouseEvent type 6313
Controls.TMouseMoveEvent 6314
Controls.TMouseMoveEvent type 6314
Controls.TMouseWheelEvent 6314
Controls.TMouseWheelEvent type 6314
Controls.TMouseWheelUpDownEvent 6314
Controls.TMouseWheelUpDownEvent type 6314
Controls.TPadding 6103
Controls.TPadding.Bottom 6106
Controls.TPadding.InitDefaults 6105
Controls.TPadding.Left 6106
Controls.TPadding.Right 6106
Controls.TPadding.Top 6107
Controls.TPanningWindowClass 6315
Controls.TPanningWindowClass type 6315
Controls.TPopupFormInfo 6289
Controls.TPopupFormInfo record 6289
Controls.TScalingFlags 6315
Controls.TScalingFlags type 6315
Controls.TSizeConstraints 6107
Controls.TSizeConstraints.AssignTo 6109
Controls.TSizeConstraints.Change 6109
Controls.TSizeConstraints.Control 6110
Controls.TSizeConstraints.Create 6109
Controls.TSizeConstraints.MaxHeight 6110
Controls.TSizeConstraints.MaxWidth 6111
Controls.TSizeConstraints.MinHeight 6111
Controls.TSizeConstraints.MinWidth 6111
Controls.TSizeConstraints.OnChange 6112
Controls.TStartDockEvent 6316
Controls.TStartDockEvent type 6316
Controls.TStartDragEvent 6316
Controls.TStartDragEvent type 6316
Controls.TTabOrder 6317
Controls.TTabOrder type 6317
Controls.TTime 6317
Controls.TTime type 6317
Controls.TUnDockEvent 6317
Controls.TUnDockEvent type 6317
Controls.TWinControl 6112
Controls.TWinControl.ActionChange 6127
Controls.TWinControl.AddBiDiModeExStyle 6128
Controls.TWinControl.AdjustClientRect 6128
Controls.TWinControl.AdjustSize 6129
Controls.TWinControl.AlignControls 6129
Controls.TWinControl.AlignDisabled 6192
Controls.TWinControl.ArrangeControl 6130
Controls.TWinControl.AssignTo 6130
Controls.TWinControl.BevelEdges 6192
Controls.TWinControl.BevelInner 6193
Controls.TWinControl.BevelKind 6193
Controls.TWinControl.BevelOuter 6194
Controls.TWinControl.BevelWidth 6194
Controls.TWinControl.BorderWidth 6195

Controls.TWinControl.Broadcast 6130
Controls.TWinControl.Brush 6195
Controls.TWinControl.CanAutoSize 6131
Controls.TWinControl.CanFocus 6131
Controls.TWinControl.CanResize 6132
Controls.TWinControl.ChangeScale 6132
Controls.TWinControl.ConstrainedResize 6133
Controls.TWinControl.ContainsControl 6133
Controls.TWinControl.ControlAtPos 6133
Controls.TWinControl.ControlCount 6195
Controls.TWinControl.Controls 6197
Controls.TWinControl.ControlsAligned 6134
Controls.TWinControl.Create 6123
Controls.TWinControl.CreateDockManager 6134
Controls.TWinControl.CreateHandle 6135
Controls.TWinControl.CreateParams 6135
Controls.TWinControl.CreateParented 6123
Controls.TWinControl.CreateParentedControl 6136
Controls.TWinControl.CreateSubClass 6136
Controls.TWinControl.CreateWindowHandle 6137
Controls.TWinControl.CreateWnd 6137
Controls.TWinControl.Ctl3D 6198
Controls.TWinControl.CustomAlignInsertBefore 6137
Controls.TWinControl.CustomAlignPosition 6138
Controls.TWinControl.DefaultHandler 6139
Controls.TWinControl.DefineProperties 6139
Controls.TWinControl.DefWndProc 6199
Controls.TWinControl.Destroy 6124
Controls.TWinControl.DestroyHandle 6140
Controls.TWinControl.DestroyWindowHandle 6140
Controls.TWinControl.DestroyWnd 6141
Controls.TWinControl.DisableAlign 6141
Controls.TWinControl.DoAddDockClient 6142
Controls.TWinControl.DockClientCount 6199
Controls.TWinControl.DockClients 6201
Controls.TWinControl.DockDrop 6142
Controls.TWinControl.DockManager 6202
Controls.TWinControl.DockOver 6143
Controls.TWinControl.DockReplaceDockClient 6143
Controls.TWinControl.DockSite 6204
Controls.TWinControl.DoDockOver 6143
Controls.TWinControl.DoEnter 6144
Controls.TWinControl.DoExit 6144
Controls.TWinControl.DoFlipChildren 6145
Controls.TWinControl.DoKeyDown 6145
Controls.TWinControl.DoKeyPress 6146
Controls.TWinControl.DoKeyUp 6146
Controls.TWinControl.DoRemoveDockClient 6147
Controls.TWinControl.DoubleBuffered 6205
Controls.TWinControl.DoUnDock 6147
Controls.TWinControl.EnableAlign 6148
Controls.TWinControl.FDesignSize 6124
Controls.TWinControl.FDoubleBuffered 6124
Controls.TWinControl.FindChildControl 6148
Controls.TWinControl.FindNextControl 6149
Controls.TWinControl.FInImeComposition 6125
Controls.TWinControl.FixupTabList 6149
Controls.TWinControl.FlipChildren 6150
Controls.TWinControl.Focused 6150
Controls.TWinControl.GetActionLinkClass 6152
Controls.TWinControl.GetChildren 6153
Controls.TWinControl.GetClientOrigin 6153
Controls.TWinControl.GetClientRect 6154
Controls.TWinControl.GetControlExtents 6154
Controls.TWinControl.GetDeviceContext 6155
Controls.TWinControl.GetParentHandle 6157
Controls.TWinControl.GetSiteInfo 6158
Controls.TWinControl.GetTabOrderList 6158
Controls.TWinControl.GetTopParentHandle 6159
Controls.TWinControl.Handle 6207
Controls.TWinControl.HandleAllocated 6159
Controls.TWinControl.HandleNeeded 6160
Controls.TWinControl.ImeMode 6207
Controls.TWinControl.ImeName 6208
Controls.TWinControl.InsertControl 6160
Controls.TWinControl.Invalidate 6161
Controls.TWinControl.InvalidateDockHostSite 6162
Controls.TWinControl.IsControlActivateMsg 6162

Controls.TWinControl.IsControlMouseMsg 6162
Controls.TWinControl.IsQualifyingSite 6163
Controls.TWinControl.KeyDown 6163
Controls.TWinControl.KeyPress 6163
Controls.TWinControl.KeyUp 6164
Controls.TWinControl.MainWndProc 6165
Controls.TWinControl.MouseInClient 6209
Controls.TWinControl.Notification 6165
Controls.TWinControl.NotifyControls 6165
Controls.TWinControl.OnAlignInsertBefore 6213
Controls.TWinControl.OnAlignPosition 6214
Controls.TWinControl.OnDockDrop 6214
Controls.TWinControl.OnDockOver 6216
Controls.TWinControl.OnEnter 6219
Controls.TWinControl.OnExit 6221
Controls.TWinControl.OnGetSiteInfo 6223
Controls.TWinControl.OnKeyDown 6225
Controls.TWinControl.OnKeyPress 6227
Controls.TWinControl.OnKeyUp 6228
Controls.TWinControl.OnUnDock 6230
Controls.TWinControl.Padding 6209
Controls.TWinControl.PaintControls 6166
Controls.TWinControl.PaintHandler 6166
Controls.TWinControl.PaintTo 6167
Controls.TWinControl.PaintWindow 6169
Controls.TWinControl.PaletteChanged 6170
Controls.TWinControl.ParentBackground 6209
Controls.TWinControl.ParentCtl3D 6209
Controls.TWinControl.ParentWindow 6210
Controls.TWinControl.PreProcessMessage 6170
Controls.TWinControl.ReadState 6171
Controls.TWinControl.Realign 6171
Controls.TWinControl.RecreateWnd 6171
Controls.TWinControl.ReloadDockedControl 6172
Controls.TWinControl.RemoveControl 6172
Controls.TWinControl.RemoveWindowProps 6174
Controls.TWinControl.Repaint 6174
Controls.TWinControl.ResetIme 6174
Controls.TWinControl.ResetImeComposition 6175
Controls.TWinControl.ScaleBy 6175
Controls.TWinControl.ScaleControls 6176
Controls.TWinControl.ScrollBy 6176
Controls.TWinControl.SelectFirst 6177
Controls.TWinControl.SelectNext 6177
Controls.TWinControl.SetBounds 6178
Controls.TWinControl.SetChildOrder 6178
Controls.TWinControl.SetDesignVisible 6178
Controls.TWinControl.SetFocus 6179
Controls.TWinControl.SetIme 6181
Controls.TWinControl.SetImeCompositionWindow 6181
Controls.TWinControl.SetParent 6182
Controls.TWinControl.SetParentBackground 6182
Controls.TWinControl.SetZOrder 6182
Controls.TWinControl.ShowControl 6183
Controls.TWinControl.Showing 6210
Controls.TWinControl.TabOrder 6211
Controls.TWinControl.TabStop 6211
Controls.TWinControl.Update 6183
Controls.TWinControl.UpdateBounds 6185
Controls.TWinControl.UpdateControlOriginalParentSize 6186
Controls.TWinControl.UpdateControlState 6186
Controls.TWinControl.UpdateRecreatingFlag 6186
Controls.TWinControl.UpdateUIState 6186
Controls.TWinControl.UseDockManager 6212
Controls.TWinControl.VisibleDockClientCount 6212
Controls.TWinControl.WindowHandle 6212
Controls.TWinControl.WndProc 6187
Controls.TWinControlActionLink 6233
Controls.TWinControlActionLink.AssignClient 6236
Controls.TWinControlActionLink.FClient 6236
Controls.TWinControlActionLink.IsHelpContextLinked 6236
Controls.TWinControlActionLink.SetHelpContext 6237
Controls.TWinControlActionLinkClass 6318
Controls.TWinControlActionLinkClass type 6318
Controls.TWinControlClass 6318
Controls.TWinControlClass type 6318
Controls.TWndMethod 6319
Controls.TWndMethod type 6319

Controls.Win32NLSEnableIME 6266
Controls.Win32NLSEnableIME function 6266
ConvUtils 6379
ConvUtils namespace 6379
Classes 6381
Constants 6420
Functions 6395
Types 6416
Variables 6419
ConvUtils.CIllegalConvFamily 6420
ConvUtils.CIllegalConvFamily constant 6420
ConvUtils.CIllegalConvType 6420
ConvUtils.CIllegalConvType constant 6420
ConvUtils.CompatibleConversionType 6396
ConvUtils.CompatibleConversionType function 6396
ConvUtils.CompatibleConversionTypes 6396
ConvUtils.CompatibleConversionTypes function 6396
ConvUtils.Convert 6397
ConvUtils.Convert function 6397
ConvUtils.ConvertFrom 6398
ConvUtils.ConvertFrom function 6398
ConvUtils.ConvertTo 6399
ConvUtils.ConvertTo function 6399
ConvUtils.ConvFamilyToDescription 6399
ConvUtils.ConvFamilyToDescription function 6399
ConvUtils.ConvTypeToDescription 6400
ConvUtils.ConvTypeToDescription function 6400
ConvUtils.ConvTypeToFamily 6400
ConvUtils.ConvTypeToFamily function 6400
ConvUtils.ConvUnitAdd 6401
ConvUtils.ConvUnitAdd function 6401
ConvUtils.ConvUnitCompareValue 6402
ConvUtils.ConvUnitCompareValue function 6402
ConvUtils.ConvUnitDec 6402
ConvUtils.ConvUnitDec function 6402
ConvUtils.ConvUnitDiff 6403
ConvUtils.ConvUnitDiff function 6403
ConvUtils.ConvUnitInc 6404
ConvUtils.ConvUnitInc function 6404
ConvUtils.ConvUnitSameValue 6404
ConvUtils.ConvUnitSameValue function 6404
ConvUtils.ConvUnitToStr 6405
ConvUtils.ConvUnitToStr function 6405
ConvUtils.ConvUnitWithinNext 6406
ConvUtils.ConvUnitWithinNext function 6406
ConvUtils.ConvUnitWithinPrevious 6406
ConvUtils.ConvUnitWithinPrevious function 6406
ConvUtils.DescriptionToConvFamily 6407
ConvUtils.DescriptionToConvFamily function 6407
ConvUtils.DescriptionToConvType 6408
ConvUtils.DescriptionToConvType function 6408
ConvUtils.EConversionError 6381
ConvUtils.GConvUnitToStrFmt 6419
ConvUtils.GConvUnitToStrFmt variable 6419
ConvUtils.GetConvFamilies 6408
ConvUtils.GetConvFamilies function 6408
ConvUtils.GetConvTypes 6409
ConvUtils.GetConvTypes function 6409
ConvUtils.RaiseConversionError 6409
ConvUtils.RaiseConversionError function 6409
ConvUtils.RaiseConversionRegError 6410
ConvUtils.RaiseConversionRegError function 6410
ConvUtils.RegisterConversionFamily 6410
ConvUtils.RegisterConversionFamily function 6410
ConvUtils.RegisterConversionType 6411
ConvUtils.RegisterConversionType function 6411
ConvUtils.StrToConvUnit 6412
ConvUtils.StrToConvUnit function 6412
ConvUtils.TConversionProc 6416
ConvUtils.TConversionProc type 6416
ConvUtils.TConvFamily 6416
ConvUtils.TConvFamily type 6416
ConvUtils.TConvFamilyArray 6417
ConvUtils.TConvFamilyArray type 6417
ConvUtils.TConvType 6417
ConvUtils.TConvType type 6417
ConvUtils.TConvTypeArray 6417
ConvUtils.TConvTypeArray type 6417

ConvUtils.TConvTypeFactor 6382
ConvUtils.TConvTypeFactor.Create 6383
ConvUtils.TConvTypeFactor.Factor 6385
ConvUtils.TConvTypeFactor.FromCommon 6384
ConvUtils.TConvTypeFactor.ToCommon 6385
ConvUtils.TConvTypeInfo 6386
ConvUtils.TConvTypeInfo.ConvFamily 6390
ConvUtils.TConvTypeInfo.ConvType 6390
ConvUtils.TConvTypeInfo.Create 6388
ConvUtils.TConvTypeInfo.Description 6391
ConvUtils.TConvTypeInfo.FromCommon 6389
ConvUtils.TConvTypeInfo.ToCommon 6389
ConvUtils.TConvTypeList 6418
ConvUtils.TConvTypeList type 6418
ConvUtils.TConvTypeProcs 6391
ConvUtils.TConvTypeProcs.Create 6393
ConvUtils.TConvTypeProcs.FromCommon 6393
ConvUtils.TConvTypeProcs.ToCommon 6394
ConvUtils.TryConvTypeToFamily 6413
ConvUtils.TryConvTypeToFamily function 6413
ConvUtils.TryStrToConvUnit 6414
ConvUtils.TryStrToConvUnit function 6414
ConvUtils.UnregisterConversionFamily 6414
ConvUtils.UnregisterConversionFamily function 6414
ConvUtils.UnregisterConversionType 6415
ConvUtils.UnregisterConversionType function 6415
CorbaObj 6421
CorbaObj namespace 6421
    Classes 6423
    Functions 6477
    Interfaces 6475
    Structs, Records, Enums 6484
    Types 6487
    Variables 6493
CorbaObj.AnyToObject 6477
CorbaObj.AnyToObject function 6477
CorbaObj.BOA 6477
CorbaObj.BOA function 6477
CorbaObj.CorbaBind 6478
CorbaObj.CorbaBind function 6478
CorbaObj.CorbaBoolean 6487
CorbaObj.CorbaBoolean type 6487
CorbaObj.CorbaFactoryCreateStub 6478
CorbaObj.CorbaFactoryCreateStub function 6478
CorbaObj.CorbaInitialize 6479
CorbaObj.CorbaInitialize function 6479
CorbaObj.CorbaInterfaceIDManager 6493
CorbaObj.CorbaInterfaceIDManager variable 6493
CorbaObj.CorbaSkeletonManager 6493
CorbaObj.CorbaSkeletonManager variable 6493
CorbaObj.CorbaStubManager 6493
CorbaObj.CorbaStubManager variable 6493
CorbaObj.CorbaULong 6487
CorbaObj.CorbaULong type 6487
CorbaObj.ECorbaDispatch 6423
CorbaObj.ECorbaException 6423
CorbaObj.ECorbaException.GetMessage 6425
CorbaObj.ECorbaException.Name 6425
CorbaObj.ECorbaUserException 6426
CorbaObj.ECorbaUserException.Copy 6430
CorbaObj.ECorbaUserException.Create 6427
CorbaObj.ECorbaUserException.Proxy 6430
CorbaObj.ECorbaUserException.Throw 6430
CorbaObj.ICorbaObject 6475
CorbaObj.ICorbaObject.Hash 6476
CorbaObj.ICorbaObject.IsA 6476
CorbaObj.ICorbaObject.NonExistent 6476
CorbaObj.ICorbaObject.SetPrincipal 6476
CorbaObj.IObject 6488
CorbaObj.IObject type 6488
CorbaObj.ITypeCode 6488
CorbaObj.ITypeCode type 6488
CorbaObj.MakePrincipal 6479
CorbaObj.MakePrincipal function 6479
CorbaObj.MarshalAny 6479
CorbaObj.MarshalAny function 6479
CorbaObj.MarshalObject 6480
CorbaObj.MarshalObject function 6480

CorbaObj.MarshalWordBool 6480
CorbaObj.MarshalWordBool function 6480
CorbaObj.ORB 6480
CorbaObj.ORB function 6480
CorbaObj.SequenceToVariantArray 6481
CorbaObj.SequenceToVariantArray function 6481
CorbaObj.TAny 6488
CorbaObj.TAny type 6488
CorbaObj.TBOA 6431
CorbaObj.TBOA.Deactivate 6431
CorbaObj.TBOA.GetPrincipal 6432
CorbaObj.TBOA.ImplIsReady 6432
CorbaObj.TBOA.Initialize 6432
CorbaObj.TBOA.ObjIsReady 6432
CorbaObj.TCKind 6489
CorbaObj.TCKind type 6489
CorbaObj.TCommandLine 6489
CorbaObj.TCommandLine type 6489
CorbaObj.TCorbaDispatchStub 6433
CorbaObj.TCorbaDispatchStub.GetIDsOfNames 6435
CorbaObj.TCorbaDispatchStub.GetTypeInfo 6435
CorbaObj.TCorbaDispatchStub.GetTypeInfoCount 6435
CorbaObj.TCorbaDispatchStub.Invoke 6435
CorbaObj.TCorbaFactory 6436
CorbaObj.TCorbaFactory.Create 6437
CorbaObj.TCorbaFactory.CreateInstance 6439
CorbaObj.TCorbaFactory.CreateInterface 6439
CorbaObj.TCorbaFactory.Destroy 6438
CorbaObj.TCorbaFactory.Execute 6439
CorbaObj.TCorbaFactory.GetImplementation 6439
CorbaObj.TCorbaFactory.GetSkeleton 6440
CorbaObj.TCorbaFactory.GetTypeInfo 6440
CorbaObj.TCorbaFactory.InstanceName 6440
CorbaObj.TCorbaFactory.Instancing 6441
CorbaObj.TCorbaFactory.InterfaceName 6441
CorbaObj.TCorbaFactory.RegisterFactory 6440
CorbaObj.TCorbaFactory.RepositoryID 6441
CorbaObj.TCorbaFactory.ThreadModel 6441
CorbaObj.TCorbaFactoryManager 6442
CorbaObj.TCorbaFactoryManager.AddFactory 6443
CorbaObj.TCorbaFactoryManager.Destroy 6442
CorbaObj.TCorbaFactoryManager.Find 6443
CorbaObj.TCorbaFactoryManager.RegisterFactories 6443
CorbaObj.TCorbaImplementation 6444
CorbaObj.TCorbaImplementation._AddRef 6446
CorbaObj.TCorbaImplementation._Release 6447
CorbaObj.TCorbaImplementation.Create 6445
CorbaObj.TCorbaImplementation.FController 6445
CorbaObj.TCorbaImplementation.FFactory 6446
CorbaObj.TCorbaImplementation.FRefCount 6446
CorbaObj.TCorbaImplementation.GetIDsOfNames 6447
CorbaObj.TCorbaImplementation.GetTypeInfo 6447
CorbaObj.TCorbaImplementation.GetTypeInfoCount 6447
CorbaObj.TCorbaImplementation.Invoke 6448
CorbaObj.TCorbaImplementation.ObjAddRef 6448
CorbaObj.TCorbaImplementation.ObjQueryInterface 6448
CorbaObj.TCorbaImplementation.ObjRelease 6448
CorbaObj.TCorbaImplementation.QueryInterface 6448
CorbaObj.TCorbaImplementationClass 6489
CorbaObj.TCorbaImplementationClass type 6489
CorbaObj.TCorbaInstancing 6484
CorbaObj.TCorbaInstancing enumeration 6484
CorbaObj.TCorbaInterfaceIDManager 6449
CorbaObj.TCorbaInterfaceIDManager.FindGUID 6450
CorbaObj.TCorbaInterfaceIDManager.FindID 6450
CorbaObj.TCorbaInterfaceIDManager.RegisterInterface 6450
CorbaObj.TCorbaInterfaceIDManager.SearchGUID 6450
CorbaObj.TCorbaInterfaceIDManager.SearchID 6451
CorbaObj.TCorbaListManager 6451
CorbaObj.TCorbaListManager.BeginRead 6452
CorbaObj.TCorbaListManager.BeginWrite 6452
CorbaObj.TCorbaListManager.Create 6451
CorbaObj.TCorbaListManager.Destroy 6452
CorbaObj.TCorbaListManager.EndRead 6453
CorbaObj.TCorbaListManager.EndWrite 6453
CorbaObj.TCorbaObjectFactory 6453
CorbaObj.TCorbaObjectFactory.Create 6455
CorbaObj.TCorbaObjectFactory.CreateInterface 6455

CorbaObj.TCorbaObjectFactory.ImplementationClass 6456
CorbaObj.TCorbaPrincipal 6490
CorbaObj.TCorbaPrincipal type 6490
CorbaObj.TCorbaSkeleton 6456
CorbaObj.TCorbaSkeleton.Create 6458
CorbaObj.TCorbaSkeleton.Destroy 6458
CorbaObj.TCorbaSkeleton.Execute 6459
CorbaObj.TCorbaSkeleton.FSkeleton 6459
CorbaObj.TCorbaSkeleton.GetImplementation 6459
CorbaObj.TCorbaSkeleton.GetSkeleton 6460
CorbaObj.TCorbaSkeleton.InitSkeleton 6460
CorbaObj.TCorbaSkeletonClass 6490
CorbaObj.TCorbaSkeletonClass type 6490
CorbaObj.TCorbaSkeletonManager 6460
CorbaObj.TCorbaSkeletonManager.CreateSkeleton 6461
CorbaObj.TCorbaSkeletonManager.RegisterSkeleton 6461
CorbaObj.TCorbaStub 6462
CorbaObj.TCorbaStub.Create 6463
CorbaObj.TCorbaStub.Destroy 6464
CorbaObj.TCorbaStub.FStub 6464
CorbaObj.TCorbaStub.GetStub 6465
CorbaObj.TCorbaStub.Hash 6465
CorbaObj.TCorbaStub.IsA 6465
CorbaObj.TCorbaStub.NonExistent 6465
CorbaObj.TCorbaStub.SetPrincipal 6466
CorbaObj.TCorbaStubClass 6490
CorbaObj.TCorbaStubClass type 6490
CorbaObj.TCorbaStubManager 6466
CorbaObj.TCorbaStubManager.CreateStub 6467
CorbaObj.TCorbaStubManager.RegisterStub 6467
CorbaObj.TCorbaThreadModel 6484
CorbaObj.TCorbaThreadModel enumeration 6484
CorbaObj.TFactoryList 6491
CorbaObj.TFactoryList type 6491
CorbaObj.TInterfaceIDEntryDesc 6485
CorbaObj.TInterfaceIDEntryDesc record 6485
CorbaObj.TInterfaceIDList 6491
CorbaObj.TInterfaceIDList type 6491
CorbaObj.TORB 6467
CorbaObj.TORB.Bind 6469
CorbaObj.TORB.FindTypeCode 6469
CorbaObj.TORB.Initialize 6470
CorbaObj.TORB.MakeAlias 6470
CorbaObj.TORB.MakeAliasTypeCode 6470
CorbaObj.TORB.MakeArray 6471
CorbaObj.TORB.MakeObjectRefTypeCode 6471
CorbaObj.TORB.MakeSequence 6472
CorbaObj.TORB.MakeSequenceTypeCode 6472
CorbaObj.TORB.MakeStructure 6472
CorbaObj.TORB.MakeStructureTypeCode 6473
CorbaObj.TORB.MakeTypeCode 6473
CorbaObj.TORB.ObjectToString 6473
CorbaObj.TORB.Shutdown 6473
CorbaObj.TORB.StringToObject 6473
CorbaObj.TSkeletonEntryDesc 6485
CorbaObj.TSkeletonEntryDesc record 6485
CorbaObj.TSkeletonList 6491
CorbaObj.TSkeletonList type 6491
CorbaObj.TStubEntryDesc 6486
CorbaObj.TStubEntryDesc record 6486
CorbaObj.TStubList 6492
CorbaObj.TStubList type 6492
CorbaObj.UnmarshalAny 6481
CorbaObj.UnmarshalAny function 6481
CorbaObj.UnmarshalObject 6481
CorbaObj.UnmarshalObject function 6481
CorbaObj.UnmarshalText 6482
CorbaObj.UnmarshalText function 6482
CorbaObj.UnmarshalWideText 6482
CorbaObj.UnmarshalWideText function 6482
CorbaObj.UnmarshalWordBool 6482
CorbaObj.UnmarshalWordBool function 6482
CorbaObj.VariantArrayToSequence 6483
CorbaObj.VariantArrayToSequence function 6483
coResync enumeration member 1294
coRightToLeft enumeration member 14632
CoSearchAssistantOC class 19047
about CoSearchAssistantOC class 19047

CoSearchAssistantOC Members 19048
CoSearchAssistantOC methods 19048
Create 19048
CreateRemote 19048
coSeek enumeration member 1294
coSelective enumeration member 565
CoShellBrowserWindow class 19049
about CoShellBrowserWindow class 19049
CoShellBrowserWindow Members 19049
CoShellBrowserWindow methods 19049
Create 19049
CreateRemote 19050
CoShellNameSpace class 19050
about CoShellNameSpace class 19050
CoShellNameSpace Members 19050
CoShellNameSpace methods 19050
Create 19051
CreateRemote 19051
CoShellUIHelper class 19051
about CoShellUIHelper class 19051
CoShellUIHelper Members 19051
CoShellUIHelper methods 19052
Create 19052
CreateRemote 19052
CoShellWindows class 19052
about CoShellWindows class 19052
CoShellWindows Members 19053
CoShellWindows methods 19053
Create 19053
CreateRemote 19053
coUpdate enumeration member 1294
coUpdateBatch enumeration member 1294
crAlways enumeration member 15247
crEqual enumeration member 23672
crGreaterThan enumeration member 23672
crLessThan enumeration member 23672
crOnDemand enumeration member 15247
csAllocated enumeration member 22554
csBrushValid enumeration member 14633
csContinuePending enumeration member 22047
csCustom enumeration member 10347
csCustomized enumeration member 9767
csDefault enumeration member 9767, 10347
csDirty enumeration member 17837
csDropDown enumeration member 21946
csDropDownList enumeration member 21946
csExCaseSensitive enumeration member 5442
csExDropDown enumeration member 5442
csExDropDownList enumeration member 5442
csExNoEditImage enumeration member 5442
csExNoEditImageIndent enumeration member 5442
csExNoSizeLimit enumeration member 5442
csExPathWordBreak enumeration member 5442
csExSimple enumeration member 5442
csFontValid enumeration member 14633
csHandleValid enumeration member 14633
csHasIndex enumeration member 17837
csNormal enumeration member 17675
csOwnerDrawFixed enumeration member 21946
csOwnerDrawVariable enumeration member 21946
csPaused enumeration member 22047
csPausePending enumeration member 22047
csPenValid enumeration member 14633
csPivoting enumeration member 17675
csRefreshing enumeration member 17837
csReserved enumeration member 22554
csRunning enumeration member 22047
csShowProgress enumeration member 17837
csSimple enumeration member 21946
csStartPending enumeration member 22047
csStateClosed enumeration member 20365
csStateConnecting enumeration member 20365
csStateDisconnecting enumeration member 20365
csStateExecuting enumeration member 20365
csStateFetching enumeration member 20365
csStateOpen enumeration member 20365
csStopped enumeration member 22047
csStopPending enumeration member 22047

csSuccess enumeration member 17837
csSysAllocated enumeration member 22554
csSysReserved enumeration member 22554
csUnallocated enumeration member 22554
ctAnchor enumeration member 25316
ctBlocking enumeration member 18885
ctButton enumeration member 25316
ctCaptionCol enumeration member 17673
ctCaptionRow enumeration member 17673
ctColIndicator enumeration member 17673
ctColLabel enumeration member 17673
ctColPlus enumeration member 17673
ctData enumeration member 17673
ctDCOM enumeration member 16472
ctDDL enumeration member 7621
ctDefault enumeration member 25316
ctDelete enumeration member 7621
ctDynamic enumeration member 1295
ctEventImages enumeration member 25316
ctFromData enumeration member 18013
ctFromFile enumeration member 18013
ctImage enumeration member 5452, 25316
ctInsert enumeration member 7621
ctKeyset enumeration member 1295
ctLinkFromData enumeration member 18013
ctLinkToFile enumeration member 18013
CtlPanel 6495
CtlPanel namespace 6495
    Classes 6497
    Functions 6519
    Types 6520
    Variables 6526
CtlPanel.Application 6526
CtlPanel.Application variable 6526
CtlPanel.CPlApplet 6519
CtlPanel.CPlApplet function 6519
CtlPanel.EAppletException 6497
CtlPanel.TActivateEvent 6520
CtlPanel.TActivateEvent type 6520
CtlPanel.TAppletApplication 6497
CtlPanel.TAppletApplication.ControlPanelHandle 6505
CtlPanel.TAppletApplication.Create 6501
CtlPanel.TAppletApplication.CreateForm 6502
CtlPanel.TAppletApplication.Destroy 6501
CtlPanel.TAppletApplication.DoCount 6502
CtlPanel.TAppletApplication.DoExit 6503
CtlPanel.TAppletApplication.DoHandleException 6503
CtlPanel.TAppletApplication.DoInit 6503
CtlPanel.TAppletApplication.DoSetup 6503
CtlPanel.TAppletApplication.HandleException 6503
CtlPanel.TAppletApplication.Initialize 6504
CtlPanel.TAppletApplication.ModuleCount 6505
CtlPanel.TAppletApplication.Modules 6505
CtlPanel.TAppletApplication.OnCount 6506
CtlPanel.TAppletApplication.OnException 6507
CtlPanel.TAppletApplication.OnExit 6506
CtlPanel.TAppletApplication.OnInit 6507
CtlPanel.TAppletApplication.OnSetup 6506
CtlPanel.TAppletApplication.Run 6504
CtlPanel.TAppletModule 6507
CtlPanel.TAppletModule.AppletIcon 6513
CtlPanel.TAppletModule.Caption 6514
CtlPanel.TAppletModule.Create 6511
CtlPanel.TAppletModule.Data 6514
CtlPanel.TAppletModule.Destroy 6511
CtlPanel.TAppletModule.DoActivate 6512
CtlPanel.TAppletModule.DoInquire 6512
CtlPanel.TAppletModule.DoNewInquire 6512
CtlPanel.TAppletModule.DoStartWParms 6513
CtlPanel.TAppletModule.DoStop 6513
CtlPanel.TAppletModule.Help 6514
CtlPanel.TAppletModule.OnActivate 6516
CtlPanel.TAppletModule.OnInquire 6516
CtlPanel.TAppletModule.OnNewInquire 6517
CtlPanel.TAppletModule.OnStartWParms 6517
CtlPanel.TAppletModule.OnStop 6517
CtlPanel.TAppletModule.ResidIcon 6515
CtlPanel.TAppletModule.ResidInfo 6515

CtlPanel.TAppletModule.ResidName 6515
CtlPanel.TAppletModuleClass 6520
CtlPanel.TAppletModuleClass type 6520
CtlPanel.TCountEvent 6521
CtlPanel.TCountEvent type 6521
CtlPanel.TCPLAppletClass 6521
CtlPanel.TCPLAppletClass type 6521
CtlPanel.TDataModuleClass 6522
CtlPanel.TDataModuleClass type 6522
CtlPanel.TExitEvent 6522
CtlPanel.TExitEvent type 6522
CtlPanel.TInitEvent 6522
CtlPanel.TInitEvent type 6522
CtlPanel.TInquireEvent 6523
CtlPanel.TInquireEvent type 6523
CtlPanel.TNewInquireEvent 6523
CtlPanel.TNewInquireEvent type 6523
CtlPanel.TOnAppletExceptionEvent 6524
CtlPanel.TOnAppletExceptionEvent type 6524
CtlPanel.TSetupEvent 6524
CtlPanel.TSetupEvent type 6524
CtlPanel.TStartWParmsEvent 6524
CtlPanel.TStartWParmsEvent type 6524
CtlPanel.TStopEvent 6525
CtlPanel.TStopEvent type 6525
ctNewCaptionCol enumeration member 17673
ctNewObject enumeration member 18013
ctNonBlocking enumeration member 18885
ctNone enumeration member 17673, 17838
ctOLEnterprise enumeration member 16472
ctOpenForwardOnly enumeration member 1295
ctPreCalc enumeration member 17838
ctQuery enumeration member 7621
ctRowIndicator enumeration member 17673
ctRowLabel enumeration member 17673
ctRowPlus enumeration member 17673
ctRunning enumeration member 17838
ctSelect enumeration member 7621
ctSockets enumeration member 16472
ctState enumeration member 5452
ctStatic enumeration member 1295
ctStoredProc enumeration member 7621
ctTable enumeration member 7621
ctText enumeration member 5452
ctUnknown enumeration member 7621
ctUnspecified enumeration member 1295
ctUpdate enumeration member 7621
curpropCONSTRAINT_ERROR_MESSAGE enumeration member 11706
curpropDSBASE enumeration member 11706
curpropGETORG_RECBUF enumeration member 11706
curpropSETCRACK enumeration member 11706
Currency class 22145
- 22149
-- 22150
- operator 22148
-- operator 22150
! 22150
!= 22151
!= operator 22151
% 22151
%= 22152
* 22152
* operator 22152
*= 22153
/ 22154
/ operator 22153
/= 22154
+ 22155
+ operator 22155
++ 22156
++ operator 22156
+= 22156
< 22157
< operator 22157
<= 22158
<= operator 22158
= 22159
-= 22159

= operator 22158
== 22160
== operator 22160
> 22161
> operator 22160
>= 22161
>= operator 22161
about Currency class 22145
AnsiString 22162
Currency 22147
Currency __fastcall operator -(double lhs, const Currency& rhs) 22164
Currency __fastcall operator -(int lhs, const Currency& rhs) 22164
Currency __fastcall operator *(double lhs, const Currency& rhs) 22164
Currency __fastcall operator *(int lhs, const Currency& rhs) 22164
Currency __fastcall operator /(double lhs, const Currency& rhs) 22165
Currency __fastcall operator /(int lhs, const Currency& rhs) 22165
Currency __fastcall operator +(double lhs, const Currency& rhs) 22165
Currency __fastcall operator +(int lhs, const Currency& rhs) 22165
Currency constructor 22147
Currency friends 22163
Currency Members 22145
Currency operators 22148
double 22162
int 22163
CurrencyBase class 22166
about CurrencyBase class 22166
CurrencyBase data members 22166
CurrencyBase Members 22166
Val 22166
CustomizeDlg 6527
CustomizeDlg namespace 6527
Classes 6528
CustomizeDlg.TCustomizeDlg 6528
CustomizeDlg.TCustomizeDlg.ActionManager 6533
CustomizeDlg.TCustomizeDlg.CustomizeForm 6533
CustomizeDlg.TCustomizeDlg.Notification 6531
CustomizeDlg.TCustomizeDlg.OnClose 6534
CustomizeDlg.TCustomizeDlg.OnShow 6534
CustomizeDlg.TCustomizeDlg.SetupDlg 6532
CustomizeDlg.TCustomizeDlg.Show 6532
CustomizeDlg.TCustomizeDlg.StayOnTop 6533
CustomizeDlg.TCustomizeFrm 6535
CustomizeDlg.TCustomizeFrm.ActionBarList 6554
CustomizeDlg.TCustomizeFrm.ActionBarListClick 6565
CustomizeDlg.TCustomizeFrm.ActionBarListClickCheck 6565
CustomizeDlg.TCustomizeFrm.ActionImages 6554
CustomizeDlg.TCustomizeFrm.ActionList1 6554
CustomizeDlg.TCustomizeFrm.ActionManager 6573
CustomizeDlg.TCustomizeFrm.ActionsActionsLbl 6554
CustomizeDlg.TCustomizeFrm.ActionsCatLbl 6554
CustomizeDlg.TCustomizeFrm.ActionsList 6555
CustomizeDlg.TCustomizeFrm.ActionsListClick 6566
CustomizeDlg.TCustomizeFrm.ActionsListData 6566
CustomizeDlg.TCustomizeFrm.ActionsListDrawItem 6566
CustomizeDlg.TCustomizeFrm.ActionsListMeasureItem 6566
CustomizeDlg.TCustomizeFrm.ActionsListStartDrag 6566
CustomizeDlg.TCustomizeFrm.ActionsTab 6555
CustomizeDlg.TCustomizeFrm.ActiveActionList 6574
CustomizeDlg.TCustomizeFrm.AddAction 6567
CustomizeDlg.TCustomizeFrm.ApplyToAllActn 6555
CustomizeDlg.TCustomizeFrm.ApplyToAllActnUpdate 6567
CustomizeDlg.TCustomizeFrm.ApplyToAllChk 6555
CustomizeDlg.TCustomizeFrm.CaptionOptionsCombo 6556
CustomizeDlg.TCustomizeFrm.CaptionOptionsComboChange 6567
CustomizeDlg.TCustomizeFrm.CaptionOptionsGrp 6556
CustomizeDlg.TCustomizeFrm.CatList 6556
CustomizeDlg.TCustomizeFrm.CatListClick 6567
CustomizeDlg.TCustomizeFrm.CatListStartDrag 6568
CustomizeDlg.TCustomizeFrm.ClearCatList 6568
CustomizeDlg.TCustomizeFrm.CloseActn 6556
CustomizeDlg.TCustomizeFrm.CloseBtn 6556
CustomizeDlg.TCustomizeFrm.CloseBtnClick 6568
CustomizeDlg.TCustomizeFrm.CloseItem 6557

CustomizeDlg.TCustomizeFrm.CloseMenu 6557
CustomizeDlg.TCustomizeFrm.CMVisiblechanged 6568
CustomizeDlg.TCustomizeFrm.ComboPanel 6557
CustomizeDlg.TCustomizeFrm.DescGroupBox 6557
CustomizeDlg.TCustomizeFrm.FindActionBars 6568
CustomizeDlg.TCustomizeFrm.FormClose 6569
CustomizeDlg.TCustomizeFrm.FormCreate 6569
CustomizeDlg.TCustomizeFrm.FormResize 6569
CustomizeDlg.TCustomizeFrm.FullMenusActn 6558
CustomizeDlg.TCustomizeFrm.HintLbl 6558
CustomizeDlg.TCustomizeFrm.InfoLbl 6558
CustomizeDlg.TCustomizeFrm.IsDupShortCut 6569
CustomizeDlg.TCustomizeFrm.Label1 6558
CustomizeDlg.TCustomizeFrm.Label2 6558
CustomizeDlg.TCustomizeFrm.Label4 6559
CustomizeDlg.TCustomizeFrm.LargeIconsActn 6559
CustomizeDlg.TCustomizeFrm.LargeIconsActnExecute 6570
CustomizeDlg.TCustomizeFrm.LargeIconsActnUpdate 6570
CustomizeDlg.TCustomizeFrm.LargeIconsChk 6559
CustomizeDlg.TCustomizeFrm.ListCombo 6559
CustomizeDlg.TCustomizeFrm.ListComboSelect 6570
CustomizeDlg.TCustomizeFrm.ListPanel 6560
CustomizeDlg.TCustomizeFrm.MenuAnimationStyles 6560
CustomizeDlg.TCustomizeFrm.MenuAnimationStylesChange 6570
CustomizeDlg.TCustomizeFrm.Notification 6570
CustomizeDlg.TCustomizeFrm.OptionsBevel1 6560
CustomizeDlg.TCustomizeFrm.OptionsBevel2 6560
CustomizeDlg.TCustomizeFrm.OptionsTab 6560
CustomizeDlg.TCustomizeFrm.OtherLbl 6561
CustomizeDlg.TCustomizeFrm.PersonalizeLbl 6561
CustomizeDlg.TCustomizeFrm.RecentlyUsedActn 6561
CustomizeDlg.TCustomizeFrm.RecentlyUsedActnExecute 6571
CustomizeDlg.TCustomizeFrm.RecentlyUsedActnUpdate 6571
CustomizeDlg.TCustomizeFrm.RecentlyUsedChk 6561
CustomizeDlg.TCustomizeFrm.ResetActn 6562
CustomizeDlg.TCustomizeFrm.ResetActnExecute 6571
CustomizeDlg.TCustomizeFrm.ResetActnUpdate 6572
CustomizeDlg.TCustomizeFrm.ResetBtn 6562
CustomizeDlg.TCustomizeFrm.ResetUsageBtn 6562
CustomizeDlg.TCustomizeFrm.ResetUsageDataActn 6562
CustomizeDlg.TCustomizeFrm.ResetUsageDataActnExecute 6572
CustomizeDlg.TCustomizeFrm.SeparatorBtn 6562
CustomizeDlg.TCustomizeFrm.SeparatorBtnStartDrag 6572
CustomizeDlg.TCustomizeFrm.SetActiveActionList 6572
CustomizeDlg.TCustomizeFrm.ShortCutTipsChk 6563
CustomizeDlg.TCustomizeFrm.ShowHintsActn 6563
CustomizeDlg.TCustomizeFrm.ShowHintsActnExecute 6572
CustomizeDlg.TCustomizeFrm.ShowHintsActnUpdate 6573
CustomizeDlg.TCustomizeFrm.ShowShortCutsInTipsActn 6563
CustomizeDlg.TCustomizeFrm.ShowShortCutsInTipsActnExecute 6573
CustomizeDlg.TCustomizeFrm.ShowTipsChk 6563
CustomizeDlg.TCustomizeFrm.Tabs 6564
CustomizeDlg.TCustomizeFrm.ToolbarsLbl 6564
CustomizeDlg.TCustomizeFrm.ToolbarsTab 6564
CustomizeDlg.TCustomizeFrm.UpdateDialog 6573
cvAlignment enumeration member 9766
cvColor enumeration member 9766
cvFont enumeration member 9766
cvImeMode enumeration member 9766
cvImeName enumeration member 9766
cvReadOnly enumeration member 9766
cvTitleAlignment enumeration member 9766
cvTitleCaption enumeration member 9766
cvTitleColor enumeration member 9766
cvTitleFont enumeration member 9766
cvWidth enumeration member 9766

D

daAbort enumeration member 7606
daCenter enumeration member 9538
daFail enumeration member 7606
daLeft enumeration member 9538
daRetry enumeration member 7606
daRight enumeration member 9538
DatabaseColumns class 1906
    about DatabaseColumns class 1906

ALogDir 1908
BlockSize 1908
CheckpointFrequency 1908
DatabaseColumns fields 1907
DatabaseColumns Members 1907
DefaultSchemaName 1908
LogBlockSize 1908
LogEnabled 1909
LogPreallocateCount 1909
MaxLogSize 1909
MaxOpenLogs 1909
RecordStatus 1909
SoftCommit 1910
StoreVersion 1910
DatabaseStatusTable class 1910
about DatabaseStatusTable class 1910
DatabaseStatusTable fields 1911
DatabaseStatusTable Members 1911
HighestLsn 1911
LastCheckpointLsn 1911
LowestLsn 1911
LowestLsnRemarks 1912
DataBkr 6575
DataBkr namespace 6575
Classes 6576
Functions 6600
DataBkr.DisableSocketTransport 6600
DataBkr.DisableSocketTransport function 6600
DataBkr.DisableWebTransport 6601
DataBkr.DisableWebTransport function 6601
DataBkr.EnableSocketTransport 6601
DataBkr.EnableSocketTransport function 6601
DataBkr.EnableWebTransport 6602
DataBkr.EnableWebTransport function 6602
DataBkr.RegisterPooled 6602
DataBkr.RegisterPooled function 6602
DataBkr.TCRemoteDataModule 6576
DataBkr.TCRemoteDataModule.CRDMGetProviderNames 6581
DataBkr.TCRemoteDataModule.Create 6580
DataBkr.TCRemoteDataModule.Destroy 6580
DataBkr.TCRemoteDataModule.GetProvider 6581
DataBkr.TCRemoteDataModule.Lock 6582
DataBkr.TCRemoteDataModule.ProviderCount 6584
DataBkr.TCRemoteDataModule.Providers 6584
DataBkr.TCRemoteDataModule.RegisterProvider 6582
DataBkr.TCRemoteDataModule.UnLock 6583
DataBkr.TCRemoteDataModule.UnRegisterProvider 6583
DataBkr.TRemoteDataModule 6585
DataBkr.TRemoteDataModule.AS_ApplyUpdates 6590
DataBkr.TRemoteDataModule.AS_DataRequest 6591
DataBkr.TRemoteDataModule.AS_Execute 6591
DataBkr.TRemoteDataModule.AS_GetParams 6592
DataBkr.TRemoteDataModule.AS_GetProviderNames 6593
DataBkr.TRemoteDataModule.AS_GetRecords 6593
DataBkr.TRemoteDataModule.AS_RowRequest 6594
DataBkr.TRemoteDataModule.Create 6589
DataBkr.TRemoteDataModule.Destroy 6589
DataBkr.TRemoteDataModule.GetProvider 6595
DataBkr.TRemoteDataModule.Lock 6595
DataBkr.TRemoteDataModule.ProviderCount 6599
DataBkr.TRemoteDataModule.Providers 6599
DataBkr.TRemoteDataModule.RegisterProvider 6596
DataBkr.TRemoteDataModule.Unlock 6596
DataBkr.TRemoteDataModule.UnRegisterProvider 6597
DataBkr.TRemoteDataModule.UpdateRegistry 6597
DataBkr.UnregisterPooled 6603
DataBkr.UnregisterPooled function 6603
DataStoreCommand class 1912
about DataStoreCommand class 1912
Cancel 1915
Clone 1915
Close 1916
CommandText 1919
CommandTimeout 1920
CommandType 1920
Connection 1921
CreateDbParameter 1916
CreateParameter 1916

DataReaderClosed 1916
DataStoreCommand 1914
DataStoreCommand constructor 1914
DataStoreCommand Members 1913
DataStoreCommand methods 1915
DataStoreCommand properties 1919
DbConnection 1921
DbParameterCollection 1921
DbTransaction 1922
DesignTimeVisible 1922
Dispose 1917
Dispose method 1917
ExecuteDbDataReader 1917
ExecuteNonQuery 1917
ExecuteReader 1918
ExecuteReader method 1918
ExecuteScalar 1918
FailIfNotOpen 1919
Parameters 1922
Prepare 1919
RecordsAffected 1922
Transaction 1923
UpdatedRowSource 1923
DataStoreConnection class 1923
about DataStoreConnection class 1923
BeginDbTransaction 1926
BeginTransaction 1926
BeginTransaction method 1926
ChangeDatabase 1927
Clone 1927
Close 1927
ConnectionString 1930
ConnectionTimeout 1931
CreateCommand 1928
CreateDbCommand 1928
Database 1931
DataSource 1931
DataStoreConnection 1925
DataStoreConnection constructor 1925
DataStoreConnection Members 1924
DataStoreConnection methods 1925
DataStoreConnection properties 1930
Dispose 1928
Dispose method 1928
EndTransaction 1929
FailIfNotOpen 1929
GetSchema 1929
GetSchema method 1929
InternalState 1932
MetaDataGenerator 1932
Open 1930
ServerVersion 1932
State 1932
DataStoreDataAdapter class 1933
about DataStoreDataAdapter class 1933
BeginInit 1935
CreateRowUpdatedEvent 1935
CreateRowUpdatingEvent 1936
DataSet 1937
DataStoreDataAdapter 1934
DataStoreDataAdapter constructor 1934
DataStoreDataAdapter events 1939
DataStoreDataAdapter Members 1933
DataStoreDataAdapter methods 1935
DataStoreDataAdapter properties 1936
DeleteCommand 1937
EndInit 1936
InsertCommand 1937
OnRowUpdated 1936
OnRowUpdating 1936
RowUpdated 1939
RowUpdating 1939
SelectCommand 1938
UpdateCommand 1938
DataStoreErrorCode class 1939
about DataStoreErrorCode class 1939
ActionComplete 1965
AfterLast 1966

AggNotValidHere 1966
AggOperatorNotFound 1966
AlreadyLoading 1966
AmbiguousLogdir 1966
AmbiguousType 1967
AnchorTooOld 1967
AppendFailed 1967
ApplicationError 1967
BadColumnIndex 1967
BadDirection 1968
BadFetchsize 1968
BadFieldlist 1968
BadMaxfieldsize 1968
BadMaxrows 1968
BadOrdinal 1969
BadProcedureProperties 1969
BadQueryProperties 1969
BadUrl 1969
BeforeFirst 1969
BigdecimalPrecisionError 1970
BirthstampMismatch 1970
CannotAssignTypes 1970
CannotCast 1970
CannotChangeColumn 1970
CannotChangeColumnDataType 1971
CannotDittoExisting 1971
CannotFindTableName 1971
CannotImportNullDataset 1971
CannotLockDir 1971
CannotMixNamedParameters 1972
CannotOpen 1972
CannotOrphanDetails 1972
CannotRefresh 1972
CannotRestructure 1972
CannotSaveChanges 1973
CannotStartServer 1973
CannotUpdateScopedDataRow 1973
CannotUseInternalrow 1973
CannotUseViewHere 1973
CantCreateOpenFile 1974
CantCreateOpenStream 1974
ClassNotFoundError 1974
ColumnAlreadyBound 1974
ColumnIsReadonly 1974
ColumnNeedsRestructure 1975
ColumnNotInLeftRef 1975
ColumnNotInRightRef 1975
ColumnSpecifiedTwice 1975
ColumnTypeConflict 1975
ConflictingMethodDef 1976
ConnectionClosed 1976
ConnectionCountExceeded 1976
ConnectionDescriptorNotSet 1976
ConnectionError 1976
ConnectionNotClosed 1977
ConnectionPoolClosed 1977
ConnectionTimeout 1977
ConstraintNameExists 1977
CreateErrorMethodExists 1977
CyclesInViewDefinition 1978
DataFileLoadFailed 1978
DataHasDuplicates 1978
DatasetAlreadyOpen 1978
DatasetCorrupt 1978
DatasetExists 1979
DatasetHasNoRows 1979
DatasetHasNoTables 1979
DatasetNotOpen 1979
DatasetOpen 1979
DatastoreAlreadyOpen 1980
DataStoreErrorCode fields 1959
DataStoreErrorCode Members 1952
DatastoreExists 1980
DatastoreInvalid 1980
DatastoreIsUnlicensed 1980
DatastoreNotFound 1980
DatastoreNotOpen 1981
DatastoreOpen 1981

DbAdminError 1981
Deadlock 1981
DeleteDuplicates 1981
DeleteFailed 1982
DeleteNotAllowed 1982
DesktopLicense 1982
DifferentDegree 1982
DifferentSymbol 1982
DivideByZero 1983
DriverNotLoadedAtRuntime 1983
DriverNotLoadedInDesign 1983
DropErrorDependentView 1983
DropErrorForeignKey 1983
DropErrorRoleGrant 1984
DropSchemaError 1984
DuplicateColumnName 1984
DuplicateKey 1984
DuplicatePrimary 1984
EmptyColumnNames 1985
EofExpected 1985
ExceptionChain 1985
FailoverError 1985
FeatureNotSupp 1985
FieldPostError 1986
FileExists 1986
FileNotFound 1986
FloatPrecision 1986
ForeignKeyError 1986
ForeignKeyViolation 1987
ForwardOnly 1987
GenericError 1987
GreaterThanMax 1987
HasResultset 1987
IllegalCatalog 1988
IllegalCharacter 1988
IllegalExprStart 1988
IllegalMethodCall 1988
IncompatibleBlockSize 1988
IncompatibleDataRow 1989
IncompatibleTypes 1989
IncompatibleTypesForOp 1989
IndexAlreadyExists 1989
InlineTooSmall 1989
InsertFailed 1990
InsertNotAllowed 1990
InsufficientRowid 1990
IntegerRequired 1990
InternalDuplicateKey 1990
InternalError 1991
InternalUpdateFailed 1991
InvalidAggDescriptor 1991
InvalidAggExpr 1991
InvalidAnchorFile 1991
InvalidAnchorLength 1992
InvalidBinaryString 1992
InvalidClass 1992
InvalidColumnFormat 1992
InvalidColumnname 1992
InvalidColumnPosition 1993
InvalidColumnRef 1993
InvalidColumnType 1993
InvalidColumnValue 1993
InvalidConstraintName 1993
InvalidDataFileFormat 1994
InvalidDirectoryAttributes 1994
InvalidEscapeCharInLike 1994
InvalidExtractField 1994
InvalidFormat 1994
InvalidIndexname 1995
InvalidIteratorUse 1995
InvalidJoin 1995
InvalidLikePattern 1995
InvalidLsn 1995
InvalidPattern 1996
InvalidPrecision 1996
InvalidRowValues 1996
InvalidSavepoint 1996
InvalidSchemaFile 1996

InvalidSort 1997
InvalidSortAsInserted 1997
InvalidSortColumn 1997
InvalidStoreClass 1997
InvalidStoreName 1997
InvalidTablename 1998
InvalidTableRef 1998
InvalidTrimChar 1998
InvalidUserName 1998
InvalidViewDefinition 1998
IoError 1999
IoNetError 1999
IoProtocolError 1999
IsInsertRow 1999
LessThanMin 1999
LinkColumnsError 2000
LinkfieldInUserparameters 2000
LoadingNotStarted 2000
LockTimeout 2000
LogdirMismatch 2000
LogExists 2001
MalformedDateOrTime 2001
MalformedDecimal 2001
MalformedFloating 2001
MasterDetailViewError 2001
MasterNavigationError 2002
MaxResolveFailed 2002
MethodArgMismatch 2002
MethodClassMissing 2002
MethodNameMissing 2002
MethodNonexistent 2003
MethodNotFound 2003
MirrorError 2003
MismatchColumnlistCount 2003
MismatchedParameterFormat 2003
MismatchInsertColumnCount 2004
MismatchIntoParamCount 2004
MismatchParamResult 2004
MissingCheckpointFile 2004
MissingLogFile 2004
MissingMasterDataset 2005
MissingReplacestorerow 2005
MissingResolver 2005
MoreThanOneRow 2005
MultipleRowsAffected 2005
MustBeSingleColumn 2006
NameNotUnique 2006
NeedALogdir 2006
NeedLocateStartOption 2006
NeedProcedureprovider 2006
NeedQueryprovider 2007
NeedsRecalc 2007
NeedStoragedataset 2007
NegativeSqrtArgument 2007
NegativeSubstringLength 2007
NestedAggregators 2008
NewerVersion 2008
NewerVersionForOperation 2008
NoCalcAggFields 2008
NoCalcFields 2008
NoDatabaseToResolve 2009
NodefaultForBinary 2009
NoGroupingColumn 2009
NonExistentRowid 2009
NoNonBlobColumns 2009
NonPrimitiveRequired 2010
NonTransactional 2010
NoPrimaryKey 2010
NoPriorOriginalRow 2010
NoReadonly 2010
NoResultSet 2011
NoResultsetProduced 2011
NoRowsAffected 2011
NoRowsToDelete 2011
NoServerFound 2011
NoSuchParameter 2012
NotAnOutputParameter 2012
NotDatabaseResolver 2012

NotImplemented 2012
NotInsertRow 2012
NotSelectQuery 2013
NotTransactional 2013
NotUpdateable 2013
NoUpdatableColumns 2013
NoWhereClause 2013
NowReadonly 2014
NullColumnName 2014
NumericOverflow 2014
NumericUnderflow 2014
OlderVersion 2014
OldLog 2015
OnepassInputStream 2015
OperationCanceled 2015
OutParameterRequired 2015
OverloadedDiffParams 2015
ParameterCountMismatch 2016
ParameterValueNotSet 2016
PartialSearchForString 2016
PhysUndoNeeded 2016
PrivilegeMix 2016
ProcedureFailed 2017
ProcedureInProcess 2017
ProtocolVersion 2017
ProviderFailed 2017
ProviderOwned 2017
QueryFailed 2018
QueryInProcess 2018
ReadBlockError 2018
ReadOnly 2018
ReadOnlyColumn 2018
ReadOnlyDataset 2019
ReadOnlyStore 2019
ReadonlyTimeout 2019
ReadonlytxStartFailed 2019
RecoveryDenied 2019
RefreshrowNotSupported 2020
RenamedTwice 2020
ReopenFailure 2020
ResolveFailed 2020
ResolveInProgress 2020
ResolvePartial 2021
RestructureDataLoss 2021
RestructureInProgress 2021
RestructureParseError 2021
RestructurePrecisionLoss 2021
ResultNotUpdateable 2022
ResultsetClosed 2022
RevokeErrorDependentPrivilege 2022
RevokeErrorDependentView 2022
RevokeErrorForeignKey 2022
RowLockTimeout 2023
RowNotFound 2023
RowTooWide 2023
RuntimeError 2023
SecurityError 2023
SetCalculatedFailure 2024
SingleRowHasMoreRows 2024
SqlError 2024
StatementClosed 2024
StoreNameNotSet 2024
StoreOperationUnsupported 2025
StreamClosed 2025
StreamNotFound 2025
StreamOpenTwice 2025
StringTooLong 2025
SyntaxError 2026
TableAlreadyExists 2026
TableNonexistent 2026
TooManyAutoIncrements 2026
TooManyErrors 2026
TooManyOpenFiles 2027
Toomanyspecs 2027
TransactionIsolationLevelNotSupported 2027
TxAlreadyStarted 2027
TxManagerInuse 2027
TxNotAllowed 2028

TxRequired 2028
TypenameExpected 2028
UnclosedComment 2028
UnclosedIdentifier 2028
UnclosedStringLiteral 2029
UnexpectedEndOfQuery 2029
UnknownColumn 2029
UnknownColumnName 2029
UnknownDetailName 2029
UnknownParameter 2030
UnknownParamName 2030
UnrecognizedDataType 2030
UnrecognizedEscape 2030
UnsupportedColumntype 2030
UnsupportedDbxType 2031
UpdateFailed 2031
UpdateNotAllowed 2031
UrlNotFound 2031
UrlNotFoundInDesign 2031
ViewNotUpdatable 2032
WithCheckOptionViolation 2032
WriteBlockError 2032
WrongCreateschemaname 2032
WrongDatabase 2032
WrongSchemaname 2033
WrongTableType 2033
XaActive 2033
DataStoreParameter class 2033
about DataStoreParameter class 2033
Clone 2036
DataStoreParameter 2035
DataStoreParameter constructor 2034
DataStoreParameter Members 2034
DataStoreParameter methods 2036
DataStoreParameter properties 2036
DbType 2037
Direction 2037
IsNullable 2037
JDSType 2038
ParameterName 2038
Precision 2038
ResetDbType 2036
Scale 2039
Size 2039
SourceColumn 2039
SourceColumnNullMapping 2040
SourceVersion 2040
Value 2041
WasPrepared 2041
DataStoreParameterCollection class 2041
about DataStoreParameterCollection class 2041
Add 2044
Add method 2044
AddRange 2045
Clear 2045
Contains 2046
Contains method 2045
CopyTo 2046
Count 2051
DataStoreParameterCollection 2043
DataStoreParameterCollection Members 2042
DataStoreParameterCollection methods 2043
DataStoreParameterCollection properties 2051
GetEnumerator 2046
GetParameter 2047
GetParameter method 2047
IndexOf 2048
IndexOf method 2047
Insert 2048
InvalidatePreparation 2048
IsFixedSize 2051
IsReadOnly 2052
IsSynchronized 2052
Remove 2049
RemoveAt 2049
RemoveAt method 2049
SetParameter 2050
SetParameter method 2050

SyncRoot 2052
this 2050, 2051
this indexer 2050
DataStoreRowUpdatedEventArgs class 2053
about DataStoreRowUpdatedEventArgs class 2053
Command 2054
DataStoreRowUpdatedEventArgs 2054
DataStoreRowUpdatedEventArgs Members 2053
DataStoreRowUpdatedEventArgs properties 2054
DataStoreRowUpdatingEventArgs class 2054
about DataStoreRowUpdatingEventArgs class 2054
Command 2055
DataStoreRowUpdatingEventArgs 2055
DataStoreRowUpdatingEventArgs Members 2055
DataStoreRowUpdatingEventArgs properties 2055
DataStoreTransaction class 2056
about DataStoreTransaction class 2056
Commit 2057
Connection 2058
DataStoreTransaction 2056
DataStoreTransaction Members 2056
DataStoreTransaction methods 2057
DataStoreTransaction properties 2058
DbConnection 2058
Dispose 2057
Dispose method 2057
IsolationLevel 2058
Rollback 2058
DataTypesColumns class 2140
about DataTypesColumns class 2140
ColumnSize 2143
CreateFormat 2144
CreateParameters 2144
DataType 2144
DataTypesColumns fields 2143
DataTypesColumns Members 2142
DbxDataType 2144
IsAutoIncrementable 2144
IsBestMatch 2145
IsCaseSensitive 2145
IsConcurrencyType 2145
IsFixedLength 2145
IsFixedPrecisionScale 2145
IsLiteralSupported 2146
IsLong 2146
IsNullable 2146
IsSearchable 2146
IsSearchableWithLike 2146
IsUnicode 2147
IsUnsigned 2147
LiteralPrefix 2147
LiteralSuffix 2147
MaximumScale 2147
MaximumVersion 2148
MinimumScale 2148
MinimumVersion 2148
ProviderDbType 2148
TypeName 2148
Date enumeration member 22260
DateTime enumeration member 22260
DateUtils 6605
DateUtils namespace 6605
Constants 6718
Functions 6610
Variables 6717
DateUtils.ApproxDaysPerMonth 6717
DateUtils.ApproxDaysPerMonth variable 6717
DateUtils.ApproxDaysPerYear 6717
DateUtils.ApproxDaysPerYear variable 6717
DateUtils.CompareDate 6614
DateUtils.CompareDate function 6614
DateUtils.CompareDateTime 6614
DateUtils.CompareDateTime function 6614
DateUtils.CompareTime 6615
DateUtils.CompareTime function 6615
DateUtils.DateOf 6616
DateUtils.DateOf function 6616
DateUtils.DateTimeToJulianDate 6616

DateUtils.DateTimeToJulianDate function 6616
DateUtils.DateTimeToModifiedJulianDate 6617
DateUtils.DateTimeToModifiedJulianDate function 6617
DateUtils.DateTimeToUnix 6617
DateUtils.DateTimeToUnix function 6617
DateUtils.DayFriday 6718
DateUtils.DayFriday constant 6718
DateUtils.DayMonday 6718
DateUtils.DayMonday constant 6718
DateUtils.DayOf 6618
DateUtils.DayOf function 6618
DateUtils.DayOfTheMonth 6618
DateUtils.DayOfTheMonth function 6618
DateUtils.DayOfTheWeek 6619
DateUtils.DayOfTheWeek function 6619
DateUtils.DayOfTheYear 6620
DateUtils.DayOfTheYear function 6620
DateUtils.DaySaturday 6719
DateUtils.DaySaturday constant 6719
DateUtils.DaysBetween 6621
DateUtils.DaysBetween function 6621
DateUtils.DaysInAMonth 6622
DateUtils.DaysInAMonth function 6622
DateUtils.DaysInAYear 6622
DateUtils.DaysInAYear function 6622
DateUtils.DaysInMonth 6623
DateUtils.DaysInMonth function 6623
DateUtils.DaysInYear 6623
DateUtils.DaysInYear function 6623
DateUtils.DaySpan 6624
DateUtils.DaySpan function 6624
DateUtils.DaysPerWeek 6719
DateUtils.DaysPerWeek constant 6719
DateUtils.DaysPerYear 6719
DateUtils.DaysPerYear constant 6719
DateUtils.DaySunday 6720
DateUtils.DaySunday constant 6720
DateUtils.DayThursday 6720
DateUtils.DayThursday constant 6720
DateUtils.DayTuesday 6720
DateUtils.DayTuesday constant 6720
DateUtils.DayWednesday 6721
DateUtils.DayWednesday constant 6721
DateUtils.DecodeDateDay 6624
DateUtils.DecodeDateDay function 6624
DateUtils.DecodeDateMonthWeek 6625
DateUtils.DecodeDateMonthWeek function 6625
DateUtils.DecodeDateTime 6626
DateUtils.DecodeDateTime function 6626
DateUtils.DecodeDateWeek 6627
DateUtils.DecodeDateWeek function 6627
DateUtils.DecodeDayOfWeekInMonth 6628
DateUtils.DecodeDayOfWeekInMonth function 6628
DateUtils.EncodeDateDay 6629
DateUtils.EncodeDateDay function 6629
DateUtils.EncodeDateMonthWeek 6629
DateUtils.EncodeDateMonthWeek function 6629
DateUtils.EncodeDateTime 6630
DateUtils.EncodeDateTime function 6630
DateUtils.EncodeDateWeek 6631
DateUtils.EncodeDateWeek function 6631
DateUtils.EncodeDayOfWeekInMonth 6632
DateUtils.EncodeDayOfWeekInMonth function 6632
DateUtils.EndOfADay 6633
DateUtils.EndOfADay function 6633
DateUtils.EndOfAMonth 6634
DateUtils.EndOfAMonth function 6634
DateUtils.EndOfAWeek 6635
DateUtils.EndOfAWeek function 6635
DateUtils.EndOfAYear 6636
DateUtils.EndOfAYear function 6636
DateUtils.EndOfTheDay 6636
DateUtils.EndOfTheDay function 6636
DateUtils.EndOfTheMonth 6637
DateUtils.EndOfTheMonth function 6637
DateUtils.EndOfTheWeek 6637
DateUtils.EndOfTheWeek function 6637
DateUtils.EndOfTheYear 6638

DateUtils.EndOfTheYear function 6638
DateUtils.HourOf 6638
DateUtils.HourOf function 6638
DateUtils.HourOfTheDay 6639
DateUtils.HourOfTheDay function 6639
DateUtils.HourOfTheMonth 6640
DateUtils.HourOfTheMonth function 6640
DateUtils.HourOfTheWeek 6641
DateUtils.HourOfTheWeek function 6641
DateUtils.HourOfTheYear 6641
DateUtils.HourOfTheYear function 6641
DateUtils.HoursBetween 6642
DateUtils.HoursBetween function 6642
DateUtils.HourSpan 6643
DateUtils.HourSpan function 6643
DateUtils.IncDay 6643
DateUtils.IncDay function 6643
DateUtils.IncHour 6644
DateUtils.IncHour function 6644
DateUtils.IncMilliSecond 6645
DateUtils.IncMilliSecond function 6645
DateUtils.IncMinute 6645
DateUtils.IncMinute function 6645
DateUtils.IncSecond 6646
DateUtils.IncSecond function 6646
DateUtils.IncWeek 6647
DateUtils.IncWeek function 6647
DateUtils.IncYear 6647
DateUtils.IncYear function 6647
DateUtils.InvalidDateDayError 6648
DateUtils.InvalidDateDayError function 6648
DateUtils.InvalidDateMonthWeekError 6648
DateUtils.InvalidDateMonthWeekError function 6648
DateUtils.InvalidDateTimeError 6649
DateUtils.InvalidDateTimeError function 6649
DateUtils.InvalidDateWeekError 6649
DateUtils.InvalidDateWeekError function 6649
DateUtils.InvalidDayOfWeekInMonthError 6650
DateUtils.InvalidDayOfWeekInMonthError function 6650
DateUtils.IsInLeapYear 6650
DateUtils.IsInLeapYear function 6650
DateUtils.IsPM 6650
DateUtils.IsPM function 6650
DateUtils.IsSameDay 6651
DateUtils.IsSameDay function 6651
DateUtils.IsToday 6651
DateUtils.IsToday function 6651
DateUtils.IsValidDate 6652
DateUtils.IsValidDate function 6652
DateUtils.IsValidDateDay 6652
DateUtils.IsValidDateDay function 6652
DateUtils.IsValidDateMonthWeek 6653
DateUtils.IsValidDateMonthWeek function 6653
DateUtils.IsValidDateTime 6654
DateUtils.IsValidDateTime function 6654
DateUtils.IsValidDateWeek 6654
DateUtils.IsValidDateWeek function 6654
DateUtils.IsValidTime 6655
DateUtils.IsValidTime function 6655
DateUtils.JulianDateToDateTime 6656
DateUtils.JulianDateToDateTime function 6656
DateUtils.MilliSecondOf 6656
DateUtils.MilliSecondOf function 6656
DateUtils.MilliSecondOfTheDay 6657
DateUtils.MilliSecondOfTheDay function 6657
DateUtils.MilliSecondOfTheHour 6658
DateUtils.MilliSecondOfTheHour function 6658
DateUtils.MilliSecondOfTheMinute 6659
DateUtils.MilliSecondOfTheMinute function 6659
DateUtils.MilliSecondOfTheMonth 6659
DateUtils.MilliSecondOfTheMonth function 6659
DateUtils.MilliSecondOfTheSecond 6660
DateUtils.MilliSecondOfTheSecond function 6660
DateUtils.MilliSecondOfTheWeek 6661
DateUtils.MilliSecondOfTheWeek function 6661
DateUtils.MilliSecondOfTheYear 6661
DateUtils.MilliSecondOfTheYear function 6661
DateUtils.MilliSecondsBetween 6662

DateUtils.MilliSecondsBetween function 6662
DateUtils.MilliSecondSpan 6663
DateUtils.MilliSecondSpan function 6663
DateUtils.MinuteOf 6663
DateUtils.MinuteOf function 6663
DateUtils.MinuteOfTheDay 6664
DateUtils.MinuteOfTheDay function 6664
DateUtils.MinuteOfTheHour 6665
DateUtils.MinuteOfTheHour function 6665
DateUtils.MinuteOfTheMonth 6666
DateUtils.MinuteOfTheMonth function 6666
DateUtils.MinuteOfTheWeek 6666
DateUtils.MinuteOfTheWeek function 6666
DateUtils.MinuteOfTheYear 6667
DateUtils.MinuteOfTheYear function 6667
DateUtils.MinutesBetween 6668
DateUtils.MinutesBetween function 6668
DateUtils.MinuteSpan 6668
DateUtils.MinuteSpan function 6668
DateUtils.ModifiedJulianDateToDateTime 6669
DateUtils.ModifiedJulianDateToDateTime function 6669
DateUtils.MonthOf 6670
DateUtils.MonthOf function 6670
DateUtils.MonthOfTheYear 6670
DateUtils.MonthOfTheYear function 6670
DateUtils.MonthsBetween 6671
DateUtils.MonthsBetween function 6671
DateUtils.MonthSpan 6672
DateUtils.MonthSpan function 6672
DateUtils.MonthsPerYear 6721
DateUtils.MonthsPerYear constant 6721
DateUtils.NthDayOfWeek 6672
DateUtils.NthDayOfWeek function 6672
DateUtils.OneHour 6721
DateUtils.OneHour constant 6721
DateUtils.OneMillisecond 6722
DateUtils.OneMillisecond constant 6722
DateUtils.OneMinute 6722
DateUtils.OneMinute constant 6722
DateUtils.OneSecond 6722
DateUtils.OneSecond constant 6722
DateUtils.RecodeDate 6673
DateUtils.RecodeDate function 6673
DateUtils.RecodeDateTime 6674
DateUtils.RecodeDateTime function 6674
DateUtils.RecodeDay 6675
DateUtils.RecodeDay function 6675
DateUtils.RecodeHour 6675
DateUtils.RecodeHour function 6675
DateUtils.RecodeLeaveFieldAsIs 6723
DateUtils.RecodeLeaveFieldAsIs constant 6723
DateUtils.RecodeMilliSecond 6676
DateUtils.RecodeMilliSecond function 6676
DateUtils.RecodeMinute 6677
DateUtils.RecodeMinute function 6677
DateUtils.RecodeMonth 6678
DateUtils.RecodeMonth function 6678
DateUtils.RecodeSecond 6678
DateUtils.RecodeSecond function 6678
DateUtils.RecodeTime 6679
DateUtils.RecodeTime function 6679
DateUtils.RecodeYear 6680
DateUtils.RecodeYear function 6680
DateUtils.SameDate 6681
DateUtils.SameDate function 6681
DateUtils.SameDateTime 6681
DateUtils.SameDateTime function 6681
DateUtils.SameTime 6682
DateUtils.SameTime function 6682
DateUtils.SecondOf 6682
DateUtils.SecondOf function 6682
DateUtils.SecondOfTheDay 6683
DateUtils.SecondOfTheDay function 6683
DateUtils.SecondOfTheHour 6684
DateUtils.SecondOfTheHour function 6684
DateUtils.SecondOfTheMinute 6685
DateUtils.SecondOfTheMinute function 6685
DateUtils.SecondOfTheMonth 6685

DateUtils.SecondOfTheMonth function 6685
DateUtils.SecondOfTheWeek 6686
DateUtils.SecondOfTheWeek function 6686
DateUtils.SecondOfTheYear 6687
DateUtils.SecondOfTheYear function 6687
DateUtils.SecondsBetween 6687
DateUtils.SecondsBetween function 6687
DateUtils.SecondSpan 6688
DateUtils.SecondSpan function 6688
DateUtils.StartOfADay 6689
DateUtils.StartOfADay function 6689
DateUtils.StartOfAMonth 6690
DateUtils.StartOfAMonth function 6690
DateUtils.StartOfAWeek 6690
DateUtils.StartOfAWeek function 6690
DateUtils.StartOfAYear 6691
DateUtils.StartOfAYear function 6691
DateUtils.StartOfTheDay 6692
DateUtils.StartOfTheDay function 6692
DateUtils.StartOfTheMonth 6692
DateUtils.StartOfTheMonth function 6692
DateUtils.StartOfTheWeek 6693
DateUtils.StartOfTheWeek function 6693
DateUtils.StartOfTheYear 6693
DateUtils.StartOfTheYear function 6693
DateUtils.TimeOf 6694
DateUtils.TimeOf function 6694
DateUtils.Today 6694
DateUtils.Today function 6694
DateUtils.Tomorrow 6695
DateUtils.Tomorrow function 6695
DateUtils.TryEncodeDateDay 6695
DateUtils.TryEncodeDateDay function 6695
DateUtils.TryEncodeDateMonthWeek 6696
DateUtils.TryEncodeDateMonthWeek function 6696
DateUtils.TryEncodeDateTime 6697
DateUtils.TryEncodeDateTime function 6697
DateUtils.TryEncodeDateWeek 6698
DateUtils.TryEncodeDateWeek function 6698
DateUtils.TryEncodeDayOfWeekInMonth 6699
DateUtils.TryEncodeDayOfWeekInMonth function 6699
DateUtils.TryJulianDateToDateTime 6700
DateUtils.TryJulianDateToDateTime function 6700
DateUtils.TryModifiedJulianDateToDateTime 6701
DateUtils.TryModifiedJulianDateToDateTime function 6701
DateUtils.TryRecodeDateTime 6702
DateUtils.TryRecodeDateTime function 6702
DateUtils.UnixToDateTime 6703
DateUtils.UnixToDateTime function 6703
DateUtils.WeekOf 6703
DateUtils.WeekOf function 6703
DateUtils.WeekOfTheMonth 6704
DateUtils.WeekOfTheMonth function 6704
DateUtils.WeekOfTheYear 6705
DateUtils.WeekOfTheYear function 6705
DateUtils.WeeksBetween 6706
DateUtils.WeeksBetween function 6706
DateUtils.WeeksInAYear 6706
DateUtils.WeeksInAYear function 6706
DateUtils.WeeksInYear 6707
DateUtils.WeeksInYear function 6707
DateUtils.WeekSpan 6708
DateUtils.WeekSpan function 6708
DateUtils.WeeksPerFortnight 6723
DateUtils.WeeksPerFortnight constant 6723
DateUtils.WithinPastDays 6708
DateUtils.WithinPastDays function 6708
DateUtils.WithinPastHours 6709
DateUtils.WithinPastHours function 6709
DateUtils.WithinPastMilliSeconds 6710
DateUtils.WithinPastMilliSeconds function 6710
DateUtils.WithinPastMinutes 6710
DateUtils.WithinPastMinutes function 6710
DateUtils.WithinPastMonths 6711
DateUtils.WithinPastMonths function 6711
DateUtils.WithinPastSeconds 6712
DateUtils.WithinPastSeconds function 6712
DateUtils.WithinPastWeeks 6712

DateUtils.WithinPastWeeks function 6712
DateUtils.WithinPastYears 6713
DateUtils.WithinPastYears function 6713
DateUtils.YearOf 6714
DateUtils.YearOf function 6714
DateUtils.YearsBetween 6714
DateUtils.YearsBetween function 6714
DateUtils.YearSpan 6715
DateUtils.YearSpan function 6715
DateUtils.YearsPerCentury 6723
DateUtils.YearsPerCentury constant 6723
DateUtils.YearsPerDecade 6724
DateUtils.YearsPerDecade constant 6724
DateUtils.YearsPerMillennium 6724
DateUtils.YearsPerMillennium constant 6724
DateUtils.Yesterday 6716
DateUtils.Yesterday function 6716
DB 6725
DB namespace 6725
- Classes 6730
- Constants 7645
- Functions 7599
- Interfaces 7569
- Structs, Records, Enums 7604
- Types 7625
- Variables 7641

DB.BuffersEqual 7599
DB.BuffersEqual function 7599
DB.DatabaseError 7599
DB.DatabaseError function 7599
DB.DatabaseErrorFmt 7600
DB.DatabaseErrorFmt function 7600
DB.DBApplication 7641
DB.DBApplication variable 7641
DB.DBScreen 7641
DB.DBScreen variable 7641
DB.DefaultFieldClasses 7642
DB.DefaultFieldClasses variable 7642
DB.DisposeMem 7600
DB.DisposeMem function 7600
DB.dsEditModes 7645
DB.dsEditModes constant 7645
DB.dsMaxStringSize 7645
DB.dsMaxStringSize constant 7645
DB.dsWriteModes 7646
DB.dsWriteModes constant 7646
DB.EDatabaseError 6731
DB.EUpdateError 6732
DB.EUpdateError.Context 6734
DB.EUpdateError.Create 6733
DB.EUpdateError.Destroy 6734
DB.EUpdateError.ErrorCode 6735
DB.EUpdateError.OriginalException 6735
DB.EUpdateError.PreviousError 6736
DB.ExtractFieldName 7601
DB.ExtractFieldName function 7601
DB.FieldTypeNames 7646
DB.FieldTypeNames constant 7646
DB.FieldTypeVarMap 7646
DB.FieldTypeVarMap constant 7646
DB.ftFixedSizeTypes 7647
DB.ftFixedSizeTypes constant 7647
DB.ftNonTextTypes 7647
DB.ftNonTextTypes constant 7647
DB.GetFieldProperty 7602
DB.GetFieldProperty function 7602
DB.IDBApplication 7569
DB.IDBApplication.GetTitle 7570
DB.IDBApplication.ProcessMessages 7570
DB.IDBApplication.SetTitle 7570
DB.IDBApplication.Title 7571
DB.IDBScreen 7571
DB.IDBScreen.Cursor 7573
DB.IDBScreen.GetCursor 7572
DB.IDBScreen.SetCursor 7572
DB.IDBSession 7573
DB.IDBSession.AddPassword 7574
DB.IDBSession.RemoveAllPasswords 7575

DB.IDBSession.RemovePassword 7575
DB.IProviderSupport 7576
DB.IProviderSupport.PSEndTransaction 7578
DB.IProviderSupport.PSExecute 7578
DB.IProviderSupport.PSExecuteStatement 7579
DB.IProviderSupport.PSGetAttributes 7579
DB.IProviderSupport.PSGetCommandText 7579
DB.IProviderSupport.PSGetCommandType 7580
DB.IProviderSupport.PSGetDefaultOrder 7580
DB.IProviderSupport.PSGetIndexDefs 7581
DB.IProviderSupport.PSGetKeyFields 7581
DB.IProviderSupport.PSGetParams 7582
DB.IProviderSupport.PSGetQuoteChar 7582
DB.IProviderSupport.PSGetTableName 7583
DB.IProviderSupport.PSGetUpdateException 7583
DB.IProviderSupport.PSInTransaction 7583
DB.IProviderSupport.PSIsSQLBased 7584
DB.IProviderSupport.PSIsSQLSupported 7584
DB.IProviderSupport.PSReset 7585
DB.IProviderSupport.PSSetCommandText 7585
DB.IProviderSupport.PSSetParams 7586
DB.IProviderSupport.PSStartTransaction 7586
DB.IProviderSupport.PSUpdateRecord 7586
DB.IProviderSupport2 7587
DB.IProviderSupport2.PSEndTransaction 7589
DB.IProviderSupport2.PSExecute 7590
DB.IProviderSupport2.PSExecuteStatement 7590
DB.IProviderSupport2.PSGetAttributes 7591
DB.IProviderSupport2.PSGetCommandText 7591
DB.IProviderSupport2.PSGetCommandType 7591
DB.IProviderSupport2.PSGetDefaultOrder 7592
DB.IProviderSupport2.PSGetIndexDefs 7592
DB.IProviderSupport2.PSGetKeyFields 7593
DB.IProviderSupport2.PSGetParams 7593
DB.IProviderSupport2.PSGetQuoteChar 7594
DB.IProviderSupport2.PSGetTableName 7594
DB.IProviderSupport2.PSGetUpdateException 7594
DB.IProviderSupport2.PSInTransaction 7595
DB.IProviderSupport2.PSIsSQLBased 7595
DB.IProviderSupport2.PSIsSQLSupported 7596
DB.IProviderSupport2.PSReset 7596
DB.IProviderSupport2.PSSetCommandText 7596
DB.IProviderSupport2.PSSetParams 7597
DB.IProviderSupport2.PSStartTransaction 7597
DB.IProviderSupport2.PSUpdateRecord 7598
DB.Largeint 7626
DB.Largeint type 7626
DB.LoginDialogExProc 7642
DB.LoginDialogExProc variable 7642
DB.LoginDialogProc 7642
DB.LoginDialogProc variable 7642
DB.ObjectFieldTypes 7648
DB.ObjectFieldTypes constant 7648
DB.PasswordDialog 7643
DB.PasswordDialog variable 7643
DB.PBookmarkFlag 7626
DB.PBookmarkFlag type 7626
DB.PLookupListEntry 7626
DB.PLookupListEntry type 7626
DB.PPacketAttribute 7627
DB.PPacketAttribute type 7627
DB.RegisterFields 7602
DB.RegisterFields function 7602
DB.RegisterFieldsProc 7648
DB.RegisterFieldsProc constant 7648
DB.RemoteLoginDialogProc 7643
DB.RemoteLoginDialogProc variable 7643
DB.ScreenCursorProc 7644
DB.ScreenCursorProc variable 7644
DB.TADTField 6736
DB.TADTField.Create 6742
DB.TADTField.GetSize 6743
DB.TAggregateField 6743
DB.TAggregateField.Active 6752
DB.TAggregateField.Create 6749
DB.TAggregateField.currency 6752
DB.TAggregateField.DisplayFormat 6753
DB.TAggregateField.DisplayName 6756

DB.TAggregateField.Expression 6756
DB.TAggregateField.FieldKind 6757
DB.TAggregateField.GetAsString 6750
DB.TAggregateField.GetAsVariant 6750
DB.TAggregateField.GetText 6751
DB.TAggregateField.GroupingLevel 6758
DB.TAggregateField.Handle 6758
DB.TAggregateField.IndexName 6758
DB.TAggregateField.Precision 6759
DB.TAggregateField.Reset 6751
DB.TAggregateField.ResultType 6759
DB.TAggregateField.SetDisplayFormat 6751
DB.TAggregateField.Visible 6760
DB.TArrayField 6760
DB.TArrayField.Bind 6767
DB.TArrayField.Create 6766
DB.TArrayField.SetSize 6767
DB.TArrayField.Size 6768
DB.TAutoIncField 6768
DB.TAutoIncField.Create 6775
DB.TAutoRefreshFlag 7604
DB.TAutoRefreshFlag enumeration 7604
DB.TBCDField 6776
DB.TBCDField.CheckTypeSize 6784
DB.TBCDField.CopyData 6784
DB.TBCDField.Create 6782
DB.TBCDField.currency 6788
DB.TBCDField.GetAsBCD 6784
DB.TBCDField.GetAsCurrency 6784
DB.TBCDField.GetAsFloat 6785
DB.TBCDField.GetAsInteger 6785
DB.TBCDField.GetAsString 6785
DB.TBCDField.GetAsVariant 6785
DB.TBCDField.GetDataSize 6785
DB.TBCDField.GetDefaultWidth 6786
DB.TBCDField.GetText 6786
DB.TBCDField.GetValue 6786
DB.TBCDField.MaxValue 6788
DB.TBCDField.MinValue 6789
DB.TBCDField.Precision 6789
DB.TBCDField.SetAsBCD 6786
DB.TBCDField.SetAsCurrency 6787
DB.TBCDField.SetAsFloat 6787
DB.TBCDField.SetAsInteger 6787
DB.TBCDField.SetAsString 6787
DB.TBCDField.SetVarValue 6787
DB.TBCDField.Size 6789
DB.TBCDField.Value 6790
DB.TBinaryField 6791
DB.TBinaryField.CheckTypeSize 6798
DB.TBinaryField.CopyData 6798
DB.TBinaryField.Create 6797
DB.TBinaryField.GetAsString 6799
DB.TBinaryField.GetAsVariant 6799
DB.TBinaryField.GetText 6799
DB.TBinaryField.SetAsString 6799
DB.TBinaryField.SetText 6799
DB.TBinaryField.SetVarValue 6800
DB.TBinaryField.Size 6800
DB.TBlobByteData 7627
DB.TBlobByteData type 7627
DB.TBlobData 7627
DB.TBlobData type 7627
DB.TBlobField 6801
DB.TBlobField.Assign 6809
DB.TBlobField.AssignTo 6810
DB.TBlobField.BlobSize 6819
DB.TBlobField.BlobType 6819
DB.TBlobField.Clear 6810
DB.TBlobField.Create 6808
DB.TBlobField.FreeBuffers 6811
DB.TBlobField.GetAsString 6811
DB.TBlobField.GetAsVariant 6811
DB.TBlobField.GetAsWideString 6812
DB.TBlobField.GetBlobSize 6812
DB.TBlobField.GetClassDesc 6812
DB.TBlobField.GetIsNull 6812
DB.TBlobField.GetText 6812

DB.TBlobField.GraphicHeader 6820
DB.TBlobField.IsBlob 6813
DB.TBlobField.LoadFromFile 6814
DB.TBlobField.LoadFromStream 6815
DB.TBlobField.Modified 6820
DB.TBlobField.SaveToFile 6816
DB.TBlobField.SaveToStream 6816
DB.TBlobField.SetAsString 6817
DB.TBlobField.SetAsWideString 6817
DB.TBlobField.SetFieldType 6817
DB.TBlobField.SetText 6818
DB.TBlobField.SetVarValue 6818
DB.TBlobField.Size 6820
DB.TBlobField.Transliterate 6821
DB.TBlobField.Value 6822
DB.TBlobStreamMode 7605
DB.TBlobStreamMode enumeration 7605
DB.TBlobType 7628
DB.TBlobType type 7628
DB.TBookmark 7628
DB.TBookmark type 7628
DB.TBookmarkFlag 7606
DB.TBookmarkFlag enumeration 7606
DB.TBookmarkStr 7629
DB.TBookmarkStr type 7629
DB.TBooleanField 6822
DB.TBooleanField.Create 6828
DB.TBooleanField.DisplayValues 6832
DB.TBooleanField.GetAsBoolean 6830
DB.TBooleanField.GetAsString 6830
DB.TBooleanField.GetAsVariant 6830
DB.TBooleanField.GetDataSize 6830
DB.TBooleanField.GetDefaultWidth 6831
DB.TBooleanField.SetAsBoolean 6831
DB.TBooleanField.SetAsString 6831
DB.TBooleanField.SetVarValue 6831
DB.TBooleanField.Value 6833
DB.TBufferList 7629
DB.TBufferList type 7629
DB.TBytesField 6833
DB.TBytesField.Create 6840
DB.TBytesField.GetDataSize 6841
DB.TCheckConstraint 6841
DB.TCheckConstraint.Assign 6843
DB.TCheckConstraint.CustomConstraint 6844
DB.TCheckConstraint.ErrorMessage 6845
DB.TCheckConstraint.FromDictionary 6845
DB.TCheckConstraint.GetDisplayName 6843
DB.TCheckConstraint.ImportedConstraint 6845
DB.TCheckConstraints 6846
DB.TCheckConstraints.Add 6848
DB.TCheckConstraints.Create 6848
DB.TCheckConstraints.GetOwner 6849
DB.TCheckConstraints.Items 6849
DB.TConnectChangeEvent 7630
DB.TConnectChangeEvent type 7630
DB.TCurrencyField 6850
DB.TCurrencyField.Create 6856
DB.TCurrencyField.currency 6857
DB.TCustomConnection 6858
DB.TCustomConnection.AfterConnect 6874
DB.TCustomConnection.AfterDisconnect 6875
DB.TCustomConnection.BeforeConnect 6875
DB.TCustomConnection.BeforeDisconnect 6875
DB.TCustomConnection.Close 6863
DB.TCustomConnection.Connected 6868
DB.TCustomConnection.Create 6862
DB.TCustomConnection.DataSetCount 6869
DB.TCustomConnection.DataSets 6870
DB.TCustomConnection.Destroy 6863
DB.TCustomConnection.DoConnect 6864
DB.TCustomConnection.DoDisconnect 6864
DB.TCustomConnection.GetConnected 6865
DB.TCustomConnection.GetDataSet 6865
DB.TCustomConnection.GetDataSetCount 6865
DB.TCustomConnection.Loaded 6866
DB.TCustomConnection.LoginPrompt 6872
DB.TCustomConnection.OnLogin 6873

DB.TCustomConnection.Open 6866
DB.TCustomConnection.RegisterClient 6866
DB.TCustomConnection.SendConnectEvent 6867
DB.TCustomConnection.SetConnected 6867
DB.TCustomConnection.StreamedConnected 6874
DB.TCustomConnection.UnRegisterClient 6868
DB.TDataAction 7606
DB.TDataAction enumeration 7606
DB.TDataChangeEvent 7630
DB.TDataChangeEvent type 7630
DB.TDataEvent 7607
DB.TDataEvent enumeration 7607
DB.TDataLink 6876
DB.TDataLink.Active 6888
DB.TDataLink.ActiveChanged 6880
DB.TDataLink.ActiveRecord 6889
DB.TDataLink.BOF 6889
DB.TDataLink.BufferCount 6890
DB.TDataLink.CheckBrowseMode 6880
DB.TDataLink.Create 6878
DB.TDataLink.DataEvent 6881
DB.TDataLink.DataSet 6890
DB.TDataLink.DataSetChanged 6881
DB.TDataLink.DataSetScrolled 6882
DB.TDataLink.DataSource 6891
DB.TDataLink.DataSourceFixed 6891
DB.TDataLink.Destroy 6879
DB.TDataLink.Edit 6882
DB.TDataLink.Editing 6892
DB.TDataLink.EditingChanged 6883
DB.TDataLink.Eof 6892
DB.TDataLink.ExecuteAction 6883
DB.TDataLink.FocusControl 6884
DB.TDataLink.GetActiveRecord 6884
DB.TDataLink.GetBOF 6884
DB.TDataLink.GetBufferCount 6884
DB.TDataLink.GetEOF 6885
DB.TDataLink.GetRecordCount 6885
DB.TDataLink.LayoutChanged 6885
DB.TDataLink.MoveBy 6885
DB.TDataLink.ReadOnly 6893
DB.TDataLink.RecordChanged 6886
DB.TDataLink.RecordCount 6893
DB.TDataLink.SetActiveRecord 6886
DB.TDataLink.SetBufferCount 6887
DB.TDataLink.UpdateAction 6887
DB.TDataLink.UpdateData 6887
DB.TDataLink.UpdateRecord 6888
DB.TDataLink.VisualControl 6894
DB.TDataOperation 7630
DB.TDataOperation type 7630
DB.TDataSet 6894
DB.TDataSet.ActivateBuffers 6909
DB.TDataSet.Active 7004
DB.TDataSet.ActiveBuffer 6909
DB.TDataSet.ActiveRecord 7005
DB.TDataSet.AfterCancel 7038
DB.TDataSet.AfterClose 7040
DB.TDataSet.AfterDelete 7041
DB.TDataSet.AfterEdit 7044
DB.TDataSet.AfterInsert 7044
DB.TDataSet.AfterOpen 7045
DB.TDataSet.AfterPost 7046
DB.TDataSet.AfterRefresh 7047
DB.TDataSet.AfterScroll 7047
DB.TDataSet.AggFields 7006
DB.TDataSet.AllocRecordBuffer 6909
DB.TDataSet.Append 6910
DB.TDataSet.AppendRecord 6911
DB.TDataSet.AutoCalcFields 7006
DB.TDataSet.BeforeCancel 7048
DB.TDataSet.BeforeClose 7048
DB.TDataSet.BeforeDelete 7049
DB.TDataSet.BeforeEdit 7049
DB.TDataSet.BeforeInsert 7050
DB.TDataSet.BeforeOpen 7051
DB.TDataSet.BeforePost 7052
DB.TDataSet.BeforeRefresh 7052

DB.TDataSet.BeforeScroll 7053
DB.TDataSet.BindFields 6915
DB.TDataSet.BlobFieldCount 7007
DB.TDataSet.BlockReadNext 6915
DB.TDataSet.BlockReadSize 7007
DB.TDataSet.Bof 7008
DB.TDataSet.Bookmark 7009
DB.TDataSet.BookmarkAvailable 6916
DB.TDataSet.BookmarkSize 7010
DB.TDataSet.BookmarkValid 6916
DB.TDataSet.BufferCount 7010
DB.TDataSet.Buffers 7011
DB.TDataSet.CalcBuffer 7011
DB.TDataSet.CalcFieldsSize 7012
DB.TDataSet.CalculateFields 6916
DB.TDataSet.Cancel 6916
DB.TDataSet.CanModify 7012
DB.TDataSet.CheckActive 6917
DB.TDataSet.CheckBiDirectional 6918
DB.TDataSet.CheckBrowseMode 6918
DB.TDataSet.CheckFieldCompatibility 6919
DB.TDataSet.CheckInactive 6919
DB.TDataSet.ClearBuffers 6919
DB.TDataSet.ClearCalcFields 6919
DB.TDataSet.ClearFields 6919
DB.TDataSet.Close 6920
DB.TDataSet.CloseBlob 6921
DB.TDataSet.CloseCursor 6921
DB.TDataSet.CompareBookmarks 6921
DB.TDataSet.Constraints 7013
DB.TDataSet.ControlsDisabled 6921
DB.TDataSet.CopyFields 6922
DB.TDataSet.Create 6904
DB.TDataSet.CreateBlobStream 6922
DB.TDataSet.CreateFields 6923
DB.TDataSet.CreateNestedDataSet 6923
DB.TDataSet.CurrentRecord 7013
DB.TDataSet.CursorPosChanged 6923
DB.TDataSet.DataConvert 6924
DB.TDataSet.DataEvent 6924
DB.TDataSet.DataSetField 7013
DB.TDataSet.DataSource 7014
DB.TDataSet.DefaultFields 7014
DB.TDataSet.DefChanged 6924
DB.TDataSet.Delete 6924
DB.TDataSet.Designer 7015
DB.TDataSet.Destroy 6904
DB.TDataSet.DestroyFields 6925
DB.TDataSet.DisableControls 6925
DB.TDataSet.DoAfterCancel 6927
DB.TDataSet.DoAfterClose 6927
DB.TDataSet.DoAfterDelete 6928
DB.TDataSet.DoAfterEdit 6928
DB.TDataSet.DoAfterInsert 6928
DB.TDataSet.DoAfterOpen 6928
DB.TDataSet.DoAfterPost 6929
DB.TDataSet.DoAfterRefresh 6929
DB.TDataSet.DoAfterScroll 6929
DB.TDataSet.DoBeforeCancel 6929
DB.TDataSet.DoBeforeClose 6929
DB.TDataSet.DoBeforeDelete 6930
DB.TDataSet.DoBeforeEdit 6930
DB.TDataSet.DoBeforeInsert 6930
DB.TDataSet.DoBeforeOpen 6930
DB.TDataSet.DoBeforePost 6931
DB.TDataSet.DoBeforeRefresh 6931
DB.TDataSet.DoBeforeScroll 6931
DB.TDataSet.DoOnCalcFields 6931
DB.TDataSet.DoOnNewRecord 6931
DB.TDataSet.Edit 6932
DB.TDataSet.EnableControls 6934
DB.TDataSet.Eof 7015
DB.TDataSet.FieldByName 6936
DB.TDataSet.FieldByNumber 6938
DB.TDataSet.FieldCount 7017
DB.TDataSet.FieldDefList 7019
DB.TDataSet.FieldDefs 7019
DB.TDataSet.FieldList 7022

DB.TDataSet.FieldNoOfs 7022
DB.TDataSet.Fields 7023
DB.TDataSet.FieldValues 7025
DB.TDataSet.Filter 7026
DB.TDataSet.Filtered 7027
DB.TDataSet.FilterOptions 7027
DB.TDataSet.FindField 6938
DB.TDataSet.FindFirst 6939
DB.TDataSet.FindLast 6940
DB.TDataSet.FindNext 6940
DB.TDataSet.FindPrior 6941
DB.TDataSet.FindRecord 6942
DB.TDataSet.First 6942
DB.TDataSet.Found 7028
DB.TDataSet.FreeBookmark 6945
DB.TDataSet.FreeFieldBuffers 6946
DB.TDataSet.FreeRecordBuffer 6947
DB.TDataSet.GetAggRecordCount 6947
DB.TDataSet.GetAggregateValue 6947
DB.TDataSet.GetBlobFieldData 6947
DB.TDataSet.GetBookmark 6948
DB.TDataSet.GetBookmarkData 6949
DB.TDataSet.GetBookmarkFlag 6949
DB.TDataSet.GetBookmarkStr 6949
DB.TDataSet.GetCalcFields 6950
DB.TDataSet.GetCanModify 6950
DB.TDataSet.GetChildren 6950
DB.TDataSet.GetCurrentRecord 6950
DB.TDataSet.GetDataSource 6951
DB.TDataSet.GetDetailDataSets 6951
DB.TDataSet.GetDetailLinkFields 6951
DB.TDataSet.GetFieldClass 6952
DB.TDataSet.GetFieldData 6952
DB.TDataSet.GetFieldFullName 6953
DB.TDataSet.GetFieldList 6953
DB.TDataSet.GetFieldNames 6954
DB.TDataSet.GetIndexDefs 6954
DB.TDataSet.GetIsIndexField 6954
DB.TDataSet.GetNextRecord 6955
DB.TDataSet.GetNextRecords 6955
DB.TDataSet.GetPriorRecord 6955
DB.TDataSet.GetPriorRecords 6955
DB.TDataSet.GetRecNo 6956
DB.TDataSet.GetRecord 6956
DB.TDataSet.GetRecordCount 6956
DB.TDataSet.GetRecordSize 6956
DB.TDataSet.GetStateFieldValue 6956
DB.TDataSet.GotoBookmark 6957
DB.TDataSet.InitFieldDefs 6958
DB.TDataSet.InitFieldDefsFromFields 6959
DB.TDataSet.InitRecord 6959
DB.TDataSet.Insert 6959
DB.TDataSet.InsertRecord 6961
DB.TDataSet.InternalAddRecord 6963
DB.TDataSet.InternalCalcFields 7028
DB.TDataSet.InternalCancel 6963
DB.TDataSet.InternalClose 6963
DB.TDataSet.InternalDelete 6964
DB.TDataSet.InternalEdit 6964
DB.TDataSet.InternalFirst 6964
DB.TDataSet.InternalGotoBookmark 6964
DB.TDataSet.InternalHandleException 6964
DB.TDataSet.InternalInitFieldDefs 6965
DB.TDataSet.InternalInitRecord 6965
DB.TDataSet.InternalInsert 6965
DB.TDataSet.InternalLast 6965
DB.TDataSet.InternalOpen 6966
DB.TDataSet.InternalPost 6966
DB.TDataSet.InternalRefresh 6966
DB.TDataSet.InternalSetToRecord 6966
DB.TDataSet.IsCursorOpen 6966
DB.TDataSet.IsEmpty 6967
DB.TDataSet.IsLinkedTo 6967
DB.TDataSet.IsSequenced 6967
DB.TDataSet.IsUniDirectional 7029
DB.TDataSet.Last 6968
DB.TDataSet.Loaded 6969
DB.TDataSet.Locate 6970

DB.TDataSet.Lookup 6970
DB.TDataSet.Modified 7029
DB.TDataSet.MoveBy 6971
DB.TDataSet.NestedDataSetClass 7030
DB.TDataSet.NestedDataSets 7030
DB.TDataSet.Next 6973
DB.TDataSet.ObjectView 7031
DB.TDataSet.OnCalcFields 7053
DB.TDataSet.OnDeleteError 7054
DB.TDataSet.OnEditError 7054
DB.TDataSet.OnFilterRecord 7055
DB.TDataSet.OnNewRecord 7056
DB.TDataSet.OnPostError 7056
DB.TDataSet.Open 6976
DB.TDataSet.OpenCursor 6976
DB.TDataSet.OpenCursorComplete 6977
DB.TDataSet.OpenParentDataSet 6977
DB.TDataSet.Post 6977
DB.TDataSet.Prior 6979
DB.TDataSet.PSEndTransaction 6979
DB.TDataSet.PSExecute 6980
DB.TDataSet.PSExecuteStatement 6981
DB.TDataSet.PSGetAttributes 6981
DB.TDataSet.PSGetCommandText 6982
DB.TDataSet.PSGetCommandTextW 6982
DB.TDataSet.PSGetCommandType 6982
DB.TDataSet.PSGetDefaultOrder 6983
DB.TDataSet.PSGetIndexDefs 6983
DB.TDataSet.PSGetKeyFields 6984
DB.TDataSet.PSGetKeyFieldsW 6984
DB.TDataSet.PSGetParams 6984
DB.TDataSet.PSGetQuoteChar 6985
DB.TDataSet.PSGetQuoteCharW 6985
DB.TDataSet.PSGetTableName 6985
DB.TDataSet.PSGetTableNameW 6986
DB.TDataSet.PSGetUpdateException 6986
DB.TDataSet.PSInTransaction 6987
DB.TDataSet.PSIsSQLBased 6987
DB.TDataSet.PSIsSQLSupported 6987
DB.TDataSet.PSReset 6988
DB.TDataSet.PSSetCommandText 6988, 6989
DB.TDataSet.PSSetParams 6989
DB.TDataSet.PSStartTransaction 6989
DB.TDataSet.PSUpdateRecord 6990
DB.TDataSet.RecNo 7031
DB.TDataSet.RecordCount 7032
DB.TDataSet.RecordSize 7035
DB.TDataSet.Refresh 6991
DB.TDataSet.RefreshInternalCalcFields 6993
DB.TDataSet.Reserved 7035
DB.TDataSet.ResetAggField 6993
DB.TDataSet.RestoreState 6993
DB.TDataSet.Resync 6994
DB.TDataSet.SetActive 6994
DB.TDataSet.SetBlockReadSize 6994
DB.TDataSet.SetBookmarkData 6994
DB.TDataSet.SetBookmarkFlag 6995
DB.TDataSet.SetBookmarkStr 6995
DB.TDataSet.SetBufListSize 6995
DB.TDataSet.SetChildOrder 6995
DB.TDataSet.SetCurrentRecord 6996
DB.TDataSet.SetDataSetField 6996
DB.TDataSet.SetDefaultFields 6996
DB.TDataSet.SetFieldData 6996, 6997
DB.TDataSet.SetFields 6997
DB.TDataSet.SetFiltered 6997
DB.TDataSet.SetFilterOptions 6998
DB.TDataSet.SetFilterText 6998
DB.TDataSet.SetFound 6998
DB.TDataSet.SetModified 6998
DB.TDataSet.SetName 6999
DB.TDataSet.SetObjectView 6999
DB.TDataSet.SetOnFilterRecord 6999
DB.TDataSet.SetRecNo 6999
DB.TDataSet.SetState 7000
DB.TDataSet.SetStateFieldValue 7000
DB.TDataSet.SetTempState 7000
DB.TDataSet.SetUniDirectional 7000

DB.TDataSet.SparseArrays 7036
DB.TDataSet.State 7036
DB.TDataSet.TempBuffer 7001
DB.TDataSet.Translate 7001
DB.TDataSet.UpdateBufferCount 7001
DB.TDataSet.UpdateCursorPos 7001
DB.TDataSet.UpdateIndexDefs 7002
DB.TDataSet.UpdateRecord 7002
DB.TDataSet.UpdateStatus 7002
DB.TDataSetClass 7631
DB.TDataSetClass type 7631
DB.TDataSetDesigner 7056
DB.TDataSetDesigner.BeginDesign 7059
DB.TDataSetDesigner.Create 7058
DB.TDataSetDesigner.DataEvent 7059
DB.TDataSetDesigner.DataSet 7060
DB.TDataSetDesigner.Destroy 7058
DB.TDataSetDesigner.EndDesign 7059
DB.TDataSetErrorEvent 7631
DB.TDataSetErrorEvent type 7631
DB.TDataSetField 7060
DB.TDataSetField.Assign 7068
DB.TDataSetField.Bind 7069
DB.TDataSetField.Create 7067
DB.TDataSetField.Destroy 7067
DB.TDataSetField.GetCanModify 7069
DB.TDataSetField.GetFields 7069
DB.TDataSetField.IncludeObjectField 7069
DB.TDataSetField.NestedDataSet 7070
DB.TDataSetNotifyEvent 7631
DB.TDataSetNotifyEvent type 7631
DB.TDataSetState 7608
DB.TDataSetState enumeration 7608
DB.TDataSource 7070
DB.TDataSource.AutoEdit 7077
DB.TDataSource.Create 7075
DB.TDataSource.DataLinks 7078
DB.TDataSource.DataSet 7078
DB.TDataSource.Destroy 7075
DB.TDataSource.Edit 7076
DB.TDataSource.Enabled 7079
DB.TDataSource.IsLinkedTo 7076
DB.TDataSource.OnDataChange 7080
DB.TDataSource.OnStateChange 7082
DB.TDataSource.OnUpdateData 7082
DB.TDataSource.State 7080
DB.TDateField 7083
DB.TDateField.Create 7089
DB.TDateField.GetDataSize 7090
DB.TDateTimeAlias 7632
DB.TDateTimeAlias type 7632
DB.TDateTimeField 7090
DB.TDateTimeField.CopyData 7098
DB.TDateTimeField.Create 7097
DB.TDateTimeField.DisplayFormat 7101
DB.TDateTimeField.EditMask 7102
DB.TDateTimeField.GetAsDateTime 7098
DB.TDateTimeField.GetAsFloat 7099
DB.TDateTimeField.GetAsString 7099
DB.TDateTimeField.GetAsVariant 7099
DB.TDateTimeField.GetDataSize 7099
DB.TDateTimeField.GetDefaultWidth 7100
DB.TDateTimeField.GetText 7100
DB.TDateTimeField.SetAsDateTime 7100
DB.TDateTimeField.SetAsFloat 7100
DB.TDateTimeField.SetAsString 7100
DB.TDateTimeField.SetVarValue 7101
DB.TDateTimeField.Value 7103
DB.TDateTimeRec 7610
DB.TDateTimeRec record 7610
DB.TDBScreenCursor 7610
DB.TDBScreenCursor enumeration 7610
DB.TDefCollection 7104
DB.TDefCollection.Create 7106
DB.TDefCollection.DataSet 7110
DB.TDefCollection.DoUpdate 7107
DB.TDefCollection.Find 7107
DB.TDefCollection.GetItemNames 7108

DB.TDefCollection.IndexOf 7108
DB.TDefCollection.OnUpdate 7111
DB.TDefCollection.SetItemName 7109
DB.TDefCollection.Update 7109
DB.TDefCollection.Updated 7110
DB.TDefCollection.UpdateDefs 7109
DB.TDefUpdateMethod 7632
DB.TDefUpdateMethod type 7632
DB.TDetailDataLink 7111
DB.TDetailDataLink.DetailDataSet 7114
DB.TDetailDataLink.GetDetailDataSet 7114
DB.TField 7115
DB.TField.AccessError 7124
DB.TField.Alignment 7146
DB.TField.AsBCD 7147
DB.TField.AsBoolean 7147
DB.TField.AsCurrency 7148
DB.TField.AsDateTime 7148
DB.TField.AsFloat 7149
DB.TField.AsInteger 7149
DB.TField.Assign 7124
DB.TField.AssignValue 7124
DB.TField.AsSQLTimeStamp 7150
DB.TField.AsString 7151
DB.TField.AsVariant 7152
DB.TField.AsWideString 7152
DB.TField.AttributeSet 7153
DB.TField.AutoGenerateValue 7153
DB.TField.Bind 7126
DB.TField.Calculated 7154
DB.TField.CanModify 7154
DB.TField.Change 7126
DB.TField.CheckInactive 7126
DB.TField.CheckTypeSize 7126
DB.TField.Clear 7126
DB.TField.ConstraintErrorMessage 7155
DB.TField.CopyData 7127
DB.TField.Create 7121
DB.TField.CurValue 7155
DB.TField.CustomConstraint 7156
DB.TField.DataChanged 7127
DB.TField.DataSet 7157
DB.TField.DataSize 7157
DB.TField.DataType 7159
DB.TField.DefaultExpression 7159
DB.TField.DefineProperties 7127
DB.TField.Destroy 7122
DB.TField.DisplayLabel 7160
DB.TField.DisplayName 7160
DB.TField.DisplayText 7162
DB.TField.DisplayWidth 7162
DB.TField.EditMask 7163
DB.TField.EditMaskPtr 7163
DB.TField.FieldKind 7164
DB.TField.FieldName 7164
DB.TField.FieldNo 7166
DB.TField.FocusControl 7128
DB.TField.FreeBuffers 7128
DB.TField.FullName 7166
DB.TField.GetAsBCD 7128
DB.TField.GetAsBoolean 7129
DB.TField.GetAsByteArray 7129
DB.TField.GetAsCurrency 7129
DB.TField.GetAsDateTime 7129
DB.TField.GetAsFloat 7129
DB.TField.GetAsInteger 7130
DB.TField.GetAsSQLTimeStamp 7130
DB.TField.GetAsString 7130
DB.TField.GetAsVariant 7130
DB.TField.GetAsWideString 7131
DB.TField.GetCanModify 7131
DB.TField.GetClassDesc 7131
DB.TField.GetData 7131
DB.TField.GetDataSize 7133
DB.TField.GetDefaultWidth 7133
DB.TField.GetFieldNo 7134
DB.TField.GetHasConstraints 7134
DB.TField.GetIsNull 7134

DB.TField.GetParentComponent 7134
DB.TField.GetSize 7135
DB.TField.GetText 7135
DB.TField.GetWideText 7135
DB.TField.HasConstraints 7167
DB.TField.HasParent 7135
DB.TField.ImportedConstraint 7167
DB.TField.Index 7168
DB.TField.IsBlob 7136
DB.TField.IsIndexField 7168
DB.TField.IsNull 7169
DB.TField.IsValidChar 7136
DB.TField.KeyFields 7169
DB.TField.Lookup 7170
DB.TField.LookupCache 7170
DB.TField.LookupDataSet 7171
DB.TField.LookupKeyFields 7172
DB.TField.LookupList 7172
DB.TField.LookupResultField 7173
DB.TField.NewValue 7174
DB.TField.Notification 7137
DB.TField.Offset 7175
DB.TField.OldValue 7175
DB.TField.OnChange 7181
DB.TField.OnGetText 7182
DB.TField.OnSetText 7182
DB.TField.OnValidate 7183
DB.TField.Origin 7176
DB.TField.ParentField 7176
DB.TField.PropertyChanged 7138
DB.TField.ProviderFlags 7176
DB.TField.ReadOnly 7177
DB.TField.ReadState 7138
DB.TField.RefreshLookupList 7138
DB.TField.Required 7177
DB.TField.SetAsBCD 7139
DB.TField.SetAsBoolean 7139
DB.TField.SetAsByteArray 7139
DB.TField.SetAsCurrency 7139
DB.TField.SetAsDateTime 7139
DB.TField.SetAsFloat 7140
DB.TField.SetAsInteger 7140
DB.TField.SetAsSQLTimeStamp 7140
DB.TField.SetAsString 7140
DB.TField.SetAsVariant 7141
DB.TField.SetAsWideString 7141
DB.TField.SetAutoGenerateValue 7141
DB.TField.SetData 7141
DB.TField.SetDataSet 7142
DB.TField.SetDataType 7142
DB.TField.SetFieldKind 7142
DB.TField.SetFieldType 7142
DB.TField.SetParentComponent 7143
DB.TField.SetParentField 7143
DB.TField.SetSize 7143
DB.TField.SetText 7144
DB.TField.SetVarValue 7144
DB.TField.SetWideText 7144
DB.TField.Size 7178
DB.TField.Text 7179
DB.TField.Validate 7144
DB.TField.ValidChars 7179
DB.TField.Value 7180
DB.TField.Visible 7181
DB.TFieldAttribute 7611
DB.TFieldAttribute enumeration 7611
DB.TFieldAttributes 7632
DB.TFieldAttributes type 7632
DB.TFieldChars 7633
DB.TFieldChars type 7633
DB.TFieldClass 7633
DB.TFieldClass type 7633
DB.TFieldDef 7184
DB.TFieldDef.AddChild 7187
DB.TFieldDef.Assign 7188
DB.TFieldDef.Attributes 7190
DB.TFieldDef.ChildDefs 7191
DB.TFieldDef.Create 7186

DB.TFieldDef.CreateField 7188
DB.TFieldDef.DataType 7191
DB.TFieldDef.DefineProperties 7189
DB.TFieldDef.Destroy 7187
DB.TFieldDef.FieldClass 7192
DB.TFieldDef.FieldNo 7192
DB.TFieldDef.HasChildDefs 7189
DB.TFieldDef.InternalCalcField 7192
DB.TFieldDef.ParentDef 7193
DB.TFieldDef.Precision 7193
DB.TFieldDef.Required 7194
DB.TFieldDef.Size 7194
DB.TFieldDefList 7195
DB.TFieldDefList.FieldByName 7196
DB.TFieldDefList.FieldDefs 7198
DB.TFieldDefList.Find 7197
DB.TFieldDefList.GetUpdated 7197
DB.TFieldDefList.UpdateList 7197
DB.TFieldDefs 7198
DB.TFieldDefs.Add 7202
DB.TFieldDefs.AddFieldDef 7202
DB.TFieldDefs.ChildDefUpdate 7204
DB.TFieldDefs.Create 7201
DB.TFieldDefs.FieldDefUpdate 7204
DB.TFieldDefs.Find 7204
DB.TFieldDefs.HiddenFields 7206
DB.TFieldDefs.Items 7206
DB.TFieldDefs.ParentDef 7207
DB.TFieldDefs.SetItemName 7205
DB.TFieldDefs.Update 7205
DB.TFieldGetTextEvent 7633
DB.TFieldGetTextEvent type 7633
DB.TFieldKind 7611
DB.TFieldKind enumeration 7611
DB.TFieldKinds 7634
DB.TFieldKinds type 7634
DB.TFieldList 7207
DB.TFieldList.FieldByName 7209
DB.TFieldList.Fields 7210
DB.TFieldList.Find 7209
DB.TFieldList.UpdateList 7210
DB.TFieldNotifyEvent 7634
DB.TFieldNotifyEvent type 7634
DB.TFieldRef 7634
DB.TFieldRef type 7634
DB.TFields 7211
DB.TFields.Add 7214
DB.TFields.Changed 7214
DB.TFields.CheckFieldKind 7215
DB.TFields.CheckFieldName 7215
DB.TFields.CheckFieldNames 7215
DB.TFields.Clear 7216
DB.TFields.Count 7220
DB.TFields.Create 7213
DB.TFields.DataSet 7221
DB.TFields.Destroy 7213
DB.TFields.FieldByName 7216
DB.TFields.FieldByNumber 7217
DB.TFields.Fields 7221
DB.TFields.FindField 7217
DB.TFields.GetCount 7217
DB.TFields.GetEnumerator 7218
DB.TFields.GetField 7218
DB.TFields.GetFieldNames 7218
DB.TFields.IndexOf 7219
DB.TFields.OnChange 7222
DB.TFields.Remove 7219
DB.TFields.SetField 7220
DB.TFields.SetFieldIndex 7220
DB.TFields.SparseFields 7221
DB.TFields.ValidFieldKinds 7222
DB.TFieldsEnumerator 7222
DB.TFieldsEnumerator.Create 7223
DB.TFieldsEnumerator.Current 7224
DB.TFieldsEnumerator.GetCurrent 7224
DB.TFieldsEnumerator.MoveNext 7224
DB.TFieldSetTextEvent 7635
DB.TFieldSetTextEvent type 7635

DB.TFieldType 7612
DB.TFieldType enumeration 7612
DB.TFilterOption 7615
DB.TFilterOption enumeration 7615
DB.TFilterOptions 7635
DB.TFilterOptions type 7635
DB.TFilterRecordEvent 7635
DB.TFilterRecordEvent type 7635
DB.TFlatList 7225
DB.TFlatList.Create 7226
DB.TFlatList.DataSet 7228
DB.TFlatList.FindItem 7227
DB.TFlatList.GetCount 7227
DB.TFlatList.GetUpdated 7227
DB.TFlatList.ListChanging 7227
DB.TFlatList.Locked 7229
DB.TFlatList.Update 7228
DB.TFlatList.Updated 7229
DB.TFlatList.UpdateList 7228
DB.TFloatField 7229
DB.TFloatField.Create 7236
DB.TFloatField.currency 7240
DB.TFloatField.GetAsFloat 7237
DB.TFloatField.GetAsInteger 7237
DB.TFloatField.GetAsString 7238
DB.TFloatField.GetAsVariant 7238
DB.TFloatField.GetDataSize 7238
DB.TFloatField.GetText 7238
DB.TFloatField.MaxValue 7240
DB.TFloatField.MinValue 7241
DB.TFloatField.Precision 7241
DB.TFloatField.SetAsFloat 7239
DB.TFloatField.SetAsInteger 7239
DB.TFloatField.SetAsString 7239
DB.TFloatField.SetVarValue 7239
DB.TFloatField.Value 7242
DB.TFMTBCDField 7242
DB.TFMTBCDField.CheckTypeSize 7250
DB.TFMTBCDField.CopyData 7250
DB.TFMTBCDField.Create 7248
DB.TFMTBCDField.currency 7254
DB.TFMTBCDField.GetAsBCD 7250
DB.TFMTBCDField.GetAsCurrency 7251
DB.TFMTBCDField.GetAsFloat 7251
DB.TFMTBCDField.GetAsInteger 7251
DB.TFMTBCDField.GetAsString 7251
DB.TFMTBCDField.GetAsVariant 7252
DB.TFMTBCDField.GetDataSize 7252
DB.TFMTBCDField.GetDefaultWidth 7252
DB.TFMTBCDField.GetText 7252
DB.TFMTBCDField.GetValue 7252
DB.TFMTBCDField.MaxValue 7255
DB.TFMTBCDField.MinValue 7255
DB.TFMTBCDField.Precision 7256
DB.TFMTBCDField.SetAsBCD 7253
DB.TFMTBCDField.SetAsCurrency 7253
DB.TFMTBCDField.SetAsFloat 7253
DB.TFMTBCDField.SetAsInteger 7253
DB.TFMTBCDField.SetAsString 7254
DB.TFMTBCDField.SetVarValue 7254
DB.TFMTBCDField.Size 7256
DB.TFMTBCDField.Value 7257
DB.TGetMode 7615
DB.TGetMode enumeration 7615
DB.TGetResult 7616
DB.TGetResult enumeration 7616
DB.TGraphicField 7257
DB.TGraphicField.Create 7264
DB.TGroupPosInd 7616
DB.TGroupPosInd enumeration 7616
DB.TGroupPosInds 7636
DB.TGroupPosInds type 7636
DB.TGuidField 7265
DB.TGuidField.AsGuid 7274
DB.TGuidField.CheckTypeSize 7273
DB.TGuidField.Create 7272
DB.TGuidField.GetAsGuid 7273
DB.TGuidField.GetDefaultWidth 7273

DB.TGuidField.SetAsGuid 7273
DB.TIDispatchField 7274
DB.TIDispatchField.Create 7280
DB.TIDispatchField.GetValue 7282
DB.TIDispatchField.SetValue 7282
DB.TIDispatchField.Value 7282
DB.TIndexDef 7283
DB.TIndexDef.Assign 7285
DB.TIndexDef.CaseInsFields 7287
DB.TIndexDef.Create 7285
DB.TIndexDef.DescFields 7288
DB.TIndexDef.Expression 7289
DB.TIndexDef.FieldExpression 7289
DB.TIndexDef.Fields 7290
DB.TIndexDef.GetDisplayName 7286
DB.TIndexDef.GroupingLevel 7291
DB.TIndexDef.Options 7291
DB.TIndexDef.Source 7292
DB.TIndexDefs 7292
DB.TIndexDefs.Add 7295
DB.TIndexDefs.AddIndexDef 7296
DB.TIndexDefs.Create 7295
DB.TIndexDefs.Find 7297
DB.TIndexDefs.FindIndexForFields 7298
DB.TIndexDefs.GetIndexForFields 7298
DB.TIndexDefs.Items 7300
DB.TIndexDefs.Update 7299
DB.TIndexOption 7617
DB.TIndexOption enumeration 7617
DB.TIndexOptions 7636
DB.TIndexOptions type 7636
DB.TIntegerField 7301
DB.TIntegerField.Create 7309
DB.TIntegerField.GetAsFloat 7310
DB.TIntegerField.GetAsInteger 7310
DB.TIntegerField.GetAsString 7311
DB.TIntegerField.GetAsVariant 7311
DB.TIntegerField.GetDataSize 7311
DB.TIntegerField.GetText 7311
DB.TIntegerField.GetValue 7312
DB.TIntegerField.MaxValue 7313
DB.TIntegerField.MinValue 7313
DB.TIntegerField.SetAsFloat 7312
DB.TIntegerField.SetAsInteger 7312
DB.TIntegerField.SetAsString 7312
DB.TIntegerField.SetVarValue 7312
DB.TIntegerField.Value 7314
DB.TInterfaceField 7314
DB.TInterfaceField.CheckTypeSize 7321
DB.TInterfaceField.Create 7320
DB.TInterfaceField.GetAsVariant 7321
DB.TInterfaceField.GetValue 7322
DB.TInterfaceField.SetValue 7322
DB.TInterfaceField.SetVarValue 7322
DB.TInterfaceField.Value 7322
DB.TLargeintField 7323
DB.TLargeintField.AsLargeInt 7334
DB.TLargeintField.Create 7329
DB.TLargeintField.GetAsFloat 7331
DB.TLargeintField.GetAsInteger 7331
DB.TLargeintField.GetAsLargeint 7331
DB.TLargeintField.GetAsString 7331
DB.TLargeintField.GetAsVariant 7331
DB.TLargeintField.GetDataSize 7332
DB.TLargeintField.GetDefaultWidth 7332
DB.TLargeintField.GetText 7332
DB.TLargeintField.GetValue 7332
DB.TLargeintField.MaxValue 7335
DB.TLargeintField.MinValue 7335
DB.TLargeintField.SetAsFloat 7333
DB.TLargeintField.SetAsInteger 7333
DB.TLargeintField.SetAsLargeint 7333
DB.TLargeintField.SetAsString 7333
DB.TLargeintField.SetVarValue 7333
DB.TLargeintField.Value 7335
DB.TLocateOption 7618
DB.TLocateOption enumeration 7618
DB.TLocateOptions 7637

DB.TLocateOptions type 7637
DB.TLookupList 7336
DB.TLookupList.Add 7338
DB.TLookupList.Clear 7339
DB.TLookupList.Create 7337
DB.TLookupList.Destroy 7338
DB.TLookupList.ValueOfKey 7339
DB.TLookupListEntry 7618
DB.TLookupListEntry record 7618
DB.TMasterDataLink 7339
DB.TMasterDataLink.ActiveChanged 7343
DB.TMasterDataLink.CheckBrowseMode 7344
DB.TMasterDataLink.Create 7342
DB.TMasterDataLink.Destroy 7342
DB.TMasterDataLink.FieldNames 7345
DB.TMasterDataLink.Fields 7346
DB.TMasterDataLink.GetDetailDataSet 7344
DB.TMasterDataLink.LayoutChanged 7344
DB.TMasterDataLink.OnMasterChange 7347
DB.TMasterDataLink.OnMasterDisable 7347
DB.TMasterDataLink.RecordChanged 7345
DB.TMemoField 7348
DB.TMemoField.Create 7354
DB.TMemoField.GetAsWideString 7355
DB.TMemoField.SetAsWideString 7356
DB.TMemoField.Transliterate 7356
DB.TNamedItem 7357
DB.TNamedItem.GetDisplayName 7359
DB.TNamedItem.Name 7359
DB.TNamedItem.SetDisplayName 7359
DB.TNumericField 7361
DB.TNumericField.Alignment 7369
DB.TNumericField.Create 7367
DB.TNumericField.DisplayFormat 7369
DB.TNumericField.EditFormat 7371
DB.TNumericField.RangeError 7368
DB.TNumericField.SetDisplayFormat 7368
DB.TNumericField.SetEditFormat 7368
DB.TObjectField 7372
DB.TObjectField.CheckTypeSize 7380
DB.TObjectField.Create 7378
DB.TObjectField.DefineProperties 7380
DB.TObjectField.Destroy 7379
DB.TObjectField.FieldCount 7384
DB.TObjectField.Fields 7384
DB.TObjectField.FieldValues 7385
DB.TObjectField.FreeBuffers 7380
DB.TObjectField.GetAsString 7381
DB.TObjectField.GetAsVariant 7381
DB.TObjectField.GetChildren 7381
DB.TObjectField.GetDefaultWidth 7381
DB.TObjectField.GetFieldCount 7381
DB.TObjectField.GetFields 7382
DB.TObjectField.GetFieldValue 7382
DB.TObjectField.GetHasConstraints 7382
DB.TObjectField.ObjectType 7385
DB.TObjectField.SetChildOrder 7382
DB.TObjectField.SetDataSet 7383
DB.TObjectField.SetFieldKind 7383
DB.TObjectField.SetFieldValue 7383
DB.TObjectField.SetParentField 7383
DB.TObjectField.SetUnNamed 7383
DB.TObjectField.SetVarValue 7384
DB.TObjectField.UnNamed 7386
DB.TPacketAttribute 7618
DB.TPacketAttribute record 7618
DB.TParam 7386
DB.TParam.AsBCD 7407
DB.TParam.AsBlob 7408
DB.TParam.AsBoolean 7409
DB.TParam.AsCurrency 7410
DB.TParam.AsDate 7412
DB.TParam.AsDateTime 7413
DB.TParam.AsFloat 7415
DB.TParam.AsFMTBCD 7416
DB.TParam.AsInteger 7417
DB.TParam.AsMemo 7419
DB.TParam.Assign 7391

DB.TParam.AssignField 7392
DB.TParam.AssignFieldValue 7393
DB.TParam.AssignParam 7393
DB.TParam.AssignTo 7394
DB.TParam.AsSmallInt 7419
DB.TParam.AsSQLTimeStamp 7421
DB.TParam.AsString 7421
DB.TParam.AsTime 7423
DB.TParam.AsWideString 7424
DB.TParam.AsWord 7425
DB.TParam.Bound 7426
DB.TParam.Clear 7394
DB.TParam.Create 7390
DB.TParam.DataSet 7427
DB.TParam.DataType 7427
DB.TParam.GetAsBCD 7395
DB.TParam.GetAsBoolean 7395
DB.TParam.GetAsCurrency 7395
DB.TParam.GetAsDateTime 7395
DB.TParam.GetAsFloat 7395
DB.TParam.GetAsFMTBCD 7396
DB.TParam.GetAsInteger 7396
DB.TParam.GetAsMemo 7396
DB.TParam.GetAsSQLTimeStamp 7396
DB.TParam.GetAsString 7397
DB.TParam.GetAsVariant 7397
DB.TParam.GetAsWideString 7397
DB.TParam.GetData 7397
DB.TParam.GetDataSize 7398
DB.TParam.GetDisplayName 7399
DB.TParam.GetIsNull 7400
DB.TParam.IsEqual 7400
DB.TParam.IsNull 7429
DB.TParam.LoadFromFile 7400
DB.TParam.LoadFromStream 7400
DB.TParam.Name 7430
DB.TParam.NativeStr 7431
DB.TParam.NumericScale 7431
DB.TParam.ParamType 7431
DB.TParam.Precision 7432
DB.TParam.SetAsBCD 7401
DB.TParam.SetAsBlob 7401
DB.TParam.SetAsBoolean 7401
DB.TParam.SetAsCurrency 7401
DB.TParam.SetAsDate 7402
DB.TParam.SetAsDateTime 7402
DB.TParam.SetAsFloat 7402
DB.TParam.SetAsFMTBCD 7402
DB.TParam.SetAsInteger 7403
DB.TParam.SetAsMemo 7403
DB.TParam.SetAsSmallInt 7403
DB.TParam.SetAsSQLTimeStamp 7403
DB.TParam.SetAsString 7403
DB.TParam.SetAsTime 7404
DB.TParam.SetAsVariant 7404
DB.TParam.SetAsWideString 7404
DB.TParam.SetAsWord 7404
DB.TParam.SetBlobData 7404
DB.TParam.SetData 7405
DB.TParam.SetDataType 7405
DB.TParam.SetText 7406
DB.TParam.Size 7432
DB.TParam.Text 7433
DB.TParam.Value 7433
DB.TParams 7434
DB.TParams.AddParam 7438
DB.TParams.AssignTo 7438
DB.TParams.AssignValues 7439
DB.TParams.Create 7437
DB.TParams.CreateParam 7441
DB.TParams.DefineProperties 7444
DB.TParams.FindParam 7445
DB.TParams.GetDataSet 7445
DB.TParams.GetOwner 7445
DB.TParams.GetParamList 7446
DB.TParams.IsEqual 7446
DB.TParams.Items 7449
DB.TParams.ParamByName 7447

DB.TParams.ParamValues 7451
DB.TParams.ParseSQL 7447
DB.TParams.RemoveParam 7448
DB.TParams.Update 7448
DB.TParamType 7619
DB.TParamType enumeration 7619
DB.TParamTypes 7637
DB.TParamTypes type 7637
DB.TProviderFlag 7620
DB.TProviderFlag enumeration 7620
DB.TProviderFlags 7638
DB.TProviderFlags type 7638
DB.TPSCommandType 7621
DB.TPSCommandType enumeration 7621
DB.TReferenceField 7452
DB.TReferenceField.Assign 7459
DB.TReferenceField.Create 7458
DB.TReferenceField.GetAsVariant 7459
DB.TReferenceField.GetDataSize 7460
DB.TReferenceField.ReferenceTableName 7460
DB.TReferenceField.SetVarValue 7460
DB.TReferenceField.Size 7460
DB.TResyncMode 7638
DB.TResyncMode type 7638
DB.TSmallintField 7461
DB.TSmallintField.Create 7468
DB.TSmallintField.GetDataSize 7469
DB.TSQLCommandType 7638
DB.TSQLCommandType type 7638
DB.TSQLTimeStampField 7469
DB.TSQLTimeStampField.CopyData 7477
DB.TSQLTimeStampField.Create 7475
DB.TSQLTimeStampField.DisplayFormat 7480
DB.TSQLTimeStampField.EditMask 7482
DB.TSQLTimeStampField.GetAsDateTime 7477
DB.TSQLTimeStampField.GetAsFloat 7477
DB.TSQLTimeStampField.GetAsSQLTimeStamp 7478
DB.TSQLTimeStampField.GetAsString 7478
DB.TSQLTimeStampField.GetAsVariant 7478
DB.TSQLTimeStampField.GetDataSize 7478
DB.TSQLTimeStampField.GetDefaultWidth 7478
DB.TSQLTimeStampField.GetText 7479
DB.TSQLTimeStampField.SetAsDateTime 7479
DB.TSQLTimeStampField.SetAsFloat 7479
DB.TSQLTimeStampField.SetAsSQLTimeStamp 7479
DB.TSQLTimeStampField.SetAsString 7480
DB.TSQLTimeStampField.SetVarValue 7480
DB.TSQLTimeStampField.Value 7482
DB.TStringField 7483
DB.TStringField.CheckTypeSize 7491
DB.TStringField.Create 7490
DB.TStringField.EditMask 7496
DB.TStringField.FixedChar 7497
DB.TStringField.GetAsBCD 7492
DB.TStringField.GetAsBoolean 7492
DB.TStringField.GetAsDateTime 7492
DB.TStringField.GetAsFloat 7492
DB.TStringField.GetAsInteger 7493
DB.TStringField.GetAsSQLTimeStamp 7493
DB.TStringField.GetAsString 7493
DB.TStringField.GetAsVariant 7493
DB.TStringField.GetDataSize 7493
DB.TStringField.GetDefaultWidth 7494
DB.TStringField.GetText 7494
DB.TStringField.GetValue 7494
DB.TStringField.SetAsBCD 7494
DB.TStringField.SetAsBoolean 7495
DB.TStringField.SetAsDateTime 7495
DB.TStringField.SetAsFloat 7495
DB.TStringField.SetAsInteger 7495
DB.TStringField.SetAsSQLTimeStamp 7495
DB.TStringField.SetAsString 7496
DB.TStringField.SetVarValue 7496
DB.TStringField.Size 7497
DB.TStringField.Transliterate 7498
DB.TStringField.Value 7499
DB.TTimeField 7499
DB.TTimeField.Create 7505

DB.TTimeField.GetDataSize 7507
DB.TUpdateAction 7621
DB.TUpdateAction enumeration 7621
DB.TUpdateErrorEvent 7639
DB.TUpdateErrorEvent type 7639
DB.TUpdateKind 7622
DB.TUpdateKind enumeration 7622
DB.TUpdateMode 7623
DB.TUpdateMode enumeration 7623
DB.TUpdateRecordEvent 7639
DB.TUpdateRecordEvent type 7639
DB.TUpdateRecordTypes 7639
DB.TUpdateRecordTypes type 7639
DB.TUpdateStatus 7623
DB.TUpdateStatus enumeration 7623
DB.TUpdateStatusSet 7640
DB.TUpdateStatusSet type 7640
DB.TVarBytesField 7507
DB.TVarBytesField.Create 7513
DB.TVarBytesField.GetDataSize 7514
DB.TVarBytesField.SetAsByteArray 7514
DB.TVariantField 7514
DB.TVariantField.CheckTypeSize 7522
DB.TVariantField.Create 7520
DB.TVariantField.GetAsBCD 7522
DB.TVariantField.GetAsBoolean 7522
DB.TVariantField.GetAsDateTime 7522
DB.TVariantField.GetAsFloat 7522
DB.TVariantField.GetAsInteger 7523
DB.TVariantField.GetAsSQLTimeStamp 7523
DB.TVariantField.GetAsString 7523
DB.TVariantField.GetAsVariant 7523
DB.TVariantField.GetDefaultWidth 7524
DB.TVariantField.SetAsBCD 7524
DB.TVariantField.SetAsBoolean 7524
DB.TVariantField.SetAsDateTime 7524
DB.TVariantField.SetAsFloat 7524
DB.TVariantField.SetAsInteger 7525
DB.TVariantField.SetAsSQLTimeStamp 7525
DB.TVariantField.SetAsString 7525
DB.TVariantField.SetVarValue 7525
DB.TWideDataSet 7526
DB.TWideDataSet.PSExecuteStatement 7536
DB.TWideDataSet.PSGetCommandText 7537
DB.TWideDataSet.PSGetCommandTextW 7537
DB.TWideDataSet.PSGetKeyFields 7538
DB.TWideDataSet.PSGetKeyFieldsW 7538
DB.TWideDataSet.PSGetQuoteChar 7538
DB.TWideDataSet.PSGetQuoteCharW 7539
DB.TWideDataSet.PSGetTableName 7539
DB.TWideDataSet.PSGetTableNameW 7539
DB.TWideDataSet.PSSetCommandText 7540
DB.TWideMemoField 7541
DB.TWideMemoField.Create 7547
DB.TWideMemoField.GetAsString 7548
DB.TWideMemoField.SetAsString 7548
DB.TWideMemoField.Value 7549
DB.TWideStringField 7549
DB.TWideStringField.CheckTypeSize 7557
DB.TWideStringField.CopyData 7557
DB.TWideStringField.Create 7555
DB.TWideStringField.GetAsString 7557
DB.TWideStringField.GetAsVariant 7558
DB.TWideStringField.GetAsWideString 7558
DB.TWideStringField.GetDataSize 7558
DB.TWideStringField.GetValue 7558
DB.TWideStringField.SetAsString 7558
DB.TWideStringField.SetAsWideString 7559
DB.TWideStringField.SetVarValue 7559
DB.TWideStringField.Value 7559
DB.TWordField 7560
DB.TWordField.Create 7566
DB.TWordField.GetDataSize 7568
DB.VarTypeToDataType 7603
DB.VarTypeToDataType function 7603
DBActns 7649
DBActns namespace 7649
Classes 7650

DBActns.TDataSetAction 7650
DBActns.TDataSetAction.DataSource 7657
DBActns.TDataSetAction.GetDataSet 7655
DBActns.TDataSetAction.HandlesTarget 7656
DBActns.TDataSetAction.Notification 7656
DBActns.TDataSetCancel 7657
DBActns.TDataSetCancel.DataSource 7663
DBActns.TDataSetCancel.ExecuteTarget 7662
DBActns.TDataSetCancel.UpdateTarget 7663
DBActns.TDataSetDelete 7664
DBActns.TDataSetDelete.DataSource 7670
DBActns.TDataSetDelete.ExecuteTarget 7669
DBActns.TDataSetDelete.UpdateTarget 7670
DBActns.TDataSetEdit 7671
DBActns.TDataSetEdit.DataSource 7677
DBActns.TDataSetEdit.ExecuteTarget 7676
DBActns.TDataSetEdit.UpdateTarget 7676
DBActns.TDataSetFirst 7677
DBActns.TDataSetFirst.DataSource 7684
DBActns.TDataSetFirst.ExecuteTarget 7683
DBActns.TDataSetFirst.UpdateTarget 7683
DBActns.TDataSetInsert 7684
DBActns.TDataSetInsert.DataSource 7690
DBActns.TDataSetInsert.ExecuteTarget 7689
DBActns.TDataSetInsert.UpdateTarget 7690
DBActns.TDataSetLast 7691
DBActns.TDataSetLast.DataSource 7697
DBActns.TDataSetLast.ExecuteTarget 7696
DBActns.TDataSetLast.UpdateTarget 7697
DBActns.TDataSetNext 7698
DBActns.TDataSetNext.DataSource 7704
DBActns.TDataSetNext.ExecuteTarget 7703
DBActns.TDataSetNext.UpdateTarget 7703
DBActns.TDataSetPost 7704
DBActns.TDataSetPost.DataSource 7711
DBActns.TDataSetPost.ExecuteTarget 7710
DBActns.TDataSetPost.UpdateTarget 7710
DBActns.TDataSetPrior 7711
DBActns.TDataSetPrior.DataSource 7717
DBActns.TDataSetPrior.ExecuteTarget 7716
DBActns.TDataSetPrior.UpdateTarget 7717
DBActns.TDataSetRefresh 7718
DBActns.TDataSetRefresh.DataSource 7724
DBActns.TDataSetRefresh.ExecuteTarget 7723
DBActns.TDataSetRefresh.UpdateTarget 7723
DBAdapt 7725
DBAdapt namespace 7725
- Classes 7728
- Constants 8117
- Functions 8110
- Interfaces 8102
- Structs, Records, Enums 8111
- Types 8113
- Variables 8116

DBAdapt.AddLocateParamsString 8110
DBAdapt.AddLocateParamsString function 8110
DBAdapt.AllActionParamOptions 8117
DBAdapt.AllActionParamOptions constant 8117
DBAdapt.AllImageParamOptions 8117
DBAdapt.AllImageParamOptions constant 8117
DBAdapt.DataSetAdapterImageFieldClass 8116
DBAdapt.DataSetAdapterImageFieldClass variable 8116
DBAdapt.DataSetAdapterModeNames 8117
DBAdapt.DataSetAdapterModeNames constant 8117
DBAdapt.IAdapterDataSetComponent 8102
DBAdapt.IAdapterDataSetComponent.DataSet 8103
DBAdapt.IAdapterDataSetComponent.GetDataSet 8103
DBAdapt.IAdapterDataSetComponent.SetDataSet 8103
DBAdapt.IDataSetAdapterFieldClass 8104
DBAdapt.IGetDataSetKeyName 8104
DBAdapt.IGetDataSetKeyName.GetKeyName 8105
DBAdapt.IGetDataSetKeyName.KeyName 8105
DBAdapt.IKeyParams 8105
DBAdapt.IKeyParams.Count 8106
DBAdapt.IKeyParams.GetName 8107
DBAdapt.IKeyParams.GetValue 8107
DBAdapt.IKeyParams.Names 8108
DBAdapt.IKeyParams.Values 8108

DBAdapt.IPrepareDataSet 8109
DBAdapt.IPrepareDataSet.PrepareDataSet 8109
DBAdapt.TBaseDataSetAdapterField 7729
DBAdapt.TBaseDataSetAdapterField.Adapter 7752
DBAdapt.TBaseDataSetAdapterField.CheckOrigValue 7736
DBAdapt.TBaseDataSetAdapterField.CheckValueChange 7737
DBAdapt.TBaseDataSetAdapterField.Create 7735
DBAdapt.TBaseDataSetAdapterField.DataSetField 7752
DBAdapt.TBaseDataSetAdapterField.FieldFlags 7752
DBAdapt.TBaseDataSetAdapterField.FieldModes 7753
DBAdapt.TBaseDataSetAdapterField.FindField 7737
DBAdapt.TBaseDataSetAdapterField.FullyQualifyInputName 7738
DBAdapt.TBaseDataSetAdapterField.GetAdapter 7738
DBAdapt.TBaseDataSetAdapterField.GetDataSet 7738
DBAdapt.TBaseDataSetAdapterField.GetDataSetFieldValue 7739
DBAdapt.TBaseDataSetAdapterField.GetDisplayStyle 7739
DBAdapt.TBaseDataSetAdapterField.GetDisplayStyleType 7740
DBAdapt.TBaseDataSetAdapterField.GetInputStyle 7741
DBAdapt.TBaseDataSetAdapterField.GetInputStyleType 7742
DBAdapt.TBaseDataSetAdapterField.GetKeyName 7742
DBAdapt.TBaseDataSetAdapterField.GetKeyParams 7743
DBAdapt.TBaseDataSetAdapterField.GetReadOnly 7743
DBAdapt.TBaseDataSetAdapterField.GetViewMode 7744
DBAdapt.TBaseDataSetAdapterField.GetViewModeType 7744
DBAdapt.TBaseDataSetAdapterField.ImplAsFormatted 7745
DBAdapt.TBaseDataSetAdapterField.ImplCheckOrigValue 7745
DBAdapt.TBaseDataSetAdapterField.ImplCheckValueChange 7746
DBAdapt.TBaseDataSetAdapterField.ImplGetDisplayLabel 7747
DBAdapt.TBaseDataSetAdapterField.ImplGetDisplayWidth 7747
DBAdapt.TBaseDataSetAdapterField.ImplGetRequired 7747
DBAdapt.TBaseDataSetAdapterField.ImplGetValue 7748
DBAdapt.TBaseDataSetAdapterField.ImplGetVisible 7748
DBAdapt.TBaseDataSetAdapterField.ImplSetFieldName 7749
DBAdapt.TBaseDataSetAdapterField.ImplUpdateValue 7749
DBAdapt.TBaseDataSetAdapterField.Notification 7750
DBAdapt.TBaseDataSetAdapterField.OnGetDisplayText 7754
DBAdapt.TBaseDataSetAdapterField.OnGetValue 7754
DBAdapt.TBaseDataSetAdapterField.PrepareDataSet 7750
DBAdapt.TBaseDataSetAdapterField.SetDataSetField 7751
DBAdapt.TBaseDataSetAdapterField.UpdateValue 7751
DBAdapt.TBaseDataSetAdapterImageField 7755
DBAdapt.TBaseDataSetAdapterInputField 7756
DBAdapt.TBaseDataSetAdapterInputField.ImplGetEchoActionFieldValue 7763
DBAdapt.TBaseDataSetAdapterInputField.ImplSetEchoActionFieldValue 7763
DBAdapt.TBaseDataSetAdapterInputField.ImplValidateValue 7764
DBAdapt.TBaseDataSetAdapterInputField.OnUpdateValue 7765
DBAdapt.TBaseDataSetAdapterInputField.OnValidateValue 7766
DBAdapt.TBaseDataSetAdapterInputField.TestValueChange 7764
DBAdapt.TBaseDataSetAdapterInputField.ValidateValue 7765
DBAdapt.TCustomDataSetAdapter 7766
DBAdapt.TCustomDataSetAdapter.AdapterRowNotFound 7807
DBAdapt.TCustomDataSetAdapter.AddDefaultActions 7778
DBAdapt.TCustomDataSetAdapter.AddDefaultFields 7779
DBAdapt.TCustomDataSetAdapter.ApplyAdapterMode 7780
DBAdapt.TCustomDataSetAdapter.Create 7775
DBAdapt.TCustomDataSetAdapter.DataSet 7807
DBAdapt.TCustomDataSetAdapter.DataSetPrepared 7808
DBAdapt.TCustomDataSetAdapter.Destroy 7776
DBAdapt.TCustomDataSetAdapter.DetailAdapters 7808
DBAdapt.TCustomDataSetAdapter.EncodeActionParams 7780
DBAdapt.TCustomDataSetAdapter.EncodeActionParamsFlags 7781
DBAdapt.TCustomDataSetAdapter.ExtractRequestParams 7781
DBAdapt.TCustomDataSetAdapter.ExtractRequestParamsFlags 7782
DBAdapt.TCustomDataSetAdapter.FullyQualifyInputNames 7782
DBAdapt.TCustomDataSetAdapter.GetAdapterMode 7783
DBAdapt.TCustomDataSetAdapter.GetAdapterModeNames 7783
DBAdapt.TCustomDataSetAdapter.GetDataSet 7784
DBAdapt.TCustomDataSetAdapter.GetDefaultMode 7784
DBAdapt.TCustomDataSetAdapter.GetDesigntimeWarnings

7784
DBAdapt.TCustomDataSetAdapter.GetEOF 7785
DBAdapt.TCustomDataSetAdapter.GetFirstRecord 7785
DBAdapt.TCustomDataSetAdapter.GetHiddenFieldOptions 7786
DBAdapt.TCustomDataSetAdapter.GetKeyFields 7786
DBAdapt.TCustomDataSetAdapter.GetKeyParams 7787
DBAdapt.TCustomDataSetAdapter.GetKeyParamStrings 7787
DBAdapt.TCustomDataSetAdapter.GetMasterDetailAdapters 7788
DBAdapt.TCustomDataSetAdapter.GetNeedAdapterMode 7788
DBAdapt.TCustomDataSetAdapter.GetNextRecord 7789
DBAdapt.TCustomDataSetAdapter.GetRecordCount 7789
DBAdapt.TCustomDataSetAdapter.GetRecordIndex 7790
DBAdapt.TCustomDataSetAdapter.GetRecordKeys 7790
DBAdapt.TCustomDataSetAdapter.ImplCanAddActionClass 7790
DBAdapt.TCustomDataSetAdapter.ImplCanAddFieldClass 7791
DBAdapt.TCustomDataSetAdapter.ImplEndIterator 7792
DBAdapt.TCustomDataSetAdapter.ImplGetActionsList 7792
DBAdapt.TCustomDataSetAdapter.ImplGetFieldsList 7793
DBAdapt.TCustomDataSetAdapter.ImplGetIteratorIndex 7793
DBAdapt.TCustomDataSetAdapter.ImplGetRecordKeys 7794
DBAdapt.TCustomDataSetAdapter.ImplInIterator 7794
DBAdapt.TCustomDataSetAdapter.ImplNotifyDeactivate 7795
DBAdapt.TCustomDataSetAdapter.ImplSetEchoActionFieldValues 7795
DBAdapt.TCustomDataSetAdapter.ImplStartIterator 7795
DBAdapt.TCustomDataSetAdapter.InDefaultMode 7808
DBAdapt.TCustomDataSetAdapter.Locate 7796
DBAdapt.TCustomDataSetAdapter.LocateAndApply 7796
DBAdapt.TCustomDataSetAdapter.LocateKeyParams 7797
DBAdapt.TCustomDataSetAdapter.LocateParamsList 7809
DBAdapt.TCustomDataSetAdapter.LocateRequestRecord 7798
DBAdapt.TCustomDataSetAdapter.MasterAdapter 7809
DBAdapt.TCustomDataSetAdapter.Mode 7810
DBAdapt.TCustomDataSetAdapter.Notification 7798
DBAdapt.TCustomDataSetAdapter.OnDataSetStateChange 7798
DBAdapt.TCustomDataSetAdapter.OnPrepareDataSet 7810
DBAdapt.TCustomDataSetAdapter.PrepareDataSet 7799
DBAdapt.TCustomDataSetAdapter.RaiseFieldChangedByAnotherUser 7799
DBAdapt.TCustomDataSetAdapter.RaiseFieldNotFound 7800
DBAdapt.TCustomDataSetAdapter.RaiseNilDataSet 7800
DBAdapt.TCustomDataSetAdapter.RaiseRowNotFound 7800
DBAdapt.TCustomDataSetAdapter.RefreshAfterPost 7810
DBAdapt.TCustomDataSetAdapter.RestoreDefaultAdapterMode 7801
DBAdapt.TCustomDataSetAdapter.SetAdapterMode 7801
DBAdapt.TCustomDataSetAdapter.SetDataSet 7802
DBAdapt.TCustomDataSetAdapter.SetMode 7802
DBAdapt.TCustomDataSetAdapter.SilentLocate 7802
DBAdapt.TCustomDataSetAdapter.StringToMode 7803
DBAdapt.TCustomDataSetAdapter.UpdateFieldsExecuteValues 7803
DBAdapt.TCustomDataSetAdapter.UpdateFieldsGetAnyChanges 7804
DBAdapt.TCustomDataSetAdapter.UpdateFieldsUpdateValues 7805
DBAdapt.TCustomDataSetAdapter.UpdateFieldsValidateValues 7805
DBAdapt.TCustomDataSetAdapterAction 7811
DBAdapt.TCustomDataSetAdapterAction.ActionModes 7821
DBAdapt.TCustomDataSetAdapterAction.ActionName 7821
DBAdapt.TCustomDataSetAdapterAction.Adapter 7822
DBAdapt.TCustomDataSetAdapterAction.Create 7818
DBAdapt.TCustomDataSetAdapterAction.DataSet 7822
DBAdapt.TCustomDataSetAdapterAction.GetAdapter 7819
DBAdapt.TCustomDataSetAdapterAction.GetDataSet 7819
DBAdapt.TCustomDataSetAdapterAction.ImplGetVisible 7819
DBAdapt.TCustomDataSetAdapterAction.ImplWebEnabled 7820
DBAdapt.TCustomDataSetAdapterAction.OnExecute 7822
DBAdapt.TCustomDataSetAdapterAction.OnGetEnabled 7823
DBAdapt.TCustomDataSetAdapterField 7823
DBAdapt.TCustomDataSetAdapterField.GetAdapterValuesList 7831
DBAdapt.TCustomDataSetAdapterField.GetInputStyleType 7831
DBAdapt.TCustomDataSetAdapterField.GetNeedPrevValue 7832
DBAdapt.TCustomDataSetAdapterField.GetPrevValue 7832

DBAdapt.TCustomDataSetAdapterField.ImplCheckOrigValue 7833
DBAdapt.TCustomDataSetAdapterField.ImplCheckValueChange 7833
DBAdapt.TCustomDataSetAdapterField.ImplGetNeedPrevValue 7834
DBAdapt.TCustomDataSetAdapterField.ImplGetPrevValue 7835
DBAdapt.TCustomDataSetAdapterField.ImplGetValue 7835
DBAdapt.TCustomDataSetAdapterField.ImplUpdateValue 7836
DBAdapt.TCustomDataSetAdapterField.Notification 7836
DBAdapt.TCustomDataSetAdapterField.ValuesList 7837
DBAdapt.TCustomDataSetAdapterMemoField 7837
DBAdapt.TCustomDataSetAdapterMemoField.CheckOrUpdateValue 7845
DBAdapt.TCustomDataSetAdapterMemoField.GetInputStyleType 7845
DBAdapt.TCustomDataSetAdapterMemoField.ImplAsFormatted 7845
DBAdapt.TCustomDataSetAdapterMemoField.ImplCheckValueChange 7845
DBAdapt.TCustomDataSetAdapterMemoField.ImplGetValue 7846
DBAdapt.TCustomDataSetAdapterMemoField.ImplUpdateValue 7846
DBAdapt.TCustomDataSetAdapterRowAction 7846
DBAdapt.TCustomDataSetAdapterRowAction.Create 7853
DBAdapt.TCustomDataSetAdapterRowAction.ImplGetAdapterItemRequestParams 7854
DBAdapt.TCustomDataSetAdapterRowAction.ImplWebEnabled 7855
DBAdapt.TCustomDataSetValuesList 7855
DBAdapt.TCustomDataSetValuesList.Create 7863
DBAdapt.TCustomDataSetValuesList.DataSet 7869
DBAdapt.TCustomDataSetValuesList.DataSetPrepared 7870
DBAdapt.TCustomDataSetValuesList.Destroy 7864
DBAdapt.TCustomDataSetValuesList.GetDesigntimeWarnings 7864
DBAdapt.TCustomDataSetValuesList.GetNameFieldClass 7865
DBAdapt.TCustomDataSetValuesList.GetValueFieldClass 7865
DBAdapt.TCustomDataSetValuesList.ImplEndIterator 7866
DBAdapt.TCustomDataSetValuesList.ImplGetListName 7866
DBAdapt.TCustomDataSetValuesList.ImplGetListNameOfValue 7867
DBAdapt.TCustomDataSetValuesList.ImplGetListValue 7867
DBAdapt.TCustomDataSetValuesList.ImplNextIteration 7867
DBAdapt.TCustomDataSetValuesList.ImplNotifyDeactivate 7868
DBAdapt.TCustomDataSetValuesList.ImplStartIterator 7868
DBAdapt.TCustomDataSetValuesList.NameField 7870
DBAdapt.TCustomDataSetValuesList.Notification 7869
DBAdapt.TCustomDataSetValuesList.OnPrepareDataSet 7871
DBAdapt.TCustomDataSetValuesList.ValueField 7871
DBAdapt.TDataAdapterFieldValidateValueEvent 8113
DBAdapt.TDataAdapterFieldValidateValueEvent type 8113
DBAdapt.TDataSetAdapter 7872
DBAdapt.TDataSetAdapter.Actions 7883
DBAdapt.TDataSetAdapter.Data 7883
DBAdapt.TDataSetAdapter.DataSet 7884
DBAdapt.TDataSetAdapter.MasterAdapter 7884
DBAdapt.TDataSetAdapter.ModifyAccess 7884
DBAdapt.TDataSetAdapter.OnAfterExecuteAction 7886
DBAdapt.TDataSetAdapter.OnAfterGetActionResponse 7886
DBAdapt.TDataSetAdapter.OnAfterUpdateFields 7887
DBAdapt.TDataSetAdapter.OnAfterValidateFields 7887
DBAdapt.TDataSetAdapter.OnBeforeExecuteAction 7887
DBAdapt.TDataSetAdapter.OnBeforeGetActionResponse 7888
DBAdapt.TDataSetAdapter.OnBeforeUpdateFields 7888
DBAdapt.TDataSetAdapter.OnBeforeValidateFields 7888
DBAdapt.TDataSetAdapter.OnGetActionParams 7889
DBAdapt.TDataSetAdapter.OnGetRecordCount 7889
DBAdapt.TDataSetAdapter.OnPrepareDataSet 7889
DBAdapt.TDataSetAdapter.PageSize 7884
DBAdapt.TDataSetAdapter.RefreshAfterPost 7885
DBAdapt.TDataSetAdapter.ViewAccess 7885
DBAdapt.TDataSetAdapterApplyRowAction 7890
DBAdapt.TDataSetAdapterApplyRowAction.ActionModes 7899
DBAdapt.TDataSetAdapterApplyRowAction.Create 7897
DBAdapt.TDataSetAdapterApplyRowAction.DisplayLabel 7900
DBAdapt.TDataSetAdapterApplyRowAction.ExecuteAccess 7900
DBAdapt.TDataSetAdapterApplyRowAction.GetDefaultActionNa

me
7898

DBAdapt.TDataSetAdapterApplyRowAction.ImplExecuteActionRequest
7898

DBAdapt.TDataSetAdapterApplyRowAction.ImplIsDefaultAction
7898

DBAdapt.TDataSetAdapterApplyRowAction.ImplWebEnabled
7899

DBAdapt.TDataSetAdapterApplyRowAction.OnAfterExecute
7901

DBAdapt.TDataSetAdapterApplyRowAction.OnAfterGetResponse
7902

DBAdapt.TDataSetAdapterApplyRowAction.OnBeforeExecute
7902

DBAdapt.TDataSetAdapterApplyRowAction.OnBeforeGetResponse
7902

DBAdapt.TDataSetAdapterApplyRowAction.OnGetParams 7903

DBAdapt.TDataSetAdapterApplyRowAction.RedirectOptions
7901

DBAdapt.TDataSetAdapterBrowseRowAction 7903

DBAdapt.TDataSetAdapterBrowseRowAction.ActionModes
7913

DBAdapt.TDataSetAdapterBrowseRowAction.Create 7910

DBAdapt.TDataSetAdapterBrowseRowAction.DisplayLabel 7913

DBAdapt.TDataSetAdapterBrowseRowAction.ExecuteAccess
7914

DBAdapt.TDataSetAdapterBrowseRowAction.GetDefaultActionName
7911

DBAdapt.TDataSetAdapterBrowseRowAction.ImplExecuteActionRequest
7911

DBAdapt.TDataSetAdapterBrowseRowAction.ImplIsDefaultAction
7912

DBAdapt.TDataSetAdapterBrowseRowAction.ImplWebEnabled
7912

DBAdapt.TDataSetAdapterBrowseRowAction.OnAfterExecute
7914

DBAdapt.TDataSetAdapterBrowseRowAction.OnAfterGetResponse
7915

DBAdapt.TDataSetAdapterBrowseRowAction.OnBeforeExecute
7915

DBAdapt.TDataSetAdapterBrowseRowAction.OnBeforeGetResponse
7915

DBAdapt.TDataSetAdapterBrowseRowAction.OnGetParams
7916

DBAdapt.TDataSetAdapterCancelRowAction 7916

DBAdapt.TDataSetAdapterCancelRowAction.ActionModes 7926

DBAdapt.TDataSetAdapterCancelRowAction.Create 7924

DBAdapt.TDataSetAdapterCancelRowAction.DisplayLabel 7926

DBAdapt.TDataSetAdapterCancelRowAction.ExecuteAccess
7927

DBAdapt.TDataSetAdapterCancelRowAction.GetDefaultActionName
7924

DBAdapt.TDataSetAdapterCancelRowAction.ImplExecuteActionRequest
7925

DBAdapt.TDataSetAdapterCancelRowAction.ImplIsDefaultAction
7925

DBAdapt.TDataSetAdapterCancelRowAction.ImplWebEnabled
7925

DBAdapt.TDataSetAdapterCancelRowAction.OnAfterExecute
7928

DBAdapt.TDataSetAdapterCancelRowAction.OnAfterGetResponse
7928

DBAdapt.TDataSetAdapterCancelRowAction.OnBeforeExecute
7929

DBAdapt.TDataSetAdapterCancelRowAction.OnBeforeGetResponse
7929

DBAdapt.TDataSetAdapterCancelRowAction.OnGetParams
7929

DBAdapt.TDataSetAdapterCancelRowAction.RedirectOptions
7927

DBAdapt.TDataSetAdapterDataLink 7930

DBAdapt.TDataSetAdapterDataLink.Create 7932

DBAdapt.TDataSetAdapterDataLink.DataEvent 7932

DBAdapt.TDataSetAdapterDataLink.OnDataSetStateChange
7934

DBAdapt.TDataSetAdapterDataLink.RecordChanged 7933

DBAdapt.TDataSetAdapterDeleteRowAction 7934

DBAdapt.TDataSetAdapterDeleteRowAction.DisplayLabel 7943

DBAdapt.TDataSetAdapterDeleteRowAction.ExecuteAccess
7944

DBAdapt.TDataSetAdapterDeleteRowAction.GetDefaultActionName

7942

DBAdapt.TDataSetAdapterDeleteRowAction.ImplExecuteAction Request
7942

DBAdapt.TDataSetAdapterDeleteRowAction.ImplIsDefaultAction

7943

DBAdapt.TDataSetAdapterDeleteRowAction.OnAfterExecute
7945

DBAdapt.TDataSetAdapterDeleteRowAction.OnAfterGetRespon se
7945

DBAdapt.TDataSetAdapterDeleteRowAction.OnBeforeExecute
7945

DBAdapt.TDataSetAdapterDeleteRowAction.OnBeforeGetResp onse
7946

DBAdapt.TDataSetAdapterDeleteRowAction.OnGetParams
7946

DBAdapt.TDataSetAdapterDeleteRowAction.RedirectOptions
7944

DBAdapt.TDataSetAdapterEditRowAction 7947

DBAdapt.TDataSetAdapterEditRowAction.ActionModes 7956

DBAdapt.TDataSetAdapterEditRowAction.Create 7954

DBAdapt.TDataSetAdapterEditRowAction.DisplayLabel 7957

DBAdapt.TDataSetAdapterEditRowAction.ExecuteAccess 7957

DBAdapt.TDataSetAdapterEditRowAction.GetDefaultActionNam e
7954

DBAdapt.TDataSetAdapterEditRowAction.ImplExecuteActionRe quest
7955

DBAdapt.TDataSetAdapterEditRowAction.ImplIsDefaultAction
7955

DBAdapt.TDataSetAdapterEditRowAction.ImplWebEnabled
7955

DBAdapt.TDataSetAdapterEditRowAction.OnAfterExecute 7958

DBAdapt.TDataSetAdapterEditRowAction.OnAfterGetResponse
7958

DBAdapt.TDataSetAdapterEditRowAction.OnBeforeExecute
7958

DBAdapt.TDataSetAdapterEditRowAction.OnBeforeGetRespons e
7959

DBAdapt.TDataSetAdapterEditRowAction.OnGetParams 7959

DBAdapt.TDataSetAdapterField 7960

DBAdapt.TDataSetAdapterField.DataSetField 7967

DBAdapt.TDataSetAdapterField.FieldFlags 7968

DBAdapt.TDataSetAdapterField.FieldModes 7968

DBAdapt.TDataSetAdapterField.ModifyAccess 7969

DBAdapt.TDataSetAdapterField.OnGetDisplayText 7971

DBAdapt.TDataSetAdapterField.OnGetValue 7971

DBAdapt.TDataSetAdapterField.OnUpdateValue 7969

DBAdapt.TDataSetAdapterField.OnValidateValue 7972

DBAdapt.TDataSetAdapterField.ValuesList 7970

DBAdapt.TDataSetAdapterField.ViewAccess 7970

DBAdapt.TDataSetAdapterFieldFlag 8111

DBAdapt.TDataSetAdapterFieldFlag enumeration 8111

DBAdapt.TDataSetAdapterFieldFlags 8113

DBAdapt.TDataSetAdapterFieldFlags type 8113

DBAdapt.TDataSetAdapterFieldGetStringEvent 8114

DBAdapt.TDataSetAdapterFieldGetStringEvent type 8114

DBAdapt.TDataSetAdapterFieldGetValueEvent 8114

DBAdapt.TDataSetAdapterFieldGetValueEvent type 8114

DBAdapt.TDataSetAdapterFieldValueEvent 8114

DBAdapt.TDataSetAdapterFieldValueEvent type 8114

DBAdapt.TDataSetAdapterFirstRowAction 7972

DBAdapt.TDataSetAdapterFirstRowAction.DisplayLabel 7981

DBAdapt.TDataSetAdapterFirstRowAction.ExecuteAccess 7981

DBAdapt.TDataSetAdapterFirstRowAction.GetDefaultActionNa me
7980

DBAdapt.TDataSetAdapterFirstRowAction.ImplExecuteActionRe quest
7980

DBAdapt.TDataSetAdapterFirstRowAction.ImplWebEnabled
7980

DBAdapt.TDataSetAdapterFirstRowAction.OnAfterExecute 7982

DBAdapt.TDataSetAdapterFirstRowAction.OnAfterGetResponse

7983

DBAdapt.TDataSetAdapterFirstRowAction.OnBeforeExecute
7983

DBAdapt.TDataSetAdapterFirstRowAction.OnBeforeGetRespon se
7984

DBAdapt.TDataSetAdapterFirstRowAction.OnGetParams 7984

DBAdapt.TDataSetAdapterFirstRowAction.RedirectOptions
7982

DBAdapt.TDataSetAdapterImageFieldClass 8115

DBAdapt.TDataSetAdapterImageFieldClass type 8115
DBAdapt.TDataSetAdapterLastRowAction 7984
DBAdapt.TDataSetAdapterLastRowAction.DisplayLabel 7993
DBAdapt.TDataSetAdapterLastRowAction.ExecuteAccess 7993
DBAdapt.TDataSetAdapterLastRowAction.GetDefaultActionNam
e
7992
DBAdapt.TDataSetAdapterLastRowAction.ImplExecuteActionRe
quest
7992
DBAdapt.TDataSetAdapterLastRowAction.ImplWebEnabled
7992
DBAdapt.TDataSetAdapterLastRowAction.OnAfterExecute 7994
DBAdapt.TDataSetAdapterLastRowAction.OnAfterGetResponse
7995
DBAdapt.TDataSetAdapterLastRowAction.OnBeforeExecute
7995
DBAdapt.TDataSetAdapterLastRowAction.OnBeforeGetRespon
se
7996
DBAdapt.TDataSetAdapterLastRowAction.OnGetParams 7996
DBAdapt.TDataSetAdapterLastRowAction.RedirectOptions 7994
DBAdapt.TDataSetAdapterMemoField 7996
DBAdapt.TDataSetAdapterMemoField.DataSetField 8004
DBAdapt.TDataSetAdapterMemoField.FieldFlags 8004
DBAdapt.TDataSetAdapterMemoField.FieldModes 8005
DBAdapt.TDataSetAdapterMemoField.ModifyAccess 8005
DBAdapt.TDataSetAdapterMemoField.OnGetDisplayText 8007
DBAdapt.TDataSetAdapterMemoField.OnGetValue 8007
DBAdapt.TDataSetAdapterMemoField.OnUpdateValue 8006
DBAdapt.TDataSetAdapterMemoField.OnValidateValue 8008
DBAdapt.TDataSetAdapterMemoField.ViewAccess 8006
DBAdapt.TDataSetAdapterMode 8111
DBAdapt.TDataSetAdapterMode enumeration 8111
DBAdapt.TDataSetAdapterModes 8115
DBAdapt.TDataSetAdapterModes type 8115
DBAdapt.TDataSetAdapterMoveRowAction 8008
DBAdapt.TDataSetAdapterMoveRowAction.ImplIsDefaultAction
8016
DBAdapt.TDataSetAdapterNewRowAction 8016
DBAdapt.TDataSetAdapterNewRowAction.ActionModes 8026
DBAdapt.TDataSetAdapterNewRowAction.Create 8024
DBAdapt.TDataSetAdapterNewRowAction.DisplayLabel 8027
DBAdapt.TDataSetAdapterNewRowAction.ExecuteAccess 8027
DBAdapt.TDataSetAdapterNewRowAction.GetDefaultActionNa
me
8024
DBAdapt.TDataSetAdapterNewRowAction.ImplExecuteActionRe
quest
8025
DBAdapt.TDataSetAdapterNewRowAction.ImplIsDefaultAction
8025
DBAdapt.TDataSetAdapterNewRowAction.ImplWebEnabled
8026
DBAdapt.TDataSetAdapterNewRowAction.OnAfterExecute 8028
DBAdapt.TDataSetAdapterNewRowAction.OnAfterGetResponse
8028
DBAdapt.TDataSetAdapterNewRowAction.OnBeforeExecute
8029
DBAdapt.TDataSetAdapterNewRowAction.OnBeforeGetRespon
se
8029
DBAdapt.TDataSetAdapterNewRowAction.OnGetParams 8029
DBAdapt.TDataSetAdapterNextRowAction 8030
DBAdapt.TDataSetAdapterNextRowAction.DisplayLabel 8038
DBAdapt.TDataSetAdapterNextRowAction.ExecuteAccess 8039
DBAdapt.TDataSetAdapterNextRowAction.GetDefaultActionNa
me
8037
DBAdapt.TDataSetAdapterNextRowAction.ImplExecuteActionR
equest
8037
DBAdapt.TDataSetAdapterNextRowAction.ImplWebEnabled
8038
DBAdapt.TDataSetAdapterNextRowAction.OnAfterExecute
8040
DBAdapt.TDataSetAdapterNextRowAction.OnAfterGetRespons
e
8040
DBAdapt.TDataSetAdapterNextRowAction.OnBeforeExecute
8040
DBAdapt.TDataSetAdapterNextRowAction.OnBeforeGetRespon
se
8041
DBAdapt.TDataSetAdapterNextRowAction.OnGetParams 8041
DBAdapt.TDataSetAdapterNextRowAction.RedirectOptions
8039
DBAdapt.TDataSetAdapterPrevRowAction 8042

DBAdapt.TDataSetAdapterPrevRowAction.DisplayLabel 8050
DBAdapt.TDataSetAdapterPrevRowAction.ExecuteAccess 8051
DBAdapt.TDataSetAdapterPrevRowAction.GetDefaultActionName 8049
DBAdapt.TDataSetAdapterPrevRowAction.ImplExecuteActionRequest 8049
DBAdapt.TDataSetAdapterPrevRowAction.ImplWebEnabled 8050
DBAdapt.TDataSetAdapterPrevRowAction.OnAfterExecute 8052
DBAdapt.TDataSetAdapterPrevRowAction.OnAfterGetResponse 8052
DBAdapt.TDataSetAdapterPrevRowAction.OnBeforeExecute 8052
DBAdapt.TDataSetAdapterPrevRowAction.OnBeforeGetResponse 8053
DBAdapt.TDataSetAdapterPrevRowAction.OnGetParams 8053
DBAdapt.TDataSetAdapterPrevRowAction.RedirectOptions 8051
DBAdapt.TDataSetAdapterRefreshRowAction 8054
DBAdapt.TDataSetAdapterRefreshRowAction.DisplayLabel 8063
DBAdapt.TDataSetAdapterRefreshRowAction.ExecuteAccess 8063
DBAdapt.TDataSetAdapterRefreshRowAction.GetDefaultActionName 8061
DBAdapt.TDataSetAdapterRefreshRowAction.ImplExecuteActionRequest 8061
DBAdapt.TDataSetAdapterRefreshRowAction.ImplIsDefaultAction 8062
DBAdapt.TDataSetAdapterRefreshRowAction.ImplWebEnabled 8062
DBAdapt.TDataSetAdapterRefreshRowAction.OnAfterExecute 8064
DBAdapt.TDataSetAdapterRefreshRowAction.OnAfterGetResponse 8064
DBAdapt.TDataSetAdapterRefreshRowAction.OnBeforeExecute 8065
DBAdapt.TDataSetAdapterRefreshRowAction.OnBeforeGetResponse 8065
DBAdapt.TDataSetAdapterRefreshRowAction.OnGetParams 8066
DBAdapt.TDataSetAdapterRefreshRowAction.RedirectOptions 8063
DBAdapt.TDataSetAdapterRequestParamOption 8112
DBAdapt.TDataSetAdapterRequestParamOption enumeration 8112
DBAdapt.TDataSetAdapterRequestParamOptions 8115
DBAdapt.TDataSetAdapterRequestParamOptions type 8115
DBAdapt.TDataSetAdapterRowAction 8066
DBAdapt.TDataSetAdapterRowAction.ActionModes 8075
DBAdapt.TDataSetAdapterRowAction.DisplayLabel 8076
DBAdapt.TDataSetAdapterRowAction.ExecuteAccess 8076
DBAdapt.TDataSetAdapterRowAction.ImplExecuteActionRequest 8074
DBAdapt.TDataSetAdapterRowAction.ImplGetAdapterItemRequestParams 8074
DBAdapt.TDataSetAdapterRowAction.OnAfterExecute 8077
DBAdapt.TDataSetAdapterRowAction.OnAfterGetResponse 8077
DBAdapt.TDataSetAdapterRowAction.OnBeforeExecute 8078
DBAdapt.TDataSetAdapterRowAction.OnBeforeGetResponse 8078
DBAdapt.TDataSetAdapterRowAction.OnGetParams 8078
DBAdapt.TDataSetAdapterRowAction.RedirectOptions 8076
DBAdapt.TDataSetValuesList 8079
DBAdapt.TDataSetValuesList.Actions 8088
DBAdapt.TDataSetValuesList.Data 8088
DBAdapt.TDataSetValuesList.DataSet 8088
DBAdapt.TDataSetValuesList.NameField 8089
DBAdapt.TDataSetValuesList.OnPrepareDataSet 8090
DBAdapt.TDataSetValuesList.ValueField 8089
DBAdapt.TLocateParams 8090
DBAdapt.TLocateParams.AdapterName 8094
DBAdapt.TLocateParams.AddParam 8093
DBAdapt.TLocateParams.Count 8094
DBAdapt.TLocateParams.Create 8092
DBAdapt.TLocateParams.Destroy 8092
DBAdapt.TLocateParams.Items 8095

DBAdapt.TLocateParams.ParamNames 8095

DBAdapt.TLocateParams.ParamValues 8096

DBAdapt.TLocateParamsList 8096

DBAdapt.TLocateParamsList.Add 8099

DBAdapt.TLocateParamsList.Clear 8099

DBAdapt.TLocateParamsList.Count 8100

DBAdapt.TLocateParamsList.Create 8098

DBAdapt.TLocateParamsList.Destroy 8098

DBAdapt.TLocateParamsList.ItemByAdapterName 8100

DBAdapt.TLocateParamsList.Items 8101

DBAdaptImg 8119

DBAdaptImg namespace 8119

Classes 8120

Types 8149

DBAdaptImg.TCustomDataSetAdapterImageField 8120

DBAdaptImg.TCustomDataSetAdapterImageField.CheckOrUpdateValue 8128

DBAdaptImg.TCustomDataSetAdapterImageField.Create 8127

DBAdaptImg.TCustomDataSetAdapterImageField.CreateRequestContext 8129

DBAdaptImg.TCustomDataSetAdapterImageField.GetAdapterImage 8130

DBAdaptImg.TCustomDataSetAdapterImageField.GetAdapterItemRequestParams 8130

DBAdaptImg.TCustomDataSetAdapterImageField.GetDataSetFieldValue 8131

DBAdaptImg.TCustomDataSetAdapterImageField.GetDisplayStyleType 8131

DBAdaptImg.TCustomDataSetAdapterImageField.GetInputStyleType 8132

DBAdaptImg.TCustomDataSetAdapterImageField.HandleRequest 8132

DBAdaptImg.TCustomDataSetAdapterImageField.ImplCheckValueChange 8133

DBAdaptImg.TCustomDataSetAdapterImageField.ImplCreateRequestContext 8134

DBAdaptImg.TCustomDataSetAdapterImageField.ImplHandleRequest 8134

DBAdaptImg.TCustomDataSetAdapterImageField.ImplUpdateValue 8135

DBAdaptImg.TCustomDataSetAdapterImageField.ImplWebImageHREF 8135

DBAdaptImg.TCustomDataSetAdapterImageField.OnGetHREF 8137

DBAdaptImg.TCustomDataSetAdapterImageField.OnGetImage 8137

DBAdaptImg.TCustomDataSetAdapterImageField.RenderAdapterImage 8136

DBAdaptImg.TCustomDataSetAdapterImageField.WebImageHREF 8136

DBAdaptImg.TDataSetAdapterImageField 8138

DBAdaptImg.TDataSetAdapterImageField.DataSetField 8145

DBAdaptImg.TDataSetAdapterImageField.FieldModes 8146

DBAdaptImg.TDataSetAdapterImageField.ModifyAccess 8146

DBAdaptImg.TDataSetAdapterImageField.OnGetHREF 8147

DBAdaptImg.TDataSetAdapterImageField.OnGetImage 8148

DBAdaptImg.TDataSetAdapterImageField.ViewAccess 8147

DBAdaptImg.TImageDataSetFieldGetImageEvent 8149

DBAdaptImg.TImageDataSetFieldGetImageEvent type 8149

dbAdd enumeration member 10253

dbAddAlias enumeration member 10253

dbAddDriver enumeration member 10253

DBBdeWeb 8151

DBBdeWeb namespace 8151

Classes 8152

DBBdeWeb.TQueryTableProducer 8152

DBBdeWeb.TQueryTableProducer.Caption 8159

DBBdeWeb.TQueryTableProducer.CaptionAlignment 8159

DBBdeWeb.TQueryTableProducer.Columns 8160

DBBdeWeb.TQueryTableProducer.Content 8157

DBBdeWeb.TQueryTableProducer.Footer 8160

DBBdeWeb.TQueryTableProducer.GetDataSet 8158

DBBdeWeb.TQueryTableProducer.Header 8161

DBBdeWeb.TQueryTableProducer.MaxRows 8161

DBBdeWeb.TQueryTableProducer.Notification 8158

DBBdeWeb.TQueryTableProducer.OnCreateContent 8164
DBBdeWeb.TQueryTableProducer.OnFormatCell 8164
DBBdeWeb.TQueryTableProducer.OnGetTableCaption 8165
DBBdeWeb.TQueryTableProducer.Query 8162
DBBdeWeb.TQueryTableProducer.RowAttributes 8162
DBBdeWeb.TQueryTableProducer.SetDataSet 8158
DBBdeWeb.TQueryTableProducer.TableAttributes 8163
DBCGrids 8167
DBCGrids namespace 8167
Classes 8168
Structs, Records, Enums 8257
Types 8259
DBCGrids.TDBCtrlGrid 8168
DBCGrids.TDBCtrlGrid.Align 8188
DBCGrids.TDBCtrlGrid.AllowDelete 8191
DBCGrids.TDBCtrlGrid.AllowInsert 8192
DBCGrids.TDBCtrlGrid.Anchors 8192
DBCGrids.TDBCtrlGrid.Canvas 8193
DBCGrids.TDBCtrlGrid.ColCount 8193
DBCGrids.TDBCtrlGrid.Color 8193
DBCGrids.TDBCtrlGrid.Constraints 8194
DBCGrids.TDBCtrlGrid.Create 8179
DBCGrids.TDBCtrlGrid.CreateParams 8180
DBCGrids.TDBCtrlGrid.CreateWnd 8181
DBCGrids.TDBCtrlGrid.DataSource 8195
DBCGrids.TDBCtrlGrid.Destroy 8179
DBCGrids.TDBCtrlGrid.DoKey 8181
DBCGrids.TDBCtrlGrid.DragCursor 8195
DBCGrids.TDBCtrlGrid.DragMode 8195
DBCGrids.TDBCtrlGrid.EditMode 8197
DBCGrids.TDBCtrlGrid.Enabled 8197
DBCGrids.TDBCtrlGrid.ExecuteAction 8182
DBCGrids.TDBCtrlGrid.Font 8198
DBCGrids.TDBCtrlGrid.GetChildParent 8183
DBCGrids.TDBCtrlGrid.GetChildren 8183
DBCGrids.TDBCtrlGrid.GetTabOrderList 8183
DBCGrids.TDBCtrlGrid.KeyDown 8184
DBCGrids.TDBCtrlGrid.OnClick 8213
DBCGrids.TDBCtrlGrid.OnDblClick 8215
DBCGrids.TDBCtrlGrid.OnDragDrop 8215
DBCGrids.TDBCtrlGrid.OnDragOver 8218
DBCGrids.TDBCtrlGrid.OnEndDrag 8220
DBCGrids.TDBCtrlGrid.OnEnter 8221
DBCGrids.TDBCtrlGrid.OnExit 8224
DBCGrids.TDBCtrlGrid.OnKeyDown 8225
DBCGrids.TDBCtrlGrid.OnKeyPress 8228
DBCGrids.TDBCtrlGrid.OnKeyUp 8229
DBCGrids.TDBCtrlGrid.OnMouseActivate 8231
DBCGrids.TDBCtrlGrid.OnMouseDown 8232
DBCGrids.TDBCtrlGrid.OnMouseEnter 8234
DBCGrids.TDBCtrlGrid.OnMouseLeave 8234
DBCGrids.TDBCtrlGrid.OnMouseMove 8234
DBCGrids.TDBCtrlGrid.OnMouseUp 8237
DBCGrids.TDBCtrlGrid.OnPaintPanel 8239
DBCGrids.TDBCtrlGrid.OnStartDrag 8240
DBCGrids.TDBCtrlGrid.Orientation 8199
DBCGrids.TDBCtrlGrid.PaintPanel 8185
DBCGrids.TDBCtrlGrid.PaintWindow 8185
DBCGrids.TDBCtrlGrid.Panel 8199
DBCGrids.TDBCtrlGrid.PanelBorder 8200
DBCGrids.TDBCtrlGrid.PanelCount 8200
DBCGrids.TDBCtrlGrid.PanelHeight 8200
DBCGrids.TDBCtrlGrid.PanelIndex 8201
DBCGrids.TDBCtrlGrid.PanelWidth 8201
DBCGrids.TDBCtrlGrid.ParentColor 8201
DBCGrids.TDBCtrlGrid.ParentFont 8202
DBCGrids.TDBCtrlGrid.ParentShowHint 8202
DBCGrids.TDBCtrlGrid.PopupMenu 8203
DBCGrids.TDBCtrlGrid.ReadState 8185
DBCGrids.TDBCtrlGrid.RowCount 8207
DBCGrids.TDBCtrlGrid.SelectedColor 8208
DBCGrids.TDBCtrlGrid.SetBounds 8186
DBCGrids.TDBCtrlGrid.ShowFocus 8208
DBCGrids.TDBCtrlGrid.ShowHint 8209
DBCGrids.TDBCtrlGrid.TabOrder 8210
DBCGrids.TDBCtrlGrid.TabStop 8211
DBCGrids.TDBCtrlGrid.UpdateAction 8186
DBCGrids.TDBCtrlGrid.Visible 8211

DBCGrids.TDBCtrlGridBorder 8257
DBCGrids.TDBCtrlGridBorder enumeration 8257
DBCGrids.TDBCtrlGridKey 8257
DBCGrids.TDBCtrlGridKey enumeration 8257
DBCGrids.TDBCtrlGridLink 8240
DBCGrids.TDBCtrlGridLink.ActiveChanged 8243
DBCGrids.TDBCtrlGridLink.Create 8243
DBCGrids.TDBCtrlGridLink.DataSetChanged 8243
DBCGrids.TDBCtrlGridOrientation 8258
DBCGrids.TDBCtrlGridOrientation enumeration 8258
DBCGrids.TDBCtrlPanel 8244
DBCGrids.TDBCtrlPanel.CreateLinked 8255
DBCGrids.TDBCtrlPanel.CreateParams 8255
DBCGrids.TDBCtrlPanel.PaintWindow 8256
DBCGrids.TPaintPanelEvent 8259
DBCGrids.TPaintPanelEvent type 8259
DBClient 8261
DBClient namespace 8261
Classes 8263
Constants 8545
Functions 8534
Structs, Records, Enums 8536
Types 8541
DBClient.AllParamTypes 8545
DBClient.AllParamTypes constant 8545
DBClient.AllRecords 8545
DBClient.AllRecords constant 8545
DBClient.EDBClient 8263
DBClient.EDBClient.Create 8264
DBClient.EDBClient.ErrorCode 8265
DBClient.EReconcileError 8265
DBClient.EReconcileError.Context 8267
DBClient.EReconcileError.Create 8267
DBClient.EReconcileError.PreviousError 8268
DBClient.PackageParams 8534
DBClient.PackageParams function 8534
DBClient.PKeyBuffer 8541
DBClient.PKeyBuffer type 8541
DBClient.PRecInfo 8541
DBClient.PRecInfo type 8541
DBClient.TAggregate 8268
DBClient.TAggregate.Activate 8271
DBClient.TAggregate.Active 8274
DBClient.TAggregate.AggHandle 8274
DBClient.TAggregate.AggregateName 8275
DBClient.TAggregate.Assign 8272
DBClient.TAggregate.Create 8270
DBClient.TAggregate.DataSet 8275
DBClient.TAggregate.DataSize 8275
DBClient.TAggregate.DataType 8276
DBClient.TAggregate.DependentFields 8276
DBClient.TAggregate.Destroy 8271
DBClient.TAggregate.Expression 8276
DBClient.TAggregate.GetDisplayName 8272
DBClient.TAggregate.GroupingLevel 8277
DBClient.TAggregate.IndexName 8278
DBClient.TAggregate.InUse 8278
DBClient.TAggregate.OnUpdate 8279
DBClient.TAggregate.RecBufOfs 8278
DBClient.TAggregate.Value 8273
DBClient.TAggregate.Visible 8279
DBClient.TAggregates 8280
DBClient.TAggregates.Add 8282
DBClient.TAggregates.Clear 8283
DBClient.TAggregates.Create 8282
DBClient.TAggregates.Find 8283
DBClient.TAggregates.GetOwner 8283
DBClient.TAggregates.IndexOf 8284
DBClient.TAggregates.Items 8285
DBClient.TAggUpdateEvent 8542
DBClient.TAggUpdateEvent type 8542
DBClient.TCDSKeyBuffer 8542
DBClient.TCDSKeyBuffer type 8542
DBClient.TCDSRecInfo 8542
DBClient.TCDSRecInfo type 8542
DBClient.TClientBlobStream 8285
DBClient.TClientBlobStream.Create 8287
DBClient.TClientBlobStream.Destroy 8288

DBClient.TClientBlobStream.Truncate 8288
DBClient.TClientBlobStream.Write 8289
DBClient.TClientDataSet 8289
DBClient.TClientDataSet.Active 8305
DBClient.TClientDataSet.AfterApplyUpdates 8332
DBClient.TClientDataSet.AfterCancel 8333
DBClient.TClientDataSet.AfterClose 8335
DBClient.TClientDataSet.AfterDelete 8336
DBClient.TClientDataSet.AfterEdit 8339
DBClient.TClientDataSet.AfterExecute 8340
DBClient.TClientDataSet.AfterGetParams 8340
DBClient.TClientDataSet.AfterGetRecords 8341
DBClient.TClientDataSet.AfterInsert 8342
DBClient.TClientDataSet.AfterOpen 8342
DBClient.TClientDataSet.AfterPost 8344
DBClient.TClientDataSet.AfterRefresh 8344
DBClient.TClientDataSet.AfterRowRequest 8345
DBClient.TClientDataSet.AfterScroll 8346
DBClient.TClientDataSet.Aggregates 8308
DBClient.TClientDataSet.AggregatesActive 8308
DBClient.TClientDataSet.AutoCalcFields 8309
DBClient.TClientDataSet.BeforeApplyUpdates 8346
DBClient.TClientDataSet.BeforeCancel 8347
DBClient.TClientDataSet.BeforeClose 8347
DBClient.TClientDataSet.BeforeDelete 8348
DBClient.TClientDataSet.BeforeEdit 8348
DBClient.TClientDataSet.BeforeExecute 8349
DBClient.TClientDataSet.BeforeGetParams 8350
DBClient.TClientDataSet.BeforeGetRecords 8350
DBClient.TClientDataSet.BeforeInsert 8352
DBClient.TClientDataSet.BeforeOpen 8353
DBClient.TClientDataSet.BeforePost 8354
DBClient.TClientDataSet.BeforeRefresh 8354
DBClient.TClientDataSet.BeforeRowRequest 8355
DBClient.TClientDataSet.BeforeScroll 8356
DBClient.TClientDataSet.CommandText 8310
DBClient.TClientDataSet.ConnectionBroker 8311
DBClient.TClientDataSet.Constraints 8311
DBClient.TClientDataSet.DataSetField 8312
DBClient.TClientDataSet.DisableStringTrim 8312
DBClient.TClientDataSet.FetchOnDemand 8313
DBClient.TClientDataSet.FieldDefs 8314
DBClient.TClientDataSet.FileName 8317
DBClient.TClientDataSet.Filter 8317
DBClient.TClientDataSet.Filtered 8319
DBClient.TClientDataSet.FilterOptions 8320
DBClient.TClientDataSet.IndexDefs 8320
DBClient.TClientDataSet.IndexFieldNames 8322
DBClient.TClientDataSet.IndexName 8322
DBClient.TClientDataSet.MasterFields 8323
DBClient.TClientDataSet.MasterSource 8324
DBClient.TClientDataSet.ObjectView 8324
DBClient.TClientDataSet.OnCalcFields 8356
DBClient.TClientDataSet.OnDeleteError 8357
DBClient.TClientDataSet.OnEditError 8357
DBClient.TClientDataSet.OnFilterRecord 8358
DBClient.TClientDataSet.OnNewRecord 8359
DBClient.TClientDataSet.OnPostError 8359
DBClient.TClientDataSet.OnReconcileError 8359
DBClient.TClientDataSet.PacketRecords 8325
DBClient.TClientDataSet.Params 8326
DBClient.TClientDataSet.ProviderName 8329
DBClient.TClientDataSet.Ranged 8329
DBClient.TClientDataSet.ReadOnly 8330
DBClient.TClientDataSet.RemoteServer 8330
DBClient.TClientDataSet.StoreDefs 8331
DBClient.TConnectionBroker 8360
DBClient.TConnectionBroker.AfterConnect 8371
DBClient.TConnectionBroker.AfterDisconnect 8372
DBClient.TConnectionBroker.BeforeConnect 8372
DBClient.TConnectionBroker.BeforeDisconnect 8372
DBClient.TConnectionBroker.Connected 8368
DBClient.TConnectionBroker.Connection 8368
DBClient.TConnectionBroker.Create 8364
DBClient.TConnectionBroker.GetAppServer 8365
DBClient.TConnectionBroker.GetConnected 8365
DBClient.TConnectionBroker.GetServer 8366
DBClient.TConnectionBroker.Loaded 8366

DBClient.TConnectionBroker.LoginPrompt 8369
DBClient.TConnectionBroker.Notification 8366
DBClient.TConnectionBroker.OnGetUsername 8373
DBClient.TConnectionBroker.OnLogin 8370
DBClient.TConnectionBroker.SetConnected 8367
DBClient.TCustomClientDataSet 8373
DBClient.TCustomClientDataSet.ActivateFilters 8391
DBClient.TCustomClientDataSet.Active 8458
DBClient.TCustomClientDataSet.ActiveAggs 8460
DBClient.TCustomClientDataSet.AddDataPacket 8391
DBClient.TCustomClientDataSet.AddFieldDesc 8391
DBClient.TCustomClientDataSet.AddIndex 8391
DBClient.TCustomClientDataSet.AfterApplyUpdates 8497
DBClient.TCustomClientDataSet.AfterCancel 8498
DBClient.TCustomClientDataSet.AfterClose 8500
DBClient.TCustomClientDataSet.AfterDelete 8501
DBClient.TCustomClientDataSet.AfterEdit 8504
DBClient.TCustomClientDataSet.AfterExecute 8504
DBClient.TCustomClientDataSet.AfterGetParams 8505
DBClient.TCustomClientDataSet.AfterGetRecords 8506
DBClient.TCustomClientDataSet.AfterInsert 8507
DBClient.TCustomClientDataSet.AfterOpen 8507
DBClient.TCustomClientDataSet.AfterPost 8509
DBClient.TCustomClientDataSet.AfterRefresh 8509
DBClient.TCustomClientDataSet.AfterRowRequest 8509
DBClient.TCustomClientDataSet.AfterScroll 8510
DBClient.TCustomClientDataSet.Aggregates 8461
DBClient.TCustomClientDataSet.AggregatesActive 8461
DBClient.TCustomClientDataSet.AllocKeyBuffers 8392
DBClient.TCustomClientDataSet.AllocRecordBuffer 8393
DBClient.TCustomClientDataSet.AppendData 8393
DBClient.TCustomClientDataSet.ApplyRange 8393
DBClient.TCustomClientDataSet.ApplyUpdates 8394
DBClient.TCustomClientDataSet.AppServer 8462
DBClient.TCustomClientDataSet.AutoCalcFields 8463
DBClient.TCustomClientDataSet.BeforeApplyUpdates 8511
DBClient.TCustomClientDataSet.BeforeCancel 8512
DBClient.TCustomClientDataSet.BeforeClose 8512
DBClient.TCustomClientDataSet.BeforeDelete 8512
DBClient.TCustomClientDataSet.BeforeEdit 8513
DBClient.TCustomClientDataSet.BeforeExecute 8514
DBClient.TCustomClientDataSet.BeforeGetParams 8514
DBClient.TCustomClientDataSet.BeforeGetRecords 8515
DBClient.TCustomClientDataSet.BeforeInsert 8517
DBClient.TCustomClientDataSet.BeforeOpen 8518
DBClient.TCustomClientDataSet.BeforePost 8518
DBClient.TCustomClientDataSet.BeforeRefresh 8519
DBClient.TCustomClientDataSet.BeforeRowRequest 8520
DBClient.TCustomClientDataSet.BeforeScroll 8520
DBClient.TCustomClientDataSet.BookmarkValid 8395
DBClient.TCustomClientDataSet.Cancel 8395
DBClient.TCustomClientDataSet.CancelRange 8396
DBClient.TCustomClientDataSet.CancelUpdates 8396
DBClient.TCustomClientDataSet.ChangeCount 8463
DBClient.TCustomClientDataSet.Check 8397
DBClient.TCustomClientDataSet.CheckDetailRecords 8397
DBClient.TCustomClientDataSet.CheckProviderEOF 8397
DBClient.TCustomClientDataSet.CheckSetKeyMode 8397
DBClient.TCustomClientDataSet.ClearCalcFields 8398
DBClient.TCustomClientDataSet.CloneCursor 8398
DBClient.TCustomClientDataSet.CloneSource 8464
DBClient.TCustomClientDataSet.CloseCursor 8400
DBClient.TCustomClientDataSet.CommandText 8465
DBClient.TCustomClientDataSet.CompareBookmarks 8400
DBClient.TCustomClientDataSet.ConnectionBroker 8466
DBClient.TCustomClientDataSet.Constraints 8466
DBClient.TCustomClientDataSet.ConstraintsDisabled 8400
DBClient.TCustomClientDataSet.ConstraintsStored 8401
DBClient.TCustomClientDataSet.Create 8386
DBClient.TCustomClientDataSet.CreateBlobStream 8401
DBClient.TCustomClientDataSet.CreateDataSet 8401
DBClient.TCustomClientDataSet.Data 8467
DBClient.TCustomClientDataSet.DataConvert 8404
DBClient.TCustomClientDataSet.DataEvent 8404
DBClient.TCustomClientDataSet.DataRequest 8404
DBClient.TCustomClientDataSet.DataSetField 8469
DBClient.TCustomClientDataSet.DataSize 8469
DBClient.TCustomClientDataSet.DeactivateFilters 8405

DBClient.TCustomClientDataSet.DefChanged 8405
DBClient.TCustomClientDataSet.DefineProperties 8405
DBClient.TCustomClientDataSet.DeleteIndex 8406
DBClient.TCustomClientDataSet.Delta 8470
DBClient.TCustomClientDataSet.Destroy 8387
DBClient.TCustomClientDataSet.DestroyLookupCursor 8406
DBClient.TCustomClientDataSet.DisableConstraints 8406
DBClient.TCustomClientDataSet.DisableStringTrim 8470
DBClient.TCustomClientDataSet.DoAfterApplyUpdates 8407
DBClient.TCustomClientDataSet.DoAfterExecute 8407
DBClient.TCustomClientDataSet.DoAfterGetParams 8407
DBClient.TCustomClientDataSet.DoAfterGetRecords 8407
DBClient.TCustomClientDataSet.DoAfterRowRequest 8408
DBClient.TCustomClientDataSet.DoApplyUpdates 8408
DBClient.TCustomClientDataSet.DoBeforeApplyUpdates 8408
DBClient.TCustomClientDataSet.DoBeforeExecute 8408
DBClient.TCustomClientDataSet.DoBeforeGetParams 8408
DBClient.TCustomClientDataSet.DoBeforeGetRecords 8409
DBClient.TCustomClientDataSet.DoBeforeInsert 8409
DBClient.TCustomClientDataSet.DoBeforeRowRequest 8409
DBClient.TCustomClientDataSet.DoExecute 8409
DBClient.TCustomClientDataSet.DoGetRecords 8410
DBClient.TCustomClientDataSet.DoOnNewRecord 8410
DBClient.TCustomClientDataSet.DoRowRequest 8410
DBClient.TCustomClientDataSet.DSBase 8471
DBClient.TCustomClientDataSet.DSCursor 8471
DBClient.TCustomClientDataSet.EditKey 8410
DBClient.TCustomClientDataSet.EditRangeEnd 8411
DBClient.TCustomClientDataSet.EditRangeStart 8411
DBClient.TCustomClientDataSet.EmptyDataSet 8412
DBClient.TCustomClientDataSet.EnableConstraints 8412
DBClient.TCustomClientDataSet.Execute 8413
DBClient.TCustomClientDataSet.FetchBlobs 8414
DBClient.TCustomClientDataSet.FetchDetails 8414
DBClient.TCustomClientDataSet.FetchMoreData 8415
DBClient.TCustomClientDataSet.FetchOnDemand 8471
DBClient.TCustomClientDataSet.FetchParams 8415
DBClient.TCustomClientDataSet.FieldDefs 8472
DBClient.TCustomClientDataSet.FileName 8475
DBClient.TCustomClientDataSet.Filter 8476
DBClient.TCustomClientDataSet.Filtered 8478
DBClient.TCustomClientDataSet.FilterOptions 8478
DBClient.TCustomClientDataSet.FindKey 8416
DBClient.TCustomClientDataSet.FindNearest 8416
DBClient.TCustomClientDataSet.FindRecord 8417
DBClient.TCustomClientDataSet.FreeKeyBuffers 8417
DBClient.TCustomClientDataSet.FreeRecordBuffer 8417
DBClient.TCustomClientDataSet.GetAggregateValue 8418
DBClient.TCustomClientDataSet.GetAppServer 8418
DBClient.TCustomClientDataSet.GetBookmarkData 8418
DBClient.TCustomClientDataSet.GetBookmarkFlag 8418
DBClient.TCustomClientDataSet.GetCanModify 8418
DBClient.TCustomClientDataSet.GetCurrentRecord 8419
DBClient.TCustomClientDataSet.GetDataSource 8419
DBClient.TCustomClientDataSet.GetFieldData 8419
DBClient.TCustomClientDataSet.GetGroupState 8420
DBClient.TCustomClientDataSet.GetIndexField 8421
DBClient.TCustomClientDataSet.GetIndexFieldCount 8421
DBClient.TCustomClientDataSet.GetIndexInfo 8421
DBClient.TCustomClientDataSet.GetIndexNames 8422
DBClient.TCustomClientDataSet.GetIsClone 8422
DBClient.TCustomClientDataSet.GetIsIndexField 8422
DBClient.TCustomClientDataSet.GetKeyBuffer 8423
DBClient.TCustomClientDataSet.GetKeyExclusive 8423
DBClient.TCustomClientDataSet.GetKeyFieldCount 8423
DBClient.TCustomClientDataSet.GetNextPacket 8423
DBClient.TCustomClientDataSet.GetOptionalParam 8425
DBClient.TCustomClientDataSet.GetRecNo 8425
DBClient.TCustomClientDataSet.GetRecord 8425
DBClient.TCustomClientDataSet.GetRecordCount 8426
DBClient.TCustomClientDataSet.GetRecordSize 8426
DBClient.TCustomClientDataSet.GetRemoteServer 8426
DBClient.TCustomClientDataSet.GetStateFieldValue 8426
DBClient.TCustomClientDataSet.GotoCurrent 8427
DBClient.TCustomClientDataSet.GotoKey 8427
DBClient.TCustomClientDataSet.GotoNearest 8427
DBClient.TCustomClientDataSet.GroupingLevel 8479
DBClient.TCustomClientDataSet.HasAppServer 8480

DBClient.TCustomClientDataSet.IndexDefs 8480
DBClient.TCustomClientDataSet.IndexFieldCount 8481
DBClient.TCustomClientDataSet.IndexFieldNames 8482
DBClient.TCustomClientDataSet.IndexFields 8482
DBClient.TCustomClientDataSet.IndexName 8483
DBClient.TCustomClientDataSet.InitKeyBuffer 8428
DBClient.TCustomClientDataSet.InitRecord 8428
DBClient.TCustomClientDataSet.InternalAddRecord 8428
DBClient.TCustomClientDataSet.InternalCancel 8429
DBClient.TCustomClientDataSet.InternalClose 8429
DBClient.TCustomClientDataSet.InternalDelete 8429
DBClient.TCustomClientDataSet.InternalEdit 8429
DBClient.TCustomClientDataSet.InternalFirst 8430
DBClient.TCustomClientDataSet.InternalGetOptionalParam 8430
DBClient.TCustomClientDataSet.InternalGotoBookmark 8430
DBClient.TCustomClientDataSet.InternalHandleException 8430
DBClient.TCustomClientDataSet.InternalInitFieldDefs 8430
DBClient.TCustomClientDataSet.InternalInitRecord 8431
DBClient.TCustomClientDataSet.InternalInsert 8431
DBClient.TCustomClientDataSet.InternalLast 8431
DBClient.TCustomClientDataSet.InternalOpen 8431
DBClient.TCustomClientDataSet.InternalPost 8432
DBClient.TCustomClientDataSet.InternalRefresh 8432
DBClient.TCustomClientDataSet.InternalSetOptionalParam 8432
DBClient.TCustomClientDataSet.InternalSetToRecord 8432
DBClient.TCustomClientDataSet.IsClone 8484
DBClient.TCustomClientDataSet.IsCursorOpen 8432
DBClient.TCustomClientDataSet.KeyExclusive 8484
DBClient.TCustomClientDataSet.KeyFieldCount 8484
DBClient.TCustomClientDataSet.KeySize 8485
DBClient.TCustomClientDataSet.Loaded 8433
DBClient.TCustomClientDataSet.LoadFromFile 8433
DBClient.TCustomClientDataSet.LoadFromStream 8435
DBClient.TCustomClientDataSet.Locate 8435
DBClient.TCustomClientDataSet.LocateRecord 8436
DBClient.TCustomClientDataSet.LogChanges 8485
DBClient.TCustomClientDataSet.Lookup 8436
DBClient.TCustomClientDataSet.MasterFields 8486
DBClient.TCustomClientDataSet.MasterSource 8486
DBClient.TCustomClientDataSet.MergeChangeLog 8437
DBClient.TCustomClientDataSet.Notification 8437
DBClient.TCustomClientDataSet.ObjectView 8487
DBClient.TCustomClientDataSet.OnCalcFields 8521
DBClient.TCustomClientDataSet.OnDeleteError 8522
DBClient.TCustomClientDataSet.OnEditError 8522
DBClient.TCustomClientDataSet.OnFilterRecord 8523
DBClient.TCustomClientDataSet.OnNewRecord 8523
DBClient.TCustomClientDataSet.OnPostError 8524
DBClient.TCustomClientDataSet.OnReconcileError 8524
DBClient.TCustomClientDataSet.OpenCursor 8438
DBClient.TCustomClientDataSet.PacketRecords 8488
DBClient.TCustomClientDataSet.Params 8489
DBClient.TCustomClientDataSet.Post 8438
DBClient.TCustomClientDataSet.PostKeyBuffer 8439
DBClient.TCustomClientDataSet.ProviderEOF 8491
DBClient.TCustomClientDataSet.ProviderName 8492
DBClient.TCustomClientDataSet.PSGetCommandText 8439
DBClient.TCustomClientDataSet.PSGetCommandType 8440
DBClient.TCustomClientDataSet.PSGetIndexDefs 8440
DBClient.TCustomClientDataSet.Ranged 8492
DBClient.TCustomClientDataSet.ReadDataPacket 8441
DBClient.TCustomClientDataSet.ReadOnly 8493
DBClient.TCustomClientDataSet.Reconcile 8441
DBClient.TCustomClientDataSet.RefreshInternalCalcFields 8441
DBClient.TCustomClientDataSet.RefreshRecord 8442
DBClient.TCustomClientDataSet.RemoteServer 8493
DBClient.TCustomClientDataSet.ResetAggField 8442
DBClient.TCustomClientDataSet.ResetCursorRange 8443
DBClient.TCustomClientDataSet.RevertRecord 8443
DBClient.TCustomClientDataSet.SavePoint 8494
DBClient.TCustomClientDataSet.SaveToFile 8443
DBClient.TCustomClientDataSet.SaveToStream 8445
DBClient.TCustomClientDataSet.SetAggregates 8445
DBClient.TCustomClientDataSet.SetAltRecBuffers 8446
DBClient.TCustomClientDataSet.SetAppServer 8446
DBClient.TCustomClientDataSet.SetBookmarkData 8446
DBClient.TCustomClientDataSet.SetBookmarkFlag 8446

DBClient.TCustomClientDataSet.SetCommandText 8447
DBClient.TCustomClientDataSet.SetCursorRange 8447
DBClient.TCustomClientDataSet.SetDataSetField 8447
DBClient.TCustomClientDataSet.SetFieldData 8447
DBClient.TCustomClientDataSet.SetFilterData 8448
DBClient.TCustomClientDataSet.SetFiltered 8448
DBClient.TCustomClientDataSet.SetFilterOptions 8448
DBClient.TCustomClientDataSet.SetFilterText 8448
DBClient.TCustomClientDataSet.SetIndexField 8448
DBClient.TCustomClientDataSet.SetKey 8449
DBClient.TCustomClientDataSet.SetKeyBuffer 8449
DBClient.TCustomClientDataSet.SetKeyExclusive 8449
DBClient.TCustomClientDataSet.SetKeyFieldCount 8450
DBClient.TCustomClientDataSet.SetKeyFields 8450
DBClient.TCustomClientDataSet.SetLinkRanges 8450
DBClient.TCustomClientDataSet.SetOnFilterRecord 8450
DBClient.TCustomClientDataSet.SetOptionalParam 8450
DBClient.TCustomClientDataSet.SetProvider 8451
DBClient.TCustomClientDataSet.SetProviderEOF 8452
DBClient.TCustomClientDataSet.SetRange 8452
DBClient.TCustomClientDataSet.SetRangeEnd 8453
DBClient.TCustomClientDataSet.SetRangeStart 8454
DBClient.TCustomClientDataSet.SetRecNo 8454
DBClient.TCustomClientDataSet.SetRemoteServer 8454
DBClient.TCustomClientDataSet.StatusFilter 8495
DBClient.TCustomClientDataSet.StoreDefs 8495
DBClient.TCustomClientDataSet.SwitchToIndex 8455
DBClient.TCustomClientDataSet.SyncCursors 8455
DBClient.TCustomClientDataSet.UndoLastChange 8455
DBClient.TCustomClientDataSet.UpdateIndexDefs 8455
DBClient.TCustomClientDataSet.UpdateStatus 8456
DBClient.TCustomClientDataSet.WriteDataPacket 8456
DBClient.TCustomClientDataSet.XMLData 8496
DBClient.TCustomRemoteServer 8525
DBClient.TCustomRemoteServer.AppServer 8532
DBClient.TCustomRemoteServer.Create 8529
DBClient.TCustomRemoteServer.GetAppServer 8530
DBClient.TCustomRemoteServer.GetProviderNames 8530
DBClient.TCustomRemoteServer.GetServer 8530
DBClient.TCustomRemoteServer.GetServerList 8531
DBClient.TCustomRemoteServer.OnGetUsername 8533
DBClient.TCustomRemoteServer.SetAppServer 8531
DBClient.TDataPacketFormat 8536
DBClient.TDataPacketFormat enumeration 8536
DBClient.TDataSetOption 8537
DBClient.TDataSetOption enumeration 8537
DBClient.TDataSetOptions 8543
DBClient.TDataSetOptions type 8543
DBClient.TFetchOption 8537
DBClient.TFetchOption enumeration 8537
DBClient.TFetchOptions 8543
DBClient.TFetchOptions type 8543
DBClient.TFieldDescList 8543
DBClient.TFieldDescList type 8543
DBClient.TGetUsernameEvent 8544
DBClient.TGetUsernameEvent type 8544
DBClient.TKeyBuffer 8538
DBClient.TKeyBuffer record 8538
DBClient.TRecInfo 8538
DBClient.TRecInfo record 8538
DBClient.TReconcileAction 8539
DBClient.TReconcileAction enumeration 8539
DBClient.TReconcileErrorEvent 8544
DBClient.TReconcileErrorEvent type 8544
DBClient.TReconcileInfo 8539
DBClient.TReconcileInfo record 8539
DBClient.TRemoteEvent 8544
DBClient.TRemoteEvent type 8544
DBClient.UnpackParams 8534
DBClient.UnpackParams function 8534
DBClientActns 8547
DBClientActns namespace 8547
Classes 8548
DBClientActns.TClientDataSetApply 8548
DBClientActns.TClientDataSetApply.Create 8553
DBClientActns.TClientDataSetApply.DataSource 8555
DBClientActns.TClientDataSetApply.Destroy 8553
DBClientActns.TClientDataSetApply.DisplayErrorDlg 8556

DBClientActns.TClientDataSetApply.DoReconcileError 8554
DBClientActns.TClientDataSetApply.ExecuteTarget 8554
DBClientActns.TClientDataSetApply.MaxErrors 8556
DBClientActns.TClientDataSetApply.UpdateTarget 8555
DBClientActns.TClientDataSetRevert 8556
DBClientActns.TClientDataSetRevert.DataSource 8563
DBClientActns.TClientDataSetRevert.ExecuteTarget 8562
DBClientActns.TClientDataSetRevert.UpdateTarget 8562
DBClientActns.TClientDataSetUndo 8563
DBClientActns.TClientDataSetUndo.DataSource 8569
DBClientActns.TClientDataSetUndo.ExecuteTarget 8568
DBClientActns.TClientDataSetUndo.FollowChange 8570
DBClientActns.TClientDataSetUndo.UpdateTarget 8569
DBClientActns.TReconcileActionLink 8570
DBClientActns.TReconcileActionLink.ApplyAction 8573
DBClientActns.TReconcileActionLink.DataEvent 8573
dbClose enumeration member 10253
DBCtrls 8575
DBCtrls namespace 8575
Classes 8577
Constants 9543
Functions 9536
Structs, Records, Enums 9538
Types 9541
DBCtrls.DBUseRightToLeftAlignment 9536
DBCtrls.DBUseRightToLeftAlignment function 9536
DBCtrls.ENavClick 9541
DBCtrls.ENavClick type 9541
DBCtrls.InitRepeatPause 9543
DBCtrls.InitRepeatPause constant 9543
DBCtrls.OkToChangeFieldAlignment 9536
DBCtrls.OkToChangeFieldAlignment function 9536
DBCtrls.RepeatPause 9543
DBCtrls.RepeatPause constant 9543
DBCtrls.SpaceSize 9543
DBCtrls.SpaceSize constant 9543
DBCtrls.TDataSourceLink 8577
DBCtrls.TDataSourceLink.ActiveChanged 8580
DBCtrls.TDataSourceLink.Create 8580
DBCtrls.TDataSourceLink.FocusControl 8581
DBCtrls.TDataSourceLink.LayoutChanged 8581
DBCtrls.TDataSourceLink.RecordChanged 8581
DBCtrls.TDBCheckBox 8582
DBCtrls.TDBCheckBox.Action 8598
DBCtrls.TDBCheckBox.Align 8599
DBCtrls.TDBCheckBox.Alignment 8602
DBCtrls.TDBCheckBox.AllowGrayed 8602
DBCtrls.TDBCheckBox.Anchors 8603
DBCtrls.TDBCheckBox.BiDiMode 8604
DBCtrls.TDBCheckBox.Caption 8604
DBCtrls.TDBCheckBox.Checked 8607
DBCtrls.TDBCheckBox.Color 8608
DBCtrls.TDBCheckBox.Constraints 8609
DBCtrls.TDBCheckBox.Create 8593
DBCtrls.TDBCheckBox.Ctl3D 8609
DBCtrls.TDBCheckBox.DataField 8610
DBCtrls.TDBCheckBox.DataSource 8610
DBCtrls.TDBCheckBox.Destroy 8594
DBCtrls.TDBCheckBox.DragCursor 8611
DBCtrls.TDBCheckBox.DragKind 8611
DBCtrls.TDBCheckBox.DragMode 8611
DBCtrls.TDBCheckBox.Enabled 8613
DBCtrls.TDBCheckBox.ExecuteAction 8594
DBCtrls.TDBCheckBox.Field 8613
DBCtrls.TDBCheckBox.Font 8614
DBCtrls.TDBCheckBox.KeyPress 8595
DBCtrls.TDBCheckBox.Notification 8595
DBCtrls.TDBCheckBox.OnClick 8629
DBCtrls.TDBCheckBox.OnContextPopup 8631
DBCtrls.TDBCheckBox.OnDragDrop 8631
DBCtrls.TDBCheckBox.OnDragOver 8634
DBCtrls.TDBCheckBox.OnEndDock 8636
DBCtrls.TDBCheckBox.OnEndDrag 8637
DBCtrls.TDBCheckBox.OnEnter 8638
DBCtrls.TDBCheckBox.OnExit 8640
DBCtrls.TDBCheckBox.OnKeyDown 8642
DBCtrls.TDBCheckBox.OnKeyPress 8644
DBCtrls.TDBCheckBox.OnKeyUp 8645

DBCtrls.TDBCheckBox.OnMouseActivate 8647
DBCtrls.TDBCheckBox.OnMouseDown 8648
DBCtrls.TDBCheckBox.OnMouseEnter 8650
DBCtrls.TDBCheckBox.OnMouseLeave 8651
DBCtrls.TDBCheckBox.OnMouseMove 8651
DBCtrls.TDBCheckBox.OnMouseUp 8653
DBCtrls.TDBCheckBox.OnStartDock 8655
DBCtrls.TDBCheckBox.OnStartDrag 8656
DBCtrls.TDBCheckBox.ParentBiDiMode 8615
DBCtrls.TDBCheckBox.ParentColor 8615
DBCtrls.TDBCheckBox.ParentCtl3D 8616
DBCtrls.TDBCheckBox.ParentFont 8616
DBCtrls.TDBCheckBox.ParentShowHint 8616
DBCtrls.TDBCheckBox.PopupMenu 8617
DBCtrls.TDBCheckBox.ReadOnly 8622
DBCtrls.TDBCheckBox.ShowHint 8622
DBCtrls.TDBCheckBox.State 8623
DBCtrls.TDBCheckBox.TabOrder 8624
DBCtrls.TDBCheckBox.TabStop 8625
DBCtrls.TDBCheckBox.Toggle 8596
DBCtrls.TDBCheckBox.UpdateAction 8596
DBCtrls.TDBCheckBox.UseRightToLeftAlignment 8596
DBCtrls.TDBCheckBox.ValueChecked 8625
DBCtrls.TDBCheckBox.ValueUnchecked 8626
DBCtrls.TDBCheckBox.Visible 8626
DBCtrls.TDBCheckBox.WndProc 8597
DBCtrls.TDBCheckBox.WordWrap 8628
DBCtrls.TDBComboBox 8657
DBCtrls.TDBComboBox.Align 8678
DBCtrls.TDBComboBox.Anchors 8682
DBCtrls.TDBComboBox.AutoComplete 8682
DBCtrls.TDBComboBox.AutoDropDown 8683
DBCtrls.TDBComboBox.BevelEdges 8683
DBCtrls.TDBComboBox.BevelInner 8683
DBCtrls.TDBComboBox.BevelKind 8684
DBCtrls.TDBComboBox.BevelOuter 8684
DBCtrls.TDBComboBox.BevelWidth 8685
DBCtrls.TDBComboBox.BiDiMode 8685
DBCtrls.TDBComboBox.Change 8671
DBCtrls.TDBComboBox.CharCase 8686
DBCtrls.TDBComboBox.Click 8671
DBCtrls.TDBComboBox.Color 8686
DBCtrls.TDBComboBox.ComboWndProc 8671
DBCtrls.TDBComboBox.Constraints 8687
DBCtrls.TDBComboBox.Create 8669
DBCtrls.TDBComboBox.CreateWnd 8672
DBCtrls.TDBComboBox.Ctl3D 8688
DBCtrls.TDBComboBox.DataField 8688
DBCtrls.TDBComboBox.DataSource 8689
DBCtrls.TDBComboBox.Destroy 8670
DBCtrls.TDBComboBox.DragCursor 8689
DBCtrls.TDBComboBox.DragKind 8689
DBCtrls.TDBComboBox.DragMode 8690
DBCtrls.TDBComboBox.DropDown 8672
DBCtrls.TDBComboBox.DropDownCount 8691
DBCtrls.TDBComboBox.Enabled 8692
DBCtrls.TDBComboBox.ExecuteAction 8673
DBCtrls.TDBComboBox.Field 8692
DBCtrls.TDBComboBox.Font 8693
DBCtrls.TDBComboBox.ImeMode 8694
DBCtrls.TDBComboBox.ImeName 8694
DBCtrls.TDBComboBox.ItemHeight 8695
DBCtrls.TDBComboBox.Items 8695
DBCtrls.TDBComboBox.KeyDown 8673
DBCtrls.TDBComboBox.KeyPress 8674
DBCtrls.TDBComboBox.Loaded 8674
DBCtrls.TDBComboBox.Notification 8675
DBCtrls.TDBComboBox.OnChange 8710
DBCtrls.TDBComboBox.OnClick 8712
DBCtrls.TDBComboBox.OnContextPopup 8714
DBCtrls.TDBComboBox.OnDblClick 8715
DBCtrls.TDBComboBox.OnDragDrop 8715
DBCtrls.TDBComboBox.OnDragOver 8717
DBCtrls.TDBComboBox.OnDrawItem 8720
DBCtrls.TDBComboBox.OnDropDown 8720
DBCtrls.TDBComboBox.OnEndDock 8722
DBCtrls.TDBComboBox.OnEndDrag 8722
DBCtrls.TDBComboBox.OnEnter 8723

DBCtrls.TDBComboBox.OnExit 8726
DBCtrls.TDBComboBox.OnKeyDown 8727
DBCtrls.TDBComboBox.OnKeyPress 8730
DBCtrls.TDBComboBox.OnKeyUp 8731
DBCtrls.TDBComboBox.OnMeasureItem 8733
DBCtrls.TDBComboBox.OnMouseEnter 8733
DBCtrls.TDBComboBox.OnMouseLeave 8733
DBCtrls.TDBComboBox.OnStartDock 8734
DBCtrls.TDBComboBox.OnStartDrag 8734
DBCtrls.TDBComboBox.ParentBiDiMode 8696
DBCtrls.TDBComboBox.ParentColor 8696
DBCtrls.TDBComboBox.ParentCtl3D 8697
DBCtrls.TDBComboBox.ParentFont 8697
DBCtrls.TDBComboBox.ParentShowHint 8698
DBCtrls.TDBComboBox.PopupMenu 8698
DBCtrls.TDBComboBox.ReadOnly 8703
DBCtrls.TDBComboBox.SetItems 8675
DBCtrls.TDBComboBox.SetStyle 8676
DBCtrls.TDBComboBox.ShowHint 8703
DBCtrls.TDBComboBox.Sorted 8705
DBCtrls.TDBComboBox.Style 8706
DBCtrls.TDBComboBox.TabOrder 8707
DBCtrls.TDBComboBox.TabStop 8707
DBCtrls.TDBComboBox.Text 8708
DBCtrls.TDBComboBox.UpdateAction 8676
DBCtrls.TDBComboBox.UseRightToLeftAlignment 8676
DBCtrls.TDBComboBox.Visible 8708
DBCtrls.TDBComboBox.WndProc 8677
DBCtrls.TDBEdit 8735
DBCtrls.TDBEdit.Align 8754
DBCtrls.TDBEdit.Anchors 8758
DBCtrls.TDBEdit.AutoSelect 8758
DBCtrls.TDBEdit.AutoSize 8759
DBCtrls.TDBEdit.BevelEdges 8760
DBCtrls.TDBEdit.BevelInner 8760
DBCtrls.TDBEdit.BevelKind 8761
DBCtrls.TDBEdit.BevelOuter 8761
DBCtrls.TDBEdit.BevelWidth 8761
DBCtrls.TDBEdit.BiDiMode 8762
DBCtrls.TDBEdit.BorderStyle 8762
DBCtrls.TDBEdit.Change 8749
DBCtrls.TDBEdit.CharCase 8763
DBCtrls.TDBEdit.Color 8763
DBCtrls.TDBEdit.Constraints 8764
DBCtrls.TDBEdit.Create 8747
DBCtrls.TDBEdit.Ctl3D 8764
DBCtrls.TDBEdit.DataField 8765
DBCtrls.TDBEdit.DataSource 8765
DBCtrls.TDBEdit.Destroy 8748
DBCtrls.TDBEdit.DragCursor 8766
DBCtrls.TDBEdit.DragKind 8766
DBCtrls.TDBEdit.DragMode 8767
DBCtrls.TDBEdit.EditCanModify 8749
DBCtrls.TDBEdit.Enabled 8768
DBCtrls.TDBEdit.ExecuteAction 8750
DBCtrls.TDBEdit.Field 8769
DBCtrls.TDBEdit.Font 8769
DBCtrls.TDBEdit.ImeMode 8770
DBCtrls.TDBEdit.ImeName 8771
DBCtrls.TDBEdit.KeyDown 8750
DBCtrls.TDBEdit.KeyPress 8751
DBCtrls.TDBEdit.Loaded 8751
DBCtrls.TDBEdit.MaxLength 8771
DBCtrls.TDBEdit.Notification 8752
DBCtrls.TDBEdit.OnChange 8784
DBCtrls.TDBEdit.OnClick 8784
DBCtrls.TDBEdit.OnContextPopup 8786
DBCtrls.TDBEdit.OnDblClick 8787
DBCtrls.TDBEdit.OnDragDrop 8787
DBCtrls.TDBEdit.OnDragOver 8790
DBCtrls.TDBEdit.OnEndDock 8792
DBCtrls.TDBEdit.OnEndDrag 8792
DBCtrls.TDBEdit.OnEnter 8793
DBCtrls.TDBEdit.OnExit 8796
DBCtrls.TDBEdit.OnKeyDown 8797
DBCtrls.TDBEdit.OnKeyPress 8800
DBCtrls.TDBEdit.OnKeyUp 8801
DBCtrls.TDBEdit.OnMouseActivate 8803

DBCtrls.TDBEdit.OnMouseDown 8804
DBCtrls.TDBEdit.OnMouseEnter 8806
DBCtrls.TDBEdit.OnMouseLeave 8807
DBCtrls.TDBEdit.OnMouseMove 8807
DBCtrls.TDBEdit.OnMouseUp 8809
DBCtrls.TDBEdit.OnStartDock 8811
DBCtrls.TDBEdit.OnStartDrag 8812
DBCtrls.TDBEdit.ParentBiDiMode 8772
DBCtrls.TDBEdit.ParentColor 8772
DBCtrls.TDBEdit.ParentCtl3D 8773
DBCtrls.TDBEdit.ParentFont 8773
DBCtrls.TDBEdit.ParentShowHint 8774
DBCtrls.TDBEdit.PasswordChar 8774
DBCtrls.TDBEdit.PopupMenu 8774
DBCtrls.TDBEdit.ReadOnly 8779
DBCtrls.TDBEdit.Reset 8752
DBCtrls.TDBEdit.ShowHint 8779
DBCtrls.TDBEdit.TabOrder 8781
DBCtrls.TDBEdit.TabStop 8782
DBCtrls.TDBEdit.UpdateAction 8752
DBCtrls.TDBEdit.UseRightToLeftAlignment 8753
DBCtrls.TDBEdit.Visible 8782
DBCtrls.TDBImage 8813
DBCtrls.TDBImage.Align 8831
DBCtrls.TDBImage.Anchors 8835
DBCtrls.TDBImage.AutoDisplay 8835
DBCtrls.TDBImage.BorderStyle 8836
DBCtrls.TDBImage.Center 8836
DBCtrls.TDBImage.Color 8836
DBCtrls.TDBImage.Constraints 8837
DBCtrls.TDBImage.CopyToClipboard 8825
DBCtrls.TDBImage.Create 8824
DBCtrls.TDBImage.CreateParams 8825
DBCtrls.TDBImage.Ctl3D 8838
DBCtrls.TDBImage.CutToClipboard 8826
DBCtrls.TDBImage.DataField 8838
DBCtrls.TDBImage.DataSource 8839
DBCtrls.TDBImage.Destroy 8824
DBCtrls.TDBImage.DragCursor 8839
DBCtrls.TDBImage.DragKind 8839
DBCtrls.TDBImage.DragMode 8840
DBCtrls.TDBImage.Enabled 8841
DBCtrls.TDBImage.ExecuteAction 8826
DBCtrls.TDBImage.Field 8842
DBCtrls.TDBImage.Font 8842
DBCtrls.TDBImage.GetPalette 8827
DBCtrls.TDBImage.KeyDown 8827
DBCtrls.TDBImage.KeyPress 8828
DBCtrls.TDBImage.LoadPicture 8828
DBCtrls.TDBImage.Notification 8829
DBCtrls.TDBImage.OnClick 8856
DBCtrls.TDBImage.OnContextPopup 8858
DBCtrls.TDBImage.OnDblClick 8858
DBCtrls.TDBImage.OnDragDrop 8859
DBCtrls.TDBImage.OnDragOver 8861
DBCtrls.TDBImage.OnEndDock 8863
DBCtrls.TDBImage.OnEndDrag 8864
DBCtrls.TDBImage.OnEnter 8865
DBCtrls.TDBImage.OnExit 8867
DBCtrls.TDBImage.OnKeyDown 8869
DBCtrls.TDBImage.OnKeyPress 8872
DBCtrls.TDBImage.OnKeyUp 8873
DBCtrls.TDBImage.OnMouseActivate 8874
DBCtrls.TDBImage.OnMouseDown 8875
DBCtrls.TDBImage.OnMouseEnter 8878
DBCtrls.TDBImage.OnMouseLeave 8878
DBCtrls.TDBImage.OnMouseMove 8878
DBCtrls.TDBImage.OnMouseUp 8880
DBCtrls.TDBImage.OnStartDock 8883
DBCtrls.TDBImage.OnStartDrag 8883
DBCtrls.TDBImage.Paint 8829
DBCtrls.TDBImage.ParentColor 8843
DBCtrls.TDBImage.ParentCtl3D 8844
DBCtrls.TDBImage.ParentFont 8844
DBCtrls.TDBImage.ParentShowHint 8845
DBCtrls.TDBImage.PasteFromClipboard 8830
DBCtrls.TDBImage.Picture 8845
DBCtrls.TDBImage.PopupMenu 8846

DBCtrls.TDBImage.QuickDraw 8850
DBCtrls.TDBImage.ReadOnly 8850
DBCtrls.TDBImage.ShowHint 8851
DBCtrls.TDBImage.Stretch 8852
DBCtrls.TDBImage.TabOrder 8853
DBCtrls.TDBImage.TabStop 8853
DBCtrls.TDBImage.UpdateAction 8830
DBCtrls.TDBImage.Visible 8854
DBCtrls.TDBListBox 8884
DBCtrls.TDBListBox.Align 8902
DBCtrls.TDBListBox.Anchors 8905
DBCtrls.TDBListBox.AutoComplete 8906
DBCtrls.TDBListBox.BevelEdges 8906
DBCtrls.TDBListBox.BevelInner 8906
DBCtrls.TDBListBox.BevelKind 8907
DBCtrls.TDBListBox.BevelOuter 8907
DBCtrls.TDBListBox.BevelWidth 8908
DBCtrls.TDBListBox.BiDiMode 8908
DBCtrls.TDBListBox.BorderStyle 8909
DBCtrls.TDBListBox.Click 8897
DBCtrls.TDBListBox.Color 8909
DBCtrls.TDBListBox.Constraints 8910
DBCtrls.TDBListBox.Create 8896
DBCtrls.TDBListBox.Ctl3D 8910
DBCtrls.TDBListBox.DataField 8911
DBCtrls.TDBListBox.DataSource 8911
DBCtrls.TDBListBox.Destroy 8897
DBCtrls.TDBListBox.DragCursor 8912
DBCtrls.TDBListBox.DragKind 8912
DBCtrls.TDBListBox.DragMode 8913
DBCtrls.TDBListBox.Enabled 8914
DBCtrls.TDBListBox.ExecuteAction 8898
DBCtrls.TDBListBox.Field 8915
DBCtrls.TDBListBox.Font 8915
DBCtrls.TDBListBox.ImeMode 8916
DBCtrls.TDBListBox.ImeName 8917
DBCtrls.TDBListBox.IntegralHeight 8917
DBCtrls.TDBListBox.ItemHeight 8918
DBCtrls.TDBListBox.Items 8918
DBCtrls.TDBListBox.KeyDown 8898
DBCtrls.TDBListBox.KeyPress 8899
DBCtrls.TDBListBox.Notification 8899
DBCtrls.TDBListBox.OnClick 8931
DBCtrls.TDBListBox.OnContextPopup 8933
DBCtrls.TDBListBox.OnDblClick 8934
DBCtrls.TDBListBox.OnDragDrop 8934
DBCtrls.TDBListBox.OnDragOver 8936
DBCtrls.TDBListBox.OnDrawItem 8939
DBCtrls.TDBListBox.OnEndDock 8941
DBCtrls.TDBListBox.OnEndDrag 8941
DBCtrls.TDBListBox.OnEnter 8942
DBCtrls.TDBListBox.OnExit 8945
DBCtrls.TDBListBox.OnKeyDown 8946
DBCtrls.TDBListBox.OnKeyPress 8949
DBCtrls.TDBListBox.OnKeyUp 8950
DBCtrls.TDBListBox.OnMeasureItem 8952
DBCtrls.TDBListBox.OnMouseActivate 8953
DBCtrls.TDBListBox.OnMouseDown 8954
DBCtrls.TDBListBox.OnMouseEnter 8957
DBCtrls.TDBListBox.OnMouseLeave 8957
DBCtrls.TDBListBox.OnMouseMove 8957
DBCtrls.TDBListBox.OnMouseUp 8959
DBCtrls.TDBListBox.OnStartDock 8962
DBCtrls.TDBListBox.OnStartDrag 8962
DBCtrls.TDBListBox.ParentBiDiMode 8919
DBCtrls.TDBListBox.ParentColor 8919
DBCtrls.TDBListBox.ParentCtl3D 8920
DBCtrls.TDBListBox.ParentFont 8920
DBCtrls.TDBListBox.ParentShowHint 8920
DBCtrls.TDBListBox.PopupMenu 8921
DBCtrls.TDBListBox.ReadOnly 8926
DBCtrls.TDBListBox.ShowHint 8926
DBCtrls.TDBListBox.Style 8927
DBCtrls.TDBListBox.TabOrder 8928
DBCtrls.TDBListBox.TabStop 8929
DBCtrls.TDBListBox.UpdateAction 8900
DBCtrls.TDBListBox.UseRightToLeftAlignment 8900
DBCtrls.TDBListBox.Visible 8929

DBCtrls.TDBLookupComboBox 8963
DBCtrls.TDBLookupComboBox.Align 8984
DBCtrls.TDBLookupComboBox.Anchors 8987
DBCtrls.TDBLookupComboBox.BevelEdges 8988
DBCtrls.TDBLookupComboBox.BevelInner 8988
DBCtrls.TDBLookupComboBox.BevelKind 8989
DBCtrls.TDBLookupComboBox.BevelOuter 8989
DBCtrls.TDBLookupComboBox.BevelWidth 8990
DBCtrls.TDBLookupComboBox.BiDiMode 8990
DBCtrls.TDBLookupComboBox.CloseUp 8976
DBCtrls.TDBLookupComboBox.Color 8991
DBCtrls.TDBLookupComboBox.Constraints 8991
DBCtrls.TDBLookupComboBox.Create 8975
DBCtrls.TDBLookupComboBox.CreateParams 8976
DBCtrls.TDBLookupComboBox.Ctl3D 8992
DBCtrls.TDBLookupComboBox.DataField 8992
DBCtrls.TDBLookupComboBox.DataSource 8993
DBCtrls.TDBLookupComboBox.DragCursor 8993
DBCtrls.TDBLookupComboBox.DragKind 8994
DBCtrls.TDBLookupComboBox.DragMode 8994
DBCtrls.TDBLookupComboBox.DropDown 8977
DBCtrls.TDBLookupComboBox.DropDownAlign 8996
DBCtrls.TDBLookupComboBox.DropDownRows 8996
DBCtrls.TDBLookupComboBox.DropDownWidth 8996
DBCtrls.TDBLookupComboBox.Enabled 8997
DBCtrls.TDBLookupComboBox.ExecuteAction 8977
DBCtrls.TDBLookupComboBox.Font 8997
DBCtrls.TDBLookupComboBox.ImeMode 8998
DBCtrls.TDBLookupComboBox.ImeName 8999
DBCtrls.TDBLookupComboBox.KeyDown 8977
DBCtrls.TDBLookupComboBox.KeyField 9000
DBCtrls.TDBLookupComboBox.KeyPress 8978
DBCtrls.TDBLookupComboBox.KeyValue 9000
DBCtrls.TDBLookupComboBox.KeyValueChanged 8978
DBCtrls.TDBLookupComboBox.ListField 9001
DBCtrls.TDBLookupComboBox.ListFieldIndex 9001
DBCtrls.TDBLookupComboBox.ListLinkDataChanged 8979
DBCtrls.TDBLookupComboBox.ListSource 9002
DBCtrls.TDBLookupComboBox.ListVisible 9002
DBCtrls.TDBLookupComboBox.MouseDown 8979
DBCtrls.TDBLookupComboBox.MouseMove 8980
DBCtrls.TDBLookupComboBox.MouseUp 8980
DBCtrls.TDBLookupComboBox.NullValueKey 9003
DBCtrls.TDBLookupComboBox.OnClick 9015
DBCtrls.TDBLookupComboBox.OnCloseUp 9017
DBCtrls.TDBLookupComboBox.OnContextPopup 9017
DBCtrls.TDBLookupComboBox.OnDragDrop 9018
DBCtrls.TDBLookupComboBox.OnDragOver 9020
DBCtrls.TDBLookupComboBox.OnDropDown 9023
DBCtrls.TDBLookupComboBox.OnEndDock 9023
DBCtrls.TDBLookupComboBox.OnEndDrag 9024
DBCtrls.TDBLookupComboBox.OnEnter 9025
DBCtrls.TDBLookupComboBox.OnExit 9027
DBCtrls.TDBLookupComboBox.OnKeyDown 9029
DBCtrls.TDBLookupComboBox.OnKeyPress 9031
DBCtrls.TDBLookupComboBox.OnKeyUp 9033
DBCtrls.TDBLookupComboBox.OnMouseActivate 9034
DBCtrls.TDBLookupComboBox.OnMouseDown 9035
DBCtrls.TDBLookupComboBox.OnMouseEnter 9038
DBCtrls.TDBLookupComboBox.OnMouseLeave 9038
DBCtrls.TDBLookupComboBox.OnMouseMove 9038
DBCtrls.TDBLookupComboBox.OnMouseUp 9040
DBCtrls.TDBLookupComboBox.OnStartDock 9043
DBCtrls.TDBLookupComboBox.OnStartDrag 9043
DBCtrls.TDBLookupComboBox.Paint 8981
DBCtrls.TDBLookupComboBox.ParentBiDiMode 9003
DBCtrls.TDBLookupComboBox.ParentColor 9003
DBCtrls.TDBLookupComboBox.ParentCtl3D 9004
DBCtrls.TDBLookupComboBox.ParentFont 9004
DBCtrls.TDBLookupComboBox.ParentShowHint 9005
DBCtrls.TDBLookupComboBox.PopupMenu 9005
DBCtrls.TDBLookupComboBox.ReadOnly 9010
DBCtrls.TDBLookupComboBox.SetBounds 8981
DBCtrls.TDBLookupComboBox.ShowHint 9010
DBCtrls.TDBLookupComboBox.TabOrder 9012
DBCtrls.TDBLookupComboBox.TabStop 9012
DBCtrls.TDBLookupComboBox.Text 9013
DBCtrls.TDBLookupComboBox.UpdateAction 8982

DBCtrls.TDBLookupComboBox.UpdateListFields 8982
DBCtrls.TDBLookupComboBox.UseRightToLeftAlignment 8982
DBCtrls.TDBLookupComboBox.Visible 9013
DBCtrls.TDBLookupControl 9044
DBCtrls.TDBLookupControl.CanModify 9057
DBCtrls.TDBLookupControl.Create 9055
DBCtrls.TDBLookupControl.DataField 9062
DBCtrls.TDBLookupControl.DataLink 9063
DBCtrls.TDBLookupControl.DataSource 9063
DBCtrls.TDBLookupControl.Destroy 9056
DBCtrls.TDBLookupControl.ExecuteAction 9057
DBCtrls.TDBLookupControl.Field 9064
DBCtrls.TDBLookupControl.GetBorderSize 9058
DBCtrls.TDBLookupControl.GetTextHeight 9058
DBCtrls.TDBLookupControl.HasFocus 9064
DBCtrls.TDBLookupControl.KeyField 9064
DBCtrls.TDBLookupControl.KeyValue 9065
DBCtrls.TDBLookupControl.KeyValueChanged 9058
DBCtrls.TDBLookupControl.ListActive 9065
DBCtrls.TDBLookupControl.ListField 9066
DBCtrls.TDBLookupControl.ListFieldIndex 9067
DBCtrls.TDBLookupControl.ListFields 9067
DBCtrls.TDBLookupControl.ListLink 9067
DBCtrls.TDBLookupControl.ListLinkDataChanged 9058
DBCtrls.TDBLookupControl.ListSource 9068
DBCtrls.TDBLookupControl.LocateKey 9059
DBCtrls.TDBLookupControl.Notification 9059
DBCtrls.TDBLookupControl.NullValueKey 9068
DBCtrls.TDBLookupControl.ParentColor 9069
DBCtrls.TDBLookupControl.ProcessSearchKey 9060
DBCtrls.TDBLookupControl.ReadOnly 9069
DBCtrls.TDBLookupControl.SearchText 9069
DBCtrls.TDBLookupControl.SelectKeyValue 9060
DBCtrls.TDBLookupControl.TabStop 9070
DBCtrls.TDBLookupControl.UpdateAction 9060
DBCtrls.TDBLookupControl.UpdateDataFields 9061
DBCtrls.TDBLookupControl.UpdateListFields 9061
DBCtrls.TDBLookupListBox 9070
DBCtrls.TDBLookupListBox.Align 9091
DBCtrls.TDBLookupListBox.Anchors 9094
DBCtrls.TDBLookupListBox.BevelEdges 9095
DBCtrls.TDBLookupListBox.BevelInner 9095
DBCtrls.TDBLookupListBox.BevelKind 9095
DBCtrls.TDBLookupListBox.BevelOuter 9096
DBCtrls.TDBLookupListBox.BevelWidth 9096
DBCtrls.TDBLookupListBox.BiDiMode 9097
DBCtrls.TDBLookupListBox.BorderStyle 9097
DBCtrls.TDBLookupListBox.Color 9098
DBCtrls.TDBLookupListBox.Constraints 9098
DBCtrls.TDBLookupListBox.Create 9082
DBCtrls.TDBLookupListBox.CreateParams 9082
DBCtrls.TDBLookupListBox.CreateWnd 9083
DBCtrls.TDBLookupListBox.Ctl3D 9099
DBCtrls.TDBLookupListBox.DataField 9099
DBCtrls.TDBLookupListBox.DataSource 9100
DBCtrls.TDBLookupListBox.DragCursor 9100
DBCtrls.TDBLookupListBox.DragKind 9101
DBCtrls.TDBLookupListBox.DragMode 9101
DBCtrls.TDBLookupListBox.Enabled 9103
DBCtrls.TDBLookupListBox.ExecuteAction 9083
DBCtrls.TDBLookupListBox.Font 9103
DBCtrls.TDBLookupListBox.ImeMode 9104
DBCtrls.TDBLookupListBox.ImeName 9105
DBCtrls.TDBLookupListBox.KeyDown 9084
DBCtrls.TDBLookupListBox.KeyField 9105
DBCtrls.TDBLookupListBox.KeyPress 9085
DBCtrls.TDBLookupListBox.KeyValue 9106
DBCtrls.TDBLookupListBox.KeyValueChanged 9085
DBCtrls.TDBLookupListBox.ListField 9106
DBCtrls.TDBLookupListBox.ListFieldIndex 9107
DBCtrls.TDBLookupListBox.ListLinkDataChanged 9085
DBCtrls.TDBLookupListBox.ListSource 9108
DBCtrls.TDBLookupListBox.MouseDown 9086
DBCtrls.TDBLookupListBox.MouseMove 9086
DBCtrls.TDBLookupListBox.MouseUp 9087
DBCtrls.TDBLookupListBox.NullValueKey 9108
DBCtrls.TDBLookupListBox.OnClick 9121
DBCtrls.TDBLookupListBox.OnContextPopup 9123

DBCtrls.TDBLookupListBox.OnDblClick 9124
DBCtrls.TDBLookupListBox.OnDragDrop 9124
DBCtrls.TDBLookupListBox.OnDragOver 9126
DBCtrls.TDBLookupListBox.OnEndDock 9129
DBCtrls.TDBLookupListBox.OnEndDrag 9129
DBCtrls.TDBLookupListBox.OnEnter 9130
DBCtrls.TDBLookupListBox.OnExit 9132
DBCtrls.TDBLookupListBox.OnKeyDown 9134
DBCtrls.TDBLookupListBox.OnKeyPress 9137
DBCtrls.TDBLookupListBox.OnKeyUp 9138
DBCtrls.TDBLookupListBox.OnMouseActivate 9140
DBCtrls.TDBLookupListBox.OnMouseDown 9141
DBCtrls.TDBLookupListBox.OnMouseEnter 9143
DBCtrls.TDBLookupListBox.OnMouseLeave 9143
DBCtrls.TDBLookupListBox.OnMouseMove 9143
DBCtrls.TDBLookupListBox.OnMouseUp 9146
DBCtrls.TDBLookupListBox.OnStartDock 9148
DBCtrls.TDBLookupListBox.OnStartDrag 9148
DBCtrls.TDBLookupListBox.Paint 9087
DBCtrls.TDBLookupListBox.ParentBiDiMode 9109
DBCtrls.TDBLookupListBox.ParentColor 9109
DBCtrls.TDBLookupListBox.ParentCtl3D 9109
DBCtrls.TDBLookupListBox.ParentFont 9110
DBCtrls.TDBLookupListBox.ParentShowHint 9110
DBCtrls.TDBLookupListBox.PopupMenu 9111
DBCtrls.TDBLookupListBox.ReadOnly 9115
DBCtrls.TDBLookupListBox.RowCount 9116
DBCtrls.TDBLookupListBox.SelectedItem 9116
DBCtrls.TDBLookupListBox.SetBounds 9088
DBCtrls.TDBLookupListBox.ShowHint 9116
DBCtrls.TDBLookupListBox.TabOrder 9118
DBCtrls.TDBLookupListBox.TabStop 9118
DBCtrls.TDBLookupListBox.UpdateAction 9088
DBCtrls.TDBLookupListBox.UpdateListFields 9089
DBCtrls.TDBLookupListBox.UseRightToLeftAlignment 9089
DBCtrls.TDBLookupListBox.Visible 9119
DBCtrls.TDBMemo 9149
DBCtrls.TDBMemo.Align 9168
DBCtrls.TDBMemo.Alignment 9171
DBCtrls.TDBMemo.Anchors 9172
DBCtrls.TDBMemo.AutoDisplay 9172
DBCtrls.TDBMemo.BevelEdges 9173
DBCtrls.TDBMemo.BevelInner 9173
DBCtrls.TDBMemo.BevelKind 9174
DBCtrls.TDBMemo.BevelOuter 9174
DBCtrls.TDBMemo.BevelWidth 9175
DBCtrls.TDBMemo.BiDiMode 9175
DBCtrls.TDBMemo.BorderStyle 9175
DBCtrls.TDBMemo.Change 9162
DBCtrls.TDBMemo.Color 9176
DBCtrls.TDBMemo.Constraints 9177
DBCtrls.TDBMemo.Create 9161
DBCtrls.TDBMemo.Ctl3D 9177
DBCtrls.TDBMemo.DataField 9178
DBCtrls.TDBMemo.DataSource 9178
DBCtrls.TDBMemo.Destroy 9162
DBCtrls.TDBMemo.DragCursor 9178
DBCtrls.TDBMemo.DragKind 9179
DBCtrls.TDBMemo.DragMode 9179
DBCtrls.TDBMemo.Enabled 9181
DBCtrls.TDBMemo.ExecuteAction 9163
DBCtrls.TDBMemo.Field 9181
DBCtrls.TDBMemo.Font 9182
DBCtrls.TDBMemo.HideSelection 9183
DBCtrls.TDBMemo.ImeMode 9183
DBCtrls.TDBMemo.ImeName 9184
DBCtrls.TDBMemo.KeyDown 9163
DBCtrls.TDBMemo.KeyPress 9164
DBCtrls.TDBMemo.Loaded 9164
DBCtrls.TDBMemo.LoadMemo 9164
DBCtrls.TDBMemo.MaxLength 9184
DBCtrls.TDBMemo.Notification 9165
DBCtrls.TDBMemo.OnChange 9202
DBCtrls.TDBMemo.OnClick 9202
DBCtrls.TDBMemo.OnContextPopup 9204
DBCtrls.TDBMemo.OnDblClick 9205
DBCtrls.TDBMemo.OnDragDrop 9205
DBCtrls.TDBMemo.OnDragOver 9207

DBCtrls.TDBMemo.OnEndDock 9210
DBCtrls.TDBMemo.OnEndDrag 9210
DBCtrls.TDBMemo.OnEnter 9211
DBCtrls.TDBMemo.OnExit 9214
DBCtrls.TDBMemo.OnKeyDown 9215
DBCtrls.TDBMemo.OnKeyPress 9218
DBCtrls.TDBMemo.OnKeyUp 9219
DBCtrls.TDBMemo.OnMouseActivate 9221
DBCtrls.TDBMemo.OnMouseDown 9222
DBCtrls.TDBMemo.OnMouseEnter 9224
DBCtrls.TDBMemo.OnMouseLeave 9224
DBCtrls.TDBMemo.OnMouseMove 9225
DBCtrls.TDBMemo.OnMouseUp 9227
DBCtrls.TDBMemo.OnStartDock 9229
DBCtrls.TDBMemo.OnStartDrag 9230
DBCtrls.TDBMemo.ParentBiDiMode 9185
DBCtrls.TDBMemo.ParentColor 9185
DBCtrls.TDBMemo.ParentCtl3D 9186
DBCtrls.TDBMemo.ParentFont 9186
DBCtrls.TDBMemo.ParentShowHint 9187
DBCtrls.TDBMemo.PopupMenu 9187
DBCtrls.TDBMemo.ReadOnly 9192
DBCtrls.TDBMemo.ScrollBars 9192
DBCtrls.TDBMemo.ShowHint 9194
DBCtrls.TDBMemo.TabOrder 9196
DBCtrls.TDBMemo.TabStop 9196
DBCtrls.TDBMemo.UpdateAction 9165
DBCtrls.TDBMemo.UseRightToLeftAlignment 9166
DBCtrls.TDBMemo.Visible 9197
DBCtrls.TDBMemo.WantReturns 9198
DBCtrls.TDBMemo.WantTabs 9199
DBCtrls.TDBMemo.WndProc 9166
DBCtrls.TDBMemo.WordWrap 9199
DBCtrls.TDBNavigator 9230
DBCtrls.TDBNavigator.ActiveChanged 9245
DBCtrls.TDBNavigator.Align 9249
DBCtrls.TDBNavigator.Anchors 9253
DBCtrls.TDBNavigator.BeforeAction 9269
DBCtrls.TDBNavigator.BtnClick 9245
DBCtrls.TDBNavigator.Buttons 9244
DBCtrls.TDBNavigator.CalcMinSize 9245
DBCtrls.TDBNavigator.ConfirmDelete 9253
DBCtrls.TDBNavigator.Constraints 9254
DBCtrls.TDBNavigator.Create 9243
DBCtrls.TDBNavigator.Ctl3D 9254
DBCtrls.TDBNavigator.DataChanged 9246
DBCtrls.TDBNavigator.DataSource 9255
DBCtrls.TDBNavigator.Destroy 9244
DBCtrls.TDBNavigator.DragCursor 9255
DBCtrls.TDBNavigator.DragKind 9256
DBCtrls.TDBNavigator.DragMode 9256
DBCtrls.TDBNavigator.EditingChanged 9246
DBCtrls.TDBNavigator.Enabled 9257
DBCtrls.TDBNavigator.Flat 9258
DBCtrls.TDBNavigator.GetChildren 9246
DBCtrls.TDBNavigator.Hints 9258
DBCtrls.TDBNavigator.KeyDown 9247
DBCtrls.TDBNavigator.Loaded 9247
DBCtrls.TDBNavigator.Notification 9248
DBCtrls.TDBNavigator.OnClick 9269
DBCtrls.TDBNavigator.OnContextPopup 9270
DBCtrls.TDBNavigator.OnDblClick 9271
DBCtrls.TDBNavigator.OnDragDrop 9271
DBCtrls.TDBNavigator.OnDragOver 9273
DBCtrls.TDBNavigator.OnEndDock 9276
DBCtrls.TDBNavigator.OnEndDrag 9276
DBCtrls.TDBNavigator.OnEnter 9277
DBCtrls.TDBNavigator.OnExit 9279
DBCtrls.TDBNavigator.OnResize 9281
DBCtrls.TDBNavigator.OnStartDock 9282
DBCtrls.TDBNavigator.OnStartDrag 9282
DBCtrls.TDBNavigator.ParentCtl3D 9259
DBCtrls.TDBNavigator.ParentShowHint 9259
DBCtrls.TDBNavigator.PopupMenu 9260
DBCtrls.TDBNavigator.SetBounds 9248
DBCtrls.TDBNavigator.ShowHint 9264
DBCtrls.TDBNavigator.TabOrder 9266
DBCtrls.TDBNavigator.TabStop 9266

DBCtrls.TDBNavigator.Visible 9267
DBCtrls.TDBNavigator.VisibleButtons 9268
DBCtrls.TDBRadioGroup 9283
DBCtrls.TDBRadioGroup.Align 9300
DBCtrls.TDBRadioGroup.Anchors 9303
DBCtrls.TDBRadioGroup.BiDiMode 9304
DBCtrls.TDBRadioGroup.CanModify 9295
DBCtrls.TDBRadioGroup.Caption 9304
DBCtrls.TDBRadioGroup.Change 9296
DBCtrls.TDBRadioGroup.Click 9296
DBCtrls.TDBRadioGroup.Color 9307
DBCtrls.TDBRadioGroup.Columns 9308
DBCtrls.TDBRadioGroup.Constraints 9308
DBCtrls.TDBRadioGroup.Create 9294
DBCtrls.TDBRadioGroup.Ctl3D 9308
DBCtrls.TDBRadioGroup.DataField 9309
DBCtrls.TDBRadioGroup.DataLink 9309
DBCtrls.TDBRadioGroup.DataSource 9309
DBCtrls.TDBRadioGroup.Destroy 9295
DBCtrls.TDBRadioGroup.DragCursor 9310
DBCtrls.TDBRadioGroup.DragKind 9310
DBCtrls.TDBRadioGroup.DragMode 9311
DBCtrls.TDBRadioGroup.Enabled 9312
DBCtrls.TDBRadioGroup.ExecuteAction 9296
DBCtrls.TDBRadioGroup.Field 9313
DBCtrls.TDBRadioGroup.Font 9313
DBCtrls.TDBRadioGroup.ItemIndex 9314
DBCtrls.TDBRadioGroup.Items 9315
DBCtrls.TDBRadioGroup.KeyPress 9297
DBCtrls.TDBRadioGroup.Notification 9297
DBCtrls.TDBRadioGroup.OnChange 9328
DBCtrls.TDBRadioGroup.OnClick 9328
DBCtrls.TDBRadioGroup.OnContextPopup 9330
DBCtrls.TDBRadioGroup.OnDragDrop 9331
DBCtrls.TDBRadioGroup.OnDragOver 9333
DBCtrls.TDBRadioGroup.OnEndDock 9335
DBCtrls.TDBRadioGroup.OnEndDrag 9336
DBCtrls.TDBRadioGroup.OnEnter 9337
DBCtrls.TDBRadioGroup.OnExit 9339
DBCtrls.TDBRadioGroup.OnMouseEnter 9341
DBCtrls.TDBRadioGroup.OnMouseLeave 9341
DBCtrls.TDBRadioGroup.OnStartDock 9341
DBCtrls.TDBRadioGroup.OnStartDrag 9342
DBCtrls.TDBRadioGroup.ParentBackground 9315
DBCtrls.TDBRadioGroup.ParentBiDiMode 9316
DBCtrls.TDBRadioGroup.ParentColor 9316
DBCtrls.TDBRadioGroup.ParentCtl3D 9317
DBCtrls.TDBRadioGroup.ParentFont 9317
DBCtrls.TDBRadioGroup.ParentShowHint 9317
DBCtrls.TDBRadioGroup.PopupMenu 9318
DBCtrls.TDBRadioGroup.ReadOnly 9323
DBCtrls.TDBRadioGroup.ShowHint 9323
DBCtrls.TDBRadioGroup.TabOrder 9324
DBCtrls.TDBRadioGroup.TabStop 9325
DBCtrls.TDBRadioGroup.UpdateAction 9298
DBCtrls.TDBRadioGroup.UseRightToLeftAlignment 9298
DBCtrls.TDBRadioGroup.Value 9325
DBCtrls.TDBRadioGroup.Values 9326
DBCtrls.TDBRadioGroup.Visible 9326
DBCtrls.TDBRichEdit 9343
DBCtrls.TDBRichEdit.Align 9362
DBCtrls.TDBRichEdit.Alignment 9366
DBCtrls.TDBRichEdit.Anchors 9366
DBCtrls.TDBRichEdit.AutoDisplay 9367
DBCtrls.TDBRichEdit.BevelEdges 9368
DBCtrls.TDBRichEdit.BevelInner 9368
DBCtrls.TDBRichEdit.BevelKind 9368
DBCtrls.TDBRichEdit.BevelOuter 9369
DBCtrls.TDBRichEdit.BevelWidth 9369
DBCtrls.TDBRichEdit.BiDiMode 9370
DBCtrls.TDBRichEdit.BorderStyle 9370
DBCtrls.TDBRichEdit.Change 9357
DBCtrls.TDBRichEdit.Color 9370
DBCtrls.TDBRichEdit.Constraints 9371
DBCtrls.TDBRichEdit.Create 9356
DBCtrls.TDBRichEdit.Ctl3D 9372
DBCtrls.TDBRichEdit.DataField 9372
DBCtrls.TDBRichEdit.DataSource 9373

DBCtrls.TDBRichEdit.Destroy 9356
DBCtrls.TDBRichEdit.DragCursor 9373
DBCtrls.TDBRichEdit.DragKind 9374
DBCtrls.TDBRichEdit.DragMode 9374
DBCtrls.TDBRichEdit.Enabled 9375
DBCtrls.TDBRichEdit.ExecuteAction 9358
DBCtrls.TDBRichEdit.Field 9376
DBCtrls.TDBRichEdit.Font 9376
DBCtrls.TDBRichEdit.HideScrollBars 9377
DBCtrls.TDBRichEdit.HideSelection 9379
DBCtrls.TDBRichEdit.ImeMode 9381
DBCtrls.TDBRichEdit.ImeName 9381
DBCtrls.TDBRichEdit.KeyDown 9358
DBCtrls.TDBRichEdit.KeyPress 9359
DBCtrls.TDBRichEdit.Loaded 9359
DBCtrls.TDBRichEdit.LoadMemo 9359
DBCtrls.TDBRichEdit.MaxLength 9382
DBCtrls.TDBRichEdit.Notification 9360
DBCtrls.TDBRichEdit.OnChange 9401
DBCtrls.TDBRichEdit.OnClick 9401
DBCtrls.TDBRichEdit.OnContextPopup 9403
DBCtrls.TDBRichEdit.OnDblClick 9404
DBCtrls.TDBRichEdit.OnDragDrop 9404
DBCtrls.TDBRichEdit.OnDragOver 9406
DBCtrls.TDBRichEdit.OnEndDock 9409
DBCtrls.TDBRichEdit.OnEndDrag 9409
DBCtrls.TDBRichEdit.OnEnter 9410
DBCtrls.TDBRichEdit.OnExit 9413
DBCtrls.TDBRichEdit.OnKeyDown 9414
DBCtrls.TDBRichEdit.OnKeyPress 9417
DBCtrls.TDBRichEdit.OnKeyUp 9418
DBCtrls.TDBRichEdit.OnMouseActivate 9420
DBCtrls.TDBRichEdit.OnMouseDown 9421
DBCtrls.TDBRichEdit.OnMouseEnter 9423
DBCtrls.TDBRichEdit.OnMouseLeave 9423
DBCtrls.TDBRichEdit.OnMouseMove 9424
DBCtrls.TDBRichEdit.OnMouseUp 9426
DBCtrls.TDBRichEdit.OnProtectChange 9428
DBCtrls.TDBRichEdit.OnResizeRequest 9429
DBCtrls.TDBRichEdit.OnSaveClipboard 9430
DBCtrls.TDBRichEdit.OnSelectionChange 9430
DBCtrls.TDBRichEdit.OnStartDock 9431
DBCtrls.TDBRichEdit.OnStartDrag 9432
DBCtrls.TDBRichEdit.ParentBiDiMode 9382
DBCtrls.TDBRichEdit.ParentColor 9383
DBCtrls.TDBRichEdit.ParentCtl3D 9383
DBCtrls.TDBRichEdit.ParentFont 9384
DBCtrls.TDBRichEdit.ParentShowHint 9384
DBCtrls.TDBRichEdit.PlainText 9385
DBCtrls.TDBRichEdit.PopupMenu 9386
DBCtrls.TDBRichEdit.ReadOnly 9390
DBCtrls.TDBRichEdit.ScrollBars 9391
DBCtrls.TDBRichEdit.ShowHint 9393
DBCtrls.TDBRichEdit.TabOrder 9395
DBCtrls.TDBRichEdit.TabStop 9395
DBCtrls.TDBRichEdit.UpdateAction 9360
DBCtrls.TDBRichEdit.UseRightToLeftAlignment 9361
DBCtrls.TDBRichEdit.Visible 9396
DBCtrls.TDBRichEdit.WantReturns 9397
DBCtrls.TDBRichEdit.WantTabs 9398
DBCtrls.TDBRichEdit.WordWrap 9398
DBCtrls.TDBText 9433
DBCtrls.TDBText.Align 9445
DBCtrls.TDBText.Alignment 9449
DBCtrls.TDBText.Anchors 9449
DBCtrls.TDBText.AutoSize 9450
DBCtrls.TDBText.BiDiMode 9450
DBCtrls.TDBText.Color 9451
DBCtrls.TDBText.Constraints 9452
DBCtrls.TDBText.Create 9441
DBCtrls.TDBText.DataField 9452
DBCtrls.TDBText.DataSource 9453
DBCtrls.TDBText.Destroy 9441
DBCtrls.TDBText.DragCursor 9453
DBCtrls.TDBText.DragKind 9453
DBCtrls.TDBText.DragMode 9454
DBCtrls.TDBText.Enabled 9455
DBCtrls.TDBText.ExecuteAction 9442

DBCtrls.TDBText.Field 9456
DBCtrls.TDBText.Font 9456
DBCtrls.TDBText.GetLabelText 9442
DBCtrls.TDBText.Loaded 9443
DBCtrls.TDBText.Notification 9443
DBCtrls.TDBText.OnClick 9468
DBCtrls.TDBText.OnContextPopup 9470
DBCtrls.TDBText.OnDblClick 9471
DBCtrls.TDBText.OnDragDrop 9471
DBCtrls.TDBText.OnDragOver 9473
DBCtrls.TDBText.OnEndDock 9476
DBCtrls.TDBText.OnEndDrag 9476
DBCtrls.TDBText.OnMouseActivate 9477
DBCtrls.TDBText.OnMouseDown 9478
DBCtrls.TDBText.OnMouseEnter 9480
DBCtrls.TDBText.OnMouseLeave 9481
DBCtrls.TDBText.OnMouseMove 9481
DBCtrls.TDBText.OnMouseUp 9483
DBCtrls.TDBText.OnStartDock 9485
DBCtrls.TDBText.OnStartDrag 9486
DBCtrls.TDBText.ParentBiDiMode 9457
DBCtrls.TDBText.ParentColor 9458
DBCtrls.TDBText.ParentFont 9458
DBCtrls.TDBText.ParentShowHint 9459
DBCtrls.TDBText.PopupMenu 9459
DBCtrls.TDBText.SetAutoSize 9443
DBCtrls.TDBText.ShowHint 9464
DBCtrls.TDBText.Transparent 9465
DBCtrls.TDBText.UpdateAction 9444
DBCtrls.TDBText.UseRightToLeftAlignment 9444
DBCtrls.TDBText.Visible 9466
DBCtrls.TDBText.WordWrap 9467
DBCtrls.TDropDownAlign 9538
DBCtrls.TDropDownAlign enumeration 9538
DBCtrls.TFieldDataLink 9487
DBCtrls.TFieldDataLink.ActiveChanged 9490
DBCtrls.TFieldDataLink.CanModify 9494
DBCtrls.TFieldDataLink.Control 9495
DBCtrls.TFieldDataLink.Create 9489
DBCtrls.TFieldDataLink.DataEvent 9491
DBCtrls.TFieldDataLink.Edit 9491
DBCtrls.TFieldDataLink.Editing 9495
DBCtrls.TFieldDataLink.EditingChanged 9491
DBCtrls.TFieldDataLink.Field 9496
DBCtrls.TFieldDataLink.FieldName 9496
DBCtrls.TFieldDataLink.FocusControl 9492
DBCtrls.TFieldDataLink.LayoutChanged 9492
DBCtrls.TFieldDataLink.Modified 9492
DBCtrls.TFieldDataLink.OnActiveChange 9497
DBCtrls.TFieldDataLink.OnDataChange 9497
DBCtrls.TFieldDataLink.OnEditingChange 9497
DBCtrls.TFieldDataLink.OnUpdateData 9498
DBCtrls.TFieldDataLink.RecordChanged 9493
DBCtrls.TFieldDataLink.Reset 9493
DBCtrls.TFieldDataLink.UpdateData 9494
DBCtrls.TListSourceLink 9498
DBCtrls.TListSourceLink.ActiveChanged 9501
DBCtrls.TListSourceLink.Create 9500
DBCtrls.TListSourceLink.DataSetChanged 9502
DBCtrls.TListSourceLink.LayoutChanged 9502
DBCtrls.TNavButton 9502
DBCtrls.TNavButton.Destroy 9511
DBCtrls.TNavButton.Index 9513
DBCtrls.TNavButton.MouseDown 9511
DBCtrls.TNavButton.MouseUp 9512
DBCtrls.TNavButton.NavStyle 9513
DBCtrls.TNavButton.Paint 9512
DBCtrls.TNavButtonSet 9541
DBCtrls.TNavButtonSet type 9541
DBCtrls.TNavButtonStyle 9541
DBCtrls.TNavButtonStyle type 9541
DBCtrls.TNavDataLink 9513
DBCtrls.TNavDataLink.ActiveChanged 9516
DBCtrls.TNavDataLink.Create 9515
DBCtrls.TNavDataLink.DataSetChanged 9517
DBCtrls.TNavDataLink.Destroy 9516
DBCtrls.TNavDataLink.EditingChanged 9517
DBCtrls.TNavGlyph 9539

DBCtrls.TNavGlyph enumeration 9539
DBCtrls.TNavigateBtn 9539
DBCtrls.TNavigateBtn enumeration 9539
DBCtrls.TPaintControl 9518
DBCtrls.TPaintControl.Create 9518
DBCtrls.TPaintControl.Ctl3DButton 9520
DBCtrls.TPaintControl.Destroy 9519
DBCtrls.TPaintControl.DestroyHandle 9519
DBCtrls.TPaintControl.Handle 9520
DBCtrls.TPopupDataList 9521
DBCtrls.TPopupDataList.Create 9534
DBCtrls.TPopupDataList.CreateParams 9534
dbDeleteAlias enumeration member 10253
dbDeleteDriver enumeration member 10253
DbExcept 9545
DbExcept namespace 9545
    Classes 9546
    Variables 9572
DbExcept.DbEngineErrorDlg 9572
DbExcept.DbEngineErrorDlg variable 9572
DbExcept.TDbEngineErrorDlg 9546
DbExcept.TDbEngineErrorDlg.BackBtn 9564
DbExcept.TDbEngineErrorDlg.BackClick 9568
DbExcept.TDbEngineErrorDlg.BasicPanel 9564
DbExcept.TDbEngineErrorDlg.BDELabel 9564
DbExcept.TDbEngineErrorDlg.ButtonPanel 9564
DbExcept.TDbEngineErrorDlg.DbCatSub 9564
DbExcept.TDbEngineErrorDlg.DbException 9571
DbExcept.TDbEngineErrorDlg.DbMessageText 9565
DbExcept.TDbEngineErrorDlg.DbResult 9565
DbExcept.TDbEngineErrorDlg.DetailsBtn 9565
DbExcept.TDbEngineErrorDlg.DetailsBtnClick 9568
DbExcept.TDbEngineErrorDlg.DetailsPanel 9565
DbExcept.TDbEngineErrorDlg.ErrorText 9566
DbExcept.TDbEngineErrorDlg.FormCreate 9568
DbExcept.TDbEngineErrorDlg.FormDestroy 9569
DbExcept.TDbEngineErrorDlg.FormShow 9569
DbExcept.TDbEngineErrorDlg.HookExceptions 9569
DbExcept.TDbEngineErrorDlg.IconImage 9566
DbExcept.TDbEngineErrorDlg.IconPanel 9566
DbExcept.TDbEngineErrorDlg.NativeLabel 9566
DbExcept.TDbEngineErrorDlg.NativeResult 9566
DbExcept.TDbEngineErrorDlg.NextBtn 9567
DbExcept.TDbEngineErrorDlg.NextClick 9570
DbExcept.TDbEngineErrorDlg.OKBtn 9567
DbExcept.TDbEngineErrorDlg.RightPanel 9567
DbExcept.TDbEngineErrorDlg.ShowException 9570
DbExcept.TDbEngineErrorDlg.TopPanel 9567
DBGrids 9573
DBGrids namespace 9573
    Classes 9575
    Constants 9772
    Structs, Records, Enums 9765
    Types 9769
DBGrids.cm_DeferLayout 9772
DBGrids.cm_DeferLayout constant 9772
DBGrids.ColumnTitleValues 9772
DBGrids.ColumnTitleValues constant 9772
DBGrids.IndicatorWidth 9772
DBGrids.IndicatorWidth constant 9772
DBGrids.TBookmarkList 9575
DBGrids.TBookmarkList.Clear 9577
DBGrids.TBookmarkList.Compare 9578
DBGrids.TBookmarkList.Count 9580
DBGrids.TBookmarkList.Create 9576
DBGrids.TBookmarkList.CurrentRow 9578
DBGrids.TBookmarkList.CurrentRowSelected 9581
DBGrids.TBookmarkList.Delete 9578
DBGrids.TBookmarkList.Destroy 9576
DBGrids.TBookmarkList.Find 9578
DBGrids.TBookmarkList.IndexOf 9579
DBGrids.TBookmarkList.Items 9581
DBGrids.TBookmarkList.LinkActive 9579
DBGrids.TBookmarkList.Refresh 9580
DBGrids.TColumn 9583
DBGrids.TColumn.Alignment 9592
DBGrids.TColumn.Assign 9586
DBGrids.TColumn.AssignedValues 9593

DBGrids.TColumn.ButtonStyle 9593
DBGrids.TColumn.Color 9594
DBGrids.TColumn.Create 9585
DBGrids.TColumn.CreateTitle 9587
DBGrids.TColumn.DefaultAlignment 9587
DBGrids.TColumn.DefaultColor 9587
DBGrids.TColumn.DefaultFont 9588
DBGrids.TColumn.DefaultImeMode 9588
DBGrids.TColumn.DefaultImeName 9588
DBGrids.TColumn.DefaultReadOnly 9589
DBGrids.TColumn.DefaultWidth 9589
DBGrids.TColumn.Depth 9590
DBGrids.TColumn.Destroy 9585
DBGrids.TColumn.DropDownRows 9594
DBGrids.TColumn.Expandable 9594
DBGrids.TColumn.Expanded 9595
DBGrids.TColumn.Field 9595
DBGrids.TColumn.FieldName 9595
DBGrids.TColumn.Font 9596
DBGrids.TColumn.GetDisplayName 9590
DBGrids.TColumn.GetGrid 9590
DBGrids.TColumn.Grid 9596
DBGrids.TColumn.ImeMode 9597
DBGrids.TColumn.ImeName 9597
DBGrids.TColumn.IsStored 9598
DBGrids.TColumn.ParentColumn 9598
DBGrids.TColumn.PickList 9598
DBGrids.TColumn.PopupMenu 9599
DBGrids.TColumn.ReadOnly 9599
DBGrids.TColumn.RefreshDefaultFont 9591
DBGrids.TColumn.RestoreDefaults 9591
DBGrids.TColumn.SetIndex 9591
DBGrids.TColumn.Showing 9599
DBGrids.TColumn.Title 9600
DBGrids.TColumn.Visible 9600
DBGrids.TColumn.Width 9600
DBGrids.TColumnButtonStyle 9765
DBGrids.TColumnButtonStyle enumeration 9765
DBGrids.TColumnClass 9769
DBGrids.TColumnClass type 9769
DBGrids.TColumnTitle 9601
DBGrids.TColumnTitle.Alignment 9606
DBGrids.TColumnTitle.Assign 9604
DBGrids.TColumnTitle.Caption 9607
DBGrids.TColumnTitle.Color 9607
DBGrids.TColumnTitle.Column 9607
DBGrids.TColumnTitle.Create 9603
DBGrids.TColumnTitle.DefaultAlignment 9604
DBGrids.TColumnTitle.DefaultCaption 9604
DBGrids.TColumnTitle.DefaultColor 9605
DBGrids.TColumnTitle.DefaultFont 9605
DBGrids.TColumnTitle.Destroy 9603
DBGrids.TColumnTitle.Font 9608
DBGrids.TColumnTitle.RefreshDefaultFont 9606
DBGrids.TColumnTitle.RestoreDefaults 9606
DBGrids.TColumnValue 9766
DBGrids.TColumnValue enumeration 9766
DBGrids.TColumnValues 9769
DBGrids.TColumnValues type 9769
DBGrids.TCustomDBGrid 9608
DBGrids.TCustomDBGrid.AcquireLayoutLock 9625
DBGrids.TCustomDBGrid.BeginColumnDrag 9625
DBGrids.TCustomDBGrid.BeginLayout 9626
DBGrids.TCustomDBGrid.BeginUpdate 9627
DBGrids.TCustomDBGrid.CalcSizingState 9627
DBGrids.TCustomDBGrid.CalcTitleRect 9627
DBGrids.TCustomDBGrid.CancelLayout 9628
DBGrids.TCustomDBGrid.CanEditAcceptKey 9628
DBGrids.TCustomDBGrid.CanEditModify 9629
DBGrids.TCustomDBGrid.CanEditShow 9629
DBGrids.TCustomDBGrid.CellClick 9629
DBGrids.TCustomDBGrid.CheckColumnDrag 9630
DBGrids.TCustomDBGrid.ColEnter 9630
DBGrids.TCustomDBGrid.ColExit 9630
DBGrids.TCustomDBGrid.ColumnAtDepth 9631
DBGrids.TCustomDBGrid.ColumnMoved 9631
DBGrids.TCustomDBGrid.Columns 9653
DBGrids.TCustomDBGrid.ColWidthsChanged 9631

DBGrids.TCustomDBGrid.Create 9622
DBGrids.TCustomDBGrid.CreateColumns 9632
DBGrids.TCustomDBGrid.CreateDataLink 9632
DBGrids.TCustomDBGrid.CreateEditor 9632
DBGrids.TCustomDBGrid.CreateWnd 9633
DBGrids.TCustomDBGrid.DataLink 9654
DBGrids.TCustomDBGrid.DataSource 9655
DBGrids.TCustomDBGrid.DataToRawColumn 9633
DBGrids.TCustomDBGrid.DefaultDrawColumnCell 9634
DBGrids.TCustomDBGrid.DefaultDrawDataCell 9634
DBGrids.TCustomDBGrid.DefaultDrawing 9655
DBGrids.TCustomDBGrid.DefaultHandler 9635
DBGrids.TCustomDBGrid.DeferLayout 9635
DBGrids.TCustomDBGrid.DefineFieldMap 9636
DBGrids.TCustomDBGrid.DefineProperties 9636
DBGrids.TCustomDBGrid.Destroy 9622
DBGrids.TCustomDBGrid.DoMouseWheelDown 9636
DBGrids.TCustomDBGrid.DoMouseWheelUp 9637
DBGrids.TCustomDBGrid.DrawCell 9637
DBGrids.TCustomDBGrid.DrawColumnCell 9638
DBGrids.TCustomDBGrid.DrawDataCell 9638
DBGrids.TCustomDBGrid.EditButtonClick 9638
DBGrids.TCustomDBGrid.EditorMode 9656
DBGrids.TCustomDBGrid.EndColumnDrag 9639
DBGrids.TCustomDBGrid.EndLayout 9639
DBGrids.TCustomDBGrid.EndUpdate 9640
DBGrids.TCustomDBGrid.ExecuteAction 9640
DBGrids.TCustomDBGrid.FAcquireFocus 9623
DBGrids.TCustomDBGrid.FieldCount 9656
DBGrids.TCustomDBGrid.Fields 9657
DBGrids.TCustomDBGrid.FUpdateFields 9623
DBGrids.TCustomDBGrid.GetColField 9640
DBGrids.TCustomDBGrid.GetEditLimit 9641
DBGrids.TCustomDBGrid.GetEditMask 9641
DBGrids.TCustomDBGrid.GetEditStyle 9642
DBGrids.TCustomDBGrid.GetEditText 9642
DBGrids.TCustomDBGrid.GetFieldValue 9643
DBGrids.TCustomDBGrid.HighlightCell 9643
DBGrids.TCustomDBGrid.IndicatorOffset 9658
DBGrids.TCustomDBGrid.InvalidateTitles 9644
DBGrids.TCustomDBGrid.KeyDown 9644
DBGrids.TCustomDBGrid.KeyPress 9645
DBGrids.TCustomDBGrid.LayoutChanged 9645
DBGrids.TCustomDBGrid.LayoutLock 9659
DBGrids.TCustomDBGrid.LinkActive 9646
DBGrids.TCustomDBGrid.Loaded 9646
DBGrids.TCustomDBGrid.MouseDown 9647
DBGrids.TCustomDBGrid.MouseUp 9647
DBGrids.TCustomDBGrid.Notification 9647
DBGrids.TCustomDBGrid.OnCellClick 9664
DBGrids.TCustomDBGrid.OnColEnter 9665
DBGrids.TCustomDBGrid.OnColExit 9665
DBGrids.TCustomDBGrid.OnColumnMoved 9666
DBGrids.TCustomDBGrid.OnDrawColumnCell 9666
DBGrids.TCustomDBGrid.OnDrawDataCell 9667
DBGrids.TCustomDBGrid.OnEditButtonClick 9668
DBGrids.TCustomDBGrid.OnTitleClick 9668
DBGrids.TCustomDBGrid.Options 9659
DBGrids.TCustomDBGrid.ParentColor 9660
DBGrids.TCustomDBGrid.RawToDataColumn 9648
DBGrids.TCustomDBGrid.ReadOnly 9660
DBGrids.TCustomDBGrid.Scroll 9648
DBGrids.TCustomDBGrid.SelectedField 9661
DBGrids.TCustomDBGrid.SelectedIndex 9661
DBGrids.TCustomDBGrid.SelectedRows 9662
DBGrids.TCustomDBGrid.SetColumnAttributes 9649
DBGrids.TCustomDBGrid.SetEditText 9649
DBGrids.TCustomDBGrid.ShowPopupEditor 9649
DBGrids.TCustomDBGrid.StoreColumns 9650
DBGrids.TCustomDBGrid.TimedScroll 9650
DBGrids.TCustomDBGrid.TitleClick 9651
DBGrids.TCustomDBGrid.TitleFont 9663
DBGrids.TCustomDBGrid.TopLeftChanged 9651
DBGrids.TCustomDBGrid.UpdateAction 9651
DBGrids.TCustomDBGrid.UpdateLock 9664
DBGrids.TCustomDBGrid.UpdateScrollBar 9652
DBGrids.TCustomDBGrid.UseRightToLeftAlignmentForField 9652
DBGrids.TCustomDBGrid.ValidFieldIndex 9652

DBGrids.TDBGrid 9668
DBGrids.TDBGrid.Align 9686
DBGrids.TDBGrid.Anchors 9689
DBGrids.TDBGrid.BiDiMode 9690
DBGrids.TDBGrid.BorderStyle 9690
DBGrids.TDBGrid.Canvas 9690
DBGrids.TDBGrid.Color 9692
DBGrids.TDBGrid.Columns 9693
DBGrids.TDBGrid.Constraints 9694
DBGrids.TDBGrid.Ctl3D 9694
DBGrids.TDBGrid.DataSource 9695
DBGrids.TDBGrid.DefaultDrawing 9696
DBGrids.TDBGrid.DragCursor 9696
DBGrids.TDBGrid.DragKind 9697
DBGrids.TDBGrid.DragMode 9697
DBGrids.TDBGrid.Enabled 9699
DBGrids.TDBGrid.FixedColor 9699
DBGrids.TDBGrid.Font 9699
DBGrids.TDBGrid.ImeMode 9700
DBGrids.TDBGrid.ImeName 9701
DBGrids.TDBGrid.OnCellClick 9715
DBGrids.TDBGrid.OnColEnter 9716
DBGrids.TDBGrid.OnColExit 9716
DBGrids.TDBGrid.OnColumnMoved 9717
DBGrids.TDBGrid.OnDblClick 9717
DBGrids.TDBGrid.OnDragDrop 9717
DBGrids.TDBGrid.OnDragOver 9720
DBGrids.TDBGrid.OnDrawColumnCell 9722
DBGrids.TDBGrid.OnDrawDataCell 9723
DBGrids.TDBGrid.OnEditButtonClick 9723
DBGrids.TDBGrid.OnEndDock 9724
DBGrids.TDBGrid.OnEndDrag 9724
DBGrids.TDBGrid.OnEnter 9725
DBGrids.TDBGrid.OnExit 9728
DBGrids.TDBGrid.OnKeyDown 9729
DBGrids.TDBGrid.OnKeyPress 9732
DBGrids.TDBGrid.OnKeyUp 9733
DBGrids.TDBGrid.OnMouseActivate 9735
DBGrids.TDBGrid.OnMouseDown 9736
DBGrids.TDBGrid.OnMouseEnter 9738
DBGrids.TDBGrid.OnMouseLeave 9738
DBGrids.TDBGrid.OnMouseMove 9738
DBGrids.TDBGrid.OnMouseUp 9741
DBGrids.TDBGrid.OnMouseWheel 9743
DBGrids.TDBGrid.OnMouseWheelDown 9743
DBGrids.TDBGrid.OnMouseWheelUp 9744
DBGrids.TDBGrid.OnStartDock 9744
DBGrids.TDBGrid.OnStartDrag 9745
DBGrids.TDBGrid.OnTitleClick 9745
DBGrids.TDBGrid.Options 9702
DBGrids.TDBGrid.ParentBiDiMode 9702
DBGrids.TDBGrid.ParentColor 9702
DBGrids.TDBGrid.ParentCtl3D 9703
DBGrids.TDBGrid.ParentFont 9703
DBGrids.TDBGrid.ParentShowHint 9704
DBGrids.TDBGrid.PopupMenu 9704
DBGrids.TDBGrid.ReadOnly 9709
DBGrids.TDBGrid.SelectedRows 9709
DBGrids.TDBGrid.ShowHint 9710
DBGrids.TDBGrid.TabOrder 9712
DBGrids.TDBGrid.TabStop 9712
DBGrids.TDBGrid.TitleFont 9713
DBGrids.TDBGrid.Visible 9713
DBGrids.TDBGridClickEvent 9770
DBGrids.TDBGridClickEvent type 9770
DBGrids.TDBGridColumns 9746
DBGrids.TDBGridColumns.Add 9749
DBGrids.TDBGridColumns.Create 9748
DBGrids.TDBGridColumns.GetOwner 9749
DBGrids.TDBGridColumns.Grid 9752
DBGrids.TDBGridColumns.Items 9753
DBGrids.TDBGridColumns.LoadFromFile 9750
DBGrids.TDBGridColumns.LoadFromStream 9750
DBGrids.TDBGridColumns.RebuildColumns 9750
DBGrids.TDBGridColumns.RestoreDefaults 9751
DBGrids.TDBGridColumns.SaveToFile 9751
DBGrids.TDBGridColumns.SaveToStream 9751
DBGrids.TDBGridColumns.State 9753

DBGrids.TDBGridColumns.Update 9752
DBGrids.TDBGridColumnsState 9767
DBGrids.TDBGridColumnsState enumeration 9767
DBGrids.TDBGridOption 9767
DBGrids.TDBGridOption enumeration 9767
DBGrids.TDBGridOptions 9770
DBGrids.TDBGridOptions type 9770
DBGrids.TDrawColumnCellEvent 9770
DBGrids.TDrawColumnCellEvent type 9770
DBGrids.TDrawDataCellEvent 9771
DBGrids.TDrawDataCellEvent type 9771
DBGrids.TGridDataLink 9754
DBGrids.TGridDataLink.ActiveChanged 9757
DBGrids.TGridDataLink.AddMapping 9757
DBGrids.TGridDataLink.BuildAggMap 9757
DBGrids.TGridDataLink.ClearMapping 9758
DBGrids.TGridDataLink.Create 9755
DBGrids.TGridDataLink.DataSetChanged 9758
DBGrids.TGridDataLink.DataSetScrolled 9758
DBGrids.TGridDataLink.DefaultFields 9762
DBGrids.TGridDataLink.Destroy 9756
DBGrids.TGridDataLink.EditingChanged 9759
DBGrids.TGridDataLink.FieldCount 9763
DBGrids.TGridDataLink.Fields 9763
DBGrids.TGridDataLink.FocusControl 9759
DBGrids.TGridDataLink.GetMappedIndex 9760
DBGrids.TGridDataLink.Grid 9764
DBGrids.TGridDataLink.IsAggRow 9760
DBGrids.TGridDataLink.LayoutChanged 9760
DBGrids.TGridDataLink.Modified 9761
DBGrids.TGridDataLink.RecordChanged 9761
DBGrids.TGridDataLink.Reset 9761
DBGrids.TGridDataLink.SparseMap 9764
DBGrids.TGridDataLink.UpdateData 9762
DBLogDlg 9775
DBLogDlg namespace 9775
    Classes 9776
    Functions 9797
DBLogDlg.LoginDialog 9797
DBLogDlg.LoginDialog function 9797
DBLogDlg.LoginDialogEx 9798
DBLogDlg.LoginDialogEx function 9798
DBLogDlg.RemoteLoginDialog 9798
DBLogDlg.RemoteLoginDialog function 9798
DBLogDlg.SetCursorType 9799
DBLogDlg.SetCursorType function 9799
DBLogDlg.TLoginDialog 9776
DBLogDlg.TLoginDialog.Bevel 9793
DBLogDlg.TLoginDialog.CancelButton 9793
DBLogDlg.TLoginDialog.DatabaseName 9793
DBLogDlg.TLoginDialog.FormShow 9795
DBLogDlg.TLoginDialog.Label1 9794
DBLogDlg.TLoginDialog.Label2 9794
DBLogDlg.TLoginDialog.Label3 9794
DBLogDlg.TLoginDialog.OKButton 9794
DBLogDlg.TLoginDialog.Panel 9794
DBLogDlg.TLoginDialog.Panel1 9795
DBLogDlg.TLoginDialog.Password 9795
DBLogDlg.TLoginDialog.UserName 9795
DBOleCtl 9801
DBOleCtl namespace 9801
    Classes 9802
DBOleCtl.TDataBindings 9802
DBOleCtl.TDataBindings.Add 9804
DBOleCtl.TDataBindings.Create 9804
DBOleCtl.TDataBindings.GetItemByDispID 9805
DBOleCtl.TDataBindings.GetOwner 9805
DBOleCtl.TDataBindings.Items 9806
DBOleCtl.TDataBindings.Update 9805
DBOleCtl.TDataBindItem 9806
DBOleCtl.TDataBindItem.Create 9808
DBOleCtl.TDataBindItem.DataField 9809
DBOleCtl.TDataBindItem.Destroy 9808
DBOleCtl.TDataBindItem.DispID 9809
DBOleCtl.TDBOleControl 9810
DBOleCtl.TDBOleControl.Create 9823
DBOleCtl.TDBOleControl.DataBindings 9825
DBOleCtl.TDBOleControl.DataSource 9825

DBOleCtl.TDBOleControl.Destroy 9823
DBOleCtl.TDBOleControl.Notification 9824
DBOleCtl.TDBOleControl.OnChanged 9824
DBOleCtl.TDBOleControl.OnRequestEdit 9825
dbOpen enumeration member 10253
DBPWDlg 9827
DBPWDlg namespace 9827
Classes 9828
Functions 9849
DBPWDlg.PasswordDialog 9849
DBPWDlg.PasswordDialog function 9849
DBPWDlg.TPasswordDialog 9828
DBPWDlg.TPasswordDialog.AddButton 9845
DBPWDlg.TPasswordDialog.AddButtonClick 9847
DBPWDlg.TPasswordDialog.CancelButton 9845
DBPWDlg.TPasswordDialog.Edit 9845
DBPWDlg.TPasswordDialog.EditChange 9847
DBPWDlg.TPasswordDialog.GroupBox1 9846
DBPWDlg.TPasswordDialog.OKButton 9846
DBPWDlg.TPasswordDialog.OKButtonClick 9848
DBPWDlg.TPasswordDialog.RemoveAllButton 9846
DBPWDlg.TPasswordDialog.RemoveAllButtonClick 9848
DBPWDlg.TPasswordDialog.RemoveButton 9846
DBPWDlg.TPasswordDialog.RemoveButtonClick 9848
dbRemove enumeration member 10253
DBTables 9851
DBTables namespace 9851
Classes 9854
Constants 10267
Functions 10244
Structs, Records, Enums 10251
Types 10258
Variables 10265
DBTables.AnsiToNative 10244
DBTables.AnsiToNative function 10244
DBTables.AnsiToNativeBuf 10245
DBTables.AnsiToNativeBuf function 10245
DBTables.BlobTypeMap 10267
DBTables.BlobTypeMap constant 10267
DBTables.Check 10245
DBTables.Check function 10245
DBTables.cmAll 10268
DBTables.cmAll constant 10268
DBTables.cmPersistent 10268
DBTables.cmPersistent constant 10268
DBTables.cmSession 10268
DBTables.cmSession constant 10268
DBTables.cmVirtual 10269
DBTables.cmVirtual constant 10269
DBTables.DataTypeMap 10269
DBTables.DataTypeMap constant 10269
DBTables.dbfDatabase 10269
DBTables.dbfDatabase constant 10269
DBTables.dbfExecProc 10270
DBTables.dbfExecProc constant 10270
DBTables.dbfExecSQL 10270
DBTables.dbfExecSQL constant 10270
DBTables.dbfFieldList 10270
DBTables.dbfFieldList constant 10270
DBTables.dbfIndexList 10271
DBTables.dbfIndexList constant 10271
DBTables.dbfOpened 10271
DBTables.dbfOpened constant 10271
DBTables.dbfPrepared 10271
DBTables.dbfPrepared constant 10271
DBTables.dbfProcDesc 10272
DBTables.dbfProcDesc constant 10272
DBTables.dbfProvider 10272
DBTables.dbfProvider constant 10272
DBTables.dbfStoredProc 10272
DBTables.dbfStoredProc constant 10272
DBTables.dbfTable 10273
DBTables.dbfTable constant 10273
DBTables.DbiError 10246
DBTables.DbiError function 10246
DBTables.EDBEngineError 9854
DBTables.EDBEngineError.Create 9856
DBTables.EDBEngineError.Destroy 9856

DBTables.EDBEngineError.ErrorCount 9857
DBTables.EDBEngineError.Errors 9857
DBTables.ENoResultSet 9858
DBTables.FldSubTypeMap 10273
DBTables.FldSubTypeMap constant 10273
DBTables.FldTypeMap 10273
DBTables.FldTypeMap constant 10273
DBTables.GetFieldSource 10246
DBTables.GetFieldSource function 10246
DBTables.GetObjectContextProc 10265
DBTables.GetObjectContextProc variable 10265
DBTables.NativeCompareStr 10246
DBTables.NativeCompareStr function 10246
DBTables.NativeCompareStrBuf 10247
DBTables.NativeCompareStrBuf function 10247
DBTables.NativeCompareText 10248
DBTables.NativeCompareText function 10248
DBTables.NativeCompareTextBuf 10248
DBTables.NativeCompareTextBuf function 10248
DBTables.NativeToAnsi 10249
DBTables.NativeToAnsi function 10249
DBTables.NativeToAnsiBuf 10249
DBTables.NativeToAnsiBuf function 10249
DBTables.PKeyBuffer 10258
DBTables.PKeyBuffer type 10258
DBTables.PRecInfo 10258
DBTables.PRecInfo type 10258
DBTables.RegisterBDEInitProc 10250
DBTables.RegisterBDEInitProc function 10250
DBTables.Session 10265
DBTables.Session variable 10265
DBTables.Sessions 10266
DBTables.Sessions variable 10266
DBTables.smTraceBufSize 10274
DBTables.smTraceBufSize constant 10274
DBTables.TBatchMode 10251
DBTables.TBatchMode enumeration 10251
DBTables.TBatchMove 9858
DBTables.TBatchMove.AbortOnKeyViol 9865
DBTables.TBatchMove.AbortOnProblem 9866
DBTables.TBatchMove.ChangedCount 9866
DBTables.TBatchMove.ChangedTableName 9867
DBTables.TBatchMove.CommitCount 9867
DBTables.TBatchMove.Create 9862
DBTables.TBatchMove.Destination 9868
DBTables.TBatchMove.Destroy 9863
DBTables.TBatchMove.Execute 9863
DBTables.TBatchMove.KeyViolCount 9869
DBTables.TBatchMove.KeyViolTableName 9869
DBTables.TBatchMove.Mappings 9870
DBTables.TBatchMove.Mode 9871
DBTables.TBatchMove.MovedCount 9872
DBTables.TBatchMove.Notification 9864
DBTables.TBatchMove.ProblemCount 9872
DBTables.TBatchMove.ProblemTableName 9873
DBTables.TBatchMove.RecordCount 9873
DBTables.TBatchMove.Source 9874
DBTables.TBatchMove.Transliterate 9875
DBTables.TBDECallback 9875
DBTables.TBDECallback.Create 9876
DBTables.TBDECallback.Destroy 9877
DBTables.TBDECallback.Invoke 9878
DBTables.TBDECallbackEvent 10259
DBTables.TBDECallbackEvent type 10259
DBTables.TBDEDataSet 9878
DBTables.TBDEDataSet.ActivateFilters 9893
DBTables.TBDEDataSet.Active 9929
DBTables.TBDEDataSet.AddFieldDesc 9893
DBTables.TBDEDataSet.AfterCancel 9938
DBTables.TBDEDataSet.AfterClose 9940
DBTables.TBDEDataSet.AfterDelete 9942
DBTables.TBDEDataSet.AfterEdit 9944
DBTables.TBDEDataSet.AfterInsert 9945
DBTables.TBDEDataSet.AfterOpen 9946
DBTables.TBDEDataSet.AfterPost 9947
DBTables.TBDEDataSet.AfterRefresh 9947
DBTables.TBDEDataSet.AfterScroll 9948
DBTables.TBDEDataSet.AllocCachedUpdateBuffers 9893

DBTables.TBDEDataSet.AllocKeyBuffers 9893
DBTables.TBDEDataSet.AllocRecordBuffer 9894
DBTables.TBDEDataSet.ApplyUpdates 9894
DBTables.TBDEDataSet.AutoCalcFields 9931
DBTables.TBDEDataSet.BeforeCancel 9948
DBTables.TBDEDataSet.BeforeClose 9949
DBTables.TBDEDataSet.BeforeDelete 9949
DBTables.TBDEDataSet.BeforeEdit 9950
DBTables.TBDEDataSet.BeforeInsert 9950
DBTables.TBDEDataSet.BeforeOpen 9952
DBTables.TBDEDataSet.BeforePost 9952
DBTables.TBDEDataSet.BeforeRefresh 9953
DBTables.TBDEDataSet.BeforeScroll 9953
DBTables.TBDEDataSet.BlockReadNext 9894
DBTables.TBDEDataSet.BookmarkValid 9895
DBTables.TBDEDataSet.CacheBlobs 9932
DBTables.TBDEDataSet.CachedUpdateCallBack 9895
DBTables.TBDEDataSet.CachedUpdates 9932
DBTables.TBDEDataSet.Cancel 9895
DBTables.TBDEDataSet.CancelUpdates 9896
DBTables.TBDEDataSet.CheckCachedUpdateMode 9896
DBTables.TBDEDataSet.CheckSetKeyMode 9896
DBTables.TBDEDataSet.ClearCalcFields 9897
DBTables.TBDEDataSet.CloseBlob 9897
DBTables.TBDEDataSet.CloseCursor 9897
DBTables.TBDEDataSet.CommitUpdates 9897
DBTables.TBDEDataSet.CompareBookmarks 9898
DBTables.TBDEDataSet.ConstraintCallBack 9898
DBTables.TBDEDataSet.ConstraintsDisabled 9899
DBTables.TBDEDataSet.Create 9889
DBTables.TBDEDataSet.CreateBlobStream 9899
DBTables.TBDEDataSet.CreateExprFilter 9900
DBTables.TBDEDataSet.CreateFuncFilter 9900
DBTables.TBDEDataSet.CreateHandle 9900
DBTables.TBDEDataSet.CreateLookupFilter 9901
DBTables.TBDEDataSet.DataEvent 9901
DBTables.TBDEDataSet.DeactivateFilters 9901
DBTables.TBDEDataSet.Destroy 9890
DBTables.TBDEDataSet.DestroyHandle 9901
DBTables.TBDEDataSet.DestroyLookupCursor 9902
DBTables.TBDEDataSet.DisableConstraints 9902
DBTables.TBDEDataSet.EnableConstraints 9902
DBTables.TBDEDataSet.ExpIndex 9933
DBTables.TBDEDataSet.FetchAll 9903
DBTables.TBDEDataSet.FindRecord 9903
DBTables.TBDEDataSet.FlushBuffers 9903
DBTables.TBDEDataSet.ForceUpdateCallback 9904
DBTables.TBDEDataSet.FreeKeyBuffers 9904
DBTables.TBDEDataSet.FreeRecordBuffer 9904
DBTables.TBDEDataSet.GetBlobFieldData 9904
DBTables.TBDEDataSet.GetBookmarkData 9905
DBTables.TBDEDataSet.GetBookmarkFlag 9905
DBTables.TBDEDataSet.GetCanModify 9905
DBTables.TBDEDataSet.GetCurrentRecord 9905
DBTables.TBDEDataSet.GetFieldData 9906
DBTables.TBDEDataSet.GetFieldFullName 9907
DBTables.TBDEDataSet.GetHandle 9907
DBTables.TBDEDataSet.GetIndexField 9907
DBTables.TBDEDataSet.GetIndexFieldCount 9907
DBTables.TBDEDataSet.GetIndexInfo 9907
DBTables.TBDEDataSet.GetIsIndexField 9908
DBTables.TBDEDataSet.GetKeyBuffer 9908
DBTables.TBDEDataSet.GetKeyExclusive 9908
DBTables.TBDEDataSet.GetKeyFieldCount 9909
DBTables.TBDEDataSet.GetLookupCursor 9909
DBTables.TBDEDataSet.GetObjectTypeNames 9909
DBTables.TBDEDataSet.GetRecNo 9909
DBTables.TBDEDataSet.GetRecord 9910
DBTables.TBDEDataSet.GetRecordCount 9910
DBTables.TBDEDataSet.GetRecordSize 9910
DBTables.TBDEDataSet.GetStateFieldValue 9910
DBTables.TBDEDataSet.GetUpdateRecordSet 9910
DBTables.TBDEDataSet.GetUpdatesPending 9911
DBTables.TBDEDataSet.Handle 9933
DBTables.TBDEDataSet.InitKeyBuffer 9911
DBTables.TBDEDataSet.InitRecord 9911
DBTables.TBDEDataSet.InternalAddRecord 9911
DBTables.TBDEDataSet.InternalCancel 9912

DBTables.TBDEDataSet.InternalClose 9912
DBTables.TBDEDataSet.InternalDelete 9912
DBTables.TBDEDataSet.InternalEdit 9912
DBTables.TBDEDataSet.InternalFirst 9912
DBTables.TBDEDataSet.InternalGotoBookmark 9913
DBTables.TBDEDataSet.InternalHandleException 9913
DBTables.TBDEDataSet.InternalInitFieldDefs 9913
DBTables.TBDEDataSet.InternalInitRecord 9913
DBTables.TBDEDataSet.InternalInsert 9914
DBTables.TBDEDataSet.InternalLast 9914
DBTables.TBDEDataSet.InternalOpen 9914
DBTables.TBDEDataSet.InternalPost 9914
DBTables.TBDEDataSet.InternalRefresh 9914
DBTables.TBDEDataSet.InternalSetToRecord 9915
DBTables.TBDEDataSet.IsCursorOpen 9915
DBTables.TBDEDataSet.IsSequenced 9915
DBTables.TBDEDataSet.KeySize 9934
DBTables.TBDEDataSet.Locale 9934
DBTables.TBDEDataSet.Locate 9915
DBTables.TBDEDataSet.LocateRecord 9916
DBTables.TBDEDataSet.Lookup 9916
DBTables.TBDEDataSet.MapsToIndex 9918
DBTables.TBDEDataSet.ObjectView 9935
DBTables.TBDEDataSet.OnCalcFields 9954
DBTables.TBDEDataSet.OnDeleteError 9955
DBTables.TBDEDataSet.OnEditError 9955
DBTables.TBDEDataSet.OnNewRecord 9955
DBTables.TBDEDataSet.OnPostError 9956
DBTables.TBDEDataSet.OnUpdateError 9956
DBTables.TBDEDataSet.OnUpdateRecord 9957
DBTables.TBDEDataSet.OpenCursor 9918
DBTables.TBDEDataSet.Post 9918
DBTables.TBDEDataSet.PostKeyBuffer 9919
DBTables.TBDEDataSet.PrepareCursor 9919
DBTables.TBDEDataSet.ProcessUpdates 9919
DBTables.TBDEDataSet.PSGetUpdateException 9920
DBTables.TBDEDataSet.PSIsSQLSupported 9920
DBTables.TBDEDataSet.PSReset 9921
DBTables.TBDEDataSet.PSUpdateRecord 9921
DBTables.TBDEDataSet.ResetCursorRange 9922
DBTables.TBDEDataSet.RevertRecord 9922
DBTables.TBDEDataSet.SetBlockReadSize 9922
DBTables.TBDEDataSet.SetBookmarkData 9922
DBTables.TBDEDataSet.SetBookmarkFlag 9923
DBTables.TBDEDataSet.SetCachedUpdates 9923
DBTables.TBDEDataSet.SetCursorRange 9923
DBTables.TBDEDataSet.SetFieldData 9923
DBTables.TBDEDataSet.SetFilterData 9923
DBTables.TBDEDataSet.SetFiltered 9924
DBTables.TBDEDataSet.SetFilterHandle 9924
DBTables.TBDEDataSet.SetFilterOptions 9924
DBTables.TBDEDataSet.SetFilterText 9924
DBTables.TBDEDataSet.SetIndexField 9925
DBTables.TBDEDataSet.SetKeyBuffer 9925
DBTables.TBDEDataSet.SetKeyExclusive 9925
DBTables.TBDEDataSet.SetKeyFieldCount 9925
DBTables.TBDEDataSet.SetKeyFields 9925
DBTables.TBDEDataSet.SetLinkRanges 9926
DBTables.TBDEDataSet.SetLocale 9926
DBTables.TBDEDataSet.SetOnFilterRecord 9926
DBTables.TBDEDataSet.SetOnUpdateError 9926
DBTables.TBDEDataSet.SetRecNo 9927
DBTables.TBDEDataSet.SetStateFieldValue 9927
DBTables.TBDEDataSet.SetupCallBack 9927
DBTables.TBDEDataSet.SetUpdateObject 9927
DBTables.TBDEDataSet.SetUpdateRecordSet 9927
DBTables.TBDEDataSet.StmtHandle 9935
DBTables.TBDEDataSet.SwitchToIndex 9928
DBTables.TBDEDataSet.Translate 9928
DBTables.TBDEDataSet.UpdateCallbackRequired 9928
DBTables.TBDEDataSet.UpdateObject 9936
DBTables.TBDEDataSet.UpdateRecordTypes 9936
DBTables.TBDEDataSet.UpdatesPending 9937
DBTables.TBDEDataSet.UpdateStatus 9928
DBTables.TBDEInitProc 10259
DBTables.TBDEInitProc type 10259
DBTables.TBDEKeyBuffer 10252
DBTables.TBDEKeyBuffer record 10252

DBTables.TBDERecInfo 10252
DBTables.TBDERecInfo record 10252
DBTables.TBlobDataArray 10259
DBTables.TBlobDataArray type 10259
DBTables.TBlobStream 9958
DBTables.TBlobStream.Create 9960
DBTables.TBlobStream.Destroy 9961
DBTables.TBlobStream.Read 9962
DBTables.TBlobStream.Seek 9962
DBTables.TBlobStream.Truncate 9963
DBTables.TBlobStream.Write 9963
DBTables.TConfigMode 10260
DBTables.TConfigMode type 10260
DBTables.TConfigModes 10253
DBTables.TConfigModes enumeration 10253
DBTables.TDatabase 9964
DBTables.TDatabase.AfterConnect 9992
DBTables.TDatabase.AfterDisconnect 9993
DBTables.TDatabase.AliasName 9979
DBTables.TDatabase.ApplyUpdates 9970
DBTables.TDatabase.BeforeConnect 9993
DBTables.TDatabase.BeforeDisconnect 9993
DBTables.TDatabase.CloseDataSets 9971
DBTables.TDatabase.Commit 9971
DBTables.TDatabase.Connected 9979
DBTables.TDatabase.Create 9969
DBTables.TDatabase.DatabaseName 9980
DBTables.TDatabase.DataSets 9981
DBTables.TDatabase.Destroy 9969
DBTables.TDatabase.Directory 9981
DBTables.TDatabase.DoConnect 9972
DBTables.TDatabase.DoDisconnect 9972
DBTables.TDatabase.DriverName 9981
DBTables.TDatabase.Exclusive 9982
DBTables.TDatabase.Execute 9973
DBTables.TDatabase.FlushSchemaCache 9973
DBTables.TDatabase.GetConnected 9974
DBTables.TDatabase.GetDataSet 9974
DBTables.TDatabase.GetFieldNames 9975
DBTables.TDatabase.GetTableNames 9975
DBTables.TDatabase.Handle 9983
DBTables.TDatabase.HandleShared 9983
DBTables.TDatabase.InTransaction 9983
DBTables.TDatabase.IsSQLBased 9984
DBTables.TDatabase.KeepConnection 9984
DBTables.TDatabase.Loaded 9975
DBTables.TDatabase.Locale 9985
DBTables.TDatabase.LoginPrompt 9985
DBTables.TDatabase.Notification 9976
DBTables.TDatabase.OnLogin 9994
DBTables.TDatabase.Params 9986
DBTables.TDatabase.ReadOnly 9987
DBTables.TDatabase.Rollback 9976
DBTables.TDatabase.Session 9988
DBTables.TDatabase.SessionAlias 9988
DBTables.TDatabase.SessionName 9989
DBTables.TDatabase.StartTransaction 9977
DBTables.TDatabase.Temporary 9989
DBTables.TDatabase.TraceFlags 9990
DBTables.TDatabase.TransIsolation 9991
DBTables.TDatabase.ValidateName 9977
DBTables.TDatabaseEvent 10253
DBTables.TDatabaseEvent enumeration 10253
DBTables.TDatabaseLoginEvent 10260
DBTables.TDatabaseLoginEvent type 10260
DBTables.TDatabaseNotifyEvent 10260
DBTables.TDatabaseNotifyEvent type 10260
DBTables.TDataSetUpdateObject 9994
DBTables.TDataSetUpdateObject.Apply 9998
DBTables.TDataSetUpdateObject.DataSet 9999
DBTables.TDataSetUpdateObject.GetDataSet 9998
DBTables.TDataSetUpdateObject.SetDataSet 9999
DBTables.TDBDataSet 10000
DBTables.TDBDataSet.AutoRefresh 10021
DBTables.TDBDataSet.CheckOpen 10014
DBTables.TDBDataSet.CloseCursor 10014
DBTables.TDBDataSet.CloseDatabase 10014
DBTables.TDBDataSet.ConstraintsStored 10015

DBTables.TDBDataSet.Create 10013
DBTables.TDBDataSet.Database 10021
DBTables.TDBDataSet.DatabaseName 10022
DBTables.TDBDataSet.DBFlags 10023
DBTables.TDBDataSet.DBHandle 10023
DBTables.TDBDataSet.DBLocale 10024
DBTables.TDBDataSet.DBSession 10024
DBTables.TDBDataSet.Disconnect 10015
DBTables.TDBDataSet.Filter 10025
DBTables.TDBDataSet.Filtered 10026
DBTables.TDBDataSet.FilterOptions 10026
DBTables.TDBDataSet.Handle 10027
DBTables.TDBDataSet.OnFilterRecord 10028
DBTables.TDBDataSet.OpenCursor 10015
DBTables.TDBDataSet.OpenDatabase 10016
DBTables.TDBDataSet.PSEndTransaction 10016
DBTables.TDBDataSet.PSExecuteStatement 10017
DBTables.TDBDataSet.PSGetAttributes 10017
DBTables.TDBDataSet.PSGetQuoteChar 10018
DBTables.TDBDataSet.PSInTransaction 10018
DBTables.TDBDataSet.PSIsSQLBased 10018
DBTables.TDBDataSet.PSStartTransaction 10019
DBTables.TDBDataSet.PSUpdateRecord 10019
DBTables.TDBDataSet.SessionName 10027
DBTables.TDBDataSet.SetDBFlag 10020
DBTables.TDBDataSet.SetHandle 10020
DBTables.TDBDataSet.UpdateMode 10028
DBTables.TDBError 10029
DBTables.TDBError.Category 10031
DBTables.TDBError.Create 10030
DBTables.TDBError.ErrorCode 10032
DBTables.TDBError.Message 10032
DBTables.TDBError.NativeError 10032
DBTables.TDBError.SubCode 10033
DBTables.TDBFlags 10261
DBTables.TDBFlags type 10261
DBTables.TFieldDescList 10261
DBTables.TFieldDescList type 10261
DBTables.TIndexDescList 10261
DBTables.TIndexDescList type 10261
DBTables.TIndexFiles 10033
DBTables.TIndexFiles.Add 10037
DBTables.TIndexFiles.Clear 10039
DBTables.TIndexFiles.Create 10036
DBTables.TIndexFiles.Delete 10039
DBTables.TIndexFiles.Insert 10040
DBTables.TIndexName 10262
DBTables.TIndexName type 10262
DBTables.TLocale 10262
DBTables.TLocale type 10262
DBTables.TLockType 10254
DBTables.TLockType enumeration 10254
DBTables.TNestedTable 10040
DBTables.TNestedTable.Create 10053
DBTables.TNestedTable.CreateHandle 10053
DBTables.TNestedTable.DataSetField 10054
DBTables.TNestedTable.DoAfterPost 10054
DBTables.TNestedTable.DoBeforeInsert 10054
DBTables.TNestedTable.InternalPost 10054
DBTables.TNestedTable.ObjectView 10055
DBTables.TParamBindMode 10254
DBTables.TParamBindMode enumeration 10254
DBTables.TParamList 10056
DBTables.TParamList.Buffer 10058
DBTables.TParamList.Create 10057
DBTables.TParamList.Destroy 10057
DBTables.TParamList.FieldCount 10058
DBTables.TParamList.FieldDescs 10059
DBTables.TPasswordEvent 10262
DBTables.TPasswordEvent type 10262
DBTables.TQuery 10059
DBTables.TQuery.Constrained 10083
DBTables.TQuery.Constraints 10083
DBTables.TQuery.Create 10073
DBTables.TQuery.CreateHandle 10074
DBTables.TQuery.DataLink 10084
DBTables.TQuery.DataSource 10084
DBTables.TQuery.DefineProperties 10074

DBTables.TQuery.Destroy 10073
DBTables.TQuery.Disconnect 10075
DBTables.TQuery.ExecSQL 10075
DBTables.TQuery.FreeStatement 10076
DBTables.TQuery.GetDataSource 10076
DBTables.TQuery.GetDetailLinkFields 10076
DBTables.TQuery.GetParamsCount 10076
DBTables.TQuery.GetStatementHandle 10077
DBTables.TQuery.Local 10085
DBTables.TQuery.ParamByName 10077
DBTables.TQuery.ParamCheck 10086
DBTables.TQuery.ParamCount 10086
DBTables.TQuery.Params 10087
DBTables.TQuery.Prepare 10077
DBTables.TQuery.Prepared 10087
DBTables.TQuery.PSExecute 10078
DBTables.TQuery.PSGetCommandText 10078
DBTables.TQuery.PSGetCommandType 10079
DBTables.TQuery.PSGetDefaultOrder 10079
DBTables.TQuery.PSGetParams 10079
DBTables.TQuery.PSGetTableName 10080
DBTables.TQuery.PSSetCommandText 10080
DBTables.TQuery.PSSetParams 10081
DBTables.TQuery.RequestLive 10088
DBTables.TQuery.RowsAffected 10089
DBTables.TQuery.SetDBFlag 10081
DBTables.TQuery.SQL 10089
DBTables.TQuery.SQLBinary 10090
DBTables.TQuery.StmtHandle 10091
DBTables.TQuery.Text 10091
DBTables.TQuery.UniDirectional 10092
DBTables.TQuery.UnPrepare 10081
DBTables.TQuery.UpdateMode 10092
DBTables.TQuery.UpdateObject 10092
DBTables.TRecNoStatus 10255
DBTables.TRecNoStatus enumeration 10255
DBTables.TServerDesc 10255
DBTables.TServerDesc record 10255
DBTables.TServerDescList 10263
DBTables.TServerDescList type 10263
DBTables.TSession 10093
DBTables.TSession.Active 10120
DBTables.TSession.AddAlias 10100
DBTables.TSession.AddDriver 10101
DBTables.TSession.AddPassword 10102
DBTables.TSession.AddStandardAlias 10102
DBTables.TSession.AutoSessionName 10121
DBTables.TSession.BDEOwnsLoginCbDb 10122
DBTables.TSession.Close 10103
DBTables.TSession.CloseDatabase 10103
DBTables.TSession.ConfigMode 10122
DBTables.TSession.Create 10098
DBTables.TSession.DatabaseCount 10123
DBTables.TSession.Databases 10123
DBTables.TSession.DeleteAlias 10104
DBTables.TSession.DeleteDriver 10105
DBTables.TSession.Destroy 10099
DBTables.TSession.DropConnections 10105
DBTables.TSession.FindDatabase 10105
DBTables.TSession.GetAliasDriverName 10106
DBTables.TSession.GetAliasNames 10107
DBTables.TSession.GetAliasParams 10108
DBTables.TSession.GetConfigParams 10108
DBTables.TSession.GetDatabaseNames 10109
DBTables.TSession.GetDriverNames 10110
DBTables.TSession.GetDriverParams 10110
DBTables.TSession.GetFieldNames 10110
DBTables.TSession.GetPassword 10111
DBTables.TSession.GetStoredProcNames 10112
DBTables.TSession.GetTableNames 10113
DBTables.TSession.Handle 10123
DBTables.TSession.IsAlias 10113
DBTables.TSession.KeepConnections 10124
DBTables.TSession.Loaded 10114
DBTables.TSession.Locale 10124
DBTables.TSession.ModifyAlias 10114
DBTables.TSession.ModifyConfigParams 10115
DBTables.TSession.ModifyDriver 10115

DBTables.TSession.NetFileDir 10125
DBTables.TSession.Notification 10116
DBTables.TSession.OnDBNotify 10128
DBTables.TSession.OnPassword 10128
DBTables.TSession.OnStartup 10129
DBTables.TSession.Open 10116
DBTables.TSession.OpenDatabase 10117
DBTables.TSession.PrivateDir 10125
DBTables.TSession.RemoveAllPasswords 10118
DBTables.TSession.RemovePassword 10118
DBTables.TSession.SaveConfigFile 10119
DBTables.TSession.SessionName 10126
DBTables.TSession.SetName 10119
DBTables.TSession.SQLHourGlass 10126
DBTables.TSession.TraceFlags 10127
DBTables.TSessionList 10129
DBTables.TSessionList.Count 10133
DBTables.TSessionList.Create 10131
DBTables.TSessionList.CurrentSession 10134
DBTables.TSessionList.Destroy 10131
DBTables.TSessionList.FindSession 10132
DBTables.TSessionList.GetSessionNames 10132
DBTables.TSessionList.List 10134
DBTables.TSessionList.OpenSession 10132
DBTables.TSessionList.Sessions 10134
DBTables.TSPParamDescList 10263
DBTables.TSPParamDescList type 10263
DBTables.TSQLUpdateObject 10135
DBTables.TSQLUpdateObject.GetSQL 10138
DBTables.TStoredProc 10139
DBTables.TStoredProc.CopyParams 10154
DBTables.TStoredProc.Create 10152
DBTables.TStoredProc.CreateHandle 10154
DBTables.TStoredProc.DefineProperties 10155
DBTables.TStoredProc.DescriptionsAvailable 10155
DBTables.TStoredProc.Destroy 10153
DBTables.TStoredProc.Disconnect 10156
DBTables.TStoredProc.ExecProc 10156
DBTables.TStoredProc.GetParamsCount 10156
DBTables.TStoredProc.GetResults 10157
DBTables.TStoredProc.Handle 10163
DBTables.TStoredProc.Overload 10163
DBTables.TStoredProc.ParamBindMode 10164
DBTables.TStoredProc.ParamByName 10157
DBTables.TStoredProc.ParamCount 10165
DBTables.TStoredProc.Params 10165
DBTables.TStoredProc.Prepare 10157
DBTables.TStoredProc.Prepared 10165
DBTables.TStoredProc.PSExecute 10158
DBTables.TStoredProc.PSGetCommandText 10158
DBTables.TStoredProc.PSGetCommandType 10159
DBTables.TStoredProc.PSGetParams 10159
DBTables.TStoredProc.PSGetTableName 10160
DBTables.TStoredProc.PSSetCommandText 10160
DBTables.TStoredProc.PSSetParams 10161
DBTables.TStoredProc.SetDBFlag 10161
DBTables.TStoredProc.SetOverLoad 10161
DBTables.TStoredProc.SetPrepare 10162
DBTables.TStoredProc.SetPrepared 10162
DBTables.TStoredProc.SetProcName 10162
DBTables.TStoredProc.StmtHandle 10166
DBTables.TStoredProc.StoredProcName 10167
DBTables.TStoredProc.UnPrepare 10162
DBTables.TStoredProc.UpdateObject 10167
DBTables.TTable 10168
DBTables.TTable.AddIndex 10184
DBTables.TTable.ApplyRange 10185
DBTables.TTable.BatchMove 10186
DBTables.TTable.CancelRange 10186
DBTables.TTable.CloseIndexFile 10188
DBTables.TTable.Constraints 10214
DBTables.TTable.Create 10182
DBTables.TTable.CreateHandle 10189
DBTables.TTable.CreateTable 10189
DBTables.TTable.DataEvent 10191
DBTables.TTable.DefaultIndex 10214
DBTables.TTable.DefChanged 10191
DBTables.TTable.DeleteIndex 10191

DBTables.TTable.DeleteTable 10192
DBTables.TTable.Destroy 10182
DBTables.TTable.DestroyHandle 10193
DBTables.TTable.DestroyLookupCursor 10193
DBTables.TTable.DoOnNewRecord 10194
DBTables.TTable.EditKey 10194
DBTables.TTable.EditRangeEnd 10194
DBTables.TTable.EditRangeStart 10195
DBTables.TTable.EmptyTable 10196
DBTables.TTable.EncodeFieldDesc 10196
DBTables.TTable.EncodeIndexDesc 10196
DBTables.TTable.Exclusive 10215
DBTables.TTable.Exists 10215
DBTables.TTable.FieldDefs 10217
DBTables.TTable.FindKey 10196
DBTables.TTable.FindNearest 10197
DBTables.TTable.GetCanModify 10198
DBTables.TTable.GetDataSource 10199
DBTables.TTable.GetDetailLinkFields 10199
DBTables.TTable.GetFileName 10199
DBTables.TTable.GetHandle 10199
DBTables.TTable.GetIndexNames 10200
DBTables.TTable.GetLanguageDriverName 10200
DBTables.TTable.GetLookupCursor 10200
DBTables.TTable.GetTableType 10200
DBTables.TTable.GotoCurrent 10201
DBTables.TTable.GotoKey 10201
DBTables.TTable.GotoNearest 10202
DBTables.TTable.IndexDefs 10219
DBTables.TTable.IndexFieldCount 10220
DBTables.TTable.IndexFieldNames 10220
DBTables.TTable.IndexFields 10221
DBTables.TTable.IndexFiles 10221
DBTables.TTable.IndexName 10222
DBTables.TTable.InitFieldDefs 10203
DBTables.TTable.IsProductionIndex 10203
DBTables.TTable.KeyExclusive 10223
DBTables.TTable.KeyFieldCount 10223
DBTables.TTable.LockTable 10203
DBTables.TTable.MasterFields 10224
DBTables.TTable.MasterLink 10224
DBTables.TTable.MasterSource 10224
DBTables.TTable.NativeTableName 10203
DBTables.TTable.OpenIndexFile 10204
DBTables.TTable.PrepareCursor 10204
DBTables.TTable.PSGetCommandText 10204
DBTables.TTable.PSGetCommandType 10205
DBTables.TTable.PSGetDefaultOrder 10205
DBTables.TTable.PSGetIndexDefs 10205
DBTables.TTable.PSGetKeyFields 10206
DBTables.TTable.PSGetTableName 10206
DBTables.TTable.PSSetCommandText 10207
DBTables.TTable.PSSetParams 10207
DBTables.TTable.Ranged 10225
DBTables.TTable.ReadOnly 10225
DBTables.TTable.RenameTable 10208
DBTables.TTable.SetKey 10208
DBTables.TTable.SetRange 10209
DBTables.TTable.SetRangeEnd 10211
DBTables.TTable.SetRangeStart 10212
DBTables.TTable.StoreDefs 10226
DBTables.TTable.TableLevel 10226
DBTables.TTable.TableName 10226
DBTables.TTable.TableType 10227
DBTables.TTable.UnlockTable 10212
DBTables.TTable.UpdateIndexDefs 10213
DBTables.TTable.UpdateMode 10228
DBTables.TTable.UpdateObject 10228
DBTables.TTableType 10256
DBTables.TTableType enumeration 10256
DBTables.TTraceFlag 10256
DBTables.TTraceFlag enumeration 10256
DBTables.TTraceFlags 10263
DBTables.TTraceFlags type 10263
DBTables.TTransIsolation 10257
DBTables.TTransIsolation enumeration 10257
DBTables.TUpdateSQL 10229
DBTables.TUpdateSQL.Apply 10234

DBTables.TUpdateSQL.Create 10233
DBTables.TUpdateSQL.DatabaseName 10238
DBTables.TUpdateSQL.DataSet 10239
DBTables.TUpdateSQL.DeleteSQL 10239
DBTables.TUpdateSQL.Destroy 10233
DBTables.TUpdateSQL.ExecSQL 10235
DBTables.TUpdateSQL.GetDataSet 10235
DBTables.TUpdateSQL.GetSQL 10236
DBTables.TUpdateSQL.InsertSQL 10240
DBTables.TUpdateSQL.ModifySQL 10240
DBTables.TUpdateSQL.Query 10241
DBTables.TUpdateSQL.SessionName 10242
DBTables.TUpdateSQL.SetDataSet 10236
DBTables.TUpdateSQL.SetParams 10237
DBTables.TUpdateSQL.SQL 10242
DBTables.TUpdateSQL.SQLChanged 10237
DBTables.TValCheckList 10264
DBTables.TValCheckList type 10264
DBWeb 10275
DBWeb namespace 10275
    Classes 10277
    Functions 10346
    Structs, Records, Enums 10347
    Types 10349
DBWeb.HtmlTable 10346
DBWeb.HtmlTable function 10346
DBWeb.TCreateContentEvent 10349
DBWeb.TCreateContentEvent type 10349
DBWeb.TDataSetTableProducer 10277
DBWeb.TDataSetTableProducer.Caption 10284
DBWeb.TDataSetTableProducer.CaptionAlignment 10284
DBWeb.TDataSetTableProducer.Columns 10285
DBWeb.TDataSetTableProducer.Content 10282
DBWeb.TDataSetTableProducer.DataSet 10285
DBWeb.TDataSetTableProducer.Footer 10286
DBWeb.TDataSetTableProducer.GetDataSet 10283
DBWeb.TDataSetTableProducer.Header 10286
DBWeb.TDataSetTableProducer.MaxRows 10287
DBWeb.TDataSetTableProducer.Notification 10283
DBWeb.TDataSetTableProducer.OnCreateContent 10289
DBWeb.TDataSetTableProducer.OnFormatCell 10289
DBWeb.TDataSetTableProducer.OnGetTableCaption 10290
DBWeb.TDataSetTableProducer.RowAttributes 10287
DBWeb.TDataSetTableProducer.SetDataSet 10283
DBWeb.TDataSetTableProducer.TableAttributes 10288
DBWeb.TDSTableProducer 10290
DBWeb.TDSTableProducer.AcquireLayoutLock 10296
DBWeb.TDSTableProducer.BeginLayout 10297
DBWeb.TDSTableProducer.BeginUpdate 10297
DBWeb.TDSTableProducer.Caption 10307
DBWeb.TDSTableProducer.CaptionAlignment 10308
DBWeb.TDSTableProducer.ColumnHeader 10298
DBWeb.TDSTableProducer.Columns 10308
DBWeb.TDSTableProducer.Create 10295
DBWeb.TDSTableProducer.DataLink 10309
DBWeb.TDSTableProducer.DataSet 10309
DBWeb.TDSTableProducer.DataSource 10309
DBWeb.TDSTableProducer.DefineFieldMap 10298
DBWeb.TDSTableProducer.Destroy 10295
DBWeb.TDSTableProducer.DoCreateContent 10299
DBWeb.TDSTableProducer.DoFormatCell 10299
DBWeb.TDSTableProducer.DoGetCaption 10300
DBWeb.TDSTableProducer.Editor 10310
DBWeb.TDSTableProducer.EndLayout 10300
DBWeb.TDSTableProducer.EndUpdate 10301
DBWeb.TDSTableProducer.Footer 10310
DBWeb.TDSTableProducer.FormatCell 10301
DBWeb.TDSTableProducer.GetDataSet 10302
DBWeb.TDSTableProducer.GetDataSource 10302
DBWeb.TDSTableProducer.Header 10311
DBWeb.TDSTableProducer.InternalDataSource 10311
DBWeb.TDSTableProducer.LayoutChanged 10303
DBWeb.TDSTableProducer.LinkActive 10303
DBWeb.TDSTableProducer.MaxRows 10312
DBWeb.TDSTableProducer.Notification 10304
DBWeb.TDSTableProducer.OnCreateContent 10313
DBWeb.TDSTableProducer.OnFormatCell 10314
DBWeb.TDSTableProducer.OnGetTableCaption 10314

DBWeb.TDSTableProducer.RowAttributes 10312
DBWeb.TDSTableProducer.RowHeader 10304
DBWeb.TDSTableProducer.SetColumns 10304
DBWeb.TDSTableProducer.SetDataSet 10305
DBWeb.TDSTableProducer.SetDataSource 10305
DBWeb.TDSTableProducer.StoreColumns 10305
DBWeb.TDSTableProducer.TableAttributes 10313
DBWeb.TDSTableProducer.TableCaption 10306
DBWeb.TDSTableProducer.TableHeader 10306
DBWeb.TDSTableProducerEditor 10315
DBWeb.TDSTableProducerEditor.Changed 10317
DBWeb.TDSTableProducerEditor.Create 10316
DBWeb.TDSTableProducerEditor.DataSource 10318
DBWeb.TDSTableProducerEditor.Destroy 10316
DBWeb.TDSTableProducerEditor.DSTableProducer 10318
DBWeb.TDSTableProducerEditor.PostChange 10317
DBWeb.THTMLCaptionAlignment 10347
DBWeb.THTMLCaptionAlignment enumeration 10347
DBWeb.THTMLColumnState 10347
DBWeb.THTMLColumnState enumeration 10347
DBWeb.THTMLDataSetEmpty 10349
DBWeb.THTMLDataSetEmpty type 10349
DBWeb.THTMLFormatCellEvent 10350
DBWeb.THTMLFormatCellEvent type 10350
DBWeb.THTMLGetTableCaptionEvent 10350
DBWeb.THTMLGetTableCaptionEvent type 10350
DBWeb.THTMLTableColumn 10318
DBWeb.THTMLTableColumn.Align 10324
DBWeb.THTMLTableColumn.AssignTo 10322
DBWeb.THTMLTableColumn.BgColor 10325
DBWeb.THTMLTableColumn.Create 10320
DBWeb.THTMLTableColumn.Custom 10325
DBWeb.THTMLTableColumn.Destroy 10321
DBWeb.THTMLTableColumn.DSTableProducer 10325
DBWeb.THTMLTableColumn.Field 10326
DBWeb.THTMLTableColumn.FieldName 10326
DBWeb.THTMLTableColumn.GetDisplayName 10322
DBWeb.THTMLTableColumn.GeTDSTableProducer 10323
DBWeb.THTMLTableColumn.IsStored 10323
DBWeb.THTMLTableColumn.RestoreDefaults 10323
DBWeb.THTMLTableColumn.Title 10327
DBWeb.THTMLTableColumn.Update 10324
DBWeb.THTMLTableColumn.VAlign 10327
DBWeb.THTMLTableColumnClass 10350
DBWeb.THTMLTableColumnClass type 10350
DBWeb.THTMLTableColumns 10327
DBWeb.THTMLTableColumns.Add 10330
DBWeb.THTMLTableColumns.Create 10330
DBWeb.THTMLTableColumns.DSTableProducer 10334
DBWeb.THTMLTableColumns.GetAttr 10331
DBWeb.THTMLTableColumns.GetAttrCount 10331
DBWeb.THTMLTableColumns.GetItemAttr 10331
DBWeb.THTMLTableColumns.GetOwner 10332
DBWeb.THTMLTableColumns.Items 10334
DBWeb.THTMLTableColumns.RebuildColumns 10332
DBWeb.THTMLTableColumns.RestoreDefaults 10333
DBWeb.THTMLTableColumns.State 10335
DBWeb.THTMLTableColumns.Update 10333
DBWeb.THTTPDataLink 10335
DBWeb.THTTPDataLink.ActiveChanged 10339
DBWeb.THTTPDataLink.AddMapping 10339
DBWeb.THTTPDataLink.ClearMapping 10340
DBWeb.THTTPDataLink.Create 10338
DBWeb.THTTPDataLink.DataSetChanged 10340
DBWeb.THTTPDataLink.DataSetScrolled 10340
DBWeb.THTTPDataLink.DefaultFields 10344
DBWeb.THTTPDataLink.Destroy 10338
DBWeb.THTTPDataLink.EditingChanged 10341
DBWeb.THTTPDataLink.FieldCount 10344
DBWeb.THTTPDataLink.Fields 10345
DBWeb.THTTPDataLink.FocusControl 10341
DBWeb.THTTPDataLink.GetMappedIndex 10342
DBWeb.THTTPDataLink.LayoutChanged 10342
DBWeb.THTTPDataLink.Modified 10342
DBWeb.THTTPDataLink.RecordChanged 10343
DBWeb.THTTPDataLink.Reset 10343
DBWeb.THTTPDataLink.SparseMap 10345
DBWeb.THTTPDataLink.UpdateData 10343

DBXCommon 10351
DBXCommon namespace 10351
Classes 10355
Constants 10675
Structs, Records, Enums 10668
Types 10669
DBXCommon.DBXDefaultRowSetSize 10675
DBXCommon.DBXDefaultRowSetSize constant 10675
DBXCommon.DBXVersion25 10676
DBXCommon.DBXVersion25 constant 10676
DBXCommon.DBXVersion30 10676
DBXCommon.DBXVersion30 constant 10676
DBXCommon.DBXVersion40 10676
DBXCommon.DBXVersion40 constant 10676
DBXCommon.SAlreadyPrepared 10677
DBXCommon.SAlreadyPrepared constant 10677
DBXCommon.SConnectionFactoryInitFailed 10677
DBXCommon.SConnectionFactoryInitFailed constant 10677
DBXCommon.SDefaultErrorMessage 10677
DBXCommon.SDefaultErrorMessage constant 10677
DBXCommon.SDllLoadError 10678
DBXCommon.SDllLoadError constant 10678
DBXCommon.SDllProcLoadError 10678
DBXCommon.SDllProcLoadError constant 10678
DBXCommon.SDriverLoadError 10678
DBXCommon.SDriverLoadError constant 10678
DBXCommon.SInvalidArgument 10679
DBXCommon.SInvalidArgument constant 10679
DBXCommon.SInvalidDelegationDepth 10679
DBXCommon.SInvalidDelegationDepth constant 10679
DBXCommon.SInvalidOrdinal 10679
DBXCommon.SInvalidOrdinal constant 10679
DBXCommon.SInvalidTransaction 10680
DBXCommon.SInvalidTransaction constant 10680
DBXCommon.SInvalidTypeAccess 10680
DBXCommon.SInvalidTypeAccess constant 10680
DBXCommon.SNotImplemented 10680
DBXCommon.SNotImplemented constant 10680
DBXCommon.SReaderClosed 10681
DBXCommon.SReaderClosed constant 10681
DBXCommon.SReaderNew 10681
DBXCommon.SReaderNew constant 10681
DBXCommon.SReadOnlyParameter 10681
DBXCommon.SReadOnlyParameter constant 10681
DBXCommon.SReadOnlyType 10682
DBXCommon.SReadOnlyType constant 10682
DBXCommon.SRequiredProperty 10682
DBXCommon.SRequiredProperty constant 10682
DBXCommon.SSetFactoryOnce 10682
DBXCommon.SSetFactoryOnce constant 10682
DBXCommon.SUnknownDriver 10683
DBXCommon.SUnknownDriver constant 10683
DBXCommon.TDBXAnsiStringValue 10356
DBXCommon.TDBXAnsiStringValue.Create 10358
DBXCommon.TDBXAnsiStringValue.Destroy 10358
DBXCommon.TDBXAnsiStringValue.GetAnsiString 10359
DBXCommon.TDBXAnsiStringValue.IsNull 10359
DBXCommon.TDBXAnsiStringValue.SetAnsiString 10359
DBXCommon.TDBXBcdValue 10359
DBXCommon.TDBXBcdValue.GetBcd 10361
DBXCommon.TDBXBcdValue.IsNull 10361
DBXCommon.TDBXBcdValue.SetBcd 10362
DBXCommon.TDBXBooleanValue 10362
DBXCommon.TDBXBooleanValue.GetBoolean 10364
DBXCommon.TDBXBooleanValue.IsNull 10364
DBXCommon.TDBXBooleanValue.SetBoolean 10364
DBXCommon.TDBXByteArrayValue 10364
DBXCommon.TDBXByteArrayValue.Destroy 10366
DBXCommon.TDBXByteArrayValue.GetBytes 10366
DBXCommon.TDBXByteArrayValue.GetValueSize 10367
DBXCommon.TDBXByteArrayValue.IsNull 10367
DBXCommon.TDBXByteArrayValue.SetDynamicBytes 10367
DBXCommon.TDBXByteArrayValue.SetStaticBytes 10367
DBXCommon.TDBXByteReader 10368
DBXCommon.TDBXByteReader.Create 10369
DBXCommon.TDBXByteReader.FDBXContext 10369
DBXCommon.TDBXByteReader.GetAnsiString 10370
DBXCommon.TDBXByteReader.GetBcd 10370

DBXCommon.TDBXByteReader.GetByteLength 10370
DBXCommon.TDBXByteReader.GetBytes 10370
DBXCommon.TDBXByteReader.GetDate 10371
DBXCommon.TDBXByteReader.GetDouble 10371
DBXCommon.TDBXByteReader.GetInt16 10371
DBXCommon.TDBXByteReader.GetInt32 10371
DBXCommon.TDBXByteReader.GetInt64 10372
DBXCommon.TDBXByteReader.GetTime 10372
DBXCommon.TDBXByteReader.GetTimeStamp 10372
DBXCommon.TDBXByteReader.GetWideString 10372
DBXCommon.TDBXByteReaderEx 10373
DBXCommon.TDBXByteReaderEx.GetBoolean 10374
DBXCommon.TDBXByteStreamReader 10374
DBXCommon.TDBXByteStreamReader.Create 10375
DBXCommon.TDBXByteStreamReader.Destroy 10375
DBXCommon.TDBXByteStreamReader.Eos 10377
DBXCommon.TDBXByteStreamReader.FBytes 10376
DBXCommon.TDBXByteStreamReader.FCount 10376
DBXCommon.TDBXByteStreamReader.FEOS 10376
DBXCommon.TDBXByteStreamReader.FOffset 10376
DBXCommon.TDBXByteStreamReader.FPosition 10376
DBXCommon.TDBXByteStreamReader.Read 10377
DBXCommon.TDBXByteStreamReader.Size 10377
DBXCommon.TDBXChars 10669
DBXCommon.TDBXChars type 10669
DBXCommon.TDBXCommand 10378
DBXCommon.TDBXCommand.Close 10382
DBXCommon.TDBXCommand.CommandType 10388
DBXCommon.TDBXCommand.Create 10380
DBXCommon.TDBXCommand.CreateParameter 10382
DBXCommon.TDBXCommand.CreateParameterRow 10382
DBXCommon.TDBXCommand.CreateParameters 10382
DBXCommon.TDBXCommand.DerivedClose 10383
DBXCommon.TDBXCommand.DerivedExecuteQuery 10383
DBXCommon.TDBXCommand.DerivedExecuteUpdate 10383
DBXCommon.TDBXCommand.DerivedGetNextReader 10383
DBXCommon.TDBXCommand.DerivedOpen 10384
DBXCommon.TDBXCommand.DerivedPrepare 10384
DBXCommon.TDBXCommand.Destroy 10380
DBXCommon.TDBXCommand.ExecuteQuery 10384
DBXCommon.TDBXCommand.ExecuteUpdate 10384
DBXCommon.TDBXCommand.FDBXContext 10381
DBXCommon.TDBXCommand.FParameters 10381
DBXCommon.TDBXCommand.GetCommandType 10385
DBXCommon.TDBXCommand.GetErrorEvent 10385
DBXCommon.TDBXCommand.GetNextReader 10385
DBXCommon.TDBXCommand.GetParameters 10385
DBXCommon.TDBXCommand.GetRowsAffected 10386
DBXCommon.TDBXCommand.GetText 10386
DBXCommon.TDBXCommand.IsPrepared 10388
DBXCommon.TDBXCommand.MaxBlobSize 10389
DBXCommon.TDBXCommand.NotImplemented 10386
DBXCommon.TDBXCommand.OnErrorEvent 10391
DBXCommon.TDBXCommand.Open 10386
DBXCommon.TDBXCommand.Parameters 10389
DBXCommon.TDBXCommand.Prepare 10386
DBXCommon.TDBXCommand.RowsAffected 10389
DBXCommon.TDBXCommand.RowSetSize 10390
DBXCommon.TDBXCommand.SetCommandType 10387
DBXCommon.TDBXCommand.SetMaxBlobSize 10387
DBXCommon.TDBXCommand.SetRowSetSize 10387
DBXCommon.TDBXCommand.SetText 10387
DBXCommon.TDBXCommand.Text 10390
DBXCommon.TDBXCommandEx 10391
DBXCommon.TDBXCommandEx.CommandTimeout 10394
DBXCommon.TDBXCommandEx.GetCommandTimeout 10393
DBXCommon.TDBXCommandEx.LastReader 10394
DBXCommon.TDBXCommandEx.SetCommandTimeout 10393
DBXCommon.TDBXCommandEx.SetLastReader 10394
DBXCommon.TDBXCommandFactory 10395
DBXCommon.TDBXCommandFactory.CreateCommand 10395
DBXCommon.TDBXCommandTypes 10396
DBXCommon.TDBXCommandTypes.DbxCommand 10397
DBXCommon.TDBXCommandTypes.DbxMetaData 10397
DBXCommon.TDBXCommandTypes.DbxPool 10397
DBXCommon.TDBXCommandTypes.DbxSQL 10398
DBXCommon.TDBXCommandTypes.DbxStoredProcedure 10398
DBXCommon.TDBXCommandTypes.DbxTable 10398

DBXCommon.TDBXConnection 10398
DBXCommon.TDBXConnection.BeginTransaction 10402
DBXCommon.TDBXConnection.Close 10403
DBXCommon.TDBXConnection.Commit 10403
DBXCommon.TDBXConnection.CommitFreeAndNil 10404
DBXCommon.TDBXConnection.ConnectionProperties 10410
DBXCommon.TDBXConnection.Create 10400
DBXCommon.TDBXConnection.CreateAndBeginTransaction 10404
DBXCommon.TDBXConnection.CreateCommand 10404
DBXCommon.TDBXConnection.DatabaseMetaData 10410
DBXCommon.TDBXConnection.DerivedCreateCommand 10405
DBXCommon.TDBXConnection.DerivedGetCommands 10405
DBXCommon.TDBXConnection.DerivedGetCommandTypes 10405
DBXCommon.TDBXConnection.DerivedOpen 10405
DBXCommon.TDBXConnection.Destroy 10400
DBXCommon.TDBXConnection.FConnectionProperties 10401
DBXCommon.TDBXConnection.FDBXContext 10401
DBXCommon.TDBXConnection.FIsolationLevel 10401
DBXCommon.TDBXConnection.GetCommands 10406
DBXCommon.TDBXConnection.GetCommandTypes 10406
DBXCommon.TDBXConnection.GetConnectionProperties 10406
DBXCommon.TDBXConnection.GetDatabaseMetaData 10406
DBXCommon.TDBXConnection.GetErrorEvent 10407
DBXCommon.TDBXConnection.GetIsOpen 10407
DBXCommon.TDBXConnection.GetTraceFlags 10407
DBXCommon.TDBXConnection.GetTraceInfoEvent 10407
DBXCommon.TDBXConnection.IsOpen 10411
DBXCommon.TDBXConnection.OnErrorEvent 10411
DBXCommon.TDBXConnection.OnTrace 10412
DBXCommon.TDBXConnection.Open 10407
DBXCommon.TDBXConnection.Rollback 10408
DBXCommon.TDBXConnection.RollbackFreeAndNil 10408
DBXCommon.TDBXConnection.RollbackIncompleteFreeAndNil 10408
DBXCommon.TDBXConnection.SetConnectionProperties 10409
DBXCommon.TDBXConnection.SetErrorEvent 10409
DBXCommon.TDBXConnection.SetTraceFlags 10409
DBXCommon.TDBXConnection.SetTraceInfoEvent 10410
DBXCommon.TDBXConnection.TraceFlags 10411
DBXCommon.TDBXConnectionBuilder 10412
DBXCommon.TDBXConnectionBuilder.ConnectionProperties 10415
DBXCommon.TDBXConnectionBuilder.Create 10413
DBXCommon.TDBXConnectionBuilder.CreateDelegateeConnection 10414
DBXCommon.TDBXConnectionBuilder.DbxContext 10415
DBXCommon.TDBXConnectionBuilder.DelegationSignature 10415
DBXCommon.TDBXConnectionBuilder.Destroy 10414
DBXCommon.TDBXConnectionBuilder.InputConnectionName 10416
DBXCommon.TDBXConnectionBuilder.InputConnectionProperties 10416
DBXCommon.TDBXConnectionBuilder.InputPassword 10416
DBXCommon.TDBXConnectionBuilder.InputUserName 10417
DBXCommon.TDBXConnectionEx 10417
DBXCommon.TDBXConnectionEx.ConnectionProperty 10421
DBXCommon.TDBXConnectionEx.CreateMorphCommand 10419
DBXCommon.TDBXConnectionEx.Destroy 10419
DBXCommon.TDBXConnectionEx.GetConnectionProperty 10420
DBXCommon.TDBXConnectionEx.GetProductName 10420
DBXCommon.TDBXConnectionEx.GetProductVersion 10420
DBXCommon.TDBXConnectionEx.GetVendorProperty 10420
DBXCommon.TDBXConnectionEx.MetaDataReader 10421
DBXCommon.TDBXConnectionEx.ProductName 10421
DBXCommon.TDBXConnectionEx.ProductVersion 10421
DBXCommon.TDBXConnectionFactory 10422
DBXCommon.TDBXConnectionFactory.Close 10425
DBXCommon.TDBXConnectionFactory.Create 10423
DBXCommon.TDBXConnectionFactory.CreateDelegatePaths 10426
DBXCommon.TDBXConnectionFactory.DerivedClose 10426
DBXCommon.TDBXConnectionFactory.DerivedOpen 10426
DBXCommon.TDBXConnectionFactory.Destroy 10424
DBXCommon.TDBXConnectionFactory.FDBXContext 10424
DBXCommon.TDBXConnectionFactory.GetConnection 10426
DBXCommon.TDBXConnectionFactory.GetConnectionDriver

10427

DBXCommon.TDBXConnectionFactory.GetConnectionFactory 10427

DBXCommon.TDBXConnectionFactory.GetConnectionItems 10428

DBXCommon.TDBXConnectionFactory.GetConnectionPropertie s
10428

DBXCommon.TDBXConnectionFactory.GetDriver 10428

DBXCommon.TDBXConnectionFactory.GetDriverProperties 10429

DBXCommon.TDBXConnectionFactory.GetErrorEvent 10429

DBXCommon.TDBXConnectionFactory.GetTraceFlags 10429

DBXCommon.TDBXConnectionFactory.GetTraceInfoEvent 10429

DBXCommon.TDBXConnectionFactory.OnError 10432

DBXCommon.TDBXConnectionFactory.OnTrace 10432

DBXCommon.TDBXConnectionFactory.Open 10430

DBXCommon.TDBXConnectionFactory.SetConnectionFactory 10430

DBXCommon.TDBXConnectionFactory.SetErrorEvent 10430

DBXCommon.TDBXConnectionFactory.SetTraceFlags 10430

DBXCommon.TDBXConnectionFactory.SetTraceInfoEvent 10431

DBXCommon.TDBXConnectionFactory.TraceFlags 10431

DBXCommon.TDBXConnectionFile 10683

DBXCommon.TDBXConnectionFile constant 10683

DBXCommon.TDBXContext 10432

DBXCommon.TDBXContext.ClosedByteReader 10436

DBXCommon.TDBXContext.Create 10434

DBXCommon.TDBXContext.Destroy 10434

DBXCommon.TDBXContext.Error 10435

DBXCommon.TDBXContext.FOnErrorDBXContext 10435

DBXCommon.TDBXContext.FOnTraceDBXContext 10435

DBXCommon.TDBXContext.IsTracing 10436

DBXCommon.TDBXContext.OnError 10437

DBXCommon.TDBXContext.OnTrace 10438

DBXCommon.TDBXContext.Trace 10436

DBXCommon.TDBXContext.TraceFlags 10437

DBXCommon.TDBXCreateCommandEvent 10669

DBXCommon.TDBXCreateCommandEvent type 10669

DBXCommon.TDBXCursorValue 10438

DBXCommon.TDBXCursorValue.IsNull 10440

DBXCommon.TDBXDatabaseMetaData 10440

DBXCommon.TDBXDatabaseMetaData.Create 10441

DBXCommon.TDBXDatabaseMetaData.FDBXContext 10442

DBXCommon.TDBXDatabaseMetaData.MaxCommands 10442

DBXCommon.TDBXDatabaseMetaData.ProcedureQuoteChar 10442

DBXCommon.TDBXDatabaseMetaData.QuoteChar 10443

DBXCommon.TDBXDatabaseMetaData.SupportsNestedTransa ctions
10443

DBXCommon.TDBXDatabaseMetaData.SupportsRowSetSize 10443

DBXCommon.TDBXDatabaseMetaData.SupportsTransactions 10444

DBXCommon.TDBXDatabaseMetaDataEx 10444

DBXCommon.TDBXDatabaseMetaDataEx.Create 10445

DBXCommon.TDBXDatabaseMetaDataEx.MetaDataVersion 10446

DBXCommon.TDBXDatabaseMetaDataEx.QuotePrefix 10446

DBXCommon.TDBXDatabaseMetaDataEx.QuoteSuffix 10446

DBXCommon.TDBXDatabaseMetaDataEx.SupportsLowerCaseI dentifiers
10446

DBXCommon.TDBXDatabaseMetaDataEx.SupportsUpperCaseI dentifiers
10447

DBXCommon.TDBXDataTypes 10447

DBXCommon.TDBXDataTypes.AdtDateSubType 10450

DBXCommon.TDBXDataTypes.AdtNestedTableSubType 10451

DBXCommon.TDBXDataTypes.AdtType 10451

DBXCommon.TDBXDataTypes.AnsiStringType 10451

DBXCommon.TDBXDataTypes.ArrayType 10451

DBXCommon.TDBXDataTypes.AutoIncSubType 10451

DBXCommon.TDBXDataTypes.BcdType 10452

DBXCommon.TDBXDataTypes.BFileSubType 10452

DBXCommon.TDBXDataTypes.BinarySubType 10452

DBXCommon.TDBXDataTypes.BlobType 10452

DBXCommon.TDBXDataTypes.BooleanType 10453

DBXCommon.TDBXDataTypes.BytesType 10453

DBXCommon.TDBXDataTypes.CursorType 10453

DBXCommon.TDBXDataTypes.DateType 10453

DBXCommon.TDBXDataTypes.DoubleType 10453

DBXCommon.TDBXDataTypes.FixedSubType 10454
DBXCommon.TDBXDataTypes.HBinarySubType 10454
DBXCommon.TDBXDataTypes.HMemoSubType 10454
DBXCommon.TDBXDataTypes.Int16Type 10454
DBXCommon.TDBXDataTypes.Int32Type 10455
DBXCommon.TDBXDataTypes.Int64Type 10455
DBXCommon.TDBXDataTypes.MaxBaseTypes 10455
DBXCommon.TDBXDataTypes.MemoSubType 10455
DBXCommon.TDBXDataTypes.MoneySubType 10455
DBXCommon.TDBXDataTypes.OracleIntervalSubType 10456
DBXCommon.TDBXDataTypes.OracleTimeStampSubType 10456
DBXCommon.TDBXDataTypes.RefType 10456
DBXCommon.TDBXDataTypes.TableType 10456
DBXCommon.TDBXDataTypes.TimeStampType 10457
DBXCommon.TDBXDataTypes.TimeType 10457
DBXCommon.TDBXDataTypes.UInt16Type 10457
DBXCommon.TDBXDataTypes.Uint32Type 10457
DBXCommon.TDBXDataTypes.Uint64Type 10457
DBXCommon.TDBXDataTypes.UnknownType 10458
DBXCommon.TDBXDataTypes.VarBytesType 10458
DBXCommon.TDBXDataTypes.WideMemoSubType 10458
DBXCommon.TDBXDataTypes.WideStringType 10458
DBXCommon.TDBXDataTypesEx 10459
DBXCommon.TDBXDataTypesEx.CharArrayType 10460
DBXCommon.TDBXDataTypesEx.Int8Type 10460
DBXCommon.TDBXDataTypesEx.IntervalType 10460
DBXCommon.TDBXDataTypesEx.ObjectType 10460
DBXCommon.TDBXDataTypesEx.SingleType 10461
DBXCommon.TDBXDataTypesEx.Uint8Type 10461
DBXCommon.TDBXDate 10670
DBXCommon.TDBXDate type 10670
DBXCommon.TDBXDateValue 10461
DBXCommon.TDBXDateValue.GetDate 10463
DBXCommon.TDBXDateValue.IsNull 10463
DBXCommon.TDBXDateValue.SetDate 10463
DBXCommon.TDBXDelegateDriver 10464
DBXCommon.TDBXDelegateDriver.Close 10466
DBXCommon.TDBXDelegateDriver.Create 10465
DBXCommon.TDBXDelegateDriver.CreateConnection 10466
DBXCommon.TDBXDelegateDriver.CreateMorphCommand 10466
DBXCommon.TDBXDelegateDriver.Destroy 10465
DBXCommon.TDBXDelegateDriver.GetDriverName 10466
DBXCommon.TDBXDelegateDriver.GetDriverProperties 10467
DBXCommon.TDBXDelegateDriver.GetDriverVersion 10467
DBXCommon.TDBXDelegateDriver.SetDriverName 10467
DBXCommon.TDBXDelegateDriver.SetDriverProperties 10467
DBXCommon.TDBXDelegateItem 10468
DBXCommon.TDBXDelegateItem.Destroy 10468
DBXCommon.TDBXDelegateItem.FName 10469
DBXCommon.TDBXDelegateItem.FNext 10469
DBXCommon.TDBXDelegateItem.FProperties 10469
DBXCommon.TDBXDoubleValue 10469
DBXCommon.TDBXDoubleValue.GetDouble 10471
DBXCommon.TDBXDoubleValue.IsNull 10471
DBXCommon.TDBXDoubleValue.SetDouble 10472
DBXCommon.TDBXDriver 10472
DBXCommon.TDBXDriver.CacheUntilFinalization 10474
DBXCommon.TDBXDriver.Close 10474
DBXCommon.TDBXDriver.CreateConnection 10475
DBXCommon.TDBXDriver.Destroy 10473
DBXCommon.TDBXDriver.DriverName 10476
DBXCommon.TDBXDriver.DriverProperties 10477
DBXCommon.TDBXDriver.FDBXContext 10473
DBXCommon.TDBXDriver.FDriverLoader 10474
DBXCommon.TDBXDriver.GetDriverName 10475
DBXCommon.TDBXDriver.GetDriverProperties 10475
DBXCommon.TDBXDriver.GetDriverVersion 10475
DBXCommon.TDBXDriver.SetDriverName 10476
DBXCommon.TDBXDriver.SetDriverProperties 10476
DBXCommon.TDBXDriverDef 10668
DBXCommon.TDBXDriverDef record 10668
DBXCommon.TDBXDriverEx 10477
DBXCommon.TDBXDriverEx.AddCommandFactory 10479
DBXCommon.TDBXDriverEx.Create 10478
DBXCommon.TDBXDriverEx.CreateMorphCommand 10479
DBXCommon.TDBXDriverEx.Destroy 10478
DBXCommon.TDBXDriverEx.FCommandFactories 10479
DBXCommon.TDBXDriverEx.LoadMetaDataCommandFactory

10479
DBXCommon.TDBXDriverFile 10683
DBXCommon.TDBXDriverFile constant 10683
DBXCommon.TDBXDriverHelp 10480
DBXCommon.TDBXDriverHelp.CreateConnectionBuilder 10481
DBXCommon.TDBXDriverHelp.CreateTDBXContext 10481
DBXCommon.TDBXDriverHelp.CreateTDBXParameter 10481
DBXCommon.TDBXDriverHelp.CreateTDBXProperties 10481
DBXCommon.TDBXDriverHelp.CreateTDBXValueType 10481
DBXCommon.TDBXDriverHelpEx 10482
DBXCommon.TDBXDriverHelpEx.CreateTDBXValueType 10483
DBXCommon.TDBXDriverHelpEx.GetStreamReader 10483
DBXCommon.TDBXDriverHelpEx.IsReadOnlyValueType 10483
DBXCommon.TDBXDriverHelpEx.UpdateParameterType 10483
DBXCommon.TDBXDriverLoader 10483
DBXCommon.TDBXDriverLoader.FLoaderName 10484
DBXCommon.TDBXDriverLoader.Load 10485
DBXCommon.TDBXDriverLoaderClass 10670
DBXCommon.TDBXDriverLoaderClass type 10670
DBXCommon.TDBXDriverRegistry 10485
DBXCommon.TDBXDriverRegistry.Create 10486
DBXCommon.TDBXDriverRegistry.Destroy 10486
DBXCommon.TDBXError 10486
DBXCommon.TDBXError.Create 10488
DBXCommon.TDBXError.ErrorCode 10489
DBXCommon.TDBXError.ErrorCodeToString 10488
DBXCommon.TDBXError.Message 10489
DBXCommon.TDBXErrorCode 10670
DBXCommon.TDBXErrorCode type 10670
DBXCommon.TDBXErrorCodes 10489
DBXCommon.TDBXErrorCodes.ConnectionFailed 10493
DBXCommon.TDBXErrorCodes.DriverIncompatible 10493
DBXCommon.TDBXErrorCodes.DriverInitFailed 10493
DBXCommon.TDBXErrorCodes.DriverRestricted 10494
DBXCommon.TDBXErrorCodes.DuplicateTransactionId 10494
DBXCommon.TDBXErrorCodes.EOF 10494
DBXCommon.TDBXErrorCodes.InvalidArgument 10494
DBXCommon.TDBXErrorCodes.InvalidHandle 10495
DBXCommon.TDBXErrorCodes.InvalidIsolationLevel 10495
DBXCommon.TDBXErrorCodes.InvalidLength 10495
DBXCommon.TDBXErrorCodes.InvalidOperation 10495
DBXCommon.TDBXErrorCodes.InvalidOrdinal 10495
DBXCommon.TDBXErrorCodes.InvalidParameter 10496
DBXCommon.TDBXErrorCodes.InvalidPrecision 10496
DBXCommon.TDBXErrorCodes.InvalidReference 10496
DBXCommon.TDBXErrorCodes.InvalidTime 10496
DBXCommon.TDBXErrorCodes.InvalidTransactionId 10497
DBXCommon.TDBXErrorCodes.InvalidType 10497
DBXCommon.TDBXErrorCodes.InvalidUserOrPassword 10497
DBXCommon.TDBXErrorCodes.MaxCommonErrors 10497
DBXCommon.TDBXErrorCodes.MissingParameterMarker 10497
DBXCommon.TDBXErrorCodes.MultipleTransactionNotEnabled 10498
DBXCommon.TDBXErrorCodes.NoData 10498
DBXCommon.TDBXErrorCodes.NoMemory 10498
DBXCommon.TDBXErrorCodes.None 10498
DBXCommon.TDBXErrorCodes.NoTable 10499
DBXCommon.TDBXErrorCodes.NotImplemented 10499
DBXCommon.TDBXErrorCodes.NotSupported 10499
DBXCommon.TDBXErrorCodes.OptimisticLockFailed 10499
DBXCommon.TDBXErrorCodes.ParameterNotSet 10499
DBXCommon.TDBXErrorCodes.TransactionActive 10500
DBXCommon.TDBXErrorCodes.UnsupportedFieldType 10500
DBXCommon.TDBXErrorCodes.VendorError 10500
DBXCommon.TDBXErrorCodes.Warning 10500
DBXCommon.TDBXErrorEvent 10671
DBXCommon.TDBXErrorEvent type 10671
DBXCommon.TDBXIniFileConnectionFactory 10501
DBXCommon.TDBXIniFileConnectionFactory.ConnectionsFile 10503
DBXCommon.TDBXIniFileConnectionFactory.Create 10502
DBXCommon.TDBXIniFileConnectionFactory.DerivedClose 10503
DBXCommon.TDBXIniFileConnectionFactory.DerivedOpen 10503
DBXCommon.TDBXIniFileConnectionFactory.Destroy 10502
DBXCommon.TDBXIniFileConnectionFactory.DriversFile 10504
DBXCommon.TDBXIniFileConnectionFactoryEx 10504
DBXCommon.TDBXIniFileConnectionFactoryEx.GetDriverItems

10506
DBXCommon.TDBXInt16Value 10506
DBXCommon.TDBXInt16Value.GetInt16 10508
DBXCommon.TDBXInt16Value.IsNull 10508
DBXCommon.TDBXInt16Value.SetInt16 10508
DBXCommon.TDBXInt32Value 10508
DBXCommon.TDBXInt32Value.GetInt32 10510
DBXCommon.TDBXInt32Value.IsNull 10510
DBXCommon.TDBXInt32Value.SetInt32 10511
DBXCommon.TDBXInt64Value 10511
DBXCommon.TDBXInt64Value.GetInt64 10512
DBXCommon.TDBXInt64Value.IsNull 10513
DBXCommon.TDBXInt64Value.SetInt64 10513
DBXCommon.TDBXIsolation 10671
DBXCommon.TDBXIsolation type 10671
DBXCommon.TDBXIsolations 10513
DBXCommon.TDBXIsolations.DirtyRead 10515
DBXCommon.TDBXIsolations.ReadCommitted 10515
DBXCommon.TDBXIsolations.RepeatableRead 10515
DBXCommon.TDBXIsolations.Serializable 10515
DBXCommon.TDBXIsolations.SnapShot 10515
DBXCommon.TDBXLookAheadStreamReader 10516
DBXCommon.TDBXLookAheadStreamReader.ConvertToMemoryStream 10517
DBXCommon.TDBXLookAheadStreamReader.Destroy 10516
DBXCommon.TDBXLookAheadStreamReader.Eos 10517
DBXCommon.TDBXLookAheadStreamReader.Read 10517
DBXCommon.TDBXLookAheadStreamReader.SetStream 10518
DBXCommon.TDBXLookAheadStreamReader.Size 10518
DBXCommon.TDBXMemoryConnectionFactory 10518
DBXCommon.TDBXMemoryConnectionFactory.Create 10519
DBXCommon.TDBXMemoryConnectionFactory.DerivedClose 10520
DBXCommon.TDBXMemoryConnectionFactory.DerivedOpen 10520
DBXCommon.TDBXMemoryConnectionFactory.Destroy 10520
DBXCommon.TDBXMetaDatabaseColumnNames 10521
DBXCommon.TDBXMetaDatabaseColumnNames.MaxCommands 10523
DBXCommon.TDBXMetaDatabaseColumnNames.ProcedureQuoteChar 10523
DBXCommon.TDBXMetaDatabaseColumnNames.ProductName 10523
DBXCommon.TDBXMetaDatabaseColumnNames.ProductVersion 10523
DBXCommon.TDBXMetaDatabaseColumnNames.QuoteChar 10524
DBXCommon.TDBXMetaDatabaseColumnNames.QuotePrefix 10524
DBXCommon.TDBXMetaDatabaseColumnNames.QuoteSuffix 10524
DBXCommon.TDBXMetaDatabaseColumnNames.SupportsLowerCaseIdentifiers 10524
DBXCommon.TDBXMetaDatabaseColumnNames.SupportsNestedTransactions 10524
DBXCommon.TDBXMetaDatabaseColumnNames.SupportsRowSetSize 10525
DBXCommon.TDBXMetaDatabaseColumnNames.SupportsTransactions 10525
DBXCommon.TDBXMetaDatabaseColumnNames.SupportsUpperCaseIdentifiers 10525
DBXCommon.TDBXMetaDataCommands 10525
DBXCommon.TDBXMetaDataCommands.GetColumns 10529
DBXCommon.TDBXMetaDataCommands.GetDatabase 10529
DBXCommon.TDBXMetaDataCommands.GetDataTypes 10529
DBXCommon.TDBXMetaDataCommands.GetForeignKeyColumns 10529
DBXCommon.TDBXMetaDataCommands.GetForeignKeys 10530
DBXCommon.TDBXMetaDataCommands.GetIndexColumns 10530
DBXCommon.TDBXMetaDataCommands.GetIndexes 10530
DBXCommon.TDBXMetaDataCommands.GetPackages 10530
DBXCommon.TDBXMetaDataCommands.GetProcedureParameters 10531
DBXCommon.TDBXMetaDataCommands.GetProcedures 10531
DBXCommon.TDBXMetaDataCommands.GetTables 10531

DBXCommon.TDBXMetaDataCommands.GetUsers 10531
DBXCommon.TDBXMetaDataCommandsEx 10531
DBXCommon.TDBXMetaDataCommandsEx.GetCatalogs 10535
DBXCommon.TDBXMetaDataCommandsEx.GetPackageProcedureParameters 10535
DBXCommon.TDBXMetaDataCommandsEx.GetPackageProcedures 10535
DBXCommon.TDBXMetaDataCommandsEx.GetPackageSources 10535
DBXCommon.TDBXMetaDataCommandsEx.GetProcedureSources 10536
DBXCommon.TDBXMetaDataCommandsEx.GetReservedWords 10536
DBXCommon.TDBXMetaDataCommandsEx.GetRoles 10536
DBXCommon.TDBXMetaDataCommandsEx.GetSchemas 10536
DBXCommon.TDBXMetaDataCommandsEx.GetSynonyms 10536
DBXCommon.TDBXMetaDataCommandsEx.GetViews 10537
DBXCommon.TDBXMetaDataKeyword 10537
DBXCommon.TDBXMetaDataKeyword.ForeignKey 10538
DBXCommon.TDBXMetaDataKeyword.PrimaryKey 10538
DBXCommon.TDBXMetaDataProcedureTypes 10538
DBXCommon.TDBXMetaDataProcedureTypes.FunctionType 10539
DBXCommon.TDBXMetaDataProcedureTypes.ProcedureType 10539
DBXCommon.TDBXMetaDataTableTypes 10540
DBXCommon.TDBXMetaDataTableTypes.Synonym 10541
DBXCommon.TDBXMetaDataTableTypes.SystemTable 10541
DBXCommon.TDBXMetaDataTableTypes.Table 10541
DBXCommon.TDBXMetaDataTableTypes.View 10541
DBXCommon.TDBXMetaDataTableTypesEx 10542
DBXCommon.TDBXMetaDataTableTypesEx.SystemView 10543
DBXCommon.TDBXNullValue 10543
DBXCommon.TDBXNullValue.Create 10544
DBXCommon.TDBXNullValue.Destroy 10544
DBXCommon.TDBXNullValue.IsNull 10545
DBXCommon.TDBXParameter 10545
DBXCommon.TDBXParameter.Create 10547
DBXCommon.TDBXParameter.Destroy 10547
DBXCommon.TDBXParameter.GetValue 10547
DBXCommon.TDBXParameter.SetDbxRow 10548
DBXCommon.TDBXParameter.SetParameter 10548
DBXCommon.TDBXParameter.Value 10548
DBXCommon.TDBXParameterDirection 10671
DBXCommon.TDBXParameterDirection type 10671
DBXCommon.TDBXParameterDirections 10549
DBXCommon.TDBXParameterDirections.InOutParameter 10550
DBXCommon.TDBXParameterDirections.InParameter 10550
DBXCommon.TDBXParameterDirections.OutParameter 10550
DBXCommon.TDBXParameterDirections.ReturnParameter 10550
DBXCommon.TDBXParameterDirections.Unknown 10551
DBXCommon.TDBXParameterEx 10551
DBXCommon.TDBXParameterEx.Assign 10553
DBXCommon.TDBXParameterEx.Clone 10553
DBXCommon.TDBXParameterList 10553
DBXCommon.TDBXParameterList.AddParameter 10556
DBXCommon.TDBXParameterList.ClearParameters 10556
DBXCommon.TDBXParameterList.Count 10559
DBXCommon.TDBXParameterList.Create 10554
DBXCommon.TDBXParameterList.Destroy 10555
DBXCommon.TDBXParameterList.FDBXContext 10555
DBXCommon.TDBXParameterList.GetCount 10556
DBXCommon.TDBXParameterList.GetOrdinal 10556
DBXCommon.TDBXParameterList.GetParameterByOrdinal 10557
DBXCommon.TDBXParameterList.InsertParameter 10557
DBXCommon.TDBXParameterList.Parameter 10559
DBXCommon.TDBXParameterList.RemoveParameter 10558
DBXCommon.TDBXParameterList.SetCount 10558
DBXCommon.TDBXParameterList.SetParameter 10558
DBXCommon.TDBXProperties 10559
DBXCommon.TDBXProperties.Add 10561
DBXCommon.TDBXProperties.AddProperties 10561
DBXCommon.TDBXProperties.Clone 10562
DBXCommon.TDBXProperties.Create 10560
DBXCommon.TDBXProperties.Destroy 10561

DBXCommon.TDBXProperties.GetBoolean 10562
DBXCommon.TDBXProperties.GetInteger 10562
DBXCommon.TDBXProperties.GetLists 10562
DBXCommon.TDBXProperties.GetRequiredValue 10563
DBXCommon.TDBXProperties.Properties 10563
DBXCommon.TDBXProperties.SetProperties 10563
DBXCommon.TDBXProperties.Values 10564
DBXCommon.TDBXPropertiesItem 10564
DBXCommon.TDBXPropertiesItem.Create 10565
DBXCommon.TDBXPropertiesItem.Destroy 10565
DBXCommon.TDBXPropertiesItem.FDeletgatePath 10565
DBXCommon.TDBXPropertiesItem.FProperties 10566
DBXCommon.TDBXPropertyNames 10566
DBXCommon.TDBXPropertyNames.ConnectionName 10569
DBXCommon.TDBXPropertyNames.Database 10569
DBXCommon.TDBXPropertyNames.DelegateConnection 10569
DBXCommon.TDBXPropertyNames.DelegateDriver 10570
DBXCommon.TDBXPropertyNames.DriverAssembly 10570
DBXCommon.TDBXPropertyNames.DriverAssemblyLoader 10570
DBXCommon.TDBXPropertyNames.DriverName 10570
DBXCommon.TDBXPropertyNames.DriverPackage 10570
DBXCommon.TDBXPropertyNames.DriverPackageLoader 10571
DBXCommon.TDBXPropertyNames.DriverUnit 10571
DBXCommon.TDBXPropertyNames.ErrorResourceFile 10571
DBXCommon.TDBXPropertyNames.GetDriverFunc 10571
DBXCommon.TDBXPropertyNames.HostName 10572
DBXCommon.TDBXPropertyNames.InstalledDrivers 10572
DBXCommon.TDBXPropertyNames.IsolationLevel 10572
DBXCommon.TDBXPropertyNames.LibraryName 10572
DBXCommon.TDBXPropertyNames.MaxBlobSize 10572
DBXCommon.TDBXPropertyNames.Password 10573
DBXCommon.TDBXPropertyNames.Port 10573
DBXCommon.TDBXPropertyNames.Role 10573
DBXCommon.TDBXPropertyNames.SchemaOverride 10573
DBXCommon.TDBXPropertyNames.UserName 10574
DBXCommon.TDBXPropertyNames.VendorLib 10574
DBXCommon.TDBXPropertyNamesEx 10574
DBXCommon.TDBXPropertyNamesEx.MetaDataAssemblyLoader 10575
DBXCommon.TDBXPropertyNamesEx.MetaDataPackageLoader 10576
DBXCommon.TDBXPropertyNamesEx.ProductName 10576
DBXCommon.TDBXPropertyNamesEx.ProductVersion 10576
DBXCommon.TDBXReader 10576
DBXCommon.TDBXReader.ByteReader 10583
DBXCommon.TDBXReader.Closed 10583
DBXCommon.TDBXReader.ColumnCount 10584
DBXCommon.TDBXReader.Create 10578
DBXCommon.TDBXReader.DerivedClose 10580
DBXCommon.TDBXReader.DerivedNext 10580
DBXCommon.TDBXReader.Destroy 10578
DBXCommon.TDBXReader.FailIfClosed 10580
DBXCommon.TDBXReader.FDBXContext 10579
DBXCommon.TDBXReader.FDbxRow 10579
DBXCommon.TDBXReader.GetByteReader 10580
DBXCommon.TDBXReader.GetColumnCount 10580
DBXCommon.TDBXReader.GetErrorEvent 10581
DBXCommon.TDBXReader.GetObjectTypeName 10581
DBXCommon.TDBXReader.GetOrdinal 10581
DBXCommon.TDBXReader.GetValue 10582
DBXCommon.TDBXReader.GetValueByName 10582
DBXCommon.TDBXReader.GetValueType 10582
DBXCommon.TDBXReader.Next 10582
DBXCommon.TDBXReader.OnErrorEvent 10585
DBXCommon.TDBXReader.SetValues 10583
DBXCommon.TDBXReader.Value 10584
DBXCommon.TDBXReader.ValueType 10584
DBXCommon.TDBXReaderByteReader 10585
DBXCommon.TDBXReaderByteReader.Create 10586
DBXCommon.TDBXReaderByteReader.GetAnsiString 10587
DBXCommon.TDBXReaderByteReader.GetBcd 10587
DBXCommon.TDBXReaderByteReader.GetByteLength 10587
DBXCommon.TDBXReaderByteReader.GetBytes 10587
DBXCommon.TDBXReaderByteReader.GetDate 10588
DBXCommon.TDBXReaderByteReader.GetDouble 10588
DBXCommon.TDBXReaderByteReader.GetInt16 10588

DBXCommon.TDBXReaderByteReader.GetInt32 10588
DBXCommon.TDBXReaderByteReader.GetInt64 10589
DBXCommon.TDBXReaderByteReader.GetTime 10589
DBXCommon.TDBXReaderByteReader.GetTimeStamp 10589
DBXCommon.TDBXReaderByteReader.GetWideString 10589
DBXCommon.TDBXRegistryConnectionValue 10684
DBXCommon.TDBXRegistryConnectionValue constant 10684
DBXCommon.TDBXRegistryDriverValue 10684
DBXCommon.TDBXRegistryDriverValue constant 10684
DBXCommon.TDBXRegistryKey 10684
DBXCommon.TDBXRegistryKey constant 10684
DBXCommon.TDBXRow 10590
DBXCommon.TDBXRow.Create 10591
DBXCommon.TDBXRow.FDBXContext 10591
DBXCommon.TDBXRow.FGeneration 10592
DBXCommon.TDBXRow.Generation 10599
DBXCommon.TDBXRow.GetAnsiString 10593
DBXCommon.TDBXRow.GetBcd 10593
DBXCommon.TDBXRow.GetBoolean 10593
DBXCommon.TDBXRow.GetByteLength 10593
DBXCommon.TDBXRow.GetBytes 10594
DBXCommon.TDBXRow.GetDate 10594
DBXCommon.TDBXRow.GetDouble 10594
DBXCommon.TDBXRow.GetInt16 10594
DBXCommon.TDBXRow.GetInt32 10595
DBXCommon.TDBXRow.GetInt64 10595
DBXCommon.TDBXRow.GetObjectTypeName 10595
DBXCommon.TDBXRow.GetTime 10595
DBXCommon.TDBXRow.GetTimeStamp 10595
DBXCommon.TDBXRow.GetWideString 10596
DBXCommon.TDBXRow.NotImplemented 10596
DBXCommon.TDBXRow.SetBCD 10596
DBXCommon.TDBXRow.SetBoolean 10596
DBXCommon.TDBXRow.SetDate 10597
DBXCommon.TDBXRow.SetDouble 10597
DBXCommon.TDBXRow.SetDynamicBytes 10597
DBXCommon.TDBXRow.SetInt16 10597
DBXCommon.TDBXRow.SetInt32 10597
DBXCommon.TDBXRow.SetInt64 10598
DBXCommon.TDBXRow.SetNull 10598
DBXCommon.TDBXRow.SetString 10598
DBXCommon.TDBXRow.SetTime 10598
DBXCommon.TDBXRow.SetTimestamp 10599
DBXCommon.TDBXRow.SetValueType 10599
DBXCommon.TDBXRow.SetWideString 10599
DBXCommon.TDBXRowEx 10600
DBXCommon.TDBXRowEx.CreateCustomValue 10601
DBXCommon.TDBXRowEx.GetLength 10602
DBXCommon.TDBXRowEx.GetStream 10602
DBXCommon.TDBXRowEx.GetStreamBytes 10603
DBXCommon.TDBXRowEx.GetStreamLength 10603
DBXCommon.TDBXRowEx.GetWideChars 10603
DBXCommon.TDBXRowEx.SetStream 10603
DBXCommon.TDBXRowEx.SetWideChars 10604
DBXCommon.TDBXRowEx.UseExtendedTypes 10604
DBXCommon.TDBXStateItem 10604
DBXCommon.TDBXStateItem.Close 10605
DBXCommon.TDBXStateItemList 10605
DBXCommon.TDBXStateItemList.AddStateItem 10606
DBXCommon.TDBXStateItemList.Destroy 10606
DBXCommon.TDBXStateItemList.GetStateItem 10606
DBXCommon.TDBXStateItemList.RemoveAndFreeStateItem 10606
DBXCommon.TDBXStreamReader 10607
DBXCommon.TDBXStreamReader.Eos 10608
DBXCommon.TDBXStreamReader.Read 10608
DBXCommon.TDBXStreamReader.Size 10608
DBXCommon.TDBXStreamValue 10608
DBXCommon.TDBXStreamValue.Create 10610
DBXCommon.TDBXStreamValue.Destroy 10610
DBXCommon.TDBXStreamValue.GetStream 10611
DBXCommon.TDBXStreamValue.GetValueSize 10611
DBXCommon.TDBXStreamValue.SetDynamicBytes 10611
DBXCommon.TDBXStreamValue.SetStream 10611
DBXCommon.TDBXStreamValue.UpdateType 10612
DBXCommon.TDBXTime 10672
DBXCommon.TDBXTime type 10672
DBXCommon.TDBXTimeStampValue 10612
DBXCommon.TDBXTimeStampValue.GetTimeStamp 10614

DBXCommon.TDBXTimeStampValue.IsNull 10614
DBXCommon.TDBXTimeStampValue.SetTimeStamp 10614
DBXCommon.TDBXTimeValue 10614
DBXCommon.TDBXTimeValue.GetTime 10616
DBXCommon.TDBXTimeValue.IsNull 10616
DBXCommon.TDBXTimeValue.SetTime 10617
DBXCommon.TDBXTraceEvent 10672
DBXCommon.TDBXTraceEvent type 10672
DBXCommon.TDBXTraceFlag 10673
DBXCommon.TDBXTraceFlag type 10673
DBXCommon.TDBXTraceFlags 10617
DBXCommon.TDBXTraceFlags.Blob 10619
DBXCommon.TDBXTraceFlags.Command 10619
DBXCommon.TDBXTraceFlags.Connect 10619
DBXCommon.TDBXTraceFlags.Driver 10619
DBXCommon.TDBXTraceFlags.DriverLoad 10619
DBXCommon.TDBXTraceFlags.Error 10620
DBXCommon.TDBXTraceFlags.Execute 10620
DBXCommon.TDBXTraceFlags.MetaData 10620
DBXCommon.TDBXTraceFlags.Misc 10620
DBXCommon.TDBXTraceFlags.None 10621
DBXCommon.TDBXTraceFlags.Parameter 10621
DBXCommon.TDBXTraceFlags.Prepare 10621
DBXCommon.TDBXTraceFlags.Reader 10621
DBXCommon.TDBXTraceFlags.Transact 10621
DBXCommon.TDBXTraceFlags.Vendor 10622
DBXCommon.TDBXTraceInfo 10622
DBXCommon.TDBXTraceInfo.Message 10623
DBXCommon.TDBXTraceInfo.TraceFlag 10623
DBXCommon.TDBXTransaction 10623
DBXCommon.TDBXTransaction.Connection 10625
DBXCommon.TDBXTransaction.Create 10624
DBXCommon.TDBXTransaction.Destroy 10624
DBXCommon.TDBXTransaction.FIsolationLevel 10625
DBXCommon.TDBXType 10673
DBXCommon.TDBXType type 10673
DBXCommon.TDBXValue 10626
DBXCommon.TDBXValue.Create 10627
DBXCommon.TDBXValue.CreateValue 10628
DBXCommon.TDBXValue.DbxContext 10633
DBXCommon.TDBXValue.Destroy 10627
DBXCommon.TDBXValue.GetAnsiString 10629
DBXCommon.TDBXValue.GetBcd 10629
DBXCommon.TDBXValue.GetBoolean 10629
DBXCommon.TDBXValue.GetBytes 10629
DBXCommon.TDBXValue.GetDate 10630
DBXCommon.TDBXValue.GetDbxContext 10630
DBXCommon.TDBXValue.GetDouble 10630
DBXCommon.TDBXValue.GetInt16 10631
DBXCommon.TDBXValue.GetInt32 10631
DBXCommon.TDBXValue.GetInt64 10631
DBXCommon.TDBXValue.GetTime 10631
DBXCommon.TDBXValue.GetTimeStamp 10632
DBXCommon.TDBXValue.GetValueSize 10632
DBXCommon.TDBXValue.GetWideString 10632
DBXCommon.TDBXValue.IsNull 10632
DBXCommon.TDBXValue.ValueType 10633
DBXCommon.TDBXValueArray 10673
DBXCommon.TDBXValueArray type 10673
DBXCommon.TDBXValueList 10634
DBXCommon.TDBXValueList.Destroy 10634
DBXCommon.TDBXValueList.FindOrdinal 10635
DBXCommon.TDBXValueList.FLastOrdinal 10635
DBXCommon.TDBXValueList.FValueTypeArray 10635
DBXCommon.TDBXValueType 10636
DBXCommon.TDBXValueType.AutoIncrement 10645
DBXCommon.TDBXValueType.ChildPosition 10645
DBXCommon.TDBXValueType.Create 10637
DBXCommon.TDBXValueType.DataType 10646
DBXCommon.TDBXValueType.FDBXContext 10637
DBXCommon.TDBXValueType.GetChildPosition 10638
DBXCommon.TDBXValueType.GetDataType 10639
DBXCommon.TDBXValueType.GetFlags 10639
DBXCommon.TDBXValueType.GetName 10639
DBXCommon.TDBXValueType.GetOrdinal 10639
DBXCommon.TDBXValueType.GetParameterDirection 10640
DBXCommon.TDBXValueType.GetPrecision 10640
DBXCommon.TDBXValueType.GetScale 10640

DBXCommon.TDBXValueType.GetSize 10641
DBXCommon.TDBXValueType.GetSubType 10641
DBXCommon.TDBXValueType.Name 10646
DBXCommon.TDBXValueType.Nullable 10646
DBXCommon.TDBXValueType.Ordinal 10646
DBXCommon.TDBXValueType.ParameterDirection 10647
DBXCommon.TDBXValueType.Precision 10647
DBXCommon.TDBXValueType.ReadOnly 10647
DBXCommon.TDBXValueType.Scale 10647
DBXCommon.TDBXValueType.Searchable 10648
DBXCommon.TDBXValueType.SetChildPosition 10641
DBXCommon.TDBXValueType.SetDataType 10642
DBXCommon.TDBXValueType.SetFlags 10642
DBXCommon.TDBXValueType.SetName 10642
DBXCommon.TDBXValueType.SetNullable 10642
DBXCommon.TDBXValueType.SetOrdinal 10643
DBXCommon.TDBXValueType.SetParameterDirection 10643
DBXCommon.TDBXValueType.SetPrecision 10643
DBXCommon.TDBXValueType.SetScale 10644
DBXCommon.TDBXValueType.SetSize 10644
DBXCommon.TDBXValueType.SetSubType 10644
DBXCommon.TDBXValueType.Size 10648
DBXCommon.TDBXValueType.SubType 10648
DBXCommon.TDBXValueType.ValueTypeFlags 10648
DBXCommon.TDBXValueTypeEx 10649
DBXCommon.TDBXValueTypeEx.DataTypeName 10650
DBXCommon.TDBXValueTypeEx.InvalidTypeAccess 10651
DBXCommon.TDBXValueTypeFlags 10651
DBXCommon.TDBXValueTypeFlags.AutoIncrement 10652
DBXCommon.TDBXValueTypeFlags.Nullable 10652
DBXCommon.TDBXValueTypeFlags.ReadOnly 10653
DBXCommon.TDBXValueTypeFlags.ReadOnlyType 10653
DBXCommon.TDBXValueTypeFlags.Searchable 10653
DBXCommon.TDBXValueTypeFlagsEx 10653
DBXCommon.TDBXValueTypeFlagsEx.ExtendedType 10654
DBXCommon.TDBXWideCharsValue 10655
DBXCommon.TDBXWideCharsValue.Create 10656
DBXCommon.TDBXWideCharsValue.Destroy 10656
DBXCommon.TDBXWideCharsValue.GetStream 10657
DBXCommon.TDBXWideCharsValue.GetWideString 10657
DBXCommon.TDBXWideCharsValue.IsNull 10657
DBXCommon.TDBXWideCharsValue.SetWideString 10658
DBXCommon.TDBXWideStringValue 10658
DBXCommon.TDBXWideStringValue.Create 10659
DBXCommon.TDBXWideStringValue.Destroy 10660
DBXCommon.TDBXWideStringValue.GetWideString 10660
DBXCommon.TDBXWideStringValue.IsNull 10660
DBXCommon.TDBXWideStringValue.SetWideString 10661
DBXCommon.TDBXWritableValue 10661
DBXCommon.TDBXWritableValue.Create 10662
DBXCommon.TDBXWritableValue.Destroy 10663
DBXCommon.TDBXWritableValue.SetAnsiString 10663
DBXCommon.TDBXWritableValue.SetBcd 10664
DBXCommon.TDBXWritableValue.SetBoolean 10664
DBXCommon.TDBXWritableValue.SetDate 10664
DBXCommon.TDBXWritableValue.SetDouble 10664
DBXCommon.TDBXWritableValue.SetDynamicBytes 10665
DBXCommon.TDBXWritableValue.SetInt16 10665
DBXCommon.TDBXWritableValue.SetInt32 10665
DBXCommon.TDBXWritableValue.SetInt64 10665
DBXCommon.TDBXWritableValue.SetNull 10666
DBXCommon.TDBXWritableValue.SetPendingValue 10666
DBXCommon.TDBXWritableValue.SetStaticBytes 10666
DBXCommon.TDBXWritableValue.SetTime 10666
DBXCommon.TDBXWritableValue.SetTimeStamp 10667
DBXCommon.TDBXWritableValue.SetWideString 10667
DBXCommon.TWideStringArray 10674
DBXCommon.TWideStringArray type 10674
DBXMetaDataNames 10685
DBXMetaDataNames namespace 10685
Classes 10687
DBXMetaDataNames.TDBXCatalogsColumns 10688
DBXMetaDataNames.TDBXCatalogsColumns.CatalogName 10689
DBXMetaDataNames.TDBXCatalogsIndex 10689
DBXMetaDataNames.TDBXCatalogsIndex.CatalogName 10690
DBXMetaDataNames.TDBXCatalogsIndex.Last 10690
DBXMetaDataNames.TDBXColumnConstraintsColumns 10690
DBXMetaDataNames.TDBXColumnConstraintsColumns.Catalo

gName
10691

DBXMetaDataNames.TDBXColumnConstraintsColumns.ColumnName
10691

DBXMetaDataNames.TDBXColumnConstraintsColumns.ConstraintName
10692

DBXMetaDataNames.TDBXColumnConstraintsColumns.SchemaName
10692

DBXMetaDataNames.TDBXColumnConstraintsColumns.TableName
10692

DBXMetaDataNames.TDBXColumnConstraintsIndex 10692

DBXMetaDataNames.TDBXColumnConstraintsIndex.CatalogName
10693

DBXMetaDataNames.TDBXColumnConstraintsIndex.ColumnName
10693

DBXMetaDataNames.TDBXColumnConstraintsIndex.ConstraintName
10694

DBXMetaDataNames.TDBXColumnConstraintsIndex.Last
10694

DBXMetaDataNames.TDBXColumnConstraintsIndex.SchemaName
10694

DBXMetaDataNames.TDBXColumnConstraintsIndex.TableName
10694

DBXMetaDataNames.TDBXColumnsColumns 10695

DBXMetaDataNames.TDBXColumnsColumns.CatalogName
10697

DBXMetaDataNames.TDBXColumnsColumns.ColumnName
10697

DBXMetaDataNames.TDBXColumnsColumns.DbxDataType
10697

DBXMetaDataNames.TDBXColumnsColumns.DefaultValue
10698

DBXMetaDataNames.TDBXColumnsColumns.IsAutoIncrement
10698

DBXMetaDataNames.TDBXColumnsColumns.IsFixedLength
10698

DBXMetaDataNames.TDBXColumnsColumns.IsLong 10698

DBXMetaDataNames.TDBXColumnsColumns.IsNullable 10698

DBXMetaDataNames.TDBXColumnsColumns.IsUnicode 10699

DBXMetaDataNames.TDBXColumnsColumns.IsUnsigned
10699

DBXMetaDataNames.TDBXColumnsColumns.MaxInline 10699

DBXMetaDataNames.TDBXColumnsColumns.Ordinal 10699

DBXMetaDataNames.TDBXColumnsColumns.Precision 10700

DBXMetaDataNames.TDBXColumnsColumns.Scale 10700

DBXMetaDataNames.TDBXColumnsColumns.SchemaName
10700

DBXMetaDataNames.TDBXColumnsColumns.TableName
10700

DBXMetaDataNames.TDBXColumnsColumns.TypeName 10700

DBXMetaDataNames.TDBXColumnsIndex 10701

DBXMetaDataNames.TDBXColumnsIndex.CatalogName 10703

DBXMetaDataNames.TDBXColumnsIndex.ColumnName 10703

DBXMetaDataNames.TDBXColumnsIndex.DbxDataType 10703

DBXMetaDataNames.TDBXColumnsIndex.DefaultValue 10704

DBXMetaDataNames.TDBXColumnsIndex.IsAutoIncrement
10704

DBXMetaDataNames.TDBXColumnsIndex.IsFixedLength 10704

DBXMetaDataNames.TDBXColumnsIndex.IsLong 10704

DBXMetaDataNames.TDBXColumnsIndex.IsNullable 10705

DBXMetaDataNames.TDBXColumnsIndex.IsUnicode 10705

DBXMetaDataNames.TDBXColumnsIndex.IsUnsigned 10705

DBXMetaDataNames.TDBXColumnsIndex.Last 10705

DBXMetaDataNames.TDBXColumnsIndex.MaxInline 10705

DBXMetaDataNames.TDBXColumnsIndex.Ordinal 10706

DBXMetaDataNames.TDBXColumnsIndex.Precision 10706

DBXMetaDataNames.TDBXColumnsIndex.Scale 10706

DBXMetaDataNames.TDBXColumnsIndex.SchemaName 10706

DBXMetaDataNames.TDBXColumnsIndex.TableName 10707

DBXMetaDataNames.TDBXColumnsIndex.TypeName 10707

DBXMetaDataNames.TDBXDataTypesColumns 10707

DBXMetaDataNames.TDBXDataTypesColumns.ColumnSize
10710

DBXMetaDataNames.TDBXDataTypesColumns.CreateFormat
10710

DBXMetaDataNames.TDBXDataTypesColumns.CreateParameters
10711

DBXMetaDataNames.TDBXDataTypesColumns.DataType
10711

DBXMetaDataNames.TDBXDataTypesColumns.DbxDataType
10711

DBXMetaDataNames.TDBXDataTypesColumns.IsAutoIncrementable 10711
DBXMetaDataNames.TDBXDataTypesColumns.IsBestMatch 10711
DBXMetaDataNames.TDBXDataTypesColumns.IsCaseSensitive 10712
DBXMetaDataNames.TDBXDataTypesColumns.IsConcurrencyType 10712
DBXMetaDataNames.TDBXDataTypesColumns.IsFixedLength 10712
DBXMetaDataNames.TDBXDataTypesColumns.IsFixedPrecisionScale 10712
DBXMetaDataNames.TDBXDataTypesColumns.IsLiteralSupported 10713
DBXMetaDataNames.TDBXDataTypesColumns.IsLong 10713
DBXMetaDataNames.TDBXDataTypesColumns.IsNullable 10713
DBXMetaDataNames.TDBXDataTypesColumns.IsSearchable 10713
DBXMetaDataNames.TDBXDataTypesColumns.IsSearchableWithLike 10713
DBXMetaDataNames.TDBXDataTypesColumns.IsUnicode 10714
DBXMetaDataNames.TDBXDataTypesColumns.IsUnsigned 10714
DBXMetaDataNames.TDBXDataTypesColumns.LiteralPrefix 10714
DBXMetaDataNames.TDBXDataTypesColumns.LiteralSuffix 10714
DBXMetaDataNames.TDBXDataTypesColumns.MaximumScale 10715
DBXMetaDataNames.TDBXDataTypesColumns.MaximumVersion 10715
DBXMetaDataNames.TDBXDataTypesColumns.MinimumScale 10715
DBXMetaDataNames.TDBXDataTypesColumns.MinimumVersion 10715
DBXMetaDataNames.TDBXDataTypesColumns.ProviderDbType 10715
DBXMetaDataNames.TDBXDataTypesColumns.TypeName 10716
DBXMetaDataNames.TDBXDataTypesIndex 10716
DBXMetaDataNames.TDBXDataTypesIndex.ColumnSize 10719
DBXMetaDataNames.TDBXDataTypesIndex.CreateFormat 10719
DBXMetaDataNames.TDBXDataTypesIndex.CreateParameters 10719
DBXMetaDataNames.TDBXDataTypesIndex.DataType 10720
DBXMetaDataNames.TDBXDataTypesIndex.DbxDataType 10720
DBXMetaDataNames.TDBXDataTypesIndex.IsAutoIncrementable 10720
DBXMetaDataNames.TDBXDataTypesIndex.IsBestMatch 10720
DBXMetaDataNames.TDBXDataTypesIndex.IsCaseSensitive 10720
DBXMetaDataNames.TDBXDataTypesIndex.IsConcurrencyType 10721
DBXMetaDataNames.TDBXDataTypesIndex.IsFixedLength 10721
DBXMetaDataNames.TDBXDataTypesIndex.IsFixedPrecisionScale 10721
DBXMetaDataNames.TDBXDataTypesIndex.IsLiteralSupported 10721
DBXMetaDataNames.TDBXDataTypesIndex.IsLong 10722
DBXMetaDataNames.TDBXDataTypesIndex.IsNullable 10722
DBXMetaDataNames.TDBXDataTypesIndex.IsSearchable 10722
DBXMetaDataNames.TDBXDataTypesIndex.IsSearchableWithLike 10722
DBXMetaDataNames.TDBXDataTypesIndex.IsUnicode 10722
DBXMetaDataNames.TDBXDataTypesIndex.IsUnsigned 10723
DBXMetaDataNames.TDBXDataTypesIndex.Last 10723
DBXMetaDataNames.TDBXDataTypesIndex.LiteralPrefix 10723
DBXMetaDataNames.TDBXDataTypesIndex.LiteralSuffix 10723
DBXMetaDataNames.TDBXDataTypesIndex.MaximumScale 10724
DBXMetaDataNames.TDBXDataTypesIndex.MaximumVersion 10724
DBXMetaDataNames.TDBXDataTypesIndex.MinimumScale 10724
DBXMetaDataNames.TDBXDataTypesIndex.MinimumVersion 10724

DBXMetaDataNames.TDBXDataTypesIndex.ProviderDbType 10724
DBXMetaDataNames.TDBXDataTypesIndex.TypeName 10725
DBXMetaDataNames.TDBXForeignKeyColumnsColumns 10725
DBXMetaDataNames.TDBXForeignKeyColumnsColumns.CatalogName 10727
DBXMetaDataNames.TDBXForeignKeyColumnsColumns.ColumnName 10727
DBXMetaDataNames.TDBXForeignKeyColumnsColumns.ForeignKeyName 10727
DBXMetaDataNames.TDBXForeignKeyColumnsColumns.Ordinal 10727
DBXMetaDataNames.TDBXForeignKeyColumnsColumns.PrimaryCatalogName 10728
DBXMetaDataNames.TDBXForeignKeyColumnsColumns.PrimaryColumnName 10728
DBXMetaDataNames.TDBXForeignKeyColumnsColumns.PrimaryKeyName 10728
DBXMetaDataNames.TDBXForeignKeyColumnsColumns.PrimarySchemaName 10728
DBXMetaDataNames.TDBXForeignKeyColumnsColumns.PrimaryTableName 10729
DBXMetaDataNames.TDBXForeignKeyColumnsColumns.SchemaName 10729
DBXMetaDataNames.TDBXForeignKeyColumnsColumns.TableName 10729
DBXMetaDataNames.TDBXForeignKeyColumnsIndex 10729
DBXMetaDataNames.TDBXForeignKeyColumnsIndex.CatalogName 10731
DBXMetaDataNames.TDBXForeignKeyColumnsIndex.ColumnName 10731
DBXMetaDataNames.TDBXForeignKeyColumnsIndex.ForeignKeyName 10732
DBXMetaDataNames.TDBXForeignKeyColumnsIndex.Last 10732
DBXMetaDataNames.TDBXForeignKeyColumnsIndex.Ordinal 10732
DBXMetaDataNames.TDBXForeignKeyColumnsIndex.PrimaryCatalogName 10732
DBXMetaDataNames.TDBXForeignKeyColumnsIndex.PrimaryColumnName 10732
DBXMetaDataNames.TDBXForeignKeyColumnsIndex.PrimaryKeyName 10733
DBXMetaDataNames.TDBXForeignKeyColumnsIndex.PrimarySchemaName 10733
DBXMetaDataNames.TDBXForeignKeyColumnsIndex.PrimaryTableName 10733
DBXMetaDataNames.TDBXForeignKeyColumnsIndex.SchemaName 10733
DBXMetaDataNames.TDBXForeignKeyColumnsIndex.TableName 10734
DBXMetaDataNames.TDBXForeignKeysColumns 10734
DBXMetaDataNames.TDBXForeignKeysColumns.CatalogName 10735
DBXMetaDataNames.TDBXForeignKeysColumns.ForeignKeyName 10735
DBXMetaDataNames.TDBXForeignKeysColumns.SchemaName 10735
DBXMetaDataNames.TDBXForeignKeysColumns.TableName 10736
DBXMetaDataNames.TDBXForeignKeysIndex 10736
DBXMetaDataNames.TDBXForeignKeysIndex.CatalogName 10737
DBXMetaDataNames.TDBXForeignKeysIndex.ForeignKeyName 10737
DBXMetaDataNames.TDBXForeignKeysIndex.Last 10737
DBXMetaDataNames.TDBXForeignKeysIndex.SchemaName 10738
DBXMetaDataNames.TDBXForeignKeysIndex.TableName 10738
DBXMetaDataNames.TDBXIndexColumnsColumns 10738
DBXMetaDataNames.TDBXIndexColumnsColumns.CatalogName 10740

DBXMetaDataNames.TDBXIndexColumnsColumns.ColumnName 10740
DBXMetaDataNames.TDBXIndexColumnsColumns.IndexName 10740
DBXMetaDataNames.TDBXIndexColumnsColumns.IsAscending 10740
DBXMetaDataNames.TDBXIndexColumnsColumns.Ordinal 10740
DBXMetaDataNames.TDBXIndexColumnsColumns.SchemaName 10741
DBXMetaDataNames.TDBXIndexColumnsColumns.TableName 10741
DBXMetaDataNames.TDBXIndexColumnsIndex 10741
DBXMetaDataNames.TDBXIndexColumnsIndex.CatalogName 10743
DBXMetaDataNames.TDBXIndexColumnsIndex.ColumnName 10743
DBXMetaDataNames.TDBXIndexColumnsIndex.IndexName 10743
DBXMetaDataNames.TDBXIndexColumnsIndex.IsAscending 10743
DBXMetaDataNames.TDBXIndexColumnsIndex.Last 10743
DBXMetaDataNames.TDBXIndexColumnsIndex.Ordinal 10744
DBXMetaDataNames.TDBXIndexColumnsIndex.SchemaName 10744
DBXMetaDataNames.TDBXIndexColumnsIndex.TableName 10744
DBXMetaDataNames.TDBXIndexesColumns 10744
DBXMetaDataNames.TDBXIndexesColumns.CatalogName 10746
DBXMetaDataNames.TDBXIndexesColumns.ConstraintName 10746
DBXMetaDataNames.TDBXIndexesColumns.IndexName 10746
DBXMetaDataNames.TDBXIndexesColumns.IsAscending 10746
DBXMetaDataNames.TDBXIndexesColumns.IsPrimary 10747
DBXMetaDataNames.TDBXIndexesColumns.IsUnique 10747
DBXMetaDataNames.TDBXIndexesColumns.SchemaName 10747
DBXMetaDataNames.TDBXIndexesColumns.TableName 10747
DBXMetaDataNames.TDBXIndexesIndex 10748
DBXMetaDataNames.TDBXIndexesIndex.CatalogName 10749
DBXMetaDataNames.TDBXIndexesIndex.ConstraintName 10749
DBXMetaDataNames.TDBXIndexesIndex.IndexName 10749
DBXMetaDataNames.TDBXIndexesIndex.IsAscending 10750
DBXMetaDataNames.TDBXIndexesIndex.IsPrimary 10750
DBXMetaDataNames.TDBXIndexesIndex.IsUnique 10750
DBXMetaDataNames.TDBXIndexesIndex.Last 10750
DBXMetaDataNames.TDBXIndexesIndex.SchemaName 10751
DBXMetaDataNames.TDBXIndexesIndex.TableName 10751
DBXMetaDataNames.TDBXMetaDataCollectionIndex 10751
DBXMetaDataNames.TDBXMetaDataCollectionIndex.Catalogs 10753
DBXMetaDataNames.TDBXMetaDataCollectionIndex.ColumnConstraints 10754
DBXMetaDataNames.TDBXMetaDataCollectionIndex.Columns 10754
DBXMetaDataNames.TDBXMetaDataCollectionIndex.DataTypes 10754
DBXMetaDataNames.TDBXMetaDataCollectionIndex.ForeignKeyColumns 10754
DBXMetaDataNames.TDBXMetaDataCollectionIndex.ForeignKeys 10755
DBXMetaDataNames.TDBXMetaDataCollectionIndex.IndexColumns 10755
DBXMetaDataNames.TDBXMetaDataCollectionIndex.Indexes 10755
DBXMetaDataNames.TDBXMetaDataCollectionIndex.PackageProcedureParameters 10755
DBXMetaDataNames.TDBXMetaDataCollectionIndex.PackageProcedures 10755
DBXMetaDataNames.TDBXMetaDataCollectionIndex.Packages 10756
DBXMetaDataNames.TDBXMetaDataCollectionIndex.PackageSources 10756
DBXMetaDataNames.TDBXMetaDataCollectionIndex.ProcedureParameters 10756
DBXMetaDataNames.TDBXMetaDataCollectionIndex.Procedures 10756
DBXMetaDataNames.TDBXMetaDataCollectionIndex.Procedure

Sources
10757

DBXMetaDataNames.TDBXMetaDataCollectionIndex.ReservedWords
10757

DBXMetaDataNames.TDBXMetaDataCollectionIndex.Roles
10757

DBXMetaDataNames.TDBXMetaDataCollectionIndex.Schemas
10757

DBXMetaDataNames.TDBXMetaDataCollectionIndex.Synonyms

10757

DBXMetaDataNames.TDBXMetaDataCollectionIndex.Tables
10758

DBXMetaDataNames.TDBXMetaDataCollectionIndex.Users
10758

DBXMetaDataNames.TDBXMetaDataCollectionIndex.Views
10758

DBXMetaDataNames.TDBXMetaDataCollectionName 10758

DBXMetaDataNames.TDBXMetaDataCollectionName.Catalogs
10761

DBXMetaDataNames.TDBXMetaDataCollectionName.ColumnConstraints
10761

DBXMetaDataNames.TDBXMetaDataCollectionName.Columns
10761

DBXMetaDataNames.TDBXMetaDataCollectionName.DataTypes
10762

DBXMetaDataNames.TDBXMetaDataCollectionName.ForeignKeyColumns
10762

DBXMetaDataNames.TDBXMetaDataCollectionName.ForeignKeys
10762

DBXMetaDataNames.TDBXMetaDataCollectionName.IndexColumns
10762

DBXMetaDataNames.TDBXMetaDataCollectionName.Indexes
10762

DBXMetaDataNames.TDBXMetaDataCollectionName.PackageProcedureParameters
10763

DBXMetaDataNames.TDBXMetaDataCollectionName.PackageProcedures
10763

DBXMetaDataNames.TDBXMetaDataCollectionName.Packages

10763

DBXMetaDataNames.TDBXMetaDataCollectionName.Package

Sources
10763

DBXMetaDataNames.TDBXMetaDataCollectionName.ProcedureParameters
10764

DBXMetaDataNames.TDBXMetaDataCollectionName.Procedures
10764

DBXMetaDataNames.TDBXMetaDataCollectionName.ProcedureSources
10764

DBXMetaDataNames.TDBXMetaDataCollectionName.ReservedWords
10764

DBXMetaDataNames.TDBXMetaDataCollectionName.Roles
10764

DBXMetaDataNames.TDBXMetaDataCollectionName.Schemas
10765

DBXMetaDataNames.TDBXMetaDataCollectionName.Synonyms
10765

DBXMetaDataNames.TDBXMetaDataCollectionName.Tables
10765

DBXMetaDataNames.TDBXMetaDataCollectionName.Users
10765

DBXMetaDataNames.TDBXMetaDataCollectionName.Views
10766

DBXMetaDataNames.TDBXPackageProcedureParametersColumns
10766

DBXMetaDataNames.TDBXPackageProcedureParametersColumns.CatalogName
10768

DBXMetaDataNames.TDBXPackageProcedureParametersColumns.DbxDataType
10769

DBXMetaDataNames.TDBXPackageProcedureParametersColumns.IsFixedLength
10769

DBXMetaDataNames.TDBXPackageProcedureParametersColumns.IsLong
10769

DBXMetaDataNames.TDBXPackageProcedureParametersColumns.IsNullable
10769

DBXMetaDataNames.TDBXPackageProcedureParametersColumns.IsUnicode
10770

DBXMetaDataNames.TDBXPackageProcedureParametersColumns.IsUnsigned
10770

DBXMetaDataNames.TDBXPackageProcedureParametersColumns.Ordinal
10770

DBXMetaDataNames.TDBXPackageProcedureParametersColumns.PackageName
10770

DBXMetaDataNames.TDBXPackageProcedureParametersColumns.ParameterMode
10770

DBXMetaDataNames.TDBXPackageProcedureParametersColumns.ParameterName
10771

DBXMetaDataNames.TDBXPackageProcedureParametersColumns.Precision
10771

DBXMetaDataNames.TDBXPackageProcedureParametersColumns.ProcedureName
10771

DBXMetaDataNames.TDBXPackageProcedureParametersColumns.Scale
10771

DBXMetaDataNames.TDBXPackageProcedureParametersColumns.SchemaName
10772

DBXMetaDataNames.TDBXPackageProcedureParametersColumns.TypeName
10772

DBXMetaDataNames.TDBXPackageProcedureParametersIndex
10772

DBXMetaDataNames.TDBXPackageProcedureParametersIndex.CatalogName
10775

DBXMetaDataNames.TDBXPackageProcedureParametersIndex.DbxDataType
10775

DBXMetaDataNames.TDBXPackageProcedureParametersIndex.IsFixedLength
10775

DBXMetaDataNames.TDBXPackageProcedureParametersIndex.IsLong
10775

DBXMetaDataNames.TDBXPackageProcedureParametersIndex.IsNullable
10776

DBXMetaDataNames.TDBXPackageProcedureParametersIndex.IsUnicode
10776

DBXMetaDataNames.TDBXPackageProcedureParametersIndex.IsUnsigned
10776

DBXMetaDataNames.TDBXPackageProcedureParametersIndex.Last
10776

DBXMetaDataNames.TDBXPackageProcedureParametersIndex.Ordinal
10776

DBXMetaDataNames.TDBXPackageProcedureParametersIndex.PackageName
10777

DBXMetaDataNames.TDBXPackageProcedureParametersIndex.ParameterMode
10777

DBXMetaDataNames.TDBXPackageProcedureParametersIndex.ParameterName
10777

DBXMetaDataNames.TDBXPackageProcedureParametersIndex.Precision
10777

DBXMetaDataNames.TDBXPackageProcedureParametersIndex.ProcedureName
10778

DBXMetaDataNames.TDBXPackageProcedureParametersIndex.Scale
10778

DBXMetaDataNames.TDBXPackageProcedureParametersIndex.SchemaName
10778

DBXMetaDataNames.TDBXPackageProcedureParametersIndex.TypeName
10778

DBXMetaDataNames.TDBXPackageProceduresColumns 10778

DBXMetaDataNames.TDBXPackageProceduresColumns.CatalogName
10780

DBXMetaDataNames.TDBXPackageProceduresColumns.PackageName
10780

DBXMetaDataNames.TDBXPackageProceduresColumns.ProcedureName
10780

DBXMetaDataNames.TDBXPackageProceduresColumns.ProcedureType
10780

DBXMetaDataNames.TDBXPackageProceduresColumns.SchemaName
10781

DBXMetaDataNames.TDBXPackageProceduresIndex 10781

DBXMetaDataNames.TDBXPackageProceduresIndex.CatalogName
10782

DBXMetaDataNames.TDBXPackageProceduresIndex.Last

10782

DBXMetaDataNames.TDBXPackageProceduresIndex.PackageName
10783

DBXMetaDataNames.TDBXPackageProceduresIndex.ProcedureName
10783

DBXMetaDataNames.TDBXPackageProceduresIndex.ProcedureType
10783

DBXMetaDataNames.TDBXPackageProceduresIndex.SchemaName
10783

DBXMetaDataNames.TDBXPackagesColumns 10783

DBXMetaDataNames.TDBXPackagesColumns.CatalogName
10785

DBXMetaDataNames.TDBXPackagesColumns.PackageName
10785

DBXMetaDataNames.TDBXPackagesColumns.SchemaName
10785

DBXMetaDataNames.TDBXPackagesIndex 10785

DBXMetaDataNames.TDBXPackagesIndex.CatalogName
10786

DBXMetaDataNames.TDBXPackagesIndex.Last 10787

DBXMetaDataNames.TDBXPackagesIndex.PackageName
10787

DBXMetaDataNames.TDBXPackagesIndex.SchemaName
10787

DBXMetaDataNames.TDBXPackageSourcesColumns 10787

DBXMetaDataNames.TDBXPackageSourcesColumns.CatalogName
10788

DBXMetaDataNames.TDBXPackageSourcesColumns.Definition

10789

DBXMetaDataNames.TDBXPackageSourcesColumns.PackageName
10789

DBXMetaDataNames.TDBXPackageSourcesColumns.SchemaName
10789

DBXMetaDataNames.TDBXPackageSourcesIndex 10789

DBXMetaDataNames.TDBXPackageSourcesIndex.CatalogName
10790

DBXMetaDataNames.TDBXPackageSourcesIndex.Definition
10791

DBXMetaDataNames.TDBXPackageSourcesIndex.Last 10791

DBXMetaDataNames.TDBXPackageSourcesIndex.PackageName
10791

DBXMetaDataNames.TDBXPackageSourcesIndex.SchemaName
10791

DBXMetaDataNames.TDBXProcedureParametersColumns
10792

DBXMetaDataNames.TDBXProcedureParametersColumns.CatalogName
10794

DBXMetaDataNames.TDBXProcedureParametersColumns.DbxDataType
10794

DBXMetaDataNames.TDBXProcedureParametersColumns.IsFixedLength
10794

DBXMetaDataNames.TDBXProcedureParametersColumns.IsLong
10794

DBXMetaDataNames.TDBXProcedureParametersColumns.IsNullable
10795

DBXMetaDataNames.TDBXProcedureParametersColumns.IsUnicode
10795

DBXMetaDataNames.TDBXProcedureParametersColumns.IsUnsigned
10795

DBXMetaDataNames.TDBXProcedureParametersColumns.Ordinal
10795

DBXMetaDataNames.TDBXProcedureParametersColumns.ParameterMode
10796

DBXMetaDataNames.TDBXProcedureParametersColumns.ParameterName
10796

DBXMetaDataNames.TDBXProcedureParametersColumns.Precision
10796

DBXMetaDataNames.TDBXProcedureParametersColumns.ProcedureName
10796

DBXMetaDataNames.TDBXProcedureParametersColumns.Scale
10796

DBXMetaDataNames.TDBXProcedureParametersColumns.SchemaName
10797

DBXMetaDataNames.TDBXProcedureParametersColumns.Typ

eName
10797

DBXMetaDataNames.TDBXProcedureParametersIndex 10797

DBXMetaDataNames.TDBXProcedureParametersIndex.CatalogName
10799

DBXMetaDataNames.TDBXProcedureParametersIndex.DbxDataType
10800

DBXMetaDataNames.TDBXProcedureParametersIndex.IsFixedLength
10800

DBXMetaDataNames.TDBXProcedureParametersIndex.IsLong
10800

DBXMetaDataNames.TDBXProcedureParametersIndex.IsNullable
10800

DBXMetaDataNames.TDBXProcedureParametersIndex.IsUnicode
10800

DBXMetaDataNames.TDBXProcedureParametersIndex.IsUnsigned
10801

DBXMetaDataNames.TDBXProcedureParametersIndex.Last
10801

DBXMetaDataNames.TDBXProcedureParametersIndex.Ordinal
10801

DBXMetaDataNames.TDBXProcedureParametersIndex.ParameterMode
10801

DBXMetaDataNames.TDBXProcedureParametersIndex.ParameterName
10802

DBXMetaDataNames.TDBXProcedureParametersIndex.Precision
10802

DBXMetaDataNames.TDBXProcedureParametersIndex.ProcedureName
10802

DBXMetaDataNames.TDBXProcedureParametersIndex.Scale
10802

DBXMetaDataNames.TDBXProcedureParametersIndex.SchemaName
10802

DBXMetaDataNames.TDBXProcedureParametersIndex.TypeName
10803

DBXMetaDataNames.TDBXProceduresColumns 10803

DBXMetaDataNames.TDBXProceduresColumns.CatalogName
10804

DBXMetaDataNames.TDBXProceduresColumns.ProcedureName
10804

DBXMetaDataNames.TDBXProceduresColumns.ProcedureType
10804

DBXMetaDataNames.TDBXProceduresColumns.SchemaName
10805

DBXMetaDataNames.TDBXProceduresIndex 10805

DBXMetaDataNames.TDBXProceduresIndex.CatalogName
10806

DBXMetaDataNames.TDBXProceduresIndex.Last 10806

DBXMetaDataNames.TDBXProceduresIndex.ProcedureName
10807

DBXMetaDataNames.TDBXProceduresIndex.ProcedureType
10807

DBXMetaDataNames.TDBXProceduresIndex.SchemaName
10807

DBXMetaDataNames.TDBXProcedureSourcesColumns 10807

DBXMetaDataNames.TDBXProcedureSourcesColumns.CatalogName
10809

DBXMetaDataNames.TDBXProcedureSourcesColumns.Definition
10809

DBXMetaDataNames.TDBXProcedureSourcesColumns.ExternalDefinition
10809

DBXMetaDataNames.TDBXProcedureSourcesColumns.ProcedureName
10809

DBXMetaDataNames.TDBXProcedureSourcesColumns.ProcedureType
10809

DBXMetaDataNames.TDBXProcedureSourcesColumns.SchemaName
10810

DBXMetaDataNames.TDBXProcedureSourcesIndex 10810

DBXMetaDataNames.TDBXProcedureSourcesIndex.CatalogName
10811

DBXMetaDataNames.TDBXProcedureSourcesIndex.Definition
10812

DBXMetaDataNames.TDBXProcedureSourcesIndex.ExternalDefinition
10812

DBXMetaDataNames.TDBXProcedureSourcesIndex.Last 10812

DBXMetaDataNames.TDBXProcedureSourcesIndex.ProcedureName

10812

DBXMetaDataNames.TDBXProcedureSourcesIndex.ProcedureType
10812

DBXMetaDataNames.TDBXProcedureSourcesIndex.SchemaName
10813

DBXMetaDataNames.TDBXReservedWordsColumns 10813

DBXMetaDataNames.TDBXReservedWordsColumns.ReservedWord
10814

DBXMetaDataNames.TDBXReservedWordsIndex 10814

DBXMetaDataNames.TDBXReservedWordsIndex.Last 10815

DBXMetaDataNames.TDBXReservedWordsIndex.ReservedWord
10815

DBXMetaDataNames.TDBXRolesColumns 10816

DBXMetaDataNames.TDBXRolesColumns.RoleName 10817

DBXMetaDataNames.TDBXRolesIndex 10817

DBXMetaDataNames.TDBXRolesIndex.Last 10818

DBXMetaDataNames.TDBXRolesIndex.RoleName 10818

DBXMetaDataNames.TDBXSchemasColumns 10818

DBXMetaDataNames.TDBXSchemasColumns.CatalogName
10819

DBXMetaDataNames.TDBXSchemasColumns.SchemaName
10820

DBXMetaDataNames.TDBXSchemasIndex 10820

DBXMetaDataNames.TDBXSchemasIndex.CatalogName 10821

DBXMetaDataNames.TDBXSchemasIndex.Last 10821

DBXMetaDataNames.TDBXSchemasIndex.SchemaName
10821

DBXMetaDataNames.TDBXSynonymsColumns 10822

DBXMetaDataNames.TDBXSynonymsColumns.CatalogName
10823

DBXMetaDataNames.TDBXSynonymsColumns.SchemaName
10823

DBXMetaDataNames.TDBXSynonymsColumns.SynonymName
10823

DBXMetaDataNames.TDBXSynonymsColumns.TableCatalogName
10824

DBXMetaDataNames.TDBXSynonymsColumns.TableName
10824

DBXMetaDataNames.TDBXSynonymsColumns.TableSchemaName
10824

DBXMetaDataNames.TDBXSynonymsIndex 10824

DBXMetaDataNames.TDBXSynonymsIndex.CatalogName
10826

DBXMetaDataNames.TDBXSynonymsIndex.Last 10826

DBXMetaDataNames.TDBXSynonymsIndex.SchemaName
10826

DBXMetaDataNames.TDBXSynonymsIndex.SynonymName
10826

DBXMetaDataNames.TDBXSynonymsIndex.TableCatalogName
10827

DBXMetaDataNames.TDBXSynonymsIndex.TableName 10827

DBXMetaDataNames.TDBXSynonymsIndex.TableSchemaName
10827

DBXMetaDataNames.TDBXTablesColumns 10827

DBXMetaDataNames.TDBXTablesColumns.CatalogName
10828

DBXMetaDataNames.TDBXTablesColumns.SchemaName
10829

DBXMetaDataNames.TDBXTablesColumns.TableName 10829

DBXMetaDataNames.TDBXTablesColumns.TableType 10829

DBXMetaDataNames.TDBXTablesIndex 10829

DBXMetaDataNames.TDBXTablesIndex.CatalogName 10831

DBXMetaDataNames.TDBXTablesIndex.Last 10831

DBXMetaDataNames.TDBXTablesIndex.SchemaName 10831

DBXMetaDataNames.TDBXTablesIndex.TableName 10831

DBXMetaDataNames.TDBXTablesIndex.TableType 10831

DBXMetaDataNames.TDBXUsersColumns 10832

DBXMetaDataNames.TDBXUsersColumns.UserName 10833

DBXMetaDataNames.TDBXUsersIndex 10833

DBXMetaDataNames.TDBXUsersIndex.Last 10834

DBXMetaDataNames.TDBXUsersIndex.UserName 10834

DBXMetaDataNames.TDBXViewsColumns 10834

DBXMetaDataNames.TDBXViewsColumns.CatalogName 10835

DBXMetaDataNames.TDBXViewsColumns.Definition 10836

DBXMetaDataNames.TDBXViewsColumns.SchemaName
10836

DBXMetaDataNames.TDBXViewsColumns.ViewName 10836

DBXMetaDataNames.TDBXViewsIndex 10836

DBXMetaDataNames.TDBXViewsIndex.CatalogName 10838

DBXMetaDataNames.TDBXViewsIndex.Definition 10838

DBXMetaDataNames.TDBXViewsIndex.Last 10838

DBXMetaDataNames.TDBXViewsIndex.SchemaName 10838
DBXMetaDataNames.TDBXViewsIndex.ViewName 10838
DBXpressWeb 10841
DBXpressWeb namespace 10841
Classes 10842
DBXpressWeb.TSQLQueryTableProducer 10842
DBXpressWeb.TSQLQueryTableProducer.Caption 10849
DBXpressWeb.TSQLQueryTableProducer.CaptionAlignment 10849
DBXpressWeb.TSQLQueryTableProducer.Columns 10850
DBXpressWeb.TSQLQueryTableProducer.Content 10847
DBXpressWeb.TSQLQueryTableProducer.Footer 10850
DBXpressWeb.TSQLQueryTableProducer.GetDataSet 10847
DBXpressWeb.TSQLQueryTableProducer.Header 10851
DBXpressWeb.TSQLQueryTableProducer.MaxRows 10851
DBXpressWeb.TSQLQueryTableProducer.Notification 10848
DBXpressWeb.TSQLQueryTableProducer.OnCreateContent 10853
DBXpressWeb.TSQLQueryTableProducer.OnFormatCell 10854
DBXpressWeb.TSQLQueryTableProducer.OnGetTableCaption 10854
DBXpressWeb.TSQLQueryTableProducer.Query 10852
DBXpressWeb.TSQLQueryTableProducer.RowAttributes 10852
DBXpressWeb.TSQLQueryTableProducer.SetDataSet 10848
DBXpressWeb.TSQLQueryTableProducer.TableAttributes 10853
dcBrowseAllData enumeration member 17839
dcBrowseMemberData enumeration member 17839
dcBrowseMetaData enumeration member 17839
dcChangeCurrentRecordView enumeration member 19789
dcCurrentRecordView enumeration member 19789
dcInactive enumeration member 17839
dcMultipleRecordView enumeration member 19789
dcoHorizontal enumeration member 2409
dcoVertical enumeration member 2409
dcpbDown enumeration member 2409
dcpbNone enumeration member 2409
dcpbUp enumeration member 2409
dcrDefault enumeration member 7610
dcrHourGlass enumeration member 7610
dcrOther enumeration member 7610
dcrSQLWait enumeration member 7610
ddeAutomatic enumeration member 10911
DdeMan 10857
DdeMan namespace 10857
Classes 10858
Functions 10910
Structs, Records, Enums 10911
Types 10912
Variables 10913
DdeMan.ddeMgr 10913
DdeMan.ddeMgr variable 10913
DdeMan.GetPasteLinkInfo 10910
DdeMan.GetPasteLinkInfo function 10910
DdeMan.TDataMode 10911
DdeMan.TDataMode enumeration 10911
DdeMan.TDdeClientConv 10858
DdeMan.TDdeClientConv.ChangeLink 10863
DdeMan.TDdeClientConv.ClearItems 10864
DdeMan.TDdeClientConv.Close 10864
DdeMan.TDdeClientConv.CloseLink 10864
DdeMan.TDdeClientConv.ConnectMode 10873
DdeMan.TDdeClientConv.Conv 10874
DdeMan.TDdeClientConv.Create 10862
DdeMan.TDdeClientConv.CreateDdeConv 10864
DdeMan.TDdeClientConv.DataMode 10874
DdeMan.TDdeClientConv.DdeFmt 10874
DdeMan.TDdeClientConv.DdeService 10875
DdeMan.TDdeClientConv.DdeTopic 10875
DdeMan.TDdeClientConv.DefineProperties 10865
DdeMan.TDdeClientConv.Destroy 10862
DdeMan.TDdeClientConv.ExecuteMacro 10865
DdeMan.TDdeClientConv.ExecuteMacroLines 10866
DdeMan.TDdeClientConv.FormatChars 10876
DdeMan.TDdeClientConv.GetCliItemByCtrl 10866
DdeMan.TDdeClientConv.GetCliItemByName 10867
DdeMan.TDdeClientConv.Loaded 10867
DdeMan.TDdeClientConv.Notification 10867
DdeMan.TDdeClientConv.OnAttach 10868
DdeMan.TDdeClientConv.OnClose 10877

DdeMan.TDdeClientConv.OnDetach 10868
DdeMan.TDdeClientConv.OnOpen 10878
DdeMan.TDdeClientConv.OnSetItem 10868
DdeMan.TDdeClientConv.Open 10869
DdeMan.TDdeClientConv.OpenLink 10869
DdeMan.TDdeClientConv.PasteLink 10869
DdeMan.TDdeClientConv.PokeData 10870
DdeMan.TDdeClientConv.PokeDataLines 10870
DdeMan.TDdeClientConv.ReadLinkInfo 10871
DdeMan.TDdeClientConv.RequestData 10871
DdeMan.TDdeClientConv.ServiceApplication 10876
DdeMan.TDdeClientConv.SetLink 10872
DdeMan.TDdeClientConv.StartAdvise 10872
DdeMan.TDdeClientConv.WaitStat 10877
DdeMan.TDdeClientConv.WriteLinkInfo 10873
DdeMan.TDdeClientItem 10878
DdeMan.TDdeClientItem.Create 10881
DdeMan.TDdeClientItem.DdeConv 10883
DdeMan.TDdeClientItem.DdeItem 10883
DdeMan.TDdeClientItem.Destroy 10882
DdeMan.TDdeClientItem.Lines 10884
DdeMan.TDdeClientItem.Notification 10882
DdeMan.TDdeClientItem.OnChange 10885
DdeMan.TDdeClientItem.Text 10884
DdeMan.TDdeMgr 10886
DdeMan.TDdeMgr.AppName 10891
DdeMan.TDdeMgr.Create 10889
DdeMan.TDdeMgr.DdeInstId 10891
DdeMan.TDdeMgr.Destroy 10889
DdeMan.TDdeMgr.GetExeName 10890
DdeMan.TDdeMgr.LinkClipFmt 10891
DdeMan.TDdeServerConv 10891
DdeMan.TDdeServerConv.Connect 10896
DdeMan.TDdeServerConv.Create 10895
DdeMan.TDdeServerConv.Destroy 10895
DdeMan.TDdeServerConv.Disconnect 10896
DdeMan.TDdeServerConv.ExecuteMacro 10896
DdeMan.TDdeServerConv.OnClose 10897
DdeMan.TDdeServerConv.OnExecuteMacro 10897
DdeMan.TDdeServerConv.OnOpen 10898
DdeMan.TDdeServerItem 10898
DdeMan.TDdeServerItem.Change 10903
DdeMan.TDdeServerItem.CopyToClipboard 10903
DdeMan.TDdeServerItem.Create 10901
DdeMan.TDdeServerItem.Destroy 10902
DdeMan.TDdeServerItem.Fmt 10906
DdeMan.TDdeServerItem.GetText 10904
DdeMan.TDdeServerItem.Lines 10906
DdeMan.TDdeServerItem.Notification 10904
DdeMan.TDdeServerItem.OnChange 10908
DdeMan.TDdeServerItem.OnPokeData 10908
DdeMan.TDdeServerItem.PokeData 10904
DdeMan.TDdeServerItem.ServerConv 10907
DdeMan.TDdeServerItem.SetLines 10905
DdeMan.TDdeServerItem.SetServerConv 10905
DdeMan.TDdeServerItem.SetText 10905
DdeMan.TDdeServerItem.Text 10907
DdeMan.TMacroEvent 10912
DdeMan.TMacroEvent type 10912
ddeManual enumeration member 10911
deCheckBrowseMode enumeration member 7607
deConnectChange enumeration member 7607
deDataSetChange enumeration member 7607
deDataSetScroll enumeration member 7607
deDisabledStateChange enumeration member 7607
deFieldChange enumeration member 7607
deFieldListChange enumeration member 7607
deFocusControl enumeration member 7607
deLayoutChange enumeration member 7607
DelphiInterface template 22167
! 22170
& 22170
* 22170
~DelphiInterface 22168
= 22171
= operator 22170
-> 22171
about DelphiInterface template 22167

DelphiInterface 22168
DelphiInterface constructor 22168
DelphiInterface Members 22167
DelphiInterface methods 22169
DelphiInterface operators 22169
Release 22169
T* 22171
deParentScroll enumeration member 7607
dePropertyChange enumeration member 7607
deReconcileError enumeration member 7607
deRecordChange enumeration member 7607
DesignEditors 10915
DesignEditors namespace 10915
Classes 10917
Constants 11099
Functions 11090
Interfaces 11088
Structs, Records, Enums 11095
Types 11096
Variables 11098
DesignEditors.AncestorNameMatches 11090
DesignEditors.AncestorNameMatches function 11090
DesignEditors.ClassInheritsFrom 11090
DesignEditors.ClassInheritsFrom function 11090
DesignEditors.FreeEditorGroup 11091
DesignEditors.FreeEditorGroup function 11091
DesignEditors.GetComponentEditor 11091
DesignEditors.GetComponentEditor function 11091
DesignEditors.GetComponentProperties 11091
DesignEditors.GetComponentProperties function 11091
DesignEditors.GetSelectionEditors 11092
DesignEditors.GetSelectionEditors function 11092
DesignEditors.GetTopLevelComponentFunc 11098
DesignEditors.GetTopLevelComponentFunc variable 11098
DesignEditors.GuidelinesFor 11093
DesignEditors.GuidelinesFor function 11093
DesignEditors.IDesignGroupChange 11088
DesignEditors.IDesignGroupChange.FreeEditorGroup 11089
DesignEditors.NewEditorGroup 11093
DesignEditors.NewEditorGroup function 11093
DesignEditors.NotifyGroupChange 11093
DesignEditors.NotifyGroupChange function 11093
DesignEditors.PInstPropList 11096
DesignEditors.PInstPropList type 11096
DesignEditors.PossibleStream 11094
DesignEditors.PossibleStream function 11094
DesignEditors.sClassNotApplicable 11099
DesignEditors.sClassNotApplicable constant 11099
DesignEditors.sNotAvailable 11099
DesignEditors.sNotAvailable constant 11099
DesignEditors.TBit 11096
DesignEditors.TBit type 11096
DesignEditors.TBoolProperty 10918
DesignEditors.TCharProperty 10918
DesignEditors.TCharProperty.GetValue 10921
DesignEditors.TCharProperty.SetValue 10922
DesignEditors.TClassProperty 10922
DesignEditors.TClassProperty.GetAttributes 10926
DesignEditors.TClassProperty.GetProperties 10926
DesignEditors.TClassProperty.GetValue 10927
DesignEditors.TComponentEditor 10927
DesignEditors.TComponentEditor.Component 10934
DesignEditors.TComponentEditor.Copy 10930
DesignEditors.TComponentEditor.Create 10929
DesignEditors.TComponentEditor.Designer 10935
DesignEditors.TComponentEditor.Edit 10930
DesignEditors.TComponentEditor.ExecuteVerb 10931
DesignEditors.TComponentEditor.GetComponent 10931
DesignEditors.TComponentEditor.GetDesigner 10932
DesignEditors.TComponentEditor.GetVerb 10932
DesignEditors.TComponentEditor.GetVerbCount 10933
DesignEditors.TComponentEditor.IsInInlined 10933
DesignEditors.TComponentEditor.PrepareItem 10933
DesignEditors.TComponentGuidelines 10935
DesignEditors.TComponentGuidelines.CalcHorzPos 10937
DesignEditors.TComponentGuidelines.CalcVertPos 10938
DesignEditors.TComponentGuidelines.Component 10939
DesignEditors.TComponentGuidelines.Container 10939

DesignEditors.TComponentGuidelines.GetCount 10938
DesignEditors.TComponentGuidelines.GetDesignerGuideOffset 10938
DesignEditors.TComponentGuidelines.GetDesignerGuideType 10938
DesignEditors.TComponentGuidelines.Initialize 10938
DesignEditors.TComponentNameProperty 10939
DesignEditors.TComponentNameProperty.GetAttributes 10943
DesignEditors.TComponentNameProperty.GetEditLimit 10943
DesignEditors.TComponentProperty 10944
DesignEditors.TComponentProperty.AllEqual 10947
DesignEditors.TComponentProperty.Edit 10948
DesignEditors.TComponentProperty.FilterFunc 10948
DesignEditors.TComponentProperty.GetAttributes 10948
DesignEditors.TComponentProperty.GetComponentReference 10949
DesignEditors.TComponentProperty.GetEditLimit 10949
DesignEditors.TComponentProperty.GetProperties 10949
DesignEditors.TComponentProperty.GetSelections 10950
DesignEditors.TComponentProperty.GetValue 10950
DesignEditors.TComponentProperty.GetValues 10951
DesignEditors.TComponentProperty.SetValue 10951
DesignEditors.TComponentProperty.ShowReferenceProperty 10951
DesignEditors.TCustomModule 10952
DesignEditors.TCustomModule.Create 10953
DesignEditors.TCustomModule.Designer 10957
DesignEditors.TCustomModule.Destroy 10954
DesignEditors.TCustomModule.ExecuteVerb 10955
DesignEditors.TCustomModule.GetAttributes 10955
DesignEditors.TCustomModule.GetVerb 10955
DesignEditors.TCustomModule.GetVerbCount 10955
DesignEditors.TCustomModule.Nestable 10955
DesignEditors.TCustomModule.PrepareItem 10956
DesignEditors.TCustomModule.Root 10957
DesignEditors.TCustomModule.Saving 10956
DesignEditors.TCustomModule.ValidateComponent 10956
DesignEditors.TCustomModule.ValidateComponentClass 10956
DesignEditors.TDateProperty 10957
DesignEditors.TDateProperty.GetAttributes 10960
DesignEditors.TDateProperty.GetValue 10961
DesignEditors.TDateProperty.SetValue 10962
DesignEditors.TDateTimeProperty 10962
DesignEditors.TDateTimeProperty.GetAttributes 10965
DesignEditors.TDateTimeProperty.GetValue 10966
DesignEditors.TDateTimeProperty.SetValue 10966
DesignEditors.TDefaultEditor 10967
DesignEditors.TDefaultEditor.Edit 10969
DesignEditors.TDefaultEditor.EditProperty 10970
DesignEditors.TDesignerOleDropTarget 10970
DesignEditors.TDesignerOleDropTarget.Create 10972
DesignEditors.TDesignerOleDropTarget.DragEnter 10972
DesignEditors.TDesignerOleDropTarget.DragLeave 10972
DesignEditors.TDesignerOleDropTarget.DragOver 10973
DesignEditors.TDesignerOleDropTarget.Drop 10973
DesignEditors.TEditActionSelectionEditor 10973
DesignEditors.TEditActionSelectionEditor.EditAction 10975
DesignEditors.TEditActionSelectionEditor.GetEditState 10976
DesignEditors.TEditActionSelectionEditor.GetVerb 10976
DesignEditors.TEditActionSelectionEditor.GetVerbCount 10976
DesignEditors.TEditActionSelectionEditor.HandleCopy 10976
DesignEditors.TEditActionSelectionEditor.HandleCut 10977
DesignEditors.TEditActionSelectionEditor.HandleDelete 10977
DesignEditors.TEditActionSelectionEditor.HandlePaste 10977
DesignEditors.TEditActionSelectionEditor.HandleSelectAll 10977
DesignEditors.TEditActionSelectionEditor.HandleUndo 10977
DesignEditors.TEditActionSelectionEditor.PrepareItem 10978
DesignEditors.TEnumProperty 10978
DesignEditors.TEnumProperty.GetAttributes 10981
DesignEditors.TEnumProperty.GetValue 10982
DesignEditors.TEnumProperty.GetValues 10982
DesignEditors.TEnumProperty.SetValue 10983
DesignEditors.TFloatProperty 10983
DesignEditors.TFloatProperty.AllEqual 10987
DesignEditors.TFloatProperty.GetValue 10987
DesignEditors.TFloatProperty.SetValue 10988
DesignEditors.TGetTopLevelComponentFunc 11097
DesignEditors.TGetTopLevelComponentFunc type 11097
DesignEditors.TGroupChangeProc 11097
DesignEditors.TGroupChangeProc type 11097

DesignEditors.TInstProp 11095
DesignEditors.TInstProp record 11095
DesignEditors.TInstPropList 11097
DesignEditors.TInstPropList type 11097
DesignEditors.TInt64Property 10988
DesignEditors.TInt64Property.AllEqual 10991
DesignEditors.TInt64Property.GetEditLimit 10992
DesignEditors.TInt64Property.GetValue 10992
DesignEditors.TInt64Property.SetValue 10993
DesignEditors.TIntegerProperty 10993
DesignEditors.TIntegerProperty.GetValue 10996
DesignEditors.TIntegerProperty.SetValue 10997
DesignEditors.TInterfaceProperty 10998
DesignEditors.TInterfaceProperty.AllEqual 11001
DesignEditors.TInterfaceProperty.GetComponent 11002
DesignEditors.TInterfaceProperty.GetComponentReference 11002
DesignEditors.TInterfaceProperty.GetSelections 11002
DesignEditors.TInterfaceProperty.GetValues 11002
DesignEditors.TInterfaceProperty.ReceiveComponentNames 11003
DesignEditors.TInterfaceProperty.SetValue 11003
DesignEditors.TMethodProperty 11004
DesignEditors.TMethodProperty.AllEqual 11007
DesignEditors.TMethodProperty.AllNamed 11008
DesignEditors.TMethodProperty.Edit 11008
DesignEditors.TMethodProperty.GetAttributes 11008
DesignEditors.TMethodProperty.GetEditLimit 11009
DesignEditors.TMethodProperty.GetFormMethodName 11009
DesignEditors.TMethodProperty.GetTrimmedEventName 11009
DesignEditors.TMethodProperty.GetValue 11010
DesignEditors.TMethodProperty.GetValues 11010
DesignEditors.TMethodProperty.SetValue 11011
DesignEditors.TNestedProperty 11011
DesignEditors.TNestedProperty.Create 11014
DesignEditors.TNestedProperty.Destroy 11014
DesignEditors.TOrdinalProperty 11015
DesignEditors.TOrdinalProperty.AllEqual 11018
DesignEditors.TOrdinalProperty.GetEditLimit 11018
DesignEditors.TPropertyEditor 11019
DesignEditors.TPropertyEditor.Activate 11024
DesignEditors.TPropertyEditor.AllEqual 11024
DesignEditors.TPropertyEditor.AutoFill 11025
DesignEditors.TPropertyEditor.Create 11022
DesignEditors.TPropertyEditor.Designer 11044
DesignEditors.TPropertyEditor.Destroy 11022
DesignEditors.TPropertyEditor.Edit 11025
DesignEditors.TPropertyEditor.GetAttributes 11026
DesignEditors.TPropertyEditor.GetComponent 11026
DesignEditors.TPropertyEditor.GetEditLimit 11027
DesignEditors.TPropertyEditor.GetEditValue 11027
DesignEditors.TPropertyEditor.GetFloatValue 11028
DesignEditors.TPropertyEditor.GetFloatValueAt 11028
DesignEditors.TPropertyEditor.GetInt64Value 11028
DesignEditors.TPropertyEditor.GetInt64ValueAt 11029
DesignEditors.TPropertyEditor.GetIntfValue 11029
DesignEditors.TPropertyEditor.GetIntfValueAt 11030
DesignEditors.TPropertyEditor.GetIsDefault 11030
DesignEditors.TPropertyEditor.GetKind 11030
DesignEditors.TPropertyEditor.GetMethodValue 11031
DesignEditors.TPropertyEditor.GetMethodValueAt 11031
DesignEditors.TPropertyEditor.GetName 11032
DesignEditors.TPropertyEditor.GetOrdValue 11032
DesignEditors.TPropertyEditor.GetOrdValueAt 11032
DesignEditors.TPropertyEditor.GetProperties 11033
DesignEditors.TPropertyEditor.GetPropInfo 11033
DesignEditors.TPropertyEditor.GetPropType 11033
DesignEditors.TPropertyEditor.GetStrValue 11034
DesignEditors.TPropertyEditor.GetStrValueAt 11034
DesignEditors.TPropertyEditor.GetValue 11035
DesignEditors.TPropertyEditor.GetValues 11035
DesignEditors.TPropertyEditor.GetValueW 11036
DesignEditors.TPropertyEditor.GetVarValue 11036
DesignEditors.TPropertyEditor.GetVarValueAt 11036
DesignEditors.TPropertyEditor.GetVisualValue 11037
DesignEditors.TPropertyEditor.GetWideStrValue 11037
DesignEditors.TPropertyEditor.GetWideStrValueAt 11037
DesignEditors.TPropertyEditor.HasInstance 11038
DesignEditors.TPropertyEditor.Initialize 11038

DesignEditors.TPropertyEditor.Modified 11038
DesignEditors.TPropertyEditor.PrivateDirectory 11044
DesignEditors.TPropertyEditor.PropCount 11045
DesignEditors.TPropertyEditor.Revert 11039
DesignEditors.TPropertyEditor.SetFloatValue 11039
DesignEditors.TPropertyEditor.SetInt64Value 11040
DesignEditors.TPropertyEditor.SetIntfValue 11040
DesignEditors.TPropertyEditor.SetMethodValue 11040
DesignEditors.TPropertyEditor.SetOrdValue 11041
DesignEditors.TPropertyEditor.SetPropEntry 11041
DesignEditors.TPropertyEditor.SetStrValue 11042
DesignEditors.TPropertyEditor.SetValue 11042, 11043
DesignEditors.TPropertyEditor.SetVarValue 11043
DesignEditors.TPropertyEditor.SetWideStrValue 11043
DesignEditors.TPropertyEditor.Value 11045
DesignEditors.TPropertyEditor.ValueAvailable 11044
DesignEditors.TSelectionEditor 11046
DesignEditors.TSelectionEditor.Create 11047
DesignEditors.TSelectionEditor.Designer 11049
DesignEditors.TSelectionEditor.ExecuteVerb 11048
DesignEditors.TSelectionEditor.GetVerb 11048
DesignEditors.TSelectionEditor.GetVerbCount 11048
DesignEditors.TSelectionEditor.PrepareItem 11048
DesignEditors.TSelectionEditor.RequiresUnits 11049
DesignEditors.TSetElementProperty 11049
DesignEditors.TSetElementProperty.AllEqual 11053
DesignEditors.TSetElementProperty.Create 11052
DesignEditors.TSetElementProperty.Element 11056
DesignEditors.TSetElementProperty.GetAttributes 11053
DesignEditors.TSetElementProperty.GetIsDefault 11054
DesignEditors.TSetElementProperty.GetName 11054
DesignEditors.TSetElementProperty.GetValue 11055
DesignEditors.TSetElementProperty.GetValues 11055
DesignEditors.TSetElementProperty.SetValue 11056
DesignEditors.TSetProperty 11056
DesignEditors.TSetProperty.GetAttributes 11060
DesignEditors.TSetProperty.GetProperties 11060
DesignEditors.TSetProperty.GetValue 11061
DesignEditors.TStringProperty 11061
DesignEditors.TStringProperty.AllEqual 11065
DesignEditors.TStringProperty.GetEditLimit 11065
DesignEditors.TStringProperty.GetValue 11065
DesignEditors.TStringProperty.SetValue 11066
DesignEditors.TTimeProperty 11066
DesignEditors.TTimeProperty.GetAttributes 11070
DesignEditors.TTimeProperty.GetValue 11070
DesignEditors.TTimeProperty.SetValue 11071
DesignEditors.TVariantProperty 11071
DesignEditors.TVariantProperty.GetAttributes 11075
DesignEditors.TVariantProperty.GetProperties 11075
DesignEditors.TVariantProperty.GetValue 11076
DesignEditors.TVariantProperty.SetValue 11076
DesignEditors.TWideCharProperty 11077
DesignEditors.TWideCharProperty.GetValue 11080
DesignEditors.TWideCharProperty.GetValueW 11081
DesignEditors.TWideCharProperty.SetValue 11081
DesignEditors.TWideStringProperty 11082
DesignEditors.TWideStringProperty.AllEqual 11085
DesignEditors.TWideStringProperty.GetNameW 11086
DesignEditors.TWideStringProperty.GetValue 11086
DesignEditors.TWideStringProperty.GetValues 11086
DesignEditors.TWideStringProperty.GetValueW 11087
DesignEditors.TWideStringProperty.SetValue 11087
DesignEditors.UnnotifyGroupChange 11094
DesignEditors.UnnotifyGroupChange function 11094
DesignIntf 11101
DesignIntf namespace 11101
- Classes 11105
- Constants 11255
- Functions 11224
- Interfaces 11134
- Structs, Records, Enums 11233
- Types 11238
- Variables 11249

DesignIntf.CreateSelectionList 11224
DesignIntf.CreateSelectionList function 11224
DesignIntf.EnableDemandLoadReport 11225
DesignIntf.EnableDemandLoadReport function 11225

DesignIntf.EnableDemandLoadReportProc 11249
DesignIntf.EnableDemandLoadReportProc variable 11249
DesignIntf.ForceDemandLoadState 11225
DesignIntf.ForceDemandLoadState function 11225
DesignIntf.ForceDemandLoadStateProc 11249
DesignIntf.ForceDemandLoadStateProc variable 11249
DesignIntf.GReferenceExpandable 11250
DesignIntf.GReferenceExpandable variable 11250
DesignIntf.GShowReadOnlyProps 11250
DesignIntf.GShowReadOnlyProps variable 11250
DesignIntf.IActivatable 11134
DesignIntf.IActivatable.Activate 11135
DesignIntf.IClass 11135
DesignIntf.IClass.ClassName 11137
DesignIntf.IClass.ClassNameIs 11136
DesignIntf.IClass.ClassParent 11138
DesignIntf.IClass.GetClassName 11137
DesignIntf.IClass.GetClassParent 11137
DesignIntf.IClass.GetUnitName 11137
DesignIntf.IClass.UnitName 11138
DesignIntf.IComponentEditor 11138
DesignIntf.IComponentEditor.Copy 11140
DesignIntf.IComponentEditor.Edit 11140
DesignIntf.IComponentEditor.ExecuteVerb 11140
DesignIntf.IComponentEditor.GetComponent 11141
DesignIntf.IComponentEditor.GetDesigner 11141
DesignIntf.IComponentEditor.GetVerb 11141
DesignIntf.IComponentEditor.GetVerbCount 11142
DesignIntf.IComponentEditor.IsInInlined 11142
DesignIntf.IComponentEditor.PrepareItem 11142
DesignIntf.IComponentGuidelines 11143
DesignIntf.IComponentGuidelines.Count 11145
DesignIntf.IComponentGuidelines.DesignerGuideOffsets 11145
DesignIntf.IComponentGuidelines.DesignerGuideTypes 11145
DesignIntf.IComponentGuidelines.GetCount 11144
DesignIntf.IComponentGuidelines.GetDesignerGuideOffset 11144
DesignIntf.IComponentGuidelines.GetDesignerGuideType 11144
DesignIntf.ICustomModule 11145
DesignIntf.ICustomModule.ExecuteVerb 11146
DesignIntf.ICustomModule.GetAttributes 11147
DesignIntf.ICustomModule.GetVerb 11147
DesignIntf.ICustomModule.GetVerbCount 11147
DesignIntf.ICustomModule.Nestable 11147
DesignIntf.ICustomModule.PrepareItem 11147
DesignIntf.ICustomModule.Saving 11148
DesignIntf.ICustomModule.ValidateComponent 11148
DesignIntf.ICustomModule.ValidateComponentClass 11148
DesignIntf.IDefaultEditor 11148
DesignIntf.IDesignEditQuery 11238
DesignIntf.IDesignEditQuery type 11238
DesignIntf.IDesigner 11149
DesignIntf.IDesigner.DesignerExtention 11151
DesignIntf.IDesigner.GetDesignerExtension 11151
DesignIntf.IDesigner60 11151
DesignIntf.IDesigner60.Activate 11154
DesignIntf.IDesigner60.AncestorDesigner 11166
DesignIntf.IDesigner60.CanPaste 11154
DesignIntf.IDesigner60.ChainCall 11155
DesignIntf.IDesigner60.ClearSelection 11155
DesignIntf.IDesigner60.CopySelection 11155
DesignIntf.IDesigner60.CreateComponent 11156
DesignIntf.IDesigner60.CreateCurrentComponent 11156
DesignIntf.IDesigner60.CreateMethod 11156
DesignIntf.IDesigner60.CutSelection 11156
DesignIntf.IDesigner60.DeleteSelection 11157
DesignIntf.IDesigner60.Edit 11157
DesignIntf.IDesigner60.GetAncestorDesigner 11157
DesignIntf.IDesigner60.GetBaseRegKey 11157
DesignIntf.IDesigner60.GetComponent 11157
DesignIntf.IDesigner60.GetComponentName 11158
DesignIntf.IDesigner60.GetComponentNames 11158
DesignIntf.IDesigner60.GetIDEOptions 11158
DesignIntf.IDesigner60.GetIsDormant 11158
DesignIntf.IDesigner60.GetMethodName 11159
DesignIntf.IDesigner60.GetMethods 11159
DesignIntf.IDesigner60.GetObject 11159
DesignIntf.IDesigner60.GetObjectName 11159

DesignIntf.IDesigner60.GetObjectNames 11159
DesignIntf.IDesigner60.GetPathAndBaseExeName 11160
DesignIntf.IDesigner60.GetPrivateDirectory 11160
DesignIntf.IDesigner60.GetProjectModules 11160
DesignIntf.IDesigner60.GetRoot 11160
DesignIntf.IDesigner60.GetRootClassName 11161
DesignIntf.IDesigner60.GetScrollRanges 11161
DesignIntf.IDesigner60.GetSelections 11161
DesignIntf.IDesigner60.GetShiftState 11161
DesignIntf.IDesigner60.IsComponentHidden 11161
DesignIntf.IDesigner60.IsComponentLinkable 11162
DesignIntf.IDesigner60.IsDormant 11166
DesignIntf.IDesigner60.IsSourceReadOnly 11162
DesignIntf.IDesigner60.MakeComponentLinkable 11162
DesignIntf.IDesigner60.MethodExists 11162
DesignIntf.IDesigner60.MethodFromAncestor 11163
DesignIntf.IDesigner60.ModalEdit 11163
DesignIntf.IDesigner60.Modified 11163
DesignIntf.IDesigner60.ModuleFileNames 11163
DesignIntf.IDesigner60.NoSelection 11163
DesignIntf.IDesigner60.PasteSelection 11164
DesignIntf.IDesigner60.RenameMethod 11164
DesignIntf.IDesigner60.Resurrect 11164
DesignIntf.IDesigner60.Revert 11164
DesignIntf.IDesigner60.Root 11166
DesignIntf.IDesigner60.SelectComponent 11165
DesignIntf.IDesigner60.SelectItemName 11165
DesignIntf.IDesigner60.SetSelections 11165
DesignIntf.IDesigner60.ShowMethod 11165
DesignIntf.IDesigner60.UniqueName 11165
DesignIntf.IDesigner70 11167
DesignIntf.IDesigner70.ActiveClassGroup 11169
DesignIntf.IDesigner70.FindRootAncestor 11168
DesignIntf.IDesigner70.GetActiveClassGroup 11169
DesignIntf.IDesigner80 11169
DesignIntf.IDesigner80.CreateMethod 11171
DesignIntf.IDesigner80.GetMethods 11171
DesignIntf.IDesigner80.SelectComponent 11171
DesignIntf.IDesignerSelections 11172
DesignIntf.IDesignerSelections.Add 11173
DesignIntf.IDesignerSelections.Count 11175
DesignIntf.IDesignerSelections.DesignObjects 11175
DesignIntf.IDesignerSelections.Equals 11173
DesignIntf.IDesignerSelections.Get 11174
DesignIntf.IDesignerSelections.GetCount 11174
DesignIntf.IDesignerSelections.GetDesignObject 11174
DesignIntf.IDesignerSelections.Items 11175
DesignIntf.IDesignNotification 11176
DesignIntf.IDesignNotification.DesignerClosed 11177
DesignIntf.IDesignNotification.DesignerOpened 11177
DesignIntf.IDesignNotification.ItemDeleted 11177
DesignIntf.IDesignNotification.ItemInserted 11177
DesignIntf.IDesignNotification.ItemsModified 11178
DesignIntf.IDesignNotification.SelectionChanged 11178
DesignIntf.IDesignObject 11178
DesignIntf.IDesignObject.ClassName 11181
DesignIntf.IDesignObject.ClassType 11182
DesignIntf.IDesignObject.ComponentIndex 11182
DesignIntf.IDesignObject.ComponentName 11182
DesignIntf.IDesignObject.Equals 11179, 11180
DesignIntf.IDesignObject.GetClassName 11180
DesignIntf.IDesignObject.GetClassType 11180
DesignIntf.IDesignObject.GetComponentIndex 11180
DesignIntf.IDesignObject.GetComponentName 11180
DesignIntf.IDesignObject.GetIsComponent 11181
DesignIntf.IDesignObject.GetNamePath 11181
DesignIntf.IDesignObject.IsComponent 11182
DesignIntf.IDesignObject.NamePath 11182
DesignIntf.IDesignPersistent 11183
DesignIntf.IDesignPersistent.GetPersistent 11184
DesignIntf.IDesignPersistent.Persistent 11184
DesignIntf.IDesignWindow 11184
DesignIntf.IDesignWindow.WindowHide 11185
DesignIntf.IDesignWindow.WindowShow 11185
DesignIntf.IEditHandler 11186
DesignIntf.IEditHandler.EditAction 11186
DesignIntf.IEditHandler.GetEditState 11187
DesignIntf.IEventInfo 11187

DesignIntf.IEventInfo.GetMethodKind 11188
DesignIntf.IEventInfo.GetParamCount 11188
DesignIntf.IEventInfo.GetParamFlags 11188
DesignIntf.IEventInfo.GetParamName 11188
DesignIntf.IEventInfo.GetParamType 11189
DesignIntf.IEventInfo.GetResultType 11189
DesignIntf.IHostForm 11189
DesignIntf.IHostForm.BringToFront 11191
DesignIntf.IHostForm.Caption 11195
DesignIntf.IHostForm.CheckPosChanged 11191
DesignIntf.IHostForm.DeinitializeDesigner 11191
DesignIntf.IHostForm.GetCanPrint 11191
DesignIntf.IHostForm.GetCaption 11191
DesignIntf.IHostForm.GetDesignerState 11192
DesignIntf.IHostForm.GetFont 11192
DesignIntf.IHostForm.GetForm 11192
DesignIntf.IHostForm.GetFormImage 11192
DesignIntf.IHostForm.GetScrollPos 11193
DesignIntf.IHostForm.GetVisible 11193
DesignIntf.IHostForm.GetWindowState 11193
DesignIntf.IHostForm.HideWindow 11193
DesignIntf.IHostForm.IsMenuKey 11193
DesignIntf.IHostForm.SetCaption 11194
DesignIntf.IHostForm.SetDesigner 11194
DesignIntf.IHostForm.SetDesigning 11194
DesignIntf.IHostForm.SetFormDefaults 11194
DesignIntf.IHostForm.Show 11195
DesignIntf.IHostForm.ShowWindow 11195
DesignIntf.IHostForm.Unmodify 11195
DesignIntf.IHostForm.Visible 11196
DesignIntf.IImplementation 11196
DesignIntf.IImplementation.GetInstance 11197
DesignIntf.IMethodProperty 11197
DesignIntf.IProperty 11197
DesignIntf.IProperty.Activate 11199
DesignIntf.IProperty.AllEqual 11199
DesignIntf.IProperty.AutoFill 11200
DesignIntf.IProperty.Edit 11200
DesignIntf.IProperty.GetAttributes 11200
DesignIntf.IProperty.GetEditLimit 11201
DesignIntf.IProperty.GetEditValue 11201
DesignIntf.IProperty.GetName 11202
DesignIntf.IProperty.GetProperties 11202
DesignIntf.IProperty.GetPropInfo 11202
DesignIntf.IProperty.GetPropType 11203
DesignIntf.IProperty.GetValue 11203
DesignIntf.IProperty.GetValues 11203
DesignIntf.IProperty.HasInstance 11204
DesignIntf.IProperty.Revert 11204
DesignIntf.IProperty.SetValue 11204
DesignIntf.IProperty.ValueAvailable 11205
DesignIntf.IProperty70 11205
DesignIntf.IProperty70.GetIsDefault 11206
DesignIntf.IProperty70.IsDefault 11206
DesignIntf.IProperty80 11207
DesignIntf.IProperty80.Edit 11208
DesignIntf.IPropertyControl 11208
DesignIntf.IPropertyControl.ClearCache 11209
DesignIntf.IPropertyDescription 11209
DesignIntf.IPropertyDescription.GetDescription 11210
DesignIntf.IPropertyHost 11210
DesignIntf.IPropertyHost.CloseDropDown 11211
DesignIntf.IPropertyHost.DropDownControl 11211
DesignIntf.IPropertyKind 11211
DesignIntf.IPropertyKind.GetKind 11212
DesignIntf.IPropertyKind.Kind 11212
DesignIntf.IReferenceProperty 11212
DesignIntf.IReferenceProperty.GetComponentReference 11213
DesignIntf.ISelectionEditor 11213
DesignIntf.ISelectionEditor.ExecuteVerb 11214
DesignIntf.ISelectionEditor.GetVerb 11214
DesignIntf.ISelectionEditor.GetVerbCount 11215
DesignIntf.ISelectionEditor.PrepareItem 11215
DesignIntf.ISelectionEditor.RequiresUnits 11215
DesignIntf.ISelectionEditorList 11215
DesignIntf.ISelectionEditorList.Count 11217
DesignIntf.ISelectionEditorList.Get 11216
DesignIntf.ISelectionEditorList.GetCount 11216

DesignIntf.ISelectionEditorList.Items 11217
DesignIntf.ISelectionPropertyFilter 11217
DesignIntf.ISelectionPropertyFilter.FilterProperties 11218
DesignIntf.IShowReferenceProperty 11218
DesignIntf.IShowReferenceProperty.ShowReferenceProperty 11219
DesignIntf.IWideProperty 11219
DesignIntf.IWideProperty.GetEditValue 11220
DesignIntf.IWideProperty.GetValue 11220
DesignIntf.IWideProperty.SetValue 11220
DesignIntf.IWideProperty10 11221
DesignIntf.IWideProperty10.GetName 11221
DesignIntf.IWideProperty10.GetValues 11222
DesignIntf.IWidePropertyDescription 11222
DesignIntf.IWidePropertyDescription.GetDescription 11223
DesignIntf.MaxIdentLength 11255
DesignIntf.MaxIdentLength constant 11255
DesignIntf.PersistentToDesignObject 11225
DesignIntf.PersistentToDesignObject function 11225
DesignIntf.RegisterComponentEditor 11226
DesignIntf.RegisterComponentEditor function 11226
DesignIntf.RegisterComponentEditorProc 11250
DesignIntf.RegisterComponentEditorProc variable 11250
DesignIntf.RegisterComponentGuidelines 11226
DesignIntf.RegisterComponentGuidelines function 11226
DesignIntf.RegisterComponentGuidelinesProc 11251
DesignIntf.RegisterComponentGuidelinesProc variable 11251
DesignIntf.RegisterCustomModule 11226
DesignIntf.RegisterCustomModule function 11226
DesignIntf.RegisterCustomModuleProc 11251
DesignIntf.RegisterCustomModuleProc variable 11251
DesignIntf.RegisterDesignDragObject 11227
DesignIntf.RegisterDesignDragObject function 11227
DesignIntf.RegisterDesignDragObjectProc 11251
DesignIntf.RegisterDesignDragObjectProc variable 11251
DesignIntf.RegisterDesignNotification 11227
DesignIntf.RegisterDesignNotification function 11227
DesignIntf.RegisterDesignNotificationProc 11252
DesignIntf.RegisterDesignNotificationProc variable 11252
DesignIntf.RegisterDragTarget 11227
DesignIntf.RegisterDragTarget function 11227
DesignIntf.RegisterDragTargetProc 11252
DesignIntf.RegisterDragTargetProc variable 11252
DesignIntf.RegisterIDropTarget 11228
DesignIntf.RegisterIDropTarget function 11228
DesignIntf.RegisterIDropTargetProc 11252
DesignIntf.RegisterIDropTargetProc variable 11252
DesignIntf.RegisterPropertiesInCategory 11228, 11229
DesignIntf.RegisterPropertiesInCategory function 11228, 11229
DesignIntf.RegisterPropertyEditor 11229
DesignIntf.RegisterPropertyEditor function 11229
DesignIntf.RegisterPropertyEditorProc 11253
DesignIntf.RegisterPropertyEditorProc variable 11253
DesignIntf.RegisterPropertyInCategory 11229, 11230
DesignIntf.RegisterPropertyInCategory function 11229, 11230
DesignIntf.RegisterPropertyInCategoryProc 11253
DesignIntf.RegisterPropertyInCategoryProc variable 11253
DesignIntf.RegisterPropertyMapper 11231
DesignIntf.RegisterPropertyMapper function 11231
DesignIntf.RegisterPropertyMapperProc 11253
DesignIntf.RegisterPropertyMapperProc variable 11253
DesignIntf.RegisterSelectionEditor 11231
DesignIntf.RegisterSelectionEditor function 11231
DesignIntf.RegisterSelectionEditorProc 11254
DesignIntf.RegisterSelectionEditorProc variable 11254
DesignIntf.sActionCategoryName 11256
DesignIntf.sActionCategoryName constant 11256
DesignIntf.sAppearanceCategoryName 11256
DesignIntf.sAppearanceCategoryName constant 11256
DesignIntf.sBehaviorCategoryName 11256
DesignIntf.sBehaviorCategoryName constant 11256
DesignIntf.sDatabaseCategoryName 11257
DesignIntf.sDatabaseCategoryName constant 11257
DesignIntf.sDataCategoryName 11257
DesignIntf.sDataCategoryName constant 11257
DesignIntf.sDesignCategoryName 11257
DesignIntf.sDesignCategoryName constant 11257
DesignIntf.sDragNDropCategoryName 11258
DesignIntf.sDragNDropCategoryName constant 11258

DesignIntf.SetPropertyEditorGroup 11231
DesignIntf.SetPropertyEditorGroup function 11231
DesignIntf.SetPropertyEditorGroupProc 11254
DesignIntf.SetPropertyEditorGroupProc variable 11254
DesignIntf.sFocusCategoryName 11258
DesignIntf.sFocusCategoryName constant 11258
DesignIntf.sHelpCategoryName 11258
DesignIntf.sHelpCategoryName constant 11258
DesignIntf.sInputCategoryName 11259
DesignIntf.sInputCategoryName constant 11259
DesignIntf.sInvalidFilter 11259
DesignIntf.sInvalidFilter constant 11259
DesignIntf.sKeyCategoryName 11259
DesignIntf.sKeyCategoryName constant 11259
DesignIntf.sLayoutCategoryName 11260
DesignIntf.sLayoutCategoryName constant 11260
DesignIntf.sLegacyCategoryName 11260
DesignIntf.sLegacyCategoryName constant 11260
DesignIntf.sLinkageCategoryName 11260
DesignIntf.sLinkageCategoryName constant 11260
DesignIntf.sLocaleCategoryName 11261
DesignIntf.sLocaleCategoryName constant 11261
DesignIntf.sLocalizableCategoryName 11261
DesignIntf.sLocalizableCategoryName constant 11261
DesignIntf.sMiscellaneousCategoryName 11261
DesignIntf.sMiscellaneousCategoryName constant 11261
DesignIntf.sMouseCategoryName 11262
DesignIntf.sMouseCategoryName constant 11262
DesignIntf.sPropertyChangeCategoryName 11262
DesignIntf.sPropertyChangeCategoryName constant 11262
DesignIntf.sVisualCategoryName 11262
DesignIntf.sVisualCategoryName constant 11262
DesignIntf.sWindowStyleName 11263
DesignIntf.sWindowStyleName constant 11263
DesignIntf.TBaseComponentEditor 11105
DesignIntf.TBaseComponentEditor.Create 11107
DesignIntf.TBaseComponentGuidelines 11107
DesignIntf.TBaseComponentGuidelines.Create 11109
DesignIntf.TBaseComponentGuidelines.Designer 11110
DesignIntf.TBaseComponentGuidelines.Initialize 11109
DesignIntf.TBaseCustomModule 11110
DesignIntf.TBaseCustomModule.Create 11111
DesignIntf.TBaseCustomModule.DesignClass 11112
DesignIntf.TBasePropertyEditor 11112
DesignIntf.TBasePropertyEditor.Create 11114
DesignIntf.TBasePropertyEditor.Initialize 11114
DesignIntf.TBasePropertyEditor.SetPropEntry 11115
DesignIntf.TBaseSelectionEditor 11116
DesignIntf.TBaseSelectionEditor.Create 11117
DesignIntf.TClassWrapper 11117
DesignIntf.TClassWrapper.ClassNameIs 11119
DesignIntf.TClassWrapper.Create 11119
DesignIntf.TClassWrapper.GetClassName 11120
DesignIntf.TClassWrapper.GetClassParent 11120
DesignIntf.TClassWrapper.GetUnitName 11120
DesignIntf.TComponentEditorClass 11239
DesignIntf.TComponentEditorClass type 11239
DesignIntf.TComponentGuidelinesClass 11239
DesignIntf.TComponentGuidelinesClass type 11239
DesignIntf.TCustomModuleAttribute 11233
DesignIntf.TCustomModuleAttribute enumeration 11233
DesignIntf.TCustomModuleAttributes 11239
DesignIntf.TCustomModuleAttributes type 11239
DesignIntf.TCustomModuleClass 11240
DesignIntf.TCustomModuleClass type 11240
DesignIntf.TDemandLoadState 11233
DesignIntf.TDemandLoadState enumeration 11233
DesignIntf.TDesignerGuideType 11234
DesignIntf.TDesignerGuideType enumeration 11234
DesignIntf.TDesignerSelections 11121
DesignIntf.TDesignerSelections.Add 11124
DesignIntf.TDesignerSelections.Copy 11123
DesignIntf.TDesignerSelections.Count 11126
DesignIntf.TDesignerSelections.Create 11123
DesignIntf.TDesignerSelections.Destroy 11124
DesignIntf.TDesignerSelections.Equals 11125
DesignIntf.TDesignerSelections.Get 11125
DesignIntf.TDesignerSelections.GetCount 11125

DesignIntf.TDesignerSelections.GetDesignObject 11126
DesignIntf.TDesignerSelections.Items 11126
DesignIntf.TDragTarget 11127
DesignIntf.TDragTarget.Create 11128
DesignIntf.TDragTarget.Designer 11129
DesignIntf.TDragTarget.DragDrop 11128
DesignIntf.TDragTarget.DragOver 11128
DesignIntf.TDragTargetClass 11240
DesignIntf.TDragTargetClass type 11240
DesignIntf.TEditAction 11234
DesignIntf.TEditAction enumeration 11234
DesignIntf.TEditState 11240
DesignIntf.TEditState type 11240
DesignIntf.TEditStates 11235
DesignIntf.TEditStates enumeration 11235
DesignIntf.TEnableDemandLoadReportProc 11241
DesignIntf.TEnableDemandLoadReportProc type 11241
DesignIntf.TEventInfo 11129
DesignIntf.TEventInfo.Create 11131
DesignIntf.TEventInfo.GetMethodKind 11132
DesignIntf.TEventInfo.GetParamCount 11132
DesignIntf.TEventInfo.GetParamFlags 11132
DesignIntf.TEventInfo.GetParamName 11132
DesignIntf.TEventInfo.GetParamType 11132
DesignIntf.TEventInfo.GetResultType 11133
DesignIntf.TForceDemandLoadStateProc 11241
DesignIntf.TForceDemandLoadStateProc type 11241
DesignIntf.TGetDesignerEvent 11241
DesignIntf.TGetDesignerEvent type 11241
DesignIntf.TGetPropProc 11242
DesignIntf.TGetPropProc type 11242
DesignIntf.TGetWideStrProc 11242
DesignIntf.TGetWideStrProc type 11242
DesignIntf.TPropertyAttribute 11236
DesignIntf.TPropertyAttribute enumeration 11236
DesignIntf.TPropertyAttributes 11242
DesignIntf.TPropertyAttributes type 11242
DesignIntf.TPropertyEditorClass 11243
DesignIntf.TPropertyEditorClass type 11243
DesignIntf.TPropertyEditorFilterFunc 11243
DesignIntf.TPropertyEditorFilterFunc type 11243
DesignIntf.TPropertyMapperFunc 11243
DesignIntf.TPropertyMapperFunc type 11243
DesignIntf.TPropKind 11237
DesignIntf.TPropKind enumeration 11237
DesignIntf.TRegisterComponentEditorProc 11244
DesignIntf.TRegisterComponentEditorProc type 11244
DesignIntf.TRegisterComponentGuidelines 11244
DesignIntf.TRegisterComponentGuidelines type 11244
DesignIntf.TRegisterCustomModuleProc 11244
DesignIntf.TRegisterCustomModuleProc type 11244
DesignIntf.TRegisterDesignDragObject 11245
DesignIntf.TRegisterDesignDragObject type 11245
DesignIntf.TRegisterDesignNotification 11245
DesignIntf.TRegisterDesignNotification type 11245
DesignIntf.TRegisterDragTargetProc 11245
DesignIntf.TRegisterDragTargetProc type 11245
DesignIntf.TRegisterIDropTarget 11246
DesignIntf.TRegisterIDropTarget type 11246
DesignIntf.TRegisterPropertyEditorProc 11246
DesignIntf.TRegisterPropertyEditorProc type 11246
DesignIntf.TRegisterPropertyInCategoryProc 11246
DesignIntf.TRegisterPropertyInCategoryProc type 11246
DesignIntf.TRegisterPropertyMapperProc 11247
DesignIntf.TRegisterPropertyMapperProc type 11247
DesignIntf.TRegisterSelectionEditorProc 11247
DesignIntf.TRegisterSelectionEditorProc type 11247
DesignIntf.TSelectionEditorClass 11247
DesignIntf.TSelectionEditorClass type 11247
DesignIntf.TSetPropertyEditorGroupProc 11248
DesignIntf.TSetPropertyEditorGroupProc type 11248
DesignIntf.UnlistPublishedProperty 11232
DesignIntf.UnlistPublishedProperty function 11232
DesignIntf.UnregisterDesignNotification 11232
DesignIntf.UnregisterDesignNotification function 11232
DesignIntf.UnregisterDesignNotificationProc 11254
DesignIntf.UnregisterDesignNotificationProc variable 11254
deUpdateRecord enumeration member 7607

deUpdateState enumeration member 7607
dfBinary enumeration member 8536
dfLong enumeration member 5448
dfShort enumeration member 5448
dftDefault enumeration member 25318
dftImage enumeration member 25318
dftList enumeration member 25318
dftText enumeration member 25318
dfXML enumeration member 8536
dfXMLUTF8 enumeration member 8536
dgAlwaysShowEditor enumeration member 9767
dgAlwaysShowSelection enumeration member 9767
dgCancelOnExit enumeration member 9767
dgCol enumeration member 17511
dgColLines enumeration member 9767
dgColumnResize enumeration member 9767
dgConfirmDelete enumeration member 9767
dgEditing enumeration member 9767
dgIndicator enumeration member 9767
dgMultiSelect enumeration member 9767
dgNone enumeration member 17511
dgPage enumeration member 17511
dgRow enumeration member 17511
dgRowLines enumeration member 9767
dgRowSelect enumeration member 9767
dgSum enumeration member 17511
dgTabs enumeration member 9767
dgTitles enumeration member 9767
diActive enumeration member 17432
Dialogs 11265
Dialogs namespace 11265
- Classes 11269
- Constants 11586
- Functions 11531
- Structs, Records, Enums 11548
- Types 11568
- Variables 11585

Dialogs.CreateMessageDialog 11531
Dialogs.CreateMessageDialog function 11531
Dialogs.EPlatformVersionException 11269
Dialogs.ForceCurrentDirectory 11585
Dialogs.ForceCurrentDirectory variable 11585
Dialogs.InputBox 11532
Dialogs.InputBox function 11532
Dialogs.InputQuery 11533
Dialogs.InputQuery function 11533
Dialogs.MaxCustomColors 11586
Dialogs.MaxCustomColors constant 11586
Dialogs.mbAbortIgnore 11586
Dialogs.mbAbortIgnore constant 11586
Dialogs.mbAbortRetryIgnore 11587
Dialogs.mbAbortRetryIgnore constant 11587
Dialogs.mbOKCancel 11588
Dialogs.mbOKCancel constant 11588
Dialogs.mbYesAllNoAllCancel 11588
Dialogs.mbYesAllNoAllCancel constant 11588
Dialogs.mbYesNo 11589
Dialogs.mbYesNo constant 11589
Dialogs.mbYesNoCancel 11589
Dialogs.mbYesNoCancel constant 11589
Dialogs.MessageDlg 11535
Dialogs.MessageDlg function 11535
Dialogs.MessageDlgPos 11537
Dialogs.MessageDlgPos function 11537
Dialogs.MessageDlgPosHelp 11539
Dialogs.MessageDlgPosHelp function 11539
Dialogs.PromptForFileName 11540
Dialogs.PromptForFileName function 11540
Dialogs.ShowMessage 11541
Dialogs.ShowMessage function 11541
Dialogs.ShowMessageFmt 11542
Dialogs.ShowMessageFmt function 11542
Dialogs.ShowMessagePos 11543
Dialogs.ShowMessagePos function 11543
Dialogs.TaskMessageDlg 11543
Dialogs.TaskMessageDlg function 11543
Dialogs.TaskMessageDlgPos 11545
Dialogs.TaskMessageDlgPos function 11545

Dialogs.TaskMessageDlgPosHelp 11546
Dialogs.TaskMessageDlgPosHelp function 11546
Dialogs.TColorDialog 11270
Dialogs.TColorDialog.Color 11275
Dialogs.TColorDialog.Create 11273
Dialogs.TColorDialog.Ctl3D 11276
Dialogs.TColorDialog.CustomColors 11276
Dialogs.TColorDialog.Destroy 11274
Dialogs.TColorDialog.Execute 11274
Dialogs.TColorDialog.Options 11277
Dialogs.TColorDialogOption 11548
Dialogs.TColorDialogOption enumeration 11548
Dialogs.TColorDialogOptions 11568
Dialogs.TColorDialogOptions type 11568
Dialogs.TCommonDialog 11277
Dialogs.TCommonDialog.Create 11281
Dialogs.TCommonDialog.Ctl3D 11285
Dialogs.TCommonDialog.DefaultHandler 11282
Dialogs.TCommonDialog.Destroy 11281
Dialogs.TCommonDialog.DoClose 11282
Dialogs.TCommonDialog.DoShow 11283
Dialogs.TCommonDialog.Execute 11283
Dialogs.TCommonDialog.Handle 11286
Dialogs.TCommonDialog.HelpContext 11286
Dialogs.TCommonDialog.MessageHook 11284
Dialogs.TCommonDialog.OnClose 11287
Dialogs.TCommonDialog.OnShow 11288
Dialogs.TCommonDialog.TaskModalDialog 11284
Dialogs.TCommonDialog.Template 11287
Dialogs.TCommonDialog.TemplateModule 11287
Dialogs.TCommonDialog.WndProc 11285
Dialogs.TCustomColors 11569
Dialogs.TCustomColors type 11569
Dialogs.TCustomFileDialog 11288
Dialogs.TCustomFileDialog.ClientGuid 11299
Dialogs.TCustomFileDialog.Create 11292
Dialogs.TCustomFileDialog.CreateFileDialog 11293
Dialogs.TCustomFileDialog.DefaultExtension 11300
Dialogs.TCustomFileDialog.DefaultFolder 11300
Dialogs.TCustomFileDialog.Destroy 11292
Dialogs.TCustomFileDialog.Dialog 11300
Dialogs.TCustomFileDialog.DoOnExecute 11294
Dialogs.TCustomFileDialog.DoOnFileOkClick 11294
Dialogs.TCustomFileDialog.DoOnFolderChange 11294
Dialogs.TCustomFileDialog.DoOnFolderChanging 11295
Dialogs.TCustomFileDialog.DoOnOverwrite 11295
Dialogs.TCustomFileDialog.DoOnSelectionChange 11295
Dialogs.TCustomFileDialog.DoOnShareViolation 11296
Dialogs.TCustomFileDialog.DoOnTypeChange 11296
Dialogs.TCustomFileDialog.Execute 11296
Dialogs.TCustomFileDialog.FavoriteLinks 11301
Dialogs.TCustomFileDialog.FileName 11301
Dialogs.TCustomFileDialog.FileNameLabel 11301
Dialogs.TCustomFileDialog.FileOkClick 11297
Dialogs.TCustomFileDialog.Files 11302
Dialogs.TCustomFileDialog.FileTypeIndex 11302
Dialogs.TCustomFileDialog.FileTypes 11303
Dialogs.TCustomFileDialog.FolderChange 11297
Dialogs.TCustomFileDialog.FolderChanging 11297
Dialogs.TCustomFileDialog.GetFileNames 11298
Dialogs.TCustomFileDialog.GetItemName 11298
Dialogs.TCustomFileDialog.GetResults 11298
Dialogs.TCustomFileDialog.Handle 11303
Dialogs.TCustomFileDialog.OkButtonLabel 11303
Dialogs.TCustomFileDialog.OnExecute 11305
Dialogs.TCustomFileDialog.OnFileOkClick 11306
Dialogs.TCustomFileDialog.OnFolderChange 11306
Dialogs.TCustomFileDialog.OnFolderChanging 11307
Dialogs.TCustomFileDialog.OnOverwrite 11307
Dialogs.TCustomFileDialog.OnSelectionChange 11307
Dialogs.TCustomFileDialog.OnShareViolation 11308
Dialogs.TCustomFileDialog.OnTypeChange 11308
Dialogs.TCustomFileDialog.Options 11304
Dialogs.TCustomFileDialog.Overwrite 11298
Dialogs.TCustomFileDialog.SelectionChange 11298
Dialogs.TCustomFileDialog.ShareViolation 11299
Dialogs.TCustomFileDialog.ShellItem 11304
Dialogs.TCustomFileDialog.ShellItems 11304

Dialogs.TCustomFileDialog.Title 11305
Dialogs.TCustomFileDialog.TypeChange 11299
Dialogs.TCustomFileOpenDialog 11308
Dialogs.TCustomFileOpenDialog.CreateFileDialog 11313
Dialogs.TCustomFileOpenDialog.GetResults 11313
Dialogs.TCustomFileOpenDialog.SelectionChange 11313
Dialogs.TCustomFileSaveDialog 11314
Dialogs.TCustomFileSaveDialog.CreateFileDialog 11318
Dialogs.TCustomTaskDialog 11318
Dialogs.TCustomTaskDialog.Button 11328
Dialogs.TCustomTaskDialog.Buttons 11328
Dialogs.TCustomTaskDialog.CallbackProc 11323
Dialogs.TCustomTaskDialog.Caption 11328
Dialogs.TCustomTaskDialog.CommonButtons 11328
Dialogs.TCustomTaskDialog.Create 11322
Dialogs.TCustomTaskDialog.CustomFooterIcon 11329
Dialogs.TCustomTaskDialog.CustomMainIcon 11329
Dialogs.TCustomTaskDialog.DefaultButton 11329
Dialogs.TCustomTaskDialog.Destroy 11322
Dialogs.TCustomTaskDialog.DoExecute 11323
Dialogs.TCustomTaskDialog.DoOnButtonClicked 11324
Dialogs.TCustomTaskDialog.DoOnDialogContructed 11324
Dialogs.TCustomTaskDialog.DoOnDialogCreated 11324
Dialogs.TCustomTaskDialog.DoOnDialogDestroyed 11324
Dialogs.TCustomTaskDialog.DoOnExpandButtonClicked 11325
Dialogs.TCustomTaskDialog.DoOnHelp 11325
Dialogs.TCustomTaskDialog.DoOnHyperlinkClicked 11325
Dialogs.TCustomTaskDialog.DoOnNavigated 11325
Dialogs.TCustomTaskDialog.DoOnRadioButtonClicked 11326
Dialogs.TCustomTaskDialog.DoOnTimer 11326
Dialogs.TCustomTaskDialog.DoOnVerificationClicked 11326
Dialogs.TCustomTaskDialog.Execute 11327
Dialogs.TCustomTaskDialog.ExpandButtonCaption 11329
Dialogs.TCustomTaskDialog.Expanded 11330
Dialogs.TCustomTaskDialog.ExpandedText 11330
Dialogs.TCustomTaskDialog.Flags 11330
Dialogs.TCustomTaskDialog.FooterIcon 11331
Dialogs.TCustomTaskDialog.FooterText 11331
Dialogs.TCustomTaskDialog.Handle 11331
Dialogs.TCustomTaskDialog.HelpContext 11332
Dialogs.TCustomTaskDialog.MainIcon 11332
Dialogs.TCustomTaskDialog.ModalResult 11332
Dialogs.TCustomTaskDialog.OnButtonClicked 11334
Dialogs.TCustomTaskDialog.OnDialogConstructed 11335
Dialogs.TCustomTaskDialog.OnDialogCreated 11335
Dialogs.TCustomTaskDialog.OnDialogDestroyed 11335
Dialogs.TCustomTaskDialog.OnExpanded 11335
Dialogs.TCustomTaskDialog.OnHyperlinkClicked 11336
Dialogs.TCustomTaskDialog.OnNavigated 11336
Dialogs.TCustomTaskDialog.OnRadioButtonClicked 11336
Dialogs.TCustomTaskDialog.OnTimer 11336
Dialogs.TCustomTaskDialog.OnVerificationClicked 11337
Dialogs.TCustomTaskDialog.ProgressBar 11332
Dialogs.TCustomTaskDialog.RadioButton 11333
Dialogs.TCustomTaskDialog.RadioButtons 11333
Dialogs.TCustomTaskDialog.Text 11333
Dialogs.TCustomTaskDialog.Title 11333
Dialogs.TCustomTaskDialog.URL 11334
Dialogs.TCustomTaskDialog.VerificationText 11334
Dialogs.tdiError 11590
Dialogs.tdiError constant 11590
Dialogs.tdiInformation 11590
Dialogs.tdiInformation constant 11590
Dialogs.tdiNone 11591
Dialogs.tdiNone constant 11591
Dialogs.tdiShield 11591
Dialogs.tdiShield constant 11591
Dialogs.tdiWarning 11591
Dialogs.tdiWarning constant 11591
Dialogs.TFavoriteLinkItem 11337
Dialogs.TFavoriteLinkItem.GetDisplayName 11339
Dialogs.TFavoriteLinkItem.Location 11339
Dialogs.TFavoriteLinkItems 11340
Dialogs.TFavoriteLinkItems.Add 11342
Dialogs.TFavoriteLinkItems.GetEnumerator 11342
Dialogs.TFavoriteLinkItems.Items 11343
Dialogs.TFavoriteLinkItemsEnumerator 11343
Dialogs.TFavoriteLinkItemsEnumerator.Create 11344

Dialogs.TFavoriteLinkItemsEnumerator.Current 11345
Dialogs.TFavoriteLinkItemsEnumerator.GetCurrent 11344
Dialogs.TFavoriteLinkItemsEnumerator.MoveNext 11345
Dialogs.TFDApplyEvent 11569
Dialogs.TFDApplyEvent type 11569
Dialogs.TFileDialogCloseEvent 11570
Dialogs.TFileDialogCloseEvent type 11570
Dialogs.TFileDialogFolderChangingEvent 11570
Dialogs.TFileDialogFolderChangingEvent type 11570
Dialogs.TFileDialogOption 11549
Dialogs.TFileDialogOption enumeration 11549
Dialogs.TFileDialogOptions 11571
Dialogs.TFileDialogOptions type 11571
Dialogs.TFileDialogOverwriteEvent 11572
Dialogs.TFileDialogOverwriteEvent type 11572
Dialogs.TFileDialogOverwriteResponse 11551
Dialogs.TFileDialogOverwriteResponse enumeration 11551
Dialogs.TFileDialogShareViolationEvent 11572
Dialogs.TFileDialogShareViolationEvent type 11572
Dialogs.TFileDialogShareViolationResponse 11552
Dialogs.TFileDialogShareViolationResponse enumeration 11552
Dialogs.TFileEditStyle 11552
Dialogs.TFileEditStyle enumeration 11552
Dialogs.TFileOpenDialog 11346
Dialogs.TFileOpenDialog.ClientGuid 11350
Dialogs.TFileOpenDialog.DefaultExtension 11350
Dialogs.TFileOpenDialog.DefaultFolder 11351
Dialogs.TFileOpenDialog.FavoriteLinks 11351
Dialogs.TFileOpenDialog.FileName 11351
Dialogs.TFileOpenDialog.FileNameLabel 11352
Dialogs.TFileOpenDialog.FileTypeIndex 11352
Dialogs.TFileOpenDialog.FileTypes 11352
Dialogs.TFileOpenDialog.OkButtonLabel 11353
Dialogs.TFileOpenDialog.OnExecute 11354
Dialogs.TFileOpenDialog.OnFileOkClick 11355
Dialogs.TFileOpenDialog.OnFolderChange 11355
Dialogs.TFileOpenDialog.OnFolderChanging 11355
Dialogs.TFileOpenDialog.OnSelectionChange 11356
Dialogs.TFileOpenDialog.OnShareViolation 11356
Dialogs.TFileOpenDialog.OnTypeChange 11356
Dialogs.TFileOpenDialog.Options 11353
Dialogs.TFileOpenDialog.Title 11354
Dialogs.TFileSaveDialog 11357
Dialogs.TFileSaveDialog.ClientGuid 11361
Dialogs.TFileSaveDialog.DefaultExtension 11361
Dialogs.TFileSaveDialog.DefaultFolder 11362
Dialogs.TFileSaveDialog.FavoriteLinks 11362
Dialogs.TFileSaveDialog.FileName 11362
Dialogs.TFileSaveDialog.FileNameLabel 11363
Dialogs.TFileSaveDialog.FileTypeIndex 11363
Dialogs.TFileSaveDialog.FileTypes 11363
Dialogs.TFileSaveDialog.OkButtonLabel 11364
Dialogs.TFileSaveDialog.OnExecute 11365
Dialogs.TFileSaveDialog.OnFileOkClick 11365
Dialogs.TFileSaveDialog.OnFolderChange 11366
Dialogs.TFileSaveDialog.OnFolderChanging 11366
Dialogs.TFileSaveDialog.OnOverwrite 11366
Dialogs.TFileSaveDialog.OnSelectionChange 11367
Dialogs.TFileSaveDialog.OnShareViolation 11367
Dialogs.TFileSaveDialog.OnTypeChange 11367
Dialogs.TFileSaveDialog.Options 11364
Dialogs.TFileSaveDialog.Title 11364
Dialogs.TFileTypeItem 11368
Dialogs.TFileTypeItem.Create 11370
Dialogs.TFileTypeItem.Destroy 11370
Dialogs.TFileTypeItem.DisplayName 11371
Dialogs.TFileTypeItem.DisplayNameWStr 11371
Dialogs.TFileTypeItem.FileMask 11372
Dialogs.TFileTypeItem.FileMaskWStr 11372
Dialogs.TFileTypeItem.GetDisplayName 11370
Dialogs.TFileTypeItems 11372
Dialogs.TFileTypeItems.Add 11375
Dialogs.TFileTypeItems.FilterSpecArray 11375
Dialogs.TFileTypeItems.Items 11376
Dialogs.TFindDialog 11376
Dialogs.TFindDialog.CloseDialog 11381
Dialogs.TFindDialog.Create 11380
Dialogs.TFindDialog.Destroy 11380

Dialogs.TFindDialog.Execute 11381
Dialogs.TFindDialog.Find 11383
Dialogs.TFindDialog.FindText 11384
Dialogs.TFindDialog.Left 11386
Dialogs.TFindDialog.MessageHook 11383
Dialogs.TFindDialog.OnFind 11392
Dialogs.TFindDialog.Options 11386
Dialogs.TFindDialog.Position 11390
Dialogs.TFindDialog.Replace 11384
Dialogs.TFindDialog.Top 11392
Dialogs.TFindOption 11553
Dialogs.TFindOption enumeration 11553
Dialogs.TFindOptions 11573
Dialogs.TFindOptions type 11573
Dialogs.TFindReplaceFunc 11574
Dialogs.TFindReplaceFunc type 11574
Dialogs.TFontDialog 11394
Dialogs.TFontDialog.Apply 11399
Dialogs.TFontDialog.Create 11398
Dialogs.TFontDialog.Destroy 11398
Dialogs.TFontDialog.Device 11401
Dialogs.TFontDialog.Execute 11399
Dialogs.TFontDialog.Font 11401
Dialogs.TFontDialog.MaxFontSize 11403
Dialogs.TFontDialog.MinFontSize 11404
Dialogs.TFontDialog.OnApply 11406
Dialogs.TFontDialog.Options 11405
Dialogs.TFontDialog.WndProc 11400
Dialogs.TFontDialogDevice 11554
Dialogs.TFontDialogDevice enumeration 11554
Dialogs.TFontDialogOption 11555
Dialogs.TFontDialogOption enumeration 11555
Dialogs.TFontDialogOptions 11574
Dialogs.TFontDialogOptions type 11574
Dialogs.TIncludeItemEvent 11575
Dialogs.TIncludeItemEvent type 11575
Dialogs.TMsgDlgBtn 11556
Dialogs.TMsgDlgBtn enumeration 11556
Dialogs.TMsgDlgButtons 11575
Dialogs.TMsgDlgButtons type 11575
Dialogs.TMsgDlgType 11558
Dialogs.TMsgDlgType enumeration 11558
Dialogs.TOFNotifyEx 11576
Dialogs.TOFNotifyEx type 11576
Dialogs.TOpenDialog 11407
Dialogs.TOpenDialog.CanClose 11413
Dialogs.TOpenDialog.Create 11411
Dialogs.TOpenDialog.DefaultExt 11421
Dialogs.TOpenDialog.DefineProperties 11413
Dialogs.TOpenDialog.Destroy 11412
Dialogs.TOpenDialog.DoCanClose 11413
Dialogs.TOpenDialog.DoExecute 11414
Dialogs.TOpenDialog.DoFolderChange 11414
Dialogs.TOpenDialog.DoIncludeItem 11414
Dialogs.TOpenDialog.DoSelectionChange 11415
Dialogs.TOpenDialog.DoTypeChange 11415
Dialogs.TOpenDialog.Execute 11415
Dialogs.TOpenDialog.FileEditStyle 11423
Dialogs.TOpenDialog.FileName 11424
Dialogs.TOpenDialog.Files 11429
Dialogs.TOpenDialog.Filter 11431
Dialogs.TOpenDialog.FilterIndex 11434
Dialogs.TOpenDialog.GetFileNames 11417
Dialogs.TOpenDialog.GetStaticRect 11417
Dialogs.TOpenDialog.HistoryList 11436
Dialogs.TOpenDialog.InitialDir 11436
Dialogs.TOpenDialog.OnCanClose 11439
Dialogs.TOpenDialog.OnFolderChange 11440
Dialogs.TOpenDialog.OnIncludeItem 11440
Dialogs.TOpenDialog.OnSelectionChange 11441
Dialogs.TOpenDialog.OnTypeChange 11441
Dialogs.TOpenDialog.Options 11436
Dialogs.TOpenDialog.OptionsEx 11438
Dialogs.TOpenDialog.Title 11438
Dialogs.TOpenDialog.WndProc 11420
Dialogs.TOpenOption 11559
Dialogs.TOpenOption enumeration 11559
Dialogs.TOpenOptionEx 11561

Dialogs.TOpenOptionEx enumeration 11561
Dialogs.TOpenOptions 11576
Dialogs.TOpenOptions type 11576
Dialogs.TOpenOptionsEx 11578
Dialogs.TOpenOptionsEx type 11578
Dialogs.TPageMeasureUnits 11561
Dialogs.TPageMeasureUnits enumeration 11561
Dialogs.TPageSetupBeforePaintEvent 11578
Dialogs.TPageSetupBeforePaintEvent type 11578
Dialogs.TPageSetupDialog 11441
Dialogs.TPageSetupDialog.BeforePaint 11455
Dialogs.TPageSetupDialog.Create 11446
Dialogs.TPageSetupDialog.Execute 11446
Dialogs.TPageSetupDialog.GetDefaults 11446
Dialogs.TPageSetupDialog.MarginBottom 11447
Dialogs.TPageSetupDialog.MarginLeft 11448
Dialogs.TPageSetupDialog.MarginRight 11448
Dialogs.TPageSetupDialog.MarginTop 11449
Dialogs.TPageSetupDialog.MinMarginBottom 11450
Dialogs.TPageSetupDialog.MinMarginLeft 11450
Dialogs.TPageSetupDialog.MinMarginRight 11451
Dialogs.TPageSetupDialog.MinMarginTop 11452
Dialogs.TPageSetupDialog.OnDrawEnvStamp 11455
Dialogs.TPageSetupDialog.OnDrawFullPage 11456
Dialogs.TPageSetupDialog.OnDrawGreekText 11456
Dialogs.TPageSetupDialog.OnDrawMargin 11457
Dialogs.TPageSetupDialog.OnDrawMinMargin 11457
Dialogs.TPageSetupDialog.OnDrawRetAddress 11458
Dialogs.TPageSetupDialog.Options 11452
Dialogs.TPageSetupDialog.PageHeight 11453
Dialogs.TPageSetupDialog.PageSetupDlgRec 11454
Dialogs.TPageSetupDialog.PageWidth 11454
Dialogs.TPageSetupDialog.Units 11454
Dialogs.TPageSetupDialogOption 11562
Dialogs.TPageSetupDialogOption enumeration 11562
Dialogs.TPageSetupDialogOptions 11579
Dialogs.TPageSetupDialogOptions type 11579
Dialogs.TPageType 11562
Dialogs.TPageType enumeration 11562
Dialogs.TPaintPageEvent 11579
Dialogs.TPaintPageEvent type 11579
Dialogs.TPrintDialog 11458
Dialogs.TPrintDialog.Collate 11464
Dialogs.TPrintDialog.Copies 11464
Dialogs.TPrintDialog.Execute 11462
Dialogs.TPrintDialog.FromPage 11465
Dialogs.TPrintDialog.MaxPage 11467
Dialogs.TPrintDialog.MinPage 11469
Dialogs.TPrintDialog.Options 11471
Dialogs.TPrintDialog.PrintRange 11473
Dialogs.TPrintDialog.PrintToFile 11475
Dialogs.TPrintDialog.ToPage 11476
Dialogs.TPrintDialogOption 11563
Dialogs.TPrintDialogOption enumeration 11563
Dialogs.TPrintDialogOptions 11580
Dialogs.TPrintDialogOptions type 11580
Dialogs.TPrinterKind 11564
Dialogs.TPrinterKind enumeration 11564
Dialogs.TPrinterOrientation 11581
Dialogs.TPrinterOrientation type 11581
Dialogs.TPrinterSetupDialog 11478
Dialogs.TPrinterSetupDialog.Execute 11482
Dialogs.TPrintRange 11564
Dialogs.TPrintRange enumeration 11564
Dialogs.TProgressBarState 11565
Dialogs.TProgressBarState enumeration 11565
Dialogs.TReplaceDialog 11482
Dialogs.TReplaceDialog.Create 11486
Dialogs.TReplaceDialog.OnReplace 11488
Dialogs.TReplaceDialog.ReplaceText 11487
Dialogs.TSaveDialog 11489
Dialogs.TSaveDialog.Execute 11494
Dialogs.TTaskDialog 11494
Dialogs.TTaskDialog.Buttons 11499
Dialogs.TTaskDialog.Caption 11499
Dialogs.TTaskDialog.CommonButtons 11499
Dialogs.TTaskDialog.CustomFooterIcon 11500
Dialogs.TTaskDialog.CustomMainIcon 11500

Dialogs.TTaskDialog.DefaultButton 11500
Dialogs.TTaskDialog.ExpandButtonCaption 11500
Dialogs.TTaskDialog.ExpandedText 11501
Dialogs.TTaskDialog.Flags 11501
Dialogs.TTaskDialog.FooterIcon 11502
Dialogs.TTaskDialog.FooterText 11502
Dialogs.TTaskDialog.HelpContext 11502
Dialogs.TTaskDialog.MainIcon 11502
Dialogs.TTaskDialog.OnButtonClicked 11504
Dialogs.TTaskDialog.OnDialogConstructed 11504
Dialogs.TTaskDialog.OnDialogCreated 11505
Dialogs.TTaskDialog.OnDialogDestroyed 11505
Dialogs.TTaskDialog.OnExpanded 11505
Dialogs.TTaskDialog.OnHyperlinkClicked 11505
Dialogs.TTaskDialog.OnNavigated 11506
Dialogs.TTaskDialog.OnRadioButtonClicked 11506
Dialogs.TTaskDialog.OnTimer 11506
Dialogs.TTaskDialog.OnVerificationClicked 11506
Dialogs.TTaskDialog.ProgressBar 11503
Dialogs.TTaskDialog.RadioButtons 11503
Dialogs.TTaskDialog.Text 11503
Dialogs.TTaskDialog.Title 11503
Dialogs.TTaskDialog.VerificationText 11504
Dialogs.TTaskDialogBaseButtonItem 11507
Dialogs.TTaskDialogBaseButtonItem.Caption 11511
Dialogs.TTaskDialogBaseButtonItem.Click 11509
Dialogs.TTaskDialogBaseButtonItem.Client 11512
Dialogs.TTaskDialogBaseButtonItem.Create 11509
Dialogs.TTaskDialogBaseButtonItem.Default 11512
Dialogs.TTaskDialogBaseButtonItem.Destroy 11509
Dialogs.TTaskDialogBaseButtonItem.DoButtonClick 11510
Dialogs.TTaskDialogBaseButtonItem.DoSetEnabled 11510
Dialogs.TTaskDialogBaseButtonItem.Enabled 11512
Dialogs.TTaskDialogBaseButtonItem.GetButtonText 11510
Dialogs.TTaskDialogBaseButtonItem.GetDisplayName 11510
Dialogs.TTaskDialogBaseButtonItem.ModalResult 11512
Dialogs.TTaskDialogBaseButtonItem.SetInitialState 11511
Dialogs.TTaskDialogBaseButtonItem.TextWStr 11513
Dialogs.TTaskDialogButtonItem 11513
Dialogs.TTaskDialogButtonItem.CommandLinkHint 11516
Dialogs.TTaskDialogButtonItem.Create 11515
Dialogs.TTaskDialogButtonItem.ElevationRequired 11516
Dialogs.TTaskDialogButtonItem.GetButtonText 11515
Dialogs.TTaskDialogButtonItem.ModalResult 11517
Dialogs.TTaskDialogButtonItem.SetInitialState 11516
Dialogs.TTaskDialogButtonList 11581
Dialogs.TTaskDialogButtonList type 11581
Dialogs.TTaskDialogButtons 11517
Dialogs.TTaskDialogButtons.Add 11519
Dialogs.TTaskDialogButtons.Buttons 11519
Dialogs.TTaskDialogButtons.DefaultButton 11521
Dialogs.TTaskDialogButtons.FindButton 11520
Dialogs.TTaskDialogButtons.GetEnumerator 11520
Dialogs.TTaskDialogButtons.Items 11521
Dialogs.TTaskDialogButtons.SetInitialState 11520
Dialogs.TTaskDialogButtonsEnumerator 11521
Dialogs.TTaskDialogButtonsEnumerator.Create 11522
Dialogs.TTaskDialogButtonsEnumerator.Current 11523
Dialogs.TTaskDialogButtonsEnumerator.GetCurrent 11522
Dialogs.TTaskDialogButtonsEnumerator.MoveNext 11522
Dialogs.TTaskDialogCommonButton 11565
Dialogs.TTaskDialogCommonButton enumeration 11565
Dialogs.TTaskDialogCommonButtons 11582
Dialogs.TTaskDialogCommonButtons type 11582
Dialogs.TTaskDialogFlag 11566
Dialogs.TTaskDialogFlag enumeration 11566
Dialogs.TTaskDialogFlags 11582
Dialogs.TTaskDialogFlags type 11582
Dialogs.TTaskDialogIcon 11583
Dialogs.TTaskDialogIcon type 11583
Dialogs.TTaskDialogProgressBar 11523
Dialogs.TTaskDialogProgressBar.Create 11525
Dialogs.TTaskDialogProgressBar.Initialize 11525
Dialogs.TTaskDialogProgressBar.MarqueeSpeed 11526
Dialogs.TTaskDialogProgressBar.Max 11526
Dialogs.TTaskDialogProgressBar.Min 11526
Dialogs.TTaskDialogProgressBar.Position 11526
Dialogs.TTaskDialogProgressBar.State 11527

Dialogs.TTaskDialogRadioButtonItem 11527
Dialogs.TTaskDialogRadioButtonItem.Create 11529
Dialogs.TTaskDialogRadioButtonItem.DoButtonClick 11529
Dialogs.TTaskDialogRadioButtonItem.DoSetEnabled 11529
Dialogs.TTaskDlgClickEvent 11583
Dialogs.TTaskDlgClickEvent type 11583
Dialogs.TTaskDlgTimerEvent 11584
Dialogs.TTaskDlgTimerEvent type 11584
Dialogs.UseLatestCommonDialogs 11585
Dialogs.UseLatestCommonDialogs variable 11585
diAsNeeded enumeration member 17432
diInactive enumeration member 17432
dimAverage enumeration member 17433
dimCount enumeration member 17433
dimDimension enumeration member 17433
dimGenericAgg enumeration member 17433
dimMax enumeration member 17433
dimMin enumeration member 17433
dimSum enumeration member 17433
dimUnknown enumeration member 17433
dkDock enumeration member 6282
dkDrag enumeration member 6282
dlDefault enumeration member 11233
dlDisable enumeration member 11233
dlEnable enumeration member 11233
dmActiveForm enumeration member 14357
dmAutomatic enumeration member 6283
dmClosed enumeration member 17512
dmComboBox enumeration member 5448
dmDesktop enumeration member 14357
dmDragCancel enumeration member 6283
dmDragDrop enumeration member 6283
dmDragEnter enumeration member 6283
dmDragLeave enumeration member 6283
dmDragMove enumeration member 6283
dmDrilled enumeration member 17512
dmFindTarget enumeration member 6283
dmMainForm enumeration member 14357
dmManual enumeration member 6283
dmNone enumeration member 17512
dmOpen enumeration member 17512
dmPaged enumeration member 17512
dmPrimary enumeration member 14357
dmUpDown enumeration member 5448
doAttrNull enumeration member 26460
doAutoColResize enumeration member 23509
doAutoPrefix enumeration member 26460
doAutoSave enumeration member 26460
DockTabSet 11593
DockTabSet namespace 11593
    Classes 11594
    Interfaces 11643
DockTabSet.ETabDockException 11594
DockTabSet.IAlternateDockHost 11643
DockTabSet.IAlternateDockHost.SetAlternateDockHost 11644
DockTabSet.TCaptionedTabDockTree 11594
DockTabSet.TCaptionedTabDockTree.MouseUp 11597
DockTabSet.TCaptionedTabDockTree.SetAlternateDockHost 11597
DockTabSet.TDockClientInfo 11598
DockTabSet.TDockClientInfo.Create 11601
DockTabSet.TDockClientInfo.CreateTab 11602
DockTabSet.TDockClientInfo.Destroy 11601
DockTabSet.TDockClientInfo.DockClient 11603
DockTabSet.TDockClientInfo.DockTabSet 11603
DockTabSet.TDockClientInfo.ImageIndex 11603
DockTabSet.TDockClientInfo.ListIndex 11603
DockTabSet.TDockClientInfo.Notification 11602
DockTabSet.TDockClientInfo.ParentPanel 11604
DockTabSet.TDockClientInfo.TabIndex 11604
DockTabSet.TDockTabSet 11604
DockTabSet.TDockTabSet.AutoSelect 11622
DockTabSet.TDockTabSet.Create 11616
DockTabSet.TDockTabSet.DestinationDockSite 11623
DockTabSet.TDockTabSet.Destroy 11617
DockTabSet.TDockTabSet.DoAddDockClient 11617
DockTabSet.TDockTabSet.DockOver 11618
DockTabSet.TDockTabSet.DockReplaceDockClient 11619
DockTabSet.TDockTabSet.DockSite 11623

DockTabSet.TDockTabSet.DoRemoveDockClient 11619
DockTabSet.TDockTabSet.FindDockClientInfo 11619
DockTabSet.TDockTabSet.GetImageIndex 11620
DockTabSet.TDockTabSet.GetTabName 11620
DockTabSet.TDockTabSet.HideCurrentDockClient 11620
DockTabSet.TDockTabSet.Images 11623
DockTabSet.TDockTabSet.IndexOfDockClient 11620
DockTabSet.TDockTabSet.MouseMove 11620
DockTabSet.TDockTabSet.Notification 11621
DockTabSet.TDockTabSet.OnDockDrop 11624
DockTabSet.TDockTabSet.OnDockOver 11624
DockTabSet.TDockTabSet.OnGetImageIndex 11624
DockTabSet.TDockTabSet.OnGetSiteInfo 11625
DockTabSet.TDockTabSet.OnStartDock 11625
DockTabSet.TDockTabSet.OnTabAdded 11625
DockTabSet.TDockTabSet.OnTabRemoved 11625
DockTabSet.TDockTabSet.OnUnDock 11625
DockTabSet.TDockTabSet.SetTabIndex 11621
DockTabSet.TDockTabSet.ShouldAutoHide 11622
DockTabSet.TDockTabSet.ShowDockClient 11622
DockTabSet.TDockTabSet.ShowDockClientInfo 11622
DockTabSet.TTabDockPanel 11625
DockTabSet.TTabDockPanel.AnimateSpeed 11642
DockTabSet.TTabDockPanel.BeginAnimation 11639
DockTabSet.TTabDockPanel.CMShowingChanged 11640
DockTabSet.TTabDockPanel.Create 11638
DockTabSet.TTabDockPanel.Destroy 11638
DockTabSet.TTabDockPanel.DockClientInfo 11642
DockTabSet.TTabDockPanel.EndAnimation 11640
DockTabSet.TTabDockPanel.ShouldAutoHide 11640
DockTabSet.TTabDockPanel.SplitterPanel 11642
DockTabSet.TTabDockPanel.SplitterWidth 11642
DockTabSet.TTabDockPanel.UpdateDockCaptionPin 11640
DockTabSet.TTabDockPanel.WMEraseBkgnd 11640
DockTabSet.TTabDockPanel.WMNCPaint 11641
DockTabSet.TTabDockPanel.WMPaint 11641
DockTabSet.TTabDockPanel.WMPrint 11641
DockTabSet.TTabDockPanel.WMPrintClient 11641
doColumnTitles enumeration member 23509
doDisableDeletes enumeration member 8537
doDisableEdits enumeration member 8537
doDisableInserts enumeration member 8537
doHorizontal enumeration member 6282
doKeyColFixed enumeration member 23509
DOMException class 26256
    about DOMException class 26256
    code 26257
    DOMException fields 26257
    DOMException Members 26256
doNamespaceDecl enumeration member 26460
doNodeAutoCreate enumeration member 26460
doNodeAutoIndent enumeration member 26460
doNoOrient enumeration member 6282
doNoResetCall enumeration member 8537
doVertical enumeration member 6282
dowFriday enumeration member 5441
dowLocaleDefault enumeration member 5441
dowMonday enumeration member 5441
dowSaturday enumeration member 5441
dowSunday enumeration member 5441
dowThursday enumeration member 5441
dowTuesday enumeration member 5441
dowWednesday enumeration member 5441
dpPublished enumeration member 19792
drBounds enumeration member 5447
drIcon enumeration member 5447
DRIntf 11645
DRIntf namespace 11645
    Constants 11664
    Functions 11647
    Structs, Records, Enums 11661
    Types 11662
DRIntf.AssociateAttr 11647
DRIntf.AssociateAttr function 11647
DRIntf.CreateField 11648
DRIntf.CreateField function 11648
DRIntf.CurrentDictionary 11649
DRIntf.CurrentDictionary function 11649

DRIntf.DictionaryActive 11649
DRIntf.DictionaryActive function 11649
DRIntf.DictionaryDeactivate 11649
DRIntf.DictionaryDeactivate function 11649
DRIntf.FindAttrID 11650
DRIntf.FindAttrID function 11650
DRIntf.FindDatabaseID 11650
DRIntf.FindDatabaseID function 11650
DRIntf.FindFieldID 11651
DRIntf.FindFieldID function 11651
DRIntf.FindTableID 11652
DRIntf.FindTableID function 11652
DRIntf.GetAttrID 11652
DRIntf.GetAttrID function 11652
DRIntf.GetAttrName 11653
DRIntf.GetAttrName function 11653
DRIntf.GetAttrNames 11653
DRIntf.GetAttrNames function 11653
DRIntf.GetControlClass 11654
DRIntf.GetControlClass function 11654
DRIntf.HasConstraints 11654
DRIntf.HasConstraints function 11654
DRIntf.IsNullID 11655
DRIntf.IsNullID function 11655
DRIntf.NewAttr 11656
DRIntf.NewAttr function 11656
DRIntf.NullAttrID 11664
DRIntf.NullAttrID constant 11664
DRIntf.NullDatabaseID 11664
DRIntf.NullDatabaseID constant 11664
DRIntf.NullFieldID 11665
DRIntf.NullFieldID constant 11665
DRIntf.NullID 11665
DRIntf.NullID constant 11665
DRIntf.NullTableID 11665
DRIntf.NullTableID constant 11665
DRIntf.QualifyTableName 11656
DRIntf.QualifyTableName function 11656
DRIntf.QualifyTableNameByName 11657
DRIntf.QualifyTableNameByName function 11657
DRIntf.TAttrID 11662
DRIntf.TAttrID type 11662
DRIntf.TDatabaseID 11662
DRIntf.TDatabaseID type 11662
DRIntf.TFieldID 11663
DRIntf.TFieldID type 11663
DRIntf.TID 11661
DRIntf.TID record 11661
DRIntf.TTableID 11663
DRIntf.TTableID type 11663
DRIntf.UnassociateAttr 11657
DRIntf.UnassociateAttr function 11657
DRIntf.UpdateAttr 11658
DRIntf.UpdateAttr function 11658
DRIntf.UpdateConstraints 11659
DRIntf.UpdateConstraints function 11659
DRIntf.UpdateDataset 11659
DRIntf.UpdateDataset function 11659
DRIntf.UpdateField 11660
DRIntf.UpdateField function 11660
drLabel enumeration member 5447
drSelectBounds enumeration member 5447
dsAcceptRanges enumeration member 12097
dsAllData enumeration member 17838
dsBeginDownloadComponents enumeration member 12097
dsBeginDownloadData enumeration member 12097
dsBeginSyncOperation enumeration member 12097
dsBeginUploadData enumeration member 12097
dsBlockRead enumeration member 7608
dsBrowse enumeration member 7608
dsCacheFileNameAvailable enumeration member 12097
dsCalcFields enumeration member 7608
dsClassIDAvailable enumeration member 12097
dsClassInstallLocation enumeration member 12097
dsCloseAfter enumeration member 17673
dsCloseBefore enumeration member 17673
dsCLSIDCanInstantiate enumeration member 12097
dsColCaption enumeration member 17673

dsColIndicator enumeration member 17673
dsColPlus enumeration member 17673
dsColValue enumeration member 17673
dsCompactPolicyReceived enumeration member 12097
dsConnecting enumeration member 12097
dsContentDispositionAttach enumeration member 12097
dsCookeStateReject enumeration member 12097
dsCookieSent enumeration member 12097
dsCookieStateAccept enumeration member 12097
dsCookieStateDowngrade enumeration member 12097
dsCookieStateLeash enumeration member 12097
dsCookieStatePrompt enumeration member 12097
dsCookieStateUnknown enumeration member 12097
dsCookieSuppressed enumeration member 12097
dsCurValue enumeration member 7608
dsData enumeration member 17673
dsDecoding enumeration member 12097
dsDimensionData enumeration member 17838
dsDirectBind enumeration member 12097
dsDownloadingData enumeration member 12097
dsDragEnter enumeration member 6284
dsDragLeave enumeration member 6284
dsDragMove enumeration member 6284
dsEdit enumeration member 7608
dsEncoding enumeration member 12097
dsEndDownloadComponents enumeration member 12097
dsEndDownloadData enumeration member 12097
dsEndSyncOperation enumeration member 12097
dsEndUploadData enumeration member 12097
dsFilter enumeration member 7608
dsFilterReportMIMEType enumeration member 12097
dsFindingResource enumeration member 12097
dsFocus enumeration member 15394
dsGradient enumeration member 5468, 13265
dsGroupStart enumeration member 17673
DShellNameSpaceEvents interface 19278
    about DShellNameSpaceEvents interface 19278
    DoubleClick 19278
    DShellNameSpaceEvents Members 19278
    DShellNameSpaceEvents methods 19278
    FavoritesSelectionChange 19279
    Initialized 19279
    SelectionChange 19279
DShellWindowsEvents interface 19279
    about DShellWindowsEvents interface 19279
    DShellWindowsEvents Members 19280
    DShellWindowsEvents methods 19280
    WindowRegistered 19280
    WindowRevoked 19280
dsInactive enumeration member 7608
dsInsert enumeration member 7608
dsInstallingComponents enumeration member 12097
dsInternalCalc enumeration member 7608
DSIntf 11667
DSIntf namespace 11667
    Constants 11723
    Functions 11702
    Interfaces 11673
    Structs, Records, Enums 11705
    Types 11713
DSIntf.AGGSTATE 11705
DSIntf.AGGSTATE enumeration 11705
DSIntf.AGGVALUE 11706
DSIntf.AGGVALUE record 11706
DSIntf.BLANK_NOTCHANGED 11727
DSIntf.BLANK_NOTCHANGED constant 11727
DSIntf.BLANK_NULL 11727
DSIntf.BLANK_NULL constant 11727
DSIntf.BlobTypeMap 11727
DSIntf.BlobTypeMap constant 11727
DSIntf.CLSID_DSBase 11728
DSIntf.CLSID_DSBase constant 11728
DSIntf.CLSID_DSCursor 11728
DSIntf.CLSID_DSCursor constant 11728
DSIntf.CLSID_DSWriter 11728
DSIntf.CLSID_DSWriter constant 11728
DSIntf.CreateDbClientObject 11702
DSIntf.CreateDbClientObject function 11702

DSIntf.CURProp 11706
DSIntf.CURProp enumeration 11706
DSIntf.DataPacketSize 11702
DSIntf.DataPacketSize function 11702
DSIntf.DataPacketToVariant 11703
DSIntf.DataPacketToVariant function 11703
DSIntf.DataTypeMap 11729
DSIntf.DataTypeMap constant 11729
DSIntf.DBERR_ABORTED 11729
DSIntf.DBERR_ABORTED constant 11729
DSIntf.DBERR_BLOBNOTFETCHED 11729
DSIntf.DBERR_BLOBNOTFETCHED constant 11729
DSIntf.DBERR_BOF 11730
DSIntf.DBERR_BOF constant 11730
DSIntf.DBERR_CANCELLED 11730
DSIntf.DBERR_CANCELLED constant 11730
DSIntf.DBERR_CANNOTAPPEND 11730
DSIntf.DBERR_CANNOTAPPEND constant 11730
DSIntf.DBERR_DATAPACKETMISMATCH 11731
DSIntf.DBERR_DATAPACKETMISMATCH constant 11731
DSIntf.DBERR_DELTAISEMPTY 11731
DSIntf.DBERR_DELTAISEMPTY constant 11731
DSIntf.DBERR_DETAILSNOTFETCHED 11731
DSIntf.DBERR_DETAILSNOTFETCHED constant 11731
DSIntf.DBERR_EOF 11732
DSIntf.DBERR_EOF constant 11732
DSIntf.DBERR_FLYAWAY_WRONGORDER 11732
DSIntf.DBERR_FLYAWAY_WRONGORDER constant 11732
DSIntf.DBERR_LINKFIELDSNOTUNIQUE 11732
DSIntf.DBERR_LINKFIELDSNOTUNIQUE constant 11732
DSIntf.DBERR_NEWERVERSIONREQ 11733
DSIntf.DBERR_NEWERVERSIONREQ constant 11733
DSIntf.DBERR_NOCASCADEDUPDATES 11733
DSIntf.DBERR_NOCASCADEDUPDATES constant 11733
DSIntf.DBERR_NOMASTERRECORD 11733
DSIntf.DBERR_NOMASTERRECORD constant 11733
DSIntf.DBERR_NOMETADATA 11734
DSIntf.DBERR_NOMETADATA constant 11734
DSIntf.DBERR_NONE 11734
DSIntf.DBERR_NONE constant 11734
DSIntf.DBERR_NOSUCHINDEX 11734
DSIntf.DBERR_NOSUCHINDEX constant 11734
DSIntf.DBERR_NOTHINGTOUNDO 11735
DSIntf.DBERR_NOTHINGTOUNDO constant 11735
DSIntf.DBResult 11713
DSIntf.DBResult type 11713
DSIntf.DBSearchCond 11707
DSIntf.DBSearchCond enumeration 11707
DSIntf.dsArrayFldType 11735
DSIntf.dsArrayFldType constant 11735
DSIntf.DSAttr 11714
DSIntf.DSAttr type 11714
DSIntf.DSBOOKMRK 11707
DSIntf.DSBOOKMRK record 11707
DSIntf.dsCASCADEDELETES 11735
DSIntf.dsCASCADEDELETES constant 11735
DSIntf.dsCASCADEUPDATES 11736
DSIntf.dsCASCADEUPDATES constant 11736
DSIntf.dscbrABORT 11736
DSIntf.dscbrABORT constant 11736
DSIntf.dscbrAPPLY 11736
DSIntf.dscbrAPPLY constant 11736
DSIntf.dscbrCANCEL 11737
DSIntf.dscbrCANCEL constant 11737
DSIntf.dscbrCORRECT 11737
DSIntf.dscbrCORRECT constant 11737
DSIntf.dscbrIGNORE 11737
DSIntf.dscbrIGNORE constant 11737
DSIntf.dscbrMERGE 11738
DSIntf.dscbrMERGE constant 11738
DSIntf.dscbrREFRESH 11738
DSIntf.dscbrREFRESH constant 11738
DSIntf.dscbrSKIP 11738
DSIntf.dscbrSKIP constant 11738
DSIntf.dsCBRType 11714
DSIntf.dsCBRType type 11714
DSIntf.dsCompArrayFldType 11739
DSIntf.dsCompArrayFldType constant 11739

DSIntf.dsDELAYEDBIT 11739
DSIntf.dsDELAYEDBIT constant 11739
DSIntf.dsDetUpd 11739
DSIntf.dsDetUpd constant 11739
DSIntf.dsEmbeddedFldType 11740
DSIntf.dsEmbeddedFldType constant 11740
DSIntf.dsfldADT 11740
DSIntf.dsfldADT constant 11740
DSIntf.dsfldARRAY 11740
DSIntf.dsfldARRAY constant 11740
DSIntf.dsfldBCD 11741
DSIntf.dsfldBCD constant 11741
DSIntf.dsfldBOOL 11741
DSIntf.dsfldBOOL constant 11741
DSIntf.dsfldBYTES 11741
DSIntf.dsfldBYTES constant 11741
DSIntf.dsfldDATE 11742
DSIntf.dsfldDATE constant 11742
DSIntf.dsfldDATETIME 11742
DSIntf.dsfldDATETIME constant 11742
DSIntf.DSFLDDesc 11708
DSIntf.DSFLDDesc record 11708
DSIntf.dsfldEMBEDDEDTBL 11742
DSIntf.dsfldEMBEDDEDTBL constant 11742
DSIntf.dsfldFLOATIEEE 11743
DSIntf.dsfldFLOATIEEE constant 11743
DSIntf.dsfldFMTBCD 11743
DSIntf.dsfldFMTBCD constant 11743
DSIntf.dsfldINT 11743
DSIntf.dsfldINT constant 11743
DSIntf.dsfldREF 11744
DSIntf.dsfldREF constant 11744
DSIntf.dsfldTIME 11744
DSIntf.dsfldTIME constant 11744
DSIntf.dsfldTIMESTAMP 11744
DSIntf.dsfldTIMESTAMP constant 11744
DSIntf.dsfldUINT 11745
DSIntf.dsfldUINT constant 11745
DSIntf.dsfldUNICODE 11745
DSIntf.dsfldUNICODE constant 11745
DSIntf.dsfldUNKNOWN 11745
DSIntf.dsfldUNKNOWN constant 11745
DSIntf.dsfldZSTRING 11746
DSIntf.dsfldZSTRING constant 11746
DSIntf.DSIDXDesc 11708
DSIntf.DSIDXDesc record 11708
DSIntf.dsIncInDelta 11746
DSIntf.dsIncInDelta constant 11746
DSIntf.dsIsNotVisible 11746
DSIntf.dsIsNotVisible constant 11746
DSIntf.dsIsVisible 11747
DSIntf.dsIsVisible constant 11747
DSIntf.DSKEY 11714
DSIntf.DSKEY type 11714
DSIntf.DSKEYBOOL 11715
DSIntf.DSKEYBOOL type 11715
DSIntf.dskeyCASEINSENSITIVE 11747
DSIntf.dskeyCASEINSENSITIVE constant 11747
DSIntf.dskeyDESCENDING 11747
DSIntf.dskeyDESCENDING constant 11747
DSIntf.DSProp 11709
DSIntf.DSProp enumeration 11709
DSIntf.DSProps 11710
DSIntf.DSProps record 11710
DSIntf.dsPseudoFldType 11748
DSIntf.dsPseudoFldType constant 11748
DSIntf.dsRecDeleted 11748
DSIntf.dsRecDeleted constant 11748
DSIntf.dsRecModified 11748
DSIntf.dsRecModified constant 11748
DSIntf.dsRecNew 11749
DSIntf.dsRecNew constant 11749
DSIntf.dsRecOrg 11749
DSIntf.dsRecOrg constant 11749
DSIntf.dsRecUnmodified 11749
DSIntf.dsRecUnmodified constant 11749
DSIntf.dsSizeBitsLen 11750
DSIntf.dsSizeBitsLen constant 11750

DSIntf.dsSizeBitsMask 11750
DSIntf.dsSizeBitsMask constant 11750
DSIntf.dsTypeBitsMask 11750
DSIntf.dsTypeBitsMask constant 11750
DSIntf.dsUnused 11751
DSIntf.dsUnused constant 11751
DSIntf.dsVaryingFldType 11751
DSIntf.dsVaryingFldType constant 11751
DSIntf.ERRBASE_ALC 11751
DSIntf.ERRBASE_ALC constant 11751
DSIntf.ERRCAT_ALC 11752
DSIntf.ERRCAT_ALC constant 11752
DSIntf.ERRCODE_ABORTED 11752
DSIntf.ERRCODE_ABORTED constant 11752
DSIntf.ERRCODE_BLOBNOTFETCHED 11752
DSIntf.ERRCODE_BLOBNOTFETCHED constant 11752
DSIntf.ERRCODE_CANCELLED 11753
DSIntf.ERRCODE_CANCELLED constant 11753
DSIntf.ERRCODE_CANNOTAPPEND 11753
DSIntf.ERRCODE_CANNOTAPPEND constant 11753
DSIntf.ERRCODE_DATAPACKETMISMATCH 11753
DSIntf.ERRCODE_DATAPACKETMISMATCH constant 11753
DSIntf.ERRCODE_DELTAISEMPTY 11754
DSIntf.ERRCODE_DELTAISEMPTY constant 11754
DSIntf.ERRCODE_DETAILSNOTFETCHED 11754
DSIntf.ERRCODE_DETAILSNOTFETCHED constant 11754
DSIntf.ERRCODE_FLYAWAY_WRONGORDER 11754
DSIntf.ERRCODE_FLYAWAY_WRONGORDER constant 11754
DSIntf.ERRCODE_LINKFIELDSNOTUNIQUE 11755
DSIntf.ERRCODE_LINKFIELDSNOTUNIQUE constant 11755
DSIntf.ERRCODE_NEWERVERSIONREQ 11755
DSIntf.ERRCODE_NEWERVERSIONREQ constant 11755
DSIntf.ERRCODE_NOCASCADEDUPDATES 11755
DSIntf.ERRCODE_NOCASCADEDUPDATES constant 11755
DSIntf.ERRCODE_NOMASTERRECORD 11756
DSIntf.ERRCODE_NOMASTERRECORD constant 11756
DSIntf.ERRCODE_NOMETADATA 11756
DSIntf.ERRCODE_NOMETADATA constant 11756
DSIntf.ERRCODE_NOTHINGTOUNDO 11756
DSIntf.ERRCODE_NOTHINGTOUNDO constant 11756
DSIntf.FieldTypeMap 11757
DSIntf.FieldTypeMap constant 11757
DSIntf.fldADT 11757
DSIntf.fldADT constant 11757
DSIntf.fldARRAY 11757
DSIntf.fldARRAY constant 11757
DSIntf.fldAttrHIDDEN 11758
DSIntf.fldAttrHIDDEN constant 11758
DSIntf.fldAttrLINK 11758
DSIntf.fldAttrLINK constant 11758
DSIntf.fldAttrREADONLY 11758
DSIntf.fldAttrREADONLY constant 11758
DSIntf.fldAttrREQUIRED 11759
DSIntf.fldAttrREQUIRED constant 11759
DSIntf.fldBCD 11759
DSIntf.fldBCD constant 11759
DSIntf.fldBLOB 11759
DSIntf.fldBLOB constant 11759
DSIntf.fldBOOL 11760
DSIntf.fldBOOL constant 11760
DSIntf.fldBYTES 11760
DSIntf.fldBYTES constant 11760
DSIntf.fldCURSOR 11760
DSIntf.fldCURSOR constant 11760
DSIntf.fldDATE 11761
DSIntf.fldDATE constant 11761
DSIntf.fldDATETIME 11761
DSIntf.fldDATETIME constant 11761
DSIntf.fldFLOAT 11761
DSIntf.fldFLOAT constant 11761
DSIntf.fldFLOATIEEE 11762
DSIntf.fldFLOATIEEE constant 11762
DSIntf.fldFMTBCD 11762
DSIntf.fldFMTBCD constant 11762
DSIntf.fldINT16 11762
DSIntf.fldINT16 constant 11762
DSIntf.fldINT32 11763
DSIntf.fldINT32 constant 11763

DSIntf.fldINT64 11763
DSIntf.fldINT64 constant 11763
DSIntf.fldLOCKINFO 11763
DSIntf.fldLOCKINFO constant 11763
DSIntf.fldREF 11764
DSIntf.fldREF constant 11764
DSIntf.fldstACCOLEOBJ 11764
DSIntf.fldstACCOLEOBJ constant 11764
DSIntf.fldstAUTOINC 11764
DSIntf.fldstAUTOINC constant 11764
DSIntf.fldstBFILE 11765
DSIntf.fldstBFILE constant 11765
DSIntf.fldstBINARY 11765
DSIntf.fldstBINARY constant 11765
DSIntf.fldstDBSOLEOBJ 11765
DSIntf.fldstDBSOLEOBJ constant 11765
DSIntf.fldstFIXED 11766
DSIntf.fldstFIXED constant 11766
DSIntf.fldstFMTMEMO 11766
DSIntf.fldstFMTMEMO constant 11766
DSIntf.fldstGRAPHIC 11766
DSIntf.fldstGRAPHIC constant 11766
DSIntf.fldstGUID 11767
DSIntf.fldstGUID constant 11767
DSIntf.fldstHBINARY 11767
DSIntf.fldstHBINARY constant 11767
DSIntf.fldstHMEMO 11767
DSIntf.fldstHMEMO constant 11767
DSIntf.fldstMEMO 11768
DSIntf.fldstMEMO constant 11768
DSIntf.fldstMONEY 11768
DSIntf.fldstMONEY constant 11768
DSIntf.fldstOLEOBJ 11768
DSIntf.fldstOLEOBJ constant 11768
DSIntf.fldstORAINTERVAL 11769
DSIntf.fldstORAINTERVAL constant 11769
DSIntf.fldstORATIMESTAMP 11769
DSIntf.fldstORATIMESTAMP constant 11769
DSIntf.fldstPASSWORD 11769
DSIntf.fldstPASSWORD constant 11769
DSIntf.fldstReference 11770
DSIntf.fldstReference constant 11770
DSIntf.fldstTYPEDBINARY 11770
DSIntf.fldstTYPEDBINARY constant 11770
DSIntf.fldstWIDEMEMO 11770
DSIntf.fldstWIDEMEMO constant 11770
DSIntf.FldSubTypeMap 11771
DSIntf.FldSubTypeMap constant 11771
DSIntf.fldTABLE 11771
DSIntf.fldTABLE constant 11771
DSIntf.fldTIME 11771
DSIntf.fldTIME constant 11771
DSIntf.fldTIMESTAMP 11772
DSIntf.fldTIMESTAMP constant 11772
DSIntf.fldUINT16 11772
DSIntf.fldUINT16 constant 11772
DSIntf.fldUINT32 11772
DSIntf.fldUINT32 constant 11772
DSIntf.fldUINT64 11773
DSIntf.fldUINT64 constant 11773
DSIntf.fldUNICODE 11773
DSIntf.fldUNICODE constant 11773
DSIntf.fldUNKNOWN 11773
DSIntf.fldUNKNOWN constant 11773
DSIntf.fldVARBYTES 11774
DSIntf.fldVARBYTES constant 11774
DSIntf.fldWIDESTRING 11774
DSIntf.fldWIDESTRING constant 11774
DSIntf.fldZSTRING 11774
DSIntf.fldZSTRING constant 11774
DSIntf.FreeDataPacket 11703
DSIntf.FreeDataPacket function 11703
DSIntf.GROUPSTATE 11711
DSIntf.GROUPSTATE enumeration 11711
DSIntf.hDSAggregate 11715
DSIntf.hDSAggregate type 11715
DSIntf.hDSFilter 11715
DSIntf.hDSFilter type 11715

DSIntf.IDSBase 11673
DSIntf.IDSBase.AcceptChanges 11675
DSIntf.IDSBase.AddField 11675
DSIntf.IDSBase.AddOptParameter 11675
DSIntf.IDSBase.AppendData 11675
DSIntf.IDSBase.Clone 11676
DSIntf.IDSBase.Create 11676
DSIntf.IDSBase.CreateIndex 11676
DSIntf.IDSBase.DropOptParameter 11676
DSIntf.IDSBase.FldCmp 11677
DSIntf.IDSBase.GetDelta 11677
DSIntf.IDSBase.GetEmbeddedDS 11677
DSIntf.IDSBase.GetErrorString 11677
DSIntf.IDSBase.GetFieldDescs 11677
DSIntf.IDSBase.GetIndexDescs 11678
DSIntf.IDSBase.GetOptParameter 11678
DSIntf.IDSBase.GetProp 11678
DSIntf.IDSBase.GetProps 11678
DSIntf.IDSBase.MakeEmbedded 11679
DSIntf.IDSBase.PutBlank 11679
DSIntf.IDSBase.Reconcile 11679
DSIntf.IDSBase.Reconcile_MD 11679
DSIntf.IDSBase.Refresh 11680
DSIntf.IDSBase.RefreshRecords 11680
DSIntf.IDSBase.ReleaseBlobs 11680
DSIntf.IDSBase.RemoveIndex 11680
DSIntf.IDSBase.Reserved1 11681
DSIntf.IDSBase.Reserved2 11681
DSIntf.IDSBase.Reserved3 11681
DSIntf.IDSBase.Reset 11681
DSIntf.IDSBase.RollBack 11681
DSIntf.IDSBase.SetFieldCalculation 11682
DSIntf.IDSBase.SetProp 11682
DSIntf.IDSBase.StreamDS 11682
DSIntf.IDSCursor 11682
DSIntf.IDSCursor.AddAggregate 11685
DSIntf.IDSCursor.AddFilter 11685
DSIntf.IDSCursor.AddFilterCallBack 11685
DSIntf.IDSCursor.CloneCursor 11686
DSIntf.IDSCursor.CompareBookMarks 11686
DSIntf.IDSCursor.DeleteRecord 11686
DSIntf.IDSCursor.DropAggregate 11686
DSIntf.IDSCursor.DropFilter 11687
DSIntf.IDSCursor.DropRange 11687
DSIntf.IDSCursor.ExtractKey 11687
DSIntf.IDSCursor.GetAggregateDesc 11687
DSIntf.IDSCursor.GetAggregateValue 11687
DSIntf.IDSCursor.GetBlob 11688
DSIntf.IDSCursor.GetBlobLen 11688
DSIntf.IDSCursor.GetCurrentBookMark 11688
DSIntf.IDSCursor.GetCurrentRecord 11688
DSIntf.IDSCursor.GetCursorProps 11689
DSIntf.IDSCursor.GetField 11689
DSIntf.IDSCursor.GetFieldDescs 11689
DSIntf.IDSCursor.GetIndexDescs 11689
DSIntf.IDSCursor.GetProp 11690
DSIntf.IDSCursor.GetRecordAttribute 11690
DSIntf.IDSCursor.GetRecordBlock 11690
DSIntf.IDSCursor.GetRecordCount 11690
DSIntf.IDSCursor.GetRecordForKey 11690
DSIntf.IDSCursor.GetRecordNumber 11691
DSIntf.IDSCursor.GetRowRequestPacket 11691
DSIntf.IDSCursor.GetSequenceNumber 11691
DSIntf.IDSCursor.GetSubGroupState 11691
DSIntf.IDSCursor.InitCursor 11692
DSIntf.IDSCursor.InitRecord 11692
DSIntf.IDSCursor.InsertRecord 11692
DSIntf.IDSCursor.LinkCursors 11692
DSIntf.IDSCursor.LocateWithFilter 11693
DSIntf.IDSCursor.ModifyRecord 11693
DSIntf.IDSCursor.MoveRelative 11693
DSIntf.IDSCursor.MoveToBOF 11693
DSIntf.IDSCursor.MoveToBookMark 11693
DSIntf.IDSCursor.MoveToEOF 11694
DSIntf.IDSCursor.MoveToKey 11694
DSIntf.IDSCursor.MoveToNextSubGroup 11694
DSIntf.IDSCursor.MoveToSeqNo 11694
DSIntf.IDSCursor.PutBlob 11695

DSIntf.IDSCursor.PutField 11695
DSIntf.IDSCursor.RefreshRecord 11695
DSIntf.IDSCursor.ResyncWithMaster 11695
DSIntf.IDSCursor.RevertRecord 11695
DSIntf.IDSCursor.SetNotifyCallBack 11696
DSIntf.IDSCursor.SetProp 11696
DSIntf.IDSCursor.SetRange 11696
DSIntf.IDSCursor.SortOnFields 11696
DSIntf.IDSCursor.UndoLastChange 11697
DSIntf.IDSCursor.UseIndexOrder 11697
DSIntf.IDSCursor.VerifyField 11697
DSIntf.IDSWriter 11697
DSIntf.IDSWriter.AddAttribute 11698
DSIntf.IDSWriter.AddColumnDesc 11699
DSIntf.IDSWriter.EndOfNestedRows 11699
DSIntf.IDSWriter.GetColumnCount 11699
DSIntf.IDSWriter.GetColumnDescs 11699
DSIntf.IDSWriter.GetDataPacket 11700
DSIntf.IDSWriter.GetErrorString 11700
DSIntf.IDSWriter.GetSize 11700
DSIntf.IDSWriter.GetXMLMode 11700
DSIntf.IDSWriter.Init 11700
DSIntf.IDSWriter.Init_Sequential 11701
DSIntf.IDSWriter.PutField 11701
DSIntf.IDSWriter.Reset 11701
DSIntf.IDSWriter.SetXMLMode 11701
DSIntf.MAXKEYFIELDS 11775
DSIntf.MAXKEYFIELDS constant 11775
DSIntf.MAXLOGFLDTYPES 11775
DSIntf.MAXLOGFLDTYPES constant 11775
DSIntf.mdALLOWLINKCHANGE 11775
DSIntf.mdALLOWLINKCHANGE constant 11775
DSIntf.mdCASCADEDEL 11776
DSIntf.mdCASCADEDEL constant 11776
DSIntf.mdCASCADEMOD 11776
DSIntf.mdCASCADEMOD constant 11776
DSIntf.MIDASNAME 11716
DSIntf.MIDASNAME type 11716
DSIntf.MIDASPATH 11716
DSIntf.MIDASPATH type 11716
DSIntf.PACKETVERSION_1 11776
DSIntf.PACKETVERSION_1 constant 11776
DSIntf.PACKETVERSION_2 11777
DSIntf.PACKETVERSION_2 constant 11777
DSIntf.PACKETVERSION_3 11777
DSIntf.PACKETVERSION_3 constant 11777
DSIntf.pDSAttr 11716
DSIntf.pDSAttr type 11716
DSIntf.pDSBOOKMRK 11717
DSIntf.pDSBOOKMRK type 11717
DSIntf.pdsCBRType 11717
DSIntf.pdsCBRType type 11717
DSIntf.PDSDataPacketFldDesc 11717
DSIntf.PDSDataPacketFldDesc type 11717
DSIntf.pDSFLDDesc 11718
DSIntf.pDSFLDDesc type 11718
DSIntf.pDSIDXDesc 11718
DSIntf.pDSIDXDesc type 11718
DSIntf.pDSProps 11718
DSIntf.pDSProps type 11718
DSIntf.pfCHANGECallBack 11719
DSIntf.pfCHANGECallBack type 11719
DSIntf.pfDSCalcField 11719
DSIntf.pfDSCalcField type 11719
DSIntf.pfDSFilter 11719
DSIntf.pfDSFilter type 11719
DSIntf.pfDSReconcile 11720
DSIntf.pfDSReconcile type 11720
DSIntf.pfDSReconcile_MD 11720
DSIntf.pfDSReconcile_MD type 11720
DSIntf.pfFLDComp 11720
DSIntf.pfFLDComp type 11720
DSIntf.phDSAggregate 11721
DSIntf.phDSAggregate type 11721
DSIntf.phDSFilter 11721
DSIntf.phDSFilter type 11721
DSIntf.PLongBool 11721
DSIntf.PLongBool type 11721

DSIntf.pSAVEPOINT 11722
DSIntf.pSAVEPOINT type 11722
DSIntf.RegisterMidasLib 11703
DSIntf.RegisterMidasLib function 11703
DSIntf.SAVEPOINT 11722
DSIntf.SAVEPOINT type 11722
DSIntf.StringToVariantArray 11704
DSIntf.StringToVariantArray function 11704
DSIntf.szAUTOINCVALUE 11777
DSIntf.szAUTOINCVALUE constant 11777
DSIntf.szBDEDEFCL 11778
DSIntf.szBDEDEFCL constant 11778
DSIntf.szBDEDEFX 11778
DSIntf.szBDEDEFX constant 11778
DSIntf.szBDEDOMCL 11778
DSIntf.szBDEDOMCL constant 11778
DSIntf.szBDEDOMX 11779
DSIntf.szBDEDOMX constant 11779
DSIntf.szBDERECCL 11779
DSIntf.szBDERECCL constant 11779
DSIntf.szBDERECX 11779
DSIntf.szBDERECX constant 11779
DSIntf.szCHANGE_LOG 11780
DSIntf.szCHANGE_LOG constant 11780
DSIntf.szCHANGEINDEX 11780
DSIntf.szCHANGEINDEX constant 11780
DSIntf.szCONSTRAINTS 11780
DSIntf.szCONSTRAINTS constant 11780
DSIntf.szDATASET_CONTEXT 11781
DSIntf.szDATASET_CONTEXT constant 11781
DSIntf.szDATASET_DELTA 11781
DSIntf.szDATASET_DELTA constant 11781
DSIntf.szDECIMALS 11781
DSIntf.szDECIMALS constant 11781
DSIntf.szDEFAULT_ORDER 11782
DSIntf.szDEFAULT_ORDER constant 11782
DSIntf.szDISABLE_DELETES 11782
DSIntf.szDISABLE_DELETES constant 11782
DSIntf.szDISABLE_EDITS 11782
DSIntf.szDISABLE_EDITS constant 11782
DSIntf.szDISABLE_INSERTS 11783
DSIntf.szDISABLE_INSERTS constant 11783
DSIntf.szELEMENTS 11783
DSIntf.szELEMENTS constant 11783
DSIntf.szFIELDNAME 11783
DSIntf.szFIELDNAME constant 11783
DSIntf.szFIELDPROPS 11784
DSIntf.szFIELDPROPS constant 11784
DSIntf.szLCID 11784
DSIntf.szLCID constant 11784
DSIntf.szMAXVALUE 11784
DSIntf.szMAXVALUE constant 11784
DSIntf.szMD_FIELDLINKS 11785
DSIntf.szMD_FIELDLINKS constant 11785
DSIntf.szMD_SEMANTICS 11785
DSIntf.szMD_SEMANTICS constant 11785
DSIntf.szMINVALUE 11785
DSIntf.szMINVALUE constant 11785
DSIntf.szNO_RESET_CALL 11786
DSIntf.szNO_RESET_CALL constant 11786
DSIntf.szORIGIN 11786
DSIntf.szORIGIN constant 11786
DSIntf.szPRIMARY_KEY 11786
DSIntf.szPRIMARY_KEY constant 11786
DSIntf.szPROVFLAGS 11787
DSIntf.szPROVFLAGS constant 11787
DSIntf.szREADONLY 11787
DSIntf.szREADONLY constant 11787
DSIntf.szSERVER_COL 11787
DSIntf.szSERVER_COL constant 11787
DSIntf.szSERVERCALC 11788
DSIntf.szSERVERCALC constant 11788
DSIntf.szstACCOLEOBJ 11788
DSIntf.szstACCOLEOBJ constant 11788
DSIntf.szstADTNESTEDTABLE 11788
DSIntf.szstADTNESTEDTABLE constant 11788
DSIntf.szstAUTOINC 11789
DSIntf.szstAUTOINC constant 11789

DSIntf.szstBINARY 11789
DSIntf.szstBINARY constant 11789
DSIntf.szstDBSOLEOBJ 11789
DSIntf.szstDBSOLEOBJ constant 11789
DSIntf.szstFIXEDCHAR 11790
DSIntf.szstFIXEDCHAR constant 11790
DSIntf.szstFMTMEMO 11790
DSIntf.szstFMTMEMO constant 11790
DSIntf.szstGRAPHIC 11790
DSIntf.szstGRAPHIC constant 11790
DSIntf.szstGUID 11791
DSIntf.szstGUID constant 11791
DSIntf.szstHBINARY 11791
DSIntf.szstHBINARY constant 11791
DSIntf.szstHMEMO 11791
DSIntf.szstHMEMO constant 11791
DSIntf.szstMEMO 11792
DSIntf.szstMEMO constant 11792
DSIntf.szstMONEY 11792
DSIntf.szstMONEY constant 11792
DSIntf.szstOLEOBJ 11792
DSIntf.szstOLEOBJ constant 11792
DSIntf.szstREFNESTEDTABLE 11793
DSIntf.szstREFNESTEDTABLE constant 11793
DSIntf.szstTYPEDBINARY 11793
DSIntf.szstTYPEDBINARY constant 11793
DSIntf.szstWideMEMO 11793
DSIntf.szstWideMEMO constant 11793
DSIntf.szSUBTYPE 11794
DSIntf.szSUBTYPE constant 11794
DSIntf.szTABLE_NAME 11794
DSIntf.szTABLE_NAME constant 11794
DSIntf.szTYPENAME 11794
DSIntf.szTYPENAME constant 11794
DSIntf.szUNIQUE_KEY 11795
DSIntf.szUNIQUE_KEY constant 11795
DSIntf.szUPDATEMODE 11795
DSIntf.szUPDATEMODE constant 11795
DSIntf.szWIDTH 11795
DSIntf.szWIDTH constant 11795
DSIntf.TDataPacket 11722
DSIntf.TDataPacket type 11722
DSIntf.TDSDataPacketFldDesc 11711
DSIntf.TDSDataPacketFldDesc record 11711
DSIntf.TPcktAttrArea 11712
DSIntf.TPcktAttrArea enumeration 11712
DSIntf.TPcktFldStatus 11712
DSIntf.TPcktFldStatus enumeration 11712
DSIntf.VariantArrayToString 11704
DSIntf.VariantArrayToString function 11704
DSIntf.VarToDataPacket 11704
DSIntf.VarToDataPacket function 11704
DSIntf.xmlOFF 11796
DSIntf.xmlOFF constant 11796
DSIntf.xmlON 11796
DSIntf.xmlON constant 11796
DSIntf.xmlUNTYPED 11796
DSIntf.xmlUNTYPED constant 11796
DSIntf.xmlUTF8 11797
DSIntf.xmlUTF8 constant 11797
DSIntf.xmlXMLDATATYPED 11797
DSIntf.xmlXMLDATATYPED constant 11797
DSIntf.xmlXMLSCHEMA 11797
DSIntf.xmlXMLSCHEMA constant 11797
DSIntf.xmlXMLUTF8 11798
DSIntf.xmlXMLUTF8 constant 11798
dsIUnKnownAvailable enumeration member 12097
dsLoadingMIMEHandler enumeration member 12097
dsMetaData enumeration member 17838
dsMIMETypeAvailable enumeration member 12097
dsNewValue enumeration member 7608
dsNoData enumeration member 17838
dsNone enumeration member 17673
dsNormal enumeration member 5468, 13265, 15394
dsOldValue enumeration member 7608
dsOpenAfter enumeration member 17673
dsOpenBefore enumeration member 17673
dsOpening enumeration member 7608

dsP3PHeader enumeration member 12097
dsPersistentCookieReceived enumeration member 12097
dsPolicyHREF enumeration member 12097
DSProd 11799
DSProd namespace 11799
Classes 11800
DSProd.TCustomDataSetPageProducer 11800
DSProd.TCustomDataSetPageProducer.Content 11804
DSProd.TCustomDataSetPageProducer.DataSet 11806
DSProd.TCustomDataSetPageProducer.DoTagEvent 11805
DSProd.TCustomDataSetPageProducer.GetDataSet 11805
DSProd.TCustomDataSetPageProducer.Notification 11805
DSProd.TCustomDataSetPageProducer.SetDataSet 11806
DSProd.TDataSetPageProducer 11807
DSProd.TDataSetPageProducer.DataSet 11811
DSProd.TDataSetPageProducer.HTMLDoc 11812
DSProd.TDataSetPageProducer.HTMLFile 11812
DSProd.TDataSetPageProducer.OnHTMLTag 11814
DSProd.TDataSetPageProducer.ScriptEngine 11813
dspropAUTOINC_DISABLED enumeration member 11709
dspropCHANGEINDEX_VIEW enumeration member 11709
dspropCOMPRESSARRAYS enumeration member 11709
dspropCONSTRAINTS_DISABLED enumeration member 11709
dspropDATAHASCHANGED enumeration member 11709
dspropDISABLEDELETES enumeration member 11709
dspropDISABLEEDITS enumeration member 11709
dspropDISABLEINSERTS enumeration member 11709
dspropDISABLESTRINGTRIM enumeration member 11709
dspropDONTINCLMETADATA enumeration member 11709
dspropDSISPARTIAL enumeration member 11709
dspropFIELD_FULLNAME enumeration member 11709
dspropFIELDID_FORNAME enumeration member 11709
dspropFIELDID_FORPARENT enumeration member 11709
dspropGETSAVEPOINT enumeration member 11709
dspropGETUNIQUEINDEX enumeration member 11709
dspropINCLBLOBSINDELTA enumeration member 11709
dspropISDELTA enumeration member 11709
dspropLOGCHANGES enumeration member 11709
dspropMD_SEMANTICS enumeration member 11709
dspropNOOFCHANGES enumeration member 11709
dspropREADONLY enumeration member 11709
dspropRECORDSINDS enumeration member 11709
dspropREMOTE_UPDATEMODE enumeration member 11709
dspropXML_STREAMMODE enumeration member 11709
dsProtocolClassID enumeration member 12097
dsProxyDetecting enumeration member 12097
dsRawMIMEType enumeration member 12097
dsRedirecting enumeration member 12097
dsRowCaption enumeration member 17673
dsRowIndicator enumeration member 17673
dsRowPlus enumeration member 17673
dsRowValue enumeration member 17673
dsSelected enumeration member 15394
dsSendingRequest enumeration member 12097
dsSessionCookieReceived enumeration member 12097
dsSessionCookiesAllowed enumeration member 12097
dsSetKey enumeration member 7608
dsSum enumeration member 17673
dsTransparent enumeration member 15394
dsUploadingData enumeration member 12097
dsUsingCachedCopy enumeration member 12097
dsVerifiedMIMETypeAvailable enumeration member 12097
dtaLeft enumeration member 5447
dtaRight enumeration member 5447
dtAutoSelect enumeration member 17231
dtAVIVideo enumeration member 17231
dtCDAudio enumeration member 17231
dtCDROM enumeration member 13620
dtControl enumeration member 5446
dtDAT enumeration member 17231
dtDigitalVideo enumeration member 17231
dtFixed enumeration member 13620
dtFloppy enumeration member 13620
dtItem enumeration member 5446
dtkDate enumeration member 5447
dtkTime enumeration member 5447
dtMMMovie enumeration member 17231
dtNetwork enumeration member 13620

dtNoDrive enumeration member 13620
dtOther enumeration member 17231
dtOverlay enumeration member 17231
dtRAM enumeration member 13620
dtScanner enumeration member 17231
dtSequencer enumeration member 17231
dtSubItem enumeration member 5446
dtUnknown enumeration member 13620
dtVCR enumeration member 17231
dtVideodisc enumeration member 17231
dtWaveAudio enumeration member 17231
dupAccept enumeration member 3271, 17387
dupError enumeration member 3271, 17387
dupIgnore enumeration member 3271, 17387
DWebBrowserEvents interface 19281
about DWebBrowserEvents interface 19281
BeforeNavigate 19282
CommandStateChange 19282
DownloadBegin 19282
DownloadComplete 19283
DWebBrowserEvents Members 19281
DWebBrowserEvents methods 19281
FrameBeforeNavigate 19283
FrameNavigateComplete 19283
FrameNewWindow 19283
NavigateComplete 19284
NewWindow 19284
ProgressChange 19284
PropertyChange 19284
Quit 19284
StatusTextChange 19285
TitleChange 19285
WindowActivate 19285
WindowMove 19285
WindowResize 19286
DWebBrowserEvents2 interface 19286
about DWebBrowserEvents2 interface 19286
BeforeNavigate2 19288
ClientToHostWindow 19288
CommandStateChange 19288
DocumentComplete 19288
DownloadBegin 19289
DownloadComplete 19289
DWebBrowserEvents2 Members 19286
DWebBrowserEvents2 methods 19287
FileDownload 19289
NavigateComplete2 19289
NavigateError 19289
NewWindow2 19290
OnFullScreen 19290
OnMenuBar 19290
OnQuit 19290
OnStatusBar 19291
OnTheaterMode 19291
OnToolBar 19291
OnVisible 19291
PrintTemplateInstantiation 19291
PrintTemplateTeardown 19292
PrivacyImpactedStateChange 19292
ProgressChange 19292
PropertyChange 19292
SetSecureLockIcon 19293
StatusTextChange 19293
TitleChange 19293
UpdatePageStatus 19293
WindowClosing 19293
WindowSetHeight 19294
WindowSetLeft 19294
WindowSetResizable 19294
WindowSetTop 19294
WindowSetWidth 19295
DynamicArray template 22172
[] 22179
[] operator 22179
~DynamicArray 22174
= 22180
== 22180
about DynamicArray template 22172

AllocData 22176
Copy 22176
Copy method 22176
CopyRange 22177
CopyRange method 22176
DecRefCount 22177
DimCount 22177
DynamicArray 22175
DynamicArray constructor 22175
DynamicArray Members 22174
DynamicArray methods 22175
DynamicArray operators 22179
DynamicArray properties 22180
FreeData 22178
get_high 22178
get_length 22178
get_low 22178
High 22181
IncRefCount 22178
Length 22181
Low 22182
set_length 22179
SetData 22179
DynArrayException class 22182
about DynArrayException class 22182
DynArrayNullData class 22183
about DynArrayNullData class 22183
DynArrayOutOfRange class 22183
about DynArrayOutOfRange class 22183
DynArrayOutOfRange 22184
DynArrayOutOfRange data members 22184
DynArrayOutOfRange Members 22183
m_Count 22184
m_Index 22184

E

EAbort class 22722
about EAbort class 22722
EAbstractError class 22722
about EAbstractError class 22722
EAccessViolation class 22723
about EAccessViolation class 22723
eaContinue enumeration member 17839
eaCopy enumeration member 11234
eaCut enumeration member 11234
EAdapterDispatcherException class 24838
about EAdapterDispatcherException class 24838
EAdapterException class 23696
about EAdapterException class 23696
CreateID 23697
EAdapterException Members 23696
EAdapterException properties 23698
ID 23698
EAdapterFieldException class 23698
about EAdapterFieldException class 23698
Create 23700
EAdapterFieldException Members 23699
EAdapterFieldException properties 23700
FieldName 23700
EAdapterRequestException class 863
about EAdapterRequestException class 863
eaDelete enumeration member 11234
EADOError class 978
about EADOError class 978
eaElide enumeration member 11234
eaFail enumeration member 17839
eaPaste enumeration member 11234
EAppletException class 6497
about EAppletException class 6497
eaPrint enumeration member 11234
eaRedo enumeration member 11234
EArrayError class 17305, 17441
about EArrayError class 17305, 17441
eaSelectAll enumeration member 11234
EAssertionFailed class 22723
about EAssertionFailed class 22723
eaUndo enumeration member 11234
ebBottom enumeration member 23289

EBcdException class 13630
about EBcdException class 13630
EBcdOverflowException class 13630
about EBcdOverflowException class 13630
EBitsError class 2693
about EBitsError class 2693
ebLeft enumeration member 23289
ebRight enumeration member 23289
EBrokerException class 17866
about EBrokerException class 17866
ebTop enumeration member 23289
ECacheError class 17747
about ECacheError class 17747
EClassNotFound class 2694
about EClassNotFound class 2694
ecLowerCase enumeration member 21946
ecNormal enumeration member 21946
ECommonCalendarError class 3420
about ECommonCalendarError class 3420
EComponentError class 2694
about EComponentError class 2694
ECompressionError class 26617
about ECompressionError class 26617
eConnect enumeration member 20364
EControlC class 22724
about EControlC class 22724
EConversionError class 6381
about EConversionError class 6381
EConvertError class 22724
about EConvertError class 22724
ECorbaDispatch class 6423
about ECorbaDispatch class 6423
ECorbaException class 6423
about ECorbaException class 6423
ECorbaException Members 6424
ECorbaException methods 6425
ECorbaException properties 6425
GetMessage 6425
Name 6425
ECorbaUserException class 6426
about ECorbaUserException class 6426
Copy 6430
Create 6427
ECorbaUserException Members 6426
ECorbaUserException methods 6430
ECorbaUserException properties 6430
Proxy 6430
Throw 6430
ecUpperCase enumeration member 21946
EDatabaseError class 6731
about EDatabaseError class 6731
EDateTimeError class 3421
about EDateTimeError class 3421
EDBClient class 8263
about EDBClient class 8263
Create 8264
EDBClient Members 8263
EDBClient properties 8265
ErrorCode 8265
EDBEditError class 15798
about EDBEditError class 15798
EDBEngineError class 9854
about EDBEngineError class 9854
Create 9856
Destroy 9856
EDBEngineError Members 9855
EDBEngineError properties 9857
ErrorCount 9857
Errors 9857
EDecompressionError class 26617
about EDecompressionError class 26617
EDimensionMapError class 17397
about EDimensionMapError class 17397
EDimIndexError class 17441
about EDimIndexError class 17441
eDisconnect enumeration member 20364
EDivByZero class 22726
about EDivByZero class 22726

EDOMParseError class 26258
about EDOMParseError class 26258
Create 26259
EDOMParseError Members 26258
EDOMParseError properties 26260
ErrorCode 26260
FilePos 26260
Line 26261
LinePos 26261
ParseError 26261
Reason 26262
SrcText 26262
URL 26262
EDSWriter class 18421
about EDSWriter class 18421
Create 18423
EDSWriter Members 18422
EDSWriter properties 18423
ErrorCode 18423
EDynArrayRangeError class 17306
about EDynArrayRangeError class 17306
eeAccept enumeration member 18886
eeConnect enumeration member 18886
eeDisconnect enumeration member 18886
eeGeneral enumeration member 18886
eeLookup enumeration member 18886
eeReceive enumeration member 18886
eeSend enumeration member 18886
EExternal class 22727
about EExternal class 22727
EExternal fields 22729
EExternal Members 22728
ExceptionRecord 22729
EExternalException class 22729
about EExternalException class 22729
EFCreateError class 2695
about EFCreateError class 2695
EFilerError class 2696
about EFilerError class 2696
EFileStreamError class 2697
about EFileStreamError class 2697
Create 2698
EFileStreamError Members 2697
EFOpenError class 2699
about EFOpenError class 2699
EGridPanelException class 12105
about EGridPanelException class 12105
EHeapException class 22730
about EHeapException class 22730
EHeapException Members 22731
EHeapException methods 22732
FreeInstance 22732
EHelpSystemException class 15012
about EHelpSystemException class 15012
EIniFileException class 15434
about EIniFileException class 15434
EInOutError class 22732
about EInOutError class 22732
EInOutError fields 22735
EInOutError Members 22734
ErrorCode 22735
EInterfaceRTTIException class 15479
about EInterfaceRTTIException class 15479
EInterpreterError class 18897
about EInterpreterError class 18897
EIntError class 22735
about EIntError class 22735
EIntfCastError class 22736
about EIntfCastError class 22736
EIntOverflow class 22736
about EIntOverflow class 22736
EInvalidArgument class 15927
about EInvalidArgument class 15927
EInvalidCast class 22737
about EInvalidCast class 22737
EInvalidContainer class 22738
about EInvalidContainer class 22738
EInvalidGraphic class 14395

about EInvalidGraphic class 14395
EInvalidGraphicOperation class 14396
about EInvalidGraphicOperation class 14396
EInvalidGridOperation class 14767
about EInvalidGridOperation class 14767
EInvalidImage class 2699
about EInvalidImage class 2699
EInvalidInsert class 22738
about EInvalidInsert class 22738
EInvalidOp class 22738
about EInvalidOp class 22738
EInvalidOperation class 2700
about EInvalidOperation class 2700
EInvalidPointer class 22739
about EInvalidPointer class 22739
EListError class 2701
about EListError class 2701
ELowCapacityError class 17306
about ELowCapacityError class 17306
emAddColumns enumeration member 13268
emAddRows enumeration member 13268
EMaskException class 15898
about EMaskException class 15898
EMathError class 22739
about EMathError class 22739
EMCIDeviceError class 17167
about EMCIDeviceError class 17167
EMenuError class 16055
about EMenuError class 16055
EMethodNotFound class 2701
about EMethodNotFound class 2701
emFixedSize enumeration member 13268
EMonthCalError class 3421
about EMonthCalError class 3421
ENamedVariantsError class 24808
about ENamedVariantsError class 24808
ENoResultSet class 9858
about ENoResultSet class 9858
eoAsyncExecute enumeration member 1299
eoAsyncFetch enumeration member 1299
eoAsyncFetchNonBlocking enumeration member 1299
eoExecuteNoRecords enumeration member 1299
EOleCtrlError class 18023
about EOleCtrlError class 18023
EOleError class 5528, 17889
about EOleError class 5528, 17889
EOleException class 5529, 17889
about EOleException class 5529, 17889
Create 5530, 17891
EOleException Members 5529, 17890
EOleException properties 5531, 17891
ErrorCode 17891
HelpFile 5531, 17891
Source 5531, 17892
EOleRegistrationError class 5531
about EOleRegistrationError class 5531
EOleSysError class 5532, 17892
about EOleSysError class 5532, 17892
Create 5533, 17893
EOleSysError Members 5532, 17892
EOleSysError properties 5534, 17893
ErrorCode 5534, 17894
EOSError class 22740
about EOSError class 22740
EOSError fields 22742
EOSError Members 22741
ErrorCode 22742
EOutlineError class 18233
about EOutlineError class 18233
EOutOfMemory class 22742
about EOutOfMemory class 22742
EOutOfResources class 2702
about EOutOfResources class 2702
EOverflow class 22743
about EOverflow class 22743
EPackageError class 22744
about EPackageError class 22744
EPageDispatcherException class 24839

about EPageDispatcherException class 24839
EParserError class 2702
about EParserError class 2702
epEndEllipsis enumeration member 21947
EPlatformVersionException class 11269
about EPlatformVersionException class 11269
epNone enumeration member 21947
epPathEllipsis enumeration member 21947
EPrinter class 18392
about EPrinter class 18392
EPrivilege class 22744
about EPrivilege class 22744
EPropertyConvertError class 23331
about EPropertyConvertError class 23331
EPropertyError class 23331
about EPropertyError class 23331
EPropReadOnly class 22745
about EPropReadOnly class 22745
EPropWriteOnly class 22745
about EPropWriteOnly class 22745
epSelectedLeft enumeration member 23229
epSelectedRight enumeration member 23229
epUnselectedLeft enumeration member 23229
epUnselectedRight enumeration member 23229
epWordEllipsis enumeration member 21947
erAddNew enumeration member 1296
ERangeError class 22745
about ERangeError class 22745
erClose enumeration member 1296
erDelete enumeration member 1296
EReadError class 2703
about EReadError class 2703
EReconcileError class 8265
about EReconcileError class 8265
Context 8267
Create 8267
EReconcileError Members 8266
EReconcileError properties 8267
PreviousError 8268
eReconnect enumeration member 20364
ERecursiveIncludeFile class 25465
about ERecursiveIncludeFile class 25465
ERegistryException class 18634
about ERegistryException class 18634
ERemotableException class 15496
about ERemotableException class 15496
Create 15498
Create constructor 15498
ERemotableException Members 15497
ERemotableException properties 15499
FaultActor 15499
FaultCode 15499
FaultDetail 15500
EResNotFound class 2704
about EResNotFound class 2704
erFirstChange enumeration member 1296
erMove enumeration member 1296
erMoveFirst enumeration member 1296
erMoveLast enumeration member 1296
erMoveNext enumeration member 1296
erMovePrevious enumeration member 1296
erRequery enumeration member 1296
erResynch enumeration member 1296
erUndoAddNew enumeration member 1296
erUndoDelete enumeration member 1296
erUndoUpdate enumeration member 1296
erUpdate enumeration member 1296
esAddIn enumeration member 11822
ESafecallException class 22747
about ESafecallException class 22747
esCancel enumeration member 1298
esCanCopy enumeration member 11235
esCanCreateTemplate enumeration member 11235
esCanCut enumeration member 11235
esCanDelete enumeration member 11235
esCanEditOle enumeration member 11235
esCanElide enumeration member 11235
esCanPaste enumeration member 11235

esCanPrint enumeration member 11235
esCanRedo enumeration member 11235
esCanSelectAll enumeration member 11235
esCantDeny enumeration member 1298
esCanUndo enumeration member 11235
EScriptError class 15255
about EScriptError class 15255
Content 15258
Create 15257
Destroy 15257
Errors 15258
EScriptError Members 15255
EScriptError properties 15257
EScriptEvalError class 25466
about EScriptEvalError class 25466
esCritical enumeration member 22048
esEllipsis enumeration member 14997
esErrorsOccured enumeration member 1298
ESessionException class 24840
about ESessionException class 24840
ESessionsException class 19012
about ESessionsException class 19012
esFail enumeration member 969
esForm enumeration member 11822
esIgnore enumeration member 22048
esLowered enumeration member 23290
esNone enumeration member 23290
esNormal enumeration member 22048
ESOAPDomConvertError class 18137
about ESOAPDomConvertError class 18137
ESOAPHTTPException class 19958
about ESOAPHTTPException class 19958
Create 19959
CreateFmt 19960
ESOAPHTTPException Members 19958
ESOAPHTTPException properties 19960
StatusCode 19961
ESocketConnectionError class 18898
about ESocketConnectionError class 18898
ESocketError class 18739, 20041
about ESocketError class 18739, 20041
esOK enumeration member 1298
esOther enumeration member 969
esPickList enumeration member 14997
esProject enumeration member 11822
esRaised enumeration member 23290
esSevere enumeration member 22048
esSimple enumeration member 14997
esStandard enumeration member 11822
esSuccess enumeration member 969
EStackOverflow class 22747
about EStackOverflow class 22747
esTerminate enumeration member 25601
esTimeout enumeration member 25601
EStreamError class 2704
about EStreamError class 2704
EStringListError class 2705
about EStringListError class 2705
esUndefined enumeration member 969
esUnwantedEvent enumeration member 1298
ETabDockException class 11594
about ETabDockException class 11594
etFirstIsSel enumeration member 23229
etFirstNotSel enumeration member 23229
EThread class 2706
about EThread class 2706
etLastIsSel enumeration member 23229
etLastNotSel enumeration member 23229
etNone enumeration member 23229
etNotSelToNotSel enumeration member 23229
etNotSelToSel enumeration member 23229
ETreeViewError class 3422
about ETreeViewError class 3422
etSelToNotSel enumeration member 23229
ETypeRegistryException class 15500
about ETypeRegistryException class 15500
ETypeTransException class 23320
about ETypeTransException class 23320

EUnderflow class 22748
about EUnderflow class 22748
EUnsupportedTypeError class 17307
about EUnsupportedTypeError class 17307
EUpdateError class 6732
about EUpdateError class 6732
Context 6734
Create 6733
Destroy 6734
ErrorCode 6735
EUpdateError Members 6732
EUpdateError properties 6734
OriginalException 6735
PreviousError 6736
eUserDataSet enumeration member 20365
EUserIDNotFoundException class 25604
about EUserIDNotFoundException class 25604
eUserMonitor enumeration member 20365
EValidateUserException class 25604
about EValidateUserException class 25604
EVariantArrayCreateError class 23572
about EVariantArrayCreateError class 23572
EVariantArrayLockedError class 23573
about EVariantArrayLockedError class 23573
EVariantBadIndexError class 23574
about EVariantBadIndexError class 23574
EVariantBadVarTypeError class 23574
about EVariantBadVarTypeError class 23574
EVariantDispatchError class 23575
about EVariantDispatchError class 23575
EVariantError class 22748
about EVariantError class 22748
EVariantInvalidArgError class 23575
about EVariantInvalidArgError class 23575
EVariantInvalidNullOpError class 23576
about EVariantInvalidNullOpError class 23576
EVariantInvalidOpError class 23577
about EVariantInvalidOpError class 23577
EVariantNotAnArrayError class 23577
about EVariantNotAnArrayError class 23577
EVariantNotImplError class 23578
about EVariantNotImplError class 23578
EVariantOutOfMemoryError class 23578
about EVariantOutOfMemoryError class 23578
EVariantOverflowError class 23579
about EVariantOverflowError class 23579
EVariantRangeCheckError class 23580
about EVariantRangeCheckError class 23580
EVariantTypeCastError class 23580
about EVariantTypeCastError class 23580
EVariantUnexpectedError class 23581
about EVariantUnexpectedError class 23581
EWebBrokerException class 15038
about EWebBrokerException class 15038
EWebModuleException class 25327
about EWebModuleException class 25327
EWebUsersListException class 25605
about EWebUsersListException class 25605
EWin32Error class 22749
about EWin32Error class 22749
EWriteError class 2706
about EWriteError class 2706
EWSDLLoadException class 26006
about EWSDLLoadException class 26006
exceptCommand enumeration member 20367
exceptConnection enumeration member 20367
exceptCursor enumeration member 20367
Exception class 22750
about Exception class 22750
Create 22752
CreateFmt 22754
CreateFmtHelp 22755
CreateHelp 22755
CreateRes 22756
CreateRes constructor 22756
CreateResFmt 22757
CreateResFmt constructor 22756
CreateResFmtHelp 22758

CreateResFmtHelp constructor 22757
CreateResHelp 22759
CreateResHelp constructor 22759
Exception Members 22751
Exception properties 22760
HelpContext 22760
Message 22760
exceptMetaData enumeration member 20367
exceptUseLast enumeration member 20367
exDenormalized enumeration member 15981
exInvalidOp enumeration member 15981
EXMLDocError class 26381
about EXMLDocError class 26381
exOverflow enumeration member 15981
exPrecision enumeration member 15981
ExptIntf 11815
ExptIntf namespace 11815
Classes 11817
Constants 11826
Functions 11821
Structs, Records, Enums 11822
Types 11823
Variables 11825
ExptIntf.ExpertEntryPoint 11826
ExptIntf.ExpertEntryPoint constant 11826
ExptIntf.isExperts 11826
ExptIntf.isExperts constant 11826
ExptIntf.LibraryExpertProc 11825
ExptIntf.LibraryExpertProc variable 11825
ExptIntf.RegisterLibraryExpert 11821
ExptIntf.RegisterLibraryExpert function 11821
ExptIntf.TExpertInitProc 11823
ExptIntf.TExpertInitProc type 11823
ExptIntf.TExpertRegisterProc 11823
ExptIntf.TExpertRegisterProc type 11823
ExptIntf.TExpertState 11824
ExptIntf.TExpertState type 11824
ExptIntf.TExpertStyle 11822
ExptIntf.TExpertStyle enumeration 11822
ExptIntf.TExpertTerminateProc 11824
ExptIntf.TExpertTerminateProc type 11824
ExptIntf.TIExpert 11817
ExptIntf.TIExpert.Execute 11818
ExptIntf.TIExpert.GetAuthor 11818
ExptIntf.TIExpert.GetComment 11818
ExptIntf.TIExpert.GetGlyph 11818
ExptIntf.TIExpert.GetIDString 11819
ExptIntf.TIExpert.GetMenuText 11819
ExptIntf.TIExpert.GetName 11819
ExptIntf.TIExpert.GetPage 11819
ExptIntf.TIExpert.GetState 11820
ExptIntf.TIExpert.GetStyle 11820
ExptIntf.ToolServices 11825
ExptIntf.ToolServices variable 11825
ExptIntf.ValidExpertVersion 11826
ExptIntf.ValidExpertVersion constant 11826
EXSDateTimeException class 26513
about EXSDateTimeException class 26513
EXSDecimalException class 26514
about EXSDecimalException class 26514
EXSHexBinaryException class 26514
about EXSHexBinaryException class 26514
EXSLPageProducerError class 26588
about EXSLPageProducerError class 26588
ExtActns 11829
ExtActns namespace 11829
Classes 11831
Structs, Records, Enums 12096
Types 12099
ExtActns.TBrowseURL 11832
ExtActns.TBrowseURL.AfterBrowse 11842
ExtActns.TBrowseURL.BeforeBrowse 11842
ExtActns.TBrowseURL.Caption 11837
ExtActns.TBrowseURL.Enabled 11838
ExtActns.TBrowseURL.ExecuteTarget 11836
ExtActns.TBrowseURL.HelpContext 11838
ExtActns.TBrowseURL.HelpKeyword 11838
ExtActns.TBrowseURL.HelpType 11839

ExtActns.TBrowseURL.Hint 11839
ExtActns.TBrowseURL.ImageIndex 11840
ExtActns.TBrowseURL.OnHint 11842
ExtActns.TBrowseURL.OnUpdate 11843
ExtActns.TBrowseURL.SecondaryShortCuts 11840
ExtActns.TBrowseURL.ShortCut 11840
ExtActns.TBrowseURL.URL 11841
ExtActns.TBrowseURL.Visible 11841
ExtActns.TCustomFileRun 11844
ExtActns.TCustomFileRun.Browse 11852
ExtActns.TCustomFileRun.BrowseDlg 11853
ExtActns.TCustomFileRun.Create 11849
ExtActns.TCustomFileRun.Directory 11853
ExtActns.TCustomFileRun.ExecuteTarget 11849
ExtActns.TCustomFileRun.FileName 11853
ExtActns.TCustomFileRun.HandlesTarget 11850
ExtActns.TCustomFileRun.HInst 11854
ExtActns.TCustomFileRun.Notification 11850
ExtActns.TCustomFileRun.Operation 11855
ExtActns.TCustomFileRun.Parameters 11855
ExtActns.TCustomFileRun.ParentControl 11856
ExtActns.TCustomFileRun.SetupBrowseDialog 11851
ExtActns.TCustomFileRun.ShowCmd 11856
ExtActns.TCustomFileRun.UpdateTarget 11851
ExtActns.TDownloadProgressEvent 12099
ExtActns.TDownloadProgressEvent type 12099
ExtActns.TDownLoadURL 11857
ExtActns.TDownLoadURL.AfterDownload 11870
ExtActns.TDownLoadURL.BeforeDownload 11870
ExtActns.TDownLoadURL.Caption 11865
ExtActns.TDownLoadURL.Enabled 11865
ExtActns.TDownLoadURL.ExecuteTarget 11862
ExtActns.TDownLoadURL.Filename 11866
ExtActns.TDownLoadURL.GetBindInfo 11863
ExtActns.TDownLoadURL.GetPriority 11863
ExtActns.TDownLoadURL.HelpContext 11866
ExtActns.TDownLoadURL.HelpKeyword 11866
ExtActns.TDownLoadURL.HelpType 11867
ExtActns.TDownLoadURL.Hint 11867
ExtActns.TDownLoadURL.ImageIndex 11868
ExtActns.TDownLoadURL.OnDataAvailable 11863
ExtActns.TDownLoadURL.OnDownloadProgress 11870
ExtActns.TDownLoadURL.OnHint 11871
ExtActns.TDownLoadURL.OnLowResource 11863
ExtActns.TDownLoadURL.OnObjectAvailable 11864
ExtActns.TDownLoadURL.OnProgress 11864
ExtActns.TDownLoadURL.OnStartBinding 11864
ExtActns.TDownLoadURL.OnStopBinding 11864
ExtActns.TDownLoadURL.OnUpdate 11871
ExtActns.TDownLoadURL.SecondaryShortCuts 11868
ExtActns.TDownLoadURL.ShortCut 11869
ExtActns.TDownLoadURL.URL 11869
ExtActns.TDownLoadURL.Visible 11869
ExtActns.TFileRun 11872
ExtActns.TFileRun.Browse 11878
ExtActns.TFileRun.BrowseDlg 11879
ExtActns.TFileRun.Caption 11879
ExtActns.TFileRun.Directory 11879
ExtActns.TFileRun.Enabled 11880
ExtActns.TFileRun.FileName 11880
ExtActns.TFileRun.HelpContext 11881
ExtActns.TFileRun.Hint 11881
ExtActns.TFileRun.ImageIndex 11881
ExtActns.TFileRun.OnHint 11885
ExtActns.TFileRun.OnUpdate 11885
ExtActns.TFileRun.Operation 11882
ExtActns.TFileRun.Parameters 11882
ExtActns.TFileRun.ParentControl 11883
ExtActns.TFileRun.SecondaryShortCuts 11883
ExtActns.TFileRun.ShortCut 11883
ExtActns.TFileRun.ShowCmd 11884
ExtActns.TFileRun.Visible 11884
ExtActns.TListControlAction 11886
ExtActns.TListControlAction.Create 11891
ExtActns.TListControlAction.HandlesTarget 11892
ExtActns.TListControlAction.IsEnabled 11892
ExtActns.TListControlAction.ListControl 11893
ExtActns.TListControlAction.Notification 11892

ExtActns.TListControlClearSelection 11893
ExtActns.TListControlClearSelection.Caption 11900
ExtActns.TListControlClearSelection.Enabled 11900
ExtActns.TListControlClearSelection.ExecuteTarget 11898
ExtActns.TListControlClearSelection.HelpContext 11900
ExtActns.TListControlClearSelection.HelpKeyword 11901
ExtActns.TListControlClearSelection.HelpType 11901
ExtActns.TListControlClearSelection.Hint 11902
ExtActns.TListControlClearSelection.ImageIndex 11902
ExtActns.TListControlClearSelection.ListControl 11902
ExtActns.TListControlClearSelection.OnHint 11904
ExtActns.TListControlClearSelection.OnUpdate 11904
ExtActns.TListControlClearSelection.SecondaryShortCuts 11903
ExtActns.TListControlClearSelection.ShortCut 11903
ExtActns.TListControlClearSelection.UpdateTarget 11899
ExtActns.TListControlClearSelection.Visible 11904
ExtActns.TListControlCopySelection 11906
ExtActns.TListControlCopySelection.Caption 11912
ExtActns.TListControlCopySelection.Destination 11913
ExtActns.TListControlCopySelection.Enabled 11913
ExtActns.TListControlCopySelection.ExecuteTarget 11910
ExtActns.TListControlCopySelection.HandlesTarget 11911
ExtActns.TListControlCopySelection.HelpContext 11914
ExtActns.TListControlCopySelection.HelpKeyword 11914
ExtActns.TListControlCopySelection.HelpType 11915
ExtActns.TListControlCopySelection.Hint 11915
ExtActns.TListControlCopySelection.ImageIndex 11915
ExtActns.TListControlCopySelection.ListControl 11916
ExtActns.TListControlCopySelection.OnHint 11917
ExtActns.TListControlCopySelection.OnUpdate 11918
ExtActns.TListControlCopySelection.SecondaryShortCuts 11916
ExtActns.TListControlCopySelection.ShortCut 11916
ExtActns.TListControlCopySelection.UpdateTarget 11912
ExtActns.TListControlCopySelection.Visible 11917
ExtActns.TListControlDeleteSelection 11919
ExtActns.TListControlDeleteSelection.Caption 11925
ExtActns.TListControlDeleteSelection.Enabled 11925
ExtActns.TListControlDeleteSelection.ExecuteTarget 11924
ExtActns.TListControlDeleteSelection.HelpContext 11926
ExtActns.TListControlDeleteSelection.HelpKeyword 11926
ExtActns.TListControlDeleteSelection.HelpType 11927
ExtActns.TListControlDeleteSelection.Hint 11927
ExtActns.TListControlDeleteSelection.ImageIndex 11927
ExtActns.TListControlDeleteSelection.ListControl 11928
ExtActns.TListControlDeleteSelection.OnHint 11929
ExtActns.TListControlDeleteSelection.OnUpdate 11930
ExtActns.TListControlDeleteSelection.SecondaryShortCuts 11928
ExtActns.TListControlDeleteSelection.ShortCut 11929
ExtActns.TListControlDeleteSelection.UpdateTarget 11924
ExtActns.TListControlDeleteSelection.Visible 11929
ExtActns.TListControlMoveSelection 11931
ExtActns.TListControlMoveSelection.ExecuteTarget 11936
ExtActns.TListControlSelectAll 11937
ExtActns.TListControlSelectAll.Caption 11943
ExtActns.TListControlSelectAll.Enabled 11943
ExtActns.TListControlSelectAll.ExecuteTarget 11942
ExtActns.TListControlSelectAll.HelpContext 11944
ExtActns.TListControlSelectAll.HelpKeyword 11944
ExtActns.TListControlSelectAll.HelpType 11945
ExtActns.TListControlSelectAll.Hint 11945
ExtActns.TListControlSelectAll.ImageIndex 11945
ExtActns.TListControlSelectAll.ListControl 11946
ExtActns.TListControlSelectAll.OnHint 11947
ExtActns.TListControlSelectAll.OnUpdate 11948
ExtActns.TListControlSelectAll.SecondaryShortCuts 11946
ExtActns.TListControlSelectAll.ShortCut 11946
ExtActns.TListControlSelectAll.UpdateTarget 11942
ExtActns.TListControlSelectAll.Visible 11947
ExtActns.TNextTab 11949
ExtActns.TNextTab.AfterTabChange 11961
ExtActns.TNextTab.BeforeTabChange 11962
ExtActns.TNextTab.Caption 11956
ExtActns.TNextTab.Enabled 11956
ExtActns.TNextTab.ExecuteTarget 11954
ExtActns.TNextTab.HelpContext 11956
ExtActns.TNextTab.HelpKeyword 11957
ExtActns.TNextTab.HelpType 11957
ExtActns.TNextTab.Hint 11958

ExtActns.TNextTab.ImageIndex 11958
ExtActns.TNextTab.LastTabCaption 11958
ExtActns.TNextTab.OnFinish 11962
ExtActns.TNextTab.OnHint 11963
ExtActns.TNextTab.OnUpdate 11963
ExtActns.TNextTab.OnValidateTab 11964
ExtActns.TNextTab.ShortCut 11959
ExtActns.TNextTab.SkipHiddenTabs 11959
ExtActns.TNextTab.TabControl 11960
ExtActns.TNextTab.UpdateTarget 11955
ExtActns.TNextTab.Visible 11960
ExtActns.TNextTab.Wrap 11960
ExtActns.TOpenPicture 11965
ExtActns.TOpenPicture.Caption 11970
ExtActns.TOpenPicture.Dialog 11971
ExtActns.TOpenPicture.Enabled 11971
ExtActns.TOpenPicture.GetDialogClass 11970
ExtActns.TOpenPicture.HelpContext 11971
ExtActns.TOpenPicture.HelpKeyword 11972
ExtActns.TOpenPicture.HelpType 11972
ExtActns.TOpenPicture.Hint 11973
ExtActns.TOpenPicture.ImageIndex 11973
ExtActns.TOpenPicture.OnAccept 11975
ExtActns.TOpenPicture.OnCancel 11975
ExtActns.TOpenPicture.OnHint 11975
ExtActns.TOpenPicture.OnUpdate 11976
ExtActns.TOpenPicture.SecondaryShortCuts 11974
ExtActns.TOpenPicture.ShortCut 11974
ExtActns.TOpenPicture.Visible 11974
ExtActns.TPreviousTab 11977
ExtActns.TPreviousTab.AfterTabChange 11989
ExtActns.TPreviousTab.BeforeTabChange 11989
ExtActns.TPreviousTab.Caption 11983
ExtActns.TPreviousTab.Enabled 11984
ExtActns.TPreviousTab.ExecuteTarget 11982
ExtActns.TPreviousTab.HelpContext 11984
ExtActns.TPreviousTab.HelpKeyword 11985
ExtActns.TPreviousTab.HelpType 11985
ExtActns.TPreviousTab.Hint 11986
ExtActns.TPreviousTab.ImageIndex 11986
ExtActns.TPreviousTab.OnHint 11989
ExtActns.TPreviousTab.OnUpdate 11990
ExtActns.TPreviousTab.OnValidateTab 11991
ExtActns.TPreviousTab.ShortCut 11986
ExtActns.TPreviousTab.SkipHiddenTabs 11987
ExtActns.TPreviousTab.TabControl 11987
ExtActns.TPreviousTab.UpdateTarget 11982
ExtActns.TPreviousTab.Visible 11987
ExtActns.TPreviousTab.Wrap 11988
ExtActns.TRichEditAction 11992
ExtActns.TRichEditAction.Create 11997
ExtActns.TRichEditAction.CurrText 11997
ExtActns.TRichEditAction.HandlesTarget 11998
ExtActns.TRichEditAction.SetFontStyle 11998
ExtActns.TRichEditAlignCenter 11998
ExtActns.TRichEditAlignCenter.ExecuteTarget 12004
ExtActns.TRichEditAlignCenter.UpdateTarget 12004
ExtActns.TRichEditAlignLeft 12005
ExtActns.TRichEditAlignLeft.ExecuteTarget 12010
ExtActns.TRichEditAlignLeft.UpdateTarget 12010
ExtActns.TRichEditAlignRight 12011
ExtActns.TRichEditAlignRight.ExecuteTarget 12016
ExtActns.TRichEditAlignRight.UpdateTarget 12017
ExtActns.TRichEditBold 12017
ExtActns.TRichEditBold.ExecuteTarget 12022
ExtActns.TRichEditBold.UpdateTarget 12023
ExtActns.TRichEditBullets 12023
ExtActns.TRichEditBullets.ExecuteTarget 12029
ExtActns.TRichEditBullets.UpdateTarget 12029
ExtActns.TRichEditItalic 12029
ExtActns.TRichEditItalic.ExecuteTarget 12035
ExtActns.TRichEditItalic.UpdateTarget 12035
ExtActns.TRichEditStrikeOut 12036
ExtActns.TRichEditStrikeOut.ExecuteTarget 12041
ExtActns.TRichEditStrikeOut.UpdateTarget 12042
ExtActns.TRichEditUnderline 12042
ExtActns.TRichEditUnderline.ExecuteTarget 12047
ExtActns.TRichEditUnderline.UpdateTarget 12048

ExtActns.TSavePicture 12048
ExtActns.TSavePicture.Caption 12054
ExtActns.TSavePicture.Dialog 12054
ExtActns.TSavePicture.Enabled 12055
ExtActns.TSavePicture.GetDialogClass 12053
ExtActns.TSavePicture.HelpContext 12055
ExtActns.TSavePicture.HelpKeyword 12056
ExtActns.TSavePicture.HelpType 12056
ExtActns.TSavePicture.Hint 12057
ExtActns.TSavePicture.ImageIndex 12057
ExtActns.TSavePicture.OnAccept 12059
ExtActns.TSavePicture.OnCancel 12059
ExtActns.TSavePicture.OnHint 12059
ExtActns.TSavePicture.OnUpdate 12059
ExtActns.TSavePicture.SecondaryShortCuts 12057
ExtActns.TSavePicture.ShortCut 12058
ExtActns.TSavePicture.Visible 12058
ExtActns.TSendMail 12061
ExtActns.TSendMail.Caption 12067
ExtActns.TSendMail.Destroy 12065
ExtActns.TSendMail.Enabled 12067
ExtActns.TSendMail.ExecuteTarget 12065
ExtActns.TSendMail.HandlesTarget 12066
ExtActns.TSendMail.HelpContext 12068
ExtActns.TSendMail.HelpKeyword 12068
ExtActns.TSendMail.HelpType 12069
ExtActns.TSendMail.Hint 12069
ExtActns.TSendMail.ImageIndex 12069
ExtActns.TSendMail.OnHint 12071
ExtActns.TSendMail.OnUpdate 12072
ExtActns.TSendMail.SecondaryShortCuts 12070
ExtActns.TSendMail.ShortCut 12070
ExtActns.TSendMail.Text 12071
ExtActns.TSendMail.UpdateTarget 12066
ExtActns.TSendMail.Visible 12071
ExtActns.TShowCmd 12096
ExtActns.TShowCmd enumeration 12096
ExtActns.TTabAction 12073
ExtActns.TTabAction.AfterTabChange 12082
ExtActns.TTabAction.BeforeTabChange 12082
ExtActns.TTabAction.Create 12078
ExtActns.TTabAction.HandlesTarget 12078
ExtActns.TTabAction.Notification 12079
ExtActns.TTabAction.OnValidateTab 12083
ExtActns.TTabAction.SelectNextTab 12079
ExtActns.TTabAction.SkipHiddenTabs 12080
ExtActns.TTabAction.TabControl 12080
ExtActns.TTabAction.UpdateTarget 12079
ExtActns.TTabAction.Wrap 12081
ExtActns.TURLAction 12083
ExtActns.TURLAction.Caption 12089
ExtActns.TURLAction.Enabled 12089
ExtActns.TURLAction.HandlesTarget 12088
ExtActns.TURLAction.HelpContext 12090
ExtActns.TURLAction.HelpKeyword 12090
ExtActns.TURLAction.HelpType 12091
ExtActns.TURLAction.Hint 12091
ExtActns.TURLAction.ImageIndex 12091
ExtActns.TURLAction.OnHint 12093
ExtActns.TURLAction.OnUpdate 12094
ExtActns.TURLAction.SecondaryShortCuts 12092
ExtActns.TURLAction.ShortCut 12092
ExtActns.TURLAction.UpdateTarget 12088
ExtActns.TURLAction.URL 12093
ExtActns.TURLAction.Visible 12093
ExtActns.TURLDownloadStatus 12097
ExtActns.TURLDownloadStatus enumeration 12097
ExtActns.TValidateTabEvent 12099
ExtActns.TValidateTabEvent type 12099
ExtCtrls 12101
ExtCtrls namespace 12101
Classes 12104
Constants 13279
Functions 13262
Structs, Records, Enums 13264
Types 13272
ExtCtrls.EGridPanelException 12105
ExtCtrls.Frame3D 13262

ExtCtrls.Frame3D function 13262
ExtCtrls.NaturalNumber 13272
ExtCtrls.NaturalNumber type 13272
ExtCtrls.NoColorSelected 13279
ExtCtrls.NoColorSelected constant 13279
ExtCtrls.NotebookHandlesNeeded 13263
ExtCtrls.NotebookHandlesNeeded function 13263
ExtCtrls.TBalloonFlags 13264
ExtCtrls.TBalloonFlags enumeration 13264
ExtCtrls.TBandDragEvent 13272
ExtCtrls.TBandDragEvent type 13272
ExtCtrls.TBandDrawingStyle 13265
ExtCtrls.TBandDrawingStyle enumeration 13265
ExtCtrls.TBandInfoEvent 13273
ExtCtrls.TBandInfoEvent type 13273
ExtCtrls.TBandMoveEvent 13273
ExtCtrls.TBandMoveEvent type 13273
ExtCtrls.TBandPaintEvent 13273
ExtCtrls.TBandPaintEvent type 13273
ExtCtrls.TBandPaintOption 13265
ExtCtrls.TBandPaintOption enumeration 13265
ExtCtrls.TBandPaintOptions 13274
ExtCtrls.TBandPaintOptions type 13274
ExtCtrls.TBeginBandMoveEvent 13274
ExtCtrls.TBeginBandMoveEvent type 13274
ExtCtrls.TBevel 12105
ExtCtrls.TBevel.Align 12114
ExtCtrls.TBevel.Anchors 12117
ExtCtrls.TBevel.Constraints 12118
ExtCtrls.TBevel.Create 12113
ExtCtrls.TBevel.Paint 12113
ExtCtrls.TBevel.ParentShowHint 12118
ExtCtrls.TBevel.Shape 12119
ExtCtrls.TBevel.ShowHint 12119
ExtCtrls.TBevel.Style 12121
ExtCtrls.TBevel.Visible 12121
ExtCtrls.TBevelShape 13266
ExtCtrls.TBevelShape enumeration 13266
ExtCtrls.TBevelStyle 13267
ExtCtrls.TBevelStyle enumeration 13267
ExtCtrls.TBoundLabel 12123
ExtCtrls.TBoundLabel.AdjustBounds 12131
ExtCtrls.TBoundLabel.BiDiMode 12132
ExtCtrls.TBoundLabel.Caption 12133
ExtCtrls.TBoundLabel.Color 12135
ExtCtrls.TBoundLabel.Create 12131
ExtCtrls.TBoundLabel.DragCursor 12136
ExtCtrls.TBoundLabel.DragKind 12137
ExtCtrls.TBoundLabel.DragMode 12137
ExtCtrls.TBoundLabel.Font 12139
ExtCtrls.TBoundLabel.Height 12140
ExtCtrls.TBoundLabel.Layout 12141
ExtCtrls.TBoundLabel.Left 12141
ExtCtrls.TBoundLabel.OnClick 12154
ExtCtrls.TBoundLabel.OnContextPopup 12156
ExtCtrls.TBoundLabel.OnDblClick 12157
ExtCtrls.TBoundLabel.OnDragDrop 12157
ExtCtrls.TBoundLabel.OnDragOver 12159
ExtCtrls.TBoundLabel.OnEndDock 12162
ExtCtrls.TBoundLabel.OnEndDrag 12162
ExtCtrls.TBoundLabel.OnMouseActivate 12163
ExtCtrls.TBoundLabel.OnMouseDown 12164
ExtCtrls.TBoundLabel.OnMouseMove 12166
ExtCtrls.TBoundLabel.OnMouseUp 12168
ExtCtrls.TBoundLabel.OnStartDock 12171
ExtCtrls.TBoundLabel.OnStartDrag 12171
ExtCtrls.TBoundLabel.ParentBiDiMode 12142
ExtCtrls.TBoundLabel.ParentColor 12142
ExtCtrls.TBoundLabel.ParentFont 12143
ExtCtrls.TBoundLabel.ParentShowHint 12143
ExtCtrls.TBoundLabel.PopupMenu 12144
ExtCtrls.TBoundLabel.ShowAccelChar 12148
ExtCtrls.TBoundLabel.ShowHint 12149
ExtCtrls.TBoundLabel.Top 12151
ExtCtrls.TBoundLabel.Transparent 12152
ExtCtrls.TBoundLabel.Width 12153
ExtCtrls.TBoundLabel.WordWrap 12153
ExtCtrls.TCanResizeEvent 13274

ExtCtrls.TCanResizeEvent type 13274
ExtCtrls.TCellCollection 12172
ExtCtrls.TCellCollection.GetAttr 12174
ExtCtrls.TCellCollection.GetAttrCount 12175
ExtCtrls.TCellCollection.GetItem 12175
ExtCtrls.TCellCollection.GetItemAttr 12176
ExtCtrls.TCellCollection.Items 12177
ExtCtrls.TCellCollection.Owner 12176
ExtCtrls.TCellCollection.SetItem 12176
ExtCtrls.TCellCollection.Update 12176
ExtCtrls.TCellItem 12177
ExtCtrls.TCellItem.AssignTo 12180
ExtCtrls.TCellItem.AutoAdded 12181
ExtCtrls.TCellItem.Create 12179
ExtCtrls.TCellItem.SetSizeStyle 12180
ExtCtrls.TCellItem.SetValue 12181
ExtCtrls.TCellItem.Size 12181
ExtCtrls.TCellItem.SizeStyle 12181
ExtCtrls.TCellItem.Value 12182
ExtCtrls.TCellSpan 13275
ExtCtrls.TCellSpan type 13275
ExtCtrls.TColorBox 12182
ExtCtrls.TColorBox.Align 12196
ExtCtrls.TColorBox.Anchors 12200
ExtCtrls.TColorBox.AutoComplete 12200
ExtCtrls.TColorBox.AutoDropDown 12201
ExtCtrls.TColorBox.BevelEdges 12201
ExtCtrls.TColorBox.BevelInner 12201
ExtCtrls.TColorBox.BevelKind 12202
ExtCtrls.TColorBox.BevelOuter 12202
ExtCtrls.TColorBox.BiDiMode 12203
ExtCtrls.TColorBox.Color 12203
ExtCtrls.TColorBox.Constraints 12204
ExtCtrls.TColorBox.Ctl3D 12205
ExtCtrls.TColorBox.DefaultColorColor 12205
ExtCtrls.TColorBox.DropDownCount 12205
ExtCtrls.TColorBox.Enabled 12206
ExtCtrls.TColorBox.Font 12206
ExtCtrls.TColorBox.ItemHeight 12207
ExtCtrls.TColorBox.NoneColorColor 12208
ExtCtrls.TColorBox.OnChange 12221
ExtCtrls.TColorBox.OnClick 12223
ExtCtrls.TColorBox.OnCloseUp 12225
ExtCtrls.TColorBox.OnContextPopup 12225
ExtCtrls.TColorBox.OnDblClick 12226
ExtCtrls.TColorBox.OnDragDrop 12226
ExtCtrls.TColorBox.OnDragOver 12228
ExtCtrls.TColorBox.OnDropDown 12231
ExtCtrls.TColorBox.OnEndDock 12232
ExtCtrls.TColorBox.OnEndDrag 12232
ExtCtrls.TColorBox.OnEnter 12233
ExtCtrls.TColorBox.OnExit 12236
ExtCtrls.TColorBox.OnGetColors 12237
ExtCtrls.TColorBox.OnKeyDown 12238
ExtCtrls.TColorBox.OnKeyPress 12240
ExtCtrls.TColorBox.OnKeyUp 12242
ExtCtrls.TColorBox.OnMouseEnter 12243
ExtCtrls.TColorBox.OnMouseLeave 12244
ExtCtrls.TColorBox.OnSelect 12244
ExtCtrls.TColorBox.OnStartDock 12244
ExtCtrls.TColorBox.OnStartDrag 12245
ExtCtrls.TColorBox.ParentBiDiMode 12208
ExtCtrls.TColorBox.ParentColor 12209
ExtCtrls.TColorBox.ParentCtl3D 12209
ExtCtrls.TColorBox.ParentFont 12210
ExtCtrls.TColorBox.ParentShowHint 12210
ExtCtrls.TColorBox.PopupMenu 12211
ExtCtrls.TColorBox.Selected 12215
ExtCtrls.TColorBox.ShowHint 12215
ExtCtrls.TColorBox.Style 12217
ExtCtrls.TColorBox.TabOrder 12217
ExtCtrls.TColorBox.TabStop 12218
ExtCtrls.TColorBox.Visible 12218
ExtCtrls.TColorBoxStyle 13275
ExtCtrls.TColorBoxStyle type 13275
ExtCtrls.TColorBoxStyles 13267
ExtCtrls.TColorBoxStyles enumeration 13267
ExtCtrls.TColorListBox 12246

ExtCtrls.TColorListBox.Align 12259
ExtCtrls.TColorListBox.Anchors 12262
ExtCtrls.TColorListBox.AutoComplete 12263
ExtCtrls.TColorListBox.BevelEdges 12263
ExtCtrls.TColorListBox.BevelInner 12264
ExtCtrls.TColorListBox.BevelKind 12264
ExtCtrls.TColorListBox.BevelOuter 12264
ExtCtrls.TColorListBox.BiDiMode 12265
ExtCtrls.TColorListBox.Color 12265
ExtCtrls.TColorListBox.Constraints 12266
ExtCtrls.TColorListBox.Ctl3D 12266
ExtCtrls.TColorListBox.DefaultColorColor 12267
ExtCtrls.TColorListBox.Enabled 12267
ExtCtrls.TColorListBox.Font 12267
ExtCtrls.TColorListBox.ItemHeight 12268
ExtCtrls.TColorListBox.NoneColorColor 12269
ExtCtrls.TColorListBox.OnClick 12283
ExtCtrls.TColorListBox.OnContextPopup 12285
ExtCtrls.TColorListBox.OnDblClick 12286
ExtCtrls.TColorListBox.OnDragDrop 12286
ExtCtrls.TColorListBox.OnDragOver 12288
ExtCtrls.TColorListBox.OnEndDock 12291
ExtCtrls.TColorListBox.OnEndDrag 12291
ExtCtrls.TColorListBox.OnEnter 12292
ExtCtrls.TColorListBox.OnExit 12295
ExtCtrls.TColorListBox.OnGetColors 12296
ExtCtrls.TColorListBox.OnKeyDown 12296
ExtCtrls.TColorListBox.OnKeyPress 12299
ExtCtrls.TColorListBox.OnKeyUp 12300
ExtCtrls.TColorListBox.OnMouseActivate 12302
ExtCtrls.TColorListBox.OnMouseDown 12303
ExtCtrls.TColorListBox.OnMouseEnter 12305
ExtCtrls.TColorListBox.OnMouseLeave 12305
ExtCtrls.TColorListBox.OnMouseMove 12306
ExtCtrls.TColorListBox.OnMouseUp 12308
ExtCtrls.TColorListBox.OnStartDock 12310
ExtCtrls.TColorListBox.OnStartDrag 12311
ExtCtrls.TColorListBox.ParentBiDiMode 12269
ExtCtrls.TColorListBox.ParentColor 12270
ExtCtrls.TColorListBox.ParentCtl3D 12270
ExtCtrls.TColorListBox.ParentFont 12270
ExtCtrls.TColorListBox.ParentShowHint 12271
ExtCtrls.TColorListBox.PopupMenu 12271
ExtCtrls.TColorListBox.Selected 12276
ExtCtrls.TColorListBox.ShowHint 12278
ExtCtrls.TColorListBox.Style 12279
ExtCtrls.TColorListBox.TabOrder 12280
ExtCtrls.TColorListBox.TabStop 12281
ExtCtrls.TColorListBox.Visible 12281
ExtCtrls.TColumnCollection 12311
ExtCtrls.TColumnCollection.Add 12314
ExtCtrls.TColumnCollection.Create 12314
ExtCtrls.TColumnCollection.GetItemAttr 12314
ExtCtrls.TColumnCollection.Notify 12315
ExtCtrls.TColumnItem 12315
ExtCtrls.TControlBar 12316
ExtCtrls.TControlBar.Align 12328
ExtCtrls.TControlBar.Anchors 12332
ExtCtrls.TControlBar.AutoDock 12333
ExtCtrls.TControlBar.AutoDrag 12333
ExtCtrls.TControlBar.AutoSize 12333
ExtCtrls.TControlBar.BevelEdges 12334
ExtCtrls.TControlBar.BevelInner 12334
ExtCtrls.TControlBar.BevelKind 12335
ExtCtrls.TControlBar.BevelOuter 12335
ExtCtrls.TControlBar.BevelWidth 12336
ExtCtrls.TControlBar.BorderWidth 12336
ExtCtrls.TControlBar.Canvas 12336
ExtCtrls.TControlBar.Color 12338
ExtCtrls.TControlBar.Constraints 12339
ExtCtrls.TControlBar.CornerEdge 12340
ExtCtrls.TControlBar.DockSite 12340
ExtCtrls.TControlBar.DragCursor 12341
ExtCtrls.TControlBar.DragKind 12341
ExtCtrls.TControlBar.DragMode 12342
ExtCtrls.TControlBar.DrawingStyle 12343
ExtCtrls.TControlBar.Enabled 12343
ExtCtrls.TControlBar.GradientDirection 12344

ExtCtrls.TControlBar.GradientEndColor 12344
ExtCtrls.TControlBar.GradientStartColor 12344
ExtCtrls.TControlBar.OnAlignInsertBefore 12357
ExtCtrls.TControlBar.OnAlignPosition 12357
ExtCtrls.TControlBar.OnBandDrag 12358
ExtCtrls.TControlBar.OnBandInfo 12358
ExtCtrls.TControlBar.OnBandMove 12359
ExtCtrls.TControlBar.OnBandPaint 12359
ExtCtrls.TControlBar.OnBeginBandMove 12359
ExtCtrls.TControlBar.OnCanResize 12359
ExtCtrls.TControlBar.OnClick 12360
ExtCtrls.TControlBar.OnConstrainedResize 12362
ExtCtrls.TControlBar.OnContextPopup 12362
ExtCtrls.TControlBar.OnDblClick 12363
ExtCtrls.TControlBar.OnDockDrop 12363
ExtCtrls.TControlBar.OnDockOver 12365
ExtCtrls.TControlBar.OnDragDrop 12367
ExtCtrls.TControlBar.OnDragOver 12370
ExtCtrls.TControlBar.OnEndBandMove 12372
ExtCtrls.TControlBar.OnEndDock 12372
ExtCtrls.TControlBar.OnEndDrag 12373
ExtCtrls.TControlBar.OnEnter 12374
ExtCtrls.TControlBar.OnExit 12376
ExtCtrls.TControlBar.OnGetSiteInfo 12378
ExtCtrls.TControlBar.OnMouseActivate 12380
ExtCtrls.TControlBar.OnMouseDown 12381
ExtCtrls.TControlBar.OnMouseEnter 12383
ExtCtrls.TControlBar.OnMouseLeave 12383
ExtCtrls.TControlBar.OnMouseMove 12384
ExtCtrls.TControlBar.OnMouseUp 12386
ExtCtrls.TControlBar.OnPaint 12388
ExtCtrls.TControlBar.OnResize 12389
ExtCtrls.TControlBar.OnStartDock 12389
ExtCtrls.TControlBar.OnStartDrag 12390
ExtCtrls.TControlBar.OnUnDock 12390
ExtCtrls.TControlBar.ParentBackground 12345
ExtCtrls.TControlBar.ParentColor 12345
ExtCtrls.TControlBar.ParentCtl3D 12345
ExtCtrls.TControlBar.ParentFont 12346
ExtCtrls.TControlBar.ParentShowHint 12346
ExtCtrls.TControlBar.Picture 12347
ExtCtrls.TControlBar.PopupMenu 12347
ExtCtrls.TControlBar.RowSize 12351
ExtCtrls.TControlBar.RowSnap 12352
ExtCtrls.TControlBar.ShowHint 12352
ExtCtrls.TControlBar.TabOrder 12353
ExtCtrls.TControlBar.TabStop 12354
ExtCtrls.TControlBar.Visible 12354
ExtCtrls.TControlCollection 12393
ExtCtrls.TControlCollection.Add 12396
ExtCtrls.TControlCollection.AddControl 12396
ExtCtrls.TControlCollection.ControlItems 12399
ExtCtrls.TControlCollection.Controls 12399
ExtCtrls.TControlCollection.Create 12395
ExtCtrls.TControlCollection.GetControl 12397
ExtCtrls.TControlCollection.GetControlItem 12397
ExtCtrls.TControlCollection.GetItem 12397
ExtCtrls.TControlCollection.IndexOf 12397
ExtCtrls.TControlCollection.Items 12400
ExtCtrls.TControlCollection.Owner 12398
ExtCtrls.TControlCollection.RemoveControl 12398
ExtCtrls.TControlCollection.SetControl 12398
ExtCtrls.TControlCollection.SetItem 12398
ExtCtrls.TControlCollection.Update 12398
ExtCtrls.TControlItem 12400
ExtCtrls.TControlItem.AssignTo 12403
ExtCtrls.TControlItem.Column 12404
ExtCtrls.TControlItem.ColumnSpan 12404
ExtCtrls.TControlItem.Control 12405
ExtCtrls.TControlItem.Create 12402
ExtCtrls.TControlItem.Destroy 12402
ExtCtrls.TControlItem.GridPanel 12405
ExtCtrls.TControlItem.InternalSetLocation 12403
ExtCtrls.TControlItem.Pushed 12405
ExtCtrls.TControlItem.Row 12405
ExtCtrls.TControlItem.RowSpan 12406
ExtCtrls.TControlItem.SetLocation 12404
ExtCtrls.TCornerEdge 13268

ExtCtrls.TCornerEdge enumeration 13268
ExtCtrls.TCustomColorBox 12406
ExtCtrls.TCustomColorBox.CloseUp 12419
ExtCtrls.TCustomColorBox.ColorNames 12423
ExtCtrls.TCustomColorBox.Colors 12424
ExtCtrls.TCustomColorBox.Create 12419
ExtCtrls.TCustomColorBox.CreateWnd 12420
ExtCtrls.TCustomColorBox.DefaultColorColor 12424
ExtCtrls.TCustomColorBox.DrawItem 12420
ExtCtrls.TCustomColorBox.KeyDown 12420
ExtCtrls.TCustomColorBox.KeyPress 12421
ExtCtrls.TCustomColorBox.Loaded 12421
ExtCtrls.TCustomColorBox.NoneColorColor 12424
ExtCtrls.TCustomColorBox.OnGetColors 12426
ExtCtrls.TCustomColorBox.PickCustomColor 12422
ExtCtrls.TCustomColorBox.PopulateList 12422
ExtCtrls.TCustomColorBox.Select 12422
ExtCtrls.TCustomColorBox.Selected 12425
ExtCtrls.TCustomColorBox.SetStyle 12422
ExtCtrls.TCustomColorBox.Style 12425
ExtCtrls.TCustomColorListBox 12426
ExtCtrls.TCustomColorListBox.ColorNames 12442
ExtCtrls.TCustomColorListBox.Colors 12443
ExtCtrls.TCustomColorListBox.Create 12438
ExtCtrls.TCustomColorListBox.CreateWnd 12439
ExtCtrls.TCustomColorListBox.DefaultColorColor 12443
ExtCtrls.TCustomColorListBox.DrawItem 12439
ExtCtrls.TCustomColorListBox.KeyDown 12440
ExtCtrls.TCustomColorListBox.KeyPress 12441
ExtCtrls.TCustomColorListBox.Loaded 12441
ExtCtrls.TCustomColorListBox.NoneColorColor 12443
ExtCtrls.TCustomColorListBox.OnGetColors 12446
ExtCtrls.TCustomColorListBox.PickCustomColor 12441
ExtCtrls.TCustomColorListBox.PopulateList 12442
ExtCtrls.TCustomColorListBox.Selected 12443
ExtCtrls.TCustomColorListBox.SetStyle 12442
ExtCtrls.TCustomColorListBox.Style 12445
ExtCtrls.TCustomControlBar 12446
ExtCtrls.TCustomControlBar.AlignControls 12459
ExtCtrls.TCustomControlBar.AutoDock 12467
ExtCtrls.TCustomControlBar.AutoDrag 12468
ExtCtrls.TCustomControlBar.AutoSize 12468
ExtCtrls.TCustomControlBar.BevelKind 12469
ExtCtrls.TCustomControlBar.CanAutoSize 12459
ExtCtrls.TCustomControlBar.CornerEdge 12469
ExtCtrls.TCustomControlBar.Create 12457
ExtCtrls.TCustomControlBar.CreateParams 12460
ExtCtrls.TCustomControlBar.Destroy 12458
ExtCtrls.TCustomControlBar.DoBandMove 12460
ExtCtrls.TCustomControlBar.DoBandPaint 12460
ExtCtrls.TCustomControlBar.DoBeginBandMove 12461
ExtCtrls.TCustomControlBar.DockOver 12461
ExtCtrls.TCustomControlBar.DockSite 12469
ExtCtrls.TCustomControlBar.DoEndBandMove 12461
ExtCtrls.TCustomControlBar.DoPaletteChange 12461
ExtCtrls.TCustomControlBar.DragControl 12462
ExtCtrls.TCustomControlBar.DrawingStyle 12470
ExtCtrls.TCustomControlBar.FlipChildren 12462
ExtCtrls.TCustomControlBar.GetControlInfo 12463
ExtCtrls.TCustomControlBar.GetPalette 12463
ExtCtrls.TCustomControlBar.GetSiteInfo 12464
ExtCtrls.TCustomControlBar.GradientDirection 12471
ExtCtrls.TCustomControlBar.GradientEndColor 12471
ExtCtrls.TCustomControlBar.GradientStartColor 12471
ExtCtrls.TCustomControlBar.HitTest 12464
ExtCtrls.TCustomControlBar.MouseDown 12465
ExtCtrls.TCustomControlBar.MouseMove 12465
ExtCtrls.TCustomControlBar.MouseUp 12465
ExtCtrls.TCustomControlBar.OnBandDrag 12472
ExtCtrls.TCustomControlBar.OnBandInfo 12473
ExtCtrls.TCustomControlBar.OnBandMove 12473
ExtCtrls.TCustomControlBar.OnBandPaint 12473
ExtCtrls.TCustomControlBar.OnBeginBandMove 12474
ExtCtrls.TCustomControlBar.OnEndBandMove 12474
ExtCtrls.TCustomControlBar.OnPaint 12474
ExtCtrls.TCustomControlBar.Paint 12466
ExtCtrls.TCustomControlBar.PaintControlFrame 12466
ExtCtrls.TCustomControlBar.Picture 12471

ExtCtrls.TCustomControlBar.RowSize 12472
ExtCtrls.TCustomControlBar.RowSnap 12472
ExtCtrls.TCustomControlBar.StickControls 12467
ExtCtrls.TCustomFlowPanel 12475
ExtCtrls.TCustomFlowPanel.AlignControls 12487
ExtCtrls.TCustomFlowPanel.AutoWrap 12488
ExtCtrls.TCustomFlowPanel.Create 12486
ExtCtrls.TCustomFlowPanel.Destroy 12486
ExtCtrls.TCustomFlowPanel.FlowStyle 12489
ExtCtrls.TCustomFlowPanel.GetChildren 12487
ExtCtrls.TCustomFlowPanel.GetControlIndex 12488
ExtCtrls.TCustomFlowPanel.SetControlIndex 12488
ExtCtrls.TCustomGridPanel 12489
ExtCtrls.TCustomGridPanel.AlignControls 12502
ExtCtrls.TCustomGridPanel.AutoAddColumn 12502
ExtCtrls.TCustomGridPanel.AutoAddRow 12503
ExtCtrls.TCustomGridPanel.CellCount 12506
ExtCtrls.TCustomGridPanel.CellIndexToCell 12503
ExtCtrls.TCustomGridPanel.CellRect 12506
ExtCtrls.TCustomGridPanel.CellSize 12507
ExtCtrls.TCustomGridPanel.CellToCellIndex 12503
ExtCtrls.TCustomGridPanel.ColumnCollection 12507
ExtCtrls.TCustomGridPanel.ColumnSpanIndex 12507
ExtCtrls.TCustomGridPanel.ControlCollection 12507
ExtCtrls.TCustomGridPanel.Create 12500
ExtCtrls.TCustomGridPanel.Destroy 12501
ExtCtrls.TCustomGridPanel.ExpandStyle 12507
ExtCtrls.TCustomGridPanel.GetChildren 12503
ExtCtrls.TCustomGridPanel.IsColumnEmpty 12504
ExtCtrls.TCustomGridPanel.IsRowEmpty 12504
ExtCtrls.TCustomGridPanel.Loaded 12504
ExtCtrls.TCustomGridPanel.Paint 12505
ExtCtrls.TCustomGridPanel.RemoveEmptyAutoAddColumns 12505
ExtCtrls.TCustomGridPanel.RemoveEmptyAutoAddRows 12505
ExtCtrls.TCustomGridPanel.RowCollection 12508
ExtCtrls.TCustomGridPanel.RowSpanIndex 12508
ExtCtrls.TCustomGridPanel.UpdateControlOriginalParentSize 12505
ExtCtrls.TCustomLabeledEdit 12508
ExtCtrls.TCustomLabeledEdit.CMBidimodechanged 12521
ExtCtrls.TCustomLabeledEdit.CMEnabledchanged 12521
ExtCtrls.TCustomLabeledEdit.CMVisiblechanged 12521
ExtCtrls.TCustomLabeledEdit.Create 12520
ExtCtrls.TCustomLabeledEdit.EditLabel 12524
ExtCtrls.TCustomLabeledEdit.LabelPosition 12524
ExtCtrls.TCustomLabeledEdit.LabelSpacing 12525
ExtCtrls.TCustomLabeledEdit.Notification 12522
ExtCtrls.TCustomLabeledEdit.SetBounds 12522
ExtCtrls.TCustomLabeledEdit.SetName 12523
ExtCtrls.TCustomLabeledEdit.SetParent 12523
ExtCtrls.TCustomLabeledEdit.SetupInternalLabel 12523
ExtCtrls.TCustomPanel 12525
ExtCtrls.TCustomPanel.AdjustClientRect 12537
ExtCtrls.TCustomPanel.Alignment 12539
ExtCtrls.TCustomPanel.BevelInner 12540
ExtCtrls.TCustomPanel.BevelOuter 12541
ExtCtrls.TCustomPanel.BevelWidth 12542
ExtCtrls.TCustomPanel.BorderStyle 12543
ExtCtrls.TCustomPanel.BorderWidth 12543
ExtCtrls.TCustomPanel.CanAutoSize 12537
ExtCtrls.TCustomPanel.Color 12544
ExtCtrls.TCustomPanel.Create 12536
ExtCtrls.TCustomPanel.CreateParams 12537
ExtCtrls.TCustomPanel.FullRepaint 12545
ExtCtrls.TCustomPanel.GetControlsAlignment 12538
ExtCtrls.TCustomPanel.Locked 12545
ExtCtrls.TCustomPanel.Paint 12538
ExtCtrls.TCustomPanel.ParentBackground 12546
ExtCtrls.TCustomPanel.ParentColor 12546
ExtCtrls.TCustomPanel.SetParentBackground 12539
ExtCtrls.TCustomPanel.VerticalAlignment 12547
ExtCtrls.TCustomRadioGroup 12547
ExtCtrls.TCustomRadioGroup.Buttons 12561
ExtCtrls.TCustomRadioGroup.CanModify 12559
ExtCtrls.TCustomRadioGroup.Columns 12562
ExtCtrls.TCustomRadioGroup.Create 12558
ExtCtrls.TCustomRadioGroup.Destroy 12558
ExtCtrls.TCustomRadioGroup.FlipChildren 12559

ExtCtrls.TCustomRadioGroup.GetChildren 12560
ExtCtrls.TCustomRadioGroup.ItemIndex 12562
ExtCtrls.TCustomRadioGroup.Items 12563
ExtCtrls.TCustomRadioGroup.Loaded 12560
ExtCtrls.TCustomRadioGroup.ReadState 12560
ExtCtrls.TCustomTrayIcon 12564
ExtCtrls.TCustomTrayIcon.Animate 12573
ExtCtrls.TCustomTrayIcon.AnimateInterval 12574
ExtCtrls.TCustomTrayIcon.BalloonFlags 12574
ExtCtrls.TCustomTrayIcon.BalloonHint 12574
ExtCtrls.TCustomTrayIcon.BalloonTimeout 12575
ExtCtrls.TCustomTrayIcon.BalloonTitle 12575
ExtCtrls.TCustomTrayIcon.Create 12568
ExtCtrls.TCustomTrayIcon.Data 12575
ExtCtrls.TCustomTrayIcon.Destroy 12568
ExtCtrls.TCustomTrayIcon.DoOnAnimate 12569
ExtCtrls.TCustomTrayIcon.GetAnimateInterval 12569
ExtCtrls.TCustomTrayIcon.GetBalloonTimeout 12569
ExtCtrls.TCustomTrayIcon.Hint 12575
ExtCtrls.TCustomTrayIcon.Icon 12576
ExtCtrls.TCustomTrayIcon.IconIndex 12576
ExtCtrls.TCustomTrayIcon.Icons 12576
ExtCtrls.TCustomTrayIcon.OnAnimate 12577
ExtCtrls.TCustomTrayIcon.OnClick 12577
ExtCtrls.TCustomTrayIcon.OnDblClick 12578
ExtCtrls.TCustomTrayIcon.OnMouseDown 12578
ExtCtrls.TCustomTrayIcon.OnMouseMove 12578
ExtCtrls.TCustomTrayIcon.OnMouseUp 12578
ExtCtrls.TCustomTrayIcon.PopupMenu 12576
ExtCtrls.TCustomTrayIcon.Refresh 12570
ExtCtrls.TCustomTrayIcon.SetAnimate 12570
ExtCtrls.TCustomTrayIcon.SetAnimateInterval 12570
ExtCtrls.TCustomTrayIcon.SetBalloonHint 12571
ExtCtrls.TCustomTrayIcon.SetBalloonTimeout 12571
ExtCtrls.TCustomTrayIcon.SetBalloonTitle 12571
ExtCtrls.TCustomTrayIcon.SetDefaultIcon 12571
ExtCtrls.TCustomTrayIcon.SetHint 12571
ExtCtrls.TCustomTrayIcon.SetIcon 12572
ExtCtrls.TCustomTrayIcon.SetIconIndex 12572
ExtCtrls.TCustomTrayIcon.SetIconList 12572
ExtCtrls.TCustomTrayIcon.SetVisible 12572
ExtCtrls.TCustomTrayIcon.ShowBalloonHint 12573
ExtCtrls.TCustomTrayIcon.Visible 12577
ExtCtrls.TCustomTrayIcon.WindowProc 12573
ExtCtrls.TEndBandMoveEvent 13276
ExtCtrls.TEndBandMoveEvent type 13276
ExtCtrls.TExpandStyle 13268
ExtCtrls.TExpandStyle enumeration 13268
ExtCtrls.TFlowPanel 12579
ExtCtrls.TFlowPanel.Align 12591
ExtCtrls.TFlowPanel.Alignment 12594
ExtCtrls.TFlowPanel.Anchors 12595
ExtCtrls.TFlowPanel.AutoSize 12595
ExtCtrls.TFlowPanel.AutoWrap 12596
ExtCtrls.TFlowPanel.BevelEdges 12596
ExtCtrls.TFlowPanel.BevelInner 12596
ExtCtrls.TFlowPanel.BevelKind 12597
ExtCtrls.TFlowPanel.BevelOuter 12598
ExtCtrls.TFlowPanel.BevelWidth 12599
ExtCtrls.TFlowPanel.BiDiMode 12600
ExtCtrls.TFlowPanel.BorderStyle 12600
ExtCtrls.TFlowPanel.BorderWidth 12601
ExtCtrls.TFlowPanel.Caption 12602
ExtCtrls.TFlowPanel.Color 12605
ExtCtrls.TFlowPanel.Constraints 12605
ExtCtrls.TFlowPanel.Ctl3D 12606
ExtCtrls.TFlowPanel.DockSite 12606
ExtCtrls.TFlowPanel.DragCursor 12607
ExtCtrls.TFlowPanel.DragKind 12608
ExtCtrls.TFlowPanel.DragMode 12608
ExtCtrls.TFlowPanel.Enabled 12610
ExtCtrls.TFlowPanel.FlowStyle 12610
ExtCtrls.TFlowPanel.Font 12611
ExtCtrls.TFlowPanel.FullRepaint 12612
ExtCtrls.TFlowPanel.Locked 12612
ExtCtrls.TFlowPanel.OnAlignInsertBefore 12626
ExtCtrls.TFlowPanel.OnAlignPosition 12626
ExtCtrls.TFlowPanel.OnCanResize 12627

ExtCtrls.TFlowPanel.OnClick 12627
ExtCtrls.TFlowPanel.OnConstrainedResize 12629
ExtCtrls.TFlowPanel.OnContextPopup 12630
ExtCtrls.TFlowPanel.OnDblClick 12631
ExtCtrls.TFlowPanel.OnDockDrop 12631
ExtCtrls.TFlowPanel.OnDockOver 12633
ExtCtrls.TFlowPanel.OnDragDrop 12635
ExtCtrls.TFlowPanel.OnDragOver 12637
ExtCtrls.TFlowPanel.OnEndDock 12640
ExtCtrls.TFlowPanel.OnEndDrag 12640
ExtCtrls.TFlowPanel.OnEnter 12641
ExtCtrls.TFlowPanel.OnExit 12643
ExtCtrls.TFlowPanel.OnGetSiteInfo 12645
ExtCtrls.TFlowPanel.OnMouseActivate 12647
ExtCtrls.TFlowPanel.OnMouseDown 12648
ExtCtrls.TFlowPanel.OnMouseEnter 12650
ExtCtrls.TFlowPanel.OnMouseLeave 12651
ExtCtrls.TFlowPanel.OnMouseMove 12651
ExtCtrls.TFlowPanel.OnMouseUp 12653
ExtCtrls.TFlowPanel.OnResize 12655
ExtCtrls.TFlowPanel.OnStartDock 12656
ExtCtrls.TFlowPanel.OnStartDrag 12657
ExtCtrls.TFlowPanel.OnUnDock 12657
ExtCtrls.TFlowPanel.Padding 12613
ExtCtrls.TFlowPanel.ParentBackground 12613
ExtCtrls.TFlowPanel.ParentBiDiMode 12613
ExtCtrls.TFlowPanel.ParentColor 12614
ExtCtrls.TFlowPanel.ParentCtl3D 12614
ExtCtrls.TFlowPanel.ParentFont 12614
ExtCtrls.TFlowPanel.ParentShowHint 12615
ExtCtrls.TFlowPanel.PopupMenu 12616
ExtCtrls.TFlowPanel.ShowHint 12620
ExtCtrls.TFlowPanel.TabOrder 12622
ExtCtrls.TFlowPanel.TabStop 12622
ExtCtrls.TFlowPanel.UseDockManager 12623
ExtCtrls.TFlowPanel.VerticalAlignment 12623
ExtCtrls.TFlowPanel.Visible 12623
ExtCtrls.TFlowStyle 13269
ExtCtrls.TFlowStyle enumeration 13269
ExtCtrls.TGetColorsEvent 13276
ExtCtrls.TGetColorsEvent type 13276
ExtCtrls.TGridPanel 12660
ExtCtrls.TGridPanel.Align 12673
ExtCtrls.TGridPanel.Alignment 12676
ExtCtrls.TGridPanel.Anchors 12677
ExtCtrls.TGridPanel.BevelEdges 12677
ExtCtrls.TGridPanel.BevelInner 12677
ExtCtrls.TGridPanel.BevelKind 12678
ExtCtrls.TGridPanel.BevelOuter 12679
ExtCtrls.TGridPanel.BevelWidth 12680
ExtCtrls.TGridPanel.BiDiMode 12681
ExtCtrls.TGridPanel.BorderStyle 12681
ExtCtrls.TGridPanel.BorderWidth 12682
ExtCtrls.TGridPanel.Caption 12683
ExtCtrls.TGridPanel.Color 12686
ExtCtrls.TGridPanel.ColumnCollection 12687
ExtCtrls.TGridPanel.Constraints 12687
ExtCtrls.TGridPanel.ControlCollection 12687
ExtCtrls.TGridPanel.Ctl3D 12688
ExtCtrls.TGridPanel.DockManager 12688
ExtCtrls.TGridPanel.DockSite 12690
ExtCtrls.TGridPanel.DragCursor 12691
ExtCtrls.TGridPanel.DragKind 12692
ExtCtrls.TGridPanel.DragMode 12692
ExtCtrls.TGridPanel.Enabled 12693
ExtCtrls.TGridPanel.ExpandStyle 12694
ExtCtrls.TGridPanel.Font 12694
ExtCtrls.TGridPanel.FullRepaint 12695
ExtCtrls.TGridPanel.Locked 12696
ExtCtrls.TGridPanel.OnAlignInsertBefore 12709
ExtCtrls.TGridPanel.OnAlignPosition 12710
ExtCtrls.TGridPanel.OnCanResize 12711
ExtCtrls.TGridPanel.OnClick 12711
ExtCtrls.TGridPanel.OnConstrainedResize 12713
ExtCtrls.TGridPanel.OnContextPopup 12713
ExtCtrls.TGridPanel.OnDblClick 12714
ExtCtrls.TGridPanel.OnDockDrop 12714
ExtCtrls.TGridPanel.OnDockOver 12716

ExtCtrls.TGridPanel.OnDragDrop 12719
ExtCtrls.TGridPanel.OnDragOver 12721
ExtCtrls.TGridPanel.OnEndDock 12723
ExtCtrls.TGridPanel.OnEndDrag 12724
ExtCtrls.TGridPanel.OnEnter 12725
ExtCtrls.TGridPanel.OnExit 12727
ExtCtrls.TGridPanel.OnGetSiteInfo 12729
ExtCtrls.TGridPanel.OnMouseActivate 12731
ExtCtrls.TGridPanel.OnMouseDown 12732
ExtCtrls.TGridPanel.OnMouseEnter 12734
ExtCtrls.TGridPanel.OnMouseLeave 12734
ExtCtrls.TGridPanel.OnMouseMove 12734
ExtCtrls.TGridPanel.OnMouseUp 12737
ExtCtrls.TGridPanel.OnResize 12739
ExtCtrls.TGridPanel.OnStartDock 12740
ExtCtrls.TGridPanel.OnStartDrag 12740
ExtCtrls.TGridPanel.OnUnDock 12741
ExtCtrls.TGridPanel.Padding 12696
ExtCtrls.TGridPanel.ParentBackground 12696
ExtCtrls.TGridPanel.ParentBiDiMode 12697
ExtCtrls.TGridPanel.ParentColor 12697
ExtCtrls.TGridPanel.ParentCtl3D 12697
ExtCtrls.TGridPanel.ParentFont 12698
ExtCtrls.TGridPanel.ParentShowHint 12698
ExtCtrls.TGridPanel.PopupMenu 12699
ExtCtrls.TGridPanel.RowCollection 12703
ExtCtrls.TGridPanel.ShowHint 12704
ExtCtrls.TGridPanel.TabOrder 12705
ExtCtrls.TGridPanel.TabStop 12706
ExtCtrls.TGridPanel.UseDockManager 12706
ExtCtrls.TGridPanel.VerticalAlignment 12707
ExtCtrls.TGridPanel.Visible 12707
ExtCtrls.THeader 12744
ExtCtrls.THeader.Align 12759
ExtCtrls.THeader.AllowResize 12762
ExtCtrls.THeader.Anchors 12763
ExtCtrls.THeader.BorderStyle 12763
ExtCtrls.THeader.Constraints 12763
ExtCtrls.THeader.Create 12754
ExtCtrls.THeader.CreateParams 12755
ExtCtrls.THeader.Destroy 12755
ExtCtrls.THeader.Enabled 12764
ExtCtrls.THeader.Font 12764
ExtCtrls.THeader.MouseDown 12756
ExtCtrls.THeader.MouseMove 12756
ExtCtrls.THeader.MouseUp 12757
ExtCtrls.THeader.OnContextPopup 12776
ExtCtrls.THeader.OnSized 12777
ExtCtrls.THeader.OnSizing 12777
ExtCtrls.THeader.Paint 12757
ExtCtrls.THeader.ParentFont 12766
ExtCtrls.THeader.ParentShowHint 12766
ExtCtrls.THeader.PopupMenu 12767
ExtCtrls.THeader.Sections 12771
ExtCtrls.THeader.SectionWidth 12771
ExtCtrls.THeader.ShowHint 12772
ExtCtrls.THeader.Sized 12758
ExtCtrls.THeader.Sizing 12758
ExtCtrls.THeader.TabOrder 12773
ExtCtrls.THeader.TabStop 12774
ExtCtrls.THeader.Visible 12774
ExtCtrls.TImage 12777
ExtCtrls.TImage.Align 12789
ExtCtrls.TImage.Anchors 12793
ExtCtrls.TImage.AutoSize 12793
ExtCtrls.TImage.CanAutoSize 12786
ExtCtrls.TImage.Canvas 12794
ExtCtrls.TImage.Center 12797
ExtCtrls.TImage.Constraints 12797
ExtCtrls.TImage.Create 12785
ExtCtrls.TImage.DestRect 12787
ExtCtrls.TImage.Destroy 12786
ExtCtrls.TImage.DoPaletteChange 12787
ExtCtrls.TImage.DragCursor 12798
ExtCtrls.TImage.DragKind 12798
ExtCtrls.TImage.DragMode 12799
ExtCtrls.TImage.Enabled 12800
ExtCtrls.TImage.GetPalette 12787

ExtCtrls.TImage.IncrementalDisplay 12801
ExtCtrls.TImage.OnClick 12814
ExtCtrls.TImage.OnContextPopup 12816
ExtCtrls.TImage.OnDblClick 12817
ExtCtrls.TImage.OnDragDrop 12817
ExtCtrls.TImage.OnDragOver 12819
ExtCtrls.TImage.OnEndDock 12822
ExtCtrls.TImage.OnEndDrag 12822
ExtCtrls.TImage.OnMouseActivate 12823
ExtCtrls.TImage.OnMouseDown 12824
ExtCtrls.TImage.OnMouseEnter 12826
ExtCtrls.TImage.OnMouseLeave 12826
ExtCtrls.TImage.OnMouseMove 12827
ExtCtrls.TImage.OnMouseUp 12829
ExtCtrls.TImage.OnProgress 12831
ExtCtrls.TImage.OnStartDock 12832
ExtCtrls.TImage.OnStartDrag 12832
ExtCtrls.TImage.Paint 12788
ExtCtrls.TImage.ParentShowHint 12801
ExtCtrls.TImage.Picture 12802
ExtCtrls.TImage.PopupMenu 12803
ExtCtrls.TImage.Progress 12788
ExtCtrls.TImage.Proportional 12808
ExtCtrls.TImage.ShowHint 12808
ExtCtrls.TImage.Stretch 12810
ExtCtrls.TImage.Transparent 12811
ExtCtrls.TImage.Visible 12812
ExtCtrls.TLabeledEdit 12833
ExtCtrls.TLabeledEdit.Anchors 12846
ExtCtrls.TLabeledEdit.AutoSelect 12846
ExtCtrls.TLabeledEdit.AutoSize 12847
ExtCtrls.TLabeledEdit.BevelEdges 12848
ExtCtrls.TLabeledEdit.BevelInner 12848
ExtCtrls.TLabeledEdit.BevelKind 12849
ExtCtrls.TLabeledEdit.BevelOuter 12849
ExtCtrls.TLabeledEdit.BiDiMode 12849
ExtCtrls.TLabeledEdit.BorderStyle 12850
ExtCtrls.TLabeledEdit.CharCase 12850
ExtCtrls.TLabeledEdit.Color 12851
ExtCtrls.TLabeledEdit.Constraints 12851
ExtCtrls.TLabeledEdit.Ctl3D 12852
ExtCtrls.TLabeledEdit.DragCursor 12853
ExtCtrls.TLabeledEdit.DragKind 12853
ExtCtrls.TLabeledEdit.DragMode 12853
ExtCtrls.TLabeledEdit.EditLabel 12855
ExtCtrls.TLabeledEdit.Enabled 12855
ExtCtrls.TLabeledEdit.Font 12856
ExtCtrls.TLabeledEdit.HideSelection 12857
ExtCtrls.TLabeledEdit.ImeMode 12857
ExtCtrls.TLabeledEdit.ImeName 12858
ExtCtrls.TLabeledEdit.LabelPosition 12858
ExtCtrls.TLabeledEdit.LabelSpacing 12859
ExtCtrls.TLabeledEdit.MaxLength 12859
ExtCtrls.TLabeledEdit.OEMConvert 12860
ExtCtrls.TLabeledEdit.OnChange 12873
ExtCtrls.TLabeledEdit.OnClick 12874
ExtCtrls.TLabeledEdit.OnContextPopup 12876
ExtCtrls.TLabeledEdit.OnDblClick 12876
ExtCtrls.TLabeledEdit.OnDragDrop 12877
ExtCtrls.TLabeledEdit.OnDragOver 12879
ExtCtrls.TLabeledEdit.OnEndDock 12881
ExtCtrls.TLabeledEdit.OnEndDrag 12882
ExtCtrls.TLabeledEdit.OnEnter 12883
ExtCtrls.TLabeledEdit.OnExit 12885
ExtCtrls.TLabeledEdit.OnKeyDown 12887
ExtCtrls.TLabeledEdit.OnKeyPress 12890
ExtCtrls.TLabeledEdit.OnKeyUp 12891
ExtCtrls.TLabeledEdit.OnMouseActivate 12892
ExtCtrls.TLabeledEdit.OnMouseDown 12893
ExtCtrls.TLabeledEdit.OnMouseEnter 12896
ExtCtrls.TLabeledEdit.OnMouseLeave 12896
ExtCtrls.TLabeledEdit.OnMouseMove 12896
ExtCtrls.TLabeledEdit.OnMouseUp 12898
ExtCtrls.TLabeledEdit.OnStartDock 12901
ExtCtrls.TLabeledEdit.OnStartDrag 12901
ExtCtrls.TLabeledEdit.ParentBiDiMode 12860
ExtCtrls.TLabeledEdit.ParentColor 12860
ExtCtrls.TLabeledEdit.ParentCtl3D 12861

ExtCtrls.TLabeledEdit.ParentFont 12861
ExtCtrls.TLabeledEdit.ParentShowHint 12862
ExtCtrls.TLabeledEdit.PasswordChar 12862
ExtCtrls.TLabeledEdit.PopupMenu 12863
ExtCtrls.TLabeledEdit.ReadOnly 12867
ExtCtrls.TLabeledEdit.ShowHint 12868
ExtCtrls.TLabeledEdit.TabOrder 12869
ExtCtrls.TLabeledEdit.TabStop 12870
ExtCtrls.TLabeledEdit.Text 12870
ExtCtrls.TLabeledEdit.Visible 12871
ExtCtrls.TLabelPosition 13269
ExtCtrls.TLabelPosition enumeration 13269
ExtCtrls.TLBGetColorsEvent 13276
ExtCtrls.TLBGetColorsEvent type 13276
ExtCtrls.TNotebook 12902
ExtCtrls.TNotebook.ActivePage 12916
ExtCtrls.TNotebook.Align 12917
ExtCtrls.TNotebook.Anchors 12920
ExtCtrls.TNotebook.Color 12921
ExtCtrls.TNotebook.Constraints 12921
ExtCtrls.TNotebook.Create 12912
ExtCtrls.TNotebook.CreateParams 12913
ExtCtrls.TNotebook.Ctl3D 12922
ExtCtrls.TNotebook.Destroy 12913
ExtCtrls.TNotebook.DragCursor 12923
ExtCtrls.TNotebook.DragKind 12923
ExtCtrls.TNotebook.DragMode 12923
ExtCtrls.TNotebook.Enabled 12925
ExtCtrls.TNotebook.Font 12925
ExtCtrls.TNotebook.GetChildOwner 12914
ExtCtrls.TNotebook.GetChildren 12914
ExtCtrls.TNotebook.OnClick 12938
ExtCtrls.TNotebook.OnContextPopup 12940
ExtCtrls.TNotebook.OnDblClick 12941
ExtCtrls.TNotebook.OnDragDrop 12941
ExtCtrls.TNotebook.OnDragOver 12943
ExtCtrls.TNotebook.OnEndDock 12946
ExtCtrls.TNotebook.OnEndDrag 12946
ExtCtrls.TNotebook.OnEnter 12947
ExtCtrls.TNotebook.OnExit 12950
ExtCtrls.TNotebook.OnMouseActivate 12951
ExtCtrls.TNotebook.OnMouseDown 12952
ExtCtrls.TNotebook.OnMouseEnter 12954
ExtCtrls.TNotebook.OnMouseLeave 12955
ExtCtrls.TNotebook.OnMouseMove 12955
ExtCtrls.TNotebook.OnMouseUp 12957
ExtCtrls.TNotebook.OnPageChanged 12959
ExtCtrls.TNotebook.OnStartDock 12960
ExtCtrls.TNotebook.OnStartDrag 12960
ExtCtrls.TNotebook.PageIndex 12926
ExtCtrls.TNotebook.Pages 12927
ExtCtrls.TNotebook.ParentColor 12927
ExtCtrls.TNotebook.ParentCtl3D 12928
ExtCtrls.TNotebook.ParentFont 12928
ExtCtrls.TNotebook.ParentShowHint 12928
ExtCtrls.TNotebook.PopupMenu 12929
ExtCtrls.TNotebook.ReadState 12915
ExtCtrls.TNotebook.ShowControl 12915
ExtCtrls.TNotebook.ShowHint 12934
ExtCtrls.TNotebook.TabOrder 12935
ExtCtrls.TNotebook.TabStop 12936
ExtCtrls.TNotebook.Visible 12936
ExtCtrls.TPage 12961
ExtCtrls.TPage.Caption 12973
ExtCtrls.TPage.Create 12971
ExtCtrls.TPage.Height 12976
ExtCtrls.TPage.OnAlignInsertBefore 12980
ExtCtrls.TPage.OnAlignPosition 12980
ExtCtrls.TPage.Paint 12972
ExtCtrls.TPage.ReadState 12973
ExtCtrls.TPage.TabOrder 12977
ExtCtrls.TPage.Visible 12978
ExtCtrls.TPage.Width 12979
ExtCtrls.TPaintBox 12981
ExtCtrls.TPaintBox.Align 12990
ExtCtrls.TPaintBox.Anchors 12994
ExtCtrls.TPaintBox.Canvas 12994
ExtCtrls.TPaintBox.Color 12996

ExtCtrls.TPaintBox.Constraints 12997
ExtCtrls.TPaintBox.Create 12989
ExtCtrls.TPaintBox.DragCursor 12998
ExtCtrls.TPaintBox.DragKind 12998
ExtCtrls.TPaintBox.DragMode 12998
ExtCtrls.TPaintBox.Enabled 13000
ExtCtrls.TPaintBox.Font 13000
ExtCtrls.TPaintBox.OnClick 13011
ExtCtrls.TPaintBox.OnContextPopup 13013
ExtCtrls.TPaintBox.OnDblClick 13014
ExtCtrls.TPaintBox.OnDragDrop 13014
ExtCtrls.TPaintBox.OnDragOver 13016
ExtCtrls.TPaintBox.OnEndDock 13019
ExtCtrls.TPaintBox.OnEndDrag 13019
ExtCtrls.TPaintBox.OnMouseActivate 13020
ExtCtrls.TPaintBox.OnMouseDown 13021
ExtCtrls.TPaintBox.OnMouseEnter 13023
ExtCtrls.TPaintBox.OnMouseLeave 13024
ExtCtrls.TPaintBox.OnMouseMove 13024
ExtCtrls.TPaintBox.OnMouseUp 13026
ExtCtrls.TPaintBox.OnPaint 13028
ExtCtrls.TPaintBox.OnStartDock 13030
ExtCtrls.TPaintBox.OnStartDrag 13031
ExtCtrls.TPaintBox.Paint 12989
ExtCtrls.TPaintBox.ParentColor 13001
ExtCtrls.TPaintBox.ParentFont 13002
ExtCtrls.TPaintBox.ParentShowHint 13002
ExtCtrls.TPaintBox.PopupMenu 13003
ExtCtrls.TPaintBox.ShowHint 13008
ExtCtrls.TPaintBox.Visible 13009
ExtCtrls.TPanel 13032
ExtCtrls.TPanel.Align 13044
ExtCtrls.TPanel.Alignment 13048
ExtCtrls.TPanel.Anchors 13048
ExtCtrls.TPanel.AutoSize 13049
ExtCtrls.TPanel.BevelEdges 13049
ExtCtrls.TPanel.BevelInner 13050
ExtCtrls.TPanel.BevelKind 13051
ExtCtrls.TPanel.BevelOuter 13051
ExtCtrls.TPanel.BevelWidth 13052
ExtCtrls.TPanel.BiDiMode 13053
ExtCtrls.TPanel.BorderStyle 13054
ExtCtrls.TPanel.BorderWidth 13054
ExtCtrls.TPanel.Caption 13055
ExtCtrls.TPanel.Color 13058
ExtCtrls.TPanel.Constraints 13059
ExtCtrls.TPanel.Ctl3D 13059
ExtCtrls.TPanel.DockManager 13060
ExtCtrls.TPanel.DockSite 13062
ExtCtrls.TPanel.DragCursor 13063
ExtCtrls.TPanel.DragKind 13063
ExtCtrls.TPanel.DragMode 13063
ExtCtrls.TPanel.Enabled 13065
ExtCtrls.TPanel.Font 13066
ExtCtrls.TPanel.FullRepaint 13067
ExtCtrls.TPanel.Locked 13067
ExtCtrls.TPanel.OnAlignInsertBefore 13080
ExtCtrls.TPanel.OnAlignPosition 13081
ExtCtrls.TPanel.OnCanResize 13082
ExtCtrls.TPanel.OnClick 13082
ExtCtrls.TPanel.OnConstrainedResize 13084
ExtCtrls.TPanel.OnContextPopup 13084
ExtCtrls.TPanel.OnDblClick 13085
ExtCtrls.TPanel.OnDockDrop 13085
ExtCtrls.TPanel.OnDockOver 13087
ExtCtrls.TPanel.OnDragDrop 13090
ExtCtrls.TPanel.OnDragOver 13092
ExtCtrls.TPanel.OnEndDock 13094
ExtCtrls.TPanel.OnEndDrag 13095
ExtCtrls.TPanel.OnEnter 13096
ExtCtrls.TPanel.OnExit 13098
ExtCtrls.TPanel.OnGetSiteInfo 13100
ExtCtrls.TPanel.OnMouseActivate 13102
ExtCtrls.TPanel.OnMouseDown 13103
ExtCtrls.TPanel.OnMouseEnter 13105
ExtCtrls.TPanel.OnMouseLeave 13105
ExtCtrls.TPanel.OnMouseMove 13105
ExtCtrls.TPanel.OnMouseUp 13108

ExtCtrls.TPanel.OnResize 13110
ExtCtrls.TPanel.OnStartDock 13111
ExtCtrls.TPanel.OnStartDrag 13111
ExtCtrls.TPanel.OnUnDock 13112
ExtCtrls.TPanel.Padding 13067
ExtCtrls.TPanel.ParentBackground 13068
ExtCtrls.TPanel.ParentBiDiMode 13068
ExtCtrls.TPanel.ParentColor 13068
ExtCtrls.TPanel.ParentCtl3D 13069
ExtCtrls.TPanel.ParentFont 13069
ExtCtrls.TPanel.ParentShowHint 13070
ExtCtrls.TPanel.PopupMenu 13070
ExtCtrls.TPanel.ShowHint 13075
ExtCtrls.TPanel.TabOrder 13076
ExtCtrls.TPanel.TabStop 13077
ExtCtrls.TPanel.UseDockManager 13077
ExtCtrls.TPanel.VerticalAlignment 13078
ExtCtrls.TPanel.Visible 13078
ExtCtrls.TPanelBevel 13277
ExtCtrls.TPanelBevel type 13277
ExtCtrls.TRadioGroup 13115
ExtCtrls.TRadioGroup.Align 13127
ExtCtrls.TRadioGroup.Anchors 13130
ExtCtrls.TRadioGroup.BiDiMode 13131
ExtCtrls.TRadioGroup.Caption 13131
ExtCtrls.TRadioGroup.Color 13134
ExtCtrls.TRadioGroup.Columns 13135
ExtCtrls.TRadioGroup.Constraints 13135
ExtCtrls.TRadioGroup.Ctl3D 13135
ExtCtrls.TRadioGroup.DragCursor 13136
ExtCtrls.TRadioGroup.DragKind 13136
ExtCtrls.TRadioGroup.DragMode 13137
ExtCtrls.TRadioGroup.Enabled 13138
ExtCtrls.TRadioGroup.Font 13139
ExtCtrls.TRadioGroup.ItemIndex 13140
ExtCtrls.TRadioGroup.Items 13141
ExtCtrls.TRadioGroup.OnClick 13153
ExtCtrls.TRadioGroup.OnContextPopup 13155
ExtCtrls.TRadioGroup.OnDragDrop 13156
ExtCtrls.TRadioGroup.OnDragOver 13158
ExtCtrls.TRadioGroup.OnEndDock 13161
ExtCtrls.TRadioGroup.OnEndDrag 13161
ExtCtrls.TRadioGroup.OnEnter 13162
ExtCtrls.TRadioGroup.OnExit 13164
ExtCtrls.TRadioGroup.OnStartDock 13166
ExtCtrls.TRadioGroup.OnStartDrag 13167
ExtCtrls.TRadioGroup.ParentBackground 13142
ExtCtrls.TRadioGroup.ParentBiDiMode 13142
ExtCtrls.TRadioGroup.ParentColor 13143
ExtCtrls.TRadioGroup.ParentCtl3D 13143
ExtCtrls.TRadioGroup.ParentFont 13143
ExtCtrls.TRadioGroup.ParentShowHint 13144
ExtCtrls.TRadioGroup.PopupMenu 13144
ExtCtrls.TRadioGroup.ShowHint 13149
ExtCtrls.TRadioGroup.TabOrder 13150
ExtCtrls.TRadioGroup.TabStop 13151
ExtCtrls.TRadioGroup.Visible 13151
ExtCtrls.TResizeStyle 13270
ExtCtrls.TResizeStyle enumeration 13270
ExtCtrls.TRowCollection 13167
ExtCtrls.TRowCollection.Add 13170
ExtCtrls.TRowCollection.Create 13170
ExtCtrls.TRowCollection.GetItemAttr 13170
ExtCtrls.TRowCollection.Notify 13171
ExtCtrls.TRowItem 13171
ExtCtrls.TRowSize 13277
ExtCtrls.TRowSize type 13277
ExtCtrls.TSectionEvent 13278
ExtCtrls.TSectionEvent type 13278
ExtCtrls.TShape 13172
ExtCtrls.TShape.Align 13181
ExtCtrls.TShape.Anchors 13185
ExtCtrls.TShape.Brush 13185
ExtCtrls.TShape.Constraints 13186
ExtCtrls.TShape.Create 13179
ExtCtrls.TShape.Destroy 13180
ExtCtrls.TShape.DragCursor 13186
ExtCtrls.TShape.DragKind 13187

ExtCtrls.TShape.DragMode 13187
ExtCtrls.TShape.Enabled 13189
ExtCtrls.TShape.OnContextPopup 13194
ExtCtrls.TShape.OnDragDrop 13195
ExtCtrls.TShape.OnDragOver 13197
ExtCtrls.TShape.OnEndDock 13199
ExtCtrls.TShape.OnEndDrag 13200
ExtCtrls.TShape.OnMouseActivate 13201
ExtCtrls.TShape.OnMouseDown 13202
ExtCtrls.TShape.OnMouseEnter 13204
ExtCtrls.TShape.OnMouseLeave 13204
ExtCtrls.TShape.OnMouseMove 13205
ExtCtrls.TShape.OnMouseUp 13207
ExtCtrls.TShape.OnStartDock 13209
ExtCtrls.TShape.OnStartDrag 13210
ExtCtrls.TShape.Paint 13180
ExtCtrls.TShape.ParentShowHint 13189
ExtCtrls.TShape.Pen 13190
ExtCtrls.TShape.Shape 13190
ExtCtrls.TShape.ShowHint 13191
ExtCtrls.TShape.StyleChanged 13181
ExtCtrls.TShape.Visible 13192
ExtCtrls.TShapeType 13270
ExtCtrls.TShapeType enumeration 13270
ExtCtrls.TSizeStyle 13271
ExtCtrls.TSizeStyle enumeration 13271
ExtCtrls.TSplitter 13210
ExtCtrls.TSplitter.Align 13224
ExtCtrls.TSplitter.AutoSnap 13228
ExtCtrls.TSplitter.Beveled 13228
ExtCtrls.TSplitter.CanResize 13220
ExtCtrls.TSplitter.Canvas 13228
ExtCtrls.TSplitter.Color 13231
ExtCtrls.TSplitter.Constraints 13231
ExtCtrls.TSplitter.Create 13218
ExtCtrls.TSplitter.Cursor 13232
ExtCtrls.TSplitter.Destroy 13220
ExtCtrls.TSplitter.DoCanResize 13221
ExtCtrls.TSplitter.MinSize 13235
ExtCtrls.TSplitter.MouseDown 13221
ExtCtrls.TSplitter.MouseMove 13222
ExtCtrls.TSplitter.MouseUp 13222
ExtCtrls.TSplitter.OnCanResize 13240
ExtCtrls.TSplitter.OnMoved 13240
ExtCtrls.TSplitter.OnPaint 13241
ExtCtrls.TSplitter.Paint 13222
ExtCtrls.TSplitter.ParentColor 13236
ExtCtrls.TSplitter.RequestAlign 13223
ExtCtrls.TSplitter.ResizeStyle 13237
ExtCtrls.TSplitter.StopSizing 13223
ExtCtrls.TSplitter.Visible 13237
ExtCtrls.TSplitter.Width 13239
ExtCtrls.TSplitterCanResizeEvent 13278
ExtCtrls.TSplitterCanResizeEvent type 13278
ExtCtrls.TTimer 13242
ExtCtrls.TTimer.Create 13245
ExtCtrls.TTimer.Destroy 13245
ExtCtrls.TTimer.Enabled 13246
ExtCtrls.TTimer.Interval 13249
ExtCtrls.TTimer.OnTimer 13251
ExtCtrls.TTimer.Timer 13246
ExtCtrls.TTrayIcon 13252
ExtCtrls.TTrayIcon.Animate 13256
ExtCtrls.TTrayIcon.AnimateInterval 13256
ExtCtrls.TTrayIcon.BalloonFlags 13257
ExtCtrls.TTrayIcon.BalloonHint 13257
ExtCtrls.TTrayIcon.BalloonTimeout 13257
ExtCtrls.TTrayIcon.BalloonTitle 13257
ExtCtrls.TTrayIcon.Hint 13258
ExtCtrls.TTrayIcon.Icon 13258
ExtCtrls.TTrayIcon.IconIndex 13258
ExtCtrls.TTrayIcon.Icons 13258
ExtCtrls.TTrayIcon.OnAnimate 13260
ExtCtrls.TTrayIcon.OnClick 13260
ExtCtrls.TTrayIcon.OnDblClick 13260
ExtCtrls.TTrayIcon.OnMouseDown 13260
ExtCtrls.TTrayIcon.OnMouseMove 13261
ExtCtrls.TTrayIcon.OnMouseUp 13261

ExtCtrls.TTrayIcon.PopupMenu 13259
ExtCtrls.TTrayIcon.Visible 13259
ExtCtrls.WM_SYSTEM_TRAY_MESSAGE 13279
ExtCtrls.WM_SYSTEM_TRAY_MESSAGE constant 13279
ExtDlgs 13281
ExtDlgs namespace 13281
    Classes 13282
    Constants 13310
ExtDlgs.DefaultEncodingNames 13310
ExtDlgs.DefaultEncodingNames constant 13310
ExtDlgs.TOpenPictureDialog 13282
ExtDlgs.TOpenPictureDialog.Create 13286
ExtDlgs.TOpenPictureDialog.DoClose 13287
ExtDlgs.TOpenPictureDialog.DoSelectionChange 13287
ExtDlgs.TOpenPictureDialog.DoShow 13287
ExtDlgs.TOpenPictureDialog.Execute 13288
ExtDlgs.TOpenPictureDialog.Filter 13289
ExtDlgs.TOpenPictureDialog.ImageCtrl 13292
ExtDlgs.TOpenPictureDialog.PictureLabel 13292
ExtDlgs.TOpenPictureDialog.PreviewClick 13288
ExtDlgs.TOpenTextFileDialog 13292
ExtDlgs.TOpenTextFileDialog.Create 13296
ExtDlgs.TOpenTextFileDialog.Destroy 13296
ExtDlgs.TOpenTextFileDialog.DoClose 13297
ExtDlgs.TOpenTextFileDialog.DoShow 13297
ExtDlgs.TOpenTextFileDialog.EncodingIndex 13300
ExtDlgs.TOpenTextFileDialog.Encodings 13300
ExtDlgs.TOpenTextFileDialog.Execute 13298
ExtDlgs.TSavePictureDialog 13300
ExtDlgs.TSavePictureDialog.Execute 13305
ExtDlgs.TSaveTextFileDialog 13305
ExtDlgs.TSaveTextFileDialog.Execute 13309
exUnderflow enumeration member 15981
exZeroDivide enumeration member 15981
EZeroDivide class 22761
    about EZeroDivide class 22761
EZlibError class 26618
    about EZlibError class 26618

F

faFixed enumeration member 7611
faHiddenCol enumeration member 7611
faLink enumeration member 7611
faReadonly enumeration member 7611
faRequired enumeration member 7611
faUnNamed enumeration member 7611
fdAnsiOnly enumeration member 11555
fdApplyButton enumeration member 11555
fdBoth enumeration member 11554
fdEffects enumeration member 11555
fdFixedPitchOnly enumeration member 11555
fdForceFontExist enumeration member 11555
fdLimitSize enumeration member 11555
fdNoFaceSel enumeration member 11555
fdNoOEMFonts enumeration member 11555
fdNoSimulations enumeration member 11555
fdNoSizeSel enumeration member 11555
fdNoStyleSel enumeration member 11555
fdNoVectorFonts enumeration member 11555
fdoAllNonStorageItems enumeration member 11549
fdoAllowMultiSelect enumeration member 11549
fdoCreatePrompt enumeration member 11549
fdoDefaultNoMiniMode enumeration member 11549
fdoDontAddToRecent enumeration member 11549
fdoFileMustExist enumeration member 11549
fdoForceFileSystem enumeration member 11549
fdoForcePreviewPaneOn enumeration member 11549
fdoForceShowHidden enumeration member 11549
fdoHideMRUPlaces enumeration member 11549
fdoHidePinnedPlaces enumeration member 11549
fdoNoChangeDir enumeration member 11549
fdoNoDereferenceLinks enumeration member 11549
fdoNoReadOnlyReturn enumeration member 11549
fdoNoTestFileCreate enumeration member 11549
fdoNoValidate enumeration member 11549
fdoOverWritePrompt enumeration member 11549
fdoPathMustExist enumeration member 11549

fdoPickFolders enumeration member 11549
fdoShareAware enumeration member 11549
fdoStrictFileTypes enumeration member 11549
fdPrinter enumeration member 11554
fdScalableOnly enumeration member 11555
fdScreen enumeration member 11554
fdShowHelp enumeration member 11555
fdTrueTypeOnly enumeration member 11555
fdWysiwyg enumeration member 11555
ffChildPos enumeration member 3271
ffCurrency enumeration member 22999
ffExponent enumeration member 22999
ffFixed enumeration member 22999
ffGeneral enumeration member 22999
ffInherited enumeration member 3271
ffInKey enumeration member 8111
ffInline enumeration member 3271
ffInOrigValues enumeration member 8111
ffNumber enumeration member 22999
fgAffectedRecords enumeration member 1299
fgConflictingRecords enumeration member 1299
fgFetchedRecords enumeration member 1299
fgNone enumeration member 1299
fgPendingRecords enumeration member 1299
fgPredicate enumeration member 1299
fgUnassigned enumeration member 1299
FileCtrl 13311
FileCtrl namespace 13311
    Classes 13313
    Constants 13626
    Functions 13616
    Structs, Records, Enums 13620
    Types 13624
FileCtrl.DirectoryExists 13616
FileCtrl.DirectoryExists function 13616
FileCtrl.ForceDirectories 13617
FileCtrl.ForceDirectories function 13617
FileCtrl.MinimizeName 13617
FileCtrl.MinimizeName function 13617
FileCtrl.ProcessPath 13618
FileCtrl.ProcessPath function 13618
FileCtrl.SelectDirectory 13619
FileCtrl.SelectDirectory function 13619
FileCtrl.TDirectoryListBox 13313
FileCtrl.TDirectoryListBox.Align 13336
FileCtrl.TDirectoryListBox.Anchors 13339
FileCtrl.TDirectoryListBox.AutoComplete 13340
FileCtrl.TDirectoryListBox.BevelEdges 13340
FileCtrl.TDirectoryListBox.BevelInner 13340
FileCtrl.TDirectoryListBox.BevelKind 13341
FileCtrl.TDirectoryListBox.BevelOuter 13341
FileCtrl.TDirectoryListBox.BuildList 13329
FileCtrl.TDirectoryListBox.CaseSensitive 13342
FileCtrl.TDirectoryListBox.Change 13329
FileCtrl.TDirectoryListBox.ClosedBMP 13327
FileCtrl.TDirectoryListBox.Color 13342
FileCtrl.TDirectoryListBox.Columns 13343
FileCtrl.TDirectoryListBox.Constraints 13343
FileCtrl.TDirectoryListBox.Create 13326
FileCtrl.TDirectoryListBox.CreateWnd 13329
FileCtrl.TDirectoryListBox.Ctl3D 13344
FileCtrl.TDirectoryListBox.CurrentBMP 13327
FileCtrl.TDirectoryListBox.DblClick 13330
FileCtrl.TDirectoryListBox.Destroy 13326
FileCtrl.TDirectoryListBox.Directory 13344
FileCtrl.TDirectoryListBox.DirLabel 13346
FileCtrl.TDirectoryListBox.DisplayCase 13330
FileCtrl.TDirectoryListBox.DragCursor 13347
FileCtrl.TDirectoryListBox.DragMode 13347
FileCtrl.TDirectoryListBox.DrawItem 13330
FileCtrl.TDirectoryListBox.Drive 13349
FileCtrl.TDirectoryListBox.Enabled 13350
FileCtrl.TDirectoryListBox.FDirectory 13327
FileCtrl.TDirectoryListBox.FileCompareText 13331
FileCtrl.TDirectoryListBox.FileList 13350
FileCtrl.TDirectoryListBox.FOnChange 13328
FileCtrl.TDirectoryListBox.Font 13353
FileCtrl.TDirectoryListBox.GetItemPath 13331

FileCtrl.TDirectoryListBox.ImeMode 13354
FileCtrl.TDirectoryListBox.ImeName 13354
FileCtrl.TDirectoryListBox.IntegralHeight 13355
FileCtrl.TDirectoryListBox.ItemHeight 13355
FileCtrl.TDirectoryListBox.KeyPress 13331
FileCtrl.TDirectoryListBox.Notification 13332
FileCtrl.TDirectoryListBox.OnChange 13367
FileCtrl.TDirectoryListBox.OnClick 13369
FileCtrl.TDirectoryListBox.OnContextPopup 13371
FileCtrl.TDirectoryListBox.OnDblClick 13371
FileCtrl.TDirectoryListBox.OnDragDrop 13372
FileCtrl.TDirectoryListBox.OnDragOver 13374
FileCtrl.TDirectoryListBox.OnEndDrag 13376
FileCtrl.TDirectoryListBox.OnEnter 13377
FileCtrl.TDirectoryListBox.OnExit 13380
FileCtrl.TDirectoryListBox.OnKeyDown 13381
FileCtrl.TDirectoryListBox.OnKeyPress 13384
FileCtrl.TDirectoryListBox.OnKeyUp 13385
FileCtrl.TDirectoryListBox.OnMouseActivate 13387
FileCtrl.TDirectoryListBox.OnMouseDown 13388
FileCtrl.TDirectoryListBox.OnMouseEnter 13390
FileCtrl.TDirectoryListBox.OnMouseLeave 13391
FileCtrl.TDirectoryListBox.OnMouseMove 13391
FileCtrl.TDirectoryListBox.OnMouseUp 13393
FileCtrl.TDirectoryListBox.OnStartDrag 13395
FileCtrl.TDirectoryListBox.OpenCurrent 13332
FileCtrl.TDirectoryListBox.OpenedBMP 13328
FileCtrl.TDirectoryListBox.ParentColor 13356
FileCtrl.TDirectoryListBox.ParentCtl3D 13356
FileCtrl.TDirectoryListBox.ParentFont 13357
FileCtrl.TDirectoryListBox.ParentShowHint 13357
FileCtrl.TDirectoryListBox.PopupMenu 13358
FileCtrl.TDirectoryListBox.PreserveCase 13362
FileCtrl.TDirectoryListBox.ReadBitmaps 13333
FileCtrl.TDirectoryListBox.ReadDirectoryNames 13333
FileCtrl.TDirectoryListBox.ShowHint 13363
FileCtrl.TDirectoryListBox.TabOrder 13364
FileCtrl.TDirectoryListBox.TabStop 13365
FileCtrl.TDirectoryListBox.Update 13333
FileCtrl.TDirectoryListBox.Visible 13365
FileCtrl.TDriveComboBox 13396
FileCtrl.TDriveComboBox.Align 13414
FileCtrl.TDriveComboBox.Anchors 13417
FileCtrl.TDriveComboBox.AutoComplete 13418
FileCtrl.TDriveComboBox.AutoDropDown 13418
FileCtrl.TDriveComboBox.BevelEdges 13419
FileCtrl.TDriveComboBox.BevelInner 13419
FileCtrl.TDriveComboBox.BevelKind 13420
FileCtrl.TDriveComboBox.BevelOuter 13420
FileCtrl.TDriveComboBox.BuildList 13411
FileCtrl.TDriveComboBox.CDROMBMP 13410
FileCtrl.TDriveComboBox.Click 13411
FileCtrl.TDriveComboBox.Color 13420
FileCtrl.TDriveComboBox.Constraints 13421
FileCtrl.TDriveComboBox.Create 13409
FileCtrl.TDriveComboBox.CreateWnd 13412
FileCtrl.TDriveComboBox.Ctl3D 13422
FileCtrl.TDriveComboBox.Destroy 13409
FileCtrl.TDriveComboBox.DirList 13422
FileCtrl.TDriveComboBox.DragCursor 13424
FileCtrl.TDriveComboBox.DragMode 13424
FileCtrl.TDriveComboBox.DrawItem 13412
FileCtrl.TDriveComboBox.Drive 13426
FileCtrl.TDriveComboBox.Enabled 13427
FileCtrl.TDriveComboBox.FixedBMP 13410
FileCtrl.TDriveComboBox.FloppyBMP 13410
FileCtrl.TDriveComboBox.Font 13427
FileCtrl.TDriveComboBox.ImeMode 13428
FileCtrl.TDriveComboBox.ImeName 13429
FileCtrl.TDriveComboBox.NetworkBMP 13410
FileCtrl.TDriveComboBox.Notification 13413
FileCtrl.TDriveComboBox.OnChange 13442
FileCtrl.TDriveComboBox.OnClick 13444
FileCtrl.TDriveComboBox.OnContextPopup 13446
FileCtrl.TDriveComboBox.OnDblClick 13446
FileCtrl.TDriveComboBox.OnDragDrop 13447
FileCtrl.TDriveComboBox.OnDragOver 13449
FileCtrl.TDriveComboBox.OnDropDown 13451

FileCtrl.TDriveComboBox.OnEndDrag 13453
FileCtrl.TDriveComboBox.OnEnter 13454
FileCtrl.TDriveComboBox.OnExit 13456
FileCtrl.TDriveComboBox.OnKeyDown 13458
FileCtrl.TDriveComboBox.OnKeyPress 13461
FileCtrl.TDriveComboBox.OnKeyUp 13462
FileCtrl.TDriveComboBox.OnMouseEnter 13464
FileCtrl.TDriveComboBox.OnMouseLeave 13464
FileCtrl.TDriveComboBox.OnStartDrag 13464
FileCtrl.TDriveComboBox.ParentColor 13429
FileCtrl.TDriveComboBox.ParentCtl3D 13430
FileCtrl.TDriveComboBox.ParentFont 13430
FileCtrl.TDriveComboBox.ParentShowHint 13431
FileCtrl.TDriveComboBox.PopupMenu 13431
FileCtrl.TDriveComboBox.RAMBMP 13411
FileCtrl.TDriveComboBox.ShowHint 13436
FileCtrl.TDriveComboBox.TabOrder 13437
FileCtrl.TDriveComboBox.TabStop 13438
FileCtrl.TDriveComboBox.Text 13438
FileCtrl.TDriveComboBox.TextCase 13439
FileCtrl.TDriveComboBox.Visible 13440
FileCtrl.TDriveType 13620
FileCtrl.TDriveType enumeration 13620
FileCtrl.TFileAttr 13620
FileCtrl.TFileAttr enumeration 13620
FileCtrl.TFileListBox 13465
FileCtrl.TFileListBox.Align 13488
FileCtrl.TFileListBox.Anchors 13492
FileCtrl.TFileListBox.ApplyFilePath 13482
FileCtrl.TFileListBox.AutoComplete 13492
FileCtrl.TFileListBox.BevelEdges 13493
FileCtrl.TFileListBox.BevelInner 13493
FileCtrl.TFileListBox.BevelKind 13493
FileCtrl.TFileListBox.BevelOuter 13494
FileCtrl.TFileListBox.Change 13484
FileCtrl.TFileListBox.Click 13484
FileCtrl.TFileListBox.Color 13494
FileCtrl.TFileListBox.Constraints 13495
FileCtrl.TFileListBox.Create 13478
FileCtrl.TFileListBox.CreateWnd 13484
FileCtrl.TFileListBox.Ctl3D 13496
FileCtrl.TFileListBox.Destroy 13479
FileCtrl.TFileListBox.DirBMP 13479
FileCtrl.TFileListBox.Directory 13496
FileCtrl.TFileListBox.DragCursor 13497
FileCtrl.TFileListBox.DragMode 13498
FileCtrl.TFileListBox.DrawItem 13485
FileCtrl.TFileListBox.Drive 13499
FileCtrl.TFileListBox.Enabled 13500
FileCtrl.TFileListBox.ExeBMP 13480
FileCtrl.TFileListBox.ExtendedSelect 13501
FileCtrl.TFileListBox.FDirectory 13480
FileCtrl.TFileListBox.FDirList 13480
FileCtrl.TFileListBox.FFileEdit 13480
FileCtrl.TFileListBox.FFileType 13480
FileCtrl.TFileListBox.FFilterCombo 13481
FileCtrl.TFileListBox.FileEdit 13501
FileCtrl.TFileListBox.FileName 13502
FileCtrl.TFileListBox.FileType 13503
FileCtrl.TFileListBox.FLastSel 13481
FileCtrl.TFileListBox.FMask 13481
FileCtrl.TFileListBox.FOnChange 13481
FileCtrl.TFileListBox.Font 13503
FileCtrl.TFileListBox.FShowGlyphs 13482
FileCtrl.TFileListBox.GetFilePath 13486
FileCtrl.TFileListBox.ImeMode 13504
FileCtrl.TFileListBox.ImeName 13505
FileCtrl.TFileListBox.IntegralHeight 13505
FileCtrl.TFileListBox.ItemHeight 13506
FileCtrl.TFileListBox.Mask 13506
FileCtrl.TFileListBox.MultiSelect 13507
FileCtrl.TFileListBox.Notification 13486
FileCtrl.TFileListBox.OnChange 13519
FileCtrl.TFileListBox.OnClick 13520
FileCtrl.TFileListBox.OnContextPopup 13521
FileCtrl.TFileListBox.OnDblClick 13522
FileCtrl.TFileListBox.OnDragDrop 13522
FileCtrl.TFileListBox.OnDragOver 13525

FileCtrl.TFileListBox.OnEndDrag 13527
FileCtrl.TFileListBox.OnEnter 13528
FileCtrl.TFileListBox.OnExit 13531
FileCtrl.TFileListBox.OnKeyDown 13532
FileCtrl.TFileListBox.OnKeyPress 13535
FileCtrl.TFileListBox.OnKeyUp 13536
FileCtrl.TFileListBox.OnMouseActivate 13538
FileCtrl.TFileListBox.OnMouseDown 13539
FileCtrl.TFileListBox.OnMouseEnter 13541
FileCtrl.TFileListBox.OnMouseLeave 13541
FileCtrl.TFileListBox.OnMouseMove 13541
FileCtrl.TFileListBox.OnMouseUp 13544
FileCtrl.TFileListBox.OnStartDrag 13546
FileCtrl.TFileListBox.ParentColor 13508
FileCtrl.TFileListBox.ParentCtl3D 13508
FileCtrl.TFileListBox.ParentFont 13509
FileCtrl.TFileListBox.ParentShowHint 13509
FileCtrl.TFileListBox.PopupMenu 13510
FileCtrl.TFileListBox.ReadBitmaps 13486
FileCtrl.TFileListBox.ReadFileNames 13487
FileCtrl.TFileListBox.ShowGlyphs 13514
FileCtrl.TFileListBox.ShowHint 13515
FileCtrl.TFileListBox.TabOrder 13516
FileCtrl.TFileListBox.TabStop 13517
FileCtrl.TFileListBox.UnknownBMP 13482
FileCtrl.TFileListBox.Update 13487
FileCtrl.TFileListBox.Visible 13517
FileCtrl.TFileType 13624
FileCtrl.TFileType type 13624
FileCtrl.TFilterComboBox 13547
FileCtrl.TFilterComboBox.Align 13563
FileCtrl.TFilterComboBox.Anchors 13567
FileCtrl.TFilterComboBox.AutoComplete 13567
FileCtrl.TFilterComboBox.AutoDropDown 13567
FileCtrl.TFilterComboBox.BevelEdges 13568
FileCtrl.TFilterComboBox.BevelInner 13568
FileCtrl.TFilterComboBox.BevelKind 13569
FileCtrl.TFilterComboBox.BevelOuter 13569
FileCtrl.TFilterComboBox.BuildList 13560
FileCtrl.TFilterComboBox.Change 13560
FileCtrl.TFilterComboBox.Click 13561
FileCtrl.TFilterComboBox.Color 13570
FileCtrl.TFilterComboBox.Constraints 13570
FileCtrl.TFilterComboBox.Create 13559
FileCtrl.TFilterComboBox.CreateWnd 13561
FileCtrl.TFilterComboBox.Ctl3D 13571
FileCtrl.TFilterComboBox.Destroy 13560
FileCtrl.TFilterComboBox.DragCursor 13571
FileCtrl.TFilterComboBox.DragMode 13572
FileCtrl.TFilterComboBox.Enabled 13573
FileCtrl.TFilterComboBox.FileList 13574
FileCtrl.TFilterComboBox.Filter 13575
FileCtrl.TFilterComboBox.Font 13576
FileCtrl.TFilterComboBox.ImeMode 13577
FileCtrl.TFilterComboBox.ImeName 13578
FileCtrl.TFilterComboBox.Mask 13579
FileCtrl.TFilterComboBox.Notification 13562
FileCtrl.TFilterComboBox.OnChange 13592
FileCtrl.TFilterComboBox.OnClick 13594
FileCtrl.TFilterComboBox.OnContextPopup 13596
FileCtrl.TFilterComboBox.OnDblClick 13596
FileCtrl.TFilterComboBox.OnDragDrop 13597
FileCtrl.TFilterComboBox.OnDragOver 13599
FileCtrl.TFilterComboBox.OnDropDown 13601
FileCtrl.TFilterComboBox.OnEndDrag 13603
FileCtrl.TFilterComboBox.OnEnter 13604
FileCtrl.TFilterComboBox.OnExit 13606
FileCtrl.TFilterComboBox.OnKeyDown 13608
FileCtrl.TFilterComboBox.OnKeyPress 13611
FileCtrl.TFilterComboBox.OnKeyUp 13612
FileCtrl.TFilterComboBox.OnMouseEnter 13614
FileCtrl.TFilterComboBox.OnMouseLeave 13614
FileCtrl.TFilterComboBox.OnStartDrag 13614
FileCtrl.TFilterComboBox.ParentColor 13580
FileCtrl.TFilterComboBox.ParentCtl3D 13580
FileCtrl.TFilterComboBox.ParentFont 13580
FileCtrl.TFilterComboBox.ParentShowHint 13581
FileCtrl.TFilterComboBox.PopupMenu 13581

FileCtrl.TFilterComboBox.ShowHint 13586
FileCtrl.TFilterComboBox.TabOrder 13587
FileCtrl.TFilterComboBox.TabStop 13588
FileCtrl.TFilterComboBox.Text 13588
FileCtrl.TFilterComboBox.Visible 13590
FileCtrl.TSelectDirExtOpt 13621
FileCtrl.TSelectDirExtOpt enumeration 13621
FileCtrl.TSelectDirExtOpts 13624
FileCtrl.TSelectDirExtOpts type 13624
FileCtrl.TSelectDirOpt 13622
FileCtrl.TSelectDirOpt enumeration 13622
FileCtrl.TSelectDirOpts 13625
FileCtrl.TSelectDirOpts type 13625
FileCtrl.TTextCase 13623
FileCtrl.TTextCase enumeration 13623
FileCtrl.WNTYPE_DRIVE 13626
FileCtrl.WNTYPE_DRIVE constant 13626
fkAggregate enumeration member 7611
fkCalculated enumeration member 7611
fkCommand enumeration member 16198
fkData enumeration member 7611
fkHandle enumeration member 16198
fkInternalCalc enumeration member 7611
fkLookup enumeration member 7611
fkShortCut enumeration member 16198
fldAttrArea enumeration member 11712
fldIsChanged enumeration member 11712
fldIsNull enumeration member 11712
fldIsUnChanged enumeration member 11712
fmAlternate enumeration member 14634
fmGet enumeration member 17091
fmoAllBaseMethods enumeration member 15485
fmoNoBaseMethods enumeration member 15485
fmoRTTIBaseMethods enumeration member 15485
fmPost enumeration member 17091
FMTBcd 13627
FMTBcd namespace 13627
Classes 13630
Constants 13651
Functions 13632
Structs, Records, Enums 13649
Types 13650
FMTBcd._DefaultDecimals 13651
FMTBcd._DefaultDecimals constant 13651
FMTBcd._NoDecimal 13651
FMTBcd._NoDecimal constant 13651
FMTBcd.BcdAdd 13633
FMTBcd.BcdAdd function 13633
FMTBcd.BcdCompare 13633
FMTBcd.BcdCompare function 13633
FMTBcd.BcdDivide 13634
FMTBcd.BcdDivide function 13634
FMTBcd.BcdFromBytes 13635
FMTBcd.BcdFromBytes function 13635
FMTBcd.BcdMultiply 13635
FMTBcd.BcdMultiply function 13635
FMTBcd.BcdPrecision 13636
FMTBcd.BcdPrecision function 13636
FMTBcd.BcdScale 13636
FMTBcd.BcdScale function 13636
FMTBcd.BcdSubtract 13637
FMTBcd.BcdSubtract function 13637
FMTBcd.BcdToBytes 13637
FMTBcd.BcdToBytes function 13637
FMTBcd.BCDToCurr 13637
FMTBcd.BCDToCurr function 13637
FMTBcd.BcdToDouble 13638
FMTBcd.BcdToDouble function 13638
FMTBcd.BcdToInteger 13638
FMTBcd.BcdToInteger function 13638
FMTBcd.BcdToStr 13639
FMTBcd.BcdToStr function 13639
FMTBcd.BcdToStrF 13639
FMTBcd.BcdToStrF function 13639
FMTBcd.CurrToBCD 13640
FMTBcd.CurrToBCD function 13640
FMTBcd.DefaultFMTBcdScale 13652
FMTBcd.DefaultFMTBcdScale constant 13652

FMTBcd.DoubleToBcd 13641
FMTBcd.DoubleToBcd function 13641
FMTBcd.EBcdException 13630
FMTBcd.EBcdOverflowException 13630
FMTBcd.FormatBcd 13641
FMTBcd.FormatBcd function 13641
FMTBcd.IntegerToBcd 13642
FMTBcd.IntegerToBcd function 13642
FMTBcd.IsBcdNegative 13642
FMTBcd.IsBcdNegative function 13642
FMTBcd.MaxBcdPrecision 13652
FMTBcd.MaxBcdPrecision constant 13652
FMTBcd.MaxBcdScale 13652
FMTBcd.MaxBcdScale constant 13652
FMTBcd.MaxFMTBcdDigits 13653
FMTBcd.MaxFMTBcdDigits constant 13653
FMTBcd.MaxFMTBcdFractionSize 13653
FMTBcd.MaxFMTBcdFractionSize constant 13653
FMTBcd.MaxStringDigits 13653
FMTBcd.MaxStringDigits constant 13653
FMTBcd.NormalizeBcd 13643
FMTBcd.NormalizeBcd function 13643
FMTBcd.NullBcd 13654
FMTBcd.NullBcd constant 13654
FMTBcd.OldBCDToCurr 13643
FMTBcd.OldBCDToCurr function 13643
FMTBcd.OldCurrToBCD 13644
FMTBcd.OldCurrToBCD function 13644
FMTBcd.PBcd 13650
FMTBcd.PBcd type 13650
FMTBcd.RoundAt 13644
FMTBcd.RoundAt function 13644
FMTBcd.SizeOfFraction 13654
FMTBcd.SizeOfFraction constant 13654
FMTBcd.SizeOfTBCD 13654
FMTBcd.SizeOfTBCD constant 13654
FMTBcd.StrToBcd 13644
FMTBcd.StrToBcd function 13644
FMTBcd.TBcd 13649
FMTBcd.TBcd record 13649
FMTBcd.TNibbleArray 13650
FMTBcd.TNibbleArray type 13650
FMTBcd.TryStrToBcd 13645
FMTBcd.TryStrToBcd function 13645
FMTBcd.VarFMTBcd 13646
FMTBcd.VarFMTBcd function 13646
FMTBcd.VarFMTBcdCreate 13646
FMTBcd.VarFMTBcdCreate function 13646
FMTBcd.VarIsFMTBcd 13647
FMTBcd.VarIsFMTBcd function 13647
FMTBcd.VarToBcd 13647
FMTBcd.VarToBcd function 13647
fmWinding enumeration member 14634
foBlobs enumeration member 8537
foCaseInsensitive enumeration member 7615
foDetails enumeration member 8537
foNoPartialCompare enumeration member 7615
forAccept enumeration member 11551
forDefault enumeration member 11551
foRecord enumeration member 8537
ForeignKeyColumnsColumns class 2149
about ForeignKeyColumnsColumns class 2149
CatalogName 2150
ColumnName 2151
ForeignKeyColumnsColumns fields 2150
ForeignKeyColumnsColumns Members 2150
ForeignKeyName 2151
Ordinal 2151
PrimaryCatalogName 2151
PrimaryColumnName 2151
PrimaryKeyName 2152
PrimarySchemaName 2152
PrimaryTableName 2152
SchemaName 2152
TableName 2152
ForeignKeysColumns class 2153
about ForeignKeysColumns class 2153
CatalogName 2154

ForeignKeyName 2154
ForeignKeysColumns fields 2153
ForeignKeysColumns Members 2153
SchemaName 2154
TableName 2154
Forms 13657
Forms namespace 13657
Classes 13660
Functions 14342
Interfaces 14334
Structs, Records, Enums 14353
Types 14369
Variables 14383
Forms.AllocateHWnd 14342
Forms.AllocateHWnd function 14342
Forms.Application 14383
Forms.Application variable 14383
Forms.Ctl3DBtnWndProc 14383
Forms.Ctl3DBtnWndProc variable 14383
Forms.Ctl3DCtlColorEx 14384
Forms.Ctl3DCtlColorEx variable 14384
Forms.Ctl3DDlgFramePaint 14384
Forms.Ctl3DDlgFramePaint variable 14384
Forms.DeallocateHWnd 14343
Forms.DeallocateHWnd function 14343
Forms.DisableTaskWindows 14343
Forms.DisableTaskWindows function 14343
Forms.DoneCtl3D 14344
Forms.DoneCtl3D function 14344
Forms.EnableTaskWindows 14344
Forms.EnableTaskWindows function 14344
Forms.ForegroundTask 14344
Forms.ForegroundTask function 14344
Forms.FreeObjectInstance 14345
Forms.FreeObjectInstance function 14345
Forms.GetApplicationMainFormOnTaskBar 14345
Forms.GetApplicationMainFormOnTaskBar function 14345
Forms.GetCustomFormGlassFrame 14345
Forms.GetCustomFormGlassFrame function 14345
Forms.GetParentForm 14346
Forms.GetParentForm function 14346
Forms.HintWindowClass 14384
Forms.HintWindowClass variable 14384
Forms.IDesignerHook 14334
Forms.IDesignerHook.Form 14339
Forms.IDesignerHook.GetCustomForm 14335
Forms.IDesignerHook.GetIsControl 14336
Forms.IDesignerHook.GetRoot 14336
Forms.IDesignerHook.IsControl 14339
Forms.IDesignerHook.IsDesignMsg 14336
Forms.IDesignerHook.PaintGrid 14337
Forms.IDesignerHook.PaintMenu 14337
Forms.IDesignerHook.SetCustomForm 14337
Forms.IDesignerHook.SetIsControl 14338
Forms.IDesignerHook.UniqueName 14338
Forms.IDesignerHook.ValidateRename 14338
Forms.InitCtl3D 14346
Forms.InitCtl3D function 14346
Forms.IOleForm 14340
Forms.IOleForm.OnDestroy 14340
Forms.IOleForm.OnResize 14341
Forms.IsAccel 14346
Forms.IsAccel function 14346
Forms.KeyboardStateToShiftState 14347
Forms.KeyboardStateToShiftState function 14347
Forms.KeyDataToShiftState 14348
Forms.KeyDataToShiftState function 14348
Forms.KeysToShiftState 14348
Forms.KeysToShiftState function 14348
Forms.MakeObjectInstance 14349
Forms.MakeObjectInstance function 14349
Forms.PCursorRec 14369
Forms.PCursorRec type 14369
Forms.PHintInfo 14370
Forms.PHintInfo type 14370
Forms.RestoreFocusState 14349
Forms.RestoreFocusState function 14349
Forms.SaveFocusState 14349

Forms.SaveFocusState function 14349
Forms.Screen 14385
Forms.Screen variable 14385
Forms.SetApplicationMainFormOnTaskBar 14350
Forms.SetApplicationMainFormOnTaskBar function 14350
Forms.SetAutoSubClass 14350
Forms.SetAutoSubClass function 14350
Forms.SetCustomFormGlassFrame 14350
Forms.SetCustomFormGlassFrame function 14350
Forms.SetLayeredWindowAttributes 14385
Forms.SetLayeredWindowAttributes variable 14385
Forms.Subclass3DDlg 14351
Forms.Subclass3DDlg function 14351
Forms.Subclass3DWnd 14351
Forms.Subclass3DWnd function 14351
Forms.TActiveFormBorderStyle 14353
Forms.TActiveFormBorderStyle enumeration 14353
Forms.TApplication 13660
Forms.TApplication.ActionUpdateDelay 13704
Forms.TApplication.ActivateHint 13668
Forms.TApplication.Active 13704
Forms.TApplication.ActiveFormHandle 13705
Forms.TApplication.AddPopupForm 13669
Forms.TApplication.AllowTesting 13705
Forms.TApplication.AutoDragDocking 13705
Forms.TApplication.BiDiKeyboard 13706
Forms.TApplication.BiDiMode 13706
Forms.TApplication.BringToFront 13669
Forms.TApplication.CancelHint 13669
Forms.TApplication.ControlDestroyed 13670
Forms.TApplication.Create 13666
Forms.TApplication.CreateForm 13670
Forms.TApplication.CreateHandle 13672
Forms.TApplication.CurrentHelpFile 13707
Forms.TApplication.Destroy 13667
Forms.TApplication.DialogHandle 13707
Forms.TApplication.DoApplicationIdle 13673
Forms.TApplication.EnumAllWindowsOnActivateHint 13708
Forms.TApplication.ExecuteAction 13673
Forms.TApplication.ExeName 13709
Forms.TApplication.Handle 13710
Forms.TApplication.HandleException 13674
Forms.TApplication.HandleMessage 13674
Forms.TApplication.HelpCommand 13678
Forms.TApplication.HelpContext 13679
Forms.TApplication.HelpFile 13711
Forms.TApplication.HelpJump 13679
Forms.TApplication.HelpKeyword 13680
Forms.TApplication.HelpShowTableOfContents 13680
Forms.TApplication.HelpSystem 13711
Forms.TApplication.HideHint 13681
Forms.TApplication.Hint 13712
Forms.TApplication.HintColor 13715
Forms.TApplication.HintHidePause 13715
Forms.TApplication.HintMouseMessage 13681
Forms.TApplication.HintPause 13715
Forms.TApplication.HintShortCuts 13716
Forms.TApplication.HintShortPause 13716
Forms.TApplication.HookMainWindow 13681
Forms.TApplication.HookSynchronizeWakeup 13682
Forms.TApplication.Icon 13717
Forms.TApplication.Idle 13682
Forms.TApplication.Initialize 13682
Forms.TApplication.IsDlgMsg 13685
Forms.TApplication.IsHintMsg 13685
Forms.TApplication.IsKeyMsg 13685
Forms.TApplication.IsMDIMsg 13685
Forms.TApplication.IsPreProcessMessage 13685
Forms.TApplication.IsRightToLeft 13686
Forms.TApplication.IsShortCut 13686
Forms.TApplication.MainForm 13718
Forms.TApplication.MainFormHandle 13719
Forms.TApplication.MainFormOnTaskBar 13719
Forms.TApplication.MessageBox 13686
Forms.TApplication.Minimize 13688
Forms.TApplication.ModalFinished 13689
Forms.TApplication.ModalLevel 13720
Forms.TApplication.ModalPopupMode 13720

Forms.TApplication.ModalStarted 13689
Forms.TApplication.NonBiDiKeyboard 13720
Forms.TApplication.NormalizeAllTopMosts 13690
Forms.TApplication.NormalizeTopMosts 13690
Forms.TApplication.OnActionExecute 13725
Forms.TApplication.OnActionUpdate 13726
Forms.TApplication.OnActivate 13726
Forms.TApplication.OnDeactivate 13727
Forms.TApplication.OnException 13728
Forms.TApplication.OnGetActiveFormHandle 13729
Forms.TApplication.OnGetMainFormHandle 13729
Forms.TApplication.OnHelp 13730
Forms.TApplication.OnHint 13731
Forms.TApplication.OnIdle 13733
Forms.TApplication.OnMessage 13737
Forms.TApplication.OnMinimize 13739
Forms.TApplication.OnModalBegin 13740
Forms.TApplication.OnModalEnd 13740
Forms.TApplication.OnRestore 13741
Forms.TApplication.OnSettingChange 13741
Forms.TApplication.OnShortCut 13741
Forms.TApplication.OnShowHint 13742
Forms.TApplication.PopupControlWnd 13721
Forms.TApplication.ProcessMessages 13691
Forms.TApplication.RemovePopupForm 13694
Forms.TApplication.Restore 13694
Forms.TApplication.RestoreTopMosts 13695
Forms.TApplication.Run 13696
Forms.TApplication.ShowException 13698
Forms.TApplication.ShowHint 13721
Forms.TApplication.ShowMainForm 13722
Forms.TApplication.Terminate 13699
Forms.TApplication.Terminated 13723
Forms.TApplication.Title 13723
Forms.TApplication.UnhookMainWindow 13701
Forms.TApplication.UnhookSynchronizeWakeup 13701
Forms.TApplication.UpdateAction 13701
Forms.TApplication.UpdateFormatSettings 13724
Forms.TApplication.UpdateMetricSettings 13724
Forms.TApplication.UseRightToLeftAlignment 13702
Forms.TApplication.UseRightToLeftReading 13702
Forms.TApplication.UseRightToLeftScrollBar 13703
Forms.TBorderIcon 14354
Forms.TBorderIcon enumeration 14354
Forms.TBorderIcons 14370
Forms.TBorderIcons type 14370
Forms.TBorderStyle 14371
Forms.TBorderStyle type 14371
Forms.TCloseAction 14355
Forms.TCloseAction enumeration 14355
Forms.TCloseEvent 14371
Forms.TCloseEvent type 14371
Forms.TCloseQueryEvent 14372
Forms.TCloseQueryEvent type 14372
Forms.TCMHintShow 14355
Forms.TCMHintShow record 14355
Forms.TCMHintShowPause 14356
Forms.TCMHintShowPause record 14356
Forms.TControlScrollBar 13744
Forms.TControlScrollBar.Assign 13746
Forms.TControlScrollBar.ButtonSize 13748
Forms.TControlScrollBar.ChangeBiDiPosition 13746
Forms.TControlScrollBar.Color 13748
Forms.TControlScrollBar.Increment 13748
Forms.TControlScrollBar.IsScrollBarVisible 13747
Forms.TControlScrollBar.Kind 13750
Forms.TControlScrollBar.Margin 13750
Forms.TControlScrollBar.ParentColor 13751
Forms.TControlScrollBar.Position 13752
Forms.TControlScrollBar.Range 13753
Forms.TControlScrollBar.ScrollPos 13756
Forms.TControlScrollBar.Size 13756
Forms.TControlScrollBar.Smooth 13756
Forms.TControlScrollBar.Style 13757
Forms.TControlScrollBar.ThumbSize 13757
Forms.TControlScrollBar.Tracking 13757
Forms.TControlScrollBar.Visible 13758
Forms.TCursorRec 14356

Forms.TCursorRec record 14356
Forms.TCustomActiveForm 13759
Forms.TCustomActiveForm.ActiveControl 13775
Forms.TCustomActiveForm.Anchors 13776
Forms.TCustomActiveForm.AutoScroll 13777
Forms.TCustomActiveForm.AutoSize 13777
Forms.TCustomActiveForm.AxBorderStyle 13777
Forms.TCustomActiveForm.BorderWidth 13778
Forms.TCustomActiveForm.Caption 13778
Forms.TCustomActiveForm.Color 13781
Forms.TCustomActiveForm.Constraints 13782
Forms.TCustomActiveForm.Create 13773
Forms.TCustomActiveForm.CreateParams 13773
Forms.TCustomActiveForm.Font 13782
Forms.TCustomActiveForm.Height 13783
Forms.TCustomActiveForm.HorzScrollBar 13784
Forms.TCustomActiveForm.KeyPreview 13785
Forms.TCustomActiveForm.OldCreateOrder 13788
Forms.TCustomActiveForm.OnActivate 13799
Forms.TCustomActiveForm.OnClick 13803
Forms.TCustomActiveForm.OnContextPopup 13805
Forms.TCustomActiveForm.OnCreate 13805
Forms.TCustomActiveForm.OnDblClick 13807
Forms.TCustomActiveForm.OnDeactivate 13807
Forms.TCustomActiveForm.OnDestroy 13808
Forms.TCustomActiveForm.OnDragDrop 13809
Forms.TCustomActiveForm.OnDragOver 13811
Forms.TCustomActiveForm.OnKeyDown 13814
Forms.TCustomActiveForm.OnKeyPress 13816
Forms.TCustomActiveForm.OnKeyUp 13817
Forms.TCustomActiveForm.OnMouseActivate 13819
Forms.TCustomActiveForm.OnMouseDown 13820
Forms.TCustomActiveForm.OnMouseEnter 13822
Forms.TCustomActiveForm.OnMouseLeave 13823
Forms.TCustomActiveForm.OnMouseMove 13823
Forms.TCustomActiveForm.OnMouseUp 13825
Forms.TCustomActiveForm.OnPaint 13827
Forms.TCustomActiveForm.Padding 13789
Forms.TCustomActiveForm.PixelsPerInch 13789
Forms.TCustomActiveForm.PopupMenu 13789
Forms.TCustomActiveForm.PrintScale 13794
Forms.TCustomActiveForm.Scaled 13794
Forms.TCustomActiveForm.ShowHint 13795
Forms.TCustomActiveForm.VertScrollBar 13796
Forms.TCustomActiveForm.Visible 13797
Forms.TCustomActiveForm.WantChildKey 13774
Forms.TCustomActiveForm.Width 13798
Forms.TCustomDockForm 13829
Forms.TCustomDockForm.AutoScroll 13845
Forms.TCustomDockForm.BorderStyle 13846
Forms.TCustomDockForm.Create 13843
Forms.TCustomDockForm.DoAddDockClient 13843
Forms.TCustomDockForm.DoRemoveDockClient 13844
Forms.TCustomDockForm.FormStyle 13846
Forms.TCustomDockForm.GetSiteInfo 13844
Forms.TCustomDockForm.Loaded 13845
Forms.TCustomDockForm.PixelsPerInch 13848
Forms.TCustomForm 13849
Forms.TCustomForm.Action 13895
Forms.TCustomForm.Activate 13865
Forms.TCustomForm.Active 13896
Forms.TCustomForm.ActiveChanged 13865
Forms.TCustomForm.ActiveControl 13896
Forms.TCustomForm.ActiveMDIChild 13897
Forms.TCustomForm.ActiveOleControl 13897
Forms.TCustomForm.AfterConstruction 13866
Forms.TCustomForm.AlignControls 13866
Forms.TCustomForm.AlphaBlend 13898
Forms.TCustomForm.AlphaBlendValue 13898
Forms.TCustomForm.AutoScroll 13899
Forms.TCustomForm.BeforeDestruction 13866
Forms.TCustomForm.BeginAutoDrag 13867
Forms.TCustomForm.BorderIcons 13899
Forms.TCustomForm.BorderStyle 13900
Forms.TCustomForm.Canvas 13900
Forms.TCustomForm.Caption 13902
Forms.TCustomForm.ChangeScale 13867
Forms.TCustomForm.ClientHandle 13904

Forms.TCustomForm.ClientHeight 13905
Forms.TCustomForm.ClientWidth 13905
Forms.TCustomForm.Close 13867
Forms.TCustomForm.CloseModal 13868
Forms.TCustomForm.CloseQuery 13868
Forms.TCustomForm.Color 13908
Forms.TCustomForm.Create 13861
Forms.TCustomForm.CreateNew 13862
Forms.TCustomForm.CreateParams 13868
Forms.TCustomForm.CreateWindowHandle 13869
Forms.TCustomForm.CreateWnd 13869
Forms.TCustomForm.Ctl3D 13909
Forms.TCustomForm.Deactivate 13870
Forms.TCustomForm.DefaultHandler 13870
Forms.TCustomForm.DefaultMonitor 13909
Forms.TCustomForm.DefineProperties 13871
Forms.TCustomForm.DefocusControl 13871
Forms.TCustomForm.Designer 13910
Forms.TCustomForm.Destroy 13862
Forms.TCustomForm.DestroyHandle 13871
Forms.TCustomForm.DestroyWindowHandle 13872
Forms.TCustomForm.Dock 13872
Forms.TCustomForm.DoClose 13873
Forms.TCustomForm.DoCreate 13873
Forms.TCustomForm.DoDestroy 13873
Forms.TCustomForm.DoDock 13874
Forms.TCustomForm.DoHide 13874
Forms.TCustomForm.DoShow 13875
Forms.TCustomForm.DropTarget 13910
Forms.TCustomForm.FFormState 13863
Forms.TCustomForm.FocusControl 13875
Forms.TCustomForm.Font 13910
Forms.TCustomForm.FormState 13911
Forms.TCustomForm.FormStyle 13912
Forms.TCustomForm.GetBorderIconStyles 13875
Forms.TCustomForm.GetBorderStyles 13876
Forms.TCustomForm.GetChildren 13876
Forms.TCustomForm.GetClientRect 13876
Forms.TCustomForm.GetFloating 13877
Forms.TCustomForm.GetFormImage 13877
Forms.TCustomForm.GetOwnerWindow 13878
Forms.TCustomForm.GlassFrame 13914
Forms.TCustomForm.HandleCreateException 13878
Forms.TCustomForm.Height 13915
Forms.TCustomForm.HelpFile 13916
Forms.TCustomForm.Hide 13879
Forms.TCustomForm.HorzScrollBar 13916
Forms.TCustomForm.Icon 13917
Forms.TCustomForm.IsShortCut 13880
Forms.TCustomForm.KeyPreview 13918
Forms.TCustomForm.Left 13920
Forms.TCustomForm.Loaded 13880
Forms.TCustomForm.MakeFullyVisible 13881
Forms.TCustomForm.MDIChildCount 13921
Forms.TCustomForm.MDIChildren 13922
Forms.TCustomForm.Menu 13923
Forms.TCustomForm.ModalResult 13924
Forms.TCustomForm.Monitor 13926
Forms.TCustomForm.MouseWheelHandler 13881
Forms.TCustomForm.Notification 13882
Forms.TCustomForm.ObjectMenuItem 13926
Forms.TCustomForm.OldCreateOrder 13926
Forms.TCustomForm.OleFormObject 13927
Forms.TCustomForm.OnActivate 13943
Forms.TCustomForm.OnCanResize 13947
Forms.TCustomForm.OnClick 13947
Forms.TCustomForm.OnClose 13949
Forms.TCustomForm.OnCloseQuery 13950
Forms.TCustomForm.OnCreate 13951
Forms.TCustomForm.OnDblClick 13953
Forms.TCustomForm.OnDeactivate 13953
Forms.TCustomForm.OnDestroy 13953
Forms.TCustomForm.OnDragDrop 13954
Forms.TCustomForm.OnDragOver 13957
Forms.TCustomForm.OnHelp 13959
Forms.TCustomForm.OnHide 13960
Forms.TCustomForm.OnKeyDown 13962
Forms.TCustomForm.OnKeyPress 13964

Forms.TCustomForm.OnKeyUp 13966
Forms.TCustomForm.OnMouseActivate 13967
Forms.TCustomForm.OnMouseDown 13968
Forms.TCustomForm.OnMouseMove 13971
Forms.TCustomForm.OnMouseUp 13973
Forms.TCustomForm.OnPaint 13975
Forms.TCustomForm.OnResize 13976
Forms.TCustomForm.OnShortCut 13977
Forms.TCustomForm.OnShow 13977
Forms.TCustomForm.Paint 13882
Forms.TCustomForm.PaintWindow 13882
Forms.TCustomForm.PaletteChanged 13883
Forms.TCustomForm.ParentFont 13927
Forms.TCustomForm.PixelsPerInch 13928
Forms.TCustomForm.PopupChildren 13928
Forms.TCustomForm.PopupMenu 13928
Forms.TCustomForm.PopupMode 13933
Forms.TCustomForm.PopupParent 13933
Forms.TCustomForm.Position 13933
Forms.TCustomForm.Print 13883
Forms.TCustomForm.PrintScale 13934
Forms.TCustomForm.QueryInterface 13884
Forms.TCustomForm.ReadState 13884
Forms.TCustomForm.RecreateAsPopup 13884
Forms.TCustomForm.Release 13885
Forms.TCustomForm.RequestAlign 13885
Forms.TCustomForm.Resizing 13885
Forms.TCustomForm.Scaled 13935
Forms.TCustomForm.ScreenSnap 13935
Forms.TCustomForm.SendCancelMode 13886
Forms.TCustomForm.SetChildOrder 13886
Forms.TCustomForm.SetFocus 13886
Forms.TCustomForm.SetFocusedControl 13887
Forms.TCustomForm.SetParent 13887
Forms.TCustomForm.SetParentBiDiMode 13887
Forms.TCustomForm.Show 13888
Forms.TCustomForm.ShowModal 13890
Forms.TCustomForm.SnapBuffer 13936
Forms.TCustomForm.TileMode 13936
Forms.TCustomForm.Top 13937
Forms.TCustomForm.TransparentColor 13938
Forms.TCustomForm.TransparentColorValue 13939
Forms.TCustomForm.UpdateActions 13891
Forms.TCustomForm.UpdateGlassFrame 13892
Forms.TCustomForm.UpdateWindowState 13892
Forms.TCustomForm.ValidateRename 13892
Forms.TCustomForm.VertScrollBar 13939
Forms.TCustomForm.Visible 13940
Forms.TCustomForm.VisibleChanging 13893
Forms.TCustomForm.WantChildKey 13893
Forms.TCustomForm.Width 13941
Forms.TCustomForm.WindowMenu 13941
Forms.TCustomForm.WindowState 13942
Forms.TCustomForm.WndProc 13893
Forms.TCustomFormClass 14372
Forms.TCustomFormClass type 14372
Forms.TCustomFrame 13978
Forms.TCustomFrame.Create 13989
Forms.TCustomFrame.CreateParams 13990
Forms.TCustomFrame.GetChildren 13990
Forms.TCustomFrame.PaintWindow 13991
Forms.TCustomFrame.SetParent 13991
Forms.TCustomFrameClass 14373
Forms.TCustomFrameClass type 14373
Forms.TDefaultMonitor 14357
Forms.TDefaultMonitor enumeration 14357
Forms.TExceptionEvent 14373
Forms.TExceptionEvent type 14373
Forms.TFocusState 14374
Forms.TFocusState type 14374
Forms.TForm 13992
Forms.TForm.Action 14011
Forms.TForm.ActiveControl 14012
Forms.TForm.ActiveMDIChild 14013
Forms.TForm.Align 14013
Forms.TForm.AlphaBlend 14017
Forms.TForm.AlphaBlendValue 14017
Forms.TForm.Anchors 14017

Forms.TForm.ArrangeIcons 14006
Forms.TForm.AutoScroll 14018
Forms.TForm.AutoSize 14018
Forms.TForm.BiDiMode 14019
Forms.TForm.BorderIcons 14019
Forms.TForm.BorderStyle 14020
Forms.TForm.BorderWidth 14021
Forms.TForm.Caption 14021
Forms.TForm.Cascade 14007
Forms.TForm.ClientHandle 14024
Forms.TForm.ClientHeight 14024
Forms.TForm.ClientWidth 14025
Forms.TForm.Color 14027
Forms.TForm.Constraints 14028
Forms.TForm.Ctl3D 14028
Forms.TForm.DefaultMonitor 14029
Forms.TForm.DockManager 14029
Forms.TForm.DockSite 14031
Forms.TForm.DragKind 14032
Forms.TForm.DragMode 14033
Forms.TForm.Enabled 14034
Forms.TForm.Font 14035
Forms.TForm.FormStyle 14036
Forms.TForm.Height 14038
Forms.TForm.HelpFile 14039
Forms.TForm.HorzScrollBar 14040
Forms.TForm.Icon 14041
Forms.TForm.KeyPreview 14041
Forms.TForm.MDIChildCount 14044
Forms.TForm.MDIChildren 14044
Forms.TForm.Menu 14045
Forms.TForm.Next 14008
Forms.TForm.ObjectMenuItem 14047
Forms.TForm.OldCreateOrder 14047
Forms.TForm.OnActivate 14065
Forms.TForm.OnAlignInsertBefore 14069
Forms.TForm.OnAlignPosition 14069
Forms.TForm.OnCanResize 14070
Forms.TForm.OnClick 14070
Forms.TForm.OnClose 14072
Forms.TForm.OnCloseQuery 14073
Forms.TForm.OnConstrainedResize 14074
Forms.TForm.OnContextPopup 14075
Forms.TForm.OnCreate 14076
Forms.TForm.OnDblClick 14077
Forms.TForm.OnDeactivate 14078
Forms.TForm.OnDestroy 14078
Forms.TForm.OnDockDrop 14079
Forms.TForm.OnDockOver 14081
Forms.TForm.OnDragDrop 14083
Forms.TForm.OnDragOver 14085
Forms.TForm.OnEndDock 14088
Forms.TForm.OnGetSiteInfo 14088
Forms.TForm.OnHelp 14090
Forms.TForm.OnHide 14091
Forms.TForm.OnKeyDown 14093
Forms.TForm.OnKeyPress 14095
Forms.TForm.OnKeyUp 14096
Forms.TForm.OnMouseActivate 14098
Forms.TForm.OnMouseDown 14099
Forms.TForm.OnMouseEnter 14101
Forms.TForm.OnMouseLeave 14102
Forms.TForm.OnMouseMove 14102
Forms.TForm.OnMouseUp 14104
Forms.TForm.OnMouseWheel 14107
Forms.TForm.OnMouseWheelDown 14107
Forms.TForm.OnMouseWheelUp 14107
Forms.TForm.OnPaint 14108
Forms.TForm.OnResize 14109
Forms.TForm.OnShortCut 14109
Forms.TForm.OnShow 14110
Forms.TForm.OnStartDock 14111
Forms.TForm.OnUnDock 14111
Forms.TForm.Padding 14047
Forms.TForm.ParentBiDiMode 14048
Forms.TForm.ParentFont 14048
Forms.TForm.PixelsPerInch 14049
Forms.TForm.PopupMenu 14049

Forms.TForm.PopupMode 14054
Forms.TForm.PopupParent 14054
Forms.TForm.Position 14055
Forms.TForm.Previous 14009
Forms.TForm.PrintScale 14056
Forms.TForm.Scaled 14056
Forms.TForm.ScreenSnap 14056
Forms.TForm.ShowHint 14057
Forms.TForm.SnapBuffer 14058
Forms.TForm.Tile 14009
Forms.TForm.TileMode 14059
Forms.TForm.TransparentColor 14060
Forms.TForm.TransparentColorValue 14060
Forms.TForm.UseDockManager 14060
Forms.TForm.VertScrollBar 14061
Forms.TForm.Visible 14062
Forms.TForm.Width 14063
Forms.TForm.WindowMenu 14063
Forms.TForm.WindowState 14063
Forms.TFormBorderStyle 14357
Forms.TFormBorderStyle enumeration 14357
Forms.TFormClass 14374
Forms.TFormClass type 14374
Forms.TFormState 14374
Forms.TFormState type 14374
Forms.TFormStyle 14358
Forms.TFormStyle enumeration 14358
Forms.TFrame 14114
Forms.TFrame.Align 14126
Forms.TFrame.Anchors 14130
Forms.TFrame.AutoScroll 14130
Forms.TFrame.AutoSize 14131
Forms.TFrame.BiDiMode 14131
Forms.TFrame.Color 14132
Forms.TFrame.Constraints 14132
Forms.TFrame.Ctl3D 14133
Forms.TFrame.DockSite 14133
Forms.TFrame.DragCursor 14134
Forms.TFrame.DragKind 14135
Forms.TFrame.DragMode 14135
Forms.TFrame.Enabled 14137
Forms.TFrame.Font 14137
Forms.TFrame.OnAlignInsertBefore 14150
Forms.TFrame.OnAlignPosition 14151
Forms.TFrame.OnCanResize 14152
Forms.TFrame.OnClick 14152
Forms.TFrame.OnConstrainedResize 14154
Forms.TFrame.OnContextPopup 14155
Forms.TFrame.OnDblClick 14155
Forms.TFrame.OnDockDrop 14156
Forms.TFrame.OnDockOver 14158
Forms.TFrame.OnDragDrop 14160
Forms.TFrame.OnDragOver 14162
Forms.TFrame.OnEndDock 14164
Forms.TFrame.OnEndDrag 14165
Forms.TFrame.OnEnter 14166
Forms.TFrame.OnExit 14168
Forms.TFrame.OnGetSiteInfo 14170
Forms.TFrame.OnMouseActivate 14172
Forms.TFrame.OnMouseDown 14173
Forms.TFrame.OnMouseEnter 14175
Forms.TFrame.OnMouseLeave 14175
Forms.TFrame.OnMouseMove 14175
Forms.TFrame.OnMouseUp 14178
Forms.TFrame.OnMouseWheel 14180
Forms.TFrame.OnMouseWheelDown 14180
Forms.TFrame.OnMouseWheelUp 14181
Forms.TFrame.OnResize 14181
Forms.TFrame.OnStartDock 14182
Forms.TFrame.OnStartDrag 14182
Forms.TFrame.OnUnDock 14183
Forms.TFrame.Padding 14138
Forms.TFrame.ParentBackground 14139
Forms.TFrame.ParentBiDiMode 14139
Forms.TFrame.ParentColor 14139
Forms.TFrame.ParentCtl3D 14140
Forms.TFrame.ParentFont 14140
Forms.TFrame.ParentShowHint 14141

Forms.TFrame.PopupMenu 14141
Forms.TFrame.ShowHint 14146
Forms.TFrame.TabOrder 14147
Forms.TFrame.TabStop 14148
Forms.TFrame.Visible 14148
Forms.TGetHandleEvent 14375
Forms.TGetHandleEvent type 14375
Forms.TGlassFrame 14186
Forms.TGlassFrame.Assign 14188
Forms.TGlassFrame.Bottom 14190
Forms.TGlassFrame.Change 14189
Forms.TGlassFrame.Create 14187
Forms.TGlassFrame.Enabled 14190
Forms.TGlassFrame.FrameExtended 14189
Forms.TGlassFrame.IntersectsControl 14189
Forms.TGlassFrame.Left 14190
Forms.TGlassFrame.OnChange 14191
Forms.TGlassFrame.Right 14190
Forms.TGlassFrame.SheetOfGlass 14190
Forms.TGlassFrame.Top 14190
Forms.THelpEvent 14375
Forms.THelpEvent type 14375
Forms.THintInfo 14359
Forms.THintInfo record 14359
Forms.TIdleEvent 14376
Forms.TIdleEvent type 14376
Forms.TMessageEvent 14376
Forms.TMessageEvent type 14376
Forms.TMonitor 14191
Forms.TMonitor.BoundsRect 14193
Forms.TMonitor.Handle 14194
Forms.TMonitor.Height 14194
Forms.TMonitor.Left 14194
Forms.TMonitor.MonitorNum 14195
Forms.TMonitor.Primary 14195
Forms.TMonitor.Top 14196
Forms.TMonitor.Width 14196
Forms.TMonitor.WorkareaRect 14197
Forms.TMonitorDefaultTo 14361
Forms.TMonitorDefaultTo enumeration 14361
Forms.TPopupForm 14362
Forms.TPopupForm record 14362
Forms.TPopupFormArray 14377
Forms.TPopupFormArray type 14377
Forms.TPopupMode 14362
Forms.TPopupMode enumeration 14362
Forms.TPopupWnd 14363
Forms.TPopupWnd record 14363
Forms.TPopupWndArray 14377
Forms.TPopupWndArray type 14377
Forms.TPosition 14363
Forms.TPosition enumeration 14363
Forms.TPrintScale 14364
Forms.TPrintScale enumeration 14364
Forms.TScreen 14197
Forms.TScreen.ActiveControl 14206
Forms.TScreen.ActiveCustomForm 14207
Forms.TScreen.ActiveForm 14207
Forms.TScreen.Create 14202
Forms.TScreen.Cursor 14209
Forms.TScreen.CursorCount 14212
Forms.TScreen.Cursors 14212
Forms.TScreen.CustomFormCount 14214
Forms.TScreen.CustomForms 14215
Forms.TScreen.DataModuleCount 14216
Forms.TScreen.DataModules 14216
Forms.TScreen.DefaultIme 14216
Forms.TScreen.DefaultKbLayout 14217
Forms.TScreen.DesktopHeight 14217
Forms.TScreen.DesktopLeft 14218
Forms.TScreen.DesktopRect 14218
Forms.TScreen.DesktopTop 14219
Forms.TScreen.DesktopWidth 14219
Forms.TScreen.Destroy 14202
Forms.TScreen.DisableAlign 14203
Forms.TScreen.EnableAlign 14203
Forms.TScreen.FocusedForm 14220
Forms.TScreen.Fonts 14220

Forms.TScreen.FormCount 14221
Forms.TScreen.Forms 14223
Forms.TScreen.Height 14225
Forms.TScreen.HintFont 14226
Forms.TScreen.IconFont 14226
Forms.TScreen.Imes 14227
Forms.TScreen.MenuFont 14227
Forms.TScreen.MonitorCount 14228
Forms.TScreen.MonitorFromPoint 14203
Forms.TScreen.MonitorFromRect 14204
Forms.TScreen.MonitorFromWindow 14204
Forms.TScreen.Monitors 14228
Forms.TScreen.OnActiveControlChange 14234
Forms.TScreen.OnActiveFormChange 14236
Forms.TScreen.PixelsPerInch 14228
Forms.TScreen.PrimaryMonitor 14229
Forms.TScreen.Realign 14205
Forms.TScreen.ResetFonts 14205
Forms.TScreen.SaveFocusedList 14229
Forms.TScreen.Width 14229
Forms.TScreen.WorkAreaHeight 14231
Forms.TScreen.WorkAreaLeft 14231
Forms.TScreen.WorkAreaRect 14232
Forms.TScreen.WorkAreaTop 14232
Forms.TScreen.WorkAreaWidth 14233
Forms.TScrollBarInc 14377
Forms.TScrollBarInc type 14377
Forms.TScrollBarKind 14365
Forms.TScrollBarKind enumeration 14365
Forms.TScrollBarStyle 14366
Forms.TScrollBarStyle enumeration 14366
Forms.TScrollBox 14238
Forms.TScrollBox.Align 14251
Forms.TScrollBox.Anchors 14255
Forms.TScrollBox.AutoScroll 14255
Forms.TScrollBox.AutoSize 14256
Forms.TScrollBox.BevelEdges 14256
Forms.TScrollBox.BevelInner 14257
Forms.TScrollBox.BevelKind 14257
Forms.TScrollBox.BevelOuter 14258
Forms.TScrollBox.BevelWidth 14258
Forms.TScrollBox.BiDiMode 14258
Forms.TScrollBox.BorderStyle 14259
Forms.TScrollBox.Color 14259
Forms.TScrollBox.Constraints 14260
Forms.TScrollBox.Create 14249
Forms.TScrollBox.CreateParams 14250
Forms.TScrollBox.Ctl3D 14261
Forms.TScrollBox.DockSite 14261
Forms.TScrollBox.DragCursor 14262
Forms.TScrollBox.DragKind 14262
Forms.TScrollBox.DragMode 14263
Forms.TScrollBox.Enabled 14264
Forms.TScrollBox.Font 14265
Forms.TScrollBox.OnCanResize 14278
Forms.TScrollBox.OnClick 14278
Forms.TScrollBox.OnConstrainedResize 14280
Forms.TScrollBox.OnContextPopup 14281
Forms.TScrollBox.OnDblClick 14282
Forms.TScrollBox.OnDockDrop 14282
Forms.TScrollBox.OnDockOver 14284
Forms.TScrollBox.OnDragDrop 14286
Forms.TScrollBox.OnDragOver 14288
Forms.TScrollBox.OnEndDock 14291
Forms.TScrollBox.OnEndDrag 14291
Forms.TScrollBox.OnEnter 14292
Forms.TScrollBox.OnExit 14294
Forms.TScrollBox.OnGetSiteInfo 14296
Forms.TScrollBox.OnMouseActivate 14298
Forms.TScrollBox.OnMouseDown 14299
Forms.TScrollBox.OnMouseEnter 14301
Forms.TScrollBox.OnMouseLeave 14302
Forms.TScrollBox.OnMouseMove 14302
Forms.TScrollBox.OnMouseUp 14304
Forms.TScrollBox.OnMouseWheel 14306
Forms.TScrollBox.OnMouseWheelDown 14307
Forms.TScrollBox.OnMouseWheelUp 14307
Forms.TScrollBox.OnResize 14307

Forms.TScrollBox.OnStartDock 14308
Forms.TScrollBox.OnStartDrag 14309
Forms.TScrollBox.OnUnDock 14309
Forms.TScrollBox.Padding 14266
Forms.TScrollBox.PaintWindow 14250
Forms.TScrollBox.ParentBackground 14266
Forms.TScrollBox.ParentBiDiMode 14266
Forms.TScrollBox.ParentColor 14267
Forms.TScrollBox.ParentCtl3D 14267
Forms.TScrollBox.ParentFont 14268
Forms.TScrollBox.ParentShowHint 14268
Forms.TScrollBox.PopupMenu 14269
Forms.TScrollBox.ShowHint 14273
Forms.TScrollBox.TabOrder 14275
Forms.TScrollBox.TabStop 14275
Forms.TScrollBox.Visible 14276
Forms.TScrollingWinControl 14312
Forms.TScrollingWinControl.AdjustClientRect 14324
Forms.TScrollingWinControl.AlignControls 14325
Forms.TScrollingWinControl.AutoScroll 14329
Forms.TScrollingWinControl.AutoScrollEnabled 14325
Forms.TScrollingWinControl.AutoScrollInView 14325
Forms.TScrollingWinControl.ChangeScale 14326
Forms.TScrollingWinControl.Create 14323
Forms.TScrollingWinControl.CreateParams 14326
Forms.TScrollingWinControl.CreateWnd 14327
Forms.TScrollingWinControl.Destroy 14323
Forms.TScrollingWinControl.DisableAutoRange 14327
Forms.TScrollingWinControl.DoFlipChildren 14327
Forms.TScrollingWinControl.EnableAutoRange 14328
Forms.TScrollingWinControl.HorzScrollBar 14329
Forms.TScrollingWinControl.OnAlignInsertBefore 14332
Forms.TScrollingWinControl.OnAlignPosition 14332
Forms.TScrollingWinControl.Resizing 14328
Forms.TScrollingWinControl.ScrollInView 14328
Forms.TScrollingWinControl.VertScrollBar 14330
Forms.TSetLayeredWindowAttributes 14378
Forms.TSetLayeredWindowAttributes type 14378
Forms.TSettingChangeEvent 14378
Forms.TSettingChangeEvent type 14378
Forms.TShortCutEvent 14381
Forms.TShortCutEvent type 14381
Forms.TShowAction 14366
Forms.TShowAction enumeration 14366
Forms.TShowHintEvent 14381
Forms.TShowHintEvent type 14381
Forms.TTileMode 14367
Forms.TTileMode enumeration 14367
Forms.TTimerMode 14367
Forms.TTimerMode enumeration 14367
Forms.TWindowHook 14382
Forms.TWindowHook type 14382
Forms.TWindowState 14368
Forms.TWindowState enumeration 14368
Forms.ValidParentForm 14351
Forms.ValidParentForm function 14351
forRefuse enumeration member 11551
fpDefault enumeration member 14635
fpFixed enumeration member 14635
fpLoginRequired enumeration member 24688
fpPublished enumeration member 24688
fpVariable enumeration member 14635
frDisableMatchCase enumeration member 11553
frDisableUpDown enumeration member 11553
frDisableWholeWord enumeration member 11553
frDown enumeration member 11553
frFindNext enumeration member 11553
frHideMatchCase enumeration member 11553
frHideUpDown enumeration member 11553
frHideWholeWord enumeration member 11553
friend AnsiString __fastcall operator +(const char*, const AnsiString& rhs) 22127
friend class AutoCmd 22375
friend Currency __fastcall operator -(double lhs, const Currency& rhs) 22164
friend Currency __fastcall operator -(int lhs, const Currency& rhs) 22164
friend Currency __fastcall operator *(double lhs, const Currency& rhs) 22164
friend Currency __fastcall operator *(int lhs, const Currency&

rhs) 22164

friend Currency __fastcall operator /(double lhs, const Currency& rhs) 22165

friend Currency __fastcall operator /(int lhs, const Currency& rhs) 22165

friend Currency __fastcall operator +(double lhs, const Currency& rhs) 22165

friend Currency __fastcall operator +(int lhs, const Currency& rhs) 22165

frMatchCase enumeration member 11553

frReplace enumeration member 11553

frReplaceAll enumeration member 11553

frShowHelp enumeration member 11553

frWholeWord enumeration member 11553

fsBold enumeration member 14636

fsBorder enumeration member 14634

fsBottomTopLeftRight enumeration member 13269

fsBottomTopRightLeft enumeration member 13269

fsComboBox enumeration member 11552

fsEdit enumeration member 11552

fsItalic enumeration member 14636

fsLeftRightBottomTop enumeration member 13269

fsLeftRightTopBottom enumeration member 13269

fsMDIChild enumeration member 14358

fsMDIForm enumeration member 14358

fsNormal enumeration member 14358

fsrAccept enumeration member 11552

fsrDefault enumeration member 11552

fsRightLeftBottomTop enumeration member 13269

fsRightLeftTopBottom enumeration member 13269

fsrRefuse enumeration member 11552

fsStayOnTop enumeration member 14358

fsStrikeOut enumeration member 14636

fsSurface enumeration member 14634

fsTopBottomLeftRight enumeration member 13269

fsTopBottomRightLeft enumeration member 13269

fsUnderline enumeration member 14636

ftADT enumeration member 7612

ftArchive enumeration member 13620

ftArray enumeration member 7612

ftAutoInc enumeration member 7612

ftBCD enumeration member 7612

ftBlob enumeration member 7612

ftBoolean enumeration member 7612

ftBytes enumeration member 7612

ftComp enumeration member 23372

ftCurr enumeration member 23372

ftCurrency enumeration member 7612

ftCursor enumeration member 7612

ftDataSet enumeration member 7612

ftDate enumeration member 7612

ftDateTime enumeration member 7612

ftDBaseOle enumeration member 7612

ftDirectory enumeration member 13620

ftDouble enumeration member 23372

ftExtended enumeration member 23372

ftFixedChar enumeration member 7612

ftFixedWideChar enumeration member 7612

ftFloat enumeration member 7612

ftFMTBcd enumeration member 7612

ftFmtMemo enumeration member 7612

ftGraphic enumeration member 7612

ftGuid enumeration member 7612

ftHidden enumeration member 13620

ftIDispatch enumeration member 7612

ftInteger enumeration member 7612

ftInterface enumeration member 7612

ftLargeint enumeration member 7612

ftMemo enumeration member 7612

ftNormal enumeration member 13620

ftOraBlob enumeration member 7612

ftOraClob enumeration member 7612

ftOraInterval enumeration member 7612

ftOraTimeStamp enumeration member 7612

ftParadoxOle enumeration member 7612

ftReadOnly enumeration member 13620

ftReference enumeration member 7612

ftSingle enumeration member 23372

ftSmallint enumeration member 7612

ftString enumeration member 7612

ftSystem enumeration member 13620
ftTime enumeration member 7612
ftTimeStamp enumeration member 7612
ftTypedBinary enumeration member 7612
ftUnknown enumeration member 7612
ftVarBytes enumeration member 7612
ftVariant enumeration member 7612
ftVolumeID enumeration member 13620
ftWideMemo enumeration member 7612
ftWideString enumeration member 7612
ftWord enumeration member 7612
Function class 22184
    about Function class 22184
    Function 22186
    Function constructor 22186
    Function Members 22185
    Function methods 22186
    GetCallType 22187
    RequestResult 22187
fvCurrency enumeration member 23001
fvExtended enumeration member 23001
fxBoolean enumeration member 17840
fxCurrency enumeration member 17840
fxDate enumeration member 17840
fxDateTime enumeration member 17840
fxFloat enumeration member 17840
fxInteger enumeration member 17840
fxNone enumeration member 17840
fxString enumeration member 17840
fxTime enumeration member 17840

G

gbFirst enumeration member 7616
gbLast enumeration member 7616
gbMiddle enumeration member 7616
gbNone enumeration member 8257
gbRaised enumeration member 8257
gdHorizontal enumeration member 14757
gdVertical enumeration member 14757
gkAppend enumeration member 8257
gkCancel enumeration member 8257
gkDelete enumeration member 8257
gkDown enumeration member 8257
gkEditMode enumeration member 8257
gkEnd enumeration member 8257
gkHome enumeration member 8257
gkInsert enumeration member 8257
gkLeft enumeration member 8257
gkNextTab enumeration member 8257
gkNull enumeration member 8257
gkPageDown enumeration member 8257
gkPageUp enumeration member 8257
gkPriorTab enumeration member 8257
gkRight enumeration member 8257
gkScrollDown enumeration member 8257
gkScrollUp enumeration member 8257
gkUp enumeration member 8257
gmCurrent enumeration member 7615
gmNext enumeration member 7615
gmPrior enumeration member 7615
goAlwaysShowEditor enumeration member 14999
goColMoving enumeration member 14999
goColSizing enumeration member 14999
goDrawFocusSelected enumeration member 14999
goEditing enumeration member 14999
goFixedHorzLine enumeration member 14999
goFixedVertLine enumeration member 14999
goHorizontal enumeration member 8258
goHorzLine enumeration member 14999
goRangeSelect enumeration member 14999
goRowMoving enumeration member 14999
goRowSelect enumeration member 14999
goRowSizing enumeration member 14999
goTabs enumeration member 14999
goThumbTracking enumeration member 14999
goVertical enumeration member 8258
goVertLine enumeration member 14999
Graphics 14387

Graphics namespace 14387
Classes 14395
Constants 14659
Functions 14608
Interfaces 14606
Structs, Records, Enums 14630
Types 14645
Variables 14657
Graphics.AllocPatternBitmap 14608
Graphics.AllocPatternBitmap function 14608
Graphics.BytesPerScanline 14609
Graphics.BytesPerScanline function 14609
Graphics.CharsetToIdent 14609
Graphics.CharsetToIdent function 14609
Graphics.cl3DDkShadow 14663
Graphics.cl3DDkShadow constant 14663
Graphics.cl3DLight 14663
Graphics.cl3DLight constant 14663
Graphics.clActiveBorder 14664
Graphics.clActiveBorder constant 14664
Graphics.clActiveCaption 14664
Graphics.clActiveCaption constant 14664
Graphics.clAppWorkSpace 14665
Graphics.clAppWorkSpace constant 14665
Graphics.clAqua 14665
Graphics.clAqua constant 14665
Graphics.clBackground 14666
Graphics.clBackground constant 14666
Graphics.clBlack 14666
Graphics.clBlack constant 14666
Graphics.clBlue 14666
Graphics.clBlue constant 14666
Graphics.clBtnFace 14667
Graphics.clBtnFace constant 14667
Graphics.clBtnHighlight 14667
Graphics.clBtnHighlight constant 14667
Graphics.clBtnShadow 14668
Graphics.clBtnShadow constant 14668
Graphics.clBtnText 14668
Graphics.clBtnText constant 14668
Graphics.clCaptionText 14669
Graphics.clCaptionText constant 14669
Graphics.clCream 14669
Graphics.clCream constant 14669
Graphics.clDefault 14670
Graphics.clDefault constant 14670
Graphics.clDkGray 14670
Graphics.clDkGray constant 14670
Graphics.clFuchsia 14670
Graphics.clFuchsia constant 14670
Graphics.clGradientActiveCaption 14671
Graphics.clGradientActiveCaption constant 14671
Graphics.clGradientInactiveCaption 14671
Graphics.clGradientInactiveCaption constant 14671
Graphics.clGray 14672
Graphics.clGray constant 14672
Graphics.clGrayText 14672
Graphics.clGrayText constant 14672
Graphics.clGreen 14673
Graphics.clGreen constant 14673
Graphics.clHighlight 14673
Graphics.clHighlight constant 14673
Graphics.clHighlightText 14674
Graphics.clHighlightText constant 14674
Graphics.clHotLight 14674
Graphics.clHotLight constant 14674
Graphics.clInactiveBorder 14675
Graphics.clInactiveBorder constant 14675
Graphics.clInactiveCaption 14675
Graphics.clInactiveCaption constant 14675
Graphics.clInactiveCaptionText 14675
Graphics.clInactiveCaptionText constant 14675
Graphics.clInfoBk 14676
Graphics.clInfoBk constant 14676
Graphics.clInfoText 14676
Graphics.clInfoText constant 14676
Graphics.clLime 14677
Graphics.clLime constant 14677

Graphics.clLtGray 14677
Graphics.clLtGray constant 14677
Graphics.clMaroon 14678
Graphics.clMaroon constant 14678
Graphics.clMedGray 14678
Graphics.clMedGray constant 14678
Graphics.clMenu 14679
Graphics.clMenu constant 14679
Graphics.clMenuBar 14679
Graphics.clMenuBar constant 14679
Graphics.clMenuHighlight 14679
Graphics.clMenuHighlight constant 14679
Graphics.clMenuText 14680
Graphics.clMenuText constant 14680
Graphics.clMoneyGreen 14680
Graphics.clMoneyGreen constant 14680
Graphics.clNavy 14681
Graphics.clNavy constant 14681
Graphics.clNone 14681
Graphics.clNone constant 14681
Graphics.clOlive 14682
Graphics.clOlive constant 14682
Graphics.clPurple 14682
Graphics.clPurple constant 14682
Graphics.clRed 14683
Graphics.clRed constant 14683
Graphics.clScrollBar 14683
Graphics.clScrollBar constant 14683
Graphics.clSilver 14683
Graphics.clSilver constant 14683
Graphics.clSkyBlue 14684
Graphics.clSkyBlue constant 14684
Graphics.clSystemColor 14684
Graphics.clSystemColor constant 14684
Graphics.clTeal 14685
Graphics.clTeal constant 14685
Graphics.clWebAliceBlue 14685
Graphics.clWebAliceBlue constant 14685
Graphics.clWebAntiqueWhite 14686
Graphics.clWebAntiqueWhite constant 14686
Graphics.clWebAqua 14686
Graphics.clWebAqua constant 14686
Graphics.clWebAquamarine 14686
Graphics.clWebAquamarine constant 14686
Graphics.clWebAzure 14687
Graphics.clWebAzure constant 14687
Graphics.clWebBeige 14687
Graphics.clWebBeige constant 14687
Graphics.clWebBisque 14687
Graphics.clWebBisque constant 14687
Graphics.clWebBlack 14688
Graphics.clWebBlack constant 14688
Graphics.clWebBlanchedAlmond 14688
Graphics.clWebBlanchedAlmond constant 14688
Graphics.clWebBlue 14688
Graphics.clWebBlue constant 14688
Graphics.clWebBlueViolet 14689
Graphics.clWebBlueViolet constant 14689
Graphics.clWebBrown 14689
Graphics.clWebBrown constant 14689
Graphics.clWebBurlywood 14689
Graphics.clWebBurlywood constant 14689
Graphics.clWebCadetBlue 14690
Graphics.clWebCadetBlue constant 14690
Graphics.clWebChartreuse 14690
Graphics.clWebChartreuse constant 14690
Graphics.clWebChocolate 14690
Graphics.clWebChocolate constant 14690
Graphics.clWebCoral 14691
Graphics.clWebCoral constant 14691
Graphics.clWebCornFlowerBlue 14691
Graphics.clWebCornFlowerBlue constant 14691
Graphics.clWebCornSilk 14691
Graphics.clWebCornSilk constant 14691
Graphics.clWebCrimson 14692
Graphics.clWebCrimson constant 14692
Graphics.clWebCyan 14692
Graphics.clWebCyan constant 14692

Graphics.clWebDarkBlue 14692
Graphics.clWebDarkBlue constant 14692
Graphics.clWebDarkCyan 14693
Graphics.clWebDarkCyan constant 14693
Graphics.clWebDarkGoldenRod 14693
Graphics.clWebDarkGoldenRod constant 14693
Graphics.clWebDarkGray 14693
Graphics.clWebDarkGray constant 14693
Graphics.clWebDarkgreen 14694
Graphics.clWebDarkgreen constant 14694
Graphics.clWebDarkKhaki 14694
Graphics.clWebDarkKhaki constant 14694
Graphics.clWebDarkMagenta 14694
Graphics.clWebDarkMagenta constant 14694
Graphics.clWebDarkOliveGreen 14695
Graphics.clWebDarkOliveGreen constant 14695
Graphics.clWebDarkOrange 14695
Graphics.clWebDarkOrange constant 14695
Graphics.clWebDarkOrchid 14695
Graphics.clWebDarkOrchid constant 14695
Graphics.clWebDarkRed 14696
Graphics.clWebDarkRed constant 14696
Graphics.clWebDarkSalmon 14696
Graphics.clWebDarkSalmon constant 14696
Graphics.clWebDarkSeaGreen 14696
Graphics.clWebDarkSeaGreen constant 14696
Graphics.clWebDarkSlateBlue 14697
Graphics.clWebDarkSlateBlue constant 14697
Graphics.clWebDarkSlategray 14697
Graphics.clWebDarkSlategray constant 14697
Graphics.clWebDarkTurquoise 14697
Graphics.clWebDarkTurquoise constant 14697
Graphics.clWebDarkViolet 14698
Graphics.clWebDarkViolet constant 14698
Graphics.clWebDeepPink 14698
Graphics.clWebDeepPink constant 14698
Graphics.clWebDeepskyBlue 14698
Graphics.clWebDeepskyBlue constant 14698
Graphics.clWebDimGray 14699
Graphics.clWebDimGray constant 14699
Graphics.clWebDodgerBlue 14699
Graphics.clWebDodgerBlue constant 14699
Graphics.clWebFirebrick 14699
Graphics.clWebFirebrick constant 14699
Graphics.clWebFloralWhite 14700
Graphics.clWebFloralWhite constant 14700
Graphics.clWebForestGreen 14700
Graphics.clWebForestGreen constant 14700
Graphics.clWebFuchsia 14700
Graphics.clWebFuchsia constant 14700
Graphics.clWebGainsboro 14701
Graphics.clWebGainsboro constant 14701
Graphics.clWebGhostWhite 14701
Graphics.clWebGhostWhite constant 14701
Graphics.clWebGold 14701
Graphics.clWebGold constant 14701
Graphics.clWebGoldenRod 14702
Graphics.clWebGoldenRod constant 14702
Graphics.clWebGray 14702
Graphics.clWebGray constant 14702
Graphics.clWebGreen 14702
Graphics.clWebGreen constant 14702
Graphics.clWebGreenYellow 14703
Graphics.clWebGreenYellow constant 14703
Graphics.clWebHoneydew 14703
Graphics.clWebHoneydew constant 14703
Graphics.clWebHotPink 14703
Graphics.clWebHotPink constant 14703
Graphics.clWebIndianRed 14704
Graphics.clWebIndianRed constant 14704
Graphics.clWebIndigo 14704
Graphics.clWebIndigo constant 14704
Graphics.clWebIvory 14704
Graphics.clWebIvory constant 14704
Graphics.clWebKhaki 14705
Graphics.clWebKhaki constant 14705
Graphics.clWebLavender 14705
Graphics.clWebLavender constant 14705

Graphics.clWebLavenderBlush 14705
Graphics.clWebLavenderBlush constant 14705
Graphics.clWebLawnGreen 14706
Graphics.clWebLawnGreen constant 14706
Graphics.clWebLemonChiffon 14706
Graphics.clWebLemonChiffon constant 14706
Graphics.clWebLightBlue 14706
Graphics.clWebLightBlue constant 14706
Graphics.clWebLightCoral 14707
Graphics.clWebLightCoral constant 14707
Graphics.clWebLightCyan 14707
Graphics.clWebLightCyan constant 14707
Graphics.clWebLightGoldenrodYellow 14707
Graphics.clWebLightGoldenrodYellow constant 14707
Graphics.clWebLightGreen 14708
Graphics.clWebLightGreen constant 14708
Graphics.clWebLightgrey 14708
Graphics.clWebLightgrey constant 14708
Graphics.clWebLightPink 14708
Graphics.clWebLightPink constant 14708
Graphics.clWebLightSalmon 14709
Graphics.clWebLightSalmon constant 14709
Graphics.clWebLightSeaGreen 14709
Graphics.clWebLightSeaGreen constant 14709
Graphics.clWebLightSkyBlue 14709
Graphics.clWebLightSkyBlue constant 14709
Graphics.clWebLightSlateGray 14710
Graphics.clWebLightSlateGray constant 14710
Graphics.clWebLightSteelBlue 14710
Graphics.clWebLightSteelBlue constant 14710
Graphics.clWebLightYellow 14710
Graphics.clWebLightYellow constant 14710
Graphics.clWebLime 14711
Graphics.clWebLime constant 14711
Graphics.clWebLimeGreen 14711
Graphics.clWebLimeGreen constant 14711
Graphics.clWebLinen 14711
Graphics.clWebLinen constant 14711
Graphics.clWebMagenta 14712
Graphics.clWebMagenta constant 14712
Graphics.clWebMaroon 14712
Graphics.clWebMaroon constant 14712
Graphics.clWebMediumAquamarine 14712
Graphics.clWebMediumAquamarine constant 14712
Graphics.clWebMediumBlue 14713
Graphics.clWebMediumBlue constant 14713
Graphics.clWebMediumOrchid 14713
Graphics.clWebMediumOrchid constant 14713
Graphics.clWebMediumPurple 14713
Graphics.clWebMediumPurple constant 14713
Graphics.clWebMediumSeaGreen 14714
Graphics.clWebMediumSeaGreen constant 14714
Graphics.clWebMediumSlateBlue 14714
Graphics.clWebMediumSlateBlue constant 14714
Graphics.clWebMediumSpringGreen 14714
Graphics.clWebMediumSpringGreen constant 14714
Graphics.clWebMediumTurquoise 14715
Graphics.clWebMediumTurquoise constant 14715
Graphics.clWebMediumVioletRed 14715
Graphics.clWebMediumVioletRed constant 14715
Graphics.clWebMidnightBlue 14715
Graphics.clWebMidnightBlue constant 14715
Graphics.clWebMintcream 14716
Graphics.clWebMintcream constant 14716
Graphics.clWebMistyRose 14716
Graphics.clWebMistyRose constant 14716
Graphics.clWebMoccasin 14716
Graphics.clWebMoccasin constant 14716
Graphics.clWebNavajoWhite 14717
Graphics.clWebNavajoWhite constant 14717
Graphics.clWebNavy 14717
Graphics.clWebNavy constant 14717
Graphics.clWebOldLace 14717
Graphics.clWebOldLace constant 14717
Graphics.clWebOlive 14718
Graphics.clWebOlive constant 14718
Graphics.clWebOliveDrab 14718
Graphics.clWebOliveDrab constant 14718

Graphics.clWebOrange 14718
Graphics.clWebOrange constant 14718
Graphics.clWebOrangeRed 14719
Graphics.clWebOrangeRed constant 14719
Graphics.clWebOrchid 14719
Graphics.clWebOrchid constant 14719
Graphics.clWebPaleGoldenrod 14719
Graphics.clWebPaleGoldenrod constant 14719
Graphics.clWebPaleGreen 14720
Graphics.clWebPaleGreen constant 14720
Graphics.clWebPaleTurquoise 14720
Graphics.clWebPaleTurquoise constant 14720
Graphics.clWebPaleVioletRed 14720
Graphics.clWebPaleVioletRed constant 14720
Graphics.clWebPapayaWhip 14721
Graphics.clWebPapayaWhip constant 14721
Graphics.clWebPeachPuff 14721
Graphics.clWebPeachPuff constant 14721
Graphics.clWebPeru 14721
Graphics.clWebPeru constant 14721
Graphics.clWebPink 14722
Graphics.clWebPink constant 14722
Graphics.clWebPlum 14722
Graphics.clWebPlum constant 14722
Graphics.clWebPowderBlue 14722
Graphics.clWebPowderBlue constant 14722
Graphics.clWebPurple 14723
Graphics.clWebPurple constant 14723
Graphics.clWebRed 14723
Graphics.clWebRed constant 14723
Graphics.clWebRosyBrown 14723
Graphics.clWebRosyBrown constant 14723
Graphics.clWebRoyalBlue 14724
Graphics.clWebRoyalBlue constant 14724
Graphics.clWebSaddleBrown 14724
Graphics.clWebSaddleBrown constant 14724
Graphics.clWebSalmon 14724
Graphics.clWebSalmon constant 14724
Graphics.clWebSandyBrown 14725
Graphics.clWebSandyBrown constant 14725
Graphics.clWebSeaGreen 14725
Graphics.clWebSeaGreen constant 14725
Graphics.clWebSeashell 14725
Graphics.clWebSeashell constant 14725
Graphics.clWebSienna 14726
Graphics.clWebSienna constant 14726
Graphics.clWebSilver 14726
Graphics.clWebSilver constant 14726
Graphics.clWebSkyBlue 14726
Graphics.clWebSkyBlue constant 14726
Graphics.clWebSlateBlue 14727
Graphics.clWebSlateBlue constant 14727
Graphics.clWebSlateGray 14727
Graphics.clWebSlateGray constant 14727
Graphics.clWebSnow 14727
Graphics.clWebSnow constant 14727
Graphics.clWebSpringGreen 14728
Graphics.clWebSpringGreen constant 14728
Graphics.clWebSteelBlue 14728
Graphics.clWebSteelBlue constant 14728
Graphics.clWebTan 14728
Graphics.clWebTan constant 14728
Graphics.clWebTeal 14729
Graphics.clWebTeal constant 14729
Graphics.clWebThistle 14729
Graphics.clWebThistle constant 14729
Graphics.clWebTomato 14729
Graphics.clWebTomato constant 14729
Graphics.clWebTurquoise 14730
Graphics.clWebTurquoise constant 14730
Graphics.clWebViolet 14730
Graphics.clWebViolet constant 14730
Graphics.clWebWheat 14730
Graphics.clWebWheat constant 14730
Graphics.clWebWhite 14731
Graphics.clWebWhite constant 14731
Graphics.clWebWhiteSmoke 14731
Graphics.clWebWhiteSmoke constant 14731

Graphics.clWebYellow 14731
Graphics.clWebYellow constant 14731
Graphics.clWebYellowGreen 14732
Graphics.clWebYellowGreen constant 14732
Graphics.clWhite 14732
Graphics.clWhite constant 14732
Graphics.clWindow 14732
Graphics.clWindow constant 14732
Graphics.clWindowFrame 14733
Graphics.clWindowFrame constant 14733
Graphics.clWindowText 14733
Graphics.clWindowText constant 14733
Graphics.clYellow 14734
Graphics.clYellow constant 14734
Graphics.cmBlackness 14734
Graphics.cmBlackness constant 14734
Graphics.cmDstInvert 14735
Graphics.cmDstInvert constant 14735
Graphics.cmMergeCopy 14735
Graphics.cmMergeCopy constant 14735
Graphics.cmMergePaint 14736
Graphics.cmMergePaint constant 14736
Graphics.cmNotSrcCopy 14736
Graphics.cmNotSrcCopy constant 14736
Graphics.cmNotSrcErase 14737
Graphics.cmNotSrcErase constant 14737
Graphics.cmPatCopy 14737
Graphics.cmPatCopy constant 14737
Graphics.cmPatInvert 14738
Graphics.cmPatInvert constant 14738
Graphics.cmPatPaint 14738
Graphics.cmPatPaint constant 14738
Graphics.cmSrcAnd 14739
Graphics.cmSrcAnd constant 14739
Graphics.cmSrcCopy 14739
Graphics.cmSrcCopy constant 14739
Graphics.cmSrcErase 14740
Graphics.cmSrcErase constant 14740
Graphics.cmSrcInvert 14740
Graphics.cmSrcInvert constant 14740
Graphics.cmSrcPaint 14741
Graphics.cmSrcPaint constant 14741
Graphics.cmWhiteness 14741
Graphics.cmWhiteness constant 14741
Graphics.ColorToIdent 14610
Graphics.ColorToIdent function 14610
Graphics.ColorToRGB 14610
Graphics.ColorToRGB function 14610
Graphics.ColorToString 14611
Graphics.ColorToString function 14611
Graphics.CopyPalette 14612
Graphics.CopyPalette function 14612
Graphics.CreateGrayMappedBmp 14612
Graphics.CreateGrayMappedBmp function 14612
Graphics.CreateGrayMappedRes 14613
Graphics.CreateGrayMappedRes function 14613
Graphics.CreateMappedBmp 14614
Graphics.CreateMappedBmp function 14614
Graphics.CreateMappedRes 14614
Graphics.CreateMappedRes function 14614
Graphics.DDBsOnly 14657
Graphics.DDBsOnly variable 14657
Graphics.DefFontData 14658
Graphics.DefFontData variable 14658
Graphics.EInvalidGraphic 14395
Graphics.EInvalidGraphicOperation 14396
Graphics.ExtendedColorsCount 14742
Graphics.ExtendedColorsCount constant 14742
Graphics.FreeMemoryContexts 14615
Graphics.FreeMemoryContexts function 14615
Graphics.GetCharsetValues 14615
Graphics.GetCharsetValues function 14615
Graphics.GetColorValues 14619
Graphics.GetColorValues function 14619
Graphics.GetDefFontCharSet 14619
Graphics.GetDefFontCharSet function 14619
Graphics.GetDIB 14620
Graphics.GetDIB function 14620

Graphics.GetDIBSizes 14620
Graphics.GetDIBSizes function 14620
Graphics.GraphicExtension 14621
Graphics.GraphicExtension function 14621
Graphics.GraphicFileMask 14624
Graphics.GraphicFileMask function 14624
Graphics.GraphicFilter 14624
Graphics.GraphicFilter function 14624
Graphics.HENHMETAFILE 14645
Graphics.HENHMETAFILE type 14645
Graphics.HMETAFILE 14646
Graphics.HMETAFILE type 14646
Graphics.IChangeNotifier 14606
Graphics.IChangeNotifier.Changed 14607
Graphics.IdentToCharset 14627
Graphics.IdentToCharset function 14627
Graphics.IdentToColor 14627
Graphics.IdentToColor function 14627
Graphics.PaletteChanged 14628
Graphics.PaletteChanged function 14628
Graphics.PColor 14646
Graphics.PColor type 14646
Graphics.PCursorOrIcon 14646
Graphics.PCursorOrIcon type 14646
Graphics.PIconRec 14647
Graphics.PIconRec type 14647
Graphics.PResource 14647
Graphics.PResource type 14647
Graphics.rc3_Cursor 14743
Graphics.rc3_Cursor constant 14743
Graphics.rc3_Icon 14743
Graphics.rc3_Icon constant 14743
Graphics.rc3_StockIcon 14743
Graphics.rc3_StockIcon constant 14743
Graphics.StandardColorsCount 14744
Graphics.StandardColorsCount constant 14744
Graphics.StringToColor 14628
Graphics.StringToColor function 14628
Graphics.SystemPalette16 14658
Graphics.SystemPalette16 variable 14658
Graphics.TBitmap 14396
Graphics.TBitmap.Assign 14402
Graphics.TBitmap.Canvas 14419
Graphics.TBitmap.Changed 14403
Graphics.TBitmap.Create 14400
Graphics.TBitmap.Destroy 14401
Graphics.TBitmap.Dormant 14404
Graphics.TBitmap.Draw 14405
Graphics.TBitmap.FreeImage 14405
Graphics.TBitmap.GetEmpty 14407
Graphics.TBitmap.GetHeight 14407
Graphics.TBitmap.GetPalette 14407
Graphics.TBitmap.GetWidth 14407
Graphics.TBitmap.Handle 14420
Graphics.TBitmap.HandleAllocated 14407
Graphics.TBitmap.HandleNeeded 14408
Graphics.TBitmap.HandleType 14420
Graphics.TBitmap.IgnorePalette 14421
Graphics.TBitmap.LoadFromClipboardFormat 14408
Graphics.TBitmap.LoadFromResourceID 14410
Graphics.TBitmap.LoadFromResourceName 14410
Graphics.TBitmap.LoadFromStream 14412
Graphics.TBitmap.Mask 14412
Graphics.TBitmap.MaskHandle 14421
Graphics.TBitmap.MaskHandleNeeded 14412
Graphics.TBitmap.Monochrome 14421
Graphics.TBitmap.PaletteNeeded 14413
Graphics.TBitmap.PixelFormat 14423
Graphics.TBitmap.ReadData 14413
Graphics.TBitmap.ReleaseHandle 14413
Graphics.TBitmap.ReleaseMaskHandle 14414
Graphics.TBitmap.ReleasePalette 14415
Graphics.TBitmap.SaveToClipboardFormat 14415
Graphics.TBitmap.SaveToStream 14417
Graphics.TBitmap.ScanLine 14425
Graphics.TBitmap.SetHeight 14417
Graphics.TBitmap.SetPalette 14418
Graphics.TBitmap.SetSize 14418

Graphics.TBitmap.SetWidth 14418
Graphics.TBitmap.TransparentColor 14427
Graphics.TBitmap.TransparentMode 14428
Graphics.TBitmap.WriteData 14418
Graphics.TBitmapHandleType 14630
Graphics.TBitmapHandleType enumeration 14630
Graphics.TBitmapImage 14429
Graphics.TBitmapImage.Destroy 14430
Graphics.TBitmapImage.FreeHandle 14430
Graphics.TBrush 14430
Graphics.TBrush.Assign 14433
Graphics.TBrush.Bitmap 14436
Graphics.TBrush.Color 14437
Graphics.TBrush.Create 14432
Graphics.TBrush.Destroy 14433
Graphics.TBrush.GetBitmap 14434
Graphics.TBrush.GetColor 14434
Graphics.TBrush.GetHandle 14434
Graphics.TBrush.GetStyle 14434
Graphics.TBrush.Handle 14438
Graphics.TBrush.SetBitmap 14434
Graphics.TBrush.SetColor 14435
Graphics.TBrush.SetHandle 14435
Graphics.TBrush.SetStyle 14435
Graphics.TBrush.Style 14438
Graphics.TBrushData 14631
Graphics.TBrushData record 14631
Graphics.TBrushRecall 14439
Graphics.TBrushRecall.Create 14440
Graphics.TBrushStyle 14631
Graphics.TBrushStyle enumeration 14631
Graphics.TCanvas 14441
Graphics.TCanvas.Arc 14448
Graphics.TCanvas.Brush 14484
Graphics.TCanvas.BrushCopy 14449
Graphics.TCanvas.CanvasOrientation 14485
Graphics.TCanvas.Changed 14450
Graphics.TCanvas.Changing 14450
Graphics.TCanvas.Chord 14451
Graphics.TCanvas.ClipRect 14486
Graphics.TCanvas.CopyMode 14487
Graphics.TCanvas.CopyRect 14452
Graphics.TCanvas.Create 14443
Graphics.TCanvas.CreateHandle 14454
Graphics.TCanvas.Destroy 14446
Graphics.TCanvas.Draw 14454
Graphics.TCanvas.DrawFocusRect 14455
Graphics.TCanvas.Ellipse 14457
Graphics.TCanvas.FillRect 14458
Graphics.TCanvas.FloodFill 14459
Graphics.TCanvas.Font 14488
Graphics.TCanvas.FrameRect 14460
Graphics.TCanvas.Handle 14489
Graphics.TCanvas.HandleAllocated 14461
Graphics.TCanvas.LineTo 14462
Graphics.TCanvas.Lock 14463
Graphics.TCanvas.LockCount 14492
Graphics.TCanvas.MoveTo 14463
Graphics.TCanvas.OnChange 14496
Graphics.TCanvas.OnChanging 14496
Graphics.TCanvas.Pen 14492
Graphics.TCanvas.PenPos 14494
Graphics.TCanvas.Pie 14464
Graphics.TCanvas.Pixels 14494
Graphics.TCanvas.PolyBezier 14464
Graphics.TCanvas.PolyBezierTo 14465
Graphics.TCanvas.Polygon 14466
Graphics.TCanvas.Polyline 14467
Graphics.TCanvas.Rectangle 14468
Graphics.TCanvas.Refresh 14470
Graphics.TCanvas.RequiredState 14470
Graphics.TCanvas.RoundRect 14471
Graphics.TCanvas.StretchDraw 14472
Graphics.TCanvas.TextExtent 14474
Graphics.TCanvas.TextFlags 14495
Graphics.TCanvas.TextHeight 14474
Graphics.TCanvas.TextOut 14475
Graphics.TCanvas.TextRect 14480

Graphics.TCanvas.TextWidth 14481
Graphics.TCanvas.TryLock 14483
Graphics.TCanvas.Unlock 14483
Graphics.TCanvasOrientation 14632
Graphics.TCanvasOrientation enumeration 14632
Graphics.TCanvasState 14647
Graphics.TCanvasState type 14647
Graphics.TCanvasStates 14633
Graphics.TCanvasStates enumeration 14633
Graphics.TColor 14648
Graphics.TColor type 14648
Graphics.TCopyMode 14650
Graphics.TCopyMode type 14650
Graphics.TCursorOrIcon 14633
Graphics.TCursorOrIcon record 14633
Graphics.TFillMode 14634
Graphics.TFillMode enumeration 14634
Graphics.TFillStyle 14634
Graphics.TFillStyle enumeration 14634
Graphics.TFont 14497
Graphics.TFont.Assign 14500
Graphics.TFont.Changed 14502
Graphics.TFont.Charset 14506
Graphics.TFont.Color 14507
Graphics.TFont.Create 14499
Graphics.TFont.Destroy 14500
Graphics.TFont.FontAdapter 14508
Graphics.TFont.GetCharset 14502
Graphics.TFont.GetHandle 14503
Graphics.TFont.GetHeight 14503
Graphics.TFont.GetName 14503
Graphics.TFont.GetOrientation 14503
Graphics.TFont.GetPitch 14503
Graphics.TFont.GetSize 14504
Graphics.TFont.GetStyle 14504
Graphics.TFont.Handle 14509
Graphics.TFont.Height 14510
Graphics.TFont.Name 14510
Graphics.TFont.Orientation 14511
Graphics.TFont.Pitch 14512
Graphics.TFont.PixelsPerInch 14512
Graphics.TFont.SetCharset 14504
Graphics.TFont.SetColor 14504
Graphics.TFont.SetHandle 14505
Graphics.TFont.SetHeight 14505
Graphics.TFont.SetName 14505
Graphics.TFont.SetOrientation 14505
Graphics.TFont.SetPitch 14505
Graphics.TFont.SetSize 14506
Graphics.TFont.SetStyle 14506
Graphics.TFont.Size 14513
Graphics.TFont.Style 14513
Graphics.TFontCharset 14651
Graphics.TFontCharset type 14651
Graphics.TFontData 14635
Graphics.TFontData record 14635
Graphics.TFontDataName 14652
Graphics.TFontDataName type 14652
Graphics.TFontName 14653
Graphics.TFontName type 14653
Graphics.TFontPitch 14635
Graphics.TFontPitch enumeration 14635
Graphics.TFontRecall 14514
Graphics.TFontRecall.Create 14516
Graphics.TFontStyle 14636
Graphics.TFontStyle enumeration 14636
Graphics.TFontStyles 14653
Graphics.TFontStyles type 14653
Graphics.TFontStylesBase 14653
Graphics.TFontStylesBase type 14653
Graphics.TGraphic 14516
Graphics.TGraphic.Changed 14520
Graphics.TGraphic.Create 14519
Graphics.TGraphic.DefineProperties 14520
Graphics.TGraphic.Draw 14521
Graphics.TGraphic.Empty 14528
Graphics.TGraphic.Equals 14521
Graphics.TGraphic.GetEmpty 14521

Graphics.TGraphic.GetHeight 14521
Graphics.TGraphic.GetPalette 14522
Graphics.TGraphic.GetTransparent 14522
Graphics.TGraphic.GetWidth 14522
Graphics.TGraphic.Height 14528
Graphics.TGraphic.LoadFromClipboardFormat 14522
Graphics.TGraphic.LoadFromFile 14523
Graphics.TGraphic.LoadFromStream 14524
Graphics.TGraphic.Modified 14528
Graphics.TGraphic.OnChange 14531
Graphics.TGraphic.OnProgress 14531
Graphics.TGraphic.Palette 14529
Graphics.TGraphic.PaletteModified 14529
Graphics.TGraphic.Progress 14524
Graphics.TGraphic.ReadData 14524
Graphics.TGraphic.SaveToClipboardFormat 14525
Graphics.TGraphic.SaveToFile 14525
Graphics.TGraphic.SaveToStream 14526
Graphics.TGraphic.SetHeight 14526
Graphics.TGraphic.SetPalette 14527
Graphics.TGraphic.SetSize 14527
Graphics.TGraphic.SetTransparent 14527
Graphics.TGraphic.SetWidth 14527
Graphics.TGraphic.Transparent 14529
Graphics.TGraphic.Width 14530
Graphics.TGraphic.WriteData 14527
Graphics.TGraphicClass 14654
Graphics.TGraphicClass type 14654
Graphics.TGraphicsObject 14532
Graphics.TGraphicsObject.Changed 14534
Graphics.TGraphicsObject.HandleAllocated 14534
Graphics.TGraphicsObject.Lock 14534
Graphics.TGraphicsObject.OnChange 14536
Graphics.TGraphicsObject.OwnerCriticalSection 14536
Graphics.TGraphicsObject.Unlock 14535
Graphics.TIcon 14537
Graphics.TIcon.Assign 14541
Graphics.TIcon.Create 14539
Graphics.TIcon.Destroy 14541
Graphics.TIcon.Draw 14542
Graphics.TIcon.GetEmpty 14542
Graphics.TIcon.GetHeight 14542
Graphics.TIcon.GetWidth 14542
Graphics.TIcon.Handle 14546
Graphics.TIcon.HandleAllocated 14543
Graphics.TIcon.LoadFromClipboardFormat 14543
Graphics.TIcon.LoadFromResourceID 14543
Graphics.TIcon.LoadFromResourceName 14544
Graphics.TIcon.LoadFromStream 14544
Graphics.TIcon.ReleaseHandle 14544
Graphics.TIcon.SaveToClipboardFormat 14545
Graphics.TIcon.SaveToStream 14545
Graphics.TIcon.SetHeight 14545
Graphics.TIcon.SetSize 14546
Graphics.TIcon.SetTransparent 14546
Graphics.TIcon.SetWidth 14546
Graphics.TIconImage 14547
Graphics.TIconImage.Destroy 14548
Graphics.TIconImage.FreeHandle 14548
Graphics.TIconRec 14636
Graphics.TIconRec record 14636
Graphics.TMetafile 14549
Graphics.TMetafile.Assign 14554
Graphics.TMetafile.Clear 14555
Graphics.TMetafile.Create 14551
Graphics.TMetafile.CreatedBy 14561
Graphics.TMetafile.Description 14562
Graphics.TMetafile.Destroy 14553
Graphics.TMetafile.Draw 14555
Graphics.TMetafile.Enhanced 14562
Graphics.TMetafile.GetEmpty 14555
Graphics.TMetafile.GetHeight 14555
Graphics.TMetafile.GetPalette 14556
Graphics.TMetafile.GetWidth 14556
Graphics.TMetafile.Handle 14563
Graphics.TMetafile.HandleAllocated 14556
Graphics.TMetafile.Inch 14563
Graphics.TMetafile.LoadFromClipboardFormat 14556

Graphics.TMetafile.LoadFromStream 14557
Graphics.TMetafile.MMHeight 14563
Graphics.TMetafile.MMWidth 14564
Graphics.TMetafile.ReadData 14557
Graphics.TMetafile.ReadEMFStream 14557
Graphics.TMetafile.ReadWMFStream 14558
Graphics.TMetafile.ReleaseHandle 14558
Graphics.TMetafile.SaveToClipboardFormat 14558
Graphics.TMetafile.SaveToFile 14559
Graphics.TMetafile.SaveToStream 14559
Graphics.TMetafile.SetHeight 14559
Graphics.TMetafile.SetSize 14560
Graphics.TMetafile.SetTransparent 14560
Graphics.TMetafile.SetWidth 14560
Graphics.TMetafile.TestEMF 14560
Graphics.TMetafile.WriteData 14560
Graphics.TMetafile.WriteEMFStream 14561
Graphics.TMetafile.WriteWMFStream 14561
Graphics.TMetafileCanvas 14564
Graphics.TMetafileCanvas.Create 14567
Graphics.TMetafileCanvas.CreateWithComment 14568
Graphics.TMetafileCanvas.Destroy 14569
Graphics.TMetafileImage 14569
Graphics.TMetafileImage.Destroy 14570
Graphics.TMetafileImage.FreeHandle 14571
Graphics.TPen 14571
Graphics.TPen.Assign 14574
Graphics.TPen.Color 14577
Graphics.TPen.Create 14573
Graphics.TPen.Destroy 14573
Graphics.TPen.GetColor 14575
Graphics.TPen.GetHandle 14575
Graphics.TPen.GetStyle 14575
Graphics.TPen.GetWidth 14575
Graphics.TPen.Handle 14578
Graphics.TPen.Mode 14579
Graphics.TPen.SetColor 14575
Graphics.TPen.SetHandle 14576
Graphics.TPen.SetMode 14576
Graphics.TPen.SetStyle 14576
Graphics.TPen.SetWidth 14576
Graphics.TPen.Style 14579
Graphics.TPen.Width 14580
Graphics.TPenData 14637
Graphics.TPenData record 14637
Graphics.TPenMode 14637
Graphics.TPenMode enumeration 14637
Graphics.TPenRecall 14582
Graphics.TPenRecall.Create 14583
Graphics.TPenStyle 14639
Graphics.TPenStyle enumeration 14639
Graphics.TPicture 14584
Graphics.TPicture.Assign 14587
Graphics.TPicture.AssignTo 14589
Graphics.TPicture.Bitmap 14598
Graphics.TPicture.Changed 14590
Graphics.TPicture.Create 14586
Graphics.TPicture.DefineProperties 14590
Graphics.TPicture.Destroy 14586
Graphics.TPicture.Graphic 14600
Graphics.TPicture.Height 14600
Graphics.TPicture.Icon 14601
Graphics.TPicture.LoadFromClipboardFormat 14590
Graphics.TPicture.LoadFromFile 14591
Graphics.TPicture.LoadFromStream 14591
Graphics.TPicture.Metafile 14601
Graphics.TPicture.OnChange 14602
Graphics.TPicture.OnProgress 14603
Graphics.TPicture.PictureAdapter 14601
Graphics.TPicture.Progress 14592
Graphics.TPicture.RegisterClipboardFormat 14592
Graphics.TPicture.RegisterFileFormat 14594
Graphics.TPicture.RegisterFileFormatRes 14594
Graphics.TPicture.SaveToClipboardFormat 14595
Graphics.TPicture.SaveToFile 14595
Graphics.TPicture.SaveToStream 14597
Graphics.TPicture.SupportsClipboardFormat 14597
Graphics.TPicture.UnregisterGraphicClass 14597

Graphics.TPicture.Width 14602
Graphics.TPixelFormat 14640
Graphics.TPixelFormat enumeration 14640
Graphics.TProgressEvent 14655
Graphics.TProgressEvent type 14655
Graphics.TProgressStage 14641
Graphics.TProgressStage enumeration 14641
Graphics.TransparentStretchBlt 14629
Graphics.TransparentStretchBlt function 14629
Graphics.TResData 14641
Graphics.TResData record 14641
Graphics.TResource 14642
Graphics.TResource record 14642
Graphics.TSharedImage 14603
Graphics.TSharedImage.FreeHandle 14604
Graphics.TSharedImage.RefCount 14605
Graphics.TSharedImage.Reference 14604
Graphics.TSharedImage.Release 14604
Graphics.TTextFormat 14656
Graphics.TTextFormat type 14656
Graphics.TTextFormats 14642
Graphics.TTextFormats enumeration 14642
Graphics.TTransparentMode 14643
Graphics.TTransparentMode enumeration 14643
Graphics.WebColorsCount 14745
Graphics.WebColorsCount constant 14745
GraphUtil 14747
GraphUtil namespace 14747
- Constants 14760
- Functions 14749
- Structs, Records, Enums 14756
- Types 14759

GraphUtil.ColorAdjustLuma 14749
GraphUtil.ColorAdjustLuma function 14749
GraphUtil.ColorHLSToRGB 14749
GraphUtil.ColorHLSToRGB function 14749
GraphUtil.ColorRGBToHLS 14750
GraphUtil.ColorRGBToHLS function 14750
GraphUtil.ColorToWebColorName 14750
GraphUtil.ColorToWebColorName function 14750
GraphUtil.ColorToWebColorStr 14750
GraphUtil.ColorToWebColorStr function 14750
GraphUtil.DrawArrow 14751
GraphUtil.DrawArrow function 14751
GraphUtil.DrawCheck 14751
GraphUtil.DrawCheck function 14751
GraphUtil.DrawChevron 14751
GraphUtil.DrawChevron function 14751
GraphUtil.GetHighLightColor 14752
GraphUtil.GetHighLightColor function 14752
GraphUtil.GetShadowColor 14752
GraphUtil.GetShadowColor function 14752
GraphUtil.GradientFillCanvas 14752
GraphUtil.GradientFillCanvas function 14752
GraphUtil.RGBToWebColorName 14753
GraphUtil.RGBToWebColorName function 14753
GraphUtil.RGBToWebColorStr 14753
GraphUtil.RGBToWebColorStr function 14753
GraphUtil.ScaleImage 14753
GraphUtil.ScaleImage function 14753
GraphUtil.SortColorArray 14754
GraphUtil.SortColorArray function 14754
GraphUtil.TArrowType 14756
GraphUtil.TArrowType enumeration 14756
GraphUtil.TColorArray 14759
GraphUtil.TColorArray type 14759
GraphUtil.TColorArraySortType 14756
GraphUtil.TColorArraySortType enumeration 14756
GraphUtil.TGradientDirection 14757
GraphUtil.TGradientDirection enumeration 14757
GraphUtil.TScrollDirection 14757
GraphUtil.TScrollDirection enumeration 14757
GraphUtil.WebColorNameToColor 14754
GraphUtil.WebColorNameToColor function 14754
GraphUtil.WebColorStrToColor 14754
GraphUtil.WebColorStrToColor function 14754
GraphUtil.WebColorToRGB 14755
GraphUtil.WebColorToRGB function 14755

GraphUtil.WebNamedColors 14760
GraphUtil.WebNamedColors constant 14760
GraphUtil.WebNamedColorsCount 14762
GraphUtil.WebNamedColorsCount constant 14762
grBOF enumeration member 7616
grEOF enumeration member 7616
grError enumeration member 7616
Grids 14765
Grids namespace 14765
    Classes 14767
    Constants 15009
    Structs, Records, Enums 14997
    Types 15003
Grids.EInvalidGridOperation 14767
Grids.MaxCustomExtents 15009
Grids.MaxCustomExtents constant 15009
Grids.MaxShortInt 15009
Grids.MaxShortInt constant 15009
Grids.TCustomDrawGrid 14768
Grids.TCustomDrawGrid.Canvas 14785
Grids.TCustomDrawGrid.CellRect 14781
Grids.TCustomDrawGrid.Col 14787
Grids.TCustomDrawGrid.ColumnMoved 14781
Grids.TCustomDrawGrid.ColWidths 14788
Grids.TCustomDrawGrid.DrawCell 14782
Grids.TCustomDrawGrid.EditorMode 14789
Grids.TCustomDrawGrid.GetEditMask 14782
Grids.TCustomDrawGrid.GetEditText 14783
Grids.TCustomDrawGrid.GridHeight 14789
Grids.TCustomDrawGrid.GridWidth 14789
Grids.TCustomDrawGrid.LeftCol 14790
Grids.TCustomDrawGrid.MouseToCell 14783
Grids.TCustomDrawGrid.OnColumnMoved 14794
Grids.TCustomDrawGrid.OnDrawCell 14795
Grids.TCustomDrawGrid.OnGetEditMask 14796
Grids.TCustomDrawGrid.OnGetEditText 14797
Grids.TCustomDrawGrid.OnRowMoved 14798
Grids.TCustomDrawGrid.OnSelectCell 14798
Grids.TCustomDrawGrid.OnSetEditText 14800
Grids.TCustomDrawGrid.OnTopLeftChanged 14800
Grids.TCustomDrawGrid.Row 14790
Grids.TCustomDrawGrid.RowHeights 14791
Grids.TCustomDrawGrid.RowMoved 14783
Grids.TCustomDrawGrid.SelectCell 14784
Grids.TCustomDrawGrid.Selection 14791
Grids.TCustomDrawGrid.SetEditText 14784
Grids.TCustomDrawGrid.TabStops 14793
Grids.TCustomDrawGrid.TopLeftChanged 14784
Grids.TCustomDrawGrid.TopRow 14793
Grids.TCustomGrid 14800
Grids.TCustomGrid.AdjustSize 14816
Grids.TCustomGrid.BeginColumnDrag 14817
Grids.TCustomGrid.BeginRowDrag 14817
Grids.TCustomGrid.BorderStyle 14843
Grids.TCustomGrid.BoxRect 14818
Grids.TCustomGrid.CalcDrawInfo 14818
Grids.TCustomGrid.CalcFixedInfo 14818
Grids.TCustomGrid.CalcSizingState 14819
Grids.TCustomGrid.CanEditAcceptKey 14819
Grids.TCustomGrid.CanEditModify 14819
Grids.TCustomGrid.CanEditShow 14820
Grids.TCustomGrid.CanGridAcceptKey 14820
Grids.TCustomGrid.CellRect 14821
Grids.TCustomGrid.ChangeGridOrientation 14821
Grids.TCustomGrid.CheckColumnDrag 14821
Grids.TCustomGrid.CheckRowDrag 14822
Grids.TCustomGrid.Col 14843
Grids.TCustomGrid.ColCount 14844
Grids.TCustomGrid.Color 14844
Grids.TCustomGrid.ColumnMoved 14822
Grids.TCustomGrid.ColWidths 14845
Grids.TCustomGrid.ColWidthsChanged 14822
Grids.TCustomGrid.Create 14813
Grids.TCustomGrid.CreateEditor 14823
Grids.TCustomGrid.CreateParams 14823
Grids.TCustomGrid.DefaultColWidth 14846
Grids.TCustomGrid.DefaultDrawing 14847
Grids.TCustomGrid.DefaultRowHeight 14847

Grids.TCustomGrid.DefineProperties 14824
Grids.TCustomGrid.DeleteColumn 14824
Grids.TCustomGrid.DeleteRow 14825
Grids.TCustomGrid.DesignOptionsBoost 14814
Grids.TCustomGrid.Destroy 14813
Grids.TCustomGrid.DoExit 14825
Grids.TCustomGrid.DoMouseWheelDown 14825
Grids.TCustomGrid.DoMouseWheelUp 14826
Grids.TCustomGrid.DrawCell 14826
Grids.TCustomGrid.EditorMode 14848
Grids.TCustomGrid.EndColumnDrag 14827
Grids.TCustomGrid.EndRowDrag 14827
Grids.TCustomGrid.FGridState 14814
Grids.TCustomGrid.FixedColor 14848
Grids.TCustomGrid.FixedCols 14848
Grids.TCustomGrid.FixedRows 14849
Grids.TCustomGrid.FocusCell 14827
Grids.TCustomGrid.FSaveCellExtents 14814
Grids.TCustomGrid.GetEditLimit 14828
Grids.TCustomGrid.GetEditMask 14828
Grids.TCustomGrid.GetEditStyle 14828
Grids.TCustomGrid.GetEditText 14829
Grids.TCustomGrid.GetGridHeight 14829
Grids.TCustomGrid.GetGridWidth 14830
Grids.TCustomGrid.GridHeight 14850
Grids.TCustomGrid.GridLineWidth 14850
Grids.TCustomGrid.GridWidth 14851
Grids.TCustomGrid.HideEdit 14830
Grids.TCustomGrid.HideEditor 14830
Grids.TCustomGrid.HitTest 14851
Grids.TCustomGrid.InplaceEditor 14852
Grids.TCustomGrid.InvalidateCell 14831
Grids.TCustomGrid.InvalidateCol 14831
Grids.TCustomGrid.InvalidateEditor 14832
Grids.TCustomGrid.InvalidateGrid 14832
Grids.TCustomGrid.InvalidateRow 14832
Grids.TCustomGrid.KeyDown 14833
Grids.TCustomGrid.KeyPress 14833
Grids.TCustomGrid.LeftCol 14852
Grids.TCustomGrid.MouseCoord 14834
Grids.TCustomGrid.MouseDown 14834
Grids.TCustomGrid.MouseMove 14834
Grids.TCustomGrid.MouseUp 14835
Grids.TCustomGrid.MoveColRow 14835
Grids.TCustomGrid.MoveColumn 14836
Grids.TCustomGrid.MoveRow 14836
Grids.TCustomGrid.Options 14853
Grids.TCustomGrid.Paint 14836
Grids.TCustomGrid.ParentColor 14853
Grids.TCustomGrid.Row 14854
Grids.TCustomGrid.RowCount 14854
Grids.TCustomGrid.RowHeights 14855
Grids.TCustomGrid.RowHeightsChanged 14837
Grids.TCustomGrid.RowMoved 14837
Grids.TCustomGrid.ScrollBars 14856
Grids.TCustomGrid.ScrollData 14838
Grids.TCustomGrid.SelectCell 14838
Grids.TCustomGrid.Selection 14856
Grids.TCustomGrid.SetEditText 14839
Grids.TCustomGrid.ShowEditor 14839
Grids.TCustomGrid.ShowEditorChar 14839
Grids.TCustomGrid.SizeChanged 14840
Grids.TCustomGrid.Sizing 14840
Grids.TCustomGrid.TabStop 14857
Grids.TCustomGrid.TabStops 14858
Grids.TCustomGrid.TimedScroll 14840
Grids.TCustomGrid.TopLeftChanged 14841
Grids.TCustomGrid.TopRow 14858
Grids.TCustomGrid.UpdateDesigner 14841
Grids.TCustomGrid.VirtualView 14814
Grids.TCustomGrid.VisibleColCount 14859
Grids.TCustomGrid.VisibleRowCount 14859
Grids.TDrawCellEvent 15003
Grids.TDrawCellEvent type 15003
Grids.TDrawGrid 14860
Grids.TDrawGrid.Align 14874
Grids.TDrawGrid.Anchors 14878
Grids.TDrawGrid.BevelEdges 14879

Grids.TDrawGrid.BevelInner 14879
Grids.TDrawGrid.BevelKind 14879
Grids.TDrawGrid.BevelOuter 14880
Grids.TDrawGrid.BevelWidth 14880
Grids.TDrawGrid.BiDiMode 14881
Grids.TDrawGrid.BorderStyle 14881
Grids.TDrawGrid.ColCount 14881
Grids.TDrawGrid.Color 14882
Grids.TDrawGrid.Constraints 14883
Grids.TDrawGrid.Ctl3D 14884
Grids.TDrawGrid.DefaultColWidth 14884
Grids.TDrawGrid.DefaultDrawing 14885
Grids.TDrawGrid.DefaultRowHeight 14885
Grids.TDrawGrid.DragCursor 14886
Grids.TDrawGrid.DragKind 14886
Grids.TDrawGrid.DragMode 14887
Grids.TDrawGrid.Enabled 14888
Grids.TDrawGrid.FixedColor 14889
Grids.TDrawGrid.FixedCols 14889
Grids.TDrawGrid.FixedRows 14889
Grids.TDrawGrid.Font 14890
Grids.TDrawGrid.GridLineWidth 14891
Grids.TDrawGrid.OnClick 14906
Grids.TDrawGrid.OnColumnMoved 14908
Grids.TDrawGrid.OnContextPopup 14908
Grids.TDrawGrid.OnDblClick 14909
Grids.TDrawGrid.OnDragDrop 14909
Grids.TDrawGrid.OnDragOver 14911
Grids.TDrawGrid.OnDrawCell 14914
Grids.TDrawGrid.OnEndDock 14914
Grids.TDrawGrid.OnEndDrag 14915
Grids.TDrawGrid.OnEnter 14916
Grids.TDrawGrid.OnExit 14918
Grids.TDrawGrid.OnGetEditMask 14920
Grids.TDrawGrid.OnGetEditText 14920
Grids.TDrawGrid.OnKeyDown 14920
Grids.TDrawGrid.OnKeyPress 14923
Grids.TDrawGrid.OnKeyUp 14924
Grids.TDrawGrid.OnMouseActivate 14926
Grids.TDrawGrid.OnMouseDown 14927
Grids.TDrawGrid.OnMouseEnter 14929
Grids.TDrawGrid.OnMouseLeave 14929
Grids.TDrawGrid.OnMouseMove 14930
Grids.TDrawGrid.OnMouseUp 14932
Grids.TDrawGrid.OnMouseWheelDown 14934
Grids.TDrawGrid.OnMouseWheelUp 14935
Grids.TDrawGrid.OnRowMoved 14935
Grids.TDrawGrid.OnSelectCell 14935
Grids.TDrawGrid.OnSetEditText 14936
Grids.TDrawGrid.OnStartDock 14936
Grids.TDrawGrid.OnStartDrag 14937
Grids.TDrawGrid.OnTopLeftChanged 14937
Grids.TDrawGrid.Options 14892
Grids.TDrawGrid.ParentBiDiMode 14893
Grids.TDrawGrid.ParentColor 14893
Grids.TDrawGrid.ParentCtl3D 14893
Grids.TDrawGrid.ParentFont 14894
Grids.TDrawGrid.ParentShowHint 14894
Grids.TDrawGrid.PopupMenu 14895
Grids.TDrawGrid.RowCount 14899
Grids.TDrawGrid.ScrollBars 14900
Grids.TDrawGrid.ShowHint 14901
Grids.TDrawGrid.TabOrder 14902
Grids.TDrawGrid.Visible 14903
Grids.TDrawGrid.VisibleColCount 14904
Grids.TDrawGrid.VisibleRowCount 14905
Grids.TEditStyle 14997
Grids.TEditStyle enumeration 14997
Grids.TGetEditEvent 15004
Grids.TGetEditEvent type 15004
Grids.TGetExtentsFunc 15004
Grids.TGetExtentsFunc type 15004
Grids.TGridAxisDrawInfo 14998
Grids.TGridAxisDrawInfo record 14998
Grids.TGridCoord 14998
Grids.TGridCoord record 14998
Grids.TGridDrawInfo 14999
Grids.TGridDrawInfo record 14999

Grids.TGridDrawState 15004
Grids.TGridDrawState type 15004
Grids.TGridMovement 15005
Grids.TGridMovement type 15005
Grids.TGridOption 14999
Grids.TGridOption enumeration 14999
Grids.TGridOptions 15005
Grids.TGridOptions type 15005
Grids.TGridRect 15001
Grids.TGridRect record 15001
Grids.TGridScrollDirection 15006
Grids.TGridScrollDirection type 15006
Grids.TGridState 15001
Grids.TGridState enumeration 15001
Grids.TInplaceEdit 14938
Grids.TInplaceEdit.BoundsChanged 14951
Grids.TInplaceEdit.Create 14950
Grids.TInplaceEdit.CreateParams 14952
Grids.TInplaceEdit.DblClick 14952
Grids.TInplaceEdit.Deselect 14952
Grids.TInplaceEdit.DoMouseWheel 14953
Grids.TInplaceEdit.EditCanModify 14953
Grids.TInplaceEdit.Grid 14958
Grids.TInplaceEdit.Hide 14953
Grids.TInplaceEdit.Invalidate 14954
Grids.TInplaceEdit.KeyDown 14954
Grids.TInplaceEdit.KeyPress 14955
Grids.TInplaceEdit.KeyUp 14955
Grids.TInplaceEdit.Move 14956
Grids.TInplaceEdit.PosEqual 14956
Grids.TInplaceEdit.SetFocus 14956
Grids.TInplaceEdit.UpdateContents 14957
Grids.TInplaceEdit.UpdateLoc 14957
Grids.TInplaceEdit.Visible 14958
Grids.TInplaceEdit.WndProc 14958
Grids.TInplaceEditList 14959
Grids.TInplaceEditList.ActiveList 14966
Grids.TInplaceEditList.BoundsChanged 14961
Grids.TInplaceEditList.ButtonRect 14961
Grids.TInplaceEditList.ButtonWidth 14966
Grids.TInplaceEditList.CloseUp 14961
Grids.TInplaceEditList.Create 14960
Grids.TInplaceEditList.DblClick 14961
Grids.TInplaceEditList.DoDropDownKeys 14962
Grids.TInplaceEditList.DoEditButtonClick 14962
Grids.TInplaceEditList.DoGetPickListItems 14962
Grids.TInplaceEditList.DropDown 14962
Grids.TInplaceEditList.DropDownRows 14966
Grids.TInplaceEditList.EditStyle 14966
Grids.TInplaceEditList.KeyDown 14963
Grids.TInplaceEditList.ListMouseUp 14963
Grids.TInplaceEditList.ListVisible 14966
Grids.TInplaceEditList.MouseDown 14963
Grids.TInplaceEditList.MouseMove 14963
Grids.TInplaceEditList.MouseUp 14963
Grids.TInplaceEditList.OnEditButtonClick 14967
Grids.TInplaceEditList.OnGetPickListitems 14968
Grids.TInplaceEditList.OverButton 14964
Grids.TInplaceEditList.PaintWindow 14964
Grids.TInplaceEditList.PickList 14967
Grids.TInplaceEditList.PickListLoaded 14967
Grids.TInplaceEditList.Pressed 14967
Grids.TInplaceEditList.RestoreContents 14964
Grids.TInplaceEditList.StopTracking 14964
Grids.TInplaceEditList.TrackButton 14965
Grids.TInplaceEditList.UpdateContents 14965
Grids.TInplaceEditList.WndProc 14965
Grids.TMovedEvent 15007
Grids.TMovedEvent type 15007
Grids.TOnGetPickListItems 15007
Grids.TOnGetPickListItems type 15007
Grids.TSelectCellEvent 15007
Grids.TSelectCellEvent type 15007
Grids.TSetEditEvent 15008
Grids.TSetEditEvent type 15008
Grids.TStringGrid 14968
Grids.TStringGrid.Cells 14986
Grids.TStringGrid.Cols 14987

Grids.TStringGrid.ColumnMoved 14984
Grids.TStringGrid.Create 14983
Grids.TStringGrid.Destroy 14983
Grids.TStringGrid.DrawCell 14984
Grids.TStringGrid.GetEditText 14985
Grids.TStringGrid.Objects 14987
Grids.TStringGrid.RowMoved 14985
Grids.TStringGrid.Rows 14987
Grids.TStringGrid.SetEditText 14985
Grids.TStringGridStrings 14988
Grids.TStringGridStrings.Add 14992
Grids.TStringGridStrings.Assign 14992
Grids.TStringGridStrings.Clear 14993
Grids.TStringGridStrings.Create 14991
Grids.TStringGridStrings.Delete 14993
Grids.TStringGridStrings.Get 14994
Grids.TStringGridStrings.GetCount 14994
Grids.TStringGridStrings.GetObject 14994
Grids.TStringGridStrings.Insert 14995
Grids.TStringGridStrings.Put 14995
Grids.TStringGridStrings.PutObject 14996
Grids.TStringGridStrings.SetUpdateState 14996
grMetaData enumeration member 18617
grOK enumeration member 7616
grReset enumeration member 18617
grSTATEFIRST enumeration member 11711
grSTATEFIRSTLAST enumeration member 11711
grSTATELAST enumeration member 11711
grSTATEMIDDLE enumeration member 11711
grXML enumeration member 18617
grXMLUTF8 enumeration member 18617
gsColMoving enumeration member 15001
gsColSizing enumeration member 15001
gsNormal enumeration member 15001
gsRowMoving enumeration member 15001
gsRowSizing enumeration member 15001
gsSelecting enumeration member 15001
gtAlignBottom enumeration member 11234
gtAlignLeft enumeration member 11234
gtAlignRight enumeration member 11234
gtAlignTop enumeration member 11234
gtBaseline enumeration member 11234
gtMarginBottom enumeration member 11234
gtMarginLeft enumeration member 11234
gtMarginRight enumeration member 11234
gtMarginTop enumeration member 11234
gtPaddingBottom enumeration member 11234
gtPaddingLeft enumeration member 11234
gtPaddingRight enumeration member 11234
gtPaddingTop enumeration member 11234

H

haBaseline enumeration member 15336
haBottom enumeration member 15336
haCenter enumeration member 15336
haDefault enumeration member 15336
haLeft enumeration member 15336
haMiddle enumeration member 15336
haRight enumeration member 15336
haTop enumeration member 15336
haVDefault enumeration member 15336
hcAlt enumeration member 5451
hcCtrl enumeration member 5451
hcCtrlAlt enumeration member 5451
hcNone enumeration member 5451
hcShift enumeration member 5451
hcShiftAlt enumeration member 5451
hcShiftCtrl enumeration member 5451
hcShiftCtrlAlt enumeration member 5451
HelpIntfs 15011
HelpIntfs namespace 15011
    Classes 15012
    Functions 15033
    Interfaces 15013
HelpIntfs.EHelpSystemException 15012
HelpIntfs.GetHelpSystem 15033
HelpIntfs.GetHelpSystem function 15033
HelpIntfs.ICustomHelpViewer 15013

HelpIntfs.ICustomHelpViewer.CanShowTableOfContents 15014
HelpIntfs.ICustomHelpViewer.GetHelpStrings 15015
HelpIntfs.ICustomHelpViewer.GetViewerName 15015
HelpIntfs.ICustomHelpViewer.NotifyID 15016
HelpIntfs.ICustomHelpViewer.ShowHelp 15016
HelpIntfs.ICustomHelpViewer.ShowTableOfContents 15016
HelpIntfs.ICustomHelpViewer.ShutDown 15017
HelpIntfs.ICustomHelpViewer.SoftShutDown 15017
HelpIntfs.ICustomHelpViewer.UnderstandsKeyword 15017
HelpIntfs.IExtendedHelpViewer 15017
HelpIntfs.IExtendedHelpViewer.DisplayHelpByContext 15019
HelpIntfs.IExtendedHelpViewer.DisplayTopic 15019
HelpIntfs.IExtendedHelpViewer.UnderstandsContext 15019
HelpIntfs.IExtendedHelpViewer.UnderstandsTopic 15020
HelpIntfs.IHelpManager 15020
HelpIntfs.IHelpManager.GetHandle 15021
HelpIntfs.IHelpManager.GetHelpFile 15021
HelpIntfs.IHelpManager.Release 15021
HelpIntfs.IHelpSelector 15022
HelpIntfs.IHelpSelector.SelectKeyword 15023
HelpIntfs.IHelpSelector.TableOfContents 15024
HelpIntfs.IHelpSelector2 15024
HelpIntfs.IHelpSelector2.SelectContext 15025
HelpIntfs.IHelpSystem 15025
HelpIntfs.IHelpSystem.AssignHelpSelector 15026
HelpIntfs.IHelpSystem.Hook 15026
HelpIntfs.IHelpSystem.ShowContextHelp 15027
HelpIntfs.IHelpSystem.ShowHelp 15027
HelpIntfs.IHelpSystem.ShowTableOfContents 15027
HelpIntfs.IHelpSystem.ShowTopicHelp 15028
HelpIntfs.IHelpSystem2 15028
HelpIntfs.IHelpSystem2.UnderstandsKeyword 15029
HelpIntfs.IHelpSystemFlags 15029
HelpIntfs.IHelpSystemFlags.GetUseDefaultTopic 15030
HelpIntfs.IHelpSystemFlags.SetUseDefaultTopic 15030
HelpIntfs.ISpecialWinHelpViewer 15031
HelpIntfs.ISpecialWinHelpViewer.CallWinHelp 15032
HelpIntfs.RegisterViewer 15033
HelpIntfs.RegisterViewer function 15033
hkAlt enumeration member 5451
hkCtrl enumeration member 5451
hkExt enumeration member 5451
hkShift enumeration member 5451
hmtAll enumeration member 15624
hmtRequest enumeration member 15624
hmtResponse enumeration member 15624
hoHideAlways enumeration member 25317
hoHideOnFieldNotVisible enumeration member 25317
hoHideOnNoDisplayAccess enumeration member 25317
hoHideOnNoInputAccess enumeration member 25317
hoMode enumeration member 24593
hoOrigValues enumeration member 24593
hoRecordKeys enumeration member 24593
hsButtons enumeration member 5449
hsFlat enumeration member 5449
hsOwnerDraw enumeration member 5449
hsText enumeration member 5449
htAbove enumeration member 5450
htBelow enumeration member 5450
htContext enumeration member 3272
htHandPoint enumeration member 5456
htKeyword enumeration member 3272
htmlaAnchor enumeration member 19789
htmlaButton enumeration member 19789
htmlaEventImages enumeration member 19789
htmlaImage enumeration member 19789
htmldImage enumeration member 19790
htmldList enumeration member 19790
htmldNone enumeration member 19790
htmldText enumeration member 19790
htmliCheckBox enumeration member 19791
htmliFile enumeration member 19791
htmliNone enumeration member 19791
htmliPasswordInput enumeration member 19791
htmliRadio enumeration member 19791
htmliSelect enumeration member 19791
htmliSelectMultiple enumeration member 19791
htmliTextArea enumeration member 19791

htmliTextInput enumeration member 19791
htmlvmDisplay enumeration member 19790
htmlvmInput enumeration member 19790
htmlvmNone enumeration member 19790
htmlvmToggleOnAccess enumeration member 19790
htNowhere enumeration member 5450
htOnButton enumeration member 5450
htOnIcon enumeration member 5450
htOnIndent enumeration member 5450
htOnItem enumeration member 5450
htOnLabel enumeration member 5450
htOnRight enumeration member 5450
htOnStateIcon enumeration member 5450
htToLeft enumeration member 5450
htToRight enumeration member 5450
HTTPApp 15035
HTTPApp namespace 15035
Classes 15038
Constants 15250
Functions 15239
Interfaces 15219
Structs, Records, Enums 15246
Types 15248
HTTPApp.DayOfWeekStr 15239
HTTPApp.DayOfWeekStr function 15239
HTTPApp.DosPathToUnixPath 15240
HTTPApp.DosPathToUnixPath function 15240
HTTPApp.EWebBrokerException 15038
HTTPApp.ExtractHeaderFields 15240
HTTPApp.ExtractHeaderFields function 15240
HTTPApp.ExtractHTTPFields 15241
HTTPApp.ExtractHTTPFields function 15241
HTTPApp.HTMLDecode 15241
HTTPApp.HTMLDecode function 15241
HTTPApp.HTMLEncode 15242
HTTPApp.HTMLEncode function 15242
HTTPApp.HTTPDecode 15242
HTTPApp.HTTPDecode function 15242
HTTPApp.HTTPEncode 15243
HTTPApp.HTTPEncode function 15243
HTTPApp.IGetAppDispatcher 15219
HTTPApp.IGetAppDispatcher.GetAppDispatcher 15220
HTTPApp.IGetWebAppServices 15220
HTTPApp.IGetWebAppServices.GetWebAppServices 15221
HTTPApp.IGetWebRequestHandler 15221
HTTPApp.IGetWebRequestHandler.GetWebRequestHandler 15222
HTTPApp.IMultiModuleSupport 15222
HTTPApp.IMultiModuleSupport.FinishContext 15223
HTTPApp.IMultiModuleSupport.InitContext 15223
HTTPApp.IMultiModuleSupport.InitModule 15223
HTTPApp.IProduceContent 15223
HTTPApp.IProduceContent.ProduceContent 15224
HTTPApp.IProduceContentFrom 15225
HTTPApp.IProduceContentFrom.ProduceContentFromStream 15225
HTTPApp.IProduceContentFrom.ProduceContentFromString 15226
HTTPApp.ISetAppDispatcher 15226
HTTPApp.ISetAppDispatcher.SetAppDispatcher 15227
HTTPApp.IWebAppServices 15227
HTTPApp.IWebAppServices.ExceptionHandler 15230
HTTPApp.IWebAppServices.FinishContext 15228
HTTPApp.IWebAppServices.GetExceptionHandler 15229
HTTPApp.IWebAppServices.HandleRequest 15229
HTTPApp.IWebAppServices.InitContext 15229
HTTPApp.IWebDispatch 15230
HTTPApp.IWebDispatch.DispatchEnabled 15231
HTTPApp.IWebDispatch.DispatchMask 15232
HTTPApp.IWebDispatch.DispatchMethodType 15232
HTTPApp.IWebDispatch.DispatchRequest 15232
HTTPApp.IWebDispatch.Enabled 15232
HTTPApp.IWebDispatch.Mask 15233
HTTPApp.IWebDispatch.MethodType 15233
HTTPApp.IWebDispatchActions 15233
HTTPApp.IWebDispatcherAccess 15234
HTTPApp.IWebDispatcherAccess.Request 15235
HTTPApp.IWebDispatcherAccess.Response 15235
HTTPApp.IWebExceptionHandler 15236

HTTPApp.IWebExceptionHandler.HandleException 15236
HTTPApp.IWebRequestHandler 15237
HTTPApp.IWebRequestHandler.HandleRequest 15238
HTTPApp.MAX_DATETIMES 15250
HTTPApp.MAX_DATETIMES constant 15250
HTTPApp.MAX_INTEGERS 15250
HTTPApp.MAX_INTEGERS constant 15250
HTTPApp.MAX_STRINGS 15251
HTTPApp.MAX_STRINGS constant 15251
HTTPApp.MonthStr 15243
HTTPApp.MonthStr function 15243
HTTPApp.ParseDate 15244
HTTPApp.ParseDate function 15244
HTTPApp.RegisterContentParser 15244
HTTPApp.RegisterContentParser function 15244
HTTPApp.sDateFormat 15251
HTTPApp.sDateFormat constant 15251
HTTPApp.StatusString 15244
HTTPApp.StatusString function 15244
HTTPApp.TAbstractContentParser 15039
HTTPApp.TAbstractContentParser.CanParse 15041
HTTPApp.TAbstractContentParser.Create 15041
HTTPApp.TAbstractContentParser.GetContentFields 15042
HTTPApp.TAbstractContentParser.GetFiles 15042
HTTPApp.TAbstractContentParser.WebRequest 15043
HTTPApp.TAbstractWebModuleFactory 15043
HTTPApp.TAbstractWebModuleFactory.CacheMode 15048
HTTPApp.TAbstractWebModuleFactory.ComponentClass 15049
HTTPApp.TAbstractWebModuleFactory.CreateMode 15049
HTTPApp.TAbstractWebModuleFactory.GetCacheMode 15045
HTTPApp.TAbstractWebModuleFactory.GetComponentClass 15045
HTTPApp.TAbstractWebModuleFactory.GetCreateMode 15046
HTTPApp.TAbstractWebModuleFactory.GetIsAppModule 15046
HTTPApp.TAbstractWebModuleFactory.GetModule 15046
HTTPApp.TAbstractWebModuleFactory.GetModuleName 15047
HTTPApp.TAbstractWebModuleFactory.GetWebPageInfo 15047
HTTPApp.TAbstractWebModuleFactory.IsAppModule 15050
HTTPApp.TAbstractWebModuleFactory.ModuleName 15050
HTTPApp.TAbstractWebModuleFactory.PreventDestruction 15048
HTTPApp.TAbstractWebModuleFactory.WebPageInfo 15050
HTTPApp.TAbstractWebModuleList 15051
HTTPApp.TAbstractWebModuleList.AddModuleClass 15053
HTTPApp.TAbstractWebModuleList.AddModuleName 15054
HTTPApp.TAbstractWebModuleList.Factory 15057
HTTPApp.TAbstractWebModuleList.FactoryCount 15058
HTTPApp.TAbstractWebModuleList.FindModuleClass 15054
HTTPApp.TAbstractWebModuleList.FindModuleName 15055
HTTPApp.TAbstractWebModuleList.GetFactory 15055
HTTPApp.TAbstractWebModuleList.GetFactoryCount 15056
HTTPApp.TAbstractWebModuleList.GetItem 15056
HTTPApp.TAbstractWebModuleList.GetItemCount 15056
HTTPApp.TAbstractWebModuleList.GetOnModuleAdded 15056
HTTPApp.TAbstractWebModuleList.ItemCount 15058
HTTPApp.TAbstractWebModuleList.Items 15058
HTTPApp.TAbstractWebModuleList.OnModuleAdded 15059
HTTPApp.TAbstractWebModuleList.SetOnModuleAdded 15057
HTTPApp.TAbstractWebPageInfo 15060
HTTPApp.TAbstractWebPageInfo.Factory 15066
HTTPApp.TAbstractWebPageInfo.GetFactory 15062
HTTPApp.TAbstractWebPageInfo.GetIsLoginRequired 15062
HTTPApp.TAbstractWebPageInfo.GetIsPublished 15062
HTTPApp.TAbstractWebPageInfo.GetPageDescription 15063
HTTPApp.TAbstractWebPageInfo.GetPageFile 15063
HTTPApp.TAbstractWebPageInfo.GetPageHREF 15064
HTTPApp.TAbstractWebPageInfo.GetPageName 15064
HTTPApp.TAbstractWebPageInfo.GetPageTitle 15064
HTTPApp.TAbstractWebPageInfo.GetViewAccess 15065
HTTPApp.TAbstractWebPageInfo.IsLoginRequired 15066
HTTPApp.TAbstractWebPageInfo.IsPublished 15067
HTTPApp.TAbstractWebPageInfo.PageDescription 15067
HTTPApp.TAbstractWebPageInfo.PageFile 15068
HTTPApp.TAbstractWebPageInfo.PageHREF 15068
HTTPApp.TAbstractWebPageInfo.PageName 15068
HTTPApp.TAbstractWebPageInfo.PageTitle 15069
HTTPApp.TAbstractWebPageInfo.SetFactory 15065
HTTPApp.TAbstractWebPageInfo.ViewAccess 15069
HTTPApp.TAbstractWebPageModuleFactory 15070

HTTPApp.TAbstractWebPageModuleFactory.Create 15071
HTTPApp.TAbstractWebPageModuleFactory.Destroy 15071
HTTPApp.TAbstractWebPageModuleFactory.GetWebPageInfo 15072
HTTPApp.TAbstractWebRequestFile 15073
HTTPApp.TAbstractWebRequestFile.ContentType 15076
HTTPApp.TAbstractWebRequestFile.FieldName 15077
HTTPApp.TAbstractWebRequestFile.FileName 15077
HTTPApp.TAbstractWebRequestFile.GetContentType 15075
HTTPApp.TAbstractWebRequestFile.GetFieldName 15075
HTTPApp.TAbstractWebRequestFile.GetFileName 15075
HTTPApp.TAbstractWebRequestFile.GetStream 15076
HTTPApp.TAbstractWebRequestFile.Stream 15077
HTTPApp.TAbstractWebRequestFiles 15078
HTTPApp.TAbstractWebRequestFiles.Count 15081
HTTPApp.TAbstractWebRequestFiles.GetCount 15080
HTTPApp.TAbstractWebRequestFiles.GetItem 15080
HTTPApp.TAbstractWebRequestFiles.Items 15081
HTTPApp.TAbstractWebSession 15082
HTTPApp.TAbstractWebSession.GetSessionID 15084
HTTPApp.TAbstractWebSession.GetTimeoutMinutes 15084
HTTPApp.TAbstractWebSession.GetValue 15084
HTTPApp.TAbstractWebSession.SessionID 15087
HTTPApp.TAbstractWebSession.SetTimeoutMinutes 15085
HTTPApp.TAbstractWebSession.SetValue 15085
HTTPApp.TAbstractWebSession.Terminate 15086
HTTPApp.TAbstractWebSession.TimeoutMinutes 15087
HTTPApp.TAbstractWebSession.UpdateResponse 15086
HTTPApp.TAbstractWebSession.Values 15088
HTTPApp.TContentParser 15088
HTTPApp.TContentParser.CanParse 15090
HTTPApp.TContentParser.Destroy 15090
HTTPApp.TContentParser.GetContentFields 15091
HTTPApp.TContentParser.GetFiles 15091
HTTPApp.TContentParserClass 15248
HTTPApp.TContentParserClass type 15248
HTTPApp.TCookie 15092
HTTPApp.TCookie.AssignTo 15094
HTTPApp.TCookie.Create 15093
HTTPApp.TCookie.Domain 15095
HTTPApp.TCookie.Expires 15095
HTTPApp.TCookie.GetHeaderValue 15095
HTTPApp.TCookie.HeaderValue 15096
HTTPApp.TCookie.Name 15096
HTTPApp.TCookie.Path 15097
HTTPApp.TCookie.Secure 15097
HTTPApp.TCookie.Value 15097
HTTPApp.TCookieCollection 15098
HTTPApp.TCookieCollection.Add 15100
HTTPApp.TCookieCollection.Create 15100
HTTPApp.TCookieCollection.GetCookie 15101
HTTPApp.TCookieCollection.Items 15101
HTTPApp.TCookieCollection.SetCookie 15101
HTTPApp.TCookieCollection.WebResponse 15102
HTTPApp.TCustomContentProducer 15102
HTTPApp.TCustomContentProducer.Content 15106
HTTPApp.TCustomContentProducer.ContentFromStream 15107
HTTPApp.TCustomContentProducer.ContentFromString 15107
HTTPApp.TCustomContentProducer.Dispatcher 15109
HTTPApp.TCustomContentProducer.DispatcherComponent 15110
HTTPApp.TCustomContentProducer.GetAppDispatcher 15108
HTTPApp.TCustomContentProducer.Notification 15108
HTTPApp.TCustomContentProducer.ProduceContent 15108
HTTPApp.TCustomContentProducer.ProduceContentFromStream 15109
HTTPApp.TCustomContentProducer.ProduceContentFromString 15109
HTTPApp.TCustomContentProducer.SetAppDispatcher 15109
HTTPApp.TCustomWebDispatcher 15110
HTTPApp.TCustomWebDispatcher.Access_Request 15116
HTTPApp.TCustomWebDispatcher.Access_Response 15117
HTTPApp.TCustomWebDispatcher.Action 15121
HTTPApp.TCustomWebDispatcher.ActionByName 15117
HTTPApp.TCustomWebDispatcher.Actions 15122
HTTPApp.TCustomWebDispatcher.AfterDispatch 15123
HTTPApp.TCustomWebDispatcher.BeforeDispatch 15124
HTTPApp.TCustomWebDispatcher.Create 15115

HTTPApp.TCustomWebDispatcher.Destroy 15115
HTTPApp.TCustomWebDispatcher.DispatchAction 15117
HTTPApp.TCustomWebDispatcher.DoAfterDispatch 15118
HTTPApp.TCustomWebDispatcher.DoBeforeDispatch 15119
HTTPApp.TCustomWebDispatcher.FinishContext 15119
HTTPApp.TCustomWebDispatcher.HandleException 15119
HTTPApp.TCustomWebDispatcher.HandleRequest 15120
HTTPApp.TCustomWebDispatcher.InitContext 15120
HTTPApp.TCustomWebDispatcher.InitModule 15120
HTTPApp.TCustomWebDispatcher.Notification 15121
HTTPApp.TCustomWebDispatcher.OnException 15124
HTTPApp.TCustomWebDispatcher.Request 15122
HTTPApp.TCustomWebDispatcher.Response 15122
HTTPApp.TDefaultWebAppServices 15125
HTTPApp.TDefaultWebAppServices.FindRequestHandler 15127
HTTPApp.TDefaultWebAppServices.FinishContext 15127
HTTPApp.TDefaultWebAppServices.GetExceptionHandler 15127
HTTPApp.TDefaultWebAppServices.GetRequestHandler 15128
HTTPApp.TDefaultWebAppServices.HandleRequest 15128
HTTPApp.TDefaultWebAppServices.InitContext 15128
HTTPApp.TDefaultWebAppServices.InvokeDispatcher 15129
HTTPApp.TDefaultWebAppServices.Request 15129
HTTPApp.TDefaultWebAppServices.RequestHandler 15130
HTTPApp.TDefaultWebAppServices.Response 15130
HTTPApp.TDefaultWebAppServices.WebModules 15130
HTTPApp.THTTPMethodEvent 15248
HTTPApp.THTTPMethodEvent type 15248
HTTPApp.TMethodType 15246
HTTPApp.TMethodType enumeration 15246
HTTPApp.TModuleAddedProc 15249
HTTPApp.TModuleAddedProc type 15249
HTTPApp.TWebActionItem 15130
HTTPApp.TWebActionItem.AssignTo 15134
HTTPApp.TWebActionItem.Create 15133
HTTPApp.TWebActionItem.Default 15135
HTTPApp.TWebActionItem.Destroy 15133
HTTPApp.TWebActionItem.Enabled 15136
HTTPApp.TWebActionItem.GetDisplayName 15134
HTTPApp.TWebActionItem.GetPathInfo 15135
HTTPApp.TWebActionItem.MethodType 15137
HTTPApp.TWebActionItem.Name 15137
HTTPApp.TWebActionItem.OnAction 15139
HTTPApp.TWebActionItem.PathInfo 15138
HTTPApp.TWebActionItem.Producer 15138
HTTPApp.TWebActionItem.ProducerContent 15139
HTTPApp.TWebActionItem.SetDisplayName 15135
HTTPApp.TWebActionItems 15140
HTTPApp.TWebActionItems.Add 15143
HTTPApp.TWebActionItems.Create 15142
HTTPApp.TWebActionItems.GetAttr 15144
HTTPApp.TWebActionItems.GetAttrCount 15144
HTTPApp.TWebActionItems.GetItemAttr 15144
HTTPApp.TWebActionItems.GetOwner 15145
HTTPApp.TWebActionItems.Items 15146
HTTPApp.TWebActionItems.SetItemName 15145
HTTPApp.TWebActionItems.Update 15146
HTTPApp.TWebActionItems.WebDispatcher 15147
HTTPApp.TWebDispatcher 15147
HTTPApp.TWebDispatcher.Actions 15152
HTTPApp.TWebDispatcher.AfterDispatch 15152
HTTPApp.TWebDispatcher.BeforeDispatch 15153
HTTPApp.TWebDispatcher.OnException 15153
HTTPApp.TWebExceptionEvent 15249
HTTPApp.TWebExceptionEvent type 15249
HTTPApp.TWebModule 15154
HTTPApp.TWebModule.Actions 15159
HTTPApp.TWebModule.AfterDispatch 15160
HTTPApp.TWebModule.BeforeDispatch 15160
HTTPApp.TWebModule.Create 15159
HTTPApp.TWebModule.OnException 15161
HTTPApp.TWebModuleCacheMode 15246
HTTPApp.TWebModuleCacheMode enumeration 15246
HTTPApp.TWebModuleContext 15249
HTTPApp.TWebModuleContext type 15249
HTTPApp.TWebModuleCreateMode 15247
HTTPApp.TWebModuleCreateMode enumeration 15247
HTTPApp.TWebRequest 15161
HTTPApp.TWebRequest.Accept 15174

HTTPApp.TWebRequest.Authorization 15175
HTTPApp.TWebRequest.CacheControl 15175
HTTPApp.TWebRequest.Connection 15176
HTTPApp.TWebRequest.Content 15176
HTTPApp.TWebRequest.ContentEncoding 15177
HTTPApp.TWebRequest.ContentFields 15177
HTTPApp.TWebRequest.ContentLength 15178
HTTPApp.TWebRequest.ContentParser 15178
HTTPApp.TWebRequest.ContentType 15179
HTTPApp.TWebRequest.ContentVersion 15179
HTTPApp.TWebRequest.Cookie 15180
HTTPApp.TWebRequest.CookieFields 15180
HTTPApp.TWebRequest.Create 15165
HTTPApp.TWebRequest.Date 15181
HTTPApp.TWebRequest.DerivedFrom 15181
HTTPApp.TWebRequest.Destroy 15165
HTTPApp.TWebRequest.Expires 15182
HTTPApp.TWebRequest.ExtractContentFields 15166
HTTPApp.TWebRequest.ExtractCookieFields 15166
HTTPApp.TWebRequest.ExtractFields 15167
HTTPApp.TWebRequest.ExtractQueryFields 15167
HTTPApp.TWebRequest.Files 15182
HTTPApp.TWebRequest.From 15182
HTTPApp.TWebRequest.GetDateVariable 15168
HTTPApp.TWebRequest.GetFieldByName 15168
HTTPApp.TWebRequest.GetIntegerVariable 15169
HTTPApp.TWebRequest.GetInternalPathInfo 15169
HTTPApp.TWebRequest.GetInternalScriptName 15169
HTTPApp.TWebRequest.GetStringVariable 15169
HTTPApp.TWebRequest.Host 15183
HTTPApp.TWebRequest.IfModifiedSince 15183
HTTPApp.TWebRequest.InternalPathInfo 15184
HTTPApp.TWebRequest.InternalScriptName 15184
HTTPApp.TWebRequest.Method 15185
HTTPApp.TWebRequest.MethodType 15185
HTTPApp.TWebRequest.PathInfo 15185
HTTPApp.TWebRequest.PathTranslated 15186
HTTPApp.TWebRequest.ProtocolVersion 15186
HTTPApp.TWebRequest.Query 15186
HTTPApp.TWebRequest.QueryFields 15187
HTTPApp.TWebRequest.ReadClient 15170
HTTPApp.TWebRequest.ReadString 15171
HTTPApp.TWebRequest.Referer 15188
HTTPApp.TWebRequest.RemoteAddr 15188
HTTPApp.TWebRequest.RemoteHost 15188
HTTPApp.TWebRequest.ScriptName 15189
HTTPApp.TWebRequest.ServerPort 15189
HTTPApp.TWebRequest.Title 15190
HTTPApp.TWebRequest.TranslateURI 15171
HTTPApp.TWebRequest.UpdateMethodType 15172
HTTPApp.TWebRequest.URL 15190
HTTPApp.TWebRequest.UserAgent 15191
HTTPApp.TWebRequest.WriteClient 15172
HTTPApp.TWebRequest.WriteHeaders 15172
HTTPApp.TWebRequest.WriteString 15173
HTTPApp.TWebResponse 15191
HTTPApp.TWebResponse.AddCustomHeaders 15197
HTTPApp.TWebResponse.Allow 15207
HTTPApp.TWebResponse.Content 15207
HTTPApp.TWebResponse.ContentEncoding 15208
HTTPApp.TWebResponse.ContentLength 15208
HTTPApp.TWebResponse.ContentStream 15209
HTTPApp.TWebResponse.ContentType 15209
HTTPApp.TWebResponse.ContentVersion 15210
HTTPApp.TWebResponse.Cookies 15210
HTTPApp.TWebResponse.Create 15194
HTTPApp.TWebResponse.CustomHeaders 15211
HTTPApp.TWebResponse.Date 15211
HTTPApp.TWebResponse.DerivedFrom 15212
HTTPApp.TWebResponse.Destroy 15195
HTTPApp.TWebResponse.Expires 15212
HTTPApp.TWebResponse.FCustomHeaders 15195
HTTPApp.TWebResponse.FHTTPRequest 15196
HTTPApp.TWebResponse.FormatAuthenticate 15197
HTTPApp.TWebResponse.GetContent 15198
HTTPApp.TWebResponse.GetCustomHeader 15198
HTTPApp.TWebResponse.GetDateVariable 15198
HTTPApp.TWebResponse.GetIntegerVariable 15199

HTTPApp.TWebResponse.GetLogMessage 15199
HTTPApp.TWebResponse.GetStatusCode 15199
HTTPApp.TWebResponse.GetStringVariable 15200
HTTPApp.TWebResponse.HTTPRequest 15213
HTTPApp.TWebResponse.LastModified 15213
HTTPApp.TWebResponse.Location 15214
HTTPApp.TWebResponse.LogMessage 15214
HTTPApp.TWebResponse.Realm 15214
HTTPApp.TWebResponse.ReasonString 15215
HTTPApp.TWebResponse.SendRedirect 15201
HTTPApp.TWebResponse.SendResponse 15201
HTTPApp.TWebResponse.SendStream 15201
HTTPApp.TWebResponse.Sent 15202
HTTPApp.TWebResponse.Server 15215
HTTPApp.TWebResponse.SetContent 15202
HTTPApp.TWebResponse.SetContentStream 15203
HTTPApp.TWebResponse.SetCookieField 15203
HTTPApp.TWebResponse.SetCustomHeader 15203
HTTPApp.TWebResponse.SetDateVariable 15204
HTTPApp.TWebResponse.SetIntegerVariable 15204
HTTPApp.TWebResponse.SetLogMessage 15205
HTTPApp.TWebResponse.SetStatusCode 15205
HTTPApp.TWebResponse.SetStringVariable 15205
HTTPApp.TWebResponse.StatusCode 15215
HTTPApp.TWebResponse.Title 15217
HTTPApp.TWebResponse.Version 15217
HTTPApp.TWebResponse.WWWAuthenticate 15218
HTTPApp.UnixPathToDosPath 15245
HTTPApp.UnixPathToDosPath function 15245
HTTPProd 15253
HTTPProd namespace 15253
- Classes 15255
- Constants 15341
- Functions 15333
- Interfaces 15317
- Structs, Records, Enums 15336
- Types 15338
- Variables 15340

HTTPProd.ContentFromScriptFile 15333
HTTPProd.ContentFromScriptFile function 15333
HTTPProd.ContentFromScriptStream 15333
HTTPProd.ContentFromScriptStream function 15333
HTTPProd.ContentFromStream 15334
HTTPProd.ContentFromStream function 15334
HTTPProd.ContentFromString 15334
HTTPProd.ContentFromString function 15334
HTTPProd.DesignerFileManager 15340
HTTPProd.DesignerFileManager variable 15340
HTTPProd.EScriptError 15255
HTTPProd.EScriptError.Content 15258
HTTPProd.EScriptError.Create 15257
HTTPProd.EScriptError.Destroy 15257
HTTPProd.EScriptError.Errors 15258
HTTPProd.FindComponentWebModuleContext 15334
HTTPProd.FindComponentWebModuleContext function 15334
HTTPProd.GetTagID 15335
HTTPProd.GetTagID function 15335
HTTPProd.HTMLAlign 15341
HTTPProd.HTMLAlign constant 15341
HTTPProd.HTMLVAlign 15341
HTTPProd.HTMLVAlign constant 15341
HTTPProd.IDesignerFileManager 15317
HTTPProd.IDesignerFileManager.GetStream 15318
HTTPProd.IDesignerFileManager.QualifyFileName 15318
HTTPProd.IGetLocateFileService 15318
HTTPProd.IGetLocateFileService.GetLocateFileService 15319
HTTPProd.IGetProducerTemplate 15319
HTTPProd.IGetProducerTemplate.GetProducerTemplateFileName 15320
HTTPProd.IGetProducerTemplate.GetProducerTemplateStream 15320
HTTPProd.ILocateFileService 15320
HTTPProd.ILocateFileService.GetTemplateStream 15321
HTTPProd.IScriptContext 15321
HTTPProd.IScriptContext.GetWebModuleContext 15322
HTTPProd.IScriptContext.WebModuleContext 15322
HTTPProd.IScriptProducer 15323
HTTPProd.IScriptProducer.Content 15327

HTTPProd.IScriptProducer.Errors 15327
HTTPProd.IScriptProducer.Evaluate 15324
HTTPProd.IScriptProducer.GetContent 15324
HTTPProd.IScriptProducer.GetErrors 15324
HTTPProd.IScriptProducer.GetHTMLBlock 15325
HTTPProd.IScriptProducer.GetHTMLBlockCount 15325
HTTPProd.IScriptProducer.HandleScriptError 15325
HTTPProd.IScriptProducer.HTMLBlockCount 15328
HTTPProd.IScriptProducer.HTMLBlocks 15328
HTTPProd.IScriptProducer.ParseStream 15325
HTTPProd.IScriptProducer.ParseString 15325
HTTPProd.IScriptProducer.ReplaceTags 15326
HTTPProd.IScriptProducer.SetContent 15326
HTTPProd.IScriptProducer.Write 15326
HTTPProd.IScriptProducer.WriteItem 15327
HTTPProd.IWebVariableName 15328
HTTPProd.IWebVariableName.GetVariableName 15329
HTTPProd.IWebVariableName.VariableName 15329
HTTPProd.IWebVariablesContainer 15329
HTTPProd.IWebVariablesContainer.FindVariable 15330
HTTPProd.IWebVariablesContainer.FindVariableContainer 15331
HTTPProd.IWebVariablesContainer.GetVariable 15331
HTTPProd.IWebVariablesContainer.GetVariableCount 15331
HTTPProd.IWebVariablesContainer.VariableCount 15331
HTTPProd.IWebVariablesContainer.Variables 15332
HTTPProd.ScriptEnginesList 15340
HTTPProd.ScriptEnginesList variable 15340
HTTPProd.TAbstractScriptEnginesList 15258
HTTPProd.TAbstractScriptEnginesList.Create 15259
HTTPProd.TAbstractScriptEnginesList.FindScriptProducerClass 15260
HTTPProd.TAbstractScriptError 15260
HTTPProd.TAbstractScriptError.CharPos 15263
HTTPProd.TAbstractScriptError.Description 15263
HTTPProd.TAbstractScriptError.FileName 15264
HTTPProd.TAbstractScriptError.GetCharPos 15262
HTTPProd.TAbstractScriptError.GetDescription 15262
HTTPProd.TAbstractScriptError.GetFileName 15262
HTTPProd.TAbstractScriptError.GetLine 15262
HTTPProd.TAbstractScriptError.GetSourceLine 15263
HTTPProd.TAbstractScriptError.Line 15264
HTTPProd.TAbstractScriptError.SetDescription 15263
HTTPProd.TAbstractScriptError.SourceLine 15264
HTTPProd.TAbstractScriptErrors 15264
HTTPProd.TAbstractScriptErrors.Add 15266
HTTPProd.TAbstractScriptErrors.Count 15267
HTTPProd.TAbstractScriptErrors.Errors 15267
HTTPProd.TAbstractScriptErrors.GetError 15266
HTTPProd.TAbstractScriptErrors.GetErrorCount 15266
HTTPProd.TAbstractScriptProducer 15267
HTTPProd.TAbstractScriptProducer.Create 15269
HTTPProd.TBasePageProducer 15269
HTTPProd.TBasePageProducer.Content 15274
HTTPProd.TBasePageProducer.ContentFromStream 15275
HTTPProd.TBasePageProducer.ContentFromString 15275
HTTPProd.TBasePageProducer.Create 15274
HTTPProd.TBasePageProducer.DoTagEvent 15276
HTTPProd.TBasePageProducer.GetProducerTemplateFileNam e 15276
HTTPProd.TBasePageProducer.GetProducerTemplateStream 15276
HTTPProd.TBasePageProducer.GetScriptEngine 15277
HTTPProd.TBasePageProducer.GetTagID 15277
HTTPProd.TBasePageProducer.GetTemplateFileName 15277
HTTPProd.TBasePageProducer.GetTemplateStream 15277
HTTPProd.TBasePageProducer.HandleScriptTag 15277
HTTPProd.TBasePageProducer.HandleTag 15278
HTTPProd.TBasePageProducer.ImplHandleTag 15278
HTTPProd.TBasePageProducer.OnHTMLTag 15280
HTTPProd.TBasePageProducer.ScriptEngine 15279
HTTPProd.TBasePageProducer.ServerScriptFromStream 15278
HTTPProd.TBasePageProducer.StripParamQuotes 15279
HTTPProd.TBasePageProducer.UseScriptEngine 15278
HTTPProd.TBasePageProducer.WebModuleContext 15279
HTTPProd.TCustomPageProducer 15281
HTTPProd.TCustomPageProducer.Create 15285
HTTPProd.TCustomPageProducer.Destroy 15285
HTTPProd.TCustomPageProducer.GetTemplateFileName

15286
HTTPProd.TCustomPageProducer.GetTemplateStream 15286
HTTPProd.TCustomPageProducer.HandleTag 15286
HTTPProd.TCustomPageProducer.HTMLDoc 15287
HTTPProd.TCustomPageProducer.HTMLFile 15287
HTTPProd.THandledTagProc 15338
HTTPProd.THandledTagProc type 15338
HTTPProd.THandleTagProc 15338
HTTPProd.THandleTagProc type 15338
HTTPProd.THTMLAlign 15336
HTTPProd.THTMLAlign enumeration 15336
HTTPProd.THTMLBgColor 15339
HTTPProd.THTMLBgColor type 15339
HTTPProd.THTMLTableAttributes 15288
HTTPProd.THTMLTableAttributes.Align 15292
HTTPProd.THTMLTableAttributes.AssignTo 15291
HTTPProd.THTMLTableAttributes.BgColor 15293
HTTPProd.THTMLTableAttributes.Border 15293
HTTPProd.THTMLTableAttributes.CellPadding 15294
HTTPProd.THTMLTableAttributes.CellSpacing 15294
HTTPProd.THTMLTableAttributes.Create 15290
HTTPProd.THTMLTableAttributes.RestoreDefaults 15291
HTTPProd.THTMLTableAttributes.Width 15294
HTTPProd.THTMLTableCellAttributes 15295
HTTPProd.THTMLTableElementAttributes 15295
HTTPProd.THTMLTableElementAttributes.Align 15299
HTTPProd.THTMLTableElementAttributes.AssignTo 15297
HTTPProd.THTMLTableElementAttributes.BgColor 15299
HTTPProd.THTMLTableElementAttributes.RestoreDefaults 15298
HTTPProd.THTMLTableElementAttributes.VAlign 15300
HTTPProd.THTMLTableHeaderAttributes 15300
HTTPProd.THTMLTableHeaderAttributes.AssignTo 15302
HTTPProd.THTMLTableHeaderAttributes.Caption 15303
HTTPProd.THTMLTableHeaderAttributes.RestoreDefaults 15303
HTTPProd.THTMLTableRowAttributes 15304
HTTPProd.THTMLTagAttributes 15304
HTTPProd.THTMLTagAttributes.Changed 15307
HTTPProd.THTMLTagAttributes.Create 15306
HTTPProd.THTMLTagAttributes.Custom 15307
HTTPProd.THTMLTagAttributes.OnChange 15308
HTTPProd.THTMLTagAttributes.Producer 15308
HTTPProd.THTMLTagAttributes.RestoreDefaults 15307
HTTPProd.THTMLTagEvent 15339
HTTPProd.THTMLTagEvent type 15339
HTTPProd.THTMLVAlign 15336
HTTPProd.THTMLVAlign enumeration 15336
HTTPProd.TPageProducer 15309
HTTPProd.TPageProducer.HTMLDoc 15314
HTTPProd.TPageProducer.HTMLFile 15314
HTTPProd.TPageProducer.OnHTMLTag 15316
HTTPProd.TPageProducer.ScriptEngine 15315
HTTPProd.TPageProducer.StripParamQuotes 15315
HTTPProd.TScriptProducerClass 15339
HTTPProd.TScriptProducerClass type 15339
HTTPProd.TTag 15337
HTTPProd.TTag enumeration 15337
HTTPUtil 15343
HTTPUtil namespace 15343
    Functions 15353
    Interfaces 15344
HTTPUtil.FirstDelimiter 15353
HTTPUtil.FirstDelimiter function 15353
HTTPUtil.GetDefaultStreamLoader 15354
HTTPUtil.GetDefaultStreamLoader function 15354
HTTPUtil.HTMLEscape 15354
HTTPUtil.HTMLEscape function 15354
HTTPUtil.IStreamLoader 15344
HTTPUtil.IStreamLoader.GetPassword 15345
HTTPUtil.IStreamLoader.GetProxy 15346
HTTPUtil.IStreamLoader.GetTimeout 15346
HTTPUtil.IStreamLoader.GetUserName 15346
HTTPUtil.IStreamLoader.Load 15346
HTTPUtil.IStreamLoader.Password 15348
HTTPUtil.IStreamLoader.Proxy 15349
HTTPUtil.IStreamLoader.SetPassword 15347
HTTPUtil.IStreamLoader.SetProxy 15347
HTTPUtil.IStreamLoader.SetTimeout 15348

HTTPUtil.IStreamLoader.SetUserName 15348
HTTPUtil.IStreamLoader.Timeout 15349
HTTPUtil.IStreamLoader.UserName 15349
HTTPUtil.IStringTokenizer 15350
HTTPUtil.IStringTokenizer.countTokens 15352
HTTPUtil.IStringTokenizer.getTokenCounts 15351
HTTPUtil.IStringTokenizer.hasMoreTokens 15351
HTTPUtil.IStringTokenizer.nextToken 15351
HTTPUtil.StartsWith 15354
HTTPUtil.StartsWith function 15354
HTTPUtil.StringTokenizer 15355
HTTPUtil.StringTokenizer function 15355
HTTPUtil.StringToStringArray 15355
HTTPUtil.StringToStringArray function 15355
htUnderlineCold enumeration member 5456
htUnderlineHot enumeration member 5456

I

IActionBarDesigner interface 560
- about IActionBarDesigner interface 560
- CreateAction 561
- EditAction 561
- IActionBarDesigner Members 560
- IActionBarDesigner methods 560
- Modified 561
- SetActiveMenu 561
- SetItemSelection 562
- SetSelection 562

IActionFieldValue interface 920
- about IActionFieldValue interface 920
- FieldName 923
- FileCount 924
- Files 924
- GetFieldName 922
- GetFile 922
- GetFileCount 922
- GetValue 923
- GetValueCount 923
- IActionFieldValue Members 921
- IActionFieldValue methods 921
- IActionFieldValue properties 923
- ValueCount 925
- Values 925

IActionFieldValues interface 925
- about IActionFieldValues interface 925
- AddValue 927
- FieldCount 930
- FieldNames 931
- GetFieldCount 927
- GetFieldName 928
- GetOrigValue 928
- GetValue 928
- IActionFieldValues Members 926
- IActionFieldValues methods 927
- IActionFieldValues properties 930
- IndexOfField 929
- OrigValueOfField 929
- OrigValues 931
- ValueOfField 930
- Values 932

IActionFieldValuesOfAdapter interface 933
- about IActionFieldValuesOfAdapter interface 933
- IActionFieldValuesOfAdapter Members 933
- IActionFieldValuesOfAdapter methods 933
- LocateActionFieldValuesOfAdapter 934

IActionHandleResponse interface 934
- about IActionHandleResponse interface 934
- HandleResponse 935
- IActionHandleResponse Members 934
- IActionHandleResponse methods 935

IActionImageProducer interface 19646
- about IActionImageProducer interface 19646
- GetAdapterEventImage 19648
- GetDisplayStyle 19648
- IActionImageProducer Members 19647
- IActionImageProducer methods 19647

IActionRequest interface 935
- about IActionRequest interface 935

ActionName 938
GetActionName 937
GetHandled 937
Handled 938
IActionRequest Members 936
IActionRequest methods 936
IActionRequest properties 937
SetHandled 937
IActionRespondWith interface 938
about IActionRespondWith interface 938
GetRedirectOptions 939
GetRespondWith 939
IActionRespondWith Members 939
IActionRespondWith methods 939
IActionRespondWith properties 941
RedirectOptions 941
RespondWith 941
RespondWithComponentPage 940
RespondWithPage 940
RespondWithURL 940
SetExecuteStatus 940
SetRedirectOptions 941
IActionResponse interface 941
about IActionResponse interface 941
ExecuteStatus 944
GetExecuteStatus 942
GetHandled 943
Handled 944
IActionResponse Members 942
IActionResponse methods 942
IActionResponse properties 944
SetExecuteStatus 943
SetHandled 943
IActivatable interface 11134
about IActivatable interface 11134
Activate 11135
IActivatable Members 11135
IActivatable methods 11135
IAdapterAccess interface 19649
about IAdapterAccess interface 19649
HasExecuteAccess 19650
HasModifyAccess 19650
HasViewAccess 19650
IAdapterAccess Members 19649
IAdapterAccess methods 19649
IAdapterActionAccess interface 19650
about IAdapterActionAccess interface 19650
HasExecuteAccess 19651
IAdapterActionAccess Members 19651
IAdapterActionAccess methods 19651
IAdapterActionAttributes interface 19651
about IAdapterActionAttributes interface 19651
GetVisible 19652
IAdapterActionAttributes Members 19652
IAdapterActionAttributes methods 19652
IAdapterActionAttributes properties 19652
Visible 19652
IAdapterConverter interface 20032
about IAdapterConverter interface 20032
GetDomConverter 20033
IAdapterConverter Members 20032
IAdapterConverter methods 20032
IAdapterDataSetComponent interface 8102
about IAdapterDataSetComponent interface 8102
DataSet 8103
GetDataSet 8103
IAdapterDataSetComponent Members 8102
IAdapterDataSetComponent methods 8103
IAdapterDataSetComponent properties 8103
SetDataSet 8103
IAdapterDispatcher interface 19653
about IAdapterDispatcher interface 19653
EncodeAdapterItemRequestAsFieldValue 19654
EncodeAdapterItemRequestAsHREF 19654
GetAdapterItemRequestParams 19655
IAdapterDispatcher Members 19653
IAdapterDispatcher methods 19654
IAdapterDispatchParams interface 19655

about IAdapterDispatchParams interface 19655
GetHandler 19657
GetObjectIdentifier 19657
GetParams 19657
GetRequestIdentifier 19657
Handler 19658
IAdapterDispatchParams Members 19656
IAdapterDispatchParams methods 19656
IAdapterDispatchParams properties 19658
ObjectIdentifier 19658
Params 19659
RequestIdentifier 19659
IAdapterDisplay interface 25303
about IAdapterDisplay interface 25303
IAdapterEditor interface 24566
about IAdapterEditor interface 24566
CanAddActionClass 24567
CanAddFieldClass 24567
IAdapterEditor Members 24567
IAdapterEditor methods 24567
IAdapterError interface 19660
about IAdapterError interface 19660
ErrorText 19662
Field 19662
GetErrorText 19660
GetField 19661
GetID 19661
IAdapterError Members 19660
IAdapterError methods 19660
IAdapterError properties 19661
ID 19662
IAdapterErrors interface 19663
about IAdapterErrors interface 19663
ClearErrors 19664
DefineLabel 19664
ErrorCount 19666
Errors 19666
GetError 19664
GetErrorCount 19665
GetObjectErrors 19665
HasObjectErrors 19665
IAdapterErrors Members 19663
IAdapterErrors methods 19663
IAdapterErrors properties 19665
IAdapterErrorsStub interface 1434
about IAdapterErrorsStub interface 1434
DefineLabel 1435
FieldErrors 1435
IAdapterErrorsStub Members 1434
IAdapterErrorsStub methods 1435
IAdapterErrorStub interface 1435
about IAdapterErrorStub interface 1435
Field 1437
Get_Field 1436
Get_ID 1436
Get_Message 1437
IAdapterErrorStub Members 1436
IAdapterErrorStub methods 1436
IAdapterErrorStub properties 1437
ID 1437
Message 1437
IAdapterFieldAccess interface 19666
about IAdapterFieldAccess interface 19666
HasModifyAccess 19667
HasViewAccess 19667
IAdapterFieldAccess Members 19667
IAdapterFieldAccess methods 19667
IAdapterFieldAttributes interface 19667
about IAdapterFieldAttributes interface 19667
GetRequired 19668
GetVisible 19668
IAdapterFieldAttributes Members 19668
IAdapterFieldAttributes methods 19668
IAdapterFieldAttributes properties 19669
Required 19669
Visible 19669
IAdapterFieldStub interface 1438
about IAdapterFieldStub interface 1438

CanModify 1444
CanView 1444
DisplayLabel 1445
DisplayStyle 1445
DisplayText 1445
DisplayWidth 1445
EditText 1445
Get_CanModify 1439
Get_CanView 1440
Get_DisplayLabel 1440
Get_DisplayStyle 1440
Get_DisplayText 1440
Get_DisplayWidth 1441
Get_EditText 1441
Get_Image 1441
Get_InputName 1441
Get_InputStyle 1441
Get_MaxLength 1442
Get_Name 1442
Get_Required 1442
Get_Value 1442
Get_Values 1443
Get_ValuesList 1443
Get_ViewMode 1443
Get_Visible 1443
IAdapterFieldStub Members 1438
IAdapterFieldStub methods 1439
IAdapterFieldStub properties 1444
Image 1446
InputName 1446
InputStyle 1446
IsEqual 1443
MaxLength 1446
Name 1447
Required 1447
Value 1447
Values 1447
ValuesList 1447
ViewMode 1448
Visible 1448
IAdapterFieldValuesStub interface 1448
about IAdapterFieldValuesStub interface 1448
Get_Value 1449
HasValue 1449
IAdapterFieldValuesStub Members 1448
IAdapterFieldValuesStub methods 1449
IAdapterFieldValuesStub properties 1450
Value 1450
IAdapterHiddenField interface 19669
about IAdapterHiddenField interface 19669
GetName 19670
GetValue 19670
IAdapterHiddenField Members 19670
IAdapterHiddenField methods 19670
IAdapterHiddenField properties 19671
Name 19671
Value 19671
IAdapterHiddenFields interface 19671
about IAdapterHiddenFields interface 19671
FieldCount 19673
Fields 19673
GetField 19672
GetFieldCount 19672
IAdapterHiddenFields Members 19672
IAdapterHiddenFields methods 19672
IAdapterHiddenFields properties 19672
IAdapterHiddenFieldsStub interface 1450
about IAdapterHiddenFieldsStub interface 1450
IAdapterHiddenFieldsStub Members 1450
IAdapterHiddenFieldsStub methods 1450
WriteFields 1451
IAdapterHiddenFieldStub interface 1451
about IAdapterHiddenFieldStub interface 1451
Get_Name 1452
Get_Value 1452
IAdapterHiddenFieldStub Members 1451
IAdapterHiddenFieldStub methods 1452
IAdapterHiddenFieldStub properties 1452

Name 1452
Value 1453
IAdapterImageStub interface 1453
about IAdapterImageStub interface 1453
AsHREF 1454
Get_AsHREF 1454
IAdapterImageStub Members 1453
IAdapterImageStub methods 1453
IAdapterImageStub properties 1454
IAdapterItemRequestParams interface 19673
about IAdapterItemRequestParams interface 19673
FailurePage 19676
GetFailurePage 19674
GetParamValues 19675
GetSuccessPage 19675
IAdapterItemRequestParams Members 19674
IAdapterItemRequestParams methods 19674
IAdapterItemRequestParams properties 19676
ParamValues 19677
SetFailurePage 19675
SetSuccessPage 19676
SuccessPage 19677
IAdapterMode interface 19677
about IAdapterMode interface 19677
AdapterMode 19679
GetAdapterMode 19678
GetNeedAdapterMode 19678
IAdapterMode Members 19678
IAdapterMode methods 19678
IAdapterMode properties 19679
NeedAdapterMode 19679
RestoreDefaultAdapterMode 19679
SetAdapterMode 19679
IAdapterModes interface 944
about IAdapterModes interface 944
AdapterCount 947
AdapterModes 948
AdapterNames 948
FindAdapterMode 946
GetAdapterCount 946
GetAdapterMode 947
GetAdapterName 947
IAdapterModes Members 945
IAdapterModes methods 946
IAdapterModes properties 947
IAdapterNotifyAdapterChange interface 24567
about IAdapterNotifyAdapterChange interface 24567
IAdapterNotifyAdapterChange Members 24568
IAdapterNotifyAdapterChange methods 24568
NotifyAdapterChange 24568
IAdapterRecordKeys interface 949
about IAdapterRecordKeys interface 949
GetRecordKeyCount 950
GetRecordKeyName 950
GetRecordKeyValue 951
IAdapterRecordKeys Members 950
IAdapterRecordKeys methods 950
IAdapterRecordKeys properties 951
RecordKeyNames 951
RecordKeyValues 952
IAdapterRecordStub interface 1454
about IAdapterRecordStub interface 1454
IAdapterRequestHandler interface 19680
about IAdapterRequestHandler interface 19680
CreateRequestContext 19681
HandleRequest 19681
IAdapterRequestHandler Members 19680
IAdapterRequestHandler methods 19680
IAdapterRequestParams interface 952
about IAdapterRequestParams interface 952
AssignTo 954
GetParamCount 954
GetParamName 954
GetParamValue 955
IAdapterRequestParams Members 953
IAdapterRequestParams methods 953
IAdapterRequestParams properties 956
IndexOfParam 955

ParamCount 956
ParamNames 956
ParamValues 957
ValueOfParam 955
IAdapterStub interface 1455
about IAdapterStub interface 1455
CanModify 1458
CanView 1458
Errors 1458
Get_CanModify 1456
Get_CanView 1456
Get_Errors 1456
Get_HiddenFields 1457
Get_HiddenRecordFields 1457
Get_Mode 1457
HiddenFields 1458
HiddenRecordFields 1459
IAdapterStub Members 1455
IAdapterStub methods 1456
IAdapterStub properties 1457
Mode 1459
Set_Mode 1457
IAdapterValuesListStub interface 1459
about IAdapterValuesListStub interface 1459
Get_Image 1460
Get_Value 1460
Get_ValueName 1461
IAdapterValuesListStub Members 1459
IAdapterValuesListStub methods 1460
IAdapterValuesListStub properties 1461
Image 1461
ImageOfValue 1461
NameOfValue 1461
Value 1462
ValueName 1462
IAddFieldsEditor interface 24769
about IAddFieldsEditor interface 24769
GetAddAllFieldsItem 24770
GetAddFieldsDlgCaption 24770
GetAddFieldsItem 24770
GetNewFieldPrefix 24771
IAddFieldsEditor Members 24769
IAddFieldsEditor methods 24770
iaLeft enumeration member 5452
IAlternateDockHost interface 11643
about IAlternateDockHost interface 11643
IAlternateDockHost Members 11643
IAlternateDockHost methods 11643
SetAlternateDockHost 11644
IAmbientDispatch interface 1702
about IAmbientDispatch interface 1702
AutoClip 1703
BackColor 1703
DisplayAsDefault 1703
DisplayName 1704
Font 1704
ForeColor 1704
IAmbientDispatch Members 1702
IAmbientDispatch properties 1703
LocaleID 1704
MessageReflect 1705
ScaleUnits 1705
ShowGrabHandles 1705
ShowHatching 1705
SupportsMnemonics 1705
TextAlign 1706
UIDead 1706
UserMode 1706
IApplicationStub interface 1462
about IApplicationStub interface 1462
Designing 1465
Get_Designing 1463
Get_ModuleFileName 1464
Get_ModulePath 1464
Get_Title 1464
IApplicationStub Members 1462
IApplicationStub methods 1463
IApplicationStub properties 1465

ModuleFileName 1465
ModulePath 1465
QualifyFileName 1464
Title 1465
IAppServer interface 16450
about IAppServer interface 16450
AS_ApplyUpdates 16451
AS_DataRequest 16452
AS_Execute 16453
AS_GetParams 16454
AS_GetProviderNames 16454
AS_GetRecords 16455
AS_RowRequest 16456
IAppServer Members 16451
IAppServer methods 16451
IAppServerDisp interface 16457
about IAppServerDisp interface 16457
AS_ApplyUpdates 16458
AS_DataRequest 16458
AS_Execute 16458
AS_GetParams 16458
AS_GetProviderNames 16459
AS_GetRecords 16459
AS_RowRequest 16459
IAppServerDisp Members 16457
IAppServerDisp methods 16457
IAppServerSOAP interface 20012
about IAppServerSOAP interface 20012
IAppServerSOAP Members 20012
IAppServerSOAP methods 20013
SAS_ApplyUpdates 20013
SAS_DataRequest 20013
SAS_Execute 20014
SAS_GetParams 20015
SAS_GetProviderNames 20015
SAS_GetRecords 20015
SAS_RowRequest 20016
IAsFormatted interface 19681
about IAsFormatted interface 19681
AsFormatted 19682
IAsFormatted Members 19682
IAsFormatted methods 19682
iaTop enumeration member 5452
IBaseAdapterStub interface 1466
about IBaseAdapterStub interface 1466
Actions 1468
Fields 1468
Get_Actions 1467
Get_ClassName_ 1467
Get_Fields 1467
Get_Name_ 1467
IBaseAdapterStub Members 1466
IBaseAdapterStub methods 1466
IBaseAdapterStub properties 1467
IBaseRecordsAdapterStub interface 1468
about IBaseRecordsAdapterStub interface 1468
Get_Records 1469
IBaseRecordsAdapterStub Members 1468
IBaseRecordsAdapterStub methods 1469
IBaseRecordsAdapterStub properties 1469
Records 1469
IBinding interface 25815
about IBinding interface 25815
BindingOperations 25819
GetBindingOperations 25817
GetName 25818
GetType_ 25818
IBinding Members 25816
IBinding methods 25817
IBinding properties 25818
Name 25819
SetName 25818
SetType_ 25818
Type_ 25819
IBindingInput interface 25819
about IBindingInput interface 25819
IBindingOperation interface 25820
about IBindingOperation interface 25820

Faults 25823
GetFaults 25822
GetInput 25822
GetName 25822
GetOutput 25822
IBindingOperation Members 25820
IBindingOperation methods 25822
IBindingOperation properties 25823
Input 25823
Name 25824
Output 25824
SetName 25823
IBindingOperations interface 25824
about IBindingOperations interface 25824
Add 25826
BindingOperation 25827
GetBindingOperation 25826
IBindingOperations Members 25824
IBindingOperations methods 25826
IBindingOperations properties 25827
Insert 25827
IBindingOutput interface 25827
about IBindingOutput interface 25827
IBindings interface 25828
about IBindings interface 25828
Add 25830
Binding 25831
GetBinding 25830
IBindings Members 25828
IBindings methods 25829
IBindings properties 25830
Insert 25830
IBOA interface 18189
about IBOA interface 18189
Deactivate 18190
GetPrincipal 18190
GetPrincipalLength 18190
IBOA Members 18189
IBOA methods 18190
ImplIsReady 18190
ObjIsReady 18191
IChangeNotifier interface 14606
about IChangeNotifier interface 14606
Changed 14607
IChangeNotifier Members 14606
IChangeNotifier methods 14606
ICheckOrigValue interface 24568
about ICheckOrigValue interface 24568
CheckOrigValue 24569
ICheckOrigValue Members 24569
ICheckOrigValue methods 24569
ICheckValueChange interface 24569
about ICheckValueChange interface 24569
CheckValueChange 24570
ICheckValueChange Members 24570
ICheckValueChange methods 24570
IClass interface 11135
about IClass interface 11135
ClassName 11137
ClassNameIs 11136
ClassParent 11138
GetClassName 11137
GetClassParent 11137
GetUnitName 11137
IClass Members 11136
IClass methods 11136
IClass properties 11137
UnitName 11138
IClearAdapterValues interface 24570
about IClearAdapterValues interface 24570
ClearAdapterValues 24571
IClearAdapterValues Members 24571
IClearAdapterValues methods 24571
IComponentEditor interface 11138
about IComponentEditor interface 11138
Copy 11140
Edit 11140
ExecuteVerb 11140

GetComponent 11141
GetDesigner 11141
GetVerb 11141
GetVerbCount 11142
IComponentEditor Members 11139
IComponentEditor methods 11139
IsInInlined 11142
PrepareItem 11142
IComponentGuidelines interface 11143
about IComponentGuidelines interface 11143
Count 11145
DesignerGuideOffsets 11145
DesignerGuideTypes 11145
GetCount 11144
GetDesignerGuideOffset 11144
GetDesignerGuideType 11144
IComponentGuidelines Members 11143
IComponentGuidelines methods 11144
IComponentGuidelines properties 11144
ICorbaObj interface 18191
about ICorbaObj interface 18191
CorbaObject 18192
ICorbaObj Members 18191
ICorbaObj methods 18191
IsLocal 18192
ICorbaObject interface 6475
about ICorbaObject interface 6475
Hash 6476
ICorbaObject Members 6475
ICorbaObject methods 6475
IsA 6476
NonExistent 6476
SetPrincipal 6476
ICreateActionRequestContext interface 24571
about ICreateActionRequestContext interface 24571
CreateActionRequestContext 24572
ICreateActionRequestContext Members 24571
ICreateActionRequestContext methods 24572
ICustomHelpViewer interface 15013
about ICustomHelpViewer interface 15013
CanShowTableOfContents 15014
GetHelpStrings 15015
GetViewerName 15015
ICustomHelpViewer Members 15014
ICustomHelpViewer methods 15014
NotifyID 15016
ShowHelp 15016
ShowTableOfContents 15016
ShutDown 15017
SoftShutDown 15017
UnderstandsKeyword 15017
ICustomModule interface 11145
about ICustomModule interface 11145
ExecuteVerb 11146
GetAttributes 11147
GetVerb 11147
GetVerbCount 11147
ICustomModule Members 11146
ICustomModule methods 11146
Nestable 11147
PrepareItem 11147
Saving 11148
ValidateComponent 11148
ValidateComponentClass 11148
IDataBlock interface 18983
about IDataBlock interface 18983
BytesReserved 18989
Clear 18984
GetBytesReserved 18985
GetMemory 18985
GetSignature 18985
GetSize 18986
GetStream 18986
IDataBlock Members 18984
IDataBlock methods 18984
IDataBlock properties 18989
IgnoreStream 18986
InitData 18987

Memory 18989
Read 18987
SetSignature 18988
SetSize 18988
Signature 18990
Size 18990
Stream 18991
Write 18988
IDataIntercept interface 18991
about IDataIntercept interface 18991
DataIn 18992
DataOut 18993
IDataIntercept Members 18992
IDataIntercept methods 18992
IDataSetAdapterFieldClass interface 8104
about IDataSetAdapterFieldClass interface 8104
IDataSetComponent interface 17053
about IDataSetComponent interface 17053
DataSet 17055
GetDataSet 17054
GetXMLBroker 17054
GetXMLDataSet 17055
IDataSetComponent Members 17054
IDataSetComponent methods 17054
IDataSetComponent properties 17055
XMLBroker 17056
XMLDataSet 17056
IDataSetField interface 17057
about IDataSetField interface 17057
FieldName 17058
GetFieldName 17057
GetRowSetFieldAttributes 17058
IDataSetField Members 17057
IDataSetField methods 17057
IDataSetField properties 17058
SetFieldName 17058
IDataSetFields interface 17059
about IDataSetFields interface 17059
GetFieldsList 17060
GetVisibleFields 17060
HasStatusField 17060
IDataSetFields Members 17059
IDataSetFields methods 17059
IDataSetFields properties 17061
IsFieldInUse 17060
ValidateDataSet 17060
VisibleFields 17061
IDBApplication interface 7569
about IDBApplication interface 7569
GetTitle 7570
IDBApplication Members 7569
IDBApplication methods 7570
IDBApplication properties 7571
ProcessMessages 7570
SetTitle 7570
Title 7571
IDBScreen interface 7571
about IDBScreen interface 7571
Cursor 7573
GetCursor 7572
IDBScreen Members 7572
IDBScreen methods 7572
IDBScreen properties 7573
SetCursor 7572
IDBSession interface 7573
about IDBSession interface 7573
AddPassword 7574
IDBSession Members 7574
IDBSession methods 7574
RemoveAllPasswords 7575
RemovePassword 7575
IDefaultEditor interface 11148
about IDefaultEditor interface 11148
IDefaultPageFileName interface 19682
about IDefaultPageFileName interface 19682
IDefaultPageFileName Members 19683
IDefaultPageFileName methods 19683
SetDefaultPageFileName 19683

IDefinition interface 25831
about IDefinition interface 25831
Bindings 25836
GetBindings 25833
GetImports 25833
GetMessages 25834
GetName 25834
GetPortTypes 25834
GetServices 25834
GetTargetNamespace 25834
GetTypes 25835
IDefinition Members 25831
IDefinition methods 25833
IDefinition properties 25835
Imports 25836
Messages 25836
Name 25836
PortTypes 25836
Services 25837
SetName 25835
SetTargetNamespace 25835
TargetNamespace 25837
Types 25837
IDesigner interface 11149
about IDesigner interface 11149
DesignerExtention 11151
GetDesignerExtension 11151
IDesigner Members 11149
IDesigner methods 11151
IDesigner properties 11151
IDesigner60 interface 11151
about IDesigner60 interface 11151
Activate 11154
AncestorDesigner 11166
CanPaste 11154
ChainCall 11155
ChainCall method 11154
ClearSelection 11155
CopySelection 11155
CreateComponent 11156
CreateCurrentComponent 11156
CreateMethod 11156
CutSelection 11156
DeleteSelection 11157
Edit 11157
GetAncestorDesigner 11157
GetBaseRegKey 11157
GetComponent 11157
GetComponentName 11158
GetComponentNames 11158
GetIDEOptions 11158
GetIsDormant 11158
GetMethodName 11159
GetMethods 11159
GetObject 11159
GetObjectName 11159
GetObjectNames 11159
GetPathAndBaseExeName 11160
GetPrivateDirectory 11160
GetProjectModules 11160
GetRoot 11160
GetRootClassName 11161
GetScrollRanges 11161
GetSelections 11161
GetShiftState 11161
IDesigner60 Members 11152
IDesigner60 methods 11153
IDesigner60 properties 11166
IsComponentHidden 11161
IsComponentLinkable 11162
IsDormant 11166
IsSourceReadOnly 11162
MakeComponentLinkable 11162
MethodExists 11162
MethodFromAncestor 11163
ModalEdit 11163
Modified 11163
ModuleFileNames 11163

NoSelection 11163
PasteSelection 11164
RenameMethod 11164
Resurrect 11164
Revert 11164
Root 11166
SelectComponent 11165
SelectItemName 11165
SetSelections 11165
ShowMethod 11165
UniqueName 11165
IDesigner70 interface 11167
about IDesigner70 interface 11167
ActiveClassGroup 11169
FindRootAncestor 11168
GetActiveClassGroup 11169
IDesigner70 Members 11167
IDesigner70 methods 11168
IDesigner70 properties 11169
IDesigner80 interface 11169
about IDesigner80 interface 11169
CreateMethod 11171
GetMethods 11171
IDesigner80 Members 11170
IDesigner80 methods 11171
SelectComponent 11171
IDesignerFileManager interface 15317
about IDesignerFileManager interface 15317
GetStream 15318
IDesignerFileManager Members 15317
IDesignerFileManager methods 15317
QualifyFileName 15318
IDesignerHook interface 14334
about IDesignerHook interface 14334
Form 14339
GetCustomForm 14335
GetIsControl 14336
GetRoot 14336
IDesignerHook Members 14335
IDesignerHook methods 14335
IDesignerHook properties 14339
IsControl 14339
IsDesignMsg 14336
PaintGrid 14337
PaintMenu 14337
SetCustomForm 14337
SetIsControl 14338
UniqueName 14338
ValidateRename 14338
IDesignerNotify interface 3195
about IDesignerNotify interface 3195
IDesignerNotify Members 3195
IDesignerNotify methods 3196
Modified 3196
Notification 3196
IDesignerSelections interface 11172
about IDesignerSelections interface 11172
Add 11173
Count 11175
DesignObjects 11175
Equals 11173
Get 11174
GetCount 11174
GetDesignObject 11174
IDesignerSelections Members 11172
IDesignerSelections methods 11173
IDesignerSelections properties 11175
Items 11175
IDesignGroupChange interface 11088
about IDesignGroupChange interface 11088
FreeEditorGroup 11089
IDesignGroupChange Members 11088
IDesignGroupChange methods 11088
IDesignNotification interface 11176
about IDesignNotification interface 11176
DesignerClosed 11177
DesignerOpened 11177
IDesignNotification Members 11176

IDesignNotification methods 11176
ItemDeleted 11177
ItemInserted 11177
ItemsModified 11178
SelectionChanged 11178
IDesignObject interface 11178
about IDesignObject interface 11178
ClassName 11181
ClassType 11182
ComponentIndex 11182
ComponentName 11182
Equals 11179, 11180
Equals method 11179
GetClassName 11180
GetClassType 11180
GetComponentIndex 11180
GetComponentName 11180
GetIsComponent 11181
GetNamePath 11181
IDesignObject Members 11178
IDesignObject methods 11179
IDesignObject properties 11181
IsComponent 11182
NamePath 11182
IDesignPersistent interface 11183
about IDesignPersistent interface 11183
GetPersistent 11184
IDesignPersistent Members 11183
IDesignPersistent methods 11183
IDesignPersistent properties 11184
Persistent 11184
IDesignWindow interface 11184
about IDesignWindow interface 11184
IDesignWindow Members 11185
IDesignWindow methods 11185
WindowHide 11185
WindowShow 11185
IDispatch interface 22394
about IDispatch interface 22394
GetIDsOfNames 22395
GetTypeInfo 22395
GetTypeInfoCount 22395
IDispatch Members 22394
IDispatch methods 22394
Invoke 22395
IDockManager interface 6238
about IDockManager interface 6238
BeginUpdate 6239
EndUpdate 6240
GetControlBounds 6240
IDockManager Members 6239
IDockManager methods 6239
InsertControl 6240
LoadFromStream 6241
PaintSite 6241
PositionDockRect 6242
RemoveControl 6242
ResetBounds 6243
SaveToStream 6245
SetReplacingControl 6245
IDocumentation interface 25837
about IDocumentation interface 25837
IDocumentationCollection interface 25838
about IDocumentationCollection interface 25838
Add 25840
Documentation 25841
GetDocumentation 25840
IDocumentationCollection Members 25838
IDocumentationCollection methods 25840
IDocumentationCollection properties 25840
Insert 25840
IDocumented interface 25841
about IDocumented interface 25841
Documentation 25843
GetDocumentationCollection 25843
IDocumented Members 25841
IDocumented methods 25842
IDocumented properties 25843

IDOMAttr interface 26270
about IDOMAttr interface 26270
get_name 26272
get_ownerElement 26272
get_specified 26272
get_value 26273
IDOMAttr Members 26271
IDOMAttr methods 26272
IDOMAttr properties 26273
name 26273
ownerElement 26273
set_value 26273
specified 26274
value 26274
IDOMCDATASection interface 26274
about IDOMCDATASection interface 26274
IDOMCharacterData interface 26275
about IDOMCharacterData interface 26275
appendData 26276
data 26278
deleteData 26277
get_data 26277
get_length 26277
IDOMCharacterData Members 26275
IDOMCharacterData methods 26276
IDOMCharacterData properties 26278
insertData 26277
length 26278
replaceData 26277
set_data 26278
substringData 26278
IDOMComment interface 26279
about IDOMComment interface 26279
IDOMDocument interface 26279
about IDOMDocument interface 26279
createAttribute 26281
createAttributeNS 26282
createCDATASection 26282
createComment 26282
createDocumentFragment 26282
createElement 26282
createElementNS 26283
createEntityReference 26283
createProcessingInstruction 26283
createTextNode 26283
doctype 26286
documentElement 26286
domImplementation 26286
get_doctype 26284
get_documentElement 26284
get_domImplementation 26284
getElementById 26284
getElementsByTagName 26285
getElementsByTagNameNS 26285
IDOMDocument Members 26279
IDOMDocument methods 26281
IDOMDocument properties 26285
importNode 26285
set_documentElement 26285
IDOMDocumentFragment interface 26286
about IDOMDocumentFragment interface 26286
IDOMDocumentType interface 26287
about IDOMDocumentType interface 26287
entities 26290
get_entities 26289
get_internalSubset 26289
get_name 26289
get_notations 26289
get_publicId 26289
get_systemId 26290
IDOMDocumentType Members 26287
IDOMDocumentType methods 26288
IDOMDocumentType properties 26290
internalSubset 26290
name 26291
notations 26291
publicId 26291
systemId 26291

IDOMElement interface 26291
about IDOMElement interface 26291
get_tagName 26294
getAttribute 26294
getAttributeNode 26294
getAttributeNodeNS 26294
getAttributeNS 26294
getElementsByTagName 26295
getElementsByTagNameNS 26295
hasAttribute 26295
hasAttributeNS 26295
IDOMElement Members 26292
IDOMElement methods 26293
IDOMElement properties 26297
removeAttribute 26296
removeAttributeNode 26296
removeAttributeNS 26296
setAttribute 26296
setAttributeNode 26296
setAttributeNodeNS 26297
setAttributeNS 26297
tagName 26297
IDOMEntity interface 26298
about IDOMEntity interface 26298
get_notationName 26299
get_publicId 26299
get_systemId 26300
IDOMEntity Members 26298
IDOMEntity methods 26299
IDOMEntity properties 26300
notationName 26300
publicId 26300
systemId 26300
IDOMEntityReference interface 26301
about IDOMEntityReference interface 26301
IDOMHeaderProcessor interface 19921
about IDOMHeaderProcessor interface 19921
CreateHeader 19922
IDOMHeaderProcessor Members 19922
IDOMHeaderProcessor methods 19922
ProcessHeader 19922
IDOMImplementation interface 26301
about IDOMImplementation interface 26301
createDocument 26302
createDocumentType 26302
hasFeature 26302
IDOMImplementation Members 26301
IDOMImplementation methods 26302
IDOMNamedNodeMap interface 26303
about IDOMNamedNodeMap interface 26303
get_item 26304
get_length 26304
getNamedItem 26304
getNamedItemNS 26304
IDOMNamedNodeMap Members 26303
IDOMNamedNodeMap methods 26303
IDOMNamedNodeMap properties 26305
item 26306
length 26306
removeNamedItem 26305
removeNamedItemNS 26305
setNamedItem 26305
setNamedItemNS 26305
IDOMNode interface 26306
about IDOMNode interface 26306
appendChild 26308
attributes 26317
childNodes 26317
cloneNode 26309
firstChild 26317
get_attributes 26309
get_childNodes 26309
get_firstChild 26309
get_lastChild 26310
get_localName 26310
get_namespaceURI 26310
get_nextSibling 26311
get_nodeName 26311

get_nodeType 26311
get_nodeValue 26312
get_ownerDocument 26312
get_parentNode 26312
get_prefix 26312
get_previousSibling 26313
hasChildNodes 26313
IDOMNode Members 26307
IDOMNode methods 26308
IDOMNode properties 26316
insertBefore 26313
lastChild 26317
localName 26318
namespaceURI 26318
nextSibling 26318
nodeName 26318
nodeType 26318
nodeValue 26319
normalize 26314
ownerDocument 26319
parentNode 26319
prefix 26319
previousSibling 26320
removeChild 26315
replaceChild 26315
set_nodeValue 26316
supports 26316
IDOMNodeEx interface 26320
about IDOMNodeEx interface 26320
get_text 26321
get_xml 26322
IDOMNodeEx Members 26320
IDOMNodeEx methods 26321
IDOMNodeEx properties 26323
set_text 26322
text 26323
transformNode 26322, 26323
transformNode method 26322
xml 26323
IDOMNodeList interface 26323
about IDOMNodeList interface 26323
get_item 26324
get_length 26324
IDOMNodeList Members 26324
IDOMNodeList methods 26324
IDOMNodeList properties 26325
item 26325
length 26325
IDOMNodeSelect interface 26325
about IDOMNodeSelect interface 26325
IDOMNodeSelect Members 26326
IDOMNodeSelect methods 26326
selectNode 26326
selectNodes 26326
IDOMNotation interface 26327
about IDOMNotation interface 26327
get_publicId 26328
get_systemId 26328
IDOMNotation Members 26327
IDOMNotation methods 26328
IDOMNotation properties 26329
publicId 26329
systemId 26329
IDOMParseError interface 26329
about IDOMParseError interface 26329
errorCode 26332
filePos 26332
get_errorCode 26330
get_filepos 26331
get_line 26331
get_linepos 26331
get_reason 26331
get_srcText 26331
get_url 26332
IDOMParseError Members 26330
IDOMParseError methods 26330
IDOMParseError properties 26332
line 26333

linePos 26333
reason 26333
srcText 26333
url 26333
IDOMParseOptions interface 26334
about IDOMParseOptions interface 26334
async 26337
get_async 26335
get_preserveWhiteSpace 26335
get_resolveExternals 26335
get_validate 26335
IDOMParseOptions Members 26334
IDOMParseOptions methods 26334
IDOMParseOptions properties 26336
preserveWhiteSpace 26337
resolveExternals 26337
set_async 26336
set_preserveWhiteSpace 26336
set_resolveExternals 26336
set_validate 26336
validate 26337
IDOMPersist interface 26338
about IDOMPersist interface 26338
asyncLoadState 26339
get_xml 26339
IDOMPersist Members 26338
IDOMPersist methods 26338
IDOMPersist properties 26341
load 26339
loadFromStream 26339, 26340
loadFromStream method 26339
loadxml 26340
save 26340
saveToStream 26341
saveToStream method 26340
set_OnAsyncLoad 26341
xml 26341
IDOMProcessingInstruction interface 26342
about IDOMProcessingInstruction interface 26342
data 26344
get_data 26343
get_target 26343
IDOMProcessingInstruction Members 26342
IDOMProcessingInstruction methods 26343
IDOMProcessingInstruction properties 26344
set_data 26344
target 26344
IDOMText interface 26345
about IDOMText interface 26345
IDOMText Members 26345
IDOMText methods 26346
splitText 26346
IDOMXMLProlog interface 26347
about IDOMXMLProlog interface 26347
Encoding 26349
get_Encoding 26348
get_Standalone 26348
get_Version 26348
IDOMXMLProlog Members 26347
IDOMXMLProlog methods 26347
IDOMXMLProlog properties 26349
set_Encoding 26348
set_Standalone 26348
set_Version 26349
Standalone 26349
Version 26349
IDOMXSLProcessor interface 26350
about IDOMXSLProcessor interface 26350
Get_input 26351
Get_output 26351
Get_stylesheet 26351
IDOMXSLProcessor Members 26350
IDOMXSLProcessor methods 26351
IDOMXSLProcessor properties 26353
input 26353
output 26353
reset 26352
Set_input 26352

Set_output 26352
setParameter 26352
stylesheet 26353
transform 26352
IDSBase interface 11673
about IDSBase interface 11673
AcceptChanges 11675
AddField 11675
AddOptParameter 11675
AppendData 11675
Clone 11676
Create 11676
CreateIndex 11676
DropOptParameter 11676
FldCmp 11677
GetDelta 11677
GetEmbeddedDS 11677
GetErrorString 11677
GetFieldDescs 11677
GetIndexDescs 11678
GetOptParameter 11678
GetProp 11678
GetProps 11678
IDSBase Members 11673
IDSBase methods 11674
MakeEmbedded 11679
PutBlank 11679
Reconcile 11679
Reconcile_MD 11679
Refresh 11680
RefreshRecords 11680
ReleaseBlobs 11680
RemoveIndex 11680
Reserved1 11681
Reserved2 11681
Reserved3 11681
Reset 11681
RollBack 11681
SetFieldCalculation 11682
SetProp 11682
StreamDS 11682
IDSCursor interface 11682
about IDSCursor interface 11682
AddAggregate 11685
AddFilter 11685
AddFilterCallBack 11685
CloneCursor 11686
CompareBookMarks 11686
DeleteRecord 11686
DropAggregate 11686
DropFilter 11687
DropRange 11687
ExtractKey 11687
GetAggregateDesc 11687
GetAggregateValue 11687
GetBlob 11688
GetBlobLen 11688
GetCurrentBookMark 11688
GetCurrentRecord 11688
GetCursorProps 11689
GetField 11689
GetFieldDescs 11689
GetIndexDescs 11689
GetProp 11690
GetRecordAttribute 11690
GetRecordBlock 11690
GetRecordCount 11690
GetRecordForKey 11690
GetRecordNumber 11691
GetRowRequestPacket 11691
GetSequenceNumber 11691
GetSubGroupState 11691
IDSCursor Members 11683
IDSCursor methods 11684
InitCursor 11692
InitRecord 11692
InsertRecord 11692
LinkCursors 11692

LocateWithFilter 11693
ModifyRecord 11693
MoveRelative 11693
MoveToBOF 11693
MoveToBookMark 11693
MoveToEOF 11694
MoveToKey 11694
MoveToNextSubGroup 11694
MoveToSeqNo 11694
PutBlob 11695
PutField 11695
RefreshRecord 11695
ResyncWithMaster 11695
RevertRecord 11695
SetNotifyCallBack 11696
SetProp 11696
SetRange 11696
SortOnFields 11696
UndoLastChange 11697
UseIndexOrder 11697
VerifyField 11697
IDSWriter interface 11697
about IDSWriter interface 11697
AddAttribute 11698
AddColumnDesc 11699
EndOfNestedRows 11699
GetColumnCount 11699
GetColumnDescs 11699
GetDataPacket 11700
GetErrorString 11700
GetSize 11700
GetXMLMode 11700
IDSWriter Members 11698
IDSWriter methods 11698
Init 11700
Init_Sequential 11701
PutField 11701
Reset 11701
SetXMLMode 11701
idxDeleted enumeration member 17840
idxFiltered enumeration member 17840
idxNormal enumeration member 17840
idxSparsed enumeration member 17840
idxSubTotals enumeration member 17840
IEchoActionFieldValue interface 24572
about IEchoActionFieldValue interface 24572
EchoActionFieldValue 24574
GetEchoActionFieldValue 24573
IEchoActionFieldValue Members 24572
IEchoActionFieldValue methods 24573
IEchoActionFieldValue properties 24573
SetEchoActionFieldValue 24573
IEchoAdapterFieldValues interface 24574
about IEchoAdapterFieldValues interface 24574
EchoActionFieldValues 24575
GetEchoActionFieldValues 24575
IEchoAdapterFieldValues Members 24574
IEchoAdapterFieldValues methods 24574
IEchoAdapterFieldValues properties 24575
SetEchoActionFieldValues 24575
IEditHandler interface 11186
about IEditHandler interface 11186
EditAction 11186
GetEditState 11187
IEditHandler Members 11186
IEditHandler methods 11186
IEndUserStub interface 1470
about IEndUserStub interface 1470
DisplayName 1472
Get_DisplayName 1471
Get_LoggedIn 1471
Get_LoginFormAction 1471
Get_LogoutAction 1472
IEndUserStub Members 1470
IEndUserStub methods 1471
IEndUserStub properties 1472
LoggedIn 1472
LoginFormAction 1472

LogoutAction 1473
IEnumActionFieldValueRecords interface 957
about IEnumActionFieldValueRecords interface 957
EndIterator 959
Eof 962
FindPopulatedRecord 959
GetEOF 960
GetFirstRecord 960
GetInIterator 960
GetNextRecord 961
GetRecord 961
GetRecordCount 962
IEnumActionFieldValueRecords Members 958
IEnumActionFieldValueRecords methods 959
IEnumActionFieldValueRecords properties 962
InIterator 962
RecordCount 963
IEnumNames interface 17271
about IEnumNames interface 17271
Clone 17272
IEnumNames Members 17271
IEnumNames methods 17272
Next 17272
Reset 17272
Skip 17272
IEventInfo interface 11187
about IEventInfo interface 11187
GetMethodKind 11188
GetParamCount 11188
GetParamFlags 11188
GetParamName 11188
GetParamType 11189
GetResultType 11189
IEventInfo Members 11187
IEventInfo methods 11187
IExecuteValue interface 24575
about IExecuteValue interface 24575
ExecuteValue 24576
IExecuteValue Members 24576
IExecuteValue methods 24576
IExtendedHelpViewer interface 15017
about IExtendedHelpViewer interface 15017
DisplayHelpByContext 15019
DisplayTopic 15019
IExtendedHelpViewer Members 15018
IExtendedHelpViewer methods 15018
UnderstandsContext 15019
UnderstandsTopic 15020
IFault interface 25843
about IFault interface 25843
IFaults interface 25844
about IFaults interface 25844
Add 25846
Fault 25847
GetFault 25846
IFaults Members 25844
IFaults methods 25846
IFaults properties 25846
Insert 25846
ifData enumeration member 5453
ifDispatch enumeration member 23373
ifDispInterface enumeration member 23373
ifExactString enumeration member 5453
ifHasGuid enumeration member 23373
IFieldValuesAdapter interface 19683
about IFieldValuesAdapter interface 19683
GetField 19684
GetValueIndex 19684
IFieldValuesAdapter Members 19684
IFieldValuesAdapter methods 19684
IFindIncludeFile interface 19684
about IFindIncludeFile interface 19684
FindIncludeFile 19685
IFindIncludeFile Members 19685
IFindIncludeFile methods 19685
ifNearest enumeration member 5453
IFontAccess interface 1706
about IFontAccess interface 1706

GetOleFont 1707
IFontAccess Members 1707
IFontAccess methods 1707
SetOleFont 1707
IFormatColumn interface 17061
about IFormatColumn interface 17061
FormatColumnData 17062
FormatColumnHeading 17062
IFormatColumn Members 17061
IFormatColumn methods 17062
ifPartialString enumeration member 5453
iftCheckBox enumeration member 25319
iftDefault enumeration member 25319
iftFile enumeration member 25319
iftPasswordInput enumeration member 25319
iftRadio enumeration member 25319
iftSelect enumeration member 25319
iftSelectMultiple enumeration member 25319
iftTextArea enumeration member 25319
iftTextInput enumeration member 25319
IFullyQualifyInputNames interface 24577
about IFullyQualifyInputNames interface 24577
FullyQualifyInputNames 24577
IFullyQualifyInputNames Members 24577
IFullyQualifyInputNames methods 24577
IGetAdapterActionRedirectOptions interface 963
about IGetAdapterActionRedirectOptions interface 963
FailureOptions 965
GetFailureOptions 964
GetSuccessOptions 964
IGetAdapterActionRedirectOptions Members 964
IGetAdapterActionRedirectOptions methods 964
IGetAdapterActionRedirectOptions properties 965
SuccessOptions 965
IGetAdapterActions interface 19685
about IGetAdapterActions interface 19685
GetAdapterActions 19686
IGetAdapterActions Members 19686
IGetAdapterActions methods 19686
IGetAdapterDisplayCharacteristics interface 19686
about IGetAdapterDisplayCharacteristics interface 19686
GetAdapterDisplayCharacteristics 19687
IGetAdapterDisplayCharacteristics Members 19687
IGetAdapterDisplayCharacteristics methods 19687
IGetAdapterErrors interface 19687
about IGetAdapterErrors interface 19687
GetAdapterErrors 19688
IGetAdapterErrors Members 19688
IGetAdapterErrors methods 19688
IGetAdapterErrorsList interface 24578
about IGetAdapterErrorsList interface 24578
GetAdapterErrorsList 24578
IGetAdapterErrorsList Members 24578
IGetAdapterErrorsList methods 24578
IGetAdapterFields interface 19688
about IGetAdapterFields interface 19688
GetAdapterFields 19689
IGetAdapterFields Members 19689
IGetAdapterFields methods 19689
IGetAdapterFormLayout interface 25303
about IGetAdapterFormLayout interface 25303
GetAdapterFormLayout 25304
IGetAdapterFormLayout Members 25304
IGetAdapterFormLayout methods 25304
IGetAdapterHiddenFields interface 19689
about IGetAdapterHiddenFields interface 19689
GetAdapterHiddenFields 19690
GetAdapterHiddenRecordFields 19690
IGetAdapterHiddenFields Members 19689
IGetAdapterHiddenFields methods 19690
IGetAdapterImage interface 19690
about IGetAdapterImage interface 19690
GetAdapterImage 19691
IGetAdapterImage Members 19691
IGetAdapterImage methods 19691
IGetAdapterItemRequestParams interface 19691
about IGetAdapterItemRequestParams interface 19691
GetAdapterItemRequestParams 19692

IGetAdapterItemRequestParams Members 19692
IGetAdapterItemRequestParams methods 19692
IGetAdapterModeNames interface 24579
about IGetAdapterModeNames interface 24579
GetAdapterModeNames 24579
IGetAdapterModeNames Members 24579
IGetAdapterModeNames methods 24579
IGetAdapterModeOfDisplay interface 25304
about IGetAdapterModeOfDisplay interface 25304
GetAdapterModeOfAdapter 25305
GetAdapterModeOfDisplay 25305
IGetAdapterModeOfDisplay Members 25305
IGetAdapterModeOfDisplay methods 25305
IGetAdapterOfDisplay interface 25305
about IGetAdapterOfDisplay interface 25305
GetAdapter 25306
IGetAdapterOfDisplay Members 25306
IGetAdapterOfDisplay methods 25306
IGetAdapterRequestDefaultResponse interface 19692
about IGetAdapterRequestDefaultResponse interface 19692
FailurePage 19694
GetFailurePage 19693
GetSuccessPage 19693
IGetAdapterRequestDefaultResponse Members 19693
IGetAdapterRequestDefaultResponse methods 19693
IGetAdapterRequestDefaultResponse properties 19694
SuccessPage 19694
IGetAdapterValuesList interface 19694
about IGetAdapterValuesList interface 19694
GetAdapterValuesList 19695
IGetAdapterValuesList Members 19695
IGetAdapterValuesList methods 19695
IGetAdapterVariableName interface 25306
about IGetAdapterVariableName interface 25306
AdapterVariableName 25308
GetAdapterVariableName 25307
IGetAdapterVariableName Members 25307
IGetAdapterVariableName methods 25307
IGetAdapterVariableName properties 25307
IGetAppDispatcher interface 15219
about IGetAppDispatcher interface 15219
GetAppDispatcher 15220
IGetAppDispatcher Members 15219
IGetAppDispatcher methods 15220
IGetContextProperties interface 17273
about IGetContextProperties interface 17273
Count 17273
EnumNames 17274
GetProperty 17274
IGetContextProperties Members 17273
IGetContextProperties methods 17273
IGetControlRequirements interface 25308
about IGetControlRequirements interface 25308
GetControlRequirements 25309
IGetControlRequirements Members 25308
IGetControlRequirements methods 25308
IGetDataSetKeyName interface 8104
about IGetDataSetKeyName interface 8104
GetKeyName 8105
IGetDataSetKeyName Members 8104
IGetDataSetKeyName methods 8105
IGetDataSetKeyName properties 8105
KeyName 8105
IGetDefaultAction interface 19695
about IGetDefaultAction interface 19695
DefaultAction 19696
GetDefaultAction 19696
IGetDefaultAction Members 19695
IGetDefaultAction methods 19696
IGetDefaultAction properties 19696
IGetDesigntimeWarnings interface 19697
about IGetDesigntimeWarnings interface 19697
GetDesigntimeWarnings 19697
IGetDesigntimeWarnings Members 19697
IGetDesigntimeWarnings methods 19697
IGetFieldValuesAdapter interface 19698
about IGetFieldValuesAdapter interface 19698
GetFieldValuesAdapter 19698

IGetFieldValuesAdapter Members 19698
IGetFieldValuesAdapter methods 19698
IGetFormRequirements interface 25309
about IGetFormRequirements interface 25309
GetFormRequirements 25310
IGetFormRequirements Members 25309
IGetFormRequirements methods 25309
IGetHTMLStyle interface 19699
about IGetHTMLStyle interface 19699
GetDisplayStyle 19700
GetInputStyle 19700
GetViewMode 19701
IGetHTMLStyle Members 19699
IGetHTMLStyle methods 19699
IGetLocateFileService interface 15318
about IGetLocateFileService interface 15318
GetLocateFileService 15319
IGetLocateFileService Members 15318
IGetLocateFileService methods 15319
IGetProducerComponent interface 19702
about IGetProducerComponent interface 19702
GetProducerComponent 19702
IGetProducerComponent Members 19702
IGetProducerComponent methods 19702
IGetProducerTemplate interface 15319
about IGetProducerTemplate interface 15319
GetProducerTemplateFileName 15320
GetProducerTemplateStream 15320
IGetProducerTemplate Members 15319
IGetProducerTemplate methods 15320
IGetRecordKeys interface 24580
about IGetRecordKeys interface 24580
GetRecordKeys 24580
IGetRecordKeys Members 24580
IGetRecordKeys methods 24580
IGetScriptObject interface 19703
about IGetScriptObject interface 19703
GetScriptObject 19703
IGetScriptObject Members 19703
IGetScriptObject methods 19703
IGetScriptObjectClassKey interface 19704
about IGetScriptObjectClassKey interface 19704
GetScriptObjectClassKey 19704
IGetScriptObjectClassKey Members 19704
IGetScriptObjectClassKey methods 19704
IGetVariablesContainerOfDisplay interface 25310
about IGetVariablesContainerOfDisplay interface 25310
GetVariablesContainer 25311
IGetVariablesContainerOfDisplay Members 25310
IGetVariablesContainerOfDisplay methods 25310
IGetWebAppComponents interface 19705
about IGetWebAppComponents interface 19705
GetAdapterDispatcher 19706
GetLocateFileService 19706
GetPageDispatcher 19706
GetWebApplicationInfo 19706
GetWebEndUser 19706
GetWebUserList 19707
IGetWebAppComponents Members 19705
IGetWebAppComponents methods 19705
IGetWebAppComponents properties 19707
WebApplicationInfo 19707
WebEndUser 19707
IGetWebAppServices interface 15220
about IGetWebAppServices interface 15220
GetWebAppServices 15221
IGetWebAppServices Members 15220
IGetWebAppServices methods 15221
IGetWebComponentList interface 24771
about IGetWebComponentList interface 24771
GetComponentList 24772
GetDefaultComponentList 24772
IGetWebComponentList Members 24771
IGetWebComponentList methods 24771
IGetWebComponentNodeName interface 24772
about IGetWebComponentNodeName interface 24772
GetWebComponentNodeName 24773
IGetWebComponentNodeName Members 24772

IGetWebComponentNodeName methods 24773
IGetWebRequestHandler interface 15221
about IGetWebRequestHandler interface 15221
GetWebRequestHandler 15222
IGetWebRequestHandler Members 15221
IGetWebRequestHandler methods 15221
IGetXMLStream interface 24773
about IGetXMLStream interface 24773
GetXMLStream 24774
IGetXMLStream Members 24773
IGetXMLStream methods 24774
IHeaderInfo interface 25989
about IHeaderInfo interface 25989
EncodingStyle 25993
GetEncodingStyle 25990
GetName 25990
GetNamespace 25991
GetPart 25991
GetRequired 25991
GetUse 25991
IHeaderInfo Members 25989
IHeaderInfo methods 25990
IHeaderInfo properties 25993
Name 25993
Namespace 25993
Part 25994
Required 25994
SetEncodingStyle 25991
SetName 25992
SetNamespace 25992
SetPart 25992
SetRequired 25992
SetUse 25993
Use 25994
IHeadersSetter interface 15609
about IHeadersSetter interface 15609
IHeadersSetter Members 15609
IHeadersSetter methods 15609
SetHeadersInOut 15610
IHelpManager interface 15020
about IHelpManager interface 15020
GetHandle 15021
GetHelpFile 15021
IHelpManager Members 15021
IHelpManager methods 15021
Release 15021
IHelpSelector interface 15022
about IHelpSelector interface 15022
IHelpSelector Members 15023
IHelpSelector methods 15023
SelectKeyword 15023
TableOfContents 15024
IHelpSelector2 interface 15024
about IHelpSelector2 interface 15024
IHelpSelector2 Members 15024
IHelpSelector2 methods 15025
SelectContext 15025
IHelpSystem interface 15025
about IHelpSystem interface 15025
AssignHelpSelector 15026
Hook 15026
IHelpSystem Members 15025
IHelpSystem methods 15026
ShowContextHelp 15027
ShowHelp 15027
ShowTableOfContents 15027
ShowTopicHelp 15028
IHelpSystem2 interface 15028
about IHelpSystem2 interface 15028
IHelpSystem2 Members 15028
IHelpSystem2 methods 15029
UnderstandsKeyword 15029
IHelpSystemFlags interface 15029
about IHelpSystemFlags interface 15029
GetUseDefaultTopic 15030
IHelpSystemFlags Members 15029
IHelpSystemFlags methods 15030
SetUseDefaultTopic 15030

IHostForm interface 11189
about IHostForm interface 11189
BringToFront 11191
Caption 11195
CheckPosChanged 11191
DeinitializeDesigner 11191
GetCanPrint 11191
GetCaption 11191
GetDesignerState 11192
GetFont 11192
GetForm 11192
GetFormImage 11192
GetScrollPos 11193
GetVisible 11193
GetWindowState 11193
HideWindow 11193
IHostForm Members 11189
IHostForm methods 11190
IHostForm properties 11195
IsMenuKey 11193
SetCaption 11194
SetDesigner 11194
SetDesigning 11194
SetFormDefaults 11194
Show 11195
ShowWindow 11195
Unmodify 11195
Visible 11196
IHTMLField interface 17062
about IHTMLField interface 17062
GetHTMLControlName 17063
HTMLControlName 17064
IHTMLField Members 17063
IHTMLField methods 17063
IHTMLField properties 17063
IHTMLForm interface 17064
about IHTMLForm interface 17064
GetHTMLFormName 17065
GetHTMLFormVarName 17065
HTMLFormName 17066
HTMLFormVarName 17066
IHTMLForm Members 17064
IHTMLForm methods 17065
IHTMLForm properties 17066
IHTTPReqResp interface 19983
about IHTTPReqResp interface 19983
GetHTTPReqResp 19984
IHTTPReqResp Members 19983
IHTTPReqResp methods 19983
IHTTPSoapDispatch interface 19946
about IHTTPSoapDispatch interface 19946
DispatchSOAP 19947
IHTTPSoapDispatch Members 19946
IHTTPSoapDispatch methods 19947
IIDEFileManager interface 19708
about IIDEFileManager interface 19708
IIDEFileManager Members 19708
IIDEFileManager methods 19708
SearchFile 19708
IIdentifyAdapter interface 24581
about IIdentifyAdapter interface 24581
IImageProducer interface 19709
about IImageProducer interface 19709
GetAdapterImage 19710
IImageProducer Members 19709
IImageProducer methods 19710
IImageProducerStub interface 1473
about IImageProducerStub interface 1473
GetDisplayStyle 1474
GetEventImage 1474
GetImage 1474
IImageProducerStub Members 1473
IImageProducerStub methods 1473
IImageRequest interface 965
about IImageRequest interface 965
GetImageName 966
IImageRequest Members 966
IImageRequest methods 966

IImageRequest properties 967
ImageName 967
IImageResponse interface 967
about IImageResponse interface 967
IImplementation interface 11196
about IImplementation interface 11196
GetInstance 11197
IImplementation Members 11196
IImplementation methods 11196
IImport interface 25847
about IImport interface 25847
GetLocation 25849
GetNamespace 25849
IImport Members 25847
IImport methods 25849
IImport properties 25850
Location 25850
Namespace 25850
SetLocation 25849
SetNamespace 25849
IImports interface 25850
about IImports interface 25850
Add 25852
GetImport 25853
IImports Members 25851
IImports methods 25852
IImports properties 25853
Import 25853
Insert 25853
IInetXWebDataSet interface 17066
about IInetXWebDataSet interface 17066
AddOnDataChange 17068
FetchParams 17068
FieldCount 17068
Fields 17069
IInetXWebDataSet Members 17067
IInetXWebDataSet methods 17067
ParamCount 17069
Params 17069
RemoveOnDataChange 17070
IInterface interface 22396
_AddRef 22397
_Release 22397
about IInterface interface 22396
IInterface Members 22396
IInterface methods 22397
QueryInterface 22398
IInterfaceComponentReference interface 3197
about IInterfaceComponentReference interface 3197
GetComponent 3197
IInterfaceComponentReference Members 3197
IInterfaceComponentReference methods 3197
IInterfaceList interface 3198
about IInterfaceList interface 3198
Add 3199
Capacity 3205
Clear 3200
Count 3206
Delete 3200
Exchange 3201
First 3201
Get 3201
GetCapacity 3202
GetCount 3202
IInterfaceList Members 3198
IInterfaceList methods 3199
IInterfaceList properties 3205
IndexOf 3202
Insert 3202
Items 3206
Last 3203
Lock 3203
Put 3204
Remove 3204
SetCapacity 3204
SetCount 3205
Unlock 3205
IInternetEnvOptions interface 19710

about IInternetEnvOptions interface 19710
DisableDebugger 19712
GetDisableDebugging 19711
GetHTMLExt 19711
GetSampleImageFile 19711
GetScriptTimeoutSecs 19712
HTMLExt 19712
IInternetEnvOptions Members 19710
IInternetEnvOptions methods 19711
IInternetEnvOptions properties 19712
SampleImageFile 19712
ScriptTimeoutSecs 19713
IInternetExecuteScript interface 19713
about IInternetExecuteScript interface 19713
CanExecuteScript 19714
ExecuteScript 19714
IInternetExecuteScript Members 19713
IInternetExecuteScript methods 19713
IInternetFileTypes interface 19714
about IInternetFileTypes interface 19714
FindFileExt 19715
IInternetFileTypes Members 19714
IInternetFileTypes methods 19715
IInternetScriptDebugger interface 19715
about IInternetScriptDebugger interface 19715
CanDebugScript 19716
DebugScript 19716
IInternetScriptDebugger Members 19715
IInternetScriptDebugger methods 19716
IInvokable interface 22398
about IInvokable interface 22398
IInvokerConverter interface 20033
about IInvokerConverter interface 20033
GetInvoker 20034
IInvokerConverter Members 20033
IInvokerConverter methods 20033
IIsAdapterActionList interface 19716
about IIsAdapterActionList interface 19716
GetIsAdapterActionList 19717
IIsAdapterActionList Members 19717
IIsAdapterActionList methods 19717
IIsAdapterActionList properties 19717
IsAdapterActionList 19717
IIterateIntfSupport interface 19718
about IIterateIntfSupport interface 19718
GetCurrentIntf 19718
IIterateIntfSupport Members 19718
IIterateIntfSupport methods 19718
IIterateObjectSupport interface 19719
about IIterateObjectSupport interface 19719
GetCurrentObject 19719
IIterateObjectSupport Members 19719
IIterateObjectSupport methods 19719
IIteratorIndex interface 19720
about IIteratorIndex interface 19720
GetIteratorIndex 19721
IIteratorIndex Members 19720
IIteratorIndex methods 19720
IIteratorIndex properties 19721
InIterator 19721
IteratorIndex 19721
IIteratorSupport interface 19722
about IIteratorSupport interface 19722
EndIterator 19723
IIteratorSupport Members 19723
IIteratorSupport methods 19723
NextIteration 19723
StartIterator 19724
IKeyParams interface 8105
about IKeyParams interface 8105
Count 8106
GetName 8107
GetValue 8107
IKeyParams Members 8106
IKeyParams methods 8106
IKeyParams properties 8107
Names 8108
Values 8108

ILayoutWebContent interface 24774
about ILayoutWebContent interface 24774
ILayoutWebContent Members 24775
ILayoutWebContent methods 24775
LayoutButton 24775
LayoutField 24775
LayoutLabelAndField 24775
LayoutTable 24776
ilBrowse enumeration member 1300
ilChaos enumeration member 1300
ilCursorStability enumeration member 1300
ilIsolated enumeration member 1300
ILocateFileService interface 15320
about ILocateFileService interface 15320
GetTemplateStream 15321
ILocateFileService Members 15321
ILocateFileService methods 15321
ilReadCommitted enumeration member 1300
ilReadUncommitted enumeration member 1300
ilRepeatableRead enumeration member 1300
ilSerializable enumeration member 1300
ilUnspecified enumeration member 1300
imAlpha enumeration member 6285
IMarshalInBuffer interface 18192
about IMarshalInBuffer interface 18192
Buffer 18193
GetAny 18194
GetChar 18194
GetDouble 18194
GetFloat 18194
GetLong 18194
GetObject 18195
GetShort 18195
GetText 18195
GetUnsignedChar 18195
GetUnsignedLong 18196
GetUnsignedShort 18196
GetWidechar 18196
GetWideText 18196
IMarshalInBuffer Members 18192
IMarshalInBuffer methods 18193
IMarshalOutBuffer interface 18196
about IMarshalOutBuffer interface 18196
Buffer 18198
IMarshalOutBuffer Members 18197
IMarshalOutBuffer methods 18197
PutAny 18198
PutChar 18198
PutDouble 18198
PutFloat 18198
PutLong 18199
PutObject 18199
PutShort 18199
PutText 18199
PutUnsignedChar 18200
PutUnsignedLong 18200
PutUnsignedShort 18200
PutWidechar 18200
PutWideText 18200
imChinese enumeration member 6285
imClose enumeration member 6285
imDisable enumeration member 6285
imDontCare enumeration member 6285
IMessage interface 25853
about IMessage interface 25853
GetName 25855
GetParts 25856
IMessage Members 25854
IMessage methods 25855
IMessage properties 25856
Name 25856
Parts 25856
SetName 25856
IMessages interface 25857
about IMessages interface 25857
Add 25859
GetMessage 25859
IMessages Members 25857

IMessages methods 25859
IMessages properties 25859
Insert 25859
Message 25860
IMethodProperty interface 11197
about IMethodProperty interface 11197
ImgList 15357
ImgList namespace 15357
Classes 15358
Structs, Records, Enums 15394
Types 15397
ImgList.TChangeLink 15358
ImgList.TChangeLink.Change 15360
ImgList.TChangeLink.Destroy 15359
ImgList.TChangeLink.OnChange 15361
ImgList.TChangeLink.Sender 15360
ImgList.TCustomImageList 15361
ImgList.TCustomImageList.Add 15368
ImgList.TCustomImageList.AddIcon 15368
ImgList.TCustomImageList.AddImage 15368
ImgList.TCustomImageList.AddImages 15369
ImgList.TCustomImageList.AddMasked 15369
ImgList.TCustomImageList.AllocBy 15387
ImgList.TCustomImageList.Assign 15370
ImgList.TCustomImageList.AssignTo 15370
ImgList.TCustomImageList.BkColor 15387
ImgList.TCustomImageList.BlendColor 15387
ImgList.TCustomImageList.Change 15371
ImgList.TCustomImageList.Clear 15371
ImgList.TCustomImageList.Count 15388
ImgList.TCustomImageList.Create 15365
ImgList.TCustomImageList.CreateSize 15366
ImgList.TCustomImageList.DefineProperties 15371
ImgList.TCustomImageList.Delete 15372
ImgList.TCustomImageList.Destroy 15366
ImgList.TCustomImageList.DoDraw 15372
ImgList.TCustomImageList.Draw 15373
ImgList.TCustomImageList.DrawingStyle 15390
ImgList.TCustomImageList.DrawOverlay 15375
ImgList.TCustomImageList.FileLoad 15375
ImgList.TCustomImageList.GetBitmap 15376
ImgList.TCustomImageList.GetHotSpot 15376
ImgList.TCustomImageList.GetIcon 15377
ImgList.TCustomImageList.GetImageBitmap 15378
ImgList.TCustomImageList.GetImages 15378
ImgList.TCustomImageList.GetInstRes 15379
ImgList.TCustomImageList.GetMaskBitmap 15379
ImgList.TCustomImageList.GetResource 15380
ImgList.TCustomImageList.Handle 15390
ImgList.TCustomImageList.HandleAllocated 15380
ImgList.TCustomImageList.HandleNeeded 15381
ImgList.TCustomImageList.Height 15391
ImgList.TCustomImageList.ImageType 15391
ImgList.TCustomImageList.Initialize 15381
ImgList.TCustomImageList.Insert 15381
ImgList.TCustomImageList.InsertIcon 15382
ImgList.TCustomImageList.InsertMasked 15382
ImgList.TCustomImageList.Masked 15392
ImgList.TCustomImageList.Move 15382
ImgList.TCustomImageList.OnChange 15393
ImgList.TCustomImageList.Overlay 15383
ImgList.TCustomImageList.ReadData 15383
ImgList.TCustomImageList.RegisterChanges 15383
ImgList.TCustomImageList.Replace 15384
ImgList.TCustomImageList.ReplaceIcon 15384
ImgList.TCustomImageList.ReplaceMasked 15384
ImgList.TCustomImageList.ResInstLoad 15385
ImgList.TCustomImageList.ResourceLoad 15385
ImgList.TCustomImageList.ShareImages 15392
ImgList.TCustomImageList.UnRegisterChanges 15386
ImgList.TCustomImageList.Width 15392
ImgList.TCustomImageList.WriteData 15386
ImgList.TDrawingStyle 15394
ImgList.TDrawingStyle enumeration 15394
ImgList.TImageIndex 15397
ImgList.TImageIndex type 15397
ImgList.TImageType 15395
ImgList.TImageType enumeration 15395

ImgList.TLoadResource 15395
ImgList.TLoadResource enumeration 15395
ImgList.TLoadResources 15397
ImgList.TLoadResources type 15397
ImgList.TOverlay 15397
ImgList.TOverlay type 15397
ImgList.TResType 15396
ImgList.TResType enumeration 15396
imHanguel enumeration member 6285
imHira enumeration member 6285
IMimeAttachmentHandler interface 19853
about IMimeAttachmentHandler interface 19853
AddSoapHeader 19854
CreateMimeStream 19855
FinalizeStream 19856
GetMIMEBoundary 19856
GetMIMEStream 19856
GetOnGetAttachmentEvent 19857
GetOnSendAttachmentEvent 19857
IMimeAttachmentHandler events 19860
IMimeAttachmentHandler Members 19853
IMimeAttachmentHandler methods 19854
IMimeAttachmentHandler properties 19859
MIMEBoundary 19859
OnGetAttachment 19860
OnSendAttachment 19860
ProcessMultiPartForm 19857
SetMIMEBoundary 19858
SetOnGetAttachmentEvent 19858
SetOnSendAttachmentEvent 19859
imKata enumeration member 6285
imOpen enumeration member 6285
IMouse 15399
IMouse namespace 15399
Classes 15401
Constants 15428
Functions 15423
Structs, Records, Enums 15424
Types 15426
IMouse.crPanAll 15428
IMouse.crPanAll constant 15428
IMouse.crPanDown 15428
IMouse.crPanDown constant 15428
IMouse.crPanDownLeft 15429
IMouse.crPanDownLeft constant 15429
IMouse.crPanDownRight 15429
IMouse.crPanDownRight constant 15429
IMouse.crPanLeft 15429
IMouse.crPanLeft constant 15429
IMouse.crPanLeftRight 15430
IMouse.crPanLeftRight constant 15430
IMouse.crPanRight 15430
IMouse.crPanRight constant 15430
IMouse.crPanUp 15430
IMouse.crPanUp constant 15430
IMouse.crPanUpDown 15431
IMouse.crPanUpDown constant 15431
IMouse.crPanUpLeft 15431
IMouse.crPanUpLeft constant 15431
IMouse.crPanUpRight 15431
IMouse.crPanUpRight constant 15431
IMouse.StartPan 15423
IMouse.StartPan function 15423
IMouse.TPanDirection 15424
IMouse.TPanDirection enumeration 15424
IMouse.TPanDirections 15426
IMouse.TPanDirections type 15426
IMouse.TPanningWindow 15401
IMouse.TPanningWindow.Create 15412
IMouse.TPanningWindow.CreateParams 15413
IMouse.TPanningWindow.Destroy 15412
IMouse.TPanningWindow.DoUpdate 15414
IMouse.TPanningWindow.GetIsPanning 15414
IMouse.TPanningWindow.GetPanOptions 15414
IMouse.TPanningWindow.IsPanning 15421
IMouse.TPanningWindow.MouseMove 15414
IMouse.TPanningWindow.OnUpdate 15422
IMouse.TPanningWindow.Paint 15415

IMouse.TPanningWindow.PanControl 15421
IMouse.TPanningWindow.PanInterval 15421
IMouse.TPanningWindow.PanOptions 15422
IMouse.TPanningWindow.SetPanCursor 15415
IMouse.TPanningWindow.StartPanning 15416
IMouse.TPanningWindow.StopPanning 15416
IMouse.TPanningWindow.UpdateScroll 15416
IMouse.TPanningWindow.WndProc 15416
IMouse.TPanOption 15424
IMouse.TPanOption enumeration 15424
IMouse.TPanOptions 15426
IMouse.TPanOptions type 15426
IMouse.TUpdateEvent 15426
IMouse.TUpdateEvent type 15426
imSAlpha enumeration member 6285
imSHanguel enumeration member 6285
imSKata enumeration member 6285
iMultiInstance enumeration member 6484
IMultiModuleSupport interface 15222
about IMultiModuleSupport interface 15222
FinishContext 15223
IMultiModuleSupport Members 15222
IMultiModuleSupport methods 15222
InitContext 15223
InitModule 15223
incFile enumeration member 25545
incPage enumeration member 25545
IndexColumnsColumns class 2154
about IndexColumnsColumns class 2154
CatalogName 2156
ColumnName 2156
IndexColumnsColumns fields 2156
IndexColumnsColumns Members 2155
IndexName 2156
IsAscending 2157
Ordinal 2157
SchemaName 2157
TableName 2157
IndexesColumns class 2157
about IndexesColumns class 2157
CatalogName 2159
ConstraintName 2159
IndexesColumns fields 2159
IndexesColumns Members 2158
IndexName 2159
IsAscending 2159
IsPrimary 2160
IsUnique 2160
SchemaName 2160
TableName 2160
IniFiles 15433
IniFiles namespace 15433
Classes 15434
Structs, Records, Enums 15475
Types 15476
IniFiles.EIniFileException 15434
IniFiles.PHashItem 15476
IniFiles.PHashItem type 15476
IniFiles.PPHashItem 15476
IniFiles.PPHashItem type 15476
IniFiles.TCustomIniFile 15434
IniFiles.TCustomIniFile.Create 15436
IniFiles.TCustomIniFile.DeleteKey 15437
IniFiles.TCustomIniFile.EraseSection 15438
IniFiles.TCustomIniFile.FileName 15450
IniFiles.TCustomIniFile.ReadBinaryStream 15438
IniFiles.TCustomIniFile.ReadBool 15439
IniFiles.TCustomIniFile.ReadDate 15439
IniFiles.TCustomIniFile.ReadDateTime 15440
IniFiles.TCustomIniFile.ReadFloat 15440
IniFiles.TCustomIniFile.ReadInteger 15441
IniFiles.TCustomIniFile.ReadSection 15442
IniFiles.TCustomIniFile.ReadSections 15442
IniFiles.TCustomIniFile.ReadSectionValues 15443
IniFiles.TCustomIniFile.ReadString 15443
IniFiles.TCustomIniFile.ReadTime 15444
IniFiles.TCustomIniFile.SectionExists 15444
IniFiles.TCustomIniFile.UpdateFile 15445

IniFiles.TCustomIniFile.ValueExists 15445
IniFiles.TCustomIniFile.WriteBinaryStream 15446
IniFiles.TCustomIniFile.WriteBool 15446
IniFiles.TCustomIniFile.WriteDate 15446
IniFiles.TCustomIniFile.WriteDateTime 15447
IniFiles.TCustomIniFile.WriteFloat 15447
IniFiles.TCustomIniFile.WriteInteger 15448
IniFiles.TCustomIniFile.WriteString 15448
IniFiles.TCustomIniFile.WriteTime 15449
IniFiles.THashedStringList 15450
IniFiles.THashedStringList.Changed 15454
IniFiles.THashedStringList.Destroy 15453
IniFiles.THashedStringList.IndexOf 15454
IniFiles.THashedStringList.IndexOfName 15455
IniFiles.THashItem 15475
IniFiles.THashItem record 15475
IniFiles.TIniFile 15455
IniFiles.TIniFile.DeleteKey 15458
IniFiles.TIniFile.Destroy 15457
IniFiles.TIniFile.EraseSection 15458
IniFiles.TIniFile.ReadSection 15459
IniFiles.TIniFile.ReadSections 15459
IniFiles.TIniFile.ReadSectionValues 15460
IniFiles.TIniFile.ReadString 15460
IniFiles.TIniFile.UpdateFile 15461
IniFiles.TIniFile.WriteString 15461
IniFiles.TMemIniFile 15462
IniFiles.TMemIniFile.CaseSensitive 15471
IniFiles.TMemIniFile.Clear 15465
IniFiles.TMemIniFile.Create 15464
IniFiles.TMemIniFile.DeleteKey 15465
IniFiles.TMemIniFile.Destroy 15464
IniFiles.TMemIniFile.EraseSection 15466
IniFiles.TMemIniFile.GetStrings 15466
IniFiles.TMemIniFile.ReadSection 15467
IniFiles.TMemIniFile.ReadSections 15467
IniFiles.TMemIniFile.ReadSectionValues 15468
IniFiles.TMemIniFile.ReadString 15468
IniFiles.TMemIniFile.Rename 15469
IniFiles.TMemIniFile.SetStrings 15469
IniFiles.TMemIniFile.UpdateFile 15470
IniFiles.TMemIniFile.WriteString 15470
IniFiles.TStringHash 15471
IniFiles.TStringHash.Add 15473
IniFiles.TStringHash.Clear 15473
IniFiles.TStringHash.Create 15472
IniFiles.TStringHash.Destroy 15472
IniFiles.TStringHash.Find 15473
IniFiles.TStringHash.HashOf 15474
IniFiles.TStringHash.Modify 15474
IniFiles.TStringHash.Remove 15474
IniFiles.TStringHash.ValueOf 15474
INotifyActivateModule interface 24776
about INotifyActivateModule interface 24776
INotifyActivateModule Members 24777
INotifyActivateModule methods 24777
NotifyActivateModule 24777
NotifyDeactivateModule 24777
INotifyAdapterChange interface 19724
about INotifyAdapterChange interface 19724
INotifyAdapterChange Members 19724
INotifyAdapterChange methods 19725
NotifyAdapterChange 19725
INotifyConnectionChange interface 26086
about INotifyConnectionChange interface 26086
ConnectionChange 26087
INotifyConnectionChange Members 26086
INotifyConnectionChange methods 26086
INotifyList interface 19725
about INotifyList interface 19725
AddNotify 19726
INotifyList Members 19726
INotifyList methods 19726
RemoveNotify 19726
INotifyWebActivate interface 24777
about INotifyWebActivate interface 24777
INotifyWebActivate Members 24778
INotifyWebActivate methods 24778

NotifyActivate 24778
NotifyDeactivate 24778
IntfInfo 15477
IntfInfo namespace 15477
Classes 15479
Constants 15491
Functions 15480
Structs, Records, Enums 15485
Types 15488
IntfInfo.CallingConventionName 15491
IntfInfo.CallingConventionName constant 15491
IntfInfo.EInterfaceRTTIException 15479
IntfInfo.GetClsMemberTypeInfo 15480
IntfInfo.GetClsMemberTypeInfo function 15480
IntfInfo.GetDynArrayElTypeInfo 15480
IntfInfo.GetDynArrayElTypeInfo function 15480
IntfInfo.GetDynArrayNextInfo 15481
IntfInfo.GetDynArrayNextInfo function 15481
IntfInfo.GetDynArrayNextInfo2 15481
IntfInfo.GetDynArrayNextInfo2 function 15481
IntfInfo.GetIntfMetaData 15481, 15482
IntfInfo.GetIntfMetaData function 15481, 15482
IntfInfo.GetMethNum 15482
IntfInfo.GetMethNum function 15482
IntfInfo.GetPropListFlat 15482
IntfInfo.GetPropListFlat function 15482
IntfInfo.IsStoredPropConst 15483
IntfInfo.IsStoredPropConst function 15483
IntfInfo.OtherTypeName 15483
IntfInfo.OtherTypeName function 15483
IntfInfo.PIntfMetaData 15488
IntfInfo.PIntfMetaData type 15488
IntfInfo.PIntfMethEntry 15488
IntfInfo.PIntfMethEntry type 15488
IntfInfo.PIntfParamEntry 15489
IntfInfo.PIntfParamEntry type 15489
IntfInfo.SameTypeInfo 15483
IntfInfo.SameTypeInfo function 15483
IntfInfo.TCallConv 15485
IntfInfo.TCallConv enumeration 15485
IntfInfo.TFillMethodArrayOpt 15485
IntfInfo.TFillMethodArrayOpt enumeration 15485
IntfInfo.TIntfMetaData 15486
IntfInfo.TIntfMetaData record 15486
IntfInfo.TIntfMethEntry 15486
IntfInfo.TIntfMethEntry record 15486
IntfInfo.TIntfMethEntryArray 15489
IntfInfo.TIntfMethEntryArray type 15489
IntfInfo.TIntfParamEntry 15487
IntfInfo.TIntfParamEntry record 15487
IntfInfo.TIntfParamEntryArray 15489
IntfInfo.TIntfParamEntryArray type 15489
IntfInfo.TPIntfMethEntryArray 15490
IntfInfo.TPIntfMethEntryArray type 15490
IntfInfo.TypeInfoNames 15491
IntfInfo.TypeInfoNames constant 15491
IntfInfo.TypeNamesMatch 15484
IntfInfo.TypeNamesMatch function 15484
InvokeRegistry 15493
InvokeRegistry namespace 15493
Classes 15496
Constants 15645
Functions 15619
Interfaces 15609
Structs, Records, Enums 15624
Types 15636
Variables 15644
InvokeRegistry.AppNameSpacePrefix 15644
InvokeRegistry.AppNameSpacePrefix variable 15644
InvokeRegistry.AS_ATTRIBUTE 15645
InvokeRegistry.AS_ATTRIBUTE constant 15645
InvokeRegistry.eHeaderMethodType 15624
InvokeRegistry.eHeaderMethodType enumeration 15624
InvokeRegistry.ERemotableException 15496
InvokeRegistry.ERemotableException.Create 15498
InvokeRegistry.ERemotableException.FaultActor 15499
InvokeRegistry.ERemotableException.FaultCode 15499
InvokeRegistry.ERemotableException.FaultDetail 15500

InvokeRegistry.ERemotableExceptionClass 15636
InvokeRegistry.ERemotableExceptionClass type 15636
InvokeRegistry.ETypeRegistryException 15500
InvokeRegistry.ExtNameMapItem 15625
InvokeRegistry.ExtNameMapItem record 15625
InvokeRegistry.GetRemotableDataContext 15619
InvokeRegistry.GetRemotableDataContext function 15619
InvokeRegistry.IHeadersSetter 15609
InvokeRegistry.IHeadersSetter.SetHeadersInOut 15610
InvokeRegistry.IntfExceptionItem 15625
InvokeRegistry.IntfExceptionItem record 15625
InvokeRegistry.IntfHeaderItem 15626
InvokeRegistry.IntfHeaderItem record 15626
InvokeRegistry.InvRegClassEntry 15627
InvokeRegistry.InvRegClassEntry record 15627
InvokeRegistry.InvRegIntfEntry 15627
InvokeRegistry.InvRegIntfEntry record 15627
InvokeRegistry.InvRegistry 15620
InvokeRegistry.InvRegistry function 15620
InvokeRegistry.InvString 15637
InvokeRegistry.InvString type 15637
InvokeRegistry.IObjConverter 15610
InvokeRegistry.IObjConverter.AddAttachment 15611
InvokeRegistry.IObjConverter.FindAttachment 15612
InvokeRegistry.IObjConverter.InitObjectFromSOAP 15612
InvokeRegistry.IObjConverter.ObjInstanceToSOAP 15612
InvokeRegistry.ISOAPHeaders 15613
InvokeRegistry.ISOAPHeaders.Get 15615
InvokeRegistry.ISOAPHeaders.GetOwnsSentHeaders 15615
InvokeRegistry.ISOAPHeaders.OwnsSentHeaders 15617
InvokeRegistry.ISOAPHeaders.Send 15616
InvokeRegistry.ISOAPHeaders.SendAt 15616
InvokeRegistry.ISOAPHeaders.SendCount 15617
InvokeRegistry.ISOAPHeaders.SetOwnsSentHeaders 15617
InvokeRegistry.MethParamNameMapItem 15629
InvokeRegistry.MethParamNameMapItem record 15629
InvokeRegistry.ObjectConvertOptions 15629
InvokeRegistry.ObjectConvertOptions enumeration 15629
InvokeRegistry.PTRemotable 15637
InvokeRegistry.PTRemotable type 15637
InvokeRegistry.RemClassRegistry 15620
InvokeRegistry.RemClassRegistry function 15620
InvokeRegistry.RemTypeRegistry 15621
InvokeRegistry.RemTypeRegistry function 15621
InvokeRegistry.SerializationOptions 15630
InvokeRegistry.SerializationOptions enumeration 15630
InvokeRegistry.SetRemotableDataContext 15622
InvokeRegistry.SetRemotableDataContext function 15622
InvokeRegistry.SubstituteStrings 15623
InvokeRegistry.SubstituteStrings function 15623
InvokeRegistry.TCreateInstanceProc 15637
InvokeRegistry.TCreateInstanceProc type 15637
InvokeRegistry.TDataContext 15501
InvokeRegistry.TDataContext.AddDynArrayToClear 15505
InvokeRegistry.TDataContext.AddObjectToDestroy 15506
InvokeRegistry.TDataContext.AddStrToClear 15506
InvokeRegistry.TDataContext.AddVariantToClear 15507
InvokeRegistry.TDataContext.AddWStrToClear 15507
InvokeRegistry.TDataContext.AllocData 15508
InvokeRegistry.TDataContext.Create 15502
InvokeRegistry.TDataContext.Data 15503
InvokeRegistry.TDataContext.DataOffset 15503
InvokeRegistry.TDataContext.DataP 15503
InvokeRegistry.TDataContext.Destroy 15502
InvokeRegistry.TDataContext.DynArrayToClear 15504
InvokeRegistry.TDataContext.FObjsToDestroy 15504
InvokeRegistry.TDataContext.GetDataPointer 15508
InvokeRegistry.TDataContext.RemoveObjectToDestroy 15508
InvokeRegistry.TDataContext.SetDataPointer 15509
InvokeRegistry.TDataContext.StrToClear 15504
InvokeRegistry.TDataContext.VarToClear 15504
InvokeRegistry.TDataContext.WStrToClear 15504
InvokeRegistry.TDynToClear 15631
InvokeRegistry.TDynToClear record 15631
InvokeRegistry.TExceptionItemArray 15638
InvokeRegistry.TExceptionItemArray type 15638
InvokeRegistry.THeaderItemArray 15638
InvokeRegistry.THeaderItemArray type 15638

InvokeRegistry.THeaderList 15509
InvokeRegistry.THeaderList.Add 15512
InvokeRegistry.THeaderList.Clear 15512
InvokeRegistry.THeaderList.Count 15514
InvokeRegistry.THeaderList.Create 15510
InvokeRegistry.THeaderList.Destroy 15511
InvokeRegistry.THeaderList.Extract 15512
InvokeRegistry.THeaderList.GetCount 15513
InvokeRegistry.THeaderList.GetHeader 15513
InvokeRegistry.THeaderList.GetOwnsObjects 15513
InvokeRegistry.THeaderList.Headers 15515
InvokeRegistry.THeaderList.IndexOf 15513
InvokeRegistry.THeaderList.OwnsObjects 15515
InvokeRegistry.THeaderList.SetOwnsObjects 15514
InvokeRegistry.THeaderMethodTypeArray 15638
InvokeRegistry.THeaderMethodTypeArray type 15638
InvokeRegistry.TIntfInvokeOption 15632
InvokeRegistry.TIntfInvokeOption enumeration 15632
InvokeRegistry.TIntfInvokeOptions 15639
InvokeRegistry.TIntfInvokeOptions type 15639
InvokeRegistry.TInvContext 15516
InvokeRegistry.TInvContext.AllocServerData 15517
InvokeRegistry.TInvContext.GetParamPointer 15518
InvokeRegistry.TInvContext.GetResultPointer 15518
InvokeRegistry.TInvContext.SetMethodInfo 15518
InvokeRegistry.TInvContext.SetParamPointer 15519
InvokeRegistry.TInvContext.SetResultPointer 15519
InvokeRegistry.TInvokableClass 15520
InvokeRegistry.TInvokableClass._AddRef 15523
InvokeRegistry.TInvokableClass._Release 15524
InvokeRegistry.TInvokableClass.AfterConstruction 15524
InvokeRegistry.TInvokableClass.BeforeDestruction 15524
InvokeRegistry.TInvokableClass.Create 15522
InvokeRegistry.TInvokableClass.FRefCount 15522
InvokeRegistry.TInvokableClass.FSOAPHeaders 15523
InvokeRegistry.TInvokableClass.NewInstance 15525
InvokeRegistry.TInvokableClass.QueryInterface 15525
InvokeRegistry.TInvokableClass.RefCount 15526
InvokeRegistry.TInvokableClassClass 15639
InvokeRegistry.TInvokableClassClass type 15639
InvokeRegistry.TInvokableClassRegistry 15526
InvokeRegistry.TInvokableClassRegistry.Create 15529
InvokeRegistry.TInvokableClassRegistry.Destroy 15530
InvokeRegistry.TInvokableClassRegistry.GetActionURIOfIID 15531
InvokeRegistry.TInvokableClassRegistry.GetActionURIOfInfo 15532
InvokeRegistry.TInvokableClassRegistry.GetClassFromIntfInfo 15532
InvokeRegistry.TInvokableClassRegistry.GetExceptionInfoForInterface 15533
InvokeRegistry.TInvokableClassRegistry.GetHeaderClass 15533
InvokeRegistry.TInvokableClassRegistry.GetHeaderInfoForInterface 15534
InvokeRegistry.TInvokableClassRegistry.GetHeaderName 15534
InvokeRegistry.TInvokableClassRegistry.GetHeaderNamespace 15535
InvokeRegistry.TInvokableClassRegistry.GetInfoForURI 15536
InvokeRegistry.TInvokableClassRegistry.GetInterfaceCount 15536
InvokeRegistry.TInvokableClassRegistry.GetInterfaceExternalName 15537
InvokeRegistry.TInvokableClassRegistry.GetInterfaceInfoFromName 15538
InvokeRegistry.TInvokableClassRegistry.GetInterfaceTypeInfo 15538
InvokeRegistry.TInvokableClassRegistry.GetIntfInvokeOptions 15539
InvokeRegistry.TInvokableClassRegistry.GetInvokableObjectFromClass 15539
InvokeRegistry.TInvokableClassRegistry.GetMethExternalName 15540
InvokeRegistry.TInvokableClassRegistry.GetMethInternalName 15540
InvokeRegistry.TInvokableClassRegistry.GetNamespaceByGUID 15541
InvokeRegistry.TInvokableClassRegistry.GetParamExternalNam

e
15541

InvokeRegistry.TInvokableClassRegistry.GetParamInternalNam
e
15541

InvokeRegistry.TInvokableClassRegistry.GetRegInterfaceEntry
15542

InvokeRegistry.TInvokableClassRegistry.GetRequestHeaderInfo
ForInterface
15542

InvokeRegistry.TInvokableClassRegistry.GetResponseHeaderIn
foForInterface
15543

InvokeRegistry.TInvokableClassRegistry.GetReturnParamName
s
15544

InvokeRegistry.TInvokableClassRegistry.GetUDDIInfo 15544

InvokeRegistry.TInvokableClassRegistry.GetWSDLEncoding
15545

InvokeRegistry.TInvokableClassRegistry.HasRegInterfaceImpl
15546

InvokeRegistry.TInvokableClassRegistry.RegisterAllSOAPAction
s
15546

InvokeRegistry.TInvokableClassRegistry.RegisterDefaultSOAPA
ction
15547

InvokeRegistry.TInvokableClassRegistry.RegisterException
15547

InvokeRegistry.TInvokableClassRegistry.RegisterExceptionMeth
od
15548

InvokeRegistry.TInvokableClassRegistry.RegisterExternalMethN
ame
15548

InvokeRegistry.TInvokableClassRegistry.RegisterExternalParam
Name
15549

InvokeRegistry.TInvokableClassRegistry.RegisterHeaderClass
15550

InvokeRegistry.TInvokableClassRegistry.RegisterHeaderMethod

15551

InvokeRegistry.TInvokableClassRegistry.RegisterInterface
15552

InvokeRegistry.TInvokableClassRegistry.RegisterInvokableClas
s
15553

InvokeRegistry.TInvokableClassRegistry.RegisterInvokeOptions
15554

InvokeRegistry.TInvokableClassRegistry.RegisterReturnParamN
ames
15554

InvokeRegistry.TInvokableClassRegistry.RegisterUDDIInfo
15555

InvokeRegistry.TInvokableClassRegistry.UnRegisterInterface
15555

InvokeRegistry.TInvokableClassRegistry.UnRegisterInvokableCl
ass
15556

InvokeRegistry.TInvRegIntfEntryArray 15639

InvokeRegistry.TInvRegIntfEntryArray type 15639

InvokeRegistry.TObjectConvertOptions 15640

InvokeRegistry.TObjectConvertOptions type 15640

InvokeRegistry.TObjMultiOptions 15633

InvokeRegistry.TObjMultiOptions enumeration 15633

InvokeRegistry.TPascalRemotableClassRegistry 15640

InvokeRegistry.TPascalRemotableClassRegistry type 15640

InvokeRegistry.TPascalRemotableTypeRegistry 15640

InvokeRegistry.TPascalRemotableTypeRegistry type 15640

InvokeRegistry.TRemHeaderEntry 15634

InvokeRegistry.TRemHeaderEntry record 15634

InvokeRegistry.TRemotable 15556

InvokeRegistry.TRemotable.Create 15558

InvokeRegistry.TRemotable.DataContext 15561

InvokeRegistry.TRemotable.Destroy 15558

InvokeRegistry.TRemotable.FSerializationOptions 15559

InvokeRegistry.TRemotable.ObjectToSOAP 15559

InvokeRegistry.TRemotable.SerializationOptions 15561

InvokeRegistry.TRemotable.SOAPToObject 15560

InvokeRegistry.TRemotableClass 15641

InvokeRegistry.TRemotableClass type 15641

InvokeRegistry.TRemotableClassRegistry 15641

InvokeRegistry.TRemotableClassRegistry type 15641

InvokeRegistry.TRemotableTypeRegistry 15562

InvokeRegistry.TRemotableTypeRegistry.AutoRegisterNativeTy
pes
15579

InvokeRegistry.TRemotableTypeRegistry.ClassOptions 15565

InvokeRegistry.TRemotableTypeRegistry.ClassToURI 15566

InvokeRegistry.TRemotableTypeRegistry.Create 15563

InvokeRegistry.TRemotableTypeRegistry.Destroy 15564

InvokeRegistry.TRemotableTypeRegistry.GetExternalPropName 15566
InvokeRegistry.TRemotableTypeRegistry.GetInternalPropName 15567
InvokeRegistry.TRemotableTypeRegistry.GetURICount 15567
InvokeRegistry.TRemotableTypeRegistry.GetURIMap 15568
InvokeRegistry.TRemotableTypeRegistry.GetVarTypeFromXSD 15569
InvokeRegistry.TRemotableTypeRegistry.GetXSDInfoForClass 15569
InvokeRegistry.TRemotableTypeRegistry.InfoToURI 15569
InvokeRegistry.TRemotableTypeRegistry.IsClassScalar 15570
InvokeRegistry.TRemotableTypeRegistry.Lock 15570
InvokeRegistry.TRemotableTypeRegistry.RegisterExternalPropName 15571
InvokeRegistry.TRemotableTypeRegistry.RegisterHolderClsMember 15571
InvokeRegistry.TRemotableTypeRegistry.RegisterSerializeOptions 15572
InvokeRegistry.TRemotableTypeRegistry.RegisterXSClass 15573
InvokeRegistry.TRemotableTypeRegistry.RegisterXSInfo 15574
InvokeRegistry.TRemotableTypeRegistry.SerializeOptions 15575
InvokeRegistry.TRemotableTypeRegistry.TypeInfoToXSD 15575
InvokeRegistry.TRemotableTypeRegistry.UnLock 15576
InvokeRegistry.TRemotableTypeRegistry.UnRegisterXSClass 15576
InvokeRegistry.TRemotableTypeRegistry.UnRegisterXSInfo 15577
InvokeRegistry.TRemotableTypeRegistry.URIToClass 15577
InvokeRegistry.TRemotableTypeRegistry.URIToInfo 15578
InvokeRegistry.TRemotableTypeRegistry.VariantToInfo 15578
InvokeRegistry.TRemotableTypeRegistry.XSDToTypeInfo 15579
InvokeRegistry.TRemotableXS 15580
InvokeRegistry.TRemotableXS.NativeToXS 15581
InvokeRegistry.TRemotableXS.ObjectToSOAP 15582
InvokeRegistry.TRemotableXS.SOAPToObject 15582
InvokeRegistry.TRemotableXS.XSToNative 15583
InvokeRegistry.TRemotableXSClass 15641
InvokeRegistry.TRemotableXSClass type 15641
InvokeRegistry.TRemRegEntry 15634
InvokeRegistry.TRemRegEntry record 15634
InvokeRegistry.TRemRegEntryArray 15642
InvokeRegistry.TRemRegEntryArray type 15642
InvokeRegistry.TRequiredArray 15642
InvokeRegistry.TRequiredArray type 15642
InvokeRegistry.TSerializationOptions 15642
InvokeRegistry.TSerializationOptions type 15642
InvokeRegistry.TSOAPAttachment 15583
InvokeRegistry.TSOAPAttachment.CacheFile 15592
InvokeRegistry.TSOAPAttachment.CacheFilePersist 15593
InvokeRegistry.TSOAPAttachment.ContentType 15593
InvokeRegistry.TSOAPAttachment.Create 15585
InvokeRegistry.TSOAPAttachment.Destroy 15586
InvokeRegistry.TSOAPAttachment.Encoding 15594
InvokeRegistry.TSOAPAttachment.GetSourceStream 15587
InvokeRegistry.TSOAPAttachment.Headers 15594
InvokeRegistry.TSOAPAttachment.Init 15587
InvokeRegistry.TSOAPAttachment.InternalSetCacheFile 15588
InvokeRegistry.TSOAPAttachment.InternalSetSourceStream 15588
InvokeRegistry.TSOAPAttachment.ObjectToSOAP 15588
InvokeRegistry.TSOAPAttachment.Ownership 15595
InvokeRegistry.TSOAPAttachment.SaveToFile 15589
InvokeRegistry.TSOAPAttachment.SaveToStream 15589
InvokeRegistry.TSOAPAttachment.SetOwnership 15590
InvokeRegistry.TSOAPAttachment.SetSourceFile 15590
InvokeRegistry.TSOAPAttachment.SetSourceStream 15591
InvokeRegistry.TSOAPAttachment.SetSourceString 15591
InvokeRegistry.TSOAPAttachment.SOAPToObject 15591
InvokeRegistry.TSOAPAttachment.SourceStream 15595
InvokeRegistry.TSOAPAttachment.SourceString 15596
InvokeRegistry.TSOAPAttachmentClass 15643
InvokeRegistry.TSOAPAttachmentClass type 15643
InvokeRegistry.TSOAPHeader 15596
InvokeRegistry.TSOAPHeader.Actor 15599
InvokeRegistry.TSOAPHeader.MustUnderstand 15600
InvokeRegistry.TSOAPHeader.ObjectToSOAP 15598

InvokeRegistry.TSOAPHeader.SOAPToObject 15599
InvokeRegistry.TSOAPHeaderClass 15643
InvokeRegistry.TSOAPHeaderClass type 15643
InvokeRegistry.TSOAPHeaders 15600
InvokeRegistry.TSOAPHeaders.Get 15603
InvokeRegistry.TSOAPHeaders.GetOwnsSentHeaders 15603
InvokeRegistry.TSOAPHeaders.Send 15604
InvokeRegistry.TSOAPHeaders.SendAt 15604
InvokeRegistry.TSOAPHeaders.SendCount 15605
InvokeRegistry.TSOAPHeaders.SetOwnsSentHeaders 15605
InvokeRegistry.TSOAPHeadersBase 15605
InvokeRegistry.TSOAPHeadersBase.FHeadersInbound 15607
InvokeRegistry.TSOAPHeadersBase.FHeadersOutBound 15608
InvokeRegistry.TSOAPHeadersBase.SetHeadersInOut 15608
InvokeRegistry.XMLBase64Types 15645
InvokeRegistry.XMLBase64Types constant 15645
InvokeRegistry.XMLSchemaInstNamepspaces 15646
InvokeRegistry.XMLSchemaInstNamepspaces constant 15646
InvokeRegistry.XMLSchemaNamepspaces 15646
InvokeRegistry.XMLSchemaNamepspaces constant 15646
ioBeforeLoad enumeration member 26002
IObjConverter interface 15610
about IObjConverter interface 15610
AddAttachment 15611
FindAttachment 15612
InitObjectFromSOAP 15612
IObjConverter Members 15611
IObjConverter methods 15611
ObjInstanceToSOAP 15612
IObjectContext interface 17274
about IObjectContext interface 17274
CreateInstance 17275
DisableCommit 17275
EnableCommit 17276
IObjectContext Members 17274
IObjectContext methods 17275
IsCallerInRole 17276
IsInTransaction 17276
IsSecurityEnabled 17276
SetAbort 17276
SetComplete 17277
IObjectContextActivity interface 17277
about IObjectContextActivity interface 17277
GetActivityId 17278
IObjectContextActivity Members 17277
IObjectContextActivity methods 17277
IObjectControl interface 17278
about IObjectControl interface 17278
Activate 17279
CanBePooled 17279
Deactivate 17279
IObjectControl Members 17278
IObjectControl methods 17278
ioDefault enumeration member 15632
ioDocument enumeration member 15632
ioHasAllSOAPActions enumeration member 15632
ioHasDefaultSOAPAction enumeration member 15632
ioHasNamespace enumeration member 15632
ioHasReturnParamNames enumeration member 15632
ioHasUDDIInfo enumeration member 15632
ioIsAppServerSOAP enumeration member 15632
IOleForm interface 14340
about IOleForm interface 14340
IOleForm Members 14340
IOleForm methods 14340
OnDestroy 14340
OnResize 14341
ioLiteral enumeration member 15632
ioLoaded enumeration member 26002
IOPConvert interface 18122
about IOPConvert interface 18122
Attachments 18130
Encoding 18130
GetAttachments 18124
GetEncoding 18124
GetOptions 18124
GetTempDir 18124
InvContextToMsg 18125

IOPConvert Members 18123
IOPConvert methods 18123
IOPConvert properties 18130
MakeFault 18125
MakeResponse 18126
MsgToInvContext 18127
MsgToInvContext method 18126
Options 18131
ProcessResponse 18128
ProcessResponse method 18127
SetAttachments 18128
SetEncoding 18129
SetOptions 18129
SetTempDir 18129
TempDir 18131
IOperation interface 25860
about IOperation interface 25860
Faults 25863
GetFaults 25862
GetInput 25862
GetName 25862
GetOutput 25863
Input 25863
IOperation Members 25860
IOperation methods 25862
IOperation properties 25863
Name 25864
Output 25864
SetName 25863
IOperations interface 25864
about IOperations interface 25864
Add 25866, 25867
Add method 25866
GetOperation 25867
Insert 25867
IOperations Members 25864
IOperations methods 25866
IOperations properties 25867
Operation 25867
IORB interface 18201
about IORB interface 18201
BOAInit 18202
CreateAliasTC 18202
CreateArrayTC 18202
CreateObjRefTC 18203
CreateSequenceTC 18203
CreateStructTC 18203
CreateTC 18203
DispatchStruct 18203
FindRepositoryTC 18204
IORB Members 18201
IORB methods 18201
MakeAny 18204
ObjectToString 18204
Orb 18204
Shutdown 18205
StringToObject 18205
IPageDispatcher interface 19727
about IPageDispatcher interface 19727
DefaultPageName 19731
DispatchPageName 19728
GetCanView 19729
GetDefaultPageName 19729
GetLoginRequired 19729
IPageDispatcher Members 19727
IPageDispatcher methods 19728
IPageDispatcher properties 19731
PageNameOfRequest 19730
RedirectToPageName 19730
IPageResult interface 19732
about IPageResult interface 19732
DispatchPage 19732
IncludePage 19733
IPageResult Members 19732
IPageResult methods 19732
RedirectToPage 19733
IParam interface 25868
about IParam interface 25868

GetMessage 25870
GetName 25870
IParam Members 25868
IParam methods 25869
IParam properties 25871
Message 25871
Name 25871
SetMessage 25870
SetName 25870
IPart interface 25871
about IPart interface 25871
Element 25875
GetElement 25873
GetName 25874
GetType_ 25874
IPart Members 25872
IPart methods 25873
IPart properties 25875
Name 25875
SetElement 25874
SetName 25874
SetType_ 25874
Type_ 25875
IParts interface 25876
about IParts interface 25876
Add 25878
GetPart 25878
Insert 25878
IParts Members 25876
IParts methods 25877
IParts properties 25878
Part 25879
IPictureAccess interface 1707
about IPictureAccess interface 1707
GetOlePicture 1708
IPictureAccess Members 1708
IPictureAccess methods 1708
SetOlePicture 1708
IPort interface 25879
about IPort interface 25879
Binding 25882
GetBinding 25881
GetName 25881
IPort Members 25879
IPort methods 25881
IPort properties 25882
Name 25882
SetBinding 25881
SetName 25881
IPorts interface 25882
about IPorts interface 25882
Add 25884
GetPort 25885
Insert 25885
IPorts Members 25883
IPorts methods 25884
IPorts properties 25885
Port 25885
IPortType interface 25885
about IPortType interface 25885
GetName 25887
GetOperations 25888
IPortType Members 25886
IPortType methods 25887
IPortType properties 25888
Name 25888
Operations 25888
SetName 25888
IPortTypes interface 25889
about IPortTypes interface 25889
Add 25891
GetPortType 25891
Insert 25891
IPortTypes Members 25889
IPortTypes methods 25891
IPortTypes properties 25891
PortType 25892
IPrepareDataSet interface 8109

about IPrepareDataSet interface 8109
IPrepareDataSet Members 8109
IPrepareDataSet methods 8109
PrepareDataSet 8109
IProduceContent interface 15223
about IProduceContent interface 15223
IProduceContent Members 15224
IProduceContent methods 15224
ProduceContent 15224
IProduceContentFrom interface 15225
about IProduceContentFrom interface 15225
IProduceContentFrom Members 15225
IProduceContentFrom methods 15225
ProduceContentFromStream 15225
ProduceContentFromString 15226
IProducerEditorViewSupport interface 24779
about IProducerEditorViewSupport interface 24779
GetTemplateFileType 24780
GetXMLData 24780
GetXSLData 24780
HasHTMLBrowserView 24780
HasScriptView 24780
HasXMLBrowserView 24781
HasXSLBrowserView 24781
IProducerEditorViewSupport Members 24779
IProducerEditorViewSupport methods 24779
IProperty interface 11197
about IProperty interface 11197
Activate 11199
AllEqual 11199
AutoFill 11200
Edit 11200
GetAttributes 11200
GetEditLimit 11201
GetEditValue 11201
GetName 11202
GetProperties 11202
GetPropInfo 11202
GetPropType 11203
GetValue 11203
GetValues 11203
HasInstance 11204
IProperty Members 11198
IProperty methods 11198
Revert 11204
SetValue 11204
ValueAvailable 11205
IProperty70 interface 11205
about IProperty70 interface 11205
GetIsDefault 11206
IProperty70 Members 11205
IProperty70 methods 11206
IProperty70 properties 11206
IsDefault 11206
IProperty80 interface 11207
about IProperty80 interface 11207
Edit 11208
IProperty80 Members 11207
IProperty80 methods 11208
IPropertyControl interface 11208
about IPropertyControl interface 11208
ClearCache 11209
IPropertyControl Members 11208
IPropertyControl methods 11208
IPropertyDescription interface 11209
about IPropertyDescription interface 11209
GetDescription 11210
IPropertyDescription Members 11209
IPropertyDescription methods 11209
IPropertyHost interface 11210
about IPropertyHost interface 11210
CloseDropDown 11211
DropDownControl 11211
IPropertyHost Members 11210
IPropertyHost methods 11210
IPropertyKind interface 11211
about IPropertyKind interface 11211
GetKind 11212

IPropertyKind Members 11211
IPropertyKind methods 11212
IPropertyKind properties 11212
Kind 11212
IProviderContainer interface 18613
about IProviderContainer interface 18613
IProviderContainer Members 18613
IProviderContainer methods 18613
RegisterProvider 18614
UnRegisterProvider 18614
IProviderSupport interface 7576
about IProviderSupport interface 7576
IProviderSupport Members 7576
IProviderSupport methods 7577
PSEndTransaction 7578
PSExecute 7578
PSExecuteStatement 7579
PSGetAttributes 7579
PSGetCommandText 7579
PSGetCommandType 7580
PSGetDefaultOrder 7580
PSGetIndexDefs 7581
PSGetKeyFields 7581
PSGetParams 7582
PSGetQuoteChar 7582
PSGetTableName 7583
PSGetUpdateException 7583
PSInTransaction 7583
PSIsSQLBased 7584
PSIsSQLSupported 7584
PSReset 7585
PSSetCommandText 7585
PSSetParams 7586
PSStartTransaction 7586
PSUpdateRecord 7586
IProviderSupport2 interface 7587
about IProviderSupport2 interface 7587
IProviderSupport2 Members 7588
IProviderSupport2 methods 7588
PSEndTransaction 7589
PSExecute 7590
PSExecuteStatement 7590
PSGetAttributes 7591
PSGetCommandText 7591
PSGetCommandType 7591
PSGetDefaultOrder 7592
PSGetIndexDefs 7592
PSGetKeyFields 7593
PSGetParams 7593
PSGetQuoteChar 7594
PSGetTableName 7594
PSGetUpdateException 7594
PSInTransaction 7595
PSIsSQLBased 7595
PSIsSQLSupported 7596
PSReset 7596
PSSetCommandText 7596
PSSetParams 7597
PSStartTransaction 7597
PSUpdateRecord 7598
IQualifiedName interface 25994
about IQualifiedName interface 25994
GetName 25995
GetNamespace 25996
IQualifiedName Members 25995
IQualifiedName methods 25995
IQualifiedName properties 25997
Name 25997
Namespace 25997
SetName 25996
SetNamespace 25996
IQueryField interface 17070
about IQueryField interface 17070
GetParamName 17071
GetText 17071
IQueryField Members 17070
IQueryField methods 17071
IQueryField properties 17072

ParamName 17072
SetParamName 17071
SetText 17072
Text 17072
IQueryFields interface 17073
about IQueryFields interface 17073
GetParamsList 17073
IQueryFields Members 17073
IQueryFields methods 17073
IsParamInUse 17074
ValidateDataSet 17074
IQueryForm interface 17074
about IQueryForm interface 17074
IReadWriteSync interface 22778
about IReadWriteSync interface 22778
BeginRead 22779
BeginWrite 22779
EndRead 22779
EndWrite 22779
IReadWriteSync Members 22778
IReadWriteSync methods 22778
IReferenceProperty interface 11212
about IReferenceProperty interface 11212
GetComponentReference 11213
IReferenceProperty Members 11213
IReferenceProperty methods 11213
IRestoreDefaults interface 17074
about IRestoreDefaults interface 17074
IRestoreDefaults Members 17075
IRestoreDefaults methods 17075
RestoreDefaults 17075
irImage enumeration member 5454
irIndent enumeration member 5454
IRIOAccess interface 18733
about IRIOAccess interface 18733
GetRIO 18734
IRIOAccess Members 18733
IRIOAccess methods 18734
IRIOAccess properties 18734
RIO 18734
irParam enumeration member 5454
irState enumeration member 5454
irText enumeration member 5454
isActivating enumeration member 5454
ISAPIApp 15647
ISAPIApp namespace 15647
Classes 15648
Functions 15657
Variables 15659
ISAPIApp.DispatchThread 15659
ISAPIApp.DispatchThread variable 15659
ISAPIApp.GetExtensionVersion 15657
ISAPIApp.GetExtensionVersion function 15657
ISAPIApp.HttpExtensionProc 15657
ISAPIApp.HttpExtensionProc function 15657
ISAPIApp.TerminateExtension 15658
ISAPIApp.TerminateExtension function 15658
ISAPIApp.TISAPIApplication 15648
ISAPIApp.TISAPIApplication.Create 15652
ISAPIApp.TISAPIApplication.Destroy 15652
ISAPIApp.TISAPIApplication.GetExtensionVersion 15652
ISAPIApp.TISAPIApplication.HttpExtensionProc 15653
ISAPIApp.TISAPIApplication.TerminateExtension 15653
ISAPIApp.TISAPIFactory 15654
ISAPIApp.TISAPIFactory.Create 15655
ISAPIApp.TISAPIFactory.NewRequest 15655
ISAPIApp.TISAPIFactory.NewResponse 15655
IsapiHTTP 15661
IsapiHTTP namespace 15661
Classes 15662
IsapiHTTP.TISAPIRequest 15662
IsapiHTTP.TISAPIRequest.Create 15665
IsapiHTTP.TISAPIRequest.ECB 15670
IsapiHTTP.TISAPIRequest.GetDateVariable 15665
IsapiHTTP.TISAPIRequest.GetFieldByName 15666
IsapiHTTP.TISAPIRequest.GetIntegerVariable 15666
IsapiHTTP.TISAPIRequest.GetStringVariable 15666
IsapiHTTP.TISAPIRequest.ReadClient 15667

IsapiHTTP.TISAPIRequest.ReadString 15668
IsapiHTTP.TISAPIRequest.TranslateURI 15668
IsapiHTTP.TISAPIRequest.WriteClient 15669
IsapiHTTP.TISAPIRequest.WriteHeaders 15669
IsapiHTTP.TISAPIRequest.WriteString 15670
IsapiHTTP.TISAPIResponse 15670
IsapiHTTP.TISAPIResponse.Create 15673
IsapiHTTP.TISAPIResponse.GetContent 15674
IsapiHTTP.TISAPIResponse.GetDateVariable 15674
IsapiHTTP.TISAPIResponse.GetIntegerVariable 15675
IsapiHTTP.TISAPIResponse.GetLogMessage 15675
IsapiHTTP.TISAPIResponse.GetStatusCode 15675
IsapiHTTP.TISAPIResponse.GetStringVariable 15676
IsapiHTTP.TISAPIResponse.InitResponse 15676
IsapiHTTP.TISAPIResponse.SendRedirect 15677
IsapiHTTP.TISAPIResponse.SendResponse 15677
IsapiHTTP.TISAPIResponse.SendStream 15677
IsapiHTTP.TISAPIResponse.Sent 15678
IsapiHTTP.TISAPIResponse.SetContent 15678
IsapiHTTP.TISAPIResponse.SetDateVariable 15678
IsapiHTTP.TISAPIResponse.SetIntegerVariable 15679
IsapiHTTP.TISAPIResponse.SetLogMessage 15679
IsapiHTTP.TISAPIResponse.SetStatusCode 15680
IsapiHTTP.TISAPIResponse.SetStringVariable 15680
IScriptContext interface 15321
about IScriptContext interface 15321
GetWebModuleContext 15322
IScriptContext Members 15322
IScriptContext methods 15322
IScriptContext properties 15322
WebModuleContext 15322
IScriptErrorList interface 19295
about IScriptErrorList interface 19295
advanceError 19296
canAdvanceError 19296
canRetreatError 19296
getAlwaysShowLockState 19297
getDetailsPaneOpen 19297
getErrorChar 19297
getErrorCode 19297
getErrorLine 19298
getErrorMsg 19298
getErrorUrl 19298
getPerErrorDisplay 19298
IScriptErrorList Members 19295
IScriptErrorList methods 19296
retreatError 19298
setDetailsPaneOpen 19299
setPerErrorDisplay 19299
IScriptErrorListDisp interface 19299
about IScriptErrorListDisp interface 19299
advanceError 19300
canAdvanceError 19301
canRetreatError 19301
getAlwaysShowLockState 19301
getDetailsPaneOpen 19301
getErrorChar 19301
getErrorCode 19302
getErrorLine 19302
getErrorMsg 19302
getErrorUrl 19302
getPerErrorDisplay 19303
IScriptErrorListDisp Members 19299
IScriptErrorListDisp methods 19300
retreatError 19303
setDetailsPaneOpen 19303
setPerErrorDisplay 19303
IScriptProducer interface 15323
about IScriptProducer interface 15323
Content 15327
Errors 15327
Evaluate 15324
GetContent 15324
GetErrors 15324
GetHTMLBlock 15325
GetHTMLBlockCount 15325
HandleScriptError 15325
HTMLBlockCount 15328

HTMLBlocks 15328
IScriptProducer Members 15323
IScriptProducer methods 15323
IScriptProducer properties 15327
ParseStream 15325
ParseString 15325
ReplaceTags 15326
SetContent 15326
Write 15326
Write method 15326
WriteItem 15327
isCut enumeration member 5454
isDropHilited enumeration member 5454
ISearch interface 19303
about ISearch interface 19303
Get_Id 19304
Get_Title 19305
Get_URL 19305
Id 19305
ISearch Members 19304
ISearch methods 19304
ISearch properties 19305
Title 19305
URL 19306
ISearchAssistantOC interface 19306
about ISearchAssistantOC interface 19306
AddNextMenuItem 19308
ASProvider 19314
ASSetting 19314
EncodeString 19308
EventHandled 19315
FindComputer 19308
FindFilesOrFolders 19309
FindOnWeb 19309
FindPeople 19309
FindPrinter 19309
Get_ASProvider 19309
Get_ASSetting 19310
Get_InWebFolder 19310
Get_SearchAssistantDefault 19310
Get_Searches 19310
Get_ShellFeaturesEnabled 19311
Get_Version 19311
GetProperty 19311
GetSearchAssistantURL 19311
InWebFolder 19315
ISearchAssistantOC Members 19306
ISearchAssistantOC methods 19307
ISearchAssistantOC properties 19314
IsRestricted 19311
NavigateToDefaultSearch 19312
NETDetectNextNavigate 19312
NotifySearchSettingsChanged 19312
PutFindText 19312
PutProperty 19313
ResetNextMenu 19313
SearchAssistantDefault 19315
Searches 19315
Set_ASProvider 19313
Set_ASSetting 19313
Set_EventHandled 19313
SetDefaultSearchUrl 19314
ShellFeaturesEnabled 19316
Version 19316
ISearchAssistantOC2 interface 19316
about ISearchAssistantOC2 interface 19316
Get_ShowFindPrinter 19318
ISearchAssistantOC2 Members 19316
ISearchAssistantOC2 methods 19317
ISearchAssistantOC2 properties 19318
ShowFindPrinter 19318
ISearchAssistantOC2Disp interface 19318
about ISearchAssistantOC2Disp interface 19318
AddNextMenuItem 19320
ASProvider 19324
ASSetting 19324
EncodeString 19320
EventHandled 19324

FindComputer 19320
FindFilesOrFolders 19320
FindOnWeb 19321
FindPeople 19321
FindPrinter 19321
GetProperty 19321
GetSearchAssistantURL 19322
InWebFolder 19325
ISearchAssistantOC2Disp Members 19319
ISearchAssistantOC2Disp methods 19319
ISearchAssistantOC2Disp properties 19324
IsRestricted 19322
NavigateToDefaultSearch 19322
NETDetectNextNavigate 19322
NotifySearchSettingsChanged 19322
PutFindText 19323
PutProperty 19323
ResetNextMenu 19323
SearchAssistantDefault 19325
Searches 19325
SetDefaultSearchUrl 19323
ShellFeaturesEnabled 19325
ShowFindPrinter 19325
Version 19326
ISearchAssistantOC3 interface 19326
about ISearchAssistantOC3 interface 19326
Get_SearchCompanionAvailable 19328
Get_UseSearchCompanion 19328
ISearchAssistantOC3 Members 19326
ISearchAssistantOC3 methods 19327
ISearchAssistantOC3 properties 19328
SearchCompanionAvailable 19328
Set_UseSearchCompanion 19328
UseSearchCompanion 19329
ISearchAssistantOC3Disp interface 19329
about ISearchAssistantOC3Disp interface 19329
AddNextMenuItem 19330
ASProvider 19335
ASSetting 19335
EncodeString 19331
EventHandled 19335
FindComputer 19331
FindFilesOrFolders 19331
FindOnWeb 19331
FindPeople 19332
FindPrinter 19332
GetProperty 19332
GetSearchAssistantURL 19332
InWebFolder 19335
ISearchAssistantOC3Disp Members 19329
ISearchAssistantOC3Disp methods 19330
ISearchAssistantOC3Disp properties 19334
IsRestricted 19332
NavigateToDefaultSearch 19333
NETDetectNextNavigate 19333
NotifySearchSettingsChanged 19333
PutFindText 19333
PutProperty 19334
ResetNextMenu 19334
SearchAssistantDefault 19335
SearchCompanionAvailable 19336
Searches 19336
SetDefaultSearchUrl 19334
ShellFeaturesEnabled 19336
ShowFindPrinter 19336
UseSearchCompanion 19337
Version 19337
ISearchAssistantOCDisp interface 19337
about ISearchAssistantOCDisp interface 19337
AddNextMenuItem 19339
ASProvider 19343
ASSetting 19343
EncodeString 19339
EventHandled 19343
FindComputer 19339
FindFilesOrFolders 19339
FindOnWeb 19339
FindPeople 19340

FindPrinter 19340
GetProperty 19340
GetSearchAssistantURL 19340
InWebFolder 19343
ISearchAssistantOCDisp Members 19337
ISearchAssistantOCDisp methods 19338
ISearchAssistantOCDisp properties 19342
IsRestricted 19341
NavigateToDefaultSearch 19341
NETDetectNextNavigate 19341
NotifySearchSettingsChanged 19341
PutFindText 19341
PutProperty 19342
ResetNextMenu 19342
SearchAssistantDefault 19343
Searches 19344
SetDefaultSearchUrl 19342
ShellFeaturesEnabled 19344
Version 19344
ISearchDisp interface 19344
about ISearchDisp interface 19344
Id 19345
ISearchDisp Members 19345
ISearchDisp properties 19345
Title 19345
URL 19346
ISearches interface 19346
_NewEnum 19347
about ISearches interface 19346
Count 19348
Default 19348
Get_Count 19347
Get_Default 19347
ISearches Members 19346
ISearches methods 19347
ISearches properties 19348
Item 19347
ISearchesDisp interface 19348
_NewEnum 19349
about ISearchesDisp interface 19348
Count 19350
Default 19350
ISearchesDisp Members 19349
ISearchesDisp methods 19349
ISearchesDisp properties 19350
Item 19349
ISecurityProperty interface 17279
about ISecurityProperty interface 17279
GetDirectCallerSID 17280
GetDirectCreatorSID 17280
GetOriginalCallerSID 17280
GetOriginalCreatorSID 17281
ISecurityProperty Members 17280
ISecurityProperty methods 17280
ReleaseSID 17281
ISelectionEditor interface 11213
about ISelectionEditor interface 11213
ExecuteVerb 11214
GetVerb 11214
GetVerbCount 11215
ISelectionEditor Members 11214
ISelectionEditor methods 11214
PrepareItem 11215
RequiresUnits 11215
ISelectionEditorList interface 11215
about ISelectionEditorList interface 11215
Count 11217
Get 11216
GetCount 11216
ISelectionEditorList Members 11216
ISelectionEditorList methods 11216
ISelectionEditorList properties 11217
Items 11217
ISelectionPropertyFilter interface 11217
about ISelectionPropertyFilter interface 11217
FilterProperties 11218
ISelectionPropertyFilter Members 11218
ISelectionPropertyFilter methods 11218

ISendDataBlock interface 18993
about ISendDataBlock interface 18993
ISendDataBlock Members 18994
ISendDataBlock methods 18994
Send 18994
IServerExceptionHandler interface 5588
about IServerExceptionHandler interface 5588
IServerExceptionHandler Members 5588
IServerExceptionHandler methods 5588
OnException 5589
IService interface 25892
about IService interface 25892
GetName 25894
GetPorts 25894
IService Members 25892
IService methods 25894
IService properties 25894
Name 25895
Ports 25895
SetName 25894
IServices interface 25895
about IServices interface 25895
Add 25897
GetService 25897
Insert 25898
IServices Members 25895
IServices methods 25897
IServices properties 25898
Service 25898
ISessionsService interface 19733
about ISessionsService interface 19733
CheckStatusChange 19734
Clear 19734
ISessionsService Members 19734
ISessionsService methods 19734
NewSession 19735
NotifyStatusChange 19735
ISessionsServiceConnectID interface 19736
about ISessionsServiceConnectID interface 19736
Connect 19736
ConnectActive 19737
ISessionsServiceConnectID Members 19736
ISessionsServiceConnectID methods 19736
ISessionsServicePersist interface 19737
about ISessionsServicePersist interface 19737
ISessionsServicePersist Members 19737
ISessionsServicePersist methods 19737
LoadFromFile 19738
LoadFromStream 19738
SaveToFile 19738
SaveToStream 19738
ISessionsServiceStoreID interface 19739
about ISessionsServiceStoreID interface 19739
ISessionsServiceStoreID Members 19739
ISessionsServiceStoreID methods 19739
ReadSessionID 19739
WriteSessionID 19740
ISetAppDispatcher interface 15226
about ISetAppDispatcher interface 15226
ISetAppDispatcher Members 15226
ISetAppDispatcher methods 15226
SetAppDispatcher 15227
ISetSubComponents interface 24581
about ISetSubComponents interface 24581
ISetSubComponents Members 24581
ISetSubComponents methods 24582
SetActions 24582
SetData 24582
ISetWebContentOptions interface 24781
about ISetWebContentOptions interface 24781
ISetWebContentOptions Members 24781
ISetWebContentOptions methods 24782
SetWebContentOptions 24782
ISetWebFieldValue interface 24582
about ISetWebFieldValue interface 24582
ISetWebFieldValue Members 24583
ISetWebFieldValue methods 24583
SetValue 24583

isFocused enumeration member 5454
ISharedProperty interface 17281
about ISharedProperty interface 17281
Get_Value 17282
ISharedProperty Members 17281
ISharedProperty methods 17282
ISharedProperty properties 17282
Set_Value 17282
Value 17283
ISharedPropertyGroup interface 17283
about ISharedPropertyGroup interface 17283
CreateProperty 17284
CreatePropertyByPosition 17284
Get_PropertyByName 17284
Get_PropertyByPosition 17285
ISharedPropertyGroup Members 17283
ISharedPropertyGroup methods 17284
ISharedPropertyGroup properties 17285
PropertyByName 17285
PropertyByPosition 17285
ISharedPropertyGroupManager interface 17285
_NewEnum 17287
about ISharedPropertyGroupManager interface 17285
CreatePropertyGroup 17286
Get__NewEnum 17287
Get_Group 17287
Group 17287
ISharedPropertyGroupManager Members 17286
ISharedPropertyGroupManager methods 17286
ISharedPropertyGroupManager properties 17287
IShellFavoritesNameSpace interface 19350
about IShellFavoritesNameSpace interface 19350
CreateSubscriptionForSelection 19352
DeleteSubscriptionForSelection 19352
Export 19352
Get_SubscriptionsEnabled 19352
Import 19353
InvokeContextMenuCommand 19353
IShellFavoritesNameSpace Members 19351
IShellFavoritesNameSpace methods 19351
IShellFavoritesNameSpace properties 19354
MoveSelectionDown 19353
MoveSelectionTo 19353
MoveSelectionUp 19353
NewFolder 19354
ResetSort 19354
SetRoot 19354
SubscriptionsEnabled 19355
Synchronize 19354
IShellFavoritesNameSpaceDisp interface 19355
about IShellFavoritesNameSpaceDisp interface 19355
CreateSubscriptionForSelection 19356
DeleteSubscriptionForSelection 19356
Export 19357
Import 19357
InvokeContextMenuCommand 19357
IShellFavoritesNameSpaceDisp Members 19355
IShellFavoritesNameSpaceDisp methods 19356
IShellFavoritesNameSpaceDisp properties 19359
MoveSelectionDown 19357
MoveSelectionTo 19358
MoveSelectionUp 19358
NewFolder 19358
ResetSort 19358
SetRoot 19358
SubscriptionsEnabled 19359
Synchronize 19359
IShellNameSpace interface 19359
about IShellNameSpace interface 19359
Columns 19366
CountViewTypes 19367
Depth 19367
EnumOptions 19367
Expand 19361
Flags 19367
Get_Columns 19362
Get_CountViewTypes 19362
Get_Depth 19362

Get_EnumOptions 19362
Get_Flags 19362
Get_Mode 19363
Get_Root 19363
Get_SelectedItem 19363
Get_TVFlags 19363
IShellNameSpace Members 19360
IShellNameSpace methods 19361
IShellNameSpace properties 19366
Mode 19367
Root 19368
SelectedItem 19368
SelectedItems 19364
Set_Columns 19364
Set_Depth 19364
Set_EnumOptions 19364
Set_Flags 19364
Set_Mode 19365
Set_Root 19365
Set_SelectedItem 19365
Set_TVFlags 19365
SetViewType 19366
TVFlags 19368
UnselectAll 19366
IShellNameSpaceDisp interface 19368
about IShellNameSpaceDisp interface 19368
Columns 19374
CountViewTypes 19374
CreateSubscriptionForSelection 19370
DeleteSubscriptionForSelection 19370
Depth 19374
EnumOptions 19374
Expand 19370
Export 19370
Flags 19374
Import 19371
InvokeContextMenuCommand 19371
IShellNameSpaceDisp Members 19369
IShellNameSpaceDisp methods 19369
IShellNameSpaceDisp properties 19373
Mode 19375
MoveSelectionDown 19371
MoveSelectionTo 19371
MoveSelectionUp 19372
NewFolder 19372
ResetSort 19372
Root 19375
SelectedItem 19375
SelectedItems 19372
SetRoot 19372
SetViewType 19373
SubscriptionsEnabled 19375
Synchronize 19373
TVFlags 19376
UnselectAll 19373
IShellUIHelper interface 19376
about IShellUIHelper interface 19376
AddChannel 19377
AddDesktopComponent 19377
AddFavorite 19377
AutoCompleteAttach 19378
AutoCompleteSaveForm 19378
AutoScan 19378
ImportExportFavorites 19378
IShellUIHelper Members 19376
IShellUIHelper methods 19377
IsSubscribed 19379
NavigateAndFind 19379
RefreshOfflineDesktop 19379
ResetFirstBootMode 19379
ResetSafeMode 19379
ShowBrowserUI 19380
IShellUIHelperDisp interface 19380
about IShellUIHelperDisp interface 19380
AddChannel 19381
AddDesktopComponent 19381
AddFavorite 19382
AutoCompleteAttach 19382

AutoCompleteSaveForm 19382
AutoScan 19382
ImportExportFavorites 19382
IShellUIHelperDisp Members 19380
IShellUIHelperDisp methods 19381
IsSubscribed 19383
NavigateAndFind 19383
RefreshOfflineDesktop 19383
ResetFirstBootMode 19383
ResetSafeMode 19384
ShowBrowserUI 19384
IShellWindows interface 19384
_NewEnum 19385
about IShellWindows interface 19384
Count 19388
FindWindowSW 19385
Get_Count 19386
IShellWindows Members 19384
IShellWindows methods 19385
IShellWindows properties 19388
Item 19386
OnActivated 19386
OnCreated 19386
OnNavigate 19387
ProcessAttachDetach 19387
Register 19387
RegisterPending 19387
Revoke 19387
IShellWindowsDisp interface 19388
_NewEnum 19389
about IShellWindowsDisp interface 19388
Count 19392
FindWindowSW 19389
IShellWindowsDisp Members 19388
IShellWindowsDisp methods 19389
IShellWindowsDisp properties 19391
Item 19390
OnActivated 19390
OnCreated 19390
OnNavigate 19390
ProcessAttachDetach 19391
Register 19391
RegisterPending 19391
Revoke 19391
IShowReferenceProperty interface 11218
about IShowReferenceProperty interface 11218
IShowReferenceProperty Members 11219
IShowReferenceProperty methods 11219
ShowReferenceProperty 11219
iSingleInstance enumeration member 6484
ISkeleton interface 18205
about ISkeleton interface 18205
GetImplementation 18206
GetReplyBuffer 18206
ISkeleton Members 18205
ISkeleton methods 18206
ISkeletonObject interface 18206
about ISkeletonObject interface 18206
Execute 18207
GetImplementation 18207
GetSkeleton 18207
ISkeletonObject Members 18207
ISkeletonObject methods 18207
isNone enumeration member 5454
ISOAPHeaders interface 15613
about ISOAPHeaders interface 15613
Get 15615
Get method 15614
GetOwnsSentHeaders 15615
ISOAPHeaders Members 15614
ISOAPHeaders methods 15614
ISOAPHeaders properties 15617
OwnsSentHeaders 15617
Send 15616
SendAt 15616
SendCount 15617
SetOwnsSentHeaders 15617
ISpecialWinHelpViewer interface 15031

about ISpecialWinHelpViewer interface 15031
CallWinHelp 15032
ISpecialWinHelpViewer Members 15031
ISpecialWinHelpViewer methods 15032
isSelected enumeration member 5454
IStatusField interface 17075
about IStatusField interface 17075
IStreamLoader interface 15344
about IStreamLoader interface 15344
GetPassword 15345
GetProxy 15346
GetTimeout 15346
GetUserName 15346
IStreamLoader Members 15344
IStreamLoader methods 15345
IStreamLoader properties 15348
Load 15346
Password 15348
Proxy 15349
SetPassword 15347
SetProxy 15347
SetTimeout 15348
SetUserName 15348
Timeout 15349
UserName 15349
IStreamPersist interface 3207
about IStreamPersist interface 3207
IStreamPersist Members 3207
IStreamPersist methods 3207
LoadFromStream 3208
SaveToStream 3208
IStringsAdapter interface 3208
about IStringsAdapter interface 3208
IStringsAdapter Members 3209
IStringsAdapter methods 3209
ReferenceStrings 3209
ReleaseStrings 3209
IStringTokenizer interface 15350
about IStringTokenizer interface 15350
countTokens 15352
getTokenCounts 15351
hasMoreTokens 15351
IStringTokenizer Members 15350
IStringTokenizer methods 15351
IStringTokenizer properties 15352
nextToken 15351
IStub interface 18208
about IStub interface 18208
CreateRequest 18209
Dispatch 18209
GetInterface 18209
Hash 18209
Invoke 18210
IsA 18210
IStub Members 18208
IStub methods 18208
NonExistent 18210
RepositoryID 18210
SetPrincipal 18211
IStubObject interface 18211
about IStubObject interface 18211
GetStub 18212
IStubObject Members 18211
IStubObject methods 18211
itEnd enumeration member 24593
itImage enumeration member 15395
itMask enumeration member 15395
itNext enumeration member 24593
ITopLevelWebComponent interface 24782
about ITopLevelWebComponent interface 24782
ITransactionContext interface 17288
Abort 17289
about ITransactionContext interface 17288
Commit 17289
CreateInstance 17289
ITransactionContext Members 17288
ITransactionContext methods 17288
ITransactionContextEx interface 17289

Abort 17290
about ITransactionContextEx interface 17289
Commit 17290
CreateInstance 17290
ITransactionContextEx Members 17290
ITransactionContextEx methods 17290
ITransport interface 18995
about ITransport interface 18995
Connected 18998
GetConnected 18996
GetWaitEvent 18996
ITransport Members 18995
ITransport methods 18996
ITransport properties 18998
Receive 18997
Send 18997
SetConnected 18997
itStart enumeration member 24593
ITypeCode interface 18212
about ITypeCode interface 18212
ContentType 18213
DefaultIndex 18213
DiscriminatorType 18213
Equal 18214
Id 18214
ITypeCode Members 18212
ITypeCode methods 18213
Kind 18214
Length 18214
MemberCount 18215
MemberLabel 18215
MemberName 18215
MemberType 18215
Name 18215
Parameter 18216
ParameterCount 18216
ITypes interface 25898
about ITypes interface 25898
GetSchemaDefs 25900
ITypes Members 25899
ITypes methods 25900
ITypes properties 25900
SchemaDefs 25900
IUpdateValue interface 24583
about IUpdateValue interface 24583
IUpdateValue Members 24583
IUpdateValue methods 24584
UpdateValue 24584
IValidateField interface 17076
about IValidateField interface 17076
IValidateField Members 17076
IValidateField methods 17076
ValidateField 17076
IValidateFields interface 17077
about IValidateFields interface 17077
EnableValidateFields 17078
GetEnableValidateFields 17078
IValidateFields Members 17077
IValidateFields methods 17077
IValidateFields properties 17078
SetEnableValidateFields 17078
ValidateFields 17078
IValidateValue interface 24584
about IValidateValue interface 24584
IValidateValue Members 24584
IValidateValue methods 24585
ValidateValue 24585
IValuesListAdapter interface 19740
about IValuesListAdapter interface 19740
GetListImage 19741
GetListImageOfValue 19742
GetListName 19742
GetListNameOfValue 19742
GetListValue 19743
IValuesListAdapter Members 19741
IValuesListAdapter methods 19741
IVarInstanceReference interface 23614
about IVarInstanceReference interface 23614

GetInstance 23615
IVarInstanceReference Members 23614
IVarInstanceReference methods 23614
IVarInvokeable interface 23615
about IVarInvokeable interface 23615
DoFunction 23616
DoProcedure 23616
GetProperty 23616
IVarInvokeable Members 23615
IVarInvokeable methods 23615
SetProperty 23616
IVarStreamable interface 3210
about IVarStreamable interface 3210
IVarStreamable Members 3210
IVarStreamable methods 3210
StreamIn 3211
StreamOut 3211
IVCLComObject interface 3211
about IVCLComObject interface 3211
FreeOnRelease 3212
GetIDsOfNames 3212
GetTypeInfo 3213
GetTypeInfoCount 3213
Invoke 3213
IVCLComObject Members 3212
IVCLComObject methods 3212
SafeCallException 3213
IVCLFrameForm interface 18009
about IVCLFrameForm interface 18009
AddContainer 18010
ClearBorderSpace 18010
Form 18010
IVCLFrameForm Members 18009
IVCLFrameForm methods 18009
RemoveContainer 18010
IWebActionsList interface 19743
about IWebActionsList interface 19743
FindAction 19744
GetActionsList 19744
GetVisibleActions 19745
IsActionInUse 19745
IWebActionsList Members 19744
IWebActionsList methods 19744
IWebActionsList properties 19745
VisibleActions 19745
IWebActionsSubcomponent interface 24585
about IWebActionsSubcomponent interface 24585
IWebApplicationInfo interface 19745
about IWebApplicationInfo interface 19745
ApplicationAdapter 19747
ApplicationTitle 19748
GetApplicationAdapter 19747
GetApplicationTitle 19747
IWebApplicationInfo Members 19746
IWebApplicationInfo methods 19746
IWebApplicationInfo properties 19747
IWebAppServices interface 15227
about IWebAppServices interface 15227
ExceptionHandler 15230
FinishContext 15228
GetExceptionHandler 15229
HandleRequest 15229
InitContext 15229
IWebAppServices Members 15228
IWebAppServices methods 15228
IWebAppServices properties 15230
IWebBrowser interface 19392
about IWebBrowser interface 19392
Application 19400
Busy 19400
Container 19400
Document 19400
Get_Application 19394
Get_Busy 19394
Get_Container 19394
Get_Document 19394
Get_Height 19395
Get_Left 19395

Get_LocationName 19395
Get_LocationURL 19395
Get_Parent 19396
Get_Top 19396
Get_TopLevelContainer 19396
Get_type_ 19396
Get_Width 19396
GoBack 19397
GoForward 19397
GoHome 19397
GoSearch 19397
Height 19401
IWebBrowser Members 19392
IWebBrowser methods 19393
IWebBrowser properties 19399
Left 19401
LocationName 19401
LocationURL 19401
Navigate 19398
Parent 19401
Refresh 19398
Refresh2 19398
Set_Height 19398
Set_Left 19398
Set_Top 19399
Set_Width 19399
Stop 19399
Top 19402
TopLevelContainer 19402
type_ 19402
Width 19402
IWebBrowser2 interface 19403
about IWebBrowser2 interface 19403
AddressBar 19410
ExecWB 19405
Get_AddressBar 19406
Get_Offline 19406
Get_ReadyState 19406
Get_RegisterAsBrowser 19406
Get_RegisterAsDropTarget 19407
Get_Resizable 19407
Get_Silent 19407
Get_TheaterMode 19407
IWebBrowser2 Members 19403
IWebBrowser2 methods 19405
IWebBrowser2 properties 19410
Navigate2 19407
Offline 19410
QueryStatusWB 19408
ReadyState 19410
RegisterAsBrowser 19411
RegisterAsDropTarget 19411
Resizable 19411
Set_AddressBar 19408
Set_Offline 19408
Set_RegisterAsBrowser 19408
Set_RegisterAsDropTarget 19409
Set_Resizable 19409
Set_Silent 19409
Set_TheaterMode 19409
ShowBrowserBar 19409
Silent 19411
TheaterMode 19412
IWebBrowser2Disp interface 19412
about IWebBrowser2Disp interface 19412
AddressBar 19418
Application 19418
Busy 19418
ClientToWindow 19414
Container 19419
Document 19419
ExecWB 19414
FullName 19419
FullScreen 19419
GetProperty 19414
GoBack 19414
GoForward 19415
GoHome 19415

GoSearch 19415
Height 19419
HWnd 19420
IWebBrowser2Disp Members 19412
IWebBrowser2Disp methods 19413
IWebBrowser2Disp properties 19417
Left 19420
LocationName 19420
LocationURL 19420
MenuBar 19421
Name 19421
Navigate 19415
Navigate2 19415
Offline 19421
Parent 19421
Path 19421
PutProperty 19416
QueryStatusWB 19416
Quit 19416
ReadyState 19422
Refresh 19416
Refresh2 19417
RegisterAsBrowser 19422
RegisterAsDropTarget 19422
Resizable 19422
ShowBrowserBar 19417
Silent 19423
StatusBar 19423
StatusText 19423
Stop 19417
TheaterMode 19423
ToolBar 19423
Top 19424
TopLevelContainer 19424
type_ 19424
Visible 19424
Width 19425
IWebBrowserApp interface 19425
about IWebBrowserApp interface 19425
ClientToWindow 19427
FullName 19432
FullScreen 19432
Get_FullName 19427
Get_FullScreen 19428
Get_HWnd 19428
Get_MenuBar 19428
Get_Name 19428
Get_Path 19429
Get_StatusBar 19429
Get_StatusText 19429
Get_ToolBar 19429
Get_Visible 19429
GetProperty 19430
HWnd 19432
IWebBrowserApp Members 19425
IWebBrowserApp methods 19427
IWebBrowserApp properties 19432
MenuBar 19433
Name 19433
Path 19433
PutProperty 19430
Quit 19430
Set_FullScreen 19430
Set_MenuBar 19431
Set_StatusBar 19431
Set_StatusText 19431
Set_ToolBar 19431
Set_Visible 19431
StatusBar 19433
StatusText 19434
ToolBar 19434
Visible 19434
IWebBrowserAppDisp interface 19434
about IWebBrowserAppDisp interface 19434
Application 19439
Busy 19439
ClientToWindow 19436
Container 19439

Document 19440
FullName 19440
FullScreen 19440
GetProperty 19436
GoBack 19436
GoForward 19436
GoHome 19437
GoSearch 19437
Height 19440
HWnd 19441
IWebBrowserAppDisp Members 19435
IWebBrowserAppDisp methods 19435
IWebBrowserAppDisp properties 19438
Left 19441
LocationName 19441
LocationURL 19441
MenuBar 19441
Name 19442
Navigate 19437
Parent 19442
Path 19442
PutProperty 19437
Quit 19438
Refresh 19438
Refresh2 19438
StatusBar 19442
StatusText 19443
Stop 19438
ToolBar 19443
Top 19443
TopLevelContainer 19443
type_ 19443
Visible 19444
Width 19444
IWebBrowserDisp interface 19444
about IWebBrowserDisp interface 19444
Application 19448
Busy 19448
Container 19448
Document 19448
GoBack 19445
GoForward 19446
GoHome 19446
GoSearch 19446
Height 19448
IWebBrowserDisp Members 19444
IWebBrowserDisp methods 19445
IWebBrowserDisp properties 19447
Left 19449
LocationName 19449
LocationURL 19449
Navigate 19446
Parent 19449
Refresh 19446
Refresh2 19447
Stop 19447
Top 19450
TopLevelContainer 19450
type_ 19450
Width 19450
IWebComponent interface 24782
about IWebComponent interface 24782
ComponentList 24785
Container 24786
GetContainer 24784
GetIndex 24784
Index 24786
IWebComponent Members 24783
IWebComponent methods 24783
IWebComponent properties 24785
SetComponentList 24784
SetContainer 24784
SetIndex 24785
IWebComponentCommands interface 24786
about IWebComponentCommands interface 24786
ExecuteVerb 24787
GetVerb 24787
GetVerbCount 24788

IWebComponentCommands Members 24787
IWebComponentCommands methods 24787
IWebComponentContainer interface 24788
about IWebComponentContainer interface 24788
ComponentCount 24789
Components 24790
GetComponent 24789
GetComponentCount 24789
IWebComponentContainer Members 24788
IWebComponentContainer methods 24789
IWebComponentContainer properties 24789
IWebComponentEditor interface 24790
about IWebComponentEditor interface 24790
CanAddClass 24791
IWebComponentEditor Members 24790
IWebComponentEditor methods 24791
IWebComponentEditorHelper interface 24791
about IWebComponentEditorHelper interface 24791
CanAddClassHelper 24792
IWebComponentEditorHelper Members 24792
IWebComponentEditorHelper methods 24792
IWebContainerSubComponent interface 19748
about IWebContainerSubComponent interface 19748
IWebContent interface 24792
about IWebContent interface 24792
Content 24793
IWebContent Members 24793
IWebContent methods 24793
IWebDataFields interface 19749
about IWebDataFields interface 19749
FindField 19749
GetFieldsList 19750
GetVisibleFields 19750
IsFieldInUse 19750
IWebDataFields Members 19749
IWebDataFields methods 19749
IWebDataFields properties 19750
VisibleFields 19751
IWebDispatch interface 15230
about IWebDispatch interface 15230
DispatchEnabled 15231
DispatchMask 15232
DispatchMethodType 15232
DispatchRequest 15232
Enabled 15232
IWebDispatch Members 15231
IWebDispatch methods 15231
IWebDispatch properties 15232
Mask 15233
MethodType 15233
IWebDispatchActions interface 15233
about IWebDispatchActions interface 15233
IWebDispatcherAccess interface 15234
about IWebDispatcherAccess interface 15234
IWebDispatcherAccess Members 15235
IWebDispatcherAccess methods 15235
Request 15235
Response 15235
IWebDisplay interface 17079
about IWebDisplay interface 17079
GetIsMultipleRecordView 17079
IsMultipleRecordView 17080
IWebDisplay Members 17079
IWebDisplay methods 17079
IWebDisplay properties 17080
IWebDisplayReference interface 17080
about IWebDisplayReference interface 17080
GetWebDisplayComponent 17081
IWebDisplayReference Members 17080
IWebDisplayReference methods 17081
IWebDisplayReference properties 17081
WebDisplayComponent 17081
IWebEnabled interface 19751
about IWebEnabled interface 19751
IWebEnabled Members 19751
IWebEnabled methods 19751
WebEnabled 19752
IWebEndUser interface 24672

about IWebEndUser interface 24672
DisplayName 24678
EndUserAdapter 24678
GetDisplayName 24673
GetEndUserAdapter 24674
GetLoggedIn 24674
GetLoginFormAction 24674
GetLogoutAction 24675
HasExecuteRights 24675
HasModifyRights 24676
HasViewRights 24676
IWebEndUser Members 24673
IWebEndUser methods 24673
IWebEndUser properties 24678
LoggedIn 24678
LogIn 24677
LoginFormAction 24679
LogoutAction 24680
ProduceLoginPage 24677
IWebExceptionHandler interface 15236
about IWebExceptionHandler interface 15236
HandleException 15236
IWebExceptionHandler Members 15236
IWebExceptionHandler methods 15236
IWebFieldsSubcomponent interface 24585
about IWebFieldsSubcomponent interface 24585
IWebFormComponentContainer interface 19752
about IWebFormComponentContainer interface 19752
Count 19753
GetCount 19753
GetItem 19753
Items 19754
IWebFormComponentContainer Members 19752
IWebFormComponentContainer methods 19752
IWebFormComponentContainer properties 19753
IWebGetActionName interface 19754
about IWebGetActionName interface 19754
ActionName 19755
GetActionName 19755
IWebGetActionName Members 19754
IWebGetActionName methods 19754
IWebGetActionName properties 19755
IWebGetDisplayLabel interface 19755
about IWebGetDisplayLabel interface 19755
GetDisplayLabel 19756
IWebGetDisplayLabel Members 19756
IWebGetDisplayLabel methods 19756
IWebGetDisplayWidth interface 19756
about IWebGetDisplayWidth interface 19756
GetDisplayWidth 19757
IWebGetDisplayWidth Members 19756
IWebGetDisplayWidth methods 19757
IWebGetFieldName interface 19757
about IWebGetFieldName interface 19757
FieldName 19758
GetFieldName 19758
IWebGetFieldName Members 19757
IWebGetFieldName methods 19758
IWebGetFieldName properties 19758
IWebGetFieldPrevValue interface 24586
about IWebGetFieldPrevValue interface 24586
GetNeedPrevValue 24587
GetPrevValue 24587
IWebGetFieldPrevValue Members 24586
IWebGetFieldPrevValue methods 24586
IWebGetFieldPrevValue properties 24587
NeedPrevValue 24587
PrevValue 24587
IWebGetFieldValue interface 19758
about IWebGetFieldValue interface 19758
GetValue 19759
IWebGetFieldValue Members 19759
IWebGetFieldValue methods 19759
IWebGetFieldValue properties 19759
Value 19760
IWebGetFieldValues interface 19760
about IWebGetFieldValues interface 19760
EndValueIterator 19761

GetCurrentValue 19761
HasValue 19761
IWebGetFieldValues Members 19760
IWebGetFieldValues methods 19760
NextValueIteration 19762
StartValueIterator 19762
IWebGetMaxLength interface 19763
about IWebGetMaxLength interface 19763
GetMaxLength 19763
IWebGetMaxLength Members 19763
IWebGetMaxLength methods 19763
IWebGetSessionIDs interface 19764
about IWebGetSessionIDs interface 19764
GetSessionIDs 19764
IWebGetSessionIDs Members 19764
IWebGetSessionIDs methods 19764
IWebImageHREF interface 19765
about IWebImageHREF interface 19765
IWebImageHREF Members 19765
IWebImageHREF methods 19765
WebImageHREF 19766
IWebInputName interface 19766
about IWebInputName interface 19766
GetInputName 19767
InputName 19767
IWebInputName Members 19766
IWebInputName methods 19767
IWebInputName properties 19767
IWebIsDefaultAction interface 19768
about IWebIsDefaultAction interface 19768
IsDefaultAction 19768
IWebIsDefaultAction Members 19768
IWebIsDefaultAction methods 19768
IWebIsDefaultField interface 19769
about IWebIsDefaultField interface 19769
IsDefaultField 19769
IWebIsDefaultField Members 19769
IWebIsDefaultField methods 19769
IWebNode interface 25430
about IWebNode interface 25430
BeforeExecute 25431
Execute 25432
Execute method 25432
GetMimeBoundary 25433
IWebNode Members 25431
IWebNode methods 25431
IWebNode properties 25433
MimeBoundary 25433
SetMimeBoundary 25433
IWebPageInfo interface 24680
about IWebPageInfo interface 24680
GetHasViewAccess 24681
GetLoginRequired 24682
GetPageDescription 24682
GetPageHREF 24682
GetPageIsPublished 24683
GetPageName 24683
GetPageTitle 24683
HasViewAccess 24684
IWebPageInfo Members 24681
IWebPageInfo methods 24681
IWebPageInfo properties 24683
LoginRequired 24684
PageDescription 24684
PageHREF 24685
PageIsPublished 24685
PageName 24686
PageTitle 24686
IWebRequestHandler interface 15237
about IWebRequestHandler interface 15237
HandleRequest 15238
IWebRequestHandler Members 15237
IWebRequestHandler methods 15237
IWebServExp interface 25559
about IWebServExp interface 25559
FindOrAddSchema 25560
GenerateXMLSchema 25560
IWebServExp Members 25559

IWebServExp methods 25560
IWebServExpAccess interface 25561
about IWebServExpAccess interface 25561
GetWebServExp 25561
IWebServExpAccess Members 25561
IWebServExpAccess methods 25561
IWebSession interface 19770
about IWebSession interface 19770
IWebSession Members 19770
IWebSession methods 19770
Terminate 19771
IWebSessionAccess interface 24975
about IWebSessionAccess interface 24975
GetSessionsService 24976
GetWebSession 24976
IWebSessionAccess Members 24975
IWebSessionAccess methods 24975
IWebSessionAttributes interface 19771
about IWebSessionAttributes interface 19771
ExpirationTime 19774
Expired 19774
GetExpirationTime 19772
GetExpired 19772
GetIsActive 19773
GetStartTime 19773
GetTerminated 19773
GetTimeoutMinutes 19773
GetTouchTime 19773
IsActive 19775
IWebSessionAttributes Members 19771
IWebSessionAttributes methods 19772
IWebSessionAttributes properties 19774
SetTimeoutMinutes 19774
StartTime 19775
Terminated 19775
TimeoutMinutes 19775
TouchTime 19775
IWebSessionID interface 19776
about IWebSessionID interface 19776
GetID 19776
ID 19777
IWebSessionID Members 19776
IWebSessionID methods 19776
IWebSessionID properties 19777
IWebSessionIDs interface 19777
about IWebSessionIDs interface 19777
GetID 19778
GetIDCount 19778
IDCount 19779
IDs 19779
IWebSessionIDs Members 19778
IWebSessionIDs methods 19778
IWebSessionIDs properties 19779
IWebSessionValues interface 19780
about IWebSessionValues interface 19780
GetItemCount 19780
GetItems 19781
GetValue 19781
IWebSessionValues Members 19780
IWebSessionValues methods 19780
IWebSessionValues properties 19781
SetValue 19781
Values 19782
IWebSetActionName interface 19782
about IWebSetActionName interface 19782
IWebSetActionName Members 19782
IWebSetActionName methods 19782
SetActionName 19783
IWebSetFieldName interface 19783
about IWebSetFieldName interface 19783
IWebSetFieldName Members 19783
IWebSetFieldName methods 19783
SetFieldName 19784
IWebUserList interface 19784
about IWebUserList interface 19784
CheckAccessRights 19785
IWebUserList Members 19784
IWebUserList methods 19785

ValidateUser 19785
IWebValueIsEqual interface 19786
about IWebValueIsEqual interface 19786
IsEqual 19787
IWebValueIsEqual Members 19786
IWebValueIsEqual methods 19786
IWebVariableName interface 15328
about IWebVariableName interface 15328
GetVariableName 15329
IWebVariableName Members 15328
IWebVariableName methods 15329
IWebVariableName properties 15329
VariableName 15329
IWebVariablesContainer interface 15329
about IWebVariablesContainer interface 15329
FindVariable 15330
FindVariableContainer 15331
GetVariable 15331
GetVariableCount 15331
IWebVariablesContainer Members 15330
IWebVariablesContainer methods 15330
IWebVariablesContainer properties 15331
VariableCount 15331
Variables 15332
IWebVariablesContainerName interface 24794
about IWebVariablesContainerName interface 24794
ContainerName 24795
GetContainerName 24794
IWebVariablesContainerName Members 24794
IWebVariablesContainerName methods 24794
IWebVariablesContainerName properties 24795
IWideProperty interface 11219
about IWideProperty interface 11219
GetEditValue 11220
GetValue 11220
IWideProperty Members 11219
IWideProperty methods 11220
SetValue 11220
IWideProperty10 interface 11221
about IWideProperty10 interface 11221
GetName 11221
GetValues 11222
IWideProperty10 Members 11221
IWideProperty10 methods 11221
IWidePropertyDescription interface 11222
about IWidePropertyDescription interface 11222
GetDescription 11223
IWidePropertyDescription Members 11222
IWidePropertyDescription methods 11222
IWSDLDocument interface 25901
about IWSDLDocument interface 25901
Definition 25904
GetDefinition 25903
IWSDLDocument Members 25901
IWSDLDocument methods 25903
IWSDLDocument properties 25904
SetDefinition 25903
IWSDLItems interface 25998
about IWSDLItems interface 25998
GetWSDLDoc 25999
GetWSDLItems 25999
IWSDLItems Members 25998
IWSDLItems methods 25998
Load 25999
IWSDLPublish interface 26042
about IWSDLPublish interface 26042
GetPortTypeList 26043
GetTypeSystemsList 26043
GetWSDLForPortType 26043
GetXSDForTypeSystem 26043
IWSDLPublish Members 26042
IWSDLPublish methods 26042
ixCaseInsensitive enumeration member 7617
ixDescending enumeration member 7617
ixExpression enumeration member 7617
IXMLDisplay interface 17082
about IXMLDisplay interface 17082
GetIsMultipleRecordView 17082

GetXMLDisplayName 17083
GetXMLRowSetName 17083
IsMultipleRecordView 17083
IXMLDisplay Members 17082
IXMLDisplay methods 17082
IXMLDisplay properties 17083
XMLDisplayName 17084
XMLRowSetName 17084
IXMLDocument interface 26382
about IXMLDocument interface 26382
Active 26403
AddChild 26385
AddChild method 26385
AsyncLoadState 26404
ChildNodes 26405
CreateElement 26386
CreateNode 26387
DocumentElement 26405
DOMDocument 26405
Encoding 26406
FileName 26406
GeneratePrefix 26388
GetActive 26388
GetAsyncLoadState 26389
GetChildNodes 26389
GetDocBinding 26389
GetDocumentElement 26390
GetDocumentNode 26390
GetDOMDocument 26390
GetEncoding 26391
GetFileName 26391
GetModified 26391
GetNodeIndentStr 26392
GetOptions 26392
GetParseOptions 26392
GetSchemaRef 26392
GetStandAlone 26393
GetVersion 26393
GetXML 26393
IsEmptyDoc 26394
IXMLDocument Members 26383
IXMLDocument methods 26384
IXMLDocument properties 26403
LoadFromFile 26394
LoadFromStream 26395
LoadFromXML 26395
LoadFromXML method 26395
Modified 26407
Node 26407
NodeIndentStr 26408
Options 26408
ParseOptions 26409
Refresh 26396
RegisterDocBinding 26396
Resync 26397
SaveToFile 26397
SaveToStream 26398
SaveToXML 26398
SaveToXML method 26398
SchemaRef 26409
SetActive 26399
SetDocumentElement 26399
SetDOMDocument 26399
SetEncoding 26400
SetFileName 26400
SetNodeIndentStr 26400
SetOnAsyncLoad 26401
SetOptions 26401
SetParseOptions 26402
SetStandAlone 26402
SetVersion 26402
SetXML 26403
StandAlone 26410
Version 26410
XML 26410
IXMLDocumentAccess interface 26228
about IXMLDocumentAccess interface 26228
DocumentObject 26229

DOMPersist 26230
GetDocumentObject 26229
GetDOMPersist 26229
IXMLDocumentAccess Members 26228
IXMLDocumentAccess methods 26228
IXMLDocumentAccess properties 26229
IXMLNode interface 26411
about IXMLNode interface 26411
AddChild 26414
AddChild method 26414
AttributeNodes 26430
Attributes 26430
ChildNodes 26431
ChildValues 26432
CloneNode 26415
Collection 26432
DeclareNamespace 26416
DOMNode 26433
FindNamespaceDecl 26416
FindNamespaceURI 26416
GetAttribute 26417
GetAttributeNodes 26417
GetAttributeNS 26418
GetChildNodes 26418
GetChildValue 26418
GetCollection 26419
GetDOMNode 26419
GetHasChildNodes 26420
GetIsTextElement 26420
GetLocalName 26420
GetNamespaceURI 26421
GetNodeName 26421
GetNodeType 26421
GetNodeValue 26422
GetOwnerDocument 26422
GetParentNode 26422
GetPrefix 26422
GetReadOnly 26423
GetText 26423
GetXML 26423
HasAttribute 26424
HasAttribute method 26424
HasChildNodes 26433
IsTextElement 26434
IXMLNode Members 26412
IXMLNode methods 26413
IXMLNode properties 26429
LocalName 26434
NamespaceURI 26435
NextSibling 26424
NodeName 26435
NodeType 26436
NodeValue 26436
Normalize 26425
OwnerDocument 26437
ParentNode 26437
Prefix 26438
PreviousSibling 26425
ReadOnly 26438
Resync 26426
SetAttribute 26426
SetAttributeNS 26426
SetChildValue 26427
SetNodeValue 26427
SetReadOnly 26428
SetText 26428
Text 26439
TransformNode 26429
TransformNode method 26428
XML 26439
IXMLNodeAccess interface 26230
about IXMLNodeAccess interface 26230
AddChild 26232
CheckTextNode 26233
ChildNodeClasses 26237
ClearDocumentRef 26233
CreateAttributeNode 26233
CreateChildNode 26233

CreateCollection 26233
DOMElement 26234
FindHostedNode 26234
GetChildNodeClasses 26234
GetHostNode 26234
GetNodeObject 26235
HasChildNode 26235
HasChildNode method 26235
HostNode 26237
InternalAddChild 26235
IXMLNodeAccess Members 26230
IXMLNodeAccess methods 26232
IXMLNodeAccess properties 26237
NestingLevel 26236
RegisterChildNode 26236
RegisterChildNodes 26236
SetCollection 26236
SetParentNode 26237
IXMLNodeCollection interface 26440
about IXMLNodeCollection interface 26440
Clear 26442
Count 26444
Delete 26442
GetCount 26443
GetNode 26443
IXMLNodeCollection Members 26440
IXMLNodeCollection methods 26442
IXMLNodeCollection properties 26444
Nodes 26445
Remove 26443
IXMLNodeList interface 26445
about IXMLNodeList interface 26445
Add 26447
BeginUpdate 26447
Clear 26448
Count 26456
Delete 26449
Delete method 26448
EndUpdate 26449
FindNode 26450
FindNode method 26450
FindSibling 26451
First 26451
Get 26452
GetCount 26452
GetNode 26452
GetUpdateCount 26453
IndexOf 26453
IndexOf method 26453
Insert 26454
IXMLNodeList Members 26446
IXMLNodeList methods 26446
IXMLNodeList properties 26456
Last 26454
Nodes 26456
Remove 26455
ReplaceNode 26455
UpdateCount 26457
IXMLSchemaDefs interface 25905
about IXMLSchemaDefs interface 25905
Add 25907
GetXMLSchemaDef 25907
IXMLSchemaDefs Members 25905
IXMLSchemaDefs methods 25906
IXMLSchemaDefs properties 25907
SchemaDef 25907
ixNonMaintained enumeration member 7617
ixPrimary enumeration member 7617
ixUnique enumeration member 7617

J

Java 15681
Java namespace 15681
Java.Sql 15682
Java.Sql namespace 15682
Classes 15682
Java.Sql.SQLException 15682
Java.Sql.SQLException.ErrorCode 15684

Java.Sql.SQLException.getErrorCode 15684
Java.Sql.SQLException.getSQLState 15684
Java.Sql.SQLException.SQLException 15683
Java.Sql.SQLException.SQLState 15684
jf24Bit enumeration member 15707
jf8Bit enumeration member 15707
jpBestQuality enumeration member 15707
jpBestSpeed enumeration member 15707
jpeg 15685
jpeg namespace 15685
Classes 15686
Structs, Records, Enums 15706
Types 15709
Variables 15710
jpeg.JPEGDefaults 15710
jpeg.JPEGDefaults variable 15710
jpeg.TJPEGData 15686
jpeg.TJPEGData.Destroy 15687
jpeg.TJPEGData.FreeHandle 15687
jpeg.TJPEGDefaults 15706
jpeg.TJPEGDefaults record 15706
jpeg.TJPEGImage 15688
jpeg.TJPEGImage.Assign 15693
jpeg.TJPEGImage.AssignTo 15694
jpeg.TJPEGImage.Bitmap 15702
jpeg.TJPEGImage.Changed 15695
jpeg.TJPEGImage.Compress 15695
jpeg.TJPEGImage.CompressionQuality 15702
jpeg.TJPEGImage.Create 15691
jpeg.TJPEGImage.Destroy 15692
jpeg.TJPEGImage.DIBNeeded 15696
jpeg.TJPEGImage.Draw 15696
jpeg.TJPEGImage.Equals 15696
jpeg.TJPEGImage.FreeBitmap 15696
jpeg.TJPEGImage.GetEmpty 15697
jpeg.TJPEGImage.GetHeight 15697
jpeg.TJPEGImage.GetPalette 15697
jpeg.TJPEGImage.GetWidth 15697
jpeg.TJPEGImage.Grayscale 15702
jpeg.TJPEGImage.JPEGNeeded 15697
jpeg.TJPEGImage.LoadFromClipboardFormat 15698
jpeg.TJPEGImage.LoadFromStream 15698
jpeg.TJPEGImage.NewBitmap 15699
jpeg.TJPEGImage.NewImage 15699
jpeg.TJPEGImage.Performance 15703
jpeg.TJPEGImage.PixelFormat 15703
jpeg.TJPEGImage.ProgressiveDisplay 15703
jpeg.TJPEGImage.ProgressiveEncoding 15704
jpeg.TJPEGImage.ReadData 15699
jpeg.TJPEGImage.ReadStream 15699
jpeg.TJPEGImage.SaveToClipboardFormat 15700
jpeg.TJPEGImage.SaveToStream 15700
jpeg.TJPEGImage.Scale 15704
jpeg.TJPEGImage.SetHeight 15700
jpeg.TJPEGImage.SetPalette 15701
jpeg.TJPEGImage.SetWidth 15701
jpeg.TJPEGImage.Smoothing 15704
jpeg.TJPEGImage.WriteData 15701
jpeg.TJPEGPerformance 15707
jpeg.TJPEGPerformance enumeration 15707
jpeg.TJPEGPixelFormat 15707
jpeg.TJPEGPixelFormat enumeration 15707
jpeg.TJPEGQualityRange 15709
jpeg.TJPEGQualityRange type 15709
jpeg.TJPEGScale 15708
jpeg.TJPEGScale enumeration 15708
jsBind enumeration member 17161
jsDB enumeration member 17161
jsDisp enumeration member 17161
jsDom enumeration member 17161
jsEighth enumeration member 15708
jsErrDisp enumeration member 17161
jsFullSize enumeration member 15708
jsHalf enumeration member 15708
jsQuarter enumeration member 15708
jsShow enumeration member 17161

K

keyAdd enumeration member 23510
keyDelete enumeration member 23510
keyEdit enumeration member 23510
keySEARCHEQ enumeration member 11707
keySEARCHGEQ enumeration member 11707
keySEARCHGT enumeration member 11707
keyUnique enumeration member 23510

L

laAnd enumeration member 3273
laCopy enumeration member 3273
laDestUnique enumeration member 3273
laOr enumeration member 3273
laSrcUnique enumeration member 3273
laXor enumeration member 3273
lbOwnerDrawFixed enumeration member 21948
lbOwnerDrawVariable enumeration member 21948
lbStandard enumeration member 21948
lbVirtual enumeration member 21948
lbVirtualOwnerDraw enumeration member 21948
LicenseColumns class 2059
about LicenseColumns class 2059
CacheSizeLimit 2060
DeploymentLimit 2060
Desktop 2060
Expired 2061
FileName 2061
FileSizeLimit 2061
LicenseColumns fields 2060
LicenseColumns Members 2059
ProductId 2061
SerialNumber 2061
SkuId 2062
UserLimit 2062
ListActns 15711
ListActns namespace 15711
Classes 15713
Structs, Records, Enums 15792
Types 15793
ListActns.TCustomListAction 15713
ListActns.TCustomListAction.Active 15721
ListActns.TCustomListAction.Count 15722
ListActns.TCustomListAction.Create 15718
ListActns.TCustomListAction.ExecuteTarget 15718
ListActns.TCustomListAction.GetCount 15719
ListActns.TCustomListAction.GetString 15719
ListActns.TCustomListAction.HandlesTarget 15719
ListActns.TCustomListAction.Images 15722
ListActns.TCustomListAction.ItemIndex 15722
ListActns.TCustomListAction.Loaded 15720
ListActns.TCustomListAction.Loading 15723
ListActns.TCustomListAction.OnGetItemCount 15723
ListActns.TCustomListAction.OnItemSelected 15724
ListActns.TCustomListAction.SetString 15720
ListActns.TCustomListAction.Strings 15723
ListActns.TCustomStaticListAction 15724
ListActns.TCustomStaticListAction.Active 15731
ListActns.TCustomStaticListAction.Caption 15732
ListActns.TCustomStaticListAction.Count 15732
ListActns.TCustomStaticListAction.Create 15729
ListActns.TCustomStaticListAction.Destroy 15729
ListActns.TCustomStaticListAction.Enabled 15733
ListActns.TCustomStaticListAction.GetCount 15730
ListActns.TCustomStaticListAction.GetItem 15730
ListActns.TCustomStaticListAction.GetItemClass 15730
ListActns.TCustomStaticListAction.GetString 15730
ListActns.TCustomStaticListAction.HelpContext 15733
ListActns.TCustomStaticListAction.Hint 15733
ListActns.TCustomStaticListAction.Images 15734
ListActns.TCustomStaticListAction.ItemIndex 15734
ListActns.TCustomStaticListAction.Items 15734
ListActns.TCustomStaticListAction.OnGetItem 15736
ListActns.TCustomStaticListAction.OnHint 15736
ListActns.TCustomStaticListAction.OnItemSelected 15736
ListActns.TCustomStaticListAction.OnUpdate 15737

ListActns.TCustomStaticListAction.SecondaryShortCuts 15735
ListActns.TCustomStaticListAction.SetString 15731
ListActns.TCustomStaticListAction.ShortCut 15735
ListActns.TCustomStaticListAction.Visible 15735
ListActns.TCustomVirtualListAction 15738
ListActns.TCustomVirtualListAction.Count 15744
ListActns.TCustomVirtualListAction.GetItem 15743
ListActns.TCustomVirtualListAction.GetString 15743
ListActns.TCustomVirtualListAction.OnGetItem 15744
ListActns.TGetItemCountEvent 15793
ListActns.TGetItemCountEvent type 15793
ListActns.TGetItemEvent 15793
ListActns.TGetItemEvent type 15793
ListActns.TGetVirtualItemEvent 15794
ListActns.TGetVirtualItemEvent type 15794
ListActns.TItemSelectedEvent 15794
ListActns.TItemSelectedEvent type 15794
ListActns.TListActionLink 15744
ListActns.TListActionLink.AddItem 15748
ListActns.TListActionLink.IsActiveLinked 15748
ListActns.TListActionLink.IsImagesLinked 15748
ListActns.TListActionLink.RefreshControl 15748
ListActns.TListActionLink.SetAction 15749
ListActns.TListActionLink.SetActive 15749
ListActns.TListActionLink.SetImages 15749
ListActns.TListActionLink.SetItemIndex 15749
ListActns.TListCompareEvent 15794
ListActns.TListCompareEvent type 15794
ListActns.TListControlItem 15750
ListActns.TListControlItem.Assign 15753
ListActns.TListControlItem.Caption 15756
ListActns.TListControlItem.Changed 15754
ListActns.TListControlItem.Create 15752
ListActns.TListControlItem.Data 15756
ListActns.TListControlItem.FCaption 15752
ListActns.TListControlItem.FData 15752
ListActns.TListControlItem.FImageIndex 15753
ListActns.TListControlItem.GetDisplayName 15754
ListActns.TListControlItem.ImageIndex 15757
ListActns.TListControlItem.SetCaption 15755
ListActns.TListControlItem.SetData 15755
ListActns.TListControlItem.SetImageIndex 15755
ListActns.TListControlItemClass 15795
ListActns.TListControlItemClass type 15795
ListActns.TListControlItems 15757
ListActns.TListControlItems.Add 15760
ListActns.TListControlItems.CaseSensitive 15762
ListActns.TListControlItems.CompareItems 15761
ListActns.TListControlItems.Create 15760
ListActns.TListControlItems.CustomSort 15761
ListActns.TListControlItems.Items 15763
ListActns.TListControlItems.OnCompare 15764
ListActns.TListControlItems.Sort 15762
ListActns.TListControlItems.SortType 15764
ListActns.TListItemsCompare 15795
ListActns.TListItemsCompare type 15795
ListActns.TListItemsSortType 15792
ListActns.TListItemsSortType enumeration 15792
ListActns.TStaticListAction 15765
ListActns.TStaticListAction.Active 15770
ListActns.TStaticListAction.Caption 15771
ListActns.TStaticListAction.Enabled 15771
ListActns.TStaticListAction.HelpContext 15771
ListActns.TStaticListAction.Hint 15772
ListActns.TStaticListAction.Images 15772
ListActns.TStaticListAction.ItemIndex 15773
ListActns.TStaticListAction.Items 15773
ListActns.TStaticListAction.OnGetItem 15775
ListActns.TStaticListAction.OnHint 15775
ListActns.TStaticListAction.OnItemSelected 15775
ListActns.TStaticListAction.OnUpdate 15776
ListActns.TStaticListAction.SecondaryShortCuts 15773
ListActns.TStaticListAction.ShortCut 15774
ListActns.TStaticListAction.Visible 15774
ListActns.TStaticListItems 15777
ListActns.TStaticListItems.Notify 15779
ListActns.TStaticListItems.Update 15780
ListActns.TVirtualListAction 15780

ListActns.TVirtualListAction.Active 15786
ListActns.TVirtualListAction.Caption 15786
ListActns.TVirtualListAction.Enabled 15787
ListActns.TVirtualListAction.HelpContext 15787
ListActns.TVirtualListAction.Hint 15788
ListActns.TVirtualListAction.Images 15788
ListActns.TVirtualListAction.ItemIndex 15788
ListActns.TVirtualListAction.OnGetItem 15790
ListActns.TVirtualListAction.OnGetItemCount 15790
ListActns.TVirtualListAction.OnHint 15791
ListActns.TVirtualListAction.OnItemSelected 15791
ListActns.TVirtualListAction.SecondaryShortCuts 15789
ListActns.TVirtualListAction.ShortCut 15789
ListActns.TVirtualListAction.Visible 15789
lnAdded enumeration member 3274
lnDeleted enumeration member 3274
lnExtracted enumeration member 3274
loCaseInsensitive enumeration member 7618
LockColumns class 2062
about LockColumns class 2062
CompMask 2063
Granted 2064
LockColumns fields 2063
LockColumns Members 2063
LockMask 2064
Name 2064
Row 2064
RowLock 2064
Shared 2065
TableLock 2065
Type 2065
User 2065
LogFilter class 2065
about LogFilter class 2065
Connect 2067
Errors 2067
Execute 2067
Init 2068
LockErrors 2068
LogFilter fields 2067
LogFilter Members 2066
Mirror 2068
Prepare 2068
SetParameter 2068
System 2069
Transactions 2069
Xa 2069
loInvariantLocale enumeration member 23003
loPartialKey enumeration member 7618
loUserLocale enumeration member 23003
lpAbove enumeration member 13269
lpBelow enumeration member 13269
lpLeft enumeration member 13269
lposAbove enumeration member 24799
lposBelow enumeration member 24799
lposLeft enumeration member 24799
lposRight enumeration member 24799
lpRight enumeration member 13269
lrDefaultColor enumeration member 15395
lrDefaultSize enumeration member 15395
lrFromFile enumeration member 15395
lrMap3DColors enumeration member 15395
lrMonoChrome enumeration member 15395
lrTransparent enumeration member 15395
lsButtons enumeration member 17091, 25320
lsCursor enumeration member 17841
lsFields enumeration member 17091, 25320
lsIdle enumeration member 18887
lsLookupAddress enumeration member 18887
lsLookupService enumeration member 18887
lsShowProgress enumeration member 17841
lsSparsing enumeration member 17841
lsUnknown enumeration member 17091, 25320
ltBatchOptimistic enumeration member 1289
ltOptimistic enumeration member 1289
ltPessimistic enumeration member 1289
ltReadLock enumeration member 10254
ltReadOnly enumeration member 1289

ltUnspecified enumeration member 1289
ltWriteLock enumeration member 10254

M

maActivate enumeration member 6287
maActivateAndEat enumeration member 6287
maAutomatic enumeration member 16200
maBottomToTop enumeration member 16199
maDefault enumeration member 6287
maLeftToRight enumeration member 16199
maManual enumeration member 16200
maNoActivate enumeration member 6287
maNoActivateAndEat enumeration member 6287
maNone enumeration member 16199
maParent enumeration member 16200
maRightToLeft enumeration member 16199
Mask 15797
Mask namespace 15797
    Classes 15798
    Types 15895
Mask.EDBEditError 15798
Mask.TCustomMaskEdit 15798
Mask.TCustomMaskEdit.CheckCursor 15811
Mask.TCustomMaskEdit.Clear 15812
Mask.TCustomMaskEdit.Create 15810
Mask.TCustomMaskEdit.EditCanModify 15812
Mask.TCustomMaskEdit.EditMask 15821
Mask.TCustomMaskEdit.EditText 15821
Mask.TCustomMaskEdit.GetFirstEditChar 15813
Mask.TCustomMaskEdit.GetLastEditChar 15813
Mask.TCustomMaskEdit.GetMaxChars 15814
Mask.TCustomMaskEdit.GetNextEditChar 15814
Mask.TCustomMaskEdit.GetPriorEditChar 15815
Mask.TCustomMaskEdit.GetSel 15815
Mask.TCustomMaskEdit.GetTextLen 15816
Mask.TCustomMaskEdit.IsMasked 15822
Mask.TCustomMaskEdit.KeyDown 15816
Mask.TCustomMaskEdit.KeyPress 15816
Mask.TCustomMaskEdit.KeyUp 15817
Mask.TCustomMaskEdit.MaskState 15822
Mask.TCustomMaskEdit.MaxLength 15822
Mask.TCustomMaskEdit.ReformatText 15817
Mask.TCustomMaskEdit.Reset 15818
Mask.TCustomMaskEdit.SetCursor 15818
Mask.TCustomMaskEdit.SetSel 15819
Mask.TCustomMaskEdit.Text 15823
Mask.TCustomMaskEdit.Validate 15819
Mask.TCustomMaskEdit.ValidateEdit 15819
Mask.TCustomMaskEdit.ValidateError 15820
Mask.TMaskEdit 15824
Mask.TMaskEdit.Align 15837
Mask.TMaskEdit.Anchors 15840
Mask.TMaskEdit.AutoSelect 15841
Mask.TMaskEdit.AutoSize 15841
Mask.TMaskEdit.BevelEdges 15842
Mask.TMaskEdit.BevelInner 15843
Mask.TMaskEdit.BevelKind 15843
Mask.TMaskEdit.BevelOuter 15843
Mask.TMaskEdit.BevelWidth 15844
Mask.TMaskEdit.BiDiMode 15844
Mask.TMaskEdit.BorderStyle 15845
Mask.TMaskEdit.CharCase 15845
Mask.TMaskEdit.Color 15845
Mask.TMaskEdit.Constraints 15846
Mask.TMaskEdit.Ctl3D 15847
Mask.TMaskEdit.DragCursor 15847
Mask.TMaskEdit.DragKind 15848
Mask.TMaskEdit.DragMode 15848
Mask.TMaskEdit.EditMask 15850
Mask.TMaskEdit.Enabled 15850
Mask.TMaskEdit.Font 15851
Mask.TMaskEdit.ImeMode 15852
Mask.TMaskEdit.ImeName 15852
Mask.TMaskEdit.MaxLength 15853
Mask.TMaskEdit.OnChange 15866
Mask.TMaskEdit.OnClick 15866
Mask.TMaskEdit.OnDblClick 15868
Mask.TMaskEdit.OnDragDrop 15869

Mask.TMaskEdit.OnDragOver 15871
Mask.TMaskEdit.OnEndDock 15873
Mask.TMaskEdit.OnEndDrag 15874
Mask.TMaskEdit.OnEnter 15875
Mask.TMaskEdit.OnExit 15877
Mask.TMaskEdit.OnKeyDown 15879
Mask.TMaskEdit.OnKeyPress 15882
Mask.TMaskEdit.OnKeyUp 15883
Mask.TMaskEdit.OnMouseActivate 15884
Mask.TMaskEdit.OnMouseDown 15885
Mask.TMaskEdit.OnMouseEnter 15888
Mask.TMaskEdit.OnMouseLeave 15888
Mask.TMaskEdit.OnMouseMove 15888
Mask.TMaskEdit.OnMouseUp 15890
Mask.TMaskEdit.OnStartDock 15893
Mask.TMaskEdit.OnStartDrag 15893
Mask.TMaskEdit.ParentBiDiMode 15853
Mask.TMaskEdit.ParentColor 15854
Mask.TMaskEdit.ParentCtl3D 15854
Mask.TMaskEdit.ParentFont 15854
Mask.TMaskEdit.ParentShowHint 15855
Mask.TMaskEdit.PasswordChar 15856
Mask.TMaskEdit.PopupMenu 15856
Mask.TMaskEdit.ReadOnly 15861
Mask.TMaskEdit.ShowHint 15861
Mask.TMaskEdit.TabOrder 15862
Mask.TMaskEdit.TabStop 15863
Mask.TMaskEdit.Text 15863
Mask.TMaskEdit.Visible 15864
Mask.TMaskedState 15895
Mask.TMaskedState type 15895
Masks 15897
Masks namespace 15897
    Classes 15898
    Functions 15902
Masks.EMaskException 15898
Masks.MatchesMask 15902
Masks.MatchesMask function 15902
Masks.TMask 15898
Masks.TMask.Create 15899
Masks.TMask.Destroy 15900
Masks.TMask.Matches 15900
MaskUtils 15903
MaskUtils namespace 15903
    Constants 15917
    Functions 15905
    Structs, Records, Enums 15911
    Types 15912
    Variables 15915
MaskUtils.DefaultBlank 15915
MaskUtils.DefaultBlank variable 15915
MaskUtils.FormatMaskText 15905
MaskUtils.FormatMaskText function 15905
MaskUtils.IsLiteralChar 15907
MaskUtils.IsLiteralChar function 15907
MaskUtils.MaskDoFormatText 15907
MaskUtils.MaskDoFormatText function 15907
MaskUtils.MaskFieldSeparator 15915
MaskUtils.MaskFieldSeparator variable 15915
MaskUtils.MaskGetCharType 15907
MaskUtils.MaskGetCharType function 15907
MaskUtils.MaskGetCurrentDirectives 15908
MaskUtils.MaskGetCurrentDirectives function 15908
MaskUtils.MaskGetFldSeparator 15908
MaskUtils.MaskGetFldSeparator function 15908
MaskUtils.MaskGetMaskBlank 15908
MaskUtils.MaskGetMaskBlank function 15908
MaskUtils.MaskGetMaskSave 15909
MaskUtils.MaskGetMaskSave function 15909
MaskUtils.MaskIntlLiteralToChar 15909
MaskUtils.MaskIntlLiteralToChar function 15909
MaskUtils.MaskNoSave 15915
MaskUtils.MaskNoSave variable 15915
MaskUtils.MaskOffsetToOffset 15909
MaskUtils.MaskOffsetToOffset function 15909
MaskUtils.MaskOffsetToWideOffset 15910
MaskUtils.MaskOffsetToWideOffset function 15910
MaskUtils.mDirLiteral 15917

MaskUtils.mDirLiteral constant 15917
MaskUtils.mDirLowerCase 15917
MaskUtils.mDirLowerCase constant 15917
MaskUtils.mDirReverse 15918
MaskUtils.mDirReverse constant 15918
MaskUtils.mDirUpperCase 15918
MaskUtils.mDirUpperCase constant 15918
MaskUtils.mMskAlpha 15918
MaskUtils.mMskAlpha constant 15918
MaskUtils.mMskAlphaNum 15919
MaskUtils.mMskAlphaNum constant 15919
MaskUtils.mMskAlphaNumOpt 15919
MaskUtils.mMskAlphaNumOpt constant 15919
MaskUtils.mMskAlphaOpt 15919
MaskUtils.mMskAlphaOpt constant 15919
MaskUtils.mMskAscii 15920
MaskUtils.mMskAscii constant 15920
MaskUtils.mMskAsciiOpt 15920
MaskUtils.mMskAsciiOpt constant 15920
MaskUtils.mMskDateSeparator 15920
MaskUtils.mMskDateSeparator constant 15920
MaskUtils.mMskNumeric 15921
MaskUtils.mMskNumeric constant 15921
MaskUtils.mMskNumericOpt 15921
MaskUtils.mMskNumericOpt constant 15921
MaskUtils.mMskNumSymOpt 15921
MaskUtils.mMskNumSymOpt constant 15921
MaskUtils.mMskTimeSeparator 15922
MaskUtils.mMskTimeSeparator constant 15922
MaskUtils.OffsetToMaskOffset 15910
MaskUtils.OffsetToMaskOffset function 15910
MaskUtils.PadInputLiterals 15910
MaskUtils.PadInputLiterals function 15910
MaskUtils.TEditMask 15912
MaskUtils.TEditMask type 15912
MaskUtils.TMaskCharType 15911
MaskUtils.TMaskCharType enumeration 15911
MaskUtils.TMaskDirectives 15913
MaskUtils.TMaskDirectives type 15913
MaskUtils.TMaskedText 15914
MaskUtils.TMaskedText type 15914
Math 15923
Math namespace 15923
    Classes 15927
    Constants 15988
    Functions 15929
    Structs, Records, Enums 15981
    Types 15985
Math.ArcCos 15931
Math.ArcCos function 15931
Math.ArcCosh 15932
Math.ArcCosh function 15932
Math.ArcCot 15932
Math.ArcCot function 15932
Math.ArcCotH 15933
Math.ArcCotH function 15933
Math.ArcCsc 15933
Math.ArcCsc function 15933
Math.ArcCscH 15934
Math.ArcCscH function 15934
Math.ArcSec 15934
Math.ArcSec function 15934
Math.ArcSecH 15934
Math.ArcSecH function 15934
Math.ArcSin 15935
Math.ArcSin function 15935
Math.ArcSinh 15935
Math.ArcSinh function 15935
Math.ArcTan2 15936
Math.ArcTan2 function 15936
Math.ArcTanh 15936
Math.ArcTanh function 15936
Math.Ceil 15937
Math.Ceil function 15937
Math.ClearExceptions 15937
Math.ClearExceptions function 15937
Math.CompareValue 15938
Math.CompareValue function 15938

Math.Cosecant 15938
Math.Cosecant function 15938
Math.Cosh 15939
Math.Cosh function 15939
Math.Cot 15939
Math.Cot function 15939
Math.Cotan 15940
Math.Cotan function 15940
Math.CotH 15940
Math.CotH function 15940
Math.Csc 15941
Math.Csc function 15941
Math.CscH 15941
Math.CscH function 15941
Math.CycleToDeg 15942
Math.CycleToDeg function 15942
Math.CycleToGrad 15942
Math.CycleToGrad function 15942
Math.CycleToRad 15942
Math.CycleToRad function 15942
Math.DegToCycle 15943
Math.DegToCycle function 15943
Math.DegToGrad 15943
Math.DegToGrad function 15943
Math.DegToRad 15944
Math.DegToRad function 15944
Math.DivMod 15944
Math.DivMod function 15944
Math.DoubleDecliningBalance 15944
Math.DoubleDecliningBalance function 15944
Math.EInvalidArgument 15927
Math.EnsureRange 15945
Math.EnsureRange function 15945
Math.Floor 15945
Math.Floor function 15945
Math.Frexp 15946
Math.Frexp function 15946
Math.FutureValue 15946
Math.FutureValue function 15946
Math.GetExceptionMask 15947
Math.GetExceptionMask function 15947
Math.GetPrecisionMode 15947
Math.GetPrecisionMode function 15947
Math.GetRoundMode 15948
Math.GetRoundMode function 15948
Math.GradToCycle 15949
Math.GradToCycle function 15949
Math.GradToDeg 15949
Math.GradToDeg function 15949
Math.GradToRad 15949
Math.GradToRad function 15949
Math.Hypot 15950
Math.Hypot function 15950
Math.IfThen 15950
Math.IfThen function 15950
Math.Infinity 15988
Math.Infinity constant 15988
Math.InRange 15950
Math.InRange function 15950
Math.InterestPayment 15951
Math.InterestPayment function 15951
Math.InterestRate 15952
Math.InterestRate function 15952
Math.InternalRateOfReturn 15952
Math.InternalRateOfReturn function 15952
Math.IntPower 15953
Math.IntPower function 15953
Math.IsInfinite 15953
Math.IsInfinite function 15953
Math.IsNan 15954
Math.IsNan function 15954
Math.IsZero 15954
Math.IsZero function 15954
Math.Ldexp 15955
Math.Ldexp function 15955
Math.LnXP1 15955
Math.LnXP1 function 15955
Math.Log10 15955

Math.Log10 function 15955
Math.Log2 15956
Math.Log2 function 15956
Math.LogN 15956
Math.LogN function 15956
Math.Max 15957
Math.Max function 15957
Math.MaxComp 15989
Math.MaxComp constant 15989
Math.MaxDouble 15989
Math.MaxDouble constant 15989
Math.MaxExtended 15989
Math.MaxExtended constant 15989
Math.MaxIntValue 15957
Math.MaxIntValue function 15957
Math.MaxSingle 15990
Math.MaxSingle constant 15990
Math.MaxValue 15958
Math.MaxValue function 15958
Math.Mean 15958
Math.Mean function 15958
Math.MeanAndStdDev 15959
Math.MeanAndStdDev function 15959
Math.Min 15959
Math.Min function 15959
Math.MinComp 15990
Math.MinComp constant 15990
Math.MinDouble 15991
Math.MinDouble constant 15991
Math.MinExtended 15991
Math.MinExtended constant 15991
Math.MinIntValue 15960
Math.MinIntValue function 15960
Math.MinSingle 15991
Math.MinSingle constant 15991
Math.MinValue 15960
Math.MinValue function 15960
Math.MomentSkewKurtosis 15961
Math.MomentSkewKurtosis function 15961
Math.NaN 15992
Math.NaN constant 15992
Math.NegativeValue 15992
Math.NegativeValue constant 15992
Math.NegInfinity 15993
Math.NegInfinity constant 15993
Math.NetPresentValue 15961
Math.NetPresentValue function 15961
Math.Norm 15962
Math.Norm function 15962
Math.NumberOfPeriods 15962
Math.NumberOfPeriods function 15962
Math.Payment 15963
Math.Payment function 15963
Math.PeriodPayment 15963
Math.PeriodPayment function 15963
Math.Poly 15963
Math.Poly function 15963
Math.PopnStdDev 15964
Math.PopnStdDev function 15964
Math.PopnVariance 15964
Math.PopnVariance function 15964
Math.PositiveValue 15993
Math.PositiveValue constant 15993
Math.Power 15965
Math.Power function 15965
Math.PresentValue 15965
Math.PresentValue function 15965
Math.RadToCycle 15966
Math.RadToCycle function 15966
Math.RadToDeg 15966
Math.RadToDeg function 15966
Math.RadToGrad 15966
Math.RadToGrad function 15966
Math.RandG 15967
Math.RandG function 15967
Math.RandomFrom 15967
Math.RandomFrom function 15967
Math.RandomRange 15968

Math.RandomRange function 15968
Math.RoundTo 15968
Math.RoundTo function 15968
Math.SameValue 15969
Math.SameValue function 15969
Math.Sec 15970
Math.Sec function 15970
Math.Secant 15970
Math.Secant function 15970
Math.SecH 15971
Math.SecH function 15971
Math.SetExceptionMask 15971
Math.SetExceptionMask function 15971
Math.SetPrecisionMode 15972
Math.SetPrecisionMode function 15972
Math.SetRoundMode 15972
Math.SetRoundMode function 15972
Math.Sign 15973
Math.Sign function 15973
Math.SimpleRoundTo 15973
Math.SimpleRoundTo function 15973
Math.SinCos 15974
Math.SinCos function 15974
Math.Sinh 15974
Math.Sinh function 15974
Math.SLNDepreciation 15975
Math.SLNDepreciation function 15975
Math.StdDev 15975
Math.StdDev function 15975
Math.Sum 15976
Math.Sum function 15976
Math.SumInt 15976
Math.SumInt function 15976
Math.SumOfSquares 15977
Math.SumOfSquares function 15977
Math.SumsAndSquares 15977
Math.SumsAndSquares function 15977
Math.SYDDepreciation 15978
Math.SYDDepreciation function 15978
Math.Tan 15978
Math.Tan function 15978
Math.Tanh 15979
Math.Tanh function 15979
Math.TFPUException 15981
Math.TFPUException enumeration 15981
Math.TFPUExceptionMask 15985
Math.TFPUExceptionMask type 15985
Math.TFPUPrecisionMode 15982
Math.TFPUPrecisionMode enumeration 15982
Math.TFPURoundingMode 15983
Math.TFPURoundingMode enumeration 15983
Math.TotalVariance 15979
Math.TotalVariance function 15979
Math.TPaymentTime 15984
Math.TPaymentTime enumeration 15984
Math.TRoundToRange 15986
Math.TRoundToRange type 15986
Math.TValueSign 15986
Math.TValueSign type 15986
Math.Variance 15980
Math.Variance function 15980
Math.ZeroValue 15994
Math.ZeroValue constant 15994
maTopToBottom enumeration member 16199
mba16Byte enumeration member 22566
mba8Byte enumeration member 22566
mbAbort enumeration member 11556
mbAll enumeration member 11556
mbBarBreak enumeration member 16199
mbBreak enumeration member 16199
mbCancel enumeration member 11556
mbClose enumeration member 839
mbHelp enumeration member 11556
mbIgnore enumeration member 11556
mbLeadByte enumeration member 22101, 23004
mbLeft enumeration member 6288
mbMiddle enumeration member 6288
mbMinimize enumeration member 839

mbNo enumeration member 11556
mbNone enumeration member 16199
mbNoToAll enumeration member 11556
mbOK enumeration member 11556
mbRestore enumeration member 839
mbRetry enumeration member 11556
mbRight enumeration member 6288
mbSingleByte enumeration member 22101, 23004
mbTrailByte enumeration member 22101, 23004
mbYes enumeration member 11556
mbYesToAll enumeration member 11556
mcDirective enumeration member 15911
mcField enumeration member 15911
mcFieldSeparator enumeration member 15911
mcIntlLiteral enumeration member 15911
mcLiteral enumeration member 15911
mcMask enumeration member 15911
mcMaskOpt enumeration member 15911
mcNone enumeration member 15911
MConnect 15995
MConnect namespace 15995
    Classes 15996
    Functions 16051
MConnect.GetMIDASAppServerList 16051
MConnect.GetMIDASAppServerList function 16051
MConnect.TCOMConnection 15996
MConnect.TCOMConnection.DoConnect 16001
MConnect.TCOMConnection.SetConnected 16001
MConnect.TCustomObjectBroker 16002
MConnect.TCustomObjectBroker.GetComputerForGUID 16006
MConnect.TCustomObjectBroker.GetComputerForProgID 16006
MConnect.TCustomObjectBroker.GetPortForComputer 16007
MConnect.TCustomObjectBroker.SetConnectStatus 16008
MConnect.TDCOMConnection 16008
MConnect.TDCOMConnection.ComputerName 16014
MConnect.TDCOMConnection.Create 16013
MConnect.TDCOMConnection.DoConnect 16013
MConnect.TDCOMConnection.ObjectBroker 16014
MConnect.TDispatchAppServer 16015
MConnect.TDispatchAppServer.Create 16017
MConnect.TDispatchAppServer.SafeCallException 16017
MConnect.TDispatchConnection 16017
MConnect.TDispatchConnection.AfterConnect 16031
MConnect.TDispatchConnection.AfterDisconnect 16031
MConnect.TDispatchConnection.BeforeConnect 16032
MConnect.TDispatchConnection.BeforeDisconnect 16032
MConnect.TDispatchConnection.Connected 16026
MConnect.TDispatchConnection.Create 16022
MConnect.TDispatchConnection.DoDisconnect 16023
MConnect.TDispatchConnection.GetConnected 16023
MConnect.TDispatchConnection.GetServer 16023
MConnect.TDispatchConnection.GetServerCLSID 16024
MConnect.TDispatchConnection.GetServerList 16024
MConnect.TDispatchConnection.LoginPrompt 16027
MConnect.TDispatchConnection.Notification 16025
MConnect.TDispatchConnection.ObjectBroker 16028
MConnect.TDispatchConnection.OnGetUsername 16033
MConnect.TDispatchConnection.OnLogin 16028
MConnect.TDispatchConnection.ServerGUID 16030
MConnect.TDispatchConnection.ServerName 16030
MConnect.TDispatchConnection.SetConnected 16025
MConnect.TOLEnterpriseConnection 16033
MConnect.TOLEnterpriseConnection.BrokerName 16038
MConnect.TOLEnterpriseConnection.ComputerName 16039
MConnect.TOLEnterpriseConnection.DoConnect 16038
MConnect.TSharedConnection 16039
MConnect.TSharedConnection.AfterConnect 16048
MConnect.TSharedConnection.AfterDisconnect 16049
MConnect.TSharedConnection.BeforeConnect 16049
MConnect.TSharedConnection.BeforeDisconnect 16049
MConnect.TSharedConnection.ChildName 16047
MConnect.TSharedConnection.ConnectEvent 16045
MConnect.TSharedConnection.Create 16044
MConnect.TSharedConnection.Destroy 16044
MConnect.TSharedConnection.DoConnect 16045
MConnect.TSharedConnection.DoDisconnect 16045
MConnect.TSharedConnection.GetConnected 16046
MConnect.TSharedConnection.GetServer 16046

MConnect.TSharedConnection.Notification 16046
MConnect.TSharedConnection.ParentConnection 16048
mdNearest enumeration member 14361
mdNull enumeration member 14361
mdPrimary enumeration member 14361
Menus 16053
Menus namespace 16053
Classes 16055
Constants 16209
Functions 16186
Structs, Records, Enums 16198
Types 16203
Variables 16207
Menus.AnsiSameCaption 16186
Menus.AnsiSameCaption function 16186
Menus.cDialogSuffix 16209
Menus.cDialogSuffix constant 16209
Menus.cHotkeyPrefix 16209
Menus.cHotkeyPrefix constant 16209
Menus.cLineCaption 16209
Menus.cLineCaption constant 16209
Menus.DrawMenuItem 16187
Menus.DrawMenuItem function 16187
Menus.EMenuError 16055
Menus.GetHotkey 16187
Menus.GetHotkey function 16187
Menus.IsAltGRPressed 16187
Menus.IsAltGRPressed function 16187
Menus.NewItem 16188
Menus.NewItem function 16188
Menus.NewLine 16189
Menus.NewLine function 16189
Menus.NewMenu 16190
Menus.NewMenu function 16190
Menus.NewPopupMenu 16191
Menus.NewPopupMenu function 16191
Menus.NewSubMenu 16192
Menus.NewSubMenu function 16192
Menus.PMenuItem 16203
Menus.PMenuItem type 16203
Menus.PopupList 16207
Menus.PopupList variable 16207
Menus.ShortCut 16193
Menus.ShortCut function 16193
Menus.ShortCutFromMessage 16194
Menus.ShortCutFromMessage function 16194
Menus.ShortCutItems 16207
Menus.ShortCutItems variable 16207
Menus.ShortCutToKey 16194
Menus.ShortCutToKey function 16194
Menus.ShortCutToText 16195
Menus.ShortCutToText function 16195
Menus.StripHotkey 16196
Menus.StripHotkey function 16196
Menus.TAdvancedMenuDrawItemEvent 16203
Menus.TAdvancedMenuDrawItemEvent type 16203
Menus.TextToShortCut 16196
Menus.TextToShortCut function 16196
Menus.TFindItemKind 16198
Menus.TFindItemKind enumeration 16198
Menus.TMainMenu 16056
Menus.TMainMenu.AutoHotkeys 16063
Menus.TMainMenu.AutoLineReduction 16064
Menus.TMainMenu.AutoMerge 16065
Menus.TMainMenu.BiDiMode 16065
Menus.TMainMenu.GetHandle 16060
Menus.TMainMenu.GetOle2AcceleratorTable 16060
Menus.TMainMenu.Images 16065
Menus.TMainMenu.MenuChanged 16061
Menus.TMainMenu.Merge 16061
Menus.TMainMenu.OnChange 16067
Menus.TMainMenu.OwnerDraw 16066
Menus.TMainMenu.ParentBiDiMode 16066
Menus.TMainMenu.PopulateOle2Menu 16062
Menus.TMainMenu.SetOle2MenuHandle 16062
Menus.TMainMenu.Unmerge 16063
Menus.TMenu 16067
Menus.TMenu.AdjustBiDiBehavior 16073

Menus.TMenu.AutoHotkeys 16082
Menus.TMenu.AutoLineReduction 16082
Menus.TMenu.BiDiMode 16083
Menus.TMenu.Create 16072
Menus.TMenu.CreateMenuItem 16073
Menus.TMenu.Destroy 16072
Menus.TMenu.DispatchCommand 16074
Menus.TMenu.DispatchPopup 16074
Menus.TMenu.DoBiDiModeChanged 16074
Menus.TMenu.DoChange 16075
Menus.TMenu.DoGetMenuString 16075
Menus.TMenu.FindItem 16076
Menus.TMenu.GetChildren 16076
Menus.TMenu.GetHandle 16076
Menus.TMenu.GetHelpContext 16077
Menus.TMenu.Handle 16083
Menus.TMenu.Images 16083
Menus.TMenu.IsOwnerDraw 16077
Menus.TMenu.IsRightToLeft 16077
Menus.TMenu.IsShortCut 16078
Menus.TMenu.Items 16084
Menus.TMenu.Loaded 16078
Menus.TMenu.MenuChanged 16079
Menus.TMenu.Notification 16079
Menus.TMenu.OnChange 16090
Menus.TMenu.OwnerDraw 16088
Menus.TMenu.ParentBiDiMode 16089
Menus.TMenu.ParentBiDiModeChanged 16080
Menus.TMenu.ProcessMenuChar 16080
Menus.TMenu.SetChildOrder 16081
Menus.TMenu.UpdateItems 16081
Menus.TMenu.WindowHandle 16089
Menus.TMenuActionLink 16090
Menus.TMenuActionLink.AssignClient 16093
Menus.TMenuActionLink.FClient 16093
Menus.TMenuActionLink.IsAutoCheckLinked 16094
Menus.TMenuActionLink.IsCaptionLinked 16094
Menus.TMenuActionLink.IsCheckedLinked 16095
Menus.TMenuActionLink.IsEnabledLinked 16095
Menus.TMenuActionLink.IsGroupIndexLinked 16095
Menus.TMenuActionLink.IsHelpContextLinked 16096
Menus.TMenuActionLink.IsHintLinked 16096
Menus.TMenuActionLink.IsImageIndexLinked 16097
Menus.TMenuActionLink.IsOnExecuteLinked 16097
Menus.TMenuActionLink.IsShortCutLinked 16097
Menus.TMenuActionLink.IsVisibleLinked 16098
Menus.TMenuActionLink.SetAutoCheck 16098
Menus.TMenuActionLink.SetCaption 16099
Menus.TMenuActionLink.SetChecked 16099
Menus.TMenuActionLink.SetEnabled 16099
Menus.TMenuActionLink.SetHelpContext 16100
Menus.TMenuActionLink.SetHint 16100
Menus.TMenuActionLink.SetImageIndex 16100
Menus.TMenuActionLink.SetOnExecute 16101
Menus.TMenuActionLink.SetShortCut 16101
Menus.TMenuActionLink.SetVisible 16102
Menus.TMenuActionLinkClass 16204
Menus.TMenuActionLinkClass type 16204
Menus.TMenuAnimation 16204
Menus.TMenuAnimation type 16204
Menus.TMenuAnimations 16199
Menus.TMenuAnimations enumeration 16199
Menus.TMenuAutoFlag 16205
Menus.TMenuAutoFlag type 16205
Menus.TMenuBreak 16199
Menus.TMenuBreak enumeration 16199
Menus.TMenuChangeEvent 16205
Menus.TMenuChangeEvent type 16205
Menus.TMenuDrawItemEvent 16206
Menus.TMenuDrawItemEvent type 16206
Menus.TMenuItem 16102
Menus.TMenuItem.Action 16133
Menus.TMenuItem.ActionChange 16110
Menus.TMenuItem.ActionLink 16134
Menus.TMenuItem.Add 16110
Menus.TMenuItem.AdvancedDrawItem 16115
Menus.TMenuItem.AssignTo 16115
Menus.TMenuItem.AutoCheck 16134

Menus.TMenuItem.AutoHotkeys 16134
Menus.TMenuItem.AutoLineReduction 16135
Menus.TMenuItem.Bitmap 16135
Menus.TMenuItem.Break 16136
Menus.TMenuItem.Caption 16136
Menus.TMenuItem.Checked 16140
Menus.TMenuItem.Clear 16115
Menus.TMenuItem.Click 16116
Menus.TMenuItem.Command 16141
Menus.TMenuItem.Count 16141
Menus.TMenuItem.Create 16107
Menus.TMenuItem.Default 16145
Menus.TMenuItem.DefineProperties 16116
Menus.TMenuItem.Delete 16117
Menus.TMenuItem.Destroy 16108
Menus.TMenuItem.DoDrawText 16117
Menus.TMenuItem.DrawItem 16118
Menus.TMenuItem.Enabled 16145
Menus.TMenuItem.Find 16118
Menus.TMenuItem.GetActionLinkClass 16118
Menus.TMenuItem.GetAutoHotkeys 16118
Menus.TMenuItem.GetAutoLineReduction 16119
Menus.TMenuItem.GetChildren 16119
Menus.TMenuItem.GetCount 16119
Menus.TMenuItem.GetEnumerator 16119
Menus.TMenuItem.GetHandle 16120
Menus.TMenuItem.GetImageList 16120
Menus.TMenuItem.GetItem 16120
Menus.TMenuItem.GetMenuIndex 16121
Menus.TMenuItem.GetParentComponent 16121
Menus.TMenuItem.GetParentMenu 16121
Menus.TMenuItem.GroupIndex 16146
Menus.TMenuItem.Handle 16147
Menus.TMenuItem.HasParent 16121
Menus.TMenuItem.HelpContext 16148
Menus.TMenuItem.Hint 16148
Menus.TMenuItem.ImageIndex 16148
Menus.TMenuItem.IndexOf 16122
Menus.TMenuItem.InitiateAction 16122
Menus.TMenuItem.Insert 16123
Menus.TMenuItem.InsertNewLine 16124
Menus.TMenuItem.InsertNewLineAfter 16124
Menus.TMenuItem.InsertNewLineBefore 16124
Menus.TMenuItem.IsLine 16125
Menus.TMenuItem.Items 16149
Menus.TMenuItem.Loaded 16125
Menus.TMenuItem.MeasureItem 16126
Menus.TMenuItem.MenuChanged 16126
Menus.TMenuItem.MenuIndex 16150
Menus.TMenuItem.NewBottomLine 16126
Menus.TMenuItem.NewTopLine 16127
Menus.TMenuItem.Notification 16127
Menus.TMenuItem.OnAdvancedDrawItem 16154
Menus.TMenuItem.OnClick 16154
Menus.TMenuItem.OnDrawItem 16159
Menus.TMenuItem.OnMeasureItem 16160
Menus.TMenuItem.Parent 16151
Menus.TMenuItem.RadioItem 16151
Menus.TMenuItem.Remove 16128
Menus.TMenuItem.RethinkHotkeys 16128
Menus.TMenuItem.RethinkLines 16129
Menus.TMenuItem.SetBreak 16129
Menus.TMenuItem.SetCaption 16130
Menus.TMenuItem.SetChecked 16130
Menus.TMenuItem.SetChildOrder 16130
Menus.TMenuItem.SetDefault 16130
Menus.TMenuItem.SetEnabled 16131
Menus.TMenuItem.SetGroupIndex 16131
Menus.TMenuItem.SetImageIndex 16131
Menus.TMenuItem.SetMenuIndex 16131
Menus.TMenuItem.SetParentComponent 16131
Menus.TMenuItem.SetRadioItem 16132
Menus.TMenuItem.SetShortCut 16132
Menus.TMenuItem.SetVisible 16132
Menus.TMenuItem.ShortCut 16152
Menus.TMenuItem.SubMenuImages 16152
Menus.TMenuItem.UpdateItems 16132
Menus.TMenuItem.Visible 16153

Menus.TMenuItemAutoFlag 16200
Menus.TMenuItemAutoFlag enumeration 16200
Menus.TMenuItemEnumerator 16160
Menus.TMenuItemEnumerator.Create 16161
Menus.TMenuItemEnumerator.Current 16162
Menus.TMenuItemEnumerator.GetCurrent 16161
Menus.TMenuItemEnumerator.MoveNext 16162
Menus.TMenuItemStack 16162
Menus.TMenuItemStack.ClearItem 16164
Menus.TMenuMeasureItemEvent 16206
Menus.TMenuMeasureItemEvent type 16206
Menus.TPopupAlignment 16201
Menus.TPopupAlignment enumeration 16201
Menus.TPopupList 16164
Menus.TPopupList.Add 16167
Menus.TPopupList.FWindow 16166
Menus.TPopupList.MainWndProc 16167
Menus.TPopupList.Remove 16167
Menus.TPopupList.Window 16168
Menus.TPopupList.WndProc 16168
Menus.TPopupMenu 16169
Menus.TPopupMenu.Alignment 16176
Menus.TPopupMenu.AutoHotkeys 16176
Menus.TPopupMenu.AutoLineReduction 16177
Menus.TPopupMenu.AutoPopup 16178
Menus.TPopupMenu.BiDiMode 16178
Menus.TPopupMenu.Create 16173
Menus.TPopupMenu.Destroy 16173
Menus.TPopupMenu.DoPopup 16174
Menus.TPopupMenu.HelpContext 16179
Menus.TPopupMenu.Images 16179
Menus.TPopupMenu.MenuAnimation 16180
Menus.TPopupMenu.OnChange 16184
Menus.TPopupMenu.OnPopup 16184
Menus.TPopupMenu.OwnerDraw 16180
Menus.TPopupMenu.ParentBiDiMode 16180
Menus.TPopupMenu.Popup 16174
Menus.TPopupMenu.PopupComponent 16181
Menus.TPopupMenu.PopupPoint 16183
Menus.TPopupMenu.SetPopupPoint 16175
Menus.TPopupMenu.TrackButton 16183
Menus.TPopupMenu.UseRightToLeftAlignment 16175
Menus.TTrackButton 16201
Menus.TTrackButton enumeration 16201
Menus.ValidMenuHotkeys 16207
Menus.ValidMenuHotkeys variable 16207
Messages 16211
Messages namespace 16211
    Constants 16305
    Structs, Records, Enums 16223
    Types 16278
Messages.BM_CLICK 16312
Messages.BM_CLICK constant 16312
Messages.BM_GETCHECK 16313
Messages.BM_GETCHECK constant 16313
Messages.BM_GETIMAGE 16313
Messages.BM_GETIMAGE constant 16313
Messages.BM_GETSTATE 16313
Messages.BM_GETSTATE constant 16313
Messages.BM_SETCHECK 16314
Messages.BM_SETCHECK constant 16314
Messages.BM_SETIMAGE 16314
Messages.BM_SETIMAGE constant 16314
Messages.BM_SETSTATE 16314
Messages.BM_SETSTATE constant 16314
Messages.BM_SETSTYLE 16315
Messages.BM_SETSTYLE constant 16315
Messages.BN_CLICKED 16315
Messages.BN_CLICKED constant 16315
Messages.BN_DBLCLK 16315
Messages.BN_DBLCLK constant 16315
Messages.BN_DISABLE 16316
Messages.BN_DISABLE constant 16316
Messages.BN_DOUBLECLICKED 16316
Messages.BN_DOUBLECLICKED constant 16316
Messages.BN_HILITE 16316
Messages.BN_HILITE constant 16316
Messages.BN_KILLFOCUS 16317

Messages.BN_KILLFOCUS constant 16317
Messages.BN_PAINT 16317
Messages.BN_PAINT constant 16317
Messages.BN_PUSHED 16317
Messages.BN_PUSHED constant 16317
Messages.BN_SETFOCUS 16318
Messages.BN_SETFOCUS constant 16318
Messages.BN_UNHILITE 16318
Messages.BN_UNHILITE constant 16318
Messages.BN_UNPUSHED 16318
Messages.BN_UNPUSHED constant 16318
Messages.CB_ADDSTRING 16319
Messages.CB_ADDSTRING constant 16319
Messages.CB_DELETESTRING 16319
Messages.CB_DELETESTRING constant 16319
Messages.CB_DIR 16319
Messages.CB_DIR constant 16319
Messages.CB_FINDSTRING 16320
Messages.CB_FINDSTRING constant 16320
Messages.CB_FINDSTRINGEXACT 16320
Messages.CB_FINDSTRINGEXACT constant 16320
Messages.CB_GETCOUNT 16320
Messages.CB_GETCOUNT constant 16320
Messages.CB_GETCURSEL 16321
Messages.CB_GETCURSEL constant 16321
Messages.CB_GETDROPPEDCONTROLRECT 16321
Messages.CB_GETDROPPEDCONTROLRECT constant 16321
Messages.CB_GETDROPPEDSTATE 16321
Messages.CB_GETDROPPEDSTATE constant 16321
Messages.CB_GETDROPPEDWIDTH 16322
Messages.CB_GETDROPPEDWIDTH constant 16322
Messages.CB_GETEDITSEL 16322
Messages.CB_GETEDITSEL constant 16322
Messages.CB_GETEXTENDEDUI 16322
Messages.CB_GETEXTENDEDUI constant 16322
Messages.CB_GETHORIZONTALEXTENT 16323
Messages.CB_GETHORIZONTALEXTENT constant 16323
Messages.CB_GETITEMDATA 16323
Messages.CB_GETITEMDATA constant 16323
Messages.CB_GETITEMHEIGHT 16323
Messages.CB_GETITEMHEIGHT constant 16323
Messages.CB_GETLBTEXT 16324
Messages.CB_GETLBTEXT constant 16324
Messages.CB_GETLBTEXTLEN 16324
Messages.CB_GETLBTEXTLEN constant 16324
Messages.CB_GETLOCALE 16324
Messages.CB_GETLOCALE constant 16324
Messages.CB_GETTOPINDEX 16325
Messages.CB_GETTOPINDEX constant 16325
Messages.CB_INITSTORAGE 16325
Messages.CB_INITSTORAGE constant 16325
Messages.CB_INSERTSTRING 16325
Messages.CB_INSERTSTRING constant 16325
Messages.CB_LIMITTEXT 16326
Messages.CB_LIMITTEXT constant 16326
Messages.CB_MSGMAX 16326
Messages.CB_MSGMAX constant 16326
Messages.CB_RESETCONTENT 16326
Messages.CB_RESETCONTENT constant 16326
Messages.CB_SELECTSTRING 16327
Messages.CB_SELECTSTRING constant 16327
Messages.CB_SETCURSEL 16327
Messages.CB_SETCURSEL constant 16327
Messages.CB_SETDROPPEDWIDTH 16327
Messages.CB_SETDROPPEDWIDTH constant 16327
Messages.CB_SETEDITSEL 16328
Messages.CB_SETEDITSEL constant 16328
Messages.CB_SETEXTENDEDUI 16328
Messages.CB_SETEXTENDEDUI constant 16328
Messages.CB_SETHORIZONTALEXTENT 16328
Messages.CB_SETHORIZONTALEXTENT constant 16328
Messages.CB_SETITEMDATA 16329
Messages.CB_SETITEMDATA constant 16329
Messages.CB_SETITEMHEIGHT 16329
Messages.CB_SETITEMHEIGHT constant 16329
Messages.CB_SETLOCALE 16329
Messages.CB_SETLOCALE constant 16329
Messages.CB_SETTOPINDEX 16330

Messages.CB_SETTOPINDEX constant 16330
Messages.CB_SHOWDROPDOWN 16330
Messages.CB_SHOWDROPDOWN constant 16330
Messages.CBN_CLOSEUP 16330
Messages.CBN_CLOSEUP constant 16330
Messages.CBN_DBLCLK 16331
Messages.CBN_DBLCLK constant 16331
Messages.CBN_DROPDOWN 16331
Messages.CBN_DROPDOWN constant 16331
Messages.CBN_EDITCHANGE 16331
Messages.CBN_EDITCHANGE constant 16331
Messages.CBN_EDITUPDATE 16332
Messages.CBN_EDITUPDATE constant 16332
Messages.CBN_ERRSPACE 16332
Messages.CBN_ERRSPACE constant 16332
Messages.CBN_KILLFOCUS 16332
Messages.CBN_KILLFOCUS constant 16332
Messages.CBN_SELCHANGE 16333
Messages.CBN_SELCHANGE constant 16333
Messages.CBN_SELENDCANCEL 16333
Messages.CBN_SELENDCANCEL constant 16333
Messages.CBN_SELENDOK 16333
Messages.CBN_SELENDOK constant 16333
Messages.CBN_SETFOCUS 16334
Messages.CBN_SETFOCUS constant 16334
Messages.DM_GETDEFID 16334
Messages.DM_GETDEFID constant 16334
Messages.DM_REPOSITION 16334
Messages.DM_REPOSITION constant 16334
Messages.DM_SETDEFID 16335
Messages.DM_SETDEFID constant 16335
Messages.EM_CANUNDO 16335
Messages.EM_CANUNDO constant 16335
Messages.EM_CHARFROMPOS 16335
Messages.EM_CHARFROMPOS constant 16335
Messages.EM_EMPTYUNDOBUFFER 16336
Messages.EM_EMPTYUNDOBUFFER constant 16336
Messages.EM_FMTLINES 16336
Messages.EM_FMTLINES constant 16336
Messages.EM_GETFIRSTVISIBLELINE 16336
Messages.EM_GETFIRSTVISIBLELINE constant 16336
Messages.EM_GETHANDLE 16337
Messages.EM_GETHANDLE constant 16337
Messages.EM_GETIMESTATUS 16337
Messages.EM_GETIMESTATUS constant 16337
Messages.EM_GETLIMITTEXT 16337
Messages.EM_GETLIMITTEXT constant 16337
Messages.EM_GETLINE 16338
Messages.EM_GETLINE constant 16338
Messages.EM_GETLINECOUNT 16338
Messages.EM_GETLINECOUNT constant 16338
Messages.EM_GETMARGINS 16338
Messages.EM_GETMARGINS constant 16338
Messages.EM_GETMODIFY 16339
Messages.EM_GETMODIFY constant 16339
Messages.EM_GETPASSWORDCHAR 16339
Messages.EM_GETPASSWORDCHAR constant 16339
Messages.EM_GETRECT 16339
Messages.EM_GETRECT constant 16339
Messages.EM_GETSEL 16340
Messages.EM_GETSEL constant 16340
Messages.EM_GETTHUMB 16340
Messages.EM_GETTHUMB constant 16340
Messages.EM_GETWORDBREAKPROC 16340
Messages.EM_GETWORDBREAKPROC constant 16340
Messages.EM_LIMITTEXT 16341
Messages.EM_LIMITTEXT constant 16341
Messages.EM_LINEFROMCHAR 16341
Messages.EM_LINEFROMCHAR constant 16341
Messages.EM_LINEINDEX 16341
Messages.EM_LINEINDEX constant 16341
Messages.EM_LINELENGTH 16342
Messages.EM_LINELENGTH constant 16342
Messages.EM_LINESCROLL 16342
Messages.EM_LINESCROLL constant 16342
Messages.EM_POSFROMCHAR 16342
Messages.EM_POSFROMCHAR constant 16342
Messages.EM_REPLACESEL 16343

Messages.EM_REPLACESEL constant 16343
Messages.EM_SCROLL 16343
Messages.EM_SCROLL constant 16343
Messages.EM_SCROLLCARET 16343
Messages.EM_SCROLLCARET constant 16343
Messages.EM_SETHANDLE 16344
Messages.EM_SETHANDLE constant 16344
Messages.EM_SETIMESTATUS 16344
Messages.EM_SETIMESTATUS constant 16344
Messages.EM_SETLIMITTEXT 16344
Messages.EM_SETLIMITTEXT constant 16344
Messages.EM_SETMARGINS 16345
Messages.EM_SETMARGINS constant 16345
Messages.EM_SETMODIFY 16345
Messages.EM_SETMODIFY constant 16345
Messages.EM_SETPASSWORDCHAR 16345
Messages.EM_SETPASSWORDCHAR constant 16345
Messages.EM_SETREADONLY 16346
Messages.EM_SETREADONLY constant 16346
Messages.EM_SETRECT 16346
Messages.EM_SETRECT constant 16346
Messages.EM_SETRECTNP 16346
Messages.EM_SETRECTNP constant 16346
Messages.EM_SETSEL 16347
Messages.EM_SETSEL constant 16347
Messages.EM_SETTABSTOPS 16347
Messages.EM_SETTABSTOPS constant 16347
Messages.EM_SETWORDBREAKPROC 16347
Messages.EM_SETWORDBREAKPROC constant 16347
Messages.EM_UNDO 16348
Messages.EM_UNDO constant 16348
Messages.EN_CHANGE 16348
Messages.EN_CHANGE constant 16348
Messages.EN_ERRSPACE 16348
Messages.EN_ERRSPACE constant 16348
Messages.EN_HSCROLL 16349
Messages.EN_HSCROLL constant 16349
Messages.EN_KILLFOCUS 16349
Messages.EN_KILLFOCUS constant 16349
Messages.EN_MAXTEXT 16349
Messages.EN_MAXTEXT constant 16349
Messages.EN_SETFOCUS 16350
Messages.EN_SETFOCUS constant 16350
Messages.EN_UPDATE 16350
Messages.EN_UPDATE constant 16350
Messages.EN_VSCROLL 16350
Messages.EN_VSCROLL constant 16350
Messages.LB_ADDFILE 16351
Messages.LB_ADDFILE constant 16351
Messages.LB_ADDSTRING 16351
Messages.LB_ADDSTRING constant 16351
Messages.LB_DELETESTRING 16351
Messages.LB_DELETESTRING constant 16351
Messages.LB_DIR 16352
Messages.LB_DIR constant 16352
Messages.LB_FINDSTRING 16352
Messages.LB_FINDSTRING constant 16352
Messages.LB_FINDSTRINGEXACT 16352
Messages.LB_FINDSTRINGEXACT constant 16352
Messages.LB_GETANCHORINDEX 16353
Messages.LB_GETANCHORINDEX constant 16353
Messages.LB_GETCARETINDEX 16353
Messages.LB_GETCARETINDEX constant 16353
Messages.LB_GETCOUNT 16353
Messages.LB_GETCOUNT constant 16353
Messages.LB_GETCURSEL 16354
Messages.LB_GETCURSEL constant 16354
Messages.LB_GETHORIZONTALEXTENT 16354
Messages.LB_GETHORIZONTALEXTENT constant 16354
Messages.LB_GETITEMDATA 16354
Messages.LB_GETITEMDATA constant 16354
Messages.LB_GETITEMHEIGHT 16355
Messages.LB_GETITEMHEIGHT constant 16355
Messages.LB_GETITEMRECT 16355
Messages.LB_GETITEMRECT constant 16355
Messages.LB_GETLOCALE 16355
Messages.LB_GETLOCALE constant 16355
Messages.LB_GETSEL 16356

Messages.LB_GETSEL constant 16356
Messages.LB_GETSELCOUNT 16356
Messages.LB_GETSELCOUNT constant 16356
Messages.LB_GETSELITEMS 16356
Messages.LB_GETSELITEMS constant 16356
Messages.LB_GETTEXT 16357
Messages.LB_GETTEXT constant 16357
Messages.LB_GETTEXTLEN 16357
Messages.LB_GETTEXTLEN constant 16357
Messages.LB_GETTOPINDEX 16357
Messages.LB_GETTOPINDEX constant 16357
Messages.LB_INITSTORAGE 16358
Messages.LB_INITSTORAGE constant 16358
Messages.LB_INSERTSTRING 16358
Messages.LB_INSERTSTRING constant 16358
Messages.LB_ITEMFROMPOINT 16358
Messages.LB_ITEMFROMPOINT constant 16358
Messages.LB_MSGMAX 16359
Messages.LB_MSGMAX constant 16359
Messages.LB_RESETCONTENT 16359
Messages.LB_RESETCONTENT constant 16359
Messages.LB_SELECTSTRING 16359
Messages.LB_SELECTSTRING constant 16359
Messages.LB_SELITEMRANGE 16360
Messages.LB_SELITEMRANGE constant 16360
Messages.LB_SELITEMRANGEEX 16360
Messages.LB_SELITEMRANGEEX constant 16360
Messages.LB_SETANCHORINDEX 16360
Messages.LB_SETANCHORINDEX constant 16360
Messages.LB_SETCARETINDEX 16361
Messages.LB_SETCARETINDEX constant 16361
Messages.LB_SETCOLUMNWIDTH 16361
Messages.LB_SETCOLUMNWIDTH constant 16361
Messages.LB_SETCOUNT 16361
Messages.LB_SETCOUNT constant 16361
Messages.LB_SETCURSEL 16362
Messages.LB_SETCURSEL constant 16362
Messages.LB_SETHORIZONTALEXTENT 16362
Messages.LB_SETHORIZONTALEXTENT constant 16362
Messages.LB_SETITEMDATA 16362
Messages.LB_SETITEMDATA constant 16362
Messages.LB_SETITEMHEIGHT 16363
Messages.LB_SETITEMHEIGHT constant 16363
Messages.LB_SETLOCALE 16363
Messages.LB_SETLOCALE constant 16363
Messages.LB_SETSEL 16363
Messages.LB_SETSEL constant 16363
Messages.LB_SETTABSTOPS 16364
Messages.LB_SETTABSTOPS constant 16364
Messages.LB_SETTOPINDEX 16364
Messages.LB_SETTOPINDEX constant 16364
Messages.LBN_DBLCLK 16364
Messages.LBN_DBLCLK constant 16364
Messages.LBN_ERRSPACE 16365
Messages.LBN_ERRSPACE constant 16365
Messages.LBN_KILLFOCUS 16365
Messages.LBN_KILLFOCUS constant 16365
Messages.LBN_SELCANCEL 16365
Messages.LBN_SELCANCEL constant 16365
Messages.LBN_SELCHANGE 16366
Messages.LBN_SELCHANGE constant 16366
Messages.LBN_SETFOCUS 16366
Messages.LBN_SETFOCUS constant 16366
Messages.PMessage 16279
Messages.PMessage type 16279
Messages.PSM_PAGEINFO 16366
Messages.PSM_PAGEINFO constant 16366
Messages.PSM_SHEETINFO 16367
Messages.PSM_SHEETINFO constant 16367
Messages.SBM_ENABLE_ARROWS 16367
Messages.SBM_ENABLE_ARROWS constant 16367
Messages.SBM_GETPOS 16367
Messages.SBM_GETPOS constant 16367
Messages.SBM_GETRANGE 16368
Messages.SBM_GETRANGE constant 16368
Messages.SBM_GETSCROLLINFO 16368
Messages.SBM_GETSCROLLINFO constant 16368
Messages.SBM_SETPOS 16368

Messages.SBM_SETPOS constant 16368
Messages.SBM_SETRANGE 16369
Messages.SBM_SETRANGE constant 16369
Messages.SBM_SETRANGEREDRAW 16369
Messages.SBM_SETRANGEREDRAW constant 16369
Messages.SBM_SETSCROLLINFO 16369
Messages.SBM_SETSCROLLINFO constant 16369
Messages.TMessage 16225
Messages.TMessage record 16225
Messages.TMSHMouseWheel 16226
Messages.TMSHMouseWheel record 16226
Messages.TWMActivate 16226
Messages.TWMActivate record 16226
Messages.TWMActivateApp 16227
Messages.TWMActivateApp record 16227
Messages.TWMAskCBFormatName 16227
Messages.TWMAskCBFormatName record 16227
Messages.TWMCancelMode 16280
Messages.TWMCancelMode type 16280
Messages.TWMChangeCBChain 16228
Messages.TWMChangeCBChain record 16228
Messages.TWMChangeUIState 16280
Messages.TWMChangeUIState type 16280
Messages.TWMChar 16280
Messages.TWMChar type 16280
Messages.TWMCharToItem 16228
Messages.TWMCharToItem record 16228
Messages.TWMChildActivate 16281
Messages.TWMChildActivate type 16281
Messages.TWMChooseFont_GetLogFont 16229
Messages.TWMChooseFont_GetLogFont record 16229
Messages.TWMClear 16281
Messages.TWMClear type 16281
Messages.TWMClose 16281
Messages.TWMClose type 16281
Messages.TWMCommand 16229
Messages.TWMCommand record 16229
Messages.TWMCompacting 16230
Messages.TWMCompacting record 16230
Messages.TWMCompareItem 16230
Messages.TWMCompareItem record 16230
Messages.TWMContextMenu 16231
Messages.TWMContextMenu record 16231
Messages.TWMCopy 16282
Messages.TWMCopy type 16282
Messages.TWMCopyData 16231
Messages.TWMCopyData record 16231
Messages.TWMCreate 16232
Messages.TWMCreate record 16232
Messages.TWMCtlColor 16232
Messages.TWMCtlColor record 16232
Messages.TWMCtlColorBtn 16282
Messages.TWMCtlColorBtn type 16282
Messages.TWMCtlColorDlg 16282
Messages.TWMCtlColorDlg type 16282
Messages.TWMCtlColorEdit 16283
Messages.TWMCtlColorEdit type 16283
Messages.TWMCtlColorListbox 16283
Messages.TWMCtlColorListbox type 16283
Messages.TWMCtlColorMsgbox 16283
Messages.TWMCtlColorMsgbox type 16283
Messages.TWMCtlColorScrollbar 16284
Messages.TWMCtlColorScrollbar type 16284
Messages.TWMCtlColorStatic 16284
Messages.TWMCtlColorStatic type 16284
Messages.TWMCut 16284
Messages.TWMCut type 16284
Messages.TWMDDE_Ack 16233
Messages.TWMDDE_Ack record 16233
Messages.TWMDDE_Advise 16233
Messages.TWMDDE_Advise record 16233
Messages.TWMDDE_Data 16234
Messages.TWMDDE_Data record 16234
Messages.TWMDDE_Execute 16234
Messages.TWMDDE_Execute record 16234
Messages.TWMDDE_Initiate 16235
Messages.TWMDDE_Initiate record 16235
Messages.TWMDDE_Poke 16235

Messages.TWMDDE_Poke record 16235
Messages.TWMDDE_Request 16236
Messages.TWMDDE_Request record 16236
Messages.TWMDDE_Terminate 16236
Messages.TWMDDE_Terminate record 16236
Messages.TWMDDE_Unadvise 16237
Messages.TWMDDE_Unadvise record 16237
Messages.TWMDeadChar 16285
Messages.TWMDeadChar type 16285
Messages.TWMDeleteItem 16237
Messages.TWMDeleteItem record 16237
Messages.TWMDestroy 16285
Messages.TWMDestroy type 16285
Messages.TWMDestroyClipboard 16285
Messages.TWMDestroyClipboard type 16285
Messages.TWMDevModeChange 16238
Messages.TWMDevModeChange record 16238
Messages.TWMDisplayChange 16238
Messages.TWMDisplayChange record 16238
Messages.TWMDrawClipboard 16286
Messages.TWMDrawClipboard type 16286
Messages.TWMDrawItem 16239
Messages.TWMDrawItem record 16239
Messages.TWMDropFiles 16239
Messages.TWMDropFiles record 16239
Messages.TWMEnable 16240
Messages.TWMEnable record 16240
Messages.TWMEndSession 16240
Messages.TWMEndSession record 16240
Messages.TWMEnterIdle 16241
Messages.TWMEnterIdle record 16241
Messages.TWMEnterMenuLoop 16241
Messages.TWMEnterMenuLoop record 16241
Messages.TWMEraseBkgnd 16242
Messages.TWMEraseBkgnd record 16242
Messages.TWMExitMenuLoop 16286
Messages.TWMExitMenuLoop type 16286
Messages.TWMFontChange 16286
Messages.TWMFontChange type 16286
Messages.TWMGetDlgCode 16287
Messages.TWMGetDlgCode type 16287
Messages.TWMGetFont 16287
Messages.TWMGetFont type 16287
Messages.TWMGetHotKey 16287
Messages.TWMGetHotKey type 16287
Messages.TWMGetIcon 16242
Messages.TWMGetIcon record 16242
Messages.TWMGetMinMaxInfo 16243
Messages.TWMGetMinMaxInfo record 16243
Messages.TWMGetText 16243
Messages.TWMGetText record 16243
Messages.TWMGetTextLength 16288
Messages.TWMGetTextLength type 16288
Messages.TWMHelp 16244
Messages.TWMHelp record 16244
Messages.TWMHotKey 16244
Messages.TWMHotKey record 16244
Messages.TWMHScroll 16288
Messages.TWMHScroll type 16288
Messages.TWMHScrollClipboard 16245
Messages.TWMHScrollClipboard record 16245
Messages.TWMIconEraseBkgnd 16288
Messages.TWMIconEraseBkgnd type 16288
Messages.TWMInitDialog 16245
Messages.TWMInitDialog record 16245
Messages.TWMInitMenu 16246
Messages.TWMInitMenu record 16246
Messages.TWMInitMenuPopup 16246
Messages.TWMInitMenuPopup record 16246
Messages.TWMKey 16247
Messages.TWMKey record 16247
Messages.TWMKeyDown 16289
Messages.TWMKeyDown type 16289
Messages.TWMKeyUp 16289
Messages.TWMKeyUp type 16289
Messages.TWMKillFocus 16247
Messages.TWMKillFocus record 16247
Messages.TWMLButtonDblClk 16289

Messages.TWMLButtonDblClk type 16289
Messages.TWMLButtonDown 16290
Messages.TWMLButtonDown type 16290
Messages.TWMLButtonUp 16290
Messages.TWMLButtonUp type 16290
Messages.TWMMButtonDblClk 16290
Messages.TWMMButtonDblClk type 16290
Messages.TWMMButtonDown 16291
Messages.TWMMButtonDown type 16291
Messages.TWMMButtonUp 16291
Messages.TWMMButtonUp type 16291
Messages.TWMMDIActivate 16248
Messages.TWMMDIActivate record 16248
Messages.TWMMDICascade 16248
Messages.TWMMDICascade record 16248
Messages.TWMMDICreate 16249
Messages.TWMMDICreate record 16249
Messages.TWMMDIDestroy 16249
Messages.TWMMDIDestroy record 16249
Messages.TWMMDIGetActive 16291
Messages.TWMMDIGetActive type 16291
Messages.TWMMDIIconArrange 16292
Messages.TWMMDIIconArrange type 16292
Messages.TWMMDIMaximize 16250
Messages.TWMMDIMaximize record 16250
Messages.TWMMDINext 16250
Messages.TWMMDINext record 16250
Messages.TWMMDIRefreshMenu 16292
Messages.TWMMDIRefreshMenu type 16292
Messages.TWMMDIRestore 16251
Messages.TWMMDIRestore record 16251
Messages.TWMMDISetMenu 16251
Messages.TWMMDISetMenu record 16251
Messages.TWMMDITile 16252
Messages.TWMMDITile record 16252
Messages.TWMMeasureItem 16252
Messages.TWMMeasureItem record 16252
Messages.TWMMenuChar 16253
Messages.TWMMenuChar record 16253
Messages.TWMMenuSelect 16253
Messages.TWMMenuSelect record 16253
Messages.TWMMouse 16254
Messages.TWMMouse record 16254
Messages.TWMMouseActivate 16254
Messages.TWMMouseActivate record 16254
Messages.TWMMouseMove 16292
Messages.TWMMouseMove type 16292
Messages.TWMMouseWheel 16255
Messages.TWMMouseWheel record 16255
Messages.TWMMove 16255
Messages.TWMMove record 16255
Messages.TWMMoving 16256
Messages.TWMMoving record 16256
Messages.TWMNCActivate 16256
Messages.TWMNCActivate record 16256
Messages.TWMNCCalcSize 16257
Messages.TWMNCCalcSize record 16257
Messages.TWMNCCreate 16257
Messages.TWMNCCreate record 16257
Messages.TWMNCDestroy 16293
Messages.TWMNCDestroy type 16293
Messages.TWMNCHitMessage 16258
Messages.TWMNCHitMessage record 16258
Messages.TWMNCHitTest 16258
Messages.TWMNCHitTest record 16258
Messages.TWMNCLButtonDblClk 16293
Messages.TWMNCLButtonDblClk type 16293
Messages.TWMNCLButtonDown 16293
Messages.TWMNCLButtonDown type 16293
Messages.TWMNCLButtonUp 16294
Messages.TWMNCLButtonUp type 16294
Messages.TWMNCMButtonDblClk 16294
Messages.TWMNCMButtonDblClk type 16294
Messages.TWMNCMButtonDown 16294
Messages.TWMNCMButtonDown type 16294
Messages.TWMNCMButtonUp 16295
Messages.TWMNCMButtonUp type 16295
Messages.TWMNCMouseMove 16295

Messages.TWMNCMouseMove type 16295
Messages.TWMNCPaint 16259
Messages.TWMNCPaint record 16259
Messages.TWMNCRButtonDblClk 16295
Messages.TWMNCRButtonDblClk type 16295
Messages.TWMNCRButtonDown 16296
Messages.TWMNCRButtonDown type 16296
Messages.TWMNCRButtonUp 16296
Messages.TWMNCRButtonUp type 16296
Messages.TWMNextDlgCtl 16259
Messages.TWMNextDlgCtl record 16259
Messages.TWMNoParams 16260
Messages.TWMNoParams record 16260
Messages.TWMNotify 16260
Messages.TWMNotify record 16260
Messages.TWMNotifyFormat 16261
Messages.TWMNotifyFormat record 16261
Messages.TWMPaint 16261
Messages.TWMPaint record 16261
Messages.TWMPaintClipboard 16262
Messages.TWMPaintClipboard record 16262
Messages.TWMPaintIcon 16296
Messages.TWMPaintIcon type 16296
Messages.TWMPaletteChanged 16262
Messages.TWMPaletteChanged record 16262
Messages.TWMPaletteIsChanging 16263
Messages.TWMPaletteIsChanging record 16263
Messages.TWMParentNotify 16263
Messages.TWMParentNotify record 16263
Messages.TWMPaste 16297
Messages.TWMPaste type 16297
Messages.TWMPower 16264
Messages.TWMPower record 16264
Messages.TWMPrint 16264
Messages.TWMPrint record 16264
Messages.TWMPrintClient 16297
Messages.TWMPrintClient type 16297
Messages.TWMQueryDragIcon 16297
Messages.TWMQueryDragIcon type 16297
Messages.TWMQueryEndSession 16265
Messages.TWMQueryEndSession record 16265
Messages.TWMQueryNewPalette 16298
Messages.TWMQueryNewPalette type 16298
Messages.TWMQueryOpen 16298
Messages.TWMQueryOpen type 16298
Messages.TWMQueryUIState 16298
Messages.TWMQueryUIState type 16298
Messages.TWMQueueSync 16299
Messages.TWMQueueSync type 16299
Messages.TWMQuit 16265
Messages.TWMQuit record 16265
Messages.TWMRButtonDblClk 16299
Messages.TWMRButtonDblClk type 16299
Messages.TWMRButtonDown 16299
Messages.TWMRButtonDown type 16299
Messages.TWMRButtonUp 16300
Messages.TWMRButtonUp type 16300
Messages.TWMRenderAllFormats 16300
Messages.TWMRenderAllFormats type 16300
Messages.TWMRenderFormat 16266
Messages.TWMRenderFormat record 16266
Messages.TWMScroll 16266
Messages.TWMScroll record 16266
Messages.TWMSetCursor 16267
Messages.TWMSetCursor record 16267
Messages.TWMSetFocus 16267
Messages.TWMSetFocus record 16267
Messages.TWMSetFont 16268
Messages.TWMSetFont record 16268
Messages.TWMSetHotKey 16268
Messages.TWMSetHotKey record 16268
Messages.TWMSetIcon 16269
Messages.TWMSetIcon record 16269
Messages.TWMSetRedraw 16269
Messages.TWMSetRedraw record 16269
Messages.TWMSetText 16270
Messages.TWMSetText record 16270
Messages.TWMSettingChange 16270

Messages.TWMSettingChange record 16270
Messages.TWMShowWindow 16271
Messages.TWMShowWindow record 16271
Messages.TWMSize 16271
Messages.TWMSize record 16271
Messages.TWMSizeClipboard 16272
Messages.TWMSizeClipboard record 16272
Messages.TWMSpoolerStatus 16272
Messages.TWMSpoolerStatus record 16272
Messages.TWMStyleChange 16273
Messages.TWMStyleChange record 16273
Messages.TWMStyleChanged 16300
Messages.TWMStyleChanged type 16300
Messages.TWMStyleChanging 16301
Messages.TWMStyleChanging type 16301
Messages.TWMSysChar 16301
Messages.TWMSysChar type 16301
Messages.TWMSysColorChange 16301
Messages.TWMSysColorChange type 16301
Messages.TWMSysCommand 16273
Messages.TWMSysCommand record 16273
Messages.TWMSysDeadChar 16274
Messages.TWMSysDeadChar record 16274
Messages.TWMSysKeyDown 16302
Messages.TWMSysKeyDown type 16302
Messages.TWMSysKeyUp 16302
Messages.TWMSysKeyUp type 16302
Messages.TWMSystemError 16274
Messages.TWMSystemError record 16274
Messages.TWMTimeChange 16302
Messages.TWMTimeChange type 16302
Messages.TWMTimer 16275
Messages.TWMTimer record 16275
Messages.TWMUIState 16275
Messages.TWMUIState record 16275
Messages.TWMUndo 16303
Messages.TWMUndo type 16303
Messages.TWMUpdateUIState 16303
Messages.TWMUpdateUIState type 16303
Messages.TWMVKeyToItem 16303
Messages.TWMVKeyToItem type 16303
Messages.TWMVScroll 16304
Messages.TWMVScroll type 16304
Messages.TWMVScrollClipboard 16276
Messages.TWMVScrollClipboard record 16276
Messages.TWMWindowPosChanged 16304
Messages.TWMWindowPosChanged type 16304
Messages.TWMWindowPosChanging 16304
Messages.TWMWindowPosChanging type 16304
Messages.TWMWindowPosMsg 16276
Messages.TWMWindowPosMsg record 16276
Messages.TWMWinIniChange 16277
Messages.TWMWinIniChange record 16277
Messages.WM_ACTIVATE 16370
Messages.WM_ACTIVATE constant 16370
Messages.WM_ACTIVATEAPP 16370
Messages.WM_ACTIVATEAPP constant 16370
Messages.WM_APP 16370
Messages.WM_APP constant 16370
Messages.WM_APPCOMMAND 16371
Messages.WM_APPCOMMAND constant 16371
Messages.WM_ASKCBFORMATNAME 16371
Messages.WM_ASKCBFORMATNAME constant 16371
Messages.WM_CANCELJOURNAL 16371
Messages.WM_CANCELJOURNAL constant 16371
Messages.WM_CANCELMODE 16372
Messages.WM_CANCELMODE constant 16372
Messages.WM_CAPTURECHANGED 16372
Messages.WM_CAPTURECHANGED constant 16372
Messages.WM_CHANGECBCHAIN 16372
Messages.WM_CHANGECBCHAIN constant 16372
Messages.WM_CHANGEUISTATE 16373
Messages.WM_CHANGEUISTATE constant 16373
Messages.WM_CHAR 16373
Messages.WM_CHAR constant 16373
Messages.WM_CHARTOITEM 16373
Messages.WM_CHARTOITEM constant 16373
Messages.WM_CHILDACTIVATE 16374

Messages.WM_CHILDACTIVATE constant 16374
Messages.WM_CLEAR 16374
Messages.WM_CLEAR constant 16374
Messages.WM_CLOSE 16374
Messages.WM_CLOSE constant 16374
Messages.WM_COALESCE_FIRST 16375
Messages.WM_COALESCE_FIRST constant 16375
Messages.WM_COALESCE_LAST 16375
Messages.WM_COALESCE_LAST constant 16375
Messages.WM_COMMAND 16375
Messages.WM_COMMAND constant 16375
Messages.WM_COMMNOTIFY 16376
Messages.WM_COMMNOTIFY constant 16376
Messages.WM_COMPACTING 16376
Messages.WM_COMPACTING constant 16376
Messages.WM_COMPAREITEM 16376
Messages.WM_COMPAREITEM constant 16376
Messages.WM_CONTEXTMENU 16377
Messages.WM_CONTEXTMENU constant 16377
Messages.WM_COPY 16377
Messages.WM_COPY constant 16377
Messages.WM_COPYDATA 16377
Messages.WM_COPYDATA constant 16377
Messages.WM_CREATE 16378
Messages.WM_CREATE constant 16378
Messages.WM_CTLCOLOR 16378
Messages.WM_CTLCOLOR constant 16378
Messages.WM_CTLCOLORBTN 16378
Messages.WM_CTLCOLORBTN constant 16378
Messages.WM_CTLCOLORDLG 16379
Messages.WM_CTLCOLORDLG constant 16379
Messages.WM_CTLCOLOREDIT 16379
Messages.WM_CTLCOLOREDIT constant 16379
Messages.WM_CTLCOLORLISTBOX 16379
Messages.WM_CTLCOLORLISTBOX constant 16379
Messages.WM_CTLCOLORMSGBOX 16380
Messages.WM_CTLCOLORMSGBOX constant 16380
Messages.WM_CTLCOLORSCROLLBAR 16380
Messages.WM_CTLCOLORSCROLLBAR constant 16380
Messages.WM_CTLCOLORSTATIC 16380
Messages.WM_CTLCOLORSTATIC constant 16380
Messages.WM_CUT 16381
Messages.WM_CUT constant 16381
Messages.WM_DDE_ACK 16381
Messages.WM_DDE_ACK constant 16381
Messages.WM_DDE_ADVISE 16381
Messages.WM_DDE_ADVISE constant 16381
Messages.WM_DDE_DATA 16382
Messages.WM_DDE_DATA constant 16382
Messages.WM_DDE_EXECUTE 16382
Messages.WM_DDE_EXECUTE constant 16382
Messages.WM_DDE_FIRST 16382
Messages.WM_DDE_FIRST constant 16382
Messages.WM_DDE_INITIATE 16383
Messages.WM_DDE_INITIATE constant 16383
Messages.WM_DDE_LAST 16383
Messages.WM_DDE_LAST constant 16383
Messages.WM_DDE_POKE 16383
Messages.WM_DDE_POKE constant 16383
Messages.WM_DDE_REQUEST 16384
Messages.WM_DDE_REQUEST constant 16384
Messages.WM_DDE_TERMINATE 16384
Messages.WM_DDE_TERMINATE constant 16384
Messages.WM_DDE_UNADVISE 16384
Messages.WM_DDE_UNADVISE constant 16384
Messages.WM_DEADCHAR 16385
Messages.WM_DEADCHAR constant 16385
Messages.WM_DELETEITEM 16385
Messages.WM_DELETEITEM constant 16385
Messages.WM_DESTROY 16385
Messages.WM_DESTROY constant 16385
Messages.WM_DESTROYCLIPBOARD 16386
Messages.WM_DESTROYCLIPBOARD constant 16386
Messages.WM_DEVICECHANGE 16386
Messages.WM_DEVICECHANGE constant 16386
Messages.WM_DEVMODECHANGE 16386
Messages.WM_DEVMODECHANGE constant 16386
Messages.WM_DISPLAYCHANGE 16387

Messages.WM_DISPLAYCHANGE constant 16387
Messages.WM_DRAWCLIPBOARD 16387
Messages.WM_DRAWCLIPBOARD constant 16387
Messages.WM_DRAWITEM 16387
Messages.WM_DRAWITEM constant 16387
Messages.WM_DROPFILES 16388
Messages.WM_DROPFILES constant 16388
Messages.WM_DWMCOLORIZATIONCOLORCHANGED 16388
Messages.WM_DWMCOLORIZATIONCOLORCHANGED constant 16388
Messages.WM_DWMCOMPOSITIONCHANGED 16388
Messages.WM_DWMCOMPOSITIONCHANGED constant 16388
Messages.WM_DWMNCRENDERINGCHANGED 16389
Messages.WM_DWMNCRENDERINGCHANGED constant 16389
Messages.WM_DWMWINDOWMAXIMIZEDCHANGE 16389
Messages.WM_DWMWINDOWMAXIMIZEDCHANGE constant 16389
Messages.WM_ENABLE 16389
Messages.WM_ENABLE constant 16389
Messages.WM_ENDSESSION 16390
Messages.WM_ENDSESSION constant 16390
Messages.WM_ENTERIDLE 16390
Messages.WM_ENTERIDLE constant 16390
Messages.WM_ENTERMENULOOP 16390
Messages.WM_ENTERMENULOOP constant 16390
Messages.WM_ENTERSIZEMOVE 16391
Messages.WM_ENTERSIZEMOVE constant 16391
Messages.WM_ERASEBKGND 16391
Messages.WM_ERASEBKGND constant 16391
Messages.WM_EXITMENULOOP 16391
Messages.WM_EXITMENULOOP constant 16391
Messages.WM_EXITSIZEMOVE 16392
Messages.WM_EXITSIZEMOVE constant 16392
Messages.WM_FONTCHANGE 16392
Messages.WM_FONTCHANGE constant 16392
Messages.WM_GETDLGCODE 16392
Messages.WM_GETDLGCODE constant 16392
Messages.WM_GETFONT 16393
Messages.WM_GETFONT constant 16393
Messages.WM_GETHOTKEY 16393
Messages.WM_GETHOTKEY constant 16393
Messages.WM_GETICON 16393
Messages.WM_GETICON constant 16393
Messages.WM_GETMINMAXINFO 16394
Messages.WM_GETMINMAXINFO constant 16394
Messages.WM_GETOBJECT 16394
Messages.WM_GETOBJECT constant 16394
Messages.WM_GETTEXT 16394
Messages.WM_GETTEXT constant 16394
Messages.WM_GETTEXTLENGTH 16395
Messages.WM_GETTEXTLENGTH constant 16395
Messages.WM_HANDHELDFIRST 16395
Messages.WM_HANDHELDFIRST constant 16395
Messages.WM_HANDHELDLAST 16395
Messages.WM_HANDHELDLAST constant 16395
Messages.WM_HELP 16396
Messages.WM_HELP constant 16396
Messages.WM_HOTKEY 16396
Messages.WM_HOTKEY constant 16396
Messages.WM_HSCROLL 16396
Messages.WM_HSCROLL constant 16396
Messages.WM_HSCROLLCLIPBOARD 16397
Messages.WM_HSCROLLCLIPBOARD constant 16397
Messages.WM_ICONERASEBKGND 16397
Messages.WM_ICONERASEBKGND constant 16397
Messages.WM_IME_CHAR 16397
Messages.WM_IME_CHAR constant 16397
Messages.WM_IME_COMPOSITION 16398
Messages.WM_IME_COMPOSITION constant 16398
Messages.WM_IME_COMPOSITIONFULL 16398
Messages.WM_IME_COMPOSITIONFULL constant 16398
Messages.WM_IME_CONTROL 16398
Messages.WM_IME_CONTROL constant 16398
Messages.WM_IME_ENDCOMPOSITION 16399
Messages.WM_IME_ENDCOMPOSITION constant 16399
Messages.WM_IME_KEYDOWN 16399
Messages.WM_IME_KEYDOWN constant 16399

Messages.WM_IME_KEYLAST 16399
Messages.WM_IME_KEYLAST constant 16399
Messages.WM_IME_KEYUP 16400
Messages.WM_IME_KEYUP constant 16400
Messages.WM_IME_NOTIFY 16400
Messages.WM_IME_NOTIFY constant 16400
Messages.WM_IME_REQUEST 16400
Messages.WM_IME_REQUEST constant 16400
Messages.WM_IME_SELECT 16401
Messages.WM_IME_SELECT constant 16401
Messages.WM_IME_SETCONTEXT 16401
Messages.WM_IME_SETCONTEXT constant 16401
Messages.WM_IME_STARTCOMPOSITION 16401
Messages.WM_IME_STARTCOMPOSITION constant 16401
Messages.WM_INITDIALOG 16402
Messages.WM_INITDIALOG constant 16402
Messages.WM_INITMENU 16402
Messages.WM_INITMENU constant 16402
Messages.WM_INITMENUPOPUP 16402
Messages.WM_INITMENUPOPUP constant 16402
Messages.WM_INPUT 16403
Messages.WM_INPUT constant 16403
Messages.WM_INPUTLANGCHANGE 16403
Messages.WM_INPUTLANGCHANGE constant 16403
Messages.WM_INPUTLANGCHANGEREQUEST 16403
Messages.WM_INPUTLANGCHANGEREQUEST constant 16403
Messages.WM_KEYDOWN 16404
Messages.WM_KEYDOWN constant 16404
Messages.WM_KEYFIRST 16404
Messages.WM_KEYFIRST constant 16404
Messages.WM_KEYLAST 16404
Messages.WM_KEYLAST constant 16404
Messages.WM_KEYUP 16405
Messages.WM_KEYUP constant 16405
Messages.WM_KILLFOCUS 16405
Messages.WM_KILLFOCUS constant 16405
Messages.WM_LBUTTONDBLCLK 16405
Messages.WM_LBUTTONDBLCLK constant 16405
Messages.WM_LBUTTONDOWN 16406
Messages.WM_LBUTTONDOWN constant 16406
Messages.WM_LBUTTONUP 16406
Messages.WM_LBUTTONUP constant 16406
Messages.WM_MBUTTONDBLCLK 16406
Messages.WM_MBUTTONDBLCLK constant 16406
Messages.WM_MBUTTONDOWN 16407
Messages.WM_MBUTTONDOWN constant 16407
Messages.WM_MBUTTONUP 16407
Messages.WM_MBUTTONUP constant 16407
Messages.WM_MDIACTIVATE 16407
Messages.WM_MDIACTIVATE constant 16407
Messages.WM_MDICASCADE 16408
Messages.WM_MDICASCADE constant 16408
Messages.WM_MDICREATE 16408
Messages.WM_MDICREATE constant 16408
Messages.WM_MDIDESTROY 16408
Messages.WM_MDIDESTROY constant 16408
Messages.WM_MDIGETACTIVE 16409
Messages.WM_MDIGETACTIVE constant 16409
Messages.WM_MDIICONARRANGE 16409
Messages.WM_MDIICONARRANGE constant 16409
Messages.WM_MDIMAXIMIZE 16409
Messages.WM_MDIMAXIMIZE constant 16409
Messages.WM_MDINEXT 16410
Messages.WM_MDINEXT constant 16410
Messages.WM_MDIREFRESHMENU 16410
Messages.WM_MDIREFRESHMENU constant 16410
Messages.WM_MDIRESTORE 16410
Messages.WM_MDIRESTORE constant 16410
Messages.WM_MDISETMENU 16411
Messages.WM_MDISETMENU constant 16411
Messages.WM_MDITILE 16411
Messages.WM_MDITILE constant 16411
Messages.WM_MEASUREITEM 16411
Messages.WM_MEASUREITEM constant 16411
Messages.WM_MENUCHAR 16412
Messages.WM_MENUCHAR constant 16412
Messages.WM_MENUCOMMAND 16412
Messages.WM_MENUCOMMAND constant 16412

Messages.WM_MENUDRAG 16412
Messages.WM_MENUDRAG constant 16412
Messages.WM_MENUGETOBJECT 16413
Messages.WM_MENUGETOBJECT constant 16413
Messages.WM_MENURBUTTONUP 16413
Messages.WM_MENURBUTTONUP constant 16413
Messages.WM_MENUSELECT 16413
Messages.WM_MENUSELECT constant 16413
Messages.WM_MOUSEACTIVATE 16414
Messages.WM_MOUSEACTIVATE constant 16414
Messages.WM_MOUSEFIRST 16414
Messages.WM_MOUSEFIRST constant 16414
Messages.WM_MOUSEHOVER 16414
Messages.WM_MOUSEHOVER constant 16414
Messages.WM_MOUSELAST 16415
Messages.WM_MOUSELAST constant 16415
Messages.WM_MOUSELEAVE 16415
Messages.WM_MOUSELEAVE constant 16415
Messages.WM_MOUSEMOVE 16415
Messages.WM_MOUSEMOVE constant 16415
Messages.WM_MOUSEWHEEL 16416
Messages.WM_MOUSEWHEEL constant 16416
Messages.WM_MOVE 16416
Messages.WM_MOVE constant 16416
Messages.WM_MOVING 16416
Messages.WM_MOVING constant 16416
Messages.WM_NCACTIVATE 16417
Messages.WM_NCACTIVATE constant 16417
Messages.WM_NCCALCSIZE 16417
Messages.WM_NCCALCSIZE constant 16417
Messages.WM_NCCREATE 16417
Messages.WM_NCCREATE constant 16417
Messages.WM_NCDESTROY 16418
Messages.WM_NCDESTROY constant 16418
Messages.WM_NCHITTEST 16418
Messages.WM_NCHITTEST constant 16418
Messages.WM_NCLBUTTONDBLCLK 16418
Messages.WM_NCLBUTTONDBLCLK constant 16418
Messages.WM_NCLBUTTONDOWN 16419
Messages.WM_NCLBUTTONDOWN constant 16419
Messages.WM_NCLBUTTONUP 16419
Messages.WM_NCLBUTTONUP constant 16419
Messages.WM_NCMBUTTONDBLCLK 16419
Messages.WM_NCMBUTTONDBLCLK constant 16419
Messages.WM_NCMBUTTONDOWN 16420
Messages.WM_NCMBUTTONDOWN constant 16420
Messages.WM_NCMBUTTONUP 16420
Messages.WM_NCMBUTTONUP constant 16420
Messages.WM_NCMOUSEHOVER 16420
Messages.WM_NCMOUSEHOVER constant 16420
Messages.WM_NCMOUSELEAVE 16421
Messages.WM_NCMOUSELEAVE constant 16421
Messages.WM_NCMOUSEMOVE 16421
Messages.WM_NCMOUSEMOVE constant 16421
Messages.WM_NCPAINT 16421
Messages.WM_NCPAINT constant 16421
Messages.WM_NCRBUTTONDBLCLK 16422
Messages.WM_NCRBUTTONDBLCLK constant 16422
Messages.WM_NCRBUTTONDOWN 16422
Messages.WM_NCRBUTTONDOWN constant 16422
Messages.WM_NCRBUTTONUP 16422
Messages.WM_NCRBUTTONUP constant 16422
Messages.WM_NCXBUTTONDBLCLK 16423
Messages.WM_NCXBUTTONDBLCLK constant 16423
Messages.WM_NCXBUTTONDOWN 16423
Messages.WM_NCXBUTTONDOWN constant 16423
Messages.WM_NCXBUTTONUP 16423
Messages.WM_NCXBUTTONUP constant 16423
Messages.WM_NEXTDLGCTL 16424
Messages.WM_NEXTDLGCTL constant 16424
Messages.WM_NEXTMENU 16424
Messages.WM_NEXTMENU constant 16424
Messages.WM_NOTIFY 16424
Messages.WM_NOTIFY constant 16424
Messages.WM_NOTIFYFORMAT 16425
Messages.WM_NOTIFYFORMAT constant 16425
Messages.WM_NULL 16425
Messages.WM_NULL constant 16425

Messages.WM_PAINT 16425
Messages.WM_PAINT constant 16425
Messages.WM_PAINTCLIPBOARD 16426
Messages.WM_PAINTCLIPBOARD constant 16426
Messages.WM_PAINTICON 16426
Messages.WM_PAINTICON constant 16426
Messages.WM_PALETTECHANGED 16426
Messages.WM_PALETTECHANGED constant 16426
Messages.WM_PALETTEISCHANGING 16427
Messages.WM_PALETTEISCHANGING constant 16427
Messages.WM_PARENTNOTIFY 16427
Messages.WM_PARENTNOTIFY constant 16427
Messages.WM_PASTE 16427
Messages.WM_PASTE constant 16427
Messages.WM_PENWINFIRST 16428
Messages.WM_PENWINFIRST constant 16428
Messages.WM_PENWINLAST 16428
Messages.WM_PENWINLAST constant 16428
Messages.WM_POWER 16428
Messages.WM_POWER constant 16428
Messages.WM_POWERBROADCAST 16429
Messages.WM_POWERBROADCAST constant 16429
Messages.WM_PRINT 16429
Messages.WM_PRINT constant 16429
Messages.WM_PRINTCLIENT 16429
Messages.WM_PRINTCLIENT constant 16429
Messages.WM_QUERYDRAGICON 16430
Messages.WM_QUERYDRAGICON constant 16430
Messages.WM_QUERYENDSESSION 16430
Messages.WM_QUERYENDSESSION constant 16430
Messages.WM_QUERYNEWPALETTE 16430
Messages.WM_QUERYNEWPALETTE constant 16430
Messages.WM_QUERYOPEN 16431
Messages.WM_QUERYOPEN constant 16431
Messages.WM_QUERYUISTATE 16431
Messages.WM_QUERYUISTATE constant 16431
Messages.WM_QUEUESYNC 16431
Messages.WM_QUEUESYNC constant 16431
Messages.WM_QUIT 16432
Messages.WM_QUIT constant 16432
Messages.WM_RBUTTONDBLCLK 16432
Messages.WM_RBUTTONDBLCLK constant 16432
Messages.WM_RBUTTONDOWN 16432
Messages.WM_RBUTTONDOWN constant 16432
Messages.WM_RBUTTONUP 16433
Messages.WM_RBUTTONUP constant 16433
Messages.WM_RENDERALLFORMATS 16433
Messages.WM_RENDERALLFORMATS constant 16433
Messages.WM_RENDERFORMAT 16433
Messages.WM_RENDERFORMAT constant 16433
Messages.WM_SETCURSOR 16434
Messages.WM_SETCURSOR constant 16434
Messages.WM_SETFOCUS 16434
Messages.WM_SETFOCUS constant 16434
Messages.WM_SETFONT 16434
Messages.WM_SETFONT constant 16434
Messages.WM_SETHOTKEY 16435
Messages.WM_SETHOTKEY constant 16435
Messages.WM_SETICON 16435
Messages.WM_SETICON constant 16435
Messages.WM_SETREDRAW 16435
Messages.WM_SETREDRAW constant 16435
Messages.WM_SETTEXT 16436
Messages.WM_SETTEXT constant 16436
Messages.WM_SETTINGCHANGE 16436
Messages.WM_SETTINGCHANGE constant 16436
Messages.WM_SHOWWINDOW 16436
Messages.WM_SHOWWINDOW constant 16436
Messages.WM_SIZE 16437
Messages.WM_SIZE constant 16437
Messages.WM_SIZECLIPBOARD 16437
Messages.WM_SIZECLIPBOARD constant 16437
Messages.WM_SIZING 16437
Messages.WM_SIZING constant 16437
Messages.WM_SPOOLERSTATUS 16438
Messages.WM_SPOOLERSTATUS constant 16438
Messages.WM_STYLECHANGED 16438
Messages.WM_STYLECHANGED constant 16438

Messages.WM_STYLECHANGING 16438
Messages.WM_STYLECHANGING constant 16438
Messages.WM_SYSCHAR 16439
Messages.WM_SYSCHAR constant 16439
Messages.WM_SYSCOLORCHANGE 16439
Messages.WM_SYSCOLORCHANGE constant 16439
Messages.WM_SYSCOMMAND 16439
Messages.WM_SYSCOMMAND constant 16439
Messages.WM_SYSDEADCHAR 16440
Messages.WM_SYSDEADCHAR constant 16440
Messages.WM_SYSKEYDOWN 16440
Messages.WM_SYSKEYDOWN constant 16440
Messages.WM_SYSKEYUP 16440
Messages.WM_SYSKEYUP constant 16440
Messages.WM_SYSTEMERROR 16441
Messages.WM_SYSTEMERROR constant 16441
Messages.WM_TABLET_FIRST 16441
Messages.WM_TABLET_FIRST constant 16441
Messages.WM_TABLET_LAST 16441
Messages.WM_TABLET_LAST constant 16441
Messages.WM_TCARD 16442
Messages.WM_TCARD constant 16442
Messages.WM_THEMECHANGED 16442
Messages.WM_THEMECHANGED constant 16442
Messages.WM_TIMECHANGE 16442
Messages.WM_TIMECHANGE constant 16442
Messages.WM_TIMER 16443
Messages.WM_TIMER constant 16443
Messages.WM_UNDO 16443
Messages.WM_UNDO constant 16443
Messages.WM_UNICHAR 16443
Messages.WM_UNICHAR constant 16443
Messages.WM_UNINITMENUPOPUP 16444
Messages.WM_UNINITMENUPOPUP constant 16444
Messages.WM_UPDATEUISTATE 16444
Messages.WM_UPDATEUISTATE constant 16444
Messages.WM_USER 16444
Messages.WM_USER constant 16444
Messages.WM_USERCHANGED 16445
Messages.WM_USERCHANGED constant 16445
Messages.WM_VKEYTOITEM 16445
Messages.WM_VKEYTOITEM constant 16445
Messages.WM_VSCROLL 16445
Messages.WM_VSCROLL constant 16445
Messages.WM_VSCROLLCLIPBOARD 16446
Messages.WM_VSCROLLCLIPBOARD constant 16446
Messages.WM_WINDOWPOSCHANGED 16446
Messages.WM_WINDOWPOSCHANGED constant 16446
Messages.WM_WINDOWPOSCHANGING 16446
Messages.WM_WINDOWPOSCHANGING constant 16446
Messages.WM_WININICHANGE 16447
Messages.WM_WININICHANGE constant 16447
Messages.WM_WTSSESSION_CHANGE 16447
Messages.WM_WTSSESSION_CHANGE constant 16447
MetaDataCollectionName class 2160
- about MetaDataCollectionName class 2160
- Catalogs 2163
- ColumnConstraints 2163
- Columns 2163
- DataTypes 2164
- ForeignKeyColumns 2165
- ForeignKeys 2166
- IndexColumns 2167
- Indexes 2167
- MetaDataCollectionName fields 2162
- MetaDataCollectionName Members 2161
- PackageProcedureParameters 2168
- PackageProcedures 2168
- Packages 2169
- PackageSources 2169
- ProcedureParameters 2170
- Procedures 2171
- ProcedureSources 2171
- ReservedWords 2172
- Roles 2172
- Schemas 2172
- Synonyms 2173
- Tables 2173

Users 2174
Views 2174
mgColored enumeration member 17233
mgDisabled enumeration member 17233
mgEnabled enumeration member 17233
Midas 16449
Midas namespace 16449
Constants 16460
Interfaces 16450
Midas.IAppServer 16450
Midas.IAppServer.AS_ApplyUpdates 16451
Midas.IAppServer.AS_DataRequest 16452
Midas.IAppServer.AS_Execute 16453
Midas.IAppServer.AS_GetParams 16454
Midas.IAppServer.AS_GetProviderNames 16454
Midas.IAppServer.AS_GetRecords 16455
Midas.IAppServer.AS_RowRequest 16456
Midas.IAppServerDisp 16457
Midas.IAppServerDisp.AS_ApplyUpdates 16458
Midas.IAppServerDisp.AS_DataRequest 16458
Midas.IAppServerDisp.AS_Execute 16458
Midas.IAppServerDisp.AS_GetParams 16458
Midas.IAppServerDisp.AS_GetProviderNames 16459
Midas.IAppServerDisp.AS_GetRecords 16459
Midas.IAppServerDisp.AS_RowRequest 16459
Midas.IID_IAppServer 16460
Midas.IID_IAppServer constant 16460
Midas.LIBID_Midas 16460
Midas.LIBID_Midas constant 16460
MidasCon 16461
MidasCon namespace 16461
Classes 16462
Structs, Records, Enums 16472
MidasCon.TConnectType 16472
MidasCon.TConnectType enumeration 16472
MidasCon.TMidasConnection 16462
MidasCon.TMidasConnection.ConnectType 16469
MidasCon.TMidasConnection.Create 16466
MidasCon.TMidasConnection.DoConnect 16467
MidasCon.TMidasConnection.DoDisconnect 16468
MidasCon.TMidasConnection.ServerPort 16469
MidasCon.TMidasConnection.SetConnectType 16468
MidasCon.TMidasConnection.SetUseBroker 16468
MidasCon.TMidasConnection.UseBroker 16470
MidasCon.TRemoteServer 16470
MidItems 16473
MidItems namespace 16473
Classes 16478
Constants 17095
Functions 17085
Interfaces 17053
Structs, Records, Enums 17090
Types 17093
MidItems.AddBoolAttrib 17085
MidItems.AddBoolAttrib function 17085
MidItems.AddCustomAttrib 17085
MidItems.AddCustomAttrib function 17085
MidItems.AddIntAttrib 17086
MidItems.AddIntAttrib function 17086
MidItems.AddQuotedAttrib 17086
MidItems.AddQuotedAttrib function 17086
MidItems.AddStringAttrib 17086
MidItems.AddStringAttrib function 17086
MidItems.CanAddClassHelper 17087
MidItems.CanAddClassHelper function 17087
MidItems.CreateDefaultButtonClasses 17087
MidItems.CreateDefaultButtonClasses function 17087
MidItems.DeclareSubmitForm 17087
MidItems.DeclareSubmitForm function 17087
MidItems.FindDispatcher 17088
MidItems.FindDispatcher function 17088
MidItems.FindProducer 17088
MidItems.FindProducer function 17088
MidItems.FormatColumnData 17088
MidItems.FormatColumnData function 17088
MidItems.FormatColumnHeading 17089
MidItems.FormatColumnHeading function 17089
MidItems.FormMethodNames 17095

MidItems.FormMethodNames constant 17095
MidItems.GetItemValuesFromDataSet 17089
MidItems.GetItemValuesFromDataSet function 17089
MidItems.HTMLAlignValues 17096
MidItems.HTMLAlignValues constant 17096
MidItems.HTMLVAlignValues 17096
MidItems.HTMLVAlignValues constant 17096
MidItems.IDataSetComponent 17053
MidItems.IDataSetComponent.DataSet 17055
MidItems.IDataSetComponent.GetDataSet 17054
MidItems.IDataSetComponent.GetXMLBroker 17054
MidItems.IDataSetComponent.GetXMLDataSet 17055
MidItems.IDataSetComponent.XMLBroker 17056
MidItems.IDataSetComponent.XMLDataSet 17056
MidItems.IDataSetField 17057
MidItems.IDataSetField.FieldName 17058
MidItems.IDataSetField.GetFieldName 17057
MidItems.IDataSetField.GetRowSetFieldAttributes 17058
MidItems.IDataSetField.SetFieldName 17058
MidItems.IDataSetFields 17059
MidItems.IDataSetFields.GetFieldsList 17060
MidItems.IDataSetFields.GetVisibleFields 17060
MidItems.IDataSetFields.HasStatusField 17060
MidItems.IDataSetFields.IsFieldInUse 17060
MidItems.IDataSetFields.ValidateDataSet 17060
MidItems.IDataSetFields.VisibleFields 17061
MidItems.IFormatColumn 17061
MidItems.IFormatColumn.FormatColumnData 17062
MidItems.IFormatColumn.FormatColumnHeading 17062
MidItems.IHTMLField 17062
MidItems.IHTMLField.GetHTMLControlName 17063
MidItems.IHTMLField.HTMLControlName 17064
MidItems.IHTMLForm 17064
MidItems.IHTMLForm.GetHTMLFormName 17065
MidItems.IHTMLForm.GetHTMLFormVarName 17065
MidItems.IHTMLForm.HTMLFormName 17066
MidItems.IHTMLForm.HTMLFormVarName 17066
MidItems.IInetXWebDataSet 17066
MidItems.IInetXWebDataSet.AddOnDataChange 17068
MidItems.IInetXWebDataSet.FetchParams 17068
MidItems.IInetXWebDataSet.FieldCount 17068
MidItems.IInetXWebDataSet.Fields 17069
MidItems.IInetXWebDataSet.ParamCount 17069
MidItems.IInetXWebDataSet.Params 17069
MidItems.IInetXWebDataSet.RemoveOnDataChange 17070
MidItems.IMidasWebDataSet 17093
MidItems.IMidasWebDataSet type 17093
MidItems.IQueryField 17070
MidItems.IQueryField.GetParamName 17071
MidItems.IQueryField.GetText 17071
MidItems.IQueryField.ParamName 17072
MidItems.IQueryField.SetParamName 17071
MidItems.IQueryField.SetText 17072
MidItems.IQueryField.Text 17072
MidItems.IQueryFields 17073
MidItems.IQueryFields.GetParamsList 17073
MidItems.IQueryFields.IsParamInUse 17074
MidItems.IQueryFields.ValidateDataSet 17074
MidItems.IQueryForm 17074
MidItems.IRestoreDefaults 17074
MidItems.IRestoreDefaults.RestoreDefaults 17075
MidItems.IStatusField 17075
MidItems.IValidateField 17076
MidItems.IValidateField.ValidateField 17076
MidItems.IValidateFields 17077
MidItems.IValidateFields.EnableValidateFields 17078
MidItems.IValidateFields.GetEnableValidateFields 17078
MidItems.IValidateFields.SetEnableValidateFields 17078
MidItems.IValidateFields.ValidateFields 17078
MidItems.IWebDisplay 17079
MidItems.IWebDisplay.GetIsMultipleRecordView 17079
MidItems.IWebDisplay.IsMultipleRecordView 17080
MidItems.IWebDisplayReference 17080
MidItems.IWebDisplayReference.GetWebDisplayComponent 17081
MidItems.IWebDisplayReference.WebDisplayComponent 17081
MidItems.IXMLDisplay 17082
MidItems.IXMLDisplay.GetIsMultipleRecordView 17082
MidItems.IXMLDisplay.GetXMLDisplayName 17083

MidItems.IXMLDisplay.GetXMLRowSetName 17083
MidItems.IXMLDisplay.IsMultipleRecordView 17083
MidItems.IXMLDisplay.XMLDisplayName 17084
MidItems.IXMLDisplay.XMLRowSetName 17084
MidItems.sActionAttr 17096
MidItems.sActionAttr constant 17096
MidItems.sAlignAttr 17097
MidItems.sAlignAttr constant 17097
MidItems.sBgColorAttr 17097
MidItems.sBgColorAttr constant 17097
MidItems.sBorderAttr 17097
MidItems.sBorderAttr constant 17097
MidItems.sCellPaddingAttr 17098
MidItems.sCellPaddingAttr constant 17098
MidItems.sCellSpacingAttr 17098
MidItems.sCellSpacingAttr constant 17098
MidItems.sCheckedAttr 17098
MidItems.sCheckedAttr constant 17098
MidItems.sClassAttr 17099
MidItems.sClassAttr constant 17099
MidItems.sColsAttr 17099
MidItems.sColsAttr constant 17099
MidItems.sColSpanAttr 17099
MidItems.sColSpanAttr constant 17099
MidItems.sEncTypeAttr 17100
MidItems.sEncTypeAttr constant 17100
MidItems.sImageHeightAttr 17100
MidItems.sImageHeightAttr constant 17100
MidItems.sImageWidthAttr 17100
MidItems.sImageWidthAttr constant 17100
MidItems.sMaxLengthAttr 17101
MidItems.sMaxLengthAttr constant 17101
MidItems.sMethodAttr 17101
MidItems.sMethodAttr constant 17101
MidItems.sMultipleAttr 17101
MidItems.sMultipleAttr constant 17101
MidItems.sNameAttr 17102
MidItems.sNameAttr constant 17102
MidItems.sOnClickAttr 17102
MidItems.sOnClickAttr constant 17102
MidItems.sReadOnlyAttr 17102
MidItems.sReadOnlyAttr constant 17102
MidItems.sRowsAttr 17103
MidItems.sRowsAttr constant 17103
MidItems.sSelectedAttr 17103
MidItems.sSelectedAttr constant 17103
MidItems.sSizeAttr 17103
MidItems.sSizeAttr constant 17103
MidItems.sStyleAttr 17104
MidItems.sStyleAttr constant 17104
MidItems.sTabIndexAttr 17104
MidItems.sTabIndexAttr constant 17104
MidItems.sTypeAttr 17104
MidItems.sTypeAttr constant 17104
MidItems.sVAlignAttr 17105
MidItems.sVAlignAttr constant 17105
MidItems.sValueAttr 17105
MidItems.sValueAttr constant 17105
MidItems.sWidthAttr 17105
MidItems.sWidthAttr constant 17105
MidItems.sWrapAttr 17106
MidItems.sWrapAttr constant 17106
MidItems.TApplyUpdatesButton 16480
MidItems.TApplyUpdatesButton.AddElements 16485
MidItems.TApplyUpdatesButton.Caption 16486
MidItems.TApplyUpdatesButton.Create 16484
MidItems.TApplyUpdatesButton.Custom 16486
MidItems.TApplyUpdatesButton.GetSubComponents 16485
MidItems.TApplyUpdatesButton.ImplAddElements 16485
MidItems.TApplyUpdatesButton.ImplContent 16485
MidItems.TApplyUpdatesButton.Style 16487
MidItems.TApplyUpdatesButton.StyleRule 16487
MidItems.TApplyUpdatesButton.XMLBroker 16488
MidItems.TApplyUpdatesButton.XMLUseParent 16488
MidItems.TCaptionAttributes 16489
MidItems.TCaptionAttributes.Align 16491
MidItems.TCaptionAttributes.Create 16490
MidItems.TCaptionAttributes.Custom 16491

MidItems.TCaptionAttributes.Parent 16492
MidItems.TCaptionAttributes.Style 16492
MidItems.TCaptionAttributes.StyleRule 16492
MidItems.TCaptionAttributes.VAlign 16493
MidItems.TCaptionPosition 17090
MidItems.TCaptionPosition enumeration 17090
MidItems.TCustomDataForm 16493
MidItems.TCustomDataForm.GetHTMLFormTag 16498
MidItems.TCustomDataForm.ImplCanAddClass 16498
MidItems.TCustomDataForm.ImplContent 16499
MidItems.TCustomDataGrid 16499
MidItems.TCustomDataGrid.Create 16505
MidItems.TCustomDataGrid.Destroy 16506
MidItems.TCustomDataGrid.DisplayRows 16509
MidItems.TCustomDataGrid.FormatTable 16506
MidItems.TCustomDataGrid.HeadingAttributes 16509
MidItems.TCustomDataGrid.ImplAddElements 16506
MidItems.TCustomDataGrid.ImplCanAddClass 16507
MidItems.TCustomDataGrid.ImplContent 16507
MidItems.TCustomDataGrid.ImplIsMultipleRecordView 16508
MidItems.TCustomDataGrid.RowAttributes 16510
MidItems.TCustomDataGrid.SetHeadingAttributes 16508
MidItems.TCustomDataGrid.SetRowAttributes 16508
MidItems.TCustomDataGrid.SetTableAttributes 16508
MidItems.TCustomDataGrid.TableAttributes 16510
MidItems.TCustomDataNavigator 16511
MidItems.TCustomDataNavigator.Custom 16518
MidItems.TCustomDataNavigator.GetDefaultButtons 16516
MidItems.TCustomDataNavigator.ImplCanAddClass 16516
MidItems.TCustomDataNavigator.ImplContent 16516
MidItems.TCustomDataNavigator.ImplGetSubComponents 16517
MidItems.TCustomDataNavigator.Style 16518
MidItems.TCustomDataNavigator.StyleRule 16519
MidItems.TCustomDataNavigator.VisibleButtons 16517
MidItems.TCustomFieldGroup 16519
MidItems.TCustomFieldGroup.Create 16525
MidItems.TCustomFieldGroup.Custom 16527
MidItems.TCustomFieldGroup.Destroy 16525
MidItems.TCustomFieldGroup.GetLayoutAttributes 16526
MidItems.TCustomFieldGroup.ImplContent 16526
MidItems.TCustomFieldGroup.Style 16527
MidItems.TCustomFieldGroup.StyleRule 16528
MidItems.TCustomLayoutGroup 16528
MidItems.TCustomLayoutGroup.Create 16533
MidItems.TCustomLayoutGroup.Custom 16535
MidItems.TCustomLayoutGroup.Destroy 16533
MidItems.TCustomLayoutGroup.DisplayColumns 16536
MidItems.TCustomLayoutGroup.GetLayoutAttributes 16534
MidItems.TCustomLayoutGroup.ImplCanAddClass 16534
MidItems.TCustomLayoutGroup.ImplContent 16535
MidItems.TCustomLayoutGroup.Style 16536
MidItems.TCustomLayoutGroup.StyleRule 16537
MidItems.TCustomQueryButtons 16537
MidItems.TCustomQueryButtons.Custom 16544
MidItems.TCustomQueryButtons.GetDefaultButtons 16542
MidItems.TCustomQueryButtons.ImplAddElements 16542
MidItems.TCustomQueryButtons.ImplCanAddClass 16543
MidItems.TCustomQueryButtons.ImplContent 16543
MidItems.TCustomQueryButtons.Style 16544
MidItems.TCustomQueryButtons.StyleRule 16545
MidItems.TCustomQueryButtons.VisibleButtons 16543
MidItems.TCustomQueryForm 16545
MidItems.TCustomQueryForm.Action 16553
MidItems.TCustomQueryForm.AssignStrings 16551
MidItems.TCustomQueryForm.AssignStringsCallback 16551
MidItems.TCustomQueryForm.Create 16550
MidItems.TCustomQueryForm.GetHTMLFormTag 16551
MidItems.TCustomQueryForm.ImplAddElements 16552
MidItems.TCustomQueryForm.ImplCanAddClass 16552
MidItems.TCustomQueryForm.ImplContent 16552
MidItems.TCustomQueryForm.Method 16554
MidItems.TCustomQueryForm.MethodString 16553
MidItems.TDataForm 16554
MidItems.TDataForm.Custom 16559
MidItems.TDataForm.Style 16559
MidItems.TDataForm.StyleRule 16560
MidItems.TDataGrid 16560
MidItems.TDataGrid.DisplayRows 16567

MidItems.TDataGrid.HeadingAttributes 16567
MidItems.TDataGrid.RowAttributes 16568
MidItems.TDataGrid.TableAttributes 16568
MidItems.TDataNavigator 16569
MidItems.TDataNavigator.Custom 16574
MidItems.TDataNavigator.Style 16574
MidItems.TDataNavigator.StyleRule 16575
MidItems.TDataNavigator.XMLComponent 16575
MidItems.TDataSetButton 16576
MidItems.TDataSetButton.AddElements 16582
MidItems.TDataSetButton.Caption 16584
MidItems.TDataSetButton.Custom 16584
MidItems.TDataSetButton.DefaultCaption 16581
MidItems.TDataSetButton.GetCaption 16582
MidItems.TDataSetButton.GetMethodName 16582
MidItems.TDataSetButton.GetSubComponents 16582
MidItems.TDataSetButton.ImplAddElements 16583
MidItems.TDataSetButton.ImplContent 16583
MidItems.TDataSetButton.Style 16584
MidItems.TDataSetButton.StyleRule 16585
MidItems.TDataSetButton.XMLComponent 16585
MidItems.TDataSetButton.XMLMethodName 16581
MidItems.TDataSetButton.XMLUseParent 16586
MidItems.TDeleteButton 16586
MidItems.TDeleteButton.Create 16591
MidItems.TextAreaWrap 17106
MidItems.TextAreaWrap constant 17106
MidItems.TFieldGroup 16591
MidItems.TFieldGroup.Custom 16598
MidItems.TFieldGroup.ImplAddElements 16597
MidItems.TFieldGroup.ImplCanAddClass 16597
MidItems.TFieldGroup.Style 16598
MidItems.TFieldGroup.StyleRule 16599
MidItems.TFieldRadioGroup 16599
MidItems.TFieldRadioGroup.Caption 16606
MidItems.TFieldRadioGroup.CaptionAttributes 16606
MidItems.TFieldRadioGroup.CaptionPosition 16606
MidItems.TFieldRadioGroup.Custom 16607
MidItems.TFieldRadioGroup.DataSet 16607
MidItems.TFieldRadioGroup.DisplayColumns 16608
MidItems.TFieldRadioGroup.DisplayWidth 16608
MidItems.TFieldRadioGroup.FieldName 16608
MidItems.TFieldRadioGroup.Items 16609
MidItems.TFieldRadioGroup.ItemsField 16609
MidItems.TFieldRadioGroup.ReadOnly 16610
MidItems.TFieldRadioGroup.Style 16610
MidItems.TFieldRadioGroup.StyleRule 16611
MidItems.TFieldRadioGroup.TabIndex 16611
MidItems.TFieldRadioGroup.Values 16611
MidItems.TFieldRadioGroup.ValuesField 16612
MidItems.TFieldSelectOptions 16612
MidItems.TFieldSelectOptions.Caption 16618
MidItems.TFieldSelectOptions.CaptionAttributes 16618
MidItems.TFieldSelectOptions.CaptionPosition 16619
MidItems.TFieldSelectOptions.Custom 16619
MidItems.TFieldSelectOptions.DataSet 16620
MidItems.TFieldSelectOptions.DisplayRows 16620
MidItems.TFieldSelectOptions.FieldName 16621
MidItems.TFieldSelectOptions.Items 16621
MidItems.TFieldSelectOptions.ItemsField 16621
MidItems.TFieldSelectOptions.Style 16622
MidItems.TFieldSelectOptions.StyleRule 16622
MidItems.TFieldSelectOptions.TabIndex 16623
MidItems.TFieldSelectOptions.Values 16623
MidItems.TFieldSelectOptions.ValuesField 16623
MidItems.TFieldStatus 16624
MidItems.TFieldStatus.Caption 16629
MidItems.TFieldStatus.CaptionAttributes 16629
MidItems.TFieldStatus.CaptionPosition 16630
MidItems.TFieldStatus.Custom 16630
MidItems.TFieldStatus.Style 16630
MidItems.TFieldStatus.StyleRule 16631
MidItems.TFieldText 16631
MidItems.TFieldText.Caption 16637
MidItems.TFieldText.CaptionAttributes 16637
MidItems.TFieldText.CaptionPosition 16637
MidItems.TFieldText.Custom 16638
MidItems.TFieldText.DisplayWidth 16638

MidItems.TFieldText.FieldName 16639
MidItems.TFieldText.ReadOnly 16639
MidItems.TFieldText.Style 16639
MidItems.TFieldText.StyleRule 16640
MidItems.TFieldText.TabIndex 16640
MidItems.TFieldTextArea 16640
MidItems.TFieldTextArea.Caption 16646
MidItems.TFieldTextArea.CaptionAttributes 16647
MidItems.TFieldTextArea.CaptionPosition 16647
MidItems.TFieldTextArea.Custom 16647
MidItems.TFieldTextArea.DisplayRows 16648
MidItems.TFieldTextArea.DisplayWidth 16648
MidItems.TFieldTextArea.FieldName 16649
MidItems.TFieldTextArea.ReadOnly 16649
MidItems.TFieldTextArea.Style 16649
MidItems.TFieldTextArea.StyleRule 16650
MidItems.TFieldTextArea.TabIndex 16650
MidItems.TFieldTextArea.Wrap 16650
MidItems.TFirstButton 16651
MidItems.TFirstButton.Create 16655
MidItems.TFormLayout 16656
MidItems.TFormLayout.BreakButtons 16663
MidItems.TFormLayout.ColumnCount 16663
MidItems.TFormLayout.Create 16658
MidItems.TFormLayout.EndButtons 16659
MidItems.TFormLayout.EndFields 16659
MidItems.TFormLayout.EndLayout 16659
MidItems.TFormLayout.EndRow 16659
MidItems.TFormLayout.EndTable 16660
MidItems.TFormLayout.ImplLayoutButton 16660
MidItems.TFormLayout.ImplLayoutField 16660
MidItems.TFormLayout.ImplLayoutLabelAndField 16661
MidItems.TFormLayout.ImplLayoutTable 16661
MidItems.TFormLayout.NextColumn 16662
MidItems.TFormLayout.StartButtons 16662
MidItems.TFormLayout.StartFields 16662
MidItems.TFormLayout.StartRow 16662
MidItems.TFormLayout.StartTable 16663
MidItems.TFormLayout.TableHeader 16664
MidItems.TFormMethod 17091
MidItems.TFormMethod enumeration 17091
MidItems.TGridAttributes 16664
MidItems.TGridAttributes.Align 16668
MidItems.TGridAttributes.AssignTo 16667
MidItems.TGridAttributes.BgColor 16668
MidItems.TGridAttributes.Border 16669
MidItems.TGridAttributes.CellPadding 16669
MidItems.TGridAttributes.CellSpacing 16669
MidItems.TGridAttributes.Create 16666
MidItems.TGridAttributes.Custom 16670
MidItems.TGridAttributes.Parent 16670
MidItems.TGridAttributes.Style 16670
MidItems.TGridAttributes.StyleRule 16671
MidItems.TGridRowAttributes 16671
MidItems.TGridRowAttributes.Align 16674
MidItems.TGridRowAttributes.AssignTo 16673
MidItems.TGridRowAttributes.BgColor 16675
MidItems.TGridRowAttributes.Create 16673
MidItems.TGridRowAttributes.Custom 16675
MidItems.TGridRowAttributes.Parent 16676
MidItems.TGridRowAttributes.Style 16676
MidItems.TGridRowAttributes.StyleRule 16676
MidItems.TGridRowAttributes.VAlign 16677
MidItems.TInsertButton 16677
MidItems.TInsertButton.Create 16682
MidItems.TLastButton 16682
MidItems.TLastButton.Create 16687
MidItems.TLayoutGroup 16687
MidItems.TLayoutGroup.Custom 16692
MidItems.TLayoutGroup.DisplayColumns 16693
MidItems.TLayoutGroup.Style 16693
MidItems.TLayoutGroup.StyleRule 16694
MidItems.TLayoutState 17091
MidItems.TLayoutState enumeration 17091
MidItems.TNextButton 16694
MidItems.TNextButton.Create 16699
MidItems.TNextPageButton 16699
MidItems.TNextPageButton.Create 16703

MidItems.TPostButton 16704
MidItems.TPostButton.Create 16708
MidItems.TPriorButton 16709
MidItems.TPriorButton.Create 16713
MidItems.TPriorPageButton 16713
MidItems.TPriorPageButton.Create 16718
MidItems.TQueryButton 16718
MidItems.TQueryButton.Caption 16727
MidItems.TQueryButton.Create 16723
MidItems.TQueryButton.Custom 16728
MidItems.TQueryButton.DefaultCaption 16723
MidItems.TQueryButton.EventContent 16724
MidItems.TQueryButton.GetCaption 16724
MidItems.TQueryButton.GetHTMLControlName 16725
MidItems.TQueryButton.GetHTMLForm 16725
MidItems.TQueryButton.GetInputType 16725
MidItems.TQueryButton.GetXMLComponent 16725
MidItems.TQueryButton.ImplContent 16725
MidItems.TQueryButton.InputType 16723
MidItems.TQueryButton.Notification 16726
MidItems.TQueryButton.ParentXMLComponent 16726
MidItems.TQueryButton.SetXMLComponent 16727
MidItems.TQueryButton.SetXMLUseParent 16727
MidItems.TQueryButton.Style 16728
MidItems.TQueryButton.StyleRule 16729
MidItems.TQueryButton.XMLComponent 16729
MidItems.TQueryButton.XMLUseParent 16729
MidItems.TQueryButtons 16730
MidItems.TQueryButtons.Custom 16735
MidItems.TQueryButtons.Style 16735
MidItems.TQueryButtons.StyleRule 16736
MidItems.TQueryFieldGroup 16736
MidItems.TQueryFieldGroup.Custom 16744
MidItems.TQueryFieldGroup.FindParam 16743
MidItems.TQueryFieldGroup.GetParamsList 16743
MidItems.TQueryFieldGroup.ImplCanAddClass 16743
MidItems.TQueryFieldGroup.IsParamInUse 16744
MidItems.TQueryFieldGroup.Style 16745
MidItems.TQueryFieldGroup.StyleRule 16745
MidItems.TQueryForm 16745
MidItems.TQueryForm.Action 16751
MidItems.TQueryForm.Custom 16751
MidItems.TQueryForm.Method 16752
MidItems.TQueryForm.Style 16752
MidItems.TQueryForm.StyleRule 16752
MidItems.TQueryRadioGroup 16753
MidItems.TQueryRadioGroup.AddAttributes 16759
MidItems.TQueryRadioGroup.Caption 16761
MidItems.TQueryRadioGroup.CaptionAttributes 16761
MidItems.TQueryRadioGroup.CaptionPosition 16761
MidItems.TQueryRadioGroup.Custom 16762
MidItems.TQueryRadioGroup.DataSet 16762
MidItems.TQueryRadioGroup.DisplayColumns 16763
MidItems.TQueryRadioGroup.DisplayWidth 16763
MidItems.TQueryRadioGroup.FieldName 16764
MidItems.TQueryRadioGroup.GetCheckIndex 16759
MidItems.TQueryRadioGroup.GetText 16759
MidItems.TQueryRadioGroup.IsQueryField 16760
MidItems.TQueryRadioGroup.Items 16764
MidItems.TQueryRadioGroup.ItemsField 16765
MidItems.TQueryRadioGroup.ParamName 16765
MidItems.TQueryRadioGroup.ReadOnly 16766
MidItems.TQueryRadioGroup.SetText 16760
MidItems.TQueryRadioGroup.Style 16766
MidItems.TQueryRadioGroup.StyleRule 16766
MidItems.TQueryRadioGroup.TabIndex 16767
MidItems.TQueryRadioGroup.Text 16767
MidItems.TQueryRadioGroup.Values 16767
MidItems.TQueryRadioGroup.ValuesField 16768
MidItems.TQuerySelectOptions 16768
MidItems.TQuerySelectOptions.Caption 16776
MidItems.TQuerySelectOptions.CaptionAttributes 16777
MidItems.TQuerySelectOptions.CaptionPosition 16777
MidItems.TQuerySelectOptions.Custom 16777
MidItems.TQuerySelectOptions.DataSet 16778
MidItems.TQuerySelectOptions.DisplayRows 16778
MidItems.TQuerySelectOptions.FieldName 16779
MidItems.TQuerySelectOptions.GetSelectIndex 16775

MidItems.TQuerySelectOptions.GetText 16775
MidItems.TQuerySelectOptions.IsQueryField 16775
MidItems.TQuerySelectOptions.Items 16779
MidItems.TQuerySelectOptions.ItemsField 16780
MidItems.TQuerySelectOptions.ParamName 16780
MidItems.TQuerySelectOptions.SetText 16776
MidItems.TQuerySelectOptions.Style 16781
MidItems.TQuerySelectOptions.StyleRule 16781
MidItems.TQuerySelectOptions.TabIndex 16781
MidItems.TQuerySelectOptions.Text 16782
MidItems.TQuerySelectOptions.Values 16782
MidItems.TQuerySelectOptions.ValuesField 16782
MidItems.TQueryText 16783
MidItems.TQueryText.AddAttributes 16789
MidItems.TQueryText.Caption 16790
MidItems.TQueryText.CaptionAttributes 16791
MidItems.TQueryText.CaptionPosition 16791
MidItems.TQueryText.Create 16788
MidItems.TQueryText.Custom 16791
MidItems.TQueryText.DisplayWidth 16792
MidItems.TQueryText.FieldName 16792
MidItems.TQueryText.GetText 16789
MidItems.TQueryText.IsQueryField 16789
MidItems.TQueryText.MaxWidth 16793
MidItems.TQueryText.ParamName 16793
MidItems.TQueryText.ReadOnly 16793
MidItems.TQueryText.SetText 16790
MidItems.TQueryText.Style 16794
MidItems.TQueryText.StyleRule 16794
MidItems.TQueryText.TabIndex 16794
MidItems.TQueryText.Text 16795
MidItems.TQueryTextArea 16795
MidItems.TQueryTextArea.AddAttributes 16801
MidItems.TQueryTextArea.Caption 16803
MidItems.TQueryTextArea.CaptionAttributes 16804
MidItems.TQueryTextArea.CaptionPosition 16804
MidItems.TQueryTextArea.ControlContent 16801
MidItems.TQueryTextArea.Create 16800
MidItems.TQueryTextArea.Custom 16804
MidItems.TQueryTextArea.Destroy 16801
MidItems.TQueryTextArea.DisplayRows 16805
MidItems.TQueryTextArea.DisplayWidth 16805
MidItems.TQueryTextArea.FieldName 16805
MidItems.TQueryTextArea.GetText 16802
MidItems.TQueryTextArea.IsQueryField 16802
MidItems.TQueryTextArea.Lines 16806
MidItems.TQueryTextArea.ParamName 16806
MidItems.TQueryTextArea.ReadOnly 16807
MidItems.TQueryTextArea.SetLines 16802
MidItems.TQueryTextArea.SetText 16803
MidItems.TQueryTextArea.Style 16807
MidItems.TQueryTextArea.StyleRule 16808
MidItems.TQueryTextArea.TabIndex 16808
MidItems.TQueryTextArea.Wrap 16808
MidItems.TResetQueryButton 16809
MidItems.TResetQueryButton.Create 16813
MidItems.TResetQueryButton.EventContent 16813
MidItems.TSelectOptionsColumn 16814
MidItems.TSelectOptionsColumn.Caption 16820
MidItems.TSelectOptionsColumn.CaptionAttributes 16821
MidItems.TSelectOptionsColumn.Custom 16821
MidItems.TSelectOptionsColumn.DataSet 16822
MidItems.TSelectOptionsColumn.DisplayRows 16822
MidItems.TSelectOptionsColumn.FieldName 16822
MidItems.TSelectOptionsColumn.IsColumn 16819
MidItems.TSelectOptionsColumn.Items 16823
MidItems.TSelectOptionsColumn.ItemsField 16823
MidItems.TSelectOptionsColumn.Style 16824
MidItems.TSelectOptionsColumn.StyleRule 16824
MidItems.TSelectOptionsColumn.TabIndex 16825
MidItems.TSelectOptionsColumn.Values 16825
MidItems.TSelectOptionsColumn.ValuesField 16825
MidItems.TStatusColumn 16826
MidItems.TStatusColumn.Caption 16831
MidItems.TStatusColumn.CaptionAttributes 16832
MidItems.TStatusColumn.Custom 16832
MidItems.TStatusColumn.IsColumn 16831
MidItems.TStatusColumn.Style 16832

MidItems.TStatusColumn.StyleRule 16833
MidItems.TSubmitQueryButton 16833
MidItems.TSubmitQueryButton.Create 16838
MidItems.TSubmitQueryButton.EventContent 16838
MidItems.TTextAreaColumn 16838
MidItems.TTextAreaColumn.Caption 16845
MidItems.TTextAreaColumn.CaptionAttributes 16845
MidItems.TTextAreaColumn.Custom 16846
MidItems.TTextAreaColumn.DisplayRows 16846
MidItems.TTextAreaColumn.DisplayWidth 16846
MidItems.TTextAreaColumn.FieldName 16847
MidItems.TTextAreaColumn.IsColumn 16844
MidItems.TTextAreaColumn.ReadOnly 16847
MidItems.TTextAreaColumn.Style 16847
MidItems.TTextAreaColumn.StyleRule 16848
MidItems.TTextAreaColumn.TabIndex 16848
MidItems.TTextAreaColumn.Wrap 16848
MidItems.TTextAreaWrap 17092
MidItems.TTextAreaWrap enumeration 17092
MidItems.TTextColumn 16849
MidItems.TTextColumn.Caption 16855
MidItems.TTextColumn.CaptionAttributes 16855
MidItems.TTextColumn.Custom 16856
MidItems.TTextColumn.DisplayWidth 16856
MidItems.TTextColumn.FieldName 16857
MidItems.TTextColumn.IsColumn 16854
MidItems.TTextColumn.ReadOnly 16857
MidItems.TTextColumn.Style 16857
MidItems.TTextColumn.StyleRule 16858
MidItems.TTextColumn.TabIndex 16858
MidItems.TUndoButton 16858
MidItems.TUndoButton.Create 16863
MidItems.TWebButton 16863
MidItems.TWebButton.Caption 16870
MidItems.TWebButton.Content 16868
MidItems.TWebButton.Create 16868
MidItems.TWebButton.Custom 16871
MidItems.TWebButton.GetCaption 16869
MidItems.TWebButton.GetLayoutAttributes 16869
MidItems.TWebButton.ImplContent 16869
MidItems.TWebButton.SetCaption 16870
MidItems.TWebButton.Style 16871
MidItems.TWebButton.StyleRule 16872
MidItems.TWebButtonClass 17093
MidItems.TWebButtonClass type 17093
MidItems.TWebControlGroup 16872
MidItems.TWebControlGroup.AddElements 16878
MidItems.TWebControlGroup.CanAddClass 16878
MidItems.TWebControlGroup.ChangeFieldControls 16879
MidItems.TWebControlGroup.Content 16879
MidItems.TWebControlGroup.Create 16877
MidItems.TWebControlGroup.DefaultWebComponents 16883
MidItems.TWebControlGroup.Destroy 16877
MidItems.TWebControlGroup.GetDefaultWebComponents 16880
MidItems.TWebControlGroup.GetLayoutAttributes 16880
MidItems.TWebControlGroup.GetSubComponents 16880
MidItems.TWebControlGroup.ImplAddElements 16881
MidItems.TWebControlGroup.ImplCanAddClass 16881
MidItems.TWebControlGroup.ImplContent 16882
MidItems.TWebControlGroup.ImplGetSubComponents 16882
MidItems.TWebControlGroup.SetDefaultWebComponents 16883
MidItems.TWebControlGroup.WebFieldControls 16884
MidItems.TWebDataDisplay 16884
MidItems.TWebDataDisplay.Caption 16897
MidItems.TWebDataDisplay.CaptionAttributes 16897
MidItems.TWebDataDisplay.CaptionPosition 16897
MidItems.TWebDataDisplay.Content 16890
MidItems.TWebDataDisplay.ControlContent 16891
MidItems.TWebDataDisplay.Create 16889
MidItems.TWebDataDisplay.Custom 16898
MidItems.TWebDataDisplay.Destroy 16889
MidItems.TWebDataDisplay.EventContent 16891
MidItems.TWebDataDisplay.FormatCaption 16891
MidItems.TWebDataDisplay.GetCaption 16891
MidItems.TWebDataDisplay.GetHTMLControlName 16892
MidItems.TWebDataDisplay.GetHTMLForm 16892
MidItems.TWebDataDisplay.GetLayoutAttributes 16892

MidItems.TWebDataDisplay.GetXmlDisplayName 16893
MidItems.TWebDataDisplay.GetXmlRowSetName 16893
MidItems.TWebDataDisplay.ImplContent 16894
MidItems.TWebDataDisplay.ImplGetHTMLControlName 16894
MidItems.TWebDataDisplay.IsColumn 16895
MidItems.TWebDataDisplay.IsQueryField 16895
MidItems.TWebDataDisplay.LabelContent 16895
MidItems.TWebDataDisplay.LayoutAttributes 16898
MidItems.TWebDataDisplay.SetCaption 16896
MidItems.TWebDataDisplay.SetCaptionAttributes 16896
MidItems.TWebDataDisplay.Style 16899
MidItems.TWebDataDisplay.StyleRule 16899
MidItems.TWebDataDisplay.TabIndex 16899
MidItems.TWebDataDisplayClass 17094
MidItems.TWebDataDisplayClass type 17094
MidItems.TWebDataInput 16900
MidItems.TWebDataInput.EventContent 16905
MidItems.TWebDataInput.FieldName 16909
MidItems.TWebDataInput.FindAssociatedField 16906
MidItems.TWebDataInput.GetCaption 16906
MidItems.TWebDataInput.GetFieldName 16906
MidItems.TWebDataInput.GetParamName 16906
MidItems.TWebDataInput.GetRowSetFieldAttributes 16907
MidItems.TWebDataInput.ImplGetFieldName 16907
MidItems.TWebDataInput.ImplGetHTMLControlName 16907
MidItems.TWebDataInput.ImplGetRowSetFieldAttributes 16907
MidItems.TWebDataInput.ImplRestoreDefaults 16908
MidItems.TWebDataInput.ParamName 16910
MidItems.TWebDataInput.RestoreDefaults 16908
MidItems.TWebDataInput.RestoreFieldDefaults 16908
MidItems.TWebDataInput.SetCaption 16908
MidItems.TWebDataInput.SetFieldName 16909
MidItems.TWebDataInput.SetParamName 16909
MidItems.TWebDataInput.ValidateField 16909
MidItems.TWebDataInputClass 17094
MidItems.TWebDataInputClass type 17094
MidItems.TWebDisplay 16911
MidItems.TWebDisplay.Changed 16912
MidItems.TWebDisplay.ChangedDisplayComponent 16913
MidItems.TWebDisplay.Create 16912
MidItems.TWebDisplay.DisplayComponent 16914
MidItems.TWebDisplay.GetDisplayComponent 16913
MidItems.TWebDisplay.Notification 16913
MidItems.TWebDisplay.OnChange 16914
MidItems.TWebDisplay.Parent 16914
MidItems.TWebDisplay.SetDisplayComponent 16913
MidItems.TWebDisplayReferenceGroup 16915
MidItems.TWebDisplayReferenceGroup.Create 16919
MidItems.TWebDisplayReferenceGroup.Destroy 16919
MidItems.TWebDisplayReferenceGroup.GetWebDisplayCompo
nent
16920
MidItems.TWebDisplayReferenceGroup.Notification 16920
MidItems.TWebDisplayReferenceGroup.SetWebDisplayCompon
ent
16921
MidItems.TWebDisplayReferenceGroup.WebDisplay 16921
MidItems.TWebDisplayReferenceGroup.WebDisplayComponent
16921
MidItems.TWebForm 16922
MidItems.TWebForm.AddElements 16927
MidItems.TWebForm.CanAddClass 16928
MidItems.TWebForm.Content 16928
MidItems.TWebForm.Create 16926
MidItems.TWebForm.Custom 16933
MidItems.TWebForm.Destroy 16926
MidItems.TWebForm.GetDefaultWebComponents 16929
MidItems.TWebForm.GetHTMLFormName 16929
MidItems.TWebForm.GetHTMLFormVarName 16929
MidItems.TWebForm.GetLayoutAttributes 16930
MidItems.TWebForm.GetSubComponents 16930
MidItems.TWebForm.GetVisibleFields 16930
MidItems.TWebForm.ImplAddElements 16931
MidItems.TWebForm.ImplCanAddClass 16931
MidItems.TWebForm.ImplContent 16932
MidItems.TWebForm.SetDefaultWebComponents 16932
MidItems.TWebForm.Style 16933
MidItems.TWebForm.StyleRule 16934
MidItems.TWebForm.VisibleFields 16934

MidItems.TWebForm.WebComponents 16934
MidItems.TWebListInput 16935
MidItems.TWebListInput.AddElements 16941
MidItems.TWebListInput.ControlContent 16942
MidItems.TWebListInput.Create 16940
MidItems.TWebListInput.DataSet 16944
MidItems.TWebListInput.Destroy 16941
MidItems.TWebListInput.FormatInputs 16942
MidItems.TWebListInput.GetSubComponents 16942
MidItems.TWebListInput.ImplAddElements 16942
MidItems.TWebListInput.Items 16945
MidItems.TWebListInput.ItemsField 16945
MidItems.TWebListInput.Notification 16943
MidItems.TWebListInput.SetDataSet 16943
MidItems.TWebListInput.SetItems 16943
MidItems.TWebListInput.SetValues 16944
MidItems.TWebListInput.Values 16946
MidItems.TWebListInput.ValuesField 16946
MidItems.TWebRadioGroupInput 16947
MidItems.TWebRadioGroupInput.AddAttributes 16953
MidItems.TWebRadioGroupInput.Create 16952
MidItems.TWebRadioGroupInput.DisplayColumns 16954
MidItems.TWebRadioGroupInput.DisplayWidth 16955
MidItems.TWebRadioGroupInput.EventContent 16953
MidItems.TWebRadioGroupInput.FormatInputs 16953
MidItems.TWebRadioGroupInput.GetCheckIndex 16954
MidItems.TWebRadioGroupInput.ImplGetRowSetFieldAttributes 16954
MidItems.TWebRadioGroupInput.ReadOnly 16955
MidItems.TWebRadioGroupInput.RestoreFieldDefaults 16954
MidItems.TWebSelectOptionsInput 16956
MidItems.TWebSelectOptionsInput.AddAttributes 16962
MidItems.TWebSelectOptionsInput.Create 16961
MidItems.TWebSelectOptionsInput.DisplayRows 16963
MidItems.TWebSelectOptionsInput.EventContent 16962
MidItems.TWebSelectOptionsInput.FormatInputs 16962
MidItems.TWebSelectOptionsInput.GetSelectIndex 16962
MidItems.TWebStatus 16963
MidItems.TWebStatus.Caption 16970
MidItems.TWebStatus.CaptionAttributes 16970
MidItems.TWebStatus.CaptionPosition 16971
MidItems.TWebStatus.ControlContent 16969
MidItems.TWebStatus.Create 16968
MidItems.TWebStatus.DisplayWidth 16971
MidItems.TWebStatus.GetCaption 16969
MidItems.TWebStatus.Identifier 16969
MidItems.TWebStatus.ImplGetHTMLControlName 16969
MidItems.TWebStatusClass 17094
MidItems.TWebStatusClass type 17094
MidItems.TWebTextAreaInput 16971
MidItems.TWebTextAreaInput.AddAttributes 16977
MidItems.TWebTextAreaInput.ControlContent 16978
MidItems.TWebTextAreaInput.Create 16977
MidItems.TWebTextAreaInput.DisplayRows 16979
MidItems.TWebTextAreaInput.DisplayWidth 16979
MidItems.TWebTextAreaInput.EventContent 16978
MidItems.TWebTextAreaInput.ImplGetRowSetFieldAttributes 16978
MidItems.TWebTextAreaInput.ReadOnly 16979
MidItems.TWebTextAreaInput.RestoreFieldDefaults 16978
MidItems.TWebTextAreaInput.Wrap 16980
MidItems.TWebTextInput 16980
MidItems.TWebTextInput.AddAttributes 16986
MidItems.TWebTextInput.ControlContent 16987
MidItems.TWebTextInput.Create 16986
MidItems.TWebTextInput.Custom 16987
MidItems.TWebTextInput.DisplayWidth 16988
MidItems.TWebTextInput.ImplGetRowSetFieldAttributes 16987
MidItems.TWebTextInput.ReadOnly 16988
MidItems.TWebTextInput.RestoreFieldDefaults 16987
MidItems.TWebTextInput.Style 16989
MidItems.TWebTextInput.StyleRule 16989
MidItems.TXMLButton 16989
MidItems.TXMLButton.Create 16993
MidItems.TXMLButton.DefaultCaption 16996
MidItems.TXMLButton.Destroy 16994
MidItems.TXMLButton.GetCaption 16994
MidItems.TXMLButton.GetXMLBroker 16995
MidItems.TXMLButton.GetXMLUseParent 16995
MidItems.TXMLButton.Notification 16995

MidItems.TXMLButton.SetXMLBroker 16996
MidItems.TXMLButton.SetXMLUseParent 16996
MidItems.TXMLButton.XMLBroker 16996
MidItems.TXMLButton.XMLData 16997
MidItems.TXMLButton.XMLUseParent 16997
MidItems.TXMLData 16998
MidItems.TXMLData.Changed 17000
MidItems.TXMLData.ChangedXMLBroker 17000
MidItems.TXMLData.Create 16999
MidItems.TXMLData.GetXMLBroker 17000
MidItems.TXMLData.Notification 17001
MidItems.TXMLData.OnChange 17002
MidItems.TXMLData.Parent 17001
MidItems.TXMLData.SetXMLBroker 17001
MidItems.TXMLData.XMLBroker 17002
MidItems.TXMLDataParent 17003
MidItems.TXMLDataParent.ChangedUseParent 17004
MidItems.TXMLDataParent.ChangedXMLBroker 17005
MidItems.TXMLDataParent.GetXMLBroker 17005
MidItems.TXMLDataParent.ParentXMLBroker 17005
MidItems.TXMLDataParent.SetUseParent 17005
MidItems.TXMLDataParent.UseParent 17006
MidItems.TXMLDataSet 17006
MidItems.TXMLDataSet._AddRef 17009
MidItems.TXMLDataSet._Release 17009
MidItems.TXMLDataSet.ChangedDataSetField 17009
MidItems.TXMLDataSet.ChangedXMLBroker 17010
MidItems.TXMLDataSet.CompareXMLData 17010
MidItems.TXMLDataSet.ConnectionChange 17010
MidItems.TXMLDataSet.Create 17008
MidItems.TXMLDataSet.CreateDataSet 17010
MidItems.TXMLDataSet.DataSet 17012
MidItems.TXMLDataSet.DataSetField 17013
MidItems.TXMLDataSet.DataSetPath 17013
MidItems.TXMLDataSet.Destroy 17008
MidItems.TXMLDataSet.FindDataSet 17010
MidItems.TXMLDataSet.GetDataSet 17011
MidItems.TXMLDataSet.GetDataSetField 17011
MidItems.TXMLDataSet.GetDataSetPath 17011
MidItems.TXMLDataSet.Parent 17014
MidItems.TXMLDataSet.QueryInterface 17011
MidItems.TXMLDataSet.SetDataSetField 17012
MidItems.TXMLDataSet.SetXMLBroker 17012
MidItems.TXMLDataSet.WrapperDataChange 17012
MidItems.TXMLDataSetParent 17014
MidItems.TXMLDataSetParent.ChangedDataSetField 17017
MidItems.TXMLDataSetParent.ChangedUseParent 17017
MidItems.TXMLDataSetParent.ChangedXMLBroker 17017
MidItems.TXMLDataSetParent.GetDataSetField 17017
MidItems.TXMLDataSetParent.GetXMLBroker 17017
MidItems.TXMLDataSetParent.ParentDataSetField 17018
MidItems.TXMLDataSetParent.ParentXMLBroker 17018
MidItems.TXMLDataSetParent.ParentXMLDataSet 17018
MidItems.TXMLDataSetParent.SetUseParent 17018
MidItems.TXMLDataSetParent.UseParent 17019
MidItems.TXMLDisplayGroup 17019
MidItems.TXMLDisplayGroup.Create 17024
MidItems.TXMLDisplayGroup.CreateDefaultFields 17025
MidItems.TXMLDisplayGroup.Destroy 17024
MidItems.TXMLDisplayGroup.EnableValidateFields 17032
MidItems.TXMLDisplayGroup.FindField 17026
MidItems.TXMLDisplayGroup.GetDataSet 17026
MidItems.TXMLDisplayGroup.GetDataSetField 17026
MidItems.TXMLDisplayGroup.GetEnableValidateFields 17026
MidItems.TXMLDisplayGroup.GetFieldsList 17027
MidItems.TXMLDisplayGroup.GetIsMultipleRecordView 17027
MidItems.TXMLDisplayGroup.GetVisibleFields 17027
MidItems.TXMLDisplayGroup.GetXMLBroker 17027
MidItems.TXMLDisplayGroup.GetXMLDataSet 17028
MidItems.TXMLDisplayGroup.GetXMLDisplayName 17028
MidItems.TXMLDisplayGroup.GetXMLRowSetName 17028
MidItems.TXMLDisplayGroup.HasStatusField 17028
MidItems.TXMLDisplayGroup.ImplAddElements 17029
MidItems.TXMLDisplayGroup.ImplGetSubComponents 17029
MidItems.TXMLDisplayGroup.ImplIsMultipleRecordView 17029
MidItems.TXMLDisplayGroup.IsFieldInUse 17030
MidItems.TXMLDisplayGroup.Notification 17030
MidItems.TXMLDisplayGroup.SetDataSetField 17030

MidItems.TXMLDisplayGroup.SetEnableValidateFields 17031
MidItems.TXMLDisplayGroup.SetXMLBroker 17031
MidItems.TXMLDisplayGroup.ValidateDataSet 17031
MidItems.TXMLDisplayGroup.ValidateFields 17031
MidItems.TXMLDisplayGroup.VisibleFields 17032
MidItems.TXMLDisplayGroup.XMLBroker 17033
MidItems.TXMLDisplayGroup.XMLDataChange 17031
MidItems.TXMLDisplayGroup.XMLDataSet 17033
MidItems.TXMLDisplayGroup.XMLDataSetField 17033
MidItems.TXMLDisplayParent 17034
MidItems.TXMLDisplayParent.ChangedDisplayComponent 17036
MidItems.TXMLDisplayParent.ChangedUseParent 17036
MidItems.TXMLDisplayParent.Create 17036
MidItems.TXMLDisplayParent.GetDisplayComponent 17037
MidItems.TXMLDisplayParent.SetUseParent 17037
MidItems.TXMLDisplayParent.UseParent 17037
MidItems.TXMLDisplayReferenceButton 17037
MidItems.TXMLDisplayReferenceButton.Create 17042
MidItems.TXMLDisplayReferenceButton.Destroy 17042
MidItems.TXMLDisplayReferenceButton.GetDisplayComponent 17043
MidItems.TXMLDisplayReferenceButton.GetDisplayComponent Parent 17043
MidItems.TXMLDisplayReferenceButton.GetXmlDisplayName 17043
MidItems.TXMLDisplayReferenceButton.Notification 17044
MidItems.TXMLDisplayReferenceButton.SetDisplayComponent 17044
MidItems.TXMLDisplayReferenceButton.SetDisplayComponent Parent 17044
MidItems.TXMLDisplayReferenceButton.XMLComponent 17045
MidItems.TXMLDisplayReferenceButton.XMLDisplay 17045
MidItems.TXMLDisplayReferenceButton.XMLUseParent 17045
MidItems.TXMLDisplayReferenceGroup 17046
MidItems.TXMLDisplayReferenceGroup.ImplAddElements 17051
MidItems.TXMLDisplayReferenceGroup.XMLComponent 17051
MidProd 17107
MidProd namespace 17107
Classes 17109
Constants 17164
Functions 17160
Structs, Records, Enums 17161
Types 17162
MidProd.AddStdIncludes 17160
MidProd.AddStdIncludes function 17160
MidProd.PathInfoToRelativePath 17160
MidProd.PathInfoToRelativePath function 17160
MidProd.sXMLBroker 17164
MidProd.sXMLBroker constant 17164
MidProd.sXMLReadyVar 17164
MidProd.sXMLReadyVar constant 17164
MidProd.TBaseInetXPageProducer 17109
MidProd.TBaseInetXPageProducer.AddIncludes 17116
MidProd.TBaseInetXPageProducer.AddScriptComponents 17117
MidProd.TBaseInetXPageProducer.CanAddClass 17117
MidProd.TBaseInetXPageProducer.ContentFromStream 17117
MidProd.TBaseInetXPageProducer.Create 17115
MidProd.TBaseInetXPageProducer.CreatePageElements 17118
MidProd.TBaseInetXPageProducer.CreateScriptManager 17118
MidProd.TBaseInetXPageProducer.DeclareFunctions 17118
MidProd.TBaseInetXPageProducer.DeclareIncludes 17118
MidProd.TBaseInetXPageProducer.DeclareRowSets 17119
MidProd.TBaseInetXPageProducer.DeclareScript 17119
MidProd.TBaseInetXPageProducer.DeclareStyles 17119
MidProd.TBaseInetXPageProducer.DeclareWarnings 17119
MidProd.TBaseInetXPageProducer.DeclareXMLDocuments 17120
MidProd.TBaseInetXPageProducer.DeclareXMLIslands 17120
MidProd.TBaseInetXPageProducer.Destroy 17115
MidProd.TBaseInetXPageProducer.DoAfterGetContent 17120
MidProd.TBaseInetXPageProducer.DoAfterGetXMLData 17120
MidProd.TBaseInetXPageProducer.DoBeforeGetContent 17120
MidProd.TBaseInetXPageProducer.DoBeforeGetXMLData 17121
MidProd.TBaseInetXPageProducer.DoTagEvent 17121
MidProd.TBaseInetXPageProducer.EnableXMLIslands 17126
MidProd.TBaseInetXPageProducer.FindXMLBroker 17121

MidProd.TBaseInetXPageProducer.FindXMLBrokerName 17122
MidProd.TBaseInetXPageProducer.GeneratePageElements 17122
MidProd.TBaseInetXPageProducer.GetContentOptions 17122
MidProd.TBaseInetXPageProducer.GetDefaultTemplate 17122
MidProd.TBaseInetXPageProducer.GetRequiredTags 17123
MidProd.TBaseInetXPageProducer.GetScriptManager 17123
MidProd.TBaseInetXPageProducer.GetUseXMLIslands 17123
MidProd.TBaseInetXPageProducer.GetXMLData 17124
MidProd.TBaseInetXPageProducer.GetXMLDelta 17124
MidProd.TBaseInetXPageProducer.GetXMLErrors 17124
MidProd.TBaseInetXPageProducer.ImplCanAddClass 17124
MidProd.TBaseInetXPageProducer.IncludePathURL 17127
MidProd.TBaseInetXPageProducer.OnAfterGetContent 17129
MidProd.TBaseInetXPageProducer.OnAfterGetXMLData 17129
MidProd.TBaseInetXPageProducer.OnBeforeGetContent 17130
MidProd.TBaseInetXPageProducer.OnBeforeGetXMLData 17130
MidProd.TBaseInetXPageProducer.PageElements 17127
MidProd.TBaseInetXPageProducer.ScriptManager 17127
MidProd.TBaseInetXPageProducer.SetIncludePathURL 17125
MidProd.TBaseInetXPageProducer.SetStyles 17125
MidProd.TBaseInetXPageProducer.SetStylesFile 17125
MidProd.TBaseInetXPageProducer.SetWebContentOptions 17126
MidProd.TBaseInetXPageProducer.Styles 17127
MidProd.TBaseInetXPageProducer.StylesFile 17128
MidProd.TBaseMidasPageProducer 17162
MidProd.TBaseMidasPageProducer type 17162
MidProd.TCustomInetXPageProducer 17131
MidProd.TCustomInetXPageProducer.Content 17138
MidProd.TCustomInetXPageProducer.Create 17137
MidProd.TCustomInetXPageProducer.Destroy 17137
MidProd.TCustomInetXPageProducer.GetTemplateFileName 17139
MidProd.TCustomInetXPageProducer.GetTemplateStream 17139
MidProd.TCustomInetXPageProducer.HTMLDoc 17140
MidProd.TCustomInetXPageProducer.HTMLFile 17141
MidProd.TCustomInetXPageProducer.WebContent 17140
MidProd.TCustomMidasPageProducer 17162
MidProd.TCustomMidasPageProducer type 17162
MidProd.TInetXPageElements 17142
MidProd.TInetXPageElements.BodyContent 17144
MidProd.TInetXPageElements.FormsContent 17143
MidProd.TInetXPageElements.IncludesContent 17143
MidProd.TInetXPageElements.ScriptContent 17144
MidProd.TInetXPageElements.StylesContent 17144
MidProd.TInetXPageElements.WarningsContent 17144
MidProd.TInetXPageProducer 17145
MidProd.TInetXPageProducer.EnableXMLIslands 17151
MidProd.TInetXPageProducer.HTMLDoc 17152
MidProd.TInetXPageProducer.HTMLFile 17153
MidProd.TInetXPageProducer.IncludePathURL 17153
MidProd.TInetXPageProducer.OnAfterGetContent 17156
MidProd.TInetXPageProducer.OnAfterGetXMLData 17156
MidProd.TInetXPageProducer.OnBeforeGetContent 17157
MidProd.TInetXPageProducer.OnBeforeGetXMLData 17157
MidProd.TInetXPageProducer.ScriptEngine 17154
MidProd.TInetXPageProducer.Styles 17154
MidProd.TInetXPageProducer.StylesFile 17155
MidProd.TInetXPageProducer.WebPageItems 17155
MidProd.TMidasPageElements 17163
MidProd.TMidasPageElements type 17163
MidProd.TMidasPageProducer 17158
MidProd.TStdInclude 17161
MidProd.TStdInclude enumeration 17161
MidProd.TStdIncludes 17163
MidProd.TStdIncludes type 17163
MidProd.TXMLDataEvent 17163
MidProd.TXMLDataEvent type 17163
MirrorStatusColumns class 2069
- about MirrorStatusColumns class 2069
- AttachedAutoFailOverCount 2071
- AttachedInstantSynchCount 2071
- ConnectError 2071
- HighestRedoStamp 2071
- MirrorReplayStamp 2072
- MirrorStatusColumns fields 2071
- MirrorStatusColumns Members 2070

TotalAutoFailOverCount 2072
TotalInstantSynchCount 2072
MirrorTypes class 2072
about MirrorTypes class 2072
Directory 2073
MirrorTypes fields 2073
MirrorTypes Members 2073
None 2074
Primary 2074
Readonly 2074
mkAmbiguous enumeration member 22998
mkClassConstructor enumeration member 23373
mkClassFunction enumeration member 23373
mkClassProcedure enumeration member 23373
mkConstructor enumeration member 23373
mkDestructor enumeration member 23373
mkExactMatch enumeration member 22998
mkFunction enumeration member 23373
mkNone enumeration member 22998
mkOperatorOverload enumeration member 23373
mkProcedure enumeration member 23373
mkSafeFunction enumeration member 23373
mkSafeProcedure enumeration member 23373
mkSingleMatch enumeration member 22998
moMarshalAll enumeration member 1301
moMarshalModifiedOnly enumeration member 1301
mpCanEject enumeration member 17231
mpCanPlay enumeration member 17231
mpCanRecord enumeration member 17231
mpCanStep enumeration member 17231
MPlayer 17165
MPlayer namespace 17165
Classes 17167
Structs, Records, Enums 17229
Types 17237
MPlayer.EMCIDeviceError 17167
MPlayer.EMPNotify 17237
MPlayer.EMPNotify type 17237
MPlayer.EMPPostNotify 17237
MPlayer.EMPPostNotify type 17237
MPlayer.TButtonSet 17238
MPlayer.TButtonSet type 17238
MPlayer.TMediaPlayer 17167
MPlayer.TMediaPlayer.Anchors 17197
MPlayer.TMediaPlayer.AutoButtonSet 17181
MPlayer.TMediaPlayer.AutoEnable 17197
MPlayer.TMediaPlayer.AutoOpen 17198
MPlayer.TMediaPlayer.AutoRewind 17199
MPlayer.TMediaPlayer.Back 17182
MPlayer.TMediaPlayer.Capabilities 17199
MPlayer.TMediaPlayer.Click 17182
MPlayer.TMediaPlayer.Close 17183
MPlayer.TMediaPlayer.ColoredButtons 17200
MPlayer.TMediaPlayer.Constraints 17200
MPlayer.TMediaPlayer.Create 17180
MPlayer.TMediaPlayer.Destroy 17180
MPlayer.TMediaPlayer.DeviceID 17201
MPlayer.TMediaPlayer.DeviceType 17201
MPlayer.TMediaPlayer.Display 17202
MPlayer.TMediaPlayer.DisplayRect 17202
MPlayer.TMediaPlayer.DoNotify 17184
MPlayer.TMediaPlayer.Eject 17184
MPlayer.TMediaPlayer.Enabled 17203
MPlayer.TMediaPlayer.EnabledButtons 17203
MPlayer.TMediaPlayer.EndPos 17204
MPlayer.TMediaPlayer.Error 17204
MPlayer.TMediaPlayer.ErrorMessage 17205
MPlayer.TMediaPlayer.FileName 17205
MPlayer.TMediaPlayer.Frames 17205
MPlayer.TMediaPlayer.KeyDown 17186
MPlayer.TMediaPlayer.Length 17206
MPlayer.TMediaPlayer.Loaded 17186
MPlayer.TMediaPlayer.MMNotify 17187
MPlayer.TMediaPlayer.Mode 17206
MPlayer.TMediaPlayer.Next 17187
MPlayer.TMediaPlayer.Notification 17188
MPlayer.TMediaPlayer.Notify 17206
MPlayer.TMediaPlayer.NotifyValue 17207

MPlayer.TMediaPlayer.OnClick 17222
MPlayer.TMediaPlayer.OnContextPopup 17222
MPlayer.TMediaPlayer.OnEnter 17223
MPlayer.TMediaPlayer.OnExit 17226
MPlayer.TMediaPlayer.OnMouseEnter 17227
MPlayer.TMediaPlayer.OnMouseLeave 17227
MPlayer.TMediaPlayer.OnNotify 17228
MPlayer.TMediaPlayer.OnPostClick 17228
MPlayer.TMediaPlayer.Open 17188
MPlayer.TMediaPlayer.Paint 17189
MPlayer.TMediaPlayer.ParentShowHint 17208
MPlayer.TMediaPlayer.Pause 17189
MPlayer.TMediaPlayer.PauseOnly 17190
MPlayer.TMediaPlayer.Play 17190
MPlayer.TMediaPlayer.PopupMenu 17208
MPlayer.TMediaPlayer.Position 17213
MPlayer.TMediaPlayer.PostClick 17191
MPlayer.TMediaPlayer.Previous 17191
MPlayer.TMediaPlayer.Resume 17192
MPlayer.TMediaPlayer.Rewind 17193
MPlayer.TMediaPlayer.Save 17193
MPlayer.TMediaPlayer.Shareable 17213
MPlayer.TMediaPlayer.ShowHint 17214
MPlayer.TMediaPlayer.Start 17215
MPlayer.TMediaPlayer.StartPos 17216
MPlayer.TMediaPlayer.StartRecording 17194
MPlayer.TMediaPlayer.Step 17194
MPlayer.TMediaPlayer.Stop 17195
MPlayer.TMediaPlayer.TabOrder 17216
MPlayer.TMediaPlayer.TabStop 17217
MPlayer.TMediaPlayer.TimeFormat 17217
MPlayer.TMediaPlayer.TrackLength 17218
MPlayer.TMediaPlayer.TrackPosition 17218
MPlayer.TMediaPlayer.Tracks 17218
MPlayer.TMediaPlayer.Updated 17195
MPlayer.TMediaPlayer.Visible 17219
MPlayer.TMediaPlayer.VisibleButtons 17220
MPlayer.TMediaPlayer.Wait 17221
MPlayer.TMPBtnType 17229
MPlayer.TMPBtnType enumeration 17229
MPlayer.TMPButton 17230
MPlayer.TMPButton record 17230
MPlayer.TMPDevCaps 17231
MPlayer.TMPDevCaps enumeration 17231
MPlayer.TMPDevCapsSet 17238
MPlayer.TMPDevCapsSet type 17238
MPlayer.TMPDeviceTypes 17231
MPlayer.TMPDeviceTypes enumeration 17231
MPlayer.TMPGlyph 17233
MPlayer.TMPGlyph enumeration 17233
MPlayer.TMPModes 17233
MPlayer.TMPModes enumeration 17233
MPlayer.TMPNotifyValues 17234
MPlayer.TMPNotifyValues enumeration 17234
MPlayer.TMPTimeFormats 17235
MPlayer.TMPTimeFormats enumeration 17235
mpNotReady enumeration member 17233
mpOpen enumeration member 17233
mpPaused enumeration member 17233
mpPlaying enumeration member 17233
mpRecording enumeration member 17233
mpSeeking enumeration member 17233
mpStopped enumeration member 17233
mpUsesWindow enumeration member 17231
msControlSelect enumeration member 5456
msShiftSelect enumeration member 5456
msSiblingOnly enumeration member 5456
msVisibleOnly enumeration member 5456
mtAny enumeration member 15246
mtConfirmation enumeration member 11558
mtCustom enumeration member 11558
mtError enumeration member 11558
mtFault enumeration member 25564
mtGet enumeration member 15246
mtHead enumeration member 15246
mtHeaderInput enumeration member 25564
mtHeaderOutput enumeration member 25564
mtInformation enumeration member 11558

mtInput enumeration member 25564
mtOutput enumeration member 25564
mtPost enumeration member 15246
mtPut enumeration member 15246
Mtsobj 17239
Mtsobj namespace 17239
Classes 17240
Mtsobj.TMtsAutoObject 17240
Mtsobj.TMtsAutoObject.Activate 17243
Mtsobj.TMtsAutoObject.CanBePooled 17243
Mtsobj.TMtsAutoObject.Deactivate 17243
Mtsobj.TMtsAutoObject.DisableCommit 17243
Mtsobj.TMtsAutoObject.EnableCommit 17244
Mtsobj.TMtsAutoObject.IsCallerInRole 17245
Mtsobj.TMtsAutoObject.IsInTransaction 17245
Mtsobj.TMtsAutoObject.IsSecurityEnabled 17246
Mtsobj.TMtsAutoObject.ObjectContext 17247
Mtsobj.TMtsAutoObject.OnActivate 17246
Mtsobj.TMtsAutoObject.OnDeactivate 17246
Mtsobj.TMtsAutoObject.Pooled 17248
Mtsobj.TMtsAutoObject.SetAbort 17246
Mtsobj.TMtsAutoObject.SetComplete 17247
MtsRdm 17249
MtsRdm namespace 17249
Classes 17250
MtsRdm.TMtsDataModule 17250
MtsRdm.TMtsDataModule.Activate 17255
MtsRdm.TMtsDataModule.AS_ApplyUpdates 17255
MtsRdm.TMtsDataModule.AS_DataRequest 17256
MtsRdm.TMtsDataModule.AS_Execute 17257
MtsRdm.TMtsDataModule.AS_GetParams 17258
MtsRdm.TMtsDataModule.AS_GetProviderNames 17258
MtsRdm.TMtsDataModule.AS_GetRecords 17259
MtsRdm.TMtsDataModule.AS_RowRequest 17260
MtsRdm.TMtsDataModule.AutoComplete 17265
MtsRdm.TMtsDataModule.CallComplete 17261
MtsRdm.TMtsDataModule.CanBePooled 17261
MtsRdm.TMtsDataModule.Create 17254
MtsRdm.TMtsDataModule.Deactivate 17261
MtsRdm.TMtsDataModule.DisableCommit 17261
MtsRdm.TMtsDataModule.EnableCommit 17262
MtsRdm.TMtsDataModule.IsCallerInRole 17263
MtsRdm.TMtsDataModule.IsInTransaction 17263
MtsRdm.TMtsDataModule.IsSecurityEnabled 17264
MtsRdm.TMtsDataModule.ObjectContext 17266
MtsRdm.TMtsDataModule.OnActivate 17267
MtsRdm.TMtsDataModule.OnDeactivate 17267
MtsRdm.TMtsDataModule.Pooled 17266
MtsRdm.TMtsDataModule.SetAbort 17264
MtsRdm.TMtsDataModule.SetComplete 17265
mtWarning enumeration member 11558
Mtx 17269
Mtx namespace 17269
Constants 17297
Functions 17292
Interfaces 17271
Mtx.CLASS_SharedPropertyGroupManager 17297
Mtx.CLASS_SharedPropertyGroupManager constant 17297
Mtx.CLASS_TransactionContext 17297
Mtx.CLASS_TransactionContext constant 17297
Mtx.CLASS_TransactionContextEx 17298
Mtx.CLASS_TransactionContextEx constant 17298
Mtx.CreateSharedPropertyGroup 17292
Mtx.CreateSharedPropertyGroup function 17292
Mtx.CreateSharedPropertyGroupManager 17293
Mtx.CreateSharedPropertyGroupManager function 17293
Mtx.CreateTransactionContext 17293
Mtx.CreateTransactionContext function 17293
Mtx.CreateTransactionContextEx 17294
Mtx.CreateTransactionContextEx function 17294
Mtx.GetObjectContext 17295
Mtx.GetObjectContext function 17295
Mtx.IEnumNames 17271
Mtx.IEnumNames.Clone 17272
Mtx.IEnumNames.Next 17272
Mtx.IEnumNames.Reset 17272
Mtx.IEnumNames.Skip 17272
Mtx.IGetContextProperties 17273

Mtx.IGetContextProperties.Count 17273
Mtx.IGetContextProperties.EnumNames 17274
Mtx.IGetContextProperties.GetProperty 17274
Mtx.IObjectContext 17274
Mtx.IObjectContext.CreateInstance 17275
Mtx.IObjectContext.DisableCommit 17275
Mtx.IObjectContext.EnableCommit 17276
Mtx.IObjectContext.IsCallerInRole 17276
Mtx.IObjectContext.IsInTransaction 17276
Mtx.IObjectContext.IsSecurityEnabled 17276
Mtx.IObjectContext.SetAbort 17276
Mtx.IObjectContext.SetComplete 17277
Mtx.IObjectContextActivity 17277
Mtx.IObjectContextActivity.GetActivityId 17278
Mtx.IObjectControl 17278
Mtx.IObjectControl.Activate 17279
Mtx.IObjectControl.CanBePooled 17279
Mtx.IObjectControl.Deactivate 17279
Mtx.ISecurityProperty 17279
Mtx.ISecurityProperty.GetDirectCallerSID 17280
Mtx.ISecurityProperty.GetDirectCreatorSID 17280
Mtx.ISecurityProperty.GetOriginalCallerSID 17280
Mtx.ISecurityProperty.GetOriginalCreatorSID 17281
Mtx.ISecurityProperty.ReleaseSID 17281
Mtx.ISharedProperty 17281
Mtx.ISharedProperty.Get_Value 17282
Mtx.ISharedProperty.Set_Value 17282
Mtx.ISharedProperty.Value 17283
Mtx.ISharedPropertyGroup 17283
Mtx.ISharedPropertyGroup.CreateProperty 17284
Mtx.ISharedPropertyGroup.CreatePropertyByPosition 17284
Mtx.ISharedPropertyGroup.Get_PropertyByName 17284
Mtx.ISharedPropertyGroup.Get_PropertyByPosition 17285
Mtx.ISharedPropertyGroup.PropertyByName 17285
Mtx.ISharedPropertyGroup.PropertyByPosition 17285
Mtx.ISharedPropertyGroupManager 17285
Mtx.ISharedPropertyGroupManager._NewEnum 17287
Mtx.ISharedPropertyGroupManager.CreatePropertyGroup 17286
Mtx.ISharedPropertyGroupManager.Get__NewEnum 17287
Mtx.ISharedPropertyGroupManager.Get_Group 17287
Mtx.ISharedPropertyGroupManager.Group 17287
Mtx.ITransactionContext 17288
Mtx.ITransactionContext.Abort 17289
Mtx.ITransactionContext.Commit 17289
Mtx.ITransactionContext.CreateInstance 17289
Mtx.ITransactionContextEx 17289
Mtx.ITransactionContextEx.Abort 17290
Mtx.ITransactionContextEx.Commit 17290
Mtx.ITransactionContextEx.CreateInstance 17290
Mtx.LockMethod 17298
Mtx.LockMethod constant 17298
Mtx.LockSetGet 17298
Mtx.LockSetGet constant 17298
Mtx.mtsErrCtxAborted 17299
Mtx.mtsErrCtxAborted constant 17299
Mtx.mtsErrCtxAborting 17299
Mtx.mtsErrCtxAborting constant 17299
Mtx.mtsErrCtxActivityTimeout 17299
Mtx.mtsErrCtxActivityTimeout constant 17299
Mtx.mtsErrCtxNoContext 17300
Mtx.mtsErrCtxNoContext constant 17300
Mtx.mtsErrCtxNoSecurity 17300
Mtx.mtsErrCtxNoSecurity constant 17300
Mtx.mtsErrCtxNotRegistered 17300
Mtx.mtsErrCtxNotRegistered constant 17300
Mtx.mtsErrCtxOldReference 17301
Mtx.mtsErrCtxOldReference constant 17301
Mtx.mtsErrCtxRoleNotFound 17301
Mtx.mtsErrCtxRoleNotFound constant 17301
Mtx.mtsErrCtxTMNotAvailable 17301
Mtx.mtsErrCtxTMNotAvailable constant 17301
Mtx.mtsErrCtxWrongThread 17302
Mtx.mtsErrCtxWrongThread constant 17302
Mtx.Process 17302
Mtx.Process constant 17302
Mtx.SafeRef 17295
Mtx.SafeRef function 17295
Mtx.Standard 17302

Mtx.Standard constant 17302
Mxarrays 17303
Mxarrays namespace 17303
Classes 17305
Constants 17394
Functions 17386
Structs, Records, Enums 17387
Types 17390
Mxarrays.EArrayError 17305
Mxarrays.EDynArrayRangeError 17306
Mxarrays.ELowCapacityError 17306
Mxarrays.EUnsupportedTypeError 17307
Mxarrays.GetMemoryCapacity 17386
Mxarrays.GetMemoryCapacity function 17386
Mxarrays.PIndexElements 17390
Mxarrays.PIndexElements type 17390
Mxarrays.PMatrixElements 17390
Mxarrays.PMatrixElements type 17390
Mxarrays.SetMemoryCapacity 17386
Mxarrays.SetMemoryCapacity function 17386
Mxarrays.TArrayFlags 17387
Mxarrays.TArrayFlags enumeration 17387
Mxarrays.TArrayFlagSet 17391
Mxarrays.TArrayFlagSet type 17391
Mxarrays.TBaseArray 17307
Mxarrays.TBaseArray.AutoSize 17318
Mxarrays.TBaseArray.BlockCopy 17311
Mxarrays.TBaseArray.Capacity 17319
Mxarrays.TBaseArray.Clear 17311
Mxarrays.TBaseArray.ClearFlag 17311
Mxarrays.TBaseArray.CompareProc 17319
Mxarrays.TBaseArray.Count 17320
Mxarrays.TBaseArray.Create 17309
Mxarrays.TBaseArray.Delete 17312
Mxarrays.TBaseArray.Destroy 17310
Mxarrays.TBaseArray.Duplicates 17320
Mxarrays.TBaseArray.Exchange 17312
Mxarrays.TBaseArray.FindItem 17312
Mxarrays.TBaseArray.GetAutoSize 17313
Mxarrays.TBaseArray.GetItem 17313
Mxarrays.TBaseArray.Grow 17313
Mxarrays.TBaseArray.HasFlag 17314
Mxarrays.TBaseArray.IndexOf 17314
Mxarrays.TBaseArray.Insert 17314
Mxarrays.TBaseArray.InsertAt 17315
Mxarrays.TBaseArray.InternalHandleException 17315
Mxarrays.TBaseArray.InvalidateItems 17315
Mxarrays.TBaseArray.ItemSize 17321
Mxarrays.TBaseArray.Limit 17321
Mxarrays.TBaseArray.List 17321
Mxarrays.TBaseArray.PutItem 17315
Mxarrays.TBaseArray.RemoveItem 17316
Mxarrays.TBaseArray.RemoveRange 17316
Mxarrays.TBaseArray.SetAutoSize 17316
Mxarrays.TBaseArray.SetCapacity 17317
Mxarrays.TBaseArray.SetFlag 17317
Mxarrays.TBaseArray.Sort 17317
Mxarrays.TBaseArray.SortOrder 17322
Mxarrays.TBaseArray.ValidateBounds 17317
Mxarrays.TBaseArray.ValidIndex 17318
Mxarrays.TCompareProc 17391
Mxarrays.TCompareProc type 17391
Mxarrays.TCurrencyArray 17322
Mxarrays.TCurrencyArray.Add 17325
Mxarrays.TCurrencyArray.Assign 17325
Mxarrays.TCurrencyArray.Create 17324
Mxarrays.TCurrencyArray.Find 17326
Mxarrays.TCurrencyArray.GetItem 17326
Mxarrays.TCurrencyArray.IndexOf 17327
Mxarrays.TCurrencyArray.Items 17327
Mxarrays.TCurrencyArray.PutItem 17327
Mxarrays.TCustomArray 17328
Mxarrays.TCustomArray.Add 17330
Mxarrays.TCustomArray.Assign 17330
Mxarrays.TCustomArray.BlankBoolVal 17334
Mxarrays.TCustomArray.BlankCount 17335
Mxarrays.TCustomArray.BlankDateVal 17335
Mxarrays.TCustomArray.BlankStringVal 17335

Mxarrays.TCustomArray.CompareProc 17335
Mxarrays.TCustomArray.ConvertVar 17330
Mxarrays.TCustomArray.Create 17329
Mxarrays.TCustomArray.DataType 17335
Mxarrays.TCustomArray.Destroy 17329
Mxarrays.TCustomArray.Duplicates 17336
Mxarrays.TCustomArray.GetCompProc 17330
Mxarrays.TCustomArray.GetCurrency 17331
Mxarrays.TCustomArray.GetDouble 17331
Mxarrays.TCustomArray.GetDups 17331
Mxarrays.TCustomArray.GetInteger 17331
Mxarrays.TCustomArray.GetItem 17332
Mxarrays.TCustomArray.GetMemberCount 17332
Mxarrays.TCustomArray.GetSort 17332
Mxarrays.TCustomArray.IndexOf 17332
Mxarrays.TCustomArray.IsBlank 17332
Mxarrays.TCustomArray.Items 17336
Mxarrays.TCustomArray.List 17336
Mxarrays.TCustomArray.MemberCount 17336
Mxarrays.TCustomArray.MemoryUsage 17333
Mxarrays.TCustomArray.SetCompProc 17333
Mxarrays.TCustomArray.SetDups 17333
Mxarrays.TCustomArray.SetItem 17333
Mxarrays.TCustomArray.SetSize 17334
Mxarrays.TCustomArray.SetSort 17334
Mxarrays.TCustomArray.Sorted 17337
Mxarrays.TDoubleArray 17337
Mxarrays.TDoubleArray.Add 17340
Mxarrays.TDoubleArray.Assign 17340
Mxarrays.TDoubleArray.Create 17339
Mxarrays.TDoubleArray.Find 17341
Mxarrays.TDoubleArray.GetItem 17341
Mxarrays.TDoubleArray.IndexOf 17341
Mxarrays.TDoubleArray.Items 17342
Mxarrays.TDoubleArray.PutItem 17341
Mxarrays.TDuplicates 17387
Mxarrays.TDuplicates enumeration 17387
Mxarrays.TDynArrayNDX 17391
Mxarrays.TDynArrayNDX type 17391
Mxarrays.TIndexArray 17342
Mxarrays.TIndexArray.Add 17344
Mxarrays.TIndexArray.Autosize 17345
Mxarrays.TIndexArray.Capacity 17345
Mxarrays.TIndexArray.Count 17345
Mxarrays.TIndexArray.Create 17343
Mxarrays.TIndexArray.Destroy 17343
Mxarrays.TIndexArray.Expand 17344
Mxarrays.TIndexArray.idxElements 17344
Mxarrays.TIndexArray.Items 17346
Mxarrays.TIndexArray.MemoryUsage 17346
Mxarrays.TIndexArray.SetSize 17344
Mxarrays.TIndexElements 17392
Mxarrays.TIndexElements type 17392
Mxarrays.TIndexNDX 17392
Mxarrays.TIndexNDX type 17392
Mxarrays.TIntArray 17346
Mxarrays.TIntArray.Add 17349
Mxarrays.TIntArray.Assign 17349
Mxarrays.TIntArray.Create 17348
Mxarrays.TIntArray.Find 17350
Mxarrays.TIntArray.GetItem 17350
Mxarrays.TIntArray.Items 17351
Mxarrays.TIntArray.PutItem 17350
Mxarrays.TMatrixElements 17392
Mxarrays.TMatrixElements type 17392
Mxarrays.TMatrixNDX 17393
Mxarrays.TMatrixNDX type 17393
Mxarrays.TPointerArray 17351
Mxarrays.TPointerArray.AsPtr 17355
Mxarrays.TPointerArray.CloneItem 17354
Mxarrays.TPointerArray.CopyFrom 17354
Mxarrays.TPointerArray.CopyTo 17354
Mxarrays.TPointerArray.Create 17353
Mxarrays.TPointerArray.Data 17356
Mxarrays.TPointerArray.FreeItem 17354
Mxarrays.TPointerArray.GetData 17355
Mxarrays.TPointerArray.InvalidateItems 17355
Mxarrays.TPointerArray.PutData 17355

Mxarrays.TSingleArray 17356
Mxarrays.TSingleArray.Add 17359
Mxarrays.TSingleArray.Assign 17359
Mxarrays.TSingleArray.Create 17358
Mxarrays.TSingleArray.Find 17360
Mxarrays.TSingleArray.GetItem 17360
Mxarrays.TSingleArray.IndexOf 17360
Mxarrays.TSingleArray.Items 17361
Mxarrays.TSingleArray.PutItem 17360
Mxarrays.TSmallIntArray 17361
Mxarrays.TSmallIntArray.Add 17364
Mxarrays.TSmallIntArray.Assign 17365
Mxarrays.TSmallIntArray.Create 17363
Mxarrays.TSmallIntArray.GetItem 17365
Mxarrays.TSmallIntArray.Items 17366
Mxarrays.TSmallIntArray.PutItem 17366
Mxarrays.TSortOrder 17388
Mxarrays.TSortOrder enumeration 17388
Mxarrays.TStringArray 17367
Mxarrays.TStringArray.Add 17370
Mxarrays.TStringArray.Assign 17370
Mxarrays.TStringArray.Create 17369
Mxarrays.TStringArray.Exchange 17371
Mxarrays.TStringArray.Find 17371
Mxarrays.TStringArray.GetObject 17371
Mxarrays.TStringArray.GetString 17372
Mxarrays.TStringArray.Grow 17372
Mxarrays.TStringArray.IndexOf 17372
Mxarrays.TStringArray.Insert 17372
Mxarrays.TStringArray.InvalidateItems 17373
Mxarrays.TStringArray.PutObject 17373
Mxarrays.TStringArray.PutString 17373
Mxarrays.TStringArray.Sort 17373
Mxarrays.TStringArray.Strings 17374
Mxarrays.TStringItem 17388
Mxarrays.TStringItem record 17388
Mxarrays.TThreadCustomArray 17374
Mxarrays.TThreadCustomArray.Add 17376
Mxarrays.TThreadCustomArray.Create 17375
Mxarrays.TThreadCustomArray.Destroy 17375
Mxarrays.TThreadCustomArray.GetItem 17376
Mxarrays.TThreadCustomArray.LockArray 17376
Mxarrays.TThreadCustomArray.MemoryUsage 17376
Mxarrays.TThreadCustomArray.UnlockArray 17376
Mxarrays.TTwoDimArray 17377
Mxarrays.TTwoDimArray.columns 17379
Mxarrays.TTwoDimArray.Create 17377
Mxarrays.TTwoDimArray.Destroy 17378
Mxarrays.TTwoDimArray.Element 17379
Mxarrays.TTwoDimArray.mtxElements 17378
Mxarrays.TTwoDimArray.rows 17379
Mxarrays.TTwoDimArray.SetSize 17378
Mxarrays.TWordArray 17380
Mxarrays.TWordArray.Add 17382
Mxarrays.TWordArray.Assign 17383
Mxarrays.TWordArray.Create 17382
Mxarrays.TWordArray.Find 17383
Mxarrays.TWordArray.GetItem 17384
Mxarrays.TWordArray.IndexOf 17384
Mxarrays.TWordArray.Items 17385
Mxarrays.TWordArray.PutItem 17384
Mxarrays.vMaxCol 17394
Mxarrays.vMaxCol constant 17394
Mxarrays.vMaxRow 17394
Mxarrays.vMaxRow constant 17394
MXCOMMON 17395
MXCOMMON namespace 17395
    Classes 17397
    Functions 17427
    Structs, Records, Enums 17432
    Types 17436
MXCOMMON.AddToQuerySelect 17427
MXCOMMON.AddToQuerySelect function 17427
MXCOMMON.BuildDataSetMap 17427
MXCOMMON.BuildDataSetMap function 17427
MXCOMMON.CheckIfEmptyQuery 17428
MXCOMMON.CheckIfEmptyQuery function 17428
MXCOMMON.EDimensionMapError 17397

MXCOMMON.FieldTypeToVarType 17428
MXCOMMON.FieldTypeToVarType function 17428
MXCOMMON.FormatVariant 17428
MXCOMMON.FormatVariant function 17428
MXCOMMON.GetAggName 17429
MXCOMMON.GetAggName function 17429
MXCOMMON.HookToDataBase 17429
MXCOMMON.HookToDataBase function 17429
MXCOMMON.IsAggValid 17429
MXCOMMON.IsAggValid function 17429
MXCOMMON.IsBDEAvailable 17430
MXCOMMON.IsBDEAvailable function 17430
MXCOMMON.NextArg 17430
MXCOMMON.NextArg function 17430
MXCOMMON.PTickInfoRec 17436
MXCOMMON.PTickInfoRec type 17436
MXCOMMON.TActiveFlags 17432
MXCOMMON.TActiveFlags enumeration 17432
MXCOMMON.TBinData 17397
MXCOMMON.TBinData.AddBin 17400
MXCOMMON.TBinData.AddBinValue 17400
MXCOMMON.TBinData.AddBinValues 17400
MXCOMMON.TBinData.AddIBinValue 17401
MXCOMMON.TBinData.Assign 17401
MXCOMMON.TBinData.BinCount 17404
MXCOMMON.TBinData.BinName 17404
MXCOMMON.TBinData.BinValueCount 17402
MXCOMMON.TBinData.Clear 17402
MXCOMMON.TBinData.create 17399
MXCOMMON.TBinData.destroy 17399
MXCOMMON.TBinData.GetAllBinValueCount 17402
MXCOMMON.TBinData.GetBinCount 17402
MXCOMMON.TBinData.GetBinName 17402
MXCOMMON.TBinData.GetBinNameDataType 17403
MXCOMMON.TBinData.GetBinValue 17403
MXCOMMON.TBinData.GetBinValues 17403
MXCOMMON.TBinData.GetIBinValue 17403
MXCOMMON.TBinData.GetIBinValueCount 17404
MXCOMMON.TBinData.GetMaxBinNameSize 17404
MXCOMMON.TBinData.NameList 17405
MXCOMMON.TBinData.OtherBinName 17405
MXCOMMON.TBinData.ValueList 17405
MXCOMMON.TBinType 17432
MXCOMMON.TBinType enumeration 17432
MXCOMMON.TBinTypes 17436
MXCOMMON.TBinTypes type 17436
MXCOMMON.TCDNotifyType 17433
MXCOMMON.TCDNotifyType enumeration 17433
MXCOMMON.TDimensionItem 17405
MXCOMMON.TDimensionItem.Active 17411
MXCOMMON.TDimensionItem.ActiveFlag 17411
MXCOMMON.TDimensionItem.Assign 17408
MXCOMMON.TDimensionItem.BaseName 17411
MXCOMMON.TDimensionItem.Create 17408
MXCOMMON.TDimensionItem.DerivedFrom 17412
MXCOMMON.TDimensionItem.DimensionType 17412
MXCOMMON.TDimensionItem.Fieldname 17413
MXCOMMON.TDimensionItem.FieldType 17413
MXCOMMON.TDimensionItem.Format 17414
MXCOMMON.TDimensionItem.InitializeRange 17409
MXCOMMON.TDimensionItem.IsDimension 17409
MXCOMMON.TDimensionItem.IsSummary 17410
MXCOMMON.TDimensionItem.Name 17414
MXCOMMON.TDimensionItem.NotifyCollection 17410
MXCOMMON.TDimensionItem.Owner 17414
MXCOMMON.TDimensionItemClass 17436
MXCOMMON.TDimensionItemClass type 17436
MXCOMMON.TDimensionItems 17415
MXCOMMON.TDimensionItems.ActiveDimensionCount 17421
MXCOMMON.TDimensionItems.ActiveSummaryCount 17421
MXCOMMON.TDimensionItems.Add 17418
MXCOMMON.TDimensionItems.AddDerivedField 17418
MXCOMMON.TDimensionItems.AlreadyExists 17419
MXCOMMON.TDimensionItems.Assign 17419
MXCOMMON.TDimensionItems.AverageFieldCheck 17419
MXCOMMON.TDimensionItems.Create 17417
MXCOMMON.TDimensionItems.DimensionCount 17422
MXCOMMON.TDimensionItems.GetOwner 17420

MXCOMMON.TDimensionItems.Items 17422
MXCOMMON.TDimensionItems.NotifyOwner 17420
MXCOMMON.TDimensionItems.SummaryCount 17423
MXCOMMON.TDimFlags 17433
MXCOMMON.TDimFlags enumeration 17433
MXCOMMON.TQueryError 17434
MXCOMMON.TQueryError enumeration 17434
MXCOMMON.TTickInfoRec 17434
MXCOMMON.TTickInfoRec record 17434
MXCOMMON.TTicks 17424
MXCOMMON.TTicks.Clear 17425
MXCOMMON.TTicks.Create 17424
MXCOMMON.TTicks.Destroy 17425
MXCOMMON.TTicks.Dump 17425
MXCOMMON.TTicks.NumberOfValues 17426
MXCOMMON.TTicks.Ticks 17425
MXCOMMON.TTicks.TicksIntArray 17426
MXCOMMON.TTicks.TicksSmallIntArray 17426
MXCOMMON.UpdateDesigner 17430
MXCOMMON.UpdateDesigner function 17430
MXCOMMON.VerifyRTQuery 17431
MXCOMMON.VerifyRTQuery function 17431
MXDB 17439
MXDB namespace 17439
Classes 17441
Structs, Records, Enums 17510
Types 17514
MXDB.EArrayError 17441
MXDB.EDimIndexError 17441
MXDB.PArrayDimInfo 17514
MXDB.PArrayDimInfo type 17514
MXDB.PArrayInt 17514
MXDB.PArrayInt type 17514
MXDB.PDimInfo 17515
MXDB.PDimInfo type 17515
MXDB.TArrayDimInfo 17515
MXDB.TArrayDimInfo type 17515
MXDB.TArrayInt 17515
MXDB.TArrayInt type 17515
MXDB.TBigStr 17516
MXDB.TBigStr type 17516
MXDB.TDecisionControlType 17510
MXDB.TDecisionControlType enumeration 17510
MXDB.TDecisionCube 17442
MXDB.TDecisionCube.Active 17451
MXDB.TDecisionCube.AfterClose 17456
MXDB.TDecisionCube.AfterOpen 17456
MXDB.TDecisionCube.BeforeClose 17456
MXDB.TDecisionCube.BeforeOpen 17457
MXDB.TDecisionCube.CanDimBeClosed 17448
MXDB.TDecisionCube.CanSumBeClosed 17448
MXDB.TDecisionCube.Create 17447
MXDB.TDecisionCube.DataSet 17451
MXDB.TDecisionCube.DesignState 17452
MXDB.TDecisionCube.Destroy 17447
MXDB.TDecisionCube.DimensionMap 17453
MXDB.TDecisionCube.GetDetailSQL 17448
MXDB.TDecisionCube.GetSQL 17449
MXDB.TDecisionCube.MaxCells 17453
MXDB.TDecisionCube.MaxDimensions 17454
MXDB.TDecisionCube.MaxSummaries 17454
MXDB.TDecisionCube.OnLowCapacity 17457
MXDB.TDecisionCube.OnRefresh 17458
MXDB.TDecisionCube.ShowCubeDialog 17450
MXDB.TDecisionCube.ShowProgressDialog 17455
MXDB.TDecisionCube.StateChanged 17450
MXDB.TDecisionDataEvent 17510
MXDB.TDecisionDataEvent enumeration 17510
MXDB.TDecisionDataLink 17458
MXDB.TDecisionDataLink.Create 17459
MXDB.TDecisionDataLink.DecisionDataEvent 17461
MXDB.TDecisionDataLink.DecisionSource 17462
MXDB.TDecisionDataLink.Destroy 17460
MXDB.TDecisionDataLink.FBlocked 17460
MXDB.TDecisionDataLink.FDecisionSource 17461
MXDB.TDecisionDataLink.SetDecisionSource 17461
MXDB.TDecisionSource 17462
MXDB.TDecisionSource.CloseDimIndexRight 17470

MXDB.TDecisionSource.ControlType 17485
MXDB.TDecisionSource.Create 17467
MXDB.TDecisionSource.CurrentSum 17486
MXDB.TDecisionSource.DecisionCube 17487
MXDB.TDecisionSource.DefineProperties 17470
MXDB.TDecisionSource.Destroy 17467
MXDB.TDecisionSource.DrillDimIndex 17471
MXDB.TDecisionSource.FOnAfterPivot 17468
MXDB.TDecisionSource.FOnBeforePivot 17468
MXDB.TDecisionSource.FOnLayoutChange 17468
MXDB.TDecisionSource.FOnNewDimensions 17468
MXDB.TDecisionSource.FOnStateChange 17468
MXDB.TDecisionSource.FonSummaryChange 17469
MXDB.TDecisionSource.Get2DDataAsVariant 17472
MXDB.TDecisionSource.GetActiveDim 17472
MXDB.TDecisionSource.GetDataAsString 17473
MXDB.TDecisionSource.GetDataAsVariant 17473
MXDB.TDecisionSource.GetDimensionMemberCount 17474
MXDB.TDecisionSource.GetDimensionName 17474
MXDB.TDecisionSource.GetGroup 17475
MXDB.TDecisionSource.GetGroupCount 17475
MXDB.TDecisionSource.GetGroupExtent 17476
MXDB.TDecisionSource.GetIndex 17477
MXDB.TDecisionSource.GetMemberAsString 17477
MXDB.TDecisionSource.GetMemberAsVariant 17478
MXDB.TDecisionSource.GetRowState 17478
MXDB.TDecisionSource.GetState 17479
MXDB.TDecisionSource.GetSummaryName 17479
MXDB.TDecisionSource.GetValue 17480
MXDB.TDecisionSource.GetValueArray 17480
MXDB.TDecisionSource.GetValueIndex 17481
MXDB.TDecisionSource.MoveDimIndexes 17481
MXDB.TDecisionSource.Name 17487
MXDB.TDecisionSource.nColDims 17489
MXDB.TDecisionSource.nDataCols 17489
MXDB.TDecisionSource.nDataRows 17490
MXDB.TDecisionSource.nDims 17490
MXDB.TDecisionSource.nOpenColDims 17491
MXDB.TDecisionSource.nOpenRowDims 17491
MXDB.TDecisionSource.nRowDims 17491
MXDB.TDecisionSource.nSums 17492
MXDB.TDecisionSource.OnAfterPivot 17494
MXDB.TDecisionSource.OnBeforePivot 17494
MXDB.TDecisionSource.OnLayoutChange 17495
MXDB.TDecisionSource.OnNewDimensions 17495
MXDB.TDecisionSource.OnStateChange 17496
MXDB.TDecisionSource.OnSummaryChange 17496
MXDB.TDecisionSource.OpenDimIndexLeft 17482
MXDB.TDecisionSource.OpenDimIndexRight 17482
MXDB.TDecisionSource.Ready 17492
MXDB.TDecisionSource.SetCurrentSummary 17483
MXDB.TDecisionSource.SetDimGroup 17483
MXDB.TDecisionSource.SetDimState 17484
MXDB.TDecisionSource.SparseCols 17493
MXDB.TDecisionSource.SparseRows 17493
MXDB.TDecisionSource.SwapDimIndexes 17484
MXDB.TDecisionSource.ToggleDimIndex 17484
MXDB.TDimGroup 17511
MXDB.TDimGroup enumeration 17511
MXDB.TDimInfo 17511
MXDB.TDimInfo record 17511
MXDB.TDimInfoArray 17497
MXDB.TDimInfoArray.Assign 17498
MXDB.TDimInfoArray.Create 17498
MXDB.TDimInfoArray.Destroy 17498
MXDB.TDimInfoArray.Find 17499
MXDB.TDimInfoArray.GetGroupArray 17499
MXDB.TDimInfoArray.GetGroupIndex 17499
MXDB.TDimInfoArray.GetGroupItem 17499
MXDB.TDimInfoArray.GetGroupSize 17499
MXDB.TDimInfoArray.GetItem 17500
MXDB.TDimInfoArray.IsEqual 17500
MXDB.TDimInfoArray.Items 17500
MXDB.TDimInfoArray.Limit 17500
MXDB.TDimRange 17512
MXDB.TDimRange record 17512
MXDB.TDimState 17512
MXDB.TDimState enumeration 17512

MXDB.TIArray 17501
MXDB.TIArray.Assign 17502
MXDB.TIArray.AutoSize 17504
MXDB.TIArray.Create 17502
MXDB.TIArray.Destroy 17502
MXDB.TIArray.GetItem 17503
MXDB.TIArray.InsertAt 17503
MXDB.TIArray.Items 17504
MXDB.TIArray.Limit 17504
MXDB.TIArray.RemoveItem 17503
MXDB.TIArray.SetItem 17503
MXDB.TPivotState 17504
MXDB.TPivotState.Assign 17508
MXDB.TPivotState.Create 17505
MXDB.TPivotState.Destroy 17505
MXDB.TPivotState.DimInfo 17506
MXDB.TPivotState.FColSparse 17506
MXDB.TPivotState.FColSubs 17506
MXDB.TPivotState.FCurrentSum 17507
MXDB.TPivotState.FDims 17507
MXDB.TPivotState.FRowSparse 17507
MXDB.TPivotState.FRowSubs 17507
MXDB.TPivotState.FSums 17507
MXDB.TPivotState.IsEqual 17508
MXDB.TQADecisionSource 17508
MXDB.TRowState 17516
MXDB.TRowState type 17516
MXDB.TRowStates 17513
MXDB.TRowStates enumeration 17513
MXGRAPH 17517
MXGRAPH namespace 17517
Classes 17518
Structs, Records, Enums 17549
Types 17550
MXGRAPH.TCustomDecisionGraph 17518
MXGRAPH.TCustomDecisionGraph.Create 17519
MXGRAPH.TCustomDecisionGraph.DecisionSource 17521
MXGRAPH.TCustomDecisionGraph.Destroy 17519
MXGRAPH.TCustomDecisionGraph.Notification 17520
MXGRAPH.TCustomDecisionGraph.Updated 17520
MXGRAPH.TDecisionGraph 17521
MXGRAPH.TDecisionGraph.Align 17526
MXGRAPH.TDecisionGraph.AllowPanning 17526
MXGRAPH.TDecisionGraph.AllowZoom 17526
MXGRAPH.TDecisionGraph.Anchors 17526
MXGRAPH.TDecisionGraph.AnimatedZoom 17527
MXGRAPH.TDecisionGraph.AnimatedZoomSteps 17527
MXGRAPH.TDecisionGraph.AutoSize 17527
MXGRAPH.TDecisionGraph.AxisVisible 17527
MXGRAPH.TDecisionGraph.BackColor 17528
MXGRAPH.TDecisionGraph.BackImage 17528
MXGRAPH.TDecisionGraph.BackImageInside 17528
MXGRAPH.TDecisionGraph.BackImageMode 17528
MXGRAPH.TDecisionGraph.BevelInner 17528
MXGRAPH.TDecisionGraph.BevelOuter 17529
MXGRAPH.TDecisionGraph.BevelWidth 17529
MXGRAPH.TDecisionGraph.BorderStyle 17529
MXGRAPH.TDecisionGraph.BorderWidth 17529
MXGRAPH.TDecisionGraph.BottomAxis 17530
MXGRAPH.TDecisionGraph.BottomWall 17530
MXGRAPH.TDecisionGraph.Chart3DPercent 17530
MXGRAPH.TDecisionGraph.ClipPoints 17530
MXGRAPH.TDecisionGraph.Color 17530
MXGRAPH.TDecisionGraph.Constraints 17531
MXGRAPH.TDecisionGraph.DecisionSource 17531
MXGRAPH.TDecisionGraph.DragCursor 17531
MXGRAPH.TDecisionGraph.DragKind 17531
MXGRAPH.TDecisionGraph.DragMode 17532
MXGRAPH.TDecisionGraph.Enabled 17532
MXGRAPH.TDecisionGraph.Foot 17532
MXGRAPH.TDecisionGraph.Frame 17532
MXGRAPH.TDecisionGraph.Gradient 17533
MXGRAPH.TDecisionGraph.LeftAxis 17533
MXGRAPH.TDecisionGraph.LeftWall 17533
MXGRAPH.TDecisionGraph.Legend 17533
MXGRAPH.TDecisionGraph.MarginBottom 17533
MXGRAPH.TDecisionGraph.MarginLeft 17534
MXGRAPH.TDecisionGraph.MarginRight 17534

MXGRAPH.TDecisionGraph.MarginTop 17534
MXGRAPH.TDecisionGraph.MaxPointsPerPage 17534
MXGRAPH.TDecisionGraph.Monochrome 17535
MXGRAPH.TDecisionGraph.OnAfterDraw 17535
MXGRAPH.TDecisionGraph.OnAllowScroll 17535
MXGRAPH.TDecisionGraph.OnCanResize 17535
MXGRAPH.TDecisionGraph.OnClick 17535
MXGRAPH.TDecisionGraph.OnClickAxis 17536
MXGRAPH.TDecisionGraph.OnClickBackground 17536
MXGRAPH.TDecisionGraph.OnClickLegend 17536
MXGRAPH.TDecisionGraph.OnClickSeries 17536
MXGRAPH.TDecisionGraph.OnConstrainedResize 17537
MXGRAPH.TDecisionGraph.OnDblClick 17537
MXGRAPH.TDecisionGraph.OnDockDrop 17537
MXGRAPH.TDecisionGraph.OnDockOver 17537
MXGRAPH.TDecisionGraph.OnDragDrop 17537
MXGRAPH.TDecisionGraph.OnDragOver 17538
MXGRAPH.TDecisionGraph.OnEndDock 17538
MXGRAPH.TDecisionGraph.OnEndDrag 17538
MXGRAPH.TDecisionGraph.OnEnter 17538
MXGRAPH.TDecisionGraph.OnExit 17539
MXGRAPH.TDecisionGraph.OnGetAxisLabel 17539
MXGRAPH.TDecisionGraph.OnGetLegendPos 17539
MXGRAPH.TDecisionGraph.OnGetLegendRect 17539
MXGRAPH.TDecisionGraph.OnGetLegendText 17539
MXGRAPH.TDecisionGraph.OnGetNextAxisLabel 17540
MXGRAPH.TDecisionGraph.OnGetSiteInfo 17540
MXGRAPH.TDecisionGraph.OnMouseDown 17540
MXGRAPH.TDecisionGraph.OnMouseMove 17540
MXGRAPH.TDecisionGraph.OnMouseUp 17541
MXGRAPH.TDecisionGraph.OnPageChange 17541
MXGRAPH.TDecisionGraph.OnResize 17541
MXGRAPH.TDecisionGraph.OnScroll 17541
MXGRAPH.TDecisionGraph.OnStartDock 17541
MXGRAPH.TDecisionGraph.OnStartDrag 17542
MXGRAPH.TDecisionGraph.OnUnDock 17542
MXGRAPH.TDecisionGraph.OnUndoZoom 17542
MXGRAPH.TDecisionGraph.OnZoom 17542
MXGRAPH.TDecisionGraph.Page 17543
MXGRAPH.TDecisionGraph.ParentColor 17543
MXGRAPH.TDecisionGraph.ParentShowHint 17543
MXGRAPH.TDecisionGraph.PopupMenu 17543
MXGRAPH.TDecisionGraph.RightAxis 17543
MXGRAPH.TDecisionGraph.ScaleLastPage 17544
MXGRAPH.TDecisionGraph.SeriesList 17544
MXGRAPH.TDecisionGraph.ShowHint 17544
MXGRAPH.TDecisionGraph.TabOrder 17544
MXGRAPH.TDecisionGraph.TabStop 17545
MXGRAPH.TDecisionGraph.Title 17545
MXGRAPH.TDecisionGraph.TopAxis 17545
MXGRAPH.TDecisionGraph.View3D 17545
MXGRAPH.TDecisionGraph.View3DWalls 17545
MXGRAPH.TDecisionGraph.Visible 17546
MXGRAPH.TDecisionGraphDataLink 17546
MXGRAPH.TDecisionGraphDataLink.Create 17547
MXGRAPH.TDecisionGraphDataLink.DecisionDataEvent 17548
MXGRAPH.TDecisionGraphDataLink.Destroy 17548
MXGRAPH.TDimEvent 17550
MXGRAPH.TDimEvent type 17550
MXGRAPH.TSeriesType 17549
MXGRAPH.TSeriesType enumeration 17549
MXGRID 17551
MXGRID namespace 17551
    Classes 17553
    Structs, Records, Enums 17672
    Types 17676
MXGRID.TButtonPlace 17672
MXGRID.TButtonPlace enumeration 17672
MXGRID.TCustomDecisionGrid 17553
MXGRID.TCustomDecisionGrid.CaptionColor 17574
MXGRID.TCustomDecisionGrid.CaptionFont 17575
MXGRID.TCustomDecisionGrid.CellDrawState 17568
MXGRID.TCustomDecisionGrid.CellRect 17569
MXGRID.TCustomDecisionGrid.Cells 17575
MXGRID.TCustomDecisionGrid.CellValueArray 17569
MXGRID.TCustomDecisionGrid.ColCount 17576
MXGRID.TCustomDecisionGrid.ColWidthsChanged 17569
MXGRID.TCustomDecisionGrid.Create 17566

MXGRID.TCustomDecisionGrid.DataColor 17576
MXGRID.TCustomDecisionGrid.DataFont 17577
MXGRID.TCustomDecisionGrid.DataSumColor 17577
MXGRID.TCustomDecisionGrid.DecisionSource 17578
MXGRID.TCustomDecisionGrid.DefaultColWidth 17578
MXGRID.TCustomDecisionGrid.DefaultRowHeight 17578
MXGRID.TCustomDecisionGrid.Destroy 17566
MXGRID.TCustomDecisionGrid.Dimensions 17579
MXGRID.TCustomDecisionGrid.DrawCell 17570
MXGRID.TCustomDecisionGrid.FGridStateX 17567
MXGRID.TCustomDecisionGrid.FixedCols 17579
MXGRID.TCustomDecisionGrid.FixedRows 17580
MXGRID.TCustomDecisionGrid.GetCells 17570
MXGRID.TCustomDecisionGrid.GridLineColor 17580
MXGRID.TCustomDecisionGrid.GridLineWidth 17580
MXGRID.TCustomDecisionGrid.LabelColor 17581
MXGRID.TCustomDecisionGrid.LabelFont 17581
MXGRID.TCustomDecisionGrid.LabelSumColor 17581
MXGRID.TCustomDecisionGrid.MouseDown 17570
MXGRID.TCustomDecisionGrid.MouseMove 17571
MXGRID.TCustomDecisionGrid.MouseUp 17571
MXGRID.TCustomDecisionGrid.Notification 17572
MXGRID.TCustomDecisionGrid.OnDecisionDrawCell 17584
MXGRID.TCustomDecisionGrid.OnDecisionExamineCell 17585
MXGRID.TCustomDecisionGrid.OnTopLeftChanged 17586
MXGRID.TCustomDecisionGrid.Options 17582
MXGRID.TCustomDecisionGrid.Paint 17572
MXGRID.TCustomDecisionGrid.RowCount 17582
MXGRID.TCustomDecisionGrid.RowHeightsChanged 17573
MXGRID.TCustomDecisionGrid.SelectCell 17573
MXGRID.TCustomDecisionGrid.ShowCubeEditor 17583
MXGRID.TCustomDecisionGrid.TopLeftChanged 17573
MXGRID.TCustomDecisionGrid.Totals 17583
MXGRID.TDDNotifyType 17672
MXGRID.TDDNotifyType enumeration 17672
MXGRID.TDecisionCellType 17673
MXGRID.TDecisionCellType enumeration 17673
MXGRID.TDecisionDrawCellEvent 17676
MXGRID.TDecisionDrawCellEvent type 17676
MXGRID.TDecisionDrawState 17676
MXGRID.TDecisionDrawState type 17676
MXGRID.TDecisionDrawStates 17673
MXGRID.TDecisionDrawStates enumeration 17673
MXGRID.TDecisionExamineCellEvent 17677
MXGRID.TDecisionExamineCellEvent type 17677
MXGRID.TDecisionGrid 17586
MXGRID.TDecisionGrid.Align 17601
MXGRID.TDecisionGrid.Anchors 17605
MXGRID.TDecisionGrid.BorderStyle 17605
MXGRID.TDecisionGrid.CaptionColor 17605
MXGRID.TDecisionGrid.CaptionFont 17606
MXGRID.TDecisionGrid.ColCount 17606
MXGRID.TDecisionGrid.Color 17607
MXGRID.TDecisionGrid.Constraints 17608
MXGRID.TDecisionGrid.Ctl3D 17608
MXGRID.TDecisionGrid.DataColor 17609
MXGRID.TDecisionGrid.DataFont 17609
MXGRID.TDecisionGrid.DataSumColor 17610
MXGRID.TDecisionGrid.DecisionSource 17610
MXGRID.TDecisionGrid.DefaultColWidth 17610
MXGRID.TDecisionGrid.DefaultDrawing 17611
MXGRID.TDecisionGrid.DefaultRowHeight 17611
MXGRID.TDecisionGrid.Dimensions 17612
MXGRID.TDecisionGrid.DragCursor 17612
MXGRID.TDecisionGrid.DragMode 17613
MXGRID.TDecisionGrid.Enabled 17614
MXGRID.TDecisionGrid.FixedCols 17615
MXGRID.TDecisionGrid.FixedRows 17615
MXGRID.TDecisionGrid.GridLineColor 17615
MXGRID.TDecisionGrid.GridLineWidth 17616
MXGRID.TDecisionGrid.LabelColor 17616
MXGRID.TDecisionGrid.LabelFont 17616
MXGRID.TDecisionGrid.LabelSumColor 17617
MXGRID.TDecisionGrid.OnClick 17630
MXGRID.TDecisionGrid.OnDblClick 17632
MXGRID.TDecisionGrid.OnDecisionDrawCell 17632
MXGRID.TDecisionGrid.OnDecisionExamineCell 17634
MXGRID.TDecisionGrid.OnDragDrop 17634

MXGRID.TDecisionGrid.OnDragOver 17637
MXGRID.TDecisionGrid.OnEndDrag 17639
MXGRID.TDecisionGrid.OnEnter 17640
MXGRID.TDecisionGrid.OnExit 17642
MXGRID.TDecisionGrid.OnKeyDown 17644
MXGRID.TDecisionGrid.OnKeyPress 17647
MXGRID.TDecisionGrid.OnKeyUp 17648
MXGRID.TDecisionGrid.OnMouseDown 17650
MXGRID.TDecisionGrid.OnMouseMove 17652
MXGRID.TDecisionGrid.OnMouseUp 17654
MXGRID.TDecisionGrid.OnResize 17656
MXGRID.TDecisionGrid.OnStartDrag 17657
MXGRID.TDecisionGrid.OnTopLeftChanged 17658
MXGRID.TDecisionGrid.Options 17617
MXGRID.TDecisionGrid.ParentColor 17618
MXGRID.TDecisionGrid.ParentCtl3D 17618
MXGRID.TDecisionGrid.ParentFont 17619
MXGRID.TDecisionGrid.ParentShowHint 17619
MXGRID.TDecisionGrid.PopupMenu 17620
MXGRID.TDecisionGrid.RowCount 17624
MXGRID.TDecisionGrid.ScrollBars 17625
MXGRID.TDecisionGrid.ShowCubeEditor 17625
MXGRID.TDecisionGrid.ShowHint 17625
MXGRID.TDecisionGrid.TabOrder 17627
MXGRID.TDecisionGrid.TabStop 17627
MXGRID.TDecisionGrid.Totals 17628
MXGRID.TDecisionGrid.Visible 17628
MXGRID.TDecisionGridDataLink 17658
MXGRID.TDecisionGridDataLink.Create 17660
MXGRID.TDecisionGridDataLink.DecisionDataEvent 17660
MXGRID.TDecisionGridDataLink.Destroy 17660
MXGRID.TDecisionGridOption 17674
MXGRID.TDecisionGridOption enumeration 17674
MXGRID.TDecisionGridOptions 17677
MXGRID.TDecisionGridOptions type 17677
MXGRID.TDecisionGridState 17675
MXGRID.TDecisionGridState enumeration 17675
MXGRID.TDecisionPivotState 17675
MXGRID.TDecisionPivotState enumeration 17675
MXGRID.TDisplayDim 17661
MXGRID.TDisplayDim.Alignment 17664
MXGRID.TDisplayDim.assign 17663
MXGRID.TDisplayDim.Color 17664
MXGRID.TDisplayDim.Create 17662
MXGRID.TDisplayDim.DisplayName 17665
MXGRID.TDisplayDim.FieldName 17665
MXGRID.TDisplayDim.Format 17665
MXGRID.TDisplayDim.Subtotals 17666
MXGRID.TDisplayDimClass 17677
MXGRID.TDisplayDimClass type 17677
MXGRID.TDisplayDims 17666
MXGRID.TDisplayDims.FGrid 17669
MXGRID.TDisplayDims.GetOwner 17669
MXGRID.TDisplayDims.Items 17670
MXGRID.TDisplayDims.NotifyOwner 17670
MXGRID.TValueArray 17671
MXPIVSRC 17679
MXPIVSRC namespace 17679
    Classes 17680
    Structs, Records, Enums 17741
    Types 17743
MXPIVSRC.TDecisionButtonPosition 17741
MXPIVSRC.TDecisionButtonPosition enumeration 17741
MXPIVSRC.TDecisionPivot 17680
MXPIVSRC.TDecisionPivot.AdjustClientRect 17692
MXPIVSRC.TDecisionPivot.Align 17696
MXPIVSRC.TDecisionPivot.Alignment 17700
MXPIVSRC.TDecisionPivot.BevelInner 17700
MXPIVSRC.TDecisionPivot.BevelOuter 17701
MXPIVSRC.TDecisionPivot.BevelWidth 17702
MXPIVSRC.TDecisionPivot.BorderStyle 17703
MXPIVSRC.TDecisionPivot.BorderWidth 17704
MXPIVSRC.TDecisionPivot.ButtonAutoSize 17704
MXPIVSRC.TDecisionPivot.ButtonHeight 17705
MXPIVSRC.TDecisionPivot.ButtonSpacing 17705
MXPIVSRC.TDecisionPivot.ButtonWidth 17705
MXPIVSRC.TDecisionPivot.Color 17706
MXPIVSRC.TDecisionPivot.Cols 17707

MXPIVSRC.TDecisionPivot.Create 17691
MXPIVSRC.TDecisionPivot.Ctl3D 17707
MXPIVSRC.TDecisionPivot.DecisionSource 17707
MXPIVSRC.TDecisionPivot.Destroy 17692
MXPIVSRC.TDecisionPivot.DimStateChange 17693
MXPIVSRC.TDecisionPivot.DragCursor 17708
MXPIVSRC.TDecisionPivot.DragMode 17708
MXPIVSRC.TDecisionPivot.Enabled 17710
MXPIVSRC.TDecisionPivot.Extras 17710
MXPIVSRC.TDecisionPivot.Font 17710
MXPIVSRC.TDecisionPivot.GetBorderStyle 17693
MXPIVSRC.TDecisionPivot.GetBorderWidth 17693
MXPIVSRC.TDecisionPivot.GroupLayout 17711
MXPIVSRC.TDecisionPivot.Groups 17712
MXPIVSRC.TDecisionPivot.GroupSpacing 17712
MXPIVSRC.TDecisionPivot.NewDimLayout 17694
MXPIVSRC.TDecisionPivot.NewPanelSetup 17694
MXPIVSRC.TDecisionPivot.Notification 17694
MXPIVSRC.TDecisionPivot.OnClick 17724
MXPIVSRC.TDecisionPivot.OnDblClick 17726
MXPIVSRC.TDecisionPivot.OnDragDrop 17726
MXPIVSRC.TDecisionPivot.OnDragOver 17728
MXPIVSRC.TDecisionPivot.OnEndDrag 17731
MXPIVSRC.TDecisionPivot.OnEnter 17732
MXPIVSRC.TDecisionPivot.OnExit 17734
MXPIVSRC.TDecisionPivot.OnResize 17736
MXPIVSRC.TDecisionPivot.OnStartDrag 17737
MXPIVSRC.TDecisionPivot.Pages 17713
MXPIVSRC.TDecisionPivot.ParentColor 17713
MXPIVSRC.TDecisionPivot.ParentCtl3D 17713
MXPIVSRC.TDecisionPivot.ParentFont 17714
MXPIVSRC.TDecisionPivot.ParentShowHint 17714
MXPIVSRC.TDecisionPivot.PopupMenu 17715
MXPIVSRC.TDecisionPivot.Rows 17719
MXPIVSRC.TDecisionPivot.SetBorderStyle 17695
MXPIVSRC.TDecisionPivot.SetBorderWidth 17695
MXPIVSRC.TDecisionPivot.SetBounds 17695
MXPIVSRC.TDecisionPivot.ShowHint 17720
MXPIVSRC.TDecisionPivot.TabOrder 17721
MXPIVSRC.TDecisionPivot.TabStop 17722
MXPIVSRC.TDecisionPivot.Visible 17722
MXPIVSRC.TDecisionPivotOption 17741
MXPIVSRC.TDecisionPivotOption enumeration 17741
MXPIVSRC.TDecisionPivotOptions 17743
MXPIVSRC.TDecisionPivotOptions type 17743
MXPIVSRC.TPivotDataLink 17737
MXPIVSRC.TPivotDataLink.Create 17739
MXPIVSRC.TPivotDataLink.DecisionDataEvent 17739
MXPIVSRC.TPivotDataLink.Destroy 17739
Mxstore 17745
Mxstore namespace 17745
    Classes 17747
    Constants 17849
    Functions 17836
    Structs, Records, Enums 17837
    Types 17843
Mxstore.ECacheError 17747
Mxstore.LargeValueCount 17849
Mxstore.LargeValueCount constant 17849
Mxstore.MaxBinDimensions 17849
Mxstore.MaxBinDimensions constant 17849
Mxstore.NonSparseAgg 17850
Mxstore.NonSparseAgg constant 17850
Mxstore.PIndexInfoRec 17843
Mxstore.PIndexInfoRec type 17843
Mxstore.SparseAgg 17850
Mxstore.SparseAgg constant 17850
Mxstore.SparseSum 17850
Mxstore.SparseSum constant 17850
Mxstore.SparseUnknown 17851
Mxstore.SparseUnknown constant 17851
Mxstore.SubTotal 17851
Mxstore.SubTotal constant 17851
Mxstore.TAggDefinition 17748
Mxstore.TAggDefinition.AggProc 17748
Mxstore.TAggProc 17843
Mxstore.TAggProc type 17843
Mxstore.TBuilderDim 17749

Mxstore.TBuilderDim.Add 17751
Mxstore.TBuilderDim.AddSummary 17751
Mxstore.TBuilderDim.Create 17750
Mxstore.TBuilderDim.Destroy 17750
Mxstore.TBuilderDim.GetLastVal 17751
Mxstore.TBuilderDim.GetSumCount 17752
Mxstore.TBuilderDim.GetSummary 17752
Mxstore.TBuilderDim.GroupBreak 17753
Mxstore.TBuilderDim.InitSummary 17752
Mxstore.TBuilderDim.LastVal 17753
Mxstore.TBuilderDim.MatchLastVal 17752
Mxstore.TBuilderDim.SetLastVal 17753
Mxstore.TBuilderDim.SumCount 17753
Mxstore.TBuildType 17837
Mxstore.TBuildType enumeration 17837
Mxstore.TCacheState 17844
Mxstore.TCacheState type 17844
Mxstore.TCacheStateFlags 17837
Mxstore.TCacheStateFlags enumeration 17837
Mxstore.TCalcTotals 17844
Mxstore.TCalcTotals type 17844
Mxstore.TCalcTotalsFlags 17838
Mxstore.TCalcTotalsFlags enumeration 17838
Mxstore.TCapacityErrorEvent 17844
Mxstore.TCapacityErrorEvent type 17844
Mxstore.TCubeDataState 17838
Mxstore.TCubeDataState enumeration 17838
Mxstore.TCubeDim 17754
Mxstore.TCubeDim.Assign 17757
Mxstore.TCubeDim.AssignBinTypeFormat 17758
Mxstore.TCubeDim.AssignBinTypeTransform 17758
Mxstore.TCubeDim.BinData 17762
Mxstore.TCubeDim.BinFormat 17763
Mxstore.TCubeDim.BinType 17763
Mxstore.TCubeDim.Create 17756
Mxstore.TCubeDim.DataSetTransform 17759
Mxstore.TCubeDim.DefineProperties 17759
Mxstore.TCubeDim.Destroy 17757
Mxstore.TCubeDim.Dirty 17764
Mxstore.TCubeDim.DoTransform 17759
Mxstore.TCubeDim.GetBinValues 17760
Mxstore.TCubeDim.GetLoaded 17760
Mxstore.TCubeDim.InitializeRange 17760
Mxstore.TCubeDim.IsBinData 17761
Mxstore.TCubeDim.Loaded 17764
Mxstore.TCubeDim.MonthTransform 17761
Mxstore.TCubeDim.NotifyCollection 17761
Mxstore.TCubeDim.OnTransform 17766
Mxstore.TCubeDim.QuarterTransform 17761
Mxstore.TCubeDim.SetLoaded 17762
Mxstore.TCubeDim.StartDate 17765
Mxstore.TCubeDim.StartValue 17765
Mxstore.TCubeDim.ValueCount 17765
Mxstore.TCubeDim.wasActive 17766
Mxstore.TCubeDim.YearTransform 17762
Mxstore.TCubeDimClass 17845
Mxstore.TCubeDimClass type 17845
Mxstore.TCubeDims 17766
Mxstore.TCubeDims.Add 17769
Mxstore.TCubeDims.Assign 17770
Mxstore.TCubeDims.Create 17769
Mxstore.TCubeDims.GetDirtyFlag 17770
Mxstore.TCubeDims.GetOwner 17770
Mxstore.TCubeDims.IsDirty 17771
Mxstore.TCubeDims.Items 17771
Mxstore.TCubeDimTransformEvent 17845
Mxstore.TCubeDimTransformEvent type 17845
Mxstore.TCubeNotifyEvent 17845
Mxstore.TCubeNotifyEvent type 17845
Mxstore.TCubeRefreshEvent 17846
Mxstore.TCubeRefreshEvent type 17846
Mxstore.TCubeState 17839
Mxstore.TCubeState enumeration 17839
Mxstore.TCustomDataStore 17772
Mxstore.TCustomDataStore.Active 17787
Mxstore.TCustomDataStore.ActiveChanged 17778
Mxstore.TCustomDataStore.AfterClose 17795
Mxstore.TCustomDataStore.AfterOpen 17795

Mxstore.TCustomDataStore.BeforeClose 17796
Mxstore.TCustomDataStore.BeforeOpen 17796
Mxstore.TCustomDataStore.BinData 17787
Mxstore.TCustomDataStore.BinMapHasBinData 17778
Mxstore.TCustomDataStore.CalcSubTotals 17778
Mxstore.TCustomDataStore.CanDimBeClosed 17778
Mxstore.TCustomDataStore.CanSumBeClosed 17779
Mxstore.TCustomDataStore.Capacity 17788
Mxstore.TCustomDataStore.Create 17776
Mxstore.TCustomDataStore.CurrentSummary 17788
Mxstore.TCustomDataStore.DataCache 17789
Mxstore.TCustomDataStore.DataLink 17789
Mxstore.TCustomDataStore.DataSet 17789
Mxstore.TCustomDataStore.DataSource 17790
Mxstore.TCustomDataStore.DesignState 17790
Mxstore.TCustomDataStore.Destroy 17777
Mxstore.TCustomDataStore.DimensionCount 17790
Mxstore.TCustomDataStore.DimensionMap 17790
Mxstore.TCustomDataStore.DimensionMapCount 17791
Mxstore.TCustomDataStore.DoAfterClose 17779
Mxstore.TCustomDataStore.DoAfterOpen 17779
Mxstore.TCustomDataStore.DoBeforeClose 17779
Mxstore.TCustomDataStore.DoBeforeOpen 17780
Mxstore.TCustomDataStore.DoOnRefresh 17780
Mxstore.TCustomDataStore.GetCurrentSummary 17780
Mxstore.TCustomDataStore.GetDataSet 17780
Mxstore.TCustomDataStore.GetDimensionMemberCount 17780
Mxstore.TCustomDataStore.GetDimensionName 17781
Mxstore.TCustomDataStore.GetDomain 17781
Mxstore.TCustomDataStore.GetMemberAsString 17782
Mxstore.TCustomDataStore.GetMemberAsVariant 17782
Mxstore.TCustomDataStore.GetMemoryUsage 17783
Mxstore.TCustomDataStore.GetSummaryAsString 17783
Mxstore.TCustomDataStore.GetSummaryAsVariant 17784
Mxstore.TCustomDataStore.GetSummaryName 17784
Mxstore.TCustomDataStore.InternalDataSource 17792
Mxstore.TCustomDataStore.LayoutChanged 17785
Mxstore.TCustomDataStore.MaxCells 17792
Mxstore.TCustomDataStore.MaxDimensions 17793
Mxstore.TCustomDataStore.MaxSummaries 17793
Mxstore.TCustomDataStore.Notification 17785
Mxstore.TCustomDataStore.OnLowCapacity 17796
Mxstore.TCustomDataStore.OnRefresh 17797
Mxstore.TCustomDataStore.Refresh 17785
Mxstore.TCustomDataStore.SetCurrentSummary 17786
Mxstore.TCustomDataStore.SetDataSet 17786
Mxstore.TCustomDataStore.ShowProgressDialog 17794
Mxstore.TCustomDataStore.State 17794
Mxstore.TCustomDataStore.StateChanged 17786
Mxstore.TCustomDataStore.SummaryCount 17794
Mxstore.TDataCache 17797
Mxstore.TDataCache.Active 17806
Mxstore.TDataCache.AddAggIndex 17800
Mxstore.TDataCache.AddIndex 17800
Mxstore.TDataCache.AppendDimension 17800
Mxstore.TDataCache.AppendSummary 17801
Mxstore.TDataCache.CalcSubTotals 17801
Mxstore.TDataCache.ClearIndexInfo 17801
Mxstore.TDataCache.Create 17799
Mxstore.TDataCache.CreateTable 17801
Mxstore.TDataCache.CurrentSummary 17807
Mxstore.TDataCache.Destroy 17799
Mxstore.TDataCache.DimensionCount 17807
Mxstore.TDataCache.DimensionFromFieldName 17801
Mxstore.TDataCache.Dimensions 17807
Mxstore.TDataCache.ErrorCode 17807
Mxstore.TDataCache.FreeCache 17802
Mxstore.TDataCache.GetDimensionMember 17802
Mxstore.TDataCache.GetDimensionMemberAsVariant 17802
Mxstore.TDataCache.GetDimensionMemberCount 17802
Mxstore.TDataCache.GetDimensionName 17803
Mxstore.TDataCache.GetDomain 17803
Mxstore.TDataCache.GetIndexCount 17803
Mxstore.TDataCache.GetMemoryUsage 17803
Mxstore.TDataCache.GetSummaryAsString 17803
Mxstore.TDataCache.GetSummaryAsVariant 17804
Mxstore.TDataCache.GetSummaryName 17804
Mxstore.TDataCache.HasIndex 17807

Mxstore.TDataCache.HasSubTotals 17804
Mxstore.TDataCache.HasValidSubTotals 17804
Mxstore.TDataCache.IncSummaryIndex 17805
Mxstore.TDataCache.IndexCount 17808
Mxstore.TDataCache.IsDimension 17805
Mxstore.TDataCache.IsSummary 17805
Mxstore.TDataCache.PreCalculateTotals 17808
Mxstore.TDataCache.Sparsing 17808
Mxstore.TDataCache.Success 17808
Mxstore.TDataCache.Summaries 17809
Mxstore.TDataCache.SummaryCount 17809
Mxstore.TDataCache.SummaryFromCubeDimIndex 17805
Mxstore.TDataCache.SummaryFromFieldName 17805
Mxstore.TDataCache.SummaryFromPosition 17806
Mxstore.TDerivedAggProc 17846
Mxstore.TDerivedAggProc type 17846
Mxstore.TDimension 17809
Mxstore.TDimension.AssignSorted 17811
Mxstore.TDimension.Attributes 17813
Mxstore.TDimension.ClearFlag 17811
Mxstore.TDimension.Create 17810
Mxstore.TDimension.Destroy 17811
Mxstore.TDimension.DimensionName 17813
Mxstore.TDimension.FieldDefinition 17813
Mxstore.TDimension.FieldName 17813
Mxstore.TDimension.FieldType 17814
Mxstore.TDimension.HasFlag 17812
Mxstore.TDimension.IsString 17812
Mxstore.TDimension.Position 17814
Mxstore.TDimension.Range 17814
Mxstore.TDimension.SetFlag 17812
Mxstore.TDimension.SetRangeCounting 17812
Mxstore.TDimFlagSet 17846
Mxstore.TDimFlagSet type 17846
Mxstore.TErrorAction 17839
Mxstore.TErrorAction enumeration 17839
Mxstore.TestMatch 17836
Mxstore.TestMatch function 17836
Mxstore.TFieldArgs 17847
Mxstore.TFieldArgs type 17847
Mxstore.TFieldDefinition 17814
Mxstore.TFieldDefinition.Create 17815
Mxstore.TFieldDefinition.DisplayName 17816
Mxstore.TFieldDefinition.FieldNo 17817
Mxstore.TFieldDefinition.FieldType 17817
Mxstore.TFieldDefinition.FormatString 17817
Mxstore.TFieldDefinition.FormatVariantToStr 17815
Mxstore.TFieldDefinition.Precision 17817
Mxstore.TFieldDefinition.SetFieldType 17816
Mxstore.TFieldDefinition.SetName 17816
Mxstore.TFieldDefinition.Width 17817
Mxstore.TFormatType 17840
Mxstore.TFormatType enumeration 17840
Mxstore.TIndexFlag 17840
Mxstore.TIndexFlag enumeration 17840
Mxstore.TIndexFlags 17847
Mxstore.TIndexFlags type 17847
Mxstore.TIndexInfo 17818
Mxstore.TIndexInfo.Add 17819
Mxstore.TIndexInfo.AddAggs 17821
Mxstore.TIndexInfo.AddOffset 17819
Mxstore.TIndexInfo.Capacity 17821
Mxstore.TIndexInfo.Count 17822
Mxstore.TIndexInfo.Create 17818
Mxstore.TIndexInfo.Destroy 17819
Mxstore.TIndexInfo.GetCapacity 17820
Mxstore.TIndexInfo.IsSparse 17820
Mxstore.TIndexInfo.IsSparseAgg 17820
Mxstore.TIndexInfo.LockIndex 17820
Mxstore.TIndexInfo.SetCapacity 17821
Mxstore.TIndexInfo.UnlockIndex 17821
Mxstore.TIndexInfoRec 17841
Mxstore.TIndexInfoRec record 17841
Mxstore.TLookupState 17847
Mxstore.TLookupState type 17847
Mxstore.TLookupStateFlags 17841
Mxstore.TLookupStateFlags enumeration 17841
Mxstore.TMultiDimDataLink 17822

Mxstore.TMultiDimDataLink.ActiveChanged 17825
Mxstore.TMultiDimDataLink.AddDimension 17825
Mxstore.TMultiDimDataLink.AddSummary 17826
Mxstore.TMultiDimDataLink.Create 17824
Mxstore.TMultiDimDataLink.CreateSummaryIndex 17826
Mxstore.TMultiDimDataLink.Destroy 17824
Mxstore.TMultiDimDataLink.FetchAndBinValues 17826
Mxstore.TMultiDimDataLink.FetchValues 17826
Mxstore.TMultiDimDataLink.LayoutChanged 17827
Mxstore.TMultiDimDataLink.UpdateCache 17827
Mxstore.TMultiDimDataLink.UpdateDimensions 17827
Mxstore.TMultiDimDataLink.UpdateFormatStrings 17827
Mxstore.TSummary 17828
Mxstore.TSummary.AddIndexInfo 17830
Mxstore.TSummary.AddSubTotal 17830
Mxstore.TSummary.AddSum 17830
Mxstore.TSummary.AggDefinition 17832
Mxstore.TSummary.Attributes 17833
Mxstore.TSummary.ClearTotals 17831
Mxstore.TSummary.Create 17829
Mxstore.TSummary.CubeDimIndex 17833
Mxstore.TSummary.Destroy 17829
Mxstore.TSummary.FieldDefinition 17833
Mxstore.TSummary.FieldName 17833
Mxstore.TSummary.FieldType 17833
Mxstore.TSummary.HasFlag 17831
Mxstore.TSummary.IsDerived 17834
Mxstore.TSummary.IsSparse 17831
Mxstore.TSummary.MemoryUsage 17831
Mxstore.TSummary.Name 17834
Mxstore.TSummary.Position 17834
Mxstore.TSummary.SetAggregator 17832
Mxstore.TSummary.SumMethod 17834
Mxstore.TSummary.UpdateIndexInfo 17832
Mxstore.TSumMethod 17848
Mxstore.TSumMethod type 17848
MXTABLES 17853
MXTABLES namespace 17853
    Classes 17854
    Functions 17862
    Types 17864
MXTABLES.AddDimensionItem 17862
MXTABLES.AddDimensionItem function 17862
MXTABLES.BuildDimensionMap 17862
MXTABLES.BuildDimensionMap function 17862
MXTABLES.BuildMasterDimensionMap 17863
MXTABLES.BuildMasterDimensionMap function 17863
MXTABLES.RemoveDimensionItem 17863
MXTABLES.RemoveDimensionItem function 17863
MXTABLES.TDecisionQuery 17854
MXTABLES.TQueryDim 17855
MXTABLES.TQueryDim.Active 17857
MXTABLES.TQueryDimClass 17864
MXTABLES.TQueryDimClass type 17864
MXTABLES.TQueryDims 17857
MXTABLES.TQueryDims.Add 17860
MXTABLES.TQueryDims.Create 17859
MXTABLES.TQueryDims.GetQueryDim 17860
MXTABLES.TQueryDims.Items 17861
MXTABLES.TQueryDims.SetQueryDim 17860

N

naAdd enumeration member 5457
naAddChild enumeration member 5457
naAddChildFirst enumeration member 5457
naAddFirst enumeration member 5457
naInsert enumeration member 5457
NamedParm class 22187
    about NamedParm class 22187
    AddElement 22192
    GetParm 22192
    GetParmName 22193
    GetType 22193
    Name 22191
    NamedParm 22190
    NamedParm constructor 22188
    NamedParm data members 22191
    NamedParm Members 22188

NamedParm methods 22192
Parm 22191
Type 22192
nbCancel enumeration member 9539
nbDelete enumeration member 9539
nbEdit enumeration member 9539
nbFirst enumeration member 9539
nbInsert enumeration member 9539
nbLast enumeration member 9539
nbNext enumeration member 9539
nbPost enumeration member 9539
nbPrior enumeration member 9539
nbRefresh enumeration member 9539
ncAddAttribute enumeration member 26243
ncInsertChild enumeration member 26243
ncRemoveAttribute enumeration member 26243
ncRemoveChild enumeration member 26243
ncrError enumeration member 23671
ncrLoose enumeration member 23671
ncrStrict enumeration member 23671
ncUpdateValue enumeration member 26243
ngDisabled enumeration member 9539
ngEnabled enumeration member 9539
nlCreateNode enumeration member 26244
nlInsert enumeration member 26244
nlRemove enumeration member 26244
nsBullet enumeration member 5459
nsClientReceive enumeration member 18179
nsClientSend enumeration member 18179
nsCut enumeration member 5459
nsDropHilited enumeration member 5459
nsExpanded enumeration member 5459
nsFocused enumeration member 5459
nsNone enumeration member 5459
nsSelected enumeration member 5459
nsServerReceive enumeration member 18179
nsServerSend enumeration member 18179
ntAttribute enumeration member 26458
ntCData enumeration member 26458
ntComment enumeration member 26458
ntContainsUnit enumeration member 23005
ntDcpBpiName enumeration member 23005
ntDocFragment enumeration member 26458
ntDocType enumeration member 26458
ntDocument enumeration member 26458
ntElement enumeration member 26458
ntEntity enumeration member 26458
ntEntityRef enumeration member 26458
ntNotation enumeration member 26458
ntProcessingInstr enumeration member 26458
ntRequiresPackage enumeration member 23005
ntReserved enumeration member 26458
ntText enumeration member 26458
nvAborted enumeration member 17234
nvFailure enumeration member 17234
nvSuccessful enumeration member 17234
nvSuperseded enumeration member 17234

O

oaAdd enumeration member 18366
oaAddChild enumeration member 18366
oaInsert enumeration member 18366
obClose enumeration member 18367
ObjBrkr 17865
ObjBrkr namespace 17865
Classes 17866
ObjBrkr.EBrokerException 17866
ObjBrkr.TServerCollection 17866
ObjBrkr.TServerCollection.Create 17869
ObjBrkr.TServerCollection.FindServer 17869
ObjBrkr.TServerCollection.GetBalancedName 17870
ObjBrkr.TServerCollection.GetNextName 17870
ObjBrkr.TServerCollection.Items 17871
ObjBrkr.TServerItem 17871
ObjBrkr.TServerItem.ComputerName 17874
ObjBrkr.TServerItem.Create 17873
ObjBrkr.TServerItem.Enabled 17875
ObjBrkr.TServerItem.GetDisplayName 17874

ObjBrkr.TServerItem.HasFailed 17876
ObjBrkr.TServerItem.Port 17876
ObjBrkr.TSimpleObjectBroker 17877
ObjBrkr.TSimpleObjectBroker.Create 17880
ObjBrkr.TSimpleObjectBroker.Destroy 17880
ObjBrkr.TSimpleObjectBroker.GetComputerForGUID 17881
ObjBrkr.TSimpleObjectBroker.GetComputerForProgID 17882
ObjBrkr.TSimpleObjectBroker.GetPortForComputer 17882
ObjBrkr.TSimpleObjectBroker.LoadBalanced 17884
ObjBrkr.TSimpleObjectBroker.LoadFromStream 17883
ObjBrkr.TSimpleObjectBroker.SaveToStream 17883
ObjBrkr.TSimpleObjectBroker.ServerData 17885
ObjBrkr.TSimpleObjectBroker.Servers 17885
ObjBrkr.TSimpleObjectBroker.SetConnectStatus 17884
obLeaf enumeration member 18367
obMinus enumeration member 18367
obOpen enumeration member 18367
obPlus enumeration member 18367
ocDefault enumeration member 15633
ocGreater enumeration member 18367
ocInvalid enumeration member 18367
ocLess enumeration member 18367
ocMultiRef enumeration member 15633
ocNoMultiRef enumeration member 15633
ocoDontPrefixNode enumeration member 15629
ocoDontPutTypeAttr enumeration member 15629
ocoDontSerializeProps enumeration member 15629
ocSame enumeration member 18367
ofAllowMultiSelect enumeration member 11559
ofCreatePrompt enumeration member 11559
ofDontAddToRecent enumeration member 11559
ofEnableIncludeNotify enumeration member 11559
ofEnableSizing enumeration member 11559
ofExNoPlacesBar enumeration member 11561
ofExtensionDifferent enumeration member 11559
ofFileMustExist enumeration member 11559
ofForceShowHidden enumeration member 11559
ofHideReadOnly enumeration member 11559
ofNoChangeDir enumeration member 11559
ofNoDereferenceLinks enumeration member 11559
ofNoLongNames enumeration member 11559
ofNoNetworkButton enumeration member 11559
ofNoReadOnlyReturn enumeration member 11559
ofNoTestFileCreate enumeration member 11559
ofNoValidate enumeration member 11559
ofOldStyleDialog enumeration member 11559
ofOverwritePrompt enumeration member 11559
ofPathMustExist enumeration member 11559
ofReadOnly enumeration member 11559
ofShareAware enumeration member 11559
ofShowHelp enumeration member 11559
OleAuto 17887
OleAuto namespace 17887
    Classes 17889
    Constants 17917
    Functions 17907
    Structs, Records, Enums 17913
    Types 17915
    Variables 17916
OleAuto.Automation 17916
OleAuto.Automation variable 17916
OleAuto.ClassIDToProgID 17907
OleAuto.ClassIDToProgID function 17907
OleAuto.ClassIDToString 17907
OleAuto.ClassIDToString function 17907
OleAuto.CreateOleObject 17908
OleAuto.CreateOleObject function 17908
OleAuto.DispInvoke 17908
OleAuto.DispInvoke function 17908
OleAuto.DispInvokeError 17908
OleAuto.DispInvokeError function 17908
OleAuto.DllCanUnloadNow 17909
OleAuto.DllCanUnloadNow function 17909
OleAuto.DllGetClassObject 17909
OleAuto.DllGetClassObject function 17909
OleAuto.DllRegisterServer 17909
OleAuto.DllRegisterServer function 17909
OleAuto.DllUnregisterServer 17910

OleAuto.DllUnregisterServer function 17910
OleAuto.EOleError 17889
OleAuto.EOleException 17889
OleAuto.EOleException.Create 17891
OleAuto.EOleException.ErrorCode 17891
OleAuto.EOleException.HelpFile 17891
OleAuto.EOleException.Source 17892
OleAuto.EOleSysError 17892
OleAuto.EOleSysError.Create 17893
OleAuto.EOleSysError.ErrorCode 17894
OleAuto.GetActiveOleObject 17910
OleAuto.GetActiveOleObject function 17910
OleAuto.MaxDispArgs 17917
OleAuto.MaxDispArgs constant 17917
OleAuto.OleCheck 17910
OleAuto.OleCheck function 17910
OleAuto.OleError 17911
OleAuto.OleError function 17911
OleAuto.ProgIDToClassID 17911
OleAuto.ProgIDToClassID function 17911
OleAuto.StringToClassID 17911
OleAuto.StringToClassID function 17911
OleAuto.TAutoClass 17915
OleAuto.TAutoClass type 17915
OleAuto.TAutoClassInfo 17913
OleAuto.TAutoClassInfo record 17913
OleAuto.TAutoClassInstancing 17913
OleAuto.TAutoClassInstancing enumeration 17913
OleAuto.TAutoDispatch 17894
OleAuto.TAutoDispatch.AddRef 17895
OleAuto.TAutoDispatch.AutoObject 17897
OleAuto.TAutoDispatch.Create 17895
OleAuto.TAutoDispatch.GetAutoObject 17895
OleAuto.TAutoDispatch.GetIDsOfNames 17896
OleAuto.TAutoDispatch.GetTypeInfo 17896
OleAuto.TAutoDispatch.GetTypeInfoCount 17896
OleAuto.TAutoDispatch.Invoke 17896
OleAuto.TAutoDispatch.QueryInterface 17897
OleAuto.TAutoDispatch.Release 17897
OleAuto.TAutomation 17897
OleAuto.TAutomation.AutoObjectCount 17899
OleAuto.TAutomation.Create 17898
OleAuto.TAutomation.Destroy 17898
OleAuto.TAutomation.IsInprocServer 17900
OleAuto.TAutomation.OnLastRelease 17900
OleAuto.TAutomation.RegisterClass 17899
OleAuto.TAutomation.StartMode 17900
OleAuto.TAutomation.UpdateRegistry 17899
OleAuto.TAutoObject 17901
OleAuto.TAutoObject.AddRef 17903
OleAuto.TAutoObject.AutoDispatch 17904
OleAuto.TAutoObject.Create 17902
OleAuto.TAutoObject.CreateAutoDispatch 17903
OleAuto.TAutoObject.Destroy 17902
OleAuto.TAutoObject.GetExceptionInfo 17904
OleAuto.TAutoObject.OleObject 17904
OleAuto.TAutoObject.RefCount 17905
OleAuto.TAutoObject.Release 17904
OleAuto.TLastReleaseEvent 17915
OleAuto.TLastReleaseEvent type 17915
OleAuto.TRegistryClass 17905
OleAuto.TRegistryClass.Create 17905
OleAuto.TRegistryClass.Destroy 17906
OleAuto.TRegistryClass.UpdateRegistry 17906
OleAuto.TStartMode 17914
OleAuto.TStartMode enumeration 17914
OleAuto.VarFromInterface 17912
OleAuto.VarFromInterface function 17912
OleAuto.VarToAutoObject 17912
OleAuto.VarToAutoObject function 17912
OleAuto.VarToInterface 17912
OleAuto.VarToInterface function 17912
OleCtnrs 17919
OleCtnrs namespace 17919
Classes 17921
Constants 18017
Functions 18011
Interfaces 18009

Structs, Records, Enums 18012
Types 18016
OleCtnrs.DestroyMetaPict 18011
OleCtnrs.DestroyMetaPict function 18011
OleCtnrs.IVCLFrameForm 18009
OleCtnrs.IVCLFrameForm.AddContainer 18010
OleCtnrs.IVCLFrameForm.ClearBorderSpace 18010
OleCtnrs.IVCLFrameForm.Form 18010
OleCtnrs.IVCLFrameForm.RemoveContainer 18010
OleCtnrs.ovDiscardUndoState 18017
OleCtnrs.ovDiscardUndoState constant 18017
OleCtnrs.ovHide 18017
OleCtnrs.ovHide constant 18017
OleCtnrs.ovInPlaceActivate 18018
OleCtnrs.ovInPlaceActivate constant 18018
OleCtnrs.ovOpen 18018
OleCtnrs.ovOpen constant 18018
OleCtnrs.ovPrimary 18018
OleCtnrs.ovPrimary constant 18018
OleCtnrs.ovShow 18019
OleCtnrs.ovShow constant 18019
OleCtnrs.ovUIActivate 18019
OleCtnrs.ovUIActivate constant 18019
OleCtnrs.TAutoActivate 18012
OleCtnrs.TAutoActivate enumeration 18012
OleCtnrs.TCreateInfo 18012
OleCtnrs.TCreateInfo record 18012
OleCtnrs.TCreateType 18013
OleCtnrs.TCreateType enumeration 18013
OleCtnrs.TObjectMoveEvent 18016
OleCtnrs.TObjectMoveEvent type 18016
OleCtnrs.TObjectState 18013
OleCtnrs.TObjectState enumeration 18013
OleCtnrs.TOleContainer 17921
OleCtnrs.TOleContainer._AddRef 17935
OleCtnrs.TOleContainer._Release 17935
OleCtnrs.TOleContainer.Align 17951
OleCtnrs.TOleContainer.AllowActiveDoc 17954
OleCtnrs.TOleContainer.AllowInPlace 17955
OleCtnrs.TOleContainer.Anchors 17955
OleCtnrs.TOleContainer.AutoActivate 17956
OleCtnrs.TOleContainer.AutoVerbMenu 17956
OleCtnrs.TOleContainer.BorderStyle 17957
OleCtnrs.TOleContainer.CanPaste 17957
OleCtnrs.TOleContainer.Caption 17957
OleCtnrs.TOleContainer.Changed 17936
OleCtnrs.TOleContainer.ChangeIconDialog 17936
OleCtnrs.TOleContainer.Close 17937
OleCtnrs.TOleContainer.Color 17960
OleCtnrs.TOleContainer.Constraints 17961
OleCtnrs.TOleContainer.Copy 17937
OleCtnrs.TOleContainer.CopyOnSave 17961
OleCtnrs.TOleContainer.Create 17933
OleCtnrs.TOleContainer.CreateLinkToFile 17937
OleCtnrs.TOleContainer.CreateObject 17938
OleCtnrs.TOleContainer.CreateObjectFromFile 17938
OleCtnrs.TOleContainer.CreateObjectFromInfo 17939
OleCtnrs.TOleContainer.CreateParams 17939
OleCtnrs.TOleContainer.Ctl3D 17962
OleCtnrs.TOleContainer.DblClick 17940
OleCtnrs.TOleContainer.DefineProperties 17940
OleCtnrs.TOleContainer.Destroy 17934
OleCtnrs.TOleContainer.DestroyObject 17941
OleCtnrs.TOleContainer.DoEnter 17941
OleCtnrs.TOleContainer.DoVerb 17941
OleCtnrs.TOleContainer.DragCursor 17962
OleCtnrs.TOleContainer.DragMode 17963
OleCtnrs.TOleContainer.Enabled 17964
OleCtnrs.TOleContainer.GetIconMetaPict 17942
OleCtnrs.TOleContainer.GetPopupMenu 17942
OleCtnrs.TOleContainer.Iconic 17965
OleCtnrs.TOleContainer.InsertObjectDialog 17943
OleCtnrs.TOleContainer.KeyDown 17943
OleCtnrs.TOleContainer.Linked 17965
OleCtnrs.TOleContainer.LoadFromFile 17944
OleCtnrs.TOleContainer.LoadFromStream 17944
OleCtnrs.TOleContainer.Modified 17965
OleCtnrs.TOleContainer.MouseDown 17945

OleCtnrs.TOleContainer.NewInserted 17966
OleCtnrs.TOleContainer.ObjectPropertiesDialog 17945
OleCtnrs.TOleContainer.ObjectVerbs 17966
OleCtnrs.TOleContainer.OldStreamFormat 17966
OleCtnrs.TOleContainer.OleClassName 17967
OleCtnrs.TOleContainer.OleObject 17967
OleCtnrs.TOleContainer.OleObjectInterface 17968
OleCtnrs.TOleContainer.OnActivate 17981
OleCtnrs.TOleContainer.OnDeactivate 17981
OleCtnrs.TOleContainer.OnDragDrop 17982
OleCtnrs.TOleContainer.OnDragOver 17984
OleCtnrs.TOleContainer.OnEndDrag 17986
OleCtnrs.TOleContainer.OnEnter 17987
OleCtnrs.TOleContainer.OnExit 17990
OleCtnrs.TOleContainer.OnKeyDown 17991
OleCtnrs.TOleContainer.OnKeyPress 17994
OleCtnrs.TOleContainer.OnKeyUp 17995
OleCtnrs.TOleContainer.OnMouseActivate 17997
OleCtnrs.TOleContainer.OnMouseDown 17998
OleCtnrs.TOleContainer.OnMouseMove 18000
OleCtnrs.TOleContainer.OnMouseUp 18002
OleCtnrs.TOleContainer.OnObjectMove 18005
OleCtnrs.TOleContainer.OnResize 18005
OleCtnrs.TOleContainer.OnStartDrag 18005
OleCtnrs.TOleContainer.Paint 17945
OleCtnrs.TOleContainer.ParentColor 17968
OleCtnrs.TOleContainer.ParentCtl3D 17969
OleCtnrs.TOleContainer.ParentShowHint 17969
OleCtnrs.TOleContainer.Paste 17946
OleCtnrs.TOleContainer.PasteSpecialDialog 17946
OleCtnrs.TOleContainer.PopupMenu 17969
OleCtnrs.TOleContainer.PrimaryVerb 17974
OleCtnrs.TOleContainer.QueryInterface 17947
OleCtnrs.TOleContainer.Run 17947
OleCtnrs.TOleContainer.SaveAsDocument 17948
OleCtnrs.TOleContainer.SaveToFile 17948
OleCtnrs.TOleContainer.SaveToStream 17949
OleCtnrs.TOleContainer.ShowHint 17974
OleCtnrs.TOleContainer.SizeMode 17976
OleCtnrs.TOleContainer.SourceDoc 17977
OleCtnrs.TOleContainer.State 17977
OleCtnrs.TOleContainer.StorageInterface 17977
OleCtnrs.TOleContainer.TabOrder 17978
OleCtnrs.TOleContainer.TabStop 17978
OleCtnrs.TOleContainer.UpdateObject 17949
OleCtnrs.TOleContainer.UpdateVerbs 17949
OleCtnrs.TOleContainer.Visible 17979
OleCtnrs.TOleForm 18006
OleCtnrs.TOleForm.Create 18008
OleCtnrs.TOleForm.Destroy 18008
OleCtnrs.TSizeMode 18014
OleCtnrs.TSizeMode enumeration 18014
OleCtnrs.TVerbInfo 18014
OleCtnrs.TVerbInfo record 18014
OleCtrls 18021
OleCtrls namespace 18021
    Classes 18023
    Functions 18088
    Structs, Records, Enums 18090
    Types 18093
    Variables 18096
OleCtrls.EmptyParam 18096
OleCtrls.EmptyParam variable 18096
OleCtrls.EOleCtrlError 18023
OleCtrls.FontToOleFont 18088
OleCtrls.FontToOleFont function 18088
OleCtrls.OleFontToFont 18088
OleCtrls.OleFontToFont function 18088
OleCtrls.PControlData 18093
OleCtrls.PControlData type 18093
OleCtrls.PControlData2 18093
OleCtrls.PControlData2 type 18093
OleCtrls.PEnumValueList 18094
OleCtrls.PEnumValueList type 18094
OleCtrls.TControlData 18090
OleCtrls.TControlData record 18090
OleCtrls.TControlData2 18091
OleCtrls.TControlData2 record 18091

OleCtrls.TEnumPropDesc 18023
OleCtrls.TEnumPropDesc.Create 18024
OleCtrls.TEnumPropDesc.Destroy 18025
OleCtrls.TEnumPropDesc.GetStrings 18025
OleCtrls.TEnumPropDesc.StringToValue 18026
OleCtrls.TEnumPropDesc.ValueToString 18026
OleCtrls.TEnumValue 18091
OleCtrls.TEnumValue record 18091
OleCtrls.TEnumValueList 18094
OleCtrls.TEnumValueList type 18094
OleCtrls.TEventDispatch 18026
OleCtrls.TEventDispatch._AddRef 18028
OleCtrls.TEventDispatch._Release 18028
OleCtrls.TEventDispatch.Control 18030
OleCtrls.TEventDispatch.Create 18028
OleCtrls.TEventDispatch.GetIDsOfNames 18029
OleCtrls.TEventDispatch.GetTypeInfo 18029
OleCtrls.TEventDispatch.GetTypeInfoCount 18029
OleCtrls.TEventDispatch.Invoke 18029
OleCtrls.TEventDispatch.QueryInterface 18030
OleCtrls.TOleControl 18030
OleCtrls.TOleControl._AddRef 18046
OleCtrls.TOleControl._Release 18047
OleCtrls.TOleControl._SetColorProp 18047
OleCtrls.TOleControl._SetTColorProp 18048
OleCtrls.TOleControl._SetTFontProp 18048
OleCtrls.TOleControl._SetTOleEnumProp 18048
OleCtrls.TOleControl._SetTPictureProp 18048
OleCtrls.TOleControl.Anchors 18085
OleCtrls.TOleControl.BrowseProperties 18048
OleCtrls.TOleControl.CanInPlaceActivate 18049
OleCtrls.TOleControl.ContextSensitiveHelp 18049
OleCtrls.TOleControl.ControlData 18086
OleCtrls.TOleControl.Create 18043
OleCtrls.TOleControl.CreateWnd 18049
OleCtrls.TOleControl.DeactivateAndUndo 18050
OleCtrls.TOleControl.DefaultDispatch 18086
OleCtrls.TOleControl.DefaultHandler 18050
OleCtrls.TOleControl.DefineProperties 18050
OleCtrls.TOleControl.Destroy 18043
OleCtrls.TOleControl.DestroyWindowHandle 18051
OleCtrls.TOleControl.DiscardUndoState 18051
OleCtrls.TOleControl.DoObjectVerb 18051
OleCtrls.TOleControl.EnableModeless 18052
OleCtrls.TOleControl.GetBorder 18052
OleCtrls.TOleControl.GetByteProp 18052
OleCtrls.TOleControl.GetColorProp 18053
OleCtrls.TOleControl.GetCompProp 18053
OleCtrls.TOleControl.GetContainer 18053
OleCtrls.TOleControl.GetCurrencyProp 18053
OleCtrls.TOleControl.GetDoubleProp 18053
OleCtrls.TOleControl.GetEnumPropDesc 18054
OleCtrls.TOleControl.GetExtendedControl 18054
OleCtrls.TOleControl.GetHelpContext 18054
OleCtrls.TOleControl.GetIDispatchProp 18055
OleCtrls.TOleControl.GetIDsOfNames 18055
OleCtrls.TOleControl.GetIntegerProp 18055
OleCtrls.TOleControl.GetIUnknownProp 18055
OleCtrls.TOleControl.GetMoniker 18056
OleCtrls.TOleControl.GetObjectVerbs 18056
OleCtrls.TOleControl.GetOleBoolProp 18056
OleCtrls.TOleControl.GetOleDateProp 18056
OleCtrls.TOleControl.GetOleEnumProp 18057
OleCtrls.TOleControl.GetOleVariantProp 18057
OleCtrls.TOleControl.GetPropDisplayString 18057
OleCtrls.TOleControl.GetPropDisplayStrings 18057
OleCtrls.TOleControl.GetProperty 18058
OleCtrls.TOleControl.GetShortIntProp 18058
OleCtrls.TOleControl.GetSingleProp 18058
OleCtrls.TOleControl.GetSmallintProp 18058
OleCtrls.TOleControl.GetStringProp 18059
OleCtrls.TOleControl.GetTColorProp 18059
OleCtrls.TOleControl.GetTDateTimeProp 18059
OleCtrls.TOleControl.GetTFontProp 18059
OleCtrls.TOleControl.GetTOleEnumProp 18059
OleCtrls.TOleControl.GetTPictureProp 18060
OleCtrls.TOleControl.GetTypeInfo 18060
OleCtrls.TOleControl.GetTypeInfoCount 18060

wwww

OleCtrls.TOleControl.GetVariantProp 18061
OleCtrls.TOleControl.GetWideStringProp 18061
OleCtrls.TOleControl.GetWindowContext 18061
OleCtrls.TOleControl.GetWordBoolProp 18061
OleCtrls.TOleControl.GetWordProp 18062
OleCtrls.TOleControl.InitControlData 18062
OleCtrls.TOleControl.InitControlInterface 18062
OleCtrls.TOleControl.InsertMenus 18062
OleCtrls.TOleControl.Invoke 18062
OleCtrls.TOleControl.InvokeEvent 18063
OleCtrls.TOleControl.InvokeMethod 18063
OleCtrls.TOleControl.IsCustomProperty 18063
OleCtrls.TOleControl.IsPropPageProperty 18064
OleCtrls.TOleControl.LockInPlaceActive 18064
OleCtrls.TOleControl.OleControlSite_TranslateAccelerator 18064
OleCtrls.TOleControl.OleInPlaceFrame_GetWindow 18064
OleCtrls.TOleControl.OleInPlaceFrame_TranslateAccelerator 18065
OleCtrls.TOleControl.OleInPlaceSite_GetWindow 18065
OleCtrls.TOleControl.OleObject 18086
OleCtrls.TOleControl.OnChanged 18065
OleCtrls.TOleControl.OnControlInfoChanged 18065
OleCtrls.TOleControl.OnFocus 18066
OleCtrls.TOleControl.OnInPlaceActivate 18066
OleCtrls.TOleControl.OnInPlaceDeactivate 18066
OleCtrls.TOleControl.OnPosRectChange 18066
OleCtrls.TOleControl.OnRequestEdit 18066
OleCtrls.TOleControl.OnShowWindow 18067
OleCtrls.TOleControl.OnUIActivate 18067
OleCtrls.TOleControl.OnUIDeactivate 18067
OleCtrls.TOleControl.PaletteChanged 18067
OleCtrls.TOleControl.PerPropBrowsing 18086
OleCtrls.TOleControl.PictureChanged 18068
OleCtrls.TOleControl.PostMessageFilter 18068
OleCtrls.TOleControl.PreMessageFilter 18069
OleCtrls.TOleControl.QueryInterface 18069
OleCtrls.TOleControl.QueryService 18069
OleCtrls.TOleControl.RemoveMenus 18070
OleCtrls.TOleControl.RequestBorderSpace 18070
OleCtrls.TOleControl.RequestNewObjectLayout 18070
OleCtrls.TOleControl.SaveObject 18070
OleCtrls.TOleControl.Scroll 18070
OleCtrls.TOleControl.ServiceQuery 18087
OleCtrls.TOleControl.SetActiveObject 18071
OleCtrls.TOleControl.SetBorderSpace 18071
OleCtrls.TOleControl.SetBounds 18071
OleCtrls.TOleControl.SetByteProp 18071
OleCtrls.TOleControl.SetColorProp 18072
OleCtrls.TOleControl.SetCompProp 18072
OleCtrls.TOleControl.SetCurrencyProp 18072
OleCtrls.TOleControl.SetDoubleProp 18072
OleCtrls.TOleControl.SetIDispatchProp 18073
OleCtrls.TOleControl.SetIntegerProp 18073
OleCtrls.TOleControl.SetIUnknownProp 18073
OleCtrls.TOleControl.SetMenu 18073
OleCtrls.TOleControl.SetName 18073
OleCtrls.TOleControl.SetOleBoolProp 18074
OleCtrls.TOleControl.SetOleDateProp 18074
OleCtrls.TOleControl.SetOleEnumProp 18074
OleCtrls.TOleControl.SetOleVariantProp 18075
OleCtrls.TOleControl.SetParent 18075
OleCtrls.TOleControl.SetPropDisplayString 18075
OleCtrls.TOleControl.SetProperty 18076
OleCtrls.TOleControl.SetShortIntProp 18076
OleCtrls.TOleControl.SetSingleProp 18076
OleCtrls.TOleControl.SetSmallintProp 18076
OleCtrls.TOleControl.SetStatusText 18077
OleCtrls.TOleControl.SetStringProp 18077
OleCtrls.TOleControl.SetTColorProp 18077
OleCtrls.TOleControl.SetTDateTimeProp 18077
OleCtrls.TOleControl.SetTFontProp 18077
OleCtrls.TOleControl.SetTOleEnumProp 18078
OleCtrls.TOleControl.SetTPictureProp 18078
OleCtrls.TOleControl.SetVariantProp 18078
OleCtrls.TOleControl.SetWideStringProp 18078
OleCtrls.TOleControl.SetWordBoolProp 18079
OleCtrls.TOleControl.SetWordProp 18079
OleCtrls.TOleControl.ShowAboutBox 18079

OleCtrls.TOleControl.ShowObject 18079
OleCtrls.TOleControl.ShowPropertyFrame 18079
OleCtrls.TOleControl.StandardEvent 18080
OleCtrls.TOleControl.SuppressException 18080
OleCtrls.TOleControl.TabStop 18087
OleCtrls.TOleControl.TransformCoords 18080
OleCtrls.TOleControl.WndProc 18080
OleCtrls.TOleEnum 18094
OleCtrls.TOleEnum type 18094
OleCtrls.TServiceQuery 18095
OleCtrls.TServiceQuery type 18095
OleServer 18097
OleServer namespace 18097
    Classes 18098
    Structs, Records, Enums 18117
    Types 18119
OleServer.PServerData 18119
OleServer.PServerData type 18119
OleServer.TConnectKind 18117
OleServer.TConnectKind enumeration 18117
OleServer.TOleServer 18098
OleServer.TOleServer._AddRef 18104
OleServer.TOleServer._Release 18104
OleServer.TOleServer.AutoConnect 18110
OleServer.TOleServer.Connect 18104
OleServer.TOleServer.ConnectEvents 18105
OleServer.TOleServer.ConnectKind 18110
OleServer.TOleServer.Create 18101
OleServer.TOleServer.Destroy 18102
OleServer.TOleServer.Disconnect 18105
OleServer.TOleServer.DisconnectEvents 18106
OleServer.TOleServer.EventDispatch 18111
OleServer.TOleServer.GetAutoConnect 18106
OleServer.TOleServer.GetConnectKind 18106
OleServer.TOleServer.GetServer 18107
OleServer.TOleServer.InitServerData 18107
OleServer.TOleServer.InvokeEvent 18107
OleServer.TOleServer.Loaded 18108
OleServer.TOleServer.QueryInterface 18108
OleServer.TOleServer.RemoteMachineName 18111
OleServer.TOleServer.ServerData 18111
OleServer.TOleServer.SetAutoConnect 18109
OleServer.TOleServer.SetConnectKind 18109
OleServer.TServerData 18117
OleServer.TServerData record 18117
OleServer.TServerEventDispatch 18112
OleServer.TServerEventDispatch._AddRef 18114
OleServer.TServerEventDispatch._Release 18114
OleServer.TServerEventDispatch.Create 18113
OleServer.TServerEventDispatch.GetIDsOfNames 18114
OleServer.TServerEventDispatch.GetTypeInfo 18114
OleServer.TServerEventDispatch.GetTypeInfoCount 18114
OleServer.TServerEventDispatch.Invoke 18115
OleServer.TServerEventDispatch.QueryInterface 18115
OleServer.TServerEventDispatch.Server 18116
OleServer.TServerEventDispatch.ServerDisconnect 18115
OleServer.TVariantArray 18119
OleServer.TVariantArray type 18119
OleVariant template 22193
    != 22210
    != operator 22209
    ~OleVariant 22197
    < 22210
    < operator 22210
    <= 22211
    <= operator 22211
    = 22213
    = operator 22211
    == 22214
    == operator 22214
    > 22215
    > operator 22214
    >= 22215
    >= operator 22215
    about OleVariant template 22193
    OleVariant 22200, 22201, 22202, 22203, 22204, 22205, 22206, 22207, 22208, 22209
    OleVariant constructor 22198
    OleVariant Members 22195

OleVariant operators 22209
ooDrawFocusRect enumeration member 18368
ooDrawTreeRoot enumeration member 18368
ooStretchBitmaps enumeration member 18368
OPConvert 18121
OPConvert namespace 18121
Interfaces 18122
Structs, Records, Enums 18132
Types 18133
OPConvert.IOPConvert 18122
OPConvert.IOPConvert.Attachments 18130
OPConvert.IOPConvert.Encoding 18130
OPConvert.IOPConvert.GetAttachments 18124
OPConvert.IOPConvert.GetEncoding 18124
OPConvert.IOPConvert.GetOptions 18124
OPConvert.IOPConvert.GetTempDir 18124
OPConvert.IOPConvert.InvContextToMsg 18125
OPConvert.IOPConvert.MakeFault 18125
OPConvert.IOPConvert.MakeResponse 18126
OPConvert.IOPConvert.MsgToInvContext 18127
OPConvert.IOPConvert.Options 18131
OPConvert.IOPConvert.ProcessResponse 18128
OPConvert.IOPConvert.SetAttachments 18128
OPConvert.IOPConvert.SetEncoding 18129
OPConvert.IOPConvert.SetOptions 18129
OPConvert.IOPConvert.SetTempDir 18129
OPConvert.IOPConvert.TempDir 18131
OPConvert.TSOAPConvertOption 18132
OPConvert.TSOAPConvertOption enumeration 18132
OPConvert.TSOAPConvertOptions 18133
OPConvert.TSOAPConvertOptions type 18133
OpenArray template 22216
~OpenArray 22216
= 22223
about OpenArray template 22216
GetHigh 22223
OpenArray 22217, 22218, 22219, 22220, 22221, 22222
OpenArray constructor 22216
OpenArray Members 22216
OpenArray methods 22222
OpenArray operators 22223
T* 22223
OpenArrayCount template 22223
about OpenArrayCount template 22223
GetHigh 22230
OpenArrayCount 22225, 22226, 22227, 22228, 22229, 22230
OpenArrayCount constructor 22224
OpenArrayCount Members 22224
OpenArrayCount methods 22230
opInsert enumeration member 3275
opRemove enumeration member 3275
OPToSOAPDomConv 18135
OPToSOAPDomConv namespace 18135
Classes 18137
Constants 18186
Functions 18178
Structs, Records, Enums 18179
Types 18182
Variables 18185
OPToSOAPDomConv.ConvNodeState 18179
OPToSOAPDomConv.ConvNodeState enumeration 18179
OPToSOAPDomConv.DefArrayElemName 18185
OPToSOAPDomConv.DefArrayElemName variable 18185
OPToSOAPDomConv.ESOAPDomConvertError 18137
OPToSOAPDomConv.GetOrdPropEx 18178
OPToSOAPDomConv.GetOrdPropEx function 18178
OPToSOAPDomConv.SVarArrayType 18186
OPToSOAPDomConv.SVarArrayType constant 18186
OPToSOAPDomConv.TMemberDataNotReceivedEvent 18182
OPToSOAPDomConv.TMemberDataNotReceivedEvent type 18182
OPToSOAPDomConv.TMultiRefNodeElem 18179
OPToSOAPDomConv.TMultiRefNodeElem record 18179
OPToSOAPDomConv.TMultiRefNodeMap 18182
OPToSOAPDomConv.TMultiRefNodeMap type 18182
OPToSOAPDomConv.TMultiRefNodeMapElem 18180
OPToSOAPDomConv.TMultiRefNodeMapElem record 18180
OPToSOAPDomConv.TMultiRefNodes 18183

OPToSOAPDomConv.TMultiRefNodes type 18183
OPToSOAPDomConv.TOPToSoapDomConvert 18137
OPToSOAPDomConv.TOPToSoapDomConvert.Attachments 18148
OPToSOAPDomConv.TOPToSoapDomConvert.Create 18143
OPToSOAPDomConv.TOPToSoapDomConvert.Destroy 18143
OPToSOAPDomConv.TOPToSoapDomConvert.Encoding 18149
OPToSOAPDomConv.TOPToSoapDomConvert.InvContextToMsg 18144
OPToSOAPDomConv.TOPToSoapDomConvert.MakeFault 18145
OPToSOAPDomConv.TOPToSoapDomConvert.MakeResponse 18145
OPToSOAPDomConv.TOPToSoapDomConvert.MsgToInvContext 18146
OPToSOAPDomConv.TOPToSoapDomConvert.NewXMLDocument 18147
OPToSOAPDomConv.TOPToSoapDomConvert.ProcessResponse 18147
OPToSOAPDomConv.TOPToSoapDomConvert.TempDir 18149
OPToSOAPDomConv.TOPToSoapDomConvert.WSDLView 18150
OPToSOAPDomConv.TSOAPArrayDesc 18183
OPToSOAPDomConv.TSOAPArrayDesc type 18183
OPToSOAPDomConv.TSOAPArrayElemDesc 18180
OPToSOAPDomConv.TSOAPArrayElemDesc record 18180
OPToSOAPDomConv.TSOAPDomConv 18150
OPToSOAPDomConv.TSOAPDomConv.AddAttachment 18157
OPToSOAPDomConv.TSOAPDomConv.AddMultiRefNode 18158
OPToSOAPDomConv.TSOAPDomConv.AddNamespaceDecl 18158
OPToSOAPDomConv.TSOAPDomConv.AddObjectAsWriting 18158
OPToSOAPDomConv.TSOAPDomConv.CheckEncodingStyle 18158
OPToSOAPDomConv.TSOAPDomConv.ChildNodesAreNull 18159
OPToSOAPDomConv.TSOAPDomConv.ConvertByteArrayToSoap 18159
OPToSOAPDomConv.TSOAPDomConv.ConvertEnumToSoap 18159
OPToSOAPDomConv.TSOAPDomConv.ConvertNativeArrayToSoap 18159
OPToSOAPDomConv.TSOAPDomConv.ConvertNativeDataToSoap 18160
OPToSOAPDomConv.TSOAPDomConv.ConvertObjectToSOAP 18160
OPToSOAPDomConv.TSOAPDomConv.ConvertSoapToEnum 18161
OPToSOAPDomConv.TSOAPDomConv.ConvertSoapToNativeArray 18161
OPToSOAPDomConv.TSOAPDomConv.ConvertSoapToNativeArrayElem 18161
OPToSOAPDomConv.TSOAPDomConv.ConvertSoapToNativeData 18161
OPToSOAPDomConv.TSOAPDomConv.ConvertSOAPToObject 18162
OPToSOAPDomConv.TSOAPDomConv.ConvertSoapToVariant 18162
OPToSOAPDomConv.TSOAPDomConv.ConvertVariantToSoap 18162
OPToSOAPDomConv.TSOAPDomConv.Create 18155
OPToSOAPDomConv.TSOAPDomConv.CreateMultiRefNode 18163
OPToSOAPDomConv.TSOAPDomConv.CreateNULLNode 18163
OPToSOAPDomConv.TSOAPDomConv.CreateObjectNode 18163
OPToSOAPDomConv.TSOAPDomConv.CreateScalarNodeXS 18163
OPToSOAPDomConv.TSOAPDomConv.CreateTypedNode 18164
OPToSOAPDomConv.TSOAPDomConv.Destroy 18156
OPToSOAPDomConv.TSOAPDomConv.FinalizeMultiRefNodes 18164
OPToSOAPDomConv.TSOAPDomConv.FindAttachment 18164
OPToSOAPDomConv.TSOAPDomConv.FindMultiRefNodeByID 18164

OPToSOAPDomConv.TSOAPDomConv.FindMultiRefNodeByInstance 18165
OPToSOAPDomConv.TSOAPDomConv.FindNodeByHREF 18165
OPToSOAPDomConv.TSOAPDomConv.FindPrefixForURI 18165
OPToSOAPDomConv.TSOAPDomConv.GetDataNode 18165
OPToSOAPDomConv.TSOAPDomConv.GetElementType 18166
OPToSOAPDomConv.TSOAPDomConv.GetNewID 18166
OPToSOAPDomConv.TSOAPDomConv.GetNodeAsText 18166
OPToSOAPDomConv.TSOAPDomConv.GetObjectPropAsText 18166
OPToSOAPDomConv.TSOAPDomConv.GetOptions 18166
OPToSOAPDomConv.TSOAPDomConv.GetTypeBySchemaNS 18167
OPToSOAPDomConv.TSOAPDomConv.InitObjectFromSOAP 18167
OPToSOAPDomConv.TSOAPDomConv.IsNodeAVarArray 18167
OPToSOAPDomConv.TSOAPDomConv.IsObjectWriting 18168
OPToSOAPDomConv.TSOAPDomConv.LoadObject 18168
OPToSOAPDomConv.TSOAPDomConv.MakeArrayNode 18168, 18169
OPToSOAPDomConv.TSOAPDomConv.MultiRefObject 18169
OPToSOAPDomConv.TSOAPDomConv.NodeIsNULL 18169
OPToSOAPDomConv.TSOAPDomConv.ObjectMemberNoShow 18169
OPToSOAPDomConv.TSOAPDomConv.ObjInstanceToSOAP 18169
OPToSOAPDomConv.TSOAPDomConv.OnMemberDataNotReceived 18177
OPToSOAPDomConv.TSOAPDomConv.OnUnhandledNode 18177
OPToSOAPDomConv.TSOAPDomConv.Options 18176
OPToSOAPDomConv.TSOAPDomConv.ReadHeader 18170
OPToSOAPDomConv.TSOAPDomConv.ReadRectDynArray 18170
OPToSOAPDomConv.TSOAPDomConv.ReadRectDynArrayElem 18171
OPToSOAPDomConv.TSOAPDomConv.ReadRow 18171
OPToSOAPDomConv.TSOAPDomConv.ReadVarArrayDim 18171
OPToSOAPDomConv.TSOAPDomConv.ReadVariant 18171
OPToSOAPDomConv.TSOAPDomConv.RemoveObjectAsWriting 18172
OPToSOAPDomConv.TSOAPDomConv.ResetMultiRef 18172
OPToSOAPDomConv.TSOAPDomConv.SerializationOptions 18172, 18173
OPToSOAPDomConv.TSOAPDomConv.SetNodeType 18173
OPToSOAPDomConv.TSOAPDomConv.SetObjectPropFromText 18173
OPToSOAPDomConv.TSOAPDomConv.SetOptions 18173
OPToSOAPDomConv.TSOAPDomConv.UnhandledNode 18174
OPToSOAPDomConv.TSOAPDomConv.WriteHeader 18174
OPToSOAPDomConv.TSOAPDomConv.WriteNonRectDynArray 18174
OPToSOAPDomConv.TSOAPDomConv.WriteNonRectDynArrayElem 18174
OPToSOAPDomConv.TSOAPDomConv.WriteRectDynArray 18175
OPToSOAPDomConv.TSOAPDomConv.WriteRectDynArrayElem 18175
OPToSOAPDomConv.TSOAPDomConv.WriteVarArray 18175
OPToSOAPDomConv.TSOAPDomConv.WriteVarArrayAsB64 18175
OPToSOAPDomConv.TSOAPDomConv.WriteVariant 18176
OPToSOAPDomConv.TUnhandledNodeEvent 18183
OPToSOAPDomConv.TUnhandledNodeEvent type 18183
OPToSOAPDomConv.TXMLNodeArray 18184
OPToSOAPDomConv.TXMLNodeArray type 18184
OrbPas 18187
OrbPas namespace 18187
Functions 18217
Interfaces 18189
Structs, Records, Enums 18222
Types 18224
OrbPas.AnyToVariant 18217
OrbPas.AnyToVariant function 18217
OrbPas.BindStub 18217
OrbPas.BindStub function 18217

OrbPas.CorbaAnyType 18218
OrbPas.CorbaAnyType function 18218
OrbPas.CorbaBoolean 18224
OrbPas.CorbaBoolean type 18224
OrbPas.CorbaDuplicateAny 18218
OrbPas.CorbaDuplicateAny function 18218
OrbPas.CorbaReleaseAny 18218
OrbPas.CorbaReleaseAny function 18218
OrbPas.CorbaStringFree 18219
OrbPas.CorbaStringFree function 18219
OrbPas.CorbaULong 18224
OrbPas.CorbaULong type 18224
OrbPas.CorbaWStringFree 18219
OrbPas.CorbaWStringFree function 18219
OrbPas.CreateSkeleton 18219
OrbPas.CreateSkeleton function 18219
OrbPas.CreateUserException 18220
OrbPas.CreateUserException function 18220
OrbPas.IBOA 18189
OrbPas.IBOA.Deactivate 18190
OrbPas.IBOA.GetPrincipal 18190
OrbPas.IBOA.GetPrincipalLength 18190
OrbPas.IBOA.ImplIsReady 18190
OrbPas.IBOA.ObjIsReady 18191
OrbPas.ICorbaObj 18191
OrbPas.ICorbaObj.CorbaObject 18192
OrbPas.ICorbaObj.IsLocal 18192
OrbPas.IMarshalInBuffer 18192
OrbPas.IMarshalInBuffer.Buffer 18193
OrbPas.IMarshalInBuffer.GetAny 18194
OrbPas.IMarshalInBuffer.GetChar 18194
OrbPas.IMarshalInBuffer.GetDouble 18194
OrbPas.IMarshalInBuffer.GetFloat 18194
OrbPas.IMarshalInBuffer.GetLong 18194
OrbPas.IMarshalInBuffer.GetObject 18195
OrbPas.IMarshalInBuffer.GetShort 18195
OrbPas.IMarshalInBuffer.GetText 18195
OrbPas.IMarshalInBuffer.GetUnsignedChar 18195
OrbPas.IMarshalInBuffer.GetUnsignedLong 18196
OrbPas.IMarshalInBuffer.GetUnsignedShort 18196
OrbPas.IMarshalInBuffer.GetWidechar 18196
OrbPas.IMarshalInBuffer.GetWideText 18196
OrbPas.IMarshalOutBuffer 18196
OrbPas.IMarshalOutBuffer.Buffer 18198
OrbPas.IMarshalOutBuffer.PutAny 18198
OrbPas.IMarshalOutBuffer.PutChar 18198
OrbPas.IMarshalOutBuffer.PutDouble 18198
OrbPas.IMarshalOutBuffer.PutFloat 18198
OrbPas.IMarshalOutBuffer.PutLong 18199
OrbPas.IMarshalOutBuffer.PutObject 18199
OrbPas.IMarshalOutBuffer.PutShort 18199
OrbPas.IMarshalOutBuffer.PutText 18199
OrbPas.IMarshalOutBuffer.PutUnsignedChar 18200
OrbPas.IMarshalOutBuffer.PutUnsignedLong 18200
OrbPas.IMarshalOutBuffer.PutUnsignedShort 18200
OrbPas.IMarshalOutBuffer.PutWidechar 18200
OrbPas.IMarshalOutBuffer.PutWideText 18200
OrbPas.InitORB 18220
OrbPas.InitORB function 18220
OrbPas.IORB 18201
OrbPas.IORB.BOAInit 18202
OrbPas.IORB.CreateAliasTC 18202
OrbPas.IORB.CreateArrayTC 18202
OrbPas.IORB.CreateObjRefTC 18203
OrbPas.IORB.CreateSequenceTC 18203
OrbPas.IORB.CreateStructTC 18203
OrbPas.IORB.CreateTC 18203
OrbPas.IORB.DispatchStruct 18203
OrbPas.IORB.FindRepositoryTC 18204
OrbPas.IORB.MakeAny 18204
OrbPas.IORB.ObjectToString 18204
OrbPas.IORB.Orb 18204
OrbPas.IORB.Shutdown 18205
OrbPas.IORB.StringToObject 18205
OrbPas.ISkeleton 18205
OrbPas.ISkeleton.GetImplementation 18206
OrbPas.ISkeleton.GetReplyBuffer 18206
OrbPas.ISkeletonObject 18206

OrbPas.ISkeletonObject.Execute 18207
OrbPas.ISkeletonObject.GetImplementation 18207
OrbPas.ISkeletonObject.GetSkeleton 18207
OrbPas.IStub 18208
OrbPas.IStub.CreateRequest 18209
OrbPas.IStub.Dispatch 18209
OrbPas.IStub.GetInterface 18209
OrbPas.IStub.Hash 18209
OrbPas.IStub.Invoke 18210
OrbPas.IStub.IsA 18210
OrbPas.IStub.NonExistent 18210
OrbPas.IStub.RepositoryID 18210
OrbPas.IStub.SetPrincipal 18211
OrbPas.IStubObject 18211
OrbPas.IStubObject.GetStub 18212
OrbPas.ITypeCode 18212
OrbPas.ITypeCode.ContentType 18213
OrbPas.ITypeCode.DefaultIndex 18213
OrbPas.ITypeCode.DiscriminatorType 18213
OrbPas.ITypeCode.Equal 18214
OrbPas.ITypeCode.Id 18214
OrbPas.ITypeCode.Kind 18214
OrbPas.ITypeCode.Length 18214
OrbPas.ITypeCode.MemberCount 18215
OrbPas.ITypeCode.MemberLabel 18215
OrbPas.ITypeCode.MemberName 18215
OrbPas.ITypeCode.MemberType 18215
OrbPas.ITypeCode.Name 18215
OrbPas.ITypeCode.Parameter 18216
OrbPas.ITypeCode.ParameterCount 18216
OrbPas.PCorbaAny 18225
OrbPas.PCorbaAny type 18225
OrbPas.PCorbaInterfaceDef 18225
OrbPas.PCorbaInterfaceDef type 18225
OrbPas.PCorbaObject 18225
OrbPas.PCorbaObject type 18225
OrbPas.PCorbaOrb 18226
OrbPas.PCorbaOrb type 18226
OrbPas.PExceptionDescription 18226
OrbPas.PExceptionDescription type 18226
OrbPas.PMarshalInbuffer 18226
OrbPas.PMarshalInbuffer type 18226
OrbPas.PMarshalOutbuffer 18227
OrbPas.PMarshalOutbuffer type 18227
OrbPas.PUserExceptionProxy 18227
OrbPas.PUserExceptionProxy type 18227
OrbPas.RegisterUserException 18220
OrbPas.RegisterUserException function 18220
OrbPas.SequenceToVariant 18221
OrbPas.SequenceToVariant function 18221
OrbPas.TArgv 18227
OrbPas.TArgv type 18227
OrbPas.TCKind 18222
OrbPas.TCKind enumeration 18222
OrbPas.TCopyUserExceptionProc 18228
OrbPas.TCopyUserExceptionProc type 18228
OrbPas.TStructMember 18223
OrbPas.TStructMember record 18223
OrbPas.TStructMembers 18228
OrbPas.TStructMembers type 18228
OrbPas.TThrowUserExceptionProc 18228
OrbPas.TThrowUserExceptionProc type 18228
OrbPas.TUserExceptionFactoryProc 18229
OrbPas.TUserExceptionFactoryProc type 18229
OrbPas.UnRegisterUserException 18221
OrbPas.UnRegisterUserException function 18221
OrbPas.VariantToAny 18221
OrbPas.VariantToAny function 18221
osEmpty enumeration member 18013
osInPlaceActive enumeration member 18013
osLoaded enumeration member 18013
osOpen enumeration member 18013
osPictureText enumeration member 18368
osPlusMinusPictureText enumeration member 18368
osPlusMinusText enumeration member 18368
osRunning enumeration member 18013
osText enumeration member 18368
osTreePictureText enumeration member 18368

osTreeText enumeration member 18368
osUIActive enumeration member 18013
otOwnerDraw enumeration member 18369
otSByte enumeration member 23374
otSLong enumeration member 23374
otStandard enumeration member 18369
otSWord enumeration member 23374
otUByte enumeration member 23374
otULong enumeration member 23374
otUWord enumeration member 23374
Outline 18231
Outline namespace 18231
    Classes 18233
    Structs, Records, Enums 18366
    Types 18371
Outline.EOutlineChange 18371
Outline.EOutlineChange type 18371
Outline.EOutlineError 18233
Outline.OutlineError 18234
Outline.TAttachMode 18366
Outline.TAttachMode enumeration 18366
Outline.TBitmapArray 18371
Outline.TBitmapArray type 18371
Outline.TBitmapArrayRange 18372
Outline.TBitmapArrayRange type 18372
Outline.TChangeRange 18372
Outline.TChangeRange type 18372
Outline.TCustomOutline 18234
Outline.TCustomOutline.Add 18250
Outline.TCustomOutline.AddChild 18250
Outline.TCustomOutline.AddChildObject 18251
Outline.TCustomOutline.AddObject 18252
Outline.TCustomOutline.BadIndex 18253
Outline.TCustomOutline.BeginUpdate 18253
Outline.TCustomOutline.Canvas 18268
Outline.TCustomOutline.Clear 18253
Outline.TCustomOutline.Click 18254
Outline.TCustomOutline.Collapse 18254
Outline.TCustomOutline.Create 18248
Outline.TCustomOutline.DblClick 18254
Outline.TCustomOutline.DefineProperties 18255
Outline.TCustomOutline.Delete 18255
Outline.TCustomOutline.DeleteNode 18256
Outline.TCustomOutline.Destroy 18248
Outline.TCustomOutline.DrawCell 18256
Outline.TCustomOutline.EndUpdate 18257
Outline.TCustomOutline.Expand 18257
Outline.TCustomOutline.FullCollapse 18257
Outline.TCustomOutline.FullExpand 18258
Outline.TCustomOutline.GetDataItem 18258
Outline.TCustomOutline.GetItem 18259
Outline.TCustomOutline.GetNodeDisplayWidth 18259
Outline.TCustomOutline.GetTextItem 18259
Outline.TCustomOutline.GetVisibleNode 18260
Outline.TCustomOutline.Insert 18260
Outline.TCustomOutline.InsertObject 18261
Outline.TCustomOutline.ItemCount 18270
Outline.TCustomOutline.ItemHeight 18270
Outline.TCustomOutline.Items 18271
Outline.TCustomOutline.ItemSeparator 18271
Outline.TCustomOutline.KeyDown 18262
Outline.TCustomOutline.KeyPress 18262
Outline.TCustomOutline.Lines 18272
Outline.TCustomOutline.Loaded 18263
Outline.TCustomOutline.LoadFromFile 18263
Outline.TCustomOutline.LoadFromStream 18264
Outline.TCustomOutline.MouseDown 18264
Outline.TCustomOutline.Move 18265
Outline.TCustomOutline.OnCollapse 18278
Outline.TCustomOutline.OnDrawItem 18278
Outline.TCustomOutline.OnExpand 18279
Outline.TCustomOutline.Options 18272
Outline.TCustomOutline.OutlineStyle 18273
Outline.TCustomOutline.PictureClosed 18274
Outline.TCustomOutline.PictureLeaf 18274
Outline.TCustomOutline.PictureMinus 18275
Outline.TCustomOutline.PictureOpen 18275
Outline.TCustomOutline.PicturePlus 18276

Outline.TCustomOutline.Row 18276
Outline.TCustomOutline.SaveToFile 18265
Outline.TCustomOutline.SaveToStream 18265
Outline.TCustomOutline.SelectedItem 18277
Outline.TCustomOutline.SetDisplayWidth 18266
Outline.TCustomOutline.SetGoodIndex 18266
Outline.TCustomOutline.SetLevel 18267
Outline.TCustomOutline.SetUpdateState 18267
Outline.TCustomOutline.Style 18277
Outline.TOutline 18279
Outline.TOutline.Align 18294
Outline.TOutline.BorderStyle 18298
Outline.TOutline.Color 18298
Outline.TOutline.Ctl3D 18299
Outline.TOutline.DragCursor 18299
Outline.TOutline.DragKind 18299
Outline.TOutline.DragMode 18300
Outline.TOutline.Enabled 18301
Outline.TOutline.Font 18302
Outline.TOutline.ItemHeight 18303
Outline.TOutline.ItemSeparator 18303
Outline.TOutline.Lines 18304
Outline.TOutline.OnClick 18320
Outline.TOutline.OnCollapse 18322
Outline.TOutline.OnContextPopup 18322
Outline.TOutline.OnDblClick 18323
Outline.TOutline.OnDragDrop 18323
Outline.TOutline.OnDragOver 18325
Outline.TOutline.OnDrawItem 18328
Outline.TOutline.OnEndDock 18328
Outline.TOutline.OnEndDrag 18329
Outline.TOutline.OnEnter 18330
Outline.TOutline.OnExit 18332
Outline.TOutline.OnExpand 18334
Outline.TOutline.OnKeyDown 18334
Outline.TOutline.OnKeyPress 18337
Outline.TOutline.OnKeyUp 18338
Outline.TOutline.OnMouseActivate 18339
Outline.TOutline.OnMouseDown 18340
Outline.TOutline.OnMouseEnter 18343
Outline.TOutline.OnMouseLeave 18343
Outline.TOutline.OnMouseMove 18343
Outline.TOutline.OnMouseUp 18346
Outline.TOutline.OnStartDock 18348
Outline.TOutline.OnStartDrag 18348
Outline.TOutline.Options 18304
Outline.TOutline.OutlineStyle 18305
Outline.TOutline.ParentColor 18306
Outline.TOutline.ParentCtl3D 18306
Outline.TOutline.ParentFont 18307
Outline.TOutline.ParentShowHint 18307
Outline.TOutline.PictureClosed 18308
Outline.TOutline.PictureLeaf 18308
Outline.TOutline.PictureMinus 18308
Outline.TOutline.PictureOpen 18309
Outline.TOutline.PicturePlus 18309
Outline.TOutline.PopupMenu 18310
Outline.TOutline.ScrollBars 18314
Outline.TOutline.ShowHint 18315
Outline.TOutline.Style 18316
Outline.TOutline.TabOrder 18317
Outline.TOutline.TabStop 18317
Outline.TOutline.Visible 18318
Outline.TOutlineBitmap 18367
Outline.TOutlineBitmap enumeration 18367
Outline.TOutlineBitmaps 18372
Outline.TOutlineBitmaps type 18372
Outline.TOutlineNode 18349
Outline.TOutlineNode.AddNode 18353
Outline.TOutlineNode.ChangeLevelBy 18353
Outline.TOutlineNode.Collapse 18354
Outline.TOutlineNode.Create 18351
Outline.TOutlineNode.Data 18360
Outline.TOutlineNode.Destroy 18352
Outline.TOutlineNode.Expand 18354
Outline.TOutlineNode.ExpandCount 18361
Outline.TOutlineNode.Expanded 18361
Outline.TOutlineNode.FullExpand 18355

Outline.TOutlineNode.FullPath 18361
Outline.TOutlineNode.GetDataItem 18355
Outline.TOutlineNode.GetDisplayWidth 18355
Outline.TOutlineNode.getFirstChild 18356
Outline.TOutlineNode.GetLastChild 18356
Outline.TOutlineNode.GetNextChild 18356
Outline.TOutlineNode.GetNodeAtIndex 18357
Outline.TOutlineNode.GetPrevChild 18357
Outline.TOutlineNode.GetRowOfNode 18357
Outline.TOutlineNode.GetTextItem 18358
Outline.TOutlineNode.GetVisibleNode 18358
Outline.TOutlineNode.HasAsParent 18358
Outline.TOutlineNode.HasItems 18362
Outline.TOutlineNode.Index 18362
Outline.TOutlineNode.InsertNode 18358
Outline.TOutlineNode.InternalRemove 18359
Outline.TOutlineNode.IsVisible 18363
Outline.TOutlineNode.Items 18363
Outline.TOutlineNode.Level 18363
Outline.TOutlineNode.List 18364
Outline.TOutlineNode.MoveTo 18359
Outline.TOutlineNode.Outline 18364
Outline.TOutlineNode.Parent 18364
Outline.TOutlineNode.Remove 18359
Outline.TOutlineNode.Text 18364
Outline.TOutlineNode.TopItem 18365
Outline.TOutlineNode.WriteNode 18359
Outline.TOutlineNodeCompare 18367
Outline.TOutlineNodeCompare enumeration 18367
Outline.TOutlineOption 18368
Outline.TOutlineOption enumeration 18368
Outline.TOutlineOptions 18373
Outline.TOutlineOptions type 18373
Outline.TOutlineStyle 18368
Outline.TOutlineStyle enumeration 18368
Outline.TOutlineType 18369
Outline.TOutlineType enumeration 18369
OutlineError class 18234
    about OutlineError class 18234

P

paAutoUpdate enumeration member 11236
paCenter enumeration member 16201
paCustomDropDown enumeration member 11236
paDialog enumeration member 11236
paDisplayReadOnly enumeration member 11236
paFullWidthName enumeration member 11236
PagItems 18375
PagItems namespace 18375
    Classes 18376
PagItems.TBasePageItemsProducer 18376
PagItems.TBasePageItemsProducer.Create 18380
PagItems.TBasePageItemsProducer.Destroy 18381
PagItems.TBasePageItemsProducer.GetChildren 18381
PagItems.TBasePageItemsProducer.GetComponentList 18382
PagItems.TBasePageItemsProducer.GetDefaultComponentList 18382
PagItems.TBasePageItemsProducer.SetChildOrder 18382
PagItems.TBasePageItemsProducer.WebPageItems 18382
PagItems.TPageItemsProducer 18383
PagItems.TPageItemsProducer.Create 18388
PagItems.TPageItemsProducer.Destroy 18388
PagItems.TPageItemsProducer.GetTemplateStream 18388
PagItems.TPageItemsProducer.HTMLDoc 18389
PagItems.TPageItemsProducer.HTMLFile 18389
paLeft enumeration member 16201
paLong enumeration member 1302
paMultiSelect enumeration member 11236
paNotNestable enumeration member 11236
paNullable enumeration member 1302
paReadOnly enumeration member 11236
paRevertable enumeration member 11236
paRight enumeration member 16201
paSigned enumeration member 1302
paSortList enumeration member 11236
paSubProperties enumeration member 11236
paValueEditable enumeration member 11236
paValueList enumeration member 11236

paVCL enumeration member 11236
paVolatileSubProperties enumeration member 11236
pbByName enumeration member 10254
pbByNumber enumeration member 10254
pbHorizontal enumeration member 5462
pbLowered enumeration member 5466
pbNone enumeration member 5466
pbRaised enumeration member 5466
pbsError enumeration member 11565
pbsNormal enumeration member 11565
pbsPaused enumeration member 11565
pbVertical enumeration member 5462
pcCollation enumeration member 18416
pcCopies enumeration member 18416
pcktAttrArea enumeration member 11712
pcOrientation enumeration member 18416
pdDown enumeration member 15424
pdInput enumeration member 1303
pdInputOutput enumeration member 1303
pdLeft enumeration member 15424
pdOutput enumeration member 1303
pdReturnValue enumeration member 1303
pdRight enumeration member 15424
pdUnknown enumeration member 1303
pdUp enumeration member 15424
pf15bit enumeration member 14640
pf16bit enumeration member 14640
pf1bit enumeration member 14640
pf24bit enumeration member 14640
pf32bit enumeration member 14640
pf4bit enumeration member 14640
pf8bit enumeration member 14640
pfAddress enumeration member 23375
pfADTG enumeration member 1303
pfAppleTalk enumeration member 20136
pfArray enumeration member 23375
pfBan enumeration member 20136
pfCcitt enumeration member 20136
pfChaos enumeration member 20136
pfConst enumeration member 23375
pfCustom enumeration member 14640
pfDataKit enumeration member 20136
pfDecNet enumeration member 20136
pfDevice enumeration member 14640
pfDli enumeration member 20136
pfEcma enumeration member 20136
pfFireFox enumeration member 20136
pfHidden enumeration member 7620
pfHylink enumeration member 20136
pfImpLink enumeration member 20136
pfInet enumeration member 20136
pfInKey enumeration member 7620
pfInUpdate enumeration member 7620
pfInWhere enumeration member 7620
pfIpx enumeration member 20136
pfIso enumeration member 20136
pfLat enumeration member 20136
pfMax enumeration member 20136
pfNs enumeration member 20136
pfOsi enumeration member 20136
pfOut enumeration member 23375
pfPup enumeration member 20136
pfReference enumeration member 23375
pfSna enumeration member 20136
pfUnix enumeration member 20136
pfUnknown1 enumeration member 20136
pfUnspec enumeration member 20136
pfVar enumeration member 23375
pfVoiceView enumeration member 20136
pfXML enumeration member 1303
pkDotMatrix enumeration member 11564
pkEvents enumeration member 11237
pkHPPCL enumeration member 11564
pkProperties enumeration member 11237
pmAuto enumeration member 14362
pmBlack enumeration member 14637
pmCopy enumeration member 14637
pmDefault enumeration member 11561

pmDouble enumeration member 15982
pmExplicit enumeration member 14362
pmExtended enumeration member 15982
pmInches enumeration member 11561
pmMask enumeration member 14637
pmMaskNotPen enumeration member 14637
pmMaskPenNot enumeration member 14637
pmMerge enumeration member 14637
pmMergeNotPen enumeration member 14637
pmMergePenNot enumeration member 14637
pmMillimeters enumeration member 11561
pmNone enumeration member 14362
pmNop enumeration member 14637
pmNot enumeration member 14637
pmNotCopy enumeration member 14637
pmNotMask enumeration member 14637
pmNotMerge enumeration member 14637
pmNotXor enumeration member 14637
pmReserved enumeration member 15982
pmSingle enumeration member 15982
pmWhite enumeration member 14637
pmXor enumeration member 14637
poAllowCommandText enumeration member 18618
poAllowMultiRecordUpdates enumeration member 18618
poAsyncLoad enumeration member 26460
poAutoRefresh enumeration member 18618
poCascadeDeletes enumeration member 18618
poCascadeUpdates enumeration member 18618
poCurrentPage enumeration member 8112
poDefault enumeration member 14363, 26045
poDefaultPosOnly enumeration member 14363
poDefaultSizeOnly enumeration member 14363
poDesigned enumeration member 14363
poDesktopCenter enumeration member 14363
poDisableDeletes enumeration member 18618
poDisableEdits enumeration member 18618
poDisableInserts enumeration member 18618
poDisablePrintToFile enumeration member 11563
poFetchBlobsOnDemand enumeration member 18618
poFetchDetailsOnDemand enumeration member 18618
poHelp enumeration member 11563
poHorizontal enumeration member 15424
poIncFieldProps enumeration member 18618
poLandscape enumeration member 18416
poLocateParams enumeration member 8112
poMainFormCenter enumeration member 14363
poNone enumeration member 14364
poNoReset enumeration member 18618
poOwnerFormCenter enumeration member 14363
poPageNums enumeration member 11563
poPortrait enumeration member 18416
poPreserveWhiteSpace enumeration member 26460
poPrintToFile enumeration member 11563
poPrintToFit enumeration member 14364
poPropogateChanges enumeration member 18618
poProportional enumeration member 14364
poPublishLocationAsSecure enumeration member 26045
poReadOnly enumeration member 18618
poResolveExternals enumeration member 26460
poRetainServerOrder enumeration member 18618
poScreenCenter enumeration member 14363
poSelection enumeration member 11563
poUseQuoteChar enumeration member 18618
poValidateOnParse enumeration member 26460
poVertical enumeration member 15424
poWarning enumeration member 11563
prAllPages enumeration member 11564
Printers 18391
Printers namespace 18391
    Classes 18392
    Functions 18412
    Structs, Records, Enums 18416
    Types 18418
Printers.AssignPrn 18412
Printers.AssignPrn function 18412
Printers.EPrinter 18392
Printers.Printer 18412
Printers.Printer function 18412

Printers.SetPrinter 18414
Printers.SetPrinter function 18414
Printers.TPrinter 18392
Printers.TPrinter.Abort 18395
Printers.TPrinter.Aborted 18403
Printers.TPrinter.BeginDoc 18397
Printers.TPrinter.Canvas 18404
Printers.TPrinter.Capabilities 18404
Printers.TPrinter.Copies 18405
Printers.TPrinter.Create 18394
Printers.TPrinter.Destroy 18395
Printers.TPrinter.EndDoc 18399
Printers.TPrinter.Fonts 18405
Printers.TPrinter.GetPrinter 18400
Printers.TPrinter.Handle 18405
Printers.TPrinter.NewPage 18400
Printers.TPrinter.Orientation 18406
Printers.TPrinter.PageHeight 18407
Printers.TPrinter.PageNumber 18408
Printers.TPrinter.PageWidth 18408
Printers.TPrinter.PrinterIndex 18409
Printers.TPrinter.Printers 18409
Printers.TPrinter.Printing 18409
Printers.TPrinter.Refresh 18401
Printers.TPrinter.SetPrinter 18402
Printers.TPrinter.Title 18411
Printers.TPrinterCapabilities 18418
Printers.TPrinterCapabilities type 18418
Printers.TPrinterCapability 18416
Printers.TPrinterCapability enumeration 18416
Printers.TPrinterOrientation 18416
Printers.TPrinterOrientation enumeration 18416
Printers.TPrinterState 18417
Printers.TPrinterState enumeration 18417
Procedure class 22230
about Procedure class 22230
GetCallType 22232
Procedure 22232
Procedure constructor 22232
Procedure Members 22231
Procedure methods 22232
RequestResult 22233
ProcedureParametersColumns class 2175
about ProcedureParametersColumns class 2175
CatalogName 2177
DbxDataType 2177
IsFixedLength 2177
IsLong 2177
IsNullable 2178
IsUnicode 2178
IsUnsigned 2178
Ordinal 2178
ParameterMode 2178
ParameterName 2179
Precision 2179
ProcedureName 2179
ProcedureParametersColumns fields 2176
ProcedureParametersColumns Members 2176
Scale 2179
SchemaName 2179
TypeName 2180
ProceduresColumns class 2180
about ProceduresColumns class 2180
CatalogName 2181
ProcedureName 2181
ProceduresColumns fields 2181
ProceduresColumns Members 2181
ProcedureType 2181
SchemaName 2182
ProcedureSourcesColumns class 2182
about ProcedureSourcesColumns class 2182
CatalogName 2183
Definition 2183
ExternalDefinition 2184
ProcedureName 2184
ProcedureSourcesColumns fields 2183
ProcedureSourcesColumns Members 2183
ProcedureType 2184

SchemaName 2184
PropertyGet class 22233
about PropertyGet class 22233
GetCallType 22235
PropertyGet 22235
PropertyGet constructor 22235
PropertyGet Members 22234
PropertyGet methods 22235
RequestResult 22236
PropertySet class 22236
about PropertySet class 22236
GetCallType 22238
PropertySet 22238
PropertySet constructor 22238
PropertySet Members 22237
PropertySet methods 22238
RequestResult 22239
ProtocolValue class 2074
about ProtocolValue class 2074
InProcess 2075
ProtocolValue fields 2075
ProtocolValue Members 2075
Tcp 2075
Provider 18419
Provider namespace 18419
Classes 18421
Constants 18629
Functions 18615
Interfaces 18613
Structs, Records, Enums 18617
Types 18622
Variables 18628
Provider.EDSWriter 18421
Provider.EDSWriter.Create 18423
Provider.EDSWriter.ErrorCode 18423
Provider.GetObjectProperty 18615
Provider.GetObjectProperty function 18615
Provider.GetStringProperty 18615
Provider.GetStringProperty function 18615
Provider.InformixLob 18628
Provider.InformixLob variable 18628
Provider.IProviderContainer 18613
Provider.IProviderContainer.RegisterProvider 18614
Provider.IProviderContainer.UnRegisterProvider 18614
Provider.MetaDataOption 18629
Provider.MetaDataOption constant 18629
Provider.PPutFieldInfo 18622
Provider.PPutFieldInfo type 18622
Provider.ResetOption 18630
Provider.ResetOption constant 18630
Provider.TAfterUpdateRecordEvent 18622
Provider.TAfterUpdateRecordEvent type 18622
Provider.TBaseProvider 18423
Provider.TBaseProvider.AfterUpdateRecord 18442
Provider.TBaseProvider.BeforeUpdateRecord 18442
Provider.TBaseProvider.CheckResolver 18433
Provider.TBaseProvider.Create 18429
Provider.TBaseProvider.CreateDataPacket 18433
Provider.TBaseProvider.CreateResolver 18434
Provider.TBaseProvider.Destroy 18430
Provider.TBaseProvider.DoAfterUpdateRecord 18434
Provider.TBaseProvider.DoBeforeCommit 18435
Provider.TBaseProvider.DoBeforeUpdateRecord 18435
Provider.TBaseProvider.DoOnGetData 18435
Provider.TBaseProvider.DoOnUpdateData 18435
Provider.TBaseProvider.FAfterUpdateRecord 18430
Provider.TBaseProvider.FBeforeCommit 18430
Provider.TBaseProvider.FBeforeUpdateRecord 18431
Provider.TBaseProvider.FDataDS 18431
Provider.TBaseProvider.FetchDetails 18436
Provider.TBaseProvider.FOnGetData 18431
Provider.TBaseProvider.FOnUpdateData 18431
Provider.TBaseProvider.FOnUpdateError 18432
Provider.TBaseProvider.FProviderOptions 18432
Provider.TBaseProvider.FreeResolver 18436
Provider.TBaseProvider.FResolver 18432
Provider.TBaseProvider.FUpdateMode 18432
Provider.TBaseProvider.InternalApplyUpdates 18436

Provider.TBaseProvider.InternalGetRecords 18437
Provider.TBaseProvider.InternalRowRequest 18438
Provider.TBaseProvider.LocateRecord 18439
Provider.TBaseProvider.OnDataRequest 18442
Provider.TBaseProvider.OnGetData 18443
Provider.TBaseProvider.OnUpdateData 18443
Provider.TBaseProvider.OnUpdateError 18444
Provider.TBaseProvider.Options 18440
Provider.TBaseProvider.PacketDataSet 18440
Provider.TBaseProvider.Resolver 18441
Provider.TBaseProvider.UpdateMode 18441
Provider.TBaseProvider.UpdateRecord 18439
Provider.TBeforeCommitEvent 18623
Provider.TBeforeCommitEvent type 18623
Provider.TBeforeUpdateRecordEvent 18623
Provider.TBeforeUpdateRecordEvent type 18623
Provider.TCustomPacketWriter 18445
Provider.TCustomPacketWriter.AddAttribute 18448
Provider.TCustomPacketWriter.Check 18448
Provider.TCustomPacketWriter.Create 18446
Provider.TCustomPacketWriter.Destroy 18446
Provider.TCustomPacketWriter.DSWriter 18449
Provider.TCustomPacketWriter.FBuffer 18447
Provider.TCustomPacketWriter.FIDSWriter 18447
Provider.TCustomPacketWriter.SetDSWriter 18448
Provider.TCustomProvider 18449
Provider.TCustomProvider.ActiveUpdateException 18474
Provider.TCustomProvider.AfterApplyUpdates 18475
Provider.TCustomProvider.AfterExecute 18476
Provider.TCustomProvider.AfterGetParams 18477
Provider.TCustomProvider.AfterGetRecords 18477
Provider.TCustomProvider.AfterRowRequest 18478
Provider.TCustomProvider.ApplyUpdates 18459, 18460
Provider.TCustomProvider.BeforeApplyUpdates 18479
Provider.TCustomProvider.BeforeExecute 18479
Provider.TCustomProvider.BeforeGetParams 18480
Provider.TCustomProvider.BeforeGetRecords 18480
Provider.TCustomProvider.BeforeRowRequest 18482
Provider.TCustomProvider.Create 18454
Provider.TCustomProvider.Data 18474
Provider.TCustomProvider.DataRequest 18460
Provider.TCustomProvider.Destroy 18454
Provider.TCustomProvider.DoAfterApplyUpdates 18460
Provider.TCustomProvider.DoAfterExecute 18461
Provider.TCustomProvider.DoAfterGetParams 18461
Provider.TCustomProvider.DoAfterGetRecords 18462
Provider.TCustomProvider.DoAfterRowRequest 18462
Provider.TCustomProvider.DoApplyUpdates 18463
Provider.TCustomProvider.DoBeforeApplyUpdates 18463
Provider.TCustomProvider.DoBeforeExecute 18464
Provider.TCustomProvider.DoBeforeGetParams 18464
Provider.TCustomProvider.DoBeforeGetRecords 18465
Provider.TCustomProvider.DoBeforeRowRequest 18465
Provider.TCustomProvider.DoGetRecords 18466
Provider.TCustomProvider.Execute 18466
Provider.TCustomProvider.Exported 18474
Provider.TCustomProvider.FActiveUpdateAbortException 18455
Provider.TCustomProvider.FAfterApplyUpdates 18455
Provider.TCustomProvider.FAfterExecute 18456
Provider.TCustomProvider.FAfterGetParams 18456
Provider.TCustomProvider.FAfterGetRecords 18456
Provider.TCustomProvider.FAfterRowRequest 18456
Provider.TCustomProvider.FBeforeApplyUpdates 18456
Provider.TCustomProvider.FBeforeExecute 18457
Provider.TCustomProvider.FBeforeGetParams 18457
Provider.TCustomProvider.FBeforeGetRecords 18457
Provider.TCustomProvider.FBeforeRowRequest 18457
Provider.TCustomProvider.FExported 18458
Provider.TCustomProvider.FOnDataRequest 18458
Provider.TCustomProvider.FOnValidate 18458
Provider.TCustomProvider.FOwnerData 18458
Provider.TCustomProvider.GetData 18467
Provider.TCustomProvider.GetParams 18467
Provider.TCustomProvider.GetRecords 18468
Provider.TCustomProvider.InternalApplyUpdates 18469
Provider.TCustomProvider.InternalExecute 18470
Provider.TCustomProvider.InternalGetParams 18471
Provider.TCustomProvider.InternalGetRecords 18471

Provider.TCustomProvider.InternalRowRequest 18472
Provider.TCustomProvider.OnDataRequest 18482
Provider.TCustomProvider.OnValidate 18483
Provider.TCustomProvider.OwnerData 18475
Provider.TCustomProvider.RowRequest 18472
Provider.TCustomProvider.SetActiveUpdateException 18473
Provider.TCustomResolver 18483
Provider.TCustomResolver.ApplyUpdates 18490
Provider.TCustomResolver.BeginUpdate 18490
Provider.TCustomResolver.Create 18487
Provider.TCustomResolver.Destroy 18487
Provider.TCustomResolver.DoDelete 18491
Provider.TCustomResolver.DoInsert 18491
Provider.TCustomResolver.DoUpdate 18491
Provider.TCustomResolver.EndUpdate 18492
Provider.TCustomResolver.FErrorCount 18488
Provider.TCustomResolver.FMaxErrors 18488
Provider.TCustomResolver.FPrevResponse 18488
Provider.TCustomResolver.FProvider 18489
Provider.TCustomResolver.FreeTreeData 18492
Provider.TCustomResolver.FUpdateTree 18489
Provider.TCustomResolver.HandleUpdateError 18493
Provider.TCustomResolver.InitializeConflictBuffer 18493
Provider.TCustomResolver.InitKeyFields 18494
Provider.TCustomResolver.InitTreeData 18494
Provider.TCustomResolver.InternalBeforeResolve 18494
Provider.TCustomResolver.InternalUpdateRecord 18495
Provider.TCustomResolver.LogUpdateError 18495
Provider.TCustomResolver.LogUpdateRecord 18495
Provider.TCustomResolver.Provider 18496
Provider.TCustomResolver.RowRequest 18496
Provider.TCustomResolver.UpdateTree 18497
Provider.TDataPacketWriter 18497
Provider.TDataPacketWriter.AddColumn 18502
Provider.TDataPacketWriter.AddConstraints 18502
Provider.TDataPacketWriter.AddDataSetAttributes 18502
Provider.TDataPacketWriter.AddExtraFieldProps 18502
Provider.TDataPacketWriter.AddFieldLinks 18503
Provider.TDataPacketWriter.AddIndexDefs 18503
Provider.TDataPacketWriter.Constraints 18507
Provider.TDataPacketWriter.Destroy 18499
Provider.TDataPacketWriter.FConstraints 18500
Provider.TDataPacketWriter.FOnGetParams 18500
Provider.TDataPacketWriter.FOptions 18500
Provider.TDataPacketWriter.FPacketOptions 18501
Provider.TDataPacketWriter.FPutFieldInfo 18501
Provider.TDataPacketWriter.FreeInfoRecords 18503
Provider.TDataPacketWriter.GetDataPacket 18503
Provider.TDataPacketWriter.GetFieldIdx 18503
Provider.TDataPacketWriter.InitPutProcs 18504
Provider.TDataPacketWriter.OnGetParams 18508
Provider.TDataPacketWriter.Options 18507
Provider.TDataPacketWriter.PacketOptions 18507
Provider.TDataPacketWriter.PutADTField 18504
Provider.TDataPacketWriter.PutArrayField 18504
Provider.TDataPacketWriter.PutBlobField 18504
Provider.TDataPacketWriter.PutCalcField 18505
Provider.TDataPacketWriter.PutDataSetField 18505
Provider.TDataPacketWriter.PutField 18505
Provider.TDataPacketWriter.PutStringField 18505
Provider.TDataPacketWriter.PutVarBytesField 18505
Provider.TDataPacketWriter.PutWideStringField 18506
Provider.TDataPacketWriter.RefreshPutProcs 18506
Provider.TDataPacketWriter.Reset 18506
Provider.TDataPacketWriter.WriteDataSet 18506
Provider.TDataPacketWriter.WriteMetaData 18507
Provider.TDataRequestEvent 18623
Provider.TDataRequestEvent type 18623
Provider.TDataSetProvider 18508
Provider.TDataSetProvider.ActiveUpdateException 18528
Provider.TDataSetProvider.AfterApplyUpdates 18533
Provider.TDataSetProvider.AfterExecute 18534
Provider.TDataSetProvider.AfterGetParams 18535
Provider.TDataSetProvider.AfterGetRecords 18535
Provider.TDataSetProvider.AfterRowRequest 18536
Provider.TDataSetProvider.AfterUpdateRecord 18536
Provider.TDataSetProvider.BeforeApplyUpdates 18537
Provider.TDataSetProvider.BeforeExecute 18538

Provider.TDataSetProvider.BeforeGetParams 18538
Provider.TDataSetProvider.BeforeGetRecords 18539
Provider.TDataSetProvider.BeforeRowRequest 18540
Provider.TDataSetProvider.BeforeUpdateRecord 18541
Provider.TDataSetProvider.CheckDataSet 18518
Provider.TDataSetProvider.Constraints 18528
Provider.TDataSetProvider.Create 18514
Provider.TDataSetProvider.CreateDataPacket 18518
Provider.TDataSetProvider.CreateResolver 18519
Provider.TDataSetProvider.DataSet 18529
Provider.TDataSetProvider.Destroy 18514
Provider.TDataSetProvider.DoBeforeExecute 18519
Provider.TDataSetProvider.DoBeforeGetRecords 18520
Provider.TDataSetProvider.DoGetProviderAttributes 18521
Provider.TDataSetProvider.DoGetTableName 18521
Provider.TDataSetProvider.Exported 18530
Provider.TDataSetProvider.FConstraints 18515
Provider.TDataSetProvider.FDataSet 18515
Provider.TDataSetProvider.FDataSetOpened 18516
Provider.TDataSetProvider.FDSWriter 18516
Provider.TDataSetProvider.FetchDetails 18521
Provider.TDataSetProvider.FGetDSProps 18516
Provider.TDataSetProvider.FGetTableName 18516
Provider.TDataSetProvider.FindRecord 18522
Provider.TDataSetProvider.FParams 18516
Provider.TDataSetProvider.FRecordsSent 18517
Provider.TDataSetProvider.FResolveToDataSet 18517
Provider.TDataSetProvider.FTransactionStarted 18517
Provider.TDataSetProvider.GetDataSetFromDelta 18522
Provider.TDataSetProvider.InternalApplyUpdates 18522
Provider.TDataSetProvider.InternalExecute 18523
Provider.TDataSetProvider.InternalGetParams 18524
Provider.TDataSetProvider.InternalGetRecords 18524
Provider.TDataSetProvider.InternalRowRequest 18525
Provider.TDataSetProvider.LocateRecord 18525
Provider.TDataSetProvider.Notification 18526
Provider.TDataSetProvider.OnDataRequest 18541
Provider.TDataSetProvider.OnGetData 18541
Provider.TDataSetProvider.OnGetDataSetProperties 18542
Provider.TDataSetProvider.OnGetTableName 18542
Provider.TDataSetProvider.OnUpdateData 18543
Provider.TDataSetProvider.OnUpdateError 18543
Provider.TDataSetProvider.OnValidate 18544
Provider.TDataSetProvider.Options 18530
Provider.TDataSetProvider.OwnerData 18531
Provider.TDataSetProvider.Params 18531
Provider.TDataSetProvider.Reset 18526
Provider.TDataSetProvider.ResolveToDataSet 18531
Provider.TDataSetProvider.SetCommandText 18526
Provider.TDataSetProvider.SetDataSet 18527
Provider.TDataSetProvider.SetParams 18527
Provider.TDataSetProvider.SetResolveToDataSet 18527
Provider.TDataSetProvider.TransactionStarted 18532
Provider.TDataSetProvider.UpdateMode 18532
Provider.TDataSetProvider.UpdateRecord 18527
Provider.TDataSetResolver 18544
Provider.TDataSetResolver.BeginUpdate 18549
Provider.TDataSetResolver.Create 18548
Provider.TDataSetResolver.DoDelete 18550
Provider.TDataSetResolver.DoInsert 18550
Provider.TDataSetResolver.DoUpdate 18551
Provider.TDataSetResolver.EndUpdate 18551
Provider.TDataSetResolver.FBookmark 18548
Provider.TDataSetResolver.FOpened 18549
Provider.TDataSetResolver.GetProvider 18551
Provider.TDataSetResolver.InitializeConflictBuffer 18552
Provider.TDataSetResolver.InternalBeforeResolve 18552
Provider.TDataSetResolver.Provider 18553
Provider.TDataSetResolver.PutRecord 18552
Provider.TGetDSProps 18624
Provider.TGetDSProps type 18624
Provider.TGetParamsEvent 18624
Provider.TGetParamsEvent type 18624
Provider.TGetRecordOption 18617
Provider.TGetRecordOption enumeration 18617
Provider.TGetRecordOptions 18624
Provider.TGetRecordOptions type 18624
Provider.TGetTableNameEvent 18625

Provider.TGetTableNameEvent type 18625
Provider.TInfoArray 18625
Provider.TInfoArray type 18625
Provider.TLocalAppServer 18553
Provider.TLocalAppServer.AS_ApplyUpdates 18557
Provider.TLocalAppServer.AS_DataRequest 18558
Provider.TLocalAppServer.AS_Execute 18559
Provider.TLocalAppServer.AS_GetParams 18560
Provider.TLocalAppServer.AS_GetProviderNames 18560
Provider.TLocalAppServer.AS_GetRecords 18561
Provider.TLocalAppServer.AS_RowRequest 18562
Provider.TLocalAppServer.Create 18555, 18556
Provider.TLocalAppServer.Destroy 18556
Provider.TLocalAppServer.FProvider 18557
Provider.TLocalAppServer.FProviderCreated 18557
Provider.TLocalAppServer.GetIDsOfNames 18563
Provider.TLocalAppServer.GetTypeInfo 18563
Provider.TLocalAppServer.GetTypeInfoCount 18563
Provider.TLocalAppServer.InterfaceSupportsErrorInfo 18563
Provider.TLocalAppServer.Invoke 18564
Provider.TLocalAppServer.SafeCallException 18564
Provider.TOnValidate 18625
Provider.TOnValidate type 18625
Provider.TPacketDataSet 18564
Provider.TPacketDataSet.AssignCurValues 18583
Provider.TPacketDataSet.Create 18580
Provider.TPacketDataSet.CreateFromDelta 18583
Provider.TPacketDataSet.DataEvent 18583
Provider.TPacketDataSet.FCurRecBuf 18581
Provider.TPacketDataSet.FCurValues 18581
Provider.TPacketDataSet.FNewValuesModified 18581
Provider.TPacketDataSet.FOldRecBuf 18581
Provider.TPacketDataSet.FUseCurValues 18581
Provider.TPacketDataSet.FWritingCurValues 18582
Provider.TPacketDataSet.GetStateFieldValue 18584
Provider.TPacketDataSet.GetStreamMetaData 18584
Provider.TPacketDataSet.HasCurValues 18584
Provider.TPacketDataSet.HasMergeConflicts 18585
Provider.TPacketDataSet.InitAltRecBuffers 18585
Provider.TPacketDataSet.InternalClose 18585
Provider.TPacketDataSet.InternalInitRecord 18586
Provider.TPacketDataSet.InternalOpen 18586
Provider.TPacketDataSet.NewValuesModified 18588
Provider.TPacketDataSet.SetFieldData 18586
Provider.TPacketDataSet.SetStateFieldValue 18586
Provider.TPacketDataSet.SetStreamMetaData 18587
Provider.TPacketDataSet.SetWritingCurValues 18587
Provider.TPacketDataSet.StreamMetaData 18588
Provider.TPacketDataSet.UpdateKind 18587
Provider.TPacketDataSet.UseCurValues 18588
Provider.TPacketDataSet.WritingCurValues 18589
Provider.TProvider 18589
Provider.TProviderDataEvent 18626
Provider.TProviderDataEvent type 18626
Provider.TProviderOption 18618
Provider.TProviderOption enumeration 18618
Provider.TProviderOptions 18626
Provider.TProviderOptions type 18626
Provider.TPutFieldInfo 18620
Provider.TPutFieldInfo record 18620
Provider.TPutFieldProc 18626
Provider.TPutFieldProc type 18626
Provider.TResolverErrorEvent 18627
Provider.TResolverErrorEvent type 18627
Provider.TResolverResponse 18620
Provider.TResolverResponse enumeration 18620
Provider.TSQLResolver 18589
Provider.TSQLResolver.Create 18593
Provider.TSQLResolver.Destroy 18594
Provider.TSQLResolver.DoDelete 18595
Provider.TSQLResolver.DoExecSQL 18596
Provider.TSQLResolver.DoGetValues 18596
Provider.TSQLResolver.DoInsert 18596
Provider.TSQLResolver.DoUpdate 18596
Provider.TSQLResolver.FParams 18594
Provider.TSQLResolver.FreeTreeData 18597
Provider.TSQLResolver.FSQL 18594
Provider.TSQLResolver.GenDeleteSQL 18597

Provider.TSQLResolver.GenInsertSQL 18597
Provider.TSQLResolver.GenSelectSQL 18598
Provider.TSQLResolver.GenUpdateSQL 18598
Provider.TSQLResolver.GenWhereSQL 18598
Provider.TSQLResolver.GetProvider 18598
Provider.TSQLResolver.InitializeConflictBuffer 18599
Provider.TSQLResolver.InitTreeData 18599
Provider.TSQLResolver.InternalDoUpdate 18599
Provider.TSQLResolver.Provider 18600
Provider.TSQLResolver.UseFieldInUpdate 18599
Provider.TSQLResolver.UseFieldInWhere 18600
Provider.TUpdateTree 18600
Provider.TUpdateTree.Clear 18606
Provider.TUpdateTree.Create 18603
Provider.TUpdateTree.Data 18610
Provider.TUpdateTree.Delta 18610
Provider.TUpdateTree.Destroy 18603
Provider.TUpdateTree.DetailCount 18610
Provider.TUpdateTree.DetailList 18610
Provider.TUpdateTree.Details 18610
Provider.TUpdateTree.DoUpdates 18606
Provider.TUpdateTree.ErrorDS 18611
Provider.TUpdateTree.FData 18604
Provider.TUpdateTree.FDeltaDS 18604
Provider.TUpdateTree.FDetails 18604
Provider.TUpdateTree.FErrorDS 18604
Provider.TUpdateTree.FName 18604
Provider.TUpdateTree.FOpened 18605
Provider.TUpdateTree.FParent 18605
Provider.TUpdateTree.FResolver 18605
Provider.TUpdateTree.FSourceDS 18605
Provider.TUpdateTree.GetDetail 18606
Provider.TUpdateTree.GetDetailCount 18607
Provider.TUpdateTree.GetErrorDS 18607
Provider.TUpdateTree.GetHasErrors 18607
Provider.TUpdateTree.GetIsNested 18607
Provider.TUpdateTree.GetTree 18608
Provider.TUpdateTree.HasErrors 18611
Provider.TUpdateTree.InitData 18608
Provider.TUpdateTree.InitDelta 18608
Provider.TUpdateTree.InitErrorPacket 18609
Provider.TUpdateTree.IsNested 18611
Provider.TUpdateTree.Name 18611
Provider.TUpdateTree.Opened 18612
Provider.TUpdateTree.Parent 18612
Provider.TUpdateTree.RefreshData 18609
Provider.TUpdateTree.Resolver 18612
Provider.TUpdateTree.Source 18612
Provider.VarArrayFromStrings 18615
Provider.VarArrayFromStrings function 18615
Provider.VarArrayFromWideStrings 18616
Provider.VarArrayFromWideStrings function 18616
Provider.XMLOption 18630
Provider.XMLOption constant 18630
Provider.XMLUTF8Option 18631
Provider.XMLUTF8Option constant 18631
prPageNums enumeration member 11564
prSelection enumeration member 11564
psAlternate enumeration member 14639
psClear enumeration member 14639
psDash enumeration member 14639
psDashDot enumeration member 14639
psDashDotDot enumeration member 14639
psDot enumeration member 14639
psEnding enumeration member 14641
psHandleDC enumeration member 18417
psHandleIC enumeration member 18417
psInsideFrame enumeration member 14639
psLeftPivot enumeration member 17675
psMiddlePivot enumeration member 17675
psNoHandle enumeration member 18417
psNone enumeration member 17675
psoDefaultMinMargins enumeration member 11562
psoDisableMargins enumeration member 11562
psoDisableOrientation enumeration member 11562
psoDisablePagePainting enumeration member 11562
psoDisablePaper enumeration member 11562
psoDisablePrinter enumeration member 11562

psoMargins enumeration member 11562
psoMinMargins enumeration member 11562
psoNoNetworkButton enumeration member 11562
psoShowHelp enumeration member 11562
psoWarning enumeration member 11562
psOwnerDraw enumeration member 5466
psRightPivot enumeration member 17675
psRunning enumeration member 14641
psSolid enumeration member 14639
psStarting enumeration member 14641
psText enumeration member 5466
psTopPivot enumeration member 17675
psUserStyle enumeration member 14639
ptBottom enumeration member 5462
ptEndOfPeriod enumeration member 15984
ptEnvelope enumeration member 11562
ptInput enumeration member 7619
ptInputOutput enumeration member 7619
ptLeft enumeration member 5462
ptNone enumeration member 5462
ptOutput enumeration member 7619
ptPaper enumeration member 11562
ptResult enumeration member 7619
ptRight enumeration member 5462
ptStartOfPeriod enumeration member 15984
ptTop enumeration member 5462
ptUnknown enumeration member 7619

Q

qaNoModuleName enumeration member 19792
qaNoModulesVar enumeration member 19792
qaUseModulesVar enumeration member 19792

R

raAbort enumeration member 8539
raCancel enumeration member 8539
raCorrect enumeration member 8539
raMerge enumeration member 8539
raRefresh enumeration member 8539
raSkip enumeration member 8539
rcNextClosed enumeration member 17513
rcNextOpen enumeration member 17513
rcPrevClosed enumeration member 17513
rcPrevOpen enumeration member 17513
rdBinary enumeration member 18691
rdExpandString enumeration member 18691
rdInteger enumeration member 18691
rdString enumeration member 18691
rdUnknown enumeration member 18691
reAccessViolation enumeration member 22567
reAssertionFailed enumeration member 22567
reControlBreak enumeration member 22567
reDivByZero enumeration member 22567
reExternalException enumeration member 22567
Registry 18633
Registry namespace 18633
    Classes 18634
    Structs, Records, Enums 18691
Registry.ERegistryException 18634
Registry.TRegDataInfo 18691
Registry.TRegDataInfo record 18691
Registry.TRegDataType 18691
Registry.TRegDataType enumeration 18691
Registry.TRegIniFile 18635
Registry.TRegIniFile.Create 18638
Registry.TRegIniFile.DeleteKey 18639
Registry.TRegIniFile.EraseSection 18639
Registry.TRegIniFile.FileName 18644
Registry.TRegIniFile.ReadBool 18640
Registry.TRegIniFile.ReadInteger 18640
Registry.TRegIniFile.ReadSection 18641
Registry.TRegIniFile.ReadSections 18641
Registry.TRegIniFile.ReadSectionValues 18642
Registry.TRegIniFile.ReadString 18642
Registry.TRegIniFile.WriteBool 18643
Registry.TRegIniFile.WriteInteger 18643
Registry.TRegIniFile.WriteString 18644
Registry.TRegistry 18645

Registry.TRegistry.Access 18672
Registry.TRegistry.ChangeKey 18650
Registry.TRegistry.CloseKey 18650
Registry.TRegistry.Create 18648
Registry.TRegistry.CreateKey 18651
Registry.TRegistry.CurrentKey 18673
Registry.TRegistry.CurrentPath 18673
Registry.TRegistry.DeleteKey 18651
Registry.TRegistry.DeleteValue 18652
Registry.TRegistry.Destroy 18648
Registry.TRegistry.GetBaseKey 18653
Registry.TRegistry.GetData 18653
Registry.TRegistry.GetDataInfo 18653
Registry.TRegistry.GetDataSize 18654
Registry.TRegistry.GetDataType 18654
Registry.TRegistry.GetKey 18655
Registry.TRegistry.GetKeyInfo 18655
Registry.TRegistry.GetKeyNames 18655
Registry.TRegistry.GetValueNames 18656
Registry.TRegistry.HasSubKeys 18656
Registry.TRegistry.KeyExists 18656
Registry.TRegistry.LazyWrite 18674
Registry.TRegistry.LoadKey 18657
Registry.TRegistry.MoveKey 18657
Registry.TRegistry.OpenKey 18658
Registry.TRegistry.OpenKeyReadOnly 18658
Registry.TRegistry.PutData 18659
Registry.TRegistry.ReadBinaryData 18659
Registry.TRegistry.ReadBool 18660
Registry.TRegistry.ReadCurrency 18660
Registry.TRegistry.ReadDate 18661
Registry.TRegistry.ReadDateTime 18661
Registry.TRegistry.ReadFloat 18662
Registry.TRegistry.ReadInteger 18662
Registry.TRegistry.ReadString 18662
Registry.TRegistry.ReadTime 18663
Registry.TRegistry.RegistryConnect 18663
Registry.TRegistry.RenameValue 18664
Registry.TRegistry.ReplaceKey 18664
Registry.TRegistry.RestoreKey 18665
Registry.TRegistry.RootKey 18674
Registry.TRegistry.SaveKey 18666
Registry.TRegistry.SetCurrentKey 18666
Registry.TRegistry.UnLoadKey 18666
Registry.TRegistry.ValueExists 18667
Registry.TRegistry.WriteBinaryData 18667
Registry.TRegistry.WriteBool 18668
Registry.TRegistry.WriteCurrency 18668
Registry.TRegistry.WriteDate 18669
Registry.TRegistry.WriteDateTime 18669
Registry.TRegistry.WriteExpandString 18670
Registry.TRegistry.WriteFloat 18670
Registry.TRegistry.WriteInteger 18671
Registry.TRegistry.WriteString 18671
Registry.TRegistry.WriteTime 18671
Registry.TRegistryIniFile 18675
Registry.TRegistryIniFile.Create 18677
Registry.TRegistryIniFile.DeleteKey 18679
Registry.TRegistryIniFile.Destroy 18678
Registry.TRegistryIniFile.EraseSection 18679
Registry.TRegistryIniFile.ReadBinaryStream 18679
Registry.TRegistryIniFile.ReadDate 18680
Registry.TRegistryIniFile.ReadDateTime 18681
Registry.TRegistryIniFile.ReadFloat 18681
Registry.TRegistryIniFile.ReadInteger 18682
Registry.TRegistryIniFile.ReadSection 18682
Registry.TRegistryIniFile.ReadSections 18683
Registry.TRegistryIniFile.ReadSectionValues 18683
Registry.TRegistryIniFile.ReadString 18684
Registry.TRegistryIniFile.ReadTime 18685
Registry.TRegistryIniFile.RegIniFile 18689
Registry.TRegistryIniFile.UpdateFile 18685
Registry.TRegistryIniFile.WriteBinaryStream 18685
Registry.TRegistryIniFile.WriteDate 18686
Registry.TRegistryIniFile.WriteDateTime 18687
Registry.TRegistryIniFile.WriteFloat 18687
Registry.TRegistryIniFile.WriteInteger 18688
Registry.TRegistryIniFile.WriteString 18688

Registry.TRegistryIniFile.WriteTime 18689
Registry.TRegKeyInfo 18692
Registry.TRegKeyInfo record 18692
reIntfCastError enumeration member 22567
reIntOverflow enumeration member 22567
reInvalidCast enumeration member 22567
reInvalidOp enumeration member 22567
reInvalidPtr enumeration member 22567
reNone enumeration member 22567
reOutOfMemory enumeration member 22567
reOverflow enumeration member 22567
rePrivInstruction enumeration member 22567
ReqFiles 18695
ReqFiles namespace 18695
Classes 18696
ReqFiles.TWebRequestFile 18696
ReqFiles.TWebRequestFile.Create 18698
ReqFiles.TWebRequestFile.Destroy 18698
ReqFiles.TWebRequestFile.GetContentType 18699
ReqFiles.TWebRequestFile.GetFieldName 18699
ReqFiles.TWebRequestFile.GetFileName 18699
ReqFiles.TWebRequestFile.GetStream 18700
ReqFiles.TWebRequestFiles 18700
ReqFiles.TWebRequestFiles.Add 18703
ReqFiles.TWebRequestFiles.Create 18702
ReqFiles.TWebRequestFiles.Destroy 18702
ReqFiles.TWebRequestFiles.GetCount 18704
ReqFiles.TWebRequestFiles.GetItem 18704
ReqFiles.TWebRequestFileStream 18704
ReqFiles.TWebRequestFileStream.Create 18706
ReqFiles.TWebRequestFileStream.Write 18706
ReqMulti 18709
ReqMulti namespace 18709
Classes 18710
ReqMulti.TMultipartContentParser 18710
ReqMulti.TMultipartContentParser.BufferContent 18712
ReqMulti.TMultipartContentParser.CanParse 18713
ReqMulti.TMultipartContentParser.Destroy 18712
ReqMulti.TMultipartContentParser.ExtractContentTypeFields 18713
ReqMulti.TMultipartContentParser.ExtractMultipartContentFields 18713
ReqMulti.TMultipartContentParser.GetContentFields 18714
ReqMulti.TMultipartContentParser.GetFiles 18714
ReqMulti.TMultipartContentParser.ParseMultiPart 18715
ReqMulti.TMultipartContentParser.ParseMultiPartContent 18715
ReqMulti.TMultipartContentParser.ParseMultipartHeaders 18715
reRangeError enumeration member 22567
reSafeCallError enumeration member 22567
ReservedWordsColumns class 2184
about ReservedWordsColumns class 2184
ReservedWord 2185
ReservedWordsColumns fields 2185
ReservedWordsColumns Members 2185
reStackOverflow enumeration member 22567
reUnderflow enumeration member 22567
reVarArrayBounds enumeration member 22567
reVarArrayCreate enumeration member 22567
reVarDispatch enumeration member 22567
reVarInvalidOp enumeration member 22567
reVarNotArray enumeration member 22567
reVarTypeCast enumeration member 22567
reZeroDivide enumeration member 22567
Rio 18717
Rio namespace 18717
Classes 18718
Constants 18736
Interfaces 18733
Types 18735
Rio.IRIOAccess 18733
Rio.IRIOAccess.GetRIO 18734
Rio.IRIOAccess.RIO 18734
Rio.StubSize 18736
Rio.StubSize constant 18736
Rio.TAfterExecuteEvent 18735
Rio.TAfterExecuteEvent type 18735
Rio.TBeforeExecuteEvent 18735
Rio.TBeforeExecuteEvent type 18735

Rio.TRIO 18718
Rio.TRIO.AfterConstruction 18724
Rio.TRIO.BeforeDestruction 18725
Rio.TRIO.Converter 18728
Rio.TRIO.Create 18722
Rio.TRIO.Destroy 18722
Rio.TRIO.DoAfterExecute 18725
Rio.TRIO.DoBeforeExecute 18725
Rio.TRIO.FConverter 18723
Rio.TRIO.FIID 18723
Rio.TRIO.FWebNode 18723
Rio.TRIO.GetResponseStream 18726
Rio.TRIO.IntfMD 18723
Rio.TRIO.NewInstance 18726
Rio.TRIO.OnAfterExecute 18730
Rio.TRIO.OnBeforeExecute 18730
Rio.TRIO.OnGetAttachment 18731
Rio.TRIO.OnSendAttachment 18731
Rio.TRIO.QueryInterface 18727
Rio.TRIO.RefCount 18728
Rio.TRIO.SOAPHeaders 18728
Rio.TRIO.WebNode 18729
rmDown enumeration member 15983
rmNearest enumeration member 15983
rmTruncate enumeration member 15983
rmUp enumeration member 15983
rnActivateModule enumeration member 25460
rnCreateModule enumeration member 25460
rnDbase enumeration member 10255
rnDeactivateModule enumeration member 25460
rnFinishRequest enumeration member 25460
rnFreeModule enumeration member 25460
rnNotSupported enumeration member 10255
rnParadox enumeration member 10255
rnStartRequest enumeration member 25460
roExcludeID enumeration member 19792
RolesColumns class 2185
    about RolesColumns class 2185
    RoleName 2186
    RolesColumns fields 2186
    RolesColumns Members 2186
roRedirectHTTPGet enumeration member 969
roRedirectHTTPPost enumeration member 969
rrAbort enumeration member 18620
rrApply enumeration member 18620
rrIgnore enumeration member 18620
rrMerge enumeration member 18620
rrSkip enumeration member 18620
rsCanceled enumeration member 1304
rsCantRelease enumeration member 1304
rsConcurrencyViolation enumeration member 1304
rsDBDeleted enumeration member 1304
rsDeleted enumeration member 1304
rsIntegrityViolation enumeration member 1304
rsInvalid enumeration member 1304
rsLine enumeration member 13270
rsMaxChangesExceeded enumeration member 1304
rsModified enumeration member 1304
rsMultipleChanges enumeration member 1304
rsNew enumeration member 1304
rsNone enumeration member 13270
rsObjectOpen enumeration member 1304
rsOK enumeration member 1304
rsOutOfMemory enumeration member 1304
rsPattern enumeration member 13270
rsPendingChanges enumeration member 1304
rsPermissionDenied enumeration member 1304
rsSchemaViolation enumeration member 1304
rsUnmodified enumeration member 1304
rsUpdate enumeration member 13270
rtBitmap enumeration member 15396
rtCursor enumeration member 15396
rtIcon enumeration member 15396
rwComponentPage enumeration member 970
rwPage enumeration member 970
rwUndefined enumeration member 970
rwURL enumeration member 970

S

saIgnore enumeration member 14366
saMaximize enumeration member 14366
saMinimize enumeration member 14366
saRestore enumeration member 14366
sbFirst enumeration member 5460
sbHorizontal enumeration member 14365
sbLast enumeration member 5460
sbLeft enumeration member 23230
sbRight enumeration member 23230
sbsNone enumeration member 21951
sbsSingle enumeration member 21951
sbsSunken enumeration member 21951
sbVertical enumeration member 14365
scBottom enumeration member 21949
scEndScroll enumeration member 21949
ScheduleColumns class 2076
- about ScheduleColumns class 2076
- Day 2077
- JavaMethod 2077
- Millis 2077
- Period 2077
- ScheduleColumns fields 2076
- ScheduleColumns Members 2076
- Time 2078

SchedulePeriod class 2078
- about SchedulePeriod class 2078
- Day 2079
- Millis 2079
- SchedulePeriod fields 2078
- SchedulePeriod Members 2078
- Week 2079

SchemasColumns class 2187
- about SchemasColumns class 2187
- CatalogName 2188
- SchemaName 2188
- SchemasColumns fields 2187
- SchemasColumns Members 2187

scHide enumeration member 12096
ScktComp 18737
ScktComp namespace 18737
- Classes 18739
- Constants 18893
- Functions 18883
- Structs, Records, Enums 18884
- Types 18889

ScktComp.CM_DEFERFREE 18893
ScktComp.CM_DEFERFREE constant 18893
ScktComp.CM_LOOKUPCOMPLETE 18893
ScktComp.CM_LOOKUPCOMPLETE constant 18893
ScktComp.CM_SOCKETMESSAGE 18893
ScktComp.CM_SOCKETMESSAGE constant 18893
ScktComp.ESocketError 18739
ScktComp.SetErrorProc 18883
ScktComp.SetErrorProc function 18883
ScktComp.TAbstractSocket 18740
ScktComp.TAbstractSocket.Active 18750
ScktComp.TAbstractSocket.Address 18750
ScktComp.TAbstractSocket.Close 18744
ScktComp.TAbstractSocket.DoActivate 18744
ScktComp.TAbstractSocket.Error 18745
ScktComp.TAbstractSocket.Event 18746
ScktComp.TAbstractSocket.Host 18751
ScktComp.TAbstractSocket.InitSocket 18746
ScktComp.TAbstractSocket.Loaded 18747
ScktComp.TAbstractSocket.Open 18747
ScktComp.TAbstractSocket.Port 18751
ScktComp.TAbstractSocket.Service 18752
ScktComp.TAbstractSocket.SetActive 18748
ScktComp.TAbstractSocket.SetAddress 18748
ScktComp.TAbstractSocket.SetHost 18748
ScktComp.TAbstractSocket.SetPort 18749
ScktComp.TAbstractSocket.SetService 18749
ScktComp.TAsyncStyle 18884
ScktComp.TAsyncStyle enumeration 18884
ScktComp.TAsyncStyles 18889
ScktComp.TAsyncStyles type 18889

ScktComp.TClientSocket 18752
ScktComp.TClientSocket.Active 18759
ScktComp.TClientSocket.Address 18759
ScktComp.TClientSocket.ClientType 18760
ScktComp.TClientSocket.Create 18756
ScktComp.TClientSocket.Destroy 18757
ScktComp.TClientSocket.DoActivate 18757
ScktComp.TClientSocket.GetClientType 18758
ScktComp.TClientSocket.Host 18760
ScktComp.TClientSocket.OnConnect 18762
ScktComp.TClientSocket.OnConnecting 18763
ScktComp.TClientSocket.OnDisconnect 18764
ScktComp.TClientSocket.OnError 18764
ScktComp.TClientSocket.OnLookup 18764
ScktComp.TClientSocket.OnRead 18765
ScktComp.TClientSocket.OnWrite 18765
ScktComp.TClientSocket.Port 18761
ScktComp.TClientSocket.Service 18761
ScktComp.TClientSocket.SetClientType 18758
ScktComp.TClientSocket.Socket 18762
ScktComp.TClientType 18885
ScktComp.TClientType enumeration 18885
ScktComp.TClientWinSocket 18766
ScktComp.TClientWinSocket.ClientType 18769
ScktComp.TClientWinSocket.Connect 18768
ScktComp.TClientWinSocket.SetClientType 18769
ScktComp.TCMLookupComplete 18885
ScktComp.TCMLookupComplete record 18885
ScktComp.TCMSocketMessage 18886
ScktComp.TCMSocketMessage record 18886
ScktComp.TCustomServerSocket 18770
ScktComp.TCustomServerSocket.Destroy 18774
ScktComp.TCustomServerSocket.DoActivate 18775
ScktComp.TCustomServerSocket.FServerSocket 18775
ScktComp.TCustomServerSocket.GetGetSocketEvent 18776
ScktComp.TCustomServerSocket.GetGetThreadEvent 18776
ScktComp.TCustomServerSocket.GetOnClientError 18776
ScktComp.TCustomServerSocket.GetOnClientEvent 18777
ScktComp.TCustomServerSocket.GetOnThreadEnd 18777
ScktComp.TCustomServerSocket.GetOnThreadStart 18778
ScktComp.TCustomServerSocket.GetServerType 18778
ScktComp.TCustomServerSocket.GetThreadCacheSize 18778
ScktComp.TCustomServerSocket.OnClientConnect 18784
ScktComp.TCustomServerSocket.OnClientDisconnect 18785
ScktComp.TCustomServerSocket.OnClientError 18785
ScktComp.TCustomServerSocket.OnClientRead 18786
ScktComp.TCustomServerSocket.OnClientWrite 18786
ScktComp.TCustomServerSocket.OnGetSocket 18787
ScktComp.TCustomServerSocket.OnGetThread 18787
ScktComp.TCustomServerSocket.OnThreadEnd 18788
ScktComp.TCustomServerSocket.OnThreadStart 18788
ScktComp.TCustomServerSocket.ServerType 18782
ScktComp.TCustomServerSocket.SetGetSocketEvent 18779
ScktComp.TCustomServerSocket.SetGetThreadEvent 18779
ScktComp.TCustomServerSocket.SetOnClientError 18779
ScktComp.TCustomServerSocket.SetOnClientEvent 18780
ScktComp.TCustomServerSocket.SetOnThreadEnd 18780
ScktComp.TCustomServerSocket.SetOnThreadStart 18781
ScktComp.TCustomServerSocket.SetServerType 18781
ScktComp.TCustomServerSocket.SetThreadCacheSize 18781
ScktComp.TCustomServerSocket.ThreadCacheSize 18783
ScktComp.TCustomSocket 18789
ScktComp.TCustomSocket.Error 18793
ScktComp.TCustomSocket.Event 18794
ScktComp.TCustomSocket.OnAccept 18795
ScktComp.TCustomSocket.OnConnect 18795
ScktComp.TCustomSocket.OnConnecting 18796
ScktComp.TCustomSocket.OnDisconnect 18796
ScktComp.TCustomSocket.OnError 18797
ScktComp.TCustomSocket.OnListen 18797
ScktComp.TCustomSocket.OnLookup 18797
ScktComp.TCustomSocket.OnRead 18798
ScktComp.TCustomSocket.OnWrite 18799
ScktComp.TCustomWinSocket 18799
ScktComp.TCustomWinSocket.Accept 18803
ScktComp.TCustomWinSocket.Addr 18817
ScktComp.TCustomWinSocket.AsyncInitSocket 18803
ScktComp.TCustomWinSocket.ASyncStyles 18817

ScktComp.TCustomWinSocket.Close 18804
ScktComp.TCustomWinSocket.Connect 18805
ScktComp.TCustomWinSocket.Connected 18818
ScktComp.TCustomWinSocket.Create 18801
ScktComp.TCustomWinSocket.Data 18818
ScktComp.TCustomWinSocket.DefaultHandler 18805
ScktComp.TCustomWinSocket.Destroy 18802
ScktComp.TCustomWinSocket.Disconnect 18805
ScktComp.TCustomWinSocket.DoListen 18806
ScktComp.TCustomWinSocket.DoOpen 18806
ScktComp.TCustomWinSocket.Error 18806
ScktComp.TCustomWinSocket.Event 18807
ScktComp.TCustomWinSocket.Handle 18819
ScktComp.TCustomWinSocket.InitSocket 18807
ScktComp.TCustomWinSocket.Listen 18808
ScktComp.TCustomWinSocket.LocalAddress 18819
ScktComp.TCustomWinSocket.LocalHost 18820
ScktComp.TCustomWinSocket.LocalPort 18820
ScktComp.TCustomWinSocket.Lock 18809
ScktComp.TCustomWinSocket.LookupName 18809
ScktComp.TCustomWinSocket.LookupService 18810
ScktComp.TCustomWinSocket.LookupState 18821
ScktComp.TCustomWinSocket.OnErrorEvent 18824
ScktComp.TCustomWinSocket.OnSocketEvent 18824
ScktComp.TCustomWinSocket.Open 18810
ScktComp.TCustomWinSocket.Read 18811
ScktComp.TCustomWinSocket.ReceiveBuf 18812
ScktComp.TCustomWinSocket.ReceiveLength 18812
ScktComp.TCustomWinSocket.ReceiveText 18812
ScktComp.TCustomWinSocket.RemoteAddr 18821
ScktComp.TCustomWinSocket.RemoteAddress 18822
ScktComp.TCustomWinSocket.RemoteHost 18822
ScktComp.TCustomWinSocket.RemotePort 18823
ScktComp.TCustomWinSocket.SendBuf 18813
ScktComp.TCustomWinSocket.SendStream 18814
ScktComp.TCustomWinSocket.SendStreamThenDrop 18814
ScktComp.TCustomWinSocket.SendText 18814
ScktComp.TCustomWinSocket.SetAsyncStyles 18815
ScktComp.TCustomWinSocket.SocketHandle 18823
ScktComp.TCustomWinSocket.Unlock 18815
ScktComp.TCustomWinSocket.Write 18816
ScktComp.TErrorEvent 18886
ScktComp.TErrorEvent enumeration 18886
ScktComp.TGetSocketEvent 18889
ScktComp.TGetSocketEvent type 18889
ScktComp.TGetThreadEvent 18890
ScktComp.TGetThreadEvent type 18890
ScktComp.TLookupState 18887
ScktComp.TLookupState enumeration 18887
ScktComp.TServerAcceptThread 18825
ScktComp.TServerAcceptThread.Create 18826
ScktComp.TServerAcceptThread.Execute 18827
ScktComp.TServerAcceptThread.ServerSocket 18831
ScktComp.TServerClientThread 18831
ScktComp.TServerClientThread.ClientExecute 18834
ScktComp.TServerClientThread.ClientSocket 18838
ScktComp.TServerClientThread.Create 18833
ScktComp.TServerClientThread.Data 18838
ScktComp.TServerClientThread.Destroy 18833
ScktComp.TServerClientThread.DoTerminate 18835
ScktComp.TServerClientThread.EndConnect 18835
ScktComp.TServerClientThread.Error 18836
ScktComp.TServerClientThread.Event 18836
ScktComp.TServerClientThread.Execute 18836
ScktComp.TServerClientThread.HandleException 18837
ScktComp.TServerClientThread.KeepInCache 18839
ScktComp.TServerClientThread.ReActivate 18837
ScktComp.TServerClientThread.ServerSocket 18839
ScktComp.TServerClientThread.StartConnect 18837
ScktComp.TServerClientWinSocket 18840
ScktComp.TServerClientWinSocket.Create 18842
ScktComp.TServerClientWinSocket.Destroy 18842
ScktComp.TServerClientWinSocket.ServerWinSocket 18843
ScktComp.TServerSocket 18843
ScktComp.TServerSocket.Active 18849
ScktComp.TServerSocket.Create 18848
ScktComp.TServerSocket.OnAccept 18852
ScktComp.TServerSocket.OnClientConnect 18852

wwww

ScktComp.TServerSocket.OnClientDisconnect 18853
ScktComp.TServerSocket.OnClientError 18854
ScktComp.TServerSocket.OnClientRead 18855
ScktComp.TServerSocket.OnClientWrite 18855
ScktComp.TServerSocket.OnGetSocket 18856
ScktComp.TServerSocket.OnGetThread 18856
ScktComp.TServerSocket.OnListen 18857
ScktComp.TServerSocket.OnThreadEnd 18857
ScktComp.TServerSocket.OnThreadStart 18858
ScktComp.TServerSocket.Port 18849
ScktComp.TServerSocket.ServerType 18850
ScktComp.TServerSocket.Service 18850
ScktComp.TServerSocket.Socket 18851
ScktComp.TServerSocket.ThreadCacheSize 18851
ScktComp.TServerType 18887
ScktComp.TServerType enumeration 18887
ScktComp.TServerWinSocket 18858
ScktComp.TServerWinSocket.Accept 18862
ScktComp.TServerWinSocket.ActiveConnections 18867
ScktComp.TServerWinSocket.ActiveThreads 18867
ScktComp.TServerWinSocket.ClientConnect 18863
ScktComp.TServerWinSocket.ClientDisconnect 18863
ScktComp.TServerWinSocket.ClientErrorEvent 18863
ScktComp.TServerWinSocket.ClientRead 18863
ScktComp.TServerWinSocket.ClientWrite 18863
ScktComp.TServerWinSocket.Connections 18868
ScktComp.TServerWinSocket.Create 18861
ScktComp.TServerWinSocket.Destroy 18861
ScktComp.TServerWinSocket.Disconnect 18864
ScktComp.TServerWinSocket.DoCreateThread 18864
ScktComp.TServerWinSocket.GetClientSocket 18864
ScktComp.TServerWinSocket.GetClientThread 18865
ScktComp.TServerWinSocket.GetServerThread 18865
ScktComp.TServerWinSocket.IdleThreads 18869
ScktComp.TServerWinSocket.Listen 18865
ScktComp.TServerWinSocket.OnClientConnect 18871
ScktComp.TServerWinSocket.OnClientDisconnect 18872
ScktComp.TServerWinSocket.OnClientError 18872
ScktComp.TServerWinSocket.OnClientRead 18873
ScktComp.TServerWinSocket.OnClientWrite 18874
ScktComp.TServerWinSocket.OnGetSocket 18874
ScktComp.TServerWinSocket.OnGetThread 18875
ScktComp.TServerWinSocket.OnThreadEnd 18875
ScktComp.TServerWinSocket.OnThreadStart 18876
ScktComp.TServerWinSocket.ServerType 18869
ScktComp.TServerWinSocket.SetServerType 18866
ScktComp.TServerWinSocket.SetThreadCacheSize 18866
ScktComp.TServerWinSocket.ThreadCacheSize 18870
ScktComp.TServerWinSocket.ThreadEnd 18866
ScktComp.TServerWinSocket.ThreadStart 18867
ScktComp.TSocketErrorEvent 18890
ScktComp.TSocketErrorEvent type 18890
ScktComp.TSocketErrorProc 18891
ScktComp.TSocketErrorProc type 18891
ScktComp.TSocketEvent 18887
ScktComp.TSocketEvent enumeration 18887
ScktComp.TSocketEventEvent 18891
ScktComp.TSocketEventEvent type 18891
ScktComp.TSocketNotifyEvent 18891
ScktComp.TSocketNotifyEvent type 18891
ScktComp.TThreadNotifyEvent 18892
ScktComp.TThreadNotifyEvent type 18892
ScktComp.TWinSocketStream 18876
ScktComp.TWinSocketStream.Create 18878
ScktComp.TWinSocketStream.Destroy 18879
ScktComp.TWinSocketStream.Read 18880
ScktComp.TWinSocketStream.Seek 18880
ScktComp.TWinSocketStream.TimeOut 18882
ScktComp.TWinSocketStream.WaitForData 18881
ScktComp.TWinSocketStream.Write 18881
scLineDown enumeration member 21949
scLineUp enumeration member 21949
scMaximize enumeration member 12096
scMinimize enumeration member 12096
SConnect 18895
SConnect namespace 18895
    Classes 18897
    Constants 19005

Functions 18999
Interfaces 18983
Structs, Records, Enums 19000
Types 19001
Variables 19003
SConnect.asCreateObject 19005
SConnect.asCreateObject constant 19005
SConnect.asError 19005
SConnect.asError constant 19005
SConnect.asFreeObject 19006
SConnect.asFreeObject constant 19006
SConnect.asGetAppServers 19006
SConnect.asGetAppServers constant 19006
SConnect.asGetGUID 19006
SConnect.asGetGUID constant 19006
SConnect.asGetID 19007
SConnect.asGetID constant 19007
SConnect.asGetServers 19007
SConnect.asGetServers constant 19007
SConnect.asInvoke 19007
SConnect.asInvoke constant 19007
SConnect.asMask 19008
SConnect.asMask constant 19008
SConnect.asSoapCommand 19008
SConnect.asSoapCommand constant 19008
SConnect.CallSig 19008
SConnect.CallSig constant 19008
SConnect.EInterpreterError 18897
SConnect.ESocketConnectionError 18898
SConnect.GetPacketInterceptorList 18999
SConnect.GetPacketInterceptorList function 18999
SConnect.IDataBlock 18983
SConnect.IDataBlock.BytesReserved 18989
SConnect.IDataBlock.Clear 18984
SConnect.IDataBlock.GetBytesReserved 18985
SConnect.IDataBlock.GetMemory 18985
SConnect.IDataBlock.GetSignature 18985
SConnect.IDataBlock.GetSize 18986
SConnect.IDataBlock.GetStream 18986
SConnect.IDataBlock.IgnoreStream 18986
SConnect.IDataBlock.InitData 18987
SConnect.IDataBlock.Memory 18989
SConnect.IDataBlock.Read 18987
SConnect.IDataBlock.SetSignature 18988
SConnect.IDataBlock.SetSize 18988
SConnect.IDataBlock.Signature 18990
SConnect.IDataBlock.Size 18990
SConnect.IDataBlock.Stream 18991
SConnect.IDataBlock.Write 18988
SConnect.IDataIntercept 18991
SConnect.IDataIntercept.DataIn 18992
SConnect.IDataIntercept.DataOut 18993
SConnect.ISendDataBlock 18993
SConnect.ISendDataBlock.Send 18994
SConnect.ITransport 18995
SConnect.ITransport.Connected 18998
SConnect.ITransport.GetConnected 18996
SConnect.ITransport.GetWaitEvent 18996
SConnect.ITransport.Receive 18997
SConnect.ITransport.Send 18997
SConnect.ITransport.SetConnected 18997
SConnect.LoadWinSock2 18999
SConnect.LoadWinSock2 function 18999
SConnect.PIntArray 19001
SConnect.PIntArray type 19001
SConnect.PVariantArray 19001
SConnect.PVariantArray type 19001
SConnect.ResultSig 19009
SConnect.ResultSig constant 19009
SConnect.TCustomDataBlockInterpreter 18898
SConnect.TCustomDataBlockInterpreter.AddDispatch 18900
SConnect.TCustomDataBlockInterpreter.CallCreateObject 18900
SConnect.TCustomDataBlockInterpreter.CallFreeObject 18900
SConnect.TCustomDataBlockInterpreter.CallGetIDsOfNames 18900
SConnect.TCustomDataBlockInterpreter.CallGetServerList 18901
SConnect.TCustomDataBlockInterpreter.CallInvoke 18901

SConnect.TCustomDataBlockInterpreter.CanCreateObject 18901
SConnect.TCustomDataBlockInterpreter.CreateObject 18901
SConnect.TCustomDataBlockInterpreter.InternalCreateObject 18901
SConnect.TCustomDataBlockInterpreter.InterpretData 18902
SConnect.TCustomDataBlockInterpreter.LockObject 18902
SConnect.TCustomDataBlockInterpreter.ReleaseObject 18902
SConnect.TCustomDataBlockInterpreter.RemoveDispatch 18902
SConnect.TCustomDataBlockInterpreter.StoreObject 18903
SConnect.TCustomDataBlockInterpreter.UnlockObject 18903
SConnect.TDataBlock 18903
SConnect.TDataBlock.BytesReserved 18911
SConnect.TDataBlock.Clear 18906
SConnect.TDataBlock.Create 18905
SConnect.TDataBlock.Destroy 18905
SConnect.TDataBlock.GetBytesReserved 18907
SConnect.TDataBlock.GetMemory 18907
SConnect.TDataBlock.GetSignature 18907
SConnect.TDataBlock.GetSize 18907
SConnect.TDataBlock.GetStream 18908
SConnect.TDataBlock.IgnoreStream 18908
SConnect.TDataBlock.InitData 18908
SConnect.TDataBlock.Memory 18911
SConnect.TDataBlock.Read 18909
SConnect.TDataBlock.SetSignature 18909
SConnect.TDataBlock.SetSize 18910
SConnect.TDataBlock.Signature 18911
SConnect.TDataBlock.Size 18912
SConnect.TDataBlock.Stream 18912
SConnect.TDataBlock.Write 18910
SConnect.TDataBlockInterpreter 18913
SConnect.TDataBlockInterpreter.AddDispatch 18915
SConnect.TDataBlockInterpreter.CallCreateObject 18915
SConnect.TDataBlockInterpreter.CallFreeObject 18916
SConnect.TDataBlockInterpreter.CallGetIDsOfNames 18916
SConnect.TDataBlockInterpreter.CallGetServerList 18916
SConnect.TDataBlockInterpreter.CallInvoke 18917
SConnect.TDataBlockInterpreter.CanCreateObject 18917
SConnect.TDataBlockInterpreter.Create 18914
SConnect.TDataBlockInterpreter.CreateObject 18917
SConnect.TDataBlockInterpreter.Destroy 18914
SConnect.TDataBlockInterpreter.DoCreateObject 18917
SConnect.TDataBlockInterpreter.DoCustomAction 18917
SConnect.TDataBlockInterpreter.DoFreeObject 18918
SConnect.TDataBlockInterpreter.DoGetAppServerList 18918
SConnect.TDataBlockInterpreter.DoGetIDsOfNames 18918
SConnect.TDataBlockInterpreter.DoGetServerList 18918
SConnect.TDataBlockInterpreter.DoInvoke 18919
SConnect.TDataBlockInterpreter.InternalCreateObject 18919
SConnect.TDataBlockInterpreter.InterpretData 18919
SConnect.TDataBlockInterpreter.LockObject 18919
SConnect.TDataBlockInterpreter.ReleaseObject 18920
SConnect.TDataBlockInterpreter.RemoveDispatch 18920
SConnect.TDataBlockInterpreter.StoreObject 18920
SConnect.TDataBlockInterpreter.UnlockObject 18920
SConnect.TDataDispatch 18921
SConnect.TDataDispatch.Create 18922
SConnect.TDataDispatch.Destroy 18922
SConnect.TDataDispatch.DispatchIndex 18924
SConnect.TDataDispatch.GetIDsOfNames 18923
SConnect.TDataDispatch.GetTypeInfo 18924
SConnect.TDataDispatch.GetTypeInfoCount 18924
SConnect.TDataDispatch.Invoke 18924
SConnect.THREAD_EXCEPTION 19009
SConnect.THREAD_EXCEPTION constant 19009
SConnect.THREAD_RECEIVEDSTREAM 19009
SConnect.THREAD_RECEIVEDSTREAM constant 19009
SConnect.THREAD_REPLACETRANSPORT 19010
SConnect.THREAD_REPLACETRANSPORT constant 19010
SConnect.THREAD_SENDNOTIFY 19010
SConnect.THREAD_SENDNOTIFY constant 19010
SConnect.THREAD_SENDSTREAM 19010
SConnect.THREAD_SENDSTREAM constant 19010
SConnect.TIntArray 19002
SConnect.TIntArray type 19002
SConnect.TPacketInterceptFactory 18925
SConnect.TPacketInterceptFactory.UpdateRegistry 18927

SConnect.TSocketConnection 18928
SConnect.TSocketConnection.Address 18935
SConnect.TSocketConnection.Create 18933
SConnect.TSocketConnection.CreateTransport 18934
SConnect.TSocketConnection.DoConnect 18934
SConnect.TSocketConnection.Host 18936
SConnect.TSocketConnection.InterceptGUID 18936
SConnect.TSocketConnection.InterceptName 18937
SConnect.TSocketConnection.ObjectBroker 18937
SConnect.TSocketConnection.Port 18938
SConnect.TSocketConnection.SupportCallbacks 18939
SConnect.TSocketTransport 18939
SConnect.TSocketTransport.Address 18944
SConnect.TSocketTransport.Create 18941
SConnect.TSocketTransport.Destroy 18941
SConnect.TSocketTransport.GetConnected 18942
SConnect.TSocketTransport.GetWaitEvent 18943
SConnect.TSocketTransport.Host 18945
SConnect.TSocketTransport.InterceptGUID 18945
SConnect.TSocketTransport.Port 18945
SConnect.TSocketTransport.Receive 18943
SConnect.TSocketTransport.Send 18943
SConnect.TSocketTransport.SetConnected 18944
SConnect.TSocketTransport.Socket 18945
SConnect.TStreamedConnection 18945
SConnect.TStreamedConnection._AddRef 18952
SConnect.TStreamedConnection._Release 18952
SConnect.TStreamedConnection.Create 18950
SConnect.TStreamedConnection.CreateTransport 18952
SConnect.TStreamedConnection.Destroy 18951
SConnect.TStreamedConnection.DoConnect 18953
SConnect.TStreamedConnection.DoDisconnect 18953
SConnect.TStreamedConnection.DoError 18954
SConnect.TStreamedConnection.GetInterceptorList 18954
SConnect.TStreamedConnection.GetInterpreter 18955
SConnect.TStreamedConnection.GetServerList 18955
SConnect.TStreamedConnection.Handle 18959
SConnect.TStreamedConnection.InterceptGUID 18959
SConnect.TStreamedConnection.InterceptName 18960
SConnect.TStreamedConnection.InternalClose 18956
SConnect.TStreamedConnection.InternalOpen 18956
SConnect.TStreamedConnection.Interpreter 18960
SConnect.TStreamedConnection.QueryInterface 18956
SConnect.TStreamedConnection.Send 18957
SConnect.TStreamedConnection.SupportCallbacks 18961
SConnect.TStreamedConnection.ThreadException 18957
SConnect.TStreamedConnection.ThreadReceivedStream 18957
SConnect.TStreamedConnection.WndProc 18958
SConnect.TTransportThread 18961
SConnect.TTransportThread.Create 18963
SConnect.TTransportThread.Destroy 18963
SConnect.TTransportThread.Execute 18963
SConnect.TTransportThread.Semaphore 18967
SConnect.TVarFlag 19000
SConnect.TVarFlag enumeration 19000
SConnect.TVarFlags 19002
SConnect.TVarFlags type 19002
SConnect.TVariantArray 19002
SConnect.TVariantArray type 19002
SConnect.TWebConnection 18968
SConnect.TWebConnection.Agent 18979
SConnect.TWebConnection.Create 18973
SConnect.TWebConnection.CreateTransport 18974
SConnect.TWebConnection.Destroy 18974
SConnect.TWebConnection.DoConnect 18975
SConnect.TWebConnection.GetInterpreter 18975
SConnect.TWebConnection.GetWaitEvent 18976
SConnect.TWebConnection.ObjectBroker 18979
SConnect.TWebConnection.Password 18980
SConnect.TWebConnection.Proxy 18980
SConnect.TWebConnection.ProxyByPass 18980
SConnect.TWebConnection.Receive 18976
SConnect.TWebConnection.Send 18977
SConnect.TWebConnection.SetConnected 18977
SConnect.TWebConnection.SupportCallbacks 18981
SConnect.TWebConnection.Transport_GetConnected 18978
SConnect.TWebConnection.Transport_SetConnected 18978
SConnect.TWebConnection.URL 18981

SConnect.TWebConnection.UserName 18982
SConnect.WSACloseEvent 19003
SConnect.WSACloseEvent variable 19003
SConnect.WSACreateEvent 19003
SConnect.WSACreateEvent variable 19003
SConnect.WSAEventSelect 19004
SConnect.WSAEventSelect variable 19004
SConnect.WSAResetEvent 19004
SConnect.WSAResetEvent variable 19004
scPageDown enumeration member 21949
scPageUp enumeration member 21949
scPosition enumeration member 21949
scRestore enumeration member 12096
scShow enumeration member 12096
scShowDefault enumeration member 12096
scShowMaximized enumeration member 12096
scShowMinimized enumeration member 12096
scShowMinNoActive enumeration member 12096
scShowNA enumeration member 12096
scShowNoActivate enumeration member 12096
scShowNormal enumeration member 12096
scTop enumeration member 21949
scTrack enumeration member 21949
sdAbove enumeration member 5463
sdAll enumeration member 5463
sdAllowCreate enumeration member 13622
sdBelow enumeration member 5463
sdDown enumeration member 14757
sdLeft enumeration member 5463, 14757
sdNewFolder enumeration member 13621
sdNewUI enumeration member 13621
sdPerformCreate enumeration member 13622
sdPrompt enumeration member 13622
sdRight enumeration member 5463, 14757
sdShowEdit enumeration member 13621
sdShowFiles enumeration member 13621
sdShowShares enumeration member 13621
sdUp enumeration member 14757
sdValidateDir enumeration member 13621
seAccept enumeration member 18887
seConnect enumeration member 18887
seConnecting enumeration member 18887
seDisconnect enumeration member 18887
seListen enumeration member 18887
seLookup enumeration member 18887
seRead enumeration member 18887
SessColn 19011
SessColn namespace 19011
    Classes 19012
    Structs, Records, Enums 19035
    Types 19036
    Variables 19037
SessColn.ESessionsException 19012
SessColn.Sessions 19037
SessColn.Sessions variable 19037
SessColn.TAbstractSessionIDGen 19012
SessColn.TAbstractSessionIDGen.NextID 19013
SessColn.TAbstractSessionStatusChange 19013
SessColn.TAbstractSessionStatusChange.SessionStarted 19014
SessColn.TAbstractSessionStatusChange.SessionTerminated 19014
SessColn.TAbstractSessionStatusChange.SessionTimeout 19015
SessColn.TSessionIDGen 19015
SessColn.TSessionIDGen.Create 19015
SessColn.TSessionIDGen.NextID 19016
SessColn.TSessionItem 19016
SessColn.TSessionItem.Assign 19019
SessColn.TSessionItem.Create 19018
SessColn.TSessionItem.Destroy 19018
SessColn.TSessionItem.ExpirationTime 19020
SessColn.TSessionItem.Expired 19020
SessColn.TSessionItem.ID 19020
SessColn.TSessionItem.IsActive 19019
SessColn.TSessionItem.Items 19021
SessColn.TSessionItem.StartTime 19021
SessColn.TSessionItem.StatusFlags 19021
SessColn.TSessionItem.Terminated 19021

SessColn.TSessionItem.TimeoutMinutes 19022
SessColn.TSessionItem.TouchTime 19022
SessColn.TSessions 19022
SessColn.TSessions.ActiveCount 19033
SessColn.TSessions.CheckStatusChange 19026
SessColn.TSessions.ConnectActiveSession 19026
SessColn.TSessions.ConnectSession 19027
SessColn.TSessions.Create 19025
SessColn.TSessions.CreateSessionIDGen 19027
SessColn.TSessions.Destroy 19025
SessColn.TSessions.DisconnectSession 19027
SessColn.TSessions.FileName 19033
SessColn.TSessions.GetExpirationTime 19027
SessColn.TSessions.GetExpired 19027
SessColn.TSessions.GetIDCount 19028
SessColn.TSessions.GetIDs 19028
SessColn.TSessions.GetIsActive 19028
SessColn.TSessions.GetItemCount 19028
SessColn.TSessions.GetItems 19029
SessColn.TSessions.GetSession 19029
SessColn.TSessions.GetSessionIDGen 19029
SessColn.TSessions.GetStartTime 19029
SessColn.TSessions.GetTerminated 19029
SessColn.TSessions.GetTimeoutMinutes 19030
SessColn.TSessions.GetTouchTime 19030
SessColn.TSessions.ItemValue 19030
SessColn.TSessions.LoadFromFile 19030
SessColn.TSessions.LoadFromStream 19031
SessColn.TSessions.MaxSessions 19033
SessColn.TSessions.NotifyStatusChange 19031
SessColn.TSessions.SaveToFile 19031
SessColn.TSessions.SaveToStream 19031
SessColn.TSessions.SessionExists 19031
SessColn.TSessions.SetItemValue 19032
SessColn.TSessions.SetTimeoutMinutes 19032
SessColn.TSessions.StartSession 19032
SessColn.TSessions.TerminateSession 19032
SessColn.TSessions.TouchSession 19033
SessColn.TSessionStatusFlag 19035
SessColn.TSessionStatusFlag enumeration 19035
SessColn.TSessionStatusFlags 19036
SessColn.TSessionStatusFlags type 19036
seWrite enumeration member 18887
sffCurrency enumeration member 22101
sffExponent enumeration member 22101
sffFixed enumeration member 22101
sffGeneral enumeration member 22101
sffNumber enumeration member 22101
SHDocVw 19039
SHDocVw namespace 19039
    Classes 19045
    Constants 19486
    Interfaces 19276
    Types 19451
SHDocVw._SearchAssistantEvents 19276
SHDocVw._SearchAssistantEvents.OnNewSearch 19277
SHDocVw._SearchAssistantEvents.OnNextMenuSelect 19277
SHDocVw.BrowserNavConstants 19453
SHDocVw.BrowserNavConstants type 19453
SHDocVw.CGID_IWebBrowser 19488
SHDocVw.CGID_IWebBrowser constant 19488
SHDocVw.CLASS_CScriptErrorList 19489
SHDocVw.CLASS_CScriptErrorList constant 19489
SHDocVw.CLASS_InternetExplorer 19489
SHDocVw.CLASS_InternetExplorer constant 19489
SHDocVw.CLASS_SearchAssistantOC 19489
SHDocVw.CLASS_SearchAssistantOC constant 19489
SHDocVw.CLASS_ShellBrowserWindow 19490
SHDocVw.CLASS_ShellBrowserWindow constant 19490
SHDocVw.CLASS_ShellNameSpace 19490
SHDocVw.CLASS_ShellNameSpace constant 19490
SHDocVw.CLASS_ShellUIHelper 19490
SHDocVw.CLASS_ShellUIHelper constant 19490
SHDocVw.CLASS_ShellWindows 19491
SHDocVw.CLASS_ShellWindows constant 19491
SHDocVw.CLASS_WebBrowser 19491
SHDocVw.CLASS_WebBrowser constant 19491
SHDocVw.CLASS_WebBrowser_V1 19491

SHDocVw.CLASS_WebBrowser_V1 constant 19491
SHDocVw.CoCScriptErrorList 19045
SHDocVw.CoCScriptErrorList.Create 19046
SHDocVw.CoCScriptErrorList.CreateRemote 19046
SHDocVw.CoInternetExplorer 19046
SHDocVw.CoInternetExplorer.Create 19047
SHDocVw.CoInternetExplorer.CreateRemote 19047
SHDocVw.CommandStateChangeConstants 19453
SHDocVw.CommandStateChangeConstants type 19453
SHDocVw.CoSearchAssistantOC 19047
SHDocVw.CoSearchAssistantOC.Create 19048
SHDocVw.CoSearchAssistantOC.CreateRemote 19048
SHDocVw.CoShellBrowserWindow 19049
SHDocVw.CoShellBrowserWindow.Create 19049
SHDocVw.CoShellBrowserWindow.CreateRemote 19050
SHDocVw.CoShellNameSpace 19050
SHDocVw.CoShellNameSpace.Create 19051
SHDocVw.CoShellNameSpace.CreateRemote 19051
SHDocVw.CoShellUIHelper 19051
SHDocVw.CoShellUIHelper.Create 19052
SHDocVw.CoShellUIHelper.CreateRemote 19052
SHDocVw.CoShellWindows 19052
SHDocVw.CoShellWindows.Create 19053
SHDocVw.CoShellWindows.CreateRemote 19053
SHDocVw.CSC_NAVIGATEBACK 19492
SHDocVw.CSC_NAVIGATEBACK constant 19492
SHDocVw.CSC_NAVIGATEFORWARD 19492
SHDocVw.CSC_NAVIGATEFORWARD constant 19492
SHDocVw.CSC_UPDATECOMMANDS 19492
SHDocVw.CSC_UPDATECOMMANDS constant 19492
SHDocVw.CScriptErrorList 19453
SHDocVw.CScriptErrorList type 19453
SHDocVw.DIID__SearchAssistantEvents 19493
SHDocVw.DIID__SearchAssistantEvents constant 19493
SHDocVw.DIID_DShellNameSpaceEvents 19493
SHDocVw.DIID_DShellNameSpaceEvents constant 19493
SHDocVw.DIID_DShellWindowsEvents 19493
SHDocVw.DIID_DShellWindowsEvents constant 19493
SHDocVw.DIID_DWebBrowserEvents 19494
SHDocVw.DIID_DWebBrowserEvents constant 19494
SHDocVw.DIID_DWebBrowserEvents2 19494
SHDocVw.DIID_DWebBrowserEvents2 constant 19494
SHDocVw.DShellNameSpaceEvents 19278
SHDocVw.DShellNameSpaceEvents.DoubleClick 19278
SHDocVw.DShellNameSpaceEvents.FavoritesSelectionChange 19279
SHDocVw.DShellNameSpaceEvents.Initialized 19279
SHDocVw.DShellNameSpaceEvents.SelectionChange 19279
SHDocVw.DShellWindowsEvents 19279
SHDocVw.DShellWindowsEvents.WindowRegistered 19280
SHDocVw.DShellWindowsEvents.WindowRevoked 19280
SHDocVw.dtlOcxPage 19494
SHDocVw.dtlOcxPage constant 19494
SHDocVw.dtlServerPage 19495
SHDocVw.dtlServerPage constant 19495
SHDocVw.DWebBrowserEvents 19281
SHDocVw.DWebBrowserEvents.BeforeNavigate 19282
SHDocVw.DWebBrowserEvents.CommandStateChange 19282
SHDocVw.DWebBrowserEvents.DownloadBegin 19282
SHDocVw.DWebBrowserEvents.DownloadComplete 19283
SHDocVw.DWebBrowserEvents.FrameBeforeNavigate 19283
SHDocVw.DWebBrowserEvents.FrameNavigateComplete 19283
SHDocVw.DWebBrowserEvents.FrameNewWindow 19283
SHDocVw.DWebBrowserEvents.NavigateComplete 19284
SHDocVw.DWebBrowserEvents.NewWindow 19284
SHDocVw.DWebBrowserEvents.ProgressChange 19284
SHDocVw.DWebBrowserEvents.PropertyChange 19284
SHDocVw.DWebBrowserEvents.Quit 19284
SHDocVw.DWebBrowserEvents.StatusTextChange 19285
SHDocVw.DWebBrowserEvents.TitleChange 19285
SHDocVw.DWebBrowserEvents.WindowActivate 19285
SHDocVw.DWebBrowserEvents.WindowMove 19285
SHDocVw.DWebBrowserEvents.WindowResize 19286
SHDocVw.DWebBrowserEvents2 19286
SHDocVw.DWebBrowserEvents2.BeforeNavigate2 19288
SHDocVw.DWebBrowserEvents2.ClientToHostWindow 19288
SHDocVw.DWebBrowserEvents2.CommandStateChange 19288

SHDocVw.DWebBrowserEvents2.DocumentComplete 19288
SHDocVw.DWebBrowserEvents2.DownloadBegin 19289
SHDocVw.DWebBrowserEvents2.DownloadComplete 19289
SHDocVw.DWebBrowserEvents2.FileDownload 19289
SHDocVw.DWebBrowserEvents2.NavigateComplete2 19289
SHDocVw.DWebBrowserEvents2.NavigateError 19289
SHDocVw.DWebBrowserEvents2.NewWindow2 19290
SHDocVw.DWebBrowserEvents2.OnFullScreen 19290
SHDocVw.DWebBrowserEvents2.OnMenuBar 19290
SHDocVw.DWebBrowserEvents2.OnQuit 19290
SHDocVw.DWebBrowserEvents2.OnStatusBar 19291
SHDocVw.DWebBrowserEvents2.OnTheaterMode 19291
SHDocVw.DWebBrowserEvents2.OnToolBar 19291
SHDocVw.DWebBrowserEvents2.OnVisible 19291
SHDocVw.DWebBrowserEvents2.PrintTemplateInstantiation 19291
SHDocVw.DWebBrowserEvents2.PrintTemplateTeardown 19292
SHDocVw.DWebBrowserEvents2.PrivacyImpactedStateChange 19292
SHDocVw.DWebBrowserEvents2.ProgressChange 19292
SHDocVw.DWebBrowserEvents2.PropertyChange 19292
SHDocVw.DWebBrowserEvents2.SetSecureLockIcon 19293
SHDocVw.DWebBrowserEvents2.StatusTextChange 19293
SHDocVw.DWebBrowserEvents2.TitleChange 19293
SHDocVw.DWebBrowserEvents2.UpdatePageStatus 19293
SHDocVw.DWebBrowserEvents2.WindowClosing 19293
SHDocVw.DWebBrowserEvents2.WindowSetHeight 19294
SHDocVw.DWebBrowserEvents2.WindowSetLeft 19294
SHDocVw.DWebBrowserEvents2.WindowSetResizable 19294
SHDocVw.DWebBrowserEvents2.WindowSetTop 19294
SHDocVw.DWebBrowserEvents2.WindowSetWidth 19295
SHDocVw.HTMLID_FIND 19495
SHDocVw.HTMLID_FIND constant 19495
SHDocVw.HTMLID_OPTIONS 19495
SHDocVw.HTMLID_OPTIONS constant 19495
SHDocVw.HTMLID_VIEWSOURCE 19496
SHDocVw.HTMLID_VIEWSOURCE constant 19496
SHDocVw.IID_IScriptErrorList 19496
SHDocVw.IID_IScriptErrorList constant 19496
SHDocVw.IID_ISearch 19496
SHDocVw.IID_ISearch constant 19496
SHDocVw.IID_ISearchAssistantOC 19497
SHDocVw.IID_ISearchAssistantOC constant 19497
SHDocVw.IID_ISearchAssistantOC2 19497
SHDocVw.IID_ISearchAssistantOC2 constant 19497
SHDocVw.IID_ISearchAssistantOC3 19497
SHDocVw.IID_ISearchAssistantOC3 constant 19497
SHDocVw.IID_ISearches 19498
SHDocVw.IID_ISearches constant 19498
SHDocVw.IID_IShellFavoritesNameSpace 19498
SHDocVw.IID_IShellFavoritesNameSpace constant 19498
SHDocVw.IID_IShellNameSpace 19498
SHDocVw.IID_IShellNameSpace constant 19498
SHDocVw.IID_IShellUIHelper 19499
SHDocVw.IID_IShellUIHelper constant 19499
SHDocVw.IID_IShellWindows 19499
SHDocVw.IID_IShellWindows constant 19499
SHDocVw.IID_IWebBrowser 19499
SHDocVw.IID_IWebBrowser constant 19499
SHDocVw.IID_IWebBrowser2 19500
SHDocVw.IID_IWebBrowser2 constant 19500
SHDocVw.IID_IWebBrowserApp 19500
SHDocVw.IID_IWebBrowserApp constant 19500
SHDocVw.InternetExplorer 19454
SHDocVw.InternetExplorer type 19454
SHDocVw.IScriptErrorList 19295
SHDocVw.IScriptErrorList.advanceError 19296
SHDocVw.IScriptErrorList.canAdvanceError 19296
SHDocVw.IScriptErrorList.canRetreatError 19296
SHDocVw.IScriptErrorList.getAlwaysShowLockState 19297
SHDocVw.IScriptErrorList.getDetailsPaneOpen 19297
SHDocVw.IScriptErrorList.getErrorChar 19297
SHDocVw.IScriptErrorList.getErrorCode 19297
SHDocVw.IScriptErrorList.getErrorLine 19298
SHDocVw.IScriptErrorList.getErrorMsg 19298
SHDocVw.IScriptErrorList.getErrorUrl 19298
SHDocVw.IScriptErrorList.getPerErrorDisplay 19298
SHDocVw.IScriptErrorList.retreatError 19298

SHDocVw.IScriptErrorList.setDetailsPaneOpen 19299
SHDocVw.IScriptErrorList.setPerErrorDisplay 19299
SHDocVw.IScriptErrorListDisp 19299
SHDocVw.IScriptErrorListDisp.advanceError 19300
SHDocVw.IScriptErrorListDisp.canAdvanceError 19301
SHDocVw.IScriptErrorListDisp.canRetreatError 19301
SHDocVw.IScriptErrorListDisp.getAlwaysShowLockState 19301
SHDocVw.IScriptErrorListDisp.getDetailsPaneOpen 19301
SHDocVw.IScriptErrorListDisp.getErrorChar 19301
SHDocVw.IScriptErrorListDisp.getErrorCode 19302
SHDocVw.IScriptErrorListDisp.getErrorLine 19302
SHDocVw.IScriptErrorListDisp.getErrorMsg 19302
SHDocVw.IScriptErrorListDisp.getErrorUrl 19302
SHDocVw.IScriptErrorListDisp.getPerErrorDisplay 19303
SHDocVw.IScriptErrorListDisp.retreatError 19303
SHDocVw.IScriptErrorListDisp.setDetailsPaneOpen 19303
SHDocVw.IScriptErrorListDisp.setPerErrorDisplay 19303
SHDocVw.ISearch 19303
SHDocVw.ISearch.Get_Id 19304
SHDocVw.ISearch.Get_Title 19305
SHDocVw.ISearch.Get_URL 19305
SHDocVw.ISearch.Id 19305
SHDocVw.ISearch.Title 19305
SHDocVw.ISearch.URL 19306
SHDocVw.ISearchAssistantOC 19306
SHDocVw.ISearchAssistantOC.AddNextMenuItem 19308
SHDocVw.ISearchAssistantOC.ASProvider 19314
SHDocVw.ISearchAssistantOC.ASSetting 19314
SHDocVw.ISearchAssistantOC.EncodeString 19308
SHDocVw.ISearchAssistantOC.EventHandled 19315
SHDocVw.ISearchAssistantOC.FindComputer 19308
SHDocVw.ISearchAssistantOC.FindFilesOrFolders 19309
SHDocVw.ISearchAssistantOC.FindOnWeb 19309
SHDocVw.ISearchAssistantOC.FindPeople 19309
SHDocVw.ISearchAssistantOC.FindPrinter 19309
SHDocVw.ISearchAssistantOC.Get_ASProvider 19309
SHDocVw.ISearchAssistantOC.Get_ASSetting 19310
SHDocVw.ISearchAssistantOC.Get_InWebFolder 19310
SHDocVw.ISearchAssistantOC.Get_SearchAssistantDefault 19310
SHDocVw.ISearchAssistantOC.Get_Searches 19310
SHDocVw.ISearchAssistantOC.Get_ShellFeaturesEnabled 19311
SHDocVw.ISearchAssistantOC.Get_Version 19311
SHDocVw.ISearchAssistantOC.GetProperty 19311
SHDocVw.ISearchAssistantOC.GetSearchAssistantURL 19311
SHDocVw.ISearchAssistantOC.InWebFolder 19315
SHDocVw.ISearchAssistantOC.IsRestricted 19311
SHDocVw.ISearchAssistantOC.NavigateToDefaultSearch 19312
SHDocVw.ISearchAssistantOC.NETDetectNextNavigate 19312
SHDocVw.ISearchAssistantOC.NotifySearchSettingsChanged 19312
SHDocVw.ISearchAssistantOC.PutFindText 19312
SHDocVw.ISearchAssistantOC.PutProperty 19313
SHDocVw.ISearchAssistantOC.ResetNextMenu 19313
SHDocVw.ISearchAssistantOC.SearchAssistantDefault 19315
SHDocVw.ISearchAssistantOC.Searches 19315
SHDocVw.ISearchAssistantOC.Set_ASProvider 19313
SHDocVw.ISearchAssistantOC.Set_ASSetting 19313
SHDocVw.ISearchAssistantOC.Set_EventHandled 19313
SHDocVw.ISearchAssistantOC.SetDefaultSearchUrl 19314
SHDocVw.ISearchAssistantOC.ShellFeaturesEnabled 19316
SHDocVw.ISearchAssistantOC.Version 19316
SHDocVw.ISearchAssistantOC2 19316
SHDocVw.ISearchAssistantOC2.Get_ShowFindPrinter 19318
SHDocVw.ISearchAssistantOC2.ShowFindPrinter 19318
SHDocVw.ISearchAssistantOC2Disp 19318
SHDocVw.ISearchAssistantOC2Disp.AddNextMenuItem 19320
SHDocVw.ISearchAssistantOC2Disp.ASProvider 19324
SHDocVw.ISearchAssistantOC2Disp.ASSetting 19324
SHDocVw.ISearchAssistantOC2Disp.EncodeString 19320
SHDocVw.ISearchAssistantOC2Disp.EventHandled 19324
SHDocVw.ISearchAssistantOC2Disp.FindComputer 19320
SHDocVw.ISearchAssistantOC2Disp.FindFilesOrFolders 19320
SHDocVw.ISearchAssistantOC2Disp.FindOnWeb 19321
SHDocVw.ISearchAssistantOC2Disp.FindPeople 19321
SHDocVw.ISearchAssistantOC2Disp.FindPrinter 19321
SHDocVw.ISearchAssistantOC2Disp.GetProperty 19321
SHDocVw.ISearchAssistantOC2Disp.GetSearchAssistantURL 19322

SHDocVw.ISearchAssistantOC2Disp.InWebFolder 19325
SHDocVw.ISearchAssistantOC2Disp.IsRestricted 19322
SHDocVw.ISearchAssistantOC2Disp.NavigateToDefaultSearch 19322
SHDocVw.ISearchAssistantOC2Disp.NETDetectNextNavigate 19322
SHDocVw.ISearchAssistantOC2Disp.NotifySearchSettingsChanged 19322
SHDocVw.ISearchAssistantOC2Disp.PutFindText 19323
SHDocVw.ISearchAssistantOC2Disp.PutProperty 19323
SHDocVw.ISearchAssistantOC2Disp.ResetNextMenu 19323
SHDocVw.ISearchAssistantOC2Disp.SearchAssistantDefault 19325
SHDocVw.ISearchAssistantOC2Disp.Searches 19325
SHDocVw.ISearchAssistantOC2Disp.SetDefaultSearchUrl 19323
SHDocVw.ISearchAssistantOC2Disp.ShellFeaturesEnabled 19325
SHDocVw.ISearchAssistantOC2Disp.ShowFindPrinter 19325
SHDocVw.ISearchAssistantOC2Disp.Version 19326
SHDocVw.ISearchAssistantOC3 19326
SHDocVw.ISearchAssistantOC3.Get_SearchCompanionAvailable 19328
SHDocVw.ISearchAssistantOC3.Get_UseSearchCompanion 19328
SHDocVw.ISearchAssistantOC3.SearchCompanionAvailable 19328
SHDocVw.ISearchAssistantOC3.Set_UseSearchCompanion 19328
SHDocVw.ISearchAssistantOC3.UseSearchCompanion 19329
SHDocVw.ISearchAssistantOC3Disp 19329
SHDocVw.ISearchAssistantOC3Disp.AddNextMenuItem 19330
SHDocVw.ISearchAssistantOC3Disp.ASProvider 19335
SHDocVw.ISearchAssistantOC3Disp.ASSetting 19335
SHDocVw.ISearchAssistantOC3Disp.EncodeString 19331
SHDocVw.ISearchAssistantOC3Disp.EventHandled 19335
SHDocVw.ISearchAssistantOC3Disp.FindComputer 19331
SHDocVw.ISearchAssistantOC3Disp.FindFilesOrFolders 19331
SHDocVw.ISearchAssistantOC3Disp.FindOnWeb 19331
SHDocVw.ISearchAssistantOC3Disp.FindPeople 19332
SHDocVw.ISearchAssistantOC3Disp.FindPrinter 19332
SHDocVw.ISearchAssistantOC3Disp.GetProperty 19332
SHDocVw.ISearchAssistantOC3Disp.GetSearchAssistantURL 19332
SHDocVw.ISearchAssistantOC3Disp.InWebFolder 19335
SHDocVw.ISearchAssistantOC3Disp.IsRestricted 19332
SHDocVw.ISearchAssistantOC3Disp.NavigateToDefaultSearch 19333
SHDocVw.ISearchAssistantOC3Disp.NETDetectNextNavigate 19333
SHDocVw.ISearchAssistantOC3Disp.NotifySearchSettingsChanged 19333
SHDocVw.ISearchAssistantOC3Disp.PutFindText 19333
SHDocVw.ISearchAssistantOC3Disp.PutProperty 19334
SHDocVw.ISearchAssistantOC3Disp.ResetNextMenu 19334
SHDocVw.ISearchAssistantOC3Disp.SearchAssistantDefault 19335
SHDocVw.ISearchAssistantOC3Disp.SearchCompanionAvailable 19336
SHDocVw.ISearchAssistantOC3Disp.Searches 19336
SHDocVw.ISearchAssistantOC3Disp.SetDefaultSearchUrl 19334
SHDocVw.ISearchAssistantOC3Disp.ShellFeaturesEnabled 19336
SHDocVw.ISearchAssistantOC3Disp.ShowFindPrinter 19336
SHDocVw.ISearchAssistantOC3Disp.UseSearchCompanion 19337
SHDocVw.ISearchAssistantOC3Disp.Version 19337
SHDocVw.ISearchAssistantOCDisp 19337
SHDocVw.ISearchAssistantOCDisp.AddNextMenuItem 19339
SHDocVw.ISearchAssistantOCDisp.ASProvider 19343
SHDocVw.ISearchAssistantOCDisp.ASSetting 19343
SHDocVw.ISearchAssistantOCDisp.EncodeString 19339
SHDocVw.ISearchAssistantOCDisp.EventHandled 19343
SHDocVw.ISearchAssistantOCDisp.FindComputer 19339
SHDocVw.ISearchAssistantOCDisp.FindFilesOrFolders 19339
SHDocVw.ISearchAssistantOCDisp.FindOnWeb 19339
SHDocVw.ISearchAssistantOCDisp.FindPeople 19340
SHDocVw.ISearchAssistantOCDisp.FindPrinter 19340
SHDocVw.ISearchAssistantOCDisp.GetProperty 19340
SHDocVw.ISearchAssistantOCDisp.GetSearchAssistantURL 19340
SHDocVw.ISearchAssistantOCDisp.InWebFolder 19343
SHDocVw.ISearchAssistantOCDisp.IsRestricted 19341

SHDocVw.ISearchAssistantOCDisp.NavigateToDefaultSearch 19341
SHDocVw.ISearchAssistantOCDisp.NETDetectNextNavigate 19341
SHDocVw.ISearchAssistantOCDisp.NotifySearchSettingsChanged 19341
SHDocVw.ISearchAssistantOCDisp.PutFindText 19341
SHDocVw.ISearchAssistantOCDisp.PutProperty 19342
SHDocVw.ISearchAssistantOCDisp.ResetNextMenu 19342
SHDocVw.ISearchAssistantOCDisp.SearchAssistantDefault 19343
SHDocVw.ISearchAssistantOCDisp.Searches 19344
SHDocVw.ISearchAssistantOCDisp.SetDefaultSearchUrl 19342
SHDocVw.ISearchAssistantOCDisp.ShellFeaturesEnabled 19344
SHDocVw.ISearchAssistantOCDisp.Version 19344
SHDocVw.ISearchDisp 19344
SHDocVw.ISearchDisp.Id 19345
SHDocVw.ISearchDisp.Title 19345
SHDocVw.ISearchDisp.URL 19346
SHDocVw.ISearches 19346
SHDocVw.ISearches._NewEnum 19347
SHDocVw.ISearches.Count 19348
SHDocVw.ISearches.Default 19348
SHDocVw.ISearches.Get_Count 19347
SHDocVw.ISearches.Get_Default 19347
SHDocVw.ISearches.Item 19347
SHDocVw.ISearchesDisp 19348
SHDocVw.ISearchesDisp._NewEnum 19349
SHDocVw.ISearchesDisp.Count 19350
SHDocVw.ISearchesDisp.Default 19350
SHDocVw.ISearchesDisp.Item 19349
SHDocVw.IShellFavoritesNameSpace 19350
SHDocVw.IShellFavoritesNameSpace.CreateSubscriptionForSelection 19352
SHDocVw.IShellFavoritesNameSpace.DeleteSubscriptionForSelection 19352
SHDocVw.IShellFavoritesNameSpace.Export 19352
SHDocVw.IShellFavoritesNameSpace.Get_SubscriptionsEnabled 19352
SHDocVw.IShellFavoritesNameSpace.Import 19353
SHDocVw.IShellFavoritesNameSpace.InvokeContextMenuCommand 19353
SHDocVw.IShellFavoritesNameSpace.MoveSelectionDown 19353
SHDocVw.IShellFavoritesNameSpace.MoveSelectionTo 19353
SHDocVw.IShellFavoritesNameSpace.MoveSelectionUp 19353
SHDocVw.IShellFavoritesNameSpace.NewFolder 19354
SHDocVw.IShellFavoritesNameSpace.ResetSort 19354
SHDocVw.IShellFavoritesNameSpace.SetRoot 19354
SHDocVw.IShellFavoritesNameSpace.SubscriptionsEnabled 19355
SHDocVw.IShellFavoritesNameSpace.Synchronize 19354
SHDocVw.IShellFavoritesNameSpaceDisp 19355
SHDocVw.IShellFavoritesNameSpaceDisp.CreateSubscriptionForSelection 19356
SHDocVw.IShellFavoritesNameSpaceDisp.DeleteSubscriptionForSelection 19356
SHDocVw.IShellFavoritesNameSpaceDisp.Export 19357
SHDocVw.IShellFavoritesNameSpaceDisp.Import 19357
SHDocVw.IShellFavoritesNameSpaceDisp.InvokeContextMenuCommand 19357
SHDocVw.IShellFavoritesNameSpaceDisp.MoveSelectionDown 19357
SHDocVw.IShellFavoritesNameSpaceDisp.MoveSelectionTo 19358
SHDocVw.IShellFavoritesNameSpaceDisp.MoveSelectionUp 19358
SHDocVw.IShellFavoritesNameSpaceDisp.NewFolder 19358
SHDocVw.IShellFavoritesNameSpaceDisp.ResetSort 19358
SHDocVw.IShellFavoritesNameSpaceDisp.SetRoot 19358
SHDocVw.IShellFavoritesNameSpaceDisp.SubscriptionsEnabled 19359
SHDocVw.IShellFavoritesNameSpaceDisp.Synchronize 19359
SHDocVw.IShellNameSpace 19359
SHDocVw.IShellNameSpace.Columns 19366
SHDocVw.IShellNameSpace.CountViewTypes 19367
SHDocVw.IShellNameSpace.Depth 19367
SHDocVw.IShellNameSpace.EnumOptions 19367
SHDocVw.IShellNameSpace.Expand 19361

SHDocVw.IShellNameSpace.Flags 19367
SHDocVw.IShellNameSpace.Get_Columns 19362
SHDocVw.IShellNameSpace.Get_CountViewTypes 19362
SHDocVw.IShellNameSpace.Get_Depth 19362
SHDocVw.IShellNameSpace.Get_EnumOptions 19362
SHDocVw.IShellNameSpace.Get_Flags 19362
SHDocVw.IShellNameSpace.Get_Mode 19363
SHDocVw.IShellNameSpace.Get_Root 19363
SHDocVw.IShellNameSpace.Get_SelectedItem 19363
SHDocVw.IShellNameSpace.Get_TVFlags 19363
SHDocVw.IShellNameSpace.Mode 19367
SHDocVw.IShellNameSpace.Root 19368
SHDocVw.IShellNameSpace.SelectedItem 19368
SHDocVw.IShellNameSpace.SelectedItems 19364
SHDocVw.IShellNameSpace.Set_Columns 19364
SHDocVw.IShellNameSpace.Set_Depth 19364
SHDocVw.IShellNameSpace.Set_EnumOptions 19364
SHDocVw.IShellNameSpace.Set_Flags 19364
SHDocVw.IShellNameSpace.Set_Mode 19365
SHDocVw.IShellNameSpace.Set_Root 19365
SHDocVw.IShellNameSpace.Set_SelectedItem 19365
SHDocVw.IShellNameSpace.Set_TVFlags 19365
SHDocVw.IShellNameSpace.SetViewType 19366
SHDocVw.IShellNameSpace.TVFlags 19368
SHDocVw.IShellNameSpace.UnselectAll 19366
SHDocVw.IShellNameSpaceDisp 19368
SHDocVw.IShellNameSpaceDisp.Columns 19374
SHDocVw.IShellNameSpaceDisp.CountViewTypes 19374
SHDocVw.IShellNameSpaceDisp.CreateSubscriptionForSelection 19370
SHDocVw.IShellNameSpaceDisp.DeleteSubscriptionForSelection 19370
SHDocVw.IShellNameSpaceDisp.Depth 19374
SHDocVw.IShellNameSpaceDisp.EnumOptions 19374
SHDocVw.IShellNameSpaceDisp.Expand 19370
SHDocVw.IShellNameSpaceDisp.Export 19370
SHDocVw.IShellNameSpaceDisp.Flags 19374
SHDocVw.IShellNameSpaceDisp.Import 19371
SHDocVw.IShellNameSpaceDisp.InvokeContextMenuCommand 19371
SHDocVw.IShellNameSpaceDisp.Mode 19375
SHDocVw.IShellNameSpaceDisp.MoveSelectionDown 19371
SHDocVw.IShellNameSpaceDisp.MoveSelectionTo 19371
SHDocVw.IShellNameSpaceDisp.MoveSelectionUp 19372
SHDocVw.IShellNameSpaceDisp.NewFolder 19372
SHDocVw.IShellNameSpaceDisp.ResetSort 19372
SHDocVw.IShellNameSpaceDisp.Root 19375
SHDocVw.IShellNameSpaceDisp.SelectedItem 19375
SHDocVw.IShellNameSpaceDisp.SelectedItems 19372
SHDocVw.IShellNameSpaceDisp.SetRoot 19372
SHDocVw.IShellNameSpaceDisp.SetViewType 19373
SHDocVw.IShellNameSpaceDisp.SubscriptionsEnabled 19375
SHDocVw.IShellNameSpaceDisp.Synchronize 19373
SHDocVw.IShellNameSpaceDisp.TVFlags 19376
SHDocVw.IShellNameSpaceDisp.UnselectAll 19373
SHDocVw.IShellUIHelper 19376
SHDocVw.IShellUIHelper.AddChannel 19377
SHDocVw.IShellUIHelper.AddDesktopComponent 19377
SHDocVw.IShellUIHelper.AddFavorite 19377
SHDocVw.IShellUIHelper.AutoCompleteAttach 19378
SHDocVw.IShellUIHelper.AutoCompleteSaveForm 19378
SHDocVw.IShellUIHelper.AutoScan 19378
SHDocVw.IShellUIHelper.ImportExportFavorites 19378
SHDocVw.IShellUIHelper.IsSubscribed 19379
SHDocVw.IShellUIHelper.NavigateAndFind 19379
SHDocVw.IShellUIHelper.RefreshOfflineDesktop 19379
SHDocVw.IShellUIHelper.ResetFirstBootMode 19379
SHDocVw.IShellUIHelper.ResetSafeMode 19379
SHDocVw.IShellUIHelper.ShowBrowserUI 19380
SHDocVw.IShellUIHelperDisp 19380
SHDocVw.IShellUIHelperDisp.AddChannel 19381
SHDocVw.IShellUIHelperDisp.AddDesktopComponent 19381
SHDocVw.IShellUIHelperDisp.AddFavorite 19382
SHDocVw.IShellUIHelperDisp.AutoCompleteAttach 19382
SHDocVw.IShellUIHelperDisp.AutoCompleteSaveForm 19382
SHDocVw.IShellUIHelperDisp.AutoScan 19382
SHDocVw.IShellUIHelperDisp.ImportExportFavorites 19382
SHDocVw.IShellUIHelperDisp.IsSubscribed 19383

SHDocVw.IShellUIHelperDisp.NavigateAndFind 19383
SHDocVw.IShellUIHelperDisp.RefreshOfflineDesktop 19383
SHDocVw.IShellUIHelperDisp.ResetFirstBootMode 19383
SHDocVw.IShellUIHelperDisp.ResetSafeMode 19384
SHDocVw.IShellUIHelperDisp.ShowBrowserUI 19384
SHDocVw.IShellWindows 19384
SHDocVw.IShellWindows._NewEnum 19385
SHDocVw.IShellWindows.Count 19388
SHDocVw.IShellWindows.FindWindowSW 19385
SHDocVw.IShellWindows.Get_Count 19386
SHDocVw.IShellWindows.Item 19386
SHDocVw.IShellWindows.OnActivated 19386
SHDocVw.IShellWindows.OnCreated 19386
SHDocVw.IShellWindows.OnNavigate 19387
SHDocVw.IShellWindows.ProcessAttachDetach 19387
SHDocVw.IShellWindows.Register 19387
SHDocVw.IShellWindows.RegisterPending 19387
SHDocVw.IShellWindows.Revoke 19387
SHDocVw.IShellWindowsDisp 19388
SHDocVw.IShellWindowsDisp._NewEnum 19389
SHDocVw.IShellWindowsDisp.Count 19392
SHDocVw.IShellWindowsDisp.FindWindowSW 19389
SHDocVw.IShellWindowsDisp.Item 19390
SHDocVw.IShellWindowsDisp.OnActivated 19390
SHDocVw.IShellWindowsDisp.OnCreated 19390
SHDocVw.IShellWindowsDisp.OnNavigate 19390
SHDocVw.IShellWindowsDisp.ProcessAttachDetach 19391
SHDocVw.IShellWindowsDisp.Register 19391
SHDocVw.IShellWindowsDisp.RegisterPending 19391
SHDocVw.IShellWindowsDisp.Revoke 19391
SHDocVw.IWebBrowser 19392
SHDocVw.IWebBrowser.Application 19400
SHDocVw.IWebBrowser.Busy 19400
SHDocVw.IWebBrowser.Container 19400
SHDocVw.IWebBrowser.Document 19400
SHDocVw.IWebBrowser.Get_Application 19394
SHDocVw.IWebBrowser.Get_Busy 19394
SHDocVw.IWebBrowser.Get_Container 19394
SHDocVw.IWebBrowser.Get_Document 19394
SHDocVw.IWebBrowser.Get_Height 19395
SHDocVw.IWebBrowser.Get_Left 19395
SHDocVw.IWebBrowser.Get_LocationName 19395
SHDocVw.IWebBrowser.Get_LocationURL 19395
SHDocVw.IWebBrowser.Get_Parent 19396
SHDocVw.IWebBrowser.Get_Top 19396
SHDocVw.IWebBrowser.Get_TopLevelContainer 19396
SHDocVw.IWebBrowser.Get_type_ 19396
SHDocVw.IWebBrowser.Get_Width 19396
SHDocVw.IWebBrowser.GoBack 19397
SHDocVw.IWebBrowser.GoForward 19397
SHDocVw.IWebBrowser.GoHome 19397
SHDocVw.IWebBrowser.GoSearch 19397
SHDocVw.IWebBrowser.Height 19401
SHDocVw.IWebBrowser.Left 19401
SHDocVw.IWebBrowser.LocationName 19401
SHDocVw.IWebBrowser.LocationURL 19401
SHDocVw.IWebBrowser.Navigate 19398
SHDocVw.IWebBrowser.Parent 19401
SHDocVw.IWebBrowser.Refresh 19398
SHDocVw.IWebBrowser.Refresh2 19398
SHDocVw.IWebBrowser.Set_Height 19398
SHDocVw.IWebBrowser.Set_Left 19398
SHDocVw.IWebBrowser.Set_Top 19399
SHDocVw.IWebBrowser.Set_Width 19399
SHDocVw.IWebBrowser.Stop 19399
SHDocVw.IWebBrowser.Top 19402
SHDocVw.IWebBrowser.TopLevelContainer 19402
SHDocVw.IWebBrowser.type_ 19402
SHDocVw.IWebBrowser.Width 19402
SHDocVw.IWebBrowser2 19403
SHDocVw.IWebBrowser2.AddressBar 19410
SHDocVw.IWebBrowser2.ExecWB 19405
SHDocVw.IWebBrowser2.Get_AddressBar 19406
SHDocVw.IWebBrowser2.Get_Offline 19406
SHDocVw.IWebBrowser2.Get_ReadyState 19406
SHDocVw.IWebBrowser2.Get_RegisterAsBrowser 19406
SHDocVw.IWebBrowser2.Get_RegisterAsDropTarget 19407
SHDocVw.IWebBrowser2.Get_Resizable 19407

SHDocVw.IWebBrowser2.Get_Silent 19407
SHDocVw.IWebBrowser2.Get_TheaterMode 19407
SHDocVw.IWebBrowser2.Navigate2 19407
SHDocVw.IWebBrowser2.Offline 19410
SHDocVw.IWebBrowser2.QueryStatusWB 19408
SHDocVw.IWebBrowser2.ReadyState 19410
SHDocVw.IWebBrowser2.RegisterAsBrowser 19411
SHDocVw.IWebBrowser2.RegisterAsDropTarget 19411
SHDocVw.IWebBrowser2.Resizable 19411
SHDocVw.IWebBrowser2.Set_AddressBar 19408
SHDocVw.IWebBrowser2.Set_Offline 19408
SHDocVw.IWebBrowser2.Set_RegisterAsBrowser 19408
SHDocVw.IWebBrowser2.Set_RegisterAsDropTarget 19409
SHDocVw.IWebBrowser2.Set_Resizable 19409
SHDocVw.IWebBrowser2.Set_Silent 19409
SHDocVw.IWebBrowser2.Set_TheaterMode 19409
SHDocVw.IWebBrowser2.ShowBrowserBar 19409
SHDocVw.IWebBrowser2.Silent 19411
SHDocVw.IWebBrowser2.TheaterMode 19412
SHDocVw.IWebBrowser2Disp 19412
SHDocVw.IWebBrowser2Disp.AddressBar 19418
SHDocVw.IWebBrowser2Disp.Application 19418
SHDocVw.IWebBrowser2Disp.Busy 19418
SHDocVw.IWebBrowser2Disp.ClientToWindow 19414
SHDocVw.IWebBrowser2Disp.Container 19419
SHDocVw.IWebBrowser2Disp.Document 19419
SHDocVw.IWebBrowser2Disp.ExecWB 19414
SHDocVw.IWebBrowser2Disp.FullName 19419
SHDocVw.IWebBrowser2Disp.FullScreen 19419
SHDocVw.IWebBrowser2Disp.GetProperty 19414
SHDocVw.IWebBrowser2Disp.GoBack 19414
SHDocVw.IWebBrowser2Disp.GoForward 19415
SHDocVw.IWebBrowser2Disp.GoHome 19415
SHDocVw.IWebBrowser2Disp.GoSearch 19415
SHDocVw.IWebBrowser2Disp.Height 19419
SHDocVw.IWebBrowser2Disp.HWnd 19420
SHDocVw.IWebBrowser2Disp.Left 19420
SHDocVw.IWebBrowser2Disp.LocationName 19420
SHDocVw.IWebBrowser2Disp.LocationURL 19420
SHDocVw.IWebBrowser2Disp.MenuBar 19421
SHDocVw.IWebBrowser2Disp.Name 19421
SHDocVw.IWebBrowser2Disp.Navigate 19415
SHDocVw.IWebBrowser2Disp.Navigate2 19415
SHDocVw.IWebBrowser2Disp.Offline 19421
SHDocVw.IWebBrowser2Disp.Parent 19421
SHDocVw.IWebBrowser2Disp.Path 19421
SHDocVw.IWebBrowser2Disp.PutProperty 19416
SHDocVw.IWebBrowser2Disp.QueryStatusWB 19416
SHDocVw.IWebBrowser2Disp.Quit 19416
SHDocVw.IWebBrowser2Disp.ReadyState 19422
SHDocVw.IWebBrowser2Disp.Refresh 19416
SHDocVw.IWebBrowser2Disp.Refresh2 19417
SHDocVw.IWebBrowser2Disp.RegisterAsBrowser 19422
SHDocVw.IWebBrowser2Disp.RegisterAsDropTarget 19422
SHDocVw.IWebBrowser2Disp.Resizable 19422
SHDocVw.IWebBrowser2Disp.ShowBrowserBar 19417
SHDocVw.IWebBrowser2Disp.Silent 19423
SHDocVw.IWebBrowser2Disp.StatusBar 19423
SHDocVw.IWebBrowser2Disp.StatusText 19423
SHDocVw.IWebBrowser2Disp.Stop 19417
SHDocVw.IWebBrowser2Disp.TheaterMode 19423
SHDocVw.IWebBrowser2Disp.ToolBar 19423
SHDocVw.IWebBrowser2Disp.Top 19424
SHDocVw.IWebBrowser2Disp.TopLevelContainer 19424
SHDocVw.IWebBrowser2Disp.type_ 19424
SHDocVw.IWebBrowser2Disp.Visible 19424
SHDocVw.IWebBrowser2Disp.Width 19425
SHDocVw.IWebBrowserApp 19425
SHDocVw.IWebBrowserApp.ClientToWindow 19427
SHDocVw.IWebBrowserApp.FullName 19432
SHDocVw.IWebBrowserApp.FullScreen 19432
SHDocVw.IWebBrowserApp.Get_FullName 19427
SHDocVw.IWebBrowserApp.Get_FullScreen 19428
SHDocVw.IWebBrowserApp.Get_HWnd 19428
SHDocVw.IWebBrowserApp.Get_MenuBar 19428
SHDocVw.IWebBrowserApp.Get_Name 19428
SHDocVw.IWebBrowserApp.Get_Path 19429
SHDocVw.IWebBrowserApp.Get_StatusBar 19429

SHDocVw.IWebBrowserApp.Get_StatusText 19429
SHDocVw.IWebBrowserApp.Get_ToolBar 19429
SHDocVw.IWebBrowserApp.Get_Visible 19429
SHDocVw.IWebBrowserApp.GetProperty 19430
SHDocVw.IWebBrowserApp.HWnd 19432
SHDocVw.IWebBrowserApp.MenuBar 19433
SHDocVw.IWebBrowserApp.Name 19433
SHDocVw.IWebBrowserApp.Path 19433
SHDocVw.IWebBrowserApp.PutProperty 19430
SHDocVw.IWebBrowserApp.Quit 19430
SHDocVw.IWebBrowserApp.Set_FullScreen 19430
SHDocVw.IWebBrowserApp.Set_MenuBar 19431
SHDocVw.IWebBrowserApp.Set_StatusBar 19431
SHDocVw.IWebBrowserApp.Set_StatusText 19431
SHDocVw.IWebBrowserApp.Set_ToolBar 19431
SHDocVw.IWebBrowserApp.Set_Visible 19431
SHDocVw.IWebBrowserApp.StatusBar 19433
SHDocVw.IWebBrowserApp.StatusText 19434
SHDocVw.IWebBrowserApp.ToolBar 19434
SHDocVw.IWebBrowserApp.Visible 19434
SHDocVw.IWebBrowserAppDisp 19434
SHDocVw.IWebBrowserAppDisp.Application 19439
SHDocVw.IWebBrowserAppDisp.Busy 19439
SHDocVw.IWebBrowserAppDisp.ClientToWindow 19436
SHDocVw.IWebBrowserAppDisp.Container 19439
SHDocVw.IWebBrowserAppDisp.Document 19440
SHDocVw.IWebBrowserAppDisp.FullName 19440
SHDocVw.IWebBrowserAppDisp.FullScreen 19440
SHDocVw.IWebBrowserAppDisp.GetProperty 19436
SHDocVw.IWebBrowserAppDisp.GoBack 19436
SHDocVw.IWebBrowserAppDisp.GoForward 19436
SHDocVw.IWebBrowserAppDisp.GoHome 19437
SHDocVw.IWebBrowserAppDisp.GoSearch 19437
SHDocVw.IWebBrowserAppDisp.Height 19440
SHDocVw.IWebBrowserAppDisp.HWnd 19441
SHDocVw.IWebBrowserAppDisp.Left 19441
SHDocVw.IWebBrowserAppDisp.LocationName 19441
SHDocVw.IWebBrowserAppDisp.LocationURL 19441
SHDocVw.IWebBrowserAppDisp.MenuBar 19441
SHDocVw.IWebBrowserAppDisp.Name 19442
SHDocVw.IWebBrowserAppDisp.Navigate 19437
SHDocVw.IWebBrowserAppDisp.Parent 19442
SHDocVw.IWebBrowserAppDisp.Path 19442
SHDocVw.IWebBrowserAppDisp.PutProperty 19437
SHDocVw.IWebBrowserAppDisp.Quit 19438
SHDocVw.IWebBrowserAppDisp.Refresh 19438
SHDocVw.IWebBrowserAppDisp.Refresh2 19438
SHDocVw.IWebBrowserAppDisp.StatusBar 19442
SHDocVw.IWebBrowserAppDisp.StatusText 19443
SHDocVw.IWebBrowserAppDisp.Stop 19438
SHDocVw.IWebBrowserAppDisp.ToolBar 19443
SHDocVw.IWebBrowserAppDisp.Top 19443
SHDocVw.IWebBrowserAppDisp.TopLevelContainer 19443
SHDocVw.IWebBrowserAppDisp.type_ 19443
SHDocVw.IWebBrowserAppDisp.Visible 19444
SHDocVw.IWebBrowserAppDisp.Width 19444
SHDocVw.IWebBrowserDisp 19444
SHDocVw.IWebBrowserDisp.Application 19448
SHDocVw.IWebBrowserDisp.Busy 19448
SHDocVw.IWebBrowserDisp.Container 19448
SHDocVw.IWebBrowserDisp.Document 19448
SHDocVw.IWebBrowserDisp.GoBack 19445
SHDocVw.IWebBrowserDisp.GoForward 19446
SHDocVw.IWebBrowserDisp.GoHome 19446
SHDocVw.IWebBrowserDisp.GoSearch 19446
SHDocVw.IWebBrowserDisp.Height 19448
SHDocVw.IWebBrowserDisp.Left 19449
SHDocVw.IWebBrowserDisp.LocationName 19449
SHDocVw.IWebBrowserDisp.LocationURL 19449
SHDocVw.IWebBrowserDisp.Navigate 19446
SHDocVw.IWebBrowserDisp.Parent 19449
SHDocVw.IWebBrowserDisp.Refresh 19446
SHDocVw.IWebBrowserDisp.Refresh2 19447
SHDocVw.IWebBrowserDisp.Stop 19447
SHDocVw.IWebBrowserDisp.Top 19450
SHDocVw.IWebBrowserDisp.TopLevelContainer 19450
SHDocVw.IWebBrowserDisp.type_ 19450
SHDocVw.IWebBrowserDisp.Width 19450

SHDocVw.LIBID_SHDocVw 19500
SHDocVw.LIBID_SHDocVw constant 19500
SHDocVw.navAllowAutosearch 19501
SHDocVw.navAllowAutosearch constant 19501
SHDocVw.navBrowserBar 19501
SHDocVw.navBrowserBar constant 19501
SHDocVw.navNoHistory 19501
SHDocVw.navNoHistory constant 19501
SHDocVw.navNoReadFromCache 19502
SHDocVw.navNoReadFromCache constant 19502
SHDocVw.navNoWriteToCache 19502
SHDocVw.navNoWriteToCache constant 19502
SHDocVw.navOpenInNewWindow 19502
SHDocVw.navOpenInNewWindow constant 19502
SHDocVw.OLECMDEXECOPT 19454
SHDocVw.OLECMDEXECOPT type 19454
SHDocVw.OLECMDEXECOPT_DODEFAULT 19503
SHDocVw.OLECMDEXECOPT_DODEFAULT constant 19503
SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER 19503
SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER constant 19503
SHDocVw.OLECMDEXECOPT_PROMPTUSER 19503
SHDocVw.OLECMDEXECOPT_PROMPTUSER constant 19503
SHDocVw.OLECMDEXECOPT_SHOWHELP 19504
SHDocVw.OLECMDEXECOPT_SHOWHELP constant 19504
SHDocVw.OLECMDF 19455
SHDocVw.OLECMDF type 19455
SHDocVw.OLECMDF_DEFHIDEONCTXTMENU 19504
SHDocVw.OLECMDF_DEFHIDEONCTXTMENU constant 19504
SHDocVw.OLECMDF_ENABLED 19504
SHDocVw.OLECMDF_ENABLED constant 19504
SHDocVw.OLECMDF_INVISIBLE 19505
SHDocVw.OLECMDF_INVISIBLE constant 19505
SHDocVw.OLECMDF_LATCHED 19505
SHDocVw.OLECMDF_LATCHED constant 19505
SHDocVw.OLECMDF_NINCHED 19505
SHDocVw.OLECMDF_NINCHED constant 19505
SHDocVw.OLECMDF_SUPPORTED 19506
SHDocVw.OLECMDF_SUPPORTED constant 19506
SHDocVw.OLECMDID 19455
SHDocVw.OLECMDID type 19455
SHDocVw.OLECMDID_ALLOWUILESSSAVEAS 19506
SHDocVw.OLECMDID_ALLOWUILESSSAVEAS constant 19506
SHDocVw.OLECMDID_CLEARSELECTION 19506
SHDocVw.OLECMDID_CLEARSELECTION constant 19506
SHDocVw.OLECMDID_CLOSE 19507
SHDocVw.OLECMDID_CLOSE constant 19507
SHDocVw.OLECMDID_COPY 19507
SHDocVw.OLECMDID_COPY constant 19507
SHDocVw.OLECMDID_CUT 19507
SHDocVw.OLECMDID_CUT constant 19507
SHDocVw.OLECMDID_DELETE 19508
SHDocVw.OLECMDID_DELETE constant 19508
SHDocVw.OLECMDID_DONTDOWNLOADCSS 19508
SHDocVw.OLECMDID_DONTDOWNLOADCSS constant 19508
SHDocVw.OLECMDID_ENABLE_INTERACTION 19508
SHDocVw.OLECMDID_ENABLE_INTERACTION constant 19508
SHDocVw.OLECMDID_FIND 19509
SHDocVw.OLECMDID_FIND constant 19509
SHDocVw.OLECMDID_GETPRINTTEMPLATE 19509
SHDocVw.OLECMDID_GETPRINTTEMPLATE constant 19509
SHDocVw.OLECMDID_GETZOOMRANGE 19509
SHDocVw.OLECMDID_GETZOOMRANGE constant 19509
SHDocVw.OLECMDID_HIDETOOLBARS 19510
SHDocVw.OLECMDID_HIDETOOLBARS constant 19510
SHDocVw.OLECMDID_HTTPEQUIV 19510
SHDocVw.OLECMDID_HTTPEQUIV constant 19510
SHDocVw.OLECMDID_HTTPEQUIV_DONE 19510
SHDocVw.OLECMDID_HTTPEQUIV_DONE constant 19510
SHDocVw.OLECMDID_NEW 19511
SHDocVw.OLECMDID_NEW constant 19511
SHDocVw.OLECMDID_ONTOOLBARACTIVATED 19511
SHDocVw.OLECMDID_ONTOOLBARACTIVATED constant 19511
SHDocVw.OLECMDID_ONUNLOAD 19511
SHDocVw.OLECMDID_ONUNLOAD constant 19511
SHDocVw.OLECMDID_OPEN 19512

SHDocVw.OLECMDID_OPEN constant 19512
SHDocVw.OLECMDID_PAGESETUP 19512
SHDocVw.OLECMDID_PAGESETUP constant 19512
SHDocVw.OLECMDID_PASTE 19512
SHDocVw.OLECMDID_PASTE constant 19512
SHDocVw.OLECMDID_PASTESPECIAL 19513
SHDocVw.OLECMDID_PASTESPECIAL constant 19513
SHDocVw.OLECMDID_PREREFRESH 19513
SHDocVw.OLECMDID_PREREFRESH constant 19513
SHDocVw.OLECMDID_PRINT 19513
SHDocVw.OLECMDID_PRINT constant 19513
SHDocVw.OLECMDID_PRINT2 19514
SHDocVw.OLECMDID_PRINT2 constant 19514
SHDocVw.OLECMDID_PRINTPREVIEW 19514
SHDocVw.OLECMDID_PRINTPREVIEW constant 19514
SHDocVw.OLECMDID_PRINTPREVIEW2 19514
SHDocVw.OLECMDID_PRINTPREVIEW2 constant 19514
SHDocVw.OLECMDID_PROPERTIES 19515
SHDocVw.OLECMDID_PROPERTIES constant 19515
SHDocVw.OLECMDID_PROPERTYBAG2 19515
SHDocVw.OLECMDID_PROPERTYBAG2 constant 19515
SHDocVw.OLECMDID_REDO 19515
SHDocVw.OLECMDID_REDO constant 19515
SHDocVw.OLECMDID_REFRESH 19516
SHDocVw.OLECMDID_REFRESH constant 19516
SHDocVw.OLECMDID_SAVE 19516
SHDocVw.OLECMDID_SAVE constant 19516
SHDocVw.OLECMDID_SAVEAS 19516
SHDocVw.OLECMDID_SAVEAS constant 19516
SHDocVw.OLECMDID_SAVECOPYAS 19517
SHDocVw.OLECMDID_SAVECOPYAS constant 19517
SHDocVw.OLECMDID_SELECTALL 19517
SHDocVw.OLECMDID_SELECTALL constant 19517
SHDocVw.OLECMDID_SETDOWNLOADSTATE 19517
SHDocVw.OLECMDID_SETDOWNLOADSTATE constant 19517
SHDocVw.OLECMDID_SETPRINTTEMPLATE 19518
SHDocVw.OLECMDID_SETPRINTTEMPLATE constant 19518
SHDocVw.OLECMDID_SETPROGRESSMAX 19518
SHDocVw.OLECMDID_SETPROGRESSMAX constant 19518
SHDocVw.OLECMDID_SETPROGRESSPOS 19518
SHDocVw.OLECMDID_SETPROGRESSPOS constant 19518
SHDocVw.OLECMDID_SETPROGRESSTEXT 19519
SHDocVw.OLECMDID_SETPROGRESSTEXT constant 19519
SHDocVw.OLECMDID_SETTITLE 19519
SHDocVw.OLECMDID_SETTITLE constant 19519
SHDocVw.OLECMDID_SHOWFIND 19519
SHDocVw.OLECMDID_SHOWFIND constant 19519
SHDocVw.OLECMDID_SHOWMESSAGE 19520
SHDocVw.OLECMDID_SHOWMESSAGE constant 19520
SHDocVw.OLECMDID_SHOWPAGESETUP 19520
SHDocVw.OLECMDID_SHOWPAGESETUP constant 19520
SHDocVw.OLECMDID_SHOWPRINT 19520
SHDocVw.OLECMDID_SHOWPRINT constant 19520
SHDocVw.OLECMDID_SHOWSCRIPTERROR 19521
SHDocVw.OLECMDID_SHOWSCRIPTERROR constant 19521
SHDocVw.OLECMDID_SPELL 19521
SHDocVw.OLECMDID_SPELL constant 19521
SHDocVw.OLECMDID_STOP 19521
SHDocVw.OLECMDID_STOP constant 19521
SHDocVw.OLECMDID_STOPDOWNLOAD 19522
SHDocVw.OLECMDID_STOPDOWNLOAD constant 19522
SHDocVw.OLECMDID_UNDO 19522
SHDocVw.OLECMDID_UNDO constant 19522
SHDocVw.OLECMDID_UPDATECOMMANDS 19522
SHDocVw.OLECMDID_UPDATECOMMANDS constant 19522
SHDocVw.OLECMDID_UPDATEPAGESTATUS 19523
SHDocVw.OLECMDID_UPDATEPAGESTATUS constant 19523
SHDocVw.OLECMDID_ZOOM 19523
SHDocVw.OLECMDID_ZOOM constant 19523
SHDocVw.POleVariant1 19456
SHDocVw.POleVariant1 type 19456
SHDocVw.READYSTATE_COMPLETE 19523
SHDocVw.READYSTATE_COMPLETE constant 19523
SHDocVw.READYSTATE_INTERACTIVE 19524
SHDocVw.READYSTATE_INTERACTIVE constant 19524
SHDocVw.READYSTATE_LOADED 19524
SHDocVw.READYSTATE_LOADED constant 19524
SHDocVw.READYSTATE_LOADING 19524

SHDocVw.READYSTATE_LOADING constant 19524
SHDocVw.READYSTATE_UNINITIALIZED 19525
SHDocVw.READYSTATE_UNINITIALIZED constant 19525
SHDocVw.REFRESH_COMPLETELY 19525
SHDocVw.REFRESH_COMPLETELY constant 19525
SHDocVw.REFRESH_CONTINUE 19525
SHDocVw.REFRESH_CONTINUE constant 19525
SHDocVw.REFRESH_IFEXPIRED 19526
SHDocVw.REFRESH_IFEXPIRED constant 19526
SHDocVw.REFRESH_NORMAL 19526
SHDocVw.REFRESH_NORMAL constant 19526
SHDocVw.RefreshConstants 19456
SHDocVw.RefreshConstants type 19456
SHDocVw.SearchAssistantOC 19456
SHDocVw.SearchAssistantOC type 19456
SHDocVw.SecureLockIconConstants 19457
SHDocVw.SecureLockIconConstants type 19457
SHDocVw.secureLockIconMixed 19526
SHDocVw.secureLockIconMixed constant 19526
SHDocVw.secureLockIconSecure128Bit 19527
SHDocVw.secureLockIconSecure128Bit constant 19527
SHDocVw.secureLockIconSecure40Bit 19527
SHDocVw.secureLockIconSecure40Bit constant 19527
SHDocVw.secureLockIconSecure56Bit 19527
SHDocVw.secureLockIconSecure56Bit constant 19527
SHDocVw.secureLockIconSecureFortezza 19528
SHDocVw.secureLockIconSecureFortezza constant 19528
SHDocVw.secureLockIconSecureUnknownBits 19528
SHDocVw.secureLockIconSecureUnknownBits constant 19528
SHDocVw.secureLockIconUnsecure 19528
SHDocVw.secureLockIconUnsecure constant 19528
SHDocVw.SHDocVwMajorVersion 19529
SHDocVw.SHDocVwMajorVersion constant 19529
SHDocVw.SHDocVwMinorVersion 19529
SHDocVw.SHDocVwMinorVersion constant 19529
SHDocVw.ShellBrowserWindow 19457
SHDocVw.ShellBrowserWindow type 19457
SHDocVw.ShellNameSpace 19457
SHDocVw.ShellNameSpace type 19457
SHDocVw.ShellUIHelper 19458
SHDocVw.ShellUIHelper type 19458
SHDocVw.ShellWindowFindWindowOptions 19458
SHDocVw.ShellWindowFindWindowOptions type 19458
SHDocVw.ShellWindows 19458
SHDocVw.ShellWindows type 19458
SHDocVw.ShellWindowTypeConstants 19459
SHDocVw.ShellWindowTypeConstants type 19459
SHDocVw.SWC_3RDPARTY 19529
SHDocVw.SWC_3RDPARTY constant 19529
SHDocVw.SWC_BROWSER 19530
SHDocVw.SWC_BROWSER constant 19530
SHDocVw.SWC_CALLBACK 19530
SHDocVw.SWC_CALLBACK constant 19530
SHDocVw.SWC_EXPLORER 19530
SHDocVw.SWC_EXPLORER constant 19530
SHDocVw.SWFO_COOKIEPASSED 19531
SHDocVw.SWFO_COOKIEPASSED constant 19531
SHDocVw.SWFO_INCLUDEPENDING 19531
SHDocVw.SWFO_INCLUDEPENDING constant 19531
SHDocVw.SWFO_NEEDDISPATCH 19531
SHDocVw.SWFO_NEEDDISPATCH constant 19531
SHDocVw.tagREADYSTATE 19459
SHDocVw.tagREADYSTATE type 19459
SHDocVw.TInternetExplorer 19053
SHDocVw.TInternetExplorer.AddressBar 19084
SHDocVw.TInternetExplorer.Application 19084
SHDocVw.TInternetExplorer.Busy 19084
SHDocVw.TInternetExplorer.ClientToWindow 19062
SHDocVw.TInternetExplorer.Connect 19062
SHDocVw.TInternetExplorer.ConnectTo 19063
SHDocVw.TInternetExplorer.Container 19084
SHDocVw.TInternetExplorer.Create 19059
SHDocVw.TInternetExplorer.DefaultInterface 19085
SHDocVw.TInternetExplorer.Destroy 19060
SHDocVw.TInternetExplorer.Disconnect 19063
SHDocVw.TInternetExplorer.Document 19085
SHDocVw.TInternetExplorer.ExecWB 19064
SHDocVw.TInternetExplorer.FullName 19085

SHDocVw.TInternetExplorer.FullScreen 19085
SHDocVw.TInternetExplorer.Get_AddressBar 19065
SHDocVw.TInternetExplorer.Get_Application 19065
SHDocVw.TInternetExplorer.Get_Busy 19065
SHDocVw.TInternetExplorer.Get_Container 19065
SHDocVw.TInternetExplorer.Get_Document 19065
SHDocVw.TInternetExplorer.Get_FullName 19066
SHDocVw.TInternetExplorer.Get_FullScreen 19066
SHDocVw.TInternetExplorer.Get_Height 19066
SHDocVw.TInternetExplorer.Get_HWnd 19066
SHDocVw.TInternetExplorer.Get_Left 19067
SHDocVw.TInternetExplorer.Get_LocationName 19067
SHDocVw.TInternetExplorer.Get_LocationURL 19067
SHDocVw.TInternetExplorer.Get_MenuBar 19067
SHDocVw.TInternetExplorer.Get_Name 19067
SHDocVw.TInternetExplorer.Get_Offline 19068
SHDocVw.TInternetExplorer.Get_Parent 19068
SHDocVw.TInternetExplorer.Get_Path 19068
SHDocVw.TInternetExplorer.Get_ReadyState 19068
SHDocVw.TInternetExplorer.Get_RegisterAsBrowser 19069
SHDocVw.TInternetExplorer.Get_RegisterAsDropTarget 19069
SHDocVw.TInternetExplorer.Get_Resizable 19069
SHDocVw.TInternetExplorer.Get_Silent 19069
SHDocVw.TInternetExplorer.Get_StatusBar 19069
SHDocVw.TInternetExplorer.Get_StatusText 19070
SHDocVw.TInternetExplorer.Get_TheaterMode 19070
SHDocVw.TInternetExplorer.Get_ToolBar 19070
SHDocVw.TInternetExplorer.Get_Top 19070
SHDocVw.TInternetExplorer.Get_TopLevelContainer 19071
SHDocVw.TInternetExplorer.Get_type_ 19071
SHDocVw.TInternetExplorer.Get_Visible 19071
SHDocVw.TInternetExplorer.Get_Width 19071
SHDocVw.TInternetExplorer.GetProperty 19071
SHDocVw.TInternetExplorer.GoBack 19072
SHDocVw.TInternetExplorer.GoForward 19072
SHDocVw.TInternetExplorer.GoHome 19072
SHDocVw.TInternetExplorer.GoSearch 19072
SHDocVw.TInternetExplorer.Height 19086
SHDocVw.TInternetExplorer.HWnd 19086
SHDocVw.TInternetExplorer.InitServerData 19072
SHDocVw.TInternetExplorer.InvokeEvent 19073
SHDocVw.TInternetExplorer.Left 19086
SHDocVw.TInternetExplorer.LocationName 19086
SHDocVw.TInternetExplorer.LocationURL 19086
SHDocVw.TInternetExplorer.MenuBar 19087
SHDocVw.TInternetExplorer.Name 19087
SHDocVw.TInternetExplorer.Navigate 19074, 19075
SHDocVw.TInternetExplorer.Navigate2 19075, 19076
SHDocVw.TInternetExplorer.Offline 19089
SHDocVw.TInternetExplorer.OnBeforeNavigate2 19094
SHDocVw.TInternetExplorer.OnClientToHostWindow 19094
SHDocVw.TInternetExplorer.OnCommandStateChange 19094
SHDocVw.TInternetExplorer.OnDocumentComplete 19094
SHDocVw.TInternetExplorer.OnDownloadBegin 19094
SHDocVw.TInternetExplorer.OnDownloadComplete 19095
SHDocVw.TInternetExplorer.OnFileDownload 19095
SHDocVw.TInternetExplorer.OnFullScreen 19095
SHDocVw.TInternetExplorer.OnMenuBar 19095
SHDocVw.TInternetExplorer.OnNavigateComplete2 19096
SHDocVw.TInternetExplorer.OnNavigateError 19096
SHDocVw.TInternetExplorer.OnNewWindow2 19096
SHDocVw.TInternetExplorer.OnPrintTemplateInstantiation 19096
SHDocVw.TInternetExplorer.OnPrintTemplateTeardown 19096
SHDocVw.TInternetExplorer.OnPrivacyImpactedStateChange 19097
SHDocVw.TInternetExplorer.OnProgressChange 19097
SHDocVw.TInternetExplorer.OnPropertyChange 19097
SHDocVw.TInternetExplorer.OnQuit 19097
SHDocVw.TInternetExplorer.OnSetSecureLockIcon 19098
SHDocVw.TInternetExplorer.OnStatusBar 19098
SHDocVw.TInternetExplorer.OnStatusTextChange 19098
SHDocVw.TInternetExplorer.OnTheaterMode 19098
SHDocVw.TInternetExplorer.OnTitleChange 19098
SHDocVw.TInternetExplorer.OnToolBar 19099
SHDocVw.TInternetExplorer.OnUpdatePageStatus 19099
SHDocVw.TInternetExplorer.OnVisible 19099
SHDocVw.TInternetExplorer.OnWindowClosing 19099
SHDocVw.TInternetExplorer.OnWindowSetHeight 19100

SHDocVw.TInternetExplorer.OnWindowSetLeft 19100
SHDocVw.TInternetExplorer.OnWindowSetResizable 19100
SHDocVw.TInternetExplorer.OnWindowSetTop 19100
SHDocVw.TInternetExplorer.OnWindowSetWidth 19100
SHDocVw.TInternetExplorer.Parent 19089
SHDocVw.TInternetExplorer.Path 19089
SHDocVw.TInternetExplorer.PutProperty 19077
SHDocVw.TInternetExplorer.QueryStatusWB 19077
SHDocVw.TInternetExplorer.Quit 19077
SHDocVw.TInternetExplorer.ReadyState 19090
SHDocVw.TInternetExplorer.Refresh 19077
SHDocVw.TInternetExplorer.Refresh2 19078
SHDocVw.TInternetExplorer.RegisterAsBrowser 19090
SHDocVw.TInternetExplorer.RegisterAsDropTarget 19090
SHDocVw.TInternetExplorer.Resizable 19090
SHDocVw.TInternetExplorer.Set_AddressBar 19078
SHDocVw.TInternetExplorer.Set_FullScreen 19078
SHDocVw.TInternetExplorer.Set_Height 19079
SHDocVw.TInternetExplorer.Set_Left 19079
SHDocVw.TInternetExplorer.Set_MenuBar 19079
SHDocVw.TInternetExplorer.Set_Offline 19079
SHDocVw.TInternetExplorer.Set_RegisterAsBrowser 19079
SHDocVw.TInternetExplorer.Set_RegisterAsDropTarget 19080
SHDocVw.TInternetExplorer.Set_Resizable 19080
SHDocVw.TInternetExplorer.Set_Silent 19080
SHDocVw.TInternetExplorer.Set_StatusBar 19080
SHDocVw.TInternetExplorer.Set_StatusText 19081
SHDocVw.TInternetExplorer.Set_TheaterMode 19081
SHDocVw.TInternetExplorer.Set_ToolBar 19081
SHDocVw.TInternetExplorer.Set_Top 19081
SHDocVw.TInternetExplorer.Set_Visible 19081
SHDocVw.TInternetExplorer.Set_Width 19082
SHDocVw.TInternetExplorer.ShowBrowserBar 19082, 19083
SHDocVw.TInternetExplorer.Silent 19091
SHDocVw.TInternetExplorer.StatusBar 19091
SHDocVw.TInternetExplorer.StatusText 19091
SHDocVw.TInternetExplorer.Stop 19083
SHDocVw.TInternetExplorer.TheaterMode 19091
SHDocVw.TInternetExplorer.ToolBar 19091
SHDocVw.TInternetExplorer.Top 19092
SHDocVw.TInternetExplorer.TopLevelContainer 19092
SHDocVw.TInternetExplorer.type_ 19092
SHDocVw.TInternetExplorer.Visible 19092
SHDocVw.TInternetExplorer.Width 19093
SHDocVw.TInternetExplorerBeforeNavigate2 19459
SHDocVw.TInternetExplorerBeforeNavigate2 type 19459
SHDocVw.TInternetExplorerClientToHostWindow 19460
SHDocVw.TInternetExplorerClientToHostWindow type 19460
SHDocVw.TInternetExplorerCommandStateChange 19460
SHDocVw.TInternetExplorerCommandStateChange type 19460
SHDocVw.TInternetExplorerDocumentComplete 19460
SHDocVw.TInternetExplorerDocumentComplete type 19460
SHDocVw.TInternetExplorerFileDownload 19461
SHDocVw.TInternetExplorerFileDownload type 19461
SHDocVw.TInternetExplorerNavigateComplete2 19461
SHDocVw.TInternetExplorerNavigateComplete2 type 19461
SHDocVw.TInternetExplorerNavigateError 19461
SHDocVw.TInternetExplorerNavigateError type 19461
SHDocVw.TInternetExplorerNewWindow2 19462
SHDocVw.TInternetExplorerNewWindow2 type 19462
SHDocVw.TInternetExplorerOnFullScreen 19462
SHDocVw.TInternetExplorerOnFullScreen type 19462
SHDocVw.TInternetExplorerOnMenuBar 19462
SHDocVw.TInternetExplorerOnMenuBar type 19462
SHDocVw.TInternetExplorerOnStatusBar 19463
SHDocVw.TInternetExplorerOnStatusBar type 19463
SHDocVw.TInternetExplorerOnTheaterMode 19463
SHDocVw.TInternetExplorerOnTheaterMode type 19463
SHDocVw.TInternetExplorerOnToolBar 19463
SHDocVw.TInternetExplorerOnToolBar type 19463
SHDocVw.TInternetExplorerOnVisible 19464
SHDocVw.TInternetExplorerOnVisible type 19464
SHDocVw.TInternetExplorerPrintTemplateInstantiation 19464
SHDocVw.TInternetExplorerPrintTemplateInstantiation type 19464
SHDocVw.TInternetExplorerPrintTemplateTeardown 19464
SHDocVw.TInternetExplorerPrintTemplateTeardown type 19464
SHDocVw.TInternetExplorerPrivacyImpactedStateChange 19465

SHDocVw.TInternetExplorerPrivacyImpactedStateChange type 19465
SHDocVw.TInternetExplorerProgressChange 19465
SHDocVw.TInternetExplorerProgressChange type 19465
SHDocVw.TInternetExplorerPropertyChange 19465
SHDocVw.TInternetExplorerPropertyChange type 19465
SHDocVw.TInternetExplorerSetSecureLockIcon 19466
SHDocVw.TInternetExplorerSetSecureLockIcon type 19466
SHDocVw.TInternetExplorerStatusTextChange 19466
SHDocVw.TInternetExplorerStatusTextChange type 19466
SHDocVw.TInternetExplorerTitleChange 19466
SHDocVw.TInternetExplorerTitleChange type 19466
SHDocVw.TInternetExplorerUpdatePageStatus 19467
SHDocVw.TInternetExplorerUpdatePageStatus type 19467
SHDocVw.TInternetExplorerWindowClosing 19467
SHDocVw.TInternetExplorerWindowClosing type 19467
SHDocVw.TInternetExplorerWindowSetHeight 19467
SHDocVw.TInternetExplorerWindowSetHeight type 19467
SHDocVw.TInternetExplorerWindowSetLeft 19468
SHDocVw.TInternetExplorerWindowSetLeft type 19468
SHDocVw.TInternetExplorerWindowSetResizable 19468
SHDocVw.TInternetExplorerWindowSetResizable type 19468
SHDocVw.TInternetExplorerWindowSetTop 19468
SHDocVw.TInternetExplorerWindowSetTop type 19468
SHDocVw.TInternetExplorerWindowSetWidth 19469
SHDocVw.TInternetExplorerWindowSetWidth type 19469
SHDocVw.TShellNameSpace 19101
SHDocVw.TShellNameSpace.Columns 19117
SHDocVw.TShellNameSpace.Connect 19107
SHDocVw.TShellNameSpace.ConnectTo 19108
SHDocVw.TShellNameSpace.CountViewTypes 19118
SHDocVw.TShellNameSpace.Create 19105
SHDocVw.TShellNameSpace.CreateSubscriptionForSelection 19108
SHDocVw.TShellNameSpace.DefaultInterface 19118
SHDocVw.TShellNameSpace.DeleteSubscriptionForSelection 19108
SHDocVw.TShellNameSpace.Depth 19118
SHDocVw.TShellNameSpace.Destroy 19106
SHDocVw.TShellNameSpace.Disconnect 19109
SHDocVw.TShellNameSpace.EnumOptions 19118
SHDocVw.TShellNameSpace.Expand 19109
SHDocVw.TShellNameSpace.Export 19109
SHDocVw.TShellNameSpace.Flags 19118
SHDocVw.TShellNameSpace.Get_Columns 19110
SHDocVw.TShellNameSpace.Get_CountViewTypes 19110
SHDocVw.TShellNameSpace.Get_Depth 19110
SHDocVw.TShellNameSpace.Get_EnumOptions 19110
SHDocVw.TShellNameSpace.Get_Flags 19110
SHDocVw.TShellNameSpace.Get_Mode 19111
SHDocVw.TShellNameSpace.Get_Root 19111
SHDocVw.TShellNameSpace.Get_SelectedItem 19111
SHDocVw.TShellNameSpace.Get_SubscriptionsEnabled 19111
SHDocVw.TShellNameSpace.Get_TVFlags 19112
SHDocVw.TShellNameSpace.Import 19112
SHDocVw.TShellNameSpace.InitServerData 19112
SHDocVw.TShellNameSpace.InvokeContextMenuCommand 19112
SHDocVw.TShellNameSpace.InvokeEvent 19113
SHDocVw.TShellNameSpace.Mode 19119
SHDocVw.TShellNameSpace.MoveSelectionDown 19113
SHDocVw.TShellNameSpace.MoveSelectionTo 19113
SHDocVw.TShellNameSpace.MoveSelectionUp 19113
SHDocVw.TShellNameSpace.NewFolder 19114
SHDocVw.TShellNameSpace.OnDoubleClick 19120
SHDocVw.TShellNameSpace.OnFavoritesSelectionChange 19120
SHDocVw.TShellNameSpace.OnInitialized 19120
SHDocVw.TShellNameSpace.OnSelectionChange 19121
SHDocVw.TShellNameSpace.ResetSort 19114
SHDocVw.TShellNameSpace.Root 19119
SHDocVw.TShellNameSpace.SelectedItem 19119
SHDocVw.TShellNameSpace.SelectedItems 19114
SHDocVw.TShellNameSpace.Set_Columns 19114
SHDocVw.TShellNameSpace.Set_Depth 19115
SHDocVw.TShellNameSpace.Set_EnumOptions 19115
SHDocVw.TShellNameSpace.Set_Flags 19115
SHDocVw.TShellNameSpace.Set_Mode 19115
SHDocVw.TShellNameSpace.Set_Root 19115
SHDocVw.TShellNameSpace.Set_SelectedItem 19116

SHDocVw.TShellNameSpace.Set_TVFlags 19116
SHDocVw.TShellNameSpace.SetRoot 19116
SHDocVw.TShellNameSpace.SetViewType 19116
SHDocVw.TShellNameSpace.SubscriptionsEnabled 19119
SHDocVw.TShellNameSpace.Synchronize 19117
SHDocVw.TShellNameSpace.TVFlags 19120
SHDocVw.TShellNameSpace.UnselectAll 19117
SHDocVw.TShellNameSpaceFavoritesSelectionChange 19469
SHDocVw.TShellNameSpaceFavoritesSelectionChange type 19469
SHDocVw.TShellUIHelper 19121
SHDocVw.TShellUIHelper.AddChannel 19127
SHDocVw.TShellUIHelper.AddDesktopComponent 19127, 19128
SHDocVw.TShellUIHelper.AddFavorite 19129
SHDocVw.TShellUIHelper.AutoCompleteAttach 19129, 19130
SHDocVw.TShellUIHelper.AutoCompleteSaveForm 19130
SHDocVw.TShellUIHelper.AutoScan 19131
SHDocVw.TShellUIHelper.Connect 19131
SHDocVw.TShellUIHelper.ConnectTo 19132
SHDocVw.TShellUIHelper.Create 19125
SHDocVw.TShellUIHelper.DefaultInterface 19135
SHDocVw.TShellUIHelper.Destroy 19125
SHDocVw.TShellUIHelper.Disconnect 19132
SHDocVw.TShellUIHelper.ImportExportFavorites 19132
SHDocVw.TShellUIHelper.InitServerData 19133
SHDocVw.TShellUIHelper.IsSubscribed 19133
SHDocVw.TShellUIHelper.NavigateAndFind 19133
SHDocVw.TShellUIHelper.RefreshOfflineDesktop 19133
SHDocVw.TShellUIHelper.ResetFirstBootMode 19134
SHDocVw.TShellUIHelper.ResetSafeMode 19134
SHDocVw.TShellUIHelper.ShowBrowserUI 19134
SHDocVw.TShellWindows 19135
SHDocVw.TShellWindows._NewEnum 19141
SHDocVw.TShellWindows.Connect 19141
SHDocVw.TShellWindows.ConnectTo 19141
SHDocVw.TShellWindows.Count 19146
SHDocVw.TShellWindows.Create 19139
SHDocVw.TShellWindows.DefaultInterface 19146
SHDocVw.TShellWindows.Destroy 19139
SHDocVw.TShellWindows.Disconnect 19142
SHDocVw.TShellWindows.FindWindowSW 19142
SHDocVw.TShellWindows.Get_Count 19142
SHDocVw.TShellWindows.InitServerData 19142
SHDocVw.TShellWindows.InvokeEvent 19143
SHDocVw.TShellWindows.Item 19144
SHDocVw.TShellWindows.OnActivated 19144
SHDocVw.TShellWindows.OnCreated 19144
SHDocVw.TShellWindows.OnNavigate 19144
SHDocVw.TShellWindows.OnWindowRegistered 19146
SHDocVw.TShellWindows.OnWindowRevoked 19147
SHDocVw.TShellWindows.ProcessAttachDetach 19145
SHDocVw.TShellWindows.Register 19145
SHDocVw.TShellWindows.RegisterPending 19145
SHDocVw.TShellWindows.Revoke 19145
SHDocVw.TShellWindowsWindowRegistered 19470
SHDocVw.TShellWindowsWindowRegistered type 19470
SHDocVw.TShellWindowsWindowRevoked 19470
SHDocVw.TShellWindowsWindowRevoked type 19470
SHDocVw.TWebBrowser 19147
SHDocVw.TWebBrowser.AddressBar 19175
SHDocVw.TWebBrowser.Align 19175
SHDocVw.TWebBrowser.Anchors 19179
SHDocVw.TWebBrowser.Application 19179
SHDocVw.TWebBrowser.Busy 19180
SHDocVw.TWebBrowser.ClientToWindow 19163
SHDocVw.TWebBrowser.Container 19180
SHDocVw.TWebBrowser.ControlInterface 19181
SHDocVw.TWebBrowser.CreateControl 19163
SHDocVw.TWebBrowser.DefaultInterface 19181
SHDocVw.TWebBrowser.Document 19182
SHDocVw.TWebBrowser.DragCursor 19182
SHDocVw.TWebBrowser.DragMode 19182
SHDocVw.TWebBrowser.ExecWB 19163
SHDocVw.TWebBrowser.FullName 19184
SHDocVw.TWebBrowser.FullScreen 19184
SHDocVw.TWebBrowser.Get_Application 19164
SHDocVw.TWebBrowser.Get_Container 19164
SHDocVw.TWebBrowser.Get_Document 19164

SHDocVw.TWebBrowser.Get_Parent 19165
SHDocVw.TWebBrowser.GetProperty 19165
SHDocVw.TWebBrowser.GoBack 19165
SHDocVw.TWebBrowser.GoForward 19166
SHDocVw.TWebBrowser.GoHome 19166
SHDocVw.TWebBrowser.GoSearch 19167
SHDocVw.TWebBrowser.HWnd 19185
SHDocVw.TWebBrowser.InitControlData 19167
SHDocVw.TWebBrowser.LocationName 19185
SHDocVw.TWebBrowser.LocationURL 19185
SHDocVw.TWebBrowser.MenuBar 19186
SHDocVw.TWebBrowser.Name 19186
SHDocVw.TWebBrowser.Navigate 19168
SHDocVw.TWebBrowser.Navigate2 19169
SHDocVw.TWebBrowser.Offline 19186
SHDocVw.TWebBrowser.OnBeforeNavigate2 19200
SHDocVw.TWebBrowser.OnClientToHostWindow 19200
SHDocVw.TWebBrowser.OnCommandStateChange 19201
SHDocVw.TWebBrowser.OnDocumentComplete 19201
SHDocVw.TWebBrowser.OnDownloadBegin 19202
SHDocVw.TWebBrowser.OnDownloadComplete 19202
SHDocVw.TWebBrowser.OnDragDrop 19203
SHDocVw.TWebBrowser.OnDragOver 19205
SHDocVw.TWebBrowser.OnEndDrag 19207
SHDocVw.TWebBrowser.OnEnter 19208
SHDocVw.TWebBrowser.OnExit 19211
SHDocVw.TWebBrowser.OnFileDownload 19212
SHDocVw.TWebBrowser.OnFullScreen 19213
SHDocVw.TWebBrowser.OnMenuBar 19213
SHDocVw.TWebBrowser.OnNavigateComplete2 19213
SHDocVw.TWebBrowser.OnNavigateError 19214
SHDocVw.TWebBrowser.OnNewWindow2 19214
SHDocVw.TWebBrowser.OnPrintTemplateInstantiation 19214
SHDocVw.TWebBrowser.OnPrintTemplateTeardown 19215
SHDocVw.TWebBrowser.OnPrivacyImpactedStateChange 19215
SHDocVw.TWebBrowser.OnProgressChange 19215
SHDocVw.TWebBrowser.OnPropertyChange 19215
SHDocVw.TWebBrowser.OnQuit 19216
SHDocVw.TWebBrowser.OnSetSecureLockIcon 19216
SHDocVw.TWebBrowser.OnStartDrag 19216
SHDocVw.TWebBrowser.OnStatusBar 19217
SHDocVw.TWebBrowser.OnStatusTextChange 19217
SHDocVw.TWebBrowser.OnTheaterMode 19217
SHDocVw.TWebBrowser.OnTitleChange 19218
SHDocVw.TWebBrowser.OnToolBar 19218
SHDocVw.TWebBrowser.OnUpdatePageStatus 19218
SHDocVw.TWebBrowser.OnVisible 19219
SHDocVw.TWebBrowser.OnWindowClosing 19219
SHDocVw.TWebBrowser.OnWindowSetHeight 19219
SHDocVw.TWebBrowser.OnWindowSetLeft 19219
SHDocVw.TWebBrowser.OnWindowSetResizable 19220
SHDocVw.TWebBrowser.OnWindowSetTop 19220
SHDocVw.TWebBrowser.OnWindowSetWidth 19220
SHDocVw.TWebBrowser.Parent 19187
SHDocVw.TWebBrowser.ParentShowHint 19187
SHDocVw.TWebBrowser.Path 19188
SHDocVw.TWebBrowser.PopupMenu 19188
SHDocVw.TWebBrowser.PutProperty 19171
SHDocVw.TWebBrowser.QueryStatusWB 19171
SHDocVw.TWebBrowser.Quit 19171
SHDocVw.TWebBrowser.ReadyState 19193
SHDocVw.TWebBrowser.Refresh 19172
SHDocVw.TWebBrowser.Refresh2 19172
SHDocVw.TWebBrowser.RegisterAsBrowser 19193
SHDocVw.TWebBrowser.RegisterAsDropTarget 19194
SHDocVw.TWebBrowser.Resizable 19194
SHDocVw.TWebBrowser.ShowBrowserBar 19173
SHDocVw.TWebBrowser.ShowHint 19194
SHDocVw.TWebBrowser.Silent 19196
SHDocVw.TWebBrowser.StatusBar 19196
SHDocVw.TWebBrowser.StatusText 19196
SHDocVw.TWebBrowser.Stop 19174
SHDocVw.TWebBrowser.TabOrder 19197
SHDocVw.TWebBrowser.TabStop 19197
SHDocVw.TWebBrowser.TheaterMode 19198
SHDocVw.TWebBrowser.ToolBar 19198
SHDocVw.TWebBrowser.TopLevelContainer 19198
SHDocVw.TWebBrowser.type_ 19199

SHDocVw.TWebBrowser.Visible 19199
SHDocVw.TWebBrowser_V1 19221
SHDocVw.TWebBrowser_V1.Align 19241
SHDocVw.TWebBrowser_V1.Anchors 19245
SHDocVw.TWebBrowser_V1.Application 19245
SHDocVw.TWebBrowser_V1.Busy 19245
SHDocVw.TWebBrowser_V1.Container 19246
SHDocVw.TWebBrowser_V1.ControlInterface 19246
SHDocVw.TWebBrowser_V1.CreateControl 19235
SHDocVw.TWebBrowser_V1.DefaultInterface 19246
SHDocVw.TWebBrowser_V1.Document 19246
SHDocVw.TWebBrowser_V1.DragCursor 19246
SHDocVw.TWebBrowser_V1.DragMode 19247
SHDocVw.TWebBrowser_V1.Get_Application 19236
SHDocVw.TWebBrowser_V1.Get_Container 19236
SHDocVw.TWebBrowser_V1.Get_Document 19236
SHDocVw.TWebBrowser_V1.Get_Parent 19236
SHDocVw.TWebBrowser_V1.GoBack 19236
SHDocVw.TWebBrowser_V1.GoForward 19237
SHDocVw.TWebBrowser_V1.GoHome 19237
SHDocVw.TWebBrowser_V1.GoSearch 19237
SHDocVw.TWebBrowser_V1.InitControlData 19237
SHDocVw.TWebBrowser_V1.LocationName 19248
SHDocVw.TWebBrowser_V1.LocationURL 19249
SHDocVw.TWebBrowser_V1.Navigate 19238, 19239
SHDocVw.TWebBrowser_V1.OnBeforeNavigate 19261
SHDocVw.TWebBrowser_V1.OnCommandStateChange 19261
SHDocVw.TWebBrowser_V1.OnDownloadBegin 19261
SHDocVw.TWebBrowser_V1.OnDownloadComplete 19261
SHDocVw.TWebBrowser_V1.OnDragDrop 19261
SHDocVw.TWebBrowser_V1.OnDragOver 19264
SHDocVw.TWebBrowser_V1.OnEndDrag 19266
SHDocVw.TWebBrowser_V1.OnEnter 19267
SHDocVw.TWebBrowser_V1.OnExit 19270
SHDocVw.TWebBrowser_V1.OnFrameBeforeNavigate 19271
SHDocVw.TWebBrowser_V1.OnFrameNavigateComplete 19271
SHDocVw.TWebBrowser_V1.OnFrameNewWindow 19272
SHDocVw.TWebBrowser_V1.OnNavigateComplete 19272
SHDocVw.TWebBrowser_V1.OnNewWindow 19272
SHDocVw.TWebBrowser_V1.OnProgressChange 19272
SHDocVw.TWebBrowser_V1.OnPropertyChange 19272
SHDocVw.TWebBrowser_V1.OnQuit 19273
SHDocVw.TWebBrowser_V1.OnStartDrag 19273
SHDocVw.TWebBrowser_V1.OnStatusTextChange 19274
SHDocVw.TWebBrowser_V1.OnTitleChange 19274
SHDocVw.TWebBrowser_V1.OnWindowActivate 19274
SHDocVw.TWebBrowser_V1.OnWindowMove 19274
SHDocVw.TWebBrowser_V1.OnWindowResize 19274
SHDocVw.TWebBrowser_V1.Parent 19249
SHDocVw.TWebBrowser_V1.ParentShowHint 19250
SHDocVw.TWebBrowser_V1.PopupMenu 19251
SHDocVw.TWebBrowser_V1.Refresh 19239
SHDocVw.TWebBrowser_V1.Refresh2 19240
SHDocVw.TWebBrowser_V1.ShowHint 19256
SHDocVw.TWebBrowser_V1.Stop 19240
SHDocVw.TWebBrowser_V1.TabOrder 19257
SHDocVw.TWebBrowser_V1.TabStop 19258
SHDocVw.TWebBrowser_V1.TopLevelContainer 19258
SHDocVw.TWebBrowser_V1.type_ 19258
SHDocVw.TWebBrowser_V1.Visible 19258
SHDocVw.TWebBrowser_V1BeforeNavigate 19470
SHDocVw.TWebBrowser_V1BeforeNavigate type 19470
SHDocVw.TWebBrowser_V1CommandStateChange 19471
SHDocVw.TWebBrowser_V1CommandStateChange type 19471
SHDocVw.TWebBrowser_V1FrameBeforeNavigate 19471
SHDocVw.TWebBrowser_V1FrameBeforeNavigate type 19471
SHDocVw.TWebBrowser_V1FrameNavigateComplete 19471
SHDocVw.TWebBrowser_V1FrameNavigateComplete type 19471
SHDocVw.TWebBrowser_V1FrameNewWindow 19472
SHDocVw.TWebBrowser_V1FrameNewWindow type 19472
SHDocVw.TWebBrowser_V1NavigateComplete 19472
SHDocVw.TWebBrowser_V1NavigateComplete type 19472
SHDocVw.TWebBrowser_V1NewWindow 19472
SHDocVw.TWebBrowser_V1NewWindow type 19472
SHDocVw.TWebBrowser_V1ProgressChange 19473
SHDocVw.TWebBrowser_V1ProgressChange type 19473
SHDocVw.TWebBrowser_V1PropertyChange 19473
SHDocVw.TWebBrowser_V1PropertyChange type 19473

SHDocVw.TWebBrowser_V1Quit 19474
SHDocVw.TWebBrowser_V1Quit type 19474
SHDocVw.TWebBrowser_V1StatusTextChange 19474
SHDocVw.TWebBrowser_V1StatusTextChange type 19474
SHDocVw.TWebBrowser_V1TitleChange 19474
SHDocVw.TWebBrowser_V1TitleChange type 19474
SHDocVw.TWebBrowserBeforeNavigate2 19475
SHDocVw.TWebBrowserBeforeNavigate2 type 19475
SHDocVw.TWebBrowserClientToHostWindow 19475
SHDocVw.TWebBrowserClientToHostWindow type 19475
SHDocVw.TWebBrowserCommandStateChange 19475
SHDocVw.TWebBrowserCommandStateChange type 19475
SHDocVw.TWebBrowserDocumentComplete 19476
SHDocVw.TWebBrowserDocumentComplete type 19476
SHDocVw.TWebBrowserFileDownload 19476
SHDocVw.TWebBrowserFileDownload type 19476
SHDocVw.TWebBrowserNavigateComplete2 19476
SHDocVw.TWebBrowserNavigateComplete2 type 19476
SHDocVw.TWebBrowserNavigateError 19477
SHDocVw.TWebBrowserNavigateError type 19477
SHDocVw.TWebBrowserNewWindow2 19477
SHDocVw.TWebBrowserNewWindow2 type 19477
SHDocVw.TWebBrowserOnFullScreen 19477
SHDocVw.TWebBrowserOnFullScreen type 19477
SHDocVw.TWebBrowserOnMenuBar 19478
SHDocVw.TWebBrowserOnMenuBar type 19478
SHDocVw.TWebBrowserOnStatusBar 19478
SHDocVw.TWebBrowserOnStatusBar type 19478
SHDocVw.TWebBrowserOnTheaterMode 19478
SHDocVw.TWebBrowserOnTheaterMode type 19478
SHDocVw.TWebBrowserOnToolBar 19479
SHDocVw.TWebBrowserOnToolBar type 19479
SHDocVw.TWebBrowserOnVisible 19479
SHDocVw.TWebBrowserOnVisible type 19479
SHDocVw.TWebBrowserPrintTemplateInstantiation 19479
SHDocVw.TWebBrowserPrintTemplateInstantiation type 19479
SHDocVw.TWebBrowserPrintTemplateTeardown 19480
SHDocVw.TWebBrowserPrintTemplateTeardown type 19480
SHDocVw.TWebBrowserPrivacyImpactedStateChange 19480
SHDocVw.TWebBrowserPrivacyImpactedStateChange type 19480
SHDocVw.TWebBrowserProgressChange 19480
SHDocVw.TWebBrowserProgressChange type 19480
SHDocVw.TWebBrowserPropertyChange 19481
SHDocVw.TWebBrowserPropertyChange type 19481
SHDocVw.TWebBrowserSetSecureLockIcon 19481
SHDocVw.TWebBrowserSetSecureLockIcon type 19481
SHDocVw.TWebBrowserStatusTextChange 19481
SHDocVw.TWebBrowserStatusTextChange type 19481
SHDocVw.TWebBrowserTitleChange 19482
SHDocVw.TWebBrowserTitleChange type 19482
SHDocVw.TWebBrowserUpdatePageStatus 19482
SHDocVw.TWebBrowserUpdatePageStatus type 19482
SHDocVw.TWebBrowserWindowClosing 19482
SHDocVw.TWebBrowserWindowClosing type 19482
SHDocVw.TWebBrowserWindowSetHeight 19483
SHDocVw.TWebBrowserWindowSetHeight type 19483
SHDocVw.TWebBrowserWindowSetLeft 19483
SHDocVw.TWebBrowserWindowSetLeft type 19483
SHDocVw.TWebBrowserWindowSetResizable 19483
SHDocVw.TWebBrowserWindowSetResizable type 19483
SHDocVw.TWebBrowserWindowSetTop 19484
SHDocVw.TWebBrowserWindowSetTop type 19484
SHDocVw.TWebBrowserWindowSetWidth 19484
SHDocVw.TWebBrowserWindowSetWidth type 19484
SHDocVw.WebBrowser 19484
SHDocVw.WebBrowser type 19484
SHDocVw.WebBrowser_V1 19485
SHDocVw.WebBrowser_V1 type 19485
ShellAnimations 19533
ShellAnimations namespace 19533
    Classes 19534
ShellAnimations.TShellResources 19534
siAsserts enumeration member 1305
siCatalogs enumeration member 1305
siCharacterSets enumeration member 1305
siCheckConstraints enumeration member 1305
siCollations enumeration member 1305
siColumnPrivileges enumeration member 1305

siColumns enumeration member 1305
siColumnsDomainUsage enumeration member 1305
siConstraintColumnUsage enumeration member 1305
siConstraintTableUsage enumeration member 1305
siCubes enumeration member 1305
siDBInfoKeywords enumeration member 1305
siDBInfoLiterals enumeration member 1305
siDimensions enumeration member 1305
siForeignKeys enumeration member 1305
siHierarchies enumeration member 1305
siIndexes enumeration member 1305
siKeyColumnUsage enumeration member 1305
siLevels enumeration member 1305
siMeasures enumeration member 1305
siMembers enumeration member 1305
SimpleDS 19537
SimpleDS namespace 19537
Classes 19538
SimpleDS.TInternalSQLDataSet 19538
SimpleDS.TInternalSQLDataSet.CommandText 19550
SimpleDS.TInternalSQLDataSet.CommandType 19551
SimpleDS.TInternalSQLDataSet.DataSource 19552
SimpleDS.TInternalSQLDataSet.GetMetadata 19552
SimpleDS.TInternalSQLDataSet.MaxBlobSize 19553
SimpleDS.TInternalSQLDataSet.ParamCheck 19553
SimpleDS.TInternalSQLDataSet.Params 19554
SimpleDS.TInternalSQLDataSet.SortFieldNames 19555
SimpleDS.TSimpleDataSet 19555
SimpleDS.TSimpleDataSet.Active 19575
SimpleDS.TSimpleDataSet.AfterApplyUpdates 19598
SimpleDS.TSimpleDataSet.AfterCancel 19599
SimpleDS.TSimpleDataSet.AfterClose 19601
SimpleDS.TSimpleDataSet.AfterDelete 19602
SimpleDS.TSimpleDataSet.AfterEdit 19605
SimpleDS.TSimpleDataSet.AfterExecute 19606
SimpleDS.TSimpleDataSet.AfterGetParams 19606
SimpleDS.TSimpleDataSet.AfterGetRecords 19607
SimpleDS.TSimpleDataSet.AfterInsert 19608
SimpleDS.TSimpleDataSet.AfterOpen 19609
SimpleDS.TSimpleDataSet.AfterPost 19610
SimpleDS.TSimpleDataSet.AfterRefresh 19610
SimpleDS.TSimpleDataSet.AfterRowRequest 19611
SimpleDS.TSimpleDataSet.AfterScroll 19612
SimpleDS.TSimpleDataSet.Aggregates 19577
SimpleDS.TSimpleDataSet.AggregatesActive 19578
SimpleDS.TSimpleDataSet.AllocConnection 19571
SimpleDS.TSimpleDataSet.AllocDataSet 19571
SimpleDS.TSimpleDataSet.AllocProvider 19572
SimpleDS.TSimpleDataSet.AutoCalcFields 19578
SimpleDS.TSimpleDataSet.BeforeApplyUpdates 19612
SimpleDS.TSimpleDataSet.BeforeCancel 19613
SimpleDS.TSimpleDataSet.BeforeClose 19613
SimpleDS.TSimpleDataSet.BeforeDelete 19614
SimpleDS.TSimpleDataSet.BeforeEdit 19614
SimpleDS.TSimpleDataSet.BeforeExecute 19615
SimpleDS.TSimpleDataSet.BeforeGetParams 19616
SimpleDS.TSimpleDataSet.BeforeGetRecords 19616
SimpleDS.TSimpleDataSet.BeforeInsert 19618
SimpleDS.TSimpleDataSet.BeforeOpen 19619
SimpleDS.TSimpleDataSet.BeforePost 19620
SimpleDS.TSimpleDataSet.BeforeRefresh 19620
SimpleDS.TSimpleDataSet.BeforeRowRequest 19621
SimpleDS.TSimpleDataSet.BeforeScroll 19622
SimpleDS.TSimpleDataSet.Connection 19579
SimpleDS.TSimpleDataSet.Constraints 19579
SimpleDS.TSimpleDataSet.Create 19570
SimpleDS.TSimpleDataSet.DataSet 19580
SimpleDS.TSimpleDataSet.Destroy 19571
SimpleDS.TSimpleDataSet.DisableStringTrim 19580
SimpleDS.TSimpleDataSet.FetchOnDemand 19581
SimpleDS.TSimpleDataSet.FieldDefs 19582
SimpleDS.TSimpleDataSet.FileName 19585
SimpleDS.TSimpleDataSet.Filter 19585
SimpleDS.TSimpleDataSet.Filtered 19587
SimpleDS.TSimpleDataSet.FilterOptions 19588
SimpleDS.TSimpleDataSet.IndexDefs 19588
SimpleDS.TSimpleDataSet.IndexFieldNames 19589
SimpleDS.TSimpleDataSet.IndexName 19590

wwww

SimpleDS.TSimpleDataSet.Loaded 19572
SimpleDS.TSimpleDataSet.MasterFields 19591
SimpleDS.TSimpleDataSet.MasterSource 19592
SimpleDS.TSimpleDataSet.Notification 19572
SimpleDS.TSimpleDataSet.ObjectView 19592
SimpleDS.TSimpleDataSet.OnCalcFields 19622
SimpleDS.TSimpleDataSet.OnDeleteError 19623
SimpleDS.TSimpleDataSet.OnEditError 19623
SimpleDS.TSimpleDataSet.OnFilterRecord 19624
SimpleDS.TSimpleDataSet.OnNewRecord 19625
SimpleDS.TSimpleDataSet.OnPostError 19625
SimpleDS.TSimpleDataSet.OnReconcileError 19625
SimpleDS.TSimpleDataSet.OpenCursor 19573
SimpleDS.TSimpleDataSet.PacketRecords 19593
SimpleDS.TSimpleDataSet.Params 19594
SimpleDS.TSimpleDataSet.PSGetCommandText 19573
SimpleDS.TSimpleDataSet.ReadOnly 19596
SimpleDS.TSimpleDataSet.SetConnection 19574
SimpleDS.TSimpleDataSet.StoreDefs 19597
siPrimaryKeys enumeration member 1305
siProcedureColumns enumeration member 1305
siProcedureParameters enumeration member 1305
siProcedures enumeration member 1305
siProperties enumeration member 1305
siProviderSpecific enumeration member 1305
siProviderTypes enumeration member 1305
siReferentialConstraints enumeration member 1305
siSchemata enumeration member 1305
siSQLLanguages enumeration member 1305
siStatistics enumeration member 1305
siTableConstraints enumeration member 1305
siTablePrivileges enumeration member 1305
siTables enumeration member 1305
SiteComp 19627
SiteComp namespace 19627
Classes 19631
Constants 19799
Functions 19788
Interfaces 19645
Structs, Records, Enums 19789
Types 19794
Variables 19796
SiteComp.AdapterActionHTMLElementTypeNames 19799
SiteComp.AdapterActionHTMLElementTypeNames constant 19799
SiteComp.AdapterDisplayHTMLElementTypeNames 19799
SiteComp.AdapterDisplayHTMLElementTypeNames constant 19799
SiteComp.AdapterDisplayViewModeTypeNames 19800
SiteComp.AdapterDisplayViewModeTypeNames constant 19800
SiteComp.AdapterInputHTMLElementTypeNames 19800
SiteComp.AdapterInputHTMLElementTypeNames constant 19800
SiteComp.ExecuteScript 19796
SiteComp.ExecuteScript variable 19796
SiteComp.FullyQualifiedFieldName 19788
SiteComp.FullyQualifiedFieldName function 19788
SiteComp.FullyQualifiedName 19788
SiteComp.FullyQualifiedName function 19788
SiteComp.GetHTMLSampleImage 19796
SiteComp.GetHTMLSampleImage variable 19796
SiteComp.IActionImageProducer 19646
SiteComp.IActionImageProducer.GetAdapterEventImage 19648
SiteComp.IActionImageProducer.GetDisplayStyle 19648
SiteComp.IAdapterAccess 19649
SiteComp.IAdapterAccess.HasExecuteAccess 19650
SiteComp.IAdapterAccess.HasModifyAccess 19650
SiteComp.IAdapterAccess.HasViewAccess 19650
SiteComp.IAdapterActionAccess 19650
SiteComp.IAdapterActionAccess.HasExecuteAccess 19651
SiteComp.IAdapterActionAttributes 19651
SiteComp.IAdapterActionAttributes.GetVisible 19652
SiteComp.IAdapterActionAttributes.Visible 19652
SiteComp.IAdapterDispatcher 19653
SiteComp.IAdapterDispatcher.EncodeAdapterItemRequestAsFieldValue 19654
SiteComp.IAdapterDispatcher.EncodeAdapterItemRequestAsHREF 19654
SiteComp.IAdapterDispatcher.GetAdapterItemRequestParams

19655
SiteComp.IAdapterDispatchParams 19655
SiteComp.IAdapterDispatchParams.GetHandler 19657
SiteComp.IAdapterDispatchParams.GetObjectIdentifier 19657
SiteComp.IAdapterDispatchParams.GetParams 19657
SiteComp.IAdapterDispatchParams.GetRequestIdentifier 19657
SiteComp.IAdapterDispatchParams.Handler 19658
SiteComp.IAdapterDispatchParams.ObjectIdentifier 19658
SiteComp.IAdapterDispatchParams.Params 19659
SiteComp.IAdapterDispatchParams.RequestIdentifier 19659
SiteComp.IAdapterError 19660
SiteComp.IAdapterError.ErrorText 19662
SiteComp.IAdapterError.Field 19662
SiteComp.IAdapterError.GetErrorText 19660
SiteComp.IAdapterError.GetField 19661
SiteComp.IAdapterError.GetID 19661
SiteComp.IAdapterError.ID 19662
SiteComp.IAdapterErrors 19663
SiteComp.IAdapterErrors.ClearErrors 19664
SiteComp.IAdapterErrors.DefineLabel 19664
SiteComp.IAdapterErrors.ErrorCount 19666
SiteComp.IAdapterErrors.Errors 19666
SiteComp.IAdapterErrors.GetError 19664
SiteComp.IAdapterErrors.GetErrorCount 19665
SiteComp.IAdapterErrors.GetObjectErrors 19665
SiteComp.IAdapterErrors.HasObjectErrors 19665
SiteComp.IAdapterFieldAccess 19666
SiteComp.IAdapterFieldAccess.HasModifyAccess 19667
SiteComp.IAdapterFieldAccess.HasViewAccess 19667
SiteComp.IAdapterFieldAttributes 19667
SiteComp.IAdapterFieldAttributes.GetRequired 19668
SiteComp.IAdapterFieldAttributes.GetVisible 19668
SiteComp.IAdapterFieldAttributes.Required 19669
SiteComp.IAdapterFieldAttributes.Visible 19669
SiteComp.IAdapterHiddenField 19669
SiteComp.IAdapterHiddenField.GetName 19670
SiteComp.IAdapterHiddenField.GetValue 19670
SiteComp.IAdapterHiddenField.Name 19671
SiteComp.IAdapterHiddenField.Value 19671
SiteComp.IAdapterHiddenFields 19671
SiteComp.IAdapterHiddenFields.FieldCount 19673
SiteComp.IAdapterHiddenFields.Fields 19673
SiteComp.IAdapterHiddenFields.GetField 19672
SiteComp.IAdapterHiddenFields.GetFieldCount 19672
SiteComp.IAdapterItemRequestParams 19673
SiteComp.IAdapterItemRequestParams.FailurePage 19676
SiteComp.IAdapterItemRequestParams.GetFailurePage 19674
SiteComp.IAdapterItemRequestParams.GetParamValues 19675
SiteComp.IAdapterItemRequestParams.GetSuccessPage 19675
SiteComp.IAdapterItemRequestParams.ParamValues 19677
SiteComp.IAdapterItemRequestParams.SetFailurePage 19675
SiteComp.IAdapterItemRequestParams.SetSuccessPage 19676
SiteComp.IAdapterItemRequestParams.SuccessPage 19677
SiteComp.IAdapterMode 19677
SiteComp.IAdapterMode.AdapterMode 19679
SiteComp.IAdapterMode.GetAdapterMode 19678
SiteComp.IAdapterMode.GetNeedAdapterMode 19678
SiteComp.IAdapterMode.NeedAdapterMode 19679
SiteComp.IAdapterMode.RestoreDefaultAdapterMode 19679
SiteComp.IAdapterMode.SetAdapterMode 19679
SiteComp.IAdapterRequestHandler 19680
SiteComp.IAdapterRequestHandler.CreateRequestContext 19681
SiteComp.IAdapterRequestHandler.HandleRequest 19681
SiteComp.IAsFormatted 19681
SiteComp.IAsFormatted.AsFormatted 19682
SiteComp.IDefaultPageFileName 19682
SiteComp.IDefaultPageFileName.SetDefaultPageFileName 19683
SiteComp.IDEFileManager 19797
SiteComp.IDEFileManager variable 19797
SiteComp.IFieldValuesAdapter 19683
SiteComp.IFieldValuesAdapter.GetField 19684
SiteComp.IFieldValuesAdapter.GetValueIndex 19684
SiteComp.IFindIncludeFile 19684
SiteComp.IFindIncludeFile.FindIncludeFile 19685
SiteComp.IGetAdapterActions 19685
SiteComp.IGetAdapterActions.GetAdapterActions 19686
SiteComp.IGetAdapterDisplayCharacteristics 19686

SiteComp.IGetAdapterDisplayCharacteristics.GetAdapterDisplay Characteristics
19687

SiteComp.IGetAdapterErrors 19687

SiteComp.IGetAdapterErrors.GetAdapterErrors 19688

SiteComp.IGetAdapterFields 19688

SiteComp.IGetAdapterFields.GetAdapterFields 19689

SiteComp.IGetAdapterHiddenFields 19689

SiteComp.IGetAdapterHiddenFields.GetAdapterHiddenFields
19690

SiteComp.IGetAdapterHiddenFields.GetAdapterHiddenRecordFi elds
19690

SiteComp.IGetAdapterImage 19690

SiteComp.IGetAdapterImage.GetAdapterImage 19691

SiteComp.IGetAdapterItemRequestParams 19691

SiteComp.IGetAdapterItemRequestParams.GetAdapterItemReq uestParams
19692

SiteComp.IGetAdapterRequestDefaultResponse 19692

SiteComp.IGetAdapterRequestDefaultResponse.FailurePage
19694

SiteComp.IGetAdapterRequestDefaultResponse.GetFailurePag e
19693

SiteComp.IGetAdapterRequestDefaultResponse.GetSuccessPa ge
19693

SiteComp.IGetAdapterRequestDefaultResponse.SuccessPage
19694

SiteComp.IGetAdapterValuesList 19694

SiteComp.IGetAdapterValuesList.GetAdapterValuesList 19695

SiteComp.IGetDefaultAction 19695

SiteComp.IGetDefaultAction.DefaultAction 19696

SiteComp.IGetDefaultAction.GetDefaultAction 19696

SiteComp.IGetDesigntimeWarnings 19697

SiteComp.IGetDesigntimeWarnings.GetDesigntimeWarnings
19697

SiteComp.IGetFieldValuesAdapter 19698

SiteComp.IGetFieldValuesAdapter.GetFieldValuesAdapter
19698

SiteComp.IGetHTMLStyle 19699

SiteComp.IGetHTMLStyle.GetDisplayStyle 19700

SiteComp.IGetHTMLStyle.GetInputStyle 19700

SiteComp.IGetHTMLStyle.GetViewMode 19701

SiteComp.IGetProducerComponent 19702

SiteComp.IGetProducerComponent.GetProducerComponent
19702

SiteComp.IGetScriptObject 19703

SiteComp.IGetScriptObject.GetScriptObject 19703

SiteComp.IGetScriptObjectClassKey 19704

SiteComp.IGetScriptObjectClassKey.GetScriptObjectClassKey
19704

SiteComp.IGetWebAppComponents 19705

SiteComp.IGetWebAppComponents.GetAdapterDispatcher
19706

SiteComp.IGetWebAppComponents.GetLocateFileService
19706

SiteComp.IGetWebAppComponents.GetPageDispatcher 19706

SiteComp.IGetWebAppComponents.GetWebApplicationInfo
19706

SiteComp.IGetWebAppComponents.GetWebEndUser 19706

SiteComp.IGetWebAppComponents.GetWebUserList 19707

SiteComp.IGetWebAppComponents.WebApplicationInfo 19707

SiteComp.IGetWebAppComponents.WebEndUser 19707

SiteComp.IIDEFileManager 19708

SiteComp.IIDEFileManager.SearchFile 19708

SiteComp.IImageProducer 19709

SiteComp.IImageProducer.GetAdapterImage 19710

SiteComp.IInternetEnvOptions 19710

SiteComp.IInternetEnvOptions.DisableDebugger 19712

SiteComp.IInternetEnvOptions.GetDisableDebugging 19711

SiteComp.IInternetEnvOptions.GetHTMLExt 19711

SiteComp.IInternetEnvOptions.GetSampleImageFile 19711

SiteComp.IInternetEnvOptions.GetScriptTimeoutSecs 19712

SiteComp.IInternetEnvOptions.HTMLExt 19712

SiteComp.IInternetEnvOptions.SampleImageFile 19712

SiteComp.IInternetEnvOptions.ScriptTimeoutSecs 19713

SiteComp.IInternetExecuteScript 19713

SiteComp.IInternetExecuteScript.CanExecuteScript 19714

SiteComp.IInternetExecuteScript.ExecuteScript 19714

SiteComp.IInternetFileTypes 19714

SiteComp.IInternetFileTypes.FindFileExt 19715

SiteComp.IInternetScriptDebugger 19715

SiteComp.IInternetScriptDebugger.CanDebugScript 19716

SiteComp.IInternetScriptDebugger.DebugScript 19716

SiteComp.IIsAdapterActionList 19716
SiteComp.IIsAdapterActionList.GetIsAdapterActionList 19717
SiteComp.IIsAdapterActionList.IsAdapterActionList 19717
SiteComp.IIterateIntfSupport 19718
SiteComp.IIterateIntfSupport.GetCurrentIntf 19718
SiteComp.IIterateObjectSupport 19719
SiteComp.IIterateObjectSupport.GetCurrentObject 19719
SiteComp.IIteratorIndex 19720
SiteComp.IIteratorIndex.GetIteratorIndex 19721
SiteComp.IIteratorIndex.InIterator 19721
SiteComp.IIteratorIndex.IteratorIndex 19721
SiteComp.IIteratorSupport 19722
SiteComp.IIteratorSupport.EndIterator 19723
SiteComp.IIteratorSupport.NextIteration 19723
SiteComp.IIteratorSupport.StartIterator 19724
SiteComp.InetFileTypes 19797
SiteComp.InetFileTypes variable 19797
SiteComp.INotifyAdapterChange 19724
SiteComp.INotifyAdapterChange.NotifyAdapterChange 19725
SiteComp.INotifyList 19725
SiteComp.INotifyList.AddNotify 19726
SiteComp.INotifyList.RemoveNotify 19726
SiteComp.InternetEnvOptions 19797
SiteComp.InternetEnvOptions variable 19797
SiteComp.IPageDispatcher 19727
SiteComp.IPageDispatcher.DefaultPageName 19731
SiteComp.IPageDispatcher.DispatchPageName 19728
SiteComp.IPageDispatcher.GetCanView 19729
SiteComp.IPageDispatcher.GetDefaultPageName 19729
SiteComp.IPageDispatcher.GetLoginRequired 19729
SiteComp.IPageDispatcher.PageNameOfRequest 19730
SiteComp.IPageDispatcher.RedirectToPageName 19730
SiteComp.IPageResult 19732
SiteComp.IPageResult.DispatchPage 19732
SiteComp.IPageResult.IncludePage 19733
SiteComp.IPageResult.RedirectToPage 19733
SiteComp.ISessionsService 19733
SiteComp.ISessionsService.CheckStatusChange 19734
SiteComp.ISessionsService.Clear 19734
SiteComp.ISessionsService.NewSession 19735
SiteComp.ISessionsService.NotifyStatusChange 19735
SiteComp.ISessionsServiceConnectID 19736
SiteComp.ISessionsServiceConnectID.Connect 19736
SiteComp.ISessionsServiceConnectID.ConnectActive 19737
SiteComp.ISessionsServicePersist 19737
SiteComp.ISessionsServicePersist.LoadFromFile 19738
SiteComp.ISessionsServicePersist.LoadFromStream 19738
SiteComp.ISessionsServicePersist.SaveToFile 19738
SiteComp.ISessionsServicePersist.SaveToStream 19738
SiteComp.ISessionsServiceStoreID 19739
SiteComp.ISessionsServiceStoreID.ReadSessionID 19739
SiteComp.ISessionsServiceStoreID.WriteSessionID 19740
SiteComp.IValuesListAdapter 19740
SiteComp.IValuesListAdapter.GetListImage 19741
SiteComp.IValuesListAdapter.GetListImageOfValue 19742
SiteComp.IValuesListAdapter.GetListName 19742
SiteComp.IValuesListAdapter.GetListNameOfValue 19742
SiteComp.IValuesListAdapter.GetListValue 19743
SiteComp.IWebActionsList 19743
SiteComp.IWebActionsList.FindAction 19744
SiteComp.IWebActionsList.GetActionsList 19744
SiteComp.IWebActionsList.GetVisibleActions 19745
SiteComp.IWebActionsList.IsActionInUse 19745
SiteComp.IWebActionsList.VisibleActions 19745
SiteComp.IWebApplicationInfo 19745
SiteComp.IWebApplicationInfo.ApplicationAdapter 19747
SiteComp.IWebApplicationInfo.ApplicationTitle 19748
SiteComp.IWebApplicationInfo.GetApplicationAdapter 19747
SiteComp.IWebApplicationInfo.GetApplicationTitle 19747
SiteComp.IWebContainerSubComponent 19748
SiteComp.IWebDataFields 19749
SiteComp.IWebDataFields.FindField 19749
SiteComp.IWebDataFields.GetFieldsList 19750
SiteComp.IWebDataFields.GetVisibleFields 19750
SiteComp.IWebDataFields.IsFieldInUse 19750
SiteComp.IWebDataFields.VisibleFields 19751
SiteComp.IWebEnabled 19751
SiteComp.IWebEnabled.WebEnabled 19752

SiteComp.IWebFormComponentContainer 19752
SiteComp.IWebFormComponentContainer.Count 19753
SiteComp.IWebFormComponentContainer.GetCount 19753
SiteComp.IWebFormComponentContainer.GetItem 19753
SiteComp.IWebFormComponentContainer.Items 19754
SiteComp.IWebGetActionName 19754
SiteComp.IWebGetActionName.ActionName 19755
SiteComp.IWebGetActionName.GetActionName 19755
SiteComp.IWebGetDisplayLabel 19755
SiteComp.IWebGetDisplayLabel.GetDisplayLabel 19756
SiteComp.IWebGetDisplayWidth 19756
SiteComp.IWebGetDisplayWidth.GetDisplayWidth 19757
SiteComp.IWebGetFieldName 19757
SiteComp.IWebGetFieldName.FieldName 19758
SiteComp.IWebGetFieldName.GetFieldName 19758
SiteComp.IWebGetFieldValue 19758
SiteComp.IWebGetFieldValue.GetValue 19759
SiteComp.IWebGetFieldValue.Value 19760
SiteComp.IWebGetFieldValues 19760
SiteComp.IWebGetFieldValues.EndValueIterator 19761
SiteComp.IWebGetFieldValues.GetCurrentValue 19761
SiteComp.IWebGetFieldValues.HasValue 19761
SiteComp.IWebGetFieldValues.NextValueIteration 19762
SiteComp.IWebGetFieldValues.StartValueIterator 19762
SiteComp.IWebGetMaxLength 19763
SiteComp.IWebGetMaxLength.GetMaxLength 19763
SiteComp.IWebGetSessionIDs 19764
SiteComp.IWebGetSessionIDs.GetSessionIDs 19764
SiteComp.IWebImageHREF 19765
SiteComp.IWebImageHREF.WebImageHREF 19766
SiteComp.IWebInputName 19766
SiteComp.IWebInputName.GetInputName 19767
SiteComp.IWebInputName.InputName 19767
SiteComp.IWebIsDefaultAction 19768
SiteComp.IWebIsDefaultAction.IsDefaultAction 19768
SiteComp.IWebIsDefaultField 19769
SiteComp.IWebIsDefaultField.IsDefaultField 19769
SiteComp.IWebSession 19770
SiteComp.IWebSession.Terminate 19771
SiteComp.IWebSessionAttributes 19771
SiteComp.IWebSessionAttributes.ExpirationTime 19774
SiteComp.IWebSessionAttributes.Expired 19774
SiteComp.IWebSessionAttributes.GetExpirationTime 19772
SiteComp.IWebSessionAttributes.GetExpired 19772
SiteComp.IWebSessionAttributes.GetIsActive 19773
SiteComp.IWebSessionAttributes.GetStartTime 19773
SiteComp.IWebSessionAttributes.GetTerminated 19773
SiteComp.IWebSessionAttributes.GetTimeoutMinutes 19773
SiteComp.IWebSessionAttributes.GetTouchTime 19773
SiteComp.IWebSessionAttributes.IsActive 19775
SiteComp.IWebSessionAttributes.SetTimeoutMinutes 19774
SiteComp.IWebSessionAttributes.StartTime 19775
SiteComp.IWebSessionAttributes.Terminated 19775
SiteComp.IWebSessionAttributes.TimeoutMinutes 19775
SiteComp.IWebSessionAttributes.TouchTime 19775
SiteComp.IWebSessionID 19776
SiteComp.IWebSessionID.GetID 19776
SiteComp.IWebSessionID.ID 19777
SiteComp.IWebSessionIDs 19777
SiteComp.IWebSessionIDs.GetID 19778
SiteComp.IWebSessionIDs.GetIDCount 19778
SiteComp.IWebSessionIDs.IDCount 19779
SiteComp.IWebSessionIDs.IDs 19779
SiteComp.IWebSessionValues 19780
SiteComp.IWebSessionValues.GetItemCount 19780
SiteComp.IWebSessionValues.GetItems 19781
SiteComp.IWebSessionValues.GetValue 19781
SiteComp.IWebSessionValues.SetValue 19781
SiteComp.IWebSessionValues.Values 19782
SiteComp.IWebSetActionName 19782
SiteComp.IWebSetActionName.SetActionName 19783
SiteComp.IWebSetFieldName 19783
SiteComp.IWebSetFieldName.SetFieldName 19784
SiteComp.IWebUserList 19784
SiteComp.IWebUserList.CheckAccessRights 19785
SiteComp.IWebUserList.ValidateUser 19785
SiteComp.IWebValueIsEqual 19786
SiteComp.IWebValueIsEqual.IsEqual 19787

SiteComp.ScriptDebugger 19798
SiteComp.ScriptDebugger variable 19798
SiteComp.TAbstractDesigntimeWarnings 19631
SiteComp.TAbstractDesigntimeWarnings.AddObject 19632
SiteComp.TAbstractDesigntimeWarnings.AddString 19632
SiteComp.TAbstractInetFileType 19633
SiteComp.TAbstractInetFileType.FileExt 19635
SiteComp.TAbstractInetFileType.FileExtCount 19635
SiteComp.TAbstractInetFileType.GetExtCount 19634
SiteComp.TAbstractInetFileType.GetFileExt 19634
SiteComp.TAbstractInetFileType.GetTypeID 19635
SiteComp.TAbstractInetFileType.TypeID 19635
SiteComp.TAdapterActionHTMLElementType 19789
SiteComp.TAdapterActionHTMLElementType enumeration 19789
SiteComp.TAdapterDisplayCharacteristic 19789
SiteComp.TAdapterDisplayCharacteristic enumeration 19789
SiteComp.TAdapterDisplayCharacteristics 19794
SiteComp.TAdapterDisplayCharacteristics type 19794
SiteComp.TAdapterDisplayHTMLElementType 19790
SiteComp.TAdapterDisplayHTMLElementType enumeration 19790
SiteComp.TAdapterDisplayViewModeType 19790
SiteComp.TAdapterDisplayViewModeType enumeration 19790
SiteComp.TAdapterInputHTMLElementType 19791
SiteComp.TAdapterInputHTMLElementType enumeration 19791
SiteComp.TAdapterItemRequestParamOption 19792
SiteComp.TAdapterItemRequestParamOption enumeration 19792
SiteComp.TAdapterItemRequestParamOptions 19794
SiteComp.TAdapterItemRequestParamOptions type 19794
SiteComp.TDispatchPageFlag 19792
SiteComp.TDispatchPageFlag enumeration 19792
SiteComp.TDispatchPageFlags 19795
SiteComp.TDispatchPageFlags type 19795
SiteComp.TNotifyList 19636
SiteComp.TNotifyList.AddNotify 19638
SiteComp.TNotifyList.Count 19640
SiteComp.TNotifyList.Create 19637
SiteComp.TNotifyList.Destroy 19637
SiteComp.TNotifyList.GetCount 19638
SiteComp.TNotifyList.GetItem 19639
SiteComp.TNotifyList.Items 19640
SiteComp.TNotifyList.RemoveNotify 19639
SiteComp.TQualifyOption 19792
SiteComp.TQualifyOption enumeration 19792
SiteComp.TScriptComponent 19641
SiteComp.TScriptComponent.Create 19642
SiteComp.TScriptComponent.GetComponent 19643
SiteComp.TScriptComponent.Obj 19643
SiteComp.TScriptObject 19643
SiteComp.TSessionID 19795
SiteComp.TSessionID type 19795
SiteProd 19801
SiteProd namespace 19801
    Classes 19802
SiteProd.TAdapterPageProducer 19802
SiteProd.TAdapterPageProducer.HTMLDoc 19808
SiteProd.TAdapterPageProducer.HTMLFile 19809
SiteProd.TAdapterPageProducer.OnAfterGetContent 19812
SiteProd.TAdapterPageProducer.OnBeforeGetContent 19812
SiteProd.TAdapterPageProducer.Styles 19810
SiteProd.TAdapterPageProducer.StylesFile 19811
SiteProd.TAdapterPageProducer.WebPageItems 19811
SiteProd.TBaseAdapterPageProducer 19813
SiteProd.TBaseAdapterPageProducer.CanAddClass 19820
SiteProd.TBaseAdapterPageProducer.ContentFromStream 19820
SiteProd.TBaseAdapterPageProducer.Create 19818
SiteProd.TBaseAdapterPageProducer.CreatePageElements 19821
SiteProd.TBaseAdapterPageProducer.DeclareStyles 19821
SiteProd.TBaseAdapterPageProducer.DeclareWarnings 19821
SiteProd.TBaseAdapterPageProducer.Destroy 19818
SiteProd.TBaseAdapterPageProducer.DoAfterGetContent 19822
SiteProd.TBaseAdapterPageProducer.DoBeforeGetContent 19822
SiteProd.TBaseAdapterPageProducer.DoTagEvent 19822
SiteProd.TBaseAdapterPageProducer.GeneratePageElements 19823

SiteProd.TBaseAdapterPageProducer.GetContentOptions 19823
SiteProd.TBaseAdapterPageProducer.GetDefaultTemplate 19824
SiteProd.TBaseAdapterPageProducer.GetRequiredTags 19824
SiteProd.TBaseAdapterPageProducer.GetScriptEngines 19825
SiteProd.TBaseAdapterPageProducer.GetTemplateFileType 19825
SiteProd.TBaseAdapterPageProducer.GetXMLData 19825
SiteProd.TBaseAdapterPageProducer.GetXSLData 19826
SiteProd.TBaseAdapterPageProducer.HandleScriptTag 19826
SiteProd.TBaseAdapterPageProducer.HasHTMLBrowserView 19827
SiteProd.TBaseAdapterPageProducer.HasScriptView 19827
SiteProd.TBaseAdapterPageProducer.HasXMLBrowserView 19827
SiteProd.TBaseAdapterPageProducer.HasXSLBrowserView 19828
SiteProd.TBaseAdapterPageProducer.ImplCanAddClass 19828
SiteProd.TBaseAdapterPageProducer.OnAfterGetContent 19832
SiteProd.TBaseAdapterPageProducer.OnBeforeGetContent 19832
SiteProd.TBaseAdapterPageProducer.PageElements 19830
SiteProd.TBaseAdapterPageProducer.SetStyles 19829
SiteProd.TBaseAdapterPageProducer.SetStylesFile 19829
SiteProd.TBaseAdapterPageProducer.SetWebContentOptions 19829
SiteProd.TBaseAdapterPageProducer.Styles 19830
SiteProd.TBaseAdapterPageProducer.StylesFile 19831
SiteProd.TCustomAdapterPageProducer 19832
SiteProd.TCustomAdapterPageProducer.Create 19838
SiteProd.TCustomAdapterPageProducer.Destroy 19839
SiteProd.TCustomAdapterPageProducer.GetTemplateFileName 19839
SiteProd.TCustomAdapterPageProducer.GetTemplateStream 19840
SiteProd.TCustomAdapterPageProducer.HTMLDoc 19842
SiteProd.TCustomAdapterPageProducer.HTMLFile 19843
SiteProd.TCustomAdapterPageProducer.SetHTMLDoc 19840
SiteProd.TCustomAdapterPageProducer.SetHTMLFile 19840
SiteProd.TCustomAdapterPageProducer.WebContent 19841
SiteProd.TSitePageElements 19844
SiteProd.TSitePageElements.ServerFunctionsContent 19845
SiteProd.TSitePageElements.ServerScriptContent 19845
SiteProd.TSitePageElements.ServerVariablesContent 19846
SiteProd.TSitePageElements.StylesContent 19846
SiteProd.TSitePageElements.WarningsContent 19846
siTranslations enumeration member 1305
siUsagePrivileges enumeration member 1305
siViewColumnUsage enumeration member 1305
siViews enumeration member 1305
siViewTableUsage enumeration member 1305
SmallString template 22239
  [] 22241
  = 22241
  about SmallString template 22239
  AnsiString 22242
  SmallString 22240, 22241
  SmallString constructor 22240
  SmallString Members 22239
  SmallString operators 22241
SmallStringBase template 22242
  about SmallStringBase template 22242
  Data 22243
  Len 22243
  SmallStringBase data members 22242
  SmallStringBase Members 22242
smAutomation enumeration member 5638, 17914
smAutoSize enumeration member 18014
smCenter enumeration member 18014
smClip enumeration member 18014
smRegServer enumeration member 5638, 17914
smScale enumeration member 18014
smStandalone enumeration member 5638, 17914
smStretch enumeration member 18014
smUnregServer enumeration member 5638, 17914
soAfter enumeration member 1308
soAfterEQ enumeration member 1308
SOAPAttachIntf 19847
SOAPAttachIntf namespace 19847
  Classes 19848

Interfaces 19853
Types 19862
SOAPAttachIntf.IMimeAttachmentHandler 19853
SOAPAttachIntf.IMimeAttachmentHandler.AddSoapHeader 19854
SOAPAttachIntf.IMimeAttachmentHandler.CreateMimeStream 19855
SOAPAttachIntf.IMimeAttachmentHandler.FinalizeStream 19856
SOAPAttachIntf.IMimeAttachmentHandler.GetMIMEBoundary 19856
SOAPAttachIntf.IMimeAttachmentHandler.GetMIMEStream 19856
SOAPAttachIntf.IMimeAttachmentHandler.GetOnGetAttachmentEvent 19857
SOAPAttachIntf.IMimeAttachmentHandler.GetOnSendAttachmentEvent 19857
SOAPAttachIntf.IMimeAttachmentHandler.MIMEBoundary 19859
SOAPAttachIntf.IMimeAttachmentHandler.OnGetAttachment 19860
SOAPAttachIntf.IMimeAttachmentHandler.OnSendAttachment 19860
SOAPAttachIntf.IMimeAttachmentHandler.ProcessMultiPartForm 19857
SOAPAttachIntf.IMimeAttachmentHandler.SetMIMEBoundary 19858
SOAPAttachIntf.IMimeAttachmentHandler.SetOnGetAttachmentEvent 19858
SOAPAttachIntf.IMimeAttachmentHandler.SetOnSendAttachmentEvent 19859
SOAPAttachIntf.TOnGetAttachmentEvent 19862
SOAPAttachIntf.TOnGetAttachmentEvent type 19862
SOAPAttachIntf.TOnSendAttachmentEvent 19862
SOAPAttachIntf.TOnSendAttachmentEvent type 19862
SOAPAttachIntf.TSoapDataList 19848
SOAPAttachIntf.TSoapDataList.Add 19850
SOAPAttachIntf.TSoapDataList.Assign 19850
SOAPAttachIntf.TSoapDataList.Clear 19851
SOAPAttachIntf.TSoapDataList.Insert 19851
SOAPAttachIntf.TSoapDataList.Remove 19852
SOAPConn 19863
SOAPConn namespace 19863
Classes 19864
SOAPConn.TSoapConnection 19864
SOAPConn.TSoapConnection.AfterConnect 19880
SOAPConn.TSoapConnection.AfterDisconnect 19880
SOAPConn.TSoapConnection.Agent 19874
SOAPConn.TSoapConnection.BeforeConnect 19880
SOAPConn.TSoapConnection.BeforeDisconnect 19881
SOAPConn.TSoapConnection.Connected 19875
SOAPConn.TSoapConnection.Create 19869
SOAPConn.TSoapConnection.Destroy 19869
SOAPConn.TSoapConnection.DoConnect 19870
SOAPConn.TSoapConnection.DoDisconnect 19870
SOAPConn.TSoapConnection.GetConnected 19871
SOAPConn.TSoapConnection.GetProviderNames 19871
SOAPConn.TSoapConnection.GetServer 19871
SOAPConn.TSoapConnection.GetServerList 19872
SOAPConn.TSoapConnection.GetSOAPServer 19872
SOAPConn.TSoapConnection.GetSOAPServerIID 19873
SOAPConn.TSoapConnection.Notification 19873
SOAPConn.TSoapConnection.OnAfterExecute 19881
SOAPConn.TSoapConnection.OnBeforeExecute 19882
SOAPConn.TSoapConnection.Password 19875
SOAPConn.TSoapConnection.Proxy 19876
SOAPConn.TSoapConnection.ProxyByPass 19876
SOAPConn.TSoapConnection.RIO 19876
SOAPConn.TSoapConnection.SetSOAPServerIID 19874
SOAPConn.TSoapConnection.SetUseSoapAdapter 19874
SOAPConn.TSoapConnection.SOAPServerIID 19877
SOAPConn.TSoapConnection.URL 19878
SOAPConn.TSoapConnection.UserName 19878
SOAPConn.TSoapConnection.UseSOAPAdapter 19879
SOAPDm 19883
SOAPDm namespace 19883
Classes 19884
SOAPDm.TSoapDataModule 19884
SOAPDm.TSoapDataModule._AddRef 19890
SOAPDm.TSoapDataModule._Release 19890
SOAPDm.TSoapDataModule.AfterConstruction 19891

SOAPDm.TSoapDataModule.ApplyUpdates 19891
SOAPDm.TSoapDataModule.AS_ApplyUpdates 19892
SOAPDm.TSoapDataModule.AS_DataRequest 19893
SOAPDm.TSoapDataModule.AS_Execute 19893
SOAPDm.TSoapDataModule.AS_GetParams 19894
SOAPDm.TSoapDataModule.AS_GetProviderNames 19895
SOAPDm.TSoapDataModule.AS_GetRecords 19895
SOAPDm.TSoapDataModule.AS_RowRequest 19897
SOAPDm.TSoapDataModule.BeforeDestruction 19897
SOAPDm.TSoapDataModule.Create 19888
SOAPDm.TSoapDataModule.DataRequest 19898
SOAPDm.TSoapDataModule.Destroy 19889
SOAPDm.TSoapDataModule.Execute 19898
SOAPDm.TSoapDataModule.GetParams 19899
SOAPDm.TSoapDataModule.GetProvider 19899
SOAPDm.TSoapDataModule.GetProviderNames 19900
SOAPDm.TSoapDataModule.GetRecords 19900
SOAPDm.TSoapDataModule.Lock 19901
SOAPDm.TSoapDataModule.NewInstance 19901
SOAPDm.TSoapDataModule.ProviderCount 19908
SOAPDm.TSoapDataModule.Providers 19908
SOAPDm.TSoapDataModule.RefCount 19909
SOAPDm.TSoapDataModule.RegisterProvider 19902
SOAPDm.TSoapDataModule.RowRequest 19902
SOAPDm.TSoapDataModule.SafeCallException 19903
SOAPDm.TSoapDataModule.SAS_ApplyUpdates 19903
SOAPDm.TSoapDataModule.SAS_DataRequest 19903
SOAPDm.TSoapDataModule.SAS_Execute 19904
SOAPDm.TSoapDataModule.SAS_GetParams 19905
SOAPDm.TSoapDataModule.SAS_GetProviderNames 19906
SOAPDm.TSoapDataModule.SAS_GetRecords 19906
SOAPDm.TSoapDataModule.SAS_RowRequest 19907
SOAPDm.TSoapDataModule.Unlock 19907
SOAPDm.TSoapDataModule.UnRegisterProvider 19907
SOAPDomConv 19911
SOAPDomConv namespace 19911
Classes 19912
Interfaces 19921
Structs, Records, Enums 19924
Types 19925
SOAPDomConv.IDOMHeaderProcessor 19921
SOAPDomConv.IDOMHeaderProcessor.CreateHeader 19922
SOAPDomConv.IDOMHeaderProcessor.ProcessHeader 19922
SOAPDomConv.TDOMHeaderProcessorArray 19925
SOAPDomConv.TDOMHeaderProcessorArray type 19925
SOAPDomConv.TDOMHeaderProcessorEntry 19924
SOAPDomConv.TDOMHeaderProcessorEntry record 19924
SOAPDomConv.TSOAPDOMProcessor 19912
SOAPDomConv.TSOAPDOMProcessor._AddRef 19916
SOAPDomConv.TSOAPDOMProcessor._Release 19917
SOAPDomConv.TSOAPDOMProcessor.AddHeaderProcessor 19917
SOAPDomConv.TSOAPDOMProcessor.AfterConstruction 19918
SOAPDomConv.TSOAPDOMProcessor.DefaultProcessHeader 19918
SOAPDomConv.TSOAPDOMProcessor.Destroy 19915
SOAPDomConv.TSOAPDOMProcessor.FHeaderProcessors 19915
SOAPDomConv.TSOAPDOMProcessor.FindHeaderProcessor 19919
SOAPDomConv.TSOAPDOMProcessor.MakeHeaderNodes 19919
SOAPDomConv.TSOAPDOMProcessor.NewInstance 19919
SOAPHTTPClient 19927
SOAPHTTPClient namespace 19927
Classes 19928
SOAPHTTPClient.THTTPRIO 19928
SOAPHTTPClient.THTTPRIO.Converter 19935
SOAPHTTPClient.THTTPRIO.Create 19932
SOAPHTTPClient.THTTPRIO.Destroy 19932
SOAPHTTPClient.THTTPRIO.HTTPWebNode 19935
SOAPHTTPClient.THTTPRIO.Notification 19933
SOAPHTTPClient.THTTPRIO.Port 19936
SOAPHTTPClient.THTTPRIO.QueryInterface 19934
SOAPHTTPClient.THTTPRIO.Service 19936
SOAPHTTPClient.THTTPRIO.URL 19937
SOAPHTTPClient.THTTPRIO.WSDLItems 19937
SOAPHTTPClient.THTTPRIO.WSDLLocation 19938
SOAPHTTPDisp 19939

SOAPHTTPDisp namespace 19939
Classes 19940
Interfaces 19946
SOAPHTTPDisp.IHTTPSoapDispatch 19946
SOAPHTTPDisp.IHTTPSoapDispatch.DispatchSOAP 19947
SOAPHTTPDisp.THTTPSoapDispatchNode 19940
SOAPHTTPDisp.THTTPSoapDispatchNode.Dispatcher 19945
SOAPHTTPDisp.THTTPSoapDispatchNode.DispatchSOAP 19944
SOAPHTTPDisp.THTTPSoapDispatchNode.FSoapDispatcher 19943
SOAPHTTPDisp.THTTPSoapDispatchNode.Notification 19944
SOAPHTTPPasInv 19949
SOAPHTTPPasInv namespace 19949
Classes 19950
SOAPHTTPPasInv.THTTPSoapCppInvoker 19950
SOAPHTTPPasInv.THTTPSoapPascalInvoker 19951
SOAPHTTPPasInv.THTTPSoapPascalInvoker.DispatchSOAP 19955
SOAPHTTPTrans 19957
SOAPHTTPTrans namespace 19957
Classes 19958
Interfaces 19983
Structs, Records, Enums 19985
Types 19986
SOAPHTTPTrans.ESOAPHTTPException 19958
SOAPHTTPTrans.ESOAPHTTPException.Create 19959
SOAPHTTPTrans.ESOAPHTTPException.CreateFmt 19960
SOAPHTTPTrans.ESOAPHTTPException.StatusCode 19961
SOAPHTTPTrans.IHTTPReqResp 19983
SOAPHTTPTrans.IHTTPReqResp.GetHTTPReqResp 19984
SOAPHTTPTrans.SOAPInvokeOptions 19985
SOAPHTTPTrans.SOAPInvokeOptions enumeration 19985
SOAPHTTPTrans.TBeforePostEvent 19986
SOAPHTTPTrans.TBeforePostEvent type 19986
SOAPHTTPTrans.THTTPReqResp 19961
SOAPHTTPTrans.THTTPReqResp._AddRef 19967
SOAPHTTPTrans.THTTPReqResp._Release 19967
SOAPHTTPTrans.THTTPReqResp.AfterConstruction 19968
SOAPHTTPTrans.THTTPReqResp.Agent 19975
SOAPHTTPTrans.THTTPReqResp.BeforeExecute 19968
SOAPHTTPTrans.THTTPReqResp.Check 19969
SOAPHTTPTrans.THTTPReqResp.CheckContentType 19969
SOAPHTTPTrans.THTTPReqResp.Connect 19970
SOAPHTTPTrans.THTTPReqResp.ConnectTimeout 19976
SOAPHTTPTrans.THTTPReqResp.Create 19966
SOAPHTTPTrans.THTTPReqResp.Destroy 19966
SOAPHTTPTrans.THTTPReqResp.Execute 19971
SOAPHTTPTrans.THTTPReqResp.Get 19971
SOAPHTTPTrans.THTTPReqResp.GetHTTPReqResp 19972
SOAPHTTPTrans.THTTPReqResp.GetMimeBoundary 19972
SOAPHTTPTrans.THTTPReqResp.InvokeOptions 19976
SOAPHTTPTrans.THTTPReqResp.MaxSinglePostSize 19976
SOAPHTTPTrans.THTTPReqResp.NewInstance 19972
SOAPHTTPTrans.THTTPReqResp.OnBeforePost 19981
SOAPHTTPTrans.THTTPReqResp.OnPostingData 19982
SOAPHTTPTrans.THTTPReqResp.OnReceivingData 19982
SOAPHTTPTrans.THTTPReqResp.Password 19976
SOAPHTTPTrans.THTTPReqResp.Proxy 19977
SOAPHTTPTrans.THTTPReqResp.ProxyByPass 19977
SOAPHTTPTrans.THTTPReqResp.Receive 19973
SOAPHTTPTrans.THTTPReqResp.ReceiveTimeout 19978
SOAPHTTPTrans.THTTPReqResp.Send 19973
SOAPHTTPTrans.THTTPReqResp.SendGet 19974
SOAPHTTPTrans.THTTPReqResp.SendTimeout 19978
SOAPHTTPTrans.THTTPReqResp.SetMimeBoundary 19974
SOAPHTTPTrans.THTTPReqResp.SoapAction 19978
SOAPHTTPTrans.THTTPReqResp.UDDIBindingKey 19979
SOAPHTTPTrans.THTTPReqResp.UDDIOperator 19979
SOAPHTTPTrans.THTTPReqResp.URL 19980
SOAPHTTPTrans.THTTPReqResp.UserName 19980
SOAPHTTPTrans.THTTPReqResp.UseUTF8InHeader 19980
SOAPHTTPTrans.THTTPReqResp.WSDLView 19981
SOAPHTTPTrans.TPostingDataEvent 19986
SOAPHTTPTrans.TPostingDataEvent type 19986
SOAPHTTPTrans.TReceivingDataEvent 19987
SOAPHTTPTrans.TReceivingDataEvent type 19987
SOAPHTTPTrans.TSOAPInvokeOptions 19987
SOAPHTTPTrans.TSOAPInvokeOptions type 19987

SOAPLinked 19989
SOAPLinked namespace 19989
Classes 19990
SOAPLinked.TLinkedRIO 19990
SOAPLinked.TLinkedRIO.Create 19994
SOAPLinked.TLinkedRIO.CreateFile 19994
SOAPLinked.TLinkedRIO.Destroy 19995
SOAPLinked.TLinkedRIO.GetResponseStream 19996
SOAPLinked.TLinkedRIO.QueryInterface 19996
SOAPLinked.TLinkedRIO.WebNode 19997
SOAPLinked.TLinkedWebNode 19997
SOAPLinked.TLinkedWebNode.BeforeExecute 20002
SOAPLinked.TLinkedWebNode.Create 20001
SOAPLinked.TLinkedWebNode.Destroy 20001
SOAPLinked.TLinkedWebNode.Execute 20003
SOAPLinked.TLinkedWebNode.GetMimeBoundary 20003
SOAPLinked.TLinkedWebNode.GetResponseStream 20004
SOAPLinked.TLinkedWebNode.Invoker 20004
SOAPLinked.TLinkedWebNode.MimeBoundary 20005
SOAPLinked.TLinkedWebNode.SetMimeBoundary 20004
SOAPLinked.TLogLinkedWebNode 20005
SOAPLinked.TLogLinkedWebNode.Create 20008
SOAPLinked.TLogLinkedWebNode.Destroy 20009
SOAPLinked.TLogLinkedWebNode.Execute 20009
SOAPLinked.TLogLinkedWebNode.GetResponseStream 20010
SOAPLinked.TLogLinkedWebNode.ReqFile 20010
SOAPLinked.TLogLinkedWebNode.RespFile 20010
SOAPMidas 20011
SOAPMidas namespace 20011
Functions 20017
Interfaces 20012
SOAPMidas.IAppServerSOAP 20012
SOAPMidas.IAppServerSOAP.SAS_ApplyUpdates 20013
SOAPMidas.IAppServerSOAP.SAS_DataRequest 20013
SOAPMidas.IAppServerSOAP.SAS_Execute 20014
SOAPMidas.IAppServerSOAP.SAS_GetParams 20015
SOAPMidas.IAppServerSOAP.SAS_GetProviderNames 20015
SOAPMidas.IAppServerSOAP.SAS_GetRecords 20015
SOAPMidas.IAppServerSOAP.SAS_RowRequest 20016
SOAPMidas.RegDefIAppServerInvClass 20017
SOAPMidas.RegDefIAppServerInvClass function 20017
SOAPPasInv 20019
SOAPPasInv namespace 20019
Classes 20020
Functions 20035
Interfaces 20032
Types 20036
SOAPPasInv.GetSOAPHeaders 20035
SOAPPasInv.GetSOAPHeaders function 20035
SOAPPasInv.IAdapterConverter 20032
SOAPPasInv.IAdapterConverter.GetDomConverter 20033
SOAPPasInv.IInvokerConverter 20033
SOAPPasInv.IInvokerConverter.GetInvoker 20034
SOAPPasInv.TAfterDispatchEvent 20036
SOAPPasInv.TAfterDispatchEvent type 20036
SOAPPasInv.TBeforeDispatchEvent 20036
SOAPPasInv.TBeforeDispatchEvent type 20036
SOAPPasInv.TBeforeDispatchEvent2 20037
SOAPPasInv.TBeforeDispatchEvent2 type 20037
SOAPPasInv.TOnExceptionEvent 20037
SOAPPasInv.TOnExceptionEvent type 20037
SOAPPasInv.TSoapPascalInvoker 20020
SOAPPasInv.TSoapPascalInvoker.AfterDispatchEvent 20028
SOAPPasInv.TSoapPascalInvoker.BeforeDispatchEvent 20029
SOAPPasInv.TSoapPascalInvoker.BeforeDispatchEvent2 20029
SOAPPasInv.TSoapPascalInvoker.Converter 20028
SOAPPasInv.TSoapPascalInvoker.Create 20024
SOAPPasInv.TSoapPascalInvoker.Destroy 20024
SOAPPasInv.TSoapPascalInvoker.FConverter 20025
SOAPPasInv.TSoapPascalInvoker.FDomConverter 20025
SOAPPasInv.TSoapPascalInvoker.FHeadersIn 20025
SOAPPasInv.TSoapPascalInvoker.FHeadersOut 20025
SOAPPasInv.TSoapPascalInvoker.FSOAPHeaders 20025
SOAPPasInv.TSoapPascalInvoker.GetDomConverter 20026
SOAPPasInv.TSoapPascalInvoker.GetInvoker 20026
SOAPPasInv.TSoapPascalInvoker.Invoke 20026
SOAPPasInv.TSoapPascalInvoker.Notification 20027
SOAPPasInv.TSoapPascalInvoker.OnExceptionEvent 20030

SOAPPasInv.TSoapPascalInvoker.OnGetAttachment 20030
SOAPPasInv.TSoapPascalInvoker.OnSendAttachment 20030
soAutoCheckAccessPointViaUDDI enumeration member 19985
soBefore enumeration member 1308
soBeforeEQ enumeration member 1308
soBeginning enumeration member 3276
soCacheMimeResponse enumeration member 18132
Sockets 20039
Sockets namespace 20039
    Classes 20041
    Constants 20144
    Structs, Records, Enums 20135
    Types 20140
Sockets.CRLF 20144
Sockets.CRLF constant 20144
Sockets.ESocketError 20041
Sockets.TBaseSocket 20042
Sockets.TBaseSocket.Active 20054
Sockets.TBaseSocket.BlockMode 20054
Sockets.TBaseSocket.BytesReceived 20055
Sockets.TBaseSocket.BytesSent 20055
Sockets.TBaseSocket.Close 20047
Sockets.TBaseSocket.Create 20046
Sockets.TBaseSocket.Destroy 20046
Sockets.TBaseSocket.DoCreateHandle 20047
Sockets.TBaseSocket.DoDestroyHandle 20047
Sockets.TBaseSocket.DoHandleError 20048
Sockets.TBaseSocket.Domain 20055
Sockets.TBaseSocket.DoReceive 20048
Sockets.TBaseSocket.DoSend 20048
Sockets.TBaseSocket.ErrorCheck 20048
Sockets.TBaseSocket.Handle 20056
Sockets.TBaseSocket.Loaded 20049
Sockets.TBaseSocket.MapDomain 20049
Sockets.TBaseSocket.MapSockType 20050
Sockets.TBaseSocket.OnCreateHandle 20057
Sockets.TBaseSocket.OnDestroyHandle 20057
Sockets.TBaseSocket.OnError 20058
Sockets.TBaseSocket.OnReceive 20058
Sockets.TBaseSocket.OnSend 20058
Sockets.TBaseSocket.Open 20050
Sockets.TBaseSocket.PeekBuf 20050
Sockets.TBaseSocket.Protocol 20056
Sockets.TBaseSocket.ReceiveBuf 20051
Sockets.TBaseSocket.Receiveln 20051
Sockets.TBaseSocket.Select 20051
Sockets.TBaseSocket.SendBuf 20052
Sockets.TBaseSocket.Sendln 20052
Sockets.TBaseSocket.SendStream 20053
Sockets.TBaseSocket.SetBytesReceived 20053
Sockets.TBaseSocket.SetBytesSent 20053
Sockets.TBaseSocket.SockType 20056
Sockets.TBaseSocket.WaitForData 20053
Sockets.TClientSocketThread 20059
Sockets.TClientSocketThread.ClientSocket 20063
Sockets.TClientSocketThread.Create 20060
Sockets.TClientSocketThread.Destroy 20061
Sockets.TClientSocketThread.Execute 20061
Sockets.TClientSocketThread.ExecuteSyncProc 20062
Sockets.TClientSocketThread.ServerSocketThread 20063
Sockets.TClientSocketThread.SyncProc 20062
Sockets.TCustomIpClient 20063
Sockets.TCustomIpClient.Close 20068
Sockets.TCustomIpClient.Connect 20069
Sockets.TCustomIpClient.Connected 20071
Sockets.TCustomIpClient.Create 20068
Sockets.TCustomIpClient.Disconnect 20069
Sockets.TCustomIpClient.DoConnect 20069
Sockets.TCustomIpClient.DoDisconnect 20070
Sockets.TCustomIpClient.GetThreadObject 20070
Sockets.TCustomIpClient.OnConnect 20071
Sockets.TCustomIpClient.OnDisconnect 20072
Sockets.TCustomIpClient.Open 20070
Sockets.TCustomTcpServer 20072
Sockets.TCustomTcpServer.Accept 20077
Sockets.TCustomTcpServer.BlockMode 20080
Sockets.TCustomTcpServer.Close 20078
Sockets.TCustomTcpServer.Create 20076

Sockets.TCustomTcpServer.Destroy 20077
Sockets.TCustomTcpServer.DoAccept 20078
Sockets.TCustomTcpServer.Listen 20079
Sockets.TCustomTcpServer.Listening 20080
Sockets.TCustomTcpServer.OnAccept 20081
Sockets.TCustomTcpServer.OnGetThread 20081
Sockets.TCustomTcpServer.OnListening 20082
Sockets.TCustomTcpServer.Open 20079
Sockets.TCustomTcpServer.ServerSocketThread 20080
Sockets.TCustomTcpServer.WaitForConnection 20079
Sockets.TGetThreadEvent 20140
Sockets.TGetThreadEvent type 20140
Sockets.TIPHeader 20135
Sockets.TIPHeader record 20135
Sockets.TIpSocket 20082
Sockets.TIpSocket.Bind 20087
Sockets.TIpSocket.Create 20086
Sockets.TIpSocket.Domain 20091
Sockets.TIpSocket.GetSocketAddr 20087
Sockets.TIpSocket.LocalDomainName 20088
Sockets.TIpSocket.LocalHost 20092
Sockets.TIpSocket.LocalHostAddr 20088
Sockets.TIpSocket.LocalHostName 20088
Sockets.TIpSocket.LocalPort 20092
Sockets.TIpSocket.LookupHostAddr 20089
Sockets.TIpSocket.LookupHostName 20089
Sockets.TIpSocket.LookupPort 20089
Sockets.TIpSocket.LookupProtocol 20090
Sockets.TIpSocket.ReceiveFrom 20090
Sockets.TIpSocket.RemoteHost 20093
Sockets.TIpSocket.RemotePort 20093
Sockets.TIpSocket.SendTo 20091
Sockets.TRawSocket 20094
Sockets.TRawSocket.Create 20098
Sockets.TRawSocket.SockType 20098
Sockets.TServerSocketBlockMode 20136
Sockets.TServerSocketBlockMode enumeration 20136
Sockets.TServerSocketThread 20099
Sockets.TServerSocketThread.AddClientSocketThread 20101
Sockets.TServerSocketThread.ClearThreadPool 20102
Sockets.TServerSocketThread.Create 20100
Sockets.TServerSocketThread.CreateThread 20102
Sockets.TServerSocketThread.Destroy 20101
Sockets.TServerSocketThread.Execute 20102
Sockets.TServerSocketThread.FetchClientSocketThread 20103
Sockets.TServerSocketThread.OnGetThread 20105
Sockets.TServerSocketThread.RemoveClientSocketThread 20103
Sockets.TServerSocketThread.ServerSocket 20103
Sockets.TServerSocketThread.ThreadCacheSize 20104
Sockets.TServerSocketThread.ThreadPool 20104
Sockets.TSocketAcceptEvent 20140
Sockets.TSocketAcceptEvent type 20140
Sockets.TSocketBlockMode 20141
Sockets.TSocketBlockMode type 20141
Sockets.TSocketDataEvent 20141
Sockets.TSocketDataEvent type 20141
Sockets.TSocketDomain 20136
Sockets.TSocketDomain enumeration 20136
Sockets.TSocketErrorEvent 20141
Sockets.TSocketErrorEvent type 20141
Sockets.TSocketHost 20142
Sockets.TSocketHost type 20142
Sockets.TSocketNotifyEvent 20142
Sockets.TSocketNotifyEvent type 20142
Sockets.TSocketPort 20142
Sockets.TSocketPort type 20142
Sockets.TSocketProtocol 20143
Sockets.TSocketProtocol type 20143
Sockets.TSocketType 20138
Sockets.TSocketType enumeration 20138
Sockets.TTcpClient 20105
Sockets.TTcpClient.Active 20110
Sockets.TTcpClient.BlockMode 20110
Sockets.TTcpClient.Connected 20111
Sockets.TTcpClient.OnConnect 20112
Sockets.TTcpClient.OnCreateHandle 20113
Sockets.TTcpClient.OnDestroyHandle 20113
Sockets.TTcpClient.OnDisconnect 20113

Sockets.TTcpClient.OnError 20114
Sockets.TTcpClient.OnReceive 20114
Sockets.TTcpClient.OnSend 20114
Sockets.TTcpClient.RemoteHost 20111
Sockets.TTcpClient.RemotePort 20111
Sockets.TTcpServer 20115
Sockets.TTcpServer.Active 20120
Sockets.TTcpServer.BlockMode 20120
Sockets.TTcpServer.LocalHost 20120
Sockets.TTcpServer.LocalPort 20121
Sockets.TTcpServer.OnAccept 20122
Sockets.TTcpServer.OnCreateHandle 20122
Sockets.TTcpServer.OnDestroyHandle 20122
Sockets.TTcpServer.OnGetThread 20123
Sockets.TTcpServer.OnListening 20123
Sockets.TUdpSocket 20123
Sockets.TUdpSocket.Active 20128
Sockets.TUdpSocket.BlockMode 20129
Sockets.TUdpSocket.Create 20128
Sockets.TUdpSocket.LocalHost 20129
Sockets.TUdpSocket.LocalPort 20130
Sockets.TUdpSocket.OnConnect 20131
Sockets.TUdpSocket.OnCreateHandle 20132
Sockets.TUdpSocket.OnDestroyHandle 20132
Sockets.TUdpSocket.OnDisconnect 20132
Sockets.TUdpSocket.OnError 20133
Sockets.TUdpSocket.OnReceive 20133
Sockets.TUdpSocket.OnSend 20133
Sockets.TUdpSocket.RemoteHost 20130
Sockets.TUdpSocket.RemotePort 20130
soCurrent enumeration member 3276
soCustomFaultAtDetailsNode enumeration member 18132
soDocument enumeration member 18132
soDontClearOutBoundHeaders enumeration member 18132
soDontSendEmptyNodes enumeration member 18132
soDown enumeration member 21992
soEnd enumeration member 3276
sofBinary enumeration member 3276
soFirstEQ enumeration member 1308
sofText enumeration member 3276
sofUnknown enumeration member 3276
soHorizontal enumeration member 5461, 23290
soIgnoreInvalidCerts enumeration member 19985
soLastEQ enumeration member 1308
soLeftRight enumeration member 23230
soLiteralParams enumeration member 18132
soMatchCase enumeration member 21992
soNone enumeration member 23290
soNoSOAPActionHeader enumeration member 19985
soNoValueForEmptySOAPAction enumeration member 19985
soOwned enumeration member 3277
soReference enumeration member 3277
soReturnSuccessForFault enumeration member 18132
soRootRefNodesToBody enumeration member 18132
soSendMultiRefArray enumeration member 18132
soSendMultiRefObj enumeration member 18132
soSendUntyped enumeration member 18132
soTryAllSchema enumeration member 18132
soUpDown enumeration member 23230
soUTF8EncodeXML enumeration member 18132
soUTF8InHeader enumeration member 18132
soVertical enumeration member 5461, 23290
soWholeWord enumeration member 21992
soXXXXHdr enumeration member 18132
sPointer template 22243
& 22245
~sPointer 22244
-> 22245
about sPointer template 22243
sPointer 22244
sPointer constructor 22244
sPointer Members 22243
sPointer operators 22245
SPParamDesc class 20147
about SPParamDesc class 20147
iArgType 20148
iDataType 20149
iLen 20149

iParamNum 20149
iUnits1 20149
iUnits2 20150
SPParamDesc fields 20148
SPParamDesc Members 20148
szName 20150
SQLException class 15682
about SQLException class 15682
ErrorCode 15684
getErrorCode 15684
getSQLState 15684
SQLException 15683
SQLException constructor 15683
SQLException Members 15682
SQLException methods 15683
SQLException properties 15684
SQLState 15684
SqlExpr 20145
SqlExpr namespace 20145
Classes 20147
Constants 20375
Functions 20361
Structs, Records, Enums 20364
Types 20371
SqlExpr.BlobTypeMap 20375
SqlExpr.BlobTypeMap constant 20375
SqlExpr.DataTypeMap 20376
SqlExpr.DataTypeMap constant 20376
SqlExpr.DefaultCursor 20376
SqlExpr.DefaultCursor constant 20376
SqlExpr.DefaultMaxBlobSize 20376
SqlExpr.DefaultMaxBlobSize constant 20376
SqlExpr.DefaultRowsetSize 20377
SqlExpr.DefaultRowsetSize constant 20377
SqlExpr.EConnectFlag 20364
SqlExpr.EConnectFlag enumeration 20364
SqlExpr.FldSubTypeMap 20377
SqlExpr.FldSubTypeMap constant 20377
SqlExpr.FldTypeMap 20377
SqlExpr.FldTypeMap constant 20377
SqlExpr.FreeProcParams 20361
SqlExpr.FreeProcParams function 20361
SqlExpr.GetConnectionNames 20361
SqlExpr.GetConnectionNames function 20361
SqlExpr.GetConnectionRegistryFile 20362
SqlExpr.GetConnectionRegistryFile function 20362
SqlExpr.GetDriverNames 20362
SqlExpr.GetDriverNames function 20362
SqlExpr.GetDriverRegistryFile 20363
SqlExpr.GetDriverRegistryFile function 20363
SqlExpr.HourGlassCursor 20378
SqlExpr.HourGlassCursor constant 20378
SqlExpr.LoadParamListItems 20363
SqlExpr.LoadParamListItems function 20363
SqlExpr.SAnd 20378
SqlExpr.SAnd constant 20378
SqlExpr.SOrderBy 20379
SqlExpr.SOrderBy constant 20379
SqlExpr.SParam 20379
SqlExpr.SParam constant 20379
SqlExpr.SPParamDesc 20147
SqlExpr.SPParamDesc.iArgType 20148
SqlExpr.SPParamDesc.iDataType 20149
SqlExpr.SPParamDesc.iLen 20149
SqlExpr.SPParamDesc.iParamNum 20149
SqlExpr.SPParamDesc.iUnits1 20149
SqlExpr.SPParamDesc.iUnits2 20150
SqlExpr.SPParamDesc.szName 20150
SqlExpr.SSelect 20379
SqlExpr.SSelect constant 20379
SqlExpr.SSelectSpaces 20380
SqlExpr.SSelectSpaces constant 20380
SqlExpr.SSelectStar 20380
SqlExpr.SSelectStar constant 20380
SqlExpr.SSelectStarFrom 20380
SqlExpr.SSelectStarFrom constant 20380
SqlExpr.SUB_TYPE_MEMO 20381
SqlExpr.SUB_TYPE_MEMO constant 20381

SqlExpr.SWhere 20381
SqlExpr.SWhere constant 20381
SqlExpr.TConnectionState 20365
SqlExpr.TConnectionState enumeration 20365
SqlExpr.TConnectionUserType 20365
SqlExpr.TConnectionUserType enumeration 20365
SqlExpr.TCustomSQLDataSet 20150
SqlExpr.TCustomSQLDataSet.Active 20187
SqlExpr.TCustomSQLDataSet.AddFieldDesc 20163
SqlExpr.TCustomSQLDataSet.AddIndexDefs 20163
SqlExpr.TCustomSQLDataSet.AfterClose 20203
SqlExpr.TCustomSQLDataSet.AfterOpen 20203
SqlExpr.TCustomSQLDataSet.AfterRefresh 20203
SqlExpr.TCustomSQLDataSet.AfterScroll 20204
SqlExpr.TCustomSQLDataSet.BeforeClose 20204
SqlExpr.TCustomSQLDataSet.BeforeOpen 20204
SqlExpr.TCustomSQLDataSet.BeforeRefresh 20204
SqlExpr.TCustomSQLDataSet.BeforeScroll 20204
SqlExpr.TCustomSQLDataSet.BlobBuffer 20189
SqlExpr.TCustomSQLDataSet.CheckPrepareError 20164
SqlExpr.TCustomSQLDataSet.ClearIndexDefs 20164
SqlExpr.TCustomSQLDataSet.CloseCursor 20164
SqlExpr.TCustomSQLDataSet.CloseStatement 20164
SqlExpr.TCustomSQLDataSet.CommandText 20189
SqlExpr.TCustomSQLDataSet.CommandType 20190
SqlExpr.TCustomSQLDataSet.Create 20160
SqlExpr.TCustomSQLDataSet.CreateBlobStream 20164
SqlExpr.TCustomSQLDataSet.CurrentBlobSize 20191
SqlExpr.TCustomSQLDataSet.DataLink 20191
SqlExpr.TCustomSQLDataSet.DataSource 20192
SqlExpr.TCustomSQLDataSet.DefineProperties 20165
SqlExpr.TCustomSQLDataSet.DesignerData 20192
SqlExpr.TCustomSQLDataSet.Destroy 20161
SqlExpr.TCustomSQLDataSet.ExecSQL 20165
SqlExpr.TCustomSQLDataSet.ExecuteStatement 20166
SqlExpr.TCustomSQLDataSet.FreeBuffers 20166
SqlExpr.TCustomSQLDataSet.FreeCommand 20166
SqlExpr.TCustomSQLDataSet.FreeReader 20166
SqlExpr.TCustomSQLDataSet.GetBlobFieldData 20166
SqlExpr.TCustomSQLDataSet.GetCanModify 20167
SqlExpr.TCustomSQLDataSet.GetDataSource 20167
SqlExpr.TCustomSQLDataSet.GetDetailLinkFields 20167
SqlExpr.TCustomSQLDataSet.GetFieldData 20168
SqlExpr.TCustomSQLDataSet.GetKeyFieldNames 20169
SqlExpr.TCustomSQLDataSet.GetMetadata 20193
SqlExpr.TCustomSQLDataSet.GetObjectTypeNames 20169
SqlExpr.TCustomSQLDataSet.GetOutputParams 20169
SqlExpr.TCustomSQLDataSet.GetQuoteChar 20169
SqlExpr.TCustomSQLDataSet.GetRecord 20170
SqlExpr.TCustomSQLDataSet.GetRecordCount 20170
SqlExpr.TCustomSQLDataSet.GetSortFieldNames 20170
SqlExpr.TCustomSQLDataSet.IndexDefs 20193
SqlExpr.TCustomSQLDataSet.InitRecord 20170
SqlExpr.TCustomSQLDataSet.InternalClose 20171
SqlExpr.TCustomSQLDataSet.InternalConnection 20194
SqlExpr.TCustomSQLDataSet.InternalHandleException 20171
SqlExpr.TCustomSQLDataSet.InternalInitFieldDefs 20171
SqlExpr.TCustomSQLDataSet.InternalOpen 20171
SqlExpr.TCustomSQLDataSet.InternalRefresh 20171
SqlExpr.TCustomSQLDataSet.IsCursorOpen 20172
SqlExpr.TCustomSQLDataSet.IsSequenced 20172
SqlExpr.TCustomSQLDataSet.LastError 20194
SqlExpr.TCustomSQLDataSet.Loaded 20172
SqlExpr.TCustomSQLDataSet.Locate 20173
SqlExpr.TCustomSQLDataSet.LocateRecord 20173
SqlExpr.TCustomSQLDataSet.Lookup 20174
SqlExpr.TCustomSQLDataSet.MaxBlobSize 20194
SqlExpr.TCustomSQLDataSet.NativeCommand 20195
SqlExpr.TCustomSQLDataSet.NumericMapping 20195
SqlExpr.TCustomSQLDataSet.ObjectView 20196
SqlExpr.TCustomSQLDataSet.OnCalcFields 20204
SqlExpr.TCustomSQLDataSet.OpenCursor 20174
SqlExpr.TCustomSQLDataSet.OpenSchema 20174
SqlExpr.TCustomSQLDataSet.ParamByName 20175
SqlExpr.TCustomSQLDataSet.ParamCheck 20196
SqlExpr.TCustomSQLDataSet.ParamCount 20197
SqlExpr.TCustomSQLDataSet.Params 20197
SqlExpr.TCustomSQLDataSet.ParseDeleteSql 20205

SqlExpr.TCustomSQLDataSet.ParseInsertSql 20205
SqlExpr.TCustomSQLDataSet.ParseSelectSql 20205
SqlExpr.TCustomSQLDataSet.ParseUpdateSql 20205
SqlExpr.TCustomSQLDataSet.Prepared 20198
SqlExpr.TCustomSQLDataSet.PrepareStatement 20175
SqlExpr.TCustomSQLDataSet.ProcParams 20199
SqlExpr.TCustomSQLDataSet.PropertyChanged 20175
SqlExpr.TCustomSQLDataSet.PSEndTransaction 20175
SqlExpr.TCustomSQLDataSet.PSExecute 20176
SqlExpr.TCustomSQLDataSet.PSExecuteStatement 20176
SqlExpr.TCustomSQLDataSet.PSGetAttributes 20176
SqlExpr.TCustomSQLDataSet.PSGetCommandText 20177
SqlExpr.TCustomSQLDataSet.PSGetCommandType 20177
SqlExpr.TCustomSQLDataSet.PSGetDefaultOrder 20178
SqlExpr.TCustomSQLDataSet.PSGetIndexDefs 20178
SqlExpr.TCustomSQLDataSet.PSGetKeyFieldsW 20179
SqlExpr.TCustomSQLDataSet.PSGetParams 20179
SqlExpr.TCustomSQLDataSet.PSGetQuoteCharW 20179
SqlExpr.TCustomSQLDataSet.PSGetTableNameW 20179
SqlExpr.TCustomSQLDataSet.PSGetUpdateException 20180
SqlExpr.TCustomSQLDataSet.PSInTransaction 20180
SqlExpr.TCustomSQLDataSet.PSIsSQLBased 20180
SqlExpr.TCustomSQLDataSet.PSIsSQLSupported 20181
SqlExpr.TCustomSQLDataSet.PSReset 20181
SqlExpr.TCustomSQLDataSet.PSSetCommandText 20182
SqlExpr.TCustomSQLDataSet.PSSetParams 20182
SqlExpr.TCustomSQLDataSet.PSStartTransaction 20182
SqlExpr.TCustomSQLDataSet.PSUpdateRecord 20183
SqlExpr.TCustomSQLDataSet.RecordCount 20199
SqlExpr.TCustomSQLDataSet.RowsAffected 20200
SqlExpr.TCustomSQLDataSet.SchemaInfo 20200
SqlExpr.TCustomSQLDataSet.SchemaName 20201
SqlExpr.TCustomSQLDataSet.SetBufListSize 20184
SqlExpr.TCustomSQLDataSet.SetCommandText 20184
SqlExpr.TCustomSQLDataSet.SetFCommandText 20184
SqlExpr.TCustomSQLDataSet.SetFieldData 20184
SqlExpr.TCustomSQLDataSet.SetMaxBlobSize 20184
SqlExpr.TCustomSQLDataSet.SetParamsFromCursor 20185
SqlExpr.TCustomSQLDataSet.SetSchemaInfo 20185
SqlExpr.TCustomSQLDataSet.SetSortFieldNames 20186
SqlExpr.TCustomSQLDataSet.SortFieldNames 20201
SqlExpr.TCustomSQLDataSet.SQLConnection 20201
SqlExpr.TCustomSQLDataSet.TransactionLevel 20202
SqlExpr.TCustomSQLDataSet.UpdateIndexDefs 20186
SqlExpr.TErrorMessageSize 20381
SqlExpr.TErrorMessageSize constant 20381
SqlExpr.TFieldDescList 20371
SqlExpr.TFieldDescList type 20371
SqlExpr.TFieldList 20371
SqlExpr.TFieldList type 20371
SqlExpr.TFLDDesc 20205
SqlExpr.TFLDDesc.bCalcField 20210
SqlExpr.TFLDDesc.efldrRights 20210
SqlExpr.TFLDDesc.efldvVchk 20210
SqlExpr.TFLDDesc.FCalcField 20207
SqlExpr.TFLDDesc.FFldNum 20207
SqlExpr.TFLDDesc.FFLDRights 20207
SqlExpr.TFLDDesc.FFldType 20208
SqlExpr.TFLDDesc.FFLDVchk 20208
SqlExpr.TFLDDesc.FLen 20208
SqlExpr.TFLDDesc.FName 20208
SqlExpr.TFLDDesc.FNullOffset 20208
SqlExpr.TFLDDesc.FOffset 20209
SqlExpr.TFLDDesc.FSubType 20209
SqlExpr.TFLDDesc.FUnits1 20209
SqlExpr.TFLDDesc.FUnits2 20209
SqlExpr.TFLDDesc.iFldNum 20211
SqlExpr.TFLDDesc.iFldType 20211
SqlExpr.TFLDDesc.iLen 20211
SqlExpr.TFLDDesc.iNullOffset 20211
SqlExpr.TFLDDesc.iOffset 20211
SqlExpr.TFLDDesc.iSubType 20212
SqlExpr.TFLDDesc.iUnits1 20212
SqlExpr.TFLDDesc.iUnits2 20212
SqlExpr.TFLDDesc.szName 20212
SqlExpr.TGetDriverFunc 20372
SqlExpr.TGetDriverFunc type 20372
SqlExpr.TLocale 20372

SqlExpr.TLocale type 20372
SqlExpr.TLocaleCode 20372
SqlExpr.TLocaleCode type 20372
SqlExpr.TParseInsertSqlEvent 20373
SqlExpr.TParseInsertSqlEvent type 20373
SqlExpr.TParseSqlEvent 20373
SqlExpr.TParseSqlEvent type 20373
SqlExpr.TSchemaType 20366
SqlExpr.TSchemaType enumeration 20366
SqlExpr.TSQLBlobStream 20213
SqlExpr.TSQLBlobStream.Create 20215
SqlExpr.TSQLBlobStream.Destroy 20216
SqlExpr.TSQLBlobStream.ReadBlobData 20216
SqlExpr.TSQLConnection 20217
SqlExpr.TSQLConnection.ActiveStatements 20242
SqlExpr.TSQLConnection.AfterConnect 20258
SqlExpr.TSQLConnection.AfterDisconnect 20258
SqlExpr.TSQLConnection.AutoClone 20243
SqlExpr.TSQLConnection.BeforeConnect 20258
SqlExpr.TSQLConnection.BeforeDisconnect 20259
SqlExpr.TSQLConnection.BeginTransaction 20224
SqlExpr.TSQLConnection.CheckConnection 20225
SqlExpr.TSQLConnection.CheckDisconnect 20225
SqlExpr.TSQLConnection.CloneConnection 20225
SqlExpr.TSQLConnection.CloseDataSets 20225
SqlExpr.TSQLConnection.Commit 20226
SqlExpr.TSQLConnection.CommitAndFree 20226
SqlExpr.TSQLConnection.Connected 20243
SqlExpr.TSQLConnection.Connection 20244
SqlExpr.TSQLConnection.ConnectionName 20244
SqlExpr.TSQLConnection.ConnectionOptions 20227
SqlExpr.TSQLConnection.ConnectionRegistryFile 20245
SqlExpr.TSQLConnection.ConnectionState 20245
SqlExpr.TSQLConnection.Create 20222
SqlExpr.TSQLConnection.DataSets 20245
SqlExpr.TSQLConnection.Destroy 20222
SqlExpr.TSQLConnection.DoConnect 20227
SqlExpr.TSQLConnection.DoDisconnect 20227
SqlExpr.TSQLConnection.DriverName 20246
SqlExpr.TSQLConnection.DriverRegistryFile 20247
SqlExpr.TSQLConnection.Execute 20227
SqlExpr.TSQLConnection.ExecuteDirect 20228
SqlExpr.TSQLConnection.GetConnected 20229
SqlExpr.TSQLConnection.GetDataSet 20229
SqlExpr.TSQLConnection.GetDefaultSchemaName 20229
SqlExpr.TSQLConnection.GetDriverFunc 20247
SqlExpr.TSQLConnection.GetFieldNames 20230
SqlExpr.TSQLConnection.GetIndexNames 20231
SqlExpr.TSQLConnection.GetLoginUsername 20231
SqlExpr.TSQLConnection.GetPackageNames 20231
SqlExpr.TSQLConnection.GetProcedureNames 20232
SqlExpr.TSQLConnection.GetProcedureParams 20233
SqlExpr.TSQLConnection.GetSchemaNames 20234
SqlExpr.TSQLConnection.GetTableNames 20235
SqlExpr.TSQLConnection.InTransaction 20247
SqlExpr.TSQLConnection.KeepConnection 20248
SqlExpr.TSQLConnection.LastError 20248
SqlExpr.TSQLConnection.LibraryName 20248
SqlExpr.TSQLConnection.Loaded 20235
SqlExpr.TSQLConnection.LoadParamsFromIniFile 20236
SqlExpr.TSQLConnection.LoadParamsOnConnect 20249
SqlExpr.TSQLConnection.LocaleCode 20250
SqlExpr.TSQLConnection.LoginPrompt 20250
SqlExpr.TSQLConnection.MaxStmtsPerConn 20251
SqlExpr.TSQLConnection.MetaData 20252
SqlExpr.TSQLConnection.MultipleTransactionsSupported 20253
SqlExpr.TSQLConnection.Notification 20237
SqlExpr.TSQLConnection.OnLogin 20259
SqlExpr.TSQLConnection.OpenSchema 20237, 20238
SqlExpr.TSQLConnection.Params 20253
SqlExpr.TSQLConnection.ParamsLoaded 20254
SqlExpr.TSQLConnection.QuoteChar 20254
SqlExpr.TSQLConnection.RegisterClient 20238
SqlExpr.TSQLConnection.Rollback 20238
SqlExpr.TSQLConnection.RollbackAndFree 20239
SqlExpr.TSQLConnection.SetTraceEvent 20239
SqlExpr.TSQLConnection.SQLConnection 20255
SqlExpr.TSQLConnection.SQLDllHandle 20255

SqlExpr.TSQLConnection.SQLError 20240
SqlExpr.TSQLConnection.SQLHourGlass 20255
SqlExpr.TSQLConnection.StartTransaction 20240
SqlExpr.TSQLConnection.TableScope 20255
SqlExpr.TSQLConnection.TraceCallbackEvent 20260
SqlExpr.TSQLConnection.TransactionsSupported 20256
SqlExpr.TSQLConnection.UnRegisterClient 20241
SqlExpr.TSQLConnection.VendorLib 20257
SqlExpr.TSQLConnectionLoginEvent 20373
SqlExpr.TSQLConnectionLoginEvent type 20373
SqlExpr.TSQLDataLink 20260
SqlExpr.TSQLDataLink.ActiveChanged 20263
SqlExpr.TSQLDataLink.CheckBrowseMode 20264
SqlExpr.TSQLDataLink.Create 20263
SqlExpr.TSQLDataLink.GetDetailDataSet 20264
SqlExpr.TSQLDataLink.RecordChanged 20264
SqlExpr.TSQLDataSet 20265
SqlExpr.TSQLDataSet.CommandText 20279
SqlExpr.TSQLDataSet.CommandType 20280
SqlExpr.TSQLDataSet.Create 20278
SqlExpr.TSQLDataSet.DataSource 20281
SqlExpr.TSQLDataSet.ExecSQL 20278
SqlExpr.TSQLDataSet.MaxBlobSize 20282
SqlExpr.TSQLDataSet.ParamCheck 20282
SqlExpr.TSQLDataSet.Params 20283
SqlExpr.TSQLDataSet.SortFieldNames 20284
SqlExpr.TSQLDataSet.SQLConnection 20284
SqlExpr.TSQLExceptionType 20367
SqlExpr.TSQLExceptionType enumeration 20367
SqlExpr.TSQLMonitor 20285
SqlExpr.TSQLMonitor.Active 20293
SqlExpr.TSQLMonitor.AutoSave 20294
SqlExpr.TSQLMonitor.Create 20288
SqlExpr.TSQLMonitor.Destroy 20289
SqlExpr.TSQLMonitor.FileName 20294
SqlExpr.TSQLMonitor.InvokeCallBack 20290
SqlExpr.TSQLMonitor.LoadFromFile 20290
SqlExpr.TSQLMonitor.MaxTraceCount 20295
SqlExpr.TSQLMonitor.OnLogTrace 20297
SqlExpr.TSQLMonitor.OnTrace 20297
SqlExpr.TSQLMonitor.SaveToFile 20290
SqlExpr.TSQLMonitor.SetActive 20291
SqlExpr.TSQLMonitor.SetFileName 20291
SqlExpr.TSQLMonitor.SetSQLConnection 20291
SqlExpr.TSQLMonitor.SetStreamedActive 20291
SqlExpr.TSQLMonitor.SetTraceList 20292
SqlExpr.TSQLMonitor.SQLConnection 20295
SqlExpr.TSQLMonitor.SwitchConnection 20292
SqlExpr.TSQLMonitor.Trace 20292
SqlExpr.TSQLMonitor.TraceCount 20295
SqlExpr.TSQLMonitor.TraceList 20296
SqlExpr.TSQLMonitor.UpdateTraceCallBack 20292
SqlExpr.TSQLQuery 20298
SqlExpr.TSQLQuery.Create 20310
SqlExpr.TSQLQuery.DataSource 20313
SqlExpr.TSQLQuery.Destroy 20311
SqlExpr.TSQLQuery.ExecSQL 20311
SqlExpr.TSQLQuery.GetMetadata 20313
SqlExpr.TSQLQuery.MaxBlobSize 20314
SqlExpr.TSQLQuery.ParamCheck 20314
SqlExpr.TSQLQuery.Params 20315
SqlExpr.TSQLQuery.PrepareStatement 20312
SqlExpr.TSQLQuery.RowsAffected 20316
SqlExpr.TSQLQuery.SQL 20316
SqlExpr.TSQLQuery.SQLConnection 20317
SqlExpr.TSQLQuery.Text 20318
SqlExpr.TSQLSchemaInfo 20368
SqlExpr.TSQLSchemaInfo record 20368
SqlExpr.TSQLStoredProc 20318
SqlExpr.TSQLStoredProc.Create 20331
SqlExpr.TSQLStoredProc.ExecProc 20331
SqlExpr.TSQLStoredProc.MaxBlobSize 20333
SqlExpr.TSQLStoredProc.NextRecordSet 20332
SqlExpr.TSQLStoredProc.PackageName 20334
SqlExpr.TSQLStoredProc.ParamCheck 20334
SqlExpr.TSQLStoredProc.Params 20334
SqlExpr.TSQLStoredProc.PrepareStatement 20332
SqlExpr.TSQLStoredProc.SQLConnection 20335

SqlExpr.TSQLStoredProc.StoredProcName 20336
SqlExpr.TSQLTable 20337
SqlExpr.TSQLTable.Active 20353
SqlExpr.TSQLTable.Create 20349
SqlExpr.TSQLTable.DeleteRecords 20350
SqlExpr.TSQLTable.Destroy 20350
SqlExpr.TSQLTable.GetIndexNames 20351
SqlExpr.TSQLTable.IndexFieldCount 20354
SqlExpr.TSQLTable.IndexFieldNames 20355
SqlExpr.TSQLTable.IndexFields 20355
SqlExpr.TSQLTable.IndexName 20356
SqlExpr.TSQLTable.MasterFields 20356
SqlExpr.TSQLTable.MasterLink 20357
SqlExpr.TSQLTable.MasterSource 20357
SqlExpr.TSQLTable.MaxBlobSize 20358
SqlExpr.TSQLTable.OpenCursor 20351
SqlExpr.TSQLTable.PrepareStatement 20351
SqlExpr.TSQLTable.SetIndexField 20352
SqlExpr.TSQLTable.SQLConnection 20358
SqlExpr.TSQLTable.TableName 20359
SqlExpr.TTableScope 20368
SqlExpr.TTableScope enumeration 20368
SqlExpr.TTableScopes 20374
SqlExpr.TTableScopes type 20374
SqlExpr.TTraceEvent 20374
SqlExpr.TTraceEvent type 20374
SqlExpr.TTraceLogEvent 20374
SqlExpr.TTraceLogEvent type 20374
SqlExpr.TTransactionDesc 20369
SqlExpr.TTransactionDesc record 20369
SqlExpr.TTransactionItem 20359
SqlExpr.TTransactionItem.FNext 20360
SqlExpr.TTransactionItem.FTransaction 20360
SqlExpr.TTransactionItem.FTransactionDesc 20360
SqlExpr.TTransIsolationLevel 20369
SqlExpr.TTransIsolationLevel enumeration 20369
SqlStateCode class 2079
about SqlStateCode class 2079
Sql02000 2082
Sql08001 2082
Sql08003 2082
Sql08004 2083
Sql08006 2083
Sql0A000 2083
Sql21000 2083
Sql22000 2083
Sql22001 2084
Sql22003 2084
Sql22005 2084
Sql22007 2084
Sql22011 2084
Sql22012 2085
Sql22019 2085
Sql22023 2085
Sql22025 2085
Sql22027 2085
Sql23000 2086
Sql24000 2086
Sql42000 2086
Sql44000 2086
SqlHz309 2086
SqlHz316 2087
SqlHz320 2087
SqlIm000 2087
SqlStateCode fields 2081
SqlStateCode Members 2081
SqlTimSt 20383
SqlTimSt namespace 20383
Constants 20395
Functions 20385
Structs, Records, Enums 20392
Types 20394
SqlTimSt.CheckSqlTimeStamp 20385
SqlTimSt.CheckSqlTimeStamp function 20385
SqlTimSt.DateTimeToSQLTimeStamp 20386
SqlTimSt.DateTimeToSQLTimeStamp function 20386
SqlTimSt.NullSQLTimeStamp 20395
SqlTimSt.NullSQLTimeStamp constant 20395

SqlTimSt.PSQLTimeStamp 20394
SqlTimSt.PSQLTimeStamp type 20394
SqlTimSt.SQLDayOfWeek 20386
SqlTimSt.SQLDayOfWeek function 20386
SqlTimSt.SQLTimeStampToDateTime 20387
SqlTimSt.SQLTimeStampToDateTime function 20387
SqlTimSt.SQLTimeStampToStr 20387
SqlTimSt.SQLTimeStampToStr function 20387
SqlTimSt.StrToSQLTimeStamp 20388
SqlTimSt.StrToSQLTimeStamp function 20388
SqlTimSt.TryStrToSQLTimeStamp 20388
SqlTimSt.TryStrToSQLTimeStamp function 20388
SqlTimSt.TSQLTimeStamp 20392
SqlTimSt.TSQLTimeStamp record 20392
SqlTimSt.VarIsSQLTimeStamp 20389
SqlTimSt.VarIsSQLTimeStamp function 20389
SqlTimSt.VarSQLTimeStamp 20389
SqlTimSt.VarSQLTimeStamp function 20389
SqlTimSt.VarSQLTimeStampCreate 20390
SqlTimSt.VarSQLTimeStampCreate function 20390
SqlTimSt.VarToSQLTimeStamp 20391
SqlTimSt.VarToSQLTimeStamp function 20391
ssAbsolute enumeration member 13271
ssAuto enumeration member 13271
ssBoth enumeration member 21950
ssFlat enumeration member 14366
ssHorizontal enumeration member 21950
ssHotTrack enumeration member 14366
ssNone enumeration member 21950
ssPercent enumeration member 13271
ssRegular enumeration member 14366
ssStarted enumeration member 19035
ssTerminated enumeration member 19035
ssTimeOut enumeration member 19035
ssVertical enumeration member 21950
st1D enumeration member 17549
StaticArray structure 22245
    [] 22247
    [] operator 22246
    about StaticArray structure 22245
    const T* 22247
    data 22246
    StaticArray data members 22246
    StaticArray Members 22246
    StaticArray operators 22246
    T* 22247
StatusLogColumns class 2087
    about StatusLogColumns class 2087
    Id 2088
    LastModified 2088
    Length 2089
    Name 2089
    StatusLogColumns fields 2088
    StatusLogColumns Members 2088
stAuto enumeration member 22049
stBlue enumeration member 14756
stBoot enumeration member 22049
stBoth enumeration member 5465, 15792
stCircle enumeration member 13270
stClosed enumeration member 1302
stColumns enumeration member 20366
stCombo enumeration member 14756
stConnecting enumeration member 1302
StdActnMenus 20397
StdActnMenus namespace 20397
    Classes 20398
    Functions 20504
StdActnMenus.RegisterStandardMenus 20504
StdActnMenus.RegisterStandardMenus function 20504
StdActnMenus.TStandardAddRemoveItem 20398
StdActnMenus.TStandardAddRemoveItem.DrawBackground 20407
StdActnMenus.TStandardAddRemoveItem.DrawGlyph 20408
StdActnMenus.TStandardButtonControl 20408
StdActnMenus.TStandardButtonControl.DrawFrame 20417
StdActnMenus.TStandardButtonControl.DrawGlyph 20418
StdActnMenus.TStandardButtonControl.DrawText 20418
StdActnMenus.TStandardCustomizePopup 20418
StdActnMenus.TStandardCustomizePopup.CanAutoSize 20433

StdActnMenus.TStandardCustomizePopup.NCPaint 20434
StdActnMenus.TStandardCustomizePopup.PositionPopup 20434
StdActnMenus.TStandardDropDownButton 20434
StdActnMenus.TStandardDropDownButton.DrawFrame 20444
StdActnMenus.TStandardDropDownButton.GetPopupClass 20444
StdActnMenus.TStandardMenuButton 20444
StdActnMenus.TStandardMenuButton.DrawBackground 20454
StdActnMenus.TStandardMenuButton.DrawDesignFocus 20454
StdActnMenus.TStandardMenuButton.DrawFrame 20454
StdActnMenus.TStandardMenuButton.DrawText 20455
StdActnMenus.TStandardMenuExpandBtn 20455
StdActnMenus.TStandardMenuExpandBtn.DrawBackground 20464
StdActnMenus.TStandardMenuExpandBtn.DrawFrame 20465
StdActnMenus.TStandardMenuItem 20465
StdActnMenus.TStandardMenuItem.CMTextchanged 20475
StdActnMenus.TStandardMenuItem.DrawBackground 20475
StdActnMenus.TStandardMenuItem.DrawDesignFocus 20475
StdActnMenus.TStandardMenuItem.DrawEdge 20476
StdActnMenus.TStandardMenuItem.DrawGlyph 20476
StdActnMenus.TStandardMenuItem.DrawGlyphFrame 20476
StdActnMenus.TStandardMenuItem.DrawLargeGlyph 20476
StdActnMenus.TStandardMenuItem.DrawSeparator 20477
StdActnMenus.TStandardMenuItem.DrawText 20477
StdActnMenus.TStandardMenuPopup 20478
StdActnMenus.TStandardMenuPopup.CanAutoSize 20492
StdActnMenus.TStandardMenuPopup.GetExpandBtnClass 20493
StdActnMenus.TStandardMenuPopup.NCPaint 20493
StdActnMenus.TStandardMenuPopup.PositionPopup 20493
StdActnMenus.TStandardToolScrollBtn 20494
StdActnMenus.TStandardToolScrollBtn.DrawFrame 20503
StdActns 20505
StdActns namespace 20505
- Classes 20507
- Structs, Records, Enums 20822
- Types 20824

StdActns.TBrowseForFolder 20508
StdActns.TBrowseForFolder.BeforeExecute 20519
StdActns.TBrowseForFolder.BrowseOptions 20514
StdActns.TBrowseForFolder.Caption 20514
StdActns.TBrowseForFolder.DialogCaption 20515
StdActns.TBrowseForFolder.DoCallback 20513
StdActns.TBrowseForFolder.DoValidate 20513
StdActns.TBrowseForFolder.Enabled 20515
StdActns.TBrowseForFolder.ExecuteTarget 20513
StdActns.TBrowseForFolder.Folder 20515
StdActns.TBrowseForFolder.GetDialogClass 20514
StdActns.TBrowseForFolder.HelpContext 20516
StdActns.TBrowseForFolder.HelpKeyword 20516
StdActns.TBrowseForFolder.HelpType 20517
StdActns.TBrowseForFolder.Hint 20517
StdActns.TBrowseForFolder.ImageIndex 20517
StdActns.TBrowseForFolder.OnAccept 20520
StdActns.TBrowseForFolder.OnCallback 20520
StdActns.TBrowseForFolder.OnCancel 20520
StdActns.TBrowseForFolder.OnHint 20520
StdActns.TBrowseForFolder.OnValidate 20521
StdActns.TBrowseForFolder.RootDir 20518
StdActns.TBrowseForFolder.SecondaryShortCuts 20518
StdActns.TBrowseForFolder.ShortCut 20518
StdActns.TBrowseForFolder.Visible 20519
StdActns.TBrowseForFolderCallbackEvent 20824
StdActns.TBrowseForFolderCallbackEvent type 20824
StdActns.TBrowseForFolderOption 20822
StdActns.TBrowseForFolderOption enumeration 20822
StdActns.TBrowseOptions 20824
StdActns.TBrowseOptions type 20824
StdActns.TColorSelect 20521
StdActns.TColorSelect.BeforeExecute 20531
StdActns.TColorSelect.Caption 20526
StdActns.TColorSelect.Dialog 20527
StdActns.TColorSelect.Enabled 20527
StdActns.TColorSelect.GetDialogClass 20526
StdActns.TColorSelect.HelpContext 20527
StdActns.TColorSelect.HelpKeyword 20528
StdActns.TColorSelect.HelpType 20528
StdActns.TColorSelect.Hint 20529

StdActns.TColorSelect.ImageIndex 20529
StdActns.TColorSelect.OnAccept 20531
StdActns.TColorSelect.OnCancel 20532
StdActns.TColorSelect.OnHint 20532
StdActns.TColorSelect.OnUpdate 20532
StdActns.TColorSelect.SecondaryShortCuts 20530
StdActns.TColorSelect.ShortCut 20530
StdActns.TColorSelect.Visible 20530
StdActns.TCommonDialogAction 20533
StdActns.TCommonDialogAction.BeforeExecute 20543
StdActns.TCommonDialogAction.Create 20538
StdActns.TCommonDialogAction.Destroy 20538
StdActns.TCommonDialogAction.DoAccept 20540
StdActns.TCommonDialogAction.DoCancel 20540
StdActns.TCommonDialogAction.ExecuteResult 20542
StdActns.TCommonDialogAction.ExecuteTarget 20540
StdActns.TCommonDialogAction.FDialog 20539
StdActns.TCommonDialogAction.GetDialogClass 20541
StdActns.TCommonDialogAction.HandlesTarget 20541
StdActns.TCommonDialogAction.Notification 20541
StdActns.TCommonDialogAction.OnAccept 20543
StdActns.TCommonDialogAction.OnCancel 20543
StdActns.TCommonDialogAction.SetupDialog 20542
StdActns.TCommonDialogClass 20825
StdActns.TCommonDialogClass type 20825
StdActns.TEditAction 20544
StdActns.TEditAction.Control 20551
StdActns.TEditAction.Destroy 20549
StdActns.TEditAction.GetControl 20549
StdActns.TEditAction.HandlesTarget 20550
StdActns.TEditAction.Notification 20550
StdActns.TEditAction.UpdateTarget 20551
StdActns.TEditCopy 20552
StdActns.TEditCopy.ExecuteTarget 20557
StdActns.TEditCut 20557
StdActns.TEditCut.ExecuteTarget 20563
StdActns.TEditDelete 20563
StdActns.TEditDelete.ExecuteTarget 20569
StdActns.TEditDelete.UpdateTarget 20569
StdActns.TEditPaste 20570
StdActns.TEditPaste.ExecuteTarget 20575
StdActns.TEditPaste.UpdateTarget 20575
StdActns.TEditSelectAll 20576
StdActns.TEditSelectAll.ExecuteTarget 20581
StdActns.TEditSelectAll.UpdateTarget 20582
StdActns.TEditUndo 20582
StdActns.TEditUndo.ExecuteTarget 20587
StdActns.TEditUndo.UpdateTarget 20588
StdActns.TFileAction 20588
StdActns.TFileAction.FileName 20593
StdActns.TFileAction.GetDialog 20593
StdActns.TFileExit 20594
StdActns.TFileExit.Caption 20600
StdActns.TFileExit.Enabled 20600
StdActns.TFileExit.ExecuteTarget 20598
StdActns.TFileExit.HandlesTarget 20599
StdActns.TFileExit.HelpContext 20600
StdActns.TFileExit.HelpKeyword 20601
StdActns.TFileExit.HelpType 20601
StdActns.TFileExit.Hint 20602
StdActns.TFileExit.ImageIndex 20602
StdActns.TFileExit.OnHint 20604
StdActns.TFileExit.OnUpdate 20604
StdActns.TFileExit.SecondaryShortCuts 20602
StdActns.TFileExit.ShortCut 20603
StdActns.TFileExit.Visible 20603
StdActns.TFileOpen 20605
StdActns.TFileOpen.BeforeExecute 20617
StdActns.TFileOpen.Caption 20612
StdActns.TFileOpen.Create 20610
StdActns.TFileOpen.Dialog 20612
StdActns.TFileOpen.Enabled 20612
StdActns.TFileOpen.ExecuteTarget 20610
StdActns.TFileOpen.GetDialogClass 20611
StdActns.TFileOpen.HelpContext 20613
StdActns.TFileOpen.HelpKeyword 20613
StdActns.TFileOpen.HelpType 20614
StdActns.TFileOpen.Hint 20614

StdActns.TFileOpen.ImageIndex 20615
StdActns.TFileOpen.OnAccept 20617
StdActns.TFileOpen.OnCancel 20617
StdActns.TFileOpen.OnHint 20617
StdActns.TFileOpen.OnUpdate 20618
StdActns.TFileOpen.SecondaryShortCuts 20615
StdActns.TFileOpen.ShortCut 20615
StdActns.TFileOpen.UseDefaultApp 20616
StdActns.TFileOpen.Visible 20616
StdActns.TFileOpenWith 20619
StdActns.TFileOpenWith.AfterOpen 20627
StdActns.TFileOpenWith.BeforeExecute 20628
StdActns.TFileOpenWith.ExecuteTarget 20624
StdActns.TFileOpenWith.FileName 20625
StdActns.TFileOpenWith.HelpContext 20626
StdActns.TFileOpenWith.HelpKeyword 20626
StdActns.TFileOpenWith.HelpType 20627
StdActns.TFileOpenWith.OnUpdate 20628
StdActns.TFilePageSetup 20629
StdActns.TFilePageSetup.BeforeExecute 20639
StdActns.TFilePageSetup.Caption 20635
StdActns.TFilePageSetup.Dialog 20635
StdActns.TFilePageSetup.Enabled 20635
StdActns.TFilePageSetup.GetDialogClass 20634
StdActns.TFilePageSetup.HelpContext 20636
StdActns.TFilePageSetup.HelpKeyword 20636
StdActns.TFilePageSetup.HelpType 20637
StdActns.TFilePageSetup.Hint 20637
StdActns.TFilePageSetup.ImageIndex 20638
StdActns.TFilePageSetup.OnAccept 20640
StdActns.TFilePageSetup.OnCancel 20640
StdActns.TFilePageSetup.OnHint 20640
StdActns.TFilePageSetup.OnUpdate 20640
StdActns.TFilePageSetup.SecondaryShortCuts 20638
StdActns.TFilePageSetup.ShortCut 20638
StdActns.TFilePageSetup.Visible 20639
StdActns.TFilePrintSetup 20642
StdActns.TFilePrintSetup.BeforeExecute 20652
StdActns.TFilePrintSetup.Caption 20647
StdActns.TFilePrintSetup.Dialog 20647
StdActns.TFilePrintSetup.Enabled 20648
StdActns.TFilePrintSetup.GetDialogClass 20647
StdActns.TFilePrintSetup.HelpContext 20648
StdActns.TFilePrintSetup.HelpKeyword 20649
StdActns.TFilePrintSetup.HelpType 20649
StdActns.TFilePrintSetup.Hint 20650
StdActns.TFilePrintSetup.ImageIndex 20650
StdActns.TFilePrintSetup.OnAccept 20652
StdActns.TFilePrintSetup.OnCancel 20652
StdActns.TFilePrintSetup.OnHint 20652
StdActns.TFilePrintSetup.OnUpdate 20653
StdActns.TFilePrintSetup.SecondaryShortCuts 20650
StdActns.TFilePrintSetup.ShortCut 20651
StdActns.TFilePrintSetup.Visible 20651
StdActns.TFileSaveAs 20654
StdActns.TFileSaveAs.BeforeExecute 20663
StdActns.TFileSaveAs.Caption 20660
StdActns.TFileSaveAs.Create 20659
StdActns.TFileSaveAs.Dialog 20660
StdActns.TFileSaveAs.Enabled 20661
StdActns.TFileSaveAs.GetDialogClass 20659
StdActns.TFileSaveAs.HelpContext 20661
StdActns.TFileSaveAs.Hint 20661
StdActns.TFileSaveAs.ImageIndex 20662
StdActns.TFileSaveAs.OnAccept 20664
StdActns.TFileSaveAs.OnCancel 20664
StdActns.TFileSaveAs.OnHint 20664
StdActns.TFileSaveAs.OnUpdate 20665
StdActns.TFileSaveAs.SecondaryShortCuts 20662
StdActns.TFileSaveAs.ShortCut 20663
StdActns.TFileSaveAs.Visible 20663
StdActns.TFontEdit 20666
StdActns.TFontEdit.BeforeExecute 20676
StdActns.TFontEdit.Caption 20671
StdActns.TFontEdit.Dialog 20672
StdActns.TFontEdit.Enabled 20672
StdActns.TFontEdit.GetDialogClass 20671
StdActns.TFontEdit.HelpContext 20672

StdActns.TFontEdit.HelpKeyword 20673
StdActns.TFontEdit.HelpType 20673
StdActns.TFontEdit.Hint 20674
StdActns.TFontEdit.ImageIndex 20674
StdActns.TFontEdit.OnAccept 20676
StdActns.TFontEdit.OnCancel 20676
StdActns.TFontEdit.OnHint 20677
StdActns.TFontEdit.OnUpdate 20677
StdActns.TFontEdit.SecondaryShortCuts 20674
StdActns.TFontEdit.ShortCut 20675
StdActns.TFontEdit.Visible 20675
StdActns.THelpAction 20678
StdActns.THelpAction.Create 20683
StdActns.THelpAction.HandlesTarget 20684
StdActns.THelpAction.UpdateTarget 20684
StdActns.THelpContents 20684
StdActns.THelpContents.ExecuteTarget 20690
StdActns.THelpContextAction 20690
StdActns.THelpContextAction.ExecuteTarget 20695
StdActns.THelpContextAction.UpdateTarget 20695
StdActns.THelpOnHelp 20696
StdActns.THelpOnHelp.ExecuteTarget 20701
StdActns.THelpTopicSearch 20701
StdActns.THelpTopicSearch.ExecuteTarget 20707
StdActns.THintAction 20707
StdActns.THintAction.Create 20712
StdActns.THintAction.Hint 20713
StdActns.TPrintDlg 20713
StdActns.TPrintDlg.BeforeExecute 20723
StdActns.TPrintDlg.Caption 20718
StdActns.TPrintDlg.Dialog 20719
StdActns.TPrintDlg.Enabled 20719
StdActns.TPrintDlg.GetDialogClass 20718
StdActns.TPrintDlg.HelpContext 20720
StdActns.TPrintDlg.HelpKeyword 20720
StdActns.TPrintDlg.HelpType 20721
StdActns.TPrintDlg.Hint 20721
StdActns.TPrintDlg.ImageIndex 20721
StdActns.TPrintDlg.OnAccept 20724
StdActns.TPrintDlg.OnCancel 20724
StdActns.TPrintDlg.OnHint 20724
StdActns.TPrintDlg.OnUpdate 20724
StdActns.TPrintDlg.SecondaryShortCuts 20722
StdActns.TPrintDlg.ShortCut 20722
StdActns.TPrintDlg.Visible 20723
StdActns.TSearchAction 20725
StdActns.TSearchAction.Create 20730
StdActns.TSearchAction.Destroy 20731
StdActns.TSearchAction.ExecuteTarget 20732
StdActns.TSearchAction.FControl 20731
StdActns.TSearchAction.FFindFirst 20731
StdActns.TSearchAction.HandlesTarget 20733
StdActns.TSearchAction.Notification 20733
StdActns.TSearchAction.Search 20733
StdActns.TSearchAction.UpdateTarget 20734
StdActns.TSearchFind 20734
StdActns.TSearchFind.BeforeExecute 20745
StdActns.TSearchFind.Caption 20740
StdActns.TSearchFind.Dialog 20740
StdActns.TSearchFind.Enabled 20741
StdActns.TSearchFind.GetDialogClass 20740
StdActns.TSearchFind.HelpContext 20741
StdActns.TSearchFind.HelpKeyword 20742
StdActns.TSearchFind.HelpType 20742
StdActns.TSearchFind.Hint 20743
StdActns.TSearchFind.ImageIndex 20743
StdActns.TSearchFind.OnAccept 20745
StdActns.TSearchFind.OnCancel 20745
StdActns.TSearchFind.OnHint 20745
StdActns.TSearchFind.OnUpdate 20746
StdActns.TSearchFind.SecondaryShortCuts 20743
StdActns.TSearchFind.ShortCut 20744
StdActns.TSearchFind.Visible 20744
StdActns.TSearchFindFirst 20747
StdActns.TSearchFindFirst.Create 20752
StdActns.TSearchFindNext 20753
StdActns.TSearchFindNext.Caption 20759
StdActns.TSearchFindNext.Create 20757

StdActns.TSearchFindNext.Enabled 20760
StdActns.TSearchFindNext.ExecuteTarget 20758
StdActns.TSearchFindNext.HandlesTarget 20758
StdActns.TSearchFindNext.HelpContext 20760
StdActns.TSearchFindNext.HelpKeyword 20761
StdActns.TSearchFindNext.HelpType 20761
StdActns.TSearchFindNext.Hint 20762
StdActns.TSearchFindNext.ImageIndex 20762
StdActns.TSearchFindNext.OnHint 20764
StdActns.TSearchFindNext.OnUpdate 20764
StdActns.TSearchFindNext.SearchFind 20762
StdActns.TSearchFindNext.SecondaryShortCuts 20763
StdActns.TSearchFindNext.ShortCut 20763
StdActns.TSearchFindNext.UpdateTarget 20759
StdActns.TSearchFindNext.Visible 20763
StdActns.TSearchReplace 20765
StdActns.TSearchReplace.BeforeExecute 20777
StdActns.TSearchReplace.Caption 20772
StdActns.TSearchReplace.Dialog 20772
StdActns.TSearchReplace.Enabled 20773
StdActns.TSearchReplace.ExecuteTarget 20771
StdActns.TSearchReplace.GetDialogClass 20771
StdActns.TSearchReplace.HelpContext 20773
StdActns.TSearchReplace.HelpKeyword 20774
StdActns.TSearchReplace.HelpType 20774
StdActns.TSearchReplace.Hint 20775
StdActns.TSearchReplace.ImageIndex 20775
StdActns.TSearchReplace.OnAccept 20777
StdActns.TSearchReplace.OnCancel 20777
StdActns.TSearchReplace.OnHint 20777
StdActns.TSearchReplace.OnUpdate 20778
StdActns.TSearchReplace.SecondaryShortCuts 20775
StdActns.TSearchReplace.ShortCut 20776
StdActns.TSearchReplace.Visible 20776
StdActns.TWindowAction 20779
StdActns.TWindowAction.Create 20784
StdActns.TWindowAction.Form 20786
StdActns.TWindowAction.GetForm 20784
StdActns.TWindowAction.HandlesTarget 20785
StdActns.TWindowAction.Notification 20785
StdActns.TWindowAction.UpdateTarget 20785
StdActns.TWindowArrange 20786
StdActns.TWindowArrange.ExecuteTarget 20791
StdActns.TWindowCascade 20792
StdActns.TWindowCascade.ExecuteTarget 20797
StdActns.TWindowClose 20798
StdActns.TWindowClose.ExecuteTarget 20803
StdActns.TWindowClose.UpdateTarget 20803
StdActns.TWindowMinimizeAll 20804
StdActns.TWindowMinimizeAll.ExecuteTarget 20809
StdActns.TWindowTileHorizontal 20809
StdActns.TWindowTileHorizontal.ExecuteTarget 20815
StdActns.TWindowTileVertical 20815
StdActns.TWindowTileVertical.ExecuteTarget 20820
stData enumeration member 5465, 15792
StdConvs 20827
StdConvs namespace 20827
    Constants 20883
    Functions 20832
    Variables 20834
StdConvs.auAcres 20836
StdConvs.auAcres variable 20836
StdConvs.auAres 20837
StdConvs.auAres variable 20837
StdConvs.auCentares 20837
StdConvs.auCentares variable 20837
StdConvs.auHectares 20837
StdConvs.auHectares variable 20837
StdConvs.auSquareCentimeters 20838
StdConvs.auSquareCentimeters variable 20838
StdConvs.auSquareDecameters 20838
StdConvs.auSquareDecameters variable 20838
StdConvs.auSquareDecimeters 20838
StdConvs.auSquareDecimeters variable 20838
StdConvs.auSquareFeet 20839
StdConvs.auSquareFeet variable 20839
StdConvs.auSquareHectometers 20839
StdConvs.auSquareHectometers variable 20839

wwww

StdConvs.auSquareInches 20839
StdConvs.auSquareInches variable 20839
StdConvs.auSquareKilometers 20840
StdConvs.auSquareKilometers variable 20840
StdConvs.auSquareMeters 20840
StdConvs.auSquareMeters variable 20840
StdConvs.auSquareMiles 20840
StdConvs.auSquareMiles variable 20840
StdConvs.auSquareMillimeters 20841
StdConvs.auSquareMillimeters variable 20841
StdConvs.auSquareRods 20841
StdConvs.auSquareRods variable 20841
StdConvs.auSquareYards 20841
StdConvs.auSquareYards variable 20841
StdConvs.cbArea 20842
StdConvs.cbArea variable 20842
StdConvs.cbDistance 20842
StdConvs.cbDistance variable 20842
StdConvs.cbMass 20842
StdConvs.cbMass variable 20842
StdConvs.cbTemperature 20843
StdConvs.cbTemperature variable 20843
StdConvs.cbTime 20843
StdConvs.cbTime variable 20843
StdConvs.cbVolume 20843
StdConvs.cbVolume variable 20843
StdConvs.CelsiusToFahrenheit 20832
StdConvs.CelsiusToFahrenheit function 20832
StdConvs.CubicMetersPerAcreFoot 20884
StdConvs.CubicMetersPerAcreFoot constant 20884
StdConvs.CubicMetersPerAcreInch 20884
StdConvs.CubicMetersPerAcreInch constant 20884
StdConvs.CubicMetersPerCord 20885
StdConvs.CubicMetersPerCord constant 20885
StdConvs.CubicMetersPerCordFoot 20885
StdConvs.CubicMetersPerCordFoot constant 20885
StdConvs.CubicMetersPerCubicFoot 20885
StdConvs.CubicMetersPerCubicFoot constant 20885
StdConvs.CubicMetersPerCubicInch 20886
StdConvs.CubicMetersPerCubicInch constant 20886
StdConvs.CubicMetersPerCubicMile 20886
StdConvs.CubicMetersPerCubicMile constant 20886
StdConvs.CubicMetersPerCubicYard 20886
StdConvs.CubicMetersPerCubicYard constant 20886
StdConvs.CubicMetersPerUKBucket 20887
StdConvs.CubicMetersPerUKBucket constant 20887
StdConvs.CubicMetersPerUKBushel 20887
StdConvs.CubicMetersPerUKBushel constant 20887
StdConvs.CubicMetersPerUKGallon 20887
StdConvs.CubicMetersPerUKGallon constant 20887
StdConvs.CubicMetersPerUKGill 20888
StdConvs.CubicMetersPerUKGill constant 20888
StdConvs.CubicMetersPerUKOunce 20888
StdConvs.CubicMetersPerUKOunce constant 20888
StdConvs.CubicMetersPerUKPeck 20888
StdConvs.CubicMetersPerUKPeck constant 20888
StdConvs.CubicMetersPerUKPint 20889
StdConvs.CubicMetersPerUKPint constant 20889
StdConvs.CubicMetersPerUKPottle 20889
StdConvs.CubicMetersPerUKPottle constant 20889
StdConvs.CubicMetersPerUKQuart 20889
StdConvs.CubicMetersPerUKQuart constant 20889
StdConvs.CubicMetersPerUSDryBucket 20890
StdConvs.CubicMetersPerUSDryBucket constant 20890
StdConvs.CubicMetersPerUSDryBushel 20890
StdConvs.CubicMetersPerUSDryBushel constant 20890
StdConvs.CubicMetersPerUSDryGallon 20890
StdConvs.CubicMetersPerUSDryGallon constant 20890
StdConvs.CubicMetersPerUSDryPeck 20891
StdConvs.CubicMetersPerUSDryPeck constant 20891
StdConvs.CubicMetersPerUSDryPint 20891
StdConvs.CubicMetersPerUSDryPint constant 20891
StdConvs.CubicMetersPerUSDryQuart 20891
StdConvs.CubicMetersPerUSDryQuart constant 20891
StdConvs.CubicMetersPerUSFluidCup 20892
StdConvs.CubicMetersPerUSFluidCup constant 20892
StdConvs.CubicMetersPerUSFluidGallon 20892
StdConvs.CubicMetersPerUSFluidGallon constant 20892

StdConvs.CubicMetersPerUSFluidGill 20892
StdConvs.CubicMetersPerUSFluidGill constant 20892
StdConvs.CubicMetersPerUSFluidOunce 20893
StdConvs.CubicMetersPerUSFluidOunce constant 20893
StdConvs.CubicMetersPerUSFluidPint 20893
StdConvs.CubicMetersPerUSFluidPint constant 20893
StdConvs.CubicMetersPerUSFluidQuart 20893
StdConvs.CubicMetersPerUSFluidQuart constant 20893
StdConvs.CubicMetersPerUSFluidTablespoon 20894
StdConvs.CubicMetersPerUSFluidTablespoon constant 20894
StdConvs.CubicMetersPerUSFluidTeaspoon 20894
StdConvs.CubicMetersPerUSFluidTeaspoon constant 20894
StdConvs.duAngstroms 20844
StdConvs.duAngstroms variable 20844
StdConvs.duAstronomicalUnits 20844
StdConvs.duAstronomicalUnits variable 20844
StdConvs.duCentimeters 20844
StdConvs.duCentimeters variable 20844
StdConvs.duChains 20845
StdConvs.duChains variable 20845
StdConvs.duCubits 20845
StdConvs.duCubits variable 20845
StdConvs.duDecameters 20845
StdConvs.duDecameters variable 20845
StdConvs.duDecimeters 20846
StdConvs.duDecimeters variable 20846
StdConvs.duFathoms 20846
StdConvs.duFathoms variable 20846
StdConvs.duFeet 20846
StdConvs.duFeet variable 20846
StdConvs.duFurlongs 20847
StdConvs.duFurlongs variable 20847
StdConvs.duGigameters 20847
StdConvs.duGigameters variable 20847
StdConvs.duHands 20847
StdConvs.duHands variable 20847
StdConvs.duHectometers 20848
StdConvs.duHectometers variable 20848
StdConvs.duInches 20848
StdConvs.duInches variable 20848
StdConvs.duKilometers 20848
StdConvs.duKilometers variable 20848
StdConvs.duLightYears 20849
StdConvs.duLightYears variable 20849
StdConvs.duLinks 20849
StdConvs.duLinks variable 20849
StdConvs.duMegameters 20849
StdConvs.duMegameters variable 20849
StdConvs.duMeters 20850
StdConvs.duMeters variable 20850
StdConvs.duMicromicrons 20850
StdConvs.duMicromicrons variable 20850
StdConvs.duMicrons 20850
StdConvs.duMicrons variable 20850
StdConvs.duMiles 20851
StdConvs.duMiles variable 20851
StdConvs.duMillimeters 20851
StdConvs.duMillimeters variable 20851
StdConvs.duMillimicrons 20851
StdConvs.duMillimicrons variable 20851
StdConvs.duNauticalMiles 20852
StdConvs.duNauticalMiles variable 20852
StdConvs.duPaces 20852
StdConvs.duPaces variable 20852
StdConvs.duParsecs 20852
StdConvs.duParsecs variable 20852
StdConvs.duPicas 20853
StdConvs.duPicas variable 20853
StdConvs.duPoints 20853
StdConvs.duPoints variable 20853
StdConvs.duRods 20853
StdConvs.duRods variable 20853
StdConvs.duYards 20854
StdConvs.duYards variable 20854
StdConvs.FahrenheitToCelsius 20832
StdConvs.FahrenheitToCelsius function 20832
StdConvs.GramsPerDrams 20894
StdConvs.GramsPerDrams constant 20894

StdConvs.GramsPerGrains 20895
StdConvs.GramsPerGrains constant 20895
StdConvs.GramsPerLongTons 20895
StdConvs.GramsPerLongTons constant 20895
StdConvs.GramsPerOunces 20895
StdConvs.GramsPerOunces constant 20895
StdConvs.GramsPerPound 20896
StdConvs.GramsPerPound constant 20896
StdConvs.GramsPerStones 20896
StdConvs.GramsPerStones constant 20896
StdConvs.GramsPerTons 20896
StdConvs.GramsPerTons constant 20896
StdConvs.InitStdConvs 20833
StdConvs.InitStdConvs function 20833
StdConvs.MetersPerAstronomicalUnit 20897
StdConvs.MetersPerAstronomicalUnit constant 20897
StdConvs.MetersPerChain 20897
StdConvs.MetersPerChain constant 20897
StdConvs.MetersPerCubit 20897
StdConvs.MetersPerCubit constant 20897
StdConvs.MetersPerFathom 20898
StdConvs.MetersPerFathom constant 20898
StdConvs.MetersPerFoot 20898
StdConvs.MetersPerFoot constant 20898
StdConvs.MetersPerFurlong 20898
StdConvs.MetersPerFurlong constant 20898
StdConvs.MetersPerHand 20899
StdConvs.MetersPerHand constant 20899
StdConvs.MetersPerInch 20899
StdConvs.MetersPerInch constant 20899
StdConvs.MetersPerLightSecond 20899
StdConvs.MetersPerLightSecond constant 20899
StdConvs.MetersPerLightYear 20900
StdConvs.MetersPerLightYear constant 20900
StdConvs.MetersPerLink 20900
StdConvs.MetersPerLink constant 20900
StdConvs.MetersPerMile 20900
StdConvs.MetersPerMile constant 20900
StdConvs.MetersPerNauticalMiles 20901
StdConvs.MetersPerNauticalMiles constant 20901
StdConvs.MetersPerPace 20901
StdConvs.MetersPerPace constant 20901
StdConvs.MetersPerParsec 20901
StdConvs.MetersPerParsec constant 20901
StdConvs.MetersPerPica 20902
StdConvs.MetersPerPica constant 20902
StdConvs.MetersPerPoint 20902
StdConvs.MetersPerPoint constant 20902
StdConvs.MetersPerRod 20902
StdConvs.MetersPerRod constant 20902
StdConvs.MetersPerYard 20903
StdConvs.MetersPerYard constant 20903
StdConvs.muCentigrams 20854
StdConvs.muCentigrams variable 20854
StdConvs.muDecagrams 20854
StdConvs.muDecagrams variable 20854
StdConvs.muDecigrams 20855
StdConvs.muDecigrams variable 20855
StdConvs.muDrams 20855
StdConvs.muDrams variable 20855
StdConvs.muGrains 20855
StdConvs.muGrains variable 20855
StdConvs.muGrams 20856
StdConvs.muGrams variable 20856
StdConvs.muHectograms 20856
StdConvs.muHectograms variable 20856
StdConvs.muKilograms 20856
StdConvs.muKilograms variable 20856
StdConvs.muLongTons 20857
StdConvs.muLongTons variable 20857
StdConvs.muMetricTons 20857
StdConvs.muMetricTons variable 20857
StdConvs.muMicrograms 20857
StdConvs.muMicrograms variable 20857
StdConvs.muMilligrams 20858
StdConvs.muMilligrams variable 20858
StdConvs.muNanograms 20858
StdConvs.muNanograms variable 20858

StdConvs.muOunces 20858
StdConvs.muOunces variable 20858
StdConvs.muPounds 20859
StdConvs.muPounds variable 20859
StdConvs.muStones 20859
StdConvs.muStones variable 20859
StdConvs.muTons 20859
StdConvs.muTons variable 20859
StdConvs.SquareMetersPerAcre 20903
StdConvs.SquareMetersPerAcre constant 20903
StdConvs.SquareMetersPerSquareFoot 20903
StdConvs.SquareMetersPerSquareFoot constant 20903
StdConvs.SquareMetersPerSquareInch 20904
StdConvs.SquareMetersPerSquareInch constant 20904
StdConvs.SquareMetersPerSquareMile 20904
StdConvs.SquareMetersPerSquareMile constant 20904
StdConvs.SquareMetersPerSquareRod 20904
StdConvs.SquareMetersPerSquareRod constant 20904
StdConvs.SquareMetersPerSquareYard 20905
StdConvs.SquareMetersPerSquareYard constant 20905
StdConvs.tuCelsius 20860
StdConvs.tuCelsius variable 20860
StdConvs.tuCenturies 20860
StdConvs.tuCenturies variable 20860
StdConvs.tuDateTime 20860
StdConvs.tuDateTime variable 20860
StdConvs.tuDays 20861
StdConvs.tuDays variable 20861
StdConvs.tuDecades 20861
StdConvs.tuDecades variable 20861
StdConvs.tuFahrenheit 20861
StdConvs.tuFahrenheit variable 20861
StdConvs.tuFortnights 20862
StdConvs.tuFortnights variable 20862
StdConvs.tuHours 20862
StdConvs.tuHours variable 20862
StdConvs.tuJulianDate 20862
StdConvs.tuJulianDate variable 20862
StdConvs.tuKelvin 20863
StdConvs.tuKelvin variable 20863
StdConvs.tuMillennia 20863
StdConvs.tuMillennia variable 20863
StdConvs.tuMilliSeconds 20863
StdConvs.tuMilliSeconds variable 20863
StdConvs.tuMinutes 20864
StdConvs.tuMinutes variable 20864
StdConvs.tuModifiedJulianDate 20864
StdConvs.tuModifiedJulianDate variable 20864
StdConvs.tuMonths 20864
StdConvs.tuMonths variable 20864
StdConvs.tuRankine 20865
StdConvs.tuRankine variable 20865
StdConvs.tuReaumur 20865
StdConvs.tuReaumur variable 20865
StdConvs.tuSeconds 20865
StdConvs.tuSeconds variable 20865
StdConvs.tuWeeks 20866
StdConvs.tuWeeks variable 20866
StdConvs.tuYears 20866
StdConvs.tuYears variable 20866
StdConvs.vuAcreFeet 20866
StdConvs.vuAcreFeet variable 20866
StdConvs.vuAcreInches 20867
StdConvs.vuAcreInches variable 20867
StdConvs.vuCentiLiters 20867
StdConvs.vuCentiLiters variable 20867
StdConvs.vuCordFeet 20867
StdConvs.vuCordFeet variable 20867
StdConvs.vuCords 20868
StdConvs.vuCords variable 20868
StdConvs.vuCubicCentimeters 20868
StdConvs.vuCubicCentimeters variable 20868
StdConvs.vuCubicDecameters 20868
StdConvs.vuCubicDecameters variable 20868
StdConvs.vuCubicDecimeters 20869
StdConvs.vuCubicDecimeters variable 20869
StdConvs.vuCubicFeet 20869
StdConvs.vuCubicFeet variable 20869

StdConvs.vuCubicHectometers 20869
StdConvs.vuCubicHectometers variable 20869
StdConvs.vuCubicInches 20870
StdConvs.vuCubicInches variable 20870
StdConvs.vuCubicKilometers 20870
StdConvs.vuCubicKilometers variable 20870
StdConvs.vuCubicMeters 20870
StdConvs.vuCubicMeters variable 20870
StdConvs.vuCubicMiles 20871
StdConvs.vuCubicMiles variable 20871
StdConvs.vuCubicMillimeters 20871
StdConvs.vuCubicMillimeters variable 20871
StdConvs.vuCubicYards 20871
StdConvs.vuCubicYards variable 20871
StdConvs.vuDecaLiters 20872
StdConvs.vuDecaLiters variable 20872
StdConvs.vuDecasteres 20872
StdConvs.vuDecasteres variable 20872
StdConvs.vuDeciLiters 20872
StdConvs.vuDeciLiters variable 20872
StdConvs.vuDecisteres 20873
StdConvs.vuDecisteres variable 20873
StdConvs.vuDryBuckets 20873
StdConvs.vuDryBuckets variable 20873
StdConvs.vuDryBushels 20873
StdConvs.vuDryBushels variable 20873
StdConvs.vuDryGallons 20874
StdConvs.vuDryGallons variable 20874
StdConvs.vuDryPecks 20874
StdConvs.vuDryPecks variable 20874
StdConvs.vuDryPints 20874
StdConvs.vuDryPints variable 20874
StdConvs.vuDryQuarts 20875
StdConvs.vuDryQuarts variable 20875
StdConvs.vuFluidCups 20875
StdConvs.vuFluidCups variable 20875
StdConvs.vuFluidGallons 20875
StdConvs.vuFluidGallons variable 20875
StdConvs.vuFluidGills 20876
StdConvs.vuFluidGills variable 20876
StdConvs.vuFluidOunces 20876
StdConvs.vuFluidOunces variable 20876
StdConvs.vuFluidPints 20876
StdConvs.vuFluidPints variable 20876
StdConvs.vuFluidQuarts 20877
StdConvs.vuFluidQuarts variable 20877
StdConvs.vuFluidTablespoons 20877
StdConvs.vuFluidTablespoons variable 20877
StdConvs.vuFluidTeaspoons 20877
StdConvs.vuFluidTeaspoons variable 20877
StdConvs.vuHectoLiters 20878
StdConvs.vuHectoLiters variable 20878
StdConvs.vuKiloLiters 20878
StdConvs.vuKiloLiters variable 20878
StdConvs.vuLiters 20878
StdConvs.vuLiters variable 20878
StdConvs.vuMilliLiters 20879
StdConvs.vuMilliLiters variable 20879
StdConvs.vuSteres 20879
StdConvs.vuSteres variable 20879
StdConvs.vuUKBuckets 20879
StdConvs.vuUKBuckets variable 20879
StdConvs.vuUKBushels 20880
StdConvs.vuUKBushels variable 20880
StdConvs.vuUKGallons 20880
StdConvs.vuUKGallons variable 20880
StdConvs.vuUKGills 20880
StdConvs.vuUKGills variable 20880
StdConvs.vuUKOunces 20881
StdConvs.vuUKOunces variable 20881
StdConvs.vuUKPecks 20881
StdConvs.vuUKPecks variable 20881
StdConvs.vuUKPints 20881
StdConvs.vuUKPints variable 20881
StdConvs.vuUKPottles 20882
StdConvs.vuUKPottles variable 20882
StdConvs.vuUKQuarts 20882
StdConvs.vuUKQuarts variable 20882

StdCtrls 20907
StdCtrls namespace 20907
Classes 20909
Structs, Records, Enums 21945
Types 21953
StdCtrls.TButton 20909
StdCtrls.TButton.Action 20924
StdCtrls.TButton.Align 20924
StdCtrls.TButton.Anchors 20927
StdCtrls.TButton.BiDiMode 20928
StdCtrls.TButton.Cancel 20929
StdCtrls.TButton.Caption 20929
StdCtrls.TButton.Click 20921
StdCtrls.TButton.Constraints 20932
StdCtrls.TButton.Create 20920
StdCtrls.TButton.CreateParams 20921
StdCtrls.TButton.CreateWnd 20922
StdCtrls.TButton.Default 20932
StdCtrls.TButton.DragCursor 20933
StdCtrls.TButton.DragKind 20934
StdCtrls.TButton.DragMode 20934
StdCtrls.TButton.Enabled 20936
StdCtrls.TButton.Font 20936
StdCtrls.TButton.ModalResult 20937
StdCtrls.TButton.OnClick 20949
StdCtrls.TButton.OnContextPopup 20951
StdCtrls.TButton.OnDragDrop 20951
StdCtrls.TButton.OnDragOver 20954
StdCtrls.TButton.OnEndDock 20956
StdCtrls.TButton.OnEndDrag 20956
StdCtrls.TButton.OnEnter 20957
StdCtrls.TButton.OnExit 20960
StdCtrls.TButton.OnKeyDown 20961
StdCtrls.TButton.OnKeyPress 20964
StdCtrls.TButton.OnKeyUp 20965
StdCtrls.TButton.OnMouseActivate 20967
StdCtrls.TButton.OnMouseDown 20968
StdCtrls.TButton.OnMouseEnter 20970
StdCtrls.TButton.OnMouseLeave 20970
StdCtrls.TButton.OnMouseMove 20971
StdCtrls.TButton.OnMouseUp 20973
StdCtrls.TButton.OnStartDock 20975
StdCtrls.TButton.OnStartDrag 20976
StdCtrls.TButton.ParentBiDiMode 20938
StdCtrls.TButton.ParentFont 20938
StdCtrls.TButton.ParentShowHint 20939
StdCtrls.TButton.PopupMenu 20939
StdCtrls.TButton.SetButtonStyle 20922
StdCtrls.TButton.ShowHint 20944
StdCtrls.TButton.TabOrder 20945
StdCtrls.TButton.TabStop 20946
StdCtrls.TButton.UseRightToLeftAlignment 20923
StdCtrls.TButton.Visible 20946
StdCtrls.TButton.WordWrap 20948
StdCtrls.TButtonActionLink 20976
StdCtrls.TButtonActionLink.AssignClient 20980
StdCtrls.TButtonActionLink.FClient 20979
StdCtrls.TButtonActionLink.IsCheckedLinked 20980
StdCtrls.TButtonActionLink.SetChecked 20980
StdCtrls.TButtonActionLinkClass 21953
StdCtrls.TButtonActionLinkClass type 21953
StdCtrls.TButtonControl 20981
StdCtrls.TButtonControl.ActionChange 20992
StdCtrls.TButtonControl.Checked 20995
StdCtrls.TButtonControl.ClicksDisabled 20996
StdCtrls.TButtonControl.Create 20992
StdCtrls.TButtonControl.CreateParams 20993
StdCtrls.TButtonControl.GetActionLinkClass 20993
StdCtrls.TButtonControl.GetChecked 20993
StdCtrls.TButtonControl.SetChecked 20994
StdCtrls.TButtonControl.WndProc 20994
StdCtrls.TButtonControl.WordWrap 20996
StdCtrls.TCheckBox 20996
StdCtrls.TCheckBox.Action 21008
StdCtrls.TCheckBox.Align 21009
StdCtrls.TCheckBox.Alignment 21012
StdCtrls.TCheckBox.AllowGrayed 21013
StdCtrls.TCheckBox.Anchors 21013

StdCtrls.TCheckBox.BiDiMode 21014
StdCtrls.TCheckBox.Caption 21014
StdCtrls.TCheckBox.Checked 21017
StdCtrls.TCheckBox.Color 21018
StdCtrls.TCheckBox.Constraints 21019
StdCtrls.TCheckBox.Ctl3D 21020
StdCtrls.TCheckBox.DragCursor 21020
StdCtrls.TCheckBox.DragKind 21020
StdCtrls.TCheckBox.DragMode 21021
StdCtrls.TCheckBox.Enabled 21022
StdCtrls.TCheckBox.Font 21023
StdCtrls.TCheckBox.OnClick 21036
StdCtrls.TCheckBox.OnContextPopup 21038
StdCtrls.TCheckBox.OnDragDrop 21039
StdCtrls.TCheckBox.OnDragOver 21041
StdCtrls.TCheckBox.OnEndDock 21044
StdCtrls.TCheckBox.OnEndDrag 21044
StdCtrls.TCheckBox.OnEnter 21045
StdCtrls.TCheckBox.OnExit 21048
StdCtrls.TCheckBox.OnKeyDown 21049
StdCtrls.TCheckBox.OnKeyPress 21052
StdCtrls.TCheckBox.OnKeyUp 21053
StdCtrls.TCheckBox.OnMouseActivate 21055
StdCtrls.TCheckBox.OnMouseDown 21056
StdCtrls.TCheckBox.OnMouseEnter 21058
StdCtrls.TCheckBox.OnMouseLeave 21058
StdCtrls.TCheckBox.OnMouseMove 21058
StdCtrls.TCheckBox.OnMouseUp 21061
StdCtrls.TCheckBox.OnStartDock 21063
StdCtrls.TCheckBox.OnStartDrag 21064
StdCtrls.TCheckBox.ParentBiDiMode 21024
StdCtrls.TCheckBox.ParentColor 21024
StdCtrls.TCheckBox.ParentCtl3D 21025
StdCtrls.TCheckBox.ParentFont 21025
StdCtrls.TCheckBox.ParentShowHint 21026
StdCtrls.TCheckBox.PopupMenu 21026
StdCtrls.TCheckBox.ShowHint 21031
StdCtrls.TCheckBox.State 21032
StdCtrls.TCheckBox.TabOrder 21033
StdCtrls.TCheckBox.TabStop 21034
StdCtrls.TCheckBox.Visible 21034
StdCtrls.TCheckBox.WordWrap 21035
StdCtrls.TCheckBoxState 21945
StdCtrls.TCheckBoxState enumeration 21945
StdCtrls.TComboBox 21064
StdCtrls.TComboBox.Align 21078
StdCtrls.TComboBox.Anchors 21082
StdCtrls.TComboBox.AutoCloseUp 21082
StdCtrls.TComboBox.AutoComplete 21083
StdCtrls.TComboBox.AutoCompleteDelay 21083
StdCtrls.TComboBox.AutoDropDown 21083
StdCtrls.TComboBox.BevelEdges 21084
StdCtrls.TComboBox.BevelInner 21084
StdCtrls.TComboBox.BevelKind 21085
StdCtrls.TComboBox.BevelOuter 21085
StdCtrls.TComboBox.BiDiMode 21085
StdCtrls.TComboBox.CharCase 21086
StdCtrls.TComboBox.Color 21086
StdCtrls.TComboBox.Constraints 21087
StdCtrls.TComboBox.Ctl3D 21088
StdCtrls.TComboBox.DragCursor 21088
StdCtrls.TComboBox.DragKind 21089
StdCtrls.TComboBox.DragMode 21089
StdCtrls.TComboBox.DropDownCount 21091
StdCtrls.TComboBox.Enabled 21091
StdCtrls.TComboBox.Font 21091
StdCtrls.TComboBox.ImeMode 21092
StdCtrls.TComboBox.ImeName 21093
StdCtrls.TComboBox.ItemHeight 21094
StdCtrls.TComboBox.ItemIndex 21094
StdCtrls.TComboBox.Items 21095
StdCtrls.TComboBox.MaxLength 21097
StdCtrls.TComboBox.OnChange 21112
StdCtrls.TComboBox.OnClick 21114
StdCtrls.TComboBox.OnCloseUp 21116
StdCtrls.TComboBox.OnContextPopup 21116
StdCtrls.TComboBox.OnDblClick 21117
StdCtrls.TComboBox.OnDragDrop 21117

StdCtrls.TComboBox.OnDragOver 21120
StdCtrls.TComboBox.OnDrawItem 21122
StdCtrls.TComboBox.OnDropDown 21123
StdCtrls.TComboBox.OnEndDock 21124
StdCtrls.TComboBox.OnEndDrag 21124
StdCtrls.TComboBox.OnEnter 21125
StdCtrls.TComboBox.OnExit 21128
StdCtrls.TComboBox.OnKeyDown 21129
StdCtrls.TComboBox.OnKeyPress 21132
StdCtrls.TComboBox.OnKeyUp 21133
StdCtrls.TComboBox.OnMeasureItem 21135
StdCtrls.TComboBox.OnMouseEnter 21135
StdCtrls.TComboBox.OnMouseLeave 21136
StdCtrls.TComboBox.OnSelect 21136
StdCtrls.TComboBox.OnStartDock 21136
StdCtrls.TComboBox.OnStartDrag 21137
StdCtrls.TComboBox.ParentBiDiMode 21097
StdCtrls.TComboBox.ParentColor 21098
StdCtrls.TComboBox.ParentCtl3D 21098
StdCtrls.TComboBox.ParentFont 21099
StdCtrls.TComboBox.ParentShowHint 21099
StdCtrls.TComboBox.PopupMenu 21100
StdCtrls.TComboBox.ShowHint 21104
StdCtrls.TComboBox.Sorted 21106
StdCtrls.TComboBox.Style 21107
StdCtrls.TComboBox.TabOrder 21108
StdCtrls.TComboBox.TabStop 21108
StdCtrls.TComboBox.Text 21109
StdCtrls.TComboBox.Visible 21110
StdCtrls.TComboBoxStyle 21946
StdCtrls.TComboBoxStyle enumeration 21946
StdCtrls.TCustomCheckBox 21138
StdCtrls.TCustomCheckBox.Alignment 21152
StdCtrls.TCustomCheckBox.AllowGrayed 21152
StdCtrls.TCustomCheckBox.Click 21149
StdCtrls.TCustomCheckBox.Create 21148
StdCtrls.TCustomCheckBox.CreateParams 21149
StdCtrls.TCustomCheckBox.CreateWnd 21150
StdCtrls.TCustomCheckBox.GetChecked 21150
StdCtrls.TCustomCheckBox.GetControlsAlignment 21151
StdCtrls.TCustomCheckBox.SetChecked 21151
StdCtrls.TCustomCheckBox.State 21153
StdCtrls.TCustomCheckBox.TabStop 21154
StdCtrls.TCustomCheckBox.Toggle 21151
StdCtrls.TCustomCombo 21154
StdCtrls.TCustomCombo.AddItem 21170
StdCtrls.TCustomCombo.AdjustDropDown 21171
StdCtrls.TCustomCombo.Canvas 21184
StdCtrls.TCustomCombo.Change 21171
StdCtrls.TCustomCombo.Clear 21172
StdCtrls.TCustomCombo.ClearSelection 21172
StdCtrls.TCustomCombo.CloseUp 21172
StdCtrls.TCustomCombo.ComboWndProc 21173
StdCtrls.TCustomCombo.CopySelection 21173
StdCtrls.TCustomCombo.Create 21166
StdCtrls.TCustomCombo.CreateWnd 21174
StdCtrls.TCustomCombo.DeleteSelected 21174
StdCtrls.TCustomCombo.Destroy 21166
StdCtrls.TCustomCombo.DestroyWindowHandle 21174
StdCtrls.TCustomCombo.DropDown 21175
StdCtrls.TCustomCombo.DropDownCount 21185
StdCtrls.TCustomCombo.DroppedDown 21185
StdCtrls.TCustomCombo.EditHandle 21185
StdCtrls.TCustomCombo.EditWndProc 21175
StdCtrls.TCustomCombo.FDefEditProc 21167
StdCtrls.TCustomCombo.FDefListProc 21167
StdCtrls.TCustomCombo.FDropHandle 21168
StdCtrls.TCustomCombo.FDroppingDown 21168
StdCtrls.TCustomCombo.FEditHandle 21168
StdCtrls.TCustomCombo.FEditInstance 21168
StdCtrls.TCustomCombo.FFocusChanged 21168
StdCtrls.TCustomCombo.FIsFocused 21169
StdCtrls.TCustomCombo.FListHandle 21169
StdCtrls.TCustomCombo.FListInstance 21169
StdCtrls.TCustomCombo.Focused 21175
StdCtrls.TCustomCombo.FSaveIndex 21169
StdCtrls.TCustomCombo.GetCount 21176
StdCtrls.TCustomCombo.GetDroppedDown 21176

StdCtrls.TCustomCombo.GetItemCount 21176
StdCtrls.TCustomCombo.GetItemHt 21177
StdCtrls.TCustomCombo.GetItemIndex 21177
StdCtrls.TCustomCombo.GetItemsClass 21177
StdCtrls.TCustomCombo.GetSelLength 21178
StdCtrls.TCustomCombo.GetSelStart 21178
StdCtrls.TCustomCombo.ItemCount 21186
StdCtrls.TCustomCombo.ItemHeight 21186
StdCtrls.TCustomCombo.Items 21187
StdCtrls.TCustomCombo.ListHandle 21189
StdCtrls.TCustomCombo.ListWndProc 21178
StdCtrls.TCustomCombo.Loaded 21179
StdCtrls.TCustomCombo.MaxLength 21189
StdCtrls.TCustomCombo.OnChange 21197
StdCtrls.TCustomCombo.OnCloseUp 21199
StdCtrls.TCustomCombo.OnDropDown 21199
StdCtrls.TCustomCombo.OnSelect 21201
StdCtrls.TCustomCombo.ParentColor 21190
StdCtrls.TCustomCombo.Select 21179
StdCtrls.TCustomCombo.SelectAll 21180
StdCtrls.TCustomCombo.SelLength 21190
StdCtrls.TCustomCombo.SelStart 21193
StdCtrls.TCustomCombo.SetDropDownCount 21180
StdCtrls.TCustomCombo.SetDroppedDown 21180
StdCtrls.TCustomCombo.SetItemHeight 21181
StdCtrls.TCustomCombo.SetItemIndex 21181
StdCtrls.TCustomCombo.SetItems 21181
StdCtrls.TCustomCombo.SetMaxLength 21182
StdCtrls.TCustomCombo.SetSelLength 21182
StdCtrls.TCustomCombo.SetSelStart 21182
StdCtrls.TCustomCombo.TabStop 21196
StdCtrls.TCustomCombo.WndProc 21183
StdCtrls.TCustomComboBox 21201
StdCtrls.TCustomComboBox.AutoCloseUp 21220
StdCtrls.TCustomComboBox.AutoComplete 21220
StdCtrls.TCustomComboBox.AutoCompleteDelay 21220
StdCtrls.TCustomComboBox.AutoDropDown 21221
StdCtrls.TCustomComboBox.CharCase 21221
StdCtrls.TCustomComboBox.Create 21214
StdCtrls.TCustomComboBox.CreateParams 21215
StdCtrls.TCustomComboBox.CreateWnd 21215
StdCtrls.TCustomComboBox.Destroy 21214
StdCtrls.TCustomComboBox.DestroyWnd 21215
StdCtrls.TCustomComboBox.DrawItem 21216
StdCtrls.TCustomComboBox.GetItemCount 21216
StdCtrls.TCustomComboBox.GetItemHt 21216
StdCtrls.TCustomComboBox.GetItemsClass 21217
StdCtrls.TCustomComboBox.GetSelText 21217
StdCtrls.TCustomComboBox.KeyPress 21217
StdCtrls.TCustomComboBox.MeasureItem 21218
StdCtrls.TCustomComboBox.OnDrawItem 21224
StdCtrls.TCustomComboBox.OnMeasureItem 21225
StdCtrls.TCustomComboBox.SelectItem 21218
StdCtrls.TCustomComboBox.SelText 21222
StdCtrls.TCustomComboBox.SetStyle 21219
StdCtrls.TCustomComboBox.Sorted 21222
StdCtrls.TCustomComboBox.Style 21224
StdCtrls.TCustomComboBox.WndProc 21219
StdCtrls.TCustomComboBoxStrings 21225
StdCtrls.TCustomComboBoxStrings.Clear 21228
StdCtrls.TCustomComboBoxStrings.ComboBox 21232
StdCtrls.TCustomComboBoxStrings.Delete 21229
StdCtrls.TCustomComboBoxStrings.Get 21229
StdCtrls.TCustomComboBoxStrings.GetCount 21230
StdCtrls.TCustomComboBoxStrings.GetObject 21230
StdCtrls.TCustomComboBoxStrings.IndexOf 21230
StdCtrls.TCustomComboBoxStrings.PutObject 21231
StdCtrls.TCustomComboBoxStrings.SetUpdateState 21231
StdCtrls.TCustomComboBoxStringsClass 21953
StdCtrls.TCustomComboBoxStringsClass type 21953
StdCtrls.TCustomEdit 21232
StdCtrls.TCustomEdit.AutoSelect 21260
StdCtrls.TCustomEdit.AutoSize 21260
StdCtrls.TCustomEdit.BorderStyle 21261
StdCtrls.TCustomEdit.CanUndo 21262
StdCtrls.TCustomEdit.Change 21245
StdCtrls.TCustomEdit.CharCase 21262
StdCtrls.TCustomEdit.Clear 21245

StdCtrls.TCustomEdit.ClearSelection 21245
StdCtrls.TCustomEdit.ClearUndo 21246
StdCtrls.TCustomEdit.CopyToClipboard 21246
StdCtrls.TCustomEdit.Create 21244
StdCtrls.TCustomEdit.CreateParams 21248
StdCtrls.TCustomEdit.CreateWindowHandle 21248
StdCtrls.TCustomEdit.CreateWnd 21248
StdCtrls.TCustomEdit.CutToClipboard 21249
StdCtrls.TCustomEdit.DefaultHandler 21250
StdCtrls.TCustomEdit.DestroyWnd 21251
StdCtrls.TCustomEdit.DoSetMaxLength 21251
StdCtrls.TCustomEdit.GetSelLength 21252
StdCtrls.TCustomEdit.GetSelStart 21252
StdCtrls.TCustomEdit.GetSelText 21252
StdCtrls.TCustomEdit.GetSelTextBuf 21253
StdCtrls.TCustomEdit.HideSelection 21262
StdCtrls.TCustomEdit.MaxLength 21263
StdCtrls.TCustomEdit.Modified 21263
StdCtrls.TCustomEdit.OEMConvert 21264
StdCtrls.TCustomEdit.OnChange 21275
StdCtrls.TCustomEdit.ParentColor 21264
StdCtrls.TCustomEdit.PasswordChar 21265
StdCtrls.TCustomEdit.PasteFromClipboard 21254
StdCtrls.TCustomEdit.ReadOnly 21265
StdCtrls.TCustomEdit.SelectAll 21255
StdCtrls.TCustomEdit.SelLength 21265
StdCtrls.TCustomEdit.SelStart 21269
StdCtrls.TCustomEdit.SelText 21272
StdCtrls.TCustomEdit.SetAutoSize 21257
StdCtrls.TCustomEdit.SetSelLength 21257
StdCtrls.TCustomEdit.SetSelStart 21257
StdCtrls.TCustomEdit.SetSelTextBuf 21258
StdCtrls.TCustomEdit.TabStop 21273
StdCtrls.TCustomEdit.Text 21273
StdCtrls.TCustomEdit.Undo 21259
StdCtrls.TCustomGroupBox 21275
StdCtrls.TCustomGroupBox.AdjustClientRect 21287
StdCtrls.TCustomGroupBox.Create 21286
StdCtrls.TCustomGroupBox.CreateParams 21287
StdCtrls.TCustomGroupBox.Paint 21287
StdCtrls.TCustomLabel 21288
StdCtrls.TCustomLabel.AdjustBounds 21296
StdCtrls.TCustomLabel.Alignment 21299
StdCtrls.TCustomLabel.AutoSize 21299
StdCtrls.TCustomLabel.Canvas 21300
StdCtrls.TCustomLabel.Caption 21303
StdCtrls.TCustomLabel.Create 21295
StdCtrls.TCustomLabel.DoDrawText 21296
StdCtrls.TCustomLabel.EllipsisPosition 21305
StdCtrls.TCustomLabel.FocusControl 21306
StdCtrls.TCustomLabel.GetLabelText 21297
StdCtrls.TCustomLabel.Layout 21306
StdCtrls.TCustomLabel.Loaded 21297
StdCtrls.TCustomLabel.Notification 21298
StdCtrls.TCustomLabel.Paint 21298
StdCtrls.TCustomLabel.SetAutoSize 21298
StdCtrls.TCustomLabel.ShowAccelChar 21306
StdCtrls.TCustomLabel.Transparent 21307
StdCtrls.TCustomLabel.WordWrap 21308
StdCtrls.TCustomListBox 21308
StdCtrls.TCustomListBox.AddItem 21322
StdCtrls.TCustomListBox.AutoComplete 21342
StdCtrls.TCustomListBox.AutoCompleteDelay 21342
StdCtrls.TCustomListBox.BorderStyle 21343
StdCtrls.TCustomListBox.Canvas 21343
StdCtrls.TCustomListBox.Clear 21322
StdCtrls.TCustomListBox.ClearSelection 21323
StdCtrls.TCustomListBox.Columns 21346
StdCtrls.TCustomListBox.CopySelection 21323
StdCtrls.TCustomListBox.Count 21346
StdCtrls.TCustomListBox.Create 21320
StdCtrls.TCustomListBox.CreateParams 21324
StdCtrls.TCustomListBox.CreateWnd 21325
StdCtrls.TCustomListBox.DeleteSelected 21325
StdCtrls.TCustomListBox.DeleteString 21326
StdCtrls.TCustomListBox.Destroy 21320
StdCtrls.TCustomListBox.DestroyWnd 21326
StdCtrls.TCustomListBox.DoFindData 21326

gggggg

StdCtrls.TCustomListBox.DoGetData 21327
StdCtrls.TCustomListBox.DoGetDataObject 21327
StdCtrls.TCustomListBox.DragCanceled 21328
StdCtrls.TCustomListBox.DrawItem 21328
StdCtrls.TCustomListBox.ExtendedSelect 21346
StdCtrls.TCustomListBox.FMoving 21321
StdCtrls.TCustomListBox.GetCount 21329
StdCtrls.TCustomListBox.GetItemData 21329
StdCtrls.TCustomListBox.GetItemIndex 21329
StdCtrls.TCustomListBox.GetSelCount 21330
StdCtrls.TCustomListBox.GetSelected 21330
StdCtrls.TCustomListBox.IntegralHeight 21347
StdCtrls.TCustomListBox.InternalGetItemData 21330
StdCtrls.TCustomListBox.InternalSetItemData 21330
StdCtrls.TCustomListBox.ItemAtPos 21331
StdCtrls.TCustomListBox.ItemHeight 21347
StdCtrls.TCustomListBox.ItemRect 21333
StdCtrls.TCustomListBox.Items 21348
StdCtrls.TCustomListBox.KeyPress 21334
StdCtrls.TCustomListBox.LoadRecreateItems 21334
StdCtrls.TCustomListBox.MeasureItem 21335
StdCtrls.TCustomListBox.OnData 21357
StdCtrls.TCustomListBox.OnDataFind 21357
StdCtrls.TCustomListBox.OnDataObject 21358
StdCtrls.TCustomListBox.OnDrawItem 21358
StdCtrls.TCustomListBox.OnMeasureItem 21360
StdCtrls.TCustomListBox.ParentColor 21350
StdCtrls.TCustomListBox.ResetContent 21335
StdCtrls.TCustomListBox.SaveRecreateItems 21336
StdCtrls.TCustomListBox.ScrollWidth 21351
StdCtrls.TCustomListBox.SelectAll 21336
StdCtrls.TCustomListBox.Selected 21351
StdCtrls.TCustomListBox.SetItemData 21336
StdCtrls.TCustomListBox.SetItemIndex 21336
StdCtrls.TCustomListBox.SetMultiSelect 21337
StdCtrls.TCustomListBox.Sorted 21353
StdCtrls.TCustomListBox.Style 21354
StdCtrls.TCustomListBox.TabStop 21355
StdCtrls.TCustomListBox.TabWidth 21355
StdCtrls.TCustomListBox.TopIndex 21356
StdCtrls.TCustomListBox.WndProc 21337
StdCtrls.TCustomMemo 21362
StdCtrls.TCustomMemo.Alignment 21379
StdCtrls.TCustomMemo.CaretPos 21380
StdCtrls.TCustomMemo.Create 21374
StdCtrls.TCustomMemo.CreateParams 21375
StdCtrls.TCustomMemo.CreateWindowHandle 21376
StdCtrls.TCustomMemo.Destroy 21374
StdCtrls.TCustomMemo.GetCaretPos 21376
StdCtrls.TCustomMemo.GetControlsAlignment 21376
StdCtrls.TCustomMemo.KeyPress 21377
StdCtrls.TCustomMemo.Lines 21380
StdCtrls.TCustomMemo.Loaded 21377
StdCtrls.TCustomMemo.ScrollBars 21381
StdCtrls.TCustomMemo.SetAlignment 21377
StdCtrls.TCustomMemo.SetCaretPos 21378
StdCtrls.TCustomMemo.SetLines 21378
StdCtrls.TCustomMemo.SetScrollBars 21378
StdCtrls.TCustomMemo.SetWordWrap 21379
StdCtrls.TCustomMemo.WantReturns 21384
StdCtrls.TCustomMemo.WantTabs 21384
StdCtrls.TCustomMemo.WordWrap 21384
StdCtrls.TCustomStaticText 21387
StdCtrls.TCustomStaticText.Alignment 21400
StdCtrls.TCustomStaticText.AutoSize 21400
StdCtrls.TCustomStaticText.BorderStyle 21401
StdCtrls.TCustomStaticText.Create 21397
StdCtrls.TCustomStaticText.CreateParams 21398
StdCtrls.TCustomStaticText.FocusControl 21401
StdCtrls.TCustomStaticText.Loaded 21399
StdCtrls.TCustomStaticText.Notification 21399
StdCtrls.TCustomStaticText.SetAutoSize 21399
StdCtrls.TCustomStaticText.ShowAccelChar 21401
StdCtrls.TCustomStaticText.Transparent 21402
StdCtrls.TDrawItemEvent 21954
StdCtrls.TDrawItemEvent type 21954
StdCtrls.TEdit 21402
StdCtrls.TEdit.Align 21415

StdCtrls.TEdit.Anchors 21418
StdCtrls.TEdit.AutoSelect 21419
StdCtrls.TEdit.AutoSize 21419
StdCtrls.TEdit.BevelEdges 21420
StdCtrls.TEdit.BevelInner 21421
StdCtrls.TEdit.BevelKind 21421
StdCtrls.TEdit.BevelOuter 21422
StdCtrls.TEdit.BevelWidth 21422
StdCtrls.TEdit.BiDiMode 21423
StdCtrls.TEdit.BorderStyle 21423
StdCtrls.TEdit.CharCase 21423
StdCtrls.TEdit.Color 21424
StdCtrls.TEdit.Constraints 21425
StdCtrls.TEdit.Ctl3D 21425
StdCtrls.TEdit.DragCursor 21426
StdCtrls.TEdit.DragKind 21426
StdCtrls.TEdit.DragMode 21426
StdCtrls.TEdit.Enabled 21428
StdCtrls.TEdit.Font 21428
StdCtrls.TEdit.HideSelection 21429
StdCtrls.TEdit.ImeMode 21430
StdCtrls.TEdit.ImeName 21430
StdCtrls.TEdit.MaxLength 21431
StdCtrls.TEdit.OEMConvert 21432
StdCtrls.TEdit.OnChange 21445
StdCtrls.TEdit.OnClick 21446
StdCtrls.TEdit.OnContextPopup 21448
StdCtrls.TEdit.OnDblClick 21448
StdCtrls.TEdit.OnDragDrop 21449
StdCtrls.TEdit.OnDragOver 21451
StdCtrls.TEdit.OnEndDock 21453
StdCtrls.TEdit.OnEndDrag 21454
StdCtrls.TEdit.OnEnter 21455
StdCtrls.TEdit.OnExit 21457
StdCtrls.TEdit.OnKeyDown 21459
StdCtrls.TEdit.OnKeyPress 21462
StdCtrls.TEdit.OnKeyUp 21463
StdCtrls.TEdit.OnMouseActivate 21464
StdCtrls.TEdit.OnMouseDown 21465
StdCtrls.TEdit.OnMouseEnter 21468
StdCtrls.TEdit.OnMouseLeave 21468
StdCtrls.TEdit.OnMouseMove 21468
StdCtrls.TEdit.OnMouseUp 21470
StdCtrls.TEdit.OnStartDock 21473
StdCtrls.TEdit.OnStartDrag 21473
StdCtrls.TEdit.ParentBiDiMode 21432
StdCtrls.TEdit.ParentColor 21432
StdCtrls.TEdit.ParentCtl3D 21433
StdCtrls.TEdit.ParentFont 21433
StdCtrls.TEdit.ParentShowHint 21434
StdCtrls.TEdit.PasswordChar 21434
StdCtrls.TEdit.PopupMenu 21435
StdCtrls.TEdit.ReadOnly 21439
StdCtrls.TEdit.ShowHint 21440
StdCtrls.TEdit.TabOrder 21441
StdCtrls.TEdit.TabStop 21442
StdCtrls.TEdit.Text 21442
StdCtrls.TEdit.Visible 21443
StdCtrls.TEditCharCase 21946
StdCtrls.TEditCharCase enumeration 21946
StdCtrls.TEllipsisPosition 21947
StdCtrls.TEllipsisPosition enumeration 21947
StdCtrls.TGroupBox 21474
StdCtrls.TGroupBox.Align 21485
StdCtrls.TGroupBox.Anchors 21489
StdCtrls.TGroupBox.BiDiMode 21490
StdCtrls.TGroupBox.Caption 21490
StdCtrls.TGroupBox.Color 21493
StdCtrls.TGroupBox.Constraints 21493
StdCtrls.TGroupBox.Ctl3D 21494
StdCtrls.TGroupBox.DockSite 21495
StdCtrls.TGroupBox.DragCursor 21496
StdCtrls.TGroupBox.DragKind 21496
StdCtrls.TGroupBox.DragMode 21496
StdCtrls.TGroupBox.Enabled 21498
StdCtrls.TGroupBox.Font 21498
StdCtrls.TGroupBox.OnAlignInsertBefore 21511
StdCtrls.TGroupBox.OnAlignPosition 21512

StdCtrls.TGroupBox.OnClick 21513
StdCtrls.TGroupBox.OnContextPopup 21514
StdCtrls.TGroupBox.OnDblClick 21515
StdCtrls.TGroupBox.OnDockDrop 21515
StdCtrls.TGroupBox.OnDockOver 21517
StdCtrls.TGroupBox.OnDragDrop 21520
StdCtrls.TGroupBox.OnDragOver 21522
StdCtrls.TGroupBox.OnEndDock 21524
StdCtrls.TGroupBox.OnEndDrag 21525
StdCtrls.TGroupBox.OnEnter 21526
StdCtrls.TGroupBox.OnExit 21528
StdCtrls.TGroupBox.OnGetSiteInfo 21530
StdCtrls.TGroupBox.OnMouseActivate 21532
StdCtrls.TGroupBox.OnMouseDown 21533
StdCtrls.TGroupBox.OnMouseEnter 21535
StdCtrls.TGroupBox.OnMouseLeave 21535
StdCtrls.TGroupBox.OnMouseMove 21535
StdCtrls.TGroupBox.OnMouseUp 21538
StdCtrls.TGroupBox.OnStartDock 21540
StdCtrls.TGroupBox.OnStartDrag 21541
StdCtrls.TGroupBox.OnUnDock 21541
StdCtrls.TGroupBox.Padding 21499
StdCtrls.TGroupBox.ParentBackground 21500
StdCtrls.TGroupBox.ParentBiDiMode 21500
StdCtrls.TGroupBox.ParentColor 21500
StdCtrls.TGroupBox.ParentCtl3D 21501
StdCtrls.TGroupBox.ParentFont 21501
StdCtrls.TGroupBox.ParentShowHint 21502
StdCtrls.TGroupBox.PopupMenu 21502
StdCtrls.TGroupBox.ShowHint 21507
StdCtrls.TGroupBox.TabOrder 21508
StdCtrls.TGroupBox.TabStop 21509
StdCtrls.TGroupBox.Visible 21509
StdCtrls.TLabel 21544
StdCtrls.TLabel.Align 21553
StdCtrls.TLabel.Alignment 21556
StdCtrls.TLabel.Anchors 21557
StdCtrls.TLabel.AutoSize 21557
StdCtrls.TLabel.BiDiMode 21558
StdCtrls.TLabel.Caption 21559
StdCtrls.TLabel.Color 21561
StdCtrls.TLabel.Constraints 21562
StdCtrls.TLabel.DragCursor 21563
StdCtrls.TLabel.DragKind 21563
StdCtrls.TLabel.DragMode 21563
StdCtrls.TLabel.EllipsisPosition 21565
StdCtrls.TLabel.Enabled 21565
StdCtrls.TLabel.FocusControl 21566
StdCtrls.TLabel.Font 21566
StdCtrls.TLabel.Layout 21567
StdCtrls.TLabel.OnClick 21579
StdCtrls.TLabel.OnContextPopup 21581
StdCtrls.TLabel.OnDblClick 21582
StdCtrls.TLabel.OnDragDrop 21582
StdCtrls.TLabel.OnDragOver 21584
StdCtrls.TLabel.OnEndDock 21587
StdCtrls.TLabel.OnEndDrag 21587
StdCtrls.TLabel.OnMouseActivate 21588
StdCtrls.TLabel.OnMouseDown 21589
StdCtrls.TLabel.OnMouseEnter 21592
StdCtrls.TLabel.OnMouseLeave 21592
StdCtrls.TLabel.OnMouseMove 21592
StdCtrls.TLabel.OnMouseUp 21594
StdCtrls.TLabel.OnStartDock 21597
StdCtrls.TLabel.OnStartDrag 21597
StdCtrls.TLabel.ParentBiDiMode 21567
StdCtrls.TLabel.ParentColor 21568
StdCtrls.TLabel.ParentFont 21568
StdCtrls.TLabel.ParentShowHint 21569
StdCtrls.TLabel.PopupMenu 21569
StdCtrls.TLabel.ShowAccelChar 21574
StdCtrls.TLabel.ShowHint 21575
StdCtrls.TLabel.Transparent 21576
StdCtrls.TLabel.Visible 21577
StdCtrls.TLabel.WordWrap 21578
StdCtrls.TLBFindDataEvent 21954
StdCtrls.TLBFindDataEvent type 21954
StdCtrls.TLBGetDataEvent 21955

StdCtrls.TLBGetDataEvent type 21955
StdCtrls.TLBGetDataObjectEvent 21955
StdCtrls.TLBGetDataObjectEvent type 21955
StdCtrls.TListBox 21598
StdCtrls.TListBox.Align 21611
StdCtrls.TListBox.Anchors 21615
StdCtrls.TListBox.AutoComplete 21615
StdCtrls.TListBox.AutoCompleteDelay 21615
StdCtrls.TListBox.BevelEdges 21616
StdCtrls.TListBox.BevelInner 21616
StdCtrls.TListBox.BevelKind 21617
StdCtrls.TListBox.BevelOuter 21617
StdCtrls.TListBox.BevelWidth 21617
StdCtrls.TListBox.BiDiMode 21618
StdCtrls.TListBox.BorderStyle 21618
StdCtrls.TListBox.Color 21619
StdCtrls.TListBox.Columns 21620
StdCtrls.TListBox.Constraints 21620
StdCtrls.TListBox.Ctl3D 21621
StdCtrls.TListBox.DragCursor 21621
StdCtrls.TListBox.DragKind 21621
StdCtrls.TListBox.DragMode 21622
StdCtrls.TListBox.Enabled 21623
StdCtrls.TListBox.ExtendedSelect 21624
StdCtrls.TListBox.Font 21624
StdCtrls.TListBox.ImeMode 21625
StdCtrls.TListBox.ImeName 21626
StdCtrls.TListBox.IntegralHeight 21627
StdCtrls.TListBox.ItemHeight 21627
StdCtrls.TListBox.Items 21628
StdCtrls.TListBox.MultiSelect 21630
StdCtrls.TListBox.OnClick 21645
StdCtrls.TListBox.OnContextPopup 21647
StdCtrls.TListBox.OnData 21647
StdCtrls.TListBox.OnDataFind 21648
StdCtrls.TListBox.OnDataObject 21648
StdCtrls.TListBox.OnDblClick 21649
StdCtrls.TListBox.OnDragDrop 21649
StdCtrls.TListBox.OnDragOver 21651
StdCtrls.TListBox.OnDrawItem 21654
StdCtrls.TListBox.OnEndDock 21656
StdCtrls.TListBox.OnEndDrag 21656
StdCtrls.TListBox.OnEnter 21657
StdCtrls.TListBox.OnExit 21660
StdCtrls.TListBox.OnKeyDown 21661
StdCtrls.TListBox.OnKeyPress 21664
StdCtrls.TListBox.OnKeyUp 21665
StdCtrls.TListBox.OnMeasureItem 21667
StdCtrls.TListBox.OnMouseActivate 21668
StdCtrls.TListBox.OnMouseDown 21669
StdCtrls.TListBox.OnMouseEnter 21671
StdCtrls.TListBox.OnMouseLeave 21672
StdCtrls.TListBox.OnMouseMove 21672
StdCtrls.TListBox.OnMouseUp 21674
StdCtrls.TListBox.OnStartDock 21676
StdCtrls.TListBox.OnStartDrag 21677
StdCtrls.TListBox.ParentBiDiMode 21631
StdCtrls.TListBox.ParentColor 21631
StdCtrls.TListBox.ParentCtl3D 21631
StdCtrls.TListBox.ParentFont 21632
StdCtrls.TListBox.ParentShowHint 21632
StdCtrls.TListBox.PopupMenu 21633
StdCtrls.TListBox.ScrollWidth 21637
StdCtrls.TListBox.ShowHint 21638
StdCtrls.TListBox.Sorted 21639
StdCtrls.TListBox.Style 21641
StdCtrls.TListBox.TabOrder 21641
StdCtrls.TListBox.TabStop 21642
StdCtrls.TListBox.TabWidth 21642
StdCtrls.TListBox.Visible 21643
StdCtrls.TListBoxStyle 21948
StdCtrls.TListBoxStyle enumeration 21948
StdCtrls.TMeasureItemEvent 21955
StdCtrls.TMeasureItemEvent type 21955
StdCtrls.TMemo 21678
StdCtrls.TMemo.Align 21691
StdCtrls.TMemo.Alignment 21695
StdCtrls.TMemo.Anchors 21695

StdCtrls.TMemo.BevelEdges 21696
StdCtrls.TMemo.BevelInner 21696
StdCtrls.TMemo.BevelKind 21696
StdCtrls.TMemo.BevelOuter 21697
StdCtrls.TMemo.BiDiMode 21697
StdCtrls.TMemo.BorderStyle 21698
StdCtrls.TMemo.Color 21698
StdCtrls.TMemo.Constraints 21699
StdCtrls.TMemo.Ctl3D 21699
StdCtrls.TMemo.DragCursor 21700
StdCtrls.TMemo.DragKind 21700
StdCtrls.TMemo.DragMode 21701
StdCtrls.TMemo.Enabled 21702
StdCtrls.TMemo.Font 21703
StdCtrls.TMemo.HideSelection 21704
StdCtrls.TMemo.ImeMode 21704
StdCtrls.TMemo.ImeName 21705
StdCtrls.TMemo.Lines 21705
StdCtrls.TMemo.MaxLength 21706
StdCtrls.TMemo.OEMConvert 21707
StdCtrls.TMemo.OnChange 21724
StdCtrls.TMemo.OnClick 21725
StdCtrls.TMemo.OnContextPopup 21727
StdCtrls.TMemo.OnDblClick 21727
StdCtrls.TMemo.OnDragDrop 21728
StdCtrls.TMemo.OnDragOver 21730
StdCtrls.TMemo.OnEndDock 21732
StdCtrls.TMemo.OnEndDrag 21733
StdCtrls.TMemo.OnEnter 21734
StdCtrls.TMemo.OnExit 21736
StdCtrls.TMemo.OnKeyDown 21738
StdCtrls.TMemo.OnKeyPress 21741
StdCtrls.TMemo.OnKeyUp 21742
StdCtrls.TMemo.OnMouseActivate 21743
StdCtrls.TMemo.OnMouseDown 21744
StdCtrls.TMemo.OnMouseEnter 21747
StdCtrls.TMemo.OnMouseLeave 21747
StdCtrls.TMemo.OnMouseMove 21747
StdCtrls.TMemo.OnMouseUp 21749
StdCtrls.TMemo.OnStartDock 21752
StdCtrls.TMemo.OnStartDrag 21752
StdCtrls.TMemo.ParentBiDiMode 21707
StdCtrls.TMemo.ParentColor 21708
StdCtrls.TMemo.ParentCtl3D 21708
StdCtrls.TMemo.ParentFont 21709
StdCtrls.TMemo.ParentShowHint 21709
StdCtrls.TMemo.PopupMenu 21710
StdCtrls.TMemo.ReadOnly 21714
StdCtrls.TMemo.ScrollBars 21715
StdCtrls.TMemo.ShowHint 21717
StdCtrls.TMemo.TabOrder 21718
StdCtrls.TMemo.TabStop 21719
StdCtrls.TMemo.Visible 21719
StdCtrls.TMemo.WantReturns 21721
StdCtrls.TMemo.WantTabs 21721
StdCtrls.TMemo.WordWrap 21722
StdCtrls.TOwnerDrawState 21956
StdCtrls.TOwnerDrawState type 21956
StdCtrls.TRadioButton 21753
StdCtrls.TRadioButton.Action 21767
StdCtrls.TRadioButton.Align 21768
StdCtrls.TRadioButton.Alignment 21771
StdCtrls.TRadioButton.Anchors 21772
StdCtrls.TRadioButton.BiDiMode 21772
StdCtrls.TRadioButton.Caption 21773
StdCtrls.TRadioButton.Checked 21775
StdCtrls.TRadioButton.Color 21776
StdCtrls.TRadioButton.Constraints 21776
StdCtrls.TRadioButton.Create 21764
StdCtrls.TRadioButton.CreateParams 21765
StdCtrls.TRadioButton.CreateWnd 21765
StdCtrls.TRadioButton.Ctl3D 21777
StdCtrls.TRadioButton.DragCursor 21777
StdCtrls.TRadioButton.DragKind 21778
StdCtrls.TRadioButton.DragMode 21778
StdCtrls.TRadioButton.Enabled 21780
StdCtrls.TRadioButton.Font 21780
StdCtrls.TRadioButton.GetChecked 21766

StdCtrls.TRadioButton.GetControlsAlignment 21766
StdCtrls.TRadioButton.OnClick 21793
StdCtrls.TRadioButton.OnContextPopup 21795
StdCtrls.TRadioButton.OnDblClick 21796
StdCtrls.TRadioButton.OnDragDrop 21796
StdCtrls.TRadioButton.OnDragOver 21798
StdCtrls.TRadioButton.OnEndDock 21801
StdCtrls.TRadioButton.OnEndDrag 21801
StdCtrls.TRadioButton.OnEnter 21802
StdCtrls.TRadioButton.OnExit 21805
StdCtrls.TRadioButton.OnKeyDown 21806
StdCtrls.TRadioButton.OnKeyPress 21809
StdCtrls.TRadioButton.OnKeyUp 21810
StdCtrls.TRadioButton.OnMouseActivate 21812
StdCtrls.TRadioButton.OnMouseDown 21813
StdCtrls.TRadioButton.OnMouseEnter 21815
StdCtrls.TRadioButton.OnMouseLeave 21815
StdCtrls.TRadioButton.OnMouseMove 21815
StdCtrls.TRadioButton.OnMouseUp 21818
StdCtrls.TRadioButton.OnStartDock 21820
StdCtrls.TRadioButton.OnStartDrag 21821
StdCtrls.TRadioButton.ParentBiDiMode 21781
StdCtrls.TRadioButton.ParentColor 21782
StdCtrls.TRadioButton.ParentCtl3D 21782
StdCtrls.TRadioButton.ParentFont 21783
StdCtrls.TRadioButton.ParentShowHint 21783
StdCtrls.TRadioButton.PopupMenu 21784
StdCtrls.TRadioButton.SetChecked 21766
StdCtrls.TRadioButton.ShowHint 21788
StdCtrls.TRadioButton.TabOrder 21790
StdCtrls.TRadioButton.TabStop 21790
StdCtrls.TRadioButton.Visible 21791
StdCtrls.TRadioButton.WordWrap 21792
StdCtrls.TScrollBar 21821
StdCtrls.TScrollBar.Align 21835
StdCtrls.TScrollBar.Anchors 21839
StdCtrls.TScrollBar.BiDiMode 21840
StdCtrls.TScrollBar.Change 21833
StdCtrls.TScrollBar.Constraints 21840
StdCtrls.TScrollBar.Create 21832
StdCtrls.TScrollBar.CreateParams 21833
StdCtrls.TScrollBar.CreateWnd 21834
StdCtrls.TScrollBar.Ctl3D 21841
StdCtrls.TScrollBar.DragCursor 21841
StdCtrls.TScrollBar.DragKind 21841
StdCtrls.TScrollBar.DragMode 21842
StdCtrls.TScrollBar.Enabled 21843
StdCtrls.TScrollBar.Kind 21844
StdCtrls.TScrollBar.LargeChange 21845
StdCtrls.TScrollBar.Max 21847
StdCtrls.TScrollBar.Min 21849
StdCtrls.TScrollBar.OnChange 21866
StdCtrls.TScrollBar.OnContextPopup 21867
StdCtrls.TScrollBar.OnDragDrop 21868
StdCtrls.TScrollBar.OnDragOver 21870
StdCtrls.TScrollBar.OnEndDock 21872
StdCtrls.TScrollBar.OnEndDrag 21873
StdCtrls.TScrollBar.OnEnter 21874
StdCtrls.TScrollBar.OnExit 21876
StdCtrls.TScrollBar.OnKeyDown 21878
StdCtrls.TScrollBar.OnKeyPress 21880
StdCtrls.TScrollBar.OnKeyUp 21882
StdCtrls.TScrollBar.OnMouseEnter 21883
StdCtrls.TScrollBar.OnMouseLeave 21884
StdCtrls.TScrollBar.OnScroll 21884
StdCtrls.TScrollBar.OnStartDock 21886
StdCtrls.TScrollBar.OnStartDrag 21886
StdCtrls.TScrollBar.PageSize 21851
StdCtrls.TScrollBar.ParentBiDiMode 21852
StdCtrls.TScrollBar.ParentCtl3D 21853
StdCtrls.TScrollBar.ParentShowHint 21853
StdCtrls.TScrollBar.PopupMenu 21853
StdCtrls.TScrollBar.Position 21858
StdCtrls.TScrollBar.Scroll 21834
StdCtrls.TScrollBar.SetParams 21834
StdCtrls.TScrollBar.ShowHint 21860
StdCtrls.TScrollBar.SmallChange 21861
StdCtrls.TScrollBar.TabOrder 21863

StdCtrls.TScrollBar.TabStop 21864
StdCtrls.TScrollBar.Visible 21864
StdCtrls.TScrollCode 21949
StdCtrls.TScrollCode enumeration 21949
StdCtrls.TScrollEvent 21957
StdCtrls.TScrollEvent type 21957
StdCtrls.TScrollStyle 21950
StdCtrls.TScrollStyle enumeration 21950
StdCtrls.TSelection 21950
StdCtrls.TSelection record 21950
StdCtrls.TStaticBorderStyle 21951
StdCtrls.TStaticBorderStyle enumeration 21951
StdCtrls.TStaticText 21887
StdCtrls.TStaticText.Align 21899
StdCtrls.TStaticText.Alignment 21902
StdCtrls.TStaticText.Anchors 21903
StdCtrls.TStaticText.AutoSize 21903
StdCtrls.TStaticText.BevelEdges 21904
StdCtrls.TStaticText.BevelInner 21904
StdCtrls.TStaticText.BevelKind 21904
StdCtrls.TStaticText.BevelOuter 21905
StdCtrls.TStaticText.BiDiMode 21905
StdCtrls.TStaticText.BorderStyle 21906
StdCtrls.TStaticText.Caption 21906
StdCtrls.TStaticText.Color 21909
StdCtrls.TStaticText.Constraints 21909
StdCtrls.TStaticText.DragCursor 21910
StdCtrls.TStaticText.DragKind 21910
StdCtrls.TStaticText.DragMode 21911
StdCtrls.TStaticText.Enabled 21912
StdCtrls.TStaticText.FocusControl 21913
StdCtrls.TStaticText.Font 21913
StdCtrls.TStaticText.OnClick 21926
StdCtrls.TStaticText.OnContextPopup 21928
StdCtrls.TStaticText.OnDblClick 21928
StdCtrls.TStaticText.OnDragDrop 21929
StdCtrls.TStaticText.OnDragOver 21931
StdCtrls.TStaticText.OnEndDock 21933
StdCtrls.TStaticText.OnEndDrag 21934
StdCtrls.TStaticText.OnMouseActivate 21935
StdCtrls.TStaticText.OnMouseDown 21936
StdCtrls.TStaticText.OnMouseEnter 21938
StdCtrls.TStaticText.OnMouseLeave 21938
StdCtrls.TStaticText.OnMouseMove 21939
StdCtrls.TStaticText.OnMouseUp 21941
StdCtrls.TStaticText.OnStartDock 21943
StdCtrls.TStaticText.OnStartDrag 21944
StdCtrls.TStaticText.ParentBiDiMode 21914
StdCtrls.TStaticText.ParentColor 21915
StdCtrls.TStaticText.ParentFont 21915
StdCtrls.TStaticText.ParentShowHint 21916
StdCtrls.TStaticText.PopupMenu 21916
StdCtrls.TStaticText.ShowAccelChar 21921
StdCtrls.TStaticText.ShowHint 21921
StdCtrls.TStaticText.TabOrder 21923
StdCtrls.TStaticText.TabStop 21923
StdCtrls.TStaticText.Transparent 21924
StdCtrls.TStaticText.Visible 21924
StdCtrls.TTextLayout 21951
StdCtrls.TTextLayout enumeration 21951
stDevice enumeration member 22048
stDgram enumeration member 20138
stDisabled enumeration member 22049
stEllipse enumeration member 13270
stExecuting enumeration member 1302
stFetching enumeration member 1302
stFileSystem enumeration member 22048
stGreen enumeration member 14756
stHue enumeration member 14756
stIndexes enumeration member 20366
stLuminance enumeration member 14756
stManual enumeration member 22049
stMatchCase enumeration member 5464
stNonBlocking enumeration member 18887
stNone enumeration member 5465, 15792
stNormal enumeration member 17549
stNoSchema enumeration member 20366
stOpen enumeration member 1302

stPackages enumeration member 20366
stProcedureParams enumeration member 20366
stProcedures enumeration member 20366
stRaw enumeration member 20138
stRdm enumeration member 20138
stRectangle enumeration member 13270
stRed enumeration member 14756
stRoundRect enumeration member 13270
stRoundSquare enumeration member 13270
StrUtils 21959
StrUtils namespace 21959
Constants 21997
Functions 21962
Structs, Records, Enums 21992
Types 21993
Variables 21995
StrUtils.AnsiContainsStr 21963
StrUtils.AnsiContainsStr function 21963
StrUtils.AnsiContainsText 21964
StrUtils.AnsiContainsText function 21964
StrUtils.AnsiEndsStr 21964
StrUtils.AnsiEndsStr function 21964
StrUtils.AnsiEndsText 21965
StrUtils.AnsiEndsText function 21965
StrUtils.AnsiIndexStr 21965
StrUtils.AnsiIndexStr function 21965
StrUtils.AnsiIndexText 21966
StrUtils.AnsiIndexText function 21966
StrUtils.AnsiLeftStr 21966
StrUtils.AnsiLeftStr function 21966
StrUtils.AnsiMatchStr 21967
StrUtils.AnsiMatchStr function 21967
StrUtils.AnsiMatchText 21967
StrUtils.AnsiMatchText function 21967
StrUtils.AnsiMidStr 21968
StrUtils.AnsiMidStr function 21968
StrUtils.AnsiReplaceStr 21969
StrUtils.AnsiReplaceStr function 21969
StrUtils.AnsiReplaceText 21969
StrUtils.AnsiReplaceText function 21969
StrUtils.AnsiResemblesProc 21995
StrUtils.AnsiResemblesProc variable 21995
StrUtils.AnsiResemblesText 21970
StrUtils.AnsiResemblesText function 21970
StrUtils.AnsiReverseString 21970
StrUtils.AnsiReverseString function 21970
StrUtils.AnsiRightStr 21971
StrUtils.AnsiRightStr function 21971
StrUtils.AnsiStartsStr 21971
StrUtils.AnsiStartsStr function 21971
StrUtils.AnsiStartsText 21972
StrUtils.AnsiStartsText function 21972
StrUtils.ContainsStr 21972
StrUtils.ContainsStr function 21972
StrUtils.ContainsText 21973
StrUtils.ContainsText function 21973
StrUtils.DecodeSoundexInt 21973
StrUtils.DecodeSoundexInt function 21973
StrUtils.DecodeSoundexWord 21974
StrUtils.DecodeSoundexWord function 21974
StrUtils.DupeString 21974
StrUtils.DupeString function 21974
StrUtils.EndsStr 21975
StrUtils.EndsStr function 21975
StrUtils.EndsText 21975
StrUtils.EndsText function 21975
StrUtils.IfThen 21976
StrUtils.IfThen function 21976
StrUtils.IndexStr 21976
StrUtils.IndexStr function 21976
StrUtils.IndexText 21977
StrUtils.IndexText function 21977
StrUtils.LeftBStr 21977
StrUtils.LeftBStr function 21977
StrUtils.LeftStr 21978
StrUtils.LeftStr function 21978
StrUtils.MatchStr 21979
StrUtils.MatchStr function 21979

StrUtils.MatchText 21979
StrUtils.MatchText function 21979
StrUtils.MidBStr 21980
StrUtils.MidBStr function 21980
StrUtils.MidStr 21980
StrUtils.MidStr function 21980
StrUtils.PosEx 21981
StrUtils.PosEx function 21981
StrUtils.RandomFrom 21981
StrUtils.RandomFrom function 21981
StrUtils.ReplaceStr 21982
StrUtils.ReplaceStr function 21982
StrUtils.ReplaceText 21982
StrUtils.ReplaceText function 21982
StrUtils.ResemblesProc 21995
StrUtils.ResemblesProc variable 21995
StrUtils.ResemblesText 21983
StrUtils.ResemblesText function 21983
StrUtils.ReverseString 21983
StrUtils.ReverseString function 21983
StrUtils.RightBStr 21984
StrUtils.RightBStr function 21984
StrUtils.RightStr 21984
StrUtils.RightStr function 21984
StrUtils.SearchBuf 21985
StrUtils.SearchBuf function 21985
StrUtils.Soundex 21986
StrUtils.Soundex function 21986
StrUtils.SoundexCompare 21986
StrUtils.SoundexCompare function 21986
StrUtils.SoundexInt 21987
StrUtils.SoundexInt function 21987
StrUtils.SoundexProc 21987
StrUtils.SoundexProc function 21987
StrUtils.SoundexSimilar 21988
StrUtils.SoundexSimilar function 21988
StrUtils.SoundexWord 21989
StrUtils.SoundexWord function 21989
StrUtils.StartsStr 21989
StrUtils.StartsStr function 21989
StrUtils.StartsText 21990
StrUtils.StartsText function 21990
StrUtils.StuffString 21990
StrUtils.StuffString function 21990
StrUtils.TCompareTextProc 21993
StrUtils.TCompareTextProc type 21993
StrUtils.TSoundexIntLength 21993
StrUtils.TSoundexIntLength type 21993
StrUtils.TSoundexLength 21994
StrUtils.TSoundexLength type 21994
StrUtils.TStringSeachOption 21992
StrUtils.TStringSeachOption enumeration 21992
StrUtils.TStringSearchOptions 21994
StrUtils.TStringSearchOptions type 21994
StrUtils.WordDelimiters 21997
StrUtils.WordDelimiters constant 21997
stSaturation enumeration member 14756
stSeqPacket enumeration member 20138
stSquare enumeration member 13270
stStream enumeration member 20138
stSysTables enumeration member 20366
stSystem enumeration member 22049
stTables enumeration member 20366
stTemplate enumeration member 17549
stText enumeration member 5465, 15792
stThreadBlocking enumeration member 18887
stUserNames enumeration member 20366
stWholeWord enumeration member 5464
stWin32 enumeration member 22048
svApplication enumeration member 25544
SvcMgr 21999
SvcMgr namespace 21999
- Classes 22001
- Constants 22055
- Structs, Records, Enums 22047
- Types 22051
- Variables 22054

SvcMgr.Application 22054

SvcMgr.Application variable 22054
SvcMgr.CM_SERVICE_CONTROL_CODE 22055
SvcMgr.CM_SERVICE_CONTROL_CODE constant 22055
SvcMgr.TContinueEvent 22051
SvcMgr.TContinueEvent type 22051
SvcMgr.TCurrentStatus 22047
SvcMgr.TCurrentStatus enumeration 22047
SvcMgr.TDependencies 22001
SvcMgr.TDependencies.Create 22003
SvcMgr.TDependencies.GetOwner 22004
SvcMgr.TDependencies.Items 22004
SvcMgr.TDependency 22005
SvcMgr.TDependency.GetDisplayName 22007
SvcMgr.TDependency.IsGroup 22008
SvcMgr.TDependency.Name 22008
SvcMgr.TErrorSeverity 22048
SvcMgr.TErrorSeverity enumeration 22048
SvcMgr.TEventLogger 22009
SvcMgr.TEventLogger.Create 22010
SvcMgr.TEventLogger.Destroy 22010
SvcMgr.TEventLogger.LogMessage 22011
SvcMgr.TPauseEvent 22051
SvcMgr.TPauseEvent type 22051
SvcMgr.TService 22011
SvcMgr.TService.AfterInstall 22028
SvcMgr.TService.AfterUninstall 22029
SvcMgr.TService.AllowPause 22021
SvcMgr.TService.AllowStop 22021
SvcMgr.TService.BeforeInstall 22029
SvcMgr.TService.BeforeUninstall 22030
SvcMgr.TService.Controller 22017
SvcMgr.TService.CreateNew 22015
SvcMgr.TService.Dependencies 22021
SvcMgr.TService.Destroy 22016
SvcMgr.TService.DisplayName 22022
SvcMgr.TService.DoContinue 22017
SvcMgr.TService.DoCustomControl 22017
SvcMgr.TService.DoInterrogate 22017
SvcMgr.TService.DoPause 22018
SvcMgr.TService.DoShutdown 22018
SvcMgr.TService.DoStart 22018
SvcMgr.TService.DoStop 22018
SvcMgr.TService.ErrCode 22022
SvcMgr.TService.ErrorSeverity 22022
SvcMgr.TService.GetServiceController 22018
SvcMgr.TService.Interactive 22023
SvcMgr.TService.LoadGroup 22023
SvcMgr.TService.LogMessage 22019
SvcMgr.TService.Main 22019
SvcMgr.TService.OnContinue 22030
SvcMgr.TService.OnExecute 22030
SvcMgr.TService.OnPause 22031
SvcMgr.TService.OnShutdown 22031
SvcMgr.TService.OnStart 22031
SvcMgr.TService.OnStop 22032
SvcMgr.TService.Param 22024
SvcMgr.TService.ParamCount 22024
SvcMgr.TService.Password 22024
SvcMgr.TService.ReportStatus 22020
SvcMgr.TService.ServiceStartName 22025
SvcMgr.TService.ServiceThread 22025
SvcMgr.TService.ServiceType 22026
SvcMgr.TService.StartType 22026
SvcMgr.TService.Status 22026
SvcMgr.TService.TagID 22026
SvcMgr.TService.Terminated 22027
SvcMgr.TService.WaitHint 22027
SvcMgr.TService.Win32ErrCode 22028
SvcMgr.TServiceApplication 22032
SvcMgr.TServiceApplication.Create 22036
SvcMgr.TServiceApplication.CreateForm 22037
SvcMgr.TServiceApplication.DelayInitialize 22039
SvcMgr.TServiceApplication.Destroy 22036
SvcMgr.TServiceApplication.DispatchServiceMain 22037
SvcMgr.TServiceApplication.DoHandleException 22037
SvcMgr.TServiceApplication.Hook 22038
SvcMgr.TServiceApplication.Initialize 22038
SvcMgr.TServiceApplication.Installing 22038

SvcMgr.TServiceApplication.RegisterServices 22038
SvcMgr.TServiceApplication.Run 22039
SvcMgr.TServiceApplication.ServiceCount 22039
SvcMgr.TServiceApplication.Title 22040
SvcMgr.TServiceController 22052
SvcMgr.TServiceController type 22052
SvcMgr.TServiceEvent 22052
SvcMgr.TServiceEvent type 22052
SvcMgr.TServiceThread 22040
SvcMgr.TServiceThread.Create 22042
SvcMgr.TServiceThread.Execute 22042
SvcMgr.TServiceThread.ProcessRequests 22046
SvcMgr.TServiceType 22048
SvcMgr.TServiceType enumeration 22048
SvcMgr.TStartEvent 22052
SvcMgr.TStartEvent type 22052
SvcMgr.TStartType 22049
SvcMgr.TStartType enumeration 22049
SvcMgr.TStopEvent 22053
SvcMgr.TStopEvent type 22053
svEndUser enumeration member 25544
svHTMLItems enumeration member 25544
svModules enumeration member 25544
svPage enumeration member 25544
svPages enumeration member 25544
svProducer enumeration member 25544
svRequest enumeration member 25544
svResponse enumeration member 25544
svSession enumeration member 25544
SyncObjs 22057
SyncObjs namespace 22057
Classes 22058
Structs, Records, Enums 22078
SyncObjs.TCriticalSection 22058
SyncObjs.TCriticalSection.Acquire 22061
SyncObjs.TCriticalSection.Create 22059
SyncObjs.TCriticalSection.Destroy 22060
SyncObjs.TCriticalSection.Enter 22061
SyncObjs.TCriticalSection.FSection 22060
SyncObjs.TCriticalSection.Leave 22062
SyncObjs.TCriticalSection.Release 22062
SyncObjs.TCriticalSection.TryEnter 22062
SyncObjs.TEvent 22063
SyncObjs.TEvent.Create 22065
SyncObjs.TEvent.ResetEvent 22066
SyncObjs.TEvent.SetEvent 22066
SyncObjs.THandleObject 22066
SyncObjs.THandleObject.Create 22068
SyncObjs.THandleObject.Destroy 22068
SyncObjs.THandleObject.FHandle 22069
SyncObjs.THandleObject.FLastError 22069
SyncObjs.THandleObject.FUseCOMWait 22069
SyncObjs.THandleObject.Handle 22070
SyncObjs.THandleObject.LastError 22070
SyncObjs.THandleObject.WaitFor 22070
SyncObjs.TMutex 22071
SyncObjs.TMutex.Acquire 22073
SyncObjs.TMutex.Create 22072, 22073
SyncObjs.TMutex.Release 22074
SyncObjs.TSimpleEvent 22074
SyncObjs.TSynchroObject 22075
SyncObjs.TSynchroObject.Acquire 22076
SyncObjs.TSynchroObject.Release 22076
SyncObjs.TWaitResult 22078
SyncObjs.TWaitResult enumeration 22078
SynonymsColumns class 2188
about SynonymsColumns class 2188
CatalogName 2189
SchemaName 2189
SynonymName 2189
SynonymsColumns fields 2189
SynonymsColumns Members 2188
TableCatalogName 2189
TableName 2190
TableSchemaName 2190
SysMirrors class 2089
about SysMirrors class 2089
AutoFailOver 2092

BirthStamp 2092
ClientHashCode 2092
FailOverPriority 2092
FileName 2093
Host 2093
Id 2093
InstantSynch 2093
LastKnownReplay 2093
Name 2094
Port 2094
SynchMirrorName 2094
SysMirrors fields 2091
SysMirrors Members 2091
Type 2094
ValidatedPrimary 2094
SysSchedule class 2095
about SysSchedule class 2095
Day 2096
Id 2096
Millis 2096
Period 2096
Ref 2097
ScheduleItemClass 2097
SysSchedule fields 2096
SysSchedule Members 2095
Time 2097
System 22079
System namespace 22079
Classes 22092
Constants 22660
Functions 22400
Interfaces 22394
Structs, Records, Enums 22552
Types 22579
Variables 22629
System._CVR_PROBE 22405
System._CVR_PROBE function 22405
System._CVR_STMTPROBE 22406
System._CVR_STMTPROBE function 22406
System.Abs 22410
System.Abs function 22410
System.AbstractErrorProc 22630
System.AbstractErrorProc variable 22630
System.AcquireExceptionObject 22410
System.AcquireExceptionObject function 22410
System.AddModuleUnloadProc 22411
System.AddModuleUnloadProc function 22411
System.Addr 22412
System.Addr function 22412
System.AllocMem 22413
System.AllocMem function 22413
System.AllocMemCount 22630
System.AllocMemCount variable 22630
System.AllocMemSize 22631
System.AllocMemSize variable 22631
System.AnsiToUtf8 22415
System.AnsiToUtf8 function 22415
System.Append 22415
System.Append function 22415
System.ArcTan 22416
System.ArcTan function 22416
System.Assert 22417
System.Assert function 22417
System.AssertErrorProc 22633
System.AssertErrorProc variable 22633
System.Assigned 22419
System.Assigned function 22419
System.AssignFile 22420
System.AssignFile function 22420
System.AttemptToUseSharedMemoryManager 22422
System.AttemptToUseSharedMemoryManager function 22422
System.BeginThread 22422
System.BeginThread function 22422
System.BlockRead 22423
System.BlockRead function 22423
System.BlockWrite 22425
System.BlockWrite function 22425
System.Break 22426

System.Break function 22426
System.ChDir 22429
System.ChDir function 22429
System.Chr 22430
System.Chr function 22430
System.Close 22432
System.Close function 22432
System.CloseFile 22433
System.CloseFile function 22433
System.CmdLine 22635
System.CmdLine variable 22635
System.CmdShow 22635
System.CmdShow variable 22635
System.CompilerVersion 22662
System.CompilerVersion constant 22662
System.CompToCurrency 22435
System.CompToCurrency function 22435
System.CompToDouble 22435
System.CompToDouble function 22435
System.Concat 22436
System.Concat function 22436
System.Continue 22437
System.Continue function 22437
System.Copy 22438
System.Copy function 22438
System.Cos 22440
System.Cos function 22440
System.CPUi386 22662
System.CPUi386 constant 22662
System.CPUi486 22663
System.CPUi486 constant 22663
System.CPUPentium 22663
System.CPUPentium constant 22663
System.CurrencyToComp 22441
System.CurrencyToComp function 22441
System.DebugHook 22636
System.DebugHook variable 22636
System.Dec 22441
System.Dec function 22441
System.Default8087CW 22636
System.Default8087CW variable 22636
System.DefaultTextLineBreakStyle 22636
System.DefaultTextLineBreakStyle variable 22636
System.Delete 22443
System.Delete function 22443
System.DispCallByIDProc 22637
System.DispCallByIDProc variable 22637
System.Dispose 22444
System.Dispose function 22444
System.DLLProc 22637
System.DLLProc variable 22637
System.DoubleToComp 22446
System.DoubleToComp function 22446
System.DynArrayClear 22446
System.DynArrayClear function 22446
System.DynArrayDim 22446
System.DynArrayDim function 22446
System.DynArraySetLength 22447
System.DynArraySetLength function 22447
System.E_NOINTERFACE 22663
System.E_NOINTERFACE constant 22663
System.E_NOTIMPL 22664
System.E_NOTIMPL constant 22664
System.E_UNEXPECTED 22664
System.E_UNEXPECTED constant 22664
System.EndThread 22447
System.EndThread function 22447
System.EnumModules 22447
System.EnumModules function 22447
System.EnumResourceModules 22448
System.EnumResourceModules function 22448
System.envp 22638
System.envp variable 22638
System.Eof 22449
System.Eof function 22449
System.Eoln 22450
System.Eoln function 22450
System.Erase 22451

System.Erase function 22451
System.Error 22451
System.Error function 22451
System.ErrorAddr 22638
System.ErrorAddr variable 22638
System.ErrorProc 22639
System.ErrorProc variable 22639
System.ErrOutput 22640
System.ErrOutput variable 22640
System.ExceptAddr 22452
System.ExceptAddr function 22452
System.ExceptClsProc 22640
System.ExceptClsProc variable 22640
System.ExceptionClass 22641
System.ExceptionClass variable 22641
System.ExceptObject 22452
System.ExceptObject function 22452
System.ExceptObjProc 22641
System.ExceptObjProc variable 22641
System.ExceptProc 22641
System.ExceptProc variable 22641
System.Exclude 22453
System.Exclude function 22453
System.Exit 22453
System.Exit function 22453
System.ExitCode 22642
System.ExitCode variable 22642
System.ExitProc 22643
System.ExitProc variable 22643
System.ExitProcessProc 22644
System.ExitProcessProc variable 22644
System.Exp 22455
System.Exp function 22455
System.FileMode 22644
System.FileMode variable 22644
System.FilePos 22456
System.FilePos function 22456
System.FileSize 22457
System.FileSize function 22457
System.FillChar 22459
System.FillChar function 22459
System.Finalize 22460
System.Finalize function 22460
System.FindClassHInstance 22461
System.FindClassHInstance function 22461
System.FindHInstance 22461
System.FindHInstance function 22461
System.FindResourceHInstance 22462
System.FindResourceHInstance function 22462
System.Flush 22462
System.Flush function 22462
System.fmClosed 22664
System.fmClosed constant 22664
System.fmInOut 22665
System.fmInOut constant 22665
System.fmInput 22665
System.fmInput constant 22665
System.fmOutput 22665
System.fmOutput constant 22665
System.FPower10 22463
System.FPower10 function 22463
System.Frac 22463
System.Frac function 22463
System.FreeMem 22464
System.FreeMem function 22464
System.FreeMemory 22466
System.FreeMemory function 22466
System.Get8087CW 22466
System.Get8087CW function 22466
System.GetDir 22467
System.GetDir function 22467
System.GetHeapStatus 22468
System.GetHeapStatus function 22468
System.GetLastError 22469
System.GetLastError function 22469
System.GetMem 22469
System.GetMem function 22469
System.GetMemory 22470

System.GetMemory function 22470
System.GetMemoryManager 22472
System.GetMemoryManager function 22472
System.GetMemoryManagerState 22472
System.GetMemoryManagerState function 22472
System.GetMemoryMap 22473
System.GetMemoryMap function 22473
System.GetMinimumBlockAlignment 22473
System.GetMinimumBlockAlignment function 22473
System.GetModuleFileName 22474
System.GetModuleFileName function 22474
System.GetPackageInfoTable 22584
System.GetPackageInfoTable type 22584
System.GetVariantManager 22474
System.GetVariantManager function 22474
System.Halt 22475
System.Halt function 22475
System.HeapAllocFlags 22645
System.HeapAllocFlags variable 22645
System.HGLOBAL 22584
System.HGLOBAL type 22584
System.Hi 22476
System.Hi function 22476
System.High 22476
System.High function 22476
System.HINST 22585
System.HINST type 22585
System.HMODULE 22585
System.HMODULE type 22585
System.HPrevInst 22645
System.HPrevInst variable 22645
System.HRESULT 22585
System.HRESULT type 22585
System.HRSRC 22586
System.HRSRC type 22586
System.IDispatch 22394
System.IDispatch.GetIDsOfNames 22395
System.IDispatch.GetTypeInfo 22395
System.IDispatch.GetTypeInfoCount 22395
System.IDispatch.Invoke 22395
System.IInterface 22396
System.IInterface._AddRef 22397
System.IInterface._Release 22397
System.IInterface.QueryInterface 22398
System.IInvokable 22398
System.Inc 22478
System.Inc function 22478
System.Include 22479
System.Include function 22479
System.Initialize 22480
System.Initialize function 22480
System.InitProc 22646
System.InitProc variable 22646
System.Input 22647
System.Input variable 22647
System.Insert 22480
System.Insert function 22480
System.Int 22481
System.Int function 22481
System.IntegerArray 22586
System.IntegerArray type 22586
System.IOResult 22482
System.IOResult function 22482
System.IsConsole 22647
System.IsConsole variable 22647
System.IsLibrary 22648
System.IsLibrary variable 22648
System.IsMemoryManagerSet 22483
System.IsMemoryManagerSet function 22483
System.IsMultiThread 22648
System.IsMultiThread variable 22648
System.IsVariantManagerSet 22483
System.IsVariantManagerSet function 22483
System.IUnknown 22586
System.IUnknown type 22586
System.JITEnable 22649
System.JITEnable variable 22649
System.Length 22484

System.Length function 22484
System.LibModuleList 22650
System.LibModuleList variable 22650
System.Ln 22485
System.Ln function 22485
System.Lo 22486
System.Lo function 22486
System.LoadResourceModule 22486
System.LoadResourceModule function 22486
System.LoadResString 22487
System.LoadResString function 22487
System.Low 22487
System.Low function 22487
System.MainInstance 22650
System.MainInstance variable 22650
System.MainThreadID 22651
System.MainThreadID variable 22651
System.MaxInt 22666
System.MaxInt constant 22666
System.MaxLongint 22666
System.MaxLongint constant 22666
System.MkDir 22488
System.MkDir function 22488
System.ModuleUnloadList 22651
System.ModuleUnloadList variable 22651
System.Move 22490
System.Move function 22490
System.NeverSleepOnMMThreadContention 22652
System.NeverSleepOnMMThreadContention variable 22652
System.New 22491
System.New function 22491
System.NoErrMsg 22652
System.NoErrMsg variable 22652
System.NumSmallBlockTypes 22666
System.NumSmallBlockTypes constant 22666
System.Odd 22493
System.Odd function 22493
System.OleStrToString 22493
System.OleStrToString function 22493
System.OleStrToStrVar 22494
System.OleStrToStrVar function 22494
System.opAdd 22667
System.opAdd constant 22667
System.opAnd 22667
System.opAnd constant 22667
System.opCmpEQ 22668
System.opCmpEQ constant 22668
System.opCmpGE 22668
System.opCmpGE constant 22668
System.opCmpGT 22669
System.opCmpGT constant 22669
System.opCmpLE 22669
System.opCmpLE constant 22669
System.opCmpLT 22670
System.opCmpLT constant 22670
System.opCmpNE 22670
System.opCmpNE constant 22670
System.opCompare 22671
System.opCompare constant 22671
System.opDivide 22671
System.opDivide constant 22671
System.opIntDivide 22672
System.opIntDivide constant 22672
System.opModulus 22672
System.opModulus constant 22672
System.opMultiply 22673
System.opMultiply constant 22673
System.opNegate 22673
System.opNegate constant 22673
System.opNot 22674
System.opNot constant 22674
System.opOr 22674
System.opOr constant 22674
System.opShiftLeft 22675
System.opShiftLeft constant 22675
System.opShiftRight 22675
System.opShiftRight constant 22675
System.opSubtract 22676

wwww

System.opSubtract constant 22676
System.opXor 22676
System.opXor constant 22676
System.Ord 22494
System.Ord function 22494
System.Output 22653
System.Output variable 22653
System.PackageInfo 22587
System.PackageInfo type 22587
System.PackageInfoTable 22553
System.PackageInfoTable record 22553
System.PackageUnitEntry 22553
System.PackageUnitEntry record 22553
System.PAnsiString 22587
System.PAnsiString type 22587
System.ParamCount 22495
System.ParamCount function 22495
System.ParamStr 22496
System.ParamStr function 22496
System.PBoolean 22587
System.PBoolean type 22587
System.PByte 22588
System.PByte type 22588
System.PCardinal 22589
System.PCardinal type 22589
System.PComp 22589
System.PComp type 22589
System.PCurrency 22589
System.PCurrency type 22589
System.PCVModInfo 22590
System.PCVModInfo type 22590
System.PDate 22590
System.PDate type 22590
System.PDateTime 22591
System.PDateTime type 22591
System.PDispatch 22591
System.PDispatch type 22591
System.PDouble 22592
System.PDouble type 22592
System.PDynArrayTypeInfo 22592
System.PDynArrayTypeInfo type 22592
System.PError 22592
System.PError type 22592
System.PExtended 22593
System.PExtended type 22593
System.PGUID 22593
System.PGUID type 22593
System.Pi 22497
System.Pi function 22497
System.PInitContext 22593
System.PInitContext type 22593
System.PInt64 22594
System.PInt64 type 22594
System.PInteger 22594
System.PInteger type 22594
System.PIntegerArray 22595
System.PIntegerArray type 22595
System.PLibModule 22596
System.PLibModule type 22596
System.PLongint 22596
System.PLongint type 22596
System.PLongWord 22596
System.PLongWord type 22596
System.PMemoryManager 22597
System.PMemoryManager type 22597
System.PMemoryManagerEx 22597
System.PMemoryManagerEx type 22597
System.PModuleUnloadRec 22598
System.PModuleUnloadRec type 22598
System.PointerArray 22598
System.PointerArray type 22598
System.POleVariant 22598
System.POleVariant type 22598
System.Pos 22497
System.Pos function 22497
System.PPAnsiChar 22599
System.PPAnsiChar type 22599
System.PPChar 22599

System.PPChar type 22599
System.PPCharArray 22599
System.PPCharArray type 22599
System.PPDispatch 22600
System.PPDispatch type 22600
System.PPointer 22600
System.PPointer type 22600
System.PPointerArray 22600
System.PPointerArray type 22600
System.PPUnknown 22601
System.PPUnknown type 22601
System.PPWideChar 22601
System.PPWideChar type 22601
System.Pred 22499
System.Pred function 22499
System.PResStringRec 22601
System.PResStringRec type 22601
System.PShortInt 22602
System.PShortInt type 22602
System.PShortString 22602
System.PShortString type 22602
System.PSingle 22602
System.PSingle type 22602
System.PSmallInt 22603
System.PSmallInt type 22603
System.PString 22603
System.PString type 22603
System.PTextBuf 22604
System.PTextBuf type 22604
System.Ptr 22501
System.Ptr function 22501
System.PUCS2Char 22604
System.PUCS2Char type 22604
System.PUCS4Char 22604
System.PUCS4Char type 22604
System.PUCS4CharArray 22605
System.PUCS4CharArray type 22605
System.PUCS4Chars 22501
System.PUCS4Chars function 22501
System.PUnitEntryTable 22605
System.PUnitEntryTable type 22605
System.PUnknown 22605
System.PUnknown type 22605
System.PUTF8String 22606
System.PUTF8String type 22606
System.PVarArray 22606
System.PVarArray type 22606
System.PVarArrayBoundArray 22606
System.PVarArrayBoundArray type 22606
System.PVarArrayCoorArray 22607
System.PVarArrayCoorArray type 22607
System.PVarData 22607
System.PVarData type 22607
System.PVariant 22607
System.PVariant type 22607
System.PVariantManager 22608
System.PVariantManager type 22608
System.PWideString 22608
System.PWideString type 22608
System.PWord 22609
System.PWord type 22609
System.PWordBool 22609
System.PWordBool type 22609
System.RaiseExceptionProc 22653
System.RaiseExceptionProc variable 22653
System.RaiseList 22502
System.RaiseList function 22502
System.Random 22502
System.Random function 22502
System.Randomize 22503
System.Randomize function 22503
System.RandSeed 22654
System.RandSeed variable 22654
System.Read 22504
System.Read function 22504
System.ReadLn 22506
System.ReadLn function 22506
System.ReallocMem 22507

System.ReallocMem function 22507
System.ReallocMemory 22508
System.ReallocMemory function 22508
System.RegisterExpectedMemoryLeak 22508
System.RegisterExpectedMemoryLeak function 22508
System.RegisterModule 22509
System.RegisterModule function 22509
System.ReleaseExceptionObject 22509
System.ReleaseExceptionObject function 22509
System.RemoveModuleUnloadProc 22510
System.RemoveModuleUnloadProc function 22510
System.Rename 22510
System.Rename function 22510
System.ReportMemoryLeaksOnShutdown 22654
System.ReportMemoryLeaksOnShutdown variable 22654
System.Reset 22511
System.Reset function 22511
System.Rewrite 22513
System.Rewrite function 22513
System.RmDir 22515
System.RmDir function 22515
System.Round 22516
System.Round function 22516
System.RTLUnwindProc 22655
System.RTLUnwindProc variable 22655
System.RTLVersion 22677
System.RTLVersion constant 22677
System.RunError 22517
System.RunError function 22517
System.S_FALSE 22677
System.S_FALSE constant 22677
System.S_OK 22677
System.S_OK constant 22677
System.SafeCallErrorProc 22655
System.SafeCallErrorProc variable 22655
System.Seek 22518
System.Seek function 22518
System.SeekEof 22519
System.SeekEof function 22519
System.SeekEoln 22520
System.SeekEoln function 22520
System.Set8087CW 22520
System.Set8087CW function 22520
System.SetInOutRes 22521
System.SetInOutRes function 22521
System.SetLastError 22521
System.SetLastError function 22521
System.SetLength 22522
System.SetLength function 22522
System.SetLineBreakStyle 22522
System.SetLineBreakStyle function 22522
System.SetMemoryManager 22523
System.SetMemoryManager function 22523
System.SetMinimumBlockAlignment 22524
System.SetMinimumBlockAlignment function 22524
System.SetMultiByteConversionCodePage 22524
System.SetMultiByteConversionCodePage function 22524
System.SetRaiseList 22525
System.SetRaiseList function 22525
System.SetString 22525
System.SetString function 22525
System.SetTextBuf 22526
System.SetTextBuf function 22526
System.SetVariantManager 22527
System.SetVariantManager function 22527
System.ShareMemoryManager 22527
System.ShareMemoryManager function 22527
System.Sin 22528
System.Sin function 22528
System.SizeOf 22528
System.SizeOf function 22528
System.Slice 22529
System.Slice function 22529
System.sLineBreak 22678
System.sLineBreak constant 22678
System.Sqr 22529
System.Sqr function 22529
System.Sqrt 22530

System.Sqrt function 22530
System.Str 22530
System.Str function 22530
System.StringOfChar 22531
System.StringOfChar function 22531
System.StringToOleStr 22531
System.StringToOleStr function 22531
System.StringToWideChar 22532
System.StringToWideChar function 22532
System.Succ 22532
System.Succ function 22532
System.Swap 22533
System.Swap function 22533
System.SysAllocMem 22533
System.SysAllocMem function 22533
System.SysFreeMem 22534
System.SysFreeMem function 22534
System.SysGetMem 22534
System.SysGetMem function 22534
System.SysReallocMem 22535
System.SysReallocMem function 22535
System.SysRegisterExpectedMemoryLeak 22536
System.SysRegisterExpectedMemoryLeak function 22536
System.SystemThreadEndProc 22656
System.SystemThreadEndProc variable 22656
System.SystemThreadFuncProc 22656
System.SystemThreadFuncProc variable 22656
System.SysUnregisterExpectedMemoryLeak 22536
System.SysUnregisterExpectedMemoryLeak function 22536
System.TAssertErrorProc 22610
System.TAssertErrorProc type 22610
System.TBoundArray 22610
System.TBoundArray type 22610
System.TCallDesc 22553
System.TCallDesc record 22553
System.TChunkStatus 22554
System.TChunkStatus enumeration 22554
System.TClass 22611
System.TClass type 22611
System.TClassHelperBase 22251
System.TClassHelperBase._Create 22253
System.TClassHelperBase.FInstance 22253
System.TCVModInfo 22555
System.TCVModInfo record 22555
System.TDateTime 22612
System.TDateTime type 22612
System.TDispatchMessage 22555
System.TDispatchMessage record 22555
System.TDispDesc 22556
System.TDispDesc record 22556
System.TDLLProc 22613
System.TDLLProc type 22613
System.TDLLProcEx 22613
System.TDLLProcEx type 22613
System.TDynArrayTypeInfo 22556
System.TDynArrayTypeInfo record 22556
System.TEnumModuleFunc 22614
System.TEnumModuleFunc type 22614
System.TEnumModuleFuncLW 22614
System.TEnumModuleFuncLW type 22614
System.Test8086 22656
System.Test8086 variable 22656
System.Test8087 22657
System.Test8087 variable 22657
System.TestFDIV 22657
System.TestFDIV variable 22657
System.TextStart 22536
System.TextStart function 22536
System.tfCRLF 22678
System.tfCRLF constant 22678
System.TFileIOFunc 22615
System.TFileIOFunc type 22615
System.TFileRec 22557
System.TFileRec record 22557
System.TGUID 22558
System.TGUID record 22558
System.THandle 22615
System.THandle type 22615

System.THeapStatus 22558
System.THeapStatus record 22558
System.TInitContext 22560
System.TInitContext record 22560
System.TInterfacedClass 22616
System.TInterfacedClass type 22616
System.TLibModule 22561
System.TLibModule record 22561
System.TMemoryManager 22562
System.TMemoryManager record 22562
System.TMemoryManagerEx 22563
System.TMemoryManagerEx record 22563
System.TMemoryManagerState 22564
System.TMemoryManagerState record 22564
System.TMemoryMap 22617
System.TMemoryMap type 22617
System.TMethod 22565
System.TMethod record 22565
System.TMinimumBlockAlignment 22566
System.TMinimumBlockAlignment enumeration 22566
System.TModuleUnloadProc 22617
System.TModuleUnloadProc type 22617
System.TModuleUnloadProcLW 22618
System.TModuleUnloadProcLW type 22618
System.TModuleUnloadRec 22566
System.TModuleUnloadRec record 22566
System.TPCharArray 22618
System.TPCharArray type 22618
System.TResourceHandle 22619
System.TResourceHandle type 22619
System.TResStringRec 22567
System.TResStringRec record 22567
System.Trunc 22537
System.Trunc function 22537
System.Truncate 22537
System.Truncate function 22537
System.TRuntimeError 22567
System.TRuntimeError enumeration 22567
System.TSafeCallErrorProc 22619
System.TSafeCallErrorProc type 22619
System.TSmallBlockTypeState 22568
System.TSmallBlockTypeState record 22568
System.TSmallBlockTypeStates 22620
System.TSmallBlockTypeStates type 22620
System.TSystemThreadEndProc 22621
System.TSystemThreadEndProc type 22621
System.TSystemThreadFuncProc 22621
System.TSystemThreadFuncProc type 22621
System.TTextBuf 22622
System.TTextBuf type 22622
System.TTextIOFunc 22622
System.TTextIOFunc type 22622
System.TTextLineBreakStyle 22569
System.TTextLineBreakStyle enumeration 22569
System.TTextRec 22569
System.TTextRec record 22569
System.TThreadFunc 22622
System.TThreadFunc type 22622
System.TUCS4CharArray 22623
System.TUCS4CharArray type 22623
System.TVarArray 22570
System.TVarArray record 22570
System.TVarArrayBound 22571
System.TVarArrayBound record 22571
System.TVarArrayBoundArray 22623
System.TVarArrayBoundArray type 22623
System.TVarArrayCoorArray 22623
System.TVarArrayCoorArray type 22623
System.TVarData 22571
System.TVarData record 22571
System.TVariantManager 22575
System.TVariantManager record 22575
System.TVarOp 22624
System.TVarOp type 22624
System.TVarType 22625
System.TVarType type 22625
System.UCS2Char 22626
System.UCS2Char type 22626

System.UCS4Char 22626
System.UCS4Char type 22626
System.UCS4String 22627
System.UCS4String type 22627
System.UCS4StringToWideString 22538
System.UCS4StringToWideString function 22538
System.UnicodeToUtf8 22538
System.UnicodeToUtf8 function 22538
System.UniqueString 22539
System.UniqueString function 22539
System.UnitEntryTable 22627
System.UnitEntryTable type 22627
System.UnregisterExpectedMemoryLeak 22539
System.UnregisterExpectedMemoryLeak function 22539
System.UnregisterModule 22540
System.UnregisterModule function 22540
System.UpCase 22540
System.UpCase function 22540
System.UTF8Decode 22541
System.UTF8Decode function 22541
System.UTF8Encode 22541
System.UTF8Encode function 22541
System.UTF8String 22627
System.UTF8String type 22627
System.Utf8ToAnsi 22542
System.Utf8ToAnsi function 22542
System.Utf8ToUnicode 22542
System.Utf8ToUnicode function 22542
System.Val 22543
System.Val function 22543
System.VarAddRefProc 22657
System.VarAddRefProc variable 22657
System.varAny 22678
System.varAny constant 22678
System.varArray 22679
System.varArray constant 22679
System.VarArrayRedim 22544
System.VarArrayRedim function 22544
System.varBoolean 22680
System.varBoolean constant 22680
System.varByRef 22680
System.varByRef constant 22680
System.varByte 22681
System.varByte constant 22681
System.VarClear 22545
System.VarClear function 22545
System.VarClearProc 22658
System.VarClearProc variable 22658
System.VarCopyProc 22658
System.VarCopyProc variable 22658
System.varCurrency 22681
System.varCurrency constant 22681
System.varDate 22682
System.varDate constant 22682
System.varDispatch 22682
System.varDispatch constant 22682
System.varDouble 22683
System.varDouble constant 22683
System.varEmpty 22683
System.varEmpty constant 22683
System.varError 22684
System.varError constant 22684
System.varInt64 22684
System.varInt64 constant 22684
System.varInteger 22685
System.varInteger constant 22685
System.varLongWord 22685
System.varLongWord constant 22685
System.varNull 22686
System.varNull constant 22686
System.varOleStr 22686
System.varOleStr constant 22686
System.varShortInt 22687
System.varShortInt constant 22687
System.varSingle 22687
System.varSingle constant 22687
System.varSmallint 22688
System.varSmallint constant 22688

System.varStrArg 22688
System.varStrArg constant 22688
System.varString 22688
System.varString constant 22688
System.VarToLStrProc 22658
System.VarToLStrProc variable 22658
System.VarToWStrProc 22659
System.VarToWStrProc variable 22659
System.varTypeMask 22689
System.varTypeMask constant 22689
System.varUnknown 22690
System.varUnknown constant 22690
System.varVariant 22690
System.varVariant constant 22690
System.varWord 22691
System.varWord constant 22691
System.vmtAddRef 22691
System.vmtAddRef constant 22691
System.vmtAfterConstruction 22691
System.vmtAfterConstruction constant 22691
System.vmtAutoTable 22692
System.vmtAutoTable constant 22692
System.vmtBeforeDestruction 22692
System.vmtBeforeDestruction constant 22692
System.vmtClassName 22692
System.vmtClassName constant 22692
System.vmtCreateObject 22693
System.vmtCreateObject constant 22693
System.vmtDefaultHandler 22693
System.vmtDefaultHandler constant 22693
System.vmtDestroy 22693
System.vmtDestroy constant 22693
System.vmtDispatch 22694
System.vmtDispatch constant 22694
System.vmtDynamicTable 22694
System.vmtDynamicTable constant 22694
System.vmtFieldTable 22694
System.vmtFieldTable constant 22694
System.vmtFreeInstance 22695
System.vmtFreeInstance constant 22695
System.vmtInitTable 22695
System.vmtInitTable constant 22695
System.vmtInstanceSize 22695
System.vmtInstanceSize constant 22695
System.vmtIntfTable 22696
System.vmtIntfTable constant 22696
System.vmtMethodTable 22696
System.vmtMethodTable constant 22696
System.vmtNewInstance 22696
System.vmtNewInstance constant 22696
System.vmtParent 22697
System.vmtParent constant 22697
System.vmtQueryInterface 22697
System.vmtQueryInterface constant 22697
System.vmtRelease 22697
System.vmtRelease constant 22697
System.vmtSafeCallException 22698
System.vmtSafeCallException constant 22698
System.vmtSelfPtr 22698
System.vmtSelfPtr constant 22698
System.vmtTypeInfo 22698
System.vmtTypeInfo constant 22698
System.vtAnsiString 22699
System.vtAnsiString constant 22699
System.vtBoolean 22699
System.vtBoolean constant 22699
System.vtChar 22700
System.vtChar constant 22700
System.vtClass 22700
System.vtClass constant 22700
System.vtCurrency 22701
System.vtCurrency constant 22701
System.vtExtended 22701
System.vtExtended constant 22701
System.vtInt64 22702
System.vtInt64 constant 22702
System.vtInteger 22702
System.vtInteger constant 22702

System.vtInterface 22703
System.vtInterface constant 22703
System.vtObject 22703
System.vtObject constant 22703
System.vtPChar 22704
System.vtPChar constant 22704
System.vtPointer 22704
System.vtPointer constant 22704
System.vtPWideChar 22705
System.vtPWideChar constant 22705
System.vtString 22705
System.vtString constant 22705
System.vtVariant 22706
System.vtVariant constant 22706
System.vtWideChar 22706
System.vtWideChar constant 22706
System.vtWideString 22706
System.vtWideString constant 22706
System.WideCharLenToString 22546
System.WideCharLenToString function 22546
System.WideCharLenToStrVar 22546
System.WideCharLenToStrVar function 22546
System.WideCharToString 22547
System.WideCharToString function 22547
System.WideCharToStrVar 22547
System.WideCharToStrVar function 22547
System.WideStringToUCS4String 22548
System.WideStringToUCS4String function 22548
System.Write 22548
System.Write function 22548
System.WriteLn 22550
System.WriteLn function 22550
System::!= 22405
System::!= function 22405
System::__currMul 22405
System::__currMul function 22405
System::_roundToInt64 22406
System::_roundToInt64 function 22406
System::+ 22406
System::+ function 22406
System::< 22407
System::< function 22407
System::<< 22407, 22408
System::<< function 22407, 22408
System::<= 22408
System::<= function 22408
System::== 22408
System::== function 22408
System::> 22409
System::> function 22409
System::>= 22409
System::>= function 22409
System::>> 22409, 22410
System::>> function 22409, 22410
System::AnsiString 22093
System::AnsiString::!= 22123
System::AnsiString::[] 22123
System::AnsiString::~AnsiString 22096
System::AnsiString::+ 22124
System::AnsiString::+= 22124
System::AnsiString::< 22124
System::AnsiString::<= 22125
System::AnsiString::= 22125
System::AnsiString::== 22126
System::AnsiString::> 22126
System::AnsiString::>= 22126
System::AnsiString::AnsiCompare 22103
System::AnsiString::AnsiCompareIC 22104
System::AnsiString::AnsiLastChar 22104
System::AnsiString::AnsiPos 22104
System::AnsiString::AnsiString 22097, 22098, 22099, 22100
System::AnsiString::ByteType 22105
System::AnsiString::c_str 22105
System::AnsiString::cat_printf 22106
System::AnsiString::cat_sprintf 22106
System::AnsiString::cat_vprintf 22107
System::AnsiString::CurrToStr 22107
System::AnsiString::CurrToStrF 22108

System::AnsiString::data 22108
System::AnsiString::Delete 22108
System::AnsiString::FloatToStrF 22109
System::AnsiString::FmtLoadStr 22110
System::AnsiString::Format 22110
System::AnsiString::FormatFloat 22110
System::AnsiString::GetRec 22112
System::AnsiString::Insert 22112
System::AnsiString::IntToHex 22113
System::AnsiString::IsDelimiter 22113
System::AnsiString::IsEmpty 22113
System::AnsiString::IsLeadByte 22114
System::AnsiString::IsPathDelimiter 22114
System::AnsiString::IsTrailByte 22114
System::AnsiString::LastDelimiter 22115
System::AnsiString::Length 22115
System::AnsiString::LoadStr 22115
System::AnsiString::LoadString 22116
System::AnsiString::LowerCase 22116
System::AnsiString::Pos 22117
System::AnsiString::printf 22117
System::AnsiString::SetLength 22117
System::AnsiString::sprintf 22118
System::AnsiString::StringOfChar 22118
System::AnsiString::StrRec 22100
System::AnsiString::StrRec structure 22100
System::AnsiString::SubString 22118
System::AnsiString::ThrowIfOutOfRange 22119
System::AnsiString::ToDouble 22119
System::AnsiString::ToInt 22119
System::AnsiString::ToIntDef 22120
System::AnsiString::Trim 22120
System::AnsiString::TrimLeft 22120
System::AnsiString::TrimRight 22120
System::AnsiString::TStringFloatFormat 22101
System::AnsiString::TStringFloatFormat enumeration 22101
System::AnsiString::TStringMbcsByteType 22101
System::AnsiString::TStringMbcsByteType enumeration 22101
System::AnsiString::Unique 22121
System::AnsiString::UpperCase 22121
System::AnsiString::vprintf 22121
System::AnsiString::WideChar 22122
System::AnsiString::WideCharBufSize 22122
System::ArrayOfboolean 22581
System::ArrayOfboolean type 22581
System::ArrayOfbyte 22582
System::ArrayOfbyte type 22582
System::ArrayOfdouble 22582
System::ArrayOfdouble type 22582
System::ArrayOfint 22582
System::ArrayOfint type 22582
System::ArrayOflong 22583
System::ArrayOflong type 22583
System::ArrayOfshort 22583
System::ArrayOfshort type 22583
System::ArrayOfstring 22583
System::ArrayOfstring type 22583
System::ArrayOfwidestring 22584
System::ArrayOfwidestring type 22584
System::AutoCmd 22127
System::AutoCmd::~AutoCmd 22129
System::AutoCmd::<< 22139, 22140, 22141, 22142, 22143, 22144
System::AutoCmd::AddElement 22132
System::AutoCmd::AutoCmd 22129
System::AutoCmd::Clear 22132
System::AutoCmd::ClearArgs 22132
System::AutoCmd::ClearName 22133
System::AutoCmd::GetArgCount 22133
System::AutoCmd::GetArgType 22134
System::AutoCmd::GetCallType 22134
System::AutoCmd::GetName 22134
System::AutoCmd::GetNamedArgCount 22135
System::AutoCmd::GetNamedArgType 22135
System::AutoCmd::GetNamedParm 22136
System::AutoCmd::GetNamedParmName 22136
System::AutoCmd::GetParm 22137
System::AutoCmd::Name 22130
System::AutoCmd::NamedParmNames 22130

System::AutoCmd::NamedParms 22130
System::AutoCmd::NamedParmTypes 22131
System::AutoCmd::Parms 22131
System::AutoCmd::ParmTypes 22131
System::AutoCmd::RequestResult 22137
System::AutoCmd::SetName 22138
System::Currency 22145
System::Currency::- 22149
System::Currency::-- 22150
System::Currency::! 22150
System::Currency::!= 22151
System::Currency::% 22151
System::Currency::%= 22152
System::Currency::* 22152
System::Currency::*= 22153
System::Currency::/ 22154
System::Currency::/= 22154
System::Currency::+ 22155
System::Currency::++ 22156
System::Currency::+= 22156
System::Currency::< 22157
System::Currency::<= 22158
System::Currency::= 22159
System::Currency::-= 22159
System::Currency::== 22160
System::Currency::> 22161
System::Currency::>= 22161
System::Currency::AnsiString 22162
System::Currency::Currency 22147
System::Currency::double 22162
System::Currency::int 22163
System::CurrencyBase 22166
System::CurrencyBase::Val 22166
System::DelphiInterface 22167
System::DelphiInterface::! 22170
System::DelphiInterface::& 22170
System::DelphiInterface::* 22170
System::DelphiInterface::~DelphiInterface 22168
System::DelphiInterface::= 22171
System::DelphiInterface::-> 22171
System::DelphiInterface::DelphiInterface 22168
System::DelphiInterface::Release 22169
System::DelphiInterface::T* 22171
System::DynamicArray 22172
System::DynamicArray::[] 22179
System::DynamicArray::~DynamicArray 22174
System::DynamicArray::= 22180
System::DynamicArray::== 22180
System::DynamicArray::AllocData 22176
System::DynamicArray::Copy 22176
System::DynamicArray::CopyRange 22177
System::DynamicArray::DecRefCount 22177
System::DynamicArray::DimCount 22177
System::DynamicArray::DynamicArray 22175
System::DynamicArray::FreeData 22178
System::DynamicArray::get_high 22178
System::DynamicArray::get_length 22178
System::DynamicArray::get_low 22178
System::DynamicArray::High 22181
System::DynamicArray::IncRefCount 22178
System::DynamicArray::Length 22181
System::DynamicArray::Low 22182
System::DynamicArray::set_length 22179
System::DynamicArray::SetData 22179
System::DynArrayException 22182
System::DynArrayNullData 22183
System::DynArrayOutOfRange 22183
System::DynArrayOutOfRange::DynArrayOutOfRange 22184
System::DynArrayOutOfRange::m_Count 22184
System::DynArrayOutOfRange::m_Index 22184
System::Function 22184
System::Function::Function 22186
System::Function::GetCallType 22187
System::Function::RequestResult 22187
System::MaxDispArgs 22651
System::MaxDispArgs variable 22651
System::NamedParm 22187
System::NamedParm::AddElement 22192

System::NamedParm::GetParm 22192
System::NamedParm::GetParmName 22193
System::NamedParm::GetType 22193
System::NamedParm::Name 22191
System::NamedParm::NamedParm 22190
System::NamedParm::Parm 22191
System::NamedParm::Type 22192
System::OleVariant 22193
System::OleVariant::!= 22210
System::OleVariant::~OleVariant 22197
System::OleVariant::< 22210
System::OleVariant::<= 22211
System::OleVariant::= 22213
System::OleVariant::== 22214
System::OleVariant::> 22215
System::OleVariant::>= 22215
System::OleVariant::OleVariant 22200, 22201, 22202, 22203, 22204, 22205, 22206, 22207, 22208, 22209
System::OpenArray 22216
System::OpenArray::~OpenArray 22216
System::OpenArray::= 22223
System::OpenArray::GetHigh 22223
System::OpenArray::OpenArray 22217, 22218, 22219, 22220, 22221, 22222
System::OpenArray::T* 22223
System::OpenArrayCount 22223
System::OpenArrayCount::GetHigh 22230
System::OpenArrayCount::OpenArrayCount 22225, 22226, 22227, 22228, 22229, 22230
System::PCallDesc 22588
System::PCallDesc type 22588
System::PDispDesc 22591
System::PDispDesc type 22591
System::PInterfaceEntry 22595
System::PInterfaceEntry type 22595
System::PInterfaceTable 22595
System::PInterfaceTable type 22595
System::Procedure 22230
System::Procedure::GetCallType 22232
System::Procedure::Procedure 22232
System::Procedure::RequestResult 22233
System::PropertyGet 22233
System::PropertyGet::GetCallType 22235
System::PropertyGet::PropertyGet 22235
System::PropertyGet::RequestResult 22236
System::PropertySet 22236
System::PropertySet::GetCallType 22238
System::PropertySet::PropertySet 22238
System::PropertySet::RequestResult 22239
System::PVarRec 22608
System::PVarRec type 22608
System::SCODE 22609
System::SCODE type 22609
System::SmallString 22239
System::SmallString::[] 22241
System::SmallString::= 22241
System::SmallString::AnsiString 22242
System::SmallString::SmallString 22240, 22241
System::SmallStringBase 22242
System::SmallStringBase::Data 22243
System::SmallStringBase::Len 22243
System::sPointer 22243
System::sPointer::& 22245
System::sPointer::~sPointer 22244
System::sPointer::-> 22245
System::sPointer::sPointer 22244
System::StaticArray 22245
System::StaticArray::[] 22247
System::StaticArray::const T* 22247
System::StaticArray::data 22246
System::StaticArray::T* 22247
System::TAggregatedObject 22248
System::TAggregatedObject::_AddRef 22250
System::TAggregatedObject::_Release 22250
System::TAggregatedObject::Controller 22251
System::TAggregatedObject::Create 22249
System::TAggregatedObject::QueryInterface 22250
System::TBooleanDynArray 22610
System::TBooleanDynArray type 22610

System::TByteDynArray 22611
System::TByteDynArray type 22611
System::TCallDesc 22578
System::TCallDesc structure 22578
System::TCardinalDynArray 22611
System::TCardinalDynArray type 22611
System::TContainedObject 22253
System::TContainedObject::QueryInterface 22255
System::TDateTime 22256
System::TDateTime::- 22267
System::TDateTime::-- 22268
System::TDateTime::!= 22268
System::TDateTime::+ 22269
System::TDateTime::++ 22269
System::TDateTime::+= 22270
System::TDateTime::< 22270
System::TDateTime::<= 22271
System::TDateTime::= 22271
System::TDateTime::-= 22272
System::TDateTime::== 22273
System::TDateTime::> 22273
System::TDateTime::>= 22273
System::TDateTime::AnsiString 22273
System::TDateTime::CurrentDate 22261
System::TDateTime::CurrentDateTime 22261
System::TDateTime::CurrentTime 22262
System::TDateTime::DateString 22262
System::TDateTime::DateTimeString 22262
System::TDateTime::DayOfWeek 22263
System::TDateTime::DecodeDate 22263
System::TDateTime::DecodeTime 22263
System::TDateTime::double 22274
System::TDateTime::FileDate 22264
System::TDateTime::FileDateToDateTime 22264
System::TDateTime::FormatString 22265
System::TDateTime::int 22274
System::TDateTime::TDateTime 22259
System::TDateTime::TDateTimeFlag 22260
System::TDateTime::TDateTimeFlag enumeration 22260
System::TDateTime::TimeString 22266
System::TDateTimeBase 22275
System::TDateTimeBase::Val 22275
System::TDispDesc 22578
System::TDispDesc structure 22578
System::TDoubleDynArray 22614
System::TDoubleDynArray type 22614
System::TInt64DynArray 22615
System::TInt64DynArray type 22615
System::TIntegerDynArray 22616
System::TIntegerDynArray type 22616
System::TInterfacedObject 22276
System::TInterfacedObject::_AddRef 22278
System::TInterfacedObject::_Release 22279
System::TInterfacedObject::AfterConstruction 22279
System::TInterfacedObject::BeforeDestruction 22279
System::TInterfacedObject::FRefCount 22278
System::TInterfacedObject::NewInstance 22280
System::TInterfacedObject::QueryInterface 22280
System::TInterfacedObject::RefCount 22281
System::TInterfaceEntry 22560
System::TInterfaceEntry record 22560
System::TInterfaceTable 22561
System::TInterfaceTable record 22561
System::TLongWordDynArray 22616
System::TLongWordDynArray type 22616
System::TMetaClass 22281
System::TMetaClass::ClassInfo 22283
System::TMetaClass::ClassName 22283
System::TMetaClass::ClassNameIs 22284
System::TMetaClass::ClassParent 22284
System::TMetaClass::InheritsFrom 22284
System::TMetaClass::InitInstance 22284
System::TMetaClass::InstanceSize 22285
System::TMetaClass::MethodAddress 22285
System::TMetaClass::MethodName 22285
System::TObject 22285
System::TObject::AfterConstruction 22289
System::TObject::BeforeDestruction 22289

System::TObject::ClassInfo 22290
System::TObject::ClassName 22290
System::TObject::ClassNameIs 22293
System::TObject::ClassParent 22293
System::TObject::ClassType 22295
System::TObject::CleanupInstance 22298
System::TObject::Create 22287
System::TObject::DefaultHandler 22299
System::TObject::Destroy 22287
System::TObject::Dispatch 22299
System::TObject::FieldAddress 22299
System::TObject::Free 22300
System::TObject::FreeInstance 22303
System::TObject::GetInterface 22303
System::TObject::GetInterfaceEntry 22304
System::TObject::GetInterfaceTable 22304
System::TObject::InheritsFrom 22305
System::TObject::InitInstance 22305
System::TObject::InstanceSize 22306
System::TObject::MethodAddress 22306
System::TObject::MethodName 22306
System::TObject::NewInstance 22306
System::TObject::SafeCallException 22307
System::TShortIntDynArray 22619
System::TShortIntDynArray type 22619
System::TSingleDynArray 22620
System::TSingleDynArray type 22620
System::TSmallIntDynArray 22620
System::TSmallIntDynArray type 22620
System::TStringDynArray 22621
System::TStringDynArray type 22621
System::TVarRec 22577
System::TVarRec record 22577
System::TWideStringDynArray 22625
System::TWideStringDynArray type 22625
System::TWordDynArray 22626
System::TWordDynArray type 22626
System::VarArrayRedim 22544
System::VarArrayRedim function 22544
System::VarCast 22545
System::VarCast function 22545
System::VarClear 22546
System::VarClear function 22546
System::Variant 22307
System::Variant::- 22343, 22344
System::Variant::! 22344
System::Variant::!= 22345
System::Variant::% 22346
System::Variant::%= 22346
System::Variant::& 22347
System::Variant::&= 22347
System::Variant::* 22348
System::Variant::*= 22348
System::Variant::/ 22349
System::Variant::/= 22349
System::Variant::^ 22350
System::Variant::^= 22350
System::Variant::__int64 22350
System::Variant::| 22351
System::Variant::|= 22351
System::Variant::~Variant 22311
System::Variant::+ 22352
System::Variant::+= 22352
System::Variant::< 22353
System::Variant::<< 22354
System::Variant::<<= 22354
System::Variant::<= 22355
System::Variant::= 22359
System::Variant::-= 22361
System::Variant::== 22361
System::Variant::> 22362
System::Variant::>= 22363
System::Variant::>> 22364
System::Variant::>>= 22365
System::Variant::AnsiString 22365
System::Variant::ArrayDimCount 22317
System::Variant::ArrayHighBound 22317
System::Variant::ArrayLock 22317

System::Variant::ArrayLowBound 22318
System::Variant::ArrayRedim 22318
System::Variant::ArrayUnlock 22318
System::Variant::Assign 22318
System::Variant::AsType 22319
System::Variant::bool 22365
System::Variant::ChangeType 22319
System::Variant::char 22365
System::Variant::Clear 22319
System::Variant::CreateObject 22320
System::Variant::Currency 22366
System::Variant::CURRENCY 22366
System::Variant::Currency* 22366
System::Variant::DECIMAL 22366
System::Variant::double 22367
System::Variant::double* 22367
System::Variant::Empty 22320
System::Variant::Exec 22320
System::Variant::float 22367
System::Variant::float* 22368
System::Variant::GetActiveObject 22321
System::Variant::GetBaseVariant 22321
System::Variant::GetElement 22322
System::Variant::GetError 22322
System::Variant::IDispatch* 22368
System::Variant::IInterface* 22368
System::Variant::int 22368
System::Variant::int* 22369
System::Variant::IsArray 22323
System::Variant::IsEmpty 22323
System::Variant::IsNull 22323
System::Variant::IUnknown* 22369
System::Variant::long 22369
System::Variant::long* 22369
System::Variant::LPSAFEARRAY 22369
System::Variant::NoParam 22324
System::Variant::OleFunction 22326
System::Variant::OleProcedure 22332
System::Variant::OlePropertyGet 22337
System::Variant::OlePropertySet 22338
System::Variant::PutElement 22340
System::Variant::Sanitized 22340
System::Variant::SetError 22341
System::Variant::short 22370
System::Variant::short* 22370
System::Variant::signed char 22370
System::Variant::signed char* 22370
System::Variant::TDateTime 22371
System::Variant::TDateTime* 22371
System::Variant::TVariant 22371
System::Variant::TVariant* 22371
System::Variant::Type 22341
System::Variant::unsigned char 22372
System::Variant::unsigned char* 22372
System::Variant::unsigned int 22372
System::Variant::unsigned int* 22372
System::Variant::unsigned long 22373
System::Variant::unsigned long* 22373
System::Variant::unsigned short 22373
System::Variant::unsigned short* 22373
System::Variant::Variant 22314
System::Variant::VARIANT 22373
System::Variant::VARIANT* 22374
System::Variant::wchar_t* 22374
System::Variant::wchar_t** 22374
System::Variant::WideString 22374
System::VariantError 22375
System::VariantError::= 22377
System::VariantError::m_scode 22376
System::VariantError::SCODE 22377
System::VariantError::VariantError 22376
System::WideString 22377
System::WideString::!= 22387
System::WideString::& 22387
System::WideString::[] 22388
System::WideString::~WideString 22378
System::WideString::+ 22388
System::WideString::+= 22389

System::WideString::< 22389
System::WideString::<= 22390
System::WideString::= 22390
System::WideString::== 22391
System::WideString::> 22392
System::WideString::>= 22392
System::WideString::Attach 22381
System::WideString::BSTR 22393
System::WideString::c_bstr 22382
System::WideString::Copy 22382
System::WideString::data 22383
System::WideString::Delete 22383
System::WideString::Detach 22383
System::WideString::Empty 22384
System::WideString::Insert 22384
System::WideString::IsEmpty 22385
System::WideString::Length 22385
System::WideString::Pos 22385
System::WideString::SetLength 22385
System::WideString::SubString 22386
System::WideString::WideString 22380
SystemProperties class 2097
about SystemProperties class 2097
AssemblyPath 2099
BlackfishSqlNamespace 2100
DataDirectory 2100
LicenseDirectory 2100
LogFile 2100
LogFilters 2100
MaxSortBuffer 2101
MinCacheSize 2101
Port 2101
SystemProperties fields 2099
SystemProperties Members 2099
TempDir 2101
SysUtils 22709
SysUtils namespace 22709
Classes 22721
Constants 23038
Functions 22780
Interfaces 22778
Structs, Records, Enums 22996
Types 23012
Variables 23021
SysUtils.Abort 22786
SysUtils.Abort function 22786
SysUtils.AddExitProc 22787
SysUtils.AddExitProc function 22787
SysUtils.AddTerminateProc 22787
SysUtils.AddTerminateProc function 22787
SysUtils.AdjustLineBreaks 22788
SysUtils.AdjustLineBreaks function 22788
SysUtils.AnsiCompareFileName 22788
SysUtils.AnsiCompareFileName function 22788
SysUtils.AnsiCompareStr 22789
SysUtils.AnsiCompareStr function 22789
SysUtils.AnsiCompareText 22790
SysUtils.AnsiCompareText function 22790
SysUtils.AnsiDequotedStr 22790
SysUtils.AnsiDequotedStr function 22790
SysUtils.AnsiExtractQuotedStr 22791
SysUtils.AnsiExtractQuotedStr function 22791
SysUtils.AnsiLastChar 22791
SysUtils.AnsiLastChar function 22791
SysUtils.AnsiLowerCase 22792
SysUtils.AnsiLowerCase function 22792
SysUtils.AnsiLowerCaseFileName 22792
SysUtils.AnsiLowerCaseFileName function 22792
SysUtils.AnsiPos 22793
SysUtils.AnsiPos function 22793
SysUtils.AnsiQuotedStr 22793
SysUtils.AnsiQuotedStr function 22793
SysUtils.AnsiSameStr 22794
SysUtils.AnsiSameStr function 22794
SysUtils.AnsiSameText 22794
SysUtils.AnsiSameText function 22794
SysUtils.AnsiStrComp 22795
SysUtils.AnsiStrComp function 22795

SysUtils.AnsiStrIComp 22795
SysUtils.AnsiStrIComp function 22795
SysUtils.AnsiStrLastChar 22796
SysUtils.AnsiStrLastChar function 22796
SysUtils.AnsiStrLComp 22796
SysUtils.AnsiStrLComp function 22796
SysUtils.AnsiStrLIComp 22797
SysUtils.AnsiStrLIComp function 22797
SysUtils.AnsiStrLower 22798
SysUtils.AnsiStrLower function 22798
SysUtils.AnsiStrPos 22798
SysUtils.AnsiStrPos function 22798
SysUtils.AnsiStrRScan 22799
SysUtils.AnsiStrRScan function 22799
SysUtils.AnsiStrScan 22799
SysUtils.AnsiStrScan function 22799
SysUtils.AnsiStrUpper 22800
SysUtils.AnsiStrUpper function 22800
SysUtils.AnsiUpperCase 22800
SysUtils.AnsiUpperCase function 22800
SysUtils.AnsiUpperCaseFileName 22801
SysUtils.AnsiUpperCaseFileName function 22801
SysUtils.AppendStr 22801
SysUtils.AppendStr function 22801
SysUtils.AssignStr 22802
SysUtils.AssignStr function 22802
SysUtils.Beep 22802
SysUtils.Beep function 22802
SysUtils.BoolToStr 22804
SysUtils.BoolToStr function 22804
SysUtils.ByteToCharIndex 22806
SysUtils.ByteToCharIndex function 22806
SysUtils.ByteToCharLen 22807
SysUtils.ByteToCharLen function 22807
SysUtils.ByteType 22807
SysUtils.ByteType function 22807
SysUtils.CallTerminateProcs 22808
SysUtils.CallTerminateProcs function 22808
SysUtils.ChangeFileExt 22808
SysUtils.ChangeFileExt function 22808
SysUtils.ChangeFilePath 22810
SysUtils.ChangeFilePath function 22810
SysUtils.CharLength 22810
SysUtils.CharLength function 22810
SysUtils.CharToByteIndex 22811
SysUtils.CharToByteIndex function 22811
SysUtils.CharToByteLen 22811
SysUtils.CharToByteLen function 22811
SysUtils.CheckWin32Version 22812
SysUtils.CheckWin32Version function 22812
SysUtils.CompareMem 22812
SysUtils.CompareMem function 22812
SysUtils.CompareStr 22813
SysUtils.CompareStr function 22813
SysUtils.CompareText 22814
SysUtils.CompareText function 22814
SysUtils.CreateDir 22817
SysUtils.CreateDir function 22817
SysUtils.CreateGUID 22819
SysUtils.CreateGUID function 22819
SysUtils.CurrencyDecimals 23022
SysUtils.CurrencyDecimals variable 23022
SysUtils.CurrencyFormat 23022
SysUtils.CurrencyFormat variable 23022
SysUtils.CurrencyString 23023
SysUtils.CurrencyString variable 23023
SysUtils.CurrentYear 22819
SysUtils.CurrentYear function 22819
SysUtils.CurrToStr 22819
SysUtils.CurrToStr function 22819
SysUtils.CurrToStrF 22820
SysUtils.CurrToStrF function 22820
SysUtils.Date 22820
SysUtils.Date function 22820
SysUtils.DateDelta 23039
SysUtils.DateDelta constant 23039
SysUtils.DateSeparator 23023
SysUtils.DateSeparator variable 23023

SysUtils.DateTimeToFileDate 22821
SysUtils.DateTimeToFileDate function 22821
SysUtils.DateTimeToStr 22822
SysUtils.DateTimeToStr function 22822
SysUtils.DateTimeToString 22823
SysUtils.DateTimeToString function 22823
SysUtils.DateTimeToSystemTime 22825
SysUtils.DateTimeToSystemTime function 22825
SysUtils.DateTimeToTimeStamp 22825
SysUtils.DateTimeToTimeStamp function 22825
SysUtils.DateToStr 22825
SysUtils.DateToStr function 22825
SysUtils.DayOfWeek 22827
SysUtils.DayOfWeek function 22827
SysUtils.DecimalSeparator 23023
SysUtils.DecimalSeparator variable 23023
SysUtils.DecodeDate 22828
SysUtils.DecodeDate function 22828
SysUtils.DecodeDateFully 22829
SysUtils.DecodeDateFully function 22829
SysUtils.DecodeTime 22830
SysUtils.DecodeTime function 22830
SysUtils.DefaultFalseBoolStr 23039
SysUtils.DefaultFalseBoolStr constant 23039
SysUtils.DefaultTrueBoolStr 23040
SysUtils.DefaultTrueBoolStr constant 23040
SysUtils.DeleteFile 22831
SysUtils.DeleteFile function 22831
SysUtils.DirectoryExists 22832
SysUtils.DirectoryExists function 22832
SysUtils.DiskFree 22833
SysUtils.DiskFree function 22833
SysUtils.DiskSize 22835
SysUtils.DiskSize function 22835
SysUtils.DisposeStr 22836
SysUtils.DisposeStr function 22836
SysUtils.DriveDelim 23041
SysUtils.DriveDelim constant 23041
SysUtils.EAbort 22722
SysUtils.EAbstractError 22722
SysUtils.EAccessViolation 22723
SysUtils.EAssertionFailed 22723
SysUtils.EControlC 22724
SysUtils.EConvertError 22724
SysUtils.EDivByZero 22726
SysUtils.EExternal 22727
SysUtils.EExternal.ExceptionRecord 22729
SysUtils.EExternalException 22729
SysUtils.EHeapException 22730
SysUtils.EHeapException.FreeInstance 22732
SysUtils.EInOutError 22732
SysUtils.EInOutError.ErrorCode 22735
SysUtils.EIntError 22735
SysUtils.EIntfCastError 22736
SysUtils.EIntOverflow 22736
SysUtils.EInvalidCast 22737
SysUtils.EInvalidContainer 22738
SysUtils.EInvalidInsert 22738
SysUtils.EInvalidOp 22738
SysUtils.EInvalidPointer 22739
SysUtils.EMathError 22739
SysUtils.EmptyStr 23024
SysUtils.EmptyStr variable 23024
SysUtils.EmptyWideStr 23024
SysUtils.EmptyWideStr variable 23024
SysUtils.EncodeDate 22836
SysUtils.EncodeDate function 22836
SysUtils.EncodeTime 22837
SysUtils.EncodeTime function 22837
SysUtils.EOSError 22740
SysUtils.EOSError.ErrorCode 22742
SysUtils.EOutOfMemory 22742
SysUtils.EOverflow 22743
SysUtils.EPackageError 22744
SysUtils.EPrivilege 22744
SysUtils.EPropReadOnly 22745
SysUtils.EPropWriteOnly 22745
SysUtils.EraNames 23024

SysUtils.EraNames variable 23024
SysUtils.ERangeError 22745
SysUtils.EraYearOffsets 23025
SysUtils.EraYearOffsets variable 23025
SysUtils.ESafecallException 22747
SysUtils.EStackOverflow 22747
SysUtils.EUnderflow 22748
SysUtils.EVariantError 22748
SysUtils.EWin32Error 22749
SysUtils.ExceptClass 23012
SysUtils.ExceptClass type 23012
SysUtils.Exception 22750
SysUtils.Exception.Create 22752
SysUtils.Exception.CreateFmt 22754
SysUtils.Exception.CreateFmtHelp 22755
SysUtils.Exception.CreateHelp 22755
SysUtils.Exception.CreateRes 22756
SysUtils.Exception.CreateResFmt 22757
SysUtils.Exception.CreateResFmtHelp 22758
SysUtils.Exception.CreateResHelp 22759
SysUtils.Exception.HelpContext 22760
SysUtils.Exception.Message 22760
SysUtils.ExceptionErrorMessage 22839
SysUtils.ExceptionErrorMessage function 22839
SysUtils.ExcludeTrailingBackslash 22839
SysUtils.ExcludeTrailingBackslash function 22839
SysUtils.ExcludeTrailingPathDelimiter 22840
SysUtils.ExcludeTrailingPathDelimiter function 22840
SysUtils.ExpandFileName 22840
SysUtils.ExpandFileName function 22840
SysUtils.ExpandFileNameCase 22841
SysUtils.ExpandFileNameCase function 22841
SysUtils.ExpandUNCFileName 22842
SysUtils.ExpandUNCFileName function 22842
SysUtils.ExtractFileDir 22843
SysUtils.ExtractFileDir function 22843
SysUtils.ExtractFileDrive 22843
SysUtils.ExtractFileDrive function 22843
SysUtils.ExtractFileExt 22844
SysUtils.ExtractFileExt function 22844
SysUtils.ExtractFileName 22844
SysUtils.ExtractFileName function 22844
SysUtils.ExtractFilePath 22846
SysUtils.ExtractFilePath function 22846
SysUtils.ExtractRelativePath 22848
SysUtils.ExtractRelativePath function 22848
SysUtils.ExtractShortPathName 22849
SysUtils.ExtractShortPathName function 22849
SysUtils.EZeroDivide 22761
SysUtils.faAnyFile 23041
SysUtils.faAnyFile constant 23041
SysUtils.faArchive 23042
SysUtils.faArchive constant 23042
SysUtils.faDirectory 23042
SysUtils.faDirectory constant 23042
SysUtils.faHidden 23043
SysUtils.faHidden constant 23043
SysUtils.FalseBoolStrs 23025
SysUtils.FalseBoolStrs variable 23025
SysUtils.faReadOnly 23044
SysUtils.faReadOnly constant 23044
SysUtils.faSymLink 23044
SysUtils.faSymLink constant 23044
SysUtils.faSysFile 23045
SysUtils.faSysFile constant 23045
SysUtils.faVolumeID 23045
SysUtils.faVolumeID constant 23045
SysUtils.FileAge 22850
SysUtils.FileAge function 22850
SysUtils.FileClose 22852
SysUtils.FileClose function 22852
SysUtils.FileCreate 22855
SysUtils.FileCreate function 22855
SysUtils.FileDateToDateTime 22858
SysUtils.FileDateToDateTime function 22858
SysUtils.FileExists 22860
SysUtils.FileExists function 22860
SysUtils.FileGetAttr 22863

SysUtils.FileGetAttr function 22863
SysUtils.FileGetDate 22866
SysUtils.FileGetDate function 22866
SysUtils.FileIsReadOnly 22866
SysUtils.FileIsReadOnly function 22866
SysUtils.FileOpen 22867
SysUtils.FileOpen function 22867
SysUtils.FileRead 22870
SysUtils.FileRead function 22870
SysUtils.FileSearch 22872
SysUtils.FileSearch function 22872
SysUtils.FileSeek 22873
SysUtils.FileSeek function 22873
SysUtils.FileSetAttr 22875
SysUtils.FileSetAttr function 22875
SysUtils.FileSetDate 22878
SysUtils.FileSetDate function 22878
SysUtils.FileSetReadOnly 22878
SysUtils.FileSetReadOnly function 22878
SysUtils.FileWrite 22879
SysUtils.FileWrite function 22879
SysUtils.FinalizePackage 22882
SysUtils.FinalizePackage function 22882
SysUtils.FindClose 22882
SysUtils.FindClose function 22882
SysUtils.FindCmdLineSwitch 22884
SysUtils.FindCmdLineSwitch function 22884
SysUtils.FindFirst 22884
SysUtils.FindFirst function 22884
SysUtils.FindNext 22887
SysUtils.FindNext function 22887
SysUtils.FloatToCurr 22889
SysUtils.FloatToCurr function 22889
SysUtils.FloatToDateTime 22889
SysUtils.FloatToDateTime function 22889
SysUtils.FloatToDecimal 22890
SysUtils.FloatToDecimal function 22890
SysUtils.FloatToStr 22890
SysUtils.FloatToStr function 22890
SysUtils.FloatToStrF 22891
SysUtils.FloatToStrF function 22891
SysUtils.FloatToText 22892
SysUtils.FloatToText function 22892
SysUtils.FloatToTextFmt 22893
SysUtils.FloatToTextFmt function 22893
SysUtils.fmOpenRead 23046
SysUtils.fmOpenRead constant 23046
SysUtils.fmOpenReadWrite 23047
SysUtils.fmOpenReadWrite constant 23047
SysUtils.fmOpenWrite 23047
SysUtils.fmOpenWrite constant 23047
SysUtils.fmShareCompat 23048
SysUtils.fmShareCompat constant 23048
SysUtils.fmShareDenyNone 23048
SysUtils.fmShareDenyNone constant 23048
SysUtils.fmShareDenyRead 23049
SysUtils.fmShareDenyRead constant 23049
SysUtils.fmShareDenyWrite 23049
SysUtils.fmShareDenyWrite constant 23049
SysUtils.fmShareExclusive 23049
SysUtils.fmShareExclusive constant 23049
SysUtils.FmtLoadStr 22893
SysUtils.FmtLoadStr function 22893
SysUtils.FmtStr 22894
SysUtils.FmtStr function 22894
SysUtils.ForceDirectories 22895
SysUtils.ForceDirectories function 22895
SysUtils.Format 22897
SysUtils.Format function 22897
SysUtils.FormatBuf 22900
SysUtils.FormatBuf function 22900
SysUtils.FormatCurr 22901
SysUtils.FormatCurr function 22901
SysUtils.FormatDateTime 22902
SysUtils.FormatDateTime function 22902
SysUtils.FormatFloat 22904
SysUtils.FormatFloat function 22904
SysUtils.FreeAndNil 22907

SysUtils.FreeAndNil function 22907
SysUtils.GDAL 22909
SysUtils.GDAL function 22909
SysUtils.GetCurrentDir 22910
SysUtils.GetCurrentDir function 22910
SysUtils.GetDiskFreeSpaceEx 23026
SysUtils.GetDiskFreeSpaceEx variable 23026
SysUtils.GetEnvironmentVariable 22911
SysUtils.GetEnvironmentVariable function 22911
SysUtils.GetFileVersion 22912
SysUtils.GetFileVersion function 22912
SysUtils.GetFormatSettings 22912
SysUtils.GetFormatSettings function 22912
SysUtils.GetLocaleChar 22913
SysUtils.GetLocaleChar function 22913
SysUtils.GetLocaleFormatSettings 22913
SysUtils.GetLocaleFormatSettings function 22913
SysUtils.GetLocaleStr 22913
SysUtils.GetLocaleStr function 22913
SysUtils.GetModuleName 22914
SysUtils.GetModuleName function 22914
SysUtils.GetPackageDescription 22915
SysUtils.GetPackageDescription function 22915
SysUtils.GetPackageInfo 22915
SysUtils.GetPackageInfo function 22915
SysUtils.GetTime 22916
SysUtils.GetTime function 22916
SysUtils.GUIDToString 22916
SysUtils.GUIDToString function 22916
SysUtils.HexDisplayPrefix 23026
SysUtils.HexDisplayPrefix variable 23026
SysUtils.HoursPerDay 23050
SysUtils.HoursPerDay constant 23050
SysUtils.IncAMonth 22917
SysUtils.IncAMonth function 22917
SysUtils.IncludeTrailingBackslash 22918
SysUtils.IncludeTrailingBackslash function 22918
SysUtils.IncludeTrailingPathDelimiter 22918
SysUtils.IncludeTrailingPathDelimiter function 22918
SysUtils.IncMonth 22919
SysUtils.IncMonth function 22919
SysUtils.InitializePackage 22920
SysUtils.InitializePackage function 22920
SysUtils.Int64Rec 22996
SysUtils.Int64Rec record 22996
SysUtils.IntToHex 22920
SysUtils.IntToHex function 22920
SysUtils.IntToStr 22921
SysUtils.IntToStr function 22921
SysUtils.IReadWriteSync 22778
SysUtils.IReadWriteSync.BeginRead 22779
SysUtils.IReadWriteSync.BeginWrite 22779
SysUtils.IReadWriteSync.EndRead 22779
SysUtils.IReadWriteSync.EndWrite 22779
SysUtils.IsAssembly 22922
SysUtils.IsAssembly function 22922
SysUtils.IsDelimiter 22923
SysUtils.IsDelimiter function 22923
SysUtils.IsEqualGUID 22923
SysUtils.IsEqualGUID function 22923
SysUtils.IsLeapYear 22924
SysUtils.IsLeapYear function 22924
SysUtils.IsPathDelimiter 22925
SysUtils.IsPathDelimiter function 22925
SysUtils.IsValidIdent 22925
SysUtils.IsValidIdent function 22925
SysUtils.Languages 22926
SysUtils.Languages function 22926
SysUtils.LastDelimiter 22926
SysUtils.LastDelimiter function 22926
SysUtils.LeadBytes 23026
SysUtils.LeadBytes variable 23026
SysUtils.ListSeparator 23027
SysUtils.ListSeparator variable 23027
SysUtils.LoadPackage 22927
SysUtils.LoadPackage function 22927
SysUtils.LoadStr 22927
SysUtils.LoadStr function 22927

SysUtils.LongDateFormat 23027
SysUtils.LongDateFormat variable 23027
SysUtils.LongDayNames 23027
SysUtils.LongDayNames variable 23027
SysUtils.LongMonthNames 23028
SysUtils.LongMonthNames variable 23028
SysUtils.LongRec 22997
SysUtils.LongRec record 22997
SysUtils.LongTimeFormat 23028
SysUtils.LongTimeFormat variable 23028
SysUtils.LowerCase 22928
SysUtils.LowerCase function 22928
SysUtils.MaxCurrency 23050
SysUtils.MaxCurrency constant 23050
SysUtils.MaxDateTime 23051
SysUtils.MaxDateTime constant 23051
SysUtils.MaxEraCount 23051
SysUtils.MaxEraCount constant 23051
SysUtils.MinCurrency 23051
SysUtils.MinCurrency constant 23051
SysUtils.MinDateTime 23052
SysUtils.MinDateTime constant 23052
SysUtils.MinsPerDay 23052
SysUtils.MinsPerDay constant 23052
SysUtils.MinsPerHour 23053
SysUtils.MinsPerHour constant 23053
SysUtils.MonthDays 23054
SysUtils.MonthDays constant 23054
SysUtils.MSecsPerDay 23054
SysUtils.MSecsPerDay constant 23054
SysUtils.MSecsPerSec 23055
SysUtils.MSecsPerSec constant 23055
SysUtils.MSecsToTimeStamp 22929
SysUtils.MSecsToTimeStamp function 22929
SysUtils.NegCurrFormat 23029
SysUtils.NegCurrFormat variable 23029
SysUtils.NewStr 22929
SysUtils.NewStr function 22929
SysUtils.NextCharIndex 22930
SysUtils.NextCharIndex function 22930
SysUtils.Now 22930
SysUtils.Now function 22930
SysUtils.NullStr 23029
SysUtils.NullStr variable 23029
SysUtils.NullWideStr 23030
SysUtils.NullWideStr variable 23030
SysUtils.OutOfMemoryError 22931
SysUtils.OutOfMemoryError function 22931
SysUtils.PathDelim 23055
SysUtils.PathDelim constant 23055
SysUtils.PathSep 23055
SysUtils.PathSep constant 23055
SysUtils.PByteArray 23013
SysUtils.PByteArray type 23013
SysUtils.PDayTable 23013
SysUtils.PDayTable type 23013
SysUtils.PExceptionRecord 23014
SysUtils.PExceptionRecord type 23014
SysUtils.pfBCB4Produced 23056
SysUtils.pfBCB4Produced constant 23056
SysUtils.pfConsumerBCB 23056
SysUtils.pfConsumerBCB constant 23056
SysUtils.pfConsumerCompat 23057
SysUtils.pfConsumerCompat constant 23057
SysUtils.pfConsumerDelphi 23057
SysUtils.pfConsumerDelphi constant 23057
SysUtils.pfConsumerMask 23058
SysUtils.pfConsumerMask constant 23058
SysUtils.pfDelphi4Produced 23058
SysUtils.pfDelphi4Produced constant 23058
SysUtils.pfDesignOnly 23059
SysUtils.pfDesignOnly constant 23059
SysUtils.pfExeModule 23059
SysUtils.pfExeModule constant 23059
SysUtils.pfIgnoreDupUnits 23060
SysUtils.pfIgnoreDupUnits constant 23060
SysUtils.pfLibraryModule 23060
SysUtils.pfLibraryModule constant 23060

SysUtils.pfModuleTypeMask 23060
SysUtils.pfModuleTypeMask constant 23060
SysUtils.pfNeverBuild 23061
SysUtils.pfNeverBuild constant 23061
SysUtils.pfPackageModule 23061
SysUtils.pfPackageModule constant 23061
SysUtils.pfProducerMask 23062
SysUtils.pfProducerMask constant 23062
SysUtils.pfProducerUndefined 23062
SysUtils.pfProducerUndefined constant 23062
SysUtils.pfRunOnly 23063
SysUtils.pfRunOnly constant 23063
SysUtils.pfV3Produced 23063
SysUtils.pfV3Produced constant 23063
SysUtils.PThreadInfo 23014
SysUtils.PThreadInfo type 23014
SysUtils.PWordArray 23014
SysUtils.PWordArray type 23014
SysUtils.QuotedStr 22931
SysUtils.QuotedStr function 22931
SysUtils.RaiseLastOSError 22932
SysUtils.RaiseLastOSError function 22932
SysUtils.RaiseLastWin32Error 22932
SysUtils.RaiseLastWin32Error function 22932
SysUtils.RCS 22933
SysUtils.RCS function 22933
SysUtils.RemoveDir 22933
SysUtils.RemoveDir function 22933
SysUtils.RenameFile 22934
SysUtils.RenameFile function 22934
SysUtils.ReplaceDate 22936
SysUtils.ReplaceDate function 22936
SysUtils.ReplaceTime 22937
SysUtils.ReplaceTime function 22937
SysUtils.RPR 22938
SysUtils.RPR function 22938
SysUtils.SafeLoadLibrary 22938
SysUtils.SafeLoadLibrary function 22938
SysUtils.SameFileName 22939
SysUtils.SameFileName function 22939
SysUtils.SameStr 22939
SysUtils.SameStr function 22939
SysUtils.SameText 22940
SysUtils.SameText function 22940
SysUtils.SecsPerDay 23064
SysUtils.SecsPerDay constant 23064
SysUtils.SecsPerMin 23064
SysUtils.SecsPerMin constant 23064
SysUtils.SetCurrentDir 22940
SysUtils.SetCurrentDir function 22940
SysUtils.ShortDateFormat 23030
SysUtils.ShortDateFormat variable 23030
SysUtils.ShortDayNames 23031
SysUtils.ShortDayNames variable 23031
SysUtils.ShortMonthNames 23031
SysUtils.ShortMonthNames variable 23031
SysUtils.ShortTimeFormat 23031
SysUtils.ShortTimeFormat variable 23031
SysUtils.ShowException 22941
SysUtils.ShowException function 22941
SysUtils.Sleep 22941
SysUtils.Sleep function 22941
SysUtils.StrAlloc 22942
SysUtils.StrAlloc function 22942
SysUtils.StrBufSize 22942
SysUtils.StrBufSize function 22942
SysUtils.StrByteType 22943
SysUtils.StrByteType function 22943
SysUtils.StrCat 22943
SysUtils.StrCat function 22943
SysUtils.StrCharLength 22944
SysUtils.StrCharLength function 22944
SysUtils.StrComp 22945
SysUtils.StrComp function 22945
SysUtils.StrCopy 22946
SysUtils.StrCopy function 22946
SysUtils.StrDispose 22947
SysUtils.StrDispose function 22947

SysUtils.StrECopy 22948
SysUtils.StrECopy function 22948
SysUtils.StrEnd 22949
SysUtils.StrEnd function 22949
SysUtils.StrFmt 22950
SysUtils.StrFmt function 22950
SysUtils.StrIComp 22951
SysUtils.StrIComp function 22951
SysUtils.StringReplace 22951
SysUtils.StringReplace function 22951
SysUtils.StringToGUID 22952
SysUtils.StringToGUID function 22952
SysUtils.StrLCat 22952
SysUtils.StrLCat function 22952
SysUtils.StrLComp 22953
SysUtils.StrLComp function 22953
SysUtils.StrLCopy 22953
SysUtils.StrLCopy function 22953
SysUtils.StrLen 22954
SysUtils.StrLen function 22954
SysUtils.StrLFmt 22954
SysUtils.StrLFmt function 22954
SysUtils.StrLIComp 22955
SysUtils.StrLIComp function 22955
SysUtils.StrLower 22956
SysUtils.StrLower function 22956
SysUtils.StrMove 22956
SysUtils.StrMove function 22956
SysUtils.StrNew 22957
SysUtils.StrNew function 22957
SysUtils.StrNextChar 22958
SysUtils.StrNextChar function 22958
SysUtils.StrPas 22958
SysUtils.StrPas function 22958
SysUtils.StrPCopy 22958
SysUtils.StrPCopy function 22958
SysUtils.StrPLCopy 22959
SysUtils.StrPLCopy function 22959
SysUtils.StrPos 22959
SysUtils.StrPos function 22959
SysUtils.StrRScan 22960
SysUtils.StrRScan function 22960
SysUtils.StrScan 22960
SysUtils.StrScan function 22960
SysUtils.StrToBool 22961
SysUtils.StrToBool function 22961
SysUtils.StrToBoolDef 22961
SysUtils.StrToBoolDef function 22961
SysUtils.StrToCurr 22962
SysUtils.StrToCurr function 22962
SysUtils.StrToCurrDef 22963
SysUtils.StrToCurrDef function 22963
SysUtils.StrToDate 22963
SysUtils.StrToDate function 22963
SysUtils.StrToDateDef 22965
SysUtils.StrToDateDef function 22965
SysUtils.StrToDateTime 22965
SysUtils.StrToDateTime function 22965
SysUtils.StrToDateTimeDef 22966
SysUtils.StrToDateTimeDef function 22966
SysUtils.StrToFloat 22967
SysUtils.StrToFloat function 22967
SysUtils.StrToFloatDef 22969
SysUtils.StrToFloatDef function 22969
SysUtils.StrToInt 22970
SysUtils.StrToInt function 22970
SysUtils.StrToInt64 22970
SysUtils.StrToInt64 function 22970
SysUtils.StrToInt64Def 22971
SysUtils.StrToInt64Def function 22971
SysUtils.StrToIntDef 22971
SysUtils.StrToIntDef function 22971
SysUtils.StrToTime 22972
SysUtils.StrToTime function 22972
SysUtils.StrToTimeDef 22972
SysUtils.StrToTimeDef function 22972
SysUtils.StrUpper 22973
SysUtils.StrUpper function 22973

SysUtils.Supports 22973
SysUtils.Supports function 22973
SysUtils.SwitchChars 23065
SysUtils.SwitchChars constant 23065
SysUtils.SysErrorMessage 22974
SysUtils.SysErrorMessage function 22974
SysUtils.SysLocale 23032
SysUtils.SysLocale variable 23032
SysUtils.SystemTimeToDateTime 22974
SysUtils.SystemTimeToDateTime function 22974
SysUtils.TByteArray 23015
SysUtils.TByteArray type 23015
SysUtils.TBytes 23015
SysUtils.TBytes type 23015
SysUtils.TDayTable 23015
SysUtils.TDayTable type 23015
SysUtils.TExceptionRecord 22997
SysUtils.TExceptionRecord record 22997
SysUtils.TextToFloat 22975
SysUtils.TextToFloat function 22975
SysUtils.TFileName 23016
SysUtils.TFileName type 23016
SysUtils.TFilenameCaseMatch 22998
SysUtils.TFilenameCaseMatch enumeration 22998
SysUtils.TFloatFormat 22999
SysUtils.TFloatFormat enumeration 22999
SysUtils.TFloatRec 23000
SysUtils.TFloatRec record 23000
SysUtils.TFloatValue 23001
SysUtils.TFloatValue enumeration 23001
SysUtils.TFormatSettings 23001
SysUtils.TFormatSettings record 23001
SysUtils.ThousandSeparator 23032
SysUtils.ThousandSeparator variable 23032
SysUtils.Time 22976
SysUtils.Time function 22976
SysUtils.TimeAMString 23033
SysUtils.TimeAMString variable 23033
SysUtils.TimePMString 23033
SysUtils.TimePMString variable 23033
SysUtils.TimeSeparator 23033
SysUtils.TimeSeparator variable 23033
SysUtils.TimeStampToDateTime 22976
SysUtils.TimeStampToDateTime function 22976
SysUtils.TimeStampToMSecs 22977
SysUtils.TimeStampToMSecs function 22977
SysUtils.TimeToStr 22977
SysUtils.TimeToStr function 22977
SysUtils.TIntegerSet 23016
SysUtils.TIntegerSet type 23016
SysUtils.TLangRec 23002
SysUtils.TLangRec record 23002
SysUtils.TLanguages 22762
SysUtils.TLanguages.Count 22764
SysUtils.TLanguages.Create 22763
SysUtils.TLanguages.Ext 22764
SysUtils.TLanguages.ID 22764
SysUtils.TLanguages.IndexOf 22763
SysUtils.TLanguages.LocaleID 22765
SysUtils.TLanguages.Name 22765
SysUtils.TLanguages.NameFromLCID 22766
SysUtils.TLanguages.NameFromLocaleID 22766
SysUtils.TLocaleOptions 23003
SysUtils.TLocaleOptions enumeration 23003
SysUtils.TMbcsByteType 23004
SysUtils.TMbcsByteType enumeration 23004
SysUtils.TMREWSync 23017
SysUtils.TMREWSync type 23017
SysUtils.TMultiReadExclusiveWriteSynchronizer 22766
SysUtils.TMultiReadExclusiveWriteSynchronizer.BeginRead 22769
SysUtils.TMultiReadExclusiveWriteSynchronizer.BeginWrite 22770
SysUtils.TMultiReadExclusiveWriteSynchronizer.Create 22769
SysUtils.TMultiReadExclusiveWriteSynchronizer.Destroy 22769
SysUtils.TMultiReadExclusiveWriteSynchronizer.EndRead 22770
SysUtils.TMultiReadExclusiveWriteSynchronizer.EndWrite 22771
SysUtils.TMultiReadExclusiveWriteSynchronizer.RevisionLevel

22771
SysUtils.TNameType 23005
SysUtils.TNameType enumeration 23005
SysUtils.TPackageInfoProc 23017
SysUtils.TPackageInfoProc type 23017
SysUtils.TProcedure 23018
SysUtils.TProcedure type 23018
SysUtils.TReplaceFlags 23018
SysUtils.TReplaceFlags type 23018
SysUtils.Trim 22978
SysUtils.Trim function 22978
SysUtils.TrimLeft 22979
SysUtils.TrimLeft function 22979
SysUtils.TrimRight 22979
SysUtils.TrimRight function 22979
SysUtils.TrueBoolStrs 23034
SysUtils.TrueBoolStrs variable 23034
SysUtils.TryEncodeDate 22980
SysUtils.TryEncodeDate function 22980
SysUtils.TryEncodeTime 22980
SysUtils.TryEncodeTime function 22980
SysUtils.TryFloatToCurr 22981
SysUtils.TryFloatToCurr function 22981
SysUtils.TryFloatToDateTime 22982
SysUtils.TryFloatToDateTime function 22982
SysUtils.TryStrToBool 22982
SysUtils.TryStrToBool function 22982
SysUtils.TryStrToCurr 22983
SysUtils.TryStrToCurr function 22983
SysUtils.TryStrToDate 22983
SysUtils.TryStrToDate function 22983
SysUtils.TryStrToDateTime 22984
SysUtils.TryStrToDateTime function 22984
SysUtils.TryStrToFloat 22985
SysUtils.TryStrToFloat function 22985
SysUtils.TryStrToInt 22986
SysUtils.TryStrToInt function 22986
SysUtils.TryStrToInt64 22986
SysUtils.TryStrToInt64 function 22986
SysUtils.TryStrToTime 22987
SysUtils.TryStrToTime function 22987
SysUtils.TSearchRec 23005
SysUtils.TSearchRec record 23005
SysUtils.TSimpleRWSync 22772
SysUtils.TSimpleRWSync.BeginRead 22774
SysUtils.TSimpleRWSync.BeginWrite 22775
SysUtils.TSimpleRWSync.Create 22773
SysUtils.TSimpleRWSync.Destroy 22774
SysUtils.TSimpleRWSync.EndRead 22775
SysUtils.TSimpleRWSync.EndWrite 22775
SysUtils.TSysCharSet 23019
SysUtils.TSysCharSet type 23019
SysUtils.TSysLocale 23008
SysUtils.TSysLocale record 23008
SysUtils.TTerminateProc 23019
SysUtils.TTerminateProc type 23019
SysUtils.TThreadInfo 23009
SysUtils.TThreadInfo record 23009
SysUtils.TThreadLocalCounter 22775
SysUtils.TThreadLocalCounter.Close 22776
SysUtils.TThreadLocalCounter.Delete 22777
SysUtils.TThreadLocalCounter.Destroy 22776
SysUtils.TThreadLocalCounter.Open 22777
SysUtils.TTimeStamp 23009
SysUtils.TTimeStamp record 23009
SysUtils.TValidatePackageProc 23019
SysUtils.TValidatePackageProc type 23019
SysUtils.TwoDigitYearCenturyWindow 23034
SysUtils.TwoDigitYearCenturyWindow variable 23034
SysUtils.TWordArray 23020
SysUtils.TWordArray type 23020
SysUtils.ufImplicitUnit 23065
SysUtils.ufImplicitUnit constant 23065
SysUtils.ufMainUnit 23066
SysUtils.ufMainUnit constant 23066
SysUtils.ufOrgWeakUnit 23066
SysUtils.ufOrgWeakUnit constant 23066
SysUtils.ufPackageUnit 23067

SysUtils.ufPackageUnit constant 23067
SysUtils.ufWeakPackageUnit 23067
SysUtils.ufWeakPackageUnit constant 23067
SysUtils.ufWeakUnit 23068
SysUtils.ufWeakUnit constant 23068
SysUtils.UnixDateDelta 23068
SysUtils.UnixDateDelta constant 23068
SysUtils.UnloadPackage 22987
SysUtils.UnloadPackage function 22987
SysUtils.UpperCase 22988
SysUtils.UpperCase function 22988
SysUtils.WideCompareStr 22989
SysUtils.WideCompareStr function 22989
SysUtils.WideCompareText 22990
SysUtils.WideCompareText function 22990
SysUtils.WideFmtStr 22990
SysUtils.WideFmtStr function 22990
SysUtils.WideFormat 22991
SysUtils.WideFormat function 22991
SysUtils.WideFormatBuf 22991
SysUtils.WideFormatBuf function 22991
SysUtils.WideLowerCase 22992
SysUtils.WideLowerCase function 22992
SysUtils.WideSameStr 22992
SysUtils.WideSameStr function 22992
SysUtils.WideSameText 22993
SysUtils.WideSameText function 22993
SysUtils.WideUpperCase 22993
SysUtils.WideUpperCase function 22993
SysUtils.Win32BuildNumber 23035
SysUtils.Win32BuildNumber variable 23035
SysUtils.Win32Check 22994
SysUtils.Win32Check function 22994
SysUtils.Win32CSDVersion 23035
SysUtils.Win32CSDVersion variable 23035
SysUtils.Win32MajorVersion 23036
SysUtils.Win32MajorVersion variable 23036
SysUtils.Win32MinorVersion 23036
SysUtils.Win32MinorVersion variable 23036
SysUtils.Win32Platform 23037
SysUtils.Win32Platform variable 23037
SysUtils.WordRec 23010
SysUtils.WordRec record 23010
SysUtils.WrapText 22994
SysUtils.WrapText function 22994

T

taAdd enumeration member 5439
taAddFirst enumeration member 5439
taAlignBottom enumeration member 3280
taAlignTop enumeration member 3280
TablesColumns class 2190
- about TablesColumns class 2190
- CatalogName 2191
- SchemaName 2191
- TableName 2192
- TablesColumns fields 2191
- TablesColumns Members 2191
- TableType 2192

TabNotBk 23069
TabNotBk namespace 23069
- Classes 23070
- Constants 23132
- Types 23131

TabNotBk.CM_TABFONTCHANGED 23132
TabNotBk.CM_TABFONTCHANGED constant 23132
TabNotBk.TPageChangeEvent 23131
TabNotBk.TPageChangeEvent type 23131
TabNotBk.TTabbedNotebook 23070
TabNotBk.TTabbedNotebook.ActivePage 23089
TabNotBk.TTabbedNotebook.Align 23089
TabNotBk.TTabbedNotebook.AlignControls 23082
TabNotBk.TTabbedNotebook.Anchors 23093
TabNotBk.TTabbedNotebook.Change 23083
TabNotBk.TTabbedNotebook.Click 23083
TabNotBk.TTabbedNotebook.CMTabFontChanged 23084
TabNotBk.TTabbedNotebook.Constraints 23093
TabNotBk.TTabbedNotebook.Create 23081

wwww

TabNotBk.TTabbedNotebook.CreateHandle 23084
TabNotBk.TTabbedNotebook.CreateParams 23084
TabNotBk.TTabbedNotebook.Destroy 23082
TabNotBk.TTabbedNotebook.Enabled 23094
TabNotBk.TTabbedNotebook.Font 23094
TabNotBk.TTabbedNotebook.GetChildOwner 23085
TabNotBk.TTabbedNotebook.GetChildren 23085
TabNotBk.TTabbedNotebook.GetIndexForPage 23085
TabNotBk.TTabbedNotebook.KeyDown 23086
TabNotBk.TTabbedNotebook.Loaded 23086
TabNotBk.TTabbedNotebook.OnChange 23106
TabNotBk.TTabbedNotebook.OnClick 23107
TabNotBk.TTabbedNotebook.OnContextPopup 23107
TabNotBk.TTabbedNotebook.OnEnter 23108
TabNotBk.TTabbedNotebook.OnExit 23110
TabNotBk.TTabbedNotebook.PageIndex 23095
TabNotBk.TTabbedNotebook.Pages 23096
TabNotBk.TTabbedNotebook.ParentShowHint 23096
TabNotBk.TTabbedNotebook.PopupMenu 23097
TabNotBk.TTabbedNotebook.ReadState 23087
TabNotBk.TTabbedNotebook.SetPageIndex 23087
TabNotBk.TTabbedNotebook.ShowControl 23088
TabNotBk.TTabbedNotebook.ShowHint 23101
TabNotBk.TTabbedNotebook.TabFont 23103
TabNotBk.TTabbedNotebook.TabFontChanged 23088
TabNotBk.TTabbedNotebook.TabOrder 23103
TabNotBk.TTabbedNotebook.TabsPerRow 23104
TabNotBk.TTabbedNotebook.TabStop 23104
TabNotBk.TTabbedNotebook.TopFont 23104
TabNotBk.TTabbedNotebook.Visible 23105
TabNotBk.TTabPage 23112
TabNotBk.TTabPage.Caption 23124
TabNotBk.TTabPage.Create 23122
TabNotBk.TTabPage.Enabled 23126
TabNotBk.TTabPage.Height 23127
TabNotBk.TTabPage.ReadState 23123
TabNotBk.TTabPage.TabOrder 23128
TabNotBk.TTabPage.Visible 23129
TabNotBk.TTabPage.Width 23130
Tabs 23133
Tabs namespace 23133
Classes 23134
Structs, Records, Enums 23229
Types 23232
Tabs.TDrawTabEvent 23232
Tabs.TDrawTabEvent type 23232
Tabs.TEdgePart 23229
Tabs.TEdgePart enumeration 23229
Tabs.TEdgeType 23229
Tabs.TEdgeType enumeration 23229
Tabs.TMeasureTabEvent 23232
Tabs.TMeasureTabEvent type 23232
Tabs.TScrollBtn 23230
Tabs.TScrollBtn enumeration 23230
Tabs.TScroller 23134
Tabs.TScroller.Change 23146
Tabs.TScroller.Create 23145
Tabs.TScroller.Destroy 23145
Tabs.TScroller.Max 23147
Tabs.TScroller.Min 23147
Tabs.TScroller.OnClick 23148
Tabs.TScroller.Paint 23146
Tabs.TScroller.Position 23148
Tabs.TScroller.ScrollOrientation 23148
Tabs.TScrollOrientation 23230
Tabs.TScrollOrientation enumeration 23230
Tabs.TTabChangeEvent 23233
Tabs.TTabChangeEvent type 23233
Tabs.TTabList 23149
Tabs.TTabList.Add 23153
Tabs.TTabList.AddStrings 23155
Tabs.TTabList.Clear 23157
Tabs.TTabList.Create 23152
Tabs.TTabList.Delete 23157
Tabs.TTabList.Insert 23158
Tabs.TTabList.Put 23158
Tabs.TTabSet 23159
Tabs.TTabSet.Align 23178

Tabs.TTabSet.Anchors 23182
Tabs.TTabSet.AutoScroll 23182
Tabs.TTabSet.BackgroundColor 23183
Tabs.TTabSet.CanChange 23171
Tabs.TTabSet.Canvas 23183
Tabs.TTabSet.CMHintShow 23171
Tabs.TTabSet.Constraints 23185
Tabs.TTabSet.Create 23170
Tabs.TTabSet.CreateParams 23172
Tabs.TTabSet.Destroy 23170
Tabs.TTabSet.DitherBackground 23186
Tabs.TTabSet.DragCursor 23186
Tabs.TTabSet.DragKind 23186
Tabs.TTabSet.DragMode 23187
Tabs.TTabSet.DrawTab 23172
Tabs.TTabSet.Enabled 23188
Tabs.TTabSet.EndMargin 23189
Tabs.TTabSet.FirstIndex 23189
Tabs.TTabSet.Font 23190
Tabs.TTabSet.GetChildren 23172
Tabs.TTabSet.GetImageIndex 23173
Tabs.TTabSet.Images 23191
Tabs.TTabSet.ItemAtPos 23173
Tabs.TTabSet.ItemRect 23173
Tabs.TTabSet.ItemWidth 23174
Tabs.TTabSet.MeasureTab 23174
Tabs.TTabSet.MinClientRect 23174
Tabs.TTabSet.MouseDown 23175
Tabs.TTabSet.Notification 23176
Tabs.TTabSet.OnAlignInsertBefore 23204
Tabs.TTabSet.OnAlignPosition 23204
Tabs.TTabSet.OnChange 23205
Tabs.TTabSet.OnClick 23205
Tabs.TTabSet.OnDragDrop 23207
Tabs.TTabSet.OnDragOver 23210
Tabs.TTabSet.OnDrawTab 23212
Tabs.TTabSet.OnEndDock 23213
Tabs.TTabSet.OnEndDrag 23213
Tabs.TTabSet.OnEnter 23214
Tabs.TTabSet.OnExit 23216
Tabs.TTabSet.OnGetImageIndex 23218
Tabs.TTabSet.OnMeasureTab 23218
Tabs.TTabSet.OnMouseActivate 23219
Tabs.TTabSet.OnMouseDown 23220
Tabs.TTabSet.OnMouseEnter 23222
Tabs.TTabSet.OnMouseLeave 23222
Tabs.TTabSet.OnMouseMove 23222
Tabs.TTabSet.OnMouseUp 23225
Tabs.TTabSet.OnStartDock 23227
Tabs.TTabSet.OnStartDrag 23228
Tabs.TTabSet.Paint 23176
Tabs.TTabSet.ParentBackground 23191
Tabs.TTabSet.ParentShowHint 23191
Tabs.TTabSet.PopupMenu 23192
Tabs.TTabSet.Scroller 23196
Tabs.TTabSet.SelectedColor 23197
Tabs.TTabSet.SelectNext 23177
Tabs.TTabSet.SetTabIndex 23177
Tabs.TTabSet.ShowHint 23197
Tabs.TTabSet.ShrinkToFit 23198
Tabs.TTabSet.SoftTop 23199
Tabs.TTabSet.StartMargin 23199
Tabs.TTabSet.Style 23199
Tabs.TTabSet.TabHeight 23200
Tabs.TTabSet.TabIndex 23200
Tabs.TTabSet.TabPosition 23201
Tabs.TTabSet.Tabs 23201
Tabs.TTabSet.UnselectedColor 23201
Tabs.TTabSet.Visible 23201
Tabs.TTabSet.VisibleTabs 23203
Tabs.TTabSetDrawTabEvent 23233
Tabs.TTabSetDrawTabEvent type 23233
Tabs.TTabSetTabStyle 23231
Tabs.TTabSetTabStyle enumeration 23231
Tabs.TTabStyle 23234
Tabs.TTabStyle type 23234
TAbstractAdapterRequest class 24639
about TAbstractAdapterRequest class 24639

Create 24641
Request 24642
TAbstractAdapterRequest Members 24640
TAbstractAdapterRequest properties 24641
TAbstractAdapterResponse class 24642
about TAbstractAdapterResponse class 24642
Create 24644
Response 24644
TAbstractAdapterResponse Members 24642
TAbstractAdapterResponse properties 24644
TAbstractContentParser class 15039
about TAbstractContentParser class 15039
CanParse 15041
Create 15041
GetContentFields 15042
GetFiles 15042
TAbstractContentParser Members 15040
TAbstractContentParser methods 15041
TAbstractContentParser properties 15043
WebRequest 15043
TAbstractControlRequirements class 25029
about TAbstractControlRequirements class 25029
DefineFunction 25030
DefineVariable 25030
IsFunctionDefined 25031
IsVariableDefined 25031
RequiresMultipartForm 25032
TAbstractControlRequirements Members 25029
TAbstractControlRequirements methods 25030
TAbstractDesigntimeWarnings class 19631
about TAbstractDesigntimeWarnings class 19631
AddObject 19632
AddString 19632
TAbstractDesigntimeWarnings Members 19631
TAbstractDesigntimeWarnings methods 19632
TAbstractFieldValuesAdapter class 23701
about TAbstractFieldValuesAdapter class 23701
GetField 23708
GetValueFieldClass 23709
GetValueIndex 23709
ImplCanAddFieldClass 23710
ImplEndIterator 23710
ImplGetField 23711
ImplGetFieldsList 23711
ImplGetValueIndex 23711
ImplNextIteration 23712
ImplStartIterator 23712
TAbstractFieldValuesAdapter Members 23702
TAbstractFieldValuesAdapter methods 23708
TAbstractFormRequirements class 25032
about TAbstractFormRequirements class 25032
DefineFunction 25033
DefineVariable 25033
IsFunctionDefined 25033
IsVariableDefined 25034
TAbstractFormRequirements Members 25032
TAbstractFormRequirements methods 25033
TAbstractInetFileType class 19633
about TAbstractInetFileType class 19633
FileExt 19635
FileExtCount 19635
GetExtCount 19634
GetFileExt 19634
GetTypeID 19635
TAbstractInetFileType Members 19633
TAbstractInetFileType methods 19634
TAbstractInetFileType properties 19635
TypeID 19635
TAbstractNamedVariants class 24808
about TAbstractNamedVariants class 24808
Add 24812
Append 24812
Assign 24813
BeginUpdate 24813
Capacity 24821
Clear 24813
CompareStrings 24814
Count 24822

DefineProperties 24814
Delete 24815
Destroy 24810
EndUpdate 24815
Equals 24816
Error 24816
Exchange 24817
Get 24817
GetCapacity 24817
GetCount 24818
IndexOfName 24818
Insert 24819
Move 24819
Names 24822
Put 24820
SetCapacity 24820
SetUpdateState 24821
TAbstractNamedVariants Members 24809
TAbstractNamedVariants methods 24811
TAbstractNamedVariants properties 24821
UpdateCount 24823
Values 24823
Variants 24824
TAbstractScriptEngine class 25466
about TAbstractScriptEngine class 25466
AddObjectFactory 25468
Create 25467
CreateComponentStub 25468
Destroy 25467
ListLanguageNames 25469
Name 25469
ScriptObjectFactories 25470
ScriptProducerClass 25470
SupportsLanguage 25469
TAbstractScriptEngine Members 25466
TAbstractScriptEngine methods 25468
TAbstractScriptEngine properties 25469
TAbstractScriptEnginesList class 15258
about TAbstractScriptEnginesList class 15258
Create 15259
FindScriptProducerClass 15260
TAbstractScriptEnginesList Members 15258
TAbstractScriptEnginesList methods 15260
TAbstractScriptError class 15260
about TAbstractScriptError class 15260
CharPos 15263
Description 15263
FileName 15264
GetCharPos 15262
GetDescription 15262
GetFileName 15262
GetLine 15262
GetSourceLine 15263
Line 15264
SetDescription 15263
SourceLine 15264
TAbstractScriptError Members 15260
TAbstractScriptError methods 15261
TAbstractScriptError properties 15263
TAbstractScriptErrors class 15264
about TAbstractScriptErrors class 15264
Add 15266
Count 15267
Errors 15267
GetError 15266
GetErrorCount 15266
TAbstractScriptErrors Members 15265
TAbstractScriptErrors methods 15266
TAbstractScriptErrors properties 15267
TAbstractScriptObjectFactories class 25470
about TAbstractScriptObjectFactories class 25470
Count 25472
Factories 25473
GetCount 25472
GetFactory 25472
GetObjectList 25472
ObjectList 25473
TAbstractScriptObjectFactories Members 25470

TAbstractScriptObjectFactories methods 25471
TAbstractScriptObjectFactories properties 25472
TAbstractScriptObjectFactory class 25473
about TAbstractScriptObjectFactory class 25473
AddGlobalObjects 25474
CreateComponentObject 25474
CreateGlobalObject 25475
CreateProducerObject 25475
GetScriptEngine 25475
TAbstractScriptObjectFactory Members 25473
TAbstractScriptObjectFactory methods 25474
TAbstractScriptProducer class 15267
about TAbstractScriptProducer class 15267
Create 15269
TAbstractScriptProducer Members 15268
TAbstractSessionIDGen class 19012
about TAbstractSessionIDGen class 19012
NextID 19013
TAbstractSessionIDGen Members 19013
TAbstractSessionIDGen methods 19013
TAbstractSessionStatusChange class 19013
about TAbstractSessionStatusChange class 19013
SessionStarted 19014
SessionTerminated 19014
SessionTimeout 19015
TAbstractSessionStatusChange Members 19014
TAbstractSessionStatusChange methods 19014
TAbstractSocket class 18740
about TAbstractSocket class 18740
Active 18750
Address 18750
Close 18744
DoActivate 18744
Error 18745
Event 18746
Host 18751
InitSocket 18746
Loaded 18747
Open 18747
Port 18751
Service 18752
SetActive 18748
SetAddress 18748
SetHost 18748
SetPort 18749
SetService 18749
TAbstractSocket Members 18741
TAbstractSocket methods 18744
TAbstractSocket properties 18749
TAbstractWebContext class 24644
about TAbstractWebContext class 24644
AdapterRequest 24655
AdapterResponse 24656
Create 24646
Destroy 24647
DispatchedPageName 24656
EndUser 24657
FindApplicationModule 24648
FindDefaultPageInfo 24648
FindModuleClass 24649
FindModuleName 24649
FindPage 24650
FindPageInfo 24650
FindVariableContainer 24651
GetAdapterRequest 24651
GetAdapterResponse 24652
GetDispatchedPageName 24652
GetEndUser 24652
GetPageCount 24652
GetPageInfo 24653
GetRequestedPageName 24653
GetSession 24653
GetWebModules 24653
GetWebRequest 24654
GetWebResponse 24654
PageCount 24657
Pages 24658
Request 24658

RequestedPageName 24658
Response 24659
Session 24659
SetAdapterRequest 24654
SetAdapterResponse 24654
SetDispatchedPageName 24655
SetRequestedPageName 24655
TAbstractWebContext Members 24645
TAbstractWebContext methods 24647
TAbstractWebContext properties 24655
WebModules 24660
TAbstractWebModuleFactory class 15043
about TAbstractWebModuleFactory class 15043
CacheMode 15048
ComponentClass 15049
CreateMode 15049
GetCacheMode 15045
GetComponentClass 15045
GetCreateMode 15046
GetIsAppModule 15046
GetModule 15046
GetModuleName 15047
GetWebPageInfo 15047
IsAppModule 15050
ModuleName 15050
PreventDestruction 15048
TAbstractWebModuleFactory Members 15044
TAbstractWebModuleFactory methods 15045
TAbstractWebModuleFactory properties 15048
WebPageInfo 15050
TAbstractWebModuleList class 15051
about TAbstractWebModuleList class 15051
AddModuleClass 15053
AddModuleName 15054
Factory 15057
FactoryCount 15058
FindModuleClass 15054
FindModuleName 15055
GetFactory 15055
GetFactoryCount 15056
GetItem 15056
GetItemCount 15056
GetOnModuleAdded 15056
ItemCount 15058
Items 15058
OnModuleAdded 15059
SetOnModuleAdded 15057
TAbstractWebModuleList events 15059
TAbstractWebModuleList Members 15052
TAbstractWebModuleList methods 15053
TAbstractWebModuleList properties 15057
TAbstractWebPageInfo class 15060
about TAbstractWebPageInfo class 15060
Factory 15066
GetFactory 15062
GetIsLoginRequired 15062
GetIsPublished 15062
GetPageDescription 15063
GetPageFile 15063
GetPageHREF 15064
GetPageName 15064
GetPageTitle 15064
GetViewAccess 15065
IsLoginRequired 15066
IsPublished 15067
PageDescription 15067
PageFile 15068
PageHREF 15068
PageName 15068
PageTitle 15069
SetFactory 15065
TAbstractWebPageInfo Members 15061
TAbstractWebPageInfo methods 15061
TAbstractWebPageInfo properties 15065
ViewAccess 15069
TAbstractWebPageModuleFactory class 15070
about TAbstractWebPageModuleFactory class 15070
Create 15071

Destroy 15071
GetWebPageInfo 15072
TAbstractWebPageModuleFactory Members 15070
TAbstractWebPageModuleFactory methods 15072
TAbstractWebRequestFile class 15073
about TAbstractWebRequestFile class 15073
ContentType 15076
FieldName 15077
FileName 15077
GetContentType 15075
GetFieldName 15075
GetFileName 15075
GetStream 15076
Stream 15077
TAbstractWebRequestFile Members 15073
TAbstractWebRequestFile methods 15074
TAbstractWebRequestFile properties 15076
TAbstractWebRequestFiles class 15078
about TAbstractWebRequestFiles class 15078
Count 15081
GetCount 15080
GetItem 15080
Items 15081
TAbstractWebRequestFiles Members 15079
TAbstractWebRequestFiles methods 15079
TAbstractWebRequestFiles properties 15080
TAbstractWebSession class 15082
about TAbstractWebSession class 15082
GetSessionID 15084
GetTimeoutMinutes 15084
GetValue 15084
SessionID 15087
SetTimeoutMinutes 15085
SetValue 15085
TAbstractWebSession Members 15082
TAbstractWebSession methods 15083
TAbstractWebSession properties 15087
Terminate 15086
TimeoutMinutes 15087
UpdateResponse 15086
Values 15088
taCenter enumeration member 3269
TAction class 249
about TAction class 249
AutoCheck 255
Caption 255
Checked 256
Create 254
Enabled 257
GroupIndex 257
HelpContext 258
HelpKeyword 258
HelpType 259
Hint 259
ImageIndex 259
OnExecute 261
OnHint 263
OnUpdate 263
SecondaryShortCuts 260
ShortCut 260
TAction events 261
TAction Members 250
TAction properties 254
Visible 261
TActionBarItem class 346
about TActionBarItem class 346
ActionBar 351
ActionBars 351
AutoSize 351
ColorChanged 349
Create 349
GetDisplayName 350
GlyphLayout 352
Refresh 350
SetVisible 350
TActionBarItem Members 347
TActionBarItem methods 349
TActionBarItem properties 350

TActionBars class 352
about TActionBars class 352
ActionBars 357
Add 355
Create 355
Customizable 357
DoShowHint 355
HintShortCuts 357
Loaded 356
Notify 356
SessionCount 358
ShowHints 358
TActionBars Members 353
TActionBars methods 355
TActionBars properties 356
TActionBarStyle class 358
about TActionBarStyle class 358
GetColorMapClass 359
GetControlClass 359
GetStyleName 360
TActionBarStyle Members 359
TActionBarStyle methods 359
TActionBarStyleEx class 583
about TActionBarStyleEx class 583
GetAddRemoveItemClass 584
GetPopupClass 584
GetScrollBtnClass 585
TActionBarStyleEx Members 584
TActionBarStyleEx methods 584
TActionBarStyleList class 360
about TActionBarStyleList class 360
Style 363
TActionBarStyleList Members 360
TActionBarStyleList properties 363
TActionClient class 364
about TActionClient class 364
Accessible 370
ActionBar 370
Assign 367
Background 370
BackgroundLayout 371
ChangesAllowed 371
ChildActionBar 372
ClearActionBar 367
Color 372
ColorChanged 367
ContextItems 372
Create 366
Destroy 366
FreeChildren 368
HasBackground 368
HasItems 373
HasVisibleItems 368
IsChildItem 368
Items 373
OwningCollection 373
Refresh 369
SetChangesAllowed 369
SetColor 369
SetVisible 369
TActionClient Members 364
TActionClient methods 366
TActionClient properties 369
Tag 374
Visible 374
TActionClientItem class 374
about TActionClientItem class 374
Action 382
ActionChange 378
ActionClients 383
ActionLink 383
Assign 378
Caption 384
CheckUnused 384
ColorChanged 379
Control 384
Create 377
Destroy 377

GetActionLinkClass 379
GetDisplayName 379
HasGlyph 379
HelpContext 385
ImageIndex 385
InitiateAction 380
LastSession 385
MergeIndex 386
ParentItem 386
Refresh 380
ResetUsageData 380
Separator 386
SetChangesAllowed 380
SetCollection 381
SetIndex 381
SetVisible 381
ShortCut 387
ShortCutText 387
ShowCaption 387
ShowGlyph 388
ShowShortCut 388
TActionClientItem Members 375
TActionClientItem methods 378
TActionClientItem properties 382
Unused 382
UsageCount 388
TActionClientLink class 389
about TActionClientLink class 389
AssignClient 392
DoShowHint 392
FClient 391
IsCaptionLinked 392
IsEnabledLinked 393
IsHelpContextLinked 393
IsHintLinked 394
IsImageIndexLinked 394
IsOnExecuteLinked 395
IsShortCutLinked 395
IsVisibleLinked 396
SetCaption 396
SetChecked 397
SetEnabled 397
SetHelpContext 398
SetHint 398
SetImageIndex 399
SetOnExecute 399
SetShortCut 399
SetVisible 400
TActionClientLink fields 391
TActionClientLink Members 389
TActionClientLink methods 391
TActionClients class 400
about TActionClients class 400
ActionClients 406
ActionManager 406
Add 404
Assign 404
AutoHotKeys 407
CaptionOptions 407
Create 403
Customizable 407
DoHideUnused 404
DoShowCaptions 404
HideUnused 408
Notify 405
SmallIcons 408
TActionClients Members 401
TActionClients methods 403
TActionClients properties 406
ToggleSmallIcons 405
VisibleCount 405
TActionClientsCollection class 408
about TActionClientsCollection class 408
ActionClients 413
ActionManager 414
AfterConstruction 411
AutoHotKeys 414
Customizable 415

GetActionManager 412
InternalRethinkHotkeys 412
IterateClients 412
ParentItem 415
TActionClientsCollection Members 409
TActionClientsCollection methods 411
TActionClientsCollection properties 413
TActionDragBaseClass class 415
about TActionDragBaseClass class 415
ActionManager 417
TActionDragBaseClass Members 416
TActionDragBaseClass properties 417
TActionDragObject class 418
about TActionDragObject class 418
ActionCount 422
Actions 422
AddAction 421
AddActionClass 421
Create 419
Destroy 420
FActionClasses 421
TActionDragObject fields 420
TActionDragObject Members 418
TActionDragObject methods 421
TActionDragObject properties 421
TActionFieldValueImpl class 864
about TActionFieldValueImpl class 864
Create 865
Destroy 866
GetFieldName 866
GetFile 867
GetFileCount 867
GetValue 867
GetValueCount 868
TActionFieldValueImpl Members 864
TActionFieldValueImpl methods 866
TActionFieldValuesImpl class 868
about TActionFieldValuesImpl class 868
AddValue 870
Create 869
Destroy 870
FieldCount 872
FieldNames 873
GetFieldName 871
GetRecordKeyCount 871
GetRecordKeyName 871
GetRecordKeyValue 871
IndexOfField 872
OrigValueOfField 872
OrigValues 873
TActionFieldValuesImpl Members 868
TActionFieldValuesImpl methods 870
TActionFieldValuesImpl properties 872
ValueOfField 872
Values 873
TActionItemDragObject class 422
about TActionItemDragObject class 422
ClientItem 425
Finished 424
TActionItemDragObject Members 422
TActionItemDragObject methods 424
TActionItemDragObject properties 424
TActionLink class 265, 1373
about TActionLink class 265, 1373
Create 1375
GetAdapterItemRequestParams 1375
GetComponent 1376
GetFailurePage 1376
GetSuccessPage 1376
IsCaptionLinked 268
IsCheckedLinked 269
IsEnabledLinked 269
IsGroupIndexLinked 270
IsHelpContextLinked 270
IsHelpLinked 271
IsHintLinked 271
IsImageIndexLinked 272
IsShortCutLinked 272

gggggg

IsVisibleLinked 273
SetAutoCheck 274
SetCaption 274
SetChecked 274
SetEnabled 275
SetGroupIndex 275
SetHelpContext 276
SetHelpKeyword 276
SetHelpType 277
SetHint 278
SetImageIndex 278
SetShortCut 279
SetVisible 279
TActionLink Members 266, 1374
TActionLink methods 267, 1375
TActionList class 280
about TActionList class 280
Images 284
OnChange 285
OnExecute 285
OnStateChange 286
OnUpdate 286
State 284
TActionList events 285
TActionList Members 281
TActionList properties 284
TActionListCollection class 425
about TActionListCollection class 425
ActionManager 427
ListItems 428
TActionListCollection Members 425
TActionListCollection properties 427
TActionListEnumerator class 287
about TActionListEnumerator class 287
Create 287
Current 288
GetCurrent 288
MoveNext 288
TActionListEnumerator Members 287
TActionListEnumerator methods 287
TActionListEnumerator properties 288
TActionListItem class 428
about TActionListItem class 428
ActionList 431
Assign 430
Caption 432
GetDisplayName 430
Owner 431
TActionListItem Members 429
TActionListItem methods 430
TActionListItem properties 431
TActionMainMenuBar class 585
about TActionMainMenuBar class 585
ActionManager 601
Align 601
AllowHiding 605
Anchors 605
AnimateDuration 605
AnimationStyle 606
AutoFocus 606
BiDiMode 606
Canvas 607
Caption 607
Color 610
ColorMap 610
Constraints 611
DragCursor 612
DragKind 612
DragMode 612
EdgeBorders 614
EdgeInner 614
EdgeOuter 615
Enabled 615
ExpandDelay 616
Font 616
HorzMargin 617
OnAlignInsertBefore 625
OnAlignPosition 626

OnCanResize 627
OnClick 627
OnConstrainedResize 629
OnContextPopup 629
OnControlCreated 630
OnDblClick 631
OnDockDrop 631
OnDockOver 633
OnDragDrop 635
OnDragOver 637
OnEndDock 640
OnEndDrag 640
OnEnter 641
OnEnterMenuLoop 643
OnExit 644
OnExitMenuLoop 645
OnGetControlClass 645
OnGetPopupClass 646
OnGetSiteInfo 646
OnMouseActivate 648
OnMouseDown 649
OnMouseMove 651
OnMouseUp 654
OnPaint 656
OnPopup 656
OnResize 657
OnStartDock 657
OnStartDrag 658
OnUnDock 658
ParentBackground 617
ParentBiDiMode 617
ParentColor 618
ParentFont 618
ParentShowHint 619
PersistentHotKeys 619
Shadows 620
ShowHint 620
Spacing 622
TActionMainMenuBar events 624
TActionMainMenuBar Members 586
TActionMainMenuBar properties 600
UseSystemFont 622
VertMargin 622
Visible 623
WindowMenu 624
TActionManager class 432
about TActionManager class 432
ActionBars 438
FileName 438
Images 438
LinkedActionLists 439
OnChange 441
OnExecute 442
OnGetBarItemClass 442
OnGetBarsClass 442
OnGetClientItemClass 443
OnGetClientsClass 443
OnStateChange 443
OnStyleChanged 444
OnUpdate 444
PrioritySchedule 439
State 440
Style 440
TActionManager events 441
TActionManager Members 433
TActionManager properties 437
TActionToolBar class 38
about TActionToolBar class 38
ActionManager 52
Align 53
AllowHiding 56
Anchors 57
BiDiMode 57
Canvas 58
Caption 58
Color 61
ColorMap 61
Constraints 62

Cursor 63
DragCursor 65
DragKind 66
DragMode 66
EdgeBorders 68
EdgeInner 68
EdgeOuter 68
Enabled 69
Font 69
HorzMargin 70
HorzSeparator 71
OnAlignInsertBefore 83
OnAlignPosition 84
OnCanResize 84
OnClick 85
OnConstrainedResize 87
OnContextPopup 87
OnControlCreated 88
OnDblClick 88
OnDockDrop 89
OnDockOver 91
OnDragDrop 93
OnDragOver 95
OnEndDock 97
OnEndDrag 98
OnEnter 99
OnExit 101
OnGetControlClass 103
OnGetSiteInfo 103
OnMouseActivate 105
OnMouseDown 106
OnMouseMove 109
OnMouseUp 111
OnPaint 113
OnResize 113
OnStartDock 114
OnStartDrag 115
OnUnDock 115
Orientation 71
ParentBackground 71
ParentBiDiMode 72
ParentColor 72
ParentFont 73
ParentShowHint 73
PersistentHotKeys 74
PopupMenu 74
ShowHint 79
Spacing 80
TActionToolBar events 82
TActionToolBar Members 39
TActionToolBar properties 52
VertMargin 81
Visible 81
TActiveForm class 1489
about TActiveForm class 1489
ActiveFormControl 1506
DefinePropertyPages 1505
DoDestroy 1505
EventSinkChanged 1506
Initialize 1506
TActiveForm Members 1490
TActiveForm methods 1505
TActiveForm properties 1506
TActiveFormControl class 1507
about TActiveFormControl class 1507
DefinePropertyPages 1512
EventSinkChanged 1512
FreeOnRelease 1512
InitializeControl 1513
Invoke 1513
ObjQueryInterface 1513
TActiveFormControl Members 1508
TActiveFormControl methods 1512
TActiveFormFactory class 1514
about TActiveFormFactory class 1514
GetIntfEntry 1516
TActiveFormFactory Members 1514
TActiveFormFactory methods 1516

TActiveXControl class 1517
about TActiveXControl class 1517
Advise 1525
ClientSite 1553
Close 1525
ConnectionPoints 1553
ContextSensitiveHelp 1525
Control 1553
DAdvise 1525
DefinePropertyPages 1526
Destroy 1522
DoVerb 1526
Draw 1526
DUnadvise 1526
EnableModeless 1527
EnumAdvise 1527
EnumDAdvise 1527
EnumFormatEtc 1527
EnumVerbs 1527
Freeze 1528
FreezeEvents 1528
Get_BackColor 1528
Get_Caption 1528
Get_Enabled 1529
Get_Font 1529
Get_ForeColor 1529
Get_HWnd 1529
Get_TabStop 1529
Get_Text 1530
GetAdvise 1530
GetCanonicalFormatEtc 1530
GetClassID 1530
GetClientSite 1531
GetClipboardData 1531
GetColorSet 1531
GetContentExtent 1531
GetControlInfo 1531
GetData 1532
GetDataHere 1532
GetDisplayString 1532
GetExtent 1532
GetInterfaceSafetyOptions 1533
GetMiscStatus 1533
GetMoniker 1533
GetPages 1533
GetPredefinedStrings 1533
GetPredefinedValue 1534
GetPropertyString 1534
GetPropertyStrings 1534
GetPropertyValue 1534
GetPropFromBag 1535
GetSizeMax 1535
GetUserClassID 1535
GetUserType 1535
GetWindow 1536
HandsOffStorage 1536
InitFromData 1536
Initialize 1536
InitializeControl 1537
InitNew 1537
InPlaceActivate 1537
InPlaceDeactivate 1537
InPlaceSite 1554
IsDirty 1537
IsUpToDate 1538
LoadFromStream 1538
MapPropertyToPage 1538
ObjQueryInterface 1538
OnAmbientPropertyChange 1539
OnDocWindowActivate 1539
OnFrameWindowActivate 1539
OnMnemonic 1539
PerformVerb 1540
PersistPropBagInitNew 1540
PersistPropBagLoad 1540
PersistPropBagSave 1540
PersistStorageInitNew 1541
PersistStorageLoad 1541

PersistStorageSave 1541
PersistStreamLoad 1541
PersistStreamSave 1541
PostMessageFilter 1542
PreMessageFilter 1542
PropChanged 1542
PropChanged method 1542
PropRequestEdit 1543
PropRequestEdit method 1543
PutPropInBag 1543
QueryGetData 1544
QuickActivate 1544
ReactivateAndUndo 1544
ResizeBorder 1544
SaveCompleted 1544
SaveToStream 1545
Set_BackColor 1545
Set_Caption 1545
Set_Enabled 1545
Set_Font 1546
Set_ForeColor 1546
Set_TabStop 1546
Set_Text 1546
SetAdvise 1546
SetClientSite 1547
SetColorScheme 1547
SetContentExtent 1547
SetData 1547
SetExtent 1548
SetHostNames 1548
SetInPlaceSite 1548
SetInterfaceSafetyOptions 1548
SetMoniker 1548
SetObjectRects 1549
ShowPropertyDialog 1549
StdClickEvent 1549
StdDblClickEvent 1549
StdKeyDownEvent 1550
StdKeyPressEvent 1550
StdKeyUpEvent 1550
StdMouseDownEvent 1550
StdMouseMoveEvent 1550
StdMouseUpEvent 1551
TActiveXControl Members 1518
TActiveXControl methods 1522
TActiveXControl properties 1552
TranslateAccelerator 1551
UIDeactivate 1551
Unadvise 1551
Unfreeze 1552
Update 1552
ViewObjectGetExtent 1552
WndProc 1552
TActiveXControlFactory class 1554
about TActiveXControlFactory class 1554
AddVerb 1557
Create 1556
Destroy 1557
GetLicenseFileName 1558
HasMachineLicense 1558
MiscStatus 1559
TActiveXControlFactory Members 1555
TActiveXControlFactory methods 1557
TActiveXControlFactory properties 1559
ToolboxBitmapID 1560
UpdateRegistry 1559
WinControlClass 1560
TActiveXPropertyPage class 1561
about TActiveXPropertyPage class 1561
Destroy 1563
Initialize 1563
PropertyPageImpl 1564
TActiveXPropertyPage Members 1561
TActiveXPropertyPage methods 1563
TActiveXPropertyPage properties 1563
TActiveXPropertyPageFactory class 1564
about TActiveXPropertyPageFactory class 1564
Create 1566

CreateComObject 1567
TActiveXPropertyPageFactory Members 1565
TActiveXPropertyPageFactory methods 1566
TAdapter class 23713
about TAdapter class 23713
Actions 23721
Data 23721
ModifyAccess 23721
OnAfterExecuteAction 23722
OnAfterGetActionResponse 23723
OnAfterUpdateFields 23723
OnAfterValidateFields 23723
OnBeforeExecuteAction 23724
OnBeforeGetActionResponse 23724
OnBeforeUpdateFields 23725
OnBeforeValidateFields 23725
OnGetActionParams 23725
OnIterateRecords 23726
TAdapter events 23722
TAdapter Members 23715
TAdapter properties 23721
ViewAccess 23722
TAdapterAction class 23726
about TAdapterAction class 23726
ActionName 23733
DisplayLabel 23734
ExecuteAccess 23734
OnAfterExecute 23735
OnAfterGetResponse 23735
OnBeforeExecute 23736
OnBeforeGetResponse 23736
OnExecute 23737
OnGetEnabled 23737
OnGetParams 23737
RedirectOptions 23734
TAdapterAction events 23735
TAdapterAction Members 23728
TAdapterAction properties 23733
TAdapterActionArrayStub class 1376
about TAdapterActionArrayStub class 1376
Action 1378
Create 1378
TAdapterActionArrayStub Members 1377
TAdapterActionArrayStub properties 1378
TAdapterActionButton class 25034
about TAdapterActionButton class 25034
ActionName 25039
Align 25040
Caption 25040
Custom 25040
DisplayColumns 25041
DisplayType 25041
ErrorPageName 25042
HideOptions 25042
ImageHeight 25043
ImageProducer 25043
ImageWidth 25043
PageName 25044
Style 25044
StyleRule 25045
TAdapterActionButton Members 25035
TAdapterActionButton properties 25039
VAlign 25045
TAdapterActionFieldValuesList class 873
about TAdapterActionFieldValuesList class 873
Create 875
CurrentActionFieldValuesList 876
Destroy 875
LocateActionFieldValuesOfAdapter 876
TAdapterActionFieldValuesList Members 874
TAdapterActionFieldValuesList methods 875
TAdapterActionFieldValuesList properties 876
TAdapterActionName class 25328
about TAdapterActionName class 25328
Action 25331
ActionName 25331
Adapter 25332
AssignTo 25330

Create 25329
TAdapterActionName Members 25328
TAdapterActionName methods 25330
TAdapterActionName properties 25331
TAdapterActions class 23738
about TAdapterActions class 23738
AdapterCanAddClass 23742
ImplCanAddClass 23743
ImplGetWebComponentNodeName 23744
SetWebParent 23744
TAdapterActions Members 23738
TAdapterActions methods 23742
TAdapterActions properties 23744
WebActions 23745
TAdapterActionsListHelper class 25046
about TAdapterActionsListHelper class 25046
AddDefaultActions 25052
Create 25047
FindAction 25048
GetActionsList 25049
GetAddAllFieldsItem 25049
GetAddFieldsDlgCaption 25049
GetAddFieldsItem 25050
GetNewFieldPrefix 25050
GetVisibleActions 25051
IsActionInUse 25051
TAdapterActionsListHelper Members 25046
TAdapterActionsListHelper methods 25048
TAdapterActionsListHelper properties 25052
TAdapterActionStub class 1379
about TAdapterActionStub class 1379
CreateActionArrayStub 1381
CreateActionStub 1381
Get_Array_ 1381
Get_AsFieldValue 1382
Get_AsHREF 1382
Get_CanExecute 1382
Get_DisplayLabel 1382
Get_DisplayStyle 1382
Get_Enabled 1383
Get_Visible 1383
LinkToPage 1383
TAdapterActionStub Members 1379
TAdapterActionStub methods 1380
TAdapterApplicationField class 23745
about TAdapterApplicationField class 23745
Adapter 23751
TAdapterApplicationField Members 23747
TAdapterApplicationField properties 23751
TAdapterApplicationTitleField class 23752
about TAdapterApplicationTitleField class 23752
GetDefaultFieldName 23758
ImplGetValue 23758
TAdapterApplicationTitleField Members 23753
TAdapterApplicationTitleField methods 23758
TAdapterBooleanField class 23758
about TAdapterBooleanField class 23758
DisplayLabel 23766
DisplayWidth 23766
FieldName 23767
ModifyAccess 23767
OnGetValue 23769
OnUpdateValue 23769
OnValidateValue 23769
ReadOnly 23767
TAdapterBooleanField events 23768
TAdapterBooleanField Members 23760
TAdapterBooleanField properties 23766
ValuesList 23768
ViewAccess 23768
TAdapterCommandColumn class 25052
about TAdapterCommandColumn class 25052
AddDefaultCommands 25058
Align 25059
Caption 25059
CaptionAttributes 25060
Custom 25060
DisplayColumns 25060

Style 25061
StyleRule 25061
TAdapterCommandColumn Members 25054
TAdapterCommandColumn properties 25058
VAlign 25062
TAdapterCommandGroup class 25062
about TAdapterCommandGroup class 25062
AddDefaultCommands 25069
Custom 25069
DisplayComponent 25069
Style 25070
StyleRule 25070
TAdapterCommandGroup Members 25064
TAdapterCommandGroup properties 25068
TAdapterComponentsStub class 1383
about TAdapterComponentsStub class 1383
FindObject 1384
TAdapterComponentsStub Members 1384
TAdapterComponentsStub methods 1384
TAdapterControlGroup class 25070
about TAdapterControlGroup class 25070
Adapter 25084
AdapterChange 25077
AdapterMode 25085
AddDefaultFields 25085
AddNotify 25077
Create 25076
CreateDefaultFields 25078
Destroy 25076
FindField 25078
GetAdapter 25079
GetAdapterModeOfAdapter 25079
GetAdapterModeOfDisplay 25080
GetAdapterVariableName 25080
GetControlRequirements 25080
GetDesigntimeWarnings 25081
GetFieldsList 25081
GetVisibleFields 25082
ImplGetControlRequirements 25082
IsFieldInUse 25082
Notification 25083
NotifyAdapterChange 25083
RemoveNotify 25083
SetAdapter 25084
TAdapterControlGroup Members 25072
TAdapterControlGroup methods 25076
TAdapterControlGroup properties 25084
VisibleFields 25086
TAdapterDataDisplay class 25086
about TAdapterDataDisplay class 25086
Align 25097
Create 25092
FieldConditions 25092
FieldName 25098
FindAssociatedField 25093
FormatCaption 25093
FormatColumnData 25093
FormatColumnHeading 25093
GetAdapterModeOfAdapter 25093
GetAdapterVariableName 25094
GetCaption 25094
GetControlRequirements 25094
GetDesigntimeWarnings 25094
GetFieldName 25095
GetVariablesContainer 25095
HideOptions 25098
ImplContent 25095
ImplFormatColumnData 25096
ImplFormatColumnHeading 25096
ImplGetControlRequirements 25096
ImplRestoreDefaults 25096
RestoreDefaults 25096
SetCaption 25097
SetFieldName 25097
TAdapterDataDisplay Members 25087
TAdapterDataDisplay methods 25092
TAdapterDataDisplay properties 25097
VAlign 25098

TAdapterDefaultActions class 23770
about TAdapterDefaultActions class 23770
ClearDefaultComponents 23775
DefaultWebComponents 23778
Destroy 23774
GetAddAllFieldsItem 23775
GetAddFieldsDlgCaption 23776
GetAddFieldsItem 23776
GetDefaultWebComponents 23776
GetNewFieldPrefix 23776
GetVariablesInUse 23777
SetDefaultWebComponents 23777
TAdapterDefaultActions Members 23770
TAdapterDefaultActions methods 23775
TAdapterDefaultActions properties 23777
TAdapterDefaultFields class 23778
about TAdapterDefaultFields class 23778
ClearDefaultComponents 23784
DefaultWebComponents 23786
Destroy 23783
GetAddAllFieldsItem 23784
GetAddFieldsDlgCaption 23784
GetAddFieldsItem 23784
GetDefaultWebComponents 23785
GetNewFieldPrefix 23785
GetVariablesInUse 23785
SetDefaultWebComponents 23786
TAdapterDefaultFields Members 23779
TAdapterDefaultFields methods 23783
TAdapterDefaultFields properties 23786
TAdapterDispatcher class 24840
about TAdapterDispatcher class 24840
OnActionRequestNotHandled 24844
OnAfterDispatchAction 24845
OnAfterDispatchImage 24845
OnBeforeDispatchAction 24846
OnBeforeDispatchImage 24846
OnGetAdapterRequestParams 24847
OnImageRequestNotHandled 24847
TAdapterDispatcher events 24844
TAdapterDispatcher Members 24841
TAdapterDispatchParams class 24848
about TAdapterDispatchParams class 24848
Create 24850
Destroy 24851
GetFailurePage 24851
GetHandler 24852
GetObjectIdentifier 24852
GetParams 24852
GetRequestIdentifier 24853
GetSuccessPage 24853
Handler 24853
ObjectIdentifier 24854
RequestIdentifier 24854
TAdapterDispatchParams Members 24849
TAdapterDispatchParams methods 24851
TAdapterDispatchParams properties 24853
TAdapterDisplayColumn class 25099
about TAdapterDisplayColumn class 25099
Align 25105
Caption 25105
CaptionAttributes 25105
Custom 25106
DisplayType 25106
FieldName 25107
HideOptions 25107
ImageHeight 25107
ImageWidth 25107
Style 25108
StyleRule 25108
TAdapterDisplayColumn Members 25100
TAdapterDisplayColumn properties 25104
VAlign 25108
TAdapterDisplayField class 25109
about TAdapterDisplayField class 25109
Align 25115
Caption 25116
CaptionAttributes 25116

CaptionPosition 25116
Custom 25117
DisplayColumns 25117
DisplayRows 25117
DisplayType 25118
DisplayWidth 25118
FieldName 25118
HideOptions 25119
ImageHeight 25119
ImageWidth 25119
InputType 25120
MaxLength 25120
SelectRows 25120
Style 25121
StyleRule 25121
TAdapterDisplayField Members 25110
TAdapterDisplayField properties 25115
TextAreaWrap 25121
VAlign 25122
ViewMode 25122
TAdapterDisplayReferenceGroup class 25122
about TAdapterDisplayReferenceGroup class 25122
AdapterChange 25128
Create 25127
Destroy 25128
DisplayComponent 25132
DisplayComponentChange 25129
GetAdapterDisplayCharacteristics 25129
GetDesigntimeWarnings 25130
NotifyAdapterChange 25130
OnDisplayComponentChange 25131
SetWebDisplayComponent 25131
TAdapterDisplayReferenceGroup Members 25123
TAdapterDisplayReferenceGroup methods 25128
TAdapterDisplayReferenceGroup properties 25131
TAdapterEditColumn class 25132
about TAdapterEditColumn class 25132
Align 25138
Caption 25139
CaptionAttributes 25139
CaptionPosition 25139
Custom 25140
DisplayColumns 25140
DisplayRows 25140
DisplayType 25141
DisplayWidth 25141
FieldName 25141
HideOptions 25142
ImageHeight 25142
ImageWidth 25142
InputType 25143
MaxLength 25143
SelectRows 25143
Style 25144
StyleRule 25144
TAdapterEditColumn Members 25133
TAdapterEditColumn properties 25138
TextAreaWrap 25144
VAlign 25145
ViewMode 25145
TAdapterEndUserDisplayNameField class 23787
about TAdapterEndUserDisplayNameField class 23787
GetDefaultFieldName 23793
ImplGetValue 23793
TAdapterEndUserDisplayNameField Members 23788
TAdapterEndUserDisplayNameField methods 23792
TAdapterEndUserField class 23793
about TAdapterEndUserField class 23793
Adapter 23800
TAdapterEndUserField Members 23795
TAdapterEndUserField properties 23799
TAdapterEndUserLoggedInField class 23800
about TAdapterEndUserLoggedInField class 23800
GetDefaultFieldName 23807
ImplGetValue 23807
TAdapterEndUserLoggedInField Members 23802
TAdapterEndUserLoggedInField methods 23806
TAdapterError class 23807

about TAdapterError class 23807
Create 23809
GetErrorText 23810
GetField 23810
GetID 23810
TAdapterError Members 23808
TAdapterError methods 23809
TAdapterErrorList class 25145
about TAdapterErrorList class 25145
Adapter 25149
TAdapterErrorList Members 25146
TAdapterErrorList properties 25149
TAdapterErrorsList class 23810
about TAdapterErrorsList class 23810
AddError 23815
AddError method 23814
Create 23813
Destroy 23814
FieldNames 23817
IDs 23817
ImplClearErrors 23815
ImplDefineLabel 23816
ImplGetError 23816
ImplGetErrorCount 23816
ImplGetObjectErrors 23816
ImplHasObjectErrors 23816
Messages 23818
TAdapterErrorsList Members 23811
TAdapterErrorsList methods 23814
TAdapterErrorsList properties 23817
TAdapterErrorsStub class 1384
about TAdapterErrorsStub class 1384
Create 1386
CreateErrorsStub 1386
DefineLabel 1387
Errors 1387
FieldErrors 1387
TAdapterErrorsStub Members 1385
TAdapterErrorsStub methods 1386
TAdapterErrorsStub properties 1387
TAdapterErrorStub class 1387
about TAdapterErrorStub class 1387
Create 1389
Get_Field 1389
Get_ID 1390
Get_Message 1390
TAdapterErrorStub Members 1388
TAdapterErrorStub methods 1389
TAdapterField class 23818
about TAdapterField class 23818
DisplayLabel 23826
DisplayWidth 23826
FieldName 23826
ModifyAccess 23827
OnGetDisplayText 23829
OnGetValue 23829
OnUpdateValue 23829
OnValidateValue 23830
ReadOnly 23827
Required 23827
TAdapterField events 23829
TAdapterField Members 23820
TAdapterField properties 23825
ValuesList 23828
ViewAccess 23828
TAdapterFieldGroup class 25150
about TAdapterFieldGroup class 25150
Adapter 25156
AdapterMode 25157
AddDefaultFields 25157
Custom 25158
Style 25158
StyleRule 25158
TAdapterFieldGroup Members 25151
TAdapterFieldGroup properties 25156
TAdapterFields class 23830
about TAdapterFields class 23830
AdapterCanAddClass 23835

ImplCanAddClass 23836
ImplGetWebComponentNodeName 23836
SetWebParent 23836
TAdapterFields Members 23831
TAdapterFields methods 23835
TAdapterFields properties 23837
WebFields 23837
TAdapterFieldStub class 1390
about TAdapterFieldStub class 1390
CreateFieldValuesStub 1393
CreateImageStub 1393
CreateValuesListStub 1394
Get_CanModify 1394
Get_CanView 1394
Get_DisplayLabel 1394
Get_DisplayStyle 1394
Get_DisplayText 1395
Get_DisplayWidth 1395
Get_EditText 1395
Get_Image 1395
Get_InputName 1396
Get_InputStyle 1396
Get_MaxLength 1396
Get_Required 1396
Get_Value 1396
Get_Values 1397
Get_ValuesList 1397
Get_ViewMode 1397
Get_Visible 1397
IsEqual 1398
TAdapterFieldStub Members 1391
TAdapterFieldStub methods 1393
TAdapterFieldValuesStub class 1398
about TAdapterFieldValuesStub class 1398
FindObject 1399
Get_Records 1399
Get_Value 1400
HasValue 1400
TAdapterFieldValuesStub Members 1398
TAdapterFieldValuesStub methods 1399
TAdapterFileField class 23837
about TAdapterFileField class 23837
DisplayLabel 23845
DisplayWidth 23846
FieldName 23846
ModifyAccess 23846
OnGetFileCount 23848
OnGetFileNames 23848
OnUploadFiles 23848
OnValidateFiles 23849
ReadOnly 23847
Required 23847
TAdapterFileField events 23848
TAdapterFileField Members 23839
TAdapterFileField properties 23845
ViewAccess 23847
TAdapterForm class 25159
about TAdapterForm class 25159
Custom 25164
FormMethod 25165
Style 25165
StyleRule 25165
TAdapterForm Members 25160
TAdapterForm properties 25164
TAdapterFormLayout class 25166
about TAdapterFormLayout class 25166
BreakButtons 25174
ChildTableIndent 25174
ColumnCount 25174
Create 25168
EndButtons 25169
EndField 25169
EndFields 25169
EndLayout 25169
EndRow 25169
EndTable 25170
FieldIndent 25175
GetAdapterFormLayout 25170

GetChildTableIndent 25170
GetFieldIndent 25170
GetTableIndent 25171
ImplLayoutButton 25171
ImplLayoutField 25171
ImplLayoutLabelAndField 25172
ImplLayoutTable 25172
NextColumn 25173
StartButtons 25173
StartFields 25173
StartRow 25173
StartTable 25174
TableHeader 25175
TableIndent 25175
TAdapterFormLayout Members 25166
TAdapterFormLayout methods 25168
TAdapterFormLayout properties 25174
TAdapterGotoPageAction class 23849
about TAdapterGotoPageAction class 23849
ExecuteAccess 23856
OnAfterExecute 23857
OnAfterGetResponse 23858
OnBeforeExecute 23858
OnBeforeGetResponse 23859
OnGetPageName 23859
OnGetParams 23859
RedirectOptions 23857
TAdapterGotoPageAction events 23857
TAdapterGotoPageAction Members 23851
TAdapterGotoPageAction properties 23856
TAdapterGrid class 25175
about TAdapterGrid class 25175
Adapter 25182
AdapterMode 25182
AddDefaultFields 25183
HeadingAttributes 25184
RowAttributes 25184
TableAttributes 25184
TAdapterGrid Members 25177
TAdapterGrid properties 25182
TAdapterHiddenField class 23860
about TAdapterHiddenField class 23860
Create 23861
GetName 23862
GetValue 23862
TAdapterHiddenField Members 23860
TAdapterHiddenField methods 23862
TAdapterHiddenFieldsList class 23862
about TAdapterHiddenFieldsList class 23862
Add 23866
Clear 23866
Create 23865
Destroy 23865
EndIterator 23866
FieldCount 23868
Fields 23868
GetCurrentIntf 23867
GetField 23867
GetFieldCount 23867
NextIteration 23867
StartIterator 23868
TAdapterHiddenFieldsList Members 23863
TAdapterHiddenFieldsList methods 23865
TAdapterHiddenFieldsList properties 23868
TAdapterHiddenFieldsStub class 1400
about TAdapterHiddenFieldsStub class 1400
Create 1402
HiddenFields 1402
TAdapterHiddenFieldsStub Members 1400
TAdapterHiddenFieldsStub methods 1402
TAdapterHiddenFieldsStub properties 1402
WriteFields 1402
TAdapterHiddenFieldStub class 1403
about TAdapterHiddenFieldStub class 1403
Create 1404
Get_Name 1405
Get_Value 1405
TAdapterHiddenFieldStub Members 1403

TAdapterHiddenFieldStub methods 1404
TAdapterImageField class 23869
about TAdapterImageField class 23869
DisplayLabel 23876
DisplayWidth 23877
FieldName 23877
HREF 23877
ModifyAccess 23878
OnGetHREF 23880
OnGetImage 23880
OnGetImageName 23881
OnGetParams 23881
OnUploadFile 23881
OnValidateFile 23882
ReadOnly 23878
Required 23879
TAdapterImageField events 23880
TAdapterImageField Members 23870
TAdapterImageField properties 23876
ValuesList 23879
ViewAccess 23879
TAdapterImageStub class 1405
about TAdapterImageStub class 1405
Create 1407
Get_AsHREF 1407
TAdapterImageStub Members 1405
TAdapterImageStub methods 1407
TAdapterItemRequestParams class 24855
about TAdapterItemRequestParams class 24855
Create 24857
Destroy 24857
FailurePage 24859
GetFailurePage 24857
GetParamValues 24858
GetSuccessPage 24858
ParamValues 24860
SetFailurePage 24858
SetSuccessPage 24859
SuccessPage 24860
TAdapterItemRequestParams Members 24855
TAdapterItemRequestParams methods 24857
TAdapterItemRequestParams properties 24859
TAdapterLoginField class 23882
about TAdapterLoginField class 23882
Adapter 23888
TAdapterLoginField Members 23884
TAdapterLoginField properties 23888
TAdapterMemoField class 23889
about TAdapterMemoField class 23889
DisplayLabel 23897
DisplayWidth 23897
FieldName 23898
ModifyAccess 23898
OnGetValue 23899
OnUpdateValue 23900
OnUploadFile 23900
ReadOnly 23898
Required 23899
TAdapterMemoField events 23899
TAdapterMemoField Members 23891
TAdapterMemoField properties 23897
ViewAccess 23899
TAdapterModes class 876
about TAdapterModes class 876
AdapterCount 880
Create 878
Destroy 878
FindAdapterMode 879
GetAdapterCount 879
GetAdapterMode 879
GetAdapterName 879
TAdapterModes Members 877
TAdapterModes methods 878
TAdapterModes properties 879
TAdapterMultiValueField class 23900
about TAdapterMultiValueField class 23900
DisplayLabel 23908
DisplayWidth 23909

FieldName 23909
ModifyAccess 23909
OnGetValueCount 23911
OnGetValues 23912
OnHasValue 23912
OnUpdateValues 23912
OnValidateValues 23913
ReadOnly 23910
Required 23910
TAdapterMultiValueField events 23911
TAdapterMultiValueField Members 23902
TAdapterMultiValueField properties 23908
ValuesList 23910
ViewAccess 23911
TAdapterNamedDisplayField class 23913
about TAdapterNamedDisplayField class 23913
Create 23919
DisplayLabel 23923
DisplayWidth 23924
GetDisplayStyle 23920
GetDisplayStyleType 23920
GetInputStyle 23920
GetInputStyleType 23921
GetViewMode 23921
GetViewModeType 23921
ImplGetDisplayLabel 23922
ImplGetDisplayWidth 23922
ImplGetRequired 23922
ReadOnly 23924
Required 23924
SetDisplayLabel 23923
TAdapterNamedDisplayField Members 23915
TAdapterNamedDisplayField methods 23919
TAdapterNamedDisplayField properties 23923
TAdapterNamedField class 23925
about TAdapterNamedField class 23925
ActionOrigValue 23943
ActionValue 23943
Adapter 23943
AsFormatted 23931
CheckModifyAccess 23931
CheckViewAccess 23931
EchoActionFieldValue 23944
FieldName 23944
FullyQualifyInputName 23932
GetAdapter 23932
GetDefaultDisplayLabel 23932
GetDefaultFieldName 23932
GetDisplayLabel 23933
GetDisplayWidth 23933
GetEchoActionFieldValue 23933
GetInputName 23934
GetMaxLength 23934
GetRequired 23934
GetValue 23935
GetVariableName 23935
GetVisible 23935
GetWebSnapObjClass 23936
HasModifyAccess 23936
HasViewAccess 23936
ImplAsFormatted 23937
ImplGetDisplayLabel 23937
ImplGetDisplayWidth 23937
ImplGetEchoActionFieldValue 23938
ImplGetInputName 23938
ImplGetMaxLength 23938
ImplGetRequired 23939
ImplGetValue 23939
ImplGetVariableName 23939
ImplGetVisible 23940
ImplHasModifyAccess 23940
ImplHasViewAccess 23940
ImplSetEchoActionFieldValue 23941
ImplSetFieldName 23941
ModifyAccess 23944
ParentChanged 23941
SetEchoActionFieldValue 23942
SetFieldName 23942

TAdapterNamedField Members 23926
TAdapterNamedField methods 23930
TAdapterNamedField properties 23942
Value 23945
ViewAccess 23945
TAdapterNextPageAction class 23945
about TAdapterNextPageAction class 23945
DisplayLabel 23954
ExecuteAccess 23954
GetDefaultActionName 23952
GetDefaultDisplayLabel 23953
GetNewPage 23953
OnAfterExecute 23955
OnAfterGetResponse 23955
OnBeforeExecute 23956
OnBeforeGetResponse 23956
OnGetParams 23957
RedirectOptions 23954
TAdapterNextPageAction events 23955
TAdapterNextPageAction Members 23947
TAdapterNextPageAction methods 23952
TAdapterNextPageAction properties 23953
TAdapterNextPageField class 23957
about TAdapterNextPageField class 23957
Destroy 23963
DisplayLabel 23966
DisplayWidth 23967
FieldName 23967
GetAdapterValuesList 23964
GetInputStyleType 23964
ImplGetEchoActionFieldValue 23965
ImplGetValue 23965
ImplSetEchoActionFieldValue 23965
TAdapterNextPageField Members 23959
TAdapterNextPageField methods 23964
TAdapterNextPageField properties 23966
UpdateValue 23966
TAdapterNotifier class 1567
about TAdapterNotifier class 1567
Create 1569
OnChanged 1569
OnRequestEdit 1570
TAdapterNotifier Members 1568
TAdapterNotifier methods 1569
TAdapterObjectErrorsList class 23968
about TAdapterObjectErrorsList class 23968
Create 23970
ImplDefineLabel 23970
ImplGetError 23970
ImplGetErrorCount 23971
TAdapterObjectErrorsList Members 23968
TAdapterObjectErrorsList methods 23970
TAdapterPageProducer class 19802
about TAdapterPageProducer class 19802
HTMLDoc 19808
HTMLFile 19809
OnAfterGetContent 19812
OnBeforeGetContent 19812
Styles 19810
StylesFile 19811
TAdapterPageProducer events 19812
TAdapterPageProducer Members 19803
TAdapterPageProducer properties 19808
WebPageItems 19811
TAdapterPasswordField class 23971
about TAdapterPasswordField class 23971
DisplayLabel 23978
DisplayWidth 23979
FieldName 23979
GetInputStyleType 23978
ImplGetValue 23978
TAdapterPasswordField Members 23973
TAdapterPasswordField methods 23977
TAdapterPasswordField properties 23978
TAdapterPrevPageAction class 23979
about TAdapterPrevPageAction class 23979
DisplayLabel 23988
ExecuteAccess 23988

wwww

GetDefaultActionName 23986
GetDefaultDisplayLabel 23987
GetNewPage 23987
OnAfterExecute 23989
OnAfterGetResponse 23989
OnBeforeExecute 23990
OnBeforeGetResponse 23990
OnGetParams 23991
RedirectOptions 23988
TAdapterPrevPageAction events 23989
TAdapterPrevPageAction Members 23981
TAdapterPrevPageAction methods 23986
TAdapterPrevPageAction properties 23987
TAdapterRecordsStub class 1407
about TAdapterRecordsStub class 1407
Adapter 1409
Create 1409
TAdapterRecordsStub Members 1408
TAdapterRecordsStub properties 1409
TAdapterRecordStub class 1409
about TAdapterRecordStub class 1409
Create 1411
Impl 1411
TAdapterRecordStub Members 1410
TAdapterRecordStub properties 1411
TAdapterRequestParamsImpl class 880
about TAdapterRequestParamsImpl class 880
AssignTo 882
Create 881
GetParamCount 882
GetParamName 882
GetParamValue 882
IndexOfParam 883
ParamCount 883
ParamNames 883
ParamValues 884
TAdapterRequestParamsImpl Members 880
TAdapterRequestParamsImpl methods 881
TAdapterRequestParamsImpl properties 883
ValueOfParam 883
TAdapterStub class 1412
about TAdapterStub class 1412
CreateErrorsStub 1413
CreateHiddenFieldsStub 1414
FindObject 1414
Get_CanModify 1414
Get_CanView 1414
Get_Errors 1414
Get_HiddenFields 1415
Get_HiddenRecordFields 1415
Get_Mode 1415
Set_Mode 1415
TAdapterStub Members 1412
TAdapterStub methods 1413
TAdapterSubComponent class 23991
about TAdapterSubComponent class 23991
Adapter 24006
CanAddClass 23996
EndIterator 23997
FindNamedVariable 23997
FindVariable 23998
FindVariableContainer 23998
GetCurrentObject 23999
GetDefaultWebComponents 23999
GetParentComponent 24000
GetVariable 24000
GetVariableCount 24000
GetVariablesInUse 24001
GetWebComponentNodeName 24001
HasParent 24002
ImplCanAddClass 24002
ImplGetWebComponentNodeName 24003
NextIteration 24003
NotifyAdapterChange 24003
ReadState 24004
SetDefaultWebComponents 24004
SetParentComponent 24005
SetWebParent 24005

StartIterator 24006
TAdapterSubComponent Members 23992
TAdapterSubComponent methods 23995
TAdapterSubComponent properties 24006
VariablesInUse 24007
TAdapterSubComponentStub class 1416
about TAdapterSubComponentStub class 1416
Create 1417, 1418
Create constructor 1417
Get_Name 1418
GetComponent 1418
ObjectIntf 1419
TAdapterSubComponentStub Members 1416
TAdapterSubComponentStub methods 1418
TAdapterSubComponentStub properties 1419
TAdapterUpdateField class 24007
about TAdapterUpdateField class 24007
CheckValueChange 24015
ConvertValue 24015
ConvertValueCount 24015
ConvertValues 24016
Create 24014
DoOnValidateValue 24016
GetAdapterValuesList 24016
GetInputStyleType 24017
GetValueCount 24017
GetValues 24017
ImplAsFormatted 24018
ImplCheckValueChange 24018
ImplGetDisplayLabel 24018
ImplGetDisplayWidth 24019
ImplGetEchoActionFieldValue 24019
ImplGetVariableName 24019
ImplSetEchoActionFieldValue 24020
ImplSetFieldName 24020
ImplUpdateValue 24020
ImplValidateValue 24020
IsMultiValueField 24021
Notification 24021
OnGetDisplayText 24023
OnValidateValue 24024
TAdapterUpdateField events 24023
TAdapterUpdateField Members 24009
TAdapterUpdateField methods 24014
TAdapterUpdateField properties 24022
UpdateValue 24021
ValidateValue 24022
Value 24022
ValuesList 24023
TAdapterUserNameField class 24024
about TAdapterUserNameField class 24024
DisplayLabel 24031
DisplayWidth 24031
FieldName 24032
ImplGetValue 24031
TAdapterUserNameField Members 24026
TAdapterUserNameField methods 24030
TAdapterUserNameField properties 24031
TAdapterValuesListStub class 1419
about TAdapterValuesListStub class 1419
CreateImageStub 1421
FindObject 1421
Get_Image 1421
Get_Records 1421
Get_Value 1421
Get_ValueName 1422
ImageOfValue 1422
NameOfValue 1422
TAdapterValuesListStub Members 1419
TAdapterValuesListStub methods 1420
TADOBlobStream class 979
about TADOBlobStream class 979
Create 981
Destroy 981
ReadBlobData 982
Realloc 982
TADOBlobStream Members 979
TADOBlobStream methods 982

Truncate 983
Write 983
TADOCommand class 984
about TADOCommand class 984
ActiveConnection 995
Assign 990
AssignCommandText 990
Cancel 991
CheckCommandText 991
ClearActiveConnection 991
CommandObject 995
CommandText 995
CommandTextAlias 996
CommandTimeout 996
CommandType 997
ComponentLoading 992
ComponentRef 997
Connection 997
ConnectionStateChange 992
ConnectionString 998
Create 988
Destroy 989
Execute 992
Execute method 992
ExecuteOptions 999
Initialize 993
OpenConnection 993
ParamCheck 999
Parameters 999
Prepared 1000
Properties 1000
SetCommandText 994
SetConnectionFlag 994
States 1001
TADOCommand Members 985
TADOCommand methods 989
TADOCommand properties 994
TADOConnection class 1001
about TADOConnection class 1001
AfterConnect 1034
AfterDisconnect 1034
Attributes 1020
BeforeConnect 1034
BeforeDisconnect 1035
BeginTrans 1008
BeginTransComplete 1008
Cancel 1009
CheckActive 1009
CheckDisconnect 1009
CheckInactive 1010
CommandCount 1021
Commands 1021
CommandTimeout 1022
CommitTrans 1010
CommitTransComplete 1010
ConnectComplete 1010
Connected 1022
ConnectionObject 1023
ConnectionPoint 1011
ConnectionString 1024
ConnectionTimeout 1025
ConnectOptions 1026
Create 1006
CursorLocation 1026
DataSets 1027
DefaultDatabase 1027
Destroy 1006
Disconnect 1011
DoConnect 1011
DoDisconnect 1011
Errors 1028
Execute 1012
Execute method 1012
ExecuteComplete 1013
GetConnected 1013
GetErrors 1013
GetFieldNames 1014
GetFieldNames method 1014

GetProcedureNames 1014
GetProcedureNames method 1014
GetTableNames 1015
GetTableNames method 1015
InfoMessage 1016
InTransaction 1028
IsolationLevel 1029
KeepConnection 1029
Loaded 1016
LoginPrompt 1029
Mode 1031
OnBeginTransComplete 1035
OnCommitTransComplete 1036
OnConnectComplete 1036
OnDisconnect 1037
OnExecuteComplete 1038
OnInfoMessage 1038
OnLogin 1031
OnRollbackTransComplete 1039
OnWillConnect 1039
OnWillExecute 1040
Open 1016
OpenSchema 1017
Properties 1032
Provider 1032
RegisterClient 1017
RollbackTrans 1018
RollbackTransComplete 1018
State 1032
TADOConnection events 1033
TADOConnection Members 1002
TADOConnection methods 1007
TADOConnection properties 1020
UnRegisterClient 1019
Version 1033
WaitForConnectComplete 1019
WillConnect 1019
WillExecute 1020
TADODataSet class 1041
about TADODataSet class 1041
AfterCancel 1067
AfterClose 1069
AfterDelete 1071
AfterEdit 1073
AfterInsert 1074
AfterOpen 1075
AfterPost 1076
AfterScroll 1076
BeforeCancel 1077
BeforeClose 1077
BeforeDelete 1078
BeforeEdit 1078
BeforeInsert 1079
BeforeOpen 1080
BeforePost 1081
BeforeScroll 1082
CommandText 1057
CommandTimeout 1057
CommandType 1057
CreateDataSet 1055
DataEvent 1055
DataSetField 1058
DataSource 1059
EnableBCD 1059
FieldDefs 1060
GetIndexNames 1055
IndexDefs 1062
IndexFieldNames 1063
IndexName 1063
MasterFields 1064
OnCalcFields 1082
OnDeleteError 1083
OnEditError 1083
OnNewRecord 1084
OnPostError 1084
OpenCursor 1056
ParamCheck 1065
Parameters 1065

Prepared 1066
RDSConnection 1066
SetConnection 1056
SetDataSetField 1056
StoreDefs 1066
TADODataSet events 1067
TADODataSet Members 1042
TADODataSet methods 1054
TADODataSet properties 1056
TAdoDbxCommand class 1763
about TAdoDbxCommand class 1763
Cancel 1767
Clone 1767
CommandText 1772
CommandTimeout 1773
CommandType 1773
CreateDbParameter 1767
DesignTimeVisible 1774
Destroy 1766
ExecuteDbDataReader 1767
ExecuteNonQuery 1768
ExecuteScalar 1768
get_CommandText 1768
get_CommandTimeout 1769
get_CommandType 1769
get_DbConnection 1769
get_DbParameterCollection 1769
get_DbTransaction 1769
get_DesignTimeVisible 1770
get_UpdatedRowSource 1770
Parameters 1774
Prepare 1770
set_CommandText 1770
set_CommandTimeout 1771
set_CommandType 1771
set_DbConnection 1771
set_DbTransaction 1771
set_DesignTimeVisible 1771
set_UpdatedRowSource 1772
TAdoDbxCommand Members 1765
TAdoDbxCommand methods 1766
TAdoDbxCommand properties 1772
Transaction 1774
UpdateRowSource 1775
TAdoDbxCommandBuilder class 1776
about TAdoDbxCommandBuilder class 1776
AdoDbxRowUpdatingHandler 1778
ApplyParameterInfo 1778
Create 1777
Create constructor 1777
GetParameterName 1778, 1779
GetParameterName method 1778
GetParameterPlaceholder 1779
SetRowUpdatingHandler 1779
TAdoDbxCommandBuilder Members 1777
TAdoDbxCommandBuilder methods 1777
TAdoDbxConnection class 1779
about TAdoDbxConnection class 1779
AdoDbxIsolation 1784
BeginDbTransaction 1784
ChangeDatabase 1784
Clone 1785
Close 1785
ConnectionString 1788
ConnectionTimeout 1789
Create 1781
Create constructor 1781
CreateDbCommand 1785
Database 1789
DataSource 1789
Dbx 1790
Destroy 1782
FConnectionString 1782
FConnectionTimeout 1783
FDbxExtension 1783
FDriverName 1783
FSchemaItems 1783
get_ConnectionString 1785

get_ConnectionTimeout 1786
get_Database 1786
get_DataSource 1786
get_ServerVersion 1786
get_State 1786
GetSchema 1787
GetSchema method 1787
Open 1787
ServerVersion 1790
set_ConnectionString 1788
State 1790
TAdoDbxConnection fields 1782
TAdoDbxConnection Members 1780
TAdoDbxConnection methods 1783
TAdoDbxConnection properties 1788
TAdoDbxConnectionStringBuilder class 1790
about TAdoDbxConnectionStringBuilder class 1790
ConnectionName 1806
Create 1793
Database 1806
DelegateConnection 1807
DelegateDriver 1807
DriverAssembly 1807
DriverAssemblyLoader 1808
DriverName 1808
DriverPackage 1808
DriverPackageLoader 1809
DriverUnit 1809
ErrorResourceFile 1809
get_ConnectionName 1795
get_Database 1795
get_DelegateConnection 1795
get_DelegateDriver 1795
get_DriverAssembly 1795
get_DriverAssemblyLoader 1796
get_DriverName 1796
get_DriverPackage 1796
get_DriverPackageLoader 1796
get_DriverUnit 1796
get_ErrorResourceFile 1797
get_GetDriverFunc 1797
get_HostName 1797
get_IsolationLevel 1797
get_Item 1797
get_LibraryName 1798
get_MaxBlobSize 1798
get_MetaDataAssemblyLoader 1798
get_MetaDataPackageLoader 1798
get_Password 1798
get_Port 1799
get_ProductVersion 1799
get_Role 1799
get_SchemaOverride 1799
get_User_Name 1799
get_VendorLib 1800
GetDriverFunc 1809
HostName 1810
IsolationLevel 1810
LibraryName 1811
MaxBlobSize 1811
MetaDataAssemblyLoader 1811
MetaDataPackageLoader 1811
Password 1812
Port 1812
ProductVersion 1812
Role 1813
SchemaOverride 1813
set_ConnectionName 1800
set_Database 1800
set_DelegateConnection 1800
set_DelegateDriver 1800
set_DriverAssembly 1801
set_DriverAssemblyLoader 1801
set_DriverName 1801
set_DriverPackage 1801
set_DriverPackageLoader 1801
set_DriverUnit 1802
set_ErrorResourceFile 1802

set_GetDriverFunc 1802
set_HostName 1802
set_IsolationLevel 1802
set_Item 1803
set_LibraryName 1803
set_MaxBlobSize 1803
set_MetaDataAssemblyLoader 1803
set_MetaDataPackageLoader 1803
set_Password 1804
set_Port 1804
set_ProductVersion 1804
set_Role 1804
set_SchemaOverride 1804
set_User_Name 1805
set_VendorLib 1805
TAdoDbxConnectionStringBuilder Members 1791
TAdoDbxConnectionStringBuilder methods 1793
TAdoDbxConnectionStringBuilder properties 1805
TryGetValue 1805
User_Name 1814
VendorLib 1814
TAdoDbxDataAdapter class 1814
about TAdoDbxDataAdapter class 1814
add_RowUpdated 1819
add_RowUpdating 1820
Create 1818
Create constructor 1818
DeleteCommand 1823
FEventRowUpdated 1819
FEventRowUpdating 1819
get_DeleteCommand 1820
get_InsertCommand 1820
get_SelectCommand 1821
get_UpdateCommand 1821
InsertCommand 1824
OnRowUpdated 1821
OnRowUpdating 1821
remove_RowUpdated 1821
remove_RowUpdating 1822
RowUpdated 1826
RowUpdating 1826
SelectCommand 1825
set_DeleteCommand 1822
set_InsertCommand 1822
set_SelectCommand 1823
set_UpdateCommand 1823
TAdoDbxDataAdapter events 1826
TAdoDbxDataAdapter fields 1818
TAdoDbxDataAdapter Members 1817
TAdoDbxDataAdapter methods 1819
TAdoDbxDataAdapter properties 1823
UpdateCommand 1825
TAdoDbxDataReader class 1827
about TAdoDbxDataReader class 1827
Close 1832
Depth 1843
Destroy 1830
FCommandBehavior 1831
FDbxCommand 1831
FDbxReader 1831
FieldCount 1843
FPeekAheadRead 1832
get_Depth 1833
get_FieldCount 1833
get_HasRows 1833
get_IsClosed 1833
get_Item 1834
get_Item method 1834
get_RecordsAffected 1834
GetBoolean 1834
GetByte 1835
GetBytes 1835
GetChar 1835
GetChars 1836
GetData 1836
GetDataTypeName 1837
GetDateTime 1837
GetDecimal 1837

GetDouble 1837
GetEnumerator 1838
GetFieldType 1838
GetFloat 1838
GetGuid 1839
GetInt16 1839
GetInt32 1839
GetInt64 1839
GetName 1840
GetOrdinal 1840
GetSchemaTable 1840
GetString 1841
GetValue 1841
GetValues 1841
HasRows 1843
IsClosed 1844
IsDBNull 1841
Item 1844
NextResult 1842
Read 1842
RecordsAffected 1844
TAdoDbxDataReader fields 1831
TAdoDbxDataReader Members 1829
TAdoDbxDataReader methods 1832
TAdoDbxDataReader properties 1842
TAdoDbxDataSourceEnumerator class 1845
about TAdoDbxDataSourceEnumerator class 1845
TAdoDbxException class 1845
about TAdoDbxException class 1845
Create 1846
Create constructor 1846
TAdoDbxException Members 1846
TAdoDbxExtension class 1846
about TAdoDbxExtension class 1846
Create 1847
FConnection 1847
GetVendorProperty 1848
TAdoDbxExtension fields 1847
TAdoDbxExtension Members 1847
TAdoDbxExtension methods 1848
TAdoDbxMetaDataColumnNames class 1848
about TAdoDbxMetaDataColumnNames class 1848
RestrictionDefault 1849
RestrictionName 1849
RestrictionNumber 1849
TAdoDbxMetaDataColumnNames constants 1849
TAdoDbxMetaDataColumnNames Members 1849
TAdoDbxParameter class 1850
about TAdoDbxParameter class 1850
Clone 1855
Create 1852
Create constructor 1852
DbType 1860
DbxDataType 1860
DbxSubType 1861
Destroy 1853
Direction 1861
FConnection 1853
FDbxContext 1853
FSourceColumn 1854
FSourceColumnNullMapping 1854
FSourceVersion 1854
get_DbType 1855
get_Direction 1855
get_IsNullable 1855
get_ParameterName 1856
get_Precision 1856
get_Scale 1856
get_Size 1856
get_SourceColumn 1856
get_SourceColumnNullMapping 1857
get_SourceVersion 1857
get_Value 1857
IsNullable 1862
ParameterName 1862
Precision 1862
ResetDbType 1857
Scale 1863

eeee

set_DbType 1857
set_Direction 1858
set_IsNullable 1858
set_ParameterName 1858
set_Precision 1858
set_Scale 1858
set_Size 1859
set_SourceColumn 1859
set_SourceColumnNullMapping 1859
set_SourceVersion 1859
set_Value 1859
Size 1863
SourceColumn 1863
SourceColumnNullMapping 1864
SourceVersion 1864
TAdoDbxParameter fields 1853
TAdoDbxParameter Members 1851
TAdoDbxParameter methods 1854
TAdoDbxParameter properties 1860
Value 1865
TAdoDbxParameterCollection class 1865
about TAdoDbxParameterCollection class 1865
Add 1868
AddRange 1868
Clear 1869
Contains 1869
Contains method 1869
CopyTo 1869
Destroy 1867
FDBXParameters 1867
FParameters 1867
get_Count 1870
get_IsFixedSize 1870
get_IsReadOnly 1870
get_IsSynchronized 1870
get_SyncRoot 1871
GetEnumerator 1871
GetParameter 1871, 1872
GetParameter method 1871
IndexOf 1872
IndexOf method 1872
Insert 1872
Remove 1873
RemoveAt 1873
RemoveAt method 1873
SetParameter 1874
SetParameter method 1874
TAdoDbxParameterCollection fields 1867
TAdoDbxParameterCollection Members 1866
TAdoDbxParameterCollection methods 1867
TAdoDbxProviderFactory class 1874
about TAdoDbxProviderFactory class 1874
AdoDbxException 1877
CanCreateDataSourceEnumerator 1880
Create 1875, 1876
Create constructor 1875
CreateCommand 1877
CreateCommandBuilder 1877
CreateConnection 1878
CreateConnectionStringBuilder 1878
CreateDataAdapter 1878
CreateDataSourceEnumerator 1879
CreateParameter 1879
DbxToAdoType 1879
Destroy 1876
Instance 1876
ReadCanCreateDataSourceEnumerator 1879
TAdoDbxProviderFactory fields 1876
TAdoDbxProviderFactory Members 1875
TAdoDbxProviderFactory methods 1876
TAdoDbxProviderFactory properties 1880
TAdoDbxRowUpdatedEventArgs class 1880
about TAdoDbxRowUpdatedEventArgs class 1880
Command 1881
get_Command 1881
TAdoDbxRowUpdatedEventArgs Members 1880
TAdoDbxRowUpdatedEventArgs methods 1881
TAdoDbxRowUpdatedEventArgs properties 1881

TAdoDbxRowUpdatingEventArgs class 1881
about TAdoDbxRowUpdatingEventArgs class 1881
Command 1883
get_Command 1882
set_Command 1882
TAdoDbxRowUpdatingEventArgs Members 1882
TAdoDbxRowUpdatingEventArgs methods 1882
TAdoDbxRowUpdatingEventArgs properties 1883
TAdoDbxSchemaTableColumnNames class 1883
about TAdoDbxSchemaTableColumnNames class 1883
DbxDataType 1884
DbxSubType 1884
TAdoDbxSchemaTableColumnNames constants 1883
TAdoDbxSchemaTableColumnNames Members 1883
TAdoDbxTransaction class 1884
about TAdoDbxTransaction class 1884
Commit 1887
Destroy 1886
FConnection 1886
FIsolation 1886
FTransaction 1887
get_DbConnection 1887
get_IsolationLevel 1888
Rollback 1888
TAdoDbxTransaction fields 1886
TAdoDbxTransaction Members 1885
TAdoDbxTransaction methods 1887
TADOQuery class 1084
about TADOQuery class 1084
CommandTimeout 1101
Create 1099
DataSource 1101
Destroy 1099
EnableBCD 1103
ExecSQL 1100
ParamCheck 1103
Parameters 1104
Prepared 1104
QueryChanged 1100
RowsAffected 1105
SQL 1105
TADOQuery Members 1085
TADOQuery methods 1099
TADOQuery properties 1100
TADOStoredProc class 1106
about TADOStoredProc class 1106
CommandTimeout 1122
Create 1120
DataSource 1122
EnableBCD 1123
ExecProc 1121
Parameters 1123
Prepared 1124
ProcedureName 1124
TADOStoredProc Members 1107
TADOStoredProc methods 1121
TADOStoredProc properties 1122
TADOTable class 1125
about TADOTable class 1125
CommandTimeout 1140
Create 1139
EnableBCD 1141
GetIndexNames 1140
IndexDefs 1141
IndexFieldNames 1142
IndexName 1143
MasterFields 1143
MasterSource 1144
ReadOnly 1144
TableDirect 1145
TableName 1145
TADOTable Members 1125
TADOTable methods 1139
TADOTable properties 1140
TADTField class 6736
about TADTField class 6736
Create 6742
GetSize 6743

gggggg

TADTField Members 6737
TADTField methods 6743
TAggDefinition class 17748
about TAggDefinition class 17748
AggProc 17748
TAggDefinition Members 17748
TAggDefinition properties 17748
TAggregate class 8268
about TAggregate class 8268
Activate 8271
Active 8274
AggHandle 8274
AggregateName 8275
Assign 8272
Create 8270
DataSet 8275
DataSize 8275
DataType 8276
DependentFields 8276
Destroy 8271
Expression 8276
GetDisplayName 8272
GroupingLevel 8277
IndexName 8278
InUse 8278
OnUpdate 8279
RecBufOfs 8278
TAggregate events 8279
TAggregate Members 8269
TAggregate methods 8271
TAggregate properties 8273
Value 8273
Visible 8279
TAggregatedObject class 22248
_AddRef 22250
_Release 22250
about TAggregatedObject class 22248
Controller 22251
Create 22249
QueryInterface 22250
TAggregatedObject Members 22248
TAggregatedObject methods 22250
TAggregatedObject properties 22251
TAggregateField class 6743
about TAggregateField class 6743
Active 6752
Create 6749
currency 6752
DisplayFormat 6753
DisplayName 6756
Expression 6756
FieldKind 6757
GetAsString 6750
GetAsVariant 6750
GetText 6751
GroupingLevel 6758
Handle 6758
IndexName 6758
Precision 6759
Reset 6751
ResultType 6759
SetDisplayFormat 6751
TAggregateField Members 6744
TAggregateField methods 6750
TAggregateField properties 6751
Visible 6760
TAggregates class 8280
about TAggregates class 8280
Add 8282
Clear 8283
Create 8282
Find 8283
GetOwner 8283
IndexOf 8284
Items 8285
TAggregates Members 8280
TAggregates methods 8282
TAggregates properties 8284

taInsert enumeration member 5439
taLeftJustify enumeration member 3269
TAnimate class 3422
about TAnimate class 3422
Active 3441
Align 3442
Anchors 3446
AutoSize 3446
BorderWidth 3447
CanAutoSize 3435
Center 3447
Color 3447
CommonAVI 3448
Constraints 3450
Create 3433
CreateParams 3436
CreateWnd 3436
DestroyWnd 3437
DoClose 3437
DoOpen 3437
DoStart 3437
DoStop 3438
FileName 3450
FrameCount 3451
FrameHeight 3451
FrameWidth 3451
Loaded 3438
OnClose 3461
OnOpen 3461
OnStart 3462
OnStop 3462
Open 3452
ParentColor 3453
ParentShowHint 3453
Play 3438
Repetitions 3454
Reset 3439
ResHandle 3454
ResId 3455
ResName 3455
Seek 3439
ShowHint 3456
StartFrame 3457
Stop 3440
StopFrame 3458
TAnimate events 3461
TAnimate Members 3423
TAnimate methods 3435
TAnimate properties 3440
Timers 3458
Transparent 3459
Visible 3459
TAppletApplication class 6497
about TAppletApplication class 6497
ControlPanelHandle 6505
Create 6501
CreateForm 6502
Destroy 6501
DoCount 6502
DoExit 6503
DoHandleException 6503
DoInit 6503
DoSetup 6503
HandleException 6503
Initialize 6504
ModuleCount 6505
Modules 6505
OnCount 6506
OnException 6507
OnExit 6506
OnInit 6507
OnSetup 6506
Run 6504
TAppletApplication events 6506
TAppletApplication Members 6498
TAppletApplication methods 6502
TAppletApplication properties 6504
TAppletModule class 6507

about TAppletModule class 6507
AppletIcon 6513
Caption 6514
Create 6511
Data 6514
Destroy 6511
DoActivate 6512
DoInquire 6512
DoNewInquire 6512
DoStartWParms 6513
DoStop 6513
Help 6514
OnActivate 6516
OnInquire 6516
OnNewInquire 6517
OnStartWParms 6517
OnStop 6517
ResidIcon 6515
ResidInfo 6515
ResidName 6515
TAppletModule events 6516
TAppletModule Members 6508
TAppletModule methods 6512
TAppletModule properties 6513
TApplication class 13660
about TApplication class 13660
ActionUpdateDelay 13704
ActivateHint 13668
Active 13704
ActiveFormHandle 13705
AddPopupForm 13669
AllowTesting 13705
AutoDragDocking 13705
BiDiKeyboard 13706
BiDiMode 13706
BringToFront 13669
CancelHint 13669
ControlDestroyed 13670
Create 13666
CreateForm 13670
CreateHandle 13672
CurrentHelpFile 13707
Destroy 13667
DialogHandle 13707
DoApplicationIdle 13673
EnumAllWindowsOnActivateHint 13708
ExecuteAction 13673
ExeName 13709
Handle 13710
HandleException 13674
HandleMessage 13674
HelpCommand 13678
HelpContext 13679
HelpFile 13711
HelpJump 13679
HelpKeyword 13680
HelpShowTableOfContents 13680
HelpSystem 13711
HideHint 13681
Hint 13712
HintColor 13715
HintHidePause 13715
HintMouseMessage 13681
HintPause 13715
HintShortCuts 13716
HintShortPause 13716
HookMainWindow 13681
HookSynchronizeWakeup 13682
Icon 13717
Idle 13682
Initialize 13682
IsDlgMsg 13685
IsHintMsg 13685
IsKeyMsg 13685
IsMDIMsg 13685
IsPreProcessMessage 13685
IsRightToLeft 13686
IsShortCut 13686

MainForm 13718
MainFormHandle 13719
MainFormOnTaskBar 13719
MessageBox 13686
Minimize 13688
ModalFinished 13689
ModalLevel 13720
ModalPopupMode 13720
ModalStarted 13689
NonBiDiKeyboard 13720
NormalizeAllTopMosts 13690
NormalizeTopMosts 13690
OnActionExecute 13725
OnActionUpdate 13726
OnActivate 13726
OnDeactivate 13727
OnException 13728
OnGetActiveFormHandle 13729
OnGetMainFormHandle 13729
OnHelp 13730
OnHint 13731
OnIdle 13733
OnMessage 13737
OnMinimize 13739
OnModalBegin 13740
OnModalEnd 13740
OnRestore 13741
OnSettingChange 13741
OnShortCut 13741
OnShowHint 13742
PopupControlWnd 13721
ProcessMessages 13691
RemovePopupForm 13694
Restore 13694
RestoreTopMosts 13695
Run 13696
ShowException 13698
ShowHint 13721
ShowMainForm 13722
TApplication events 13724
TApplication Members 13661
TApplication methods 13667
TApplication properties 13703
Terminate 13699
Terminated 13723
Title 13723
UnhookMainWindow 13701
UnhookSynchronizeWakeup 13701
UpdateAction 13701
UpdateFormatSettings 13724
UpdateMetricSettings 13724
UseRightToLeftAlignment 13702
UseRightToLeftReading 13702
UseRightToLeftScrollBar 13703
TApplicationAdapter class 24032
about TApplicationAdapter class 24032
Actions 24040
ApplicationTitle 24040
Data 24041
OnAfterExecuteAction 24041
OnAfterGetActionResponse 24042
OnBeforeExecuteAction 24042
OnBeforeGetActionResponse 24042
OnGetActionParams 24043
TApplicationAdapter events 24041
TApplicationAdapter Members 24034
TApplicationAdapter properties 24040
TApplicationEvents class 1326
about TApplicationEvents class 1326
OnActionExecute 1330
OnActionUpdate 1331
OnActivate 1333
OnDeactivate 1334
OnException 1335
OnHelp 1336
OnHint 1337
OnIdle 1339
OnMessage 1340

OnMinimize 1341
OnModalBegin 1342
OnModalEnd 1342
OnRestore 1342
OnSettingChange 1343
OnShortCut 1344
OnShowHint 1345
TApplicationEvents events 1329
TApplicationEvents Members 1326
TApplyUpdatesButton class 16480
about TApplyUpdatesButton class 16480
AddElements 16485
Caption 16486
Create 16484
Custom 16486
GetSubComponents 16485
ImplAddElements 16485
ImplContent 16485
Style 16487
StyleRule 16487
TApplyUpdatesButton Members 16481
TApplyUpdatesButton methods 16484
TApplyUpdatesButton properties 16486
XMLBroker 16488
XMLUseParent 16488
taRightJustify enumeration member 3269
TArrayField class 6760
about TArrayField class 6760
Bind 6767
Create 6766
SetSize 6767
Size 6768
TArrayField Members 6761
TArrayField methods 6767
TArrayField properties 6767
TAutoDispatch class 17894
about TAutoDispatch class 17894
AddRef 17895
AutoObject 17897
Create 17895
GetAutoObject 17895
GetIDsOfNames 17896
GetTypeInfo 17896
GetTypeInfoCount 17896
Invoke 17896
QueryInterface 17897
Release 17897
TAutoDispatch Members 17894
TAutoDispatch methods 17895
TAutoDispatch properties 17897
TAutoIncField class 6768
about TAutoIncField class 6768
Create 6775
TAutoIncField Members 6769
TAutoIntfObject class 5534
about TAutoIntfObject class 5534
Create 5536
DispIID 5539
DispIntfEntry 5539
DispTypeInfo 5539
GetIDsOfNames 5537
GetTypeInfo 5537
GetTypeInfoCount 5537
InterfaceSupportsErrorInfo 5538
Invoke 5538
SafeCallException 5538
TAutoIntfObject Members 5535
TAutoIntfObject methods 5537
TAutoIntfObject properties 5539
TAutomation class 17897
about TAutomation class 17897
AutoObjectCount 17899
Create 17898
Destroy 17898
IsInprocServer 17900
OnLastRelease 17900
RegisterClass 17899
StartMode 17900

TAutomation events 17900
TAutomation Members 17898
TAutomation methods 17899
TAutomation properties 17899
UpdateRegistry 17899
TAutoObject class 5540, 17901
about TAutoObject class 5540, 17901
AddRef 17903
AutoDispatch 17904
AutoFactory 5545
Create 17902
CreateAutoDispatch 17903
Destroy 17902
EventConnect 5542
EventSink 5545
EventSinkChanged 5543
GetExceptionInfo 17904
GetIDsOfNames 5543
GetTypeInfo 5543
GetTypeInfoCount 5544
Initialize 5544
Invoke 5544
OleObject 17904
RefCount 17905
Release 17904
TAutoObject Members 5541, 17901
TAutoObject methods 5542, 17903
TAutoObject properties 5545, 17904
TAutoObjectFactory class 5545
about TAutoObjectFactory class 5545
Create 5548
DispIntfEntry 5549
DispTypeInfo 5549
EventIID 5549
EventTypeInfo 5550
GetIntfEntry 5548
TAutoObjectFactory Members 5546
TAutoObjectFactory methods 5548
TAutoObjectFactory properties 5548
taVerticalCenter enumeration member 3280
TBaseAdapterAction class 24043
about TBaseAdapterAction class 24043
ActionName 24072
Adapter 24072
CheckExecuteAccess 24050
CreateRequestContext 24050
DisplayLabel 24073
DoAfterExecuteActionRequest 24051
DoAfterGetExecuteActionResponse 24051
DoBeforeExecuteActionRequest 24052
DoBeforeGetExecuteActionResponse 24052
EndIterator 24053
Errors 24073
ExecuteAccess 24073
ExecuteActionRequest 24053
GetActionName 24054
GetAdapter 24054
GetAdapterItemRequestParams 24055
GetDefaultActionName 24055
GetDefaultDisplayLabel 24055
GetDisplayLabel 24055
GetDisplayStyle 24056
GetDisplayStyleType 24056
GetErrors 24057
GetExecuteActionResponse 24057
GetFailureOptions 24057
GetInputStyle 24058
GetIsAdapterActionList 24058
GetItemCount 24058
GetIteratorIndex 24059
GetSuccessOptions 24059
GetVariableName 24060
GetViewMode 24060
GetViewModeType 24060
GetVisible 24061
GetWebSnapObjClass 24061
HandleRequest 24061
HasExecuteAccess 24062

ImplCreateRequestContext 24062
ImplEndIterator 24062
ImplExecuteActionRequest 24063
ImplGetActionName 24063
ImplGetAdapterItemRequestParams 24063
ImplGetDisplayLabel 24064
ImplGetExecuteActionResponse 24064
ImplGetFailureOptions 24065
ImplGetIsAdapterActionList 24065
ImplGetIteratorIndex 24065
ImplGetSuccessOptions 24066
ImplGetVariableName 24066
ImplGetVisible 24066
ImplHandleRequest 24067
ImplHasExecuteAccess 24067
ImplInIterator 24067
ImplIsDefaultAction 24068
ImplNextIteration 24068
ImplStartIterator 24068
ImplWebEnabled 24069
InIterator 24069
IsDefaultAction 24070
ItemCount 24074
ItemIndex 24074
NextIteration 24070
OnAfterExecute 24075
OnAfterGetResponse 24075
OnBeforeExecute 24076
OnBeforeGetResponse 24076
OnGetParams 24077
ParentChanged 24070
RedirectOptions 24074
SetDisplayLabel 24071
StartIterator 24071
TBaseAdapterAction events 24075
TBaseAdapterAction Members 24044
TBaseAdapterAction methods 24049
TBaseAdapterAction properties 24072
WebEnabled 24071
TBaseAdapterErrorsList class 24077
about TBaseAdapterErrorsList class 24077
AddError 24080
AddError method 24080
ClearErrors 24080
DefineLabel 24081
EndIterator 24081
ErrorCount 24085
Errors 24085
GetCurrentIntf 24081
GetError 24082
GetErrorCount 24082
GetObjectErrors 24082
HasObjectErrors 24082
ImplClearErrors 24083
ImplDefineLabel 24083
ImplGetError 24083
ImplGetErrorCount 24083
ImplGetObjectErrors 24084
ImplHasObjectErrors 24084
NextIteration 24084
StartIterator 24085
TBaseAdapterErrorsList Members 24078
TBaseAdapterErrorsList methods 24079
TBaseAdapterErrorsList properties 24085
TBaseAdapterMultiValueField class 24086
about TBaseAdapterMultiValueField class 24086
Create 24092
Destroy 24093
EndValueIterator 24093
GetCurrentValue 24094
GetFieldValuesAdapter 24094
GetValueCount 24094
GetValues 24095
HasValue 24095
ImplGetCurrentValue 24095
ImplGetFieldValuesAdapter 24096
ImplHasValue 24096
NextValueIteration 24096

StartValueIterator 24097
TBaseAdapterMultiValueField Members 24087
TBaseAdapterMultiValueField methods 24093
TBaseAdapterPageProducer class 19813
about TBaseAdapterPageProducer class 19813
CanAddClass 19820
ContentFromStream 19820
Create 19818
CreatePageElements 19821
DeclareStyles 19821
DeclareWarnings 19821
Destroy 19818
DoAfterGetContent 19822
DoBeforeGetContent 19822
DoTagEvent 19822
GeneratePageElements 19823
GetContentOptions 19823
GetDefaultTemplate 19824
GetRequiredTags 19824
GetScriptEngines 19825
GetTemplateFileType 19825
GetXMLData 19825
GetXSLData 19826
HandleScriptTag 19826
HasHTMLBrowserView 19827
HasScriptView 19827
HasXMLBrowserView 19827
HasXSLBrowserView 19828
ImplCanAddClass 19828
OnAfterGetContent 19832
OnBeforeGetContent 19832
PageElements 19830
SetStyles 19829
SetStylesFile 19829
SetWebContentOptions 19829
Styles 19830
StylesFile 19831
TBaseAdapterPageProducer events 19831
TBaseAdapterPageProducer Members 19814
TBaseAdapterPageProducer methods 19819
TBaseAdapterPageProducer properties 19830
TBaseAdapterStub class 1422
about TBaseAdapterStub class 1422
CreateComponentsStub 1424
FActions 1423
FFields 1424
FindObject 1424
Get_Actions 1424
Get_ClassName_ 1425
Get_Fields 1425
Get_Name_ 1425
TBaseAdapterStub fields 1423
TBaseAdapterStub Members 1423
TBaseAdapterStub methods 1424
TBaseArray class 17307
about TBaseArray class 17307
AutoSize 17318
BlockCopy 17311
Capacity 17319
Clear 17311
ClearFlag 17311
CompareProc 17319
Count 17320
Create 17309
Delete 17312
Destroy 17310
Duplicates 17320
Exchange 17312
FindItem 17312
GetAutoSize 17313
GetItem 17313
Grow 17313
HasFlag 17314
IndexOf 17314
Insert 17314
InsertAt 17315
InternalHandleException 17315
InvalidateItems 17315

ItemSize 17321
Limit 17321
List 17321
PutItem 17315
RemoveItem 17316
RemoveRange 17316
SetAutoSize 17316
SetCapacity 17317
SetFlag 17317
Sort 17317
SortOrder 17322
TBaseArray Members 17308
TBaseArray methods 17310
TBaseArray properties 17318
ValidateBounds 17317
ValidIndex 17318
TBaseButtonItem class 2417
about TBaseButtonItem class 2417
Action 2422
ActionChange 2420
ActionLink 2422
Assign 2420
Caption 2422
Create 2419
Data 2423
GetActionLinkClass 2421
GetDisplayName 2421
GetNotifyTarget 2421
Hint 2423
ImageIndex 2423
OnClick 2423
ScrollIntoView 2421
TBaseButtonItem events 2423
TBaseButtonItem Members 2417
TBaseButtonItem methods 2419
TBaseButtonItem properties 2422
TBaseComponentEditor class 11105
about TBaseComponentEditor class 11105
Create 11107
TBaseComponentEditor Members 11106
TBaseComponentGuidelines class 11107
about TBaseComponentGuidelines class 11107
Create 11109
Designer 11110
Initialize 11109
TBaseComponentGuidelines Members 11108
TBaseComponentGuidelines methods 11109
TBaseComponentGuidelines properties 11109
TBaseCustomModule class 11110
about TBaseCustomModule class 11110
Create 11111
DesignClass 11112
TBaseCustomModule Members 11110
TBaseCustomModule methods 11112
TBaseDataSetAdapterField class 7729
about TBaseDataSetAdapterField class 7729
Adapter 7752
CheckOrigValue 7736
CheckValueChange 7737
Create 7735
DataSetField 7752
FieldFlags 7752
FieldModes 7753
FindField 7737
FullyQualifyInputName 7738
GetAdapter 7738
GetDataSet 7738
GetDataSetFieldValue 7739
GetDisplayStyle 7739
GetDisplayStyleType 7740
GetInputStyle 7741
GetInputStyleType 7742
GetKeyName 7742
GetKeyParams 7743
GetReadOnly 7743
GetViewMode 7744
GetViewModeType 7744
ImplAsFormatted 7745

ImplCheckOrigValue 7745
ImplCheckValueChange 7746
ImplGetDisplayLabel 7747
ImplGetDisplayWidth 7747
ImplGetRequired 7747
ImplGetValue 7748
ImplGetVisible 7748
ImplSetFieldName 7749
ImplUpdateValue 7749
Notification 7750
OnGetDisplayText 7754
OnGetValue 7754
PrepareDataSet 7750
SetDataSetField 7751
TBaseDataSetAdapterField events 7753
TBaseDataSetAdapterField Members 7730
TBaseDataSetAdapterField methods 7735
TBaseDataSetAdapterField properties 7751
UpdateValue 7751
TBaseDataSetAdapterImageField class 7755
about TBaseDataSetAdapterImageField class 7755
TBaseDataSetAdapterInputField class 7756
about TBaseDataSetAdapterInputField class 7756
ImplGetEchoActionFieldValue 7763
ImplSetEchoActionFieldValue 7763
ImplValidateValue 7764
OnUpdateValue 7765
OnValidateValue 7766
TBaseDataSetAdapterInputField events 7766
TBaseDataSetAdapterInputField Members 7757
TBaseDataSetAdapterInputField methods 7762
TBaseDataSetAdapterInputField properties 7765
TestValueChange 7764
ValidateValue 7765
TBaseDragControlObject class 5719
about TBaseDragControlObject class 5719
Assign 5722
Control 5724
Create 5721
EndDrag 5723
Finished 5723
TBaseDragControlObject Members 5720
TBaseDragControlObject methods 5722
TBaseDragControlObject properties 5723
TBaseEndUserAdapter class 24097
about TBaseEndUserAdapter class 24097
CanExecuteLoginForm 24106
CanExecuteLogout 24107
CanProduceLoginPage 24107
DoOnHasRights 24107
ExecuteLogin 24108
ExecuteLoginForm 24108
ExecuteLogout 24108
GetDisplayName 24109
GetDisplayNameValue 24109
GetEndUserAdapter 24109
GetLoggedIn 24110
GetLoggedInValue 24110
GetLoginFormAction 24110
GetLogoutAction 24111
GetUserID 24111
HasExecuteRights 24111
HasModifyRights 24112
HasViewRights 24112
ImplCanAddActionClass 24113
ImplCanAddFieldClass 24113
ImplGetActionsList 24113
ImplGetDisplayName 24114
ImplGetEndUserAdapter 24114
ImplGetFieldsList 24114
ImplGetLoggedIn 24114
ImplGetLoginFormAction 24115
ImplGetLogoutAction 24115
ImplHasExecuteRights 24115
ImplHasModifyRights 24116
ImplHasViewRights 24116
ImplLogIn 24116
ImplProduceLoginPage 24117

LogIn 24117
LoginPage 24118
OnHasRights 24119
OnProduceLoginPage 24119
ProduceLoginPage 24117
TBaseEndUserAdapter events 24119
TBaseEndUserAdapter Members 24099
TBaseEndUserAdapter methods 24105
TBaseEndUserAdapter properties 24118
UserList 24118
TBaseInetXPageProducer class 17109
about TBaseInetXPageProducer class 17109
AddIncludes 17116
AddScriptComponents 17117
CanAddClass 17117
ContentFromStream 17117
Create 17115
CreatePageElements 17118
CreateScriptManager 17118
DeclareFunctions 17118
DeclareIncludes 17118
DeclareRowSets 17119
DeclareScript 17119
DeclareStyles 17119
DeclareWarnings 17119
DeclareXMLDocuments 17120
DeclareXMLIslands 17120
Destroy 17115
DoAfterGetContent 17120
DoAfterGetXMLData 17120
DoBeforeGetContent 17120
DoBeforeGetXMLData 17121
DoTagEvent 17121
EnableXMLIslands 17126
FindXMLBroker 17121
FindXMLBrokerName 17122
GeneratePageElements 17122
GetContentOptions 17122
GetDefaultTemplate 17122
GetRequiredTags 17123
GetScriptManager 17123
GetUseXMLIslands 17123
GetXMLData 17124
GetXMLDelta 17124
GetXMLErrors 17124
ImplCanAddClass 17124
IncludePathURL 17127
OnAfterGetContent 17129
OnAfterGetXMLData 17129
OnBeforeGetContent 17130
OnBeforeGetXMLData 17130
PageElements 17127
ScriptManager 17127
SetIncludePathURL 17125
SetStyles 17125
SetStylesFile 17125
SetWebContentOptions 17126
Styles 17127
StylesFile 17128
TBaseInetXPageProducer events 17129
TBaseInetXPageProducer Members 17110
TBaseInetXPageProducer methods 17116
TBaseInetXPageProducer properties 17126
TBasePageItemsProducer class 18376
about TBasePageItemsProducer class 18376
Create 18380
Destroy 18381
GetChildren 18381
GetComponentList 18382
GetDefaultComponentList 18382
SetChildOrder 18382
TBasePageItemsProducer Members 18377
TBasePageItemsProducer methods 18381
TBasePageItemsProducer properties 18382
WebPageItems 18382
TBasePageProducer class 15269
about TBasePageProducer class 15269
Content 15274

ContentFromStream 15275
ContentFromString 15275
Create 15274
DoTagEvent 15276
GetProducerTemplateFileName 15276
GetProducerTemplateStream 15276
GetScriptEngine 15277
GetTagID 15277
GetTemplateFileName 15277
GetTemplateStream 15277
HandleScriptTag 15277
HandleTag 15278
ImplHandleTag 15278
OnHTMLTag 15280
ScriptEngine 15279
ServerScriptFromStream 15278
StripParamQuotes 15279
TBasePageProducer events 15280
TBasePageProducer Members 15270
TBasePageProducer methods 15274
TBasePageProducer properties 15279
UseScriptEngine 15278
WebModuleContext 15279
TBasePropertyEditor class 11112
about TBasePropertyEditor class 11112
Create 11114
Initialize 11114
SetPropEntry 11115
TBasePropertyEditor Members 11113
TBasePropertyEditor methods 11114
TBaseProvider class 18423
about TBaseProvider class 18423
AfterUpdateRecord 18442
BeforeUpdateRecord 18442
CheckResolver 18433
Create 18429
CreateDataPacket 18433
CreateResolver 18434
Destroy 18430
DoAfterUpdateRecord 18434
DoBeforeCommit 18435
DoBeforeUpdateRecord 18435
DoOnGetData 18435
DoOnUpdateData 18435
FAfterUpdateRecord 18430
FBeforeCommit 18430
FBeforeUpdateRecord 18431
FDataDS 18431
FetchDetails 18436
FOnGetData 18431
FOnUpdateData 18431
FOnUpdateError 18432
FProviderOptions 18432
FreeResolver 18436
FResolver 18432
FUpdateMode 18432
InternalApplyUpdates 18436
InternalGetRecords 18437
InternalRowRequest 18438
LocateRecord 18439
OnDataRequest 18442
OnGetData 18443
OnUpdateData 18443
OnUpdateError 18444
Options 18440
PacketDataSet 18440
Resolver 18441
TBaseProvider events 18442
TBaseProvider fields 18430
TBaseProvider Members 18425
TBaseProvider methods 18432
TBaseProvider properties 18439
UpdateMode 18441
UpdateRecord 18439
TBaseRecordsAdapterStub class 1425
about TBaseRecordsAdapterStub class 1425
CreateRecordsStub 1427
Get_Records 1427

TBaseRecordsAdapterStub Members 1426
TBaseRecordsAdapterStub methods 1426
TBaseRequestHandler class 24860
about TBaseRequestHandler class 24860
AppDispatcher 24865
HandleRequest 24864
ImplHandleRequest 24864
SetAppDispatcher 24864
TBaseRequestHandler Members 24861
TBaseRequestHandler methods 24863
TBaseRequestHandler properties 24865
TBaseSelectionEditor class 11116
about TBaseSelectionEditor class 11116
Create 11117
TBaseSelectionEditor Members 11116
TBaseSocket class 20042
about TBaseSocket class 20042
Active 20054
BlockMode 20054
BytesReceived 20055
BytesSent 20055
Close 20047
Create 20046
Destroy 20046
DoCreateHandle 20047
DoDestroyHandle 20047
DoHandleError 20048
Domain 20055
DoReceive 20048
DoSend 20048
ErrorCheck 20048
Handle 20056
Loaded 20049
MapDomain 20049
MapSockType 20050
OnCreateHandle 20057
OnDestroyHandle 20057
OnError 20058
OnReceive 20058
OnSend 20058
Open 20050
PeekBuf 20050
Protocol 20056
ReceiveBuf 20051
Receiveln 20051
Select 20051
SendBuf 20052
Sendln 20052
SendStream 20053
SetBytesReceived 20053
SetBytesSent 20053
SockType 20056
TBaseSocket events 20057
TBaseSocket Members 20042
TBaseSocket methods 20046
TBaseSocket properties 20054
WaitForData 20053
TBaseValuesListAdapter class 24120
about TBaseValuesListAdapter class 24120
GetListImage 24128
GetListImageOfValue 24128
GetListName 24128
GetListNameOfValue 24128
GetListValue 24129
GetNameFieldClass 24129
GetValueFieldClass 24129
GetWebSnapObjClass 24130
ImplCanAddFieldClass 24130
ImplGetFieldsList 24130
ImplGetListImage 24131
ImplGetListImageOfValue 24131
ImplGetListName 24131
ImplGetListNameOfValue 24132
ImplGetListValue 24132
TBaseValuesListAdapter Members 24121
TBaseValuesListAdapter methods 24127
TBaseValuesListField class 24132
about TBaseValuesListField class 24132

Adapter 24139
GetAdapter 24138
TBaseValuesListField Members 24133
TBaseValuesListField methods 24138
TBaseValuesListField properties 24138
TBaseWebModuleFactory class 24994
about TBaseWebModuleFactory class 24994
Create 24995
GetCacheMode 24996
GetComponentClass 24997
GetCreateMode 24997
GetIsAppModule 24997
GetModule 24998
GetModuleName 24998
PreventDestruction 24998
TBaseWebModuleFactory Members 24995
TBaseWebModuleFactory methods 24996
TBaseWebPageInfo class 24999
about TBaseWebPageInfo class 24999
Create 25000
GetIsLoginRequired 25001
GetIsPublished 25002
GetPageDescription 25002
GetPageFile 25002
GetPageHREF 25003
GetPageName 25003
GetPageTitle 25003
GetViewAccess 25004
SetFactory 25004
TBaseWebPageInfo Members 25000
TBaseWebPageInfo methods 25001
TBaseWebPageModuleFactory class 25004
about TBaseWebPageModuleFactory class 25004
Create 25006
GetCacheMode 25007
GetComponentClass 25007
GetCreateMode 25007
GetIsAppModule 25008
GetModule 25008
GetModuleName 25009
PreventDestruction 25009
TBaseWebPageModuleFactory Members 25005
TBaseWebPageModuleFactory methods 25006
TBasicAction class 2707
about TBasicAction class 2707
ActionComponent 2715
Change 2712
Create 2710
Destroy 2711
Execute 2712
ExecuteTarget 2712
FClients 2711
HandlesTarget 2713
Notification 2713
OnChange 2716
OnExecute 2716
OnUpdate 2718
RegisterChanges 2713
SetOnExecute 2714
TBasicAction events 2716
TBasicAction fields 2711
TBasicAction Members 2707
TBasicAction methods 2711
TBasicAction properties 2715
UnRegisterChanges 2714
Update 2714
UpdateTarget 2715
TBasicActionLink class 2720
about TBasicActionLink class 2720
Action 2725
AssignClient 2723
Change 2723
Create 2721
Destroy 2722
Execute 2723
FAction 2722
IsOnExecuteLinked 2723
OnChange 2725

SetAction 2724
SetOnExecute 2724
TBasicActionLink events 2725
TBasicActionLink fields 2722
TBasicActionLink Members 2720
TBasicActionLink methods 2722
TBasicActionLink properties 2724
Update 2724
TBasicActionRequestImpl class 884
about TBasicActionRequestImpl class 884
ActionName 889
ActionParams 889
Create 886
Destroy 886
DispatchParams 889
FailurePage 889
GetActionName 887
GetFailurePage 887
GetHandled 888
GetSuccessPage 888
SetHandled 888
SuccessPage 890
TBasicActionRequestImpl Members 884
TBasicActionRequestImpl methods 887
TBasicActionRequestImpl properties 888
TBasicActionResponseImpl class 890
about TBasicActionResponseImpl class 890
Create 892
ExecuteStatus 896
FailurePage 896
GetExecuteStatus 892
GetHandled 893
GetRedirectOptions 893
GetRespondWith 893
Handled 896
HandleResponse 894
RedirectOptions 897
RespondWith 897
RespondWithComponentPage 894
RespondWithFailurePage 894
RespondWithPage 894
RespondWithSuccessPage 894
RespondWithURL 895
SetExecuteStatus 895
SetHandled 895
SetRedirectOptions 895
SuccessPage 897
TBasicActionResponseImpl Members 890
TBasicActionResponseImpl methods 892
TBasicActionResponseImpl properties 896
TBasicImageRequestImpl class 898
about TBasicImageRequestImpl class 898
ActionRequestParams 901
Create 900
Destroy 900
DispatchParams 901
GetImageName 901
ImageName 902
TBasicImageRequestImpl Members 898
TBasicImageRequestImpl methods 900
TBasicImageRequestImpl properties 901
TBasicImageResponseImpl class 902
about TBasicImageResponseImpl class 902
Create 903
TBasicImageResponseImpl Members 902
TBatchMove class 9858
AbortOnKeyViol 9865
AbortOnProblem 9866
about TBatchMove class 9858
ChangedCount 9866
ChangedTableName 9867
CommitCount 9867
Create 9862
Destination 9868
Destroy 9863
Execute 9863
KeyViolCount 9869
KeyViolTableName 9869

Mappings 9870
Mode 9871
MovedCount 9872
Notification 9864
ProblemCount 9872
ProblemTableName 9873
RecordCount 9873
Source 9874
TBatchMove Members 9859
TBatchMove methods 9863
TBatchMove properties 9864
Transliterate 9875
TBatchMoveSprig class 1731
about TBatchMoveSprig class 1731
AnyProblems 1732
Caption 1732
TBatchMoveSprig Members 1731
TBatchMoveSprig methods 1732
TBCDField class 6776
about TBCDField class 6776
CheckTypeSize 6784
CopyData 6784
Create 6782
currency 6788
GetAsBCD 6784
GetAsCurrency 6784
GetAsFloat 6785
GetAsInteger 6785
GetAsString 6785
GetAsVariant 6785
GetDataSize 6785
GetDefaultWidth 6786
GetText 6786
GetValue 6786
MaxValue 6788
MinValue 6789
Precision 6789
SetAsBCD 6786
SetAsCurrency 6787
SetAsFloat 6787
SetAsInteger 6787
SetAsString 6787
SetVarValue 6787
Size 6789
TBCDField Members 6777
TBCDField methods 6783
TBCDField properties 6788
Value 6790
TBDECallback class 9875
about TBDECallback class 9875
Create 9876
Destroy 9877
Invoke 9878
TBDECallback Members 9876
TBDECallback methods 9878
TBDEDataSet class 9878
about TBDEDataSet class 9878
ActivateFilters 9893
Active 9929
AddFieldDesc 9893
AfterCancel 9938
AfterClose 9940
AfterDelete 9942
AfterEdit 9944
AfterInsert 9945
AfterOpen 9946
AfterPost 9947
AfterRefresh 9947
AfterScroll 9948
AllocCachedUpdateBuffers 9893
AllocKeyBuffers 9893
AllocRecordBuffer 9894
ApplyUpdates 9894
AutoCalcFields 9931
BeforeCancel 9948
BeforeClose 9949
BeforeDelete 9949
BeforeEdit 9950

wwww

BeforeInsert 9950
BeforeOpen 9952
BeforePost 9952
BeforeRefresh 9953
BeforeScroll 9953
BlockReadNext 9894
BookmarkValid 9895
CacheBlobs 9932
CachedUpdateCallBack 9895
CachedUpdates 9932
Cancel 9895
CancelUpdates 9896
CheckCachedUpdateMode 9896
CheckSetKeyMode 9896
ClearCalcFields 9897
CloseBlob 9897
CloseCursor 9897
CommitUpdates 9897
CompareBookmarks 9898
ConstraintCallBack 9898
ConstraintsDisabled 9899
Create 9889
CreateBlobStream 9899
CreateExprFilter 9900
CreateFuncFilter 9900
CreateHandle 9900
CreateLookupFilter 9901
DataEvent 9901
DeactivateFilters 9901
Destroy 9890
DestroyHandle 9901
DestroyLookupCursor 9902
DisableConstraints 9902
EnableConstraints 9902
ExpIndex 9933
FetchAll 9903
FindRecord 9903
FlushBuffers 9903
ForceUpdateCallback 9904
FreeKeyBuffers 9904
FreeRecordBuffer 9904
GetBlobFieldData 9904
GetBookmarkData 9905
GetBookmarkFlag 9905
GetCanModify 9905
GetCurrentRecord 9905
GetFieldData 9906
GetFieldData method 9906
GetFieldFullName 9907
GetHandle 9907
GetIndexField 9907
GetIndexFieldCount 9907
GetIndexInfo 9907
GetIsIndexField 9908
GetKeyBuffer 9908
GetKeyExclusive 9908
GetKeyFieldCount 9909
GetLookupCursor 9909
GetObjectTypeNames 9909
GetRecNo 9909
GetRecord 9910
GetRecordCount 9910
GetRecordSize 9910
GetStateFieldValue 9910
GetUpdateRecordSet 9910
GetUpdatesPending 9911
Handle 9933
InitKeyBuffer 9911
InitRecord 9911
InternalAddRecord 9911
InternalCancel 9912
InternalClose 9912
InternalDelete 9912
InternalEdit 9912
InternalFirst 9912
InternalGotoBookmark 9913
InternalHandleException 9913
InternalInitFieldDefs 9913

InternalInitRecord 9913
InternalInsert 9914
InternalLast 9914
InternalOpen 9914
InternalPost 9914
InternalRefresh 9914
InternalSetToRecord 9915
IsCursorOpen 9915
IsSequenced 9915
KeySize 9934
Locale 9934
Locate 9915
LocateRecord 9916
Lookup 9916
MapsToIndex 9918
ObjectView 9935
OnCalcFields 9954
OnDeleteError 9955
OnEditError 9955
OnNewRecord 9955
OnPostError 9956
OnUpdateError 9956
OnUpdateRecord 9957
OpenCursor 9918
Post 9918
PostKeyBuffer 9919
PrepareCursor 9919
ProcessUpdates 9919
PSGetUpdateException 9920
PSIsSQLSupported 9920
PSReset 9921
PSUpdateRecord 9921
ResetCursorRange 9922
RevertRecord 9922
SetBlockReadSize 9922
SetBookmarkData 9922
SetBookmarkFlag 9923
SetCachedUpdates 9923
SetCursorRange 9923
SetFieldData 9923
SetFilterData 9923
SetFiltered 9924
SetFilterHandle 9924
SetFilterOptions 9924
SetFilterText 9924
SetIndexField 9925
SetKeyBuffer 9925
SetKeyExclusive 9925
SetKeyFieldCount 9925
SetKeyFields 9925
SetLinkRanges 9926
SetLocale 9926
SetOnFilterRecord 9926
SetOnUpdateError 9926
SetRecNo 9927
SetStateFieldValue 9927
SetupCallBack 9927
SetUpdateObject 9927
SetUpdateRecordSet 9927
StmtHandle 9935
SwitchToIndex 9928
TBDEDataSet events 9938
TBDEDataSet Members 9879
TBDEDataSet methods 9890
TBDEDataSet properties 9929
Translate 9928
UpdateCallbackRequired 9928
UpdateObject 9936
UpdateRecordTypes 9936
UpdatesPending 9937
UpdateStatus 9928
TBDEDataSetSprig class 1732
about TBDEDataSetSprig class 1732
GetDSDesignerClass 1733
TBDEDataSetSprig Members 1733
TBDEDataSetSprig methods 1733
TBevel class 12105
about TBevel class 12105

Align 12114
Anchors 12117
Constraints 12118
Create 12113
Paint 12113
ParentShowHint 12118
Shape 12119
ShowHint 12119
Style 12121
TBevel Members 12106
TBevel methods 12113
TBevel properties 12113
Visible 12121
tbHorizontal enumeration member 14367
TBinaryField class 6791
about TBinaryField class 6791
CheckTypeSize 6798
CopyData 6798
Create 6797
GetAsString 6799
GetAsVariant 6799
GetText 6799
SetAsString 6799
SetText 6799
SetVarValue 6800
Size 6800
TBinaryField Members 6791
TBinaryField methods 6798
TBinaryField properties 6800
TBinData class 17397
about TBinData class 17397
AddBin 17400
AddBinValue 17400
AddBinValues 17400
AddIBinValue 17401
Assign 17401
BinCount 17404
BinName 17404
BinValueCount 17402
Clear 17402
create 17399
destroy 17399
GetAllBinValueCount 17402
GetBinCount 17402
GetBinName 17402
GetBinNameDataType 17403
GetBinValue 17403
GetBinValues 17403
GetIBinValue 17403
GetIBinValueCount 17404
GetMaxBinNameSize 17404
NameList 17405
OtherBinName 17405
TBinData Members 17398
TBinData methods 17399
TBinData properties 17404
ValueList 17405
TBinding class 25640
about TBinding class 25640
AfterConstruction 25646
GetBindingOperations 25646
GetName 25646
GetType_ 25647
SetName 25647
SetType_ 25647
TBinding Members 25641
TBinding methods 25645
TBindingOperation class 25647
about TBindingOperation class 25647
AfterConstruction 25653
GetFaults 25653
GetInput 25653
GetName 25654
GetOutput 25654
SetName 25654
TBindingOperation Members 25648
TBindingOperation methods 25652
TBindingOperations class 25654

about TBindingOperations class 25654
Add 25660
GetBindingOperation 25660
Insert 25660
TBindingOperations Members 25655
TBindingOperations methods 25660
TBindings class 25661
about TBindings class 25661
Add 25666
GetBinding 25667
Insert 25667
TBindings Members 25661
TBindings methods 25666
TBitBtn class 2290
about TBitBtn class 2290
Action 2307
ActionChange 2303
Align 2307
Anchors 2310
BiDiMode 2311
Cancel 2312
Caption 2312
Click 2303
Constraints 2315
CopyImage 2304
Create 2302
CreateHandle 2304
CreateParams 2304
Default 2315
Destroy 2302
Enabled 2316
GetActionLinkClass 2305
GetPalette 2305
Glyph 2317
Kind 2317
Layout 2318
Margin 2318
ModalResult 2319
NumGlyphs 2320
OnEnter 2326
OnExit 2328
ParentBiDiMode 2320
ParentShowHint 2320
SetButtonStyle 2306
ShowHint 2321
Spacing 2322
Style 2323
TabOrder 2323
TabStop 2324
TBitBtn events 2326
TBitBtn Members 2291
TBitBtn methods 2302
TBitBtn properties 2306
Visible 2324
TBitBtnActionLink class 2330
about TBitBtnActionLink class 2330
AssignClient 2334
Create 2332
FClient 2333
FImageIndex 2333
IsImageIndexLinked 2334
SetImageIndex 2335
TBitBtnActionLink fields 2333
TBitBtnActionLink Members 2330
TBitBtnActionLink methods 2334
TBitmap class 14396
about TBitmap class 14396
Assign 14402
Canvas 14419
Changed 14403
Create 14400
Destroy 14401
Dormant 14404
Draw 14405
FreeImage 14405
GetEmpty 14407
GetHeight 14407
GetPalette 14407

GetWidth 14407
Handle 14420
HandleAllocated 14407
HandleNeeded 14408
HandleType 14420
IgnorePalette 14421
LoadFromClipboardFormat 14408
LoadFromResourceID 14410
LoadFromResourceName 14410
LoadFromStream 14412
Mask 14412
MaskHandle 14421
MaskHandleNeeded 14412
Monochrome 14421
PaletteNeeded 14413
PixelFormat 14423
ReadData 14413
ReleaseHandle 14413
ReleaseMaskHandle 14414
ReleasePalette 14415
SaveToClipboardFormat 14415
SaveToStream 14417
ScanLine 14425
SetHeight 14417
SetPalette 14418
SetSize 14418
SetWidth 14418
TBitmap Members 14397
TBitmap methods 14401
TBitmap properties 14418
TransparentColor 14427
TransparentMode 14428
WriteData 14418
TBitmapImage class 14429
about TBitmapImage class 14429
Destroy 14430
FreeHandle 14430
TBitmapImage Members 14429
TBitmapImage methods 14430
TBits class 2725
about TBits class 2725
Bits 2727
Destroy 2726
OpenBit 2727
Size 2728
TBits Members 2726
TBits methods 2727
TBits properties 2727
tbLeftButton enumeration member 16201
TBlobField class 6801
about TBlobField class 6801
Assign 6809
AssignTo 6810
BlobSize 6819
BlobType 6819
Clear 6810
Create 6808
FreeBuffers 6811
GetAsString 6811
GetAsVariant 6811
GetAsWideString 6812
GetBlobSize 6812
GetClassDesc 6812
GetIsNull 6812
GetText 6812
GraphicHeader 6820
IsBlob 6813
LoadFromFile 6814
LoadFromStream 6815
Modified 6820
SaveToFile 6816
SaveToStream 6816
SetAsString 6817
SetAsWideString 6817
SetFieldType 6817
SetText 6818
SetVarValue 6818
Size 6820

TBlobField Members 6802
TBlobField methods 6809
TBlobField properties 6818
Transliterate 6821
Value 6822
TBlobStream class 9958
about TBlobStream class 9958
Create 9960
Destroy 9961
Read 9962
Seek 9962
TBlobStream Members 9959
TBlobStream methods 9962
Truncate 9963
Write 9963
TBOA class 6431
about TBOA class 6431
Deactivate 6431
GetPrincipal 6432
ImplIsReady 6432
Initialize 6432
ObjIsReady 6432
TBOA Members 6431
TBOA methods 6431
TBookmarkList class 9575
about TBookmarkList class 9575
Clear 9577
Compare 9578
Count 9580
Create 9576
CurrentRow 9578
CurrentRowSelected 9581
Delete 9578
Destroy 9576
Find 9578
IndexOf 9579
Items 9581
LinkActive 9579
Refresh 9580
TBookmarkList Members 9575
TBookmarkList methods 9577
TBookmarkList properties 9580
TBooleanField class 6822
about TBooleanField class 6822
Create 6828
DisplayValues 6832
GetAsBoolean 6830
GetAsString 6830
GetAsVariant 6830
GetDataSize 6830
GetDefaultWidth 6831
SetAsBoolean 6831
SetAsString 6831
SetVarValue 6831
TBooleanField Members 6823
TBooleanField methods 6829
TBooleanField properties 6831
Value 6833
TBoolProperty class 10918
about TBoolProperty class 10918
TBoundLabel class 12123
about TBoundLabel class 12123
AdjustBounds 12131
BiDiMode 12132
Caption 12133
Color 12135
Create 12131
DragCursor 12136
DragKind 12137
DragMode 12137
Font 12139
Height 12140
Layout 12141
Left 12141
OnClick 12154
OnContextPopup 12156
OnDblClick 12157
OnDragDrop 12157

OnDragOver 12159
OnEndDock 12162
OnEndDrag 12162
OnMouseActivate 12163
OnMouseDown 12164
OnMouseMove 12166
OnMouseUp 12168
OnStartDock 12171
OnStartDrag 12171
ParentBiDiMode 12142
ParentColor 12142
ParentFont 12143
ParentShowHint 12143
PopupMenu 12144
ShowAccelChar 12148
ShowHint 12149
TBoundLabel events 12153
TBoundLabel Members 12123
TBoundLabel methods 12131
TBoundLabel properties 12132
Top 12151
Transparent 12152
Width 12153
WordWrap 12153
tbRightButton enumeration member 16201
TBrowseForFolder class 20508
about TBrowseForFolder class 20508
BeforeExecute 20519
BrowseOptions 20514
Caption 20514
DialogCaption 20515
DoCallback 20513
DoValidate 20513
Enabled 20515
ExecuteTarget 20513
Folder 20515
GetDialogClass 20514
HelpContext 20516
HelpKeyword 20516
HelpType 20517
Hint 20517
ImageIndex 20517
OnAccept 20520
OnCallback 20520
OnCancel 20520
OnHint 20520
OnValidate 20521
RootDir 20518
SecondaryShortCuts 20518
ShortCut 20518
TBrowseForFolder events 20519
TBrowseForFolder Members 20508
TBrowseForFolder methods 20512
TBrowseForFolder properties 20514
Visible 20519
TBrowseURL class 11832
about TBrowseURL class 11832
AfterBrowse 11842
BeforeBrowse 11842
Caption 11837
Enabled 11838
ExecuteTarget 11836
HelpContext 11838
HelpKeyword 11838
HelpType 11839
Hint 11839
ImageIndex 11840
OnHint 11842
OnUpdate 11843
SecondaryShortCuts 11840
ShortCut 11840
TBrowseURL events 11841
TBrowseURL Members 11832
TBrowseURL methods 11836
TBrowseURL properties 11837
URL 11841
Visible 11841
TBrush class 14430

about TBrush class 14430
Assign 14433
Bitmap 14436
Color 14437
Create 14432
Destroy 14433
GetBitmap 14434
GetColor 14434
GetHandle 14434
GetStyle 14434
Handle 14438
SetBitmap 14434
SetColor 14435
SetHandle 14435
SetStyle 14435
Style 14438
TBrush Members 14431
TBrush methods 14433
TBrush properties 14435
TBrushRecall class 14439
about TBrushRecall class 14439
Create 14440
TBrushRecall Members 14439
tbsButton enumeration member 5470
tbsCheck enumeration member 5470
tbsChecked enumeration member 5470
tbsDivider enumeration member 5470
tbsDropDown enumeration member 5470
tbsEllipses enumeration member 5470
tbsEnabled enumeration member 5470
tbsHidden enumeration member 5470
tbsIndeterminate enumeration member 5470
tbsMarked enumeration member 5470
tbsPressed enumeration member 5470
tbsSeparator enumeration member 5470
tbsTextButton enumeration member 5470
tbsWrap enumeration member 5470
TBucketList class 5643
about TBucketList class 5643
BucketFor 5645
Create 5645
TBucketList Members 5644
TBucketList methods 5645
TBuilderDim class 17749
about TBuilderDim class 17749
Add 17751
AddSummary 17751
Create 17750
Destroy 17750
GetLastVal 17751
GetSumCount 17752
GetSummary 17752
GroupBreak 17753
InitSummary 17752
LastVal 17753
MatchLastVal 17752
SetLastVal 17753
SumCount 17753
TBuilderDim Members 17749
TBuilderDim methods 17751
TBuilderDim properties 17753
TButton class 20909
about TButton class 20909
Action 20924
Align 20924
Anchors 20927
BiDiMode 20928
Cancel 20929
Caption 20929
Click 20921
Constraints 20932
Create 20920
CreateParams 20921
CreateWnd 20922
Default 20932
DragCursor 20933
DragKind 20934
DragMode 20934

eeee

Enabled 20936
Font 20936
ModalResult 20937
OnClick 20949
OnContextPopup 20951
OnDragDrop 20951
OnDragOver 20954
OnEndDock 20956
OnEndDrag 20956
OnEnter 20957
OnExit 20960
OnKeyDown 20961
OnKeyPress 20964
OnKeyUp 20965
OnMouseActivate 20967
OnMouseDown 20968
OnMouseEnter 20970
OnMouseLeave 20970
OnMouseMove 20971
OnMouseUp 20973
OnStartDock 20975
OnStartDrag 20976
ParentBiDiMode 20938
ParentFont 20938
ParentShowHint 20939
PopupMenu 20939
SetButtonStyle 20922
ShowHint 20944
TabOrder 20945
TabStop 20946
TButton events 20948
TButton Members 20910
TButton methods 20921
TButton properties 20923
UseRightToLeftAlignment 20923
Visible 20946
WordWrap 20948
TButtonActionLink class 20976
about TButtonActionLink class 20976
AssignClient 20980
FClient 20979
IsCheckedLinked 20980
SetChecked 20980
TButtonActionLink fields 20979
TButtonActionLink Members 20977
TButtonActionLink methods 20979
TButtonCategories class 2424
about TButtonCategories class 2424
Add 2426
AddItem 2426
BeginUpdate 2427
ButtonGroup 2428
Create 2426
GetOwner 2427
IndexOf 2427
Insert 2428
Items 2429
TButtonCategories Members 2424
TButtonCategories methods 2426
TButtonCategories properties 2428
Update 2428
TButtonCategory class 2429
about TButtonCategory class 2429
Assign 2432
Caption 2434
Categories 2434
Collapsed 2434
Color 2434
Create 2431
Data 2434
Destroy 2431
GradientColor 2435
IndexOf 2432
InterfaceData 2435
Items 2435
ScrollIntoView 2433
SetIndex 2433
TButtonCategory Members 2429

TButtonCategory methods 2431
TButtonCategory properties 2433
TextColor 2435
TButtonCollection class 2436
about TButtonCollection class 2436
Add 2438
AddItem 2438
Category 2440
Create 2438
GetOwner 2438
Insert 2439
Items 2440
Notify 2439
TButtonCollection Members 2436
TButtonCollection methods 2438
TButtonCollection properties 2440
Update 2440
TButtonControl class 20981
about TButtonControl class 20981
ActionChange 20992
Checked 20995
ClicksDisabled 20996
Create 20992
CreateParams 20993
GetActionLinkClass 20993
GetChecked 20993
SetChecked 20994
TButtonControl Members 20981
TButtonControl methods 20992
TButtonControl properties 20994
WndProc 20994
WordWrap 20996
TButtonGroup class 2198
about TButtonGroup class 2198
Align 2221
Anchors 2225
Assign 2210
BevelEdges 2225
BevelInner 2226
BevelKind 2226
BevelOuter 2227
BevelWidth 2227
BorderStyle 2227
BorderWidth 2228
ButtonHeight 2228
ButtonOptions 2228
ButtonWidth 2229
Canvas 2229
Create 2209
CreateButton 2211
CreateHandle 2211
CreateParams 2211
Destroy 2209
DockSite 2231
DoEndDrag 2212
DoHotButton 2212
DoItemClicked 2212
DoMouseWheelDown 2213
DoMouseWheelUp 2213
DoReorderButton 2214
DoStartDrag 2214
DragCursor 2232
DragDrop 2214
DragImageList 2232
DragIndex 2233
DragKind 2233
DragMode 2233
DragOver 2215
DrawButton 2215
Enabled 2235
Font 2235
GetButtonClass 2215
GetButtonRect 2215
GetButtonsClass 2216
GetDragImages 2216
Height 2237
Images 2238
IndexOfButtonAt 2216

gggggg

ItemIndex 2238
Items 2238
MouseDown 2216
MouseMove 2217
MouseUp 2217
Notification 2218
OnAfterDrawButton 2248
OnAlignInsertBefore 2248
OnAlignPosition 2249
OnBeforeDrawButton 2249
OnButtonClicked 2249
OnClick 2249
OnContextPopup 2251
OnDockDrop 2251
OnDockOver 2251
OnDragDrop 2252
OnDragOver 2254
OnDrawButton 2256
OnDrawIcon 2256
OnEndDock 2257
OnEndDrag 2257
OnEnter 2258
OnExit 2258
OnHotButton 2258
OnKeyDown 2259
OnKeyPress 2261
OnKeyUp 2262
OnMouseActivate 2264
OnMouseDown 2265
OnMouseEnter 2267
OnMouseLeave 2268
OnMouseMove 2268
OnMouseUp 2270
OnMouseWheel 2273
OnMouseWheelDown 2273
OnMouseWheelUp 2273
OnReorderButton 2274
OnStartDock 2274
OnStartDrag 2274
Paint 2218
PopupMenu 2238
RemoveInsertionPoints 2219
Resize 2219
ScrollIntoView 2219
SetInsertionPoints 2220
ShowHint 2243
TabOrder 2244
TabStop 2245
TargetIndexAt 2220
TButtonGroup events 2247
TButtonGroup Members 2198
TButtonGroup methods 2210
TButtonGroup properties 2220
UpdateAllButtons 2220
UpdateButton 2220
Visible 2245
Width 2247
TButtonItem class 2441
about TButtonItem class 2441
Assign 2443
ButtonGroup 2444
Category 2445
GetNotifyTarget 2444
InterfaceData 2445
ScrollIntoView 2444
TButtonItem Members 2441
TButtonItem methods 2443
TButtonItem properties 2444
TButtonItemActionLink class 2445
about TButtonItemActionLink class 2445
AssignClient 2448
DoShowHint 2448
FClient 2447
IsCaptionLinked 2448
IsHintLinked 2449
IsImageIndexLinked 2449
IsOnExecuteLinked 2450
SetCaption 2450

SetHint 2450
SetImageIndex 2451
SetOnExecute 2452
TButtonItemActionLink fields 2447
TButtonItemActionLink Members 2445
TButtonItemActionLink methods 2447
tbVertical enumeration member 14367
TBytesField class 6833
about TBytesField class 6833
Create 6840
GetDataSize 6841
TBytesField Members 6834
TBytesField methods 6841
TCanvas class 14441
about TCanvas class 14441
Arc 14448
Brush 14484
BrushCopy 14449
CanvasOrientation 14485
Changed 14450
Changing 14450
Chord 14451
ClipRect 14486
CopyMode 14487
CopyRect 14452
Create 14443
CreateHandle 14454
Destroy 14446
Draw 14454
DrawFocusRect 14455
Ellipse 14457
Ellipse method 14457
FillRect 14458
FloodFill 14459
Font 14488
FrameRect 14460
Handle 14489
HandleAllocated 14461
LineTo 14462
Lock 14463
LockCount 14492
MoveTo 14463
OnChange 14496
OnChanging 14496
Pen 14492
PenPos 14494
Pie 14464
Pixels 14494
PolyBezier 14464
PolyBezierTo 14465
Polygon 14466
Polyline 14467
Rectangle 14468
Rectangle method 14468
Refresh 14470
RequiredState 14470
RoundRect 14471
StretchDraw 14472
TCanvas events 14496
TCanvas Members 14441
TCanvas methods 14447
TCanvas properties 14484
TextExtent 14474
TextFlags 14495
TextHeight 14474
TextOut 14475
TextRect 14480
TextRect method 14480
TextWidth 14481
TryLock 14483
Unlock 14483
TCaptionAttributes class 16489
about TCaptionAttributes class 16489
Align 16491
Create 16490
Custom 16491
Parent 16492
Style 16492

StyleRule 16492
TCaptionAttributes Members 16489
TCaptionAttributes properties 16491
VAlign 16493
TCaptionedDockTree class 2398
about TCaptionedDockTree class 2398
AdjustCaptionRect 2402
AdjustDockRect 2402
Create 2401
Create constructor 2400
Destroy 2401
DockCaptionDrawer 2405
DockCaptionOrientation 2405
GetDockCaptionDrawer 2403
GetParentFormState 2403
InternalCaptionHitTest 2403
PaintDockFrame 2404
TCaptionedDockTree Members 2398
TCaptionedDockTree methods 2402
TCaptionedDockTree properties 2405
WndProc 2404
ZoneCaptionHitTest 2404
TCaptionedTabDockTree class 11594
about TCaptionedTabDockTree class 11594
MouseUp 11597
SetAlternateDockHost 11597
TCaptionedTabDockTree Members 11595
TCaptionedTabDockTree methods 11597
TCategoryButtons class 2452
about TCategoryButtons class 2452
Align 2480
Anchors 2480
Assign 2465
BackgroundGradientColor 2481
BackgroundGradientDirection 2481
BeginAutoDrag 2466
BevelEdges 2481
BevelInner 2482
BevelKind 2482
BevelOuter 2483
BevelWidth 2483
BorderStyle 2484
BorderWidth 2484
ButtonFlow 2484
ButtonHeight 2485
ButtonOptions 2485
ButtonWidth 2486
Categories 2486
CMFontchanged 2466
Color 2486
Create 2464
CreateHandle 2466
CreateParams 2467
Cursor 2486
Destroy 2464
DoBeginDrag 2467
DockSite 2489
DoCopyButton 2468
DoEndDrag 2468
DoHotButton 2468
DoItemClicked 2469
DoMouseWheelDown 2469
DoMouseWheelUp 2469
DoReorderButton 2470
DoReorderCategory 2470
DoSelectedItemChanged 2470
DoStartDrag 2470
DragButton 2490
DragCategory 2490
DragCursor 2491
DragDrop 2471
DragImageList 2491
DragKind 2492
DragMode 2492
DragOver 2471
DrawButton 2472
Enabled 2494
FocusedItem 2494

Font 2494
GenerateDragImage 2472
GetAllowReorder 2472
GetButtonAt 2472
GetButtonCategoriesClass 2473
GetButtonCategoryClass 2473
GetButtonItemClass 2473
GetButtonRect 2473
GetCategoryAt 2473
GetCategoryRect 2474
GetDragImages 2474
GetScrollPos 2474
GetTargetAt 2474
GradientDirection 2495
Height 2496
HotButtonColor 2497
Images 2497
MouseDown 2475
MouseMove 2475
MouseUp 2476
Notification 2476
OnAfterDrawButton 2508
OnAlignInsertBefore 2508
OnAlignPosition 2508
OnBeforeDrawButton 2509
OnButtonClicked 2509
OnCategoryCollapase 2509
OnClick 2510
OnContextPopup 2510
OnCopyButton 2511
OnDockDrop 2511
OnDockOver 2513
OnDragDrop 2515
OnDragOver 2517
OnDrawButton 2519
OnDrawIcon 2520
OnDrawText 2520
OnEndDock 2520
OnEndDrag 2520
OnEnter 2521
OnExit 2524
OnGetHint 2525
OnHotButton 2526
OnKeyDown 2526
OnKeyPress 2529
OnKeyUp 2530
OnMouseActivate 2531
OnMouseDown 2532
OnMouseEnter 2535
OnMouseLeave 2535
OnMouseMove 2535
OnMouseUp 2537
OnMouseWheel 2540
OnMouseWheelDown 2540
OnMouseWheelUp 2540
OnReorderButton 2541
OnReorderCategory 2541
OnSelectedItemChange 2541
OnStartDock 2541
OnStartDrag 2542
Paint 2477
PopupMenu 2497
RegularButtonColor 2502
RemoveInsertionPoints 2477
Resize 2477
ScrollIntoView 2478
ScrollIntoView method 2478
ScrollRectIntoView 2478
SelectedButtonColor 2502
SelectedItem 2502
SetInsertionButton 2478
SetScrollPos 2478
ShowHint 2502
TabOrder 2504
TabStop 2504
TCategoryButtons events 2507
TCategoryButtons Members 2452
TCategoryButtons methods 2464

TCategoryButtons properties 2479
UpdateAllButtons 2479
UpdateButton 2479
Visible 2505
Width 2506
TCategoryDragObject class 444
about TCategoryDragObject class 444
Category 447
Create 446
TCategoryDragObject Members 445
TCategoryDragObject properties 446
tcbCancel enumeration member 11565
tcbClose enumeration member 11565
tcbNo enumeration member 11565
tcbOk enumeration member 11565
tcbRetry enumeration member 11565
tcbYes enumeration member 11565
TCellCollection class 12172
about TCellCollection class 12172
GetAttr 12174
GetAttrCount 12175
GetItem 12175
GetItemAttr 12176
Items 12177
Owner 12176
SetItem 12176
TCellCollection Members 12172
TCellCollection methods 12174
TCellCollection properties 12177
Update 12176
TCellItem class 12177
about TCellItem class 12177
AssignTo 12180
AutoAdded 12181
Create 12179
SetSizeStyle 12180
SetValue 12181
Size 12181
SizeStyle 12181
TCellItem Members 12178
TCellItem methods 12179
TCellItem properties 12181
Value 12182
TCGIApplication class 2554
about TCGIApplication class 2554
Destroy 2557
Run 2558
TCGIApplication Members 2555
TCGIApplication methods 2558
TCGIFactory class 2559
about TCGIFactory class 2559
Create 2560
NewRequest 2560
NewResponse 2561
OutputFileName 2561
TCGIFactory Members 2559
TCGIFactory methods 2560
TCGIFactory properties 2561
TCGIRequest class 2564
about TCGIRequest class 2564
Create 2567
GetDateVariable 2567
GetFieldByName 2568
GetIntegerVariable 2568
GetStringVariable 2569
ReadClient 2570
ReadString 2570
TCGIRequest Members 2565
TCGIRequest methods 2567
TranslateURI 2570
WriteClient 2571
WriteHeaders 2571
WriteString 2572
TCGIResponse class 2572
about TCGIResponse class 2572
Create 2575
GetContent 2576
GetDateVariable 2576

GetIntegerVariable 2577
GetLogMessage 2577
GetStatusCode 2578
GetStringVariable 2578
InitResponse 2579
SendRedirect 2579
SendResponse 2579
SendStream 2580
Sent 2580
SetContent 2580
SetDateVariable 2581
SetIntegerVariable 2581
SetLogMessage 2582
SetStatusCode 2582
SetStringVariable 2582
TCGIResponse Members 2573
TCGIResponse methods 2575
TChangeLink class 15358
about TChangeLink class 15358
Change 15360
Destroy 15359
OnChange 15361
Sender 15360
TChangeLink events 15361
TChangeLink Members 15358
TChangeLink methods 15360
TChangeLink properties 15360
TCharProperty class 10918
about TCharProperty class 10918
GetValue 10921
SetValue 10922
TCharProperty Members 10918
TCharProperty methods 10921
TCheckBox class 20996
about TCheckBox class 20996
Action 21008
Align 21009
Alignment 21012
AllowGrayed 21013
Anchors 21013
BiDiMode 21014
Caption 21014
Checked 21017
Color 21018
Constraints 21019
Ctl3D 21020
DragCursor 21020
DragKind 21020
DragMode 21021
Enabled 21022
Font 21023
OnClick 21036
OnContextPopup 21038
OnDragDrop 21039
OnDragOver 21041
OnEndDock 21044
OnEndDrag 21044
OnEnter 21045
OnExit 21048
OnKeyDown 21049
OnKeyPress 21052
OnKeyUp 21053
OnMouseActivate 21055
OnMouseDown 21056
OnMouseEnter 21058
OnMouseLeave 21058
OnMouseMove 21058
OnMouseUp 21061
OnStartDock 21063
OnStartDrag 21064
ParentBiDiMode 21024
ParentColor 21024
ParentCtl3D 21025
ParentFont 21025
ParentShowHint 21026
PopupMenu 21026
ShowHint 21031
State 21032

TabOrder 21033
TabStop 21034
TCheckBox events 21036
TCheckBox Members 20997
TCheckBox properties 21008
Visible 21034
WordWrap 21035
TCheckConstraint class 6841
about TCheckConstraint class 6841
Assign 6843
CustomConstraint 6844
ErrorMessage 6845
FromDictionary 6845
GetDisplayName 6843
ImportedConstraint 6845
TCheckConstraint Members 6842
TCheckConstraint methods 6843
TCheckConstraint properties 6844
TCheckConstraints class 6846
about TCheckConstraints class 6846
Add 6848
Create 6848
GetOwner 6849
Items 6849
TCheckConstraints Members 6846
TCheckConstraints methods 6848
TCheckConstraints properties 6849
TCheckListBox class 2594
about TCheckListBox class 2594
Align 2614
AllowGrayed 2617
Anchors 2618
AutoComplete 2618
BevelEdges 2619
BevelInner 2619
BevelKind 2619
BevelOuter 2620
BevelWidth 2620
BiDiMode 2621
BorderStyle 2621
Checked 2621
ClickCheck 2607
Color 2622
Columns 2623
Constraints 2623
Create 2606
CreateParams 2607
CreateWnd 2608
Ctl3D 2624
DeleteString 2609
Destroy 2606
DragCursor 2624
DragKind 2625
DragMode 2625
DrawItem 2609
Enabled 2627
Flat 2627
Font 2627
GetCheckWidth 2609
GetItemData 2610
Header 2628
HeaderBackgroundColor 2629
HeaderColor 2629
ImeMode 2630
ImeName 2630
IntegralHeight 2631
InternalGetItemData 2610
InternalSetItemData 2610
ItemEnabled 2631
ItemHeight 2632
Items 2632
KeyPress 2611
LoadRecreateItems 2611
MouseDown 2611
OnClick 2649
OnClickCheck 2651
OnContextPopup 2652
OnData 2652

OnDataFind 2653
OnDataObject 2653
OnDblClick 2654
OnDragDrop 2654
OnDragOver 2656
OnDrawItem 2659
OnEndDock 2661
OnEndDrag 2661
OnEnter 2662
OnExit 2665
OnKeyDown 2666
OnKeyPress 2669
OnKeyUp 2670
OnMeasureItem 2672
OnMouseActivate 2673
OnMouseDown 2674
OnMouseEnter 2677
OnMouseLeave 2677
OnMouseMove 2677
OnMouseUp 2679
OnStartDock 2682
OnStartDrag 2682
ParentBiDiMode 2635
ParentColor 2635
ParentCtl3D 2636
ParentFont 2636
ParentShowHint 2637
PopupMenu 2637
ResetContent 2612
SaveRecreateItems 2612
SetItemData 2612
ShowHint 2642
Sorted 2643
State 2645
Style 2645
TabOrder 2646
TabStop 2647
TabWidth 2647
TCheckListBox events 2649
TCheckListBox Members 2594
TCheckListBox methods 2607
TCheckListBox properties 2613
Visible 2647
TClassFinder class 2728
about TClassFinder class 2728
Create 2729
Destroy 2729
GetClass 2730
GetClasses 2731
TClassFinder Members 2729
TClassFinder methods 2730
TClassHelperBase class 22251
_Create 22253
about TClassHelperBase class 22251
FInstance 22253
TClassHelperBase fields 22253
TClassHelperBase Members 22252
TClassList class 5646
about TClassList class 5646
Add 5648
Extract 5649
First 5649
GetItems 5649
IndexOf 5650
Insert 5650
Items 5652
Last 5651
Remove 5651
SetItems 5652
TClassList Members 5646
TClassList methods 5648
TClassList properties 5652
TClassProperty class 10922
about TClassProperty class 10922
GetAttributes 10926
GetProperties 10926
GetValue 10927
TClassProperty Members 10923

TClassProperty methods 10925
TClassWrapper class 11117
about TClassWrapper class 11117
ClassNameIs 11119
Create 11119
GetClassName 11120
GetClassParent 11120
GetUnitName 11120
TClassWrapper Members 11118
TClassWrapper methods 11119
TClientBlobStream class 8285
about TClientBlobStream class 8285
Create 8287
Destroy 8288
TClientBlobStream Members 8286
TClientBlobStream methods 8288
Truncate 8288
Write 8289
TClientDataSet class 8289
about TClientDataSet class 8289
Active 8305
AfterApplyUpdates 8332
AfterCancel 8333
AfterClose 8335
AfterDelete 8336
AfterEdit 8339
AfterExecute 8340
AfterGetParams 8340
AfterGetRecords 8341
AfterInsert 8342
AfterOpen 8342
AfterPost 8344
AfterRefresh 8344
AfterRowRequest 8345
AfterScroll 8346
Aggregates 8308
AggregatesActive 8308
AutoCalcFields 8309
BeforeApplyUpdates 8346
BeforeCancel 8347
BeforeClose 8347
BeforeDelete 8348
BeforeEdit 8348
BeforeExecute 8349
BeforeGetParams 8350
BeforeGetRecords 8350
BeforeInsert 8352
BeforeOpen 8353
BeforePost 8354
BeforeRefresh 8354
BeforeRowRequest 8355
BeforeScroll 8356
CommandText 8310
ConnectionBroker 8311
Constraints 8311
DataSetField 8312
DisableStringTrim 8312
FetchOnDemand 8313
FieldDefs 8314
FileName 8317
Filter 8317
Filtered 8319
FilterOptions 8320
IndexDefs 8320
IndexFieldNames 8322
IndexName 8322
MasterFields 8323
MasterSource 8324
ObjectView 8324
OnCalcFields 8356
OnDeleteError 8357
OnEditError 8357
OnFilterRecord 8358
OnNewRecord 8359
OnPostError 8359
OnReconcileError 8359
PacketRecords 8325
Params 8326

ProviderName 8329
Ranged 8329
ReadOnly 8330
RemoteServer 8330
StoreDefs 8331
TClientDataSet events 8331
TClientDataSet Members 8291
TClientDataSet properties 8304
TClientDataSetApply class 8548
about TClientDataSetApply class 8548
Create 8553
DataSource 8555
Destroy 8553
DisplayErrorDlg 8556
DoReconcileError 8554
ExecuteTarget 8554
MaxErrors 8556
TClientDataSetApply Members 8549
TClientDataSetApply methods 8554
TClientDataSetApply properties 8555
UpdateTarget 8555
TClientDataSetRevert class 8556
about TClientDataSetRevert class 8556
DataSource 8563
ExecuteTarget 8562
TClientDataSetRevert Members 8557
TClientDataSetRevert methods 8561
TClientDataSetRevert properties 8563
UpdateTarget 8562
TClientDataSetUndo class 8563
about TClientDataSetUndo class 8563
DataSource 8569
ExecuteTarget 8568
FollowChange 8570
TClientDataSetUndo Members 8564
TClientDataSetUndo methods 8568
TClientDataSetUndo properties 8569
UpdateTarget 8569
TClientSocket class 18752
about TClientSocket class 18752
Active 18759
Address 18759
ClientType 18760
Create 18756
Destroy 18757
DoActivate 18757
GetClientType 18758
Host 18760
OnConnect 18762
OnConnecting 18763
OnDisconnect 18764
OnError 18764
OnLookup 18764
OnRead 18765
OnWrite 18765
Port 18761
Service 18761
SetClientType 18758
Socket 18762
TClientSocket events 18762
TClientSocket Members 18753
TClientSocket methods 18757
TClientSocket properties 18758
TClientSocketThread class 20059
about TClientSocketThread class 20059
ClientSocket 20063
Create 20060
Destroy 20061
Execute 20061
ExecuteSyncProc 20062
ServerSocketThread 20063
SyncProc 20062
TClientSocketThread Members 20059
TClientSocketThread methods 20061
TClientSocketThread properties 20063
TClientWinSocket class 18766
about TClientWinSocket class 18766
ClientType 18769

Connect 18768
SetClientType 18769
TClientWinSocket Members 18766
TClientWinSocket methods 18768
TClientWinSocket properties 18769
TClipboard class 3328
about TClipboard class 3328
Assign 3331
AssignTo 3333
AsText 3347
Clear 3333
Close 3336
Destroy 3330
FormatCount 3348
Formats 3348
GetAsHandle 3336
GetComponent 3339
GetTextBuf 3340
Handle 3349
HasFormat 3340
MainWndProc 3342
Open 3342
OpenRefCount 3349
SetAsHandle 3343
SetBuffer 3345
SetComponent 3345
SetTextBuf 3346
TClipboard Members 3329
TClipboard methods 3330
TClipboard properties 3346
WndProc 3346
tcLowerCase enumeration member 13623
TCollection class 2731
about TCollection class 2731
Add 2736
Added 2736
Assign 2736
BeginUpdate 2739
Capacity 2748
Changed 2740
Clear 2741
Count 2749
Create 2734
Delete 2741
Deleting 2741
Destroy 2735
EndUpdate 2742
FindItemID 2743
GetAttr 2743
GetAttrCount 2744
GetEnumerator 2744
GetItem 2744
GetItemAttr 2745
GetNamePath 2745
Insert 2746
ItemClass 2751
Items 2751
NextID 2752
Notify 2746
Owner 2746
PropName 2752
SetItem 2747
SetItemName 2747
TCollection Members 2733
TCollection methods 2735
TCollection properties 2748
Update 2747
UpdateCount 2753
TCollectionEnumerator class 2753
about TCollectionEnumerator class 2753
Create 2754
Current 2755
GetCurrent 2754
MoveNext 2754
TCollectionEnumerator Members 2753
TCollectionEnumerator methods 2754
TCollectionEnumerator properties 2755
TCollectionItem class 2755

about TCollectionItem class 2755
Changed 2758
Collection 2761
Create 2757
Destroy 2757
DisplayName 2761
GetDisplayName 2758
GetNamePath 2759
GetOwner 2759
ID 2761
Index 2762
SetCollection 2759
SetDisplayName 2760
SetIndex 2760
TCollectionItem Members 2756
TCollectionItem methods 2757
TCollectionItem properties 2760
TColorBox class 12182
about TColorBox class 12182
Align 12196
Anchors 12200
AutoComplete 12200
AutoDropDown 12201
BevelEdges 12201
BevelInner 12201
BevelKind 12202
BevelOuter 12202
BiDiMode 12203
Color 12203
Constraints 12204
Ctl3D 12205
DefaultColorColor 12205
DropDownCount 12205
Enabled 12206
Font 12206
ItemHeight 12207
NoneColorColor 12208
OnChange 12221
OnClick 12223
OnCloseUp 12225
OnContextPopup 12225
OnDblClick 12226
OnDragDrop 12226
OnDragOver 12228
OnDropDown 12231
OnEndDock 12232
OnEndDrag 12232
OnEnter 12233
OnExit 12236
OnGetColors 12237
OnKeyDown 12238
OnKeyPress 12240
OnKeyUp 12242
OnMouseEnter 12243
OnMouseLeave 12244
OnSelect 12244
OnStartDock 12244
OnStartDrag 12245
ParentBiDiMode 12208
ParentColor 12209
ParentCtl3D 12209
ParentFont 12210
ParentShowHint 12210
PopupMenu 12211
Selected 12215
ShowHint 12215
Style 12217
TabOrder 12217
TabStop 12218
TColorBox events 12220
TColorBox Members 12183
TColorBox properties 12195
Visible 12218
TColorDialog class 11270
about TColorDialog class 11270
Color 11275
Create 11273
Ctl3D 11276

CustomColors 11276
Destroy 11274
Execute 11274
Options 11277
TColorDialog Members 11270
TColorDialog methods 11274
TColorDialog properties 11275
TColorListBox class 12246
about TColorListBox class 12246
Align 12259
Anchors 12262
AutoComplete 12263
BevelEdges 12263
BevelInner 12264
BevelKind 12264
BevelOuter 12264
BiDiMode 12265
Color 12265
Constraints 12266
Ctl3D 12266
DefaultColorColor 12267
Enabled 12267
Font 12267
ItemHeight 12268
NoneColorColor 12269
OnClick 12283
OnContextPopup 12285
OnDblClick 12286
OnDragDrop 12286
OnDragOver 12288
OnEndDock 12291
OnEndDrag 12291
OnEnter 12292
OnExit 12295
OnGetColors 12296
OnKeyDown 12296
OnKeyPress 12299
OnKeyUp 12300
OnMouseActivate 12302
OnMouseDown 12303
OnMouseEnter 12305
OnMouseLeave 12305
OnMouseMove 12306
OnMouseUp 12308
OnStartDock 12310
OnStartDrag 12311
ParentBiDiMode 12269
ParentColor 12270
ParentCtl3D 12270
ParentFont 12270
ParentShowHint 12271
PopupMenu 12271
Selected 12276
ShowHint 12278
Style 12279
TabOrder 12280
TabStop 12281
TColorListBox events 12283
TColorListBox Members 12246
TColorListBox properties 12258
Visible 12281
TColorSelect class 20521
about TColorSelect class 20521
BeforeExecute 20531
Caption 20526
Dialog 20527
Enabled 20527
GetDialogClass 20526
HelpContext 20527
HelpKeyword 20528
HelpType 20528
Hint 20529
ImageIndex 20529
OnAccept 20531
OnCancel 20532
OnHint 20532
OnUpdate 20532
SecondaryShortCuts 20530

ShortCut 20530
TColorSelect events 20531
TColorSelect Members 20521
TColorSelect methods 20526
TColorSelect properties 20526
Visible 20530
TColumn class 9583
about TColumn class 9583
Alignment 9592
Assign 9586
AssignedValues 9593
ButtonStyle 9593
Color 9594
Create 9585
CreateTitle 9587
DefaultAlignment 9587
DefaultColor 9587
DefaultFont 9588
DefaultImeMode 9588
DefaultImeName 9588
DefaultReadOnly 9589
DefaultWidth 9589
Depth 9590
Destroy 9585
DropDownRows 9594
Expandable 9594
Expanded 9595
Field 9595
FieldName 9595
Font 9596
GetDisplayName 9590
GetGrid 9590
Grid 9596
ImeMode 9597
ImeName 9597
IsStored 9598
ParentColumn 9598
PickList 9598
PopupMenu 9599
ReadOnly 9599
RefreshDefaultFont 9591
RestoreDefaults 9591
SetIndex 9591
Showing 9599
TColumn Members 9583
TColumn methods 9586
TColumn properties 9592
Title 9600
Visible 9600
Width 9600
TColumnCollection class 12311
about TColumnCollection class 12311
Add 12314
Create 12314
GetItemAttr 12314
Notify 12315
TColumnCollection Members 12312
TColumnCollection methods 12314
TColumnItem class 12315
about TColumnItem class 12315
TColumnTitle class 9601
about TColumnTitle class 9601
Alignment 9606
Assign 9604
Caption 9607
Color 9607
Column 9607
Create 9603
DefaultAlignment 9604
DefaultCaption 9604
DefaultColor 9605
DefaultFont 9605
Destroy 9603
Font 9608
RefreshDefaultFont 9606
RestoreDefaults 9606
TColumnTitle Members 9601
TColumnTitle methods 9603

TColumnTitle properties 9606
TCOMAdminCatalog class 3359
about TCOMAdminCatalog class 3359
BackupREGDB 3365
Connect 3365
ConnectTo 3366
Create 3363
DefaultInterface 3379
Destroy 3364
Disconnect 3366
ExportApplication 3367
Get_MajorVersion 3367
Get_MinorVersion 3368
GetCollection 3368
GetCollectionByQuery 3368
GetEventClassesForIID 3369
GetMultipleComponentsInfo 3370
ICOMAdminCatalog_Connect 3371
ImportComponent 3371
InitServerData 3372
InstallApplication 3372
InstallComponent 3373
InstallEventClass 3373
InstallMultipleComponents 3374
InstallMultipleEventClasses 3375
MajorVersion 3379
MinorVersion 3380
QueryApplicationFile 3375
RefreshComponents 3376
RefreshRouter 3376
RestoreREGDB 3376
ServiceCheck 3377
ShutdownApplication 3377
StartApplication 3378
StartRouter 3378
StopRouter 3378
TCOMAdminCatalog Members 3360
TCOMAdminCatalog methods 3364
TCOMAdminCatalog properties 3379
TCOMAdminCatalogCollection class 3380
about TCOMAdminCatalogCollection class 3380
Add 3386
AddEnabled 3393
Connect 3386
ConnectTo 3386
Count 3393
Create 3384
DataStoreMajorVersion 3394
DataStoreMinorVersion 3394
DefaultInterface 3395
Destroy 3385
Disconnect 3387
Get_AddEnabled 3387
Get_Count 3388
Get_DataStoreMajorVersion 3388
Get_DataStoreMinorVersion 3388
Get_Item 3389
Get_Name 3389
Get_RemoveEnabled 3389
GetCollection 3389
GetUtilInterface 3390
InitServerData 3390
Name 3395
Populate 3391
PopulateByKey 3391
PopulateByQuery 3392
Remove 3392
RemoveEnabled 3395
SaveChanges 3392
TCOMAdminCatalogCollection Members 3381
TCOMAdminCatalogCollection methods 3385
TCOMAdminCatalogCollection properties 3393
TCOMAdminCatalogObject class 3396
about TCOMAdminCatalogObject class 3396
Connect 3401
ConnectTo 3401
Create 3400
DefaultInterface 3405

Destroy 3400
Disconnect 3402
Get_Key 3402
Get_Name 3402
Get_Valid 3403
Get_Value 3403
InitServerData 3403
IsPropertyReadOnly 3404
IsPropertyWriteOnly 3404
Key 3405
Name 3406
Set_Value 3404
TCOMAdminCatalogObject Members 3396
TCOMAdminCatalogObject methods 3400
TCOMAdminCatalogObject properties 3405
Valid 3406
Value 3407
TComboBox class 21064
about TComboBox class 21064
Align 21078
Anchors 21082
AutoCloseUp 21082
AutoComplete 21083
AutoCompleteDelay 21083
AutoDropDown 21083
BevelEdges 21084
BevelInner 21084
BevelKind 21085
BevelOuter 21085
BiDiMode 21085
CharCase 21086
Color 21086
Constraints 21087
Ctl3D 21088
DragCursor 21088
DragKind 21089
DragMode 21089
DropDownCount 21091
Enabled 21091
Font 21091
ImeMode 21092
ImeName 21093
ItemHeight 21094
ItemIndex 21094
Items 21095
MaxLength 21097
OnChange 21112
OnClick 21114
OnCloseUp 21116
OnContextPopup 21116
OnDblClick 21117
OnDragDrop 21117
OnDragOver 21120
OnDrawItem 21122
OnDropDown 21123
OnEndDock 21124
OnEndDrag 21124
OnEnter 21125
OnExit 21128
OnKeyDown 21129
OnKeyPress 21132
OnKeyUp 21133
OnMeasureItem 21135
OnMouseEnter 21135
OnMouseLeave 21136
OnSelect 21136
OnStartDock 21136
OnStartDrag 21137
ParentBiDiMode 21097
ParentColor 21098
ParentCtl3D 21098
ParentFont 21099
ParentShowHint 21099
PopupMenu 21100
ShowHint 21104
Sorted 21106
Style 21107
TabOrder 21108

wwww

TabStop 21108
TComboBox events 21111
TComboBox Members 21065
TComboBox properties 21077
Text 21109
Visible 21110
TComboBoxEx class 3463
about TComboBoxEx class 3463
Action 3476
Align 3477
Anchors 3480
AutoCompleteOptions 3481
BiDiMode 3481
Color 3482
Constraints 3482
Ctl3D 3483
DragCursor 3483
DragKind 3484
DragMode 3484
DropDownCount 3486
Enabled 3486
Font 3487
Images 3488
ImeMode 3488
ImeName 3489
ItemHeight 3489
ItemsEx 3490
MaxLength 3490
OnBeginEdit 3504
OnChange 3504
OnClick 3507
OnContextPopup 3509
OnDblClick 3509
OnDragDrop 3510
OnDragOver 3512
OnDropDown 3514
OnEndDock 3516
OnEndDrag 3516
OnEndEdit 3517
OnEnter 3517
OnExit 3520
OnKeyDown 3521
OnKeyPress 3524
OnKeyUp 3525
OnMouseMove 3527
OnSelect 3529
OnStartDock 3530
OnStartDrag 3530
ParentBiDiMode 3491
ParentColor 3491
ParentCtl3D 3492
ParentFont 3492
ParentShowHint 3492
PopupMenu 3493
ShowHint 3498
Style 3499
StyleEx 3499
TabOrder 3500
TabStop 3500
TComboBoxEx events 3504
TComboBoxEx Members 3463
TComboBoxEx properties 3476
Text 3501
Visible 3502
TComboBoxExActionLink class 3531
about TComboBoxExActionLink class 3531
AddItem 3534
AddItem method 3534
TComboBoxExActionLink Members 3531
TComboBoxExActionLink methods 3534
TComboBoxExStrings class 3535
about TComboBoxExStrings class 3535
Add 3538
AddItem 3539
AddObject 3539
Clear 3539
Create 3537
Delete 3539

Destroy 3537
Exchange 3540
Get 3540
GetCapacity 3541
GetCount 3541
GetItemClass 3542
GetItemsClass 3542
GetObject 3542
IndexOf 3542
IndexOfName 3543
Insert 3543
ItemsEx 3545
Move 3543
PutObject 3544
SetUpdateState 3545
SortType 3545
TComboBoxExStrings Members 3535
TComboBoxExStrings methods 3538
TComboBoxExStrings properties 3545
TComboExItem class 3546
about TComboExItem class 3546
Assign 3548
Indent 3551
OverlayImageIndex 3551
SelectedImageIndex 3552
SetCaption 3549
SetData 3549
SetDisplayName 3549
SetImageIndex 3550
SetIndex 3550
SetOverlayImageIndex 3550
SetSelectedImageIndex 3551
TComboExItem Members 3546
TComboExItem methods 3548
TComboExItem properties 3551
TComboExItems class 3552
about TComboExItems class 3552
Add 3555
AddItem 3556
ComboItems 3557
Insert 3556
Notify 3557
SetItem 3557
TComboExItems Members 3553
TComboExItems methods 3555
TComboExItems properties 3557
TComClassManager class 5550
about TComClassManager class 5550
Create 5551
Destroy 5552
ForEachFactory 5552
GetFactoryFromClass 5553
GetFactoryFromClassID 5553
TComClassManager Members 5550
TComClassManager methods 5552
TCOMConnection class 15996
about TCOMConnection class 15996
DoConnect 16001
SetConnected 16001
TCOMConnection Members 15997
TCOMConnection methods 16001
TCommonCalendar class 3558
about TCommonCalendar class 3558
BoldDays 3570
CalColors 3575
CalendarHandle 3576
CalExceptionClass 3576
CheckEmptyDate 3571
CheckValidDate 3571
Create 3569
CreateWnd 3572
Date 3576
DateTime 3577
Destroy 3570
EndDate 3577
FirstDayOfWeek 3578
GetCalendarHandle 3572
GetCalStyles 3573

MaxDate 3578
MaxSelectRange 3579
MinDate 3579
MonthDelta 3579
MsgSetCalColors 3573
MsgSetDateTime 3574
MsgSetRange 3574
MultiSelect 3580
OnGetMonthInfo 3581
ShowToday 3580
ShowTodayCircle 3580
TCommonCalendar events 3581
TCommonCalendar Members 3558
TCommonCalendar methods 3570
TCommonCalendar properties 3575
WeekNumbers 3581
TCommonDialog class 11277
about TCommonDialog class 11277
Create 11281
Ctl3D 11285
DefaultHandler 11282
Destroy 11281
DoClose 11282
DoShow 11283
Execute 11283
Execute method 11283
Handle 11286
HelpContext 11286
MessageHook 11284
OnClose 11287
OnShow 11288
TaskModalDialog 11284
TCommonDialog events 11287
TCommonDialog Members 11278
TCommonDialog methods 11282
TCommonDialog properties 11285
Template 11287
TemplateModule 11287
WndProc 11285
TCommonDialogAction class 20533
about TCommonDialogAction class 20533
BeforeExecute 20543
Create 20538
Destroy 20538
DoAccept 20540
DoCancel 20540
ExecuteResult 20542
ExecuteTarget 20540
FDialog 20539
GetDialogClass 20541
HandlesTarget 20541
Notification 20541
OnAccept 20543
OnCancel 20543
SetupDialog 20542
TCommonDialogAction events 20542
TCommonDialogAction fields 20539
TCommonDialogAction Members 20534
TCommonDialogAction methods 20539
TCommonDialogAction properties 20542
TComObject class 5553
_AddRef 5558
_Release 5558
about TComObject class 5553
Controller 5561
Create 5555
CreateAggregated 5556
CreateFromFactory 5556
Destroy 5557
Factory 5561
Initialize 5558
InterfaceSupportsErrorInfo 5559
ObjAddRef 5559
ObjQueryInterface 5559
ObjRelease 5559
QueryInterface 5560
RefCount 5562
SafeCallException 5560

ServerExceptionHandler 5562
TComObject Members 5554
TComObject methods 5558
TComObject properties 5560
TComObjectFactory class 5563
_AddRef 5567
_Release 5567
about TComObjectFactory class 5563
ClassID 5571
ClassName 5572
ComClass 5572
ComServer 5572
Create 5565
CreateComObject 5567
CreateInstance 5568
CreateInstanceLic 5568
Description 5572
Destroy 5566
ErrorIID 5573
GetLicenseString 5568
GetLicInfo 5568
GetProgID 5569
HasMachineLicense 5569
Instancing 5573
LicString 5573
LockServer 5569
ProgID 5574
QueryInterface 5569
RegisterClassObject 5569
RequestLicKey 5570
ShowErrors 5574
SupportsLicensing 5574
TComObjectFactory Members 5564
TComObjectFactory methods 5566
TComObjectFactory properties 5571
ThreadingModel 5575
UpdateRegistry 5570
ValidateUserLicense 5571
TComponent class 2762
_AddRef 2769
_Release 2769
about TComponent class 2762
BeforeDestruction 2770
ChangeName 2770
ComObject 2793
ComponentCount 2794
ComponentIndex 2799
Components 2799
ComponentState 2804
ComponentStyle 2805
Create 2766
DefineProperties 2771
DesignInfo 2805
Destroy 2766
DestroyComponents 2771
Destroying 2772
ExecuteAction 2772
FComponentStyle 2767
FindComponent 2772
FreeNotification 2775
FreeOnRelease 2776
GetChildOwner 2776
GetChildParent 2776
GetChildren 2777
GetEnumerator 2777
GetIDsOfNames 2777
GetNamePath 2778
GetOwner 2778
GetParentComponent 2778
GetTypeInfo 2779
GetTypeInfoCount 2779
HasParent 2780
InsertComponent 2780
Invoke 2781
IsImplementorOf 2782
Loaded 2782
Name 2805
Notification 2783

Owner 2807
PaletteCreated 2783
QueryInterface 2784
ReadState 2784
ReferenceInterface 2785
RemoveComponent 2785
RemoveFreeNotification 2787
SafeCallException 2787
SetAncestor 2787
SetChildOrder 2788
SetDesigning 2788
SetDesignInstance 2788
SetInline 2789
SetName 2789
SetParentComponent 2790
SetSubComponent 2790
Tag 2808
TComponent fields 2767
TComponent Members 2763
TComponent methods 2768
TComponent properties 2793
UpdateAction 2790
Updated 2791
UpdateRegistry 2791
Updating 2791
ValidateContainer 2792
ValidateInsert 2792
ValidateRename 2792
VCLComObject 2812
WriteState 2793
TComponentEditor class 10927
about TComponentEditor class 10927
Component 10934
Copy 10930
Create 10929
Designer 10935
Edit 10930
ExecuteVerb 10931
GetComponent 10931
GetDesigner 10932
GetVerb 10932
GetVerbCount 10933
IsInInlined 10933
PrepareItem 10933
TComponentEditor Members 10928
TComponentEditor methods 10930
TComponentEditor properties 10934
TComponentEnumerator class 2812
about TComponentEnumerator class 2812
Create 2813
Current 2814
GetCurrent 2813
MoveNext 2814
TComponentEnumerator Members 2812
TComponentEnumerator methods 2813
TComponentEnumerator properties 2814
TComponentFactory class 23684
about TComponentFactory class 23684
Create 23686
CreateComObject 23686
CreateInstance 23687
TComponentFactory Members 23684
TComponentFactory methods 23686
UpdateRegistry 23687
TComponentGuidelines class 10935
about TComponentGuidelines class 10935
CalcHorzPos 10937
CalcVertPos 10938
Component 10939
Container 10939
GetCount 10938
GetDesignerGuideOffset 10938
GetDesignerGuideType 10938
Initialize 10938
TComponentGuidelines Members 10936
TComponentGuidelines methods 10937
TComponentGuidelines properties 10939
TComponentList class 5653

about TComponentList class 5653
Add 5656
Destroy 5655
Extract 5656
First 5657
GetItems 5657
HandleFreeNotify 5657
IndexOf 5658
Insert 5658
Items 5660
Last 5659
Notify 5659
Remove 5659
SetItems 5660
TComponentList Members 5653
TComponentList methods 5655
TComponentList properties 5660
TComponentNameProperty class 10939
about TComponentNameProperty class 10939
GetAttributes 10943
GetEditLimit 10943
TComponentNameProperty Members 10940
TComponentNameProperty methods 10942
TComponentProperty class 10944
about TComponentProperty class 10944
AllEqual 10947
Edit 10948
FilterFunc 10948
GetAttributes 10948
GetComponentReference 10949
GetEditLimit 10949
GetProperties 10949
GetSelections 10950
GetValue 10950
GetValues 10951
SetValue 10951
ShowReferenceProperty 10951
TComponentProperty Members 10944
TComponentProperty methods 10947
TComponentsPageProducer class 5614
about TComponentsPageProducer class 5614
DoTagEvent 5618
FindComponent 5619
FindComponentName 5619
GetComponentContent 5619
GetContentOptions 5619
HandleTag 5620
OnHTMLTag 5620
TComponentsPageProducer events 5620
TComponentsPageProducer Members 5615
TComponentsPageProducer methods 5618
TCompressionStream class 26618
about TCompressionStream class 26618
CompressionRate 26622
Create 26620
Destroy 26620
OnProgress 26623
Read 26621
Seek 26621
TCompressionStream events 26623
TCompressionStream Members 26619
TCompressionStream methods 26621
TCompressionStream properties 26622
Write 26622
TComServer class 5623
about TComServer class 5623
CountFactory 5626
CountObject 5627
Create 5625
Destroy 5626
GetHelpFileName 5627
GetServerFileName 5627
GetServerKey 5627
GetServerName 5628
GetStartSuspended 5628
GetTypeLib 5628
Initialize 5628
IsInprocServer 5630

LoadTypeLib 5629
ObjectCount 5631
OnLastRelease 5632
SetHelpFileName 5629
SetServerName 5629
StartMode 5631
TComServer events 5632
TComServer Members 5624
TComServer methods 5626
TComServer properties 5630
UIInteractive 5632
UpdateRegistry 5630
TComServerObject class 5575
about TComServerObject class 5575
CountFactory 5577
CountObject 5577
GetHelpFileName 5577
GetServerFileName 5578
GetServerKey 5578
GetServerName 5578
GetStartSuspended 5578
GetTypeLib 5579
HelpFileName 5579
ServerFileName 5580
ServerKey 5580
ServerName 5580
SetHelpFileName 5579
StartSuspended 5580
TComServerObject Members 5576
TComServerObject methods 5577
TComServerObject properties 5579
TypeLib 5581
TConnect 23235
TConnect namespace 23235
Classes 23236
TConnect.TLocalConnection 23236
TConnect.TLocalConnection.AppServer 23250
TConnect.TLocalConnection.AS_ApplyUpdates 23242
TConnect.TLocalConnection.AS_DataRequest 23242
TConnect.TLocalConnection.AS_Execute 23243
TConnect.TLocalConnection.AS_GetParams 23244
TConnect.TLocalConnection.AS_GetProviderNames 23245
TConnect.TLocalConnection.AS_GetRecords 23245
TConnect.TLocalConnection.AS_RowRequest 23246
TConnect.TLocalConnection.Create 23240
TConnect.TLocalConnection.Destroy 23241
TConnect.TLocalConnection.GetConnected 23247
TConnect.TLocalConnection.GetProvider 23247
TConnect.TLocalConnection.GetProviderNames 23248
TConnect.TLocalConnection.GetServer 23248
TConnect.TLocalConnection.InterfaceSupportsErrorInfo 23248
TConnect.TLocalConnection.Notification 23248
TConnect.TLocalConnection.ProviderCount 23250
TConnect.TLocalConnection.Providers 23251
TConnect.TLocalConnection.RegisterProvider 23249
TConnect.TLocalConnection.SafeCallException 23249
TConnect.TLocalConnection.UnRegisterProvider 23250
TConnectionBroker class 8360
about TConnectionBroker class 8360
AfterConnect 8371
AfterDisconnect 8372
BeforeConnect 8372
BeforeDisconnect 8372
Connected 8368
Connection 8368
Create 8364
GetAppServer 8365
GetConnected 8365
GetServer 8366
Loaded 8366
LoginPrompt 8369
Notification 8366
OnGetUsername 8373
OnLogin 8370
SetConnected 8367
TConnectionBroker events 8371
TConnectionBroker Members 8361
TConnectionBroker methods 8365

TConnectionBroker properties 8367
TConnectionPoint class 1570
about TConnectionPoint class 1570
Advise 1573
Create 1572
Destroy 1572
EnumConnections 1573
GetConnectionInterface 1573
GetConnectionPointContainer 1574
SinkList 1574
TConnectionPoint Members 1571
TConnectionPoint methods 1573
TConnectionPoint properties 1574
Unadvise 1574
TConnectionPoints class 1575
about TConnectionPoints class 1575
Controller 1578
Create 1576
CreateConnectionPoint 1577
Destroy 1576
EnumConnectionPoints 1577
FindConnectionPoint 1578
TConnectionPoints Members 1575
TConnectionPoints methods 1576
TConnectionPoints properties 1578
TConnectionStringCategories class 1888
about TConnectionStringCategories class 1888
DataBase 1889
DriverConfiguration 1889
Dynalink 1889
TConnectionStringCategories constants 1889
TConnectionStringCategories Members 1889
TContainedAction class 289
about TContainedAction class 289
ActionList 295
Category 296
Destroy 292
Execute 293
GetParentComponent 293
HasParent 294
Index 296
ReadState 294
SetParentComponent 294
TContainedAction Members 289
TContainedAction methods 293
TContainedAction properties 295
Update 295
TContainedObject class 22253
about TContainedObject class 22253
QueryInterface 22255
TContainedObject Members 22254
TContainedObject methods 22255
TContentParser class 15088
about TContentParser class 15088
CanParse 15090
Destroy 15090
GetContentFields 15091
GetFiles 15091
TContentParser Members 15089
TContentParser methods 15090
TControl class 5724
about TControl class 5724
Action 5797
ActionChange 5737
ActionLink 5798
AdjustSize 5738
Align 5798
AlignWithMargins 5802
Anchors 5802
AssignTo 5739
AutoSize 5803
BeginAutoDrag 5739
BeginDrag 5740
BiDiMode 5803
BoundsRect 5803
BringToFront 5740
CalcCursorPos 5741
CanAutoSize 5741

CanResize 5742
Caption 5806
Changed 5742
ChangeScale 5743
Click 5743
ClientHeight 5808
ClientOrigin 5809
ClientRect 5810
ClientToParent 5744
ClientToScreen 5744
ClientWidth 5811
Color 5812
ConstrainedResize 5746
Constraints 5812
ControlState 5813
ControlStyle 5814
Create 5731
Cursor 5815
DblClick 5747
DefaultDockImage 5747
DefaultHandler 5748
DefineProperties 5748
DesignWndProc 5749
DesktopFont 5818
Destroy 5732
Dock 5749
DockOrientation 5818
DockTrackNoTarget 5749
DoContextPopup 5750
DoDock 5750
DoEndDock 5750
DoEndDrag 5751
DoMouseWheel 5751
DoMouseWheelDown 5752
DoMouseWheelUp 5752
DoStartDock 5753
DoStartDrag 5753
DragCanceled 5754
DragCursor 5818
DragDrop 5754
Dragging 5754
DragKind 5819
DragMode 5819
DragOver 5755
DrawDragDockImage 5755
DrawTextBiDiModeFlags 5756
DrawTextBiDiModeFlagsReadingOnly 5756
Enabled 5821
EndDrag 5757
EraseDragDockImage 5757
ExplicitHeight 5821
ExplicitLeft 5822
ExplicitTop 5822
ExplicitWidth 5822
FAnchorMove 5733
FAnchorOrigin 5734
FAnchorRules 5734
FExplicitHeight 5734
FExplicitLeft 5734
FExplicitTop 5734
FExplicitWidth 5735
Floating 5823
FloatingDockSiteClass 5823
Font 5823
FOriginalParentSize 5735
GetAction 5758
GetActionLinkClass 5758
GetClientOrigin 5758
GetClientRect 5759
GetControlsAlignment 5759
GetDeviceContext 5760
GetDockEdge 5763
GetDragImages 5763
GetEnabled 5763
GetFloating 5764
GetFloatingDockSiteClass 5764
GetPalette 5765
GetParentComponent 5765

GetPopupMenu 5765
GetTextBuf 5766
GetTextLen 5767
HasParent 5768
Height 5825
HelpContext 5826
HelpKeyword 5826
HelpType 5827
Hide 5769
Hint 5827
HostDockSite 5829
InitiateAction 5769
Invalidate 5770
IsControl 5829
IsRightToLeft 5770
Left 5830
Loaded 5771
LRDockWidth 5831
ManualDock 5771
ManualFloat 5772
Margins 5831
MouseActivate 5773
MouseCapture 5831
MouseDown 5773
MouseMove 5773
MouseUp 5774
MouseWheelHandler 5774
Notification 5775
OnCanResize 5851
OnClick 5851
OnConstrainedResize 5853
OnContextPopup 5854
OnDblClick 5854
OnDragDrop 5855
OnDragOver 5857
OnEndDock 5859
OnEndDrag 5860
OnMouseActivate 5861
OnMouseDown 5862
OnMouseEnter 5864
OnMouseLeave 5864
OnMouseMove 5865
OnMouseUp 5867
OnMouseWheel 5869
OnMouseWheelDown 5869
OnMouseWheelUp 5870
OnResize 5870
OnStartDock 5871
OnStartDrag 5871
PaletteChanged 5775
Parent 5832
ParentBiDiMode 5833
ParentColor 5834
ParentFont 5834
ParentShowHint 5835
ParentToClient 5776
Perform 5776
PopupMenu 5835
PositionDockRect 5777
ReadState 5778
Refresh 5779
Repaint 5779
ReplaceDockedControl 5779
RequestAlign 5780
Resize 5780
ScaleConstraints 5781
ScalingFlags 5840
ScreenToClient 5781
SendCancelMode 5781
SendDockNotification 5782
SendToBack 5782
SetAutoSize 5783
SetBiDiMode 5783
SetBounds 5783
SetDesignVisible 5784
SetDragMode 5785
SetEnabled 5785
SetName 5785

gggggg

SetParent 5786
SetParentBiDiMode 5786
SetParentComponent 5787
SetTextBuf 5787
SetZOrder 5788
Show 5788
ShowHint 5840
TBDockHeight 5842
TControl events 5850
TControl fields 5733
TControl Members 5725
TControl methods 5735
TControl properties 5796
Text 5842
Top 5844
UndockHeight 5845
UndockWidth 5846
Update 5789
UpdateBoundsRect 5789
UpdateExplicitBounds 5789
UseRightToLeftAlignment 5790
UseRightToLeftReading 5790
UseRightToLeftScrollBar 5791
Visible 5847
VisibleChanging 5791
WheelAccumulator 5849
Width 5849
WindowProc 5872
WindowText 5850
WndProc 5791
TControlAction class 5877
about TControlAction class 5877
AutoCheck 5881
Caption 5882
Checked 5882
DropdownMenu 5883
Enabled 5884
EnableDropdown 5884
GroupIndex 5884
HelpContext 5885
HelpKeyword 5885
HelpType 5886
Hint 5886
ImageIndex 5886
OnExecute 5888
OnHint 5890
OnUpdate 5890
PopupMenu 5887
SecondaryShortCuts 5887
ShortCut 5887
TControlAction events 5888
TControlAction Members 5877
TControlAction properties 5881
Visible 5888
TControlActionLink class 5892
about TControlActionLink class 5892
AssignClient 5895
DoShowHint 5896
FClient 5894
IsCaptionLinked 5896
IsDropdownMenuLinked 5896
IsEnabledLinked 5897
IsEnableDropdownLinked 5897
IsHelpLinked 5897
IsHintLinked 5898
IsOnExecuteLinked 5898
IsPopupMenuLinked 5898
IsVisibleLinked 5899
SetCaption 5899
SetDropdownMenu 5899
SetEnabled 5900
SetEnableDropdown 5900
SetHelpContext 5900
SetHelpKeyword 5901
SetHelpType 5901
SetHint 5901
SetOnExecute 5902
SetPopupMenu 5902

SetVisible 5902
TControlActionLink fields 5894
TControlActionLink Members 5892
TControlActionLink methods 5895
TControlBar class 12316
about TControlBar class 12316
Align 12328
Anchors 12332
AutoDock 12333
AutoDrag 12333
AutoSize 12333
BevelEdges 12334
BevelInner 12334
BevelKind 12335
BevelOuter 12335
BevelWidth 12336
BorderWidth 12336
Canvas 12336
Color 12338
Constraints 12339
CornerEdge 12340
DockSite 12340
DragCursor 12341
DragKind 12341
DragMode 12342
DrawingStyle 12343
Enabled 12343
GradientDirection 12344
GradientEndColor 12344
GradientStartColor 12344
OnAlignInsertBefore 12357
OnAlignPosition 12357
OnBandDrag 12358
OnBandInfo 12358
OnBandMove 12359
OnBandPaint 12359
OnBeginBandMove 12359
OnCanResize 12359
OnClick 12360
OnConstrainedResize 12362
OnContextPopup 12362
OnDblClick 12363
OnDockDrop 12363
OnDockOver 12365
OnDragDrop 12367
OnDragOver 12370
OnEndBandMove 12372
OnEndDock 12372
OnEndDrag 12373
OnEnter 12374
OnExit 12376
OnGetSiteInfo 12378
OnMouseActivate 12380
OnMouseDown 12381
OnMouseEnter 12383
OnMouseLeave 12383
OnMouseMove 12384
OnMouseUp 12386
OnPaint 12388
OnResize 12389
OnStartDock 12389
OnStartDrag 12390
OnUnDock 12390
ParentBackground 12345
ParentColor 12345
ParentCtl3D 12345
ParentFont 12346
ParentShowHint 12346
Picture 12347
PopupMenu 12347
RowSize 12351
RowSnap 12352
ShowHint 12352
TabOrder 12353
TabStop 12354
TControlBar events 12356
TControlBar Members 12316
TControlBar properties 12327

Visible 12354
TControlCanvas class 5903
about TControlCanvas class 5903
Control 5907
CreateHandle 5906
Destroy 5906
FreeHandle 5906
TControlCanvas Members 5903
TControlCanvas methods 5906
TControlCanvas properties 5907
UpdateTextFlags 5907
TControlCollection class 12393
about TControlCollection class 12393
Add 12396
AddControl 12396
ControlItems 12399
Controls 12399
Create 12395
GetControl 12397
GetControlItem 12397
GetItem 12397
IndexOf 12397
Items 12400
Owner 12398
RemoveControl 12398
SetControl 12398
SetItem 12398
TControlCollection Members 12394
TControlCollection methods 12396
TControlCollection properties 12399
Update 12398
TControlItem class 12400
about TControlItem class 12400
AssignTo 12403
Column 12404
ColumnSpan 12404
Control 12405
Create 12402
Destroy 12402
GridPanel 12405
InternalSetLocation 12403
Pushed 12405
Row 12405
RowSpan 12406
SetLocation 12404
TControlItem Members 12400
TControlItem methods 12403
TControlItem properties 12404
TControlRequirements class 25185
about TControlRequirements class 25185
Create 25186
DefineFunction 25187
DefineVariable 25187
Destroy 25186
IsFunctionDefined 25188
IsVariableDefined 25188
NeedMultipartForm 25189
RequiresMultipartForm 25189
TControlRequirements Members 25186
TControlRequirements methods 25187
TControlRequirements properties 25189
TControlScrollBar class 13744
about TControlScrollBar class 13744
Assign 13746
ButtonSize 13748
ChangeBiDiPosition 13746
Color 13748
Increment 13748
IsScrollBarVisible 13747
Kind 13750
Margin 13750
ParentColor 13751
Position 13752
Range 13753
ScrollPos 13756
Size 13756
Smooth 13756
Style 13757

TControlScrollBar Members 13744
TControlScrollBar methods 13746
TControlScrollBar properties 13747
ThumbSize 13757
Tracking 13757
Visible 13758
TConversion class 3582
about TConversion class 3582
ConvertReadStream 3583
ConvertWriteStream 3584
TConversion Members 3582
TConversion methods 3583
TConvTypeFactor class 6382
about TConvTypeFactor class 6382
Create 6383
Factor 6385
FromCommon 6384
TConvTypeFactor Members 6382
TConvTypeFactor methods 6384
TConvTypeFactor properties 6385
ToCommon 6385
TConvTypeInfo class 6386
about TConvTypeInfo class 6386
ConvFamily 6390
ConvType 6390
Create 6388
Description 6391
FromCommon 6389
TConvTypeInfo Members 6387
TConvTypeInfo methods 6388
TConvTypeInfo properties 6390
ToCommon 6389
TConvTypeProcs class 6391
about TConvTypeProcs class 6391
Create 6393
FromCommon 6393
TConvTypeProcs Members 6392
TConvTypeProcs methods 6393
ToCommon 6394
TCookie class 15092
about TCookie class 15092
AssignTo 15094
Create 15093
Domain 15095
Expires 15095
GetHeaderValue 15095
HeaderValue 15096
Name 15096
Path 15097
Secure 15097
TCookie Members 15092
TCookie methods 15094
TCookie properties 15095
Value 15097
TCookieCollection class 15098
about TCookieCollection class 15098
Add 15100
Create 15100
GetCookie 15101
Items 15101
SetCookie 15101
TCookieCollection Members 15098
TCookieCollection methods 15100
TCookieCollection properties 15101
WebResponse 15102
TCoolBand class 3584
about TCoolBand class 3584
Assign 3587
Bitmap 3589
BorderStyle 3589
Break 3590
Color 3590
Control 3590
Create 3586
Destroy 3586
FixedBackground 3591
FixedSize 3591
GetDisplayName 3587

Height 3591
HorizontalOnly 3592
ImageIndex 3592
MinHeight 3592
MinWidth 3593
ParentBitmap 3593
ParentBitmapChanged 3588
ParentColor 3593
ParentColorChanged 3588
TCoolBand Members 3585
TCoolBand methods 3587
TCoolBand properties 3588
Text 3594
Visible 3595
Width 3596
TCoolBands class 3596
about TCoolBands class 3596
Add 3599
CoolBar 3601
Create 3598
FindBand 3599
GetOwner 3599
HaveGraphic 3600
Items 3601
TCoolBands Members 3597
TCoolBands methods 3599
TCoolBands properties 3601
Update 3600
TCoolBar class 3603
about TCoolBar class 3603
Align 3626
AlignControls 3616
Anchors 3626
AutoSize 3627
BandBorderStyle 3627
BandMaximize 3628
Bands 3628
Bitmap 3630
BorderWidth 3630
CanAutoSize 3617
Change 3617
Color 3631
Constraints 3631
Create 3614
CreateParams 3617
CreateWnd 3618
Ctl3D 3632
Destroy 3615
DockSite 3632
DragCursor 3633
DragKind 3634
DragMode 3634
EdgeBorders 3636
EdgeInner 3636
EdgeOuter 3637
Enabled 3637
FixedOrder 3637
FixedSize 3638
FlipChildren 3618
Font 3638
GetPalette 3619
HitTest 3619
Images 3639
Loaded 3619
Notification 3620
OnChange 3650
OnClick 3650
OnContextPopup 3652
OnDblClick 3653
OnDockDrop 3653
OnDockOver 3655
OnDragDrop 3657
OnDragOver 3660
OnEndDock 3662
OnEndDrag 3662
OnGetSiteInfo 3663
OnMouseActivate 3665
OnMouseDown 3666

OnMouseEnter 3669
OnMouseLeave 3669
OnMouseMove 3669
OnMouseUp 3672
OnResize 3674
OnStartDock 3674
OnStartDrag 3675
OnUnDock 3676
PaintWindow 3620
ParentColor 3640
ParentFont 3640
ParentShowHint 3641
PopupMenu 3641
ShowHint 3646
ShowText 3647
TCoolBar events 3649
TCoolBar Members 3604
TCoolBar methods 3616
TCoolBar properties 3625
Vertical 3647
Visible 3648
WndProc 3621
TCorbaDispatchStub class 6433
about TCorbaDispatchStub class 6433
GetIDsOfNames 6435
GetTypeInfo 6435
GetTypeInfoCount 6435
Invoke 6435
TCorbaDispatchStub Members 6433
TCorbaDispatchStub methods 6435
TCorbaFactory class 6436
about TCorbaFactory class 6436
Create 6437
CreateInstance 6439
CreateInterface 6439
Destroy 6438
Execute 6439
GetImplementation 6439
GetSkeleton 6440
GetTypeInfo 6440
InstanceName 6440
Instancing 6441
InterfaceName 6441
RegisterFactory 6440
RepositoryID 6441
TCorbaFactory Members 6436
TCorbaFactory methods 6438
TCorbaFactory properties 6440
ThreadModel 6441
TCorbaFactoryManager class 6442
about TCorbaFactoryManager class 6442
AddFactory 6443
Destroy 6442
Find 6443
RegisterFactories 6443
TCorbaFactoryManager Members 6442
TCorbaFactoryManager methods 6443
TCorbaImplementation class 6444
_AddRef 6446
_Release 6447
about TCorbaImplementation class 6444
Create 6445
FController 6445
FFactory 6446
FRefCount 6446
GetIDsOfNames 6447
GetTypeInfo 6447
GetTypeInfoCount 6447
Invoke 6448
ObjAddRef 6448
ObjQueryInterface 6448
ObjRelease 6448
QueryInterface 6448
TCorbaImplementation fields 6445
TCorbaImplementation Members 6444
TCorbaImplementation methods 6446
TCorbaInterfaceIDManager class 6449
about TCorbaInterfaceIDManager class 6449

FindGUID 6450
FindID 6450
RegisterInterface 6450
SearchGUID 6450
SearchID 6451
TCorbaInterfaceIDManager Members 6449
TCorbaInterfaceIDManager methods 6449
TCorbaListManager class 6451
about TCorbaListManager class 6451
BeginRead 6452
BeginWrite 6452
Create 6451
Destroy 6452
EndRead 6453
EndWrite 6453
TCorbaListManager Members 6451
TCorbaListManager methods 6452
TCorbaObjectFactory class 6453
about TCorbaObjectFactory class 6453
Create 6455
CreateInterface 6455
ImplementationClass 6456
TCorbaObjectFactory Members 6453
TCorbaObjectFactory methods 6455
TCorbaObjectFactory properties 6456
TCorbaSkeleton class 6456
about TCorbaSkeleton class 6456
Create 6458
Destroy 6458
Execute 6459
FSkeleton 6459
GetImplementation 6459
GetSkeleton 6460
InitSkeleton 6460
TCorbaSkeleton fields 6458
TCorbaSkeleton Members 6456
TCorbaSkeleton methods 6459
TCorbaSkeletonManager class 6460
about TCorbaSkeletonManager class 6460
CreateSkeleton 6461
RegisterSkeleton 6461
TCorbaSkeletonManager Members 6460
TCorbaSkeletonManager methods 6461
TCorbaStub class 6462
about TCorbaStub class 6462
Create 6463
Destroy 6464
FStub 6464
GetStub 6465
Hash 6465
IsA 6465
NonExistent 6465
SetPrincipal 6466
TCorbaStub fields 6464
TCorbaStub Members 6462
TCorbaStub methods 6465
TCorbaStubManager class 6466
about TCorbaStubManager class 6466
CreateStub 6467
RegisterStub 6467
TCorbaStubManager Members 6466
TCorbaStubManager methods 6467
TCRemoteDataModule class 6576
about TCRemoteDataModule class 6576
CRDMGetProviderNames 6581
Create 6580
Destroy 6580
GetProvider 6581
Lock 6582
ProviderCount 6584
Providers 6584
RegisterProvider 6582
TCRemoteDataModule Members 6577
TCRemoteDataModule methods 6580
TCRemoteDataModule properties 6584
UnLock 6583
UnRegisterProvider 6583
TCriticalSection class 22058

about TCriticalSection class 22058
Acquire 22061
Create 22059
Destroy 22060
Enter 22061
FSection 22060
Leave 22062
Release 22062
TCriticalSection fields 22060
TCriticalSection Members 22059
TCriticalSection methods 22060
TryEnter 22062
TCubeDim class 17754
about TCubeDim class 17754
Assign 17757
AssignBinTypeFormat 17758
AssignBinTypeTransform 17758
BinData 17762
BinFormat 17763
BinType 17763
Create 17756
DataSetTransform 17759
DefineProperties 17759
Destroy 17757
Dirty 17764
DoTransform 17759
GetBinValues 17760
GetLoaded 17760
InitializeRange 17760
IsBinData 17761
Loaded 17764
MonthTransform 17761
NotifyCollection 17761
OnTransform 17766
QuarterTransform 17761
SetLoaded 17762
StartDate 17765
StartValue 17765
TCubeDim events 17766
TCubeDim Members 17754
TCubeDim methods 17757
TCubeDim properties 17762
ValueCount 17765
wasActive 17766
YearTransform 17762
TCubeDims class 17766
about TCubeDims class 17766
Add 17769
Assign 17770
Create 17769
GetDirtyFlag 17770
GetOwner 17770
IsDirty 17771
Items 17771
TCubeDims Members 17767
TCubeDims methods 17769
TCubeDims properties 17771
tcUpperCase enumeration member 13623
TCurrencyArray class 17322
about TCurrencyArray class 17322
Add 17325
Assign 17325
Create 17324
Find 17326
GetItem 17326
IndexOf 17327
Items 17327
PutItem 17327
TCurrencyArray Members 17323
TCurrencyArray methods 17325
TCurrencyArray properties 17327
TCurrencyField class 6850
about TCurrencyField class 6850
Create 6856
currency 6857
TCurrencyField Members 6850
TCurrencyField properties 6857
TCurrentWebPageInfo class 25475

about TCurrentWebPageInfo class 25475
Create 25477
GetHasViewAccess 25478
GetLoginRequired 25478
GetPageDescription 25478
GetPageHREF 25479
GetPageIsPublished 25479
GetPageName 25479
GetPageTitle 25479
TCurrentWebPageInfo Members 25476
TCurrentWebPageInfo methods 25477
WebPageInfo 25480
TCustomAction class 297
about TCustomAction class 297
AssignTo 303
AutoCheck 308
Caption 308
Checked 308
Create 301
Destroy 302
DisableIfNoHandler 310
DoHint 303
Enabled 310
Execute 304
FImage 302
FMask 303
GroupIndex 311
HandleShortCut 306
HelpContext 311
HelpKeyword 311
HelpType 312
Hint 312
ImageIndex 313
OnHint 315
SavedEnabledState 313
SecondaryShortCuts 314
SetHelpContext 306
SetHelpKeyword 306
SetName 307
ShortCut 314
TCustomAction events 315
TCustomAction fields 302
TCustomAction Members 298
TCustomAction methods 303
TCustomAction properties 307
Visible 314
TCustomActionBar class 447
about TCustomActionBar class 447
ActionClient 479
ActionControls 479
ActionManager 480
AlignControls 462
AllowHiding 480
AutoSizing 480
AutoSizingChanged 462
BackgroundSize 462
CalcInsertPosition 462
Canvas 481
Caption 481
Clear 463
Color 483
ColorMap 484
ContextBar 485
Create 459
CreateControl 463
CreateControls 463
CustomAlignInsertBefore 463
CustomAlignPosition 464
Designable 485
DesignMode 486
DesignWndProc 465
Destroy 459
DoControlCreated 465
DoDropActions 465
DoDropCategory 465
DoDropItem 466
DoGetControlClass 466
DragDrop 466

DragOver 466
DrawBackground 467
FDefaultColorMap 460
FindAccelItem 467
FindFirst 467
FindFirstVisibleItem 468
FindItemAt 468
FindItemCaption 468
FindLast 468
FindLastVisibleItem 469
FindLeastUsedItem 469
FindNearestControl 469
FindNext 470
FindNextVisibleItem 470
FindPrevious 470
FindPreviousVisibleItem 470
Font 486
FSavedWrapState 460
GetActionControl 471
GetBannerWidth 471
GetBarHeight 471
GetBarWidth 471
GetControlClass 471
GetDefaultColorMapClass 472
GetDesignMode 472
GetPopupMenuClass 472
HasItems 472
HorzMargin 487
HorzSeparator 487
HRowCount 488
ItemCount 473
Items 488
Loaded 473
NCPaint 473
Notification 473
OnControlCreated 491
OnGetControlClass 492
OnPaint 492
Orientation 488
Paint 474
PaintWindow 474
PersistentHotKeys 489
RecreateControls 475
Reset 475
SetActionClient 475
SetAutoSizing 475
SetBiDiMode 475
SetBounds 476
SetColorMap 476
SetDesignMode 476
SetOrientation 477
SetPersistentHotKeys 477
SetSpacing 477
SetupDefaultColorMap 477
Spacing 489
Style 477
TCustomActionBar events 491
TCustomActionBar fields 460
TCustomActionBar Members 447
TCustomActionBar methods 460
TCustomActionBar properties 478
VertMargin 490
VertSeparator 490
VisibleChanging 478
VRowCount 490
Widest 491
TCustomActionBarColorMap class 492
about TCustomActionBarColorMap class 492
Add 497
Assign 497
BtnFrameColor 500
BtnSelectedColor 500
BtnSelectedFont 501
Color 501
Create 496
DesignFocus 501
Destroy 496
DisabledColor 501

DisabledFontColor 502
DisabledFontShadow 502
FontColor 502
FrameBottomRightInner 502
FrameBottomRightOuter 503
FrameTopLeftInner 503
FrameTopLeftOuter 503
HighlightColor 503
HotColor 504
HotFontColor 504
MenuColor 504
Notification 498
OnColorChange 506
Remove 498
SelectedColor 505
SelectedFontColor 505
SetColor 499
SetFontColor 499
ShadowColor 505
TCustomActionBarColorMap events 506
TCustomActionBarColorMap Members 493
TCustomActionBarColorMap methods 497
TCustomActionBarColorMap properties 499
UnusedColor 505
UpdateActionBars 499
UpdateColors 499
TCustomActionClass class 506
about TCustomActionClass class 506
TCustomActionCombo class 118
about TCustomActionCombo class 118
BeginAutoDrag 131
ComboWndProc 131
DesignWndProc 132
DragDrop 132
DragOver 132
TCustomActionCombo Members 118
TCustomActionCombo methods 131
TCustomActionControl class 507
about TCustomActionControl class 507
ActionBar 528
ActionClient 529
BeginAutoDrag 517
CalcBounds 518
CalcLayout 518
Caption 529
Click 518
CMDesignHitTest 519
CMFontChanged 519
CMItemDropPoint 519
CMItemSelected 519
Create 515
DesignWndProc 519
Destroy 516
DoDragDrop 520
DoStartDrag 520
DragDrop 520
DragOver 521
DrawBackground 521
DrawDesignFocus 521
DrawDragDropPoint 522
DrawGlyph 522
DrawLargeGlyph 522
DrawSeparator 522
DrawShadowedText 523
DrawText 523
DropPoint 532
GetAction 523
GetImageSize 523
GetSeparator 524
GetShowCaption 524
GetShowShortCut 524
GlyphLayout 532
GlyphPos 532
HasGlyph 524
InitiateAction 524
IsChecked 525
IsGrouped 525
Keyed 525

Margins 532
OnClick 535
Paint 526
PositionChanged 526
ResetUsageData 526
Selected 533
Separator 533
SetActionClient 526
SetDropPoint 527
SetGlyphLayout 527
SetSelected 527
ShowCaption 533
ShowShortCut 533
SmallIcon 534
Spacing 534
TCustomActionControl events 535
TCustomActionControl Members 507
TCustomActionControl methods 516
TCustomActionControl properties 528
TextBounds 534
Transparent 534
UpdateSelection 527
UpdateTextBounds 527
WMSetText 528
TCustomActionDockBar class 133
about TCustomActionDockBar class 133
CalcDockedWidth 146
Create 145
Dock 146
DoEndDock 146
DoStartDock 147
GetFloatingDockSiteClass 147
GetSiteInfo 147
Loaded 148
TCustomActionDockBar Members 133
TCustomActionDockBar methods 145
TCustomActionList class 315
about TCustomActionList class 315
ActionCount 324
Actions 325
AddAction 320
Change 320
Create 319
Destroy 319
ExecuteAction 321
GetChildren 321
GetEnumerator 321
Images 325
IsShortCut 322
Notification 322
OnChange 326
OnExecute 327
OnStateChange 327
OnUpdate 327
RemoveAction 323
SetChildOrder 323
SetImages 323
SetState 324
State 326
TCustomActionList events 326
TCustomActionList Members 316
TCustomActionList methods 319
TCustomActionList properties 324
UpdateAction 324
TCustomActionMainMenuBar class 661
about TCustomActionMainMenuBar class 661
CanFocus 677
CloseMenu 677
CMWininichange 678
Create 675
CreateControl 678
CreateControls 678
CustomAlignPosition 678
DesignWndProc 679
Destroy 676
DoneMenuLoop 679
FWindowMenuItem 676
GetActiveMDIChild 679

Inactive 683
MainWndHook 680
MDIChildMaximized 680
MDIChildRestored 680
NewPopup 680
Notification 681
PersistentHotKeys 684
ProcessMouseMsg 681
RefreshMDIMenu 681
SetDesignMode 681
SetupWindowMenu 682
Shadows 684
TCustomActionMainMenuBar fields 676
TCustomActionMainMenuBar Members 662
TCustomActionMainMenuBar methods 676
TCustomActionMainMenuBar properties 683
TrackMenu 682
WindowMenu 684
WMKeyDown 682
WMNCHitTest 682
WMSysCommand 683
WMSysKeyDown 683
WMSysKeyUp 683
TCustomActionManager class 537
about TCustomActionManager class 537
ActionBars 553
AddAction 543
AddCategory 543
AddSeparator 544
AfterConstruction 544
Assign 544
Change 545
Create 541
CreateActionBars 545
DefaultActionBars 554
DefineProperties 545
DeleteActionItems 546
DeleteItem 546
Destroy 542
FileName 554
FindActionBar 547
FindActionClient 547
FindClient 547
FindItemByAction 547
FindItemByCaption 548
GetActionBarItemClass 548
GetActionBarsClass 548
GetActionClientItemClass 548
GetActionClientsClass 549
GetUsage 549
HasLinkedActionLists 549
Images 554
LinkedActionLists 555
Loaded 549
LoadFromFile 550
LoadFromStream 550
Loading 555
LoadMenu 551
Notification 551
OnGetBarItemClass 557
OnGetBarsClass 558
OnGetClientItemClass 558
OnGetClientsClass 558
OnStyleChanged 558
PrioritySchedule 555
ResetActionBar 551
ResetUsageData 551
SaveToFile 552
SaveToStream 552
SetImages 552
SetState 553
SetupActionBars 553
Style 556
TCustomActionManager events 557
TCustomActionManager Members 537
TCustomActionManager methods 542
TCustomActionManager properties 553
TCustomActionMenuBar class 685

about TCustomActionMenuBar class 685
Animate 701
AnimateDuration 716
AnimationStyle 716
AutoFocus 717
CleanupStack 701
ClearSubMenus 702
CloseMenu 702
Create 698
CreateControl 702
CreatePopup 702
DelayItem 717
Destroy 698
DoGetPopupClass 703
DoItemClicked 703
DoItemKeyed 703
DoItemSelected 703
DoMenuDelay 703
DoMouseIdle 704
DoneMenuLoop 704
DoPopup 704
ExecAction 704
Expand 704
Expandable 717
ExpandDelay 717
Expanded 718
ExpandTimer 705
FCachedMenu 699
FChildMenu 699
FExpandTimer 700
FPopupTimer 700
FSelectedItem 700
GetDefaultSounds 705
GetDesignMode 705
GetMouseHoverItem 705
GetPopupClass 706
GetSelected 706
Idle 706
InitMenuLoop 706
InMenuLoop 718
IsDesignMsg 707
Loaded 707
MouseActivate 707
MouseControl 718
NewPopup 707
Notification 708
OnEnterMenuLoop 720
OnExitMenuLoop 721
OnGetPopupClass 721
OnPopup 721
ParentControl 718
ParentMenu 719
PopupStack 719
ProcessMenuLoop 708
ProcessMessage 708
ProcessMessages 709
ProcessMouseMsg 709
RecreateControls 709
RootMenu 719
Select 709
Selected 719
SelectItem 710
SetDesignMode 710
SetParent 710
SetPersistentHotKeys 711
Style 711
TCustomActionMenuBar events 720
TCustomActionMenuBar fields 699
TCustomActionMenuBar Members 685
TCustomActionMenuBar methods 700
TCustomActionMenuBar properties 716
TrackMenu 711
UseSystemFont 720
WndProc 711
TCustomActionPopupMenu class 721
about TCustomActionPopupMenu class 721
AddEdges 737
CanAutoSize 737

CloseMenu 737
CMMouseLeave 738
CMVisibleChanged 738
Create 735
CreateControl 738
CreateControls 738
CreateParams 739
DesignWndProc 739
Destroy 736
DisplayShadow 739
DrawBackground 740
ExecAction 740
Expand 740
ExpandBtn 744
ExpandClick 740
GetClientRect 741
GetExpandBtnClass 741
HideShadow 741
NewPopup 741
Notification 742
Popup 742
PositionPopup 742
SetColorMap 743
SetOrientation 743
SetupExpandBtn 743
Shadow 745
TCustomActionPopupMenu Members 722
TCustomActionPopupMenu methods 736
TCustomActionPopupMenu properties 744
VisibleChanging 743
WMKeyDown 744
WMPrint 744
WMPrintClient 744
TCustomActionPopupMenuEx class 850
about TCustomActionPopupMenuEx class 850
DoPopup 851
ExecAction 851
GetPopupClass 851
LoadMenu 851
TCustomActionPopupMenuEx Members 850
TCustomActionPopupMenuEx methods 850
TCustomActionToolBar class 148
about TCustomActionToolBar class 148
AutoSizingChanged 163
CalcButtonWidth 163
CalcDockedWidth 163
CMColorchanged 163
Create 161
CreateControl 163
CreateControls 164
Destroy 162
DisableHiding 164
DoDropCategory 164
DragDrop 164
DrawBackground 165
DrawSeparator 165
EnableHiding 165
GetControlClass 165
GetPopupClass 166
GetScrollBtnClass 166
HiddenCount 168
HideUnusedItems 166
Reset 166
ScrollBtnClick 166
SetBounds 167
SetOrientation 167
SetupDropDownBtn 167
TCustomActionToolBar Members 149
TCustomActionToolBar methods 162
TCustomActionToolBar properties 168
TCustomActiveForm class 13759
about TCustomActiveForm class 13759
ActiveControl 13775
Anchors 13776
AutoScroll 13777
AutoSize 13777
AxBorderStyle 13777
BorderWidth 13778

Caption 13778
Color 13781
Constraints 13782
Create 13773
CreateParams 13773
Font 13782
Height 13783
HorzScrollBar 13784
KeyPreview 13785
OldCreateOrder 13788
OnActivate 13799
OnClick 13803
OnContextPopup 13805
OnCreate 13805
OnDblClick 13807
OnDeactivate 13807
OnDestroy 13808
OnDragDrop 13809
OnDragOver 13811
OnKeyDown 13814
OnKeyPress 13816
OnKeyUp 13817
OnMouseActivate 13819
OnMouseDown 13820
OnMouseEnter 13822
OnMouseLeave 13823
OnMouseMove 13823
OnMouseUp 13825
OnPaint 13827
Padding 13789
PixelsPerInch 13789
PopupMenu 13789
PrintScale 13794
Scaled 13794
ShowHint 13795
TCustomActiveForm events 13798
TCustomActiveForm Members 13760
TCustomActiveForm methods 13773
TCustomActiveForm properties 13774
VertScrollBar 13796
Visible 13797
WantChildKey 13774
Width 13798
TCustomAdapter class 1578, 24139
about TCustomAdapter class 1578, 24139
Actions 24181
AddAdapterModeHiddenField 24149
AddOrigValueHiddenFields 24149
AddRecordKeysHiddenFields 24149
CanAddActionClass 24150
CanAddFieldClass 24150
Changed 1582
CheckModifyAccess 24150
ClearAdapterValues 24151
ClearErrors 24151
ConnectOleObject 1582
Create 1580, 24146
CreateActionRequestContext 24151
CreateErrorsList 24152
Data 24181
Destroy 1581, 24146
EchoActionFieldValues 24182
EncodeActionParams 24152
EndIterator 24152
Errors 24182
ExecuteAccess 24183
ExtractRequestParams 24153
FindAction 24153
FindField 24154
FindNamedAction 24154
FindNamedField 24154
FindNamedVariable 24155
FindVariable 24155
FindVariableContainer 24155
GetActionsClass 24156
GetActionsList 24156
GetAdapterActions 24156
GetAdapterErrors 24157

wwww

GetAdapterErrorsList 24157
GetAdapterFields 24157
GetAdapterHiddenFields 24157
GetAdapterHiddenRecordFields 24158
GetChildren 24158
GetDataClass 24158
GetEchoActionFieldValues 24159
GetFieldsList 24159
GetHiddenFieldOptions 24159
GetIteratorIndex 24160
GetVariable 24160
GetVariableCount 24161
GetVariableName 24161
GetVisibleActions 24161
GetVisibleFields 24161
GetWebSnapObjClass 24162
HasExecuteAccess 24162
HasModifyAccess 24162
HasViewAccess 24162
ImplCanAddActionClass 24163
ImplCanAddFieldClass 24163
ImplClearAdapterValues 24163
ImplEndIterator 24164
ImplFindVariable 24164
ImplFindVariableContainer 24164
ImplGetActionsList 24165
ImplGetEchoActionFieldValues 24165
ImplGetFieldsList 24165
ImplGetIteratorIndex 24166
ImplGetVariable 24166
ImplGetVariableCount 24167
ImplGetVariableName 24167
ImplGetVisibleActions 24167
ImplGetVisibleFields 24167
ImplHasExecuteAccess 24168
ImplHasModifyAccess 24168
ImplHasViewAccess 24169
ImplInIterator 24169
ImplNextIteration 24169
ImplNotifyActivate 24170
ImplNotifyDeactivate 24170
ImplSetActions 24170
ImplSetData 24171
ImplSetEchoActionFieldValues 24171
ImplStartIterator 24171
InIterator 24172
IsActionInUse 24172
IsFieldInUse 24172
ListHiddenFields 24173
ListHiddenRecordFields 24173
ModifyAccess 24183
NextIteration 24173
Notification 24174
NotifyActivate 24174
NotifyAdapterChange 24174
NotifyDeactivate 24175
NotifyList 24183
OnAfterExecuteAction 24184
OnAfterGetActionResponse 24185
OnAfterUpdateFields 24185
OnAfterValidateFields 24186
OnBeforeExecuteAction 24186
OnBeforeGetActionResponse 24186
OnBeforeUpdateFields 24187
OnBeforeValidateFields 24187
OnGetActionParams 24187
OnIterateRecords 24188
ReleaseOleObject 1582
SetActions 24175
SetData 24175
SetEchoActionFieldValues 24176
StartIterator 24176
TCustomAdapter events 24184
TCustomAdapter fields 1581
TCustomAdapter Members 1579, 24141
TCustomAdapter methods 1581, 24147
TCustomAdapter properties 24181
Update 1583

UpdateFieldsExecuteValues 24177
UpdateFieldsGetAnyChanges 24177
UpdateFieldsUpdateValues 24178
UpdateFieldsValidateValues 24178
UpdateRecords 24179
UpdateRecordsGetAnyChanges 24179
UpdateRecordsUpdateValues 24180
UpdateRecordsValidateValues 24180
Updating 1581
ViewAccess 24183
VisibleActions 24184
VisibleFields 24184
WebComponentsChange 24180
TCustomAdapterAction class 24188
about TCustomAdapterAction class 24188
ActionName 24198
ImplExecuteActionRequest 24195
ImplGetActionName 24195
ImplGetAdapterItemRequestParams 24196
ImplGetSuccessOptions 24196
ImplSetActionName 24197
ImplWebEnabled 24197
OnAfterExecute 24198
OnAfterGetResponse 24199
OnBeforeExecute 24199
OnBeforeGetResponse 24200
OnExecute 24200
OnGetEnabled 24200
SetActionName 24197
TCustomAdapterAction events 24198
TCustomAdapterAction Members 24190
TCustomAdapterAction methods 24195
TCustomAdapterAction properties 24198
TCustomAdapterActionButton class 25189
about TCustomAdapterActionButton class 25189
ActionName 25200
Align 25201
Create 25194
Destroy 25195
DisplayColumns 25201
DisplayType 25202
ErrorPageName 25202
FindAssociatedAction 25195
GetActionName 25196
GetAdapterModeOfAdapter 25196
GetAdapterVariableName 25196
GetCaption 25196
GetControlRequirements 25197
GetDesigntimeWarnings 25197
GetHTMLFormVarName 25197
GetLayoutAttributes 25197
HideOptions 25202
ImageHeight 25203
ImageProducer 25203
ImageWidth 25204
ImplContent 25198
ImplGetControlRequirements 25198
ImplRestoreDefaults 25198
Notification 25199
PageName 25204
RestoreDefaults 25199
SetActionName 25199
SetCaption 25200
TCustomAdapterActionButton Members 25190
TCustomAdapterActionButton methods 25195
TCustomAdapterActionButton properties 25200
VAlign 25204
TCustomAdapterBooleanField class 24201
about TCustomAdapterBooleanField class 24201
ConvertValue 24208
GetDisplayStyleType 24208
GetInputStyleType 24208
ImplGetValue 24209
ImplIsEqual 24209
ImplUpdateValue 24209
IsEqual 24210
OnGetValue 24210
OnUpdateValue 24211

TCustomAdapterBooleanField events 24210
TCustomAdapterBooleanField Members 24202
TCustomAdapterBooleanField methods 24207
TCustomAdapterCommandColumn class 25205
about TCustomAdapterCommandColumn class 25205
AdapterActionsListHelper 25215
AddDefaultCommands 25215
Align 25216
Caption 25216
CaptionAttributes 25217
Create 25210
Custom 25217
Destroy 25211
DisplayColumns 25217
FormatColumnData 25211
FormatColumnHeading 25212
GetAdapterDisplayCharacteristics 25212
GetLayoutAttributes 25212
ImplCanAddClass 25213
ImplContent 25213
NotifyAdapterChange 25214
Style 25218
StyleRule 25218
TCustomAdapterCommandColumn Members 25206
TCustomAdapterCommandColumn methods 25211
TCustomAdapterCommandColumn properties 25215
VAlign 25219
VisibleCommands 25219
TCustomAdapterCommandGroup class 25220
about TCustomAdapterCommandGroup class 25220
Adapter 25228
AdapterActionsListHelper 25229
AddDefaultCommands 25229
Create 25225
Custom 25229
Destroy 25225
GetAdapter 25226
GetAdapterFormLayout 25226
GetAdapterModeOfAdapter 25227
GetAdapterModeOfDisplay 25227
ImplCanAddClass 25227
ImplContent 25228
Style 25229
StyleRule 25230
TCustomAdapterCommandGroup Members 25221
TCustomAdapterCommandGroup methods 25226
TCustomAdapterCommandGroup properties 25228
VisibleCommands 25230
TCustomAdapterDispatcher class 24865
about TCustomAdapterDispatcher class 24865
AddSpecialParams 24870
DoAdapterRequestNotHandled 24870
DoAfterDispatchAdapterRequest 24871
DoBeforeDispatchAdapterRequest 24871
EncodeAdapterItemRequestAsFieldValue 24872
EncodeAdapterItemRequestAsHREF 24872
GetAdapterItemRequestParams 24873
GetDispatchParams 24873
ImplEncodeAdapterItemRequestAsFieldValue 24874
ImplEncodeAdapterItemRequestAsHREF 24874
ImplGetAdapterItemRequestParams 24875
ImplHandleRequest 24876
OnActionRequestNotHandled 24877
OnAfterDispatchAction 24877
OnAfterDispatchImage 24878
OnBeforeDispatchAction 24878
OnBeforeDispatchImage 24878
OnGetAdapterRequestParams 24879
OnImageRequestNotHandled 24879
TCustomAdapterDispatcher events 24876
TCustomAdapterDispatcher Members 24866
TCustomAdapterDispatcher methods 24869
TCustomAdapterDisplayColumn class 25230
about TCustomAdapterDisplayColumn class 25230
CanAddClass 25237
ControlContent 25237
Create 25236
DisplayType 25239

FormatImage 25237
FormatListDisplay 25237
FormatTextDisplay 25238
ImageHeight 25240
ImageWidth 25240
ImplCanAddClass 25238
ImplFormatColumnHeading 25238
ImplGetControlRequirements 25238
ImplRestoreDefaults 25239
IsColumn 25239
TCustomAdapterDisplayColumn Members 25231
TCustomAdapterDisplayColumn methods 25236
TCustomAdapterDisplayColumn properties 25239
TCustomAdapterDisplayField class 25240
about TCustomAdapterDisplayField class 25240
AddAttributes 25247
ControlContent 25247
Create 25246
DisplayColumns 25249
DisplayRows 25249
DisplayType 25249
DisplayWidth 25250
GetLayoutAttributes 25247
ImageHeight 25250
ImageWidth 25250
ImplContent 25247
ImplGetControlRequirements 25248
ImplRestoreDefaults 25248
InputType 25251
MaxLength 25251
SelectRows 25251
TCustomAdapterDisplayField Members 25241
TCustomAdapterDisplayField methods 25246
TCustomAdapterDisplayField properties 25248
TextAreaWrap 25251
ViewMode 25252
TCustomAdapterEditColumn class 25252
about TCustomAdapterEditColumn class 25252
ImplContent 25258
TCustomAdapterEditColumn Members 25253
TCustomAdapterEditColumn methods 25258
TCustomAdapterErrorList class 25259
about TCustomAdapterErrorList class 25259
Adapter 25265
Content 25263
Create 25262
Destroy 25263
GetDesigntimeWarnings 25264
ImplContent 25264
Notification 25264
TCustomAdapterErrorList Members 25259
TCustomAdapterErrorList methods 25263
TCustomAdapterErrorList properties 25265
TCustomAdapterField class 24211
about TCustomAdapterField class 24211
DoOnValidateValue 24218
ImplGetValue 24219
ImplUpdateValue 24219
OnGetValue 24220
OnUpdateValue 24220
TCustomAdapterField events 24219
TCustomAdapterField Members 24213
TCustomAdapterField methods 24218
TCustomAdapterFieldGroup class 25265
about TCustomAdapterFieldGroup class 25265
Create 25271
Custom 25275
Destroy 25272
GetAdapterDisplayCharacteristics 25272
GetAdapterFormLayout 25272
GetAddAllFieldsItem 25273
GetAddFieldsDlgCaption 25273
GetAddFieldsItem 25273
GetLayoutAttributes 25273
GetNewFieldPrefix 25274
ImplCanAddClass 25274
ImplContent 25274
Style 25275

StyleRule 25276
TCustomAdapterFieldGroup Members 25267
TCustomAdapterFieldGroup methods 25272
TCustomAdapterFieldGroup properties 25275
TCustomAdapterFileField class 24220
about TCustomAdapterFileField class 24220
DoOnValidateValue 24228
FileCount 24231
FileNames 24231
GetDisplayStyleType 24228
GetFileNames 24229
GetInputStyleType 24229
GetValueCount 24229
GetValues 24230
ImplCheckValueChange 24230
ImplUpdateValue 24230
OnGetFileCount 24232
OnGetFileNames 24232
OnUploadFiles 24233
OnValidateFiles 24233
TCustomAdapterFileField events 24232
TCustomAdapterFileField Members 24222
TCustomAdapterFileField methods 24228
TCustomAdapterFileField properties 24231
TCustomAdapterForm class 25276
about TCustomAdapterForm class 25276
Create 25281
Custom 25285
Destroy 25281
FormMethod 25285
GetAdapterFormLayout 25282
GetFormRequirements 25282
GetHTMLFormName 25282
GetHTMLFormVarName 25283
GetLayoutAttributes 25283
ImplCanAddClass 25283
ImplContent 25284
Style 25285
StyleRule 25286
TCustomAdapterForm Members 25277
TCustomAdapterForm methods 25282
TCustomAdapterForm properties 25284
TCustomAdapterGotoPageAction class 24233
about TCustomAdapterGotoPageAction class 24233
GetDefaultActionName 24240
GetDisplayStyleType 24241
GetItemCount 24241
ImplExecuteActionRequest 24241
ImplGetAdapterItemRequestParams 24242
ImplGetDisplayLabel 24242
ImplGetIsAdapterActionList 24243
ImplIsDefaultAction 24243
ImplWebEnabled 24243
OnGetPageName 24244
TCustomAdapterGotoPageAction events 24244
TCustomAdapterGotoPageAction Members 24234
TCustomAdapterGotoPageAction methods 24240
TCustomAdapterGrid class 25286
about TCustomAdapterGrid class 25286
Create 25292
Destroy 25293
GetAdapterDisplayCharacteristics 25293
GetAdapterFormLayout 25294
GetAddAllFieldsItem 25294
GetAddFieldsDlgCaption 25294
GetAddFieldsItem 25294
GetNewFieldPrefix 25294
HeadingAttributes 25296
ImplCanAddClass 25295
ImplContent 25295
RowAttributes 25296
TableAttributes 25297
TCustomAdapterGrid Members 25288
TCustomAdapterGrid methods 25293
TCustomAdapterGrid properties 25296
TCustomAdapterImageField class 24244
about TCustomAdapterImageField class 24244
CreateRequestContext 24252

DoOnValidateValue 24253
GetAdapterImage 24253
GetAdapterItemRequestParams 24253
GetDisplayStyleType 24254
GetInputStyleType 24254
HandleRequest 24255
HREF 24259
ImplCheckValueChange 24255
ImplCreateRequestContext 24255
ImplGetValue 24256
ImplHandleRequest 24256
ImplUpdateValue 24257
ImplWebImageHREF 24257
OnGetHREF 24260
OnGetImage 24260
OnGetImageName 24260
OnGetParams 24261
OnUploadFile 24261
OnValidateFile 24261
ReadOnly 24259
RenderAdapterImage 24258
TCustomAdapterImageField events 24259
TCustomAdapterImageField Members 24246
TCustomAdapterImageField methods 24252
TCustomAdapterImageField properties 24258
WebImageHREF 24258
TCustomAdapterMemoField class 24262
about TCustomAdapterMemoField class 24262
DoOnValidateValue 24269
GetDisplayStyleType 24270
GetInputStyleType 24270
ImplCheckValueChange 24270
ImplGetValue 24271
ImplUpdateValue 24271
OnGetValue 24272
OnUpdateValue 24272
OnUploadFile 24273
TCustomAdapterMemoField events 24272
TCustomAdapterMemoField Members 24264
TCustomAdapterMemoField methods 24269
TCustomAdapterMultiValueField class 24273
about TCustomAdapterMultiValueField class 24273
DoOnValidateValue 24281
GetDisplayStyleType 24281
GetInputStyleType 24282
GetValueCount 24282
GetValues 24282
ImplHasValue 24283
ImplUpdateValue 24283
OnGetValueCount 24285
OnGetValues 24285
OnHasValue 24285
OnUpdateValues 24286
OnValidateValues 24286
TCustomAdapterMultiValueField events 24284
TCustomAdapterMultiValueField Members 24275
TCustomAdapterMultiValueField methods 24281
TCustomAdapterMultiValueField properties 24284
ValueCount 24284
Values 24284
TCustomAdapterPageAction class 24286
about TCustomAdapterPageAction class 24286
GetDisplayStyleType 24293
GetNewPage 24294
ImplExecuteActionRequest 24294
ImplGetAdapterItemRequestParams 24294
ImplIsDefaultAction 24295
ImplWebEnabled 24295
TCustomAdapterPageAction Members 24287
TCustomAdapterPageAction methods 24293
TCustomAdapterPageProducer class 19832
about TCustomAdapterPageProducer class 19832
Create 19838
Destroy 19839
GetTemplateFileName 19839
GetTemplateStream 19840
HTMLDoc 19842
HTMLFile 19843

SetHTMLDoc 19840
SetHTMLFile 19840
TCustomAdapterPageProducer Members 19833
TCustomAdapterPageProducer methods 19839
TCustomAdapterPageProducer properties 19842
WebContent 19841
TCustomAddRemoveItem class 745
about TCustomAddRemoveItem class 745
CalcBounds 754
IsActionVisible 755
MouseUp 755
SetEnabled 756
TCustomAddRemoveItem Members 746
TCustomAddRemoveItem methods 754
TCustomADODataSet class 1145
about TCustomADODataSet class 1145
ActivateTextFilter 1160
Active 1201
AfterCancel 1224
AfterClose 1226
AfterDelete 1227
AfterEdit 1230
AfterInsert 1231
AfterOpen 1231
AfterPost 1233
AfterRefresh 1233
AfterScroll 1233
AllocRecordBuffer 1160
AutoCalcFields 1203
BeforeCancel 1234
BeforeClose 1234
BeforeDelete 1235
BeforeEdit 1235
BeforeInsert 1236
BeforeOpen 1237
BeforePost 1238
BeforeRefresh 1239
BeforeScroll 1239
BookmarkValid 1160
CacheSize 1203
CancelBatch 1161
CancelUpdates 1162
CheckActive 1162
CheckFieldCompatibility 1162
ClearCalcFields 1162
Clone 1163
Command 1204
CommandText 1204
CommandTimeout 1204
CommandType 1205
CompareBookmarks 1163
Connection 1205
ConnectionPoint 1163
ConnectionString 1206
Create 1156
CreateBlobStream 1164
CursorLocation 1207
CursorType 1208
DataEvent 1164
DataSource 1208
DeactivateFilters 1164
DefChanged 1165
DefineProperties 1165
DeleteRecords 1165
DesignerData 1208
Destroy 1157
DestroyLookupCursor 1166
DoOnNewRecord 1166
EnableBCD 1209
EnableEvents 1166
EndOfRecordset 1167
ExecuteOptions 1209
FetchComplete 1167
FetchProgress 1167
FieldChangeComplete 1167
FieldDefs 1210
Filter 1212
Filtered 1214

FilterGroup 1214
FilterOnBookmarks 1168
FindRecord 1168
FreeRecordBuffer 1168
GetActiveRecBuf 1168
GetBlobFieldData 1169
GetBookmarkData 1169
GetBookmarkFlag 1169
GetCanModify 1169
GetCommandText 1170
GetDataSource 1170
GetDetailLinkFields 1170
GetFieldData 1171
GetFieldData method 1171
GetRecNo 1171
GetRecord 1172
GetRecordCount 1172
GetRecordSize 1172
GetStateFieldValue 1172
IndexDefs 1215
IndexFieldCount 1216
IndexFieldNames 1216
IndexFields 1217
IndexName 1217
InitOleDBAccess 1173
InternalAddRecord 1173
InternalCancel 1173
InternalClose 1173
InternalDelete 1173
InternalEdit 1174
InternalFirst 1174
InternalGetRecord 1174
InternalGotoBookmark 1174
InternalHandleException 1175
InternalInitFieldDefs 1175
InternalInitRecord 1175
InternalInsert 1175
InternalLast 1175
InternalOpen 1176
InternalPost 1176
InternalRefresh 1176
InternalRequery 1176
InternalSetSort 1177
InternalSetToRecord 1177
IsCursorOpen 1177
IsSequenced 1177
Loaded 1177
LoadFromFile 1178
Locate 1179
LocateRecord 1179
LockType 1218
Lookup 1180
MarshalOptions 1218
MasterChanged 1180
MasterDataLink 1218
MasterDisabled 1180
MasterFields 1219
MaxRecords 1219
MoveComplete 1180
NextRecordset 1181
OnCalcFields 1239
OnDeleteError 1240
OnEditError 1241
OnEndOfRecordset 1241
OnFetchComplete 1242
OnFetchProgress 1242
OnFieldChangeComplete 1243
OnFilterRecord 1244
OnMoveComplete 1244
OnNewRecord 1245
OnPostError 1245
OnRecordChangeComplete 1246
OnRecordsetChangeComplete 1247
OnRecordsetCreate 1247
OnWillChangeField 1248
OnWillChangeRecord 1248
OnWillChangeRecordset 1249
OnWillMove 1250

OpenCursor 1181
ParamCheck 1220
Parameters 1220
PrepareCursor 1181
Prepared 1221
Properties 1221
PSEndTransaction 1181
PSExecute 1182
PSExecuteStatement 1182
PSGetAttributes 1183
PSGetCommandText 1183
PSGetCommandType 1184
PSGetDefaultOrder 1184
PSGetIndexDefs 1185
PSGetKeyFieldsW 1185
PSGetParams 1185
PSGetQuoteCharW 1186
PSGetTableNameW 1186
PSGetUpdateException 1186
PSInTransaction 1187
PSIsSQLBased 1187
PSIsSQLSupported 1188
PSReset 1188
PSSetCommandText 1188
PSSetParams 1189
PSStartTransaction 1189
PSUpdateRecord 1190
RecordChangeComplete 1190
Recordset 1221
RecordsetChangeComplete 1190
RecordsetState 1222
RecordStatus 1222
RefreshParams 1191
ReleaseLock 1191
Requery 1191
SaveToFile 1192
Seek 1192
SetBlockReadSize 1194
SetBookmarkData 1194
SetBookmarkFlag 1194
SetCommandText 1194
SetConnection 1194
SetConnectionFlag 1195
SetDataSource 1195
SetDetailFilter 1195
SetFieldData 1196
SetFieldData method 1195
SetFiltered 1196
SetFilterOptions 1196
SetFilterText 1196
SetParamsFromCursor 1197
SetRecNo 1197
Sort 1222
StoreDefs 1223
Supports 1197
TCustomADODataSet events 1223
TCustomADODataSet Members 1146
TCustomADODataSet methods 1157
TCustomADODataSet properties 1200
UpdateBatch 1197
UpdateIndexDefs 1198
UpdateRecordSetPosition 1198
UpdateStatus 1199
WillChangeField 1199
WillChangeRecord 1199
WillChangeRecordset 1200
WillMove 1200
TCustomApplicationAdapter class 24296
about TCustomApplicationAdapter class 24296
ApplicationTitle 24305
GetApplicationAdapter 24303
GetApplicationTitle 24303
GetWebSnapObjClass 24304
ImplCanAddActionClass 24304
ImplCanAddFieldClass 24304
ImplGetActionsList 24305
ImplGetFieldsList 24305
TCustomApplicationAdapter Members 24297

TCustomApplicationAdapter methods 24303
TCustomApplicationAdapter properties 24305
TCustomApplicationAdapterAction class 24306
about TCustomApplicationAdapterAction class 24306
Adapter 24312
TCustomApplicationAdapterAction Members 24307
TCustomApplicationAdapterAction properties 24312
TCustomApplicationEvents class 1347
about TCustomApplicationEvents class 1347
Activate 1351
CancelDispatch 1351
Create 1350
OnActionExecute 1352
OnActionUpdate 1354
OnActivate 1355
OnDeactivate 1356
OnException 1357
OnHelp 1358
OnHint 1359
OnIdle 1361
OnMessage 1362
OnMinimize 1363
OnModalBegin 1364
OnModalEnd 1364
OnRestore 1364
OnSettingChange 1365
OnShortCut 1366
OnShowHint 1367
TCustomApplicationEvents events 1352
TCustomApplicationEvents Members 1347
TCustomApplicationEvents methods 1351
TCustomArray class 17328
about TCustomArray class 17328
Add 17330
Assign 17330
BlankBoolVal 17334
BlankCount 17335
BlankDateVal 17335
BlankStringVal 17335
CompareProc 17335
ConvertVar 17330
Create 17329
DataType 17335
Destroy 17329
Duplicates 17336
GetCompProc 17330
GetCurrency 17331
GetDouble 17331
GetDups 17331
GetInteger 17331
GetItem 17332
GetMemberCount 17332
GetSort 17332
IndexOf 17332
IsBlank 17332
Items 17336
List 17336
MemberCount 17336
MemoryUsage 17333
SetCompProc 17333
SetDups 17333
SetItem 17333
SetSize 17334
SetSort 17334
Sorted 17337
TCustomArray Members 17328
TCustomArray methods 17329
TCustomArray properties 17334
TCustomBucketList class 5661
about TCustomBucketList class 5661
Add 5664
AddItem 5664
Assign 5665
BucketCount 5668
BucketFor 5665
Buckets 5669
Clear 5665
Data 5669

gggggg

DeleteItem 5666
Destroy 5663
Exists 5666
Find 5666
FindItem 5667
ForEach 5667
ForEach method 5667
Remove 5668
TCustomBucketList Members 5662
TCustomBucketList methods 5663
TCustomBucketList properties 5668
TCustomButtonControl class 168
about TCustomButtonControl class 168
AllowAllUp 186
CalcBounds 179
Caption 186
Click 179
CMDialogChar 180
CMEnabledChanged 180
CMMouseEnter 180
CMMouseLeave 180
CMTextChanged 180
Create 177
Down 189
DrawBackground 181
DrawFrame 181
DrawLargeGlyph 181
FFlat 178
Flat 189
FState 178
GetShowShortCut 181
Loaded 182
MouseDown 182
MouseInControl 189
MouseMove 183
MouseUp 183
Paint 184
SetActionClient 184
SetBounds 184
SetGlyphLayout 185
SetSelected 185
ShowCaption 189
ShowShortCut 190
State 190
TCustomButtonControl fields 178
TCustomButtonControl Members 169
TCustomButtonControl methods 178
TCustomButtonControl properties 185
TrackButton 190
WMLButtonDblClk 185
TCustomCheckBox class 21138
about TCustomCheckBox class 21138
Alignment 21152
AllowGrayed 21152
Click 21149
Create 21148
CreateParams 21149
CreateWnd 21150
GetChecked 21150
GetControlsAlignment 21151
SetChecked 21151
State 21153
TabStop 21154
TCustomCheckBox Members 21138
TCustomCheckBox methods 21149
TCustomCheckBox properties 21152
Toggle 21151
TCustomClientDataSet class 8373
about TCustomClientDataSet class 8373
ActivateFilters 8391
Active 8458
ActiveAggs 8460
AddDataPacket 8391
AddFieldDesc 8391
AddIndex 8391
AfterApplyUpdates 8497
AfterCancel 8498
AfterClose 8500

AfterDelete 8501
AfterEdit 8504
AfterExecute 8504
AfterGetParams 8505
AfterGetRecords 8506
AfterInsert 8507
AfterOpen 8507
AfterPost 8509
AfterRefresh 8509
AfterRowRequest 8509
AfterScroll 8510
Aggregates 8461
AggregatesActive 8461
AllocKeyBuffers 8392
AllocRecordBuffer 8393
AppendData 8393
ApplyRange 8393
ApplyUpdates 8394
AppServer 8462
AutoCalcFields 8463
BeforeApplyUpdates 8511
BeforeCancel 8512
BeforeClose 8512
BeforeDelete 8512
BeforeEdit 8513
BeforeExecute 8514
BeforeGetParams 8514
BeforeGetRecords 8515
BeforeInsert 8517
BeforeOpen 8518
BeforePost 8518
BeforeRefresh 8519
BeforeRowRequest 8520
BeforeScroll 8520
BookmarkValid 8395
Cancel 8395
CancelRange 8396
CancelUpdates 8396
ChangeCount 8463
Check 8397
CheckDetailRecords 8397
CheckProviderEOF 8397
CheckSetKeyMode 8397
ClearCalcFields 8398
CloneCursor 8398
CloneSource 8464
CloseCursor 8400
CommandText 8465
CompareBookmarks 8400
ConnectionBroker 8466
Constraints 8466
ConstraintsDisabled 8400
ConstraintsStored 8401
Create 8386
CreateBlobStream 8401
CreateDataSet 8401
Data 8467
DataConvert 8404
DataEvent 8404
DataRequest 8404
DataSetField 8469
DataSize 8469
DeactivateFilters 8405
DefChanged 8405
DefineProperties 8405
DeleteIndex 8406
Delta 8470
Destroy 8387
DestroyLookupCursor 8406
DisableConstraints 8406
DisableStringTrim 8470
DoAfterApplyUpdates 8407
DoAfterExecute 8407
DoAfterGetParams 8407
DoAfterGetRecords 8407
DoAfterRowRequest 8408
DoApplyUpdates 8408
DoBeforeApplyUpdates 8408

DoBeforeExecute 8408
DoBeforeGetParams 8408
DoBeforeGetRecords 8409
DoBeforeInsert 8409
DoBeforeRowRequest 8409
DoExecute 8409
DoGetRecords 8410
DoOnNewRecord 8410
DoRowRequest 8410
DSBase 8471
DSCursor 8471
EditKey 8410
EditRangeEnd 8411
EditRangeStart 8411
EmptyDataSet 8412
EnableConstraints 8412
Execute 8413
FetchBlobs 8414
FetchDetails 8414
FetchMoreData 8415
FetchOnDemand 8471
FetchParams 8415
FieldDefs 8472
FileName 8475
Filter 8476
Filtered 8478
FilterOptions 8478
FindKey 8416
FindNearest 8416
FindRecord 8417
FreeKeyBuffers 8417
FreeRecordBuffer 8417
GetAggregateValue 8418
GetAppServer 8418
GetBookmarkData 8418
GetBookmarkFlag 8418
GetCanModify 8418
GetCurrentRecord 8419
GetDataSource 8419
GetFieldData 8419
GetFieldData method 8419
GetGroupState 8420
GetIndexField 8421
GetIndexFieldCount 8421
GetIndexInfo 8421
GetIndexNames 8422
GetIsClone 8422
GetIsIndexField 8422
GetKeyBuffer 8423
GetKeyExclusive 8423
GetKeyFieldCount 8423
GetNextPacket 8423
GetOptionalParam 8425
GetRecNo 8425
GetRecord 8425
GetRecordCount 8426
GetRecordSize 8426
GetRemoteServer 8426
GetStateFieldValue 8426
GotoCurrent 8427
GotoKey 8427
GotoNearest 8427
GroupingLevel 8479
HasAppServer 8480
IndexDefs 8480
IndexFieldCount 8481
IndexFieldNames 8482
IndexFields 8482
IndexName 8483
InitKeyBuffer 8428
InitRecord 8428
InternalAddRecord 8428
InternalCancel 8429
InternalClose 8429
InternalDelete 8429
InternalEdit 8429
InternalFirst 8430
InternalGetOptionalParam 8430

InternalGotoBookmark 8430
InternalHandleException 8430
InternalInitFieldDefs 8430
InternalInitRecord 8431
InternalInsert 8431
InternalLast 8431
InternalOpen 8431
InternalPost 8432
InternalRefresh 8432
InternalSetOptionalParam 8432
InternalSetToRecord 8432
IsClone 8484
IsCursorOpen 8432
KeyExclusive 8484
KeyFieldCount 8484
KeySize 8485
Loaded 8433
LoadFromFile 8433
LoadFromStream 8435
Locate 8435
LocateRecord 8436
LogChanges 8485
Lookup 8436
MasterFields 8486
MasterSource 8486
MergeChangeLog 8437
Notification 8437
ObjectView 8487
OnCalcFields 8521
OnDeleteError 8522
OnEditError 8522
OnFilterRecord 8523
OnNewRecord 8523
OnPostError 8524
OnReconcileError 8524
OpenCursor 8438
PacketRecords 8488
Params 8489
Post 8438
PostKeyBuffer 8439
ProviderEOF 8491
ProviderName 8492
PSGetCommandText 8439
PSGetCommandType 8440
PSGetIndexDefs 8440
Ranged 8492
ReadDataPacket 8441
ReadOnly 8493
Reconcile 8441
RefreshInternalCalcFields 8441
RefreshRecord 8442
RemoteServer 8493
ResetAggField 8442
ResetCursorRange 8443
RevertRecord 8443
SavePoint 8494
SaveToFile 8443
SaveToStream 8445
SetAggregates 8445
SetAltRecBuffers 8446
SetAppServer 8446
SetBookmarkData 8446
SetBookmarkFlag 8446
SetCommandText 8447
SetCursorRange 8447
SetDataSetField 8447
SetFieldData 8447
SetFilterData 8448
SetFiltered 8448
SetFilterOptions 8448
SetFilterText 8448
SetIndexField 8448
SetKey 8449
SetKeyBuffer 8449
SetKeyExclusive 8449
SetKeyFieldCount 8450
SetKeyFields 8450
SetLinkRanges 8450

SetOnFilterRecord 8450
SetOptionalParam 8450
SetProvider 8451
SetProviderEOF 8452
SetRange 8452
SetRangeEnd 8453
SetRangeStart 8454
SetRecNo 8454
SetRemoteServer 8454
StatusFilter 8495
StoreDefs 8495
SwitchToIndex 8455
SyncCursors 8455
TCustomClientDataSet events 8496
TCustomClientDataSet Members 8374
TCustomClientDataSet methods 8387
TCustomClientDataSet properties 8456
UndoLastChange 8455
UpdateIndexDefs 8455
UpdateStatus 8456
WriteDataPacket 8456
XMLData 8496
TCustomColorBox class 12406
about TCustomColorBox class 12406
CloseUp 12419
ColorNames 12423
Colors 12424
Create 12419
CreateWnd 12420
DefaultColorColor 12424
DrawItem 12420
KeyDown 12420
KeyPress 12421
Loaded 12421
NoneColorColor 12424
OnGetColors 12426
PickCustomColor 12422
PopulateList 12422
Select 12422
Selected 12425
SetStyle 12422
Style 12425
TCustomColorBox events 12426
TCustomColorBox Members 12406
TCustomColorBox methods 12419
TCustomColorBox properties 12423
TCustomColorListBox class 12426
about TCustomColorListBox class 12426
ColorNames 12442
Colors 12443
Create 12438
CreateWnd 12439
DefaultColorColor 12443
DrawItem 12439
KeyDown 12440
KeyPress 12441
Loaded 12441
NoneColorColor 12443
OnGetColors 12446
PickCustomColor 12441
PopulateList 12442
Selected 12443
SetStyle 12442
Style 12445
TCustomColorListBox events 12446
TCustomColorListBox Members 12426
TCustomColorListBox methods 12438
TCustomColorListBox properties 12442
TCustomCombo class 21154
about TCustomCombo class 21154
AddItem 21170
AdjustDropDown 21171
Canvas 21184
Change 21171
Clear 21172
ClearSelection 21172
CloseUp 21172
ComboWndProc 21173

CopySelection 21173
Create 21166
CreateWnd 21174
DeleteSelected 21174
Destroy 21166
DestroyWindowHandle 21174
DropDown 21175
DropDownCount 21185
DroppedDown 21185
EditHandle 21185
EditWndProc 21175
FDefEditProc 21167
FDefListProc 21167
FDropHandle 21168
FDroppingDown 21168
FEditHandle 21168
FEditInstance 21168
FFocusChanged 21168
FIsFocused 21169
FListHandle 21169
FListInstance 21169
Focused 21175
FSaveIndex 21169
GetCount 21176
GetDroppedDown 21176
GetItemCount 21176
GetItemHt 21177
GetItemIndex 21177
GetItemsClass 21177
GetSelLength 21178
GetSelStart 21178
ItemCount 21186
ItemHeight 21186
Items 21187
ListHandle 21189
ListWndProc 21178
Loaded 21179
MaxLength 21189
OnChange 21197
OnCloseUp 21199
OnDropDown 21199
OnSelect 21201
ParentColor 21190
Select 21179
SelectAll 21180
SelLength 21190
SelStart 21193
SetDropDownCount 21180
SetDroppedDown 21180
SetItemHeight 21181
SetItemIndex 21181
SetItems 21181
SetMaxLength 21182
SetSelLength 21182
SetSelStart 21182
TabStop 21196
TCustomCombo events 21196
TCustomCombo fields 21167
TCustomCombo Members 21155
TCustomCombo methods 21169
TCustomCombo properties 21183
WndProc 21183
TCustomComboBox class 21201
about TCustomComboBox class 21201
AutoCloseUp 21220
AutoComplete 21220
AutoCompleteDelay 21220
AutoDropDown 21221
CharCase 21221
Create 21214
CreateParams 21215
CreateWnd 21215
Destroy 21214
DestroyWnd 21215
DrawItem 21216
GetItemCount 21216
GetItemHt 21216
GetItemsClass 21217

GetSelText 21217
KeyPress 21217
MeasureItem 21218
OnDrawItem 21224
OnMeasureItem 21225
SelectItem 21218
SelText 21222
SetStyle 21219
Sorted 21222
Style 21224
TCustomComboBox events 21224
TCustomComboBox Members 21202
TCustomComboBox methods 21214
TCustomComboBox properties 21219
WndProc 21219
TCustomComboBoxEx class 3678
about TCustomComboBoxEx class 3678
ActionChange 3692
AutoCompleteOptions 3699
CMColorChanged 3693
CMParentColorChanged 3693
CNNotify 3693
ComboExWndProc 3694
Create 3691
CreateParams 3694
CreateWnd 3694
Destroy 3691
DestroyWnd 3695
DropDownCount 3699
Focused 3695
GetActionLinkClass 3695
GetItemCount 3696
GetItemHt 3696
GetItemsClass 3696
Images 3700
ItemsEx 3700
Notification 3697
OnBeginEdit 3702
OnEndEdit 3702
SelText 3701
SetDropDownCount 3697
Style 3701
StyleEx 3701
TCustomComboBoxEx events 3702
TCustomComboBoxEx Members 3679
TCustomComboBoxEx methods 3691
TCustomComboBoxEx properties 3699
WMHelp 3697
WMLButtonDown 3698
WndProc 3698
TCustomComboBoxStrings class 21225
about TCustomComboBoxStrings class 21225
Clear 21228
ComboBox 21232
Delete 21229
Get 21229
GetCount 21230
GetObject 21230
IndexOf 21230
PutObject 21231
SetUpdateState 21231
TCustomComboBoxStrings Members 21226
TCustomComboBoxStrings methods 21228
TCustomComboBoxStrings properties 21232
TCustomComboControl class 191
about TCustomComboControl class 191
CalcBounds 201
Click 201
CMVisiblechanged 201
ComboBox 205
ComboClick 202
Create 199
Destroy 200
Paint 202
SetActionClient 202
SetBounds 202
SetDragMode 203
SetParent 204

TCustomComboControl Members 191
TCustomComboControl methods 200
TCustomComboControl properties 204
VisibleChanging 204
TCustomConnection class 6858
about TCustomConnection class 6858
AfterConnect 6874
AfterDisconnect 6875
BeforeConnect 6875
BeforeDisconnect 6875
Close 6863
Connected 6868
Create 6862
DataSetCount 6869
DataSets 6870
Destroy 6863
DoConnect 6864
DoDisconnect 6864
GetConnected 6865
GetDataSet 6865
GetDataSetCount 6865
Loaded 6866
LoginPrompt 6872
OnLogin 6873
Open 6866
RegisterClient 6866
SendConnectEvent 6867
SetConnected 6867
StreamedConnected 6874
TCustomConnection events 6874
TCustomConnection Members 6859
TCustomConnection methods 6863
TCustomConnection properties 6868
UnRegisterClient 6868
TCustomContentProducer class 15102
about TCustomContentProducer class 15102
Content 15106
ContentFromStream 15107
ContentFromString 15107
Dispatcher 15109
DispatcherComponent 15110
GetAppDispatcher 15108
Notification 15108
ProduceContent 15108
ProduceContentFromStream 15109
ProduceContentFromString 15109
SetAppDispatcher 15109
TCustomContentProducer Members 15103
TCustomContentProducer methods 15106
TCustomContentProducer properties 15109
TCustomControl class 5908
about TCustomControl class 5908
Canvas 5921
Create 5918
Destroy 5919
Paint 5920
PaintWindow 5920
TCustomControl Members 5908
TCustomControl methods 5920
TCustomControl properties 5921
TCustomControlAction class 5923
about TCustomControlAction class 5923
DropdownMenu 5927
EnableDropdown 5928
PopupMenu 5928
TCustomControlAction Members 5923
TCustomControlAction properties 5927
TCustomControlBar class 12446
about TCustomControlBar class 12446
AlignControls 12459
AutoDock 12467
AutoDrag 12468
AutoSize 12468
BevelKind 12469
CanAutoSize 12459
CornerEdge 12469
Create 12457
CreateParams 12460

Destroy 12458
DoBandMove 12460
DoBandPaint 12460
DoBeginBandMove 12461
DockOver 12461
DockSite 12469
DoEndBandMove 12461
DoPaletteChange 12461
DragControl 12462
DrawingStyle 12470
FlipChildren 12462
GetControlInfo 12463
GetPalette 12463
GetSiteInfo 12464
GradientDirection 12471
GradientEndColor 12471
GradientStartColor 12471
HitTest 12464
MouseDown 12465
MouseMove 12465
MouseUp 12465
OnBandDrag 12472
OnBandInfo 12473
OnBandMove 12473
OnBandPaint 12473
OnBeginBandMove 12474
OnEndBandMove 12474
OnPaint 12474
Paint 12466
PaintControlFrame 12466
Picture 12471
RowSize 12472
RowSnap 12472
StickControls 12467
TCustomControlBar events 12472
TCustomControlBar Members 12447
TCustomControlBar methods 12458
TCustomControlBar properties 12467
TCustomDataBlockInterpreter class 18898
about TCustomDataBlockInterpreter class 18898
AddDispatch 18900
CallCreateObject 18900
CallFreeObject 18900
CallGetIDsOfNames 18900
CallGetServerList 18901
CallInvoke 18901
CanCreateObject 18901
CreateObject 18901
InternalCreateObject 18901
InterpretData 18902
LockObject 18902
ReleaseObject 18902
RemoveDispatch 18902
StoreObject 18903
TCustomDataBlockInterpreter Members 18899
TCustomDataBlockInterpreter methods 18899
UnlockObject 18903
TCustomDataForm class 16493
about TCustomDataForm class 16493
GetHTMLFormTag 16498
ImplCanAddClass 16498
ImplContent 16499
TCustomDataForm Members 16494
TCustomDataForm methods 16498
TCustomDataGrid class 16499
about TCustomDataGrid class 16499
Create 16505
Destroy 16506
DisplayRows 16509
FormatTable 16506
HeadingAttributes 16509
ImplAddElements 16506
ImplCanAddClass 16507
ImplContent 16507
ImplIsMultipleRecordView 16508
RowAttributes 16510
SetHeadingAttributes 16508
SetRowAttributes 16508

SetTableAttributes 16508
TableAttributes 16510
TCustomDataGrid Members 16501
TCustomDataGrid methods 16506
TCustomDataGrid properties 16509
TCustomDataNavigator class 16511
about TCustomDataNavigator class 16511
Custom 16518
GetDefaultButtons 16516
ImplCanAddClass 16516
ImplContent 16516
ImplGetSubComponents 16517
Style 16518
StyleRule 16519
TCustomDataNavigator Members 16511
TCustomDataNavigator methods 16516
TCustomDataNavigator properties 16518
VisibleButtons 16517
TCustomDataSetAdapter class 7766
about TCustomDataSetAdapter class 7766
AdapterRowNotFound 7807
AddDefaultActions 7778
AddDefaultFields 7779
ApplyAdapterMode 7780
Create 7775
DataSet 7807
DataSetPrepared 7808
Destroy 7776
DetailAdapters 7808
EncodeActionParams 7780
EncodeActionParamsFlags 7781
ExtractRequestParams 7781
ExtractRequestParamsFlags 7782
FullyQualifyInputNames 7782
GetAdapterMode 7783
GetAdapterModeNames 7783
GetDataSet 7784
GetDefaultMode 7784
GetDesigntimeWarnings 7784
GetEOF 7785
GetFirstRecord 7785
GetHiddenFieldOptions 7786
GetKeyFields 7786
GetKeyParams 7787
GetKeyParamStrings 7787
GetMasterDetailAdapters 7788
GetNeedAdapterMode 7788
GetNextRecord 7789
GetRecordCount 7789
GetRecordIndex 7790
GetRecordKeys 7790
ImplCanAddActionClass 7790
ImplCanAddFieldClass 7791
ImplEndIterator 7792
ImplGetActionsList 7792
ImplGetFieldsList 7793
ImplGetIteratorIndex 7793
ImplGetRecordKeys 7794
ImplInIterator 7794
ImplNotifyDeactivate 7795
ImplSetEchoActionFieldValues 7795
ImplStartIterator 7795
InDefaultMode 7808
Locate 7796
LocateAndApply 7796
LocateKeyParams 7797
LocateParamsList 7809
LocateRequestRecord 7798
MasterAdapter 7809
Mode 7810
Notification 7798
OnDataSetStateChange 7798
OnPrepareDataSet 7810
PrepareDataSet 7799
RaiseFieldChangedByAnotherUser 7799
RaiseFieldNotFound 7800
RaiseNilDataSet 7800
RaiseRowNotFound 7800

RefreshAfterPost 7810
RestoreDefaultAdapterMode 7801
SetAdapterMode 7801
SetDataSet 7802
SetMode 7802
SilentLocate 7802
StringToMode 7803
TCustomDataSetAdapter events 7810
TCustomDataSetAdapter Members 7768
TCustomDataSetAdapter methods 7776
TCustomDataSetAdapter properties 7806
UpdateFieldsExecuteValues 7803
UpdateFieldsGetAnyChanges 7804
UpdateFieldsUpdateValues 7805
UpdateFieldsValidateValues 7805
TCustomDataSetAdapterAction class 7811
about TCustomDataSetAdapterAction class 7811
ActionModes 7821
ActionName 7821
Adapter 7822
Create 7818
DataSet 7822
GetAdapter 7819
GetDataSet 7819
ImplGetVisible 7819
ImplWebEnabled 7820
OnExecute 7822
OnGetEnabled 7823
TCustomDataSetAdapterAction events 7822
TCustomDataSetAdapterAction Members 7813
TCustomDataSetAdapterAction methods 7819
TCustomDataSetAdapterAction properties 7820
TCustomDataSetAdapterField class 7823
about TCustomDataSetAdapterField class 7823
GetAdapterValuesList 7831
GetInputStyleType 7831
GetNeedPrevValue 7832
GetPrevValue 7832
ImplCheckOrigValue 7833
ImplCheckValueChange 7833
ImplGetNeedPrevValue 7834
ImplGetPrevValue 7835
ImplGetValue 7835
ImplUpdateValue 7836
Notification 7836
TCustomDataSetAdapterField Members 7825
TCustomDataSetAdapterField methods 7831
TCustomDataSetAdapterField properties 7837
ValuesList 7837
TCustomDataSetAdapterImageField class 8120
about TCustomDataSetAdapterImageField class 8120
CheckOrUpdateValue 8128
Create 8127
CreateRequestContext 8129
GetAdapterImage 8130
GetAdapterItemRequestParams 8130
GetDataSetFieldValue 8131
GetDisplayStyleType 8131
GetInputStyleType 8132
HandleRequest 8132
ImplCheckValueChange 8133
ImplCreateRequestContext 8134
ImplHandleRequest 8134
ImplUpdateValue 8135
ImplWebImageHREF 8135
OnGetHREF 8137
OnGetImage 8137
RenderAdapterImage 8136
TCustomDataSetAdapterImageField events 8137
TCustomDataSetAdapterImageField Members 8122
TCustomDataSetAdapterImageField methods 8128
WebImageHREF 8136
TCustomDataSetAdapterMemoField class 7837
about TCustomDataSetAdapterMemoField class 7837
CheckOrUpdateValue 7845
GetInputStyleType 7845
ImplAsFormatted 7845
ImplCheckValueChange 7845

ImplGetValue 7846
ImplUpdateValue 7846
TCustomDataSetAdapterMemoField Members 7839
TCustomDataSetAdapterMemoField methods 7844
TCustomDataSetAdapterRowAction class 7846
about TCustomDataSetAdapterRowAction class 7846
Create 7853
ImplGetAdapterItemRequestParams 7854
ImplWebEnabled 7855
TCustomDataSetAdapterRowAction Members 7848
TCustomDataSetAdapterRowAction methods 7854
TCustomDataSetPageProducer class 11800
about TCustomDataSetPageProducer class 11800
Content 11804
DataSet 11806
DoTagEvent 11805
GetDataSet 11805
Notification 11805
SetDataSet 11806
TCustomDataSetPageProducer Members 11801
TCustomDataSetPageProducer methods 11804
TCustomDataSetPageProducer properties 11806
TCustomDataSetValuesList class 7855
about TCustomDataSetValuesList class 7855
Create 7863
DataSet 7869
DataSetPrepared 7870
Destroy 7864
GetDesigntimeWarnings 7864
GetNameFieldClass 7865
GetValueFieldClass 7865
ImplEndIterator 7866
ImplGetListName 7866
ImplGetListNameOfValue 7867
ImplGetListValue 7867
ImplNextIteration 7867
ImplNotifyDeactivate 7868
ImplStartIterator 7868
NameField 7870
Notification 7869
OnPrepareDataSet 7871
TCustomDataSetValuesList events 7871
TCustomDataSetValuesList Members 7857
TCustomDataSetValuesList methods 7864
TCustomDataSetValuesList properties 7869
ValueField 7871
TCustomDataStore class 17772
about TCustomDataStore class 17772
Active 17787
ActiveChanged 17778
AfterClose 17795
AfterOpen 17795
BeforeClose 17796
BeforeOpen 17796
BinData 17787
BinMapHasBinData 17778
CalcSubTotals 17778
CanDimBeClosed 17778
CanSumBeClosed 17779
Capacity 17788
Create 17776
CurrentSummary 17788
DataCache 17789
DataLink 17789
DataSet 17789
DataSource 17790
DesignState 17790
Destroy 17777
DimensionCount 17790
DimensionMap 17790
DimensionMapCount 17791
DoAfterClose 17779
DoAfterOpen 17779
DoBeforeClose 17779
DoBeforeOpen 17780
DoOnRefresh 17780
GetCurrentSummary 17780
GetDataSet 17780

GetDimensionMemberCount 17780
GetDimensionName 17781
GetDomain 17781
GetMemberAsString 17782
GetMemberAsVariant 17782
GetMemoryUsage 17783
GetSummaryAsString 17783
GetSummaryAsVariant 17784
GetSummaryName 17784
InternalDataSource 17792
LayoutChanged 17785
MaxCells 17792
MaxDimensions 17793
MaxSummaries 17793
Notification 17785
OnLowCapacity 17796
OnRefresh 17797
Refresh 17785
SetCurrentSummary 17786
SetDataSet 17786
ShowProgressDialog 17794
State 17794
StateChanged 17786
SummaryCount 17794
TCustomDataStore events 17795
TCustomDataStore Members 17772
TCustomDataStore methods 17777
TCustomDataStore properties 17786
TCustomDBGrid class 9608
about TCustomDBGrid class 9608
AcquireLayoutLock 9625
BeginColumnDrag 9625
BeginLayout 9626
BeginUpdate 9627
CalcSizingState 9627
CalcTitleRect 9627
CancelLayout 9628
CanEditAcceptKey 9628
CanEditModify 9629
CanEditShow 9629
CellClick 9629
CheckColumnDrag 9630
ColEnter 9630
ColExit 9630
ColumnAtDepth 9631
ColumnMoved 9631
Columns 9653
ColWidthsChanged 9631
Create 9622
CreateColumns 9632
CreateDataLink 9632
CreateEditor 9632
CreateWnd 9633
DataLink 9654
DataSource 9655
DataToRawColumn 9633
DefaultDrawColumnCell 9634
DefaultDrawDataCell 9634
DefaultDrawing 9655
DefaultHandler 9635
DeferLayout 9635
DefineFieldMap 9636
DefineProperties 9636
Destroy 9622
DoMouseWheelDown 9636
DoMouseWheelUp 9637
DrawCell 9637
DrawColumnCell 9638
DrawDataCell 9638
EditButtonClick 9638
EditorMode 9656
EndColumnDrag 9639
EndLayout 9639
EndUpdate 9640
ExecuteAction 9640
FAcquireFocus 9623
FieldCount 9656
Fields 9657

FUpdateFields 9623
GetColField 9640
GetEditLimit 9641
GetEditMask 9641
GetEditStyle 9642
GetEditText 9642
GetFieldValue 9643
HighlightCell 9643
IndicatorOffset 9658
InvalidateTitles 9644
KeyDown 9644
KeyPress 9645
LayoutChanged 9645
LayoutLock 9659
LinkActive 9646
Loaded 9646
MouseDown 9647
MouseUp 9647
Notification 9647
OnCellClick 9664
OnColEnter 9665
OnColExit 9665
OnColumnMoved 9666
OnDrawColumnCell 9666
OnDrawDataCell 9667
OnEditButtonClick 9668
OnTitleClick 9668
Options 9659
ParentColor 9660
RawToDataColumn 9648
ReadOnly 9660
Scroll 9648
SelectedField 9661
SelectedIndex 9661
SelectedRows 9662
SetColumnAttributes 9649
SetEditText 9649
ShowPopupEditor 9649
StoreColumns 9650
TCustomDBGrid events 9664
TCustomDBGrid fields 9623
TCustomDBGrid Members 9609
TCustomDBGrid methods 9623
TCustomDBGrid properties 9653
TimedScroll 9650
TitleClick 9651
TitleFont 9663
TopLeftChanged 9651
UpdateAction 9651
UpdateLock 9664
UpdateScrollBar 9652
UseRightToLeftAlignmentForField 9652
ValidFieldIndex 9652
TCustomDecisionGraph class 17518
about TCustomDecisionGraph class 17518
Create 17519
DecisionSource 17521
Destroy 17519
Notification 17520
TCustomDecisionGraph Members 17518
TCustomDecisionGraph methods 17520
TCustomDecisionGraph properties 17520
Updated 17520
TCustomDecisionGrid class 17553
about TCustomDecisionGrid class 17553
CaptionColor 17574
CaptionFont 17575
CellDrawState 17568
CellRect 17569
Cells 17575
CellValueArray 17569
ColCount 17576
ColWidthsChanged 17569
Create 17566
DataColor 17576
DataFont 17577
DataSumColor 17577
DecisionSource 17578

DefaultColWidth 17578
DefaultRowHeight 17578
Destroy 17566
Dimensions 17579
DrawCell 17570
FGridStateX 17567
FixedCols 17579
FixedRows 17580
GetCells 17570
GridLineColor 17580
GridLineWidth 17580
LabelColor 17581
LabelFont 17581
LabelSumColor 17581
MouseDown 17570
MouseMove 17571
MouseUp 17571
Notification 17572
OnDecisionDrawCell 17584
OnDecisionExamineCell 17585
OnTopLeftChanged 17586
Options 17582
Paint 17572
RowCount 17582
RowHeightsChanged 17573
SelectCell 17573
ShowCubeEditor 17583
TCustomDecisionGrid events 17584
TCustomDecisionGrid fields 17567
TCustomDecisionGrid Members 17554
TCustomDecisionGrid methods 17567
TCustomDecisionGrid properties 17574
TopLeftChanged 17573
Totals 17583
TCustomDockForm class 13829
about TCustomDockForm class 13829
AutoScroll 13845
BorderStyle 13846
Create 13843
DoAddDockClient 13843
DoRemoveDockClient 13844
FormStyle 13846
GetSiteInfo 13844
Loaded 13845
PixelsPerInch 13848
TCustomDockForm Members 13829
TCustomDockForm methods 13843
TCustomDockForm properties 13845
TCustomDrawGrid class 14768
about TCustomDrawGrid class 14768
Canvas 14785
CellRect 14781
Col 14787
ColumnMoved 14781
ColWidths 14788
DrawCell 14782
EditorMode 14789
GetEditMask 14782
GetEditText 14783
GridHeight 14789
GridWidth 14789
LeftCol 14790
MouseToCell 14783
OnColumnMoved 14794
OnDrawCell 14795
OnGetEditMask 14796
OnGetEditText 14797
OnRowMoved 14798
OnSelectCell 14798
OnSetEditText 14800
OnTopLeftChanged 14800
Row 14790
RowHeights 14791
RowMoved 14783
SelectCell 14784
Selection 14791
SetEditText 14784
TabStops 14793

TCustomDrawGrid events 14794
TCustomDrawGrid Members 14768
TCustomDrawGrid methods 14781
TCustomDrawGrid properties 14785
TopLeftChanged 14784
TopRow 14793
TCustomDropDownButton class 205
about TCustomDropDownButton class 205
CalcBounds 214
Click 215
DrawFrame 215
DropDownClick 215
DroppedDown 217
GetPopupClass 215
MouseDown 216
MouseOverDropDown 216
Paint 216
TCustomDropDownButton Members 205
TCustomDropDownButton methods 214
TCustomDropDownButton properties 217
TCustomEdit class 21232
about TCustomEdit class 21232
AutoSelect 21260
AutoSize 21260
BorderStyle 21261
CanUndo 21262
Change 21245
CharCase 21262
Clear 21245
ClearSelection 21245
ClearUndo 21246
CopyToClipboard 21246
Create 21244
CreateParams 21248
CreateWindowHandle 21248
CreateWnd 21248
CutToClipboard 21249
DefaultHandler 21250
DestroyWnd 21251
DoSetMaxLength 21251
GetSelLength 21252
GetSelStart 21252
GetSelText 21252
GetSelTextBuf 21253
HideSelection 21262
MaxLength 21263
Modified 21263
OEMConvert 21264
OnChange 21275
ParentColor 21264
PasswordChar 21265
PasteFromClipboard 21254
ReadOnly 21265
SelectAll 21255
SelLength 21265
SelStart 21269
SelText 21272
SetAutoSize 21257
SetSelLength 21257
SetSelStart 21257
SetSelTextBuf 21258
TabStop 21273
TCustomEdit events 21275
TCustomEdit Members 21233
TCustomEdit methods 21244
TCustomEdit properties 21259
Text 21273
Undo 21259
TCustomEndUserAdapter class 24313
about TCustomEndUserAdapter class 24313
ExecuteLogin 24321
ExecuteLogout 24322
GetDisplayNameValue 24322
GetLoggedInValue 24323
GetUserID 24323
OnGetDisplayName 24324
OnGetUserID 24324
OnIsLoggedIn 24324

wwww

OnLogin 24325
OnLogout 24326
TCustomEndUserAdapter events 24323
TCustomEndUserAdapter Members 24314
TCustomEndUserAdapter methods 24321
TCustomEndUserAdapterAction class 24326
about TCustomEndUserAdapterAction class 24326
Adapter 24333
TCustomEndUserAdapterAction Members 24327
TCustomEndUserAdapterAction properties 24333
TCustomEndUserSessionAdapter class 24333
about TCustomEndUserSessionAdapter class 24333
ExecuteLogin 24342
ExecuteLogout 24342
GetDisplayNameValue 24343
GetLoggedInValue 24343
GetUserID 24343
TCustomEndUserSessionAdapter Members 24335
TCustomEndUserSessionAdapter methods 24342
TCustomEndUserSessionAdapter properties 24344
UserID 24344
TCustomFieldGroup class 16519
about TCustomFieldGroup class 16519
Create 16525
Custom 16527
Destroy 16525
GetLayoutAttributes 16526
ImplContent 16526
Style 16527
StyleRule 16528
TCustomFieldGroup Members 16520
TCustomFieldGroup methods 16526
TCustomFieldGroup properties 16527
TCustomFileDialog class 11288
about TCustomFileDialog class 11288
ClientGuid 11299
Create 11292
CreateFileDialog 11293
DefaultExtension 11300
DefaultFolder 11300
Destroy 11292
Dialog 11300
DoOnExecute 11294
DoOnFileOkClick 11294
DoOnFolderChange 11294
DoOnFolderChanging 11295
DoOnOverwrite 11295
DoOnSelectionChange 11295
DoOnShareViolation 11296
DoOnTypeChange 11296
Execute 11296
Execute method 11296
FavoriteLinks 11301
FileName 11301
FileNameLabel 11301
FileOkClick 11297
Files 11302
FileTypeIndex 11302
FileTypes 11303
FolderChange 11297
FolderChanging 11297
GetFileNames 11298
GetItemName 11298
GetResults 11298
Handle 11303
OkButtonLabel 11303
OnExecute 11305
OnFileOkClick 11306
OnFolderChange 11306
OnFolderChanging 11307
OnOverwrite 11307
OnSelectionChange 11307
OnShareViolation 11308
OnTypeChange 11308
Options 11304
Overwrite 11298
SelectionChange 11298
ShareViolation 11299

ShellItem 11304
ShellItems 11304
TCustomFileDialog events 11305
TCustomFileDialog Members 11289
TCustomFileDialog methods 11293
TCustomFileDialog properties 11299
Title 11305
TypeChange 11299
TCustomFileOpenDialog class 11308
about TCustomFileOpenDialog class 11308
CreateFileDialog 11313
GetResults 11313
SelectionChange 11313
TCustomFileOpenDialog Members 11309
TCustomFileOpenDialog methods 11312
TCustomFileRun class 11844
about TCustomFileRun class 11844
Browse 11852
BrowseDlg 11853
Create 11849
Directory 11853
ExecuteTarget 11849
FileName 11853
HandlesTarget 11850
HInst 11854
Notification 11850
Operation 11855
Parameters 11855
ParentControl 11856
SetupBrowseDialog 11851
ShowCmd 11856
TCustomFileRun Members 11845
TCustomFileRun methods 11849
TCustomFileRun properties 11852
UpdateTarget 11851
TCustomFileSaveDialog class 11314
about TCustomFileSaveDialog class 11314
CreateFileDialog 11318
TCustomFileSaveDialog Members 11314
TCustomFileSaveDialog methods 11318
TCustomFlowPanel class 12475
about TCustomFlowPanel class 12475
AlignControls 12487
AutoWrap 12488
Create 12486
Destroy 12486
FlowStyle 12489
GetChildren 12487
GetControlIndex 12488
SetControlIndex 12488
TCustomFlowPanel Members 12475
TCustomFlowPanel methods 12487
TCustomFlowPanel properties 12488
TCustomForm class 13849
about TCustomForm class 13849
Action 13895
Activate 13865
Active 13896
ActiveChanged 13865
ActiveControl 13896
ActiveMDIChild 13897
ActiveOleControl 13897
AfterConstruction 13866
AlignControls 13866
AlphaBlend 13898
AlphaBlendValue 13898
AutoScroll 13899
BeforeDestruction 13866
BeginAutoDrag 13867
BorderIcons 13899
BorderStyle 13900
Canvas 13900
Caption 13902
ChangeScale 13867
ClientHandle 13904
ClientHeight 13905
ClientWidth 13905
Close 13867

CloseModal 13868
CloseQuery 13868
Color 13908
Create 13861
CreateNew 13862
CreateParams 13868
CreateWindowHandle 13869
CreateWnd 13869
Ctl3D 13909
Deactivate 13870
DefaultHandler 13870
DefaultMonitor 13909
DefineProperties 13871
DefocusControl 13871
Designer 13910
Destroy 13862
DestroyHandle 13871
DestroyWindowHandle 13872
Dock 13872
DoClose 13873
DoCreate 13873
DoDestroy 13873
DoDock 13874
DoHide 13874
DoShow 13875
DropTarget 13910
FFormState 13863
FocusControl 13875
Font 13910
FormState 13911
FormStyle 13912
GetBorderIconStyles 13875
GetBorderStyles 13876
GetChildren 13876
GetClientRect 13876
GetFloating 13877
GetFormImage 13877
GetOwnerWindow 13878
GlassFrame 13914
HandleCreateException 13878
Height 13915
HelpFile 13916
Hide 13879
HorzScrollBar 13916
Icon 13917
IsShortCut 13880
KeyPreview 13918
Left 13920
Loaded 13880
MakeFullyVisible 13881
MDIChildCount 13921
MDIChildren 13922
Menu 13923
ModalResult 13924
Monitor 13926
MouseWheelHandler 13881
Notification 13882
ObjectMenuItem 13926
OldCreateOrder 13926
OleFormObject 13927
OnActivate 13943
OnCanResize 13947
OnClick 13947
OnClose 13949
OnCloseQuery 13950
OnCreate 13951
OnDblClick 13953
OnDeactivate 13953
OnDestroy 13953
OnDragDrop 13954
OnDragOver 13957
OnHelp 13959
OnHide 13960
OnKeyDown 13962
OnKeyPress 13964
OnKeyUp 13966
OnMouseActivate 13967
OnMouseDown 13968

OnMouseMove 13971
OnMouseUp 13973
OnPaint 13975
OnResize 13976
OnShortCut 13977
OnShow 13977
Paint 13882
PaintWindow 13882
PaletteChanged 13883
ParentFont 13927
PixelsPerInch 13928
PopupChildren 13928
PopupMenu 13928
PopupMode 13933
PopupParent 13933
Position 13933
Print 13883
PrintScale 13934
QueryInterface 13884
ReadState 13884
RecreateAsPopup 13884
Release 13885
RequestAlign 13885
Resizing 13885
Scaled 13935
ScreenSnap 13935
SendCancelMode 13886
SetChildOrder 13886
SetFocus 13886
SetFocusedControl 13887
SetParent 13887
SetParentBiDiMode 13887
Show 13888
ShowModal 13890
SnapBuffer 13936
TCustomForm events 13942
TCustomForm fields 13863
TCustomForm Members 13849
TCustomForm methods 13863
TCustomForm properties 13894
TileMode 13936
Top 13937
TransparentColor 13938
TransparentColorValue 13939
UpdateActions 13891
UpdateGlassFrame 13892
UpdateWindowState 13892
ValidateRename 13892
VertScrollBar 13939
Visible 13940
VisibleChanging 13893
WantChildKey 13893
Width 13941
WindowMenu 13941
WindowState 13942
WndProc 13893
TCustomFrame class 13978
about TCustomFrame class 13978
Create 13989
CreateParams 13990
GetChildren 13990
PaintWindow 13991
SetParent 13991
TCustomFrame Members 13979
TCustomFrame methods 13990
TCustomGrid class 14800
about TCustomGrid class 14800
AdjustSize 14816
BeginColumnDrag 14817
BeginRowDrag 14817
BorderStyle 14843
BoxRect 14818
CalcDrawInfo 14818
CalcFixedInfo 14818
CalcSizingState 14819
CanEditAcceptKey 14819
CanEditModify 14819
CanEditShow 14820

CanGridAcceptKey 14820
CellRect 14821
ChangeGridOrientation 14821
CheckColumnDrag 14821
CheckRowDrag 14822
Col 14843
ColCount 14844
Color 14844
ColumnMoved 14822
ColWidths 14845
ColWidthsChanged 14822
Create 14813
CreateEditor 14823
CreateParams 14823
DefaultColWidth 14846
DefaultDrawing 14847
DefaultRowHeight 14847
DefineProperties 14824
DeleteColumn 14824
DeleteRow 14825
DesignOptionsBoost 14814
Destroy 14813
DoExit 14825
DoMouseWheelDown 14825
DoMouseWheelUp 14826
DrawCell 14826
EditorMode 14848
EndColumnDrag 14827
EndRowDrag 14827
FGridState 14814
FixedColor 14848
FixedCols 14848
FixedRows 14849
FocusCell 14827
FSaveCellExtents 14814
GetEditLimit 14828
GetEditMask 14828
GetEditStyle 14828
GetEditText 14829
GetGridHeight 14829
GetGridWidth 14830
GridHeight 14850
GridLineWidth 14850
GridWidth 14851
HideEdit 14830
HideEditor 14830
HitTest 14851
InplaceEditor 14852
InvalidateCell 14831
InvalidateCol 14831
InvalidateEditor 14832
InvalidateGrid 14832
InvalidateRow 14832
KeyDown 14833
KeyPress 14833
LeftCol 14852
MouseCoord 14834
MouseDown 14834
MouseMove 14834
MouseUp 14835
MoveColRow 14835
MoveColumn 14836
MoveRow 14836
Options 14853
Paint 14836
ParentColor 14853
Row 14854
RowCount 14854
RowHeights 14855
RowHeightsChanged 14837
RowMoved 14837
ScrollBars 14856
ScrollData 14838
SelectCell 14838
Selection 14856
SetEditText 14839
ShowEditor 14839
ShowEditorChar 14839

SizeChanged 14840
Sizing 14840
TabStop 14857
TabStops 14858
TCustomGrid fields 14814
TCustomGrid Members 14801
TCustomGrid methods 14815
TCustomGrid properties 14842
TimedScroll 14840
TopLeftChanged 14841
TopRow 14858
UpdateDesigner 14841
VirtualView 14814
VisibleColCount 14859
VisibleRowCount 14859
TCustomGridPanel class 12489
about TCustomGridPanel class 12489
AlignControls 12502
AutoAddColumn 12502
AutoAddRow 12503
CellCount 12506
CellIndexToCell 12503
CellRect 12506
CellSize 12507
CellToCellIndex 12503
ColumnCollection 12507
ColumnSpanIndex 12507
ControlCollection 12507
Create 12500
Destroy 12501
ExpandStyle 12507
GetChildren 12503
IsColumnEmpty 12504
IsRowEmpty 12504
Loaded 12504
Paint 12505
RemoveEmptyAutoAddColumns 12505
RemoveEmptyAutoAddRows 12505
RowCollection 12508
RowSpanIndex 12508
TCustomGridPanel Members 12490
TCustomGridPanel methods 12501
TCustomGridPanel properties 12506
UpdateControlOriginalParentSize 12505
TCustomGroupBox class 21275
about TCustomGroupBox class 21275
AdjustClientRect 21287
Create 21286
CreateParams 21287
Paint 21287
TCustomGroupBox Members 21276
TCustomGroupBox methods 21286
TCustomHeaderControl class 3702
about TCustomHeaderControl class 3702
Align 3723
Anchors 3727
BiDiMode 3728
BorderWidth 3728
Canvas 3728
Constraints 3730
Create 3714
CreateParams 3715
CreateSection 3715
CreateSections 3715
CreateWnd 3716
Destroy 3714
DestroyWnd 3716
DragCursor 3730
DragKind 3731
DragMode 3731
DragReorder 3733
DrawSection 3716
Enabled 3733
FlipChildren 3717
Font 3733
FullDrag 3735
HotTrack 3735
Images 3735

Notification 3717
OnCreateSectionClass 3736
OnDrawSection 3737
OnSectionClick 3737
OnSectionDrag 3737
OnSectionEndDrag 3738
OnSectionResize 3738
OnSectionTrack 3738
SectionClick 3717
SectionDrag 3717
SectionEndDrag 3718
SectionResize 3718
Sections 3735
SectionTrack 3718
SetImages 3718
Style 3736
TCustomHeaderControl events 3736
TCustomHeaderControl Members 3703
TCustomHeaderControl methods 3714
TCustomHeaderControl properties 3723
WndProc 3719
TCustomHotKey class 3739
about TCustomHotKey class 3739
AutoSize 3752
CNCommand 3750
Create 3750
CreateParams 3751
CreateWnd 3751
HotKey 3752
InvalidKeys 3753
Modifiers 3755
OnChange 3757
SetAutoSize 3752
TabStop 3756
TCustomHotKey events 3756
TCustomHotKey Members 3739
TCustomHotKey methods 3750
TCustomHotKey properties 3752
TCustomImageList class 15361
about TCustomImageList class 15361
Add 15368
AddIcon 15368
AddImage 15368
AddImages 15369
AddMasked 15369
AllocBy 15387
Assign 15370
AssignTo 15370
BkColor 15387
BlendColor 15387
Change 15371
Clear 15371
Count 15388
Create 15365
CreateSize 15366
DefineProperties 15371
Delete 15372
Destroy 15366
DoDraw 15372
Draw 15373
Draw method 15373
DrawingStyle 15390
DrawOverlay 15375
DrawOverlay method 15374
FileLoad 15375
GetBitmap 15376
GetHotSpot 15376
GetIcon 15377
GetIcon method 15377
GetImageBitmap 15378
GetImages 15378
GetInstRes 15379
GetInstRes method 15378
GetMaskBitmap 15379
GetResource 15380
Handle 15390
HandleAllocated 15380
HandleNeeded 15381

Height 15391
ImageType 15391
Initialize 15381
Insert 15381
InsertIcon 15382
InsertMasked 15382
Masked 15392
Move 15382
OnChange 15393
Overlay 15383
ReadData 15383
RegisterChanges 15383
Replace 15384
ReplaceIcon 15384
ReplaceMasked 15384
ResInstLoad 15385
ResourceLoad 15385
ShareImages 15392
TCustomImageList events 15393
TCustomImageList Members 15362
TCustomImageList methods 15367
TCustomImageList properties 15386
UnRegisterChanges 15386
Width 15392
WriteData 15386
TCustomInetXPageProducer class 17131
about TCustomInetXPageProducer class 17131
Content 17138
Create 17137
Destroy 17137
GetTemplateFileName 17139
GetTemplateStream 17139
HTMLDoc 17140
HTMLFile 17141
TCustomInetXPageProducer Members 17132
TCustomInetXPageProducer methods 17138
TCustomInetXPageProducer properties 17140
WebContent 17140
TCustomIniFile class 15434
about TCustomIniFile class 15434
Create 15436
DeleteKey 15437
EraseSection 15438
FileName 15450
ReadBinaryStream 15438
ReadBool 15439
ReadDate 15439
ReadDateTime 15440
ReadFloat 15440
ReadInteger 15441
ReadSection 15442
ReadSections 15442
ReadSections method 15442
ReadSectionValues 15443
ReadString 15443
ReadTime 15444
SectionExists 15444
TCustomIniFile Members 15435
TCustomIniFile methods 15437
TCustomIniFile properties 15449
UpdateFile 15445
ValueExists 15445
WriteBinaryStream 15446
WriteBool 15446
WriteDate 15446
WriteDateTime 15447
WriteFloat 15447
WriteInteger 15448
WriteString 15448
WriteTime 15449
TCustomIpClient class 20063
about TCustomIpClient class 20063
Close 20068
Connect 20069
Connected 20071
Create 20068
Disconnect 20069
DoConnect 20069

eeee

DoDisconnect 20070
GetThreadObject 20070
OnConnect 20071
OnDisconnect 20072
Open 20070
TCustomIpClient events 20071
TCustomIpClient Members 20064
TCustomIpClient methods 20068
TCustomIpClient properties 20071
TCustomizeActionBars class 1720
about TCustomizeActionBars class 1720
ActionManager 1726
Create 1725
CustomizeDlg 1726
ExecuteTarget 1725
HandlesTarget 1726
TCustomizeActionBars Members 1720
TCustomizeActionBars methods 1725
TCustomizeActionBars properties 1726
TCustomizeActionClientItem class 756
about TCustomizeActionClientItem class 756
GetActionLinkClass 759
TCustomizeActionClientItem Members 756
TCustomizeActionClientItem methods 759
TCustomizeActionLink class 759
about TCustomizeActionLink class 759
SetEnabled 762
TCustomizeActionLink Members 760
TCustomizeActionLink methods 762
TCustomizeActionToolBar class 762
about TCustomizeActionToolBar class 762
AdditionalItem 780
AddlItem 780
CanAutoSize 778
Clear 778
Create 777
Destroy 777
DoItemClicked 778
GetActionControl 779
GetAddRemoveItemClass 779
GetSubMenuControlClass 779
ItemCount 779
NewPopup 780
SetActionClient 780
TCustomizeActionToolBar Members 763
TCustomizeActionToolBar methods 777
TCustomizeActionToolBar properties 780
TCustomizeDlg class 6528
about TCustomizeDlg class 6528
ActionManager 6533
CustomizeForm 6533
Notification 6531
OnClose 6534
OnShow 6534
SetupDlg 6532
Show 6532
StayOnTop 6533
TCustomizeDlg events 6534
TCustomizeDlg Members 6528
TCustomizeDlg methods 6531
TCustomizeDlg properties 6532
TCustomizeFrm class 6535
about TCustomizeFrm class 6535
ActionBarList 6554
ActionBarListClick 6565
ActionBarListClickCheck 6565
ActionImages 6554
ActionList1 6554
ActionManager 6573
ActionsActionsLbl 6554
ActionsCatLbl 6554
ActionsList 6555
ActionsListClick 6566
ActionsListData 6566
ActionsListDrawItem 6566
ActionsListMeasureItem 6566
ActionsListStartDrag 6566
ActionsTab 6555

ActiveActionList 6574
AddAction 6567
ApplyToAllActn 6555
ApplyToAllActnUpdate 6567
ApplyToAllChk 6555
CaptionOptionsCombo 6556
CaptionOptionsComboChange 6567
CaptionOptionsGrp 6556
CatList 6556
CatListClick 6567
CatListStartDrag 6568
ClearCatList 6568
CloseActn 6556
CloseBtn 6556
CloseBtnClick 6568
CloseItem 6557
CloseMenu 6557
CMVisiblechanged 6568
ComboPanel 6557
DescGroupBox 6557
FindActionBars 6568
FormClose 6569
FormCreate 6569
FormResize 6569
FullMenusActn 6558
HintLbl 6558
InfoLbl 6558
IsDupShortCut 6569
Label1 6558
Label2 6558
Label4 6559
LargeIconsActn 6559
LargeIconsActnExecute 6570
LargeIconsActnUpdate 6570
LargeIconsChk 6559
ListCombo 6559
ListComboSelect 6570
ListPanel 6560
MenuAnimationStyles 6560
MenuAnimationStylesChange 6570
Notification 6570
OptionsBevel1 6560
OptionsBevel2 6560
OptionsTab 6560
OtherLbl 6561
PersonalizeLbl 6561
RecentlyUsedActn 6561
RecentlyUsedActnExecute 6571
RecentlyUsedActnUpdate 6571
RecentlyUsedChk 6561
ResetActn 6562
ResetActnExecute 6571
ResetActnUpdate 6572
ResetBtn 6562
ResetUsageBtn 6562
ResetUsageDataActn 6562
ResetUsageDataActnExecute 6572
SeparatorBtn 6562
SeparatorBtnStartDrag 6572
SetActiveActionList 6572
ShortCutTipsChk 6563
ShowHintsActn 6563
ShowHintsActnExecute 6572
ShowHintsActnUpdate 6573
ShowShortCutsInTipsActn 6563
ShowShortCutsInTipsActnExecute 6573
ShowTipsChk 6563
Tabs 6564
TCustomizeFrm fields 6553
TCustomizeFrm Members 6535
TCustomizeFrm methods 6564
TCustomizeFrm properties 6573
ToolbarsLbl 6564
ToolbarsTab 6564
UpdateDialog 6573
TCustomLabel class 21288
about TCustomLabel class 21288
AdjustBounds 21296

Alignment 21299
AutoSize 21299
Canvas 21300
Caption 21303
Create 21295
DoDrawText 21296
EllipsisPosition 21305
FocusControl 21306
GetLabelText 21297
Layout 21306
Loaded 21297
Notification 21298
Paint 21298
SetAutoSize 21298
ShowAccelChar 21306
TCustomLabel Members 21288
TCustomLabel methods 21296
TCustomLabel properties 21299
Transparent 21307
WordWrap 21308
TCustomLabeledEdit class 12508
about TCustomLabeledEdit class 12508
CMBidimodechanged 12521
CMEnabledchanged 12521
CMVisiblechanged 12521
Create 12520
EditLabel 12524
LabelPosition 12524
LabelSpacing 12525
Notification 12522
SetBounds 12522
SetName 12523
SetParent 12523
SetupInternalLabel 12523
TCustomLabeledEdit Members 12509
TCustomLabeledEdit methods 12520
TCustomLabeledEdit properties 12524
TCustomLayoutGroup class 16528
about TCustomLayoutGroup class 16528
Create 16533
Custom 16535
Destroy 16533
DisplayColumns 16536
GetLayoutAttributes 16534
ImplCanAddClass 16534
ImplContent 16535
Style 16536
StyleRule 16537
TCustomLayoutGroup Members 16529
TCustomLayoutGroup methods 16534
TCustomLayoutGroup properties 16535
TCustomListAction class 15713
about TCustomListAction class 15713
Active 15721
Count 15722
Create 15718
ExecuteTarget 15718
GetCount 15719
GetString 15719
HandlesTarget 15719
Images 15722
ItemIndex 15722
Loaded 15720
Loading 15723
OnGetItemCount 15723
OnItemSelected 15724
SetString 15720
Strings 15723
TCustomListAction events 15723
TCustomListAction Members 15714
TCustomListAction methods 15718
TCustomListAction properties 15721
TCustomListBox class 21308
about TCustomListBox class 21308
AddItem 21322
AutoComplete 21342
AutoCompleteDelay 21342
BorderStyle 21343

Canvas 21343
Clear 21322
ClearSelection 21323
Columns 21346
CopySelection 21323
Count 21346
Create 21320
CreateParams 21324
CreateWnd 21325
DeleteSelected 21325
DeleteString 21326
Destroy 21320
DestroyWnd 21326
DoFindData 21326
DoGetData 21327
DoGetDataObject 21327
DragCanceled 21328
DrawItem 21328
ExtendedSelect 21346
FMoving 21321
GetCount 21329
GetItemData 21329
GetItemIndex 21329
GetSelCount 21330
GetSelected 21330
IntegralHeight 21347
InternalGetItemData 21330
InternalSetItemData 21330
ItemAtPos 21331
ItemHeight 21347
ItemRect 21333
Items 21348
KeyPress 21334
LoadRecreateItems 21334
MeasureItem 21335
OnData 21357
OnDataFind 21357
OnDataObject 21358
OnDrawItem 21358
OnMeasureItem 21360
ParentColor 21350
ResetContent 21335
SaveRecreateItems 21336
ScrollWidth 21351
SelectAll 21336
Selected 21351
SetItemData 21336
SetItemIndex 21336
SetMultiSelect 21337
Sorted 21353
Style 21354
TabStop 21355
TabWidth 21355
TCustomListBox events 21357
TCustomListBox fields 21321
TCustomListBox Members 21309
TCustomListBox methods 21321
TCustomListBox properties 21341
TopIndex 21356
WndProc 21337
TCustomListControl class 5928
about TCustomListControl class 5928
AddItem 5940
Clear 5940
ClearSelection 5940
CopySelection 5941
DeleteSelected 5941
GetCount 5942
GetItemIndex 5942
ItemIndex 5944
MoveSelection 5942
SelectAll 5943
SetItemIndex 5943
TCustomListControl Members 5929
TCustomListControl methods 5939
TCustomListControl properties 5944
TCustomListView class 3757
about TCustomListView class 3757

ActionChange 3773
AddItem 3773
AllocBy 3803
AlphaSort 3774
Arrange 3777
BorderStyle 3803
BoundingRect 3804
CanChange 3777
CanEdit 3778
Canvas 3804
Change 3778
ChangeScale 3778
Checkboxes 3805
Clear 3779
ClearSelection 3779
ColClick 3780
ColRightClick 3780
Column 3806
ColumnClick 3808
Columns 3808
ColumnsShowing 3780
CopySelection 3781
Create 3770
CreateListItem 3781
CreateListItems 3782
CreateParams 3782
CreateWnd 3782
CustomDraw 3783
CustomDrawItem 3783
CustomDrawSubItem 3783
CustomSort 3784
Delete 3785
DeleteSelected 3786
Destroy 3771
DestroyWnd 3786
DoEndDrag 3787
DoInfoTip 3787
DoStartDrag 3787
DrawItem 3788
DropTarget 3810
Edit 3788
FindCaption 3788
FindData 3789
FlatScrollBars 3810
FullDrag 3811
GetActionLinkClass 3789
GetCount 3789
GetDragImages 3790
GetHitTestInfoAt 3790
GetItemAt 3790
GetItemIndex 3791
GetItemIndex method 3791
GetNearestItem 3791
GetNextItem 3792
GetSearchString 3793
GetSelCount 3794
GridLines 3811
HideSelection 3812
HotTrack 3812
HotTrackStyles 3812
HoverTime 3813
IconOptions 3813
InsertItem 3794
IsCustomDrawn 3794
IsEditing 3795
ItemFocused 3813
Items 3814
LargeImages 3817
MouseUp 3795
MultiSelect 3820
Notification 3796
OnAdvancedCustomDraw 3834
OnAdvancedCustomDrawItem 3835
OnAdvancedCustomDrawSubItem 3835
OnChange 3836
OnChanging 3837
OnColumnClick 3837
OnColumnDragged 3840

OnColumnRightClick 3841
OnCompare 3841
OnCreateItemClass 3844
OnCustomDraw 3844
OnCustomDrawItem 3845
OnCustomDrawSubItem 3846
OnData 3846
OnDataFind 3847
OnDataHint 3847
OnDataStateChange 3848
OnDeletion 3848
OnDrawItem 3848
OnEdited 3849
OnEditing 3850
OnGetImageIndex 3850
OnGetSubItemImage 3850
OnInfoTip 3851
OnInsert 3851
OnSelectItem 3852
OwnerData 3820
OwnerDataFetch 3796
OwnerDataFind 3797
OwnerDataHint 3797
OwnerDataStateChange 3798
OwnerDraw 3821
ReadOnly 3821
RowSelect 3821
Scroll 3798
SelCount 3822
SelectAll 3799
Selected 3822
SetItemIndex 3799
SetMultiSelect 3800
SetViewStyle 3800
ShowColumnHeaders 3824
ShowWorkAreas 3824
SmallImages 3825
SortType 3827
StateImages 3828
StringWidth 3800
TCustomListView events 3833
TCustomListView Members 3757
TCustomListView methods 3772
TCustomListView properties 3802
TopItem 3828
UpdateColumn 3801
UpdateColumns 3801
UpdateItems 3801
ViewOrigin 3829
ViewStyle 3829
VisibleRowCount 3832
WndProc 3802
WorkAreas 3833
TCustomLocateFileService class 24880
about TCustomLocateFileService class 24880
FindIncludeFile 24884
FindStream 24885
FindTemplateFile 24885
GetFileStream 24886
GetTemplateStream 24886
ImplFindIncludeFile 24887
ImplGetTemplateStream 24887
OnFindIncludeFile 24888
OnFindStream 24889
OnFindTemplateFile 24889
TCustomLocateFileService events 24888
TCustomLocateFileService Members 24881
TCustomLocateFileService methods 24884
TCustomLoginFormAdapter class 24344
about TCustomLoginFormAdapter class 24344
EndUser 24353
ExecuteLogin 24353
GetNextPage 24354
ImplCanAddActionClass 24354
ImplCanAddFieldClass 24355
ImplGetActionsList 24355
ImplGetFieldsList 24356
ImplNotifyDeactivate 24356

NextPage 24359
OnLogin 24361
OnValidatePassword 24362
OnValidateUserName 24362
Password 24360
PasswordRequired 24360
RaiseBlankPassword 24357
RaiseBlankUserName 24357
TCustomLoginFormAdapter events 24361
TCustomLoginFormAdapter Members 24346
TCustomLoginFormAdapter methods 24352
TCustomLoginFormAdapter properties 24359
UserList 24357
UserName 24360
ValidatePassword 24358
ValidateUserName 24358
TCustomLoginFormAdapterAction class 24363
about TCustomLoginFormAdapterAction class 24363
Adapter 24369
TCustomLoginFormAdapterAction Members 24364
TCustomLoginFormAdapterAction properties 24369
TCustomMaskEdit class 15798
about TCustomMaskEdit class 15798
CheckCursor 15811
Clear 15812
Create 15810
EditCanModify 15812
EditMask 15821
EditText 15821
GetFirstEditChar 15813
GetLastEditChar 15813
GetMaxChars 15814
GetNextEditChar 15814
GetPriorEditChar 15815
GetSel 15815
GetTextLen 15816
IsMasked 15822
KeyDown 15816
KeyPress 15816
KeyUp 15817
MaskState 15822
MaxLength 15822
ReformatText 15817
Reset 15818
SetCursor 15818
SetSel 15819
TCustomMaskEdit Members 15799
TCustomMaskEdit methods 15811
TCustomMaskEdit properties 15820
Text 15823
Validate 15819
ValidateEdit 15819
ValidateError 15820
TCustomMDIMenuButton class 781
about TCustomMDIMenuButton class 781
ButtonStyle 790
Paint 789
TCustomMDIMenuButton Members 781
TCustomMDIMenuButton methods 789
TCustomMDIMenuButton properties 790
TCustomMemo class 21362
about TCustomMemo class 21362
Alignment 21379
CaretPos 21380
Create 21374
CreateParams 21375
CreateWindowHandle 21376
Destroy 21374
GetCaretPos 21376
GetControlsAlignment 21376
KeyPress 21377
Lines 21380
Loaded 21377
ScrollBars 21381
SetAlignment 21377
SetCaretPos 21378
SetLines 21378
SetScrollBars 21378

SetWordWrap 21379
TCustomMemo Members 21363
TCustomMemo methods 21375
TCustomMemo properties 21379
WantReturns 21384
WantTabs 21384
WordWrap 21384
TCustomMemoryStream class 2815
about TCustomMemoryStream class 2815
Memory 2823
Read 2817
SaveToFile 2818
SaveToStream 2818
Seek 2819
SetPointer 2823
TCustomMemoryStream Members 2815
TCustomMemoryStream methods 2817
TCustomMemoryStream properties 2823
TCustomMenuButton class 790
about TCustomMenuButton class 790
BeginAutoDrag 801
Click 801
CMMouseEnter 801
Create 799
Destroy 800
DoDragDrop 802
DragOver 802
DrawText 802
Keyed 803
Menu 805
MouseDown 803
MouseUp 804
Paint 804
TCustomMenuButton Members 791
TCustomMenuButton methods 800
TCustomMenuButton properties 804
TCustomMenuExpandBtn class 805
about TCustomMenuExpandBtn class 805
Create 814
DragOver 815
DrawBackground 815
DrawFrame 816
Menu 816
TCustomMenuExpandBtn Members 805
TCustomMenuExpandBtn methods 815
TCustomMenuExpandBtn properties 816
TCustomMenuItem class 816
about TCustomMenuItem class 816
CalcBounds 827
CalcLayout 827
Click 827
Create 825
Destroy 825
DoDragDrop 828
DragDrop 828
DragOver 828
DrawEdge 829
DrawShadowedText 829
DrawSubMenuGlyph 829
DrawText 829
DrawUnusedEdges 830
Edges 832
FNoPrefix 826
Keyed 830
Menu 832
MouseSelected 832
MouseUp 830
Paint 831
PositionChanged 831
SetSelected 831
ShortCutBounds 832
TCustomMenuItem fields 826
TCustomMenuItem Members 817
TCustomMenuItem methods 826
TCustomMenuItem properties 831
TCustomModule class 10952
about TCustomModule class 10952
Create 10953

Designer 10957
Destroy 10954
ExecuteVerb 10955
GetAttributes 10955
GetVerb 10955
GetVerbCount 10955
Nestable 10955
PrepareItem 10956
Root 10957
Saving 10956
TCustomModule Members 10952
TCustomModule methods 10954
TCustomModule properties 10956
ValidateComponent 10956
ValidateComponentClass 10956
TCustomMultiSelectListControl class 5944
about TCustomMultiSelectListControl class 5944
FMultiSelect 5956
GetSelCount 5956
MultiSelect 5957
SelCount 5957
SetMultiSelect 5956
TCustomMultiSelectListControl fields 5955
TCustomMultiSelectListControl Members 5945
TCustomMultiSelectListControl methods 5956
TCustomMultiSelectListControl properties 5957
TCustomObjectBroker class 16002
about TCustomObjectBroker class 16002
GetComputerForGUID 16006
GetComputerForProgID 16006
GetPortForComputer 16007
SetConnectStatus 16008
TCustomObjectBroker Members 16003
TCustomObjectBroker methods 16006
TCustomOutline class 18234
about TCustomOutline class 18234
Add 18250
AddChild 18250
AddChildObject 18251
AddObject 18252
BadIndex 18253
BeginUpdate 18253
Canvas 18268
Clear 18253
Click 18254
Collapse 18254
Create 18248
DblClick 18254
DefineProperties 18255
Delete 18255
DeleteNode 18256
Destroy 18248
DrawCell 18256
EndUpdate 18257
Expand 18257
FullCollapse 18257
FullExpand 18258
GetDataItem 18258
GetItem 18259
GetNodeDisplayWidth 18259
GetTextItem 18259
GetVisibleNode 18260
Insert 18260
InsertObject 18261
ItemCount 18270
ItemHeight 18270
Items 18271
ItemSeparator 18271
KeyDown 18262
KeyPress 18262
Lines 18272
Loaded 18263
LoadFromFile 18263
LoadFromStream 18264
MouseDown 18264
Move 18265
OnCollapse 18278
OnDrawItem 18278

OnExpand 18279
Options 18272
OutlineStyle 18273
PictureClosed 18274
PictureLeaf 18274
PictureMinus 18275
PictureOpen 18275
PicturePlus 18276
Row 18276
SaveToFile 18265
SaveToStream 18265
SelectedItem 18277
SetDisplayWidth 18266
SetGoodIndex 18266
SetLevel 18267
SetUpdateState 18267
Style 18277
TCustomOutline events 18278
TCustomOutline Members 18235
TCustomOutline methods 18249
TCustomOutline properties 18268
TCustomPacketWriter class 18445
about TCustomPacketWriter class 18445
AddAttribute 18448
Check 18448
Create 18446
Destroy 18446
DSWriter 18449
FBuffer 18447
FIDSWriter 18447
SetDSWriter 18448
TCustomPacketWriter fields 18447
TCustomPacketWriter Members 18445
TCustomPacketWriter methods 18448
TCustomPacketWriter properties 18449
TCustomPagedAdapter class 24370
about TCustomPagedAdapter class 24370
TCustomPagedAdapterAction class 24371
about TCustomPagedAdapterAction class 24371
Adapter 24378
GetAdapter 24377
TCustomPagedAdapterAction Members 24372
TCustomPagedAdapterAction methods 24377
TCustomPagedAdapterAction properties 24378
TCustomPageDispatcher class 24889
about TCustomPageDispatcher class 24889
DefaultPage 24900
DispatchPageName 24894
EndUser 24895
GetCanView 24895
GetDefaultPageName 24895
GetLoginRequired 24896
GetRequestedPage 24896
ImplDispatchPageName 24896
ImplGetCanView 24896
ImplGetDefaultPageName 24897
ImplGetLoginRequired 24897
ImplHandleRequest 24897
ImplPageNameOfRequest 24897
ImplRedirectToPageName 24898
IsLoggedIn 24898
OnAfterDispatchPage 24901
OnAfterRedirectToPage 24901
OnBeforeDispatchPage 24902
OnBeforeRedirectToPage 24902
OnCanViewPage 24903
OnIsLoginRequired 24904
OnPageAccessDenied 24904
OnPageNotFound 24905
PageNameOfRequest 24898
ProduceLoginPage 24899
RedirectToPageName 24899
TCustomPageDispatcher events 24900
TCustomPageDispatcher Members 24890
TCustomPageDispatcher methods 24894
TCustomPageDispatcher properties 24900
ValidatePage 24899
TCustomPageProducer class 15281

about TCustomPageProducer class 15281
Create 15285
Destroy 15285
GetTemplateFileName 15286
GetTemplateStream 15286
HandleTag 15286
HTMLDoc 15287
HTMLFile 15287
TCustomPageProducer Members 15281
TCustomPageProducer methods 15285
TCustomPageProducer properties 15286
TCustomPanel class 12525
about TCustomPanel class 12525
AdjustClientRect 12537
Alignment 12539
BevelInner 12540
BevelOuter 12541
BevelWidth 12542
BorderStyle 12543
BorderWidth 12543
CanAutoSize 12537
Color 12544
Create 12536
CreateParams 12537
FullRepaint 12545
GetControlsAlignment 12538
Locked 12545
Paint 12538
ParentBackground 12546
ParentColor 12546
SetParentBackground 12539
TCustomPanel Members 12526
TCustomPanel methods 12536
TCustomPanel properties 12539
VerticalAlignment 12547
TCustomPanningWindow class 5958
about TCustomPanningWindow class 5958
GetIsPanning 5969
StartPanning 5969
StopPanning 5969
TCustomPanningWindow Members 5958
TCustomPanningWindow methods 5969
TCustomProvider class 18449
about TCustomProvider class 18449
ActiveUpdateException 18474
AfterApplyUpdates 18475
AfterExecute 18476
AfterGetParams 18477
AfterGetRecords 18477
AfterRowRequest 18478
ApplyUpdates 18459, 18460
ApplyUpdates method 18459
BeforeApplyUpdates 18479
BeforeExecute 18479
BeforeGetParams 18480
BeforeGetRecords 18480
BeforeRowRequest 18482
Create 18454
Data 18474
DataRequest 18460
Destroy 18454
DoAfterApplyUpdates 18460
DoAfterExecute 18461
DoAfterGetParams 18461
DoAfterGetRecords 18462
DoAfterRowRequest 18462
DoApplyUpdates 18463
DoBeforeApplyUpdates 18463
DoBeforeExecute 18464
DoBeforeGetParams 18464
DoBeforeGetRecords 18465
DoBeforeRowRequest 18465
DoGetRecords 18466
Execute 18466
Exported 18474
FActiveUpdateAbortException 18455
FAfterApplyUpdates 18455
FAfterExecute 18456

FAfterGetParams 18456
FAfterGetRecords 18456
FAfterRowRequest 18456
FBeforeApplyUpdates 18456
FBeforeExecute 18457
FBeforeGetParams 18457
FBeforeGetRecords 18457
FBeforeRowRequest 18457
FExported 18458
FOnDataRequest 18458
FOnValidate 18458
FOwnerData 18458
GetData 18467
GetParams 18467
GetRecords 18468
GetRecords method 18468
InternalApplyUpdates 18469
InternalExecute 18470
InternalGetParams 18471
InternalGetRecords 18471
InternalRowRequest 18472
OnDataRequest 18482
OnValidate 18483
OwnerData 18475
RowRequest 18472
SetActiveUpdateException 18473
TCustomProvider events 18475
TCustomProvider fields 18455
TCustomProvider Members 18450
TCustomProvider methods 18458
TCustomProvider properties 18473
TCustomQueryButtons class 16537
about TCustomQueryButtons class 16537
Custom 16544
GetDefaultButtons 16542
ImplAddElements 16542
ImplCanAddClass 16543
ImplContent 16543
Style 16544
StyleRule 16545
TCustomQueryButtons Members 16538
TCustomQueryButtons methods 16542
TCustomQueryButtons properties 16544
VisibleButtons 16543
TCustomQueryForm class 16545
about TCustomQueryForm class 16545
Action 16553
AssignStrings 16551
AssignStringsCallback 16551
Create 16550
GetHTMLFormTag 16551
ImplAddElements 16552
ImplCanAddClass 16552
ImplContent 16552
Method 16554
MethodString 16553
TCustomQueryForm Members 16546
TCustomQueryForm methods 16551
TCustomQueryForm properties 16553
TCustomRadioGroup class 12547
about TCustomRadioGroup class 12547
Buttons 12561
CanModify 12559
Columns 12562
Create 12558
Destroy 12558
FlipChildren 12559
GetChildren 12560
ItemIndex 12562
Items 12563
Loaded 12560
ReadState 12560
TCustomRadioGroup Members 12548
TCustomRadioGroup methods 12559
TCustomRadioGroup properties 12561
TCustomRemoteServer class 8525
about TCustomRemoteServer class 8525
AppServer 8532

Create 8529
GetAppServer 8530
GetProviderNames 8530
GetServer 8530
GetServerList 8531
OnGetUsername 8533
SetAppServer 8531
TCustomRemoteServer events 8533
TCustomRemoteServer Members 8526
TCustomRemoteServer methods 8529
TCustomRemoteServer properties 8532
TCustomResolver class 18483
about TCustomResolver class 18483
ApplyUpdates 18490
BeginUpdate 18490
Create 18487
Destroy 18487
DoDelete 18491
DoInsert 18491
DoUpdate 18491
EndUpdate 18492
FErrorCount 18488
FMaxErrors 18488
FPrevResponse 18488
FProvider 18489
FreeTreeData 18492
FUpdateTree 18489
HandleUpdateError 18493
InitializeConflictBuffer 18493
InitKeyFields 18494
InitTreeData 18494
InternalBeforeResolve 18494
InternalUpdateRecord 18495
LogUpdateError 18495
LogUpdateRecord 18495
Provider 18496
RowRequest 18496
TCustomResolver fields 18488
TCustomResolver Members 18484
TCustomResolver methods 18489
TCustomResolver properties 18496
UpdateTree 18497
TCustomRichEdit class 3854
about TCustomRichEdit class 3854
Clear 3867
Create 3866
CreateParams 3868
CreateWnd 3868
DefAttributes 3877
DefaultConverter 3878
Destroy 3866
DestroyWnd 3868
DoSetMaxLength 3869
FindText 3869
GetCaretPos 3871
GetSelLength 3871
GetSelStart 3872
GetSelText 3872
GetSelTextBuf 3872
HideScrollBars 3879
HideSelection 3881
Lines 3882
OnProtectChange 3888
OnResizeRequest 3889
OnSaveClipboard 3890
OnSelectionChange 3890
PageRect 3884
Paragraph 3884
PlainText 3886
Print 3873
RegisterConversionFormat 3874
RequestSize 3874
SelAttributes 3887
SelectionChange 3875
SetCaretPos 3875
SetSelLength 3876
SetSelStart 3876
TCustomRichEdit events 3888

TCustomRichEdit Members 3854
TCustomRichEdit methods 3867
TCustomRichEdit properties 3877
TCustomServerSocket class 18770
about TCustomServerSocket class 18770
Destroy 18774
DoActivate 18775
FServerSocket 18775
GetGetSocketEvent 18776
GetGetThreadEvent 18776
GetOnClientError 18776
GetOnClientEvent 18777
GetOnThreadEnd 18777
GetOnThreadStart 18778
GetServerType 18778
GetThreadCacheSize 18778
OnClientConnect 18784
OnClientDisconnect 18785
OnClientError 18785
OnClientRead 18786
OnClientWrite 18786
OnGetSocket 18787
OnGetThread 18787
OnThreadEnd 18788
OnThreadStart 18788
ServerType 18782
SetGetSocketEvent 18779
SetGetThreadEvent 18779
SetOnClientError 18779
SetOnClientEvent 18780
SetOnThreadEnd 18780
SetOnThreadStart 18781
SetServerType 18781
SetThreadCacheSize 18781
TCustomServerSocket events 18783
TCustomServerSocket fields 18774
TCustomServerSocket Members 18770
TCustomServerSocket methods 18775
TCustomServerSocket properties 18782
ThreadCacheSize 18783
TCustomSessionsService class 25570
about TCustomSessionsService class 25570
CheckStatusChange 25575
Clear 25575
Connect 25575
ConnectActive 25576
Create 25574
DefaultTimeout 25580
DoEndSession 25576
DoStartSession 25577
GetSessionIDs 25577
LoadFromFile 25577
LoadFromStream 25578
MaxSessions 25581
NewSession 25578
NotifyStatusChange 25579
OnEndSession 25581
OnStartSession 25582
SaveToFile 25579
SaveToStream 25580
TCustomSessionsService events 25581
TCustomSessionsService Members 25571
TCustomSessionsService methods 25574
TCustomSessionsService properties 25580
TCustomSocket class 18789
about TCustomSocket class 18789
Error 18793
Event 18794
OnAccept 18795
OnConnect 18795
OnConnecting 18796
OnDisconnect 18796
OnError 18797
OnListen 18797
OnLookup 18797
OnRead 18798
OnWrite 18799
TCustomSocket events 18794

TCustomSocket Members 18790
TCustomSocket methods 18793
TCustomSQLDataSet class 20150
about TCustomSQLDataSet class 20150
Active 20187
AddFieldDesc 20163
AddIndexDefs 20163
AfterClose 20203
AfterOpen 20203
AfterRefresh 20203
AfterScroll 20204
BeforeClose 20204
BeforeOpen 20204
BeforeRefresh 20204
BeforeScroll 20204
BlobBuffer 20189
CheckPrepareError 20164
ClearIndexDefs 20164
CloseCursor 20164
CloseStatement 20164
CommandText 20189
CommandType 20190
Create 20160
CreateBlobStream 20164
CurrentBlobSize 20191
DataLink 20191
DataSource 20192
DefineProperties 20165
DesignerData 20192
Destroy 20161
ExecSQL 20165
ExecuteStatement 20166
FreeBuffers 20166
FreeCommand 20166
FreeReader 20166
GetBlobFieldData 20166
GetCanModify 20167
GetDataSource 20167
GetDetailLinkFields 20167
GetFieldData 20168
GetFieldData method 20168
GetKeyFieldNames 20169
GetMetadata 20193
GetObjectTypeNames 20169
GetOutputParams 20169
GetQuoteChar 20169
GetRecord 20170
GetRecordCount 20170
GetSortFieldNames 20170
IndexDefs 20193
InitRecord 20170
InternalClose 20171
InternalConnection 20194
InternalHandleException 20171
InternalInitFieldDefs 20171
InternalOpen 20171
InternalRefresh 20171
IsCursorOpen 20172
IsSequenced 20172
LastError 20194
Loaded 20172
Locate 20173
LocateRecord 20173
Lookup 20174
MaxBlobSize 20194
NativeCommand 20195
NumericMapping 20195
ObjectView 20196
OnCalcFields 20204
OpenCursor 20174
OpenSchema 20174
ParamByName 20175
ParamCheck 20196
ParamCount 20197
Params 20197
ParseDeleteSql 20205
ParseInsertSql 20205
ParseSelectSql 20205

ParseUpdateSql 20205
Prepared 20198
PrepareStatement 20175
ProcParams 20199
PropertyChanged 20175
PSEndTransaction 20175
PSExecute 20176
PSExecuteStatement 20176
PSGetAttributes 20176
PSGetCommandText 20177
PSGetCommandType 20177
PSGetDefaultOrder 20178
PSGetIndexDefs 20178
PSGetKeyFieldsW 20179
PSGetParams 20179
PSGetQuoteCharW 20179
PSGetTableNameW 20179
PSGetUpdateException 20180
PSInTransaction 20180
PSIsSQLBased 20180
PSIsSQLSupported 20181
PSReset 20181
PSSetCommandText 20182
PSSetParams 20182
PSStartTransaction 20182
PSUpdateRecord 20183
RecordCount 20199
RowsAffected 20200
SchemaInfo 20200
SchemaName 20201
SetBufListSize 20184
SetCommandText 20184
SetFCommandText 20184
SetFieldData 20184
SetMaxBlobSize 20184
SetParamsFromCursor 20185
SetSchemaInfo 20185
SetSortFieldNames 20186
SortFieldNames 20201
SQLConnection 20201
TCustomSQLDataSet events 20203
TCustomSQLDataSet Members 20151
TCustomSQLDataSet methods 20161
TCustomSQLDataSet properties 20186
TransactionLevel 20202
UpdateIndexDefs 20186
TCustomStaticListAction class 15724
about TCustomStaticListAction class 15724
Active 15731
Caption 15732
Count 15732
Create 15729
Destroy 15729
Enabled 15733
GetCount 15730
GetItem 15730
GetItemClass 15730
GetString 15730
HelpContext 15733
Hint 15733
Images 15734
ItemIndex 15734
Items 15734
OnGetItem 15736
OnHint 15736
OnItemSelected 15736
OnUpdate 15737
SecondaryShortCuts 15735
SetString 15731
ShortCut 15735
TCustomStaticListAction events 15736
TCustomStaticListAction Members 15725
TCustomStaticListAction methods 15729
TCustomStaticListAction properties 15731
Visible 15735
TCustomStaticText class 21387
about TCustomStaticText class 21387
Alignment 21400

AutoSize 21400
BorderStyle 21401
Create 21397
CreateParams 21398
FocusControl 21401
Loaded 21399
Notification 21399
SetAutoSize 21399
ShowAccelChar 21401
TCustomStaticText Members 21387
TCustomStaticText methods 21398
TCustomStaticText properties 21400
Transparent 21402
TCustomStatusBar class 3892
about TCustomStatusBar class 3892
AutoHint 3910
Canvas 3910
ChangeScale 3904
Create 3903
CreatePanel 3905
CreatePanels 3905
CreateParams 3905
CreateWnd 3906
Destroy 3903
DoHint 3906
DrawPanel 3907
ExecuteAction 3907
FlipChildren 3908
GetPanelClass 3908
IsFontStored 3908
OnCreatePanelClass 3919
OnDrawPanel 3919
OnHint 3920
Panels 3912
SetBounds 3909
SetParent 3909
SimplePanel 3915
SimpleText 3916
SizeGrip 3918
TCustomStatusBar events 3919
TCustomStatusBar Members 3892
TCustomStatusBar methods 3904
TCustomStatusBar properties 3910
UseSystemFont 3918
TCustomStringsValuesList class 24378
about TCustomStringsValuesList class 24378
Create 24386
Destroy 24386
GetNameFieldClass 24387
GetValueFieldClass 24387
ImplEndIterator 24388
ImplGetListName 24388
ImplGetListNameOfValue 24388
ImplGetListValue 24389
ImplNextIteration 24389
ImplNotifyDeactivate 24389
ImplStartIterator 24390
OnPrepareStrings 24391
PrepareStrings 24390
Strings 24390
StringsPrepared 24391
TCustomStringsValuesList events 24391
TCustomStringsValuesList Members 24380
TCustomStringsValuesList methods 24387
TCustomStringsValuesList properties 24390
TCustomTabControl class 3921
about TCustomTabControl class 3921
AdjustClientRect 3933
CanChange 3933
CanShowTab 3934
Canvas 3940
Change 3934
Create 3932
CreateParams 3935
CreateWnd 3935
Destroy 3932
DisplayRect 3941
DrawTab 3935

GetHitTestInfoAt 3936
GetImageIndex 3936
HotTrack 3941
Images 3941
IndexOfTabAt 3936
Loaded 3937
MultiLine 3942
MultiSelect 3943
Notification 3937
OnChange 3952
OnChanging 3955
OnDrawTab 3956
OnGetImageIndex 3956
OwnerDraw 3944
RaggedRight 3944
RowCount 3938
ScrollOpposite 3944
ScrollTabs 3938
SetTabIndex 3938
Style 3945
TabHeight 3945
TabIndex 3946
TabPosition 3948
TabRect 3939
Tabs 3948
TabStop 3950
TabWidth 3950
TCustomTabControl events 3951
TCustomTabControl Members 3921
TCustomTabControl methods 3933
TCustomTabControl properties 3939
UpdateTabImages 3939
TCustomTaskDialog class 11318
about TCustomTaskDialog class 11318
Button 11328
Buttons 11328
CallbackProc 11323
Caption 11328
CommonButtons 11328
Create 11322
CustomFooterIcon 11329
CustomMainIcon 11329
DefaultButton 11329
Destroy 11322
DoExecute 11323
DoOnButtonClicked 11324
DoOnDialogContructed 11324
DoOnDialogCreated 11324
DoOnDialogDestroyed 11324
DoOnExpandButtonClicked 11325
DoOnHelp 11325
DoOnHyperlinkClicked 11325
DoOnNavigated 11325
DoOnRadioButtonClicked 11326
DoOnTimer 11326
DoOnVerificationClicked 11326
Execute 11327
Execute method 11326
ExpandButtonCaption 11329
Expanded 11330
ExpandedText 11330
Flags 11330
FooterIcon 11331
FooterText 11331
Handle 11331
HelpContext 11332
MainIcon 11332
ModalResult 11332
OnButtonClicked 11334
OnDialogConstructed 11335
OnDialogCreated 11335
OnDialogDestroyed 11335
OnExpanded 11335
OnHyperlinkClicked 11336
OnNavigated 11336
OnRadioButtonClicked 11336
OnTimer 11336
OnVerificationClicked 11337

wwww

ProgressBar 11332
RadioButton 11333
RadioButtons 11333
TCustomTaskDialog events 11334
TCustomTaskDialog Members 11319
TCustomTaskDialog methods 11323
TCustomTaskDialog properties 11327
Text 11333
Title 11333
URL 11334
VerificationText 11334
TCustomTcpServer class 20072
about TCustomTcpServer class 20072
Accept 20077
Accept method 20077
BlockMode 20080
Close 20078
Create 20076
Destroy 20077
DoAccept 20078
Listen 20079
Listening 20080
OnAccept 20081
OnGetThread 20081
OnListening 20082
Open 20079
ServerSocketThread 20080
TCustomTcpServer events 20081
TCustomTcpServer Members 20073
TCustomTcpServer methods 20077
TCustomTcpServer properties 20080
WaitForConnection 20079
TCustomToolScrollBtn class 217
about TCustomToolScrollBtn class 217
Create 227
DrawArrows 227
TCustomToolScrollBtn Members 218
TCustomToolScrollBtn methods 227
TCustomTransparentControl class 5970
about TCustomTransparentControl class 5970
Create 5980
CreateParams 5981
InterceptMouse 5983
Invalidate 5982
InvalidateControlsUnderneath 5982
TCustomTransparentControl Members 5970
TCustomTransparentControl methods 5981
TCustomTransparentControl properties 5983
WMNCHitTest 5982
TCustomTrayIcon class 12564
about TCustomTrayIcon class 12564
Animate 12573
AnimateInterval 12574
BalloonFlags 12574
BalloonHint 12574
BalloonTimeout 12575
BalloonTitle 12575
Create 12568
Data 12575
Destroy 12568
DoOnAnimate 12569
GetAnimateInterval 12569
GetBalloonTimeout 12569
Hint 12575
Icon 12576
IconIndex 12576
Icons 12576
OnAnimate 12577
OnClick 12577
OnDblClick 12578
OnMouseDown 12578
OnMouseMove 12578
OnMouseUp 12578
PopupMenu 12576
Refresh 12570
Refresh method 12570
SetAnimate 12570
SetAnimateInterval 12570

SetBalloonHint 12571
SetBalloonTimeout 12571
SetBalloonTitle 12571
SetDefaultIcon 12571
SetHint 12571
SetIcon 12572
SetIconIndex 12572
SetIconList 12572
SetVisible 12572
ShowBalloonHint 12573
TCustomTrayIcon events 12577
TCustomTrayIcon Members 12564
TCustomTrayIcon methods 12568
TCustomTrayIcon properties 12573
Visible 12577
WindowProc 12573
TCustomTreeView class 3957
about TCustomTreeView class 3957
Added 3972
AlphaSort 3972
AutoExpand 3996
BorderStyle 3997
CanChange 3972
CanCollapse 3973
CanEdit 3973
CanExpand 3974
Canvas 3997
Change 3974
ChangeDelay 3999
ClearSelection 3974
Collapse 3975
Create 3969
CreateNode 3975
CreateNodes 3976
CreateParams 3976
CreateWnd 3976
CreateWndRestores 4000
CustomDraw 3977
CustomDrawItem 3977
CustomSort 3978
Delete 3979
Deselect 3980
Destroy 3970
DestroyWnd 3980
DoEndDrag 3981
DoEnter 3981
DoExit 3981
DoStartDrag 3982
DropTarget 4000
Edit 3982
Expand 3983
FChangeTimer 3970
FindNextToSelect 3983
FullCollapse 3983
FullExpand 3984
GetDragImages 3984
GetHitTestInfoAt 3985
GetImageIndex 3986
GetNodeAt 3987
GetSelectedIndex 3988
GetSelections 3989
HideSelection 4000
HotTrack 4001
Images 4001
Indent 4001
InvalidateSelectionsRects 3989
IsCustomDrawn 3989
IsEditing 3990
Items 4004
Loaded 3990
LoadFromFile 3990
LoadFromStream 3991
MouseDown 3991
MultiSelect 4005
MultiSelectStyle 4005
Notification 3992
OnAddition 4016
OnAdvancedCustomDraw 4016

OnAdvancedCustomDrawItem 4016
OnCancelEdit 4017
OnChange 4017
OnChanging 4018
OnCollapsed 4018
OnCollapsing 4018
OnCompare 4019
OnCreateNodeClass 4019
OnCustomDraw 4019
OnCustomDrawItem 4021
OnDeletion 4024
OnEdited 4024
OnEditing 4024
OnExpanded 4025
OnExpanding 4025
OnGetImageIndex 4026
OnGetSelectedIndex 4026
ReadOnly 4006
RightClickSelect 4006
RowSelect 4007
SaveToFile 3992
SaveToStream 3993
Select 3994
Select method 3993
Selected 4007
SelectionCount 4012
Selections 4012
SetDragMode 3994
ShowButtons 4013
ShowLines 4013
ShowRoot 4013
SortType 4014
StateImages 4014
Subselect 3994
TCustomTreeView events 4015
TCustomTreeView fields 3970
TCustomTreeView Members 3957
TCustomTreeView methods 3970
TCustomTreeView properties 3995
ToolTips 4014
TopItem 4015
ValidateSelection 3995
WndProc 3995
TCustomUpDown class 4027
about TCustomUpDown class 4027
AlignButton 4042
ArrowKeys 4042
Associate 4043
CanChange 4039
Click 4039
Create 4038
CreateParams 4039
CreateWnd 4040
DoCanChange 4040
Increment 4043
Max 4043
Min 4044
Notification 4041
OnChanging 4046
OnChangingEx 4047
OnClick 4047
Orientation 4044
Position 4045
TCustomUpDown events 4046
TCustomUpDown Members 4028
TCustomUpDown methods 4038
TCustomUpDown properties 4041
Thousands 4045
Wrap 4045
TCustomUtilityButton class 228
about TCustomUtilityButton class 228
ArrowSize 240
ArrowType 241
AutoScroll 241
Click 238
CMMouseEnter 238
CMMouseLeave 239
Color 241

Create 237
Destroy 237
Direction 242
DrawArrows 239
MouseDown 239
OnClick 243
OnDelay 240
Paint 240
RepeatRate 242
TCustomUtilityButton events 242
TCustomUtilityButton Members 228
TCustomUtilityButton methods 238
TCustomUtilityButton properties 240
TCustomVariantType class 23581
_AddRef 23586
_Release 23587
about TCustomVariantType class 23581
BinaryOp 23587
Cast 23588
CastTo 23589
CastToOle 23589
Clear 23589
Compare 23590
CompareOp 23590
Copy 23591
Create 23584
Create constructor 23584
Destroy 23585
DispInvoke 23592
IsClear 23592
LeftPromotion 23593
OlePromotion 23594
QueryInterface 23594
RaiseCastError 23595
RaiseDispError 23595
RaiseInvalidOp 23596
RightPromotion 23596
SimplisticClear 23597
SimplisticCopy 23598
TCustomVariantType Members 23582
TCustomVariantType methods 23585
TCustomVariantType properties 23606
UnaryOp 23598
VarDataCast 23599
VarDataCastTo 23599
VarDataCastTo method 23599
VarDataCastToOleStr 23600
VarDataClear 23600
VarDataCopy 23601
VarDataCopyNoInd 23601
VarDataFromOleStr 23602
VarDataFromStr 23602
VarDataInit 23602
VarDataIsArray 23603
VarDataIsByRef 23603
VarDataIsEmptyParam 23603
VarDataIsFloat 23604
VarDataIsNumeric 23604
VarDataIsOrdinal 23605
VarDataIsStr 23605
VarDataToStr 23605
VarType 23606
TCustomVirtualListAction class 15738
about TCustomVirtualListAction class 15738
Count 15744
GetItem 15743
GetString 15743
OnGetItem 15744
TCustomVirtualListAction events 15744
TCustomVirtualListAction Members 15739
TCustomVirtualListAction methods 15743
TCustomVirtualListAction properties 15744
TCustomWebAppComponents class 24906
about TCustomWebAppComponents class 24906
AdapterDispatcher 24919
ApplicationAdapter 24919
CreateContext 24911
DispatchActions 24919

EndUserAdapter 24920
GetAdapterDispatcher 24912
GetLocateFileService 24912
GetPageDispatcher 24912
GetWebApplicationInfo 24913
GetWebEndUser 24913
GetWebUserList 24913
ImplFinishContext 24914
ImplHandleException 24914
ImplHandleRequest 24915
ImplInitContext 24915
InitModule 24916
InitModules 24917
ListRequestHandlers 24917
LocateFileService 24920
Notification 24918
OnException 24922
PageDispatcher 24921
Sessions 24921
TCustomWebAppComponents events 24922
TCustomWebAppComponents Members 24907
TCustomWebAppComponents methods 24911
TCustomWebAppComponents properties 24918
UserListService 24922
TCustomWebAppDataModule class 25332
about TCustomWebAppDataModule class 25332
AppServices 25339
Create 25336
CreateNew 25337
Destroy 25337
GetWebAppServices 25337
Notification 25338
QueryInterface 25338
TCustomWebAppDataModule Members 25333
TCustomWebAppDataModule methods 25337
TCustomWebAppDataModule properties 25339
TCustomWebAppPageModule class 25339
about TCustomWebAppPageModule class 25339
AppServices 25347
Create 25344
CreateNew 25345
Destroy 25345
GetWebAppServices 25346
Notification 25346
QueryInterface 25346
SetAppServices 25347
TCustomWebAppPageModule Members 25340
TCustomWebAppPageModule methods 25345
TCustomWebAppPageModule properties 25347
TCustomWebDataModule class 25348
about TCustomWebDataModule class 25348
Create 25352
CreateNew 25352
Destroy 25353
GetSiteModuleHelper 25353
GetWebSnapObjClass 25354
Notification 25354
OnActivate 25356
OnDeactivate 25357
Request 25355
Response 25355
Session 25355
SiteModuleHelper 25356
TCustomWebDataModule events 25356
TCustomWebDataModule Members 25349
TCustomWebDataModule methods 25353
TCustomWebDataModule properties 25354
TCustomWebDispatcher class 15110
about TCustomWebDispatcher class 15110
Access_Request 15116
Access_Response 15117
Action 15121
ActionByName 15117
Actions 15122
AfterDispatch 15123
BeforeDispatch 15124
Create 15115
Destroy 15115

DispatchAction 15117
DoAfterDispatch 15118
DoBeforeDispatch 15119
FinishContext 15119
HandleException 15119
HandleRequest 15120
InitContext 15120
InitModule 15120
Notification 15121
OnException 15124
Request 15122
Response 15122
TCustomWebDispatcher events 15123
TCustomWebDispatcher Members 15112
TCustomWebDispatcher methods 15116
TCustomWebDispatcher properties 15121
TCustomWebPageModule class 25357
about TCustomWebPageModule class 25357
Create 25362
CreateNew 25362
DefaultAction 25365
Destroy 25363
GetSiteModuleHelper 25363
GetSitePageModuleHelper 25364
Notification 25364
OnAfterDispatchPage 25366
OnAfterRedirectToPage 25367
OnBeforeDispatchPage 25367
OnBeforeRedirectToPage 25368
PageProducer 25365
SitePageModuleHelper 25366
TCustomWebPageModule events 25366
TCustomWebPageModule Members 25358
TCustomWebPageModule methods 25363
TCustomWebPageModule properties 25364
TCustomWebUserList class 25606
about TCustomWebUserList class 25606
CheckAccessRights 25610
Create 25609
Destroy 25610
LoadFromFile 25611
OnAfterCheckAccessRights 25613
OnAfterValidateUser 25613
OnBeforeCheckAccessRights 25613
OnBeforeValidateUser 25614
OnUserIDNotFound 25614
OnValidateUserError 25614
SaveToFile 25611
TCustomWebUserList events 25612
TCustomWebUserList Members 25606
TCustomWebUserList methods 25610
TCustomWebUserList properties 25611
UserItems 25612
Users 25612
ValidateUser 25611
TCustomWinSocket class 18799
about TCustomWinSocket class 18799
Accept 18803
Addr 18817
AsyncInitSocket 18803
ASyncStyles 18817
Close 18804
Connect 18805
Connected 18818
Create 18801
Data 18818
DefaultHandler 18805
Destroy 18802
Disconnect 18805
DoListen 18806
DoOpen 18806
Error 18806
Event 18807
Handle 18819
InitSocket 18807
Listen 18808
LocalAddress 18819
LocalHost 18820

LocalPort 18820
Lock 18809
LookupName 18809
LookupService 18810
LookupState 18821
OnErrorEvent 18824
OnSocketEvent 18824
Open 18810
Read 18811
ReceiveBuf 18812
ReceiveLength 18812
ReceiveText 18812
RemoteAddr 18821
RemoteAddress 18822
RemoteHost 18822
RemotePort 18823
SendBuf 18813
SendStream 18814
SendStreamThenDrop 18814
SendText 18814
SetAsyncStyles 18815
SocketHandle 18823
TCustomWinSocket events 18824
TCustomWinSocket Members 18800
TCustomWinSocket methods 18802
TCustomWinSocket properties 18816
Unlock 18815
Write 18816
TCustomXSLPageProducer class 26588
about TCustomXSLPageProducer class 26588
Content 26597
ContentFromStream 26598
ContentFromString 26598
ContentFromWideString 26599
Dispatcher 26603
DispatcherComponent 26604
GetAppDispatcher 26599
GetProducerTemplateFileName 26599
GetProducerTemplateStream 26599
GetTemplateFileType 26600
GetTemplateStream 26600
GetXMLData 26600
GetXMLDocument 26600
GetXSLData 26601
HasHTMLBrowserView 26601
HasScriptView 26601
HasXMLBrowserView 26601
HasXSLBrowserView 26601
Notification 26602
ProduceContent 26602
ProduceContentFromStream 26602
ProduceContentFromString 26603
SetAppDispatcher 26603
TCustomXSLPageProducer Members 26589
TCustomXSLPageProducer methods 26596
TCustomXSLPageProducer properties 26603
XMLData 26605
TCustomZlibStream class 26623
about TCustomZlibStream class 26623
Create 26625
OnProgress 26626
Progress 26626
TCustomZlibStream events 26626
TCustomZlibStream Members 26624
TCustomZlibStream methods 26625
TDatabase class 9964
about TDatabase class 9964
AfterConnect 9992
AfterDisconnect 9993
AliasName 9979
ApplyUpdates 9970
BeforeConnect 9993
BeforeDisconnect 9993
CloseDataSets 9971
Commit 9971
Connected 9979
Create 9969
DatabaseName 9980

DataSets 9981
Destroy 9969
Directory 9981
DoConnect 9972
DoDisconnect 9972
DriverName 9981
Exclusive 9982
Execute 9973
FlushSchemaCache 9973
GetConnected 9974
GetDataSet 9974
GetFieldNames 9975
GetTableNames 9975
Handle 9983
HandleShared 9983
InTransaction 9983
IsSQLBased 9984
KeepConnection 9984
Loaded 9975
Locale 9985
LoginPrompt 9985
Notification 9976
OnLogin 9994
Params 9986
ReadOnly 9987
Rollback 9976
Session 9988
SessionAlias 9988
SessionName 9989
StartTransaction 9977
TDatabase events 9992
TDatabase Members 9965
TDatabase methods 9970
TDatabase properties 9978
Temporary 9989
TraceFlags 9990
TransIsolation 9991
ValidateName 9977
TDatabaseSprig class 1733
about TDatabaseSprig class 1733
AnyProblems 1734
Caption 1734
DragDropTo 1734
DragOverTo 1735
FigureParent 1735
Name 1735
PaletteOverTo 1735
TDatabaseSprig Members 1733
TDatabaseSprig methods 1734
TDataBindings class 9802
about TDataBindings class 9802
Add 9804
Create 9804
GetItemByDispID 9805
GetOwner 9805
Items 9806
TDataBindings Members 9802
TDataBindings methods 9804
TDataBindings properties 9806
Update 9805
TDataBindItem class 9806
about TDataBindItem class 9806
Create 9808
DataField 9809
Destroy 9808
DispID 9809
TDataBindItem Members 9807
TDataBindItem properties 9809
TDataBlock class 18903
about TDataBlock class 18903
BytesReserved 18911
Clear 18906
Create 18905
Destroy 18905
GetBytesReserved 18907
GetMemory 18907
GetSignature 18907
GetSize 18907

GetStream 18908
IgnoreStream 18908
InitData 18908
Memory 18911
Read 18909
SetSignature 18909
SetSize 18910
Signature 18911
Size 18912
Stream 18912
TDataBlock Members 18903
TDataBlock methods 18906
TDataBlock properties 18910
Write 18910
TDataBlockInterpreter class 18913
about TDataBlockInterpreter class 18913
AddDispatch 18915
CallCreateObject 18915
CallFreeObject 18916
CallGetIDsOfNames 18916
CallGetServerList 18916
CallInvoke 18917
CanCreateObject 18917
Create 18914
CreateObject 18917
Destroy 18914
DoCreateObject 18917
DoCustomAction 18917
DoFreeObject 18918
DoGetAppServerList 18918
DoGetIDsOfNames 18918
DoGetServerList 18918
DoInvoke 18919
InternalCreateObject 18919
InterpretData 18919
LockObject 18919
ReleaseObject 18920
RemoveDispatch 18920
StoreObject 18920
TDataBlockInterpreter Members 18913
TDataBlockInterpreter methods 18915
UnlockObject 18920
TDataCache class 17797
about TDataCache class 17797
Active 17806
AddAggIndex 17800
AddIndex 17800
AppendDimension 17800
AppendSummary 17801
CalcSubTotals 17801
ClearIndexInfo 17801
Create 17799
CreateTable 17801
CurrentSummary 17807
Destroy 17799
DimensionCount 17807
DimensionFromFieldName 17801
Dimensions 17807
ErrorCode 17807
FreeCache 17802
GetDimensionMember 17802
GetDimensionMemberAsVariant 17802
GetDimensionMemberCount 17802
GetDimensionName 17803
GetDomain 17803
GetIndexCount 17803
GetMemoryUsage 17803
GetSummaryAsString 17803
GetSummaryAsVariant 17804
GetSummaryName 17804
HasIndex 17807
HasSubTotals 17804
HasValidSubTotals 17804
IncSummaryIndex 17805
IndexCount 17808
IsDimension 17805
IsSummary 17805
PreCalculateTotals 17808

Sparsing 17808
Success 17808
Summaries 17809
SummaryCount 17809
SummaryFromCubeDimIndex 17805
SummaryFromFieldName 17805
SummaryFromPosition 17806
TDataCache Members 17798
TDataCache methods 17799
TDataCache properties 17806
TDataContext class 15501
about TDataContext class 15501
AddDynArrayToClear 15505
AddObjectToDestroy 15506
AddStrToClear 15506
AddVariantToClear 15507
AddWStrToClear 15507
AllocData 15508
Create 15502
Data 15503
DataOffset 15503
DataP 15503
Destroy 15502
DynArrayToClear 15504
FObjsToDestroy 15504
GetDataPointer 15508
RemoveObjectToDestroy 15508
SetDataPointer 15509
StrToClear 15504
TDataContext fields 15503
TDataContext Members 15501
TDataContext methods 15505
VarToClear 15504
WStrToClear 15504
TDataDispatch class 18921
about TDataDispatch class 18921
Create 18922
Destroy 18922
DispatchIndex 18924
GetIDsOfNames 18923
GetTypeInfo 18924
GetTypeInfoCount 18924
Invoke 18924
TDataDispatch Members 18921
TDataDispatch methods 18923
TDataDispatch properties 18924
TDataForm class 16554
about TDataForm class 16554
Custom 16559
Style 16559
StyleRule 16560
TDataForm Members 16555
TDataForm properties 16559
TDataGrid class 16560
about TDataGrid class 16560
DisplayRows 16567
HeadingAttributes 16567
RowAttributes 16568
TableAttributes 16568
TDataGrid Members 16562
TDataGrid properties 16567
TDataLink class 6876
about TDataLink class 6876
Active 6888
ActiveChanged 6880
ActiveRecord 6889
BOF 6889
BufferCount 6890
CheckBrowseMode 6880
Create 6878
DataEvent 6881
DataSet 6890
DataSetChanged 6881
DataSetScrolled 6882
DataSource 6891
DataSourceFixed 6891
Destroy 6879
Edit 6882

Editing 6892
EditingChanged 6883
Eof 6892
ExecuteAction 6883
FocusControl 6884
GetActiveRecord 6884
GetBOF 6884
GetBufferCount 6884
GetEOF 6885
GetRecordCount 6885
LayoutChanged 6885
MoveBy 6885
ReadOnly 6893
RecordChanged 6886
RecordCount 6893
SetActiveRecord 6886
SetBufferCount 6887
TDataLink Members 6877
TDataLink methods 6879
TDataLink properties 6888
UpdateAction 6887
UpdateData 6887
UpdateRecord 6888
VisualControl 6894
TDataModule class 2824
about TDataModule class 2824
AfterConstruction 2829
BeforeDestruction 2830
Create 2827
CreateNew 2828
DefineProperties 2830
DesignOffset 2832
DesignSize 2832
Destroy 2829
DoCreate 2831
DoDestroy 2831
GetChildren 2831
HandleCreateException 2831
OldCreateOrder 2833
OnCreate 2833
OnDestroy 2834
ReadState 2832
TDataModule events 2833
TDataModule Members 2824
TDataModule methods 2829
TDataModule properties 2832
TDataNavigator class 16569
about TDataNavigator class 16569
Custom 16574
Style 16574
StyleRule 16575
TDataNavigator Members 16569
TDataNavigator properties 16574
XMLComponent 16575
TDataPacketWriter class 18497
about TDataPacketWriter class 18497
AddColumn 18502
AddConstraints 18502
AddDataSetAttributes 18502
AddExtraFieldProps 18502
AddFieldLinks 18503
AddIndexDefs 18503
Constraints 18507
Destroy 18499
FConstraints 18500
FOnGetParams 18500
FOptions 18500
FPacketOptions 18501
FPutFieldInfo 18501
FreeInfoRecords 18503
GetDataPacket 18503
GetFieldIdx 18503
InitPutProcs 18504
OnGetParams 18508
Options 18507
PacketOptions 18507
PutADTField 18504
PutArrayField 18504

PutBlobField 18504
PutCalcField 18505
PutDataSetField 18505
PutField 18505
PutStringField 18505
PutVarBytesField 18505
PutWideStringField 18506
RefreshPutProcs 18506
Reset 18506
TDataPacketWriter events 18508
TDataPacketWriter fields 18500
TDataPacketWriter Members 18497
TDataPacketWriter methods 18501
TDataPacketWriter properties 18507
WriteDataSet 18506
WriteMetaData 18507
TDataSet class 6894
about TDataSet class 6894
ActivateBuffers 6909
Active 7004
ActiveBuffer 6909
ActiveRecord 7005
AfterCancel 7038
AfterClose 7040
AfterDelete 7041
AfterEdit 7044
AfterInsert 7044
AfterOpen 7045
AfterPost 7046
AfterRefresh 7047
AfterScroll 7047
AggFields 7006
AllocRecordBuffer 6909
Append 6910
AppendRecord 6911
AutoCalcFields 7006
BeforeCancel 7048
BeforeClose 7048
BeforeDelete 7049
BeforeEdit 7049
BeforeInsert 7050
BeforeOpen 7051
BeforePost 7052
BeforeRefresh 7052
BeforeScroll 7053
BindFields 6915
BlobFieldCount 7007
BlockReadNext 6915
BlockReadSize 7007
Bof 7008
Bookmark 7009
BookmarkAvailable 6916
BookmarkSize 7010
BookmarkValid 6916
BufferCount 7010
Buffers 7011
CalcBuffer 7011
CalcFieldsSize 7012
CalculateFields 6916
Cancel 6916
CanModify 7012
CheckActive 6917
CheckBiDirectional 6918
CheckBrowseMode 6918
CheckFieldCompatibility 6919
CheckInactive 6919
ClearBuffers 6919
ClearCalcFields 6919
ClearFields 6919
Close 6920
CloseBlob 6921
CloseCursor 6921
CompareBookmarks 6921
Constraints 7013
ControlsDisabled 6921
CopyFields 6922
Create 6904
CreateBlobStream 6922

iiii

CreateFields 6923
CreateNestedDataSet 6923
CurrentRecord 7013
CursorPosChanged 6923
DataConvert 6924
DataEvent 6924
DataSetField 7013
DataSource 7014
DefaultFields 7014
DefChanged 6924
Delete 6924
Designer 7015
Destroy 6904
DestroyFields 6925
DisableControls 6925
DoAfterCancel 6927
DoAfterClose 6927
DoAfterDelete 6928
DoAfterEdit 6928
DoAfterInsert 6928
DoAfterOpen 6928
DoAfterPost 6929
DoAfterRefresh 6929
DoAfterScroll 6929
DoBeforeCancel 6929
DoBeforeClose 6929
DoBeforeDelete 6930
DoBeforeEdit 6930
DoBeforeInsert 6930
DoBeforeOpen 6930
DoBeforePost 6931
DoBeforeRefresh 6931
DoBeforeScroll 6931
DoOnCalcFields 6931
DoOnNewRecord 6931
Edit 6932
EnableControls 6934
Eof 7015
FieldByName 6936
FieldByNumber 6938
FieldCount 7017
FieldDefList 7019
FieldDefs 7019
FieldList 7022
FieldNoOfs 7022
Fields 7023
FieldValues 7025
Filter 7026
Filtered 7027
FilterOptions 7027
FindField 6938
FindFirst 6939
FindLast 6940
FindNext 6940
FindPrior 6941
FindRecord 6942
First 6942
Found 7028
FreeBookmark 6945
FreeFieldBuffers 6946
FreeRecordBuffer 6947
GetAggRecordCount 6947
GetAggregateValue 6947
GetBlobFieldData 6947
GetBookmark 6948
GetBookmarkData 6949
GetBookmarkFlag 6949
GetBookmarkStr 6949
GetCalcFields 6950
GetCanModify 6950
GetChildren 6950
GetCurrentRecord 6950
GetDataSource 6951
GetDetailDataSets 6951
GetDetailLinkFields 6951
GetFieldClass 6952
GetFieldData 6952
GetFieldData method 6952

GetFieldFullName 6953
GetFieldList 6953
GetFieldNames 6954
GetFieldNames method 6953
GetIndexDefs 6954
GetIsIndexField 6954
GetNextRecord 6955
GetNextRecords 6955
GetPriorRecord 6955
GetPriorRecords 6955
GetRecNo 6956
GetRecord 6956
GetRecordCount 6956
GetRecordSize 6956
GetStateFieldValue 6956
GotoBookmark 6957
InitFieldDefs 6958
InitFieldDefsFromFields 6959
InitRecord 6959
Insert 6959
InsertRecord 6961
InternalAddRecord 6963
InternalCalcFields 7028
InternalCancel 6963
InternalClose 6963
InternalDelete 6964
InternalEdit 6964
InternalFirst 6964
InternalGotoBookmark 6964
InternalHandleException 6964
InternalInitFieldDefs 6965
InternalInitRecord 6965
InternalInsert 6965
InternalLast 6965
InternalOpen 6966
InternalPost 6966
InternalRefresh 6966
InternalSetToRecord 6966
IsCursorOpen 6966
IsEmpty 6967
IsLinkedTo 6967
IsSequenced 6967
IsUniDirectional 7029
Last 6968
Loaded 6969
Locate 6970
Lookup 6970
Modified 7029
MoveBy 6971
NestedDataSetClass 7030
NestedDataSets 7030
Next 6973
ObjectView 7031
OnCalcFields 7053
OnDeleteError 7054
OnEditError 7054
OnFilterRecord 7055
OnNewRecord 7056
OnPostError 7056
Open 6976
OpenCursor 6976
OpenCursorComplete 6977
OpenParentDataSet 6977
Post 6977
Prior 6979
PSEndTransaction 6979
PSExecute 6980
PSExecuteStatement 6981
PSExecuteStatement method 6980
PSGetAttributes 6981
PSGetCommandText 6982
PSGetCommandTextW 6982
PSGetCommandType 6982
PSGetDefaultOrder 6983
PSGetIndexDefs 6983
PSGetKeyFields 6984
PSGetKeyFieldsW 6984
PSGetParams 6984

PSGetQuoteChar 6985
PSGetQuoteCharW 6985
PSGetTableName 6985
PSGetTableNameW 6986
PSGetUpdateException 6986
PSInTransaction 6987
PSIsSQLBased 6987
PSIsSQLSupported 6987
PSReset 6988
PSSetCommandText 6988, 6989
PSSetCommandText method 6988
PSSetParams 6989
PSStartTransaction 6989
PSUpdateRecord 6990
RecNo 7031
RecordCount 7032
RecordSize 7035
Refresh 6991
RefreshInternalCalcFields 6993
Reserved 7035
ResetAggField 6993
RestoreState 6993
Resync 6994
SetActive 6994
SetBlockReadSize 6994
SetBookmarkData 6994
SetBookmarkFlag 6995
SetBookmarkStr 6995
SetBufListSize 6995
SetChildOrder 6995
SetCurrentRecord 6996
SetDataSetField 6996
SetDefaultFields 6996
SetFieldData 6996, 6997
SetFieldData method 6996
SetFields 6997
SetFiltered 6997
SetFilterOptions 6998
SetFilterText 6998
SetFound 6998
SetModified 6998
SetName 6999
SetObjectView 6999
SetOnFilterRecord 6999
SetRecNo 6999
SetState 7000
SetStateFieldValue 7000
SetTempState 7000
SetUniDirectional 7000
SparseArrays 7036
State 7036
TDataSet events 7037
TDataSet Members 6896
TDataSet methods 6905
TDataSet properties 7003
TempBuffer 7001
Translate 7001
UpdateBufferCount 7001
UpdateCursorPos 7001
UpdateIndexDefs 7002
UpdateRecord 7002
UpdateStatus 7002
TDataSetAction class 7650
about TDataSetAction class 7650
DataSource 7657
GetDataSet 7655
HandlesTarget 7656
Notification 7656
TDataSetAction Members 7651
TDataSetAction methods 7655
TDataSetAction properties 7656
TDataSetAdapter class 7872
about TDataSetAdapter class 7872
Actions 7883
Data 7883
DataSet 7884
MasterAdapter 7884
ModifyAccess 7884

OnAfterExecuteAction 7886
OnAfterGetActionResponse 7886
OnAfterUpdateFields 7887
OnAfterValidateFields 7887
OnBeforeExecuteAction 7887
OnBeforeGetActionResponse 7888
OnBeforeUpdateFields 7888
OnBeforeValidateFields 7888
OnGetActionParams 7889
OnGetRecordCount 7889
OnPrepareDataSet 7889
PageSize 7884
RefreshAfterPost 7885
TDataSetAdapter events 7885
TDataSetAdapter Members 7874
TDataSetAdapter properties 7883
ViewAccess 7885
TDataSetAdapterApplyRowAction class 7890
about TDataSetAdapterApplyRowAction class 7890
ActionModes 7899
Create 7897
DisplayLabel 7900
ExecuteAccess 7900
GetDefaultActionName 7898
ImplExecuteActionRequest 7898
ImplIsDefaultAction 7898
ImplWebEnabled 7899
OnAfterExecute 7901
OnAfterGetResponse 7902
OnBeforeExecute 7902
OnBeforeGetResponse 7902
OnGetParams 7903
RedirectOptions 7901
TDataSetAdapterApplyRowAction events 7901
TDataSetAdapterApplyRowAction Members 7891
TDataSetAdapterApplyRowAction methods 7897
TDataSetAdapterApplyRowAction properties 7899
TDataSetAdapterBrowseRowAction class 7903
about TDataSetAdapterBrowseRowAction class 7903
ActionModes 7913
Create 7910
DisplayLabel 7913
ExecuteAccess 7914
GetDefaultActionName 7911
ImplExecuteActionRequest 7911
ImplIsDefaultAction 7912
ImplWebEnabled 7912
OnAfterExecute 7914
OnAfterGetResponse 7915
OnBeforeExecute 7915
OnBeforeGetResponse 7915
OnGetParams 7916
TDataSetAdapterBrowseRowAction events 7914
TDataSetAdapterBrowseRowAction Members 7905
TDataSetAdapterBrowseRowAction methods 7911
TDataSetAdapterBrowseRowAction properties 7913
TDataSetAdapterCancelRowAction class 7916
about TDataSetAdapterCancelRowAction class 7916
ActionModes 7926
Create 7924
DisplayLabel 7926
ExecuteAccess 7927
GetDefaultActionName 7924
ImplExecuteActionRequest 7925
ImplIsDefaultAction 7925
ImplWebEnabled 7925
OnAfterExecute 7928
OnAfterGetResponse 7928
OnBeforeExecute 7929
OnBeforeGetResponse 7929
OnGetParams 7929
RedirectOptions 7927
TDataSetAdapterCancelRowAction events 7928
TDataSetAdapterCancelRowAction Members 7918
TDataSetAdapterCancelRowAction methods 7924
TDataSetAdapterCancelRowAction properties 7926
TDataSetAdapterDataLink class 7930
about TDataSetAdapterDataLink class 7930

Create 7932
DataEvent 7932
OnDataSetStateChange 7934
RecordChanged 7933
TDataSetAdapterDataLink events 7934
TDataSetAdapterDataLink Members 7930
TDataSetAdapterDataLink methods 7932
TDataSetAdapterDeleteRowAction class 7934
about TDataSetAdapterDeleteRowAction class 7934
DisplayLabel 7943
ExecuteAccess 7944
GetDefaultActionName 7942
ImplExecuteActionRequest 7942
ImplIsDefaultAction 7943
OnAfterExecute 7945
OnAfterGetResponse 7945
OnBeforeExecute 7945
OnBeforeGetResponse 7946
OnGetParams 7946
RedirectOptions 7944
TDataSetAdapterDeleteRowAction events 7944
TDataSetAdapterDeleteRowAction Members 7936
TDataSetAdapterDeleteRowAction methods 7942
TDataSetAdapterDeleteRowAction properties 7943
TDataSetAdapterEditRowAction class 7947
about TDataSetAdapterEditRowAction class 7947
ActionModes 7956
Create 7954
DisplayLabel 7957
ExecuteAccess 7957
GetDefaultActionName 7954
ImplExecuteActionRequest 7955
ImplIsDefaultAction 7955
ImplWebEnabled 7955
OnAfterExecute 7958
OnAfterGetResponse 7958
OnBeforeExecute 7958
OnBeforeGetResponse 7959
OnGetParams 7959
TDataSetAdapterEditRowAction events 7957
TDataSetAdapterEditRowAction Members 7948
TDataSetAdapterEditRowAction methods 7954
TDataSetAdapterEditRowAction properties 7956
TDataSetAdapterField class 7960
about TDataSetAdapterField class 7960
DataSetField 7967
FieldFlags 7968
FieldModes 7968
ModifyAccess 7969
OnGetDisplayText 7971
OnGetValue 7971
OnUpdateValue 7969
OnValidateValue 7972
TDataSetAdapterField events 7971
TDataSetAdapterField Members 7961
TDataSetAdapterField properties 7967
ValuesList 7970
ViewAccess 7970
TDataSetAdapterFirstRowAction class 7972
about TDataSetAdapterFirstRowAction class 7972
DisplayLabel 7981
ExecuteAccess 7981
GetDefaultActionName 7980
ImplExecuteActionRequest 7980
ImplWebEnabled 7980
OnAfterExecute 7982
OnAfterGetResponse 7983
OnBeforeExecute 7983
OnBeforeGetResponse 7984
OnGetParams 7984
RedirectOptions 7982
TDataSetAdapterFirstRowAction events 7982
TDataSetAdapterFirstRowAction Members 7974
TDataSetAdapterFirstRowAction methods 7979
TDataSetAdapterFirstRowAction properties 7981
TDataSetAdapterImageField class 8138
about TDataSetAdapterImageField class 8138
DataSetField 8145

FieldModes 8146
ModifyAccess 8146
OnGetHREF 8147
OnGetImage 8148
TDataSetAdapterImageField events 8147
TDataSetAdapterImageField Members 8139
TDataSetAdapterImageField properties 8145
ViewAccess 8147
TDataSetAdapterLastRowAction class 7984
about TDataSetAdapterLastRowAction class 7984
DisplayLabel 7993
ExecuteAccess 7993
GetDefaultActionName 7992
ImplExecuteActionRequest 7992
ImplWebEnabled 7992
OnAfterExecute 7994
OnAfterGetResponse 7995
OnBeforeExecute 7995
OnBeforeGetResponse 7996
OnGetParams 7996
RedirectOptions 7994
TDataSetAdapterLastRowAction events 7994
TDataSetAdapterLastRowAction Members 7986
TDataSetAdapterLastRowAction methods 7991
TDataSetAdapterLastRowAction properties 7993
TDataSetAdapterMemoField class 7996
about TDataSetAdapterMemoField class 7996
DataSetField 8004
FieldFlags 8004
FieldModes 8005
ModifyAccess 8005
OnGetDisplayText 8007
OnGetValue 8007
OnUpdateValue 8006
OnValidateValue 8008
TDataSetAdapterMemoField events 8007
TDataSetAdapterMemoField Members 7998
TDataSetAdapterMemoField properties 8004
ViewAccess 8006
TDataSetAdapterMoveRowAction class 8008
about TDataSetAdapterMoveRowAction class 8008
ImplIsDefaultAction 8016
TDataSetAdapterMoveRowAction Members 8010
TDataSetAdapterMoveRowAction methods 8015
TDataSetAdapterNewRowAction class 8016
about TDataSetAdapterNewRowAction class 8016
ActionModes 8026
Create 8024
DisplayLabel 8027
ExecuteAccess 8027
GetDefaultActionName 8024
ImplExecuteActionRequest 8025
ImplIsDefaultAction 8025
ImplWebEnabled 8026
OnAfterExecute 8028
OnAfterGetResponse 8028
OnBeforeExecute 8029
OnBeforeGetResponse 8029
OnGetParams 8029
TDataSetAdapterNewRowAction events 8027
TDataSetAdapterNewRowAction Members 8018
TDataSetAdapterNewRowAction methods 8024
TDataSetAdapterNewRowAction properties 8026
TDataSetAdapterNextRowAction class 8030
about TDataSetAdapterNextRowAction class 8030
DisplayLabel 8038
ExecuteAccess 8039
GetDefaultActionName 8037
ImplExecuteActionRequest 8037
ImplWebEnabled 8038
OnAfterExecute 8040
OnAfterGetResponse 8040
OnBeforeExecute 8040
OnBeforeGetResponse 8041
OnGetParams 8041
RedirectOptions 8039
TDataSetAdapterNextRowAction events 8039
TDataSetAdapterNextRowAction Members 8031

TDataSetAdapterNextRowAction methods 8037
TDataSetAdapterNextRowAction properties 8038
TDataSetAdapterPrevRowAction class 8042
about TDataSetAdapterPrevRowAction class 8042
DisplayLabel 8050
ExecuteAccess 8051
GetDefaultActionName 8049
ImplExecuteActionRequest 8049
ImplWebEnabled 8050
OnAfterExecute 8052
OnAfterGetResponse 8052
OnBeforeExecute 8052
OnBeforeGetResponse 8053
OnGetParams 8053
RedirectOptions 8051
TDataSetAdapterPrevRowAction events 8051
TDataSetAdapterPrevRowAction Members 8043
TDataSetAdapterPrevRowAction methods 8049
TDataSetAdapterPrevRowAction properties 8050
TDataSetAdapterRefreshRowAction class 8054
about TDataSetAdapterRefreshRowAction class 8054
DisplayLabel 8063
ExecuteAccess 8063
GetDefaultActionName 8061
ImplExecuteActionRequest 8061
ImplIsDefaultAction 8062
ImplWebEnabled 8062
OnAfterExecute 8064
OnAfterGetResponse 8064
OnBeforeExecute 8065
OnBeforeGetResponse 8065
OnGetParams 8066
RedirectOptions 8063
TDataSetAdapterRefreshRowAction events 8064
TDataSetAdapterRefreshRowAction Members 8055
TDataSetAdapterRefreshRowAction methods 8061
TDataSetAdapterRefreshRowAction properties 8062
TDataSetAdapterRowAction class 8066
about TDataSetAdapterRowAction class 8066
ActionModes 8075
DisplayLabel 8076
ExecuteAccess 8076
ImplExecuteActionRequest 8074
ImplGetAdapterItemRequestParams 8074
OnAfterExecute 8077
OnAfterGetResponse 8077
OnBeforeExecute 8078
OnBeforeGetResponse 8078
OnGetParams 8078
RedirectOptions 8076
TDataSetAdapterRowAction events 8077
TDataSetAdapterRowAction Members 8068
TDataSetAdapterRowAction methods 8073
TDataSetAdapterRowAction properties 8075
TDataSetButton class 16576
about TDataSetButton class 16576
AddElements 16582
Caption 16584
Custom 16584
DefaultCaption 16581
GetCaption 16582
GetMethodName 16582
GetSubComponents 16582
ImplAddElements 16583
ImplContent 16583
Style 16584
StyleRule 16585
TDataSetButton fields 16581
TDataSetButton Members 16577
TDataSetButton methods 16581
TDataSetButton properties 16583
XMLComponent 16585
XMLMethodName 16581
XMLUseParent 16586
TDataSetCancel class 7657
about TDataSetCancel class 7657
DataSource 7663
ExecuteTarget 7662

TDataSetCancel Members 7658
TDataSetCancel methods 7662
TDataSetCancel properties 7663
UpdateTarget 7663
TDataSetDelete class 7664
about TDataSetDelete class 7664
DataSource 7670
ExecuteTarget 7669
TDataSetDelete Members 7665
TDataSetDelete methods 7669
TDataSetDelete properties 7670
UpdateTarget 7670
TDataSetDesigner class 7056
about TDataSetDesigner class 7056
BeginDesign 7059
Create 7058
DataEvent 7059
DataSet 7060
Destroy 7058
EndDesign 7059
TDataSetDesigner Members 7057
TDataSetDesigner methods 7059
TDataSetDesigner properties 7060
TDataSetEdit class 7671
about TDataSetEdit class 7671
DataSource 7677
ExecuteTarget 7676
TDataSetEdit Members 7671
TDataSetEdit methods 7676
TDataSetEdit properties 7677
UpdateTarget 7676
TDataSetField class 7060
about TDataSetField class 7060
Assign 7068
Bind 7069
Create 7067
Destroy 7067
GetCanModify 7069
GetFields 7069
IncludeObjectField 7069
NestedDataSet 7070
TDataSetField Members 7061
TDataSetField methods 7068
TDataSetField properties 7069
TDataSetFirst class 7677
about TDataSetFirst class 7677
DataSource 7684
ExecuteTarget 7683
TDataSetFirst Members 7678
TDataSetFirst methods 7682
TDataSetFirst properties 7684
UpdateTarget 7683
TDataSetInsert class 7684
about TDataSetInsert class 7684
DataSource 7690
ExecuteTarget 7689
TDataSetInsert Members 7685
TDataSetInsert methods 7689
TDataSetInsert properties 7690
UpdateTarget 7690
TDataSetLast class 7691
about TDataSetLast class 7691
DataSource 7697
ExecuteTarget 7696
TDataSetLast Members 7692
TDataSetLast methods 7696
TDataSetLast properties 7697
UpdateTarget 7697
TDataSetNext class 7698
about TDataSetNext class 7698
DataSource 7704
ExecuteTarget 7703
TDataSetNext Members 7698
TDataSetNext methods 7703
TDataSetNext properties 7704
UpdateTarget 7703
TDataSetPageProducer class 11807
about TDataSetPageProducer class 11807

DataSet 11811
HTMLDoc 11812
HTMLFile 11812
OnHTMLTag 11814
ScriptEngine 11813
TDataSetPageProducer events 11814
TDataSetPageProducer Members 11808
TDataSetPageProducer properties 11811
TDataSetPost class 7704
about TDataSetPost class 7704
DataSource 7711
ExecuteTarget 7710
TDataSetPost Members 7705
TDataSetPost methods 7709
TDataSetPost properties 7710
UpdateTarget 7710
TDataSetPrior class 7711
about TDataSetPrior class 7711
DataSource 7717
ExecuteTarget 7716
TDataSetPrior Members 7712
TDataSetPrior methods 7716
TDataSetPrior properties 7717
UpdateTarget 7717
TDataSetProvider class 18508
about TDataSetProvider class 18508
ActiveUpdateException 18528
AfterApplyUpdates 18533
AfterExecute 18534
AfterGetParams 18535
AfterGetRecords 18535
AfterRowRequest 18536
AfterUpdateRecord 18536
BeforeApplyUpdates 18537
BeforeExecute 18538
BeforeGetParams 18538
BeforeGetRecords 18539
BeforeRowRequest 18540
BeforeUpdateRecord 18541
CheckDataSet 18518
Constraints 18528
Create 18514
CreateDataPacket 18518
CreateResolver 18519
DataSet 18529
Destroy 18514
DoBeforeExecute 18519
DoBeforeGetRecords 18520
DoGetProviderAttributes 18521
DoGetTableName 18521
Exported 18530
FConstraints 18515
FDataSet 18515
FDataSetOpened 18516
FDSWriter 18516
FetchDetails 18521
FGetDSProps 18516
FGetTableName 18516
FindRecord 18522
FParams 18516
FRecordsSent 18517
FResolveToDataSet 18517
FTransactionStarted 18517
GetDataSetFromDelta 18522
InternalApplyUpdates 18522
InternalExecute 18523
InternalGetParams 18524
InternalGetRecords 18524
InternalRowRequest 18525
LocateRecord 18525
Notification 18526
OnDataRequest 18541
OnGetData 18541
OnGetDataSetProperties 18542
OnGetTableName 18542
OnUpdateData 18543
OnUpdateError 18543
OnValidate 18544

Options 18530
OwnerData 18531
Params 18531
Reset 18526
ResolveToDataSet 18531
SetCommandText 18526
SetDataSet 18527
SetParams 18527
SetResolveToDataSet 18527
TDataSetProvider events 18533
TDataSetProvider fields 18515
TDataSetProvider Members 18509
TDataSetProvider methods 18517
TDataSetProvider properties 18528
TransactionStarted 18532
UpdateMode 18532
UpdateRecord 18527
TDataSetRefresh class 7718
about TDataSetRefresh class 7718
DataSource 7724
ExecuteTarget 7723
TDataSetRefresh Members 7719
TDataSetRefresh methods 7723
TDataSetRefresh properties 7724
UpdateTarget 7723
TDataSetResolver class 18544
about TDataSetResolver class 18544
BeginUpdate 18549
Create 18548
DoDelete 18550
DoInsert 18550
DoUpdate 18551
EndUpdate 18551
FBookmark 18548
FOpened 18549
GetProvider 18551
InitializeConflictBuffer 18552
InternalBeforeResolve 18552
Provider 18553
PutRecord 18552
TDataSetResolver fields 18548
TDataSetResolver Members 18544
TDataSetResolver methods 18549
TDataSetResolver properties 18553
TDataSetTableProducer class 10277
about TDataSetTableProducer class 10277
Caption 10284
CaptionAlignment 10284
Columns 10285
Content 10282
DataSet 10285
Footer 10286
GetDataSet 10283
Header 10286
MaxRows 10287
Notification 10283
OnCreateContent 10289
OnFormatCell 10289
OnGetTableCaption 10290
RowAttributes 10287
SetDataSet 10283
TableAttributes 10288
TDataSetTableProducer events 10289
TDataSetTableProducer Members 10278
TDataSetTableProducer methods 10282
TDataSetTableProducer properties 10284
TDataSetUpdateObject class 9994
about TDataSetUpdateObject class 9994
Apply 9998
DataSet 9999
GetDataSet 9998
SetDataSet 9999
TDataSetUpdateObject Members 9995
TDataSetUpdateObject methods 9998
TDataSetUpdateObject properties 9999
TDataSetValuesList class 8079
about TDataSetValuesList class 8079
Actions 8088

Data 8088
DataSet 8088
NameField 8089
OnPrepareDataSet 8090
TDataSetValuesList events 8090
TDataSetValuesList Members 8081
TDataSetValuesList properties 8088
ValueField 8089
TDataSource class 7070
about TDataSource class 7070
AutoEdit 7077
Create 7075
DataLinks 7078
DataSet 7078
Destroy 7075
Edit 7076
Enabled 7079
IsLinkedTo 7076
OnDataChange 7080
OnStateChange 7082
OnUpdateData 7082
State 7080
TDataSource events 7080
TDataSource Members 7072
TDataSource methods 7076
TDataSource properties 7077
TDataSourceLink class 8577
about TDataSourceLink class 8577
ActiveChanged 8580
Create 8580
FocusControl 8581
LayoutChanged 8581
RecordChanged 8581
TDataSourceLink Members 8578
TDataSourceLink methods 8580
TDateField class 7083
about TDateField class 7083
Create 7089
GetDataSize 7090
TDateField Members 7083
TDateField methods 7090
TDateProperty class 10957
about TDateProperty class 10957
GetAttributes 10960
GetValue 10961
SetValue 10962
TDateProperty Members 10958
TDateProperty methods 10960
TDateTime class 22256
- 22267
-- 22268
- operator 22267
-- operator 22268
!= 22268
+ 22269
+ operator 22268
++ 22269
++ operator 22269
+= 22270
+= operator 22270
< 22270
<= 22271
= 22271
-= 22272
= operator 22271
-= operator 22272
== 22273
> 22273
>= 22273
about TDateTime class 22256
AnsiString 22273
CurrentDate 22261
CurrentDateTime 22261
CurrentTime 22262
DateString 22262
DateTimeString 22262
DayOfWeek 22263
DecodeDate 22263

tttttt

DecodeTime 22263
double 22274
FileDate 22264
FileDateToDateTime 22264
FormatString 22265
int 22274
TDateTime 22259
TDateTime constructor 22258
TDateTime enumerations 22260
TDateTime Members 22257
TDateTime methods 22260
TDateTime operators 22266
TimeString 22266
TDateTimeBase class 22275
about TDateTimeBase class 22275
TDateTimeBase data members 22275
TDateTimeBase Members 22275
Val 22275
TDateTimeField class 7090
about TDateTimeField class 7090
CopyData 7098
Create 7097
DisplayFormat 7101
EditMask 7102
GetAsDateTime 7098
GetAsFloat 7099
GetAsString 7099
GetAsVariant 7099
GetDataSize 7099
GetDefaultWidth 7100
GetText 7100
SetAsDateTime 7100
SetAsFloat 7100
SetAsString 7100
SetVarValue 7101
TDateTimeField Members 7091
TDateTimeField methods 7098
TDateTimeField properties 7101
Value 7103
TDateTimePicker class 4049
about TDateTimePicker class 4049
Align 4066
Anchors 4069
BevelEdges 4070
BevelInner 4070
BevelKind 4071
BevelOuter 4071
BevelWidth 4072
BiDiMode 4072
CalAlignment 4073
CalColors 4073
Change 4061
Checked 4073
CheckEmptyDate 4061
Color 4074
Constraints 4074
Create 4060
CreateParams 4062
CreateWnd 4062
Date 4075
DateFormat 4076
DateMode 4076
DateTime 4076
DragCursor 4077
DragKind 4077
DragMode 4077
DroppedDown 4079
Enabled 4079
Font 4080
Format 4081
GetCalendarHandle 4063
ImeMode 4082
ImeName 4082
Kind 4083
MaxDate 4084
MinDate 4084
MsgSetCalColors 4063
MsgSetDateTime 4064

MsgSetRange 4064
OnChange 4096
OnClick 4097
OnCloseUp 4099
OnContextPopup 4099
OnDblClick 4100
OnDragDrop 4100
OnDragOver 4102
OnDropDown 4105
OnEndDock 4105
OnEndDrag 4105
OnEnter 4106
OnExit 4109
OnKeyDown 4110
OnKeyPress 4113
OnKeyUp 4114
OnMouseEnter 4116
OnMouseLeave 4116
OnStartDock 4116
OnStartDrag 4117
OnUserInput 4118
ParentBiDiMode 4084
ParentColor 4085
ParentFont 4085
ParentShowHint 4086
ParseInput 4086
PopupMenu 4086
ShowCheckbox 4091
ShowHint 4091
TabOrder 4093
TabStop 4093
TDateTimePicker events 4096
TDateTimePicker Members 4049
TDateTimePicker methods 4061
TDateTimePicker properties 4065
Time 4094
Visible 4094
TDateTimeProperty class 10962
about TDateTimeProperty class 10962
GetAttributes 10965
GetValue 10966
SetValue 10966
TDateTimeProperty Members 10962
TDateTimeProperty methods 10965
TDBCheckBox class 8582
about TDBCheckBox class 8582
Action 8598
Align 8599
Alignment 8602
AllowGrayed 8602
Anchors 8603
BiDiMode 8604
Caption 8604
Checked 8607
Color 8608
Constraints 8609
Create 8593
Ctl3D 8609
DataField 8610
DataSource 8610
Destroy 8594
DragCursor 8611
DragKind 8611
DragMode 8611
Enabled 8613
ExecuteAction 8594
Field 8613
Font 8614
KeyPress 8595
Notification 8595
OnClick 8629
OnContextPopup 8631
OnDragDrop 8631
OnDragOver 8634
OnEndDock 8636
OnEndDrag 8637
OnEnter 8638
OnExit 8640

OnKeyDown 8642
OnKeyPress 8644
OnKeyUp 8645
OnMouseActivate 8647
OnMouseDown 8648
OnMouseEnter 8650
OnMouseLeave 8651
OnMouseMove 8651
OnMouseUp 8653
OnStartDock 8655
OnStartDrag 8656
ParentBiDiMode 8615
ParentColor 8615
ParentCtl3D 8616
ParentFont 8616
ParentShowHint 8616
PopupMenu 8617
ReadOnly 8622
ShowHint 8622
State 8623
TabOrder 8624
TabStop 8625
TDBCheckBox events 8628
TDBCheckBox Members 8583
TDBCheckBox methods 8594
TDBCheckBox properties 8597
Toggle 8596
UpdateAction 8596
UseRightToLeftAlignment 8596
ValueChecked 8625
ValueUnchecked 8626
Visible 8626
WndProc 8597
WordWrap 8628
TDBComboBox class 8657
about TDBComboBox class 8657
Align 8678
Anchors 8682
AutoComplete 8682
AutoDropDown 8683
BevelEdges 8683
BevelInner 8683
BevelKind 8684
BevelOuter 8684
BevelWidth 8685
BiDiMode 8685
Change 8671
CharCase 8686
Click 8671
Color 8686
ComboWndProc 8671
Constraints 8687
Create 8669
CreateWnd 8672
Ctl3D 8688
DataField 8688
DataSource 8689
Destroy 8670
DragCursor 8689
DragKind 8689
DragMode 8690
DropDown 8672
DropDownCount 8691
Enabled 8692
ExecuteAction 8673
Field 8692
Font 8693
ImeMode 8694
ImeName 8694
ItemHeight 8695
Items 8695
KeyDown 8673
KeyPress 8674
Loaded 8674
Notification 8675
OnChange 8710
OnClick 8712
OnContextPopup 8714

wwww

OnDblClick 8715
OnDragDrop 8715
OnDragOver 8717
OnDrawItem 8720
OnDropDown 8720
OnEndDock 8722
OnEndDrag 8722
OnEnter 8723
OnExit 8726
OnKeyDown 8727
OnKeyPress 8730
OnKeyUp 8731
OnMeasureItem 8733
OnMouseEnter 8733
OnMouseLeave 8733
OnStartDock 8734
OnStartDrag 8734
ParentBiDiMode 8696
ParentColor 8696
ParentCtl3D 8697
ParentFont 8697
ParentShowHint 8698
PopupMenu 8698
ReadOnly 8703
SetItems 8675
SetStyle 8676
ShowHint 8703
Sorted 8705
Style 8706
TabOrder 8707
TabStop 8707
TDBComboBox events 8710
TDBComboBox Members 8657
TDBComboBox methods 8670
TDBComboBox properties 8677
Text 8708
UpdateAction 8676
UseRightToLeftAlignment 8676
Visible 8708
WndProc 8677
TDBCtrlGrid class 8168
about TDBCtrlGrid class 8168
Align 8188
AllowDelete 8191
AllowInsert 8192
Anchors 8192
Canvas 8193
ColCount 8193
Color 8193
Constraints 8194
Create 8179
CreateParams 8180
CreateWnd 8181
DataSource 8195
Destroy 8179
DoKey 8181
DragCursor 8195
DragMode 8195
EditMode 8197
Enabled 8197
ExecuteAction 8182
Font 8198
GetChildParent 8183
GetChildren 8183
GetTabOrderList 8183
KeyDown 8184
OnClick 8213
OnDblClick 8215
OnDragDrop 8215
OnDragOver 8218
OnEndDrag 8220
OnEnter 8221
OnExit 8224
OnKeyDown 8225
OnKeyPress 8228
OnKeyUp 8229
OnMouseActivate 8231
OnMouseDown 8232

OnMouseEnter 8234
OnMouseLeave 8234
OnMouseMove 8234
OnMouseUp 8237
OnPaintPanel 8239
OnStartDrag 8240
Orientation 8199
PaintPanel 8185
PaintWindow 8185
Panel 8199
PanelBorder 8200
PanelCount 8200
PanelHeight 8200
PanelIndex 8201
PanelWidth 8201
ParentColor 8201
ParentFont 8202
ParentShowHint 8202
PopupMenu 8203
ReadState 8185
RowCount 8207
SelectedColor 8208
SetBounds 8186
ShowFocus 8208
ShowHint 8209
TabOrder 8210
TabStop 8211
TDBCtrlGrid events 8213
TDBCtrlGrid Members 8169
TDBCtrlGrid methods 8180
TDBCtrlGrid properties 8187
UpdateAction 8186
Visible 8211
TDBCtrlGridLink class 8240
about TDBCtrlGridLink class 8240
ActiveChanged 8243
Create 8243
DataSetChanged 8243
TDBCtrlGridLink Members 8241
TDBCtrlGridLink methods 8243
TDBCtrlPanel class 8244
about TDBCtrlPanel class 8244
CreateLinked 8255
CreateParams 8255
PaintWindow 8256
TDBCtrlPanel Members 8245
TDBCtrlPanel methods 8255
TDBDataSet class 10000
about TDBDataSet class 10000
AutoRefresh 10021
CheckOpen 10014
CloseCursor 10014
CloseDatabase 10014
ConstraintsStored 10015
Create 10013
Database 10021
DatabaseName 10022
DBFlags 10023
DBHandle 10023
DBLocale 10024
DBSession 10024
Disconnect 10015
Filter 10025
Filtered 10026
FilterOptions 10026
Handle 10027
OnFilterRecord 10028
OpenCursor 10015
OpenDatabase 10016
PSEndTransaction 10016
PSExecuteStatement 10017
PSGetAttributes 10017
PSGetQuoteChar 10018
PSInTransaction 10018
PSIsSQLBased 10018
PSStartTransaction 10019
PSUpdateRecord 10019
SessionName 10027

SetDBFlag 10020
SetHandle 10020
TDBDataSet events 10028
TDBDataSet Members 10000
TDBDataSet methods 10013
TDBDataSet properties 10020
UpdateMode 10028
TDBDataSetSprig class 1736
about TDBDataSetSprig class 1736
AnyProblems 1736
DragDropTo 1737
DragOverTo 1737
FigureParent 1737
PaletteOverTo 1737
Reparent 1738
TDBDataSetSprig Members 1736
TDBDataSetSprig methods 1736
TDBEdit class 8735
about TDBEdit class 8735
Align 8754
Anchors 8758
AutoSelect 8758
AutoSize 8759
BevelEdges 8760
BevelInner 8760
BevelKind 8761
BevelOuter 8761
BevelWidth 8761
BiDiMode 8762
BorderStyle 8762
Change 8749
CharCase 8763
Color 8763
Constraints 8764
Create 8747
Ctl3D 8764
DataField 8765
DataSource 8765
Destroy 8748
DragCursor 8766
DragKind 8766
DragMode 8767
EditCanModify 8749
Enabled 8768
ExecuteAction 8750
Field 8769
Font 8769
ImeMode 8770
ImeName 8771
KeyDown 8750
KeyPress 8751
Loaded 8751
MaxLength 8771
Notification 8752
OnChange 8784
OnClick 8784
OnContextPopup 8786
OnDblClick 8787
OnDragDrop 8787
OnDragOver 8790
OnEndDock 8792
OnEndDrag 8792
OnEnter 8793
OnExit 8796
OnKeyDown 8797
OnKeyPress 8800
OnKeyUp 8801
OnMouseActivate 8803
OnMouseDown 8804
OnMouseEnter 8806
OnMouseLeave 8807
OnMouseMove 8807
OnMouseUp 8809
OnStartDock 8811
OnStartDrag 8812
ParentBiDiMode 8772
ParentColor 8772
ParentCtl3D 8773

ParentFont 8773
ParentShowHint 8774
PasswordChar 8774
PopupMenu 8774
ReadOnly 8779
Reset 8752
ShowHint 8779
TabOrder 8781
TabStop 8782
TDBEdit events 8783
TDBEdit Members 8736
TDBEdit methods 8748
TDBEdit properties 8753
UpdateAction 8752
UseRightToLeftAlignment 8753
Visible 8782
TDbEngineErrorDlg class 9546
about TDbEngineErrorDlg class 9546
BackBtn 9564
BackClick 9568
BasicPanel 9564
BDELabel 9564
ButtonPanel 9564
DbCatSub 9564
DbException 9571
DbMessageText 9565
DbResult 9565
DetailsBtn 9565
DetailsBtnClick 9568
DetailsPanel 9565
ErrorText 9566
FormCreate 9568
FormDestroy 9569
FormShow 9569
HookExceptions 9569
IconImage 9566
IconPanel 9566
NativeLabel 9566
NativeResult 9566
NextBtn 9567
NextClick 9570
OKBtn 9567
RightPanel 9567
ShowException 9570
TDbEngineErrorDlg fields 9563
TDbEngineErrorDlg Members 9546
TDbEngineErrorDlg methods 9567
TDbEngineErrorDlg properties 9571
TopPanel 9567
TDBError class 10029
about TDBError class 10029
Category 10031
Create 10030
ErrorCode 10032
Message 10032
NativeError 10032
SubCode 10033
TDBError Members 10030
TDBError properties 10031
TDBGrid class 9668
about TDBGrid class 9668
Align 9686
Anchors 9689
BiDiMode 9690
BorderStyle 9690
Canvas 9690
Color 9692
Columns 9693
Constraints 9694
Ctl3D 9694
DataSource 9695
DefaultDrawing 9696
DragCursor 9696
DragKind 9697
DragMode 9697
Enabled 9699
FixedColor 9699
Font 9699

ImeMode 9700
ImeName 9701
OnCellClick 9715
OnColEnter 9716
OnColExit 9716
OnColumnMoved 9717
OnDblClick 9717
OnDragDrop 9717
OnDragOver 9720
OnDrawColumnCell 9722
OnDrawDataCell 9723
OnEditButtonClick 9723
OnEndDock 9724
OnEndDrag 9724
OnEnter 9725
OnExit 9728
OnKeyDown 9729
OnKeyPress 9732
OnKeyUp 9733
OnMouseActivate 9735
OnMouseDown 9736
OnMouseEnter 9738
OnMouseLeave 9738
OnMouseMove 9738
OnMouseUp 9741
OnMouseWheel 9743
OnMouseWheelDown 9743
OnMouseWheelUp 9744
OnStartDock 9744
OnStartDrag 9745
OnTitleClick 9745
Options 9702
ParentBiDiMode 9702
ParentColor 9702
ParentCtl3D 9703
ParentFont 9703
ParentShowHint 9704
PopupMenu 9704
ReadOnly 9709
SelectedRows 9709
ShowHint 9710
TabOrder 9712
TabStop 9712
TDBGrid events 9714
TDBGrid Members 9670
TDBGrid properties 9685
TitleFont 9713
Visible 9713
TDBGridColumns class 9746
about TDBGridColumns class 9746
Add 9749
Create 9748
GetOwner 9749
Grid 9752
Items 9753
LoadFromFile 9750
LoadFromStream 9750
RebuildColumns 9750
RestoreDefaults 9751
SaveToFile 9751
SaveToStream 9751
State 9753
TDBGridColumns Members 9746
TDBGridColumns methods 9749
TDBGridColumns properties 9752
Update 9752
TDBImage class 8813
about TDBImage class 8813
Align 8831
Anchors 8835
AutoDisplay 8835
BorderStyle 8836
Center 8836
Color 8836
Constraints 8837
CopyToClipboard 8825
Create 8824
CreateParams 8825

Ctl3D 8838
CutToClipboard 8826
DataField 8838
DataSource 8839
Destroy 8824
DragCursor 8839
DragKind 8839
DragMode 8840
Enabled 8841
ExecuteAction 8826
Field 8842
Font 8842
GetPalette 8827
KeyDown 8827
KeyPress 8828
LoadPicture 8828
Notification 8829
OnClick 8856
OnContextPopup 8858
OnDblClick 8858
OnDragDrop 8859
OnDragOver 8861
OnEndDock 8863
OnEndDrag 8864
OnEnter 8865
OnExit 8867
OnKeyDown 8869
OnKeyPress 8872
OnKeyUp 8873
OnMouseActivate 8874
OnMouseDown 8875
OnMouseEnter 8878
OnMouseLeave 8878
OnMouseMove 8878
OnMouseUp 8880
OnStartDock 8883
OnStartDrag 8883
Paint 8829
ParentColor 8843
ParentCtl3D 8844
ParentFont 8844
ParentShowHint 8845
PasteFromClipboard 8830
Picture 8845
PopupMenu 8846
QuickDraw 8850
ReadOnly 8850
ShowHint 8851
Stretch 8852
TabOrder 8853
TabStop 8853
TDBImage events 8855
TDBImage Members 8813
TDBImage methods 8825
TDBImage properties 8830
UpdateAction 8830
Visible 8854
TDBListBox class 8884
about TDBListBox class 8884
Align 8902
Anchors 8905
AutoComplete 8906
BevelEdges 8906
BevelInner 8906
BevelKind 8907
BevelOuter 8907
BevelWidth 8908
BiDiMode 8908
BorderStyle 8909
Click 8897
Color 8909
Constraints 8910
Create 8896
Ctl3D 8910
DataField 8911
DataSource 8911
Destroy 8897
DragCursor 8912

DragKind 8912
DragMode 8913
Enabled 8914
ExecuteAction 8898
Field 8915
Font 8915
ImeMode 8916
ImeName 8917
IntegralHeight 8917
ItemHeight 8918
Items 8918
KeyDown 8898
KeyPress 8899
Notification 8899
OnClick 8931
OnContextPopup 8933
OnDblClick 8934
OnDragDrop 8934
OnDragOver 8936
OnDrawItem 8939
OnEndDock 8941
OnEndDrag 8941
OnEnter 8942
OnExit 8945
OnKeyDown 8946
OnKeyPress 8949
OnKeyUp 8950
OnMeasureItem 8952
OnMouseActivate 8953
OnMouseDown 8954
OnMouseEnter 8957
OnMouseLeave 8957
OnMouseMove 8957
OnMouseUp 8959
OnStartDock 8962
OnStartDrag 8962
ParentBiDiMode 8919
ParentColor 8919
ParentCtl3D 8920
ParentFont 8920
ParentShowHint 8920
PopupMenu 8921
ReadOnly 8926
ShowHint 8926
Style 8927
TabOrder 8928
TabStop 8929
TDBListBox events 8931
TDBListBox Members 8885
TDBListBox methods 8897
TDBListBox properties 8901
UpdateAction 8900
UseRightToLeftAlignment 8900
Visible 8929
TDBLookupComboBox class 8963
about TDBLookupComboBox class 8963
Align 8984
Anchors 8987
BevelEdges 8988
BevelInner 8988
BevelKind 8989
BevelOuter 8989
BevelWidth 8990
BiDiMode 8990
CloseUp 8976
Color 8991
Constraints 8991
Create 8975
CreateParams 8976
Ctl3D 8992
DataField 8992
DataSource 8993
DragCursor 8993
DragKind 8994
DragMode 8994
DropDown 8977
DropDownAlign 8996
DropDownRows 8996

DropDownWidth 8996
Enabled 8997
ExecuteAction 8977
Font 8997
ImeMode 8998
ImeName 8999
KeyDown 8977
KeyField 9000
KeyPress 8978
KeyValue 9000
KeyValueChanged 8978
ListField 9001
ListFieldIndex 9001
ListLinkDataChanged 8979
ListSource 9002
ListVisible 9002
MouseDown 8979
MouseMove 8980
MouseUp 8980
NullValueKey 9003
OnClick 9015
OnCloseUp 9017
OnContextPopup 9017
OnDragDrop 9018
OnDragOver 9020
OnDropDown 9023
OnEndDock 9023
OnEndDrag 9024
OnEnter 9025
OnExit 9027
OnKeyDown 9029
OnKeyPress 9031
OnKeyUp 9033
OnMouseActivate 9034
OnMouseDown 9035
OnMouseEnter 9038
OnMouseLeave 9038
OnMouseMove 9038
OnMouseUp 9040
OnStartDock 9043
OnStartDrag 9043
Paint 8981
ParentBiDiMode 9003
ParentColor 9003
ParentCtl3D 9004
ParentFont 9004
ParentShowHint 9005
PopupMenu 9005
ReadOnly 9010
SetBounds 8981
ShowHint 9010
TabOrder 9012
TabStop 9012
TDBLookupComboBox events 9014
TDBLookupComboBox Members 8964
TDBLookupComboBox methods 8975
TDBLookupComboBox properties 8983
Text 9013
UpdateAction 8982
UpdateListFields 8982
UseRightToLeftAlignment 8982
Visible 9013
TDBLookupControl class 9044
about TDBLookupControl class 9044
CanModify 9057
Create 9055
DataField 9062
DataLink 9063
DataSource 9063
Destroy 9056
ExecuteAction 9057
Field 9064
GetBorderSize 9058
GetTextHeight 9058
HasFocus 9064
KeyField 9064
KeyValue 9065
KeyValueChanged 9058

eeee

ListActive 9065
ListField 9066
ListFieldIndex 9067
ListFields 9067
ListLink 9067
ListLinkDataChanged 9058
ListSource 9068
LocateKey 9059
Notification 9059
NullValueKey 9068
ParentColor 9069
ProcessSearchKey 9060
ReadOnly 9069
SearchText 9069
SelectKeyValue 9060
TabStop 9070
TDBLookupControl Members 9045
TDBLookupControl methods 9056
TDBLookupControl properties 9062
UpdateAction 9060
UpdateDataFields 9061
UpdateListFields 9061
TDBLookupListBox class 9070
about TDBLookupListBox class 9070
Align 9091
Anchors 9094
BevelEdges 9095
BevelInner 9095
BevelKind 9095
BevelOuter 9096
BevelWidth 9096
BiDiMode 9097
BorderStyle 9097
Color 9098
Constraints 9098
Create 9082
CreateParams 9082
CreateWnd 9083
Ctl3D 9099
DataField 9099
DataSource 9100
DragCursor 9100
DragKind 9101
DragMode 9101
Enabled 9103
ExecuteAction 9083
Font 9103
ImeMode 9104
ImeName 9105
KeyDown 9084
KeyField 9105
KeyPress 9085
KeyValue 9106
KeyValueChanged 9085
ListField 9106
ListFieldIndex 9107
ListLinkDataChanged 9085
ListSource 9108
MouseDown 9086
MouseMove 9086
MouseUp 9087
NullValueKey 9108
OnClick 9121
OnContextPopup 9123
OnDblClick 9124
OnDragDrop 9124
OnDragOver 9126
OnEndDock 9129
OnEndDrag 9129
OnEnter 9130
OnExit 9132
OnKeyDown 9134
OnKeyPress 9137
OnKeyUp 9138
OnMouseActivate 9140
OnMouseDown 9141
OnMouseEnter 9143
OnMouseLeave 9143

OnMouseMove 9143
OnMouseUp 9146
OnStartDock 9148
OnStartDrag 9148
Paint 9087
ParentBiDiMode 9109
ParentColor 9109
ParentCtl3D 9109
ParentFont 9110
ParentShowHint 9110
PopupMenu 9111
ReadOnly 9115
RowCount 9116
SelectedItem 9116
SetBounds 9088
ShowHint 9116
TabOrder 9118
TabStop 9118
TDBLookupListBox events 9120
TDBLookupListBox Members 9071
TDBLookupListBox methods 9082
TDBLookupListBox properties 9090
UpdateAction 9088
UpdateListFields 9089
UseRightToLeftAlignment 9089
Visible 9119
TDBMemo class 9149
about TDBMemo class 9149
Align 9168
Alignment 9171
Anchors 9172
AutoDisplay 9172
BevelEdges 9173
BevelInner 9173
BevelKind 9174
BevelOuter 9174
BevelWidth 9175
BiDiMode 9175
BorderStyle 9175
Change 9162
Color 9176
Constraints 9177
Create 9161
Ctl3D 9177
DataField 9178
DataSource 9178
Destroy 9162
DragCursor 9178
DragKind 9179
DragMode 9179
Enabled 9181
ExecuteAction 9163
Field 9181
Font 9182
HideSelection 9183
ImeMode 9183
ImeName 9184
KeyDown 9163
KeyPress 9164
Loaded 9164
LoadMemo 9164
MaxLength 9184
Notification 9165
OnChange 9202
OnClick 9202
OnContextPopup 9204
OnDblClick 9205
OnDragDrop 9205
OnDragOver 9207
OnEndDock 9210
OnEndDrag 9210
OnEnter 9211
OnExit 9214
OnKeyDown 9215
OnKeyPress 9218
OnKeyUp 9219
OnMouseActivate 9221
OnMouseDown 9222

OnMouseEnter 9224
OnMouseLeave 9224
OnMouseMove 9225
OnMouseUp 9227
OnStartDock 9229
OnStartDrag 9230
ParentBiDiMode 9185
ParentColor 9185
ParentCtl3D 9186
ParentFont 9186
ParentShowHint 9187
PopupMenu 9187
ReadOnly 9192
ScrollBars 9192
ShowHint 9194
TabOrder 9196
TabStop 9196
TDBMemo events 9201
TDBMemo Members 9150
TDBMemo methods 9162
TDBMemo properties 9167
UpdateAction 9165
UseRightToLeftAlignment 9166
Visible 9197
WantReturns 9198
WantTabs 9199
WndProc 9166
WordWrap 9199
TDBNavigator class 9230
about TDBNavigator class 9230
ActiveChanged 9245
Align 9249
Anchors 9253
BeforeAction 9269
BtnClick 9245
Buttons 9244
CalcMinSize 9245
ConfirmDelete 9253
Constraints 9254
Create 9243
Ctl3D 9254
DataChanged 9246
DataSource 9255
Destroy 9244
DragCursor 9255
DragKind 9256
DragMode 9256
EditingChanged 9246
Enabled 9257
Flat 9258
GetChildren 9246
Hints 9258
KeyDown 9247
Loaded 9247
Notification 9248
OnClick 9269
OnContextPopup 9270
OnDblClick 9271
OnDragDrop 9271
OnDragOver 9273
OnEndDock 9276
OnEndDrag 9276
OnEnter 9277
OnExit 9279
OnResize 9281
OnStartDock 9282
OnStartDrag 9282
ParentCtl3D 9259
ParentShowHint 9259
PopupMenu 9260
SetBounds 9248
ShowHint 9264
TabOrder 9266
TabStop 9266
TDBNavigator events 9269
TDBNavigator fields 9244
TDBNavigator Members 9233
TDBNavigator methods 9244

TDBNavigator properties 9249
Visible 9267
VisibleButtons 9268
TDBOleControl class 9810
about TDBOleControl class 9810
Create 9823
DataBindings 9825
DataSource 9825
Destroy 9823
Notification 9824
OnChanged 9824
OnRequestEdit 9825
TDBOleControl Members 9810
TDBOleControl methods 9824
TDBOleControl properties 9825
TDBRadioGroup class 9283
about TDBRadioGroup class 9283
Align 9300
Anchors 9303
BiDiMode 9304
CanModify 9295
Caption 9304
Change 9296
Click 9296
Color 9307
Columns 9308
Constraints 9308
Create 9294
Ctl3D 9308
DataField 9309
DataLink 9309
DataSource 9309
Destroy 9295
DragCursor 9310
DragKind 9310
DragMode 9311
Enabled 9312
ExecuteAction 9296
Field 9313
Font 9313
ItemIndex 9314
Items 9315
KeyPress 9297
Notification 9297
OnChange 9328
OnClick 9328
OnContextPopup 9330
OnDragDrop 9331
OnDragOver 9333
OnEndDock 9335
OnEndDrag 9336
OnEnter 9337
OnExit 9339
OnMouseEnter 9341
OnMouseLeave 9341
OnStartDock 9341
OnStartDrag 9342
ParentBackground 9315
ParentBiDiMode 9316
ParentColor 9316
ParentCtl3D 9317
ParentFont 9317
ParentShowHint 9317
PopupMenu 9318
ReadOnly 9323
ShowHint 9323
TabOrder 9324
TabStop 9325
TDBRadioGroup events 9328
TDBRadioGroup Members 9284
TDBRadioGroup methods 9295
TDBRadioGroup properties 9299
UpdateAction 9298
UseRightToLeftAlignment 9298
Value 9325
Values 9326
Visible 9326
TDBRichEdit class 9343

iiii

about TDBRichEdit class 9343
Align 9362
Alignment 9366
Anchors 9366
AutoDisplay 9367
BevelEdges 9368
BevelInner 9368
BevelKind 9368
BevelOuter 9369
BevelWidth 9369
BiDiMode 9370
BorderStyle 9370
Change 9357
Color 9370
Constraints 9371
Create 9356
Ctl3D 9372
DataField 9372
DataSource 9373
Destroy 9356
DragCursor 9373
DragKind 9374
DragMode 9374
Enabled 9375
ExecuteAction 9358
Field 9376
Font 9376
HideScrollBars 9377
HideSelection 9379
ImeMode 9381
ImeName 9381
KeyDown 9358
KeyPress 9359
Loaded 9359
LoadMemo 9359
MaxLength 9382
Notification 9360
OnChange 9401
OnClick 9401
OnContextPopup 9403
OnDblClick 9404
OnDragDrop 9404
OnDragOver 9406
OnEndDock 9409
OnEndDrag 9409
OnEnter 9410
OnExit 9413
OnKeyDown 9414
OnKeyPress 9417
OnKeyUp 9418
OnMouseActivate 9420
OnMouseDown 9421
OnMouseEnter 9423
OnMouseLeave 9423
OnMouseMove 9424
OnMouseUp 9426
OnProtectChange 9428
OnResizeRequest 9429
OnSaveClipboard 9430
OnSelectionChange 9430
OnStartDock 9431
OnStartDrag 9432
ParentBiDiMode 9382
ParentColor 9383
ParentCtl3D 9383
ParentFont 9384
ParentShowHint 9384
PlainText 9385
PopupMenu 9386
ReadOnly 9390
ScrollBars 9391
ShowHint 9393
TabOrder 9395
TabStop 9395
TDBRichEdit events 9400
TDBRichEdit Members 9344
TDBRichEdit methods 9357
TDBRichEdit properties 9361

UpdateAction 9360
UseRightToLeftAlignment 9361
Visible 9396
WantReturns 9397
WantTabs 9398
WordWrap 9398
TDBText class 9433
about TDBText class 9433
Align 9445
Alignment 9449
Anchors 9449
AutoSize 9450
BiDiMode 9450
Color 9451
Constraints 9452
Create 9441
DataField 9452
DataSource 9453
Destroy 9441
DragCursor 9453
DragKind 9453
DragMode 9454
Enabled 9455
ExecuteAction 9442
Field 9456
Font 9456
GetLabelText 9442
Loaded 9443
Notification 9443
OnClick 9468
OnContextPopup 9470
OnDblClick 9471
OnDragDrop 9471
OnDragOver 9473
OnEndDock 9476
OnEndDrag 9476
OnMouseActivate 9477
OnMouseDown 9478
OnMouseEnter 9480
OnMouseLeave 9481
OnMouseMove 9481
OnMouseUp 9483
OnStartDock 9485
OnStartDrag 9486
ParentBiDiMode 9457
ParentColor 9458
ParentFont 9458
ParentShowHint 9459
PopupMenu 9459
SetAutoSize 9443
ShowHint 9464
TDBText events 9468
TDBText Members 9433
TDBText methods 9442
TDBText properties 9444
Transparent 9465
UpdateAction 9444
UseRightToLeftAlignment 9444
Visible 9466
WordWrap 9467
TDBXAnsiStringValue class 10356
about TDBXAnsiStringValue class 10356
Create 10358
Destroy 10358
GetAnsiString 10359
IsNull 10359
SetAnsiString 10359
TDBXAnsiStringValue Members 10357
TDBXAnsiStringValue methods 10358
TDBXBcdValue class 10359
about TDBXBcdValue class 10359
GetBcd 10361
IsNull 10361
SetBcd 10362
TDBXBcdValue Members 10360
TDBXBcdValue methods 10361
TDBXBooleanValue class 10362
about TDBXBooleanValue class 10362

GetBoolean 10364
IsNull 10364
SetBoolean 10364
TDBXBooleanValue Members 10362
TDBXBooleanValue methods 10363
TDBXByteArrayValue class 10364
about TDBXByteArrayValue class 10364
Destroy 10366
GetBytes 10366
GetValueSize 10367
IsNull 10367
SetDynamicBytes 10367
SetStaticBytes 10367
TDBXByteArrayValue Members 10365
TDBXByteArrayValue methods 10366
TDBXByteReader class 10368
about TDBXByteReader class 10368
Create 10369
FDBXContext 10369
GetAnsiString 10370
GetBcd 10370
GetByteLength 10370
GetBytes 10370
GetDate 10371
GetDouble 10371
GetInt16 10371
GetInt32 10371
GetInt64 10372
GetTime 10372
GetTimeStamp 10372
GetWideString 10372
TDBXByteReader fields 10369
TDBXByteReader Members 10368
TDBXByteReader methods 10369
TDBXByteReaderEx class 10373
about TDBXByteReaderEx class 10373
GetBoolean 10374
TDBXByteReaderEx Members 10373
TDBXByteReaderEx methods 10374
TDBXByteStreamReader class 10374
about TDBXByteStreamReader class 10374
Create 10375
Destroy 10375
Eos 10377
FBytes 10376
FCount 10376
FEOS 10376
FOffset 10376
FPosition 10376
Read 10377
Size 10377
TDBXByteStreamReader fields 10375
TDBXByteStreamReader Members 10374
TDBXByteStreamReader methods 10377
TDBXCatalogsColumns class 10688
about TDBXCatalogsColumns class 10688
CatalogName 10689
TDBXCatalogsColumns constants 10688
TDBXCatalogsColumns Members 10688
TDBXCatalogsIndex class 10689
about TDBXCatalogsIndex class 10689
CatalogName 10690
Last 10690
TDBXCatalogsIndex constants 10690
TDBXCatalogsIndex Members 10689
TDBXColumnConstraintsColumns class 10690
about TDBXColumnConstraintsColumns class 10690
CatalogName 10691
ColumnName 10691
ConstraintName 10692
SchemaName 10692
TableName 10692
TDBXColumnConstraintsColumns constants 10691
TDBXColumnConstraints Columns Members 10691
TDBXColumnConstraintsIndex class 10692
about TDBXColumnConstraintsIndex class 10692
CatalogName 10693
ColumnName 10693

ConstraintName 10694
Last 10694
SchemaName 10694
TableName 10694
TDBXColumnConstraintsIndex constants 10693
TDBXColumnConstraintsIndex Members 10693
TDBXColumnsColumns class 10695
about TDBXColumnsColumns class 10695
CatalogName 10697
ColumnName 10697
DbxDataType 10697
DefaultValue 10698
IsAutoIncrement 10698
IsFixedLength 10698
IsLong 10698
IsNullable 10698
IsUnicode 10699
IsUnsigned 10699
MaxInline 10699
Ordinal 10699
Precision 10700
Scale 10700
SchemaName 10700
TableName 10700
TDBXColumnsColumns constants 10696
TDBXColumnsColumns Members 10696
TypeName 10700
TDBXColumnsIndex class 10701
about TDBXColumnsIndex class 10701
CatalogName 10703
ColumnName 10703
DbxDataType 10703
DefaultValue 10704
IsAutoIncrement 10704
IsFixedLength 10704
IsLong 10704
IsNullable 10705
IsUnicode 10705
IsUnsigned 10705
Last 10705
MaxInline 10705
Ordinal 10706
Precision 10706
Scale 10706
SchemaName 10706
TableName 10707
TDBXColumnsIndex constants 10703
TDBXColumnsIndex Members 10702
TypeName 10707
TDBXCommand class 10378
about TDBXCommand class 10378
Close 10382
CommandType 10388
Create 10380
CreateParameter 10382
CreateParameterRow 10382
CreateParameters 10382
DerivedClose 10383
DerivedExecuteQuery 10383
DerivedExecuteUpdate 10383
DerivedGetNextReader 10383
DerivedOpen 10384
DerivedPrepare 10384
Destroy 10380
ExecuteQuery 10384
ExecuteUpdate 10384
FDBXContext 10381
FParameters 10381
GetCommandType 10385
GetErrorEvent 10385
GetNextReader 10385
GetParameters 10385
GetRowsAffected 10386
GetText 10386
IsPrepared 10388
MaxBlobSize 10389
NotImplemented 10386
OnErrorEvent 10391

Open 10386
Parameters 10389
Prepare 10386
RowsAffected 10389
RowSetSize 10390
SetCommandType 10387
SetMaxBlobSize 10387
SetRowSetSize 10387
SetText 10387
TDBXCommand events 10390
TDBXCommand fields 10380
TDBXCommand Members 10379
TDBXCommand methods 10381
TDBXCommand properties 10388
Text 10390
TDBXCommandEx class 10391
about TDBXCommandEx class 10391
CommandTimeout 10394
GetCommandTimeout 10393
LastReader 10394
SetCommandTimeout 10393
SetLastReader 10394
TDBXCommandEx Members 10391
TDBXCommandEx methods 10393
TDBXCommandEx properties 10394
TDBXCommandFactory class 10395
about TDBXCommandFactory class 10395
CreateCommand 10395
TDBXCommandFactory Members 10395
TDBXCommandFactory methods 10395
TDBXCommandTypes class 10396
about TDBXCommandTypes class 10396
DbxCommand 10397
DbxMetaData 10397
DbxPool 10397
DbxSQL 10398
DbxStoredProcedure 10398
DbxTable 10398
TDBXCommandTypes constants 10397
TDBXCommandTypes Members 10396
TDBXConnection class 10398
about TDBXConnection class 10398
BeginTransaction 10402
BeginTransaction method 10402
Close 10403
Commit 10403
CommitFreeAndNil 10404
ConnectionProperties 10410
Create 10400
CreateAndBeginTransaction 10404
CreateCommand 10404
DatabaseMetaData 10410
DerivedCreateCommand 10405
DerivedGetCommands 10405
DerivedGetCommandTypes 10405
DerivedOpen 10405
Destroy 10400
FConnectionProperties 10401
FDBXContext 10401
FIsolationLevel 10401
GetCommands 10406
GetCommandTypes 10406
GetConnectionProperties 10406
GetDatabaseMetaData 10406
GetErrorEvent 10407
GetIsOpen 10407
GetTraceFlags 10407
GetTraceInfoEvent 10407
IsOpen 10411
OnErrorEvent 10411
OnTrace 10412
Open 10407
Rollback 10408
RollbackFreeAndNil 10408
RollbackIncompleteFreeAndNil 10408
SetConnectionProperties 10409
SetErrorEvent 10409
SetTraceFlags 10409

SetTraceInfoEvent 10410
TDBXConnection events 10411
TDBXConnection fields 10401
TDBXConnection Members 10399
TDBXConnection methods 10402
TDBXConnection properties 10410
TraceFlags 10411
TDBXConnectionBuilder class 10412
about TDBXConnectionBuilder class 10412
ConnectionProperties 10415
Create 10413
Create constructor 10413
CreateDelegateeConnection 10414
DbxContext 10415
DelegationSignature 10415
Destroy 10414
InputConnectionName 10416
InputConnectionProperties 10416
InputPassword 10416
InputUserName 10417
TDBXConnectionBuilder Members 10412
TDBXConnectionBuilder methods 10414
TDBXConnectionBuilder properties 10414
TDBXConnectionEx class 10417
about TDBXConnectionEx class 10417
ConnectionProperty 10421
CreateMorphCommand 10419
Destroy 10419
GetConnectionProperty 10420
GetProductName 10420
GetProductVersion 10420
GetVendorProperty 10420
MetaDataReader 10421
ProductName 10421
ProductVersion 10421
TDBXConnectionEx Members 10418
TDBXConnectionEx methods 10419
TDBXConnectionEx properties 10420
TDBXConnectionFactory class 10422
about TDBXConnectionFactory class 10422
Close 10425
Create 10423
CreateDelegatePaths 10426
DerivedClose 10426
DerivedOpen 10426
Destroy 10424
FDBXContext 10424
GetConnection 10426
GetConnection method 10426
GetConnectionDriver 10427
GetConnectionFactory 10427
GetConnectionItems 10428
GetConnectionProperties 10428
GetDriver 10428
GetDriverProperties 10429
GetErrorEvent 10429
GetTraceFlags 10429
GetTraceInfoEvent 10429
OnError 10432
OnTrace 10432
Open 10430
SetConnectionFactory 10430
SetErrorEvent 10430
SetTraceFlags 10430
SetTraceInfoEvent 10431
TDBXConnectionFactory events 10431
TDBXConnectionFactory fields 10424
TDBXConnectionFactory Members 10423
TDBXConnectionFactory methods 10425
TDBXConnectionFactory properties 10431
TraceFlags 10431
TDBXContext class 10432
about TDBXContext class 10432
ClosedByteReader 10436
Create 10434
Create constructor 10434
Destroy 10434
Error 10435

FOnErrorDBXContext 10435
FOnTraceDBXContext 10435
IsTracing 10436
OnError 10437
OnTrace 10438
TDBXContext events 10437
TDBXContext fields 10434
TDBXContext Members 10433
TDBXContext methods 10435
TDBXContext properties 10436
Trace 10436
TraceFlags 10437
TDBXCursorValue class 10438
about TDBXCursorValue class 10438
IsNull 10440
TDBXCursorValue Members 10439
TDBXCursorValue methods 10440
TDBXDatabaseMetaData class 10440
about TDBXDatabaseMetaData class 10440
Create 10441
FDBXContext 10442
MaxCommands 10442
ProcedureQuoteChar 10442
QuoteChar 10443
SupportsNestedTransactions 10443
SupportsRowSetSize 10443
SupportsTransactions 10444
TDBXDatabaseMetaData fields 10441
TDBXDatabaseMetaData Members 10441
TDBXDatabaseMetaData properties 10442
TDBXDatabaseMetaDataEx class 10444
about TDBXDatabaseMetaDataEx class 10444
Create 10445
MetaDataVersion 10446
QuotePrefix 10446
QuoteSuffix 10446
SupportsLowerCaseIdentifiers 10446
SupportsUpperCaseIdentifiers 10447
TDBXDatabaseMetaDataEx Members 10445
TDBXDatabaseMetaDataEx properties 10445
TDBXDataTypes class 10447
about TDBXDataTypes class 10447
AdtDateSubType 10450
AdtNestedTableSubType 10451
AdtType 10451
AnsiStringType 10451
ArrayType 10451
AutoIncSubType 10451
BcdType 10452
BFileSubType 10452
BinarySubType 10452
BlobType 10452
BooleanType 10453
BytesType 10453
CursorType 10453
DateType 10453
DoubleType 10453
FixedSubType 10454
HBinarySubType 10454
HMemoSubType 10454
Int16Type 10454
Int32Type 10455
Int64Type 10455
MaxBaseTypes 10455
MemoSubType 10455
MoneySubType 10455
OracleIntervalSubType 10456
OracleTimeStampSubType 10456
RefType 10456
TableType 10456
TDBXDataTypes constants 10449
TDBXDataTypes Members 10449
TimeStampType 10457
TimeType 10457
UInt16Type 10457
Uint32Type 10457
Uint64Type 10457
UnknownType 10458

VarBytesType 10458
WideMemoSubType 10458
WideStringType 10458
TDBXDataTypesColumns class 10707
about TDBXDataTypesColumns class 10707
ColumnSize 10710
CreateFormat 10710
CreateParameters 10711
DataType 10711
DbxDataType 10711
IsAutoIncrementable 10711
IsBestMatch 10711
IsCaseSensitive 10712
IsConcurrencyType 10712
IsFixedLength 10712
IsFixedPrecisionScale 10712
IsLiteralSupported 10713
IsLong 10713
IsNullable 10713
IsSearchable 10713
IsSearchableWithLike 10713
IsUnicode 10714
IsUnsigned 10714
LiteralPrefix 10714
LiteralSuffix 10714
MaximumScale 10715
MaximumVersion 10715
MinimumScale 10715
MinimumVersion 10715
ProviderDbType 10715
TDBXDataTypesColumns constants 10709
TDBXDataTypesColumns Members 10709
TypeName 10716
TDBXDataTypesEx class 10459
about TDBXDataTypesEx class 10459
CharArrayType 10460
Int8Type 10460
IntervalType 10460
ObjectType 10460
SingleType 10461
TDBXDataTypesEx constants 10460
TDBXDataTypesEx Members 10459
Uint8Type 10461
TDBXDataTypesIndex class 10716
about TDBXDataTypesIndex class 10716
ColumnSize 10719
CreateFormat 10719
CreateParameters 10719
DataType 10720
DbxDataType 10720
IsAutoIncrementable 10720
IsBestMatch 10720
IsCaseSensitive 10720
IsConcurrencyType 10721
IsFixedLength 10721
IsFixedPrecisionScale 10721
IsLiteralSupported 10721
IsLong 10722
IsNullable 10722
IsSearchable 10722
IsSearchableWithLike 10722
IsUnicode 10722
IsUnsigned 10723
Last 10723
LiteralPrefix 10723
LiteralSuffix 10723
MaximumScale 10724
MaximumVersion 10724
MinimumScale 10724
MinimumVersion 10724
ProviderDbType 10724
TDBXDataTypesIndex constants 10718
TDBXDataTypesIndex Members 10718
TypeName 10725
TDBXDateValue class 10461
about TDBXDateValue class 10461
GetDate 10463
IsNull 10463

SetDate 10463
TDBXDateValue Members 10462
TDBXDateValue methods 10463
TDBXDelegateDriver class 10464
about TDBXDelegateDriver class 10464
Close 10466
Create 10465
CreateConnection 10466
CreateMorphCommand 10466
Destroy 10465
GetDriverName 10466
GetDriverProperties 10467
GetDriverVersion 10467
SetDriverName 10467
SetDriverProperties 10467
TDBXDelegateDriver Members 10464
TDBXDelegateDriver methods 10465
TDBXDelegateItem class 10468
about TDBXDelegateItem class 10468
Destroy 10468
FName 10469
FNext 10469
FProperties 10469
TDBXDelegateItem fields 10469
TDBXDelegateItem Members 10468
TDBXDoubleValue class 10469
about TDBXDoubleValue class 10469
GetDouble 10471
IsNull 10471
SetDouble 10472
TDBXDoubleValue Members 10470
TDBXDoubleValue methods 10471
TDBXDriver class 10472
about TDBXDriver class 10472
CacheUntilFinalization 10474
Close 10474
CreateConnection 10475
Destroy 10473
DriverName 10476
DriverProperties 10477
FDBXContext 10473
FDriverLoader 10474
GetDriverName 10475
GetDriverProperties 10475
GetDriverVersion 10475
SetDriverName 10476
SetDriverProperties 10476
TDBXDriver fields 10473
TDBXDriver Members 10472
TDBXDriver methods 10474
TDBXDriver properties 10476
TDBXDriverEx class 10477
about TDBXDriverEx class 10477
AddCommandFactory 10479
Create 10478
CreateMorphCommand 10479
Destroy 10478
FCommandFactories 10479
LoadMetaDataCommandFactory 10479
TDBXDriverEx fields 10478
TDBXDriverEx Members 10477
TDBXDriverEx methods 10479
TDBXDriverHelp class 10480
about TDBXDriverHelp class 10480
CreateConnectionBuilder 10481
CreateTDBXContext 10481
CreateTDBXParameter 10481
CreateTDBXProperties 10481
CreateTDBXValueType 10481
TDBXDriverHelp Members 10480
TDBXDriverHelp methods 10480
TDBXDriverHelpEx class 10482
about TDBXDriverHelpEx class 10482
CreateTDBXValueType 10483
GetStreamReader 10483
IsReadOnlyValueType 10483
TDBXDriverHelpEx Members 10482
TDBXDriverHelpEx methods 10482

UpdateParameterType 10483
TDBXDriverLoader class 10483
about TDBXDriverLoader class 10483
FLoaderName 10484
Load 10485
TDBXDriverLoader fields 10484
TDBXDriverLoader Members 10484
TDBXDriverLoader methods 10485
TDBXDriverRegistry class 10485
about TDBXDriverRegistry class 10485
Create 10486
Destroy 10486
TDBXDriverRegistry Members 10485
TDBXError class 10486
about TDBXError class 10486
Create 10488
ErrorCode 10489
ErrorCodeToString 10488
Message 10489
TDBXError Members 10487
TDBXError methods 10488
TDBXError properties 10489
TDBXErrorCodes class 10489
about TDBXErrorCodes class 10489
ConnectionFailed 10493
DriverIncompatible 10493
DriverInitFailed 10493
DriverRestricted 10494
DuplicateTransactionId 10494
EOF 10494
InvalidArgument 10494
InvalidHandle 10495
InvalidIsolationLevel 10495
InvalidLength 10495
InvalidOperation 10495
InvalidOrdinal 10495
InvalidParameter 10496
InvalidPrecision 10496
InvalidReference 10496
InvalidTime 10496
InvalidTransactionId 10497
InvalidType 10497
InvalidUserOrPassword 10497
MaxCommonErrors 10497
MissingParameterMarker 10497
MultipleTransactionNotEnabled 10498
NoData 10498
NoMemory 10498
None 10498
NoTable 10499
NotImplemented 10499
NotSupported 10499
OptimisticLockFailed 10499
ParameterNotSet 10499
TDBXErrorCodes constants 10492
TDBXErrorCodes Members 10491
TransactionActive 10500
UnsupportedFieldType 10500
VendorError 10500
Warning 10500
TDBXForeignKeyColumnsColumns class 10725
about TDBXForeignKeyColumnsColumns class 10725
CatalogName 10727
ColumnName 10727
ForeignKeyName 10727
Ordinal 10727
PrimaryCatalogName 10728
PrimaryColumnName 10728
PrimaryKeyName 10728
PrimarySchemaName 10728
PrimaryTableName 10729
SchemaName 10729
TableName 10729
TDBXForeignKeyColumnsColumns constants 10726
TDBXForeignKeyColumnsColumns Members 10726
TDBXForeignKeyColumnsIndex class 10729
about TDBXForeignKeyColumnsIndex class 10729
CatalogName 10731

ColumnName 10731
ForeignKeyName 10732
Last 10732
Ordinal 10732
PrimaryCatalogName 10732
PrimaryColumnName 10732
PrimaryKeyName 10733
PrimarySchemaName 10733
PrimaryTableName 10733
SchemaName 10733
TableName 10734
TDBXForeignKeyColumnsIndex constants 10731
TDBXForeignKeyColumnsIndex Members 10730
TDBXForeignKeysColumns class 10734
about TDBXForeignKeysColumns class 10734
CatalogName 10735
ForeignKeyName 10735
SchemaName 10735
TableName 10736
TDBXForeignKeysColumns constants 10735
TDBXForeignKeysColumns Members 10734
TDBXForeignKeysIndex class 10736
about TDBXForeignKeysIndex class 10736
CatalogName 10737
ForeignKeyName 10737
Last 10737
SchemaName 10738
TableName 10738
TDBXForeignKeysIndex constants 10737
TDBXForeignKeysIndex Members 10736
TDBXIndexColumnsColumns class 10738
about TDBXIndexColumnsColumns class 10738
CatalogName 10740
ColumnName 10740
IndexName 10740
IsAscending 10740
Ordinal 10740
SchemaName 10741
TableName 10741
TDBXIndexColumnsColumns constants 10739
TDBXIndexColumnsColumns Members 10739
TDBXIndexColumnsIndex class 10741
about TDBXIndexColumnsIndex class 10741
CatalogName 10743
ColumnName 10743
IndexName 10743
IsAscending 10743
Last 10743
Ordinal 10744
SchemaName 10744
TableName 10744
TDBXIndexColumnsIndex constants 10742
TDBXIndexColumnsIndex Members 10742
TDBXIndexesColumns class 10744
about TDBXIndexesColumns class 10744
CatalogName 10746
ConstraintName 10746
IndexName 10746
IsAscending 10746
IsPrimary 10747
IsUnique 10747
SchemaName 10747
TableName 10747
TDBXIndexesColumns constants 10745
TDBXIndexesColumns Members 10745
TDBXIndexesIndex class 10748
about TDBXIndexesIndex class 10748
CatalogName 10749
ConstraintName 10749
IndexName 10749
IsAscending 10750
IsPrimary 10750
IsUnique 10750
Last 10750
SchemaName 10751
TableName 10751
TDBXIndexesIndex constants 10749
TDBXIndexesIndex Members 10748

TDBXIniFileConnectionFactory class 10501
about TDBXIniFileConnectionFactory class 10501
ConnectionsFile 10503
Create 10502
DerivedClose 10503
DerivedOpen 10503
Destroy 10502
DriversFile 10504
TDBXIniFileConnectionFactory Members 10501
TDBXIniFileConnectionFactory methods 10503
TDBXIniFileConnectionFactory properties 10503
TDBXIniFileConnectionFactoryEx class 10504
about TDBXIniFileConnectionFactoryEx class 10504
GetDriverItems 10506
TDBXIniFileConnectionFactoryEx Members 10504
TDBXIniFileConnectionFactoryEx methods 10505
TDBXInt16Value class 10506
about TDBXInt16Value class 10506
GetInt16 10508
IsNull 10508
SetInt16 10508
TDBXInt16Value Members 10506
TDBXInt16Value methods 10507
TDBXInt32Value class 10508
about TDBXInt32Value class 10508
GetInt32 10510
IsNull 10510
SetInt32 10511
TDBXInt32Value Members 10509
TDBXInt32Value methods 10510
TDBXInt64Value class 10511
about TDBXInt64Value class 10511
GetInt64 10512
IsNull 10513
SetInt64 10513
TDBXInt64Value Members 10511
TDBXInt64Value methods 10512
TDBXIsolations class 10513
about TDBXIsolations class 10513
DirtyRead 10515
ReadCommitted 10515
RepeatableRead 10515
Serializable 10515
SnapShot 10515
TDBXIsolations constants 10514
TDBXIsolations Members 10514
TDBXLookAheadStreamReader class 10516
about TDBXLookAheadStreamReader class 10516
ConvertToMemoryStream 10517
Destroy 10516
Eos 10517
Read 10517
SetStream 10518
Size 10518
TDBXLookAheadStreamReader Members 10516
TDBXLookAheadStreamReader methods 10517
TDBXMemoryConnectionFactory class 10518
about TDBXMemoryConnectionFactory class 10518
Create 10519
DerivedClose 10520
DerivedOpen 10520
Destroy 10520
TDBXMemoryConnectionFactory Members 10518
TDBXMemoryConnectionFactory methods 10520
TDBXMetaDatabaseColumnNames class 10521
about TDBXMetaDatabaseColumnNames class 10521
MaxCommands 10523
ProcedureQuoteChar 10523
ProductName 10523
ProductVersion 10523
QuoteChar 10524
QuotePrefix 10524
QuoteSuffix 10524
SupportsLowerCaseIdentifiers 10524
SupportsNestedTransactions 10524
SupportsRowSetSize 10525
SupportsTransactions 10525
SupportsUpperCaseIdentifiers 10525

TDBXMetaDatabaseColumnNames constants 10522
TDBXMetaDatabaseColumnNames Members 10522
TDBXMetaDataCollectionIndex class 10751
about TDBXMetaDataCollectionIndex class 10751
Catalogs 10753
ColumnConstraints 10754
Columns 10754
DataTypes 10754
ForeignKeyColumns 10754
ForeignKeys 10755
IndexColumns 10755
Indexes 10755
PackageProcedureParameters 10755
PackageProcedures 10755
Packages 10756
PackageSources 10756
ProcedureParameters 10756
Procedures 10756
ProcedureSources 10757
ReservedWords 10757
Roles 10757
Schemas 10757
Synonyms 10757
Tables 10758
TDBXMetaDataCollectionIndex constants 10753
TDBXMetaDataCollectionIndex Members 10752
Users 10758
Views 10758
TDBXMetaDataCollectionName class 10758
about TDBXMetaDataCollectionName class 10758
Catalogs 10761
ColumnConstraints 10761
Columns 10761
DataTypes 10762
ForeignKeyColumns 10762
ForeignKeys 10762
IndexColumns 10762
Indexes 10762
PackageProcedureParameters 10763
PackageProcedures 10763
Packages 10763
PackageSources 10763
ProcedureParameters 10764
Procedures 10764
ProcedureSources 10764
ReservedWords 10764
Roles 10764
Schemas 10765
Synonyms 10765
Tables 10765
TDBXMetaDataCollectionName constants 10760
TDBXMetaDataCollectionName Members 10760
Users 10765
Views 10766
TDBXMetaDataCommands class 10525
about TDBXMetaDataCommands class 10525
GetColumns 10529
GetDatabase 10529
GetDataTypes 10529
GetForeignKeyColumns 10529
GetForeignKeys 10530
GetIndexColumns 10530
GetIndexes 10530
GetPackages 10530
GetProcedureParameters 10531
GetProcedures 10531
GetTables 10531
GetUsers 10531
TDBXMetaDataCommands constants 10528
TDBXMetaDataCommands Members 10528
TDBXMetaDataCommandsEx class 10531
about TDBXMetaDataCommandsEx class 10531
GetCatalogs 10535
GetPackageProcedureParameters 10535
GetPackageProcedures 10535
GetPackageSources 10535
GetProcedureSources 10536
GetReservedWords 10536

GetRoles 10536
GetSchemas 10536
GetSynonyms 10536
GetViews 10537
TDBXMetaDataCommandsEx constants 10534
TDBXMetaDataCommandsEx Members 10534
TDBXMetaDataKeyword class 10537
about TDBXMetaDataKeyword class 10537
ForeignKey 10538
PrimaryKey 10538
TDBXMetaDataKeyword constants 10538
TDBXMetaDataKeyword Members 10537
TDBXMetaDataProcedureTypes class 10538
about TDBXMetaDataProcedureTypes class 10538
FunctionType 10539
ProcedureType 10539
TDBXMetaDataProcedureTypes constants 10539
TDBXMetaDataProcedureTypes Members 10539
TDBXMetaDataTableTypes class 10540
about TDBXMetaDataTableTypes class 10540
Synonym 10541
SystemTable 10541
Table 10541
TDBXMetaDataTableTypes constants 10540
TDBXMetaDataTableTypes Members 10540
View 10541
TDBXMetaDataTableTypesEx class 10542
about TDBXMetaDataTableTypesEx class 10542
SystemView 10543
TDBXMetaDataTableTypesEx constants 10542
TDBXMetaDataTableTypesEx Members 10542
TDBXNullValue class 10543
about TDBXNullValue class 10543
Create 10544
Destroy 10544
IsNull 10545
TDBXNullValue Members 10543
TDBXNullValue methods 10545
TDBXPackageProcedureParametersColumns class 10766
about TDBXPackageProcedureParametersColumns class 10766
CatalogName 10768
DbxDataType 10769
IsFixedLength 10769
IsLong 10769
IsNullable 10769
IsUnicode 10770
IsUnsigned 10770
Ordinal 10770
PackageName 10770
ParameterMode 10770
ParameterName 10771
Precision 10771
ProcedureName 10771
Scale 10771
SchemaName 10772
TDBXPackageProcedureParametersColumns constants 10768
TDBXPackageProcedureParametersColumns Members 10767
TypeName 10772
TDBXPackageProcedureParametersIndex class 10772
about TDBXPackageProcedureParametersIndex class 10772
CatalogName 10775
DbxDataType 10775
IsFixedLength 10775
IsLong 10775
IsNullable 10776
IsUnicode 10776
IsUnsigned 10776
Last 10776
Ordinal 10776
PackageName 10777
ParameterMode 10777
ParameterName 10777
Precision 10777
ProcedureName 10778
Scale 10778
SchemaName 10778

wwww

TDBXPackageProcedureParametersIndex constants 10774
TDBXPackageProcedureParametersIndex Members 10773
TypeName 10778
TDBXPackageProceduresColumns class 10778
about TDBXPackageProceduresColumns class 10778
CatalogName 10780
PackageName 10780
ProcedureName 10780
ProcedureType 10780
SchemaName 10781
TDBXPackageProceduresColumns constants 10779
TDBXPackageProceduresColumns Members 10779
TDBXPackageProceduresIndex class 10781
about TDBXPackageProceduresIndex class 10781
CatalogName 10782
Last 10782
PackageName 10783
ProcedureName 10783
ProcedureType 10783
SchemaName 10783
TDBXPackageProceduresIndex constants 10782
TDBXPackageProceduresIndex Members 10782
TDBXPackagesColumns class 10783
about TDBXPackagesColumns class 10783
CatalogName 10785
PackageName 10785
SchemaName 10785
TDBXPackagesColumns constants 10784
TDBXPackagesColumns Members 10784
TDBXPackagesIndex class 10785
about TDBXPackagesIndex class 10785
CatalogName 10786
Last 10787
PackageName 10787
SchemaName 10787
TDBXPackagesIndex constants 10786
TDBXPackagesIndex Members 10786
TDBXPackageSourcesColumns class 10787
about TDBXPackageSourcesColumns class 10787
CatalogName 10788
Definition 10789
PackageName 10789
SchemaName 10789
TDBXPackageSourcesColumns constants 10788
TDBXPackageSourcesColumns Members 10788
TDBXPackageSourcesIndex class 10789
about TDBXPackageSourcesIndex class 10789
CatalogName 10790
Definition 10791
Last 10791
PackageName 10791
SchemaName 10791
TDBXPackageSourcesIndex constants 10790
TDBXPackageSourcesIndex Members 10790
TDBXParameter class 10545
about TDBXParameter class 10545
Create 10547
Destroy 10547
GetValue 10547
SetDbxRow 10548
SetParameter 10548
TDBXParameter Members 10546
TDBXParameter methods 10547
TDBXParameter properties 10548
Value 10548
TDBXParameterDirections class 10549
about TDBXParameterDirections class 10549
InOutParameter 10550
InParameter 10550
OutParameter 10550
ReturnParameter 10550
TDBXParameterDirections constants 10549
TDBXParameterDirections Members 10549
Unknown 10551
TDBXParameterEx class 10551
about TDBXParameterEx class 10551
Assign 10553
Clone 10553

TDBXParameterEx Members 10551
TDBXParameterEx methods 10552
TDBXParameterList class 10553
about TDBXParameterList class 10553
AddParameter 10556
ClearParameters 10556
Count 10559
Create 10554
Destroy 10555
FDBXContext 10555
GetCount 10556
GetOrdinal 10556
GetParameterByOrdinal 10557
InsertParameter 10557
Parameter 10559
RemoveParameter 10558
RemoveParameter method 10557
SetCount 10558
SetParameter 10558
TDBXParameterList fields 10555
TDBXParameterList Members 10554
TDBXParameterList methods 10555
TDBXParameterList properties 10558
TDBXProcedureParametersColumns class 10792
about TDBXProcedureParametersColumns class 10792
CatalogName 10794
DbxDataType 10794
IsFixedLength 10794
IsLong 10794
IsNullable 10795
IsUnicode 10795
IsUnsigned 10795
Ordinal 10795
ParameterMode 10796
ParameterName 10796
Precision 10796
ProcedureName 10796
Scale 10796
SchemaName 10797
TDBXProcedureParametersColumns constants 10793
TDBXProcedureParametersColumns Members 10793
TypeName 10797
TDBXProcedureParametersIndex class 10797
about TDBXProcedureParametersIndex class 10797
CatalogName 10799
DbxDataType 10800
IsFixedLength 10800
IsLong 10800
IsNullable 10800
IsUnicode 10800
IsUnsigned 10801
Last 10801
Ordinal 10801
ParameterMode 10801
ParameterName 10802
Precision 10802
ProcedureName 10802
Scale 10802
SchemaName 10802
TDBXProcedureParametersIndex constants 10799
TDBXProcedureParametersIndex Members 10798
TypeName 10803
TDBXProceduresColumns class 10803
about TDBXProceduresColumns class 10803
CatalogName 10804
ProcedureName 10804
ProcedureType 10804
SchemaName 10805
TDBXProceduresColumns constants 10804
TDBXProceduresColumns Members 10804
TDBXProceduresIndex class 10805
about TDBXProceduresIndex class 10805
CatalogName 10806
Last 10806
ProcedureName 10807
ProcedureType 10807
SchemaName 10807
TDBXProceduresIndex constants 10806

TDBXProceduresIndex Members 10806
TDBXProcedureSourcesColumns class 10807
about TDBXProcedureSourcesColumns class 10807
CatalogName 10809
Definition 10809
ExternalDefinition 10809
ProcedureName 10809
ProcedureType 10809
SchemaName 10810
TDBXProcedureSourcesColumns constants 10808
TDBXProcedureSourcesColumns Members 10808
TDBXProcedureSourcesIndex class 10810
about TDBXProcedureSourcesIndex class 10810
CatalogName 10811
Definition 10812
ExternalDefinition 10812
Last 10812
ProcedureName 10812
ProcedureType 10812
SchemaName 10813
TDBXProcedureSourcesIndex constants 10811
TDBXProcedureSourcesIndex Members 10811
TDBXProperties class 10559
about TDBXProperties class 10559
Add 10561
AddProperties 10561
Clone 10562
Create 10560
Create constructor 10560
Destroy 10561
GetBoolean 10562
GetInteger 10562
GetLists 10562
GetRequiredValue 10563
Properties 10563
SetProperties 10563
TDBXProperties Members 10559
TDBXProperties methods 10561
TDBXProperties properties 10563
Values 10564
TDBXPropertiesItem class 10564
about TDBXPropertiesItem class 10564
Create 10565
Destroy 10565
FDeletgatePath 10565
FProperties 10566
TDBXPropertiesItem fields 10565
TDBXPropertiesItem Members 10564
TDBXPropertyNames class 10566
about TDBXPropertyNames class 10566
ConnectionName 10569
Database 10569
DelegateConnection 10569
DelegateDriver 10570
DriverAssembly 10570
DriverAssemblyLoader 10570
DriverName 10570
DriverPackage 10570
DriverPackageLoader 10571
DriverUnit 10571
ErrorResourceFile 10571
GetDriverFunc 10571
HostName 10572
InstalledDrivers 10572
IsolationLevel 10572
LibraryName 10572
MaxBlobSize 10572
Password 10573
Port 10573
Role 10573
SchemaOverride 10573
TDBXPropertyNames constants 10568
TDBXPropertyNames Members 10568
UserName 10574
VendorLib 10574
TDBXPropertyNamesEx class 10574
about TDBXPropertyNamesEx class 10574
MetaDataAssemblyLoader 10575

MetaDataPackageLoader 10576
ProductName 10576
ProductVersion 10576
TDBXPropertyNamesEx constants 10575
TDBXPropertyNamesEx Members 10575
TDBXReader class 10576
about TDBXReader class 10576
ByteReader 10583
Closed 10583
ColumnCount 10584
Create 10578
DerivedClose 10580
DerivedNext 10580
Destroy 10578
FailIfClosed 10580
FDBXContext 10579
FDbxRow 10579
GetByteReader 10580
GetColumnCount 10580
GetErrorEvent 10581
GetObjectTypeName 10581
GetOrdinal 10581
GetValue 10582
GetValueByName 10582
GetValueType 10582
Next 10582
OnErrorEvent 10585
SetValues 10583
TDBXReader events 10585
TDBXReader fields 10578
TDBXReader Members 10577
TDBXReader methods 10579
TDBXReader properties 10583
Value 10584
ValueType 10584
TDBXReaderByteReader class 10585
about TDBXReaderByteReader class 10585
Create 10586
GetAnsiString 10587
GetBcd 10587
GetByteLength 10587
GetBytes 10587
GetDate 10588
GetDouble 10588
GetInt16 10588
GetInt32 10588
GetInt64 10589
GetTime 10589
GetTimeStamp 10589
GetWideString 10589
TDBXReaderByteReader Members 10586
TDBXReaderByteReader methods 10586
TDBXReservedWordsColumns class 10813
about TDBXReservedWordsColumns class 10813
ReservedWord 10814
TDBXReservedWordsColumns constants 10814
TDBXReservedWordsColumns Members 10814
TDBXReservedWordsIndex class 10814
about TDBXReservedWordsIndex class 10814
Last 10815
ReservedWord 10815
TDBXReservedWordsIndex constants 10815
TDBXReservedWordsIndex Members 10815
TDBXRolesColumns class 10816
about TDBXRolesColumns class 10816
RoleName 10817
TDBXRolesColumns constants 10816
TDBXRolesColumns Members 10816
TDBXRolesIndex class 10817
about TDBXRolesIndex class 10817
Last 10818
RoleName 10818
TDBXRolesIndex constants 10818
TDBXRolesIndex Members 10817
TDBXRow class 10590
about TDBXRow class 10590
Create 10591
FDBXContext 10591

FGeneration 10592
Generation 10599
GetAnsiString 10593
GetBcd 10593
GetBoolean 10593
GetByteLength 10593
GetBytes 10594
GetDate 10594
GetDouble 10594
GetInt16 10594
GetInt32 10595
GetInt64 10595
GetObjectTypeName 10595
GetTime 10595
GetTimeStamp 10595
GetWideString 10596
NotImplemented 10596
SetBCD 10596
SetBoolean 10596
SetDate 10597
SetDouble 10597
SetDynamicBytes 10597
SetInt16 10597
SetInt32 10597
SetInt64 10598
SetNull 10598
SetString 10598
SetTime 10598
SetTimestamp 10599
SetValueType 10599
SetWideString 10599
TDBXRow fields 10591
TDBXRow Members 10590
TDBXRow methods 10592
TDBXRow properties 10599
TDBXRowEx class 10600
about TDBXRowEx class 10600
CreateCustomValue 10601
GetLength 10602
GetStream 10602
GetStream method 10602
GetStreamBytes 10603
GetStreamLength 10603
GetWideChars 10603
SetStream 10603
SetWideChars 10604
TDBXRowEx Members 10600
TDBXRowEx methods 10601
UseExtendedTypes 10604
TDBXSchemasColumns class 10818
about TDBXSchemasColumns class 10818
CatalogName 10819
SchemaName 10820
TDBXSchemasColumns constants 10819
TDBXSchemasColumns Members 10819
TDBXSchemasIndex class 10820
about TDBXSchemasIndex class 10820
CatalogName 10821
Last 10821
SchemaName 10821
TDBXSchemasIndex constants 10821
TDBXSchemasIndex Members 10820
TDBXStateItem class 10604
about TDBXStateItem class 10604
Close 10605
TDBXStateItem Members 10604
TDBXStateItem methods 10605
TDBXStateItemList class 10605
about TDBXStateItemList class 10605
AddStateItem 10606
Destroy 10606
GetStateItem 10606
RemoveAndFreeStateItem 10606
TDBXStateItemList Members 10605
TDBXStateItemList methods 10606
TDBXStreamReader class 10607
about TDBXStreamReader class 10607
Eos 10608

Read 10608
Size 10608
TDBXStreamReader Members 10607
TDBXStreamReader methods 10607
TDBXStreamValue class 10608
about TDBXStreamValue class 10608
Create 10610
Destroy 10610
GetStream 10611
GetValueSize 10611
SetDynamicBytes 10611
SetStream 10611
TDBXStreamValue Members 10609
TDBXStreamValue methods 10610
UpdateType 10612
TDBXSynonymsColumns class 10822
about TDBXSynonymsColumns class 10822
CatalogName 10823
SchemaName 10823
SynonymName 10823
TableCatalogName 10824
TableName 10824
TableSchemaName 10824
TDBXSynonymsColumns constants 10823
TDBXSynonymsColumns Members 10822
TDBXSynonymsIndex class 10824
about TDBXSynonymsIndex class 10824
CatalogName 10826
Last 10826
SchemaName 10826
SynonymName 10826
TableCatalogName 10827
TableName 10827
TableSchemaName 10827
TDBXSynonymsIndex constants 10825
TDBXSynonymsIndex Members 10825
TDBXTablesColumns class 10827
about TDBXTablesColumns class 10827
CatalogName 10828
SchemaName 10829
TableName 10829
TableType 10829
TDBXTablesColumns constants 10828
TDBXTablesColumns Members 10828
TDBXTablesIndex class 10829
about TDBXTablesIndex class 10829
CatalogName 10831
Last 10831
SchemaName 10831
TableName 10831
TableType 10831
TDBXTablesIndex constants 10830
TDBXTablesIndex Members 10830
TDBXTimeStampValue class 10612
about TDBXTimeStampValue class 10612
GetTimeStamp 10614
IsNull 10614
SetTimeStamp 10614
TDBXTimeStampValue Members 10612
TDBXTimeStampValue methods 10613
TDBXTimeValue class 10614
about TDBXTimeValue class 10614
GetTime 10616
IsNull 10616
SetTime 10617
TDBXTimeValue Members 10615
TDBXTimeValue methods 10616
TDBXTraceFlags class 10617
about TDBXTraceFlags class 10617
Blob 10619
Command 10619
Connect 10619
Driver 10619
DriverLoad 10619
Error 10620
Execute 10620
MetaData 10620
Misc 10620

None 10621
Parameter 10621
Prepare 10621
Reader 10621
TDBXTraceFlags constants 10618
TDBXTraceFlags Members 10618
Transact 10621
Vendor 10622
TDBXTraceInfo record 10622
about TDBXTraceInfo record 10622
Message 10623
TDBXTraceInfo Members 10622
TDBXTraceInfo properties 10623
TraceFlag 10623
TDBXTransaction class 10623
about TDBXTransaction class 10623
Connection 10625
Create 10624
Destroy 10624
FIsolationLevel 10625
TDBXTransaction fields 10625
TDBXTransaction Members 10624
TDBXTransaction properties 10625
TDBXUsersColumns class 10832
about TDBXUsersColumns class 10832
TDBXUsersColumns constants 10832
TDBXUsersColumns Members 10832
UserName 10833
TDBXUsersIndex class 10833
about TDBXUsersIndex class 10833
Last 10834
TDBXUsersIndex constants 10834
TDBXUsersIndex Members 10833
UserName 10834
TDBXValue class 10626
about TDBXValue class 10626
Create 10627
CreateValue 10628
DbxContext 10633
Destroy 10627
GetAnsiString 10629
GetBcd 10629
GetBoolean 10629
GetBytes 10629
GetDate 10630
GetDbxContext 10630
GetDouble 10630
GetInt16 10631
GetInt32 10631
GetInt64 10631
GetTime 10631
GetTimeStamp 10632
GetValueSize 10632
GetWideString 10632
IsNull 10632
TDBXValue Members 10626
TDBXValue methods 10628
TDBXValue properties 10633
ValueType 10633
TDBXValueList class 10634
about TDBXValueList class 10634
Destroy 10634
FindOrdinal 10635
FLastOrdinal 10635
FValueTypeArray 10635
TDBXValueList fields 10635
TDBXValueList Members 10634
TDBXValueList methods 10635
TDBXValueType class 10636
about TDBXValueType class 10636
AutoIncrement 10645
ChildPosition 10645
Create 10637
DataType 10646
FDBXContext 10637
GetChildPosition 10638
GetDataType 10639
GetFlags 10639

GetName 10639
GetOrdinal 10639
GetParameterDirection 10640
GetPrecision 10640
GetScale 10640
GetSize 10641
GetSubType 10641
Name 10646
Nullable 10646
Ordinal 10646
ParameterDirection 10647
Precision 10647
ReadOnly 10647
Scale 10647
Searchable 10648
SetChildPosition 10641
SetDataType 10642
SetFlags 10642
SetName 10642
SetNullable 10642
SetOrdinal 10643
SetParameterDirection 10643
SetPrecision 10643
SetScale 10644
SetSize 10644
SetSubType 10644
Size 10648
SubType 10648
TDBXValueType fields 10637
TDBXValueType Members 10636
TDBXValueType methods 10638
TDBXValueType properties 10645
ValueTypeFlags 10648
TDBXValueTypeEx class 10649
about TDBXValueTypeEx class 10649
DataTypeName 10650
InvalidTypeAccess 10651
TDBXValueTypeEx Members 10649
TDBXValueTypeEx methods 10650
TDBXValueTypeFlags class 10651
about TDBXValueTypeFlags class 10651
AutoIncrement 10652
Nullable 10652
ReadOnly 10653
ReadOnlyType 10653
Searchable 10653
TDBXValueTypeFlags constants 10652
TDBXValueTypeFlags Members 10652
TDBXValueTypeFlagsEx class 10653
about TDBXValueTypeFlagsEx class 10653
ExtendedType 10654
TDBXValueTypeFlagsEx constants 10654
TDBXValueTypeFlagsEx Members 10654
TDBXViewsColumns class 10834
about TDBXViewsColumns class 10834
CatalogName 10835
Definition 10836
SchemaName 10836
TDBXViewsColumns constants 10835
TDBXViewsColumns Members 10835
ViewName 10836
TDBXViewsIndex class 10836
about TDBXViewsIndex class 10836
CatalogName 10838
Definition 10838
Last 10838
SchemaName 10838
TDBXViewsIndex constants 10837
TDBXViewsIndex Members 10837
ViewName 10838
TDBXWideCharsValue class 10655
about TDBXWideCharsValue class 10655
Create 10656
Destroy 10656
GetStream 10657
GetWideString 10657
IsNull 10657
SetWideString 10658

eeee

TDBXWideCharsValue Members 10655
TDBXWideCharsValue methods 10657
TDBXWideStringValue class 10658
about TDBXWideStringValue class 10658
Create 10659
Destroy 10660
GetWideString 10660
IsNull 10660
SetWideString 10661
TDBXWideStringValue Members 10658
TDBXWideStringValue methods 10660
TDBXWritableValue class 10661
about TDBXWritableValue class 10661
Create 10662
Destroy 10663
SetAnsiString 10663
SetBcd 10664
SetBoolean 10664
SetDate 10664
SetDouble 10664
SetDynamicBytes 10665
SetInt16 10665
SetInt32 10665
SetInt64 10665
SetNull 10666
SetPendingValue 10666
SetStaticBytes 10666
SetTime 10666
SetTimeStamp 10667
SetWideString 10667
TDBXWritableValue Members 10661
TDBXWritableValue methods 10663
tdChanged enumeration member 17433
TDCOMConnection class 16008
about TDCOMConnection class 16008
ComputerName 16014
Create 16013
DoConnect 16013
ObjectBroker 16014
TDCOMConnection Members 16009
TDCOMConnection methods 16013
TDCOMConnection properties 16014
TDdeClientConv class 10858
about TDdeClientConv class 10858
ChangeLink 10863
ClearItems 10864
Close 10864
CloseLink 10864
ConnectMode 10873
Conv 10874
Create 10862
CreateDdeConv 10864
DataMode 10874
DdeFmt 10874
DdeService 10875
DdeTopic 10875
DefineProperties 10865
Destroy 10862
ExecuteMacro 10865
ExecuteMacroLines 10866
FormatChars 10876
GetCliItemByCtrl 10866
GetCliItemByName 10867
Loaded 10867
Notification 10867
OnAttach 10868
OnClose 10877
OnDetach 10868
OnOpen 10878
OnSetItem 10868
Open 10869
OpenLink 10869
PasteLink 10869
PokeData 10870
PokeDataLines 10870
ReadLinkInfo 10871
RequestData 10871
ServiceApplication 10876

SetLink 10872
StartAdvise 10872
TDdeClientConv events 10877
TDdeClientConv Members 10859
TDdeClientConv methods 10863
TDdeClientConv properties 10873
WaitStat 10877
WriteLinkInfo 10873
TDdeClientItem class 10878
about TDdeClientItem class 10878
Create 10881
DdeConv 10883
DdeItem 10883
Destroy 10882
Lines 10884
Notification 10882
OnChange 10885
TDdeClientItem events 10885
TDdeClientItem Members 10879
TDdeClientItem methods 10882
TDdeClientItem properties 10883
Text 10884
TDdeMgr class 10886
about TDdeMgr class 10886
AppName 10891
Create 10889
DdeInstId 10891
Destroy 10889
GetExeName 10890
LinkClipFmt 10891
TDdeMgr Members 10886
TDdeMgr methods 10890
TDdeMgr properties 10890
TDdeServerConv class 10891
about TDdeServerConv class 10891
Connect 10896
Create 10895
Destroy 10895
Disconnect 10896
ExecuteMacro 10896
OnClose 10897
OnExecuteMacro 10897
OnOpen 10898
TDdeServerConv events 10897
TDdeServerConv Members 10892
TDdeServerConv methods 10896
TDdeServerItem class 10898
about TDdeServerItem class 10898
Change 10903
CopyToClipboard 10903
Create 10901
Destroy 10902
Fmt 10906
GetText 10904
Lines 10906
Notification 10904
OnChange 10908
OnPokeData 10908
PokeData 10904
ServerConv 10907
SetLines 10905
SetServerConv 10905
SetText 10905
TDdeServerItem events 10908
TDdeServerItem Members 10899
TDdeServerItem methods 10902
TDdeServerItem properties 10906
Text 10907
tdDisplay enumeration member 17672
TDecisionCube class 17442
about TDecisionCube class 17442
Active 17451
AfterClose 17456
AfterOpen 17456
BeforeClose 17456
BeforeOpen 17457
CanDimBeClosed 17448
CanSumBeClosed 17448

Create 17447
DataSet 17451
DesignState 17452
Destroy 17447
DimensionMap 17453
GetDetailSQL 17448
GetSQL 17449
MaxCells 17453
MaxDimensions 17454
MaxSummaries 17454
OnLowCapacity 17457
OnRefresh 17458
ShowCubeDialog 17450
ShowProgressDialog 17455
StateChanged 17450
TDecisionCube events 17455
TDecisionCube Members 17443
TDecisionCube methods 17447
TDecisionCube properties 17450
TDecisionDataLink class 17458
about TDecisionDataLink class 17458
Create 17459
DecisionDataEvent 17461
DecisionSource 17462
Destroy 17460
FBlocked 17460
FDecisionSource 17461
SetDecisionSource 17461
TDecisionDataLink fields 17460
TDecisionDataLink Members 17458
TDecisionDataLink methods 17461
TDecisionDataLink properties 17461
TDecisionGraph class 17521
about TDecisionGraph class 17521
Align 17526
AllowPanning 17526
AllowZoom 17526
Anchors 17526
AnimatedZoom 17527
AnimatedZoomSteps 17527
AutoSize 17527
AxisVisible 17527
BackColor 17528
BackImage 17528
BackImageInside 17528
BackImageMode 17528
BevelInner 17528
BevelOuter 17529
BevelWidth 17529
BorderStyle 17529
BorderWidth 17529
BottomAxis 17530
BottomWall 17530
Chart3DPercent 17530
ClipPoints 17530
Color 17530
Constraints 17531
DecisionSource 17531
DragCursor 17531
DragKind 17531
DragMode 17532
Enabled 17532
Foot 17532
Frame 17532
Gradient 17533
LeftAxis 17533
LeftWall 17533
Legend 17533
MarginBottom 17533
MarginLeft 17534
MarginRight 17534
MarginTop 17534
MaxPointsPerPage 17534
Monochrome 17535
OnAfterDraw 17535
OnAllowScroll 17535
OnCanResize 17535
OnClick 17535

OnClickAxis 17536
OnClickBackground 17536
OnClickLegend 17536
OnClickSeries 17536
OnConstrainedResize 17537
OnDblClick 17537
OnDockDrop 17537
OnDockOver 17537
OnDragDrop 17537
OnDragOver 17538
OnEndDock 17538
OnEndDrag 17538
OnEnter 17538
OnExit 17539
OnGetAxisLabel 17539
OnGetLegendPos 17539
OnGetLegendRect 17539
OnGetLegendText 17539
OnGetNextAxisLabel 17540
OnGetSiteInfo 17540
OnMouseDown 17540
OnMouseMove 17540
OnMouseUp 17541
OnPageChange 17541
OnResize 17541
OnScroll 17541
OnStartDock 17541
OnStartDrag 17542
OnUnDock 17542
OnUndoZoom 17542
OnZoom 17542
Page 17543
ParentColor 17543
ParentShowHint 17543
PopupMenu 17543
RightAxis 17543
ScaleLastPage 17544
SeriesList 17544
ShowHint 17544
TabOrder 17544
TabStop 17545
TDecisionGraph Members 17522
TDecisionGraph properties 17524
Title 17545
TopAxis 17545
View3D 17545
View3DWalls 17545
Visible 17546
TDecisionGraphDataLink class 17546
about TDecisionGraphDataLink class 17546
Create 17547
DecisionDataEvent 17548
Destroy 17548
TDecisionGraphDataLink Members 17546
TDecisionGraphDataLink methods 17548
TDecisionGrid class 17586
about TDecisionGrid class 17586
Align 17601
Anchors 17605
BorderStyle 17605
CaptionColor 17605
CaptionFont 17606
ColCount 17606
Color 17607
Constraints 17608
Ctl3D 17608
DataColor 17609
DataFont 17609
DataSumColor 17610
DecisionSource 17610
DefaultColWidth 17610
DefaultDrawing 17611
DefaultRowHeight 17611
Dimensions 17612
DragCursor 17612
DragMode 17613
Enabled 17614
FixedCols 17615

FixedRows 17615
GridLineColor 17615
GridLineWidth 17616
LabelColor 17616
LabelFont 17616
LabelSumColor 17617
OnClick 17630
OnDblClick 17632
OnDecisionDrawCell 17632
OnDecisionExamineCell 17634
OnDragDrop 17634
OnDragOver 17637
OnEndDrag 17639
OnEnter 17640
OnExit 17642
OnKeyDown 17644
OnKeyPress 17647
OnKeyUp 17648
OnMouseDown 17650
OnMouseMove 17652
OnMouseUp 17654
OnResize 17656
OnStartDrag 17657
OnTopLeftChanged 17658
Options 17617
ParentColor 17618
ParentCtl3D 17618
ParentFont 17619
ParentShowHint 17619
PopupMenu 17620
RowCount 17624
ScrollBars 17625
ShowCubeEditor 17625
ShowHint 17625
TabOrder 17627
TabStop 17627
TDecisionGrid events 17630
TDecisionGrid Members 17587
TDecisionGrid properties 17600
Totals 17628
Visible 17628
TDecisionGridDataLink class 17658
about TDecisionGridDataLink class 17658
Create 17660
DecisionDataEvent 17660
Destroy 17660
TDecisionGridDataLink Members 17658
TDecisionGridDataLink methods 17660
TDecisionPivot class 17680
about TDecisionPivot class 17680
AdjustClientRect 17692
Align 17696
Alignment 17700
BevelInner 17700
BevelOuter 17701
BevelWidth 17702
BorderStyle 17703
BorderWidth 17704
ButtonAutoSize 17704
ButtonHeight 17705
ButtonSpacing 17705
ButtonWidth 17705
Color 17706
Cols 17707
Create 17691
Ctl3D 17707
DecisionSource 17707
Destroy 17692
DimStateChange 17693
DragCursor 17708
DragMode 17708
Enabled 17710
Extras 17710
Font 17710
GetBorderStyle 17693
GetBorderWidth 17693
GroupLayout 17711
Groups 17712

jjjj

GroupSpacing 17712
NewDimLayout 17694
NewPanelSetup 17694
Notification 17694
OnClick 17724
OnDblClick 17726
OnDragDrop 17726
OnDragOver 17728
OnEndDrag 17731
OnEnter 17732
OnExit 17734
OnResize 17736
OnStartDrag 17737
Pages 17713
ParentColor 17713
ParentCtl3D 17713
ParentFont 17714
ParentShowHint 17714
PopupMenu 17715
Rows 17719
SetBorderStyle 17695
SetBorderWidth 17695
SetBounds 17695
ShowHint 17720
TabOrder 17721
TabStop 17722
TDecisionPivot events 17724
TDecisionPivot Members 17681
TDecisionPivot methods 17692
TDecisionPivot properties 17696
Visible 17722
TDecisionQuery class 17854
about TDecisionQuery class 17854
TDecisionSource class 17462
about TDecisionSource class 17462
CloseDimIndexRight 17470
ControlType 17485
Create 17467
CurrentSum 17486
DecisionCube 17487
DefineProperties 17470
Destroy 17467
DrillDimIndex 17471
FOnAfterPivot 17468
FOnBeforePivot 17468
FOnLayoutChange 17468
FOnNewDimensions 17468
FOnStateChange 17468
FonSummaryChange 17469
Get2DDataAsVariant 17472
GetActiveDim 17472
GetDataAsString 17473
GetDataAsVariant 17473
GetDimensionMemberCount 17474
GetDimensionName 17474
GetGroup 17475
GetGroupCount 17475
GetGroupExtent 17476
GetIndex 17477
GetMemberAsString 17477
GetMemberAsVariant 17478
GetRowState 17478
GetState 17479
GetSummaryName 17479
GetValue 17480
GetValueArray 17480
GetValueIndex 17481
MoveDimIndexes 17481
Name 17487
nColDims 17489
nDataCols 17489
nDataRows 17490
nDims 17490
nOpenColDims 17491
nOpenRowDims 17491
nRowDims 17491
nSums 17492
OnAfterPivot 17494

OnBeforePivot 17494
OnLayoutChange 17495
OnNewDimensions 17495
OnStateChange 17496
OnSummaryChange 17496
OpenDimIndexLeft 17482
OpenDimIndexRight 17482
Ready 17492
SetCurrentSummary 17483
SetDimGroup 17483
SetDimState 17484
SparseCols 17493
SparseRows 17493
SwapDimIndexes 17484
TDecisionSource events 17493
TDecisionSource fields 17467
TDecisionSource Members 17463
TDecisionSource methods 17469
TDecisionSource properties 17485
ToggleDimIndex 17484
TDecompressionStream class 26627
about TDecompressionStream class 26627
Create 26628
Destroy 26629
OnProgress 26631
Read 26629
Seek 26630
TDecompressionStream events 26631
TDecompressionStream Members 26627
TDecompressionStream methods 26629
Write 26630
TDefaultEditor class 10967
about TDefaultEditor class 10967
Edit 10969
EditProperty 10970
TDefaultEditor Members 10968
TDefaultEditor methods 10969
TDefaultFieldsAdapter class 24392
about TDefaultFieldsAdapter class 24392
Actions 24402
AddDefaultActions 24399
AddDefaultFields 24399
Data 24402
DefaultWebComponentsChange 24400
GetActionsClass 24400
GetDataClass 24400
ImplGetVisibleActions 24401
ImplGetVisibleFields 24401
ImplSetActions 24401
ImplSetData 24402
TDefaultFieldsAdapter Members 24393
TDefaultFieldsAdapter methods 24399
TDefaultFieldsAdapter properties 24402
TDefaultFieldsPagedAdapter class 24403
about TDefaultFieldsPagedAdapter class 24403
AddDefaultActions 24411
CurrentPage 24418
EncodeActionParams 24411
EndOfPage 24412
ExtractRequestParams 24412
GetEOF 24412
GetFirstRecord 24413
GetNextRecord 24413
GetPageCount 24413
GetRecordCount 24414
GetRecordIndex 24414
ImplCanAddActionClass 24415
ImplEndIterator 24415
ImplGetActionsList 24415
ImplNextIteration 24416
ImplNotifyDeactivate 24416
ImplStartIterator 24416
OnGetEOF 24420
OnGetFirstRecord 24420
OnGetNextRecord 24421
OnGetRecordCount 24421
OnGetRecordIndex 24421
PageCount 24418

PageSize 24419
RecordCount 24419
RecordIndex 24419
SetPageSize 24417
SkipToPage 24417
TDefaultFieldsPagedAdapter events 24420
TDefaultFieldsPagedAdapter Members 24404
TDefaultFieldsPagedAdapter methods 24410
TDefaultFieldsPagedAdapter properties 24418
TDefaultSessionSprig class 1738
about TDefaultSessionSprig class 1738
Caption 1739
ItemClass 1739
TDefaultSessionSprig Members 1738
TDefaultSessionSprig methods 1738
UniqueName 1739
TDefaultWebAppServices class 15125
about TDefaultWebAppServices class 15125
FindRequestHandler 15127
FinishContext 15127
GetExceptionHandler 15127
GetRequestHandler 15128
HandleRequest 15128
InitContext 15128
InvokeDispatcher 15129
Request 15129
RequestHandler 15130
Response 15130
TDefaultWebAppServices Members 15125
TDefaultWebAppServices methods 15127
TDefaultWebAppServices properties 15129
WebModules 15130
TDefCollection class 7104
about TDefCollection class 7104
Create 7106
DataSet 7110
DoUpdate 7107
Find 7107
GetItemNames 7108
GetItemNames method 7108
IndexOf 7108
OnUpdate 7111
SetItemName 7109
TDefCollection events 7111
TDefCollection Members 7104
TDefCollection methods 7107
TDefCollection properties 7110
Update 7109
Updated 7110
UpdateDefs 7109
TDefinition class 25667
about TDefinition class 25667
AfterConstruction 25673
GetBindings 25673
GetImports 25673
GetMessages 25674
GetName 25674
GetPortTypes 25674
GetServices 25674
GetTargetNamespace 25675
GetTypes 25675
SetName 25675
SetTargetNamespace 25675
TDefinition Members 25667
TDefinition methods 25672
TDeleteButton class 16586
about TDeleteButton class 16586
Create 16591
TDeleteButton Members 16587
TDependencies class 22001
about TDependencies class 22001
Create 22003
GetOwner 22004
Items 22004
TDependencies Members 22001
TDependencies methods 22003
TDependencies properties 22004
TDependency class 22005

about TDependency class 22005
GetDisplayName 22007
IsGroup 22008
Name 22008
TDependency Members 22006
TDependency methods 22007
TDependency properties 22007
TDesignerOleDropTarget class 10970
about TDesignerOleDropTarget class 10970
Create 10972
DragEnter 10972
DragLeave 10972
DragOver 10973
Drop 10973
TDesignerOleDropTarget Members 10971
TDesignerOleDropTarget methods 10972
TDesignerSelections class 11121
about TDesignerSelections class 11121
Add 11124
Copy 11123
Count 11126
Create 11123
Destroy 11124
Equals 11125
Get 11125
GetCount 11125
GetDesignObject 11126
Items 11126
TDesignerSelections Members 11121
TDesignerSelections methods 11124
TDesignerSelections properties 11126
TDesigntimeWarnings class 25297
about TDesigntimeWarnings class 25297
AddObject 25298
AddString 25299
Create 25298
Destroy 25298
Strings 25299
TDesigntimeWarnings Members 25297
TDesigntimeWarnings methods 25298
TDesigntimeWarnings properties 25299
TDetailDataLink class 7111
about TDetailDataLink class 7111
DetailDataSet 7114
GetDetailDataSet 7114
TDetailDataLink Members 7112
TDetailDataLink methods 7113
TDetailDataLink properties 7114
TDimension class 17809
about TDimension class 17809
AssignSorted 17811
Attributes 17813
ClearFlag 17811
Create 17810
Destroy 17811
DimensionName 17813
FieldDefinition 17813
FieldName 17813
FieldType 17814
HasFlag 17812
IsString 17812
Position 17814
Range 17814
SetFlag 17812
SetRangeCounting 17812
TDimension Members 17809
TDimension methods 17811
TDimension properties 17812
TDimensionItem class 17405
about TDimensionItem class 17405
Active 17411
ActiveFlag 17411
Assign 17408
BaseName 17411
Create 17408
DerivedFrom 17412
DimensionType 17412
Fieldname 17413

FieldType 17413
Format 17414
InitializeRange 17409
IsDimension 17409
IsSummary 17410
Name 17414
NotifyCollection 17410
Owner 17414
TDimensionItem Members 17406
TDimensionItem methods 17408
TDimensionItem properties 17410
TDimensionItems class 17415
about TDimensionItems class 17415
ActiveDimensionCount 17421
ActiveSummaryCount 17421
Add 17418
AddDerivedField 17418
AlreadyExists 17419
Assign 17419
AverageFieldCheck 17419
Create 17417
DimensionCount 17422
GetOwner 17420
Items 17422
NotifyOwner 17420
SummaryCount 17423
TDimensionItems Members 17415
TDimensionItems methods 17417
TDimensionItems properties 17421
TDimInfoArray class 17497
about TDimInfoArray class 17497
Assign 17498
Create 17498
Destroy 17498
Find 17499
GetGroupArray 17499
GetGroupIndex 17499
GetGroupItem 17499
GetGroupSize 17499
GetItem 17500
IsEqual 17500
Items 17500
Limit 17500
TDimInfoArray Members 17497
TDimInfoArray methods 17498
TDimInfoArray properties 17500
TDirectoryListBox class 13313
about TDirectoryListBox class 13313
Align 13336
Anchors 13339
AutoComplete 13340
BevelEdges 13340
BevelInner 13340
BevelKind 13341
BevelOuter 13341
BuildList 13329
CaseSensitive 13342
Change 13329
ClosedBMP 13327
Color 13342
Columns 13343
Constraints 13343
Create 13326
CreateWnd 13329
Ctl3D 13344
CurrentBMP 13327
DblClick 13330
Destroy 13326
Directory 13344
DirLabel 13346
DisplayCase 13330
DragCursor 13347
DragMode 13347
DrawItem 13330
Drive 13349
Enabled 13350
FDirectory 13327
FileCompareText 13331

FileList 13350
FOnChange 13328
Font 13353
GetItemPath 13331
ImeMode 13354
ImeName 13354
IntegralHeight 13355
ItemHeight 13355
KeyPress 13331
Notification 13332
OnChange 13367
OnClick 13369
OnContextPopup 13371
OnDblClick 13371
OnDragDrop 13372
OnDragOver 13374
OnEndDrag 13376
OnEnter 13377
OnExit 13380
OnKeyDown 13381
OnKeyPress 13384
OnKeyUp 13385
OnMouseActivate 13387
OnMouseDown 13388
OnMouseEnter 13390
OnMouseLeave 13391
OnMouseMove 13391
OnMouseUp 13393
OnStartDrag 13395
OpenCurrent 13332
OpenedBMP 13328
ParentColor 13356
ParentCtl3D 13356
ParentFont 13357
ParentShowHint 13357
PopupMenu 13358
PreserveCase 13362
ReadBitmaps 13333
ReadDirectoryNames 13333
ShowHint 13363
TabOrder 13364
TabStop 13365
TDirectoryListBox events 13367
TDirectoryListBox fields 13327
TDirectoryListBox Members 13314
TDirectoryListBox methods 13328
TDirectoryListBox properties 13335
Update 13333
Visible 13365
TDispatchAppServer class 16015
about TDispatchAppServer class 16015
Create 16017
SafeCallException 16017
TDispatchAppServer Members 16015
TDispatchAppServer methods 16017
TDispatchConnection class 16017
about TDispatchConnection class 16017
AfterConnect 16031
AfterDisconnect 16031
BeforeConnect 16032
BeforeDisconnect 16032
Connected 16026
Create 16022
DoDisconnect 16023
GetConnected 16023
GetServer 16023
GetServerCLSID 16024
GetServerList 16024
LoginPrompt 16027
Notification 16025
ObjectBroker 16028
OnGetUsername 16033
OnLogin 16028
ServerGUID 16030
ServerName 16030
SetConnected 16025
TDispatchConnection events 16031
TDispatchConnection Members 16018

TDispatchConnection methods 16022
TDispatchConnection properties 16026
TDisplayDim class 17661
about TDisplayDim class 17661
Alignment 17664
assign 17663
Color 17664
Create 17662
DisplayName 17665
FieldName 17665
Format 17665
Subtotals 17666
TDisplayDim Members 17661
TDisplayDim methods 17663
TDisplayDim properties 17664
TDisplayDims class 17666
about TDisplayDims class 17666
FGrid 17669
GetOwner 17669
Items 17670
NotifyOwner 17670
TDisplayDims fields 17669
TDisplayDims Members 17667
TDisplayDims methods 17669
TDisplayDims properties 17670
tdMetaData enumeration member 17672
TDockCaptionDrawer class 2405
about TDockCaptionDrawer class 2405
Create 2406
DockCaptionHitTest 2407
DockCaptionPinButton 2408
DrawDockCaption 2407
TDockCaptionDrawer Members 2406
TDockCaptionDrawer methods 2407
TDockCaptionDrawer properties 2407
TDockClientInfo class 11598
about TDockClientInfo class 11598
Create 11601
CreateTab 11602
Destroy 11601
DockClient 11603
DockTabSet 11603
ImageIndex 11603
ListIndex 11603
Notification 11602
ParentPanel 11604
TabIndex 11604
TDockClientInfo Members 11598
TDockClientInfo methods 11602
TDockClientInfo properties 11603
TDockTabSet class 11604
about TDockTabSet class 11604
AutoSelect 11622
Create 11616
DestinationDockSite 11623
Destroy 11617
DoAddDockClient 11617
DockOver 11618
DockReplaceDockClient 11619
DockSite 11623
DoRemoveDockClient 11619
FindDockClientInfo 11619
GetImageIndex 11620
GetTabName 11620
HideCurrentDockClient 11620
Images 11623
IndexOfDockClient 11620
MouseMove 11620
Notification 11621
OnDockDrop 11624
OnDockOver 11624
OnGetImageIndex 11624
OnGetSiteInfo 11625
OnStartDock 11625
OnTabAdded 11625
OnTabRemoved 11625
OnUnDock 11625
SetTabIndex 11621

ShouldAutoHide 11622
ShowDockClient 11622
ShowDockClientInfo 11622
TDockTabSet events 11624
TDockTabSet Members 11604
TDockTabSet methods 11617
TDockTabSet properties 11622
TDockTree class 5983
about TDockTree class 5983
AdjustDockRect 5987
AdjustFrameRect 5988
BeginUpdate 5988
Create 5985
Destroy 5986
DockSite 5996
EndUpdate 5988
FindControlAtPos 5989
GetControlBounds 5989
HitTest 5990
InsertControl 5990
LoadFromStream 5991
MouseDown 5991
MouseMove 5991
MouseUp 5992
PaintDockFrame 5992
PaintSite 5992
PositionDockRect 5993
ReferenceFromOrient 5993
RelativeSizes 5997
RemoveControl 5993
ResetBounds 5994
SaveToStream 5994
SetReplacingControl 5995
ShowHint 5995
TDockTree Members 5984
TDockTree methods 5987
TDockTree properties 5996
TopZone 5997
UpdateAll 5995
WndProc 5996
ZoneCaptionHitTest 5996
TDockZone class 5997
about TDockZone class 5997
ChildControl 6001
ChildCount 6001
Create 5998
ExpandZoneLimit 5999
FirstVisibleChild 5999
Height 6001
Left 6001
LimitBegin 6002
LimitSize 6002
NextVisible 5999
PrevVisible 5999
ResetChildren 6000
ResetZoneLimits 6000
TDockZone Members 5998
TDockZone methods 5998
TDockZone properties 6000
Top 6002
Update 6000
Visible 6002
VisibleChildCount 6002
Width 6003
ZoneLimit 6003
TDocumentation class 25676
about TDocumentation class 25676
TDocumentationCollection class 25676
about TDocumentationCollection class 25676
Add 25682
GetDocumentation 25682
Insert 25682
TDocumentationCollection Members 25676
TDocumentationCollection methods 25682
TDocumented class 25683
about TDocumented class 25683
AfterConstruction 25688
GetDocumentationCollection 25688

TDocumented Members 25683
TDocumented methods 25687
TDOMVendor class 26263
about TDOMVendor class 26263
Description 26264
DOMImplementation 26264
TDOMVendor Members 26263
TDOMVendor methods 26263
TDOMVendorList class 26265
about TDOMVendorList class 26265
Add 26266
Count 26266
Find 26267
GetVendors 26267
Remove 26268
TDOMVendorList Members 26265
TDOMVendorList methods 26266
TDOMVendorList properties 26268
Vendors 26268
TDoubleArray class 17337
about TDoubleArray class 17337
Add 17340
Assign 17340
Create 17339
Find 17341
GetItem 17341
IndexOf 17341
Items 17342
PutItem 17341
TDoubleArray Members 17337
TDoubleArray methods 17339
TDoubleArray properties 17342
TDownLoadURL class 11857
about TDownLoadURL class 11857
AfterDownload 11870
BeforeDownload 11870
Caption 11865
Enabled 11865
ExecuteTarget 11862
Filename 11866
GetBindInfo 11863
GetPriority 11863
HelpContext 11866
HelpKeyword 11866
HelpType 11867
Hint 11867
ImageIndex 11868
OnDataAvailable 11863
OnDownloadProgress 11870
OnHint 11871
OnLowResource 11863
OnObjectAvailable 11864
OnProgress 11864
OnStartBinding 11864
OnStopBinding 11864
OnUpdate 11871
SecondaryShortCuts 11868
ShortCut 11869
TDownLoadURL events 11870
TDownLoadURL Members 11857
TDownLoadURL methods 11862
TDownLoadURL properties 11864
URL 11869
Visible 11869
TDragControlObject class 6003
about TDragControlObject class 6003
GetDragCursor 6005
GetDragImages 6006
HideDragImage 6006
ShowDragImage 6006
TDragControlObject Members 6004
TDragControlObject methods 6005
TDragControlObjectEx class 6007
about TDragControlObjectEx class 6007
BeforeDestruction 6009
TDragControlObjectEx Members 6007
TDragControlObjectEx methods 6009
TDragDockObject class 6009

about TDragDockObject class 6009
AdjustDockRect 6012
Assign 6012
Brush 6015
Create 6011
Destroy 6012
DockRect 6015
DrawDragDockImage 6013
DropAlign 6017
DropOnControl 6017
EndDrag 6013
EraseDockRect 6017
EraseDragDockImage 6013
EraseWhenMoving 6018
Floating 6018
FrameWidth 6018
GetDragCursor 6014
GetEraseWhenMoving 6014
GetFrameWidth 6014
TDragDockObject Members 6010
TDragDockObject methods 6012
TDragDockObject properties 6015
TDragDockObjectEx class 6018
about TDragDockObjectEx class 6018
BeforeDestruction 6021
TDragDockObjectEx Members 6019
TDragDockObjectEx methods 6021
TDragImageList class 6021
about TDragImageList class 6021
BeginDrag 6026
DragCursor 6030
Dragging 6030
DragHotspot 6030
DragLock 6026
DragMove 6027
DragUnlock 6027
EndDrag 6028
GetHotSpot 6028
HideDragImage 6028
Initialize 6028
SetDragImage 6029
ShowDragImage 6029
TDragImageList Members 6022
TDragImageList methods 6026
TDragImageList properties 6029
TDragObject class 6031
about TDragObject class 6031
AfterConstruction 6033
AlwaysShowDragImages 6038
Assign 6034
BeforeDestruction 6034
Cancelling 6038
DragHandle 6039
DragPos 6039
DragTarget 6039
DragTargetPos 6040
Dropped 6040
Finished 6035
GetDragCursor 6035
GetDragImages 6035
GetName 6036
HideDragImage 6036
Instance 6037
MainWndProc 6037
MouseDeltaX 6040
MouseDeltaY 6041
RightClickCancels 6041
ShowDragImage 6037
TDragObject Members 6032
TDragObject methods 6033
TDragObject properties 6038
WndProc 6038
TDragObjectEx class 6041
about TDragObjectEx class 6041
BeforeDestruction 6043
TDragObjectEx Members 6042
TDragObjectEx methods 6043
TDragTarget class 11127

about TDragTarget class 11127
Create 11128
Designer 11129
DragDrop 11128
DragOver 11128
TDragTarget Members 11127
TDragTarget methods 11128
TDragTarget properties 11129
TDrawGrid class 14860
about TDrawGrid class 14860
Align 14874
Anchors 14878
BevelEdges 14879
BevelInner 14879
BevelKind 14879
BevelOuter 14880
BevelWidth 14880
BiDiMode 14881
BorderStyle 14881
ColCount 14881
Color 14882
Constraints 14883
Ctl3D 14884
DefaultColWidth 14884
DefaultDrawing 14885
DefaultRowHeight 14885
DragCursor 14886
DragKind 14886
DragMode 14887
Enabled 14888
FixedColor 14889
FixedCols 14889
FixedRows 14889
Font 14890
GridLineWidth 14891
OnClick 14906
OnColumnMoved 14908
OnContextPopup 14908
OnDblClick 14909
OnDragDrop 14909
OnDragOver 14911
OnDrawCell 14914
OnEndDock 14914
OnEndDrag 14915
OnEnter 14916
OnExit 14918
OnGetEditMask 14920
OnGetEditText 14920
OnKeyDown 14920
OnKeyPress 14923
OnKeyUp 14924
OnMouseActivate 14926
OnMouseDown 14927
OnMouseEnter 14929
OnMouseLeave 14929
OnMouseMove 14930
OnMouseUp 14932
OnMouseWheelDown 14934
OnMouseWheelUp 14935
OnRowMoved 14935
OnSelectCell 14935
OnSetEditText 14936
OnStartDock 14936
OnStartDrag 14937
OnTopLeftChanged 14937
Options 14892
ParentBiDiMode 14893
ParentColor 14893
ParentCtl3D 14893
ParentFont 14894
ParentShowHint 14894
PopupMenu 14895
RowCount 14899
ScrollBars 14900
ShowHint 14901
TabOrder 14902
TDrawGrid events 14905
TDrawGrid Members 14861

TDrawGrid properties 14874
Visible 14903
VisibleColCount 14904
VisibleRowCount 14905
TDriveComboBox class 13396
about TDriveComboBox class 13396
Align 13414
Anchors 13417
AutoComplete 13418
AutoDropDown 13418
BevelEdges 13419
BevelInner 13419
BevelKind 13420
BevelOuter 13420
BuildList 13411
CDROMBMP 13410
Click 13411
Color 13420
Constraints 13421
Create 13409
CreateWnd 13412
Ctl3D 13422
Destroy 13409
DirList 13422
DragCursor 13424
DragMode 13424
DrawItem 13412
Drive 13426
Enabled 13427
FixedBMP 13410
FloppyBMP 13410
Font 13427
ImeMode 13428
ImeName 13429
NetworkBMP 13410
Notification 13413
OnChange 13442
OnClick 13444
OnContextPopup 13446
OnDblClick 13446
OnDragDrop 13447
OnDragOver 13449
OnDropDown 13451
OnEndDrag 13453
OnEnter 13454
OnExit 13456
OnKeyDown 13458
OnKeyPress 13461
OnKeyUp 13462
OnMouseEnter 13464
OnMouseLeave 13464
OnStartDrag 13464
ParentColor 13429
ParentCtl3D 13430
ParentFont 13430
ParentShowHint 13431
PopupMenu 13431
RAMBMP 13411
ShowHint 13436
TabOrder 13437
TabStop 13438
TDriveComboBox events 13441
TDriveComboBox fields 13409
TDriveComboBox Members 13397
TDriveComboBox methods 13411
TDriveComboBox properties 13413
Text 13438
TextCase 13439
Visible 13440
TDSTableProducer class 10290
about TDSTableProducer class 10290
AcquireLayoutLock 10296
BeginLayout 10297
BeginUpdate 10297
Caption 10307
CaptionAlignment 10308
ColumnHeader 10298
Columns 10308

Create 10295
DataLink 10309
DataSet 10309
DataSource 10309
DefineFieldMap 10298
Destroy 10295
DoCreateContent 10299
DoFormatCell 10299
DoGetCaption 10300
Editor 10310
EndLayout 10300
EndUpdate 10301
Footer 10310
FormatCell 10301
GetDataSet 10302
GetDataSource 10302
Header 10311
InternalDataSource 10311
LayoutChanged 10303
LinkActive 10303
MaxRows 10312
Notification 10304
OnCreateContent 10313
OnFormatCell 10314
OnGetTableCaption 10314
RowAttributes 10312
RowHeader 10304
SetColumns 10304
SetDataSet 10305
SetDataSource 10305
StoreColumns 10305
TableAttributes 10313
TableCaption 10306
TableHeader 10306
TDSTableProducer events 10313
TDSTableProducer Members 10291
TDSTableProducer methods 10296
TDSTableProducer properties 10307
TDSTableProducerEditor class 10315
about TDSTableProducerEditor class 10315
Changed 10317
Create 10316
DataSource 10318
Destroy 10316
DSTableProducer 10318
PostChange 10317
TDSTableProducerEditor Members 10315
TDSTableProducerEditor methods 10317
TDSTableProducerEditor properties 10318
tdSubTotals enumeration member 17672
TEdit class 21402
about TEdit class 21402
Align 21415
Anchors 21418
AutoSelect 21419
AutoSize 21419
BevelEdges 21420
BevelInner 21421
BevelKind 21421
BevelOuter 21422
BevelWidth 21422
BiDiMode 21423
BorderStyle 21423
CharCase 21423
Color 21424
Constraints 21425
Ctl3D 21425
DragCursor 21426
DragKind 21426
DragMode 21426
Enabled 21428
Font 21428
HideSelection 21429
ImeMode 21430
ImeName 21430
MaxLength 21431
OEMConvert 21432
OnChange 21445

wwww

OnClick 21446
OnContextPopup 21448
OnDblClick 21448
OnDragDrop 21449
OnDragOver 21451
OnEndDock 21453
OnEndDrag 21454
OnEnter 21455
OnExit 21457
OnKeyDown 21459
OnKeyPress 21462
OnKeyUp 21463
OnMouseActivate 21464
OnMouseDown 21465
OnMouseEnter 21468
OnMouseLeave 21468
OnMouseMove 21468
OnMouseUp 21470
OnStartDock 21473
OnStartDrag 21473
ParentBiDiMode 21432
ParentColor 21432
ParentCtl3D 21433
ParentFont 21433
ParentShowHint 21434
PasswordChar 21434
PopupMenu 21435
ReadOnly 21439
ShowHint 21440
TabOrder 21441
TabStop 21442
TEdit events 21445
TEdit Members 21403
TEdit properties 21414
Text 21442
Visible 21443
TEditAction class 20544
about TEditAction class 20544
Control 20551
Destroy 20549
GetControl 20549
HandlesTarget 20550
Notification 20550
TEditAction Members 20545
TEditAction methods 20549
TEditAction properties 20551
UpdateTarget 20551
TEditActionSelectionEditor class 10973
about TEditActionSelectionEditor class 10973
EditAction 10975
GetEditState 10976
GetVerb 10976
GetVerbCount 10976
HandleCopy 10976
HandleCut 10977
HandleDelete 10977
HandlePaste 10977
HandleSelectAll 10977
HandleUndo 10977
PrepareItem 10978
TEditActionSelectionEditor Members 10974
TEditActionSelectionEditor methods 10975
TEditCopy class 20552
about TEditCopy class 20552
ExecuteTarget 20557
TEditCopy Members 20552
TEditCopy methods 20557
TEditCut class 20557
about TEditCut class 20557
ExecuteTarget 20563
TEditCut Members 20558
TEditCut methods 20563
TEditDelete class 20563
about TEditDelete class 20563
ExecuteTarget 20569
TEditDelete Members 20564
TEditDelete methods 20568
UpdateTarget 20569

TEditPaste class 20570
about TEditPaste class 20570
ExecuteTarget 20575
TEditPaste Members 20570
TEditPaste methods 20575
UpdateTarget 20575
TEditSelectAll class 20576
about TEditSelectAll class 20576
ExecuteTarget 20581
TEditSelectAll Members 20577
TEditSelectAll methods 20581
UpdateTarget 20582
TEditUndo class 20582
about TEditUndo class 20582
ExecuteTarget 20587
TEditUndo Members 20583
TEditUndo methods 20587
UpdateTarget 20588
TEndUserAdapter class 24421
about TEndUserAdapter class 24421
Actions 24431
Data 24431
LoginPage 24431
OnAfterExecuteAction 24432
OnAfterGetActionResponse 24432
OnBeforeExecuteAction 24433
OnBeforeGetActionResponse 24433
OnGetActionParams 24433
OnGetDisplayName 24434
OnHasRights 24434
OnIsLoggedIn 24434
OnLogin 24435
OnLogout 24436
OnProduceLoginPage 24436
TEndUserAdapter events 24431
TEndUserAdapter Members 24423
TEndUserAdapter properties 24430
TEndUserAdapterLoginFormAction class 24436
about TEndUserAdapterLoginFormAction class 24436
DisplayLabel 24444
GetDefaultActionName 24443
ImplExecuteActionRequest 24443
ImplWebEnabled 24444
OnAfterExecute 24445
OnAfterGetResponse 24445
OnBeforeExecute 24446
OnBeforeGetResponse 24446
OnGetParams 24447
TEndUserAdapterLoginFormAction events 24445
TEndUserAdapterLoginFormAction Members 24437
TEndUserAdapterLoginFormAction methods 24443
TEndUserAdapterLoginFormAction properties 24444
TEndUserAdapterLogoutAction class 24447
about TEndUserAdapterLogoutAction class 24447
DisplayLabel 24455
GetDefaultActionName 24454
ImplExecuteActionRequest 24454
ImplWebEnabled 24455
OnAfterExecute 24456
OnAfterGetResponse 24456
OnBeforeExecute 24457
OnBeforeGetResponse 24457
OnGetParams 24458
TEndUserAdapterLogoutAction events 24456
TEndUserAdapterLogoutAction Members 24448
TEndUserAdapterLogoutAction methods 24454
TEndUserAdapterLogoutAction properties 24455
TEndUserSessionAdapter class 24458
about TEndUserSessionAdapter class 24458
Actions 24468
Data 24468
LoginPage 24468
OnAfterExecuteAction 24469
OnAfterGetActionResponse 24469
OnBeforeExecuteAction 24470
OnBeforeGetActionResponse 24470
OnGetActionParams 24470
OnHasRights 24471

OnProduceLoginPage 24471
TEndUserSessionAdapter events 24469
TEndUserSessionAdapter Members 24460
TEndUserSessionAdapter properties 24467
TEnumActionFieldValueRecordsImpl class 904
about TEnumActionFieldValueRecordsImpl class 904
AdapterName 908
Create 905
CurrentFieldValues 909
Destroy 905
EndIterator 906
Eof 909
FindPopulatedRecord 907
GetEOF 907
GetFirstRecord 907
GetInIterator 907
GetNextRecord 907
GetRecord 908
GetRecordCount 908
TEnumActionFieldValueRecordsImpl Members 904
TEnumActionFieldValueRecordsImpl methods 906
TEnumActionFieldValueRecordsImpl properties 908
TEnumPropDesc class 18023
about TEnumPropDesc class 18023
Create 18024
Destroy 18025
GetStrings 18025
StringToValue 18026
TEnumPropDesc Members 18024
TEnumPropDesc methods 18025
ValueToString 18026
TEnumProperty class 10978
about TEnumProperty class 10978
GetAttributes 10981
GetValue 10982
GetValues 10982
SetValue 10983
TEnumProperty Members 10978
TEnumProperty methods 10981
TEvent class 22063
about TEvent class 22063
Create 22065
Create constructor 22064
ResetEvent 22066
SetEvent 22066
TEvent Members 22063
TEvent methods 22065
TEventDispatch class 18026
_AddRef 18028
_Release 18028
about TEventDispatch class 18026
Control 18030
Create 18028
GetIDsOfNames 18029
GetTypeInfo 18029
GetTypeInfoCount 18029
Invoke 18029
QueryInterface 18030
TEventDispatch Members 18027
TEventDispatch methods 18028
TEventDispatch properties 18030
TEventInfo class 11129
about TEventInfo class 11129
Create 11131
Create constructor 11131
GetMethodKind 11132
GetParamCount 11132
GetParamFlags 11132
GetParamName 11132
GetParamType 11132
GetResultType 11133
TEventInfo Members 11129
TEventInfo methods 11131
TEventLogger class 22009
about TEventLogger class 22009
Create 22010
Destroy 22010
LogMessage 22011

TEventLogger Members 22009
TEventLogger methods 22010
tfAllowDialogCancellation enumeration member 11566
TFault class 25689
about TFault class 25689
TFaults class 25689
about TFaults class 25689
Add 25695
GetFault 25695
Insert 25695
TFaults Members 25690
TFaults methods 25695
TFavoriteLinkItem class 11337
about TFavoriteLinkItem class 11337
GetDisplayName 11339
Location 11339
TFavoriteLinkItem Members 11337
TFavoriteLinkItem methods 11339
TFavoriteLinkItem properties 11339
TFavoriteLinkItems class 11340
about TFavoriteLinkItems class 11340
Add 11342
GetEnumerator 11342
Items 11343
TFavoriteLinkItems Members 11340
TFavoriteLinkItems methods 11342
TFavoriteLinkItems properties 11343
TFavoriteLinkItemsEnumerator class 11343
about TFavoriteLinkItemsEnumerator class 11343
Create 11344
Current 11345
GetCurrent 11344
MoveNext 11345
TFavoriteLinkItemsEnumerator Members 11344
TFavoriteLinkItemsEnumerator methods 11344
TFavoriteLinkItemsEnumerator properties 11345
tfBlob enumeration member 10256
tfBottom enumeration member 14642
tfBytes enumeration member 17235
tfCalcRect enumeration member 14642
tfCallbackTimer enumeration member 11566
tfCanBeMinimized enumeration member 11566
tfCenter enumeration member 14642
tfConnect enumeration member 10256
tfDataIn enumeration member 10256
tfDataOut enumeration member 10256
tfEditControl enumeration member 14642
tfEnableHyperlinks enumeration member 11566
tfEndEllipsis enumeration member 14642
tfError enumeration member 10256
tfExpandedByDefault enumeration member 11566
tfExpandFooterArea enumeration member 11566
tfExpandTabs enumeration member 14642
tfExternalLeading enumeration member 14642
tfFrames enumeration member 17235
tfHMS enumeration member 17235
TField class 7115
about TField class 7115
AccessError 7124
Alignment 7146
AsBCD 7147
AsBoolean 7147
AsCurrency 7148
AsDateTime 7148
AsFloat 7149
AsInteger 7149
Assign 7124
AssignValue 7124
AsSQLTimeStamp 7150
AsString 7151
AsVariant 7152
AsWideString 7152
AttributeSet 7153
AutoGenerateValue 7153
Bind 7126
Calculated 7154
CanModify 7154
Change 7126

CheckInactive 7126
CheckTypeSize 7126
Clear 7126
ConstraintErrorMessage 7155
CopyData 7127
Create 7121
CurValue 7155
CustomConstraint 7156
DataChanged 7127
DataSet 7157
DataSize 7157
DataType 7159
DefaultExpression 7159
DefineProperties 7127
Destroy 7122
DisplayLabel 7160
DisplayName 7160
DisplayText 7162
DisplayWidth 7162
EditMask 7163
EditMaskPtr 7163
FieldKind 7164
FieldName 7164
FieldNo 7166
FocusControl 7128
FreeBuffers 7128
FullName 7166
GetAsBCD 7128
GetAsBoolean 7129
GetAsByteArray 7129
GetAsCurrency 7129
GetAsDateTime 7129
GetAsFloat 7129
GetAsInteger 7130
GetAsSQLTimeStamp 7130
GetAsString 7130
GetAsVariant 7130
GetAsWideString 7131
GetCanModify 7131
GetClassDesc 7131
GetData 7131
GetDataSize 7133
GetDefaultWidth 7133
GetFieldNo 7134
GetHasConstraints 7134
GetIsNull 7134
GetParentComponent 7134
GetSize 7135
GetText 7135
GetWideText 7135
HasConstraints 7167
HasParent 7135
ImportedConstraint 7167
Index 7168
IsBlob 7136
IsIndexField 7168
IsNull 7169
IsValidChar 7136
KeyFields 7169
Lookup 7170
LookupCache 7170
LookupDataSet 7171
LookupKeyFields 7172
LookupList 7172
LookupResultField 7173
NewValue 7174
Notification 7137
Offset 7175
OldValue 7175
OnChange 7181
OnGetText 7182
OnSetText 7182
OnValidate 7183
Origin 7176
ParentField 7176
PropertyChanged 7138
ProviderFlags 7176
ReadOnly 7177

ReadState 7138
RefreshLookupList 7138
Required 7177
SetAsBCD 7139
SetAsBoolean 7139
SetAsByteArray 7139
SetAsCurrency 7139
SetAsDateTime 7139
SetAsFloat 7140
SetAsInteger 7140
SetAsSQLTimeStamp 7140
SetAsString 7140
SetAsVariant 7141
SetAsWideString 7141
SetAutoGenerateValue 7141
SetData 7141
SetDataSet 7142
SetDataType 7142
SetFieldKind 7142
SetFieldType 7142
SetParentComponent 7143
SetParentField 7143
SetSize 7143
SetText 7144
SetVarValue 7144
SetWideText 7144
Size 7178
Text 7179
TField events 7181
TField Members 7116
TField methods 7122
TField properties 7145
Validate 7144
ValidChars 7179
Value 7180
Visible 7181
TFieldDataLink class 9487
about TFieldDataLink class 9487
ActiveChanged 9490
CanModify 9494
Control 9495
Create 9489
DataEvent 9491
Edit 9491
Editing 9495
EditingChanged 9491
Field 9496
FieldName 9496
FocusControl 9492
LayoutChanged 9492
Modified 9492
OnActiveChange 9497
OnDataChange 9497
OnEditingChange 9497
OnUpdateData 9498
RecordChanged 9493
Reset 9493
TFieldDataLink events 9496
TFieldDataLink Members 9487
TFieldDataLink methods 9490
TFieldDataLink properties 9494
UpdateData 9494
TFieldDef class 7184
about TFieldDef class 7184
AddChild 7187
Assign 7188
Attributes 7190
ChildDefs 7191
Create 7186
CreateField 7188
DataType 7191
DefineProperties 7189
Destroy 7187
FieldClass 7192
FieldNo 7192
HasChildDefs 7189
InternalCalcField 7192
ParentDef 7193

Precision 7193
Required 7194
Size 7194
TFieldDef Members 7185
TFieldDef methods 7187
TFieldDef properties 7190
TFieldDefinition class 17814
about TFieldDefinition class 17814
Create 17815
DisplayName 17816
FieldNo 17817
FieldType 17817
FormatString 17817
FormatVariantToStr 17815
Precision 17817
SetFieldType 17816
SetName 17816
TFieldDefinition Members 17815
TFieldDefinition methods 17815
TFieldDefinition properties 17816
Width 17817
TFieldDefList class 7195
about TFieldDefList class 7195
FieldByName 7196
FieldDefs 7198
Find 7197
GetUpdated 7197
TFieldDefList Members 7195
TFieldDefList methods 7196
TFieldDefList properties 7197
UpdateList 7197
TFieldDefs class 7198
about TFieldDefs class 7198
Add 7202
AddFieldDef 7202
ChildDefUpdate 7204
Create 7201
FieldDefUpdate 7204
Find 7204
HiddenFields 7206
Items 7206
ParentDef 7207
SetItemName 7205
TFieldDefs Members 7199
TFieldDefs methods 7201
TFieldDefs properties 7206
Update 7205
TFieldGroup class 16591
about TFieldGroup class 16591
Custom 16598
ImplAddElements 16597
ImplCanAddClass 16597
Style 16598
StyleRule 16599
TFieldGroup Members 16592
TFieldGroup methods 16597
TFieldGroup properties 16598
TFieldList class 7207
about TFieldList class 7207
FieldByName 7209
Fields 7210
Find 7209
TFieldList Members 7208
TFieldList methods 7209
TFieldList properties 7210
UpdateList 7210
TFieldRadioGroup class 16599
about TFieldRadioGroup class 16599
Caption 16606
CaptionAttributes 16606
CaptionPosition 16606
Custom 16607
DataSet 16607
DisplayColumns 16608
DisplayWidth 16608
FieldName 16608
Items 16609
ItemsField 16609

ReadOnly 16610
Style 16610
StyleRule 16611
TabIndex 16611
TFieldRadioGroup Members 16600
TFieldRadioGroup properties 16605
Values 16611
ValuesField 16612
TFields class 7211
about TFields class 7211
Add 7214
Changed 7214
CheckFieldKind 7215
CheckFieldName 7215
CheckFieldNames 7215
Clear 7216
Count 7220
Create 7213
DataSet 7221
Destroy 7213
FieldByName 7216
FieldByNumber 7217
Fields 7221
FindField 7217
GetCount 7217
GetEnumerator 7218
GetField 7218
GetFieldNames 7218
GetFieldNames method 7218
IndexOf 7219
OnChange 7222
Remove 7219
SetField 7220
SetFieldIndex 7220
SparseFields 7221
TFields events 7222
TFields Members 7211
TFields methods 7213
TFields properties 7220
ValidFieldKinds 7222
TFieldSelectOptions class 16612
about TFieldSelectOptions class 16612
Caption 16618
CaptionAttributes 16618
CaptionPosition 16619
Custom 16619
DataSet 16620
DisplayRows 16620
FieldName 16621
Items 16621
ItemsField 16621
Style 16622
StyleRule 16622
TabIndex 16623
TFieldSelectOptions Members 16613
TFieldSelectOptions properties 16618
Values 16623
ValuesField 16623
TFieldsEnumerator class 7222
about TFieldsEnumerator class 7222
Create 7223
Current 7224
GetCurrent 7224
MoveNext 7224
TFieldsEnumerator Members 7223
TFieldsEnumerator methods 7223
TFieldsEnumerator properties 7224
TFieldStatus class 16624
about TFieldStatus class 16624
Caption 16629
CaptionAttributes 16629
CaptionPosition 16630
Custom 16630
Style 16630
StyleRule 16631
TFieldStatus Members 16625
TFieldStatus properties 16629
TFieldText class 16631

eeee

about TFieldText class 16631
Caption 16637
CaptionAttributes 16637
CaptionPosition 16637
Custom 16638
DisplayWidth 16638
FieldName 16639
ReadOnly 16639
Style 16639
StyleRule 16640
TabIndex 16640
TFieldText Members 16632
TFieldText properties 16636
TFieldTextArea class 16640
about TFieldTextArea class 16640
Caption 16646
CaptionAttributes 16647
CaptionPosition 16647
Custom 16647
DisplayRows 16648
DisplayWidth 16648
FieldName 16649
ReadOnly 16649
Style 16649
StyleRule 16650
TabIndex 16650
TFieldTextArea Members 16641
TFieldTextArea properties 16646
Wrap 16650
TFieldValuesAdapter class 24471
about TFieldValuesAdapter class 24471
Create 24479
Field 24483
GetValueFieldClass 24480
ImplEndIterator 24480
ImplGetField 24481
ImplGetValueIndex 24481
ImplNextIteration 24481
ImplStartIterator 24482
TFieldValuesAdapter Members 24473
TFieldValuesAdapter methods 24479
TFieldValuesAdapter properties 24482
TFieldValuesValueField class 24483
about TFieldValuesValueField class 24483
Field 24490
GetDefaultFieldName 24489
ImplAsFormatted 24489
ImplGetValue 24490
TFieldValuesValueField Members 24484
TFieldValuesValueField methods 24489
TFieldValuesValueField properties 24490
TFileAction class 20588
about TFileAction class 20588
FileName 20593
GetDialog 20593
TFileAction Members 20589
TFileAction methods 20593
TFileAction properties 20593
TFileExit class 20594
about TFileExit class 20594
Caption 20600
Enabled 20600
ExecuteTarget 20598
HandlesTarget 20599
HelpContext 20600
HelpKeyword 20601
HelpType 20601
Hint 20602
ImageIndex 20602
OnHint 20604
OnUpdate 20604
SecondaryShortCuts 20602
ShortCut 20603
TFileExit events 20603
TFileExit Members 20594
TFileExit methods 20598
TFileExit properties 20599
Visible 20603

TFileListBox class 13465
about TFileListBox class 13465
Align 13488
Anchors 13492
ApplyFilePath 13482
AutoComplete 13492
BevelEdges 13493
BevelInner 13493
BevelKind 13493
BevelOuter 13494
Change 13484
Click 13484
Color 13494
Constraints 13495
Create 13478
CreateWnd 13484
Ctl3D 13496
Destroy 13479
DirBMP 13479
Directory 13496
DragCursor 13497
DragMode 13498
DrawItem 13485
Drive 13499
Enabled 13500
ExeBMP 13480
ExtendedSelect 13501
FDirectory 13480
FDirList 13480
FFileEdit 13480
FFileType 13480
FFilterCombo 13481
FileEdit 13501
FileName 13502
FileType 13503
FLastSel 13481
FMask 13481
FOnChange 13481
Font 13503
FShowGlyphs 13482
GetFilePath 13486
ImeMode 13504
ImeName 13505
IntegralHeight 13505
ItemHeight 13506
Mask 13506
MultiSelect 13507
Notification 13486
OnChange 13519
OnClick 13520
OnContextPopup 13521
OnDblClick 13522
OnDragDrop 13522
OnDragOver 13525
OnEndDrag 13527
OnEnter 13528
OnExit 13531
OnKeyDown 13532
OnKeyPress 13535
OnKeyUp 13536
OnMouseActivate 13538
OnMouseDown 13539
OnMouseEnter 13541
OnMouseLeave 13541
OnMouseMove 13541
OnMouseUp 13544
OnStartDrag 13546
ParentColor 13508
ParentCtl3D 13508
ParentFont 13509
ParentShowHint 13509
PopupMenu 13510
ReadBitmaps 13486
ReadFileNames 13487
ShowGlyphs 13514
ShowHint 13515
TabOrder 13516
TabStop 13517

TFileListBox events 13519
TFileListBox fields 13479
TFileListBox Members 13466
TFileListBox methods 13482
TFileListBox properties 13487
UnknownBMP 13482
Update 13487
Visible 13517
TFileOpen class 20605
about TFileOpen class 20605
BeforeExecute 20617
Caption 20612
Create 20610
Dialog 20612
Enabled 20612
ExecuteTarget 20610
GetDialogClass 20611
HelpContext 20613
HelpKeyword 20613
HelpType 20614
Hint 20614
ImageIndex 20615
OnAccept 20617
OnCancel 20617
OnHint 20617
OnUpdate 20618
SecondaryShortCuts 20615
ShortCut 20615
TFileOpen events 20616
TFileOpen Members 20606
TFileOpen methods 20610
TFileOpen properties 20611
UseDefaultApp 20616
Visible 20616
TFileOpenDialog class 11346
about TFileOpenDialog class 11346
ClientGuid 11350
DefaultExtension 11350
DefaultFolder 11351
FavoriteLinks 11351
FileName 11351
FileNameLabel 11352
FileTypeIndex 11352
FileTypes 11352
OkButtonLabel 11353
OnExecute 11354
OnFileOkClick 11355
OnFolderChange 11355
OnFolderChanging 11355
OnSelectionChange 11356
OnShareViolation 11356
OnTypeChange 11356
Options 11353
TFileOpenDialog events 11354
TFileOpenDialog Members 11346
TFileOpenDialog properties 11350
Title 11354
TFileOpenWith class 20619
about TFileOpenWith class 20619
AfterOpen 20627
BeforeExecute 20628
ExecuteTarget 20624
FileName 20625
HelpContext 20626
HelpKeyword 20626
HelpType 20627
OnUpdate 20628
TFileOpenWith events 20627
TFileOpenWith Members 20620
TFileOpenWith methods 20624
TFileOpenWith properties 20625
TFilePageSetup class 20629
about TFilePageSetup class 20629
BeforeExecute 20639
Caption 20635
Dialog 20635
Enabled 20635
GetDialogClass 20634

HelpContext 20636
HelpKeyword 20636
HelpType 20637
Hint 20637
ImageIndex 20638
OnAccept 20640
OnCancel 20640
OnHint 20640
OnUpdate 20640
SecondaryShortCuts 20638
ShortCut 20638
TFilePageSetup events 20639
TFilePageSetup Members 20630
TFilePageSetup methods 20634
TFilePageSetup properties 20634
Visible 20639
TFilePrintSetup class 20642
about TFilePrintSetup class 20642
BeforeExecute 20652
Caption 20647
Dialog 20647
Enabled 20648
GetDialogClass 20647
HelpContext 20648
HelpKeyword 20649
HelpType 20649
Hint 20650
ImageIndex 20650
OnAccept 20652
OnCancel 20652
OnHint 20652
OnUpdate 20653
SecondaryShortCuts 20650
ShortCut 20651
TFilePrintSetup events 20651
TFilePrintSetup Members 20642
TFilePrintSetup methods 20646
TFilePrintSetup properties 20647
Visible 20651
TFiler class 2834
about TFiler class 2834
Ancestor 2839
Create 2836
DefineBinaryProperty 2837
DefineProperty 2838
Destroy 2837
FlushBuffer 2838
IgnoreChildren 2839
LookupRoot 2840
Root 2840
SetRoot 2838
TFiler Members 2835
TFiler methods 2837
TFiler properties 2839
TFileRun class 11872
about TFileRun class 11872
Browse 11878
BrowseDlg 11879
Caption 11879
Directory 11879
Enabled 11880
FileName 11880
HelpContext 11881
Hint 11881
ImageIndex 11881
OnHint 11885
OnUpdate 11885
Operation 11882
Parameters 11882
ParentControl 11883
SecondaryShortCuts 11883
ShortCut 11883
ShowCmd 11884
TFileRun events 11885
TFileRun Members 11873
TFileRun properties 11878
Visible 11884
TFileSaveAs class 20654

iiii

about TFileSaveAs class 20654
BeforeExecute 20663
Caption 20660
Create 20659
Dialog 20660
Enabled 20661
GetDialogClass 20659
HelpContext 20661
Hint 20661
ImageIndex 20662
OnAccept 20664
OnCancel 20664
OnHint 20664
OnUpdate 20665
SecondaryShortCuts 20662
ShortCut 20663
TFileSaveAs events 20663
TFileSaveAs Members 20655
TFileSaveAs methods 20659
TFileSaveAs properties 20660
Visible 20663
TFileSaveDialog class 11357
about TFileSaveDialog class 11357
ClientGuid 11361
DefaultExtension 11361
DefaultFolder 11362
FavoriteLinks 11362
FileName 11362
FileNameLabel 11363
FileTypeIndex 11363
FileTypes 11363
OkButtonLabel 11364
OnExecute 11365
OnFileOkClick 11365
OnFolderChange 11366
OnFolderChanging 11366
OnOverwrite 11366
OnSelectionChange 11367
OnShareViolation 11367
OnTypeChange 11367
Options 11364
TFileSaveDialog events 11365
TFileSaveDialog Members 11357
TFileSaveDialog properties 11361
Title 11364
TFileStream class 2841
about TFileStream class 2841
Create 2843
Create constructor 2843
Destroy 2844
FFileName 2844
FileName 2845
TFileStream fields 2844
TFileStream Members 2841
TFileStream properties 2845
TFileTypeItem class 11368
about TFileTypeItem class 11368
Create 11370
Destroy 11370
DisplayName 11371
DisplayNameWStr 11371
FileMask 11372
FileMaskWStr 11372
GetDisplayName 11370
TFileTypeItem Members 11368
TFileTypeItem methods 11370
TFileTypeItem properties 11371
TFileTypeItems class 11372
about TFileTypeItems class 11372
Add 11375
FilterSpecArray 11375
Items 11376
TFileTypeItems Members 11373
TFileTypeItems methods 11375
TFileTypeItems properties 11376
TFilterComboBox class 13547
about TFilterComboBox class 13547
Align 13563

jjjj

Anchors 13567
AutoComplete 13567
AutoDropDown 13567
BevelEdges 13568
BevelInner 13568
BevelKind 13569
BevelOuter 13569
BuildList 13560
Change 13560
Click 13561
Color 13570
Constraints 13570
Create 13559
CreateWnd 13561
Ctl3D 13571
Destroy 13560
DragCursor 13571
DragMode 13572
Enabled 13573
FileList 13574
Filter 13575
Font 13576
ImeMode 13577
ImeName 13578
Mask 13579
Notification 13562
OnChange 13592
OnClick 13594
OnContextPopup 13596
OnDblClick 13596
OnDragDrop 13597
OnDragOver 13599
OnDropDown 13601
OnEndDrag 13603
OnEnter 13604
OnExit 13606
OnKeyDown 13608
OnKeyPress 13611
OnKeyUp 13612
OnMouseEnter 13614
OnMouseLeave 13614
OnStartDrag 13614
ParentColor 13580
ParentCtl3D 13580
ParentFont 13580
ParentShowHint 13581
PopupMenu 13581
ShowHint 13586
TabOrder 13587
TabStop 13588
Text 13588
TFilterComboBox events 13591
TFilterComboBox Members 13547
TFilterComboBox methods 13560
TFilterComboBox properties 13562
Visible 13590
TFindDialog class 11376
about TFindDialog class 11376
CloseDialog 11381
Create 11380
Destroy 11380
Execute 11381
Find 11383
FindText 11384
Left 11386
MessageHook 11383
OnFind 11392
Options 11386
Position 11390
Replace 11384
TFindDialog events 11392
TFindDialog Members 11377
TFindDialog methods 11380
TFindDialog properties 11384
Top 11392
TFirstButton class 16651
about TFirstButton class 16651
Create 16655

TFirstButton Members 16652
TFlatList class 7225
about TFlatList class 7225
Create 7226
DataSet 7228
FindItem 7227
GetCount 7227
GetUpdated 7227
ListChanging 7227
Locked 7229
TFlatList Members 7225
TFlatList methods 7226
TFlatList properties 7228
Update 7228
Updated 7229
UpdateList 7228
TFLDDesc class 20205
about TFLDDesc class 20205
bCalcField 20210
efldrRights 20210
efldvVchk 20210
FCalcField 20207
FFldNum 20207
FFLDRights 20207
FFldType 20208
FFLDVchk 20208
FLen 20208
FName 20208
FNullOffset 20208
FOffset 20209
FSubType 20209
FUnits1 20209
FUnits2 20209
iFldNum 20211
iFldType 20211
iLen 20211
iNullOffset 20211
iOffset 20211
iSubType 20212
iUnits1 20212
iUnits2 20212
szName 20212
TFLDDesc fields 20206
TFLDDesc Members 20206
TFLDDesc properties 20209
tfLeft enumeration member 14642
TFloatField class 7229
about TFloatField class 7229
Create 7236
currency 7240
GetAsFloat 7237
GetAsInteger 7237
GetAsString 7238
GetAsVariant 7238
GetDataSize 7238
GetText 7238
MaxValue 7240
MinValue 7241
Precision 7241
SetAsFloat 7239
SetAsInteger 7239
SetAsString 7239
SetVarValue 7239
TFloatField Members 7230
TFloatField methods 7237
TFloatField properties 7239
Value 7242
TFloatProperty class 10983
about TFloatProperty class 10983
AllEqual 10987
GetValue 10987
SetValue 10988
TFloatProperty Members 10984
TFloatProperty methods 10986
TFlowPanel class 12579
about TFlowPanel class 12579
Align 12591
Alignment 12594

Anchors 12595
AutoSize 12595
AutoWrap 12596
BevelEdges 12596
BevelInner 12596
BevelKind 12597
BevelOuter 12598
BevelWidth 12599
BiDiMode 12600
BorderStyle 12600
BorderWidth 12601
Caption 12602
Color 12605
Constraints 12605
Ctl3D 12606
DockSite 12606
DragCursor 12607
DragKind 12608
DragMode 12608
Enabled 12610
FlowStyle 12610
Font 12611
FullRepaint 12612
Locked 12612
OnAlignInsertBefore 12626
OnAlignPosition 12626
OnCanResize 12627
OnClick 12627
OnConstrainedResize 12629
OnContextPopup 12630
OnDblClick 12631
OnDockDrop 12631
OnDockOver 12633
OnDragDrop 12635
OnDragOver 12637
OnEndDock 12640
OnEndDrag 12640
OnEnter 12641
OnExit 12643
OnGetSiteInfo 12645
OnMouseActivate 12647
OnMouseDown 12648
OnMouseEnter 12650
OnMouseLeave 12651
OnMouseMove 12651
OnMouseUp 12653
OnResize 12655
OnStartDock 12656
OnStartDrag 12657
OnUnDock 12657
Padding 12613
ParentBackground 12613
ParentBiDiMode 12613
ParentColor 12614
ParentCtl3D 12614
ParentFont 12614
ParentShowHint 12615
PopupMenu 12616
ShowHint 12620
TabOrder 12622
TabStop 12622
TFlowPanel events 12625
TFlowPanel Members 12579
TFlowPanel properties 12590
UseDockManager 12623
VerticalAlignment 12623
Visible 12623
tfMilliseconds enumeration member 17235
tfMisc enumeration member 10256
tfModifyString enumeration member 14642
tfMSF enumeration member 17235
TFMTBCDField class 7242
about TFMTBCDField class 7242
CheckTypeSize 7250
CopyData 7250
Create 7248
currency 7254
GetAsBCD 7250

GetAsCurrency 7251
GetAsFloat 7251
GetAsInteger 7251
GetAsString 7251
GetAsVariant 7252
GetDataSize 7252
GetDefaultWidth 7252
GetText 7252
GetValue 7252
MaxValue 7255
MinValue 7255
Precision 7256
SetAsBCD 7253
SetAsCurrency 7253
SetAsFloat 7253
SetAsInteger 7253
SetAsString 7254
SetVarValue 7254
Size 7256
TFMTBCDField Members 7243
TFMTBCDField methods 7249
TFMTBCDField properties 7254
Value 7257
tfNoClip enumeration member 14642
tfNoDefaultRadioButton enumeration member 11566
tfNoPrefix enumeration member 14642
TFont class 14497
about TFont class 14497
Assign 14500
Changed 14502
Charset 14506
Color 14507
Create 14499
Destroy 14500
FontAdapter 14508
GetCharset 14502
GetHandle 14503
GetHeight 14503
GetName 14503
GetOrientation 14503
GetPitch 14503
GetSize 14504
GetStyle 14504
Handle 14509
Height 14510
Name 14510
Orientation 14511
Pitch 14512
PixelsPerInch 14512
SetCharset 14504
SetColor 14504
SetHandle 14505
SetHeight 14505
SetName 14505
SetOrientation 14505
SetPitch 14505
SetSize 14506
SetStyle 14506
Size 14513
Style 14513
TFont Members 14497
TFont methods 14500
TFont properties 14506
TFontAdapter class 1583
about TFontAdapter class 1583
Changed 1585
Create 1585
GetOleFont 1586
SetOleFont 1586
TFontAdapter Members 1584
TFontAdapter methods 1585
Update 1587
TFontDialog class 11394
about TFontDialog class 11394
Apply 11399
Create 11398
Destroy 11398
Device 11401

Execute 11399
Font 11401
MaxFontSize 11403
MinFontSize 11404
OnApply 11406
Options 11405
TFontDialog events 11406
TFontDialog Members 11395
TFontDialog methods 11398
TFontDialog properties 11401
WndProc 11400
TFontEdit class 20666
about TFontEdit class 20666
BeforeExecute 20676
Caption 20671
Dialog 20672
Enabled 20672
GetDialogClass 20671
HelpContext 20672
HelpKeyword 20673
HelpType 20673
Hint 20674
ImageIndex 20674
OnAccept 20676
OnCancel 20676
OnHint 20677
OnUpdate 20677
SecondaryShortCuts 20674
ShortCut 20675
TFontEdit events 20676
TFontEdit Members 20666
TFontEdit methods 20670
TFontEdit properties 20671
Visible 20675
TFontRecall class 14514
about TFontRecall class 14514
Create 14516
TFontRecall Members 14515
TForm class 13992
about TForm class 13992
Action 14011
ActiveControl 14012
ActiveMDIChild 14013
Align 14013
AlphaBlend 14017
AlphaBlendValue 14017
Anchors 14017
ArrangeIcons 14006
AutoScroll 14018
AutoSize 14018
BiDiMode 14019
BorderIcons 14019
BorderStyle 14020
BorderWidth 14021
Caption 14021
Cascade 14007
ClientHandle 14024
ClientHeight 14024
ClientWidth 14025
Color 14027
Constraints 14028
Ctl3D 14028
DefaultMonitor 14029
DockManager 14029
DockSite 14031
DragKind 14032
DragMode 14033
Enabled 14034
Font 14035
FormStyle 14036
Height 14038
HelpFile 14039
HorzScrollBar 14040
Icon 14041
KeyPreview 14041
MDIChildCount 14044
MDIChildren 14044
Menu 14045

Next 14008
ObjectMenuItem 14047
OldCreateOrder 14047
OnActivate 14065
OnAlignInsertBefore 14069
OnAlignPosition 14069
OnCanResize 14070
OnClick 14070
OnClose 14072
OnCloseQuery 14073
OnConstrainedResize 14074
OnContextPopup 14075
OnCreate 14076
OnDblClick 14077
OnDeactivate 14078
OnDestroy 14078
OnDockDrop 14079
OnDockOver 14081
OnDragDrop 14083
OnDragOver 14085
OnEndDock 14088
OnGetSiteInfo 14088
OnHelp 14090
OnHide 14091
OnKeyDown 14093
OnKeyPress 14095
OnKeyUp 14096
OnMouseActivate 14098
OnMouseDown 14099
OnMouseEnter 14101
OnMouseLeave 14102
OnMouseMove 14102
OnMouseUp 14104
OnMouseWheel 14107
OnMouseWheelDown 14107
OnMouseWheelUp 14107
OnPaint 14108
OnResize 14109
OnShortCut 14109
OnShow 14110
OnStartDock 14111
OnUnDock 14111
Padding 14047
ParentBiDiMode 14048
ParentFont 14048
PixelsPerInch 14049
PopupMenu 14049
PopupMode 14054
PopupParent 14054
Position 14055
Previous 14009
PrintScale 14056
Scaled 14056
ScreenSnap 14056
ShowHint 14057
SnapBuffer 14058
TForm events 14064
TForm Members 13993
TForm methods 14006
TForm properties 14010
Tile 14009
TileMode 14059
TransparentColor 14060
TransparentColorValue 14060
UseDockManager 14060
VertScrollBar 14061
Visible 14062
Width 14063
WindowMenu 14063
WindowState 14063
TFormLayout class 16656
about TFormLayout class 16656
BreakButtons 16663
ColumnCount 16663
Create 16658
EndButtons 16659
EndFields 16659
EndLayout 16659

EndRow 16659
EndTable 16660
ImplLayoutButton 16660
ImplLayoutField 16660
ImplLayoutLabelAndField 16661
ImplLayoutTable 16661
NextColumn 16662
StartButtons 16662
StartFields 16662
StartRow 16662
StartTable 16663
TableHeader 16664
TFormLayout Members 16656
TFormLayout methods 16658
TFormLayout properties 16663
tfPathEllipsis enumeration member 14642
tfPositionRelativeToWindow enumeration member 11566
tfQExecute enumeration member 10256
tfQPrepare enumeration member 10256
TFrame class 14114
about TFrame class 14114
Align 14126
Anchors 14130
AutoScroll 14130
AutoSize 14131
BiDiMode 14131
Color 14132
Constraints 14132
Ctl3D 14133
DockSite 14133
DragCursor 14134
DragKind 14135
DragMode 14135
Enabled 14137
Font 14137
OnAlignInsertBefore 14150
OnAlignPosition 14151
OnCanResize 14152
OnClick 14152
OnConstrainedResize 14154
OnContextPopup 14155
OnDblClick 14155
OnDockDrop 14156
OnDockOver 14158
OnDragDrop 14160
OnDragOver 14162
OnEndDock 14164
OnEndDrag 14165
OnEnter 14166
OnExit 14168
OnGetSiteInfo 14170
OnMouseActivate 14172
OnMouseDown 14173
OnMouseEnter 14175
OnMouseLeave 14175
OnMouseMove 14175
OnMouseUp 14178
OnMouseWheel 14180
OnMouseWheelDown 14180
OnMouseWheelUp 14181
OnResize 14181
OnStartDock 14182
OnStartDrag 14182
OnUnDock 14183
Padding 14138
ParentBackground 14139
ParentBiDiMode 14139
ParentColor 14139
ParentCtl3D 14140
ParentFont 14140
ParentShowHint 14141
PopupMenu 14141
ShowHint 14146
TabOrder 14147
TabStop 14148
TFrame events 14150
TFrame Members 14115
TFrame properties 14125

Visible 14148
tfRight enumeration member 14642
tfRtlLayout enumeration member 11566
tfRtlReading enumeration member 14642
tfSamples enumeration member 17235
tfShowMarqueeProgressBar enumeration member 11566
tfShowProgressBar enumeration member 11566
tfSingleLine enumeration member 14642
tfSMPTE24 enumeration member 17235
tfSMPTE25 enumeration member 17235
tfSMPTE30 enumeration member 17235
tfSMPTE30Drop enumeration member 17235
tfStmt enumeration member 10256
tfTMSF enumeration member 17235
tfTop enumeration member 14642
tfTransact enumeration member 10256
tfUseCommandLinks enumeration member 11566
tfUseCommandLinksNoIcon enumeration member 11566
tfUseHiconFooter enumeration member 11566
tfUseHiconMain enumeration member 11566
tfVendor enumeration member 10256
tfVerificationFlagChecked enumeration member 11566
tfVerticalCenter enumeration member 14642
tfWordBreak enumeration member 14642
tgCustom enumeration member 15337
tgEmbed enumeration member 15337
TGeneratedFunctions class 25300
about TGeneratedFunctions class 25300
Add 25301
Create 25300
Destroy 25301
Exists 25301
ItemCount 25302
Names 25302
Script 25302
TGeneratedFunctions Members 25300
TGeneratedFunctions methods 25301
TGeneratedFunctions properties 25301
tgImage enumeration member 15337
tgImageMap enumeration member 15337
TGlassFrame class 14186
about TGlassFrame class 14186
Assign 14188
Bottom 14190
Change 14189
Create 14187
Enabled 14190
FrameExtended 14189
IntersectsControl 14189
Left 14190
OnChange 14191
Right 14190
SheetOfGlass 14190
TGlassFrame events 14191
TGlassFrame Members 14186
TGlassFrame methods 14188
TGlassFrame properties 14189
Top 14190
tgLink enumeration member 15337
tgObject enumeration member 15337
TGraphic class 14516
about TGraphic class 14516
Changed 14520
Create 14519
DefineProperties 14520
Draw 14521
Empty 14528
Equals 14521
GetEmpty 14521
GetHeight 14521
GetPalette 14522
GetTransparent 14522
GetWidth 14522
Height 14528
LoadFromClipboardFormat 14522
LoadFromFile 14523
LoadFromStream 14524
Modified 14528

OnChange 14531
OnProgress 14531
Palette 14529
PaletteModified 14529
Progress 14524
ReadData 14524
SaveToClipboardFormat 14525
SaveToFile 14525
SaveToStream 14526
SetHeight 14526
SetPalette 14527
SetSize 14527
SetTransparent 14527
SetWidth 14527
TGraphic events 14531
TGraphic Members 14517
TGraphic methods 14519
TGraphic properties 14528
Transparent 14529
Width 14530
WriteData 14527
TGraphicControl class 6044
about TGraphicControl class 6044
Canvas 6053
Create 6051
Destroy 6052
Paint 6052
TGraphicControl Members 6044
TGraphicControl methods 6052
TGraphicControl properties 6053
TGraphicField class 7257
about TGraphicField class 7257
Create 7264
TGraphicField Members 7258
TGraphicsObject class 14532
about TGraphicsObject class 14532
Changed 14534
HandleAllocated 14534
Lock 14534
OnChange 14536
OwnerCriticalSection 14536
TGraphicsObject events 14536
TGraphicsObject Members 14532
TGraphicsObject methods 14533
TGraphicsObject properties 14535
Unlock 14535
TGridAttributes class 16664
about TGridAttributes class 16664
Align 16668
AssignTo 16667
BgColor 16668
Border 16669
CellPadding 16669
CellSpacing 16669
Create 16666
Custom 16670
Parent 16670
Style 16670
StyleRule 16671
TGridAttributes Members 16665
TGridAttributes methods 16666
TGridAttributes properties 16667
TGridDataLink class 9754
about TGridDataLink class 9754
ActiveChanged 9757
AddMapping 9757
BuildAggMap 9757
ClearMapping 9758
Create 9755
DataSetChanged 9758
DataSetScrolled 9758
DefaultFields 9762
Destroy 9756
EditingChanged 9759
FieldCount 9763
Fields 9763
FocusControl 9759
GetMappedIndex 9760

Grid 9764
IsAggRow 9760
LayoutChanged 9760
Modified 9761
RecordChanged 9761
Reset 9761
SparseMap 9764
TGridDataLink Members 9754
TGridDataLink methods 9756
TGridDataLink properties 9762
UpdateData 9762
TGridPanel class 12660
about TGridPanel class 12660
Align 12673
Alignment 12676
Anchors 12677
BevelEdges 12677
BevelInner 12677
BevelKind 12678
BevelOuter 12679
BevelWidth 12680
BiDiMode 12681
BorderStyle 12681
BorderWidth 12682
Caption 12683
Color 12686
ColumnCollection 12687
Constraints 12687
ControlCollection 12687
Ctl3D 12688
DockManager 12688
DockSite 12690
DragCursor 12691
DragKind 12692
DragMode 12692
Enabled 12693
ExpandStyle 12694
Font 12694
FullRepaint 12695
Locked 12696
OnAlignInsertBefore 12709
OnAlignPosition 12710
OnCanResize 12711
OnClick 12711
OnConstrainedResize 12713
OnContextPopup 12713
OnDblClick 12714
OnDockDrop 12714
OnDockOver 12716
OnDragDrop 12719
OnDragOver 12721
OnEndDock 12723
OnEndDrag 12724
OnEnter 12725
OnExit 12727
OnGetSiteInfo 12729
OnMouseActivate 12731
OnMouseDown 12732
OnMouseEnter 12734
OnMouseLeave 12734
OnMouseMove 12734
OnMouseUp 12737
OnResize 12739
OnStartDock 12740
OnStartDrag 12740
OnUnDock 12741
Padding 12696
ParentBackground 12696
ParentBiDiMode 12697
ParentColor 12697
ParentCtl3D 12697
ParentFont 12698
ParentShowHint 12698
PopupMenu 12699
RowCollection 12703
ShowHint 12704
TabOrder 12705
TabStop 12706

TGridPanel events 12709
TGridPanel Members 12661
TGridPanel properties 12672
UseDockManager 12706
VerticalAlignment 12707
Visible 12707
TGridRowAttributes class 16671
about TGridRowAttributes class 16671
Align 16674
AssignTo 16673
BgColor 16675
Create 16673
Custom 16675
Parent 16676
Style 16676
StyleRule 16676
TGridRowAttributes Members 16672
TGridRowAttributes methods 16673
TGridRowAttributes properties 16674
VAlign 16677
TGroupBox class 21474
about TGroupBox class 21474
Align 21485
Anchors 21489
BiDiMode 21490
Caption 21490
Color 21493
Constraints 21493
Ctl3D 21494
DockSite 21495
DragCursor 21496
DragKind 21496
DragMode 21496
Enabled 21498
Font 21498
OnAlignInsertBefore 21511
OnAlignPosition 21512
OnClick 21513
OnContextPopup 21514
OnDblClick 21515
OnDockDrop 21515
OnDockOver 21517
OnDragDrop 21520
OnDragOver 21522
OnEndDock 21524
OnEndDrag 21525
OnEnter 21526
OnExit 21528
OnGetSiteInfo 21530
OnMouseActivate 21532
OnMouseDown 21533
OnMouseEnter 21535
OnMouseLeave 21535
OnMouseMove 21535
OnMouseUp 21538
OnStartDock 21540
OnStartDrag 21541
OnUnDock 21541
Padding 21499
ParentBackground 21500
ParentBiDiMode 21500
ParentColor 21500
ParentCtl3D 21501
ParentFont 21501
ParentShowHint 21502
PopupMenu 21502
ShowHint 21507
TabOrder 21508
TabStop 21509
TGroupBox events 21511
TGroupBox Members 21474
TGroupBox properties 21485
Visible 21509
TGrpButtonItem class 2275
about TGrpButtonItem class 2275
ButtonGroup 2278
Collection 2278
GetNotifyTarget 2277

ScrollIntoView 2278
TGrpButtonItem Members 2275
TGrpButtonItem methods 2277
TGrpButtonItem properties 2278
TGrpButtonItems class 2278
about TGrpButtonItems class 2278
Add 2281
AddItem 2281
BeginUpdate 2281
ButtonGroup 2283
Create 2280
GetOwner 2281
Insert 2282
Items 2283
TGrpButtonItems Members 2279
TGrpButtonItems methods 2281
TGrpButtonItems properties 2283
Update 2282
tgTable enumeration member 15337
TGuidField class 7265
about TGuidField class 7265
AsGuid 7274
CheckTypeSize 7273
Create 7272
GetAsGuid 7273
GetDefaultWidth 7273
SetAsGuid 7273
TGuidField Members 7266
TGuidField methods 7273
TGuidField properties 7274
THandleObject class 22066
about THandleObject class 22066
Create 22068
Destroy 22068
FHandle 22069
FLastError 22069
FUseCOMWait 22069
Handle 22070
LastError 22070
THandleObject fields 22069
THandleObject Members 22067
THandleObject methods 22069
THandleObject properties 22070
WaitFor 22070
THandleStream class 2845
about THandleStream class 2845
Create 2847
FHandle 2848
Handle 2850
Read 2848
Seek 2849
SetSize 2849
SetSize method 2849
THandleStream fields 2847
THandleStream Members 2846
THandleStream methods 2848
THandleStream properties 2850
Write 2850
THashedStringList class 15450
about THashedStringList class 15450
Changed 15454
Destroy 15453
IndexOf 15454
IndexOfName 15455
THashedStringList Members 15451
THashedStringList methods 15454
THeader class 12744
about THeader class 12744
Align 12759
AllowResize 12762
Anchors 12763
BorderStyle 12763
Constraints 12763
Create 12754
CreateParams 12755
Destroy 12755
Enabled 12764
Font 12764

MouseDown 12756
MouseMove 12756
MouseUp 12757
OnContextPopup 12776
OnSized 12777
OnSizing 12777
Paint 12757
ParentFont 12766
ParentShowHint 12766
PopupMenu 12767
Sections 12771
SectionWidth 12771
ShowHint 12772
Sized 12758
Sizing 12758
TabOrder 12773
TabStop 12774
THeader events 12776
THeader Members 12744
THeader methods 12755
THeader properties 12758
Visible 12774
THeaderControl class 4118
about THeaderControl class 4118
Align 4130
Anchors 4134
BiDiMode 4134
BorderWidth 4135
Constraints 4135
DragCursor 4136
DragKind 4136
DragMode 4137
DragReorder 4138
Enabled 4138
Font 4139
FullDrag 4140
HotTrack 4140
Images 4141
OnContextPopup 4152
OnCreateSectionClass 4153
OnDragDrop 4153
OnDragOver 4155
OnDrawSection 4158
OnEndDock 4159
OnEndDrag 4160
OnMouseActivate 4161
OnMouseDown 4162
OnMouseEnter 4164
OnMouseLeave 4164
OnMouseMove 4164
OnMouseUp 4167
OnResize 4169
OnSectionClick 4170
OnSectionDrag 4170
OnSectionEndDrag 4170
OnSectionResize 4171
OnSectionTrack 4171
OnStartDock 4172
OnStartDrag 4172
ParentBiDiMode 4141
ParentFont 4141
ParentShowHint 4142
PopupMenu 4142
Sections 4147
ShowHint 4148
Style 4150
THeaderControl events 4152
THeaderControl Members 4119
THeaderControl properties 4130
Visible 4150
THeaderList class 15509
about THeaderList class 15509
Add 15512
Add method 15511
Clear 15512
Count 15514
Create 15510
Destroy 15511

wwww

Extract 15512
GetCount 15513
GetHeader 15513
GetOwnsObjects 15513
Headers 15515
IndexOf 15513
OwnsObjects 15515
SetOwnsObjects 15514
THeaderList Members 15510
THeaderList methods 15511
THeaderList properties 15514
THeaderSection class 4173
about THeaderSection class 4173
Alignment 4178
AllowClick 4178
Assign 4176
AutoSize 4179
BiDiMode 4179
Create 4175
GetDisplayName 4176
ImageIndex 4179
Left 4180
MaxWidth 4180
MinWidth 4180
ParentBiDiMode 4181
ParentBiDiModeChanged 4176
Right 4181
Style 4181
Text 4183
THeaderSection Members 4173
THeaderSection methods 4175
THeaderSection properties 4178
UseRightToLeftAlignment 4177
UseRightToLeftReading 4177
Width 4184
THeaderSections class 4185
about THeaderSections class 4185
Add 4188
AddItem 4188
Create 4187
GetOwner 4188
Insert 4189
Items 4190
THeaderSections Members 4185
THeaderSections methods 4187
THeaderSections properties 4190
Update 4189
THelpAction class 20678
about THelpAction class 20678
Create 20683
HandlesTarget 20684
THelpAction Members 20679
THelpAction methods 20683
UpdateTarget 20684
THelpContents class 20684
about THelpContents class 20684
ExecuteTarget 20690
THelpContents Members 20685
THelpContents methods 20689
THelpContextAction class 20690
about THelpContextAction class 20690
ExecuteTarget 20695
THelpContextAction Members 20691
THelpContextAction methods 20695
UpdateTarget 20695
THelpOnHelp class 20696
about THelpOnHelp class 20696
ExecuteTarget 20701
THelpOnHelp Members 20697
THelpOnHelp methods 20701
THelpTopicSearch class 20701
about THelpTopicSearch class 20701
ExecuteTarget 20707
THelpTopicSearch Members 20702
THelpTopicSearch methods 20706
THintAction class 20707
about THintAction class 20707
Create 20712

Hint 20713
THintAction Members 20708
THintAction properties 20712
THintWindow class 6055
about THintWindow class 6055
ActivateHint 6067
ActivateHintData 6067
BiDiMode 6071
CalcHintRect 6068
Canvas 6072
Caption 6073
Color 6076
Create 6066
CreateParams 6069
Font 6077
IsHintMsg 6069
NCPaint 6069
Paint 6070
ReleaseHandle 6070
ShouldHideHint 6070
THintWindow Members 6056
THintWindow methods 6066
THintWindow properties 6071
WMPrint 6071
THotKey class 4190
about THotKey class 4190
Anchors 4202
AutoSize 4202
BiDiMode 4203
Constraints 4203
Enabled 4204
Hint 4204
HotKey 4206
InvalidKeys 4207
Modifiers 4209
OnChange 4220
OnContextPopup 4220
OnEnter 4221
OnExit 4223
OnMouseActivate 4225
OnMouseDown 4226
OnMouseEnter 4228
OnMouseLeave 4228
OnMouseMove 4228
OnMouseUp 4231
ParentBiDiMode 4210
ParentShowHint 4210
PopupMenu 4211
ShowHint 4215
TabOrder 4217
TabStop 4217
THotKey events 4219
THotKey Members 4191
THotKey properties 4201
Visible 4218
THTMLItem class 25480
about THTMLItem class 25480
AssignString 25482
Create 25481
HTML 25482
Ident 25482
Length 25483
Offset 25483
TagToken 25483
THTMLItem Members 25480
THTMLItem methods 25481
THTMLItem properties 25482
THTMLItems class 25483
about THTMLItems class 25483
Create 25485
Destroy 25485
GetHTMLItem 25486
Items 25487
MakeIdentifier 25486
Prepare 25486
THTMLItems Members 25484
THTMLItems methods 25486
THTMLItems properties 25487

THTMLTableAttributes class 15288
about THTMLTableAttributes class 15288
Align 15292
AssignTo 15291
BgColor 15293
Border 15293
CellPadding 15294
CellSpacing 15294
Create 15290
RestoreDefaults 15291
THTMLTableAttributes Members 15289
THTMLTableAttributes methods 15290
THTMLTableAttributes properties 15292
Width 15294
THTMLTableCellAttributes class 15295
about THTMLTableCellAttributes class 15295
THTMLTableColumn class 10318
about THTMLTableColumn class 10318
Align 10324
AssignTo 10322
BgColor 10325
Create 10320
Custom 10325
Destroy 10321
DSTableProducer 10325
Field 10326
FieldName 10326
GetDisplayName 10322
GeTDSTableProducer 10323
IsStored 10323
RestoreDefaults 10323
THTMLTableColumn Members 10319
THTMLTableColumn methods 10321
THTMLTableColumn properties 10324
Title 10327
Update 10324
VAlign 10327
THTMLTableColumns class 10327
about THTMLTableColumns class 10327
Add 10330
Create 10330
DSTableProducer 10334
GetAttr 10331
GetAttrCount 10331
GetItemAttr 10331
GetOwner 10332
Items 10334
RebuildColumns 10332
RestoreDefaults 10333
State 10335
THTMLTableColumns Members 10328
THTMLTableColumns methods 10330
THTMLTableColumns properties 10334
Update 10333
THTMLTableElementAttributes class 15295
about THTMLTableElementAttributes class 15295
Align 15299
AssignTo 15297
BgColor 15299
RestoreDefaults 15298
THTMLTableElementAttributes Members 15296
THTMLTableElementAttributes methods 15297
THTMLTableElementAttributes properties 15299
VAlign 15300
THTMLTableHeaderAttributes class 15300
about THTMLTableHeaderAttributes class 15300
AssignTo 15302
Caption 15303
RestoreDefaults 15303
THTMLTableHeaderAttributes Members 15300
THTMLTableHeaderAttributes methods 15302
THTMLTableHeaderAttributes properties 15303
THTMLTableRowAttributes class 15304
about THTMLTableRowAttributes class 15304
THTMLTagAttributes class 15304
about THTMLTagAttributes class 15304
Changed 15307
Create 15306

Custom 15307
OnChange 15308
Producer 15308
RestoreDefaults 15307
THTMLTagAttributes events 15308
THTMLTagAttributes Members 15305
THTMLTagAttributes methods 15306
THTMLTagAttributes properties 15307
THTTPDataLink class 10335
about THTTPDataLink class 10335
ActiveChanged 10339
AddMapping 10339
ClearMapping 10340
Create 10338
DataSetChanged 10340
DataSetScrolled 10340
DefaultFields 10344
Destroy 10338
EditingChanged 10341
FieldCount 10344
Fields 10345
FocusControl 10341
GetMappedIndex 10342
LayoutChanged 10342
Modified 10342
RecordChanged 10343
Reset 10343
SparseMap 10345
THTTPDataLink Members 10336
THTTPDataLink methods 10338
THTTPDataLink properties 10344
UpdateData 10343
THTTPReqResp class 19961
_AddRef 19967
_Release 19967
about THTTPReqResp class 19961
AfterConstruction 19968
Agent 19975
BeforeExecute 19968
Check 19969
CheckContentType 19969
Connect 19970
ConnectTimeout 19976
Create 19966
Destroy 19966
Execute 19971
Execute method 19970
Get 19971
GetHTTPReqResp 19972
GetMimeBoundary 19972
InvokeOptions 19976
MaxSinglePostSize 19976
NewInstance 19972
OnBeforePost 19981
OnPostingData 19982
OnReceivingData 19982
Password 19976
Proxy 19977
ProxyByPass 19977
Receive 19973
ReceiveTimeout 19978
Send 19973
SendGet 19974
SendTimeout 19978
SetMimeBoundary 19974
SoapAction 19978
THTTPReqResp events 19981
THTTPReqResp Members 19962
THTTPReqResp methods 19966
THTTPReqResp properties 19975
UDDIBindingKey 19979
UDDIOperator 19979
URL 19980
UserName 19980
UseUTF8InHeader 19980
WSDLView 19981
THTTPRIO class 19928
about THTTPRIO class 19928

Converter 19935
Create 19932
Destroy 19932
HTTPWebNode 19935
Notification 19933
Port 19936
QueryInterface 19934
Service 19936
THTTPRIO Members 19929
THTTPRIO methods 19933
THTTPRIO properties 19934
URL 19937
WSDLItems 19937
WSDLLocation 19938
THTTPSoapCppInvoker class 19950
about THTTPSoapCppInvoker class 19950
THTTPSoapDispatcher class 24628
about THTTPSoapDispatcher class 24628
Create 24631
Destroy 24632
DispatchEnabled 24632
DispatchMask 24633
DispatchMethodType 24633
DispatchRequest 24634
THTTPSoapDispatcher Members 24629
THTTPSoapDispatcher methods 24632
THTTPSoapDispatcher properties 24634
WebDispatch 24634
THTTPSoapDispatchNode class 19940
about THTTPSoapDispatchNode class 19940
Dispatcher 19945
DispatchSOAP 19944
FSoapDispatcher 19943
Notification 19944
THTTPSoapDispatchNode fields 19943
THTTPSoapDispatchNode Members 19940
THTTPSoapDispatchNode methods 19943
THTTPSoapDispatchNode properties 19945
THTTPSoapPascalInvoker class 19951
about THTTPSoapPascalInvoker class 19951
DispatchSOAP 19955
THTTPSoapPascalInvoker Members 19952
THTTPSoapPascalInvoker methods 19955
TIArray class 17501
about TIArray class 17501
Assign 17502
AutoSize 17504
Create 17502
Destroy 17502
GetItem 17503
InsertAt 17503
Items 17504
Limit 17504
RemoveItem 17503
SetItem 17503
TIArray Members 17501
TIArray methods 17502
TIArray properties 17503
TIcon class 14537
about TIcon class 14537
Assign 14541
Create 14539
Destroy 14541
Draw 14542
GetEmpty 14542
GetHeight 14542
GetWidth 14542
Handle 14546
HandleAllocated 14543
LoadFromClipboardFormat 14543
LoadFromResourceID 14543
LoadFromResourceName 14544
LoadFromStream 14544
ReleaseHandle 14544
SaveToClipboardFormat 14545
SaveToStream 14545
SetHeight 14545
SetSize 14546

SetTransparent 14546
SetWidth 14546
TIcon Members 14537
TIcon methods 14541
TIcon properties 14546
TIconImage class 14547
about TIconImage class 14547
Destroy 14548
FreeHandle 14548
TIconImage Members 14548
TIconImage methods 14548
TIconOptions class 4233
about TIconOptions class 4233
Arrangement 4235
AutoArrange 4236
Create 4235
TIconOptions Members 4234
TIconOptions properties 4235
WrapText 4236
tiDirtyRead enumeration member 10257
TIDispatchField class 7274
about TIDispatchField class 7274
Create 7280
GetValue 7282
SetValue 7282
TIDispatchField Members 7275
TIDispatchField methods 7281
TIDispatchField properties 7282
Value 7282
TIExpert class 11817
about TIExpert class 11817
Execute 11818
GetAuthor 11818
GetComment 11818
GetGlyph 11818
GetIDString 11819
GetMenuText 11819
GetName 11819
GetPage 11819
GetState 11820
GetStyle 11820
TIExpert Members 11817
TIExpert methods 11817
TImage class 12777
about TImage class 12777
Align 12789
Anchors 12793
AutoSize 12793
CanAutoSize 12786
Canvas 12794
Center 12797
Constraints 12797
Create 12785
DestRect 12787
Destroy 12786
DoPaletteChange 12787
DragCursor 12798
DragKind 12798
DragMode 12799
Enabled 12800
GetPalette 12787
IncrementalDisplay 12801
OnClick 12814
OnContextPopup 12816
OnDblClick 12817
OnDragDrop 12817
OnDragOver 12819
OnEndDock 12822
OnEndDrag 12822
OnMouseActivate 12823
OnMouseDown 12824
OnMouseEnter 12826
OnMouseLeave 12826
OnMouseMove 12827
OnMouseUp 12829
OnProgress 12831
OnStartDock 12832
OnStartDrag 12832

Paint 12788
ParentShowHint 12801
Picture 12802
PopupMenu 12803
Progress 12788
Proportional 12808
ShowHint 12808
Stretch 12810
TImage events 12813
TImage Members 12778
TImage methods 12786
TImage properties 12788
Transparent 12811
Visible 12812
TImageList class 6078
about TImageList class 6078
AllocBy 6083
BkColor 6083
BlendColor 6084
DrawingStyle 6084
Height 6085
ImageType 6085
Masked 6086
OnChange 6087
ShareImages 6086
TImageList events 6087
TImageList Members 6079
TImageList properties 6083
Width 6086
Time enumeration member 22260
TImplementedAdapterAction class 24491
about TImplementedAdapterAction class 24491
TImpliedDatabaseSprig class 1739
about TImpliedDatabaseSprig class 1739
AnyProblems 1740
Caption 1740
DragDropTo 1741
DragOverTo 1741
FigureParent 1741
ItemClass 1741
TImpliedDatabaseSprig Members 1740
TImpliedDatabaseSprig methods 1740
UniqueName 1741
TImpliedSessionSprig class 1742
about TImpliedSessionSprig class 1742
Caption 1742
ItemClass 1743
TImpliedSessionSprig Members 1742
TImpliedSessionSprig methods 1742
UniqueName 1743
TImport class 25696
about TImport class 25696
GetLocation 25701
GetNamespace 25701
SetLocation 25701
SetNamespace 25702
TImport Members 25696
TImport methods 25701
TImports class 25702
about TImports class 25702
Add 25708
GetImport 25708
Insert 25708
TImports Members 25702
TImports methods 25707
TIndexArray class 17342
about TIndexArray class 17342
Add 17344
Autosize 17345
Capacity 17345
Count 17345
Create 17343
Destroy 17343
Expand 17344
idxElements 17344
Items 17346
MemoryUsage 17346
SetSize 17344

TIndexArray fields 17343
TIndexArray Members 17342
TIndexArray methods 17344
TIndexArray properties 17345
TIndexDef class 7283
about TIndexDef class 7283
Assign 7285
CaseInsFields 7287
Create 7285
DescFields 7288
Expression 7289
FieldExpression 7289
Fields 7290
GetDisplayName 7286
GroupingLevel 7291
Options 7291
Source 7292
TIndexDef Members 7283
TIndexDef methods 7285
TIndexDef properties 7286
TIndexDefs class 7292
about TIndexDefs class 7292
Add 7295
AddIndexDef 7296
Create 7295
Find 7297
FindIndexForFields 7298
GetIndexForFields 7298
Items 7300
TIndexDefs Members 7293
TIndexDefs methods 7295
TIndexDefs properties 7300
Update 7299
TIndexFiles class 10033
about TIndexFiles class 10033
Add 10037
Clear 10039
Create 10036
Delete 10039
Insert 10040
TIndexFiles Members 10033
TIndexFiles methods 10036
TIndexInfo class 17818
about TIndexInfo class 17818
Add 17819
AddAggs 17821
AddOffset 17819
Capacity 17821
Count 17822
Create 17818
Destroy 17819
GetCapacity 17820
IsSparse 17820
IsSparseAgg 17820
LockIndex 17820
SetCapacity 17821
TIndexInfo Members 17818
TIndexInfo methods 17819
TIndexInfo properties 17821
UnlockIndex 17821
TInetXPageElements class 17142
about TInetXPageElements class 17142
BodyContent 17144
FormsContent 17143
IncludesContent 17143
ScriptContent 17144
StylesContent 17144
TInetXPageElements fields 17143
TInetXPageElements Members 17142
TInetXPageElements methods 17144
WarningsContent 17144
TInetXPageProducer class 17145
about TInetXPageProducer class 17145
EnableXMLIslands 17151
HTMLDoc 17152
HTMLFile 17153
IncludePathURL 17153
OnAfterGetContent 17156

eeee

OnAfterGetXMLData 17156
OnBeforeGetContent 17157
OnBeforeGetXMLData 17157
ScriptEngine 17154
Styles 17154
StylesFile 17155
TInetXPageProducer events 17156
TInetXPageProducer Members 17146
TInetXPageProducer properties 17151
WebPageItems 17155
TIniFile class 15455
about TIniFile class 15455
DeleteKey 15458
Destroy 15457
EraseSection 15458
ReadSection 15459
ReadSections 15459
ReadSectionValues 15460
ReadString 15460
TIniFile Members 15456
TIniFile methods 15458
UpdateFile 15461
WriteString 15461
TInplaceEdit class 14938
about TInplaceEdit class 14938
BoundsChanged 14951
Create 14950
CreateParams 14952
DblClick 14952
Deselect 14952
DoMouseWheel 14953
EditCanModify 14953
Grid 14958
Hide 14953
Invalidate 14954
KeyDown 14954
KeyPress 14955
KeyUp 14955
Move 14956
PosEqual 14956
SetFocus 14956
TInplaceEdit Members 14938
TInplaceEdit methods 14951
TInplaceEdit properties 14958
UpdateContents 14957
UpdateLoc 14957
Visible 14958
WndProc 14958
TInplaceEditList class 14959
about TInplaceEditList class 14959
ActiveList 14966
BoundsChanged 14961
ButtonRect 14961
ButtonWidth 14966
CloseUp 14961
Create 14960
DblClick 14961
DoDropDownKeys 14962
DoEditButtonClick 14962
DoGetPickListItems 14962
DropDown 14962
DropDownRows 14966
EditStyle 14966
KeyDown 14963
ListMouseUp 14963
ListVisible 14966
MouseDown 14963
MouseMove 14963
MouseUp 14963
OnEditButtonClick 14967
OnGetPickListitems 14968
OverButton 14964
PaintWindow 14964
PickList 14967
PickListLoaded 14967
Pressed 14967
RestoreContents 14964
StopTracking 14964

TInplaceEditList events 14967
TInplaceEditList Members 14959
TInplaceEditList methods 14960
TInplaceEditList properties 14965
TrackButton 14965
UpdateContents 14965
WndProc 14965
TInsertButton class 16677
about TInsertButton class 16677
Create 16682
TInsertButton Members 16678
TInt64Property class 10988
about TInt64Property class 10988
AllEqual 10991
GetEditLimit 10992
GetValue 10992
SetValue 10993
TInt64Property Members 10988
TInt64Property methods 10991
TIntArray class 17346
about TIntArray class 17346
Add 17349
Assign 17349
Create 17348
Find 17350
GetItem 17350
Items 17351
PutItem 17350
TIntArray Members 17346
TIntArray methods 17348
TIntArray properties 17350
TIntegerBucketList class 5670
about TIntegerBucketList class 5670
Add 5672
Data 5673
GetData 5672
Remove 5672
SetData 5672
TIntegerBucketList Members 5670
TIntegerBucketList methods 5671
TIntegerBucketList properties 5672
TIntegerField class 7301
about TIntegerField class 7301
Create 7309
GetAsFloat 7310
GetAsInteger 7310
GetAsString 7311
GetAsVariant 7311
GetDataSize 7311
GetText 7311
GetValue 7312
MaxValue 7313
MinValue 7313
SetAsFloat 7312
SetAsInteger 7312
SetAsString 7312
SetVarValue 7312
TIntegerField Members 7303
TIntegerField methods 7310
TIntegerField properties 7313
Value 7314
TIntegerProperty class 10993
about TIntegerProperty class 10993
GetValue 10996
SetValue 10997
TIntegerProperty Members 10994
TIntegerProperty methods 10996
TInterfacedObject class 22276
_AddRef 22278
_Release 22279
about TInterfacedObject class 22276
AfterConstruction 22279
BeforeDestruction 22279
FRefCount 22278
NewInstance 22280
QueryInterface 22280
RefCount 22281
TInterfacedObject data members 22277

TInterfacedObject Members 22276
TInterfacedObject methods 22278
TInterfacedObject properties 22281
TInterfacedPersistent class 2851
_AddRef 2853
_Release 2854
about TInterfacedPersistent class 2851
AfterConstruction 2854
QueryInterface 2854
TInterfacedPersistent Members 2852
TInterfacedPersistent methods 2853
TInterfaceField class 7314
about TInterfaceField class 7314
CheckTypeSize 7321
Create 7320
GetAsVariant 7321
GetValue 7322
SetValue 7322
SetVarValue 7322
TInterfaceField Members 7315
TInterfaceField methods 7321
TInterfaceField properties 7322
Value 7322
TInterfaceList class 2855
about TInterfaceList class 2855
Add 2858
Capacity 2865
Clear 2858
Count 2865
Create 2857
Delete 2859
Destroy 2857
Exchange 2859
Expand 2859
First 2860
Get 2860
GetCapacity 2860
GetCount 2861
GetEnumerator 2861
IndexOf 2861
Insert 2862
Items 2866
Last 2862
Lock 2862
Put 2863
Remove 2863
SetCapacity 2864
SetCount 2864
TInterfaceList Members 2855
TInterfaceList methods 2857
TInterfaceList properties 2865
Unlock 2864
TInterfaceListEnumerator class 2866
about TInterfaceListEnumerator class 2866
Create 2867
Current 2868
GetCurrent 2867
MoveNext 2868
TInterfaceListEnumerator Members 2867
TInterfaceListEnumerator methods 2867
TInterfaceListEnumerator properties 2868
TInterfaceProperty class 10998
about TInterfaceProperty class 10998
AllEqual 11001
GetComponent 11002
GetComponentReference 11002
GetSelections 11002
GetValues 11002
ReceiveComponentNames 11003
SetValue 11003
TInterfaceProperty Members 10998
TInterfaceProperty methods 11001
TInternalSQLDataSet class 19538
about TInternalSQLDataSet class 19538
CommandText 19550
CommandType 19551
DataSource 19552
GetMetadata 19552

MaxBlobSize 19553
ParamCheck 19553
Params 19554
SortFieldNames 19555
TInternalSQLDataSet Members 19538
TInternalSQLDataSet properties 19550
TInternetExplorer class 19053
about TInternetExplorer class 19053
AddressBar 19084
Application 19084
Busy 19084
ClientToWindow 19062
Connect 19062
ConnectTo 19063
Container 19084
Create 19059
DefaultInterface 19085
Destroy 19060
Disconnect 19063
Document 19085
ExecWB 19064
ExecWB method 19064
FullName 19085
FullScreen 19085
Get_AddressBar 19065
Get_Application 19065
Get_Busy 19065
Get_Container 19065
Get_Document 19065
Get_FullName 19066
Get_FullScreen 19066
Get_Height 19066
Get_HWnd 19066
Get_Left 19067
Get_LocationName 19067
Get_LocationURL 19067
Get_MenuBar 19067
Get_Name 19067
Get_Offline 19068
Get_Parent 19068
Get_Path 19068
Get_ReadyState 19068
Get_RegisterAsBrowser 19069
Get_RegisterAsDropTarget 19069
Get_Resizable 19069
Get_Silent 19069
Get_StatusBar 19069
Get_StatusText 19070
Get_TheaterMode 19070
Get_ToolBar 19070
Get_Top 19070
Get_TopLevelContainer 19071
Get_type_ 19071
Get_Visible 19071
Get_Width 19071
GetProperty 19071
GoBack 19072
GoForward 19072
GoHome 19072
GoSearch 19072
Height 19086
HWnd 19086
InitServerData 19072
InvokeEvent 19073
Left 19086
LocationName 19086
LocationURL 19086
MenuBar 19087
Name 19087
Navigate 19074, 19075
Navigate method 19073
Navigate2 19075, 19076
Navigate2 method 19075
Offline 19089
OnBeforeNavigate2 19094
OnClientToHostWindow 19094
OnCommandStateChange 19094
OnDocumentComplete 19094

OnDownloadBegin 19094
OnDownloadComplete 19095
OnFileDownload 19095
OnFullScreen 19095
OnMenuBar 19095
OnNavigateComplete2 19096
OnNavigateError 19096
OnNewWindow2 19096
OnPrintTemplateInstantiation 19096
OnPrintTemplateTeardown 19096
OnPrivacyImpactedStateChange 19097
OnProgressChange 19097
OnPropertyChange 19097
OnQuit 19097
OnSetSecureLockIcon 19098
OnStatusBar 19098
OnStatusTextChange 19098
OnTheaterMode 19098
OnTitleChange 19098
OnToolBar 19099
OnUpdatePageStatus 19099
OnVisible 19099
OnWindowClosing 19099
OnWindowSetHeight 19100
OnWindowSetLeft 19100
OnWindowSetResizable 19100
OnWindowSetTop 19100
OnWindowSetWidth 19100
Parent 19089
Path 19089
PutProperty 19077
QueryStatusWB 19077
Quit 19077
ReadyState 19090
Refresh 19077
Refresh2 19078
Refresh2 method 19077
RegisterAsBrowser 19090
RegisterAsDropTarget 19090
Resizable 19090
Set_AddressBar 19078
Set_FullScreen 19078
Set_Height 19079
Set_Left 19079
Set_MenuBar 19079
Set_Offline 19079
Set_RegisterAsBrowser 19079
Set_RegisterAsDropTarget 19080
Set_Resizable 19080
Set_Silent 19080
Set_StatusBar 19080
Set_StatusText 19081
Set_TheaterMode 19081
Set_ToolBar 19081
Set_Top 19081
Set_Visible 19081
Set_Width 19082
ShowBrowserBar 19082, 19083
ShowBrowserBar method 19082
Silent 19091
StatusBar 19091
StatusText 19091
Stop 19083
TheaterMode 19091
TInternetExplorer events 19093
TInternetExplorer Members 19054
TInternetExplorer methods 19061
TInternetExplorer properties 19083
ToolBar 19091
Top 19092
TopLevelContainer 19092
type_ 19092
Visible 19092
Width 19093
TInvalidWebSession class 24923
about TInvalidWebSession class 24923
Create 24924
GetSessionID 24925

jjjj

GetTimeoutMinutes 24925
GetValue 24926
SetTimeoutMinutes 24926
SetValue 24927
Terminate 24927
TInvalidWebSession Members 24923
TInvalidWebSession methods 24925
UpdateResponse 24927
TInvContext class 15516
about TInvContext class 15516
AllocServerData 15517
GetParamPointer 15518
GetResultPointer 15518
SetMethodInfo 15518
SetParamPointer 15519
SetResultPointer 15519
TInvContext Members 15516
TInvContext methods 15517
TInvokableClass class 15520
_AddRef 15523
_Release 15524
about TInvokableClass class 15520
AfterConstruction 15524
BeforeDestruction 15524
Create 15522
FRefCount 15522
FSOAPHeaders 15523
NewInstance 15525
QueryInterface 15525
RefCount 15526
TInvokableClass fields 15522
TInvokableClass Members 15521
TInvokableClass methods 15523
TInvokableClass properties 15526
TInvokableClassRegistry class 15526
about TInvokableClassRegistry class 15526
Create 15529
Destroy 15530
GetActionURIOfIID 15531
GetActionURIOfInfo 15532
GetClassFromIntfInfo 15532
GetExceptionInfoForInterface 15533
GetHeaderClass 15533
GetHeaderInfoForInterface 15534
GetHeaderName 15534
GetHeaderNamespace 15535
GetHeaderNamespace method 15535
GetInfoForURI 15536
GetInterfaceCount 15536
GetInterfaceExternalName 15537
GetInterfaceExternalName method 15537
GetInterfaceInfoFromName 15538
GetInterfaceTypeInfo 15538
GetIntfInvokeOptions 15539
GetIntfInvokeOptions method 15538
GetInvokableObjectFromClass 15539
GetMethExternalName 15540
GetMethInternalName 15540
GetNamespaceByGUID 15541
GetParamExternalName 15541
GetParamInternalName 15541
GetRegInterfaceEntry 15542
GetRequestHeaderInfoForInterface 15542
GetResponseHeaderInfoForInterface 15543
GetReturnParamNames 15544
GetUDDIInfo 15544
GetUDDIInfo method 15544
GetWSDLEncoding 15545
GetWSDLEncoding method 15545
HasRegInterfaceImpl 15546
RegisterAllSOAPActions 15546
RegisterDefaultSOAPAction 15547
RegisterException 15547
RegisterExceptionMethod 15548
RegisterExternalMethName 15548
RegisterExternalParamName 15549
RegisterHeaderClass 15550
RegisterHeaderClass method 15549

RegisterHeaderMethod 15551
RegisterInterface 15552
RegisterInvokableClass 15553
RegisterInvokableClass method 15553
RegisterInvokeOptions 15554
RegisterInvokeOptions method 15553
RegisterReturnParamNames 15554
RegisterUDDIInfo 15555
TInvokableClassRegistry Members 15527
TInvokableClassRegistry methods 15530
UnRegisterInterface 15555
UnRegisterInvokableClass 15556
TInvokeableVariantType class 23606
about TInvokeableVariantType class 23606
DispInvoke 23610
DoFunction 23610
DoProcedure 23611
FixupIdent 23612
GetProperty 23612
SetProperty 23613
TInvokeableVariantType Members 23608
TInvokeableVariantType methods 23609
TIpSocket class 20082
about TIpSocket class 20082
Bind 20087
Create 20086
Domain 20091
GetSocketAddr 20087
LocalDomainName 20088
LocalHost 20092
LocalHostAddr 20088
LocalHostName 20088
LocalPort 20092
LookupHostAddr 20089
LookupHostName 20089
LookupPort 20089
LookupProtocol 20090
ReceiveFrom 20090
RemoteHost 20093
RemotePort 20093
SendTo 20091
TIpSocket Members 20083
TIpSocket methods 20087
TIpSocket properties 20091
tiReadCommitted enumeration member 10257
tiRepeatableRead enumeration member 10257
TISAPIApplication class 15648
about TISAPIApplication class 15648
Create 15652
Destroy 15652
GetExtensionVersion 15652
HttpExtensionProc 15653
TerminateExtension 15653
TISAPIApplication Members 15649
TISAPIApplication methods 15652
TISAPIFactory class 15654
about TISAPIFactory class 15654
Create 15655
NewRequest 15655
NewResponse 15655
TISAPIFactory Members 15654
TISAPIFactory methods 15655
TISAPIRequest class 15662
about TISAPIRequest class 15662
Create 15665
ECB 15670
GetDateVariable 15665
GetFieldByName 15666
GetIntegerVariable 15666
GetStringVariable 15666
ReadClient 15667
ReadString 15668
TISAPIRequest Members 15662
TISAPIRequest methods 15665
TISAPIRequest properties 15670
TranslateURI 15668
WriteClient 15669
WriteHeaders 15669

WriteString 15670
TISAPIResponse class 15670
about TISAPIResponse class 15670
Create 15673
GetContent 15674
GetDateVariable 15674
GetIntegerVariable 15675
GetLogMessage 15675
GetStatusCode 15675
GetStringVariable 15676
InitResponse 15676
SendRedirect 15677
SendResponse 15677
SendStream 15677
Sent 15678
SetContent 15678
SetDateVariable 15678
SetIntegerVariable 15679
SetLogMessage 15679
SetStatusCode 15680
SetStringVariable 15680
TISAPIResponse Members 15671
TISAPIResponse methods 15674
TItemProp class 23394
about TItemProp class 23394
AssignTo 23396
Create 23395
Destroy 23396
EditMask 23398
EditStyle 23398
HasPickList 23397
KeyDesc 23399
MaxLength 23399
PickList 23400
ReadOnly 23400
TItemProp Members 23394
TItemProp methods 23396
TItemProp properties 23398
UpdateEdit 23397
TJPEGData class 15686
about TJPEGData class 15686
Destroy 15687
FreeHandle 15687
TJPEGData Members 15687
TJPEGData methods 15687
TJPEGImage class 15688
about TJPEGImage class 15688
Assign 15693
AssignTo 15694
Bitmap 15702
Changed 15695
Compress 15695
CompressionQuality 15702
Create 15691
Destroy 15692
DIBNeeded 15696
Draw 15696
Equals 15696
FreeBitmap 15696
GetEmpty 15697
GetHeight 15697
GetPalette 15697
GetWidth 15697
Grayscale 15702
JPEGNeeded 15697
LoadFromClipboardFormat 15698
LoadFromStream 15698
NewBitmap 15699
NewImage 15699
Performance 15703
PixelFormat 15703
ProgressiveDisplay 15703
ProgressiveEncoding 15704
ReadData 15699
ReadStream 15699
SaveToClipboardFormat 15700
SaveToStream 15700
Scale 15704

SetHeight 15700
SetPalette 15701
SetWidth 15701
Smoothing 15704
TJPEGImage Members 15689
TJPEGImage methods 15692
TJPEGImage properties 15701
WriteData 15701
tk_alias enumeration member 18222
tk_any enumeration member 18222
tk_array enumeration member 18222
tk_boolean enumeration member 18222
tk_char enumeration member 18222
tk_double enumeration member 18222
tk_enum enumeration member 18222
tk_except enumeration member 18222
tk_fixed enumeration member 18222
tk_float enumeration member 18222
tk_long enumeration member 18222
tk_longdouble enumeration member 18222
tk_longlong enumeration member 18222
tk_null enumeration member 18222
tk_objref enumeration member 18222
tk_octet enumeration member 18222
tk_Principal enumeration member 18222
tk_sequence enumeration member 18222
tk_short enumeration member 18222
tk_string enumeration member 18222
tk_struct enumeration member 18222
tk_TypeCode enumeration member 18222
tk_ulong enumeration member 18222
tk_ulonglong enumeration member 18222
tk_union enumeration member 18222
tk_ushort enumeration member 18222
tk_void enumeration member 18222
tk_wchar enumeration member 18222
tk_wstring enumeration member 18222
tkArray enumeration member 23379
tkChar enumeration member 23379
tkClass enumeration member 23379
tkDynArray enumeration member 23379
tkEnumeration enumeration member 23379
tkFloat enumeration member 23379
tkInt64 enumeration member 23379
tkInteger enumeration member 23379
tkInterface enumeration member 23379
tkLString enumeration member 23379
tkMethod enumeration member 23379
tkRecord enumeration member 23379
tkSet enumeration member 23379
tkString enumeration member 23379
tkUnknown enumeration member 23379
tkVariant enumeration member 23379
tkWChar enumeration member 23379
tkWString enumeration member 23379
TLabel class 21544
about TLabel class 21544
Align 21553
Alignment 21556
Anchors 21557
AutoSize 21557
BiDiMode 21558
Caption 21559
Color 21561
Constraints 21562
DragCursor 21563
DragKind 21563
DragMode 21563
EllipsisPosition 21565
Enabled 21565
FocusControl 21566
Font 21566
Layout 21567
OnClick 21579
OnContextPopup 21581
OnDblClick 21582
OnDragDrop 21582
OnDragOver 21584

OnEndDock 21587
OnEndDrag 21587
OnMouseActivate 21588
OnMouseDown 21589
OnMouseEnter 21592
OnMouseLeave 21592
OnMouseMove 21592
OnMouseUp 21594
OnStartDock 21597
OnStartDrag 21597
ParentBiDiMode 21567
ParentColor 21568
ParentFont 21568
ParentShowHint 21569
PopupMenu 21569
ShowAccelChar 21574
ShowHint 21575
TLabel events 21579
TLabel Members 21545
TLabel properties 21552
Transparent 21576
Visible 21577
WordWrap 21578
TLabeledEdit class 12833
about TLabeledEdit class 12833
Anchors 12846
AutoSelect 12846
AutoSize 12847
BevelEdges 12848
BevelInner 12848
BevelKind 12849
BevelOuter 12849
BiDiMode 12849
BorderStyle 12850
CharCase 12850
Color 12851
Constraints 12851
Ctl3D 12852
DragCursor 12853
DragKind 12853
DragMode 12853
EditLabel 12855
Enabled 12855
Font 12856
HideSelection 12857
ImeMode 12857
ImeName 12858
LabelPosition 12858
LabelSpacing 12859
MaxLength 12859
OEMConvert 12860
OnChange 12873
OnClick 12874
OnContextPopup 12876
OnDblClick 12876
OnDragDrop 12877
OnDragOver 12879
OnEndDock 12881
OnEndDrag 12882
OnEnter 12883
OnExit 12885
OnKeyDown 12887
OnKeyPress 12890
OnKeyUp 12891
OnMouseActivate 12892
OnMouseDown 12893
OnMouseEnter 12896
OnMouseLeave 12896
OnMouseMove 12896
OnMouseUp 12898
OnStartDock 12901
OnStartDrag 12901
ParentBiDiMode 12860
ParentColor 12860
ParentCtl3D 12861
ParentFont 12861
ParentShowHint 12862
PasswordChar 12862

PopupMenu 12863
ReadOnly 12867
ShowHint 12868
TabOrder 12869
TabStop 12870
Text 12870
TLabeledEdit events 12873
TLabeledEdit Members 12834
TLabeledEdit properties 12845
Visible 12871
TLanguages class 22762
about TLanguages class 22762
Count 22764
Create 22763
Ext 22764
ID 22764
IndexOf 22763
LocaleID 22765
Name 22765
NameFromLCID 22766
NameFromLocaleID 22766
TLanguages Members 22762
TLanguages methods 22763
TLanguages properties 22763
TLargeintField class 7323
about TLargeintField class 7323
AsLargeInt 7334
Create 7329
GetAsFloat 7331
GetAsInteger 7331
GetAsLargeint 7331
GetAsString 7331
GetAsVariant 7331
GetDataSize 7332
GetDefaultWidth 7332
GetText 7332
GetValue 7332
MaxValue 7335
MinValue 7335
SetAsFloat 7333
SetAsInteger 7333
SetAsLargeint 7333
SetAsString 7333
SetVarValue 7333
TLargeintField Members 7324
TLargeintField methods 7330
TLargeintField properties 7334
Value 7335
TLastButton class 16682
about TLastButton class 16682
Create 16687
TLastButton Members 16683
TLayoutAttributes class 24695
about TLayoutAttributes class 24695
ControlAttributes 24697
Create 24696
LabelAttributes 24697
LabelPosition 24697
TLayoutAttributes fields 24697
TLayoutAttributes Members 24696
TLayoutGroup class 16687
about TLayoutGroup class 16687
Custom 16692
DisplayColumns 16693
Style 16693
StyleRule 16694
TLayoutGroup Members 16688
TLayoutGroup properties 16692
TLayoutWebContent class 24698
_AddRef 24700
_Release 24701
about TLayoutWebContent class 24698
Create 24700
ImplLayoutButton 24701
ImplLayoutField 24702
ImplLayoutLabelAndField 24702
ImplLayoutTable 24703
LayoutButton 24703

LayoutField 24704
LayoutLabelAndField 24704
LayoutTable 24705
ParentLayout 24706
QueryInterface 24705
TLayoutWebContent Members 24698
TLayoutWebContent methods 24700
TLayoutWebContent properties 24705
tlBottom enumeration member 21951
tlbsCRLF enumeration member 22569
tlbsLF enumeration member 22569
tlCenter enumeration member 21951
TLineNumberMap class 25488
about TLineNumberMap class 25488
Count 25490
DestLineToSourceLine 25489
Items 25490
New 25489
TLineNumberMap Members 25488
TLineNumberMap methods 25489
TLineNumberMap properties 25490
TLinkedRIO class 19990
about TLinkedRIO class 19990
Create 19994
CreateFile 19994
Destroy 19995
GetResponseStream 19996
QueryInterface 19996
TLinkedRIO Members 19991
TLinkedRIO methods 19995
TLinkedRIO properties 19997
WebNode 19997
TLinkedWebNode class 19997
about TLinkedWebNode class 19997
BeforeExecute 20002
Create 20001
Destroy 20001
Execute 20003
Execute method 20002
GetMimeBoundary 20003
GetResponseStream 20004
Invoker 20004
MimeBoundary 20005
SetMimeBoundary 20004
TLinkedWebNode Members 19998
TLinkedWebNode methods 20001
TLinkedWebNode properties 20004
TList class 2868
about TList class 2868
Add 2871
Assign 2874
Capacity 2895
Clear 2874
Count 2896
Delete 2874
Destroy 2870
Error 2875
Error method 2875
Exchange 2876
Expand 2877
Extract 2877
First 2878
Get 2878
GetEnumerator 2878
Grow 2879
IndexOf 2879
Insert 2882
Items 2898
Last 2882
List 2900
Move 2884
Notify 2884
Pack 2885
Put 2887
Remove 2887
SetCapacity 2891
SetCount 2892
Sort 2892

TList Members 2869
TList methods 2871
TList properties 2895
TListActionLink class 15744
about TListActionLink class 15744
AddItem 15748
AddItem method 15747
IsActiveLinked 15748
IsImagesLinked 15748
RefreshControl 15748
SetAction 15749
SetActive 15749
SetImages 15749
SetItemIndex 15749
TListActionLink Members 15745
TListActionLink methods 15747
TListBox class 21598
about TListBox class 21598
Align 21611
Anchors 21615
AutoComplete 21615
AutoCompleteDelay 21615
BevelEdges 21616
BevelInner 21616
BevelKind 21617
BevelOuter 21617
BevelWidth 21617
BiDiMode 21618
BorderStyle 21618
Color 21619
Columns 21620
Constraints 21620
Ctl3D 21621
DragCursor 21621
DragKind 21621
DragMode 21622
Enabled 21623
ExtendedSelect 21624
Font 21624
ImeMode 21625
ImeName 21626
IntegralHeight 21627
ItemHeight 21627
Items 21628
MultiSelect 21630
OnClick 21645
OnContextPopup 21647
OnData 21647
OnDataFind 21648
OnDataObject 21648
OnDblClick 21649
OnDragDrop 21649
OnDragOver 21651
OnDrawItem 21654
OnEndDock 21656
OnEndDrag 21656
OnEnter 21657
OnExit 21660
OnKeyDown 21661
OnKeyPress 21664
OnKeyUp 21665
OnMeasureItem 21667
OnMouseActivate 21668
OnMouseDown 21669
OnMouseEnter 21671
OnMouseLeave 21672
OnMouseMove 21672
OnMouseUp 21674
OnStartDock 21676
OnStartDrag 21677
ParentBiDiMode 21631
ParentColor 21631
ParentCtl3D 21631
ParentFont 21632
ParentShowHint 21632
PopupMenu 21633
ScrollWidth 21637
ShowHint 21638

Sorted 21639
Style 21641
TabOrder 21641
TabStop 21642
TabWidth 21642
TListBox events 21644
TListBox Members 21598
TListBox properties 21610
Visible 21643
TListColumn class 4236
about TListColumn class 4236
Alignment 4242
Assign 4239
AutoSize 4242
Caption 4242
Create 4238
DefineProperties 4240
Destroy 4239
GetDisplayName 4241
ImageIndex 4244
MaxWidth 4244
MinWidth 4244
SetIndex 4241
Tag 4244
TListColumn Members 4237
TListColumn methods 4239
TListColumn properties 4241
Width 4245
WidthType 4245
TListColumns class 4245
about TListColumns class 4245
Add 4248
Create 4247
GetOwner 4248
Items 4250
Owner 4249
TListColumns Members 4246
TListColumns methods 4248
TListColumns properties 4250
Update 4249
TListControlAction class 11886
about TListControlAction class 11886
Create 11891
HandlesTarget 11892
IsEnabled 11892
ListControl 11893
Notification 11892
TListControlAction Members 11887
TListControlAction methods 11891
TListControlAction properties 11893
TListControlClearSelection class 11893
about TListControlClearSelection class 11893
Caption 11900
Enabled 11900
ExecuteTarget 11898
HelpContext 11900
HelpKeyword 11901
HelpType 11901
Hint 11902
ImageIndex 11902
ListControl 11902
OnHint 11904
OnUpdate 11904
SecondaryShortCuts 11903
ShortCut 11903
TListControlClearSelection events 11904
TListControlClearSelection Members 11894
TListControlClearSelection methods 11898
TListControlClearSelection properties 11899
UpdateTarget 11899
Visible 11904
TListControlCopySelection class 11906
about TListControlCopySelection class 11906
Caption 11912
Destination 11913
Enabled 11913
ExecuteTarget 11910
HandlesTarget 11911

HelpContext 11914
HelpKeyword 11914
HelpType 11915
Hint 11915
ImageIndex 11915
ListControl 11916
OnHint 11917
OnUpdate 11918
SecondaryShortCuts 11916
ShortCut 11916
TListControlCopySelection events 11917
TListControlCopySelection Members 11906
TListControlCopySelection methods 11910
TListControlCopySelection properties 11912
UpdateTarget 11912
Visible 11917
TListControlDeleteSelection class 11919
about TListControlDeleteSelection class 11919
Caption 11925
Enabled 11925
ExecuteTarget 11924
HelpContext 11926
HelpKeyword 11926
HelpType 11927
Hint 11927
ImageIndex 11927
ListControl 11928
OnHint 11929
OnUpdate 11930
SecondaryShortCuts 11928
ShortCut 11929
TListControlDeleteSelection events 11929
TListControlDeleteSelection Members 11920
TListControlDeleteSelection methods 11923
TListControlDeleteSelection properties 11925
UpdateTarget 11924
Visible 11929
TListControlItem class 15750
about TListControlItem class 15750
Assign 15753
Caption 15756
Changed 15754
Create 15752
Data 15756
FCaption 15752
FData 15752
FImageIndex 15753
GetDisplayName 15754
ImageIndex 15757
SetCaption 15755
SetData 15755
SetImageIndex 15755
TListControlItem fields 15752
TListControlItem Members 15750
TListControlItem methods 15753
TListControlItem properties 15756
TListControlItems class 15757
about TListControlItems class 15757
Add 15760
CaseSensitive 15762
CompareItems 15761
Create 15760
CustomSort 15761
Items 15763
OnCompare 15764
Sort 15762
SortType 15764
TListControlItems events 15764
TListControlItems Members 15758
TListControlItems methods 15760
TListControlItems properties 15762
TListControlMoveSelection class 11931
about TListControlMoveSelection class 11931
ExecuteTarget 11936
TListControlMoveSelection Members 11932
TListControlMoveSelection methods 11936
TListControlSelectAll class 11937
about TListControlSelectAll class 11937

Caption 11943
Enabled 11943
ExecuteTarget 11942
HelpContext 11944
HelpKeyword 11944
HelpType 11945
Hint 11945
ImageIndex 11945
ListControl 11946
OnHint 11947
OnUpdate 11948
SecondaryShortCuts 11946
ShortCut 11946
TListControlSelectAll events 11947
TListControlSelectAll Members 11937
TListControlSelectAll methods 11941
TListControlSelectAll properties 11943
UpdateTarget 11942
Visible 11947
TListEnumerator class 2900
about TListEnumerator class 2900
Create 2901
Current 2902
GetCurrent 2901
MoveNext 2902
TListEnumerator Members 2901
TListEnumerator methods 2901
TListEnumerator properties 2902
TListItem class 4250
about TListItem class 4250
Assign 4253
CancelEdit 4254
Caption 4258
Checked 4258
Create 4252
Cut 4259
Data 4259
Delete 4254
Deleting 4260
Destroy 4252
DisplayRect 4254
DropTarget 4260
EditCaption 4255
Focused 4260
GetPosition 4255
Handle 4261
ImageIndex 4261
Indent 4263
Index 4264
Left 4265
ListView 4265
MakeVisible 4256
OverlayIndex 4266
Owner 4266
Position 4267
Selected 4267
SetPosition 4256
StateIndex 4267
SubItemImages 4268
SubItems 4268
TListItem Members 4251
TListItem methods 4253
TListItem properties 4257
Top 4270
Update 4256
WorkArea 4257
TListItems class 4270
about TListItems class 4270
Add 4273
AddItem 4275
Assign 4275
BeginUpdate 4276
Clear 4276
Count 4281
Create 4272
CreateItem 4276
DefineProperties 4277
Delete 4277

Destroy 4272
EndUpdate 4277
GetCount 4278
GetEnumerator 4278
GetHandle 4278
GetItem 4278
Handle 4281
IndexOf 4278
Insert 4279
Item 4282
Owner 4283
SetCount 4280
SetItem 4280
SetUpdateState 4281
TListItems Members 4271
TListItems methods 4273
TListItems properties 4281
TListItemsEnumerator class 4283
about TListItemsEnumerator class 4283
Create 4284
Current 4285
GetCurrent 4284
MoveNext 4284
TListItemsEnumerator Members 4283
TListItemsEnumerator methods 4284
TListItemsEnumerator properties 4284
TListSourceLink class 9498
about TListSourceLink class 9498
ActiveChanged 9501
Create 9500
DataSetChanged 9502
LayoutChanged 9502
TListSourceLink Members 9499
TListSourceLink methods 9501
TListView class 4285
about TListView class 4285
Action 4300
Align 4300
AllocBy 4304
Anchors 4304
BevelEdges 4304
BevelInner 4305
BevelKind 4305
BevelOuter 4306
BevelWidth 4306
BiDiMode 4307
BorderStyle 4307
BorderWidth 4307
Checkboxes 4308
Color 4308
ColumnClick 4309
Columns 4309
Constraints 4311
Ctl3D 4312
DragCursor 4312
DragKind 4312
DragMode 4313
Enabled 4314
FlatScrollBars 4315
Font 4315
FullDrag 4316
GridLines 4317
HideSelection 4317
HotTrack 4317
HotTrackStyles 4318
HoverTime 4318
IconOptions 4318
Items 4319
LargeImages 4322
MultiSelect 4324
OnAdvancedCustomDraw 4346
OnAdvancedCustomDrawItem 4347
OnAdvancedCustomDrawSubItem 4347
OnChange 4348
OnChanging 4349
OnClick 4349
OnColumnClick 4351
OnColumnDragged 4352

OnColumnRightClick 4352
OnCompare 4353
OnContextPopup 4353
OnCustomDraw 4354
OnCustomDrawItem 4355
OnCustomDrawSubItem 4355
OnData 4356
OnDataFind 4356
OnDataHint 4357
OnDataStateChange 4357
OnDblClick 4358
OnDeletion 4358
OnDragDrop 4359
OnDragOver 4361
OnDrawItem 4363
OnEdited 4364
OnEditing 4364
OnEndDock 4365
OnEndDrag 4365
OnEnter 4366
OnExit 4369
OnGetImageIndex 4370
OnGetSubItemImage 4371
OnInfoTip 4371
OnInsert 4372
OnKeyDown 4372
OnKeyPress 4375
OnKeyUp 4376
OnMouseActivate 4378
OnMouseDown 4379
OnMouseEnter 4381
OnMouseLeave 4381
OnMouseMove 4381
OnMouseUp 4384
OnResize 4386
OnSelectItem 4387
OnStartDock 4387
OnStartDrag 4387
OwnerData 4325
OwnerDraw 4325
ParentBiDiMode 4326
ParentColor 4326
ParentFont 4327
ParentShowHint 4327
PopupMenu 4328
ReadOnly 4332
RowSelect 4333
ShowColumnHeaders 4333
ShowHint 4334
ShowWorkAreas 4335
SmallImages 4336
SortType 4338
StateImages 4339
TabOrder 4339
TabStop 4340
TListView events 4345
TListView Members 4285
TListView properties 4298
ViewStyle 4340
Visible 4343
TListViewActionLink class 4388
about TListViewActionLink class 4388
AddItem 4391
AddItem method 4391
SetImages 4392
TListViewActionLink Members 4389
TListViewActionLink methods 4391
TLocalAppServer class 18553
about TLocalAppServer class 18553
AS_ApplyUpdates 18557
AS_DataRequest 18558
AS_Execute 18559
AS_GetParams 18560
AS_GetProviderNames 18560
AS_GetRecords 18561
AS_RowRequest 18562
Create 18555, 18556
Create constructor 18555

wwww

Destroy 18556
FProvider 18557
FProviderCreated 18557
GetIDsOfNames 18563
GetTypeInfo 18563
GetTypeInfoCount 18563
InterfaceSupportsErrorInfo 18563
Invoke 18564
SafeCallException 18564
TLocalAppServer fields 18556
TLocalAppServer Members 18554
TLocalAppServer methods 18557
TLocalConnection class 23236
about TLocalConnection class 23236
AppServer 23250
AS_ApplyUpdates 23242
AS_DataRequest 23242
AS_Execute 23243
AS_GetParams 23244
AS_GetProviderNames 23245
AS_GetRecords 23245
AS_RowRequest 23246
Create 23240
Destroy 23241
GetConnected 23247
GetProvider 23247
GetProviderNames 23248
GetServer 23248
InterfaceSupportsErrorInfo 23248
Notification 23248
ProviderCount 23250
Providers 23251
RegisterProvider 23249
SafeCallException 23249
TLocalConnection Members 23237
TLocalConnection methods 23241
TLocalConnection properties 23250
UnRegisterProvider 23250
TLocateFileService class 24928
about TLocateFileService class 24928
OnFindIncludeFile 24932
OnFindStream 24932
OnFindTemplateFile 24932
TLocateFileService events 24931
TLocateFileService Members 24928
TLocateParams class 8090
about TLocateParams class 8090
AdapterName 8094
AddParam 8093
AddParam method 8093
Count 8094
Create 8092
Destroy 8092
Items 8095
ParamNames 8095
ParamValues 8096
TLocateParams Members 8091
TLocateParams methods 8093
TLocateParams properties 8094
TLocateParamsList class 8096
about TLocateParamsList class 8096
Add 8099
Clear 8099
Count 8100
Create 8098
Destroy 8098
ItemByAdapterName 8100
Items 8101
TLocateParamsList Members 8097
TLocateParamsList methods 8099
TLocateParamsList properties 8100
TLogicalPageList class 24492
about TLogicalPageList class 24492
Create 24500
TLogicalPageList Members 24493
TLoginDialog class 9776
about TLoginDialog class 9776
Bevel 9793

CancelButton 9793
DatabaseName 9793
FormShow 9795
Label1 9794
Label2 9794
Label3 9794
OKButton 9794
Panel 9794
Panel1 9795
Password 9795
TLoginDialog fields 9793
TLoginDialog Members 9776
TLoginDialog methods 9795
UserName 9795
TLoginFormAdapter class 24500
about TLoginFormAdapter class 24500
Actions 24509
Data 24509
OnAfterExecuteAction 24510
OnAfterGetActionResponse 24511
OnBeforeExecuteAction 24511
OnBeforeGetActionResponse 24511
OnGetActionParams 24512
OnLogin 24512
OnValidatePassword 24513
OnValidateUserName 24513
PasswordRequired 24509
TLoginFormAdapter events 24510
TLoginFormAdapter Members 24502
TLoginFormAdapter properties 24509
TLoginFormAdapterLoginAction class 24514
about TLoginFormAdapterLoginAction class 24514
DisplayLabel 24522
GetDefaultActionName 24520
ImplExecuteActionRequest 24521
ImplGetAdapterItemRequestParams 24521
NextPage 24522
OnAfterExecute 24523
OnAfterGetResponse 24523
OnBeforeExecute 24524
OnBeforeGetResponse 24524
OnGetParams 24525
TLoginFormAdapterLoginAction events 24523
TLoginFormAdapterLoginAction Members 24515
TLoginFormAdapterLoginAction methods 24520
TLoginFormAdapterLoginAction properties 24522
TLogLinkedWebNode class 20005
about TLogLinkedWebNode class 20005
Create 20008
Destroy 20009
Execute 20009
GetResponseStream 20010
ReqFile 20010
RespFile 20010
TLogLinkedWebNode Members 20006
TLogLinkedWebNode methods 20009
TLogLinkedWebNode properties 20010
TLookupList class 7336
about TLookupList class 7336
Add 7338
Clear 7339
Create 7337
Destroy 7338
TLookupList Members 7336
TLookupList methods 7338
ValueOfKey 7339
tlTop enumeration member 21951
TMainMenu class 16056
about TMainMenu class 16056
AutoHotkeys 16063
AutoLineReduction 16064
AutoMerge 16065
BiDiMode 16065
GetHandle 16060
GetOle2AcceleratorTable 16060
Images 16065
MenuChanged 16061
Merge 16061

OnChange 16067
OwnerDraw 16066
ParentBiDiMode 16066
PopulateOle2Menu 16062
SetOle2MenuHandle 16062
TMainMenu events 16067
TMainMenu Members 16056
TMainMenu methods 16060
TMainMenu properties 16063
Unmerge 16063
tmApartment enumeration member 5604
TMargins class 6087
about TMargins class 6087
AssignTo 6090
Bottom 6093
Change 6091
Control 6093
ControlHeight 6093
ControlLeft 6093
ControlTop 6094
ControlWidth 6094
Create 6089
ExplicitHeight 6094
ExplicitLeft 6094
ExplicitTop 6095
ExplicitWidth 6095
GetControlBound 6091
InitDefaults 6091
Left 6095
OnChange 6096
Right 6095
SetBounds 6091
SetControlBounds 6092
SetControlBounds method 6091
TMargins events 6096
TMargins Members 6088
TMargins methods 6090
TMargins properties 6092
Top 6095
TMask class 15898
about TMask class 15898
Create 15899
Destroy 15900
Matches 15900
TMask Members 15899
TMask methods 15900
TMaskEdit class 15824
about TMaskEdit class 15824
Align 15837
Anchors 15840
AutoSelect 15841
AutoSize 15841
BevelEdges 15842
BevelInner 15843
BevelKind 15843
BevelOuter 15843
BevelWidth 15844
BiDiMode 15844
BorderStyle 15845
CharCase 15845
Color 15845
Constraints 15846
Ctl3D 15847
DragCursor 15847
DragKind 15848
DragMode 15848
EditMask 15850
Enabled 15850
Font 15851
ImeMode 15852
ImeName 15852
MaxLength 15853
OnChange 15866
OnClick 15866
OnDblClick 15868
OnDragDrop 15869
OnDragOver 15871
OnEndDock 15873

OnEndDrag 15874
OnEnter 15875
OnExit 15877
OnKeyDown 15879
OnKeyPress 15882
OnKeyUp 15883
OnMouseActivate 15884
OnMouseDown 15885
OnMouseEnter 15888
OnMouseLeave 15888
OnMouseMove 15888
OnMouseUp 15890
OnStartDock 15893
OnStartDrag 15893
ParentBiDiMode 15853
ParentColor 15854
ParentCtl3D 15854
ParentFont 15854
ParentShowHint 15855
PasswordChar 15856
PopupMenu 15856
ReadOnly 15861
ShowHint 15861
TabOrder 15862
TabStop 15863
Text 15863
TMaskEdit events 15865
TMaskEdit Members 15824
TMaskEdit properties 15836
Visible 15864
TMasterDataLink class 7339
about TMasterDataLink class 7339
ActiveChanged 7343
CheckBrowseMode 7344
Create 7342
Destroy 7342
FieldNames 7345
Fields 7346
GetDetailDataSet 7344
LayoutChanged 7344
OnMasterChange 7347
OnMasterDisable 7347
RecordChanged 7345
TMasterDataLink events 7346
TMasterDataLink Members 7340
TMasterDataLink methods 7343
TMasterDataLink properties 7345
tmAuto enumeration member 14643
tmBoth enumeration member 5468, 5604
tmBottomRight enumeration member 5468
TMediaPlayer class 17167
about TMediaPlayer class 17167
Anchors 17197
AutoButtonSet 17181
AutoEnable 17197
AutoOpen 17198
AutoRewind 17199
Back 17182
Capabilities 17199
Click 17182
Close 17183
ColoredButtons 17200
Constraints 17200
Create 17180
Destroy 17180
DeviceID 17201
DeviceType 17201
Display 17202
DisplayRect 17202
DoNotify 17184
Eject 17184
Enabled 17203
EnabledButtons 17203
EndPos 17204
Error 17204
ErrorMessage 17205
FileName 17205
Frames 17205

KeyDown 17186
Length 17206
Loaded 17186
MMNotify 17187
Mode 17206
Next 17187
Notification 17188
Notify 17206
NotifyValue 17207
OnClick 17222
OnContextPopup 17222
OnEnter 17223
OnExit 17226
OnMouseEnter 17227
OnMouseLeave 17227
OnNotify 17228
OnPostClick 17228
Open 17188
Paint 17189
ParentShowHint 17208
Pause 17189
PauseOnly 17190
Play 17190
PopupMenu 17208
Position 17213
PostClick 17191
Previous 17191
Resume 17192
Rewind 17193
Save 17193
Shareable 17213
ShowHint 17214
Start 17215
StartPos 17216
StartRecording 17194
Step 17194
Stop 17195
TabOrder 17216
TabStop 17217
TimeFormat 17217
TMediaPlayer events 17221
TMediaPlayer Members 17169
TMediaPlayer methods 17181
TMediaPlayer properties 17196
TrackLength 17218
TrackPosition 17218
Tracks 17218
Updated 17195
Visible 17219
VisibleButtons 17220
Wait 17221
TMemIniFile class 15462
about TMemIniFile class 15462
CaseSensitive 15471
Clear 15465
Create 15464
DeleteKey 15465
Destroy 15464
EraseSection 15466
GetStrings 15466
ReadSection 15467
ReadSections 15467
ReadSectionValues 15468
ReadString 15468
Rename 15469
SetStrings 15469
TMemIniFile Members 15462
TMemIniFile methods 15465
TMemIniFile properties 15471
UpdateFile 15470
WriteString 15470
TMemo class 21678
about TMemo class 21678
Align 21691
Alignment 21695
Anchors 21695
BevelEdges 21696
BevelInner 21696

BevelKind 21696
BevelOuter 21697
BiDiMode 21697
BorderStyle 21698
Color 21698
Constraints 21699
Ctl3D 21699
DragCursor 21700
DragKind 21700
DragMode 21701
Enabled 21702
Font 21703
HideSelection 21704
ImeMode 21704
ImeName 21705
Lines 21705
MaxLength 21706
OEMConvert 21707
OnChange 21724
OnClick 21725
OnContextPopup 21727
OnDblClick 21727
OnDragDrop 21728
OnDragOver 21730
OnEndDock 21732
OnEndDrag 21733
OnEnter 21734
OnExit 21736
OnKeyDown 21738
OnKeyPress 21741
OnKeyUp 21742
OnMouseActivate 21743
OnMouseDown 21744
OnMouseEnter 21747
OnMouseLeave 21747
OnMouseMove 21747
OnMouseUp 21749
OnStartDock 21752
OnStartDrag 21752
ParentBiDiMode 21707
ParentColor 21708
ParentCtl3D 21708
ParentFont 21709
ParentShowHint 21709
PopupMenu 21710
ReadOnly 21714
ScrollBars 21715
ShowHint 21717
TabOrder 21718
TabStop 21719
TMemo events 21724
TMemo Members 21679
TMemo properties 21690
Visible 21719
WantReturns 21721
WantTabs 21721
WordWrap 21722
TMemoField class 7348
about TMemoField class 7348
Create 7354
GetAsWideString 7355
SetAsWideString 7356
TMemoField Members 7348
TMemoField methods 7355
TMemoField properties 7356
Transliterate 7356
TMemoryStream class 2902
about TMemoryStream class 2902
Capacity 2912
Clear 2909
Destroy 2908
LoadFromFile 2909
LoadFromStream 2909
Realloc 2910
SetSize 2910
TMemoryStream Members 2906
TMemoryStream methods 2908
TMemoryStream properties 2911

Write 2911
TMenu class 16067
about TMenu class 16067
AdjustBiDiBehavior 16073
AutoHotkeys 16082
AutoLineReduction 16082
BiDiMode 16083
Create 16072
CreateMenuItem 16073
Destroy 16072
DispatchCommand 16074
DispatchPopup 16074
DoBiDiModeChanged 16074
DoChange 16075
DoGetMenuString 16075
FindItem 16076
GetChildren 16076
GetHandle 16076
GetHelpContext 16077
Handle 16083
Images 16083
IsOwnerDraw 16077
IsRightToLeft 16077
IsShortCut 16078
Items 16084
Loaded 16078
MenuChanged 16079
Notification 16079
OnChange 16090
OwnerDraw 16088
ParentBiDiMode 16089
ParentBiDiModeChanged 16080
ParentBiDiModeChanged method 16079
ProcessMenuChar 16080
SetChildOrder 16081
TMenu events 16089
TMenu Members 16068
TMenu methods 16072
TMenu properties 16081
UpdateItems 16081
WindowHandle 16089
TMenuActionLink class 16090
about TMenuActionLink class 16090
AssignClient 16093
FClient 16093
IsAutoCheckLinked 16094
IsCaptionLinked 16094
IsCheckedLinked 16095
IsEnabledLinked 16095
IsGroupIndexLinked 16095
IsHelpContextLinked 16096
IsHintLinked 16096
IsImageIndexLinked 16097
IsOnExecuteLinked 16097
IsShortCutLinked 16097
IsVisibleLinked 16098
SetAutoCheck 16098
SetCaption 16099
SetChecked 16099
SetEnabled 16099
SetHelpContext 16100
SetHint 16100
SetImageIndex 16100
SetOnExecute 16101
SetShortCut 16101
SetVisible 16102
TMenuActionLink fields 16092
TMenuActionLink Members 16091
TMenuActionLink methods 16093
TMenuItem class 16102
about TMenuItem class 16102
Action 16133
ActionChange 16110
ActionLink 16134
Add 16110
Add method 16110
AdvancedDrawItem 16115
AssignTo 16115

AutoCheck 16134
AutoHotkeys 16134
AutoLineReduction 16135
Bitmap 16135
Break 16136
Caption 16136
Checked 16140
Clear 16115
Click 16116
Command 16141
Count 16141
Create 16107
Default 16145
DefineProperties 16116
Delete 16117
Destroy 16108
DoDrawText 16117
DrawItem 16118
Enabled 16145
Find 16118
GetActionLinkClass 16118
GetAutoHotkeys 16118
GetAutoLineReduction 16119
GetChildren 16119
GetCount 16119
GetEnumerator 16119
GetHandle 16120
GetImageList 16120
GetItem 16120
GetMenuIndex 16121
GetParentComponent 16121
GetParentMenu 16121
GroupIndex 16146
Handle 16147
HasParent 16121
HelpContext 16148
Hint 16148
ImageIndex 16148
IndexOf 16122
InitiateAction 16122
Insert 16123
InsertNewLine 16124
InsertNewLineAfter 16124
InsertNewLineBefore 16124
IsLine 16125
Items 16149
Loaded 16125
MeasureItem 16126
MenuChanged 16126
MenuIndex 16150
NewBottomLine 16126
NewTopLine 16127
Notification 16127
OnAdvancedDrawItem 16154
OnClick 16154
OnDrawItem 16159
OnMeasureItem 16160
Parent 16151
RadioItem 16151
Remove 16128
RethinkHotkeys 16128
RethinkLines 16129
SetBreak 16129
SetCaption 16130
SetChecked 16130
SetChildOrder 16130
SetDefault 16130
SetEnabled 16131
SetGroupIndex 16131
SetImageIndex 16131
SetMenuIndex 16131
SetParentComponent 16131
SetRadioItem 16132
SetShortCut 16132
SetVisible 16132
ShortCut 16152
SubMenuImages 16152
TMenuItem events 16153

eeee

TMenuItem Members 16103
TMenuItem methods 16109
TMenuItem properties 16132
UpdateItems 16132
Visible 16153
TMenuItemEnumerator class 16160
about TMenuItemEnumerator class 16160
Create 16161
Current 16162
GetCurrent 16161
MoveNext 16162
TMenuItemEnumerator Members 16161
TMenuItemEnumerator methods 16161
TMenuItemEnumerator properties 16162
TMenuItemStack class 16162
about TMenuItemStack class 16162
ClearItem 16164
TMenuItemStack Members 16163
TMenuItemStack methods 16164
TMenuStack class 833
about TMenuStack class 833
Bars 836
Create 834
Peek 834
Pop 835
Push 835
TMenuStack Members 833
TMenuStack methods 834
TMenuStack properties 835
TMessage class 25708
about TMessage class 25708
AfterConstruction 25714
GetName 25714
GetParts 25714
SetName 25714
TMessage Members 25709
TMessage methods 25713
TMessages class 25715
about TMessages class 25715
Add 25720
GetMessage 25721
Insert 25721
TMessages Members 25715
TMessages methods 25720
TMetaClass class 22281
about TMetaClass class 22281
ClassInfo 22283
ClassName 22283
ClassNameIs 22284
ClassParent 22284
InheritsFrom 22284
InitInstance 22284
InstanceSize 22285
MethodAddress 22285
MethodName 22285
TMetaClass Members 22283
TMetaClass methods 22283
TMetafile class 14549
about TMetafile class 14549
Assign 14554
Clear 14555
Create 14551
CreatedBy 14561
Description 14562
Destroy 14553
Draw 14555
Enhanced 14562
GetEmpty 14555
GetHeight 14555
GetPalette 14556
GetWidth 14556
Handle 14563
HandleAllocated 14556
Inch 14563
LoadFromClipboardFormat 14556
LoadFromStream 14557
MMHeight 14563
MMWidth 14564

ReadData 14557
ReadEMFStream 14557
ReadWMFStream 14558
ReleaseHandle 14558
SaveToClipboardFormat 14558
SaveToFile 14559
SaveToStream 14559
SetHeight 14559
SetSize 14560
SetTransparent 14560
SetWidth 14560
TestEMF 14560
TMetafile Members 14549
TMetafile methods 14554
TMetafile properties 14561
WriteData 14560
WriteEMFStream 14561
WriteWMFStream 14561
TMetafileCanvas class 14564
about TMetafileCanvas class 14564
Create 14567
CreateWithComment 14568
Destroy 14569
TMetafileCanvas Members 14564
TMetafileImage class 14569
about TMetafileImage class 14569
Destroy 14570
FreeHandle 14571
TMetafileImage Members 14570
TMetafileImage methods 14571
TMethodProperty class 11004
about TMethodProperty class 11004
AllEqual 11007
AllNamed 11008
Edit 11008
GetAttributes 11008
GetEditLimit 11009
GetFormMethodName 11009
GetTrimmedEventName 11009
GetValue 11010
GetValues 11010
SetValue 11011
TMethodProperty Members 11004
TMethodProperty methods 11007
tmFixed enumeration member 14643
tmFree enumeration member 5604
tmHide enumeration member 14367
TMidasConnection class 16462
about TMidasConnection class 16462
ConnectType 16469
Create 16466
DoConnect 16467
DoDisconnect 16468
ServerPort 16469
SetConnectType 16468
SetUseBroker 16468
TMidasConnection Members 16462
TMidasConnection methods 16467
TMidasConnection properties 16468
UseBroker 16470
TMidasPageProducer class 17158
about TMidasPageProducer class 17158
tmMultiThreaded enumeration member 6484
tmNeutral enumeration member 5604
TMonitor class 14191
about TMonitor class 14191
BoundsRect 14193
Handle 14194
Height 14194
Left 14194
MonitorNum 14195
Primary 14195
TMonitor Members 14192
TMonitor properties 14193
Top 14196
Width 14196
WorkareaRect 14197
TMonthCalColors class 4392

about TMonthCalColors class 4392
Assign 4394
BackColor 4395
Create 4394
MonthBackColor 4395
TextColor 4395
TitleBackColor 4396
TitleTextColor 4396
TMonthCalColors Members 4392
TMonthCalColors methods 4394
TMonthCalColors properties 4394
TrailingTextColor 4396
TMonthCalendar class 4397
about TMonthCalendar class 4397
Align 4412
Anchors 4416
AutoSize 4416
BiDiMode 4417
BorderWidth 4417
CalColors 4417
CanAutoSize 4408
ConstrainedResize 4409
Constraints 4418
Create 4408
CreateParams 4409
Date 4418
DragCursor 4419
DragKind 4419
DragMode 4420
Enabled 4421
EndDate 4422
FirstDayOfWeek 4422
Font 4422
GetCalendarHandle 4410
ImeMode 4423
ImeName 4424
MaxDate 4425
MaxSelectRange 4425
MinDate 4425
MsgSetCalColors 4410
MsgSetDateTime 4410
MsgSetRange 4411
MultiSelect 4426
OnClick 4438
OnContextPopup 4440
OnDblClick 4440
OnDragDrop 4441
OnDragOver 4443
OnEndDock 4445
OnEndDrag 4446
OnEnter 4447
OnExit 4449
OnGetMonthInfo 4451
OnKeyDown 4451
OnKeyPress 4454
OnKeyUp 4455
OnMouseEnter 4457
OnMouseLeave 4457
OnStartDock 4457
OnStartDrag 4458
ParentBiDiMode 4426
ParentFont 4427
ParentShowHint 4427
PopupMenu 4428
ShowHint 4432
ShowToday 4434
ShowTodayCircle 4434
TabOrder 4434
TabStop 4435
TMonthCalendar events 4437
TMonthCalendar Members 4397
TMonthCalendar methods 4408
TMonthCalendar properties 4411
Visible 4435
WeekNumbers 4437
TMouse class 6096
about TMouse class 6096
Capture 6099

Create 6097
CreatePanningWindow 6098
CursorPos 6100
Destroy 6098
DragImmediate 6100
DragThreshold 6101
IsDragging 6101
IsPanning 6101
MousePresent 6101
PanningWindow 6102
PanningWindowClass 6102
RegWheelMessage 6102
SettingChanged 6098
TMouse Members 6097
TMouse methods 6098
TMouse properties 6099
WheelPresent 6103
WheelScrollLines 6103
tmShow enumeration member 14367
tmSingle enumeration member 5604
tmSingleThread enumeration member 6484
tmTopLeft enumeration member 5468
TMtsAutoObject class 17240
about TMtsAutoObject class 17240
Activate 17243
CanBePooled 17243
Deactivate 17243
DisableCommit 17243
EnableCommit 17244
IsCallerInRole 17245
IsInTransaction 17245
IsSecurityEnabled 17246
ObjectContext 17247
OnActivate 17246
OnDeactivate 17246
Pooled 17248
SetAbort 17246
SetComplete 17247
TMtsAutoObject Members 17240
TMtsAutoObject methods 17242
TMtsAutoObject properties 17247
TMtsDataModule class 17250
about TMtsDataModule class 17250
Activate 17255
AS_ApplyUpdates 17255
AS_DataRequest 17256
AS_Execute 17257
AS_GetParams 17258
AS_GetProviderNames 17258
AS_GetRecords 17259
AS_RowRequest 17260
AutoComplete 17265
CallComplete 17261
CanBePooled 17261
Create 17254
Deactivate 17261
DisableCommit 17261
EnableCommit 17262
IsCallerInRole 17263
IsInTransaction 17263
IsSecurityEnabled 17264
ObjectContext 17266
OnActivate 17267
OnDeactivate 17267
Pooled 17266
SetAbort 17264
SetComplete 17265
TMtsDataModule events 17267
TMtsDataModule Members 17251
TMtsDataModule methods 17255
TMtsDataModule properties 17265
TMultiDimDataLink class 17822
about TMultiDimDataLink class 17822
ActiveChanged 17825
AddDimension 17825
AddSummary 17826
Create 17824
CreateSummaryIndex 17826

Destroy 17824
FetchAndBinValues 17826
FetchValues 17826
LayoutChanged 17827
TMultiDimDataLink Members 17822
TMultiDimDataLink methods 17825
UpdateCache 17827
UpdateDimensions 17827
UpdateFormatStrings 17827
TMultiModuleWebAppServices class 24933
about TMultiModuleWebAppServices class 24933
Access_Request 24937
Access_Response 24938
CreateContext 24938
DoAfterDispatch 24939
DoBeforeDispatch 24939
DoneModule 24939
FinishContext 24940
GetExceptionHandler 24940
HandleException 24941
HandleRequest 24941
ImplFinishContext 24942
ImplHandleException 24942
ImplHandleRequest 24943
ImplInitContext 24943
InitContext 24944
InitModule 24945
InitModules 24945
ModuleAdded 24946
OnAfterDispatch 24948
OnBeforeDispatch 24948
Request 24946
Response 24947
TMultiModuleWebAppServices events 24947
TMultiModuleWebAppServices Members 24934
TMultiModuleWebAppServices methods 24937
TMultiModuleWebAppServices properties 24946
WebModules 24947
TMultipartContentParser class 18710
about TMultipartContentParser class 18710
BufferContent 18712
CanParse 18713
Destroy 18712
ExtractContentTypeFields 18713
ExtractMultipartContentFields 18713
GetContentFields 18714
GetFiles 18714
ParseMultiPart 18715
ParseMultiPartContent 18715
ParseMultipartHeaders 18715
TMultipartContentParser Members 18711
TMultipartContentParser methods 18712
TMultiReadExclusiveWriteSynchronizer class 22766
about TMultiReadExclusiveWriteSynchronizer class 22766
BeginRead 22769
BeginWrite 22770
Create 22769
Destroy 22769
EndRead 22770
EndWrite 22771
RevisionLevel 22771
TMultiReadExclusiveWriteSynchronizer Members 22767
TMultiReadExclusiveWriteSynchronizer methods 22769
TMultiReadExclusiveWriteSynchronizer properties 22771
TMultiRecordActionRequestImpl class 909
about TMultiRecordActionRequestImpl class 909
ActionFieldValues 913
ActionFieldValues2 914
ActionFieldValuesList 914
AdapterActionFieldValuesList 914
AdapterModes 914
Create 912
Destroy 912
TMultiRecordActionRequestImpl Members 910
TMultiRecordActionRequestImpl properties 913
TMutex class 22071
about TMutex class 22071
Acquire 22073

jjjj

Create 22072, 22073
Create constructor 22072
Release 22074
TMutex Members 22071
TMutex methods 22073
TNamedItem class 7357
about TNamedItem class 7357
GetDisplayName 7359
Name 7359
SetDisplayName 7359
TNamedItem Members 7357
TNamedItem methods 7359
TNamedItem properties 7359
TNamedVariantsList class 24824
about TNamedVariantsList class 24824
Clear 24827
Delete 24827
Destroy 24826
Get 24828
GetCapacity 24828
GetCount 24829
Insert 24829
Put 24830
SetCapacity 24830
TNamedVariantsList Members 24825
TNamedVariantsList methods 24827
TNavButton class 9502
about TNavButton class 9502
Destroy 9511
Index 9513
MouseDown 9511
MouseUp 9512
NavStyle 9513
Paint 9512
TNavButton Members 9503
TNavButton methods 9511
TNavButton properties 9513
TNavDataLink class 9513
about TNavDataLink class 9513
ActiveChanged 9516
Create 9515
DataSetChanged 9517
Destroy 9516
EditingChanged 9517
TNavDataLink Members 9514
TNavDataLink methods 9516
TNestedProperty class 11011
about TNestedProperty class 11011
Create 11014
Destroy 11014
TNestedProperty Members 11011
TNestedTable class 10040
about TNestedTable class 10040
Create 10053
CreateHandle 10053
DataSetField 10054
DoAfterPost 10054
DoBeforeInsert 10054
InternalPost 10054
ObjectView 10055
TNestedTable Members 10041
TNestedTable methods 10053
TNestedTable properties 10054
TNestedTableSprig class 1743
about TNestedTableSprig class 1743
ParentProperty 1744
TNestedTableSprig Members 1743
TNestedTableSprig methods 1744
TNextButton class 16694
about TNextButton class 16694
Create 16699
TNextButton Members 16695
TNextPageButton class 16699
about TNextPageButton class 16699
Create 16703
TNextPageButton Members 16700
TNextTab class 11949
about TNextTab class 11949

AfterTabChange 11961
BeforeTabChange 11962
Caption 11956
Enabled 11956
ExecuteTarget 11954
HelpContext 11956
HelpKeyword 11957
HelpType 11957
Hint 11958
ImageIndex 11958
LastTabCaption 11958
OnFinish 11962
OnHint 11963
OnUpdate 11963
OnValidateTab 11964
ShortCut 11959
SkipHiddenTabs 11959
TabControl 11960
TNextTab events 11961
TNextTab Members 11950
TNextTab methods 11954
TNextTab properties 11955
UpdateTarget 11955
Visible 11960
Wrap 11960
TNotebook class 12902
about TNotebook class 12902
ActivePage 12916
Align 12917
Anchors 12920
Color 12921
Constraints 12921
Create 12912
CreateParams 12913
Ctl3D 12922
Destroy 12913
DragCursor 12923
DragKind 12923
DragMode 12923
Enabled 12925
Font 12925
GetChildOwner 12914
GetChildren 12914
OnClick 12938
OnContextPopup 12940
OnDblClick 12941
OnDragDrop 12941
OnDragOver 12943
OnEndDock 12946
OnEndDrag 12946
OnEnter 12947
OnExit 12950
OnMouseActivate 12951
OnMouseDown 12952
OnMouseEnter 12954
OnMouseLeave 12955
OnMouseMove 12955
OnMouseUp 12957
OnPageChanged 12959
OnStartDock 12960
OnStartDrag 12960
PageIndex 12926
Pages 12927
ParentColor 12927
ParentCtl3D 12928
ParentFont 12928
ParentShowHint 12928
PopupMenu 12929
ReadState 12915
ShowControl 12915
ShowHint 12934
TabOrder 12935
TabStop 12936
TNotebook events 12938
TNotebook Members 12902
TNotebook methods 12913
TNotebook properties 12916
Visible 12936

TNotifyList class 19636
about TNotifyList class 19636
AddNotify 19638
Count 19640
Create 19637
Destroy 19637
GetCount 19638
GetItem 19639
Items 19640
RemoveNotify 19639
TNotifyList Members 19636
TNotifyList methods 19638
TNotifyList properties 19640
TNumericField class 7361
about TNumericField class 7361
Alignment 7369
Create 7367
DisplayFormat 7369
EditFormat 7371
RangeError 7368
SetDisplayFormat 7368
SetEditFormat 7368
TNumericField Members 7362
TNumericField methods 7367
TNumericField properties 7368
TObject class 22285
about TObject class 22285
AfterConstruction 22289
BeforeDestruction 22289
ClassInfo 22290
ClassName 22290
ClassNameIs 22293
ClassParent 22293
ClassType 22295
CleanupInstance 22298
Create 22287
DefaultHandler 22299
Destroy 22287
Dispatch 22299
FieldAddress 22299
Free 22300
FreeInstance 22303
GetInterface 22303
GetInterfaceEntry 22304
GetInterfaceTable 22304
InheritsFrom 22305
InitInstance 22305
InstanceSize 22306
MethodAddress 22306
MethodName 22306
NewInstance 22306
SafeCallException 22307
TObject Members 22286
TObject methods 22288
TObjectBucketList class 5673
about TObjectBucketList class 5673
Add 5675
Data 5677
GetData 5676
Remove 5676
SetData 5676
TObjectBucketList Members 5674
TObjectBucketList methods 5675
TObjectBucketList properties 5676
TObjectField class 7372
about TObjectField class 7372
CheckTypeSize 7380
Create 7378
DefineProperties 7380
Destroy 7379
FieldCount 7384
Fields 7384
FieldValues 7385
FreeBuffers 7380
GetAsString 7381
GetAsVariant 7381
GetChildren 7381
GetDefaultWidth 7381

GetFieldCount 7381
GetFields 7382
GetFieldValue 7382
GetHasConstraints 7382
ObjectType 7385
SetChildOrder 7382
SetDataSet 7383
SetFieldKind 7383
SetFieldValue 7383
SetParentField 7383
SetUnNamed 7383
SetVarValue 7384
TObjectField Members 7373
TObjectField methods 7379
TObjectField properties 7384
UnNamed 7386
TObjectList class 5677
about TObjectList class 5677
Add 5680
Create 5680
Create constructor 5679
Extract 5681
FindInstanceOf 5682
First 5682
GetItem 5682
IndexOf 5683
Insert 5683
Items 5685
Last 5684
Notify 5684
OwnsObjects 5686
Remove 5684
SetItem 5685
TObjectList Members 5678
TObjectList methods 5680
TObjectList properties 5685
TObjectQueue class 5687
about TObjectQueue class 5687
Peek 5689
Pop 5689
Push 5689
TObjectQueue Members 5687
TObjectQueue methods 5688
TObjectStack class 5690
about TObjectStack class 5690
Peek 5692
Pop 5692
Push 5693
TObjectStack Members 5690
TObjectStack methods 5692
TOleContainer class 17921
_AddRef 17935
_Release 17935
about TOleContainer class 17921
Align 17951
AllowActiveDoc 17954
AllowInPlace 17955
Anchors 17955
AutoActivate 17956
AutoVerbMenu 17956
BorderStyle 17957
CanPaste 17957
Caption 17957
Changed 17936
ChangeIconDialog 17936
Close 17937
Color 17960
Constraints 17961
Copy 17937
CopyOnSave 17961
Create 17933
CreateLinkToFile 17937
CreateObject 17938
CreateObjectFromFile 17938
CreateObjectFromInfo 17939
CreateParams 17939
Ctl3D 17962
DblClick 17940

DefineProperties 17940
Destroy 17934
DestroyObject 17941
DoEnter 17941
DoVerb 17941
DragCursor 17962
DragMode 17963
Enabled 17964
GetIconMetaPict 17942
GetPopupMenu 17942
Iconic 17965
InsertObjectDialog 17943
KeyDown 17943
Linked 17965
LoadFromFile 17944
LoadFromStream 17944
Modified 17965
MouseDown 17945
NewInserted 17966
ObjectPropertiesDialog 17945
ObjectVerbs 17966
OldStreamFormat 17966
OleClassName 17967
OleObject 17967
OleObjectInterface 17968
OnActivate 17981
OnDeactivate 17981
OnDragDrop 17982
OnDragOver 17984
OnEndDrag 17986
OnEnter 17987
OnExit 17990
OnKeyDown 17991
OnKeyPress 17994
OnKeyUp 17995
OnMouseActivate 17997
OnMouseDown 17998
OnMouseMove 18000
OnMouseUp 18002
OnObjectMove 18005
OnResize 18005
OnStartDrag 18005
Paint 17945
ParentColor 17968
ParentCtl3D 17969
ParentShowHint 17969
Paste 17946
PasteSpecialDialog 17946
PopupMenu 17969
PrimaryVerb 17974
QueryInterface 17947
Run 17947
SaveAsDocument 17948
SaveToFile 17948
SaveToStream 17949
ShowHint 17974
SizeMode 17976
SourceDoc 17977
State 17977
StorageInterface 17977
TabOrder 17978
TabStop 17978
TOleContainer events 17980
TOleContainer Members 17922
TOleContainer methods 17934
TOleContainer properties 17950
UpdateObject 17949
UpdateVerbs 17949
Visible 17979
TOleControl class 18030
_AddRef 18046
_Release 18047
_SetColorProp 18047
_SetTColorProp 18048
_SetTFontProp 18048
_SetTOleEnumProp 18048
_SetTPictureProp 18048
about TOleControl class 18030

Anchors 18085
BrowseProperties 18048
CanInPlaceActivate 18049
ContextSensitiveHelp 18049
ControlData 18086
Create 18043
CreateWnd 18049
DeactivateAndUndo 18050
DefaultDispatch 18086
DefaultHandler 18050
DefineProperties 18050
Destroy 18043
DestroyWindowHandle 18051
DiscardUndoState 18051
DoObjectVerb 18051
EnableModeless 18052
GetBorder 18052
GetByteProp 18052
GetColorProp 18053
GetCompProp 18053
GetContainer 18053
GetCurrencyProp 18053
GetDoubleProp 18053
GetEnumPropDesc 18054
GetExtendedControl 18054
GetHelpContext 18054
GetIDispatchProp 18055
GetIDsOfNames 18055
GetIntegerProp 18055
GetIUnknownProp 18055
GetMoniker 18056
GetObjectVerbs 18056
GetOleBoolProp 18056
GetOleDateProp 18056
GetOleEnumProp 18057
GetOleVariantProp 18057
GetPropDisplayString 18057
GetPropDisplayStrings 18057
GetProperty 18058
GetShortIntProp 18058
GetSingleProp 18058
GetSmallintProp 18058
GetStringProp 18059
GetTColorProp 18059
GetTDateTimeProp 18059
GetTFontProp 18059
GetTOleEnumProp 18059
GetTPictureProp 18060
GetTypeInfo 18060
GetTypeInfoCount 18060
GetVariantProp 18061
GetWideStringProp 18061
GetWindowContext 18061
GetWordBoolProp 18061
GetWordProp 18062
InitControlData 18062
InitControlInterface 18062
InsertMenus 18062
Invoke 18062
InvokeEvent 18063
InvokeMethod 18063
IsCustomProperty 18063
IsPropPageProperty 18064
LockInPlaceActive 18064
OleControlSite_TranslateAccelerator 18064
OleInPlaceFrame_GetWindow 18064
OleInPlaceFrame_TranslateAccelerator 18065
OleInPlaceSite_GetWindow 18065
OleObject 18086
OnChanged 18065
OnControlInfoChanged 18065
OnFocus 18066
OnInPlaceActivate 18066
OnInPlaceDeactivate 18066
OnPosRectChange 18066
OnRequestEdit 18066
OnShowWindow 18067
OnUIActivate 18067

OnUIDeactivate 18067
PaletteChanged 18067
PerPropBrowsing 18086
PictureChanged 18068
PostMessageFilter 18068
PreMessageFilter 18069
QueryInterface 18069
QueryService 18069
RemoveMenus 18070
RequestBorderSpace 18070
RequestNewObjectLayout 18070
SaveObject 18070
Scroll 18070
ServiceQuery 18087
SetActiveObject 18071
SetBorderSpace 18071
SetBounds 18071
SetByteProp 18071
SetColorProp 18072
SetCompProp 18072
SetCurrencyProp 18072
SetDoubleProp 18072
SetIDispatchProp 18073
SetIntegerProp 18073
SetIUnknownProp 18073
SetMenu 18073
SetName 18073
SetOleBoolProp 18074
SetOleDateProp 18074
SetOleEnumProp 18074
SetOleVariantProp 18075
SetParent 18075
SetPropDisplayString 18075
SetProperty 18076
SetShortIntProp 18076
SetSingleProp 18076
SetSmallintProp 18076
SetStatusText 18077
SetStringProp 18077
SetTColorProp 18077
SetTDateTimeProp 18077
SetTFontProp 18077
SetTOleEnumProp 18078
SetTPictureProp 18078
SetVariantProp 18078
SetWideStringProp 18078
SetWordBoolProp 18079
SetWordProp 18079
ShowAboutBox 18079
ShowObject 18079
ShowPropertyFrame 18079
StandardEvent 18080
SuppressException 18080
TabStop 18087
TOleControl events 18087
TOleControl Members 18031
TOleControl methods 18044
TOleControl properties 18085
TransformCoords 18080
WndProc 18080
TOleForm class 18006
about TOleForm class 18006
Create 18008
Destroy 18008
TOleForm Members 18007
TOleGraphic class 1587
about TOleGraphic class 1587
Assign 1590
Changed 1590
Draw 1590
GetEmpty 1591
GetHeight 1591
GetPalette 1591
GetTransparent 1591
GetWidth 1591
LoadFromClipboardFormat 1592
LoadFromFile 1592
LoadFromStream 1592

MMHeight 1594
MMWidth 1595
Picture 1595
SaveToClipboardFormat 1593
SaveToStream 1593
SetHeight 1593
SetPalette 1594
SetWidth 1594
TOleGraphic Members 1587
TOleGraphic methods 1589
TOleGraphic properties 1594
TOLEnterpriseConnection class 16033
about TOLEnterpriseConnection class 16033
BrokerName 16038
ComputerName 16039
DoConnect 16038
TOLEnterpriseConnection Members 16034
TOLEnterpriseConnection methods 16038
TOLEnterpriseConnection properties 16038
TOleServer class 18098
_AddRef 18104
_Release 18104
about TOleServer class 18098
AutoConnect 18110
Connect 18104
ConnectEvents 18105
ConnectKind 18110
Create 18101
Destroy 18102
Disconnect 18105
DisconnectEvents 18106
EventDispatch 18111
GetAutoConnect 18106
GetConnectKind 18106
GetServer 18107
InitServerData 18107
InvokeEvent 18107
Loaded 18108
QueryInterface 18108
RemoteMachineName 18111
ServerData 18111
SetAutoConnect 18109
SetConnectKind 18109
TOleServer Members 18099
TOleServer methods 18103
TOleServer properties 18109
TOleStream class 1595
about TOleStream class 1595
Create 1597
GetIStream 1598
Read 1598
Seek 1598
TOleStream Members 1596
TOleStream methods 1597
Write 1599
ToolWin 23253
ToolWin namespace 23253
Classes 23254
Structs, Records, Enums 23289
Types 23292
ToolWin.TEdgeBorder 23289
ToolWin.TEdgeBorder enumeration 23289
ToolWin.TEdgeBorders 23292
ToolWin.TEdgeBorders type 23292
ToolWin.TEdgeStyle 23290
ToolWin.TEdgeStyle enumeration 23290
ToolWin.TSizingOrientation 23290
ToolWin.TSizingOrientation enumeration 23290
ToolWin.TToolDockForm 23254
ToolWin.TToolDockForm.CanResize 23269
ToolWin.TToolDockForm.Create 23268
ToolWin.TToolDockForm.CreateParams 23269
ToolWin.TToolDockForm.DoAddDockClient 23269
ToolWin.TToolDockForm.WMNCCreate 23270
ToolWin.TToolDockForm.WMNCHitTest 23270
ToolWin.TToolDockForm.WMNCLButtonDown 23270
ToolWin.TToolDockForm.WMSize 23270
ToolWin.TToolDockForm.WMSysCommand 23271

ToolWin.TToolDockObject 23271
ToolWin.TToolDockObject.AdjustDockRect 23273
ToolWin.TToolDockObject.Create 23273
ToolWin.TToolDockObject.DrawDragDockImage 23274
ToolWin.TToolDockObject.EraseDragDockImage 23274
ToolWin.TToolWindow 23274
ToolWin.TToolWindow.CMBorderChanged 23285
ToolWin.TToolWindow.CMCtl3DChanged 23286
ToolWin.TToolWindow.Create 23285
ToolWin.TToolWindow.EdgeBorders 23287
ToolWin.TToolWindow.EdgeInner 23287
ToolWin.TToolWindow.EdgeOuter 23288
ToolWin.TToolWindow.NCPaint 23286
ToolWin.TToolWindow.WMNCCalcSize 23286
ToolWin.TToolWindow.WMNCPaint 23286
TOpenDialog class 11407
about TOpenDialog class 11407
CanClose 11413
Create 11411
DefaultExt 11421
DefineProperties 11413
Destroy 11412
DoCanClose 11413
DoExecute 11414
DoExecute method 11413
DoFolderChange 11414
DoIncludeItem 11414
DoSelectionChange 11415
DoTypeChange 11415
Execute 11415
FileEditStyle 11423
FileName 11424
Files 11429
Filter 11431
FilterIndex 11434
GetFileNames 11417
GetStaticRect 11417
HistoryList 11436
InitialDir 11436
OnCanClose 11439
OnFolderChange 11440
OnIncludeItem 11440
OnSelectionChange 11441
OnTypeChange 11441
Options 11436
OptionsEx 11438
Title 11438
TOpenDialog events 11439
TOpenDialog Members 11408
TOpenDialog methods 11412
TOpenDialog properties 11420
WndProc 11420
TOpenPicture class 11965
about TOpenPicture class 11965
Caption 11970
Dialog 11971
Enabled 11971
GetDialogClass 11970
HelpContext 11971
HelpKeyword 11972
HelpType 11972
Hint 11973
ImageIndex 11973
OnAccept 11975
OnCancel 11975
OnHint 11975
OnUpdate 11976
SecondaryShortCuts 11974
ShortCut 11974
TOpenPicture events 11975
TOpenPicture Members 11965
TOpenPicture methods 11970
TOpenPicture properties 11970
Visible 11974
TOpenPictureDialog class 13282
about TOpenPictureDialog class 13282
Create 13286
DoClose 13287

DoSelectionChange 13287
DoShow 13287
Execute 13288
Filter 13289
ImageCtrl 13292
PictureLabel 13292
PreviewClick 13288
TOpenPictureDialog Members 13283
TOpenPictureDialog methods 13287
TOpenPictureDialog properties 13288
TOpenTextFileDialog class 13292
about TOpenTextFileDialog class 13292
Create 13296
Destroy 13296
DoClose 13297
DoShow 13297
EncodingIndex 13300
Encodings 13300
Execute 13298
TOpenTextFileDialog Members 13292
TOpenTextFileDialog methods 13297
TOpenTextFileDialog properties 13300
TOperation class 25721
about TOperation class 25721
AfterConstruction 25727
GetFaults 25727
GetInput 25727
GetName 25727
GetOutput 25728
SetName 25728
TOperation Members 25722
TOperation methods 25726
TOperations class 25728
about TOperations class 25728
Add 25734
Add method 25734
GetOperation 25735
Insert 25735
TOperations Members 25729
TOperations methods 25734
TOPToSoapDomConvert class 18137
about TOPToSoapDomConvert class 18137
Attachments 18148
Create 18143
Destroy 18143
Encoding 18149
InvContextToMsg 18144
MakeFault 18145
MakeResponse 18145
MsgToInvContext 18146
MsgToInvContext method 18146
NewXMLDocument 18147
ProcessResponse 18147
ProcessResponse method 18147
TempDir 18149
TOPToSoapDomConvert Members 18138
TOPToSoapDomConvert methods 18144
TOPToSoapDomConvert properties 18148
WSDLView 18150
TORB class 6467
about TORB class 6467
Bind 6469
Bind method 6468
FindTypeCode 6469
Initialize 6470
Initialize method 6469
MakeAlias 6470
MakeAliasTypeCode 6470
MakeArray 6471
MakeArray method 6471
MakeObjectRefTypeCode 6471
MakeSequence 6472
MakeSequence method 6471
MakeSequenceTypeCode 6472
MakeStructure 6472
MakeStructureTypeCode 6473
MakeTypeCode 6473
ObjectToString 6473

Shutdown 6473
StringToObject 6473
TORB Members 6468
TORB methods 6468
TOrderedList class 5693
about TOrderedList class 5693
AtLeast 5696
Count 5696
Create 5695
Destroy 5695
List 5699
Peek 5696
PeekItem 5697
Pop 5697
PopItem 5697
Push 5698
PushItem 5698
TOrderedList Members 5694
TOrderedList methods 5695
TOrderedList properties 5699
TOrdinalProperty class 11015
about TOrdinalProperty class 11015
AllEqual 11018
GetEditLimit 11018
TOrdinalProperty Members 11015
TOrdinalProperty methods 11018
TOutline class 18279
about TOutline class 18279
Align 18294
BorderStyle 18298
Color 18298
Ctl3D 18299
DragCursor 18299
DragKind 18299
DragMode 18300
Enabled 18301
Font 18302
ItemHeight 18303
ItemSeparator 18303
Lines 18304
OnClick 18320
OnCollapse 18322
OnContextPopup 18322
OnDblClick 18323
OnDragDrop 18323
OnDragOver 18325
OnDrawItem 18328
OnEndDock 18328
OnEndDrag 18329
OnEnter 18330
OnExit 18332
OnExpand 18334
OnKeyDown 18334
OnKeyPress 18337
OnKeyUp 18338
OnMouseActivate 18339
OnMouseDown 18340
OnMouseEnter 18343
OnMouseLeave 18343
OnMouseMove 18343
OnMouseUp 18346
OnStartDock 18348
OnStartDrag 18348
Options 18304
OutlineStyle 18305
ParentColor 18306
ParentCtl3D 18306
ParentFont 18307
ParentShowHint 18307
PictureClosed 18308
PictureLeaf 18308
PictureMinus 18308
PictureOpen 18309
PicturePlus 18309
PopupMenu 18310
ScrollBars 18314
ShowHint 18315
Style 18316

TabOrder 18317
TabStop 18317
TOutline events 18319
TOutline Members 18280
TOutline properties 18293
Visible 18318
TOutlineNode class 18349
about TOutlineNode class 18349
AddNode 18353
ChangeLevelBy 18353
Collapse 18354
Create 18351
Data 18360
Destroy 18352
Expand 18354
ExpandCount 18361
Expanded 18361
FullExpand 18355
FullPath 18361
GetDataItem 18355
GetDisplayWidth 18355
getFirstChild 18356
GetLastChild 18356
GetNextChild 18356
GetNodeAtIndex 18357
GetPrevChild 18357
GetRowOfNode 18357
GetTextItem 18358
GetVisibleNode 18358
HasAsParent 18358
HasItems 18362
Index 18362
InsertNode 18358
InternalRemove 18359
IsVisible 18363
Items 18363
Level 18363
List 18364
MoveTo 18359
Outline 18364
Parent 18364
Remove 18359
Text 18364
TopItem 18365
TOutlineNode Members 18350
TOutlineNode methods 18352
TOutlineNode properties 18360
WriteNode 18359
TOwnedCollection class 2912
about TOwnedCollection class 2912
Create 2914
GetOwner 2915
TOwnedCollection Members 2912
TOwnedCollection methods 2914
TPacketDataSet class 18564
about TPacketDataSet class 18564
AssignCurValues 18583
AssignCurValues method 18582
Create 18580
CreateFromDelta 18583
DataEvent 18583
FCurRecBuf 18581
FCurValues 18581
FNewValuesModified 18581
FOldRecBuf 18581
FUseCurValues 18581
FWritingCurValues 18582
GetStateFieldValue 18584
GetStreamMetaData 18584
HasCurValues 18584
HasMergeConflicts 18585
InitAltRecBuffers 18585
InternalClose 18585
InternalInitRecord 18586
InternalOpen 18586
NewValuesModified 18588
SetFieldData 18586
SetStateFieldValue 18586

SetStreamMetaData 18587
SetWritingCurValues 18587
StreamMetaData 18588
TPacketDataSet fields 18580
TPacketDataSet Members 18565
TPacketDataSet methods 18582
TPacketDataSet properties 18587
UpdateKind 18587
UseCurValues 18588
WritingCurValues 18589
TPacketInterceptFactory class 18925
about TPacketInterceptFactory class 18925
TPacketInterceptFactory Members 18925
TPacketInterceptFactory methods 18927
UpdateRegistry 18927
TPadding class 6103
about TPadding class 6103
Bottom 6106
InitDefaults 6105
Left 6106
Right 6106
Top 6107
TPadding Members 6104
TPadding methods 6105
TPadding properties 6106
TPage class 12961
about TPage class 12961
Caption 12973
Create 12971
Height 12976
OnAlignInsertBefore 12980
OnAlignPosition 12980
Paint 12972
ReadState 12973
TabOrder 12977
TPage events 12980
TPage Members 12961
TPage methods 12972
TPage properties 12973
Visible 12978
Width 12979
TPageControl class 4459
about TPageControl class 4459
ActivePage 4482
ActivePageIndex 4484
Align 4485
Anchors 4488
BiDiMode 4489
CanShowTab 4473
Change 4473
Constraints 4489
Create 4470
Destroy 4472
DoAddDockClient 4474
DockOver 4474
DockSite 4490
DoRemoveDockClient 4474
DragCursor 4491
DragKind 4491
DragMode 4491
Enabled 4493
FindNextPage 4475
Font 4493
GetChildren 4476
GetImageIndex 4477
GetPageFromDockClient 4477
GetSiteInfo 4477
HotTrack 4495
Images 4495
Loaded 4478
MultiLine 4495
OnChange 4516
OnChanging 4516
OnContextPopup 4517
OnDockDrop 4518
OnDockOver 4520
OnDragDrop 4522
OnDragOver 4524

wwww

OnDrawTab 4527
OnEndDock 4527
OnEndDrag 4527
OnEnter 4528
OnExit 4531
OnGetImageIndex 4532
OnGetSiteInfo 4533
OnMouseActivate 4535
OnMouseDown 4536
OnMouseEnter 4538
OnMouseLeave 4538
OnMouseMove 4539
OnMouseUp 4541
OnResize 4543
OnStartDock 4544
OnStartDrag 4544
OnUnDock 4545
OwnerDraw 4497
PageCount 4497
Pages 4498
ParentBiDiMode 4500
ParentFont 4500
ParentShowHint 4501
PopupMenu 4501
RaggedRight 4506
ScrollOpposite 4506
SelectNextPage 4478
SetActivePage 4480
SetChildOrder 4480
SetTabIndex 4480
ShowControl 4481
ShowHint 4506
Style 4508
TabHeight 4508
TabIndex 4509
TabOrder 4511
TabPosition 4512
TabStop 4512
TabWidth 4512
TPageControl events 4515
TPageControl Members 4459
TPageControl methods 4472
TPageControl properties 4481
UpdateActivePage 4481
Visible 4514
TPagedAdapter class 24525
about TPagedAdapter class 24525
Actions 24533
Data 24534
ModifyAccess 24534
OnAfterExecuteAction 24536
OnAfterGetActionResponse 24536
OnAfterUpdateFields 24536
OnAfterValidateFields 24537
OnBeforeExecuteAction 24537
OnBeforeGetActionResponse 24537
OnBeforeUpdateFields 24538
OnBeforeValidateFields 24538
OnGetActionParams 24539
OnGetEOF 24539
OnGetFirstRecord 24539
OnGetNextRecord 24539
OnGetRecordCount 24540
OnGetRecordIndex 24540
PageSize 24534
TPagedAdapter events 24535
TPagedAdapter Members 24527
TPagedAdapter properties 24533
ViewAccess 24535
TPageDispatcher class 24949
about TPageDispatcher class 24949
DefaultPage 24953
OnAfterDispatchPage 24954
OnBeforeDispatchPage 24954
OnCanViewPage 24955
OnIsLoginRequired 24956
OnPageAccessDenied 24956
OnPageNotFound 24957

TPageDispatcher events 24953
TPageDispatcher Members 24949
TPageDispatcher properties 24953
TPageItemsProducer class 18383
about TPageItemsProducer class 18383
Create 18388
Destroy 18388
GetTemplateStream 18388
HTMLDoc 18389
HTMLFile 18389
TPageItemsProducer Members 18384
TPageItemsProducer methods 18388
TPageItemsProducer properties 18389
TPageProducer class 15309
about TPageProducer class 15309
HTMLDoc 15314
HTMLFile 15314
OnHTMLTag 15316
ScriptEngine 15315
StripParamQuotes 15315
TPageProducer events 15316
TPageProducer Members 15310
TPageProducer properties 15313
TPageScroller class 4548
about TPageScroller class 4548
Align 4562
AlignControls 4559
Anchors 4565
AutoScroll 4566
BorderWidth 4566
ButtonSize 4567
Color 4567
Constraints 4568
Control 4568
Create 4558
CreateParams 4559
CreateWnd 4560
DockSite 4568
DragCursor 4569
DragKind 4570
DragMode 4570
DragScroll 4572
Enabled 4572
Font 4572
GetButtonState 4560
Margin 4574
Notification 4560
OnClick 4585
OnContextPopup 4587
OnDblClick 4588
OnDragDrop 4588
OnDragOver 4590
OnEndDock 4593
OnEndDrag 4593
OnEnter 4594
OnExit 4597
OnKeyDown 4598
OnKeyPress 4601
OnKeyUp 4602
OnMouseWheel 4604
OnResize 4604
OnScroll 4605
OnStartDock 4605
OnStartDrag 4605
Orientation 4574
ParentBackground 4574
ParentColor 4574
ParentFont 4575
ParentShowHint 4575
PopupMenu 4576
Position 4580
Scroll 4561
ShowHint 4581
TabOrder 4582
TabStop 4583
TPageScroller events 4585
TPageScroller Members 4548
TPageScroller methods 4559

TPageScroller properties 4561
Visible 4583
TPageSetupDialog class 11441
about TPageSetupDialog class 11441
BeforePaint 11455
Create 11446
Execute 11446
GetDefaults 11446
MarginBottom 11447
MarginLeft 11448
MarginRight 11448
MarginTop 11449
MinMarginBottom 11450
MinMarginLeft 11450
MinMarginRight 11451
MinMarginTop 11452
OnDrawEnvStamp 11455
OnDrawFullPage 11456
OnDrawGreekText 11456
OnDrawMargin 11457
OnDrawMinMargin 11457
OnDrawRetAddress 11458
Options 11452
PageHeight 11453
PageSetupDlgRec 11454
PageWidth 11454
TPageSetupDialog events 11455
TPageSetupDialog Members 11442
TPageSetupDialog methods 11446
TPageSetupDialog properties 11447
Units 11454
TPaintBox class 12981
about TPaintBox class 12981
Align 12990
Anchors 12994
Canvas 12994
Color 12996
Constraints 12997
Create 12989
DragCursor 12998
DragKind 12998
DragMode 12998
Enabled 13000
Font 13000
OnClick 13011
OnContextPopup 13013
OnDblClick 13014
OnDragDrop 13014
OnDragOver 13016
OnEndDock 13019
OnEndDrag 13019
OnMouseActivate 13020
OnMouseDown 13021
OnMouseEnter 13023
OnMouseLeave 13024
OnMouseMove 13024
OnMouseUp 13026
OnPaint 13028
OnStartDock 13030
OnStartDrag 13031
Paint 12989
ParentColor 13001
ParentFont 13002
ParentShowHint 13002
PopupMenu 13003
ShowHint 13008
TPaintBox events 13011
TPaintBox Members 12982
TPaintBox methods 12989
TPaintBox properties 12990
Visible 13009
TPaintControl class 9518
about TPaintControl class 9518
Create 9518
Ctl3DButton 9520
Destroy 9519
DestroyHandle 9519
Handle 9520

TPaintControl Members 9518
TPaintControl methods 9519
TPaintControl properties 9520
TPanel class 13032
about TPanel class 13032
Align 13044
Alignment 13048
Anchors 13048
AutoSize 13049
BevelEdges 13049
BevelInner 13050
BevelKind 13051
BevelOuter 13051
BevelWidth 13052
BiDiMode 13053
BorderStyle 13054
BorderWidth 13054
Caption 13055
Color 13058
Constraints 13059
Ctl3D 13059
DockManager 13060
DockSite 13062
DragCursor 13063
DragKind 13063
DragMode 13063
Enabled 13065
Font 13066
FullRepaint 13067
Locked 13067
OnAlignInsertBefore 13080
OnAlignPosition 13081
OnCanResize 13082
OnClick 13082
OnConstrainedResize 13084
OnContextPopup 13084
OnDblClick 13085
OnDockDrop 13085
OnDockOver 13087
OnDragDrop 13090
OnDragOver 13092
OnEndDock 13094
OnEndDrag 13095
OnEnter 13096
OnExit 13098
OnGetSiteInfo 13100
OnMouseActivate 13102
OnMouseDown 13103
OnMouseEnter 13105
OnMouseLeave 13105
OnMouseMove 13105
OnMouseUp 13108
OnResize 13110
OnStartDock 13111
OnStartDrag 13111
OnUnDock 13112
Padding 13067
ParentBackground 13068
ParentBiDiMode 13068
ParentColor 13068
ParentCtl3D 13069
ParentFont 13069
ParentShowHint 13070
PopupMenu 13070
ShowHint 13075
TabOrder 13076
TabStop 13077
TPanel events 13080
TPanel Members 13033
TPanel properties 13043
UseDockManager 13077
VerticalAlignment 13078
Visible 13078
TPanningWindow class 15401
about TPanningWindow class 15401
Create 15412
CreateParams 15413
Destroy 15412

aaaa

DoUpdate 15414
GetIsPanning 15414
GetPanOptions 15414
IsPanning 15421
MouseMove 15414
OnUpdate 15422
Paint 15415
PanControl 15421
PanInterval 15421
PanOptions 15422
SetPanCursor 15415
StartPanning 15416
StopPanning 15416
TPanningWindow events 15422
TPanningWindow Members 15401
TPanningWindow methods 15413
TPanningWindow properties 15421
UpdateScroll 15416
WndProc 15416
TParaAttributes class 4606
about TParaAttributes class 4606
Alignment 4609
Assign 4608
Create 4608
FirstIndent 4610
LeftIndent 4612
Numbering 4612
RightIndent 4614
Tab 4614
TabCount 4615
TParaAttributes Members 4607
TParaAttributes methods 4608
TParaAttributes properties 4609
TParam class 7386, 25735
about TParam class 7386, 25735
AsBCD 7407
AsBlob 7408
AsBoolean 7409
AsCurrency 7410
AsDate 7412
AsDateTime 7413
AsFloat 7415
AsFMTBCD 7416
AsInteger 7417
AsMemo 7419
Assign 7391
AssignField 7392
AssignFieldValue 7393
AssignParam 7393
AssignTo 7394
AsSmallInt 7419
AsSQLTimeStamp 7421
AsString 7421
AsTime 7423
AsWideString 7424
AsWord 7425
Bound 7426
Clear 7394
Create 7390
Create constructor 7389
DataSet 7427
DataType 7427
GetAsBCD 7395
GetAsBoolean 7395
GetAsCurrency 7395
GetAsDateTime 7395
GetAsFloat 7395
GetAsFMTBCD 7396
GetAsInteger 7396
GetAsMemo 7396
GetAsSQLTimeStamp 7396
GetAsString 7397
GetAsVariant 7397
GetAsWideString 7397
GetData 7397
GetDataSize 7398
GetDisplayName 7399
GetIsNull 7400

GetMessage 25740
GetName 25741
IsEqual 7400
IsNull 7429
LoadFromFile 7400
LoadFromStream 7400
Name 7430
NativeStr 7431
NumericScale 7431
ParamType 7431
Precision 7432
SetAsBCD 7401
SetAsBlob 7401
SetAsBoolean 7401
SetAsCurrency 7401
SetAsDate 7402
SetAsDateTime 7402
SetAsFloat 7402
SetAsFMTBCD 7402
SetAsInteger 7403
SetAsMemo 7403
SetAsSmallInt 7403
SetAsSQLTimeStamp 7403
SetAsString 7403
SetAsTime 7404
SetAsVariant 7404
SetAsWideString 7404
SetAsWord 7404
SetBlobData 7404
SetData 7405
SetDataType 7405
SetMessage 25741
SetName 25741
SetText 7406
Size 7432
Text 7433
TParam Members 7387, 25735
TParam methods 7390, 25740
TParam properties 7406
Value 7433
TParameter class 1250
about TParameter class 1250
AppendChunk 1253
Assign 1253
AssignTo 1254
Attributes 1257
DataType 1257
Direction 1257
GetDisplayName 1255
IsEqual 1255
LoadFromFile 1255
LoadFromStream 1255
Name 1258
NumericScale 1258
ParameterObject 1259
Parameters 1259
Precision 1259
Properties 1260
Size 1260
TParameter Members 1251
TParameter methods 1252
TParameter properties 1256
Value 1260
TParameters class 1261
about TParameters class 1261
AddParameter 1264
AppendParameters 1264
AssignValues 1265
Command 1272
Create_Parameter 1265
CreateParameter 1266
FindParam 1266
GetAttr 1267
GetAttrCount 1267
GetItemAttr 1268
GetParamList 1268
InternalRefresh 1269
IsEqual 1269

Items 1272
Modified 1272
ParamByName 1269
ParameterCollection 1273
ParamValues 1273
ParseSQL 1270
Refresh 1271
TParameters Members 1261
TParameters methods 1263
TParameters properties 1271
Update 1271
TParamList class 10056
about TParamList class 10056
Buffer 10058
Create 10057
Destroy 10057
FieldCount 10058
FieldDescs 10059
TParamList Members 10056
TParamList properties 10058
TParams class 7434
about TParams class 7434
AddParam 7438
AssignTo 7438
AssignValues 7439
Create 7437
Create constructor 7437
CreateParam 7441
DefineProperties 7444
FindParam 7445
GetDataSet 7445
GetOwner 7445
GetParamList 7446
IsEqual 7446
Items 7449
ParamByName 7447
ParamValues 7451
ParseSQL 7447
RemoveParam 7448
TParams Members 7435
TParams methods 7437
TParams properties 7449
Update 7448
TParser class 2915
about TParser class 2915
CheckToken 2919
CheckTokenSymbol 2919
Create 2917
Destroy 2917
Error 2920
ErrorFmt 2920
ErrorStr 2921
FloatType 2925
GetLinePos 2921
HexToBinary 2921
LinePos 2926
NextToken 2922
OnError 2927
SourceLine 2926
SourcePos 2922
Token 2926
TokenComponentIdent 2923
TokenFloat 2923
TokenInt 2923
TokenString 2924
TokenSymbolIs 2924
TokenWideString 2925
TParser events 2927
TParser Members 2916
TParser methods 2918
TParser properties 2925
TPart class 25741
about TPart class 25741
GetElement 25747
GetName 25747
GetType_ 25747
SetElement 25747
SetName 25748

SetType_ 25748
TPart Members 25742
TPart methods 25746
TParts class 25748
about TParts class 25748
Add 25754
GetPart 25754
Insert 25754
TParts Members 25748
TParts methods 25754
TPasswordDialog class 9828
about TPasswordDialog class 9828
AddButton 9845
AddButtonClick 9847
CancelButton 9845
Edit 9845
EditChange 9847
GroupBox1 9846
OKButton 9846
OKButtonClick 9848
RemoveAllButton 9846
RemoveAllButtonClick 9848
RemoveButton 9846
RemoveButtonClick 9848
TPasswordDialog fields 9845
TPasswordDialog Members 9828
TPasswordDialog methods 9847
tpBottom enumeration member 5467
TPen class 14571
about TPen class 14571
Assign 14574
Color 14577
Create 14573
Destroy 14573
GetColor 14575
GetHandle 14575
GetStyle 14575
GetWidth 14575
Handle 14578
Mode 14579
SetColor 14575
SetHandle 14576
SetMode 14576
SetStyle 14576
SetWidth 14576
Style 14579
TPen Members 14572
TPen methods 14574
TPen properties 14577
Width 14580
TPenRecall class 14582
about TPenRecall class 14582
Create 14583
TPenRecall Members 14582
TPersistent class 2928
about TPersistent class 2928
Assign 2930
AssignTo 2931
DefineProperties 2931
Destroy 2929
GetNamePath 2932
GetOwner 2932
TPersistent Members 2928
TPersistent methods 2930
tpHigher enumeration member 3278
tpHighest enumeration member 3278
TPicture class 14584
about TPicture class 14584
Assign 14587
AssignTo 14589
Bitmap 14598
Changed 14590
Create 14586
DefineProperties 14590
Destroy 14586
Graphic 14600
Height 14600
Icon 14601

LoadFromClipboardFormat 14590
LoadFromFile 14591
LoadFromStream 14591
Metafile 14601
OnChange 14602
OnProgress 14603
PictureAdapter 14601
Progress 14592
RegisterClipboardFormat 14592
RegisterFileFormat 14594
RegisterFileFormatRes 14594
SaveToClipboardFormat 14595
SaveToFile 14595
SaveToStream 14597
SupportsClipboardFormat 14597
TPicture events 14602
TPicture Members 14584
TPicture methods 14587
TPicture properties 14598
UnregisterGraphicClass 14597
Width 14602
TPictureAdapter class 1599
about TPictureAdapter class 1599
Create 1601
GetOlePicture 1602
SetOlePicture 1602
TPictureAdapter Members 1600
TPictureAdapter methods 1602
Update 1602
tpIdle enumeration member 3278
TPivotDataLink class 17737
about TPivotDataLink class 17737
Create 17739
DecisionDataEvent 17739
Destroy 17739
TPivotDataLink Members 17738
TPivotDataLink methods 17739
TPivotState class 17504
about TPivotState class 17504
Assign 17508
Create 17505
Destroy 17505
DimInfo 17506
FColSparse 17506
FColSubs 17506
FCurrentSum 17507
FDims 17507
FRowSparse 17507
FRowSubs 17507
FSums 17507
IsEqual 17508
TPivotState fields 17506
TPivotState Members 17505
TPivotState methods 17508
tpLeft enumeration member 5467
tpLower enumeration member 3278
tpLowest enumeration member 3278
tpNormal enumeration member 3278
TPointerArray class 17351
about TPointerArray class 17351
AsPtr 17355
CloneItem 17354
CopyFrom 17354
CopyTo 17354
Create 17353
Data 17356
FreeItem 17354
GetData 17355
InvalidateItems 17355
PutData 17355
TPointerArray Members 17351
TPointerArray methods 17353
TPointerArray properties 17355
TPopupActionBar class 852
about TPopupActionBar class 852
Create 856
MenuActive 857
OnGetControlClass 858

Popup 856
PopupMenu 858
TPopupActionBar events 858
TPopupActionBar Members 852
TPopupActionBar methods 856
TPopupActionBar properties 857
TPopupDataList class 9521
about TPopupDataList class 9521
Create 9534
CreateParams 9534
TPopupDataList Members 9521
TPopupDataList methods 9534
TPopupList class 16164
about TPopupList class 16164
Add 16167
FWindow 16166
MainWndProc 16167
Remove 16167
TPopupList fields 16166
TPopupList Members 16165
TPopupList methods 16166
TPopupList properties 16168
Window 16168
WndProc 16168
TPopupMenu class 16169
about TPopupMenu class 16169
Alignment 16176
AutoHotkeys 16176
AutoLineReduction 16177
AutoPopup 16178
BiDiMode 16178
Create 16173
Destroy 16173
DoPopup 16174
HelpContext 16179
Images 16179
MenuAnimation 16180
OnChange 16184
OnPopup 16184
OwnerDraw 16180
ParentBiDiMode 16180
Popup 16174
PopupComponent 16181
PopupPoint 16183
SetPopupPoint 16175
TPopupMenu events 16184
TPopupMenu Members 16169
TPopupMenu methods 16174
TPopupMenu properties 16175
TrackButton 16183
UseRightToLeftAlignment 16175
TPort class 25755
about TPort class 25755
GetBinding 25760
GetName 25760
SetBinding 25760
SetName 25761
TPort Members 25755
TPort methods 25760
TPorts class 25761
about TPorts class 25761
Add 25767
GetPort 25767
Insert 25767
TPorts Members 25761
TPorts methods 25766
TPortType class 25767
about TPortType class 25767
AfterConstruction 25773
GetName 25773
GetOperations 25773
SetName 25773
TPortType Members 25768
TPortType methods 25772
TPortTypes class 25774
about TPortTypes class 25774
Add 25779
GetPortType 25780

Insert 25780
TPortTypes Members 25774
TPortTypes methods 25779
TPostButton class 16704
about TPostButton class 16704
Create 16708
TPostButton Members 16705
TPreviousTab class 11977
about TPreviousTab class 11977
AfterTabChange 11989
BeforeTabChange 11989
Caption 11983
Enabled 11984
ExecuteTarget 11982
HelpContext 11984
HelpKeyword 11985
HelpType 11985
Hint 11986
ImageIndex 11986
OnHint 11989
OnUpdate 11990
OnValidateTab 11991
ShortCut 11986
SkipHiddenTabs 11987
TabControl 11987
TPreviousTab events 11988
TPreviousTab Members 11977
TPreviousTab methods 11982
TPreviousTab properties 11983
UpdateTarget 11982
Visible 11987
Wrap 11988
tpRight enumeration member 5467
TPrintDialog class 11458
about TPrintDialog class 11458
Collate 11464
Copies 11464
Execute 11462
FromPage 11465
MaxPage 11467
MinPage 11469
Options 11471
PrintRange 11473
PrintToFile 11475
ToPage 11476
TPrintDialog Members 11458
TPrintDialog methods 11462
TPrintDialog properties 11464
TPrintDlg class 20713
about TPrintDlg class 20713
BeforeExecute 20723
Caption 20718
Dialog 20719
Enabled 20719
GetDialogClass 20718
HelpContext 20720
HelpKeyword 20720
HelpType 20721
Hint 20721
ImageIndex 20721
OnAccept 20724
OnCancel 20724
OnHint 20724
OnUpdate 20724
SecondaryShortCuts 20722
ShortCut 20722
TPrintDlg events 20723
TPrintDlg Members 20714
TPrintDlg methods 20718
TPrintDlg properties 20718
Visible 20723
TPrinter class 18392
Abort 18395
Aborted 18403
about TPrinter class 18392
BeginDoc 18397
Canvas 18404
Capabilities 18404

hhhh

Copies 18405
Create 18394
Destroy 18395
EndDoc 18399
Fonts 18405
GetPrinter 18400
Handle 18405
NewPage 18400
Orientation 18406
PageHeight 18407
PageNumber 18408
PageWidth 18408
PrinterIndex 18409
Printers 18409
Printing 18409
Refresh 18401
SetPrinter 18402
Title 18411
TPrinter Members 18393
TPrinter methods 18395
TPrinter properties 18402
TPrinterSetupDialog class 11478
about TPrinterSetupDialog class 11478
Execute 11482
TPrinterSetupDialog Members 11479
TPrinterSetupDialog methods 11482
TPriorButton class 16709
about TPriorButton class 16709
Create 16713
TPriorButton Members 16709
TPriorPageButton class 16713
about TPriorPageButton class 16713
Create 16718
TPriorPageButton Members 16714
TProgressBar class 4615
about TProgressBar class 4615
Align 4633
Anchors 4637
BorderWidth 4637
Constraints 4637
Create 4626
CreateParams 4627
CreateWnd 4628
DestroyWnd 4628
DragCursor 4638
DragKind 4638
DragMode 4639
Enabled 4640
Hint 4641
Max 4643
Min 4646
OnContextPopup 4665
OnDragDrop 4665
OnDragOver 4668
OnEndDock 4670
OnEndDrag 4670
OnEnter 4671
OnExit 4674
OnMouseActivate 4675
OnMouseDown 4676
OnMouseEnter 4679
OnMouseLeave 4679
OnMouseMove 4679
OnMouseUp 4681
OnStartDock 4684
OnStartDrag 4684
Orientation 4649
ParentShowHint 4649
PopupMenu 4650
Position 4655
ShowHint 4658
Smooth 4660
Step 4660
StepBy 4629
StepIt 4631
TabOrder 4662
TabStop 4662
TProgressBar events 4664

TProgressBar Members 4616
TProgressBar methods 4627
TProgressBar properties 4632
Visible 4663
TPropertyEditor class 11019
about TPropertyEditor class 11019
Activate 11024
AllEqual 11024
AutoFill 11025
Create 11022
Designer 11044
Destroy 11022
Edit 11025
GetAttributes 11026
GetComponent 11026
GetEditLimit 11027
GetEditValue 11027
GetEditValue method 11027
GetFloatValue 11028
GetFloatValueAt 11028
GetInt64Value 11028
GetInt64ValueAt 11029
GetIntfValue 11029
GetIntfValueAt 11030
GetIsDefault 11030
GetKind 11030
GetMethodValue 11031
GetMethodValueAt 11031
GetName 11032
GetOrdValue 11032
GetOrdValueAt 11032
GetProperties 11033
GetPropInfo 11033
GetPropType 11033
GetStrValue 11034
GetStrValueAt 11034
GetValue 11035
GetValues 11035
GetValueW 11036
GetVarValue 11036
GetVarValueAt 11036
GetVisualValue 11037
GetWideStrValue 11037
GetWideStrValueAt 11037
HasInstance 11038
Initialize 11038
Modified 11038
PrivateDirectory 11044
PropCount 11045
Revert 11039
SetFloatValue 11039
SetInt64Value 11040
SetIntfValue 11040
SetMethodValue 11040
SetOrdValue 11041
SetPropEntry 11041
SetStrValue 11042
SetValue 11042, 11043
SetValue method 11042
SetVarValue 11043
SetWideStrValue 11043
TPropertyEditor Members 11020
TPropertyEditor methods 11023
TPropertyEditor properties 11044
Value 11045
ValueAvailable 11044
TPropertyPage class 1603
about TPropertyPage class 1603
ActiveControl 1620
AutoScroll 1621
Caption 1621
ClientHeight 1624
ClientWidth 1624
Color 1626
Create 1616
Ctl3D 1627
Destroy 1617
Enabled 1628

jjjj

EnumCtlProps 1617
Font 1628
Height 1629
HorzScrollBar 1630
KeyPreview 1631
Modified 1618
OldCreateOrder 1634
OleObject 1635
OleObjects 1635
OnActivate 1646
OnClick 1650
OnClose 1652
OnContextPopup 1653
OnCreate 1654
OnDblClick 1655
OnDeactivate 1656
OnDestroy 1656
OnDragDrop 1657
OnDragOver 1659
OnHide 1662
OnKeyDown 1664
OnKeyPress 1667
OnKeyUp 1668
OnMouseDown 1669
OnMouseMove 1672
OnMouseUp 1674
OnPaint 1676
OnResize 1677
OnShow 1678
ParentFont 1636
PixelsPerInch 1636
PopupMenu 1636
PrintScale 1641
Scaled 1641
ShowHint 1642
TPropertyPage events 1646
TPropertyPage Members 1603
TPropertyPage methods 1617
TPropertyPage properties 1619
UpdateObject 1618
UpdatePropertyPage 1619
VertScrollBar 1643
Visible 1644
Width 1645
TPropertyPageImpl class 1679
about TPropertyPageImpl class 1679
Activate 1681
Apply 1681
Deactivate 1681
EditProperty 1682
GetPageInfo 1682
Help 1682
InitPropertyPage 1682
IsPageDirty 1683
Move 1683
PropertyPage 1684
SetObjects 1683
SetPageSite 1683
Show 1684
TPropertyPageImpl Members 1679
TPropertyPageImpl methods 1681
TPropertyPageImpl properties 1684
TranslateAccelerator 1684
TProvider class 18589
about TProvider class 18589
tpTimeCritical enumeration member 3278
tpTop enumeration member 5467
TPublishableVariantType class 23332
about TPublishableVariantType class 23332
GetInstance 23336
GetProperty 23336
SetProperty 23337
TPublishableVariantType Members 23333
TPublishableVariantType methods 23335
TQADecisionSource class 17508
about TQADecisionSource class 17508
tqeMapCorrupt enumeration member 17434
tqeNoAggs enumeration member 17434

tqeNoDimensions enumeration member 17434
tqeNotGrouped enumeration member 17434
tqeNotInitialized enumeration member 17434
tqeOK enumeration member 17434
tqeUnKnownDims enumeration member 17434
TQuery class 10059
about TQuery class 10059
Constrained 10083
Constraints 10083
Create 10073
CreateHandle 10074
DataLink 10084
DataSource 10084
DefineProperties 10074
Destroy 10073
Disconnect 10075
ExecSQL 10075
FreeStatement 10076
GetDataSource 10076
GetDetailLinkFields 10076
GetParamsCount 10076
GetStatementHandle 10077
Local 10085
ParamByName 10077
ParamCheck 10086
ParamCount 10086
Params 10087
Prepare 10077
Prepared 10087
PSExecute 10078
PSGetCommandText 10078
PSGetCommandType 10079
PSGetDefaultOrder 10079
PSGetParams 10079
PSGetTableName 10080
PSSetCommandText 10080
PSSetParams 10081
RequestLive 10088
RowsAffected 10089
SetDBFlag 10081
SQL 10089
SQLBinary 10090
StmtHandle 10091
Text 10091
TQuery Members 10060
TQuery methods 10073
TQuery properties 10082
UniDirectional 10092
UnPrepare 10081
UpdateMode 10092
UpdateObject 10092
TQueryButton class 16718
about TQueryButton class 16718
Caption 16727
Create 16723
Custom 16728
DefaultCaption 16723
EventContent 16724
GetCaption 16724
GetHTMLControlName 16725
GetHTMLForm 16725
GetInputType 16725
GetXMLComponent 16725
ImplContent 16725
InputType 16723
Notification 16726
ParentXMLComponent 16726
SetXMLComponent 16727
SetXMLUseParent 16727
Style 16728
StyleRule 16729
TQueryButton fields 16723
TQueryButton Members 16719
TQueryButton methods 16724
TQueryButton properties 16727
XMLComponent 16729
XMLUseParent 16729
TQueryButtons class 16730

about TQueryButtons class 16730
Custom 16735
Style 16735
StyleRule 16736
TQueryButtons Members 16731
TQueryButtons properties 16735
TQueryDim class 17855
about TQueryDim class 17855
Active 17857
TQueryDim Members 17855
TQueryDim properties 17857
TQueryDims class 17857
about TQueryDims class 17857
Add 17860
Create 17859
GetQueryDim 17860
Items 17861
SetQueryDim 17860
TQueryDims Members 17857
TQueryDims methods 17860
TQueryDims properties 17861
TQueryFieldGroup class 16736
about TQueryFieldGroup class 16736
Custom 16744
FindParam 16743
GetParamsList 16743
ImplCanAddClass 16743
IsParamInUse 16744
Style 16745
StyleRule 16745
TQueryFieldGroup Members 16738
TQueryFieldGroup methods 16743
TQueryFieldGroup properties 16744
TQueryForm class 16745
about TQueryForm class 16745
Action 16751
Custom 16751
Method 16752
Style 16752
StyleRule 16752
TQueryForm Members 16746
TQueryForm properties 16751
TQueryRadioGroup class 16753
about TQueryRadioGroup class 16753
AddAttributes 16759
Caption 16761
CaptionAttributes 16761
CaptionPosition 16761
Custom 16762
DataSet 16762
DisplayColumns 16763
DisplayWidth 16763
FieldName 16764
GetCheckIndex 16759
GetText 16759
IsQueryField 16760
Items 16764
ItemsField 16765
ParamName 16765
ReadOnly 16766
SetText 16760
Style 16766
StyleRule 16766
TabIndex 16767
Text 16767
TQueryRadioGroup Members 16754
TQueryRadioGroup methods 16759
TQueryRadioGroup properties 16760
Values 16767
ValuesField 16768
TQuerySelectOptions class 16768
about TQuerySelectOptions class 16768
Caption 16776
CaptionAttributes 16777
CaptionPosition 16777
Custom 16777
DataSet 16778
DisplayRows 16778

FieldName 16779
GetSelectIndex 16775
GetText 16775
IsQueryField 16775
Items 16779
ItemsField 16780
ParamName 16780
SetText 16776
Style 16781
StyleRule 16781
TabIndex 16781
Text 16782
TQuerySelectOptions Members 16770
TQuerySelectOptions methods 16774
TQuerySelectOptions properties 16776
Values 16782
ValuesField 16782
TQuerySprig class 1744
about TQuerySprig class 1744
AnyProblems 1745
TQuerySprig Members 1744
TQuerySprig methods 1745
TQueryTableProducer class 8152
about TQueryTableProducer class 8152
Caption 8159
CaptionAlignment 8159
Columns 8160
Content 8157
Footer 8160
GetDataSet 8158
Header 8161
MaxRows 8161
Notification 8158
OnCreateContent 8164
OnFormatCell 8164
OnGetTableCaption 8165
Query 8162
RowAttributes 8162
SetDataSet 8158
TableAttributes 8163
TQueryTableProducer events 8164
TQueryTableProducer Members 8153
TQueryTableProducer methods 8157
TQueryTableProducer properties 8159
TQueryText class 16783
about TQueryText class 16783
AddAttributes 16789
Caption 16790
CaptionAttributes 16791
CaptionPosition 16791
Create 16788
Custom 16791
DisplayWidth 16792
FieldName 16792
GetText 16789
IsQueryField 16789
MaxWidth 16793
ParamName 16793
ReadOnly 16793
SetText 16790
Style 16794
StyleRule 16794
TabIndex 16794
Text 16795
TQueryText Members 16784
TQueryText methods 16788
TQueryText properties 16790
TQueryTextArea class 16795
about TQueryTextArea class 16795
AddAttributes 16801
Caption 16803
CaptionAttributes 16804
CaptionPosition 16804
ControlContent 16801
Create 16800
Custom 16804
Destroy 16801
DisplayRows 16805

DisplayWidth 16805
FieldName 16805
GetText 16802
IsQueryField 16802
Lines 16806
ParamName 16806
ReadOnly 16807
SetLines 16802
SetText 16803
Style 16807
StyleRule 16808
TabIndex 16808
TQueryTextArea Members 16796
TQueryTextArea methods 16801
TQueryTextArea properties 16803
Wrap 16808
TQueue class 5699
about TQueue class 5699
PushItem 5701
TQueue Members 5700
TQueue methods 5701
TRadioButton class 21753
about TRadioButton class 21753
Action 21767
Align 21768
Alignment 21771
Anchors 21772
BiDiMode 21772
Caption 21773
Checked 21775
Color 21776
Constraints 21776
Create 21764
CreateParams 21765
CreateWnd 21765
Ctl3D 21777
DragCursor 21777
DragKind 21778
DragMode 21778
Enabled 21780
Font 21780
GetChecked 21766
GetControlsAlignment 21766
OnClick 21793
OnContextPopup 21795
OnDblClick 21796
OnDragDrop 21796
OnDragOver 21798
OnEndDock 21801
OnEndDrag 21801
OnEnter 21802
OnExit 21805
OnKeyDown 21806
OnKeyPress 21809
OnKeyUp 21810
OnMouseActivate 21812
OnMouseDown 21813
OnMouseEnter 21815
OnMouseLeave 21815
OnMouseMove 21815
OnMouseUp 21818
OnStartDock 21820
OnStartDrag 21821
ParentBiDiMode 21781
ParentColor 21782
ParentCtl3D 21782
ParentFont 21783
ParentShowHint 21783
PopupMenu 21784
SetChecked 21766
ShowHint 21788
TabOrder 21790
TabStop 21790
TRadioButton events 21793
TRadioButton Members 21754
TRadioButton methods 21764
TRadioButton properties 21767
Visible 21791

WordWrap 21792
TRadioGroup class 13115
about TRadioGroup class 13115
Align 13127
Anchors 13130
BiDiMode 13131
Caption 13131
Color 13134
Columns 13135
Constraints 13135
Ctl3D 13135
DragCursor 13136
DragKind 13136
DragMode 13137
Enabled 13138
Font 13139
ItemIndex 13140
Items 13141
OnClick 13153
OnContextPopup 13155
OnDragDrop 13156
OnDragOver 13158
OnEndDock 13161
OnEndDrag 13161
OnEnter 13162
OnExit 13164
OnStartDock 13166
OnStartDrag 13167
ParentBackground 13142
ParentBiDiMode 13142
ParentColor 13143
ParentCtl3D 13143
ParentFont 13143
ParentShowHint 13144
PopupMenu 13144
ShowHint 13149
TabOrder 13150
TabStop 13151
TRadioGroup events 13153
TRadioGroup Members 13115
TRadioGroup properties 13126
Visible 13151
TranslateException class 26466
about TranslateException class 26466
TRawSocket class 20094
about TRawSocket class 20094
Create 20098
SockType 20098
TRawSocket Members 20094
TRawSocket properties 20098
TRDSConnection class 1274
about TRDSConnection class 1274
AfterConnect 1284
AfterDisconnect 1284
AppServer 1281
BeforeConnect 1284
BeforeDisconnect 1285
ComputerName 1281
Connected 1282
Create 1278
DataSpaceObject 1282
Destroy 1278
DoConnect 1279
DoDisconnect 1279
GetConnected 1280
GetRecordset 1280
InternetTimeout 1283
ServerName 1283
TRDSConnection events 1283
TRDSConnection Members 1275
TRDSConnection methods 1279
TRDSConnection properties 1281
TReader class 2933
about TReader class 2933
BeginReferences 2938
CanHandleExceptions 2958
CheckValue 2939
CopyValue 2939

DefineBinaryProperty 2939
DefineProperty 2940
Destroy 2936
EndOfList 2941
EndReferences 2941
Error 2942
FindAncestorComponent 2942
FindMethod 2942
FindMethodInstance 2942
FixupReferences 2942
FlushBuffer 2943
NextValue 2943
OnAncestorNotFound 2961
OnCreateComponent 2961
OnError 2961
OnFindComponentClass 2962
OnFindComponentInstance 2962
OnFindMethod 2962
OnFindMethodInstance 2963
OnReferenceName 2963
OnSetName 2963
Owner 2959
Parent 2959
Position 2960
PropertyError 2944
PropName 2960
Read 2944
ReadBoolean 2944
ReadChar 2945
ReadCollection 2945
ReadComponent 2945
ReadComponents 2946
ReadCurrency 2946
ReadData 2947
ReadDate 2947
ReadDouble 2947
ReadFloat 2948
ReadIdent 2948
ReadInt64 2949
ReadInteger 2949
ReadListBegin 2950
ReadListEnd 2950
ReadPrefix 2951
ReadProperty 2951
ReadPropValue 2952
ReadRootComponent 2952
ReadSet 2952
ReadSignature 2953
ReadSingle 2953
ReadStr 2953
ReadString 2954
ReadValue 2954
ReadVariant 2955
ReadWideChar 2955
ReadWideString 2956
ReferenceName 2956
SetName 2956
SetPosition 2957
SkipBytes 2957
SkipComponent 2957
SkipProperty 2957
SkipSetBody 2958
SkipValue 2958
TReader events 2960
TReader Members 2934
TReader methods 2937
TReader properties 2958
TRecall class 2964
about TRecall class 2964
Create 2965
Destroy 2966
Forget 2967
Reference 2968
Store 2967
TRecall Members 2964
TRecall methods 2967
TRecall properties 2968
TReconcileActionLink class 8570

qqqq

about TReconcileActionLink class 8570
ApplyAction 8573
DataEvent 8573
TReconcileActionLink Members 8571
TReconcileActionLink methods 8573
TReconcileActionLink properties 8573
TReferenceField class 7452
about TReferenceField class 7452
Assign 7459
Create 7458
GetAsVariant 7459
GetDataSize 7460
ReferenceTableName 7460
SetVarValue 7460
Size 7460
TReferenceField Members 7452
TReferenceField methods 7459
TReferenceField properties 7460
TReflectorWindow class 1684
about TReflectorWindow class 1684
Create 1695
CreateParams 1695
TReflectorWindow Members 1685
TReflectorWindow methods 1695
TRegIniFile class 18635
about TRegIniFile class 18635
Create 18638
Create constructor 18638
DeleteKey 18639
EraseSection 18639
FileName 18644
ReadBool 18640
ReadInteger 18640
ReadSection 18641
ReadSections 18641
ReadSectionValues 18642
ReadString 18642
TRegIniFile Members 18635
TRegIniFile methods 18638
TRegIniFile properties 18644
WriteBool 18643
WriteInteger 18643
WriteString 18644
TRegistry class 18645
about TRegistry class 18645
Access 18672
ChangeKey 18650
CloseKey 18650
Create 18648
Create constructor 18647
CreateKey 18651
CurrentKey 18673
CurrentPath 18673
DeleteKey 18651
DeleteValue 18652
Destroy 18648
GetBaseKey 18653
GetData 18653
GetDataInfo 18653
GetDataSize 18654
GetDataType 18654
GetKey 18655
GetKeyInfo 18655
GetKeyNames 18655
GetValueNames 18656
HasSubKeys 18656
KeyExists 18656
LazyWrite 18674
LoadKey 18657
MoveKey 18657
OpenKey 18658
OpenKeyReadOnly 18658
PutData 18659
ReadBinaryData 18659
ReadBool 18660
ReadCurrency 18660
ReadDate 18661
ReadDateTime 18661

ReadFloat 18662
ReadInteger 18662
ReadString 18662
ReadTime 18663
RegistryConnect 18663
RenameValue 18664
ReplaceKey 18664
RestoreKey 18665
RootKey 18674
SaveKey 18666
SetCurrentKey 18666
TRegistry Members 18645
TRegistry methods 18649
TRegistry properties 18672
UnLoadKey 18666
ValueExists 18667
WriteBinaryData 18667
WriteBool 18668
WriteCurrency 18668
WriteDate 18669
WriteDateTime 18669
WriteExpandString 18670
WriteFloat 18670
WriteInteger 18671
WriteString 18671
WriteTime 18671
TRegistryClass class 17905
about TRegistryClass class 17905
Create 17905
Destroy 17906
TRegistryClass Members 17905
TRegistryClass methods 17906
UpdateRegistry 17906
TRegistryIniFile class 18675
about TRegistryIniFile class 18675
Create 18677
Create constructor 18677
DeleteKey 18679
Destroy 18678
EraseSection 18679
ReadBinaryStream 18679
ReadDate 18680
ReadDateTime 18681
ReadFloat 18681
ReadInteger 18682
ReadSection 18682
ReadSections 18683
ReadSections method 18683
ReadSectionValues 18683
ReadString 18684
ReadTime 18685
RegIniFile 18689
TRegistryIniFile Members 18676
TRegistryIniFile methods 18678
TRegistryIniFile properties 18689
UpdateFile 18685
WriteBinaryStream 18685
WriteDate 18686
WriteDateTime 18687
WriteFloat 18687
WriteInteger 18688
WriteString 18688
WriteTime 18689
TRemotable class 15556
about TRemotable class 15556
Create 15558
DataContext 15561
Destroy 15558
FSerializationOptions 15559
ObjectToSOAP 15559
SerializationOptions 15561
SOAPToObject 15560
TRemotable fields 15559
TRemotable Members 15557
TRemotable methods 15559
TRemotable properties 15560
TRemotableTypeRegistry class 15562
about TRemotableTypeRegistry class 15562

AutoRegisterNativeTypes 15579
ClassOptions 15565
ClassToURI 15566
ClassToURI method 15566
Create 15563
Destroy 15564
GetExternalPropName 15566
GetInternalPropName 15567
GetURICount 15567
GetURIMap 15568
GetURIMap method 15568
GetVarTypeFromXSD 15569
GetXSDInfoForClass 15569
InfoToURI 15569
IsClassScalar 15570
Lock 15570
RegisterExternalPropName 15571
RegisterHolderClsMember 15571
RegisterSerializeOptions 15572
RegisterSerializeOptions method 15572
RegisterXSClass 15573
RegisterXSInfo 15574
SerializeOptions 15575
SerializeOptions method 15574
TRemotableTypeRegistry Members 15562
TRemotableTypeRegistry methods 15564
TRemotableTypeRegistry properties 15579
TypeInfoToXSD 15575
UnLock 15576
UnRegisterXSClass 15576
UnRegisterXSInfo 15577
URIToClass 15577
URIToClass method 15577
URIToInfo 15578
VariantToInfo 15578
XSDToTypeInfo 15579
TRemotableXS class 15580
about TRemotableXS class 15580
NativeToXS 15581
ObjectToSOAP 15582
SOAPToObject 15582
TRemotableXS Members 15581
TRemotableXS methods 15581
XSToNative 15583
TRemoteDataModule class 6585
about TRemoteDataModule class 6585
AS_ApplyUpdates 6590
AS_DataRequest 6591
AS_Execute 6591
AS_GetParams 6592
AS_GetProviderNames 6593
AS_GetRecords 6593
AS_RowRequest 6594
Create 6589
Destroy 6589
GetProvider 6595
Lock 6595
ProviderCount 6599
Providers 6599
RegisterProvider 6596
TRemoteDataModule Members 6585
TRemoteDataModule methods 6589
TRemoteDataModule properties 6598
Unlock 6596
UnRegisterProvider 6597
UpdateRegistry 6597
TRemoteServer class 16470
about TRemoteServer class 16470
TReplaceDialog class 11482
about TReplaceDialog class 11482
Create 11486
OnReplace 11488
ReplaceText 11487
TReplaceDialog events 11488
TReplaceDialog Members 11483
TReplaceDialog properties 11487
TResetQueryButton class 16809
about TResetQueryButton class 16809

Create 16813
EventContent 16813
TResetQueryButton Members 16809
TResetQueryButton methods 16813
TResourceStream class 2968
about TResourceStream class 2968
Create 2970
CreateFromID 2971
Destroy 2972
TResourceStream Members 2969
TResourceStream methods 2972
Write 2973
trHorizontal enumeration member 5471
TRichEdit class 4685
about TRichEdit class 4685
Align 4699
Alignment 4703
Anchors 4703
BevelEdges 4704
BevelInner 4704
BevelKind 4705
BevelOuter 4705
BevelWidth 4705
BiDiMode 4706
BorderStyle 4706
BorderWidth 4707
Color 4707
Constraints 4708
Ctl3D 4708
DragCursor 4709
DragKind 4709
DragMode 4710
Enabled 4711
Font 4712
HideScrollBars 4713
HideSelection 4715
ImeMode 4716
ImeName 4716
Lines 4717
MaxLength 4719
OnChange 4738
OnClick 4738
OnContextPopup 4740
OnDblClick 4741
OnDragDrop 4741
OnDragOver 4743
OnEndDock 4746
OnEndDrag 4746
OnEnter 4747
OnExit 4749
OnKeyDown 4751
OnKeyPress 4754
OnKeyUp 4755
OnMouseActivate 4757
OnMouseDown 4758
OnMouseEnter 4760
OnMouseLeave 4760
OnMouseMove 4760
OnMouseUp 4763
OnMouseWheel 4765
OnMouseWheelDown 4765
OnMouseWheelUp 4766
OnProtectChange 4766
OnResizeRequest 4766
OnSaveClipboard 4767
OnSelectionChange 4767
OnStartDock 4768
OnStartDrag 4768
ParentBiDiMode 4719
ParentColor 4720
ParentCtl3D 4720
ParentFont 4721
ParentShowHint 4721
PlainText 4722
PopupMenu 4723
ReadOnly 4727
ScrollBars 4728
ShowHint 4730

TabOrder 4731
TabStop 4732
TRichEdit events 4737
TRichEdit Members 4686
TRichEdit properties 4698
Visible 4732
WantReturns 4734
WantTabs 4734
WordWrap 4735
TRichEditAction class 11992
about TRichEditAction class 11992
Create 11997
CurrText 11997
HandlesTarget 11998
SetFontStyle 11998
TRichEditAction Members 11992
TRichEditAction methods 11997
TRichEditAlignCenter class 11998
about TRichEditAlignCenter class 11998
ExecuteTarget 12004
TRichEditAlignCenter Members 11999
TRichEditAlignCenter methods 12004
UpdateTarget 12004
TRichEditAlignLeft class 12005
about TRichEditAlignLeft class 12005
ExecuteTarget 12010
TRichEditAlignLeft Members 12005
TRichEditAlignLeft methods 12010
UpdateTarget 12010
TRichEditAlignRight class 12011
about TRichEditAlignRight class 12011
ExecuteTarget 12016
TRichEditAlignRight Members 12012
TRichEditAlignRight methods 12016
UpdateTarget 12017
TRichEditBold class 12017
about TRichEditBold class 12017
ExecuteTarget 12022
TRichEditBold Members 12018
TRichEditBold methods 12022
UpdateTarget 12023
TRichEditBullets class 12023
about TRichEditBullets class 12023
ExecuteTarget 12029
TRichEditBullets Members 12024
TRichEditBullets methods 12028
UpdateTarget 12029
TRichEditItalic class 12029
about TRichEditItalic class 12029
ExecuteTarget 12035
TRichEditItalic Members 12030
TRichEditItalic methods 12035
UpdateTarget 12035
TRichEditStrikeOut class 12036
about TRichEditStrikeOut class 12036
ExecuteTarget 12041
TRichEditStrikeOut Members 12036
TRichEditStrikeOut methods 12041
UpdateTarget 12042
TRichEditUnderline class 12042
about TRichEditUnderline class 12042
ExecuteTarget 12047
TRichEditUnderline Members 12043
TRichEditUnderline methods 12047
UpdateTarget 12048
TriggerContext class 2101
about TriggerContext class 2101
Connection 2103
method 2102
NewRow 2103
OldRow 2103
TriggerContext fields 2102
TriggerContext Members 2102
TriggerContext properties 2102
TriggerReadRow class 2103
about TriggerReadRow class 2103
FieldCount 2117
FindModified 2105

GetArrayLength 2106
GetBoolean 2106
GetBoolean method 2106
GetByte 2107
GetByte method 2106
GetBytes 2107
GetBytes method 2107
GetColumnNames 2108
GetDateTime 2108
GetDateTime method 2108
GetDecimal 2109
GetDecimal method 2109
GetDouble 2109
GetDouble method 2109
GetFloat 2110
GetFloat method 2110
GetInt16 2111
GetInt16 method 2110
GetInt32 2111
GetInt32 method 2111
GetInt64 2112
GetInt64 method 2112
GetObject 2113
GetObject method 2112
GetOrdinal 2113
GetString 2113
GetString method 2113
IsAssignedNull 2114
IsAssignedNull method 2114
IsModified 2115
IsModified method 2114
IsNull 2115
IsNull method 2115
IsUnassignedNull 2116
IsUnassignedNull method 2116
method 2105
ToString 2116
TriggerReadRow fields 2105
TriggerReadRow Members 2104
TriggerReadRow methods 2105
TriggerReadRow properties 2117
TriggerReadWriteRow class 2117
about TriggerReadWriteRow class 2117
ClearValues 2119
SetAssignedNull 2119
SetAssignedNull method 2119
SetBoolean 2120
SetBoolean method 2120
SetByte 2121
SetByte method 2120
SetBytes 2121
SetBytes method 2121
SetDateTime 2122
SetDateTime method 2122
SetDecimal 2122
SetDecimal method 2122
SetDefault 2123
SetDefault method 2123
SetDefaultValues 2123
SetDouble 2124
SetDouble method 2124
SetFloat 2125
SetFloat method 2124
SetInt16 2125
SetInt16 method 2125
SetInt32 2126
SetInt32 method 2126
SetInt64 2127
SetInt64 method 2126
SetObject 2127
SetObject method 2127
SetString 2128
SetString method 2128
SetUnassignedNull 2129
SetUnassignedNull method 2128
TriggerReadWriteRow Members 2117
TriggerReadWriteRow methods 2118
TRIO class 18718

wwww

about TRIO class 18718
AfterConstruction 18724
BeforeDestruction 18725
Converter 18728
Create 18722
Destroy 18722
DoAfterExecute 18725
DoBeforeExecute 18725
FConverter 18723
FIID 18723
FWebNode 18723
GetResponseStream 18726
IntfMD 18723
NewInstance 18726
OnAfterExecute 18730
OnBeforeExecute 18730
OnGetAttachment 18731
OnSendAttachment 18731
QueryInterface 18727
RefCount 18728
SOAPHeaders 18728
TRIO events 18729
TRIO fields 18723
TRIO Members 18719
TRIO methods 18724
TRIO properties 18727
WebNode 18729
TRowCollection class 13167
about TRowCollection class 13167
Add 13170
Create 13170
GetItemAttr 13170
Notify 13171
TRowCollection Members 13168
TRowCollection methods 13170
TRowItem class 13171
about TRowItem class 13171
trVertical enumeration member 5471
tsAscending enumeration member 17388
tsAuto enumeration member 5469
TSaveDialog class 11489
about TSaveDialog class 11489
Execute 11494
TSaveDialog Members 11490
TSaveDialog methods 11494
TSavePicture class 12048
about TSavePicture class 12048
Caption 12054
Dialog 12054
Enabled 12055
GetDialogClass 12053
HelpContext 12055
HelpKeyword 12056
HelpType 12056
Hint 12057
ImageIndex 12057
OnAccept 12059
OnCancel 12059
OnHint 12059
OnUpdate 12059
SecondaryShortCuts 12057
ShortCut 12058
TSavePicture events 12058
TSavePicture Members 12049
TSavePicture methods 12053
TSavePicture properties 12054
Visible 12058
TSavePictureDialog class 13300
about TSavePictureDialog class 13300
Execute 13305
TSavePictureDialog Members 13301
TSavePictureDialog methods 13305
TSaveTextFileDialog class 13305
about TSaveTextFileDialog class 13305
Execute 13309
TSaveTextFileDialog Members 13305
TSaveTextFileDialog methods 13309
tsButtons enumeration member 5467

TScreen class 14197
about TScreen class 14197
ActiveControl 14206
ActiveCustomForm 14207
ActiveForm 14207
Create 14202
Cursor 14209
CursorCount 14212
Cursors 14212
CustomFormCount 14214
CustomForms 14215
DataModuleCount 14216
DataModules 14216
DefaultIme 14216
DefaultKbLayout 14217
DesktopHeight 14217
DesktopLeft 14218
DesktopRect 14218
DesktopTop 14219
DesktopWidth 14219
Destroy 14202
DisableAlign 14203
EnableAlign 14203
FocusedForm 14220
Fonts 14220
FormCount 14221
Forms 14223
Height 14225
HintFont 14226
IconFont 14226
Imes 14227
MenuFont 14227
MonitorCount 14228
MonitorFromPoint 14203
MonitorFromRect 14204
MonitorFromWindow 14204
Monitors 14228
OnActiveControlChange 14234
OnActiveFormChange 14236
PixelsPerInch 14228
PrimaryMonitor 14229
Realign 14205
ResetFonts 14205
SaveFocusedList 14229
TScreen events 14234
TScreen Members 14198
TScreen methods 14202
TScreen properties 14205
Width 14229
WorkAreaHeight 14231
WorkAreaLeft 14231
WorkAreaRect 14232
WorkAreaTop 14232
WorkAreaWidth 14233
TScriptCodeItem class 25490
about TScriptCodeItem class 25490
AssignString 25492
GetScript 25492
TScriptCodeItem Members 25491
TScriptCodeItem methods 25491
TScriptComponent class 19641
about TScriptComponent class 19641
Create 19642
GetComponent 19643
Obj 19643
TScriptComponent Members 19641
TScriptComponent methods 19642
TScriptComponent properties 19643
TScriptEnginesList class 25492
about TScriptEnginesList class 25492
AddEngine 25495
AddObjectFactory 25495
Count 25496
Create 25493
Destroy 25494
Find 25495
FindScriptProducerClass 25495
Items 25496

ListLanguageNames 25496
RemoveEngine 25496
TScriptEnginesList Members 25493
TScriptEnginesList methods 25494
TScriptEnginesList properties 25496
TScriptError class 25497
about TScriptError class 25497
Create 25498
FCharPos 25499
FDescription 25499
FFileName 25499
FLine 25499
FSourceLine 25500
GetCharPos 25500
GetDescription 25500
GetFileName 25500
GetLine 25501
GetSourceLine 25501
SetDescription 25501
TScriptError fields 25498
TScriptError Members 25497
TScriptError methods 25500
TScriptErrors class 25501
about TScriptErrors class 25501
Add 25504
Create 25503
Destroy 25503
GetError 25504
GetErrorCount 25504
TScriptErrors Members 25502
TScriptErrors methods 25504
TScriptFile class 25505
about TScriptFile class 25505
ApplicationName 25507
Create 25505
Destroy 25506
FLineNumberMap 25506
LongName 25507
ParseStream 25506
ParseString 25507
Script 25508
ShortName 25508
Source 25508
TScriptFile fields 25506
TScriptFile Members 25505
TScriptFile methods 25506
TScriptFile properties 25507
TScriptGlobalObjects class 25508
about TScriptGlobalObjects class 25508
AddNamedIntf 25511
Create 25510
Destroy 25510
FindNamedItemIntf 25511
GetNamedItemCount 25511
GetNamedItemName 25511
GetScriptEngine 25512
NamedItemCount 25512
NamedItemName 25512
Producer 25512
ScriptEngine 25513
TScriptGlobalObjects Members 25509
TScriptGlobalObjects methods 25510
TScriptGlobalObjects properties 25512
TScriptIncludeFile class 25513
about TScriptIncludeFile class 25513
Create 25514
LoadFile 25514
TScriptIncludeFile Members 25513
TScriptIncludeFile methods 25514
TScriptIncludeFiles class 25514
about TScriptIncludeFiles class 25514
Create 25516
GetScriptIncludeFile 25517
Items 25517
TScriptIncludeFiles Members 25515
TScriptIncludeFiles methods 25517
TScriptIncludeFiles properties 25517
TScriptIncludeItem class 25518

about TScriptIncludeItem class 25518
Create 25519
GetScript 25520
TScriptIncludeItem Members 25518
TScriptIncludeItem methods 25520
TScriptItem class 25520
about TScriptItem class 25520
Create 25521
GetScript 25522
Script 25522
TScriptItem Members 25520
TScriptItem methods 25521
TScriptItem properties 25522
TScriptItems class 25522
about TScriptItems class 25522
Create 25524
GetScriptItem 25525
Items 25525
TScriptItems Members 25523
TScriptItems methods 25524
TScriptItems properties 25525
TScriptObject class 19643
about TScriptObject class 19643
TScriptObjectFactories class 25526
about TScriptObjectFactories class 25526
Count 25530
Create 25527
CreateGlobalObject 25529
Destroy 25528
Factories 25530
GetCount 25529
GetFactory 25529
GetObjectList 25529
TScriptObjectFactories Members 25526
TScriptObjectFactories methods 25528
TScriptObjectFactories properties 25529
TScriptProducer class 25530
about TScriptProducer class 25530
Component 25539
Content 25539
Create 25532
Designing 25540
Destroy 25533
Errors 25540
Evaluate 25535
EvaluateExpression 25535
FErrors 25534
FScriptEngine 25534
FScriptFile 25534
GetContent 25535
GetErrors 25535
GetHTMLBlock 25536
GetHTMLBlockCount 25536
GetWebModuleContext 25536
HandleParserError 25536
HandleScriptError 25536
ImplHandleScriptError 25537
ParseStream 25537
ParseString 25537
ReplaceTags 25537
Script 25540
SetContent 25538
TScriptProducer fields 25533
TScriptProducer Members 25531
TScriptProducer methods 25534
TScriptProducer properties 25539
Write 25538
Write method 25538
WriteItem 25539
TScrollBar class 21821
about TScrollBar class 21821
Align 21835
Anchors 21839
BiDiMode 21840
Change 21833
Constraints 21840
Create 21832
CreateParams 21833

aaaa

CreateWnd 21834
Ctl3D 21841
DragCursor 21841
DragKind 21841
DragMode 21842
Enabled 21843
Kind 21844
LargeChange 21845
Max 21847
Min 21849
OnChange 21866
OnContextPopup 21867
OnDragDrop 21868
OnDragOver 21870
OnEndDock 21872
OnEndDrag 21873
OnEnter 21874
OnExit 21876
OnKeyDown 21878
OnKeyPress 21880
OnKeyUp 21882
OnMouseEnter 21883
OnMouseLeave 21884
OnScroll 21884
OnStartDock 21886
OnStartDrag 21886
PageSize 21851
ParentBiDiMode 21852
ParentCtl3D 21853
ParentShowHint 21853
PopupMenu 21853
Position 21858
Scroll 21834
SetParams 21834
ShowHint 21860
SmallChange 21861
TabOrder 21863
TabStop 21864
TScrollBar events 21866
TScrollBar Members 21822
TScrollBar methods 21833
TScrollBar properties 21835
Visible 21864
TScrollBox class 14238
about TScrollBox class 14238
Align 14251
Anchors 14255
AutoScroll 14255
AutoSize 14256
BevelEdges 14256
BevelInner 14257
BevelKind 14257
BevelOuter 14258
BevelWidth 14258
BiDiMode 14258
BorderStyle 14259
Color 14259
Constraints 14260
Create 14249
CreateParams 14250
Ctl3D 14261
DockSite 14261
DragCursor 14262
DragKind 14262
DragMode 14263
Enabled 14264
Font 14265
OnCanResize 14278
OnClick 14278
OnConstrainedResize 14280
OnContextPopup 14281
OnDblClick 14282
OnDockDrop 14282
OnDockOver 14284
OnDragDrop 14286
OnDragOver 14288
OnEndDock 14291
OnEndDrag 14291

bbbb

OnEnter 14292
OnExit 14294
OnGetSiteInfo 14296
OnMouseActivate 14298
OnMouseDown 14299
OnMouseEnter 14301
OnMouseLeave 14302
OnMouseMove 14302
OnMouseUp 14304
OnMouseWheel 14306
OnMouseWheelDown 14307
OnMouseWheelUp 14307
OnResize 14307
OnStartDock 14308
OnStartDrag 14309
OnUnDock 14309
Padding 14266
PaintWindow 14250
ParentBackground 14266
ParentBiDiMode 14266
ParentColor 14267
ParentCtl3D 14267
ParentFont 14268
ParentShowHint 14268
PopupMenu 14269
ShowHint 14273
TabOrder 14275
TabStop 14275
TScrollBox events 14277
TScrollBox Members 14239
TScrollBox methods 14250
TScrollBox properties 14251
Visible 14276
TScroller class 23134
about TScroller class 23134
Change 23146
Create 23145
Destroy 23145
Max 23147
Min 23147
OnClick 23148
Paint 23146
Position 23148
ScrollOrientation 23148
TScroller events 23148
TScroller Members 23134
TScroller methods 23146
TScroller properties 23146
TScrollingWinControl class 14312
about TScrollingWinControl class 14312
AdjustClientRect 14324
AlignControls 14325
AutoScroll 14329
AutoScrollEnabled 14325
AutoScrollInView 14325
ChangeScale 14326
Create 14323
CreateParams 14326
CreateWnd 14327
Destroy 14323
DisableAutoRange 14327
DoFlipChildren 14327
EnableAutoRange 14328
HorzScrollBar 14329
OnAlignInsertBefore 14332
OnAlignPosition 14332
Resizing 14328
ScrollInView 14328
TScrollingWinControl events 14331
TScrollingWinControl Members 14313
TScrollingWinControl methods 14324
TScrollingWinControl properties 14329
VertScrollBar 14330
tsDescending enumeration member 17388
TSearchAction class 20725
about TSearchAction class 20725
Create 20730
Destroy 20731

ExecuteTarget 20732
FControl 20731
FFindFirst 20731
HandlesTarget 20733
Notification 20733
Search 20733
TSearchAction fields 20731
TSearchAction Members 20726
TSearchAction methods 20732
UpdateTarget 20734
TSearchFind class 20734
about TSearchFind class 20734
BeforeExecute 20745
Caption 20740
Dialog 20740
Enabled 20741
GetDialogClass 20740
HelpContext 20741
HelpKeyword 20742
HelpType 20742
Hint 20743
ImageIndex 20743
OnAccept 20745
OnCancel 20745
OnHint 20745
OnUpdate 20746
SecondaryShortCuts 20743
ShortCut 20744
TSearchFind events 20744
TSearchFind Members 20735
TSearchFind methods 20739
TSearchFind properties 20740
Visible 20744
TSearchFindFirst class 20747
about TSearchFindFirst class 20747
Create 20752
TSearchFindFirst Members 20748
TSearchFindNext class 20753
about TSearchFindNext class 20753
Caption 20759
Create 20757
Enabled 20760
ExecuteTarget 20758
HandlesTarget 20758
HelpContext 20760
HelpKeyword 20761
HelpType 20761
Hint 20762
ImageIndex 20762
OnHint 20764
OnUpdate 20764
SearchFind 20762
SecondaryShortCuts 20763
ShortCut 20763
TSearchFindNext events 20764
TSearchFindNext Members 20753
TSearchFindNext methods 20758
TSearchFindNext properties 20759
UpdateTarget 20759
Visible 20763
TSearchReplace class 20765
about TSearchReplace class 20765
BeforeExecute 20777
Caption 20772
Dialog 20772
Enabled 20773
ExecuteTarget 20771
GetDialogClass 20771
HelpContext 20773
HelpKeyword 20774
HelpType 20774
Hint 20775
ImageIndex 20775
OnAccept 20777
OnCancel 20777
OnHint 20777
OnUpdate 20778
SecondaryShortCuts 20775

ShortCut 20776
TSearchReplace events 20776
TSearchReplace Members 20766
TSearchReplace methods 20771
TSearchReplace properties 20772
Visible 20776
TSelectionEditor class 11046
about TSelectionEditor class 11046
Create 11047
Designer 11049
ExecuteVerb 11048
GetVerb 11048
GetVerbCount 11048
PrepareItem 11048
RequiresUnits 11049
TSelectionEditor Members 11046
TSelectionEditor methods 11048
TSelectionEditor properties 11049
TSelectOptionsColumn class 16814
about TSelectOptionsColumn class 16814
Caption 16820
CaptionAttributes 16821
Custom 16821
DataSet 16822
DisplayRows 16822
FieldName 16822
IsColumn 16819
Items 16823
ItemsField 16823
Style 16824
StyleRule 16824
TabIndex 16825
TSelectOptionsColumn Members 16815
TSelectOptionsColumn methods 16819
TSelectOptionsColumn properties 16820
Values 16825
ValuesField 16825
TSendMail class 12061
about TSendMail class 12061
Caption 12067
Destroy 12065
Enabled 12067
ExecuteTarget 12065
HandlesTarget 12066
HelpContext 12068
HelpKeyword 12068
HelpType 12069
Hint 12069
ImageIndex 12069
OnHint 12071
OnUpdate 12072
SecondaryShortCuts 12070
ShortCut 12070
Text 12071
TSendMail events 12071
TSendMail Members 12061
TSendMail methods 12065
TSendMail properties 12067
UpdateTarget 12066
Visible 12071
TServerAcceptThread class 18825
about TServerAcceptThread class 18825
Create 18826
Execute 18827
ServerSocket 18831
TServerAcceptThread Members 18825
TServerAcceptThread methods 18826
TServerAcceptThread properties 18830
TServerClientThread class 18831
about TServerClientThread class 18831
ClientExecute 18834
ClientSocket 18838
Create 18833
Data 18838
Destroy 18833
DoTerminate 18835
EndConnect 18835
Error 18836

eeee

Event 18836
Execute 18836
HandleException 18837
KeepInCache 18839
ReActivate 18837
ServerSocket 18839
StartConnect 18837
TServerClientThread Members 18832
TServerClientThread methods 18834
TServerClientThread properties 18838
TServerClientWinSocket class 18840
about TServerClientWinSocket class 18840
Create 18842
Destroy 18842
ServerWinSocket 18843
TServerClientWinSocket Members 18840
TServerClientWinSocket properties 18843
TServerCollection class 17866
about TServerCollection class 17866
Create 17869
FindServer 17869
GetBalancedName 17870
GetNextName 17870
Items 17871
TServerCollection Members 17867
TServerCollection methods 17869
TServerCollection properties 17870
TServerEventDispatch class 18112
_AddRef 18114
_Release 18114
about TServerEventDispatch class 18112
Create 18113
GetIDsOfNames 18114
GetTypeInfo 18114
GetTypeInfoCount 18114
Invoke 18115
QueryInterface 18115
Server 18116
ServerDisconnect 18115
TServerEventDispatch Members 18112
TServerEventDispatch methods 18113
TServerEventDispatch properties 18115
TServerItem class 17871
about TServerItem class 17871
ComputerName 17874
Create 17873
Enabled 17875
GetDisplayName 17874
HasFailed 17876
Port 17876
TServerItem Members 17872
TServerItem methods 17874
TServerItem properties 17874
TServerSocket class 18843
about TServerSocket class 18843
Active 18849
Create 18848
OnAccept 18852
OnClientConnect 18852
OnClientDisconnect 18853
OnClientError 18854
OnClientRead 18855
OnClientWrite 18855
OnGetSocket 18856
OnGetThread 18856
OnListen 18857
OnThreadEnd 18857
OnThreadStart 18858
Port 18849
ServerType 18850
Service 18850
Socket 18851
ThreadCacheSize 18851
TServerSocket events 18852
TServerSocket Members 18844
TServerSocket properties 18848
TServerSocketThread class 20099
about TServerSocketThread class 20099

AddClientSocketThread 20101
ClearThreadPool 20102
Create 20100
CreateThread 20102
Destroy 20101
Execute 20102
FetchClientSocketThread 20103
OnGetThread 20105
RemoveClientSocketThread 20103
ServerSocket 20103
ThreadCacheSize 20104
ThreadPool 20104
TServerSocketThread events 20104
TServerSocketThread Members 20099
TServerSocketThread methods 20101
TServerSocketThread properties 20103
TServerWinSocket class 18858
about TServerWinSocket class 18858
Accept 18862
ActiveConnections 18867
ActiveThreads 18867
ClientConnect 18863
ClientDisconnect 18863
ClientErrorEvent 18863
ClientRead 18863
ClientWrite 18863
Connections 18868
Create 18861
Destroy 18861
Disconnect 18864
DoCreateThread 18864
GetClientSocket 18864
GetClientThread 18865
GetServerThread 18865
IdleThreads 18869
Listen 18865
OnClientConnect 18871
OnClientDisconnect 18872
OnClientError 18872
OnClientRead 18873
OnClientWrite 18874
OnGetSocket 18874
OnGetThread 18875
OnThreadEnd 18875
OnThreadStart 18876
ServerType 18869
SetServerType 18866
SetThreadCacheSize 18866
ThreadCacheSize 18870
ThreadEnd 18866
ThreadStart 18867
TServerWinSocket events 18870
TServerWinSocket Members 18859
TServerWinSocket methods 18861
TServerWinSocket properties 18867
TService class 22011, 25780
about TService class 22011, 25780
AfterConstruction 25785
AfterInstall 22028
AfterUninstall 22029
AllowPause 22021
AllowStop 22021
BeforeInstall 22029
BeforeUninstall 22030
Controller 22017
CreateNew 22015
Dependencies 22021
Destroy 22016
DisplayName 22022
DoContinue 22017
DoCustomControl 22017
DoInterrogate 22017
DoPause 22018
DoShutdown 22018
DoStart 22018
DoStop 22018
ErrCode 22022
ErrorSeverity 22022

GetName 25786
GetPorts 25786
GetServiceController 22018
Interactive 22023
LoadGroup 22023
LogMessage 22019
Main 22019
OnContinue 22030
OnExecute 22030
OnPause 22031
OnShutdown 22031
OnStart 22031
OnStop 22032
Param 22024
ParamCount 22024
Password 22024
ReportStatus 22020
ServiceStartName 22025
ServiceThread 22025
ServiceType 22026
SetName 25786
StartType 22026
Status 22026
TagID 22026
Terminated 22027
TService events 22028
TService Members 22012, 25781
TService methods 22016, 25785
TService properties 22020
WaitHint 22027
Win32ErrCode 22028
TServiceApplication class 22032
about TServiceApplication class 22032
Create 22036
CreateForm 22037
DelayInitialize 22039
Destroy 22036
DispatchServiceMain 22037
DoHandleException 22037
Hook 22038
Initialize 22038
Installing 22038
RegisterServices 22038
Run 22039
ServiceCount 22039
Title 22040
TServiceApplication Members 22033
TServiceApplication methods 22036
TServiceApplication properties 22039
TServices class 25787
about TServices class 25787
Add 25792
GetService 25793
Insert 25793
TServices Members 25787
TServices methods 25792
TServiceThread class 22040
about TServiceThread class 22040
Create 22042
Execute 22042
ProcessRequests 22046
TServiceThread Members 22041
TServiceThread methods 22042
TSession class 10093
about TSession class 10093
Active 10120
AddAlias 10100
AddDriver 10101
AddPassword 10102
AddStandardAlias 10102
AutoSessionName 10121
BDEOwnsLoginCbDb 10122
Close 10103
CloseDatabase 10103
ConfigMode 10122
Create 10098
DatabaseCount 10123
Databases 10123

hhhh

DeleteAlias 10104
DeleteDriver 10105
Destroy 10099
DropConnections 10105
FindDatabase 10105
GetAliasDriverName 10106
GetAliasNames 10107
GetAliasParams 10108
GetConfigParams 10108
GetDatabaseNames 10109
GetDriverNames 10110
GetDriverParams 10110
GetFieldNames 10110
GetPassword 10111
GetStoredProcNames 10112
GetTableNames 10113
Handle 10123
IsAlias 10113
KeepConnections 10124
Loaded 10114
Locale 10124
ModifyAlias 10114
ModifyConfigParams 10115
ModifyDriver 10115
NetFileDir 10125
Notification 10116
OnDBNotify 10128
OnPassword 10128
OnStartup 10129
Open 10116
OpenDatabase 10117
PrivateDir 10125
RemoveAllPasswords 10118
RemovePassword 10118
SaveConfigFile 10119
SessionName 10126
SetName 10119
SQLHourGlass 10126
TraceFlags 10127
TSession events 10128
TSession Members 10094
TSession methods 10099
TSession properties 10120
TSessionIDGen class 19015
about TSessionIDGen class 19015
Create 19015
NextID 19016
TSessionIDGen Members 19015
TSessionIDGen methods 19016
TSessionItem class 19016
about TSessionItem class 19016
Assign 19019
Create 19018
Destroy 19018
ExpirationTime 19020
Expired 19020
ID 19020
IsActive 19019
Items 19021
StartTime 19021
StatusFlags 19021
Terminated 19021
TimeoutMinutes 19022
TouchTime 19022
TSessionItem Members 19016
TSessionItem methods 19018
TSessionItem properties 19020
TSessionList class 10129
about TSessionList class 10129
Count 10133
Create 10131
CurrentSession 10134
Destroy 10131
FindSession 10132
GetSessionNames 10132
List 10134
OpenSession 10132
Sessions 10134

TSessionList Members 10130
TSessionList methods 10131
TSessionList properties 10133
TSessions class 19022
about TSessions class 19022
ActiveCount 19033
CheckStatusChange 19026
ConnectActiveSession 19026
ConnectSession 19027
Create 19025
CreateSessionIDGen 19027
Destroy 19025
DisconnectSession 19027
FileName 19033
GetExpirationTime 19027
GetExpired 19027
GetIDCount 19028
GetIDs 19028
GetIsActive 19028
GetItemCount 19028
GetItems 19029
GetSession 19029
GetSessionIDGen 19029
GetStartTime 19029
GetTerminated 19029
GetTimeoutMinutes 19030
GetTouchTime 19030
ItemValue 19030
LoadFromFile 19030
LoadFromStream 19031
MaxSessions 19033
NotifyStatusChange 19031
SaveToFile 19031
SaveToStream 19031
SessionExists 19031
SetItemValue 19032
SetTimeoutMinutes 19032
StartSession 19032
TerminateSession 19032
TouchSession 19033
TSessions Members 19022
TSessions methods 19025
TSessions properties 19033
TSessionSprig class 1745
about TSessionSprig class 1745
AnyProblems 1746
Caption 1746
Name 1746
TSessionSprig Members 1745
TSessionSprig methods 1746
TSessionsService class 25582
about TSessionsService class 25582
DefaultTimeout 25586
MaxSessions 25587
OnEndSession 25587
OnStartSession 25588
TSessionsService events 25587
TSessionsService Members 25583
TSessionsService properties 25586
TSetElementProperty class 11049
about TSetElementProperty class 11049
AllEqual 11053
Create 11052
Element 11056
GetAttributes 11053
GetIsDefault 11054
GetName 11054
GetValue 11055
GetValues 11055
SetValue 11056
TSetElementProperty Members 11050
TSetElementProperty methods 11053
TSetElementProperty properties 11056
TSetProperty class 11056
about TSetProperty class 11056
GetAttributes 11060
GetProperties 11060
GetValue 11061

jjjj

TSetProperty Members 11057
TSetProperty methods 11059
tsFlatButtons enumeration member 5467
TShape class 13172
about TShape class 13172
Align 13181
Anchors 13185
Brush 13185
Constraints 13186
Create 13179
Destroy 13180
DragCursor 13186
DragKind 13187
DragMode 13187
Enabled 13189
OnContextPopup 13194
OnDragDrop 13195
OnDragOver 13197
OnEndDock 13199
OnEndDrag 13200
OnMouseActivate 13201
OnMouseDown 13202
OnMouseEnter 13204
OnMouseLeave 13204
OnMouseMove 13205
OnMouseUp 13207
OnStartDock 13209
OnStartDrag 13210
Paint 13180
ParentShowHint 13189
Pen 13190
Shape 13190
ShowHint 13191
StyleChanged 13181
TShape events 13194
TShape Members 13172
TShape methods 13180
TShape properties 13181
Visible 13192
TSharedConnection class 16039
about TSharedConnection class 16039
AfterConnect 16048
AfterDisconnect 16049
BeforeConnect 16049
BeforeDisconnect 16049
ChildName 16047
ConnectEvent 16045
Create 16044
Destroy 16044
DoConnect 16045
DoDisconnect 16045
GetConnected 16046
GetServer 16046
Notification 16046
ParentConnection 16048
TSharedConnection events 16048
TSharedConnection Members 16040
TSharedConnection methods 16044
TSharedConnection properties 16047
TSharedImage class 14603
about TSharedImage class 14603
FreeHandle 14604
RefCount 14605
Reference 14604
Release 14604
TSharedImage Members 14604
TSharedImage methods 14604
TSharedImage properties 14605
TShellNameSpace class 19101
about TShellNameSpace class 19101
Columns 19117
Connect 19107
ConnectTo 19108
CountViewTypes 19118
Create 19105
CreateSubscriptionForSelection 19108
DefaultInterface 19118
DeleteSubscriptionForSelection 19108

Depth 19118
Destroy 19106
Disconnect 19109
EnumOptions 19118
Expand 19109
Export 19109
Flags 19118
Get_Columns 19110
Get_CountViewTypes 19110
Get_Depth 19110
Get_EnumOptions 19110
Get_Flags 19110
Get_Mode 19111
Get_Root 19111
Get_SelectedItem 19111
Get_SubscriptionsEnabled 19111
Get_TVFlags 19112
Import 19112
InitServerData 19112
InvokeContextMenuCommand 19112
InvokeEvent 19113
Mode 19119
MoveSelectionDown 19113
MoveSelectionTo 19113
MoveSelectionUp 19113
NewFolder 19114
OnDoubleClick 19120
OnFavoritesSelectionChange 19120
OnInitialized 19120
OnSelectionChange 19121
ResetSort 19114
Root 19119
SelectedItem 19119
SelectedItems 19114
Set_Columns 19114
Set_Depth 19115
Set_EnumOptions 19115
Set_Flags 19115
Set_Mode 19115
Set_Root 19115
Set_SelectedItem 19116
Set_TVFlags 19116
SetRoot 19116
SetViewType 19116
SubscriptionsEnabled 19119
Synchronize 19117
TShellNameSpace events 19120
TShellNameSpace Members 19101
TShellNameSpace methods 19107
TShellNameSpace properties 19117
TVFlags 19120
UnselectAll 19117
TShellResources class 19534
about TShellResources class 19534
TShellUIHelper class 19121
about TShellUIHelper class 19121
AddChannel 19127
AddDesktopComponent 19127, 19128
AddDesktopComponent method 19127
AddFavorite 19129
AddFavorite method 19129
AutoCompleteAttach 19129, 19130
AutoCompleteAttach method 19129
AutoCompleteSaveForm 19130
AutoCompleteSaveForm method 19130
AutoScan 19131
AutoScan method 19131
Connect 19131
ConnectTo 19132
Create 19125
DefaultInterface 19135
Destroy 19125
Disconnect 19132
ImportExportFavorites 19132
InitServerData 19133
IsSubscribed 19133
NavigateAndFind 19133
RefreshOfflineDesktop 19133

ResetFirstBootMode 19134
ResetSafeMode 19134
ShowBrowserUI 19134
TShellUIHelper Members 19121
TShellUIHelper methods 19126
TShellUIHelper properties 19134
TShellWindows class 19135
_NewEnum 19141
about TShellWindows class 19135
Connect 19141
ConnectTo 19141
Count 19146
Create 19139
DefaultInterface 19146
Destroy 19139
Disconnect 19142
FindWindowSW 19142
Get_Count 19142
InitServerData 19142
InvokeEvent 19143
Item 19144
Item method 19143
OnActivated 19144
OnCreated 19144
OnNavigate 19144
OnWindowRegistered 19146
OnWindowRevoked 19147
ProcessAttachDetach 19145
Register 19145
RegisterPending 19145
Revoke 19145
TShellWindows events 19146
TShellWindows Members 19135
TShellWindows methods 19140
TShellWindows properties 19146
TShortCutList class 328
about TShortCutList class 328
Add 332
IndexOfShortCut 332
ShortCuts 333
TShortCutList Members 329
TShortCutList methods 332
TShortCutList properties 333
TSimpleDataSet class 19555
about TSimpleDataSet class 19555
Active 19575
AfterApplyUpdates 19598
AfterCancel 19599
AfterClose 19601
AfterDelete 19602
AfterEdit 19605
AfterExecute 19606
AfterGetParams 19606
AfterGetRecords 19607
AfterInsert 19608
AfterOpen 19609
AfterPost 19610
AfterRefresh 19610
AfterRowRequest 19611
AfterScroll 19612
Aggregates 19577
AggregatesActive 19578
AllocConnection 19571
AllocDataSet 19571
AllocProvider 19572
AutoCalcFields 19578
BeforeApplyUpdates 19612
BeforeCancel 19613
BeforeClose 19613
BeforeDelete 19614
BeforeEdit 19614
BeforeExecute 19615
BeforeGetParams 19616
BeforeGetRecords 19616
BeforeInsert 19618
BeforeOpen 19619
BeforePost 19620
BeforeRefresh 19620

BeforeRowRequest 19621
BeforeScroll 19622
Connection 19579
Constraints 19579
Create 19570
DataSet 19580
Destroy 19571
DisableStringTrim 19580
FetchOnDemand 19581
FieldDefs 19582
FileName 19585
Filter 19585
Filtered 19587
FilterOptions 19588
IndexDefs 19588
IndexFieldNames 19589
IndexName 19590
Loaded 19572
MasterFields 19591
MasterSource 19592
Notification 19572
ObjectView 19592
OnCalcFields 19622
OnDeleteError 19623
OnEditError 19623
OnFilterRecord 19624
OnNewRecord 19625
OnPostError 19625
OnReconcileError 19625
OpenCursor 19573
PacketRecords 19593
Params 19594
PSGetCommandText 19573
ReadOnly 19596
SetConnection 19574
StoreDefs 19597
TSimpleDataSet events 19597
TSimpleDataSet Members 19556
TSimpleDataSet methods 19571
TSimpleDataSet properties 19574
TSimpleEvent class 22074
about TSimpleEvent class 22074
TSimpleObjectBroker class 17877
about TSimpleObjectBroker class 17877
Create 17880
Destroy 17880
GetComputerForGUID 17881
GetComputerForProgID 17882
GetPortForComputer 17882
LoadBalanced 17884
LoadFromStream 17883
SaveToStream 17883
ServerData 17885
Servers 17885
SetConnectStatus 17884
TSimpleObjectBroker Members 17877
TSimpleObjectBroker methods 17881
TSimpleObjectBroker properties 17884
TSimpleRWSync class 22772
about TSimpleRWSync class 22772
BeginRead 22774
BeginWrite 22775
Create 22773
Destroy 22774
EndRead 22775
EndWrite 22775
TSimpleRWSync Members 22772
TSimpleRWSync methods 22774
TSingleArray class 17356
about TSingleArray class 17356
Add 17359
Assign 17359
Create 17358
Find 17360
GetItem 17360
IndexOf 17360
Items 17361
PutItem 17360

nnnn

TSingleArray Members 17356
TSingleArray methods 17358
TSingleArray properties 17361
TSingleRecordActionRequestImpl class 915
about TSingleRecordActionRequestImpl class 915
ActionFieldValues 918
AdapterModes 919
Create 917
Destroy 918
TSingleRecordActionRequestImpl Members 915
TSingleRecordActionRequestImpl properties 918
TSiteModuleHelper class 25368
about TSiteModuleHelper class 25368
Create 25370
Destroy 25370
EndIterator 25371
FindNamedVariable 25371
FindVariable 25372
FindVariableContainer 25372
GetCurrentObject 25372
GetVariable 25372
GetVariableCount 25373
Module 25374
NextIteration 25373
Notification 25373
NotifyActivate 25373
NotifyDeactivate 25374
OnActivate 25376
OnDeactivate 25376
Request 25375
Response 25375
Session 25375
StartIterator 25374
TSiteModuleHelper events 25376
TSiteModuleHelper Members 25369
TSiteModuleHelper methods 25371
TSiteModuleHelper properties 25374
WebModuleContext 25376
TSitePageElements class 19844
about TSitePageElements class 19844
ServerFunctionsContent 19845
ServerScriptContent 19845
ServerVariablesContent 19846
StylesContent 19846
TSitePageElements fields 19845
TSitePageElements Members 19844
WarningsContent 19846
TSitePageModuleHelper class 25377
about TSitePageModuleHelper class 25377
Create 25379
DefaultAction 25386
Destroy 25380
DispatchPage 25380
GetDefaultAction 25381
GetPageProducer 25381
GetProducerComponent 25381
GetRedirectParams 25381
GetTemplateFileType 25382
GetTemplateStream 25382
GetXMLData 25382
GetXSLData 25382
HasHTMLBrowserView 25382
HasScriptView 25383
HasXMLBrowserView 25383
HasXSLBrowserView 25383
HTMLFromTemplate 25383
ImplDispatchPage 25384
ImplIncludePage 25384
ImplRedirectToPage 25384
ImplSetWebContentOptions 25384
IncludePage 25385
OnAfterDispatchPage 25386
OnAfterRedirectToPage 25387
OnBeforeDispatchPage 25387
OnBeforeRedirectToPage 25387
RedirectToPage 25385
SetDefaultPageFileName 25385
SetWebContentOptions 25385

TSitePageModuleHelper events 25386
TSitePageModuleHelper Members 25377
TSitePageModuleHelper methods 25380
TSitePageModuleHelper properties 25385
TSizeConstraints class 6107
about TSizeConstraints class 6107
AssignTo 6109
Change 6109
Control 6110
Create 6109
MaxHeight 6110
MaxWidth 6111
MinHeight 6111
MinWidth 6111
OnChange 6112
TSizeConstraints events 6111
TSizeConstraints Members 6107
TSizeConstraints methods 6109
TSizeConstraints properties 6110
TSmallIntArray class 17361
about TSmallIntArray class 17361
Add 17364
Assign 17365
Create 17363
GetItem 17365
Items 17366
PutItem 17366
TSmallIntArray Members 17362
TSmallIntArray methods 17364
TSmallIntArray properties 17366
TSmallintField class 7461
about TSmallintField class 7461
Create 7468
GetDataSize 7469
TSmallintField Members 7462
TSmallintField methods 7469
tsManual enumeration member 5469
tsModernPopout enumeration member 23231
tsModernTabs enumeration member 23231
tsNone enumeration member 5469, 17388
TSOAPAttachment class 15583
about TSOAPAttachment class 15583
CacheFile 15592
CacheFilePersist 15593
ContentType 15593
Create 15585
Destroy 15586
Encoding 15594
GetSourceStream 15587
Headers 15594
Init 15587
InternalSetCacheFile 15588
InternalSetSourceStream 15588
ObjectToSOAP 15588
Ownership 15595
SaveToFile 15589
SaveToStream 15589
SetOwnership 15590
SetSourceFile 15590
SetSourceStream 15591
SetSourceString 15591
SOAPToObject 15591
SourceStream 15595
SourceString 15596
TSOAPAttachment Members 15585
TSOAPAttachment methods 15586
TSOAPAttachment properties 15592
TSoapConnection class 19864
about TSoapConnection class 19864
AfterConnect 19880
AfterDisconnect 19880
Agent 19874
BeforeConnect 19880
BeforeDisconnect 19881
Connected 19875
Create 19869
Destroy 19869
DoConnect 19870

DoDisconnect 19870
GetConnected 19871
GetProviderNames 19871
GetServer 19871
GetServerList 19872
GetSOAPServer 19872
GetSOAPServerIID 19873
Notification 19873
OnAfterExecute 19881
OnBeforeExecute 19882
Password 19875
Proxy 19876
ProxyByPass 19876
RIO 19876
SetSOAPServerIID 19874
SetUseSoapAdapter 19874
SOAPServerIID 19877
TSoapConnection events 19879
TSoapConnection Members 19865
TSoapConnection methods 19870
TSoapConnection properties 19874
URL 19878
UserName 19878
UseSOAPAdapter 19879
TSoapDataList class 19848
about TSoapDataList class 19848
Add 19850
Assign 19850
Clear 19851
Insert 19851
Remove 19852
TSoapDataList Members 19848
TSoapDataList methods 19850
TSoapDataModule class 19884
_AddRef 19890
_Release 19890
about TSoapDataModule class 19884
AfterConstruction 19891
ApplyUpdates 19891
AS_ApplyUpdates 19892
AS_DataRequest 19893
AS_Execute 19893
AS_GetParams 19894
AS_GetProviderNames 19895
AS_GetRecords 19895
AS_RowRequest 19897
BeforeDestruction 19897
Create 19888
DataRequest 19898
Destroy 19889
Execute 19898
GetParams 19899
GetProvider 19899
GetProviderNames 19900
GetRecords 19900
Lock 19901
NewInstance 19901
ProviderCount 19908
Providers 19908
RefCount 19909
RegisterProvider 19902
RowRequest 19902
SafeCallException 19903
SAS_ApplyUpdates 19903
SAS_DataRequest 19903
SAS_Execute 19904
SAS_GetParams 19905
SAS_GetProviderNames 19906
SAS_GetRecords 19906
SAS_RowRequest 19907
TSoapDataModule Members 19885
TSoapDataModule methods 19889
TSoapDataModule properties 19908
Unlock 19907
UnRegisterProvider 19907
TSOAPDomConv class 18150
about TSOAPDomConv class 18150
AddAttachment 18157

qqqq

AddMultiRefNode 18158
AddNamespaceDecl 18158
AddObjectAsWriting 18158
CheckEncodingStyle 18158
ChildNodesAreNull 18159
ConvertByteArrayToSoap 18159
ConvertEnumToSoap 18159
ConvertNativeArrayToSoap 18159
ConvertNativeDataToSoap 18160
ConvertObjectToSOAP 18160
ConvertSoapToEnum 18161
ConvertSoapToNativeArray 18161
ConvertSoapToNativeArrayElem 18161
ConvertSoapToNativeData 18161
ConvertSOAPToObject 18162
ConvertSoapToVariant 18162
ConvertVariantToSoap 18162
Create 18155
CreateMultiRefNode 18163
CreateNULLNode 18163
CreateObjectNode 18163
CreateScalarNodeXS 18163
CreateTypedNode 18164
Destroy 18156
FinalizeMultiRefNodes 18164
FindAttachment 18164
FindMultiRefNodeByID 18164
FindMultiRefNodeByInstance 18165
FindNodeByHREF 18165
FindPrefixForURI 18165
GetDataNode 18165
GetElementType 18166
GetNewID 18166
GetNodeAsText 18166
GetObjectPropAsText 18166
GetOptions 18166
GetTypeBySchemaNS 18167
InitObjectFromSOAP 18167
IsNodeAVarArray 18167
IsObjectWriting 18168
LoadObject 18168
MakeArrayNode 18168, 18169
MakeArrayNode method 18168
MultiRefObject 18169
NodeIsNULL 18169
ObjectMemberNoShow 18169
ObjInstanceToSOAP 18169
OnMemberDataNotReceived 18177
OnUnhandledNode 18177
Options 18176
ReadHeader 18170
ReadRectDynArray 18170
ReadRectDynArrayElem 18171
ReadRow 18171
ReadVarArrayDim 18171
ReadVariant 18171
RemoveObjectAsWriting 18172
ResetMultiRef 18172
SerializationOptions 18172, 18173
SerializationOptions method 18172
SetNodeType 18173
SetObjectPropFromText 18173
SetOptions 18173
TSOAPDomConv events 18176
TSOAPDomConv Members 18151
TSOAPDomConv methods 18156
TSOAPDomConv properties 18176
UnhandledNode 18174
WriteHeader 18174
WriteNonRectDynArray 18174
WriteNonRectDynArrayElem 18174
WriteRectDynArray 18175
WriteRectDynArrayElem 18175
WriteVarArray 18175
WriteVarArrayAsB64 18175
WriteVariant 18176
TSOAPDOMProcessor class 19912
_AddRef 19916

_Release 19917
about TSOAPDOMProcessor class 19912
AddHeaderProcessor 19917
AfterConstruction 19918
DefaultProcessHeader 19918
Destroy 19915
FHeaderProcessors 19915
FindHeaderProcessor 19919
MakeHeaderNodes 19919
NewInstance 19919
TSOAPDOMProcessor fields 19915
TSOAPDOMProcessor Members 19912
TSOAPDOMProcessor methods 19916
TSOAPHeader class 15596
about TSOAPHeader class 15596
Actor 15599
MustUnderstand 15600
ObjectToSOAP 15598
SOAPToObject 15599
TSOAPHeader Members 15597
TSOAPHeader methods 15598
TSOAPHeader properties 15599
TSOAPHeaders class 15600
about TSOAPHeaders class 15600
Get 15603
Get method 15603
GetOwnsSentHeaders 15603
Send 15604
SendAt 15604
SendCount 15605
SetOwnsSentHeaders 15605
TSOAPHeaders Members 15601
TSOAPHeaders methods 15602
TSOAPHeadersBase class 15605
about TSOAPHeadersBase class 15605
FHeadersInbound 15607
FHeadersOutBound 15608
SetHeadersInOut 15608
TSOAPHeadersBase fields 15607
TSOAPHeadersBase Members 15606
TSOAPHeadersBase methods 15608
TSoapPascalInvoker class 20020
about TSoapPascalInvoker class 20020
AfterDispatchEvent 20028
BeforeDispatchEvent 20029
BeforeDispatchEvent2 20029
Converter 20028
Create 20024
Destroy 20024
FConverter 20025
FDomConverter 20025
FHeadersIn 20025
FHeadersOut 20025
FSOAPHeaders 20025
GetDomConverter 20026
GetInvoker 20026
Invoke 20026
Notification 20027
OnExceptionEvent 20030
OnGetAttachment 20030
OnSendAttachment 20030
TSoapPascalInvoker events 20028
TSoapPascalInvoker fields 20024
TSoapPascalInvoker Members 20021
TSoapPascalInvoker methods 20026
TSoapPascalInvoker properties 20028
TSocketConnection class 18928
about TSocketConnection class 18928
Address 18935
Create 18933
CreateTransport 18934
DoConnect 18934
Host 18936
InterceptGUID 18936
InterceptName 18937
ObjectBroker 18937
Port 18938
SupportCallbacks 18939

TSocketConnection Members 18929
TSocketConnection methods 18933
TSocketConnection properties 18935
TSocketTransport class 18939
about TSocketTransport class 18939
Address 18944
Create 18941
Destroy 18941
GetConnected 18942
GetWaitEvent 18943
Host 18945
InterceptGUID 18945
Port 18945
Receive 18943
Send 18943
SetConnected 18944
Socket 18945
TSocketTransport Members 18940
TSocketTransport methods 18942
TSocketTransport properties 18944
tsOwnerDraw enumeration member 23231
TSpeedButton class 2335
about TSpeedButton class 2335
Action 2350
ActionChange 2345
Align 2350
AllowAllUp 2354
Anchors 2355
BiDiMode 2355
Caption 2356
Click 2346
Constraints 2356
CopyImage 2346
Create 2344
Destroy 2344
Down 2357
Enabled 2357
Flat 2357
Font 2358
FState 2345
GetActionLinkClass 2346
GetPalette 2347
Glyph 2359
GroupIndex 2360
Layout 2360
Loaded 2347
Margin 2361
MouseDown 2348
MouseInControl 2361
MouseMove 2348
MouseUp 2349
NumGlyphs 2361
OnClick 2372
OnDblClick 2374
OnMouseActivate 2374
OnMouseDown 2375
OnMouseEnter 2377
OnMouseLeave 2378
OnMouseMove 2378
OnMouseUp 2380
Paint 2349
ParentBiDiMode 2362
ParentFont 2362
ParentShowHint 2363
PopupMenu 2363
ShowHint 2368
Spacing 2369
Transparent 2370
TSpeedButton events 2372
TSpeedButton fields 2344
TSpeedButton Members 2337
TSpeedButton methods 2345
TSpeedButton properties 2349
Visible 2370
TSpeedButtonActionLink class 2382
about TSpeedButtonActionLink class 2382
AssignClient 2386
Create 2385

FClient 2386
FImageIndex 2386
IsCheckedLinked 2387
IsGroupIndexLinked 2387
IsImageIndexLinked 2388
SetChecked 2388
SetGroupIndex 2389
SetImageIndex 2389
TSpeedButtonActionLink fields 2385
TSpeedButtonActionLink Members 2383
TSpeedButtonActionLink methods 2386
TSplitter class 13210
about TSplitter class 13210
Align 13224
AutoSnap 13228
Beveled 13228
CanResize 13220
Canvas 13228
Color 13231
Constraints 13231
Create 13218
Cursor 13232
Destroy 13220
DoCanResize 13221
MinSize 13235
MouseDown 13221
MouseMove 13222
MouseUp 13222
OnCanResize 13240
OnMoved 13240
OnPaint 13241
Paint 13222
ParentColor 13236
RequestAlign 13223
ResizeStyle 13237
StopSizing 13223
TSplitter events 13239
TSplitter Members 13211
TSplitter methods 13220
TSplitter properties 13224
Visible 13237
Width 13239
TSQLBlobStream class 20213
about TSQLBlobStream class 20213
Create 20215
Destroy 20216
ReadBlobData 20216
TSQLBlobStream Members 20213
TSQLBlobStream methods 20216
TSQLConnection class 20217
about TSQLConnection class 20217
ActiveStatements 20242
AfterConnect 20258
AfterDisconnect 20258
AutoClone 20243
BeforeConnect 20258
BeforeDisconnect 20259
BeginTransaction 20224
CheckConnection 20225
CheckDisconnect 20225
CloneConnection 20225
CloseDataSets 20225
Commit 20226
CommitAndFree 20226
Connected 20243
Connection 20244
ConnectionName 20244
ConnectionOptions 20227
ConnectionRegistryFile 20245
ConnectionState 20245
Create 20222
DataSets 20245
Destroy 20222
DoConnect 20227
DoDisconnect 20227
DriverName 20246
DriverRegistryFile 20247
Execute 20227

uuuu

ExecuteDirect 20228
GetConnected 20229
GetDataSet 20229
GetDefaultSchemaName 20229
GetDriverFunc 20247
GetFieldNames 20230
GetFieldNames method 20229
GetIndexNames 20231
GetIndexNames method 20230
GetLoginUsername 20231
GetPackageNames 20231
GetProcedureNames 20232
GetProcedureNames method 20232
GetProcedureParams 20233
GetProcedureParams method 20233
GetSchemaNames 20234
GetTableNames 20235
GetTableNames method 20234
InTransaction 20247
KeepConnection 20248
LastError 20248
LibraryName 20248
Loaded 20235
LoadParamsFromIniFile 20236
LoadParamsOnConnect 20249
LocaleCode 20250
LoginPrompt 20250
MaxStmtsPerConn 20251
MetaData 20252
MultipleTransactionsSupported 20253
Notification 20237
OnLogin 20259
OpenSchema 20237, 20238
OpenSchema method 20237
Params 20253
ParamsLoaded 20254
QuoteChar 20254
RegisterClient 20238
Rollback 20238
RollbackAndFree 20239
SetTraceEvent 20239
SQLConnection 20255
SQLDllHandle 20255
SQLError 20240
SQLHourGlass 20255
StartTransaction 20240
TableScope 20255
TraceCallbackEvent 20260
TransactionsSupported 20256
TSQLConnection events 20257
TSQLConnection Members 20218
TSQLConnection methods 20223
TSQLConnection properties 20241
UnRegisterClient 20241
VendorLib 20257
TSQLDataLink class 20260
about TSQLDataLink class 20260
ActiveChanged 20263
CheckBrowseMode 20264
Create 20263
GetDetailDataSet 20264
RecordChanged 20264
TSQLDataLink Members 20261
TSQLDataLink methods 20263
TSQLDataSet class 20265
about TSQLDataSet class 20265
CommandText 20279
CommandType 20280
Create 20278
DataSource 20281
ExecSQL 20278
MaxBlobSize 20282
ParamCheck 20282
Params 20283
SortFieldNames 20284
SQLConnection 20284
TSQLDataSet Members 20266
TSQLDataSet methods 20278

TSQLDataSet properties 20279
TSQLMonitor class 20285
about TSQLMonitor class 20285
Active 20293
AutoSave 20294
Create 20288
Destroy 20289
FileName 20294
InvokeCallBack 20290
LoadFromFile 20290
MaxTraceCount 20295
OnLogTrace 20297
OnTrace 20297
SaveToFile 20290
SetActive 20291
SetFileName 20291
SetSQLConnection 20291
SetStreamedActive 20291
SetTraceList 20292
SQLConnection 20295
SwitchConnection 20292
Trace 20292
TraceCount 20295
TraceList 20296
TSQLMonitor events 20296
TSQLMonitor Members 20285
TSQLMonitor methods 20289
TSQLMonitor properties 20292
UpdateTraceCallBack 20292
TSQLQuery class 20298
about TSQLQuery class 20298
Create 20310
DataSource 20313
Destroy 20311
ExecSQL 20311
GetMetadata 20313
MaxBlobSize 20314
ParamCheck 20314
Params 20315
PrepareStatement 20312
RowsAffected 20316
SQL 20316
SQLConnection 20317
Text 20318
TSQLQuery Members 20299
TSQLQuery methods 20311
TSQLQuery properties 20312
TSQLQueryTableProducer class 10842
about TSQLQueryTableProducer class 10842
Caption 10849
CaptionAlignment 10849
Columns 10850
Content 10847
Footer 10850
GetDataSet 10847
Header 10851
MaxRows 10851
Notification 10848
OnCreateContent 10853
OnFormatCell 10854
OnGetTableCaption 10854
Query 10852
RowAttributes 10852
SetDataSet 10848
TableAttributes 10853
TSQLQueryTableProducer events 10853
TSQLQueryTableProducer Members 10843
TSQLQueryTableProducer methods 10847
TSQLQueryTableProducer properties 10848
TSQLResolver class 18589
about TSQLResolver class 18589
Create 18593
Destroy 18594
DoDelete 18595
DoExecSQL 18596
DoGetValues 18596
DoInsert 18596
DoUpdate 18596

wwww

FParams 18594
FreeTreeData 18597
FSQL 18594
GenDeleteSQL 18597
GenInsertSQL 18597
GenSelectSQL 18598
GenUpdateSQL 18598
GenWhereSQL 18598
GetProvider 18598
InitializeConflictBuffer 18599
InitTreeData 18599
InternalDoUpdate 18599
Provider 18600
TSQLResolver fields 18594
TSQLResolver Members 18590
TSQLResolver methods 18595
TSQLResolver properties 18600
UseFieldInUpdate 18599
UseFieldInWhere 18600
TSQLStoredProc class 20318
about TSQLStoredProc class 20318
Create 20331
ExecProc 20331
MaxBlobSize 20333
NextRecordSet 20332
PackageName 20334
ParamCheck 20334
Params 20334
PrepareStatement 20332
SQLConnection 20335
StoredProcName 20336
TSQLStoredProc Members 20319
TSQLStoredProc methods 20331
TSQLStoredProc properties 20333
TSQLTable class 20337
about TSQLTable class 20337
Active 20353
Create 20349
DeleteRecords 20350
Destroy 20350
GetIndexNames 20351
IndexFieldCount 20354
IndexFieldNames 20355
IndexFields 20355
IndexName 20356
MasterFields 20356
MasterLink 20357
MasterSource 20357
MaxBlobSize 20358
OpenCursor 20351
PrepareStatement 20351
SetIndexField 20352
SQLConnection 20358
TableName 20359
TSQLTable Members 20338
TSQLTable methods 20350
TSQLTable properties 20352
TSQLTimeStampField class 7469
about TSQLTimeStampField class 7469
CopyData 7477
Create 7475
DisplayFormat 7480
EditMask 7482
GetAsDateTime 7477
GetAsFloat 7477
GetAsSQLTimeStamp 7478
GetAsString 7478
GetAsVariant 7478
GetDataSize 7478
GetDefaultWidth 7478
GetText 7479
SetAsDateTime 7479
SetAsFloat 7479
SetAsSQLTimeStamp 7479
SetAsString 7480
SetVarValue 7480
TSQLTimeStampField Members 7470
TSQLTimeStampField methods 7476

TSQLTimeStampField properties 7480
Value 7482
TSQLUpdateObject class 10135
about TSQLUpdateObject class 10135
GetSQL 10138
TSQLUpdateObject Members 10136
TSQLUpdateObject methods 10138
tsSoftTabs enumeration member 23231
tsStandard enumeration member 23231
tsSynonym enumeration member 20368
tsSysTable enumeration member 20368
tsTable enumeration member 20368
tsTabs enumeration member 5467
TStack class 5701
about TStack class 5701
PushItem 5703
TStack Members 5702
TStack methods 5703
TStandardAddRemoveItem class 20398
about TStandardAddRemoveItem class 20398
DrawBackground 20407
DrawGlyph 20408
TStandardAddRemoveItem Members 20399
TStandardAddRemoveItem methods 20407
TStandardButtonControl class 20408
about TStandardButtonControl class 20408
DrawFrame 20417
DrawGlyph 20418
DrawText 20418
TStandardButtonControl Members 20409
TStandardButtonControl methods 20417
TStandardColorMap class 2
about TStandardColorMap class 2
BtnFrameColor 7
BtnSelectedColor 7
BtnSelectedFont 7
Color 8
DisabledColor 8
DisabledFontColor 8
DisabledFontShadow 8
FontColor 9
FrameBottomRightInner 9
FrameBottomRightOuter 9
FrameTopLeftInner 10
FrameTopLeftOuter 10
HighlightColor 10
HotColor 10
HotFontColor 11
MenuColor 11
OnColorChange 12
SelectedColor 11
SelectedFontColor 11
SetColor 6
ShadowColor 12
TStandardColorMap events 12
TStandardColorMap Members 2
TStandardColorMap methods 6
TStandardColorMap properties 6
UnusedColor 12
UpdateColors 6
TStandardCustomizePopup class 20418
about TStandardCustomizePopup class 20418
CanAutoSize 20433
NCPaint 20434
PositionPopup 20434
TStandardCustomizePopup Members 20419
TStandardCustomizePopup methods 20433
TStandardDropDownButton class 20434
about TStandardDropDownButton class 20434
DrawFrame 20444
GetPopupClass 20444
TStandardDropDownButton Members 20435
TStandardDropDownButton methods 20443
TStandardMenuButton class 20444
about TStandardMenuButton class 20444
DrawBackground 20454
DrawDesignFocus 20454
DrawFrame 20454

yyyy

DrawText 20455
TStandardMenuButton Members 20445
TStandardMenuButton methods 20454
TStandardMenuExpandBtn class 20455
about TStandardMenuExpandBtn class 20455
DrawBackground 20464
DrawFrame 20465
TStandardMenuExpandBtn Members 20455
TStandardMenuExpandBtn methods 20464
TStandardMenuItem class 20465
about TStandardMenuItem class 20465
CMTextchanged 20475
DrawBackground 20475
DrawDesignFocus 20475
DrawEdge 20476
DrawGlyph 20476
DrawGlyphFrame 20476
DrawLargeGlyph 20476
DrawSeparator 20477
DrawText 20477
TStandardMenuItem Members 20466
TStandardMenuItem methods 20474
TStandardMenuPopup class 20478
about TStandardMenuPopup class 20478
CanAutoSize 20492
GetExpandBtnClass 20493
NCPaint 20493
PositionPopup 20493
TStandardMenuPopup Members 20478
TStandardMenuPopup methods 20492
TStandardToolScrollBtn class 20494
about TStandardToolScrollBtn class 20494
DrawFrame 20503
TStandardToolScrollBtn Members 20494
TStandardToolScrollBtn methods 20503
TStaticListAction class 15765
about TStaticListAction class 15765
Active 15770
Caption 15771
Enabled 15771
HelpContext 15771
Hint 15772
Images 15772
ItemIndex 15773
Items 15773
OnGetItem 15775
OnHint 15775
OnItemSelected 15775
OnUpdate 15776
SecondaryShortCuts 15773
ShortCut 15774
TStaticListAction events 15774
TStaticListAction Members 15766
TStaticListAction properties 15770
Visible 15774
TStaticListItems class 15777
about TStaticListItems class 15777
Notify 15779
TStaticListItems Members 15777
TStaticListItems methods 15779
Update 15780
TStaticText class 21887
about TStaticText class 21887
Align 21899
Alignment 21902
Anchors 21903
AutoSize 21903
BevelEdges 21904
BevelInner 21904
BevelKind 21904
BevelOuter 21905
BiDiMode 21905
BorderStyle 21906
Caption 21906
Color 21909
Constraints 21909
DragCursor 21910
DragKind 21910

DragMode 21911
Enabled 21912
FocusControl 21913
Font 21913
OnClick 21926
OnContextPopup 21928
OnDblClick 21928
OnDragDrop 21929
OnDragOver 21931
OnEndDock 21933
OnEndDrag 21934
OnMouseActivate 21935
OnMouseDown 21936
OnMouseEnter 21938
OnMouseLeave 21938
OnMouseMove 21939
OnMouseUp 21941
OnStartDock 21943
OnStartDrag 21944
ParentBiDiMode 21914
ParentColor 21915
ParentFont 21915
ParentShowHint 21916
PopupMenu 21916
ShowAccelChar 21921
ShowHint 21921
TabOrder 21923
TabStop 21923
Transparent 21924
TStaticText events 21925
TStaticText Members 21888
TStaticText properties 21898
Visible 21924
TStatusBar class 4769
about TStatusBar class 4769
Action 4781
Align 4781
Anchors 4785
AutoHint 4786
BiDiMode 4786
BorderWidth 4786
Color 4787
Constraints 4787
DragCursor 4788
DragKind 4788
DragMode 4789
Enabled 4790
Font 4791
OnClick 4804
OnContextPopup 4806
OnCreatePanelClass 4806
OnDblClick 4807
OnDragDrop 4807
OnDragOver 4809
OnDrawPanel 4812
OnEndDock 4813
OnEndDrag 4813
OnHint 4814
OnMouseActivate 4814
OnMouseDown 4815
OnMouseEnter 4818
OnMouseLeave 4818
OnMouseMove 4818
OnMouseUp 4820
OnResize 4823
OnStartDock 4823
OnStartDrag 4824
Panels 4792
ParentBiDiMode 4792
ParentColor 4793
ParentFont 4793
ParentShowHint 4794
PopupMenu 4794
ShowHint 4799
SimplePanel 4800
SimpleText 4801
SizeGrip 4801
TStatusBar events 4803

TStatusBar Members 4770
TStatusBar properties 4780
UseSystemFont 4801
Visible 4802
TStatusColumn class 16826
about TStatusColumn class 16826
Caption 16831
CaptionAttributes 16832
Custom 16832
IsColumn 16831
Style 16832
StyleRule 16833
TStatusColumn Members 16827
TStatusColumn methods 16830
TStatusColumn properties 16831
TStatusPanel class 4825
about TStatusPanel class 4825
Alignment 4829
Assign 4827
Bevel 4829
BiDiMode 4830
Create 4827
GetDisplayName 4828
ParentBiDiMode 4830
ParentBiDiModeChanged 4828
Style 4830
Text 4831
TStatusPanel Members 4825
TStatusPanel methods 4827
TStatusPanel properties 4829
UseRightToLeftAlignment 4828
UseRightToLeftReading 4828
Width 4834
TStatusPanels class 4834
about TStatusPanels class 4834
Add 4837
AddItem 4838
Create 4836
GetOwner 4839
Insert 4839
Items 4840
TStatusPanels Members 4834
TStatusPanels methods 4836
TStatusPanels properties 4840
Update 4839
TStoredProc class 10139
about TStoredProc class 10139
CopyParams 10154
Create 10152
CreateHandle 10154
DefineProperties 10155
DescriptionsAvailable 10155
Destroy 10153
Disconnect 10156
ExecProc 10156
GetParamsCount 10156
GetResults 10157
Handle 10163
Overload 10163
ParamBindMode 10164
ParamByName 10157
ParamCount 10165
Params 10165
Prepare 10157
Prepared 10165
PSExecute 10158
PSGetCommandText 10158
PSGetCommandType 10159
PSGetParams 10159
PSGetTableName 10160
PSSetCommandText 10160
PSSetParams 10161
SetDBFlag 10161
SetOverLoad 10161
SetPrepare 10162
SetPrepared 10162
SetProcName 10162
StmtHandle 10166

StoredProcName 10167
TStoredProc Members 10140
TStoredProc methods 10153
TStoredProc properties 10163
UnPrepare 10162
UpdateObject 10167
TStoredProcSprig class 1747
about TStoredProcSprig class 1747
AnyProblems 1747
Caption 1748
TStoredProcSprig Members 1747
TStoredProcSprig methods 1747
tsTrackBegin enumeration member 5464
tsTrackEnd enumeration member 5464
tsTrackMove enumeration member 5464
TStream class 2973
about TStream class 2973
CopyFrom 2979
FixupResourceHeader 2981
GetSize 2982
Position 2996
Read 2982
ReadBuffer 2983
ReadComponent 2983
ReadComponentRes 2987
ReadResHeader 2987
Seek 2988
Seek method 2988
SetSize 2989
SetSize method 2989
Size 2997
TStream Members 2978
TStream methods 2979
TStream properties 2996
Write 2989
WriteBuffer 2990
WriteComponent 2990
WriteComponentRes 2994
WriteDescendent 2994
WriteDescendentRes 2995
WriteResourceHeader 2995
TStreamAdapter class 2997
about TStreamAdapter class 2997
Clone 3000
Commit 3001
CopyTo 3001
Create 2999
Destroy 3000
LockRegion 3002
Read 3002
Revert 3002
Seek 3003
SetSize 3003
Stat 3004
Stream 3005
StreamOwnership 3005
TStreamAdapter Members 2998
TStreamAdapter methods 3000
TStreamAdapter properties 3005
UnlockRegion 3004
Write 3004
TStreamedConnection class 18945
_AddRef 18952
_Release 18952
about TStreamedConnection class 18945
Create 18950
CreateTransport 18952
Destroy 18951
DoConnect 18953
DoDisconnect 18953
DoError 18954
GetInterceptorList 18954
GetInterpreter 18955
GetServerList 18955
Handle 18959
InterceptGUID 18959
InterceptName 18960
InternalClose 18956

InternalOpen 18956
Interpreter 18960
QueryInterface 18956
Send 18957
SupportCallbacks 18961
ThreadException 18957
ThreadReceivedStream 18957
TStreamedConnection Members 18946
TStreamedConnection methods 18951
TStreamedConnection properties 18958
WndProc 18958
TStringArray class 17367
about TStringArray class 17367
Add 17370
Assign 17370
Create 17369
Exchange 17371
Find 17371
GetObject 17371
GetString 17372
Grow 17372
IndexOf 17372
Insert 17372
InvalidateItems 17373
PutObject 17373
PutString 17373
Sort 17373
Strings 17374
TStringArray Members 17367
TStringArray methods 17369
TStringArray properties 17374
TStringField class 7483
about TStringField class 7483
CheckTypeSize 7491
Create 7490
EditMask 7496
FixedChar 7497
GetAsBCD 7492
GetAsBoolean 7492
GetAsDateTime 7492
GetAsFloat 7492
GetAsInteger 7493
GetAsSQLTimeStamp 7493
GetAsString 7493
GetAsVariant 7493
GetDataSize 7493
GetDefaultWidth 7494
GetText 7494
GetValue 7494
SetAsBCD 7494
SetAsBoolean 7495
SetAsDateTime 7495
SetAsFloat 7495
SetAsInteger 7495
SetAsSQLTimeStamp 7495
SetAsString 7496
SetVarValue 7496
Size 7497
Transliterate 7498
TStringField Members 7484
TStringField methods 7491
TStringField properties 7496
Value 7499
TStringGrid class 14968
about TStringGrid class 14968
Cells 14986
Cols 14987
ColumnMoved 14984
Create 14983
Destroy 14983
DrawCell 14984
GetEditText 14985
Objects 14987
RowMoved 14985
Rows 14987
SetEditText 14985
TStringGrid Members 14969
TStringGrid methods 14983

TStringGrid properties 14986
TStringGridStrings class 14988
about TStringGridStrings class 14988
Add 14992
Assign 14992
Clear 14993
Create 14991
Delete 14993
Get 14994
GetCount 14994
GetObject 14994
Insert 14995
Put 14995
PutObject 14996
SetUpdateState 14996
TStringGridStrings Members 14989
TStringGridStrings methods 14992
TStringHash class 15471
about TStringHash class 15471
Add 15473
Clear 15473
Create 15472
Destroy 15472
Find 15473
HashOf 15474
Modify 15474
Remove 15474
TStringHash Members 15472
TStringHash methods 15473
ValueOf 15474
TStringList class 3006
about TStringList class 3006
Add 3010
AddObject 3012
CaseSensitive 3025
Changed 3013
Changing 3013
Clear 3013
CompareStrings 3014
CustomSort 3014
Delete 3015
Destroy 3009
Duplicates 3025
Exchange 3015
Find 3016
Get 3017
GetCapacity 3017
GetCount 3018
GetObject 3018
IndexOf 3019
Insert 3019
InsertItem 3020
InsertObject 3020
OnChange 3028
OnChanging 3028
Put 3022
PutObject 3022
SetCapacity 3022
SetUpdateState 3023
Sort 3023
Sorted 3026
TStringList events 3027
TStringList Members 3007
TStringList methods 3009
TStringList properties 3025
TStringProperty class 11061
about TStringProperty class 11061
AllEqual 11065
GetEditLimit 11065
GetValue 11065
SetValue 11066
TStringProperty Members 11062
TStringProperty methods 11064
TStrings class 3029
about TStrings class 3029
Add 3033
AddObject 3036
AddStrings 3039

eeee

Append 3041
Assign 3043
BeginUpdate 3045
Capacity 3070
Clear 3047
CommaText 3070
CompareStrings 3048
Count 3071
DefineProperties 3049
Delete 3049
DelimitedText 3074
Delimiter 3074
Destroy 3032
EndUpdate 3050
Equals 3051
Error 3052
Error method 3052
Exchange 3053
ExtractName 3053
Get 3053
GetCapacity 3054
GetCount 3054
GetEnumerator 3054
GetObject 3055
GetText 3055
GetTextStr 3056
IndexOf 3056
IndexOfName 3057
IndexOfObject 3059
Insert 3059
InsertObject 3060
LineBreak 3075
LoadFromFile 3061
LoadFromStream 3062
Move 3063
Names 3075
NameValueSeparator 3077
Objects 3077
Put 3064
PutObject 3065
QuoteChar 3080
SaveToFile 3065
SaveToStream 3067
SetCapacity 3068
SetText 3068
SetTextStr 3069
SetUpdateState 3069
StrictDelimiter 3080
Strings 3080
StringsAdapter 3084
Text 3084
TStrings Members 3030
TStrings methods 3032
TStrings properties 3069
UpdateCount 3085
ValueFromIndex 3085
Values 3086
TStringsAdapter class 1696
_NewEnum 1699
about TStringsAdapter class 1696
Add 1699
Clear 1699
Count 1700
Create 1698
Get_ControlDefault 1700
Get_Item 1700
ReferenceStrings 1700
ReleaseStrings 1700
Remove 1701
Set_ControlDefault 1701
Set_Item 1701
TStringsAdapter Members 1697
TStringsAdapter methods 1699
TStringsEnumerator class 3086
about TStringsEnumerator class 3086
Create 3087
Current 3088
GetCurrent 3087

MoveNext 3087
TStringsEnumerator Members 3086
TStringsEnumerator methods 3087
TStringsEnumerator properties 3088
TStringStream class 3088
about TStringStream class 3088
Create 3093
DataString 3104
Read 3097
ReadString 3098
Seek 3098
SetSize 3102
TStringStream Members 3092
TStringStream methods 3097
TStringStream properties 3104
Write 3103
WriteString 3103
TStringsValuesList class 24540
about TStringsValuesList class 24540
Actions 24549
Data 24549
OnPrepareStrings 24550
Strings 24549
TStringsValuesList events 24550
TStringsValuesList Members 24542
TStringsValuesList properties 24548
TSubmitQueryButton class 16833
about TSubmitQueryButton class 16833
Create 16838
EventContent 16838
TSubmitQueryButton Members 16834
TSubmitQueryButton methods 16838
TSummary class 17828
about TSummary class 17828
AddIndexInfo 17830
AddSubTotal 17830
AddSum 17830
AggDefinition 17832
Attributes 17833
ClearTotals 17831
Create 17829
CubeDimIndex 17833
Destroy 17829
FieldDefinition 17833
FieldName 17833
FieldType 17833
HasFlag 17831
IsDerived 17834
IsSparse 17831
MemoryUsage 17831
Name 17834
Position 17834
SetAggregator 17832
SumMethod 17834
TSummary events 17834
TSummary Members 17828
TSummary methods 17830
TSummary properties 17832
UpdateIndexInfo 17832
tsView enumeration member 20368
TSynchroObject class 22075
about TSynchroObject class 22075
Acquire 22076
Release 22076
TSynchroObject Members 22075
TSynchroObject methods 22076
TTabAction class 12073
about TTabAction class 12073
AfterTabChange 12082
BeforeTabChange 12082
Create 12078
HandlesTarget 12078
Notification 12079
OnValidateTab 12083
SelectNextTab 12079
SkipHiddenTabs 12080
TabControl 12080
TTabAction events 12081

gggggg

TTabAction Members 12074
TTabAction methods 12078
TTabAction properties 12080
UpdateTarget 12079
Wrap 12081
TTabbedNotebook class 23070
about TTabbedNotebook class 23070
ActivePage 23089
Align 23089
AlignControls 23082
Anchors 23093
Change 23083
Click 23083
CMTabFontChanged 23084
Constraints 23093
Create 23081
CreateHandle 23084
CreateParams 23084
Destroy 23082
Enabled 23094
Font 23094
GetChildOwner 23085
GetChildren 23085
GetIndexForPage 23085
KeyDown 23086
Loaded 23086
OnChange 23106
OnClick 23107
OnContextPopup 23107
OnEnter 23108
OnExit 23110
PageIndex 23095
Pages 23096
ParentShowHint 23096
PopupMenu 23097
ReadState 23087
SetPageIndex 23087
ShowControl 23088
ShowHint 23101
TabFont 23103
TabFontChanged 23088
TabOrder 23103
TabsPerRow 23104
TabStop 23104
TopFont 23104
TTabbedNotebook events 23106
TTabbedNotebook Members 23070
TTabbedNotebook methods 23082
TTabbedNotebook properties 23088
Visible 23105
TTabControl class 4841
about TTabControl class 4841
Align 4853
Anchors 4857
BiDiMode 4858
Constraints 4858
DisplayRect 4859
DockSite 4859
DragCursor 4860
DragKind 4860
DragMode 4861
Enabled 4862
Font 4863
HotTrack 4864
Images 4864
MultiLine 4865
MultiSelect 4866
OnChange 4884
OnChanging 4888
OnContextPopup 4889
OnDockDrop 4890
OnDockOver 4892
OnDragDrop 4894
OnDragOver 4896
OnDrawTab 4898
OnEndDock 4899
OnEndDrag 4899
OnEnter 4900

OnExit 4903
OnGetImageIndex 4904
OnGetSiteInfo 4904
OnMouseActivate 4907
OnMouseDown 4908
OnMouseEnter 4910
OnMouseLeave 4910
OnMouseMove 4910
OnMouseUp 4913
OnResize 4915
OnStartDock 4915
OnStartDrag 4916
OnUnDock 4917
OwnerDraw 4866
ParentBiDiMode 4866
ParentFont 4867
ParentShowHint 4867
PopupMenu 4868
RaggedRight 4873
ScrollOpposite 4873
ShowHint 4873
Style 4875
TabHeight 4875
TabIndex 4876
TabOrder 4878
TabPosition 4879
Tabs 4879
TabStop 4881
TabWidth 4881
TTabControl events 4884
TTabControl Members 4842
TTabControl properties 4853
Visible 4882
TTabDockPanel class 11625
about TTabDockPanel class 11625
AnimateSpeed 11642
BeginAnimation 11639
CMShowingChanged 11640
Create 11638
Destroy 11638
DockClientInfo 11642
EndAnimation 11640
ShouldAutoHide 11640
SplitterPanel 11642
SplitterWidth 11642
TTabDockPanel Members 11626
TTabDockPanel methods 11639
TTabDockPanel properties 11641
UpdateDockCaptionPin 11640
WMEraseBkgnd 11640
WMNCPaint 11641
WMPaint 11641
WMPrint 11641
WMPrintClient 11641
TTable class 10168
about TTable class 10168
AddIndex 10184
ApplyRange 10185
BatchMove 10186
CancelRange 10186
CloseIndexFile 10188
Constraints 10214
Create 10182
CreateHandle 10189
CreateTable 10189
DataEvent 10191
DefaultIndex 10214
DefChanged 10191
DeleteIndex 10191
DeleteTable 10192
Destroy 10182
DestroyHandle 10193
DestroyLookupCursor 10193
DoOnNewRecord 10194
EditKey 10194
EditRangeEnd 10194
EditRangeStart 10195
EmptyTable 10196

EncodeFieldDesc 10196
EncodeIndexDesc 10196
Exclusive 10215
Exists 10215
FieldDefs 10217
FindKey 10196
FindNearest 10197
GetCanModify 10198
GetDataSource 10199
GetDetailLinkFields 10199
GetFileName 10199
GetHandle 10199
GetIndexNames 10200
GetLanguageDriverName 10200
GetLookupCursor 10200
GetTableType 10200
GotoCurrent 10201
GotoKey 10201
GotoNearest 10202
IndexDefs 10219
IndexFieldCount 10220
IndexFieldNames 10220
IndexFields 10221
IndexFiles 10221
IndexName 10222
InitFieldDefs 10203
IsProductionIndex 10203
KeyExclusive 10223
KeyFieldCount 10223
LockTable 10203
MasterFields 10224
MasterLink 10224
MasterSource 10224
NativeTableName 10203
OpenIndexFile 10204
PrepareCursor 10204
PSGetCommandText 10204
PSGetCommandType 10205
PSGetDefaultOrder 10205
PSGetIndexDefs 10205
PSGetKeyFields 10206
PSGetTableName 10206
PSSetCommandText 10207
PSSetParams 10207
Ranged 10225
ReadOnly 10225
RenameTable 10208
SetKey 10208
SetRange 10209
SetRangeEnd 10211
SetRangeStart 10212
StoreDefs 10226
TableLevel 10226
TableName 10226
TableType 10227
TTable Members 10169
TTable methods 10183
TTable properties 10213
UnlockTable 10212
UpdateIndexDefs 10213
UpdateMode 10228
UpdateObject 10228
TTableFieldLinkProperty class 1748
about TTableFieldLinkProperty class 1748
Edit 1749
GetFieldNamesForIndex 1749
GetIndexBased 1750
GetIndexDefs 1750
GetIndexFieldNames 1750
GetIndexName 1750
GetMasterFields 1751
IndexBased 1752
IndexDefs 1752
IndexFieldNames 1752
IndexName 1752
MasterFields 1753
SetIndexFieldNames 1751
SetIndexName 1751

jjjj

SetMasterFields 1751
TTableFieldLinkProperty Members 1748
TTableFieldLinkProperty methods 1749
TTableFieldLinkProperty properties 1751
TTableSprig class 1753
about TTableSprig class 1753
AnyProblems 1754
Caption 1754
TTableSprig Members 1753
TTableSprig methods 1753
TTabList class 23149
about TTabList class 23149
Add 23153
AddStrings 23155
Clear 23157
Create 23152
Delete 23157
Insert 23158
Put 23158
TTabList Members 23149
TTabList methods 23152
TTabPage class 23112
about TTabPage class 23112
Caption 23124
Create 23122
Enabled 23126
Height 23127
ReadState 23123
TabOrder 23128
TTabPage Members 23112
TTabPage methods 23123
TTabPage properties 23124
Visible 23129
Width 23130
TTabSet class 23159
about TTabSet class 23159
Align 23178
Anchors 23182
AutoScroll 23182
BackgroundColor 23183
CanChange 23171
Canvas 23183
CMHintShow 23171
Constraints 23185
Create 23170
CreateParams 23172
Destroy 23170
DitherBackground 23186
DragCursor 23186
DragKind 23186
DragMode 23187
DrawTab 23172
Enabled 23188
EndMargin 23189
FirstIndex 23189
Font 23190
GetChildren 23172
GetImageIndex 23173
Images 23191
ItemAtPos 23173
ItemRect 23173
ItemWidth 23174
MeasureTab 23174
MinClientRect 23174
MinClientRect method 23174
MouseDown 23175
Notification 23176
OnAlignInsertBefore 23204
OnAlignPosition 23204
OnChange 23205
OnClick 23205
OnDragDrop 23207
OnDragOver 23210
OnDrawTab 23212
OnEndDock 23213
OnEndDrag 23213
OnEnter 23214
OnExit 23216

kkkk

OnGetImageIndex 23218
OnMeasureTab 23218
OnMouseActivate 23219
OnMouseDown 23220
OnMouseEnter 23222
OnMouseLeave 23222
OnMouseMove 23222
OnMouseUp 23225
OnStartDock 23227
OnStartDrag 23228
Paint 23176
ParentBackground 23191
ParentShowHint 23191
PopupMenu 23192
Scroller 23196
SelectedColor 23197
SelectNext 23177
SetTabIndex 23177
ShowHint 23197
ShrinkToFit 23198
SoftTop 23199
StartMargin 23199
Style 23199
TabHeight 23200
TabIndex 23200
TabPosition 23201
Tabs 23201
TTabSet events 23203
TTabSet Members 23159
TTabSet methods 23171
TTabSet properties 23177
UnselectedColor 23201
Visible 23201
VisibleTabs 23203
TTabSheet class 4919
about TTabSheet class 4919
BorderWidth 4934
Caption 4935
Constraints 4937
Create 4930
CreateParams 4932
Destroy 4932
DoHide 4933
DoShow 4933
DragMode 4938
Enabled 4940
Font 4940
Height 4941
Highlighted 4942
ImageIndex 4943
Left 4943
OnContextPopup 4961
OnDragDrop 4962
OnDragOver 4964
OnEndDrag 4966
OnEnter 4967
OnExit 4970
OnHide 4971
OnMouseActivate 4972
OnMouseDown 4973
OnMouseEnter 4975
OnMouseLeave 4975
OnMouseMove 4975
OnMouseUp 4978
OnResize 4980
OnShow 4980
OnStartDrag 4981
PageControl 4944
PageIndex 4946
ParentFont 4948
ParentShowHint 4948
PopupMenu 4949
ReadState 4933
ShowHint 4953
TabIndex 4955
TabVisible 4956
Top 4957
TTabSheet events 4960

TTabSheet Members 4920
TTabSheet methods 4932
TTabSheet properties 4934
UpdateControlOriginalParentSize 4934
Visible 4958
Width 4960
ttASCII enumeration member 10256
TTaskDialog class 11494
about TTaskDialog class 11494
Buttons 11499
Caption 11499
CommonButtons 11499
CustomFooterIcon 11500
CustomMainIcon 11500
DefaultButton 11500
ExpandButtonCaption 11500
ExpandedText 11501
Flags 11501
FooterIcon 11502
FooterText 11502
HelpContext 11502
MainIcon 11502
OnButtonClicked 11504
OnDialogConstructed 11504
OnDialogCreated 11505
OnDialogDestroyed 11505
OnExpanded 11505
OnHyperlinkClicked 11505
OnNavigated 11506
OnRadioButtonClicked 11506
OnTimer 11506
OnVerificationClicked 11506
ProgressBar 11503
RadioButtons 11503
Text 11503
Title 11503
TTaskDialog events 11504
TTaskDialog Members 11495
TTaskDialog properties 11498
VerificationText 11504
TTaskDialogBaseButtonItem class 11507
about TTaskDialogBaseButtonItem class 11507
Caption 11511
Click 11509
Client 11512
Create 11509
Default 11512
Destroy 11509
DoButtonClick 11510
DoSetEnabled 11510
Enabled 11512
GetButtonText 11510
GetDisplayName 11510
ModalResult 11512
SetInitialState 11511
TextWStr 11513
TTaskDialogBaseButtonItem Members 11507
TTaskDialogBaseButtonItem methods 11509
TTaskDialogBaseButtonItem properties 11511
TTaskDialogButtonItem class 11513
about TTaskDialogButtonItem class 11513
CommandLinkHint 11516
Create 11515
ElevationRequired 11516
GetButtonText 11515
ModalResult 11517
SetInitialState 11516
TTaskDialogButtonItem Members 11513
TTaskDialogButtonItem methods 11515
TTaskDialogButtonItem properties 11516
TTaskDialogButtons class 11517
about TTaskDialogButtons class 11517
Add 11519
Buttons 11519
DefaultButton 11521
FindButton 11520
GetEnumerator 11520
Items 11521

SetInitialState 11520
TTaskDialogButtons Members 11517
TTaskDialogButtons methods 11519
TTaskDialogButtons properties 11520
TTaskDialogButtonsEnumerator class 11521
about TTaskDialogButtonsEnumerator class 11521
Create 11522
Current 11523
GetCurrent 11522
MoveNext 11522
TTaskDialogButtonsEnumerator Members 11521
TTaskDialogButtonsEnumerator methods 11522
TTaskDialogButtonsEnumerator properties 11523
TTaskDialogProgressBar class 11523
about TTaskDialogProgressBar class 11523
Create 11525
Initialize 11525
MarqueeSpeed 11526
Max 11526
Min 11526
Position 11526
State 11527
TTaskDialogProgressBar Members 11524
TTaskDialogProgressBar methods 11525
TTaskDialogProgressBar properties 11525
TTaskDialogRadioButtonItem class 11527
about TTaskDialogRadioButtonItem class 11527
Create 11529
DoButtonClick 11529
DoSetEnabled 11529
TTaskDialogRadioButtonItem Members 11527
TTaskDialogRadioButtonItem methods 11529
TTcpClient class 20105
about TTcpClient class 20105
Active 20110
BlockMode 20110
Connected 20111
OnConnect 20112
OnCreateHandle 20113
OnDestroyHandle 20113
OnDisconnect 20113
OnError 20114
OnReceive 20114
OnSend 20114
RemoteHost 20111
RemotePort 20111
TTcpClient events 20112
TTcpClient Members 20106
TTcpClient properties 20109
TTcpServer class 20115
about TTcpServer class 20115
Active 20120
BlockMode 20120
LocalHost 20120
LocalPort 20121
OnAccept 20122
OnCreateHandle 20122
OnDestroyHandle 20122
OnGetThread 20123
OnListening 20123
TTcpServer events 20121
TTcpServer Members 20115
TTcpServer properties 20119
ttDBase enumeration member 10256
ttDefault enumeration member 10256
TTextAreaColumn class 16838
about TTextAreaColumn class 16838
Caption 16845
CaptionAttributes 16845
Custom 16846
DisplayRows 16846
DisplayWidth 16846
FieldName 16847
IsColumn 16844
ReadOnly 16847
Style 16847
StyleRule 16848
TabIndex 16848

TTextAreaColumn Members 16839
TTextAreaColumn methods 16844
TTextAreaColumn properties 16844
Wrap 16848
TTextAttributes class 4982
about TTextAttributes class 4982
Assign 4984
AssignTo 4986
Charset 4987
Color 4987
ConsistentAttributes 4989
Create 4983
Height 4989
InitFormat 4986
Name 4991
Pitch 4991
Protected 4992
Size 4992
Style 4992
TTextAttributes Members 4982
TTextAttributes methods 4984
TTextAttributes properties 4987
TTextColumn class 16849
about TTextColumn class 16849
Caption 16855
CaptionAttributes 16855
Custom 16856
DisplayWidth 16856
FieldName 16857
IsColumn 16854
ReadOnly 16857
Style 16857
StyleRule 16858
TabIndex 16858
TTextColumn Members 16850
TTextColumn methods 16854
TTextColumn properties 16855
ttFoxPro enumeration member 10256
TThread class 3108
about TThread class 3108
AfterConstruction 3114
CheckThreadError 3115
CheckThreadError method 3114
Create 3110
Destroy 3113
DoTerminate 3115
Execute 3115
FatalException 3133
FreeOnTerminate 3134
Handle 3137
OnTerminate 3148
Priority 3138
Queue 3119
Queue method 3119
RemoveQueuedEvents 3120
Resume 3120
ReturnValue 3141
StaticQueue 3124
StaticSynchronize 3125
Suspend 3125
Suspended 3142
Synchronize 3126
Synchronize method 3126
Terminate 3126
Terminated 3142
ThreadID 3148
TThread events 3148
TThread Members 3109
TThread methods 3114
TThread properties 3133
WaitFor 3132
TThreadCustomArray class 17374
about TThreadCustomArray class 17374
Add 17376
Create 17375
Destroy 17375
GetItem 17376
LockArray 17376

MemoryUsage 17376
TThreadCustomArray Members 17375
TThreadCustomArray methods 17375
UnlockArray 17376
TThreadList class 3149
about TThreadList class 3149
Add 3151
Clear 3157
Create 3150
Destroy 3150
Duplicates 3175
LockList 3157
Remove 3163
TThreadList Members 3150
TThreadList methods 3151
TThreadList properties 3174
UnlockList 3169
TThreadLocalCounter class 22775
about TThreadLocalCounter class 22775
Close 22776
Delete 22777
Destroy 22776
Open 22777
TThreadLocalCounter Members 22776
TThreadLocalCounter methods 22776
TTicks class 17424
about TTicks class 17424
Clear 17425
Create 17424
Destroy 17425
Dump 17425
NumberOfValues 17426
Ticks 17425
TicksIntArray 17426
TicksSmallIntArray 17426
TTicks Members 17424
TTicks methods 17425
TTicks properties 17426
TTimeField class 7499
about TTimeField class 7499
Create 7505
GetDataSize 7507
TTimeField Members 7500
TTimeField methods 7506
TTimeProperty class 11066
about TTimeProperty class 11066
GetAttributes 11070
GetValue 11070
SetValue 11071
TTimeProperty Members 11067
TTimeProperty methods 11069
TTimer class 13242
about TTimer class 13242
Create 13245
Destroy 13245
Enabled 13246
Interval 13249
OnTimer 13251
Timer 13246
TTimer events 13251
TTimer Members 13242
TTimer methods 13246
TTimer properties 13246
TToolBar class 4994
about TToolBar class 4994
Align 5022
AlignControls 5009
AllowTextButtons 5025
Anchors 5026
AutoSize 5026
BorderWidth 5026
ButtonCount 5027
ButtonHeight 5028
Buttons 5029
ButtonWidth 5029
CanAutoSize 5010
CancelMenu 5010
Canvas 5029

Caption 5030
ChangeScale 5010
CheckMenuDropdown 5011
ClickButton 5011
Color 5033
Constraints 5034
Create 5006
CreateParams 5012
CreateWnd 5012
Ctl3D 5035
CustomDraw 5012
CustomDrawButton 5013
Customizable 5035
CustomizeKeyName 5036
CustomizeValueName 5036
Destroy 5008
DisabledImages 5037
DockSite 5037
DoQueryDelete 5013
DoQueryInsert 5014
DragCursor 5038
DragKind 5039
DragMode 5039
DrawingStyle 5040
EdgeBorders 5041
EdgeInner 5041
EdgeOuter 5041
Enabled 5042
FindButtonFromAccel 5014
Flat 5042
FlipChildren 5014
Font 5043
GetChildren 5015
GetEnumerator 5015
GradientDirection 5044
GradientDrawButton 5016
GradientDrawingOptions 5044
GradientDrawToolBar 5016
GradientEndColor 5044
GradientStartColor 5044
Height 5045
HideClippedButtons 5046
HotImages 5046
HotTrackColor 5047
Images 5047
Indent 5047
InitMenu 5016
IsCustomDrawn 5016
List 5048
Loaded 5017
Menu 5048
Notification 5017
OnAdvancedCustomDraw 5062
OnAdvancedCustomDrawButton 5063
OnClick 5064
OnContextPopup 5066
OnCustomDraw 5066
OnCustomDrawButton 5067
OnCustomizeAdded 5068
OnCustomizeCanDelete 5068
OnCustomizeCanInsert 5069
OnCustomized 5069
OnCustomizeDelete 5070
OnCustomizeNewButton 5070
OnCustomizeReset 5071
OnCustomizing 5072
OnDblClick 5072
OnDockDrop 5073
OnDockOver 5075
OnDragDrop 5077
OnDragOver 5079
OnEndDock 5081
OnEndDrag 5082
OnEnter 5083
OnExit 5085
OnGetSiteInfo 5087
OnMouseActivate 5089
OnMouseDown 5090

qqqq

OnMouseEnter 5092
OnMouseLeave 5092
OnMouseMove 5093
OnMouseUp 5095
OnResize 5097
OnStartDock 5098
OnStartDrag 5098
OnUnDock 5099
ParentColor 5048
ParentFont 5049
ParentShowHint 5049
PopupMenu 5050
RepositionButton 5018
RepositionButtons 5018
Resize 5019
RowCount 5054
SetAutoSize 5019
ShowCaptions 5055
ShowHint 5057
TabOrder 5058
TabStop 5059
TrackMenu 5019
Transparent 5059
TToolBar events 5062
TToolBar Members 4995
TToolBar methods 5008
TToolBar properties 5020
Visible 5060
WndProc 5020
Wrapable 5061
WrapButtons 5020
TToolBarDockObject class 5102
about TToolBarDockObject class 5102
AdjustDockRect 5104
DrawDragDockImage 5104
EraseDragDockImage 5104
GetDragCursor 5105
ToolDockImage 5105
TToolBarDockObject Members 5102
TToolBarDockObject methods 5104
TToolBarEnumerator class 5105
about TToolBarEnumerator class 5105
Create 5106
Current 5107
GetCurrent 5106
MoveNext 5106
TToolBarEnumerator Members 5106
TToolBarEnumerator methods 5106
TToolBarEnumerator properties 5107
TToolButton class 5107
about TToolButton class 5107
Action 5127
ActionChange 5118
AllowAllUp 5127
AssignTo 5119
AutoSize 5128
BeginUpdate 5119
Caption 5128
CheckMenuDropdown 5120
Click 5120
Create 5116
Down 5128
DragCursor 5129
DragKind 5129
DragMode 5130
DropdownMenu 5131
Enabled 5132
EnableDropdown 5132
EndUpdate 5120
FToolBar 5117
GetActionLinkClass 5121
Grouped 5132
Height 5133
ImageIndex 5135
Indeterminate 5135
Index 5135
Marked 5136
MenuItem 5136

MouseDown 5121
MouseMove 5122
MouseUp 5122
Notification 5123
OnClick 5148
OnContextPopup 5150
OnDragDrop 5151
OnDragOver 5153
OnEndDock 5155
OnEndDrag 5156
OnMouseActivate 5157
OnMouseDown 5158
OnMouseEnter 5160
OnMouseLeave 5160
OnMouseMove 5161
OnMouseUp 5163
OnStartDock 5165
OnStartDrag 5166
Paint 5123
ParentShowHint 5137
PopupMenu 5137
RefreshControl 5124
SetAutoSize 5124
SetBounds 5124
SetToolBar 5125
ShowHint 5142
Style 5144
TToolButton events 5148
TToolButton fields 5117
TToolButton Members 5108
TToolButton methods 5118
TToolButton properties 5126
UpdateControl 5125
ValidateContainer 5126
Visible 5145
Width 5147
Wrap 5147
TToolButtonActionLink class 5166
about TToolButtonActionLink class 5166
AssignClient 5170
FClient 5169
IsCheckedLinked 5170
IsDropdownMenuLinked 5170
IsEnableDropdownLinked 5171
IsImageIndexLinked 5171
SetChecked 5171
SetDropdownMenu 5172
SetEnableDropdown 5172
SetImageIndex 5172
TToolButtonActionLink fields 5169
TToolButtonActionLink Members 5167
TToolButtonActionLink methods 5169
TToolDockForm class 23254
about TToolDockForm class 23254
CanResize 23269
Create 23268
CreateParams 23269
DoAddDockClient 23269
TToolDockForm Members 23254
TToolDockForm methods 23269
WMNCCreate 23270
WMNCHitTest 23270
WMNCLButtonDown 23270
WMSize 23270
WMSysCommand 23271
TToolDockObject class 23271
about TToolDockObject class 23271
AdjustDockRect 23273
Create 23273
DrawDragDockImage 23274
EraseDragDockImage 23274
TToolDockObject Members 23271
TToolDockObject methods 23273
TToolWindow class 23274
about TToolWindow class 23274
CMBorderChanged 23285
CMCtl3DChanged 23286
Create 23285

SSSS

EdgeBorders 23287
EdgeInner 23287
EdgeOuter 23288
NCPaint 23286
TToolWindow Members 23275
TToolWindow methods 23285
TToolWindow properties 23287
WMNCCalcSize 23286
WMNCPaint 23286
ttParadox enumeration member 10256
TTrackBar class 5172
about TTrackBar class 5172
Align 5188
Anchors 5191
BorderWidth 5192
Changed 5185
Constraints 5192
Create 5184
CreateParams 5185
CreateWnd 5186
Ctl3D 5193
DestroyWnd 5186
DragCursor 5193
DragKind 5194
DragMode 5194
Enabled 5196
Frequency 5196
LineSize 5196
Max 5197
Min 5197
OnChange 5212
OnContextPopup 5212
OnDragDrop 5213
OnDragOver 5215
OnEndDock 5217
OnEndDrag 5218
OnEnter 5219
OnExit 5221
OnKeyDown 5223
OnKeyPress 5226
OnKeyUp 5227
OnStartDock 5228
OnStartDrag 5229
Orientation 5198
PageSize 5198
ParentCtl3D 5198
ParentShowHint 5199
PopupMenu 5199
Position 5204
PositionToolTip 5204
SelEnd 5205
SelStart 5205
SetTick 5187
ShowHint 5206
ShowSelRange 5207
SliderVisible 5207
TabOrder 5208
TabStop 5208
ThumbLength 5209
TickMarks 5209
TickStyle 5209
TTrackBar events 5211
TTrackBar Members 5173
TTrackBar methods 5185
TTrackBar properties 5187
Visible 5210
TTransactionItem class 20359
about TTransactionItem class 20359
FNext 20360
FTransaction 20360
FTransactionDesc 20360
TTransactionItem fields 20360
TTransactionItem Members 20359
TTransportThread class 18961
about TTransportThread class 18961
Create 18963
Destroy 18963
Execute 18963

Semaphore 18967
TTransportThread Members 18962
TTransportThread methods 18963
TTransportThread properties 18967
TTrayIcon class 13252
about TTrayIcon class 13252
Animate 13256
AnimateInterval 13256
BalloonFlags 13257
BalloonHint 13257
BalloonTimeout 13257
BalloonTitle 13257
Hint 13258
Icon 13258
IconIndex 13258
Icons 13258
OnAnimate 13260
OnClick 13260
OnDblClick 13260
OnMouseDown 13260
OnMouseMove 13261
OnMouseUp 13261
PopupMenu 13259
TTrayIcon events 13259
TTrayIcon Members 13252
TTrayIcon properties 13256
Visible 13259
TTreeNode class 5230
about TTreeNode class 5230
AbsoluteIndex 5252
AlphaSort 5234
Assign 5234
Collapse 5235
Count 5253
Create 5232
CustomSort 5236
Cut 5254
Data 5254
Delete 5237
DeleteChildren 5238
Deleting 5256
Destroy 5233
DisplayRect 5238
DropTarget 5257
EditText 5240
EndEdit 5241
Expand 5242
Expanded 5257
Focused 5257
getFirstChild 5242
GetHandle 5243
GetLastChild 5243
GetNext 5243
GetNextChild 5244
getNextSibling 5245
GetNextVisible 5245
GetPrev 5246
GetPrevChild 5246
getPrevSibling 5247
GetPrevVisible 5247
GetState 5248
Handle 5258
HasAsParent 5248
HasChildren 5258
ImageIndex 5259
Index 5259
IndexOf 5248
IsFirstNode 5249
IsVisible 5259
Item 5260
ItemId 5260
Level 5260
MakeVisible 5249
MoveTo 5249
OverlayIndex 5263
Owner 5263
Parent 5263
Selected 5264

uuuu

SelectedIndex 5265
SetSelectedBit 5251
SetState 5251
StateIndex 5265
Text 5265
TreeView 5266
TTreeNode Members 5230
TTreeNode methods 5233
TTreeNode properties 5251
TTreeNodes class 5266
about TTreeNodes class 5266
Add 5270
AddChild 5271
AddChildFirst 5273
AddChildObject 5273
AddChildObjectFirst 5274
AddFirst 5275
AddItem 5275
AddNode 5276
AddObject 5276
AddObjectFirst 5279
AlphaSort 5279
Assign 5279
BeginUpdate 5280
Clear 5280
Count 5287
Create 5268
CreateItem 5281
CustomSort 5281
DefineProperties 5281
Delete 5282
Destroy 5269
EndUpdate 5283
GetCount 5283
GetEnumerator 5283
GetFirstNode 5284
GetNode 5285
Handle 5289
Insert 5285
InsertNode 5285
InsertObject 5286
Item 5289
Owner 5290
Reading 5290
SetItem 5286
SetUpdateState 5286
TTreeNodes Members 5266
TTreeNodes methods 5269
TTreeNodes properties 5286
TTreeNodesEnumerator class 5291
about TTreeNodesEnumerator class 5291
Create 5291
Current 5292
GetCurrent 5292
MoveNext 5292
TTreeNodesEnumerator Members 5291
TTreeNodesEnumerator methods 5291
TTreeNodesEnumerator properties 5292
TTreeView class 5293
about TTreeView class 5293
Align 5307
Anchors 5310
AutoExpand 5311
BevelEdges 5311
BevelInner 5312
BevelKind 5312
BevelOuter 5312
BevelWidth 5313
BiDiMode 5313
BorderStyle 5314
BorderWidth 5314
ChangeDelay 5315
Color 5315
Constraints 5316
Ctl3D 5316
DragCursor 5317
DragKind 5317
DragMode 5318

Enabled 5319
Font 5320
HideSelection 5321
HotTrack 5321
Images 5321
Indent 5322
Items 5324
MultiSelect 5325
MultiSelectStyle 5326
OnAddition 5342
OnAdvancedCustomDraw 5342
OnAdvancedCustomDrawItem 5342
OnChange 5343
OnChanging 5343
OnClick 5344
OnCollapsed 5345
OnCollapsing 5346
OnCompare 5346
OnContextPopup 5347
OnCreateNodeClass 5347
OnCustomDraw 5348
OnCustomDrawItem 5349
OnDblClick 5352
OnDeletion 5352
OnDragDrop 5352
OnDragOver 5355
OnEdited 5357
OnEditing 5357
OnEndDock 5358
OnEndDrag 5358
OnEnter 5359
OnExit 5362
OnExpanded 5363
OnExpanding 5364
OnGetImageIndex 5364
OnGetSelectedIndex 5365
OnKeyDown 5365
OnKeyPress 5368
OnKeyUp 5369
OnMouseActivate 5371
OnMouseDown 5372
OnMouseEnter 5374
OnMouseLeave 5374
OnMouseMove 5375
OnMouseUp 5377
OnStartDock 5379
OnStartDrag 5380
ParentBiDiMode 5326
ParentColor 5327
ParentCtl3D 5327
ParentFont 5328
ParentShowHint 5328
PopupMenu 5329
ReadOnly 5333
RightClickSelect 5333
RowSelect 5334
ShowButtons 5334
ShowHint 5335
ShowLines 5336
ShowRoot 5336
SortType 5337
StateImages 5337
TabOrder 5338
TabStop 5338
ToolTips 5339
TTreeView events 5341
TTreeView Members 5293
TTreeView properties 5305
Visible 5339
TTwilightColorMap class 13
about TTwilightColorMap class 13
BtnFrameColor 17
BtnSelectedColor 18
BtnSelectedFont 18
Color 18
DisabledColor 19
DisabledFontColor 19
DisabledFontShadow 19

wwww

FontColor 19
FrameBottomRightInner 20
FrameBottomRightOuter 20
FrameTopLeftInner 20
FrameTopLeftOuter 20
HighlightColor 21
HotColor 21
HotFontColor 21
MenuColor 21
OnColorChange 23
SelectedColor 22
SelectedFontColor 22
ShadowColor 22
TTwilightColorMap events 23
TTwilightColorMap Members 13
TTwilightColorMap methods 17
TTwilightColorMap properties 17
UnusedColor 23
UpdateColors 17
TTwoDimArray class 17377
about TTwoDimArray class 17377
columns 17379
Create 17377
Destroy 17378
Element 17379
mtxElements 17378
rows 17379
SetSize 17378
TTwoDimArray fields 17378
TTwoDimArray Members 17377
TTwoDimArray methods 17378
TTwoDimArray properties 17379
TTypedComObject class 5581
about TTypedComObject class 5581
GetClassInfo 5583
TTypedComObject Members 5582
TTypedComObject methods 5583
TTypedComObjectFactory class 5583
about TTypedComObjectFactory class 5583
ClassInfo 5587
Create 5585
GetInterfaceTypeInfo 5586
TTypedComObjectFactory Members 5584
TTypedComObjectFactory methods 5586
TTypedComObjectFactory properties 5587
UpdateRegistry 5586
TTypes class 25793
about TTypes class 25793
AfterConstruction 25798
GetSchemaDefs 25799
TTypes Members 25793
TTypes methods 25798
TTypeTranslator class 23320
about TTypeTranslator class 23320
Base64ToVar 23322
Base64ToVar method 23322
CastNativeToSoap 23322
CastSoapToNative 23323
CastSoapToVariant 23323
CastSoapToVariant method 23323
Create 23321
Destroy 23321
TTypeTranslator Members 23321
TTypeTranslator methods 23321
TUdpSocket class 20123
about TUdpSocket class 20123
Active 20128
BlockMode 20129
Create 20128
LocalHost 20129
LocalPort 20130
OnConnect 20131
OnCreateHandle 20132
OnDestroyHandle 20132
OnDisconnect 20132
OnError 20133
OnReceive 20133
OnSend 20133

RemoteHost 20130
RemotePort 20130
TUdpSocket events 20131
TUdpSocket Members 20124
TUdpSocket properties 20128
TUndoButton class 16858
about TUndoButton class 16858
Create 16863
TUndoButton Members 16859
TUpdateFileList class 24550
about TUpdateFileList class 24550
Count 24551
Create 24551
Files 24552
TUpdateFileList Members 24551
TUpdateFileList properties 24551
TUpdateSQL class 10229
about TUpdateSQL class 10229
Apply 10234
Apply method 10234
Create 10233
DatabaseName 10238
DataSet 10239
DeleteSQL 10239
Destroy 10233
ExecSQL 10235
GetDataSet 10235
GetSQL 10236
InsertSQL 10240
ModifySQL 10240
Query 10241
SessionName 10242
SetDataSet 10236
SetParams 10237
SetParams method 10236
SQL 10242
SQLChanged 10237
TUpdateSQL Members 10230
TUpdateSQL methods 10233
TUpdateSQL properties 10238
TUpdateSQLSprig class 1754
about TUpdateSQLSprig class 1754
AnyProblems 1755
TUpdateSQLSprig Members 1754
TUpdateSQLSprig methods 1755
TUpdateTree class 18600
about TUpdateTree class 18600
Clear 18606
Create 18603
Data 18610
Delta 18610
Destroy 18603
DetailCount 18610
DetailList 18610
Details 18610
DoUpdates 18606
ErrorDS 18611
FData 18604
FDeltaDS 18604
FDetails 18604
FErrorDS 18604
FName 18604
FOpened 18605
FParent 18605
FResolver 18605
FSourceDS 18605
GetDetail 18606
GetDetailCount 18607
GetErrorDS 18607
GetHasErrors 18607
GetIsNested 18607
GetTree 18608
HasErrors 18611
InitData 18608
InitDelta 18608
InitDelta method 18608
InitErrorPacket 18609
IsNested 18611

yyyy

Name 18611
Opened 18612
Parent 18612
RefreshData 18609
Resolver 18612
Source 18612
TUpdateTree fields 18603
TUpdateTree Members 18601
TUpdateTree methods 18606
TUpdateTree properties 18609
TUpDown class 5380
about TUpDown class 5380
AlignButton 5392
Anchors 5393
ArrowKeys 5393
Associate 5394
Constraints 5394
Enabled 5395
Hint 5395
Increment 5397
Max 5397
Min 5398
OnChanging 5409
OnChangingEx 5410
OnClick 5410
OnContextPopup 5412
OnEnter 5412
OnExit 5415
OnMouseActivate 5416
OnMouseDown 5417
OnMouseEnter 5420
OnMouseLeave 5420
OnMouseMove 5420
OnMouseUp 5422
Orientation 5398
ParentShowHint 5399
PopupMenu 5399
Position 5404
ShowHint 5404
TabOrder 5406
TabStop 5406
Thousands 5407
TUpDown events 5409
TUpDown Members 5381
TUpDown properties 5392
Visible 5407
Wrap 5408
TURLAction class 12083
about TURLAction class 12083
Caption 12089
Enabled 12089
HandlesTarget 12088
HelpContext 12090
HelpKeyword 12090
HelpType 12091
Hint 12091
ImageIndex 12091
OnHint 12093
OnUpdate 12094
SecondaryShortCuts 12092
ShortCut 12092
TURLAction events 12093
TURLAction Members 12084
TURLAction methods 12088
TURLAction properties 12089
UpdateTarget 12088
URL 12093
Visible 12093
TValueArray class 17671
about TValueArray class 17671
TValueListEditor class 23401
about TValueListEditor class 23401
AdjustColWidths 23416
AdjustRowCount 23416
Align 23428
Anchors 23432
BiDiMode 23432
BorderStyle 23433

CanEditModify 23417
Cells 23433
ColCount 23434
Color 23434
ColWidthsChanged 23417
Constraints 23435
Create 23415
CreateEditor 23417
CreateWnd 23418
Ctl3D 23435
DefaultColWidth 23436
DefaultDrawing 23437
DefaultRowHeight 23437
DeleteRow 23418
Destroy 23415
DisplayOptions 23438
DoExit 23419
DoOnValidate 23419
DragCursor 23438
DragKind 23438
DragMode 23439
DrawCell 23419
DropDownRows 23440
EditList 23441
EditListGetItems 23420
Enabled 23441
FindRow 23420
FixedColor 23442
FixedCols 23442
Font 23442
GetCell 23420
GetColCount 23421
GetEditLimit 23421
GetEditMask 23421
GetEditStyle 23421
GetEditText 23422
GetPickList 23422
GetRowCount 23422
GridLineWidth 23444
InsertRow 23422
IsEmptyRow 23423
ItemProps 23445
KeyDown 23423
KeyOptions 23445
Keys 23445
MouseDown 23424
OnClick 23461
OnContextPopup 23463
OnDblClick 23463
OnDragDrop 23464
OnDragOver 23466
OnDrawCell 23468
OnEditButtonClick 23470
OnEndDock 23470
OnEndDrag 23471
OnEnter 23472
OnExit 23474
OnGetEditMask 23476
OnGetEditText 23476
OnGetPickList 23477
OnKeyDown 23478
OnKeyPress 23480
OnKeyUp 23481
OnMouseActivate 23483
OnMouseDown 23484
OnMouseEnter 23486
OnMouseLeave 23487
OnMouseMove 23487
OnMouseUp 23489
OnMouseWheelDown 23492
OnMouseWheelUp 23492
OnRowMoved 23492
OnSelectCell 23493
OnSetEditText 23494
OnStartDock 23495
OnStartDrag 23495
OnStringsChange 23496
OnStringsChanging 23496

OnTopLeftChanged 23497
OnValidate 23497
Options 23446
ParentBiDiMode 23446
ParentColor 23447
ParentCtl3D 23447
ParentFont 23448
ParentShowHint 23448
PopupMenu 23449
Refresh 23424
Resize 23425
RestoreCurrentRow 23425
RowCount 23453
RowMoved 23425
ScrollBars 23454
SelectCell 23426
SetCell 23426
SetEditText 23426
SetStrings 23426
ShowHint 23454
Strings 23456
StringsChanging 23427
TabOrder 23456
TitleCaptions 23457
TitleCaptionsStored 23427
TValueListEditor events 23460
TValueListEditor Members 23401
TValueListEditor methods 23415
TValueListEditor properties 23427
Values 23457
Visible 23458
VisibleColCount 23459
VisibleRowCount 23459
TValueListStrings class 23498
about TValueListStrings class 23498
Assign 23502
Changed 23504
Changing 23505
Clear 23505
Create 23501
CustomSort 23505
Delete 23506
Destroy 23501
Exchange 23506
FindItemProp 23507
InsertItem 23507
ItemProps 23508
KeyIsValid 23507
Put 23507
TValueListStrings Members 23498
TValueListStrings methods 23501
TValueListStrings properties 23507
TValuesListNameField class 24552
about TValuesListNameField class 24552
GetDefaultFieldName 24557
ImplGetValue 24558
TValuesListNameField Members 24553
TValuesListNameField methods 24557
TValuesListValueField class 24558
about TValuesListValueField class 24558
GetDefaultFieldName 24564
ImplGetValue 24564
TValuesListValueField Members 24559
TValuesListValueField methods 24564
TVarBytesField class 7507
about TVarBytesField class 7507
Create 7513
GetDataSize 7514
SetAsByteArray 7514
TVarBytesField Members 7507
TVarBytesField methods 7513
TVariableLookup class 25540
about TVariableLookup class 25540
Create 25541
FindGlobalVariable 25542
FindGlobalVariableContainer 25542
Ident 25543
LookupObject 25542

bbbb

Obj 25543
TVariableLookup Members 25541
TVariableLookup methods 25542
TVariableLookup properties 25542
TVariantField class 7514
about TVariantField class 7514
CheckTypeSize 7522
Create 7520
GetAsBCD 7522
GetAsBoolean 7522
GetAsDateTime 7522
GetAsFloat 7522
GetAsInteger 7523
GetAsSQLTimeStamp 7523
GetAsString 7523
GetAsVariant 7523
GetDefaultWidth 7524
SetAsBCD 7524
SetAsBoolean 7524
SetAsDateTime 7524
SetAsFloat 7524
SetAsInteger 7525
SetAsSQLTimeStamp 7525
SetAsString 7525
SetVarValue 7525
TVariantField Members 7515
TVariantField methods 7521
TVariantProperty class 11071
about TVariantProperty class 11071
GetAttributes 11075
GetProperties 11075
GetValue 11076
SetValue 11076
TVariantProperty Members 11072
TVariantProperty methods 11074
TVirtualListAction class 15780
about TVirtualListAction class 15780
Active 15786
Caption 15786
Enabled 15787
HelpContext 15787
Hint 15788
Images 15788
ItemIndex 15788
OnGetItem 15790
OnGetItemCount 15790
OnHint 15791
OnItemSelected 15791
SecondaryShortCuts 15789
ShortCut 15789
TVirtualListAction events 15790
TVirtualListAction Members 15781
TVirtualListAction properties 15786
Visible 15789
TWebActionItem class 15130
about TWebActionItem class 15130
AssignTo 15134
Create 15133
Default 15135
Destroy 15133
Enabled 15136
GetDisplayName 15134
GetPathInfo 15135
MethodType 15137
Name 15137
OnAction 15139
PathInfo 15138
Producer 15138
ProducerContent 15139
SetDisplayName 15135
TWebActionItem events 15139
TWebActionItem Members 15131
TWebActionItem methods 15133
TWebActionItem properties 15135
TWebActionItems class 15140
about TWebActionItems class 15140
Add 15143
Create 15142

GetAttr 15144
GetAttrCount 15144
GetItemAttr 15144
GetOwner 15145
Items 15146
SetItemName 15145
TWebActionItems Members 15141
TWebActionItems methods 15143
TWebActionItems properties 15146
Update 15146
WebDispatcher 15147
TWebAppAutoObjectFactory class 3410
about TWebAppAutoObjectFactory class 3410
Create 3410
TWebAppAutoObjectFactory Members 3410
TWebAppComponents class 24958
about TWebAppComponents class 24958
AdapterDispatcher 24963
ApplicationAdapter 24963
DispatchActions 24964
EndUserAdapter 24964
LocateFileService 24965
OnAfterDispatch 24967
OnBeforeDispatch 24967
OnException 24968
PageDispatcher 24965
Sessions 24966
TWebAppComponents events 24966
TWebAppComponents Members 24959
TWebAppComponents properties 24963
UserListService 24966
TWebAppDataModule class 25388
about TWebAppDataModule class 25388
AppServices 25392
OnActivate 25393
OnDeactivate 25393
TWebAppDataModule events 25393
TWebAppDataModule Members 25388
TWebAppDataModule properties 25392
TWebAppDataModuleFactory class 25009
about TWebAppDataModuleFactory class 25009
Create 25011
GetIsAppModule 25011
TWebAppDataModuleFactory Members 25010
TWebAppDataModuleFactory methods 25011
TWebApplication class 24618
about TWebApplication class 24618
Create 24622
CreateForm 24623
Destroy 24622
Initialize 24623
Run 24623
Title 24624
TWebApplication Members 24619
TWebApplication methods 24622
TWebApplication properties 24624
TWebApplicationStub class 1427
about TWebApplicationStub class 1427
Create 1429
Get_Designing 1429
Get_ModuleFileName 1429
Get_ModulePath 1430
Get_Title 1430
QualifyFileName 1430
TWebApplicationStub Members 1427
TWebApplicationStub methods 1429
TWebAppPageModule class 25394
about TWebAppPageModule class 25394
AppServices 25399
DefaultAction 25400
OnActivate 25401
OnAfterDispatchPage 25401
OnBeforeDispatchPage 25402
OnDeactivate 25402
PageProducer 25400
TWebAppPageModule events 25400
TWebAppPageModule Members 25395
TWebAppPageModule properties 25399

TWebAppPageModuleFactory class 25012
about TWebAppPageModuleFactory class 25012
Create 25013
GetIsAppModule 25014
TWebAppPageModuleFactory Members 25013
TWebAppPageModuleFactory methods 25014
TWebAppPageModuleHelper class 25403
about TWebAppPageModuleHelper class 25403
GetPageProducer 25405
TWebAppPageModuleHelper Members 25403
TWebAppPageModuleHelper methods 25405
TWebBrowser class 19147
about TWebBrowser class 19147
AddressBar 19175
Align 19175
Anchors 19179
Application 19179
Busy 19180
ClientToWindow 19163
Container 19180
ControlInterface 19181
CreateControl 19163
DefaultInterface 19181
Document 19182
DragCursor 19182
DragMode 19182
ExecWB 19163
ExecWB method 19163
FullName 19184
FullScreen 19184
Get_Application 19164
Get_Container 19164
Get_Document 19164
Get_Parent 19165
GetProperty 19165
GoBack 19165
GoForward 19166
GoHome 19166
GoSearch 19167
HWnd 19185
InitControlData 19167
LocationName 19185
LocationURL 19185
MenuBar 19186
Name 19186
Navigate 19168
Navigate method 19167
Navigate2 19169
Navigate2 method 19169
Offline 19186
OnBeforeNavigate2 19200
OnClientToHostWindow 19200
OnCommandStateChange 19201
OnDocumentComplete 19201
OnDownloadBegin 19202
OnDownloadComplete 19202
OnDragDrop 19203
OnDragOver 19205
OnEndDrag 19207
OnEnter 19208
OnExit 19211
OnFileDownload 19212
OnFullScreen 19213
OnMenuBar 19213
OnNavigateComplete2 19213
OnNavigateError 19214
OnNewWindow2 19214
OnPrintTemplateInstantiation 19214
OnPrintTemplateTeardown 19215
OnPrivacyImpactedStateChange 19215
OnProgressChange 19215
OnPropertyChange 19215
OnQuit 19216
OnSetSecureLockIcon 19216
OnStartDrag 19216
OnStatusBar 19217
OnStatusTextChange 19217
OnTheaterMode 19217

eeee

OnTitleChange 19218
OnToolBar 19218
OnUpdatePageStatus 19218
OnVisible 19219
OnWindowClosing 19219
OnWindowSetHeight 19219
OnWindowSetLeft 19219
OnWindowSetResizable 19220
OnWindowSetTop 19220
OnWindowSetWidth 19220
Parent 19187
ParentShowHint 19187
Path 19188
PopupMenu 19188
PutProperty 19171
QueryStatusWB 19171
Quit 19171
ReadyState 19193
Refresh 19172
Refresh2 19172
Refresh2 method 19172
RegisterAsBrowser 19193
RegisterAsDropTarget 19194
Resizable 19194
ShowBrowserBar 19173
ShowBrowserBar method 19173
ShowHint 19194
Silent 19196
StatusBar 19196
StatusText 19196
Stop 19174
TabOrder 19197
TabStop 19197
TheaterMode 19198
ToolBar 19198
TopLevelContainer 19198
TWebBrowser events 19199
TWebBrowser Members 19148
TWebBrowser methods 19162
TWebBrowser properties 19174
type_ 19199
Visible 19199
TWebBrowser_V1 class 19221
about TWebBrowser_V1 class 19221
Align 19241
Anchors 19245
Application 19245
Busy 19245
Container 19246
ControlInterface 19246
CreateControl 19235
DefaultInterface 19246
Document 19246
DragCursor 19246
DragMode 19247
Get_Application 19236
Get_Container 19236
Get_Document 19236
Get_Parent 19236
GoBack 19236
GoForward 19237
GoHome 19237
GoSearch 19237
InitControlData 19237
LocationName 19248
LocationURL 19249
Navigate 19238, 19239
Navigate method 19237
OnBeforeNavigate 19261
OnCommandStateChange 19261
OnDownloadBegin 19261
OnDownloadComplete 19261
OnDragDrop 19261
OnDragOver 19264
OnEndDrag 19266
OnEnter 19267
OnExit 19270
OnFrameBeforeNavigate 19271

OnFrameNavigateComplete 19271
OnFrameNewWindow 19272
OnNavigateComplete 19272
OnNewWindow 19272
OnProgressChange 19272
OnPropertyChange 19272
OnQuit 19273
OnStartDrag 19273
OnStatusTextChange 19274
OnTitleChange 19274
OnWindowActivate 19274
OnWindowMove 19274
OnWindowResize 19274
Parent 19249
ParentShowHint 19250
PopupMenu 19251
Refresh 19239
Refresh2 19240
Refresh2 method 19239
ShowHint 19256
Stop 19240
TabOrder 19257
TabStop 19258
TopLevelContainer 19258
TWebBrowser_V1 events 19260
TWebBrowser_V1 Members 19221
TWebBrowser_V1 methods 19235
TWebBrowser_V1 properties 19240
type_ 19258
Visible 19258
TWebButton class 16863
about TWebButton class 16863
Caption 16870
Content 16868
Create 16868
Custom 16871
GetCaption 16869
GetLayoutAttributes 16869
ImplContent 16869
SetCaption 16870
Style 16871
StyleRule 16872
TWebButton Members 16864
TWebButton methods 16868
TWebButton properties 16870
TWebComponentClassDescription class 24706
about TWebComponentClassDescription class 24706
ComponentClass 24708
ComponentClassName 24708
Editor 24708
FrameClass 24708
TWebComponentClassDescription fields 24707
TWebComponentClassDescription Members 24706
TWebComponentList class 24708
_AddRef 24712
_Release 24712
about TWebComponentList class 24708
Add 24713
Changed 24713
Clear 24713
Count 24718
Create 24711
Destroy 24711
GetComponent 24714
GetCount 24714
GetOwner 24714
IndexOf 24715
LockNotify 24715
NewComponent 24715
NotifyLocked 24715
OnChange 24719
ParentComponent 24718
QueryInterface 24716
Remove 24716
SetComponentIndex 24716
TWebComponentList events 24719
TWebComponentList Members 24709
TWebComponentList methods 24712

TWebComponentList properties 24717
UnlockNotify 24717
WebComponentContainer_GetComponent 24717
WebComponentContainer_GetComponentCount 24717
WebComponents 24718
TWebComponentsEditorHelper class 24719
_AddRef 24721
_Release 24721
about TWebComponentsEditorHelper class 24719
CanAddClassHelper 24721
ImplCanAddClassHelper 24721
QueryInterface 24722
TWebComponentsEditorHelper Members 24720
TWebComponentsEditorHelper methods 24721
TWebConnection class 18968
about TWebConnection class 18968
Agent 18979
Create 18973
CreateTransport 18974
Destroy 18974
DoConnect 18975
GetInterpreter 18975
GetWaitEvent 18976
ObjectBroker 18979
Password 18980
Proxy 18980
ProxyByPass 18980
Receive 18976
Send 18977
SetConnected 18977
SupportCallbacks 18981
Transport_GetConnected 18978
Transport_SetConnected 18978
TWebConnection Members 18969
TWebConnection methods 18974
TWebConnection properties 18978
URL 18981
UserName 18982
TWebContainedComponent class 24722
about TWebContainedComponent class 24722
Create 24726
Destroy 24726
GetContainerList 24727
GetParentComponent 24727
HasParent 24727
IsDefaultField 24728
ParentChanged 24728
ReadState 24729
SetParentComponent 24729
SetWebParent 24730
TWebContainedComponent Members 24723
TWebContainedComponent methods 24726
TWebContainedComponent properties 24730
WebContainedComponentHelper 24730
WebParent 24731
TWebContainedComponentHelper class 24731
about TWebContainedComponentHelper class 24731
ContainerList 24737
Create 24733
Destroy 24733
GetContainer 24734
GetIndex 24734
IsDefaultField 24735
OnParentChanged 24738
ParentChanged 24735
SetComponentList 24735
SetContainer 24736
SetIndex 24736
SetWebParent 24736
TWebContainedComponentHelper events 24738
TWebContainedComponentHelper Members 24732
TWebContainedComponentHelper methods 24733
TWebContainedComponentHelper properties 24737
WebParent 24737
TWebContainedContainerComponent class 24738
about TWebContainedContainerComponent class 24738
Create 24742
Destroy 24743

hhhh

GetChildren 24743
GetContainerList 24744
GetDefaultWebComponents 24744
GetParentComponent 24745
GetWebComponents 24745
HasParent 24745
ImplGetDefaultComponentList 24746
IsDefaultField 24746
ParentChanged 24747
ReadState 24747
SetChildOrder 24748
SetDefaultWebComponents 24748
SetParentComponent 24748
SetWebParent 24749
TWebContainedContainerComponent Members 24739
TWebContainedContainerComponent methods 24743
TWebContainedContainerComponent properties 24749
WebContainedComponentHelper 24749
WebContainerComponentHelper 24750
WebParent 24750
TWebContainerComponent class 24751
about TWebContainerComponent class 24751
Create 24754
Destroy 24755
GetChildren 24755
GetDefaultWebComponents 24756
GetWebComponents 24756
ImplGetDefaultComponentList 24756
SetChildOrder 24757
SetDefaultWebComponents 24757
TWebContainerComponent Members 24751
TWebContainerComponent methods 24755
TWebContainerComponent properties 24757
WebContainerComponentHelper 24758
TWebContainerComponentHelper class 24758
about TWebContainerComponentHelper class 24758
Create 24760
DefaultWebComponents 24764
Destroy 24760
GetChildren 24761
GetComponentList 24762
GetDefaultComponentList 24762
GetDefaultWebComponents 24762
OnGetDefaultComponentList 24765
OnSetDefaultComponentList 24765
SetChildOrder 24763
SetDefaultWebComponents 24763
TWebContainerComponentHelper events 24765
TWebContainerComponentHelper Members 24759
TWebContainerComponentHelper methods 24761
TWebContainerComponentHelper properties 24764
Webcomponents 24764
TWebContentOptions class 24766
about TWebContentOptions class 24766
Create 24767
Flags 24768
TWebContentOptions Members 24766
TWebContentOptions properties 24767
TWebContext class 24660
about TWebContext class 24660
Create 24662
Destroy 24663
FindApplicationModule 24664
FindDefaultPageInfo 24664
FindModuleClass 24665
FindModuleName 24665
FindPage 24666
FindPageInfo 24666
FindVariableContainer 24667
GetAdapterRequest 24667
GetAdapterResponse 24668
GetDispatchedPageName 24668
GetEndUser 24668
GetPageCount 24668
GetPageInfo 24669
GetRequestedPageName 24669
GetSession 24669
GetWebModules 24669

GetWebRequest 24670
GetWebResponse 24670
SetAdapterRequest 24670
SetAdapterResponse 24670
SetDispatchedPageName 24671
SetRequestedPageName 24671
TWebContext Members 24661
TWebContext methods 24663
TWebControlGroup class 16872
about TWebControlGroup class 16872
AddElements 16878
CanAddClass 16878
ChangeFieldControls 16879
Content 16879
Create 16877
DefaultWebComponents 16883
Destroy 16877
GetDefaultWebComponents 16880
GetLayoutAttributes 16880
GetSubComponents 16880
ImplAddElements 16881
ImplCanAddClass 16881
ImplContent 16882
ImplGetSubComponents 16882
SetDefaultWebComponents 16883
TWebControlGroup Members 16873
TWebControlGroup methods 16877
TWebControlGroup properties 16883
WebFieldControls 16884
TWebDataDisplay class 16884
about TWebDataDisplay class 16884
Caption 16897
CaptionAttributes 16897
CaptionPosition 16897
Content 16890
ControlContent 16891
Create 16889
Custom 16898
Destroy 16889
EventContent 16891
FormatCaption 16891
GetCaption 16891
GetHTMLControlName 16892
GetHTMLForm 16892
GetLayoutAttributes 16892
GetXmlDisplayName 16893
GetXmlRowSetName 16893
ImplContent 16894
ImplGetHTMLControlName 16894
IsColumn 16895
IsQueryField 16895
LabelContent 16895
LayoutAttributes 16898
SetCaption 16896
SetCaptionAttributes 16896
Style 16899
StyleRule 16899
TabIndex 16899
TWebDataDisplay Members 16885
TWebDataDisplay methods 16889
TWebDataDisplay properties 16896
TWebDataInput class 16900
about TWebDataInput class 16900
EventContent 16905
FieldName 16909
FindAssociatedField 16906
GetCaption 16906
GetFieldName 16906
GetParamName 16906
GetRowSetFieldAttributes 16907
ImplGetFieldName 16907
ImplGetHTMLControlName 16907
ImplGetRowSetFieldAttributes 16907
ImplRestoreDefaults 16908
ParamName 16910
RestoreDefaults 16908
RestoreFieldDefaults 16908
SetCaption 16908

jjjj

SetFieldName 16909
SetParamName 16909
TWebDataInput Members 16901
TWebDataInput methods 16905
TWebDataInput properties 16909
ValidateField 16909
TWebDataModule class 25405
about TWebDataModule class 25405
OnActivate 25409
OnDeactivate 25410
TWebDataModule events 25409
TWebDataModule Members 25406
TWebDataModuleFactory class 25014
about TWebDataModuleFactory class 25014
Create 25016
TWebDataModuleFactory Members 25015
TWebDispatch class 1478
about TWebDispatch class 1478
AssignTo 1480
Create 1480
Destroy 1480
Enabled 1482
GetPathInfo 1481
Mask 1482
MethodType 1483
PathInfo 1483
SetPathInfo 1481
TWebDispatch Members 1478
TWebDispatch methods 1480
TWebDispatch properties 1481
TWebDispatcher class 15147
about TWebDispatcher class 15147
Actions 15152
AfterDispatch 15152
BeforeDispatch 15153
OnException 15153
TWebDispatcher events 15152
TWebDispatcher Members 15148
TWebDispatcher properties 15152
TWebDisplay class 16911
about TWebDisplay class 16911
Changed 16912
ChangedDisplayComponent 16913
Create 16912
DisplayComponent 16914
GetDisplayComponent 16913
Notification 16913
OnChange 16914
Parent 16914
SetDisplayComponent 16913
TWebDisplay events 16914
TWebDisplay Members 16911
TWebDisplay methods 16912
TWebDisplay properties 16914
TWebDisplayReferenceGroup class 16915
about TWebDisplayReferenceGroup class 16915
Create 16919
Destroy 16919
GetWebDisplayComponent 16920
Notification 16920
SetWebDisplayComponent 16921
TWebDisplayReferenceGroup Members 16915
TWebDisplayReferenceGroup methods 16920
TWebDisplayReferenceGroup properties 16921
WebDisplay 16921
WebDisplayComponent 16921
TWebEndUserStub class 1430
about TWebEndUserStub class 1430
Create 1432
CreateActionStub 1432
Get_DisplayName 1433
Get_LoggedIn 1433
Get_LoginFormAction 1433
Get_LogoutAction 1433
TWebEndUserStub Members 1431
TWebEndUserStub methods 1432
TWebForm class 16922
about TWebForm class 16922

kkkk

AddElements 16927
CanAddClass 16928
Content 16928
Create 16926
Custom 16933
Destroy 16926
GetDefaultWebComponents 16929
GetHTMLFormName 16929
GetHTMLFormVarName 16929
GetLayoutAttributes 16930
GetSubComponents 16930
GetVisibleFields 16930
ImplAddElements 16931
ImplCanAddClass 16931
ImplContent 16932
SetDefaultWebComponents 16932
Style 16933
StyleRule 16934
TWebForm Members 16923
TWebForm methods 16927
TWebForm properties 16932
VisibleFields 16934
WebComponents 16934
TWebListInput class 16935
about TWebListInput class 16935
AddElements 16941
ControlContent 16942
Create 16940
DataSet 16944
Destroy 16941
FormatInputs 16942
GetSubComponents 16942
ImplAddElements 16942
Items 16945
ItemsField 16945
Notification 16943
SetDataSet 16943
SetItems 16943
SetValues 16944
TWebListInput Members 16936
TWebListInput methods 16941
TWebListInput properties 16944
Values 16946
ValuesField 16946
TWebModule class 15154
about TWebModule class 15154
Actions 15159
AfterDispatch 15160
BeforeDispatch 15160
Create 15159
OnException 15161
TWebModule events 15160
TWebModule Members 15155
TWebModule properties 15159
TWebModuleFactoryList class 25436
about TWebModuleFactoryList class 25436
AddFactory 25438
AppModuleFactory 25439
Create 25437
Destroy 25438
GetItem 25439
ItemCount 25439
Items 25440
TWebModuleFactoryList Members 25436
TWebModuleFactoryList methods 25438
TWebModuleFactoryList properties 25439
TWebModuleList class 25440
about TWebModuleList class 25440
AddModule 25443
AddModuleClass 25443
AddModuleName 25444
AutoCreateModules 25445
AutoDestroyModules 25445
Create 25442
Destroy 25442
Factories 25448
FindModuleClass 25446
FindModuleName 25446

GetFactory 25446
GetFactoryCount 25447
GetItem 25447
GetItemCount 25447
GetOnModuleAdded 25447
SetOnModuleAdded 25448
TWebModuleList Members 25441
TWebModuleList methods 25443
TWebModuleList properties 25448
TWebNotifyContainer class 25410
about TWebNotifyContainer class 25410
Add 25413
Create 25411
Destroy 25412
FList 25413
GetCount 25413
NotifyWebActivate 25414
NotifyWebDeactivate 25414
TWebNotifyContainer fields 25412
TWebNotifyContainer Members 25411
TWebNotifyContainer methods 25413
TWebPageInfo class 25016
about TWebPageInfo class 25016
Create 25018
TWebPageInfo Members 25017
TWebPageModule class 25414
about TWebPageModule class 25414
DefaultAction 25419
OnActivate 25421
OnAfterDispatchPage 25421
OnBeforeDispatchPage 25421
OnDeactivate 25422
PageProducer 25420
TWebPageModule events 25420
TWebPageModule Members 25415
TWebPageModule properties 25419
TWebPageModuleFactory class 25019
about TWebPageModuleFactory class 25019
Create 25020
TWebPageModuleFactory Members 25020
TWebPageModuleHelper class 25422
about TWebPageModuleHelper class 25422
GetPageProducer 25425
TWebPageModuleHelper Members 25423
TWebPageModuleHelper methods 25425
TWebPathInfo class 1483
about TWebPathInfo class 1483
Create 1484
Destroy 1484
Mask 1485
PathInfo 1485
TWebPathInfo Members 1484
TWebPathInfo properties 1484
TWebRadioGroupInput class 16947
about TWebRadioGroupInput class 16947
AddAttributes 16953
Create 16952
DisplayColumns 16954
DisplayWidth 16955
EventContent 16953
FormatInputs 16953
GetCheckIndex 16954
ImplGetRowSetFieldAttributes 16954
ReadOnly 16955
RestoreFieldDefaults 16954
TWebRadioGroupInput Members 16948
TWebRadioGroupInput methods 16953
TWebRadioGroupInput properties 16954
TWebRequest class 15161
about TWebRequest class 15161
Accept 15174
Authorization 15175
CacheControl 15175
Connection 15176
Content 15176
ContentEncoding 15177
ContentFields 15177
ContentLength 15178

ContentParser 15178
ContentType 15179
ContentVersion 15179
Cookie 15180
CookieFields 15180
Create 15165
Date 15181
DerivedFrom 15181
Destroy 15165
Expires 15182
ExtractContentFields 15166
ExtractCookieFields 15166
ExtractFields 15167
ExtractQueryFields 15167
Files 15182
From 15182
GetDateVariable 15168
GetFieldByName 15168
GetIntegerVariable 15169
GetInternalPathInfo 15169
GetInternalScriptName 15169
GetStringVariable 15169
Host 15183
IfModifiedSince 15183
InternalPathInfo 15184
InternalScriptName 15184
Method 15185
MethodType 15185
PathInfo 15185
PathTranslated 15186
ProtocolVersion 15186
Query 15186
QueryFields 15187
ReadClient 15170
ReadString 15171
Referer 15188
RemoteAddr 15188
RemoteHost 15188
ScriptName 15189
ServerPort 15189
Title 15190
TranslateURI 15171
TWebRequest Members 15162
TWebRequest methods 15165
TWebRequest properties 15173
UpdateMethodType 15172
URL 15190
UserAgent 15191
WriteClient 15172
WriteHeaders 15172
WriteString 15173
TWebRequestFile class 18696
about TWebRequestFile class 18696
Create 18698
Destroy 18698
GetContentType 18699
GetFieldName 18699
GetFileName 18699
GetStream 18700
TWebRequestFile Members 18697
TWebRequestFile methods 18699
TWebRequestFiles class 18700
about TWebRequestFiles class 18700
Add 18703
Add method 18702
Create 18702
Destroy 18702
GetCount 18704
GetItem 18704
TWebRequestFiles Members 18701
TWebRequestFiles methods 18702
TWebRequestFileStream class 18704
about TWebRequestFileStream class 18704
Create 18706
TWebRequestFileStream Members 18705
TWebRequestFileStream methods 18706
Write 18706
TWebRequestHandler class 25448

about TWebRequestHandler class 25448
ActivateWebModules 25453
ActiveCount 25455
AddWebModuleFactory 25453
CacheConnections 25455
Create 25452
DeactivateWebModules 25453
Destroy 25452
HandleException 25454
HandleRequest 25454
InactiveCount 25456
MaxConnections 25457
TWebRequestHandler Members 25449
TWebRequestHandler methods 25452
TWebRequestHandler properties 25455
WebModuleClass 25457
WebModuleFactory 25457
WebModuleFactoryCount 25458
TWebResponse class 15191
about TWebResponse class 15191
AddCustomHeaders 15197
Allow 15207
Content 15207
ContentEncoding 15208
ContentLength 15208
ContentStream 15209
ContentType 15209
ContentVersion 15210
Cookies 15210
Create 15194
CustomHeaders 15211
Date 15211
DerivedFrom 15212
Destroy 15195
Expires 15212
FCustomHeaders 15195
FHTTPRequest 15196
FormatAuthenticate 15197
GetContent 15198
GetCustomHeader 15198
GetDateVariable 15198
GetIntegerVariable 15199
GetLogMessage 15199
GetStatusCode 15199
GetStringVariable 15200
HTTPRequest 15213
LastModified 15213
Location 15214
LogMessage 15214
Realm 15214
ReasonString 15215
SendRedirect 15201
SendResponse 15201
SendStream 15201
Sent 15202
Server 15215
SetContent 15202
SetContentStream 15203
SetCookieField 15203
SetCustomHeader 15203
SetDateVariable 15204
SetIntegerVariable 15204
SetLogMessage 15205
SetStatusCode 15205
SetStringVariable 15205
StatusCode 15215
Title 15217
TWebResponse fields 15195
TWebResponse Members 15192
TWebResponse methods 15196
TWebResponse properties 15206
Version 15217
WWWAuthenticate 15218
TWebSelectOptionsInput class 16956
about TWebSelectOptionsInput class 16956
AddAttributes 16962
Create 16961
DisplayRows 16963

EventContent 16962
FormatInputs 16962
GetSelectIndex 16962
TWebSelectOptionsInput Members 16957
TWebSelectOptionsInput methods 16962
TWebSelectOptionsInput properties 16963
TWebServExp class 25551
about TWebServExp class 25551
ArrayAsComplexContent 25555
BindingType 25555
Create 25553
Destroy 25553
GenerateXMLSchema 25554
GetWebServExp 25554
GetWSDLForInterface 25555
ImportLocations 25556
ImportNames 25556
OnAfterPublishingWSDL 25557
OnBeforePublishingTypes 25557
OnPublishingType 25557
ServiceAddress 25556
TargetNameSpace 25556
TWebServExp events 25557
TWebServExp Members 25551
TWebServExp methods 25554
TWebServExp properties 25555
WSDLElements 25557
TWebSession class 24968
about TWebSession class 24968
Create 24970
Create constructor 24970
GetSessionID 24971
GetTimeoutMinutes 24971
GetValue 24972
SetTimeoutMinutes 24972
SetValue 24972
Terminate 24973
TWebSession Members 24969
TWebSession methods 24971
UpdateResponse 24973
TWebSessionIDsImpl class 25588
about TWebSessionIDsImpl class 25588
Create 25590
Destroy 25590
GetID 25591
GetIDCount 25591
TWebSessionIDsImpl Members 25588
TWebSessionIDsImpl methods 25591
TWebSessionImpl class 25592
about TWebSessionImpl class 25592
Create 25594
Destroy 25594
ExpirationTime 25599
Expired 25599
GetExpirationTime 25595
GetExpired 25595
GetID 25595
GetIsActive 25596
GetItemCount 25596
GetItems 25596
GetStartTime 25596
GetTerminated 25597
GetTimeoutMinutes 25597
GetTouchTime 25597
GetValue 25597
ID 25599
SetTimeoutMinutes 25597
SetValue 25598
StartTime 25599
Terminate 25598
Terminated 25599
TimeoutMinutes 25600
TouchTime 25600
TWebSessionImpl Members 25592
TWebSessionImpl methods 25595
TWebSessionImpl properties 25598
Values 25600
TWebStatus class 16963

about TWebStatus class 16963
Caption 16970
CaptionAttributes 16970
CaptionPosition 16971
ControlContent 16969
Create 16968
DisplayWidth 16971
GetCaption 16969
Identifier 16969
ImplGetHTMLControlName 16969
TWebStatus Members 16964
TWebStatus methods 16968
TWebStatus properties 16970
TWebTextAreaInput class 16971
about TWebTextAreaInput class 16971
AddAttributes 16977
ControlContent 16978
Create 16977
DisplayRows 16979
DisplayWidth 16979
EventContent 16978
ImplGetRowSetFieldAttributes 16978
ReadOnly 16979
RestoreFieldDefaults 16978
TWebTextAreaInput Members 16973
TWebTextAreaInput methods 16977
TWebTextAreaInput properties 16978
Wrap 16980
TWebTextInput class 16980
about TWebTextInput class 16980
AddAttributes 16986
ControlContent 16987
Create 16986
Custom 16987
DisplayWidth 16988
ImplGetRowSetFieldAttributes 16987
ReadOnly 16988
RestoreFieldDefaults 16987
Style 16989
StyleRule 16989
TWebTextInput Members 16982
TWebTextInput methods 16986
TWebTextInput properties 16987
TWebUserItem class 25615
about TWebUserItem class 25615
AccessRights 25618
AccessRightsStrings 25619
Assign 25617
CheckRights 25618
Create 25616
Destroy 25617
Password 25619
TWebUserItem Members 25615
TWebUserItem methods 25617
TWebUserItem properties 25618
UserID 25619
UserName 25620
TWebUserItems class 25620
about TWebUserItems class 25620
Create 25622
FindUserID 25623
FindUserName 25623
GetAttr 25623
GetAttrCount 25624
GetItemAttr 25624
GetOwner 25624
Items 25625
TWebUserItems Members 25620
TWebUserItems methods 25622
TWebUserItems properties 25625
TWebUserList class 25625
about TWebUserList class 25625
OnAfterCheckAccessRights 25630
OnAfterValidateUser 25630
OnBeforeCheckAccessRights 25630
OnBeforeValidateUser 25631
OnUserIDNotFound 25631
OnValidateUserError 25631

qqqq

TWebUserList events 25630
TWebUserList Members 25626
TWebUserList properties 25629
UserItems 25629
TWideCharProperty class 11077
about TWideCharProperty class 11077
GetValue 11080
GetValueW 11081
SetValue 11081
SetValue method 11081
TWideCharProperty Members 11077
TWideCharProperty methods 11080
TWideDataSet class 7526
about TWideDataSet class 7526
PSExecuteStatement 7536
PSExecuteStatement method 7535
PSGetCommandText 7537
PSGetCommandTextW 7537
PSGetKeyFields 7538
PSGetKeyFieldsW 7538
PSGetQuoteChar 7538
PSGetQuoteCharW 7539
PSGetTableName 7539
PSGetTableNameW 7539
PSSetCommandText 7540
PSSetCommandText method 7540
TWideDataSet Members 7526
TWideDataSet methods 7535
TWideMemoField class 7541
about TWideMemoField class 7541
Create 7547
GetAsString 7548
SetAsString 7548
TWideMemoField Members 7541
TWideMemoField methods 7548
TWideMemoField properties 7549
Value 7549
TWideStringField class 7549
about TWideStringField class 7549
CheckTypeSize 7557
CopyData 7557
Create 7555
GetAsString 7557
GetAsVariant 7558
GetAsWideString 7558
GetDataSize 7558
GetValue 7558
SetAsString 7558
SetAsWideString 7559
SetVarValue 7559
TWideStringField Members 7550
TWideStringField methods 7557
TWideStringField properties 7559
Value 7559
TWideStringProperty class 11082
about TWideStringProperty class 11082
AllEqual 11085
GetNameW 11086
GetValue 11086
GetValues 11086
GetValueW 11087
SetValue 11087
TWideStringProperty Members 11082
TWideStringProperty methods 11085
TWideStrings class 25919
about TWideStrings class 25919
Add 25921
Clear 25921
Count 25922
Create 25920
Destroy 25920
IndexOf 25922
IndexOfIgnoreCase 25922
Insert 25923
Strings 25923
TWideStrings Members 25920
TWideStrings methods 25921
TWideStrings properties 25923

TWinCGIRequest class 2583
about TWinCGIRequest class 2583
Create 2586
Destroy 2586
GetFieldByName 2587
GetStringVariable 2587
ReadClient 2588
ReadString 2589
TWinCGIRequest Members 2584
TWinCGIRequest methods 2587
WriteClient 2589
WriteString 2589
TWinCGIResponse class 2590
about TWinCGIResponse class 2590
TWinControl class 6112
about TWinControl class 6112
ActionChange 6127
AddBiDiModeExStyle 6128
AdjustClientRect 6128
AdjustSize 6129
AlignControls 6129
AlignDisabled 6192
ArrangeControl 6130
AssignTo 6130
BevelEdges 6192
BevelInner 6193
BevelKind 6193
BevelOuter 6194
BevelWidth 6194
BorderWidth 6195
Broadcast 6130
Brush 6195
CanAutoSize 6131
CanFocus 6131
CanResize 6132
ChangeScale 6132
ConstrainedResize 6133
ContainsControl 6133
ControlAtPos 6133
ControlCount 6195
Controls 6197
ControlsAligned 6134
Create 6123
CreateDockManager 6134
CreateHandle 6135
CreateParams 6135
CreateParented 6123
CreateParentedControl 6136
CreateSubClass 6136
CreateWindowHandle 6137
CreateWnd 6137
Ctl3D 6198
CustomAlignInsertBefore 6137
CustomAlignPosition 6138
DefaultHandler 6139
DefineProperties 6139
DefWndProc 6199
Destroy 6124
DestroyHandle 6140
DestroyWindowHandle 6140
DestroyWnd 6141
DisableAlign 6141
DoAddDockClient 6142
DockClientCount 6199
DockClients 6201
DockDrop 6142
DockManager 6202
DockOver 6143
DockReplaceDockClient 6143
DockSite 6204
DoDockOver 6143
DoEnter 6144
DoExit 6144
DoFlipChildren 6145
DoKeyDown 6145
DoKeyPress 6146
DoKeyUp 6146
DoRemoveDockClient 6147

DoubleBuffered 6205
DoUnDock 6147
EnableAlign 6148
FDesignSize 6124
FDoubleBuffered 6124
FindChildControl 6148
FindNextControl 6149
FInImeComposition 6125
FixupTabList 6149
FlipChildren 6150
Focused 6150
GetActionLinkClass 6152
GetChildren 6153
GetClientOrigin 6153
GetClientRect 6154
GetControlExtents 6154
GetDeviceContext 6155
GetParentHandle 6157
GetSiteInfo 6158
GetTabOrderList 6158
GetTopParentHandle 6159
Handle 6207
HandleAllocated 6159
HandleNeeded 6160
ImeMode 6207
ImeName 6208
InsertControl 6160
Invalidate 6161
InvalidateDockHostSite 6162
IsControlActivateMsg 6162
IsControlMouseMsg 6162
IsQualifyingSite 6163
KeyDown 6163
KeyPress 6163
KeyUp 6164
MainWndProc 6165
MouseInClient 6209
Notification 6165
NotifyControls 6165
OnAlignInsertBefore 6213
OnAlignPosition 6214
OnDockDrop 6214
OnDockOver 6216
OnEnter 6219
OnExit 6221
OnGetSiteInfo 6223
OnKeyDown 6225
OnKeyPress 6227
OnKeyUp 6228
OnUnDock 6230
Padding 6209
PaintControls 6166
PaintHandler 6166
PaintTo 6167
PaintTo method 6167
PaintWindow 6169
PaletteChanged 6170
ParentBackground 6209
ParentCtl3D 6209
ParentWindow 6210
PreProcessMessage 6170
ReadState 6171
Realign 6171
RecreateWnd 6171
ReloadDockedControl 6172
RemoveControl 6172
RemoveWindowProps 6174
Repaint 6174
ResetIme 6174
ResetImeComposition 6175
ScaleBy 6175
ScaleControls 6176
ScrollBy 6176
SelectFirst 6177
SelectNext 6177
SetBounds 6178
SetChildOrder 6178
SetDesignVisible 6178

SetFocus 6179
SetIme 6181
SetImeCompositionWindow 6181
SetParent 6182
SetParentBackground 6182
SetZOrder 6182
ShowControl 6183
Showing 6210
TabOrder 6211
TabStop 6211
TWinControl events 6213
TWinControl fields 6124
TWinControl Members 6113
TWinControl methods 6125
TWinControl properties 6191
Update 6183
UpdateBounds 6185
UpdateControlOriginalParentSize 6186
UpdateControlState 6186
UpdateRecreatingFlag 6186
UpdateUIState 6186
UseDockManager 6212
VisibleDockClientCount 6212
WindowHandle 6212
WndProc 6187
TWinControlActionLink class 6233
about TWinControlActionLink class 6233
AssignClient 6236
FClient 6236
IsHelpContextLinked 6236
SetHelpContext 6237
TWinControlActionLink fields 6235
TWinControlActionLink Members 6233
TWinControlActionLink methods 6236
TWindowAction class 20779
about TWindowAction class 20779
Create 20784
Form 20786
GetForm 20784
HandlesTarget 20785
Notification 20785
TWindowAction Members 20780
TWindowAction methods 20784
TWindowAction properties 20786
UpdateTarget 20785
TWindowArrange class 20786
about TWindowArrange class 20786
ExecuteTarget 20791
TWindowArrange Members 20787
TWindowArrange methods 20791
TWindowCascade class 20792
about TWindowCascade class 20792
ExecuteTarget 20797
TWindowCascade Members 20793
TWindowCascade methods 20797
TWindowClose class 20798
about TWindowClose class 20798
ExecuteTarget 20803
TWindowClose Members 20798
TWindowClose methods 20803
UpdateTarget 20803
TWindowMinimizeAll class 20804
about TWindowMinimizeAll class 20804
ExecuteTarget 20809
TWindowMinimizeAll Members 20804
TWindowMinimizeAll methods 20809
TWindowTileHorizontal class 20809
about TWindowTileHorizontal class 20809
ExecuteTarget 20815
TWindowTileHorizontal Members 20810
TWindowTileHorizontal methods 20815
TWindowTileVertical class 20815
about TWindowTileVertical class 20815
ExecuteTarget 20820
TWindowTileVertical Members 20816
TWindowTileVertical methods 20820
TWinSocketStream class 18876
about TWinSocketStream class 18876

uuuu

Create 18878
Destroy 18879
Read 18880
Seek 18880
TimeOut 18882
TWinSocketStream Members 18877
TWinSocketStream methods 18879
TWinSocketStream properties 18882
WaitForData 18881
Write 18881
TWordArray class 17380
about TWordArray class 17380
Add 17382
Assign 17383
Create 17382
Find 17383
GetItem 17384
IndexOf 17384
Items 17385
PutItem 17384
TWordArray Members 17380
TWordArray methods 17382
TWordArray properties 17384
TWordField class 7560
about TWordField class 7560
Create 7566
GetDataSize 7568
TWordField Members 7561
TWordField methods 7567
TWorkArea class 5425
about TWorkArea class 5425
Color 5428
Create 5427
GetDisplayName 5427
Rect 5428
SetDisplayName 5427
TWorkArea Members 5425
TWorkArea methods 5427
TWorkArea properties 5428
TWorkAreas class 5429
about TWorkAreas class 5429
Add 5431
Changed 5432
Delete 5432
Insert 5433
Items 5434
TWorkAreas Members 5429
TWorkAreas methods 5431
TWorkAreas properties 5434
Update 5433
TWriter class 3175
about TWriter class 3175
DefineBinaryProperty 3179
DefineProperty 3180
Destroy 3178
FindMethodName 3181
FlushBuffer 3181
OnFindAncestor 3193
OnFindMethodName 3194
Position 3192
RootAncestor 3192
SetRoot 3181
TWriter events 3193
TWriter Members 3176
TWriter methods 3178
TWriter properties 3191
UseQualifiedNames 3193
Write 3181
WriteBinary 3182
WriteBoolean 3182
WriteChar 3182
WriteCollection 3183
WriteComponent 3183
WriteCurrency 3184
WriteDate 3184
WriteDescendent 3184
WriteDouble 3185
WriteFloat 3185

WriteIdent 3186
WriteInteger 3186
WriteInteger method 3186
WriteListBegin 3187
WriteListEnd 3187
WritePrefix 3187
WriteProperties 3188
WriteProperty 3188
WritePropName 3188
WriteRootComponent 3188
WriteSignature 3189
WriteSingle 3189
WriteStr 3189
WriteString 3190
WriteValue 3190
WriteVariant 3190
WriteWideChar 3191
WriteWideString 3191
TWSDLClientNode class 26006
about TWSDLClientNode class 26006
BeforeExecute 26010
Execute 26010, 26011
Execute method 26010
GetMimeBoundary 26011
Notification 26012
SetMimeBoundary 26012
TransportNode 26013
TWSDLClientNode Members 26007
TWSDLClientNode methods 26009
TWSDLClientNode properties 26013
WSDLView 26013
TWSDLDocument class 25799
about TWSDLDocument class 25799
AfterConstruction 25807
Definition 25808
GetDefinition 25807
LoadData 25807
SetDefinition 25807
TWSDLDocument Members 25800
TWSDLDocument methods 25806
TWSDLDocument properties 25808
TWSDLHTMLPublish class 26024
about TWSDLHTMLPublish class 26024
AddInterfaceList 26029
AddPortList 26029
AdminEnabled 26034
Create 26028
Destroy 26028
DispatchEnabled 26030
DispatchMask 26030
DispatchMethodType 26030
DispatchRequest 26031
GetHostScriptBaseURL 26031
GetTargetNamespace 26031
HandleRequest 26032
OnAfterPublishingWSDL 26035
OnBeforePublishingTypes 26036
OnBeforePublishingWSDL 26036
OnPublishingType 26037
PublishOptions 26034
ServiceInfo 26032
TargetNamespace 26035
TWSDLHTMLPublish events 26035
TWSDLHTMLPublish Members 26025
TWSDLHTMLPublish methods 26029
TWSDLHTMLPublish properties 26034
UpdatePortList 26033
WebDispatch 26035
WSILInfo 26033
TWSDLItems class 25951
about TWSDLItems class 25951
CompareName 25962
Create 25960
Create constructor 25959
GetBindingForServicePort 25962
GetBindingOfType 25963
GetBindingOperationNode 25963
GetBindings 25964

wwww

GetBindingType 25964
GetHttpBindingAttribute 25965
GetHttpBindingAttribute method 25965
GetImports 25965
GetLocationForImport 25966
GetMessageNode 25966
GetMessages 25967
GetName 25967
GetOperationNode 25967
GetOperations 25968
GetOperationsForBinding 25968
GetPartNode 25969
GetParts 25969
GetPartsForOperation 25970
GetPortsForService 25970
GetPortTypeNode 25971
GetPortTypes 25971
GetSchemaNode 25972
GetSchemas 25972
GetServiceAndPortOfBinding 25973
GetServiceNode 25973
GetServices 25974
GetSoapAction 25974
GetSoapActionList 25975
GetSoapAddressForServicePort 25975
GetSoapBindingAttribute 25976
GetSoapBindingInputNode 25976
GetSoapBindingIONode 25977
GetSoapBindingOutputNode 25978
GetSoapBodyAttribute 25978
GetSoapBodyInputAttribute 25979
GetSoapBodyNamespace 25980
GetSoapBodyOutputAttribute 25980
GetSoapHeaders 25981
GetSoapInputHeaders 25982
GetSoapOperationAttribute 25982
GetSoapOperationStyle 25983
GetSoapOutputHeaders 25983
GetTargetNamespace 25984
GetTargetNamespacePrefix 25985
GetWSDLDoc 25985
GetWSDLItems 25985
HasTypesNode 25986
IsPortTypeHTTPBound 25986
IterateImports 25986
Load 25987
StreamLoader 25987
TargetNamespace 25988
TWSDLItems Members 25952
TWSDLItems methods 25960
TWSDLItems properties 25987
TWSDLPublish class 26037
about TWSDLPublish class 26037
GetPortTypeEntries 26039
GetPortTypeList 26039
GetTypeSystemsList 26039
GetWSDLForPortType 26040
GetXSDForTypeSystem 26040
OnAfterPublishingWSDL 26041
OnBeforePublishingTypes 26041
OnBeforePublishingWSDL 26041
OnPublishingType 26041
TargetNamespace 26040
TWSDLPublish events 26040
TWSDLPublish Members 26037
TWSDLPublish methods 26039
TWSDLPublish properties 26040
TWSDLView class 26013
about TWSDLView class 26013
Activate 26017
IntfInfo 26017
Operation 26018
Password 26018
Port 26019
PortType 26019
Proxy 26020
Service 26020
SetDesignState 26017

TWSDLView fields 26017
TWSDLView Members 26014
TWSDLView methods 26017
TWSDLView properties 26018
UserName 26020
WSDL 26020
TXMLBroker class 26049
about TXMLBroker class 26049
AddNotify 26055
AfterDispatch 26077
ApplyXMLUpdates 26056
AppServer 26070
AS_ApplyUpdates 26056
AS_FetchParams 26057
AS_GetRecords 26057
BeforeDispatch 26078
CheckInactive 26057
Connected 26070
Create 26054
Destroy 26054
DispatchEnabled 26057
DispatchMask 26058
DispatchMethodType 26058
DispatchRequest 26058
FetchParams 26058
GetAppServer 26059
GetConnected 26059
GetDelta 26059
GetErrorCount 26059
GetErrorResponse 26060
GetErrors 26060
GetHasAppServer 26060
GetNotify 26060
GetNotifyCount 26061
GetProducerName 26061
GetRedirect 26061
GetResponse 26061
GetXMLRecords 26062
GetXMLStream 26062
HasAppServer 26071
HTMLSubmitFormName 26063
MasterRowSetVarName 26063
MaxErrors 26071
MaxRecords 26072
Notification 26064
Notify 26073
NotifyCount 26073
OnGetErrorResponse 26078
OnGetResponse 26079
OnRequestRecords 26080
OnRequestUpdate 26080
Params 26073
ProviderName 26074
ReconcileProducer 26075
RemoteServer 26076
RemoveNotify 26064
RequestRecords 26064
RequestUpdate 26065
RowSetVarName 26066
SendConnectEvent 26067
SetAppServer 26067
SetConnected 26067
SetParams 26067
SetProvider 26068
SetProviderName 26068
SetReconcileProducer 26068
SetRemoteServer 26069
SetWebDispatch 26069
SubmitFormVarName 26069
TXMLBroker events 26077
TXMLBroker Members 26050
TXMLBroker methods 26054
TXMLBroker properties 26069
WebDispatch 26076
TXMLButton class 16989
about TXMLButton class 16989
Create 16993
DefaultCaption 16996

yyyy

Destroy 16994
GetCaption 16994
GetXMLBroker 16995
GetXMLUseParent 16995
Notification 16995
SetXMLBroker 16996
SetXMLUseParent 16996
TXMLButton Members 16990
TXMLButton methods 16994
TXMLButton properties 16996
XMLBroker 16996
XMLData 16997
XMLUseParent 16997
TXMLData class 16998
about TXMLData class 16998
Changed 17000
ChangedXMLBroker 17000
Create 16999
GetXMLBroker 17000
Notification 17001
OnChange 17002
Parent 17001
SetXMLBroker 17001
TXMLData events 17002
TXMLData Members 16998
TXMLData methods 17000
TXMLData properties 17001
XMLBroker 17002
TXMLDataParent class 17003
about TXMLDataParent class 17003
ChangedUseParent 17004
ChangedXMLBroker 17005
GetXMLBroker 17005
ParentXMLBroker 17005
SetUseParent 17005
TXMLDataParent Members 17003
TXMLDataParent methods 17004
TXMLDataParent properties 17005
UseParent 17006
TXMLDataSet class 17006
_AddRef 17009
_Release 17009
about TXMLDataSet class 17006
ChangedDataSetField 17009
ChangedXMLBroker 17010
CompareXMLData 17010
ConnectionChange 17010
Create 17008
CreateDataSet 17010
DataSet 17012
DataSetField 17013
DataSetPath 17013
Destroy 17008
FindDataSet 17010
GetDataSet 17011
GetDataSetField 17011
GetDataSetPath 17011
Parent 17014
QueryInterface 17011
SetDataSetField 17012
SetXMLBroker 17012
TXMLDataSet Members 17006
TXMLDataSet methods 17008
TXMLDataSet properties 17012
WrapperDataChange 17012
TXMLDataSetParent class 17014
about TXMLDataSetParent class 17014
ChangedDataSetField 17017
ChangedUseParent 17017
ChangedXMLBroker 17017
GetDataSetField 17017
GetXMLBroker 17017
ParentDataSetField 17018
ParentXMLBroker 17018
ParentXMLDataSet 17018
SetUseParent 17018
TXMLDataSetParent Members 17015
TXMLDataSetParent methods 17016

TXMLDataSetParent properties 17018
UseParent 17019
TXMLDisplayGroup class 17019
about TXMLDisplayGroup class 17019
Create 17024
CreateDefaultFields 17025
Destroy 17024
EnableValidateFields 17032
FindField 17026
GetDataSet 17026
GetDataSetField 17026
GetEnableValidateFields 17026
GetFieldsList 17027
GetIsMultipleRecordView 17027
GetVisibleFields 17027
GetXMLBroker 17027
GetXMLDataSet 17028
GetXMLDisplayName 17028
GetXMLRowSetName 17028
HasStatusField 17028
ImplAddElements 17029
ImplGetSubComponents 17029
ImplIsMultipleRecordView 17029
IsFieldInUse 17030
Notification 17030
SetDataSetField 17030
SetEnableValidateFields 17031
SetXMLBroker 17031
TXMLDisplayGroup Members 17020
TXMLDisplayGroup methods 17025
TXMLDisplayGroup properties 17032
ValidateDataSet 17031
ValidateFields 17031
VisibleFields 17032
XMLBroker 17033
XMLDataChange 17031
XMLDataSet 17033
XMLDataSetField 17033
TXMLDisplayParent class 17034
about TXMLDisplayParent class 17034
ChangedDisplayComponent 17036
ChangedUseParent 17036
Create 17036
GetDisplayComponent 17037
SetUseParent 17037
TXMLDisplayParent Members 17034
TXMLDisplayParent methods 17036
TXMLDisplayParent properties 17037
UseParent 17037
TXMLDisplayReferenceButton class 17037
about TXMLDisplayReferenceButton class 17037
Create 17042
Destroy 17042
GetDisplayComponent 17043
GetDisplayComponentParent 17043
GetXmlDisplayName 17043
Notification 17044
SetDisplayComponent 17044
SetDisplayComponentParent 17044
TXMLDisplayReferenceButton Members 17039
TXMLDisplayReferenceButton methods 17043
TXMLDisplayReferenceButton properties 17045
XMLComponent 17045
XMLDisplay 17045
XMLUseParent 17045
TXMLDisplayReferenceGroup class 17046
about TXMLDisplayReferenceGroup class 17046
ImplAddElements 17051
TXMLDisplayReferenceGroup Members 17047
TXMLDisplayReferenceGroup methods 17051
TXMLDisplayReferenceGroup properties 17051
XMLComponent 17051
TXMLDocument class 26097
_AddRef 26106
_Release 26106
about TXMLDocument class 26097
Active 26133
AddChild 26107

AddChild method 26107
AfterClose 26144
AfterConstruction 26107
AfterNodeChange 26145
AfterOpen 26145
AssignParseOptions 26108
AsyncLoadState 26134
BeforeClose 26146
BeforeNodeChange 26146
BeforeOpen 26147
CheckActive 26108
CheckAutoSave 26108
CheckDOM 26109
ChildNodes 26135
Create 26103
Create constructor 26103
CreateElement 26109
CreateNode 26109
DefineProperties 26111
Destroy 26104
DoAfterClose 26111
DoAfterOpen 26111
DoBeforeClose 26112
DoBeforeOpen 26112
DocBindingInfo 26135
DocSource 26135
DocumentElement 26135
DOMDocument 26136
DOMImplementation 26137
DOMParseOptions 26137
DOMPersist 26137
DOMVendor 26137
DoNodeChange 26112
Encoding 26138
FileName 26138
GeneratePrefix 26112
GetActive 26113
GetAsyncLoadState 26113
GetChildNodeClass 26113
GetChildNodes 26113
GetDocBinding 26114
GetDocumentElement 26114
GetDocumentNode 26115
GetDocumentObject 26115
GetDOMDocument 26115
GetDOMPersist 26115
GetEncoding 26116
GetFileName 26116
GetModified 26116
GetNodeIndentStr 26117
GetOptions 26117
GetParseOptions 26117
GetPrologNode 26117
GetPrologValue 26118
GetSchemaRef 26118
GetStandAlone 26118
GetVersion 26118
GetXML 26119
InternalSetPrologValue 26119
IsEmptyDoc 26119
LoadData 26120
Loaded 26120
LoadFromFile 26121
LoadFromStream 26121
LoadFromXML 26122
LoadFromXML method 26122
Modified 26139
NewInstance 26122
Node 26140
NodeIndentStr 26140
NSPrefixBase 26141
OnAsyncLoad 26147
Options 26141
ParseOptions 26142
PrefixID 26142
ReadDOMVendor 26123
Refresh 26123
RegisterDocBinding 26124

ReleaseDoc 26124
Resync 26125
SaveToFile 26125
SaveToStream 26125
SaveToUTF8String 26126
SaveToXML 26126
SaveToXML method 26126
SaveToXMLStrings 26127
SchemaRef 26142
SetActive 26127
SetDocumentElement 26127
SetDOMDocument 26128
SetEncoding 26128
SetFileName 26128
SetModified 26129
SetNodeIndentStr 26129
SetOnAsyncLoad 26129
SetOptions 26130
SetParseOptions 26130
SetPrologValue 26130
SetStandAlone 26130
SetVersion 26131
SetXML 26131
SetXMLStrings 26131
StandAlone 26143
TXMLDocument events 26144
TXMLDocument Members 26098
TXMLDocument methods 26104
TXMLDocument properties 26132
Version 26143
WriteDOMVendor 26132
XML 26144
XMLStringsChanging 26132
TXMLNode class 26147
_AddRef 26155
_Release 26155
about TXMLNode class 26147
AddChild 26156
AddChild method 26156
AddHostedNode 26157
AttributeListNotify 26158
AttributeNodes 26189
CheckNotHosted 26158
CheckReadOnly 26159
CheckTextNode 26159
ChildListNotify 26159
ChildNodeClasses 26190
ChildNodes 26190
ClearDocumentRef 26160
CloneNode 26160
Collection 26191
Create 26152
CreateAttributeList 26161
CreateAttributeNode 26161
CreateChildList 26162
CreateChildNode 26162
CreateCollection 26163
CreateHosted 26152
DeclareNamespace 26164
Destroy 26153
DOMElement 26164
DOMNode 26192
DoNodeChange 26165
FindHostedNode 26165
FindNamespaceDecl 26165
FindNamespaceURI 26166
GetAttribute 26166
GetAttributeNodes 26167
GetAttributeNS 26167
GetChildNodeClasses 26168
GetChildNodes 26168
GetChildValue 26169
GetCollection 26169
GetDOMNode 26170
GetHasChildNodes 26170
GetHostedNodes 26171
GetHostNode 26171
GetIsTextElement 26171

GetLocalName 26172
GetNamespaceURI 26172
GetNodeName 26173
GetNodeObject 26174
GetNodeType 26174
GetNodeValue 26174
GetOwnerDocument 26175
GetParentNode 26175
GetPrefix 26176
GetPrefixedName 26176
GetReadOnly 26177
GetText 26177
GetXML 26178
HasAttribute 26178
HasAttribute method 26178
HasChildNode 26179
HasChildNode method 26179
HostedNodes 26192
HostNode 26193
InternalAddChild 26179
NestingLevel 26180
NextSibling 26180
Normalize 26181
OnHostAttrNotify 26194
OnHostChildNotify 26195
OwnerDocument 26193
ParentNode 26194
PreviousSibling 26181
RegisterChildNode 26182
RegisterChildNodes 26182
RemoveHostedNode 26183
Resync 26183
SetAttribute 26183
SetAttributeNodes 26184
SetAttributeNS 26184
SetChildNodes 26185
SetChildValue 26185
SetCollection 26186
SetNodeValue 26186
SetParentNode 26187
SetReadOnly 26187
SetText 26188
TransformNode 26188
TransformNode method 26188
TXMLNode events 26194
TXMLNode Members 26148
TXMLNode methods 26153
TXMLNode properties 26189
TXMLNodeCollection class 26196
about TXMLNodeCollection class 26196
AddItem 26202
AfterConstruction 26202
ChildListNotify 26203
Clear 26204
Count 26209
CreateItemList 26204
Delete 26205
GetCount 26205
GetList 26205
GetNode 26206
InsertInCollection 26206
IsCollectionItem 26207
ItemInterface 26209
ItemNS 26210
ItemTag 26210
List 26210
Remove 26207
SetChildNodes 26208
TXMLNodeCollection Members 26197
TXMLNodeCollection methods 26201
TXMLNodeCollection properties 26209
UpdateCollectionList 26208
TXMLNodeList class 26211
about TXMLNodeList class 26211
Add 26215
BeginUpdate 26215
Clear 26216
Count 26225

Create 26213
DefaultNamespaceURI 26225
Delete 26217
Delete method 26216
Destroy 26214
DoNotify 26217
EndUpdate 26218
FindNode 26218
FindNode method 26218
FindSibling 26219
First 26220
Get 26220
GetCount 26220
GetNode 26221
GetUpdateCount 26221
IndexOf 26222
IndexOf method 26222
Insert 26223
InternalInsert 26223
Last 26223
List 26226
NotificationProc 26227
Owner 26226
Remove 26224
ReplaceNode 26224
TXMLNodeList events 26227
TXMLNodeList Members 26211
TXMLNodeList methods 26214
TXMLNodeList properties 26225
UpdateCount 26226
TXMLParams class 26081
about TXMLParams class 26081
AssignStrings 26084
AssignTo 26085
TXMLParams Members 26082
TXMLParams methods 26084
TXMLSchemaDefs class 25808
about TXMLSchemaDefs class 25808
Add 25814
GetXMLSchemaDef 25814
TXMLSchemaDefs Members 25809
TXMLSchemaDefs methods 25814
TXMLTransform class 26467
about TXMLTransform class 26467
AfterEachRow 26479
AfterEachRowSet 26480
BeforeEachRow 26480
BeforeEachRowSet 26481
Data 26473
DoTransform 26471
EmptyDestinationDocument 26474
GetData 26471
GetResultString 26471
OnTranslate 26481
ResultDocument 26475
ResultString 26475
SourceXml 26476
SourceXmlDocument 26476
SourceXmlFile 26477
Transform 26472
TransformationDocument 26478
TransformationFile 26478
TransformXML 26472
Translate 26473
TXMLTransform events 26479
TXMLTransform Members 26468
TXMLTransform methods 26471
TXMLTransform properties 26473
TXMLTransformClient class 26482
about TXMLTransformClient class 26482
ApplyUpdates 26486
Create 26485
Destroy 26485
GetDataAsXml 26487
ProviderName 26489
RemoteServer 26490
SetParams 26488
SetupAppServer 26489

eeee

TransformApplyUpdates 26490
TransformGetData 26491
TransformSetParams 26491
TXMLTransformClient Members 26482
TXMLTransformClient methods 26486
TXMLTransformClient properties 26489
TXMLTransformProvider class 26492
about TXMLTransformProvider class 26492
AfterApplyUpdates 26502
AfterGetRecords 26503
AfterRowRequest 26504
BeforeApplyUpdates 26504
BeforeGetRecords 26505
BeforeRowRequest 26506
CacheData 26499
Create 26497
Destroy 26497
InternalApplyUpdates 26498
InternalGetRecords 26498
OnDataRequest 26507
TransformRead 26500
TransformWrite 26500
TXMLTransformProvider events 26502
TXMLTransformProvider Members 26493
TXMLTransformProvider methods 26498
TXMLTransformProvider properties 26499
XMLDataFile 26501
TXPColorMap class 23
about TXPColorMap class 23
BtnFrameColor 28
BtnSelectedColor 28
BtnSelectedFont 29
Color 29
DisabledColor 29
DisabledFontColor 29
DisabledFontShadow 30
FontColor 30
FrameBottomRightInner 30
FrameBottomRightOuter 30
FrameTopLeftInner 31
FrameTopLeftOuter 31
HighlightColor 31
HotColor 31
HotFontColor 32
MenuColor 32
OnColorChange 34
SelectedColor 32
SelectedFontColor 33
ShadowColor 33
TXPColorMap events 33
TXPColorMap Members 24
TXPColorMap methods 27
TXPColorMap properties 28
UnusedColor 33
UpdateColors 27
TXSCustomDateTime class 26515
about TXSCustomDateTime class 26515
AsDateTime 26527
AsUTCDateTime 26528
Create 26517
Day 26528
Destroy 26517
FDateTime 26518
FractionalSeconds 26529
GetAsDateTime 26519
GetAsUTCDateTime 26519
GetDay 26519
GetFractionalSeconds 26520
GetHour 26520
GetHourOffset 26520
GetMillisecond 26521
GetMinute 26521
GetMinuteOffset 26521
GetMonth 26521
GetSecond 26522
GetUseZeroMilliseconds 26522
GetYear 26522
Hour 26529

HourOffset 26530
Millisecond 26530
Minute 26531
MinuteOffset 26531
Month 26532
Second 26532
SetAsDateTime 26523
SetAsUTCDateTime 26523
SetDay 26523
SetFractionalSeconds 26524
SetHour 26524
SetHourOffset 26524
SetMillisecond 26524
SetMinute 26525
SetMinuteOffset 26525
SetMonth 26525
SetSecond 26526
SetUseZeroMilliseconds 26526
SetYear 26526
TXSCustomDateTime fields 26518
TXSCustomDateTime Members 26516
TXSCustomDateTime methods 26518
TXSCustomDateTime properties 26527
UseZeroMilliseconds 26533
Year 26533
TXSDate class 26534
about TXSDate class 26534
AsDate 26539
Clone 26536
Create 26535
Day 26539
GetAsDate 26536
GetDay 26537
GetMonth 26537
GetYear 26537
Month 26540
NativeToXS 26537
SetAsDate 26538
SetDay 26538
SetMonth 26538
SetYear 26538
TXSDate Members 26534
TXSDate methods 26536
TXSDate properties 26539
XSToNative 26538
Year 26540
TXSDateTime class 26541
about TXSDateTime class 26541
Clone 26543
CompareDateTimeParam 26543
Millisecond 26544
NativeToXS 26544
TXSDateTime Members 26541
TXSDateTime methods 26543
TXSDateTime properties 26544
XSToNative 26544
TXSDecimal class 26545
about TXSDecimal class 26545
AsBcd 26547
DecimalString 26548
NativeToXS 26547
TXSDecimal Members 26546
TXSDecimal methods 26546
TXSDecimal properties 26547
XSToNative 26547
TXSDuration class 26548
about TXSDuration class 26548
Create 26551
DecimalSecond 26553
GetDecimalSecond 26551
NativeToXS 26552
SetDecimalSecond 26552
TXSDuration Members 26549
TXSDuration methods 26551
TXSDuration properties 26552
XSToNative 26552
TXSHexBinary class 26553
about TXSHexBinary class 26553

gggggg

AsByteArray 26555
HexBinaryString 26556
NativeToXS 26555
TXSHexBinary Members 26554
TXSHexBinary methods 26554
TXSHexBinary properties 26555
XSToNative 26555
TXSLPageProducer class 26605
about TXSLPageProducer class 26605
TXSLPageProducer Members 26606
TXSLPageProducer properties 26613
XMLData 26613
TXSString class 26556
about TXSString class 26556
NativeToXS 26557
TXSString Members 26556
TXSString methods 26557
XSToNative 26558
TXSTime class 26558
about TXSTime class 26558
AsTime 26566
Clone 26561
FractionalSeconds 26566
FractionalSecondString 26567
GetAsTime 26561
GetFractionalSeconds 26561
GetFractionalSecondString 26561
GetHour 26562
GetHourOffset 26562
GetMillisecond 26562
GetMinute 26562
GetMinuteOffset 26563
GetSecond 26563
Hour 26567
HourOffset 26567
Millisecond 26568
Minute 26568
MinuteOffset 26569
NativeToXS 26563
Second 26569
SetAsTime 26563
SetFractionalSeconds 26564
SetHour 26564
SetHourOffset 26564
SetMillisecond 26564
SetMinute 26564
SetMinuteOffset 26565
SetSecond 26565
TXSTime Members 26559
TXSTime methods 26560
TXSTime properties 26565
UseZeroMilliseconds 26569
XSToNative 26565
TXToolDockForm class 559
about TXToolDockForm class 559
Types 23293
Types namespace 23293
Constants 23316
Functions 23295
Structs, Records, Enums 23301
Types 23304
Types.Bounds 23295
Types.Bounds function 23295
Types.CenterPoint 23295
Types.CenterPoint function 23295
Types.DWORD 23304
Types.DWORD type 23304
Types.EqualRect 23296
Types.EqualRect function 23296
Types.EqualsValue 23316
Types.EqualsValue constant 23316
Types.GreaterThanValue 23316
Types.GreaterThanValue constant 23316
Types.IntersectRect 23296
Types.IntersectRect function 23296
Types.IsRectEmpty 23297
Types.IsRectEmpty function 23297
Types.LessThanValue 23317

hhhh

Types.LessThanValue constant 23317
Types.OffsetRect 23297
Types.OffsetRect function 23297
Types.PByte 23305
Types.PByte type 23305
Types.PDouble 23305
Types.PDouble type 23305
Types.PInteger 23306
Types.PInteger type 23306
Types.PLongint 23306
Types.PLongint type 23306
Types.Point 23298
Types.Point function 23298
Types.PPoint 23306
Types.PPoint type 23306
Types.PRect 23307
Types.PRect type 23307
Types.PSize 23307
Types.PSize type 23307
Types.PSmallInt 23308
Types.PSmallInt type 23308
Types.PSmallPoint 23308
Types.PSmallPoint type 23308
Types.PtInRect 23298
Types.PtInRect function 23298
Types.Rect 23299
Types.Rect function 23299
Types.RT_RCDATA 23317
Types.RT_RCDATA constant 23317
Types.SIZE 23308
Types.SIZE type 23308
Types.SmallPoint 23299
Types.SmallPoint function 23299
Types.tagPOINT 23309
Types.tagPOINT type 23309
Types.tagSIZE 23301
Types.tagSIZE record 23301
Types.TBooleanDynArray 23309
Types.TBooleanDynArray type 23309
Types.TByteDynArray 23309
Types.TByteDynArray type 23309
Types.TCardinalDynArray 23310
Types.TCardinalDynArray type 23310
Types.TDoubleDynArray 23310
Types.TDoubleDynArray type 23310
Types.TInt64DynArray 23311
Types.TInt64DynArray type 23311
Types.TIntegerDynArray 23311
Types.TIntegerDynArray type 23311
Types.TLongWordDynArray 23311
Types.TLongWordDynArray type 23311
Types.TPoint 23301
Types.TPoint record 23301
Types.TRect 23302
Types.TRect record 23302
Types.TShortIntDynArray 23312
Types.TShortIntDynArray type 23312
Types.TSingleDynArray 23312
Types.TSingleDynArray type 23312
Types.TSize 23313
Types.TSize type 23313
Types.TSmallIntDynArray 23313
Types.TSmallIntDynArray type 23313
Types.TSmallPoint 23303
Types.TSmallPoint record 23303
Types.TStringDynArray 23313
Types.TStringDynArray type 23313
Types.TValueRelationship 23314
Types.TValueRelationship type 23314
Types.TWideStringDynArray 23314
Types.TWideStringDynArray type 23314
Types.TWordDynArray 23315
Types.TWordDynArray type 23315
Types.UnionRect 23300
Types.UnionRect function 23300
TypeTrans 23319
TypeTrans namespace 23319
Classes 23320

Functions 23324
Variables 23326
TypeTrans.ETypeTransException 23320
TypeTrans.FloatToStrEx 23324
TypeTrans.FloatToStrEx function 23324
TypeTrans.GetEnumValueEx 23324
TypeTrans.GetEnumValueEx function 23324
TypeTrans.SetEnumPropEx 23325
TypeTrans.SetEnumPropEx function 23325
TypeTrans.StrToFloatEx 23325
TypeTrans.StrToFloatEx function 23325
TypeTrans.TTypeTranslator 23320
TypeTrans.TTypeTranslator.Base64ToVar 23322
TypeTrans.TTypeTranslator.CastNativeToSoap 23322
TypeTrans.TTypeTranslator.CastSoapToNative 23323
TypeTrans.TTypeTranslator.CastSoapToVariant 23323
TypeTrans.TTypeTranslator.Create 23321
TypeTrans.TTypeTranslator.Destroy 23321
TypeTrans.TypeTranslator 23326
TypeTrans.TypeTranslator variable 23326
TypInfo 23327
TypInfo namespace 23327
Classes 23331
Constants 23389
Functions 23338
Structs, Records, Enums 23372
Types 23381
Variables 23388
TypInfo.BooleanIdents 23388
TypInfo.BooleanIdents variable 23388
TypInfo.DotSep 23388
TypInfo.DotSep variable 23388
TypInfo.EPropertyConvertError 23331
TypInfo.EPropertyError 23331
TypInfo.FreeAndNilProperties 23339
TypInfo.FreeAndNilProperties function 23339
TypInfo.GetDynArrayProp 23340
TypInfo.GetDynArrayProp function 23340
TypInfo.GetEnumName 23341
TypInfo.GetEnumName function 23341
TypInfo.GetEnumProp 23341
TypInfo.GetEnumProp function 23341
TypInfo.GetEnumValue 23342
TypInfo.GetEnumValue function 23342
TypInfo.GetFloatProp 23343
TypInfo.GetFloatProp function 23343
TypInfo.GetInt64Prop 23343
TypInfo.GetInt64Prop function 23343
TypInfo.GetInterfaceProp 23344
TypInfo.GetInterfaceProp function 23344
TypInfo.GetMethodProp 23345
TypInfo.GetMethodProp function 23345
TypInfo.GetObjectProp 23346
TypInfo.GetObjectProp function 23346
TypInfo.GetObjectPropClass 23347
TypInfo.GetObjectPropClass function 23347
TypInfo.GetOrdProp 23347
TypInfo.GetOrdProp function 23347
TypInfo.GetPropInfo 23348
TypInfo.GetPropInfo function 23348
TypInfo.GetPropInfos 23349
TypInfo.GetPropInfos function 23349
TypInfo.GetPropList 23350
TypInfo.GetPropList function 23350
TypInfo.GetPropValue 23351
TypInfo.GetPropValue function 23351
TypInfo.GetSetElementName 23351
TypInfo.GetSetElementName function 23351
TypInfo.GetSetElementValue 23352
TypInfo.GetSetElementValue function 23352
TypInfo.GetSetProp 23352
TypInfo.GetSetProp function 23352
TypInfo.GetStrProp 23353
TypInfo.GetStrProp function 23353
TypInfo.GetTypeData 23354
TypInfo.GetTypeData function 23354
TypInfo.GetVariantProp 23354
TypInfo.GetVariantProp function 23354

jjjj

TypInfo.GetWideStrProp 23355
TypInfo.GetWideStrProp function 23355
TypInfo.IsPublishedProp 23356
TypInfo.IsPublishedProp function 23356
TypInfo.IsStoredProp 23356
TypInfo.IsStoredProp function 23356
TypInfo.PPropInfo 23381
TypInfo.PPropInfo type 23381
TypInfo.PPropList 23382
TypInfo.PPropList type 23382
TypInfo.PPTypeInfo 23382
TypInfo.PPTypeInfo type 23382
TypInfo.PropIsType 23357
TypInfo.PropIsType function 23357
TypInfo.PropType 23358
TypInfo.PropType function 23358
TypInfo.PTypeData 23383
TypInfo.PTypeData type 23383
TypInfo.PTypeInfo 23383
TypInfo.PTypeInfo type 23383
TypInfo.SetDynArrayProp 23358
TypInfo.SetDynArrayProp function 23358
TypInfo.SetEnumProp 23359
TypInfo.SetEnumProp function 23359
TypInfo.SetFloatProp 23360
TypInfo.SetFloatProp function 23360
TypInfo.SetInt64Prop 23361
TypInfo.SetInt64Prop function 23361
TypInfo.SetInterfaceProp 23362
TypInfo.SetInterfaceProp function 23362
TypInfo.SetMethodProp 23362
TypInfo.SetMethodProp function 23362
TypInfo.SetObjectProp 23363
TypInfo.SetObjectProp function 23363
TypInfo.SetOrdProp 23364
TypInfo.SetOrdProp function 23364
TypInfo.SetPropValue 23365
TypInfo.SetPropValue function 23365
TypInfo.SetSetProp 23366
TypInfo.SetSetProp function 23366
TypInfo.SetStrProp 23367
TypInfo.SetStrProp function 23367
TypInfo.SetToString 23367
TypInfo.SetToString function 23367
TypInfo.SetVariantProp 23368
TypInfo.SetVariantProp function 23368
TypInfo.SetWideStrProp 23369
TypInfo.SetWideStrProp function 23369
TypInfo.ShortStringBase 23384
TypInfo.ShortStringBase type 23384
TypInfo.SortPropList 23370
TypInfo.SortPropList function 23370
TypInfo.StringToSet 23370
TypInfo.StringToSet function 23370
TypInfo.TFloatType 23372
TypInfo.TFloatType enumeration 23372
TypInfo.TIntfFlag 23373
TypInfo.TIntfFlag enumeration 23373
TypInfo.TIntfFlags 23384
TypInfo.TIntfFlags type 23384
TypInfo.TIntfFlagsBase 23385
TypInfo.TIntfFlagsBase type 23385
TypInfo.tkAny 23389
TypInfo.tkAny constant 23389
TypInfo.tkMethods 23389
TypInfo.tkMethods constant 23389
TypInfo.tkProperties 23389
TypInfo.tkProperties constant 23389
TypInfo.TMethodKind 23373
TypInfo.TMethodKind enumeration 23373
TypInfo.TOrdType 23374
TypInfo.TOrdType enumeration 23374
TypInfo.TParamFlag 23375
TypInfo.TParamFlag enumeration 23375
TypInfo.TParamFlags 23386
TypInfo.TParamFlags type 23386
TypInfo.TParamFlagsBase 23386
TypInfo.TParamFlagsBase type 23386

kkkk

TypInfo.TPropData 23376
TypInfo.TPropData record 23376
TypInfo.TPropInfo 23376
TypInfo.TPropInfo record 23376
TypInfo.TPropInfoProc 23386
TypInfo.TPropInfoProc type 23386
TypInfo.TPropList 23387
TypInfo.TPropList type 23387
TypInfo.TPublishableVariantType 23332
TypInfo.TPublishableVariantType.GetInstance 23336
TypInfo.TPublishableVariantType.GetProperty 23336
TypInfo.TPublishableVariantType.SetProperty 23337
TypInfo.TTypeData 23377
TypInfo.TTypeData record 23377
TypInfo.TTypeInfo 23379
TypInfo.TTypeInfo record 23379
TypInfo.TTypeKind 23379
TypInfo.TTypeKind enumeration 23379
TypInfo.TTypeKinds 23387
TypInfo.TTypeKinds type 23387

U

uaAbort enumeration member 7621
uaApplied enumeration member 7621
uaFail enumeration member 7621
uaRetry enumeration member 7621
uaSkip enumeration member 7621
UDDIHelper 23391
UDDIHelper namespace 23391
    Functions 23392
UDDIHelper.GetBindingkeyAccessPoint 23392
UDDIHelper.GetBindingkeyAccessPoint function 23392
udHorizontal enumeration member 5472
udLeft enumeration member 5471
udRight enumeration member 5471
udVertical enumeration member 5472
ukDelete enumeration member 7622
ukInsert enumeration member 7622
ukModify enumeration member 7622
updDown enumeration member 5473
updNone enumeration member 5473
updUp enumeration member 5473
upWhereAll enumeration member 7623
upWhereChanged enumeration member 7623
upWhereKeyOnly enumeration member 7623
usDeleted enumeration member 7623
UsersColumns class 2192
    about UsersColumns class 2192
    UserName 2193
    UsersColumns fields 2193
    UsersColumns Members 2193
usInserted enumeration member 7623
usModified enumeration member 7623
usUnmodified enumeration member 7623

V

vaBinary enumeration member 3279
vaCollection enumeration member 3279
vaCurrency enumeration member 3279
vaDate enumeration member 3279
vaDouble enumeration member 3279
vaExtended enumeration member 3279
vaFalse enumeration member 3279
vaIdent enumeration member 3279
vaInt16 enumeration member 3279
vaInt32 enumeration member 3279
vaInt64 enumeration member 3279
vaInt8 enumeration member 3279
ValEdit 23393
ValEdit namespace 23393
    Classes 23394
    Structs, Records, Enums 23509
    Types 23511
ValEdit.TDisplayOption 23509
ValEdit.TDisplayOption enumeration 23509
ValEdit.TDisplayOptions 23511
ValEdit.TDisplayOptions type 23511
ValEdit.TGetPickListEvent 23511

ValEdit.TGetPickListEvent type 23511
ValEdit.TItemProp 23394
ValEdit.TItemProp.AssignTo 23396
ValEdit.TItemProp.Create 23395
ValEdit.TItemProp.Destroy 23396
ValEdit.TItemProp.EditMask 23398
ValEdit.TItemProp.EditStyle 23398
ValEdit.TItemProp.HasPickList 23397
ValEdit.TItemProp.KeyDesc 23399
ValEdit.TItemProp.MaxLength 23399
ValEdit.TItemProp.PickList 23400
ValEdit.TItemProp.ReadOnly 23400
ValEdit.TItemProp.UpdateEdit 23397
ValEdit.TItemProps 23512
ValEdit.TItemProps type 23512
ValEdit.TKeyOption 23510
ValEdit.TKeyOption enumeration 23510
ValEdit.TKeyOptions 23512
ValEdit.TKeyOptions type 23512
ValEdit.TOnValidateEvent 23512
ValEdit.TOnValidateEvent type 23512
ValEdit.TValueListEditor 23401
ValEdit.TValueListEditor.AdjustColWidths 23416
ValEdit.TValueListEditor.AdjustRowCount 23416
ValEdit.TValueListEditor.Align 23428
ValEdit.TValueListEditor.Anchors 23432
ValEdit.TValueListEditor.BiDiMode 23432
ValEdit.TValueListEditor.BorderStyle 23433
ValEdit.TValueListEditor.CanEditModify 23417
ValEdit.TValueListEditor.Cells 23433
ValEdit.TValueListEditor.ColCount 23434
ValEdit.TValueListEditor.Color 23434
ValEdit.TValueListEditor.ColWidthsChanged 23417
ValEdit.TValueListEditor.Constraints 23435
ValEdit.TValueListEditor.Create 23415
ValEdit.TValueListEditor.CreateEditor 23417
ValEdit.TValueListEditor.CreateWnd 23418
ValEdit.TValueListEditor.Ctl3D 23435
ValEdit.TValueListEditor.DefaultColWidth 23436
ValEdit.TValueListEditor.DefaultDrawing 23437
ValEdit.TValueListEditor.DefaultRowHeight 23437
ValEdit.TValueListEditor.DeleteRow 23418
ValEdit.TValueListEditor.Destroy 23415
ValEdit.TValueListEditor.DisplayOptions 23438
ValEdit.TValueListEditor.DoExit 23419
ValEdit.TValueListEditor.DoOnValidate 23419
ValEdit.TValueListEditor.DragCursor 23438
ValEdit.TValueListEditor.DragKind 23438
ValEdit.TValueListEditor.DragMode 23439
ValEdit.TValueListEditor.DrawCell 23419
ValEdit.TValueListEditor.DropDownRows 23440
ValEdit.TValueListEditor.EditList 23441
ValEdit.TValueListEditor.EditListGetItems 23420
ValEdit.TValueListEditor.Enabled 23441
ValEdit.TValueListEditor.FindRow 23420
ValEdit.TValueListEditor.FixedColor 23442
ValEdit.TValueListEditor.FixedCols 23442
ValEdit.TValueListEditor.Font 23442
ValEdit.TValueListEditor.GetCell 23420
ValEdit.TValueListEditor.GetColCount 23421
ValEdit.TValueListEditor.GetEditLimit 23421
ValEdit.TValueListEditor.GetEditMask 23421
ValEdit.TValueListEditor.GetEditStyle 23421
ValEdit.TValueListEditor.GetEditText 23422
ValEdit.TValueListEditor.GetPickList 23422
ValEdit.TValueListEditor.GetRowCount 23422
ValEdit.TValueListEditor.GridLineWidth 23444
ValEdit.TValueListEditor.InsertRow 23422
ValEdit.TValueListEditor.IsEmptyRow 23423
ValEdit.TValueListEditor.ItemProps 23445
ValEdit.TValueListEditor.KeyDown 23423
ValEdit.TValueListEditor.KeyOptions 23445
ValEdit.TValueListEditor.Keys 23445
ValEdit.TValueListEditor.MouseDown 23424
ValEdit.TValueListEditor.OnClick 23461
ValEdit.TValueListEditor.OnContextPopup 23463
ValEdit.TValueListEditor.OnDblClick 23463
ValEdit.TValueListEditor.OnDragDrop 23464

ValEdit.TValueListEditor.OnDragOver 23466
ValEdit.TValueListEditor.OnDrawCell 23468
ValEdit.TValueListEditor.OnEditButtonClick 23470
ValEdit.TValueListEditor.OnEndDock 23470
ValEdit.TValueListEditor.OnEndDrag 23471
ValEdit.TValueListEditor.OnEnter 23472
ValEdit.TValueListEditor.OnExit 23474
ValEdit.TValueListEditor.OnGetEditMask 23476
ValEdit.TValueListEditor.OnGetEditText 23476
ValEdit.TValueListEditor.OnGetPickList 23477
ValEdit.TValueListEditor.OnKeyDown 23478
ValEdit.TValueListEditor.OnKeyPress 23480
ValEdit.TValueListEditor.OnKeyUp 23481
ValEdit.TValueListEditor.OnMouseActivate 23483
ValEdit.TValueListEditor.OnMouseDown 23484
ValEdit.TValueListEditor.OnMouseEnter 23486
ValEdit.TValueListEditor.OnMouseLeave 23487
ValEdit.TValueListEditor.OnMouseMove 23487
ValEdit.TValueListEditor.OnMouseUp 23489
ValEdit.TValueListEditor.OnMouseWheelDown 23492
ValEdit.TValueListEditor.OnMouseWheelUp 23492
ValEdit.TValueListEditor.OnRowMoved 23492
ValEdit.TValueListEditor.OnSelectCell 23493
ValEdit.TValueListEditor.OnSetEditText 23494
ValEdit.TValueListEditor.OnStartDock 23495
ValEdit.TValueListEditor.OnStartDrag 23495
ValEdit.TValueListEditor.OnStringsChange 23496
ValEdit.TValueListEditor.OnStringsChanging 23496
ValEdit.TValueListEditor.OnTopLeftChanged 23497
ValEdit.TValueListEditor.OnValidate 23497
ValEdit.TValueListEditor.Options 23446
ValEdit.TValueListEditor.ParentBiDiMode 23446
ValEdit.TValueListEditor.ParentColor 23447
ValEdit.TValueListEditor.ParentCtl3D 23447
ValEdit.TValueListEditor.ParentFont 23448
ValEdit.TValueListEditor.ParentShowHint 23448
ValEdit.TValueListEditor.PopupMenu 23449
ValEdit.TValueListEditor.Refresh 23424
ValEdit.TValueListEditor.Resize 23425
ValEdit.TValueListEditor.RestoreCurrentRow 23425
ValEdit.TValueListEditor.RowCount 23453
ValEdit.TValueListEditor.RowMoved 23425
ValEdit.TValueListEditor.ScrollBars 23454
ValEdit.TValueListEditor.SelectCell 23426
ValEdit.TValueListEditor.SetCell 23426
ValEdit.TValueListEditor.SetEditText 23426
ValEdit.TValueListEditor.SetStrings 23426
ValEdit.TValueListEditor.ShowHint 23454
ValEdit.TValueListEditor.Strings 23456
ValEdit.TValueListEditor.StringsChanging 23427
ValEdit.TValueListEditor.TabOrder 23456
ValEdit.TValueListEditor.TitleCaptions 23457
ValEdit.TValueListEditor.TitleCaptionsStored 23427
ValEdit.TValueListEditor.Values 23457
ValEdit.TValueListEditor.Visible 23458
ValEdit.TValueListEditor.VisibleColCount 23459
ValEdit.TValueListEditor.VisibleRowCount 23459
ValEdit.TValueListStrings 23498
ValEdit.TValueListStrings.Assign 23502
ValEdit.TValueListStrings.Changed 23504
ValEdit.TValueListStrings.Changing 23505
ValEdit.TValueListStrings.Clear 23505
ValEdit.TValueListStrings.Create 23501
ValEdit.TValueListStrings.CustomSort 23505
ValEdit.TValueListStrings.Delete 23506
ValEdit.TValueListStrings.Destroy 23501
ValEdit.TValueListStrings.Exchange 23506
ValEdit.TValueListStrings.FindItemProp 23507
ValEdit.TValueListStrings.InsertItem 23507
ValEdit.TValueListStrings.ItemProps 23508
ValEdit.TValueListStrings.KeyIsValid 23507
ValEdit.TValueListStrings.Put 23507
vaList enumeration member 3279
vaLString enumeration member 3279
vaNil enumeration member 3279
vaNull enumeration member 3279
VarCmplx 23513
VarCmplx namespace 23513

Functions 23515
Variables 23559
VarCmplx.ComplexNumberDefuzzAtZero 23559
VarCmplx.ComplexNumberDefuzzAtZero variable 23559
VarCmplx.ComplexNumberSymbol 23559
VarCmplx.ComplexNumberSymbol variable 23559
VarCmplx.ComplexNumberSymbolBeforeImaginary 23560
VarCmplx.ComplexNumberSymbolBeforeImaginary variable 23560
VarCmplx.VarAsComplex 23516
VarCmplx.VarAsComplex function 23516
VarCmplx.VarComplex 23516
VarCmplx.VarComplex function 23516
VarCmplx.VarComplexAbs 23517
VarCmplx.VarComplexAbs function 23517
VarCmplx.VarComplexAbsSqr 23518
VarCmplx.VarComplexAbsSqr function 23518
VarCmplx.VarComplexAngle 23518
VarCmplx.VarComplexAngle function 23518
VarCmplx.VarComplexArcCos 23519
VarCmplx.VarComplexArcCos function 23519
VarCmplx.VarComplexArcCosH 23520
VarCmplx.VarComplexArcCosH function 23520
VarCmplx.VarComplexArcCot 23521
VarCmplx.VarComplexArcCot function 23521
VarCmplx.VarComplexArcCotH 23522
VarCmplx.VarComplexArcCotH function 23522
VarCmplx.VarComplexArcCsc 23523
VarCmplx.VarComplexArcCsc function 23523
VarCmplx.VarComplexArcCscH 23524
VarCmplx.VarComplexArcCscH function 23524
VarCmplx.VarComplexArcSec 23525
VarCmplx.VarComplexArcSec function 23525
VarCmplx.VarComplexArcSecH 23527
VarCmplx.VarComplexArcSecH function 23527
VarCmplx.VarComplexArcSin 23528
VarCmplx.VarComplexArcSin function 23528
VarCmplx.VarComplexArcSinH 23529
VarCmplx.VarComplexArcSinH function 23529
VarCmplx.VarComplexArcTan 23530
VarCmplx.VarComplexArcTan function 23530
VarCmplx.VarComplexArcTanH 23531
VarCmplx.VarComplexArcTanH function 23531
VarCmplx.VarComplexConjugate 23532
VarCmplx.VarComplexConjugate function 23532
VarCmplx.VarComplexCos 23533
VarCmplx.VarComplexCos function 23533
VarCmplx.VarComplexCosH 23534
VarCmplx.VarComplexCosH function 23534
VarCmplx.VarComplexCot 23535
VarCmplx.VarComplexCot function 23535
VarCmplx.VarComplexCotH 23536
VarCmplx.VarComplexCotH function 23536
VarCmplx.VarComplexCreate 23537
VarCmplx.VarComplexCreate function 23537
VarCmplx.VarComplexCsc 23538
VarCmplx.VarComplexCsc function 23538
VarCmplx.VarComplexCscH 23539
VarCmplx.VarComplexCscH function 23539
VarCmplx.VarComplexExp 23540
VarCmplx.VarComplexExp function 23540
VarCmplx.VarComplexFromPolar 23541
VarCmplx.VarComplexFromPolar function 23541
VarCmplx.VarComplexInverse 23541
VarCmplx.VarComplexInverse function 23541
VarCmplx.VarComplexLn 23542
VarCmplx.VarComplexLn function 23542
VarCmplx.VarComplexLog10 23543
VarCmplx.VarComplexLog10 function 23543
VarCmplx.VarComplexLog2 23543
VarCmplx.VarComplexLog2 function 23543
VarCmplx.VarComplexLogN 23544
VarCmplx.VarComplexLogN function 23544
VarCmplx.VarComplexPower 23545
VarCmplx.VarComplexPower function 23545
VarCmplx.VarComplexSec 23545
VarCmplx.VarComplexSec function 23545
VarCmplx.VarComplexSecH 23546
VarCmplx.VarComplexSecH function 23546

VarCmplx.VarComplexSign 23548
VarCmplx.VarComplexSign function 23548
VarCmplx.VarComplexSimplify 23548
VarCmplx.VarComplexSimplify function 23548
VarCmplx.VarComplexSin 23548
VarCmplx.VarComplexSin function 23548
VarCmplx.VarComplexSinH 23550
VarCmplx.VarComplexSinH function 23550
VarCmplx.VarComplexSqr 23551
VarCmplx.VarComplexSqr function 23551
VarCmplx.VarComplexSqrt 23551
VarCmplx.VarComplexSqrt function 23551
VarCmplx.VarComplexTan 23552
VarCmplx.VarComplexTan function 23552
VarCmplx.VarComplexTanH 23553
VarCmplx.VarComplexTanH function 23553
VarCmplx.VarComplexTimesImaginary 23554
VarCmplx.VarComplexTimesImaginary function 23554
VarCmplx.VarComplexTimesNegI 23555
VarCmplx.VarComplexTimesNegI function 23555
VarCmplx.VarComplexTimesPosI 23555
VarCmplx.VarComplexTimesPosI function 23555
VarCmplx.VarComplexTimesReal 23556
VarCmplx.VarComplexTimesReal function 23556
VarCmplx.VarComplexToPolar 23556
VarCmplx.VarComplexToPolar function 23556
VarCmplx.VarIsComplex 23557
VarCmplx.VarIsComplex function 23557
VarConv 23561
VarConv namespace 23561
Functions 23562
VarConv.VarAsConvert 23562
VarConv.VarAsConvert function 23562
VarConv.VarConvert 23563
VarConv.VarConvert function 23563
VarConv.VarConvertCreate 23564
VarConv.VarConvertCreate function 23564
VarConv.VarIsConvert 23565
VarConv.VarIsConvert function 23565
Variant class 22307
- 22343, 22344
- operator 22343
! 22344
!= 22345
!= operator 22344
% 22346
% operator 22345
%= 22346
& 22347
& operator 22346
&= 22347
* 22348
* operator 22347
*= 22348
/ 22349
/ operator 22348
/= 22349
^ 22350
^ operator 22349
^= 22350
__int64 22350
| 22351
| operator 22351
|= 22351
~Variant 22311
+ 22352
+ operator 22352
+= 22352
< 22353
< operator 22353
<< 22354
<< operator 22353
<<= 22354
<= 22355
<= operator 22355
= 22359
-= 22361
= operator 22355

== 22361
== operator 22361
> 22362
> operator 22362
>= 22363
>= operator 22363
>> 22364
>> operator 22363
>>= 22365
about Variant class 22307
AnsiString 22365
ArrayDimCount 22317
ArrayHighBound 22317
ArrayLock 22317
ArrayLowBound 22318
ArrayRedim 22318
ArrayUnlock 22318
Assign 22318
AsType 22319
bool 22365
ChangeType 22319
char 22365
class AutoCmd 22375
Clear 22319
CreateObject 22320
Currency 22366
CURRENCY 22366
Currency* 22366
DECIMAL 22366
double 22367
double* 22367
Empty 22320
Exec 22320
float 22367
float* 22368
GetActiveObject 22321
GetBaseVariant 22321
GetElement 22322
GetElement method 22322
GetError 22322
IDispatch* 22368
IInterface* 22368
int 22368
int* 22369
IsArray 22323
IsEmpty 22323
IsNull 22323
IUnknown* 22369
long 22369
long* 22369
LPSAFEARRAY 22369
NoParam 22324
OleFunction 22326
OleFunction method 22324
OleProcedure 22332
OleProcedure method 22330
OlePropertyGet 22337
OlePropertyGet method 22336
OlePropertySet 22338
OlePropertySet method 22338
PutElement 22340
PutElement method 22340
Sanitized 22340
SetError 22341
short 22370
short* 22370
signed char 22370
signed char* 22370
TDateTime 22371
TDateTime* 22371
TVariant 22371
TVariant* 22371
Type 22341
unsigned char 22372
unsigned char* 22372
unsigned int 22372
unsigned int* 22372
unsigned long 22373

qqqq

unsigned long* 22373
unsigned short 22373
unsigned short* 22373
Variant 22314
VARIANT 22373
Variant constructor 22312
Variant friends 22375
Variant Members 22309
Variant methods 22316
Variant operators 22341
VARIANT* 22374
wchar_t* 22374
wchar_t** 22374
WideString 22374
VariantError class 22375
= 22377
about VariantError class 22375
m_scode 22376
SCODE 22377
VariantError 22376
VariantError data members 22376
VariantError Members 22375
VariantError operators 22376
Variants 23567
Variants namespace 23567
Classes 23572
Functions 23617
Interfaces 23614
Structs, Records, Enums 23671
Types 23674
Variables 23676
Variants._DispInvoke 23619
Variants._DispInvoke function 23619
Variants._OleVarFromInt 23620
Variants._OleVarFromInt function 23620
Variants._OleVarFromInt64 23620
Variants._OleVarFromInt64 function 23620
Variants._OleVarFromLStr 23620
Variants._OleVarFromLStr function 23620
Variants._OleVarFromPStr 23621
Variants._OleVarFromPStr function 23621
Variants._OleVarFromVar 23621
Variants._OleVarFromVar function 23621
Variants._VarAdd 23621
Variants._VarAdd function 23621
Variants._VarAddRef 23622
Variants._VarAddRef function 23622
Variants._VarAnd 23622
Variants._VarAnd function 23622
Variants._VarArrayGet 23622
Variants._VarArrayGet function 23622
Variants._VarArrayPut 23623
Variants._VarArrayPut function 23623
Variants._VarArrayRedim 23623
Variants._VarArrayRedim function 23623
Variants._VarCast 23623
Variants._VarCast function 23623
Variants._VarCastOle 23624
Variants._VarCastOle function 23624
Variants._VarClear 23624
Variants._VarClear function 23624
Variants._VarClr 23624
Variants._VarClr function 23624
Variants._VarCmpEQ 23625
Variants._VarCmpEQ function 23625
Variants._VarCmpGE 23625
Variants._VarCmpGE function 23625
Variants._VarCmpGT 23625
Variants._VarCmpGT function 23625
Variants._VarCmpLE 23626
Variants._VarCmpLE function 23626
Variants._VarCmpLT 23626
Variants._VarCmpLT function 23626
Variants._VarCmpNE 23626
Variants._VarCmpNE function 23626
Variants._VarCopy 23627
Variants._VarCopy function 23627
Variants._VarCopyNoInd 23627

Variants._VarCopyNoInd function 23627
Variants._VarDiv 23627
Variants._VarDiv function 23627
Variants._VarFromBool 23628
Variants._VarFromBool function 23628
Variants._VarFromCurr 23628
Variants._VarFromCurr function 23628
Variants._VarFromDisp 23628
Variants._VarFromDisp function 23628
Variants._VarFromDynArray 23629
Variants._VarFromDynArray function 23629
Variants._VarFromInt 23629
Variants._VarFromInt function 23629
Variants._VarFromInt64 23629
Variants._VarFromInt64 function 23629
Variants._VarFromIntf 23630
Variants._VarFromIntf function 23630
Variants._VarFromLStr 23630
Variants._VarFromLStr function 23630
Variants._VarFromPStr 23630
Variants._VarFromPStr function 23630
Variants._VarFromReal 23631
Variants._VarFromReal function 23631
Variants._VarFromTDateTime 23631
Variants._VarFromTDateTime function 23631
Variants._VarFromWStr 23631
Variants._VarFromWStr function 23631
Variants._VarMod 23632
Variants._VarMod function 23632
Variants._VarMul 23632
Variants._VarMul function 23632
Variants._VarNeg 23632
Variants._VarNeg function 23632
Variants._VarNot 23633
Variants._VarNot function 23633
Variants._VarOr 23633
Variants._VarOr function 23633
Variants._VarRDiv 23633
Variants._VarRDiv function 23633
Variants._VarShl 23634
Variants._VarShl function 23634
Variants._VarShr 23634
Variants._VarShr function 23634
Variants._VarSub 23634
Variants._VarSub function 23634
Variants._VarToBool 23635
Variants._VarToBool function 23635
Variants._VarToCurrency 23635
Variants._VarToCurrency function 23635
Variants._VarToDisp 23635
Variants._VarToDisp function 23635
Variants._VarToDynArray 23636
Variants._VarToDynArray function 23636
Variants._VarToInt64 23636
Variants._VarToInt64 function 23636
Variants._VarToInteger 23636
Variants._VarToInteger function 23636
Variants._VarToIntf 23637
Variants._VarToIntf function 23637
Variants._VarToLStr 23637
Variants._VarToLStr function 23637
Variants._VarToPStr 23637
Variants._VarToPStr function 23637
Variants._VarToReal 23638
Variants._VarToReal function 23638
Variants._VarToWStr 23638
Variants._VarToWStr function 23638
Variants._VarXor 23638
Variants._VarXor function 23638
Variants.BooleanToStringRule 23676
Variants.BooleanToStringRule variable 23676
Variants.ChangeAnyProc 23677
Variants.ChangeAnyProc variable 23677
Variants.ClearAnyProc 23677
Variants.ClearAnyProc variable 23677
Variants.DynArrayFromVariant 23639
Variants.DynArrayFromVariant function 23639
Variants.DynArrayToVariant 23639

ssssss

Variants.DynArrayToVariant function 23639
Variants.EmptyParam 23677
Variants.EmptyParam variable 23677
Variants.EVariantArrayCreateError 23572
Variants.EVariantArrayLockedError 23573
Variants.EVariantBadIndexError 23574
Variants.EVariantBadVarTypeError 23574
Variants.EVariantDispatchError 23575
Variants.EVariantInvalidArgError 23575
Variants.EVariantInvalidNullOpError 23576
Variants.EVariantInvalidOpError 23577
Variants.EVariantNotAnArrayError 23577
Variants.EVariantNotImplError 23578
Variants.EVariantOutOfMemoryError 23578
Variants.EVariantOverflowError 23579
Variants.EVariantRangeCheckError 23580
Variants.EVariantTypeCastError 23580
Variants.EVariantUnexpectedError 23581
Variants.FindCustomVariantType 23640
Variants.FindCustomVariantType function 23640
Variants.FindVarData 23640
Variants.FindVarData function 23640
Variants.HandleConversionException 23641
Variants.HandleConversionException function 23641
Variants.IVarInstanceReference 23614
Variants.IVarInstanceReference.GetInstance 23615
Variants.IVarInvokeable 23615
Variants.IVarInvokeable.DoFunction 23616
Variants.IVarInvokeable.DoProcedure 23616
Variants.IVarInvokeable.GetProperty 23616
Variants.IVarInvokeable.SetProperty 23616
Variants.Null 23641
Variants.Null function 23641
Variants.NullAsStringValue 23678
Variants.NullAsStringValue variable 23678
Variants.NullEqualityRule 23678
Variants.NullEqualityRule variable 23678
Variants.NullMagnitudeRule 23679
Variants.NullMagnitudeRule variable 23679
Variants.NullStrictConvert 23679
Variants.NullStrictConvert variable 23679
Variants.OleVariantInt64AsDouble 23680
Variants.OleVariantInt64AsDouble variable 23680
Variants.PackVarCreation 23680
Variants.PackVarCreation variable 23680
Variants.RefAnyProc 23681
Variants.RefAnyProc variable 23681
Variants.TAnyProc 23674
Variants.TAnyProc type 23674
Variants.TBooleanToStringRule 23671
Variants.TBooleanToStringRule enumeration 23671
Variants.TCustomVariantType 23581
Variants.TCustomVariantType._AddRef 23586
Variants.TCustomVariantType._Release 23587
Variants.TCustomVariantType.BinaryOp 23587
Variants.TCustomVariantType.Cast 23588
Variants.TCustomVariantType.CastTo 23589
Variants.TCustomVariantType.CastToOle 23589
Variants.TCustomVariantType.Clear 23589
Variants.TCustomVariantType.Compare 23590
Variants.TCustomVariantType.CompareOp 23590
Variants.TCustomVariantType.Copy 23591
Variants.TCustomVariantType.Create 23584
Variants.TCustomVariantType.Destroy 23585
Variants.TCustomVariantType.DispInvoke 23592
Variants.TCustomVariantType.IsClear 23592
Variants.TCustomVariantType.LeftPromotion 23593
Variants.TCustomVariantType.OlePromotion 23594
Variants.TCustomVariantType.QueryInterface 23594
Variants.TCustomVariantType.RaiseCastError 23595
Variants.TCustomVariantType.RaiseDispError 23595
Variants.TCustomVariantType.RaiseInvalidOp 23596
Variants.TCustomVariantType.RightPromotion 23596
Variants.TCustomVariantType.SimplisticClear 23597
Variants.TCustomVariantType.SimplisticCopy 23598
Variants.TCustomVariantType.UnaryOp 23598
Variants.TCustomVariantType.VarDataCast 23599
Variants.TCustomVariantType.VarDataCastTo 23599

Variants.TCustomVariantType.VarDataCastToOleStr 23600
Variants.TCustomVariantType.VarDataClear 23600
Variants.TCustomVariantType.VarDataCopy 23601
Variants.TCustomVariantType.VarDataCopyNoInd 23601
Variants.TCustomVariantType.VarDataFromOleStr 23602
Variants.TCustomVariantType.VarDataFromStr 23602
Variants.TCustomVariantType.VarDataInit 23602
Variants.TCustomVariantType.VarDataIsArray 23603
Variants.TCustomVariantType.VarDataIsByRef 23603
Variants.TCustomVariantType.VarDataIsEmptyParam 23603
Variants.TCustomVariantType.VarDataIsFloat 23604
Variants.TCustomVariantType.VarDataIsNumeric 23604
Variants.TCustomVariantType.VarDataIsOrdinal 23605
Variants.TCustomVariantType.VarDataIsStr 23605
Variants.TCustomVariantType.VarDataToStr 23605
Variants.TCustomVariantType.VarType 23606
Variants.TCustomVariantTypeClass 23674
Variants.TCustomVariantTypeClass type 23674
Variants.TInvokeableVariantType 23606
Variants.TInvokeableVariantType.DispInvoke 23610
Variants.TInvokeableVariantType.DoFunction 23610
Variants.TInvokeableVariantType.DoProcedure 23611
Variants.TInvokeableVariantType.FixupIdent 23612
Variants.TInvokeableVariantType.GetProperty 23612
Variants.TInvokeableVariantType.SetProperty 23613
Variants.TNullCompareRule 23671
Variants.TNullCompareRule enumeration 23671
Variants.TVarCompareResult 23672
Variants.TVarCompareResult enumeration 23672
Variants.TVarDataArray 23675
Variants.TVarDataArray type 23675
Variants.TVarDispProc 23675
Variants.TVarDispProc type 23675
Variants.TVariantRelationship 23672
Variants.TVariantRelationship enumeration 23672
Variants.Unassigned 23643
Variants.Unassigned function 23643
Variants.VarArrayAsPSafeArray 23643
Variants.VarArrayAsPSafeArray function 23643
Variants.VarArrayCreate 23644
Variants.VarArrayCreate function 23644
Variants.VarArrayCreateError 23645
Variants.VarArrayCreateError function 23645
Variants.VarArrayDimCount 23645
Variants.VarArrayDimCount function 23645
Variants.VarArrayGet 23645
Variants.VarArrayGet function 23645
Variants.VarArrayHighBound 23646
Variants.VarArrayHighBound function 23646
Variants.VarArrayLock 23646
Variants.VarArrayLock function 23646
Variants.VarArrayLowBound 23647
Variants.VarArrayLowBound function 23647
Variants.VarArrayOf 23648
Variants.VarArrayOf function 23648
Variants.VarArrayPut 23648
Variants.VarArrayPut function 23648
Variants.VarArrayRef 23649
Variants.VarArrayRef function 23649
Variants.VarArrayUnlock 23649
Variants.VarArrayUnlock function 23649
Variants.VarAsError 23650
Variants.VarAsError function 23650
Variants.VarAsType 23650
Variants.VarAsType function 23650
Variants.VarCastError 23651
Variants.VarCastError function 23651
Variants.VarCheckEmpty 23651
Variants.VarCheckEmpty function 23651
Variants.VarCompareValue 23652
Variants.VarCompareValue function 23652
Variants.VarCopyNoInd 23652
Variants.VarCopyNoInd function 23652
Variants.VarDispProc 23681
Variants.VarDispProc variable 23681
Variants.VarEnsureRange 23653
Variants.VarEnsureRange function 23653
Variants.VarFromDateTime 23653

Variants.VarFromDateTime function 23653
Variants.VarInRange 23654
Variants.VarInRange function 23654
Variants.VarInvalidNullOp 23654
Variants.VarInvalidNullOp function 23654
Variants.VarInvalidOp 23655
Variants.VarInvalidOp function 23655
Variants.VarIsArray 23655
Variants.VarIsArray function 23655
Variants.VarIsByRef 23656
Variants.VarIsByRef function 23656
Variants.VarIsClear 23656
Variants.VarIsClear function 23656
Variants.VarIsCustom 23657
Variants.VarIsCustom function 23657
Variants.VarIsEmpty 23657
Variants.VarIsEmpty function 23657
Variants.VarIsEmptyParam 23658
Variants.VarIsEmptyParam function 23658
Variants.VarIsError 23659
Variants.VarIsError function 23659
Variants.VarIsFloat 23659
Variants.VarIsFloat function 23659
Variants.VarIsNull 23660
Variants.VarIsNull function 23660
Variants.VarIsNumeric 23660
Variants.VarIsNumeric function 23660
Variants.VarIsOrdinal 23661
Variants.VarIsOrdinal function 23661
Variants.VarIsStr 23661
Variants.VarIsStr function 23661
Variants.VarIsType 23662
Variants.VarIsType function 23662
Variants.VarOverflowError 23662
Variants.VarOverflowError function 23662
Variants.VarRangeCheckError 23663
Variants.VarRangeCheckError function 23663
Variants.VarResultCheck 23663
Variants.VarResultCheck function 23663
Variants.VarSameValue 23664
Variants.VarSameValue function 23664
Variants.VarSupports 23664
Variants.VarSupports function 23664
Variants.VarToDateTime 23665
Variants.VarToDateTime function 23665
Variants.VarToStr 23665
Variants.VarToStr function 23665
Variants.VarToStrDef 23666
Variants.VarToStrDef function 23666
Variants.VarToWideStr 23666
Variants.VarToWideStr function 23666
Variants.VarToWideStrDef 23667
Variants.VarToWideStrDef function 23667
Variants.VarType 23667
Variants.VarType function 23667
Variants.VarTypeAsText 23668
Variants.VarTypeAsText function 23668
Variants.VarTypeIsValidArrayType 23669
Variants.VarTypeIsValidArrayType function 23669
Variants.VarTypeIsValidElementType 23669
Variants.VarTypeIsValidElementType function 23669
vaSet enumeration member 3279
vaSingle enumeration member 3279
vaString enumeration member 3279
vaTrue enumeration member 3279
vaUTF8String enumeration member 3279
vaWString enumeration member 3279
VCLCom 23683
VCLCom namespace 23683
    Classes 23684
VCLCom.TComponentFactory 23684
VCLCom.TComponentFactory.Create 23686
VCLCom.TComponentFactory.CreateComObject 23686
VCLCom.TComponentFactory.CreateInstance 23687
VCLCom.TComponentFactory.UpdateRegistry 23687
vfByRef enumeration member 19000
vfVariant enumeration member 19000
ViewsColumns class 2193

about ViewsColumns class 2193
CatalogName 2194
Definition 2195
SchemaName 2195
ViewName 2195
ViewsColumns fields 2194
ViewsColumns Members 2194
vmDefault enumeration member 25318
vmDisplay enumeration member 25318
vmInput enumeration member 25318
vmToggleOnAccess enumeration member 25318
vrEqual enumeration member 23672
vrGreaterThan enumeration member 23672
vrLessThan enumeration member 23672
vrNotEqual enumeration member 23672
vsIcon enumeration member 5473
vsList enumeration member 5473
vsReport enumeration member 5473
vsSmallIcon enumeration member 5473
vuBlankPassword enumeration member 25633
vuBlankUserName enumeration member 25633
vuUnknownPassword enumeration member 25633
vuUnknownUserName enumeration member 25633

W

WebAdapt 23689
WebAdapt namespace 23689
Classes 23694
Constants 24613
Functions 24589
Interfaces 24566
Structs, Records, Enums 24593
Types 24595
Variables 24612
WebAdapt.AddDefaultActions 24589
WebAdapt.AddDefaultActions function 24589
WebAdapt.AddDefaultFields 24589
WebAdapt.AddDefaultFields function 24589
WebAdapt.EAdapterException 23696
WebAdapt.EAdapterException.CreateID 23697
WebAdapt.EAdapterException.ID 23698
WebAdapt.EAdapterFieldException 23698
WebAdapt.EAdapterFieldException.Create 23700
WebAdapt.EAdapterFieldException.FieldName 23700
WebAdapt.FindAdapterAction 24590
WebAdapt.FindAdapterAction function 24590
WebAdapt.FindAdapterField 24590
WebAdapt.FindAdapterField function 24590
WebAdapt.FindAdapterInParent 24590
WebAdapt.FindAdapterInParent function 24590
WebAdapt.IActionFieldValue 24596
WebAdapt.IActionFieldValue type 24596
WebAdapt.IActionRequest 24596
WebAdapt.IActionRequest type 24596
WebAdapt.IAdapterEditor 24566
WebAdapt.IAdapterEditor.CanAddActionClass 24567
WebAdapt.IAdapterEditor.CanAddFieldClass 24567
WebAdapt.IAdapterNotifyAdapterChange 24567
WebAdapt.IAdapterNotifyAdapterChange.NotifyAdapterChange 24568
WebAdapt.ICheckOrigValue 24568
WebAdapt.ICheckOrigValue.CheckOrigValue 24569
WebAdapt.ICheckValueChange 24569
WebAdapt.ICheckValueChange.CheckValueChange 24570
WebAdapt.IClearAdapterValues 24570
WebAdapt.IClearAdapterValues.ClearAdapterValues 24571
WebAdapt.ICreateActionRequestContext 24571
WebAdapt.ICreateActionRequestContext.CreateActionRequestContext 24572
WebAdapt.IEchoActionFieldValue 24572
WebAdapt.IEchoActionFieldValue.EchoActionFieldValue 24574
WebAdapt.IEchoActionFieldValue.GetEchoActionFieldValue 24573
WebAdapt.IEchoActionFieldValue.SetEchoActionFieldValue 24573
WebAdapt.IEchoAdapterFieldValues 24574
WebAdapt.IEchoAdapterFieldValues.EchoActionFieldValues 24575
WebAdapt.IEchoAdapterFieldValues.GetEchoActionFieldValues

wwww

24575
WebAdapt.IEchoAdapterFieldValues.SetEchoActionFieldValues 24575
WebAdapt.IExecuteValue 24575
WebAdapt.IExecuteValue.ExecuteValue 24576
WebAdapt.IFullyQualifyInputNames 24577
WebAdapt.IFullyQualifyInputNames.FullyQualifyInputNames 24577
WebAdapt.IGetAdapterErrorsList 24578
WebAdapt.IGetAdapterErrorsList.GetAdapterErrorsList 24578
WebAdapt.IGetAdapterModeNames 24579
WebAdapt.IGetAdapterModeNames.GetAdapterModeNames 24579
WebAdapt.IGetRecordKeys 24580
WebAdapt.IGetRecordKeys.GetRecordKeys 24580
WebAdapt.IIdentifyAdapter 24581
WebAdapt.ISetSubComponents 24581
WebAdapt.ISetSubComponents.SetActions 24582
WebAdapt.ISetSubComponents.SetData 24582
WebAdapt.ISetWebFieldValue 24582
WebAdapt.ISetWebFieldValue.SetValue 24583
WebAdapt.IUpdateValue 24583
WebAdapt.IUpdateValue.UpdateValue 24584
WebAdapt.IValidateValue 24584
WebAdapt.IValidateValue.ValidateValue 24585
WebAdapt.IWebActionsSubcomponent 24585
WebAdapt.IWebFieldsSubcomponent 24585
WebAdapt.IWebGetFieldPrevValue 24586
WebAdapt.IWebGetFieldPrevValue.GetNeedPrevValue 24587
WebAdapt.IWebGetFieldPrevValue.GetPrevValue 24587
WebAdapt.IWebGetFieldPrevValue.NeedPrevValue 24587
WebAdapt.IWebGetFieldPrevValue.PrevValue 24587
WebAdapt.MakeAdapterRequestIdentifier 24591
WebAdapt.MakeAdapterRequestIdentifier function 24591
WebAdapt.MakeValidAdapterFieldName 24591
WebAdapt.MakeValidAdapterFieldName function 24591
WebAdapt.NotifyPossibleHTMLChange 24612
WebAdapt.NotifyPossibleHTMLChange variable 24612
WebAdapt.RaiseFileUploadNotSupported 24591
WebAdapt.RaiseFileUploadNotSupported function 24591
WebAdapt.RaiseMultipleFilesException 24592
WebAdapt.RaiseMultipleFilesException function 24592
WebAdapt.RaiseMultipleValuesException 24592
WebAdapt.RaiseMultipleValuesException function 24592
WebAdapt.sActionRequestLong 24613
WebAdapt.sActionRequestLong constant 24613
WebAdapt.sCurrentPage 24613
WebAdapt.sCurrentPage constant 24613
WebAdapt.sGotoPageAction 24614
WebAdapt.sGotoPageAction constant 24614
WebAdapt.sImageRequest 24614
WebAdapt.sImageRequest constant 24614
WebAdapt.sLoginPassword 24614
WebAdapt.sLoginPassword constant 24614
WebAdapt.sLoginUserName 24615
WebAdapt.sLoginUserName constant 24615
WebAdapt.sNextPageAction 24615
WebAdapt.sNextPageAction constant 24615
WebAdapt.sPrevPageAction 24615
WebAdapt.sPrevPageAction constant 24615
WebAdapt.TAbstractFieldValuesAdapter 23701
WebAdapt.TAbstractFieldValuesAdapter.GetField 23708
WebAdapt.TAbstractFieldValuesAdapter.GetValueFieldClass 23709
WebAdapt.TAbstractFieldValuesAdapter.GetValueIndex 23709
WebAdapt.TAbstractFieldValuesAdapter.ImplCanAddFieldClass 23710
WebAdapt.TAbstractFieldValuesAdapter.ImplEndIterator 23710
WebAdapt.TAbstractFieldValuesAdapter.ImplGetField 23711
WebAdapt.TAbstractFieldValuesAdapter.ImplGetFieldsList 23711
WebAdapt.TAbstractFieldValuesAdapter.ImplGetValueIndex 23711
WebAdapt.TAbstractFieldValuesAdapter.ImplNextIteration 23712
WebAdapt.TAbstractFieldValuesAdapter.ImplStartIterator 23712
WebAdapt.TAbstractWebRequestFile 24596
WebAdapt.TAbstractWebRequestFile type 24596
WebAdapt.TActionExecuteEvent 24597
WebAdapt.TActionExecuteEvent type 24597

WebAdapt.TActionExecuteHandledEvent 24597
WebAdapt.TActionExecuteHandledEvent type 24597
WebAdapt.TActionGetEnabledEvent 24597
WebAdapt.TActionGetEnabledEvent type 24597
WebAdapt.TActionGetParamsEvent 24598
WebAdapt.TActionGetParamsEvent type 24598
WebAdapt.TActionRequestFieldsEvent 24598
WebAdapt.TActionRequestFieldsEvent type 24598
WebAdapt.TActionRequestFieldsHandledEvent 24598
WebAdapt.TActionRequestFieldsHandledEvent type 24598
WebAdapt.TAdapter 23713
WebAdapt.TAdapter.Actions 23721
WebAdapt.TAdapter.Data 23721
WebAdapt.TAdapter.ModifyAccess 23721
WebAdapt.TAdapter.OnAfterExecuteAction 23722
WebAdapt.TAdapter.OnAfterGetActionResponse 23723
WebAdapt.TAdapter.OnAfterUpdateFields 23723
WebAdapt.TAdapter.OnAfterValidateFields 23723
WebAdapt.TAdapter.OnBeforeExecuteAction 23724
WebAdapt.TAdapter.OnBeforeGetActionResponse 23724
WebAdapt.TAdapter.OnBeforeUpdateFields 23725
WebAdapt.TAdapter.OnBeforeValidateFields 23725
WebAdapt.TAdapter.OnGetActionParams 23725
WebAdapt.TAdapter.OnIterateRecords 23726
WebAdapt.TAdapter.ViewAccess 23722
WebAdapt.TAdapterAction 23726
WebAdapt.TAdapterAction.ActionName 23733
WebAdapt.TAdapterAction.DisplayLabel 23734
WebAdapt.TAdapterAction.ExecuteAccess 23734
WebAdapt.TAdapterAction.OnAfterExecute 23735
WebAdapt.TAdapterAction.OnAfterGetResponse 23735
WebAdapt.TAdapterAction.OnBeforeExecute 23736
WebAdapt.TAdapterAction.OnBeforeGetResponse 23736
WebAdapt.TAdapterAction.OnExecute 23737
WebAdapt.TAdapterAction.OnGetEnabled 23737
WebAdapt.TAdapterAction.OnGetParams 23737
WebAdapt.TAdapterAction.RedirectOptions 23734
WebAdapt.TAdapterActionExecuteEvent 24599
WebAdapt.TAdapterActionExecuteEvent type 24599
WebAdapt.TAdapterActionExecuteHandledEvent 24599
WebAdapt.TAdapterActionExecuteHandledEvent type 24599
WebAdapt.TAdapterActionGetParamsEvent 24599
WebAdapt.TAdapterActionGetParamsEvent type 24599
WebAdapt.TAdapterActions 23738
WebAdapt.TAdapterActions.AdapterCanAddClass 23742
WebAdapt.TAdapterActions.ImplCanAddClass 23743
WebAdapt.TAdapterActions.ImplGetWebComponentNodeName 23744
WebAdapt.TAdapterActions.SetWebParent 23744
WebAdapt.TAdapterActions.WebActions 23745
WebAdapt.TAdapterActionsClass 24600
WebAdapt.TAdapterActionsClass type 24600
WebAdapt.TAdapterApplicationField 23745
WebAdapt.TAdapterApplicationField.Adapter 23751
WebAdapt.TAdapterApplicationTitleField 23752
WebAdapt.TAdapterApplicationTitleField.GetDefaultFieldName 23758
WebAdapt.TAdapterApplicationTitleField.ImplGetValue 23758
WebAdapt.TAdapterBooleanField 23758
WebAdapt.TAdapterBooleanField.DisplayLabel 23766
WebAdapt.TAdapterBooleanField.DisplayWidth 23766
WebAdapt.TAdapterBooleanField.FieldName 23767
WebAdapt.TAdapterBooleanField.ModifyAccess 23767
WebAdapt.TAdapterBooleanField.OnGetValue 23769
WebAdapt.TAdapterBooleanField.OnUpdateValue 23769
WebAdapt.TAdapterBooleanField.OnValidateValue 23769
WebAdapt.TAdapterBooleanField.ReadOnly 23767
WebAdapt.TAdapterBooleanField.ValuesList 23768
WebAdapt.TAdapterBooleanField.ViewAccess 23768
WebAdapt.TAdapterDefaultActions 23770
WebAdapt.TAdapterDefaultActions.ClearDefaultComponents 23775
WebAdapt.TAdapterDefaultActions.DefaultWebComponents 23778
WebAdapt.TAdapterDefaultActions.Destroy 23774
WebAdapt.TAdapterDefaultActions.GetAddAllFieldsItem 23775
WebAdapt.TAdapterDefaultActions.GetAddFieldsDlgCaption 23776
WebAdapt.TAdapterDefaultActions.GetAddFieldsItem 23776

yyyy

WebAdapt.TAdapterDefaultActions.GetDefaultWebComponents 23776
WebAdapt.TAdapterDefaultActions.GetNewFieldPrefix 23776
WebAdapt.TAdapterDefaultActions.GetVariablesInUse 23777
WebAdapt.TAdapterDefaultActions.SetDefaultWebComponents 23777
WebAdapt.TAdapterDefaultFields 23778
WebAdapt.TAdapterDefaultFields.ClearDefaultComponents 23784
WebAdapt.TAdapterDefaultFields.DefaultWebComponents 23786
WebAdapt.TAdapterDefaultFields.Destroy 23783
WebAdapt.TAdapterDefaultFields.GetAddAllFieldsItem 23784
WebAdapt.TAdapterDefaultFields.GetAddFieldsDlgCaption 23784
WebAdapt.TAdapterDefaultFields.GetAddFieldsItem 23784
WebAdapt.TAdapterDefaultFields.GetDefaultWebComponents 23785
WebAdapt.TAdapterDefaultFields.GetNewFieldPrefix 23785
WebAdapt.TAdapterDefaultFields.GetVariablesInUse 23785
WebAdapt.TAdapterDefaultFields.SetDefaultWebComponents 23786
WebAdapt.TAdapterEndUserDisplayNameField 23787
WebAdapt.TAdapterEndUserDisplayNameField.GetDefaultFieldName 23793
WebAdapt.TAdapterEndUserDisplayNameField.ImplGetValue 23793
WebAdapt.TAdapterEndUserField 23793
WebAdapt.TAdapterEndUserField.Adapter 23800
WebAdapt.TAdapterEndUserLoggedInField 23800
WebAdapt.TAdapterEndUserLoggedInField.GetDefaultFieldName 23807
WebAdapt.TAdapterEndUserLoggedInField.ImplGetValue 23807
WebAdapt.TAdapterError 23807
WebAdapt.TAdapterError.Create 23809
WebAdapt.TAdapterError.GetErrorText 23810
WebAdapt.TAdapterError.GetField 23810
WebAdapt.TAdapterError.GetID 23810
WebAdapt.TAdapterErrorsList 23810
WebAdapt.TAdapterErrorsList.AddError 23815
WebAdapt.TAdapterErrorsList.Create 23813
WebAdapt.TAdapterErrorsList.Destroy 23814
WebAdapt.TAdapterErrorsList.FieldNames 23817
WebAdapt.TAdapterErrorsList.IDs 23817
WebAdapt.TAdapterErrorsList.ImplClearErrors 23815
WebAdapt.TAdapterErrorsList.ImplDefineLabel 23816
WebAdapt.TAdapterErrorsList.ImplGetError 23816
WebAdapt.TAdapterErrorsList.ImplGetErrorCount 23816
WebAdapt.TAdapterErrorsList.ImplGetObjectErrors 23816
WebAdapt.TAdapterErrorsList.ImplHasObjectErrors 23816
WebAdapt.TAdapterErrorsList.Messages 23818
WebAdapt.TAdapterField 23818
WebAdapt.TAdapterField.DisplayLabel 23826
WebAdapt.TAdapterField.DisplayWidth 23826
WebAdapt.TAdapterField.FieldName 23826
WebAdapt.TAdapterField.ModifyAccess 23827
WebAdapt.TAdapterField.OnGetDisplayText 23829
WebAdapt.TAdapterField.OnGetValue 23829
WebAdapt.TAdapterField.OnUpdateValue 23829
WebAdapt.TAdapterField.OnValidateValue 23830
WebAdapt.TAdapterField.ReadOnly 23827
WebAdapt.TAdapterField.Required 23827
WebAdapt.TAdapterField.ValuesList 23828
WebAdapt.TAdapterField.ViewAccess 23828
WebAdapt.TAdapterFieldBooleanValueEvent 24600
WebAdapt.TAdapterFieldBooleanValueEvent type 24600
WebAdapt.TAdapterFieldFileEvent 24600
WebAdapt.TAdapterFieldFileEvent type 24600
WebAdapt.TAdapterFieldFileListEvent 24601
WebAdapt.TAdapterFieldFileListEvent type 24601
WebAdapt.TAdapterFieldGetBooleanEvent 24601
WebAdapt.TAdapterFieldGetBooleanEvent type 24601
WebAdapt.TAdapterFieldGetMultiValueEvent 24601
WebAdapt.TAdapterFieldGetMultiValueEvent type 24601
WebAdapt.TAdapterFieldGetMultiValueStringEvent 24602
WebAdapt.TAdapterFieldGetMultiValueStringEvent type 24602
WebAdapt.TAdapterFieldGetStringEvent 24602
WebAdapt.TAdapterFieldGetStringEvent type 24602
WebAdapt.TAdapterFieldGetValueCountEvent 24602
WebAdapt.TAdapterFieldGetValueCountEvent type 24602

WebAdapt.TAdapterFieldGetValueEvent 24603
WebAdapt.TAdapterFieldGetValueEvent type 24603
WebAdapt.TAdapterFieldHasValueEvent 24603
WebAdapt.TAdapterFieldHasValueEvent type 24603
WebAdapt.TAdapterFields 23830
WebAdapt.TAdapterFields.AdapterCanAddClass 23835
WebAdapt.TAdapterFields.ImplCanAddClass 23836
WebAdapt.TAdapterFields.ImplGetWebComponentNodeName 23836
WebAdapt.TAdapterFields.SetWebParent 23836
WebAdapt.TAdapterFields.WebFields 23837
WebAdapt.TAdapterFieldsClass 24603
WebAdapt.TAdapterFieldsClass type 24603
WebAdapt.TAdapterFieldStringEvent 24604
WebAdapt.TAdapterFieldStringEvent type 24604
WebAdapt.TAdapterFieldStringsEvent 24604
WebAdapt.TAdapterFieldStringsEvent type 24604
WebAdapt.TAdapterFieldValidateValueEvent 24604
WebAdapt.TAdapterFieldValidateValueEvent type 24604
WebAdapt.TAdapterFieldValueEvent 24605
WebAdapt.TAdapterFieldValueEvent type 24605
WebAdapt.TAdapterFileField 23837
WebAdapt.TAdapterFileField.DisplayLabel 23845
WebAdapt.TAdapterFileField.DisplayWidth 23846
WebAdapt.TAdapterFileField.FieldName 23846
WebAdapt.TAdapterFileField.ModifyAccess 23846
WebAdapt.TAdapterFileField.OnGetFileCount 23848
WebAdapt.TAdapterFileField.OnGetFileNames 23848
WebAdapt.TAdapterFileField.OnUploadFiles 23848
WebAdapt.TAdapterFileField.OnValidateFiles 23849
WebAdapt.TAdapterFileField.ReadOnly 23847
WebAdapt.TAdapterFileField.Required 23847
WebAdapt.TAdapterFileField.ViewAccess 23847
WebAdapt.TAdapterGotoPageAction 23849
WebAdapt.TAdapterGotoPageAction.ExecuteAccess 23856
WebAdapt.TAdapterGotoPageAction.OnAfterExecute 23857
WebAdapt.TAdapterGotoPageAction.OnAfterGetResponse 23858
WebAdapt.TAdapterGotoPageAction.OnBeforeExecute 23858
WebAdapt.TAdapterGotoPageAction.OnBeforeGetResponse 23859
WebAdapt.TAdapterGotoPageAction.OnGetPageName 23859
WebAdapt.TAdapterGotoPageAction.OnGetParams 23859
WebAdapt.TAdapterGotoPageAction.RedirectOptions 23857
WebAdapt.TAdapterHiddenField 23860
WebAdapt.TAdapterHiddenField.Create 23861
WebAdapt.TAdapterHiddenField.GetName 23862
WebAdapt.TAdapterHiddenField.GetValue 23862
WebAdapt.TAdapterHiddenFieldOption 24593
WebAdapt.TAdapterHiddenFieldOption enumeration 24593
WebAdapt.TAdapterHiddenFieldOptions 24605
WebAdapt.TAdapterHiddenFieldOptions type 24605
WebAdapt.TAdapterHiddenFieldsList 23862
WebAdapt.TAdapterHiddenFieldsList.Add 23866
WebAdapt.TAdapterHiddenFieldsList.Clear 23866
WebAdapt.TAdapterHiddenFieldsList.Create 23865
WebAdapt.TAdapterHiddenFieldsList.Destroy 23865
WebAdapt.TAdapterHiddenFieldsList.EndIterator 23866
WebAdapt.TAdapterHiddenFieldsList.FieldCount 23868
WebAdapt.TAdapterHiddenFieldsList.Fields 23868
WebAdapt.TAdapterHiddenFieldsList.GetCurrentIntf 23867
WebAdapt.TAdapterHiddenFieldsList.GetField 23867
WebAdapt.TAdapterHiddenFieldsList.GetFieldCount 23867
WebAdapt.TAdapterHiddenFieldsList.NextIteration 23867
WebAdapt.TAdapterHiddenFieldsList.StartIterator 23868
WebAdapt.TAdapterImageField 23869
WebAdapt.TAdapterImageField.DisplayLabel 23876
WebAdapt.TAdapterImageField.DisplayWidth 23877
WebAdapt.TAdapterImageField.FieldName 23877
WebAdapt.TAdapterImageField.HREF 23877
WebAdapt.TAdapterImageField.ModifyAccess 23878
WebAdapt.TAdapterImageField.OnGetHREF 23880
WebAdapt.TAdapterImageField.OnGetImage 23880
WebAdapt.TAdapterImageField.OnGetImageName 23881
WebAdapt.TAdapterImageField.OnGetParams 23881
WebAdapt.TAdapterImageField.OnUploadFile 23881
WebAdapt.TAdapterImageField.OnValidateFile 23882
WebAdapt.TAdapterImageField.ReadOnly 23878
WebAdapt.TAdapterImageField.Required 23879

WebAdapt.TAdapterImageField.ValuesList 23879
WebAdapt.TAdapterImageField.ViewAccess 23879
WebAdapt.TAdapterLoginField 23882
WebAdapt.TAdapterLoginField.Adapter 23888
WebAdapt.TAdapterMemoField 23889
WebAdapt.TAdapterMemoField.DisplayLabel 23897
WebAdapt.TAdapterMemoField.DisplayWidth 23897
WebAdapt.TAdapterMemoField.FieldName 23898
WebAdapt.TAdapterMemoField.ModifyAccess 23898
WebAdapt.TAdapterMemoField.OnGetValue 23899
WebAdapt.TAdapterMemoField.OnUpdateValue 23900
WebAdapt.TAdapterMemoField.OnUploadFile 23900
WebAdapt.TAdapterMemoField.ReadOnly 23898
WebAdapt.TAdapterMemoField.Required 23899
WebAdapt.TAdapterMemoField.ViewAccess 23899
WebAdapt.TAdapterMultiValueField 23900
WebAdapt.TAdapterMultiValueField.DisplayLabel 23908
WebAdapt.TAdapterMultiValueField.DisplayWidth 23909
WebAdapt.TAdapterMultiValueField.FieldName 23909
WebAdapt.TAdapterMultiValueField.ModifyAccess 23909
WebAdapt.TAdapterMultiValueField.OnGetValueCount 23911
WebAdapt.TAdapterMultiValueField.OnGetValues 23912
WebAdapt.TAdapterMultiValueField.OnHasValue 23912
WebAdapt.TAdapterMultiValueField.OnUpdateValues 23912
WebAdapt.TAdapterMultiValueField.OnValidateValues 23913
WebAdapt.TAdapterMultiValueField.ReadOnly 23910
WebAdapt.TAdapterMultiValueField.Required 23910
WebAdapt.TAdapterMultiValueField.ValuesList 23910
WebAdapt.TAdapterMultiValueField.ViewAccess 23911
WebAdapt.TAdapterNamedDisplayField 23913
WebAdapt.TAdapterNamedDisplayField.Create 23919
WebAdapt.TAdapterNamedDisplayField.DisplayLabel 23923
WebAdapt.TAdapterNamedDisplayField.DisplayWidth 23924
WebAdapt.TAdapterNamedDisplayField.GetDisplayStyle 23920
WebAdapt.TAdapterNamedDisplayField.GetDisplayStyleType 23920
WebAdapt.TAdapterNamedDisplayField.GetInputStyle 23920
WebAdapt.TAdapterNamedDisplayField.GetInputStyleType 23921
WebAdapt.TAdapterNamedDisplayField.GetViewMode 23921
WebAdapt.TAdapterNamedDisplayField.GetViewModeType 23921
WebAdapt.TAdapterNamedDisplayField.ImplGetDisplayLabel 23922
WebAdapt.TAdapterNamedDisplayField.ImplGetDisplayWidth 23922
WebAdapt.TAdapterNamedDisplayField.ImplGetRequired 23922
WebAdapt.TAdapterNamedDisplayField.ReadOnly 23924
WebAdapt.TAdapterNamedDisplayField.Required 23924
WebAdapt.TAdapterNamedDisplayField.SetDisplayLabel 23923
WebAdapt.TAdapterNamedField 23925
WebAdapt.TAdapterNamedField.ActionOrigValue 23943
WebAdapt.TAdapterNamedField.ActionValue 23943
WebAdapt.TAdapterNamedField.Adapter 23943
WebAdapt.TAdapterNamedField.AsFormatted 23931
WebAdapt.TAdapterNamedField.CheckModifyAccess 23931
WebAdapt.TAdapterNamedField.CheckViewAccess 23931
WebAdapt.TAdapterNamedField.EchoActionFieldValue 23944
WebAdapt.TAdapterNamedField.FieldName 23944
WebAdapt.TAdapterNamedField.FullyQualifyInputName 23932
WebAdapt.TAdapterNamedField.GetAdapter 23932
WebAdapt.TAdapterNamedField.GetDefaultDisplayLabel 23932
WebAdapt.TAdapterNamedField.GetDefaultFieldName 23932
WebAdapt.TAdapterNamedField.GetDisplayLabel 23933
WebAdapt.TAdapterNamedField.GetDisplayWidth 23933
WebAdapt.TAdapterNamedField.GetEchoActionFieldValue 23933
WebAdapt.TAdapterNamedField.GetInputName 23934
WebAdapt.TAdapterNamedField.GetMaxLength 23934
WebAdapt.TAdapterNamedField.GetRequired 23934
WebAdapt.TAdapterNamedField.GetValue 23935
WebAdapt.TAdapterNamedField.GetVariableName 23935
WebAdapt.TAdapterNamedField.GetVisible 23935
WebAdapt.TAdapterNamedField.GetWebSnapObjClass 23936
WebAdapt.TAdapterNamedField.HasModifyAccess 23936
WebAdapt.TAdapterNamedField.HasViewAccess 23936
WebAdapt.TAdapterNamedField.ImplAsFormatted 23937
WebAdapt.TAdapterNamedField.ImplGetDisplayLabel 23937
WebAdapt.TAdapterNamedField.ImplGetDisplayWidth 23937
WebAdapt.TAdapterNamedField.ImplGetEchoActionFieldValue 23938

bbbb

WebAdapt.TAdapterNamedField.ImplGetInputName 23938
WebAdapt.TAdapterNamedField.ImplGetMaxLength 23938
WebAdapt.TAdapterNamedField.ImplGetRequired 23939
WebAdapt.TAdapterNamedField.ImplGetValue 23939
WebAdapt.TAdapterNamedField.ImplGetVariableName 23939
WebAdapt.TAdapterNamedField.ImplGetVisible 23940
WebAdapt.TAdapterNamedField.ImplHasModifyAccess 23940
WebAdapt.TAdapterNamedField.ImplHasViewAccess 23940
WebAdapt.TAdapterNamedField.ImplSetEchoActionFieldValue 23941
WebAdapt.TAdapterNamedField.ImplSetFieldName 23941
WebAdapt.TAdapterNamedField.ModifyAccess 23944
WebAdapt.TAdapterNamedField.ParentChanged 23941
WebAdapt.TAdapterNamedField.SetEchoActionFieldValue 23942
WebAdapt.TAdapterNamedField.SetFieldName 23942
WebAdapt.TAdapterNamedField.Value 23945
WebAdapt.TAdapterNamedField.ViewAccess 23945
WebAdapt.TAdapterNextPageAction 23945
WebAdapt.TAdapterNextPageAction.DisplayLabel 23954
WebAdapt.TAdapterNextPageAction.ExecuteAccess 23954
WebAdapt.TAdapterNextPageAction.GetDefaultActionName 23952
WebAdapt.TAdapterNextPageAction.GetDefaultDisplayLabel 23953
WebAdapt.TAdapterNextPageAction.GetNewPage 23953
WebAdapt.TAdapterNextPageAction.OnAfterExecute 23955
WebAdapt.TAdapterNextPageAction.OnAfterGetResponse 23955
WebAdapt.TAdapterNextPageAction.OnBeforeExecute 23956
WebAdapt.TAdapterNextPageAction.OnBeforeGetResponse 23956
WebAdapt.TAdapterNextPageAction.OnGetParams 23957
WebAdapt.TAdapterNextPageAction.RedirectOptions 23954
WebAdapt.TAdapterNextPageField 23957
WebAdapt.TAdapterNextPageField.Destroy 23963
WebAdapt.TAdapterNextPageField.DisplayLabel 23966
WebAdapt.TAdapterNextPageField.DisplayWidth 23967
WebAdapt.TAdapterNextPageField.FieldName 23967
WebAdapt.TAdapterNextPageField.GetAdapterValuesList 23964
WebAdapt.TAdapterNextPageField.GetInputStyleType 23964
WebAdapt.TAdapterNextPageField.ImplGetEchoActionFieldValue 23965
WebAdapt.TAdapterNextPageField.ImplGetValue 23965
WebAdapt.TAdapterNextPageField.ImplSetEchoActionFieldValue 23965
WebAdapt.TAdapterNextPageField.UpdateValue 23966
WebAdapt.TAdapterObjectErrorsList 23968
WebAdapt.TAdapterObjectErrorsList.Create 23970
WebAdapt.TAdapterObjectErrorsList.ImplDefineLabel 23970
WebAdapt.TAdapterObjectErrorsList.ImplGetError 23970
WebAdapt.TAdapterObjectErrorsList.ImplGetErrorCount 23971
WebAdapt.TAdapterPasswordField 23971
WebAdapt.TAdapterPasswordField.DisplayLabel 23978
WebAdapt.TAdapterPasswordField.DisplayWidth 23979
WebAdapt.TAdapterPasswordField.FieldName 23979
WebAdapt.TAdapterPasswordField.GetInputStyleType 23978
WebAdapt.TAdapterPasswordField.ImplGetValue 23978
WebAdapt.TAdapterPrevPageAction 23979
WebAdapt.TAdapterPrevPageAction.DisplayLabel 23988
WebAdapt.TAdapterPrevPageAction.ExecuteAccess 23988
WebAdapt.TAdapterPrevPageAction.GetDefaultActionName 23986
WebAdapt.TAdapterPrevPageAction.GetDefaultDisplayLabel 23987
WebAdapt.TAdapterPrevPageAction.GetNewPage 23987
WebAdapt.TAdapterPrevPageAction.OnAfterExecute 23989
WebAdapt.TAdapterPrevPageAction.OnAfterGetResponse 23989
WebAdapt.TAdapterPrevPageAction.OnBeforeExecute 23990
WebAdapt.TAdapterPrevPageAction.OnBeforeGetResponse 23990
WebAdapt.TAdapterPrevPageAction.OnGetParams 23991
WebAdapt.TAdapterPrevPageAction.RedirectOptions 23988
WebAdapt.TAdapterSubComponent 23991
WebAdapt.TAdapterSubComponent.Adapter 24006
WebAdapt.TAdapterSubComponent.CanAddClass 23996
WebAdapt.TAdapterSubComponent.EndIterator 23997
WebAdapt.TAdapterSubComponent.FindNamedVariable 23997
WebAdapt.TAdapterSubComponent.FindVariable 23998
WebAdapt.TAdapterSubComponent.FindVariableContainer

23998
WebAdapt.TAdapterSubComponent.GetCurrentObject 23999
WebAdapt.TAdapterSubComponent.GetDefaultWebComponent
s
23999
WebAdapt.TAdapterSubComponent.GetParentComponent
24000
WebAdapt.TAdapterSubComponent.GetVariable 24000
WebAdapt.TAdapterSubComponent.GetVariableCount 24000
WebAdapt.TAdapterSubComponent.GetVariablesInUse 24001
WebAdapt.TAdapterSubComponent.GetWebComponentNodeN
ame
24001
WebAdapt.TAdapterSubComponent.HasParent 24002
WebAdapt.TAdapterSubComponent.ImplCanAddClass 24002
WebAdapt.TAdapterSubComponent.ImplGetWebComponentNo
deName
24003
WebAdapt.TAdapterSubComponent.NextIteration 24003
WebAdapt.TAdapterSubComponent.NotifyAdapterChange
24003
WebAdapt.TAdapterSubComponent.ReadState 24004
WebAdapt.TAdapterSubComponent.SetDefaultWebComponent
s
24004
WebAdapt.TAdapterSubComponent.SetParentComponent
24005
WebAdapt.TAdapterSubComponent.SetWebParent 24005
WebAdapt.TAdapterSubComponent.StartIterator 24006
WebAdapt.TAdapterSubComponent.VariablesInUse 24007
WebAdapt.TAdapterUpdateField 24007
WebAdapt.TAdapterUpdateField.CheckValueChange 24015
WebAdapt.TAdapterUpdateField.ConvertValue 24015
WebAdapt.TAdapterUpdateField.ConvertValueCount 24015
WebAdapt.TAdapterUpdateField.ConvertValues 24016
WebAdapt.TAdapterUpdateField.Create 24014
WebAdapt.TAdapterUpdateField.DoOnValidateValue 24016
WebAdapt.TAdapterUpdateField.GetAdapterValuesList 24016
WebAdapt.TAdapterUpdateField.GetInputStyleType 24017
WebAdapt.TAdapterUpdateField.GetValueCount 24017
WebAdapt.TAdapterUpdateField.GetValues 24017
WebAdapt.TAdapterUpdateField.ImplAsFormatted 24018
WebAdapt.TAdapterUpdateField.ImplCheckValueChange 24018
WebAdapt.TAdapterUpdateField.ImplGetDisplayLabel 24018
WebAdapt.TAdapterUpdateField.ImplGetDisplayWidth 24019
WebAdapt.TAdapterUpdateField.ImplGetEchoActionFieldValue
24019
WebAdapt.TAdapterUpdateField.ImplGetVariableName 24019
WebAdapt.TAdapterUpdateField.ImplSetEchoActionFieldValue
24020
WebAdapt.TAdapterUpdateField.ImplSetFieldName 24020
WebAdapt.TAdapterUpdateField.ImplUpdateValue 24020
WebAdapt.TAdapterUpdateField.ImplValidateValue 24020
WebAdapt.TAdapterUpdateField.IsMultiValueField 24021
WebAdapt.TAdapterUpdateField.Notification 24021
WebAdapt.TAdapterUpdateField.OnGetDisplayText 24023
WebAdapt.TAdapterUpdateField.OnValidateValue 24024
WebAdapt.TAdapterUpdateField.UpdateValue 24021
WebAdapt.TAdapterUpdateField.ValidateValue 24022
WebAdapt.TAdapterUpdateField.Value 24022
WebAdapt.TAdapterUpdateField.ValuesList 24023
WebAdapt.TAdapterUserNameField 24024
WebAdapt.TAdapterUserNameField.DisplayLabel 24031
WebAdapt.TAdapterUserNameField.DisplayWidth 24031
WebAdapt.TAdapterUserNameField.FieldName 24032
WebAdapt.TAdapterUserNameField.ImplGetValue 24031
WebAdapt.TApplicationAdapter 24032
WebAdapt.TApplicationAdapter.Actions 24040
WebAdapt.TApplicationAdapter.ApplicationTitle 24040
WebAdapt.TApplicationAdapter.Data 24041
WebAdapt.TApplicationAdapter.OnAfterExecuteAction 24041
WebAdapt.TApplicationAdapter.OnAfterGetActionResponse
24042
WebAdapt.TApplicationAdapter.OnBeforeExecuteAction 24042
WebAdapt.TApplicationAdapter.OnBeforeGetActionResponse
24042
WebAdapt.TApplicationAdapter.OnGetActionParams 24043
WebAdapt.TBaseAdapterAction 24043
WebAdapt.TBaseAdapterAction.ActionName 24072
WebAdapt.TBaseAdapterAction.Adapter 24072
WebAdapt.TBaseAdapterAction.CheckExecuteAccess 24050
WebAdapt.TBaseAdapterAction.CreateRequestContext 24050
WebAdapt.TBaseAdapterAction.DisplayLabel 24073
WebAdapt.TBaseAdapterAction.DoAfterExecuteActionRequest

24051

WebAdapt.TBaseAdapterAction.DoAfterGetExecuteActionResponse
24051

WebAdapt.TBaseAdapterAction.DoBeforeExecuteActionRequest
24052

WebAdapt.TBaseAdapterAction.DoBeforeGetExecuteActionResponse
24052

WebAdapt.TBaseAdapterAction.EndIterator 24053

WebAdapt.TBaseAdapterAction.Errors 24073

WebAdapt.TBaseAdapterAction.ExecuteAccess 24073

WebAdapt.TBaseAdapterAction.ExecuteActionRequest 24053

WebAdapt.TBaseAdapterAction.GetActionName 24054

WebAdapt.TBaseAdapterAction.GetAdapter 24054

WebAdapt.TBaseAdapterAction.GetAdapterItemRequestParams

24055

WebAdapt.TBaseAdapterAction.GetDefaultActionName 24055

WebAdapt.TBaseAdapterAction.GetDefaultDisplayLabel 24055

WebAdapt.TBaseAdapterAction.GetDisplayLabel 24055

WebAdapt.TBaseAdapterAction.GetDisplayStyle 24056

WebAdapt.TBaseAdapterAction.GetDisplayStyleType 24056

WebAdapt.TBaseAdapterAction.GetErrors 24057

WebAdapt.TBaseAdapterAction.GetExecuteActionResponse
24057

WebAdapt.TBaseAdapterAction.GetFailureOptions 24057

WebAdapt.TBaseAdapterAction.GetInputStyle 24058

WebAdapt.TBaseAdapterAction.GetIsAdapterActionList 24058

WebAdapt.TBaseAdapterAction.GetItemCount 24058

WebAdapt.TBaseAdapterAction.GetIteratorIndex 24059

WebAdapt.TBaseAdapterAction.GetSuccessOptions 24059

WebAdapt.TBaseAdapterAction.GetVariableName 24060

WebAdapt.TBaseAdapterAction.GetViewMode 24060

WebAdapt.TBaseAdapterAction.GetViewModeType 24060

WebAdapt.TBaseAdapterAction.GetVisible 24061

WebAdapt.TBaseAdapterAction.GetWebSnapObjClass 24061

WebAdapt.TBaseAdapterAction.HandleRequest 24061

WebAdapt.TBaseAdapterAction.HasExecuteAccess 24062

WebAdapt.TBaseAdapterAction.ImplCreateRequestContext
24062

WebAdapt.TBaseAdapterAction.ImplEndIterator 24062

WebAdapt.TBaseAdapterAction.ImplExecuteActionRequest
24063

WebAdapt.TBaseAdapterAction.ImplGetActionName 24063

WebAdapt.TBaseAdapterAction.ImplGetAdapterItemRequestParams
24063

WebAdapt.TBaseAdapterAction.ImplGetDisplayLabel 24064

WebAdapt.TBaseAdapterAction.ImplGetExecuteActionResponse
24064

WebAdapt.TBaseAdapterAction.ImplGetFailureOptions 24065

WebAdapt.TBaseAdapterAction.ImplGetIsAdapterActionList
24065

WebAdapt.TBaseAdapterAction.ImplGetIteratorIndex 24065

WebAdapt.TBaseAdapterAction.ImplGetSuccessOptions 24066

WebAdapt.TBaseAdapterAction.ImplGetVariableName 24066

WebAdapt.TBaseAdapterAction.ImplGetVisible 24066

WebAdapt.TBaseAdapterAction.ImplHandleRequest 24067

WebAdapt.TBaseAdapterAction.ImplHasExecuteAccess 24067

WebAdapt.TBaseAdapterAction.ImplInIterator 24067

WebAdapt.TBaseAdapterAction.ImplIsDefaultAction 24068

WebAdapt.TBaseAdapterAction.ImplNextIteration 24068

WebAdapt.TBaseAdapterAction.ImplStartIterator 24068

WebAdapt.TBaseAdapterAction.ImplWebEnabled 24069

WebAdapt.TBaseAdapterAction.InIterator 24069

WebAdapt.TBaseAdapterAction.IsDefaultAction 24070

WebAdapt.TBaseAdapterAction.ItemCount 24074

WebAdapt.TBaseAdapterAction.ItemIndex 24074

WebAdapt.TBaseAdapterAction.NextIteration 24070

WebAdapt.TBaseAdapterAction.OnAfterExecute 24075

WebAdapt.TBaseAdapterAction.OnAfterGetResponse 24075

WebAdapt.TBaseAdapterAction.OnBeforeExecute 24076

WebAdapt.TBaseAdapterAction.OnBeforeGetResponse 24076

WebAdapt.TBaseAdapterAction.OnGetParams 24077

WebAdapt.TBaseAdapterAction.ParentChanged 24070

WebAdapt.TBaseAdapterAction.RedirectOptions 24074

WebAdapt.TBaseAdapterAction.SetDisplayLabel 24071

WebAdapt.TBaseAdapterAction.StartIterator 24071

WebAdapt.TBaseAdapterAction.WebEnabled 24071

WebAdapt.TBaseAdapterErrorsList 24077

WebAdapt.TBaseAdapterErrorsList.AddError 24080

eeee

WebAdapt.TBaseAdapterErrorsList.ClearErrors 24080
WebAdapt.TBaseAdapterErrorsList.DefineLabel 24081
WebAdapt.TBaseAdapterErrorsList.EndIterator 24081
WebAdapt.TBaseAdapterErrorsList.ErrorCount 24085
WebAdapt.TBaseAdapterErrorsList.Errors 24085
WebAdapt.TBaseAdapterErrorsList.GetCurrentIntf 24081
WebAdapt.TBaseAdapterErrorsList.GetError 24082
WebAdapt.TBaseAdapterErrorsList.GetErrorCount 24082
WebAdapt.TBaseAdapterErrorsList.GetObjectErrors 24082
WebAdapt.TBaseAdapterErrorsList.HasObjectErrors 24082
WebAdapt.TBaseAdapterErrorsList.ImplClearErrors 24083
WebAdapt.TBaseAdapterErrorsList.ImplDefineLabel 24083
WebAdapt.TBaseAdapterErrorsList.ImplGetError 24083
WebAdapt.TBaseAdapterErrorsList.ImplGetErrorCount 24083
WebAdapt.TBaseAdapterErrorsList.ImplGetObjectErrors 24084
WebAdapt.TBaseAdapterErrorsList.ImplHasObjectErrors 24084
WebAdapt.TBaseAdapterErrorsList.NextIteration 24084
WebAdapt.TBaseAdapterErrorsList.StartIterator 24085
WebAdapt.TBaseAdapterMultiValueField 24086
WebAdapt.TBaseAdapterMultiValueField.Create 24092
WebAdapt.TBaseAdapterMultiValueField.Destroy 24093
WebAdapt.TBaseAdapterMultiValueField.EndValueIterator 24093
WebAdapt.TBaseAdapterMultiValueField.GetCurrentValue 24094
WebAdapt.TBaseAdapterMultiValueField.GetFieldValuesAdapter 24094
WebAdapt.TBaseAdapterMultiValueField.GetValueCount 24094
WebAdapt.TBaseAdapterMultiValueField.GetValues 24095
WebAdapt.TBaseAdapterMultiValueField.HasValue 24095
WebAdapt.TBaseAdapterMultiValueField.ImplGetCurrentValue 24095
WebAdapt.TBaseAdapterMultiValueField.ImplGetFieldValuesAdapter 24096
WebAdapt.TBaseAdapterMultiValueField.ImplHasValue 24096
WebAdapt.TBaseAdapterMultiValueField.NextValueIteration 24096
WebAdapt.TBaseAdapterMultiValueField.StartValueIterator 24097
WebAdapt.TBaseEndUserAdapter 24097
WebAdapt.TBaseEndUserAdapter.CanExecuteLoginForm 24106
WebAdapt.TBaseEndUserAdapter.CanExecuteLogout 24107
WebAdapt.TBaseEndUserAdapter.CanProduceLoginPage 24107
WebAdapt.TBaseEndUserAdapter.DoOnHasRights 24107
WebAdapt.TBaseEndUserAdapter.ExecuteLogin 24108
WebAdapt.TBaseEndUserAdapter.ExecuteLoginForm 24108
WebAdapt.TBaseEndUserAdapter.ExecuteLogout 24108
WebAdapt.TBaseEndUserAdapter.GetDisplayName 24109
WebAdapt.TBaseEndUserAdapter.GetDisplayNameValue 24109
WebAdapt.TBaseEndUserAdapter.GetEndUserAdapter 24109
WebAdapt.TBaseEndUserAdapter.GetLoggedIn 24110
WebAdapt.TBaseEndUserAdapter.GetLoggedInValue 24110
WebAdapt.TBaseEndUserAdapter.GetLoginFormAction 24110
WebAdapt.TBaseEndUserAdapter.GetLogoutAction 24111
WebAdapt.TBaseEndUserAdapter.GetUserID 24111
WebAdapt.TBaseEndUserAdapter.HasExecuteRights 24111
WebAdapt.TBaseEndUserAdapter.HasModifyRights 24112
WebAdapt.TBaseEndUserAdapter.HasViewRights 24112
WebAdapt.TBaseEndUserAdapter.ImplCanAddActionClass 24113
WebAdapt.TBaseEndUserAdapter.ImplCanAddFieldClass 24113
WebAdapt.TBaseEndUserAdapter.ImplGetActionsList 24113
WebAdapt.TBaseEndUserAdapter.ImplGetDisplayName 24114
WebAdapt.TBaseEndUserAdapter.ImplGetEndUserAdapter 24114
WebAdapt.TBaseEndUserAdapter.ImplGetFieldsList 24114
WebAdapt.TBaseEndUserAdapter.ImplGetLoggedIn 24114
WebAdapt.TBaseEndUserAdapter.ImplGetLoginFormAction 24115
WebAdapt.TBaseEndUserAdapter.ImplGetLogoutAction 24115
WebAdapt.TBaseEndUserAdapter.ImplHasExecuteRights 24115
WebAdapt.TBaseEndUserAdapter.ImplHasModifyRights 24116
WebAdapt.TBaseEndUserAdapter.ImplHasViewRights 24116
WebAdapt.TBaseEndUserAdapter.ImplLogIn 24116
WebAdapt.TBaseEndUserAdapter.ImplProduceLoginPage 24117
WebAdapt.TBaseEndUserAdapter.LogIn 24117

WebAdapt.TBaseEndUserAdapter.LoginPage 24118
WebAdapt.TBaseEndUserAdapter.OnHasRights 24119
WebAdapt.TBaseEndUserAdapter.OnProduceLoginPage 24119
WebAdapt.TBaseEndUserAdapter.ProduceLoginPage 24117
WebAdapt.TBaseEndUserAdapter.UserList 24118
WebAdapt.TBaseValuesListAdapter 24120
WebAdapt.TBaseValuesListAdapter.GetListImage 24128
WebAdapt.TBaseValuesListAdapter.GetListImageOfValue 24128
WebAdapt.TBaseValuesListAdapter.GetListName 24128
WebAdapt.TBaseValuesListAdapter.GetListNameOfValue 24128
WebAdapt.TBaseValuesListAdapter.GetListValue 24129
WebAdapt.TBaseValuesListAdapter.GetNameFieldClass 24129
WebAdapt.TBaseValuesListAdapter.GetValueFieldClass 24129
WebAdapt.TBaseValuesListAdapter.GetWebSnapObjClass 24130
WebAdapt.TBaseValuesListAdapter.ImplCanAddFieldClass 24130
WebAdapt.TBaseValuesListAdapter.ImplGetFieldsList 24130
WebAdapt.TBaseValuesListAdapter.ImplGetListImage 24131
WebAdapt.TBaseValuesListAdapter.ImplGetListImageOfValue 24131
WebAdapt.TBaseValuesListAdapter.ImplGetListName 24131
WebAdapt.TBaseValuesListAdapter.ImplGetListNameOfValue 24132
WebAdapt.TBaseValuesListAdapter.ImplGetListValue 24132
WebAdapt.TBaseValuesListField 24132
WebAdapt.TBaseValuesListField.Adapter 24139
WebAdapt.TBaseValuesListField.GetAdapter 24138
WebAdapt.TCustomAdapter 24139
WebAdapt.TCustomAdapter.Actions 24181
WebAdapt.TCustomAdapter.AddAdapterModeHiddenField 24149
WebAdapt.TCustomAdapter.AddOrigValueHiddenFields 24149
WebAdapt.TCustomAdapter.AddRecordKeysHiddenFields 24149
WebAdapt.TCustomAdapter.CanAddActionClass 24150
WebAdapt.TCustomAdapter.CanAddFieldClass 24150
WebAdapt.TCustomAdapter.CheckModifyAccess 24150
WebAdapt.TCustomAdapter.ClearAdapterValues 24151
WebAdapt.TCustomAdapter.ClearErrors 24151
WebAdapt.TCustomAdapter.Create 24146
WebAdapt.TCustomAdapter.CreateActionRequestContext 24151
WebAdapt.TCustomAdapter.CreateErrorsList 24152
WebAdapt.TCustomAdapter.Data 24181
WebAdapt.TCustomAdapter.Destroy 24146
WebAdapt.TCustomAdapter.EchoActionFieldValues 24182
WebAdapt.TCustomAdapter.EncodeActionParams 24152
WebAdapt.TCustomAdapter.EndIterator 24152
WebAdapt.TCustomAdapter.Errors 24182
WebAdapt.TCustomAdapter.ExecuteAccess 24183
WebAdapt.TCustomAdapter.ExtractRequestParams 24153
WebAdapt.TCustomAdapter.FindAction 24153
WebAdapt.TCustomAdapter.FindField 24154
WebAdapt.TCustomAdapter.FindNamedAction 24154
WebAdapt.TCustomAdapter.FindNamedField 24154
WebAdapt.TCustomAdapter.FindNamedVariable 24155
WebAdapt.TCustomAdapter.FindVariable 24155
WebAdapt.TCustomAdapter.FindVariableContainer 24155
WebAdapt.TCustomAdapter.GetActionsClass 24156
WebAdapt.TCustomAdapter.GetActionsList 24156
WebAdapt.TCustomAdapter.GetAdapterActions 24156
WebAdapt.TCustomAdapter.GetAdapterErrors 24157
WebAdapt.TCustomAdapter.GetAdapterErrorsList 24157
WebAdapt.TCustomAdapter.GetAdapterFields 24157
WebAdapt.TCustomAdapter.GetAdapterHiddenFields 24157
WebAdapt.TCustomAdapter.GetAdapterHiddenRecordFields 24158
WebAdapt.TCustomAdapter.GetChildren 24158
WebAdapt.TCustomAdapter.GetDataClass 24158
WebAdapt.TCustomAdapter.GetEchoActionFieldValues 24159
WebAdapt.TCustomAdapter.GetFieldsList 24159
WebAdapt.TCustomAdapter.GetHiddenFieldOptions 24159
WebAdapt.TCustomAdapter.GetIteratorIndex 24160
WebAdapt.TCustomAdapter.GetVariable 24160
WebAdapt.TCustomAdapter.GetVariableCount 24161
WebAdapt.TCustomAdapter.GetVariableName 24161
WebAdapt.TCustomAdapter.GetVisibleActions 24161
WebAdapt.TCustomAdapter.GetVisibleFields 24161
WebAdapt.TCustomAdapter.GetWebSnapObjClass 24162

gggggg

WebAdapt.TCustomAdapter.HasExecuteAccess 24162
WebAdapt.TCustomAdapter.HasModifyAccess 24162
WebAdapt.TCustomAdapter.HasViewAccess 24162
WebAdapt.TCustomAdapter.ImplCanAddActionClass 24163
WebAdapt.TCustomAdapter.ImplCanAddFieldClass 24163
WebAdapt.TCustomAdapter.ImplClearAdapterValues 24163
WebAdapt.TCustomAdapter.ImplEndIterator 24164
WebAdapt.TCustomAdapter.ImplFindVariable 24164
WebAdapt.TCustomAdapter.ImplFindVariableContainer 24164
WebAdapt.TCustomAdapter.ImplGetActionsList 24165
WebAdapt.TCustomAdapter.ImplGetEchoActionFieldValues 24165
WebAdapt.TCustomAdapter.ImplGetFieldsList 24165
WebAdapt.TCustomAdapter.ImplGetIteratorIndex 24166
WebAdapt.TCustomAdapter.ImplGetVariable 24166
WebAdapt.TCustomAdapter.ImplGetVariableCount 24167
WebAdapt.TCustomAdapter.ImplGetVariableName 24167
WebAdapt.TCustomAdapter.ImplGetVisibleActions 24167
WebAdapt.TCustomAdapter.ImplGetVisibleFields 24167
WebAdapt.TCustomAdapter.ImplHasExecuteAccess 24168
WebAdapt.TCustomAdapter.ImplHasModifyAccess 24168
WebAdapt.TCustomAdapter.ImplHasViewAccess 24169
WebAdapt.TCustomAdapter.ImplInIterator 24169
WebAdapt.TCustomAdapter.ImplNextIteration 24169
WebAdapt.TCustomAdapter.ImplNotifyActivate 24170
WebAdapt.TCustomAdapter.ImplNotifyDeactivate 24170
WebAdapt.TCustomAdapter.ImplSetActions 24170
WebAdapt.TCustomAdapter.ImplSetData 24171
WebAdapt.TCustomAdapter.ImplSetEchoActionFieldValues 24171
WebAdapt.TCustomAdapter.ImplStartIterator 24171
WebAdapt.TCustomAdapter.InIterator 24172
WebAdapt.TCustomAdapter.IsActionInUse 24172
WebAdapt.TCustomAdapter.IsFieldInUse 24172
WebAdapt.TCustomAdapter.ListHiddenFields 24173
WebAdapt.TCustomAdapter.ListHiddenRecordFields 24173
WebAdapt.TCustomAdapter.ModifyAccess 24183
WebAdapt.TCustomAdapter.NextIteration 24173
WebAdapt.TCustomAdapter.Notification 24174
WebAdapt.TCustomAdapter.NotifyActivate 24174
WebAdapt.TCustomAdapter.NotifyAdapterChange 24174
WebAdapt.TCustomAdapter.NotifyDeactivate 24175
WebAdapt.TCustomAdapter.NotifyList 24183
WebAdapt.TCustomAdapter.OnAfterExecuteAction 24184
WebAdapt.TCustomAdapter.OnAfterGetActionResponse 24185
WebAdapt.TCustomAdapter.OnAfterUpdateFields 24185
WebAdapt.TCustomAdapter.OnAfterValidateFields 24186
WebAdapt.TCustomAdapter.OnBeforeExecuteAction 24186
WebAdapt.TCustomAdapter.OnBeforeGetActionResponse 24186
WebAdapt.TCustomAdapter.OnBeforeUpdateFields 24187
WebAdapt.TCustomAdapter.OnBeforeValidateFields 24187
WebAdapt.TCustomAdapter.OnGetActionParams 24187
WebAdapt.TCustomAdapter.OnIterateRecords 24188
WebAdapt.TCustomAdapter.SetActions 24175
WebAdapt.TCustomAdapter.SetData 24175
WebAdapt.TCustomAdapter.SetEchoActionFieldValues 24176
WebAdapt.TCustomAdapter.StartIterator 24176
WebAdapt.TCustomAdapter.UpdateFieldsExecuteValues 24177
WebAdapt.TCustomAdapter.UpdateFieldsGetAnyChanges 24177
WebAdapt.TCustomAdapter.UpdateFieldsUpdateValues 24178
WebAdapt.TCustomAdapter.UpdateFieldsValidateValues 24178
WebAdapt.TCustomAdapter.UpdateRecords 24179
WebAdapt.TCustomAdapter.UpdateRecordsGetAnyChanges 24179
WebAdapt.TCustomAdapter.UpdateRecordsUpdateValues 24180
WebAdapt.TCustomAdapter.UpdateRecordsValidateValues 24180
WebAdapt.TCustomAdapter.ViewAccess 24183
WebAdapt.TCustomAdapter.VisibleActions 24184
WebAdapt.TCustomAdapter.VisibleFields 24184
WebAdapt.TCustomAdapter.WebComponentsChange 24180
WebAdapt.TCustomAdapterAction 24188
WebAdapt.TCustomAdapterAction.ActionName 24198
WebAdapt.TCustomAdapterAction.ImplExecuteActionRequest 24195
WebAdapt.TCustomAdapterAction.ImplGetActionName 24195
WebAdapt.TCustomAdapterAction.ImplGetAdapterItemRequestParams
24196

hhhh

WebAdapt.TCustomAdapterAction.ImplGetSuccessOptions 24196
WebAdapt.TCustomAdapterAction.ImplSetActionName 24197
WebAdapt.TCustomAdapterAction.ImplWebEnabled 24197
WebAdapt.TCustomAdapterAction.OnAfterExecute 24198
WebAdapt.TCustomAdapterAction.OnAfterGetResponse 24199
WebAdapt.TCustomAdapterAction.OnBeforeExecute 24199
WebAdapt.TCustomAdapterAction.OnBeforeGetResponse 24200
WebAdapt.TCustomAdapterAction.OnExecute 24200
WebAdapt.TCustomAdapterAction.OnGetEnabled 24200
WebAdapt.TCustomAdapterAction.SetActionName 24197
WebAdapt.TCustomAdapterBooleanField 24201
WebAdapt.TCustomAdapterBooleanField.ConvertValue 24208
WebAdapt.TCustomAdapterBooleanField.GetDisplayStyleType 24208
WebAdapt.TCustomAdapterBooleanField.GetInputStyleType 24208
WebAdapt.TCustomAdapterBooleanField.ImplGetValue 24209
WebAdapt.TCustomAdapterBooleanField.ImplIsEqual 24209
WebAdapt.TCustomAdapterBooleanField.ImplUpdateValue 24209
WebAdapt.TCustomAdapterBooleanField.IsEqual 24210
WebAdapt.TCustomAdapterBooleanField.OnGetValue 24210
WebAdapt.TCustomAdapterBooleanField.OnUpdateValue 24211
WebAdapt.TCustomAdapterField 24211
WebAdapt.TCustomAdapterField.DoOnValidateValue 24218
WebAdapt.TCustomAdapterField.ImplGetValue 24219
WebAdapt.TCustomAdapterField.ImplUpdateValue 24219
WebAdapt.TCustomAdapterField.OnGetValue 24220
WebAdapt.TCustomAdapterField.OnUpdateValue 24220
WebAdapt.TCustomAdapterFileField 24220
WebAdapt.TCustomAdapterFileField.DoOnValidateValue 24228
WebAdapt.TCustomAdapterFileField.FileCount 24231
WebAdapt.TCustomAdapterFileField.FileNames 24231
WebAdapt.TCustomAdapterFileField.GetDisplayStyleType 24228
WebAdapt.TCustomAdapterFileField.GetFileNames 24229
WebAdapt.TCustomAdapterFileField.GetInputStyleType 24229
WebAdapt.TCustomAdapterFileField.GetValueCount 24229
WebAdapt.TCustomAdapterFileField.GetValues 24230
WebAdapt.TCustomAdapterFileField.ImplCheckValueChange 24230
WebAdapt.TCustomAdapterFileField.ImplUpdateValue 24230
WebAdapt.TCustomAdapterFileField.OnGetFileCount 24232
WebAdapt.TCustomAdapterFileField.OnGetFileNames 24232
WebAdapt.TCustomAdapterFileField.OnUploadFiles 24233
WebAdapt.TCustomAdapterFileField.OnValidateFiles 24233
WebAdapt.TCustomAdapterGotoPageAction 24233
WebAdapt.TCustomAdapterGotoPageAction.GetDefaultActionName 24240
WebAdapt.TCustomAdapterGotoPageAction.GetDisplayStyleType 24241
WebAdapt.TCustomAdapterGotoPageAction.GetItemCount 24241
WebAdapt.TCustomAdapterGotoPageAction.ImplExecuteActionRequest 24241
WebAdapt.TCustomAdapterGotoPageAction.ImplGetAdapterItemRequestParams 24242
WebAdapt.TCustomAdapterGotoPageAction.ImplGetDisplayLabel 24242
WebAdapt.TCustomAdapterGotoPageAction.ImplGetIsAdapterActionList 24243
WebAdapt.TCustomAdapterGotoPageAction.ImplIsDefaultAction 24243
WebAdapt.TCustomAdapterGotoPageAction.ImplWebEnabled 24243
WebAdapt.TCustomAdapterGotoPageAction.OnGetPageName 24244
WebAdapt.TCustomAdapterImageField 24244
WebAdapt.TCustomAdapterImageField.CreateRequestContext 24252
WebAdapt.TCustomAdapterImageField.DoOnValidateValue 24253
WebAdapt.TCustomAdapterImageField.GetAdapterImage 24253
WebAdapt.TCustomAdapterImageField.GetAdapterItemRequestParams 24253
WebAdapt.TCustomAdapterImageField.GetDisplayStyleType 24254
WebAdapt.TCustomAdapterImageField.GetInputStyleType

iiii

24254
WebAdapt.TCustomAdapterImageField.HandleRequest 24255
WebAdapt.TCustomAdapterImageField.HREF 24259
WebAdapt.TCustomAdapterImageField.ImplCheckValueChange 24255
WebAdapt.TCustomAdapterImageField.ImplCreateRequestContext 24255
WebAdapt.TCustomAdapterImageField.ImplGetValue 24256
WebAdapt.TCustomAdapterImageField.ImplHandleRequest 24256
WebAdapt.TCustomAdapterImageField.ImplUpdateValue 24257
WebAdapt.TCustomAdapterImageField.ImplWebImageHREF 24257
WebAdapt.TCustomAdapterImageField.OnGetHREF 24260
WebAdapt.TCustomAdapterImageField.OnGetImage 24260
WebAdapt.TCustomAdapterImageField.OnGetImageName 24260
WebAdapt.TCustomAdapterImageField.OnGetParams 24261
WebAdapt.TCustomAdapterImageField.OnUploadFile 24261
WebAdapt.TCustomAdapterImageField.OnValidateFile 24261
WebAdapt.TCustomAdapterImageField.ReadOnly 24259
WebAdapt.TCustomAdapterImageField.RenderAdapterImage 24258
WebAdapt.TCustomAdapterImageField.WebImageHREF 24258
WebAdapt.TCustomAdapterMemoField 24262
WebAdapt.TCustomAdapterMemoField.DoOnValidateValue 24269
WebAdapt.TCustomAdapterMemoField.GetDisplayStyleType 24270
WebAdapt.TCustomAdapterMemoField.GetInputStyleType 24270
WebAdapt.TCustomAdapterMemoField.ImplCheckValueChange 24270
WebAdapt.TCustomAdapterMemoField.ImplGetValue 24271
WebAdapt.TCustomAdapterMemoField.ImplUpdateValue 24271
WebAdapt.TCustomAdapterMemoField.OnGetValue 24272
WebAdapt.TCustomAdapterMemoField.OnUpdateValue 24272
WebAdapt.TCustomAdapterMemoField.OnUploadFile 24273
WebAdapt.TCustomAdapterMultiValueField 24273
WebAdapt.TCustomAdapterMultiValueField.DoOnValidateValue 24281
WebAdapt.TCustomAdapterMultiValueField.GetDisplayStyleType 24281
WebAdapt.TCustomAdapterMultiValueField.GetInputStyleType 24282
WebAdapt.TCustomAdapterMultiValueField.GetValueCount 24282
WebAdapt.TCustomAdapterMultiValueField.GetValues 24282
WebAdapt.TCustomAdapterMultiValueField.ImplHasValue 24283
WebAdapt.TCustomAdapterMultiValueField.ImplUpdateValue 24283
WebAdapt.TCustomAdapterMultiValueField.OnGetValueCount 24285
WebAdapt.TCustomAdapterMultiValueField.OnGetValues 24285
WebAdapt.TCustomAdapterMultiValueField.OnHasValue 24285
WebAdapt.TCustomAdapterMultiValueField.OnUpdateValues 24286
WebAdapt.TCustomAdapterMultiValueField.OnValidateValues 24286
WebAdapt.TCustomAdapterMultiValueField.ValueCount 24284
WebAdapt.TCustomAdapterMultiValueField.Values 24284
WebAdapt.TCustomAdapterPageAction 24286
WebAdapt.TCustomAdapterPageAction.GetDisplayStyleType 24293
WebAdapt.TCustomAdapterPageAction.GetNewPage 24294
WebAdapt.TCustomAdapterPageAction.ImplExecuteActionRequest 24294
WebAdapt.TCustomAdapterPageAction.ImplGetAdapterItemRequestParams 24294
WebAdapt.TCustomAdapterPageAction.ImplIsDefaultAction 24295
WebAdapt.TCustomAdapterPageAction.ImplWebEnabled 24295
WebAdapt.TCustomApplicationAdapter 24296
WebAdapt.TCustomApplicationAdapter.ApplicationTitle 24305
WebAdapt.TCustomApplicationAdapter.GetApplicationAdapter 24303
WebAdapt.TCustomApplicationAdapter.GetApplicationTitle 24303
WebAdapt.TCustomApplicationAdapter.GetWebSnapObjClass 24304
WebAdapt.TCustomApplicationAdapter.ImplCanAddActionClass 24304

jjjj

WebAdapt.TCustomApplicationAdapter.ImplCanAddFieldClass 24304
WebAdapt.TCustomApplicationAdapter.ImplGetActionsList 24305
WebAdapt.TCustomApplicationAdapter.ImplGetFieldsList 24305
WebAdapt.TCustomApplicationAdapterAction 24306
WebAdapt.TCustomApplicationAdapterAction.Adapter 24312
WebAdapt.TCustomEndUserAdapter 24313
WebAdapt.TCustomEndUserAdapter.ExecuteLogin 24321
WebAdapt.TCustomEndUserAdapter.ExecuteLogout 24322
WebAdapt.TCustomEndUserAdapter.GetDisplayNameValue 24322
WebAdapt.TCustomEndUserAdapter.GetLoggedInValue 24323
WebAdapt.TCustomEndUserAdapter.GetUserID 24323
WebAdapt.TCustomEndUserAdapter.OnGetDisplayName 24324
WebAdapt.TCustomEndUserAdapter.OnGetUserID 24324
WebAdapt.TCustomEndUserAdapter.OnIsLoggedIn 24324
WebAdapt.TCustomEndUserAdapter.OnLogin 24325
WebAdapt.TCustomEndUserAdapter.OnLogout 24326
WebAdapt.TCustomEndUserAdapterAction 24326
WebAdapt.TCustomEndUserAdapterAction.Adapter 24333
WebAdapt.TCustomEndUserSessionAdapter 24333
WebAdapt.TCustomEndUserSessionAdapter.ExecuteLogin 24342
WebAdapt.TCustomEndUserSessionAdapter.ExecuteLogout 24342
WebAdapt.TCustomEndUserSessionAdapter.GetDisplayNameV alue
24343
WebAdapt.TCustomEndUserSessionAdapter.GetLoggedInValue
24343
WebAdapt.TCustomEndUserSessionAdapter.GetUserID 24343
WebAdapt.TCustomEndUserSessionAdapter.UserID 24344
WebAdapt.TCustomLoginFormAdapter 24344
WebAdapt.TCustomLoginFormAdapter.EndUser 24353
WebAdapt.TCustomLoginFormAdapter.ExecuteLogin 24353
WebAdapt.TCustomLoginFormAdapter.GetNextPage 24354
WebAdapt.TCustomLoginFormAdapter.ImplCanAddActionClass 24354
WebAdapt.TCustomLoginFormAdapter.ImplCanAddFieldClass 24355
WebAdapt.TCustomLoginFormAdapter.ImplGetActionsList 24355
WebAdapt.TCustomLoginFormAdapter.ImplGetFieldsList 24356
WebAdapt.TCustomLoginFormAdapter.ImplNotifyDeactivate 24356
WebAdapt.TCustomLoginFormAdapter.NextPage 24359
WebAdapt.TCustomLoginFormAdapter.OnLogin 24361
WebAdapt.TCustomLoginFormAdapter.OnValidatePassword 24362
WebAdapt.TCustomLoginFormAdapter.OnValidateUserName 24362
WebAdapt.TCustomLoginFormAdapter.Password 24360
WebAdapt.TCustomLoginFormAdapter.PasswordRequired 24360
WebAdapt.TCustomLoginFormAdapter.RaiseBlankPassword 24357
WebAdapt.TCustomLoginFormAdapter.RaiseBlankUserName 24357
WebAdapt.TCustomLoginFormAdapter.UserList 24357
WebAdapt.TCustomLoginFormAdapter.UserName 24360
WebAdapt.TCustomLoginFormAdapter.ValidatePassword 24358
WebAdapt.TCustomLoginFormAdapter.ValidateUserName 24358
WebAdapt.TCustomLoginFormAdapterAction 24363
WebAdapt.TCustomLoginFormAdapterAction.Adapter 24369
WebAdapt.TCustomPagedAdapter 24370
WebAdapt.TCustomPagedAdapterAction 24371
WebAdapt.TCustomPagedAdapterAction.Adapter 24378
WebAdapt.TCustomPagedAdapterAction.GetAdapter 24377
WebAdapt.TCustomStringsValuesList 24378
WebAdapt.TCustomStringsValuesList.Create 24386
WebAdapt.TCustomStringsValuesList.Destroy 24386
WebAdapt.TCustomStringsValuesList.GetNameFieldClass 24387
WebAdapt.TCustomStringsValuesList.GetValueFieldClass 24387
WebAdapt.TCustomStringsValuesList.ImplEndIterator 24388
WebAdapt.TCustomStringsValuesList.ImplGetListName 24388
WebAdapt.TCustomStringsValuesList.ImplGetListNameOfValue 24388
WebAdapt.TCustomStringsValuesList.ImplGetListValue 24389
WebAdapt.TCustomStringsValuesList.ImplNextIteration 24389
WebAdapt.TCustomStringsValuesList.ImplNotifyDeactivate 24389
WebAdapt.TCustomStringsValuesList.ImplStartIterator 24390

kkkk

WebAdapt.TCustomStringsValuesList.OnPrepareStrings 24391
WebAdapt.TCustomStringsValuesList.PrepareStrings 24390
WebAdapt.TCustomStringsValuesList.Strings 24390
WebAdapt.TCustomStringsValuesList.StringsPrepared 24391
WebAdapt.TDefaultFieldsAdapter 24392
WebAdapt.TDefaultFieldsAdapter.Actions 24402
WebAdapt.TDefaultFieldsAdapter.AddDefaultActions 24399
WebAdapt.TDefaultFieldsAdapter.AddDefaultFields 24399
WebAdapt.TDefaultFieldsAdapter.Data 24402
WebAdapt.TDefaultFieldsAdapter.DefaultWebComponentsChange 24400
WebAdapt.TDefaultFieldsAdapter.GetActionsClass 24400
WebAdapt.TDefaultFieldsAdapter.GetDataClass 24400
WebAdapt.TDefaultFieldsAdapter.ImplGetVisibleActions 24401
WebAdapt.TDefaultFieldsAdapter.ImplGetVisibleFields 24401
WebAdapt.TDefaultFieldsAdapter.ImplSetActions 24401
WebAdapt.TDefaultFieldsAdapter.ImplSetData 24402
WebAdapt.TDefaultFieldsPagedAdapter 24403
WebAdapt.TDefaultFieldsPagedAdapter.AddDefaultActions 24411
WebAdapt.TDefaultFieldsPagedAdapter.CurrentPage 24418
WebAdapt.TDefaultFieldsPagedAdapter.EncodeActionParams 24411
WebAdapt.TDefaultFieldsPagedAdapter.EndOfPage 24412
WebAdapt.TDefaultFieldsPagedAdapter.ExtractRequestParams 24412
WebAdapt.TDefaultFieldsPagedAdapter.GetEOF 24412
WebAdapt.TDefaultFieldsPagedAdapter.GetFirstRecord 24413
WebAdapt.TDefaultFieldsPagedAdapter.GetNextRecord 24413
WebAdapt.TDefaultFieldsPagedAdapter.GetPageCount 24413
WebAdapt.TDefaultFieldsPagedAdapter.GetRecordCount 24414
WebAdapt.TDefaultFieldsPagedAdapter.GetRecordIndex 24414
WebAdapt.TDefaultFieldsPagedAdapter.ImplCanAddActionClass 24415
WebAdapt.TDefaultFieldsPagedAdapter.ImplEndIterator 24415
WebAdapt.TDefaultFieldsPagedAdapter.ImplGetActionsList 24415
WebAdapt.TDefaultFieldsPagedAdapter.ImplNextIteration 24416
WebAdapt.TDefaultFieldsPagedAdapter.ImplNotifyDeactivate 24416
WebAdapt.TDefaultFieldsPagedAdapter.ImplStartIterator 24416
WebAdapt.TDefaultFieldsPagedAdapter.OnGetEOF 24420
WebAdapt.TDefaultFieldsPagedAdapter.OnGetFirstRecord 24420
WebAdapt.TDefaultFieldsPagedAdapter.OnGetNextRecord 24421
WebAdapt.TDefaultFieldsPagedAdapter.OnGetRecordCount 24421
WebAdapt.TDefaultFieldsPagedAdapter.OnGetRecordIndex 24421
WebAdapt.TDefaultFieldsPagedAdapter.PageCount 24418
WebAdapt.TDefaultFieldsPagedAdapter.PageSize 24419
WebAdapt.TDefaultFieldsPagedAdapter.RecordCount 24419
WebAdapt.TDefaultFieldsPagedAdapter.RecordIndex 24419
WebAdapt.TDefaultFieldsPagedAdapter.SetPageSize 24417
WebAdapt.TDefaultFieldsPagedAdapter.SkipToPage 24417
WebAdapt.TEndUserAdapter 24421
WebAdapt.TEndUserAdapter.Actions 24431
WebAdapt.TEndUserAdapter.Data 24431
WebAdapt.TEndUserAdapter.LoginPage 24431
WebAdapt.TEndUserAdapter.OnAfterExecuteAction 24432
WebAdapt.TEndUserAdapter.OnAfterGetActionResponse 24432
WebAdapt.TEndUserAdapter.OnBeforeExecuteAction 24433
WebAdapt.TEndUserAdapter.OnBeforeGetActionResponse 24433
WebAdapt.TEndUserAdapter.OnGetActionParams 24433
WebAdapt.TEndUserAdapter.OnGetDisplayName 24434
WebAdapt.TEndUserAdapter.OnHasRights 24434
WebAdapt.TEndUserAdapter.OnIsLoggedIn 24434
WebAdapt.TEndUserAdapter.OnLogin 24435
WebAdapt.TEndUserAdapter.OnLogout 24436
WebAdapt.TEndUserAdapter.OnProduceLoginPage 24436
WebAdapt.TEndUserAdapterLoginFormAction 24436
WebAdapt.TEndUserAdapterLoginFormAction.DisplayLabel 24444
WebAdapt.TEndUserAdapterLoginFormAction.GetDefaultActionName 24443
WebAdapt.TEndUserAdapterLoginFormAction.ImplExecuteActionRequest 24443
WebAdapt.TEndUserAdapterLoginFormAction.ImplWebEnabled 24444

WebAdapt.TEndUserAdapterLoginFormAction.OnAfterExecute 24445
WebAdapt.TEndUserAdapterLoginFormAction.OnAfterGetResponse 24445
WebAdapt.TEndUserAdapterLoginFormAction.OnBeforeExecute 24446
WebAdapt.TEndUserAdapterLoginFormAction.OnBeforeGetResponse 24446
WebAdapt.TEndUserAdapterLoginFormAction.OnGetParams 24447
WebAdapt.TEndUserAdapterLogoutAction 24447
WebAdapt.TEndUserAdapterLogoutAction.DisplayLabel 24455
WebAdapt.TEndUserAdapterLogoutAction.GetDefaultActionName 24454
WebAdapt.TEndUserAdapterLogoutAction.ImplExecuteActionRequest 24454
WebAdapt.TEndUserAdapterLogoutAction.ImplWebEnabled 24455
WebAdapt.TEndUserAdapterLogoutAction.OnAfterExecute 24456
WebAdapt.TEndUserAdapterLogoutAction.OnAfterGetResponse 24456
WebAdapt.TEndUserAdapterLogoutAction.OnBeforeExecute 24457
WebAdapt.TEndUserAdapterLogoutAction.OnBeforeGetResponse 24457
WebAdapt.TEndUserAdapterLogoutAction.OnGetParams 24458
WebAdapt.TEndUserGetNameEvent 24605
WebAdapt.TEndUserGetNameEvent type 24605
WebAdapt.TEndUserGetUserIDEvent 24606
WebAdapt.TEndUserGetUserIDEvent type 24606
WebAdapt.TEndUserHandledEvent 24606
WebAdapt.TEndUserHandledEvent type 24606
WebAdapt.TEndUserIsLoggedInEvent 24606
WebAdapt.TEndUserIsLoggedInEvent type 24606
WebAdapt.TEndUserRightsEvent 24607
WebAdapt.TEndUserRightsEvent type 24607
WebAdapt.TEndUserSessionAdapter 24458
WebAdapt.TEndUserSessionAdapter.Actions 24468
WebAdapt.TEndUserSessionAdapter.Data 24468
WebAdapt.TEndUserSessionAdapter.LoginPage 24468
WebAdapt.TEndUserSessionAdapter.OnAfterExecuteAction 24469
WebAdapt.TEndUserSessionAdapter.OnAfterGetActionResponse 24469
WebAdapt.TEndUserSessionAdapter.OnBeforeExecuteAction 24470
WebAdapt.TEndUserSessionAdapter.OnBeforeGetActionResponse 24470
WebAdapt.TEndUserSessionAdapter.OnGetActionParams 24470
WebAdapt.TEndUserSessionAdapter.OnHasRights 24471
WebAdapt.TEndUserSessionAdapter.OnProduceLoginPage 24471
WebAdapt.TEndUserUserIDEvent 24607
WebAdapt.TEndUserUserIDEvent type 24607
WebAdapt.TFieldValuesAdapter 24471
WebAdapt.TFieldValuesAdapter.Create 24479
WebAdapt.TFieldValuesAdapter.Field 24483
WebAdapt.TFieldValuesAdapter.GetValueFieldClass 24480
WebAdapt.TFieldValuesAdapter.ImplEndIterator 24480
WebAdapt.TFieldValuesAdapter.ImplGetField 24481
WebAdapt.TFieldValuesAdapter.ImplGetValueIndex 24481
WebAdapt.TFieldValuesAdapter.ImplNextIteration 24481
WebAdapt.TFieldValuesAdapter.ImplStartIterator 24482
WebAdapt.TFieldValuesValueField 24483
WebAdapt.TFieldValuesValueField.Field 24490
WebAdapt.TFieldValuesValueField.GetDefaultFieldName 24489
WebAdapt.TFieldValuesValueField.ImplAsFormatted 24489
WebAdapt.TFieldValuesValueField.ImplGetValue 24490
WebAdapt.TGetEOFEvent 24607
WebAdapt.TGetEOFEvent type 24607
WebAdapt.TGetRecordCountEvent 24608
WebAdapt.TGetRecordCountEvent type 24608
WebAdapt.TGetRecordEvent 24608
WebAdapt.TGetRecordEvent type 24608
WebAdapt.TGetRecordIndexEvent 24608
WebAdapt.TGetRecordIndexEvent type 24608
WebAdapt.THandledEvent 24609

WebAdapt.THandledEvent type 24609
WebAdapt.TImageFieldGetHREFEvent 24609
WebAdapt.TImageFieldGetHREFEvent type 24609
WebAdapt.TImageFieldGetImageEvent 24609
WebAdapt.TImageFieldGetImageEvent type 24609
WebAdapt.TImageFieldGetParamsEvent 24610
WebAdapt.TImageFieldGetParamsEvent type 24610
WebAdapt.TImplementedAdapterAction 24491
WebAdapt.TIterateRecordsEvent 24610
WebAdapt.TIterateRecordsEvent type 24610
WebAdapt.TIteratorMethod 24593
WebAdapt.TIteratorMethod enumeration 24593
WebAdapt.TLogicalPageList 24492
WebAdapt.TLogicalPageList.Create 24500
WebAdapt.TLoginFormAdapter 24500
WebAdapt.TLoginFormAdapter.Actions 24509
WebAdapt.TLoginFormAdapter.Data 24509
WebAdapt.TLoginFormAdapter.OnAfterExecuteAction 24510
WebAdapt.TLoginFormAdapter.OnAfterGetActionResponse 24511
WebAdapt.TLoginFormAdapter.OnBeforeExecuteAction 24511
WebAdapt.TLoginFormAdapter.OnBeforeGetActionResponse 24511
WebAdapt.TLoginFormAdapter.OnGetActionParams 24512
WebAdapt.TLoginFormAdapter.OnLogin 24512
WebAdapt.TLoginFormAdapter.OnValidatePassword 24513
WebAdapt.TLoginFormAdapter.OnValidateUserName 24513
WebAdapt.TLoginFormAdapter.PasswordRequired 24509
WebAdapt.TLoginFormAdapterLoginAction 24514
WebAdapt.TLoginFormAdapterLoginAction.DisplayLabel 24522
WebAdapt.TLoginFormAdapterLoginAction.GetDefaultActionName 24520
WebAdapt.TLoginFormAdapterLoginAction.ImplExecuteActionRequest 24521
WebAdapt.TLoginFormAdapterLoginAction.ImplGetAdapterItemRequestParams 24521
WebAdapt.TLoginFormAdapterLoginAction.NextPage 24522
WebAdapt.TLoginFormAdapterLoginAction.OnAfterExecute 24523
WebAdapt.TLoginFormAdapterLoginAction.OnAfterGetResponse 24523
WebAdapt.TLoginFormAdapterLoginAction.OnBeforeExecute 24524
WebAdapt.TLoginFormAdapterLoginAction.OnBeforeGetResponse 24524
WebAdapt.TLoginFormAdapterLoginAction.OnGetParams 24525
WebAdapt.TLoginUserIDEvent 24610
WebAdapt.TLoginUserIDEvent type 24610
WebAdapt.TObjectList 24611
WebAdapt.TObjectList type 24611
WebAdapt.TPagedAdapter 24525
WebAdapt.TPagedAdapter.Actions 24533
WebAdapt.TPagedAdapter.Data 24534
WebAdapt.TPagedAdapter.ModifyAccess 24534
WebAdapt.TPagedAdapter.OnAfterExecuteAction 24536
WebAdapt.TPagedAdapter.OnAfterGetActionResponse 24536
WebAdapt.TPagedAdapter.OnAfterUpdateFields 24536
WebAdapt.TPagedAdapter.OnAfterValidateFields 24537
WebAdapt.TPagedAdapter.OnBeforeExecuteAction 24537
WebAdapt.TPagedAdapter.OnBeforeGetActionResponse 24537
WebAdapt.TPagedAdapter.OnBeforeUpdateFields 24538
WebAdapt.TPagedAdapter.OnBeforeValidateFields 24538
WebAdapt.TPagedAdapter.OnGetActionParams 24539
WebAdapt.TPagedAdapter.OnGetEOF 24539
WebAdapt.TPagedAdapter.OnGetFirstRecord 24539
WebAdapt.TPagedAdapter.OnGetNextRecord 24539
WebAdapt.TPagedAdapter.OnGetRecordCount 24540
WebAdapt.TPagedAdapter.OnGetRecordIndex 24540
WebAdapt.TPagedAdapter.PageSize 24534
WebAdapt.TPagedAdapter.ViewAccess 24535
WebAdapt.TStringsValuesList 24540
WebAdapt.TStringsValuesList.Actions 24549
WebAdapt.TStringsValuesList.Data 24549
WebAdapt.TStringsValuesList.OnPrepareStrings 24550
WebAdapt.TStringsValuesList.Strings 24549
WebAdapt.TUpdateFileList 24550
WebAdapt.TUpdateFileList.Count 24551

nnnn

WebAdapt.TUpdateFileList.Create 24551
WebAdapt.TUpdateFileList.Files 24552
WebAdapt.TValidateAdapterFieldEvent 24611
WebAdapt.TValidateAdapterFieldEvent type 24611
WebAdapt.TValuesListNameField 24552
WebAdapt.TValuesListNameField.GetDefaultFieldName 24557
WebAdapt.TValuesListNameField.ImplGetValue 24558
WebAdapt.TValuesListValueField 24558
WebAdapt.TValuesListValueField.GetDefaultFieldName 24564
WebAdapt.TValuesListValueField.ImplGetValue 24564
WebAdapt.TWebFormGetPageNameEvent 24611
WebAdapt.TWebFormGetPageNameEvent type 24611
WebBroker 24617
WebBroker namespace 24617
    Classes 24618
    Types 24625
    Variables 24626
WebBroker.Application 24626
WebBroker.Application variable 24626
WebBroker.HandleShutdownException 24626
WebBroker.HandleShutdownException variable 24626
WebBroker.THandleShutdownException 24625
WebBroker.THandleShutdownException type 24625
WebBroker.TServerExceptionEvent 24625
WebBroker.TServerExceptionEvent type 24625
WebBroker.TWebApplication 24618
WebBroker.TWebApplication.Create 24622
WebBroker.TWebApplication.CreateForm 24623
WebBroker.TWebApplication.Destroy 24622
WebBroker.TWebApplication.Initialize 24623
WebBroker.TWebApplication.Run 24623
WebBroker.TWebApplication.Title 24624
WebBrokerSOAP 24627
WebBrokerSOAP namespace 24627
    Classes 24628
    Functions 24636
WebBrokerSOAP.GetSOAPWebModule 24636
WebBrokerSOAP.GetSOAPWebModule function 24636
WebBrokerSOAP.THTTPSoapDispatcher 24628
WebBrokerSOAP.THTTPSoapDispatcher.Create 24631
WebBrokerSOAP.THTTPSoapDispatcher.Destroy 24632
WebBrokerSOAP.THTTPSoapDispatcher.DispatchEnabled 24632
WebBrokerSOAP.THTTPSoapDispatcher.DispatchMask 24633
WebBrokerSOAP.THTTPSoapDispatcher.DispatchMethodType 24633
WebBrokerSOAP.THTTPSoapDispatcher.DispatchRequest 24634
WebBrokerSOAP.THTTPSoapDispatcher.WebDispatch 24634
WebCntxt 24637
WebCntxt namespace 24637
    Classes 24639
    Functions 24687
    Interfaces 24672
    Structs, Records, Enums 24688
    Types 24689
    Variables 24691
WebCntxt.GetModuleFileNameProc 24691
WebCntxt.GetModuleFileNameProc variable 24691
WebCntxt.GetWebContextProc 24691
WebCntxt.GetWebContextProc variable 24691
WebCntxt.GetWebEndUserProc 24692
WebCntxt.GetWebEndUserProc variable 24692
WebCntxt.IWebEndUser 24672
WebCntxt.IWebEndUser.DisplayName 24678
WebCntxt.IWebEndUser.EndUserAdapter 24678
WebCntxt.IWebEndUser.GetDisplayName 24673
WebCntxt.IWebEndUser.GetEndUserAdapter 24674
WebCntxt.IWebEndUser.GetLoggedIn 24674
WebCntxt.IWebEndUser.GetLoginFormAction 24674
WebCntxt.IWebEndUser.GetLogoutAction 24675
WebCntxt.IWebEndUser.HasExecuteRights 24675
WebCntxt.IWebEndUser.HasModifyRights 24676
WebCntxt.IWebEndUser.HasViewRights 24676
WebCntxt.IWebEndUser.LoggedIn 24678
WebCntxt.IWebEndUser.LogIn 24677
WebCntxt.IWebEndUser.LoginFormAction 24679
WebCntxt.IWebEndUser.LogoutAction 24680
WebCntxt.IWebEndUser.ProduceLoginPage 24677

WebCntxt.IWebPageInfo 24680
WebCntxt.IWebPageInfo.GetHasViewAccess 24681
WebCntxt.IWebPageInfo.GetLoginRequired 24682
WebCntxt.IWebPageInfo.GetPageDescription 24682
WebCntxt.IWebPageInfo.GetPageHREF 24682
WebCntxt.IWebPageInfo.GetPageIsPublished 24683
WebCntxt.IWebPageInfo.GetPageName 24683
WebCntxt.IWebPageInfo.GetPageTitle 24683
WebCntxt.IWebPageInfo.HasViewAccess 24684
WebCntxt.IWebPageInfo.LoginRequired 24684
WebCntxt.IWebPageInfo.PageDescription 24684
WebCntxt.IWebPageInfo.PageHREF 24685
WebCntxt.IWebPageInfo.PageIsPublished 24685
WebCntxt.IWebPageInfo.PageName 24686
WebCntxt.IWebPageInfo.PageTitle 24686
WebCntxt.SetWebContextProc 24692
WebCntxt.SetWebContextProc variable 24692
WebCntxt.TAbstractAdapterRequest 24639
WebCntxt.TAbstractAdapterRequest.Create 24641
WebCntxt.TAbstractAdapterRequest.Request 24642
WebCntxt.TAbstractAdapterResponse 24642
WebCntxt.TAbstractAdapterResponse.Create 24644
WebCntxt.TAbstractAdapterResponse.Response 24644
WebCntxt.TAbstractWebContext 24644
WebCntxt.TAbstractWebContext.AdapterRequest 24655
WebCntxt.TAbstractWebContext.AdapterResponse 24656
WebCntxt.TAbstractWebContext.Create 24646
WebCntxt.TAbstractWebContext.Destroy 24647
WebCntxt.TAbstractWebContext.DispatchedPageName 24656
WebCntxt.TAbstractWebContext.EndUser 24657
WebCntxt.TAbstractWebContext.FindApplicationModule 24648
WebCntxt.TAbstractWebContext.FindDefaultPageInfo 24648
WebCntxt.TAbstractWebContext.FindModuleClass 24649
WebCntxt.TAbstractWebContext.FindModuleName 24649
WebCntxt.TAbstractWebContext.FindPage 24650
WebCntxt.TAbstractWebContext.FindPageInfo 24650
WebCntxt.TAbstractWebContext.FindVariableContainer 24651
WebCntxt.TAbstractWebContext.GetAdapterRequest 24651
WebCntxt.TAbstractWebContext.GetAdapterResponse 24652
WebCntxt.TAbstractWebContext.GetDispatchedPageName 24652
WebCntxt.TAbstractWebContext.GetEndUser 24652
WebCntxt.TAbstractWebContext.GetPageCount 24652
WebCntxt.TAbstractWebContext.GetPageInfo 24653
WebCntxt.TAbstractWebContext.GetRequestedPageName 24653
WebCntxt.TAbstractWebContext.GetSession 24653
WebCntxt.TAbstractWebContext.GetWebModules 24653
WebCntxt.TAbstractWebContext.GetWebRequest 24654
WebCntxt.TAbstractWebContext.GetWebResponse 24654
WebCntxt.TAbstractWebContext.PageCount 24657
WebCntxt.TAbstractWebContext.Pages 24658
WebCntxt.TAbstractWebContext.Request 24658
WebCntxt.TAbstractWebContext.RequestedPageName 24658
WebCntxt.TAbstractWebContext.Response 24659
WebCntxt.TAbstractWebContext.Session 24659
WebCntxt.TAbstractWebContext.SetAdapterRequest 24654
WebCntxt.TAbstractWebContext.SetAdapterResponse 24654
WebCntxt.TAbstractWebContext.SetDispatchedPageName 24655
WebCntxt.TAbstractWebContext.SetRequestedPageName 24655
WebCntxt.TAbstractWebContext.WebModules 24660
WebCntxt.TFindPageOption 24688
WebCntxt.TFindPageOption enumeration 24688
WebCntxt.TFindPageOptions 24689
WebCntxt.TFindPageOptions type 24689
WebCntxt.TGetWebContextProc 24689
WebCntxt.TGetWebContextProc type 24689
WebCntxt.TGetWebEndUserProc 24690
WebCntxt.TGetWebEndUserProc type 24690
WebCntxt.TSetWebContextProc 24690
WebCntxt.TSetWebContextProc type 24690
WebCntxt.TWebContext 24660
WebCntxt.TWebContext.Create 24662
WebCntxt.TWebContext.Destroy 24663
WebCntxt.TWebContext.FindApplicationModule 24664
WebCntxt.TWebContext.FindDefaultPageInfo 24664
WebCntxt.TWebContext.FindModuleClass 24665
WebCntxt.TWebContext.FindModuleName 24665

WebCntxt.TWebContext.FindPage 24666
WebCntxt.TWebContext.FindPageInfo 24666
WebCntxt.TWebContext.FindVariableContainer 24667
WebCntxt.TWebContext.GetAdapterRequest 24667
WebCntxt.TWebContext.GetAdapterResponse 24668
WebCntxt.TWebContext.GetDispatchedPageName 24668
WebCntxt.TWebContext.GetEndUser 24668
WebCntxt.TWebContext.GetPageCount 24668
WebCntxt.TWebContext.GetPageInfo 24669
WebCntxt.TWebContext.GetRequestedPageName 24669
WebCntxt.TWebContext.GetSession 24669
WebCntxt.TWebContext.GetWebModules 24669
WebCntxt.TWebContext.GetWebRequest 24670
WebCntxt.TWebContext.GetWebResponse 24670
WebCntxt.TWebContext.SetAdapterRequest 24670
WebCntxt.TWebContext.SetAdapterResponse 24670
WebCntxt.TWebContext.SetDispatchedPageName 24671
WebCntxt.TWebContext.SetRequestedPageName 24671
WebCntxt.WebContext 24687
WebCntxt.WebContext function 24687
WebComp 24693
WebComp namespace 24693
- Classes 24695
- Constants 24804
- Functions 24796
- Interfaces 24769
- Structs, Records, Enums 24799
- Types 24801

WebComp.DesigningComponent 24796
WebComp.DesigningComponent function 24796
WebComp.EnumRegisteredWebComponents 24796
WebComp.EnumRegisteredWebComponents function 24796
WebComp.EnumRegisteredWebComponentsProc 24804
WebComp.EnumRegisteredWebComponentsProc constant 24804
WebComp.FindWebComponentEditorHelper 24797
WebComp.FindWebComponentEditorHelper function 24797
WebComp.FindWebComponentEditorHelperProc 24804
WebComp.FindWebComponentEditorHelperProc constant 24804
WebComp.IAddFieldsEditor 24769
WebComp.IAddFieldsEditor.GetAddAllFieldsItem 24770
WebComp.IAddFieldsEditor.GetAddFieldsDlgCaption 24770
WebComp.IAddFieldsEditor.GetAddFieldsItem 24770
WebComp.IAddFieldsEditor.GetNewFieldPrefix 24771
WebComp.IGetWebComponentList 24771
WebComp.IGetWebComponentList.GetComponentList 24772
WebComp.IGetWebComponentList.GetDefaultComponentList 24772
WebComp.IGetWebComponentNodeName 24772
WebComp.IGetWebComponentNodeName.GetWebComponent NodeName 24773
WebComp.IGetXMLStream 24773
WebComp.IGetXMLStream.GetXMLStream 24774
WebComp.ILayoutWebContent 24774
WebComp.ILayoutWebContent.LayoutButton 24775
WebComp.ILayoutWebContent.LayoutField 24775
WebComp.ILayoutWebContent.LayoutLabelAndField 24775
WebComp.ILayoutWebContent.LayoutTable 24776
WebComp.INotifyActivateModule 24776
WebComp.INotifyActivateModule.NotifyActivateModule 24777
WebComp.INotifyActivateModule.NotifyDeactivateModule 24777
WebComp.INotifyWebActivate 24777
WebComp.INotifyWebActivate.NotifyActivate 24778
WebComp.INotifyWebActivate.NotifyDeactivate 24778
WebComp.IProducerEditorViewSupport 24779
WebComp.IProducerEditorViewSupport.GetTemplateFileType 24780
WebComp.IProducerEditorViewSupport.GetXMLData 24780
WebComp.IProducerEditorViewSupport.GetXSLData 24780
WebComp.IProducerEditorViewSupport.HasHTMLBrowserView 24780
WebComp.IProducerEditorViewSupport.HasScriptView 24780
WebComp.IProducerEditorViewSupport.HasXMLBrowserView 24781
WebComp.IProducerEditorViewSupport.HasXSLBrowserView 24781
WebComp.ISetWebContentOptions 24781
WebComp.ISetWebContentOptions.SetWebContentOptions 24782
WebComp.ITopLevelWebComponent 24782

qqqq

WebComp.IWebComponent 24782
WebComp.IWebComponent.ComponentList 24785
WebComp.IWebComponent.Container 24786
WebComp.IWebComponent.GetContainer 24784
WebComp.IWebComponent.GetIndex 24784
WebComp.IWebComponent.Index 24786
WebComp.IWebComponent.SetComponentList 24784
WebComp.IWebComponent.SetContainer 24784
WebComp.IWebComponent.SetIndex 24785
WebComp.IWebComponentCommands 24786
WebComp.IWebComponentCommands.ExecuteVerb 24787
WebComp.IWebComponentCommands.GetVerb 24787
WebComp.IWebComponentCommands.GetVerbCount 24788
WebComp.IWebComponentContainer 24788
WebComp.IWebComponentContainer.ComponentCount 24789
WebComp.IWebComponentContainer.Components 24790
WebComp.IWebComponentContainer.GetComponent 24789
WebComp.IWebComponentContainer.GetComponentCount 24789
WebComp.IWebComponentEditor 24790
WebComp.IWebComponentEditor.CanAddClass 24791
WebComp.IWebComponentEditorHelper 24791
WebComp.IWebComponentEditorHelper.CanAddClassHelper 24792
WebComp.IWebContent 24792
WebComp.IWebContent.Content 24793
WebComp.IWebVariablesContainerName 24794
WebComp.IWebVariablesContainerName.ContainerName 24795
WebComp.IWebVariablesContainerName.GetContainerName 24794
WebComp.QualifyFileName 24797
WebComp.QualifyFileName function 24797
WebComp.RegisterWebComponents 24797
WebComp.RegisterWebComponents function 24797
WebComp.RegisterWebComponentsProc 24805
WebComp.RegisterWebComponentsProc constant 24805
WebComp.TEnumWebComponentsProc 24801
WebComp.TEnumWebComponentsProc type 24801
WebComp.TGetComponentListEvent 24801
WebComp.TGetComponentListEvent type 24801
WebComp.TLabelPosition 24799
WebComp.TLabelPosition enumeration 24799
WebComp.TLayout 24802
WebComp.TLayout type 24802
WebComp.TLayoutAttributes 24695
WebComp.TLayoutAttributes.ControlAttributes 24697
WebComp.TLayoutAttributes.Create 24696
WebComp.TLayoutAttributes.LabelAttributes 24697
WebComp.TLayoutAttributes.LabelPosition 24697
WebComp.TLayoutWebContent 24698
WebComp.TLayoutWebContent._AddRef 24700
WebComp.TLayoutWebContent._Release 24701
WebComp.TLayoutWebContent.Create 24700
WebComp.TLayoutWebContent.ImplLayoutButton 24701
WebComp.TLayoutWebContent.ImplLayoutField 24702
WebComp.TLayoutWebContent.ImplLayoutLabelAndField 24702
WebComp.TLayoutWebContent.ImplLayoutTable 24703
WebComp.TLayoutWebContent.LayoutButton 24703
WebComp.TLayoutWebContent.LayoutField 24704
WebComp.TLayoutWebContent.LayoutLabelAndField 24704
WebComp.TLayoutWebContent.LayoutTable 24705
WebComp.TLayoutWebContent.ParentLayout 24706
WebComp.TLayoutWebContent.QueryInterface 24705
WebComp.TSetComponentListEvent 24802
WebComp.TSetComponentListEvent type 24802
WebComp.TWebComponentClassDescription 24706
WebComp.TWebComponentClassDescription.ComponentClass 24708
WebComp.TWebComponentClassDescription.ComponentClass Name
24708
WebComp.TWebComponentClassDescription.Editor 24708
WebComp.TWebComponentClassDescription.FrameClass 24708
WebComp.TWebComponentContainer 24802
WebComp.TWebComponentContainer type 24802
WebComp.TWebComponentList 24708
WebComp.TWebComponentList._AddRef 24712
WebComp.TWebComponentList._Release 24712
WebComp.TWebComponentList.Add 24713

WebComp.TWebComponentList.Changed 24713
WebComp.TWebComponentList.Clear 24713
WebComp.TWebComponentList.Count 24718
WebComp.TWebComponentList.Create 24711
WebComp.TWebComponentList.Destroy 24711
WebComp.TWebComponentList.GetComponent 24714
WebComp.TWebComponentList.GetCount 24714
WebComp.TWebComponentList.GetOwner 24714
WebComp.TWebComponentList.IndexOf 24715
WebComp.TWebComponentList.LockNotify 24715
WebComp.TWebComponentList.NewComponent 24715
WebComp.TWebComponentList.NotifyLocked 24715
WebComp.TWebComponentList.OnChange 24719
WebComp.TWebComponentList.ParentComponent 24718
WebComp.TWebComponentList.QueryInterface 24716
WebComp.TWebComponentList.Remove 24716
WebComp.TWebComponentList.SetComponentIndex 24716
WebComp.TWebComponentList.UnlockNotify 24717
WebComp.TWebComponentList.WebComponentContainer_GetComponent 24717
WebComp.TWebComponentList.WebComponentContainer_GetComponentCount 24717
WebComp.TWebComponentList.WebComponents 24718
WebComp.TWebComponentsEditorHelper 24719
WebComp.TWebComponentsEditorHelper._AddRef 24721
WebComp.TWebComponentsEditorHelper._Release 24721
WebComp.TWebComponentsEditorHelper.CanAddClassHelper 24721
WebComp.TWebComponentsEditorHelper.ImplCanAddClassHelper 24721
WebComp.TWebComponentsEditorHelper.QueryInterface 24722
WebComp.TWebContainedComponent 24722
WebComp.TWebContainedComponent.Create 24726
WebComp.TWebContainedComponent.Destroy 24726
WebComp.TWebContainedComponent.GetContainerList 24727
WebComp.TWebContainedComponent.GetParentComponent 24727
WebComp.TWebContainedComponent.HasParent 24727
WebComp.TWebContainedComponent.IsDefaultField 24728
WebComp.TWebContainedComponent.ParentChanged 24728
WebComp.TWebContainedComponent.ReadState 24729
WebComp.TWebContainedComponent.SetParentComponent 24729
WebComp.TWebContainedComponent.SetWebParent 24730
WebComp.TWebContainedComponent.WebContainedComponentHelper 24730
WebComp.TWebContainedComponent.WebParent 24731
WebComp.TWebContainedComponentHelper 24731
WebComp.TWebContainedComponentHelper.ContainerList 24737
WebComp.TWebContainedComponentHelper.Create 24733
WebComp.TWebContainedComponentHelper.Destroy 24733
WebComp.TWebContainedComponentHelper.GetContainer 24734
WebComp.TWebContainedComponentHelper.GetIndex 24734
WebComp.TWebContainedComponentHelper.IsDefaultField 24735
WebComp.TWebContainedComponentHelper.OnParentChanged 24738
WebComp.TWebContainedComponentHelper.ParentChanged 24735
WebComp.TWebContainedComponentHelper.SetComponentList 24735
WebComp.TWebContainedComponentHelper.SetContainer 24736
WebComp.TWebContainedComponentHelper.SetIndex 24736
WebComp.TWebContainedComponentHelper.SetWebParent 24736
WebComp.TWebContainedComponentHelper.WebParent 24737
WebComp.TWebContainedContainerComponent 24738
WebComp.TWebContainedContainerComponent.Create 24742
WebComp.TWebContainedContainerComponent.Destroy 24743
WebComp.TWebContainedContainerComponent.GetChildren 24743
WebComp.TWebContainedContainerComponent.GetContainerList 24744
WebComp.TWebContainedContainerComponent.GetDefaultWebComponents 24744

ssssss

WebComp.TWebContainedContainerComponent.GetParentComponent
24745

WebComp.TWebContainedContainerComponent.GetWebComponents
24745

WebComp.TWebContainedContainerComponent.HasParent
24745

WebComp.TWebContainedContainerComponent.ImplGetDefaultComponentList
24746

WebComp.TWebContainedContainerComponent.IsDefaultField
24746

WebComp.TWebContainedContainerComponent.ParentChanged
24747

WebComp.TWebContainedContainerComponent.ReadState
24747

WebComp.TWebContainedContainerComponent.SetChildOrder
24748

WebComp.TWebContainedContainerComponent.SetDefaultWebComponents
24748

WebComp.TWebContainedContainerComponent.SetParentComponent
24748

WebComp.TWebContainedContainerComponent.SetWebParent

24749

WebComp.TWebContainedContainerComponent.WebContainedComponentHelper
24749

WebComp.TWebContainedContainerComponent.WebContainerComponentHelper
24750

WebComp.TWebContainedContainerComponent.WebParent
24750

WebComp.TWebContainerComponent 24751

WebComp.TWebContainerComponent.Create 24754

WebComp.TWebContainerComponent.Destroy 24755

WebComp.TWebContainerComponent.GetChildren 24755

WebComp.TWebContainerComponent.GetDefaultWebComponents
24756

WebComp.TWebContainerComponent.GetWebComponents
24756

WebComp.TWebContainerComponent.ImplGetDefaultComponentList
24756

WebComp.TWebContainerComponent.SetChildOrder 24757

WebComp.TWebContainerComponent.SetDefaultWebComponents
24757

WebComp.TWebContainerComponent.WebContainerComponentHelper
24758

WebComp.TWebContainerComponentHelper 24758

WebComp.TWebContainerComponentHelper.Create 24760

WebComp.TWebContainerComponentHelper.DefaultWebComponents
24764

WebComp.TWebContainerComponentHelper.Destroy 24760

WebComp.TWebContainerComponentHelper.GetChildren
24761

WebComp.TWebContainerComponentHelper.GetComponentList
24762

WebComp.TWebContainerComponentHelper.GetDefaultComponentList
24762

WebComp.TWebContainerComponentHelper.GetDefaultWebComponents
24762

WebComp.TWebContainerComponentHelper.OnGetDefaultComponentList
24765

WebComp.TWebContainerComponentHelper.OnSetDefaultComponentList
24765

WebComp.TWebContainerComponentHelper.SetChildOrder
24763

WebComp.TWebContainerComponentHelper.SetDefaultWebComponents
24763

WebComp.TWebContainerComponentHelper.Webcomponents
24764

WebComp.TWebContentFlag 24799

WebComp.TWebContentFlag enumeration 24799

WebComp.TWebContentFlags 24803

WebComp.TWebContentFlags type 24803

WebComp.TWebContentOptions 24766

WebComp.TWebContentOptions.Create 24767

WebComp.TWebContentOptions.Flags 24768

WebComp.UnRegisterWebComponents 24798

WebComp.UnRegisterWebComponents function 24798

WebComp.UnRegisterWebComponentsProc 24805

WebComp.UnRegisterWebComponentsProc constant 24805
WebComp.ValidateWebParent 24798
WebComp.ValidateWebParent function 24798
WebContnrs 24807
WebContnrs namespace 24807
Classes 24808
Structs, Records, Enums 24832
Types 24833
WebContnrs.ENamedVariantsError 24808
WebContnrs.PNamedVariantItem 24833
WebContnrs.PNamedVariantItem type 24833
WebContnrs.PNamedVariantList 24833
WebContnrs.PNamedVariantList type 24833
WebContnrs.TAbstractNamedVariants 24808
WebContnrs.TAbstractNamedVariants.Add 24812
WebContnrs.TAbstractNamedVariants.Append 24812
WebContnrs.TAbstractNamedVariants.Assign 24813
WebContnrs.TAbstractNamedVariants.BeginUpdate 24813
WebContnrs.TAbstractNamedVariants.Capacity 24821
WebContnrs.TAbstractNamedVariants.Clear 24813
WebContnrs.TAbstractNamedVariants.CompareStrings 24814
WebContnrs.TAbstractNamedVariants.Count 24822
WebContnrs.TAbstractNamedVariants.DefineProperties 24814
WebContnrs.TAbstractNamedVariants.Delete 24815
WebContnrs.TAbstractNamedVariants.Destroy 24810
WebContnrs.TAbstractNamedVariants.EndUpdate 24815
WebContnrs.TAbstractNamedVariants.Equals 24816
WebContnrs.TAbstractNamedVariants.Error 24816
WebContnrs.TAbstractNamedVariants.Exchange 24817
WebContnrs.TAbstractNamedVariants.Get 24817
WebContnrs.TAbstractNamedVariants.GetCapacity 24817
WebContnrs.TAbstractNamedVariants.GetCount 24818
WebContnrs.TAbstractNamedVariants.IndexOfName 24818
WebContnrs.TAbstractNamedVariants.Insert 24819
WebContnrs.TAbstractNamedVariants.Move 24819
WebContnrs.TAbstractNamedVariants.Names 24822
WebContnrs.TAbstractNamedVariants.Put 24820
WebContnrs.TAbstractNamedVariants.SetCapacity 24820
WebContnrs.TAbstractNamedVariants.SetUpdateState 24821
WebContnrs.TAbstractNamedVariants.UpdateCount 24823
WebContnrs.TAbstractNamedVariants.Values 24823
WebContnrs.TAbstractNamedVariants.Variants 24824
WebContnrs.TNamedVariantItem 24832
WebContnrs.TNamedVariantItem record 24832
WebContnrs.TNamedVariantList 24833
WebContnrs.TNamedVariantList type 24833
WebContnrs.TNamedVariantsList 24824
WebContnrs.TNamedVariantsList.Clear 24827
WebContnrs.TNamedVariantsList.Delete 24827
WebContnrs.TNamedVariantsList.Destroy 24826
WebContnrs.TNamedVariantsList.Get 24828
WebContnrs.TNamedVariantsList.GetCapacity 24828
WebContnrs.TNamedVariantsList.GetCount 24829
WebContnrs.TNamedVariantsList.Insert 24829
WebContnrs.TNamedVariantsList.Put 24830
WebContnrs.TNamedVariantsList.SetCapacity 24830
WebDisp 24835
WebDisp namespace 24835
Classes 24838
Constants 24991
Functions 24977
Interfaces 24975
Structs, Records, Enums 24983
Types 24984
WebDisp.AdapterRequestParamsAsStrings 24977
WebDisp.AdapterRequestParamsAsStrings function 24977
WebDisp.DispatchPageName 24977
WebDisp.DispatchPageName function 24977
WebDisp.EAdapterDispatcherException 24838
WebDisp.EncodeParamNameValues 24978
WebDisp.EncodeParamNameValues function 24978
WebDisp.EncodeParamsAsHTTPQuery 24978
WebDisp.EncodeParamsAsHTTPQuery function 24978
WebDisp.EPageDispatcherException 24839
WebDisp.ESessionException 24840
WebDisp.ExpandScriptException 24978
WebDisp.ExpandScriptException function 24978
WebDisp.ExtractParamNameValues 24979

WebDisp.ExtractParamNameValues function 24979
WebDisp.FindDefaultAppPageName 24979
WebDisp.FindDefaultAppPageName function 24979
WebDisp.IWebSessionAccess 24975
WebDisp.IWebSessionAccess.GetSessionsService 24976
WebDisp.IWebSessionAccess.GetWebSession 24976
WebDisp.PageNameOfRequest 24979
WebDisp.PageNameOfRequest function 24979
WebDisp.RaiseDispatchBlankPageNameException 24980
WebDisp.RaiseDispatchBlankPageNameException function 24980
WebDisp.RaisePageDoesNotProvideContent 24980
WebDisp.RaisePageDoesNotProvideContent function 24980
WebDisp.RaisePageNotFound 24980
WebDisp.RaisePageNotFound function 24980
WebDisp.RedirectToPageName 24981
WebDisp.RedirectToPageName function 24981
WebDisp.RegisterAdapterRequestIdentifier 24981
WebDisp.RegisterAdapterRequestIdentifier function 24981
WebDisp.SafeRedirect 24981
WebDisp.SafeRedirect function 24981
WebDisp.sFailurePageParamName 24991
WebDisp.sFailurePageParamName constant 24991
WebDisp.sPageParamName 24991
WebDisp.sPageParamName constant 24991
WebDisp.sRequestIDParamName 24992
WebDisp.sRequestIDParamName constant 24992
WebDisp.sSessionCookieName 24992
WebDisp.sSessionCookieName constant 24992
WebDisp.sSuccessPageParamName 24992
WebDisp.sSuccessPageParamName constant 24992
WebDisp.TActivateWebModuleEvent 24984
WebDisp.TActivateWebModuleEvent type 24984
WebDisp.TAdapterDispatcher 24840
WebDisp.TAdapterDispatcher.OnActionRequestNotHandled 24844
WebDisp.TAdapterDispatcher.OnAfterDispatchAction 24845
WebDisp.TAdapterDispatcher.OnAfterDispatchImage 24845
WebDisp.TAdapterDispatcher.OnBeforeDispatchAction 24846
WebDisp.TAdapterDispatcher.OnBeforeDispatchImage 24846
WebDisp.TAdapterDispatcher.OnGetAdapterRequestParams 24847
WebDisp.TAdapterDispatcher.OnImageRequestNotHandled 24847
WebDisp.TAdapterDispatchParams 24848
WebDisp.TAdapterDispatchParams.Create 24850
WebDisp.TAdapterDispatchParams.Destroy 24851
WebDisp.TAdapterDispatchParams.GetFailurePage 24851
WebDisp.TAdapterDispatchParams.GetHandler 24852
WebDisp.TAdapterDispatchParams.GetObjectIdentifier 24852
WebDisp.TAdapterDispatchParams.GetParams 24852
WebDisp.TAdapterDispatchParams.GetRequestIdentifier 24853
WebDisp.TAdapterDispatchParams.GetSuccessPage 24853
WebDisp.TAdapterDispatchParams.Handler 24853
WebDisp.TAdapterDispatchParams.ObjectIdentifier 24854
WebDisp.TAdapterDispatchParams.RequestIdentifier 24854
WebDisp.TAdapterItemRequestParams 24855
WebDisp.TAdapterItemRequestParams.Create 24857
WebDisp.TAdapterItemRequestParams.Destroy 24857
WebDisp.TAdapterItemRequestParams.FailurePage 24859
WebDisp.TAdapterItemRequestParams.GetFailurePage 24857
WebDisp.TAdapterItemRequestParams.GetParamValues 24858
WebDisp.TAdapterItemRequestParams.GetSuccessPage 24858
WebDisp.TAdapterItemRequestParams.ParamValues 24860
WebDisp.TAdapterItemRequestParams.SetFailurePage 24858
WebDisp.TAdapterItemRequestParams.SetSuccessPage 24859
WebDisp.TAdapterItemRequestParams.SuccessPage 24860
WebDisp.TBaseRequestHandler 24860
WebDisp.TBaseRequestHandler.AppDispatcher 24865
WebDisp.TBaseRequestHandler.HandleRequest 24864
WebDisp.TBaseRequestHandler.ImplHandleRequest 24864
WebDisp.TBaseRequestHandler.SetAppDispatcher 24864
WebDisp.TCanViewPageEvent 24984
WebDisp.TCanViewPageEvent type 24984
WebDisp.TCustomAdapterDispatcher 24865
WebDisp.TCustomAdapterDispatcher.AddSpecialParams 24870
WebDisp.TCustomAdapterDispatcher.DoAdapterRequestNotHandled
24870
WebDisp.TCustomAdapterDispatcher.DoAfterDispatchAdapterRequest

vvvv

24871

WebDisp.TCustomAdapterDispatcher.DoBeforeDispatchAdapterRequest
24871

WebDisp.TCustomAdapterDispatcher.EncodeAdapterItemRequestAsFieldValue
24872

WebDisp.TCustomAdapterDispatcher.EncodeAdapterItemRequestAsHREF
24872

WebDisp.TCustomAdapterDispatcher.GetAdapterItemRequestParams
24873

WebDisp.TCustomAdapterDispatcher.GetDispatchParams
24873

WebDisp.TCustomAdapterDispatcher.ImplEncodeAdapterItemRequestAsFieldValue
24874

WebDisp.TCustomAdapterDispatcher.ImplEncodeAdapterItemRequestAsHREF
24874

WebDisp.TCustomAdapterDispatcher.ImplGetAdapterItemRequestParams
24875

WebDisp.TCustomAdapterDispatcher.ImplHandleRequest
24876

WebDisp.TCustomAdapterDispatcher.OnActionRequestNotHandled
24877

WebDisp.TCustomAdapterDispatcher.OnAfterDispatchAction
24877

WebDisp.TCustomAdapterDispatcher.OnAfterDispatchImage
24878

WebDisp.TCustomAdapterDispatcher.OnBeforeDispatchAction
24878

WebDisp.TCustomAdapterDispatcher.OnBeforeDispatchImage
24878

WebDisp.TCustomAdapterDispatcher.OnGetAdapterRequestParams
24879

WebDisp.TCustomAdapterDispatcher.OnImageRequestNotHandled
24879

WebDisp.TCustomLocateFileService 24880

WebDisp.TCustomLocateFileService.FindIncludeFile 24884

WebDisp.TCustomLocateFileService.FindStream 24885

WebDisp.TCustomLocateFileService.FindTemplateFile 24885

WebDisp.TCustomLocateFileService.GetFileStream 24886

WebDisp.TCustomLocateFileService.GetTemplateStream
24886

WebDisp.TCustomLocateFileService.ImplFindIncludeFile 24887

WebDisp.TCustomLocateFileService.ImplGetTemplateStream
24887

WebDisp.TCustomLocateFileService.OnFindIncludeFile 24888

WebDisp.TCustomLocateFileService.OnFindStream 24889

WebDisp.TCustomLocateFileService.OnFindTemplateFile
24889

WebDisp.TCustomPageDispatcher 24889

WebDisp.TCustomPageDispatcher.DefaultPage 24900

WebDisp.TCustomPageDispatcher.DispatchPageName 24894

WebDisp.TCustomPageDispatcher.EndUser 24895

WebDisp.TCustomPageDispatcher.GetCanView 24895

WebDisp.TCustomPageDispatcher.GetDefaultPageName 24895

WebDisp.TCustomPageDispatcher.GetLoginRequired 24896

WebDisp.TCustomPageDispatcher.GetRequestedPage 24896

WebDisp.TCustomPageDispatcher.ImplDispatchPageName
24896

WebDisp.TCustomPageDispatcher.ImplGetCanView 24896

WebDisp.TCustomPageDispatcher.ImplGetDefaultPageName
24897

WebDisp.TCustomPageDispatcher.ImplGetLoginRequired
24897

WebDisp.TCustomPageDispatcher.ImplHandleRequest 24897

WebDisp.TCustomPageDispatcher.ImplPageNameOfRequest
24897

WebDisp.TCustomPageDispatcher.ImplRedirectToPageName
24898

WebDisp.TCustomPageDispatcher.IsLoggedIn 24898

WebDisp.TCustomPageDispatcher.OnAfterDispatchPage 24901

WebDisp.TCustomPageDispatcher.OnAfterRedirectToPage
24901

WebDisp.TCustomPageDispatcher.OnBeforeDispatchPage
24902

WebDisp.TCustomPageDispatcher.OnBeforeRedirectToPage
24902

WebDisp.TCustomPageDispatcher.OnCanViewPage 24903

WebDisp.TCustomPageDispatcher.OnIsLoginRequired 24904

WebDisp.TCustomPageDispatcher.OnPageAccessDenied
24904

WebDisp.TCustomPageDispatcher.OnPageNotFound 24905

WebDisp.TCustomPageDispatcher.PageNameOfRequest 24898

WebDisp.TCustomPageDispatcher.ProduceLoginPage 24899

wwww

WebDisp.TCustomPageDispatcher.RedirectToPageName 24899
WebDisp.TCustomPageDispatcher.ValidatePage 24899
WebDisp.TCustomWebAppComponents 24906
WebDisp.TCustomWebAppComponents.AdapterDispatcher 24919
WebDisp.TCustomWebAppComponents.ApplicationAdapter 24919
WebDisp.TCustomWebAppComponents.CreateContext 24911
WebDisp.TCustomWebAppComponents.DispatchActions 24919
WebDisp.TCustomWebAppComponents.EndUserAdapter 24920
WebDisp.TCustomWebAppComponents.GetAdapterDispatcher 24912
WebDisp.TCustomWebAppComponents.GetLocateFileService 24912
WebDisp.TCustomWebAppComponents.GetPageDispatcher 24912
WebDisp.TCustomWebAppComponents.GetWebApplicationInfo 24913
WebDisp.TCustomWebAppComponents.GetWebEndUser 24913
WebDisp.TCustomWebAppComponents.GetWebUserList 24913
WebDisp.TCustomWebAppComponents.ImplFinishContext 24914
WebDisp.TCustomWebAppComponents.ImplHandleException 24914
WebDisp.TCustomWebAppComponents.ImplHandleRequest 24915
WebDisp.TCustomWebAppComponents.ImplInitContext 24915
WebDisp.TCustomWebAppComponents.InitModule 24916
WebDisp.TCustomWebAppComponents.InitModules 24917
WebDisp.TCustomWebAppComponents.ListRequestHandlers 24917
WebDisp.TCustomWebAppComponents.LocateFileService 24920
WebDisp.TCustomWebAppComponents.Notification 24918
WebDisp.TCustomWebAppComponents.OnException 24922
WebDisp.TCustomWebAppComponents.PageDispatcher 24921
WebDisp.TCustomWebAppComponents.Sessions 24921
WebDisp.TCustomWebAppComponents.UserListService 24922
WebDisp.TDispatchActionEvent 24985
WebDisp.TDispatchActionEvent type 24985
WebDisp.TDispatchActionHandledEvent 24985
WebDisp.TDispatchActionHandledEvent type 24985
WebDisp.TDispatchImageEvent 24985
WebDisp.TDispatchImageEvent type 24985
WebDisp.TDispatchImageHandledEvent 24986
WebDisp.TDispatchImageHandledEvent type 24986
WebDisp.TDispatchPageEvent 24986
WebDisp.TDispatchPageEvent type 24986
WebDisp.TDispatchPageHandledEvent 24986
WebDisp.TDispatchPageHandledEvent type 24986
WebDisp.TDispatchPageParamsHandledEvent 24987
WebDisp.TDispatchPageParamsHandledEvent type 24987
WebDisp.TGetAdapterRequestParamsEvent 24987
WebDisp.TGetAdapterRequestParamsEvent type 24987
WebDisp.TInvalidWebSession 24923
WebDisp.TInvalidWebSession.Create 24924
WebDisp.TInvalidWebSession.GetSessionID 24925
WebDisp.TInvalidWebSession.GetTimeoutMinutes 24925
WebDisp.TInvalidWebSession.GetValue 24926
WebDisp.TInvalidWebSession.SetTimeoutMinutes 24926
WebDisp.TInvalidWebSession.SetValue 24927
WebDisp.TInvalidWebSession.Terminate 24927
WebDisp.TInvalidWebSession.UpdateResponse 24927
WebDisp.TLocateFileService 24928
WebDisp.TLocateFileService.OnFindIncludeFile 24932
WebDisp.TLocateFileService.OnFindStream 24932
WebDisp.TLocateFileService.OnFindTemplateFile 24932
WebDisp.TLocateFileServiceFindFileEvent 24987
WebDisp.TLocateFileServiceFindFileEvent type 24987
WebDisp.TLocateFileServiceFindStreamEvent 24988
WebDisp.TLocateFileServiceFindStreamEvent type 24988
WebDisp.TLoginPagedHandledEvent 24988
WebDisp.TLoginPagedHandledEvent type 24988
WebDisp.TLoginRequiredEvent 24988
WebDisp.TLoginRequiredEvent type 24988
WebDisp.TMultiModuleWebAppServices 24933
WebDisp.TMultiModuleWebAppServices.Access_Request 24937
WebDisp.TMultiModuleWebAppServices.Access_Response 24938
WebDisp.TMultiModuleWebAppServices.CreateContext 24938

WebDisp.TMultiModuleWebAppServices.DoAfterDispatch 24939
WebDisp.TMultiModuleWebAppServices.DoBeforeDispatch 24939
WebDisp.TMultiModuleWebAppServices.DoneModule 24939
WebDisp.TMultiModuleWebAppServices.FinishContext 24940
WebDisp.TMultiModuleWebAppServices.GetExceptionHandler 24940
WebDisp.TMultiModuleWebAppServices.HandleException 24941
WebDisp.TMultiModuleWebAppServices.HandleRequest 24941
WebDisp.TMultiModuleWebAppServices.ImplFinishContext 24942
WebDisp.TMultiModuleWebAppServices.ImplHandleException 24942
WebDisp.TMultiModuleWebAppServices.ImplHandleRequest 24943
WebDisp.TMultiModuleWebAppServices.ImplInitContext 24943
WebDisp.TMultiModuleWebAppServices.InitContext 24944
WebDisp.TMultiModuleWebAppServices.InitModule 24945
WebDisp.TMultiModuleWebAppServices.InitModules 24945
WebDisp.TMultiModuleWebAppServices.ModuleAdded 24946
WebDisp.TMultiModuleWebAppServices.OnAfterDispatch 24948
WebDisp.TMultiModuleWebAppServices.OnBeforeDispatch 24948
WebDisp.TMultiModuleWebAppServices.Request 24946
WebDisp.TMultiModuleWebAppServices.Response 24947
WebDisp.TMultiModuleWebAppServices.WebModules 24947
WebDisp.TPageAccessDenied 24983
WebDisp.TPageAccessDenied enumeration 24983
WebDisp.TPageAccessDeniedEvent 24989
WebDisp.TPageAccessDeniedEvent type 24989
WebDisp.TPageDispatcher 24949
WebDisp.TPageDispatcher.DefaultPage 24953
WebDisp.TPageDispatcher.OnAfterDispatchPage 24954
WebDisp.TPageDispatcher.OnBeforeDispatchPage 24954
WebDisp.TPageDispatcher.OnCanViewPage 24955
WebDisp.TPageDispatcher.OnIsLoginRequired 24956
WebDisp.TPageDispatcher.OnPageAccessDenied 24956
WebDisp.TPageDispatcher.OnPageNotFound 24957
WebDisp.TWebAppComponents 24958
WebDisp.TWebAppComponents.AdapterDispatcher 24963
WebDisp.TWebAppComponents.ApplicationAdapter 24963
WebDisp.TWebAppComponents.DispatchActions 24964
WebDisp.TWebAppComponents.EndUserAdapter 24964
WebDisp.TWebAppComponents.LocateFileService 24965
WebDisp.TWebAppComponents.OnAfterDispatch 24967
WebDisp.TWebAppComponents.OnBeforeDispatch 24967
WebDisp.TWebAppComponents.OnException 24968
WebDisp.TWebAppComponents.PageDispatcher 24965
WebDisp.TWebAppComponents.Sessions 24966
WebDisp.TWebAppComponents.UserListService 24966
WebDisp.TWebRequest 24989
WebDisp.TWebRequest type 24989
WebDisp.TWebResponse 24989
WebDisp.TWebResponse type 24989
WebDisp.TWebSession 24968
WebDisp.TWebSession.Create 24970
WebDisp.TWebSession.GetSessionID 24971
WebDisp.TWebSession.GetTimeoutMinutes 24971
WebDisp.TWebSession.GetValue 24972
WebDisp.TWebSession.SetTimeoutMinutes 24972
WebDisp.TWebSession.SetValue 24972
WebDisp.TWebSession.Terminate 24973
WebDisp.TWebSession.UpdateResponse 24973
WebFact 24993
WebFact namespace 24993
    Classes 24994
    Structs, Records, Enums 25022
    Types 25023
WebFact.TBaseWebModuleFactory 24994
WebFact.TBaseWebModuleFactory.Create 24995
WebFact.TBaseWebModuleFactory.GetCacheMode 24996
WebFact.TBaseWebModuleFactory.GetComponentClass 24997
WebFact.TBaseWebModuleFactory.GetCreateMode 24997
WebFact.TBaseWebModuleFactory.GetIsAppModule 24997
WebFact.TBaseWebModuleFactory.GetModule 24998
WebFact.TBaseWebModuleFactory.GetModuleName 24998
WebFact.TBaseWebModuleFactory.PreventDestruction 24998
WebFact.TBaseWebPageInfo 24999
WebFact.TBaseWebPageInfo.Create 25000

yyyy

WebFact.TBaseWebPageInfo.GetIsLoginRequired 25001
WebFact.TBaseWebPageInfo.GetIsPublished 25002
WebFact.TBaseWebPageInfo.GetPageDescription 25002
WebFact.TBaseWebPageInfo.GetPageFile 25002
WebFact.TBaseWebPageInfo.GetPageHREF 25003
WebFact.TBaseWebPageInfo.GetPageName 25003
WebFact.TBaseWebPageInfo.GetPageTitle 25003
WebFact.TBaseWebPageInfo.GetViewAccess 25004
WebFact.TBaseWebPageInfo.SetFactory 25004
WebFact.TBaseWebPageModuleFactory 25004
WebFact.TBaseWebPageModuleFactory.Create 25006
WebFact.TBaseWebPageModuleFactory.GetCacheMode 25007
WebFact.TBaseWebPageModuleFactory.GetComponentClass 25007
WebFact.TBaseWebPageModuleFactory.GetCreateMode 25007
WebFact.TBaseWebPageModuleFactory.GetIsAppModule 25008
WebFact.TBaseWebPageModuleFactory.GetModule 25008
WebFact.TBaseWebPageModuleFactory.GetModuleName 25009
WebFact.TBaseWebPageModuleFactory.PreventDestruction 25009
WebFact.TWebAppDataModuleFactory 25009
WebFact.TWebAppDataModuleFactory.Create 25011
WebFact.TWebAppDataModuleFactory.GetIsAppModule 25011
WebFact.TWebAppPageModuleFactory 25012
WebFact.TWebAppPageModuleFactory.Create 25013
WebFact.TWebAppPageModuleFactory.GetIsAppModule 25014
WebFact.TWebDataModuleFactory 25014
WebFact.TWebDataModuleFactory.Create 25016
WebFact.TWebPageAccess 25023
WebFact.TWebPageAccess type 25023
WebFact.TWebPageAccessFlags 25022
WebFact.TWebPageAccessFlags enumeration 25022
WebFact.TWebPageInfo 25016
WebFact.TWebPageInfo.Create 25018
WebFact.TWebPageModuleFactory 25019
WebFact.TWebPageModuleFactory.Create 25020
WebForm 25025
WebForm namespace 25025
Classes 25028
Constants 25323
Functions 25312
Interfaces 25303
Structs, Records, Enums 25315
Types 25321
WebForm.CalcDisplayFieldHTMLElementType 25312
WebForm.CalcDisplayFieldHTMLElementType function 25312
WebForm.CalcDisplayFieldViewMode 25312
WebForm.CalcDisplayFieldViewMode function 25312
WebForm.FindVariablesContainer 25313
WebForm.FindVariablesContainer function 25313
WebForm.FindVariablesContainerInParent 25313
WebForm.FindVariablesContainerInParent function 25313
WebForm.GetAdapterModeOfAdapterInParent 25313
WebForm.GetAdapterModeOfAdapterInParent function 25313
WebForm.IAdapterDisplay 25303
WebForm.IGetAdapterFormLayout 25303
WebForm.IGetAdapterFormLayout.GetAdapterFormLayout 25304
WebForm.IGetAdapterModeOfDisplay 25304
WebForm.IGetAdapterModeOfDisplay.GetAdapterModeOfAdapter 25305
WebForm.IGetAdapterModeOfDisplay.GetAdapterModeOfDisplay 25305
WebForm.IGetAdapterOfDisplay 25305
WebForm.IGetAdapterOfDisplay.GetAdapter 25306
WebForm.IGetAdapterVariableName 25306
WebForm.IGetAdapterVariableName.AdapterVariableName 25308
WebForm.IGetAdapterVariableName.GetAdapterVariableName 25307
WebForm.IGetControlRequirements 25308
WebForm.IGetControlRequirements.GetControlRequirements 25309
WebForm.IGetFormRequirements 25309
WebForm.IGetFormRequirements.GetFormRequirements 25310
WebForm.IGetVariablesContainerOfDisplay 25310
WebForm.IGetVariablesContainerOfDisplay.GetVariablesContainer 25311
WebForm.MakeAdapterHTMLName 25314

zzzz

WebForm.MakeAdapterHTMLName function 25314
WebForm.MakeAdapterVariableName 25314
WebForm.MakeAdapterVariableName function 25314
WebForm.sDisplayText 25323
WebForm.sDisplayText constant 25323
WebForm.sFieldText 25323
WebForm.sFieldText constant 25323
WebForm.sFormGet 25324
WebForm.sFormGet constant 25324
WebForm.sFormPost 25324
WebForm.sFormPost constant 25324
WebForm.sMultipartForm 25324
WebForm.sMultipartForm constant 25324
WebForm.TAbstractControlRequirements 25029
WebForm.TAbstractControlRequirements.DefineFunction 25030
WebForm.TAbstractControlRequirements.DefineVariable 25030
WebForm.TAbstractControlRequirements.IsFunctionDefined 25031
WebForm.TAbstractControlRequirements.IsVariableDefined 25031
WebForm.TAbstractControlRequirements.RequiresMultipartForm 25032
WebForm.TAbstractFormRequirements 25032
WebForm.TAbstractFormRequirements.DefineFunction 25033
WebForm.TAbstractFormRequirements.DefineVariable 25033
WebForm.TAbstractFormRequirements.IsFunctionDefined 25033
WebForm.TAbstractFormRequirements.IsVariableDefined 25034
WebForm.TActionButtonHideOption 25315
WebForm.TActionButtonHideOption enumeration 25315
WebForm.TActionButtonHideOptions 25321
WebForm.TActionButtonHideOptions type 25321
WebForm.TAdapterActionButton 25034
WebForm.TAdapterActionButton.ActionName 25039
WebForm.TAdapterActionButton.Align 25040
WebForm.TAdapterActionButton.Caption 25040
WebForm.TAdapterActionButton.Custom 25040
WebForm.TAdapterActionButton.DisplayColumns 25041
WebForm.TAdapterActionButton.DisplayType 25041
WebForm.TAdapterActionButton.ErrorPageName 25042
WebForm.TAdapterActionButton.HideOptions 25042
WebForm.TAdapterActionButton.ImageHeight 25043
WebForm.TAdapterActionButton.ImageProducer 25043
WebForm.TAdapterActionButton.ImageWidth 25043
WebForm.TAdapterActionButton.PageName 25044
WebForm.TAdapterActionButton.Style 25044
WebForm.TAdapterActionButton.StyleRule 25045
WebForm.TAdapterActionButton.VAlign 25045
WebForm.TAdapterActionsListHelper 25046
WebForm.TAdapterActionsListHelper.AddDefaultActions 25052
WebForm.TAdapterActionsListHelper.Create 25047
WebForm.TAdapterActionsListHelper.FindAction 25048
WebForm.TAdapterActionsListHelper.GetActionsList 25049
WebForm.TAdapterActionsListHelper.GetAddAllFieldsItem 25049
WebForm.TAdapterActionsListHelper.GetAddFieldsDlgCaption 25049
WebForm.TAdapterActionsListHelper.GetAddFieldsItem 25050
WebForm.TAdapterActionsListHelper.GetNewFieldPrefix 25050
WebForm.TAdapterActionsListHelper.GetVisibleActions 25051
WebForm.TAdapterActionsListHelper.IsActionInUse 25051
WebForm.TAdapterCommandColumn 25052
WebForm.TAdapterCommandColumn.AddDefaultCommands 25058
WebForm.TAdapterCommandColumn.Align 25059
WebForm.TAdapterCommandColumn.Caption 25059
WebForm.TAdapterCommandColumn.CaptionAttributes 25060
WebForm.TAdapterCommandColumn.Custom 25060
WebForm.TAdapterCommandColumn.DisplayColumns 25060
WebForm.TAdapterCommandColumn.Style 25061
WebForm.TAdapterCommandColumn.StyleRule 25061
WebForm.TAdapterCommandColumn.VAlign 25062
WebForm.TAdapterCommandGroup 25062
WebForm.TAdapterCommandGroup.AddDefaultCommands 25069
WebForm.TAdapterCommandGroup.Custom 25069
WebForm.TAdapterCommandGroup.DisplayComponent 25069
WebForm.TAdapterCommandGroup.Style 25070
WebForm.TAdapterCommandGroup.StyleRule 25070
WebForm.TAdapterControlGroup 25070

WebForm.TAdapterControlGroup.Adapter 25084
WebForm.TAdapterControlGroup.AdapterChange 25077
WebForm.TAdapterControlGroup.AdapterMode 25085
WebForm.TAdapterControlGroup.AddDefaultFields 25085
WebForm.TAdapterControlGroup.AddNotify 25077
WebForm.TAdapterControlGroup.Create 25076
WebForm.TAdapterControlGroup.CreateDefaultFields 25078
WebForm.TAdapterControlGroup.Destroy 25076
WebForm.TAdapterControlGroup.FindField 25078
WebForm.TAdapterControlGroup.GetAdapter 25079
WebForm.TAdapterControlGroup.GetAdapterModeOfAdapter 25079
WebForm.TAdapterControlGroup.GetAdapterModeOfDisplay 25080
WebForm.TAdapterControlGroup.GetAdapterVariableName 25080
WebForm.TAdapterControlGroup.GetControlRequirements 25080
WebForm.TAdapterControlGroup.GetDesigntimeWarnings 25081
WebForm.TAdapterControlGroup.GetFieldsList 25081
WebForm.TAdapterControlGroup.GetVisibleFields 25082
WebForm.TAdapterControlGroup.ImplGetControlRequirements 25082
WebForm.TAdapterControlGroup.IsFieldInUse 25082
WebForm.TAdapterControlGroup.Notification 25083
WebForm.TAdapterControlGroup.NotifyAdapterChange 25083
WebForm.TAdapterControlGroup.RemoveNotify 25083
WebForm.TAdapterControlGroup.SetAdapter 25084
WebForm.TAdapterControlGroup.VisibleFields 25086
WebForm.TAdapterDataDisplay 25086
WebForm.TAdapterDataDisplay.Align 25097
WebForm.TAdapterDataDisplay.Create 25092
WebForm.TAdapterDataDisplay.FieldConditions 25092
WebForm.TAdapterDataDisplay.FieldName 25098
WebForm.TAdapterDataDisplay.FindAssociatedField 25093
WebForm.TAdapterDataDisplay.FormatCaption 25093
WebForm.TAdapterDataDisplay.FormatColumnData 25093
WebForm.TAdapterDataDisplay.FormatColumnHeading 25093
WebForm.TAdapterDataDisplay.GetAdapterModeOfAdapter 25093
WebForm.TAdapterDataDisplay.GetAdapterVariableName 25094
WebForm.TAdapterDataDisplay.GetCaption 25094
WebForm.TAdapterDataDisplay.GetControlRequirements 25094
WebForm.TAdapterDataDisplay.GetDesigntimeWarnings 25094
WebForm.TAdapterDataDisplay.GetFieldName 25095
WebForm.TAdapterDataDisplay.GetVariablesContainer 25095
WebForm.TAdapterDataDisplay.HideOptions 25098
WebForm.TAdapterDataDisplay.ImplContent 25095
WebForm.TAdapterDataDisplay.ImplFormatColumnData 25096
WebForm.TAdapterDataDisplay.ImplFormatColumnHeading 25096
WebForm.TAdapterDataDisplay.ImplGetControlRequirements 25096
WebForm.TAdapterDataDisplay.ImplRestoreDefaults 25096
WebForm.TAdapterDataDisplay.RestoreDefaults 25096
WebForm.TAdapterDataDisplay.SetCaption 25097
WebForm.TAdapterDataDisplay.SetFieldName 25097
WebForm.TAdapterDataDisplay.VAlign 25098
WebForm.TAdapterDataDisplayClass 25321
WebForm.TAdapterDataDisplayClass type 25321
WebForm.TAdapterDisplayColumn 25099
WebForm.TAdapterDisplayColumn.Align 25105
WebForm.TAdapterDisplayColumn.Caption 25105
WebForm.TAdapterDisplayColumn.CaptionAttributes 25105
WebForm.TAdapterDisplayColumn.Custom 25106
WebForm.TAdapterDisplayColumn.DisplayType 25106
WebForm.TAdapterDisplayColumn.FieldName 25107
WebForm.TAdapterDisplayColumn.HideOptions 25107
WebForm.TAdapterDisplayColumn.ImageHeight 25107
WebForm.TAdapterDisplayColumn.ImageWidth 25107
WebForm.TAdapterDisplayColumn.Style 25108
WebForm.TAdapterDisplayColumn.StyleRule 25108
WebForm.TAdapterDisplayColumn.VAlign 25108
WebForm.TAdapterDisplayField 25109
WebForm.TAdapterDisplayField.Align 25115
WebForm.TAdapterDisplayField.Caption 25116
WebForm.TAdapterDisplayField.CaptionAttributes 25116
WebForm.TAdapterDisplayField.CaptionPosition 25116
WebForm.TAdapterDisplayField.Custom 25117
WebForm.TAdapterDisplayField.DisplayColumns 25117

WebForm.TAdapterDisplayField.DisplayRows 25117
WebForm.TAdapterDisplayField.DisplayType 25118
WebForm.TAdapterDisplayField.DisplayWidth 25118
WebForm.TAdapterDisplayField.FieldName 25118
WebForm.TAdapterDisplayField.HideOptions 25119
WebForm.TAdapterDisplayField.ImageHeight 25119
WebForm.TAdapterDisplayField.ImageWidth 25119
WebForm.TAdapterDisplayField.InputType 25120
WebForm.TAdapterDisplayField.MaxLength 25120
WebForm.TAdapterDisplayField.SelectRows 25120
WebForm.TAdapterDisplayField.Style 25121
WebForm.TAdapterDisplayField.StyleRule 25121
WebForm.TAdapterDisplayField.TextAreaWrap 25121
WebForm.TAdapterDisplayField.VAlign 25122
WebForm.TAdapterDisplayField.ViewMode 25122
WebForm.TAdapterDisplayReferenceGroup 25122
WebForm.TAdapterDisplayReferenceGroup.AdapterChange 25128
WebForm.TAdapterDisplayReferenceGroup.Create 25127
WebForm.TAdapterDisplayReferenceGroup.Destroy 25128
WebForm.TAdapterDisplayReferenceGroup.DisplayComponent 25132
WebForm.TAdapterDisplayReferenceGroup.DisplayComponentChange 25129
WebForm.TAdapterDisplayReferenceGroup.GetAdapterDisplayCharacteristics 25129
WebForm.TAdapterDisplayReferenceGroup.GetDesigntimeWarnings 25130
WebForm.TAdapterDisplayReferenceGroup.NotifyAdapterChange 25130
WebForm.TAdapterDisplayReferenceGroup.OnDisplayComponentChange 25131
WebForm.TAdapterDisplayReferenceGroup.SetWebDisplayComponent 25131
WebForm.TAdapterEditColumn 25132
WebForm.TAdapterEditColumn.Align 25138
WebForm.TAdapterEditColumn.Caption 25139
WebForm.TAdapterEditColumn.CaptionAttributes 25139
WebForm.TAdapterEditColumn.CaptionPosition 25139
WebForm.TAdapterEditColumn.Custom 25140
WebForm.TAdapterEditColumn.DisplayColumns 25140
WebForm.TAdapterEditColumn.DisplayRows 25140
WebForm.TAdapterEditColumn.DisplayType 25141
WebForm.TAdapterEditColumn.DisplayWidth 25141
WebForm.TAdapterEditColumn.FieldName 25141
WebForm.TAdapterEditColumn.HideOptions 25142
WebForm.TAdapterEditColumn.ImageHeight 25142
WebForm.TAdapterEditColumn.ImageWidth 25142
WebForm.TAdapterEditColumn.InputType 25143
WebForm.TAdapterEditColumn.MaxLength 25143
WebForm.TAdapterEditColumn.SelectRows 25143
WebForm.TAdapterEditColumn.Style 25144
WebForm.TAdapterEditColumn.StyleRule 25144
WebForm.TAdapterEditColumn.TextAreaWrap 25144
WebForm.TAdapterEditColumn.VAlign 25145
WebForm.TAdapterEditColumn.ViewMode 25145
WebForm.TAdapterErrorList 25145
WebForm.TAdapterErrorList.Adapter 25149
WebForm.TAdapterFieldGroup 25150
WebForm.TAdapterFieldGroup.Adapter 25156
WebForm.TAdapterFieldGroup.AdapterMode 25157
WebForm.TAdapterFieldGroup.AddDefaultFields 25157
WebForm.TAdapterFieldGroup.Custom 25158
WebForm.TAdapterFieldGroup.Style 25158
WebForm.TAdapterFieldGroup.StyleRule 25158
WebForm.TAdapterForm 25159
WebForm.TAdapterForm.Custom 25164
WebForm.TAdapterForm.FormMethod 25165
WebForm.TAdapterForm.Style 25165
WebForm.TAdapterForm.StyleRule 25165
WebForm.TAdapterFormLayout 25166
WebForm.TAdapterFormLayout.BreakButtons 25174
WebForm.TAdapterFormLayout.ChildTableIndent 25174
WebForm.TAdapterFormLayout.ColumnCount 25174
WebForm.TAdapterFormLayout.Create 25168
WebForm.TAdapterFormLayout.EndButtons 25169
WebForm.TAdapterFormLayout.EndField 25169

WebForm.TAdapterFormLayout.EndFields 25169
WebForm.TAdapterFormLayout.EndLayout 25169
WebForm.TAdapterFormLayout.EndRow 25169
WebForm.TAdapterFormLayout.EndTable 25170
WebForm.TAdapterFormLayout.FieldIndent 25175
WebForm.TAdapterFormLayout.GetAdapterFormLayout 25170
WebForm.TAdapterFormLayout.GetChildTableIndent 25170
WebForm.TAdapterFormLayout.GetFieldIndent 25170
WebForm.TAdapterFormLayout.GetTableIndent 25171
WebForm.TAdapterFormLayout.ImplLayoutButton 25171
WebForm.TAdapterFormLayout.ImplLayoutField 25171
WebForm.TAdapterFormLayout.ImplLayoutLabelAndField 25172
WebForm.TAdapterFormLayout.ImplLayoutTable 25172
WebForm.TAdapterFormLayout.NextColumn 25173
WebForm.TAdapterFormLayout.StartButtons 25173
WebForm.TAdapterFormLayout.StartFields 25173
WebForm.TAdapterFormLayout.StartRow 25173
WebForm.TAdapterFormLayout.StartTable 25174
WebForm.TAdapterFormLayout.TableHeader 25175
WebForm.TAdapterFormLayout.TableIndent 25175
WebForm.TAdapterGrid 25175
WebForm.TAdapterGrid.Adapter 25182
WebForm.TAdapterGrid.AdapterMode 25182
WebForm.TAdapterGrid.AddDefaultFields 25183
WebForm.TAdapterGrid.HeadingAttributes 25184
WebForm.TAdapterGrid.RowAttributes 25184
WebForm.TAdapterGrid.TableAttributes 25184
WebForm.TColumnHTMLElementType 25316
WebForm.TColumnHTMLElementType enumeration 25316
WebForm.TCommandHTMLElementType 25316
WebForm.TCommandHTMLElementType enumeration 25316
WebForm.TControlRequirements 25185
WebForm.TControlRequirements.Create 25186
WebForm.TControlRequirements.DefineFunction 25187
WebForm.TControlRequirements.DefineVariable 25187
WebForm.TControlRequirements.Destroy 25186
WebForm.TControlRequirements.IsFunctionDefined 25188
WebForm.TControlRequirements.IsVariableDefined 25188
WebForm.TControlRequirements.NeedMultipartForm 25189
WebForm.TControlRequirements.RequiresMultipartForm 25189
WebForm.TCustomAdapterActionButton 25189
WebForm.TCustomAdapterActionButton.ActionName 25200
WebForm.TCustomAdapterActionButton.Align 25201
WebForm.TCustomAdapterActionButton.Create 25194
WebForm.TCustomAdapterActionButton.Destroy 25195
WebForm.TCustomAdapterActionButton.DisplayColumns 25201
WebForm.TCustomAdapterActionButton.DisplayType 25202
WebForm.TCustomAdapterActionButton.ErrorPageName 25202
WebForm.TCustomAdapterActionButton.FindAssociatedAction 25195
WebForm.TCustomAdapterActionButton.GetActionName 25196
WebForm.TCustomAdapterActionButton.GetAdapterModeOfAdapter 25196
WebForm.TCustomAdapterActionButton.GetAdapterVariableName 25196
WebForm.TCustomAdapterActionButton.GetCaption 25196
WebForm.TCustomAdapterActionButton.GetControlRequirements 25197
WebForm.TCustomAdapterActionButton.GetDesigntimeWarnings 25197
WebForm.TCustomAdapterActionButton.GetHTMLFormVarName 25197
WebForm.TCustomAdapterActionButton.GetLayoutAttributes 25197
WebForm.TCustomAdapterActionButton.HideOptions 25202
WebForm.TCustomAdapterActionButton.ImageHeight 25203
WebForm.TCustomAdapterActionButton.ImageProducer 25203
WebForm.TCustomAdapterActionButton.ImageWidth 25204
WebForm.TCustomAdapterActionButton.ImplContent 25198
WebForm.TCustomAdapterActionButton.ImplGetControlRequirements 25198
WebForm.TCustomAdapterActionButton.ImplRestoreDefaults 25198
WebForm.TCustomAdapterActionButton.Notification 25199
WebForm.TCustomAdapterActionButton.PageName 25204
WebForm.TCustomAdapterActionButton.RestoreDefaults 25199
WebForm.TCustomAdapterActionButton.SetActionName 25199
WebForm.TCustomAdapterActionButton.SetCaption 25200

WebForm.TCustomAdapterActionButton.VAlign 25204
WebForm.TCustomAdapterCommandColumn 25205
WebForm.TCustomAdapterCommandColumn.AdapterActionsListHelper 25215
WebForm.TCustomAdapterCommandColumn.AddDefaultCommands 25215
WebForm.TCustomAdapterCommandColumn.Align 25216
WebForm.TCustomAdapterCommandColumn.Caption 25216
WebForm.TCustomAdapterCommandColumn.CaptionAttributes 25217
WebForm.TCustomAdapterCommandColumn.Create 25210
WebForm.TCustomAdapterCommandColumn.Custom 25217
WebForm.TCustomAdapterCommandColumn.Destroy 25211
WebForm.TCustomAdapterCommandColumn.DisplayColumns 25217
WebForm.TCustomAdapterCommandColumn.FormatColumnData 25211
WebForm.TCustomAdapterCommandColumn.FormatColumnHeading 25212
WebForm.TCustomAdapterCommandColumn.GetAdapterDisplayCharacteristics 25212
WebForm.TCustomAdapterCommandColumn.GetLayoutAttributes 25212
WebForm.TCustomAdapterCommandColumn.ImplCanAddClass 25213
WebForm.TCustomAdapterCommandColumn.ImplContent 25213
WebForm.TCustomAdapterCommandColumn.NotifyAdapterChange 25214
WebForm.TCustomAdapterCommandColumn.Style 25218
WebForm.TCustomAdapterCommandColumn.StyleRule 25218
WebForm.TCustomAdapterCommandColumn.VAlign 25219
WebForm.TCustomAdapterCommandColumn.VisibleCommands 25219
WebForm.TCustomAdapterCommandGroup 25220
WebForm.TCustomAdapterCommandGroup.Adapter 25228
WebForm.TCustomAdapterCommandGroup.AdapterActionsListHelper 25229
WebForm.TCustomAdapterCommandGroup.AddDefaultCommands 25229
WebForm.TCustomAdapterCommandGroup.Create 25225
WebForm.TCustomAdapterCommandGroup.Custom 25229
WebForm.TCustomAdapterCommandGroup.Destroy 25225
WebForm.TCustomAdapterCommandGroup.GetAdapter 25226
WebForm.TCustomAdapterCommandGroup.GetAdapterFormLayout 25226
WebForm.TCustomAdapterCommandGroup.GetAdapterModeOfAdapter 25227
WebForm.TCustomAdapterCommandGroup.GetAdapterModeOfDisplay 25227
WebForm.TCustomAdapterCommandGroup.ImplCanAddClass 25227
WebForm.TCustomAdapterCommandGroup.ImplContent 25228
WebForm.TCustomAdapterCommandGroup.Style 25229
WebForm.TCustomAdapterCommandGroup.StyleRule 25230
WebForm.TCustomAdapterCommandGroup.VisibleCommands 25230
WebForm.TCustomAdapterDisplayColumn 25230
WebForm.TCustomAdapterDisplayColumn.CanAddClass 25237
WebForm.TCustomAdapterDisplayColumn.ControlContent 25237
WebForm.TCustomAdapterDisplayColumn.Create 25236
WebForm.TCustomAdapterDisplayColumn.DisplayType 25239
WebForm.TCustomAdapterDisplayColumn.FormatImage 25237
WebForm.TCustomAdapterDisplayColumn.FormatListDisplay 25237
WebForm.TCustomAdapterDisplayColumn.FormatTextDisplay 25238
WebForm.TCustomAdapterDisplayColumn.ImageHeight 25240
WebForm.TCustomAdapterDisplayColumn.ImageWidth 25240
WebForm.TCustomAdapterDisplayColumn.ImplCanAddClass 25238
WebForm.TCustomAdapterDisplayColumn.ImplFormatColumnHeading 25238
WebForm.TCustomAdapterDisplayColumn.ImplGetControlRequirements 25238
WebForm.TCustomAdapterDisplayColumn.ImplRestoreDefaults

eeee

25239
WebForm.TCustomAdapterDisplayColumn.IsColumn 25239
WebForm.TCustomAdapterDisplayField 25240
WebForm.TCustomAdapterDisplayField.AddAttributes 25247
WebForm.TCustomAdapterDisplayField.ControlContent 25247
WebForm.TCustomAdapterDisplayField.Create 25246
WebForm.TCustomAdapterDisplayField.DisplayColumns 25249
WebForm.TCustomAdapterDisplayField.DisplayRows 25249
WebForm.TCustomAdapterDisplayField.DisplayType 25249
WebForm.TCustomAdapterDisplayField.DisplayWidth 25250
WebForm.TCustomAdapterDisplayField.GetLayoutAttributes 25247
WebForm.TCustomAdapterDisplayField.ImageHeight 25250
WebForm.TCustomAdapterDisplayField.ImageWidth 25250
WebForm.TCustomAdapterDisplayField.ImplContent 25247
WebForm.TCustomAdapterDisplayField.ImplGetControlRequirements 25248
WebForm.TCustomAdapterDisplayField.ImplRestoreDefaults 25248
WebForm.TCustomAdapterDisplayField.InputType 25251
WebForm.TCustomAdapterDisplayField.MaxLength 25251
WebForm.TCustomAdapterDisplayField.SelectRows 25251
WebForm.TCustomAdapterDisplayField.TextAreaWrap 25251
WebForm.TCustomAdapterDisplayField.ViewMode 25252
WebForm.TCustomAdapterEditColumn 25252
WebForm.TCustomAdapterEditColumn.ImplContent 25258
WebForm.TCustomAdapterErrorList 25259
WebForm.TCustomAdapterErrorList.Adapter 25265
WebForm.TCustomAdapterErrorList.Content 25263
WebForm.TCustomAdapterErrorList.Create 25262
WebForm.TCustomAdapterErrorList.Destroy 25263
WebForm.TCustomAdapterErrorList.GetDesigntimeWarnings 25264
WebForm.TCustomAdapterErrorList.ImplContent 25264
WebForm.TCustomAdapterErrorList.Notification 25264
WebForm.TCustomAdapterFieldGroup 25265
WebForm.TCustomAdapterFieldGroup.Create 25271
WebForm.TCustomAdapterFieldGroup.Custom 25275
WebForm.TCustomAdapterFieldGroup.Destroy 25272
WebForm.TCustomAdapterFieldGroup.GetAdapterDisplayCharacteristics 25272
WebForm.TCustomAdapterFieldGroup.GetAdapterFormLayout 25272
WebForm.TCustomAdapterFieldGroup.GetAddAllFieldsItem 25273
WebForm.TCustomAdapterFieldGroup.GetAddFieldsDlgCaption 25273
WebForm.TCustomAdapterFieldGroup.GetAddFieldsItem 25273
WebForm.TCustomAdapterFieldGroup.GetLayoutAttributes 25273
WebForm.TCustomAdapterFieldGroup.GetNewFieldPrefix 25274
WebForm.TCustomAdapterFieldGroup.ImplCanAddClass 25274
WebForm.TCustomAdapterFieldGroup.ImplContent 25274
WebForm.TCustomAdapterFieldGroup.Style 25275
WebForm.TCustomAdapterFieldGroup.StyleRule 25276
WebForm.TCustomAdapterForm 25276
WebForm.TCustomAdapterForm.Create 25281
WebForm.TCustomAdapterForm.Custom 25285
WebForm.TCustomAdapterForm.Destroy 25281
WebForm.TCustomAdapterForm.FormMethod 25285
WebForm.TCustomAdapterForm.GetAdapterFormLayout 25282
WebForm.TCustomAdapterForm.GetFormRequirements 25282
WebForm.TCustomAdapterForm.GetHTMLFormName 25282
WebForm.TCustomAdapterForm.GetHTMLFormVarName 25283
WebForm.TCustomAdapterForm.GetLayoutAttributes 25283
WebForm.TCustomAdapterForm.ImplCanAddClass 25283
WebForm.TCustomAdapterForm.ImplContent 25284
WebForm.TCustomAdapterForm.Style 25285
WebForm.TCustomAdapterForm.StyleRule 25286
WebForm.TCustomAdapterGrid 25286
WebForm.TCustomAdapterGrid.Create 25292
WebForm.TCustomAdapterGrid.Destroy 25293
WebForm.TCustomAdapterGrid.GetAdapterDisplayCharacteristics 25293
WebForm.TCustomAdapterGrid.GetAdapterFormLayout 25294
WebForm.TCustomAdapterGrid.GetAddAllFieldsItem 25294
WebForm.TCustomAdapterGrid.GetAddFieldsDlgCaption 25294
WebForm.TCustomAdapterGrid.GetAddFieldsItem 25294

WebForm.TCustomAdapterGrid.GetNewFieldPrefix 25294
WebForm.TCustomAdapterGrid.HeadingAttributes 25296
WebForm.TCustomAdapterGrid.ImplCanAddClass 25295
WebForm.TCustomAdapterGrid.ImplContent 25295
WebForm.TCustomAdapterGrid.RowAttributes 25296
WebForm.TCustomAdapterGrid.TableAttributes 25297
WebForm.TDesigntimeWarnings 25297
WebForm.TDesigntimeWarnings.AddObject 25298
WebForm.TDesigntimeWarnings.AddString 25299
WebForm.TDesigntimeWarnings.Create 25298
WebForm.TDesigntimeWarnings.Destroy 25298
WebForm.TDesigntimeWarnings.Strings 25299
WebForm.TDisplayFieldHideOption 25317
WebForm.TDisplayFieldHideOption enumeration 25317
WebForm.TDisplayFieldHideOptions 25321
WebForm.TDisplayFieldHideOptions type 25321
WebForm.TDisplayFieldHTMLElementType 25318
WebForm.TDisplayFieldHTMLElementType enumeration 25318
WebForm.TDisplayFieldViewMode 25318
WebForm.TDisplayFieldViewMode enumeration 25318
WebForm.TGeneratedFunctions 25300
WebForm.TGeneratedFunctions.Add 25301
WebForm.TGeneratedFunctions.Create 25300
WebForm.TGeneratedFunctions.Destroy 25301
WebForm.TGeneratedFunctions.Exists 25301
WebForm.TGeneratedFunctions.ItemCount 25302
WebForm.TGeneratedFunctions.Names 25302
WebForm.TGeneratedFunctions.Script 25302
WebForm.TInputFieldHTMLElementType 25319
WebForm.TInputFieldHTMLElementType enumeration 25319
WebForm.TLayoutState 25320
WebForm.TLayoutState enumeration 25320
weBinding enumeration member 25925
WebModu 25325
WebModu namespace 25325
Classes 25327
Functions 25426
Types 25428
WebModu.EWebModuleException 25327
WebModu.FindNamedModuleVariable 25426
WebModu.FindNamedModuleVariable function 25426
WebModu.GetModuleVariable 25426
WebModu.GetModuleVariable function 25426
WebModu.GetModuleVariableCount 25426
WebModu.GetModuleVariableCount function 25426
WebModu.TAdapterActionName 25328
WebModu.TAdapterActionName.Action 25331
WebModu.TAdapterActionName.ActionName 25331
WebModu.TAdapterActionName.Adapter 25332
WebModu.TAdapterActionName.AssignTo 25330
WebModu.TAdapterActionName.Create 25329
WebModu.TCustomWebAppDataModule 25332
WebModu.TCustomWebAppDataModule.AppServices 25339
WebModu.TCustomWebAppDataModule.Create 25336
WebModu.TCustomWebAppDataModule.CreateNew 25337
WebModu.TCustomWebAppDataModule.Destroy 25337
WebModu.TCustomWebAppDataModule.GetWebAppServices 25337
WebModu.TCustomWebAppDataModule.Notification 25338
WebModu.TCustomWebAppDataModule.QueryInterface 25338
WebModu.TCustomWebAppPageModule 25339
WebModu.TCustomWebAppPageModule.AppServices 25347
WebModu.TCustomWebAppPageModule.Create 25344
WebModu.TCustomWebAppPageModule.CreateNew 25345
WebModu.TCustomWebAppPageModule.Destroy 25345
WebModu.TCustomWebAppPageModule.GetWebAppServices 25346
WebModu.TCustomWebAppPageModule.Notification 25346
WebModu.TCustomWebAppPageModule.QueryInterface 25346
WebModu.TCustomWebAppPageModule.SetAppServices 25347
WebModu.TCustomWebDataModule 25348
WebModu.TCustomWebDataModule.Create 25352
WebModu.TCustomWebDataModule.CreateNew 25352
WebModu.TCustomWebDataModule.Destroy 25353
WebModu.TCustomWebDataModule.GetSiteModuleHelper 25353
WebModu.TCustomWebDataModule.GetWebSnapObjClass 25354
WebModu.TCustomWebDataModule.Notification 25354

gggggg

WebModu.TCustomWebDataModule.OnActivate 25356
WebModu.TCustomWebDataModule.OnDeactivate 25357
WebModu.TCustomWebDataModule.Request 25355
WebModu.TCustomWebDataModule.Response 25355
WebModu.TCustomWebDataModule.Session 25355
WebModu.TCustomWebDataModule.SiteModuleHelper 25356
WebModu.TCustomWebPageModule 25357
WebModu.TCustomWebPageModule.Create 25362
WebModu.TCustomWebPageModule.CreateNew 25362
WebModu.TCustomWebPageModule.DefaultAction 25365
WebModu.TCustomWebPageModule.Destroy 25363
WebModu.TCustomWebPageModule.GetSiteModuleHelper 25363
WebModu.TCustomWebPageModule.GetSitePageModuleHelper 25364
WebModu.TCustomWebPageModule.Notification 25364
WebModu.TCustomWebPageModule.OnAfterDispatchPage 25366
WebModu.TCustomWebPageModule.OnAfterRedirectToPage 25367
WebModu.TCustomWebPageModule.OnBeforeDispatchPage 25367
WebModu.TCustomWebPageModule.OnBeforeRedirectToPage 25368
WebModu.TCustomWebPageModule.PageProducer 25365
WebModu.TCustomWebPageModule.SitePageModuleHelper 25366
WebModu.TGetParamsEvent 25428
WebModu.TGetParamsEvent type 25428
WebModu.TSiteModuleHelper 25368
WebModu.TSiteModuleHelper.Create 25370
WebModu.TSiteModuleHelper.Destroy 25370
WebModu.TSiteModuleHelper.EndIterator 25371
WebModu.TSiteModuleHelper.FindNamedVariable 25371
WebModu.TSiteModuleHelper.FindVariable 25372
WebModu.TSiteModuleHelper.FindVariableContainer 25372
WebModu.TSiteModuleHelper.GetCurrentObject 25372
WebModu.TSiteModuleHelper.GetVariable 25372
WebModu.TSiteModuleHelper.GetVariableCount 25373
WebModu.TSiteModuleHelper.Module 25374
WebModu.TSiteModuleHelper.NextIteration 25373
WebModu.TSiteModuleHelper.Notification 25373
WebModu.TSiteModuleHelper.NotifyActivate 25373
WebModu.TSiteModuleHelper.NotifyDeactivate 25374
WebModu.TSiteModuleHelper.OnActivate 25376
WebModu.TSiteModuleHelper.OnDeactivate 25376
WebModu.TSiteModuleHelper.Request 25375
WebModu.TSiteModuleHelper.Response 25375
WebModu.TSiteModuleHelper.Session 25375
WebModu.TSiteModuleHelper.StartIterator 25374
WebModu.TSiteModuleHelper.WebModuleContext 25376
WebModu.TSitePageModuleHelper 25377
WebModu.TSitePageModuleHelper.Create 25379
WebModu.TSitePageModuleHelper.DefaultAction 25386
WebModu.TSitePageModuleHelper.Destroy 25380
WebModu.TSitePageModuleHelper.DispatchPage 25380
WebModu.TSitePageModuleHelper.GetDefaultAction 25381
WebModu.TSitePageModuleHelper.GetPageProducer 25381
WebModu.TSitePageModuleHelper.GetProducerComponent 25381
WebModu.TSitePageModuleHelper.GetRedirectParams 25381
WebModu.TSitePageModuleHelper.GetTemplateFileType 25382
WebModu.TSitePageModuleHelper.GetTemplateStream 25382
WebModu.TSitePageModuleHelper.GetXMLData 25382
WebModu.TSitePageModuleHelper.GetXSLData 25382
WebModu.TSitePageModuleHelper.HasHTMLBrowserView 25382
WebModu.TSitePageModuleHelper.HasScriptView 25383
WebModu.TSitePageModuleHelper.HasXMLBrowserView 25383
WebModu.TSitePageModuleHelper.HasXSLBrowserView 25383
WebModu.TSitePageModuleHelper.HTMLFromTemplate 25383
WebModu.TSitePageModuleHelper.ImplDispatchPage 25384
WebModu.TSitePageModuleHelper.ImplIncludePage 25384
WebModu.TSitePageModuleHelper.ImplRedirectToPage 25384
WebModu.TSitePageModuleHelper.ImplSetWebContentOptions 25384
WebModu.TSitePageModuleHelper.IncludePage 25385
WebModu.TSitePageModuleHelper.OnAfterDispatchPage 25386
WebModu.TSitePageModuleHelper.OnAfterRedirectToPage

hhhh

25387
WebModu.TSitePageModuleHelper.OnBeforeDispatchPage 25387
WebModu.TSitePageModuleHelper.OnBeforeRedirectToPage 25387
WebModu.TSitePageModuleHelper.RedirectToPage 25385
WebModu.TSitePageModuleHelper.SetDefaultPageFileName 25385
WebModu.TSitePageModuleHelper.SetWebContentOptions 25385
WebModu.TWebAppDataModule 25388
WebModu.TWebAppDataModule.AppServices 25392
WebModu.TWebAppDataModule.OnActivate 25393
WebModu.TWebAppDataModule.OnDeactivate 25393
WebModu.TWebAppPageModule 25394
WebModu.TWebAppPageModule.AppServices 25399
WebModu.TWebAppPageModule.DefaultAction 25400
WebModu.TWebAppPageModule.OnActivate 25401
WebModu.TWebAppPageModule.OnAfterDispatchPage 25401
WebModu.TWebAppPageModule.OnBeforeDispatchPage 25402
WebModu.TWebAppPageModule.OnDeactivate 25402
WebModu.TWebAppPageModule.PageProducer 25400
WebModu.TWebAppPageModuleHelper 25403
WebModu.TWebAppPageModuleHelper.GetPageProducer 25405
WebModu.TWebDataModule 25405
WebModu.TWebDataModule.OnActivate 25409
WebModu.TWebDataModule.OnDeactivate 25410
WebModu.TWebNotifyContainer 25410
WebModu.TWebNotifyContainer.Add 25413
WebModu.TWebNotifyContainer.Create 25411
WebModu.TWebNotifyContainer.Destroy 25412
WebModu.TWebNotifyContainer.FList 25413
WebModu.TWebNotifyContainer.GetCount 25413
WebModu.TWebNotifyContainer.NotifyWebActivate 25414
WebModu.TWebNotifyContainer.NotifyWebDeactivate 25414
WebModu.TWebPageModule 25414
WebModu.TWebPageModule.DefaultAction 25419
WebModu.TWebPageModule.OnActivate 25421
WebModu.TWebPageModule.OnAfterDispatchPage 25421
WebModu.TWebPageModule.OnBeforeDispatchPage 25421
WebModu.TWebPageModule.OnDeactivate 25422
WebModu.TWebPageModule.PageProducer 25420
WebModu.TWebPageModuleHelper 25422
WebModu.TWebPageModuleHelper.GetPageProducer 25425
WebNode 25429
WebNode namespace 25429
Interfaces 25430
WebNode.IWebNode 25430
WebNode.IWebNode.BeforeExecute 25431
WebNode.IWebNode.Execute 25432
WebNode.IWebNode.GetMimeBoundary 25433
WebNode.IWebNode.MimeBoundary 25433
WebNode.IWebNode.SetMimeBoundary 25433
WebReq 25435
WebReq namespace 25435
Classes 25436
Functions 25459
Structs, Records, Enums 25460
Variables 25461
WebReq.TRequestNotification 25460
WebReq.TRequestNotification enumeration 25460
WebReq.TWebModuleFactoryList 25436
WebReq.TWebModuleFactoryList.AddFactory 25438
WebReq.TWebModuleFactoryList.AppModuleFactory 25439
WebReq.TWebModuleFactoryList.Create 25437
WebReq.TWebModuleFactoryList.Destroy 25438
WebReq.TWebModuleFactoryList.GetItem 25439
WebReq.TWebModuleFactoryList.ItemCount 25439
WebReq.TWebModuleFactoryList.Items 25440
WebReq.TWebModuleList 25440
WebReq.TWebModuleList.AddModule 25443
WebReq.TWebModuleList.AddModuleClass 25443
WebReq.TWebModuleList.AddModuleName 25444
WebReq.TWebModuleList.AutoCreateModules 25445
WebReq.TWebModuleList.AutoDestroyModules 25445
WebReq.TWebModuleList.Create 25442
WebReq.TWebModuleList.Destroy 25442
WebReq.TWebModuleList.Factories 25448

WebReq.TWebModuleList.FindModuleClass 25446
WebReq.TWebModuleList.FindModuleName 25446
WebReq.TWebModuleList.GetFactory 25446
WebReq.TWebModuleList.GetFactoryCount 25447
WebReq.TWebModuleList.GetItem 25447
WebReq.TWebModuleList.GetItemCount 25447
WebReq.TWebModuleList.GetOnModuleAdded 25447
WebReq.TWebModuleList.SetOnModuleAdded 25448
WebReq.TWebRequestHandler 25448
WebReq.TWebRequestHandler.ActivateWebModules 25453
WebReq.TWebRequestHandler.ActiveCount 25455
WebReq.TWebRequestHandler.AddWebModuleFactory 25453
WebReq.TWebRequestHandler.CacheConnections 25455
WebReq.TWebRequestHandler.Create 25452
WebReq.TWebRequestHandler.DeactivateWebModules 25453
WebReq.TWebRequestHandler.Destroy 25452
WebReq.TWebRequestHandler.HandleException 25454
WebReq.TWebRequestHandler.HandleRequest 25454
WebReq.TWebRequestHandler.InactiveCount 25456
WebReq.TWebRequestHandler.MaxConnections 25457
WebReq.TWebRequestHandler.WebModuleClass 25457
WebReq.TWebRequestHandler.WebModuleFactory 25457
WebReq.TWebRequestHandler.WebModuleFactoryCount 25458
WebReq.WebRequestHandler 25459
WebReq.WebRequestHandler function 25459
WebReq.WebRequestHandlerProc 25461
WebReq.WebRequestHandlerProc variable 25461
WebScript 25463
WebScript namespace 25463
- Classes 25465
- Constants 25547
- Structs, Records, Enums 25544
- Types 25546

WebScript.CreateComponentStubFactory 25546
WebScript.CreateComponentStubFactory type 25546
WebScript.ERecursiveIncludeFile 25465
WebScript.EScriptEvalError 25466
WebScript.GlobalScriptVariableNames 25547
WebScript.GlobalScriptVariableNames constant 25547
WebScript.PLineNumbers 25546
WebScript.PLineNumbers type 25546
WebScript.TAbstractScriptEngine 25466
WebScript.TAbstractScriptEngine.AddObjectFactory 25468
WebScript.TAbstractScriptEngine.Create 25467
WebScript.TAbstractScriptEngine.CreateComponentStub 25468
WebScript.TAbstractScriptEngine.Destroy 25467
WebScript.TAbstractScriptEngine.ListLanguageNames 25469
WebScript.TAbstractScriptEngine.Name 25469
WebScript.TAbstractScriptEngine.ScriptObjectFactories 25470
WebScript.TAbstractScriptEngine.ScriptProducerClass 25470
WebScript.TAbstractScriptEngine.SupportsLanguage 25469
WebScript.TAbstractScriptObjectFactories 25470
WebScript.TAbstractScriptObjectFactories.Count 25472
WebScript.TAbstractScriptObjectFactories.Factories 25473
WebScript.TAbstractScriptObjectFactories.GetCount 25472
WebScript.TAbstractScriptObjectFactories.GetFactory 25472
WebScript.TAbstractScriptObjectFactories.GetObjectList 25472
WebScript.TAbstractScriptObjectFactories.ObjectList 25473
WebScript.TAbstractScriptObjectFactory 25473
WebScript.TAbstractScriptObjectFactory.AddGlobalObjects 25474
WebScript.TAbstractScriptObjectFactory.CreateComponentObject 25474
WebScript.TAbstractScriptObjectFactory.CreateGlobalObject 25475
WebScript.TAbstractScriptObjectFactory.CreateProducerObject 25475
WebScript.TAbstractScriptObjectFactory.GetScriptEngine 25475
WebScript.TCurrentWebPageInfo 25475
WebScript.TCurrentWebPageInfo.Create 25477
WebScript.TCurrentWebPageInfo.GetHasViewAccess 25478
WebScript.TCurrentWebPageInfo.GetLoginRequired 25478
WebScript.TCurrentWebPageInfo.GetPageDescription 25478
WebScript.TCurrentWebPageInfo.GetPageHREF 25479
WebScript.TCurrentWebPageInfo.GetPageIsPublished 25479
WebScript.TCurrentWebPageInfo.GetPageName 25479
WebScript.TCurrentWebPageInfo.GetPageTitle 25479
WebScript.TCurrentWebPageInfo.WebPageInfo 25480
WebScript.TGlobalScriptVariables 25544

jjjj

WebScript.TGlobalScriptVariables enumeration 25544
WebScript.THTMLItem 25480
WebScript.THTMLItem.AssignString 25482
WebScript.THTMLItem.Create 25481
WebScript.THTMLItem.HTML 25482
WebScript.THTMLItem.Ident 25482
WebScript.THTMLItem.Length 25483
WebScript.THTMLItem.Offset 25483
WebScript.THTMLItem.TagToken 25483
WebScript.THTMLItems 25483
WebScript.THTMLItems.Create 25485
WebScript.THTMLItems.Destroy 25485
WebScript.THTMLItems.GetHTMLItem 25486
WebScript.THTMLItems.Items 25487
WebScript.THTMLItems.MakeIdentifier 25486
WebScript.THTMLItems.Prepare 25486
WebScript.TIncludeType 25545
WebScript.TIncludeType enumeration 25545
WebScript.TLineNumberMap 25488
WebScript.TLineNumberMap.Count 25490
WebScript.TLineNumberMap.DestLineToSourceLine 25489
WebScript.TLineNumberMap.Items 25490
WebScript.TLineNumberMap.New 25489
WebScript.TLineNumbers 25545
WebScript.TLineNumbers record 25545
WebScript.TScriptCodeItem 25490
WebScript.TScriptCodeItem.AssignString 25492
WebScript.TScriptCodeItem.GetScript 25492
WebScript.TScriptEnginesList 25492
WebScript.TScriptEnginesList.AddEngine 25495
WebScript.TScriptEnginesList.AddObjectFactory 25495
WebScript.TScriptEnginesList.Count 25496
WebScript.TScriptEnginesList.Create 25493
WebScript.TScriptEnginesList.Destroy 25494
WebScript.TScriptEnginesList.Find 25495
WebScript.TScriptEnginesList.FindScriptProducerClass 25495
WebScript.TScriptEnginesList.Items 25496
WebScript.TScriptEnginesList.ListLanguageNames 25496
WebScript.TScriptEnginesList.RemoveEngine 25496
WebScript.TScriptError 25497
WebScript.TScriptError.Create 25498
WebScript.TScriptError.FCharPos 25499
WebScript.TScriptError.FDescription 25499
WebScript.TScriptError.FFileName 25499
WebScript.TScriptError.FLine 25499
WebScript.TScriptError.FSourceLine 25500
WebScript.TScriptError.GetCharPos 25500
WebScript.TScriptError.GetDescription 25500
WebScript.TScriptError.GetFileName 25500
WebScript.TScriptError.GetLine 25501
WebScript.TScriptError.GetSourceLine 25501
WebScript.TScriptError.SetDescription 25501
WebScript.TScriptErrors 25501
WebScript.TScriptErrors.Add 25504
WebScript.TScriptErrors.Create 25503
WebScript.TScriptErrors.Destroy 25503
WebScript.TScriptErrors.GetError 25504
WebScript.TScriptErrors.GetErrorCount 25504
WebScript.TScriptFile 25505
WebScript.TScriptFile.ApplicationName 25507
WebScript.TScriptFile.Create 25505
WebScript.TScriptFile.Destroy 25506
WebScript.TScriptFile.FLineNumberMap 25506
WebScript.TScriptFile.LongName 25507
WebScript.TScriptFile.ParseStream 25506
WebScript.TScriptFile.ParseString 25507
WebScript.TScriptFile.Script 25508
WebScript.TScriptFile.ShortName 25508
WebScript.TScriptFile.Source 25508
WebScript.TScriptGlobalObjects 25508
WebScript.TScriptGlobalObjects.AddNamedIntf 25511
WebScript.TScriptGlobalObjects.Create 25510
WebScript.TScriptGlobalObjects.Destroy 25510
WebScript.TScriptGlobalObjects.FindNamedItemIntf 25511
WebScript.TScriptGlobalObjects.GetNamedItemCount 25511
WebScript.TScriptGlobalObjects.GetNamedItemName 25511
WebScript.TScriptGlobalObjects.GetScriptEngine 25512
WebScript.TScriptGlobalObjects.NamedItemCount 25512

kkkk

WebScript.TScriptGlobalObjects.NamedItemName 25512
WebScript.TScriptGlobalObjects.Producer 25512
WebScript.TScriptGlobalObjects.ScriptEngine 25513
WebScript.TScriptIncludeFile 25513
WebScript.TScriptIncludeFile.Create 25514
WebScript.TScriptIncludeFile.LoadFile 25514
WebScript.TScriptIncludeFiles 25514
WebScript.TScriptIncludeFiles.Create 25516
WebScript.TScriptIncludeFiles.GetScriptIncludeFile 25517
WebScript.TScriptIncludeFiles.Items 25517
WebScript.TScriptIncludeItem 25518
WebScript.TScriptIncludeItem.Create 25519
WebScript.TScriptIncludeItem.GetScript 25520
WebScript.TScriptItem 25520
WebScript.TScriptItem.Create 25521
WebScript.TScriptItem.GetScript 25522
WebScript.TScriptItem.Script 25522
WebScript.TScriptItems 25522
WebScript.TScriptItems.Create 25524
WebScript.TScriptItems.GetScriptItem 25525
WebScript.TScriptItems.Items 25525
WebScript.TScriptObjectFactories 25526
WebScript.TScriptObjectFactories.Count 25530
WebScript.TScriptObjectFactories.Create 25527
WebScript.TScriptObjectFactories.CreateGlobalObject 25529
WebScript.TScriptObjectFactories.Destroy 25528
WebScript.TScriptObjectFactories.Factories 25530
WebScript.TScriptObjectFactories.GetCount 25529
WebScript.TScriptObjectFactories.GetFactory 25529
WebScript.TScriptObjectFactories.GetObjectList 25529
WebScript.TScriptProducer 25530
WebScript.TScriptProducer.Component 25539
WebScript.TScriptProducer.Content 25539
WebScript.TScriptProducer.Create 25532
WebScript.TScriptProducer.Designing 25540
WebScript.TScriptProducer.Destroy 25533
WebScript.TScriptProducer.Errors 25540
WebScript.TScriptProducer.Evaluate 25535
WebScript.TScriptProducer.EvaluateExpression 25535
WebScript.TScriptProducer.FErrors 25534
WebScript.TScriptProducer.FScriptEngine 25534
WebScript.TScriptProducer.FScriptFile 25534
WebScript.TScriptProducer.GetContent 25535
WebScript.TScriptProducer.GetErrors 25535
WebScript.TScriptProducer.GetHTMLBlock 25536
WebScript.TScriptProducer.GetHTMLBlockCount 25536
WebScript.TScriptProducer.GetWebModuleContext 25536
WebScript.TScriptProducer.HandleParserError 25536
WebScript.TScriptProducer.HandleScriptError 25536
WebScript.TScriptProducer.ImplHandleScriptError 25537
WebScript.TScriptProducer.ParseStream 25537
WebScript.TScriptProducer.ParseString 25537
WebScript.TScriptProducer.ReplaceTags 25537
WebScript.TScriptProducer.Script 25540
WebScript.TScriptProducer.SetContent 25538
WebScript.TScriptProducer.Write 25538
WebScript.TScriptProducer.WriteItem 25539
WebScript.TVariableLookup 25540
WebScript.TVariableLookup.Create 25541
WebScript.TVariableLookup.FindGlobalVariable 25542
WebScript.TVariableLookup.FindGlobalVariableContainer 25542
WebScript.TVariableLookup.Ident 25543
WebScript.TVariableLookup.LookupObject 25542
WebScript.TVariableLookup.Obj 25543
WebServExp 25549
WebServExp namespace 25549
Classes 25551
Functions 25562
Interfaces 25559
Structs, Records, Enums 25564
Types 25566
WebServExp.ArgumentType 25564
WebServExp.ArgumentType enumeration 25564
WebServExp.GetAliasBaseTypeInfo 25562
WebServExp.GetAliasBaseTypeInfo function 25562
WebServExp.GetBindingType 25562
WebServExp.GetBindingType function 25562
WebServExp.IsBaseClassTypeInfo 25563

WebServExp.IsBaseClassTypeInfo function 25563
WebServExp.IWebServExp 25559
WebServExp.IWebServExp.FindOrAddSchema 25560
WebServExp.IWebServExp.GenerateXMLSchema 25560
WebServExp.IWebServExpAccess 25561
WebServExp.IWebServExpAccess.GetWebServExp 25561
WebServExp.MessageType 25564
WebServExp.MessageType enumeration 25564
WebServExp.TAfterPublishingWSDLEvent 25566
WebServExp.TAfterPublishingWSDLEvent type 25566
WebServExp.TBeforePublishingTypesEvent 25566
WebServExp.TBeforePublishingTypesEvent type 25566
WebServExp.TPublishingTypeEvent 25567
WebServExp.TPublishingTypeEvent type 25567
WebServExp.TSchemaType 25565
WebServExp.TSchemaType record 25565
WebServExp.TSchemaTypeArray 25567
WebServExp.TSchemaTypeArray type 25567
WebServExp.TWebServExp 25551
WebServExp.TWebServExp.ArrayAsComplexContent 25555
WebServExp.TWebServExp.BindingType 25555
WebServExp.TWebServExp.Create 25553
WebServExp.TWebServExp.Destroy 25553
WebServExp.TWebServExp.GenerateXMLSchema 25554
WebServExp.TWebServExp.GetWebServExp 25554
WebServExp.TWebServExp.GetWSDLForInterface 25555
WebServExp.TWebServExp.ImportLocations 25556
WebServExp.TWebServExp.ImportNames 25556
WebServExp.TWebServExp.OnAfterPublishingWSDL 25557
WebServExp.TWebServExp.OnBeforePublishingTypes 25557
WebServExp.TWebServExp.OnPublishingType 25557
WebServExp.TWebServExp.ServiceAddress 25556
WebServExp.TWebServExp.TargetNameSpace 25556
WebServExp.TWebServExp.WSDLElements 25557
WebSess 25569
WebSess namespace 25569
Classes 25570
Structs, Records, Enums 25601
Types 25602
WebSess.TCustomSessionsService 25570
WebSess.TCustomSessionsService.CheckStatusChange 25575
WebSess.TCustomSessionsService.Clear 25575
WebSess.TCustomSessionsService.Connect 25575
WebSess.TCustomSessionsService.ConnectActive 25576
WebSess.TCustomSessionsService.Create 25574
WebSess.TCustomSessionsService.DefaultTimeout 25580
WebSess.TCustomSessionsService.DoEndSession 25576
WebSess.TCustomSessionsService.DoStartSession 25577
WebSess.TCustomSessionsService.GetSessionIDs 25577
WebSess.TCustomSessionsService.LoadFromFile 25577
WebSess.TCustomSessionsService.LoadFromStream 25578
WebSess.TCustomSessionsService.MaxSessions 25581
WebSess.TCustomSessionsService.NewSession 25578
WebSess.TCustomSessionsService.NotifyStatusChange 25579
WebSess.TCustomSessionsService.OnEndSession 25581
WebSess.TCustomSessionsService.OnStartSession 25582
WebSess.TCustomSessionsService.SaveToFile 25579
WebSess.TCustomSessionsService.SaveToStream 25580
WebSess.TEndSessionEvent 25602
WebSess.TEndSessionEvent type 25602
WebSess.TEndSessionReason 25601
WebSess.TEndSessionReason enumeration 25601
WebSess.TSessionsService 25582
WebSess.TSessionsService.DefaultTimeout 25586
WebSess.TSessionsService.MaxSessions 25587
WebSess.TSessionsService.OnEndSession 25587
WebSess.TSessionsService.OnStartSession 25588
WebSess.TStartSessionEvent 25602
WebSess.TStartSessionEvent type 25602
WebSess.TWebSessionIDsImpl 25588
WebSess.TWebSessionIDsImpl.Create 25590
WebSess.TWebSessionIDsImpl.Destroy 25590
WebSess.TWebSessionIDsImpl.GetID 25591
WebSess.TWebSessionIDsImpl.GetIDCount 25591
WebSess.TWebSessionImpl 25592
WebSess.TWebSessionImpl.Create 25594
WebSess.TWebSessionImpl.Destroy 25594
WebSess.TWebSessionImpl.ExpirationTime 25599

WebSess.TWebSessionImpl.Expired 25599
WebSess.TWebSessionImpl.GetExpirationTime 25595
WebSess.TWebSessionImpl.GetExpired 25595
WebSess.TWebSessionImpl.GetID 25595
WebSess.TWebSessionImpl.GetIsActive 25596
WebSess.TWebSessionImpl.GetItemCount 25596
WebSess.TWebSessionImpl.GetItems 25596
WebSess.TWebSessionImpl.GetStartTime 25596
WebSess.TWebSessionImpl.GetTerminated 25597
WebSess.TWebSessionImpl.GetTimeoutMinutes 25597
WebSess.TWebSessionImpl.GetTouchTime 25597
WebSess.TWebSessionImpl.GetValue 25597
WebSess.TWebSessionImpl.ID 25599
WebSess.TWebSessionImpl.SetTimeoutMinutes 25597
WebSess.TWebSessionImpl.SetValue 25598
WebSess.TWebSessionImpl.StartTime 25599
WebSess.TWebSessionImpl.Terminate 25598
WebSess.TWebSessionImpl.Terminated 25599
WebSess.TWebSessionImpl.TimeoutMinutes 25600
WebSess.TWebSessionImpl.TouchTime 25600
WebSess.TWebSessionImpl.Values 25600
WebUsers 25603
WebUsers namespace 25603
Classes 25604
Structs, Records, Enums 25633
Types 25634
WebUsers.EUserIDNotFoundException 25604
WebUsers.EValidateUserException 25604
WebUsers.EWebUsersListException 25605
WebUsers.TCheckAccessRightsEvent 25634
WebUsers.TCheckAccessRightsEvent type 25634
WebUsers.TCheckAccessRightsHandledEvent 25634
WebUsers.TCheckAccessRightsHandledEvent type 25634
WebUsers.TCustomWebUserList 25606
WebUsers.TCustomWebUserList.CheckAccessRights 25610
WebUsers.TCustomWebUserList.Create 25609
WebUsers.TCustomWebUserList.Destroy 25610
WebUsers.TCustomWebUserList.LoadFromFile 25611
WebUsers.TCustomWebUserList.OnAfterCheckAccessRights 25613
WebUsers.TCustomWebUserList.OnAfterValidateUser 25613
WebUsers.TCustomWebUserList.OnBeforeCheckAccessRights 25613
WebUsers.TCustomWebUserList.OnBeforeValidateUser 25614
WebUsers.TCustomWebUserList.OnUserIDNotFound 25614
WebUsers.TCustomWebUserList.OnValidateUserError 25614
WebUsers.TCustomWebUserList.SaveToFile 25611
WebUsers.TCustomWebUserList.UserItems 25612
WebUsers.TCustomWebUserList.Users 25612
WebUsers.TCustomWebUserList.ValidateUser 25611
WebUsers.TValidateUserError 25633
WebUsers.TValidateUserError enumeration 25633
WebUsers.TValidateUserErrorEvent 25635
WebUsers.TValidateUserErrorEvent type 25635
WebUsers.TValidateUserEvent 25635
WebUsers.TValidateUserEvent type 25635
WebUsers.TValidateUserHandledEvent 25635
WebUsers.TValidateUserHandledEvent type 25635
WebUsers.TWebUserItem 25615
WebUsers.TWebUserItem.AccessRights 25618
WebUsers.TWebUserItem.AccessRightsStrings 25619
WebUsers.TWebUserItem.Assign 25617
WebUsers.TWebUserItem.CheckRights 25618
WebUsers.TWebUserItem.Create 25616
WebUsers.TWebUserItem.Destroy 25617
WebUsers.TWebUserItem.Password 25619
WebUsers.TWebUserItem.UserID 25619
WebUsers.TWebUserItem.UserName 25620
WebUsers.TWebUserItems 25620
WebUsers.TWebUserItems.Create 25622
WebUsers.TWebUserItems.FindUserID 25623
WebUsers.TWebUserItems.FindUserName 25623
WebUsers.TWebUserItems.GetAttr 25623
WebUsers.TWebUserItems.GetAttrCount 25624
WebUsers.TWebUserItems.GetItemAttr 25624
WebUsers.TWebUserItems.GetOwner 25624
WebUsers.TWebUserItems.Items 25625
WebUsers.TWebUserList 25625
WebUsers.TWebUserList.OnAfterCheckAccessRights 25630
WebUsers.TWebUserList.OnAfterValidateUser 25630

WebUsers.TWebUserList.OnBeforeCheckAccessRights 25630
WebUsers.TWebUserList.OnBeforeValidateUser 25631
WebUsers.TWebUserList.OnUserIDNotFound 25631
WebUsers.TWebUserList.OnValidateUserError 25631
WebUsers.TWebUserList.UserItems 25629
WeImport enumeration member 25925
weMessage enumeration member 25925
weOperation enumeration member 25925
wePart enumeration member 25925
wePortType enumeration member 25925
weService enumeration member 25925
weServiceImpl enumeration member 25925
weServiceIntf enumeration member 25925
weTypes enumeration member 25925
WideString class 22377
!= 22387
!= operator 22387
& 22387
[] 22388
[] operator 22388
~WideString 22378
+ 22388
+= 22389
< 22389
< operator 22389
<= 22390
<= operator 22390
= 22390
= operator 22390
== 22391
== operator 22391
> 22392
> operator 22392
>= 22392
>= operator 22392
about WideString class 22377
Attach 22381
BSTR 22393
c_bstr 22382
Copy 22382
Copy method 22382
data 22383
Delete 22383
Detach 22383
Empty 22384
Insert 22384
IsEmpty 22385
Length 22385
Pos 22385
SetLength 22385
SubString 22386
WideString 22380
WideString constructor 22379
WideString Members 22377
WideString methods 22381
WideString operators 22386
wpLoginRequired enumeration member 25022
wpPublished enumeration member 25022
wrAbandoned enumeration member 22078
wrError enumeration member 22078
wrOff enumeration member 17092
wrPhysical enumeration member 17092
wrSignaled enumeration member 22078
wrTimeout enumeration member 22078
wrVirtual enumeration member 17092
WSDLBind 25637
WSDLBind namespace 25637
Classes 25640
Constants 25910
Functions 25908
Interfaces 25815
WSDLBind.GetDefinition 25908
WSDLBind.GetDefinition function 25908
WSDLBind.IBinding 25815
WSDLBind.IBinding.BindingOperations 25819
WSDLBind.IBinding.GetBindingOperations 25817
WSDLBind.IBinding.GetName 25818
WSDLBind.IBinding.GetType_ 25818

WSDLBind.IBinding.Name 25819
WSDLBind.IBinding.SetName 25818
WSDLBind.IBinding.SetType_ 25818
WSDLBind.IBinding.Type_ 25819
WSDLBind.IBindingInput 25819
WSDLBind.IBindingOperation 25820
WSDLBind.IBindingOperation.Faults 25823
WSDLBind.IBindingOperation.GetFaults 25822
WSDLBind.IBindingOperation.GetInput 25822
WSDLBind.IBindingOperation.GetName 25822
WSDLBind.IBindingOperation.GetOutput 25822
WSDLBind.IBindingOperation.Input 25823
WSDLBind.IBindingOperation.Name 25824
WSDLBind.IBindingOperation.Output 25824
WSDLBind.IBindingOperation.SetName 25823
WSDLBind.IBindingOperations 25824
WSDLBind.IBindingOperations.Add 25826
WSDLBind.IBindingOperations.BindingOperation 25827
WSDLBind.IBindingOperations.GetBindingOperation 25826
WSDLBind.IBindingOperations.Insert 25827
WSDLBind.IBindingOutput 25827
WSDLBind.IBindings 25828
WSDLBind.IBindings.Add 25830
WSDLBind.IBindings.Binding 25831
WSDLBind.IBindings.GetBinding 25830
WSDLBind.IBindings.Insert 25830
WSDLBind.IDefinition 25831
WSDLBind.IDefinition.Bindings 25836
WSDLBind.IDefinition.GetBindings 25833
WSDLBind.IDefinition.GetImports 25833
WSDLBind.IDefinition.GetMessages 25834
WSDLBind.IDefinition.GetName 25834
WSDLBind.IDefinition.GetPortTypes 25834
WSDLBind.IDefinition.GetServices 25834
WSDLBind.IDefinition.GetTargetNamespace 25834
WSDLBind.IDefinition.GetTypes 25835
WSDLBind.IDefinition.Imports 25836
WSDLBind.IDefinition.Messages 25836
WSDLBind.IDefinition.Name 25836
WSDLBind.IDefinition.PortTypes 25836
WSDLBind.IDefinition.Services 25837
WSDLBind.IDefinition.SetName 25835
WSDLBind.IDefinition.SetTargetNamespace 25835
WSDLBind.IDefinition.TargetNamespace 25837
WSDLBind.IDefinition.Types 25837
WSDLBind.IDocumentation 25837
WSDLBind.IDocumentationCollection 25838
WSDLBind.IDocumentationCollection.Add 25840
WSDLBind.IDocumentationCollection.Documentation 25841
WSDLBind.IDocumentationCollection.GetDocumentation 25840
WSDLBind.IDocumentationCollection.Insert 25840
WSDLBind.IDocumented 25841
WSDLBind.IDocumented.Documentation 25843
WSDLBind.IDocumented.GetDocumentationCollection 25843
WSDLBind.IFault 25843
WSDLBind.IFaults 25844
WSDLBind.IFaults.Add 25846
WSDLBind.IFaults.Fault 25847
WSDLBind.IFaults.GetFault 25846
WSDLBind.IFaults.Insert 25846
WSDLBind.IImport 25847
WSDLBind.IImport.GetLocation 25849
WSDLBind.IImport.GetNamespace 25849
WSDLBind.IImport.Location 25850
WSDLBind.IImport.Namespace 25850
WSDLBind.IImport.SetLocation 25849
WSDLBind.IImport.SetNamespace 25849
WSDLBind.IImports 25850
WSDLBind.IImports.Add 25852
WSDLBind.IImports.GetImport 25853
WSDLBind.IImports.Import 25853
WSDLBind.IImports.Insert 25853
WSDLBind.IMessage 25853
WSDLBind.IMessage.GetName 25855
WSDLBind.IMessage.GetParts 25856
WSDLBind.IMessage.Name 25856
WSDLBind.IMessage.Parts 25856
WSDLBind.IMessage.SetName 25856

WSDLBind.IMessages 25857
WSDLBind.IMessages.Add 25859
WSDLBind.IMessages.GetMessage 25859
WSDLBind.IMessages.Insert 25859
WSDLBind.IMessages.Message 25860
WSDLBind.IOperation 25860
WSDLBind.IOperation.Faults 25863
WSDLBind.IOperation.GetFaults 25862
WSDLBind.IOperation.GetInput 25862
WSDLBind.IOperation.GetName 25862
WSDLBind.IOperation.GetOutput 25863
WSDLBind.IOperation.Input 25863
WSDLBind.IOperation.Name 25864
WSDLBind.IOperation.Output 25864
WSDLBind.IOperation.SetName 25863
WSDLBind.IOperations 25864
WSDLBind.IOperations.Add 25866, 25867
WSDLBind.IOperations.GetOperation 25867
WSDLBind.IOperations.Insert 25867
WSDLBind.IOperations.Operation 25867
WSDLBind.IParam 25868
WSDLBind.IParam.GetMessage 25870
WSDLBind.IParam.GetName 25870
WSDLBind.IParam.Message 25871
WSDLBind.IParam.Name 25871
WSDLBind.IParam.SetMessage 25870
WSDLBind.IParam.SetName 25870
WSDLBind.IPart 25871
WSDLBind.IPart.Element 25875
WSDLBind.IPart.GetElement 25873
WSDLBind.IPart.GetName 25874
WSDLBind.IPart.GetType_ 25874
WSDLBind.IPart.Name 25875
WSDLBind.IPart.SetElement 25874
WSDLBind.IPart.SetName 25874
WSDLBind.IPart.SetType_ 25874
WSDLBind.IPart.Type_ 25875
WSDLBind.IParts 25876
WSDLBind.IParts.Add 25878
WSDLBind.IParts.GetPart 25878
WSDLBind.IParts.Insert 25878
WSDLBind.IParts.Part 25879
WSDLBind.IPort 25879
WSDLBind.IPort.Binding 25882
WSDLBind.IPort.GetBinding 25881
WSDLBind.IPort.GetName 25881
WSDLBind.IPort.Name 25882
WSDLBind.IPort.SetBinding 25881
WSDLBind.IPort.SetName 25881
WSDLBind.IPorts 25882
WSDLBind.IPorts.Add 25884
WSDLBind.IPorts.GetPort 25885
WSDLBind.IPorts.Insert 25885
WSDLBind.IPorts.Port 25885
WSDLBind.IPortType 25885
WSDLBind.IPortType.GetName 25887
WSDLBind.IPortType.GetOperations 25888
WSDLBind.IPortType.Name 25888
WSDLBind.IPortType.Operations 25888
WSDLBind.IPortType.SetName 25888
WSDLBind.IPortTypes 25889
WSDLBind.IPortTypes.Add 25891
WSDLBind.IPortTypes.GetPortType 25891
WSDLBind.IPortTypes.Insert 25891
WSDLBind.IPortTypes.PortType 25892
WSDLBind.IService 25892
WSDLBind.IService.GetName 25894
WSDLBind.IService.GetPorts 25894
WSDLBind.IService.Name 25895
WSDLBind.IService.Ports 25895
WSDLBind.IService.SetName 25894
WSDLBind.IServices 25895
WSDLBind.IServices.Add 25897
WSDLBind.IServices.GetService 25897
WSDLBind.IServices.Insert 25898
WSDLBind.IServices.Service 25898
WSDLBind.ITypes 25898
WSDLBind.ITypes.GetSchemaDefs 25900

qqqq

WSDLBind.ITypes.SchemaDefs 25900
WSDLBind.IWSDLDocument 25901
WSDLBind.IWSDLDocument.Definition 25904
WSDLBind.IWSDLDocument.GetDefinition 25903
WSDLBind.IWSDLDocument.SetDefinition 25903
WSDLBind.IXMLSchemaDefs 25905
WSDLBind.IXMLSchemaDefs.Add 25907
WSDLBind.IXMLSchemaDefs.GetXMLSchemaDef 25907
WSDLBind.IXMLSchemaDefs.SchemaDef 25907
WSDLBind.LoadWSDLDoc 25908
WSDLBind.LoadWSDLDoc function 25908
WSDLBind.NewWSDLDoc 25908
WSDLBind.NewWSDLDoc function 25908
WSDLBind.SBinding 25910
WSDLBind.SBinding constant 25910
WSDLBind.SDefinitions 25910
WSDLBind.SDefinitions constant 25910
WSDLBind.SFault 25911
WSDLBind.SFault constant 25911
WSDLBind.SImport 25911
WSDLBind.SImport constant 25911
WSDLBind.SInput 25911
WSDLBind.SInput constant 25911
WSDLBind.SLocation 25912
WSDLBind.SLocation constant 25912
WSDLBind.SMessage 25912
WSDLBind.SMessage constant 25912
WSDLBind.SNamespace 25912
WSDLBind.SNamespace constant 25912
WSDLBind.SOperation 25913
WSDLBind.SOperation constant 25913
WSDLBind.SOutput 25913
WSDLBind.SOutput constant 25913
WSDLBind.SPart 25913
WSDLBind.SPart constant 25913
WSDLBind.SPort 25914
WSDLBind.SPort constant 25914
WSDLBind.SPortType 25914
WSDLBind.SPortType constant 25914
WSDLBind.SService 25914
WSDLBind.SService constant 25914
WSDLBind.SType 25915
WSDLBind.SType constant 25915
WSDLBind.STypes 25915
WSDLBind.STypes constant 25915
WSDLBind.TBinding 25640
WSDLBind.TBinding.AfterConstruction 25646
WSDLBind.TBinding.GetBindingOperations 25646
WSDLBind.TBinding.GetName 25646
WSDLBind.TBinding.GetType_ 25647
WSDLBind.TBinding.SetName 25647
WSDLBind.TBinding.SetType_ 25647
WSDLBind.TBindingOperation 25647
WSDLBind.TBindingOperation.AfterConstruction 25653
WSDLBind.TBindingOperation.GetFaults 25653
WSDLBind.TBindingOperation.GetInput 25653
WSDLBind.TBindingOperation.GetName 25654
WSDLBind.TBindingOperation.GetOutput 25654
WSDLBind.TBindingOperation.SetName 25654
WSDLBind.TBindingOperations 25654
WSDLBind.TBindingOperations.Add 25660
WSDLBind.TBindingOperations.GetBindingOperation 25660
WSDLBind.TBindingOperations.Insert 25660
WSDLBind.TBindings 25661
WSDLBind.TBindings.Add 25666
WSDLBind.TBindings.GetBinding 25667
WSDLBind.TBindings.Insert 25667
WSDLBind.TDefinition 25667
WSDLBind.TDefinition.AfterConstruction 25673
WSDLBind.TDefinition.GetBindings 25673
WSDLBind.TDefinition.GetImports 25673
WSDLBind.TDefinition.GetMessages 25674
WSDLBind.TDefinition.GetName 25674
WSDLBind.TDefinition.GetPortTypes 25674
WSDLBind.TDefinition.GetServices 25674
WSDLBind.TDefinition.GetTargetNamespace 25675
WSDLBind.TDefinition.GetTypes 25675
WSDLBind.TDefinition.SetName 25675

WSDLBind.TDefinition.SetTargetNamespace 25675
WSDLBind.TDocumentation 25676
WSDLBind.TDocumentationCollection 25676
WSDLBind.TDocumentationCollection.Add 25682
WSDLBind.TDocumentationCollection.GetDocumentation 25682
WSDLBind.TDocumentationCollection.Insert 25682
WSDLBind.TDocumented 25683
WSDLBind.TDocumented.AfterConstruction 25688
WSDLBind.TDocumented.GetDocumentationCollection 25688
WSDLBind.TFault 25689
WSDLBind.TFaults 25689
WSDLBind.TFaults.Add 25695
WSDLBind.TFaults.GetFault 25695
WSDLBind.TFaults.Insert 25695
WSDLBind.TImport 25696
WSDLBind.TImport.GetLocation 25701
WSDLBind.TImport.GetNamespace 25701
WSDLBind.TImport.SetLocation 25701
WSDLBind.TImport.SetNamespace 25702
WSDLBind.TImports 25702
WSDLBind.TImports.Add 25708
WSDLBind.TImports.GetImport 25708
WSDLBind.TImports.Insert 25708
WSDLBind.TMessage 25708
WSDLBind.TMessage.AfterConstruction 25714
WSDLBind.TMessage.GetName 25714
WSDLBind.TMessage.GetParts 25714
WSDLBind.TMessage.SetName 25714
WSDLBind.TMessages 25715
WSDLBind.TMessages.Add 25720
WSDLBind.TMessages.GetMessage 25721
WSDLBind.TMessages.Insert 25721
WSDLBind.TOperation 25721
WSDLBind.TOperation.AfterConstruction 25727
WSDLBind.TOperation.GetFaults 25727
WSDLBind.TOperation.GetInput 25727
WSDLBind.TOperation.GetName 25727
WSDLBind.TOperation.GetOutput 25728
WSDLBind.TOperation.SetName 25728
WSDLBind.TOperations 25728
WSDLBind.TOperations.Add 25734
WSDLBind.TOperations.GetOperation 25735
WSDLBind.TOperations.Insert 25735
WSDLBind.TParam 25735
WSDLBind.TParam.GetMessage 25740
WSDLBind.TParam.GetName 25741
WSDLBind.TParam.SetMessage 25741
WSDLBind.TParam.SetName 25741
WSDLBind.TPart 25741
WSDLBind.TPart.GetElement 25747
WSDLBind.TPart.GetName 25747
WSDLBind.TPart.GetType_ 25747
WSDLBind.TPart.SetElement 25747
WSDLBind.TPart.SetName 25748
WSDLBind.TPart.SetType_ 25748
WSDLBind.TParts 25748
WSDLBind.TParts.Add 25754
WSDLBind.TParts.GetPart 25754
WSDLBind.TParts.Insert 25754
WSDLBind.TPort 25755
WSDLBind.TPort.GetBinding 25760
WSDLBind.TPort.GetName 25760
WSDLBind.TPort.SetBinding 25760
WSDLBind.TPort.SetName 25761
WSDLBind.TPorts 25761
WSDLBind.TPorts.Add 25767
WSDLBind.TPorts.GetPort 25767
WSDLBind.TPorts.Insert 25767
WSDLBind.TPortType 25767
WSDLBind.TPortType.AfterConstruction 25773
WSDLBind.TPortType.GetName 25773
WSDLBind.TPortType.GetOperations 25773
WSDLBind.TPortType.SetName 25773
WSDLBind.TPortTypes 25774
WSDLBind.TPortTypes.Add 25779
WSDLBind.TPortTypes.GetPortType 25780
WSDLBind.TPortTypes.Insert 25780
WSDLBind.TService 25780

ssssss

WSDLBind.TService.AfterConstruction 25785
WSDLBind.TService.GetName 25786
WSDLBind.TService.GetPorts 25786
WSDLBind.TService.SetName 25786
WSDLBind.TServices 25787
WSDLBind.TServices.Add 25792
WSDLBind.TServices.GetService 25793
WSDLBind.TServices.Insert 25793
WSDLBind.TTypes 25793
WSDLBind.TTypes.AfterConstruction 25798
WSDLBind.TTypes.GetSchemaDefs 25799
WSDLBind.TWSDLDocument 25799
WSDLBind.TWSDLDocument.AfterConstruction 25807
WSDLBind.TWSDLDocument.Definition 25808
WSDLBind.TWSDLDocument.GetDefinition 25807
WSDLBind.TWSDLDocument.LoadData 25807
WSDLBind.TWSDLDocument.SetDefinition 25807
WSDLBind.TXMLSchemaDefs 25808
WSDLBind.TXMLSchemaDefs.Add 25814
WSDLBind.TXMLSchemaDefs.GetXMLSchemaDef 25814
WSDLIntf 25917
WSDLIntf namespace 25917
Classes 25919
Constants 25928
Structs, Records, Enums 25925
Types 25927
WSDLIntf.Directives 25929
WSDLIntf.Directives constant 25929
WSDLIntf.DynArrayTypes 25929
WSDLIntf.DynArrayTypes constant 25929
WSDLIntf.Httpns 25930
WSDLIntf.Httpns constant 25930
WSDLIntf.Operators 25930
WSDLIntf.Operators constant 25930
WSDLIntf.OtherReservedWords 25930
WSDLIntf.OtherReservedWords constant 25930
WSDLIntf.ReservedWords 25931
WSDLIntf.ReservedWords constant 25931
WSDLIntf.SAddress 25931
WSDLIntf.SAddress constant 25931
WSDLIntf.SAnySimpleType 25932
WSDLIntf.SAnySimpleType constant 25932
WSDLIntf.SAnyType 25932
WSDLIntf.SAnyType constant 25932
WSDLIntf.SArray 25932
WSDLIntf.SArray constant 25932
WSDLIntf.SArrayOf 25933
WSDLIntf.SArrayOf constant 25933
WSDLIntf.SArrayType 25933
WSDLIntf.SArrayType constant 25933
WSDLIntf.SBinding 25933
WSDLIntf.SBinding constant 25933
WSDLIntf.SBody 25934
WSDLIntf.SBody constant 25934
WSDLIntf.ScalarArrayTypes 25934
WSDLIntf.ScalarArrayTypes constant 25934
WSDLIntf.SComplexType 25934
WSDLIntf.SComplexType constant 25934
WSDLIntf.SDefinitions 25935
WSDLIntf.SDefinitions constant 25935
WSDLIntf.SDynArray 25935
WSDLIntf.SDynArray constant 25935
WSDLIntf.SElement 25935
WSDLIntf.SElement constant 25935
WSDLIntf.SEncodingStyle 25936
WSDLIntf.SEncodingStyle constant 25936
WSDLIntf.SFault 25936
WSDLIntf.SFault constant 25936
WSDLIntf.SHeader 25936
WSDLIntf.SHeader constant 25936
WSDLIntf.SImport 25937
WSDLIntf.SImport constant 25937
WSDLIntf.SInOut 25937
WSDLIntf.SInOut constant 25937
WSDLIntf.SInput 25937
WSDLIntf.SInput constant 25937
WSDLIntf.SLocation 25938
WSDLIntf.SLocation constant 25938

WSDLIntf.SMessage 25938
WSDLIntf.SMessage constant 25938
WSDLIntf.SName 25938
WSDLIntf.SName constant 25938
WSDLIntf.SNameSpace 25939
WSDLIntf.SNameSpace constant 25939
WSDLIntf.SNsPrefix 25939
WSDLIntf.SNsPrefix constant 25939
WSDLIntf.Soapns 25939
WSDLIntf.Soapns constant 25939
WSDLIntf.SOperation 25940
WSDLIntf.SOperation constant 25940
WSDLIntf.SOutput 25940
WSDLIntf.SOutput constant 25940
WSDLIntf.SPart 25940
WSDLIntf.SPart constant 25940
WSDLIntf.SParts 25941
WSDLIntf.SParts constant 25941
WSDLIntf.SPort 25941
WSDLIntf.SPort constant 25941
WSDLIntf.SPortType 25941
WSDLIntf.SPortType constant 25941
WSDLIntf.SRequest 25942
WSDLIntf.SRequest constant 25942
WSDLIntf.SResponse 25942
WSDLIntf.SResponse constant 25942
WSDLIntf.SReturn 25942
WSDLIntf.SReturn constant 25942
WSDLIntf.SSchema 25943
WSDLIntf.SSchema constant 25943
WSDLIntf.SService 25943
WSDLIntf.SService constant 25943
WSDLIntf.SSoapAction 25943
WSDLIntf.SSoapAction constant 25943
WSDLIntf.SStyle 25944
WSDLIntf.SStyle constant 25944
WSDLIntf.STns 25944
WSDLIntf.STns constant 25944
WSDLIntf.STransport 25944
WSDLIntf.STransport constant 25944
WSDLIntf.SType 25945
WSDLIntf.SType constant 25945
WSDLIntf.STypes 25945
WSDLIntf.STypes constant 25945
WSDLIntf.SUnknown 25945
WSDLIntf.SUnknown constant 25945
WSDLIntf.SUse 25946
WSDLIntf.SUse constant 25946
WSDLIntf.SVerb 25946
WSDLIntf.SVerb constant 25946
WSDLIntf.tns 25946
WSDLIntf.tns constant 25946
WSDLIntf.TWebServiceBindingType 25925
WSDLIntf.TWebServiceBindingType enumeration 25925
WSDLIntf.TWideStrings 25919
WSDLIntf.TWideStrings.Add 25921
WSDLIntf.TWideStrings.Clear 25921
WSDLIntf.TWideStrings.Count 25922
WSDLIntf.TWideStrings.Create 25920
WSDLIntf.TWideStrings.Destroy 25920
WSDLIntf.TWideStrings.IndexOf 25922
WSDLIntf.TWideStrings.IndexOfIgnoreCase 25922
WSDLIntf.TWideStrings.Insert 25923
WSDLIntf.TWideStrings.Strings 25923
WSDLIntf.TWSDLElement 25925
WSDLIntf.TWSDLElement enumeration 25925
WSDLIntf.TWSDLElements 25927
WSDLIntf.TWSDLElements type 25927
WSDLIntf.TWString 25926
WSDLIntf.TWString record 25926
WSDLIntf.Wsdlns 25947
WSDLIntf.Wsdlns constant 25947
WSDLIntf.xsdns 25947
WSDLIntf.xsdns constant 25947
WSDLItems 25949
WSDLItems namespace 25949
Classes 25951
Functions 26000

Interfaces 25989
Structs, Records, Enums 26002
Types 26003
WSDLItems.HasDefinition 26000
WSDLItems.HasDefinition function 26000
WSDLItems.IHeaderInfo 25989
WSDLItems.IHeaderInfo.EncodingStyle 25993
WSDLItems.IHeaderInfo.GetEncodingStyle 25990
WSDLItems.IHeaderInfo.GetName 25990
WSDLItems.IHeaderInfo.GetNamespace 25991
WSDLItems.IHeaderInfo.GetPart 25991
WSDLItems.IHeaderInfo.GetRequired 25991
WSDLItems.IHeaderInfo.GetUse 25991
WSDLItems.IHeaderInfo.Name 25993
WSDLItems.IHeaderInfo.Namespace 25993
WSDLItems.IHeaderInfo.Part 25994
WSDLItems.IHeaderInfo.Required 25994
WSDLItems.IHeaderInfo.SetEncodingStyle 25991
WSDLItems.IHeaderInfo.SetName 25992
WSDLItems.IHeaderInfo.SetNamespace 25992
WSDLItems.IHeaderInfo.SetPart 25992
WSDLItems.IHeaderInfo.SetRequired 25992
WSDLItems.IHeaderInfo.SetUse 25993
WSDLItems.IHeaderInfo.Use 25994
WSDLItems.IHeaderInfoArray 26003
WSDLItems.IHeaderInfoArray type 26003
WSDLItems.IQualifiedName 25994
WSDLItems.IQualifiedName.GetName 25995
WSDLItems.IQualifiedName.GetNamespace 25996
WSDLItems.IQualifiedName.Name 25997
WSDLItems.IQualifiedName.Namespace 25997
WSDLItems.IQualifiedName.SetName 25996
WSDLItems.IQualifiedName.SetNamespace 25996
WSDLItems.IQualifiedNameArray 26003
WSDLItems.IQualifiedNameArray type 26003
WSDLItems.IterateImportOptions 26002
WSDLItems.IterateImportOptions enumeration 26002
WSDLItems.IWSDLItems 25998
WSDLItems.IWSDLItems.GetWSDLDoc 25999
WSDLItems.IWSDLItems.GetWSDLItems 25999
WSDLItems.IWSDLItems.Load 25999
WSDLItems.NewQualifiedName 26000
WSDLItems.NewQualifiedName function 26000
WSDLItems.TWSDLItems 25951
WSDLItems.TWSDLItems.CompareName 25962
WSDLItems.TWSDLItems.Create 25960
WSDLItems.TWSDLItems.GetBindingForServicePort 25962
WSDLItems.TWSDLItems.GetBindingOfType 25963
WSDLItems.TWSDLItems.GetBindingOperationNode 25963
WSDLItems.TWSDLItems.GetBindings 25964
WSDLItems.TWSDLItems.GetBindingType 25964
WSDLItems.TWSDLItems.GetHttpBindingAttribute 25965
WSDLItems.TWSDLItems.GetImports 25965
WSDLItems.TWSDLItems.GetLocationForImport 25966
WSDLItems.TWSDLItems.GetMessageNode 25966
WSDLItems.TWSDLItems.GetMessages 25967
WSDLItems.TWSDLItems.GetName 25967
WSDLItems.TWSDLItems.GetOperationNode 25967
WSDLItems.TWSDLItems.GetOperations 25968
WSDLItems.TWSDLItems.GetOperationsForBinding 25968
WSDLItems.TWSDLItems.GetPartNode 25969
WSDLItems.TWSDLItems.GetParts 25969
WSDLItems.TWSDLItems.GetPartsForOperation 25970
WSDLItems.TWSDLItems.GetPortsForService 25970
WSDLItems.TWSDLItems.GetPortTypeNode 25971
WSDLItems.TWSDLItems.GetPortTypes 25971
WSDLItems.TWSDLItems.GetSchemaNode 25972
WSDLItems.TWSDLItems.GetSchemas 25972
WSDLItems.TWSDLItems.GetServiceAndPortOfBinding 25973
WSDLItems.TWSDLItems.GetServiceNode 25973
WSDLItems.TWSDLItems.GetServices 25974
WSDLItems.TWSDLItems.GetSoapAction 25974
WSDLItems.TWSDLItems.GetSoapActionList 25975
WSDLItems.TWSDLItems.GetSoapAddressForServicePort 25975
WSDLItems.TWSDLItems.GetSoapBindingAttribute 25976
WSDLItems.TWSDLItems.GetSoapBindingInputNode 25976
WSDLItems.TWSDLItems.GetSoapBindingIONode 25977
WSDLItems.TWSDLItems.GetSoapBindingOutputNode 25978

WSDLItems.TWSDLItems.GetSoapBodyAttribute 25978
WSDLItems.TWSDLItems.GetSoapBodyInputAttribute 25979
WSDLItems.TWSDLItems.GetSoapBodyNamespace 25980
WSDLItems.TWSDLItems.GetSoapBodyOutputAttribute 25980
WSDLItems.TWSDLItems.GetSoapHeaders 25981
WSDLItems.TWSDLItems.GetSoapInputHeaders 25982
WSDLItems.TWSDLItems.GetSoapOperationAttribute 25982
WSDLItems.TWSDLItems.GetSoapOperationStyle 25983
WSDLItems.TWSDLItems.GetSoapOutputHeaders 25983
WSDLItems.TWSDLItems.GetTargetNamespace 25984
WSDLItems.TWSDLItems.GetTargetNamespacePrefix 25985
WSDLItems.TWSDLItems.GetWSDLDoc 25985
WSDLItems.TWSDLItems.GetWSDLItems 25985
WSDLItems.TWSDLItems.HasTypesNode 25986
WSDLItems.TWSDLItems.IsPortTypeHTTPBound 25986
WSDLItems.TWSDLItems.IterateImports 25986
WSDLItems.TWSDLItems.Load 25987
WSDLItems.TWSDLItems.StreamLoader 25987
WSDLItems.TWSDLItems.TargetNamespace 25988
WSDLItems.TWSDLIterateProc 26003
WSDLItems.TWSDLIterateProc type 26003
WSDLNode 26005
WSDLNode namespace 26005
    Classes 26006
    Functions 26022
WSDLNode.ActivateWSDL 26022
WSDLNode.ActivateWSDL function 26022
WSDLNode.EWSDLLoadException 26006
WSDLNode.TWSDLClientNode 26006
WSDLNode.TWSDLClientNode.BeforeExecute 26010
WSDLNode.TWSDLClientNode.Execute 26010, 26011
WSDLNode.TWSDLClientNode.GetMimeBoundary 26011
WSDLNode.TWSDLClientNode.Notification 26012
WSDLNode.TWSDLClientNode.SetMimeBoundary 26012
WSDLNode.TWSDLClientNode.TransportNode 26013
WSDLNode.TWSDLClientNode.WSDLView 26013
WSDLNode.TWSDLView 26013
WSDLNode.TWSDLView.Activate 26017
WSDLNode.TWSDLView.IntfInfo 26017
WSDLNode.TWSDLView.Operation 26018
WSDLNode.TWSDLView.Password 26018
WSDLNode.TWSDLView.Port 26019
WSDLNode.TWSDLView.PortType 26019
WSDLNode.TWSDLView.Proxy 26020
WSDLNode.TWSDLView.Service 26020
WSDLNode.TWSDLView.SetDesignState 26017
WSDLNode.TWSDLView.UserName 26020
WSDLNode.TWSDLView.WSDL 26020
WSDLPub 26023
WSDLPub namespace 26023
    Classes 26024
    Functions 26044
    Interfaces 26042
    Structs, Records, Enums 26045
    Types 26046
WSDLPub.IWSDLPublish 26042
WSDLPub.IWSDLPublish.GetPortTypeList 26043
WSDLPub.IWSDLPublish.GetTypeSystemsList 26043
WSDLPub.IWSDLPublish.GetWSDLForPortType 26043
WSDLPub.IWSDLPublish.GetXSDForTypeSystem 26043
WSDLPub.SOAPPublishOptions 26045
WSDLPub.SOAPPublishOptions enumeration 26045
WSDLPub.TBeforePublishingWSDLEvent 26046
WSDLPub.TBeforePublishingWSDLEvent type 26046
WSDLPub.TSOAPPublishOptions 26046
WSDLPub.TSOAPPublishOptions type 26046
WSDLPub.TWSDLHTMLPublish 26024
WSDLPub.TWSDLHTMLPublish.AddInterfaceList 26029
WSDLPub.TWSDLHTMLPublish.AddPortList 26029
WSDLPub.TWSDLHTMLPublish.AdminEnabled 26034
WSDLPub.TWSDLHTMLPublish.Create 26028
WSDLPub.TWSDLHTMLPublish.Destroy 26028
WSDLPub.TWSDLHTMLPublish.DispatchEnabled 26030
WSDLPub.TWSDLHTMLPublish.DispatchMask 26030
WSDLPub.TWSDLHTMLPublish.DispatchMethodType 26030
WSDLPub.TWSDLHTMLPublish.DispatchRequest 26031
WSDLPub.TWSDLHTMLPublish.GetHostScriptBaseURL 26031
WSDLPub.TWSDLHTMLPublish.GetTargetNamespace 26031

wwww

WSDLPub.TWSDLHTMLPublish.HandleRequest 26032
WSDLPub.TWSDLHTMLPublish.OnAfterPublishingWSDL 26035
WSDLPub.TWSDLHTMLPublish.OnBeforePublishingTypes 26036
WSDLPub.TWSDLHTMLPublish.OnBeforePublishingWSDL 26036
WSDLPub.TWSDLHTMLPublish.OnPublishingType 26037
WSDLPub.TWSDLHTMLPublish.PublishOptions 26034
WSDLPub.TWSDLHTMLPublish.ServiceInfo 26032
WSDLPub.TWSDLHTMLPublish.TargetNamespace 26035
WSDLPub.TWSDLHTMLPublish.UpdatePortList 26033
WSDLPub.TWSDLHTMLPublish.WebDispatch 26035
WSDLPub.TWSDLHTMLPublish.WSILInfo 26033
WSDLPub.TWSDLPublish 26037
WSDLPub.TWSDLPublish.GetPortTypeEntries 26039
WSDLPub.TWSDLPublish.GetPortTypeList 26039
WSDLPub.TWSDLPublish.GetTypeSystemsList 26039
WSDLPub.TWSDLPublish.GetWSDLForPortType 26040
WSDLPub.TWSDLPublish.GetXSDForTypeSystem 26040
WSDLPub.TWSDLPublish.OnAfterPublishingWSDL 26041
WSDLPub.TWSDLPublish.OnBeforePublishingTypes 26041
WSDLPub.TWSDLPublish.OnBeforePublishingWSDL 26041
WSDLPub.TWSDLPublish.OnPublishingType 26041
WSDLPub.TWSDLPublish.TargetNamespace 26040
WSDLPub.WSDLPubFactory 26044
WSDLPub.WSDLPubFactory function 26044
wsMaximized enumeration member 14368
wsMinimized enumeration member 14368
wsNormal enumeration member 14368

X

xaAbortRetaining enumeration member 1308
xaCommitRetaining enumeration member 1308
xdsFile enumeration member 26245
xdsNone enumeration member 26245
xdsStream enumeration member 26245
xdsXMLData enumeration member 26245
xdsXMLProperty enumeration member 26245
xeNewMetaData enumeration member 17510
xePivot enumeration member 17510
xeSourceChange enumeration member 17510
xeStateChanged enumeration member 17510
xeSummaryChanged enumeration member 17510
xetEBCDICLike enumeration member 26461
xetUCS_4BE enumeration member 26461
xetUCS_4LE enumeration member 26461
xetUCS_4Like enumeration member 26461
xetUCS_4Order2134 enumeration member 26461
xetUCS_4Order3412 enumeration member 26461
xetUnknown enumeration member 26461
xetUTF_16BE enumeration member 26461
xetUTF_16BELike enumeration member 26461
xetUTF_16LE enumeration member 26461
xetUTF_16LELike enumeration member 26461
xetUTF_8 enumeration member 26461
xetUTF_8Like enumeration member 26461
xilCUSTOM enumeration member 20369
xilDIRTYREAD enumeration member 20369
xilREADCOMMITTED enumeration member 20369
xilREPEATABLEREAD enumeration member 20369
XMLBrokr 26047
XMLBrokr namespace 26047
- Classes 26049
- Constants 26092
- Functions 26088
- Interfaces 26086
- Structs, Records, Enums 26089
- Types 26090

XMLBrokr.FormatXML 26088
XMLBrokr.FormatXML function 26088
XMLBrokr.INotifyConnectionChange 26086
XMLBrokr.INotifyConnectionChange.ConnectionChange 26087
XMLBrokr.sPostDelta 26092
XMLBrokr.sPostDelta constant 26092
XMLBrokr.sProducer 26092
XMLBrokr.sProducer constant 26092
XMLBrokr.sRedirect 26092
XMLBrokr.sRedirect constant 26092

XMLBrokr.TGetErrorResponseEvent 26090
XMLBrokr.TGetErrorResponseEvent type 26090
XMLBrokr.TGetResponseEvent 26090
XMLBrokr.TGetResponseEvent type 26090
XMLBrokr.TRequestRecordsEvent 26091
XMLBrokr.TRequestRecordsEvent type 26091
XMLBrokr.TRequestUpdateEvent 26091
XMLBrokr.TRequestUpdateEvent type 26091
XMLBrokr.TXMLBroker 26049
XMLBrokr.TXMLBroker.AddNotify 26055
XMLBrokr.TXMLBroker.AfterDispatch 26077
XMLBrokr.TXMLBroker.ApplyXMLUpdates 26056
XMLBrokr.TXMLBroker.AppServer 26070
XMLBrokr.TXMLBroker.AS_ApplyUpdates 26056
XMLBrokr.TXMLBroker.AS_FetchParams 26057
XMLBrokr.TXMLBroker.AS_GetRecords 26057
XMLBrokr.TXMLBroker.BeforeDispatch 26078
XMLBrokr.TXMLBroker.CheckInactive 26057
XMLBrokr.TXMLBroker.Connected 26070
XMLBrokr.TXMLBroker.Create 26054
XMLBrokr.TXMLBroker.Destroy 26054
XMLBrokr.TXMLBroker.DispatchEnabled 26057
XMLBrokr.TXMLBroker.DispatchMask 26058
XMLBrokr.TXMLBroker.DispatchMethodType 26058
XMLBrokr.TXMLBroker.DispatchRequest 26058
XMLBrokr.TXMLBroker.FetchParams 26058
XMLBrokr.TXMLBroker.GetAppServer 26059
XMLBrokr.TXMLBroker.GetConnected 26059
XMLBrokr.TXMLBroker.GetDelta 26059
XMLBrokr.TXMLBroker.GetErrorCount 26059
XMLBrokr.TXMLBroker.GetErrorResponse 26060
XMLBrokr.TXMLBroker.GetErrors 26060
XMLBrokr.TXMLBroker.GetHasAppServer 26060
XMLBrokr.TXMLBroker.GetNotify 26060
XMLBrokr.TXMLBroker.GetNotifyCount 26061
XMLBrokr.TXMLBroker.GetProducerName 26061
XMLBrokr.TXMLBroker.GetRedirect 26061
XMLBrokr.TXMLBroker.GetResponse 26061
XMLBrokr.TXMLBroker.GetXMLRecords 26062
XMLBrokr.TXMLBroker.GetXMLStream 26062
XMLBrokr.TXMLBroker.HasAppServer 26071
XMLBrokr.TXMLBroker.HTMLSubmitFormName 26063
XMLBrokr.TXMLBroker.MasterRowSetVarName 26063
XMLBrokr.TXMLBroker.MaxErrors 26071
XMLBrokr.TXMLBroker.MaxRecords 26072
XMLBrokr.TXMLBroker.Notification 26064
XMLBrokr.TXMLBroker.Notify 26073
XMLBrokr.TXMLBroker.NotifyCount 26073
XMLBrokr.TXMLBroker.OnGetErrorResponse 26078
XMLBrokr.TXMLBroker.OnGetResponse 26079
XMLBrokr.TXMLBroker.OnRequestRecords 26080
XMLBrokr.TXMLBroker.OnRequestUpdate 26080
XMLBrokr.TXMLBroker.Params 26073
XMLBrokr.TXMLBroker.ProviderName 26074
XMLBrokr.TXMLBroker.ReconcileProducer 26075
XMLBrokr.TXMLBroker.RemoteServer 26076
XMLBrokr.TXMLBroker.RemoveNotify 26064
XMLBrokr.TXMLBroker.RequestRecords 26064
XMLBrokr.TXMLBroker.RequestUpdate 26065
XMLBrokr.TXMLBroker.RowSetVarName 26066
XMLBrokr.TXMLBroker.SendConnectEvent 26067
XMLBrokr.TXMLBroker.SetAppServer 26067
XMLBrokr.TXMLBroker.SetConnected 26067
XMLBrokr.TXMLBroker.SetParams 26067
XMLBrokr.TXMLBroker.SetProvider 26068
XMLBrokr.TXMLBroker.SetProviderName 26068
XMLBrokr.TXMLBroker.SetReconcileProducer 26068
XMLBrokr.TXMLBroker.SetRemoteServer 26069
XMLBrokr.TXMLBroker.SetWebDispatch 26069
XMLBrokr.TXMLBroker.SubmitFormVarName 26069
XMLBrokr.TXMLBroker.WebDispatch 26076
XMLBrokr.TXMLOption 26089
XMLBrokr.TXMLOption enumeration 26089
XMLBrokr.TXMLOptions 26091
XMLBrokr.TXMLOptions type 26091
XMLBrokr.TXMLParams 26081
XMLBrokr.TXMLParams.AssignStrings 26084
XMLBrokr.TXMLParams.AssignTo 26085

yyyy

XMLDoc 26095
XMLDoc namespace 26095
Classes 26097
Functions 26238
Interfaces 26228
Structs, Records, Enums 26243
Types 26247
Variables 26251
XMLDoc.CheckEncoding 26238
XMLDoc.CheckEncoding function 26238
XMLDoc.CreateDOMNode 26238
XMLDoc.CreateDOMNode function 26238
XMLDoc.DetectCharEncoding 26239
XMLDoc.DetectCharEncoding function 26239
XMLDoc.FormatXMLData 26240
XMLDoc.FormatXMLData function 26240
XMLDoc.IXMLDocumentAccess 26228
XMLDoc.IXMLDocumentAccess.DocumentObject 26229
XMLDoc.IXMLDocumentAccess.DOMPersist 26230
XMLDoc.IXMLDocumentAccess.GetDocumentObject 26229
XMLDoc.IXMLDocumentAccess.GetDOMPersist 26229
XMLDoc.IXMLNodeAccess 26230
XMLDoc.IXMLNodeAccess.AddChild 26232
XMLDoc.IXMLNodeAccess.CheckTextNode 26233
XMLDoc.IXMLNodeAccess.ChildNodeClasses 26237
XMLDoc.IXMLNodeAccess.ClearDocumentRef 26233
XMLDoc.IXMLNodeAccess.CreateAttributeNode 26233
XMLDoc.IXMLNodeAccess.CreateChildNode 26233
XMLDoc.IXMLNodeAccess.CreateCollection 26233
XMLDoc.IXMLNodeAccess.DOMElement 26234
XMLDoc.IXMLNodeAccess.FindHostedNode 26234
XMLDoc.IXMLNodeAccess.GetChildNodeClasses 26234
XMLDoc.IXMLNodeAccess.GetHostNode 26234
XMLDoc.IXMLNodeAccess.GetNodeObject 26235
XMLDoc.IXMLNodeAccess.HasChildNode 26235
XMLDoc.IXMLNodeAccess.HostNode 26237
XMLDoc.IXMLNodeAccess.InternalAddChild 26235
XMLDoc.IXMLNodeAccess.NestingLevel 26236
XMLDoc.IXMLNodeAccess.RegisterChildNode 26236
XMLDoc.IXMLNodeAccess.RegisterChildNodes 26236
XMLDoc.IXMLNodeAccess.SetCollection 26236
XMLDoc.IXMLNodeAccess.SetParentNode 26237
XMLDoc.LoadXMLData 26240
XMLDoc.LoadXMLData function 26240
XMLDoc.LoadXMLDocument 26241
XMLDoc.LoadXMLDocument function 26241
XMLDoc.NewXMLDocument 26241
XMLDoc.NewXMLDocument function 26241
XMLDoc.TNodeChange 26243
XMLDoc.TNodeChange enumeration 26243
XMLDoc.TNodeChangeEvent 26247
XMLDoc.TNodeChangeEvent type 26247
XMLDoc.TNodeClassArray 26247
XMLDoc.TNodeClassArray type 26247
XMLDoc.TNodeClassInfo 26244
XMLDoc.TNodeClassInfo record 26244
XMLDoc.TNodeListNotification 26248
XMLDoc.TNodeListNotification type 26248
XMLDoc.TNodeListOperation 26244
XMLDoc.TNodeListOperation enumeration 26244
XMLDoc.TXMLDocument 26097
XMLDoc.TXMLDocument._AddRef 26106
XMLDoc.TXMLDocument._Release 26106
XMLDoc.TXMLDocument.Active 26133
XMLDoc.TXMLDocument.AddChild 26107
XMLDoc.TXMLDocument.AfterClose 26144
XMLDoc.TXMLDocument.AfterConstruction 26107
XMLDoc.TXMLDocument.AfterNodeChange 26145
XMLDoc.TXMLDocument.AfterOpen 26145
XMLDoc.TXMLDocument.AssignParseOptions 26108
XMLDoc.TXMLDocument.AsyncLoadState 26134
XMLDoc.TXMLDocument.BeforeClose 26146
XMLDoc.TXMLDocument.BeforeNodeChange 26146
XMLDoc.TXMLDocument.BeforeOpen 26147
XMLDoc.TXMLDocument.CheckActive 26108
XMLDoc.TXMLDocument.CheckAutoSave 26108
XMLDoc.TXMLDocument.CheckDOM 26109
XMLDoc.TXMLDocument.ChildNodes 26135

XMLDoc.TXMLDocument.Create 26103
XMLDoc.TXMLDocument.CreateElement 26109
XMLDoc.TXMLDocument.CreateNode 26109
XMLDoc.TXMLDocument.DefineProperties 26111
XMLDoc.TXMLDocument.Destroy 26104
XMLDoc.TXMLDocument.DoAfterClose 26111
XMLDoc.TXMLDocument.DoAfterOpen 26111
XMLDoc.TXMLDocument.DoBeforeClose 26112
XMLDoc.TXMLDocument.DoBeforeOpen 26112
XMLDoc.TXMLDocument.DocBindingInfo 26135
XMLDoc.TXMLDocument.DocSource 26135
XMLDoc.TXMLDocument.DocumentElement 26135
XMLDoc.TXMLDocument.DOMDocument 26136
XMLDoc.TXMLDocument.DOMImplementation 26137
XMLDoc.TXMLDocument.DOMParseOptions 26137
XMLDoc.TXMLDocument.DOMPersist 26137
XMLDoc.TXMLDocument.DOMVendor 26137
XMLDoc.TXMLDocument.DoNodeChange 26112
XMLDoc.TXMLDocument.Encoding 26138
XMLDoc.TXMLDocument.FileName 26138
XMLDoc.TXMLDocument.GeneratePrefix 26112
XMLDoc.TXMLDocument.GetActive 26113
XMLDoc.TXMLDocument.GetAsyncLoadState 26113
XMLDoc.TXMLDocument.GetChildNodeClass 26113
XMLDoc.TXMLDocument.GetChildNodes 26113
XMLDoc.TXMLDocument.GetDocBinding 26114
XMLDoc.TXMLDocument.GetDocumentElement 26114
XMLDoc.TXMLDocument.GetDocumentNode 26115
XMLDoc.TXMLDocument.GetDocumentObject 26115
XMLDoc.TXMLDocument.GetDOMDocument 26115
XMLDoc.TXMLDocument.GetDOMPersist 26115
XMLDoc.TXMLDocument.GetEncoding 26116
XMLDoc.TXMLDocument.GetFileName 26116
XMLDoc.TXMLDocument.GetModified 26116
XMLDoc.TXMLDocument.GetNodeIndentStr 26117
XMLDoc.TXMLDocument.GetOptions 26117
XMLDoc.TXMLDocument.GetParseOptions 26117
XMLDoc.TXMLDocument.GetPrologNode 26117
XMLDoc.TXMLDocument.GetPrologValue 26118
XMLDoc.TXMLDocument.GetSchemaRef 26118
XMLDoc.TXMLDocument.GetStandAlone 26118
XMLDoc.TXMLDocument.GetVersion 26118
XMLDoc.TXMLDocument.GetXML 26119
XMLDoc.TXMLDocument.InternalSetPrologValue 26119
XMLDoc.TXMLDocument.IsEmptyDoc 26119
XMLDoc.TXMLDocument.LoadData 26120
XMLDoc.TXMLDocument.Loaded 26120
XMLDoc.TXMLDocument.LoadFromFile 26121
XMLDoc.TXMLDocument.LoadFromStream 26121
XMLDoc.TXMLDocument.LoadFromXML 26122
XMLDoc.TXMLDocument.Modified 26139
XMLDoc.TXMLDocument.NewInstance 26122
XMLDoc.TXMLDocument.Node 26140
XMLDoc.TXMLDocument.NodeIndentStr 26140
XMLDoc.TXMLDocument.NSPrefixBase 26141
XMLDoc.TXMLDocument.OnAsyncLoad 26147
XMLDoc.TXMLDocument.Options 26141
XMLDoc.TXMLDocument.ParseOptions 26142
XMLDoc.TXMLDocument.PrefixID 26142
XMLDoc.TXMLDocument.ReadDOMVendor 26123
XMLDoc.TXMLDocument.Refresh 26123
XMLDoc.TXMLDocument.RegisterDocBinding 26124
XMLDoc.TXMLDocument.ReleaseDoc 26124
XMLDoc.TXMLDocument.Resync 26125
XMLDoc.TXMLDocument.SaveToFile 26125
XMLDoc.TXMLDocument.SaveToStream 26125
XMLDoc.TXMLDocument.SaveToUTF8String 26126
XMLDoc.TXMLDocument.SaveToXML 26126
XMLDoc.TXMLDocument.SaveToXMLStrings 26127
XMLDoc.TXMLDocument.SchemaRef 26142
XMLDoc.TXMLDocument.SetActive 26127
XMLDoc.TXMLDocument.SetDocumentElement 26127
XMLDoc.TXMLDocument.SetDOMDocument 26128
XMLDoc.TXMLDocument.SetEncoding 26128
XMLDoc.TXMLDocument.SetFileName 26128
XMLDoc.TXMLDocument.SetModified 26129
XMLDoc.TXMLDocument.SetNodeIndentStr 26129
XMLDoc.TXMLDocument.SetOnAsyncLoad 26129

XMLDoc.TXMLDocument.SetOptions 26130
XMLDoc.TXMLDocument.SetParseOptions 26130
XMLDoc.TXMLDocument.SetPrologValue 26130
XMLDoc.TXMLDocument.SetStandAlone 26130
XMLDoc.TXMLDocument.SetVersion 26131
XMLDoc.TXMLDocument.SetXML 26131
XMLDoc.TXMLDocument.SetXMLStrings 26131
XMLDoc.TXMLDocument.StandAlone 26143
XMLDoc.TXMLDocument.Version 26143
XMLDoc.TXMLDocument.WriteDOMVendor 26132
XMLDoc.TXMLDocument.XML 26144
XMLDoc.TXMLDocument.XMLStringsChanging 26132
XMLDoc.TXMLDocumentSource 26245
XMLDoc.TXMLDocumentSource enumeration 26245
XMLDoc.TXMLNode 26147
XMLDoc.TXMLNode._AddRef 26155
XMLDoc.TXMLNode._Release 26155
XMLDoc.TXMLNode.AddChild 26156
XMLDoc.TXMLNode.AddHostedNode 26157
XMLDoc.TXMLNode.AttributeListNotify 26158
XMLDoc.TXMLNode.AttributeNodes 26189
XMLDoc.TXMLNode.CheckNotHosted 26158
XMLDoc.TXMLNode.CheckReadOnly 26159
XMLDoc.TXMLNode.CheckTextNode 26159
XMLDoc.TXMLNode.ChildListNotify 26159
XMLDoc.TXMLNode.ChildNodeClasses 26190
XMLDoc.TXMLNode.ChildNodes 26190
XMLDoc.TXMLNode.ClearDocumentRef 26160
XMLDoc.TXMLNode.CloneNode 26160
XMLDoc.TXMLNode.Collection 26191
XMLDoc.TXMLNode.Create 26152
XMLDoc.TXMLNode.CreateAttributeList 26161
XMLDoc.TXMLNode.CreateAttributeNode 26161
XMLDoc.TXMLNode.CreateChildList 26162
XMLDoc.TXMLNode.CreateChildNode 26162
XMLDoc.TXMLNode.CreateCollection 26163
XMLDoc.TXMLNode.CreateHosted 26152
XMLDoc.TXMLNode.DeclareNamespace 26164
XMLDoc.TXMLNode.Destroy 26153
XMLDoc.TXMLNode.DOMElement 26164
XMLDoc.TXMLNode.DOMNode 26192
XMLDoc.TXMLNode.DoNodeChange 26165
XMLDoc.TXMLNode.FindHostedNode 26165
XMLDoc.TXMLNode.FindNamespaceDecl 26165
XMLDoc.TXMLNode.FindNamespaceURI 26166
XMLDoc.TXMLNode.GetAttribute 26166
XMLDoc.TXMLNode.GetAttributeNodes 26167
XMLDoc.TXMLNode.GetAttributeNS 26167
XMLDoc.TXMLNode.GetChildNodeClasses 26168
XMLDoc.TXMLNode.GetChildNodes 26168
XMLDoc.TXMLNode.GetChildValue 26169
XMLDoc.TXMLNode.GetCollection 26169
XMLDoc.TXMLNode.GetDOMNode 26170
XMLDoc.TXMLNode.GetHasChildNodes 26170
XMLDoc.TXMLNode.GetHostedNodes 26171
XMLDoc.TXMLNode.GetHostNode 26171
XMLDoc.TXMLNode.GetIsTextElement 26171
XMLDoc.TXMLNode.GetLocalName 26172
XMLDoc.TXMLNode.GetNamespaceURI 26172
XMLDoc.TXMLNode.GetNodeName 26173
XMLDoc.TXMLNode.GetNodeObject 26174
XMLDoc.TXMLNode.GetNodeType 26174
XMLDoc.TXMLNode.GetNodeValue 26174
XMLDoc.TXMLNode.GetOwnerDocument 26175
XMLDoc.TXMLNode.GetParentNode 26175
XMLDoc.TXMLNode.GetPrefix 26176
XMLDoc.TXMLNode.GetPrefixedName 26176
XMLDoc.TXMLNode.GetReadOnly 26177
XMLDoc.TXMLNode.GetText 26177
XMLDoc.TXMLNode.GetXML 26178
XMLDoc.TXMLNode.HasAttribute 26178
XMLDoc.TXMLNode.HasChildNode 26179
XMLDoc.TXMLNode.HostedNodes 26192
XMLDoc.TXMLNode.HostNode 26193
XMLDoc.TXMLNode.InternalAddChild 26179
XMLDoc.TXMLNode.NestingLevel 26180
XMLDoc.TXMLNode.NextSibling 26180
XMLDoc.TXMLNode.Normalize 26181

XMLDoc.TXMLNode.OnHostAttrNotify 26194
XMLDoc.TXMLNode.OnHostChildNotify 26195
XMLDoc.TXMLNode.OwnerDocument 26193
XMLDoc.TXMLNode.ParentNode 26194
XMLDoc.TXMLNode.PreviousSibling 26181
XMLDoc.TXMLNode.RegisterChildNode 26182
XMLDoc.TXMLNode.RegisterChildNodes 26182
XMLDoc.TXMLNode.RemoveHostedNode 26183
XMLDoc.TXMLNode.Resync 26183
XMLDoc.TXMLNode.SetAttribute 26183
XMLDoc.TXMLNode.SetAttributeNodes 26184
XMLDoc.TXMLNode.SetAttributeNS 26184
XMLDoc.TXMLNode.SetChildNodes 26185
XMLDoc.TXMLNode.SetChildValue 26185
XMLDoc.TXMLNode.SetCollection 26186
XMLDoc.TXMLNode.SetNodeValue 26186
XMLDoc.TXMLNode.SetParentNode 26187
XMLDoc.TXMLNode.SetReadOnly 26187
XMLDoc.TXMLNode.SetText 26188
XMLDoc.TXMLNode.TransformNode 26188
XMLDoc.TXMLNodeArray 26249
XMLDoc.TXMLNodeArray type 26249
XMLDoc.TXMLNodeClass 26249
XMLDoc.TXMLNodeClass type 26249
XMLDoc.TXMLNodeCollection 26196
XMLDoc.TXMLNodeCollection.AddItem 26202
XMLDoc.TXMLNodeCollection.AfterConstruction 26202
XMLDoc.TXMLNodeCollection.ChildListNotify 26203
XMLDoc.TXMLNodeCollection.Clear 26204
XMLDoc.TXMLNodeCollection.Count 26209
XMLDoc.TXMLNodeCollection.CreateItemList 26204
XMLDoc.TXMLNodeCollection.Delete 26205
XMLDoc.TXMLNodeCollection.GetCount 26205
XMLDoc.TXMLNodeCollection.GetList 26205
XMLDoc.TXMLNodeCollection.GetNode 26206
XMLDoc.TXMLNodeCollection.InsertInCollection 26206
XMLDoc.TXMLNodeCollection.IsCollectionItem 26207
XMLDoc.TXMLNodeCollection.ItemInterface 26209
XMLDoc.TXMLNodeCollection.ItemNS 26210
XMLDoc.TXMLNodeCollection.ItemTag 26210
XMLDoc.TXMLNodeCollection.List 26210
XMLDoc.TXMLNodeCollection.Remove 26207
XMLDoc.TXMLNodeCollection.SetChildNodes 26208
XMLDoc.TXMLNodeCollection.UpdateCollectionList 26208
XMLDoc.TXMLNodeCollectionClass 26250
XMLDoc.TXMLNodeCollectionClass type 26250
XMLDoc.TXMLNodeList 26211
XMLDoc.TXMLNodeList.Add 26215
XMLDoc.TXMLNodeList.BeginUpdate 26215
XMLDoc.TXMLNodeList.Clear 26216
XMLDoc.TXMLNodeList.Count 26225
XMLDoc.TXMLNodeList.Create 26213
XMLDoc.TXMLNodeList.DefaultNamespaceURI 26225
XMLDoc.TXMLNodeList.Delete 26217
XMLDoc.TXMLNodeList.Destroy 26214
XMLDoc.TXMLNodeList.DoNotify 26217
XMLDoc.TXMLNodeList.EndUpdate 26218
XMLDoc.TXMLNodeList.FindNode 26218
XMLDoc.TXMLNodeList.FindSibling 26219
XMLDoc.TXMLNodeList.First 26220
XMLDoc.TXMLNodeList.Get 26220
XMLDoc.TXMLNodeList.GetCount 26220
XMLDoc.TXMLNodeList.GetNode 26221
XMLDoc.TXMLNodeList.GetUpdateCount 26221
XMLDoc.TXMLNodeList.IndexOf 26222
XMLDoc.TXMLNodeList.Insert 26223
XMLDoc.TXMLNodeList.InternalInsert 26223
XMLDoc.TXMLNodeList.Last 26223
XMLDoc.TXMLNodeList.List 26226
XMLDoc.TXMLNodeList.NotificationProc 26227
XMLDoc.TXMLNodeList.Owner 26226
XMLDoc.TXMLNodeList.Remove 26224
XMLDoc.TXMLNodeList.ReplaceNode 26224
XMLDoc.TXMLNodeList.UpdateCount 26226
XMLDoc.TXMLPrologItem 26246
XMLDoc.TXMLPrologItem enumeration 26246
XMLDoc.UseXSDBooleanStrings 26251
XMLDoc.UseXSDBooleanStrings variable 26251

XMLDoc.XMLDocError 26242
XMLDoc.XMLDocError function 26242
XMLDoc.XMLStringToWideString 26242
XMLDoc.XMLStringToWideString function 26242
xmldom 26253
xmldom namespace 26253
Classes 26256
Constants 26366
Functions 26354
Interfaces 26270
Types 26362
Variables 26365
xmldom.ATTRIBUTE_NODE 26366
xmldom.ATTRIBUTE_NODE constant 26366
xmldom.CDATA_SECTION_NODE 26367
xmldom.CDATA_SECTION_NODE constant 26367
xmldom.COMMENT_NODE 26367
xmldom.COMMENT_NODE constant 26367
xmldom.DefaultDOMVendor 26365
xmldom.DefaultDOMVendor variable 26365
xmldom.DOCUMENT_FRAGMENT_NODE 26367
xmldom.DOCUMENT_FRAGMENT_NODE constant 26367
xmldom.DOCUMENT_NODE 26368
xmldom.DOCUMENT_NODE constant 26368
xmldom.DOCUMENT_TYPE_NODE 26368
xmldom.DOCUMENT_TYPE_NODE constant 26368
xmldom.DOMException 26256
xmldom.DOMException.code 26257
xmldom.DOMNodeType 26362
xmldom.DOMNodeType type 26362
xmldom.DOMString 26362
xmldom.DOMString type 26362
xmldom.DOMSTRING_SIZE_ERR 26368
xmldom.DOMSTRING_SIZE_ERR constant 26368
xmldom.DOMTimeStamp 26363
xmldom.DOMTimeStamp type 26363
xmldom.DOMVendorNotSupported 26354
xmldom.DOMVendorNotSupported function 26354
xmldom.DOMVendors 26365
xmldom.DOMVendors variable 26365
xmldom.DOMWrapperVersion 26369
xmldom.DOMWrapperVersion constant 26369
xmldom.EDOMParseError 26258
xmldom.EDOMParseError.Create 26259
xmldom.EDOMParseError.ErrorCode 26260
xmldom.EDOMParseError.FilePos 26260
xmldom.EDOMParseError.Line 26261
xmldom.EDOMParseError.LinePos 26261
xmldom.EDOMParseError.ParseError 26261
xmldom.EDOMParseError.Reason 26262
xmldom.EDOMParseError.SrcText 26262
xmldom.EDOMParseError.URL 26262
xmldom.ELEMENT_NODE 26369
xmldom.ELEMENT_NODE constant 26369
xmldom.ENTITY_NODE 26369
xmldom.ENTITY_NODE constant 26369
xmldom.ENTITY_REFERENCE_NODE 26370
xmldom.ENTITY_REFERENCE_NODE constant 26370
xmldom.ExtractLocalName 26354
xmldom.ExtractLocalName function 26354
xmldom.ExtractPrefix 26355
xmldom.ExtractPrefix function 26355
xmldom.GetDOM 26356
xmldom.GetDOM function 26356
xmldom.GetDOMNodeEx 26356
xmldom.GetDOMNodeEx function 26356
xmldom.GetDOMVendor 26357
xmldom.GetDOMVendor function 26357
xmldom.HIERARCHY_REQUEST_ERR 26370
xmldom.HIERARCHY_REQUEST_ERR constant 26370
xmldom.IDOMAttr 26270
xmldom.IDOMAttr.get_name 26272
xmldom.IDOMAttr.get_ownerElement 26272
xmldom.IDOMAttr.get_specified 26272
xmldom.IDOMAttr.get_value 26273
xmldom.IDOMAttr.name 26273
xmldom.IDOMAttr.ownerElement 26273
xmldom.IDOMAttr.set_value 26273

xmldom.IDOMAttr.specified 26274
xmldom.IDOMAttr.value 26274
xmldom.IDOMCDATASection 26274
xmldom.IDOMCharacterData 26275
xmldom.IDOMCharacterData.appendData 26276
xmldom.IDOMCharacterData.data 26278
xmldom.IDOMCharacterData.deleteData 26277
xmldom.IDOMCharacterData.get_data 26277
xmldom.IDOMCharacterData.get_length 26277
xmldom.IDOMCharacterData.insertData 26277
xmldom.IDOMCharacterData.length 26278
xmldom.IDOMCharacterData.replaceData 26277
xmldom.IDOMCharacterData.set_data 26278
xmldom.IDOMCharacterData.substringData 26278
xmldom.IDOMComment 26279
xmldom.IDOMDocument 26279
xmldom.IDOMDocument.createAttribute 26281
xmldom.IDOMDocument.createAttributeNS 26282
xmldom.IDOMDocument.createCDATASection 26282
xmldom.IDOMDocument.createComment 26282
xmldom.IDOMDocument.createDocumentFragment 26282
xmldom.IDOMDocument.createElement 26282
xmldom.IDOMDocument.createElementNS 26283
xmldom.IDOMDocument.createEntityReference 26283
xmldom.IDOMDocument.createProcessingInstruction 26283
xmldom.IDOMDocument.createTextNode 26283
xmldom.IDOMDocument.doctype 26286
xmldom.IDOMDocument.documentElement 26286
xmldom.IDOMDocument.domImplementation 26286
xmldom.IDOMDocument.get_doctype 26284
xmldom.IDOMDocument.get_documentElement 26284
xmldom.IDOMDocument.get_domImplementation 26284
xmldom.IDOMDocument.getElementById 26284
xmldom.IDOMDocument.getElementsByTagName 26285
xmldom.IDOMDocument.getElementsByTagNameNS 26285
xmldom.IDOMDocument.importNode 26285
xmldom.IDOMDocument.set_documentElement 26285
xmldom.IDOMDocumentFragment 26286
xmldom.IDOMDocumentType 26287
xmldom.IDOMDocumentType.entities 26290
xmldom.IDOMDocumentType.get_entities 26289
xmldom.IDOMDocumentType.get_internalSubset 26289
xmldom.IDOMDocumentType.get_name 26289
xmldom.IDOMDocumentType.get_notations 26289
xmldom.IDOMDocumentType.get_publicId 26289
xmldom.IDOMDocumentType.get_systemId 26290
xmldom.IDOMDocumentType.internalSubset 26290
xmldom.IDOMDocumentType.name 26291
xmldom.IDOMDocumentType.notations 26291
xmldom.IDOMDocumentType.publicId 26291
xmldom.IDOMDocumentType.systemId 26291
xmldom.IDOMElement 26291
xmldom.IDOMElement.get_tagName 26294
xmldom.IDOMElement.getAttribute 26294
xmldom.IDOMElement.getAttributeNode 26294
xmldom.IDOMElement.getAttributeNodeNS 26294
xmldom.IDOMElement.getAttributeNS 26294
xmldom.IDOMElement.getElementsByTagName 26295
xmldom.IDOMElement.getElementsByTagNameNS 26295
xmldom.IDOMElement.hasAttribute 26295
xmldom.IDOMElement.hasAttributeNS 26295
xmldom.IDOMElement.removeAttribute 26296
xmldom.IDOMElement.removeAttributeNode 26296
xmldom.IDOMElement.removeAttributeNS 26296
xmldom.IDOMElement.setAttribute 26296
xmldom.IDOMElement.setAttributeNode 26296
xmldom.IDOMElement.setAttributeNodeNS 26297
xmldom.IDOMElement.setAttributeNS 26297
xmldom.IDOMElement.tagName 26297
xmldom.IDOMEntity 26298
xmldom.IDOMEntity.get_notationName 26299
xmldom.IDOMEntity.get_publicId 26299
xmldom.IDOMEntity.get_systemId 26300
xmldom.IDOMEntity.notationName 26300
xmldom.IDOMEntity.publicId 26300
xmldom.IDOMEntity.systemId 26300
xmldom.IDOMEntityReference 26301
xmldom.IDOMImplementation 26301

eeee

xmldom.IDOMImplementation.createDocument 26302
xmldom.IDOMImplementation.createDocumentType 26302
xmldom.IDOMImplementation.hasFeature 26302
xmldom.IDOMNamedNodeMap 26303
xmldom.IDOMNamedNodeMap.get_item 26304
xmldom.IDOMNamedNodeMap.get_length 26304
xmldom.IDOMNamedNodeMap.getNamedItem 26304
xmldom.IDOMNamedNodeMap.getNamedItemNS 26304
xmldom.IDOMNamedNodeMap.item 26306
xmldom.IDOMNamedNodeMap.length 26306
xmldom.IDOMNamedNodeMap.removeNamedItem 26305
xmldom.IDOMNamedNodeMap.removeNamedItemNS 26305
xmldom.IDOMNamedNodeMap.setNamedItem 26305
xmldom.IDOMNamedNodeMap.setNamedItemNS 26305
xmldom.IDOMNode 26306
xmldom.IDOMNode.appendChild 26308
xmldom.IDOMNode.attributes 26317
xmldom.IDOMNode.childNodes 26317
xmldom.IDOMNode.cloneNode 26309
xmldom.IDOMNode.firstChild 26317
xmldom.IDOMNode.get_attributes 26309
xmldom.IDOMNode.get_childNodes 26309
xmldom.IDOMNode.get_firstChild 26309
xmldom.IDOMNode.get_lastChild 26310
xmldom.IDOMNode.get_localName 26310
xmldom.IDOMNode.get_namespaceURI 26310
xmldom.IDOMNode.get_nextSibling 26311
xmldom.IDOMNode.get_nodeName 26311
xmldom.IDOMNode.get_nodeType 26311
xmldom.IDOMNode.get_nodeValue 26312
xmldom.IDOMNode.get_ownerDocument 26312
xmldom.IDOMNode.get_parentNode 26312
xmldom.IDOMNode.get_prefix 26312
xmldom.IDOMNode.get_previousSibling 26313
xmldom.IDOMNode.hasChildNodes 26313
xmldom.IDOMNode.insertBefore 26313
xmldom.IDOMNode.lastChild 26317
xmldom.IDOMNode.localName 26318
xmldom.IDOMNode.namespaceURI 26318
xmldom.IDOMNode.nextSibling 26318
xmldom.IDOMNode.nodeName 26318
xmldom.IDOMNode.nodeType 26318
xmldom.IDOMNode.nodeValue 26319
xmldom.IDOMNode.normalize 26314
xmldom.IDOMNode.ownerDocument 26319
xmldom.IDOMNode.parentNode 26319
xmldom.IDOMNode.prefix 26319
xmldom.IDOMNode.previousSibling 26320
xmldom.IDOMNode.removeChild 26315
xmldom.IDOMNode.replaceChild 26315
xmldom.IDOMNode.set_nodeValue 26316
xmldom.IDOMNode.supports 26316
xmldom.IDOMNodeEx 26320
xmldom.IDOMNodeEx.get_text 26321
xmldom.IDOMNodeEx.get_xml 26322
xmldom.IDOMNodeEx.set_text 26322
xmldom.IDOMNodeEx.text 26323
xmldom.IDOMNodeEx.transformNode 26322, 26323
xmldom.IDOMNodeEx.xml 26323
xmldom.IDOMNodeList 26323
xmldom.IDOMNodeList.get_item 26324
xmldom.IDOMNodeList.get_length 26324
xmldom.IDOMNodeList.item 26325
xmldom.IDOMNodeList.length 26325
xmldom.IDOMNodeSelect 26325
xmldom.IDOMNodeSelect.selectNode 26326
xmldom.IDOMNodeSelect.selectNodes 26326
xmldom.IDOMNotation 26327
xmldom.IDOMNotation.get_publicId 26328
xmldom.IDOMNotation.get_systemId 26328
xmldom.IDOMNotation.publicId 26329
xmldom.IDOMNotation.systemId 26329
xmldom.IDOMParseError 26329
xmldom.IDOMParseError.errorCode 26332
xmldom.IDOMParseError.filePos 26332
xmldom.IDOMParseError.get_errorCode 26330
xmldom.IDOMParseError.get_filepos 26331
xmldom.IDOMParseError.get_line 26331

xmldom.IDOMParseError.get_linepos 26331
xmldom.IDOMParseError.get_reason 26331
xmldom.IDOMParseError.get_srcText 26331
xmldom.IDOMParseError.get_url 26332
xmldom.IDOMParseError.line 26333
xmldom.IDOMParseError.linePos 26333
xmldom.IDOMParseError.reason 26333
xmldom.IDOMParseError.srcText 26333
xmldom.IDOMParseError.url 26333
xmldom.IDOMParseOptions 26334
xmldom.IDOMParseOptions.async 26337
xmldom.IDOMParseOptions.get_async 26335
xmldom.IDOMParseOptions.get_preserveWhiteSpace 26335
xmldom.IDOMParseOptions.get_resolveExternals 26335
xmldom.IDOMParseOptions.get_validate 26335
xmldom.IDOMParseOptions.preserveWhiteSpace 26337
xmldom.IDOMParseOptions.resolveExternals 26337
xmldom.IDOMParseOptions.set_async 26336
xmldom.IDOMParseOptions.set_preserveWhiteSpace 26336
xmldom.IDOMParseOptions.set_resolveExternals 26336
xmldom.IDOMParseOptions.set_validate 26336
xmldom.IDOMParseOptions.validate 26337
xmldom.IDOMPersist 26338
xmldom.IDOMPersist.asyncLoadState 26339
xmldom.IDOMPersist.get_xml 26339
xmldom.IDOMPersist.load 26339
xmldom.IDOMPersist.loadFromStream 26339, 26340
xmldom.IDOMPersist.loadxml 26340
xmldom.IDOMPersist.save 26340
xmldom.IDOMPersist.saveToStream 26341
xmldom.IDOMPersist.set_OnAsyncLoad 26341
xmldom.IDOMPersist.xml 26341
xmldom.IDOMProcessingInstruction 26342
xmldom.IDOMProcessingInstruction.data 26344
xmldom.IDOMProcessingInstruction.get_data 26343
xmldom.IDOMProcessingInstruction.get_target 26343
xmldom.IDOMProcessingInstruction.set_data 26344
xmldom.IDOMProcessingInstruction.target 26344
xmldom.IDOMText 26345
xmldom.IDOMText.splitText 26346
xmldom.IDOMXMLProlog 26347
xmldom.IDOMXMLProlog.Encoding 26349
xmldom.IDOMXMLProlog.get_Encoding 26348
xmldom.IDOMXMLProlog.get_Standalone 26348
xmldom.IDOMXMLProlog.get_Version 26348
xmldom.IDOMXMLProlog.set_Encoding 26348
xmldom.IDOMXMLProlog.set_Standalone 26348
xmldom.IDOMXMLProlog.set_Version 26349
xmldom.IDOMXMLProlog.Standalone 26349
xmldom.IDOMXMLProlog.Version 26349
xmldom.IDOMXSLProcessor 26350
xmldom.IDOMXSLProcessor.Get_input 26351
xmldom.IDOMXSLProcessor.Get_output 26351
xmldom.IDOMXSLProcessor.Get_stylesheet 26351
xmldom.IDOMXSLProcessor.input 26353
xmldom.IDOMXSLProcessor.output 26353
xmldom.IDOMXSLProcessor.reset 26352
xmldom.IDOMXSLProcessor.Set_input 26352
xmldom.IDOMXSLProcessor.Set_output 26352
xmldom.IDOMXSLProcessor.setParameter 26352
xmldom.IDOMXSLProcessor.stylesheet 26353
xmldom.IDOMXSLProcessor.transform 26352
xmldom.INDEX_SIZE_ERR 26370
xmldom.INDEX_SIZE_ERR constant 26370
xmldom.INUSE_ATTRIBUTE_ERR 26371
xmldom.INUSE_ATTRIBUTE_ERR constant 26371
xmldom.INVALID_ACCESS_ERR 26371
xmldom.INVALID_ACCESS_ERR constant 26371
xmldom.INVALID_CHARACTER_ERR 26371
xmldom.INVALID_CHARACTER_ERR constant 26371
xmldom.INVALID_MODIFICATION_ERR 26372
xmldom.INVALID_MODIFICATION_ERR constant 26372
xmldom.INVALID_STATE_ERR 26372
xmldom.INVALID_STATE_ERR constant 26372
xmldom.IsPrefixed 26357
xmldom.IsPrefixed function 26357
xmldom.MakeNodeName 26358
xmldom.MakeNodeName function 26358

gggggg

xmldom.NAMESPACE_ERR 26372
xmldom.NAMESPACE_ERR constant 26372
xmldom.NO_DATA_ALLOWED_ERR 26373
xmldom.NO_DATA_ALLOWED_ERR constant 26373
xmldom.NO_MODIFICATION_ALLOWED_ERR 26373
xmldom.NO_MODIFICATION_ALLOWED_ERR constant 26373
xmldom.NodeMatches 26358
xmldom.NodeMatches function 26358
xmldom.NOT_FOUND_ERR 26373
xmldom.NOT_FOUND_ERR constant 26373
xmldom.NOT_SUPPORTED_ERR 26374
xmldom.NOT_SUPPORTED_ERR constant 26374
xmldom.NOTATION_NODE 26374
xmldom.NOTATION_NODE constant 26374
xmldom.NSDelim 26374
xmldom.NSDelim constant 26374
xmldom.PROCESSING_INSTRUCTION_NODE 26375
xmldom.PROCESSING_INSTRUCTION_NODE constant 26375
xmldom.RegisterDOMVendor 26359
xmldom.RegisterDOMVendor function 26359
xmldom.SameNamespace 26359
xmldom.SameNamespace function 26359
xmldom.SEncoding 26375
xmldom.SEncoding constant 26375
xmldom.SHttp 26375
xmldom.SHttp constant 26375
xmldom.SStandalone 26376
xmldom.SStandalone constant 26376
xmldom.SVersion 26376
xmldom.SVersion constant 26376
xmldom.SXML 26376
xmldom.SXML constant 26376
xmldom.SXMLNamespaceURI 26377
xmldom.SXMLNamespaceURI constant 26377
xmldom.SXMLNS 26377
xmldom.SXMLNS constant 26377
xmldom.SXMLPrefixNamespaceURI 26377
xmldom.SXMLPrefixNamespaceURI constant 26377
xmldom.SYNTAX_ERR 26378
xmldom.SYNTAX_ERR constant 26378
xmldom.TAsyncEventHandler 26363
xmldom.TAsyncEventHandler type 26363
xmldom.TDOMVendor 26263
xmldom.TDOMVendor.Description 26264
xmldom.TDOMVendor.DOMImplementation 26264
xmldom.TDOMVendorArray 26364
xmldom.TDOMVendorArray type 26364
xmldom.TDOMVendorList 26265
xmldom.TDOMVendorList.Add 26266
xmldom.TDOMVendorList.Count 26266
xmldom.TDOMVendorList.Find 26267
xmldom.TDOMVendorList.GetVendors 26267
xmldom.TDOMVendorList.Remove 26268
xmldom.TDOMVendorList.Vendors 26268
xmldom.TEXT_NODE 26378
xmldom.TEXT_NODE constant 26378
xmldom.UnRegisterDOMVendor 26360
xmldom.UnRegisterDOMVendor function 26360
xmldom.WRONG_DOCUMENT_ERR 26378
xmldom.WRONG_DOCUMENT_ERR constant 26378
XMLIntf 26379
XMLIntf namespace 26379
Classes 26381
Constants 26464
Interfaces 26382
Structs, Records, Enums 26458
Types 26463
XMLIntf.EXMLDocError 26381
XMLIntf.IXMLDocument 26382
XMLIntf.IXMLDocument.Active 26403
XMLIntf.IXMLDocument.AddChild 26385
XMLIntf.IXMLDocument.AsyncLoadState 26404
XMLIntf.IXMLDocument.ChildNodes 26405
XMLIntf.IXMLDocument.CreateElement 26386
XMLIntf.IXMLDocument.CreateNode 26387
XMLIntf.IXMLDocument.DocumentElement 26405
XMLIntf.IXMLDocument.DOMDocument 26405
XMLIntf.IXMLDocument.Encoding 26406

hhhh

XMLIntf.IXMLDocument.FileName 26406
XMLIntf.IXMLDocument.GeneratePrefix 26388
XMLIntf.IXMLDocument.GetActive 26388
XMLIntf.IXMLDocument.GetAsyncLoadState 26389
XMLIntf.IXMLDocument.GetChildNodes 26389
XMLIntf.IXMLDocument.GetDocBinding 26389
XMLIntf.IXMLDocument.GetDocumentElement 26390
XMLIntf.IXMLDocument.GetDocumentNode 26390
XMLIntf.IXMLDocument.GetDOMDocument 26390
XMLIntf.IXMLDocument.GetEncoding 26391
XMLIntf.IXMLDocument.GetFileName 26391
XMLIntf.IXMLDocument.GetModified 26391
XMLIntf.IXMLDocument.GetNodeIndentStr 26392
XMLIntf.IXMLDocument.GetOptions 26392
XMLIntf.IXMLDocument.GetParseOptions 26392
XMLIntf.IXMLDocument.GetSchemaRef 26392
XMLIntf.IXMLDocument.GetStandAlone 26393
XMLIntf.IXMLDocument.GetVersion 26393
XMLIntf.IXMLDocument.GetXML 26393
XMLIntf.IXMLDocument.IsEmptyDoc 26394
XMLIntf.IXMLDocument.LoadFromFile 26394
XMLIntf.IXMLDocument.LoadFromStream 26395
XMLIntf.IXMLDocument.LoadFromXML 26395
XMLIntf.IXMLDocument.Modified 26407
XMLIntf.IXMLDocument.Node 26407
XMLIntf.IXMLDocument.NodeIndentStr 26408
XMLIntf.IXMLDocument.Options 26408
XMLIntf.IXMLDocument.ParseOptions 26409
XMLIntf.IXMLDocument.Refresh 26396
XMLIntf.IXMLDocument.RegisterDocBinding 26396
XMLIntf.IXMLDocument.Resync 26397
XMLIntf.IXMLDocument.SaveToFile 26397
XMLIntf.IXMLDocument.SaveToStream 26398
XMLIntf.IXMLDocument.SaveToXML 26398
XMLIntf.IXMLDocument.SchemaRef 26409
XMLIntf.IXMLDocument.SetActive 26399
XMLIntf.IXMLDocument.SetDocumentElement 26399
XMLIntf.IXMLDocument.SetDOMDocument 26399
XMLIntf.IXMLDocument.SetEncoding 26400
XMLIntf.IXMLDocument.SetFileName 26400
XMLIntf.IXMLDocument.SetNodeIndentStr 26400
XMLIntf.IXMLDocument.SetOnAsyncLoad 26401
XMLIntf.IXMLDocument.SetOptions 26401
XMLIntf.IXMLDocument.SetParseOptions 26402
XMLIntf.IXMLDocument.SetStandAlone 26402
XMLIntf.IXMLDocument.SetVersion 26402
XMLIntf.IXMLDocument.SetXML 26403
XMLIntf.IXMLDocument.StandAlone 26410
XMLIntf.IXMLDocument.Version 26410
XMLIntf.IXMLDocument.XML 26410
XMLIntf.IXMLNode 26411
XMLIntf.IXMLNode.AddChild 26414
XMLIntf.IXMLNode.AttributeNodes 26430
XMLIntf.IXMLNode.Attributes 26430
XMLIntf.IXMLNode.ChildNodes 26431
XMLIntf.IXMLNode.ChildValues 26432
XMLIntf.IXMLNode.CloneNode 26415
XMLIntf.IXMLNode.Collection 26432
XMLIntf.IXMLNode.DeclareNamespace 26416
XMLIntf.IXMLNode.DOMNode 26433
XMLIntf.IXMLNode.FindNamespaceDecl 26416
XMLIntf.IXMLNode.FindNamespaceURI 26416
XMLIntf.IXMLNode.GetAttribute 26417
XMLIntf.IXMLNode.GetAttributeNodes 26417
XMLIntf.IXMLNode.GetAttributeNS 26418
XMLIntf.IXMLNode.GetChildNodes 26418
XMLIntf.IXMLNode.GetChildValue 26418
XMLIntf.IXMLNode.GetCollection 26419
XMLIntf.IXMLNode.GetDOMNode 26419
XMLIntf.IXMLNode.GetHasChildNodes 26420
XMLIntf.IXMLNode.GetIsTextElement 26420
XMLIntf.IXMLNode.GetLocalName 26420
XMLIntf.IXMLNode.GetNamespaceURI 26421
XMLIntf.IXMLNode.GetNodeName 26421
XMLIntf.IXMLNode.GetNodeType 26421
XMLIntf.IXMLNode.GetNodeValue 26422
XMLIntf.IXMLNode.GetOwnerDocument 26422
XMLIntf.IXMLNode.GetParentNode 26422

XMLIntf.IXMLNode.GetPrefix 26422
XMLIntf.IXMLNode.GetReadOnly 26423
XMLIntf.IXMLNode.GetText 26423
XMLIntf.IXMLNode.GetXML 26423
XMLIntf.IXMLNode.HasAttribute 26424
XMLIntf.IXMLNode.HasChildNodes 26433
XMLIntf.IXMLNode.IsTextElement 26434
XMLIntf.IXMLNode.LocalName 26434
XMLIntf.IXMLNode.NamespaceURI 26435
XMLIntf.IXMLNode.NextSibling 26424
XMLIntf.IXMLNode.NodeName 26435
XMLIntf.IXMLNode.NodeType 26436
XMLIntf.IXMLNode.NodeValue 26436
XMLIntf.IXMLNode.Normalize 26425
XMLIntf.IXMLNode.OwnerDocument 26437
XMLIntf.IXMLNode.ParentNode 26437
XMLIntf.IXMLNode.Prefix 26438
XMLIntf.IXMLNode.PreviousSibling 26425
XMLIntf.IXMLNode.ReadOnly 26438
XMLIntf.IXMLNode.Resync 26426
XMLIntf.IXMLNode.SetAttribute 26426
XMLIntf.IXMLNode.SetAttributeNS 26426
XMLIntf.IXMLNode.SetChildValue 26427
XMLIntf.IXMLNode.SetNodeValue 26427
XMLIntf.IXMLNode.SetReadOnly 26428
XMLIntf.IXMLNode.SetText 26428
XMLIntf.IXMLNode.Text 26439
XMLIntf.IXMLNode.TransformNode 26429
XMLIntf.IXMLNode.XML 26439
XMLIntf.IXMLNodeCollection 26440
XMLIntf.IXMLNodeCollection.Clear 26442
XMLIntf.IXMLNodeCollection.Count 26444
XMLIntf.IXMLNodeCollection.Delete 26442
XMLIntf.IXMLNodeCollection.GetCount 26443
XMLIntf.IXMLNodeCollection.GetNode 26443
XMLIntf.IXMLNodeCollection.Nodes 26445
XMLIntf.IXMLNodeCollection.Remove 26443
XMLIntf.IXMLNodeList 26445
XMLIntf.IXMLNodeList.Add 26447
XMLIntf.IXMLNodeList.BeginUpdate 26447
XMLIntf.IXMLNodeList.Clear 26448
XMLIntf.IXMLNodeList.Count 26456
XMLIntf.IXMLNodeList.Delete 26449
XMLIntf.IXMLNodeList.EndUpdate 26449
XMLIntf.IXMLNodeList.FindNode 26450
XMLIntf.IXMLNodeList.FindSibling 26451
XMLIntf.IXMLNodeList.First 26451
XMLIntf.IXMLNodeList.Get 26452
XMLIntf.IXMLNodeList.GetCount 26452
XMLIntf.IXMLNodeList.GetNode 26452
XMLIntf.IXMLNodeList.GetUpdateCount 26453
XMLIntf.IXMLNodeList.IndexOf 26453
XMLIntf.IXMLNodeList.Insert 26454
XMLIntf.IXMLNodeList.Last 26454
XMLIntf.IXMLNodeList.Nodes 26456
XMLIntf.IXMLNodeList.Remove 26455
XMLIntf.IXMLNodeList.ReplaceNode 26455
XMLIntf.IXMLNodeList.UpdateCount 26457
XMLIntf.NodeTypeNames 26464
XMLIntf.NodeTypeNames constant 26464
XMLIntf.TNodeType 26458
XMLIntf.TNodeType enumeration 26458
XMLIntf.TParseOption 26460
XMLIntf.TParseOption enumeration 26460
XMLIntf.TParseOptions 26463
XMLIntf.TParseOptions type 26463
XMLIntf.TXMLDocOption 26460
XMLIntf.TXMLDocOption enumeration 26460
XMLIntf.TXMLDocOptions 26463
XMLIntf.TXMLDocOptions type 26463
XMLIntf.TXMLEncodingType 26461
XMLIntf.TXMLEncodingType enumeration 26461
Xmlxform 26465
Xmlxform namespace 26465
    Classes 26466
    Types 26509
Xmlxform.TranslateException 26466
Xmlxform.TRowEvent 26509

jjjj

Xmlxform.TRowEvent type 26509
Xmlxform.TTranslateEvent 26509
Xmlxform.TTranslateEvent type 26509
Xmlxform.TXMLTransform 26467
Xmlxform.TXMLTransform.AfterEachRow 26479
Xmlxform.TXMLTransform.AfterEachRowSet 26480
Xmlxform.TXMLTransform.BeforeEachRow 26480
Xmlxform.TXMLTransform.BeforeEachRowSet 26481
Xmlxform.TXMLTransform.Data 26473
Xmlxform.TXMLTransform.DoTransform 26471
Xmlxform.TXMLTransform.EmptyDestinationDocument 26474
Xmlxform.TXMLTransform.GetData 26471
Xmlxform.TXMLTransform.GetResultString 26471
Xmlxform.TXMLTransform.OnTranslate 26481
Xmlxform.TXMLTransform.ResultDocument 26475
Xmlxform.TXMLTransform.ResultString 26475
Xmlxform.TXMLTransform.SourceXml 26476
Xmlxform.TXMLTransform.SourceXmlDocument 26476
Xmlxform.TXMLTransform.SourceXmlFile 26477
Xmlxform.TXMLTransform.Transform 26472
Xmlxform.TXMLTransform.TransformationDocument 26478
Xmlxform.TXMLTransform.TransformationFile 26478
Xmlxform.TXMLTransform.TransformXML 26472
Xmlxform.TXMLTransform.Translate 26473
Xmlxform.TXMLTransformClient 26482
Xmlxform.TXMLTransformClient.ApplyUpdates 26486
Xmlxform.TXMLTransformClient.Create 26485
Xmlxform.TXMLTransformClient.Destroy 26485
Xmlxform.TXMLTransformClient.GetDataAsXml 26487
Xmlxform.TXMLTransformClient.ProviderName 26489
Xmlxform.TXMLTransformClient.RemoteServer 26490
Xmlxform.TXMLTransformClient.SetParams 26488
Xmlxform.TXMLTransformClient.SetupAppServer 26489
Xmlxform.TXMLTransformClient.TransformApplyUpdates 26490
Xmlxform.TXMLTransformClient.TransformGetData 26491
Xmlxform.TXMLTransformClient.TransformSetParams 26491
Xmlxform.TXMLTransformProvider 26492
Xmlxform.TXMLTransformProvider.AfterApplyUpdates 26502
Xmlxform.TXMLTransformProvider.AfterGetRecords 26503
Xmlxform.TXMLTransformProvider.AfterRowRequest 26504
Xmlxform.TXMLTransformProvider.BeforeApplyUpdates 26504
Xmlxform.TXMLTransformProvider.BeforeGetRecords 26505
Xmlxform.TXMLTransformProvider.BeforeRowRequest 26506
Xmlxform.TXMLTransformProvider.CacheData 26499
Xmlxform.TXMLTransformProvider.Create 26497
Xmlxform.TXMLTransformProvider.Destroy 26497
Xmlxform.TXMLTransformProvider.InternalApplyUpdates 26498
Xmlxform.TXMLTransformProvider.InternalGetRecords 26498
Xmlxform.TXMLTransformProvider.OnDataRequest 26507
Xmlxform.TXMLTransformProvider.TransformRead 26500
Xmlxform.TXMLTransformProvider.TransformWrite 26500
Xmlxform.TXMLTransformProvider.XMLDataFile 26501
xoAttributeOnLastMember enumeration member 15630
xoHolderClass enumeration member 15630
xoInlineArrays enumeration member 15630
xoLiteralParam enumeration member 15630
xoOption6 enumeration member 15630
xoOption7 enumeration member 15630
xoOption8 enumeration member 15630
xoOption9 enumeration member 15630
xoOptionA enumeration member 15630
xoOptionB enumeration member 15630
xoOptionC enumeration member 15630
xoOptionD enumeration member 15630
xoOptionE enumeration member 15630
xoOptionF enumeration member 15630
xoOptionG enumeration member 15630
xoOptionH enumeration member 15630
xoOptionI enumeration member 15630
xoQuote enumeration member 26089
xoSimpleTypeWrapper enumeration member 15630
xpEncoding enumeration member 26246
xpStandalone enumeration member 26246
xpVersion enumeration member 26246
XSBuiltIns 26511
XSBuiltIns namespace 26511
Classes 26513
Constants 26576

kkkk

Functions 26571
Types 26575
XSBuiltIns.DateTimeToXMLTime 26571
XSBuiltIns.DateTimeToXMLTime function 26571
XSBuiltIns.DateTimeToXSDateTime 26571
XSBuiltIns.DateTimeToXSDateTime function 26571
XSBuiltIns.EXSDateTimeException 26513
XSBuiltIns.EXSDecimalException 26514
XSBuiltIns.EXSHexBinaryException 26514
XSBuiltIns.GetDataFromFile 26572
XSBuiltIns.GetDataFromFile function 26572
XSBuiltIns.IncrementAmount 26576
XSBuiltIns.IncrementAmount constant 26576
XSBuiltIns.InitXSTypes 26572
XSBuiltIns.InitXSTypes function 26572
XSBuiltIns.MaxDay 26577
XSBuiltIns.MaxDay constant 26577
XSBuiltIns.MaxMonth 26577
XSBuiltIns.MaxMonth constant 26577
XSBuiltIns.MinDay 26577
XSBuiltIns.MinDay constant 26577
XSBuiltIns.MinMonth 26578
XSBuiltIns.MinMonth constant 26578
XSBuiltIns.SDecimal 26578
XSBuiltIns.SDecimal constant 26578
XSBuiltIns.SHexMarker 26578
XSBuiltIns.SHexMarker constant 26578
XSBuiltIns.SLocalTimeMarker 26579
XSBuiltIns.SLocalTimeMarker constant 26579
XSBuiltIns.SNAN 26579
XSBuiltIns.SNAN constant 26579
XSBuiltIns.SNegative 26579
XSBuiltIns.SNegative constant 26579
XSBuiltIns.SoapDecimalSeparator 26580
XSBuiltIns.SoapDecimalSeparator constant 26580
XSBuiltIns.SoapFloatToStr 26573
XSBuiltIns.SoapFloatToStr function 26573
XSBuiltIns.SoapStrToFloat 26573
XSBuiltIns.SoapStrToFloat function 26573
XSBuiltIns.SoapTimePrefix 26580
XSBuiltIns.SoapTimePrefix constant 26580
XSBuiltIns.SPlus 26580
XSBuiltIns.SPlus constant 26580
XSBuiltIns.SSciNotationMarker 26581
XSBuiltIns.SSciNotationMarker constant 26581
XSBuiltIns.TXSCustomDateTime 26515
XSBuiltIns.TXSCustomDateTime.AsDateTime 26527
XSBuiltIns.TXSCustomDateTime.AsUTCDateTime 26528
XSBuiltIns.TXSCustomDateTime.Create 26517
XSBuiltIns.TXSCustomDateTime.Day 26528
XSBuiltIns.TXSCustomDateTime.Destroy 26517
XSBuiltIns.TXSCustomDateTime.FDateTime 26518
XSBuiltIns.TXSCustomDateTime.FractionalSeconds 26529
XSBuiltIns.TXSCustomDateTime.GetAsDateTime 26519
XSBuiltIns.TXSCustomDateTime.GetAsUTCDateTime 26519
XSBuiltIns.TXSCustomDateTime.GetDay 26519
XSBuiltIns.TXSCustomDateTime.GetFractionalSeconds 26520
XSBuiltIns.TXSCustomDateTime.GetHour 26520
XSBuiltIns.TXSCustomDateTime.GetHourOffset 26520
XSBuiltIns.TXSCustomDateTime.GetMillisecond 26521
XSBuiltIns.TXSCustomDateTime.GetMinute 26521
XSBuiltIns.TXSCustomDateTime.GetMinuteOffset 26521
XSBuiltIns.TXSCustomDateTime.GetMonth 26521
XSBuiltIns.TXSCustomDateTime.GetSecond 26522
XSBuiltIns.TXSCustomDateTime.GetUseZeroMilliseconds 26522
XSBuiltIns.TXSCustomDateTime.GetYear 26522
XSBuiltIns.TXSCustomDateTime.Hour 26529
XSBuiltIns.TXSCustomDateTime.HourOffset 26530
XSBuiltIns.TXSCustomDateTime.Millisecond 26530
XSBuiltIns.TXSCustomDateTime.Minute 26531
XSBuiltIns.TXSCustomDateTime.MinuteOffset 26531
XSBuiltIns.TXSCustomDateTime.Month 26532
XSBuiltIns.TXSCustomDateTime.Second 26532
XSBuiltIns.TXSCustomDateTime.SetAsDateTime 26523
XSBuiltIns.TXSCustomDateTime.SetAsUTCDateTime 26523
XSBuiltIns.TXSCustomDateTime.SetDay 26523
XSBuiltIns.TXSCustomDateTime.SetFractionalSeconds 26524
XSBuiltIns.TXSCustomDateTime.SetHour 26524

XSBuiltIns.TXSCustomDateTime.SetHourOffset 26524
XSBuiltIns.TXSCustomDateTime.SetMillisecond 26524
XSBuiltIns.TXSCustomDateTime.SetMinute 26525
XSBuiltIns.TXSCustomDateTime.SetMinuteOffset 26525
XSBuiltIns.TXSCustomDateTime.SetMonth 26525
XSBuiltIns.TXSCustomDateTime.SetSecond 26526
XSBuiltIns.TXSCustomDateTime.SetUseZeroMilliseconds 26526
XSBuiltIns.TXSCustomDateTime.SetYear 26526
XSBuiltIns.TXSCustomDateTime.UseZeroMilliseconds 26533
XSBuiltIns.TXSCustomDateTime.Year 26533
XSBuiltIns.TXSDate 26534
XSBuiltIns.TXSDate.AsDate 26539
XSBuiltIns.TXSDate.Clone 26536
XSBuiltIns.TXSDate.Create 26535
XSBuiltIns.TXSDate.Day 26539
XSBuiltIns.TXSDate.GetAsDate 26536
XSBuiltIns.TXSDate.GetDay 26537
XSBuiltIns.TXSDate.GetMonth 26537
XSBuiltIns.TXSDate.GetYear 26537
XSBuiltIns.TXSDate.Month 26540
XSBuiltIns.TXSDate.NativeToXS 26537
XSBuiltIns.TXSDate.SetAsDate 26538
XSBuiltIns.TXSDate.SetDay 26538
XSBuiltIns.TXSDate.SetMonth 26538
XSBuiltIns.TXSDate.SetYear 26538
XSBuiltIns.TXSDate.XSToNative 26538
XSBuiltIns.TXSDate.Year 26540
XSBuiltIns.TXSDateTime 26541
XSBuiltIns.TXSDateTime.Clone 26543
XSBuiltIns.TXSDateTime.CompareDateTimeParam 26543
XSBuiltIns.TXSDateTime.Millisecond 26544
XSBuiltIns.TXSDateTime.NativeToXS 26544
XSBuiltIns.TXSDateTime.XSToNative 26544
XSBuiltIns.TXSDecimal 26545
XSBuiltIns.TXSDecimal.AsBcd 26547
XSBuiltIns.TXSDecimal.DecimalString 26548
XSBuiltIns.TXSDecimal.NativeToXS 26547
XSBuiltIns.TXSDecimal.XSToNative 26547
XSBuiltIns.TXSDuration 26548
XSBuiltIns.TXSDuration.Create 26551
XSBuiltIns.TXSDuration.DecimalSecond 26553
XSBuiltIns.TXSDuration.GetDecimalSecond 26551
XSBuiltIns.TXSDuration.NativeToXS 26552
XSBuiltIns.TXSDuration.SetDecimalSecond 26552
XSBuiltIns.TXSDuration.XSToNative 26552
XSBuiltIns.TXSHexBinary 26553
XSBuiltIns.TXSHexBinary.AsByteArray 26555
XSBuiltIns.TXSHexBinary.HexBinaryString 26556
XSBuiltIns.TXSHexBinary.NativeToXS 26555
XSBuiltIns.TXSHexBinary.XSToNative 26555
XSBuiltIns.TXSString 26556
XSBuiltIns.TXSString.NativeToXS 26557
XSBuiltIns.TXSString.XSToNative 26558
XSBuiltIns.TXSTime 26558
XSBuiltIns.TXSTime.AsTime 26566
XSBuiltIns.TXSTime.Clone 26561
XSBuiltIns.TXSTime.FractionalSeconds 26566
XSBuiltIns.TXSTime.FractionalSecondString 26567
XSBuiltIns.TXSTime.GetAsTime 26561
XSBuiltIns.TXSTime.GetFractionalSeconds 26561
XSBuiltIns.TXSTime.GetFractionalSecondString 26561
XSBuiltIns.TXSTime.GetHour 26562
XSBuiltIns.TXSTime.GetHourOffset 26562
XSBuiltIns.TXSTime.GetMillisecond 26562
XSBuiltIns.TXSTime.GetMinute 26562
XSBuiltIns.TXSTime.GetMinuteOffset 26563
XSBuiltIns.TXSTime.GetSecond 26563
XSBuiltIns.TXSTime.Hour 26567
XSBuiltIns.TXSTime.HourOffset 26567
XSBuiltIns.TXSTime.Millisecond 26568
XSBuiltIns.TXSTime.Minute 26568
XSBuiltIns.TXSTime.MinuteOffset 26569
XSBuiltIns.TXSTime.NativeToXS 26563
XSBuiltIns.TXSTime.Second 26569
XSBuiltIns.TXSTime.SetAsTime 26563
XSBuiltIns.TXSTime.SetFractionalSeconds 26564
XSBuiltIns.TXSTime.SetHour 26564
XSBuiltIns.TXSTime.SetHourOffset 26564

XSBuiltIns.TXSTime.SetMillisecond 26564
XSBuiltIns.TXSTime.SetMinute 26564
XSBuiltIns.TXSTime.SetMinuteOffset 26565
XSBuiltIns.TXSTime.SetSecond 26565
XSBuiltIns.TXSTime.UseZeroMilliseconds 26569
XSBuiltIns.TXSTime.XSToNative 26565
XSBuiltIns.TXSTimeInstant 26575
XSBuiltIns.TXSTimeInstant type 26575
XSBuiltIns.XMLDateSeparator 26581
XSBuiltIns.XMLDateSeparator constant 26581
XSBuiltIns.XMLDayPos 26581
XSBuiltIns.XMLDayPos constant 26581
XSBuiltIns.XMLDefaultYearDigits 26582
XSBuiltIns.XMLDefaultYearDigits constant 26582
XSBuiltIns.XMLDurationDay 26582
XSBuiltIns.XMLDurationDay constant 26582
XSBuiltIns.XMLDurationHour 26582
XSBuiltIns.XMLDurationHour constant 26582
XSBuiltIns.XMLDurationMinute 26583
XSBuiltIns.XMLDurationMinute constant 26583
XSBuiltIns.XMLDurationMonth 26583
XSBuiltIns.XMLDurationMonth constant 26583
XSBuiltIns.XMLDurationSecond 26583
XSBuiltIns.XMLDurationSecond constant 26583
XSBuiltIns.XMLDurationStart 26584
XSBuiltIns.XMLDurationStart constant 26584
XSBuiltIns.XMLDurationYear 26584
XSBuiltIns.XMLDurationYear constant 26584
XSBuiltIns.XMLHourOffsetMinusMarker 26584
XSBuiltIns.XMLHourOffsetMinusMarker constant 26584
XSBuiltIns.XMLHourOffsetPlusMarker 26585
XSBuiltIns.XMLHourOffsetPlusMarker constant 26585
XSBuiltIns.XMLMilSecPos 26585
XSBuiltIns.XMLMilSecPos constant 26585
XSBuiltIns.XMLMonthPos 26585
XSBuiltIns.XMLMonthPos constant 26585
XSBuiltIns.XMLTimeSeparator 26586
XSBuiltIns.XMLTimeSeparator constant 26586
XSBuiltIns.XMLTimeToDateTime 26573
XSBuiltIns.XMLTimeToDateTime function 26573
XSBuiltIns.XMLYearPos 26586
XSBuiltIns.XMLYearPos constant 26586
XSLProd 26587
XSLProd namespace 26587
Classes 26588
XSLProd.EXSLPageProducerError 26588
XSLProd.TCustomXSLPageProducer 26588
XSLProd.TCustomXSLPageProducer.Content 26597
XSLProd.TCustomXSLPageProducer.ContentFromStream 26598
XSLProd.TCustomXSLPageProducer.ContentFromString 26598
XSLProd.TCustomXSLPageProducer.ContentFromWideString 26599
XSLProd.TCustomXSLPageProducer.Dispatcher 26603
XSLProd.TCustomXSLPageProducer.DispatcherComponent 26604
XSLProd.TCustomXSLPageProducer.GetAppDispatcher 26599
XSLProd.TCustomXSLPageProducer.GetProducerTemplateFileName 26599
XSLProd.TCustomXSLPageProducer.GetProducerTemplateStream 26599
XSLProd.TCustomXSLPageProducer.GetTemplateFileType 26600
XSLProd.TCustomXSLPageProducer.GetTemplateStream 26600
XSLProd.TCustomXSLPageProducer.GetXMLData 26600
XSLProd.TCustomXSLPageProducer.GetXMLDocument 26600
XSLProd.TCustomXSLPageProducer.GetXSLData 26601
XSLProd.TCustomXSLPageProducer.HasHTMLBrowserView 26601
XSLProd.TCustomXSLPageProducer.HasScriptView 26601
XSLProd.TCustomXSLPageProducer.HasXMLBrowserView 26601
XSLProd.TCustomXSLPageProducer.HasXSLBrowserView 26601
XSLProd.TCustomXSLPageProducer.Notification 26602
XSLProd.TCustomXSLPageProducer.ProduceContent 26602
XSLProd.TCustomXSLPageProducer.ProduceContentFromStream 26602
XSLProd.TCustomXSLPageProducer.ProduceContentFromString

nnnn

26603
XSLProd.TCustomXSLPageProducer.SetAppDispatcher 26603
XSLProd.TCustomXSLPageProducer.XMLData 26605
XSLProd.TXSLPageProducer 26605
XSLProd.TXSLPageProducer.XMLData 26613
xtCheck enumeration member 17510
xtColumns enumeration member 17741
xtHorizontal enumeration member 17741
xtLeftTop enumeration member 17741
xtRadio enumeration member 17510
xtRadioEx enumeration member 17510
xtRows enumeration member 17741
xtSummaries enumeration member 17741
xtVertical enumeration member 17741

Z

ZLib 26615
ZLib namespace 26615
- Classes 26617
- Constants 26640
- Functions 26632
- Structs, Records, Enums 26637
- Types 26639

ZLib.CompressBuf 26632
ZLib.CompressBuf function 26632
ZLib.DecompressBuf 26632
ZLib.DecompressBuf function 26632
ZLib.DecompressToUserBuf 26633
ZLib.DecompressToUserBuf function 26633
ZLib.deflate 26633
ZLib.deflate function 26633
ZLib.deflateEnd 26633
ZLib.deflateEnd function 26633
ZLib.deflateInit_ 26634
ZLib.deflateInit_ function 26634
ZLib.ECompressionError 26617
ZLib.EDecompressionError 26617
ZLib.EZlibError 26618
ZLib.inflate 26634
ZLib.inflate function 26634
ZLib.inflateEnd 26634
ZLib.inflateEnd function 26634
ZLib.inflateInit_ 26635
ZLib.inflateInit_ function 26635
ZLib.inflateReset 26635
ZLib.inflateReset function 26635
ZLib.TAlloc 26639
ZLib.TAlloc type 26639
ZLib.TCompressionLevel 26637
ZLib.TCompressionLevel enumeration 26637
ZLib.TCompressionStream 26618
ZLib.TCompressionStream.CompressionRate 26622
ZLib.TCompressionStream.Create 26620
ZLib.TCompressionStream.Destroy 26620
ZLib.TCompressionStream.OnProgress 26623
ZLib.TCompressionStream.Read 26621
ZLib.TCompressionStream.Seek 26621
ZLib.TCompressionStream.Write 26622
ZLib.TCustomZlibStream 26623
ZLib.TCustomZlibStream.Create 26625
ZLib.TCustomZlibStream.OnProgress 26626
ZLib.TCustomZlibStream.Progress 26626
ZLib.TDecompressionStream 26627
ZLib.TDecompressionStream.Create 26628
ZLib.TDecompressionStream.Destroy 26629
ZLib.TDecompressionStream.OnProgress 26631
ZLib.TDecompressionStream.Read 26629
ZLib.TDecompressionStream.Seek 26630
ZLib.TDecompressionStream.Write 26630
ZLib.TFree 26639
ZLib.TFree type 26639
ZLib.TZStreamRec 26638
ZLib.TZStreamRec record 26638
ZLib.Z_ASCII 26640
ZLib.Z_ASCII constant 26640
ZLib.Z_BEST_COMPRESSION 26641
ZLib.Z_BEST_COMPRESSION constant 26641
ZLib.Z_BEST_SPEED 26641

ZLib.Z_BEST_SPEED constant 26641
ZLib.Z_BINARY 26641
ZLib.Z_BINARY constant 26641
ZLib.Z_BUF_ERROR 26642
ZLib.Z_BUF_ERROR constant 26642
ZLib.Z_DATA_ERROR 26642
ZLib.Z_DATA_ERROR constant 26642
ZLib.Z_DEFAULT_COMPRESSION 26642
ZLib.Z_DEFAULT_COMPRESSION constant 26642
ZLib.Z_DEFAULT_STRATEGY 26643
ZLib.Z_DEFAULT_STRATEGY constant 26643
ZLib.Z_DEFLATED 26643
ZLib.Z_DEFLATED constant 26643
ZLib.Z_ERRNO 26643
ZLib.Z_ERRNO constant 26643
ZLib.Z_FILTERED 26644
ZLib.Z_FILTERED constant 26644
ZLib.Z_FINISH 26644
ZLib.Z_FINISH constant 26644
ZLib.Z_FULL_FLUSH 26644
ZLib.Z_FULL_FLUSH constant 26644
ZLib.Z_HUFFMAN_ONLY 26645
ZLib.Z_HUFFMAN_ONLY constant 26645
ZLib.Z_MEM_ERROR 26645
ZLib.Z_MEM_ERROR constant 26645
ZLib.Z_NEED_DICT 26645
ZLib.Z_NEED_DICT constant 26645
ZLib.Z_NO_COMPRESSION 26646
ZLib.Z_NO_COMPRESSION constant 26646
ZLib.Z_NO_FLUSH 26646
ZLib.Z_NO_FLUSH constant 26646
ZLib.Z_OK 26646
ZLib.Z_OK constant 26646
ZLib.Z_PARTIAL_FLUSH 26647
ZLib.Z_PARTIAL_FLUSH constant 26647
ZLib.Z_STREAM_END 26647
ZLib.Z_STREAM_END constant 26647
ZLib.Z_STREAM_ERROR 26647
ZLib.Z_STREAM_ERROR constant 26647
ZLib.Z_SYNC_FLUSH 26648
ZLib.Z_SYNC_FLUSH constant 26648
ZLib.Z_UNKNOWN 26648
ZLib.Z_UNKNOWN constant 26648
ZLib.Z_VERSION_ERROR 26648
ZLib.Z_VERSION_ERROR constant 26648
ZLib.zlib_Version 26649
ZLib.zlib_Version constant 26649
ZLib.zlibAllocMem 26635
ZLib.zlibAllocMem function 26635
ZLib.zlibFreeMem 26636
ZLib.zlibFreeMem function 26636